Event Sourcing without Responsibility

Michael Sperber

@sperbsen

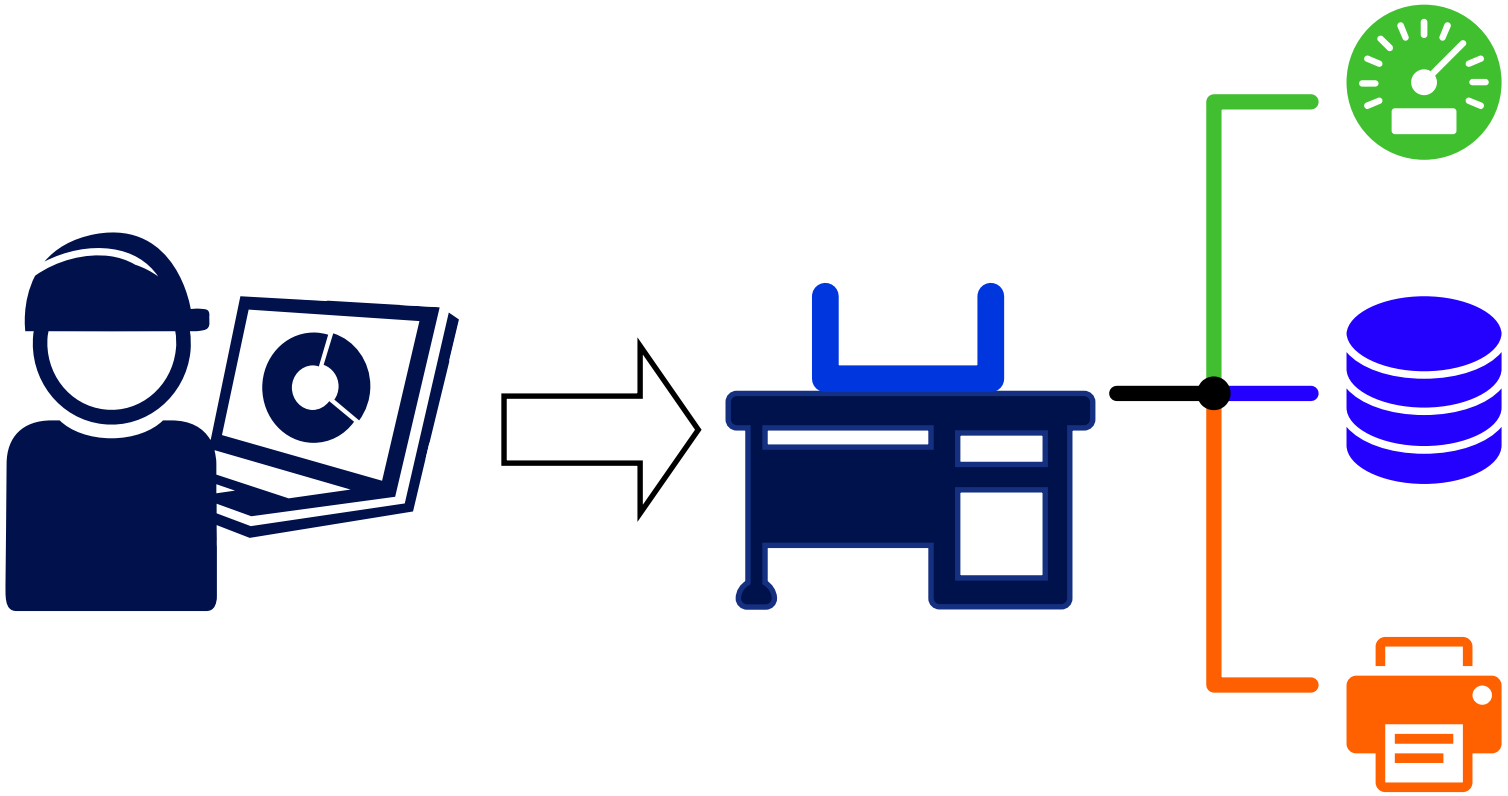
- software project development
- in many fields
- Scala, Clojure, Erlang, Haskell, F#, OCaml
- training, coaching
- iSAQB Advanced training
  Functional Architecture

www.active-group.de

funktionale-programmierung.de


2015

Configure Mobile Devices Automatically



Lokalisierung

Extras Backup-Funktionen

Active-Group 241 (automatisch)

TSC

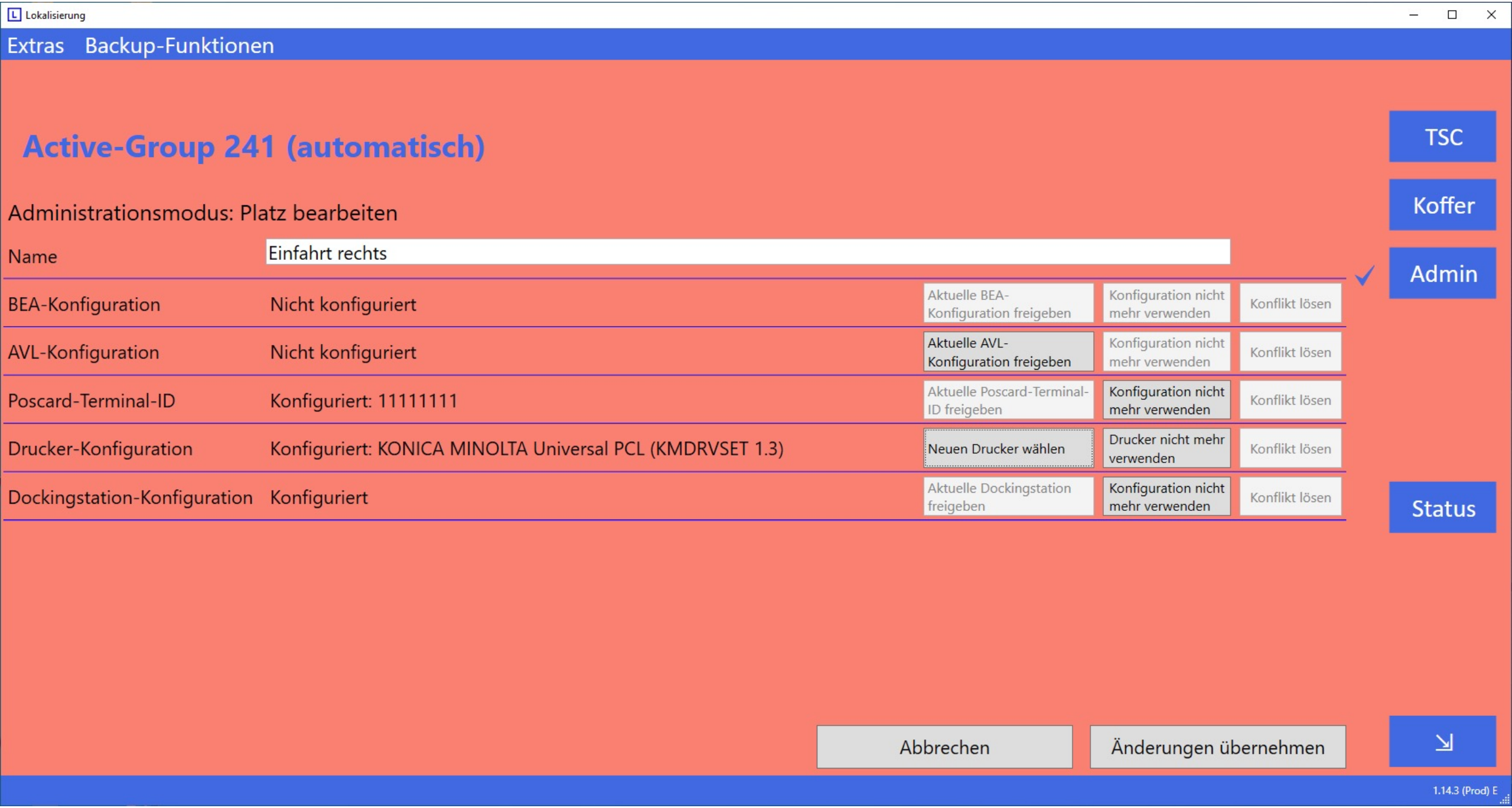
Koffer

Admin

PKW Einfahrt NEU	PKW Einfahrt 2	Einfahrt rechts	Dell MEG1
Prüfbahn 4	Prüfbahn 5	Prüfbahn 6	Prüfbahn 7
Prüfbahn 8	Prüfbahn 9	Prüfbahn 10	Prüfbahn 11

