10
Distributed Databases
Partitioning / Sharding

Reading: [Kl], chapter 6; [Ha] chapter 3

Course content

1. Introduction SQL and NoSQL Databases

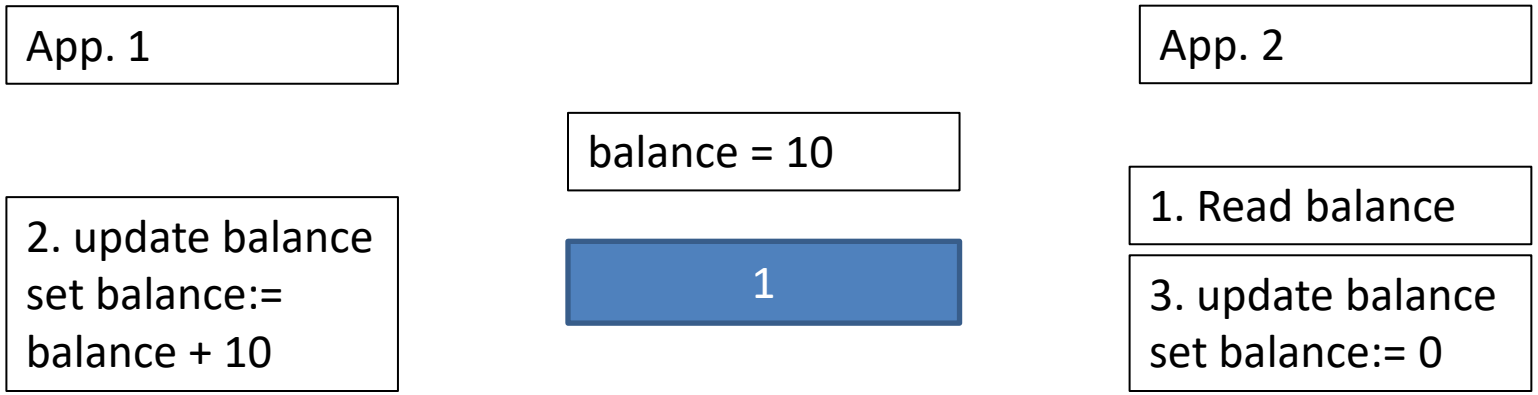
2. Relational Databases:
    1. Transactions
    2. Concurrency Control and Consistency

3. Data and Storage Models
    1. Relational (Reference)
    2. Key-Value
    3. Document
    4. Column-Family

4. Distributed Databases
    1. Replication
    2. Sharding
    3. Distributed Transactions / Consistency in Distributed Databases

knirsch@htw-berlin.de

Concurrent Writes in a Relational Single-Node Database



1. What problem could happen and how can it be prevented?

   lost update

2. What to do on the application side?

   use transactions with correct isolation which should handle rollbacks

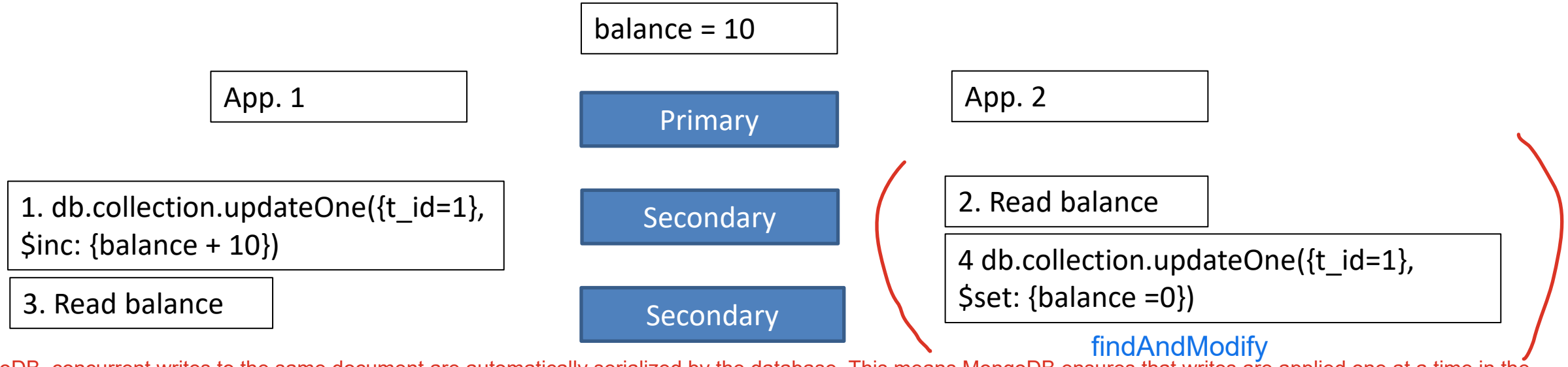
3. How are the writes processed?

   it depends on protocol: 2PL, MVCC

   2PL enforces a certain order to ensure serializability by using locks. In contrast, MVCC does not enforce such strict ordering. As a result when two transactions run concurrently in MVCC, one of them may be rolled back and this happens frequently because there is no strict execution order like in 2PL.

eva.knirsch@kiu.edu.ge

Concurrent Writes in Single-Leader Replicated Database



In MongoDB, concurrent writes to the same document are automatically serialized by the database. This means MongoDB ensures that writes are applied one at a time in the order they are received, preventing conflicting simultaneous updates. However, if your application reads a document, processes it and then writes an update, there's a race condition risk: another write could happen in between your read and your update. This can lead to inconsistent or incorrect results. To avoid this, MongoDB provides an atomic operation called findAndModify. This operation allows you to find a document and modify it in a single atomic step. Since it happens as one atomic action, no other operation can modify the document between the find and the update. This ensures correctness and consistency in concurrent environments.

In what case would the read return a stale value?

write goes to the primary and read goes to secondary and secondary has not yet applied the oplog so secondary reads stale value. But this is not a consistency problem because sooner or later it will apply oplog and we will have new value.

In what case would the read return a possibly inconsistent value ("dirty read")?

for example we apply 20 to primary and we have w:1 so it is immediately acknowledged. Then we read from primary and get 20, in this moment oplogs to secondaries are delayed. so secondaries can't reach primary and in 10 seconds they think that primary is down and start election and one of then becomes primary. and once the primary can sync with them again, primary notices that oplog now has more recent timestamp, understands that it is not primary anymore and rolls back the writes that were not replicated and those writes are lost.

In what case is the "read-your-own-writes" violated?

read-your-own-writes means that after you write data, any subsequent read you perform will return the value you just wrote even if the system has replicas or delays. So stale value is an issue if you want to have read-your-own-writes. if we have for readPreference: primary or write and read concerns majority, then it won't be violated.

thee are

NOTE: in replicated database rollback means loss of data while in transactional database it means to go back to the initial value

