Log- based Rollback Recovery

Y. V. Lokeswari

Reference: Kshemkalyani, Ajay D., and Mukesh Singhal. Distributed computing: principles, algorithms, and systems. Cambridge University Press, 2011.

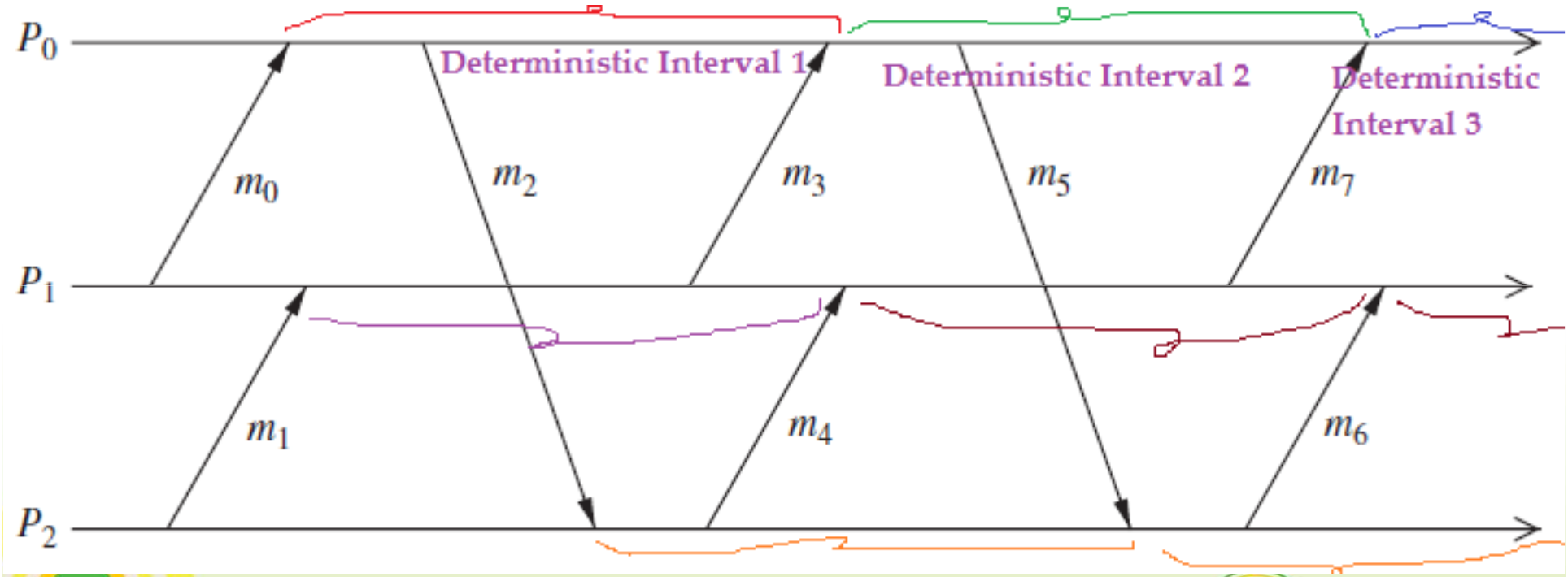
Overview

- Log-based Recovery.
- Deterministic and Non-Deterministic Events.
- No-Orphans consistency condition.
- Types of Logging Protocols.
  - Pessimistic logging.
  - Optimistic logging.
  - Causal logging.

Log-based Rollback Recovery

- Log-based rollback recovery exploits the fact that a process execution can be modelled as a **sequence of deterministic state intervals, each starting with the execution of a non-deterministic event**.
- **Send event is Deterministic**
- **Receive and internal events are non-deterministic.**
- **Non-determinism** means that the path of execution isn't fully determined by the specification of the computation, so the same input can produce different outcomes, while **deterministic** execution is guaranteed to be the same, given the same input.

