Fault Tolerance

Distributed Systems [8]

殷亚凤

Email: yafeng@nju.edu.cn

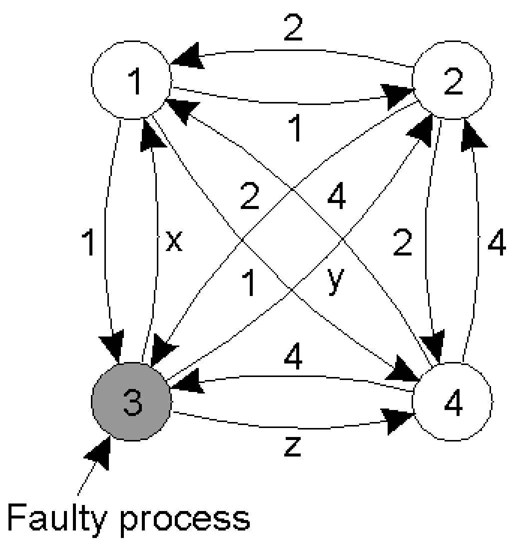
Homepage: http://cs.nju.edu.cn/yafeng/

Room 301, Building of Computer Science and Technology

Review

- Concepts about faults
- How to improve dependability
- Two-army problem
- Byzantine agreement problem

Byzantine agreement problem



(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 and 1 traitor.

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

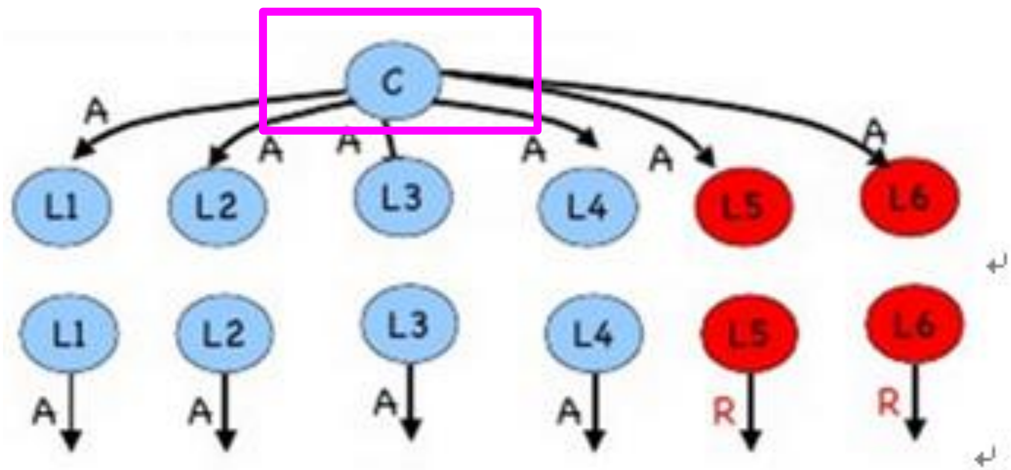
b) The vectors that each general assembles based on (a)

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

Byzantine Generals Problem

A commanding general must send an order to his n - 1 lieutenant generals such that

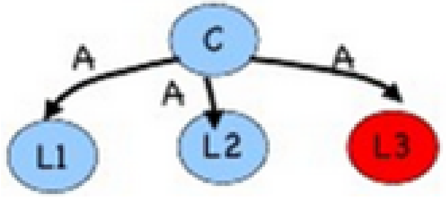
- IC1. All loyal lieutenants obey the same order.
- IC2. If the commanding general is loyal, then every loyal lieutenant obeys the order he sends.



Oral Message Algorithm

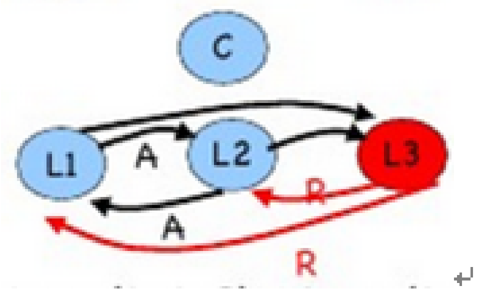
Algorithm OM(0):

1. Commander sends his value to every lieutenant
2. Each lieutenant uses the value received or "retreat" if no value received



Algorithm OM(m), m > 0:

1. Commander sends his value to every lieutenant
2. For each i, let v_i be the value that lieutenant i receives from the commander or "retreat". Lieutenant i acts as the commander in OM(m-1) to send the value v_i to each of the other n-2 other lieutenants
3. For each i, and each $j <> i$, let v_j be the value lieutenant i received from lieutenant j in step 2. Lieutenant i uses the value *majority* $(v_1, \ldots, v_{n-1})$



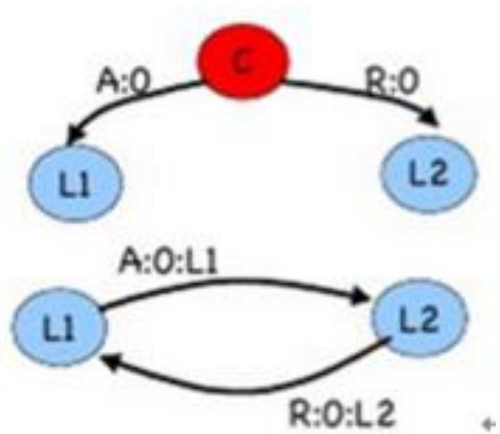
Signed Messages

- A1. Every message that is sent is delivered correctly.
- A2. The receiver of a message knows who sent it.
- A3. The absence of a message can be detected.
- A4. (a) A loyal general's signature cannot be forged, and any alteration of the contents of his signed messages can be detected.

  (b) Anyone can verify the authenticity of a general's signature.

