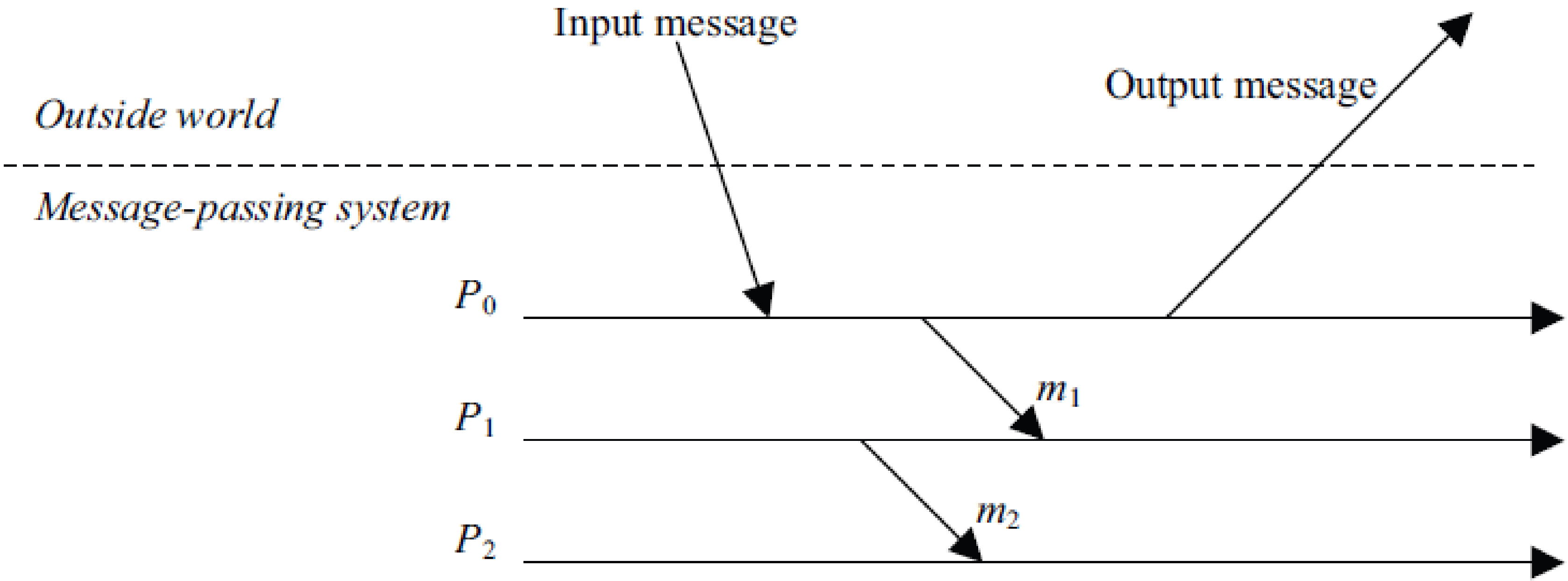
Nabil S. Al Ramli

Rollback-Recovery Protocols I

Message Passing Systems

Messages

- Message Passing System
  - **Messages**
  - **Processes**
- Outside world
  - **Input messages**
  - **Output messages**



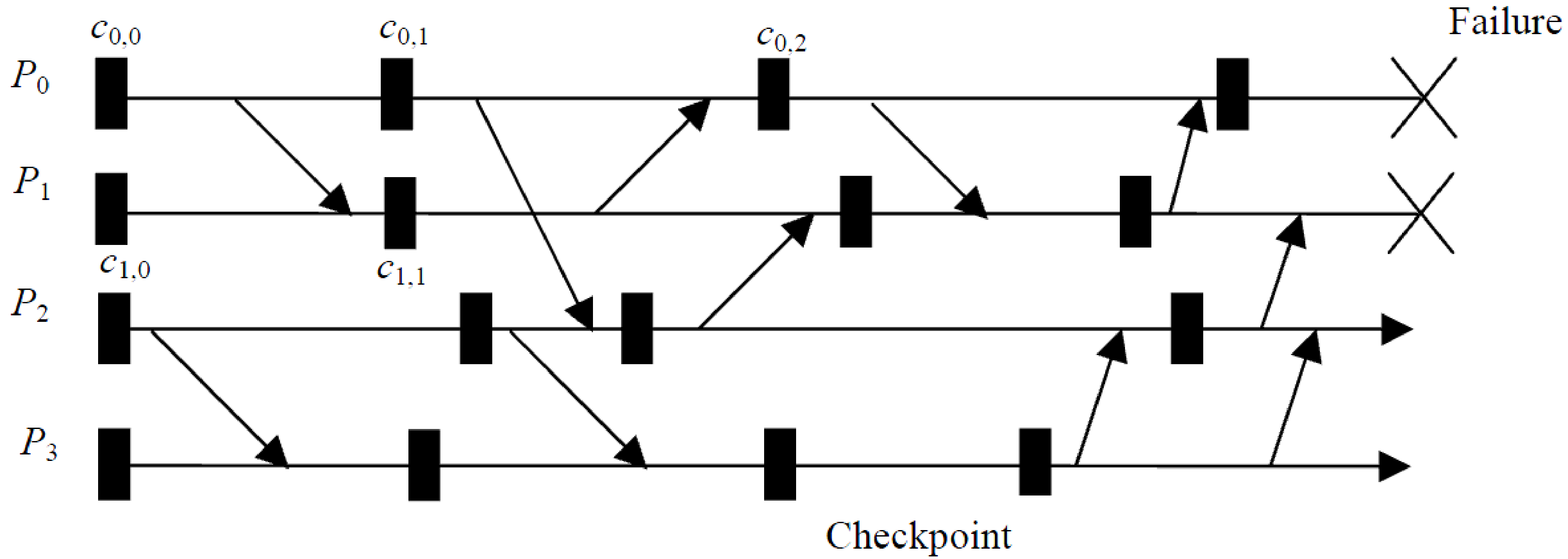
Outside World Process (OWP)

