Distributed Systems

Reliable Communication Mechanisms

WEEK 13: Reliable Messaging, Client-Server Communication, Group Communication, Message Ordering, Failure Handling.

Online Lecture Series - 13

Felix Edesa

Addis Ababa Science and Technology University

Course Outline: Reliable Communication Mechanisms (Week 13)

Topics We Will Cover

- **Reliable Messaging** — Introduction, importance of message delivery guarantees in distributed systems
- **Client-Server Communication** — Point-to-point reliability, RPC semantics.
- **Group Communication** — Multicast reliability, atomic multicast, consensus.
- **Message Ordering** — FIFO, causal, total ordering; why ordering matters for consistency
- **Failure Handling** — Detecting crashes, omission failures, rollback, retries, and recovery mechanisms.

Reliable Messaging — Importance

Why Reliable Messaging Matters

- Ensures **correctness and consistency** in distributed systems.
- Prevents **message loss** or duplication in critical applications.
- Provides foundation for **fault-tolerant systems** and consensus algorithms.
- Supports **synchronous and asynchronous communication** in distributed networks.
- Enables **robust monitoring and event logging**.

Examples

- Banking transactions: Missing messages can cause financial loss.
- Online auctions: Duplicate bids can corrupt results.

Message Delivery Guarantees

Delivery Semantics

- **At-most-once:** Delivered zero or one time.
- **At-least-once:** Delivered one or more times.
- **Exactly-once:** Delivered only once.
- Correct choice depends on **application requirements** for reliability and performance.

Practical Use

- Email: At-least-once to ensure delivery.
- Financial transfers: Exactly-once critical.
- Event logs: At-most-once acceptable.

Reliable Client-Server Communication

Concepts

- Communication between clients and servers must **handle failures**.
- Include **timeouts, retries, and acknowledgments**.
- Ensure **request-response integrity**.
- Maintain **session state consistency** across retries.
- Use **idempotent operations** to prevent duplicate processing.
- Implement **load balancing** to distribute client requests evenly.

Examples

- Web APIs retry requests after timeouts.
- Payment gateways ensure no duplicate charges.

Reliable Group Communication

Key Concepts

- Messages sent to a group of processes must be **consistently delivered**.
- Must handle **failures of individual group members**.
- **Atomic multicast** ensures either all or none receive the message.
- Supports **fault-tolerant distributed coordination**.

Practical Examples

- Distributed database replication.
- Multiplayer online games synchronizing state.
- Collaborative editing platforms like Google Docs.

Message Ordering Guarantees

Types of Ordering

- **FIFO Ordering:** Messages from the same sender are delivered in order.
- **Causal Ordering:** Messages delivered respecting causal dependencies.
- **Total Ordering:** All processes see all messages in the same order.
- Guarantees prevent **state inconsistencies** in distributed systems.

Examples

- FIFO: Chat messages from the same user.
- Causal: Event streams in social media feeds.
- Total: Distributed transactions in financial systems.

Reliable Messaging Mechanisms

Mechanisms

- **ACKs and NACKs** for message confirmation.
- **Sequence numbers** for detecting missing or duplicate messages.
- **Logging** to persist messages until confirmed.
- **Redundant paths** to improve fault tolerance.

Examples

- TCP protocol: ACKs and retransmissions.
- Kafka producer-consumer: Persistent logs and retries.
- MQTT in IoT: QoS levels 0,1,2.

Practical Problem — Lost Messages

Problem Scenario

- IoT network drops messages due to unstable connections.
- Critical sensor readings may be lost.
- Task: Ensure messages eventually reach the server reliably.
- Consider **buffering, retries, and acknowledgment protocols**.

Solution

- Implement retries with ACKs.
- Use sequence numbers to avoid duplicates.

