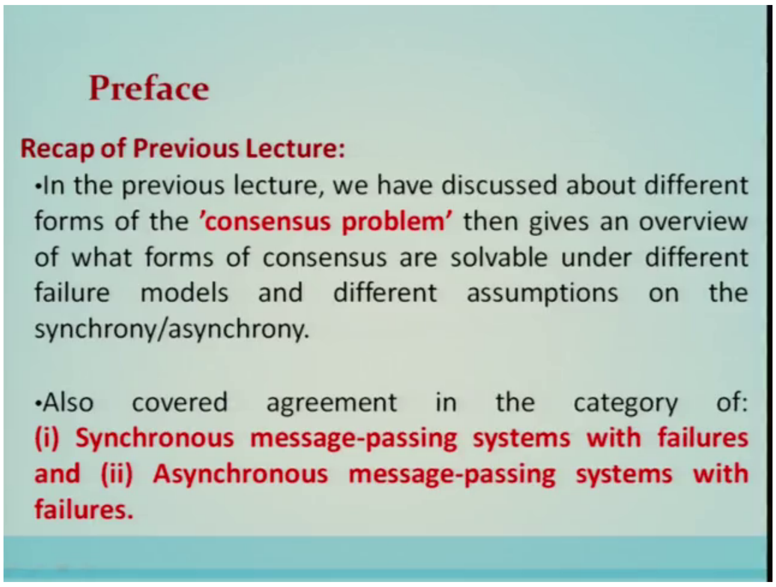
Distributed Systems
Dr. Rajiv Misra
Department of Computer Science and Engineering
Indian Institute of Technology, Patna

Lecture – 11
Checkpointing and Roll back Recovery

Lecture 11 checkpointing and roll back recovery preface recap of previous lecture.

(Refer Slide Time: 00:28)



Preface

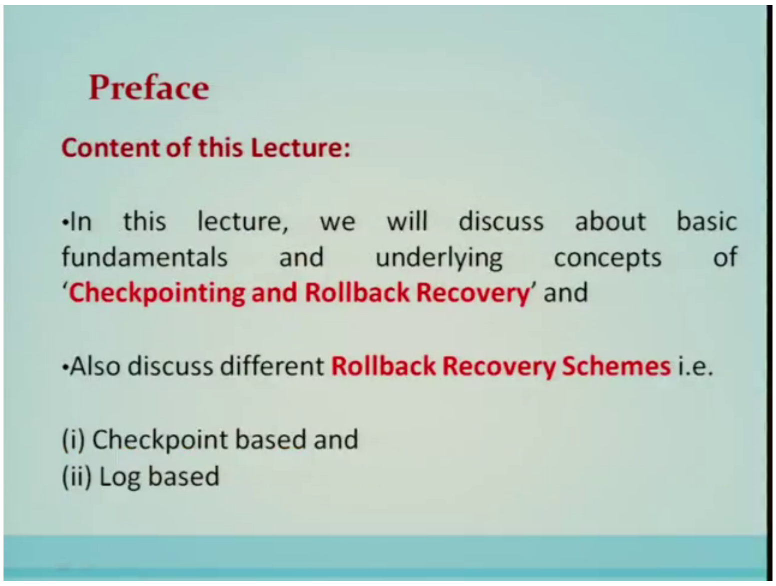
Recap of Previous Lecture:

•In the previous lecture, we have discussed about different forms of the **'consensus problem'** then gives an overview of what forms of consensus are solvable under different failure models and different assumptions on the synchrony/asynchrony.

•Also covered agreement in the category of: **(i) Synchronous message-passing systems with failures and (ii) Asynchronous message-passing systems with failures.**

In previous lecture, we have discussed about different forms of consensus problem, then it gives an overview of what forms the consensus are solvable under the different failure models and different assumptions on the synchrony and asynchrony we have also covered the agreement in the category of synchronous message passing system with failures asynchronous message passing system with failures content of this lecture.

(Refer Slide Time: 00:57)



Preface

Content of this Lecture:

•In this lecture, we will discuss about basic fundamentals and underlying concepts of '**Checkpointing and Rollback Recovery**' and

•Also discuss different **Rollback Recovery Schemes** i.e.

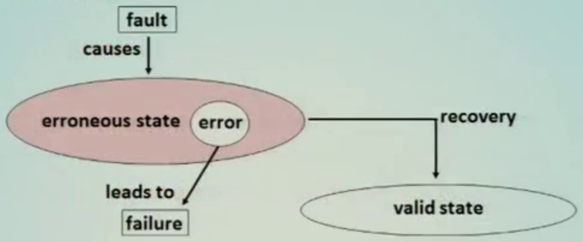
(i) Checkpoint based and
(ii) Log based

In this lecture, we will discuss about basic fundamentals and underlying principles of checkpointing and roll back recovery and also discuss different roll back recovery schemes that are checkpointing based roll back recovery and log based roll back recovery schemes.

(Refer Slide Time: 01:14)

Introduction

• **Fault Tolerance**



•An error is a manifestation of a fault that can lead to a failure.

•**Failure Recovery:**

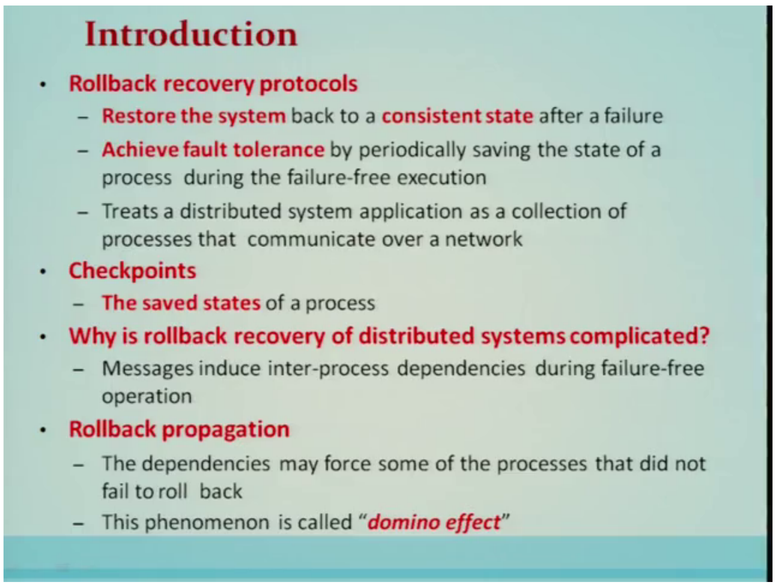
- Backward recovery- Restore system state to a previous error-free state
  - operation-based (do-undo-redo logs)
  - state-based (checkpointing/logging)
- Forward recovery- Repair the erroneous part of the system state

Introduction; the fault tolerance and error is a manifestation of the fault that can lead to a failure in any system that is shown over here.

So, a failure recovery is about either the backward recovery or a forward recovery in forward recovery the repair the erroneous part of the system state and which is very difficult to predict and maintain at any point of time. So, another way of failure recovery is the backward recovery in backward recovery, you have to restore the system state to a previous error free state and it is done using operation based method and another is called state based method and state based method is also called checkpointing of league logging methods.

So, here in our discussion we are basically looking up the backward recovery procedures for failure recovery roll back recovery algorithms, they restore the system back to a consistent state after a failure.

(Refer Slide Time: 02:21)



Introduction

- **Rollback recovery protocols**
  - **Restore the system** back to a **consistent state** after a failure
  - **Achieve fault tolerance** by periodically saving the state of a process during the failure-free execution
  - Treats a distributed system application as a collection of processes that communicate over a network
- **Checkpoints**
  - **The saved states** of a process
- **Why is rollback recovery of distributed systems complicated?**
  - Messages induce inter-process dependencies during failure-free operation
- **Rollback propagation**
  - The dependencies may force some of the processes that did not fail to roll back
  - This phenomenon is called "***domino effect***"

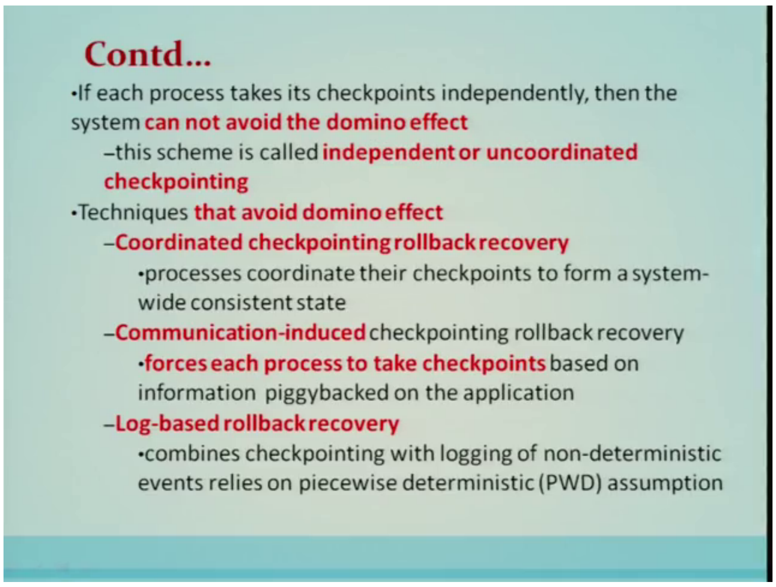
Now, they will achieve the fault tolerant by periodically saving the state of a process during the failure free execution they also treat the distributed application as a collection of a process that communicate over the network.

So, in the roll back recovery schemes we will see how we have to restore the system using the states which we save at a particular instance of time either through the checkpointing or through the logging method or including both of them. So, checkpoints checkpoints is a save state of a process. Now, why is the roll back recovery of a distributed system is complicated, the messages here in distributed system induce inter process dependencies during the failure free operation.

So, this particular dependencies need to be maintained at the time of recovery and that is why it is not so easy and it becomes a complicated now another aspect here in roll back recovery is called roll back propagation. So, the dependencies may force some of the processes that did not fail to roll back and this is called roll back propagation. So, it is not only the roll back of a affected process, but all the processes which are dependent to that affected process has to roll back and this will cascade this role backing of different process and that is called roll back propagation

So, this particular fan one of roll back propagation is called domino effect each process takes its checkpoint independently then the system cannot avoid the domino effect.

(Refer Slide Time: 04:20)



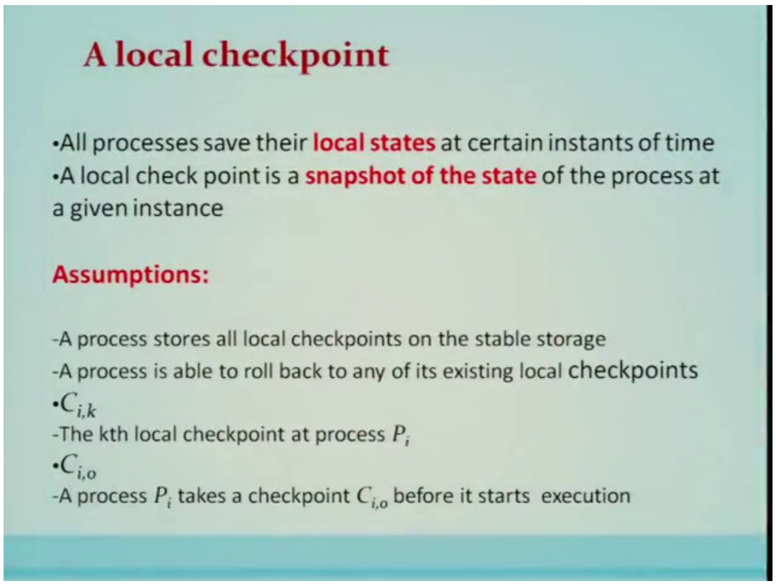
Contd...

- If each process takes its checkpoints independently, then the system **can not avoid the domino effect**
  - this scheme is called **independent or uncoordinated checkpointing**
- Techniques **that avoid domino effect**
  - **Coordinated checkpointing rollback recovery**
    - processes coordinate their checkpoints to form a system-wide consistent state
  - **Communication-induced** checkpointing rollback recovery
    - **forces each process to take checkpoints** based on information piggybacked on the application
  - **Log-based rollback recovery**
    - combines checkpointing with logging of non-deterministic events relies on piecewise deterministic (PWD) assumption

And this scheme is called independent or uncoordinated checkpointing. So, the technique that avoids the domino effect are coordinated checkpointing roll back recovery here the processes coordinate with them to take their checkpoints and such that the checkpoint form a system wide consistent state globally.

Now, another type of or another technique is called communication induced checkpointing roll back recovery, here it forces each process to take checkpoints based on the information piggybacked on the application that is called the communication induced. So, it has 2 different type of this one checkpointing one is in the normal checkpointing the other one is the post checkpointing.

(Refer Slide Time: 05:35)



A local checkpoint

•All processes save their **local states** at certain instants of time
•A local check point is a **snapshot of the state** of the process at a given instance

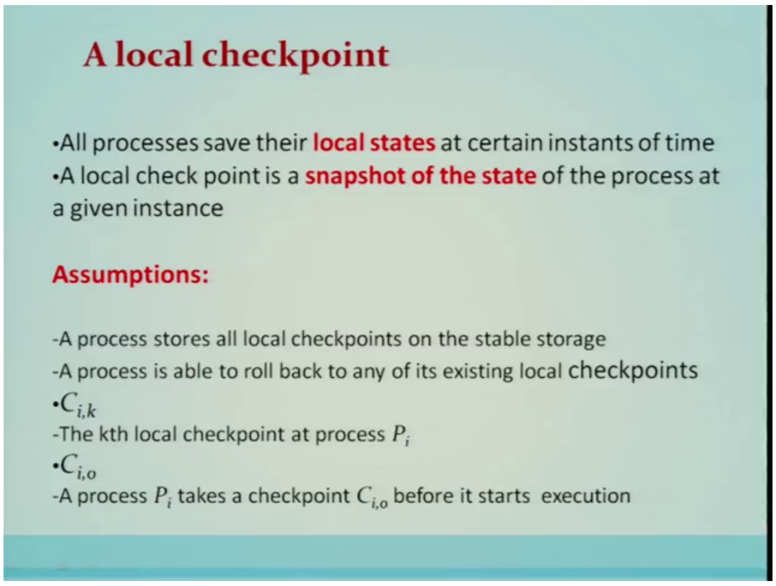
Assumptions:

-A process stores all local checkpoints on the stable storage
-A process is able to roll back to any of its existing local checkpoints
•$C_{i,k}$
-The kth local checkpoint at process P_i
•$C_{i,o}$
-A process P_i takes a checkpoint $C_{i,o}$ before it starts execution

Now, another type of checkpointing is called roll back recovery log based roll back recovery it combines the checkpointing with logging of non deterministic events relies on the piecewise did not mistake assumptions. Now preliminaries these preliminaries are useful to see when we will discuss the details of checkpointing and roll back recover. So, in this preliminary the first definition is about the local checkpoint.

