Using CRDTs beyond Text Editors

(Conflict-free Replicated Data Type s)

(Commutative

(Convergent

Oliver Wangler

2025-03-26

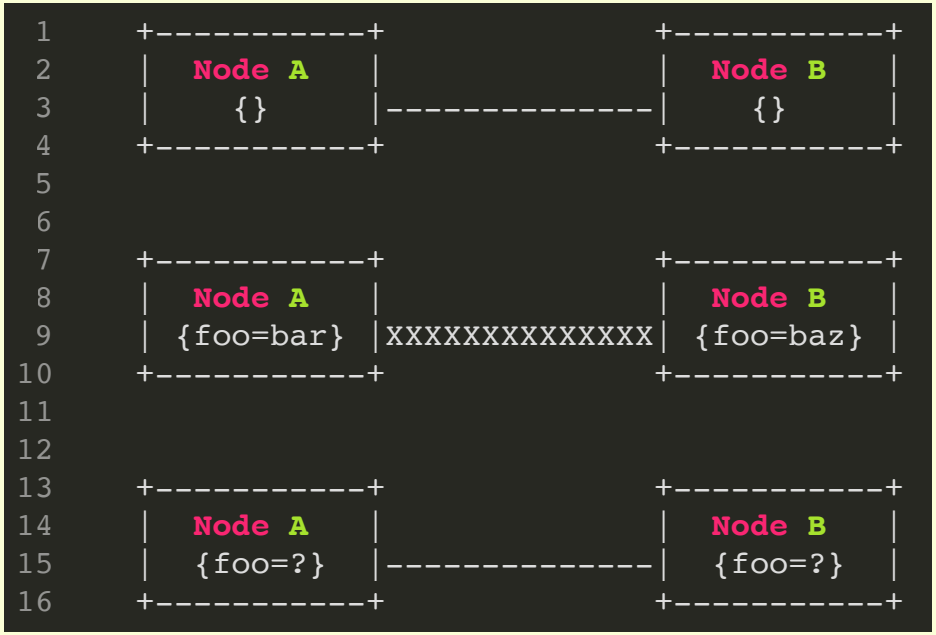
Index

Op-based CRDTs and Rust ecosystem →

δ-CRDTs →

DSON — Distributed JSON →

Mental Model



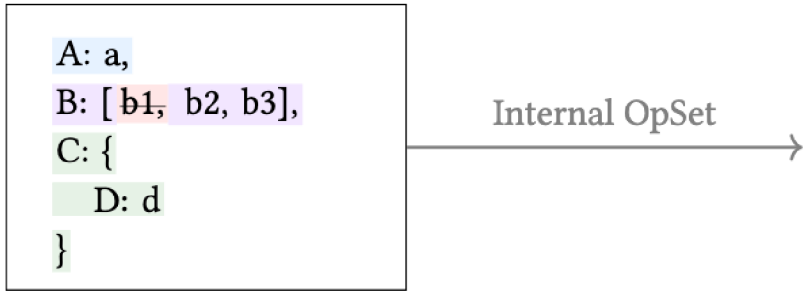
CRDTs let you expose concurrency

Conflict-free doesn't mean free of conflicts

Convergence in a Nutshell

CRDTs work by exchanging state updates that can be applied in any order, aggregated, and multiple times. This allows for state convergence in a distributed system—Strong Eventual Consistency. It's the guarantee that when all nodes have received the same set of updates, they'll be in the same state.

Operation-based CRDTs



ID	Type	Object ID	Key	Value	Predecessors
⟨1, 0⟩	put	⟨0, 0⟩	A	a	
⟨2, 0⟩	make	⟨0, 0⟩	B	list	
⟨6, 0⟩	make	⟨0, 0⟩	C	map	
⟨7, 0⟩	put	⟨6, 0⟩	D	d	
⟨3, 0⟩	put	⟨2, 0⟩	⟨0, 0⟩	b1	
⟨4, 0⟩	put	⟨2, 0⟩	⟨3, 0⟩	b2	
⟨5, 0⟩	put	⟨2, 0⟩	⟨4, 0⟩	b3	
⟨8, 0⟩	delete				⟨3, 0⟩

Figure 2. An example JSON document with its internal OpSet

require **full** log replication.