- A special process
  - Used to model how rollback recovery interacts with the outside world
    - Through messages
- Requirements
  - Cannot fail
  - Cannot maintain state
  - Cannot participate in recovery
  - Cannot roll back

Messages to OWP

- OWP must perceive a consistent behavior of the system despite failures
  - Input messages from OWP may not be reproducible during recovery
  - Output messages cannot be reverted
- State that sent message to OWP must be recoverable
- Save each input message on stable storage before allowing the application program to process it

Checkpoints



Stable Storage

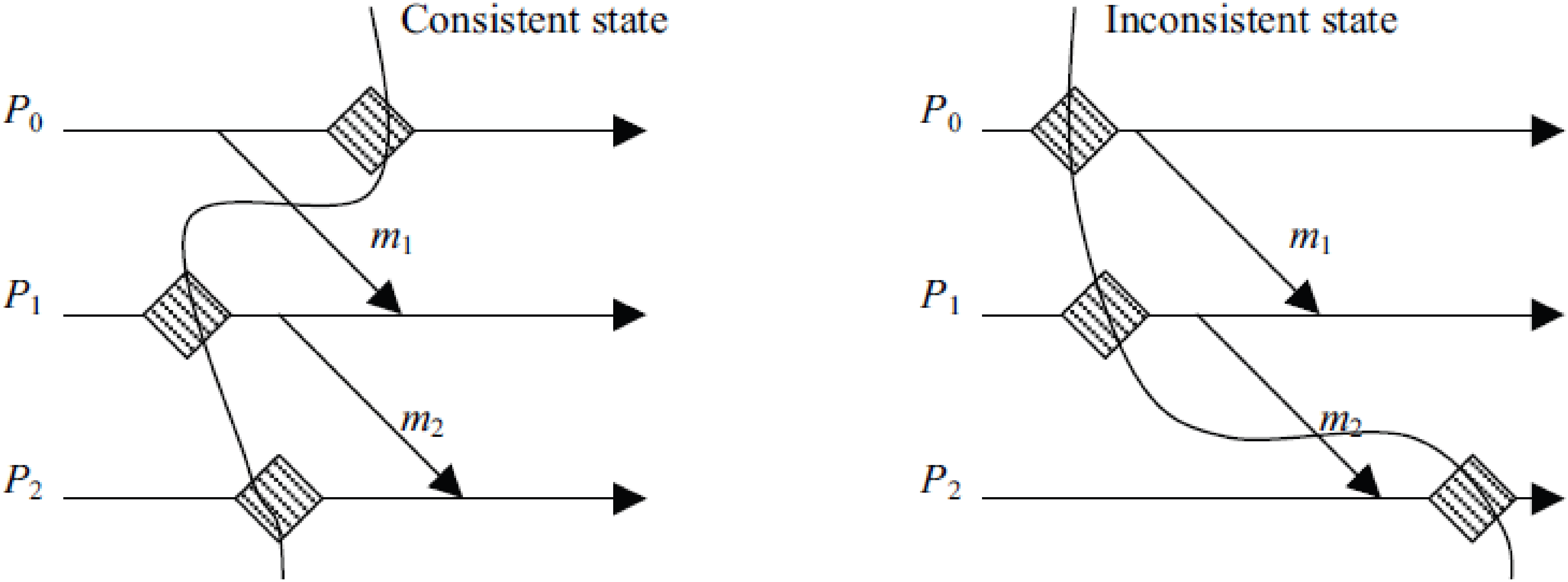
- must store recovery data through failures
  - Checkpoints, event logs, other recovery info
- Implementation options
  - A system that tolerates only a single failure
    - Volatile memory
  - A system that tolerates transient failures
    - Local disk in each host
  - A system that tolerates non-transient failures
    - A replicated file system

