Distributed Systems

Replica Management

WEEK 11: Replication Strategies, Replica Consistency, Data Synchronization.

Online Lecture Series - 11

Felix Edesa

Addis Ababa Science and Technology University

Course Outline: Replica Management (Week 11)

Topics We Will Cover

- **Replication Basics** — Purpose and advantages in distributed environments.
- **Replication Strategies** — Primary–backup and multi-primary approaches.
- **Replica Consistency** — Strong vs. eventual consistency trade-offs.
- **Data Synchronization** — Methods for keeping replicas up-to-date.
- **Conflict Handling** — Managing concurrent updates and version control.
- **Fault Recovery** — Techniques for failure detection and re-synchronization.

Replication Basics — Introduction

Definition and Purpose

- Replication is the process of creating multiple copies of data across different nodes.
- Ensures high availability: if one node fails, others can serve requests.
- Improves performance by allowing access to the nearest replica.
- Enhances fault tolerance and disaster recovery.
- Supports load balancing and scalability.

Key Examples

- Cloud storage: Google Drive, AWS S3 replicate.
- Media streaming: Netflix stores content near users to reduce latency.
- Banking systems: Multiple replicas prevent downtime.

Benefits of Replication

Advantages

- High availability: replicas provide continuous access during failures.
- Performance: read operations can be served from nearest replica.
- Fault tolerance: reduces risk of data loss.
- Load balancing: distributes requests among multiple replicas.
- Disaster recovery: replication across regions ensures backup during catastrophes.

Use Cases

- E-commerce.
- Social media.
- Healthcare.

Replication Strategies

Main Approaches

- **Synchronous Replication** — Updates propagate to all replicas before acknowledging the client.
- **Asynchronous Replication** — Updates are sent to replicas after acknowledging the client.
- **Primary-Backup** — One node handles all writes; others replicate changes.
- **Multi-Primary (Active)** — Multiple nodes can process writes simultaneously; requires conflict resolution.

Examples

- Primary-Backup: MySQL replication with a master and slaves.
- Multi-Primary: Cassandra database.
- Async replication: Dropbox syncing changes across devices.

Replica Consistency Models

Consistency Models

- **Strong Consistency** — All replicas reflect the latest write immediately.
- **Eventual Consistency** — Replicas eventually converge to the same state.
- **Weak Consistency** — No guarantees; replicas may be inconsistent temporarily.
- Choosing the model depends on application requirements for latency vs. correctness.

Examples

- Strong: Banking transaction systems.
- Eventual: DNS replication, social media feeds.
- Weak: Caching systems in content delivery networks.

Data Synchronization Mechanisms

Techniques

- **State-Based (Push/Pull)** — Replicas exchange entire state periodically.
- **Operation-Based** — Only operations/updates are propagated.
- **Delta-Based Synchronization** — Only changes/differences are sent between replicas.
- Selection depends on network cost, data size, and update frequency.

Practical Uses

- State-based: Git repository syncing.
- Operation-based: Cassandra replication logs.
- Delta-based: Dropbox file synchronization.

Conflict Detection and Resolution

Handling Conflicts

- Conflicts occur when multiple replicas are updated concurrently.
- Version vectors or timestamps help identify conflicting updates.
- Resolution strategies: last-write-wins, merge functions, or custom application logic.

Examples

- Cassandra resolves with last-write-wins.
- Git uses merge strategies for conflicts.
- Collaborative tools (Google Docs) merge edits in real-time.

Fault Tolerance and Recovery

Recovery Mechanisms

- Detecting replica or node failures automatically.
- Re-synchronization of lost updates after recovery.
- Reintegration of recovered nodes without disrupting the system.

Use Cases

- Database cluster failover (MySQL, PostgreSQL).
- Cloud storage nodes re-sync after outages.
- Distributed file systems like HDFS maintain availability.

Practical Use Case: E-commerce Replication

Scenario

- Global online store replicates inventory across regional data centers.
- Customers access the nearest replica for fast response.
- Updates propagate asynchronously to all regions to maintain consistency.

Key Points

- Reduced latency improves customer experience.
- Failures in one region do not affect others.
- Inventory conflicts handled via versioning.

Practical Use Case: Cloud Storage

Scenario

- Data replicated across multiple data centers worldwide.
- Users experience high availability and low latency.
- Synchronization ensures eventual consistency of updates.

