HermitDB

A private decentralized database replicated over Git
(or any other append only log)

Before jumping into the details, why do we need a new distributed database?

here's why I needed it:


mona
a password manager for hermits

I wanted a password manager that:

- is open source
- has mobile apps
- syncs across my devices
- and has some nice UX
  (why do closed source password managers look so good?).

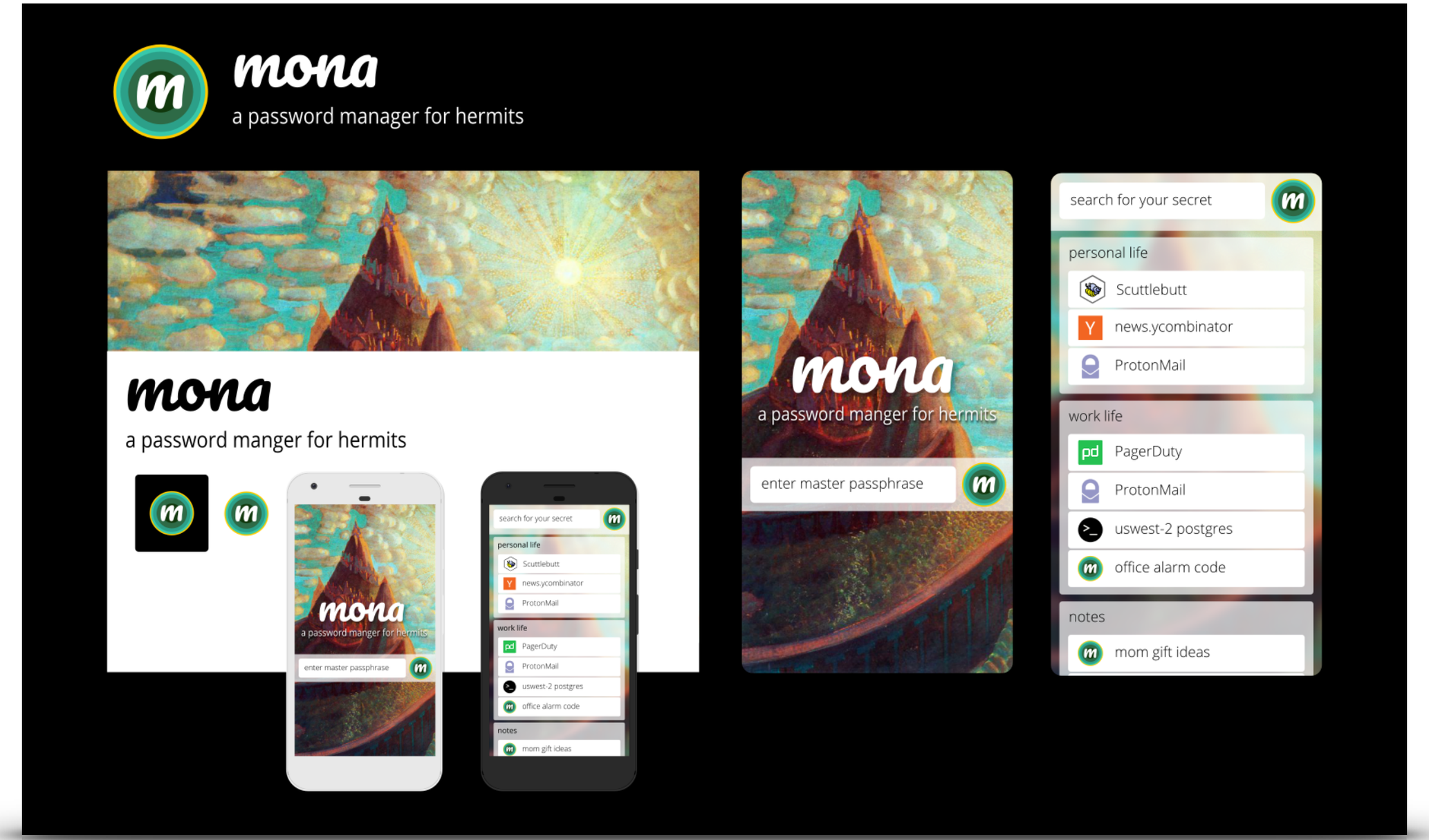
I wanted a password manager that:

- is open source
- has mobile
- syncs acros
- and has some nice UX

  (why do closed source password managers look so good?).

I couldn't find one that made me happy :(

so work starts on *mona*



Working on ***mona*** made a few things *super* obvious

1. Giving users agency over data is hard with existing tech.
2. Append-only logs are fantastic and they are everywhere.
3. CRDT's are the answer to merging out of sync data

Working on ***mona*** made a few things *super* obvious

1. Giving users agency over data is hard with existing tech.
2. Append-only logs are fantastic and they are everywhere.
3. CRDT's are the answer to merging out of sync data

HermitDB tries to solve 1. by building on insights 2. and 3.

Tools built with HermitDB give users agency over their data.

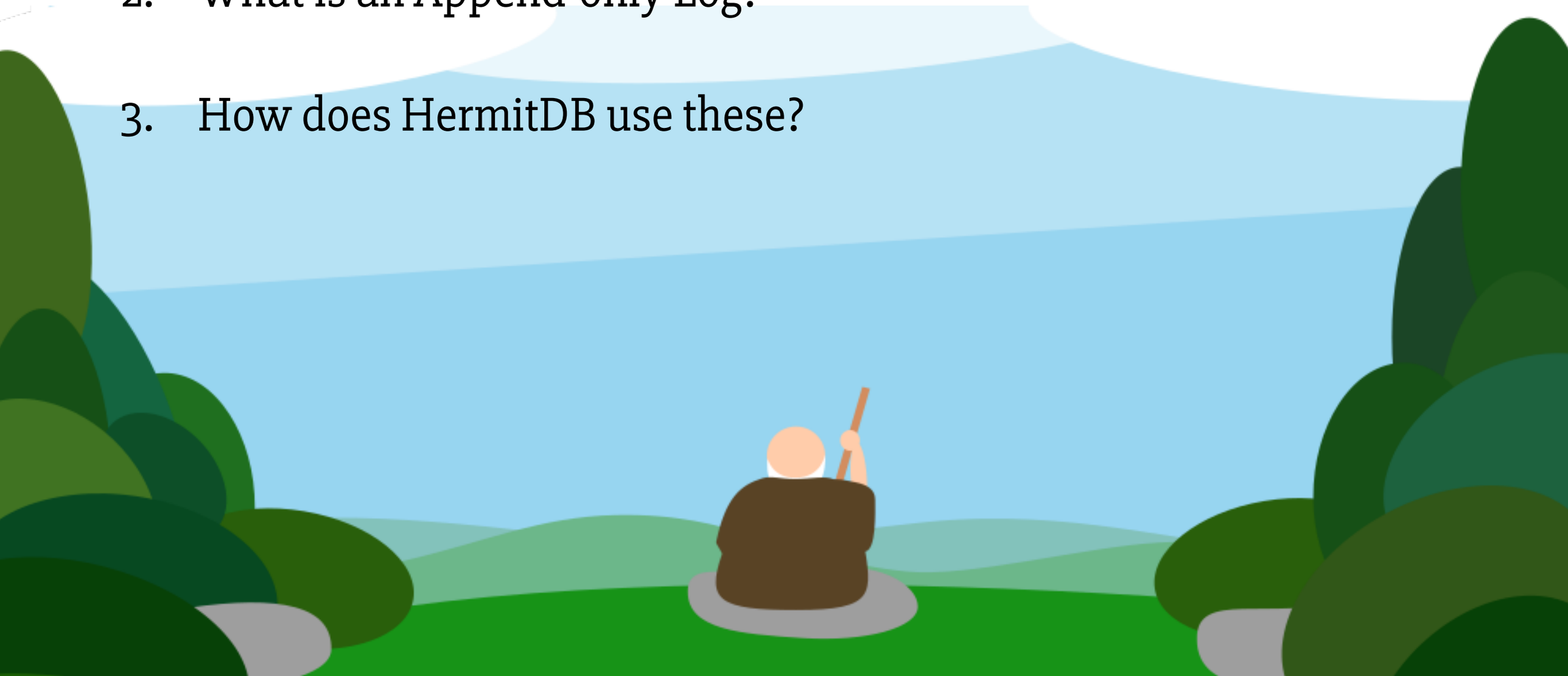
into the weeds we go

our plan

1. What is a CRDT?

2. What is an Append-only Log?

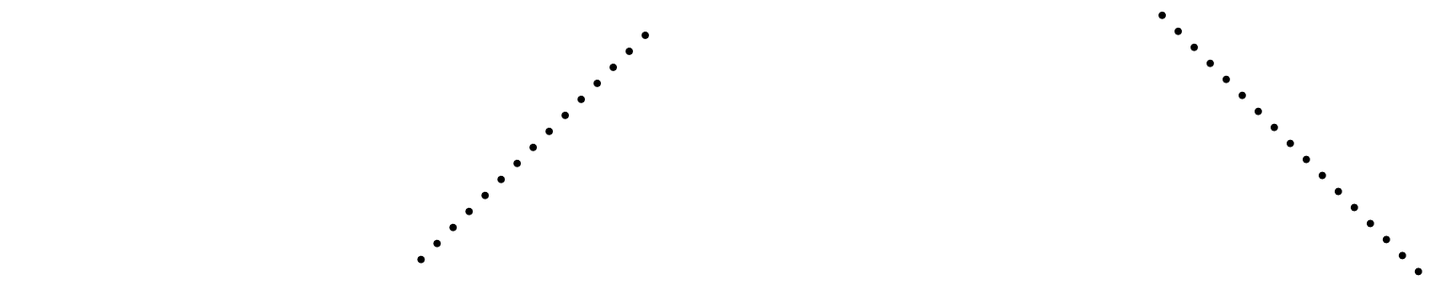
3. How does HermitDB use these?



The CRDT

(Conflict-free Replicated Data Types)

CRDT's breaks down into two categories



CmRDT

C(ommutative)RDT

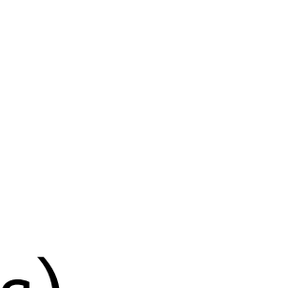
CvRDT

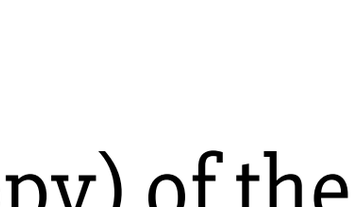
C(onvergent)RDT

The CRDT

(Conflict-free Replicated Data Types)

Alice and Bob both have a replica (copy) of the data structure

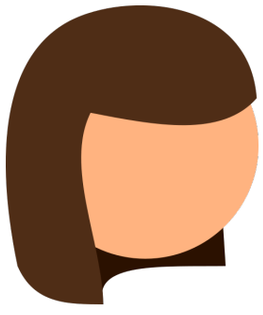


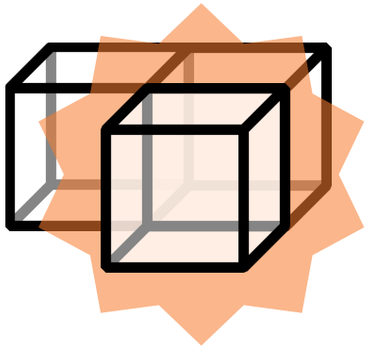


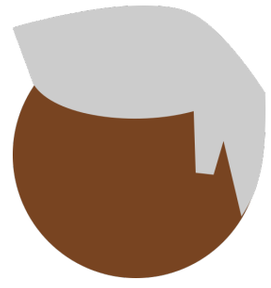
The CRDT

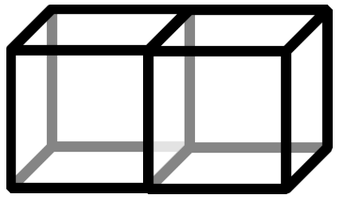
(Conflict-free Replicated Data Types)

Alice makes an edit to her replica

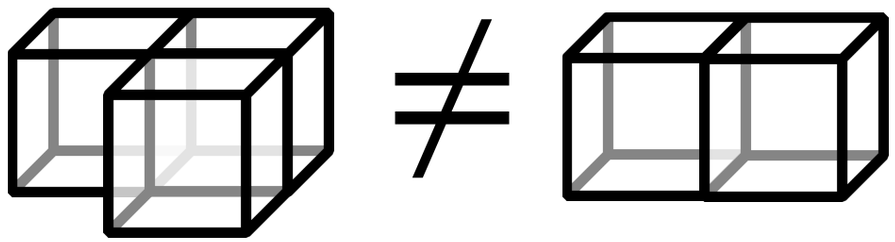








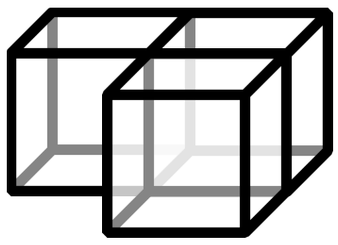
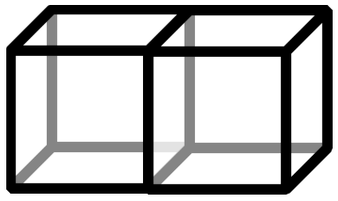
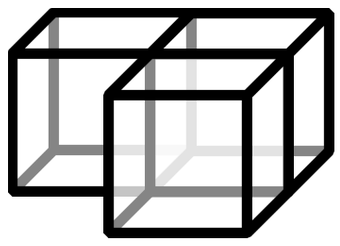
and now we are out of sync!



