Technical University of Cluj-Napoca
Computer Science Department

# Distributed Systems

**- Master in CS -**

# C9
# Rollback Recovery

**Fall 2017**



## Contents

# 1. Introduction

- Objective
  - Restore the system back to a consistent state after a failure
- How
  - Periodically saving the state of a process during the failure-free execution
  - It enables to restart from a saved state upon a failure to reduce the amount of lost work
- Checkpoint
  - The saved state is called a **checkpoint**
  - It stores the point in the execution of a process to which the process can later be restored
- Rollback recovery
  - The procedure of restarting from a previously checkpointed state is called **rollback recovery**



# 1. Introduction
## Problem Complexity

- Rollback recovery is complex due to inter-process dependencies during failure-free operation
- Inter-process dependencies are induced by the exchanged messages
- Rollback propagation
  - Upon a failure of one or more processes in a system, inter-process dependencies may force some of the processes that did not fail to roll back => a rollback propagation
  - This cascaded rollback is called the **domino effect**

# 1. Introduction
## Example of domino effect

- Process **p**, the sender of a message **m** rolls back to the state preceding sending of **m**
- Process **q**, the receiver of **m** must also roll back (domino effect) to a state that precedes **m**'s receipt
  - Otherwise, the states of the two processes would be inconsistent:
    - » They would show that message **m** was received without being sent (impossible in any correct failure-free execution)
- **Note**. In worst case, rollback propagation may extend back to the initial state of the computation, losing all the work performed before the failure



# 2. Types of rollback recovery

- Rollback recovery based on checkpointing
- Rollback recovery based on logs

# 2. Types of rollback recovery
## Checkpointing based rollback recovery

- Independent (un-coordinating) checkpointing
  - Each participating process takes its checkpoints independently
  - The system is susceptible to the domino effect
- Coordinated Checkpointing
  - Processes coordinate their checkpoints to form a system-wide consistent state
  - In case of a process failure
    - » system state can be restored to the consistent set of checkpoints **preventing the rollback propagation and the domino effect**
- Communication-induced checkpointing
  - Forces each process to take checkpoints based on information piggybacked on the application messages it receives from other processes



# 2. Types of rollback recovery
## Log-based rollback recovery

- Alternative to checkpointing rollback recovery
- Recommended for applications that frequently interact with the outside world, which consists of input and output devices that cannot roll back
- Combines checkpointing with logging of non-deterministic events
- Relies on the PieceWise Deterministic (PWD) assumption
  - All non-deterministic events that a process executes can be identified and
  - The information necessary to replay each event during recovery can be logged in a process related registry
- Objective
  - Deterministically recreate at process level, its pre-failure state by logging and replaying the non-deterministic events in their exact original order
- Log-based rollback recovery in general enables a system to recover beyond the most recent set of consistent checkpoints

# 3. Main concepts
## System Model

- DS
  - fixed number of processes, P1, P2, … PN
  - communicate only through messages
- Processes
  - cooperate to execute a distributed application and
  - interact with the outside world by receiving and sending input and output messages
- Figure 1 - a system consisting of three processes and interactions with the outside world

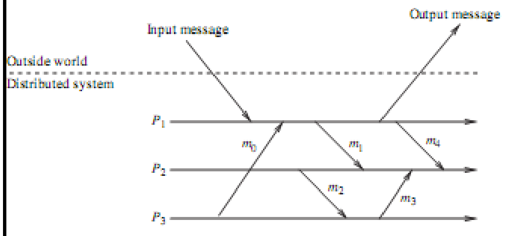



Figure 1 – Example of a DS with 3 processes (source [3])



# 3. Main concepts
## System Model (2)

- Rollback-recovery protocols generally make assumptions about the reliability of the inter-process communication
  - Some algorithms assume that the communication subsystem delivers messages reliably, in first-in-first-out (FIFO) order
  - Other algorithms assume that the communication subsystem can lose, duplicate, or reorder messages
  - The choice between these two assumptions affects the complexity of checkpointing and failure recovery
- Rollback-recovery algorithms must maintain
  - information about the internal interactions among processes and
  - the external interactions with the outside world