From *"Liangrun Da and Martin Kleppmann. Extending JSON CRDTs with Move Operations. The 11th Workshop on Principles and Practice of Consistency for Distributed Data (PaPoC '24), April 2024."*

Rust CRDT libraries


```rust
use autosurgeon::{Hydrate, Reconcile, hydrate, reconcile};

#[derive(Debug, Clone, Reconcile, Hydrate, PartialEq)]
struct Contact {
    name: String,
    email: String,
}

#[derive(Debug, Clone, Reconcile, Hydrate, PartialEq)]
struct Document {
    contacts: HashMap<String, Contact>
}

let alice = Contact {
    name: "Alice".to_owned(),
    email: "alice@example.com".to_owned(),
};

let doc_wrapper = Document {
    contacts: [("alice", alice.clone())].into(),
};


let mut peer1 = automerge::AutoCommit::new();
reconcile(&mut peer1, &doc_wrapper).unwrap();
```

Adapted example from https://docs.rs/autosurgeon/0.8.7/autosurgeon/index.html

Rust CRDT libraries


```rust
use yrs::{Doc, GetString, ReadTxn, StateVector, Text, Transact, Update};
use yrs::updates::{decoder::Decode, encoder::Encode};

let doc = Doc::new();
let text = doc.get_or_insert_text("article");
{
  let mut txn = doc.transact_mut();
  text.insert(&mut txn, 0, "hello");
  text.insert(&mut txn, 5, " world");
  // other rich text operations include formatting or inserting embedded elements
} // transaction is automatically committed when dropped
assert_eq!(text.get_string(&doc.transact()), "hello world".to_owned());

// synchronize state with remote replica
let remote_doc = Doc::new();
let remote_text = remote_doc.get_or_insert_text("article");
let remote_timestamp = remote_doc.transact().state_vector().encode_v1();

// get update with contents not observed by remote_doc
let update = doc.transact().encode_diff_v1(
    &StateVector::decode_v1(&remote_timestamp).unwrap()
);
// apply update on remote doc
remote_doc.transact_mut().apply_update(Update::decode_v1(&update).unwrap());

assert_eq!(
    text.get_string(&doc.transact()),
    remote_text.get_string(&remote_doc.transact())
);
```

Adapted example from https://docs.rs/yrs/0.23.0/yrs/index.html

Application stack

Business logic

Automerge

CRDT lib

Schema evolution 😱

Identity 😱 management

Centralized

Decentralized

iroh

Networking

Storage / backup /archival / static asset distribution

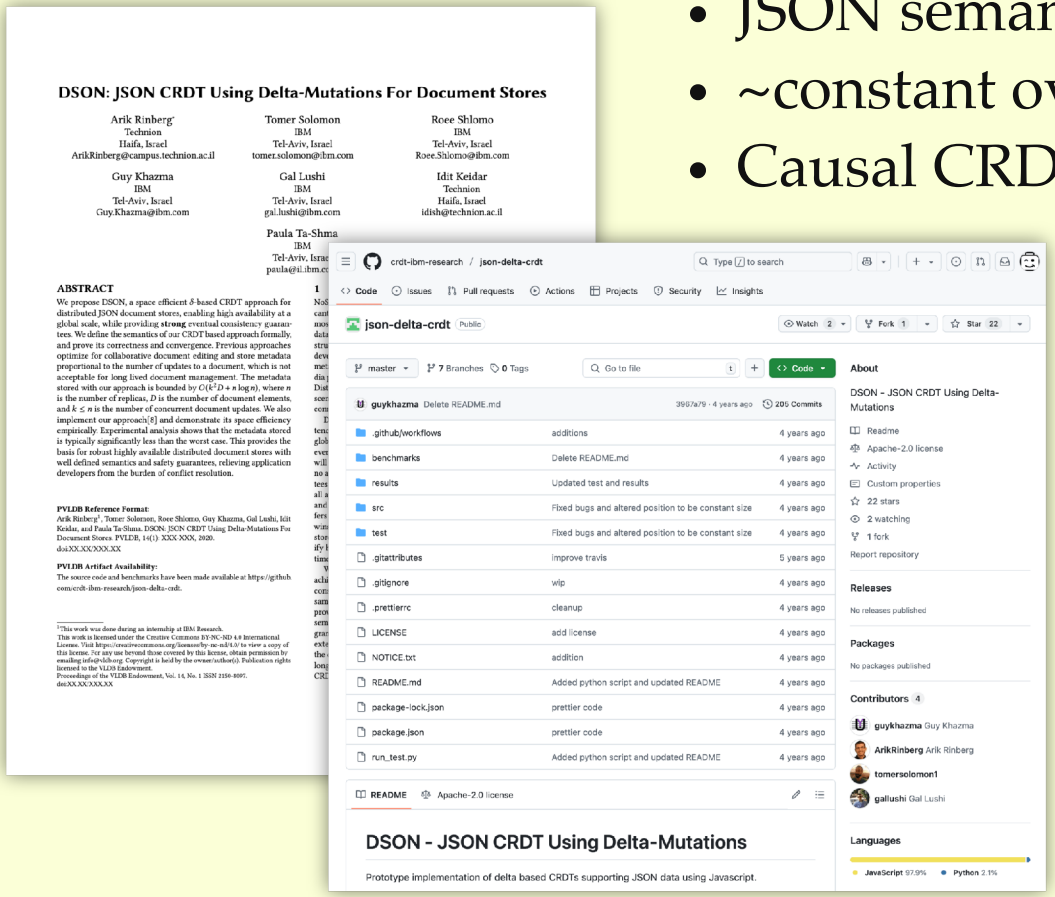
Using CRDTs in constrained environments

- Robust under high latency / low bandwidth
- Open system, thus varying membership
- Network semantics
  - broadcast
  - out-of-order and duplicate delivery
- Partial and incremental sync
- Data relaying and filtering

δ-CRDTs

- State-based CRDTs require full state exchange
- Shapiro et al. "Delta State Replicated Data Types" (2018)
- δ-CRDTs send only minimal changes (deltas) and partial causal metadata. Similar to op-based, but different way to track causality.

