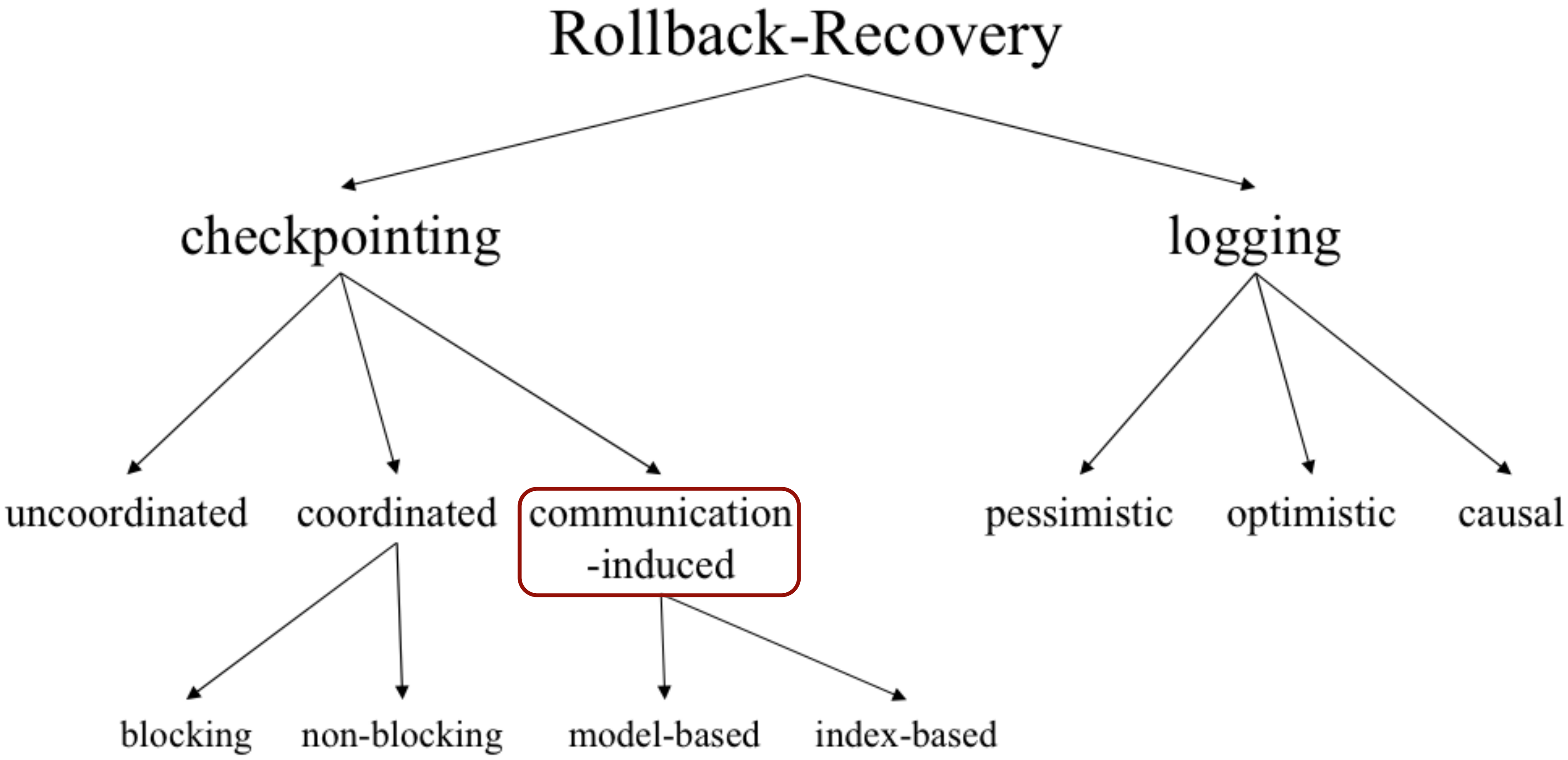
Rollback-Recovery Protocols II

Mahmoud ElGammal

Taxonomy



Rollback-Recovery

checkpointing

logging

uncoordinated

coordinated

communication-induced

pessimistic

optimistic

causal

blocking

non-blocking

model-based

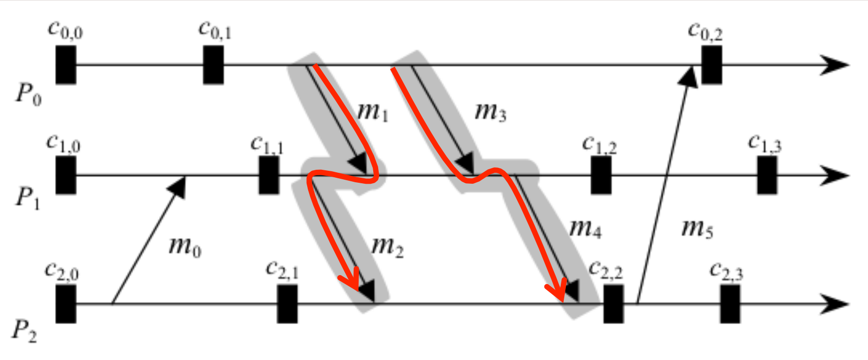
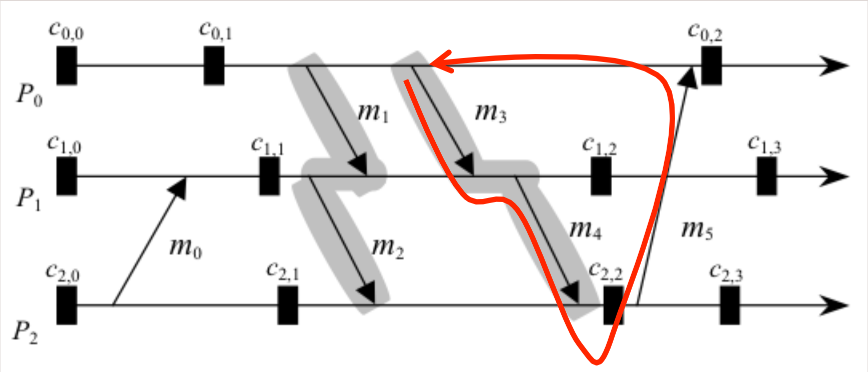
index-based

Communication-Induced Checkpointing

- Avoid the domino effect without requiring all checkpoints to be coordinated.
- Processes take two kinds of checkpoints: *local* and *forced*.