# 3. Main concepts
## Local checkpoint

- In a DS all processes save their states at certain instances of time - local checkpoints
- At any time, a process may keep several local checkpoints or just a single checkpoint (depending on the checkpointing method used)
- Assumptions
  - A process stores all local checkpoints on the stable storage
    => checkpoints are available even if the process crashes
  - A process is able to roll back to any of its existing local checkpoints and thus restore to and restart from the corresponding state
  - **Ci,k** denote the **k**-th local checkpoint at process **Pi**
  - A process **Pi** takes checkpoint **Ci,0** before it starts execution
  - A local checkpoint is shown in the process-line by the symbol " | "



# 3. Main concepts
## Consistent system states

- DS Global State (GS)
  - A collection of:
    - » the individual states of all participating processes and
    - » the states of the communication channels
- A consistent GS is one in which if a process state reflects a message receipt, then the state of the corresponding sender also reflects the sending of that message
- Figure 2 – two examples of global states
  - Consistent global state (Figure a)
    - » Message m1 was sent but not yet received (that is OK)
    - » The state is consistent - for every message that has been received, there is a corresponding message send event
  - Inconsistent global state (Figure b)
    - » Process P2 receives m2 but the state of process P1 does not reflect having sent it
    - » Such a state is impossible in any failure-free, correct computation
    - » **Inconsistent states occur because of failures**
    - » For instance, the situation shown in b) may occur if process P1 fails after sending message m2 to process P2 and then restarts at the state shown in b)

# 3. Main concepts
## Consistent system states

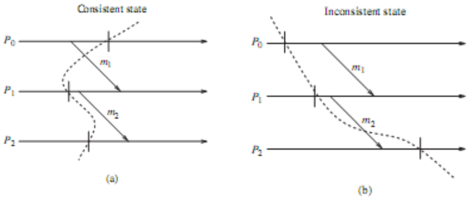



Figure 2 – Example of consistent and inconsistent states (source [3])

**Inconsistent states occur because of failures**
For instance, the situation shown in b) may occur if process P1 fails after sending message m2 to process P2 and then restarts at the state shown in b)



# 3. Main concepts
## Consistent system states

- Local checkpoint
  - A snapshot of a local state of a process
- Global checkpoint
  - A set of local checkpoints, one from each process
- An arbitrary set of local checkpoints at processes may not form a consistent global checkpoint
- When inconsistency (usually) happens?
  - When a message is sent by a process after taking its local checkpoint that is received by another process before taking its local checkpoint
  - Consistency of global checkpoints strongly depends on the flow of messages exchanged by processes
- Fundamental goal of any rollback-recovery protocol: **bring the system to a consistent state after a failure**
  - The reconstructed consistent state is not necessarily one that occurred before the failure
  - It is sufficient that the reconstructed state be one that could have occurred before the failure in a failure-free execution, provided that it is consistent with the interactions that the system had with the outside world

# 3. Main concepts
## Interactions with the outside world (1)

- A distributed application interacts with the outside world
  - to receive input data or
  - deliver the outcome of a computation
- If a failure occurs, the outside world cannot be expected to roll back
- Examples
  - A printer cannot roll back the effects of printing a character
  - An ATM cannot recover the money that it dispensed to a customer
- Outside world model in rollback-recovery protocols interactions with the outside world
  - The outside world is modeled as a **special process** OWP (Outside World Process)
  - OWP interacts with the rest of the system through message passing



# 3. Main concepts
## Interactions with the outside world (2)

- Input commit problem
  - Input messages received from the OWP may not be reproducible during recovery, because it may not be possible for the outside world to regenerate them

    => Thus, recovery protocols must arrange to save these input messages so that they can be retrieved when needed for execution replay after a failure
  - A common approach is to save each input message on the stable storage before allowing the application program to process it
  - This is commonly called **the input commit problem**
- Output commit problem
  - It is necessary that the outside world see a consistent behavior of the system despite failures

    => Before sending output to the OWP, the system must ensure that the state from which the output is sent will be recovered despite any future failure
  - This is commonly called **the output commit problem**
- An interaction with the outside world to deliver the outcome of a computation is shown on the process-line by the symbol “ || “