Signed Messages

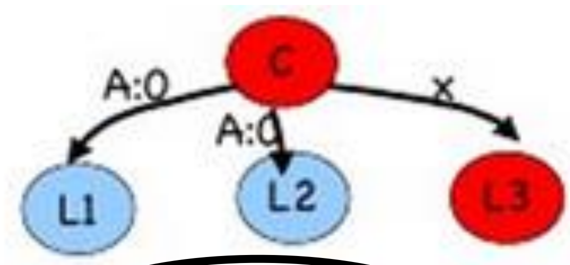
- IC1. All loyal lieutenants obey the same order.
- IC2. If the commanding general is loyal, then every loyal lieutenant obeys the order he sends.



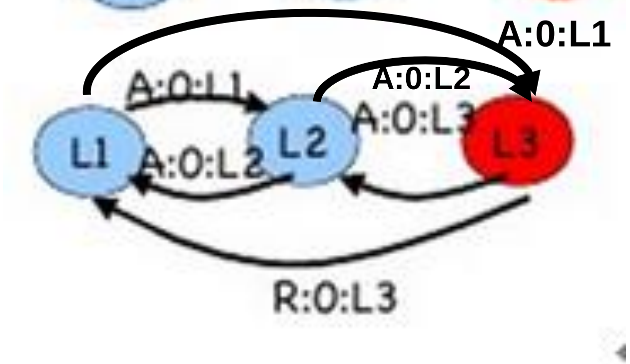
V1={A,R},
V2={A,R}。

Signed Messages

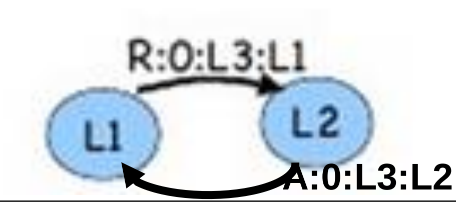
- IC1. All loyal lieutenants obey the same order.
- IC2. If the commanding general is loyal, then every loyal lieutenant obeys the order he sends.



L1={A:0}
L2={A:0}



L1={A:0, A:0:L2, R:0:L3}
L2={A:0, A:0:L1, A:0:L3}



L1={A:0, A:0:L2, R:0:L3, A:0:L3:L2}
L2={A:0, A:0:L1, A:0:L3, R:0:L3:L1}

This Lesson

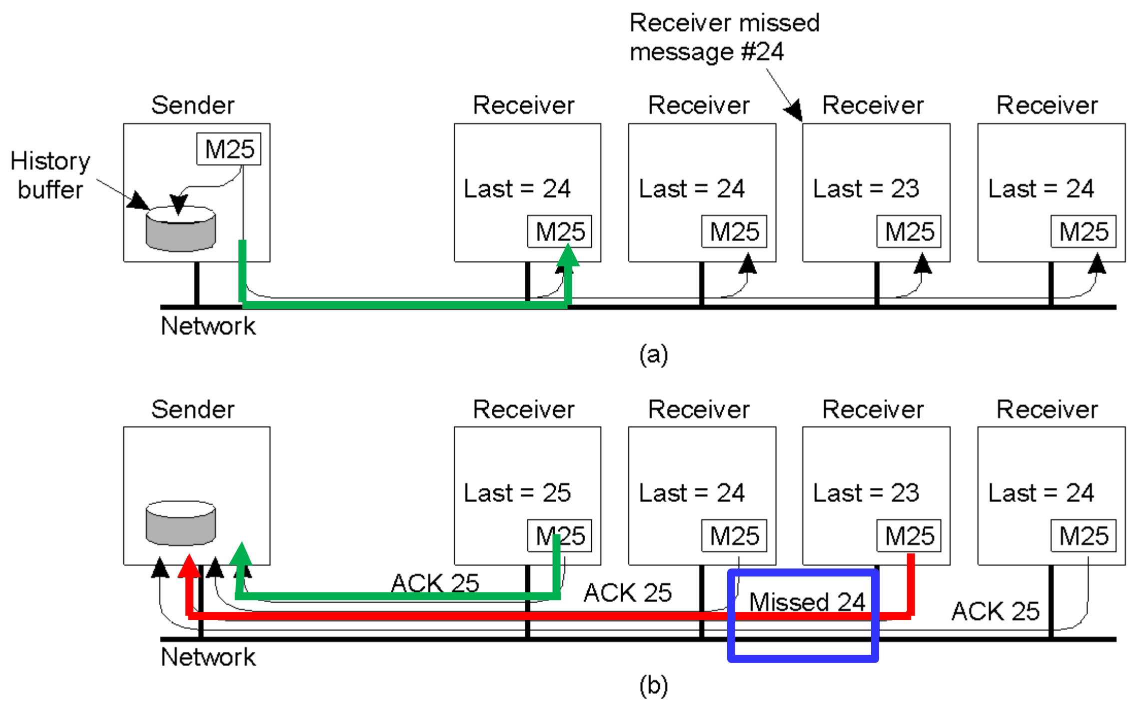
- Reliable Multicast
- Distributed commit
- Recovery