The CvRDT and the CmRDT datatypes have different approaches to bringing these replica’s back in sync

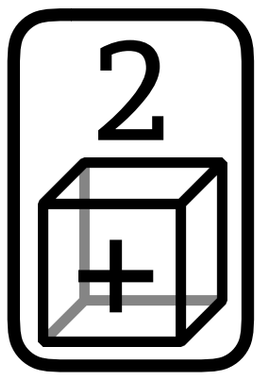
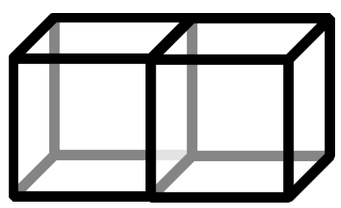
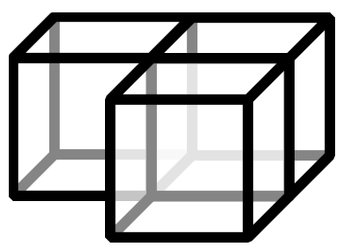
CvRDT:

We ship Alice's entire state over the network and merge with Bob's

merge( , ) = 

CmRDT:

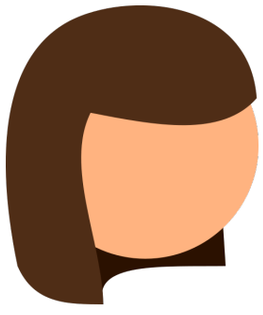
We ship a description of the edit (an Op) which is applied on Bob's

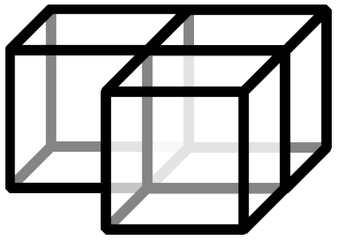
apply( , ) = 

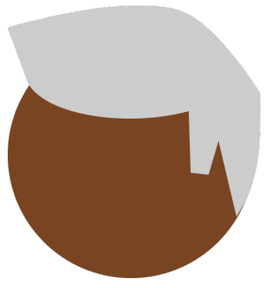
The CRDT

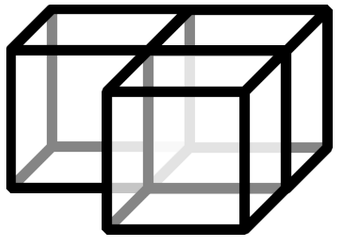
(Conflict-free Replicated Data Types)

In the end, both CvRDT and CmRDT
will result in consistent replica's.







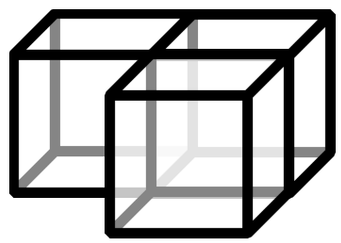


The CRDT

(Conflict-free Replicated Data Types)

CRDT's guarantee *strong* eventual consistency:

All nodes in a system will eventually converge to the same state with *no* coordination needed to handle conflicts.

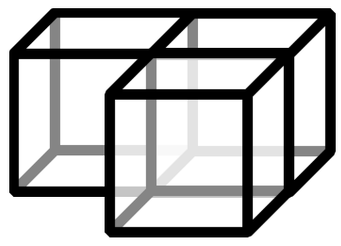


The CRDT

(Conflict-free Replicated Data Types)

Now you may be wondering:

Can we implement such a `merge` or `apply` for any structure?

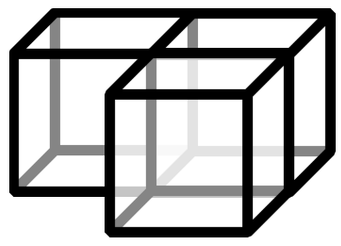


The CRDT

(Conflict-free Replicated Data Types)

Now you may be wondering:

Can we implement such a `merge` or `apply` for any structure?



not quite...

The Structure of CRDT's

(aka The Join-Semilatice)

<u>All CRDT's form a Join-Semilatice.</u>

This means you'll need:

1. a partial order over (your state space)
2. a lub (least upper bound) for any nonempty subset of your state space
3. and closure of lub over S

In math, we'd say:

$\forall a, b \in S$

1. $merge(a, b) = lub\{a, b\}$
2. $merge(a, b) \in S$

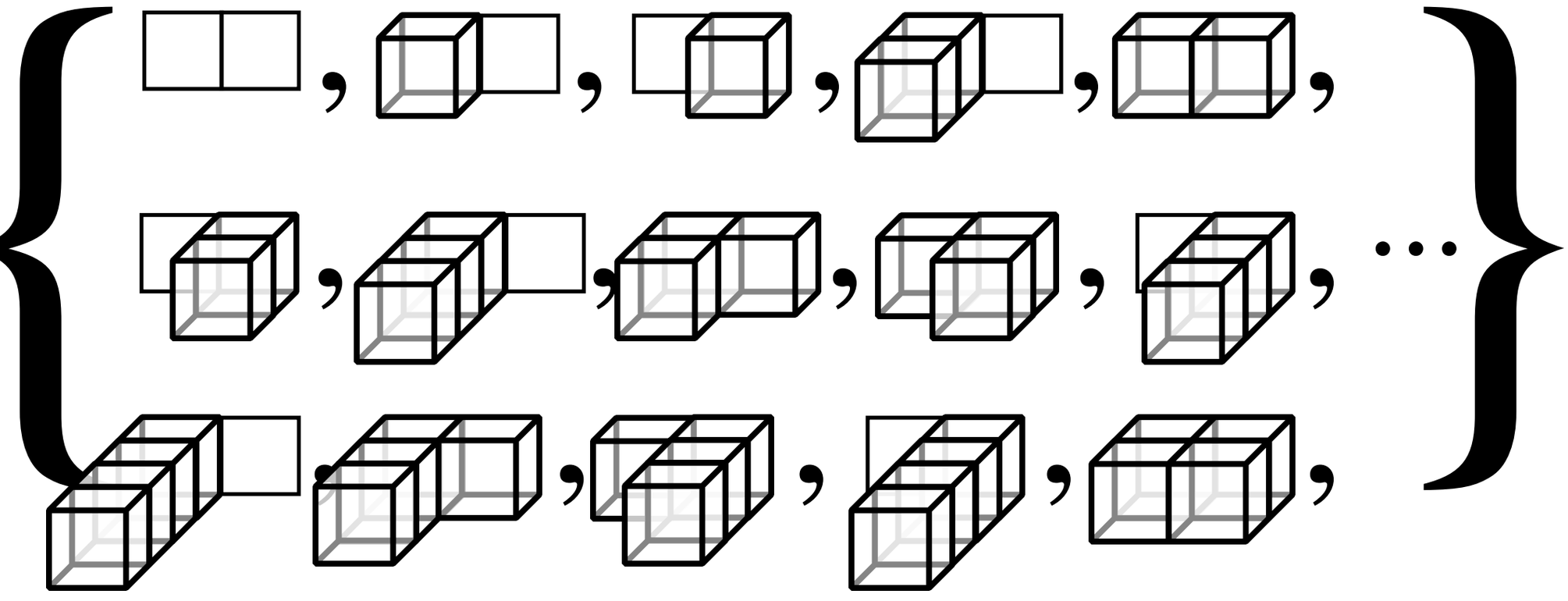
$\implies CvRDT$

(the algebra for CmRDT's is a bit more complicated but all ideas still apply)

The Structure of CRDT's

(aka The Join-Semilatice)

Let's look at the state space of Alice and Bob's CRDT

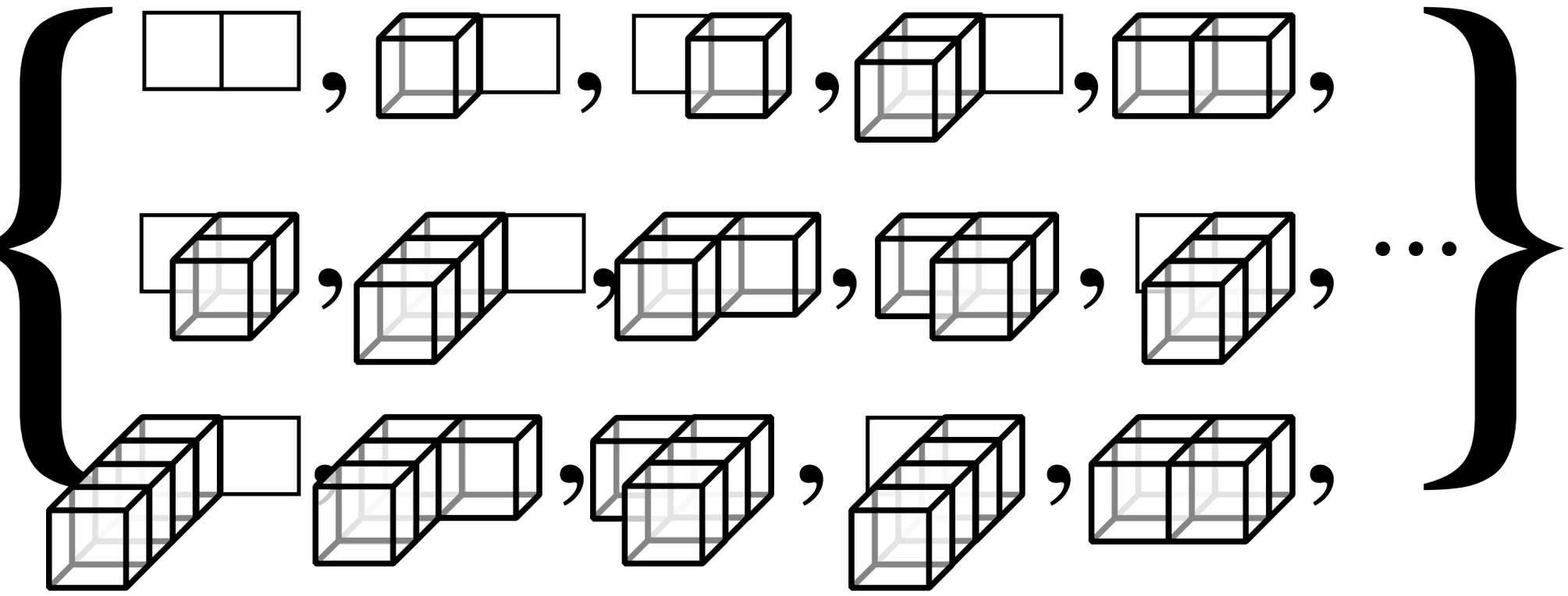
$S =$ 

$\forall a, b \in S$

1. $merge(a, b) = lub(a, b)$
2. $merge(a, b) \in S$

The Structure of CRDT's

(aka The Join-Semilatice)

