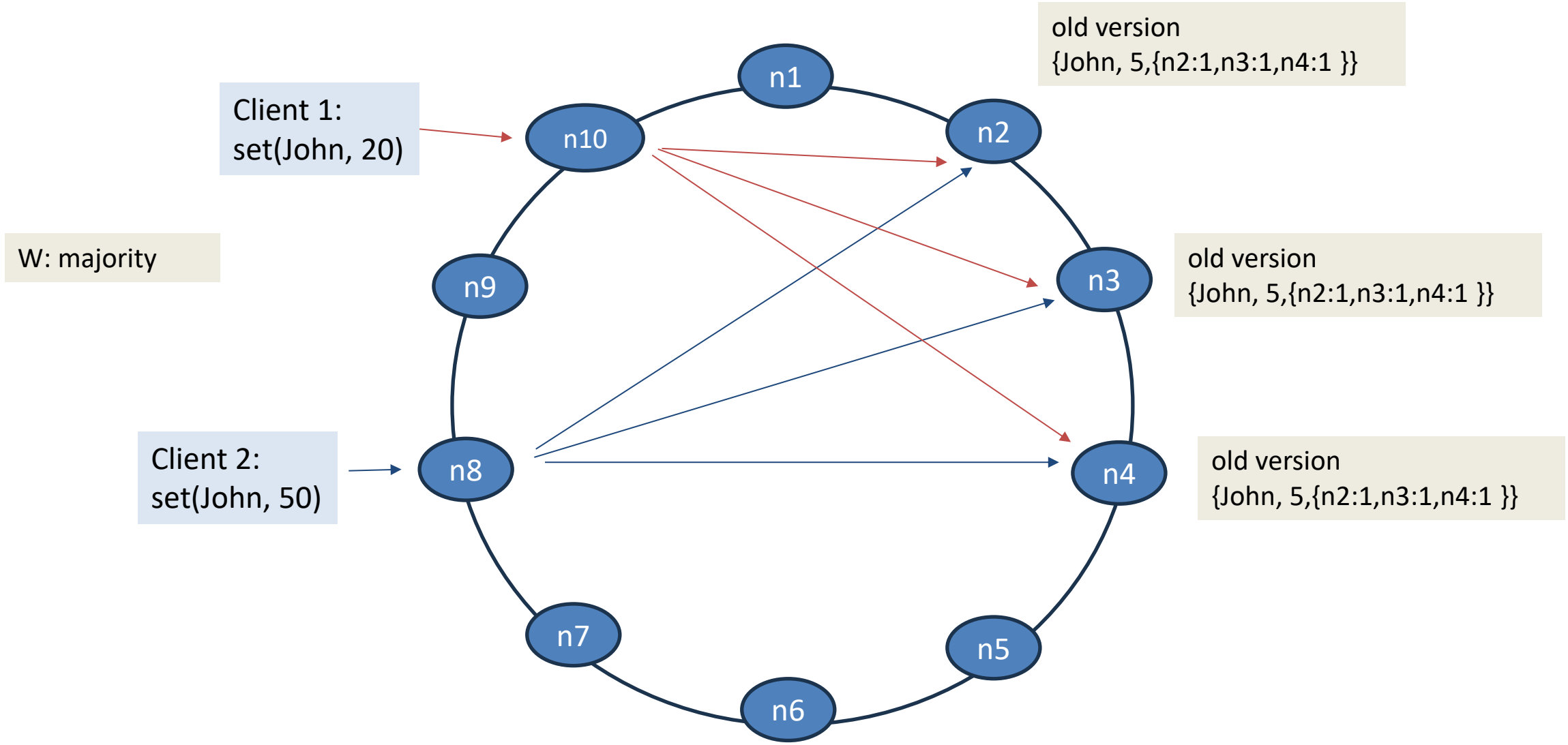
9.2
Distributed Databases
P2P Replication
Concurrent Write Conflicts, CRDTs