Key Examples

- AWS S3 cross-region replication.
- Google Drive replication for collaborative files.
- Dropbox replication across devices.

Summary and Key Takeaways

Takeaways

- Replication ensures availability, performance, and fault tolerance.
- Consistency models determine how up-to-date replicas are.
- Synchronization and conflict resolution maintain data integrity.
- Selecting the right replication strategy depends on system requirements and trade-offs.

Replication in Databases

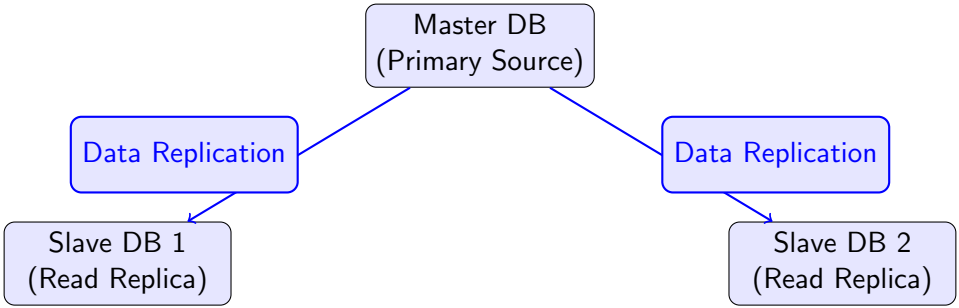
Overview

- Databases use replication to increase reliability, performance, and fault tolerance.
- Common strategies include Master-Slave, Multi-Master, and Cluster Replication.
- Supports reporting, backups, high-availability, and disaster recovery.
- Industrial impact: ensures uninterrupted operations for e-commerce, banking, and telecom systems.

Examples

- MySQL Master-Slave replication for e-commerce platforms.
- PostgreSQL streaming replication in banking systems.
- Cassandra Multi-Master replication in IoT telemetry.

Replication in Databases



Replication in Cloud Storage

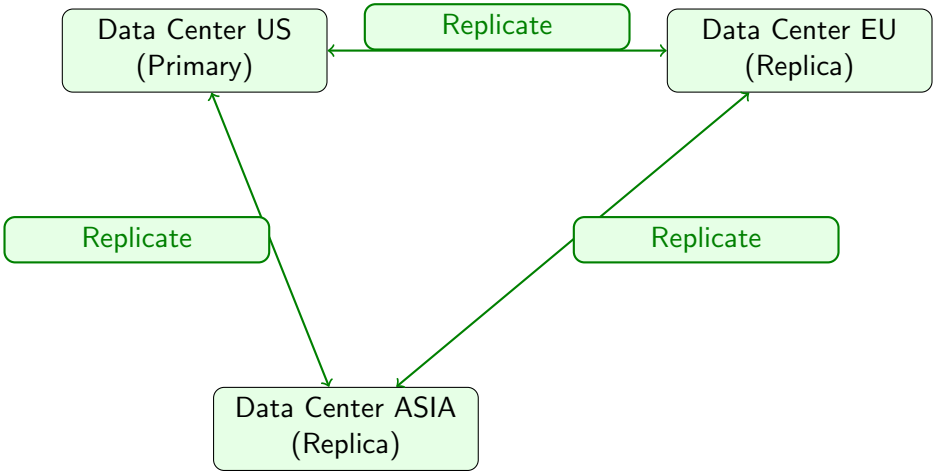
Purpose

- Ensures files and objects are available across multiple regions.
- Reduces latency by serving content from the nearest data center.
- Provides resilience against regional failures or outages.
- Supports collaboration, real-time editing, and backup strategies in industrial applications.

Examples

- AWS S3 cross-region replication.
- Google Drive multi-region replication.
- Dropbox synchronizes files across devices and regions.

