9.1
Distributed Databases
Peer-to-Peer Replication

Reading: [Kl], chapter 5; [Ha] chapter 8
Dynamo Paper, 2007 (uploaded into TEAMS)
https://martin.kleppmann.com/2020/11/18/distributed-systems-and-elliptic-curves.html

ACID Principle

Databases that support the ACID characteristics are considered transactional databases.

CAP consistency and ACID consistency are different

Atomicity:	**All-or-Nothing Principle, either all commands succeed or no command succeeds**
Consistency:	**consistent state1 db → consistent state2 db, all constraints kept**
Isolation:	**as if each transaction had db for itself, serializability**
Durability:	**Writes of transactions are persistent, no loss of data**

CAP Theorem

Eric Brewer introduced the CAP theorem in 2000. In 2002 it was mathematically proven.
It applies to distributed, replicated systems.

CAP describes three desirable properties of distributed systems:

– Consistency (C),
– Availability (A),
– Tolerance to network failure (P)

P is for Partition tolerance which means that the system continues to operate even if messages between some nodes are delayed or lost. network partition happens when parts of the distributed system can't communicate with each other due to a failure. Each partition can still work independently, but they are cut off from each other.

- Tolerance to network failure : The system continues to operate and to respond correctly even if the network connecting the nodes has a fault.
  Problem: Network failures are beyond the control of any database system. They simply happen.
- Availablity: Every request receives a non-error response in a reasonable amount of time.
- Consistency: Clients have the same view of the data. Every node returns the same, most recent data.
  (Note that this definition is different from the consistency definition in ACID!

in ACID we just have a lot of rules for our database like FKs, PKs, ... which must be reserved between state.
In CAP all applications can write to different nodes which may temporarily lead to different values on different nodes. However, eventually all nodes should return the same and most recent value. Therefore it doesn't matter how many applications write to the system, in the end nodes agree on same value.

CAP Theorem

Relational database lesson – provide an ACID consistency example:

Key-Value database "Teachers"

key: unique teacher_name

value: information about the teacher

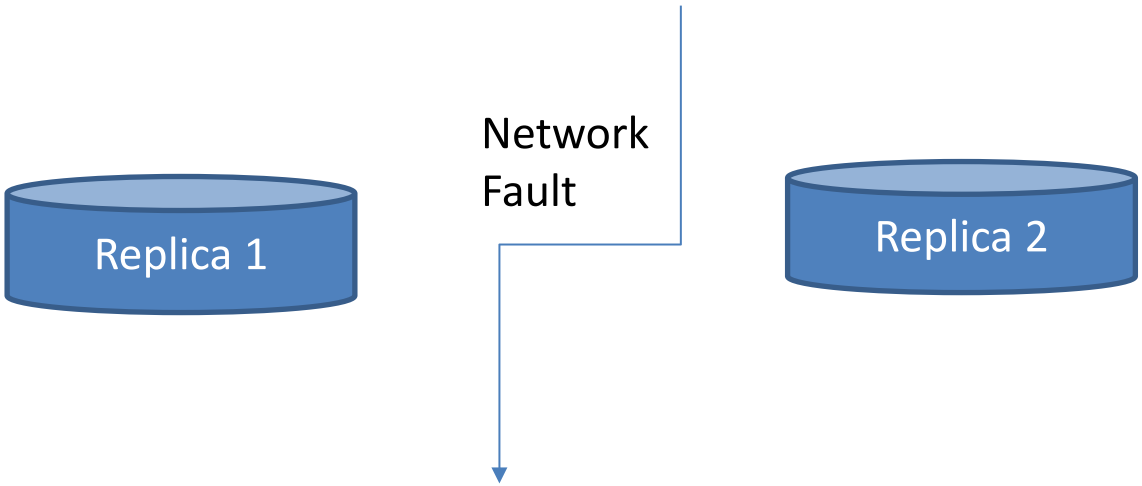
Provide a CAP Consistency example:

The CAP theorem says that a distributed database system can achieve at most only two of the three characteristics.

Does this mean: Pick two out of three? No because P is always given, it is not under our control so we either pick C/P or A/P

CAP Theorem

What are the choices in case of a network failure?



when designing application, if we prioritize consistency over availability we choose C/P, otherwise A/P

Consistency is prioritized – C/P: RDBMS, MongoDB, Google Spanner, HBase → prioritize consistency

Availability is prioritized – A/P: Cassandra, RIAK, DynamoDB, Voldemort → prioritize availability

eva.knirsch@kiu.edu.ge

KUTAISI INTERNATIONAL UNIVERSITY

CAP and Eventual Consistency

Linearizability --> Serializability --> Causal Consistency --> eventual concsistency

STRONG WEAK

Eventual consistency is a weak consistency guarantee:
All replicas converge to the same state and after some time interval (eventually) all replicas have the same data.
→ Eventually, a read will render the same result on all nodes.

eventual consistensy

It is used for applications that

- demand high availability (even if there are network partitions or node outages)
- low latency minimal delay or time taken for a system to process and respond to a request
- can tolerate stale reads and write conflict solution.

Causal consistency ensures that operations that are causally related meaning one happens as a result of another are seen by all nodes in the same order.

causal consistency is what MongoDB offers

Dynamo Database as Research Project

Dynamo was a research project. It experimented with – at that time – completely new concepts for distributed replicated databases on the basis of the CAP theorem. Dynamo was never developed into a commercial database.

It cleary prioritized availability over consistency.

Dynamo paper, 2.1:

".....data stores that provide ACID guarantees tend to have poor availability. ..... Dynamo targets applications that operate with weaker consistency (the "C" in ACID) if this results in high availability. Dynamo does not provide any isolation guarantees and permits only single key updates. "

Dynamo concepts were summarized in the Dynamo paper of 2007.

Todays Peer-to-Peer distributed databases typically implement concepts of the Dynamo paper.

DynamoDB implements many Dynamo concepts

KUTAISI INTERNATIONAL UNIVERSITY

Peer-to-Peer Distributed Databases

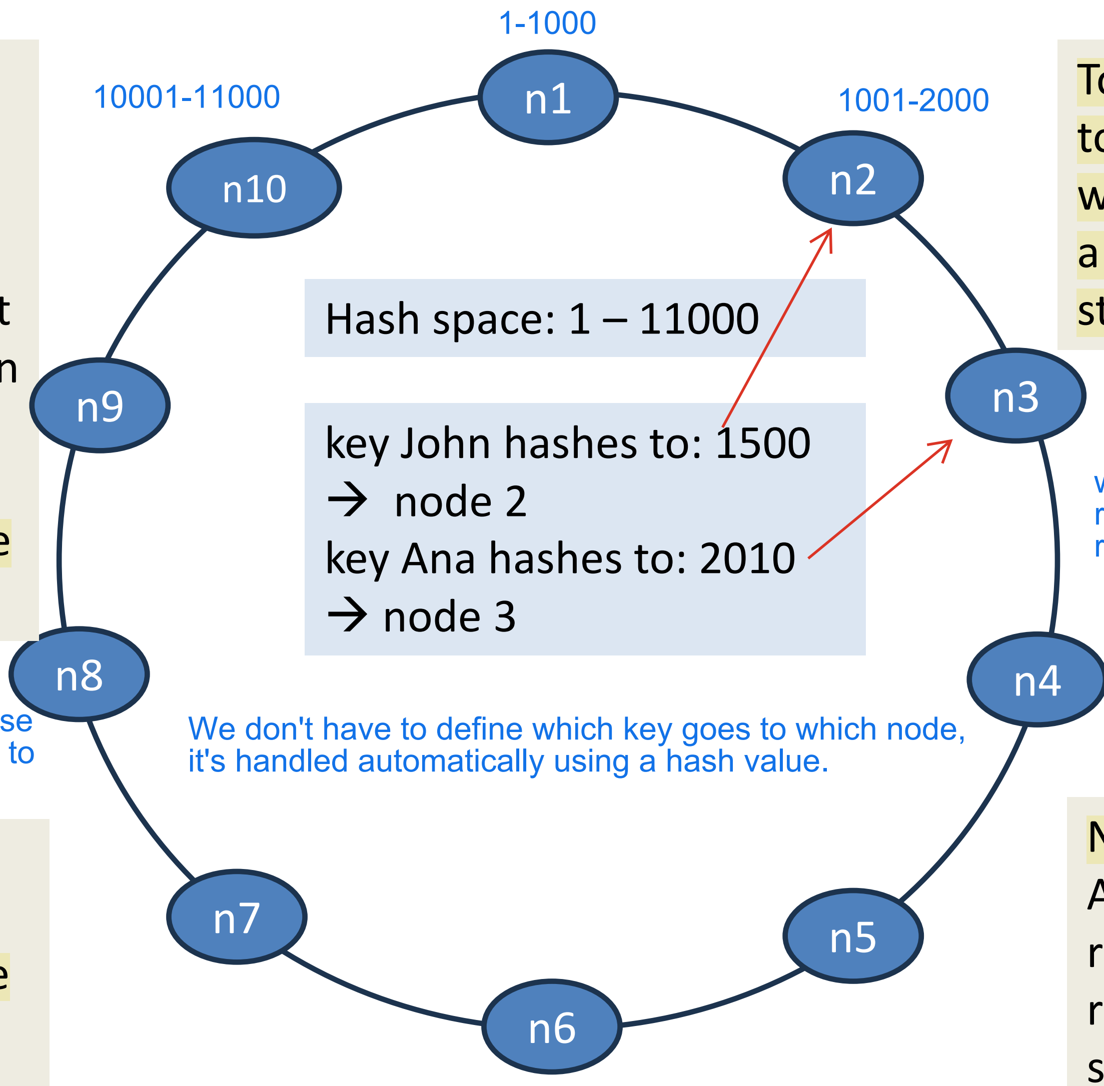
Partitioning

data itself is partitioned on different nodes

All server nodes form a virtual token ring: output range of a hash function that is used to partition the data.
Each server node holds a hash value range

hash space covers all of the nodes. so in our case hash space goes from n1 to n10

Each data item (key) is hashed and stored on the node that holds this hash range.



We don't have to define which key goes to which node, it's handled automatically using a hash value.

Token ring is used to decide on which server node a data item will be stored.

we can't place more onto the ring than our output range renders.

No primary node.
All nodes serve read / write requests and share same responsibility

KUTAISI INTERNATIONAL UNIVERSITY

Cassandra Peer-to-Peer Distributed Databases Partitioning

Cassandra:

- Cassandra uses
  - token ring to distribute data
  - LSM to store data.
- We assume that the key John hashes to 1500 and is assigned to server node2 for storage.
- Data item with key John is passed to server node2.
- At server node 2, the data item has to be stored.
- What happens at server node2?

When a key is passed to node2, it is placed in the memtable. Over time the memtable is flushed to disk as an SSTable. Supporting structures such as bloom filters, file segments and sparse indexes are then generated to speed up future reads.