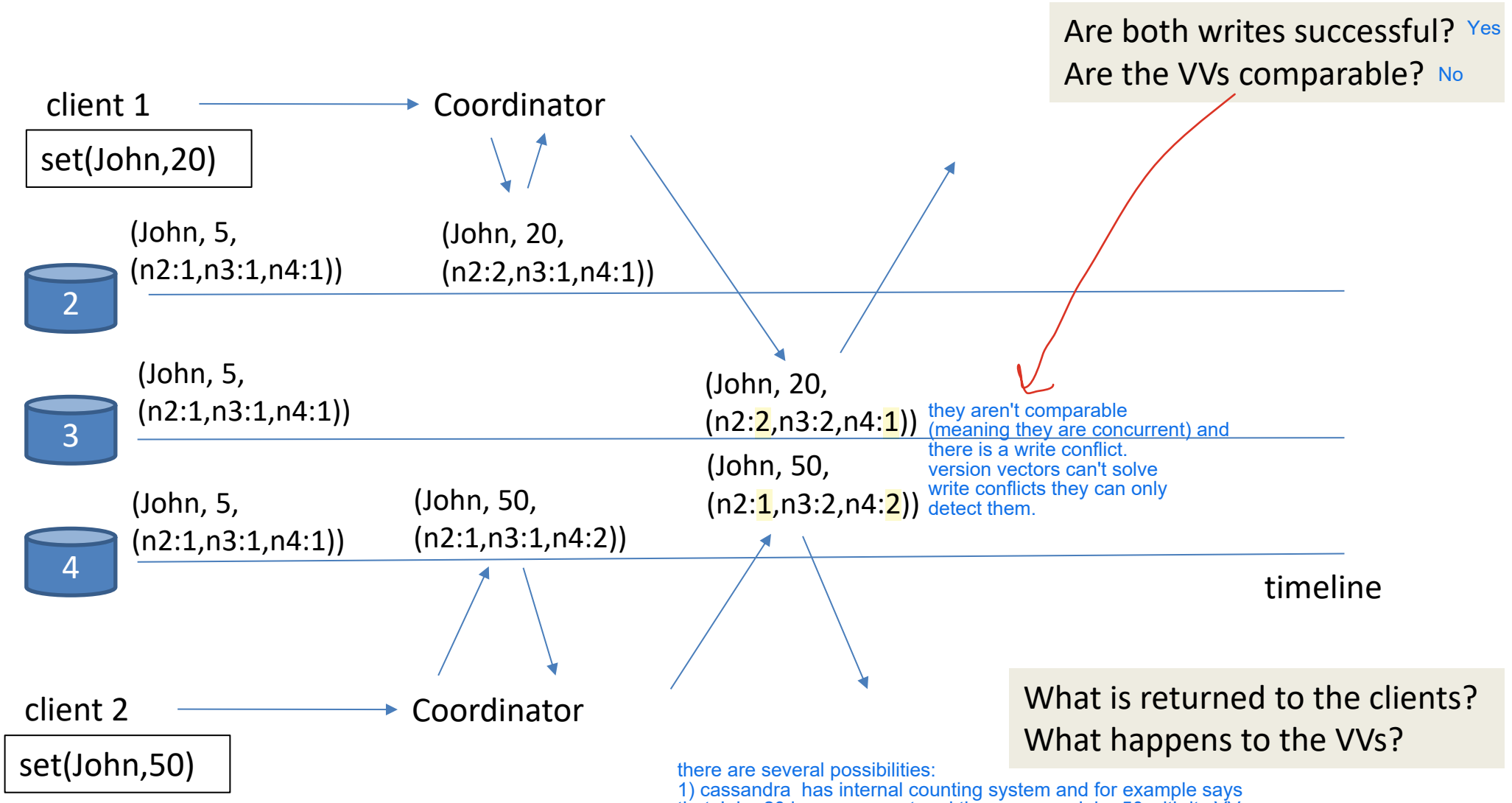
eva.knirsch@kiu.edu.ge

MongoDB: Primary-Secondary Replication

- single-document write operations are atomic
  → they are serialized using locks
- all writes are processed by the primary and written to Oplog, secondaries apply oplog.
- Replication process is asynchronuous by default; write concern can be used to force replication to be (partly) synchronuous.
- Stale reads or even dirty reads can happen, read-your-own writes may be violated
- Loss of Data during rollback: "A rollback reverts write operations on a former primary when the member rejoins its replica set after a failover. A rollback is necessary only if the primary had accepted write operations that the secondaries had **not** successfully replicated before the primary stepped down. When the primary rejoins the set as a secondary, it reverts, or "rolls back," its write operations to maintain database consistency with the other members.
  A rollback does *not* occur if the write operations replicate to another member of the replica set before the primary steps down *and* if that member remains available and accessible to a majority of the replica set.
- Multi-document operations (e.g. on different shards): Distributed transaction

eva.knirsch@kiu.edu.ge

Concurrent Writes in P2P Distributed Databases



eva.knirsch@kiu.edu.ge

Failover Process

A failover process is needed for

A) Primary-Secondary Replicated Database Systems single point of failure, if primary goes down whole thing is stalled

B) P2P Replicated Database Systems whole idea of P2P is that if nodes are down they continue to work

C) For both kind of Database Systems

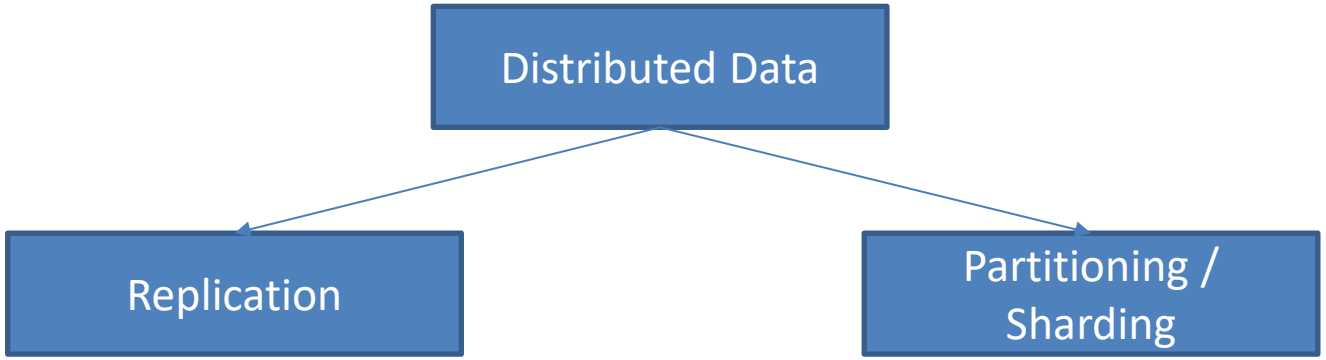
D) For none of the two Database Systems

eva.knirsch@kiu.edu.ge

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 all nodes store the same data.
  - Strong eventual consistency: Using data structures that guarantee a conflict-resolution merge.

Distributed Databases



Copies of the data are kept on mutliple nodes.

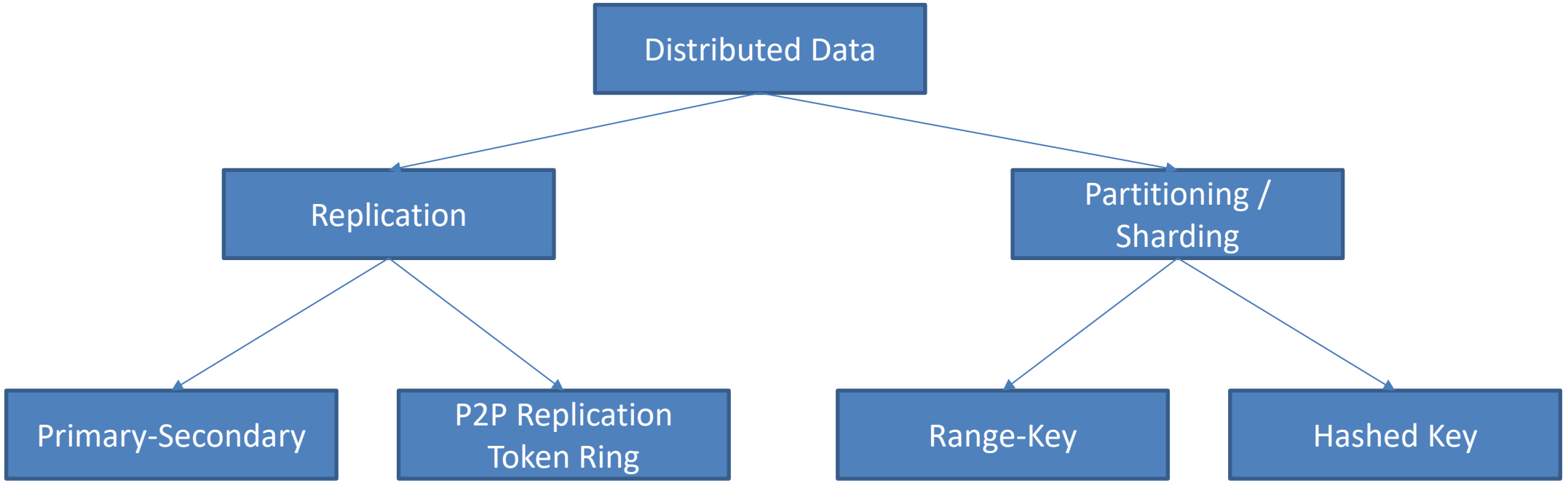
Data is distributed onto multiple nodes.

