Just-Right Consistency

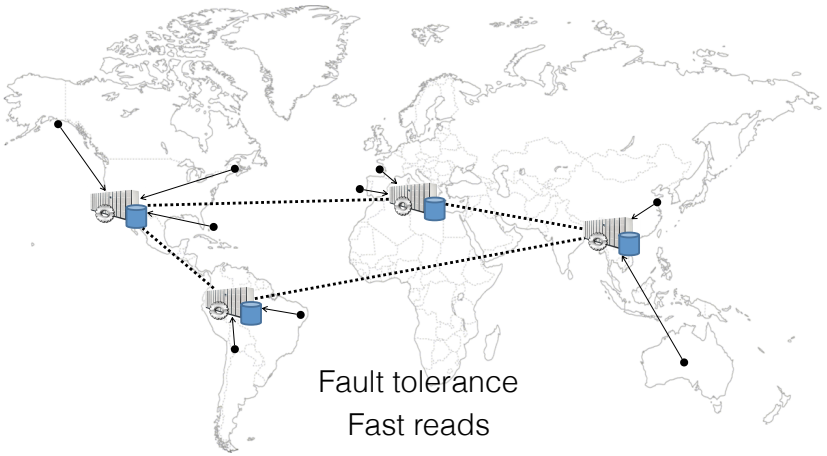
As available as possible
As consistent as necessary
Correct by design

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

informatics mathematics Inria

Geo-distributed DB

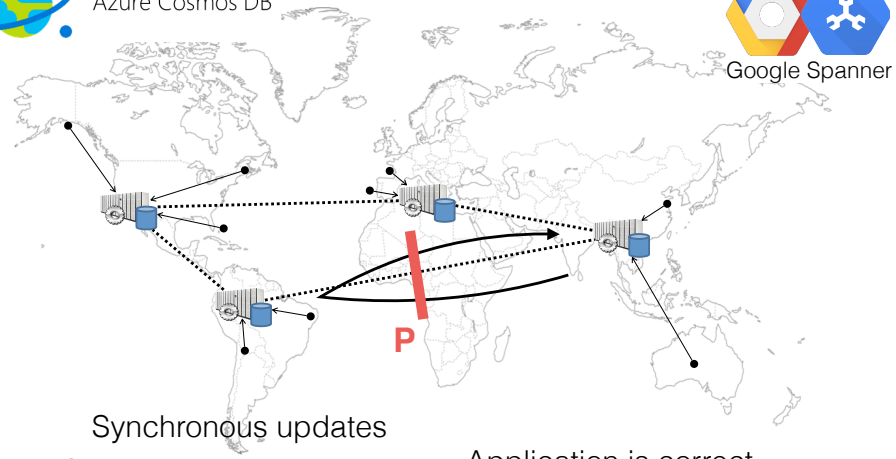


Fault tolerance
Fast reads
Replicate updates
CAP: **CP ∩ AP = ∅**

Strong Consistency

Azure Cosmos DB

Google Spanner



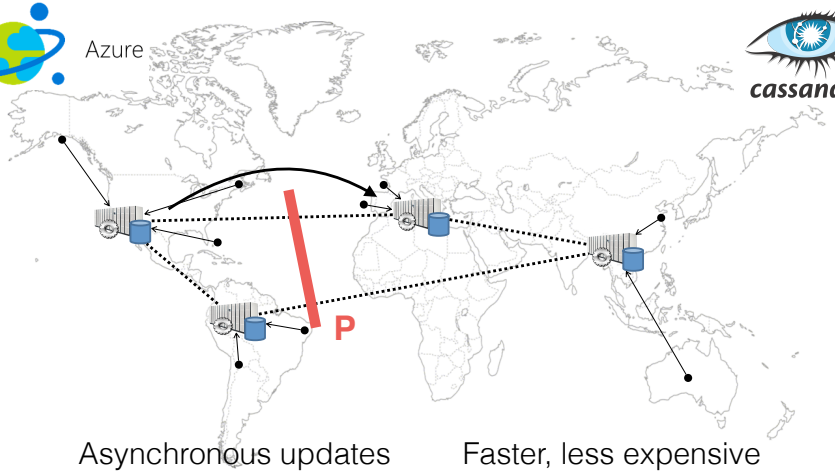
Synchronous updates
Consistent under Partition; not Available

