

# Unit 3

## Recovery

# Recovery

- ▶ Computer system recovery:
  - Restore the system to a normal operational state
- ▶ Process recovery:
  - Reclaim resources allocated to process,
  - Undo modification made to databases, and
  - Restart the process
  - Or restart process from point of failure and resume execution
- ▶ Distributed process recovery (cooperating processes):
  - Undo effect of interactions of failed process with other cooperating processes.
- ▶ Replication (hardware components, processes, data):
  - Main method for increasing system availability
- ▶ System:
  - Set of hardware and software components
  - Designed to provide a specified service (I.e. meet a set of requirements)

# Recovery (cont.)

## System failure:

- System does not meet requirements, i.e. does not perform its services as specified

↑  
Error could lead to system failure

## Erroneous System State:

- State which could lead to a system failure by a sequence of valid state transitions
- Error: the part of the system state which differs from its intended value

↑  
Error is a manifestation of a fault

## Fault:

- Anomalous physical condition, e.g. design errors, manufacturing problems, damage, external disturbances.

# Classification of failures

## ▶ Process failure:

- Behavior: process causes system state to deviate from specification (e.g. incorrect computation, process stop execution)
- Errors causing process failure: protection violation, deadlocks, timeout, wrong user input, etc...
- Recovery: Abort process or  
Restart process from prior state

## ▶ System failure:

- Behavior: processor fails to execute
- Caused by software errors or hardware faults (CPU/memory/bus/.../ failure)
- Recovery: system stopped and restarted in correct state
- Assumption: fail-stop processors, i.e. system stops execution, internal state is lost

## ▶ Secondary Storage Failure:

- Behavior: stored data cannot be accessed
- Errors causing failure: parity error, head crash, etc.
- Recovery/Design strategies:  
Reconstruct content from archive + log of activities  
Design mirrored disk system

## ▶ Communication Medium Failure:

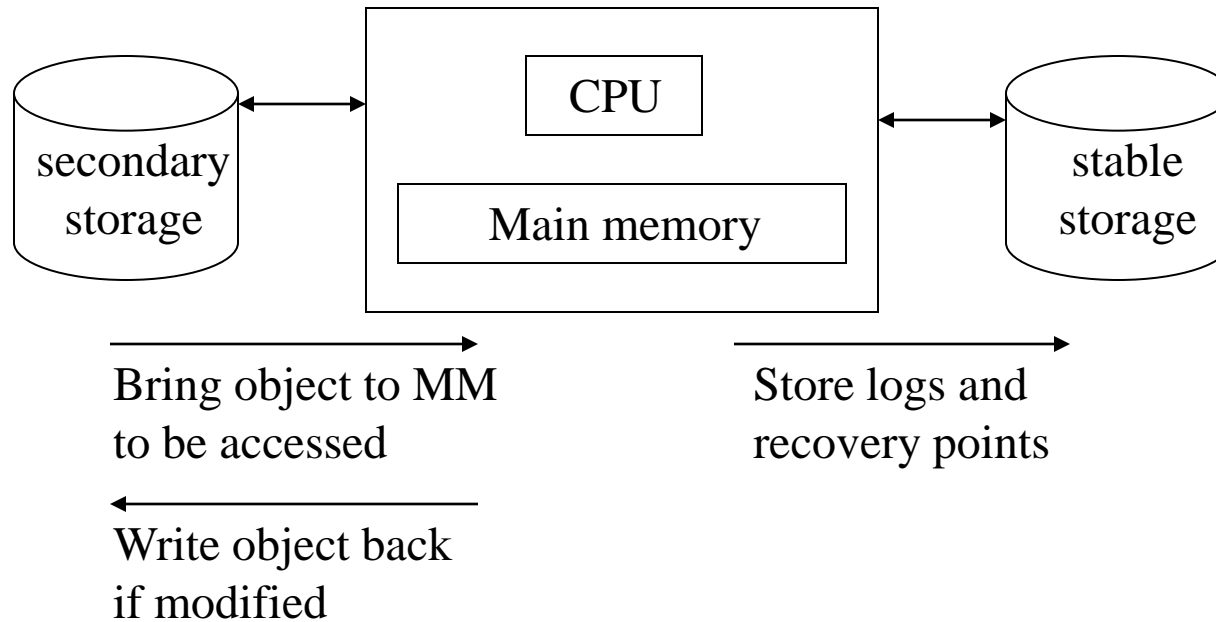
- Behavior: a site cannot communicate with another operational site
- Errors/Faults: failure of switching nodes or communication links
- Recovery/Design Strategies: reroute, error-resistant communication protocols