So, all the processes save their local states at a certain instant of time. So, a local checkpoint is nothing, but a snapshot of a state of a process at a given instance assumptions a process stores all local checkpoints on the stable storage.

(Refer Slide Time: 06:05)



A local checkpoint

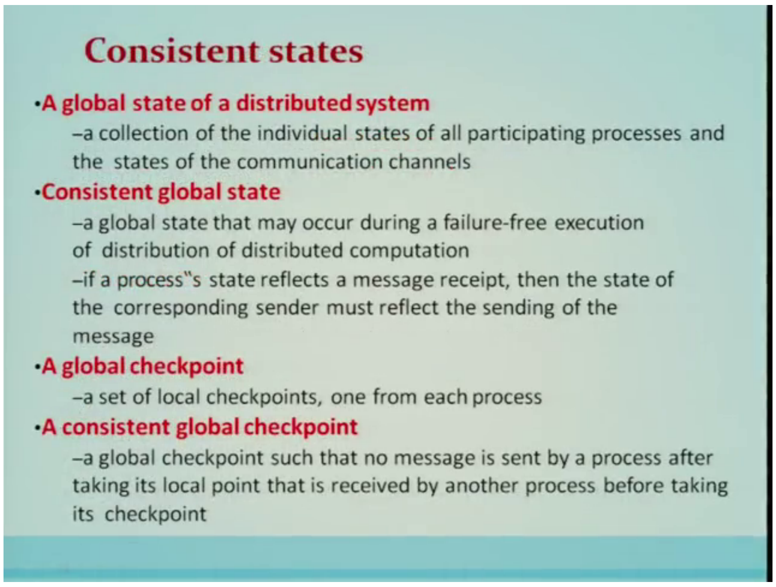
•All processes save their **local states** at certain instants of time
•A local check point is a **snapshot of the state** of the process at a given instance

Assumptions:

-A process stores all local checkpoints on the stable storage
-A process is able to roll back to any of its existing local checkpoints
•$C_{i,k}$
-The kth local checkpoint at process P_i
•$C_{i,o}$
-A process P_i takes a checkpoint $C_{i,o}$ before it starts execution

So, a process is able to roll back to any of its existing local checkpoints. So, the terminologies which are used here to represent the local checkpoint is represented by Cik here i is the ith process and the kth local checkpoint kth local checkpoint which is taken at process i and that is represented by Cik Ci 0 is basically the process i takes the checkpoint Ci 0 before it start the execution.

(Refer Slide Time: 06:41)



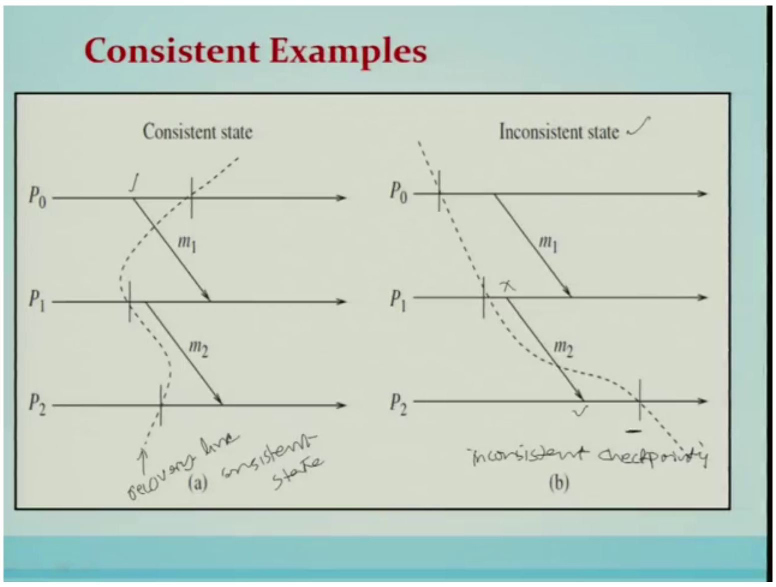
Consistent states

•**A global state of a distributed system**
–a collection of the individual states of all participating processes and the states of the communication channels
•**Consistent global state**
–a global state that may occur during a failure-free execution of distribution of distributed computation
–if a process"s state reflects a message receipt, then the state of the corresponding sender must reflect the sending of the message
•**A global checkpoint**
–a set of local checkpoints, one from each process
•**A consistent global checkpoint**
–a global checkpoint such that no message is sent by a process after taking its local point that is received by another process before taking its checkpoint

Now, consistent states a global state of a distributed system is nothing, but a collection of individual states of all participating processes and the states of a communication channel now this global state is a consistent global state a global state that may occur during the failure free of execution of a distributed computing and if a processes states reflects a message receives, then the state of the corresponding sender must reflect the sending of the message; that means, in a consistent global state a; if a message is received then its send is also received if that is maintained then it is called consistent global state.

So, the global checkpoint is a set of local checkpoints one from each process and a consistent global checkpoint is having the similar definition as the consistent global state; that means, a global checkpoint such that no message is sent by your process after taking its local checkpoint that is receive by another process before taking its checkpoint.

(Refer Slide Time: 07:52)

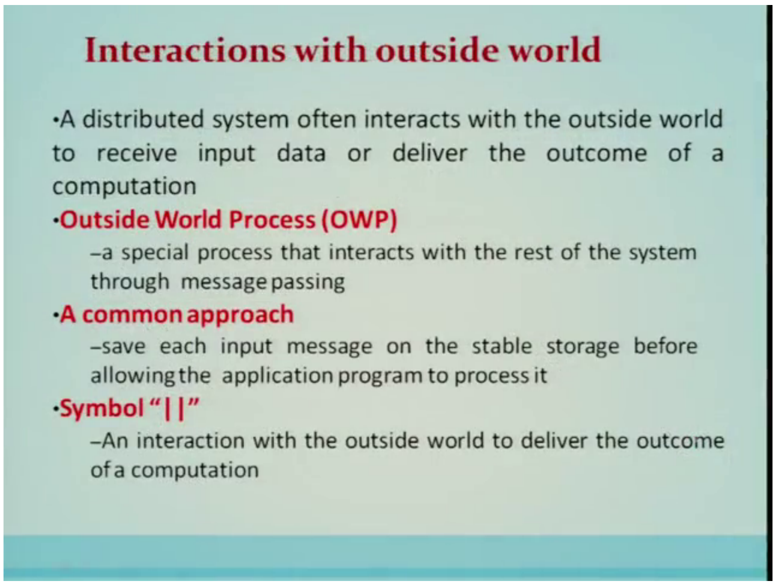


Consistent Examples

These are basically can be expressed and can be understood using these examples.

So, here we can see that this particular recovery line which is shown as the dotted line shown as dotted line is forming a consistent checkpoints and is also a consistent state. So, in this consistent in state you see that this message is sent; send is recorded, but received is not recorded and there is no message whose received is recorded, but send is not recorded. Hence, it is a consistent state in consistent states here you can see that this the receipt of a message is recorded in this particular checkpoint, but its send is not recorded in any of the collection of the checkpoints hence this collection of checkpoints is basically an inconsistent state.

(Refer Slide Time: 09:12)

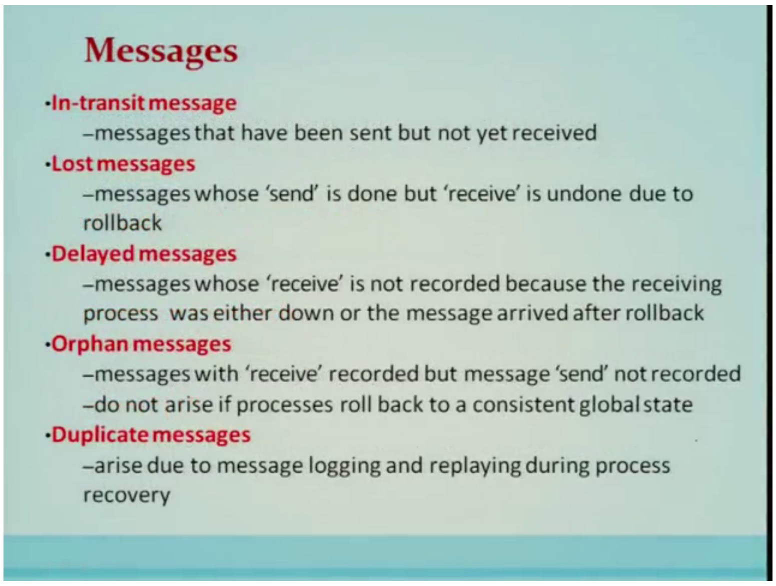


Interactions with outside world

- A distributed system often interacts with the outside world to receive input data or deliver the outcome of a computation
- **Outside World Process (OWP)**
  - a special process that interacts with the rest of the system through message passing
- **A common approach**
  - save each input message on the stable storage before allowing the application program to process it
- **Symbol "||"**
  - An interaction with the outside world to deliver the outcome of a computation

Now, interaction with the outside world a distributed system of an interacts with the outside world to receive the input data or deliver the outcome of the computation outside work process OWP; a special process that interacts with the rest of the system through the message passing a common approach save each input message on the stable storage before allowing the application to process it and parallel bars is a system that an interaction with the outside world to deliver the outcome of these particular messages.

(Refer Slide Time: 09:45)

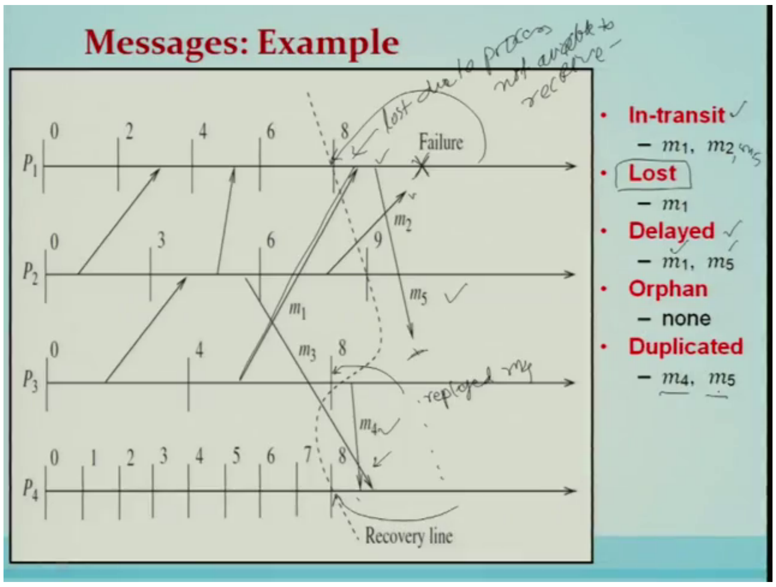


Messages

- **In-transit message**
  - messages that have been sent but not yet received
- **Lost messages**
  - messages whose 'send' is done but 'receive' is undone due to rollback
- **Delayed messages**
  - messages whose 'receive' is not recorded because the receiving process was either down or the message arrived after rollback
- **Orphan messages**
  - messages with 'receive' recorded but message 'send' not recorded
  - do not arise if processes roll back to a consistent global state
- **Duplicate messages**
  - arise due to message logging and replaying during process recovery

So, there are different types of messages which we have to see under the preliminaries. So, in transit message' message that have been sent, but not yet delivered we will explain in the next figure. So, the last message is the message whose send is done, but receive is undone due to the roll back and that is called the lost message delayed messages; messages whose receive is not recorded, but the receiving process was either down or the message arrived after the roll back that we will see orphan messages the messages with receive recorded, but message send not recorded and these orphan message do not arise if the process roll back to the consistent global state.

(Refer Slide Time: 10:47)



Messages: Example

- In-transit ✓
  - m_1, m_2, m_5
- Lost
  - m_1
- Delayed ✓
  - m_1, m_5
- Orphan
  - none
- Duplicated
  - m_4, m_5

Duplicate messages arises due to the message logging and replaying during the process recovery let us understand it in this particular diagram here the first case is about the messages which are in transit. So, m 1 and m 2. So, m 1 is this particular message and m 2 is this particular message this is in transit not yet delivered and this is m 2 also here m 5 is also under transit.

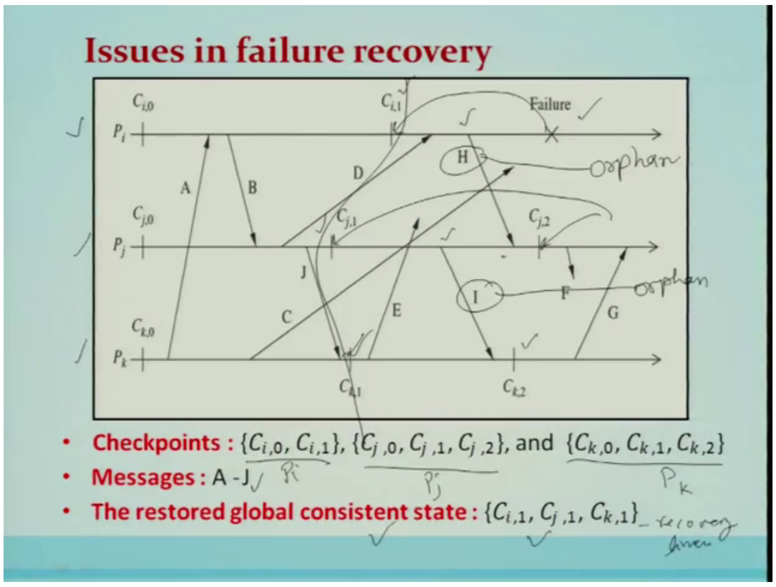
Lost message is m 1; why m 1 is lost message, this message is lost why because there is a failure and due to the failure if it roll backs to the previously consistent state or a continuously consistent checkpoint state, then this particular receipt of this particular message will not be there message will or the process will not be there to receive it. Hence this is called a lost message due to the process not available to receive it due to the failure or maybe some other region.

