REPLICATION AND FAULT TOLERANCE

Dilip Kumar Shrestha

TERMINOLOGIES

- Fault Tolerance: Increase the availability of a service or the system in the event of failures. Two ways of achieving it:
  - Masking failures: Continue to perform its specified function in the event of failure.
  - Well defined failure behavior: System may or may not function in the event of failure, but can facilitate actions suitable for recovery.
    - (e.g.,) effect of database transactions visible only if committed to by all sites. Otherwise, transaction is undone without any effect to other transactions.
- Key approach to fault tolerance: redundancy. e.g., multiple copies of data, multiple processes providing same service.
- Recovery: bringing back the failed node in step with other nodes in the system.

REASONS FOR REPLICATION

- Performance enhancement
- Increased availability
- Fault tolerance

TYPES OF FAILURE

Types of failure		Description
Crash failure		**A server halts, but is working correctly until it halts**
Omission Failure		**A server fails to respond to incoming request**
	Receiver Omission	A server fails to receive to incoming request
	Send Omission	A server fails to send message
Timing failure		**A server response lies outside the specified time interval**
Response Failure		**A server response is incorrect**
	Value failure	A value of the response is wrong
	State transition failure	A server deviates from the correct flow of control
Arbitrary failure		**A server may produce arbitrary response at a arbitrary time**

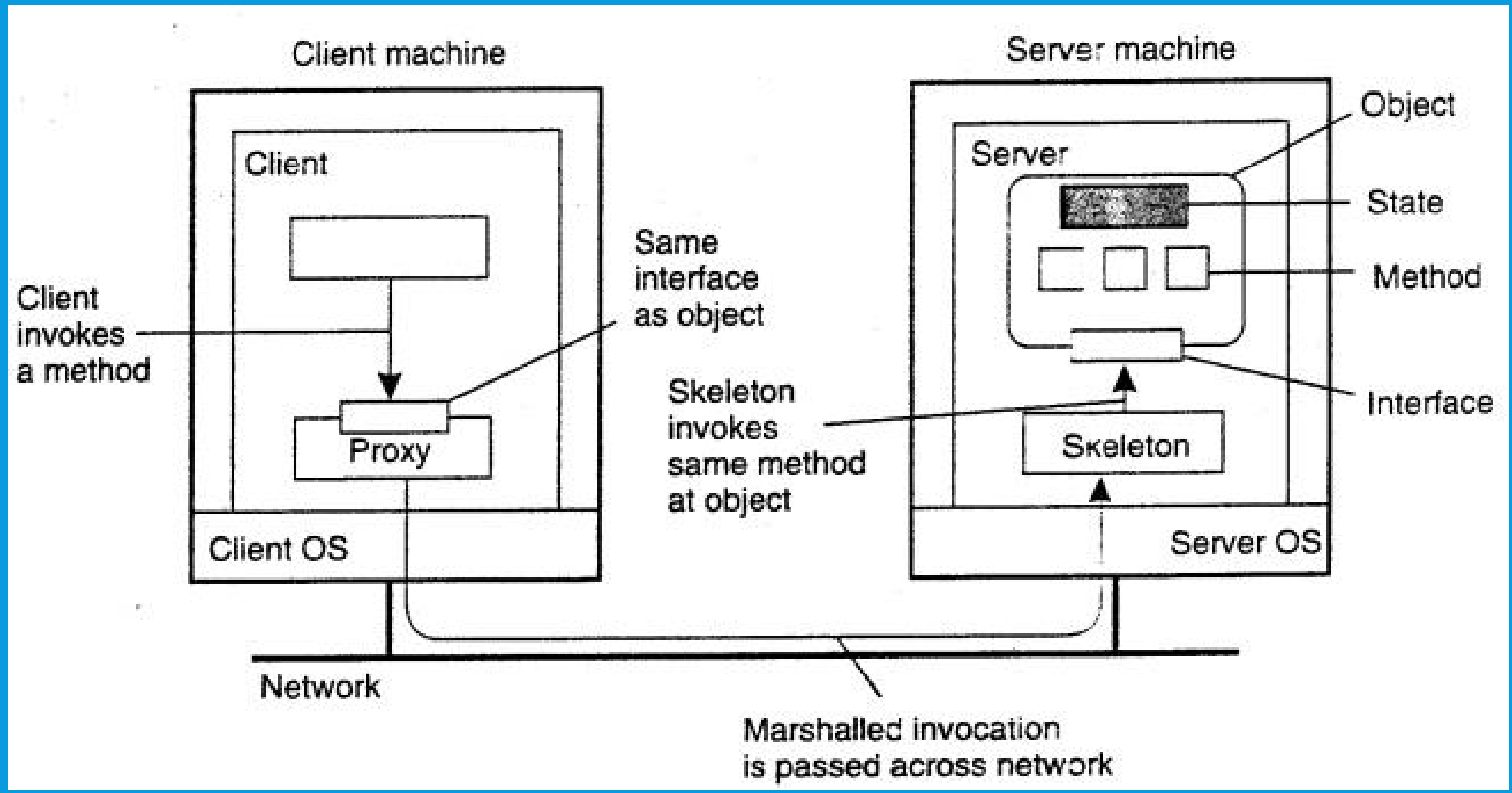
TYPES OF FAILURE

Class of failure	*Affects*	*Description*
Fail-stop	Process	Process halts and remains halted. Other processes may detect this state.
Crash	Process	Process halts and remains halted. Other processes may not be able to detect this state.
Omission	Channel	A message inserted in an outgoing message buffer never arrives at the other end's incoming message buffer.
Send-omission	Process	A process completes a *send* operation but the message is not put in its outgoing message buffer.
Receive-omission	Process	A message is put in a process's incoming message buffer, but that process does not receive it.
Arbitrary (Byzantine)	Process or channel	Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times or commit omissions; a process may stop or take an incorrect step.