Replication is very common with both, SQL and NoSQl databases.
Sharding is a lot more common in NoSQL databases and usually not done in SQL databases.
Why is sharding a lot more used with NoSQL databases?

SQL uses joins which does not go well for sharding

NoSQL is often used when there's a huge amount of data. These kind of systems need to save data very quickly and get results fast. To handle this data needs to be spread across many machines, which is exactly what sharding does.

Distributed Databases



MongoDB, HBAse, CockroachDB

RIAK, Cassandra, Dynamo, Couchbase, REDIS, MongoDB

Partitioning / Sharding Goals:

1. even distribution of workload

2. adding/removing nodes incrementally and easily

Shard Key Strategies

The shard key controls the distribution of data.

Hashed Key

Keys are hashed and the hash values control the distribution of the data items across the nodes.

This strategy reduces the chance of hotspots. It distributes the data across the nbodes evenly.

A "hotspot" happens when one shard gets much more traffic than others for example all recent users going to the same shard. Hashing avoids this by randomizing the location of the data.

Disadvantage:

breaks locality, which means data that's logically close like users in the same city could be stored on completely different shards.

Range Key

The shard key partitions the data into ranges and each node holds a range.

Example: Shard key: postalCode, ranges: 10000-30000,30000-60000, 60000-99999

This strategy can group related items together on the same node, and orders them by shard key—the shard keys are sequential. It's useful for applications that frequently retrieve sets of items using range queries.

Possible disadvantage:

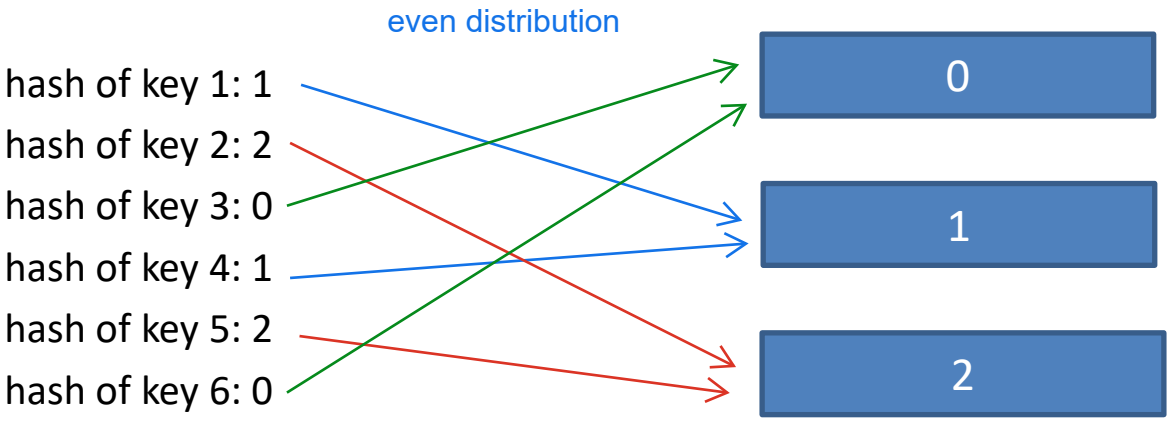
uneven distribution --> hotspots(one shard is overloaded while others are underused)

eva.knirsch@kiu.edu.ge

Hashed Shard Key in P2P distributed database
Why Token Ring?

Example with 3 nodes and
modulo 3 hash

Keys: 1,2,3,4,5,6,7,8,9

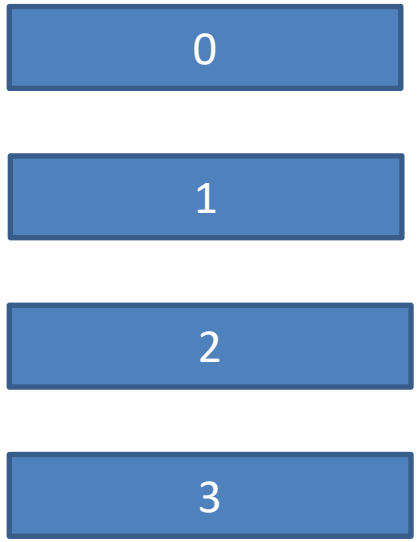


knirsch@htw-berlin.de

Hashed Shard Key in P2P distributed database
Why Token Ring?

Keys: 1,2,3,4,5,6,7,8,9

hash of key 1: 1
hash of key 2: 2
hash of key 3: 0
hash of key 4: 1
hash of key 5: 2
hash of key 6: 0
hash of key 7:
hash of key 8:



Adding another server node 4.

majority of data needs to be redistributed

for example with naive hashing if you're using simple hash(key) % N and N increases from 3 to 4, all hashes change. Now hash(key) % 4 will give different results and most of the keys will go to different servers.

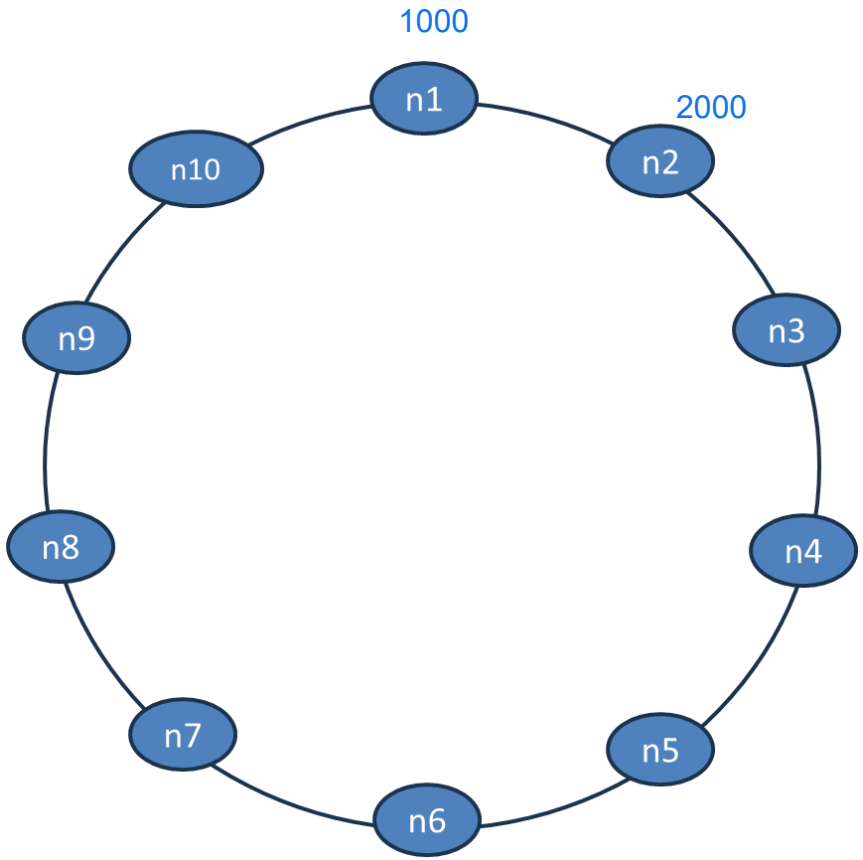
knirsch@htw-berlin.de

Basic Consistent Hashing

Dynamo Paper:

- One of the key design requirements for Dynamo is that it must scale incrementally.
- In consistent hashing, the output range of a hash function is treated as a fixed circular space or “ring” (i.e. the largest hash value wraps around to the smallest hash value).
- Each node in the system is assigned a random value within this space which represents its “position” on the ring.
- Each data item identified by a key is assigned to a node by hashing the data item’s key to yield its position on the ring, and then walking the ring clockwise to find the first node with a position larger than the item’s position.

Hashed Shard Key on the Token Ring – Consistent Hashing



1. Define hash space: the output range of the hash function, from minimum value to maximum value. Map the hash space onto the logical ring (hash ring).