Now, another type of message is called a delayed message. So, let us say that if the message is in transit and this P 1 recovers from this particular checkpoint number eight onwards and then basically it will receive the same message then that particular message is called basically the delayed message, if the message is basically arrived late, then it is called the delayed message.

So, basically these are the example of a delayed message orphan different assumptions on m 4 and m 5, this is m 4 and m 5, they are duplicated messages why because you see the dotted line is a recovery line if the system roll backs. So, you can see that let us take the example of m 4 first. So, m 4 is send and its received is also basically recorded.

Now, when it has to roll back then from the log this particular message sending of m 4 message will be replayed again resulting in another copy of the message m 4 will be generated and here on message are on process P 4 that same copy will basically be resend again and will result into a duplicate messages here. Similarly, here it is a failure m 5 is in transit, it will be received after some time, then again this particular message m 5 send of a message is replayed again. So, m 5 will be duplicated. So, these are basically the examples of different kind of messages here.

(Refer Slide Time: 14:04)



Issues in failure recovery

- Checkpoints : $\{C_{i,0}, C_{i,1}\}$, $\{C_{j,0}, C_{j,1}, C_{j,2}\}$, and $\{C_{k,0}, C_{k,1}, C_{k,2}\}$
- Messages : A - J
- The restored global consistent state : $\{C_{i,1}, C_{j,1}, C_{k,1}\}$

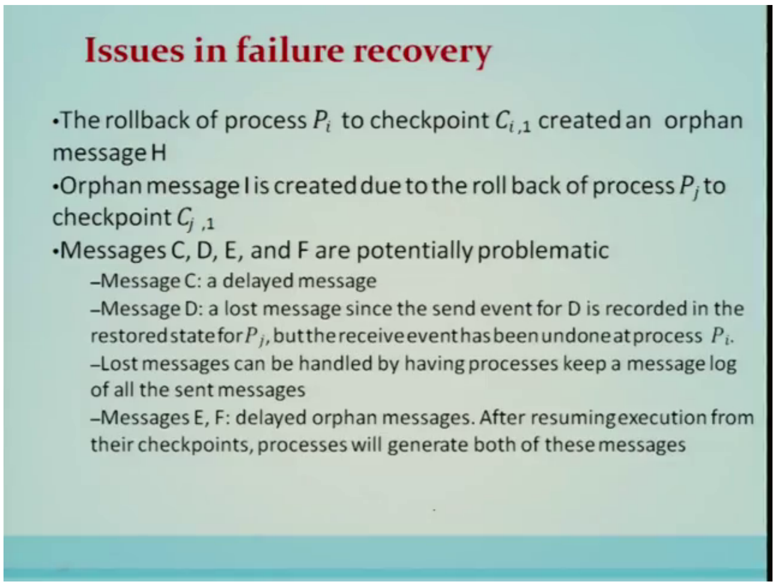
So, there are different issues in failure recovery. So, the issues are like that that these there are 3 different processes Pi, Pj and Pk; they will take their local checkpoints for example,. So, P 1 Pi will take these local checkpoints similarly the checkpoints which is taken by Pj is basically represented here similarly the process Pk will take its own local checkpoints and there is a message from a to j which are which are flowing.

Now, here when a system is failed now it has to be restored in a consistent state. So, consistent state would be to basically draw a recovery line or to find out a consistent global checkpoint state. So, here after the failure it has to roll back from this particular checkpoint. So, if this particular roll back is taking place then this send of this particular H message will be undone in that case and this if it is undone then this particular message H will result in to an message which is called a orphan message.

So, because of the orphan message the process Pj also has to roll back and it will not be rolling back at Cj 2, but it has to roll back at Cj 1. So, the recovery line is basically now following for pin Pj similarly as far as Pk is concerned Pk will not basically be able to roll back at Ck 2 why because of the same domino effect why because the this message H will become an orphan message and based on this particular message. So, it has to roll back at this point. So, this will form a recovery line and this is shown over here in this particular example. So, that consistent are restored global consistent state and this is also called a recovery line.

So, the issue here is to find out among the set of checkpoints that particular consistent global checkpoint which basically will restore the system with a minimum loss is basically a quite challenging process and that we are going to see through an algorithm how we are going to achieve it.

(Refer Slide Time: 16:47)



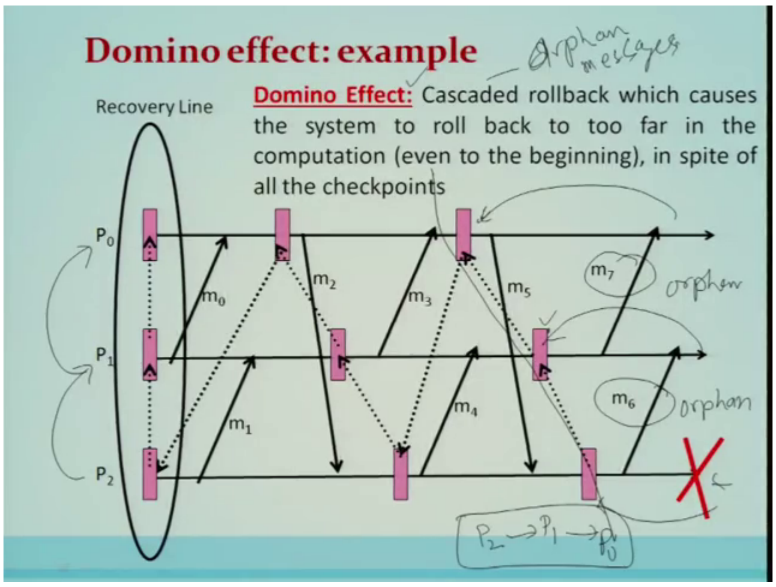
Issues in failure recovery

- The rollback of process P_i to checkpoint $C_{i,1}$ created an orphan message H
- Orphan message I is created due to the roll back of process P_j to checkpoint $C_{j,1}$
- Messages C, D, E, and F are potentially problematic
  - Message C: a delayed message
  - Message D: a lost message since the send event for D is recorded in the restored state for P_j, but the receive event has been undone at process P_i.
  - Lost messages can be handled by having processes keep a message log of all the sent messages
  - Messages E, F: delayed orphan messages. After resuming execution from their checkpoints, processes will generate both of these messages

So, the roll back of a process i to the checkpoint Cij created an orphan that i have explained orphan message i is created due to the roll back of Pj that also have explained messages C, D, E, F are potentially problematic why because message C is a delayed message; message D is a lost message.

So, the lost message can be handled by having the processes to keep message log of all the send messages message e and f they are called delayed orphan messages they are more difficult to handle. So, after resuming execution from their checkpoints the processes will generate both these particular messages. So, these are basically going to be different issues in the failure recovery and it requires coordinated or uncoordinated checkpointing its also requires a logging to replay the messages and also basically how to recover all these issues are quite intricate quite complex and need to be understood here in this particular concept.

(Refer Slide Time: 17:53)



Domino effect: example

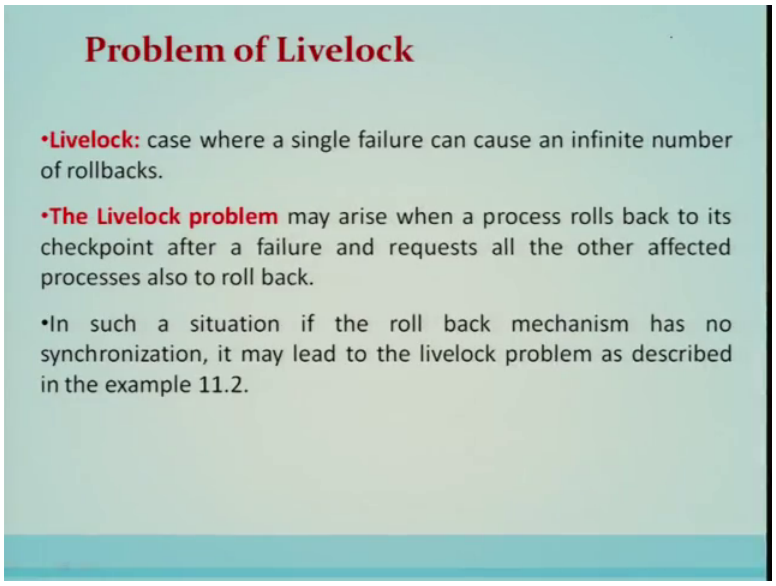
Domino Effect: Cascaded rollback which causes the system to roll back to too far in the computation (even to the beginning), in spite of all the checkpoints

Now, we are going to explain the domino effect domino effect is nothing, but it is a roll back propagation or a cascaded roll back. So, here domino effect cascaded roll back which causes the system to roll back to too far in the computation even to the beginning in spite of all the checkpoints here if you see that if the process, P 2 is failed at this point, then basically it has to roll back to the previous checkpoint and this particular previous checkpoint will basically trigger on why because this particular message m 6 will become orphan message once, it is basically rolled back will trigger the process P 1 also to roll back.

So, the roll back of P 2 will basically ensure that P 1 also has to roll back now since P 1 has roll back. So, this particular message will become orphan and hence the P 1 also has to roll back. So, this will become a recovery line P 0 also has to roll back. So, this particular rolling back of P 2 which triggers the roll back of P 1 as well as the roll back of P 0 from these on these checkpoints is basically called a cascaded roll back and this is called domino effect why because domino effect is now you can see that it is due to the orphan messages.

So, orphan messages usually basically trigger the domino effect and basically it is nothing, but a cascaded roll back we have seen through this particular example. Now another problem is called the problem of a Livelock.

(Refer Slide Time: 19:54)

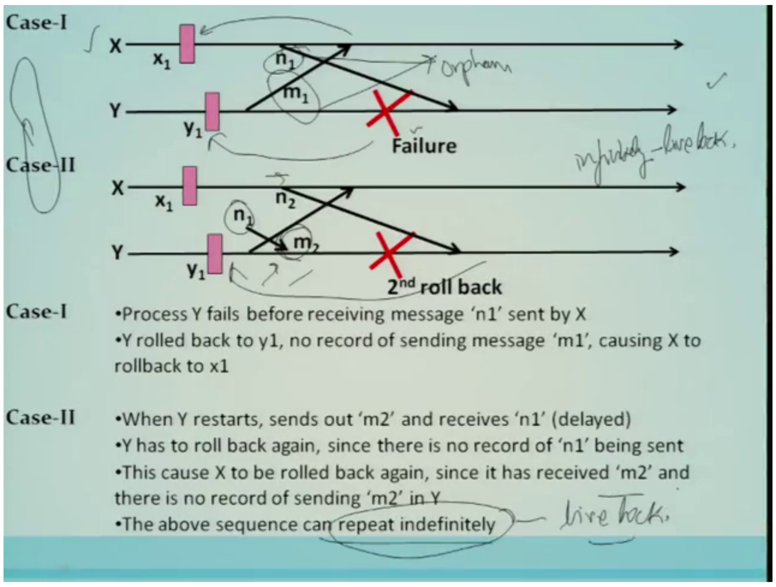


Problem of Livelock

- **Livelock:** case where a single failure can cause an infinite number of rollbacks.
- **The Livelock problem** may arise when a process rolls back to its checkpoint after a failure and requests all the other affected processes also to roll back.
- In such a situation if the roll back mechanism has no synchronization, it may lead to the livelock problem as described in the example 11.2.

So, Livelock case here where a single failure can cause infinite number of roll backs; so, Livelock problem may arise when the process roll back to its checkpoint after the failure and request all other affected process also to roll back.

(Refer Slide Time: 20:16)



Case-I
- Process Y fails before receiving message 'n1' sent by X
- Y rolled back to y1, no record of sending message 'm1', causing X to rollback to x1

Case-II
- When Y restarts, sends out 'm2' and receives 'n1' (delayed)
- Y has to roll back again, since there is no record of 'n1' being sent
- This cause X to be rolled back again, since it has received 'm2' and there is no record of sending 'm2' in Y
- The above sequence can repeat indefinitely — live lock.

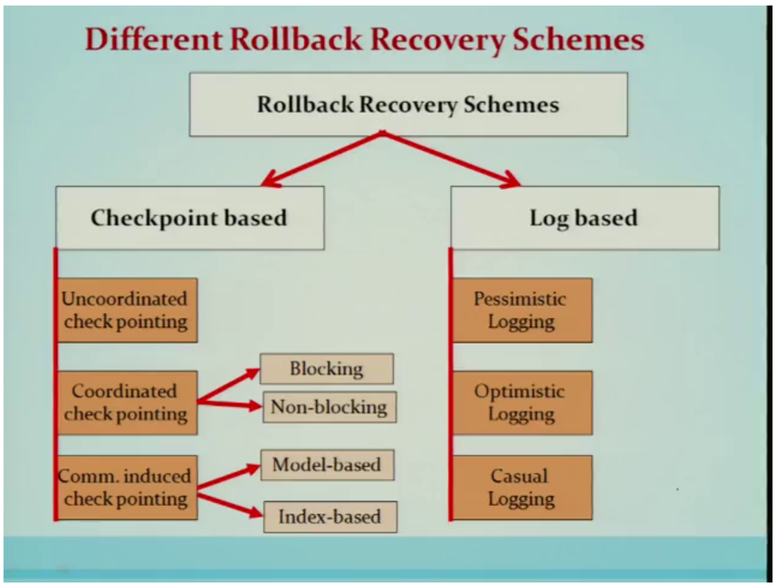
So, in such a situation if the roll back mechanism has no synchronization it will lead to a Livelock problem as described in this particular example. So, the difference between domino effect and a Livelock here is that the dominant effect is a cascaded roll back as far as Livelock is concerned it is an infinite roll backs. So, let us take an example of the

Livelock problem. Now here in the case one if you see that process is failed at this point here if it is failed, then it has to roll back to its previous checkpoint which is shown as y one if its rolling back to the previous checkpoint, then it will basically result in result into an orphan message that is called m 1 and this orphan message in turn will trigger process x also to roll back.

