Patrick White, Gianfranco Scipioni
Michael Greer, Violetta Vylegzhanina
Shashank Shekhar, Nathan Walker

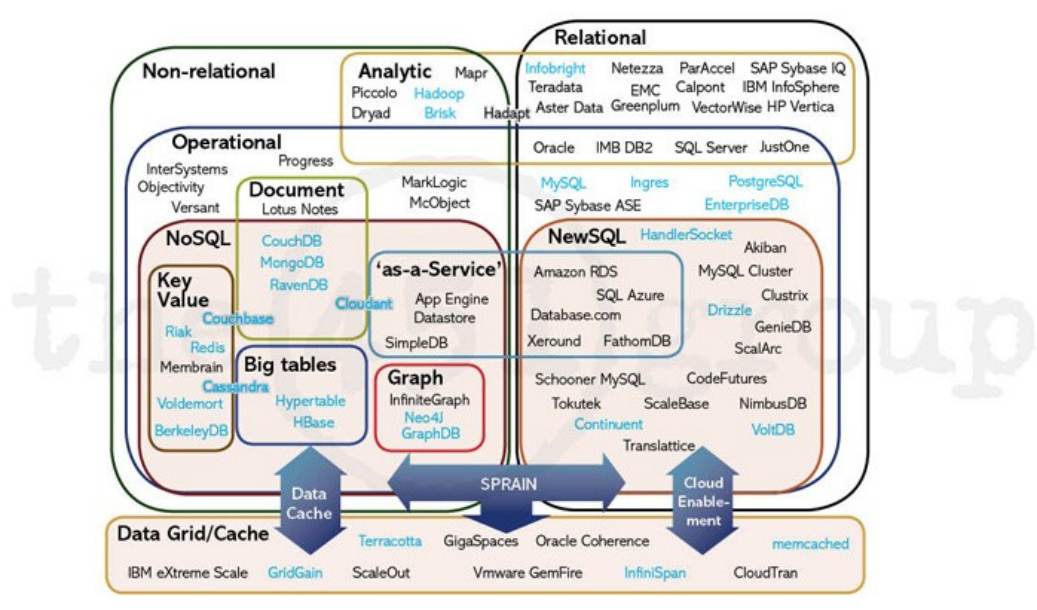
Outline

- Introduction
- Background
- Use Cases
- Data Model & Query Language
- Architecture
- Conclusion

Cassandra Background

What is Cassandra?

- Open-source database management system (DBMS)
- Several key features of Cassandra differentiate it from other similar systems



The ideal database foundation for today's modern applications

"Apache Cassandra is renowned in the industry as the only NoSQL solution that can accommodate the complex requirement of today's modern line-of-business applications. It's architected from the ground up for real-time enterprise databases that require vast scalability, high-velocity performance, flexible schema design and continuous availability."

-DataStax

History of Cassandra

- Cassandra was created to power Facebook Inbox Search
- Facebook open-sourced Cassandra in 2008 and became an Apache Incubator project
- In 2010, Cassandra graduated to a top-level project, regular update and releases followed

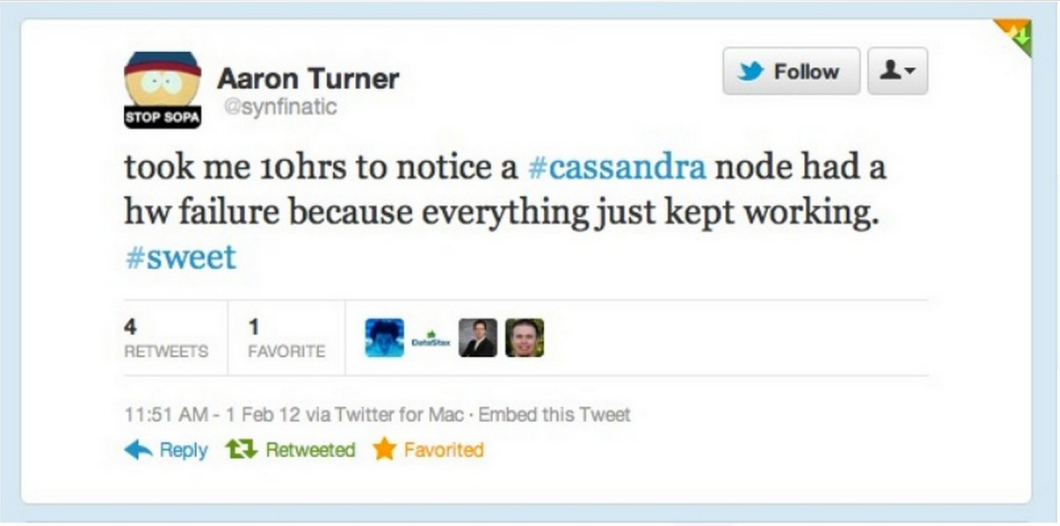
Motivation and Function

- Designed to **handle large amount of data** across multiple servers
- There is a **lot** of **unorganized data** out there
- Easy to **implement** and **deploy**
- Mimics traditional relational database systems, but with **triggers** and **lightweight transactions**
- Raw, simple data structures

Availability

"There is no such thing as standby infrastructure: there is stuff you always use and stuff that won't work when you need it."

-Ben Black, Founder, Boundary



General Design Features

Emphasis on **performance** over analysis

- Still has support for analysis tools such as Hadoop

Organization

- Rows are organized into tables
- First component of a table's primary key is the partition key
- Rows are clustered by the remaining columns of the key
- Columns may be indexed separately from the primary key
- Tables may be created, dropped, altered at runtime without blocking queries

Language

- CQL (Cassandra Query Language) introduced, similar to SQL (flattened learning curve)

Peer to Peer Cluster

- Decentralized design
  - Each node has the same role
- No single point of failure
  - Avoids issues of master-slave DBMS's
- No bottlenecking

Fault Tolerance/Durability

Failures happen all the time with multiple nodes

- Hardware Failure
- Bugs
- Operator error
- Power Outage, etc.

Solution: Buy cheap, redundant hardware, replicate, maintain consistency

Fault Tolerance/Durability

- Replication
  - Data is automatically replicated to multiple nodes
  - Allows failed nodes to be immediately replaced
- Distribution of data to multiple data centers
  - An entire data center can go down without data loss occurring

Performance

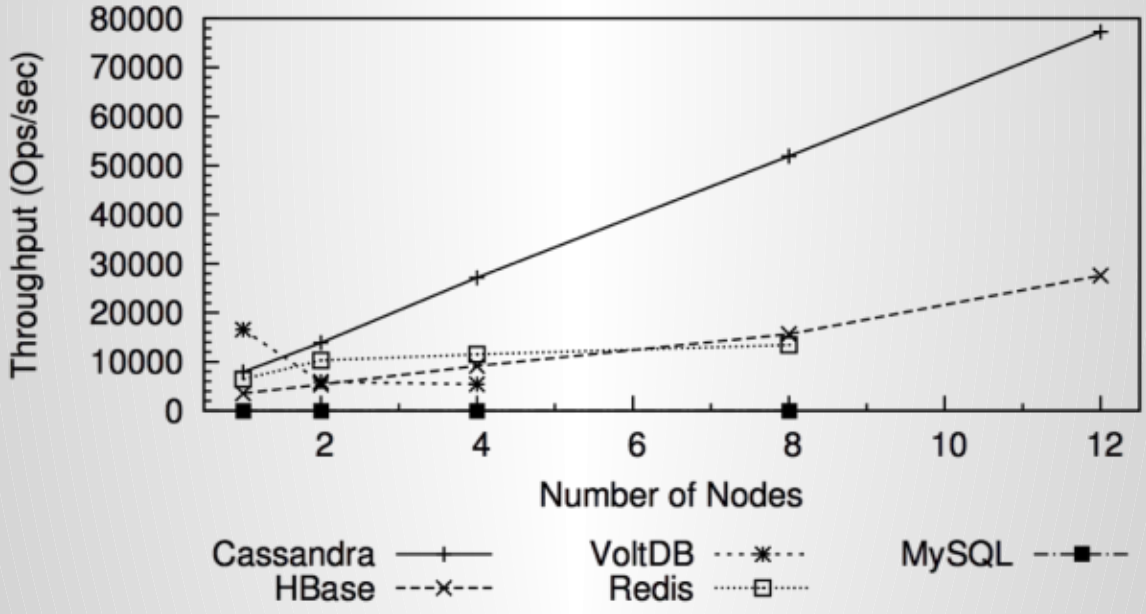
- Core architectural designs allow Cassandra to outperform its competitors
- Very good read and write throughputs
  - Consistently ranks as fastest amongst comparable NoSql DBMS's with large data sets

"In terms of scalability, there is a clear winner throughout our experiments. Cassandra achieves the highest throughput for the maximum number of nodes…" - University of Toronto

Scalability

Read and write throughput increase linearly as more machines are added

> *"In terms of scalability, there is a clear winner throughout our experiments. Cassandra achieves the highest throughput for the maximum number of nodes…"* - University of Toronto



Comparisons

	Apache Cassandra	Google Big Table	Amazon DynamoDB
Storage Type	Column	Column	Key-Value
Best Use	Write often, read less	Designed for large scalability	Large database solution
Concurrency Control	MVCC	Locks	ACID
Characteristics	High Availability Partition Tolerance Persistence	Consistency High Availability Partition Tolerance Persistence	Consistency High Availability

Key Point – Cassandra offers a healthy cross between BigTable and Dynamo.

