Checkpointing And Rollback Recovery Techniques For A Distributed System

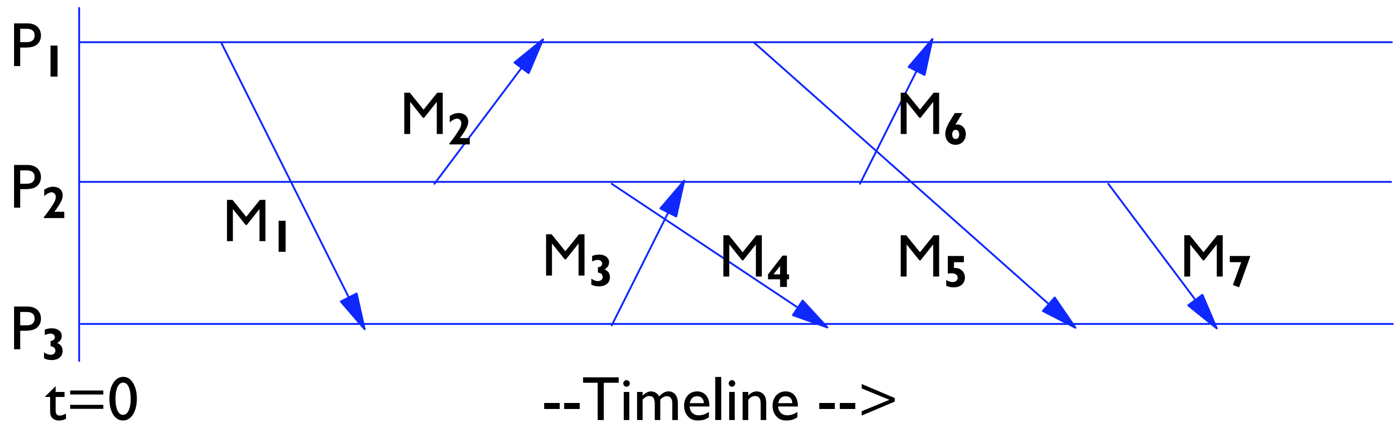
Preetha Natesan

Presentation Overview

- Distributed System
- Checkpointing Concepts
- Message Logging
- Rollback Recovery
- Quasi-Synchronous Algorithm (QSA)
  - Checkpointing
  - Basic Recovery
- Message Classification
- Comprehensive Recovery in QSA
- Conclusion

Distributed System

- Multiple processes
- States of processes depend on one another due to inter-process communication
  - Messages are sent/received between processes



Checkpointing Concepts

- **Definition**
  - saving of program state, usually to a stable storage
  - useful for reconstructing at a later time
- **Classification**
  - ASynchronous
    - checkpoints taken periodically w/o coordination
    - allows maximum process autonomy
    - low checkpointing overhead
    - suffers from Domino Effect
  - Synchronous
    - processes synchronize their checkpointing activities
    - globally consistent set of checkpoints maintained
    - Domino Effect free
    - no process autonomy
    - performance degradation
  - Communication Induced or Quasi-Synchronous
    - Checkpointing activity is partially synchronized
    - Easeness and low overhead of asynchronous checkpointing
    - Recovery time advantages of synchronous checkpointing

Message Logging

- **What is it?**
  - Generally used along with checkpointing
  - Restores the system to a consistent state in case of a failure
- **Classification**
  - Pessimistic
    - Received messages are stored in stable storage before being processed
    - Helps faster recovery
    - Performance Degradation
  - Optimistic
    - Received messages are stored in volatile storage ; periodically flushed to stable storage during idle time
    - Messages stored in volatile storage lost during failure
    - This can cause repeated rollback

Rollback Recovery

- Definition
  - Finding a consistent global snapshot from previously saved checkpoints of the processes and restarting from that state
- Goal of a good rollback recovery technique
  - Minimize the computation due to rollback

Some salient features

- Checkpointing provides the backbone for :
  - rollback recovery (fault tolerance)
  - playback debugging
  - process migration
  - job swapping
- Checkpointing and rollback recovery enable a system to :
  - tolerate failures by periodically saving the entire state and rolling back to the saved state if an error is detected
- Rollback recovery using checkpointing is :
  - a cost effective method of proving fault tolerance against transient and intermittent faults

