Chapter#9

Fault Tolerance

Fault->Error->Failure

Bishnu Hari Paudel
@IOE WRC, Pokhara

Introduction

- **Fault-** a defect, imperfection, weakness, mistake

  Fault Types : Transient -appear once and disappear

  Intermittent- appear-disappear-reappear

  Permanent - appear and persist until repaired

  (Node fault, Timing fault, Program/Process fault, Communication/Network fault)

- **Fault Tolerance-** Fault tolerance is the way in which an operating system (OS) responds even if a hardware or software failure. To handle faults gracefully, some computer systems have two or more duplicate systems.

  i.e. Fault tolerant systems use redundancy to ensure its operation continuity after a system failure.

- A system is said to be k-fault tolerant if it is able to function properly even if k nodes of the system suffer from concurrent failures.

Dependability (four Properties)

Availability: A measurement of whether a system is ready to be used immediately. System is up and running at any given moment.

Reliability: A measurement of whether a system can run continuously without failure. System continues to function for a long period of time.

Safety: A measurement of how safe failures are. System fails, nothing serious happens.

Maintainability: A measurement of how easy it is to repair a system. Failures can be detected and repaired automatically? Self-healing systems.

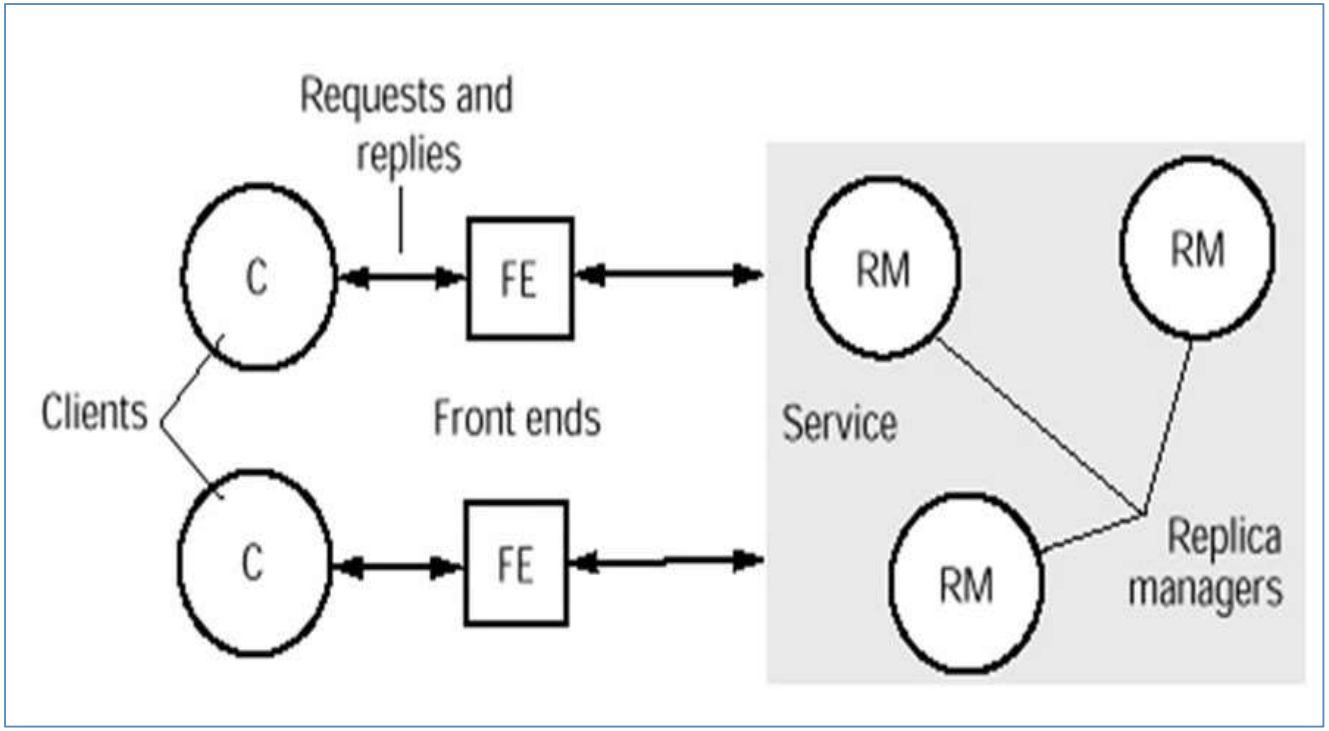
Replication

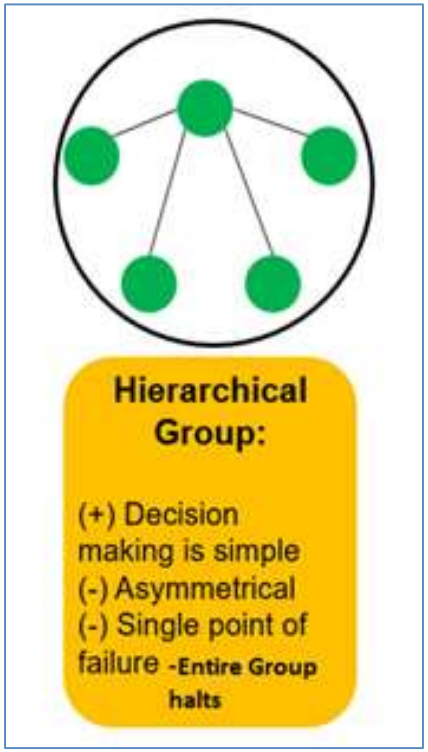
Passive Replication

At any point of time, there is a single primary replica manager and one or more secondary replica managers or backups.

The front end of the system is able to communicate only with the primary replica manager to obtain the service.

The primary replica manager executes the operations and sends updated data to the backups.