Cassandra Use Cases

Netflix

- online DVD and Blu-Ray movie retailer

- Nielsen study showed that 38% of Americans use or subscribe to Netflix

Netflix: Why Cassandra

- Using a central SQL database negatively impacted scalability and availability
- International Expansion required Multi-Datacenter solution
- Need for configurable Replication, Consistency, and Resiliency in the face of failure
- Cassandra on AWS offered high levels of scalability and availability

Jason Brown, Senior Software Engineer at Netflix

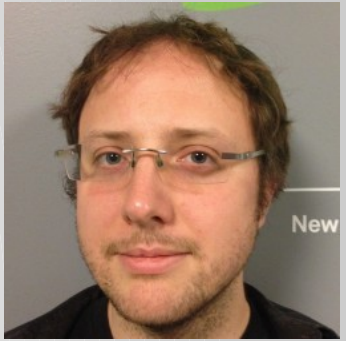
Spotify

- Streaming digital music service

- Music for every moment on computer, phone, tablet, and more

Spotify: Why Cassandra

- With more users, scalability problems arised using postgreSQL
- With multiple data centers, streaming replication in postgreSQL was problematic
  - ex: high write volumes
- Chose Cassandra
  - lack of single points of failure
  - no data loss confidence
  - Big Table design



Axel Liljencrantz, Backend Developer at Spotify

Spotify: Why Cassandra

- Cassandra behaves better in specific use cases:
  - better replication (especially writes)
  - better behavior in the presence of networking issues and failures, such as partitions or intermittent glitches
  - better behavior in certain classes of failure, such as server dies and network links going down

Hulu

- a website and a subscription service offering on-demand streaming video media

- ~30 million unique viewers per month

Hulu: Why Cassandra

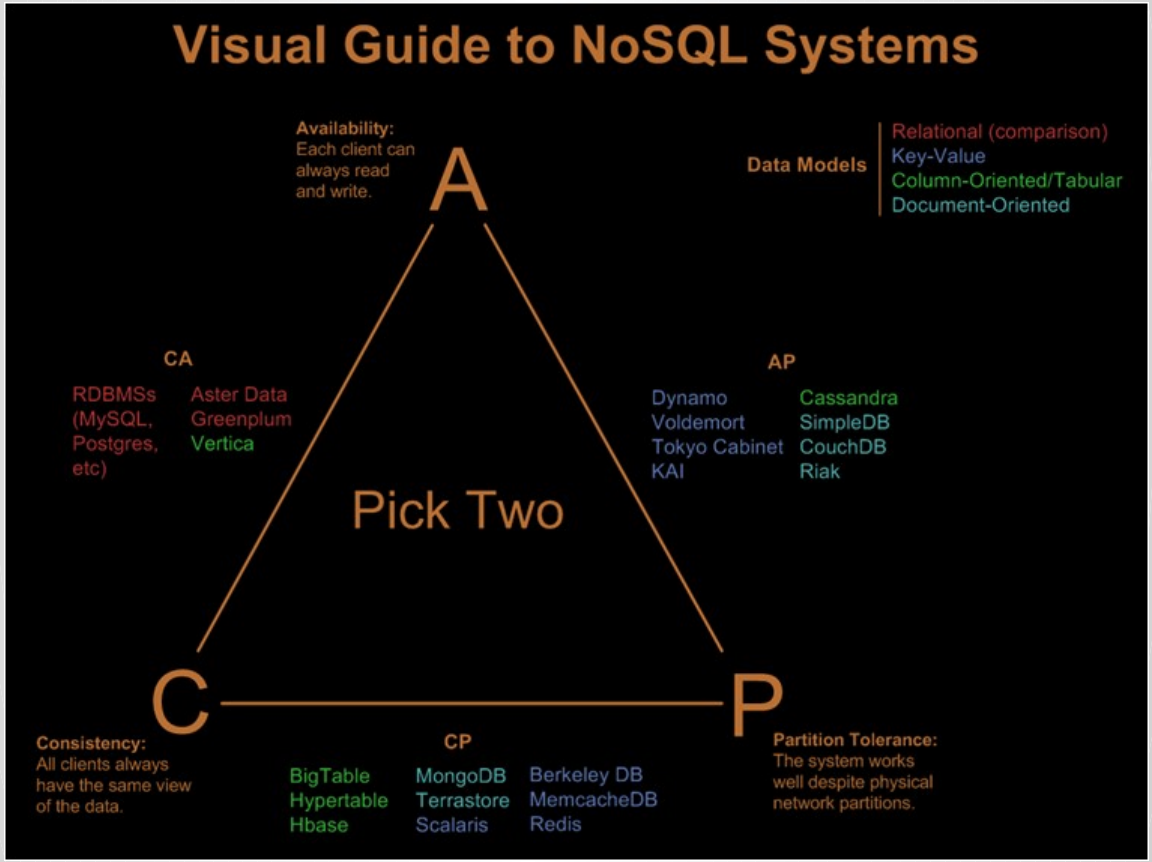
- need for Availability
- need for Scalability
- Good Performance
- Nearly Linear Scalability
- Geo-Replication
- Minimal Maintenance Requirements

Andres Rangel,
Senior Software
Engineer at Hulu

Reasons for Choosing Cassandra

- Value availability over consistency
- Require high write-throughput
- High scalability required
- No single point of failure

CAP Theorem


Visual Guide to NoSQL Systems
Availability:
Each client can always read and write.
A
Data Models
Relational (comparison)
Key-Value
Column-Oriented/Tabular
Document-Oriented
CA
RDBMSs (MySQL, Postgres, etc)
Aster Data
Greenplum
Vertica
AP
Dynamo
Voldemort
Tokyo Cabinet
KAI
Cassandra
SimpleDB
CouchDB
Riak
Pick Two
C
P
Consistency:
All clients always have the same view of the data.
CP
BigTable
Hypertable
Hbase
MongoDB
Terrastore
Scalaris
Berkeley DB
MemcacheDB
Redis
Partition Tolerance:
The system works well despite physical network partitions.

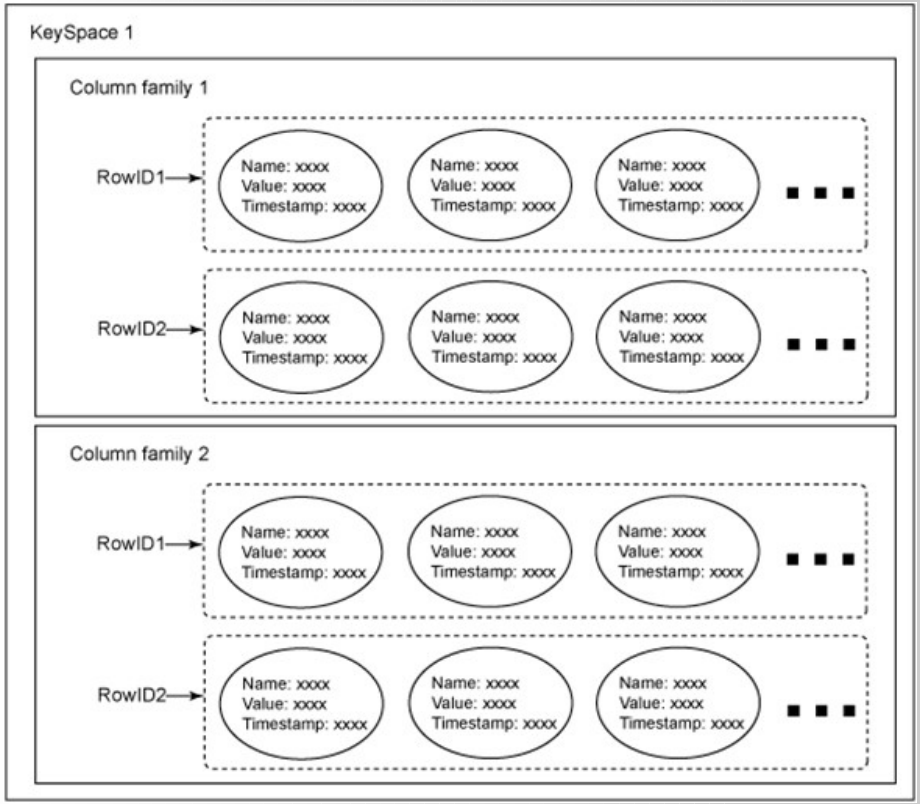
Eventual Consistency

- **BASE** – **B**asically **A**vailable **S**oft-state **E**ventual consistency (versus **ACID**)
- *If all writers stop (to a key), then all its values (replicas) will converge eventually.*
- If writes continue, then system always tries to keep converging.
  - Moving “wave” of updated values lagging behind the latest values sent by clients, but always trying to catch up
- Converges when R + W > N
  - R = # records to read, W = # records to write, N = replication factor
- Consistency Levels:
  - ONE -> R or W is 1
  - QUORUM -> R or W is ceiling (N + 1) / 2
  - ALL -> R or W is N
- If you want to write with Consistency Level of ONE and get the same data when you read, you need to read with Consistency Level of ALL