System Model

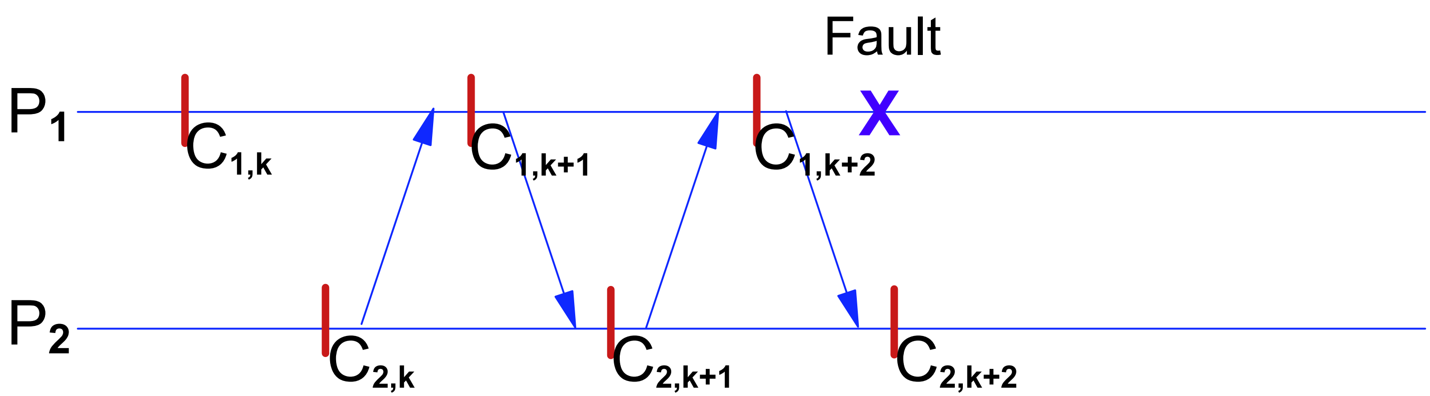
- A distributed system consists of N sequential processes [P1, P2, P3....PN].
- The concurrent execution of all processes on a network of processors is called a distributed computation.
- Message passing is the only way for processes to communicate with one another.
- No assumption is made on the FIFO nature of the channel.
- The local state of a process saved in the stable storage is called a checkpoint of the process.

Definitions and Notations

- Each Checkpoint (C) of a process is assigned a unique sequence number -denoted by C.sn.
- Each message (M) is piggybacked with the sequence number(M.sn) of the latest checkpoint of the process sending it.
- The checkpoint with sequence number m of Process Pi is denoted by Ci,m.
- Basic checkpoint
  - Independently taken by a process
- Forced checkpoint
  - Checkpoint triggered by a message reception
- Consistent Global Checkpoint
  - A set of local checkpoints. one from each process is called a consistent global checkpoint if none of them is causally dependent on any other checkpoint in the set

Domino Effect

- The fault causes process P_1 to roll back to checkpoint $C_{1,k}$ and process P_2 to roll back to checkpoint $C_{2,k}$



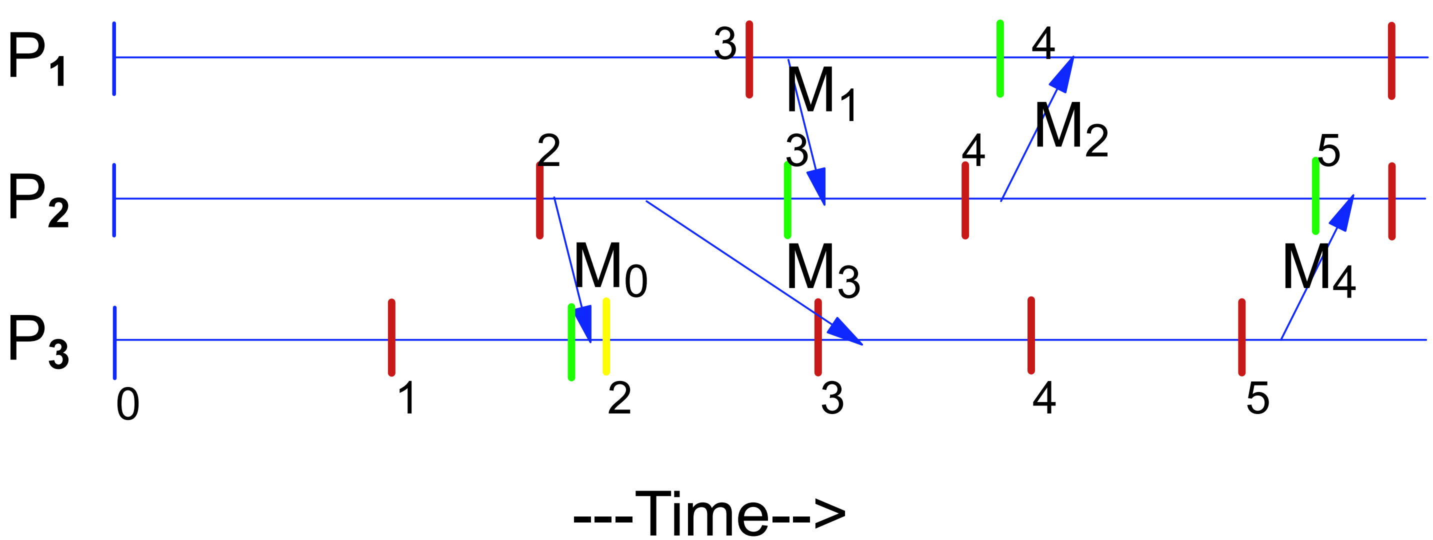
A Quasi-Synchronous Checkpointing Algorithm (QSA)

