Conflict-free Replicated Data Type (CRDT)

What is a Lock-Free Data Structure?

A lock-free data structure enables concurrent, non-blocking access to shared data without traditional locks, boosting performance in multi-threaded applications.

Key Concepts

- **No Waiting:** Avoids lock contention to enhance performance.
- **Atomic Operations:** Uses low-level atomic operations (e.g., Compare-And-Swap) for safe, consistent data updates.
- **High Concurrency:** Improves performance by allowing multiple threads to interact without blocking.

Analogy

Like updating a shared Google Doc — everyone edits freely, last valid edit wins.

LOCK FREE DATA STRUCTURES

PROS

Enhanced Performance: Threads operate concurrently without locks, boosting throughput and performance in contested scenarios.

No Deadlocks: Lock absence eliminates deadlocks, increasing system robustness and reliability under heavy loads.

CONS

Implementation Complexity: Designing lock-free algorithms is exceptionally difficult, requiring expert knowledge to manage thread synchronization and prevent race conditions.

Hardware Dependency: Lock-free algorithms lack universal portability, relying on CPU-specific atomic instructions (e.g., CAS) that necessitate code adaptation for different hardware architectures.

Message Passing Model of Process Communication

This model allows processes to communicate without sharing the same address space. They exchange information by sending messages to and receiving messages from a common **message queue**.

- Each thread (actor) has a **mailbox**
- Avoids shared-state issues

Why Message Passing Helps?

- No race conditions on shared memory
- Easier reasoning about state
- Fits distributed systems naturally

Broadcasting Local Updates via Message Passing

- **User Makes Local Edit** → Change detected
- **System Creates Update Object** → captures *what changed, where, when, by whom*
- **Query Shared Registry** → find all active users & their message queues

Conflict-Free Replicated Data Type (CRDT)

A CRDT is a special type of data structure designed to handle concurrent updates across multiple devices or servers (**replicas**) without the need for a central coordinator.

Use Cases:

- **Mobile Apps:** Syncing data (notes, calendars) across a user's devices.
- **Distributed Databases:** Ensuring data consistency across multiple database nodes.
- **Collaboration Software:** Allowing multiple users to edit the same document simultaneously.

Types of CRDTs:

There are two main types of CRDTs:

- **State-based CRDTs:** These replicate the entire state of the data structure. Merging is done by comparing and reconciling the states of different replicas.
- **Operation-based CRDTs:** These replicate the operations performed on the data structure. Merging is done by applying operations from different replicas in a commutative manner.

HOW CRDTs work?

All operations follow these simple rules:

- **Commutativity:** Order doesn't matter.

  ```
  (Update A + Update B) = (Update B + Update A)
  ```

- **Associativity:** Grouping doesn't matter.

  ```
  ((A+B) + C) = (A + (B+C))
  ```

- **Idempotence:** Duplicates have no extra effect.

  ```
  (State + Update A + Update A) = (State + Update A)
  ```

Questions?