Cassandra’s Data Model

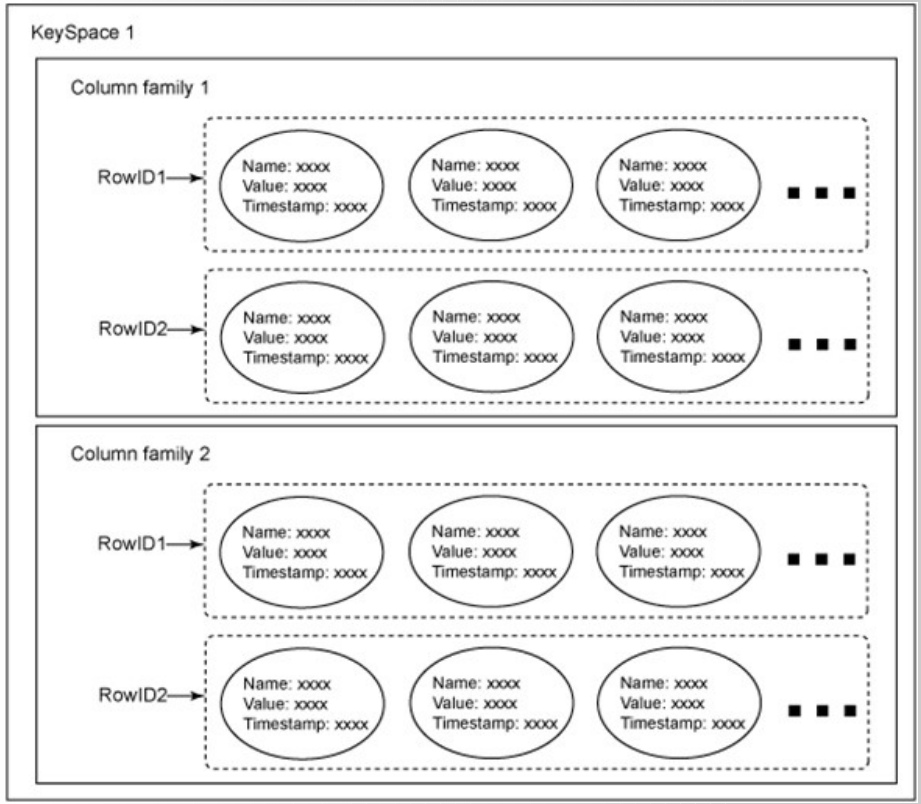
Key-Value Model

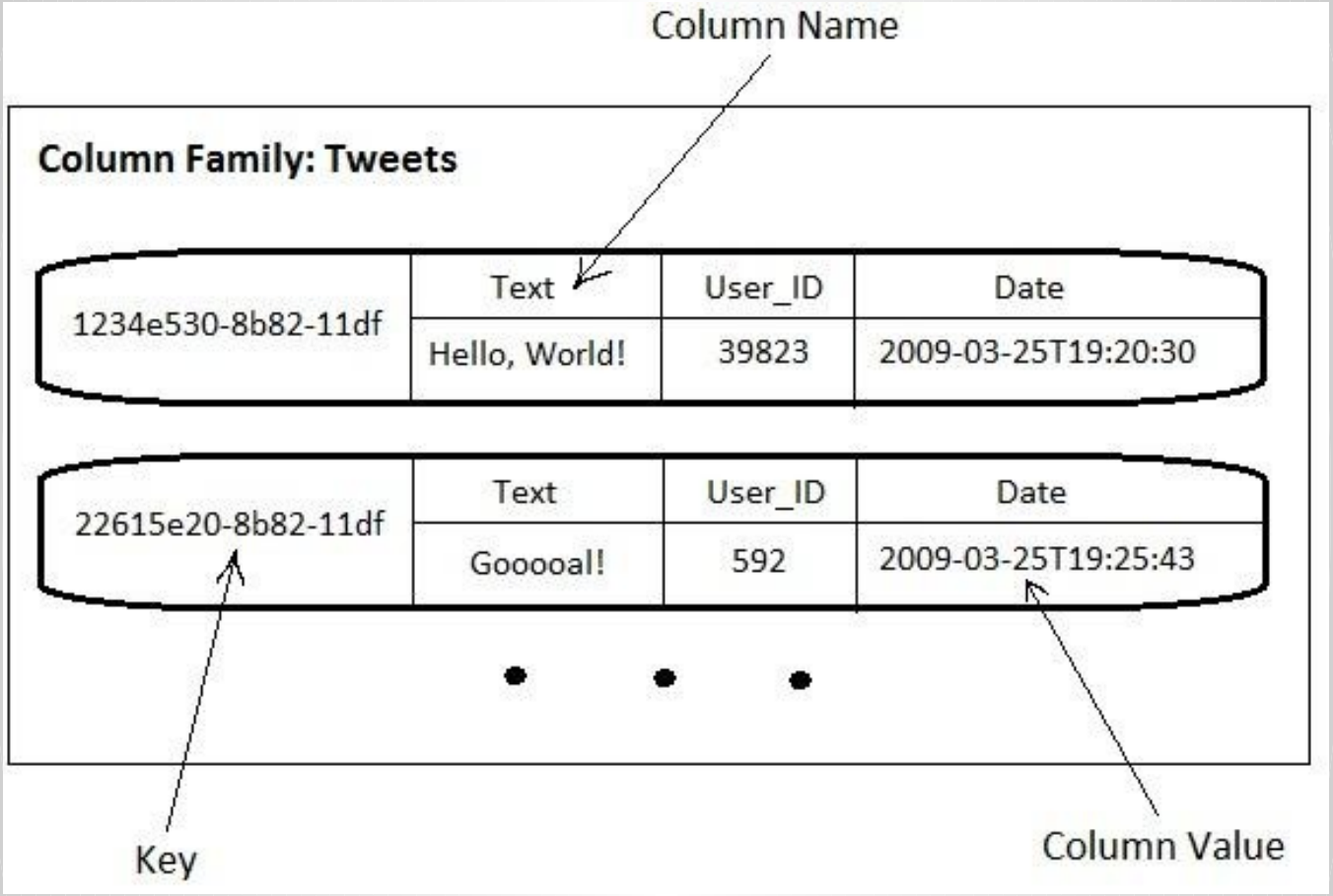
- Cassandra is a column oriented NoSQL system
- Column families: sets of key-value pairs
  - column family as a table and key-value pairs as a row (using relational database analogy)
- A row is a collection of columns labeled with a name



Cassandra Row

- the value of a row is itself a sequence of key-value pairs
- such nested key-value pairs are *columns*
- key = column name
- a row must contain at least 1 column