- Each process Pi has two variables
  - SN.i - seq. # of latest checkpoint taken by Pi; initialized to 0
  - Next.i - seq # to be assigned to the next basic checkpoint of Pi; initialized to 1
- Next.i is incremented by Pi every x time units
- When it is time to take a basic checkpoint
  - If Next.i > SN.i
    - Take checkpoint Ci; /* Basic checkpoint */
    - Ci.sn = Next.i; SN.i = Ci.sn
  - else
    - Skip the checkpoint
- When Pi sends a message M
  - M.sn = SN.i; /* piggyback M with sequence number of current checkpoint */
  - send(M)
- When process Pj receives a message M from Pi,
  - If M.sn > SN.j
    - Take checkpoint Cj; /* Forced checkpoint */
    - Cj.sn = M.sn; SN.j = Cj.sn
  - Process the message

QSA Example

Legend

- Basic chkpt (red) - Forced chkpt (green) - Schd. basic chkpt (yellow)



* The numbers in the chart are the chkpt seq. numbers

*** When its time for a process to take a basic chkpt, it takes a basic chkpt only if it did not already take a forced chkpt with the seq. # that is expected to be assigned to the next basic chkpt; otherwise it skips taking the basic chkpt.**

Basic Recovery Algorithm

- Assumptions
  - If a process fails, then no other process fails until the system is restored to a fully consistent state
  - The recovery algorithm is fully asynchronous
- Each process Pi has two more variables
  - inc.i - Incarnation number of process Pi ; initialized to 0
  - rec_line.i - Recovery line number; initialized to 0
- When Pi sends message M
  - M.rec_line = rec_line.i   /* piggy back M with current recovery line number */
  - M.inc = inc.i   /* piggy back M with current incarnation number */
- When process Pj receives message M
  - If M.inc > inc.j   /* possible if M was sent by a process that has */
    - rec_line.j = M.rec_line;   /* already rolled back based on a failure;   */
    - inc.j =M.inc;   /* in that case, do not process M; Roll_back(Pj)   */
    - Roll_back(Pj);

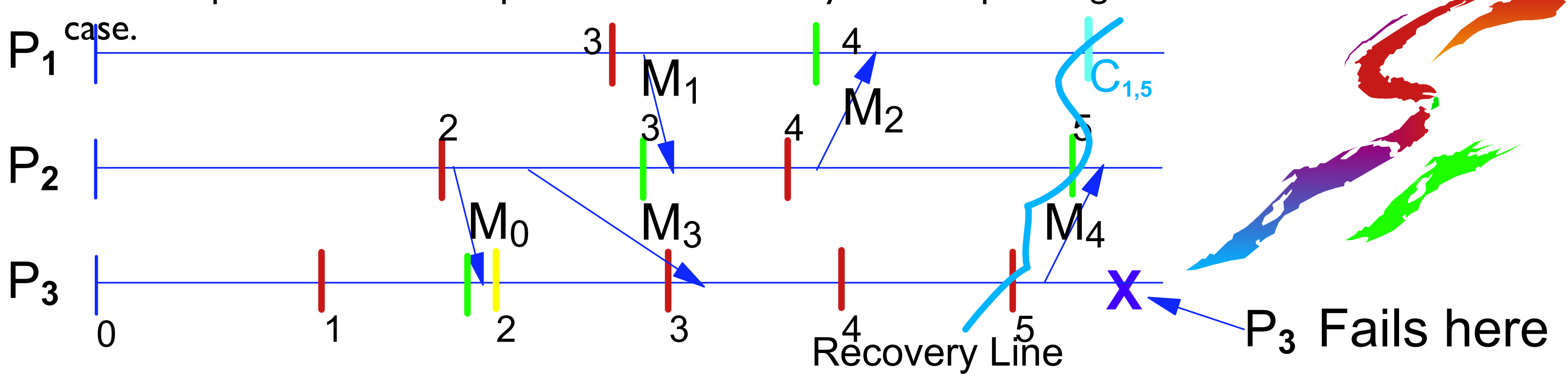
Basic Recovery Algorithm (contd)