So, process x will roll back to its previous checkpoints now once process x roll backs. So, this particular n one will become orphan then in that case and the previously send message if it is arrived here as a duplicate message because if it is restart from here. So, this particular sending of the message m 1 will be replayed and that event will become n 2. So, basically here; so, this particular message will arrive here and this is an orphan message why because it sent is undone and it is going to be received over here and this will trigger the second roll back.

So, if this second roll back happens then again this step number one will be repeated and both step number one and 2 will be repeated infinitely and this will lead to a Livelock problem and this is all explained here. So, the above sequence can repeat indefinitely and this is called basically the Livelock. So, in the recovery the how this algorithms are going to handle domino and the Livelock problems is going to be a difficult task; so, that we will see in the further slides.

(Refer Slide Time: 22:37)

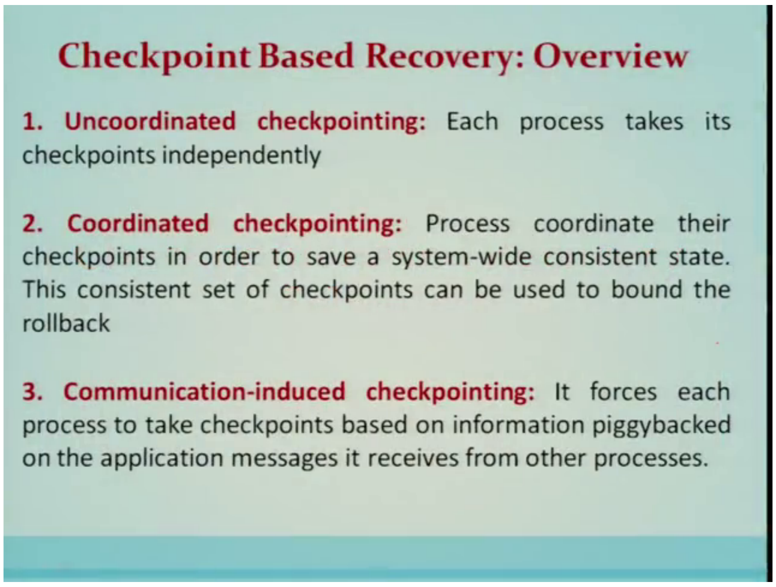


Different Rollback Recovery Schemes

Different roll back recovery schemes roll back recovery schemes can be classified into 2 different types one is the checkpointing based roll back recovery scheme the other is log based roll back recovery schemes. So, checkpointing roll back recovery schemes are also classified into 3 different types; the first one is called uncoordinated checkpointing, the second one is called coordinated checkpointing, third one is called communication induced checkpointing as far as coordinated checkpointing is concerned that is also further classified into a blocking versus non blocking checkpointing and communication induced checkpointing is also divided into 2 types that is called model based and index based checkpointing.

Log based is also classified into 3 different types pessimistic logging optimistic logging and casual logging checkpointing based recovery schemes overview checkpointing based recovery schemes are different type as we have seen in the previous slides.

(Refer Slide Time: 23:32)



Checkpoint Based Recovery: Overview

1. **Uncoordinated checkpointing:** Each process takes its checkpoints independently

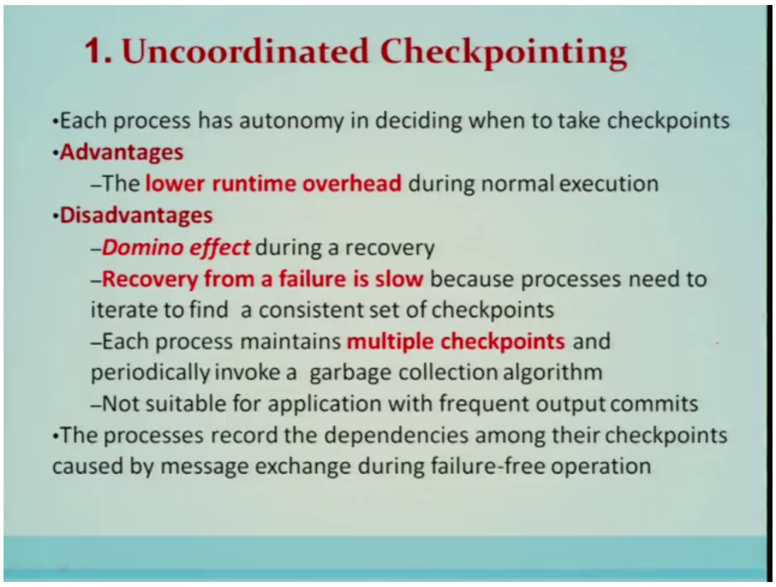
2. **Coordinated checkpointing:** Process coordinate their checkpoints in order to save a system-wide consistent state. This consistent set of checkpoints can be used to bound the rollback

3. **Communication-induced checkpointing:** It forces each process to take checkpoints based on information piggybacked on the application messages it receives from other processes.

Now, we are going to touch upon each of them the first one is called uncoordinated checkpointing here each process takes its checkpoints independently second coordinated checkpointing process coordinate their checkpoints in order to save a system wide consistent state this consistent set of checkpoints can be used to bound the roll back.

Third communication induced checkpointing, it forces each process to take checkpoints based on the information piggybacked on the application messages, it receives from the other processes besides the normal checkpoints; whether it is coordinate or uncoordinated it also includes the forced checkpoints and this will basically further optimize the checkpoints in communication induced checkpointing.

(Refer Slide Time: 24:47)



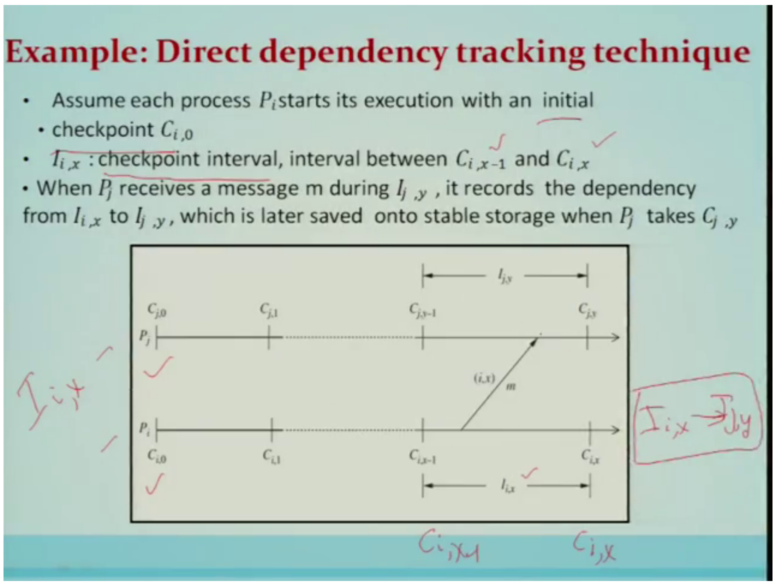
1. Uncoordinated Checkpointing

- Each process has autonomy in deciding when to take checkpoints
- **Advantages**
  - The **lower runtime overhead** during normal execution
- **Disadvantages**
  - ***Domino effect*** during a recovery
  - **Recovery from a failure is slow** because processes need to iterate to find a consistent set of checkpoints
  - Each process maintains **multiple checkpoints** and periodically invoke a garbage collection algorithm
  - Not suitable for application with frequent output commits
- The processes record the dependencies among their checkpoints caused by message exchange during failure-free operation

So, uncoordinated checkpointing each process has autonomy in deciding when to take checkpoints advantage the lower runtime overhead during the normal course of execution. Now it has many disadvantages, the first disadvantages that this uncoordinated checkpointing may lead to a domino effect during the recovery so; that means, dominoes are not avoided in uncoordinated checkpointing, second disadvantage recovery from the failure is slow why because the processes need to iterate to search a consistent set of checkpoints and that is going to take lot of time.

Now, each process maintains a multiple checkpoints and periodically invoke garbage collection algorithm and this is another disadvantage of uncoordinated checkpointing process not suitable for the application with frequent output commits and this is another disadvantage. So, the processes record the dependencies among their checkpoints caused by the message exchange during failure free operation and tracking these dependencies are very important in restoring the system state after the after the failure.

(Refer Slide Time: 26:16)



Example: Direct dependency tracking technique

- Assume each process P_i starts its execution with an initial
  - checkpoint $C_{i,0}$
- $I_{i,x}$: checkpoint interval, interval between $C_{i,x-1}$ and $C_{i,x}$
- When P_j receives a message m during $I_{j,y}$, it records the dependency from $I_{i,x}$ to $I_{j,y}$, which is later saved onto stable storage when P_j takes $C_{j,y}$

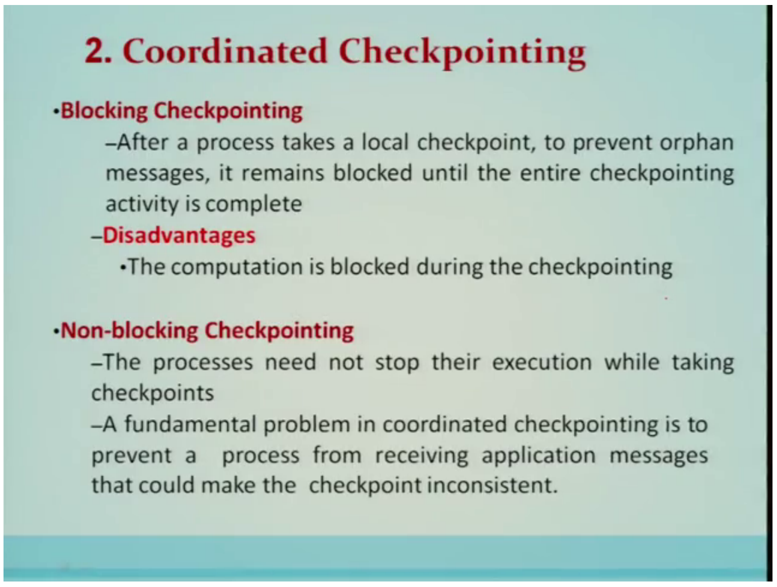
Now, how this particular dependency is to be dragged that we can see here in this kind of in this example or in this illustrative example. So, direct dependency tracking technique assume each process Pi is starts execution with an initial checkpoint Ci 0 here, Ci 0 and Cj 0, this is the initial checkpoint each process only there are 2 processes here in this example in the system. So, they will take the initial checkpoints.

Now, then we are going to define an checkpoint interval checkpoint interval is represented by capital Ix i comma x; that means,. So, i comma x is basically the interval between the checkpoints Ci x minus one to Ci x this interval is represented by i capital I i comma x. So, i comma x that is checkpoint interval is the interval between the 2 checkpoints of a process i between x minus one th and x th.

Now, when a process Pj receives a message m during its interval checkpoint interval that is capital I j comma y it recast the dependency from i capital I i comma x to capital ij comma y. So, this particular dependency ensures that between these intervals the flow of message basically creates a dependency between these 2 processes and once this dependency is saved and saved onto a stable storage and when the Pj takes its basically checkpoint Cjy.

So,. So, these dependencies are basically tracked and they are stored. So, that at the time of recovery in the uncoordinated checkpointing, they will be used to basically eliminate or avoid the domino effect another type of checkpointing is called coordinated checkpointing.

(Refer Slide Time: 28:54)



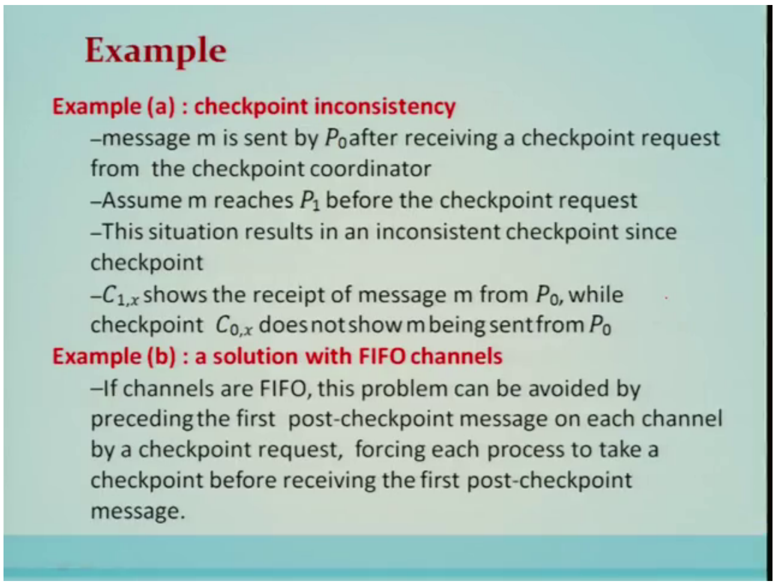
2. Coordinated Checkpointing

- **Blocking Checkpointing**
  - –After a process takes a local checkpoint, to prevent orphan messages, it remains blocked until the entire checkpointing activity is complete
  - –**Disadvantages**
    - •The computation is blocked during the checkpointing
- **Non-blocking Checkpointing**
  - –The processes need not stop their execution while taking checkpoints
  - –A fundamental problem in coordinated checkpointing is to prevent a process from receiving application messages that could make the checkpoint inconsistent.

Coordinated set pointing are of 2 types that we have seen in the previous slides the first one is the blocking checkpointing.

So, after a process takes a local checkpoint to prevent the orphan messages, it remains blocked until the entire checkpointing activity is complete so; that means, while the checkpointing is taking place, it is not allowed to send further messages hence it is called a blocked state the disadvantage here the complication is blocked during the checkpointing state another type of checkpointing is called here non blocking checkpointing in a coordinated checkpointing; the processes need not stop their execution while taking the checkpoints the fundamental problem here in coordinated checkpointing is to prevent a process from receiving application messages that could make the checkpoint inconsistent.

