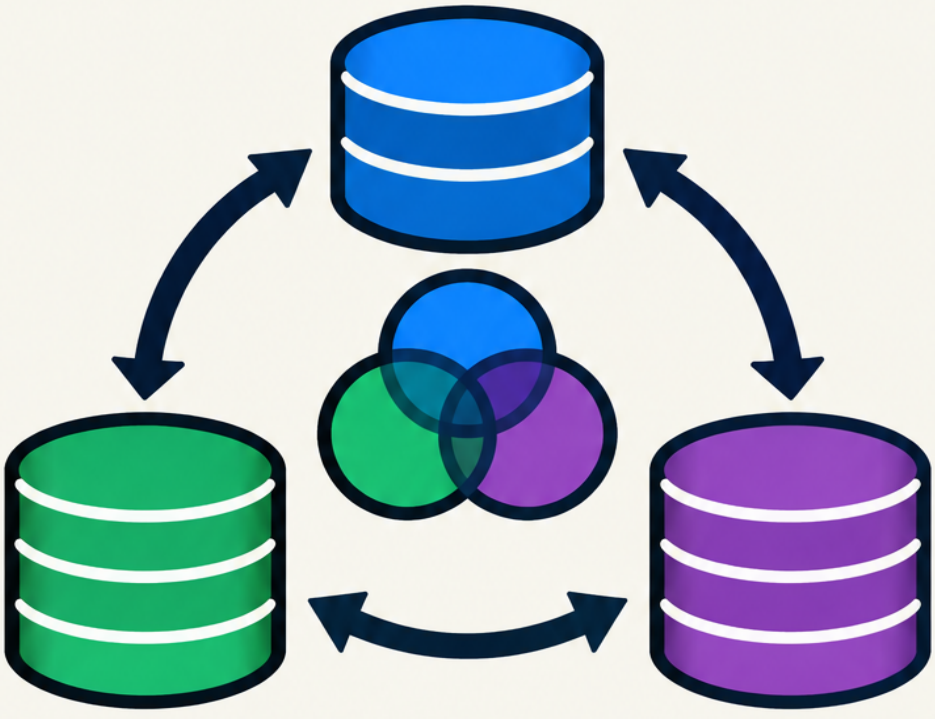
Conflict-Free Replicated Data Types

A Practical Guide to Building Distributed Collaborative Systems



LUKE MITCHELL

Conflict-Free Replicated Data Types

A Practical Guide to Building Distributed Collaborative Systems

Steve Publications

This book is available at
https://leanpub.com/conflict-freereplicateddatatypes

This version was published on 2026-09-22

Contents

A Practical Guide to Building Distributed Collaborative Systems

Conflict-free replicated data types (CRDTs) are the mathematical backbone of modern offline-first and collaborative applications. This book gives you everything you need to understand them from first principles and build real production systems. Starting with the distributed-systems foundations you need, it develops the core CRDT concepts rigorously, then takes you through complete, runnable implementations of every major data type. You will progressively build a full distributed collaboration system, then confront the production concerns most books ignore: metadata growth, garbage collection, security, scaling, and failure modes. By the end, you will not only understand why CRDTs work but also when to use them, when not to, and how to make them reliable at scale.

Introduction

Imagine this scenario. A design team in Berlin is collaborating on a document with colleagues in San Francisco. The Berlin internet goes down for twenty minutes while the rest of the world continues to work. When connectivity returns, every change must merge seamlessly without anyone noticing the interruption. No conflicts. No lost work. No "which version is right?" conversations.

Now imagine scaling this to tens of thousands of concurrent users editing different parts of a shared document, some on slow mobile connections, some offline for hours.

Now imagine building this without a central coordinator telling everyone who wins when two people try to change the same thing.

This is exactly the problem that conflict-free replicated data types solve. CRDTs are the reason your Notes app syncs across devices without drama. They power collaborative editors that feel real-time even when the network is terrible. They let distributed databases accept writes anywhere without expensive consensus rounds. They are, in practical terms, the most important data structures in modern distributed computing that most engineers have never heard of.

What This Book Is About

CRDTs are data structures replicated across multiple computers in a network with a remarkably strong guarantee: every replica can be updated independently and concurrently, without coordination, and all replicas are mathematically guaranteed to converge to the same state once they have seen the same updates.