   Example: Hash Space: 1 – 10000

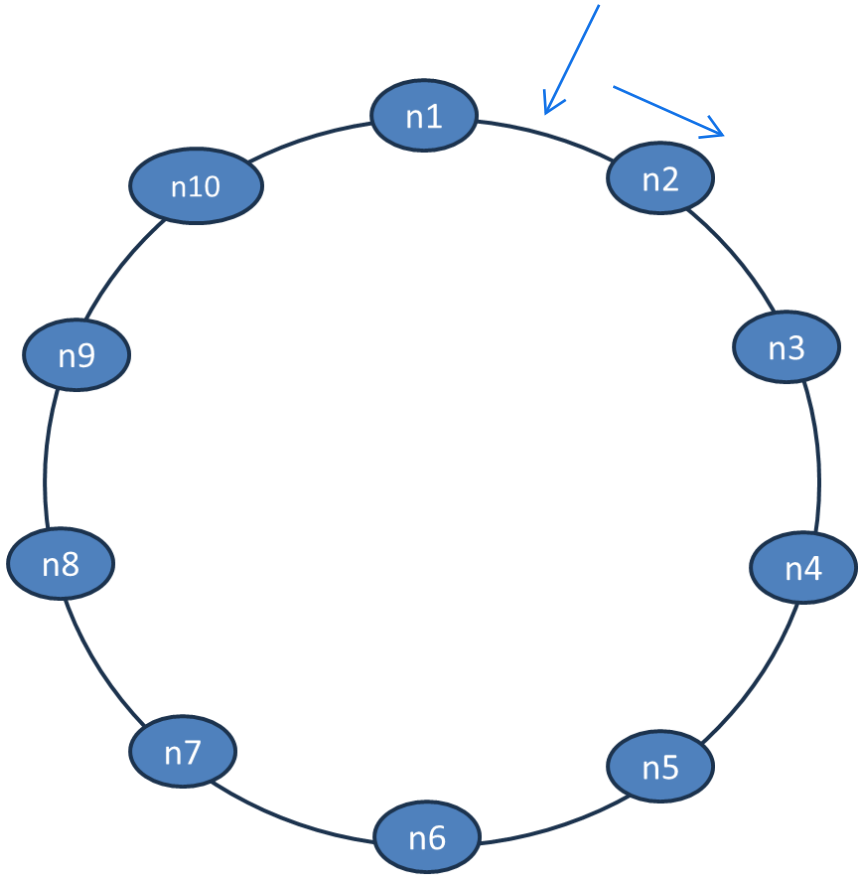
2. Map each server node to a position on the ring.

   Example: n1 → position 1000
   n2 → position 2000

3. Position the nodes onto the correct position on the ring.

eva.knirsch@kiu.edu.ge

Hashed Shard Key on the Token Ring – Consistent Hashing



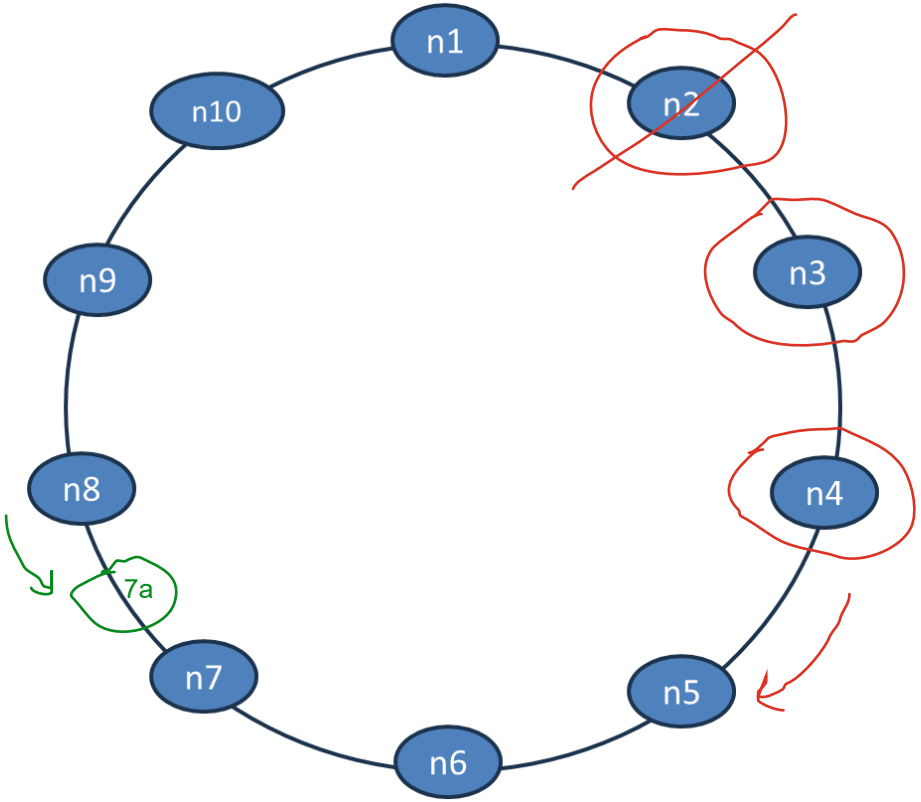
4. Calculate hash for each key.
   Example: John hashes to 1500

5. Locate hashed key on the token ring.
   Example: Key John is placed between node1 and node2

6. Each key is stored on the node that is closest to the hashed key in clockwise direction. Traverse ring clockwise to locate the host node.
   Example: Key John is stored on node 2

eva.knirsch@kiu.edu.ge

Hashed Shard Key on the Token Ring – Consistent Hashing



Remove (decommission) of n2 from the ring

→ keys with hash space 1001 – 2000 need to be moved

→ assume that databases (key spaces) have a rf = 3:

Add server node7a to position 7500

→keys with hash space 7001 – 7500 need to be moved from node 8 to new node 7a

→ keys with hash space 7001 – 7500 no longer replicated on n10 (with rf = 3)

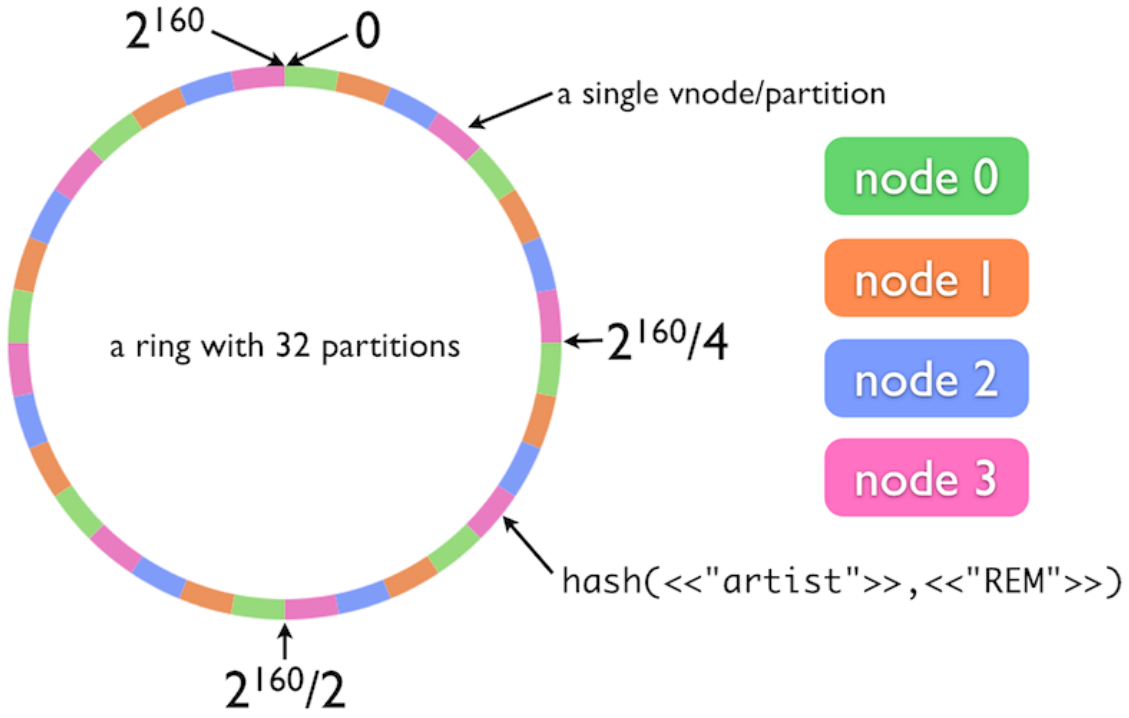
Adding/ removing servers can lead to

- heavy workload on the few concerned server nodes that handle the "moving" data
- unevenly distributed data.

eva.knirsch@kiu.edu.ge

Hashed Shard Key on the Token Ring – Consistent Hashing

in this case we have 8 virtual nodes per physical node.