Replication in Cloud Storage



Replication in Media Streaming

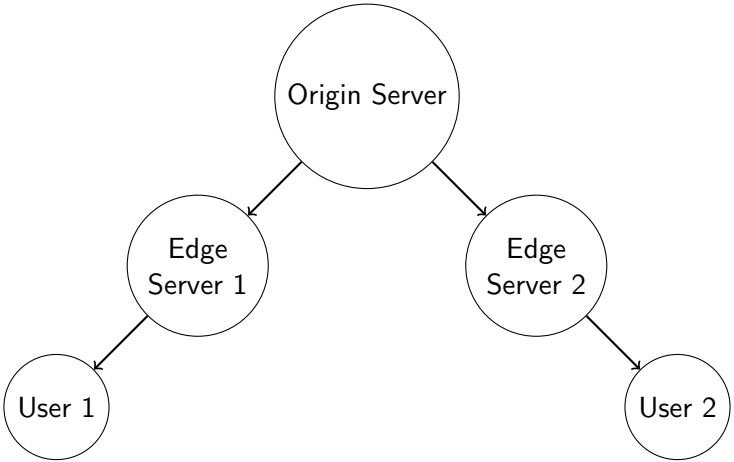
Use Case

- Media content is replicated across Content Delivery Networks (CDNs) to improve streaming quality.
- Reduces buffering, latency, and load on origin servers.
- Handles high traffic during events, premieres, or live broadcasts.

Industrial Examples

- Netflix replicates video content across global CDNs.
- YouTube edge servers replicate popular videos.
- Spotify replicates audio tracks across regions for low-latency playback.

Replication in Media Streaming — Diagram



Replication in Industrial IoT

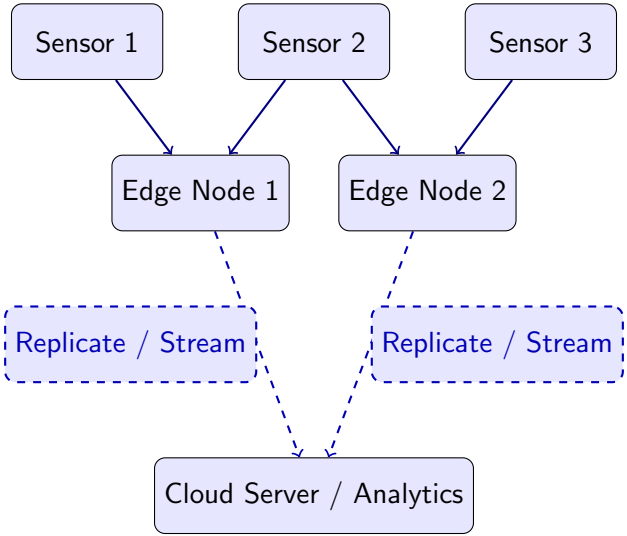
Purpose

- IoT devices replicate sensor data to cloud or edge servers for analytics.
- Ensures fault-tolerance and real-time monitoring.
- Historical data is safely stored for predictive maintenance.

Examples

- Manufacturing plants: sensors replicate data to monitoring systems.
- Smart grids: energy usage data replicated for analytics.
- Automotive telemetry: data replicated to cloud.

Replication in Industrial IoT



Strong vs Eventual Consistency — Trade-offs

Overview

- Strong consistency: ensures all replicas are immediately up-to-date; may increase latency.
- Eventual consistency: allows temporary differences; improves availability and reduces write latency.
- Trade-offs depend on application needs — consistency, latency, and availability.

Industrial Considerations

- Banking: prefer strong consistency for transactions.
- Social media IoT: eventual consistency acceptable to scale and reduce latency.

