Just-Right Consistency

Living on the edge, safely

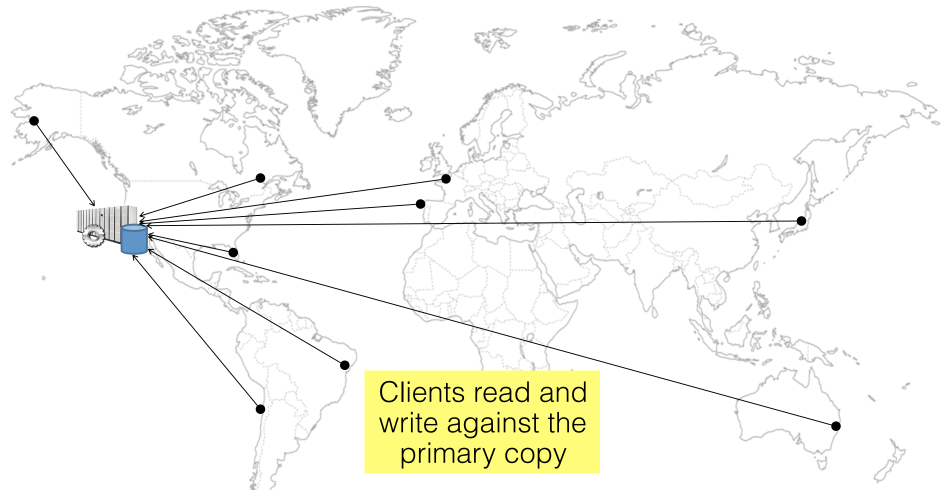
Marc Shapiro, Sorbonne-Université—LIP6 & Inria
Annette Bieniusa, U. Kaiserslautern
Nuno Preguiça, U. Nova Lisboa
Christopher Meiklejohn, U. Catholique de Louvain
Valter Balegas, U. Nova Lisboa

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

Inria informatics mathematics

I. Geo-distribution and consistency
II. Consistency — just right for your application
III. Antidote, a database for JRC
IV. WIP: Antidote towards the edge

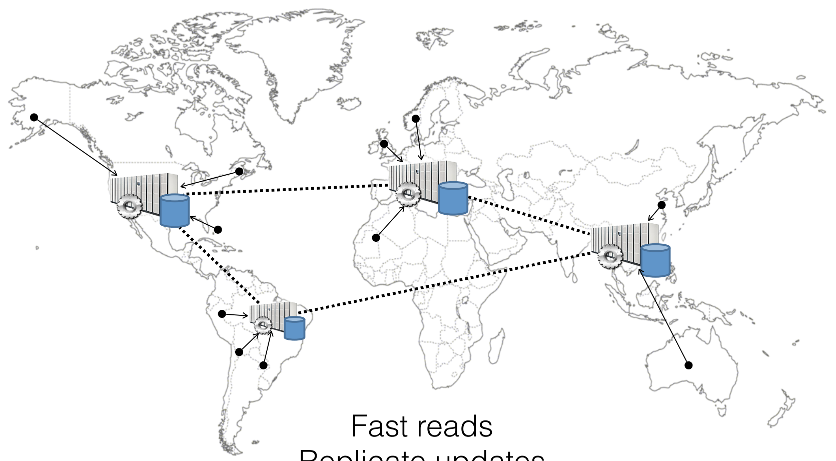
Sequential data access



Clients read and write against the primary copy

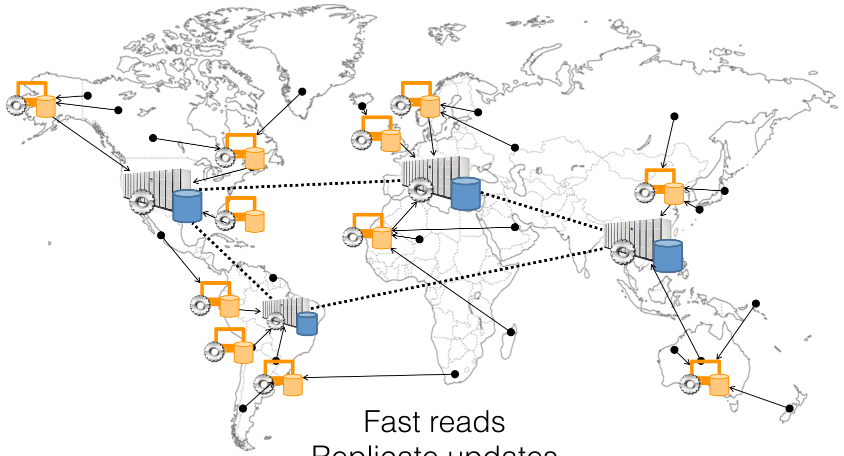
[http://vignette3.wikia.nocookie.net/the-titans-rp-and-information/images/f/f5/Blank-World-map2.gif/revision/latest/scale-to-width-down/1280?cb=20141016203452]