δ-CRDTs — DSON



DSON: JSON CRDT Using Delta-Mutations For Document Stores

Arik Rinberg[1] Technion Haifa, Israel ArikRinberg@campus.technion.ac.il
Tomer Solomon IBM Tel-Aviv, Israel tomer.solomon@ibm.com
Roee Shlomo IBM Tel-Aviv, Israel Roee.Shlomo@ibm.com
Guy Khazma IBM Tel-Aviv, Israel Guy.Khazma@ibm.com
Gal Lushi IBM Tel-Aviv, Israel gal.lushi@ibm.com
Idit Keidar Technion Haifa, Israel idish@technion.ac.il
Paula Ta-Shma IBM Tel-Aviv, Israel paula@il.ibm.com

ABSTRACT

We propose DSON, a space efficient δ-based CRDT approach for distributed JSON document stores, enabling high availability at a global scale, while providing **strong** eventual consistency guarantees. We define the semantics of our CRDT based approach formally, and prove its correctness and convergence. Previous approaches optimize for collaborative document editing and store metadata proportional to the number of updates to a document, which is not acceptable for long lived document management. The metadata stored with our approach is bounded by $O(k^2 D + n \log n)$, where n is the number of replicas, D is the number of document elements, and $k \leq n$ is the number of concurrent document updates. We also implement our approach[8] and demonstrate its space efficiency empirically. Experimental analysis shows that the metadata stored is typically significantly less than the worst case. This provides the basis for robust highly available distributed document stores with well defined semantics and safety guarantees, relieving application developers from the burden of conflict resolution.

PVLDB Reference Format:
Arik Rinberg[1], Tomer Solomon, Roee Shlomo, Guy Khazma, Gal Lushi, Idit Keidar, and Paula Ta-Shma. DSON: JSON CRDT Using Delta-Mutations For Document Stores. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code and benchmarks have been made available at https://github.com/crdt-ibm-research/json-delta-crdt.

[1]This work was done during an internship at IBM Research.
This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

crdt-ibm-research / json-delta-crdt

Code Issues Pull requests Actions Projects Security Insights

json-delta-crdt Public

Watch 2 Fork 1 Star 22

master 7 Branches 0 Tags Go to file Code

guykhazma Delete README.md 3987a79 · 4 years ago 205 Commits

.github/workflows	additions	4 years ago
benchmarks	Delete README.md	4 years ago
results	Updated test and results	4 years ago
src	Fixed bugs and altered position to be constant size	4 years ago
test	Fixed bugs and altered position to be constant size	4 years ago
.gitattributes	improve travis	5 years ago
.gitignore	wip	4 years ago
.prettierrc	cleanup	4 years ago
LICENSE	add license	4 years ago
NOTICE.txt	addition	4 years ago
README.md	Added python script and updated README	4 years ago
package-lock.json	prettier code	4 years ago
package.json	prettier code	4 years ago
run_test.py	Added python script and updated README	4 years ago

README Apache-2.0 license

DSON - JSON CRDT Using Delta-Mutations

Prototype implementation of delta based CRDTs supporting JSON data using Javascript.

About
DSON - JSON CRDT Using Delta-Mutations
Readme
Apache-2.0 license
Activity
Custom properties
22 stars
2 watching
1 fork
Report repository