- Local checkpoints can be taken independently.
- Forced checkpoints must be taken to guarantee the eventual progress of the recovery line.
- No special coordination messages are exchanged to determine when forced checkpoints should be taken.
- Protocol-specific information is piggybacked on each application message.
- The receiver uses this information to decide if it should take a forced checkpoint.

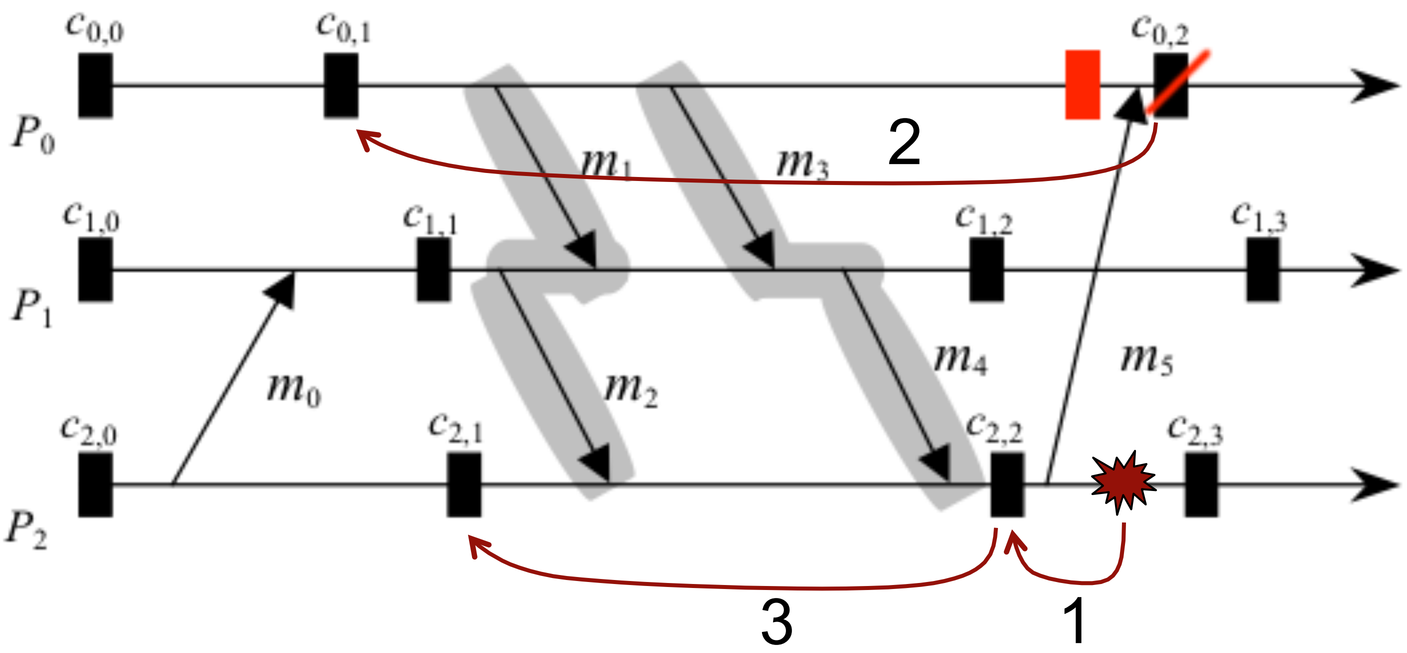
Communication-Induced Checkpointing Notation

How does a receiver decide when to take a forced checkpoint?