Application is correct
Slow, expensive

Eventual Consistency

Azure

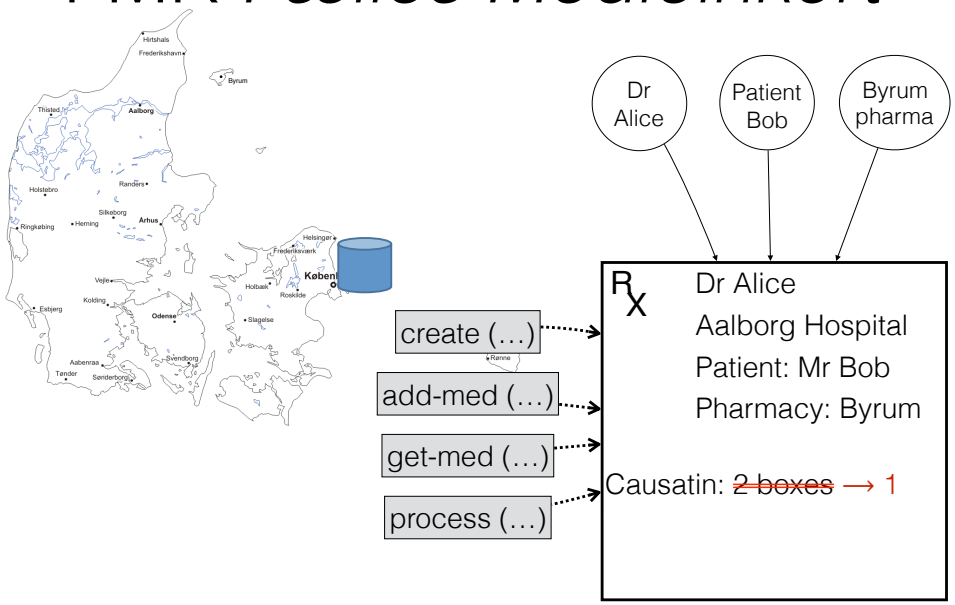
cassandra



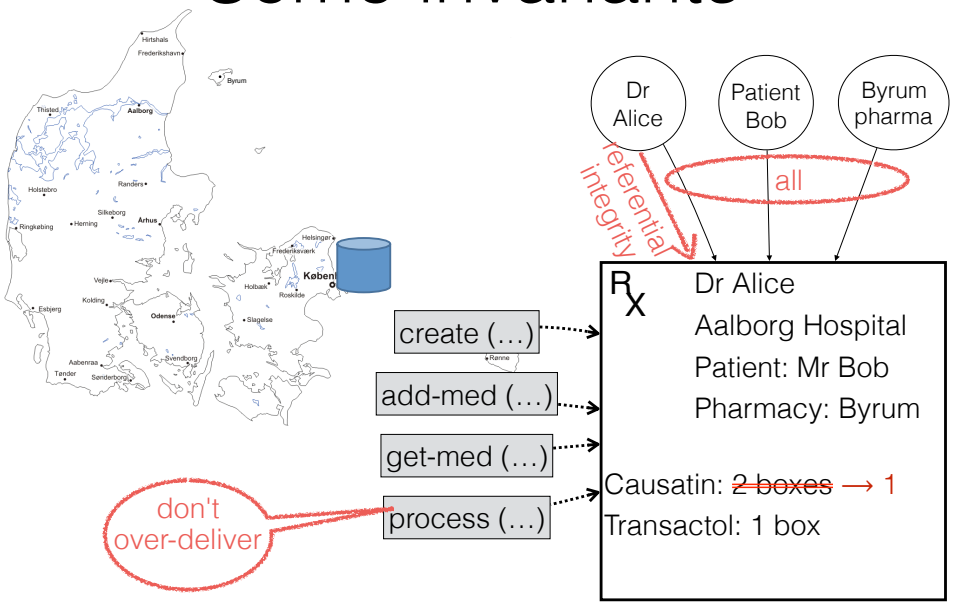
Asynchronous updates
Available under Partition; but not Consistent