Garbage Collection

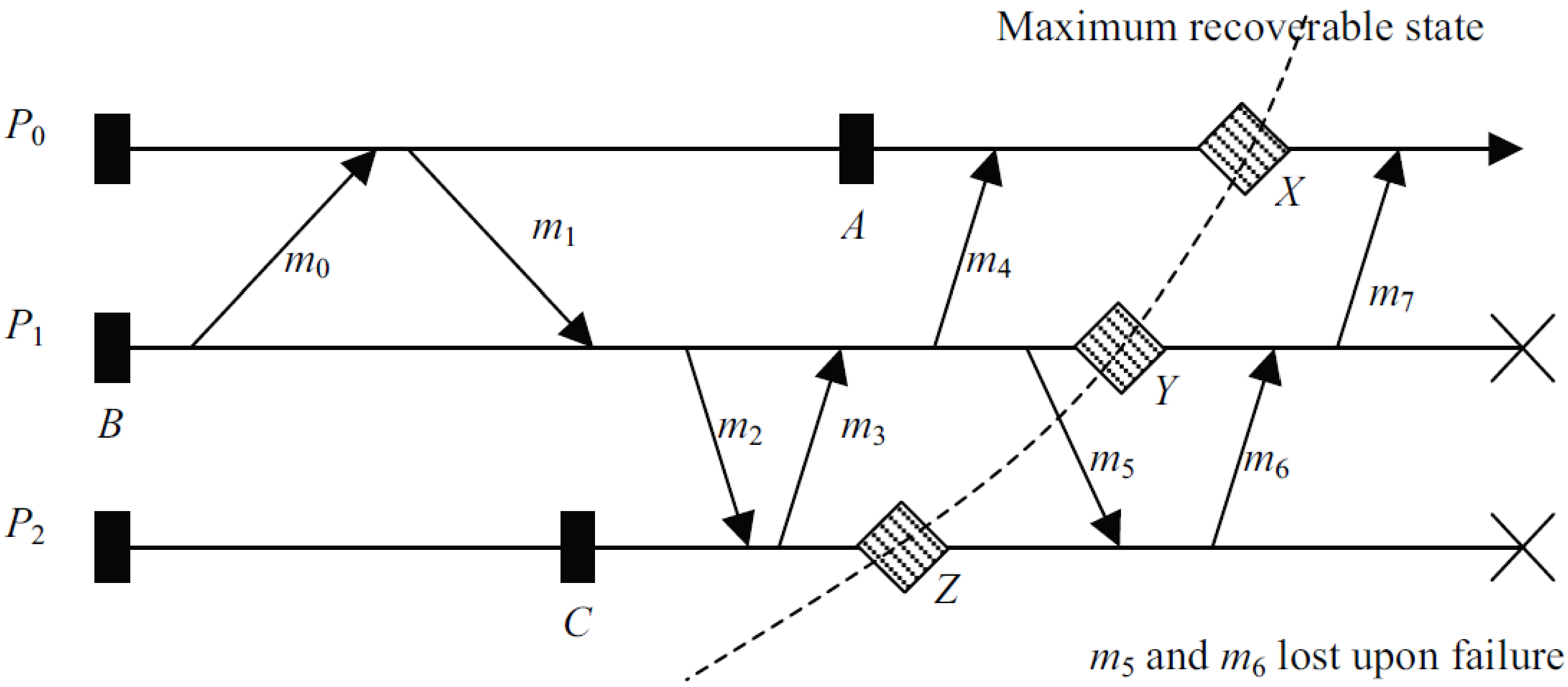
- Checkpoints and event logs consume storage
- Some information may become useless
- Identify most recent consistent set of checkpoints
  - Recovery line
- Discard information before recovery line