Replication
Single Leader Replicated Database

MongoDB: number of replicas are defined in rs.initiate()

```
> rs.initiate({
  _id: "rs1",
  members: [
    { _id: 0, host: "localhost:27018" },
    { _id: 1, host: "localhost:27019" },
    { _id: 2, host: "localhost:27020" }   ] })
```

in replica set we specify how many nodes we want to be in there

KUTAISI INTERNATIONAL UNIVERSITY

Replication in Peer-to-Peer Distributed Databases

Using a token ring architecture, replicas (copies) are NOT stored on all nodes. When creating a database keyspace (namespace), one has to determine the number of replicas (copies)

RIAK:

```
Creating a bucket (Riak version 1.x):
Bucket bucket = connection .createBucket(bucketName)
.allow_mult(true / false)
.n_Val(numberOfReplicationCopies, default 3)
.last_write_wins(default false)
.w(numberOfNodesToRespondToWrite) .r(numberOfNodesToRespondToRead)
.execute();
```

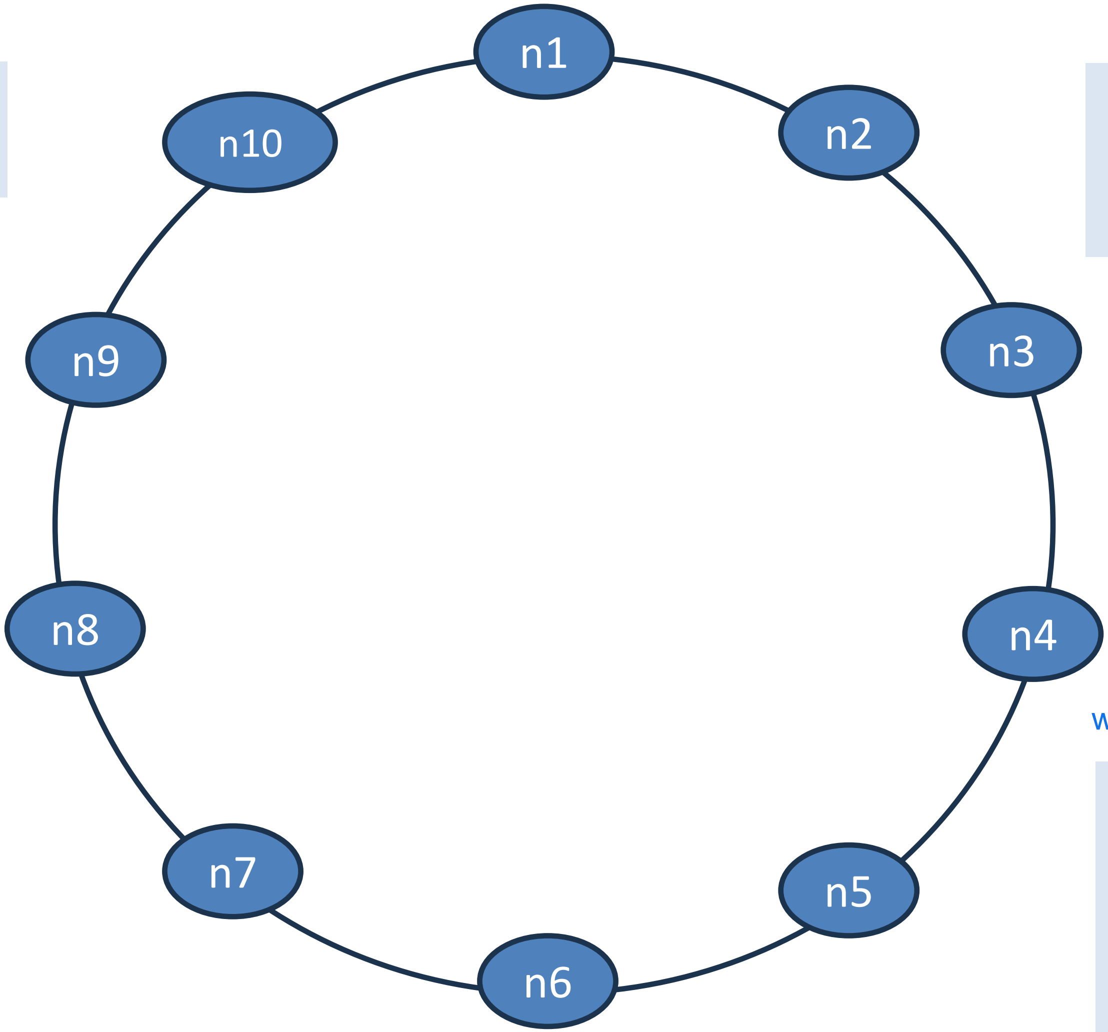
Cassandra

```
CREATE KEYSPACE lesson
 WITH replication = {'class': 'SimpleStrategy', 'replication_factor' : 3};
```

this means that if we have 'Ana' stored on node3, it will also be stored on node4 and node5(in total of 3 nodes)

Peer-to-Peer Distributed Databases

Client 1:
set(John, 20)



rf: 3
(John, 20)
→ n2, n3, n4

we don't have a primary.

no oplog that propagates writes and synchronizes replicas

eva.knirsch@kiu.edu.ge

Peer-to-Peer Replication Mechanisms

- How are the writes onto the replicas coordinated?
- When is a write / read successful?
- What happens if one or more nodes are down?
- What happens if nodes carry different values for the same object?
  - How does the system detect divergent values?
  - How are the values eventually synchronized?
  - What are the synchronization mechanims?

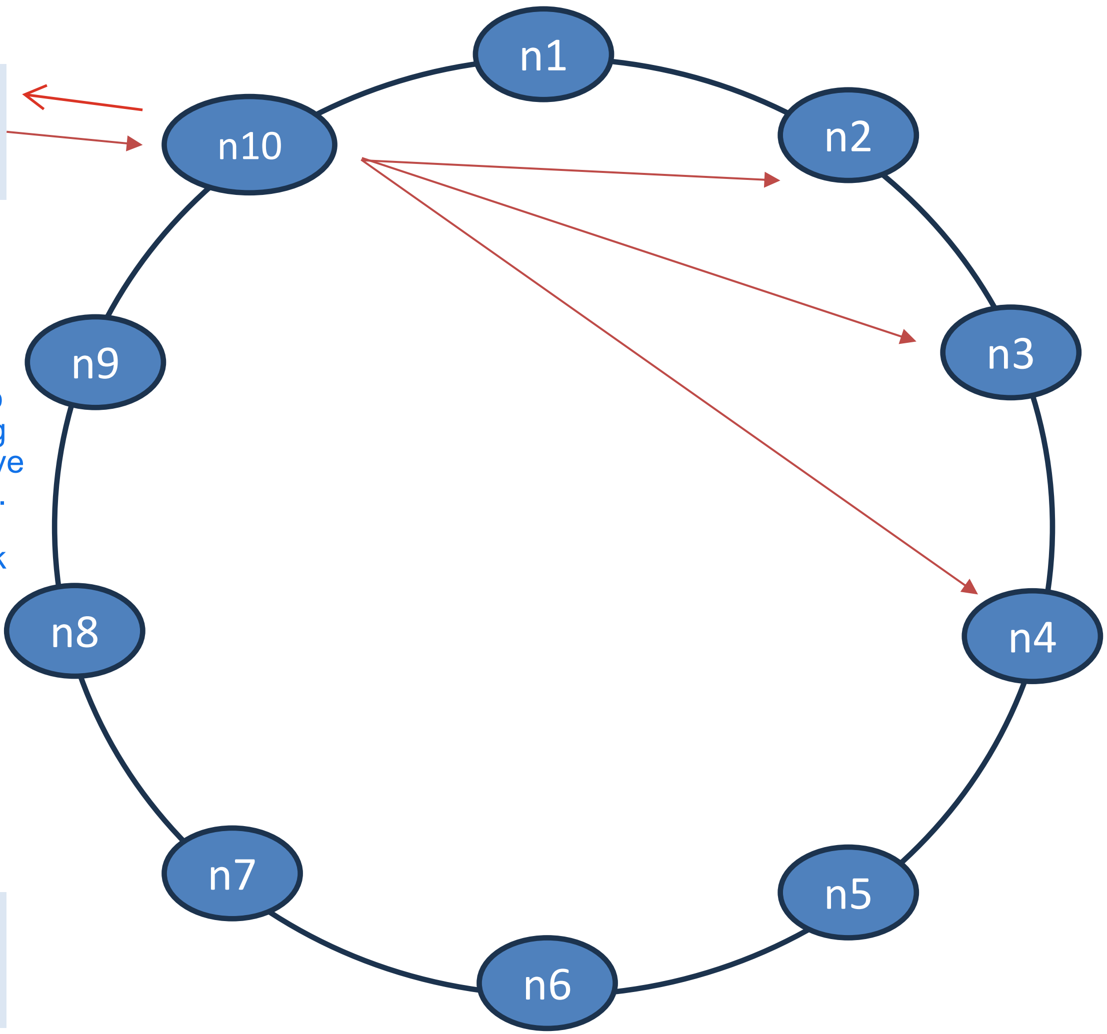
Peer-to-Peer Replication Mechanisms

- Coordinator nodes
- Write and read quorums
- Timestamps, Vector Clocks and Version Vectors
- Read Repair
- Anti-Entropy Mechanisms (e.g. Merkle Trees)

Distributed p2p databases typically use these concepts. Not all databases implement all concepts and implementations vary considerably.

Peer-to-Peer Distributed Databases

Client 1:
set(John, 20)



client can send write request to any node and the node they address becomes coordinator, so it is responsible for doing writes but it does not have to do writes in sequence. Coordinator is also responsible to send back acknowledgement to the client.

n10:
Cooordinator

Peer-to-Peer Replication Mechanisms

- Coordinator nodes
  - a client request (read or write) can go to any node
  - the node that receives the client request becomes the coordinator for that request.
  - It is responsible for executing the request according to the rules.

Example:

write request of client1 goes to node10. → node10 is coordinator for the write.

read request of client2 goes to node4. → node4 is coordinator for the read.

