Part II

Consistency and Invariant Preservation

LIGHTKONE

Lightweight computation for networks at the edge

Which consistency does my application need?

- Many applications have constraints defined on the data that might not hold when operations execute concurrently.
- No “one-size-fits-all” consistency model
  - Choosing either model will either be over-conservative or risk anomalies
- Idea: Tailor consistency choices based on application-level invariants for each operation
- AP-compatible invariants
  - Invariants that only require “one way” communications
- CAP-sensitive invariants
  - Involve operations that require coordination
  - “two way” communication invariants

AP-compatible invariants

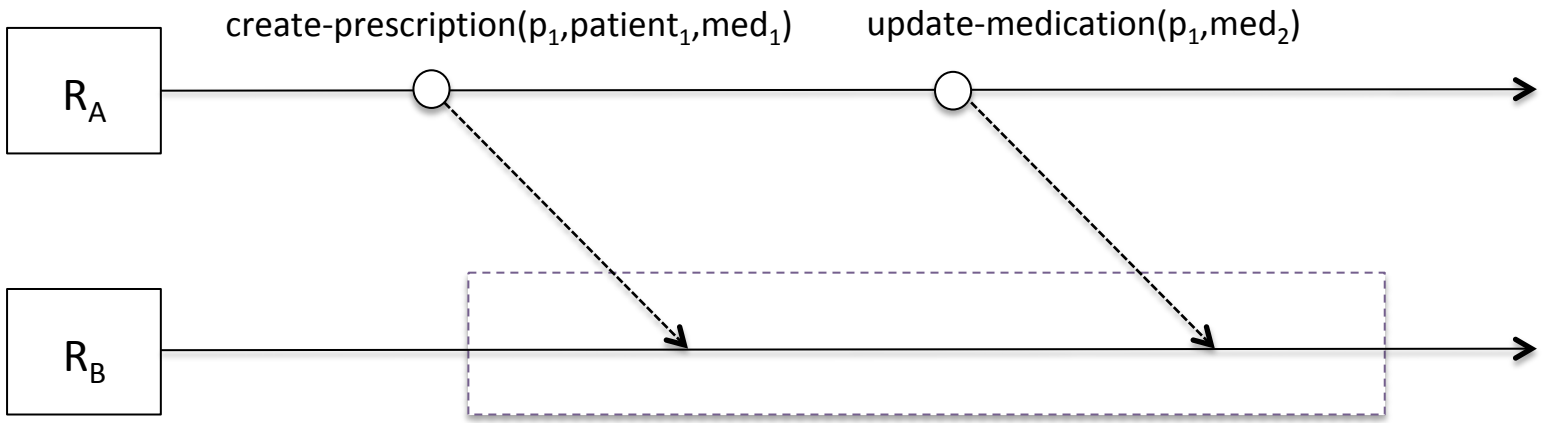
- No synchronization
  - Updates occur locally without blocking
- Asynchronous operation
  - Updates are fast, available, and exploit concurrency
- CRDTs are AP-compatible data model
- Compatible invariants
  - Relative order and joint update invariants can be preserved

Use Case: FMK

- Fælles Medicinkort: Prescription management in the Danish national healthcare system

  - **create-prescription**
    Create prescription for patient, doctor, pharmacy
  - **update-medication**
    Add or increase medication to prescription
  - **process-prescription**
    Deliver a medication by a pharmacy
  - **get-*-prescriptions**
    Query functions to return information about prescriptions

