RTB HOUSE

11.03.2024

Data storage in distributed systems - part II

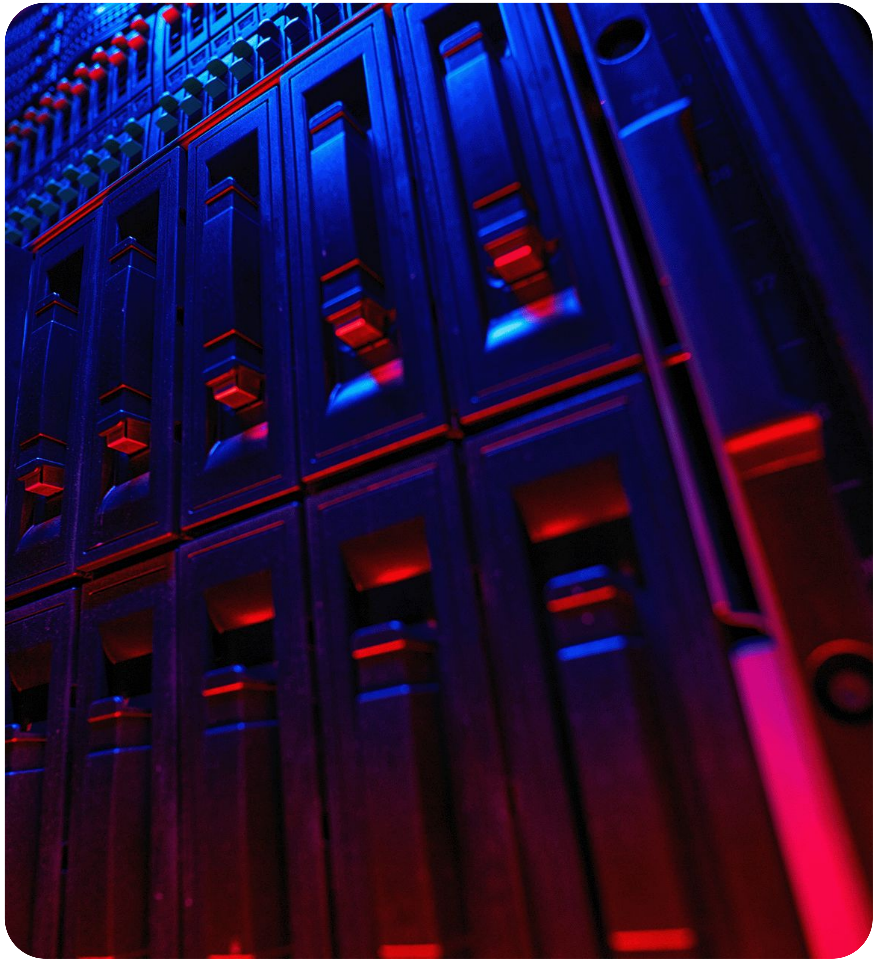
Piotr Jaczewski
RTB HOUSE

What will this lecture be about?

In this part of the course we will focus on NoSQL databases and their usage in distributed systems.

We will cover the topics of:

1. Data models in NoSQL storages.
2. Implementation details of selected NoSQL storages.
3. Data formats and schema evolution in distributed systems.

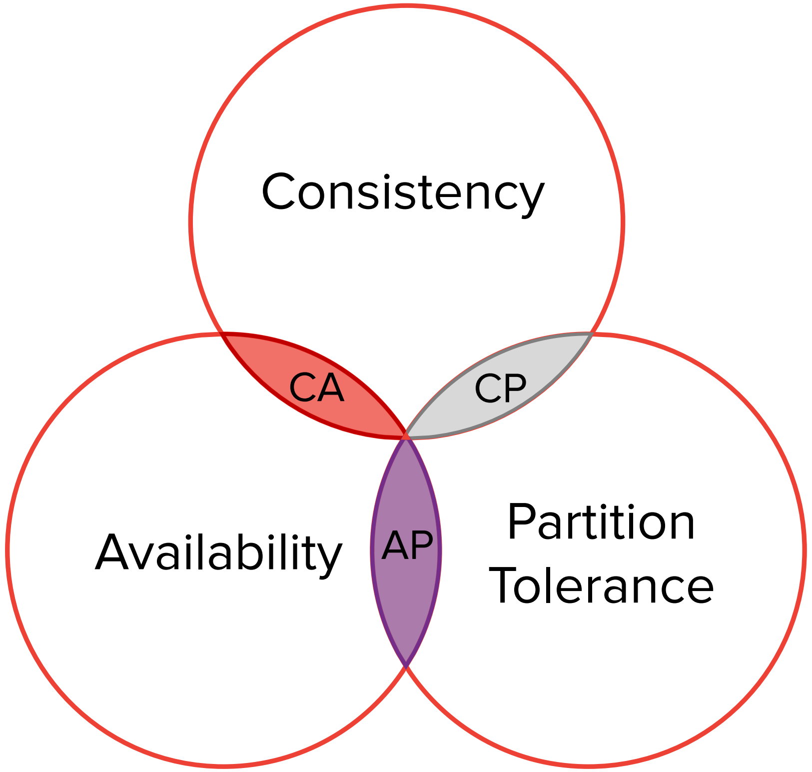


NoSQL Databases

Some of the common features of the NoSQL databases:

- Designed to easily scale horizontally.
- Usually don’t use strict schemas.
- Focused on data aggregates.
- Don’t use standardized SQL, but some have query languages.
- Mostly support limited transactional capabilities (like multi-object transactions), due to running on clustered environment.
- Provide various options for data consistency.

CAP Theorem

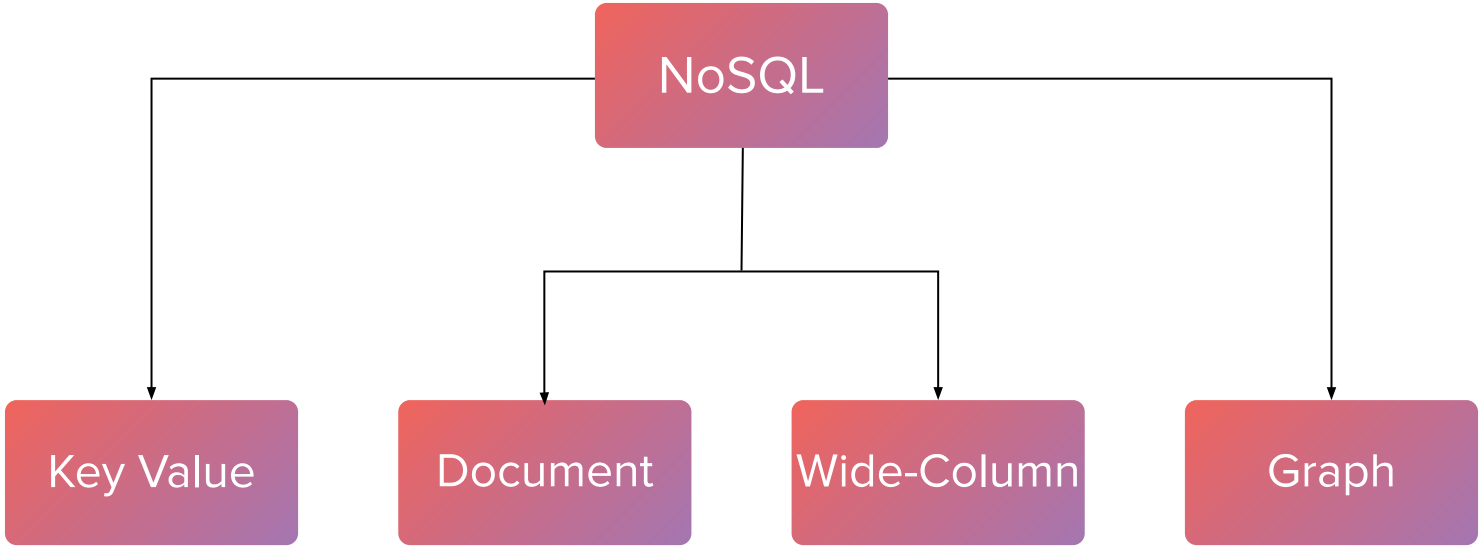


CAP Theorem - critique

A great article by Martin Kleppmann:

https://martin.kleppmann.com/2015/05/11/please-stop-calling-databases-cp-or-ap.html

NoSQL Data Models



Key Value Model

- May be viewed as a generalization of a hash table with put/get/remove operations.
- Data type agnostic - the understanding of stored value is the responsibility of client applications.
- Some implementations may include some built-in data types like maps, sets, counters.
- None or limited querying capabilities
- Offer great performance.

MEMCACHED

redis

AEROSPIKE

FOUNDATIONDB

Document Model

- Can be considered as a subtype of key-value databases.
- Have some awareness of the data stored.
- The document format is usually JSON, BSON, XML, etc.
- The documents doesn’t have to be of the same schema in a table/collection.
- Slightly improved querying capabilities.
- Support for secondary indexes.
- Allow partial document update.

Wide-Column Model

- Another variation of key-value model.
- No relations between tables.
- Map keys to rows and rows consist of groups of columns.
- Groups of columns are called column families.
- Usually each row may have a varying number of columns within a column family.
- Nonexistent columns do not take storage space.
- Some implementations feature SQL-ish query language.