Cloud



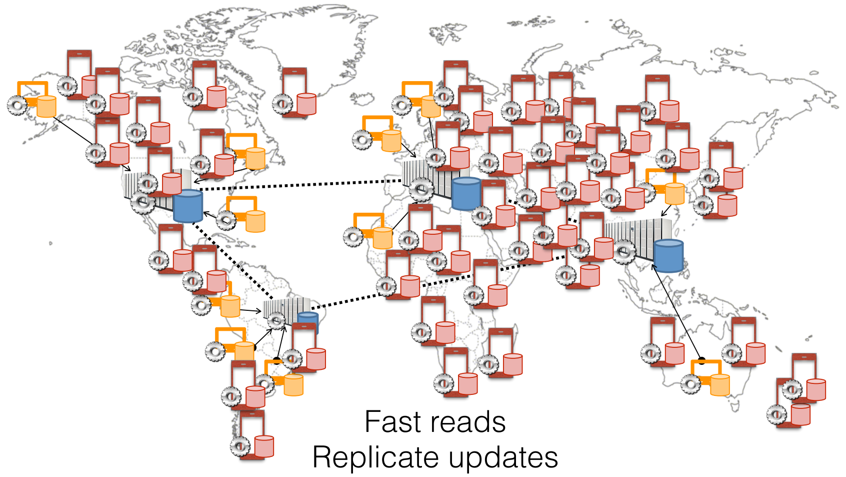
Fast reads
Replicate updates
consistent ∩ available = ∅

Fog



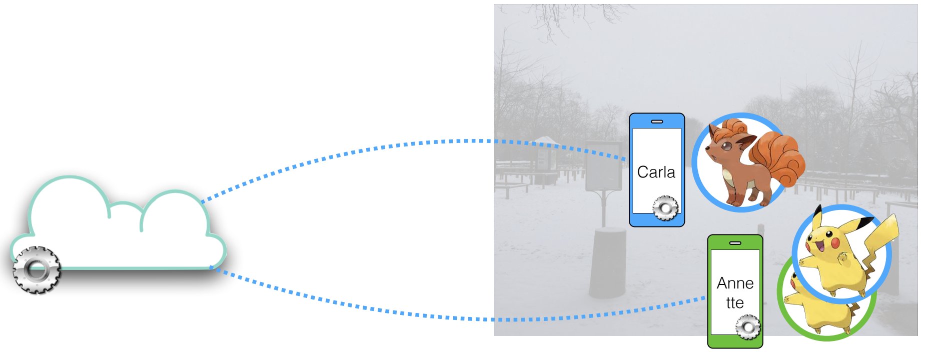
Fast reads
Replicate updates
consistent ∩ available = ∅

