Convergent data structures

- We can define distributed data structures that obey Strong Eventual Consistency
  - One approach: Conflict-free Replicated Data Type (CRDT)
- Many CRDTs exist and have millions of users

CRDT definition

- A *state-based CRDT* is defined as a triple $((s_1, \ldots, s_n),m,q)$:
  - $(s_1, \ldots, s_n)$ is the configuration on n replicas, with $s_i \in S$ where S is a join semilattice
  - $q_i:S \longrightarrow V$ is a query function (read operation)
  - $m_i:S \longrightarrow S$ is a mutator (update operation) such that $s \sqsubseteq m(s)$
  - Periodically, replicas update each other's state: $\forall$ i,j: $s_i' = s_i \sqcup s_j$
- Because the mutator only inflates the value, and because of the periodic dissemination, all replicas will eventually converge to the same final value

LIGHTKONE

Join semilattice

- A *join-semilattice* is a partially ordered set S that has a least upper bound (join) for any nonempty finite subset:
  - Partial order: $\forall x, y, z \in S$:
    Reflexivity: $x \sqsubseteq x$
    Antisymmetry: $x \sqsubseteq y \wedge y \sqsubseteq x \Rightarrow x = y$
    Transitivity: $x \sqsubseteq y \wedge y \sqsubseteq z \Rightarrow x \sqsubseteq z$
  - Least upper bound (join): $\forall x, y \in S$: $x \sqcup y \in S$
    - $z = x \sqcup y$ is an upper bound
    - All other upper bounds are at least as large as z

LIGHTKONE

CRDTs satisfy SEC

- Strong Eventual Consistency (SEC)
  - We assume eventual delivery: an update delivered at some correct replica is eventually delivered to all correct replicas
    - Eventual replica-to-replica communication satisfies this
  - An object is SEC if all correct replicas that have delivered the same updates have equivalent state
- Theorem: A state-based CRDT satisfies SEC
  - Proof by induction on the causal histories of deliveries at the replicas
  - Proof given in INRIA Research Report RR-7687 (see bibliography)

LIGHTKONE

Example: Grow-Only Counter

- Each replica i stores $s=(c_1, c_2, \ldots, c_i, \ldots, c_n)$ where $c_i \in N$ (natural)
- Each replica accepts inc, val, and $\sqcup$ (join) operations
  - inc_i: update s to s' where $s'=(c_1, c_2, \ldots, c_i+1, \ldots, c_n)$
  - val_i: return $\sum_{j \in i} s.j$
  - join: $s \sqcup s' = (\max(c_1,c_1'), \ldots, \max(c_n,c_n'))$

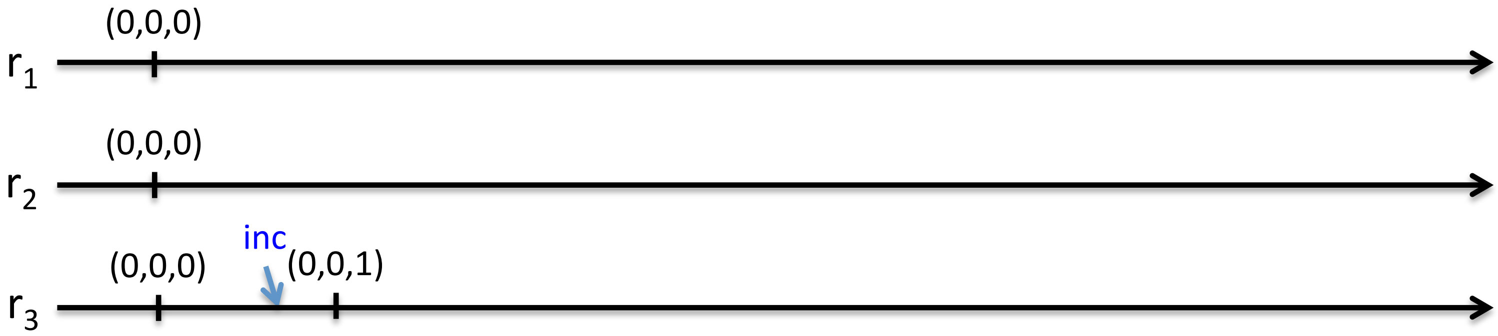
- How does this work?
  - The state vector stores the increments done at each replica
  - Eventually, all replicas' vectors will converge to know all increments