Practical Problem — Message Ordering

Problem Scenario

- Online multiplayer game updates arrive out-of-order.
- Players see inconsistent state, affecting gameplay.
- Task: Maintain consistent game state for all players.
- Ensure all actions are applied in the same order for every client.
- Handle network latency and packet loss gracefully.
- Implement rollback or correction mechanisms for out-of-order updates.
- Use sequence numbers or timestamps to track message order.
- Consider causal dependencies between game events.

Solution

- Implement total ordering using sequence numbers or consensus.
- Use causal ordering to respect game event dependencies.

Practical Problem — Group Communication

Problem Scenario

- Distributed chat app: Messages sent to group may arrive inconsistently.
- Users see different message sequences, causing confusion.
- Task: Ensure all group members receive messages in the same order.

Solution

- Use atomic multicast or total order broadcast protocols.
- Synchronize group members to maintain message order.

Practical Problem — Client-Server Communication

Problem Scenario

- Web application fails when multiple clients send requests simultaneously.
- Some requests are lost or processed out-of-order.
- Task: Ensure reliable client-server communication under high load.

Solution

- Implement request queueing and ACKs.
- Use retry mechanisms for lost requests.

Client-Server Communication — Overview

Concepts

- Communication occurs via **direct point-to-point connections** between client and server.
- Ensures **request and response integrity** even under network failures.
- Provides **reliable service execution** and predictable response behavior.
- Supports **synchronous and asynchronous request handling**.
- Forms the basis for **RPC and microservices interactions**.

Examples

- HTTP request-response between web client and server.
- IoT device sending sensor data to cloud service.

Point-to-Point Reliability

Key Concepts

- Ensures each **request reaches the intended server**.
- Guarantees that **responses reach the client reliably**.
- Handles **message loss, duplication, and reordering**.
- Uses **ACK/NACK mechanisms** to confirm delivery.
- Provides foundation for **fault-tolerant applications**.

Examples

- TCP for reliable HTTP communication.
- IoT sensor data confirmed by cloud with ACKs.

Remote Procedure Call (RPC) Basics

Concepts

- RPC allows clients to **invoke procedures on remote servers**.
- Hides network communication details, making calls appear **local**.
- Includes **requests, responses, and error handling**.
- Reliability ensures **correct execution of remote operations**.
- Forms the backbone of **microservices and distributed systems**.

Examples

- Database remote procedure execution.
- File system access over NFS.
- Cloud microservice API calls.

RPC Semantics

Types of RPC Semantics

- **At-most-once:** Operation executes zero or one time.
- **At-least-once:** Operation may execute multiple times; requires idempotency.
- **Exactly-once:** Operation executes exactly once; ideal but difficult.
- Semantics choice depends on **failure assumptions and application needs**.
- Guarantees help maintain **data consistency across distributed systems**.

Examples

- Payment processing: Exactly-once required.
- Logging service: At-least-once acceptable.

Handling Failures in RPC

Techniques

- Detect server crashes using **timeouts**.
- Apply **retries with exponential backoff** for transient failures.
- Ensure operations are **idempotent** to prevent duplication.
- Maintain **request logs** for recovery and auditing.
- Use **failover servers** to maintain availability.

Examples

- Retry payment requests until acknowledgment.
- Cache database writes locally until confirmed by server.

Client-Side Reliability Enhancements

Techniques

- Maintain **unique request identifiers** to detect duplicates.
- Use **timeouts and retries** for unresponsive servers.
- Implement **local caching** for network failures.
- Preserve **session state consistency** across retries.
- Validate **response correctness** before committing results.

Examples

- Web browsers resubmit failed API calls.
- Mobile apps queue requests offline until server reconnects.

Server-Side Reliability Enhancements

Techniques

- Implement **idempotent request handling**.
- Persist incoming requests to **durable storage** for recovery.
- Use **replication and failover** for high availability.
- Monitor and recover **failed processes automatically**.
- Validate request **sequence numbers** to avoid duplicates.

Examples

- Distributed web servers with failover.
- Replicated microservices in cloud applications.