That is a mouthful. Let me unpack it.

First, no coordination. When you modify a CRDT on one machine, you do not ask permission from anyone. You do not wait for a lock. You do not participate in a consensus round. The operation happens immediately,

locally, without any network round trip. This is why CRDT-based systems feel instantaneous and why they work offline.

Second, guaranteed convergence. Despite the lack of coordination, the system does not descend into chaos. Every replica eventually agrees on the same state. This is not hope. This is a mathematical property of how the data type is designed, proven using lattice theory and algebraic structures.

Third, no data loss (in the sense that the data type guarantees). CRDTs do not throw away concurrent updates. If two users both increment a counter while offline, both increments will be reflected in the final value. If two users both edit a document, both edits will appear in the final document. The merge is deterministic and preserves all inputs, modulo the semantics you choose for the data type.

This book takes you through how CRDTs achieve these properties, why the underlying mathematics matters, and how to implement them in real systems. Every chapter builds on the previous one, and the code examples are designed to accumulate into a working distributed system you can run, modify, and extend.

Who This Book Is For

This book assumes you are an experienced software engineer comfortable with data structures, distributed systems concepts, and production-grade code. You should be able to reason about concurrency, understand the difference between eventual and strong consistency, and feel at home with a modern programming language. TypeScript will be our primary language for code examples because it is widely used, has excellent tooling, and aligns with real-world CRDT libraries like Yjs and Automerge.

If you are reading this and already know what a join-semilattice is and can implement a gossip protocol from scratch, you might still find value here. The book goes deep into production concerns, real-world systems, and the difficult trade-offs that papers tend to leave out. If you are reading this and have heard the term “CRDT” but are not sure what it means or why it matters, this book will take you from zero to confident.

How This Book Is Organized

The book is structured as a progressive journey from foundations to mastery. Here is how it flows.

The first part establishes the distributed systems background you need. You cannot understand CRDTs without understanding what problem they solve, and you cannot appreciate that problem without understanding replication, consistency models, causality, and the fundamental trade-offs of distributed computing. We cover Lamport timestamps, vector clocks, the CAP theorem, and eventually consistency not as abstract concepts but as concrete problems you will encounter.

The second part introduces CRDTs formally and develops the mathematical properties that guarantee convergence. We define what a CRDT actually is, what strong eventual consistency means, and why join-semilattices and commutativity are not just academic curiosities but the exact conditions you need for coordination-free replication.

The third part is where you start building. We implement CRDTs one type at a time, starting with the simplest and working toward the hardest. Registers, counters, sets, maps, and finally sequences and text editors. For each type, we go from intuitive explanation to formal model to complete, runnable code with tests. We also implement a synchronization protocol that ties them together, showing how CRDTs actually communicate in a distributed system.

The fourth part addresses production concerns that most CRDT literature glosses over. Metadata growth, tombstone accumulation, garbage collection, compaction strategies, security, performance optimization, observability, and schema evolution. These are the problems you will face when moving from prototype to production.

The final part places CRDTs in context. We compare them rigorously with operational transformation, consensus protocols, centralized synchronization, event sourcing, and database replication. Not with hand-wavy claims but with concrete trade-offs, complexity analysis, and decision frameworks. We also survey real-world CRDT systems and draw lessons from production deployments.

The Codebase

A book about building systems should contain code that actually works. Throughout the book, we develop a substantial codebase in TypeScript that you can run locally. It is not a series of disconnected snippets. Instead, it grows progressively into a complete multi-replica distributed collaboration system with:

- Core CRDT implementations for all major data types
- A synchronization protocol supporting full sync, delta sync, and causal delivery
- A distributed key-value store built on CRDTs
- A collaborative text editor with real-time sync and offline capability
- Persistence, recovery, garbage collection, and configuration
- End-to-end tests you can run against multiple replicas

You can use this code as a learning tool, a reference implementation, or a starting point for your own projects. The code is designed to be production-quality in structure even if it is simplified for exposition.