Releases
No releases published

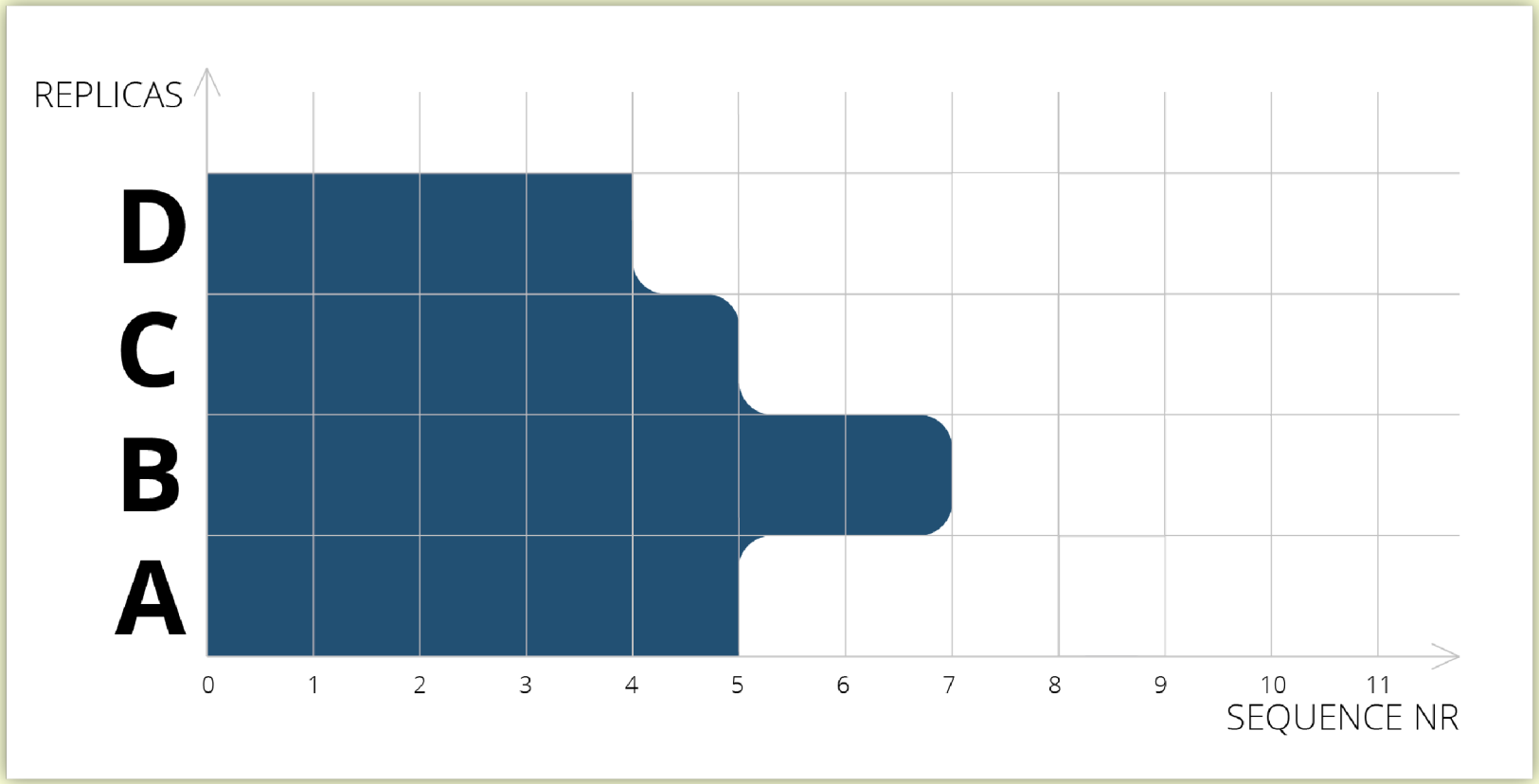
Packages
No packages published

Contributors 4
guykhazma Guy Khazma
ArikRinberg Arik Rinberg
tomersolomon1
gallushi Gal Lushi