Client1

set(John, 20)

Client2

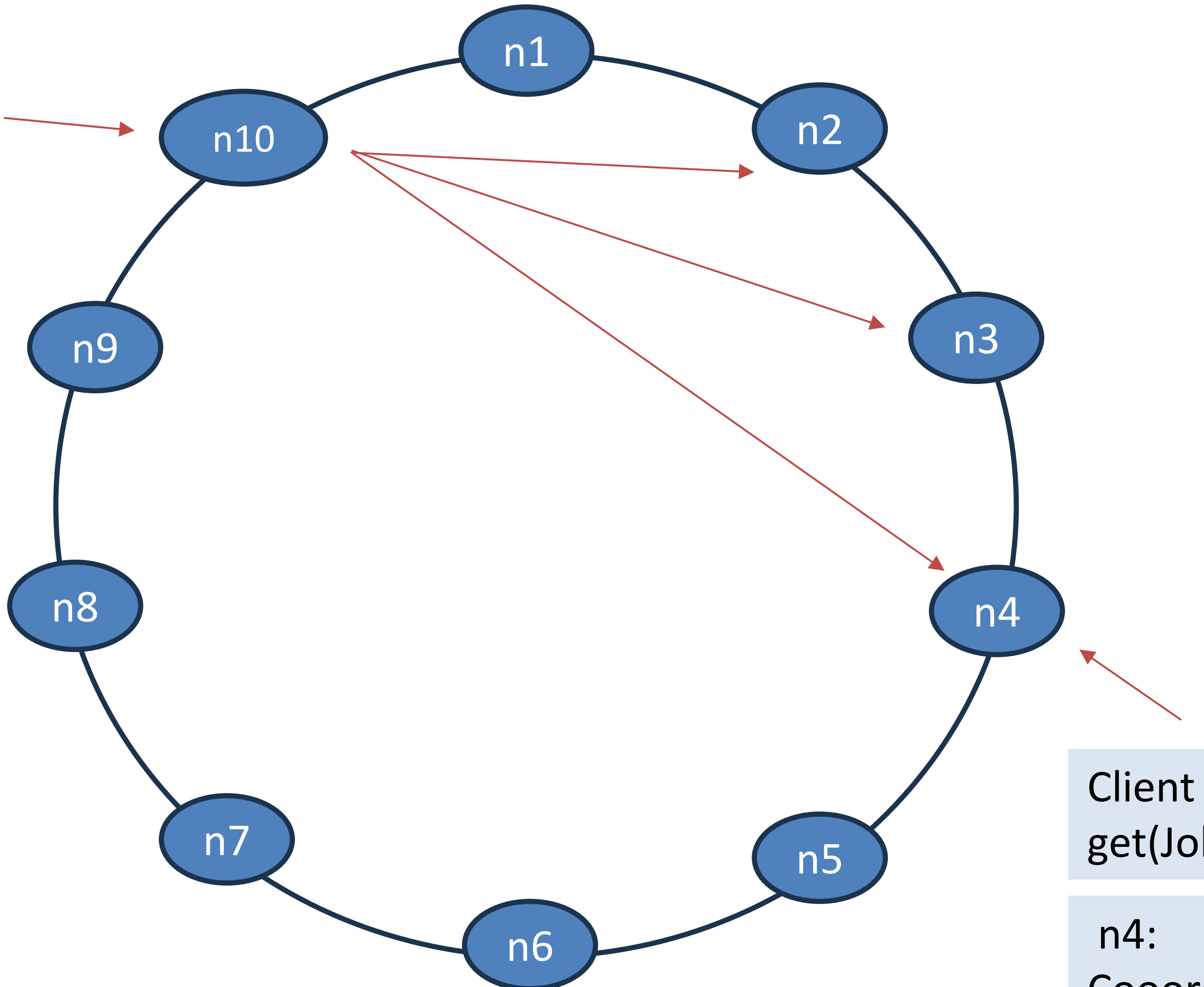
get(John)

The data of write requests are not necessarily stored on the coordinator node itself but on the nodes that the partitioning hash and the replication factor assign.

Peer-to-Peer Distributed Databases

Client 1:
set(John, 20)

n10:
Cooordinator



Client 2:
get(John)

n4:
Cooordinator

Peer-to-Peer Replication Concepts

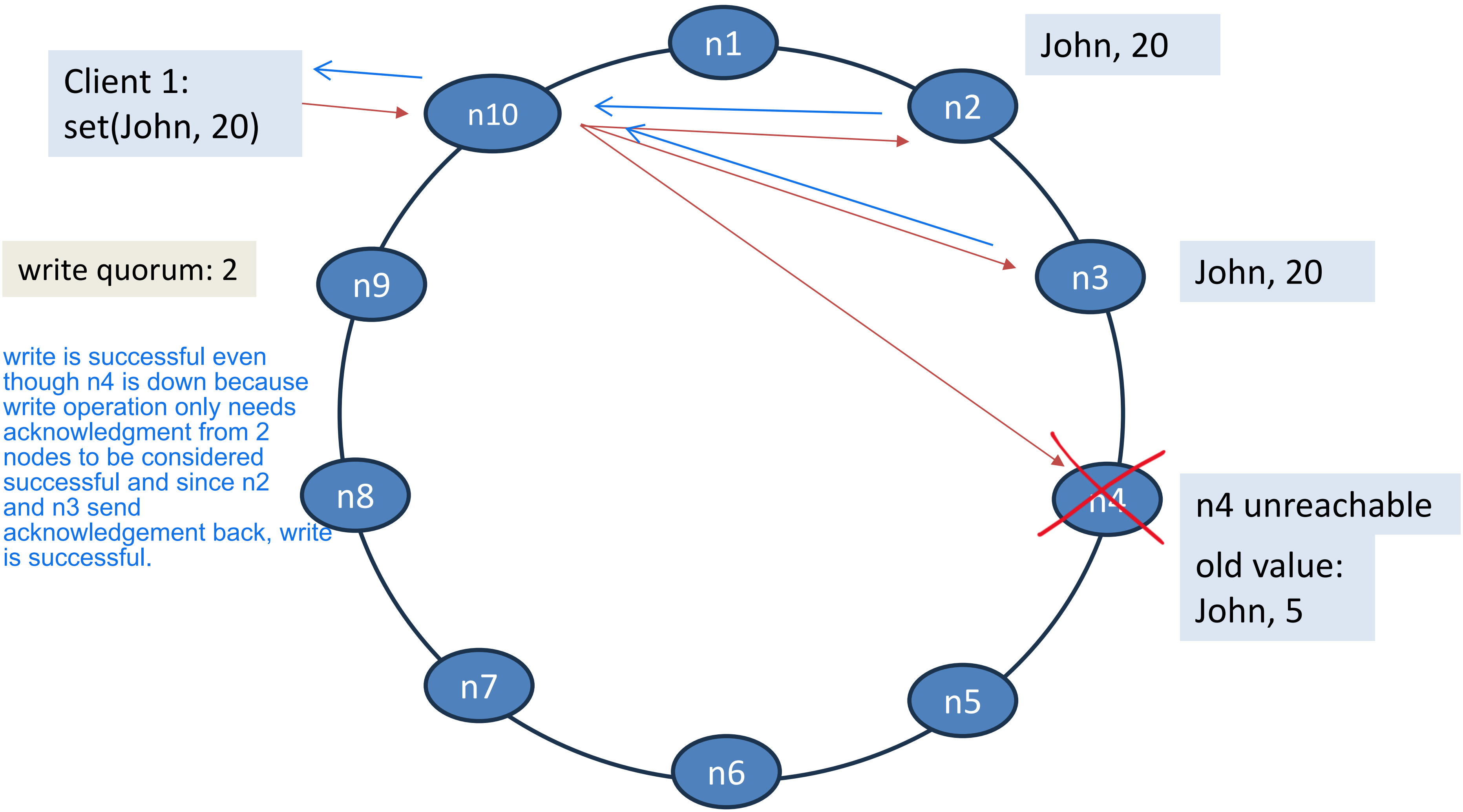
When is a write successful – acknowledged back to the client?

When is a read successful – acknowledged back to the client?

- Write and read quorums similar to write and read concerns in MongoDB
  - w quorum: a subset of replicas that must apply the write request in order for it to be acknowledged (successful)
  - r quorum: a a subset of replicas that must respond to a read request in order for it to be acknowledged (successful)it is not necessery to get same value when reading to be successful
  - write and read quorums are always given as number of nodes.
  - If a write or read does not get the quorum, it is considered to be failed and the write or read operation returns an error.
  - Coordinator node is responsible for acknowledgement / failure retun to client.

[Kl], p. 178

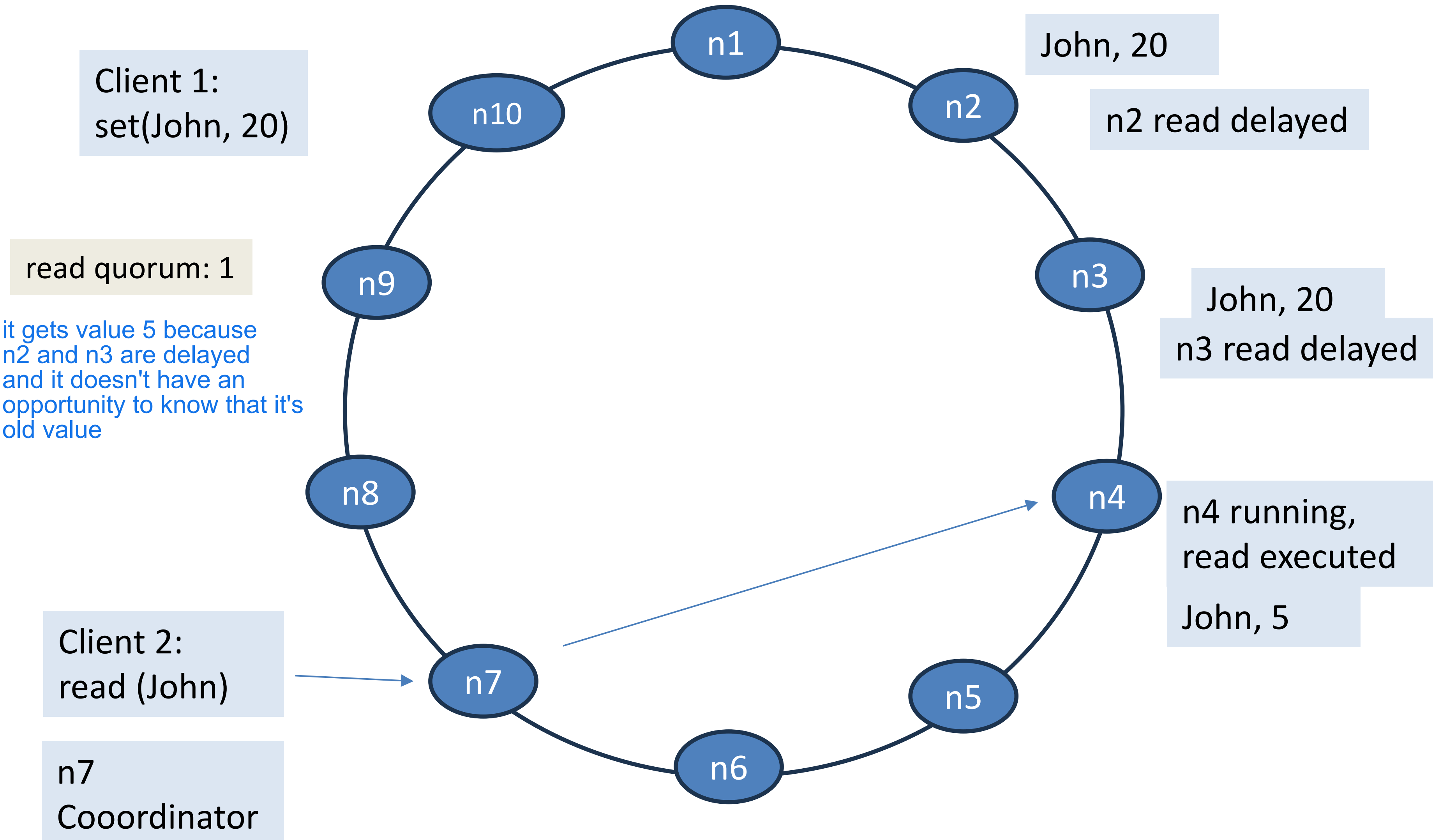
Peer-to-Peer Distributed Databases



write is successful even though n4 is down because write operation only needs acknowledgment from 2 nodes to be considered successful and since n2 and n3 send acknowledgement back, write is successful.