- When a process Pi fails
  - Restore the latest checkpoint
  - Increment inc.i ; rec_line.i = SN.i
  - send roll_back(inc.i,rec_line.i) to all other processes
  - continue normally
- When process Pj receives roll_back(inc.i,rec_line.i) from Pi
  - If inc.i > inc.j /* otherwise, **if inc.i = inc.j**, Pj is aware of this */
    - inc.j = inc.i /* recovery through a msg sent by some other */
    - rec_line.j = rec_line.i /* process that has already rolled back; hence, */
    - Roll_back(Pj) /* no need for roll back in that case */
  - continue normally

- Procedure Roll_back(Pj)
  - If rec_line.j > SN.j
    - No need to roll back;
    - take a new checkpoint which can be part of the recovery line
  - else
    - roll back to the earliest checkpoint C with C.sn >= rec_line.j
    - restore checkpoint C
    - Delete all the checkpoints beyond C

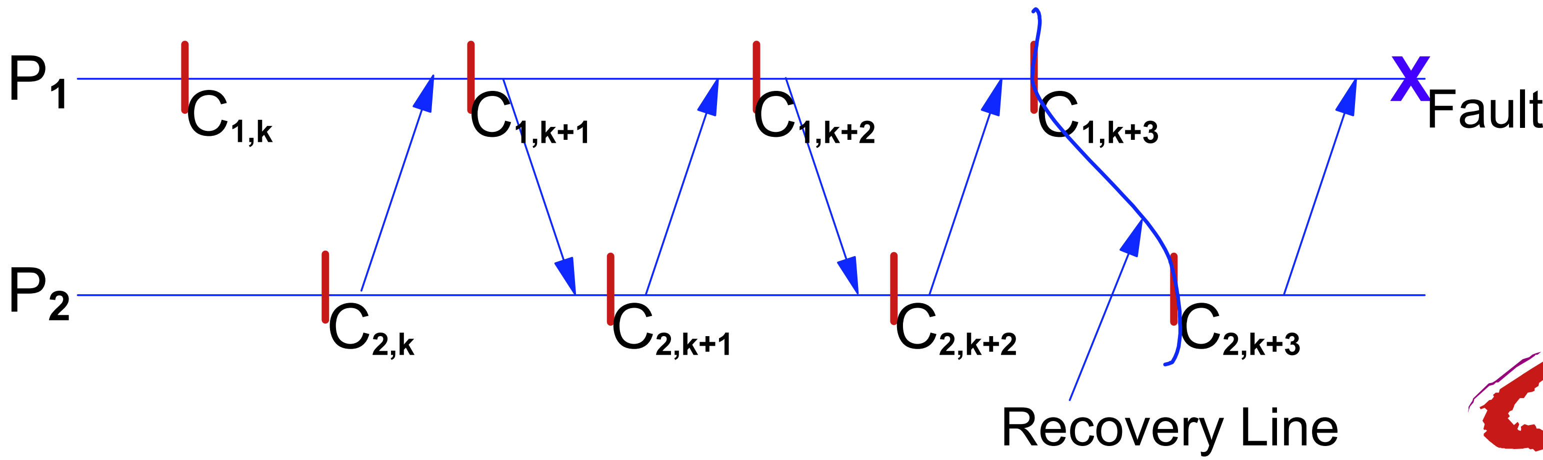
Basic Recovery Algorithm - An example

- Consider failure of P_3 , as shown below
- Steps taken by P_3
  - increment inc.3 to 1 /* it was initially zero */
  - set rec_line.3 to 5 /* the seq. # of last checkpoint */
  - roll back to latest checkpoint $C_{3,5}$
  - send rollback(1,5) to processors P_1 and P_2
- Steps taken by P_2
  - roll back to $C_{2,5}$ /* since it is the earliest chkpt whose seq. # >=5 */
- Steps taken by P_1
  - Take checkpoint ($C_{1,5}$) of the current state /* since it does not have chkpt whose seq. #>=5 */
  - Assign seq # 5 to the checkpoint taken
- Thus, {$C_{1,5}$, $C_{2,5}$, $C_{3,5}$} will be the recovery line for this failure
- Note: Seq # of all the checkpoints in the recovery line is equal. In general, that need not be the case.