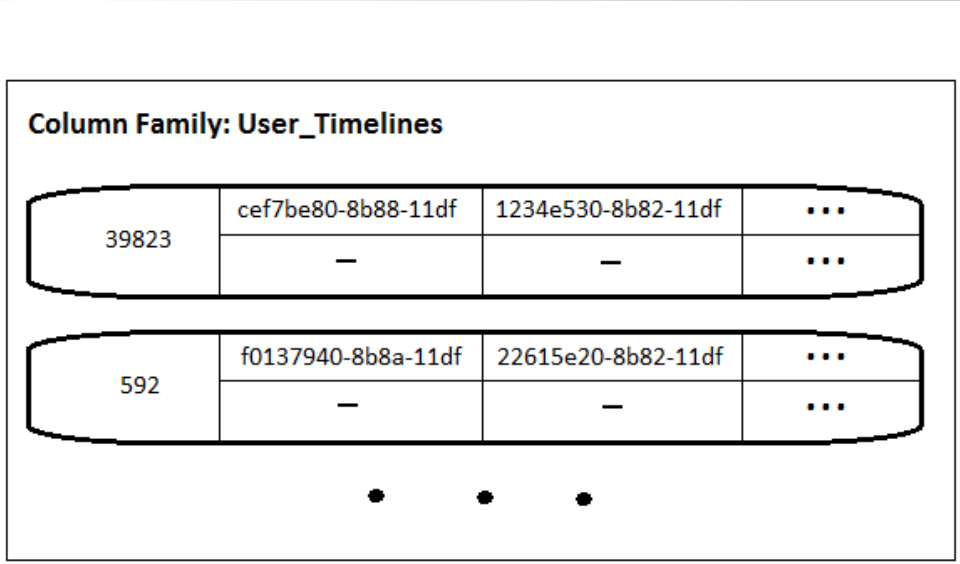




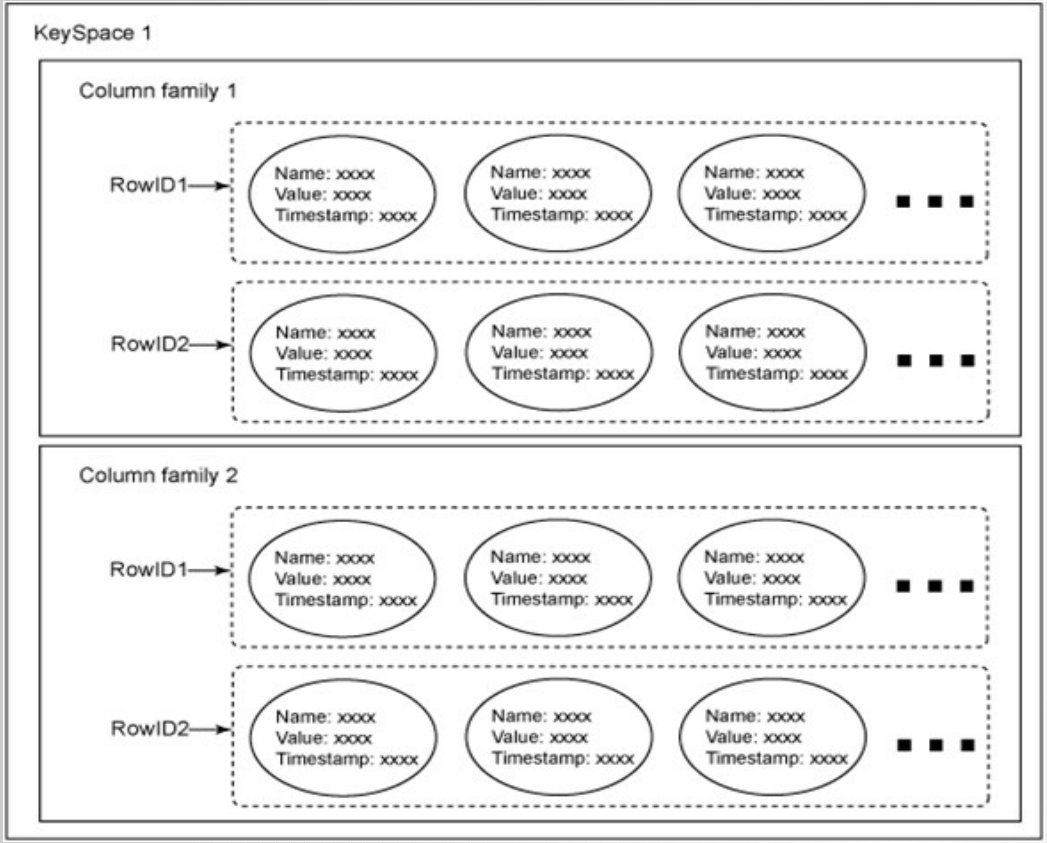
Example of Columns

Column names storing values

- key: User ID
- column names store tweet ID values
- values of all column names are set to “-” (empty byte array) as they are not used



Key Space



- A Key Space is a group of column families together. It is only a logical grouping of column families and provides an isolated scope for names

Comparing Cassandra (C*) and RDBMS

- with RDBMS, a normalized data model is created without considering the exact queries
  - SQL can return almost anything though Joins
- with C*, the data model is designed for specific queries
  - schema is adjusted as new queries introduced
- C*: NO joins, relationships, or foreign keys
  - a separate table is leveraged per query
  - data required by multiple tables is denormalized across those tables

Cassandra Query Language - CQL

- creating a *keyspace* - namespace of tables

```
CREATE KEYSPACE demo
WITH replication = {‘class’: ’SimpleStrategy’,
 replication_factor’: 3};
```

- to use namespace:

```
USE demo;
```

Cassandra Query Language - CQL

- creating tables:

```
CREATE TABLE users(
email varchar,
bio varchar,
birthday timestamp,
active boolean,
PRIMARY KEY (email));
```

```
CREATE TABLE tweets(
email varchar,
time_posted timestamp,
tweet varchar,
PRIMARY KEY (email, time_posted));
```

Cassandra Query Language - CQL

- inserting data

```
INSERT INTO users (email, bio, birthday, active)
VALUES ('john.doe@bti360.com', 'BT360 Teammate',
    516513600000, true);
```

  - ** timestamp fields are specified in *milliseconds since epoch*

Cassandra Query Language - CQL

- querying tables
  - SELECT expression reads one or more records from Cassandra column family and returns a result-set of rows

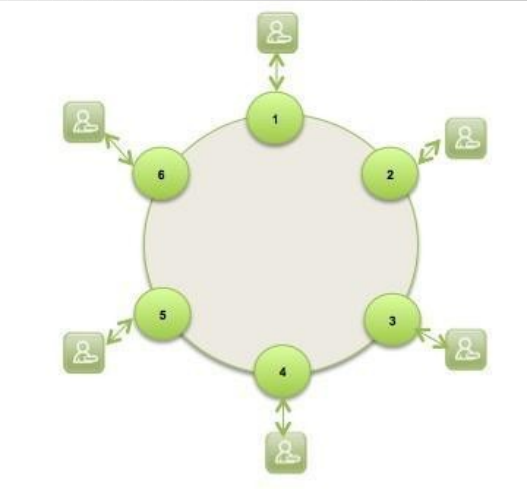
```
SELECT * FROM users;
```

```
SELECT email FROM users WHERE active = true;
```

Cassandra Architecture

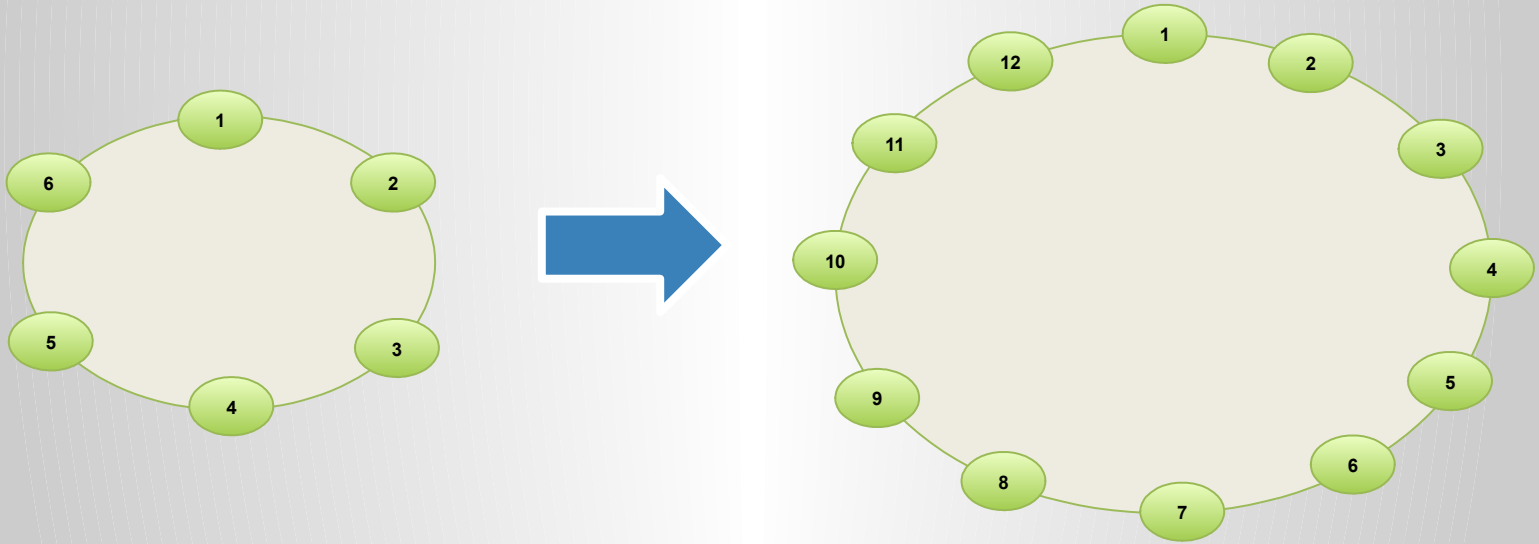
Cassandra Architecture Overview

- Cassandra was designed with the understanding that system/ hardware failures can and do occur
- Peer-to-peer, distributed system
- All nodes are the same
- Data partitioned among all nodes in the cluster
- Custom data replication to ensure fault tolerance
- Read/Write-anywhere design
- Google BigTable - data model
  - Column Families
  - Memtables
  - SSTables
- Amazon Dynamo - distributed systems technologies
  - Consistent hashing
  - Partitioning
  - Replication
  - One-hop routing