Far edge



Fast reads
Replicate updates
consistent ∩ available = ∅

Consistency: Pokémon Go

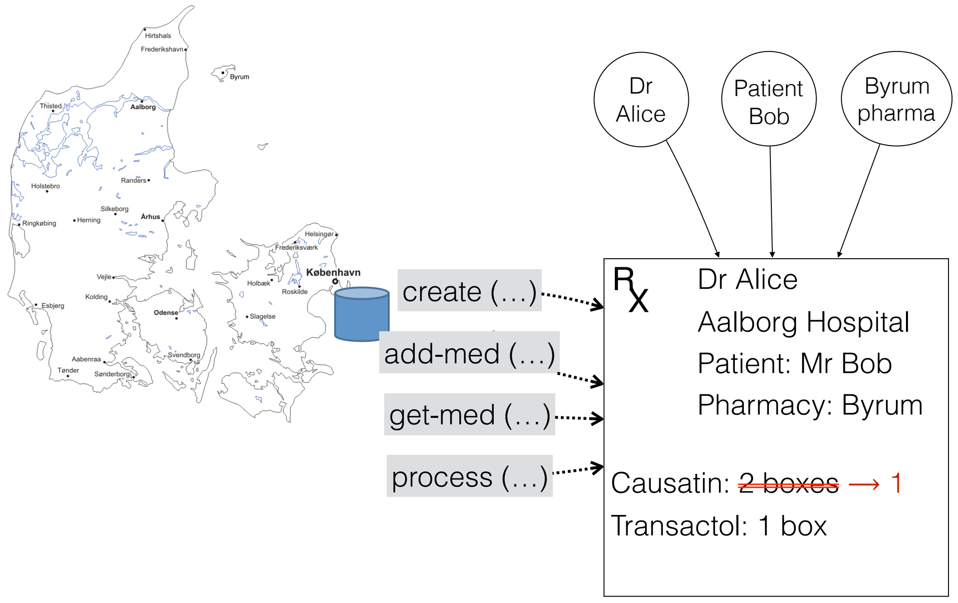


Data at the edge, cloud-centric

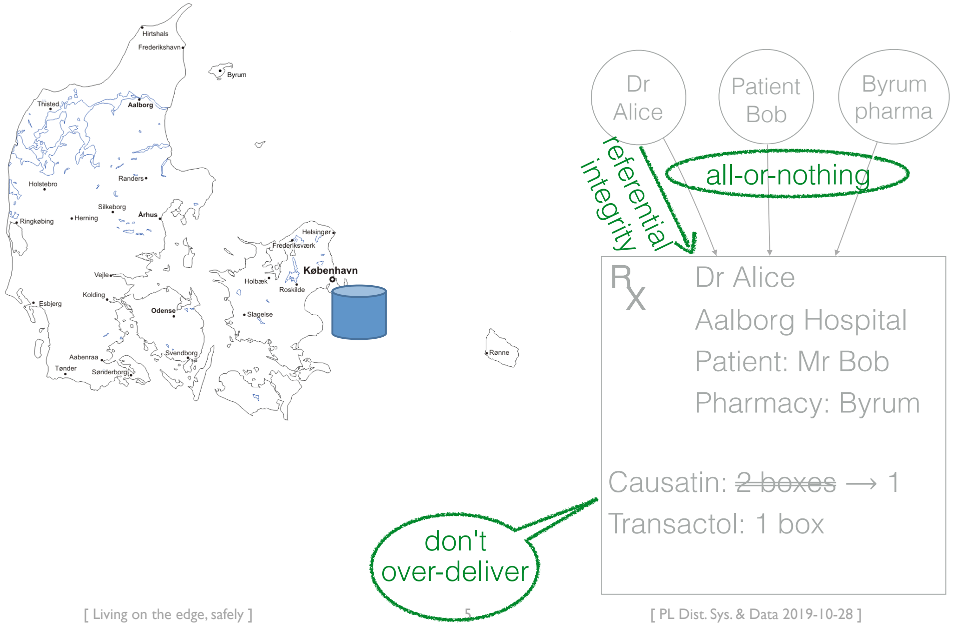
✔availability, ✗latency, ✗bandwidth

- poor interaction
- anomalies

FMK *Fælles Medicinkort*



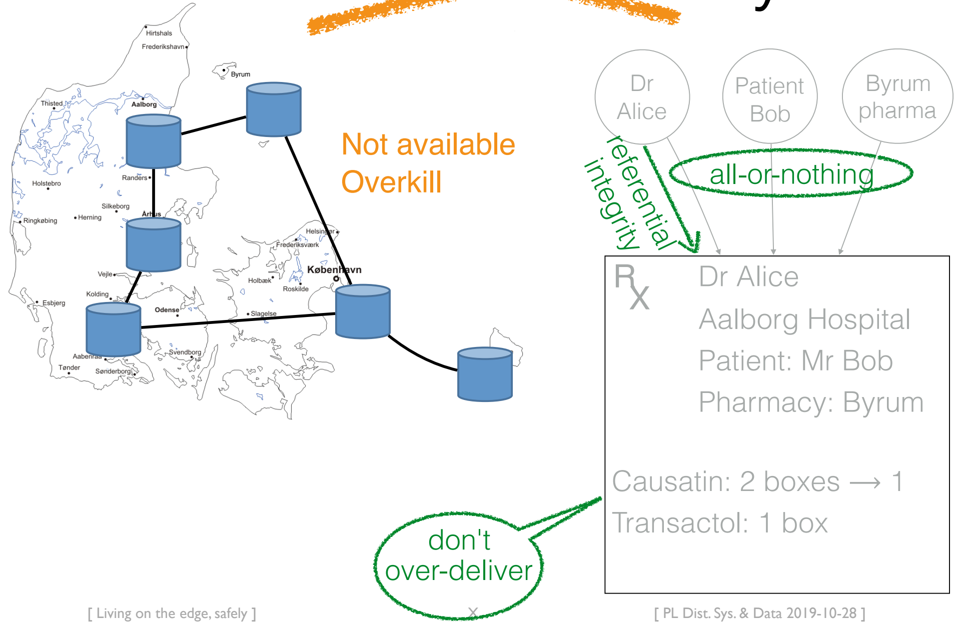
Correctness invariants



R_X Dr Alice
Aalborg Hospital
Patient: Mr Bob
Pharmacy: Byrum

Causatin: ~~2 boxes~~ ⟶ 1
Transactol: 1 box

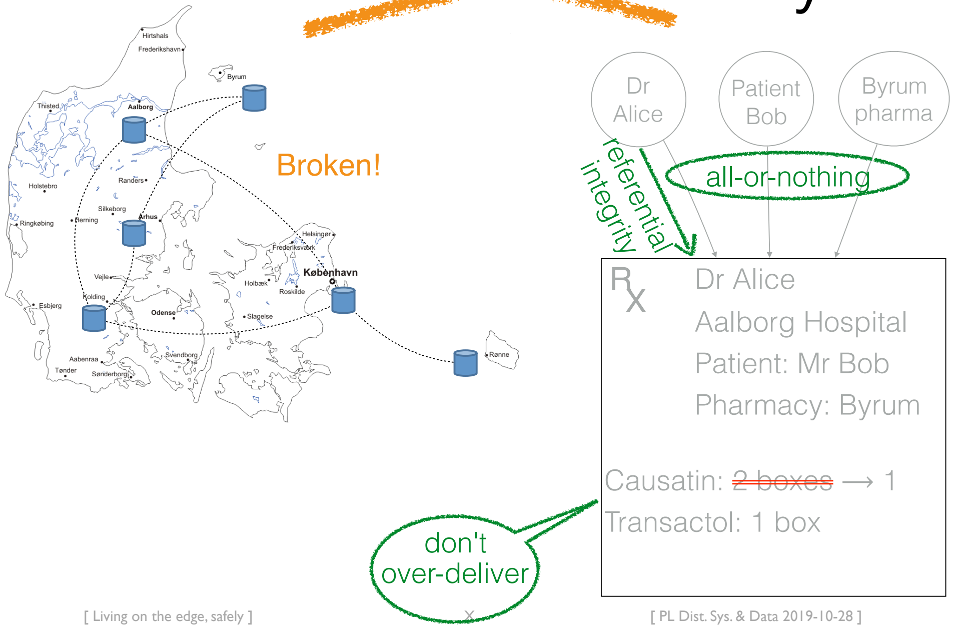
~~Strict serialisability~~



R_X Dr Alice
Aalborg Hospital
Patient: Mr Bob
Pharmacy: Byrum

Causatin: 2 boxes ⟶ 1
Transactol: 1 box

~~Eventual consistency~~



R_X Dr Alice
Aalborg Hospital
Patient: Mr Bob
Pharmacy: Byrum