eva.knirsch@kiu.edu.ge

Peer-to-Peer Distributed Databases



it gets value 5 because n2 and n3 are delayed and it doesn't have an opportunity to know that it's old value

eva.knirsch@kiu.edu.ge

Peer-to-Peer Distributed Databases

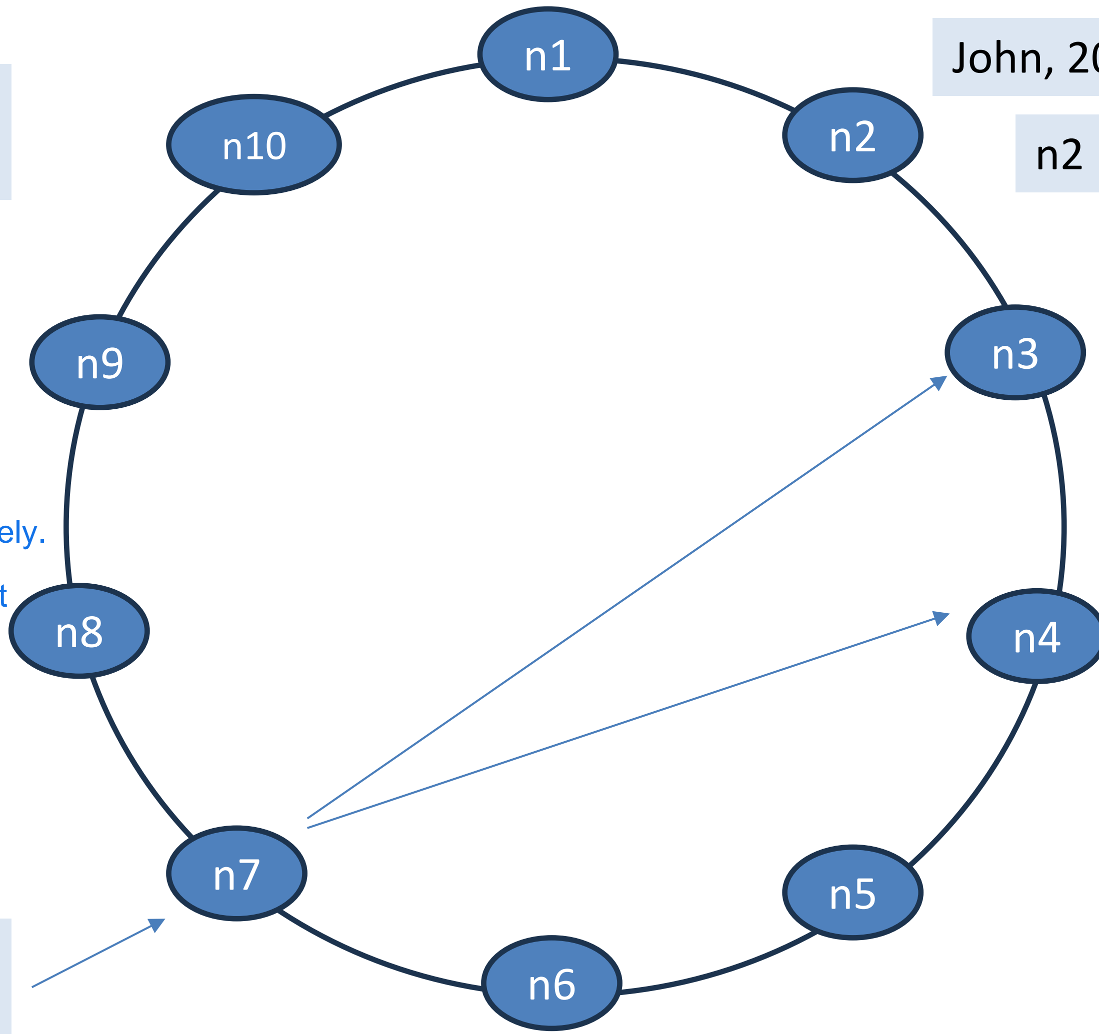
Client 1:
set(John, 20)

John, 20

n2 read delayed

read quorum: 2

since n2 is delayes and read quorum is 2 we need to read from 2 nodes and we get back values 20 and 5 from n3 and n4 respectively. At this time there is no way to find which is more recent because we don't have oplog.



John, 20

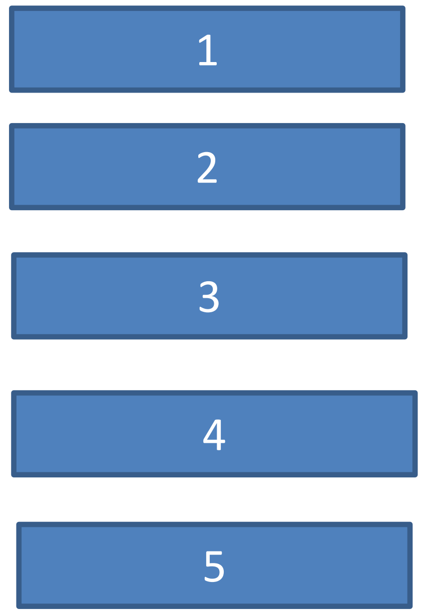
John, 5

Client 2:
read (John)

Write and Read Quorum Replication

Example 1
N: 5
W: 2
R: 2

Example 2
N: 5
W: 3
R: 3



How many nodes can be down without interrupting availability?

Example 1: 3

Example2: 2

Can you get stale reads **only**?

Example 1: yes. For example we write to node 1, 2 and we read from nodes 4, 5 so nodes 4 and 5 did not get the new data yet. So they return old data. So the read is not guaranteed to be fresh.

Example2: no, because we have at least one overlapping node with most recent data. for example we write to nodes 1, 2, 3 and then we read from nodes 3, 4, 5. Both sets include node 3. That guarantees that the read will include the latest data from at least one node.

KUTAISI INTERNATIONAL UNIVERSITY

Write and Read Quorum Replication

Example 1
N: 5
W: 2
R: 2

Example 2
N: 5
W: 3
R: 3

1

2

3

4

5

Example 3
N: 5
W: 4
R: 2

Example 4
N: 5
W: 5
R: 1

not recommended because it contradicts the concept of replication. meaning replication is used so your data survives if some nodes fail. But here we treat all nodes as required for a write, so there's no tolerance for failure.

How many nodes can be down without interrupting availability?

Example 3: 1 Example4: none

Can you get stale reads **only**?

Example 3: no Example4: no

eva.knirsch@kiu.edu.ge

KUTAISI INTERNATIONAL UNIVERSITY

Write-Read Quorums

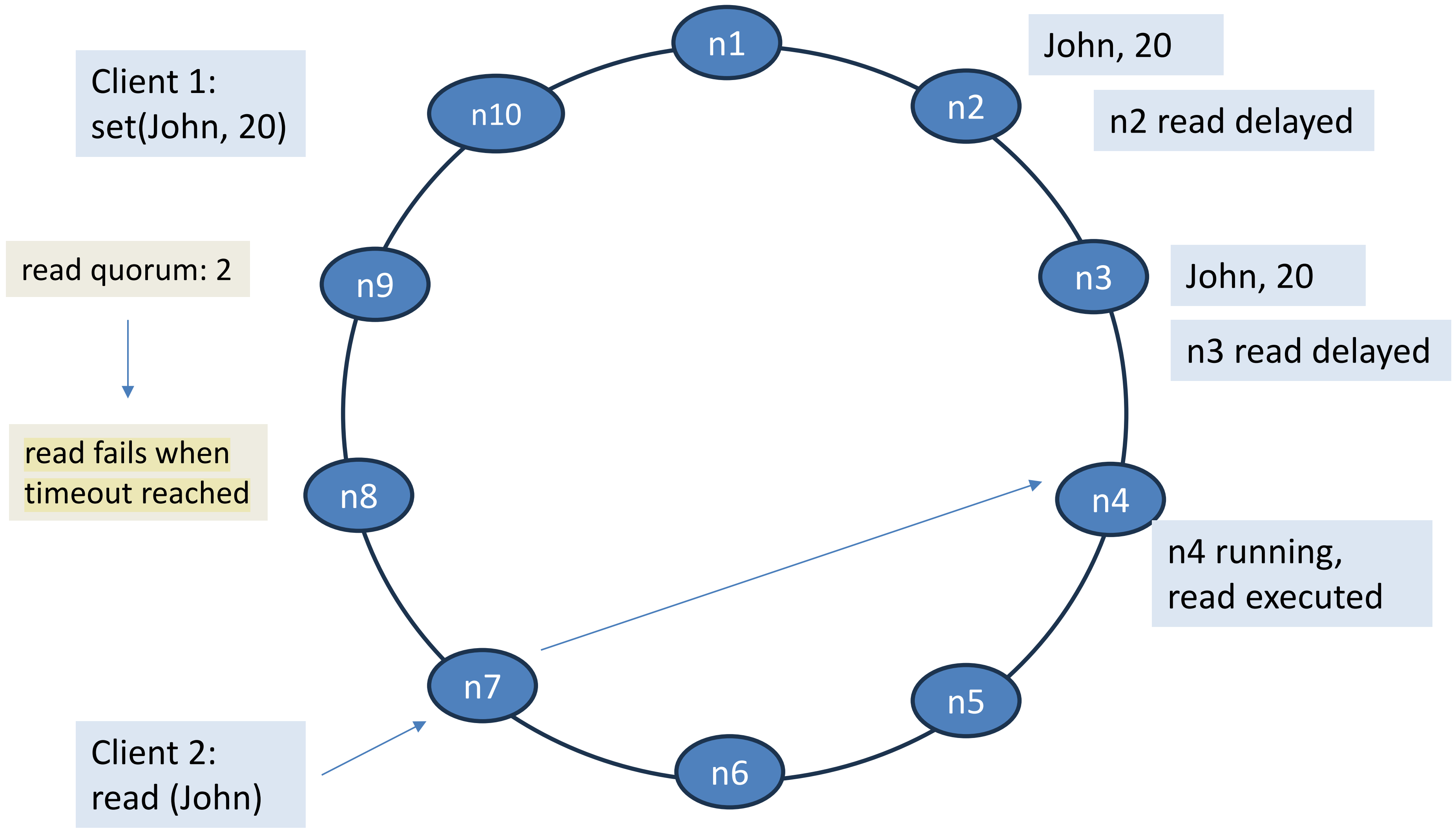
N	number of replicas
W	number of replicas that need to acknowledge a write
R	number of replicas that need to be read

$W < N$; $R < N$: allows the system to continue processing when nodes are down.