Status

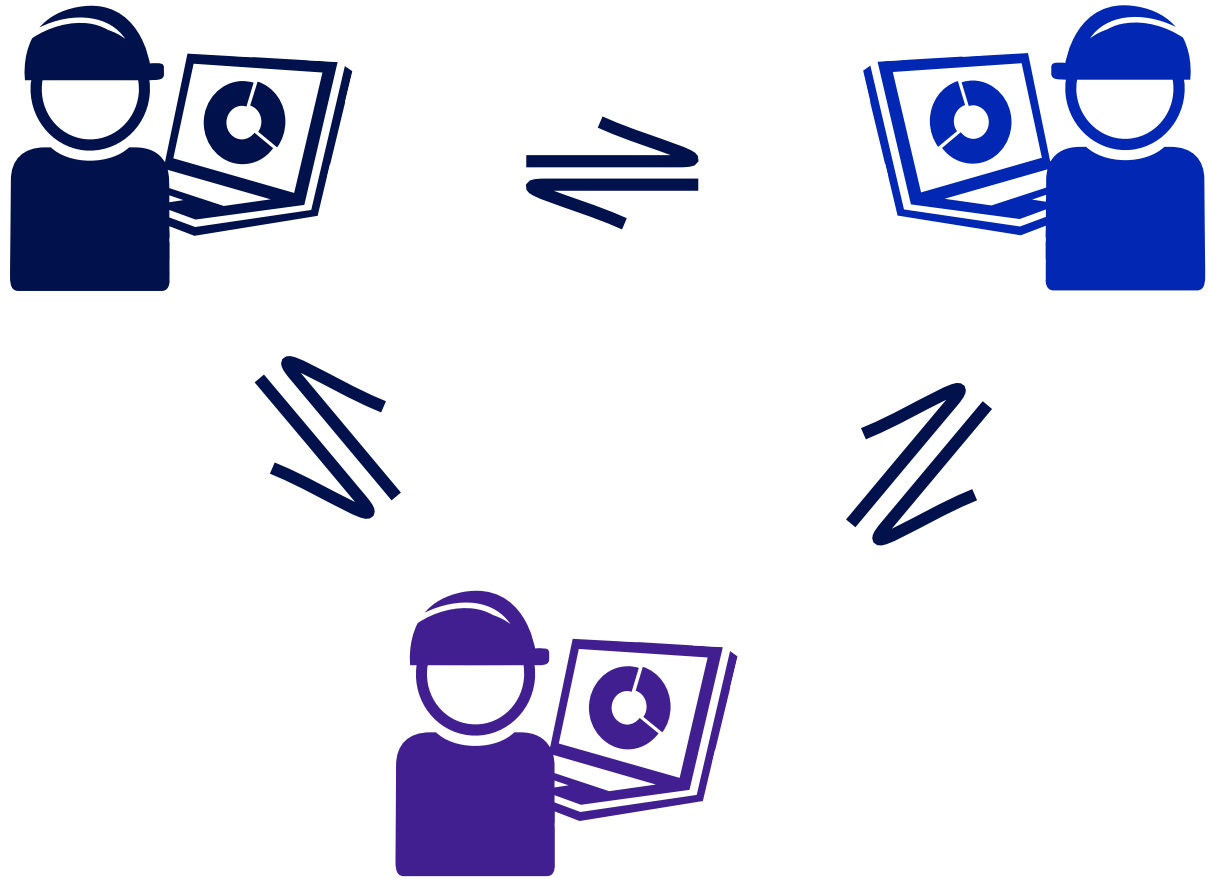
1.14.3 (Prod) E


Lokalisierung
Extras
Backup-Funktionen
Active-Group 241 (automatisch)
TSC
Koffer
Admin
Status
Administrationsmodus: Platz bearbeiten
Name
Einfahrt rechts
BEA-Konfiguration
Nicht konfiguriert
Aktuelle BEA-Konfiguration freigeben
Konfiguration nicht mehr verwenden
Konflikt lösen
AVL-Konfiguration
Nicht konfiguriert
Aktuelle AVL-Konfiguration freigeben
Konfiguration nicht mehr verwenden
Konflikt lösen
Poscard-Terminal-ID
Konfiguriert: 11111111
Aktuelle Poscard-Terminal-ID freigeben
Konfiguration nicht mehr verwenden
Konflikt lösen
Drucker-Konfiguration
Konfiguriert: KONICA MINOLTA Universal PCL (KMDRVSET 1.3)
Neuen Drucker wählen
Drucker nicht mehr verwenden
Konflikt lösen
Dockingstation-Konfiguration
Konfiguriert
Aktuelle Dockingstation freigeben
Konfiguration nicht mehr verwenden
Konflikt lösen
Abbrechen
Änderungen übernehmen
1.14.3 (Prod) E

Constraints

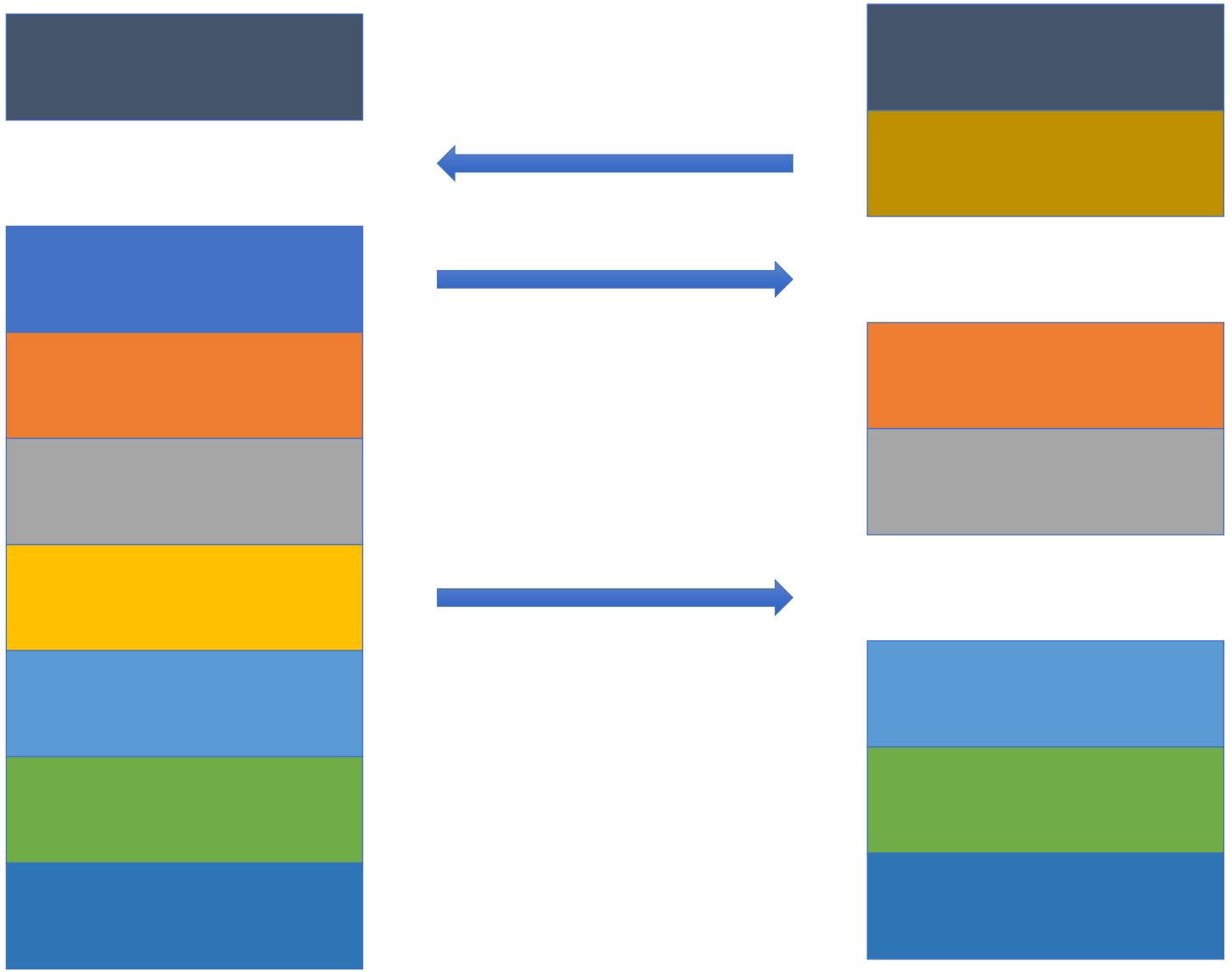
- unreliable network
- limited bandwidth between sites
- no server