Practical Example — Client-Server RPC

Scenario

- Client requests a bank transfer from a server.
- Network may drop requests or responses.
- Task: Ensure **exactly-once execution** of the transaction.
- Use **timeouts, retries, and logging** for reliability.
- Maintain **transaction identifiers** to prevent duplicates.

Solution

- Assign unique transaction IDs to each request.
- Retry requests until acknowledgment is received.

Group Communication — Introduction

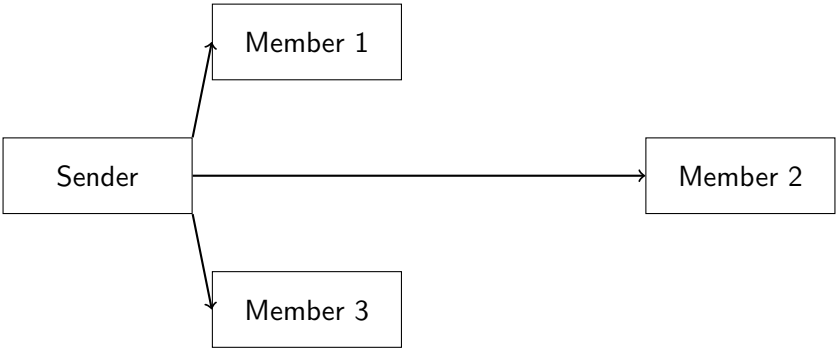
Overview

- Group communication involves sending messages to multiple processes simultaneously.
- Ensures that all group members receive messages consistently.
- Forms the foundation for distributed applications like chat apps, collaborative editing, and multiplayer games.

Key Concepts

- Multicast vs broadcast
- Reliable delivery to multiple recipients
- Coordination and consensus

Group Communication



Multicast Reliability

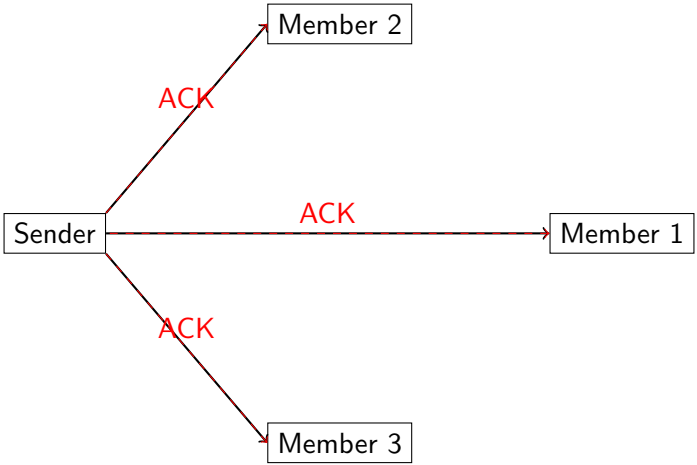
Reliable Multicast

- Ensures that all group members receive every message.
- Detects lost or duplicated messages.
- Can use ACK/NACK schemes for reliability.
- Supports retransmissions to handle network failures.

Benefits

- Prevents inconsistent group state.
- Essential for collaborative applications.
- Provides fault-tolerance in distributed systems.