$S =$ 

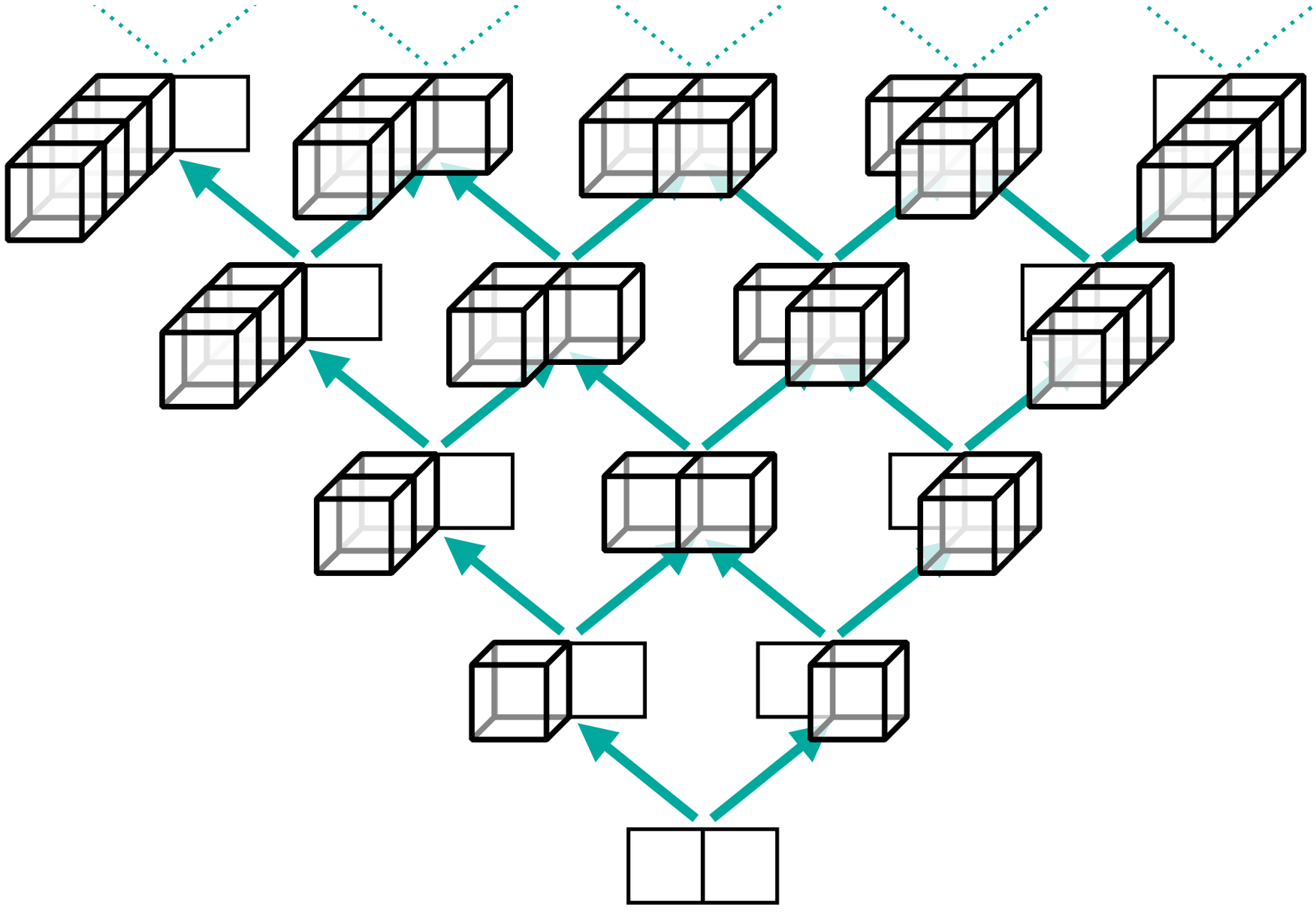
$\forall a, b \in S$

1. $merge(a, b) = lub(a, b)$
2. $merge(a, b) \in S$

The partial order we define on S will decide our merge operations

The Structure of CRDT's

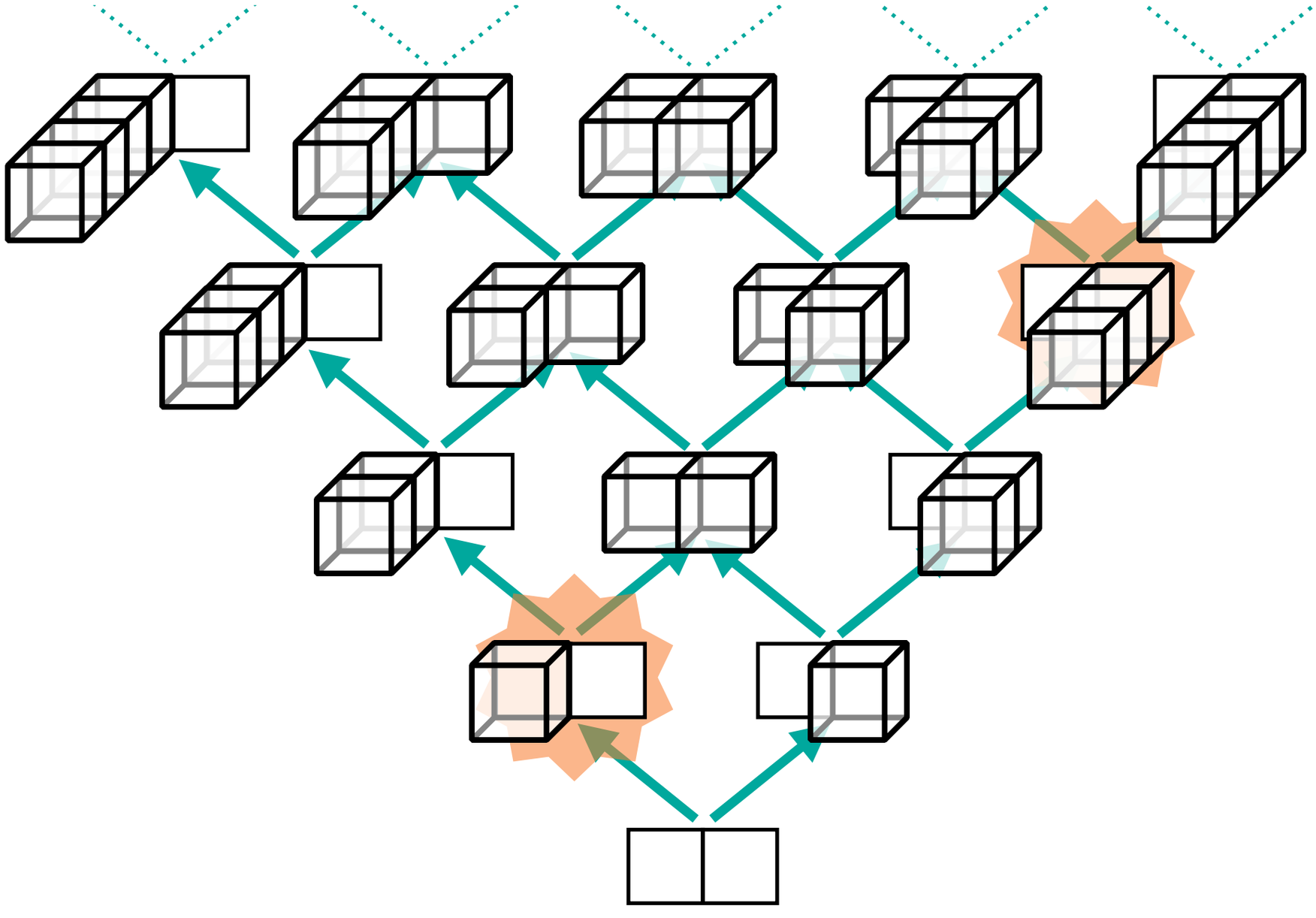
(aka The Join-Semilatice)

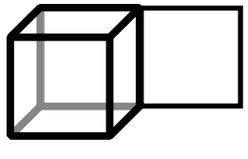
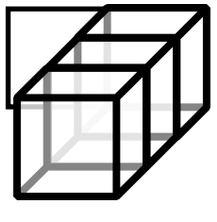


a potential partial order over S

The Structure of CRDT's

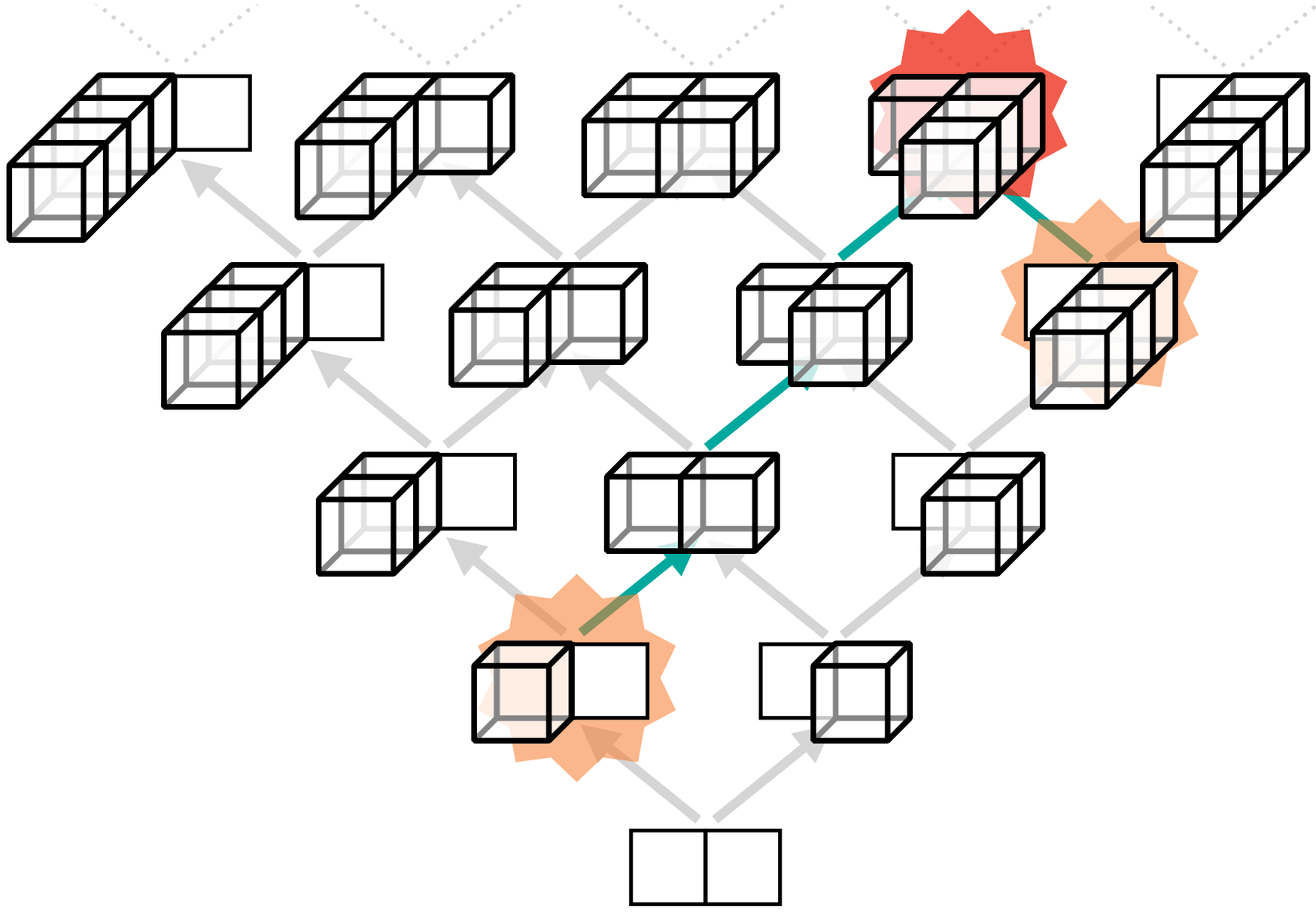
(aka The Join-Semilatice)