What You Will Be Able To Do

After reading this book and working through the examples, you should be able to:

- Explain the distributed systems problem that CRDTs solve and when they are appropriate versus unnecessary or inappropriate.
- Implement CRDTs for any standard data type from scratch, with correct convergence guarantees.
- Design and implement a synchronization protocol for CRDT-based replication.
- Build a distributed collaborative application with offline support.
- Analyze and mitigate metadata growth and tombstone accumulation.
- Secure a CRDT system against unauthorized access and malicious replicas.
- Choose between CRDTs, OT, consensus, and other approaches based on concrete trade-offs.

If you have ever been frustrated by merge conflicts in distributed systems, lost data due to poor replication strategies, or the complexity of building collaboration features, CRDTs offer an elegant, principled way forward. Let us get started.

Chapter 1: The Collaboration Problem

The Broken Promise of Centralized Collaboration

Every time you use a collaborative document, a shared todo list, or a multiplayer game with persistent state, you are depending on a system to keep data consistent across multiple users and multiple devices. This seems like it should be straightforward: put the data in a database, send updates through a server, everyone sees the same thing.

The problem is that the straightforward approach breaks under real-world conditions.

Consider the architecture of a typical centralized collaborative application. There is a single authoritative server holding the source of truth. Clients connect over websockets or HTTP and send their operations to the server. The server validates operations, applies them to its state, and broadcasts the results to other clients. When two users edit concurrently, the server serializes their operations and resolves any conflicts.

This works beautifully when all clients are continuously connected and the network is fast and reliable. It fails catastrophically when they are not.

If a client loses connectivity, it cannot send its updates. If it tries to continue editing offline and then reconnects, its accumulated changes must be merged with whatever happened on the server while it was away. That merge is hard. If the server and client both modified the same part of a document, who wins? If the client added items to a list and the server deleted that same list, what happens?

These are not theoretical questions. They are the daily engineering problems behind every collaborative product you use.

Real-World Failure Scenarios

Let us look at specific ways centralized collaboration fails in practice.

Scenario 1: The Offline Mobile User

Alice is using a mobile note-taking app on her train commute. She writes several notes offline because the train has no cellular coverage. When she arrives at the office and regains connectivity, her phone syncs those notes to the server.

Meanwhile, Bob has been using the same app on his laptop in the office. He edited some of the same notes Alice was editing on her phone. When Alice's phone syncs, the server must merge her offline changes with Bob's concurrent changes.

In a naive implementation, the server might use last-write-wins: whoever sent their update last overwrites everything. This is simple to implement but terrible for users. If Bob spent twenty minutes writing a detailed response and Alice's sync overwrites it with her stale version, Bob's work is gone.

Scenario 2: Multi-Region Replication

A global SaaS application replicates data across three regions: US, EU, and Asia. Users can read and write from their nearest region for low latency. When a user in the US writes data, that write must eventually propagate to the EU and Asia replicas.

Now imagine a network partition between the US and EU regions. Users in both regions continue to write data. When the partition heals, the two regions have diverging state that must be reconciled.

With strong consistency, the system must stop accepting writes during the partition until it can reach consensus on what the correct state is. This defeats the purpose of multi-region replication, which is to keep serving users even when some regions are unreachable.

With eventual consistency and manual conflict resolution, engineers must write custom merge logic for every data type. This logic is notoriously hard to get right. The Dynamo paper from Amazon famously described this as "the nightmare of eventually consistent systems."

Scenario 3: Real-Time Collaborative Editing

Consider a code editor where two developers are editing the same file simultaneously. Developer A inserts text at line 10. Developer B deletes line 10 and

inserts different text at the same position. Both operations arrive at the server milliseconds apart.

The server must decide what the correct state is. If it applies A's operation first, B's delete targets the wrong line. If it applies B's operation first, A's insert goes into the wrong place. The server needs to understand not just the operations but their intent and transform them accordingly.

This is exactly the problem that Operational Transformation (OT) solves, and it is what Google Docs and most collaborative editors use today. But OT requires a central server that serializes all operations and transforms them against each other. It does not work offline, it does not scale to peer-to-peer topologies, and the transformation logic is notoriously error-prone.

Why Locking Fails at Scale

One obvious solution to all these problems is distributed locking. If only one user can modify a piece of data at a time, there are no conflicts to resolve. Just acquire a lock before writing, release it after the write completes.