Peer-to-Peer Synchronization

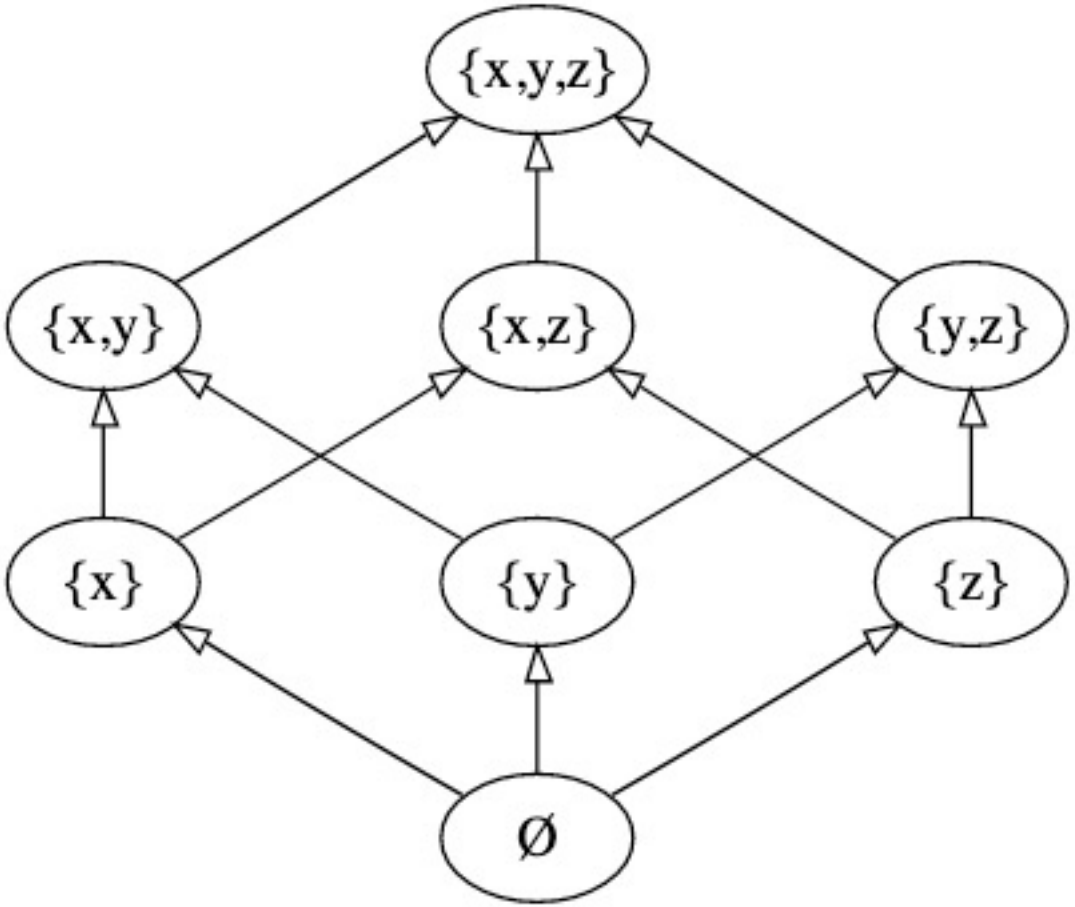


Pairwise Synchronization

Synchronize Facts



CRDT View

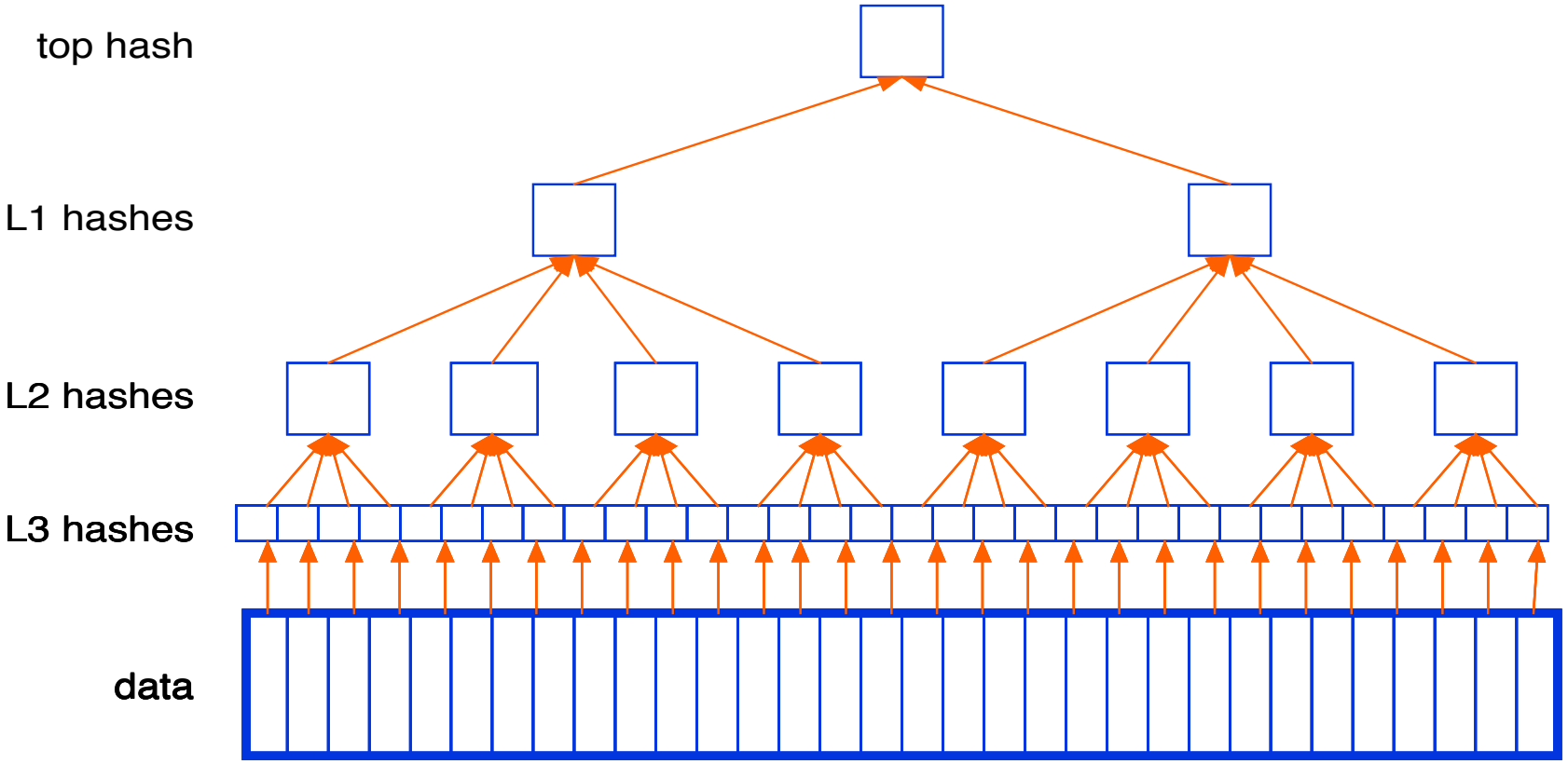


Source: Lars Hupel - An introduction to Conflict-Free Replicated Data Types https://lars.hupel.info/topics/crdt/03-lattices/