Goal is a fast, lightweight incremental scaling

- → instead of positioning (physical) server nodes on the ring, each node is "split" into virtual nodes.
- → Cassandra: "virtual nodes... facilitate flexible expansion with more streaming peers". distribution of workload is easier
- → If a node is added / removed, more peers (vnodes) take part of the data-shuffling operation and data stays distributed more evenly
- → The system generates random tokens (positions) for each vnode on the ring.
- → positions of vnodes need to be unique in order that each hashed key is assigned to exactly one vnode (one position) on the ring

hash space: 2^x

2^{20} = 1.048.576 hash values

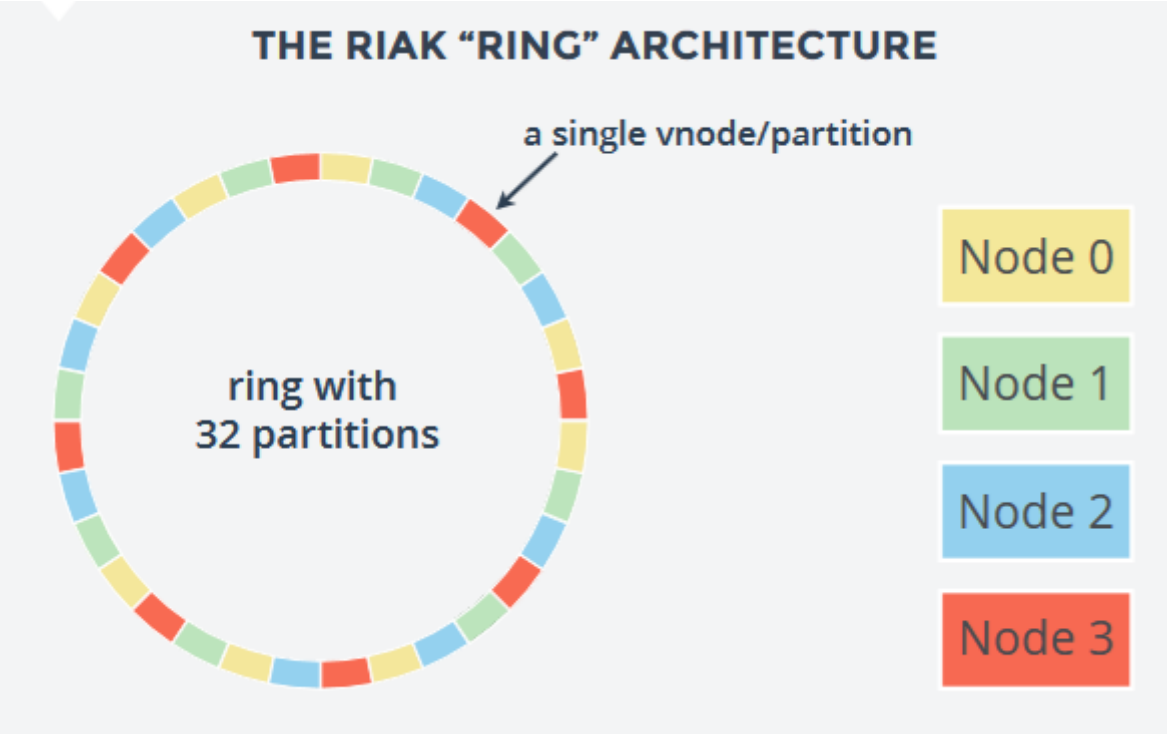
2^{160} hash values this guarantees that there is no 2 virtual nodes that collide

What is wrong with the vnode ring picture?

green node is always followed with orange node so if we remove green one then orange one will need to take all the load or if we skip orange one if it is replicated one then and so is blue node, then pink node will take all the workload

eva.knirsch@kiu.edu.ge

RIAK Cluster



we want virtual nodes to be distributed randomly

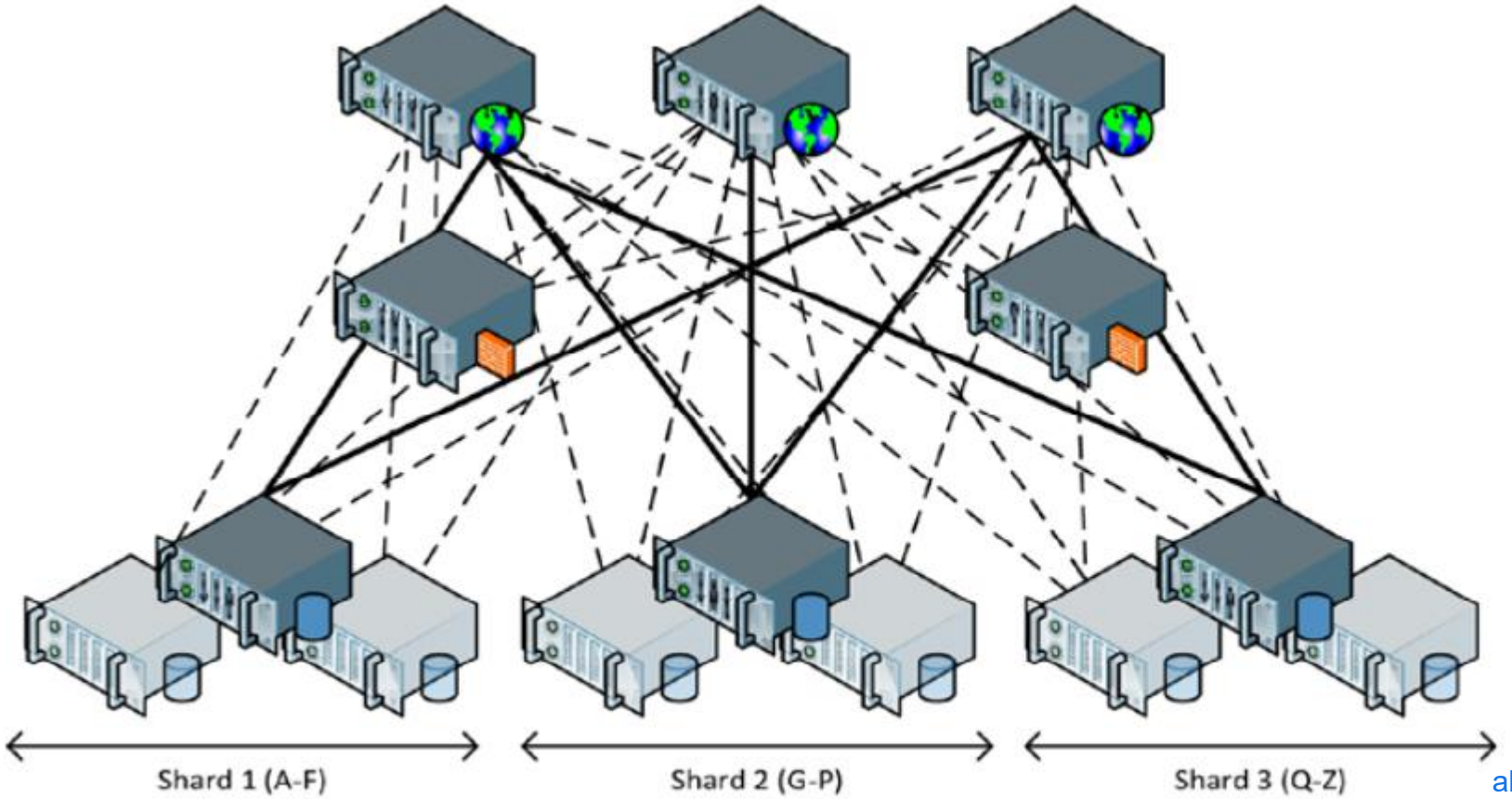
if we take down blue node then firstly red node takes workload, then yellow, then red, then yellow agian and so on.