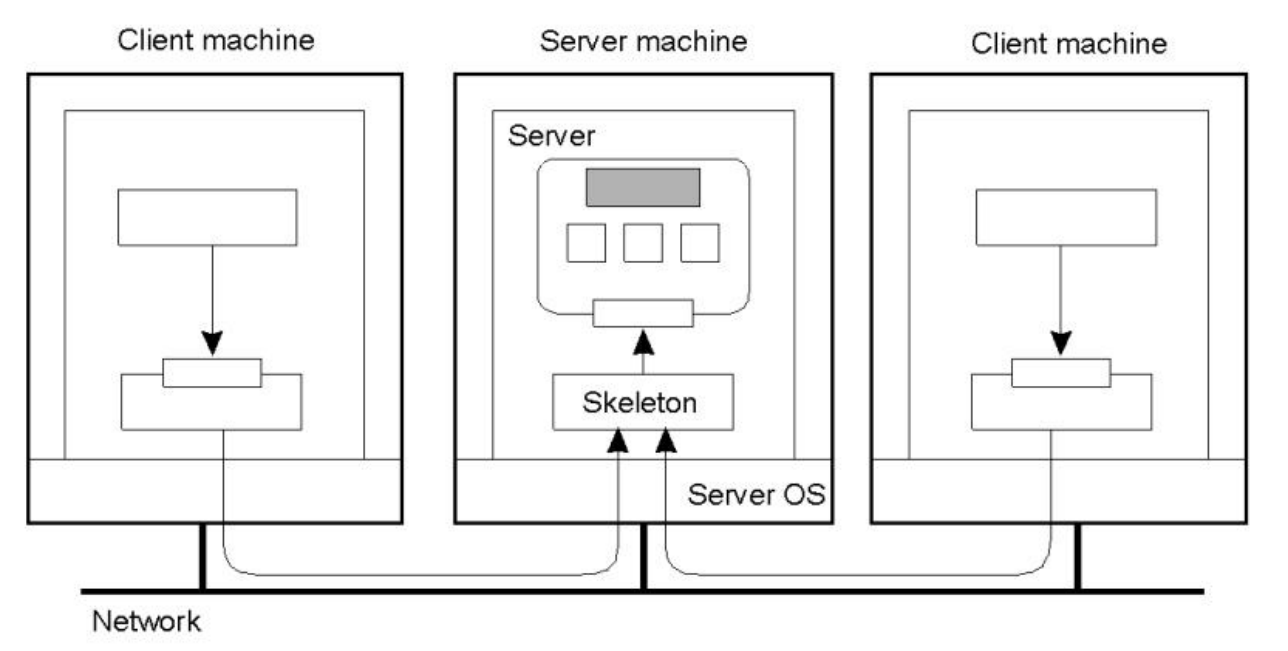
TIMING FAILURE

Class of failure	*Affects*	*Description*
Clock	Process	Process's local clock exceeds the bounds on its rate of drift from real time.
Performance	Process	Process exceeds the bounds on the interval between two steps.
Performance	Channel	A message's transmission takes longer than the stated bound.

OBJECT REPLICATION



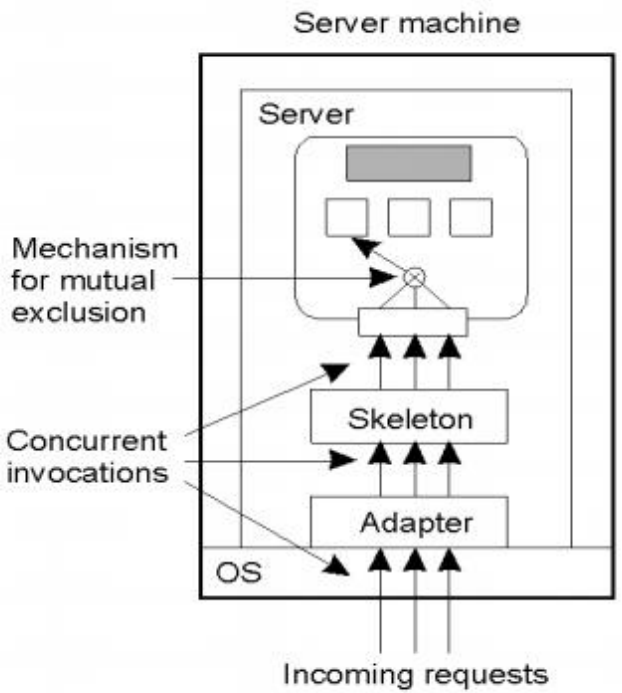
OBJECT REPLICATION



Problem: If objects (or data) are shared, we need to do something about concurrent accesses to guarantee state consistency

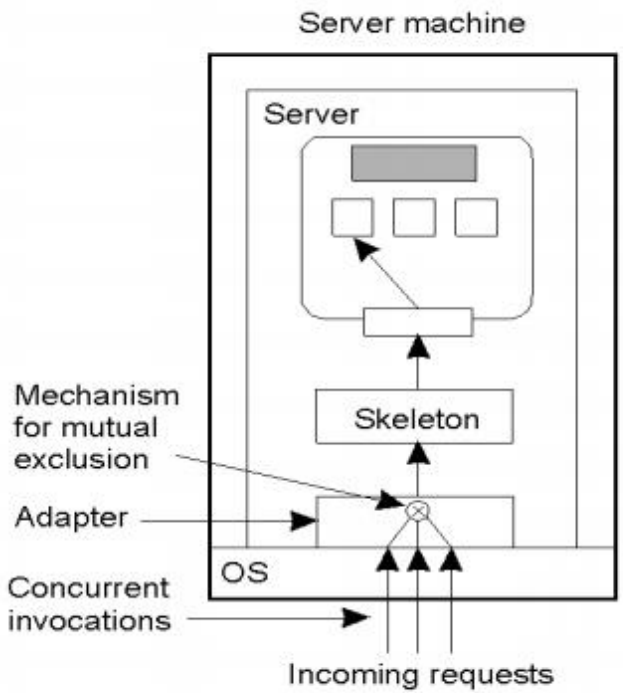
OBJECT REPLICATION SOLUTIONS

A remote object capable of handling concurrent invocations on its own.



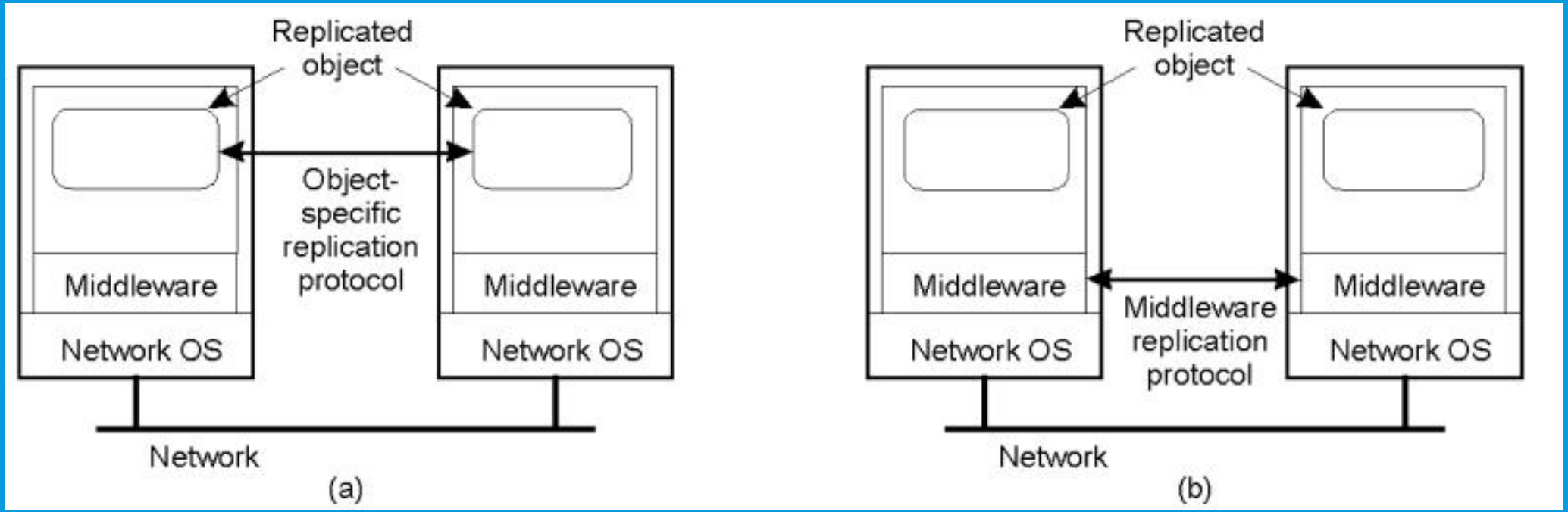
(a)