Reliable Multicast

1. Basic reliable-multicasting schemes

2. Scalability in reliable multicasting
   - Nonhierarchical Feedback Control
   - Hierarchical Feedback Control

Basic Reliable-Multicasting Schemes

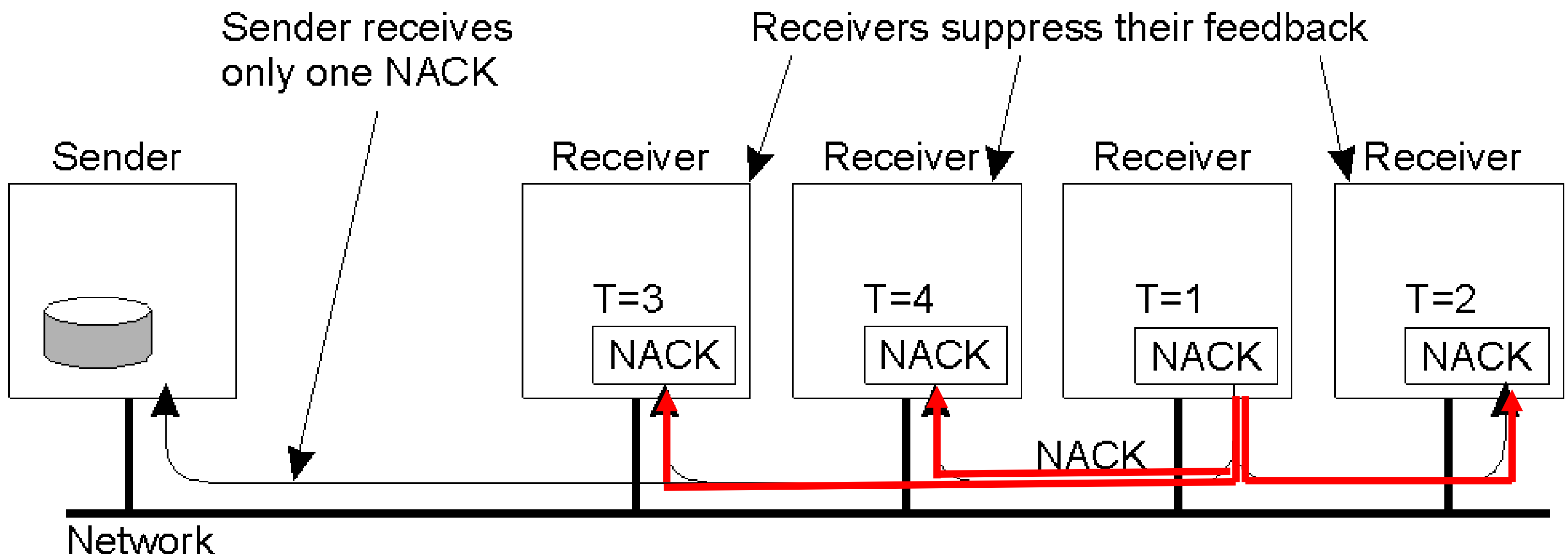


- A simple solution to reliable multicasting when all receivers are known and are assumed not to fail

a) Message transmission

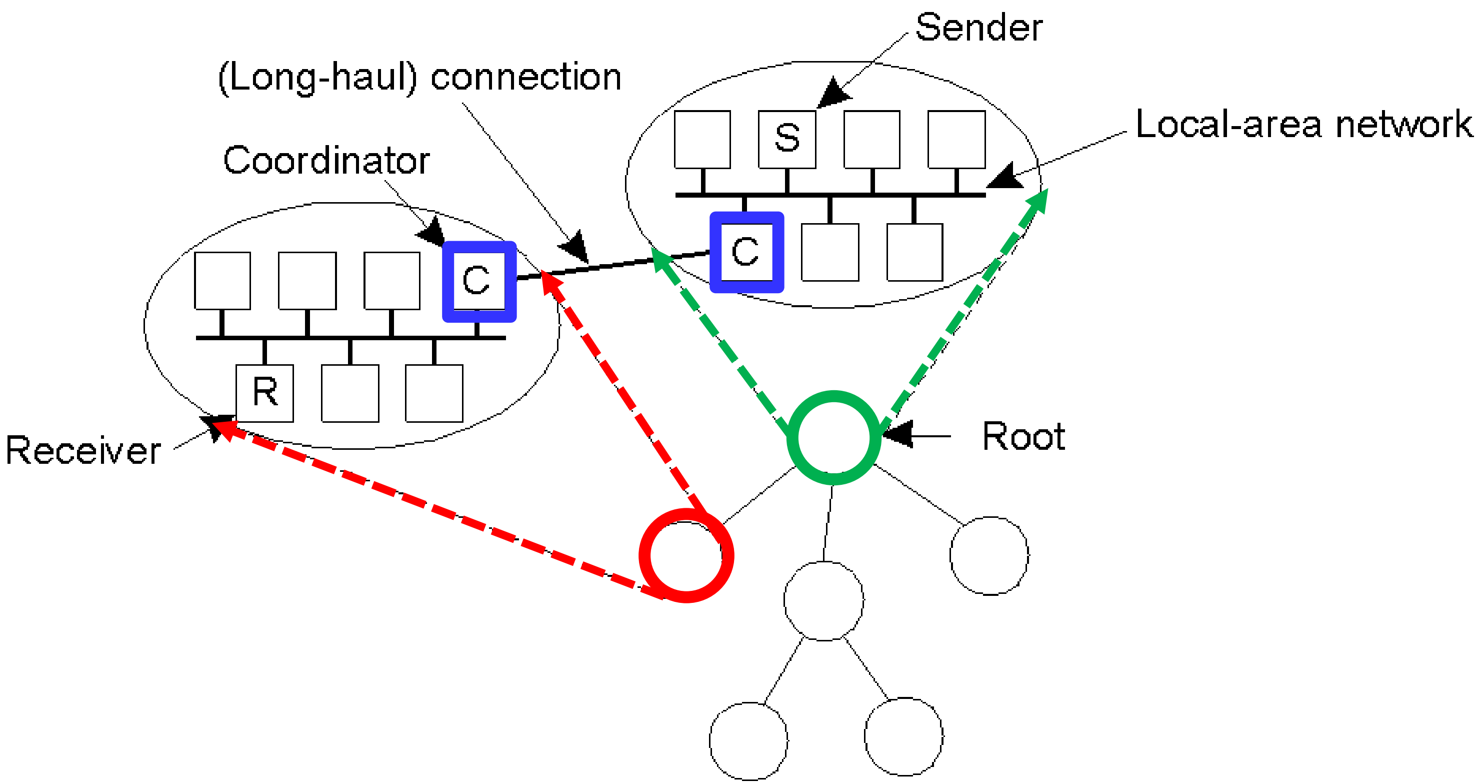
b) Reporting feedback

Nonhierarchical Feedback Control



- Several receivers have scheduled a request for retransmission, but the first retransmission request leads to the suppression of others.