Reliable Multicast



Atomic Multicast

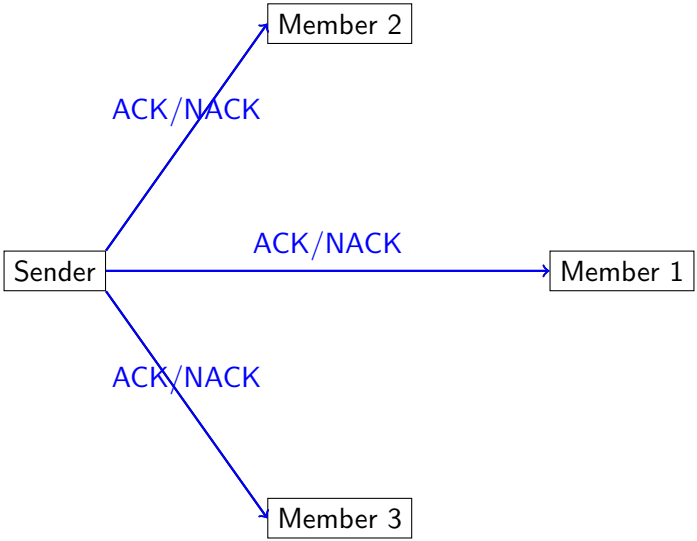
Concepts

- Ensures either all group members receive a message or none.
- Prevents partial updates that lead to inconsistent state.
- Often implemented with consensus protocols.

Use Cases

- Distributed databases
- Multiplayer games
- Collaborative editing platforms

Atomic Multicast



Consensus in Group Communication

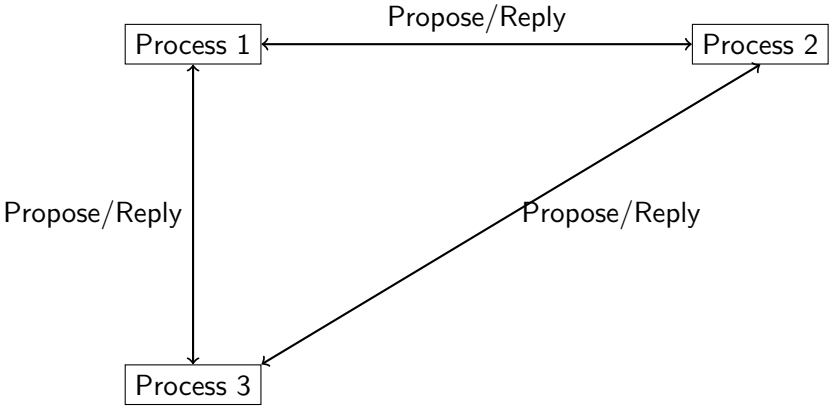
Concepts

- Consensus ensures all processes agree on a value or order.
- Essential for atomic multicast.
- Handles failures and network delays.
- Guarantees consistent group state even in distributed settings.

Applications

- Leader election
- Distributed commit
- Blockchain consensus

Consensus Protocol



Multicast Failure Handling

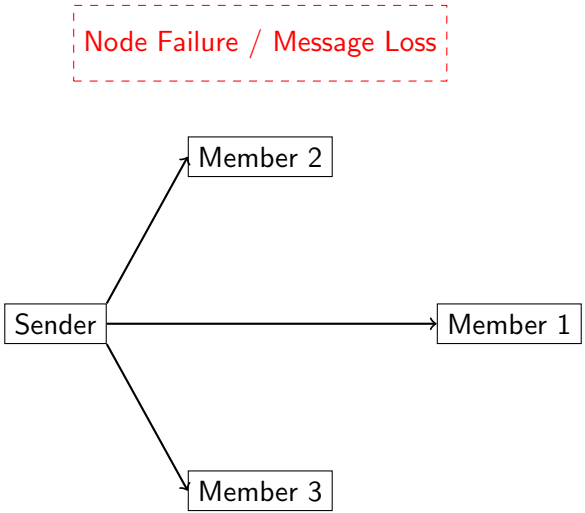
Key Points

- Detect failed nodes or lost messages.
- Retransmit messages or reconfigure group.
- Ensures consistency despite failures.
- Often integrated with consensus for reliability.

Benefits

- Maintains group state integrity.
- Minimizes message loss.
- Supports fault-tolerant distributed applications.