Deterministic State Intervals



Log-based Rollback Recovery

- 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.
- Additionally, each process **also takes checkpoints** to reduce the extent of rollback during recovery.
- After a failure occurs, the failed processes **recover** by using the **checkpoints and logged determinants to replay the corresponding non-deterministic events** precisely as they occurred during the pre-failure execution.
- 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.**

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

$$\forall(e) : \neg Stable(e) \Rightarrow Depend(e) \subseteq Log(e)$$

No-Orphans consistency condition

Always-no-orphans condition

$$\forall(e) : \neg Stable(e) \Rightarrow Depend(e) \subseteq Log(e)$$

- **This property is called the *always-no-orphans condition.***

  It states that if any surviving process depends on an event *e, then either event e is logged on* the stable storage, or the process has a copy of the determinant of event *e.*

- *If* neither condition is true, then the process is an orphan because it depends on an event *e that cannot be generated during recovery since its determinant is* lost.

Log-based rollback-recovery

- Log-based rollback-recovery protocols guarantee that **upon recovery** of all failed processes, the **system does not contain any orphan process.**

Types of Log-based rollback-recovery

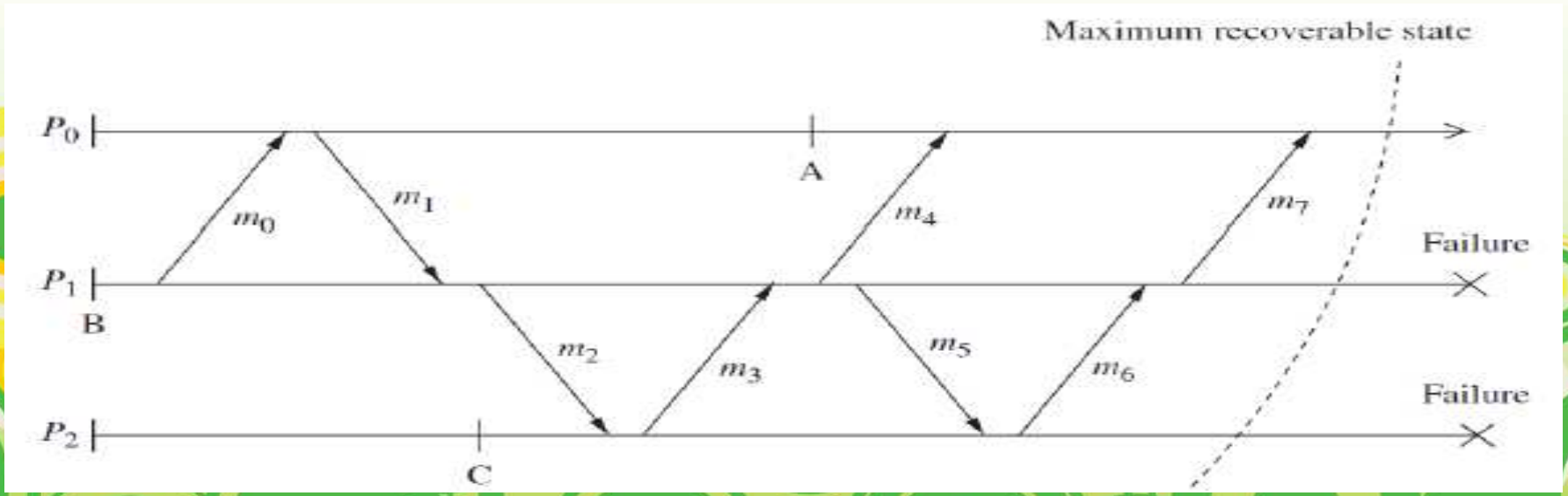
- **Pessimistic logging.**
- **Optimistic logging.**
- **Causal logging**.
- They differ in their failure-free performance overhead, latency of output commit, simplicity of recovery and garbage collection, and the potential for rolling back surviving processes.