eva.knirsch@kiu.edu.ge

Partitioning / Sharding

Single-Leader Database System
MongoDB

Sharding



Scaling with 3 web servers, 2 Memcached servers, and 3 database shards with 2 read-only database slaves per shard

all of these 3 shards are replica sets

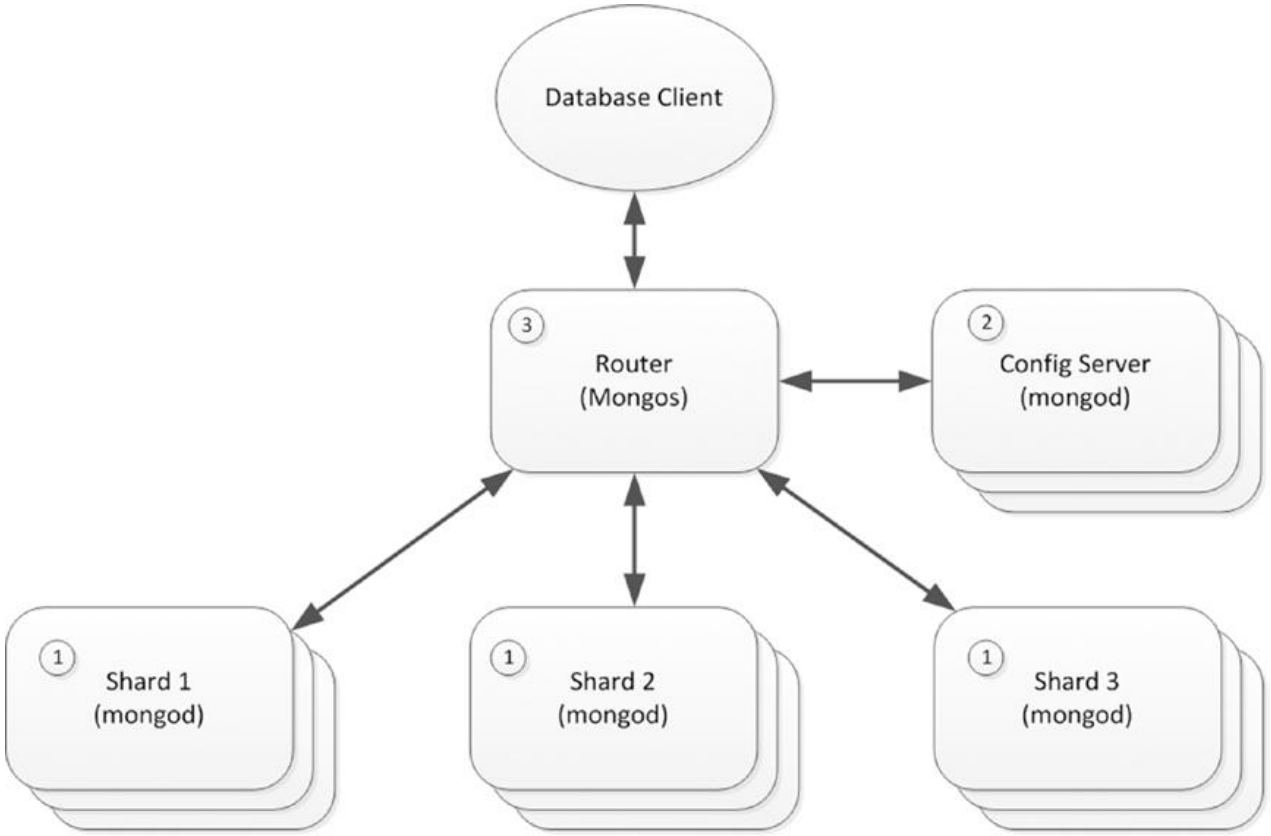
you can shard but only within replica set

Even though replication and sharding are different concepts, they typically are combined together.

Hal], p. 42

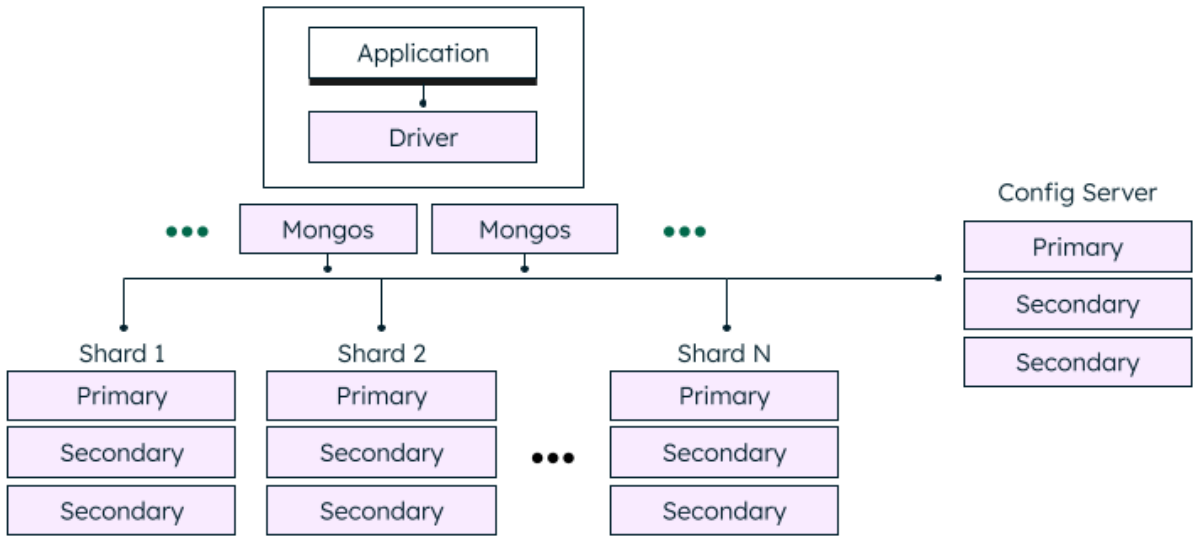
knirsch@htw-berlin.de

MongoDB Sharding



- The data is partitioned into subsets. These subsets are sometimes called partitions. MongoDB calls them chunks.
- Nodes (shards) are physical or virtual servers. A shard can hold multiple chunks.
- Each shard must be deployed as a replica set.
- The recommendation is to have more chunks than shards. What is the advantage of having multiple chunks (partitions) on one server node?

MongoDB Sharding

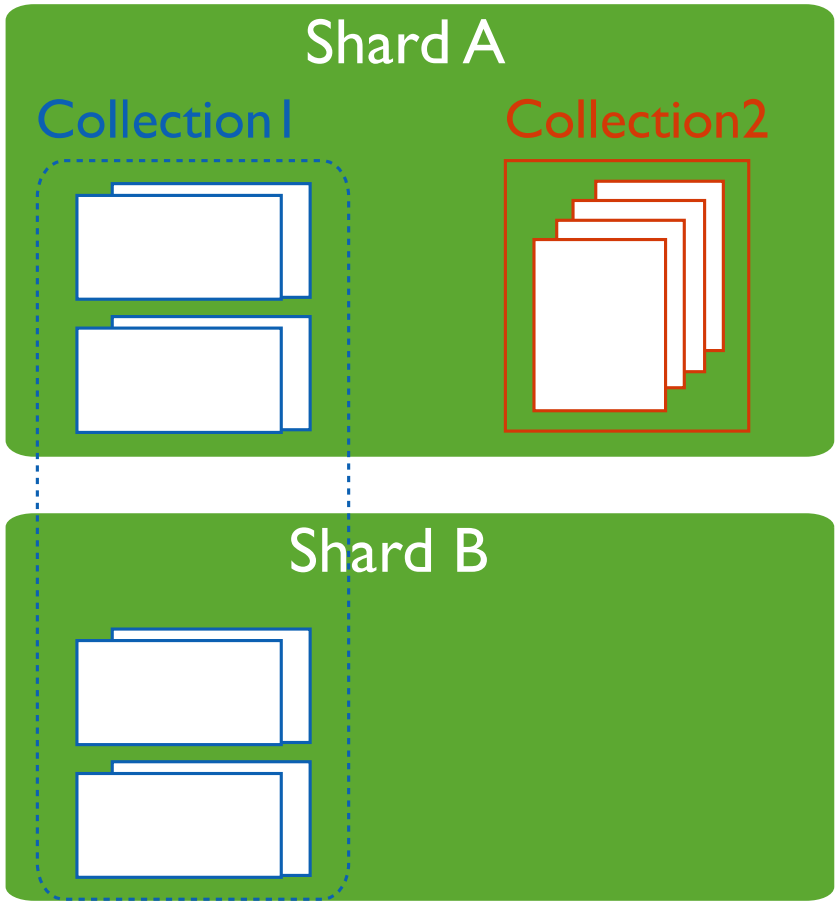


The Config Server stores the configuration settings of the whole cluster and must also be deployed as a replica set.