Causatin: ~~2 boxes~~ ⟶ 1
Transactol: 1 box

What is the right consistency model?

No “one-size-fits-all” consistency model
- All-CP: over-conservative
- All-AP: risk anomalies

Consistency options?
- Hard to choose the right one
- What happens when switching?

Just-Right Consistency

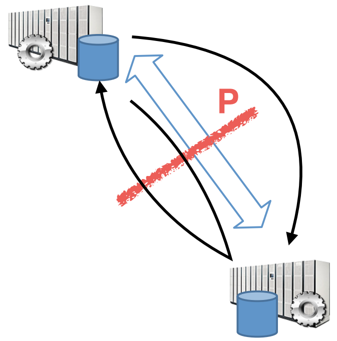
Sequential version is correct

Tailor consistency to *application invariants*

As available as possible & as consistent as necessary

- Asynchronous by default
- Synchronise (only) when required by application invariants
- Co-design application & protocol
- Correct by construction

Approach



Consistency: one size does *not* fit all

Correct: maintain invariants

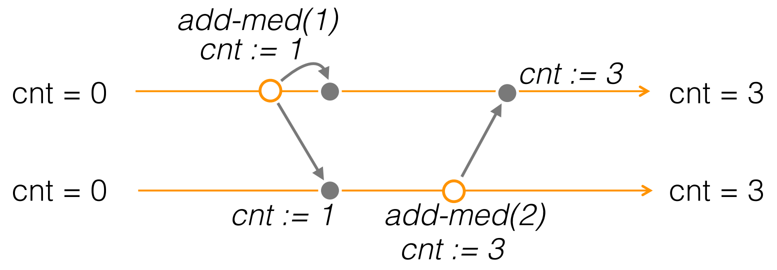
- But often unknown!