A remote object for which an object adapter is required to handle concurrent invocations



(b)

IMPLEMENTATION



A distributed system for replication-aware distributed objects is more customizable

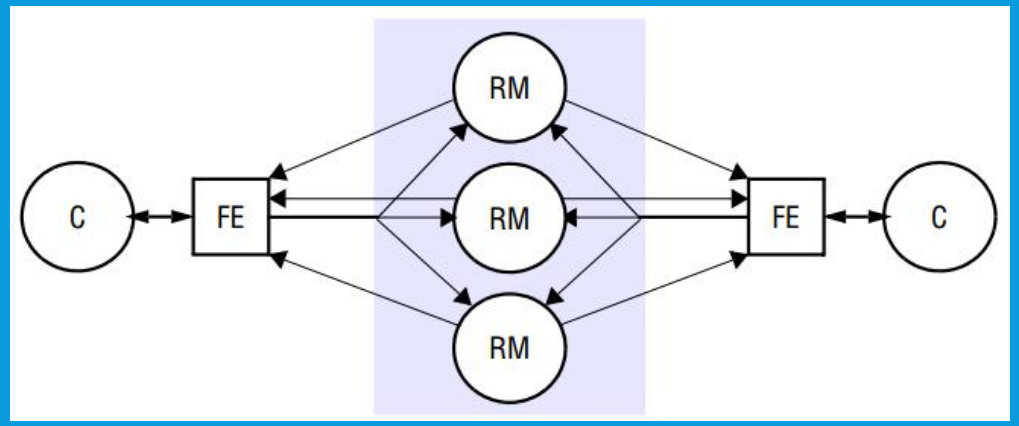
A distributed system responsible for replica management is more general and easier to implement

REPLICATION AS SCALING TECHNIQUE

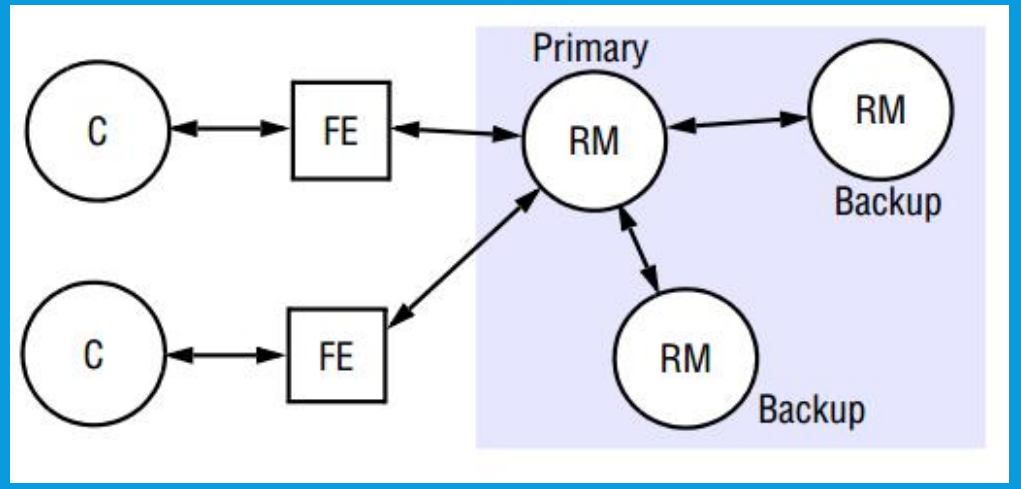
Scalability issues generally appear in the form of performance problems.