APACHE HBASE Logical Data Model

Row key

Column family

Column qualifier

Cell

ROW KEY	SOCIAL NETWORK			ACTIVITY		
	SN NAME	USER ID	PAGE ADDRESS	TYPE	DATE	TEXY
Boboo‰ a_banana	Boboo	a_banana	Boboo.com/ a_peach	Like	6/3/2018	
Boboo‰ a_pineapple	Boboo	a_pineapple	Boboo.com/ a_banana	Comment	6/13/2018	Nice pic, bro!
Boboo‰ a_watermelon	Boboo	a_watermelon	Boboo.com/ a_pineapple	Comment	6/10/2018	Hey, that's my jacket! I've been looking all over!
Chiching‰ a_cucumber	1234	a_cucumber	Chiching.com/ a_kohlrabi	Comment	5/25/2018	Wow! What a bike!
Chiching‰ a_kohlrabi	1234	a_kohlrabi	Chiching.com/ a_kohlrabi	Comment	5/25/2018	Yeah, I know :D

Logical Data Model

Row key

Column family

Boobo Activity

1673Xy035	SN_NAME	USER_ID	PAGE_ADDRESS	ACTIVITY_TYPE	ACTIVITY_DATE
	Boboo	a_banana	Boboo.com/ a_peach	Like	6/3/2018

Column

47aBb096	SN_NAME	USER_ID	PAGE_ADDRESS	ACTIVITY_TYPE	ACTIVITY_DATE	ACTIVITY_TEXT
	Boboo	a_pineapple	Boboo.com/ a_banana	Comment	6/13/2018	Nice pic, bro!

KK78B9012	SN_NAME	USER_ID	PAGE_ADDRESS	ACTIVITY_TYPE	ACTIVITY_DATE	ACTIVITY_TEXT
	Boboo	a_watermelon	Boboo.com/ a_pineapple	Comment	6/13/2018	Hey, that's my jacket! I've been looking all over!

```
cqlsh> select * from friends;
```

name	friend	email
Jo	George	George@gmail.com
Jo	Guy	Guy@gmail.com
Jo	John	John@gmail.com
Guy	Jo	Jo@gmail.com
Guy	John	John@gmail.com

```
CREATE TABLE friends
(name. text,
friend text,
email  text,
PRIMARY KEY (name, friend) )
```

Cassandra CQL view of column family

Friends column family

Rowkey = Guy

Jo:email	John:email
Jo@gmail.com	John@hotmail.com

Rowkey = Jo

John:email	George:email	Guy:email
John@hotmail.com	John@hotmail.com	guy@gmail.com

Actual column family structure

Graph Model

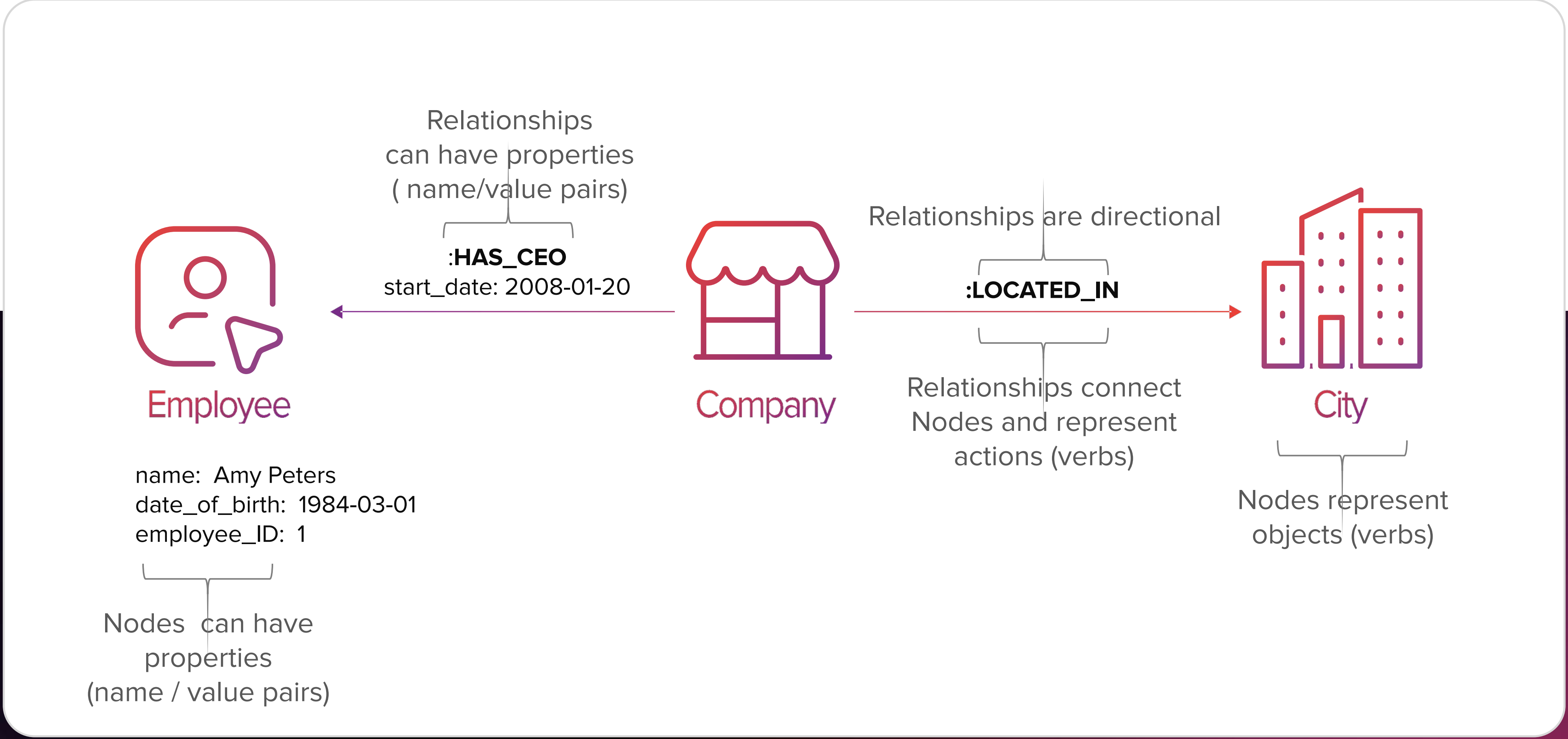
- Focused on the relationship between data entities.
- Store both entities and edges between them.
- Both entities and edges can have their custom properties.
- Support querying and traversing the object graphs.
- Traversing the graph is very fast.
- For specific graph related scenarios.

neo4j

Amazon Neptune

ArangoDB

JanusGraph


Relationships
can have properties
(name/value pairs)
:HAS_CEO
start_date: 2008-01-20
Relationships are directional
:LOCATED_IN
Employee
Company
City
Relationships connect
Nodes and represent
actions (verbs)
name: Amy Peters
date_of_birth: 1984-03-01
employee_ID: 1
Nodes represent
objects (verbs)
Nodes can have
properties
(name / value pairs)

NoSQL Storages Architecture

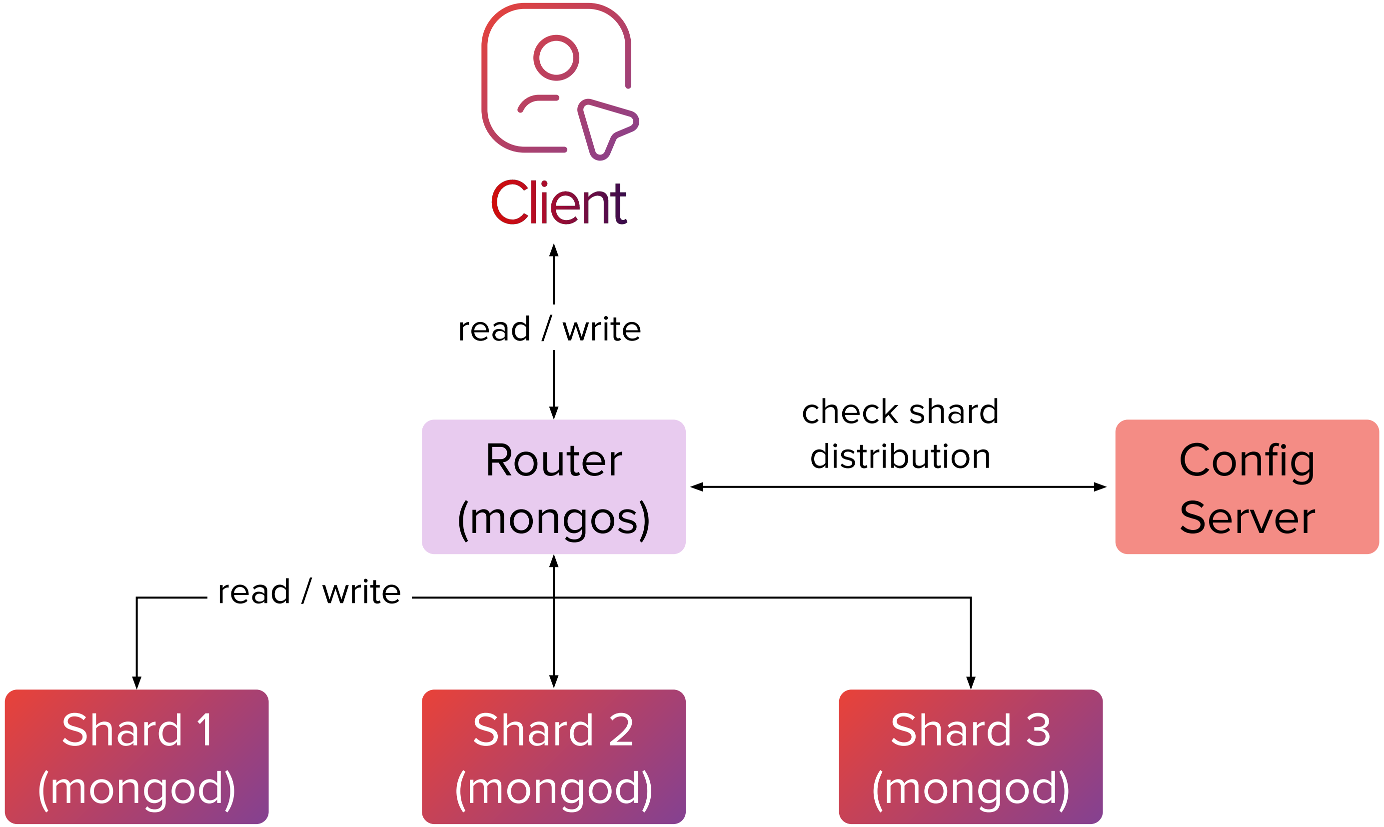
In the next part of the lecture we will discuss the high level architecture of the following NoSQL solutions:

- MongoDB
- Apache HBase
- Apache Cassandra
- Aerospike DB

MongoDB

- Document oriented database - documents in JSON (BSON internally).
- Support for large data sets.
- Supports searching by fields, range queries and using regular expressions.
- Supports indexing/secondary indexes.
- Dedicated clients/REST API.
- Mature and production ready.