Interface

```
type SyncContext<'A> when 'A : equality = {
  /// sync mode
  mode: Protocol.ProtocolMode
  /// compute hash of payload
  ahash: 'A -> Hash.T
  /// pickler for payload
  pickler: Pickler<‘A>
  /// get the hashes of all payload blocks in the system
  getHashes: unit -> Async<list<Hash.T>>
  /// get the blocks with specified hashes
  getBlocks: seq<Hash.T> -> Async<list<Block<'A>>>
  /// save these blocks, which may already be there
  saveBlocks: list<Block<'A>> -> Async<unit>
}
```

Merkle Trees



Synchronization Step

Benefits

- audit log
- magic (“virus”)
- used as social network

Merkle Trees in F#

```
type MerkleTree =
    MerkleTree of SignaturePrefix *
                  list<Hash.T * Signature>

type Signature = Signature of list<uint64>

type SignaturePrefix = list<uint64>
```

Messages for Synchronization

```
type MerkleFingerprint =
    MerkleFingerprint of
      SignaturePrefix * Hash.T


type Msg =
| IHaveFingerprints of
        MerkleFingerprintSet
| HereAreBlockHashes of
        list<Hash.T>
```

Sync Step

```
let synchronizationRound
      (allHashes: Set<Hash.T>)
      (mts: MerkleTreeSet)
      (msg: Msg)
  : Choice<Msg * MerkleTreeSet,
           list<Hash.T>>
```

Local Sync

```
let synchronize (bs1: list<Block<'A>>)
                (bs2: list<Block<'A>>)
      : list<Block<'A>> * list<Block<'A>>
```

Testing Sync with Property-Based Testing

```
Prop.forAll
    (Arb.from<Set<Block<byte[]>> *
              Set<Block<byte[]>>>)
    (fun (bs1, bs2) ->
       let all = Set.union bs1 bs2
       let (bs1', bs2') = synchronize
                            (Set.toList bs1)
                            (Set.toList bs2)
       (Set.isEmpty (Set.intersect bs1 (Set.ofList bs1‘)) &&
       (Set.isEmpty (Set.intersect bs2 (Set.ofList bs2‘)) &&
       (Set.union bs1 (Set.ofList bs1') = all) &&
       (Set.union bs2 (Set.ofList bs2') = all)
    )|> AssertProperty
```

Bugs Since Deployment #1

0

Conflicts

on 2021-10-06, 14:00 UTC
Mr. Müller configured printer
at place #1 of TSC #15:

Acme LookNice 5020

Acme LookNice 5020

on 2021-10-06, 14:25 UTC
Ms. Maier configured printer
at place #1 of TSC #15:

Acme LookNice 5021

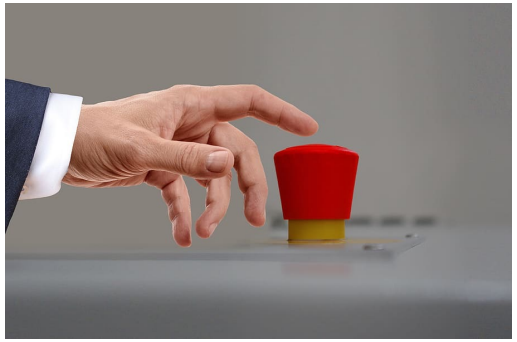
Acme LookNice 5021

Click on correct option

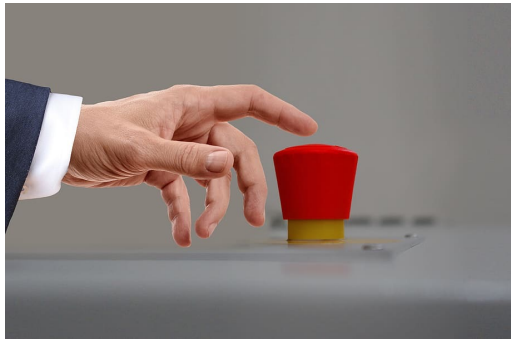
Conflict Representation

```
type OptionalVal<'a> when 'a : comparison =
| Absent
| Good of option<'a>
| Conflict of Set<WithMeta<option<'a>>>

module Meta =
  type T = { user:string
             machine:string
             datetime:DateTime.T}
```