$W + R > N$ a read returns – from at least one node - the most recent value for an object because there is an overlap of nodes.

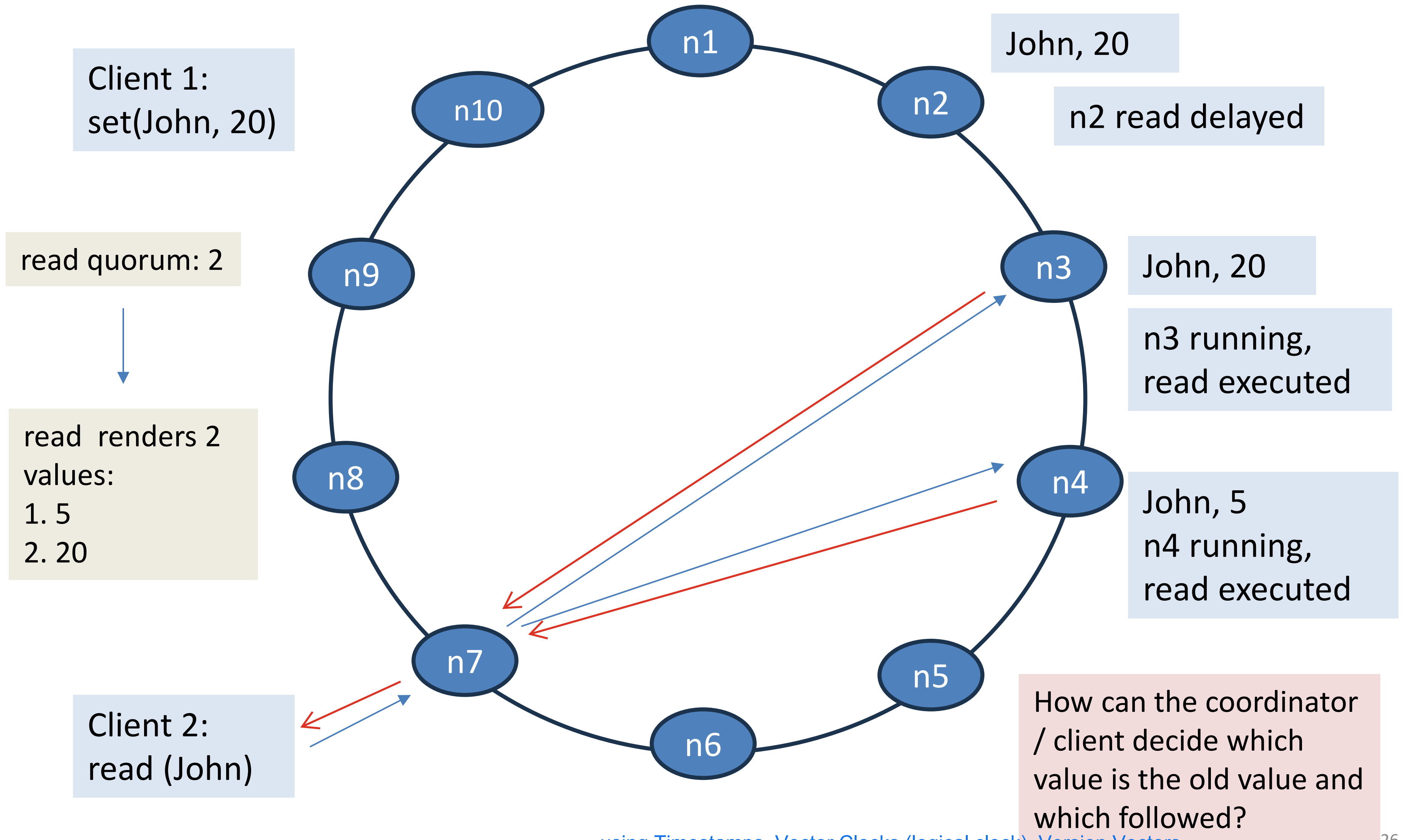
.

Peer-to-Peer Distributed Databases



eva.knirsch@kiu.edu.ge

Peer-to-Peer Distributed Databases



using Timestamps, Vector Clocks (logical clock), Version Vectors

eva.knirsch@kiu.edu.ge

Peer-to-Peer Replication Mechanisms

- Coordinator nodes
- Write and read quorums
- Timestamps, Vector Clocks (logical clock), Version Vectors
- Read Repair
- Merkle Trees

KUTAISI INTERNATIONAL UNIVERSITY

Vector Clocks / Version Vectors

In distributed databases, data is stored and processed across multiple nodes. Because these nodes operate independently and communicate over a network, ensuring the correct order of operations is important.

- In a distributed database it is necessary to order operations (happpened-before, happened-after).
- Example:
  - Insert of object a on node1
  - Read of object a on node 2

  If the read happens after the write, it should see object a

  If it doesn't, users may see stale data, which breaks consistency
- Some systems work with physical timestamps (e.g. Cassandra) to achieve this.
- Physical clocks can be inconsistent because of sync errors between nodes, NTP (network time protocol) failures / delays.
- Some systems work with logical clocks (vector clocks) or version vectors (RIAK, CoucbDB)

KUTAISI INTERNATIONAL UNIVERSITY

Peer-to-Peer Replication

- Version Vectors (Vector Clocks)
  - Wikipedia: The version vector allows the participants to determine if one update
    - preceded another (happened-before),
    - followed it (happened after) or
    - if the two updates happened concurrently.
  - Version Vectors version values / updates
  - "Logical clock"

- Version Vectors / Vector Clocks
  - both are vector counters
  - **Version vectors:** a mechanism to synchronize replicas tracks versions of replicated data across multiple nodes.
  - **Vector clocks**: a mechanism for partial (causal) ordering of events in a messaging system (so, e.g., not to get the answer before the question)
  - Even though they follow (slightly) different algorithms and have different methods, they are often used as synonyms.
  - Implementation of version vectors vary.
-

KUTAISI INTERNATIONAL UNIVERSITY

Vector

Version Vectors / V~~er~~sion Clocks

- A version counter is used per each replica and per each object.
- Each replica increments its version counter when processing a write on an object.
- A version vector (VV) is the collection of version counters of all replicas per object.
- A VV has the size of the replication factor.
- Wikipedia:
    - Each time a replica experiences a local update event, it increments its own version counter in the vector.
    - Each time two replicas a and b synchronize, they both set the counters in their copy of the vector to the maximum of the counter: $V1[x] = V2[x] = \max(V1[x], V2[x])$ After synchronization, the two replicas have identical version vectors.
- VV can synchronize according to $V1[x] = V2[x] = \max(V1[x], V2[x])$ if the are comparable.

let's say we have two replicas:
V1[x]={n1: 2, n2:1, n3:0 }

means n1 updated X twice, B once, C not at all

replica 2:
V2[x]={n1: 1, n2:2, n3:0 }

When these two sync their copies of object X, they compare version vectors. For each entry x in the VV they take the max counter value and the version they agree on is {n1:2, n2:2, n3:0 }. so basically merging is happening.

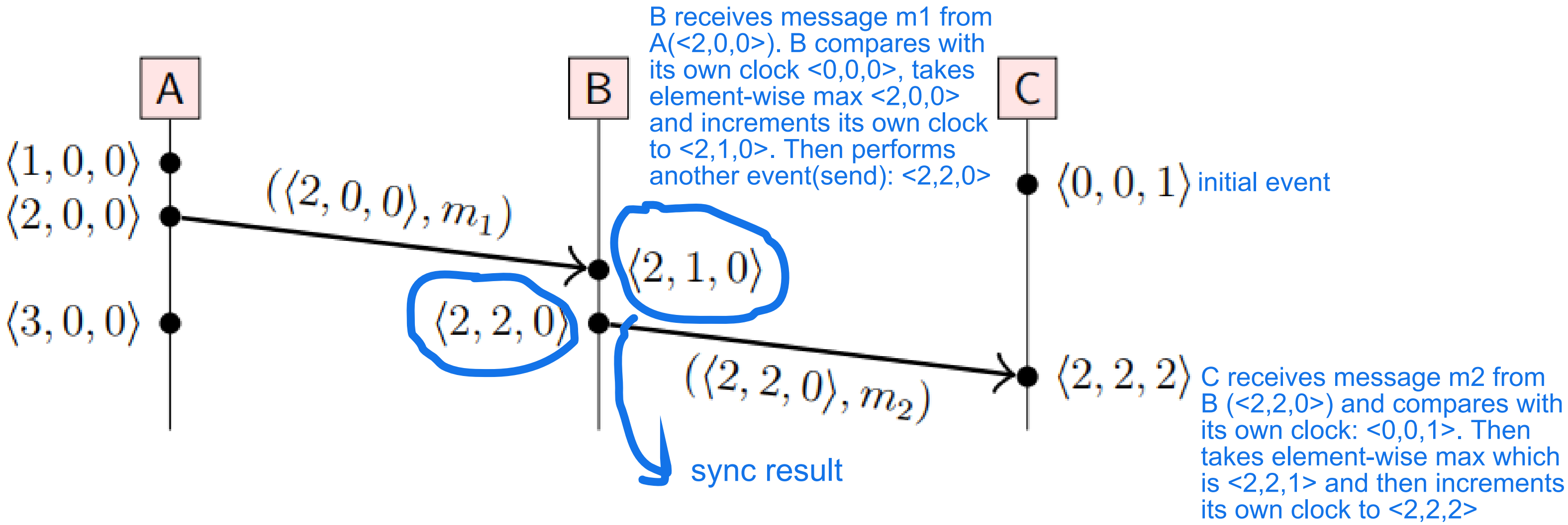
means n1 updated X once, B twice, C not at all

KUTAISI INTERNATIONAL UNIVERSITY

Vector Clocks

Vector clocks example

Assuming the vector of nodes is $N = \langle A, B, C \rangle$:



Notice the difference between vector clock and version vector: vector clock increments at each event (e.g. send, receive); version vector increments with each update.

So vector clocks are about event ordering, while version vectors are about data versioning.