Methodology:

- Preserve sequential *patterns*
- Synchronise only when *strictly necessary* for application

best possible availability and performance

assign + CP



Concurrent, asynchronous updates

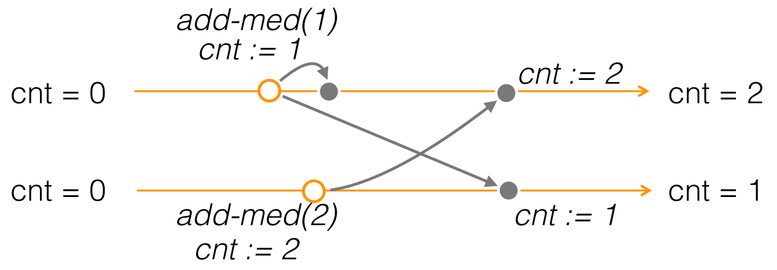
- Standard register model: assignments ⟹ CP
- AP ⟹ concurrent updates + merge

CRDT: register, counter, set, map, sequence

- Plug-in replacement for sequential type

Rx.Patient: write_once_reg. Rx.Meds: set

Concurrent assignment



Concurrent, asynchronous updates

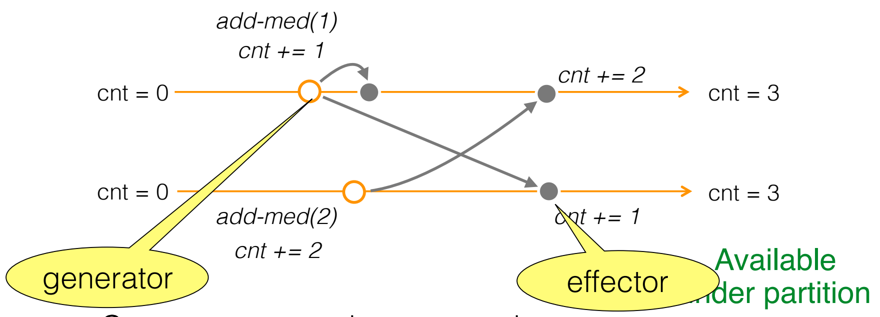
- Standard register model: assignments ⟹ CP
- AP ⟹ concurrent updates + merge

CRDT: register, counter, set, map, sequence

- Plug-in replacement for sequential type

Rx.Patient: write_once_reg. Rx.Meds: set

Conflict-free Replicated Data Type



Concurrent, asynchronous updates

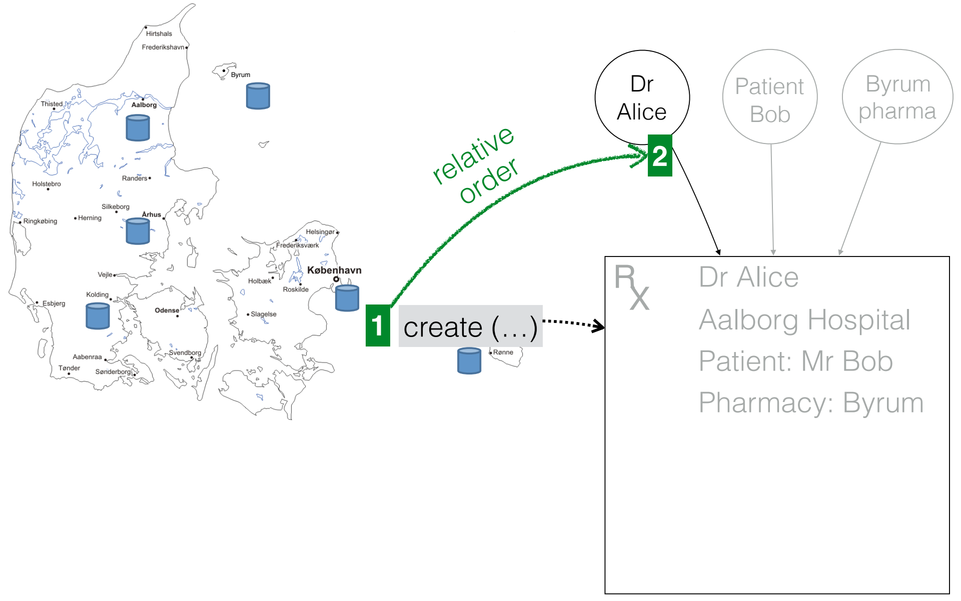
- Standard register model: assignments ⟹ CP
- AP ⟹ concurrent updates + merge