Hierarchical Feedback Control



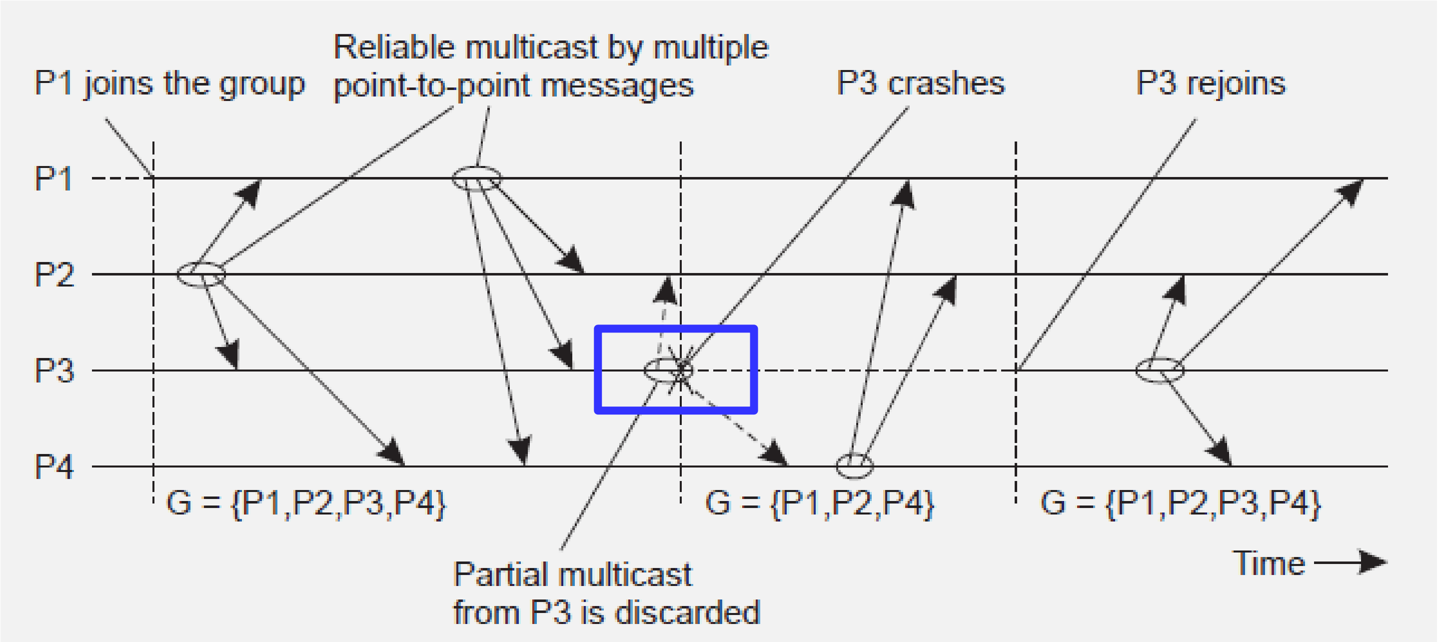
- The essence of hierarchical reliable multicasting.

a) Each local coordinator forwards the message to its children.

b) A local coordinator handles retransmission requests.

Atomic multicast

- Formulate reliable multicasting in the presence of process failures in terms of process groups and changes to group membership.



Atomic multicast

- A message is delivered only to the nonfaulty members of the current group. All members should agree on the current group membership ⇒ Virtually synchronous multicast.