Z-path	Z-cycle
	

- A checkpoint is useless if and only if it is part of a Z-cycle.
- The receiver should determine if past communication and checkpoint patterns can lead to the creation of useless checkpoints.

Communication-Induced Checkpointing Notation

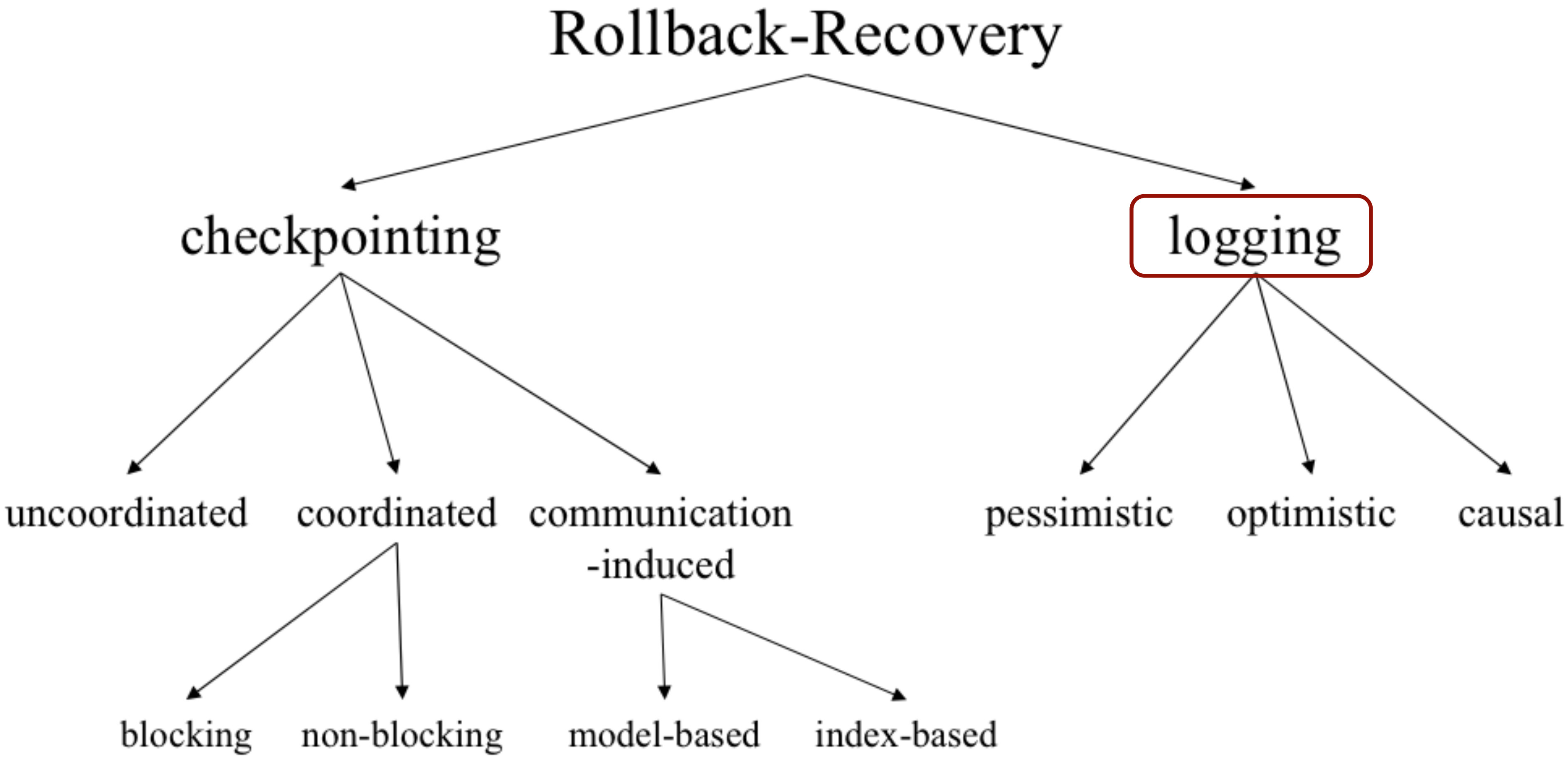


Checkpoint $c_{2,2}$ is useless under any failure scenario. P_0 must create a forced checkpoint before delivering m5 to break the m_3-m_4-m_5 Z-cycle.