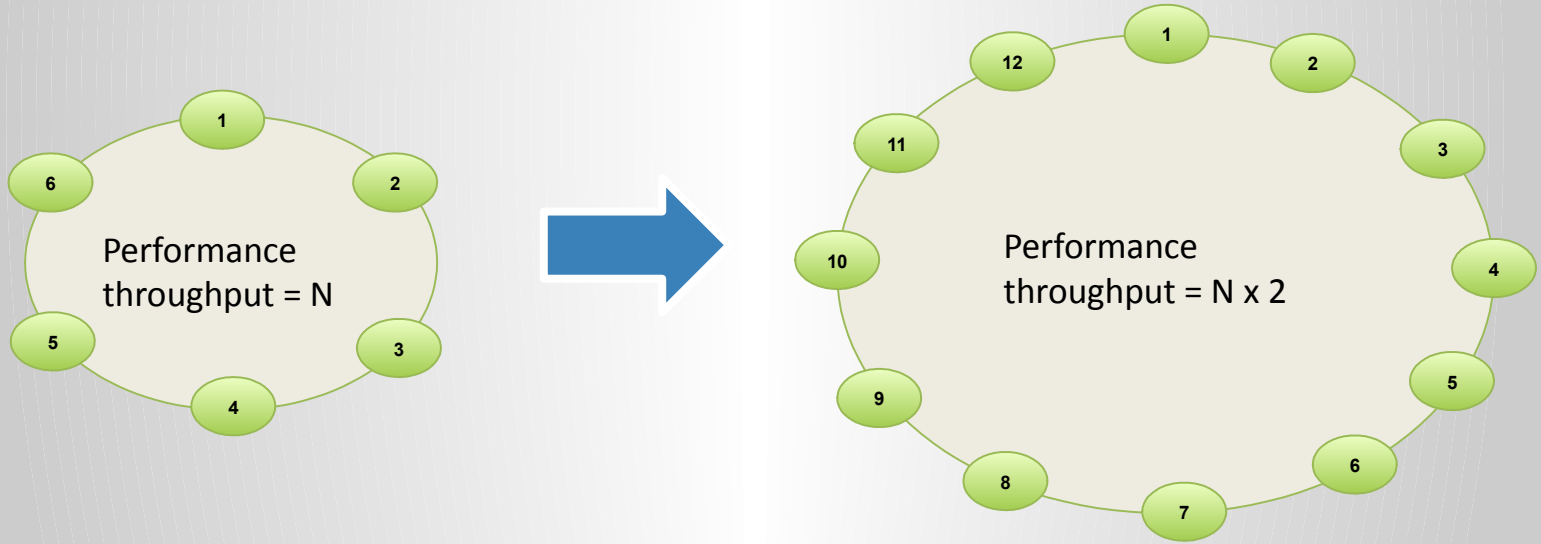
Transparent Elasticity

Nodes can be added and removed from Cassandra online, with no downtime being experienced.



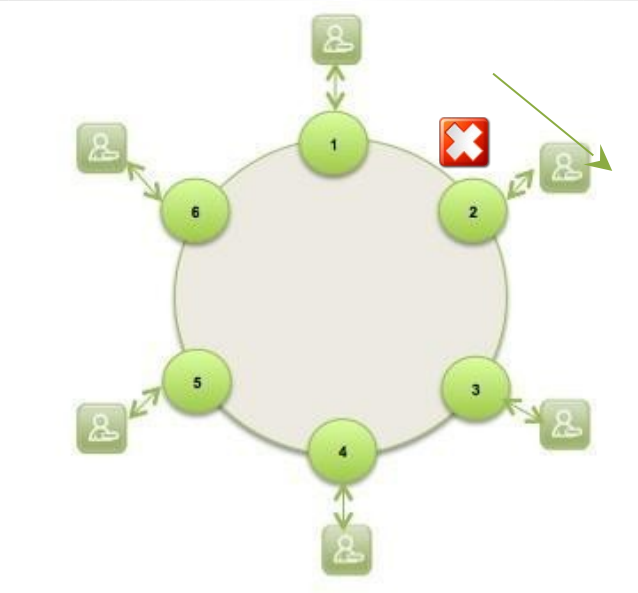
Transparent Scalability

Addition of Cassandra nodes increases performance linearly and ability to manage TB's-PB's of data.



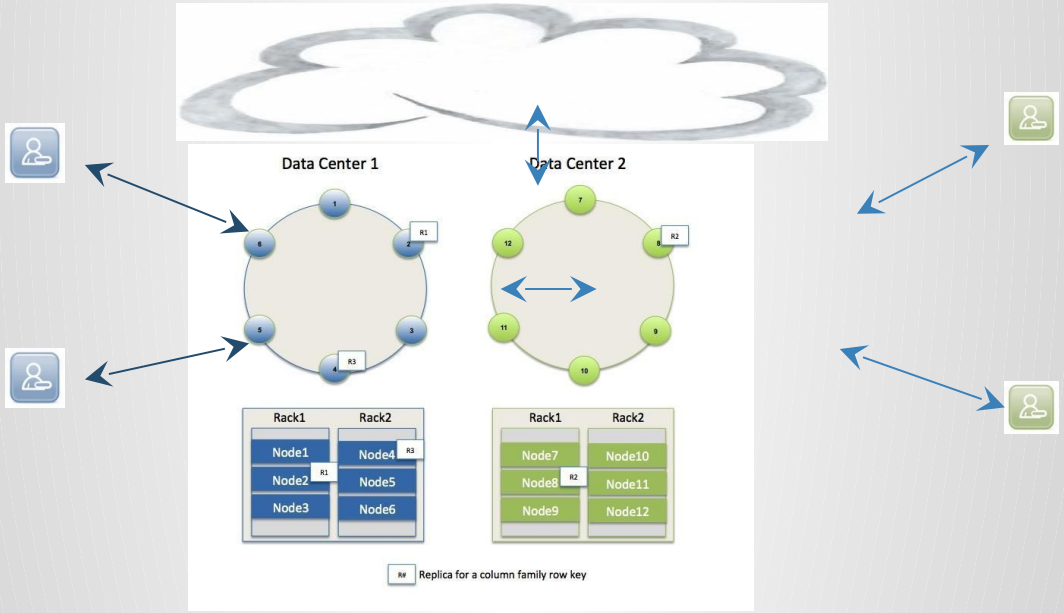
High Availability

Cassandra, with its peer-to-peer architecture has no single point of failure.



Multi-Geography/Zone Aware

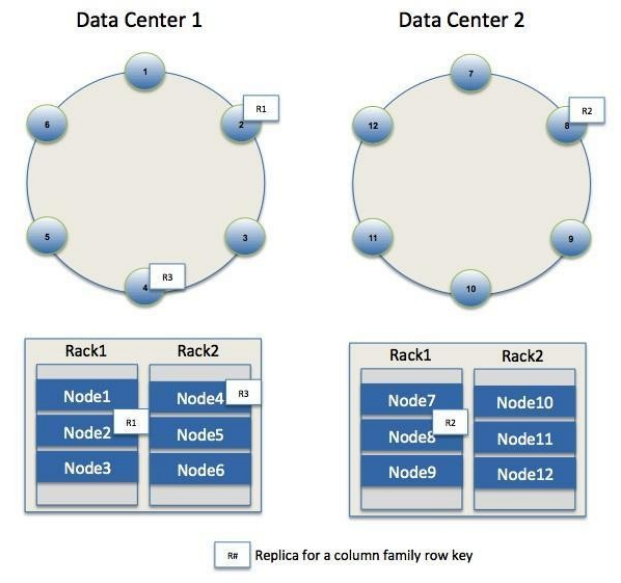
Cassandra allows a single logical database to span 1-N datacenters that are geographically dispersed. Also supports a hybrid on-premise/Cloud implementation.



Data Redundancy

Cassandra allows for customizable data redundancy so that data is completely protected. Also supports rack awareness (data can be replicated between different racks to guard against machine/rack failures).

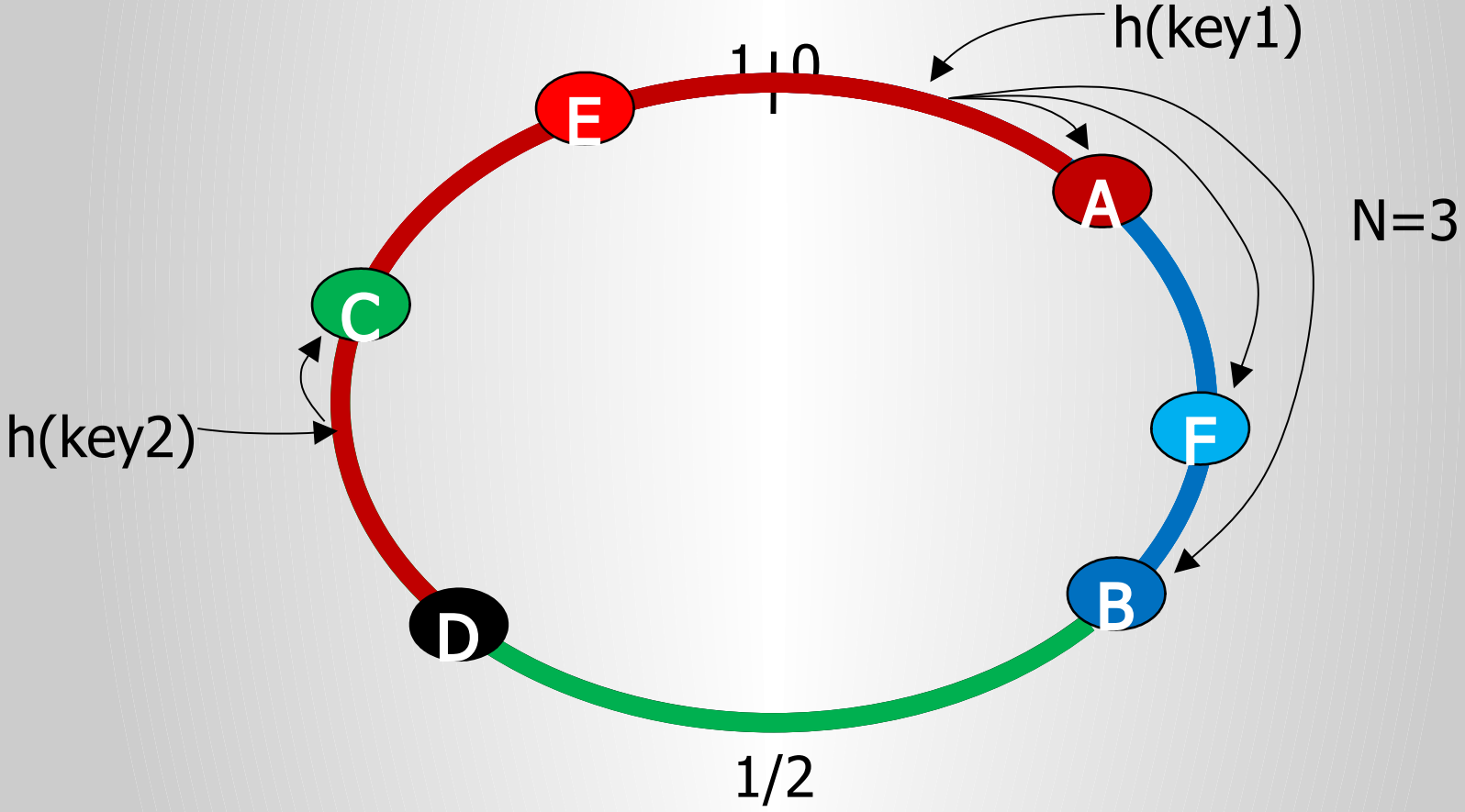
uses ‘Zookeeper’ to choose a leader which tells nodes the range they are replicas for