MongoDB Architecture



MongoDB Sharding

Methods of sharding:

- Range based sharding - may result in shard imbalance.
- Hash based sharding - more even value distribution.
- Tag-aware sharding - explicitly determine groups of shards on which range of documents will reside.

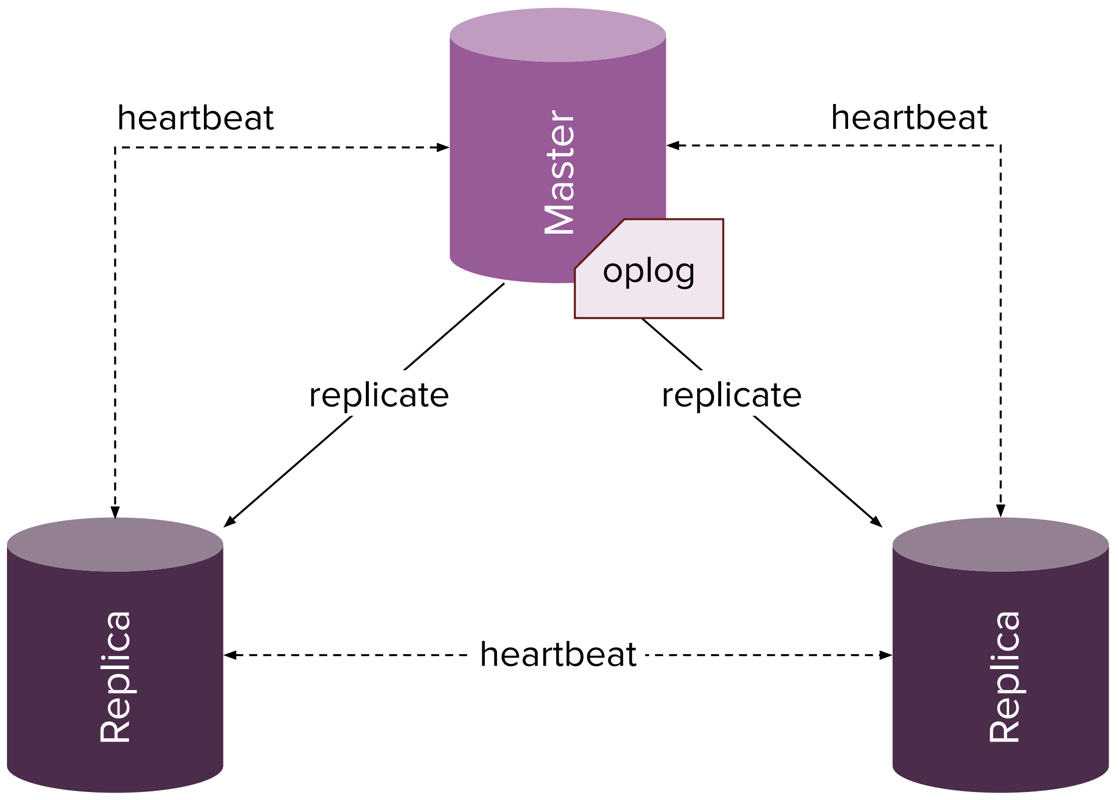
- MongoDB will periodically assess the balance of shards across the cluster.
- Rebalance operation will move chunks between shards.
- Chunks contain adjacent values of shard keys.

RTB HOUSE

MongoDB Cluster Replication

- Sharding is combined with replication.
- Each shard is replicated across a replica set.
- Master accepts writes which are then applied to replicas via oplog collection.
- Master node is determined by an election.
- To become a primary a node must be able to contact more than half of replica set.
- Election is based on priority set by administrator and timestamp of the last operation.

Replica set



MongoDB Concurrency/Consistency

- Supports (since version 4.2) ACID transactions on multiple documents between shards.
- Pessimistic concurrency control at global, database, collection levels (explicit locks)
- Optimistic concurrency control at document level (either manual or via transactions).

- Consistency is tuneable:

➢ Write concern - the client may be ordered to write synchronously only to primary or also to a specified number of replicas - strong consistency.

➢ Read preference - the client may specify whether the read request is routed to primary or secondary replica.

➢ Read concern - the client may choose to read only replicated data that is durable or read the newest data that may not be yet replicated and thus can be lost.

MongoDB Usage Considerations

Reasons to use:

- When a strict schema is a problem.
- Use for CRUD applications, Web APIs storage, Content Management Systems.
- Straightforward architecture.
- Rather easy maintenance and configuration.

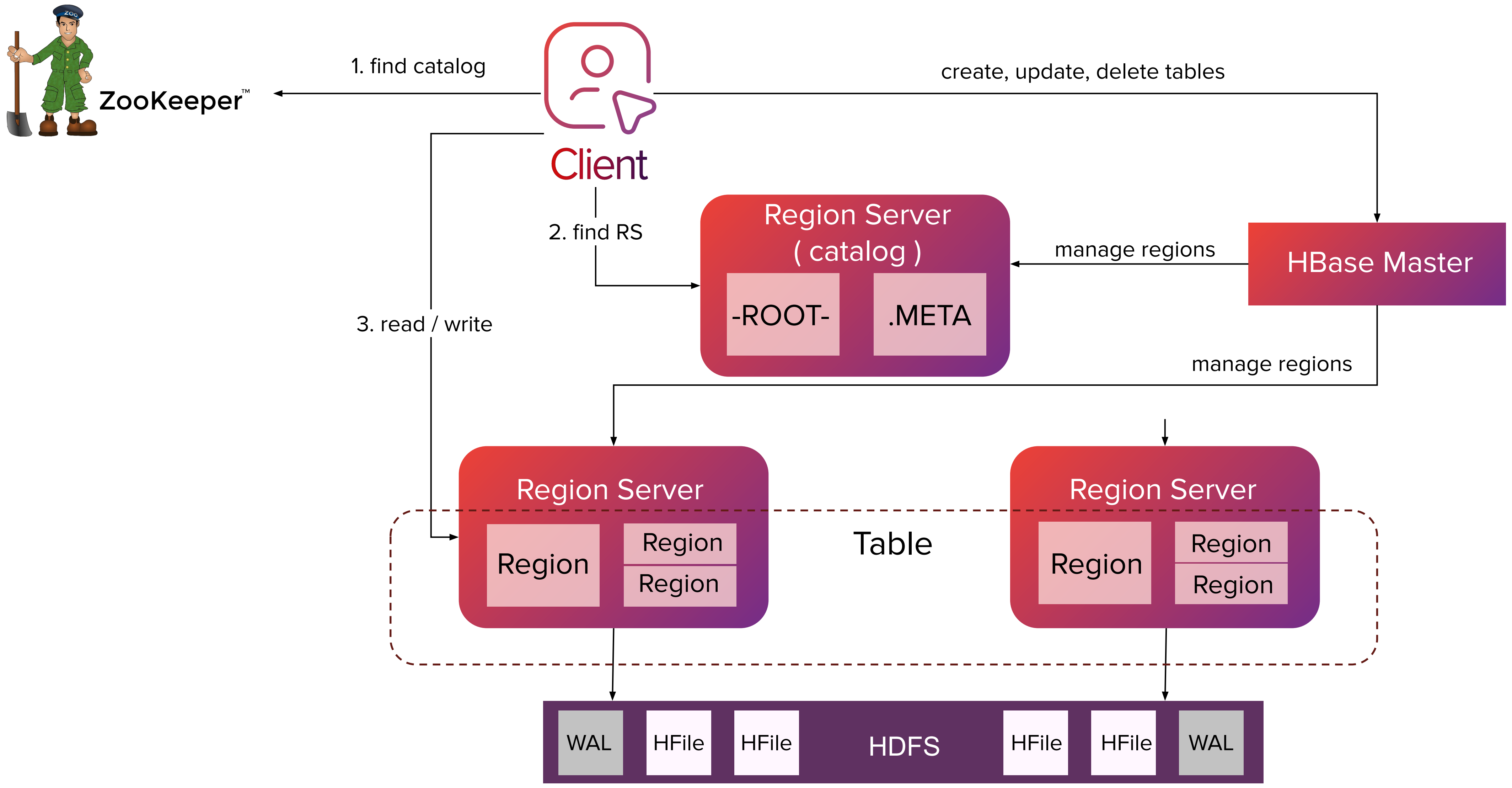
Reasons not to use:

- Be careful with relationships between documents - no constraints.
- No rigid schema is not always your friend - custom versioning patterns must be implemented by application.