Locking works for single-server systems. It fails in distributed systems for several reasons.

First, locks require coordination, which requires network round trips. Every write becomes at least two network calls: acquire lock, then write. In a system with hundreds of concurrent writers, contention for locks creates bottlenecks that kill throughput.

Second, locks are fragile under failure. If a process holding a lock crashes before releasing it, the lock is held forever unless you have a lease or timeout mechanism. Leases add complexity and can still cause false-positive lock releases if a process is merely slow rather than crashed.

Third, locks do not help with offline operation. If a client is offline, it cannot acquire locks, so it cannot write. This eliminates one of the primary benefits of replication.

Fourth, distributed locking is hard to implement correctly. The paper "Distributed Locking: The Good, the Bad, and the Ugly" by Michael Sperber et al. demonstrates how subtle bugs in lock implementations can lead to data corruption, deadlocks, and livelocks.

For systems that need to scale horizontally and handle concurrent writers, locking is a solution that solves the wrong problem. It prevents conflicts by preventing concurrency, which is the opposite of what we want.

The Offline-First Imperative

The rise of mobile computing, single-page applications, and edge computing has fundamentally changed how users expect applications to behave. The offline-first paradigm states that an application should be fully functional when offline and only use the network for synchronization when available.

This is not a nice-to-have feature. It is a requirement for modern user experience. Users expect their apps to respond instantly, to work on slow connections, to survive airplane rides, and to handle brief network interruptions transparently.

Offline-first applications must maintain a local copy of data that is always available. When the app is online, that local copy synchronizes with remote copies on servers and other devices. When the app is offline, users continue to modify their local copy. On reconnection, all modifications must merge correctly.

The offline-first requirement makes the limitations of centralized architectures obvious. A server-authoritative model requires every write to reach the server before the user sees confirmation. This adds latency and breaks offline operation. Multi-master replication without CRDTs leads to conflicts that are hard to resolve automatically.

Offline-first is also a requirement for privacy and data ownership. When your application keeps the authoritative copy of your data on your device rather than in someone else's data center, you have more control over that data. You decide when and where it synchronizes. CRDTs are natural allies of this philosophy because they make peer-to-peer synchronization practical and reliable.

A Preview of What Is Possible

Before we dive into the theory and implementation, let us see what CRDTs make possible in practice.

The note-taking app from Scenario 1 above can simply store notes as CRDTs locally on each device. When Alice edits offline, her changes are immediate and certain. When connectivity returns, her device syncs with the server and Bob's device, and all three replicas converge automatically. No merge conflicts. No lost changes. No user intervention required.

The global SaaS application from Scenario 2 can use CRDT-based data types for user-visible state. Each region accepts writes independently. When partitions heal, regions synchronize their CRDTs and converge to a consistent state without coordination. Users in all regions continue to have full write access even during partitions.

The collaborative editor from Scenario 3 can use a CRDT-based text data structure. Each client maintains its own copy of the document and applies edits locally. Edits are broadcast to other clients asynchronously. When concurrent edits arrive, the CRDT resolves them deterministically based on the document's internal structure. No server coordination is required. Offline editing works perfectly.

This is not theoretical. These capabilities are deployed in production today. The Riak distributed database uses CRDTs for counters, sets, and maps across data centers. Riak's chat system at Riot Games handles 7.5 million users and 11,000 messages per second using CRDT-based replication. Apple Notes, Figma, and dozens of other collaborative applications rely on CRDTs or CRDT-inspired architectures.

But the beauty of CRDTs is not just that they exist in production. It is that they are accessible. You do not need a PhD in distributed systems to use them. You do not need a massive engineering team. With a solid understanding of the concepts we will develop in this book, you can implement CRDTs yourself and integrate them into your own applications.

The rest of this book is the path from understanding to implementation. We will build everything from first principles so that you know not only what CRDTs do but why they work and how to adapt them to your own problems.

The collaboration problem is hard. CRDTs do not make it easy. But they make it tractable in a way that no other approach can match. Let us begin with the foundations.

Chapter 2: Foundations of Distributed Replication

Single Source of Truth and Its Limits

The simplest way to share data is to put it in one place. A single database. A single server. A single machine that everyone agrees is the authority. When you want to read data, you read from the server. When you want to write, you write to the server. Done.