AP-compatible: Relative order

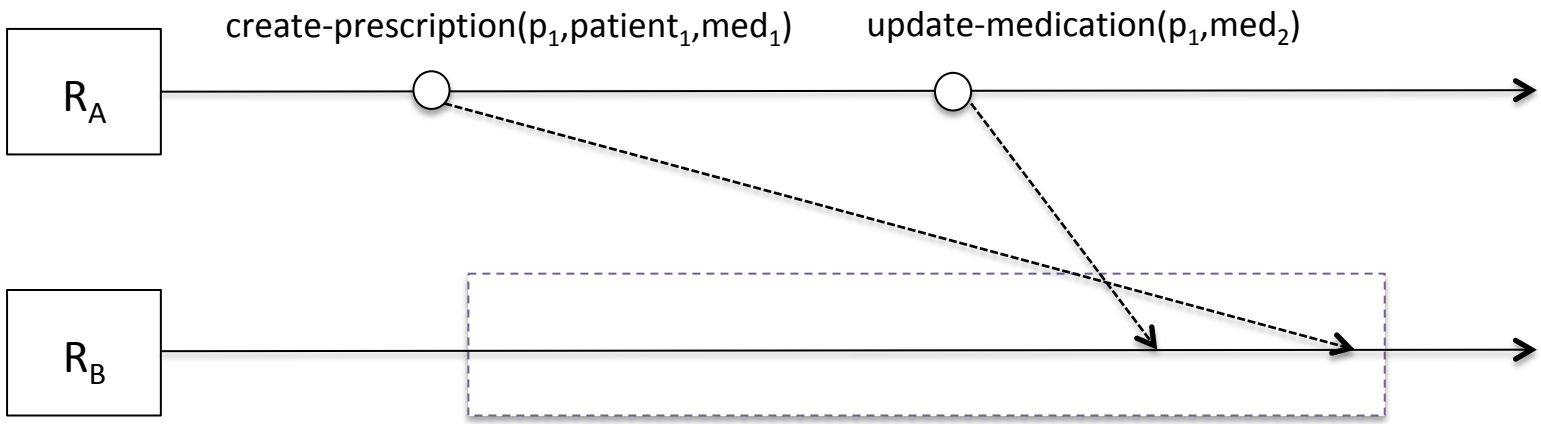


Ordering is respected at other replicas

- Maintain program order implication invariant:
  "Only if prescription exists, medication can be adapted"
- Transmit in the right order!

LIGHTKONE
Lightweight computation for networks at the edge

AP-compatible: Relative order



Out of order propagation violates invariant!

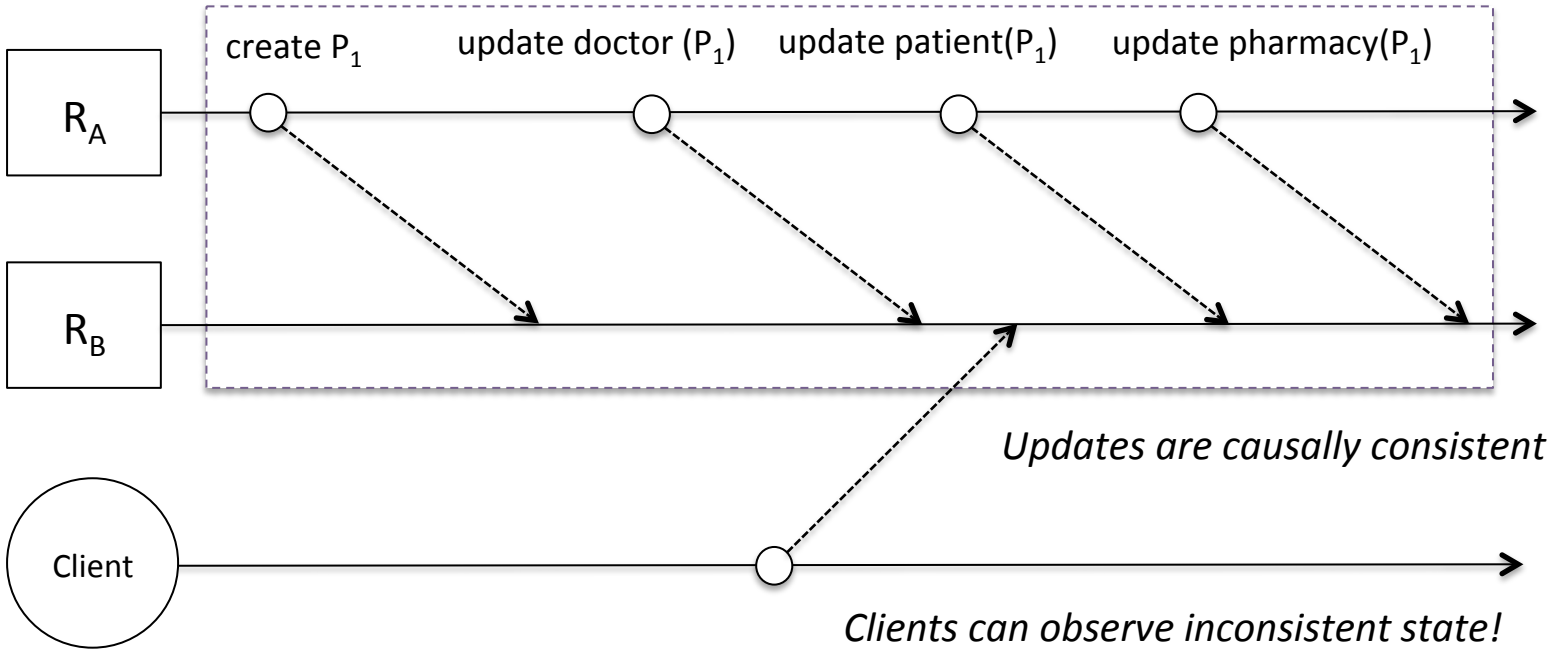
- Maintain program order implication invariant:
  "Only if prescription exists, medication can be adapted"
- Transmit in the right order!