Strong vs Eventual Consistency

Hybrid Consistency Models

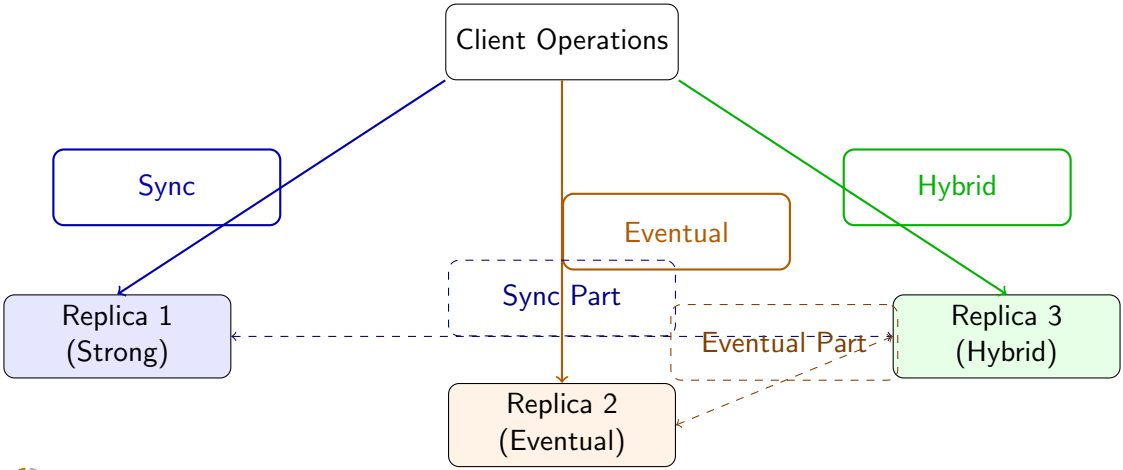
Overview

- Systems may combine strong and eventual consistency depending on operation type.
- Example: Amazon DynamoDB — eventual consistency by default, optional strong consistency reads.
- Balances latency, availability, and accuracy based on business requirements.

Industrial Examples

- E-commerce: critical writes (inventory) use strong consistency; non-critical reads (catalog) use eventual consistency.
- Financial dashboards

Hybrid Consistency Models — Diagram



Replica Consistency — Summary

Key Points

- Strong consistency ensures immediate uniformity across replicas — critical for transactional integrity.
- Eventual consistency improves availability and reduces latency but allows temporary inconsistencies.
- Hybrid models offer flexibility by applying strong or eventual consistency based on operation criticality.
- Choice depends on trade-offs between consistency, latency, and availability.

Industrial Takeaways

- Financial systems: strong consistency dominates.
- E-commerce social media: mix of strong and eventual for performance.
- IoT distributed systems.

Data Synchronization — Overview

Concept

- Data synchronization ensures replicas maintain consistent and up-to-date data.
- Critical for distributed databases, cloud storage, and IoT networks.
- Methods vary based on latency tolerance, operation criticality, and system scale.
- Synchronization can be push-based, pull-based, or hybrid.

Industrial Examples

- Cloud storage: Dropbox and Google Drive synchronize files across devices.
- IoT sensor networks synchronize readings.
- E-commerce platforms synchronize inventory.

Push-Based Synchronization

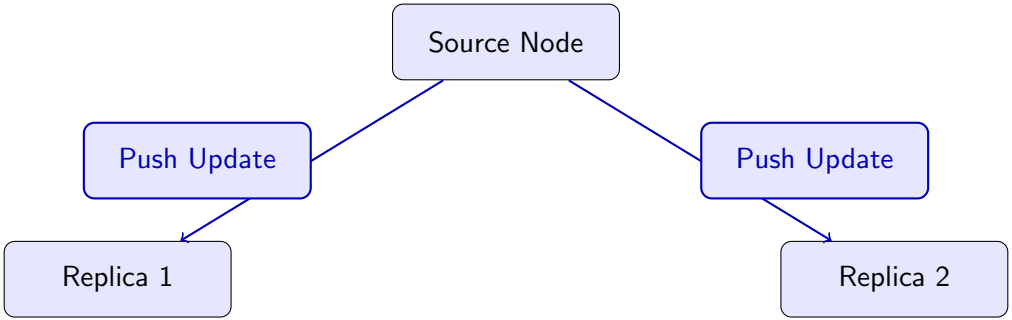
Overview

- The source actively sends updates to all replicas immediately after changes.
- Ensures faster propagation of changes.
- May increase network load if updates are frequent.

Industrial Examples

- Financial transaction systems pushing updates to all branch databases.
- Live stock trading platforms synchronize prices instantly.

