SYNCHRONOUS AND ASYNCHRONOUS CHECK POINT AND RECOVERY ALGORITHMS

BY

Y.V.LOKESHWARI

&

K.NIVETHAA SHREE

AGENDA

INTRODUCTION

- Check-Pointing

  The process of saving state

- Checkpoint

  The recovery point at which check-pointing occurs

- Rolling Back

  The process of restoring a process to a prior-state

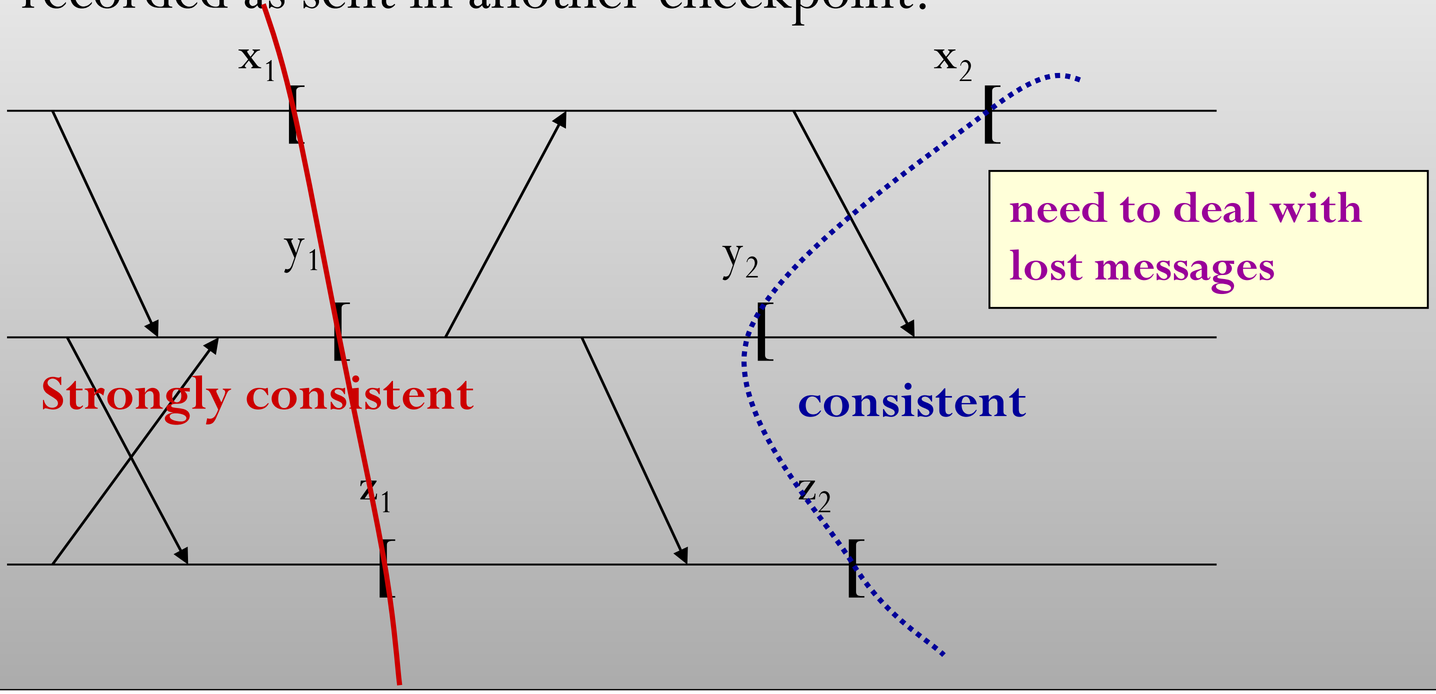
Consistency of Checkpoint

- Strongly consistent set of checkpoints

  No information flow takes place between any pair of processes in the set , during the interval spanned by the checkpoints

- Consistent set of checkpoints

  Each message recorded as received in a checkpoint should also be recorded as sent in another checkpoint.