Each shard must be deployed as a replica set.

eva.knirsch@kiu.edu.ge

Mongo Shards – Primary Shard – Chunks and Shards

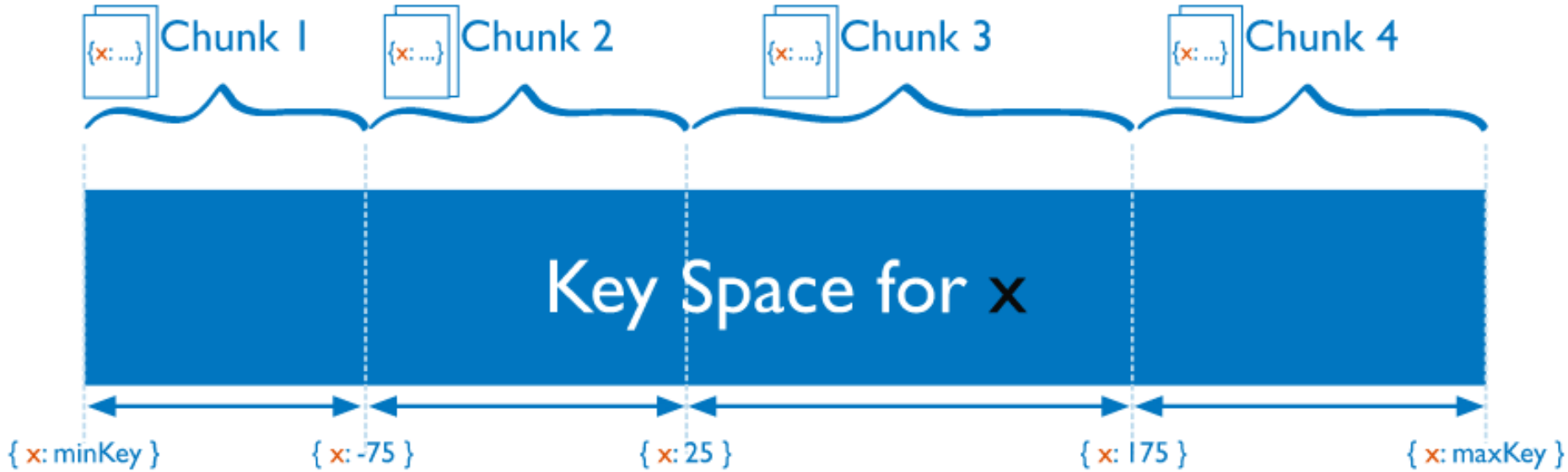


Each database in a sharded cluster has a primary shard that holds all the un-sharded collections for that database. The primary shard has no relation to the primary in a replica set.

MongoDB partitions collections into chunks.

eva.knirsch@kiu.edu.ge

Range-Key example



Each chunk has a inclusive lower and exclusive upper range based on the shard key.

eva.knirsch@kiu.edu.ge

Selection of a Range-Shard Key

The shard key determines which data item (document) ends up on which partition and which node. It controls the distribution of data.

MongoDB
https://www.mongodb.com/docs/manual/core/sharding-shard-key/

"The shard key is either a single indexed field or multiple fields covered by a compound index that determines the distribution of the collection's documents among the cluster's shards.

How are the ranges determined?

1st option: MongoDB divides the span of shard key values (or hashed shard key values) into non-overlapping ranges ..... Each range is associated with a chunk.

2nd option: Developer / DBA

A "good" shard key needs to distribute documents evenly and fitting to query patterns. It must be possible to determine ranges that distribute data evenly.

knirsch@htw-berlin.de

MongoDB Sharding

Create index on the collection field:

```
use <database>
db.<collection>.createIndex({"<shard key field>":1})
```

Enable Sharding of the database:

```
sh.enableSharding("<database>")
```

Shard the collection :

```
sh.shardCollection("<database>.<collection>", { "<shard key field>" : 1, ... } )
```


knirsch@htw-berlin.de

Good Range-Key for Teacher Collection?


```
{
  "_id": {  "$oid": "640d683998ac446f58fe76f4" },
  "teacherID": "2",
  "name": "Doe",
  "mail": "doe@we-are.xx",
  "DoB": "1998-01-15",
  "gender": "f",
  "Country": "Georgia"
  "postalCode": 4600,
  "remark": null,
  "subjects": ["Java","Neo4J" ],
  "references": [{ "fName": "Donald",
                   "recommendation": "Great teacher" },
                   { "fName": "Flower",
                   "recommendation": "learned a lot" " }
                ]
}
```

this would be good for hash key not range key

teacherID is incremental which does not distribute evenly and leads to hotspot, all new documents go to the end of the range, hitting the same shard repeatedly.

it is okay as a range key but we don't usually have to query teachers starting name from A to E for example.

Unique per teacher so might seem good but not used in range queries because it doesn't align with common access patterns.

Potential candidate if queries are made by date ranges

gender does not have enough different values so it is not good shard key.

good but one country might dominate the data

adjust range to frequency

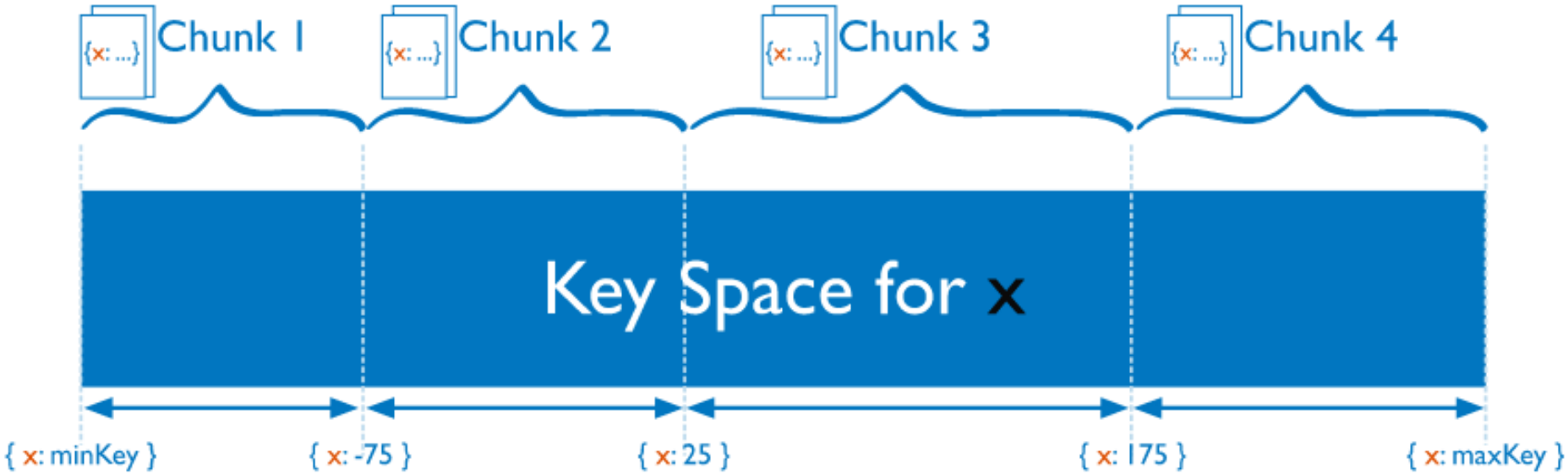
we can't use elements from array as range key because array elements are not treated as individual keys for indexing

Shard key for teacher collection that distributes data roughly evenly?

country and postalcode

eva.knirsch@kiu.edu.ge

Shard Key for Teacher Collection



0100: minKey

Shard Key example postalCode

9990:maxkey

The cardinality of a shard key determines the maximum number of chunks the balancer can create. Each unique shard key value can exist on no more than a single chunk at any given time.
The frequency of the shard key represents how often a given shard key value occurs in the data. If the majority of documents contain only a subset of the possible shard key values, then the chunks storing the documents with those values can become a bottleneck . Example: PostalCodes with 01XX (Tbilisi) are much for frequent then other postalCodes.

eva.knirsch@kiu.edu.ge

Shard Key Modifications

Chunk1

0100:
- ID1
- ID15
- ID40

0101
- ID3
- ID6
- ID..