# 4. Types of messages

- A process failure and subsequent recovery may leave messages (that were perfectly received and processed before the failure) in abnormal states
  - This is because a rollback of processes for recovery may have to rollback the send and receive operations of several messages
- Objective – Identify types of messages involved in a process failure and subsequent recovery process



# 4. Types of messages

Case study - figure 3

Four processes

P1 fails at the indicated point

The whole system recovers to the state indicated by the recovery line, i.e. to a global state :

GS1 = { C1,8; C2,9; C3,8; C4,8 }

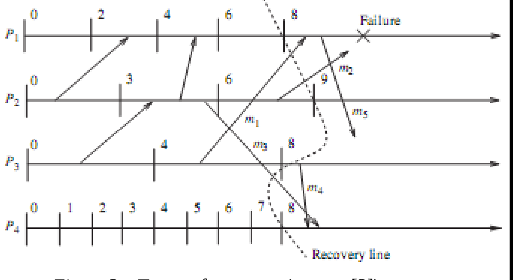
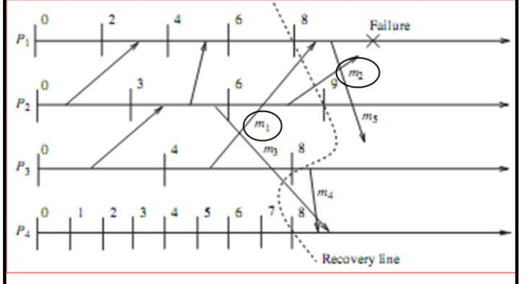



Figure 3 – Types of messages (source [3])

# 4. Types of messages

- **In transit messages**
- In GS1: m1 was sent but not yet received
  - m1 – in transit message
  - m2 – also in transit message
- When in-transit
  - Messages are part of a global system state and
  - Messages do not cause any inconsistency



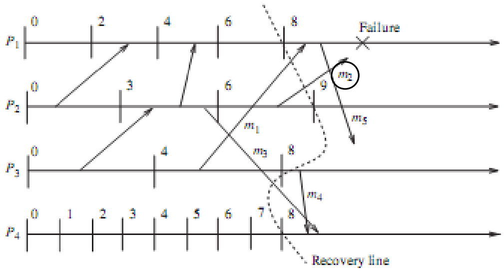



# 4. Types of messages

- **In transit messages (cont.)**
- If system model assumes **reliable communication** channels
  - Rollback-recovery protocols may have to guarantee the delivery of in-transit messages when failures occur
  - A consistent state must include in-transit messages because they will always be delivered to their destination in any legal execution of the system
- If system model assumes **loosely communication** channels
  - In-transit messages can be omitted from system state

# 4. Types of messages

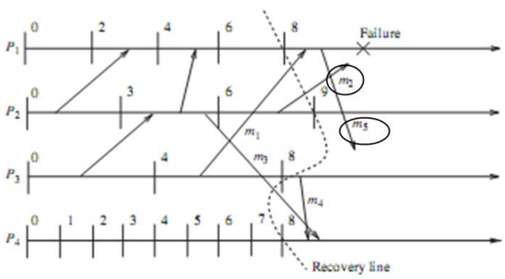
- **Lost messages**
- Messages whose send is not undone but receive is undone due to rollback
  - In Figure 3, **m2** is a lost message
- When they occur (example for lost message **m2**)?
  - When the process (**p1**) rolls back to a checkpoint (**8**) prior to reception of the message (while the sender (**p2**) does not rollback beyond the send operation of the message






# 4. Types of messages

- **Delayed messages**
- Messages whose receive is not recorded because
  - the receiving process was either down or
  - the message arrived after the rollback of the receiving process
  - Messages **m2** and **m5** in Figure 3 are delayed messages



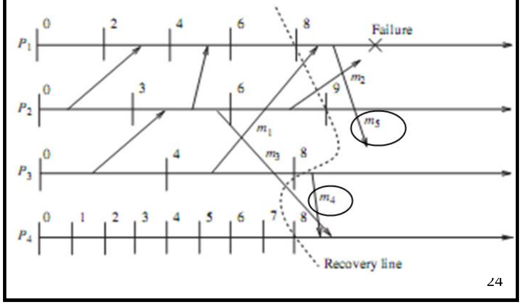
# 4. Types of messages

- **Orphan messages**
- Messages with receive recorded but message send not recorded are called orphan messages
- Example
  - A rollback might have undone the send of such messages, leaving the receive event intact at the receiving process
- Orphan messages do not arise if processes roll back to a consistent global state