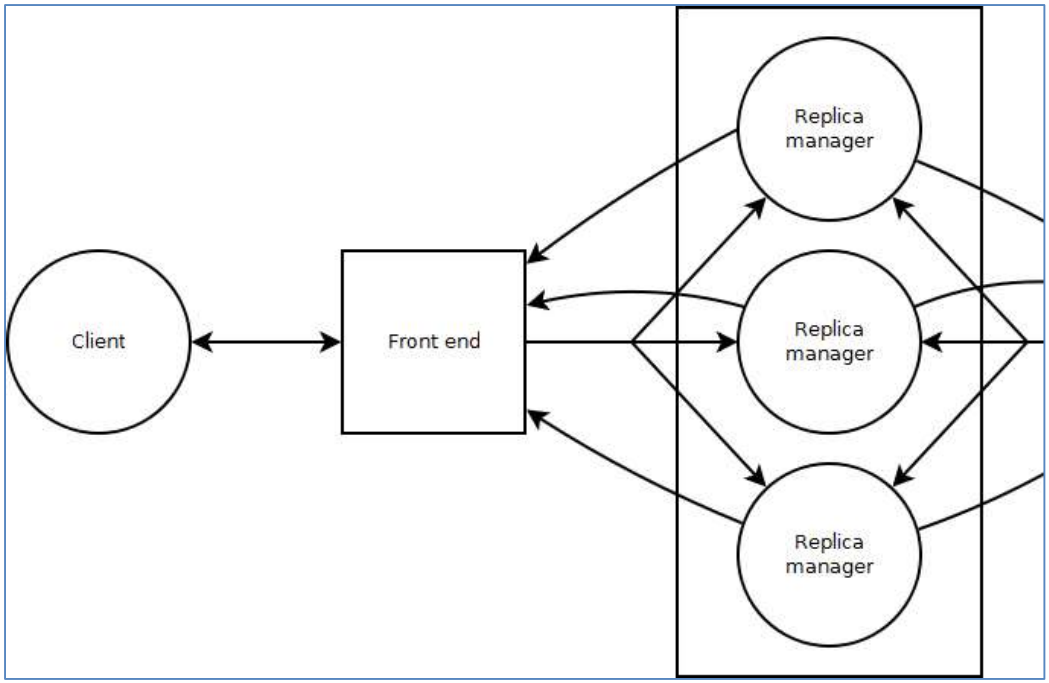


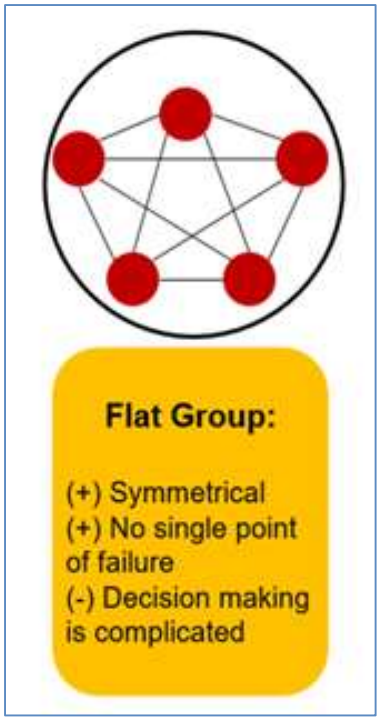


Replication

- **Active Replication:**

  The replica managers are state machines that play equivalent roles.
  FE multicast requests to the group of RM and all of them process the requests independently but identically and reply.
  If any RM crashes, others RM can responds. So, there is no degradation on performance.