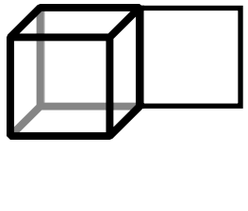
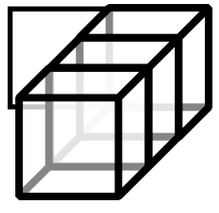
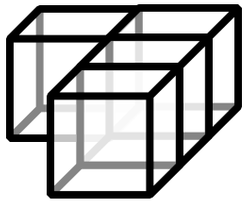


e.g. merge( , ) =

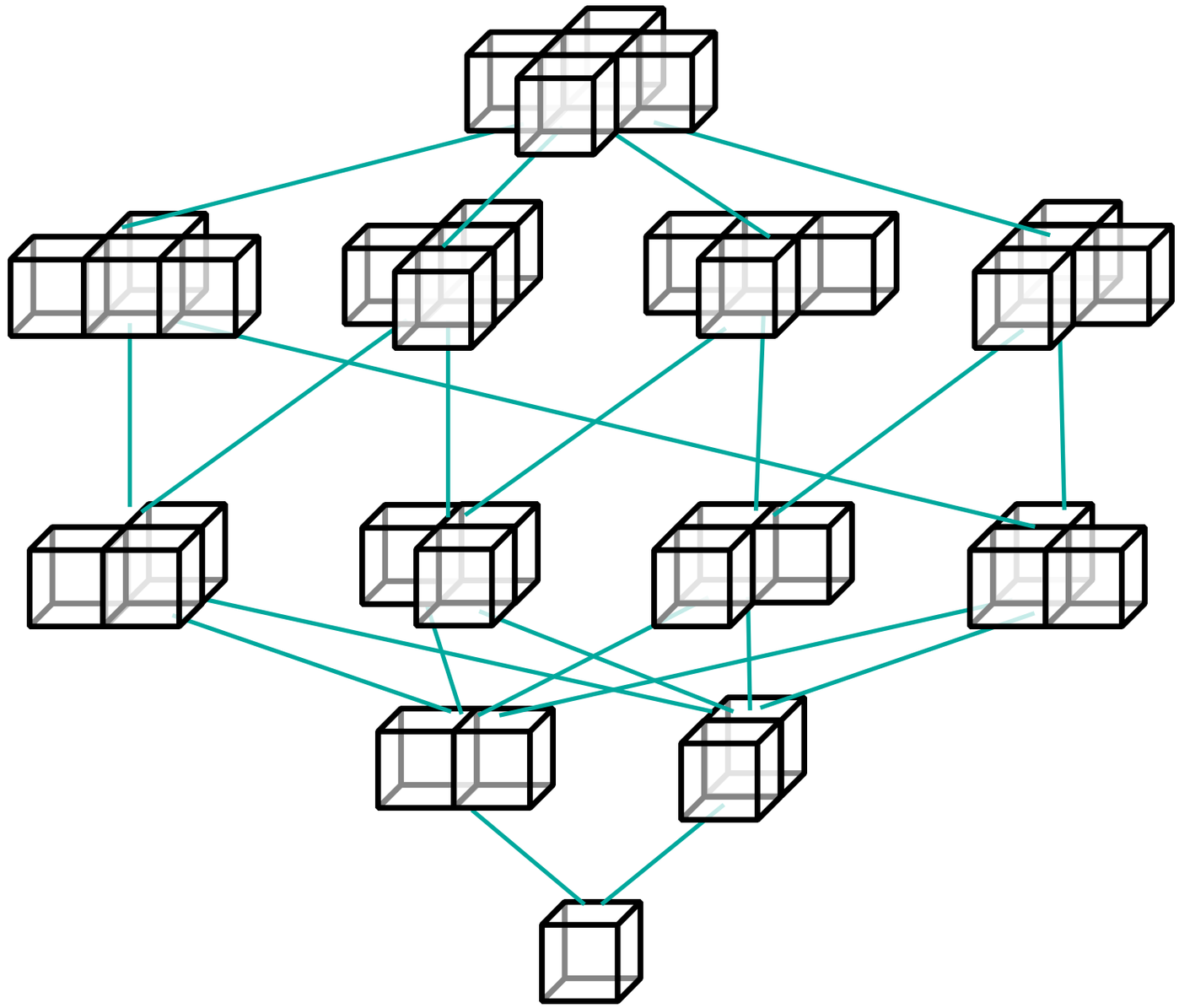
The Structure of CRDT's

(aka The Join-Semilatice)

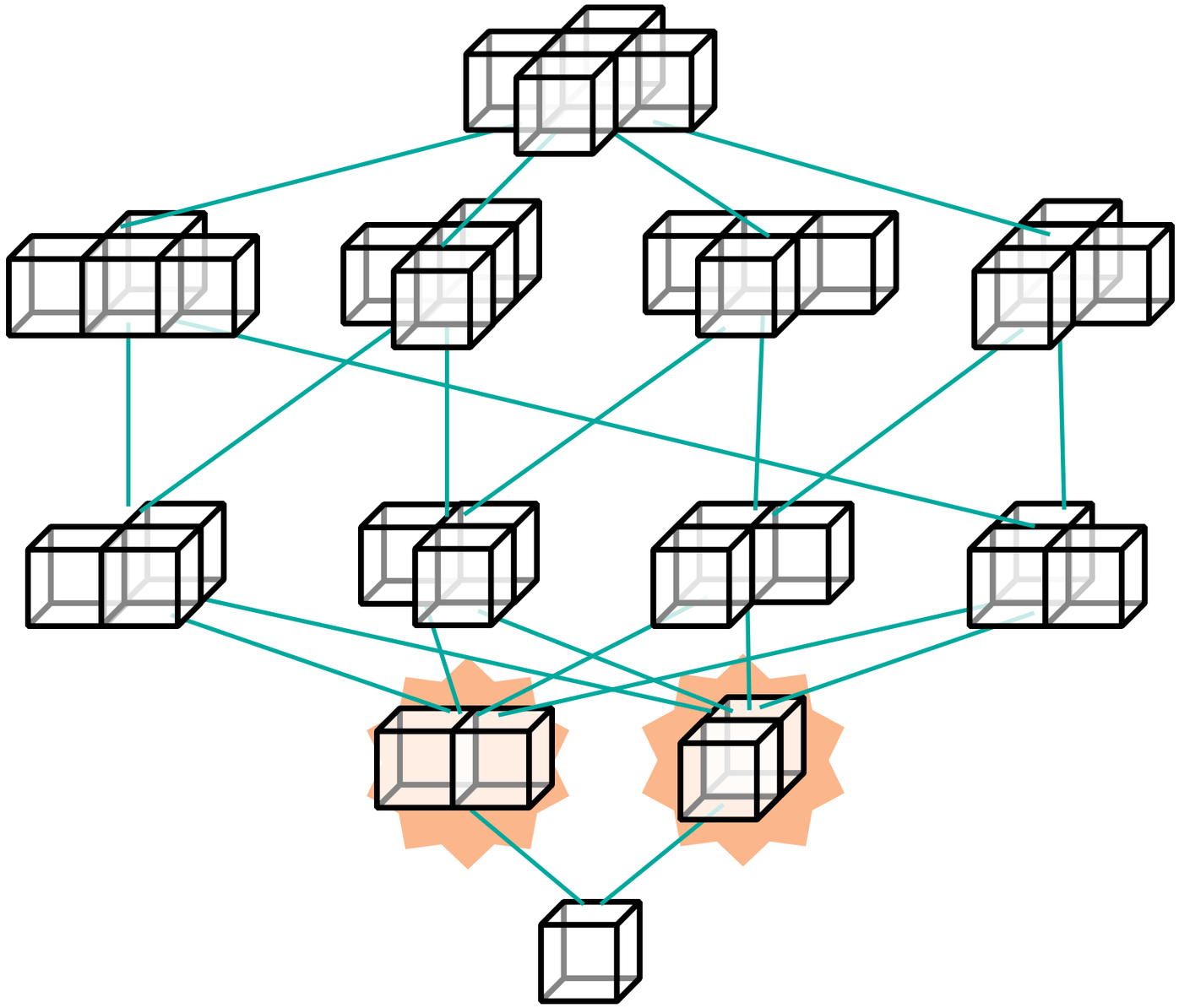


e.g. merge( , ) = 

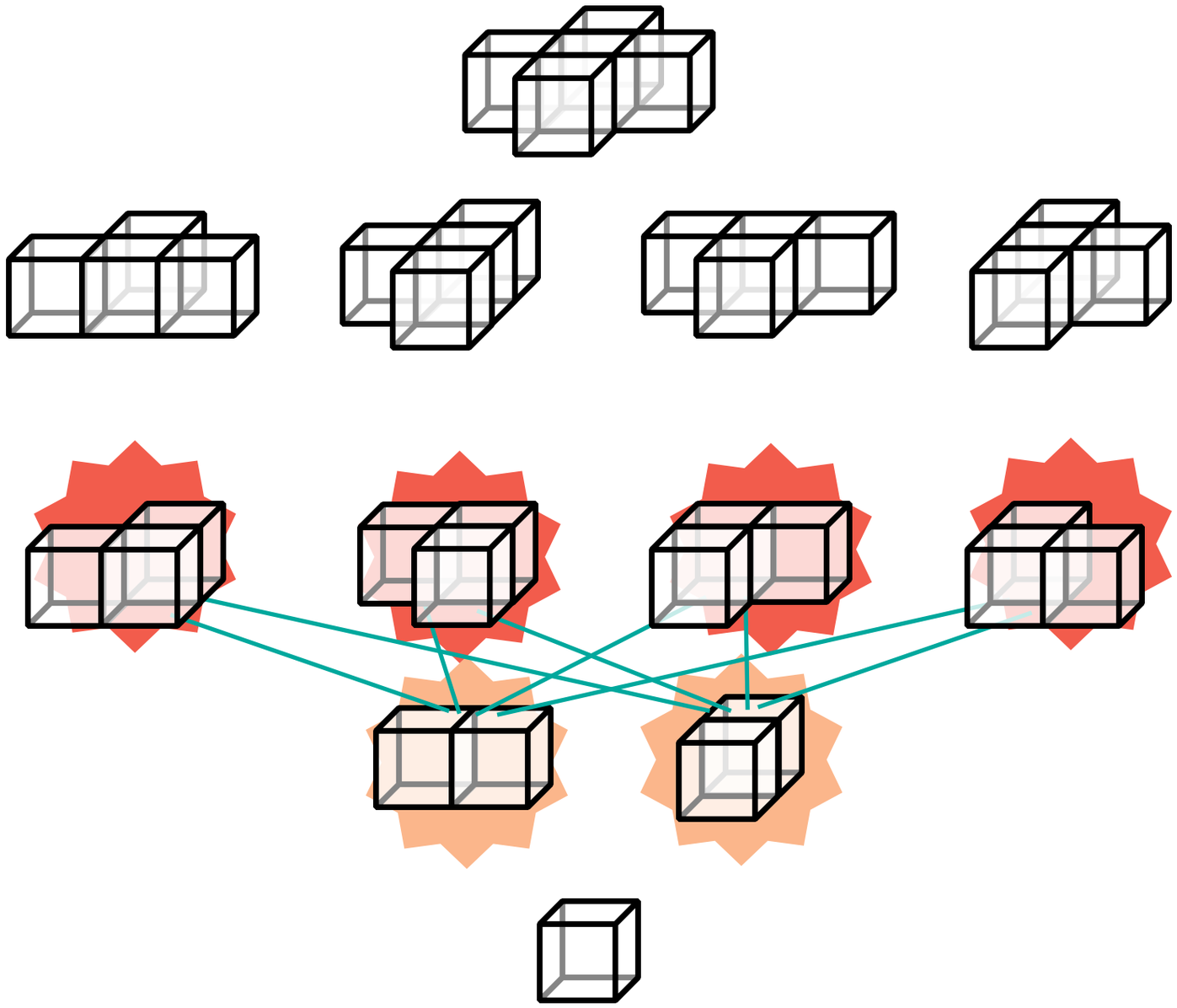
is this a CRDT?



is this a CRDT?