Partitioning

- Nodes are *logically* structured in Ring Topology.
- Hashed value of key associated with data partition is used to assign it to a node in the ring.
- Hashing rounds off after certain value to support ring structure.
- Lightly loaded nodes moves position to alleviate highly loaded nodes.

Partitioning & Replication



Gossip Protocols

- Used to discover location and state information about the other nodes participating in a Cassandra cluster
- Network Communication protocols inspired for real life rumor spreading.
- Periodic, Pairwise, inter-node communication.
- Low frequency communication ensures low cost.
- Random selection of peers.
- Example – Node A wish to search for pattern in data
  - Round 1 – Node A searches locally and then gossips with node B.
  - Round 2 – Node A,B gossips with C and D.
  - Round 3 – Nodes A,B,C and D gossips with 4 other nodes ……
- Round by round doubling makes protocol very robust.

Failure Detection

- Gossip process tracks heartbeats from other nodes both directly and indirectly
- Node Fail state is given by variable Φ
  - tells how likely a node might fail (suspicion level) instead of simple binary value (up/down).
- This type of system is known as **Accrual Failure Detector**
- Takes into account network conditions, workload, or other conditions that might affect perceived heartbeat rate
- A threshold for Φ tells is used to decide if a node is dead
- If node is correct, phi will be constant set by application.

  Generally $\Phi(t) = 0$

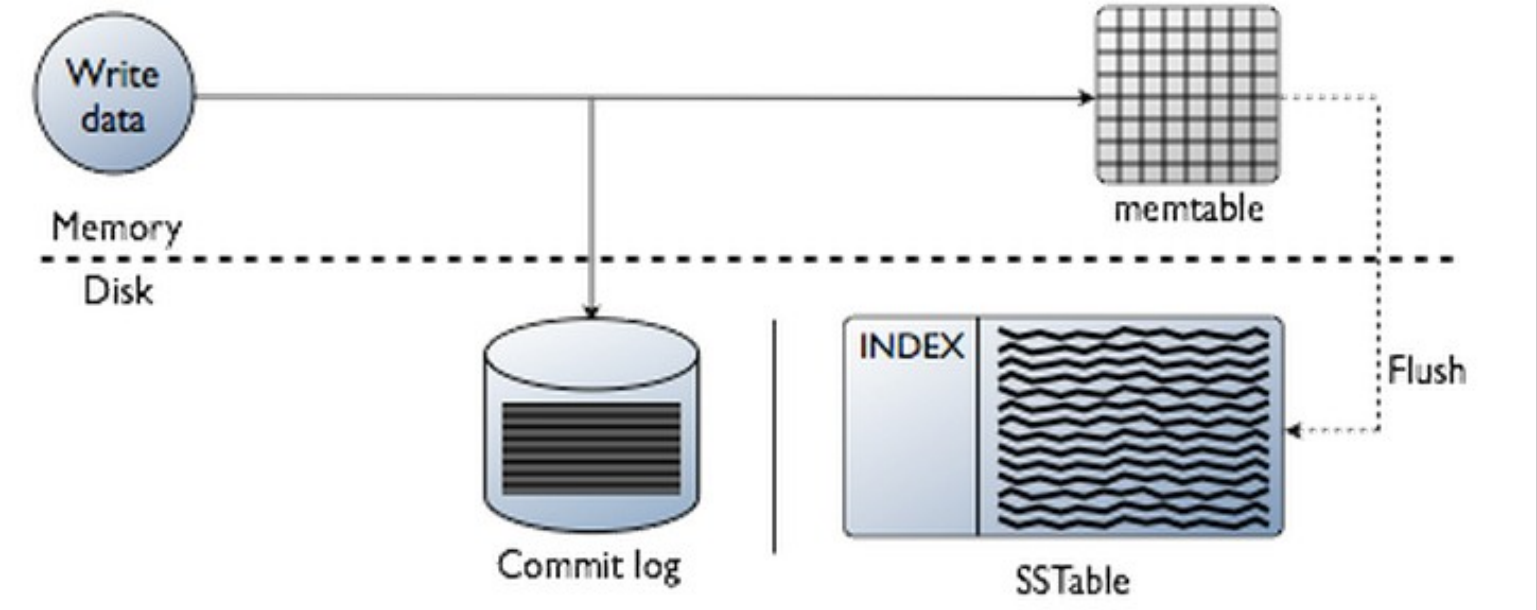
Write Operation Stages

- Logging data in the commit log
- Writing data to the memtable
- Flushing data from the memtable
- Storing data on disk in SSTables

Write Operations

- Commit Log
  - First place a write is recorded
  - Crash recovery mechanism
  - Write not successful until recorded in commit log
  - Once recorded in commit log, data is written to Memtable
- Memtable
  - Data structure in memory
  - Once memtable size reaches a threshold, it is flushed (appended) to SSTable
  - Several may exist at once (1 current, any others waiting to be flushed)
  - First place read operations look for data
- SSTable
  - Kept on disk
  - Immutable once written
  - Periodically compacted for performance

