CRDTs Illustrated

Arnout Engelen / @raboofje / Xebia

Xebia

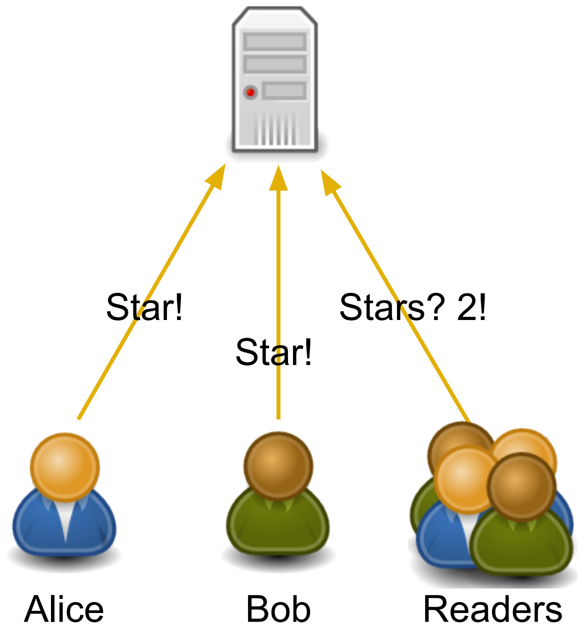
Today’s topic

Goal

Consistency in Distributed Systems

Consistency in Distributed Systems

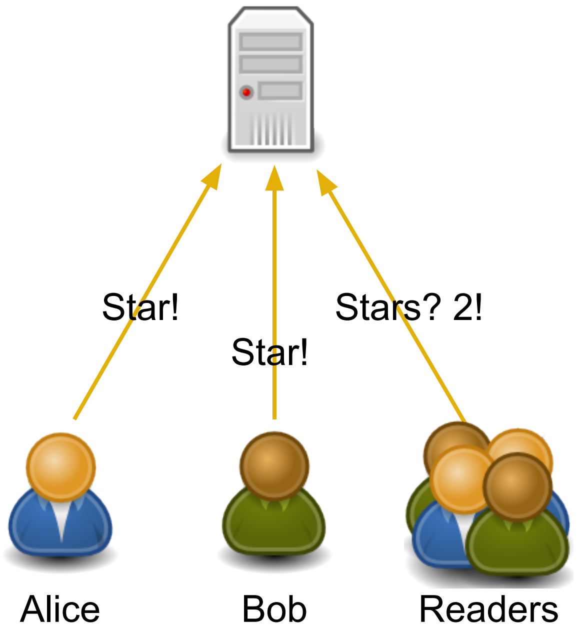
Simple (non-distributed) case: single server



Xebia

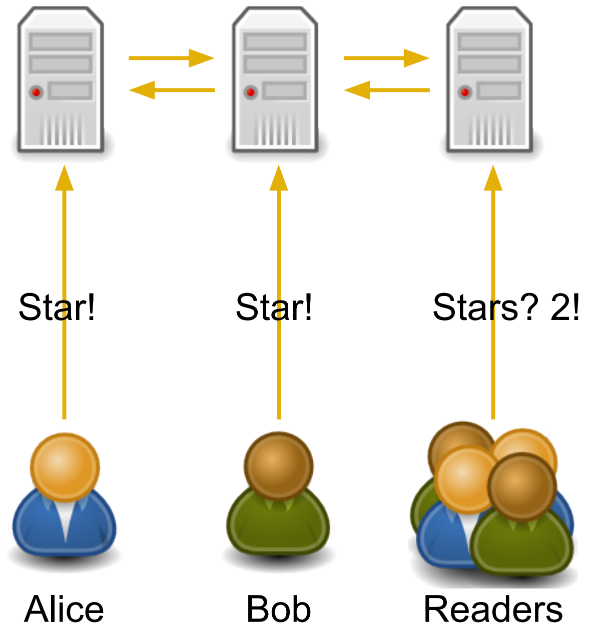
Consistency in Distributed Systems

Problem: network failures :(

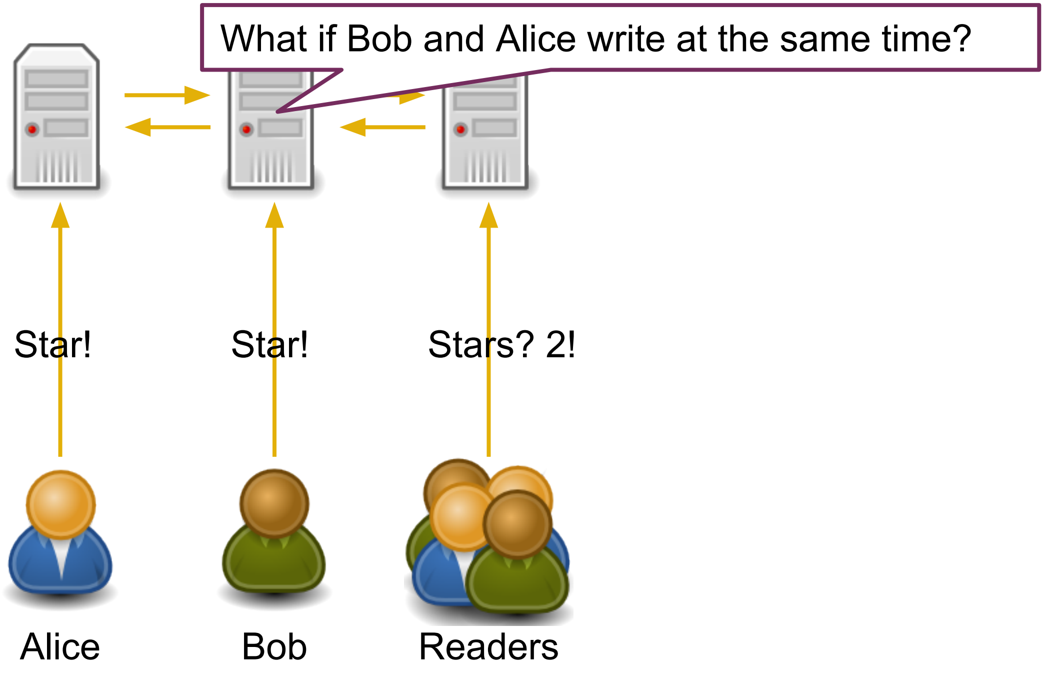


Consistency in Distributed Systems

‘Solution’: distribute!