(Refer Slide Time: 29:47)



Example

Example (a) : checkpoint inconsistency

–message m is sent by P_0after receiving a checkpoint request from the checkpoint coordinator

–Assume m reaches P_1 before the checkpoint request

–This situation results in an inconsistent checkpoint since checkpoint

–$C_{1,x}$ shows the receipt of message m from P_0, while checkpoint $C_{0,x}$ does not show m being sent from P_0

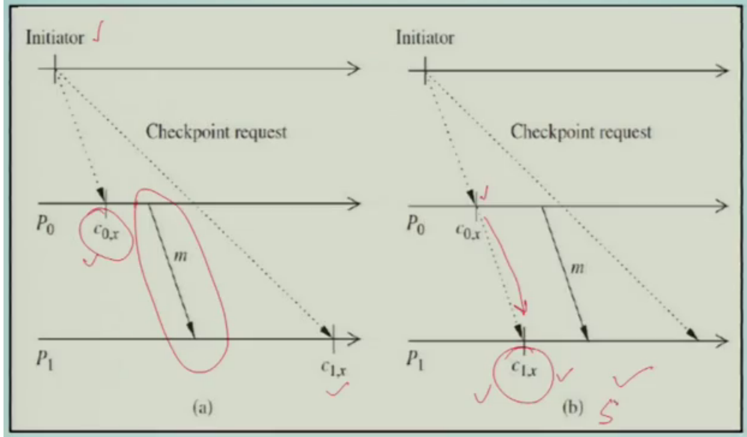
Example (b) : a solution with FIFO channels

–If channels are FIFO, this problem can be avoided by preceding the first post-checkpoint message on each channel by a checkpoint request, forcing each process to take a checkpoint before receiving the first post-checkpoint message.

So, here the examples checkpoints inconsistency message m is sent by P 0 after receiving a checkpoint request from a checkpoint coordinator assume m reaches P 1 before checkpoint request the situation results in inconsistent checkpoints in the checkpoint.

(Refer Slide Time: 30:07)

Contd...



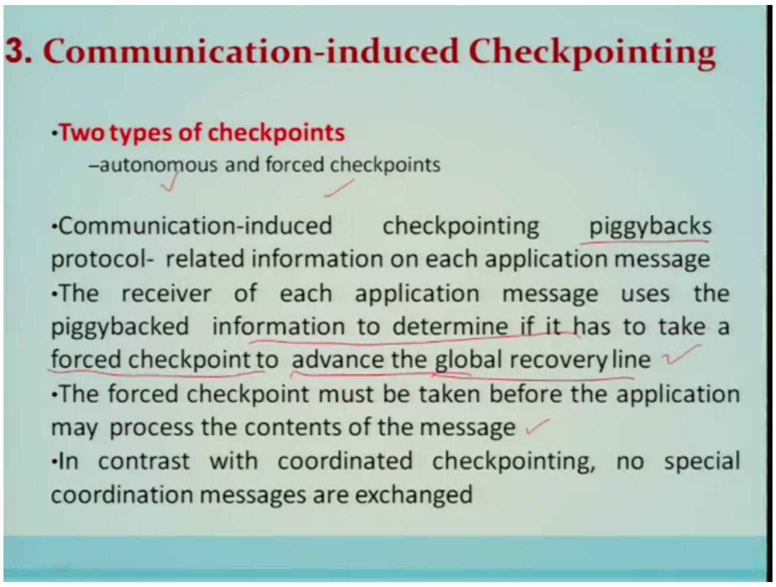
So, this we can see in this particular illustrative example. So, as in as we were talking about that that this is the initiator of a checkpoint. So, it will send this checkpoint request. So, this particular request reaches at this instance to P 0 and P 0 will take its checkpoint that is C 0 x.

And after that it will send a message to P one. So, P 0 will send a message to m and then the initiator process will come at this instant. So, basically the clocks are not synchronized or the communication channels are basically having sufficient delay. So, that all the checkpoints are not basically carried out simultaneously, but in between these 2 checkpoints these 2 processes checkpoint just see that there is a flow of message.

So, what is to be done over here is that. So, as soon as the checkpoint message is received and P 0 is taking a checkpoint, it will send an basically at another message to the next particular process that is its P 1 and this P 1 message will arise or will reach before the initiators message could reach and this particular P 0 message will initiate P 1 to take a checkpoint at this instant.

So, this particular message which P 0 will send afterwards will receive after P 1 has taken checkpoint and this is the correct state. So, the solution as you know that it is a FIFO channels. So, if the FIFO channels are there then this problem can be avoided by preceding the first message post checkpoint message on each channel by a checkpoint request forcing each process to take a checkpoint before receiving the first post checkpoint message and that is being explained.

(Refer Slide Time: 32:23)



3. Communication-induced Checkpointing

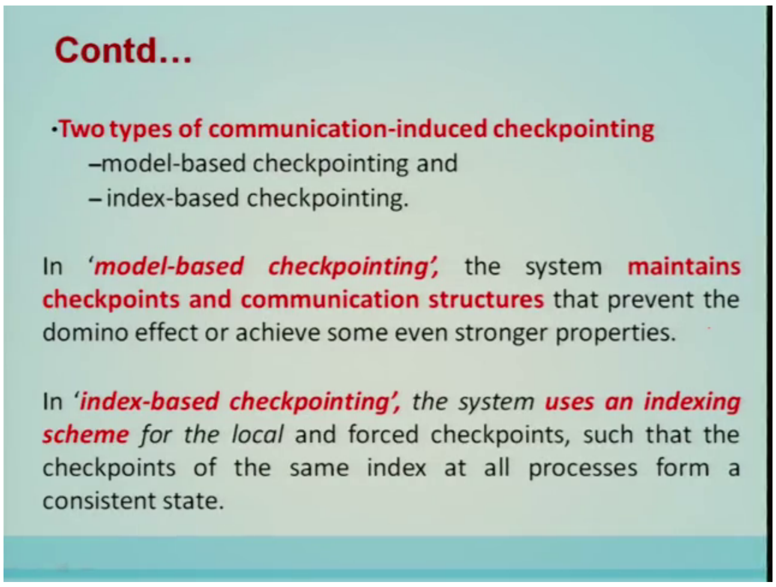
•Two types of checkpoints
–autonomous and forced checkpoints

•Communication-induced checkpointing piggybacks protocol- related information on each application message
•The receiver of each application message uses the piggybacked information to determine if it has to take a forced checkpoint to advance the global recovery line
•The forced checkpoint must be taken before the application may process the contents of the message
•In contrast with coordinated checkpointing, no special coordination messages are exchanged

Now, third type of checkpointing is called communication induced checkpointing there are 2 types of checkpoints here in this category the autonomous and force checkpoint that I have basically explained you before. So, in a communication induced checkpointing, piggybacks protocol related information on each application message. So, this particular checkpointing information is piggybacked on the application messages. So, the receiver of each application message is using this piggybacked information to determine it has to take a forced checkpoint to advance the global recovery line the forced checkpoint must be taken before the application may process the content of the message in contrast with the coordinated checkpointing no special coordinated messages are exchanged.

So, autonomous checkpoints are the normal checkpoints and forced checkpoints are sent through the message which will basically as mentioned over here, it will advance the global recovery line.

(Refer Slide Time: 33:32)



Contd...

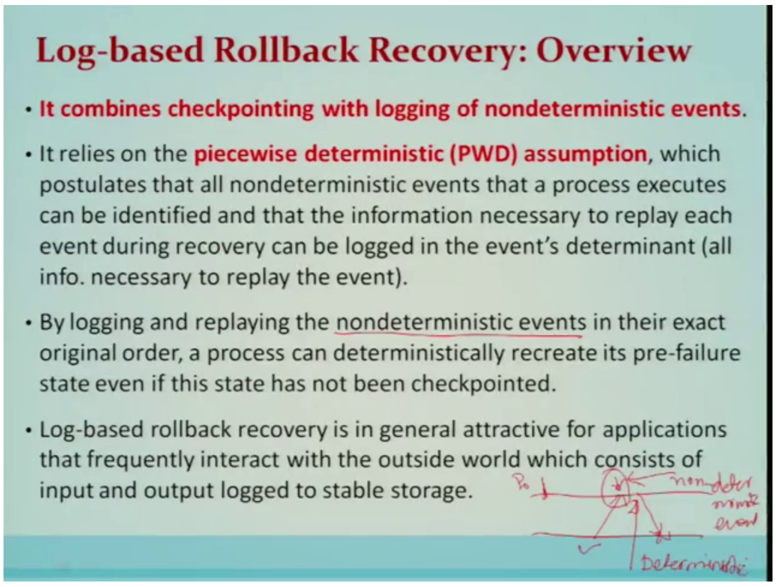
- Two types of communication-induced checkpointing
  - model-based checkpointing and
  - index-based checkpointing.

In *'model-based checkpointing'*, the system **maintains checkpoints and communication structures** that prevent the domino effect or achieve some even stronger properties.

In *'index-based checkpointing', the system **uses an indexing scheme** for the local* and forced checkpoints, such that the checkpoints of the same index at all processes form a consistent state.

So, it will further optimize and optimize the recovery process 2 types of communication induced checkpointing is there; one is called model based checkpointing, the other one is called index based checkpointing in a model based checkpointing the system maintains the checkpoints and the communication structures that prevents the domino effect are achieved some even stronger properties index base checkpointing the system uses an indexing scheme for the local and forced checkpoints such that the checkpoints of the same index at all the process forms a consistent state.

(Refer Slide Time: 34:09)



Log-based Rollback Recovery: Overview

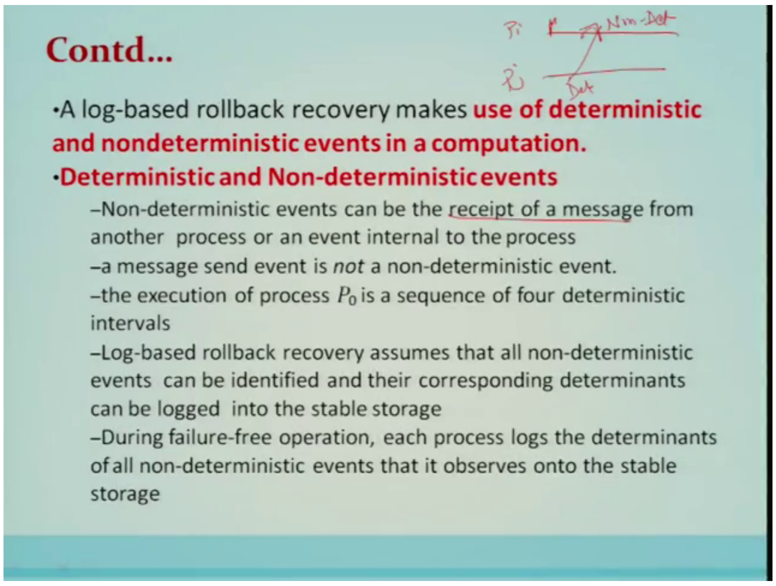
- **It combines checkpointing with logging of nondeterministic events.**
- It relies on the **piecewise deterministic (PWD) assumption**, which postulates that all nondeterministic events that a process executes can be identified and that the information necessary to replay each event during recovery can be logged in the event's determinant (all info. necessary to replay the event).
- By logging and replaying the nondeterministic events in their exact original order, a process can deterministically recreate its pre-failure state even if this state has not been checkpointed.
- Log-based rollback recovery is in general attractive for applications that frequently interact with the outside world which consists of input and output logged to stable storage.

Log based roll back recovery schemes log based roll back recovery schemes it combines checkpointing with logging of non deterministic events. So, it relies on piecewise deterministic assumptions which postulates that all non deterministic events that the process executes can be identified and that the information necessary to replay each event during the recovery can be lost in the events determinants.

So, by logging and replaying the non deterministic events in their exact original order the process can deterministically recreate its pre failure state even if this state has not been checked pointed. So, non deterministic events are the receipt of the event the receipt of a message. So, in this particular example; so, the message which is received at this end its cannot be determined cannot be known when it is going to be received, but send of a message or a start of a process P 0 that is basically called deterministic events.

So, important thing is for these deterministic. So, this non deterministic event they are they are deterministic parameters are to be basically found out and to be stored. So, this is called the non deterministic and send of a message is a deterministic event.

(Refer Slide Time: 35:53)



Contd...

Pi
Non-Det
Pj
Det

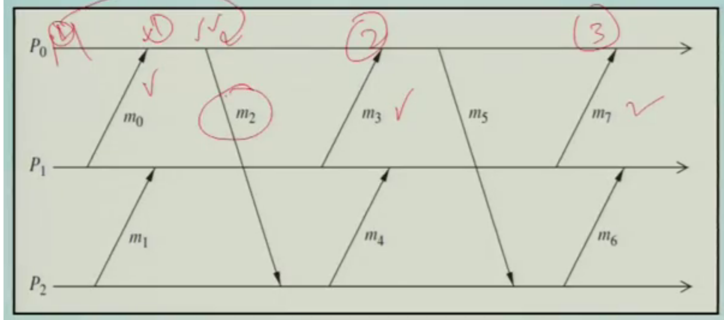
- A log-based rollback recovery makes **use of deterministic and nondeterministic events in a computation.**
- **Deterministic and Non-deterministic events**
  - Non-deterministic events can be the receipt of a message from another process or an event internal to the process
  - a message send event is *not* a non-deterministic event.
  - the execution of process P_0 is a sequence of four deterministic intervals
  - Log-based rollback recovery assumes that all non-deterministic events can be identified and their corresponding determinants can be logged into the stable storage
  - During failure-free operation, each process logs the determinants of all non-deterministic events that it observes onto the stable storage

So, log based roll back recovery makes use of deterministic and non deterministic events in the computation deterministic and non deterministic events as I explained you in the previous slide, the non deterministic events can be the receipt of a message from another process here like this process i and process j. So, this is the receipt of a message is a non deterministic.

A message send event is not a non deterministic event. So, send is basically deterministic the execution of a process P 0 here.