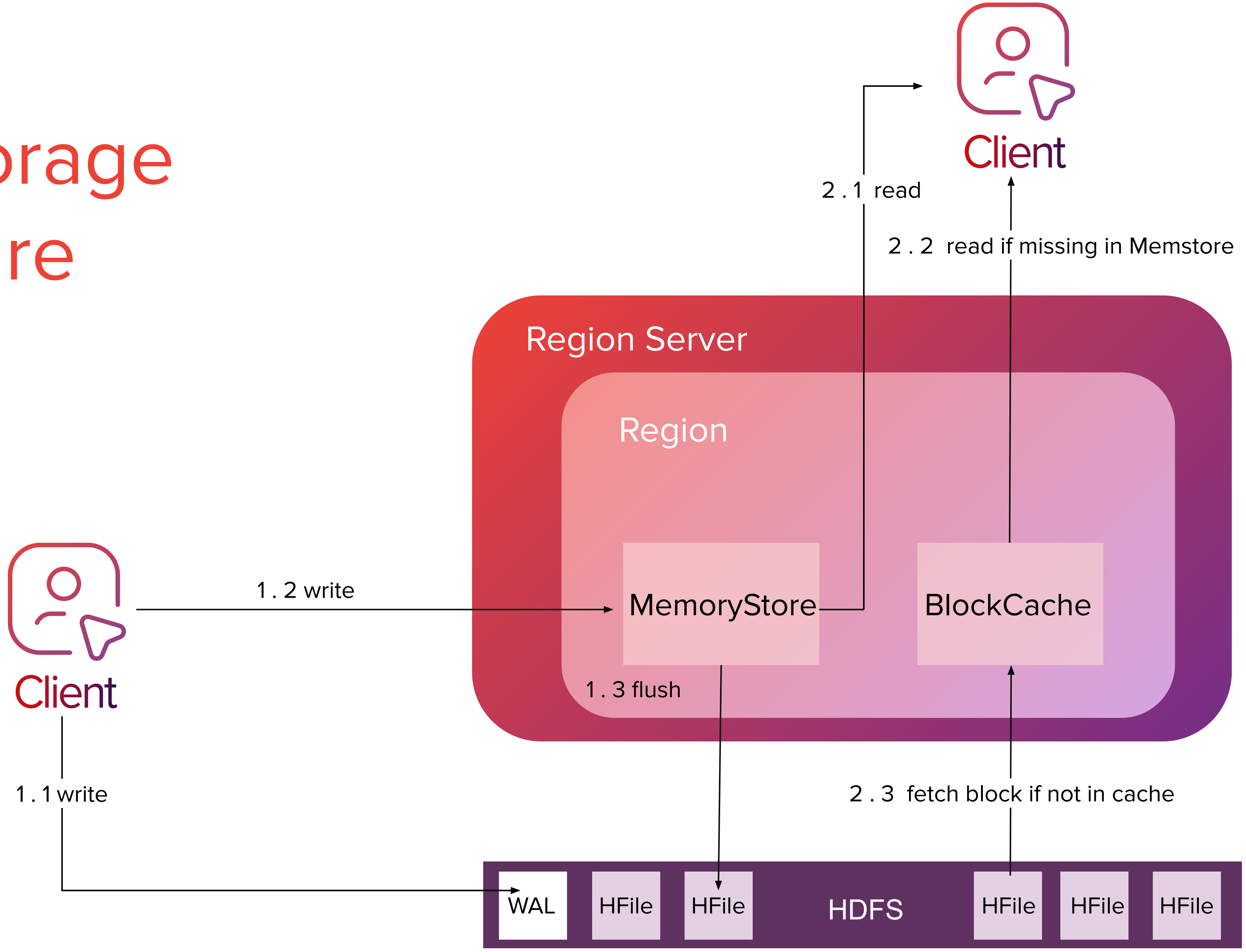
HBase

- Wide-Column oriented.
- The data model is strictly based on the original Google BigTable specification.
- Provides random access database services on top of HDFS.
- Does not bother with data redundancy or disk failures - these are handled by HDFS.
- Can be easily accessed via MapReduce jobs on Hadoop.
- Advanced querying via Hive or Apache Phoenix.

HBase Architecture



HBase Storage Architecture

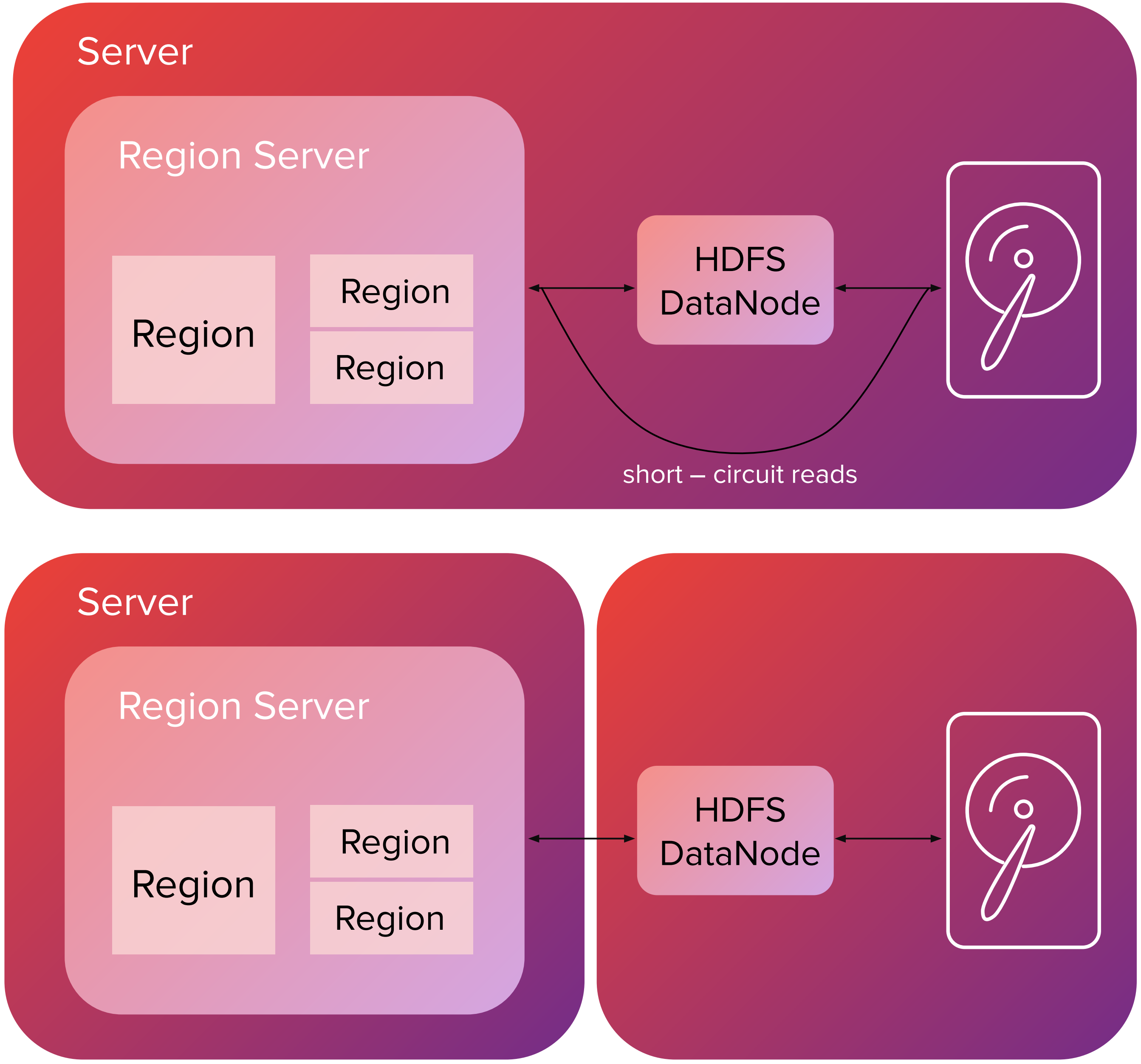


HBase RegionServer

- Regions are equivalent to range based shards.
- HBase Master will evaluate the balance of regions across all RegionServers.
- Regions can be splitted when becoming too large and can be relocated to other RegionServers by the HBase Master.

- RegionServers are co-located with the Hadoop HDFS DataNodes for good data locality.
- Data locality can be broken by RegionServer rebalance, failovers.
- Data locality is usually restored when the underlying HFiles are compacted.

HBase RegionServer



HBase Concurrency/Consistency

- No multi-object transactions, only atomicity of operations at row-level.
- Row-level locking for every update, even when mutation crosses multiple Column Families.
- Reads are not blocked by write operations - concurrent read will see the previous version before update - Multiversion Concurrency Control at row-level.

- Table scans do not use MVCC - all writes committed before the scan started will be visible, as well as those committed after.
- Lost writes are prevented via checkAndPut/Mutate/Delete family of functions (manual optimistic locking).