Consistent System States

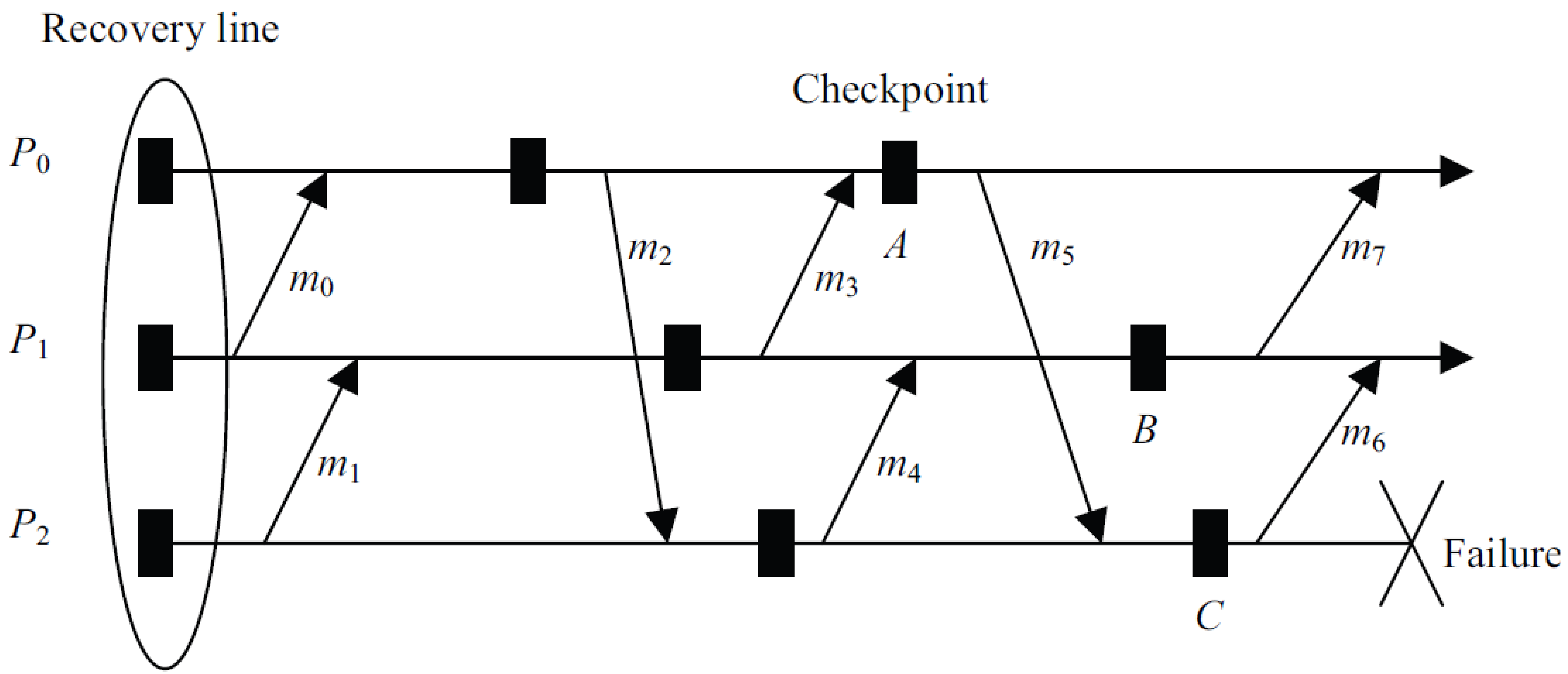
- Lost Messages
  - **Sent but never received - OK**
- "Orphan Messages"
  - **Received but never sent - bad**