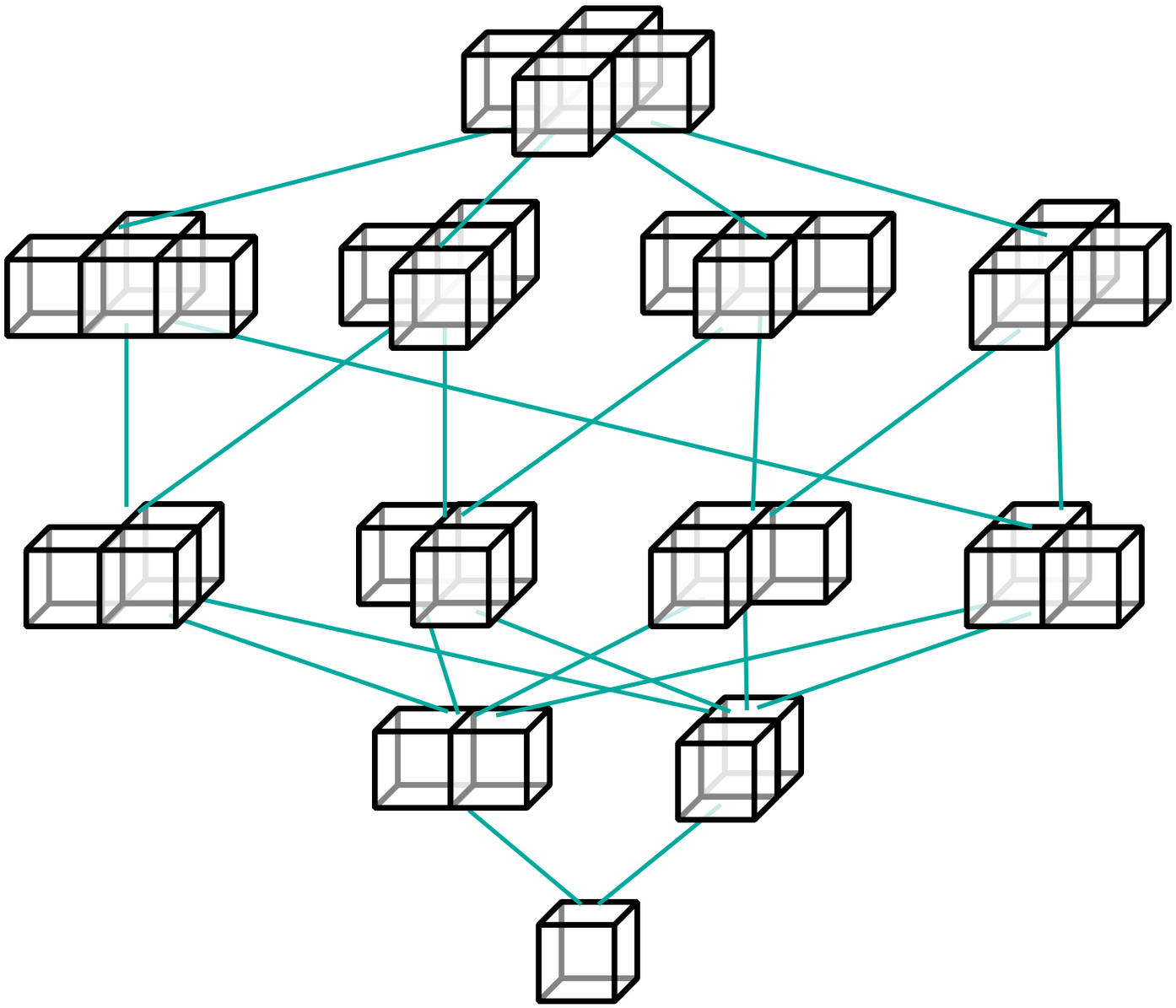


is this a CRDT?



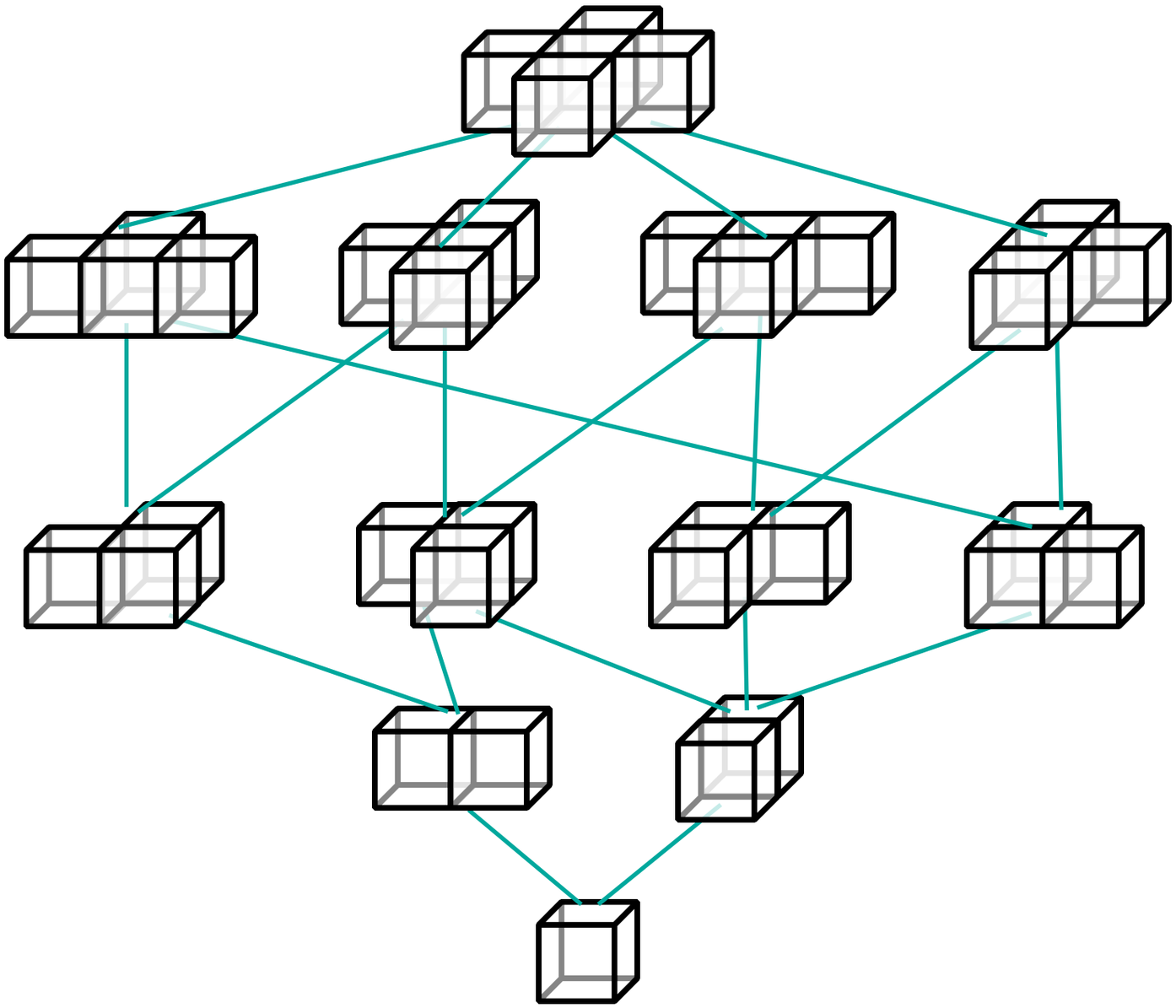
Nope, we need a LUB (least upper bound)

Lets turn it into a CRDT



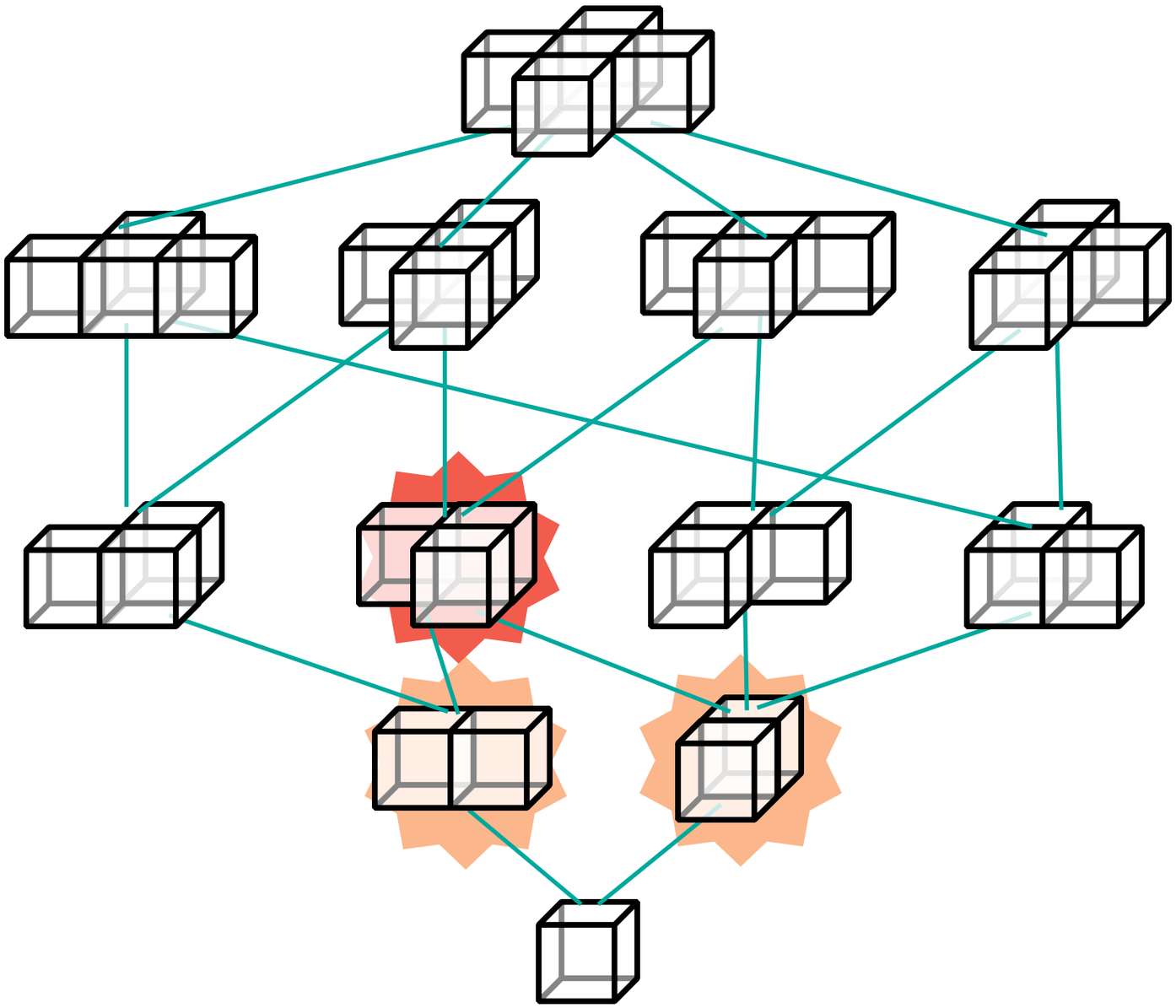
by removing some edges

Lets turn it into a CRDT



by removing some edges

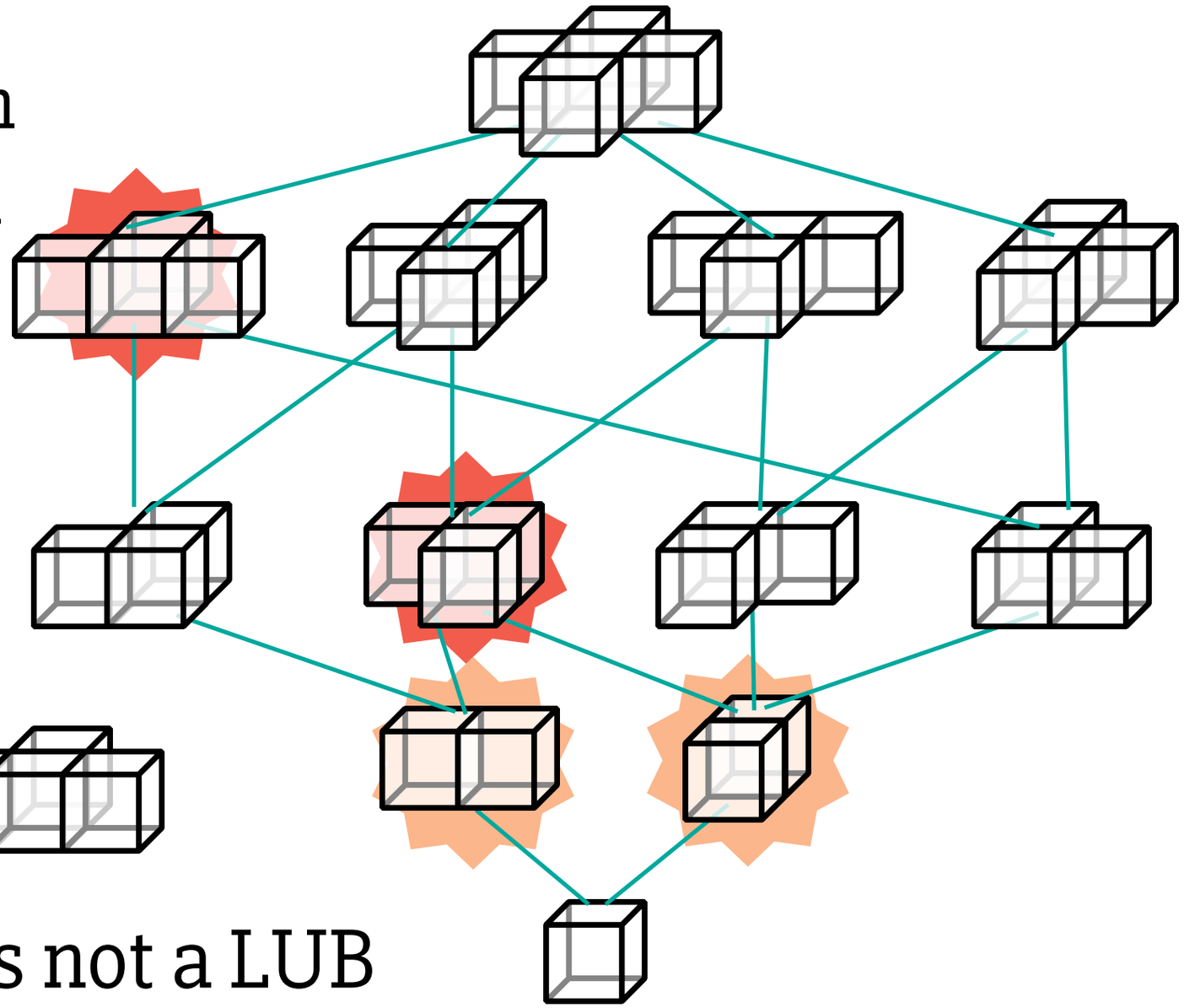
Lets turn it into a CRDT

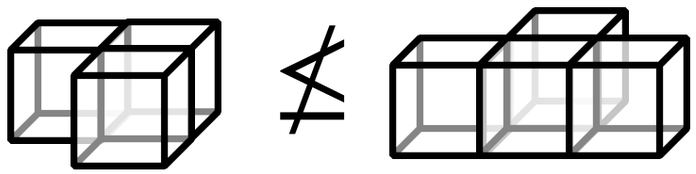


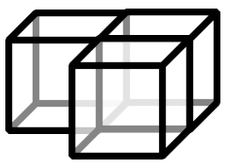
does it work?