Difference between Synchronous and Asynchronous Checkpoints

- ✓ **Synchronous Checkpoint**

  Set of all recent checkpoints are guaranteed to be consistent.

- ✓ **Asynchronous Checkpoint**

  Set of all recent checkpoints are not guaranteed to be consistent.

SYNCHRONOUS CHECK-POINTING AND RECOVERY

~Synchronous Checkpoint~

Goal

To make a consistent global checkpoint

Preliminary Assumptions

- Communication channels are FIFO
- No partition of the network
- End-to-end protocols cope with message loss due to rollback recovery and communication failure
- No failure during the execution of the algorithm
- The Checkpoint Algorithm assumes that single process invokes the Algorithm and not as several processes concurrently invoking the algorithm to take permanent checkpoint.

Preliminary (Two types of checkpoint)

~Synchronous Checkpoint~

Tentative checkpoint :

- a temporary checkpoint
- a candidate for permanent checkpoint

Permanent checkpoint :

- a local checkpoint at a process
- a part of a consistent global checkpoint

Checkpoint Algorithm

Algorithm **～Synchronous Checkpoint～**

First Phase

1. An initiating process P_i(a single process that invokes this algorithm) takes a tentative checkpoint
2. It requests all the processes to take tentative checkpoints
3. It waits for receiving from all the processes whether taking a tentative checkpoint has been succeeded
4. If it learns all the processes has succeeded, it decides all tentative checkpoints should be made permanent; otherwise, should be discarded.

Second Phase

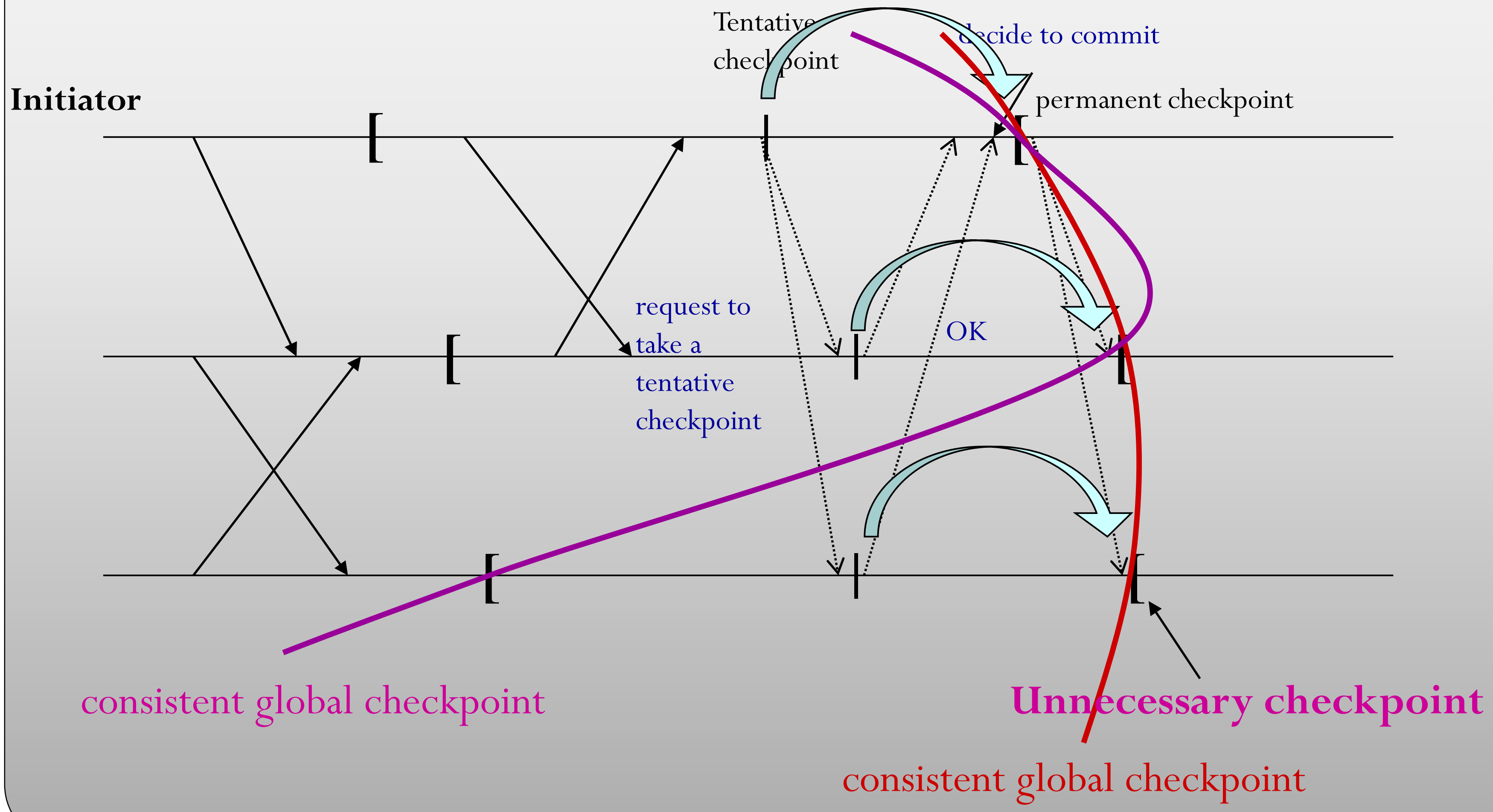
1. P_i It informs all the processes of the decision
2. The processes that receive the decision act accordingly