Reading: [Kl], chapter 5; [Ha] chapter 8
Dynamo Paper, 2007 (uploaded into TEAMS)
CRDTs 2011 (research paper, 2011, uploaded in TEAMS)

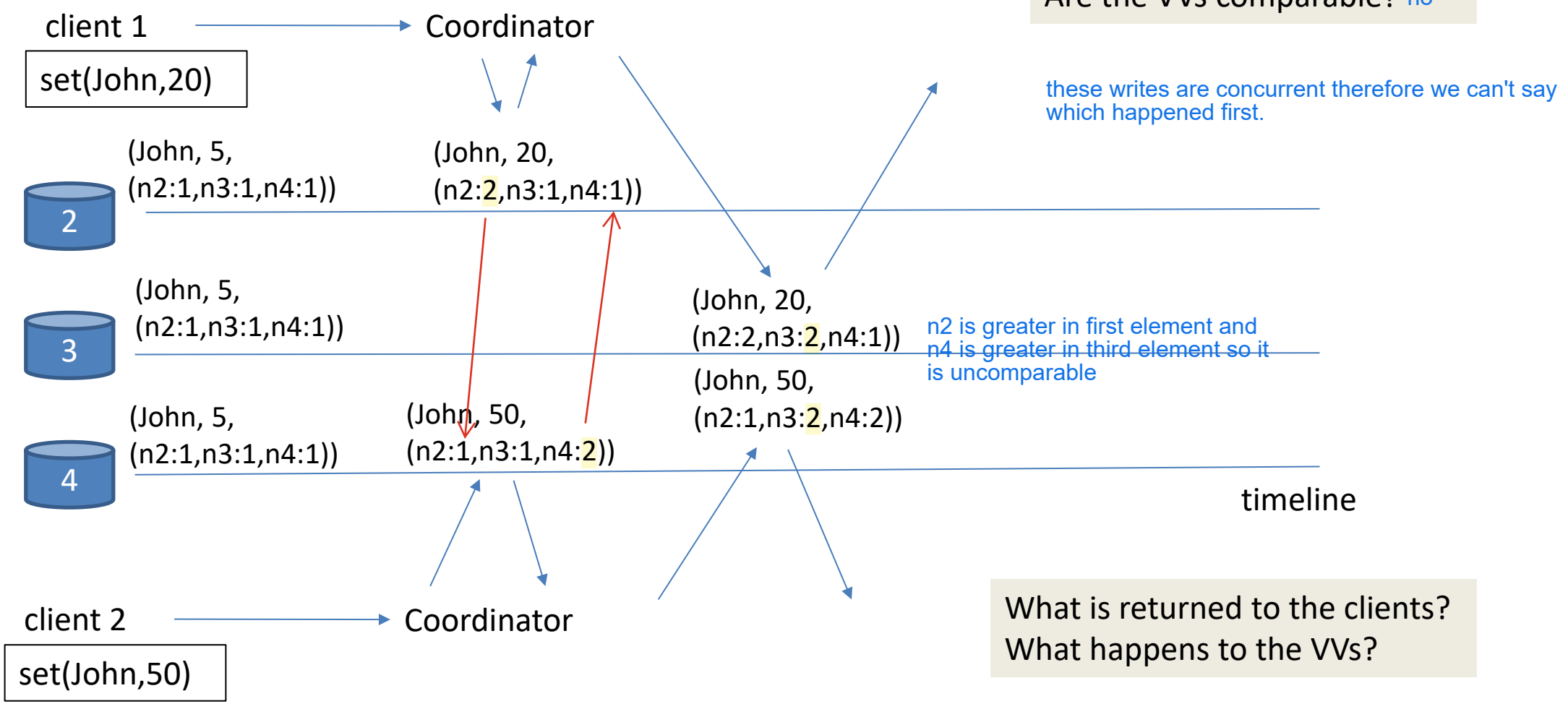
Concurrent Writes – P2P Distributed Databases