Lets turn it into a CRDT

This is also an
Upper Bound

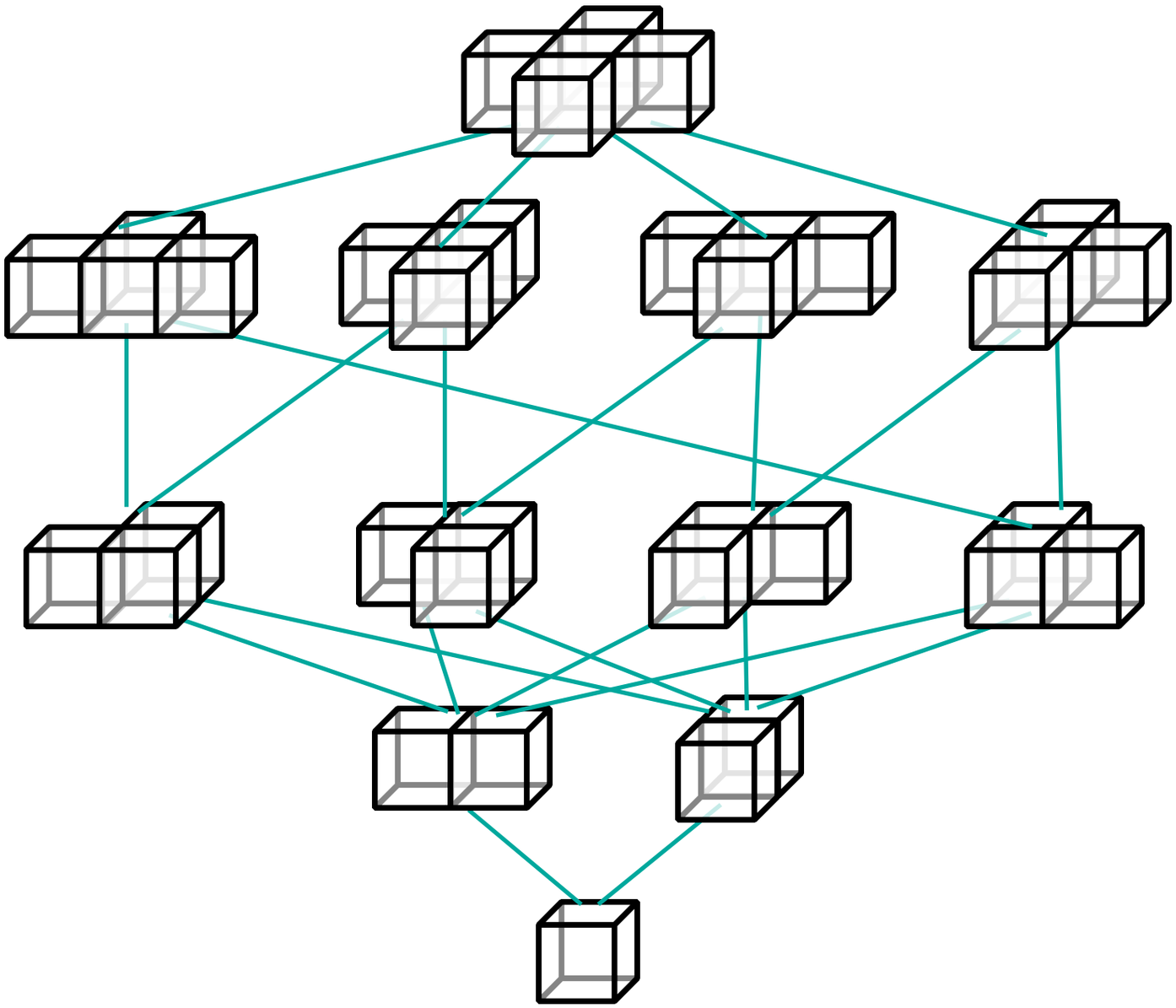





is not a LUB

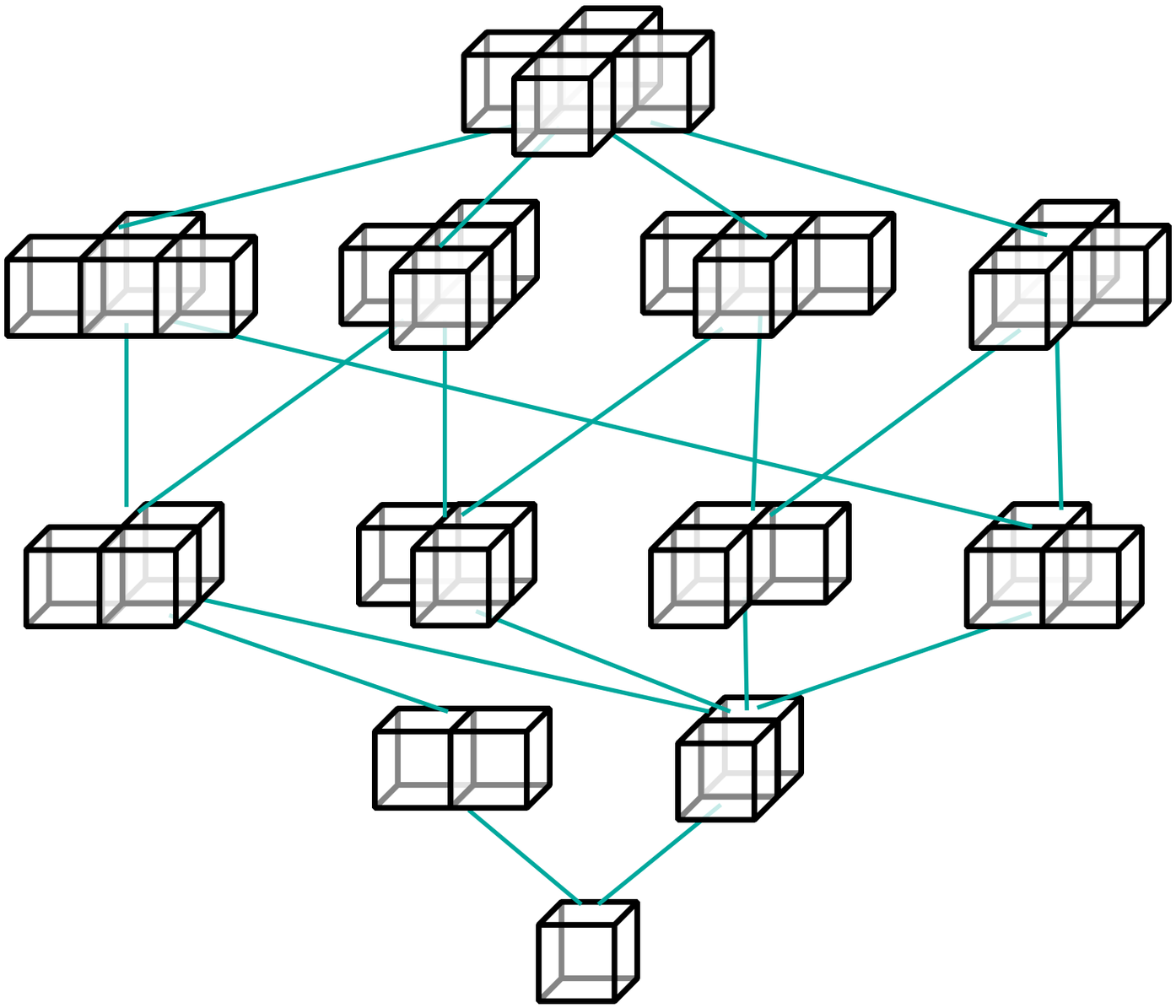
not quite..

Lets turn it into a CRDT



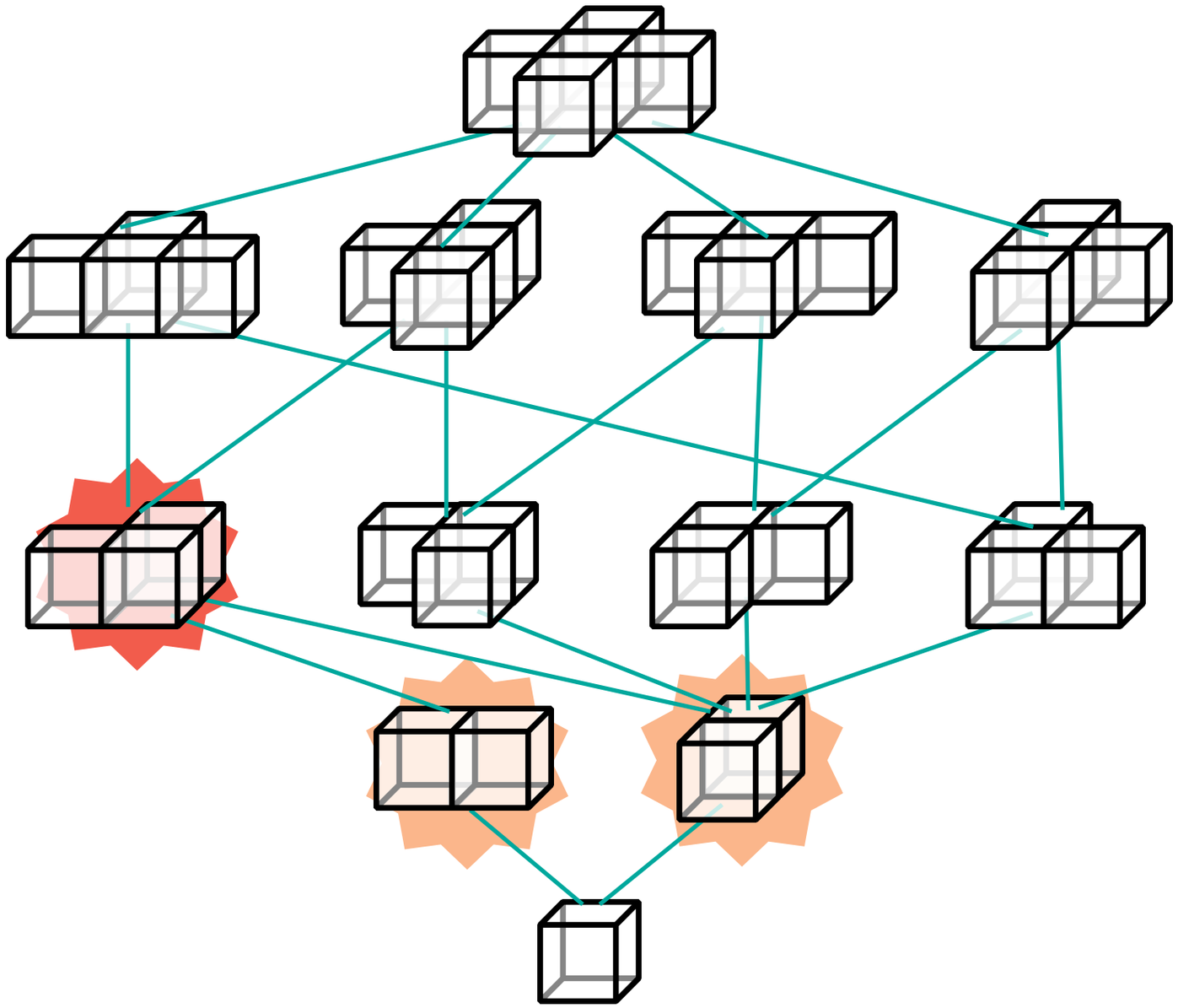
Trying again..