This works beautifully until you scale.

The first problem is availability. If the single server goes down, your entire application is down. Users cannot read or write. For applications where downtime is unacceptable, this is not acceptable.

The second problem is latency. If your users are distributed globally, they cannot all be close to your single server. A user in Tokyo reading from a server in Virginia will experience hundreds of milliseconds of latency. For real-time collaboration, this lag is perceptible and frustrating.

The third problem is throughput. A single server can only process so many requests per second. When you exceed that capacity, requests queue up and response times degrade. You can throw more CPU and memory at the problem, but eventually you hit physical limits.

The answer to all three problems is replication: keep multiple copies of the same data on different machines. Different machines can serve different users, handle failures, and process more requests in parallel.

But replication immediately introduces a new and difficult problem: keeping copies consistent.

Replication Architectures

There are several architectural patterns for replicating data in distributed systems, each with different trade-offs between consistency, availability, and

complexity.

Primary-Backup (Leader-Follower)

The primary-backup architecture designates one replica as the primary (leader) and all others as backups (followers). All writes go to the primary, which then replicates changes to the backups. Reads can go to the primary or to the backups.

This is the architecture used by most relational databases with replication (MySQL, PostgreSQL, MongoDB). It is simple to reason about: there is one source of truth, and backups are just catching up.

The primary-backup model provides strong consistency for reads from the primary. It is also simple to implement because the primary serializes all writes into a single order, and backups just replay that order.

The problems are straightforward. First, the primary is a single point of failure. If it crashes, the system must elect a new primary before writes can resume. This election takes time, during which the system is unavailable for writes. Second, the primary is a bottleneck. All writes must pass through it, limiting write throughput. Third, backups are stale. They have not yet received all the primary's updates, so reads from backups can return outdated data.

For many applications, primary-backup is a good balance of simplicity and reliability. But it does not scale to multi-master writes and does not handle network partitions gracefully.

Multi-Master (Active-Active)

In multi-master replication, any replica can accept writes. There is no designated primary. When replica A receives a write, it applies the write locally and then propagates it to replicas B, C, and so on.

Multi-master replication solves several problems from primary-backup. There is no single point of failure for writes. Users can write to the nearest replica for low latency. Write throughput scales with the number of replicas.

But it introduces a much harder problem: conflicting writes. If replica A and replica B both accept writes to the same data item concurrently, the two replicas now have different values for that item. When they eventually communicate, they must resolve the conflict.

This is exactly where CRDTs come in. Multi-master replication without a conflict resolution strategy is a nightmare of edge cases and manual merge logic. CRDTs provide a principled, automated way to handle concurrent writes across replicas without coordination.

Peer-to-Peer (Gossip)

Peer-to-peer replication treats all replicas as equal peers with no hierarchy. Each replica communicates with a subset of other replicas, exchanging state updates via gossip protocols. There is no primary, no leader election, no designated coordinator.

Gossip replication is extremely robust to failure. Individual nodes can come and go without affecting the system. The network self-heals as gossip spreads information.

The downside is that gossip is eventually consistent by nature. It takes time for information to propagate, and there is no guarantee about when any particular replica will see any particular update. For applications that require immediate consistency, gossip is not appropriate.

CRDTs are natural allies of gossip replication because CRDTs are designed for eventually consistent environments. You can disseminate CRDT states via gossip and the replicas will converge automatically.

Client-Centric Replication

Client-centric replication stores the authoritative copy of data on the client device rather than on a central server. Multiple devices each have their own copy, and they synchronize with each other when possible. The server is a sync hub, not an authority.

This is the model used by Apple Notes, Dropbox, and many offline-first applications. It provides the best offline experience because your local copy is always available. It also supports privacy because your data lives on your devices.

Client-centric replication is the hardest to implement correctly. Devices join and leave the network frequently, have intermittent connectivity, and operate for long periods offline. The sync protocol must handle all of this gracefully.

CRDTs are the foundation of client-centric replication. They are the mechanism that makes automatic conflict resolution possible when devices reconnect with diverging state.