HBase Concurrency/Consistency

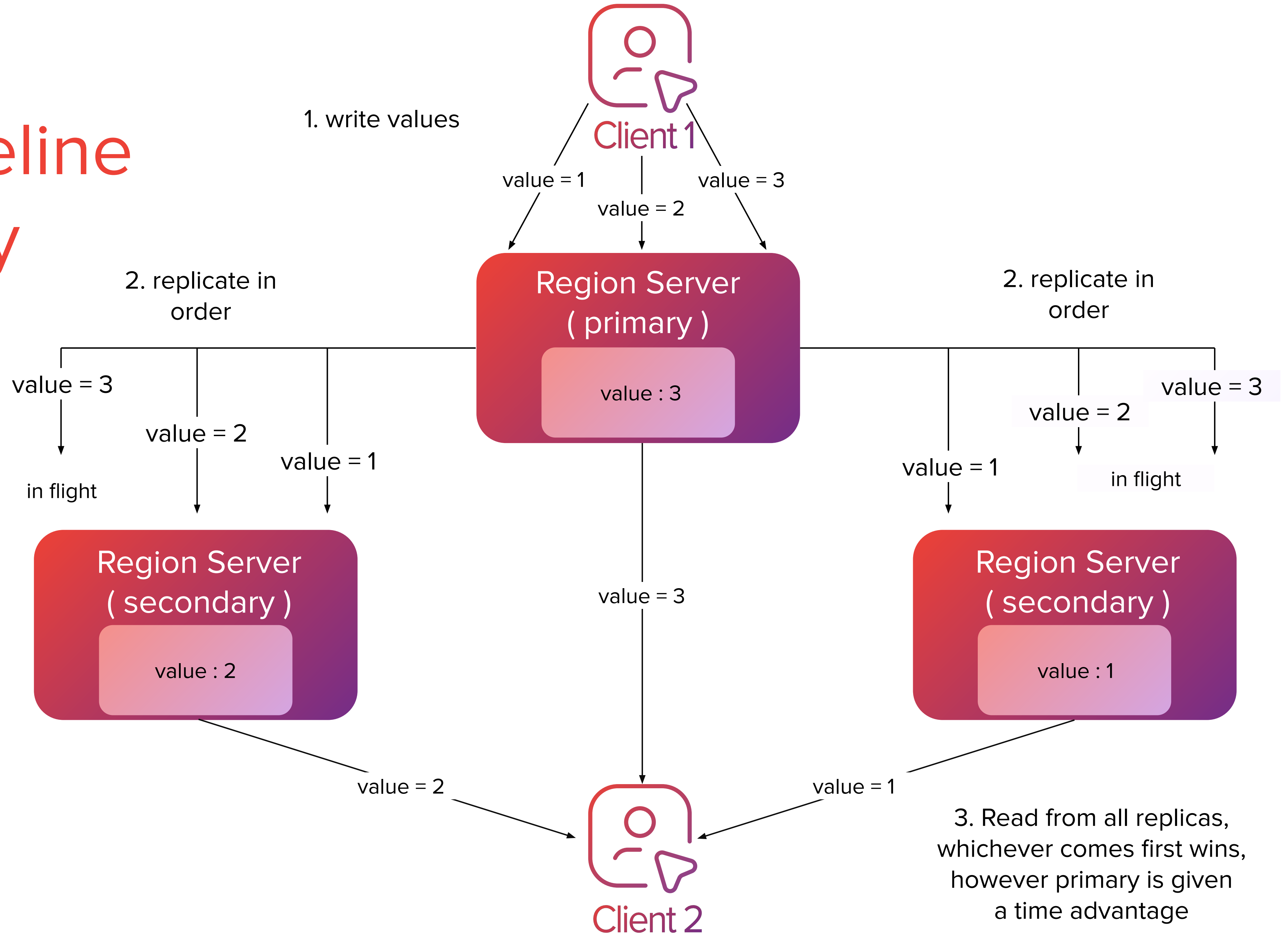
- Secondary replicas for RegionServers provide availability for read operations.
- Until failover is done the affected region is only available for reads.
- Thus secondary RegionServers are read only.

- Secondary RegionServers follow the primary and see only committed updates.
- Secondary RegionServers do not make their copy of the HFiles - no storage overhead, the data is kept in BlockCache or read from primary HFiles.
- Replica RegionServers memory state can be refreshed from primary HFiles at a interval - higher chance of stale read.

- Replica RegionServers memory state can be asynchronously updated via WAL replication - lower chance of stale reads.
- Reads from replica RegionServers can be also allowed via Timeline Consistency.

RTB HOUSE =

HBase Timeline Consistency



HBase Usage Considerations

Reasons to use:

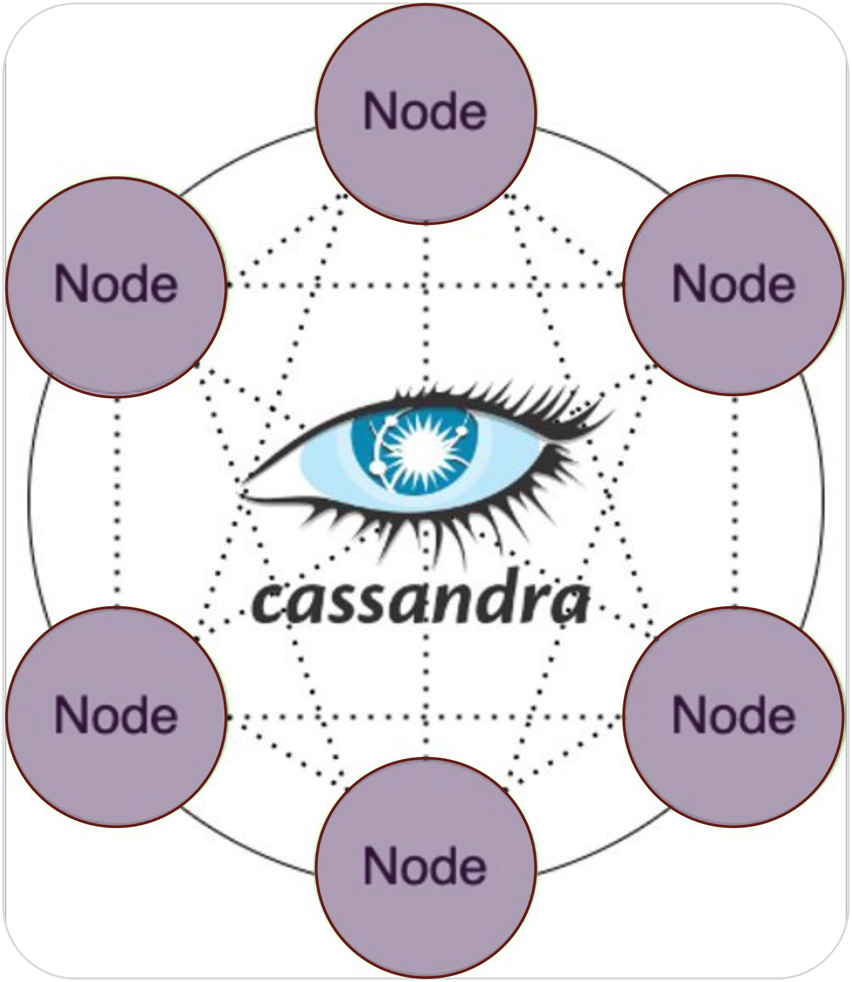
- If a true, BigTable like wide column data model is required.
- If MapReduce jobs must be run on data.
- If there is an existing Hadoop/HDFS cluster.
- If there are billions of potential rows.

Reasons not to use:

- Complex multi-element architecture.
- Painful operations and maintenance.
- High performance requires a lot of memory for BlockCache.
- A myriad of cumbersome dependencies for client libraries.

Cassandra

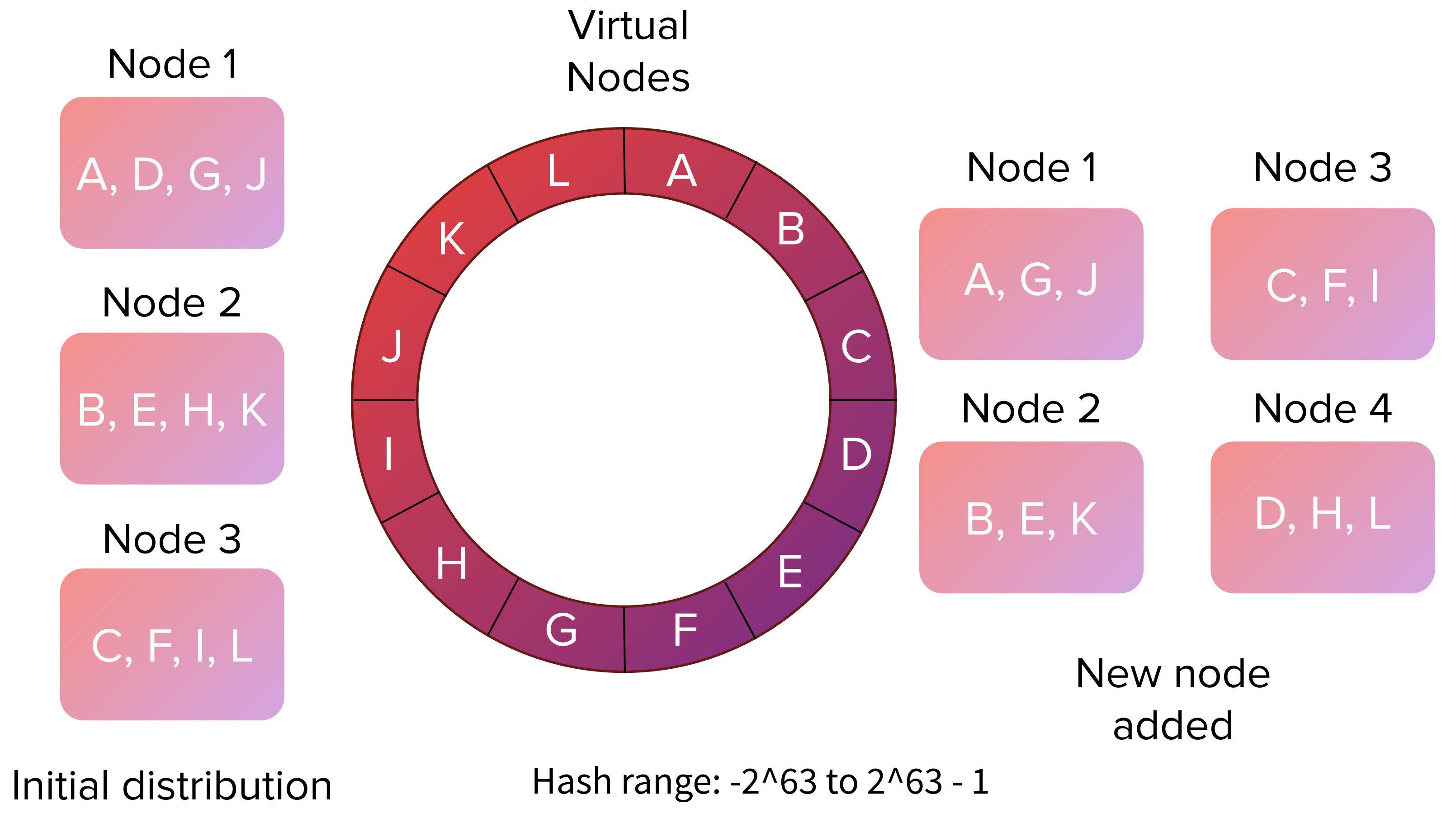
- Wide-column oriented (implicitly).
- The clustering mode is based on the concept derived from Amazon Dynamo.
- Linear scalability, you can expand the cluster or shrink horizontally whenever needed, using commodity hardware with no downtime.
- Each node in the cluster can work as a cluster coordinator and perform all operations.
- Leaderless architecture, uses gossip protocol to determine the cluster state.



