

#### 4.1 Check pointing and rollback recovery: Introduction

- Rollback recovery protocols restore the system back to a consistent state after a failure,
- It achieves fault tolerance by periodically saving the state of a process during the failure-free execution
- It treats a distributed system application as a collection of processes that communicate over a network

##### Checkpoints

The saved state is called a checkpoint, and the procedure of restarting from a previously checkpointed state is called rollback recovery. A checkpoint can be saved on either the stable storage or the volatile storage

##### Why is rollback recovery of distributed systems complicated?

Messages induce inter-process dependencies during failure-free operation

##### Rollback propagation

The dependencies among messages may force some of the processes that did not fail to roll back.

This phenomenon of cascaded rollback is called the domino effect.

##### Uncoordinated check pointing

If each process takes its checkpoints independently, then the system cannot avoid the domino effect – this scheme is called independent or uncoordinated check pointing

##### Techniques that avoid domino effect

1. Coordinated check pointing rollback recovery - Processes coordinate their checkpoints to form a system-wide consistent state
2. Communication-induced check pointing rollback recovery - Forces each process to take checkpoints based on information piggybacked on the application.
3. Log-based rollback recovery - Combines check pointing with logging of non-deterministic events • relies on piecewise deterministic (PWD) assumption.

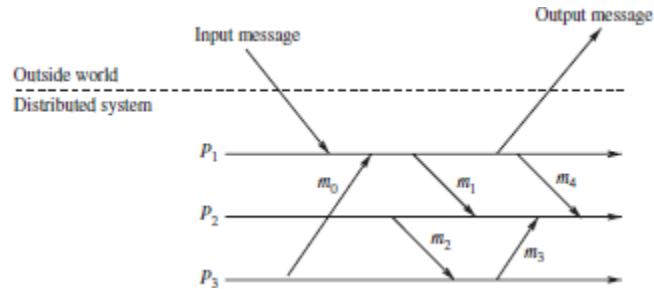
#### 4.2 Background and definitions

##### 4.2.1 System model

- A distributed system consists of a fixed number of processes,  $P_1, P_2, \dots, P_N$ , which 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, respectively.
- Rollback-recovery protocols generally make assumptions about the reliability of the inter-

process communication.

- Some protocols assume that the communication uses first-in-first-out (FIFO) order, while other protocols assume that the communication subsystem can lose, duplicate, or reorder messages.
- Rollback-recovery protocols therefore must maintain information about the internal interactions among processes and also the external interactions with the outside world.



An example of a distributed system with three processes.

#### 4.2.2 A local checkpoint

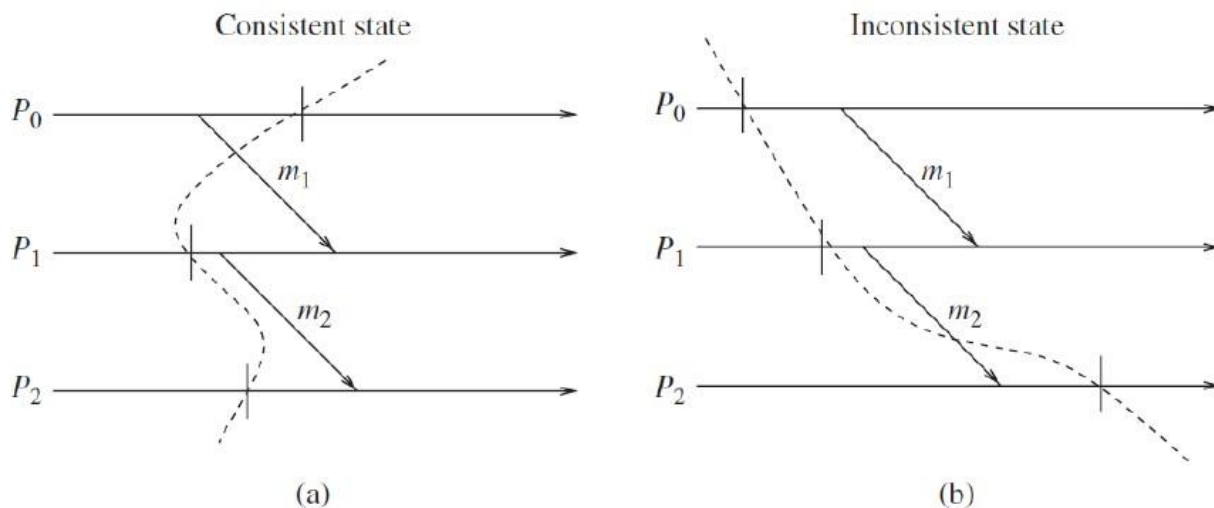
- All processes save their local states at certain instants of time
- A local check point is a snapshot of the state of the process at a given instance
- Assumption
  - A process stores all local checkpoints on the stable storage
  - A process is able to roll back to any of its existing local checkpoints
- $C_{i,k}$  – The  $k$ th local checkpoint at process  $P_i$
- $C_{i,0}$  – A process  $P_i$  takes a checkpoint  $C_{i,0}$  before it starts execution

#### 4.2.3 Consistent states

- A global state of a distributed system is 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 is a set of local checkpoints, one from each process

- A consistent global checkpoint is 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.