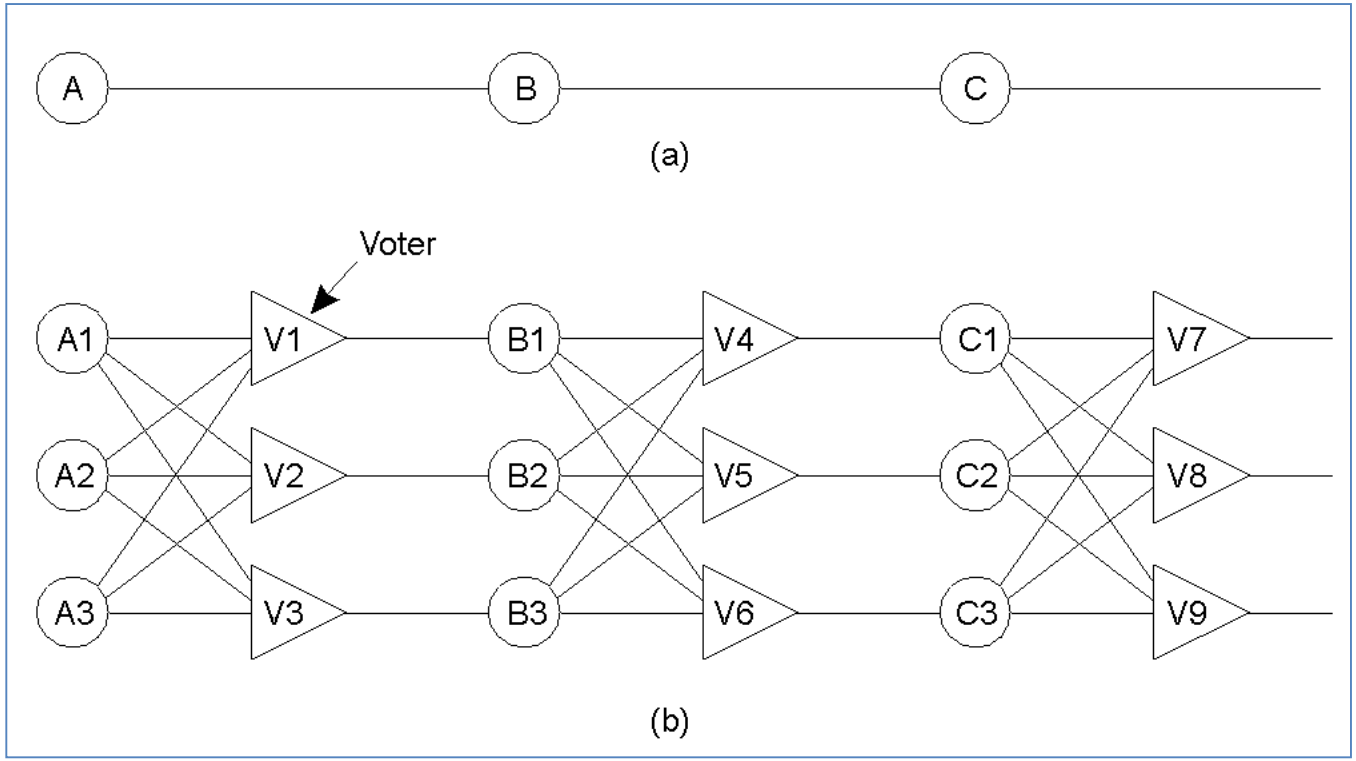
Flat and Hierarchical Groups

- In flat groups, all the processes within a group have equal roles.
  Control is **completely distributed** to all the processes.
  As information is exchanged immediately, it is good for fault tolerance.
  It imposes more overhead.
  It is difficult to implement.