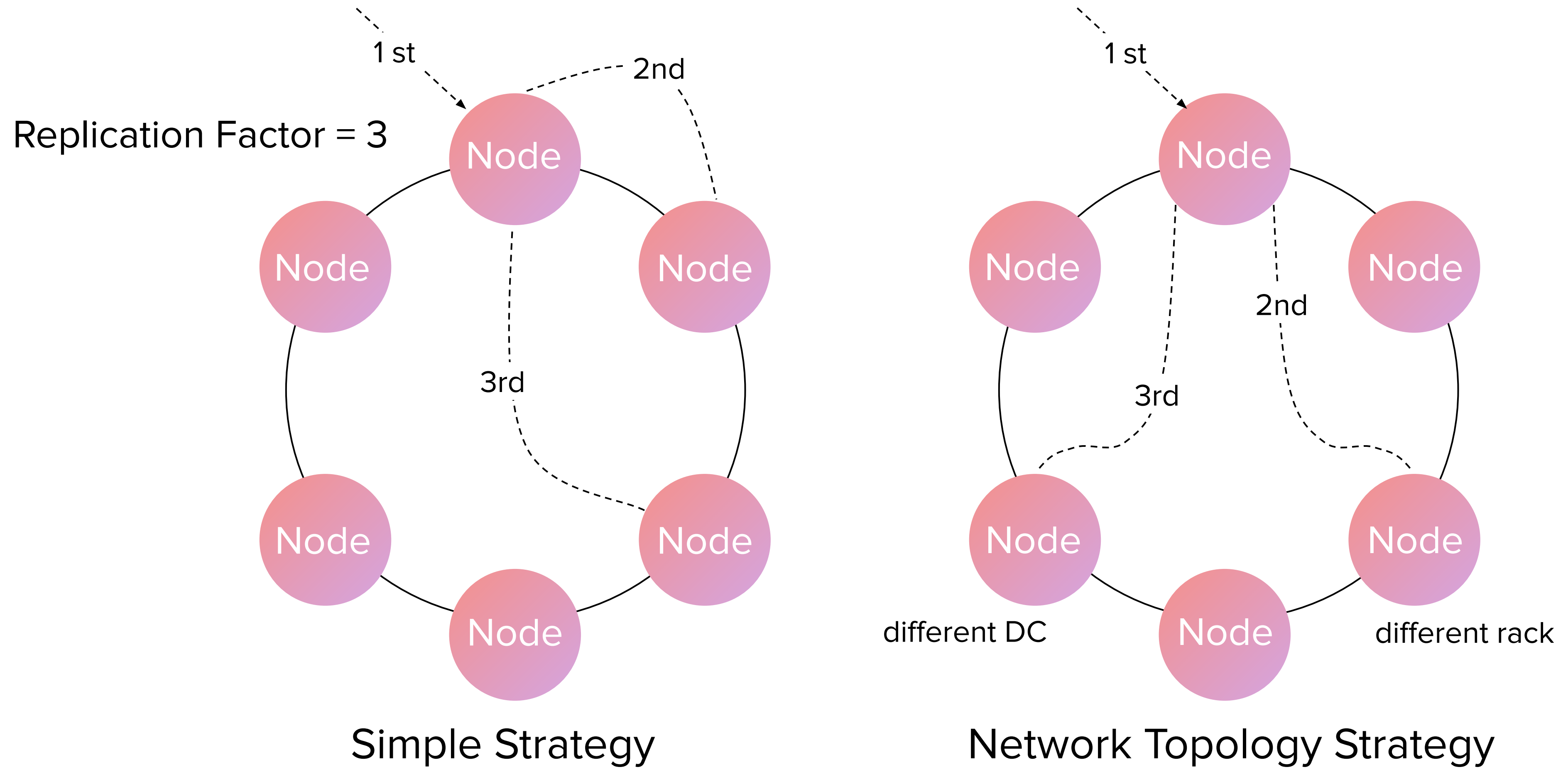
Cassandra Consistent Hashing

- Cassandra distributes data throughout the cluster by using consistent hashing technique.
- Each node is allocated a range of hash values and data is placed on the node if the primary key hash lies within the nodes range.
- If the number of ranges is equal to the number of nodes then addition or removal of node will require a lot of data movement and can result in a cluster imbalance.
- So we introduce a lot more ranges mapped to virtual nodes.
- Virtual nodes mapped to physical nodes, so that the addition/removal of node will cause few ranges to move and will leave the cluster balanced.

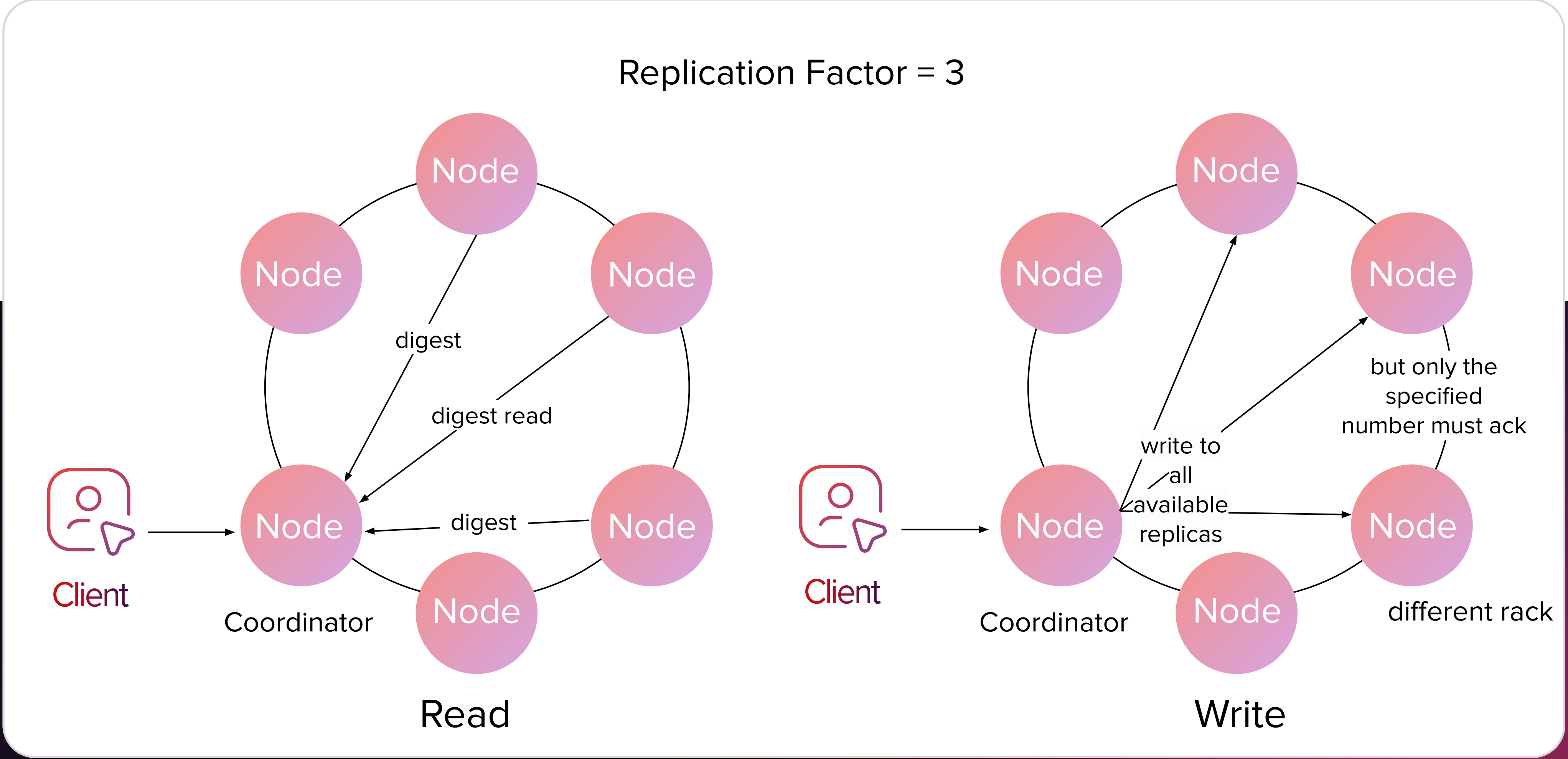
Cassandra Consistent Hashing



Cassandra Replication



Cassandra Reads / Writes



Cassandra Consistency

- Follows the Amazon Dynamo model with tunable consistency for writes and reads.
- Write Consistency levels:
  - ALL - all replicas must acknowledge the write
  - ONE\TWO\THREE - the specified amount of nodes must acknowledge the write.
  - QUORUM - majority of replica nodes must acknowledge the write.
  - ANY - any node can acknowledge, even if the node is not responsible for storing the particular data.

- Write Consistency levels in multi DC scenario:
  - LOCAL_QUORUM - majority of replica nodes in a local DC must acknowledge the write.
  - EACH_QUORUM - majority of replica nodes in each clustered DC must acknowledge the write.

Cassandra Consistency

Read Consistency levels:

ALL - all replica nodes are polled for the data.