Consistency Models From Strong to Eventual

Consistency models define the guarantees that a replicated system makes to its users about what values they will see when they read data. Understanding these models is essential for understanding where CRDTs fit.

Strong Consistency

Strong consistency guarantees that every read returns the value of the most recent write to that item, as if all operations on all replicas were executed in a single global order.

Strong consistency is what you get from a single non-replicated database. It is intuitive and easy to reason about. If you write value X and then read, you will see X.

In a replicated system, strong consistency requires coordination. Before a write is visible, it must be propagated to enough replicas and acknowledged. This coordination adds latency and reduces availability.

Linearizability is the strongest consistency model for replicated data. Introduced by Herlihy and Wing in 1990, linearizability requires that every operation appears to take effect atomically at some point between its invocation and its completion. Operations that do not overlap in time must appear in their real-time order. Operations that do overlap can appear in any order, but only one appears at any given time.

Linearizability is powerful because it is composable: if you have two linearizable objects, you can combine them and the combined system is still linearizable. This makes it easy to reason about complex systems built from linearizable components.

Linearizability is also expensive. It requires real-time coordination between replicas. During network partitions, a linearizable system cannot guarantee both consistency and availability. This is the CAP theorem in action.

Sequential Consistency

Sequential consistency, introduced by Lamport in 1979, is weaker than linearizability but still strong. It requires that all operations appear to execute in some single global order that respects the program order of each individual process.

Unlike linearizability, sequential consistency does not respect real-time ordering. If operation A completes before operation B begins in real time, sequential consistency does not require A to appear before B in the global order. This relaxation makes sequential consistency cheaper to implement than linearizability.

However, sequential consistency is not composable. Two sequentially consistent objects combined can produce behavior that is not sequentially consistent. This limitation has restricted its adoption in practice.

Causal Consistency

Causal consistency is weaker still. It requires that causally related operations be seen in the same order by all processes, but allows unrelated (concurrent) operations to be seen in any order.

If process A writes X = 1 and then process B reads X and writes Y = X + 1, then all processes must see X = 1 before seeing Y = 2. This preserves the causal relationship between the two writes.

But if process A writes X = 1 and process C writes Y = 2 independently (no causal relationship), different processes might see X = 1 before Y = 2 or vice versa.

Causal consistency is a natural model for many distributed applications, including messaging systems and collaborative editing. CRDTs are compatible with causal consistency and can achieve stronger guarantees.

Eventual Consistency

Eventual consistency is the weakest common consistency model. It promises that if no new updates are made to an item, eventually all replicas will converge to the same value.

"Eventually" is the key word. Eventual consistency makes no promises about how long convergence takes, what value you will see in the meantime, or whether all replicas see updates in the same order.

Eventual consistency is what you get when you replicate data without coordination and resolve conflicts lazily. It maximizes availability and performance but can be confusing for users who expect to see their writes immediately.

The Dynamo paper from Amazon famously embraced eventual consistency, describing conflict resolution as "the nightmare of eventually consistent systems" but accepting it as the price for high availability and low latency.

CRDTs provide something stronger than bare eventual consistency: strong eventual consistency. We will define this precisely in Chapter 5. For now, know that CRDTs guarantee not just that replicas will eventually converge, but that they will converge to the same state regardless of the order in which updates are delivered, provided they all see the same set of updates.

The CAP Theorem and Its Implications

The CAP theorem, proposed by Eric Brewer in 2000 and proved by Gilbert and Lynch in 2002, states that a distributed system can provide at most two of the following three guarantees:

- Consistency (C): Every read receives the most recent write or an error.
- Availability (A): Every request receives a non-error response, regardless of whether any replicas are down.
- Partition Tolerance (P): The system continues to operate despite arbitrary message loss or delay between nodes (network partitions).

The CAP theorem is often misunderstood. It is not saying that you can pick any two properties and ignore the third. It is saying something more specific: in the presence of a network partition, you must choose between consistency and availability.

During a partition, some nodes cannot communicate with others. If you want consistency, you must refuse to serve reads or writes from nodes that cannot confirm their data is current. This sacrifices availability. If you want availability, you must serve reads and writes from all nodes even when they cannot coordinate. This sacrifices consistency.