FAULT TOLERANCE SERVICE

Active Replication



Passive Replication



PASSIVE VS ACTIVE

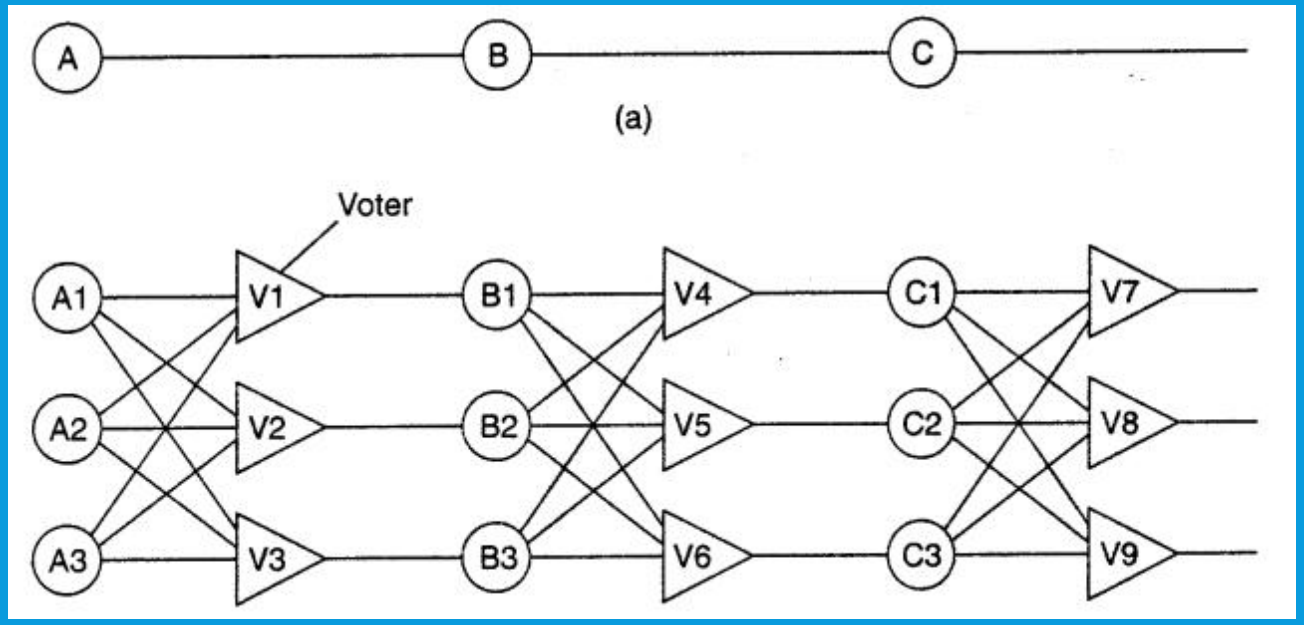
Passive Replication

- *Request*: The front end issues the request, containing a unique identifier, to the primary replica manager.
- *Coordination*: The primary takes each request atomically, in the order in which it receives it. It checks the unique identifier, in case it has already executed the request, and if so it simply resends the response.
- *Execution*: The primary executes the request and stores the response.
- *Agreement*: If the request is an update, then the primary sends the updated state, the response and the unique identifier to all the backups. The backups send an acknowledgement.
- *Response*: The primary responds to the front end, which hands the response back to the client.

Active Replication

- *Request*: The front end attaches a unique identifier to the request and multicasts it to the group of replica managers, using a totally ordered, reliable multicast primitive. The front end is assumed to fail by crashing at worst. It does not issue the next request until it has received a response.
- *Coordination*: The group communication system delivers the request to every correct replica manager in the same (total) order.
- *Execution*: Every replica manager executes the request. Since they are state machines and since requests are delivered in the same total order, correct replica managers all process the request identically. The response contains the client's unique request identifier.
- *Agreement*: No agreement phase is needed, because of the multicast delivery semantics.
- *Response*: Each replica manager sends its response to the front end. The number of replies that the front end collects depends upon the failure assumptions and the multicast algorithm.

FAILURE MASKING BY REDUNDANCY



- **time redundancy** - an action is performed, and then. if need be, it is performed again
- **physical redundancy** - extra equipment or processes are added
- Example: Triple Modular Redundancy (TMR)