LIGHTKONE
Lightweight computation for networks at the edge

Causal consistency

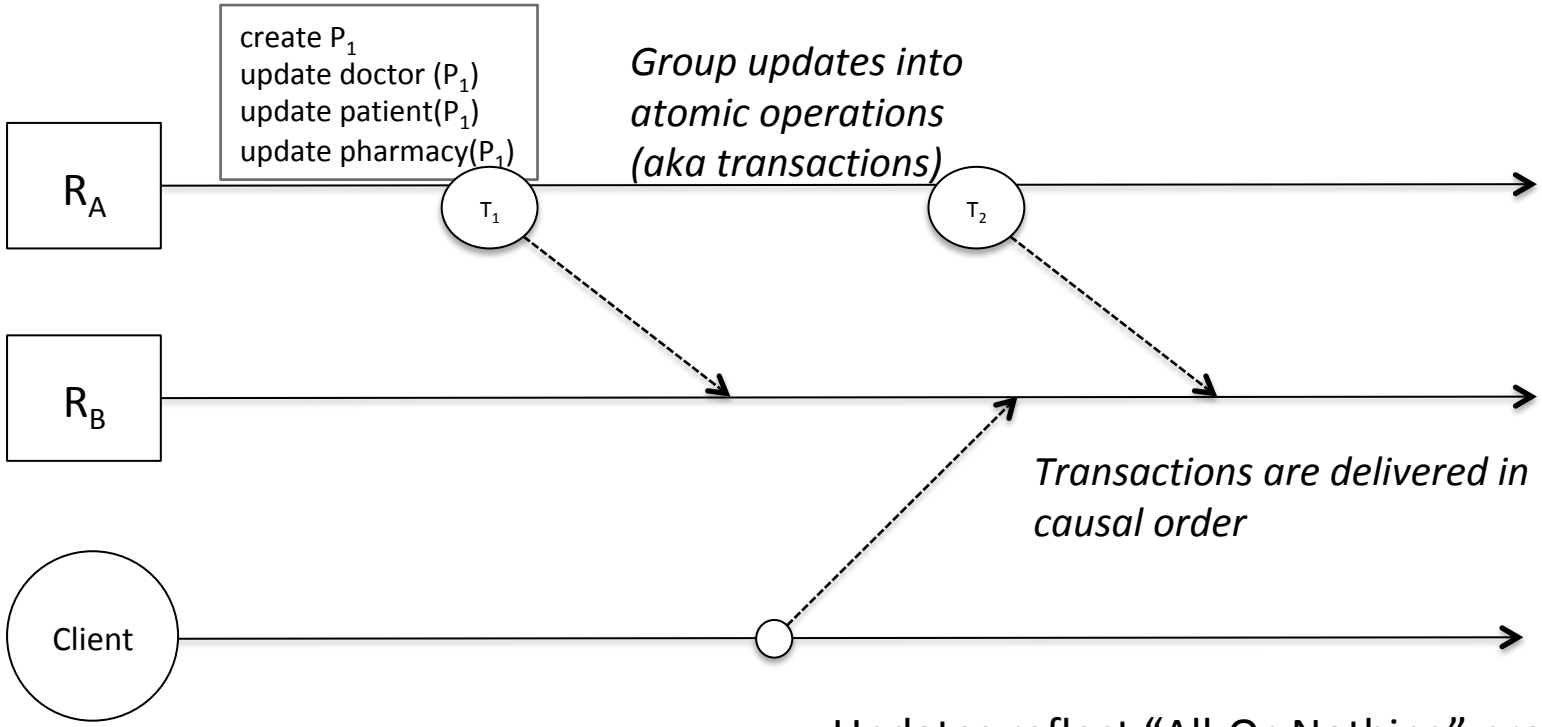
- No ordering anomalies: $u \longrightarrow v \bigwedge visible(v) \Longrightarrow visible(u)$
- Respect causality
  - Ensure updates are delivered in the causal order [Lamport 78]
- Strongest available model
  - Always able to return some compatible version for an data object
- Referential integrity
  - Causal consistency is sufficient for providing referential integrity in an AP database