Multicast Failure Handling



Message Ordering — Introduction

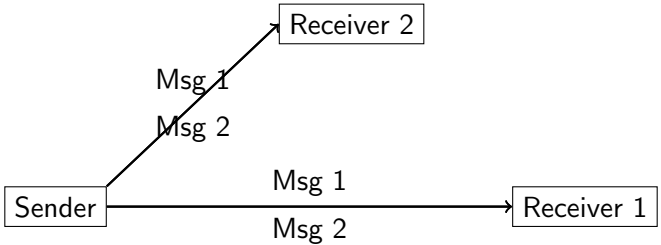
Overview

- Message ordering ensures that messages are delivered in a predictable sequence.
- Critical for maintaining consistency in distributed systems.
- Improper ordering can lead to inconsistent state and errors.

Key Concepts

- FIFO ordering
- Causal ordering
- Total ordering

Message Ordering



FIFO Ordering

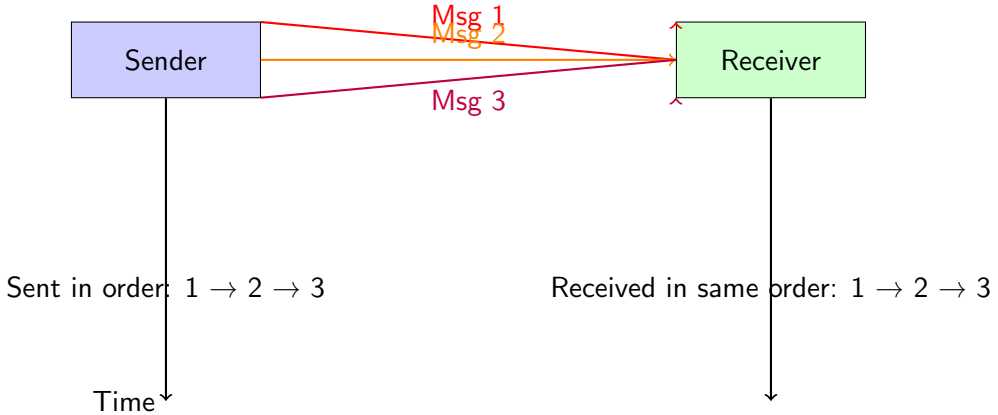
Concepts

- FIFO (First-In-First-Out) ensures messages from a single sender are received in the order sent.
- Simple and efficient for per-sender consistency.
- Does not enforce ordering between different senders.

Use Cases

- Chat messages per user
- Sequential logging per process

FIFO Ordering — Expressive



Causal Ordering

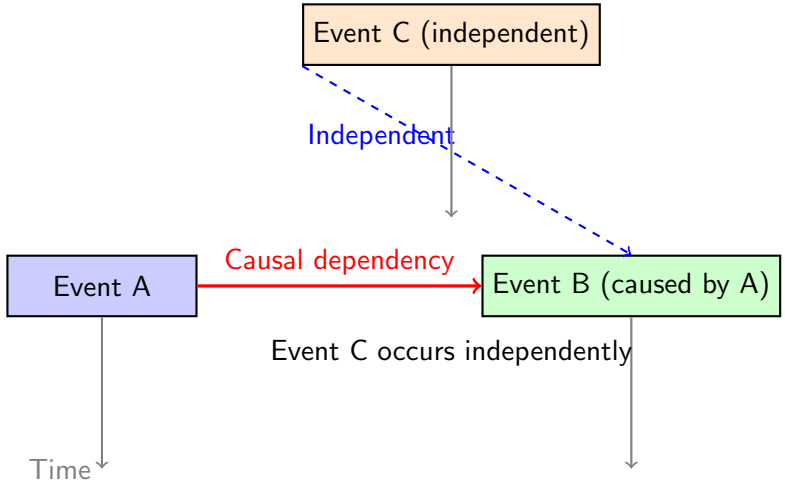
Concepts

- Causal ordering ensures that messages are delivered respecting causal dependencies.
- If event A causes event B, all receivers see A before B.
- Useful for collaborative applications and event-driven systems.