AGREEMENT IN FAULTY SYSTEM

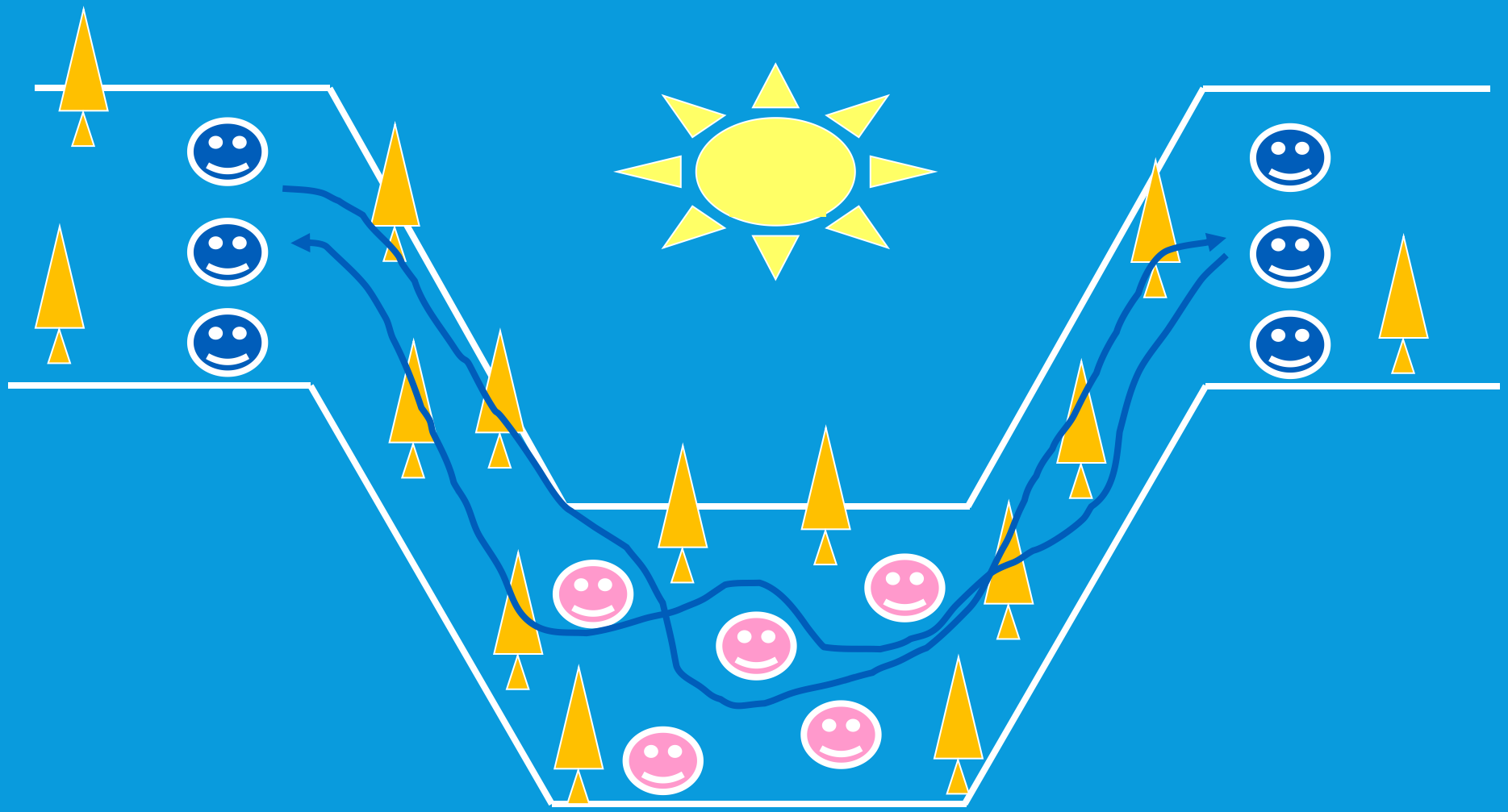
Process Failure

Channel failure

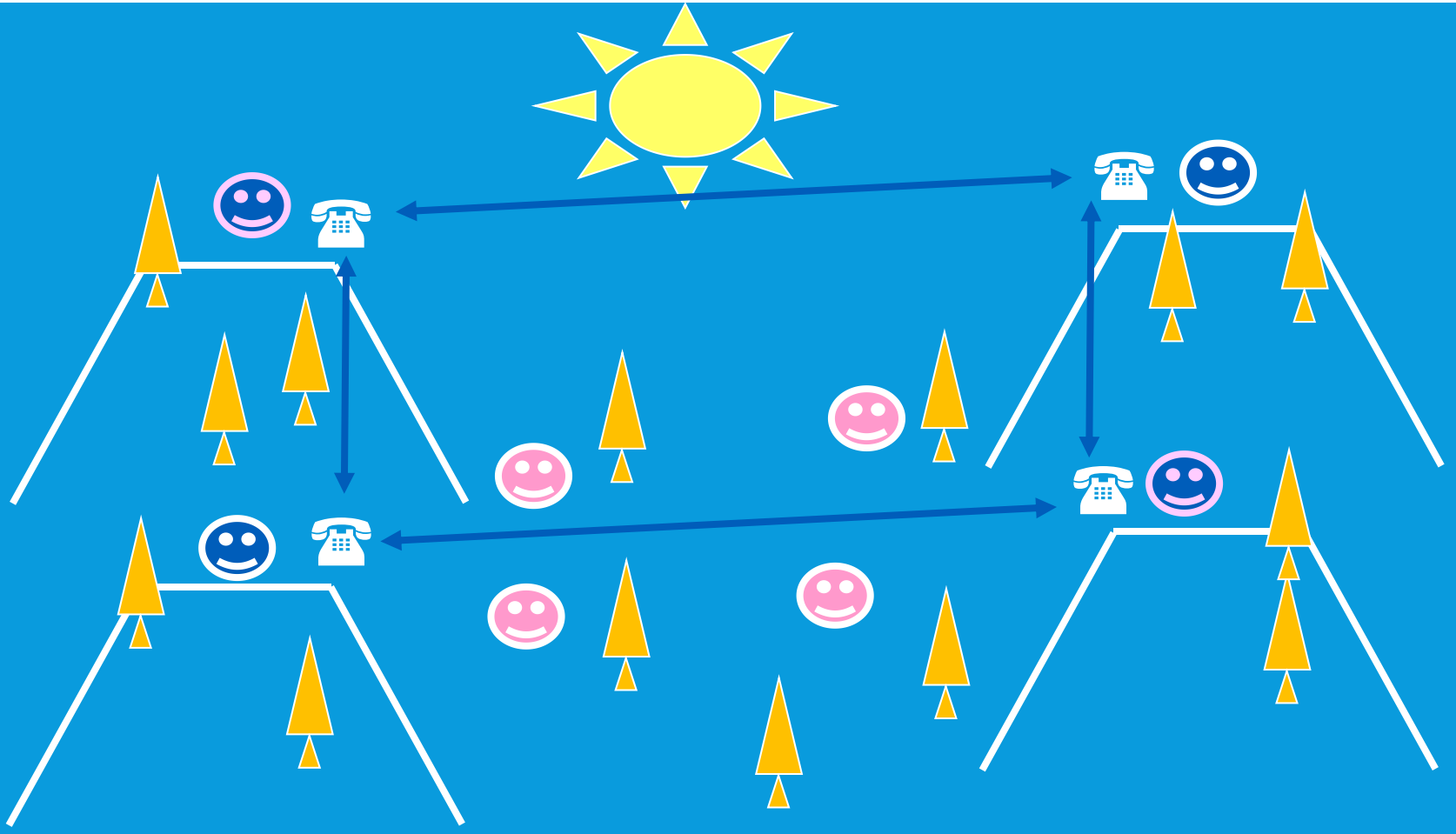
AGREEMENT

- **Distributed Agreement algorithm**: all non-faulty processes achieve consensus in a finite number of steps
- Perfect processes, faulty channels: two-army
- Faulty processes, perfect channels: Byzantine generals

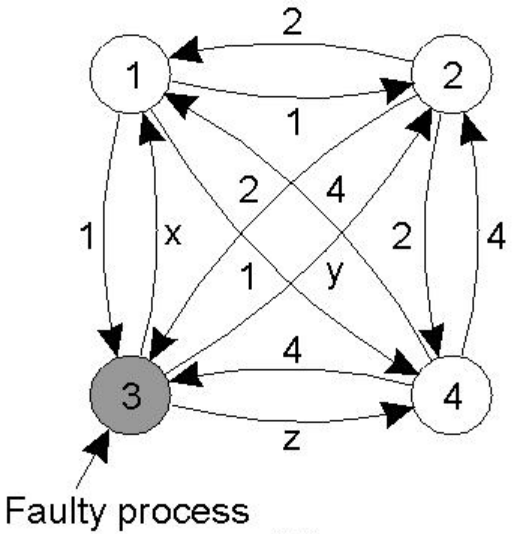
TWO-ARMY PROBLEM



BYZANTINE GENERALS PROBLEM



BYZANTINE GENERALS -EXAMPLE (1)



(a)

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

(b)

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)

(c)

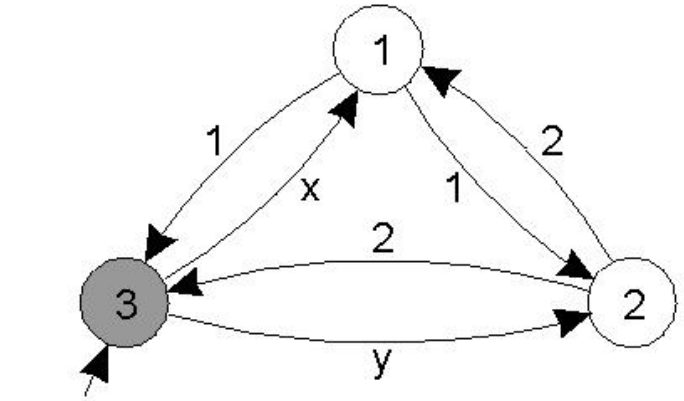
The Byzantine generals problem for 3 loyal generals and1 traitor.

a) The generals announce the time to launch the attack (by messages marked by their ids).

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

c) The vectors that each general receives in step 3, where every general passes his vector from (b) to every other general.

BYZANTINE GENERALS –EXAMPLE (2)



(a)

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

(b)

1 Got	2 Got
(1, 2, y)	(1, 2, x)
(a, b, c)	(d, e, f)

(c)

The same as in previous slide, except now with 2 loyal generals and one traitor.

CONCLUSION: BYZANTINE GENERALS

Given three processes, if one fails, consensus is impossible

Given N processes, if F processes fail, consensus is impossible if $N \leq 3F$

RELIABLE CLIENT SERVER COMMUNICATION

Point to point communication

RPC Semantics in presence of Failure

- The client is unable to locate the server.
- The request message from the client to the server is lost.
- The server crashes after receiving a request.
- The reply message from the server to the client is lost.
- The client crashes after sending a request.

RECOVERY APPROACHES

Definition: Bringing back the failed node in step with other nodes in the system.