AP-compatible: Joint updates



LIGHTKONE
Lightweight computation for networks at the edge

AP-compatible: Joint updates



Updates reflect “All-Or-Nothing” property through snapshots.

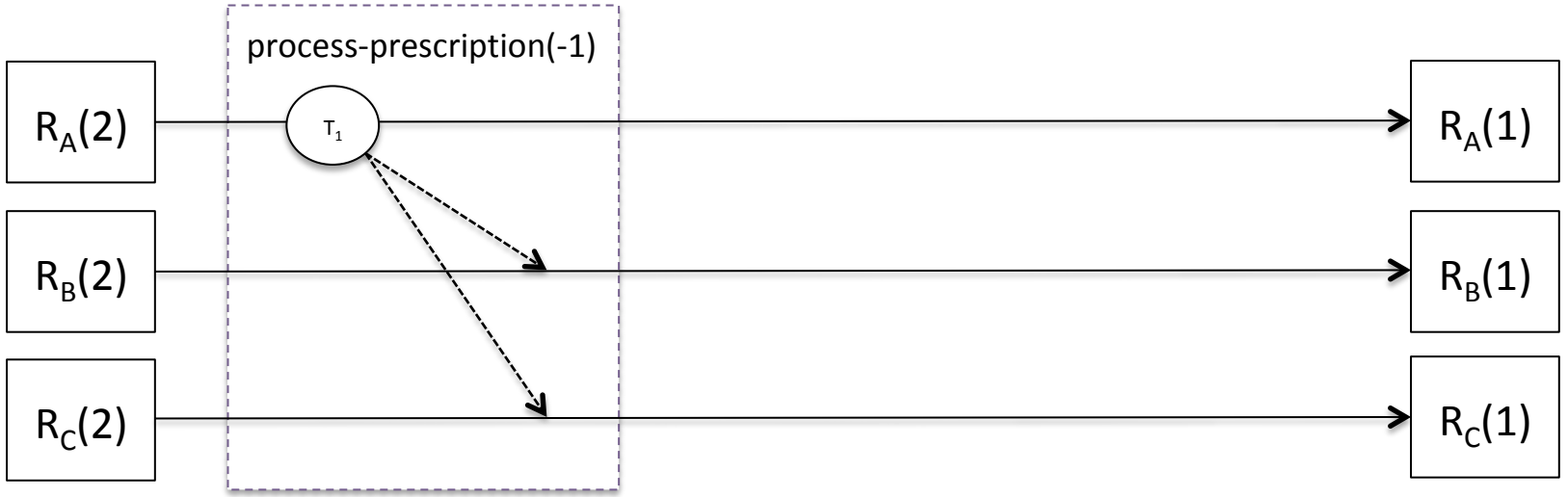
LIGHTKONE
Lightweight computation for networks at the edge