Lets turn it into a CRDT



This will work :)

Lets turn it into a CRDT



This will work :)

My point is be careful when designing your own CRDT's.

You'll often need to introduce a bias into a structure to make it a CRDT

Not every structure has a symmetry.

Many CRDT's Exists

- Registers (store a value)
  - MVReg - A multi-value register
  - LWWReg - Last-write-wins register
- Sets
  - GSet - Grow-only Set
  - 2PSet - 2 phase set
  - ORSWOT - Observed Remove Set Without Tombstones
- Map's (map flavours of all the Set CRDT's exist)
- Sequences (often used in collaborative editors)
  - LSeq
  - RGA

And many more, you'll need to choose the semantics you care about when designing your system

HermitDB is one big CRDT
Key / Value store where the Values themselves are also CRDT’s

Think about warping your biz requirements to fit the CRDT model.

It'll solve your sync problems.

Append-only Logs

This data structure
is pretty much what it sounds like.

The log is a primitive that shows
up in many eventually consistent
distributed systems.

Append-only Logs

1. Git
   We see it in the branch commit history

2. Kafka
   Distributed log service used widely in industry

3. Cryptocurrencies
   The blockchain is a persistent distributed log

4. SMS?

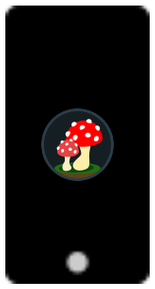
The HermitDB Log Interface

1. Log history must be (semantically) immutable
2. fn next() - read the next unacked message
3. fn commit(msg) - log a message without ack
4. fn ack(msg) - acknowledge a message
5. fn push(remote) - push messages to remote log
6. fn pull(remote) - fetch messages from remote log

How it all fits together

Imagine you are on your phone
and you'd like to remove an entry
from HermitDB

How it all fits together



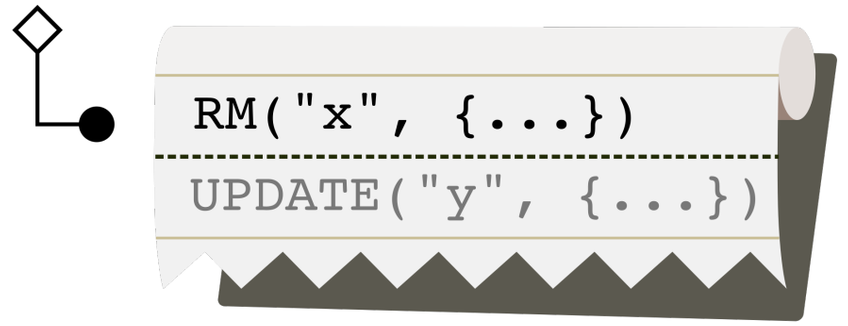
```
142    db.rm("x");
```

Imagine you are on your phone and you'd like to remove an entry from HermitDB

1. db.rm() is called

How it all fits together

```
142    db.rm("x");
```



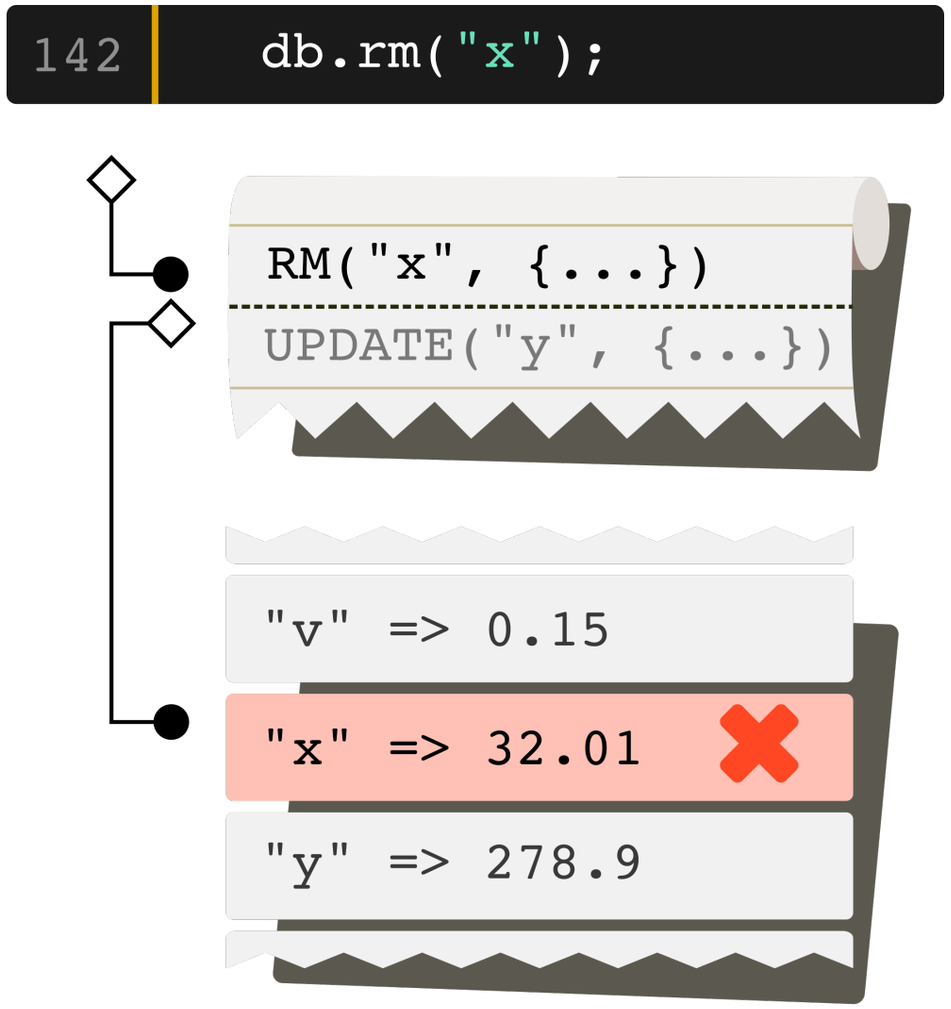
Imagine you are on your phone and you'd like to remove an entry from HermitDB

1. db.rm() is called
2. The remove Op is committed to the log

How it all fits together

Imagine you are on your phone and you'd like to remove an entry from HermitDB

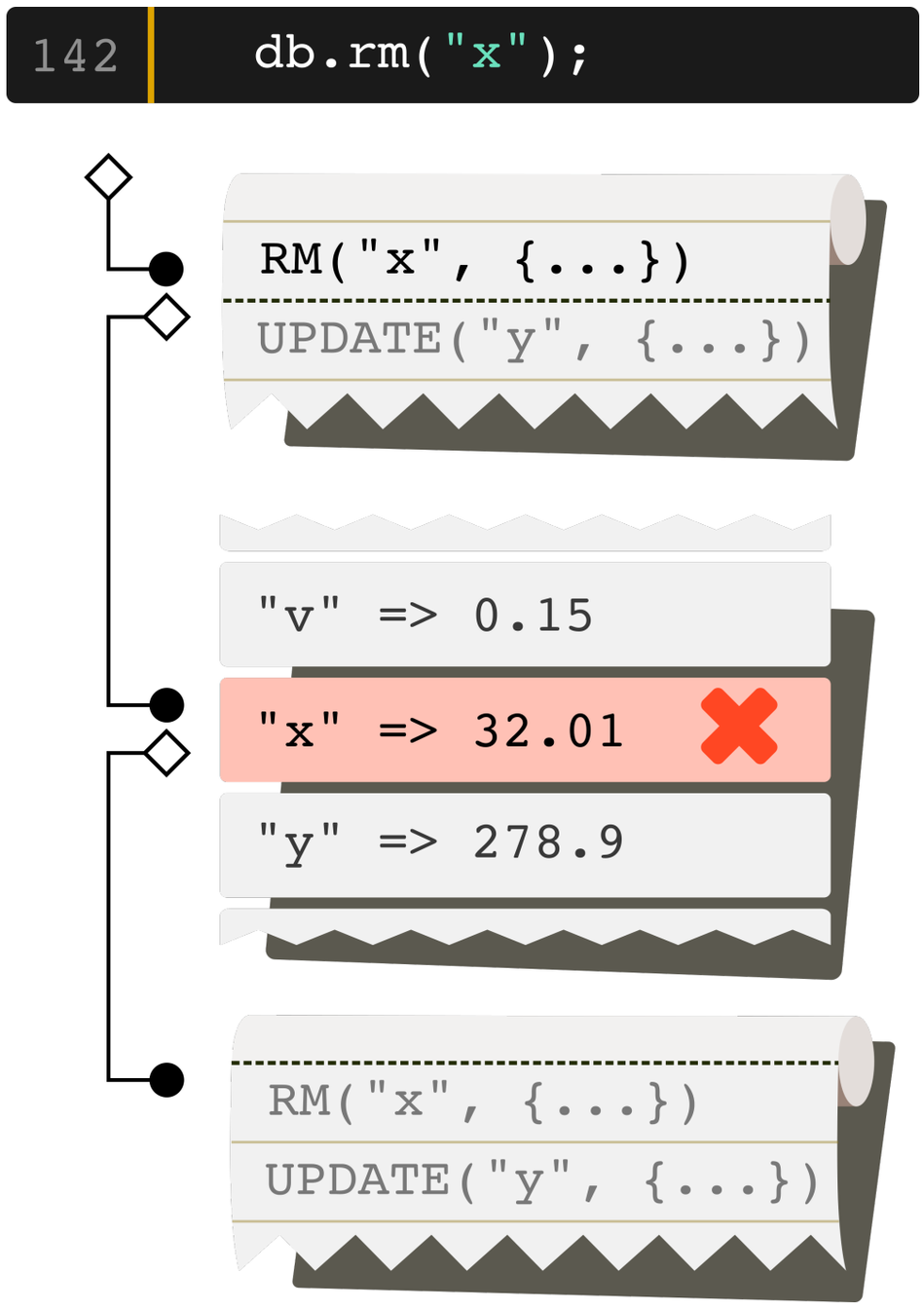
1. db.rm() is called
2. The remove Op is committed to the log
3. The Op is applied against our local state



How it all fits together

Imagine you are on your phone and you'd like to remove an entry from HermitDB

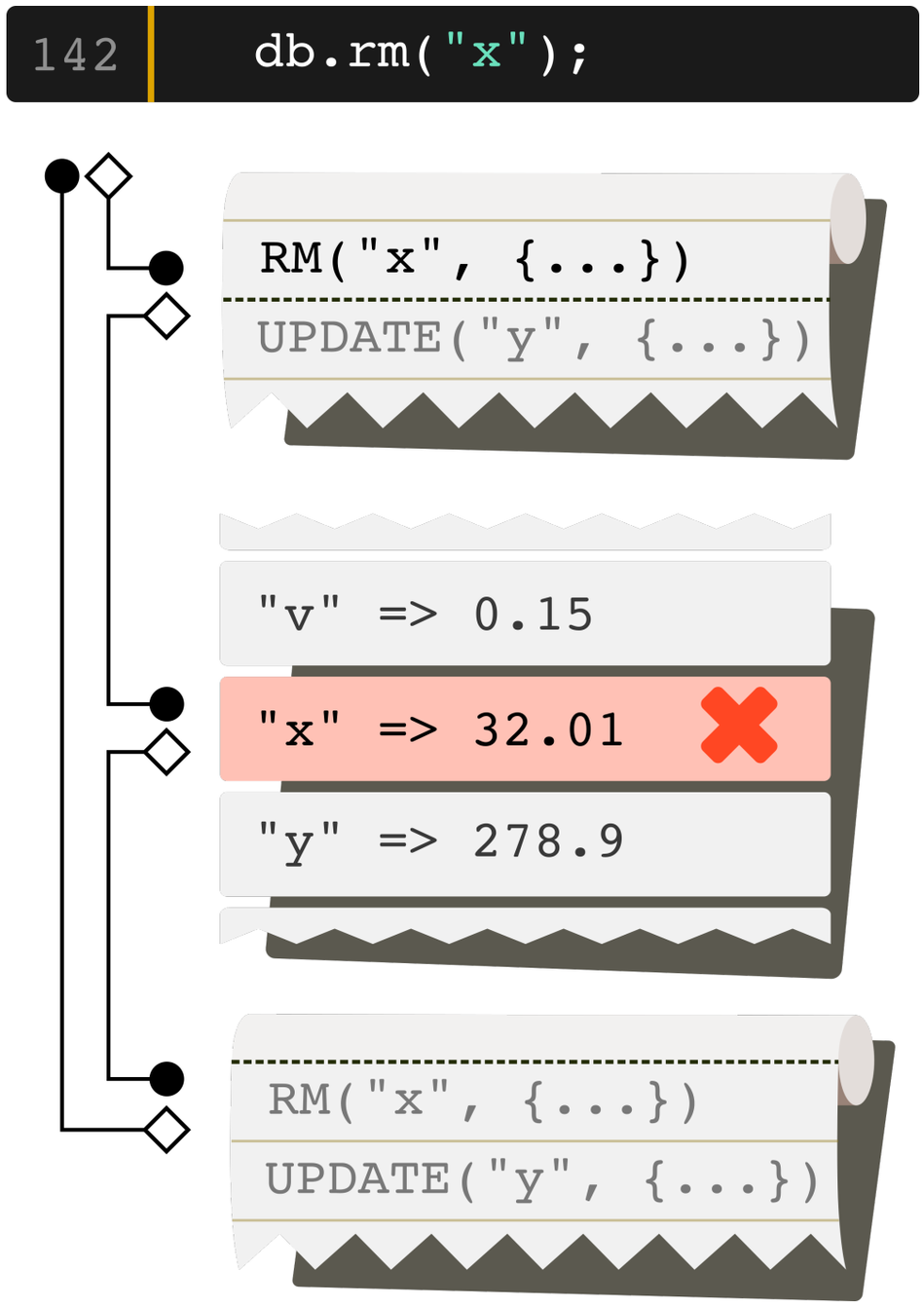
1. db.rm() is called
2. The remove Op is committed to the log
3. The Op is applied against our local state
4. The Op is marked as acked on the log