## Consistent states - examples



- For instance, Figure shows two examples of global states.
- The state in fig (a) is consistent and the state in Figure (b) is inconsistent.
- Note that the consistent state in Figure (a) shows message  $m_1$  to have been sent but not yet received, but that is alright.
- The state in Figure (a) is consistent because it represents a situation in which every message that has been received, there is a corresponding message send event.
- The state in Figure (b) is inconsistent because process  $P_2$  is shown to have received  $m_2$  but the state of process  $P_1$  does not reflect having sent it.
- Such a state is impossible in any failure-free, correct computation. Inconsistent states occur because of failures.

### 4.2.4 Interactions with outside world

A distributed system often 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. For example, a printer cannot roll back the effects of printing a character

### Outside World Process (OWP)

- It is a special process that interacts with the rest of the system through message passing.
- It is therefore necessary that the outside world see a consistent behavior of the system despite failures.
- Thus, 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.

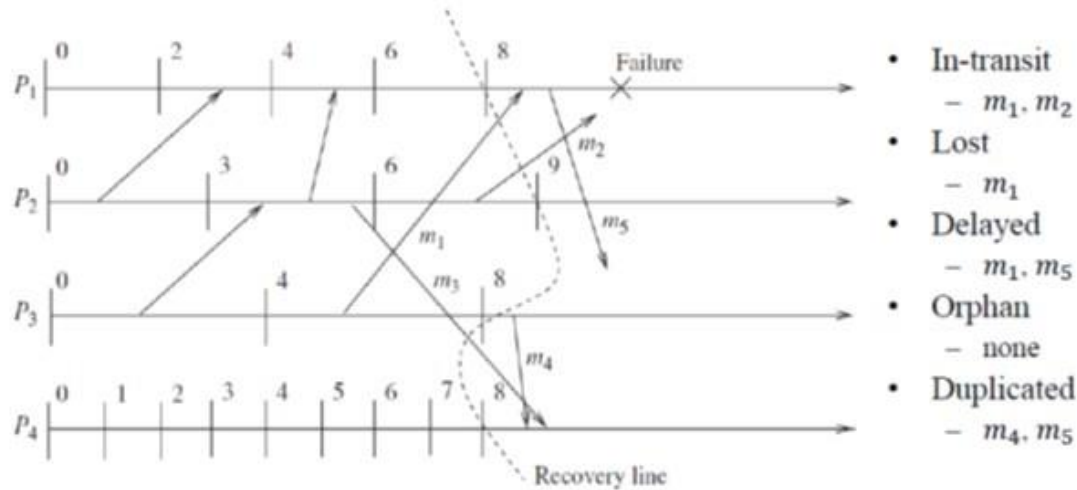
**A common approach is to** save each input message on the stable storage before allowing the application program to process it.

An interaction with the outside world to deliver the outcome of a computation is shown on the process-line by the symbol “||”.

#### 4.2.5 Different types of Messages

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

## Messages – example



### In-transit messages

In Figure , the global state  $\{C1,8, C2,9, C3,8, C4,8\}$  shows that message  $m_1$  has been sent but not yet received. We call such a message an *in-transit* message. Message  $m_2$  is also an in-transit message.

### 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, are called *delayed* messages. For example, messages  $m_2$  and  $m_5$  in Figure are delayed messages.

### Lost messages

Messages whose send is not undone but receive is undone due to rollback are called *lost* messages. This type of messages occurs when the process rolls back to a checkpoint prior to reception of the message while the sender does not rollback beyond the send operation of the message. In Figure , message  $m_1$  is a lost message.

### Duplicate messages

- Duplicate messages arise due to message logging and replaying during process recovery. For example, in Figure, message  $m_4$  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 message  $m_4$  are undone.

- When process P3 restarts from C3,8, it will resend message m4.
- Therefore, P4 should not replay message m4 from its log.
- If P4 replays message m4, then message m4 is called a duplicate message.