Transactional Causal+ Consistency

- Causal consistency
- Transactional reads
  - Clients observe a consistent snapshot
- Transactional writes+
  - Updates become observable all-or-nothing
  - Concurrent updates converge to same value for all replicas

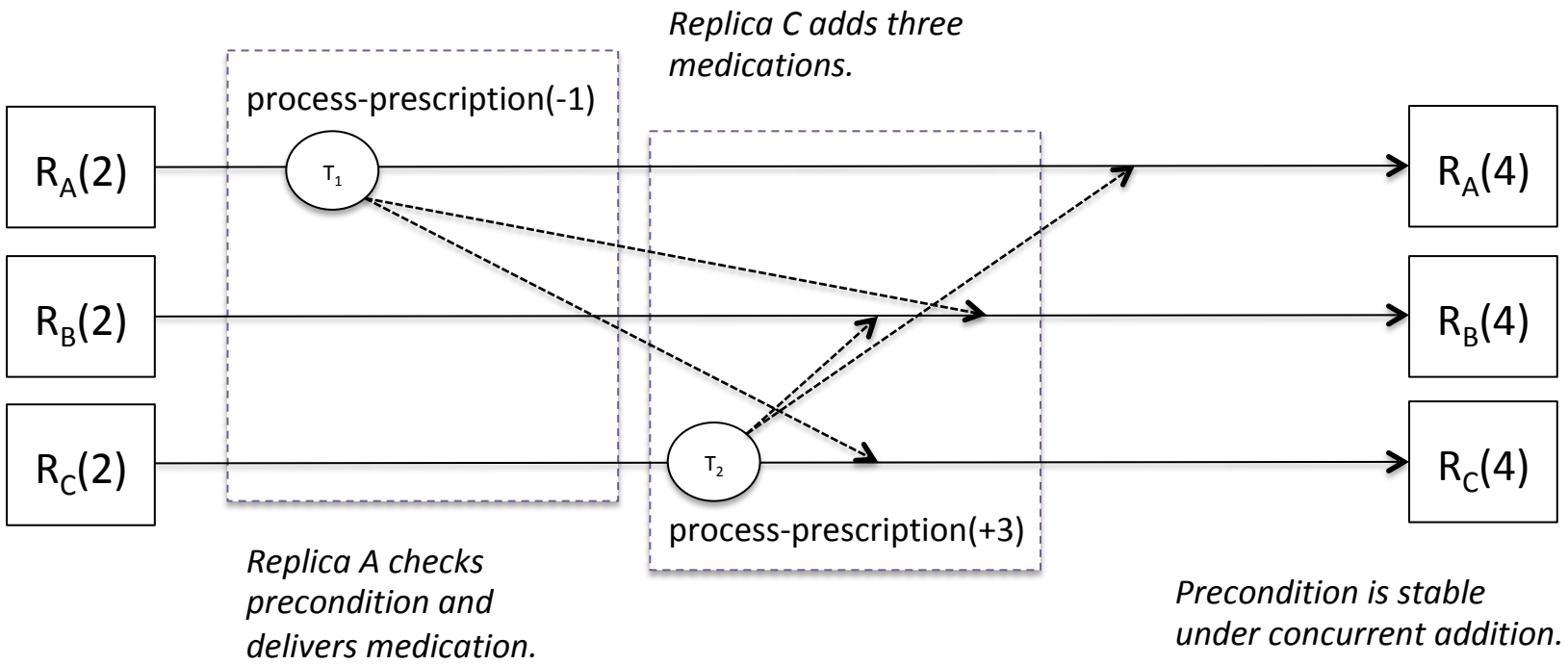
CAP-sensitive invariants

Do not over-deliver medication!