(Refer Slide Time: 36:47)

Deterministic and Non-deterministic events: Example

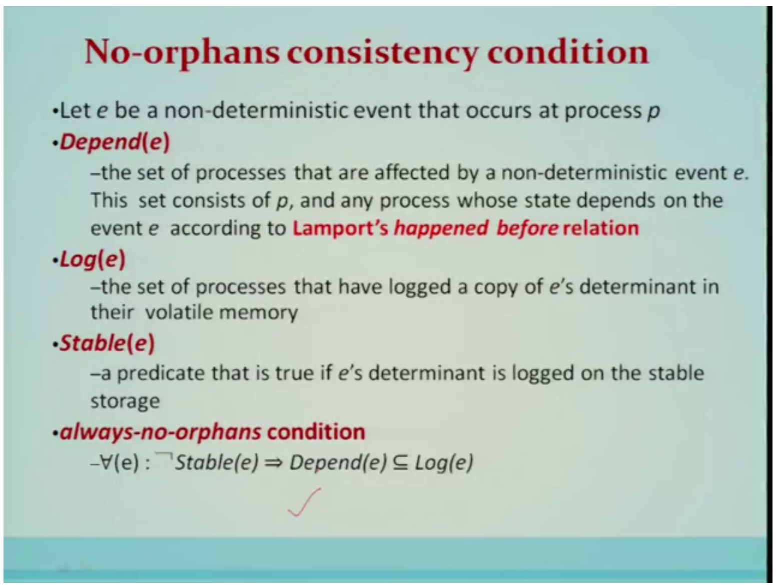


The execution of process P_0 is a sequence of four deterministic intervals. The first one starts with the creation of the process, while the remaining three start with the receipt of messages m_0, m_3, and m_7, respectively. Send event of message m_2 is uniquely determined by the initial state of P_0 and by the receipt of message m_0, and is therefore not a non-deterministic event.

When it is start is a sequence of 4 intervals here it is shown. So, it is I start of a event and then. So, here the execution of P 0 is the sequence of 4 deterministic event. So, this is the deterministic event one this is the deterministic event 2 and this is deterministic even 3-4 deterministic event the first one starts with the creation of a process while remaining 3 starts with the received of a receipt of a message this is 1 this is 2, this is 3 and this is 4; 3 starts with the receipt of a message m 0, m 3, m 0, m 3 and m 7 respectively.

So, send event of the message m 2 is uniquely determined by the initial state of the process P 0 and by the receipt of a message m 0 and therefore, it is not a non deterministic event. So, so; that means, with the relation to the start of a process you can understand or you can determine the instance when this send of this particular message is send.

(Refer Slide Time: 38:12)



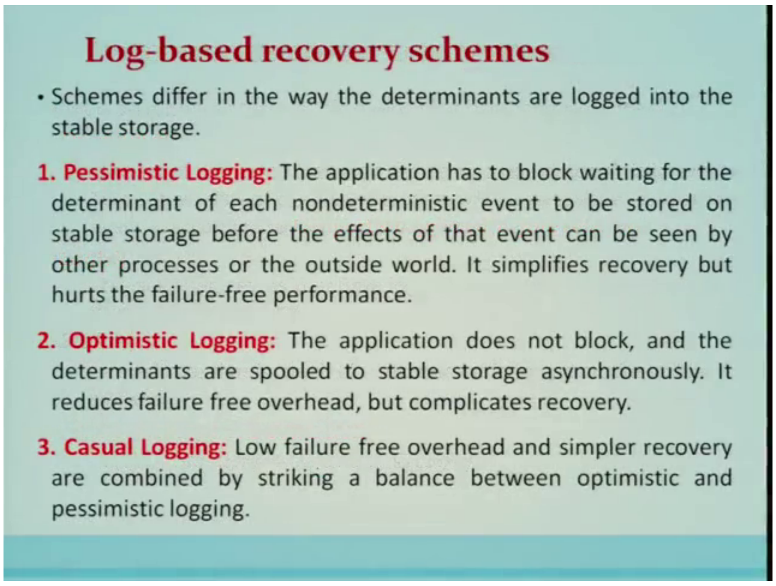
No-orphans consistency condition

- Let *e* be a non-deterministic event that occurs at process *p*
- ***Depend*(*e*)**
  - the set of processes that are affected by a non-deterministic event *e*. This set consists of *p*, and any process whose state depends on the event *e* according to **Lamport's *happened before* relation**
- ***Log*(*e*)**
  - the set of processes that have logged a copy of *e*'s determinant in their volatile memory
- ***Stable*(*e*)**
  - a predicate that is true if *e*'s determinant is logged on the stable storage
- ***always-no-orphans* condition**
  - $\forall(e): \neg Stable(e) \Rightarrow Depend(e) \subseteq Log(e)$

Hence, it is not a non deterministic it is a deterministic event no orphan consistency condition let e be a non deterministic event that occurs at a process P. So, depend e is basically the set of processes that are affected by a non deterministic even e this set consists of P and any process whose state depends on the even e according to the Lamport's happened before relation.

Then log e the set of processes that have logged a copy of es determinants in their volatile memory stable a predicate that is true if each determinant is logged on the stable storage always no orphan condition.

(Refer Slide Time: 38:58)



Log-based recovery schemes

- Schemes differ in the way the determinants are logged into the stable storage.

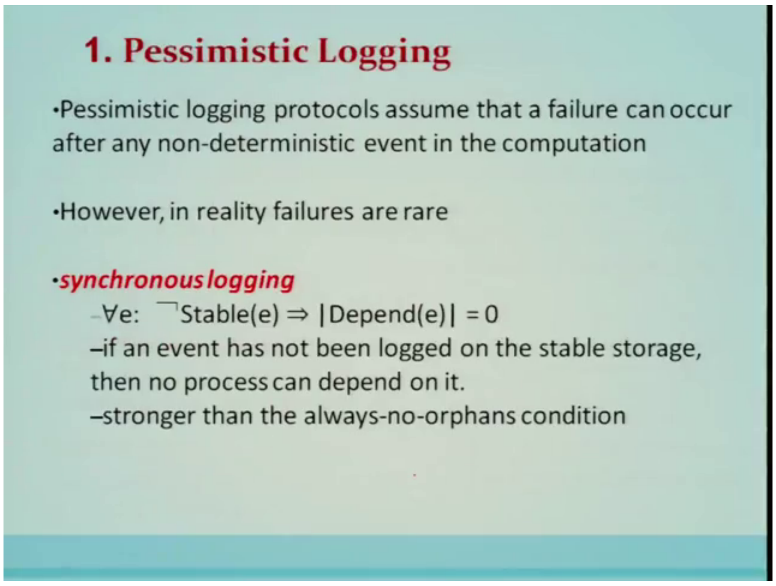
1. **Pessimistic Logging:** The application has to block waiting for the determinant of each nondeterministic event to be stored on stable storage before the effects of that event can be seen by other processes or the outside world. It simplifies recovery but hurts the failure-free performance.
2. **Optimistic Logging:** The application does not block, and the determinants are spooled to stable storage asynchronously. It reduces failure free overhead, but complicates recovery.
3. **Casual Logging:** Low failure free overhead and simpler recovery are combined by striking a balance between optimistic and pessimistic logging.

So; that means, if this condition is satisfied then there is no possibility of any orphan message. So, it says that if it is; the dependencies are not stored in the stable then basically it is stored in the in the log or it is logged in the stable storage.

Log based recovery schemes differ in the way that determinants are logged on to the stable storage and there are of 3 types pessimistic logging optimistic logging and the casual logging pessimistic logging the application has to block waiting for the determinants of each nondeterministic even to be stored on stable storage before the effects of that message can be seen by the other process or the outside world.

Optimistic logging the application does not block and the determinants are spooled to the stable storage asynchronously casual logging no failure free overhead.

(Refer Slide Time: 40:04)



1. Pessimistic Logging

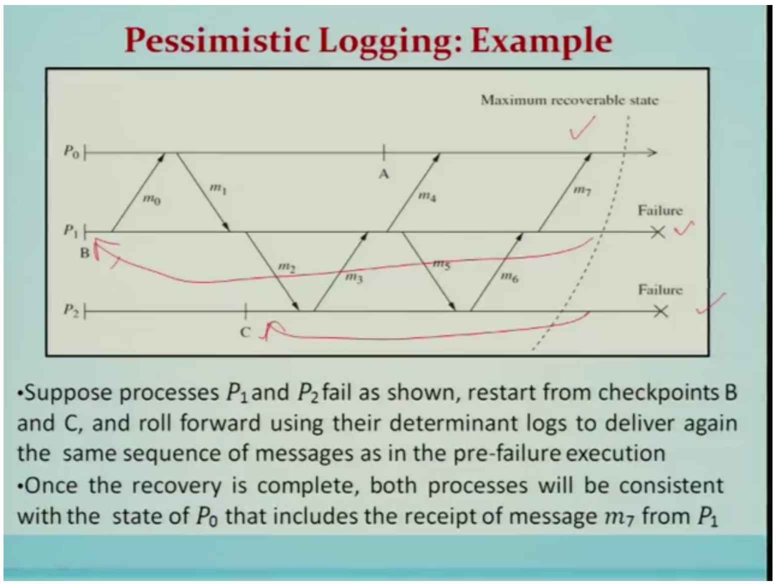
- Pessimistic logging protocols assume that a failure can occur after any non-deterministic event in the computation
- However, in reality failures are rare
- ***synchronous logging***
  - $\forall e: \neg Stable(e) \Rightarrow |Depend(e)| = 0$
  - if an event has not been logged on the stable storage, then no process can depend on it.
  - stronger than the always-no-orphans condition

And simpler recovery are combined by striking a balance between optimistic and a pessimistic logging pessimistic logging protocols assume that the failure can occur after any non deterministic event in the computation.

However in reality the failures are variants are rare synchronous logging if an event has not been logged on the stable storage then no process can depend on it stronger than always no orphan condition.

(Refer Slide Time: 40:30)

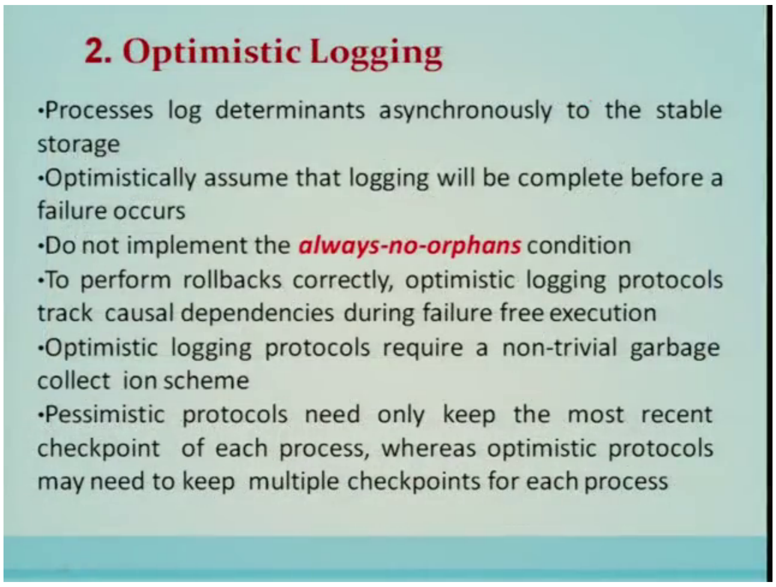
Pessimistic Logging: Example



- Suppose processes P_1 and P_2 fail as shown, restart from checkpoints B and C, and roll forward using their determinant logs to deliver again the same sequence of messages as in the pre-failure execution
- Once the recovery is complete, both processes will be consistent with the state of P_0 that includes the receipt of message m_7 from P_1

So, here this is the example of a pessimistic logging suppose P 1 and P 2 fails here at this particular instance then it will restart from checkpoints B and C it will restart from B and B and C and roll forward using their determinant log to deliver again the same sequence of the messages as in the pre failure execution once the recovery is complete both the processes will be a consistent with the state of P 0 that includes the receipt of a message m 7 from P 1.

(Refer Slide Time: 41:17)

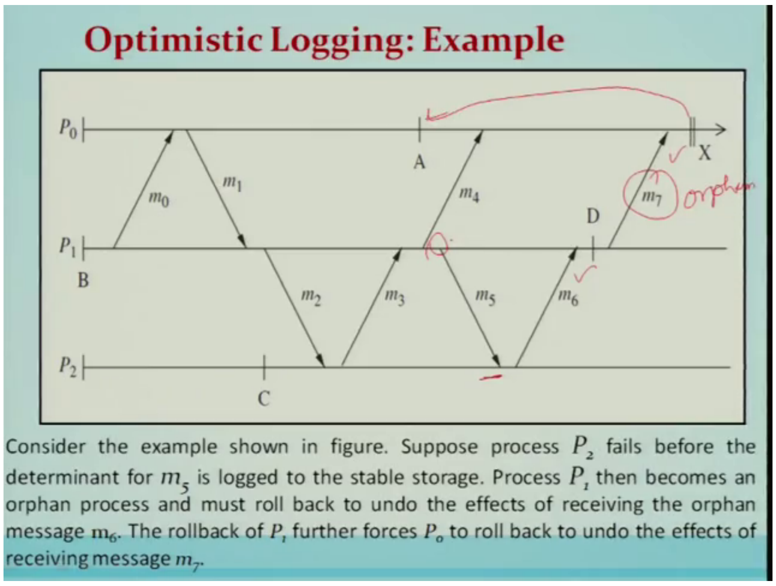


2. Optimistic Logging

- Processes log determinants asynchronously to the stable storage
- Optimistically assume that logging will be complete before a failure occurs
- Do not implement the ***always-no-orphans*** condition
- To perform rollbacks correctly, optimistic logging protocols track causal dependencies during failure free execution
- Optimistic logging protocols require a non-trivial garbage collect ion scheme
- Pessimistic protocols need only keep the most recent checkpoint of each process, whereas optimistic protocols may need to keep multiple checkpoints for each process

So; that means, using the logging, it will be taken care of optimistic logging processes log determinants asynchronously to the stable storage optimistically assume that the logging will be complete before a failure occurs do not implement always no orphan condition. So, to perform roll back correctly optimistic logging protocol track casual dependencies during the failure free execution optimistic logging protocols require non trivial garbage collection schemes pessimistic protocols need only keep most recent checkpoints of each process whereas, optimistic we need to keep track of multiple checkpoints and that will require more memory for storing the checkpoints.