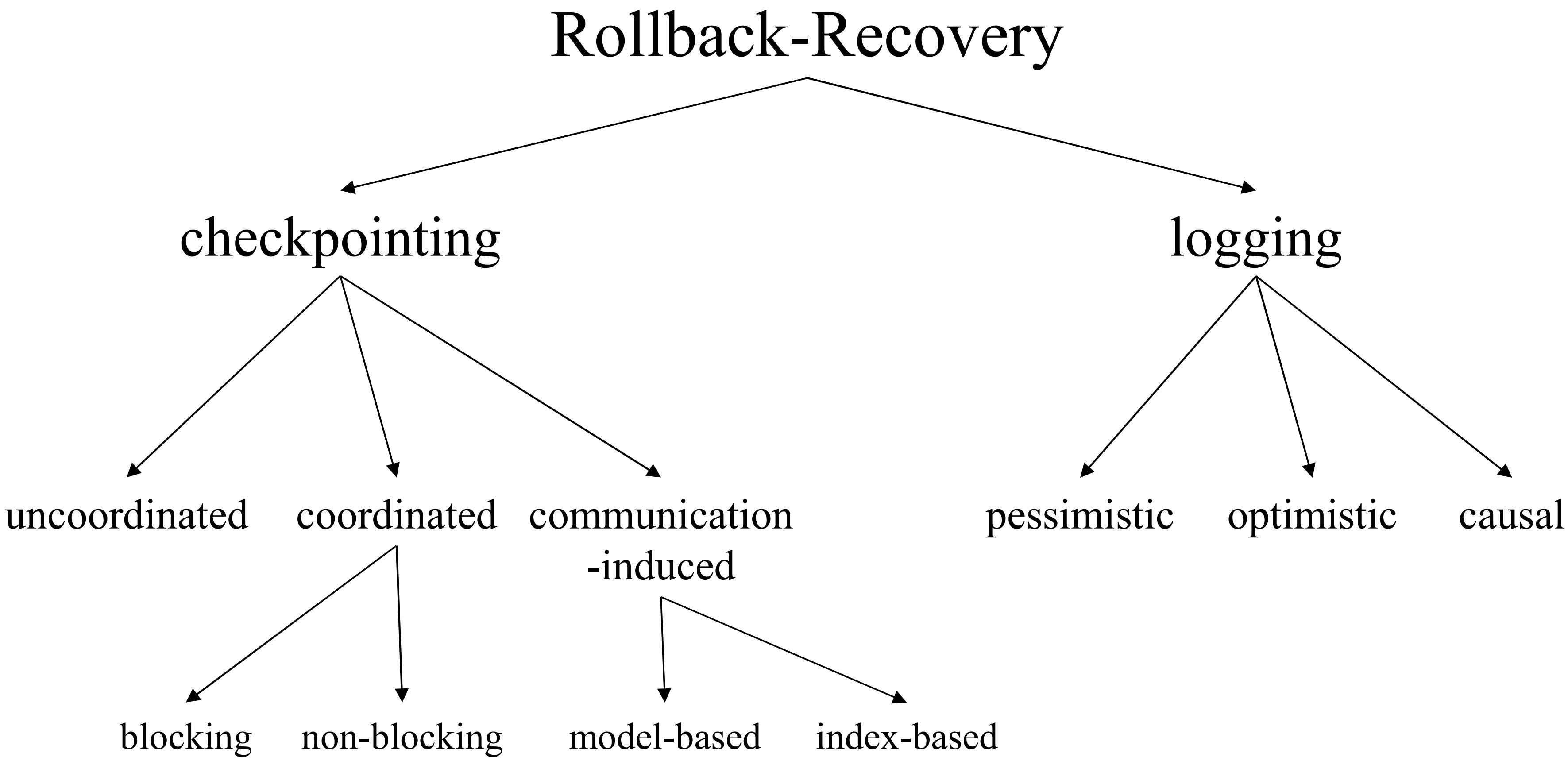
Maximum Recoverable State



The Domino Effect



Taxonomy


Rollback-Recovery
checkpointing
logging
uncoordinated
coordinated
communication
-induced
pessimistic
optimistic
causal
blocking
non-blocking
model-based
index-based