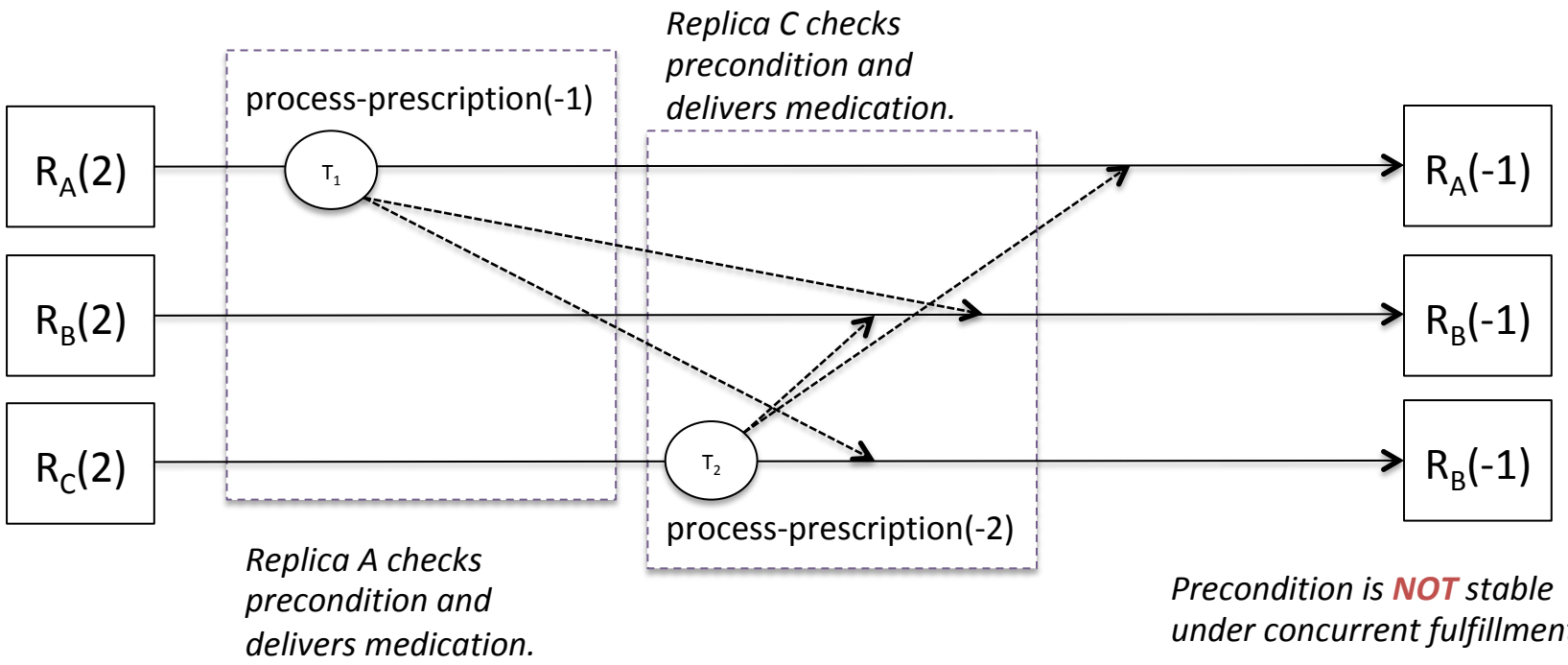
Replica A checks precondition and delivers medication.

CAP-sensitive invariants



LIGHTKONE
Lightweight computation for networks at the edge

CAP-sensitive invariants



LIGHTKONE
Lightweight computation for networks at the edge

Conclusion: Part II

- Strong consistency is often too conservative
  - Many operations are safe without cross-site coordination
- **Causal consistency** ensures that relative order invariants are preserved transparently
- **Transactional** causal consistency provides additionally atomicity and isolation
- What about operations that are really unsafe under weak consistency?
  - Require coordination (but only sporadically)