LIGHTKONE

Example execution

- Three replicas, each replica stores a 3-vector giving the increments it knows of at each replica

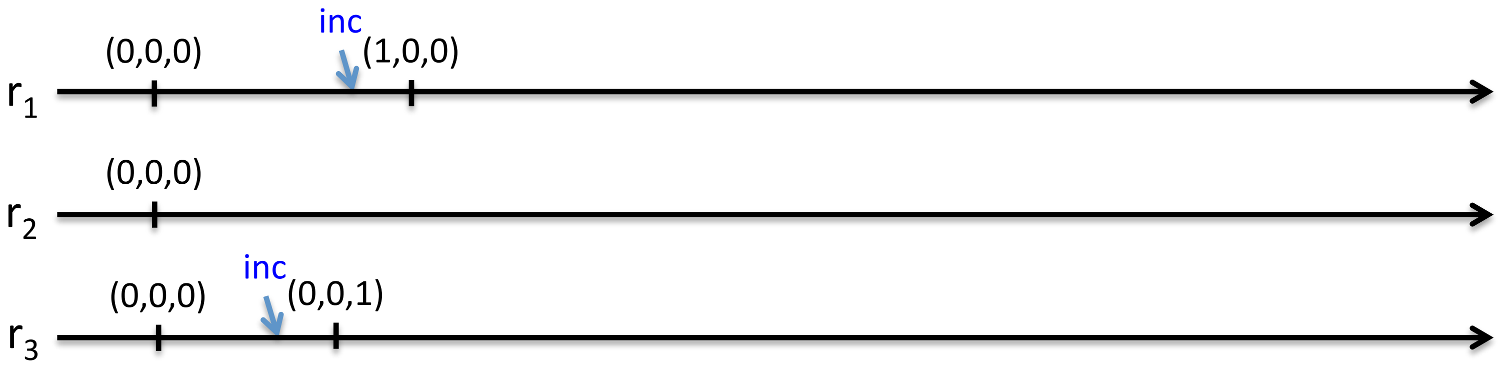
Example execution



- Increment at replica 3, its vector becomes (0,0,1)