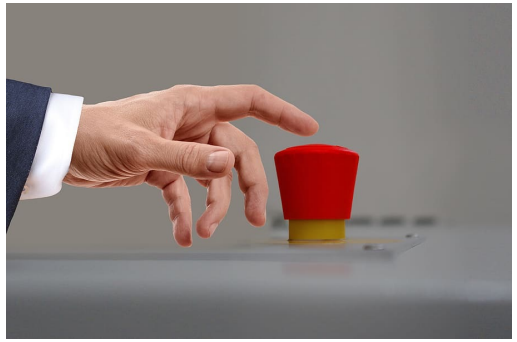
Manual Conflict Resolution



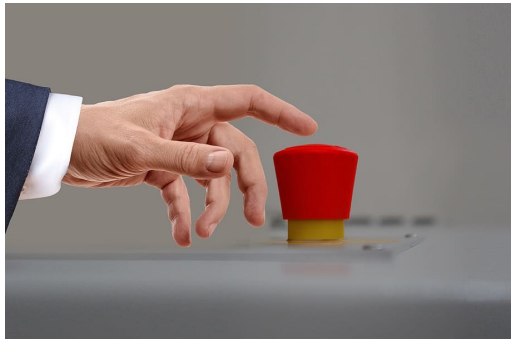
Mr. Müller



Ms. Maier



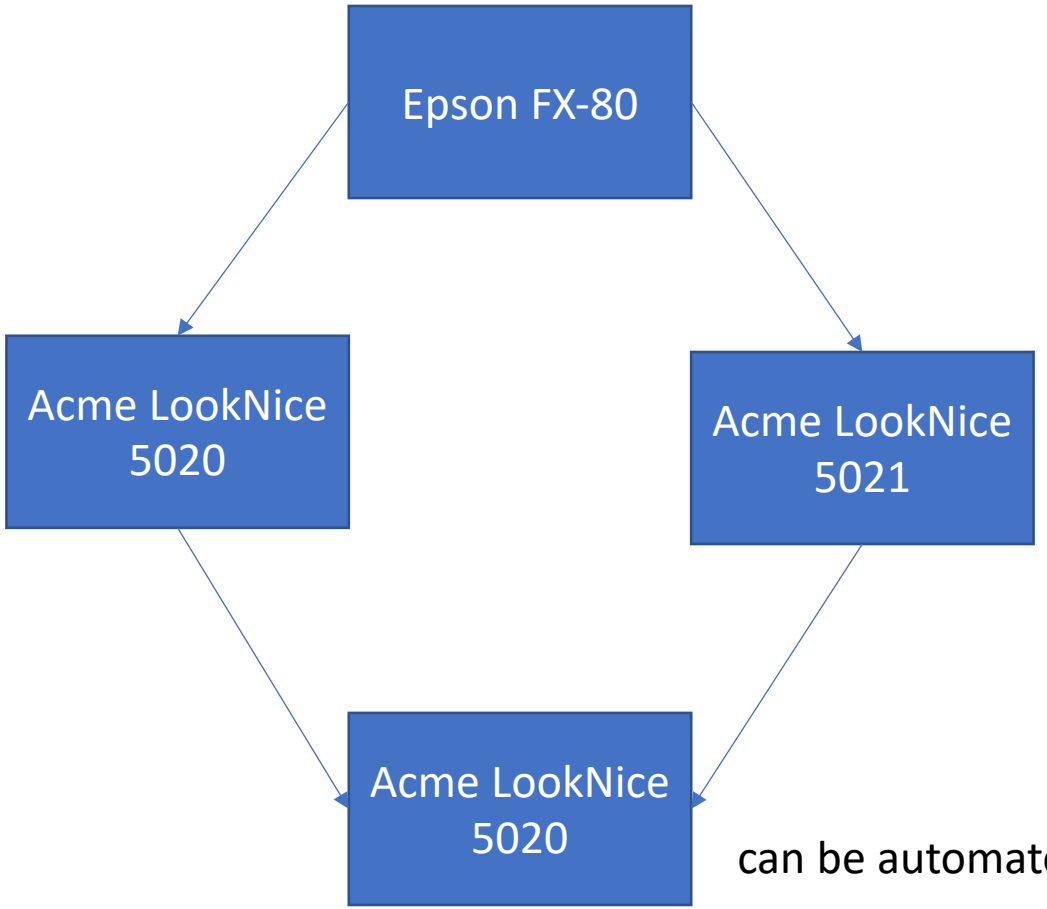
Mr. Jones



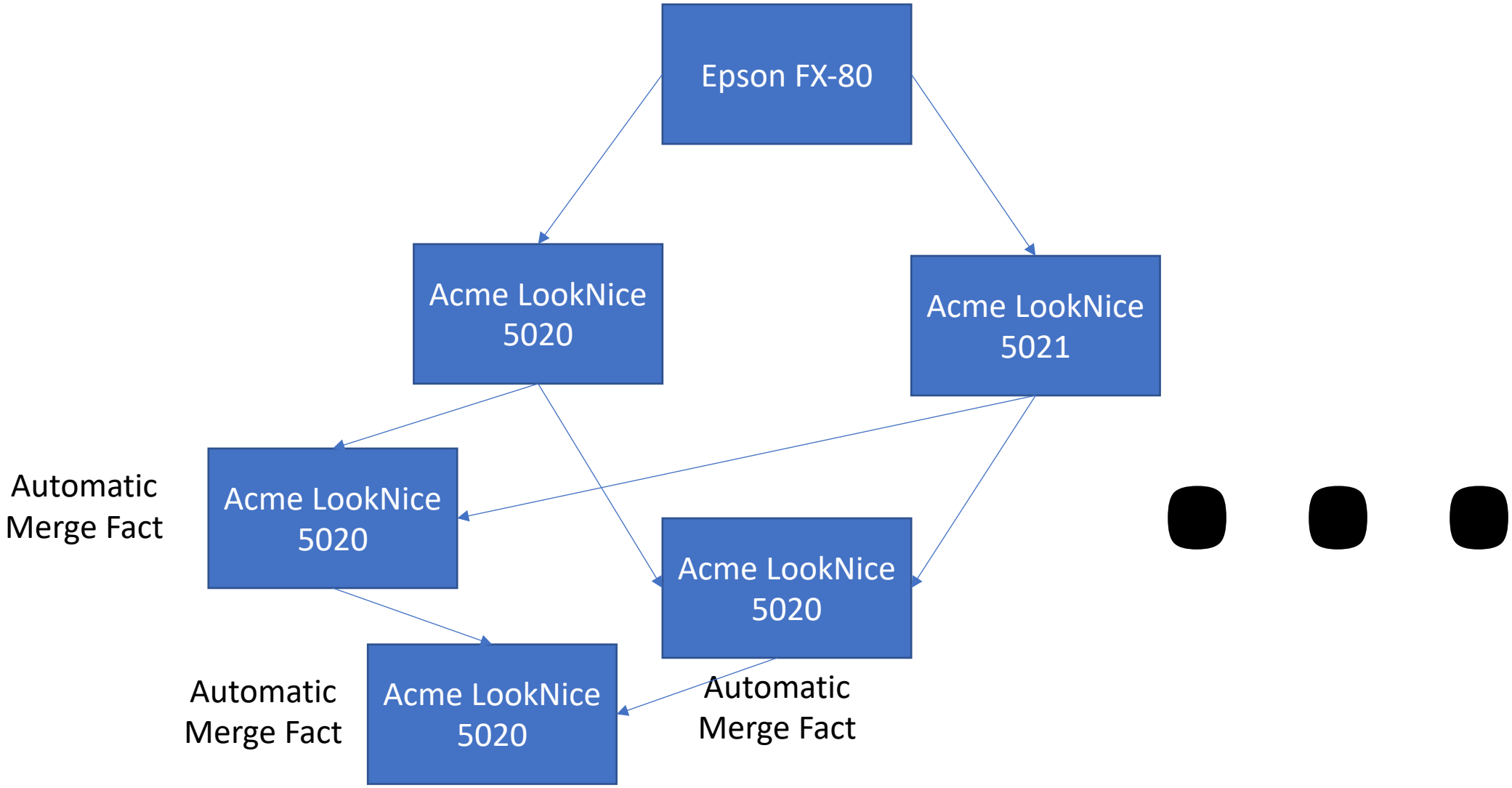
Ms. Smith

https://www.pxfuel.com/en/free-photo-oipjy

Causal Dependencies



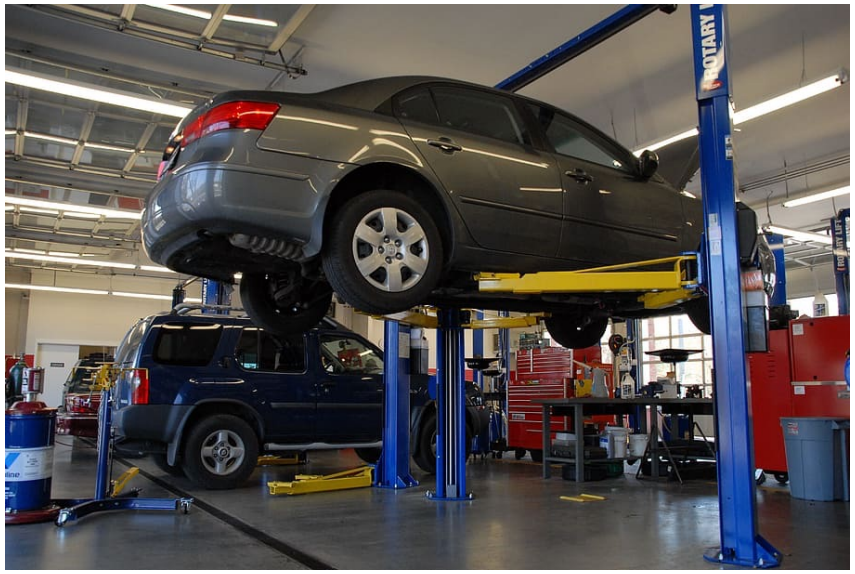
Automatic Dependency Resolution



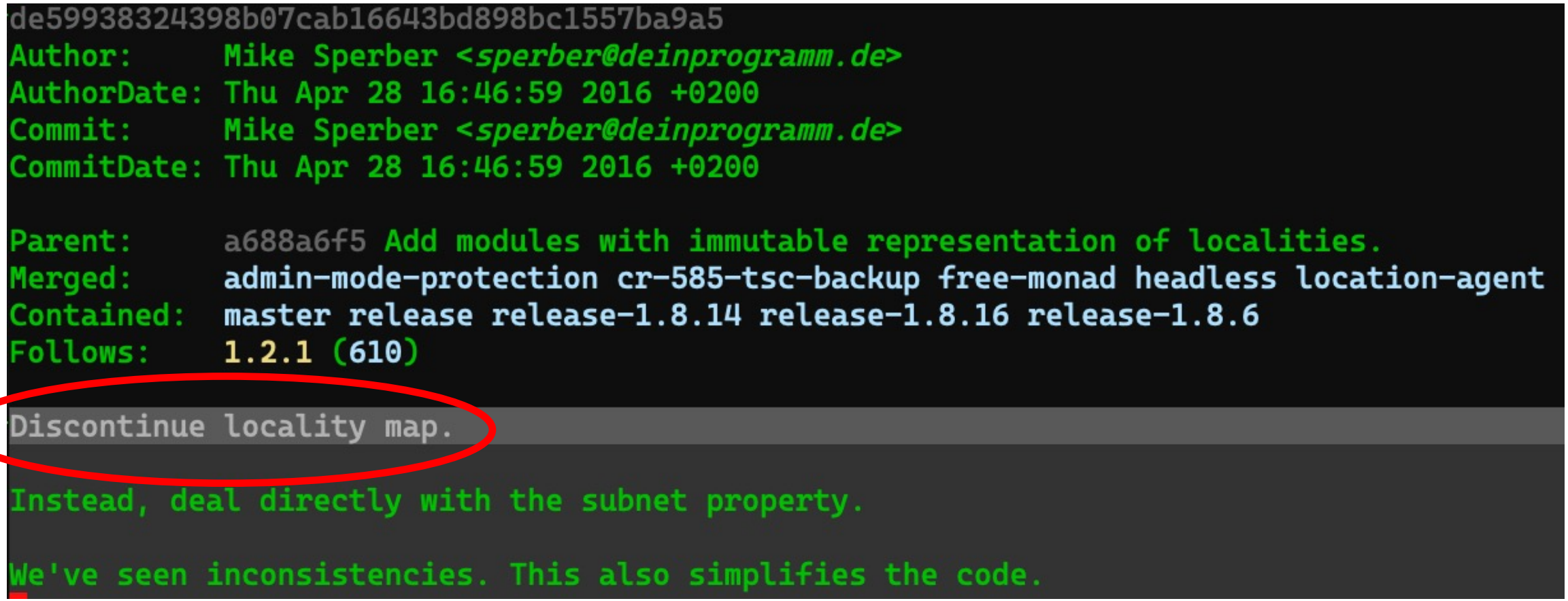
Conflict Resolution

- pick any of the current leaves
- **deterministically**
- create merge record on manual change

Locality Map



https://iconscout.com/icon/router-2691175 (IconScout, Kalash, CC BY 4.0)


de59938324398b07cab16643bd898bc1557ba9a5
Author: Mike Sperber <sperber@deinprogramm.de>
AuthorDate: Thu Apr 28 16:46:59 2016 +0200
Commit: Mike Sperber <sperber@deinprogramm.de>
CommitDate: Thu Apr 28 16:46:59 2016 +0200
Parent: a688a6f5 Add modules with immutable representation of localities.
Merged: admin-mode-protection cr-585-tsc-backup free-monad headless location-agent
Contained: master release release-1.8.14 release-1.8.16 release-1.8.6
Follows: 1.2.1 (610)
Discontinue locality map.
Instead, deal directly with the subnet property.
We've seen inconsistencies. This also simplifies the code.

DB Schema

```
CREATE TABLE hashes (
  key_hash BLOB NOT NULL,
  obsoleted_hash BLOB NOT NULL,