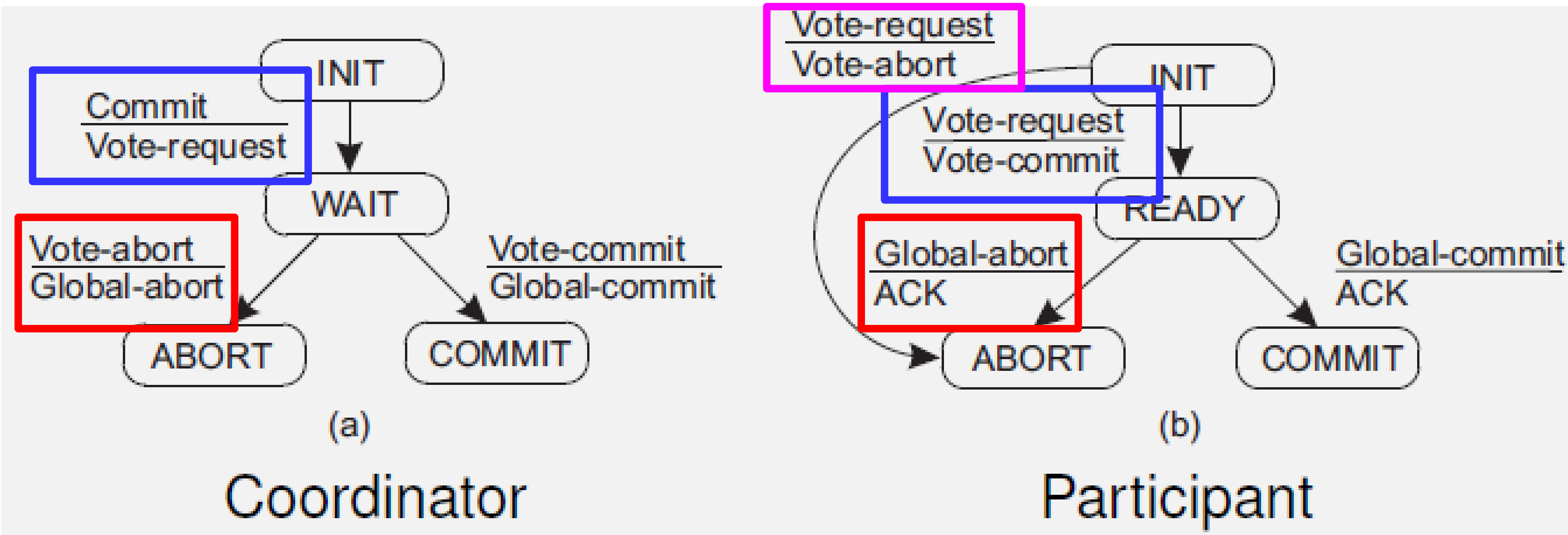
Distributed commit

- Two-phase commit
- Three-phase commit
- Essential issue
  - Given a computation distributed across a process group, how can we ensure that either all processes commit to the final result, or none of them do (atomicity)?

Two-phase commit

- The client who initiated the computation acts as coordinator; processes required to commit are the participants
  - Phase 1a: Coordinator sends vote-request to participants (also called a pre-write)
  - Phase 1b: When participant receives vote-request it returns either vote-commit or vote-abort to coordinator. If it sends vote-abort, it aborts its local computation
  - Phase 2a: Coordinator collects all votes; if all are vote-commit, it sends global-commit to all participants, otherwise it sends global-abort
  - Phase 2b: Each participant waits for global-commit or global-abort and handles accordingly.

Two-phase commit



2PC – Failing participant

- Participant crashes in state S, and recovers to S
  - Initial state: No problem: participant was unaware of protocol
  - Ready state: Participant is waiting to either commit or abort. After recovery, participant needs to know which state transition it should make ⇒ log the coordinator's decision
  - Abort state: Merely make entry into abort state idempotent, e.g., removing the workspace of results
  - Commit state: Also make entry into commit state idempotent, e.g., copying workspace to storage.
- When distributed commit is required, having participants use **temporary workspaces** to keep their results allows for simple recovery in the presence of failures.

2PC – Failing participant

- When a recovery is needed to READY state, check state of other participants ⇒ no need to log coordinator's decision.
- Recovering participant P contacts another participant Q

State of Q	Action by P
COMMIT	Make transition to COMMIT
ABORT	Make transition to ABORT
INIT	Make transition to ABORT
READY	Contact another participant

- If all participants are in the READY state, the protocol blocks. Apparently, the coordinator is failing. Note: The protocol prescribes that we need the decision from the coordinator.

2PC – Failing participant

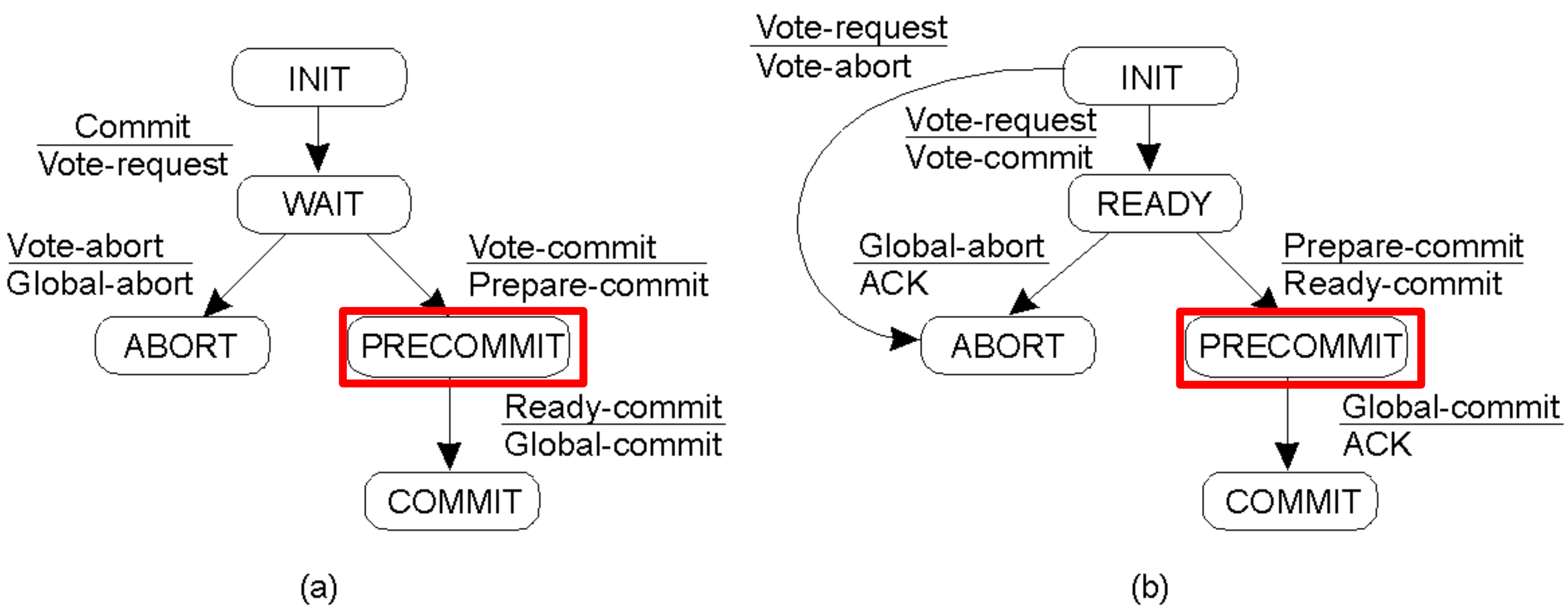
- The real problem lies in the fact that the coordinator's final decision may not be available for some time (or actually lost).
- Let a participant P in the READY state timeout when it hasn't received the coordinator's decision; P tries to find out what other participants know (as discussed).
- Essence of the problem is that a recovering participant cannot make a local decision: it is dependent on other (possibly failed) processes

Three-Phase Commit

- The states of the coordinator and each participant satisfy the following two conditions:

1. There is **no single state** from which it is possible to make a transition directly to either a COMMIT or an ABORT state.
2. There is **no state** in which it is not possible to make a final decision, and from which a transition to a COMMIT state can be made.