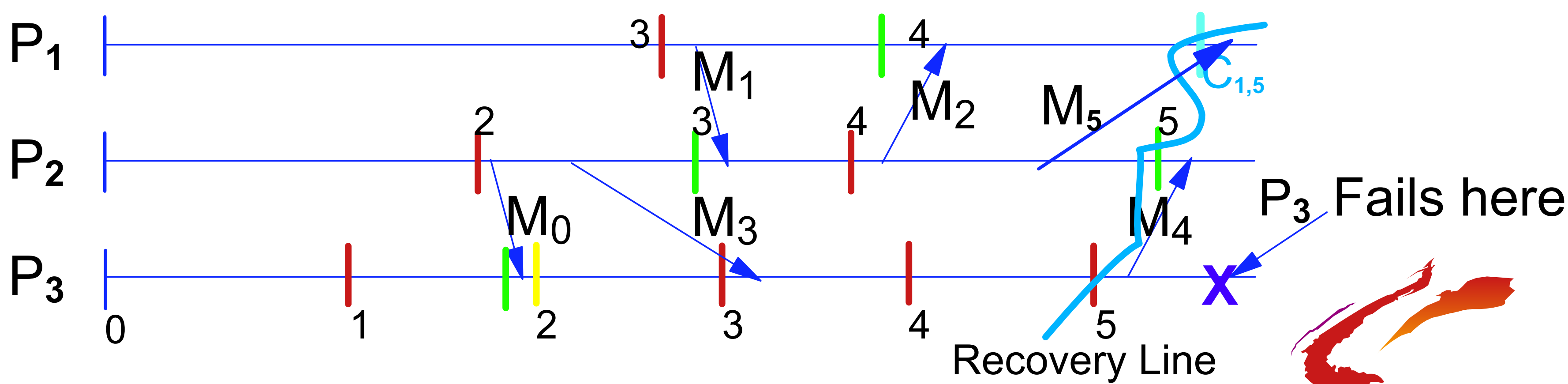
Basic Recovery Algorithm - Domino Effect Free

- The fault causes process P_1 to roll back to checkpoint $C_{1,k+3}$ and process P_2 to roll back to checkpoint $C_{2,k+3}$



Basic Recovery Algorithm - Analysis

- This algorithm guarantees that processes roll back to a consistent global checkpoint in the event of a failure
- As a result of rollback,
  - the reception of some messages might be undone while the corresponding send event might not have been undone, (message M_5 in the figure)