Supplement

Once a process has taken a tentative checkpoint, it shouldn't send messages until it is informed of initiator's decision.

Diagram of Checkpoint Algorithm

~Synchronous Checkpoint~



Tentative checkpoint
decide to commit
permanent checkpoint
Initiator
request to take a tentative checkpoint
OK
consistent global checkpoint
Unnecessary checkpoint
consistent global checkpoint

Optimized Algorithm

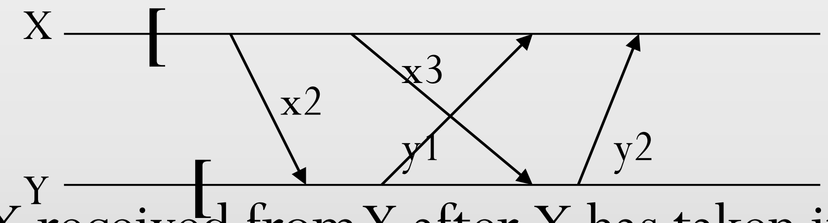
~Synchronous Checkpoint~

Each message is labeled by order of sending

Labeling Scheme

⊥ : smallest label

T : largest label



$last\_label\_rcvd_X[Y]$: y2

the last message that X received from Y after X has taken its last permanent or tentative checkpoint. if not exists, ⊥is in it.

$first\_label\_sent_X[Y]$: x2

the first message that X sent to Y after X took its last permanent or tentative checkpoint . if not exists, ⊥is in it.

$ckpt\_cohort_X$:

the set of all processes that may have to take checkpoints when X decides to take a checkpoint.

Checkpoint request need to be sent to only the processes included in *ckpt_cohort*