  FOREIGN KEY(key_hash) REFERENCES kv(hash)
);
CREATE INDEX hashesIdx1 ON hashes(obsoleted_hash);

CREATE TABLE kv (
  hash BLOB PRIMARY KEY NOT NULL UNIQUE, -- hash bytes
  guid BLOB NOT NULL, -- guid bytes
  property STRING NOT NULL, -- property name
  value BLOB NOT NULL, -- property value
  meta STRING NOT NULL -- meta JSON
);
CREATE INDEX kvIdx1 ON kv(guid,property);
```

Eliding Obsolete Entries

```
CREATE VIEW kvCurrent AS
SELECT hash,guid,property,value,meta FROM kv
WHERE kv.hash NOT IN
  (SELECT obsoleted_hash FROM hashes)
```

In-Memory Projection

```
module Locality =
  type GuidT = LocalityGuid.T

  type T = {
    guid: LocalityGuid.T
    displayName: string
    tscServer: OptionalVal<TscServer.T>
    subnet: OptionalVal<Subnet.T>
    places: list<Place.T>
  }
```

DB model vs. Data Model

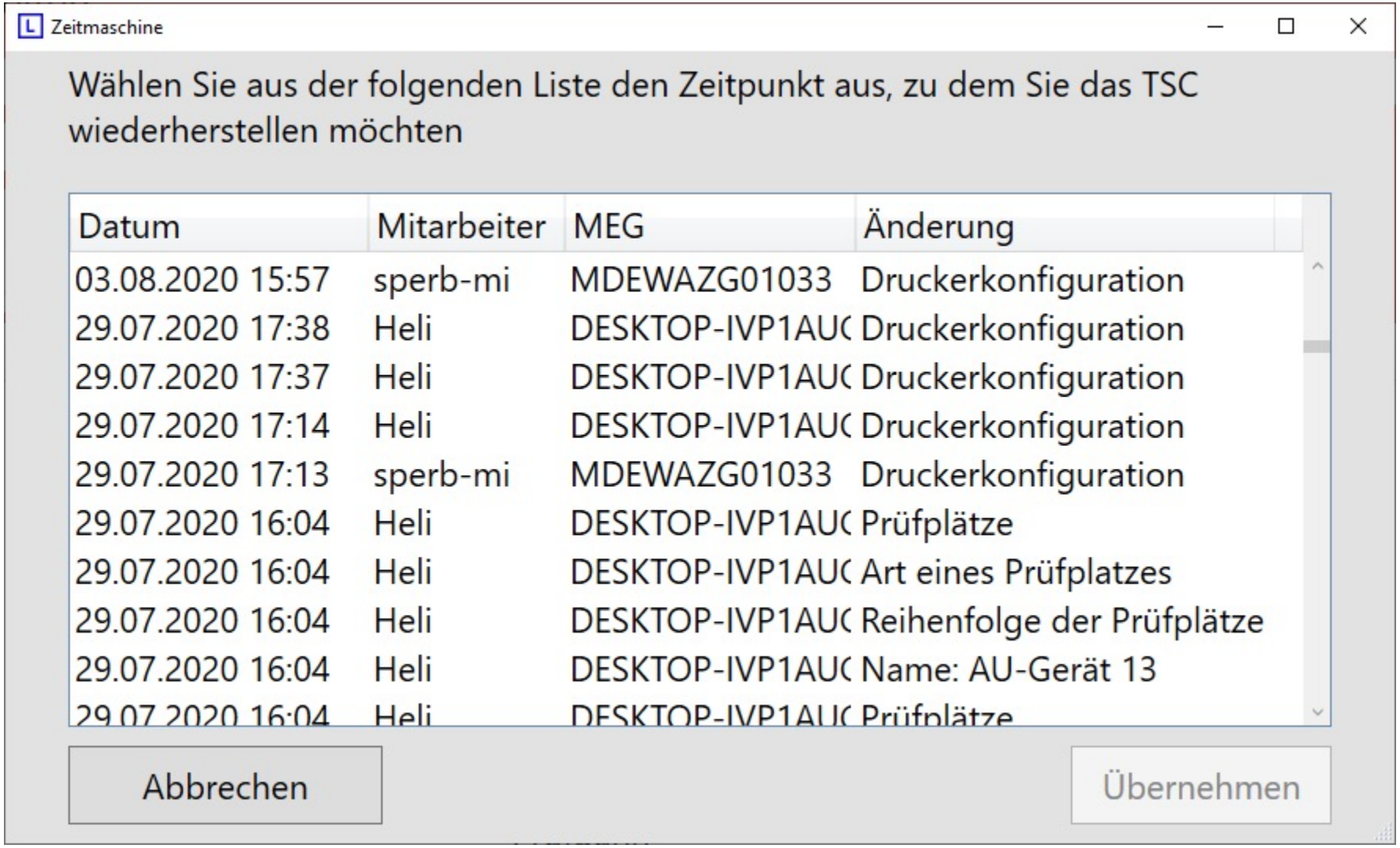
```
let readLocality getMobilePrinters luid =
  db {
    let! oldDisplayName = localityDisplayName luid
    let! oldTscServer = localityTscServerWithMeta luid
    let! oldMaybeSubnet = localitySubnetWithMeta luid
    let! oldPlacesGuids = getLocalityPlacesOp luid

    let! oldPlaces = oldPlacesGuids
                     |> mapM (readPlace getMobilePrinters)
    let locality: Active.TuevPlaces.Locality.T = { ... }
    return locality
  } |> atomically
```