Kleppmann, Distributed Systems

eva.knirsch@kiu.edu.ge

Vector clocks ordering

Define the following order on vector timestamps (in a system with n nodes):

- $T = T'$ iff $T[i] = T'[i]$ for all $i \in \{1, \dots, n\}$
- $T \leq T'$ iff $T[i] \leq T'[i]$ for all $i \in \{1, \dots, n\}$
- $T < T'$ iff $T \leq T'$ and $T \neq T'$
- $T \parallel T'$ iff $T \nleq T'$ and $T' \nleq T$

$V(a) \leq V(b)$ iff $(\{a\} \cup \{e \mid e \rightarrow a\}) \subseteq (\{b\} \cup \{e \mid e \rightarrow b\})$

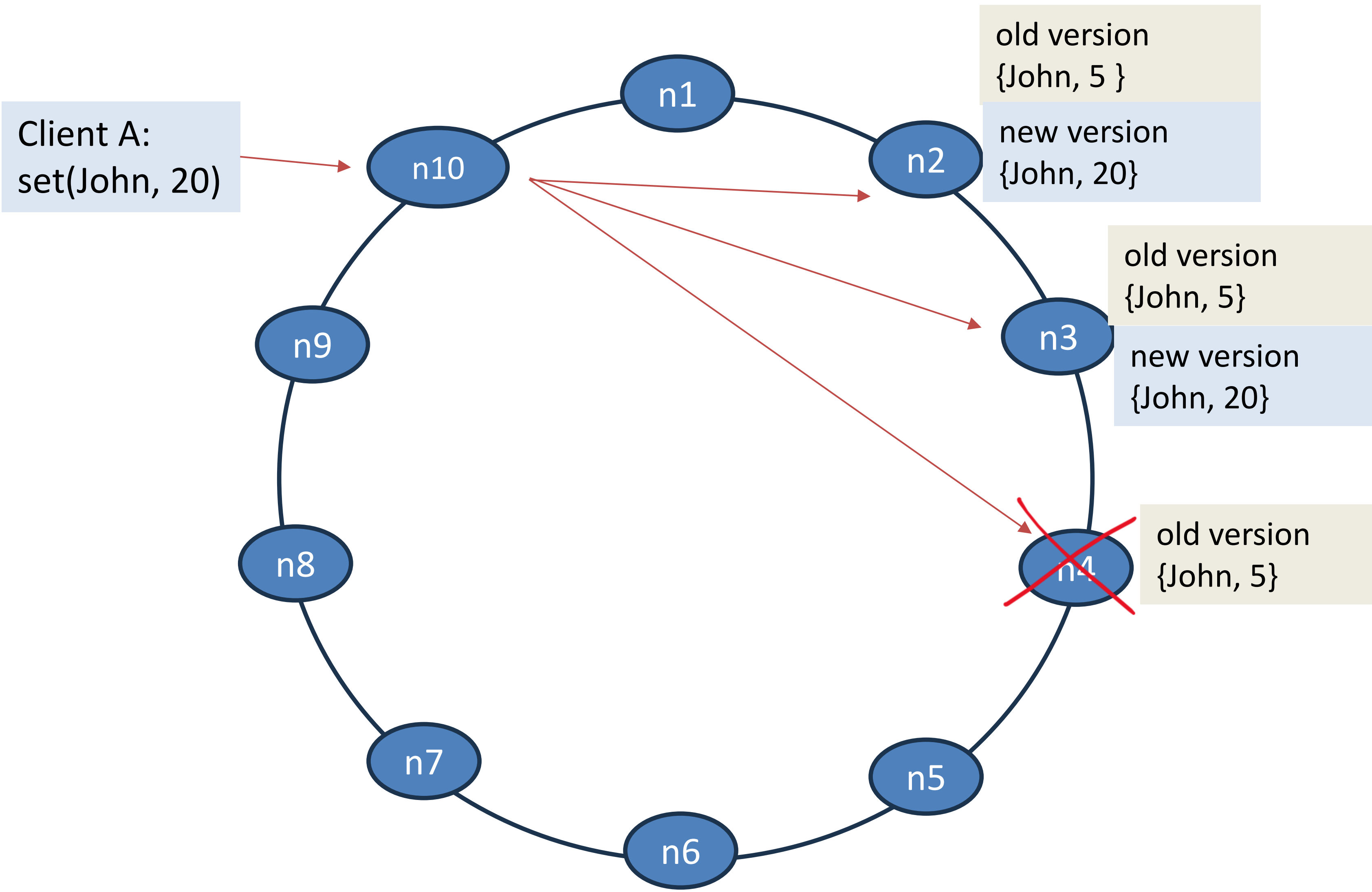
Properties of this order:

- $(V(a) < V(b)) \iff (a \rightarrow b)$
- $(V(a) = V(b)) \iff (a = b)$ equality means that vectors are in sync
- $(V(a) \parallel V(b)) \iff (a \parallel b)$ version vector detects this problem but does not solve it

"We say that one vector is less than or equal to another vector if every element of the first vector is less than or equal to the corresponding element of the second vector. One vector is strictly less than another vector if they are less than or equal, and if they differ in at least one element. However, two vectors are incomparable if one vector has a greater value in one element, and the other has a greater value in a different element." incomparable means that they are concurrent

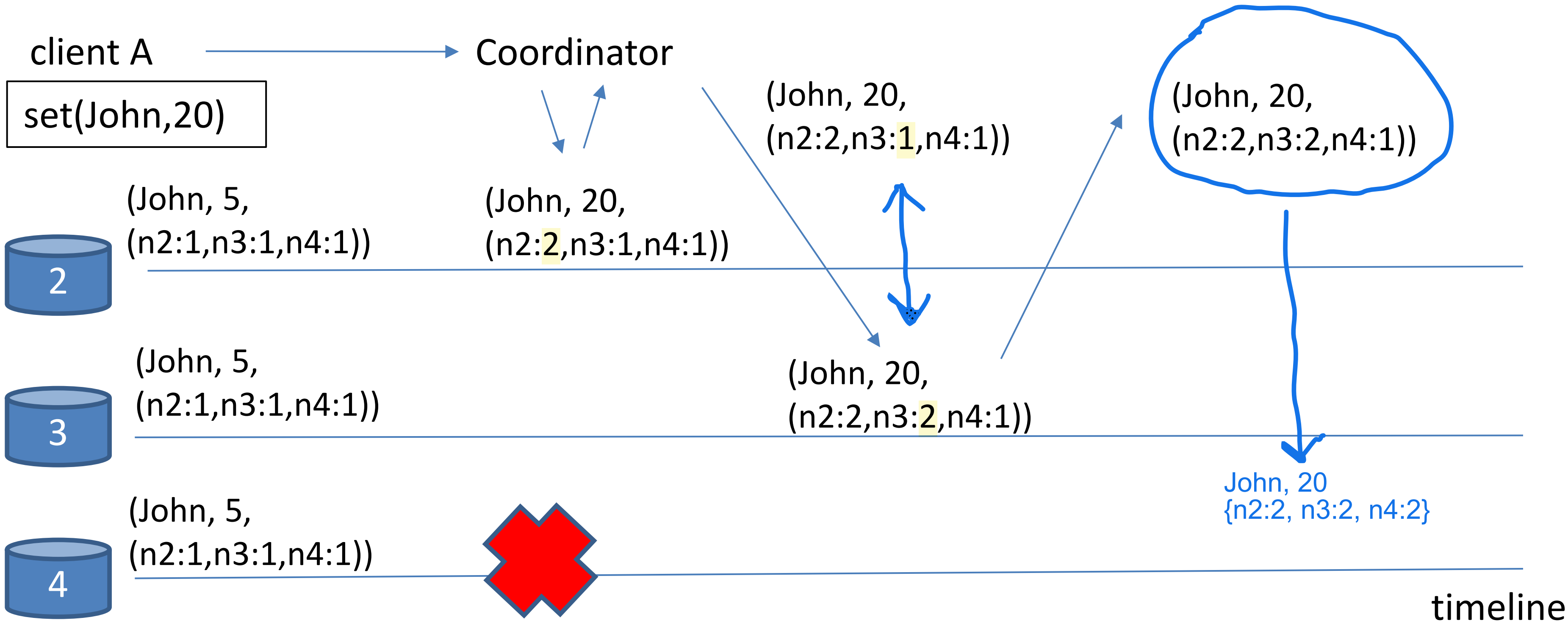
For example, T = {2; 0; 0} and T' = {0; 0; 1} are incomparable because T[1] > T'[1] but T[3] < T'[3].
Kleppmann, distributed Systems

Peer-to-Peer Distributed Databases



eva.knirsch@kiu.edu.ge

Write Operation -Version Vector (VV)



Are the VV of nodes 2 and 3 comparable after the write operation is acknowledged?

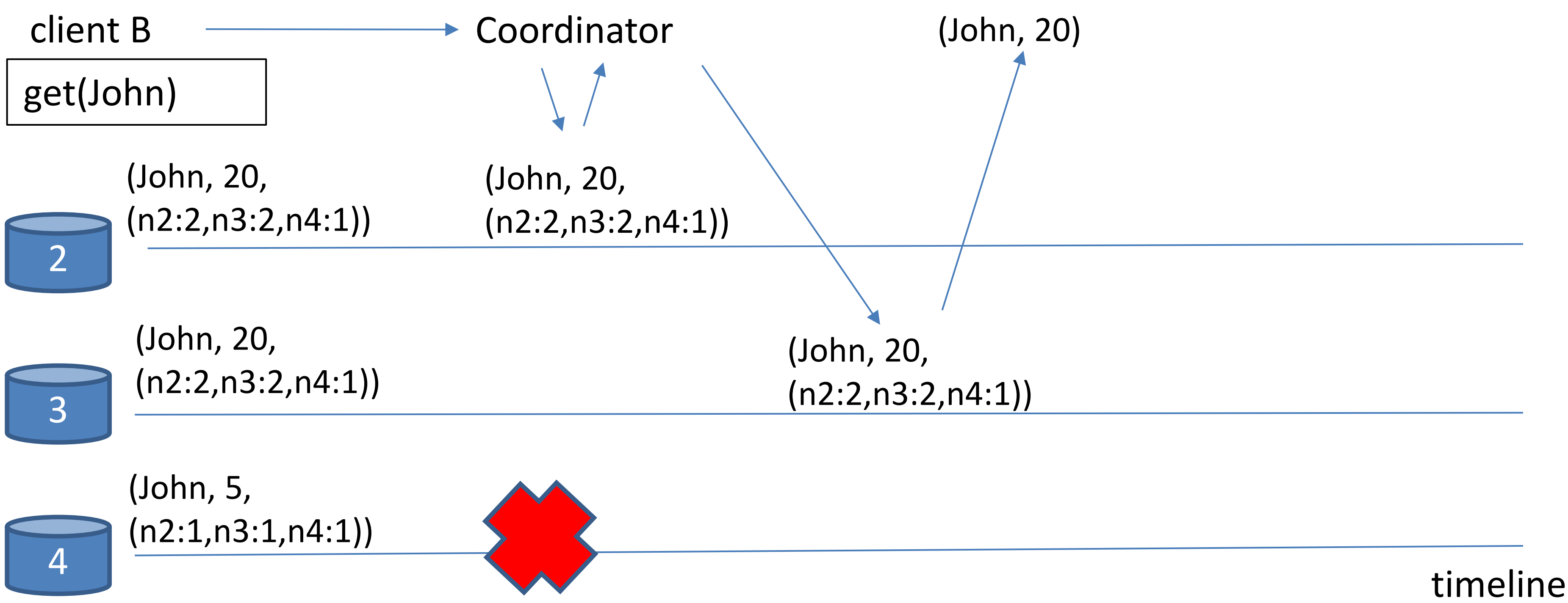
node2: (n2:2,n3:1,n4:1)

node3: (n2:2,n3:2,n4:1)