Optimized Algorithm

~Synchronous Checkpoint~

$ckpt\_cohort_X$: $\{ Y \mid last\_label\_rcvd_X[Y] > \perp \}$

Y takes a tentative checkpoint only if

$last\_label\_rcvd_X[Y] >= first\_label\_sent_Y[X] > \perp$

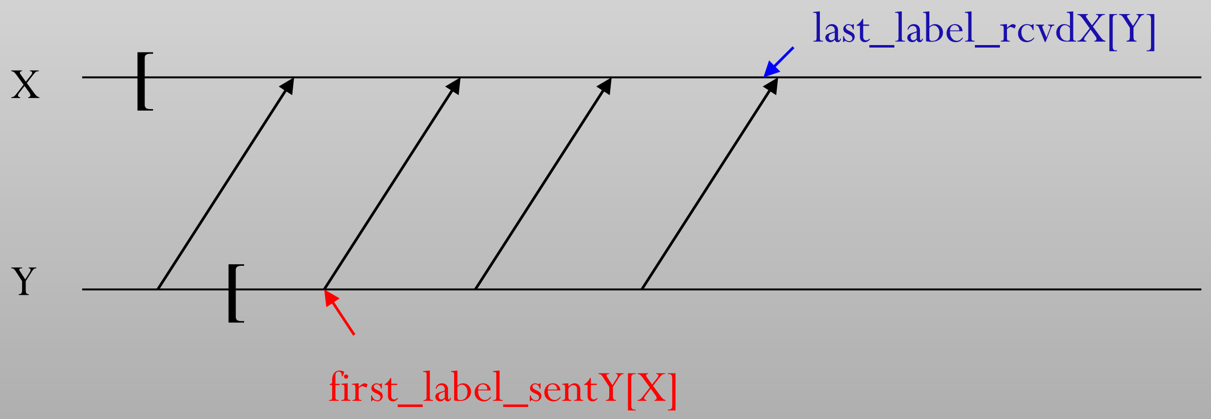
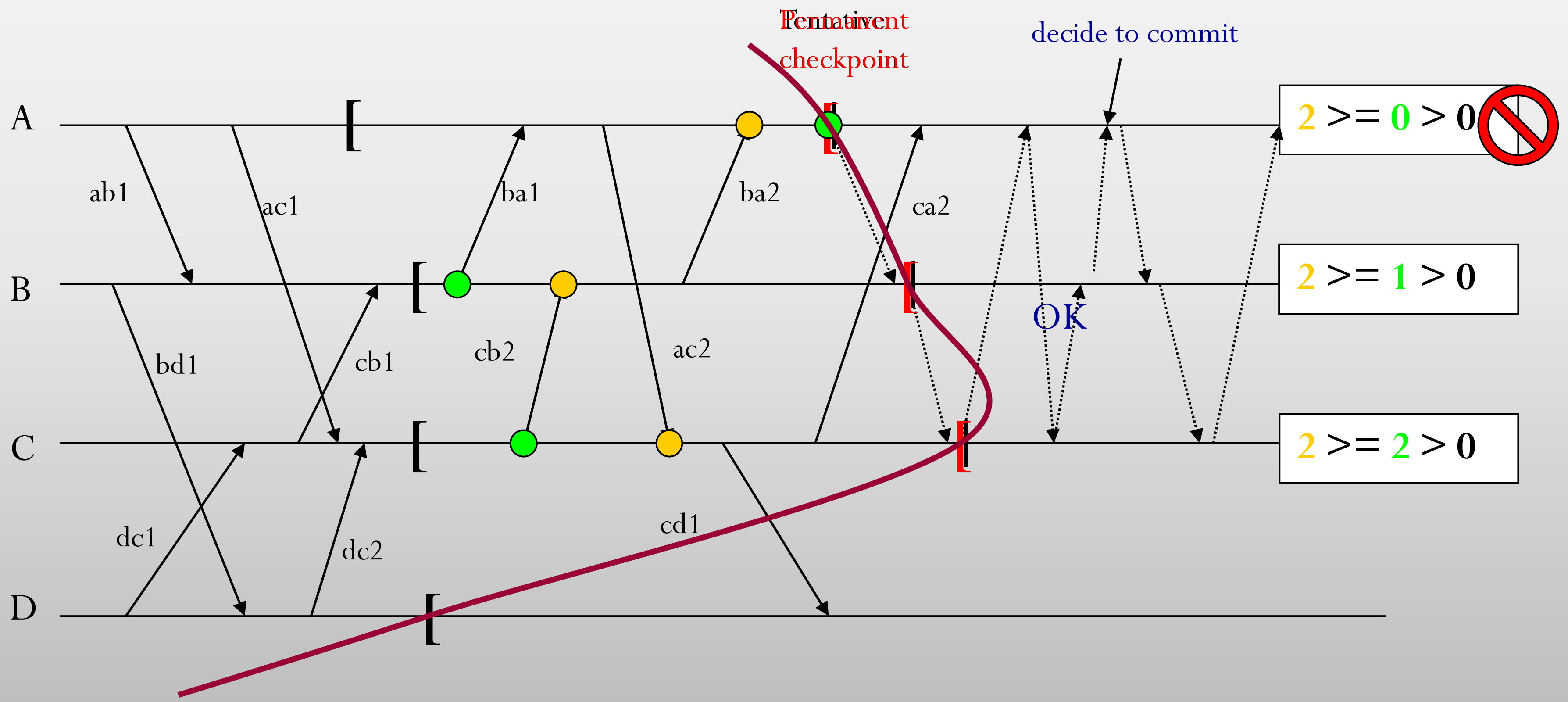