eva.knirsch@kiu.edu.ge

Concurrent Writes in P2P Distributed Databases

Are both writes successful? yes
Are the VVs comparable? no



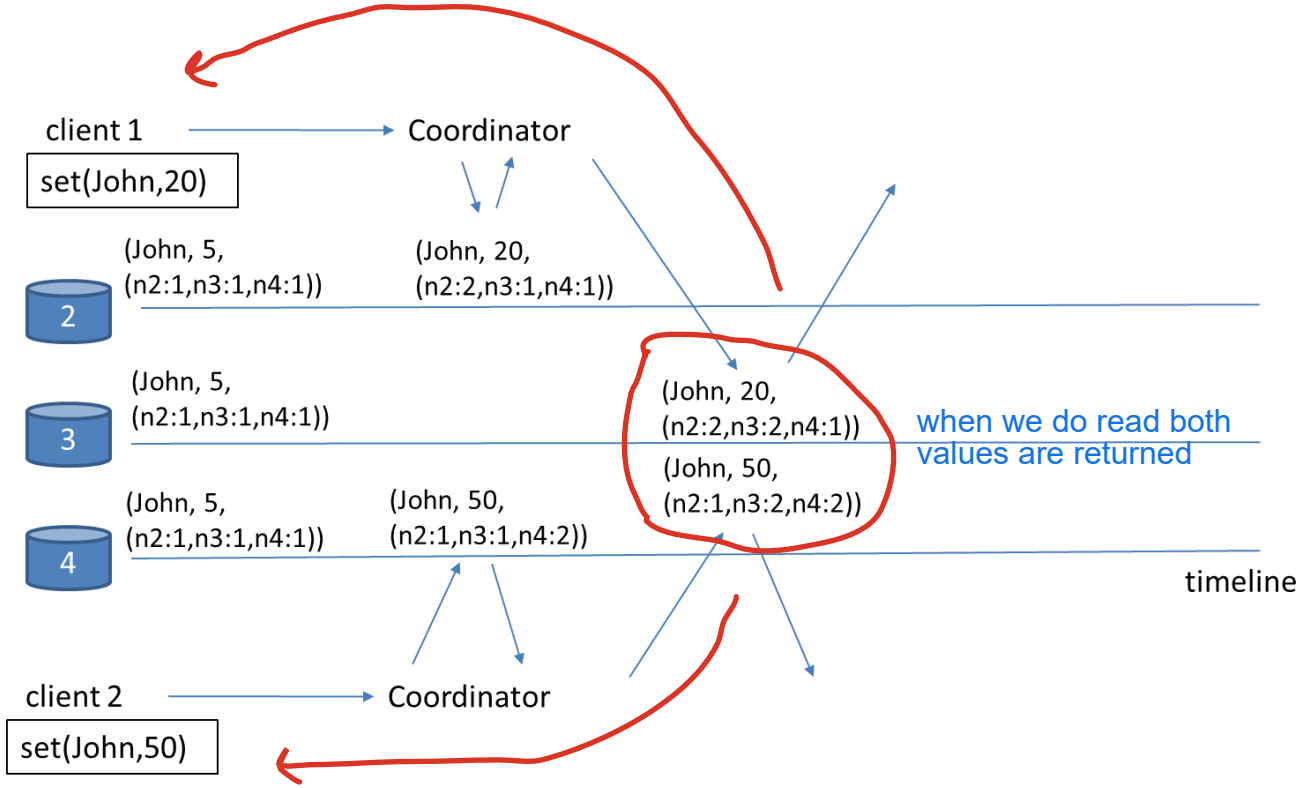
these writes are concurrent therefore we can't say which happened first.

n2 is greater in first element and n4 is greater in third element so it is uncomparable

What is returned to the clients?
What happens to the VVs?

eva.knirsch@kiu.edu.ge

Concurrent Writes Conflict Resolution in P2P Distributed Databases



VVs detect concurrent writes but cannot resolve them. Resolution depends on the system and / or setting
What is returned to the clients?
What happens to the VVs?