→ sync result: John, 20 {n2:2, n3:2, n4:2}

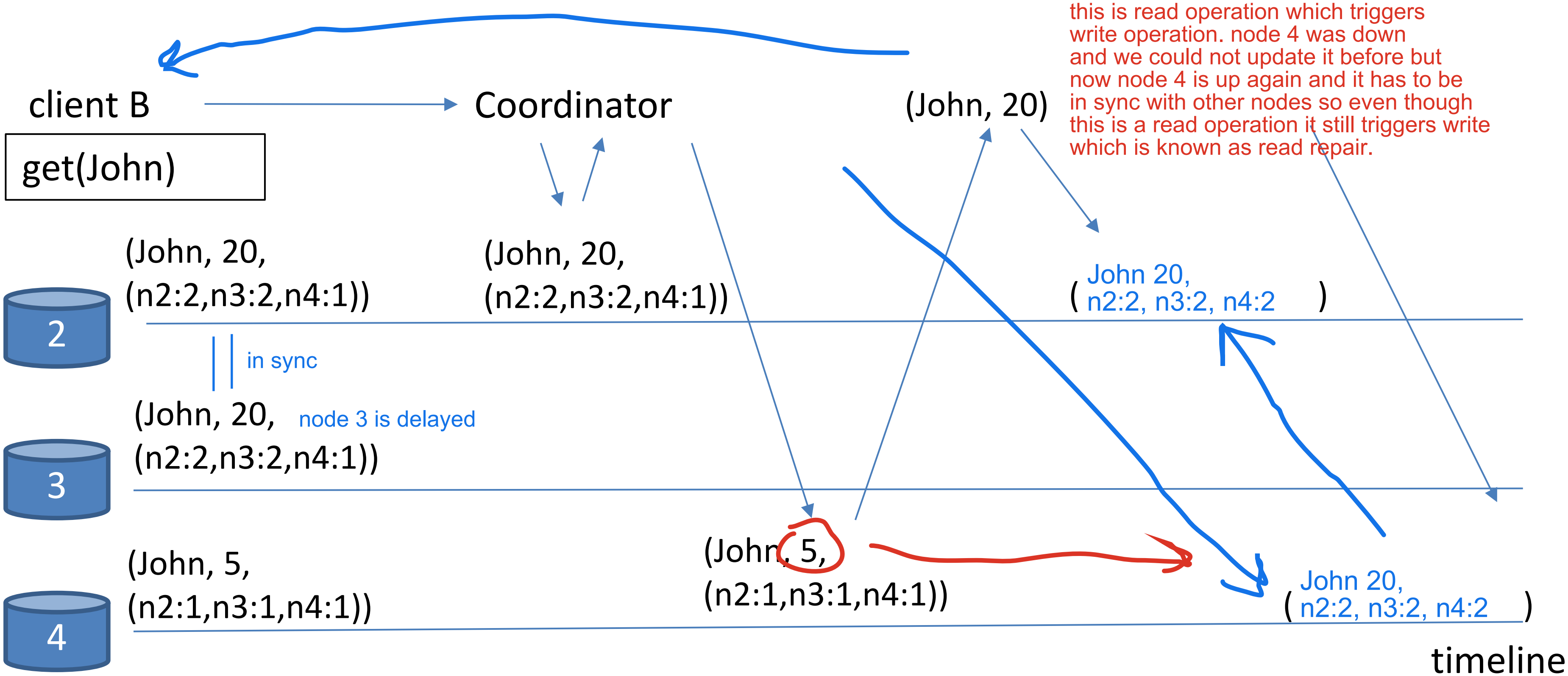
We say two version vectors are comparable if all vector elements of one VV are less than or equal to the corresponding elements in the other. since 2=2, 1<2 and 1=1 they follow our rule and they are comparable.

Read Operation - Version Vector (VV)



read repair does 2 things: it updates preceding values that happened before to current value and it syncs version vectors

Version Vector (VV) – Read Repair



The read operation reads from nodes 2 and 4. Are the VV of nodes 2 and 4 comparable? Which is the ≤ VV?

Node4 VV is strictly smaller than the node2 VV (→ value of node 2 is more recent)

→ the value of node2 is returned (John: 20)

→ the value of node4 is updated

→ the 2 nodes are syncd → same VV on both nodes -→ sync result

eva.knirsch@kiu.edu.ge

Read Repair

A successful read operation may trigger a so-called read-repair:

- With the help of a version vector the coordinator detects an old value (old version) on a replica.
- It returns the new value to the client.
- It starts a write process (read repair process) and updates the stale replica version to the latest value (version).
- The write syncs the VV

→ a read operation triggers a write operation to synchronize replica values and VV.

RIAK documentation:

Read repair occurs when a successful read occurs ..... but not all replicas of the object agree on the value. There are two possibilities here for the errant nodes:

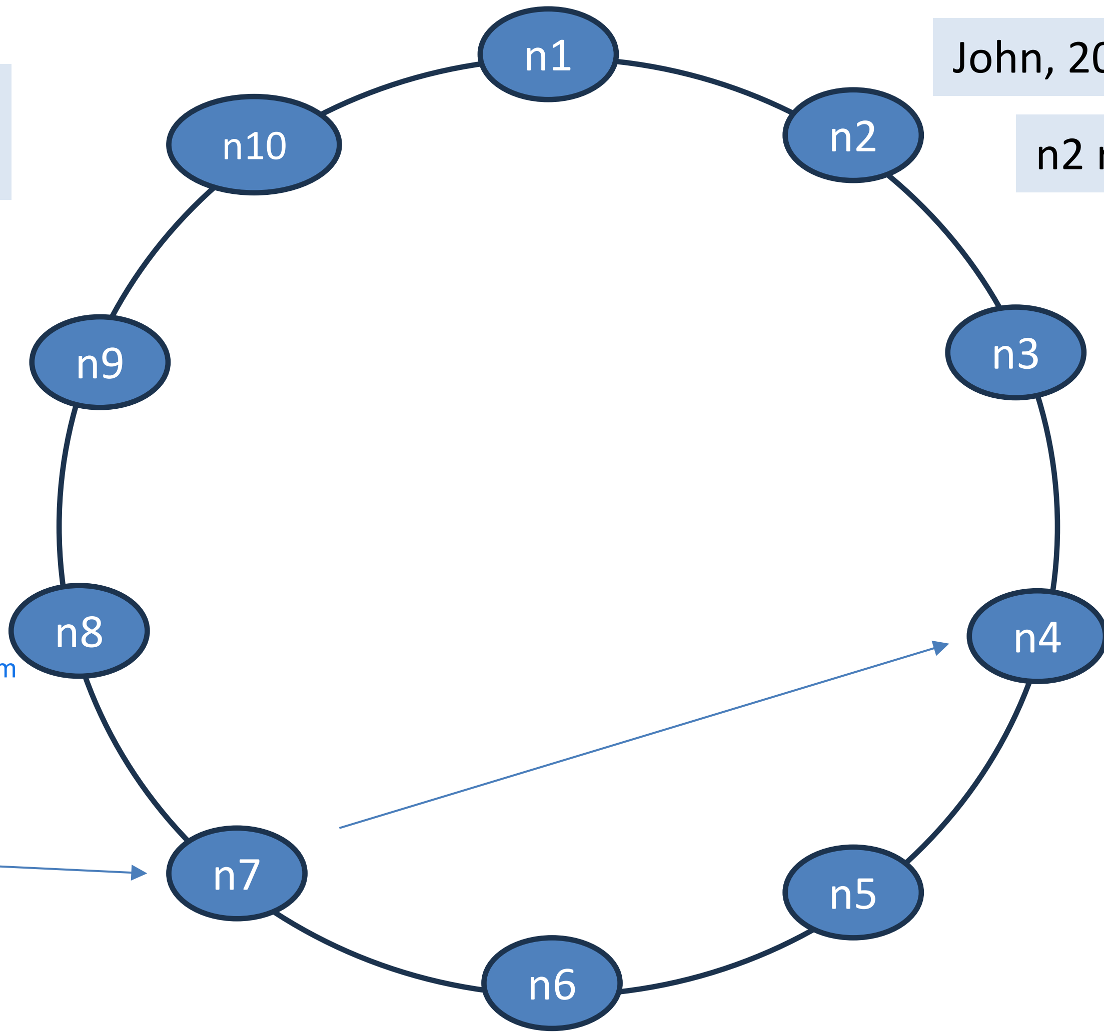
The node responded with a not found for the object, meaning that it doesn't have a copy.

The node responded with a vector clock that is an ancestor of the vector clock of the successful read.
this means node saw some earlier write, but not the most recent one

When this situation occurs, Riak will force the errant nodes to update the object's value based on the value of the successful read.

Peer-to-Peer Distributed Databases

Client 1:
set(John, 20)



John, 20

n2 read delayed

John, 20

n3 read delayed

read quorum: 1
→ no read repair

well since read quorum is 1 which means that we read from just 1 node and n2 and n3 are delayed, we read from n4 and get old value but despite that there is no need of read repair because we only read from one node and n4 basically does not have anyone to compare to.