If there is no partition, you can have both consistency and availability (with enough engineering effort). But partitions are inevitable in real networks, so you must design for them.

CRDTs sit on the AP side of the CAP trade-off. They prioritize availability and partition tolerance over strong consistency. But they do so in a principled way: rather than just accepting inconsistency, they guarantee that inconsistency will resolve automatically when the partition heals.

The CAP theorem is sometimes criticized for being too coarse. The PACELC theorem extends it by noting that even in the absence of partitions, systems face trade-offs between latency and consistency. But CAP remains a useful mental model for thinking about replication.

Linearizability Versus Sequential Consistency

The distinction between linearizability and sequential consistency is subtle but important for understanding what CRDTs provide and what they do not.

Linearizability imposes real-time constraints. If operation A completes before operation B begins, then A must appear before B in the linearization. This means that a linearizable system can detect and reject "impossible" reads: if you write X = 1 and then immediately read X and see 0, the system is not linearizable.

Sequential consistency does not impose real-time constraints. It only requires that each process's operations appear in program order and that there exists a single global order respecting all program orders. Under sequential consistency, it is possible (though undesirable) for a process to write X = 1 and then immediately read X = 0, as long as some global order exists where the read appears before the write.

For most practical purposes, linearizability is what users expect. When you submit a payment, you expect to see it reflected immediately. Sequential consistency is too weak for financial applications.

CRDTs do not provide linearizability. They provide strong eventual consistency, which is weaker. But they do so by design: the goal is availability and partition tolerance, not real-time consistency. For applications where eventual agreement is acceptable (collaborative editing, counters, shared lists), this trade-off is worthwhile.

Partition Tolerance in Real Networks

Network partitions are not rare edge cases. They happen regularly in real distributed systems.

In data centers, partitions occur when switches fail, cables are unplugged, or network misconfigurations isolate subsets of machines. Between data centers, partitions are even more common due to long distances, routing issues, and regional outages.

On the internet, partitions are the norm rather than the exception. Mobile devices connect and disconnect frequently. Users move between Wi-Fi networks and cellular coverage. Cloudflare's global edge network routinely handles regions becoming temporarily unreachable.

A system that requires all nodes to agree before accepting writes will stall during any partition. This is unacceptable for applications that need to remain available. A system that accepts writes on any node must handle the resulting conflicts when the partition heals.

CRDTs are designed for the partition-prone reality of real networks. They assume that partitions will happen, that messages will be delayed and reordered, and that replicas will diverge. They then guarantee that, despite all of this, the system will converge to a consistent state without external intervention.

This is not optimism. It is engineering for failure.

The Trade-Off Spectrum

Consistency, availability, and performance form a spectrum, not a binary choice. Different parts of the same system can sit at different points on the spectrum.

For example, a banking application might use linearizable consistency for account balances (strong consistency is non-negotiable) but eventually consistent replication for user profile information (eventual consistency is fine). A collaborative editor might use strong eventual consistency for the document content (users expect their edits to appear, but not necessarily instantly on all devices) and a central server for user authentication (strong consistency is needed).

CRDTs are one tool in the consistency toolbox. They are not a universal solution. They excel at specific problems: optimistic replication, offline-first applications, multi-master writes, peer-to-peer synchronization. For problems requiring strong consistency (financial transactions, inventory management, leader election), CRDTs are the wrong tool.

Understanding where CRDTs fit requires understanding the broader landscape of distributed system consistency. The next chapters build that understanding further by introducing the concepts of causality and logical time that underpin CRDT design.

Chapter 3: Causality and Logical Time

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

The Nature of Causality in Distributed Systems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Happens-Before and Partial Ordering

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Lamport Timestamps

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Vector Clocks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Dot Clocks and Dot Updates

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Detecting Causality Violations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Chapter 4: Eventual Consistency and Convergence

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

What Eventual Consistency Actually Means

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Convergence Conditions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Monotonicity and the Path to Agreement

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

When Eventual Consistency Is Not Enough

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Measuring and Testing Convergence

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

The Split-Brain Problem

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Causal Delivery and Reliable Broadcast

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Summary of Convergence

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Chapter 5: Introducing CRDTs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

The CRDT Promise

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Formal Definition and Required Properties