- In hierarchical group, all the communications are handled by a single process designated as **coordinator.**
  It is not completely fault tolerant and scalable.
  It is easy to implement.

Failure Masking

- Key- Redundancy
  - Information/Data Redundancy
  - Process/Time Redundancy
  - Physical Components Redundancy



Replication Number?

- Replicate Process and organize them into groups
- Replace a single vulnerable process with the whole fault tolerant Group
- A system is said to be K fault tolerant if it can survive faults in K components and still meet its specifications.
- How much replication is needed to support K Fault Tolerance?
  - ❖ K+1 or 2K+1 ?

Agreement

- Goal of Agreement
  - Make all the non-faulty processes reach consensus on some issue
  - Establish that consensus within a finite number of steps.

- A process group typically requires reaching an agreement in:
  - Electing a coordinator
  - Deciding whether or not to commit a transaction
  - Dividing tasks among servers
  - Synchronization

Process Resilience

- Processes can be made fault tolerant by arranging to have a group of processes, with each member of the group being *identical* .
- A message sent to the group is delivered to all of the “copies” of the process (the group members), and then *only one* of them performs the required service.
- If one of the processes fail, it is assumed that one of the others will still be able to function (and service any pending request or operation.

Process Resilience

When the communication and processes:

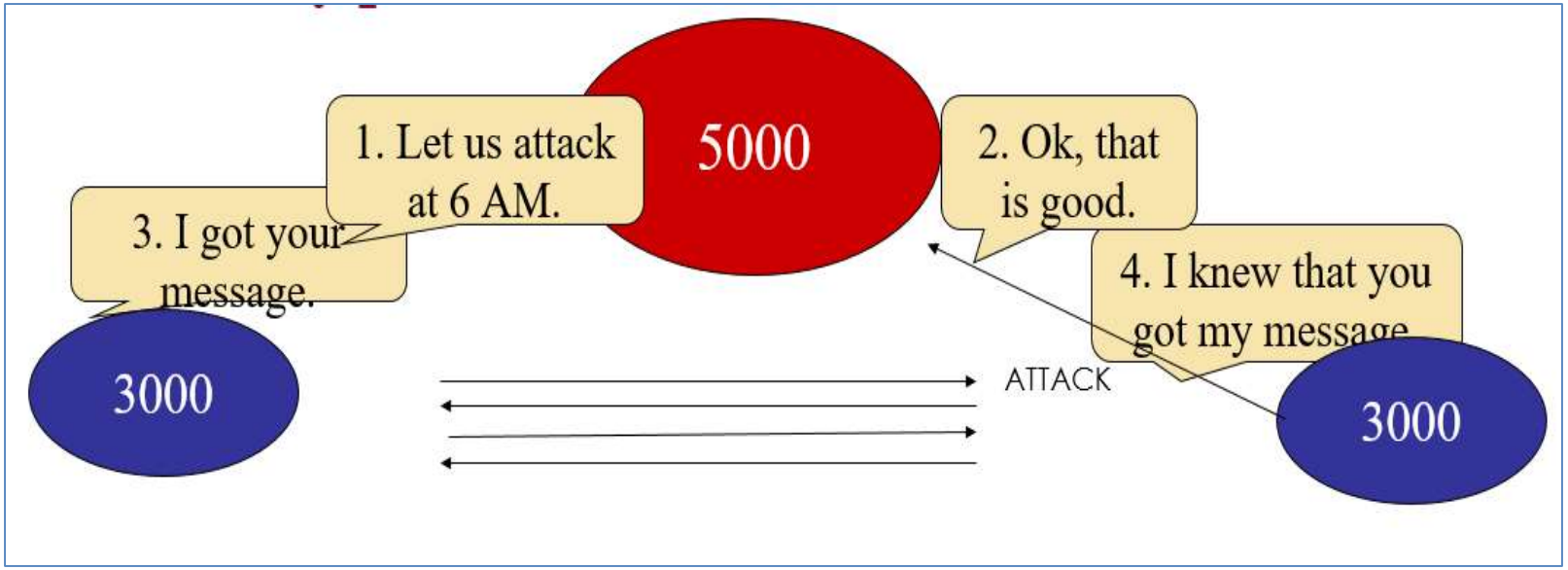
- are perfect, reaching an agreement is often straightforward
- are not perfect, there are problems in reaching an agreement
- Problems of two cases
  - Good process, but unreliable communication