Handling Change

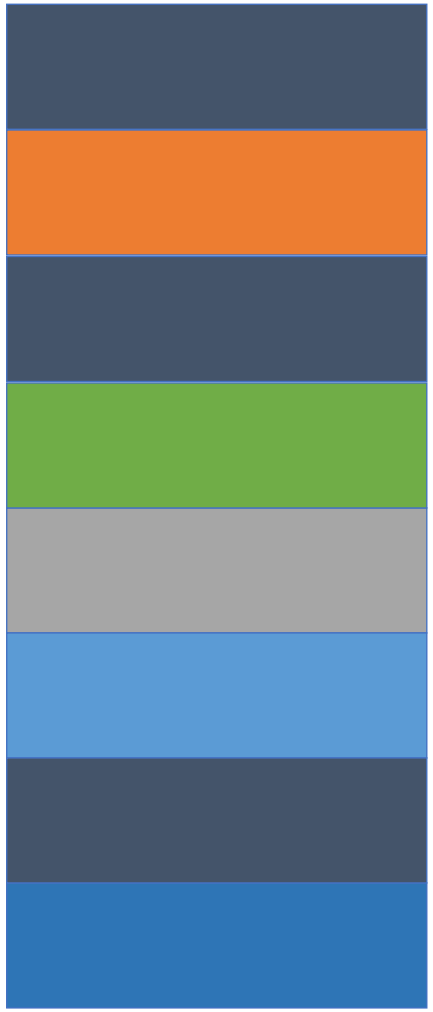
```
let saveEditOps
 (oldLocality: Active.TuevPlaces.Locality.T)
 (newLocality: Active.TuevPlaces.Locality.T)
  : (seq<LocalityChangeDescription> * unit Op) =
    ...
```

Works also for Time Machine!