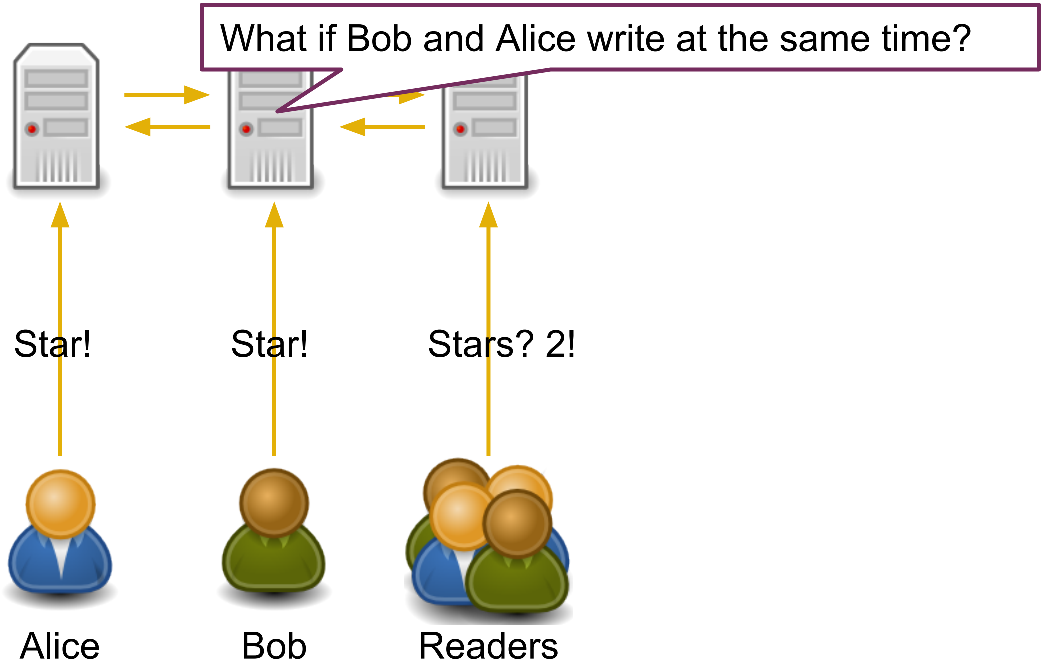


Consistency in Distributed Systems



Consistency in Distributed Systems

Problem: still network failures :(



Consistency in Distributed Systems

Writing in the presence of failure

- Strong Consistency
  - Sequential writes
  - Impossible when A and B are disconnected
  - “No availability in case of network partitions”

Consistency in Distributed Systems

Writing in the presence of failure

- Strong Consistency
  - Sequential writes
  - Impossible when A and B are disconnected
  - “No availability in case of network partitions”

- Eventual Consistency
  - Update partitions independently, converge ‘eventually’
  - Complicated algorithms, hard to verify/test

Xebia

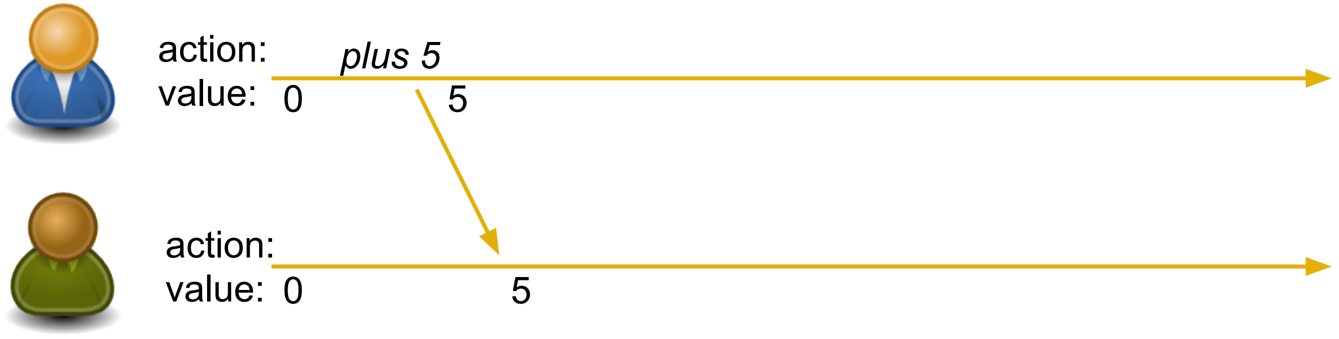
CRDTs

CRDTs

Strong Eventual Consistency

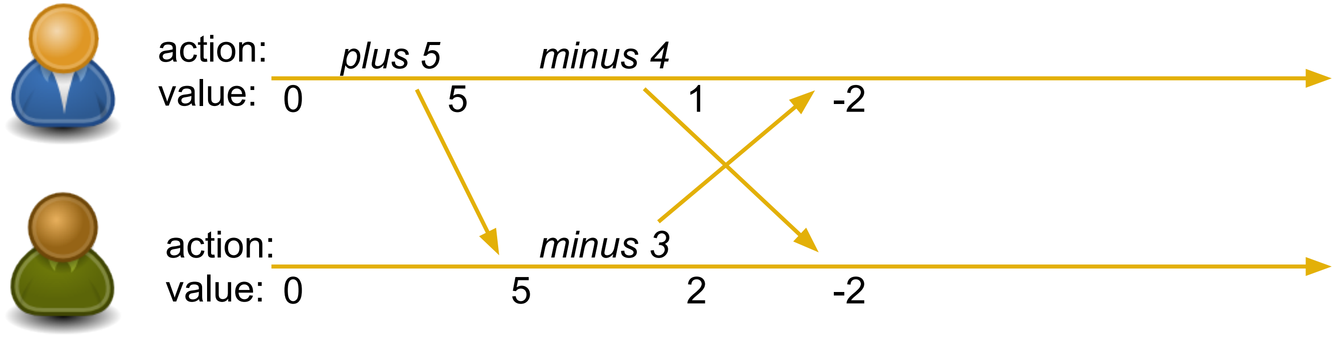
- Once you’ve seen the same events, you’re (immediately) in the same state