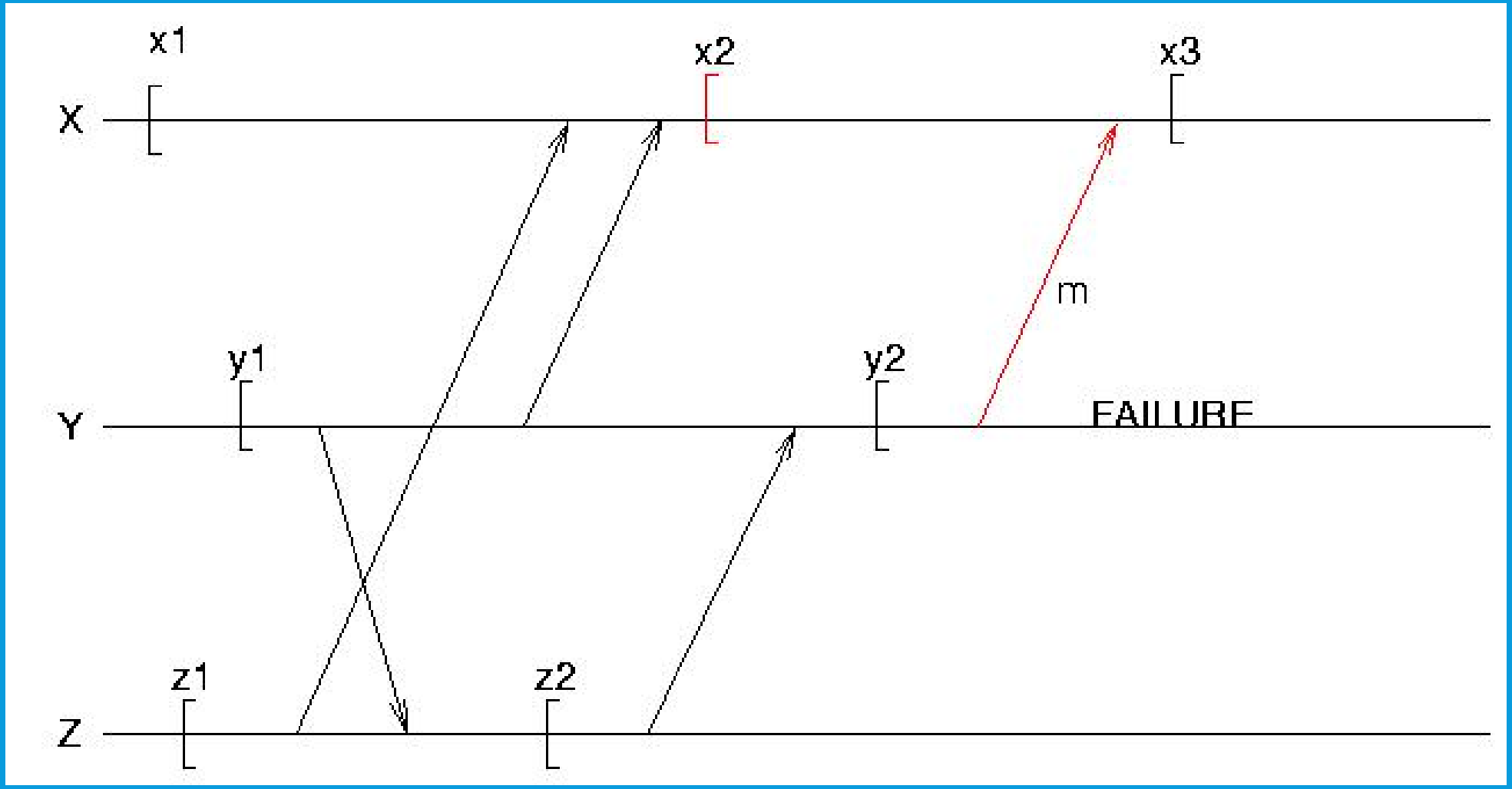
Forms of Recovery

- Backward Recovery
- Forward Recovery

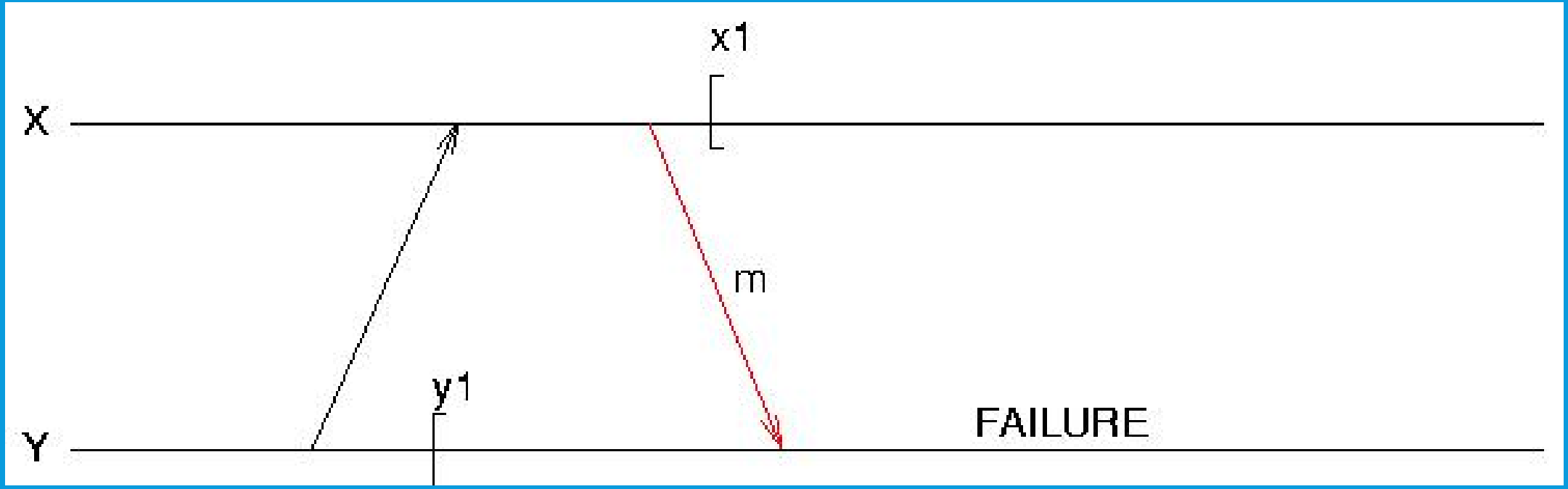
Key Approaches

- Check pointing
  - Independent Check pointing
  - Coordinated Check pointing
- Message Logging
- Recovery-Oriented Computing