(Refer Slide Time: 42:04)

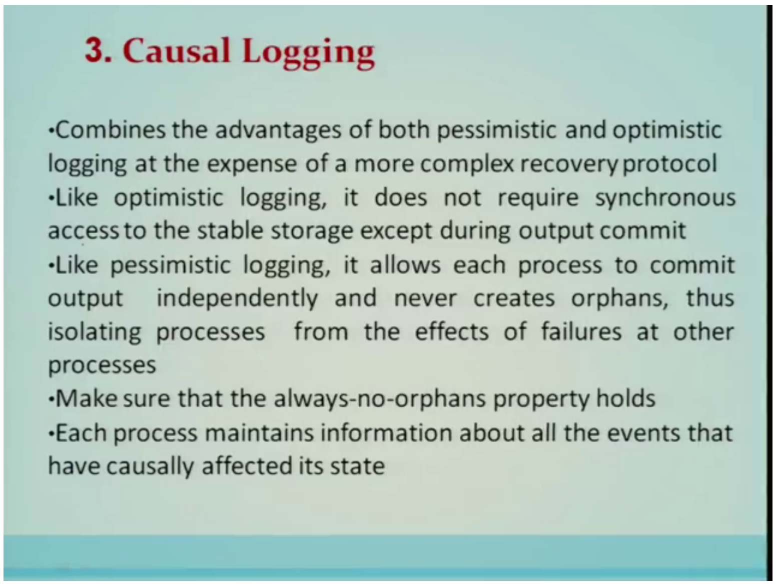


Optimistic Logging: Example

Consider the example shown in figure. Suppose process P_2 fails before the determinant for m_5 is logged to the stable storage. Process P_1 then becomes an orphan process and must roll back to undo the effects of receiving the orphan message m_6. The rollback of P_1 further forces P_0 to roll back to undo the effects of receiving message m_7.

So, in optimistic logging consider the example suppose P 2 fails before the determinant of for m 5 is logged here is logged onto the stable storage determinant because determinants is this one logged on a stable storage process P 1 then becomes an orphan process P 1 then becomes and must roll back to undone the effects of receiving the orphan message m 6 the roll back of P 1 further forces P 0 to roll back to undo the effects of the receiving message m 7 why because this will become an orphan message. So, it has to roll back why because its send is not recorded because it has to roll back at this point. So, this is called optimistic logging.

(Refer Slide Time: 42:59)

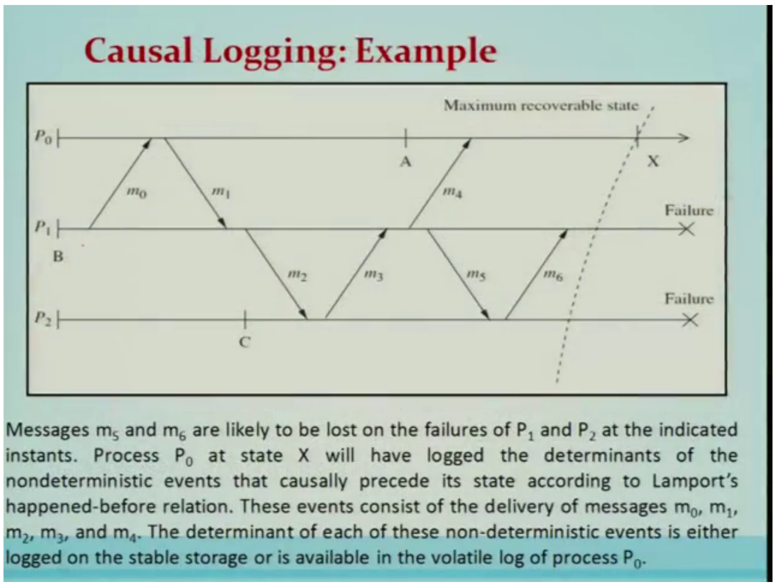


3. Causal Logging

- Combines the advantages of both pessimistic and optimistic logging at the expense of a more complex recovery protocol
- Like optimistic logging, it does not require synchronous access to the stable storage except during output commit
- Like pessimistic logging, it allows each process to commit output independently and never creates orphans, thus isolating processes from the effects of failures at other processes
- Make sure that the always-no-orphans property holds
- Each process maintains information about all the events that have causally affected its state

Casual logging combines the advantage of both pessimistic and optimistic logging at the expense of more complex recovery protocol.

(Refer Slide Time: 43:09)

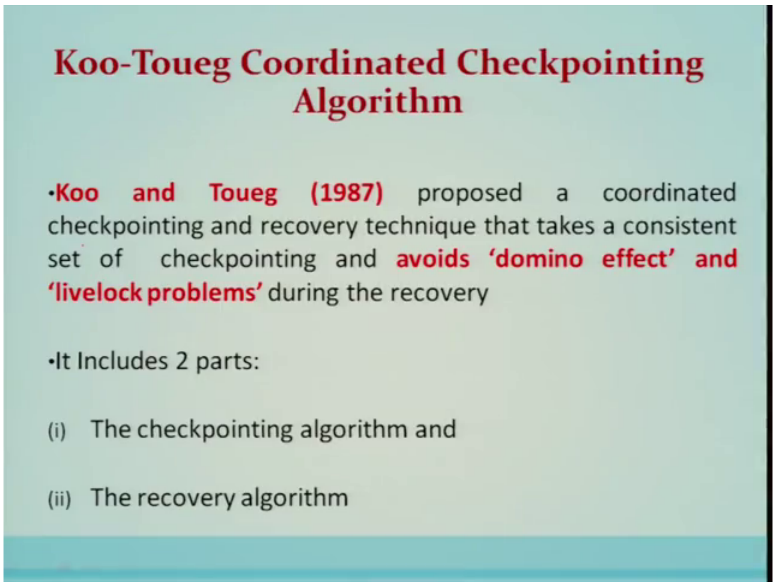


Causal Logging: Example

Messages m_5 and m_6 are likely to be lost on the failures of P_1 and P_2 at the indicated instants. Process P_0 at state X will have logged the determinants of the nondeterministic events that causally precede its state according to Lamport's happened-before relation. These events consist of the delivery of messages m_0, m_1, m_2, m_3, and m_4. The determinant of each of these non-deterministic events is either logged on the stable storage or is available in the volatile log of process P_0.

So, this particular example shows the casual logging checkpointing and recovery algorithms.

(Refer Slide Time: 43:18)



Koo-Toueg Coordinated Checkpointing Algorithm

•**Koo and Toueg (1987)** proposed a coordinated checkpointing and recovery technique that takes a consistent set of checkpointing and **avoids 'domino effect' and 'livelock problems'** during the recovery

•It Includes 2 parts:

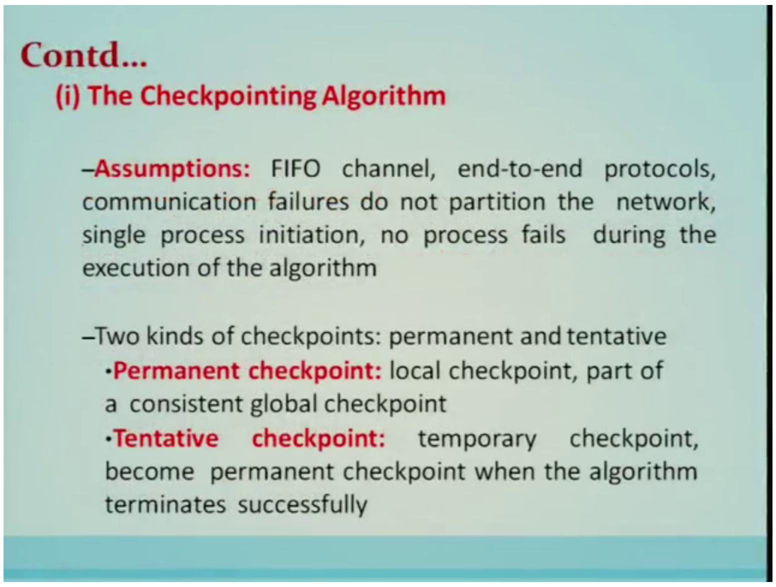
(i) The checkpointing algorithm and

(ii) The recovery algorithm

Koo Toueg coordinated checkpointing algorithm; Koo Toueg in 1987 proposed a coordinated checkpointing and recovery technique that takes a consistent set of checkpointing and avoids domino effect and Livelock problems during the recovery.

So, we will see this is the algorithm which will basically use to solve the problem of domino effect and Livelock and through the coordinated checkpointing and it includes 2 part; the first is called checkpointing algorithm.

(Refer Slide Time: 43:54)



Contd...

(i) The Checkpointing Algorithm

–**Assumptions:** FIFO channel, end-to-end protocols, communication failures do not partition the network, single process initiation, no process fails during the execution of the algorithm

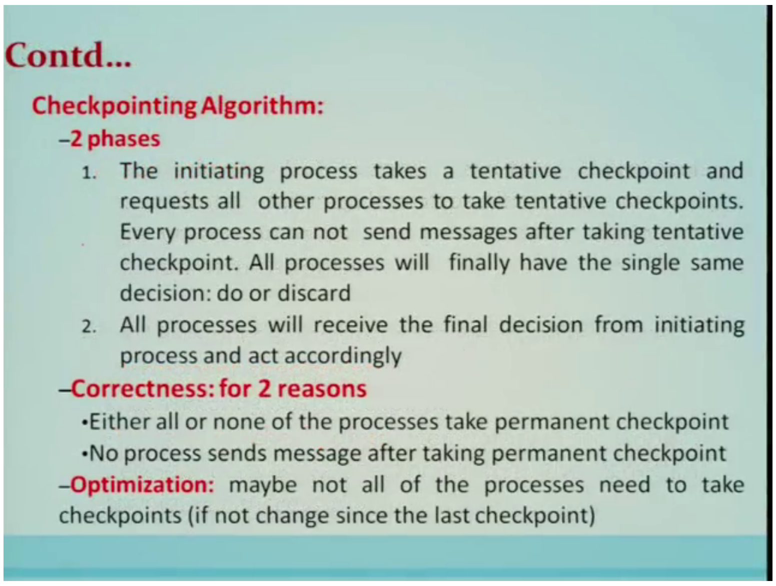
–Two kinds of checkpoints: permanent and tentative

- **Permanent checkpoint:** local checkpoint, part of a consistent global checkpoint
- **Tentative checkpoint:** temporary checkpoint, become permanent checkpoint when the algorithm terminates successfully

The other part is called recovery algorithm in checkpointing algorithm assumptions are that the algorithm assumes that the channels are following FIFO that is and it is also assumed that it is end to end protocol and communication failures do not partition the network and there is a single process for initiation and no process fails during the execution of the algorithm.

So, with this assumption the checkpointing algorithms are basically as follows. So, there are 2 kinds of checkpoints, here in this algorithm the first is called permanent the another one is called tentative checkpoints permanent checkpoint is the local checkpoint a part of a consistent global checkpoint tentative checkpoint a temporary checkpoint become permanent checkpoint when the algorithm terminates successfully.

(Refer Slide Time: 44:45)



Contd...

Checkpointing Algorithm:

–**2 phases**

1. The initiating process takes a tentative checkpoint and requests all other processes to take tentative checkpoints. Every process can not send messages after taking tentative checkpoint. All processes will finally have the single same decision: do or discard
2. All processes will receive the final decision from initiating process and act accordingly

–**Correctness: for 2 reasons**

•Either all or none of the processes take permanent checkpoint

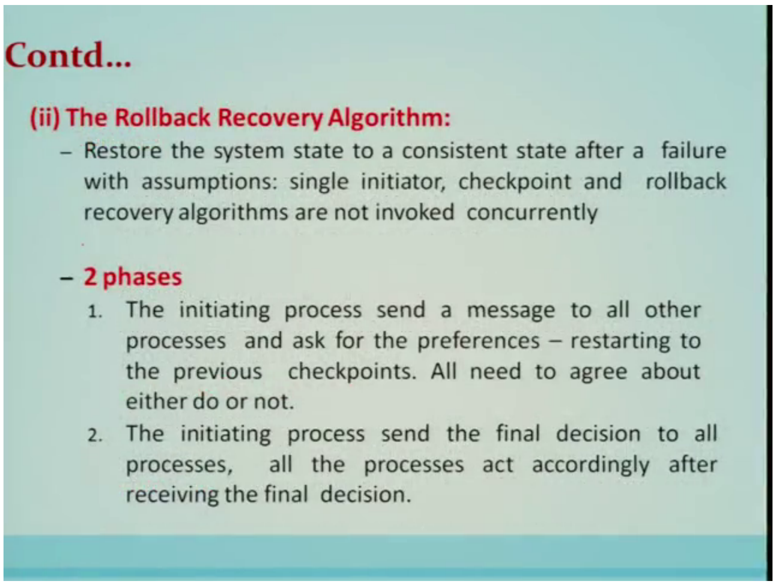
•No process sends message after taking permanent checkpoint

–**Optimization:** maybe not all of the processes need to take checkpoints (if not change since the last checkpoint)

Now, checkpointing algorithm; it has 2 phases in phase one the initiating process takes a tentative checkpoint and requests all other processes to take a tentative checkpoints now every process cannot send the message after taking the checkpoint all process will finally, have the single same decision do or discard all processes will receive the final decision from the initiating process and act accordingly in the second phase.

Now, correctness there are 2 reasons either all of or none of the processes will decide and take the permanent checkpoint. So, no process sends a message after taking a permanent checkpoint hence this basically these 2 conditions basically ensures the correctness optimization may be that not all of the processes need to take the checkpoints. So, if no;t change since the last checkpoints; so, that particular checkpoint that that point no change since the last checkpoints. So, there is a possibility of optimization here and in the checkpointing phase.

(Refer Slide Time: 45:51)

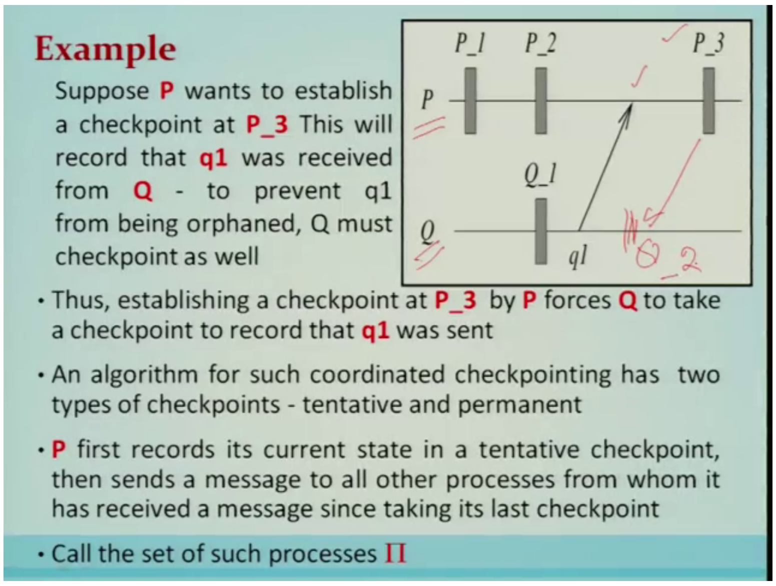

Contd...

