More powerful CRDTs

- Let us now look at some more powerful CRDTs
- We show the Up-Down Counter and the Observed-Remove set
- Many more powerful CRDTs exists; we refer you to the bibliography to find out more
- We compare CRDTs with RSMs as a way to implement distributed data structures

Up-Down Counter (PN Counter)

- Each replica i stores $s=(u_1, \ldots, u_n, d_1, \ldots, d_n)$ where $u_i, d_i \in N$ (natural)
- Each replica accepts inc, dec, val, and $\sqcup$ (join) operations
  - inc_i: update s to s' where $s'=(u_1, \ldots, u_i+1, \ldots, u_n, d_1, \ldots, d_n)$
  - dec_i: update s to s' where $s'=(u_1, \ldots, u_n, d_1, \ldots, d_i+1, \ldots, d_n)$
  - val_i: return $\sum_{1 \leq j \leq n} s.j - \sum_{n+1 \leq j \leq 2n} s.j$
  - join: $s \sqcup s' = (\max(u_1,u_1'), \ldots, \max(u_n,u_n'), \max(d_1,d_1'), \ldots, \max(d_n,d_n'))$

- How does this work?
  - Both inc and dec will inflate the value on the lattice
  - The val function calculates the correct value by doing a subtraction
  - Eventually all replicas will converge to the correct value, as before

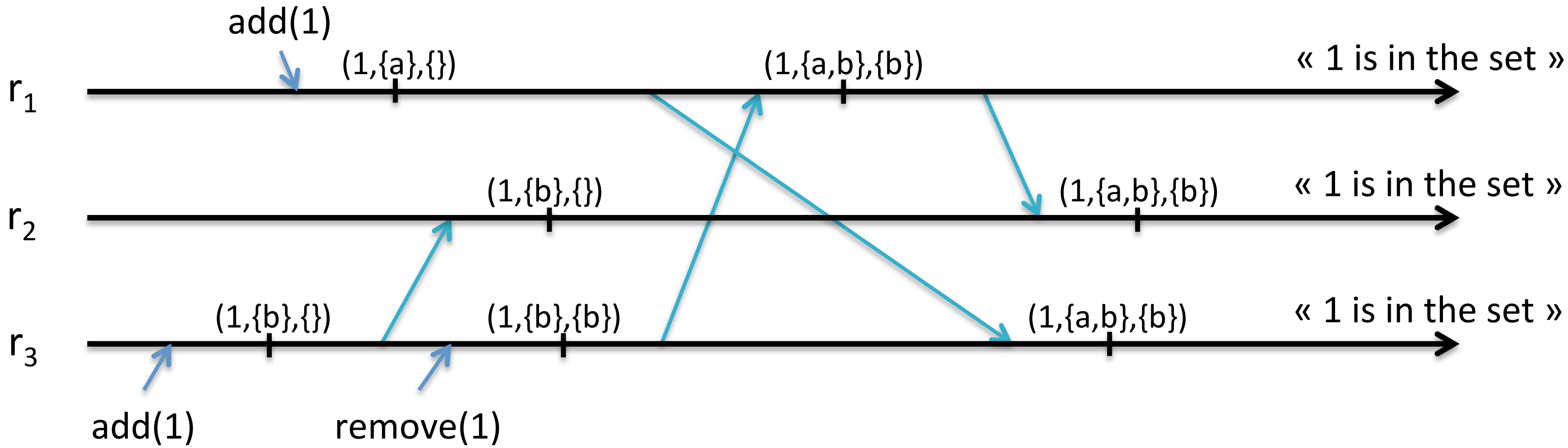
LIGHTKONE

Observed-Remove Set

- The OR-Set supports both adding and removing elements
  - The outcome of a sequence of adds and removes depends only on the causal history and conforms to the sequential specification of a set
  - In case of concurrent add and remove, the add has precedence
- The intuition is to tag each added element uniquely
  - The tag is not exposed when querying the set content
  - When removing an element, all tags are removed

LIGHTKONE

Observed-Remove Set



- Each replica stores triples (e,a,r) where e is the element, a is the set of adds and r is the set of removes
- If (e,a,r) with a-r≠{} then e is in the set
  - All updates (both adds and removes) cause monotonic increases in (e,a,r)

Other CRDTs

- Many CRDTs have been invented
  - Registers: last-writer wins, multi-value
  - Sets: grow-only, 2P, add-wins, remove-wins
  - Maps, Pairs (including recursive versions)
  - Counter: unlimited, restricted ≥ 0 (bounded)
  - Graph: directed, monotonic DAG, edit graph
  - Sequence / List

LIGHTKONE

Comparison CRDT $\longleftrightarrow$ RSM

- In the course we have now seen two ways to define replicated distributed data structures
  – Replicated State Machine (RSM) approach
  – CRDT approach
- What is the difference?
  – RSM approach ensures consistency of replicas after each update, at the cost of needing consensus (e.g., Paxos or Raft)
  – CRDT approach ensures consistency when replicas have received the same set of updates, which needs only node-to-node communication

LIGHTKONE

What's the catch?

- Many companies and applications are using CRDTs, and their number is growing daily
  - But if CRDTs are so great, why isn't everybody using them?
- Trade-offs for using CRDTs
  - CRDTs require meta-data to ensure monotonicity and causality, which grows with the number of replicas
  - State-based CRDTs have growing state (tombstones), which requires some form of (unsynchronized) garbage collection
  - Last-writer-wins with physical clocks undergoes clock skew

LIGHTKONE

Rest of the lesson

- My colleagues Chris and Annette will now explain two important directions of this work:
- Lasp: a programming language and platform based on strong eventual consistency
- Antidote: a causally consistent transactional database based on strong eventual consistency

LIGHTKONE