Diagram of Optimized Algorithm

～Synchronous Checkpoint～



ckpt_cohortX : {Y | last_label_rcvdX[Y] > ⊥ }

last_label_rcvdX[Y] >= first_label_sentY[X] > ⊥

Correctness

~Synchronous Checkpoint~

- A set of permanent checkpoints taken by this algorithm is consistent
  - No process sends messages after taking a tentative checkpoint until the receipt of the decision
  - New checkpoints include no message from the processes that don’t take a checkpoint
  - The set of tentative checkpoints is fully either made to permanent checkpoints or discarded.

The Rollback Recovery Algorithm

Preliminary Assumptions

- The Rollback Recovery algorithm assumes that a single process invokes the algorithm and not several processes concurrently invoking the Algorithm.
- The Checkpoint and Rollback Recovery algorithms are not concurrently invoked.

Two phases of Rollback Recovery Algorithm

First Phase

- An initiating process P_i checks to see if all the processes are willing to restart from their previous checkpoints.
- A process may reply "No" to restart request if it is already participating in a check-pointing or a recovery process is initiated by some other process.
- If P_i learns that all the processes are willing to restart from their previous checkpoints

  then

  P_i decides that all the process should restart.

  otherwise

  All the processes should continue their normal activities.

Phases contd..

Second phase

- P_iPropagates its decision to all processes.
- On receiving P_i ‘s decision, a process will act accordingly.
- The recovery algorithm requires that every process should not send messages related to underlying computation while it is waiting for P_i ‘s decision.

Correctness

- All Co-operating processes will restart from an appropriate state.
- All processes either restart from their previous checkpoint or continue with their normal operation
- If processes decide to re-start, then they all will resume execution in a consistent checkpoint.

Recovery Algorithm

~Synchronous Recovery~

Labeling Scheme

$\perp$: smallest label

T : largest label

$last\_label\_sent_X[Y]$:

The last message that X sent to Y before X takes its latest permanent checkpoint. If not exist, **T** is in it.

$last\_label\_recvd_Y[X]$:

The last message that Y received from X after X took its last permanent or tentative checkpoint . If not exists, $\perp$is in it.

When X request Y to restart from permanent checkpoint, it sends $last\_label\_sent_X[Y]$ along with its request.