(ii) The Rollback Recovery Algorithm:

- Restore the system state to a consistent state after a failure with assumptions: single initiator, checkpoint and rollback recovery algorithms are not invoked concurrently
- **2 phases**
  1. The initiating process send a message to all other processes and ask for the preferences – restarting to the previous checkpoints. All need to agree about either do or not.
  2. The initiating process send the final decision to all processes, all the processes act accordingly after receiving the final decision.

Now, another part of this algorithm is called roll back recovery algorithm restore the system state to a consistent state after the failure with the assumptions have the single initiator checkpoint and roll back recovery algorithms are not invoked concurrently. So, roll back recovery has 2 phases. So, the first phase says that initiating process send a message to all other process and ask for the preference that is restarting to a previous checkpoints all need to agree about either do or do not.

(Refer Slide Time: 46:43)


Example

Suppose **P** wants to establish a checkpoint at **P_3** This will record that **q1** was received from **Q** - to prevent q1 from being orphaned, Q must checkpoint as well

P_1
P_2
P_3
P
Q_1
Q
ql
Q_2

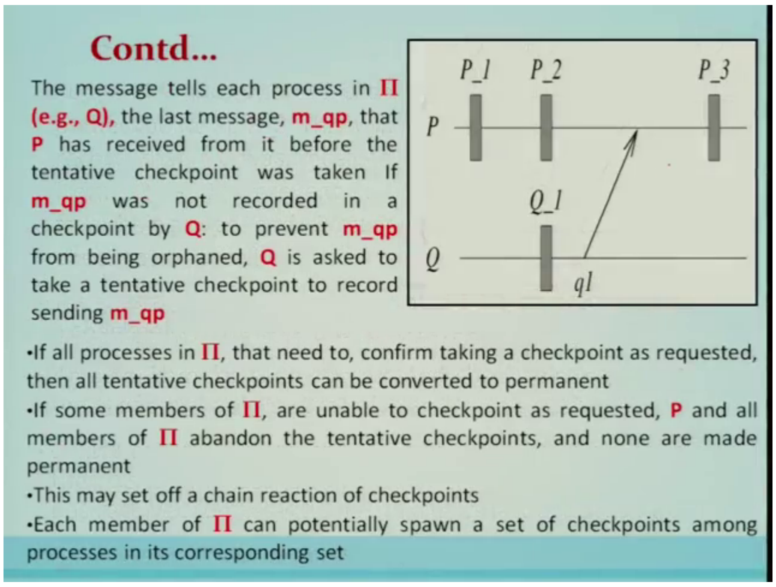
- Thus, establishing a checkpoint at **P_3** by **P** forces **Q** to take a checkpoint to record that **q1** was sent
- An algorithm for such coordinated checkpointing has two types of checkpoints - tentative and permanent
- **P** first records its current state in a tentative checkpoint, then sends a message to all other processes from whom it has received a message since taking its last checkpoint
- Call the set of such processes Π

So, the initiating process, then send the final decision to all the process and all the process x accordingly after receiving the final decision for the roll back. So, this algorithm we can explain what this algorithm is now doing we can explain through this example here suppose process P want to establish a checkpoint at P 3; this will record that Q one was received from the process Q to prevent Q 1 from being orphaned Q must also take the checkpoint as well after sending the Q one it has to take a checkpoint thus establishing a checkpoint at P 3 forces Q to take a checkpoint to record that Q one was sent.

So, this basically we can say that it is the Q 2 this checkpoint this P 3 will force the process Q to take a checkpoint. So, an algorithm for such coordinated checkpointing has 2 types of checkpoint the tentative checkpoint and a permanent checkpoint that I am explained. So, the process P first records its current state in a tentative checkpoint and then sends a message to all other process from whom; it has received a message since taking its last checkpoints.

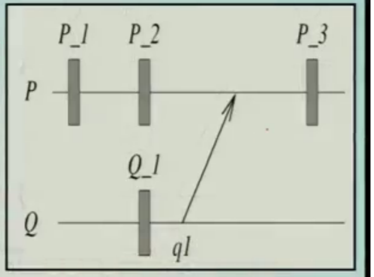
So, call this set of such processes which are required to take a checkpoint and basically it will be coordinated checkpointing as far as this particular process concerned.

(Refer Slide Time: 48:16)



Contd...

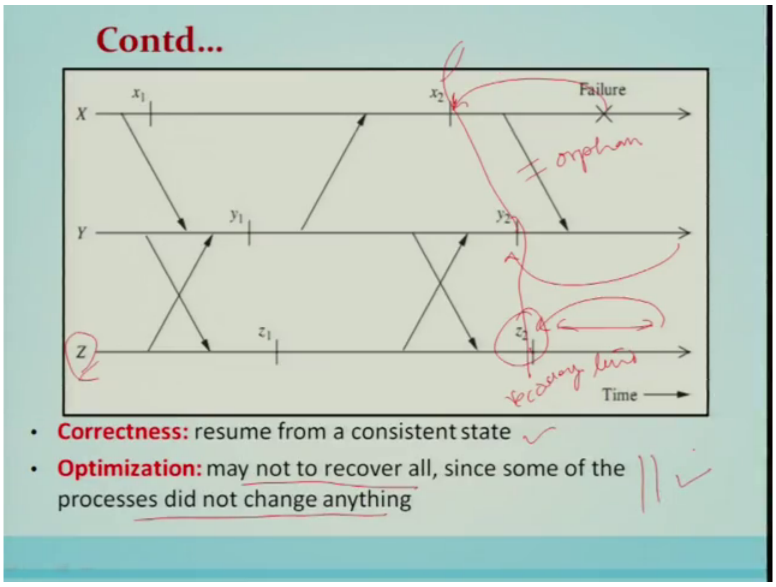
The message tells each process in Π (e.g., Q), the last message, m_qp, that P has received from it before the tentative checkpoint was taken If m_qp was not recorded in a checkpoint by Q: to prevent m_qp from being orphaned, Q is asked to take a tentative checkpoint to record sending m_qp



- If all processes in Π, that need to, confirm taking a checkpoint as requested, then all tentative checkpoints can be converted to permanent
- If some members of Π, are unable to checkpoint as requested, P and all members of Π abandon the tentative checkpoints, and none are made permanent
- This may set off a chain reaction of checkpoints
- Each member of Π can potentially spawn a set of checkpoints among processes in its corresponding set

So, the message tells each process the last message that P has received from it before tentative checkpoint was taken if that particular message was not recorded in a checkpoint by Q to prevent from being orphaned Q was asked to take a tentative checkpoints and this is basically done here in this case.

(Refer Slide Time: 48:44)



Contd...

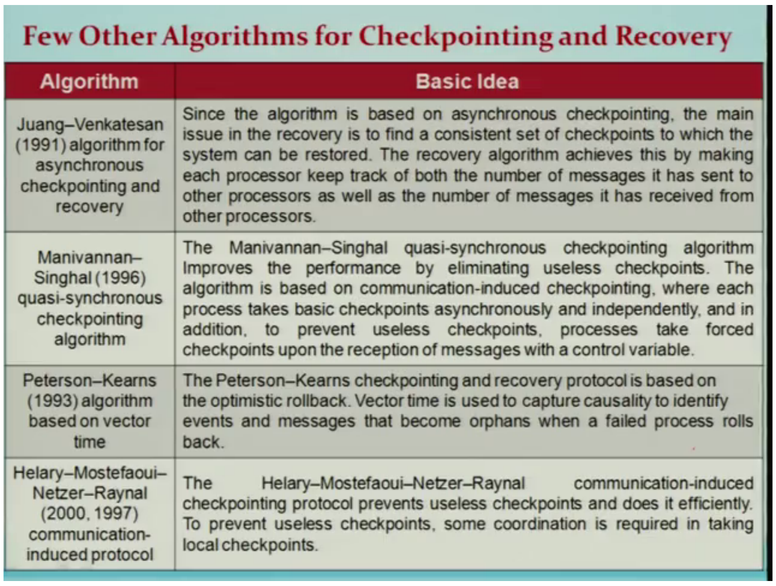
- **Correctness:** resume from a consistent state
- **Optimization:** may not to recover all, since some of the processes did not change anything

Now, as far as roll back recovery there is a possibility of optimization and also there is a correctness that it will resume from a consistent state take this example if there is a failure it will roll back or it will basically select x 2 as the checkpoint a consistent checkpoint this will basically trigger that this particular message is orphan and this will also lead why to be taken into a checkpoints. So, this recovery line is like this.

Now, here in this case this particular algorithm although has basically using its roll back able to basically able to found out the recovery line, but you can see here that z from the previous checkpoint there is no event no activity happened. So, there is no need of basically roll back of the process z and that is the optimization. So, may not to recover all the processes in some of the processes did not change anything from the last checkpoint and from the last checkpoint here the events are happening. So, they have to roll back, but process z does not have to roll back.

So, there is a possibility of optimization of this particular algorithm of the number of roll backs processes who are required to be roll back.

(Refer Slide Time: 50:34)

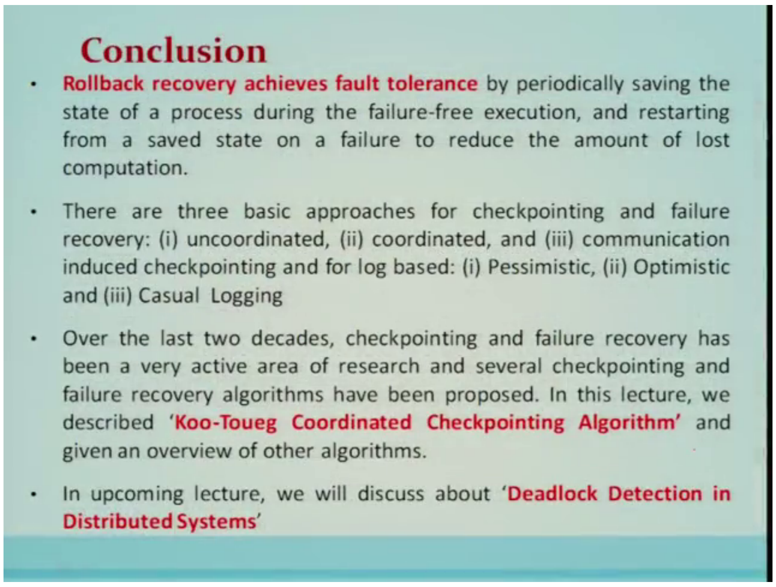


Few Other Algorithms for Checkpointing and Recovery

Algorithm	Basic Idea
Juang–Venkatesan (1991) algorithm for asynchronous checkpointing and recovery	Since the algorithm is based on asynchronous checkpointing, the main issue in the recovery is to find a consistent set of checkpoints to which the system can be restored. The recovery algorithm achieves this by making each processor keep track of both the number of messages it has sent to other processors as well as the number of messages it has received from other processors.
Manivannan–Singhal (1996) quasi-synchronous checkpointing algorithm	The Manivannan–Singhal quasi-synchronous checkpointing algorithm Improves the performance by eliminating useless checkpoints. The algorithm is based on communication-induced checkpointing, where each process takes basic checkpoints asynchronously and independently, and in addition, to prevent useless checkpoints, processes take forced checkpoints upon the reception of messages with a control variable.
Peterson–Kearns (1993) algorithm based on vector time	The Peterson–Kearns checkpointing and recovery protocol is based on the optimistic rollback. Vector time is used to capture causality to identify events and messages that become orphans when a failed process rolls back.
Helary–Mostefaoui–Netzer–Raynal (2000, 1997) communication-induced protocol	The Helary–Mostefaoui–Netzer–Raynal communication-induced checkpointing protocol prevents useless checkpoints and does it efficiently. To prevent useless checkpoints, some coordination is required in taking local checkpoints.

Now, few other algorithm for checkpointing and roll back recovery which are summarized over here juang and venkatesan algorithm for asynchronous checkpointing; so, we have seen the previous algorithm quasi asynchronous checkpointing; this is an asynchronous checkpointing and recovery and Manivannan and Singhal; they are quasi synchronous checkpointing algorithm and Peterson and Kearn's algorithm is based on the vector time and Helary and Mostefaoui; they is they have given the communication induced checkpointing algorithm.

(Refer Slide Time: 51:11)



Conclusion

- **Rollback recovery achieves fault tolerance** by periodically saving the state of a process during the failure-free execution, and restarting from a saved state on a failure to reduce the amount of lost computation.
- There are three basic approaches for checkpointing and failure recovery: (i) uncoordinated, (ii) coordinated, and (iii) communication induced checkpointing and for log based: (i) Pessimistic, (ii) Optimistic and (iii) Casual Logging
- Over the last two decades, checkpointing and failure recovery has been a very active area of research and several checkpointing and failure recovery algorithms have been proposed. In this lecture, we described **'Koo-Toueg Coordinated Checkpointing Algorithm'** and given an overview of other algorithms.
- In upcoming lecture, we will discuss about '**Deadlock Detection in Distributed Systems**'

Conclusion roll back recovery achieves the fault tolerance by periodically saving the state of a process during the failure free execution and restarting from the saved state on the failure to reduce the amount of the last computation and this we have seen in several algorithms happening. So, there are 3 basic approaches for checkpointing and failure recovery uncoordinated coordinated and communication induced checkpointing and for log based we have seen pessimistic optimistic and casual logging.

So, over last 2 decades, checkpointing and failure recovery algorithm has been very active area of research and several checkpointing and failure recovery algorithm have been proposed in designing the distributed applications. In this lecture, we have described Koo Toueg coordinated checkpointing algorithm and given the overview of other algorithm in upcoming lectures, we will discuss the deadlock detection in the distributed system.

Thank you.