CRDT: counter

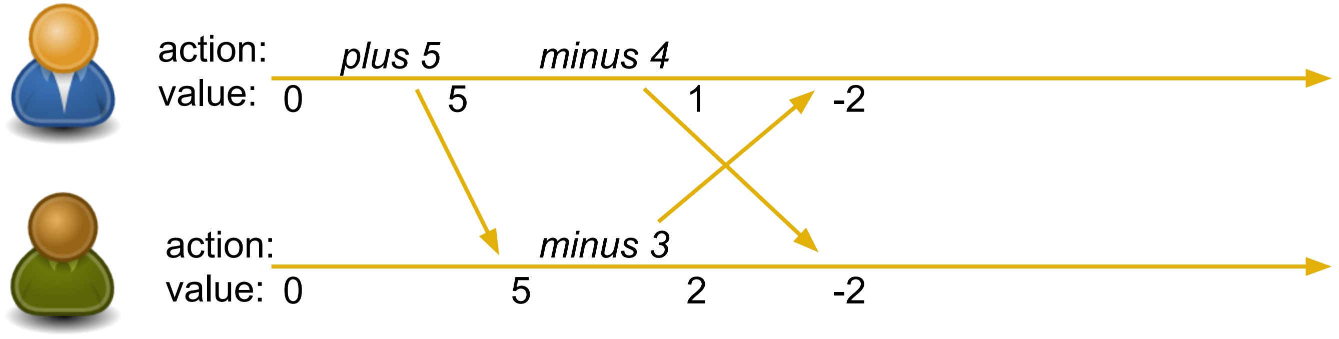


Xebia

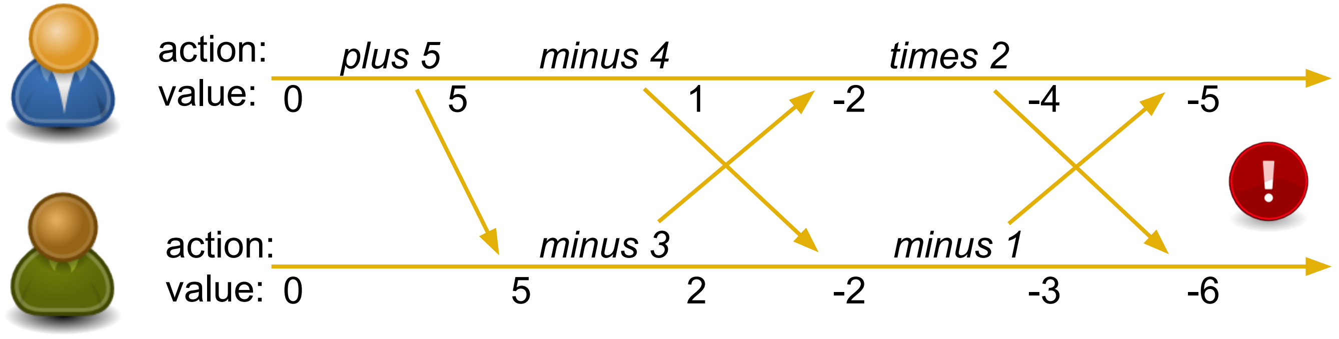
CRDT: counter



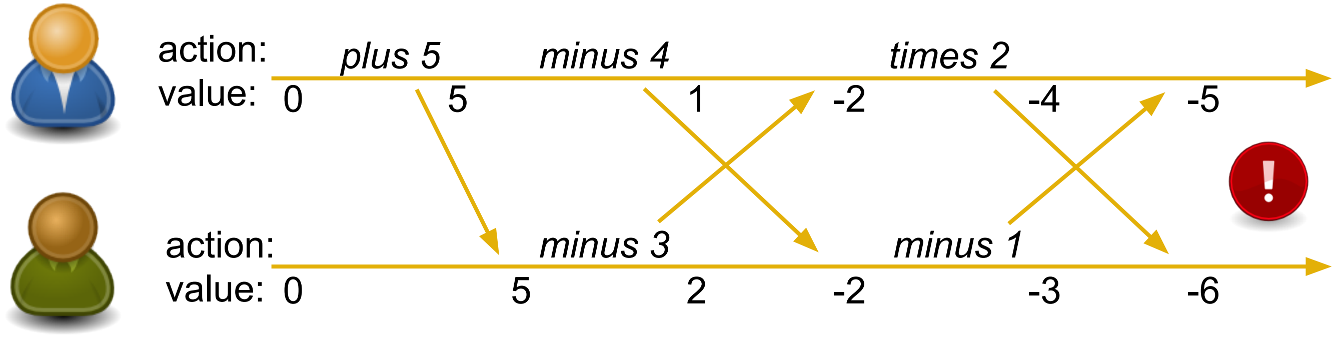
CRDT: Op-based counter



CRDT: Op-based counter



CRDT: Op-based counter

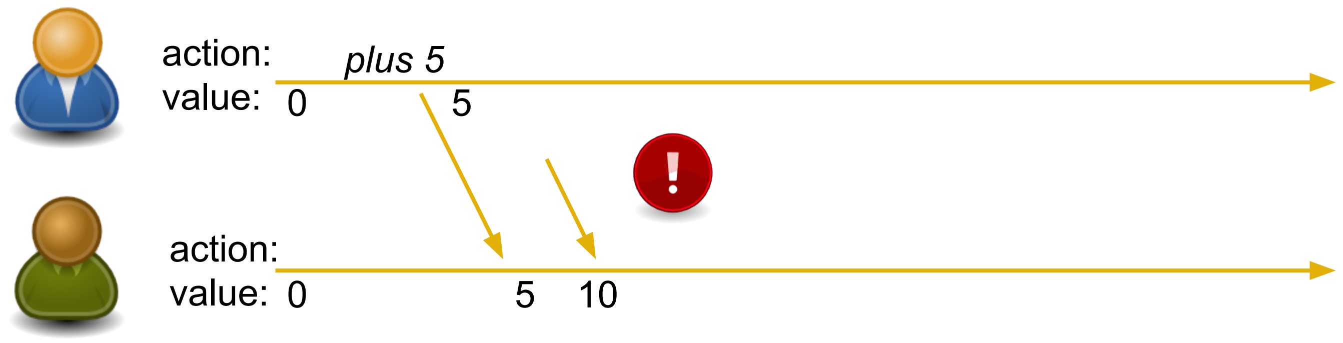


Op-based CRDT rule 1:

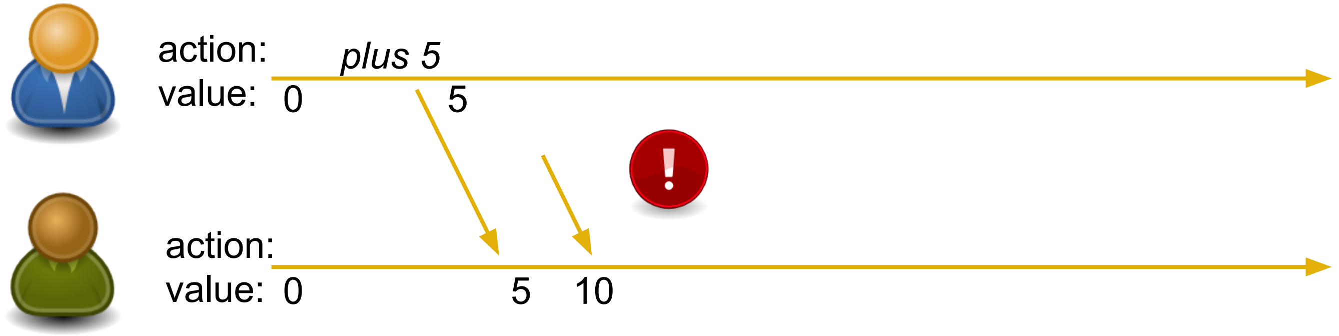
- All concurrent operations must commute
  - (x-4)-3 == (x-3)-4
  - (x*2)-1 != (x-1)*2

Xebia

CRDT: Op-based counter



CRDT: Op-based counter



Op-based CRDT rule 2:

- Updates must be applied exactly once
  - Applying ‘+5’ twice invalidates the result

Exactly Once Delivery

Exactly Once Delivery

- .. is generally impossible when partitions happen
    - Who acknowledges the acknowledgement?

Exactly Once Delivery

- .. is generally impossible when partitions happen
  - Who acknowledges the acknowledgement?

- Pick one:
  - At Most Once Delivery (fire & forget)
  - At Least Once Delivery (retry)

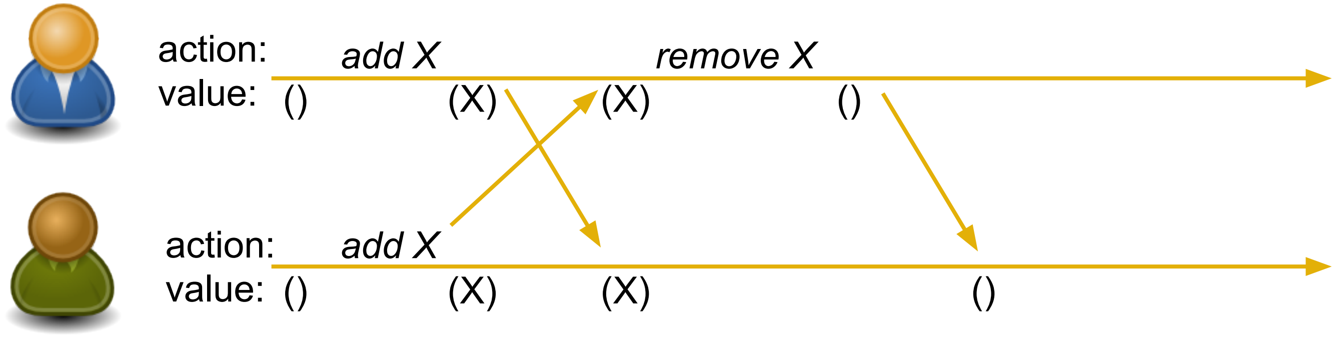
Exactly Once Delivery

- .. is generally impossible when partitions happen
  - Who acknowledges the acknowledgement?

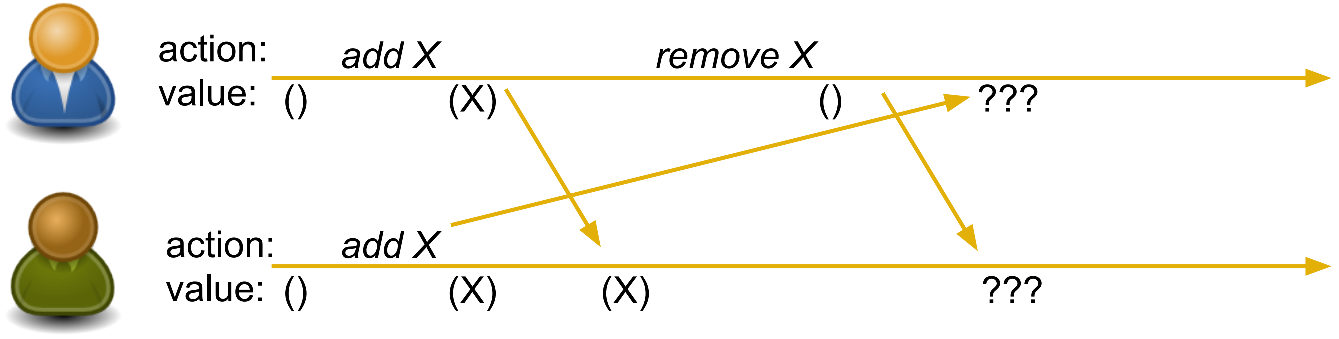
- But: Exactly Once Delivery *Semantics* are possible!
  - when processing the same message again has no effect
  - *Idempotence*