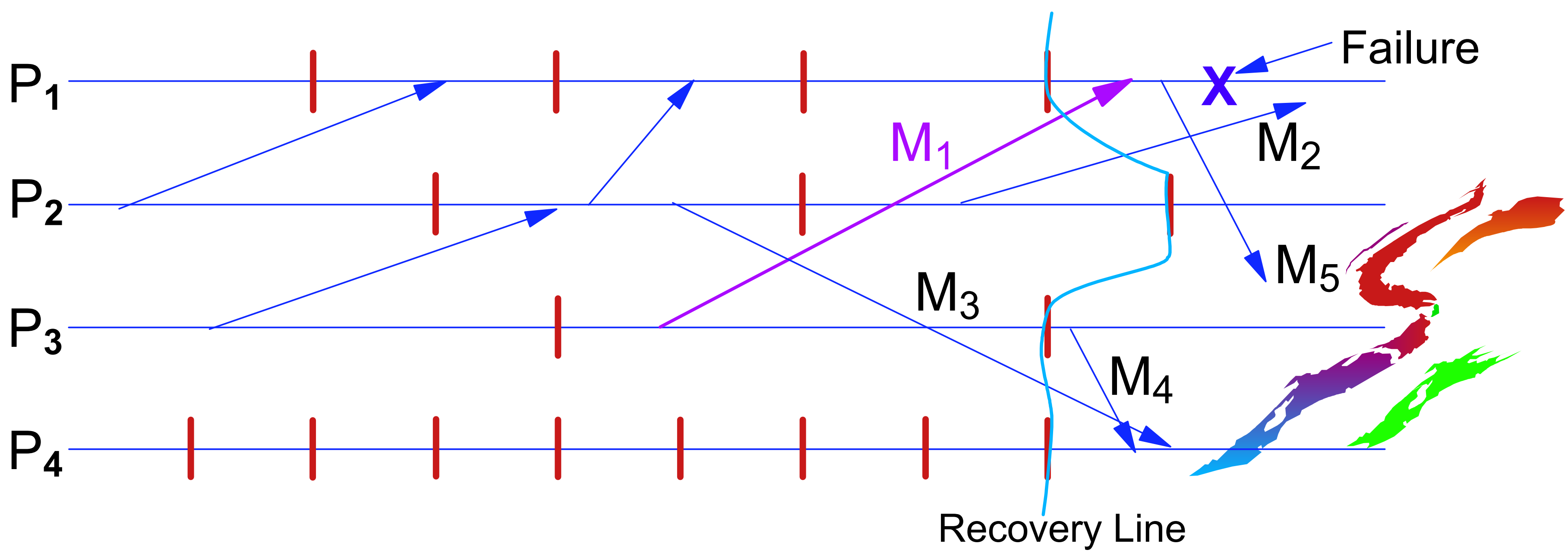
- So, even though the processes roll back to a consistent global checkpoint, it may not leave the system in a consistent state

Comprehensive Recovery

- Modify the Basic Recovery algorithm to restore the system to a consistent state after rolling back the processes to a consistent global checkpoint
- Rollbacks could result in undoing the send and/or receive events of many messages
- This may result in several abnormal situations
- These should be dealt correctly in order to restore the system to a consistent state
- Different types of messages need to be handled

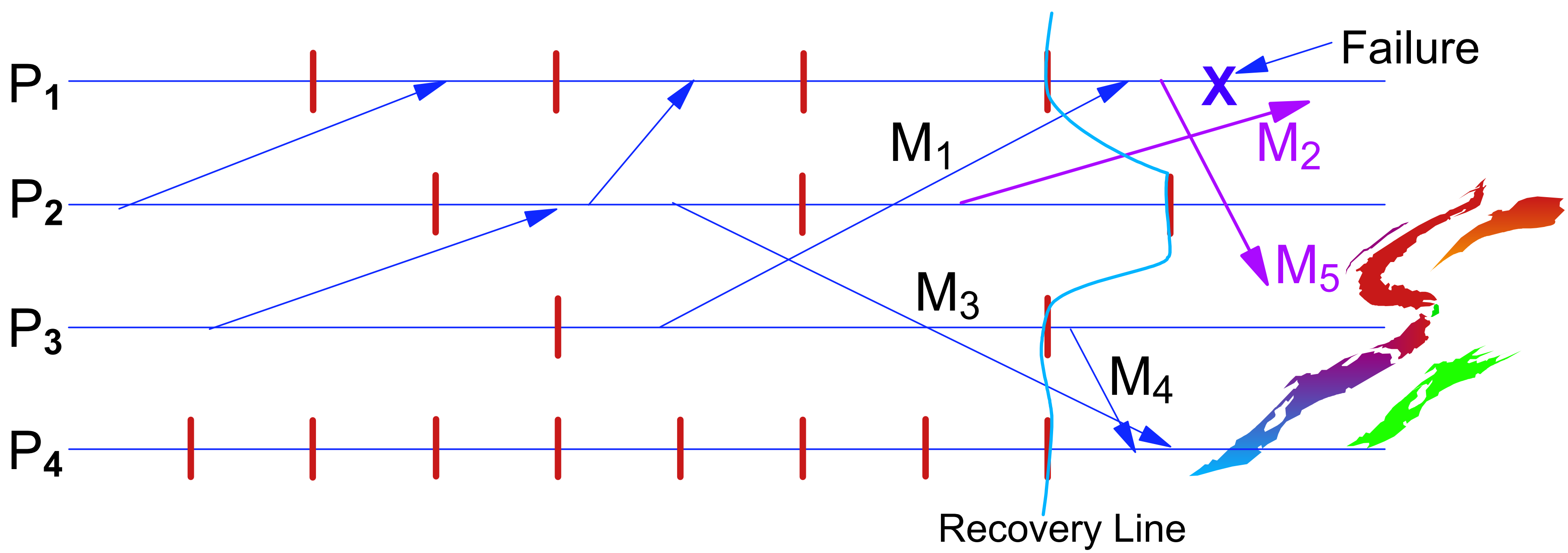
Message Classification

- Lost Messages
  - Messages whose send events are not undone but whose receive events are undone due to rollback
  - Arises when a process rolls back to a chkpt prior to the reception of the msg while the sender does not rollback to a chkpt prior to the send event
  - In the figure, M_1 is a lost message