Communication-Induced Checkpointing

- CIC protocols have been classified in two types:
  - ☐ ***Model-based Protocols*: Take more forced checkpoints than is probably necessary, because without explicit coordination, no process has complete information about the global system state.**
  - ☐ ***Index-based protocols*: Guarantee that checkpoints having the same index at different processes form a consistent state.**

Taxonomy



Rollback-Recovery

checkpointing

logging

uncoordinated

coordinated

communication -induced

pessimistic

optimistic

causal

blocking

non-blocking

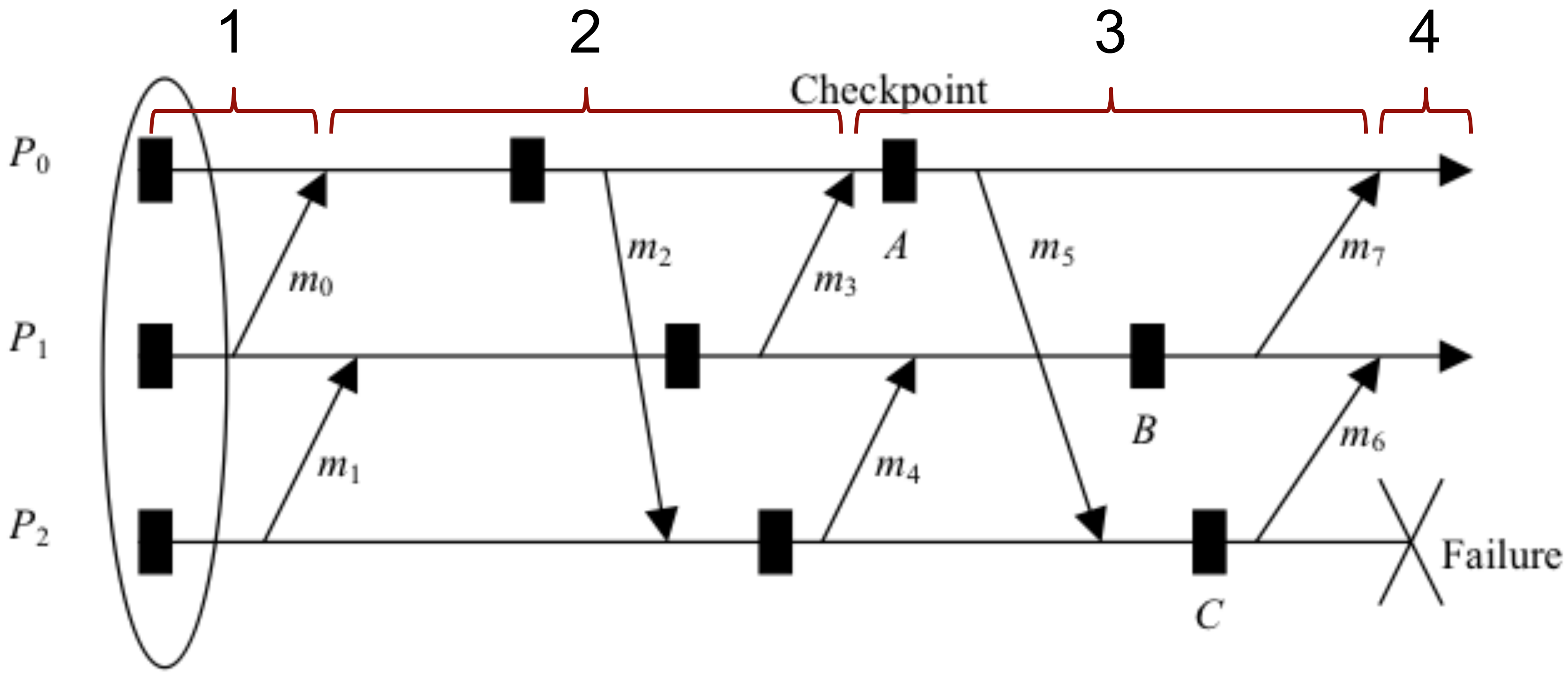
model-based

index-based

Log-Based Rollback Recovery

- Process execution is modeled as a sequence of deterministic state intervals, each starting with the execution of a nondeterministic event.
- *Non-deterministic event*: the receipt of a message or an internal event (something that affects the process).
- *Deterministic event*: sending a message (an effect caused by the process).

Log-Based Rollback Recovery



Log-Based Rollback Recovery

- All non-deterministic events can be identified and their determinants are logged to stable storage.
  - ☐ ***Determinant*: the information need to "replay" the occurrence of a non-deterministic event.**
- During failure-free operation, each process logs the determinants of all the non-deterministic events it observes onto stable storage.
- 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 nondeterministic events precisely as they occurred during the pre-failure execution.