# 4. Types of messages

- **Duplicate messages**
- Duplicate messages appear in the system due to message logging and that are replaying during process recovery
  - In Fig 3, **m4** was sent and received before the rollback
  - However, due to the rollback of process P4 to C4,8 and process P3 to C3,8, both send and receipt of **m4** are undone. When process P3 restarts from C3,8, it will resend **m4**
  - Therefore, P4 should not replay message **m4** from its log. If P4 replays message **m4**, then message **m4** is called a duplicate message
  - **m5** is also an example of a duplicate message. No matter what, the receiver of **m5** will receive a duplicate **m5** message



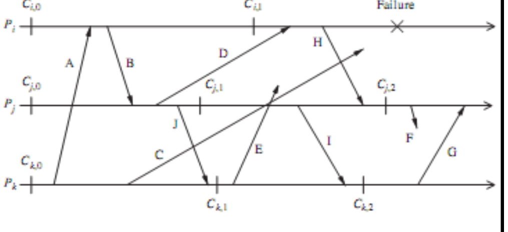
# 5. Issues in Failure Recovery

- In a failure recovery
  - The **system** should be restored to a consistent state and
  - **Messages** that are left in an abnormal state due to the failure and recovery should be appropriately handled
- Issues are described by using a computation shown in Figure 4
  - The computation comprises of three processes Pi, Pj, and Pk, connected through a communication network
  - The processes communicate solely by exchanging messages **over fault-free, FIFO communication channels**



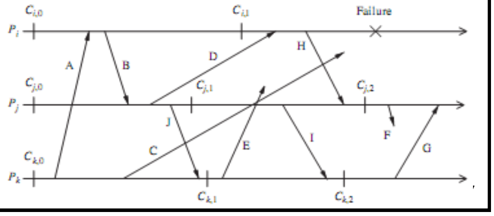
# 5. Issues in Failure Recovery

- Processes Pi, Pj, and Pk
  - Have taken check-points {Ci,0; Ci,1}, {Cj,0; Cj,1; Cj,2}, and {Ck,0; Ck,1; Ck,2}, respectively, and
  - have exchanged messages A to J as shown in Figure 4
- Suppose process Pi fails at the instance indicated in the figure
  - All contents of the volatile memory of Pi are lost and,
  - After Pi has recovered from the failure, the system needs to be restored to a consistent global state from where the processes can resume their execution



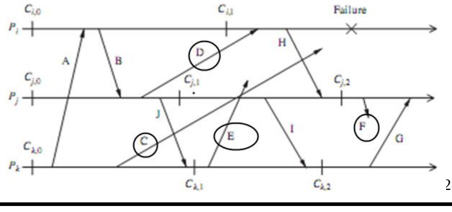
# 5. Issues in Failure Recovery

- Process Pi's state is restored to a valid state by rolling it back to its most recent checkpoint Ci,1
- To restore the system to a consistent state:
  - Process Pi rolls back to Pi,1
    - » This rollback creates an orphan message H
  - Process Pj does not roll back to checkpoint Cj,2 but to checkpoint Cj,1, because rolling back to checkpoint Cj,2 does not eliminate the orphan message H
  - Process Pk cannot be restored to Ck,2 due to the orphan message I created due to the roll back of process Pj to checkpoint Cj,1
- The restored global state **{Ci,1; Cj,1; Ck,1}** is a consistent state as it is free from orphan messages