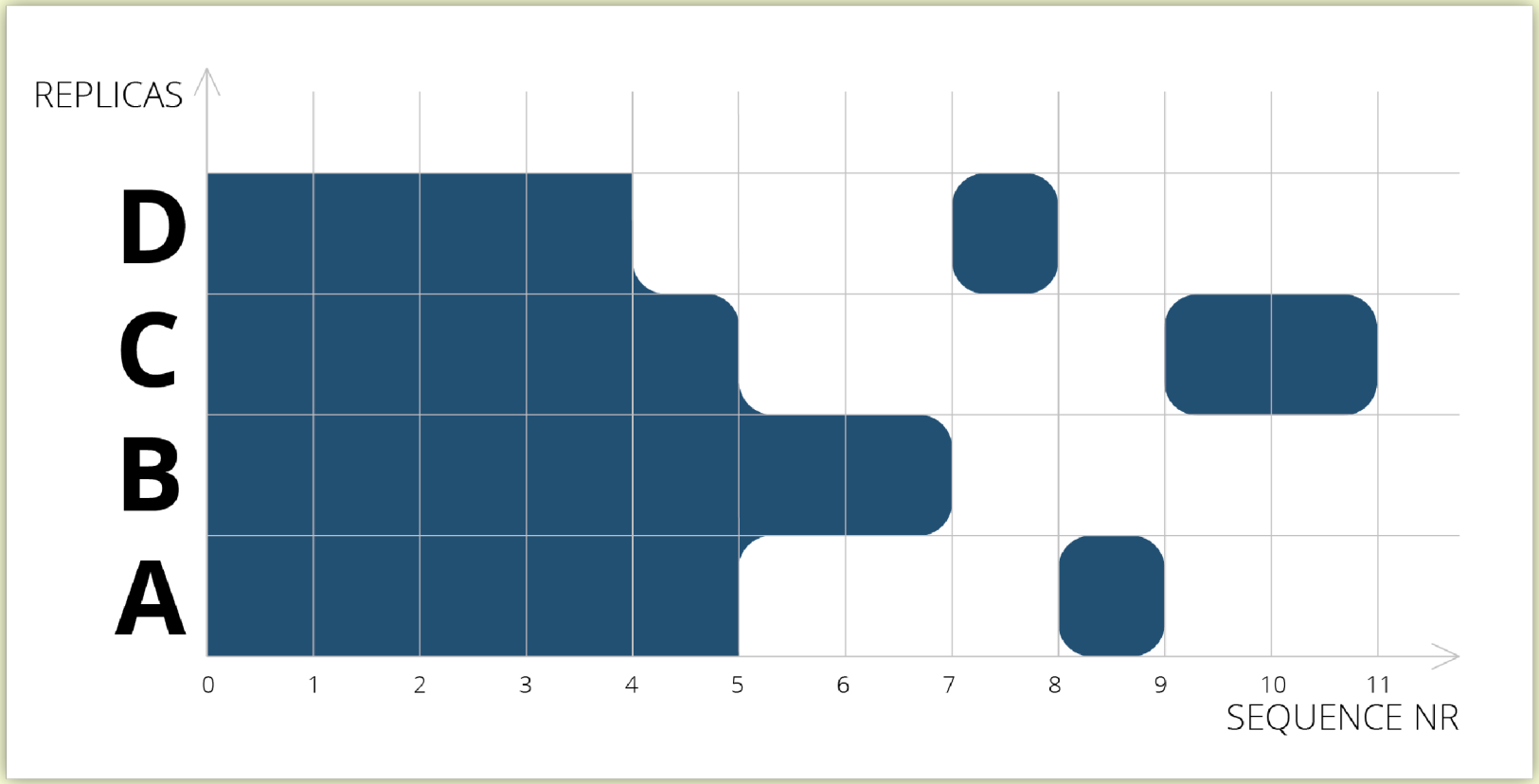
Languages
JavaScript 97.9% Python 2.1%

- JSON semantics (nesting!)
- ~constant overhead
- Causal CRDT

δ-CRDTs — Tracking Causality

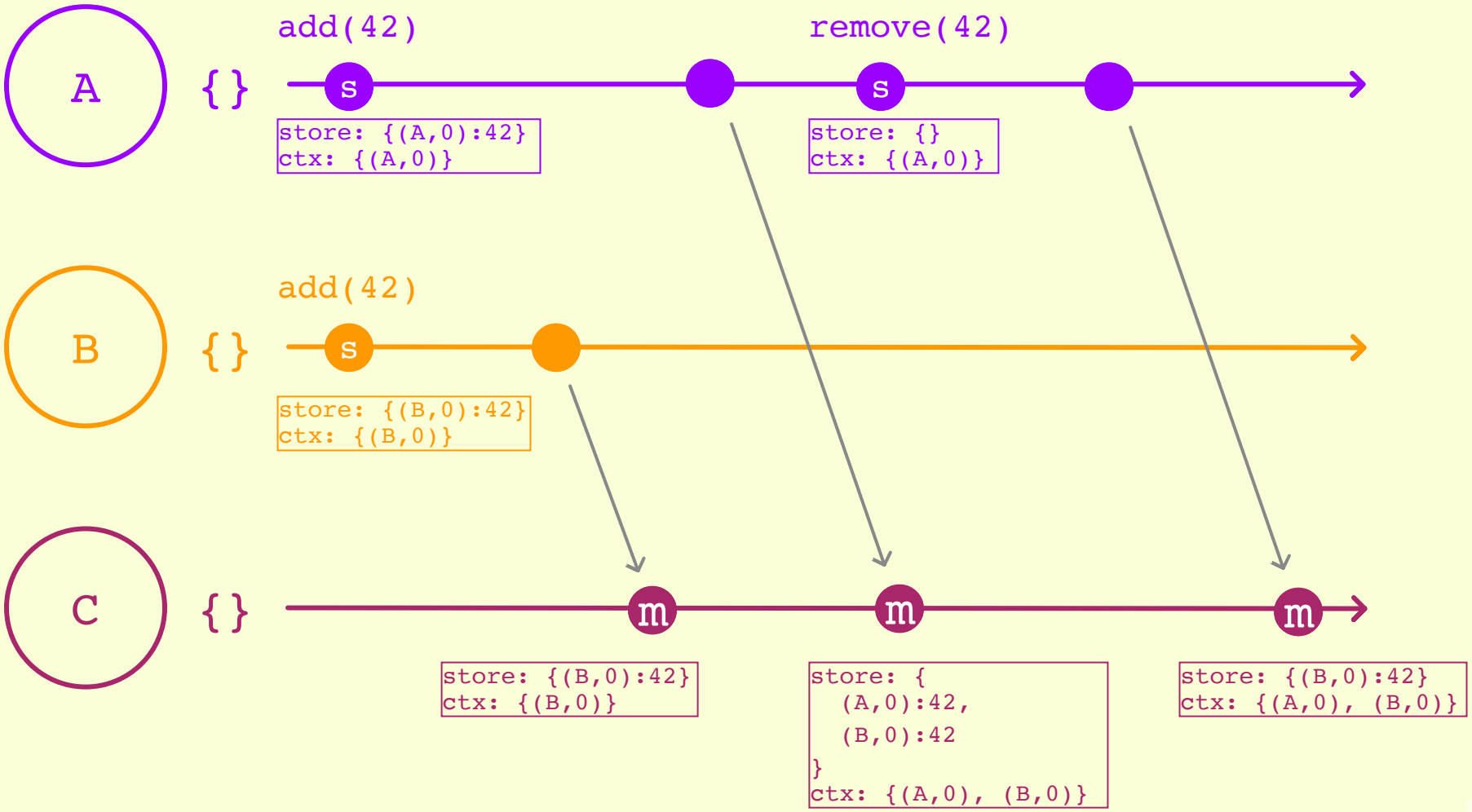


δ-CRDTs — Tracking Causality




```rust
pub struct CausalContext {
    compact_context: HashMap<Identifier, u64>,
    cloud: BTreeSet<(Identifier, u64)>,
}
```

δ-CRDTs — Observed-Remove-Set



δ-CRDTs — There's no free lunch

Code 5 Delta-state-based (Δ-SB) synchronization algorithm

```
01  upon new_neighbor(neigh_id) do
02    if (self.id < neigh_id) then
03      send(neigh_id, self.digest)
04    fi

05  upon receive(neigh_id, digest) do
06    D ↩ get_missing(digest, self.state)
07    if (D ≠ ⊥) then
08      send(neigh_id, D)
09    fi
10    if (digest after self.digest) then
11      send(neigh_id, self.digest)
12    fi

13  upon receive(neigh_id, delta) do
14    Δin ↩ get_missing(self.digest, delta)
15    if (Δin ≠ ⊥) then
16      self.state ↩ merge(self.state, Δin)
17      self.digest ↩ get_digest(self.state)
18    fi

19  upon update(m) do
20    self.digest ↩ get_digest(self.state)

21  function get_missing(DIGEST d, CRDT s): M
22    M ↩ { m ∈ s.state, m.timestamp > d }
```

Frédéric Guidec, Yves Mahéo, Camille Noûs. Supporting conflict-free replicated data types in opportunistic networks. 2022