Three-Phase Commit



Coordinator **Participant**

Recovery

1. Introduction

- Recovery:

  A process where a failure happened can recover to a correct state.

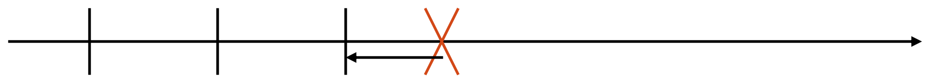
- What do we need for recovery?

  record states of a distributed system

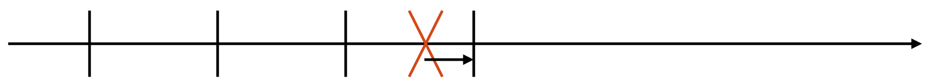
  when and how?

- Two forms of error recovery

backward recovery



forward recovery

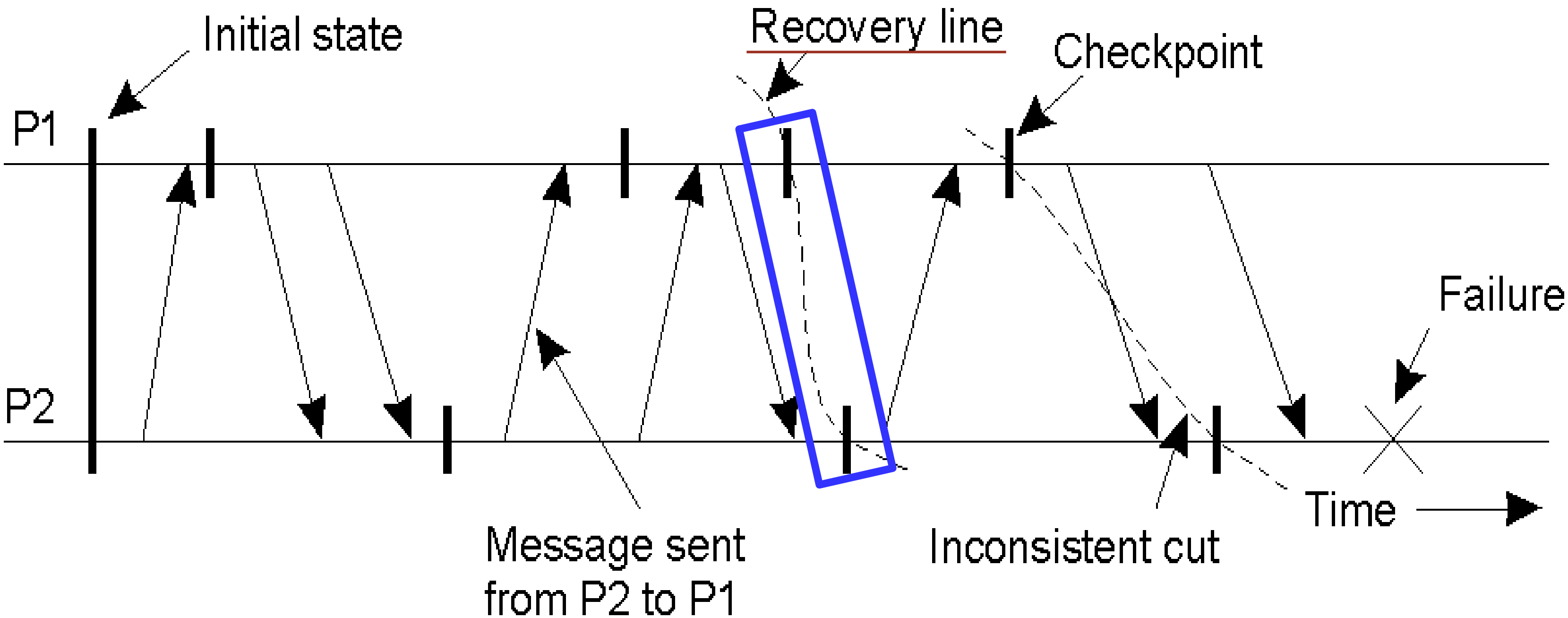