Log-Based Rollback Recovery

- The pre-failure execution of a failed process can be reconstructed during recovery up to the first nondeterministic event whose determinant is not logged.
- Upon recovery of all failed processes, the system does not contain any *orphan process*: a process whose state depends on a nondeterministic event that cannot be reproduced during recovery:

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

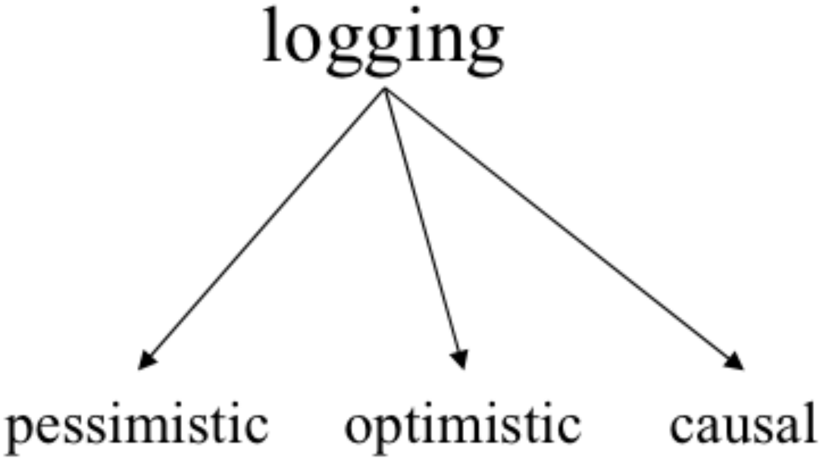
(The No-Orphans Consistency Condition)

- A process *p* becomes an orphan when *p* itself doesn't fail and *p*'s state depends on the execution of a nondeterministic event *e* whose determinant cannot be recovered from stable storage or from the volatile memory of a surviving process.

Log-Based Rollback Recovery

Key parameters:

- Failure-free performance overhead.
- Output-commit latency.
- Simplicity of recovery and garbage collection.
- Potential for rolling back correct processes.

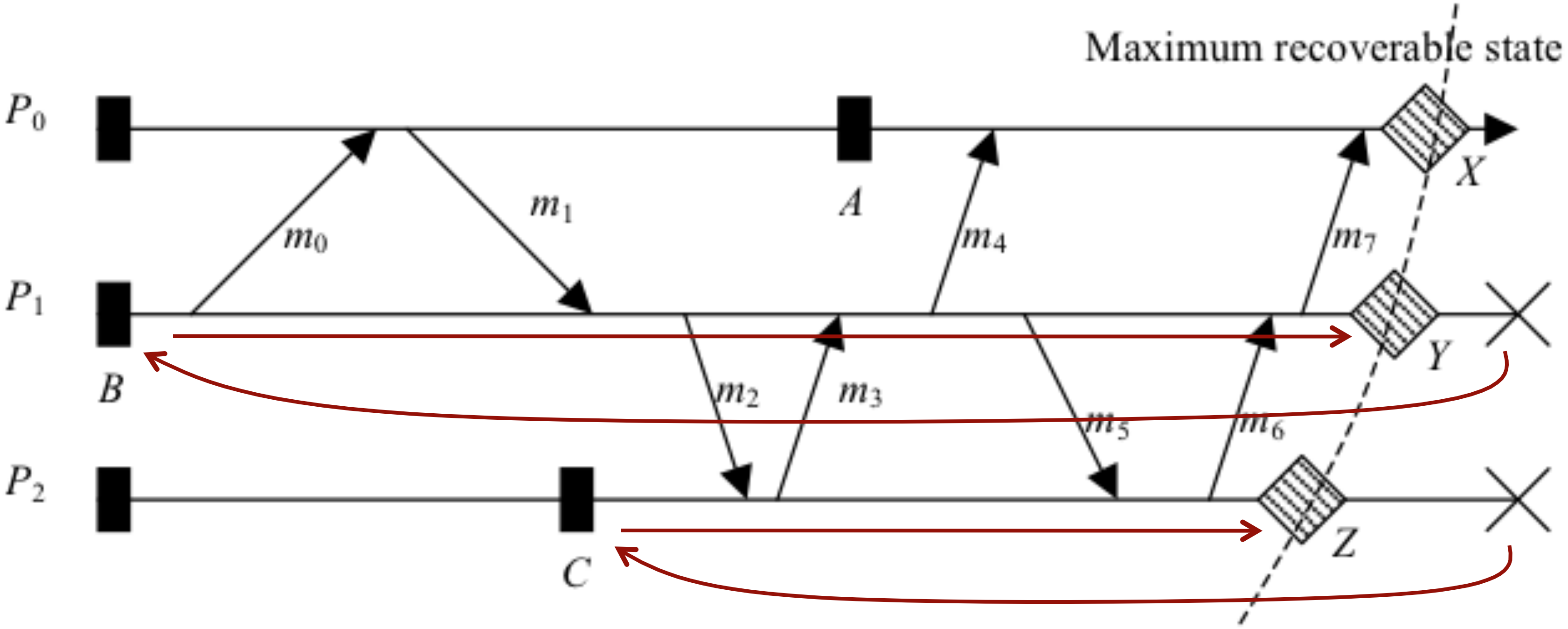


Log-Based Rollback Recovery / Pessimistic Logging

- Assumes that a failure can occur after *any* nondeterministic.
- The determinant of each nondeterministic event is logged to stable storage before the event is allowed to affect the computation.
- Employs *synchronous* logging (a strengthening of the *always-no-orphans* condition):