CRDT: register, counter, set, map, sequence

- Plug-in replacement for sequential type

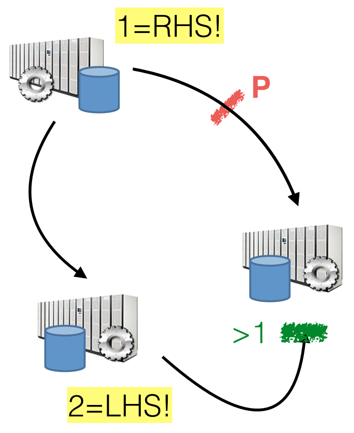
Rx.Patient: write_once_reg. Rx.Meds: set

Ordered-item invariant



without CC animation

Causal consistency



create-p before *add-med*

- "Bob points to Rx ⟹ Rx valid"
- General case: LHS ⟹ RHS
- pattern: RHS!; LHS!

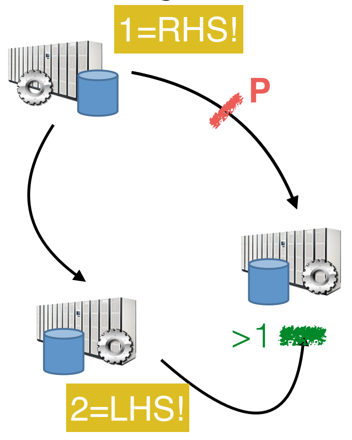
Deliver in the right order: Causal Consistency

AP-compatible

with CC animation

Causal consistency

Available under partition



create-p before *add-med*

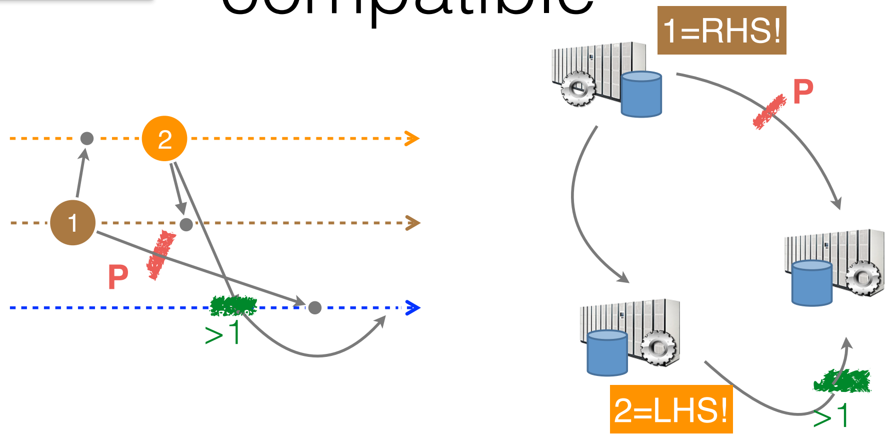
- "Bob points to Rx ⟹ Rx valid"
- General case: LHS ⟹ RHS
- pattern: RHS!; LHS!