For example, reliable communication

a packet is lost → retransmission

Checkpointing

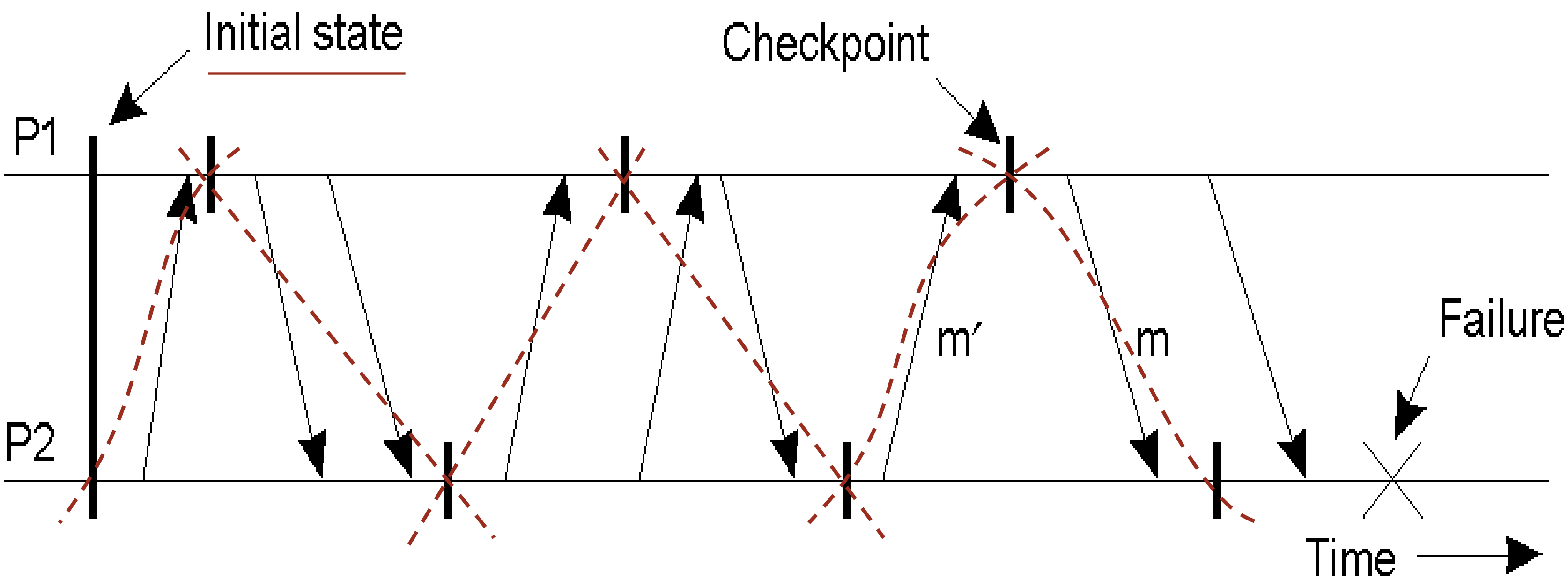
- System regularly saves its state onto stable storage



A recovery line.

- Recovery

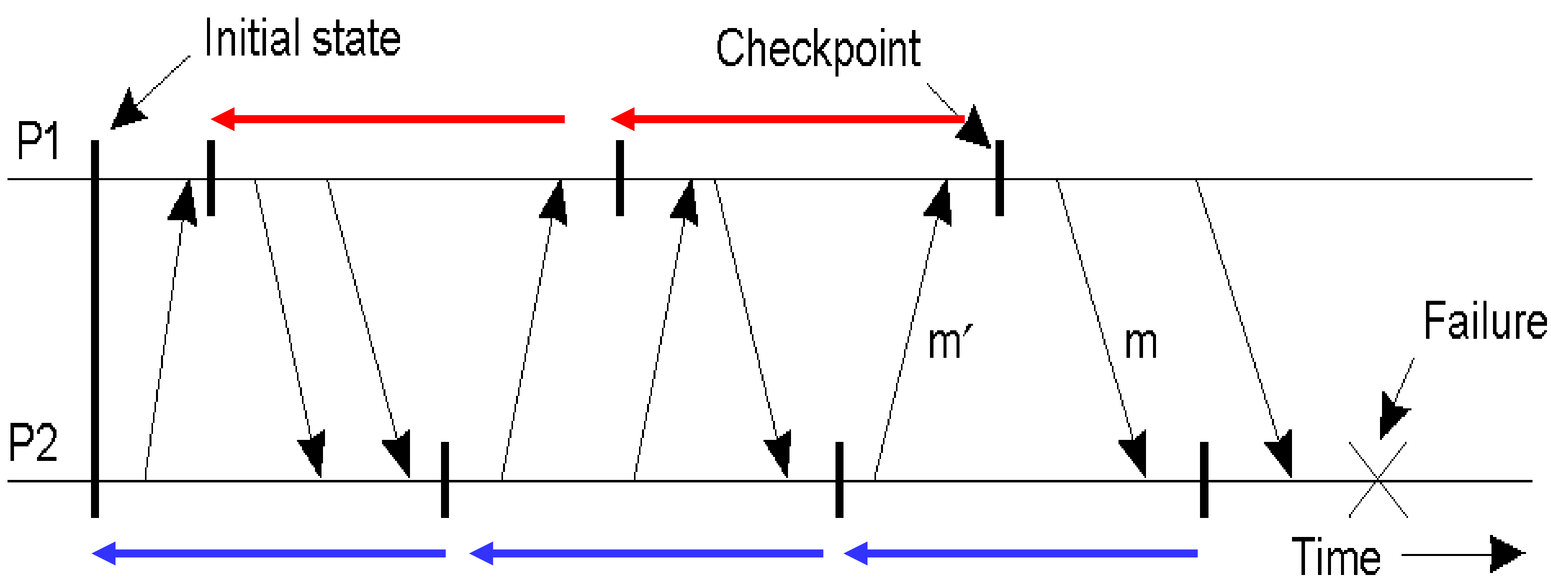
construct a consistent global state from local states.



To recover to most recently saved state, it requires that all processes coordinate checkpointing.