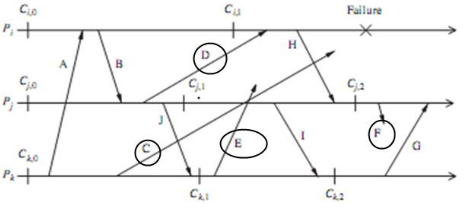

# 5. Issues in Failure Recovery

- Discussing message related problems
  - Although the system state has been restored to a **consistent state**, several messages are left in an erroneous state which must be handled correctly
- Messages A, B, D, G, H, I, and J had been received at the points indicated in the figure and messages C, E, and F were in transit when the failure occurred
- Restoration of system state to checkpoints {Ci,1, Cj,1, Ck,1} automatically handles (no consistency problems):
  - Messages A, B, and J - because the send and receive events of messages A, B, and J have been recorded
    - » Messages A, B, J are called **normal messages**
  - Messages G, H, I - because the send and receive events have been completely undone
    - » Messages G, H, I are called **vanished messages**
- Messages C, D, E, and F are potentially problematic



# 5. Issues in Failure Recovery

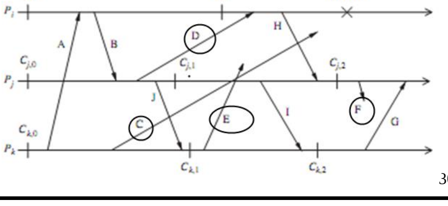
- **Message C** is in transit during the failure and it is a delayed message
- The delayed message C has several possibilities:
  - C might arrive at process Pi before it recovers,
  - C might arrive while Pi is recovering, or
  - C might arrive after Pi has completed recovery
- Each of these cases must be dealt with correctly





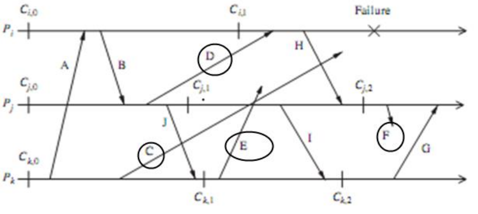

# 5. Issues in Failure Recovery

- **Message D** is a lost message
  - The send event for D is recorded in the restored state for process Pj , but the receive event has been undone at process Pi
  - Process Pj will not resend D without an additional mechanism, since the send D at Pj occurred before the checkpoint and message D was successfully delivered
  - Lost messages like D can be handled by having processes keep a message log of all the sent messages
    - » So when a process restores to a checkpoint, it replays the messages from its log to handle the lost message problem
    - » In this case **process Pi should replay the message D** from its log



## 5. Issues in Failure Recovery

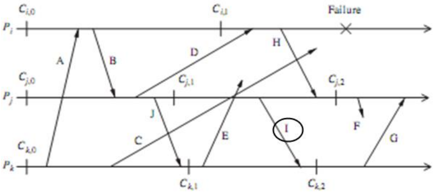
- **Messages E and F** are delayed orphan messages
  - Rise the most serious problem of all the messages
  - When messages E and F arrive at their respective destinations, they must be discarded since their send events have been undone
  - Processes, after resuming execution from their checkpoints, will generate both of these messages, and recovery techniques must be able to distinguish between messages like C and those like E and F






## 5. Issues in Failure Recovery

- **Duplicated messages**
  - Message logging and message replaying during recovery can result in duplicate messages
  - Duplicate messages must be handled properly
  - Example
    - » In Figure 4, when process $P_j$ replays messages from its log, it will regenerate message I
    - » Process $P_k$, which has already received message I, will receive it again, thereby causing inconsistency in the system state



# 5. Issues in Failure Recovery

- **Overlapping failures**
  - Further complicate the recovery process
  - A process Pj that begins rollback/recovery in response to the failure of a process Pi can itself fail and develop amnesia with respect process Pi's failure
  - That is, process Pj can act in a fashion that exhibits ignorance of process Pi 's failure
  - If overlapping failures are to be tolerated, a mechanism must be introduced to deal with amnesia and the resulting inconsistencies



# 6. Checkpoint-based Rollback Recovery

- In this method the state of each process and the communication channel is checkpointed frequently

  => upon a failure, the system can be restored to a globally consistent set of checkpoints

- It does not rely on the PWD assumption, and so does not need to detect, log, or replay non-deterministic events

  => Checkpoint-based protocols are therefore less restrictive and simpler to implement than log-based rollback recovery

  => Checkpoint-based rollback recovery may not be suitable for applications that require frequent interactions with the outside world