Push-Based Synchronization



Pull-Based Synchronization

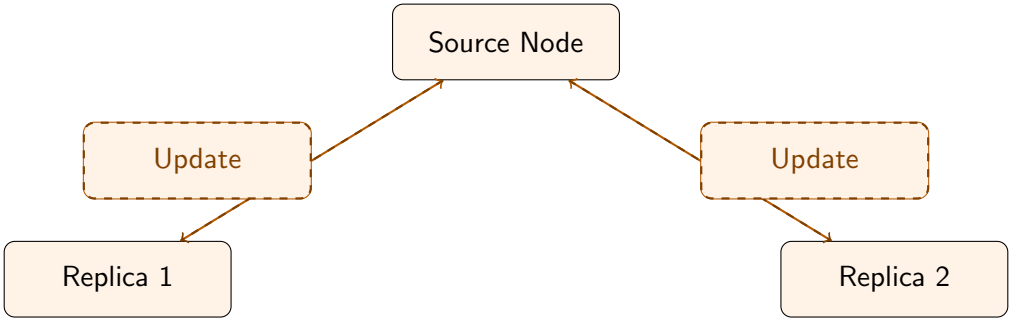
Overview

- Replicas periodically request updates from the source node.
- Reduces network usage during low-activity periods.
- May cause temporary inconsistencies between replicas.

Industrial Examples

- Content Delivery Networks fetching new videos or images.
- IoT devices pulling configuration updates from central servers.

Pull-Based Synchronization



Hybrid Synchronization

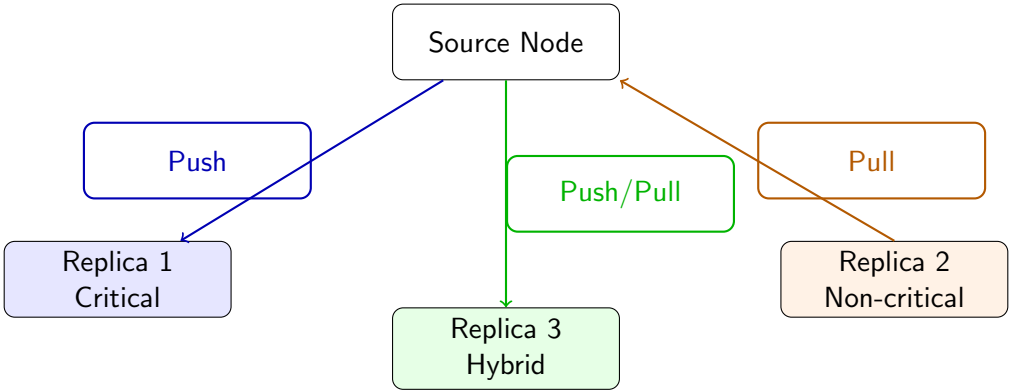
Overview

- Combines push and pull methods depending on operation criticality.
- Critical updates are pushed immediately; non-critical updates are pulled periodically.
- Balances network load, latency, and consistency requirements.

Industrial Examples

- E-commerce platforms: inventory critical updates pushed, catalog changes pulled.
- Distributed analytics: important metrics pushed; summary data pulled.

Hybrid Synchronization



Event-Driven Synchronization

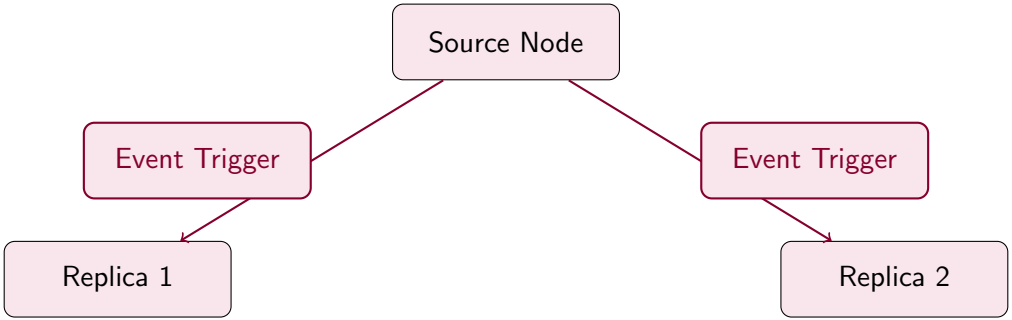
Overview

- Updates are triggered by specific events rather than fixed intervals.
- Reduces unnecessary network traffic.
- Suitable for time-sensitive applications where changes are sporadic.

Industrial Examples

- IoT alarms triggering immediate sync to central servers.
- Collaborative editing software (e.g., Google Docs) synchronizing edits.

Event-Driven Synchronization



Conflict Handling — Overview

Concept

- Conflict handling ensures data consistency when multiple updates occur concurrently on replicas.
- Critical in distributed databases, cloud storage, and collaborative applications.
- Involves detecting conflicts, resolving them automatically or manually, and maintaining version control.