1) Independent checkpointing

- processes take local checkpoints independent of each other
- dependencies are recorded in such a way that processes can jointly roll back to a consistent global state



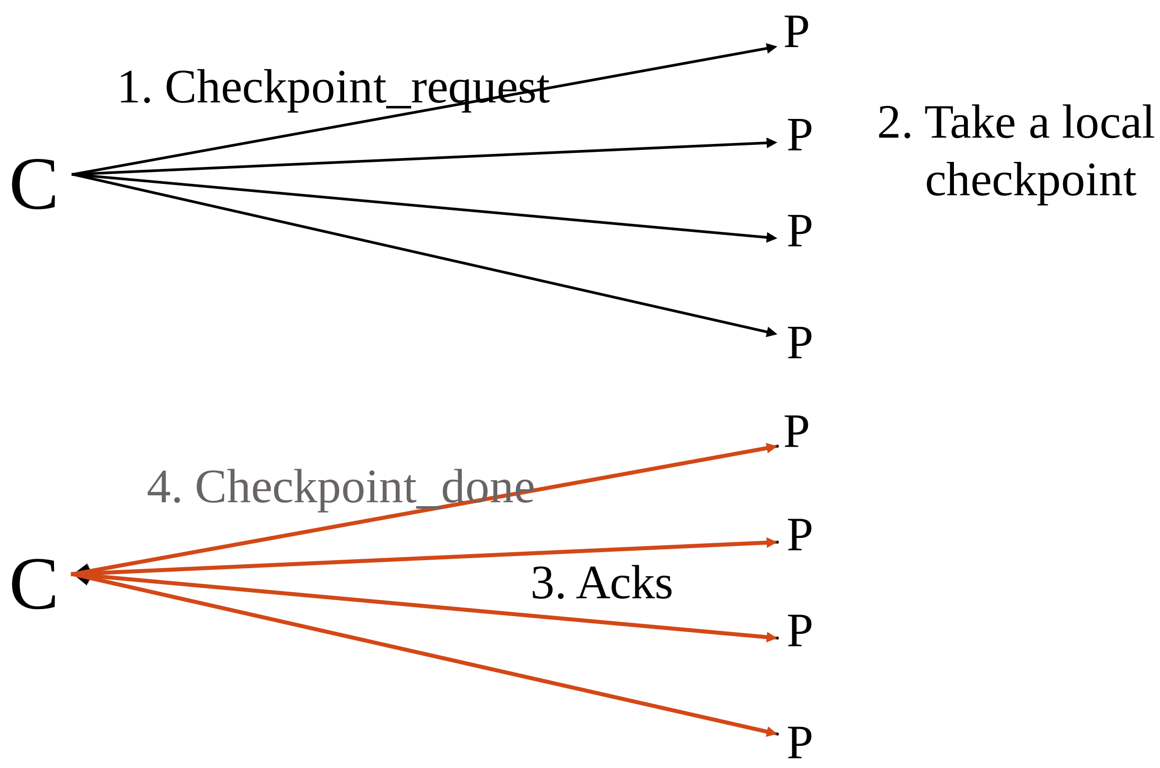
2) Coordinated checkpointing

All processes synchronize to jointly write their state to local stable storage which form a global consistent state.

Two algorithms:

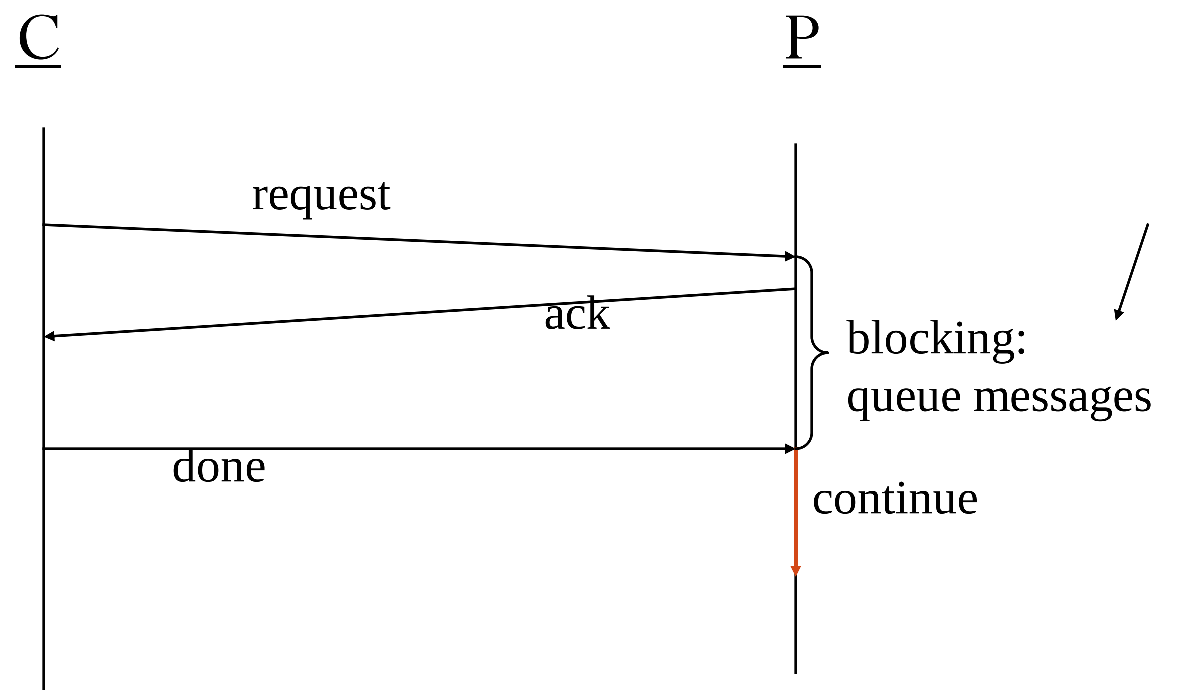
- distributed snapshot algorithm --- nonblocking one
- two-phase blocking protocol

Simple two-phase blocking protocol



Algorithm description

- A coordinator multicasts a Checkpoint_request message to all processes
- when a process receives such a message, it takes a local checkpoint, queue any subsequent message handed to it by the application it is executing, and acknowledges to the coordinator
- when the coordinator has received all acks, it multicasts a Checkpoint_done message to allow the (blocked) processes to continue



Explain that this approach will lead to a globally consistent state

Summary

- Concepts about faults
- How to improve dependability
- Two-army problem
- Byzantine agreement problem
- Reliable Multicast
- Distributed commit
- Recovery