Pessimistic Logging

- Pessimistic logging protocols assume that a failure can occur after any non-deterministic event in the computation.
- Pessimistic protocols log to the stable storage the determinant of each non-deterministic event before the event affects the computation.
- **This protocol implements *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.
- In addition to logging determinants, processes also take periodic checkpoints to minimize the amount of work that has to be repeated during recovery.
- When a process fails, the process is restarted from the most recent checkpoint and the logged determinants are used to recreate the pre-failure execution.

Pessimistic Logging

- During failure-free operation the logs of processes ***P0, P1, and P2 contain the determinants*** *needed* to **replay messages *m0, m4, m7, m1, m3, m6, and m2, m5***, *respectively.*
- *Suppose* processes *P1 and P2 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.
- This guarantees that *P1 and P2* will repeat exactly their pre-failure execution and re-send the same messages.



Pessimistic Logging

- Synchronous logging can potentially result in a **high performance overhead.**
- Special techniques to reduce the effects of synchronous logging on the performance. This overhead can be lowered using special hardware.
- **To Overcome Performance Overhead:**
- Fast non-volatile semiconductor memory can be used to implement the stable storage.
- Limit the number of failures that can be tolerated.
- Delivering a message or executing an event and deferring its logging until the process communicates with another process or with the outside world.
- ***Sender-Based Message Logging (SBML)*** *protocol keeps the determinants corresponding to the delivery* of each message *m in the volatile memory of its sender.*

Optimistic Logging

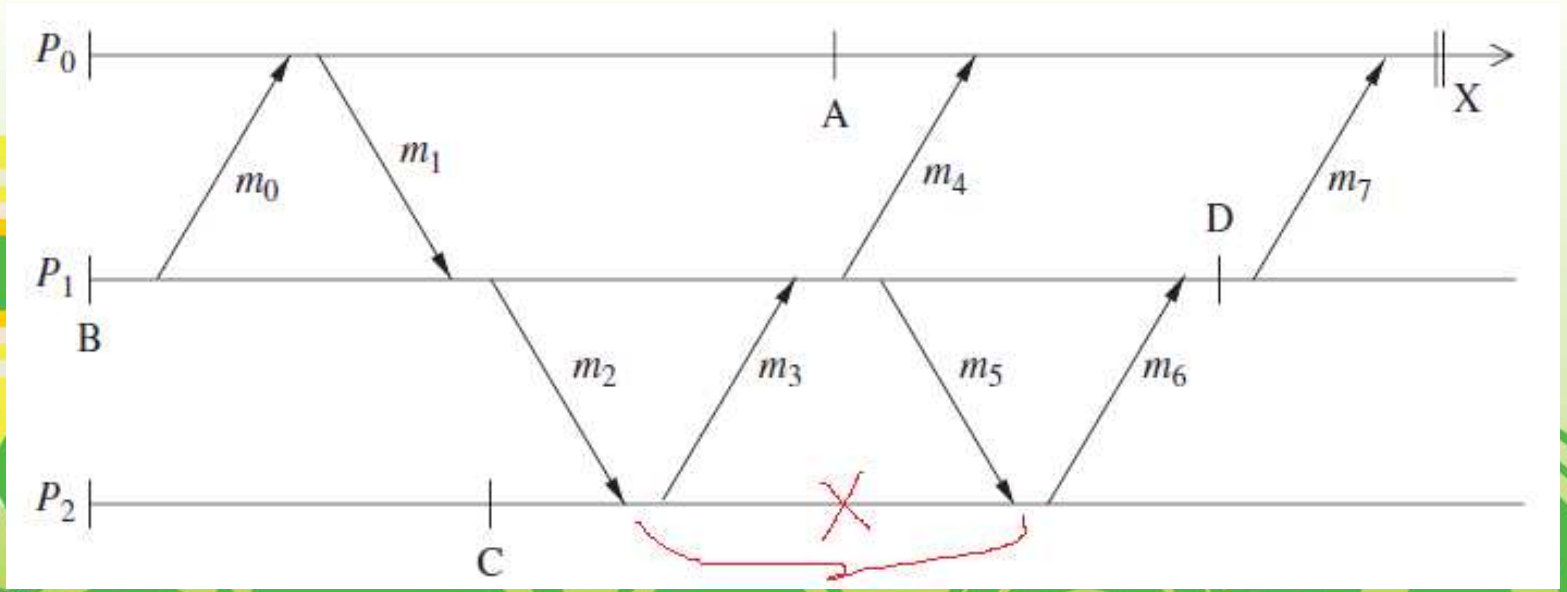
- Processes log determinants *asynchronously* to the stable storage.
- These protocols optimistically assume that logging will be complete before a failure occurs.
- Determinants are kept in a volatile log, and are periodically flushed to the stable storage.
- **Incurs much less overhead during failure-free execution.**
- **But more complicated recovery, garbage collection, and slower output commit.**