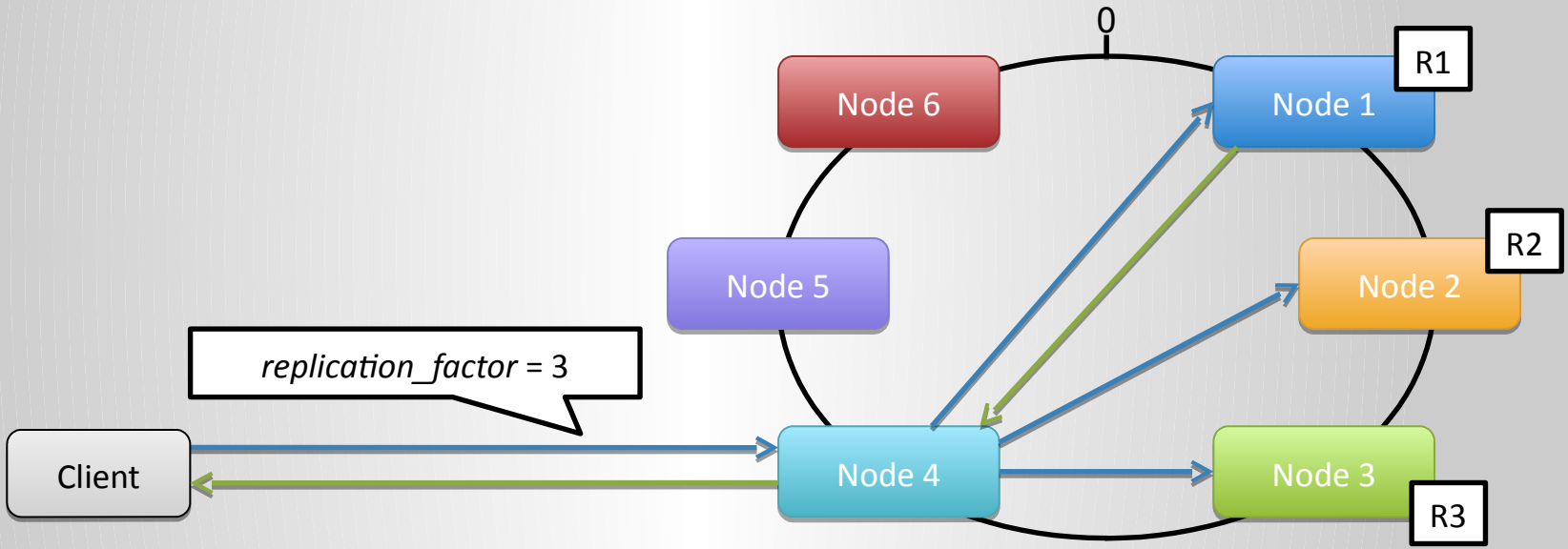
Write Operations



Consistency

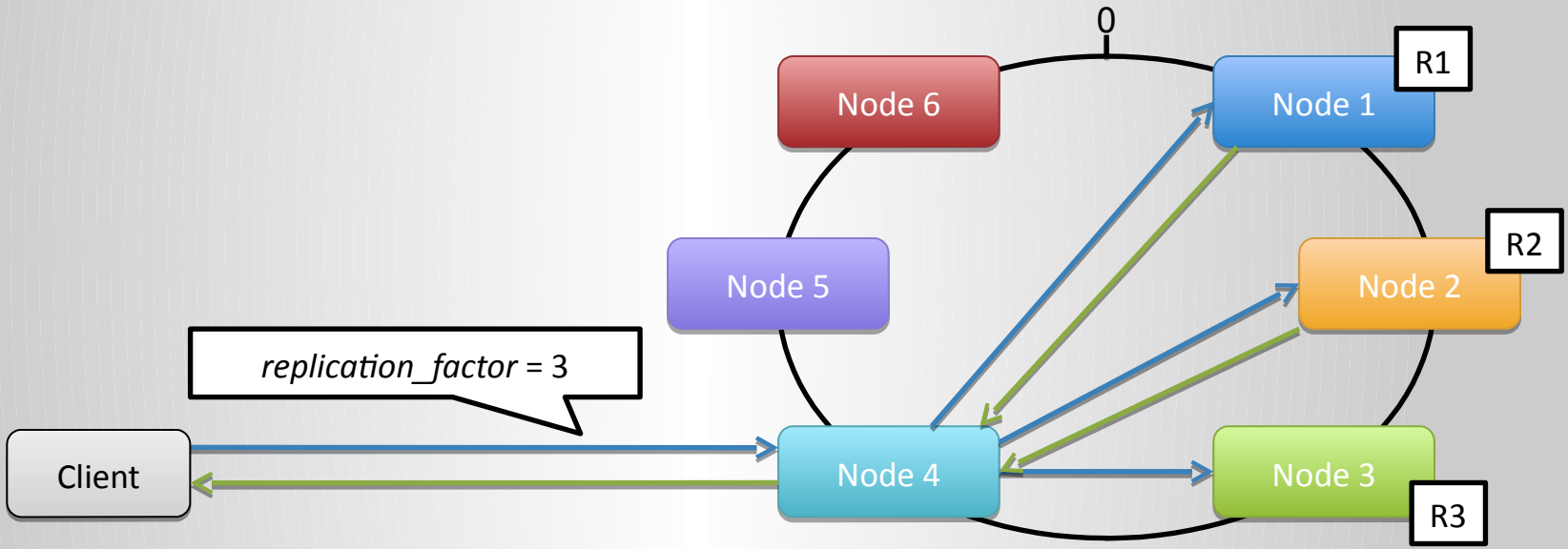
- Read Consistency
  - Number of nodes that must agree before read request returns
  - ONE to ALL
- Write Consistency
  - Number of nodes that must be updated before a write is considered successful
  - ANY to ALL
  - At ANY, a hinted handoff is all that is needed to return.
- QUORUM
  - Commonly used middle-ground consistency level
  - Defined as (replication_factor / 2) + 1

Write Consistency (ONE)



INSERT INTO table (column1, …) VALUES (value1, …) USING CONSISTENCY ONE

Write Consistency (QUORUM)

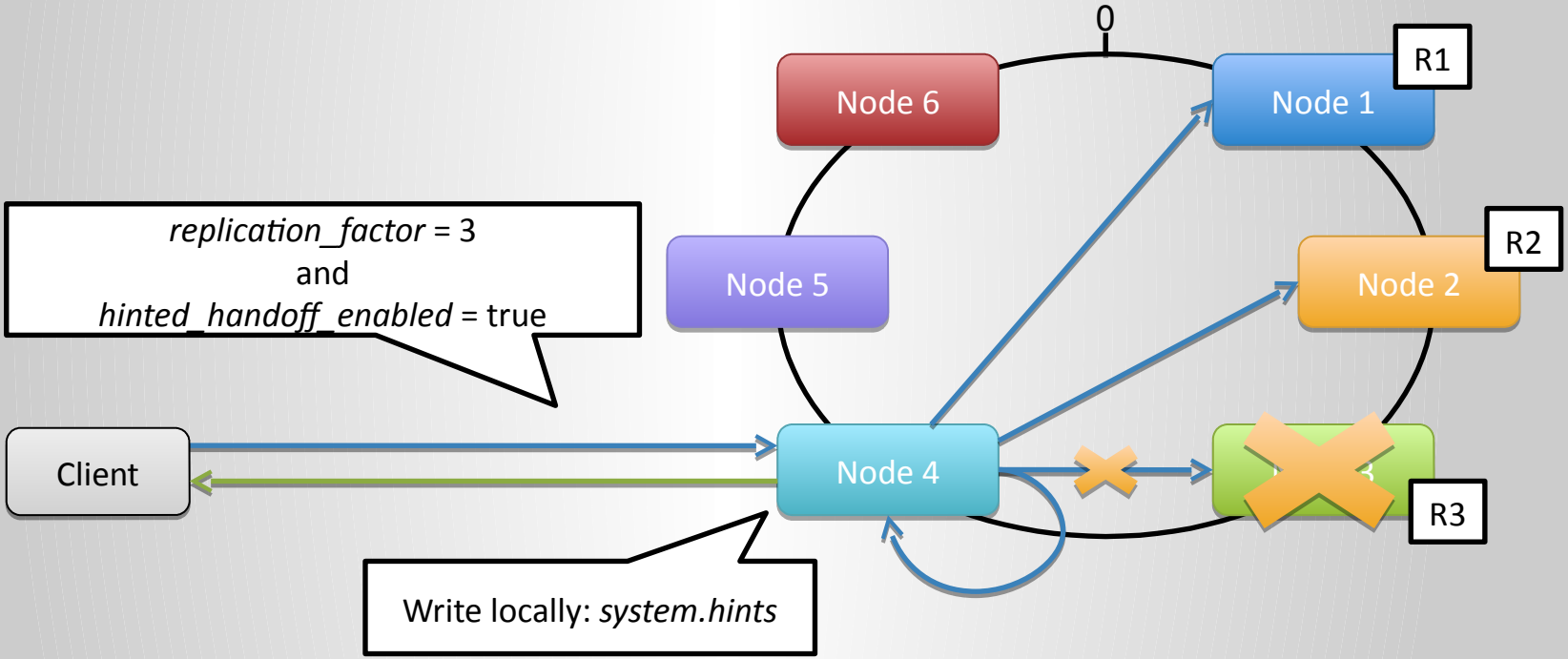


INSERT INTO table (column1, …) VALUES (value1, …) USING CONSISTENCY QUORUM

Write Operations: Hinted Handoff

- Write intended for a node that’s offline
- An online node, processing the request, makes a note to carry out the write once the node comes back online.

Hinted Handoff



INSERT INTO table (column1, …) VALUES (value1, …) USING CONSISTENCY ANY

Note: Doesn't not count toward consistency level (except ANY)

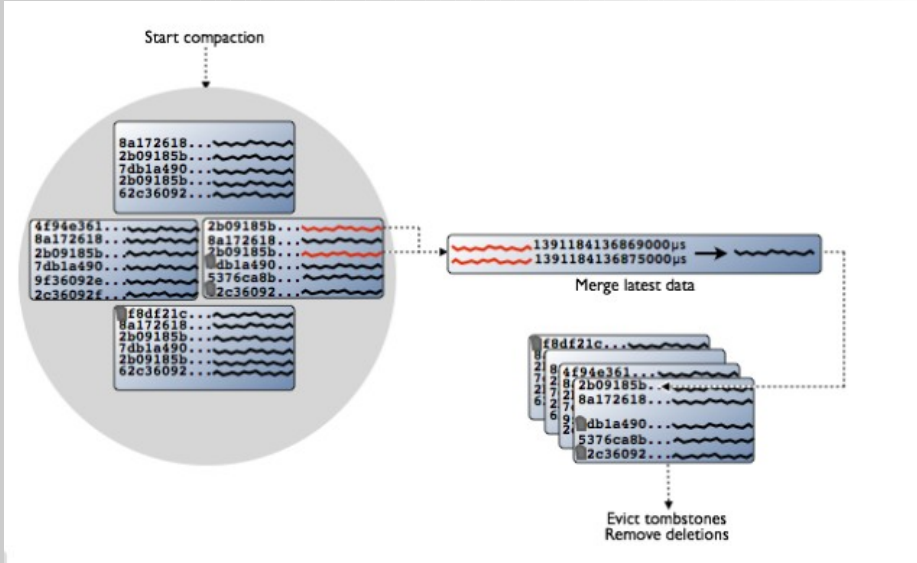
Delete Operations

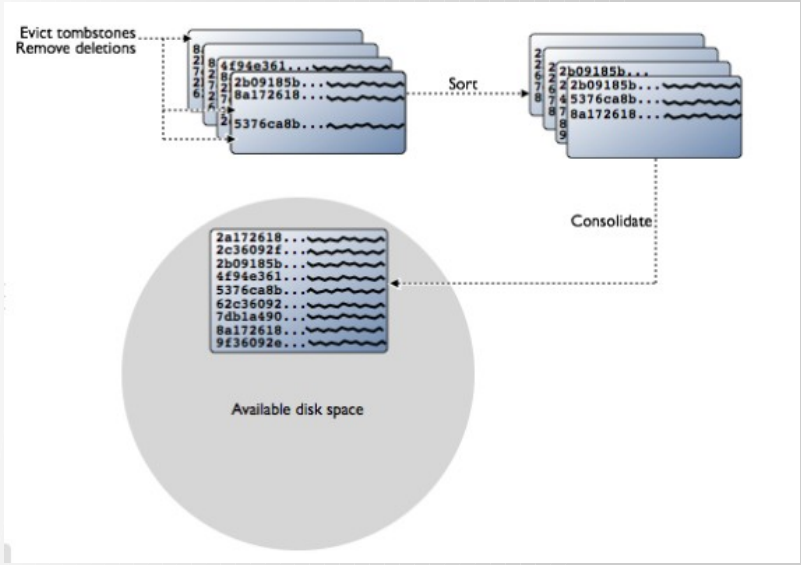
- Tombstones
  - On delete request, records are marked for deletion.
  - Similar to “Recycle Bin.”
  - Data is actually deleted on major compaction or configurable timer