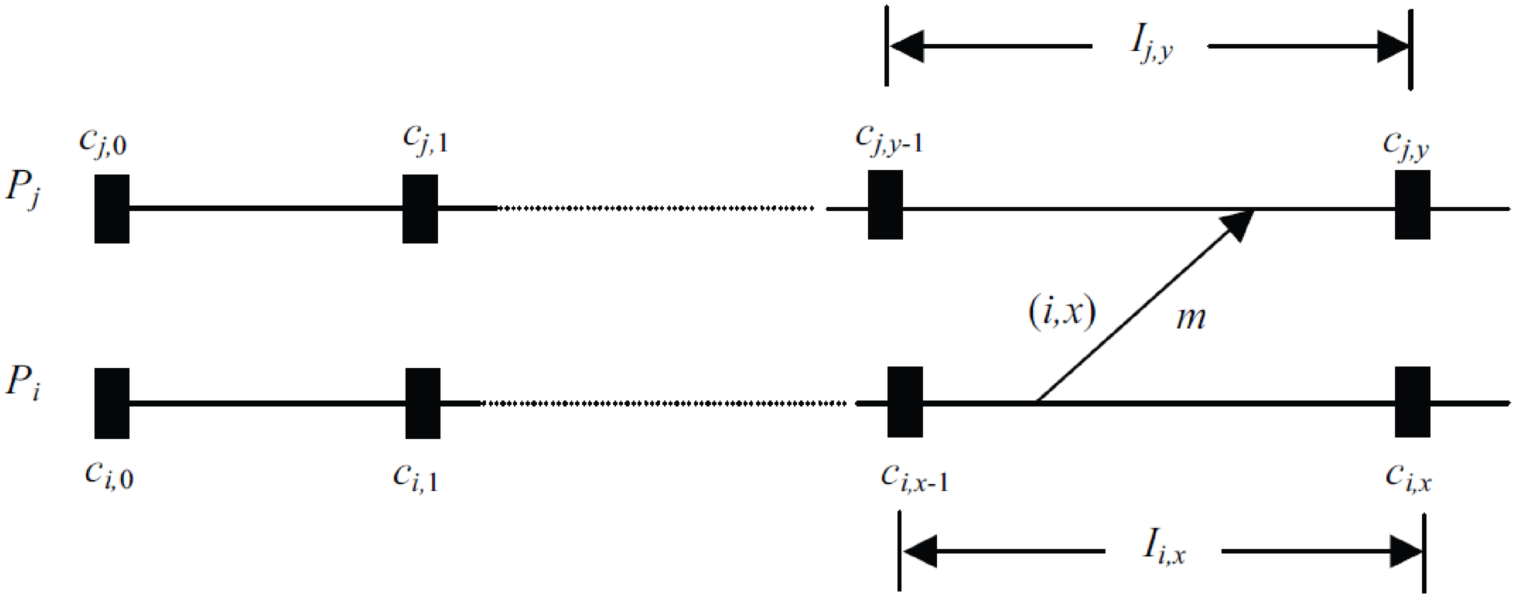
Checkpoint-Based Rollback Recovery

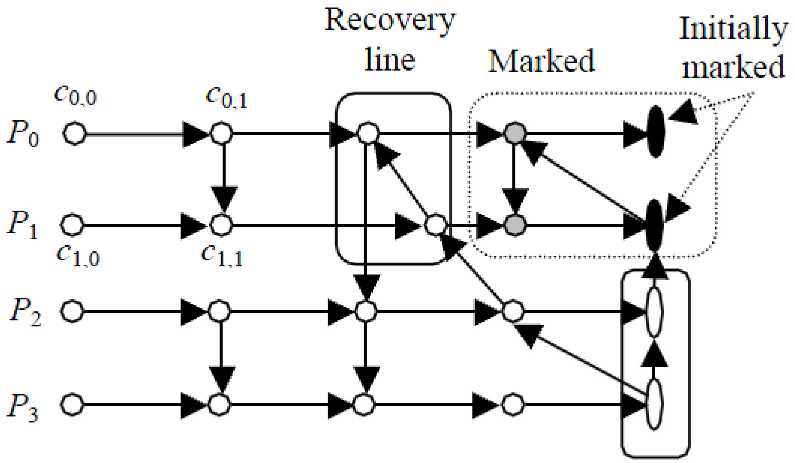
- restores the system state to the recovery line
- Does not rely on the PWD assumption
- less restrictive and simpler to implement
- Does not guarantee that prefailure execution can be deterministically regenerated after a rollback
- Not suited for interactions with the outside world
- Categories
  - Uncoordinated checkpointing
  - Coordinated checkpointing
  - Communication-induced checkpointing

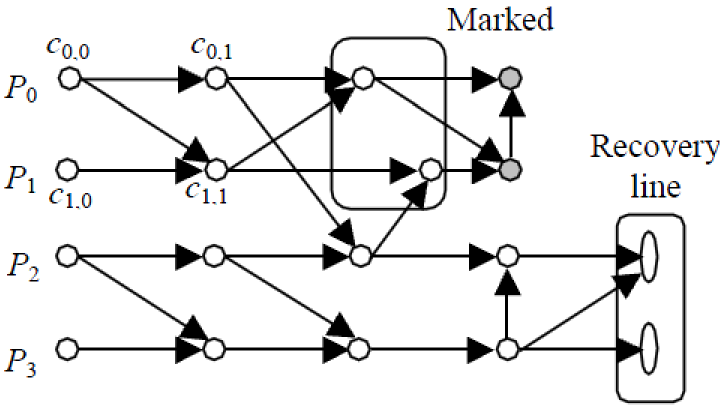
Uncoordinated Checkpointing

- Each process takes checkpoints independently
- Recovery line must be calculated after failure
- Disadvantages
  - susceptible to domino effect
  - can generate useless checkpoints
  - complicates storage/GC
  - not suitable for frequent output commits

Uncoordinated Checkpointing







Coordinated Checkpointing

- Checkpoints are orchestrated between processes
- Triggered by application decision
- Simplifies recovery
- Not susceptible to the domino effect
- Only one checkpoint per process on stable storage
- Garbage collection not necessary
- Large latency