Optimistic Logging

- **If a process fails, the determinants in its volatile log are lost**, and the state intervals that were started by the non-deterministic events corresponding to these determinants **cannot be recovered.**
- **Optimistic logging protocols do not implement the *always-no-orphans condition.***
- The ***always-no-orphans condition*** *holds after the* recovery is complete.
- This is achieved by rolling back orphan processes until their states do not depend on any message whose determinant has been lost.

Optimistic Logging

- Suppose process *P2 fails* before the determinant for *m5 is logged to the stable storage.*
- *Process P1 then* becomes an orphan process and must roll back to undo the effects of receiving the orphan message *m6.*
- *The rollback of P1 further forces P0 to roll back to* undo the effects of receiving message *m7.*
- For example, if process *P0 needs to commit output at state X, it must* log messages *m4 and m7 to the stable storage and ask P2 to log m2 and m5.*



Optimistic Logging

- To perform rollbacks correctly, **optimistic logging protocols** track **causal dependencies** during failure free execution.
- Upon a failure, the **dependency information** is used to **calculate and recover the latest global state** of the pre-failure execution in which **no process is in an orphan**.
- **Pessimistic protocols** need **only keep the most recent checkpoint** of each process.
- **Optimistic protocols** may need to **keep multiple checkpoints** for each process.