Use Cases

- Distributed collaborative editing
- Social media event streams

Causal Ordering — Expressive Diagram



Total Ordering

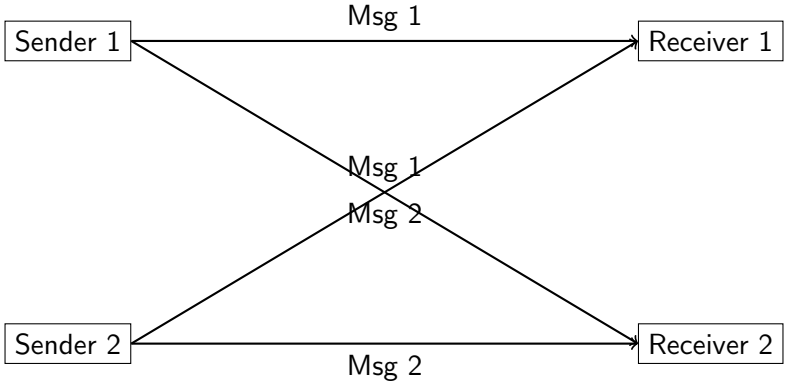
Concepts

- Total ordering ensures all messages are seen by all processes in the same order.
- Combines causal and global sequencing guarantees.
- Crucial for consistent state in distributed transactions.

Use Cases

- Distributed databases
- Financial transaction systems

Total Ordering — Diagram



Why Message Ordering Matters

Key Points

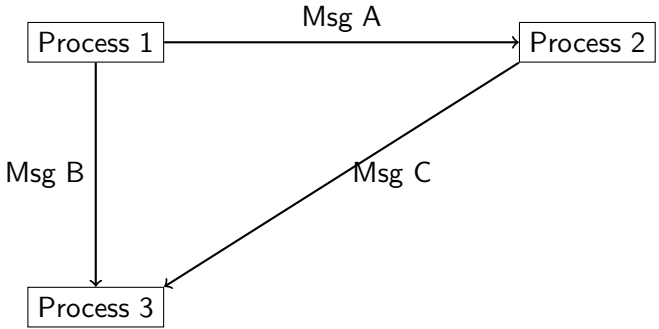
- Prevents inconsistent system states.
- Ensures reliable execution of distributed protocols.
- Supports coordination between processes.
- Reduces conflicts in collaborative applications.

Applications

- Distributed databases
- Multiplayer online games
- Real-time collaboration tools

Message Ordering Importance — Diagram

Incorrect Ordering Leads to Conflict



Failure Handling — Introduction

Overview

- Failure handling is crucial for maintaining **system reliability**.
- Types of failures include **crashes, omissions, and network errors**.
- Mechanisms like **retries, rollback, and recovery** help mitigate failures.
- Proper detection ensures **minimal disruption to services**.

Examples

- Web servers handling simultaneous crashes.
- IoT sensors failing to send readings.
- Databases recovering from aborted transactions.