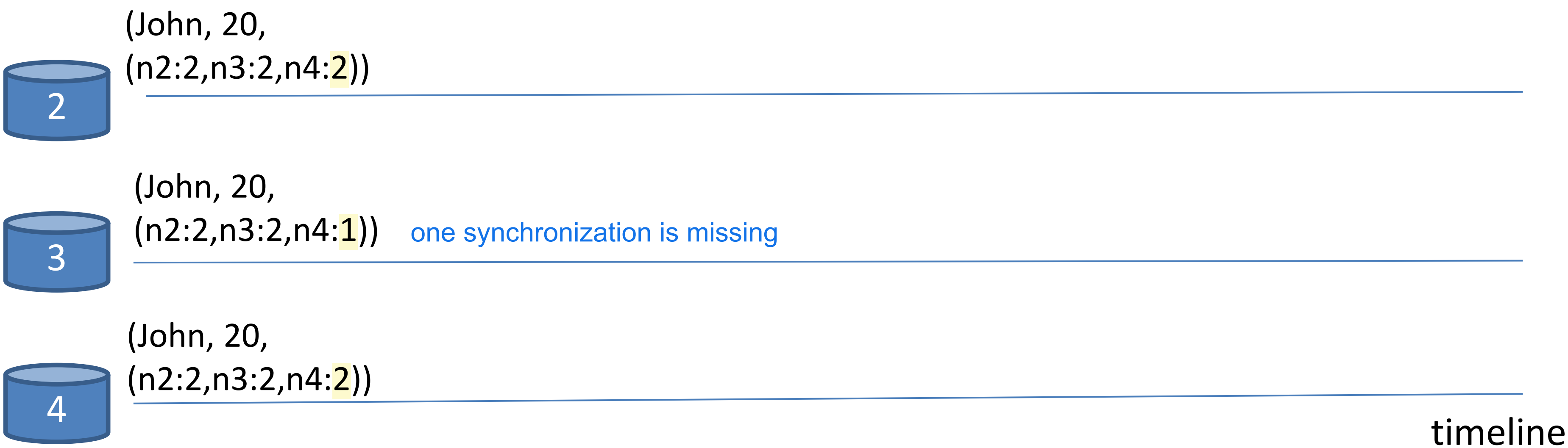
n4 running,
read executed

John, 5

Client 2:
read (John)

n7
Cooordinator

Anti-Entropy-Mechanism



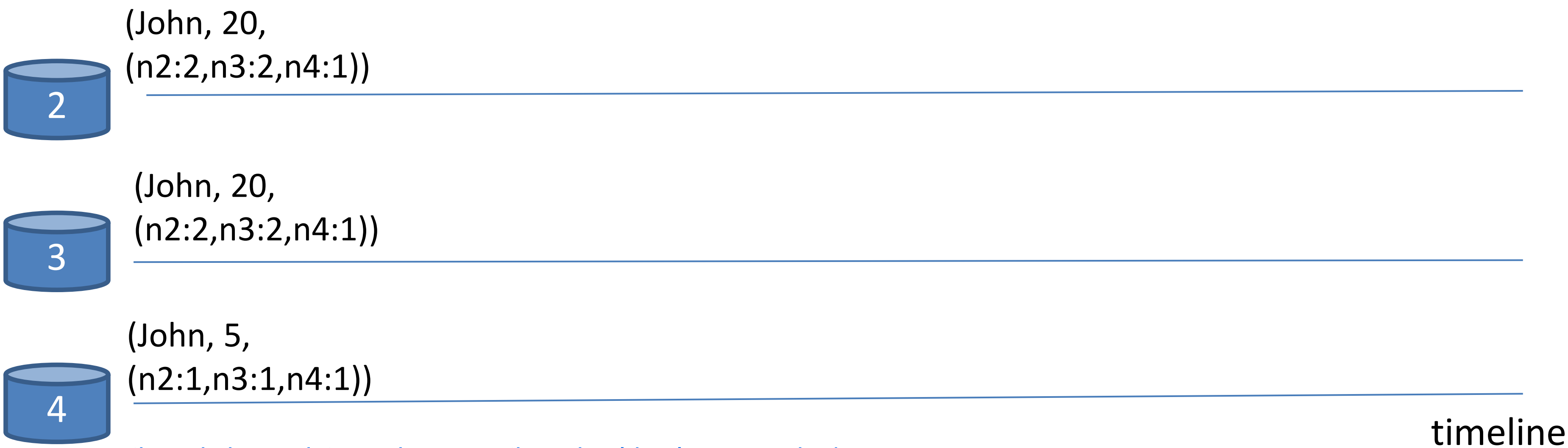
VV of node 3 is not in sync yet with nodes 2 and 4. Values are in sync and consistent.
VV are comparable. So, they can sync any time.

eventual consistency achieved

whether or not version vectors are not identical, as long as vectors are comparable, synchronization can take place any time but it's not urgent.

eva.knirsch@kiu.edu.ge

Anti-Entropy Mechanism



there is inconsistency becaues there is old value on node 4

How does the system detect inconsistent values ..... if there is no read? since there is no read there is no read repair

Eventual consistency requires that – eventually – the nodes agree on one value.

eva.knirsch@kiu.edu.ge

Read Repair and Anti-Entropy Process

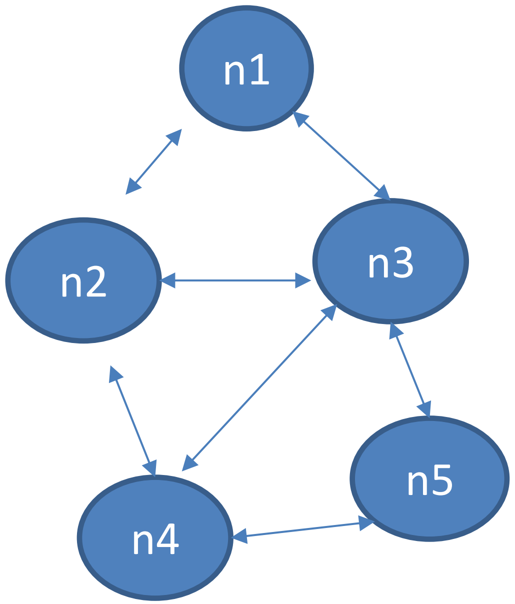
The read repair mechanism can only identify and update stale values (and sync VV) if the data is queried. Data, that is not queried, does not get updated.

That is why, many systems have additional Anti-Entropy processes running in the background. These processes constantly check whether replicas differ in data.

In principle, the anti-entropy processes have to compare data units (records, key-values pairs or collections, documents, sets, ..) of any two replicas pair-wise. As this is a cost-intensive process, Merkle trees are often used to facilitate the anti-entropy process.

entropy = inconsistency between replicas.

Entropy: describes the state of divergence between the replicas. Is is thus not desirable.



Anti-Entropy Process

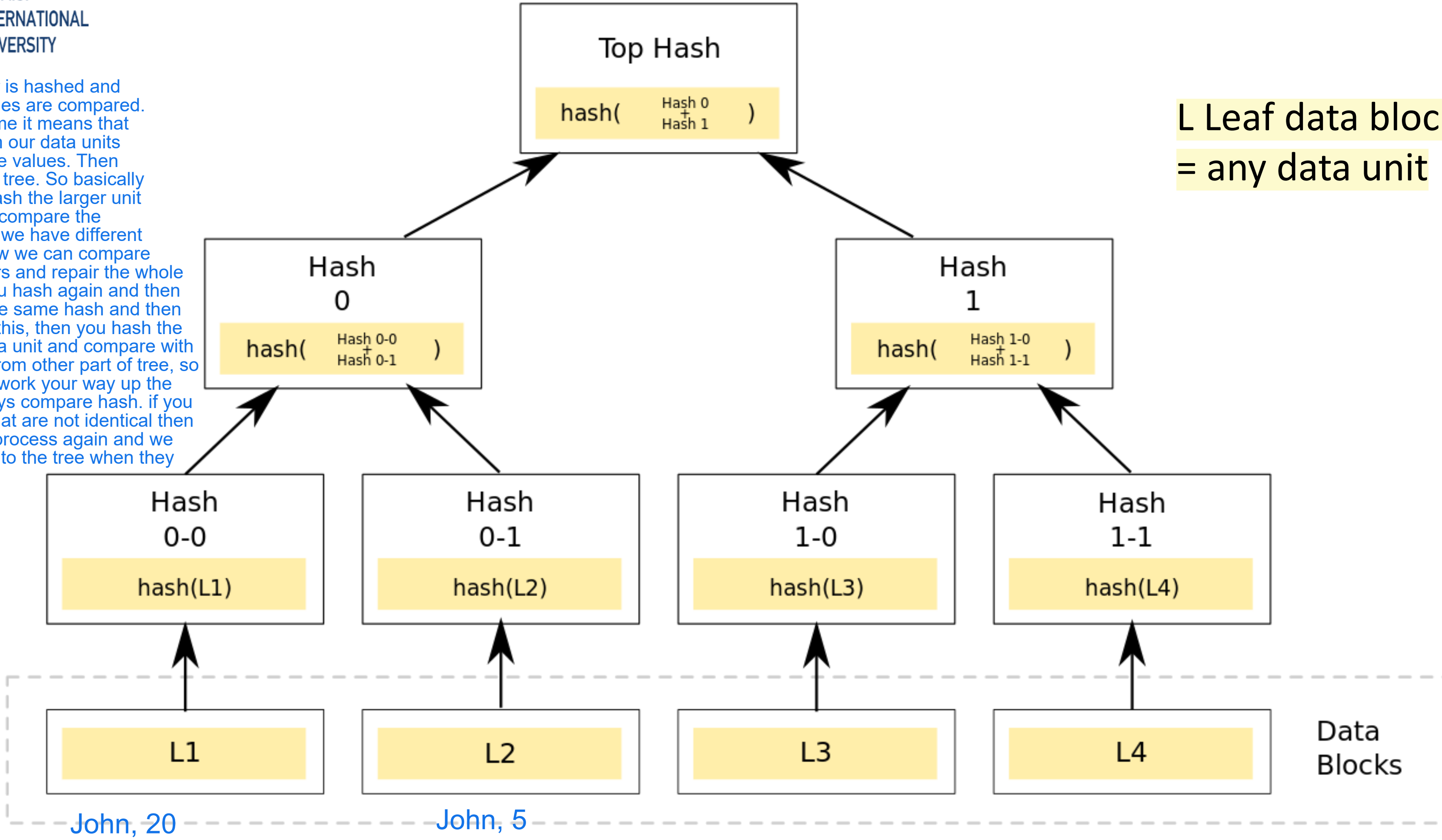
RIAK documentation:
One advantage of using read repair alone is that it doesn’t require any kind of background process to take effect, which can cut down on CPU resource usage. The drawback ... is that the healing process can only ever reach those objects that are read by clients. Any conflicts in objects that are not read by clients will go undetected. ...

The active anti-entropy (AAE) subsystem ... runs as a continuous background process ....

In order to compare object values between replicas without using more resources than necessary, Riak relies on **Merkle tree hash exchanges** between nodes.

KUTAISI INTERNATIONAL UNIVERSITY

key value pair is hashed and then two hashes are compared. if they are same it means that at very bottom our data units have the same values. Then you go up the tree. So basically you always hash the larger unit and then you compare the hashes. Here we have different hashes so now we can compare version vectors and repair the whole thing, then you hash again and then we would have same hash and then you combine this, then you hash the combined data unit and compare with another one from other part of tree, so basically you work your way up the tree and always compare hash. if you find hashes that are not identical then we do repair process again and we only move up to the tree when they are identical.



L Leaf data block = any data unit

when one value changes, the root value changes thus we only have to check all the root values to determine if data has been modified.

Explain how Merkle Trees make the anti-entropy process more efficient. Two nodes compare their data. Explain 2 cases:

1. The replicas are consistent. if their top hashes are equal then they are identical therefore consistent
2. The replicas diverge from each other. How do the processes work in each case?

Wikipedia

eva.knirsch@kiu.edu.ge