- Gossip protocol
- At-least-once delivery for each δ
  - Δ-CRDT
    - *"Making δ-CRDTs Delta-Based"*
    - *van der Linde et al. 2016*
  - Delta decomposition?
  - DSON has no tombstones!

`dson-rs`: Schema Support

```
syntax = "proto3";
import "dson.proto";
package rustikon;

message Contact {
  string name = 1;
  string email = 2;
}

message Root {
  option (dson.root) = true;
  map<string, Contact> contacts = 1 [(dson.key_type) = "ulid"];
}
```

dson-rs: APIs

```rust
use dson::{Doc, MaybeConflictRef, MaybeUnknownRef};
use crate::rustikon::{Contact, Root};

let mut doc = Doc::<Root>::new(todo!()).await?;
let alice_uid = ulid::Ulid::new();

{
    let mut lock = doc.lock();

    lock.contacts().insert(
        alice_uid,
        Contact::builder() // infallible builders from `bon` are awesome!
            .name("Alice")
            .email("alice@example.com")
            .build(),
    );
    lock.commit();
}

// Replication in background
// . . .
// . . . .
// . . . . .
// . . . . . .
// . . . . . . .

{
    // Another replica waits for updates
    doc.wait_for_changes().await?;

    let mut lock = doc.lock().await?;
```

Recap

- Op-based CRDTs can be used today
- CRDTs are useful beyond text editors
- `dson-rs` will be open-sourced soon*
- Decentralize all the things!

Thanks!

* Yes, I'm **very** serious.