What happens,

if the value of a shard key changes (e.g. teacher moves to a different location, gets a different postalCode)?

document must be moved from one shard to another since the shard key determines the physical location of data. This triggers a delete from the old shard and insert into the new shard.

If a value for the shard key is missing because shard key is not a mandatory field?

Many databases like MongoDB require the shard key to be present when inserting or updating documents. If it's missing, the operation fails.

eva.knirsch@kiu.edu.ge

Hashed Shard Key

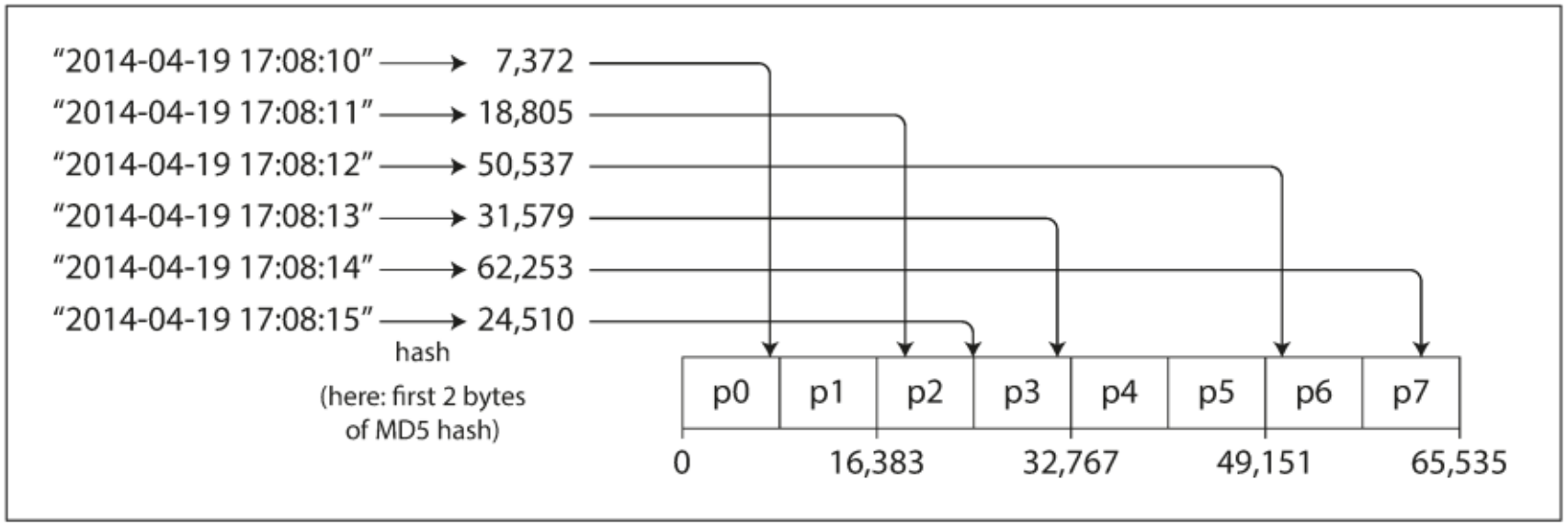
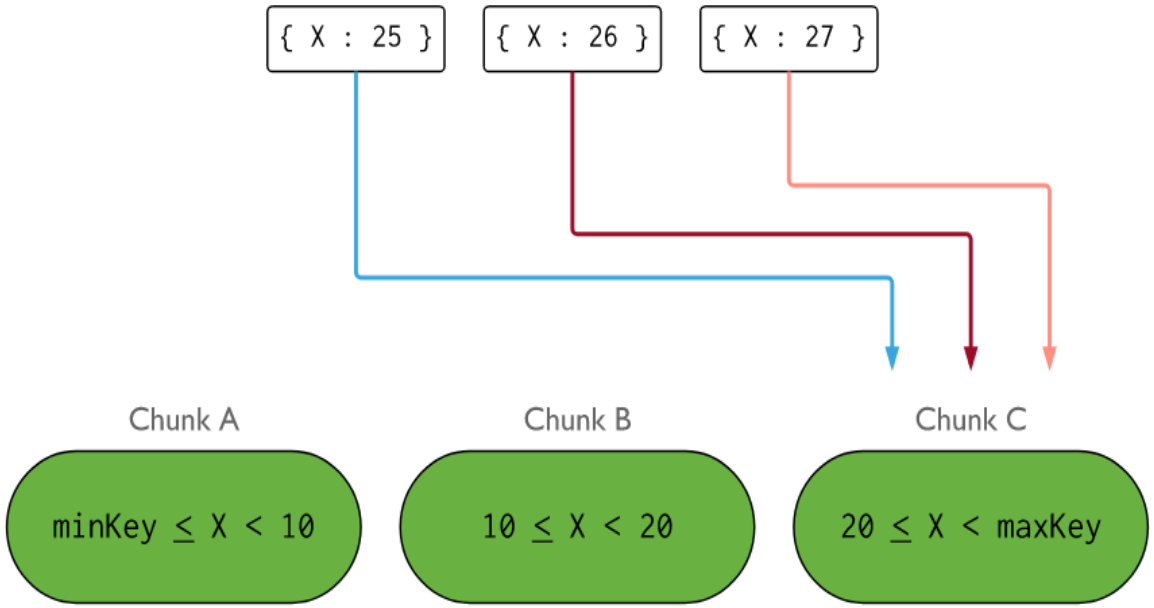


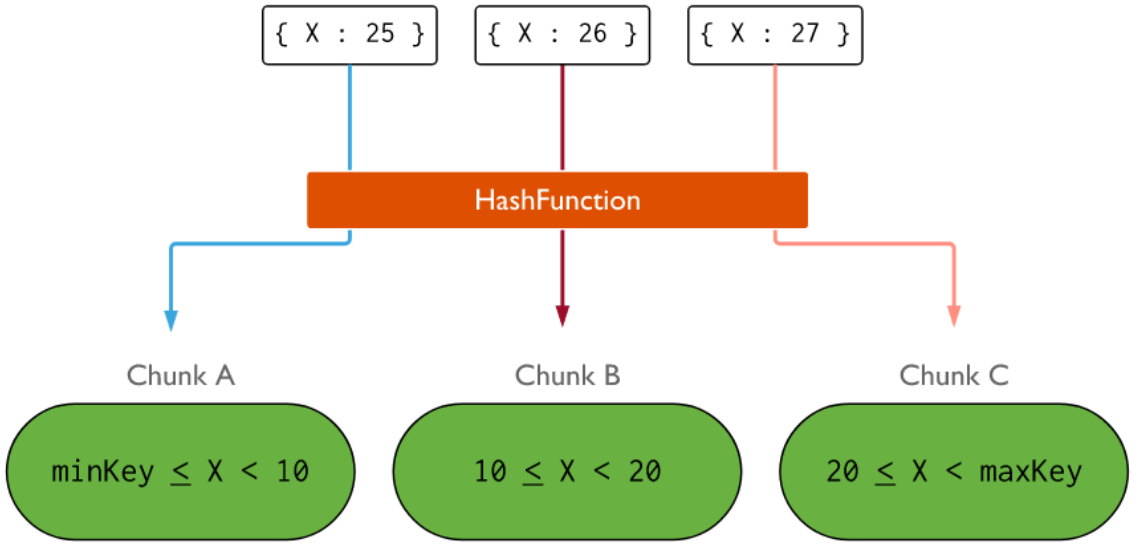
Figure 6-3. Partitioning by hash of key.

Hashed Shard Key
- used when primary concern is even distribution
- used for automatic sharding
- hash ranges are assigned to partitions (shards)
- Objects that - with a range key - would be stored close together are now randomly distributed.

[Kl], 204

eva.knirsch@kiu.edu.ge





eva.knirsch@kiu.edu.ge