Faster, less expensive
Concurrency anomalies

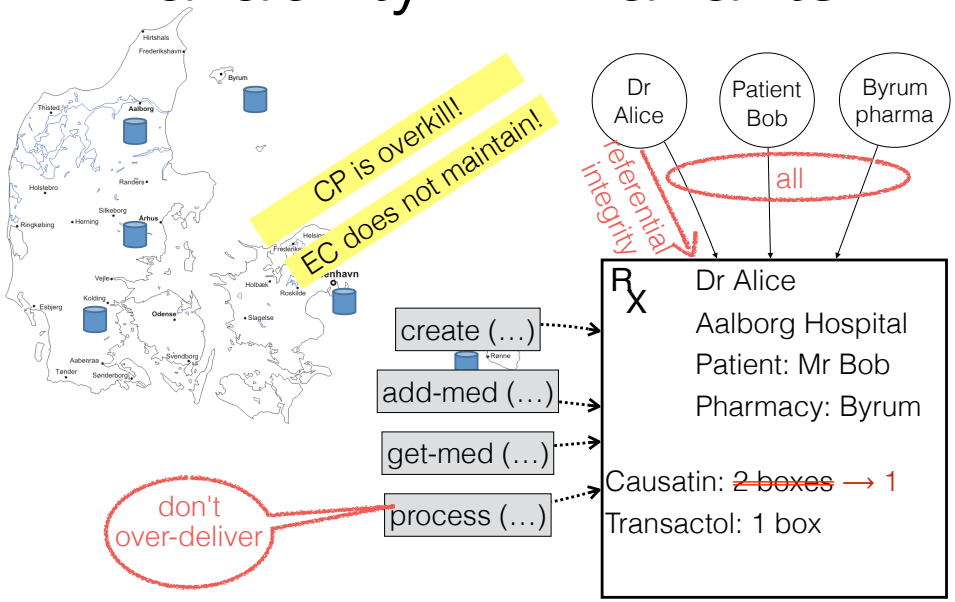
FMK *Fælles Medicinkort*



Some invariants



Availability + invariants?



Approach

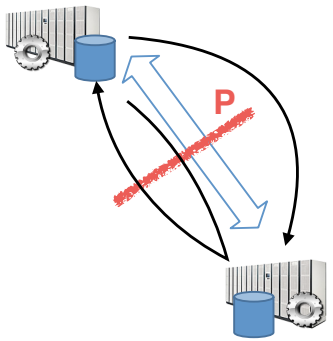
Consistency: one size does *not* fit all

Correct: maintain invariants

- But often unknown!

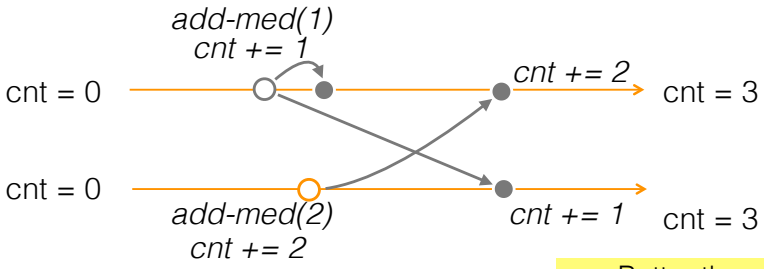
Methodology:

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



best possible availability and performance

CRDTs



Better than EC

CP is overkill!

Concurrent, asynchronous updates

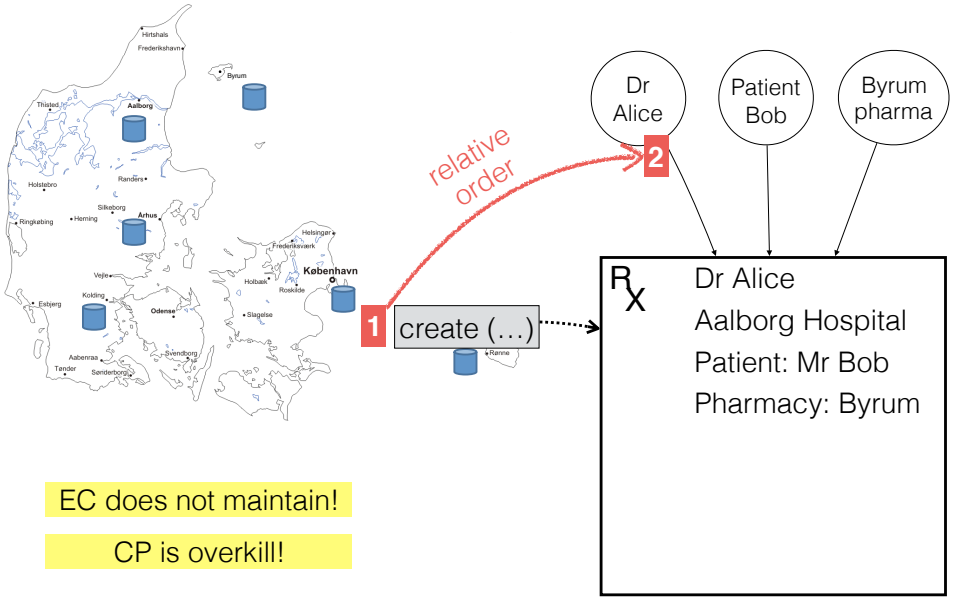
- Standard register model: assignments ⟹ CP
- AP ⟹ concurrent updates + merge

CRDT: register, counter, set, map, sequence

- Plug-in replacement for sequential type

Rx.Patient: write_once_reg. Rx.Meds: set

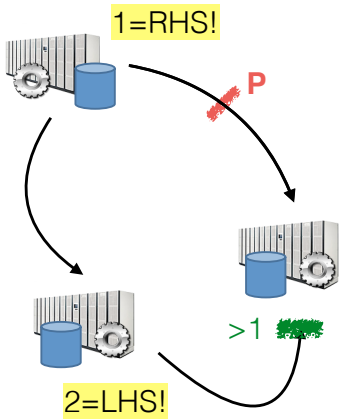
Pattern 1: relative order



EC does not maintain!