$$\forall e : \neg Stable(e) \Rightarrow \left|Depend(e)\right| = 0$$

Log-Based Rollback Recovery / Pessimistic Logging



Log-Based Rollback Recovery / Pessimistic Logging

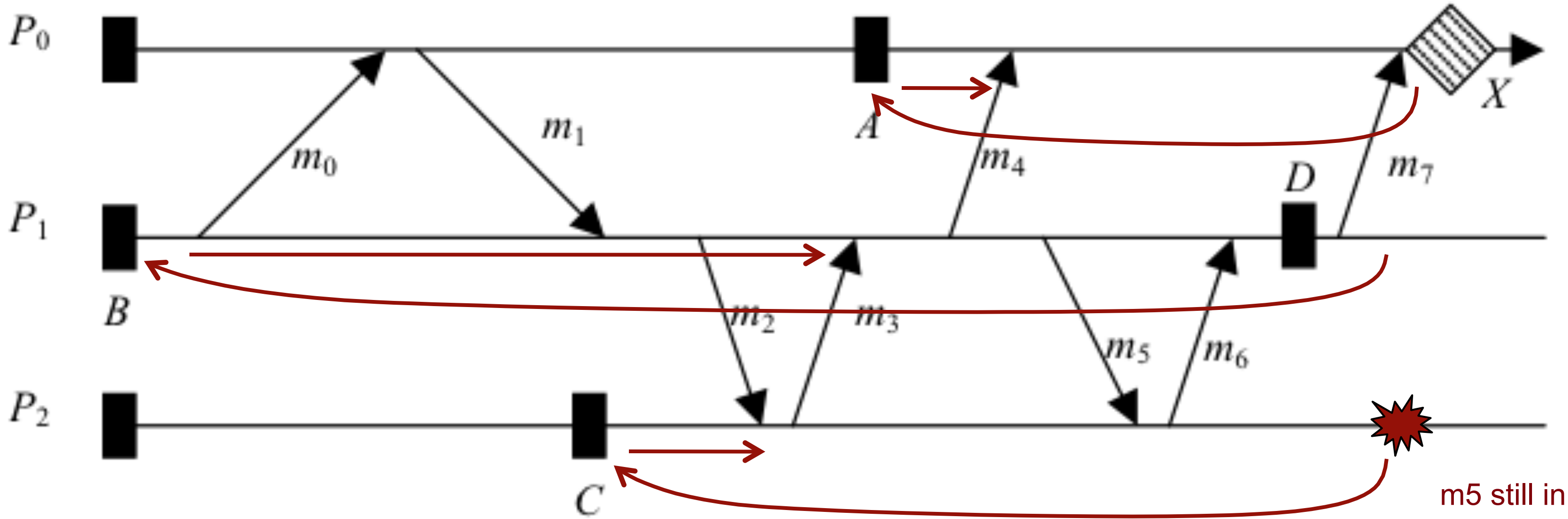
- Advantages:
  - **Processes can send messages to the outside world without running a special protocol.**
  - **Processes restart from their most recent checkpoint, limiting the extent of execution that has to be replayed.**
  - **Recovery is simplified because the effects of a failure are confined only to the processes that fail.**
  - **Garbage collection is simple.**
- Disadvantages:
  - **Synchronous logging incurs a high performance penalty during failure-free operation.**

Log-Based Rollback Recovery / Optimistic Logging

- Determinants of non-deterministic events are logged *asynchronously*: determinants are kept in a volatile log which is periodically flushed to stable storage.
- Assumes that logging will complete before a failure occurs.
- Allows the temporary creation of orphan processes, but none should exist by the time recovery is complete.

Log-Based Rollback Recovery / Optimistic Logging

- If a process fails, the determinants in its volatile log will be lost, and the state intervals that were started by such events cannot be recovered.
- If the failed process sent a message during any of these state intervals, the receiver of such message becomes an orphan process and must rollback to undo the effects of receiving the message.
- To perform these rollbacks correctly, causal dependencies must be tracked.



m5 still in volatile storage

Log-Based Rollback Recovery / Optimistic Logging

- Advantages:
  - **Incurs little overhead during failure-free execution.**
- Disadvantages:
  - **More complicated recovery and garbage collection than pessimistic logging:**
    - Must track causal dependencies.
    - May need to keep multiple checkpoints.
    - Output commit requires multi-host coordination to ensure that no failure scenario can revoke the output.