## 6. Checkpoint-based Rollback Recovery

- Checkpoint-based rollback-recovery techniques can be classified into three categories:
  - uncoordinated checkpointing, (UnCo_CkP)
    - » Each process has autonomy in deciding when to take checkpoints
  - coordinated checkpointing (Co_CkP), and
  - communication-induced checkpointing (Com_CkP)



## 6. Checkpoint-based Rollback Recovery
UnCoordinated Checkpointing (UnCo_CkP)

- **Advantages**
  - Eliminates the synchronization overhead
    - » no need for coordination between processes and
    - » allows processes to take checkpoints when it is most convenient or efficient
  - Lower runtime overhead during normal execution, because no coordination among processes is necessary
  - Autonomy in taking checkpoints also allows each process to select appropriate checkpoints positions

## 6. Checkpoint-based Rollback Recovery
### UnCoordinated Checkpointing (UnCo_CkP)

- **Shortcomings**
  - Possibility of the domino effect during a recovery
    - » may cause the loss of a large amount of useful work
  - Recovery from a failure is slow because processes need to iterate to find a consistent set of checkpoints
  - Since no coordination is done at the time the checkpoint is taken, checkpoints taken by a process may be **useless checkpoints**
    - » A useless checkpoint is never a part of any global consistent state.
    - » Useless checkpoints are undesirable because they incur overhead and do not contribute to advancing the recovery line
  - UnCo_CkP forces each process
    - » to maintain multiple checkpoints, and
    - » to periodically invoke a garbage collection algorithm to reclaim the checkpoints that are no longer required
  - Not suitable for applications with frequent output commits because these require global coordination to compute the recovery line, negating much of the advantage of autonomy



## 6. Checkpoint-based Rollback Recovery
### UnCoordinated Checkpointing (UnCo_CkP)

- **Direct dependency tracking technique**
  - Each process takes checkpoints independently
  - ⇒ A consistent global checkpoint to rollback to, when a failure occurs needs to be determined
- To determine a consistent global checkpoint during recovery, the processes record the dependencies among their checkpoints caused by message exchange during failure-free operation

## 6. Checkpoint-based Rollback Recovery
### UnCoordinated Checkpointing (UnCo_CkP)

- Notations
  - **Ci,x** - the x-th checkpoint of process Pi, where
    - **i** is the process id and
    - **x** is the checkpoint index
    - each process Pi starts its execution with an initial checkpoint Ci,0
  - **Ii,x** - the checkpoint interval or simply interval between checkpoints Ci,x−1 and Ci,x
- Direct dependency tracking technique (commonly used in UnCo_CkP - see example of Figure 5):

- When process **Pi** at interval **Ii,x** sends a message **m** to **Pj**, it piggybacks the pair **(i, x)** on **m**
- When **Pj** receives **m** during interval **Ij,y**, it records the
- dependency from **Ii,x** to **Ij,y**, which is later saved onto
- stable storage when **Pj** takes checkpoint **C j,y**

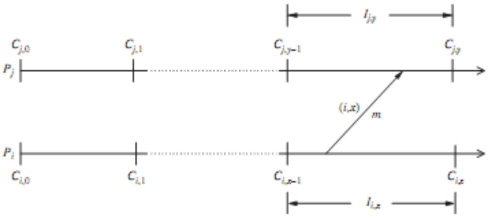


Figure 5 – Checkpoint index and interval (source [3])



## 6. Checkpoint-based Rollback Recovery
### UnCoordinated Checkpointing (UnCo_CkP)
### Protocol execution when failure occurs

*when a failure occurs:*

- the recovering process initiates rollback by broadcasting a dependency request message to collect all the dependency information maintained by each process

*when a process receives dependency request message:*

- the process stops its execution and
- replies with the dependency information saved on the stable storage as well as with the dependency information, if any, which is associated with its current state

*when the recovering process receives all replies:*

- the initiator calculates the recovery line based on the global dependency information and
- broadcasts a rollback request message containing the recovery line

*when a process received the recovery line message:*

- a process whose current state belongs to the recovery line simply resumes execution
- otherwise, it rolls back to an earlier checkpoint as indicated by the recovery line