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

State-Based CRDTs (CvRDTs)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Operation-Based CRDTs (CmRDTs)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Delta-State CRDTs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Join Semilattices and Monotonicity

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Why CRDTs Always Converge

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Strong Eventual Consistency

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

The Trade-Offs of Unconditional Convergence

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

CRDTs and the CAP Theorem

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Chapter 6: Building Your CRDT Toolkit

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Project Setup and Dependencies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Core Types and Interfaces

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Replica Identity

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Dot Clocks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Serialization and Canonical Encoding

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

The Replica Model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

A Minimal Test Harness

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Chapter 7: Registers and Flags

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

The Write-Wins Register

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

The Last-Writer-Wins Register

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

The Write-Wins Register with Timestamps

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Multi-Value Registers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Dot Clocks for Register Synchronization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Flags and Presence Bits

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Complete Example: Running Multiple Replicas

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Chapter 8: Counters and Numeric CRDTs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

The Grow-Only Counter

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Positive-Negative Counters

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

The Replica-Numbered Counter

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Bounded and Rolling Counters

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Floating-Point and Monetary Counters

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Complete Example

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Chapter 9: Sets

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

The Concurrency Problem for Sets

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

The Grow-Only Set

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

The Two-Phase Set

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Observed-Remove Sets

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Last-Writer-Wins Sets

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Union-Only and Intersection-Only Sets

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Implementation and Replication

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Chapter 10: Maps and Nested Structures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Map-Specific Concurrency Issues

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

The LWW-Register Map

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

The MMRG Map CRDT

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Nesting CRDTs Within Maps

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Deep Convergence Guarantees

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Memory and Tombstone Growth in Maps

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Chapter 11: The Synchronization Protocol

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Synchronization Requirements

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Full State Transfer

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Delta-State Replication

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Delta Sync with Dot Clocks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Causal Delivery and Message Ordering

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Anti-Entropy and Peer Discovery

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Implementation of the Sync Protocol

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Chapter 12: Sequences and Ordered Collections

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Why Sequences Are Hard

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Positioning in Concurrent Documents

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

RGA Sequences

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Logoot Sequences

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Positioning Without Gaps

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Implementation and Replication

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Chapter 13: Text Editing CRDTs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

The Collaborative Editing Problem

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Document State Representation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Concurrent Insertion Resolution

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Concurrent Deletion Resolution

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Cursor Positions and Selection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Real-Time and Offline Editing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Chapter 14: Building a Distributed Key-Value Store

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Architecture and Design

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

The Replica Process

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Persistence and Recovery

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Peer Communication

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Testing with Multiple Replicas

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Running the System

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Chapter 15: Building a Collaborative Text Editor

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Editor Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Client-Side CRDT Integration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

WebSocket Synchronization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Offline Editing and Replay

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Concurrent Editing Tests

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Complete Working Example

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Chapter 16: Metadata Growth, Tombstones, and Garbage Collection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

The Tombstone Problem

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Why CRDTs Leak Memory

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Garbage Collection in CRDTs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Compaction Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Bounded Metadata Designs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Chapter 17: Production Concerns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Security and Authorization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Malicious and Invalid Operations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Performance Optimization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Observability and Debugging

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Schema Evolution and Versioning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Scaling to Many Replicas

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Chapter 18: CRDTs Versus Alternatives

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Operational Transformation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Centralized Synchronization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Distributed Locking

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Consensus Protocols

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Event Sourcing and CQRS

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Database Replication Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Decision Framework

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Chapter 19: Real-World CRDT Systems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Riak Distributed Database

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Yjs Collaborative Framework

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Automerge Offline-First Library

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Figma and Real-Time Design Tools

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Emerging CRDT Databases

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Lessons from Production

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Chapter 20: Conclusion and Where to Go

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

The CRDT Design Philosophy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Where CRDTs Are Going

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Open Problems and Research Frontiers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

A Final Architecture Checklist

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Final Thoughts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Bibliography and Further Reading

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Foundational CRDT Papers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Distributed Systems Foundations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Sequence CRDTs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Delta-State and Synchronization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Security and Verification

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

OT Versus CRDT

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Libraries and Production Systems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.

Tutorials and Supplementary Material

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/conflict-freereplicateddatatypes.