1. Siblings Values

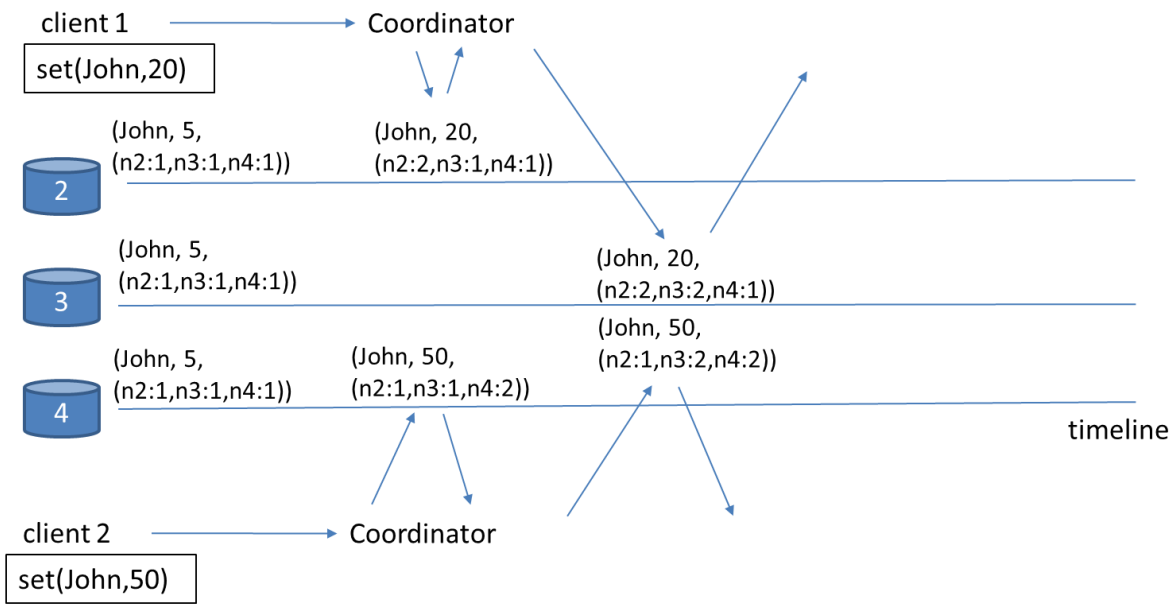
RIAK: .allow_mult(true)
→ both values are returned and both VVs are kept.
→ client / application needs to resolve and decide for one value.
→ hereafter the respective value / VV is kept

RIAK:
Concurrent writes — If two writes occur simultaneously from clients, Riak may not be able to choose a single value to store, in which case the object will be given a sibling. These writes could happen on the same node or on different nodes.

let's application decides for 50 then value 20 with its version vectors will be removed.

eva.knirsch@kiu.edu.ge

Concurrent Writes Conflict Resolution in P2P Distributed Databases



VVs detect concurrent writes but cannot resolve them.
What is returned to the clients?
What happens to the VVs?

2. LWW

RIAK: ..last_write_wins(true)) Riak accepts the last write based on timestamp and discards all others
https://docs.riak.com/riak/kv/latest/developing/usage/conflict-resolution/index.html
"This produces a so-called last-write-wins (LWW) strategy whereby Riak foregoes the use of all internal conflict resolution strategies when making writes, effectively disregarding all previous writes.
The problem with LWW is that it will necessarily drop some writes in the case of concurrent updates in the name of preventing sibling creation."

As the VV is part of the data, the VV of the value is also dropped.

eva.knirsch@kiu.edu.ge

CRDTs

CRDTs

- conflict free replication data structures,
- convergent replicted data types (RIAK),
- commutative replicated data types (REDIS)
- The CRDT concept was formally defined in 2011 by Marc Shapiro, Nuno Preguiça, Carlos Baquero and Marek Zawirski.
- data structures that fullfill certain conditions and do the merge automatically following inherent data type merge rules.

Wikipedia:

"a conflict-free replicated data type (CRDT) is a data structure that is replicated across multiple computers in a network, with the following features: order does not matter

The application can update any replica independently, concurrently and without coordinating with other replicas.