## 6. Checkpoint-based Rollback Recovery
### Coordinated Checkpointing (Co_CkP)

- In Co_CkP, processes orchestrate their CkP activities so that all local checkpoints form a consistent global state
- Advantages
  - Simplifies recovery
  - Eliminates the domino effect, since every process always restarts from its most recent checkpoint
  - Requires each process to maintain only one checkpoint on the stable storage
    - » Decreases storage overhead and
    - » Eliminates the need for garbage collection
- Disadvantages
  - Large latency is involved in committing output, as a global checkpoint is needed before a message is sent to the OWP.
  - Delays and overhead every time a new global checkpoint is taken



## 6. Checkpoint-based Rollback Recovery
### Coordinated Checkpointing (Co_CkP)

- Simple (but unrealistic) Co_CkP method
  - Assumption: Perfectly synchronized clocks available at processes
  - All processes agree at what instants of time they will take checkpoints
  - The clocks at processes trigger the local checkpointing actions at all processes
- Realistic Co_CkP methods to guarantee checkpoint consistency
  - **Blocking Co_CkP**: Either the sending of messages is blocked for the duration of the protocol, or
  - **NonBlocking Co_CkP:** Checkpoint indices are piggybacked to avoid blocking

## 6. Checkpoint-based Rollback Recovery
### Coordinated Checkpointing (Co_CkP)
### Blocking Co_CkP

- The straightforward approach to Co_CkP
  - Block communications while the checkpointing protocol executes
- To prevent orphan messages
  - After a process takes a local checkpoint
  - The process remains blocked until the entire checkpointing activity is complete
- Main problem
  - Computation processes are blocked during the checkpointing

  => Non-blocking checkpointing schemes are preferable



## 6. Checkpoint-based Rollback Recovery
### Coordinated Checkpointing (Co_CkP)
### Blocking Co_CkP – Algorithm Specification

*at Coordinator:*

- The coordinator takes a checkpoint and
- Broadcasts a request message to all processes, asking them to take a checkpoint

*at Process receiving checkpoint taking message:*

- Process stops its execution,
- Flushes all the communication channels
- Takes a tentative checkpoint, and sends an acknowledgment message back to the coordinator

*at Coordinator after receiving acknowledge from all Process:*

- Broadcasts a commit message that completes the two-phase checkpointing protocol

*at Process receiving the commit message:*

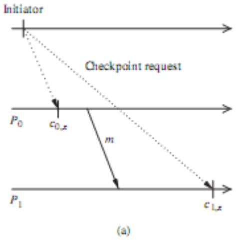
- Process removes the old permanent checkpoint and
- Atomically makes the tentative checkpoint permanent and
- Resumes its execution and exchange of messages with other processes

## 6. Checkpoint-based Rollback Recovery
Coordinated Checkpointing (Co_CkP)
Non-blocking checkpoint coordination

- In this method the processes need not stop their execution while taking checkpoints
- The fundamental problem in Co_CkP
  - Prevent a process from receiving application messages that could make the checkpoint inconsistent
- Consider the example in Figure 6(a):
  - message **m** is sent by P0 after receiving a checkpoint request from the checkpoint coordinator
  - Assume **m** reaches P1 before the checkpoint request
  - This situation results in an inconsistent checkpoint:
    - » checkpoint c1,x shows the receipt of message m from P0, while
    - » checkpoint c0,x does not show m being sent from P0
  - If channels are FIFO, this problem can be avoided (Figure 6(b)):
    - » The first post-checkpoint message on each channel is preceded by a checkpoint request
    - » It forces each process to take a checkpoint before receiving the first post-checkpoint message



## 6. Checkpoint-based Rollback Recovery
Coordinated Checkpointing (Co_CkP)
Non-blocking checkpoint coordination




(a)

Figure 6 – Non-blocking coordinated checkpoint. a) – checkpoint inconsistency;

**Figure 6(a)**
- Message **m** is sent by P0 after receiving a checkpoint request from the checkpoint coordinator. Assume **m** reaches P1 before the checkpoint request => inconsistent checkpoint
  - checkpoint c1,x shows the receipt of message m from P0, while
  - checkpoint c0,x does not show m being sent from P0