Deliver in the right order: Causal Consistency

AP compatible

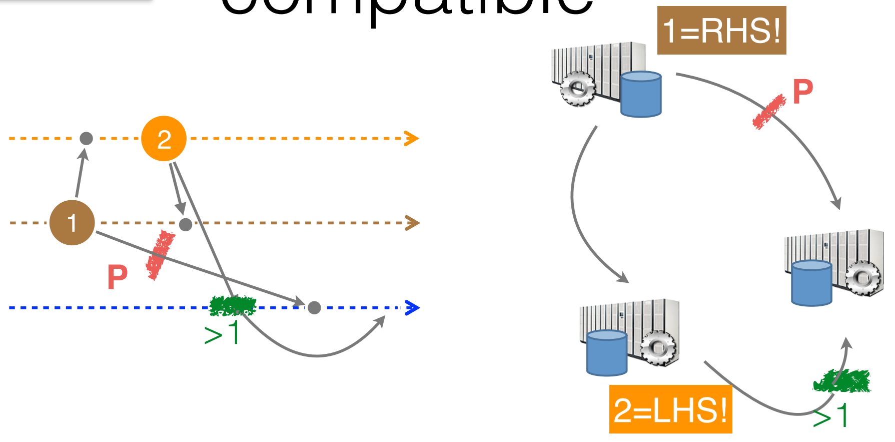
with animations

al consistency is AP-compatible

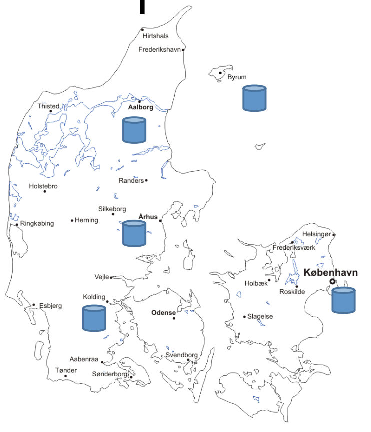


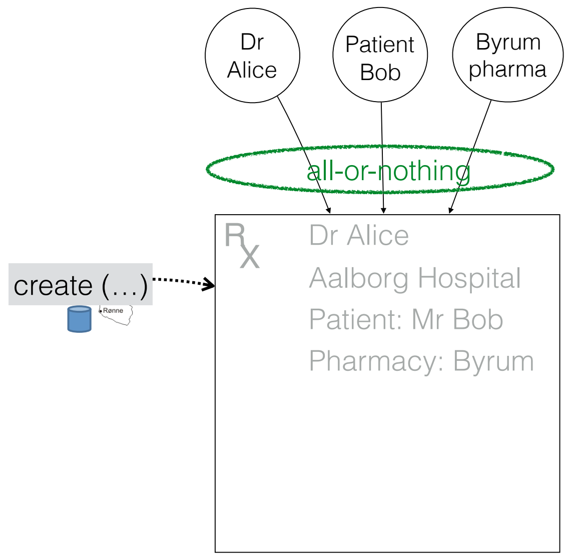
no
animations

al consistency is AP-compatible



Equivalent-item invariant





with animations

ll-or-nothing bundle

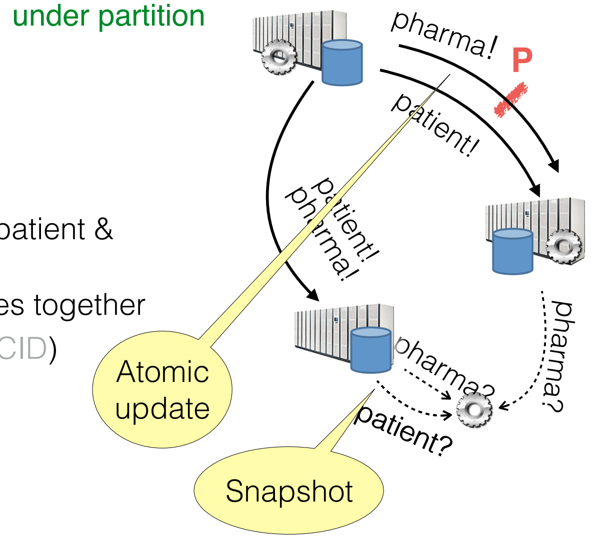
Available
under partition

create-p updates doctor, patient & pharmacy record

Transmit, read joint updates together

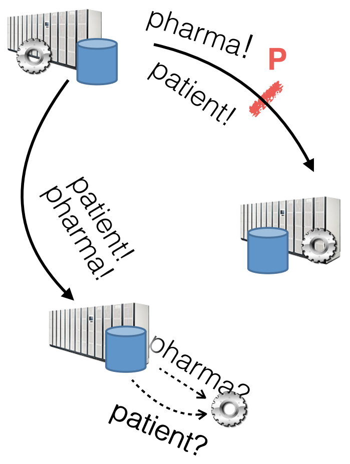
= All-or-Nothing (A of ACID)

AP-compatible



no animations, simplified

All-or-nothing



create-p updates doctor, patient & pharmacy record

Transmit, read joint updates together

= All-or-Nothing (A of ACID)

AP-compatible

Txnl Causal Consistency

Transactional Causal Consistency (TCC) = strongest AP model

Guarantees AP-compatible invariant patterns

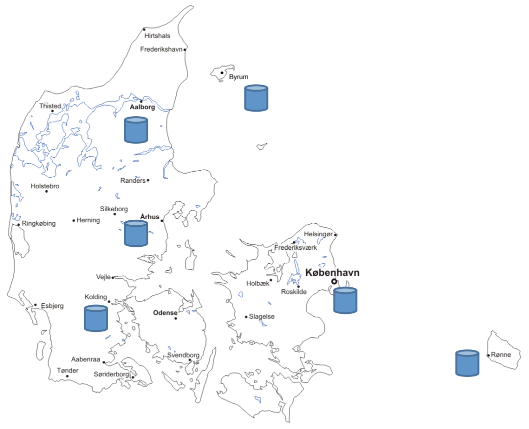
Antidote: first industrial-strength TCC data store