How it all fits together

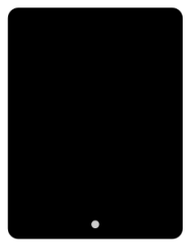
Imagine you are on your phone and you'd like to remove an entry from HermitDB

1. db.rm() is called
2. The remove Op is committed to the log
3. The Op is applied against our local state
4. The Op is marked as acked on the log
5. db.rm() returns

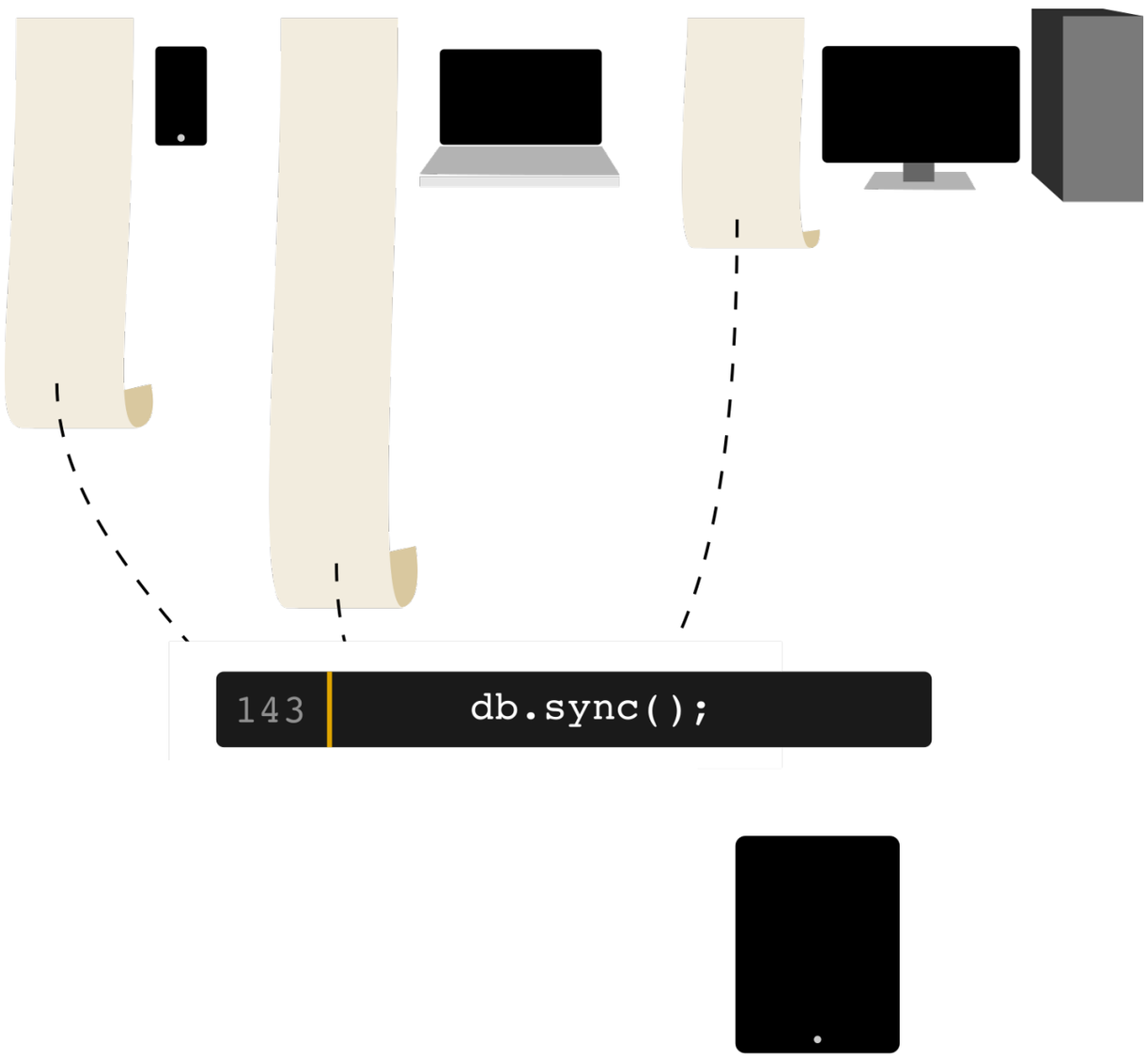


And now your tablet wants to syncs

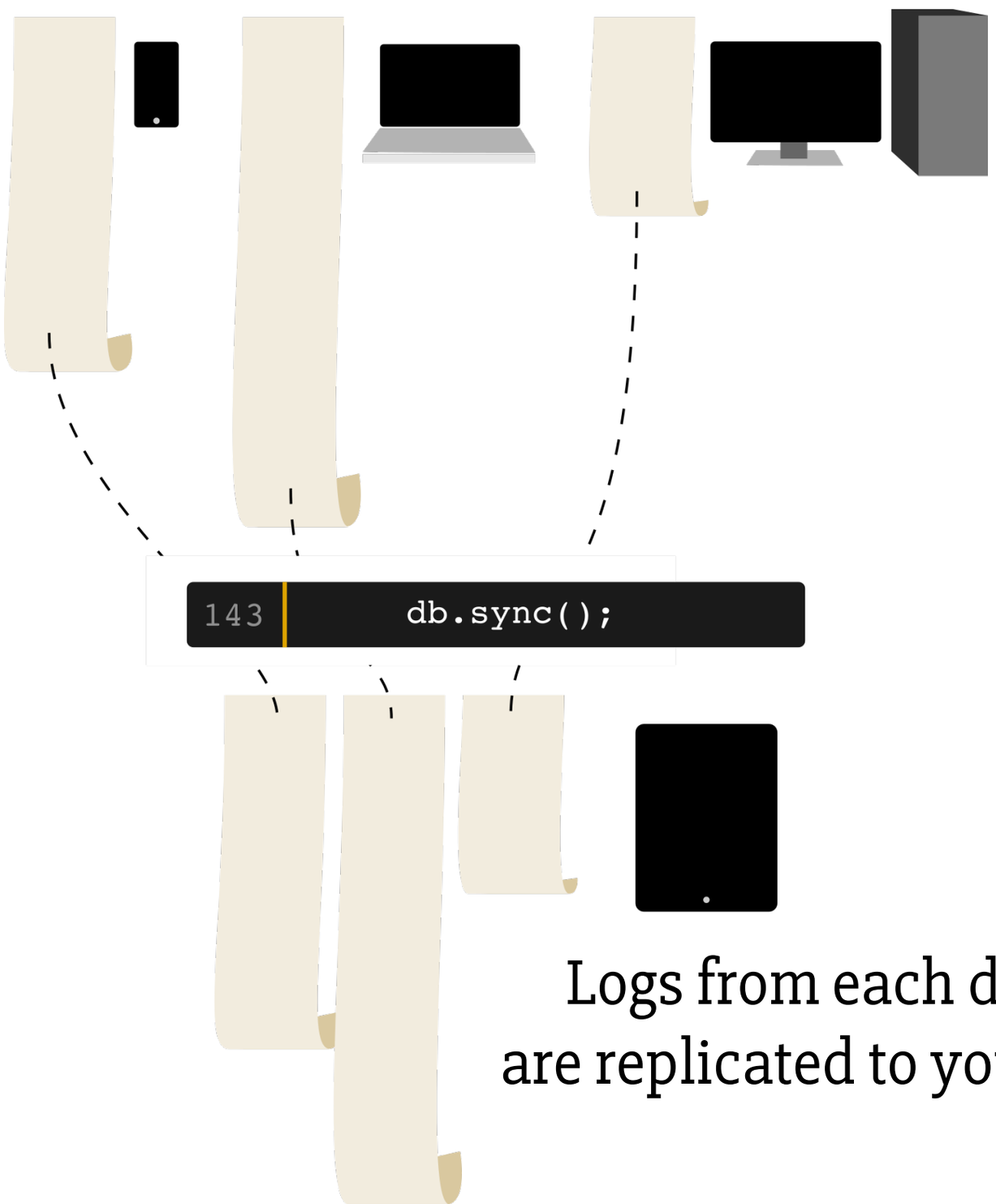
```
143    db.sync();
```



And now your tablet wants to syncs

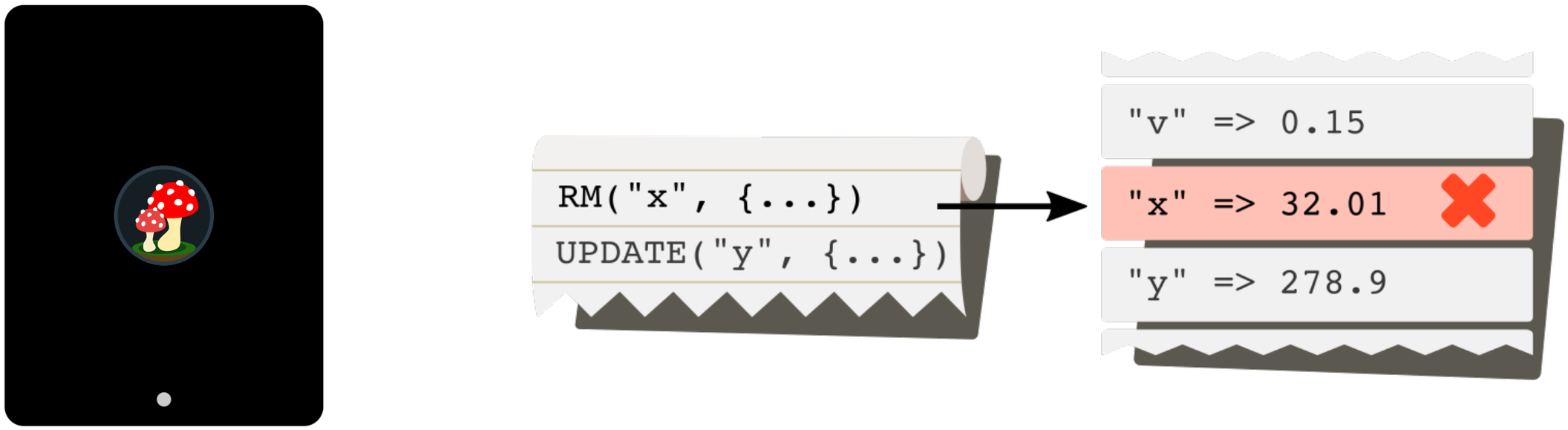


And now your tablet wants to syncs



Logs from each device
are replicated to your tablet

Log replay begins on the tablet



Since HermitDB is a CRDT, this process is guaranteed to converge

What does this give us?

You can now build applications that are give users agency over their data.

poke around, get involved :)

hermits-grove/hermitdb

http://hermitdb.com/

hermits-grove/mona

http://getmona.ca