## 6. Checkpoint-based Rollback Recovery Coordinated Checkpointing (Co_CkP) Non-blocking checkpoint coordination

**Case of FIFO Channels**

- Example of a non-blocking checkpoint coordination protocol using this idea is the snapshot algorithm of Chandy & Lamport for reliable FIFO channels
  - Markers play the role of the checkpoint request messages
  - The initiator takes a checkpoint and sends a marker (a checkpoint request) on all outgoing channels
  - Each process takes a checkpoint upon receiving the first marker and sends the marker on all outgoing channels before sending any application message
  - The protocol works assuming the channels are reliable and FIFO



## 6. Checkpoint-based Rollback Recovery Coordinated Checkpointing (Co_CkP) Non-blocking checkpoint coordination

- If the channels are non-FIFO, the following two approaches can be used:
  - The marker can be piggybacked on every post-checkpoint message
    - » When a process receives an application message with a marker, it treats it as if it has received a marker message, followed by the application message.
  - Checkpoint indices can serve the same role as markers, where a checkpoint is triggered when the receiver's local checkpoint index is lower than the piggybacked checkpoint index

## Checkpoint-based Rollback Recovery Coordinated Checkpointing (Co_CkP)
### Non-blocking checkpoint coordination

- Co_CkP requires all processes to participate in every checkpoint
  - This requirement generates concerns about its scalability
- It is desirable to reduce the number of processes involved in a Co_CkP session
- This can be achieved since only those processes that have communicated with the checkpoint initiator either directly or indirectly since the last checkpoint need to take new checkpoints
- A two-phase protocol by Koo and Toueg achieves such a minimal checkpoint coordination



## 6. Checkpoint-based Rollback Recovery
### Comunication-induced Checkpointing (Com_CkP)

- Features
  - Avoids domino effect
  - Processes may be forced to take additional checkpoints (over and above their autonomous checkpoints) in order to guarantee the eventual progress of the recovery line
  - Reduces or completely eliminates the useless checkpoints
- In Com_CkP, processes take two types of checkpoints
  - autonomous or independent checkpoints (also called local checkpoints)
  - forced checkpoints – the processes are forced to take these checkpoints

## 6. Checkpoint-based Rollback Recovery
### Comunication-induced Checkpointing (Com_CkP)

- There are two types of Com_CkP:
  - model-based checkpointing
    - » the system maintains checkpoints and communication structures that prevent the domino effect or achieve some even stronger properties
  - 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



## 7. Log-based Rollback Recovery (Log_RR)

- A log-based rollback recovery makes use of
  - deterministic and
  - non-deterministic events in a computation
- Log_RR uses the fact that a process execution can be modeled as a sequence of deterministic state intervals, each starting with the execution of a non-deterministic event
- A non-deterministic event can be
  - the receipt of a message from another process or
  - an event internal to the process
  - **Note.** A message send event is not a non-deterministic event

# 7. Log-based Rollback Recovery
## Deterministic and non-deterministic events

- Log_RR 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
  - Additionally, each process also takes checkpoints to reduce the extent of rollback during recovery
- After a failure
  - The failed processes replays the corresponding non-deterministic events, precisely as they occurred during the pre-failure execution using
    - » Checkpoints and
    - » Logged determinants
- Because execution within each deterministic interval depends only on the sequence of non-deterministic events that preceded the interval's beginning, the pre-failure execution of a failed process can be reconstructed during recovery up to the first non-deterministic event whose determinant is not logged



# 7. Log-based Rollback Recovery
## Deterministic and non-deterministic events

- Example (Figure 7)
  - The execution of process P0 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 m0, m3, and m7, respectively
  - Send event of message m2 is uniquely determined by the initial state of P0 and by the receipt of message m0, and is therefore not a non-deterministic event

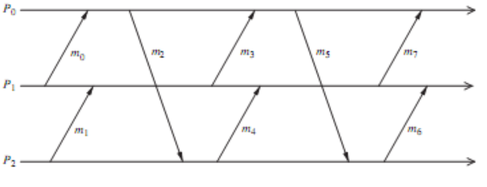



Figure 7 – Deterministic and non-deterministic events (source [3])