An algorithm (itself part of the data type) automatically resolves any inconsistencies that might occur.

Although replicas may have different state at any particular point in time, they are guaranteed to eventually converge."

CRDTs

https://docs.riak.com/riak/kv/2.2.3/learn/concepts/crdts/index.html#advantages-and-disadvantages-of-data-types

RIAK documentation

"Riak Data Types are operations-based from the standpoint of connecting clients.

Conflict resolution in Riak KV can be difficult because it involves reasoning about concurrency, eventual consistency, siblings, and other issues that many other databases don’t require you to consider.

One of the core purposes behind data types is to relieve developers using Riak KV of the burden of producing data convergence at the application level by absorbing a great deal of that complexity into Riak KV itself. Riak KV manages this complexity by building eventual consistency into the data types themselves instead of requiring clients to do so.

Conflicts between replicas are inevitable in a distributed system like Riak KV. ...

Without using data types, that conflict must be resolved using timestamps, vector clocks, dotted version vectors, or some other means. With data types, conflicts are resolved by Riak KV itself."

CRDT

a+b=b+a

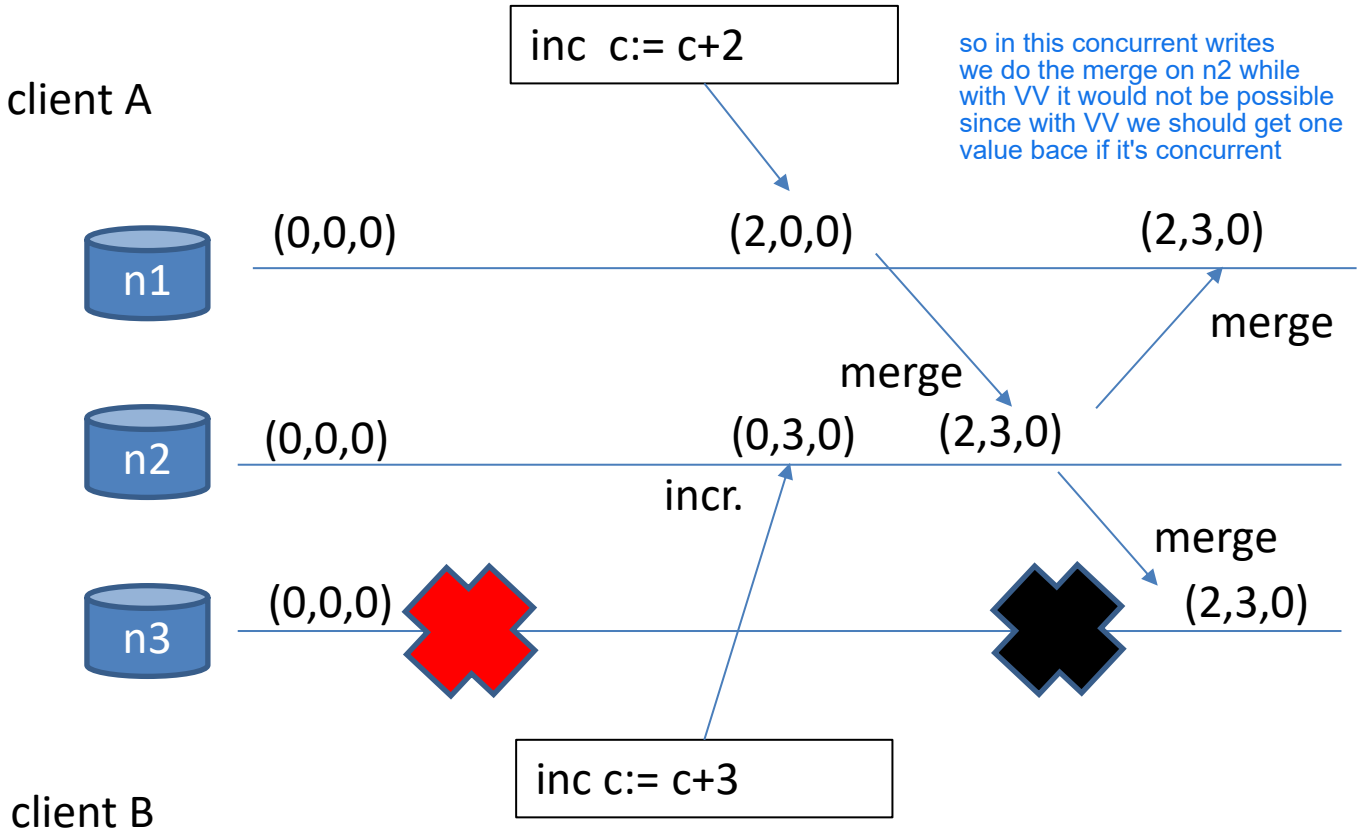
- A key approach is to reduce all writes to just commutative ones so that the order of the writes no longer matters. The merge function is part of the definition of the data structure.