# Backward and Forward Error Recovery

- ▶ Failure recovery: restore an erroneous state to an error-free state
- ▶ Approaches to failure recovery:
  - Forward-error recovery:
    - Remove errors in process/system state (if errors can be completely assessed)
    - Continue process/system forward execution
  - Backward-error recovery:
    - Restore process/system to previous error-free state and restart from there
- ▶ Comparison: Forward vs. Backward error recovery
  - Backward-error recovery
    - (+) Simple to implement
    - (+) Can be used as general recovery mechanism
    - (-) Performance penalty
    - (-) No guarantee that fault does not occur again
    - (-) Some components cannot be recovered
  - Forward-error Recovery
    - (+) Less overhead
    - (-) Limited use, i.e. only when impact of faults understood
    - (-) Cannot be used as general mechanism for error recovery

# Backward-Error Recovery: Basic approach

- ▶ Principle: restore process/system to a known, error-free “recovery point”/ “checkpoint”.
- ▶ System model:



Storage that maintains information in the event of system failure

- ▶ Approaches:
  - (1) Operation-based approach
  - (2) State-based approach

# (1) The Operation-based Approach

## ▶ Principle:

- Record all changes made to state of process ('audit trail' or 'log') such that process can be returned to a previous state
- Example: A transaction based environment where transactions update a database
  - It is possible to commit or undo updates on a per-transaction basis
  - A commit indicates that the transaction on the object was successful and changes are permanent

## (1.a) Updating-in-place

- Principle: every update (write) operation to an object creates a log in stable storage that can be used to '*undo*' and '*redo*' the operation
- Log content: object name, old object state, new object state
- Implementation of a recoverable update operation:
  - *Do* operation: update object and write log record
  - *Undo* operation: log(old)  $\rightarrow$  object (undoes the action performed by a *do*)
  - *Redo* operation: log(new)  $\rightarrow$  object (redoes the action performed by a *do*)
  - *Display* operation: display log record (optional)
- Problem: a '*do*' cannot be recovered if system crashes after write object but before log record write

## (1.b) The write-ahead log protocol

- Principle: write log record before updating object

## (2) State-based Approach

▶ Principle: establish frequent ‘recovery points’ or ‘checkpoints’ saving the entire state of process

▶ Actions:

- ‘Checkpointing’ or ‘taking a checkpoint’: saving process state
- ‘Rolling back’ a process: restoring a process to a prior state

Note: A process should be rolled back to the most recent ‘recovery point’ to minimize the overhead and delays in the completion of the process

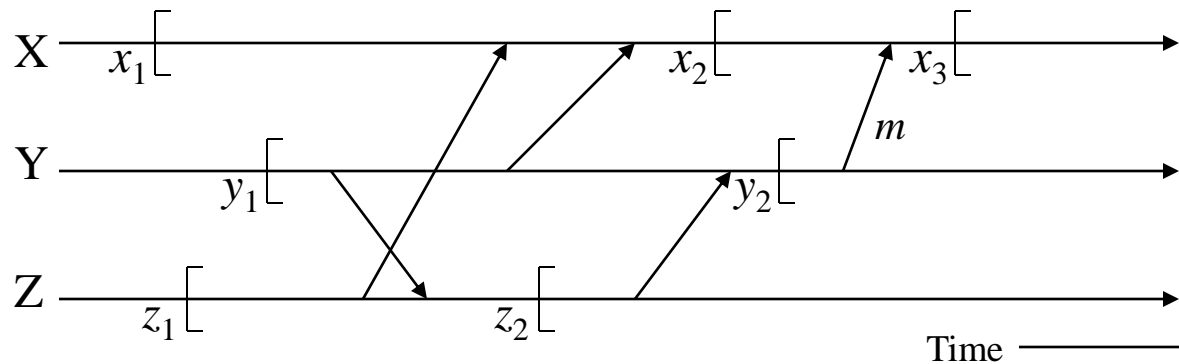
▶ Shadow Pages: Special case of state-based approach

- Only a part of the system state is saved to minimize recovery
- When an object is modified, page containing object is first copied on stable storage (shadow page)
- If process successfully commits: shadow page discarded and modified page is made part of the database
- If process fails: shadow page used and the modified page discarded



# Recovery in concurrent systems

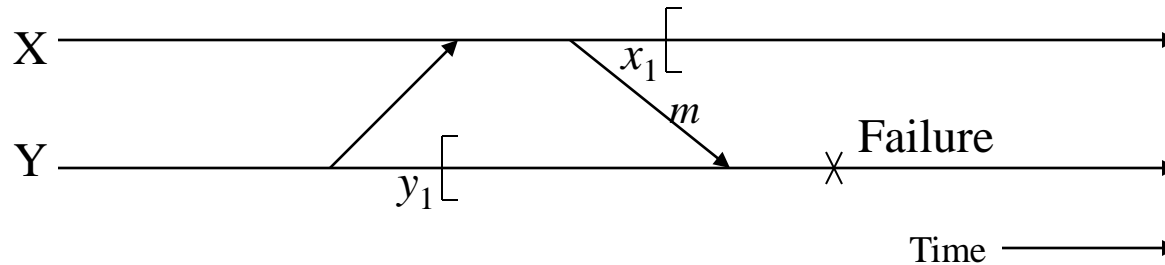
- ▶ Issue: if one of a set of cooperating processes fails and has to be rolled back to a recovery point, all processes it communicated with since the recovery point have to be rolled back.
- ▶ Conclusion: In concurrent and/or distributed systems all cooperating processes have to establish recovery points
- ▶ Orphan messages and the domino effect