    Example: Two-army problem
  - Good communication, but crashed process

    Example: Byzantine generals problem

Two-Army Problem

- This problem is classically stated as the two-army problem, and is insoluble. The agreed upon action will never take place, because the last sender will never be certain that the last confirmation went through**.(Due to unreliable communication)** ..... **(K+1)**



Byzantine Generals Problem

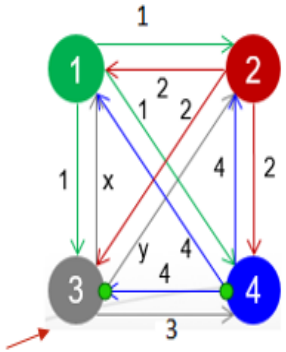
The Byzantine generals problem for 3 loyal generals and 1 traitor.

a) The generals announce their troop strengths (in units of 1 thousand soldiers).

b) The vectors that each general assembles based on (a)

c) The vectors that each general receives in step 3.

Step1: Each process sends its value to the others



Step2: Each process collects values received in a vector

1 Got(1, 2, x, 4)
2 Got(1, 2, y, 4)
3 Got(1, 2, 3, 4)
4 Got(1, 2, z, 4)

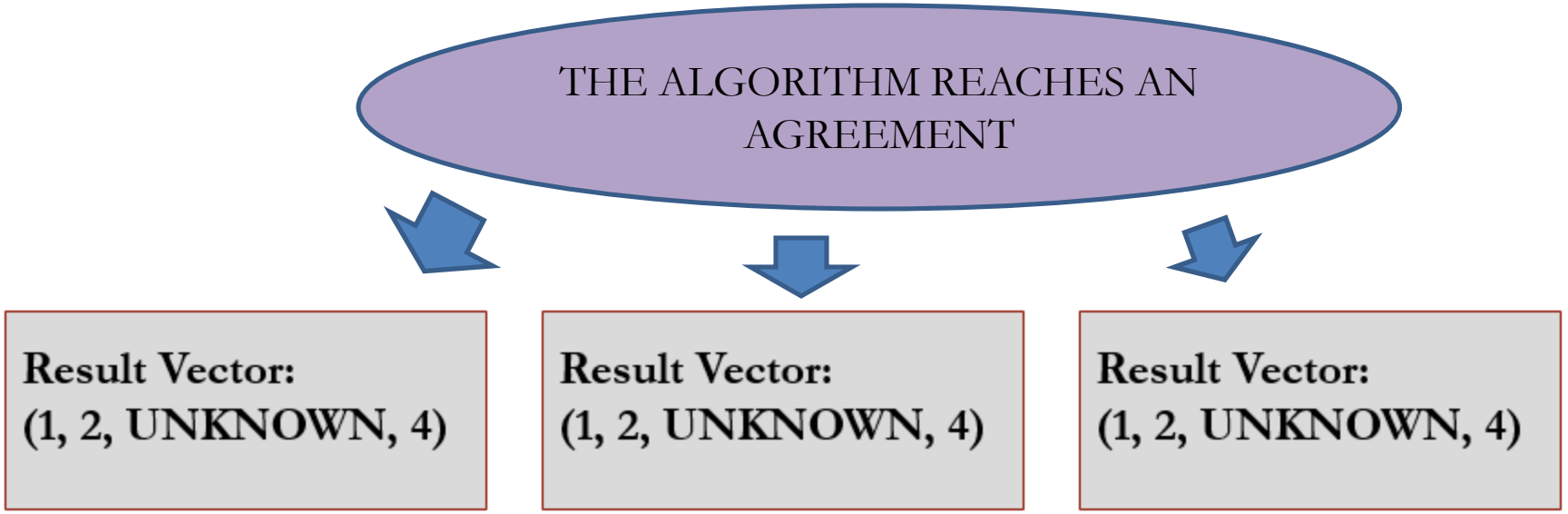
Step3: Every process passes its vector to every other process