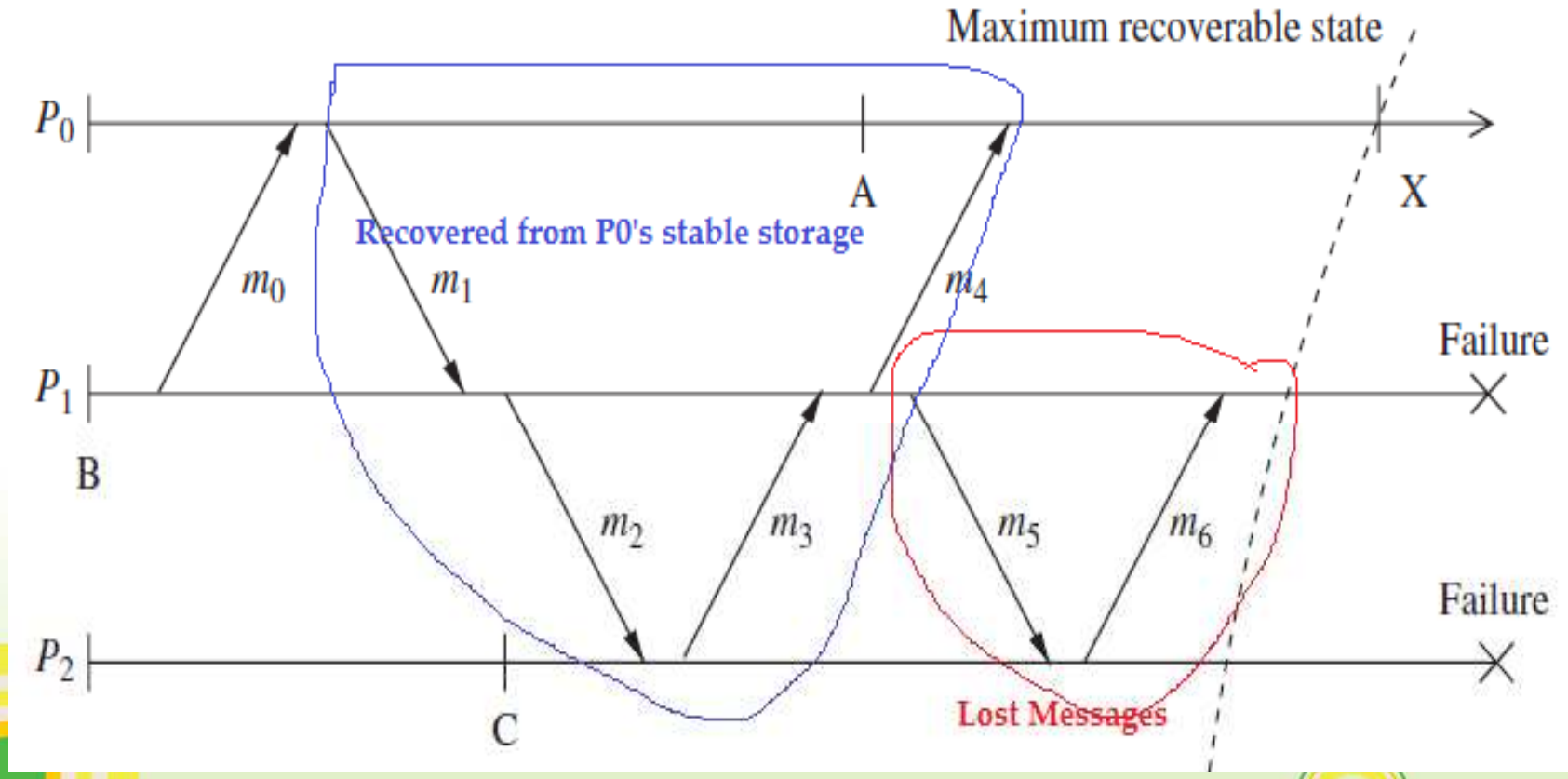
Causal Logging

- 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.**
- Moreover, **causal logging limits the rollback** of any failed process to the **most recent checkpoint on the stable storage**, thus minimizing the storage overhead and the amount of lost work.
- **Causal logging protocols make sure that the *always-no-orphans property* holds.**

Causal Logging

- Each process **maintains information** about **all** the **events** that have **causally affected** its state.
- This information **protects** it from the **failures** of other processes and also **allows** the **process** to make its **state recoverable** by simply logging the information available locally.
- It can **commit output independently**.

Causal Logging



Causal Logging

- Messages m5 and m6 are likely to be lost on the failures of P1 and P2 at the indicated instants.
- Process P0 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 m0, m1, m2, m3, and m4. 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 P0.
- The determinant of each of these events contains the order in which its original receiver delivered the corresponding message.
- The message sender, as in sender-based message logging, logs the message content.
- Process P0 will be able to "guide" the recovery of P1 and P2 since it knows the order in which P1 should replay messages m1 and m3 to reach the state from which P1 sent message m4.
- Similarly, P0 has the order in which P2 should replay message m2 to be consistent with both P0 and P1. The content of these messages is obtained from the sender log of P0 or regenerated deterministically during the recovery of P1 and P2. Note that information about messages m5 and m6 is lost due to failures.

Summary

- Log-based Recovery.
- Deterministic and Non-Deterministic Events.
- No-Orphans consistency condition.
- Types of Logging Protocols.
  - Pessimistic logging.
  - Optimistic logging.
  - Causal logging.