https://slides.com/wngr/using-crdts-beyond-text-editors

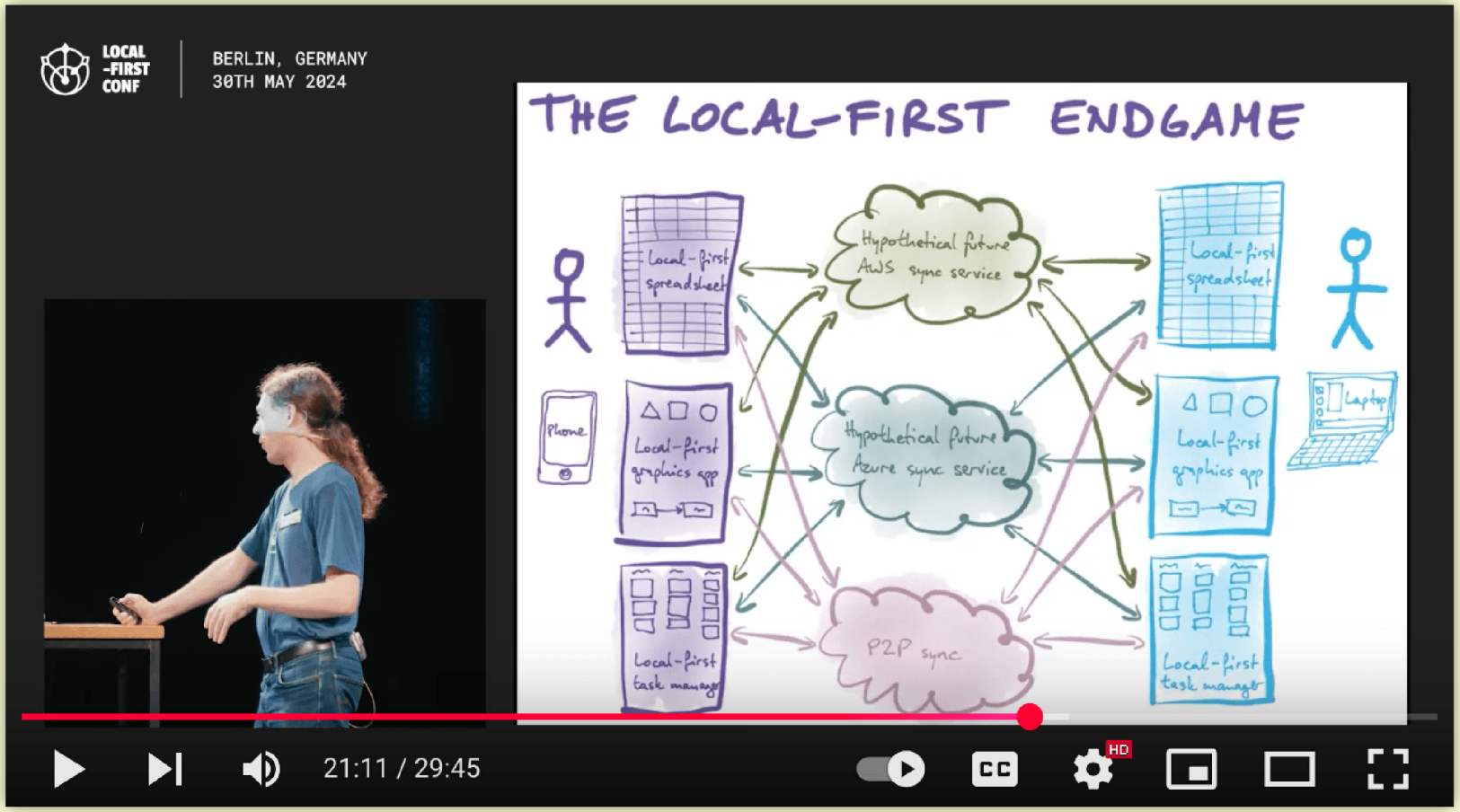
Backup

Rust CRDT libraries


```rust
use automerge::{transaction::Transactable, sync::{self, SyncDoc}, ReadDoc};

// Create a document on peer1
let mut peer1 = automerge::AutoCommit::new();

{
  // `put_object` creates a nested object in the root key/value map and
  // returns the ID of the new object, in this case a map.
  let contacts = peer1.put_object(automerge::ROOT, "contacts", ObjType::Map)?;

  // Now we can insert objects into the list
  let alice = peer1.insert_object(&contacts, "alice", ObjType::Map)?;

  // Finally we can set keys in the "alice" map
  peer1.put(&alice, "name", "Alice")?;
  peer1.put(&alice, "email", "alice@example.com")?;
}

// Create a state to track our sync with peer2
let mut peer1_state = sync::State::new();
// Generate the initial message to send to peer2, unwrap for brevity
let message1to2 = peer1.sync().generate_sync_message(&mut peer1_state).unwrap();

// We receive the message on peer2. We don't have a document at all yet
// so we create one
let mut peer2 = automerge::AutoCommit::new();
```

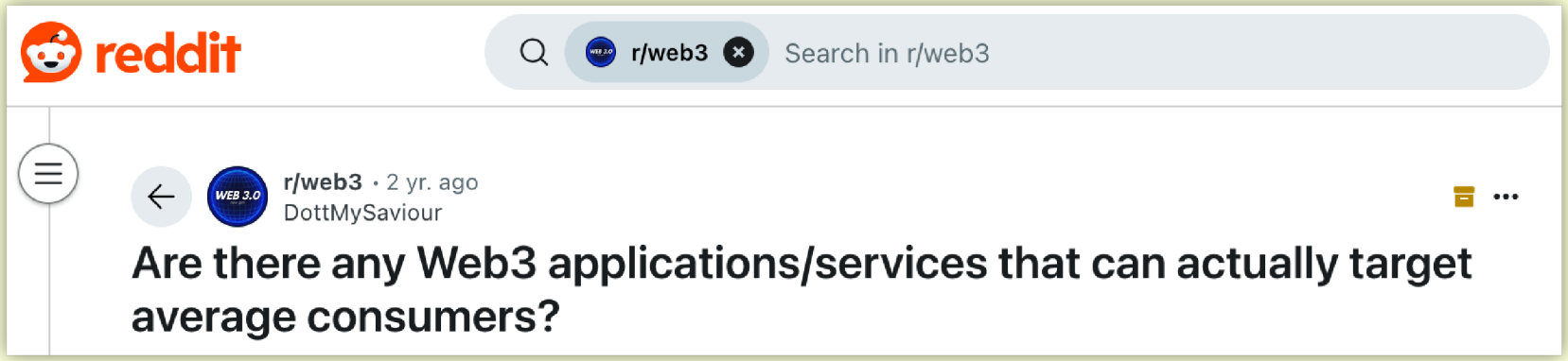
Adapted example from https://docs.rs/automerge/0.6.0/automerge/index.html