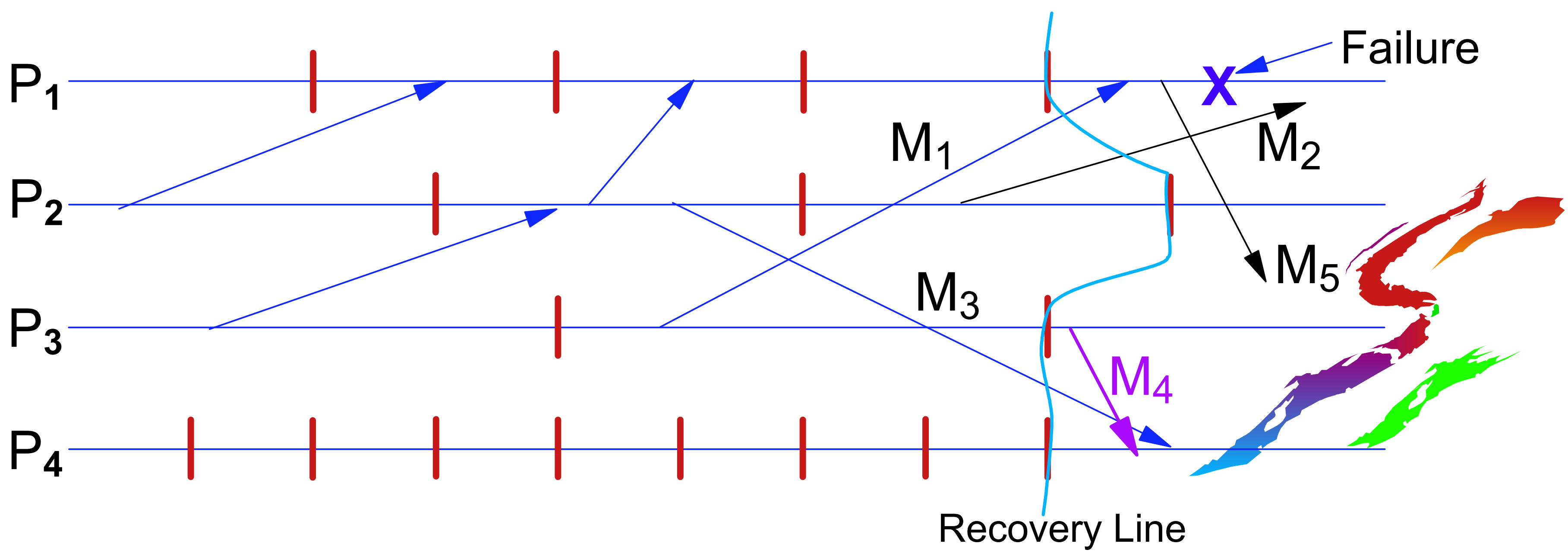
Message Classification

- Delayed Messages
  - Messages that were sent before the rollback whose receive events were not recorded
  - Arises when messages were received while the receiving process was down or received after the rollback of the receiving process
  - In the figure, M_2 and M_5 are delayed messages