1 Got	2 Got	4 Got
(1, 2, y, 4)	(1, 2, x, 4)	(1, 2, x, 4)
(a, b, c, d)	(e, f, g, h)	(1, 2, y, 4)
(1, 2, z, 4)	(1, 2, z, 4)	(i, j, k, l)

Byzantine Generals Problem

- Each process examines the i^{th} element of each of the newly received vectors
- If any value has a majority, that value is put into the result vector
- If no value has a majority, the corresponding element of the result vector is marked UNKNOWN

……..2K+1



Two Phase Commit (2PC)

First phase-voting phase

1. The coordinator sends 'canCommit?' request to each participants in the transactions.

2. When participants receive 'canCommit?' request, it replies with Yes or No to the coordinator. Before voting Yes, it prepares to commit by saving objects in permanent storage. If the vote is No, the participants abort immediately.

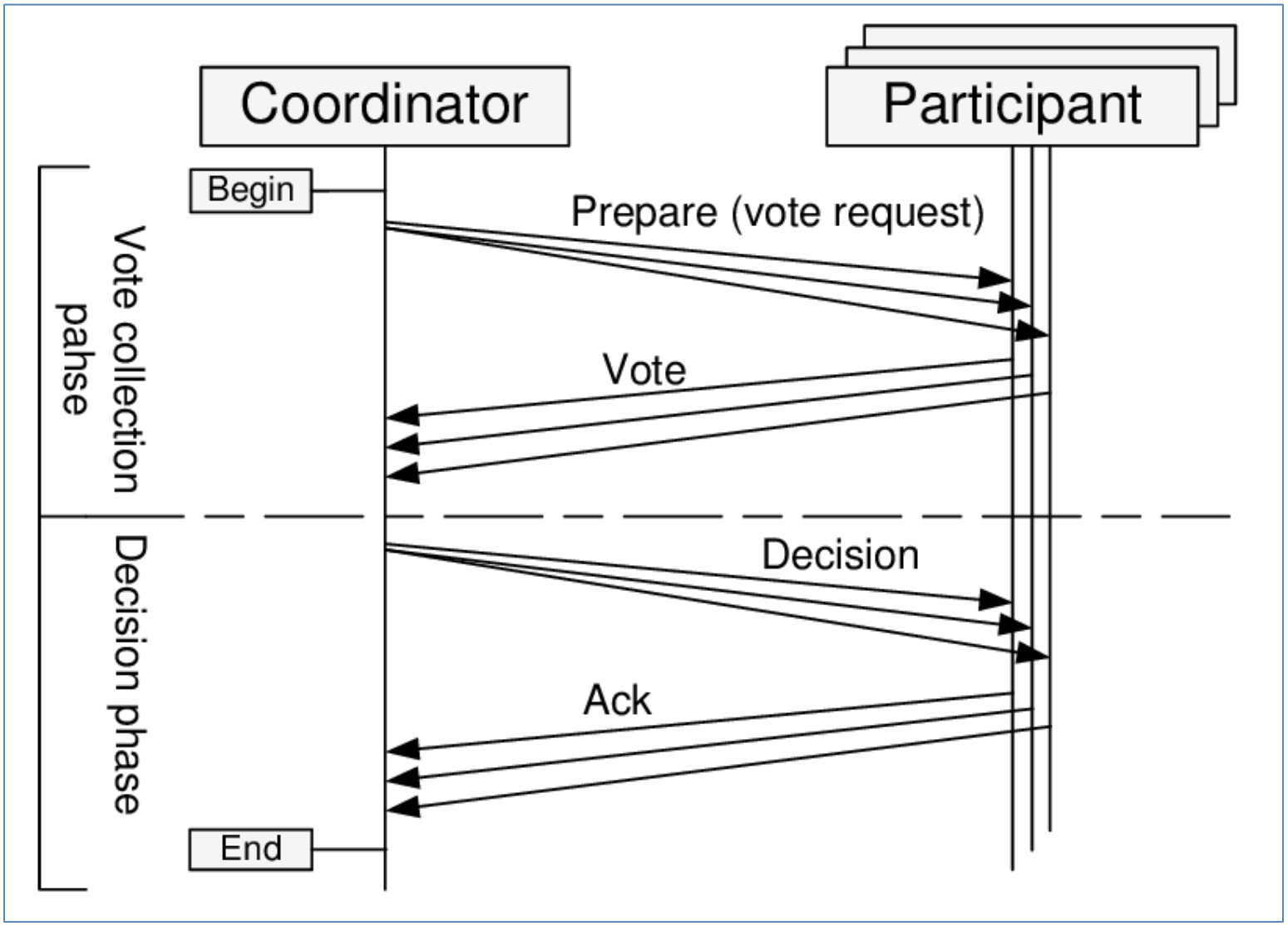
Two Phase Commit (2PC)