Industrial Examples

- Version control systems.
- Collaborative document editors, manage simultaneous edits.
- Distributed databases use conflict resolution strategies.

Conflict Types

Overview

- **Write-Write Conflicts**: Two replicas update the same record simultaneously.
- **Read-Write Conflicts**: A read occurs while another replica is writing the same data.
- Conflicts may lead to data loss or inconsistency if not properly resolved.

Industrial Examples

- Inventory systems where multiple warehouses update stock simultaneously.
- Collaborative coding platforms with multiple developers editing the same file.

Conflict Detection Methods

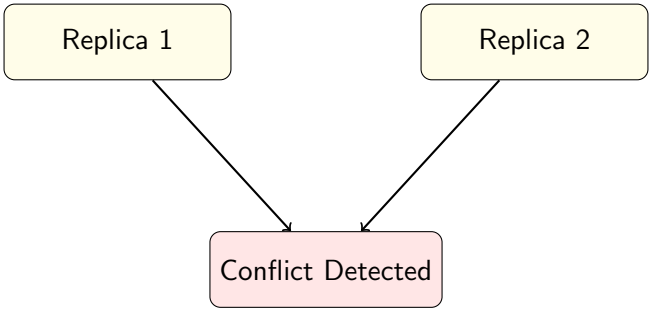
Overview

- **Version Vectors**: Track updates using timestamps or counters per replica.
- **Operational Transformation**: Detects concurrent edits and transforms operations to avoid conflicts.
- **Last Write Wins (LWW)**: Resolves conflicts by accepting the latest timestamped write.

Industrial Examples

- Git uses commit hashes and timestamps to detect conflicts.
- Cloud storage apps detect file version conflicts and prompt users.

Conflict Detection



Conflict Resolution Strategies

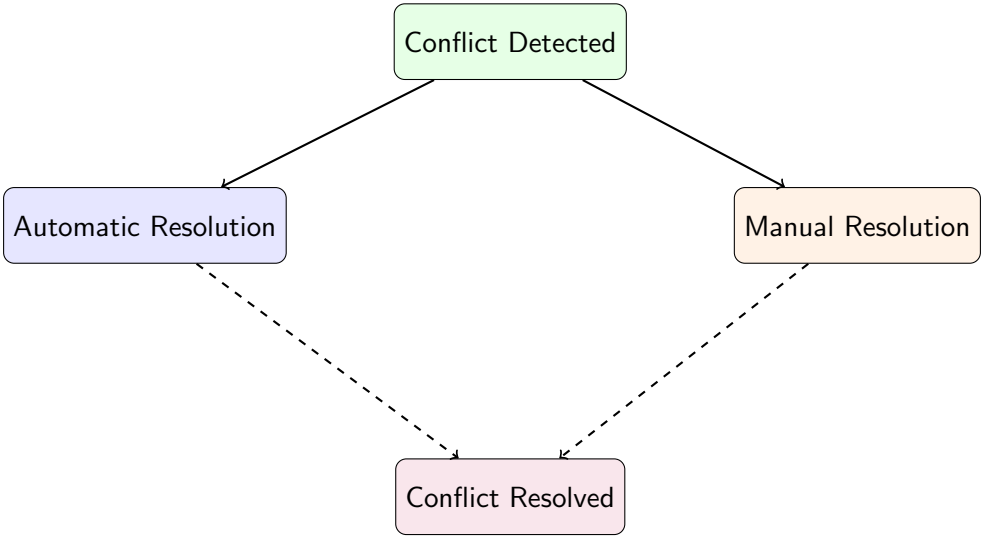
Overview

- **Automatic Resolution**: System resolves conflicts using rules (e.g., LWW, merge functions).
- **Manual Resolution**: User intervention is required to choose correct data.
- **Hybrid Resolution**: Combines automatic and manual approaches based on criticality.

Industrial Examples

- Database replication using LWW for logs.
- Collaborative editing prompting users to resolve merge conflicts.
- E-commerce inventory conflicts.