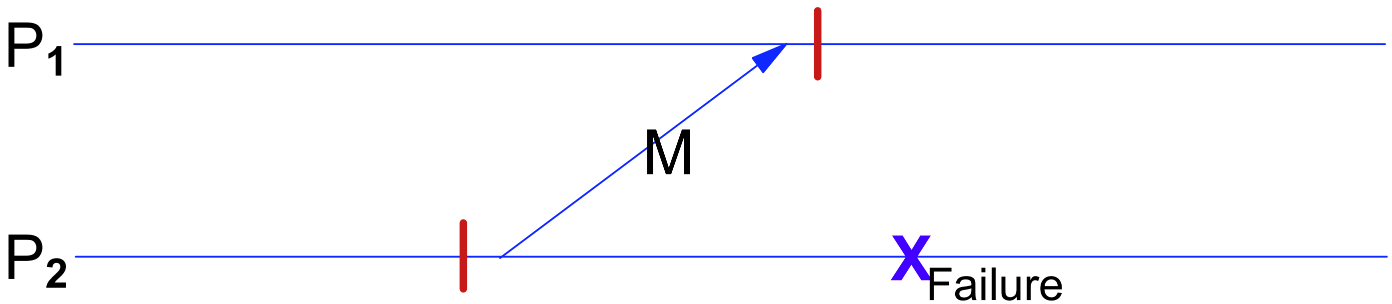
Message Classification

- Duplicate Messages
  - Happens due to message logging and replaying them during recovery
  - In the figure, message M_4 was sent and received before the rollback
  - Due to rollback, P_4 undoes the receive of M_4 and P_3 undoes the send of M4
  - If P_4 replays M_4, then M_4 will be a duplicate message since P_3 will resend M_4



Message Classification

- Orphan Messages
  - Messages which have been received and whose send has been undone due to rollback, but whose receive has not been undone
  - Orphan messages do not arise if processes roll back to a consistent global checkpoint
  - So, the Basic Recovery Algorithm does not have problems with orphan msgs
  - In the figure, message M is an orphan message



Comprehensive Recovery - Message Handling

- Goal
  - Identify the minimal set of messages that need to be logged to and replayed from the message log
- Proposed Approach
  - Need to handle only delayed messages that are received after a failed process recovers, lost messages and duplicate messages
  - This is accomplished by allowing processes to
    - log received messages selectively and
    - replay logged messages selectively after a rollback

Comprehensive Recovery - Message Handling : Replay Rule

- When a process Pj rolls back to a checkpoint C, it replays a message M from its message log if and only if M was received after the checkpoint C was taken and **M.sn < rec_line.j**
  - This means that Pj must replay all those messages whose receive was undone but whose send will not be undone
  - In other words, Pj must replay **only** those messages that **originated to the left** of the current recovery line and **delivered to the right** of the current recovery line

Comprehensive Recovery - Message Handling : Logging Rule

- Suppose Pj receives a message M from Pi.
- If Pj is replaying messages as a result of a rollback
  - Buffer message M
  - Process it only after finishing replaying
- Otherwise
  - If M is a delayed message (M.inc < inc.j) , process it only if M.sn < rec_line.j; else discard message M
  - Log the message M before processing it if
    - M.inc < inc.j && M.sn < rec_line.j OR
    - M.inc = inc.j && M.sn < SN.j
    - If M.inc > inc.j, then from the algorithm we set inc.j = M.inc and Pj rollbacks. The message is then handled as in the previous case

QSA
- Merits

- The QSA checkpointing algorithm
  - has the easness of asynchronous checkpointing and the advantages of synchronous checkpointing
  - guarantees the existence and progression of a recovery line consistent with the latest checkpoint of each process
  - has no additional control message overhead and it has nominal checkpointing overhead
- The QSA comprehensive recovery algorithm
  - uses the recovery line to restore the system to a consistent state asynchronously, in the case of a single process failure
  - has a low recovery overhead since messages are logged and replayed selectively
  - does not involve an explicit synchronization overhead
  - does not suffer from domino-effect

Conclusion

- The talk focussed on checkpointing and recovery for a distributed computing system
- Gave an overview of the various concepts
  - Checkpointing, Rollback Recovery and Message logging
- Presented the Quasi-Synchronous Algorithm consisting of
  - Checkpointing
  - Basic Recovery
  - Comprehensive Recovery
- Showed examples to explain the QSA algorithm
- Analyzed the different types of messages and how the comprehensive recovery technique handled them

References

- D.Mannivannan and M. Singhal. "A Low-Overhead Recovery Technique Using Quasi-Synchronous Checkpointing". Proc. IEEE 16th Int'l Conf. Distributed Computing Systems, pp 100-107 HongKong, May1996

- F.Quaglia, B.Ciciani and R.Baldoni. "A Checkpointing-Recovery Scheme for Domino-Free Distributed Systems". IEEE Annual Workshop on Fault -Tolerant Parallel and Distributed Systems, Geneva, April 1997

- D.Mannivannan and M. Singhal. "Quasi-Synchronous Checkpointing: Models, Charasterization, abd Classification". IEE Transactions on Parallel and Distributed Systems, Vol.10, No. 7, July 1999.

Thank You