An error is a manifestation of a fault that can lead to a failure.

Failure Recovery:
• backward recovery
  • operation-based (undo/redo logs)
  • state-based (checkpoints)
• forward recovery

Domino Effect

Cases:
• X fails after $x_3$
• Y fails after sending message $m$
• Z fails after sending message $n$
Orphan Message

The checkpoint set \( \{x_1, y_1\} \) inconsistent

Lost Messages

The checkpoint set \( \{x_1, y_1\} \) inconsistent

Strongly Consistent Checkpoints

The set of checkpoints \( \{x_1, y_1, z_1\} \) are strongly consistent - no recovery ever need rollback past this set. The set stops the domino effect. \( \{x_2, y_2, z_2\} \) is a consistent set.
Checkpoint Notation

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

\[
\text{last_label_rcvd}_x[Y]
\]

\[
\text{first_label_sent}_x[X]
\]

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

Checkpoint 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:

\[
\text{last_label_rcvd}_Y[X] \geq \text{first_label_sent}_X[X] > \text{sl}
\]

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

\[
\text{ckpt_cohort}_x = \{Y \mid \text{last_label_rcvd}_Y[X] > \text{sl}\}
\]
Checkpoint Algorithm

Initiator Process $P_i$:

for all $p \in \text{ckpt_cohort}_{P_i}$
    send $\text{TakeTentativeCheckpoint}(P_i, \text{last_label_rcvd}_{P_i}[p])$;

if all cohorts replied “yes” then
    for all $p \in \text{ckpt_cohort}(P_i)$
        send $\text{MakeCheckpointPermanent}$;
else
    for all $p \in \text{ckpt_cohort}(P_i)$
        send $\text{UndoTentativeCheckpoint}$;

Checkpoint Algorithm

A Cohort process, $p$:

On receiving $\text{TakeTentativeCheckpoint}(q, \text{last_label_rcvd}_q(p))$:
    if $\text{OK_to_take_ckpt}_p := \text{“yes”}$ and
    $\text{last_label_rcvd}_q[p] >= \text{first_label_sent}_p[q] > s1$
    then $\text{TakeTentativeCheckpoint}$;
    for all $r \in \text{ckpt_cohort}_p$
        send $\text{TakeTentativeCheckpoint}(p, \text{last_label_rcvd}_p[r])$;
    if all cohorts replied “yes”
    then $\text{OK_to_take_ckpt}_p := \text{“yes”}$;
    else $\text{OK_to_take_ckpt}_p := \text{“no”}$;
    send($p, \text{OK_to_take_ckpt}_p$) to $q$;

On receiving $\text{MakeCheckpointPermanent}$:
    make checkpoint permanent;
    for all $r \in \text{ckpt_cohort}_p$
        send $\text{MakeCheckpointPermanent}$;

On receiving $\text{UndoTentativeCheckpoint}$:
    undo tentative checkpoint;
    for all $r \in \text{ckpt_cohort}_p$
        send $\text{UndoTentativeCheckpoint}$;
Rollback 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.
\[ \text{last_label_rcvd}_Y(X) > \text{last_label_sent}_X(Y) \]
(2) Any other process to whom I can send messages.
\[ \text{roll_cohort}_X = \{ Y \mid X \text{ can send message to } Y \} \]

Rollback Algorithm

Initiator Process \( P_i \):

\[
\text{for all } p \in \text{roll_cohorts}_{P_i}, \quad \text{send PrepareToRollback}(P_i, \text{last_label_sent}_{P_i}(p));
\]

if all cohorts replied “yes” then
\[
\text{for all } p \in \text{roll_cohorts}_{P_i}, \quad \text{send Roll_back message};
\]
else
\[
\text{for all } p \in \text{roll_cohorts}_{P_i}, \quad \text{send DoNot_Roll_Back message};
\]
Rollback Algorithm

A Cohort process, p:

On receiving PrepareToRollback(q, last_label_sent_q(p)):
  if willing_to_roll_p and
    last_label_rcvd(q) > last_label_sent_q(p) and resume_execution_p
    then resume_execution_p := false;
    for all r ∈ roll_cohort_p
      send PrepareToRollback(p, last_label_sent_p(r));
    if all r ∈ roll_cohort_p replied "yes"
      then willing_to_roll_p := yes;
    else willing_to_roll_p := no;
  send(p, willing_to_roll_p ) to q;

On receiving Roll_back AND if resume_execution_p = false:
  restart from p’s permanent checkpoint;
  for all r ∈ roll_cohort_p
    send Rollback;

On receiving DoNot_Roll_Back:
  resume execution;
  for all r ∈ roll_cohort_p
    send DoNot_Roll_Back;