Xebia

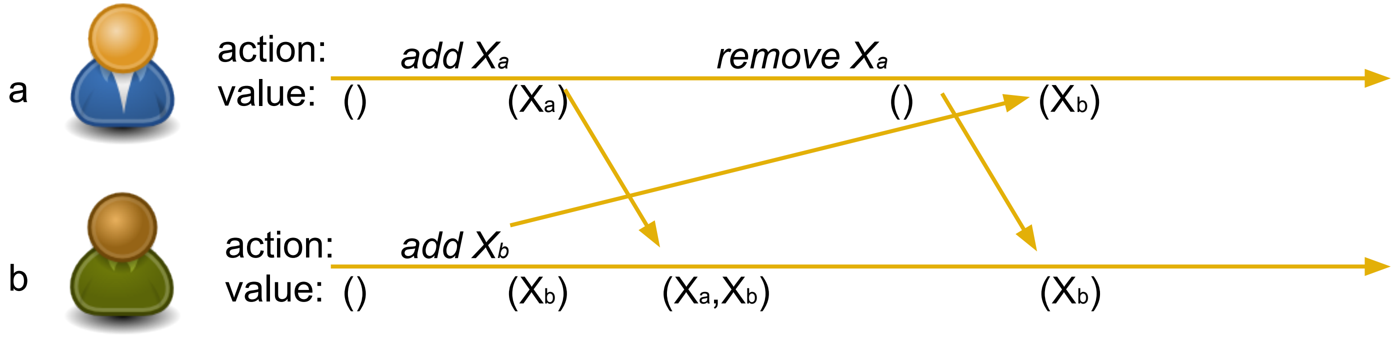
Sets: naive approach



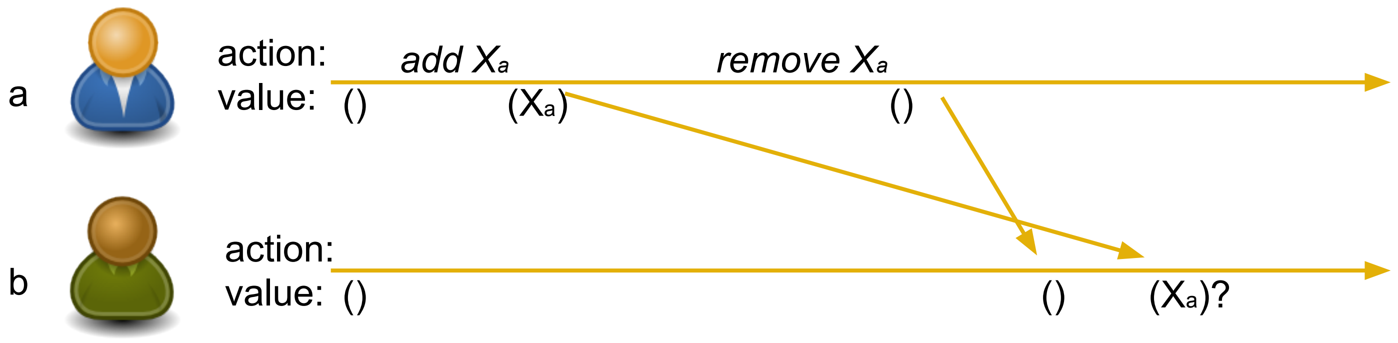
Sets: naive approach



CRDT: Observed-Remove Set

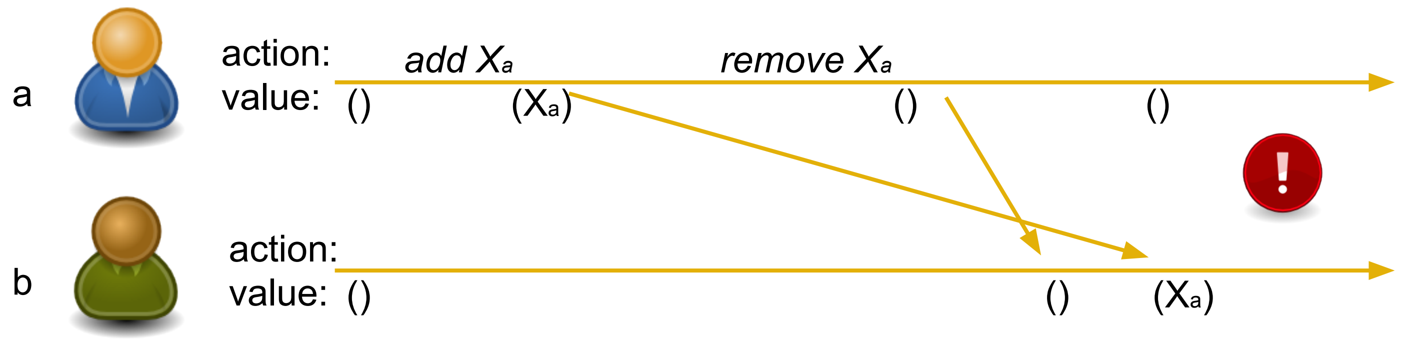


CRDT: Observed-Remove Set



○

CRDT: Observed-Remove Set



Op-based CRDT rule 3:

- Updates must be applied in-order
  - (in which they were sent from their origin)

CRDTs

Operation-based CRDTs:

- All (concurrent) operations must *commute*
- Require *exactly-once* delivery semantics
- Require *in-order* delivery

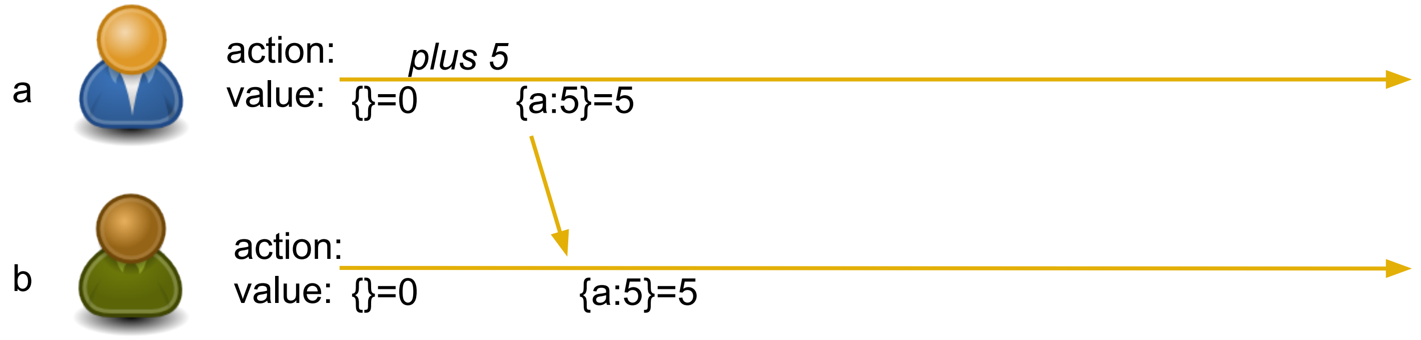
CRDTs

Operation-based CRDTs:

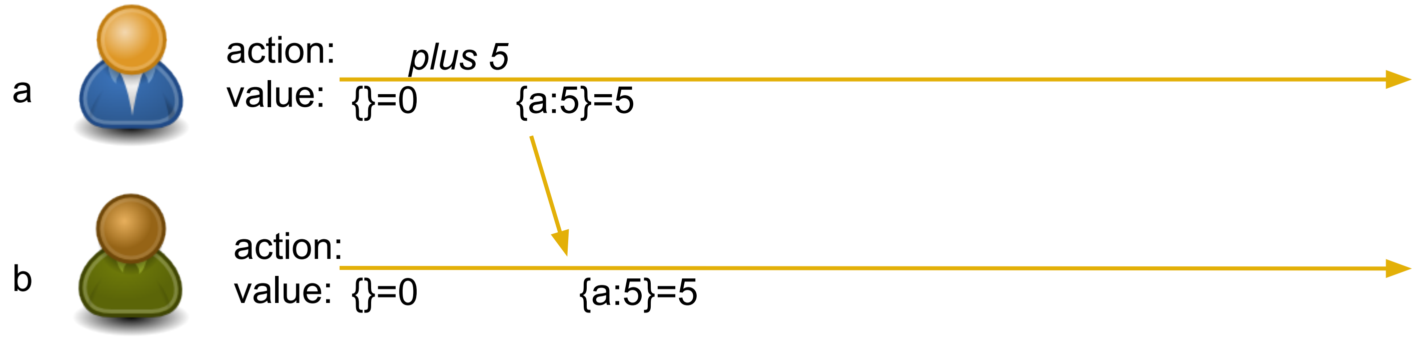
- All (concurrent) operations must *commute*
- Require *exactly-once* delivery semantics
- Require *in-order* delivery

Next up: State-based CRDTs

CRDT: State-based counter



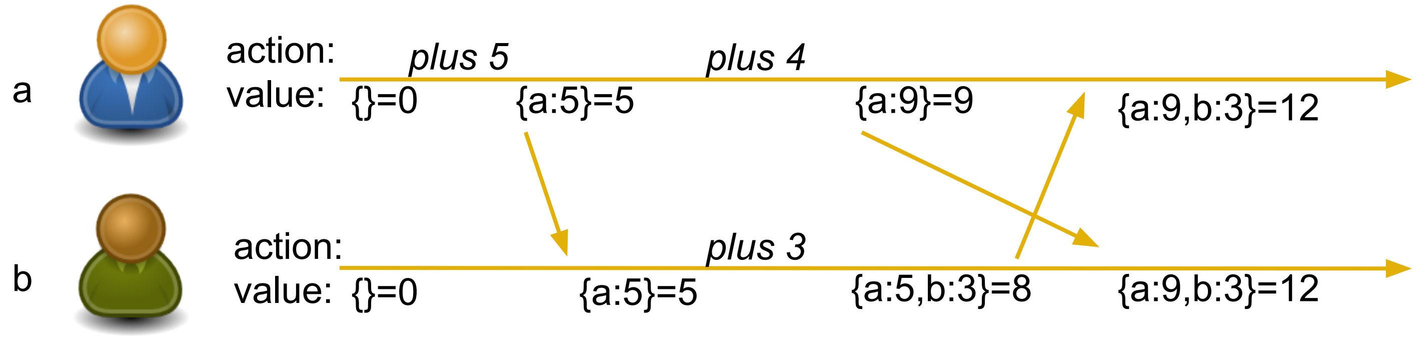
CRDT: State-based counter



State-based CRDT:

- Local *update*
- Send state and *merge*

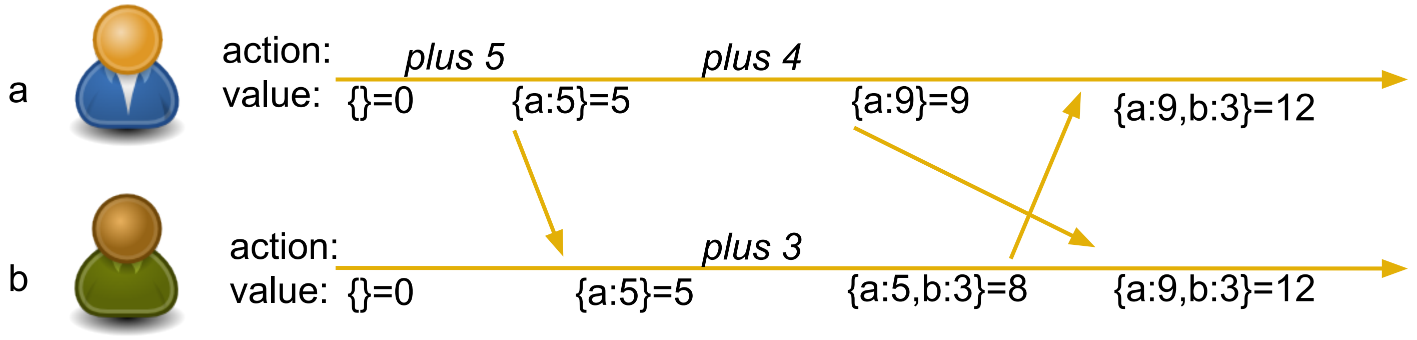
CRDT: State-based counter



State-based CRDT:

- Local *update*
- Send state and *merge*

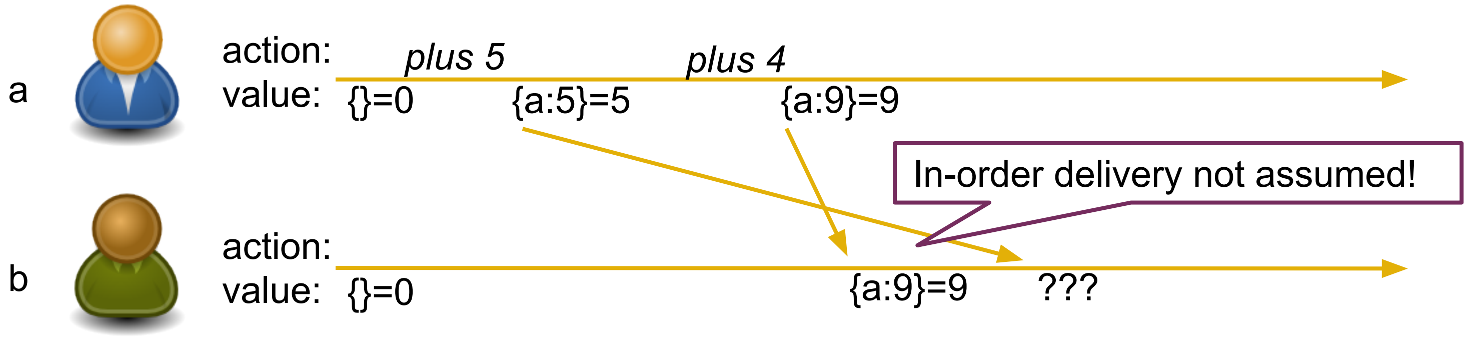
CRDT: State-based counter



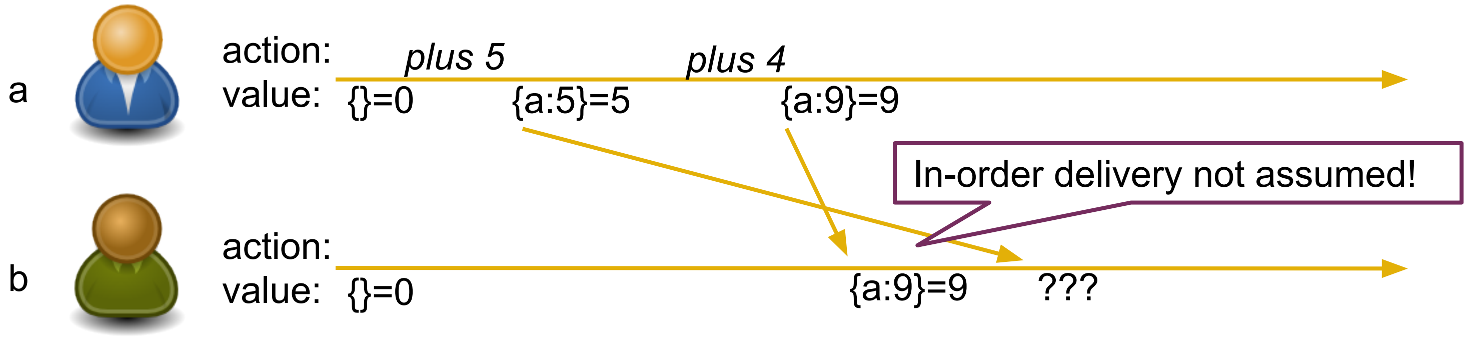
State-based CRDT rule 1:

- We allow retransmissions.
  - The merge function should be *idempotent*.

CRDT: State-based counter



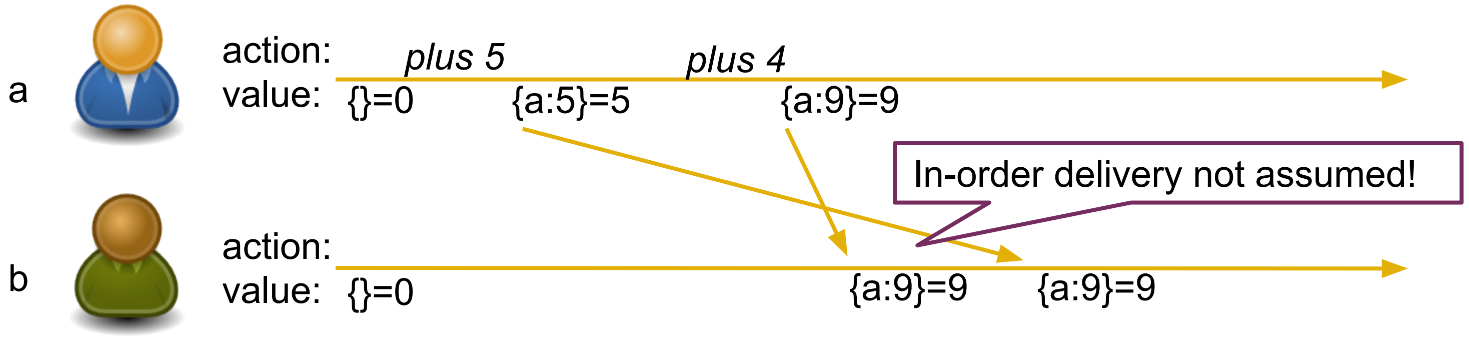
CRDT: State-based counter



State-based CRDT rule 2:

- Output must be independent of the order of merges
    - The merge function is *commutative* and *associative*

CRDT: State-based counter

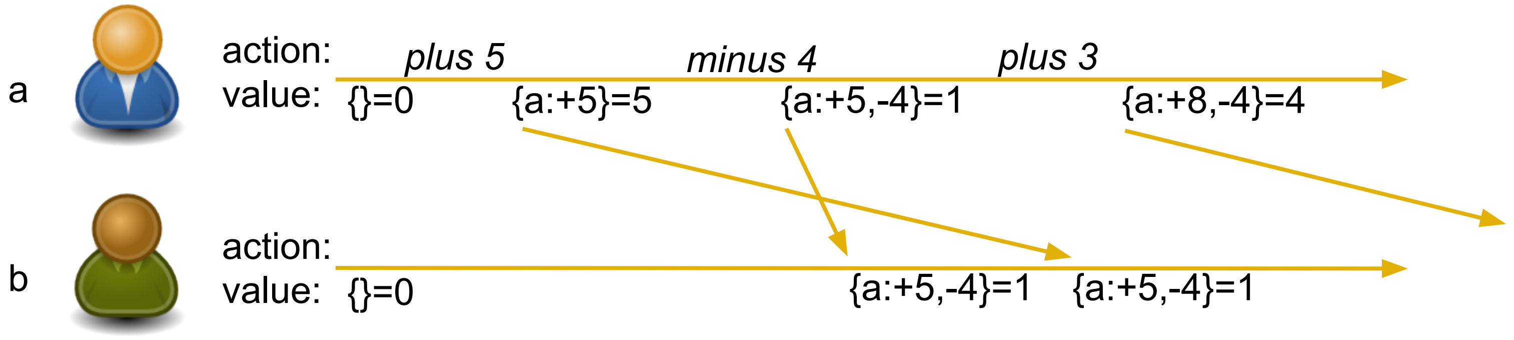


Our merge function here takes the ‘max’:

This is an *increment-only counter*.