Coordinated Checkpointing / Blocking

- No messages can be in transit during checkpointing
- Large overhead

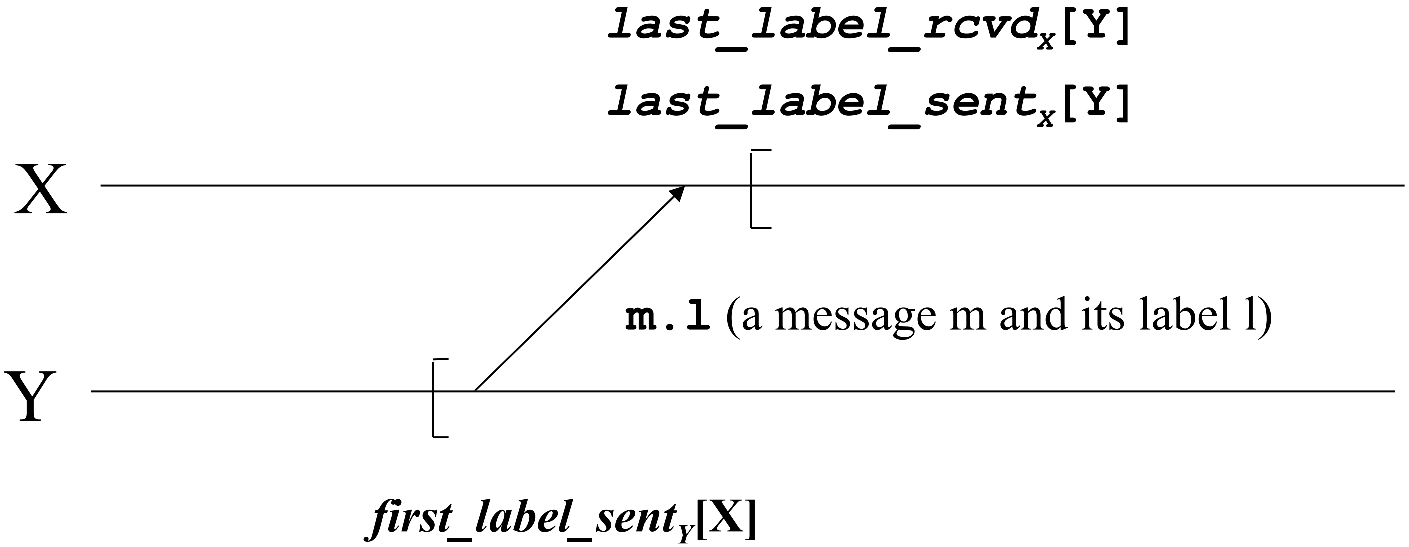
Two-Phase Checkpointing Protocol

- A coordinator takes a checkpoint
- Broadcasts a checkpoint request to all processes
- When a process receives this message, it stops its execution, takes a tentative checkpoint
- Send an acknowledgment back to coordinator
- Coordinator broadcasts a commit message
- Each process removes the old checkpoint and makes the tentative checkpoint permanent

Coordinated/Blocking Notation

Each node maintains:

- a monotonically increasing counter with which each message from that node is labeled.
- records of the last message from/to and the first message to all other nodes.



$\texttt{last\_label\_rcvd}_X\texttt{[Y]}$

$\texttt{last\_label\_sent}_X\texttt{[Y]}$

X

`m.l` (a message m and its label l)

Y

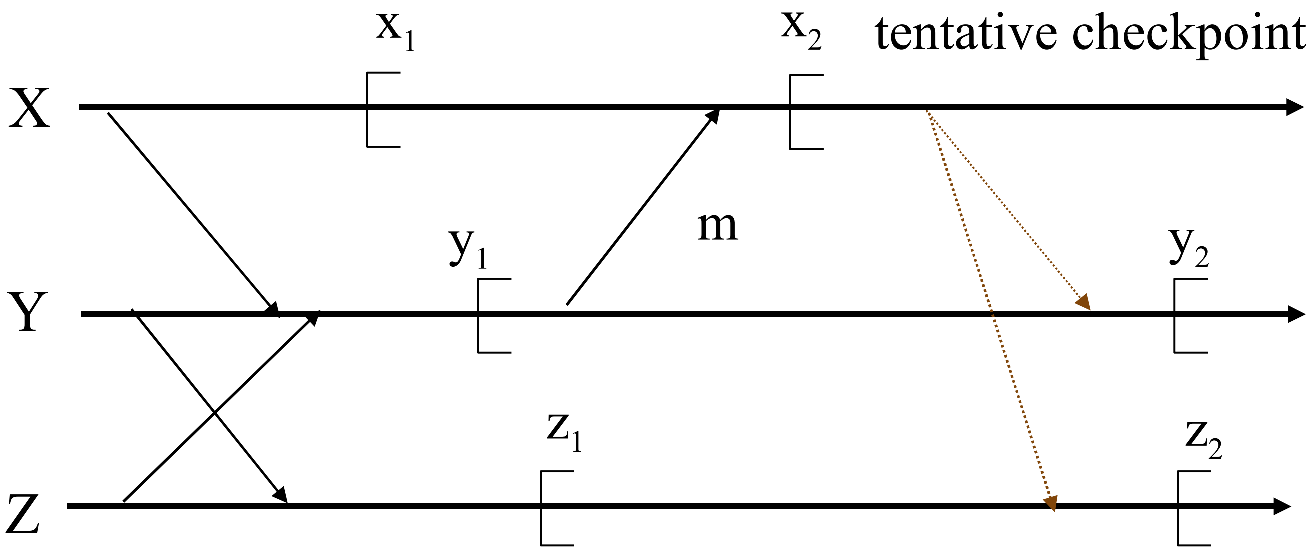
first_label_sent$_Y$[X]

Note: “sl” denotes a “smallest label” that is < any other label and
“ll” denotes a “largest label” that is > any other label

Coordinated/Blocking Algorithm

(1) When must I take a checkpoint?

(2) Who else has to take a checkpoint when I do?



(1) When I (Y) have sent a message to the checkpointing process, X, since my last checkpoint:

$$\mathit{last\_label\_rcvd}_X[Y] >= \mathit{first\_label\_sent}_Y[X] > sl$$

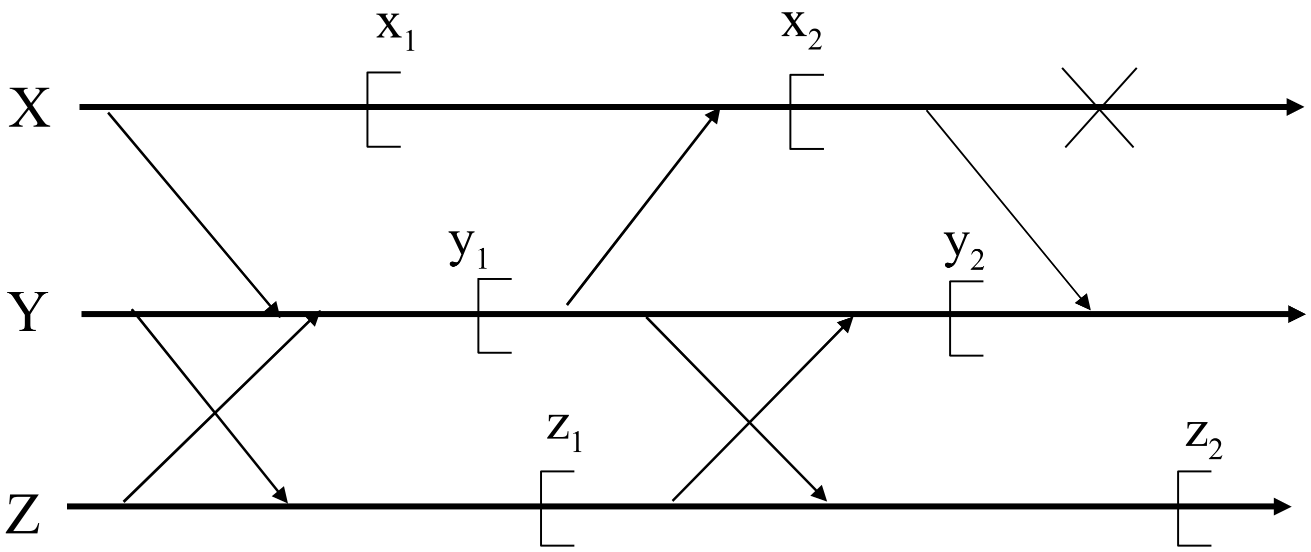
(2) Any other process from whom I have received messages since my last checkpoint.

$$\mathit{ckpt\_cohort}_X = \{Y \mid \mathit{last\_label\_rcvd}_X[Y] > sl\}$$

Coordinated/Blocking Algorithm

(1) When must I rollback?

(2) Who else might have to rollback when I do?



(1) When I ,Y, have received a message from the restarting process,X, since X's last checkpoint.

$$\mathit{last\_label\_rcvd}_Y(X) > \mathit{last\_label\_sent}_X(Y)$$

(2) Any other process to whom I can send messages.

$$\mathit{roll\_cohort}_Y = \{Z \mid Y \text{ can send message to } Z\}$$

Coordinated Checkpointing / Non-Blocking

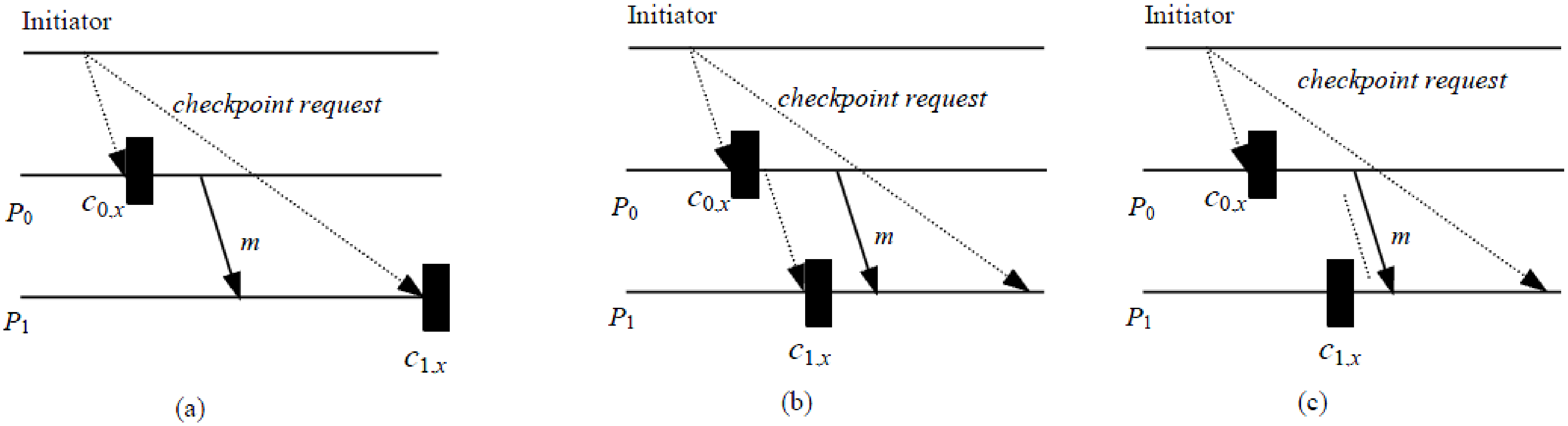


Figure 8. Non-blocking coordinated checkpointing: (a) checkpoint inconsistency; (b) with FIFO channels; (c) non-FIFO channels (short dashed line represents piggybacked *checkpoint request*).

Questions

?