CP is overkill!

Causal consistency



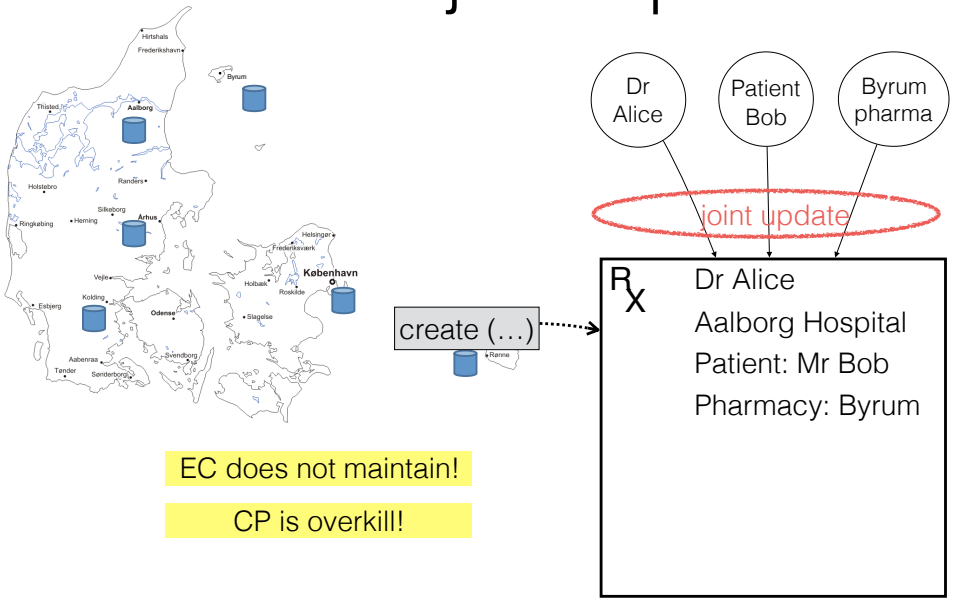
create-p before *add-med*

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

Deliver in the right order: Causal Consistency

AP-compatible

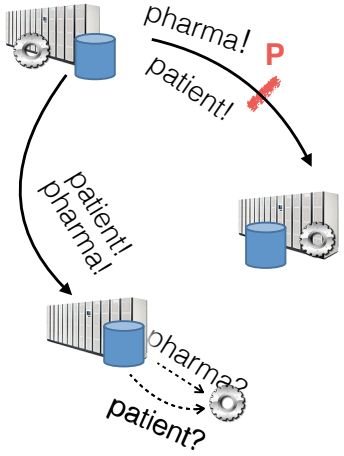
Pattern 2: joint update



EC does not maintain!

CP is overkill!

All-or-nothing



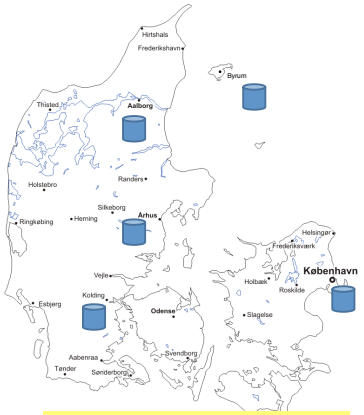
create-p updates doctor, patient & pharmacy record

Transmit joint updates together

\+ Read from same snapshot

AP-compatible

Pattern 3: precondition



EC does not maintain!

CP is overkill!