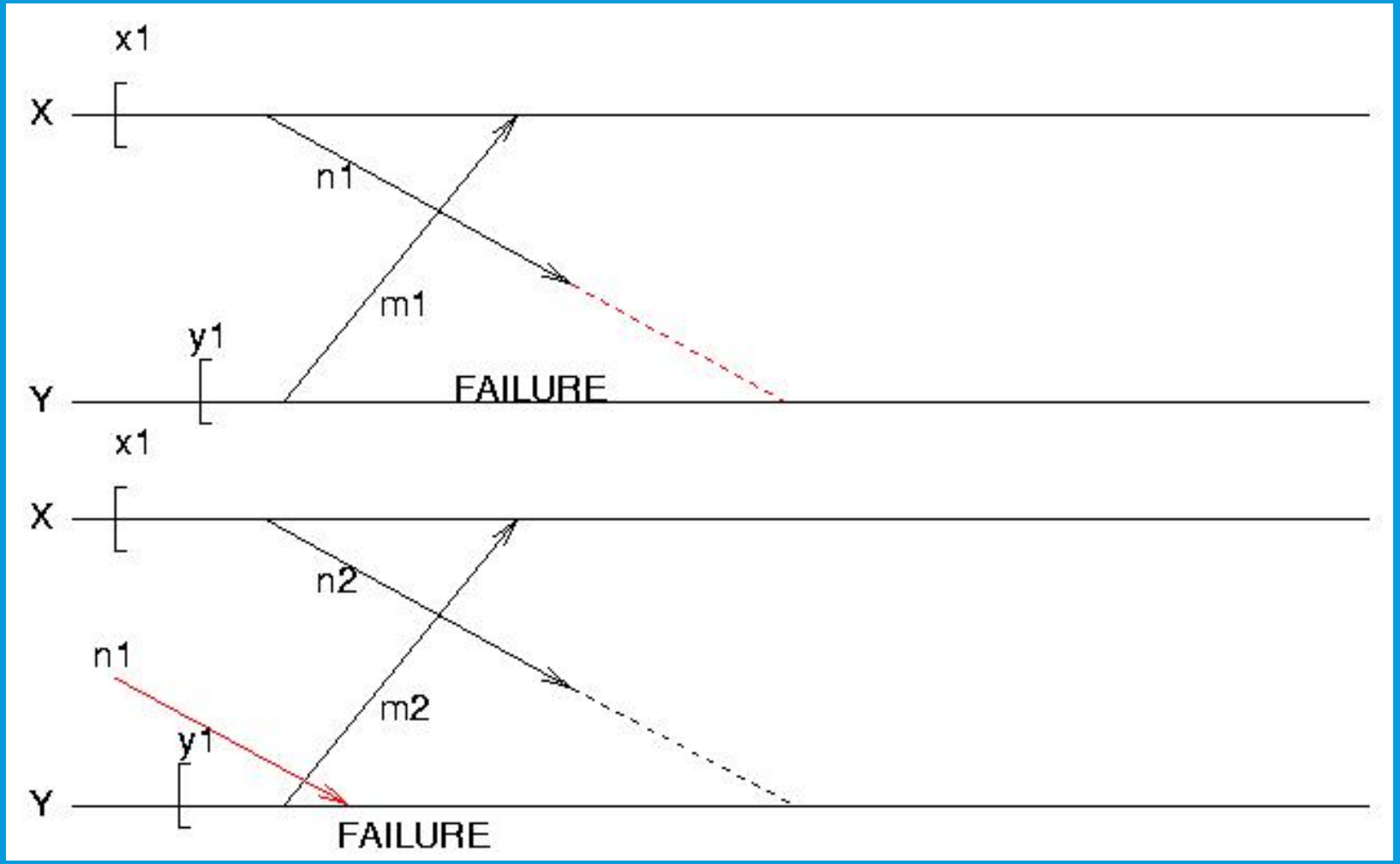
PROBLEMS: ORPHAN MESSAGES AND DOMINO EFFECT



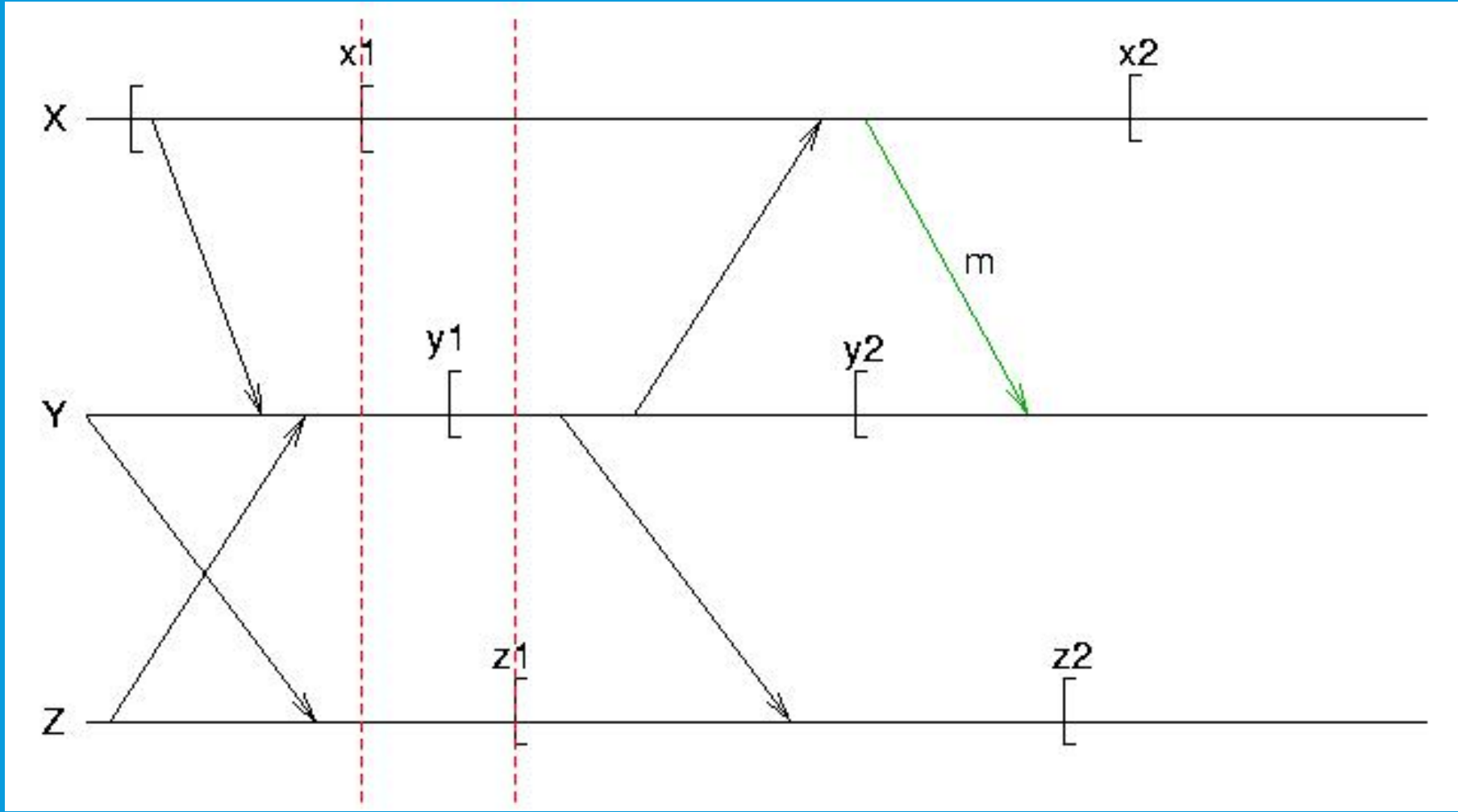
PROBLEMS: LOST MESSAGE



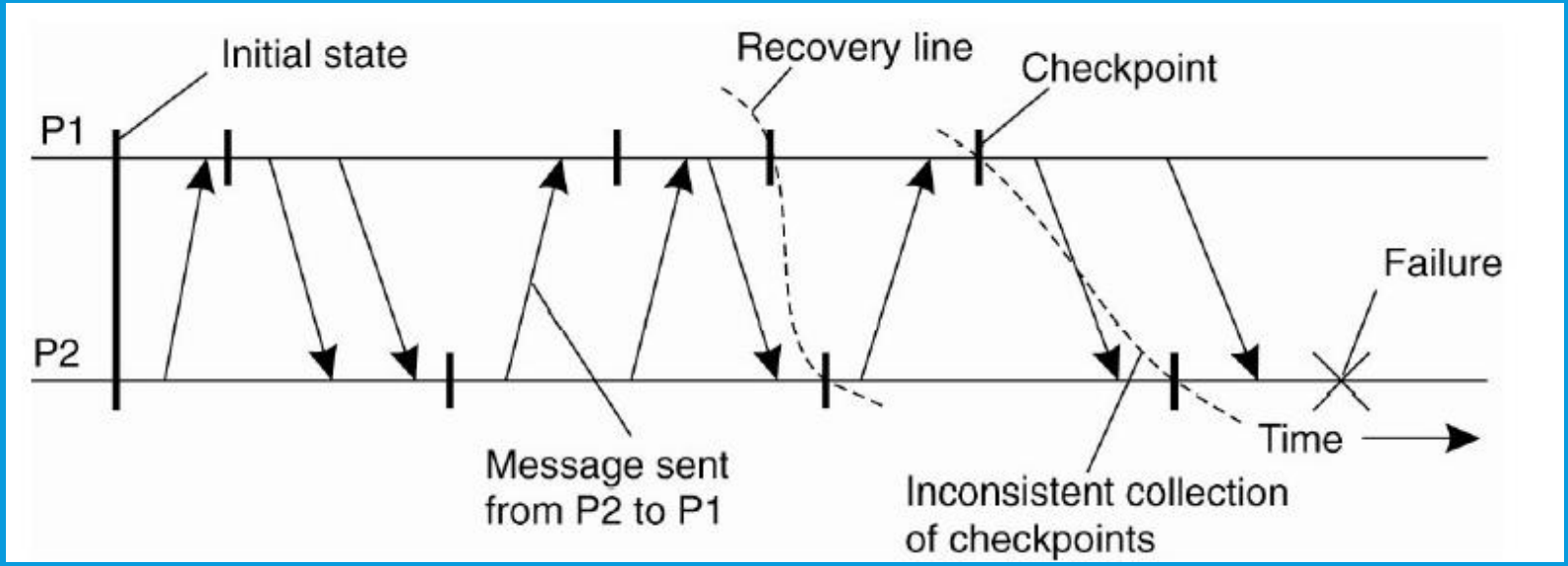
PROBLEMS: LIVE LOCK



STRONGLY CONSISTENT SET OF CHECKPOINTS



CONSISTENT RECOVERY STATE

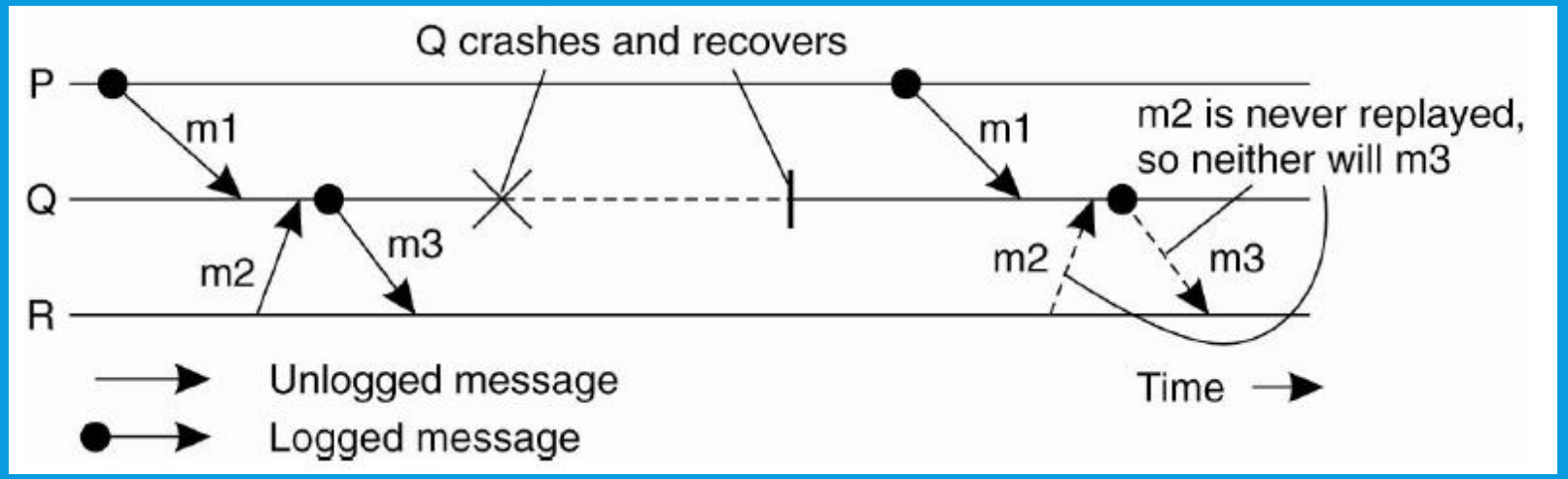


Requirement
Every message that has been received is also shown to have been
sent in the state of the sender (i.e. strongly consistent set of checkpoints).

Recovery line
Assuming processes regularly checkpoint their state, the most recent
consistent glob

Observation
If and only if the system provides reliable communication. Sent messages should also be received in a consistent state. al checkpoint.

MESSAGE LOGGING



- check pointing is expensive
- Record all events and replay later
- Events are nondeterministic, so make a deterministic model to replay

THANK YOU