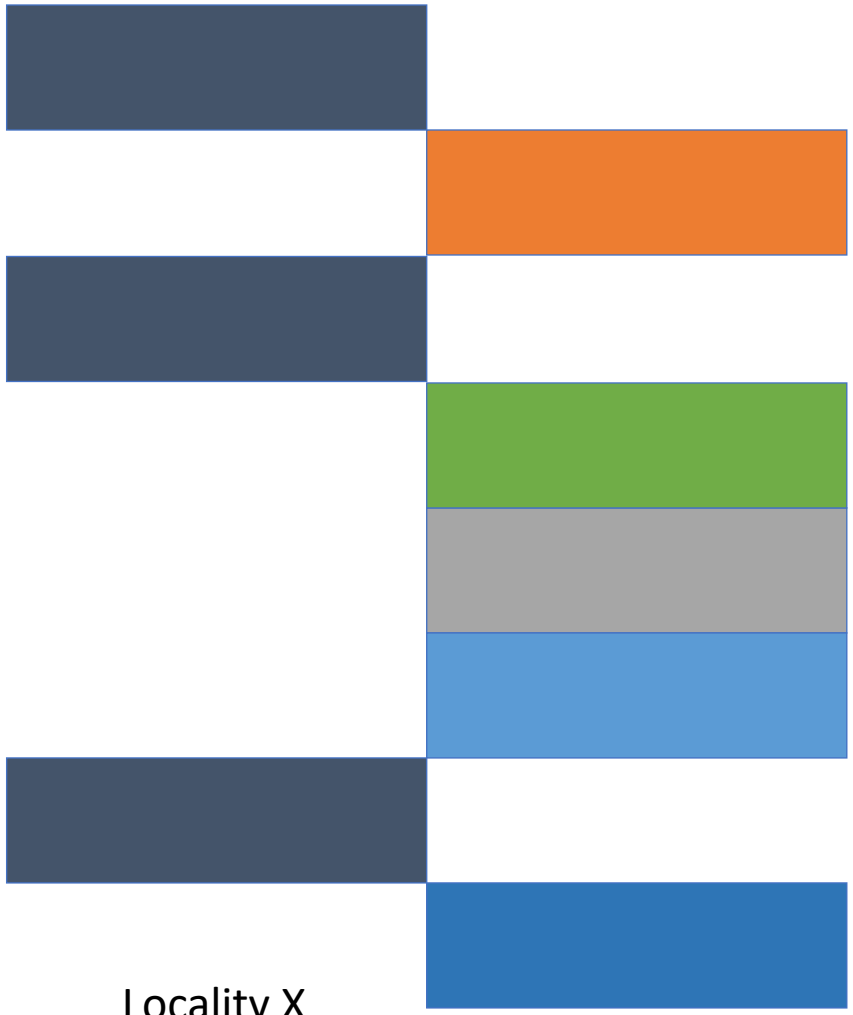
Time Machine



Time Machine

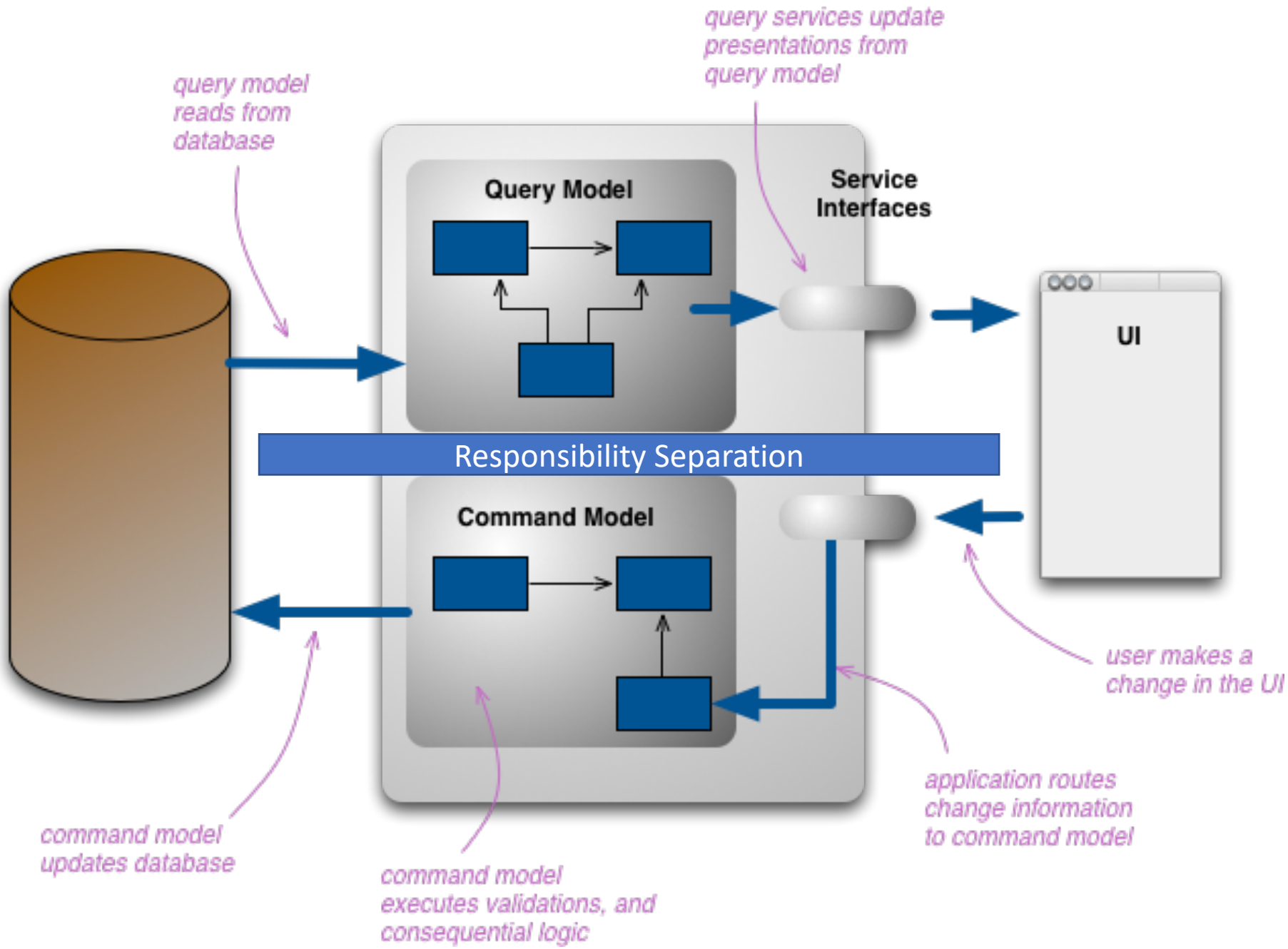


Time Machine



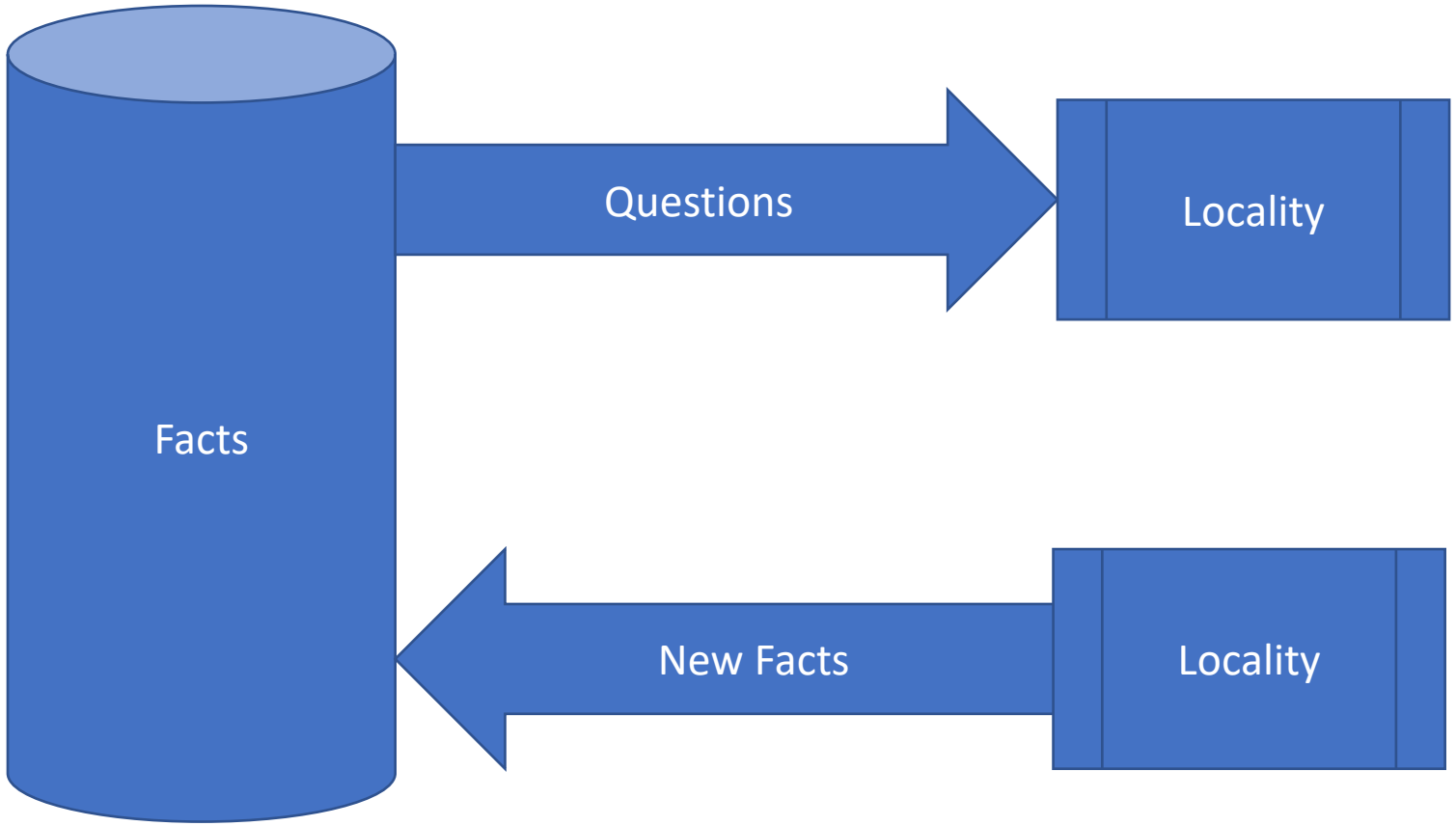
Locality X

CQRS

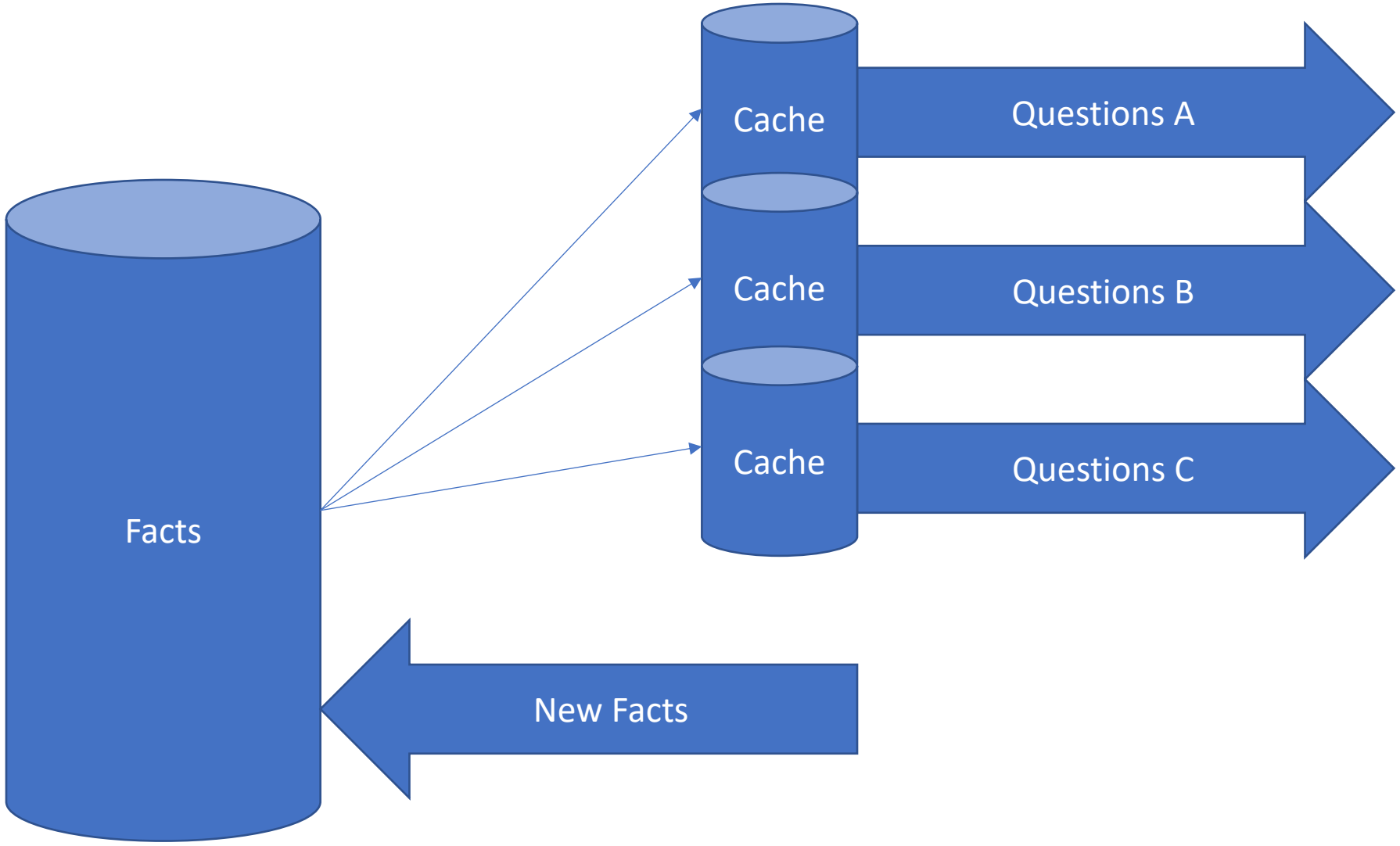


Source: Martin Fowler – CQRS https://martinfowler.com/bliki/CQRS.html

Us



Us


Cache
Cache
Cache
Questions A
Questions B
Questions C
Facts
New Facts

Conclusions

- store facts, not state
- avoid projections / read model through explicit dependencies between facts and indexing
- separate data model from DB model
- actually make that time machine
- think about questions you want answered and caches rather than CQRS
- decouple all the things