- Case 1: failure of X after  $x_3$  : no impact on Y or Z
- Case 2: failure of Y after sending msg. ' $m$ '
  - Y rolled back to  $y_2$
  - ' $m$ '  $\equiv$  orphan message
  - X rolled back to  $x_2$
- Case 3: failure of Z after  $z_2$ 
  - Y has to roll back to  $y_1$
  - X has to roll back to  $x_1$
  - Z has to roll back to  $z_1$

} Domino Effect

# Lost messages

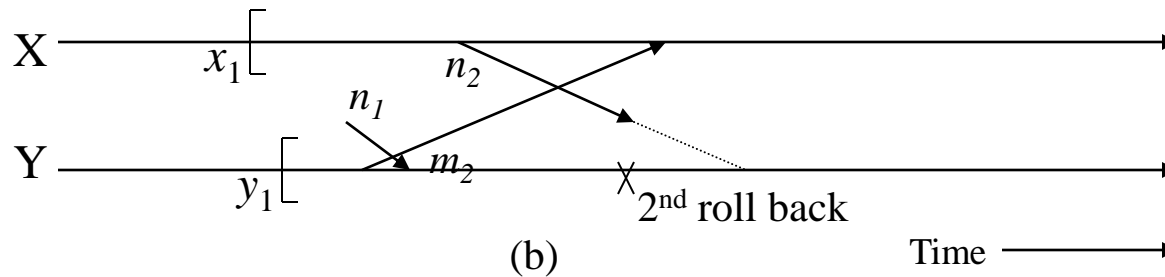
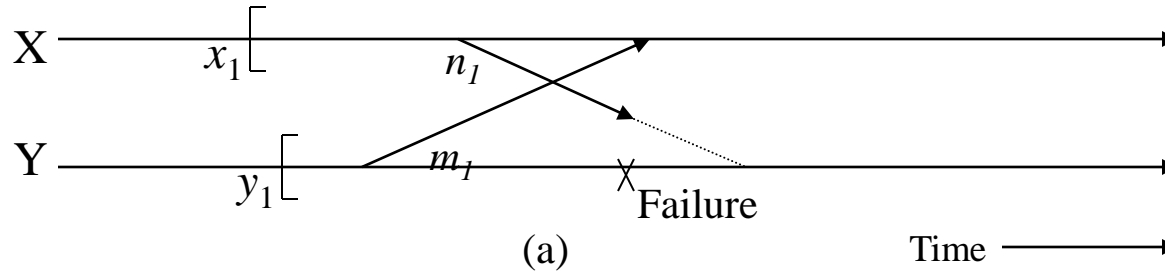


- ▶ Assume that  $x_1$  and  $y_1$  are the only recovery points for processes X and Y, respectively
- ▶ Assume Y fails after receiving message ' $m$ '
- ▶ Y rolled back to  $y_1$ , X rolled back to  $x_1$
- ▶ Message ' $m$ ' is lost

Note: there is no distinction between this case and the case where message ' $m$ ' is lost in communication channel and processes X and Y are in states  $x_1$  and  $y_1$ , respectively

# Problem of livelock

- ▶ Livelock: case where a single failure can cause an infinite number of rollbacks



- (a) ▶ Process Y fails before receiving message ' $n_1$ ' sent by X
- ▶ Y rolled back to  $y_1$ , no record of sending message ' $m_1$ ', causing X to roll back to  $x_1$
- (b) ▶ When Y restarts, sends out ' $m_2$ ' and receives ' $n_1$ ' (delayed)
- ▶ When X restarts from  $x_1$ , sends out ' $n_2$ ' and receives ' $m_2$ '
- ▶ Y has to roll back again, since there is no record of ' $n_1$ ' being sent
- ▶ This cause X to be rolled back again, since it has received ' $m_2$ ' and there is no record of sending ' $m_2$ ' in Y
- ▶ The above sequence can repeat indefinitely

# Consistent set of checkpoints

- ▶ Checkpointing in distributed systems requires that all processes (sites) that interact with one another establish periodic checkpoints
- ▶ All the sites save their local states: *local checkpoints*
- ▶ All the local checkpoints, one from each site, collectively form a *global checkpoint*
- ▶ The domino effect is caused by orphan messages, which in turn are caused by rollbacks

## 1. Strongly consistent set of checkpoints

- Establish a set of local checkpoints (one for each process in the set) such that no information flow takes place (i.e., no orphan messages) during the interval spanned by the checkpoints

## 2. Consistent set of checkpoints

- Similar to the consistent global state
- Each message that is received in a checkpoint (state) should also be recorded as sent in another checkpoint (state)