ONE/TWO/THREE - reads will be polled from the specified number of replica nodes.

QUORUM - read completes after majority of nodes have returned the data.

LOCAL_ONE/LOCAL_QUORUM/EACH_QUORUM - analogous levels for multi-DC setup.

Cassandra Consistency Levels

Write\Read	ONE	QUORUM	ALL
ONE	High performance and availability, lowest consistency.	Fast writes with high availability, moderate consistency.	Fast writes with high availability, slow reads with consistency and low availability.
QUORUM	Fast and highly available reads with moderate consistency.	Medium performance, high availability and strict consistency.	Slow reads with low availability and strict consistency.
ALL	Slow writes with low availability, fast and consistent reads.	Slow writes with low availability, consistent available reads of medium performance.	Strict consistency, lowest performance and availability.

Cassandra Consistency Repair

- If write consistency level is not set to ALL, inconsistencies may appear due to the node downtimes, network partitions etc.
- Hinted handoffs - a technique where a node will store an update for a temporarily unavailable replica node. If the failed node is restored, it will receive the update.
- Write consistency level ANY will write hinted handoff even if all replicas are down.
- Hinted handoffs are deleted after some time.
- Read repair - if hinted handoffs were deleted, the normal read operation may be used to fix consistency
- After returning the value to the client the coordinator node writes the correct data to the inconsistent replica.
- Anti-entropy repair - compares all nodes and writes most recent data to fix replicas.

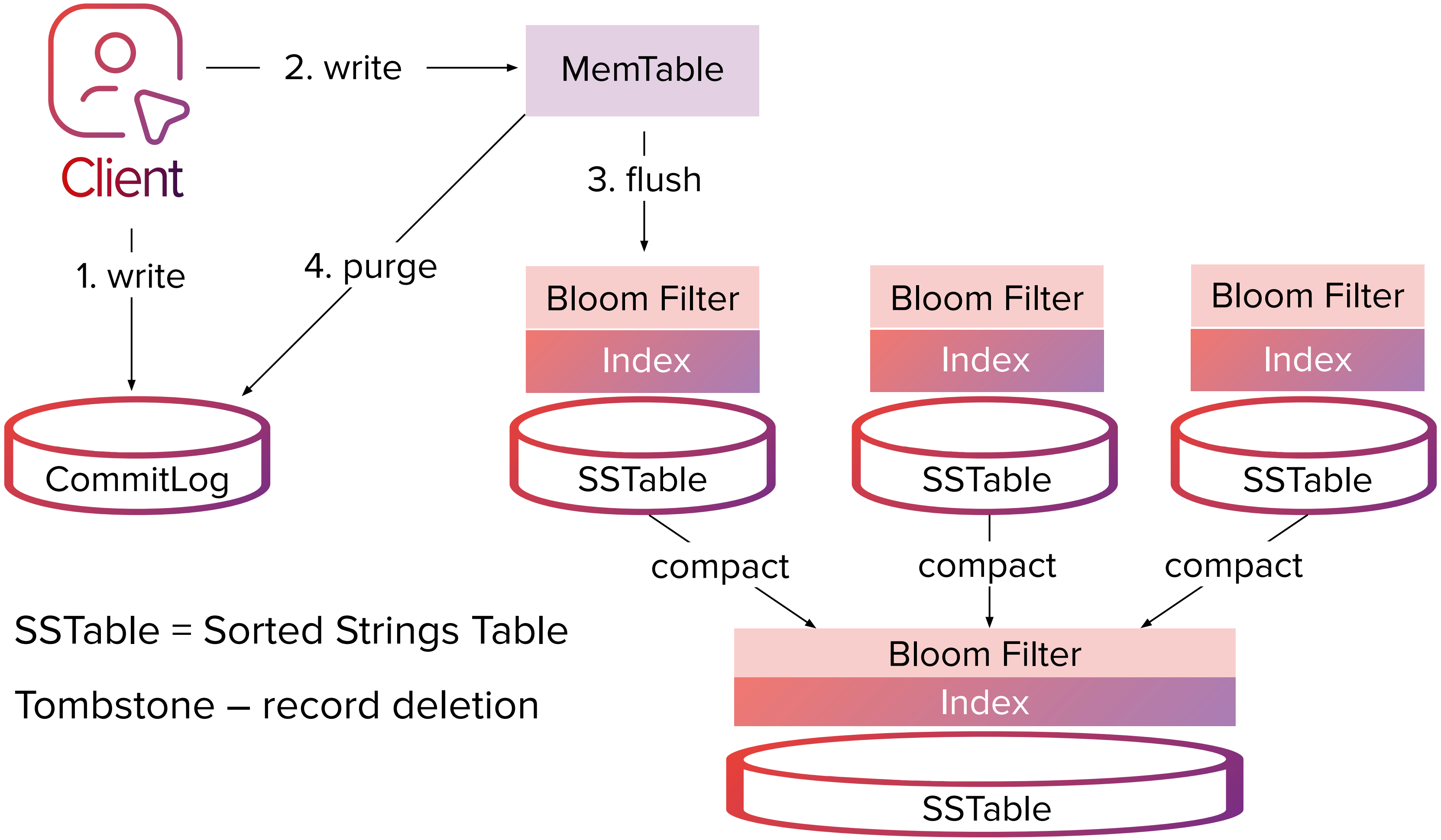
Cassandra Concurrency

- The unit of modification is a single column in a row.
- Multiple clients can update separate columns in a row without a conflict.
- Conflicting writes are resolved using timestamps - “Last Write Wins”.

- Support for “lightweight”, “optimistic” transactions limited to a single operation on a row.
- Compare-and-set - operation checks the value and if the value is as expected, updates the value, otherwise operation needs to be retried.
- Transaction implemented by a quorum-based transaction protocol - Paxos:

https://martinfowler.com/articles/patterns-of-distributed-systems/paxos.html

Cassandra Log-Structured Merge Tree



SSTable = Sorted Strings Table

Tombstone – record deletion

Cassandra Usage Considerations

Reasons to use

- Applicable for most data scenarios.
- Huge datasets, accessed by “almost” SQL (no joins) - CQL.
- Easy horizontal scaling, cross-DC replication.
- Leaderless architecture - increased availability.

Reasons not to use:

- Disk space consumption - it is difficult to tune the SSTable compaction properly in data intensive scenarios.
- Works on JVM - garbage collections, etc. may affect performance (consider using ScyllaDB).
- Relatively complex - bugs?

Aerospike

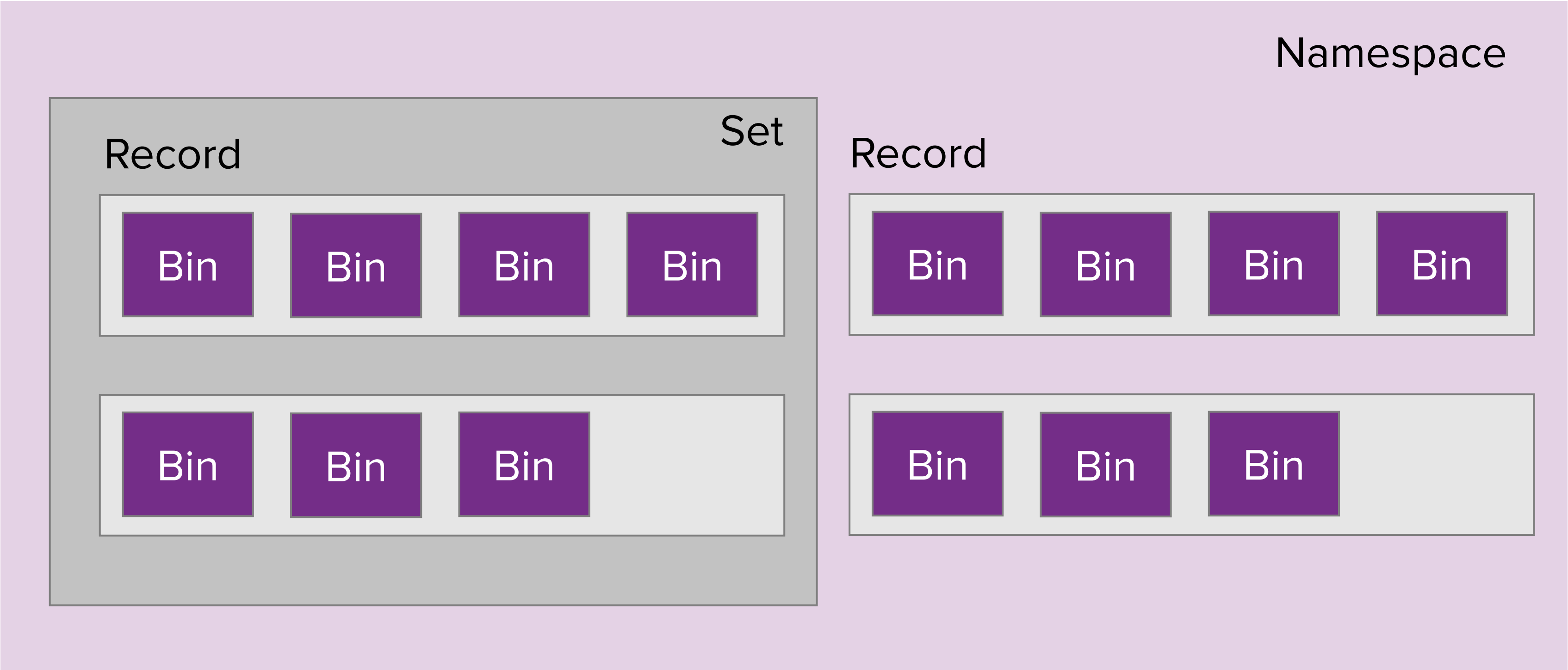
- Very fast data access by key.
- Hybrid storage - RAM + block devices + PMEM (Persistent Memory).
- Can store data on raw SSD/NVMe block devices - bypassing usual filesystem layer.
- In-memory indexes preserved on a shared memory segment (for fast recovery).
- Relatively easy single master per partition replication scheme.
- Client-tunable consistency policies
- Transactions are limited to a single record and are CAS based.
- No MVCC (Multi Version Concurrency Control)
- Strives for availability.