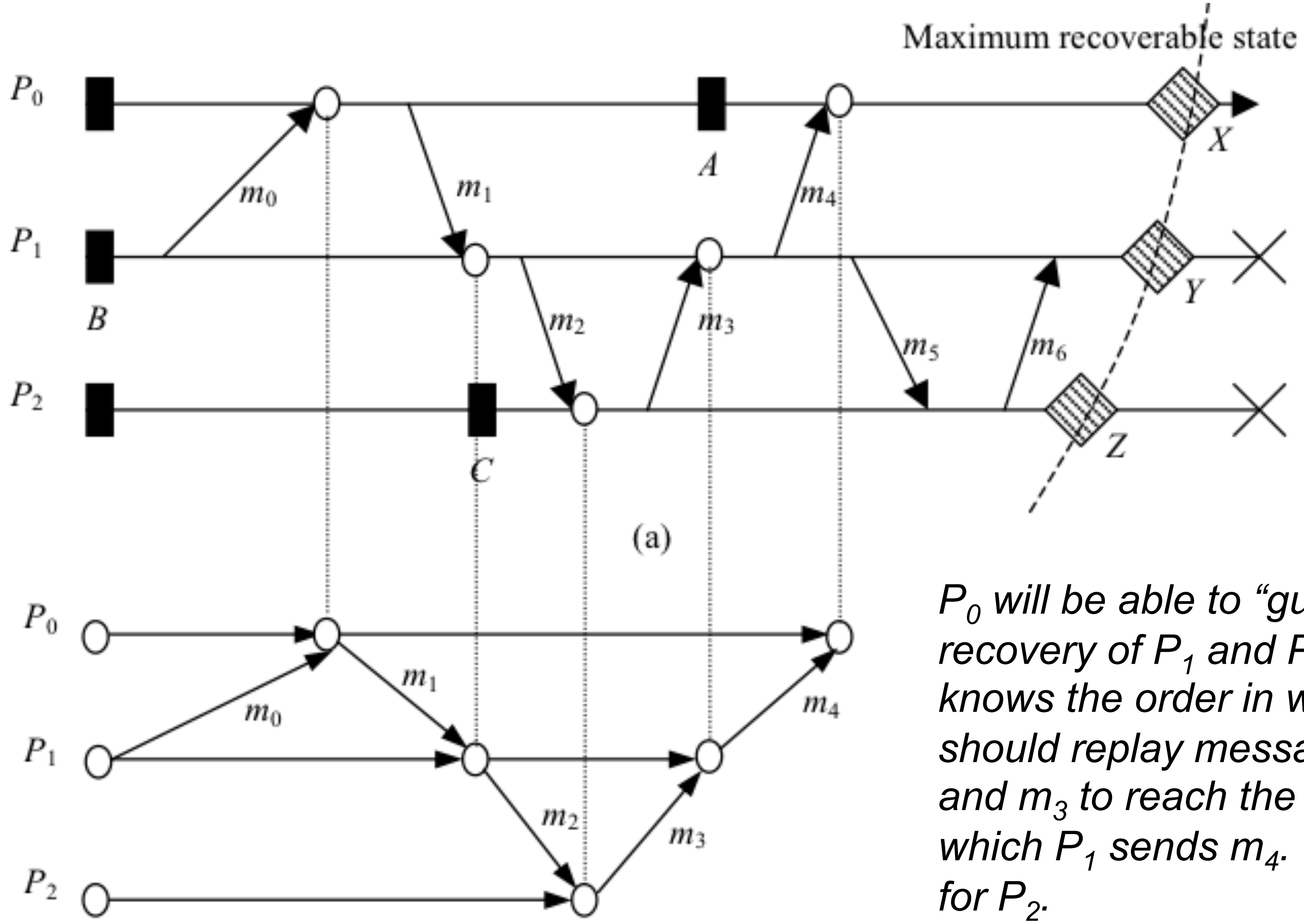
Log-Based Rollback Recovery / Causal Logging

- Has the failure-free performance advantages of optimistic logging while retaining most of the advantages of optimistic logging.
- Avoids synchronous access to stable storage except during output commit.
- Similar to pessimistic logging in:
  - ☐ **Allows each process to commit output independently.**
  - ☐ **Never creates orphan processes.**
  - ☐ **Limits the rollback of any failed process to the most recent checkpoint.**
- Cost: a more complex recovery protocol.