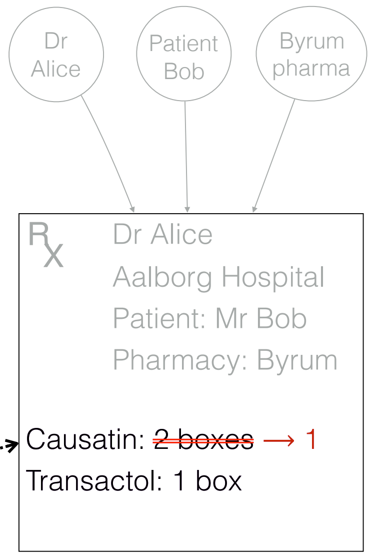
- alpha

Guarantees Relative-Order and Joint-Update invariant patterns

AntidoteDB

Item-value invariant





Dr Alice
Aalborg Hospital
Patient: Mr Bob
Pharmacy: Byrum

Causatin: ~~2 boxes~~ ⟶ 1
Transactol: 1 box

process (…)

precondition-modify

CISE result

Sequential ⟶ concurrent

Assume:

- Causal consistency, atomic effectors
- Initial state satisfies invariant
- u and v maintain the invariant in isolation
  ($\Longrightarrow u; v$ maintains invariant)

Safe sequential code

What invariant?

If

- u_{eff}, v_{eff} commute
- u_{pre} stable under v_{eff}, v_{pre} stable under u_{eff}

CRDTs

Then $u \mid\mid v$ maintains the invariant

- Otherwise, disallow $u \mid\mid v$

Synchronise only when necessary

Precondition-modify

Precondition

Invariant

```
assert (remaining ≥ 0):
If (remaining ≥ requested)
    remaining –= requested;
assert (remaining ≥ 0);
```

✔

```
assert (remaining ≥ 0):
remaining += 100;
assert (remaining ≥ 0);
```

May precondition be negated by concurrent update?

If precondition of *u* is stable under *v*, and vice-versa:
  then *u* || *v* OK.

otherwise
- weaken invariant, Available under partition
- or synchronisation required.

Precondition-modify

```
assert (remaining ≥ 0):
If (remaining ≥ requested)
    remaining –= requested;
assert (remaining ≥ 0);
```

```
assert (remaining ≥ 0): ✘
If (remaining ≥ requested)
    remaining –= requested;
assert (remaining ≥ 0);
```

May precondition be negated by concurrent update?

If precondition of *u* is stable under *v*, and vice-versa:
  then *u* || *v* OK. Available under partition

otherwise
- weaken invariant,
- or synchronisation required.

Sequential ⟶ concurrent

Assume
- Causal consistency, atomic effectors
- Sequentially-correct program

Transformations:
- Replace sequential data types with CRDTs
  - If not possible, synchronise access
- Verify precondition stability
- If not stable
  - either weaken invariant
  - or synchronise

Just-Right Consistency

Tailor consistency to application invariants
- (possibly unknown)

Three types of invariants:
- Ordered updates ⟹ Causal, AP
- Joint updates ⟹ Bundled, AP
- CAP-sensitive: precondition-modify
  - Mutually stable ⟹ concurrent OK. AP.
  - Otherwise, concurrency control. CP

Baseline: Correct app under strong consistency
- Identify, maintain programming patterns

Just-Right Consistency

Methodology for provably ensuring
As Available as Possible, Consistent Enough

TCC ⟹ AP-compatible invariants

CAP-sensitive invariants: Bounded Ctr, CISE

AntidoteDB

⟹ AP-compatible

- CRDTs
- Causal Consistency
- Transactions
- Bounded Counter

mostly AP

CP when necessary

CISE verification
& co-design

(+ related tools)

Open issues

Leap of faith: invariants ⟶ patterns; three patterns.

Scale to the edge: causal consistency expensive

Transaction semantics

- atomicity = easy
- cost of snapshot reads

AntidoteDB

CRDT data model

- Register, counter, set, map, sequence
- Extends sequential semantics

Transactional Causal Consistency Plus (TCC+)

- ≜ Relative Order + Bundles + CRDTs
- Strongest AP model

CISE: verify precondition stability

Open source, well engineered, growing community

LIGHTKONE
Lightweight computation for networks at the edge
European Commission
RainbowFS
ANR