Recovery Algorithm

~Synchronous Recovery~

Y will restart from the permanent checkpoint only if

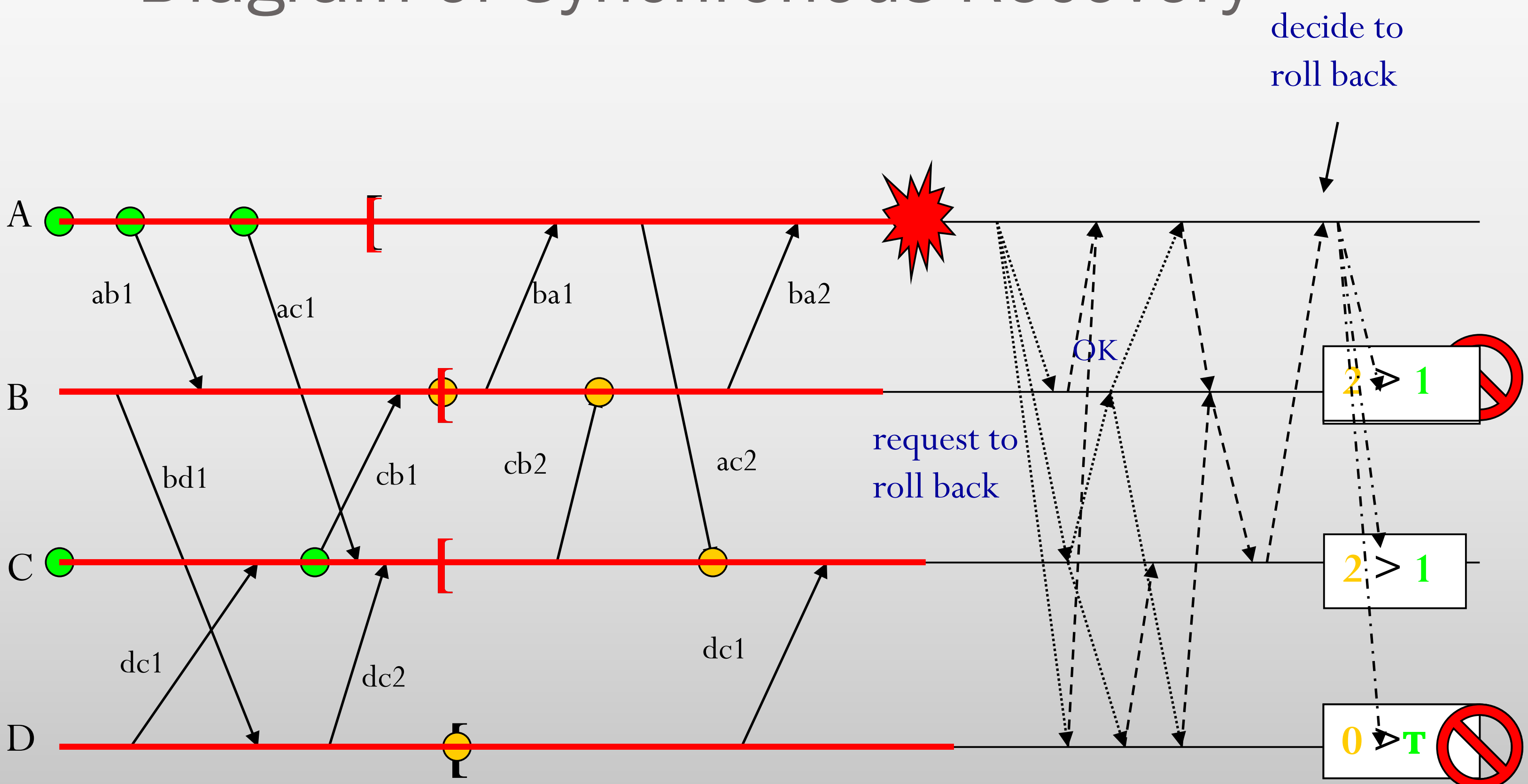
$last\_label\_rcvd_Y[X] > last\_label\_sent_X[Y]$

$roll\_cohort_X$:

The set of all processes that may have to roll back to the latest checkpoint when process X rolls back.