Application stack



https://www.youtube.com/watch?v=NMq0vncHJvU

Application stack



https://www.reddit.com/r/web3/comments/133f7j6/are_there_any_web3_applicationsservices_that_can/

δ-CRDTs

- State-based CRDTs require full state exchange
- Shapiro et al. "Delta State Replicated Data Types" (2018)
- δ-CRDTs send only minimal changes (deltas) and partial causal metadata. Similar to op-based, but different way to track causality.

Table 2. Comparison between CRDT approaches.

	operation-based		state-based	
	general	pure-operation	standard	delta-state
reuse sequential ADTs	commutative ops		idempotent inflations	
open vs closed systems	fixed set of participants		dynamic independent groups	
partition tolerance	unsuitable for long partitions		good	
state size	independent of replicas		component linear with replicas	
metadata amortization	transparent (middleware)		explicit (container)	
message size	normally small	small	large	possibly small
messaging (usual)	causal broadcast		epidemic at-least-once	
causal consistency	from messaging		for free	needs care
causal stability	useful	important	not available in general	

Paulo Sérgio Almeida. Approaches to conflict-free replicated data types. (2023)

δ-CRDTs — Tracking Causality

```
1     // On Node `A`
2     let vector_clock = { "A": 4, "B": 6 };
3     let mut state = json!({});
4
5     const diff = json!({ "foo" : "bar" });
6     vector_clock["A"]++; // Highest seq of "A" is now 5
7
8     const δ_a = { diff, vector_clock };
9
10
11    // On Node `B`
12    let vector_clock = { "A": 4, "B": 6 };
13
14    let mut state = json!({});
15
16    if (δ_a.vector_clock["A"] == vector_clock["A"] + 1) {
17        // contiguous sequence
18        state = join(state, δ_a.diff);
19    } else {
20        // handle duplicate or buffer
21    }
```

δ-CRDTs — OR Semantics

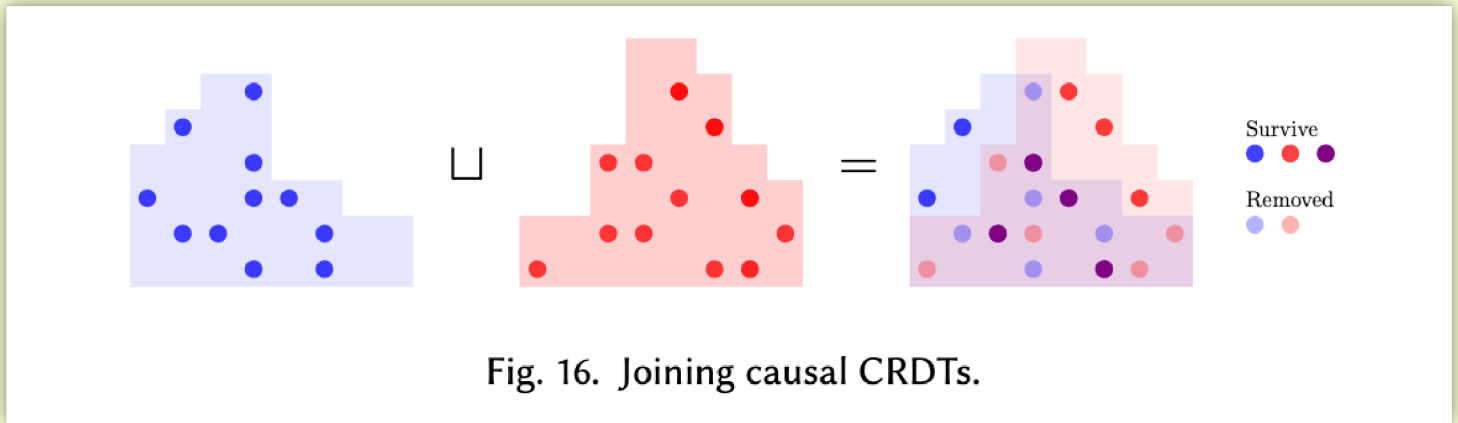


Fig. 16. Joining causal CRDTs.

Paulo Sérgio Almeida. Approaches to conflict-free replicated data types. (2023)