Second phase- completion phase

1. The coordinator collects votes including its own vote. If all the votes are Yes, the coordinator sends 'doCommit' to each of the participants. Otherwise the coordinator sends ‘doAbort’ to all the participants that voted Yes.

2. When participants receive 'doCommit' or 'doAbort', they act accordingly. If they receive 'doCommit' request, then they make 'haveCommitted' call as confirmation to the coordinator.

2PC


Coordinator
Participant
Begin
Prepare (vote request)
Vote
Vote collection pahse
Decision
Ack
Decision phase
End

Distributed Recovery

- What happens after a fault has occurred?
- A process that exhibits a failure has to be able to recover to a correct state
- There are two basic types of recovery:
  - Backward Recovery
  - Forward Recovery
- The goal of *backward recovery* is to bring the system from an erroneous state back to a prior correct state
- The state of the system must be recorded - ***checkpointed*** - from time to time, and then restored when things go wrong
- The goal of *forward recovery* is to bring a system from an erroneous state to a correct new state (not a previous state)

- Note: Backward recovery is far more widely applied

Modes of Checkpointing:

- ***Independent Checkpointing***

  - Each process periodically checkpoints independent of other processes.
  - Upon failure, locate a consistent cut backward.
  - It needs to rollback until consistent cut is found.

- ***Coordinated Checkpointing***

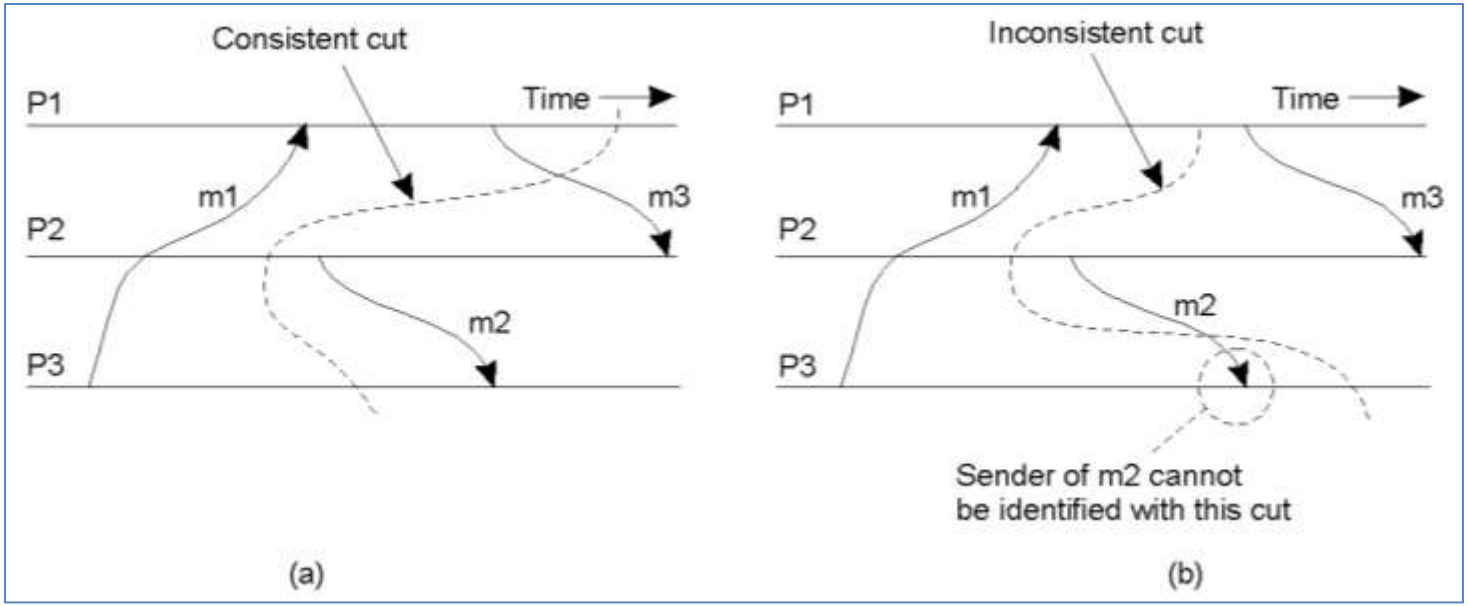
  - The state of each process in the system is periodically saved on stable storage.
  - If failure occurs, it rollbacks to previous error free state recorded by the checkpoints of all the processes.