Detecting Crashes

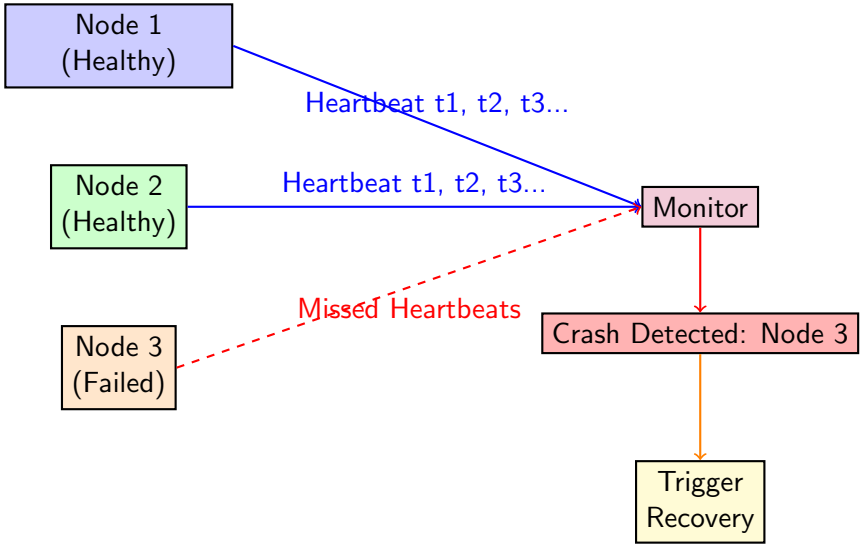
Key Points

- Crash detection monitors **unresponsive nodes**.
- Uses **heartbeats, timeouts, and health checks**.
- Enables **fast failure notification** to other system components.
- Critical for **high-availability systems**.

Examples

- Cluster node stops responding to ping.
- Web service fails to send heartbeat signals.
- IoT device stops reporting sensor data.

Crash Detection



Omission Failures

Key Points

- Omission failures occur when a process **fails to send or receive messages**.
- Detection can use **timeouts or retries**.
- Mitigation strategies include **redundant messaging and acknowledgment**.
- Helps maintain **system consistency**.

Examples

- Lost packet in network communication.
- Failed sensor data transmission.
- Missing database replication messages.

Rollback Mechanisms

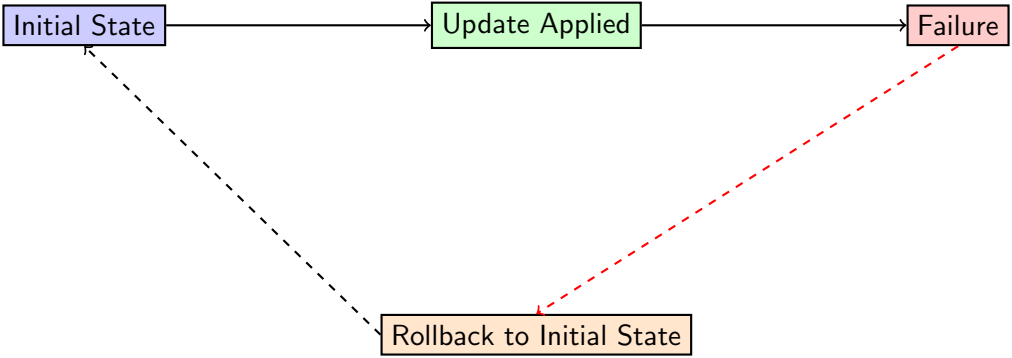
Key Points

- Rollback restores **system to a previous consistent state**.
- Essential for **transactional systems**.
- Can use **checkpoints and logs**.
- Minimizes impact of **failures on system operations**.

Examples

- Database transaction rollback after failure.
- Application state restored after crash.

Rollback


Initial State
Update Applied
Failure
Rollback to Initial State

Retries

Key Points

- Retries are **re-sending failed requests or operations**.
- Helps overcome **temporary network or server issues**.
- Can be combined with **timeouts and acknowledgments**.
- Limits impact of **transient failures**.

Examples

- API request retry after timeout.
- Resending message in unreliable network.

Recovery Mechanisms

Key Points

- Recovery restores **system to normal operation after failure**.
- Combines **rollback, retries, and redundant systems**.
- Supports **high-availability and fault tolerance**.
- Essential for **mission-critical applications**.

Examples

- Database recovery after crash.
- Web service resuming after temporary downtime.

Questions?

Thank you for your attention!

References

Suggested References

1. M. van Steen and A. S. Tanenbaum, *Distributed Systems*, 4th ed., Version 4.02, February 2024.
2. G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, *Distributed Systems: Concepts and Design*, 5th ed., Addison-Wesley, 2012.
3. N. Lynch, *Distributed Algorithms*, 2nd ed., Morgan Kaufmann, 1996.
4. R. E. Bryant and D. O'Hallaron, *Computer Systems: A Programmer's Perspective*, 4th ed., Pearson, 2024.