Log-Based Rollback Recovery / Causal Logging

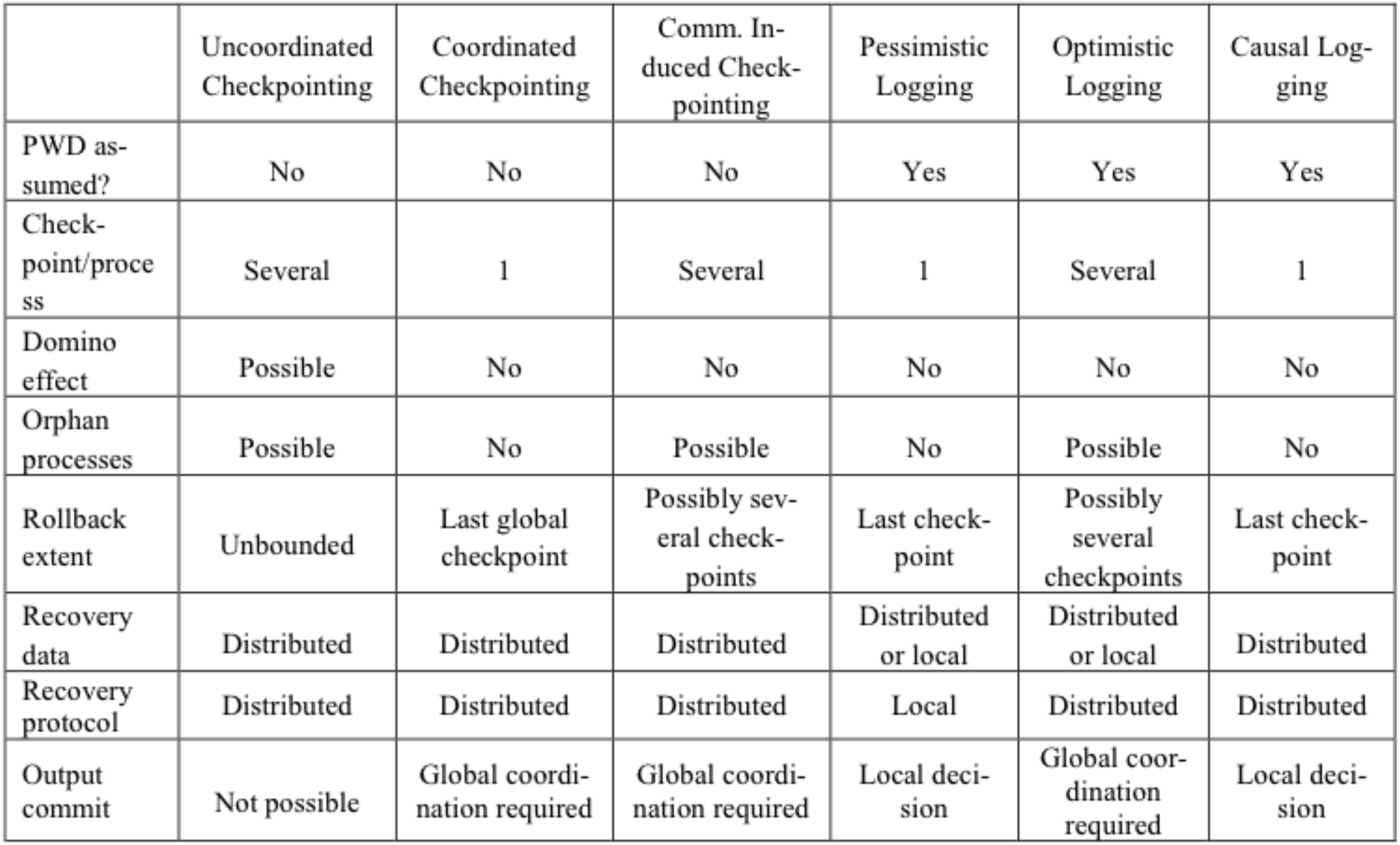
- Ensures the *always-no-orphans* property by ensuring that the determinant of each non-deterministic event that causally precedes the state of a process is either stable or it is available locally to that process.
- Processes piggyback the non-stable determinants in their volatile log on the messages they send to other processes.

Log-Based Rollback Recovery / Causal Logging



P_0 will be able to "guide" the recovery of P_1 and P_2 since it knows the order in which P_1 should replay messages m_1 and m_3 to reach the state from which P_1 sends m_4. Similarly for P_2.

Rollback-Recovery II



	Uncoordinated Checkpointing	Coordinated Checkpointing	Comm. Induced Checkpointing	Pessimistic Logging	Optimistic Logging	Causal Logging
PWD assumed?	No	No	No	Yes	Yes	Yes
Checkpoint/process	Several	1	Several	1	Several	1
Domino effect	Possible	No	No	No	No	No
Orphan processes	Possible	No	Possible	No	Possible	No
Rollback extent	Unbounded	Last global checkpoint	Possibly several checkpoints	Last checkpoint	Possibly several checkpoints	Last checkpoint
Recovery data	Distributed	Distributed	Distributed	Distributed or local	Distributed or local	Distributed
Recovery protocol	Distributed	Distributed	Distributed	Local	Distributed	Distributed
Output commit	Not possible	Global coordination required	Global coordination required	Local decision	Global coordination required	Local decision

Concluding Remarks

- Key properties: performance overhead, storage overhead, ease of output commit, ease of garbage collection, ease of recovery, freedom from domino effect, freedom from orphan processes, and the extent of rollback.
- Coordinated checkpointing generally simplifies recovery and garbage collection, and yields good performance in practice.
- the nondeterministic nature of communication-induced checkpointing protocols complicates garbage collection and degrades performance.
- Log-based rollback recovery is often a natural choice for applications that frequently interact with the outside world.

Thanks!

Questions?