- Example for a CRDT data type: Grow-Only Counter:
  - a state vector stores the increments done at each node for the object.
  - nodes replicate their state vectors so that eventually all nodes know about all increments.
  - The replication function (merge function) is commutative.

When two nodes exchange state vectors they perform a merge meaning taking the element-wise maximum of the two vectors because this ensures that no increments are double-counted and it reflects the highest known count from each node.

if node A has (5,0,0) and node B has (2,3,0) we do following: (max(5,2), max(0,3), max(0,0)) = (5, 3, 0) and it doesn't matter whether A merges B's state first or B merges A's becaues the result is always the same.

CRDT – Grow-Only Counter



Attention: Version Vectors and CRDT state vectors seem to be similar at first glance. However , they are not. Compare:

data structure with VV:

{key, value, VersionVector}

{key, value, map(nodeID, counter)}

VV: a timeline that captures "happened before" / happened after" to version updates.

data structure with CRDT:

{key, CRDT Vector}

{key, map(nodeID, counter)} counter owns the values

CRDT state vector: Captures the values: What values have been entered into the system? Time and causality ("happened before / after) are not important. CRDTs track the state of the data values.

Each node holds a state vector consisting of the counter value of its own and all other nodes, here: (0,0,0). Each node is only allowed to update its own value in the vector. The merge function merges the state of 2 nodes. The merge function takes the maximum value of each slot. The merge function is commutative. The current vector state is determined by summing up the values of all slots, here sum(2,0,3) = 5 if you do the read then all values of all slots are summed up and order does not matter.

eva.knirsch@kiu.edu.ge

State-Based CRDT

summing up

State CRDTs consist of a state application algorithms and a replication protocol. The merge method is the most important operartion of state CRDTs. The merge methods has the following three properties:

1. commutative a+b=b+a
2. associative (a+b)+c= a+(b+c)
3. idempotent operation is said to be idempotent if it can be repeated any number of times with the same result

The replication protocol only needs to satisfy the requirement that eventually all state updates are delivered to all replica nodes.

With state-based CRDTs, the client doesn't pass on the operation(s) it applied to change the data. Instead, it simply sends the new state of the data to all other nodes, that is it merges its state with the state of other nodes.

In operation-based CRDTs the update operation is propagated and applied locally. There are some delivery requirements to ensure that every update operation is transmitted just once to a replica because, in some formulations of CRDTs, operations are not always idempotent. Once they arrive at the replica, like state-based CRDTs, they can be applied in any order.

Strong Eventual Consistency

- Strong Eventual Consistency (SEC)
- An object satisfies 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.
  (INRIA research, report RR 7687).

Summary Replication

- Single leader replication:
  - The primary node accepts writes and determines the order of writes. Secondaries apply changes. Failover process.
- Peer-to-peer replication:
  - Available replicas accept writes. Unavailable replicas miss writes. Operations are successful if they satisfy the quorum.
  - No failover
  - Concurrent write conflicts.
  - Eventual Consistency: eventually each node sees every operation. Eventually all nodes have the same state.
  - Strong eventual consistency: Using data structures that guarantee a conflict-resolution merge.