Conflict Resolution



Version Control Mechanisms

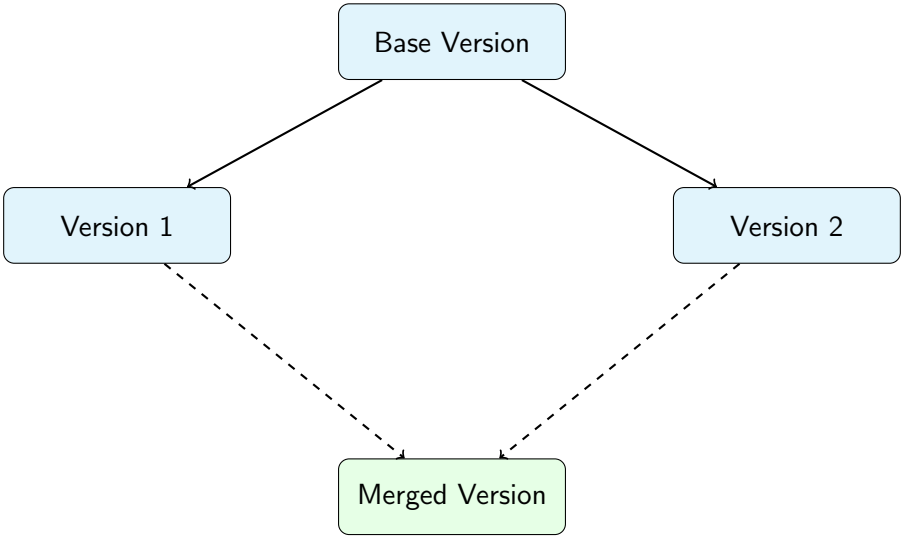
Overview

- **Centralized Versioning**: All replicas maintain a single authoritative version.
- **Distributed Versioning**: Each replica can create versions; conflicts handled via reconciliation.
- Tracks changes and allows rollback if necessary.

Industrial Examples

- Git (distributed) and SVN (centralized) for source code.
- Cloud storage file versioning to restore previous versions.

Version Control



Conflict Handling — Industrial Takeaways

Key Points

- Proper conflict detection prevents data corruption in distributed systems.
- Resolution strategies should balance automation and manual intervention based on criticality.
- Version control ensures traceability and rollback capabilities.

Industrial Applications

- Git and cloud storage systems.
- Distributed databases in financial, IoT, and e-commerce platforms.

Fault Recovery — Overview

Concept

- Fault recovery ensures systems can detect failures and restore normal operation.
- Critical in distributed databases, cloud services, and industrial IoT systems.
- Involves failure detection, logging, rollback, and re-synchronization of replicas.

Industrial Examples

- Database servers automatically detect crashes and restart.
- Cloud storage systems restore lost replicas using backups.

Failure Detection Techniques

Overview

- **Heartbeat Monitoring**: Periodic signals to check if a node is alive.
- **Timeout Mechanisms**: Detect non-responsiveness after a predefined interval.
- **Logging and Alerts**: Track system events to detect anomalies or crashes.

Practical Examples

- Kubernetes monitors container health via heartbeat probes.
- Cloud databases use timeouts to mark unhealthy replicas.

Recovery Techniques

Overview

- **Rollback**: Revert data to the last known consistent state.
- **Failover**: Switch operations to a backup replica or server.
- **Checkpointing**: Save periodic snapshots to restore system state after failure.

Practical Examples

- Database rollback after a failed transaction.
- Cloud services automatically redirect traffic to healthy servers.
- IoT gateways restore state from the last checkpoint.

Re-synchronization Techniques

Overview

- **Data Replay**: Apply missed updates to bring replicas up-to-date.
- **Merge Operations**: Combine divergent states while resolving conflicts.
- **Consistency Check**: Verify all replicas have synchronized data.

Practical Examples

- Distributed databases replay transaction logs.
- Cloud file systems merge changes from multiple replicas.
- IoT sensor data synchronized.

Fault Recovery — Industrial Takeaways

Key Points

- Early failure detection minimizes downtime and prevents data loss.
- Checkpointing and rollback strategies improve system resilience.
- Re-synchronization ensures all replicas remain consistent after recovery.
- Combining automated and manual recovery methods enhances reliability.

Industrial Applications

- Cloud computing platforms for business-critical services.
- Financial systems requiring minimal transaction downtime.
- Industrial IoT networks with multiple sensor nodes.

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.