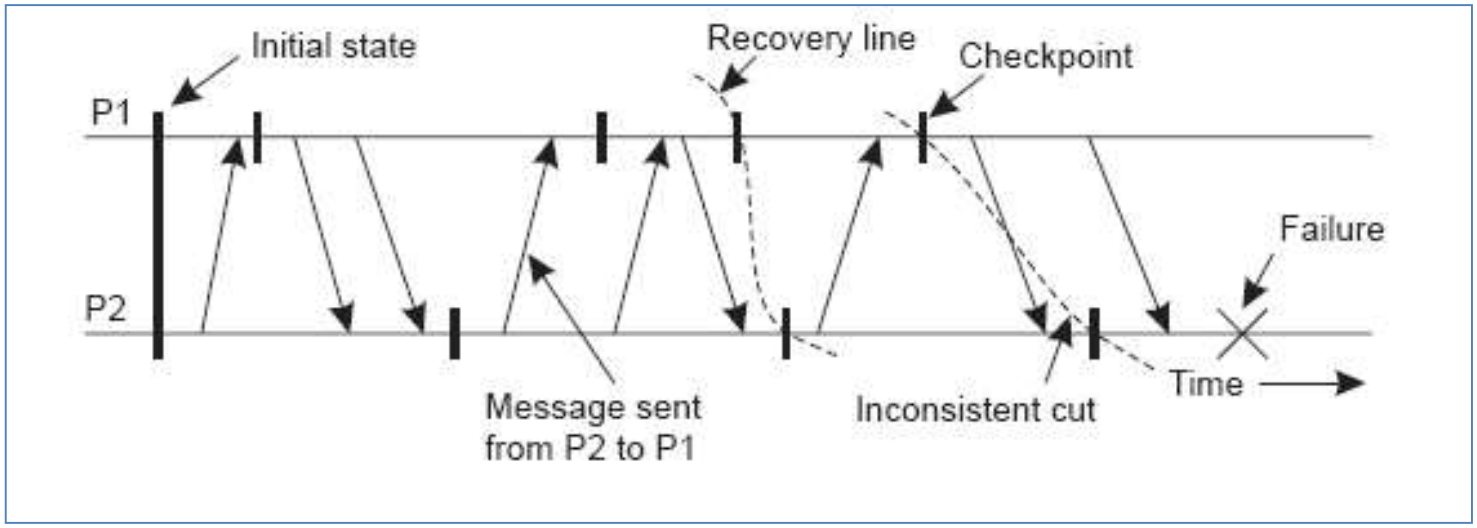
- ***Message Logging***

  - Combining checkpoint and message logs becomes cheap.

- In order to store checkpoints and logs, information needs to be stored safely - not just able to survive crashes, but also able to survive hardware faults. RAID is the typical example of stable storage

Cuts and Checkpointing:





Thank You

All the Best!!!