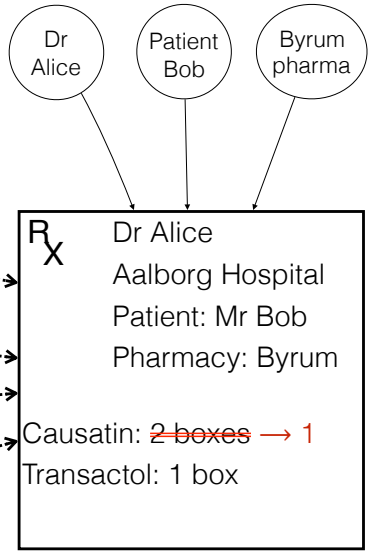
pre-condition

create (…)

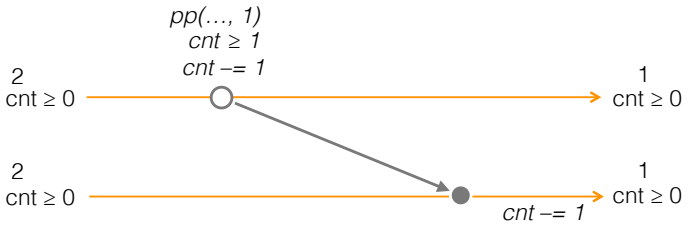
add-med (…)

get-med (…)

process (…)



CAP-sensitive invariants

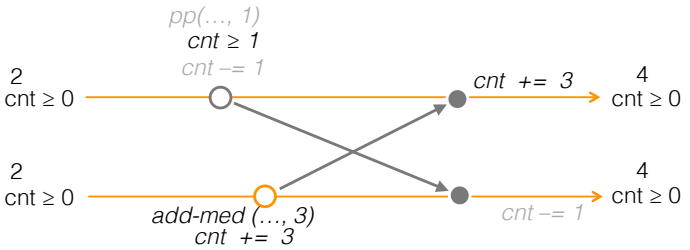


process-p (…, nb) {
  if *cnt* ≥ *nb* // precondition at source
    cnt –= *nb* // at every replica
} // *cnt* ≥ 0

Precondition *stable* w.r.t. concurrent *add-med*

Concurrency OK

Stable precondition

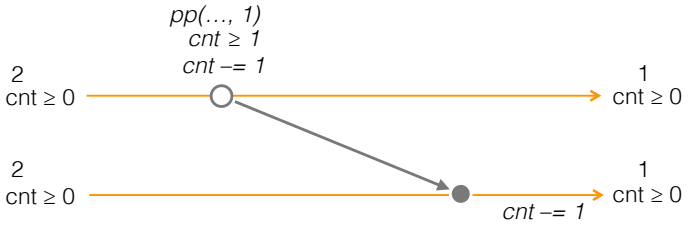


process-p (…, nb) {
  if *cnt* ≥ *nb* // precondition at source
    cnt –= *nb* // at every replica
} // *cnt* ≥ 0

Precondition *stable* w.r.t. concurrent *add-med*

Concurrency OK

CAP-sensitive invariants

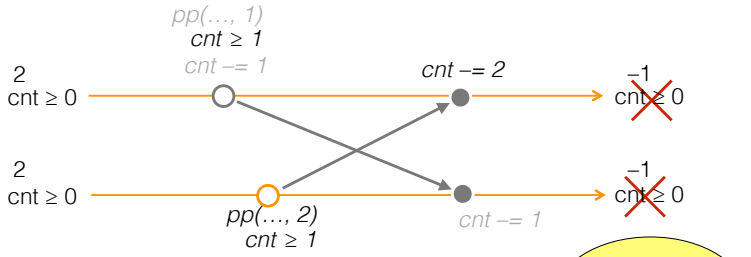


process-p (…, nb) {
  if *cnt* ≥ *nb* // precondition at source
    cnt –= *nb* // at every replica
} // *cnt* ≥ 0

Precondition *not stable* w.r.t. concurrent *process-p*

- Forbid concurrency? Synchro, CP.
- Or remove invariant? AP, degraded semantics

Not stable precondition



process-p (…, nb) {
  if *cnt* ≥ *nb* // precondition at source
    cnt –= *nb* // at every replica
} // *cnt* ≥ 0

Precondition *not stable* w.r.t. concurrent *process-p*

- Forbid concurrency
- Or, give up on invariant

Summary: Just-Right Consistency

Tailor consistency to application invariants
- (possibly unknown)

Baseline: Correct app, 1-copy, one op. at a time

Three patterns:

- Ordered updates ⟹ Causal Consistency, **AP**
- Joint updates ⟹ All-or-nothing, **AP**
- CAP-sensitive: precondition
  - Stable ⟹ concurrent OK. **AP**.
  - Otherwise, concurrency control. **CP**

CRDT data model
- Register, counter, set, map, sequence
- Extends sequential semantics

Transactional Causal Consistency (TCC)
- Strongest AP model
- Supports Joint Updates, Relative Order

CISE static analysis tools

Open source, well engineered

LIGHTKONE
Lightweight computation for networks at the edge

European Commission

RainbowFS