Aerospike Hybrid Storage

		Data Storage		
		NVMe Flash	DRAM	PMEM
Primary index storage	NVMe Flash	**ALL NVMe Flash**: Ultra large records sets	Not recommended.	Not recommended.
	DRAM	**Hybrid**: best price – performance.	High performance. No persistence.	Not common.
	PMEM	**Hybrid**: Fast restart after reboot. Very large data sets.	Not recommended.	**All PMEM**: Fast restart after reboot, with high performance.

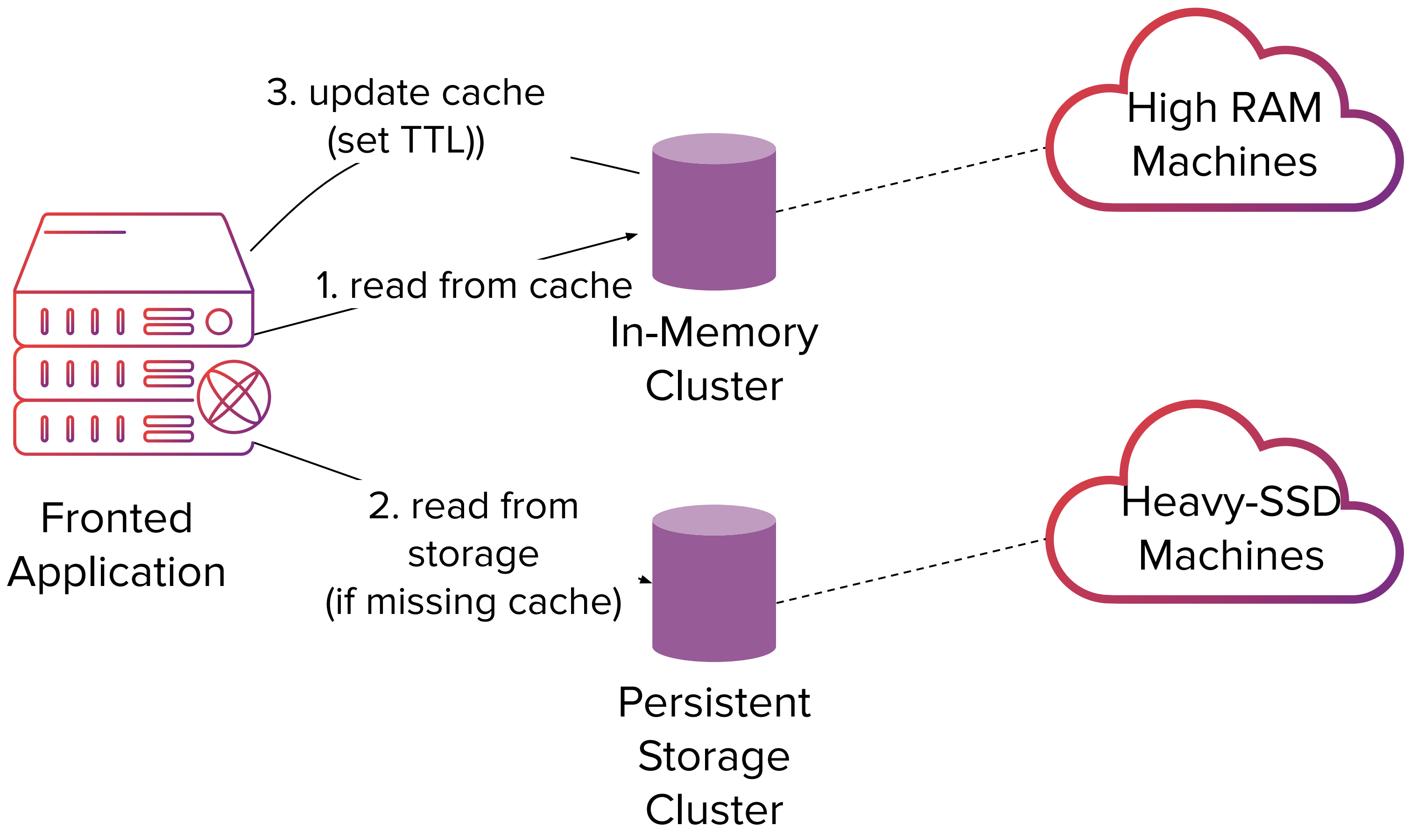
Aerospike Hybrid Storage

- Data is always distributed into 4096 partitions, evenly spread across nodes.
- Data model is straightforward:



RTB HOUSE =

Aerospike at RTB House



Aerospike Usage Considerations

Reasons to use

- Low latency access to data.
- High concurrency writes support.
- Easy cluster management.

Reasons not to use:

- Community version is severely limited (number of nodes, amount of data).
- Frequent scans are heavy and involve all nodes, due to the hash-based data distribution model.

RTB HOUSE

What to store in a (key-value) NoSQL Database?

- Unstructured data (images, text, binary files).
- Structured data in text document formats:
  - JSON
  - XML
- Structured data in binary formats:
  - BSON - Binary JSON
  - ProtocolBuffers
  - Apache Avro
- What we are aiming for is the forward/backward compatibility between schema versions.
- We want to support schema evolution.

Avro vs Protocol Buffers

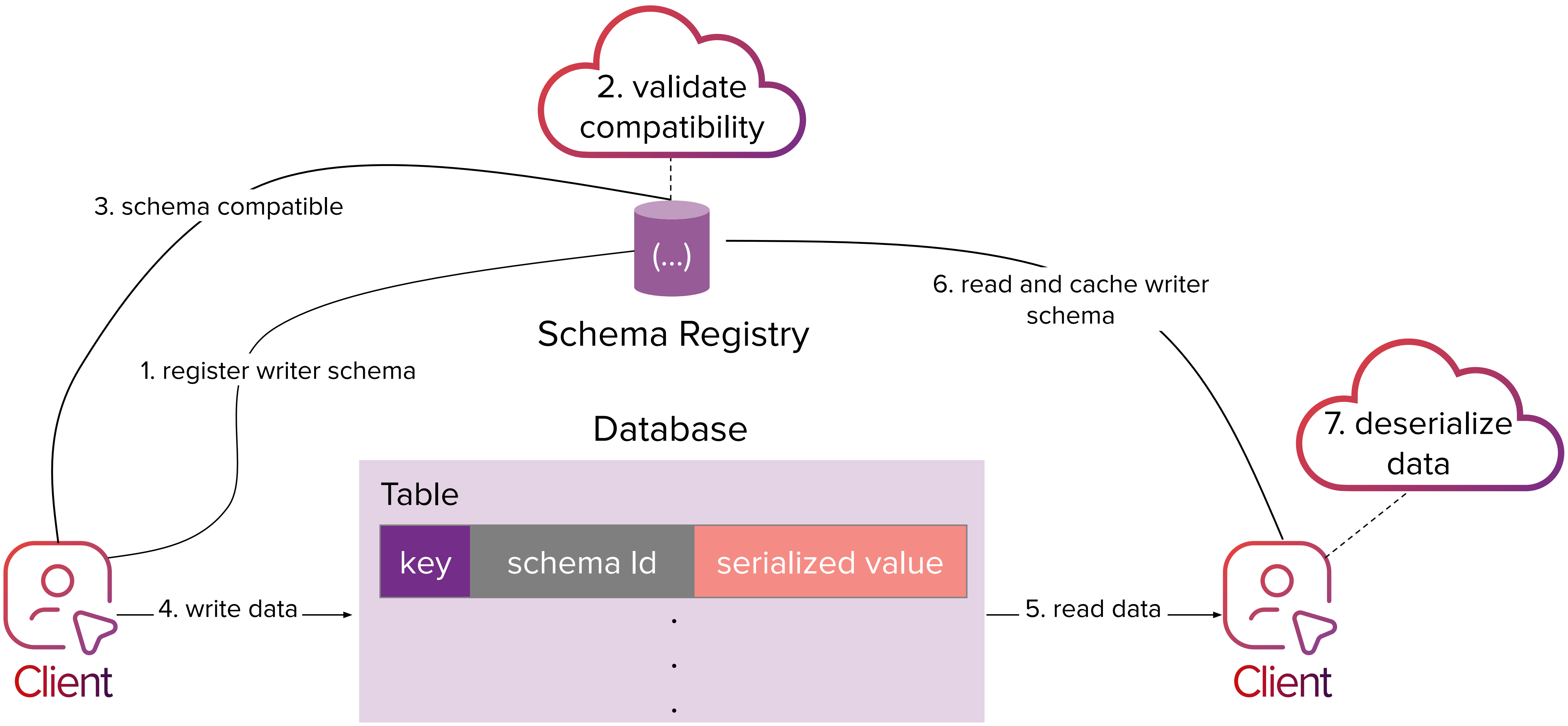
Protocol Buffers:

- Support for schema evolution via field tags (order numbers).
- Field tags cannot change, and possible change of types must be compatible.
- Field tags must be written to a serialized data.
- Prevalent in various Google ecosystem tools.

Avro:

- Must know the writer schema to support the schema evolution.
- More concise binary format (no field tags).
- Wider support in various Apache Big Data tools.

Schema Registry Pattern



Summary

We have discussed:

1.

The available data models for NoSQL databases

2.

The implementation details of MongoDB, Apache HBase, Apache Cassandra and Aerospike databases.

3.

The data formats, schema evolution and the schema registry pattern.

RTB HOUSE

Thank you.

Piotr Jaczewski