$roll\_cohort_X = \{ Y \mid X \text{ can send messages to } Y \}$

Diagram of Synchronous Recovery



roll_cohortX = { Y | X can send messages to Y }

last_label_rcvdY[X] > last_label_sentX[Y]

Asynchronous Checkpoint/Recovery Algorithm

Synchronous Approach

- It simplifies recovery
  - Since consistent set of checkpoints readily available.
- Demerits
  - Additional messages are exchanged by checkpoint algorithm when it takes checkpoint.
  - Synchronization delays occurs.

    - No computational message can be sent while checkpoint algorithm is in progress.

Asynchronous Approach

Characteristic:

- Each process takes checkpoints independently without any synchronization among process.
- No guarantee that a set of local checkpoints is consistent.
- A recovery algorithm has to search consistent set of checkpoints before recovery initiated.

- No additional message
- No synchronization delay

Asynchronous Checkpoint (Message logging)

- To minimize amount of computation undone during recovery , all incoming message logged at each processor.
- Message received can be logged in two ways:
- Pessimistic message logging:
  - Incoming message is logged before it is processed.

Asynchronous Checkpoint(contd.)

- Optimistic message logging:
  - Processors continue to perform computation and message received are stored in volatile storage , logged at certain intervals.
  - In system failure , incoming message lost as it may not have been logged.

Asynchronous Checkpoint (contd.)

- Comparison:
  - During rollback , amount of computation redone during recovery more in system that use optimistic logging when compared to system tat use pessimistic logging.

Two types of log

~Asynchronous Checkpoint / Recovery~

- Two types of log storage, volatile and stable log.
- Volatile log:
  - Access time less.
  - Contents are lost if processor fails.
  - Periodically flushed to stable storage and cleared.
- Stable log:
  - Slow access.
  - Not lost even if processors fail.

Record events

- Each processor, after event, records triplet{s,m,msg_sent} in volatile storage.
- S is state of the processor before the event.
- m is message whose arrival caused the events.
- msg_sent is the set of messages that were sent by processor during event.
- Local checkpoint at each processor consist of the record of an event occurring at processor
- Taken without any synchronization with other processors.

Preliminary (Assumptions)

～Asynchronous Checkpoint / Recovery～

- Assumptions
  - Communication channels are FIFO
  - Communication channels are reliable
  - Communication channel have infinite buffers.
  - Message transmission delay is arbitrary, but finite.

Preliminary (Notations)

～Asynchronous Checkpoint / Recovery～

Definition

$CkPt_i$: the checkpoint (stable log) that i rolled back when failure occurs

$RCVD_{i\leftarrow j}(CkPt_i)$:
the number of messages received by processor i from processor j, per the information stored in the checkpoint $CkPt_i$

$SENT_{i\rightarrow j}(CkPt_i)$:
the number of messages sent by processor i to processor j, per the information stored in the checkpoint $CkPt_i$

Recovery Algorithm

~Asynchronous Checkpoint / Recovery~

- Each processor keeps track of number of messages it has sent to other processor as well as number of messages it has received from other processor.

- When rollback occurs, other processor find out whether any message previously sent are orphan message.
- Discovered by comparing number of message sent and received.
- If number of message received is greater than number of message sent, it indicates orphan message.
- Processor have to rollback to state where number of messages received agrees with number of messages sent.