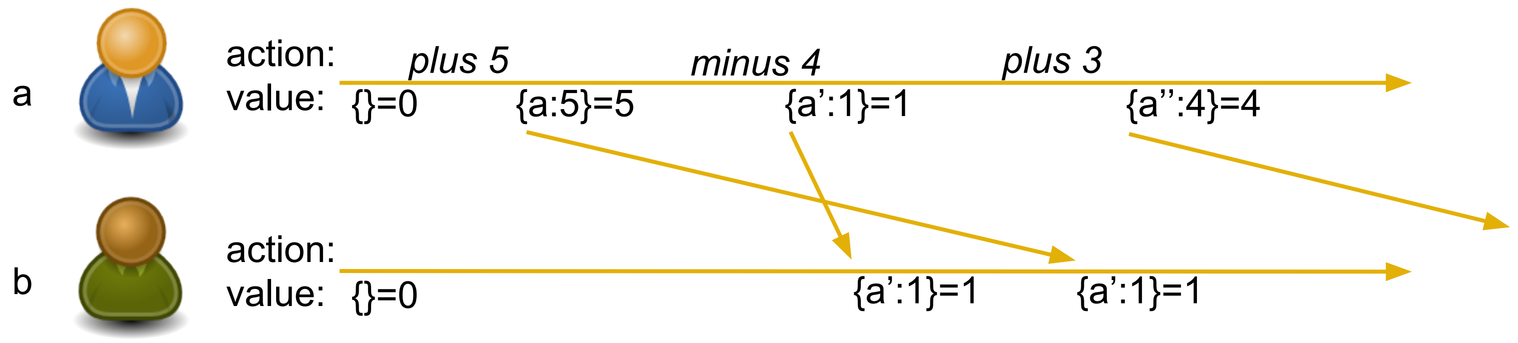
Xebia

CRDT: State-based counter



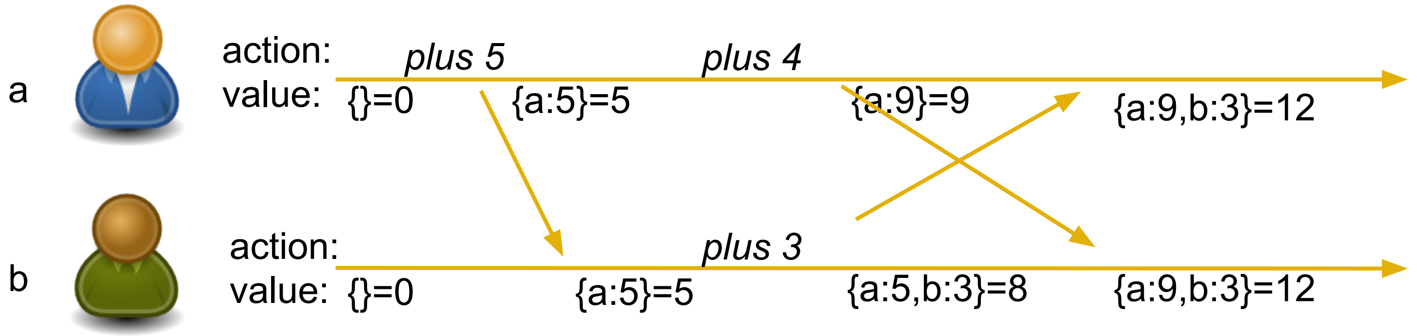
Solution 1: combine 2 counters
(positive and negative: 'PN-counter')

CRDT: State-based counter



Solution 2: version vectors

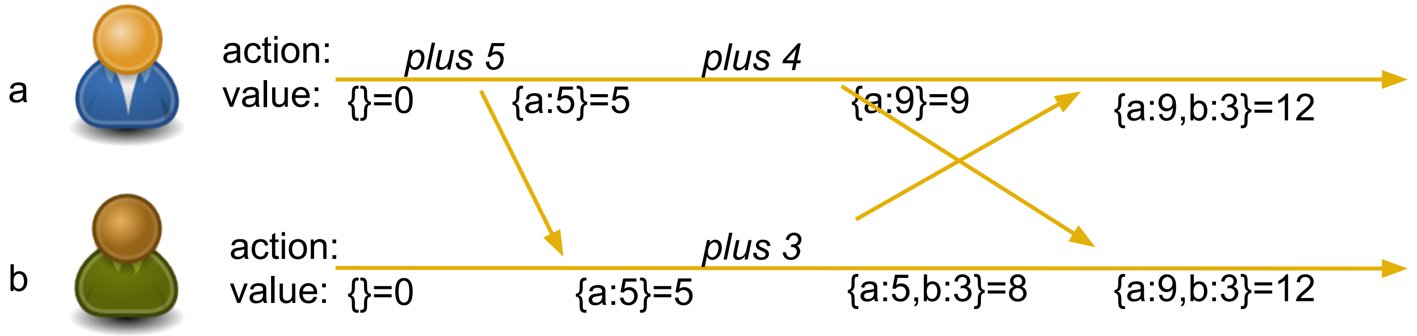
CRDT: State-based counter



State-based CRDT rules 3 and 4:

- We need a concept of ‘going forward’ (growing, clocks)
- Updates and merges always go forward
  - Concurrent states may not be comparable ({a:9} and {a:5,b:3})

CRDT: State-based counter



State-based CRDT rules 3 and 4:

- We need a concept of 'going forward' (growing, clocks)
- Updates and merges always go forward

In other words:

- There is a *partial order* on states
- Updates and merges must *increase* the state in this order

CRDTs: 2 kinds, 2 sets of rules

Operation-based CRDTs (Commutative, CmRDTs):

- All (concurrent) operations must be *commutative*
- Require *unique* and *in-order* delivery

State-based CRDTs (Convergent, CvRDTs):

- merge must be *idempotent*
- merge must be *commutative* and *associative*
- there exists a *partial order* on the states
- merge and update both *increase* the state along this order

DEMO

Conclusions

- CRDTs:
  - Have *simple rules*
  - Guarantee *Strong Eventual Consistency*
  - Can be *composed* to build more complex structures

  - Cannot model *every* data type
  - .. see if you can ‘bend your problem’

Further Material

- Watch Marc Shapiro's talks
  - (e.g. Strong Eventual Consistency at MS Research)
- Check out @cmeik's reading list
  - http://christophermeiklejohn.com/crdt/2014/07/22/readings-in-crdts.html
- Check out @cmeik's talk after lunch (Theater)

Questions?

Thank you!

-- Arnout Engelen, @raboofje