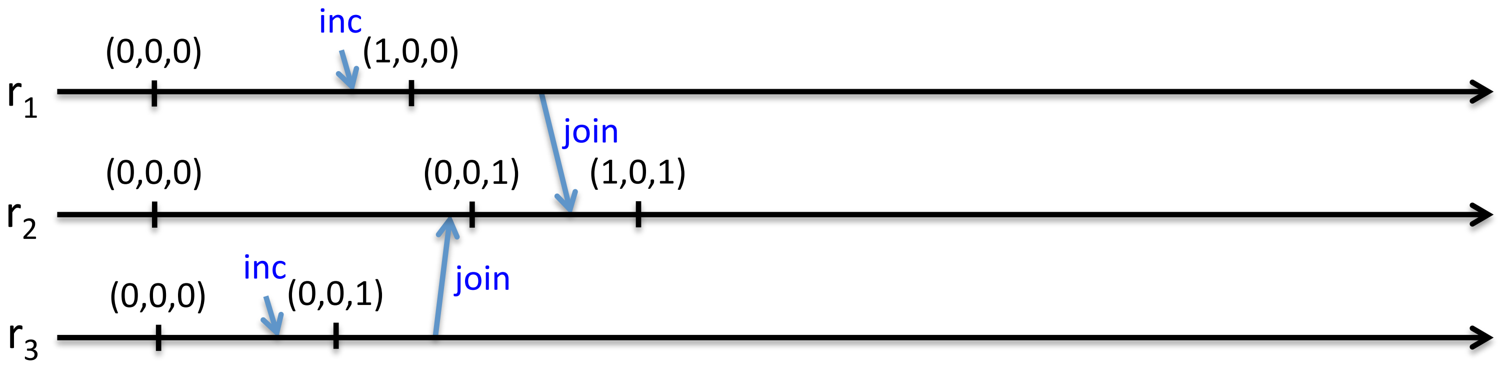
LIGHTKONE

Example execution



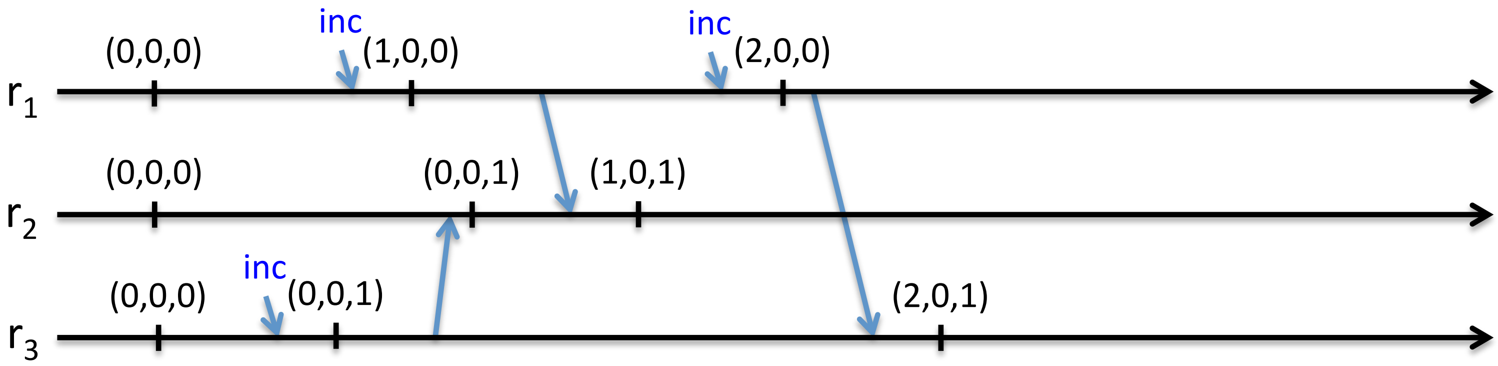
- Increment at replica 1, its vector becomes (1,0,0)

Example execution



- Join operations merge state from replica 1 and replica 3
- Replica 2’s state is updated to (0,0,1) and then to (1,0,1)

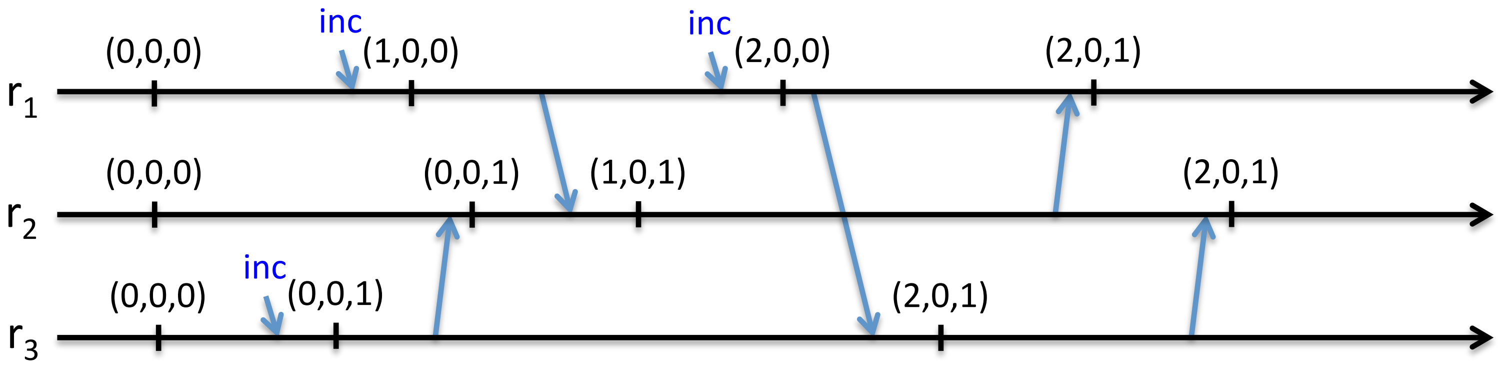
Example execution



- Another increment at replica 1 and a join to replica 3
- Replica 3's state becomes (2,0,1)
- Replica 1 is (2,0,0) and replica 2 is (1,0,1)

LIGHTKONE

Example execution



- Join operation from replica 2 to replica 1
- Join operation from replica 3 to replica 2
- All replicas have converged to the state (2,0,1)

LIGHTKONE

Carrying on

- The Grow-Only counter is one of the simplest CRDTs
  - Each replica stores information about all replicas, very much like a vector clock
- How expressive can a CRDT be?
  - Can we express counters that both increment and decrement?
  - Can we express sets where we can both add and remove elements?
- The answer is, yes, a CRDT can express all that and more
  - We will look at some smarter CRDTs in the next video

LIGHTKONE