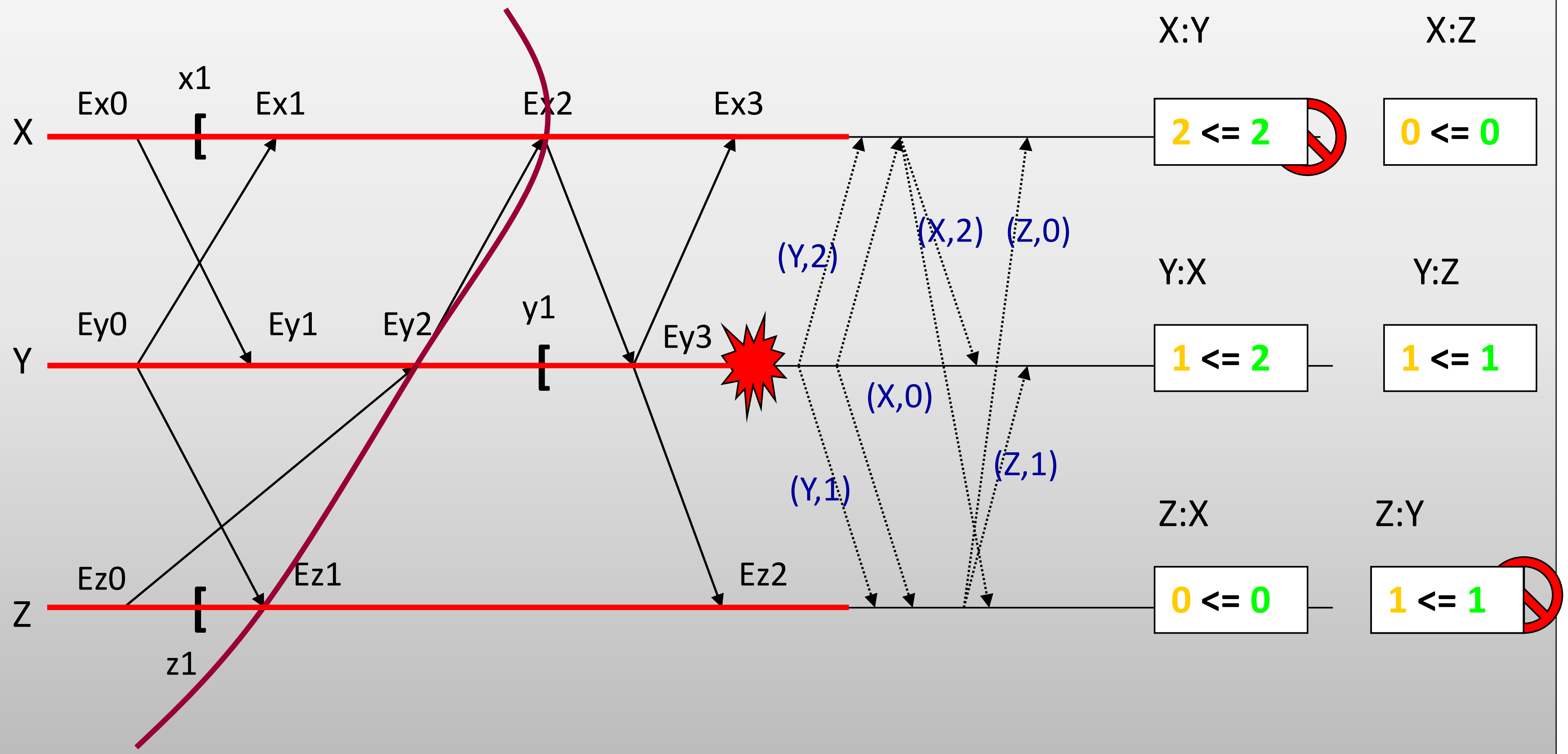
Recovery Algorithm

```
If i is a processor that is recovering after failure then
        $CkPt_i$ =latest event logged in the stable storage
else
    $CkPt_i$ =latest event that took place in it
for k=1 to N do
    begin
    for each neighboring processor j do
            Send ROLLBACK(I, $SENT_{i \rightarrow j}(CkPt_i)$)message
    wait for ROLLBACK message from every neighbor.
```

Recovery Algorithm(contd.)

```
for each ROLLBACK(j,c)message received from a neighbor
  j
                i does following
  if  $RCVD_{i\leftarrow j}$ (CkPt$_i$) >c then
                (inplies presence of orphan message)
                begin
                find the latest event e such that  $RCVD_{i\leftarrow j}$ (e)=c
                 CkPt$_i$ = e;
                end;
  end(* for k*)
```

Asynchronous Recovery



RCVDi←j (CkPti) <= SENTj→i(CkPtj)

QUERIES???

THANK YOU