Compaction

- Compaction runs periodically to merge multiple SSTables
  - Reclaims space
  - Creates new index
  - Merges keys
  - Combines columns
  - Discards tombstones
  - Improves performance by minimizing disk seeks
- Two types
  - Major
  - Read-only

Compaction

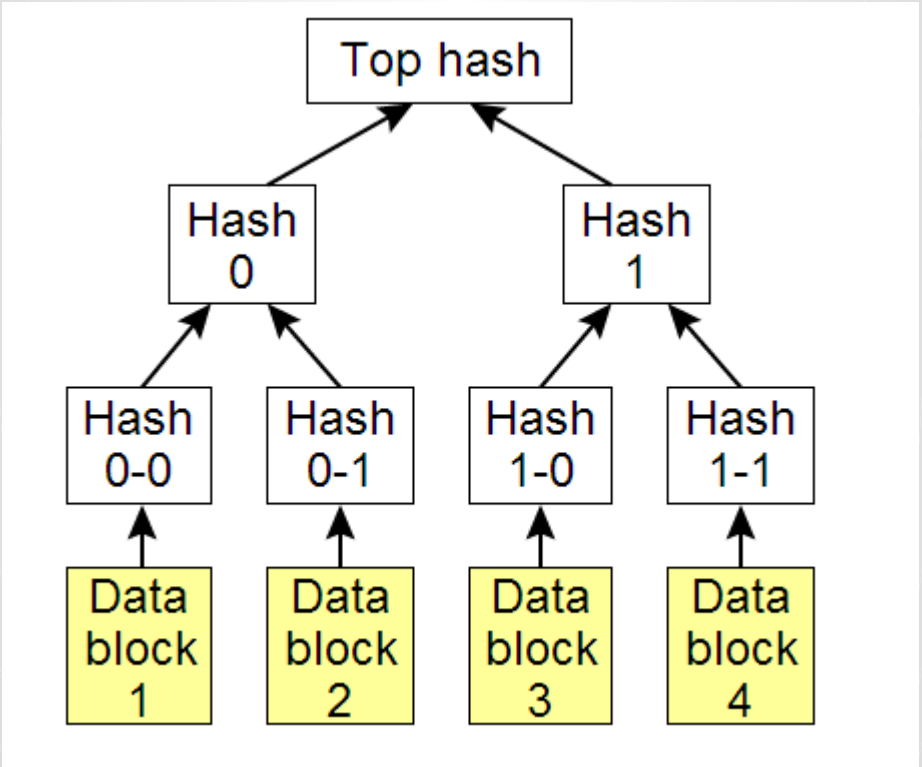

Start compaction
Merge latest data
Evict tombstones
Remove deletions


Evict tombstones
Remove deletions
Sort
Consolidate
Available disk space

Anti-Entropy

- Ensures synchronization of data across nodes
- Compares data checksums against neighboring nodes
- Uses Merkle trees (hash trees)
    - Snapshot of data sent to neighboring nodes
    - Created and broadcasted on every major compaction
    - If two nodes take snapshots within TREE_STORE_TIMEOUT of each other, snapshots are compared and data is synced.

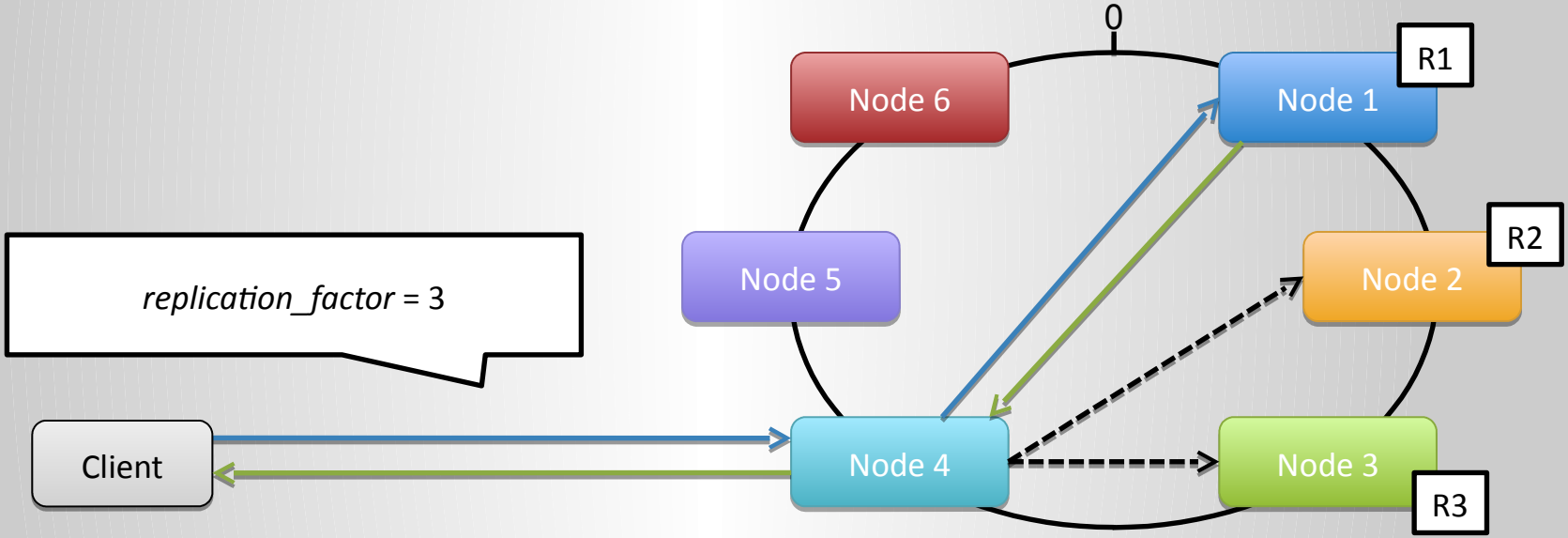
Merkle Tree


Top hash
Hash 0
Hash 1
Hash 0-0
Hash 0-1
Hash 1-0
Hash 1-1
Data block 1
Data block 2
Data block 3
Data block 4

Read Operations

- Read Repair
  - On read, nodes are queried until the number of nodes which respond with the most recent value meet a specified consistency level from ONE to ALL.
  - If the consistency level is not met, nodes are updated with the most recent value which is then returned.
  - If the consistency level is met, the value is returned and any nodes that reported old values are then updated.

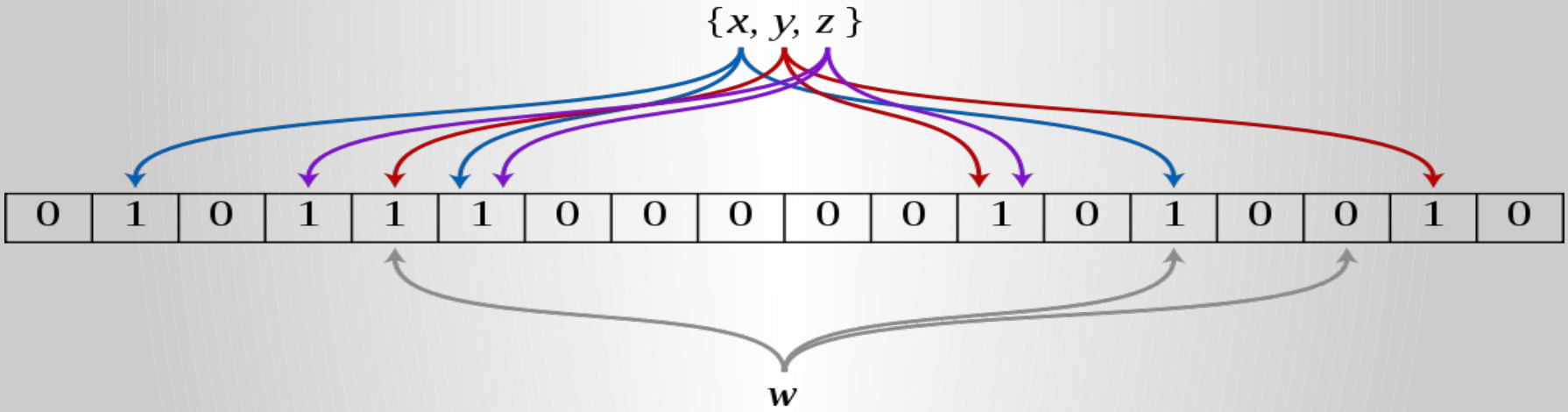
Read Repair



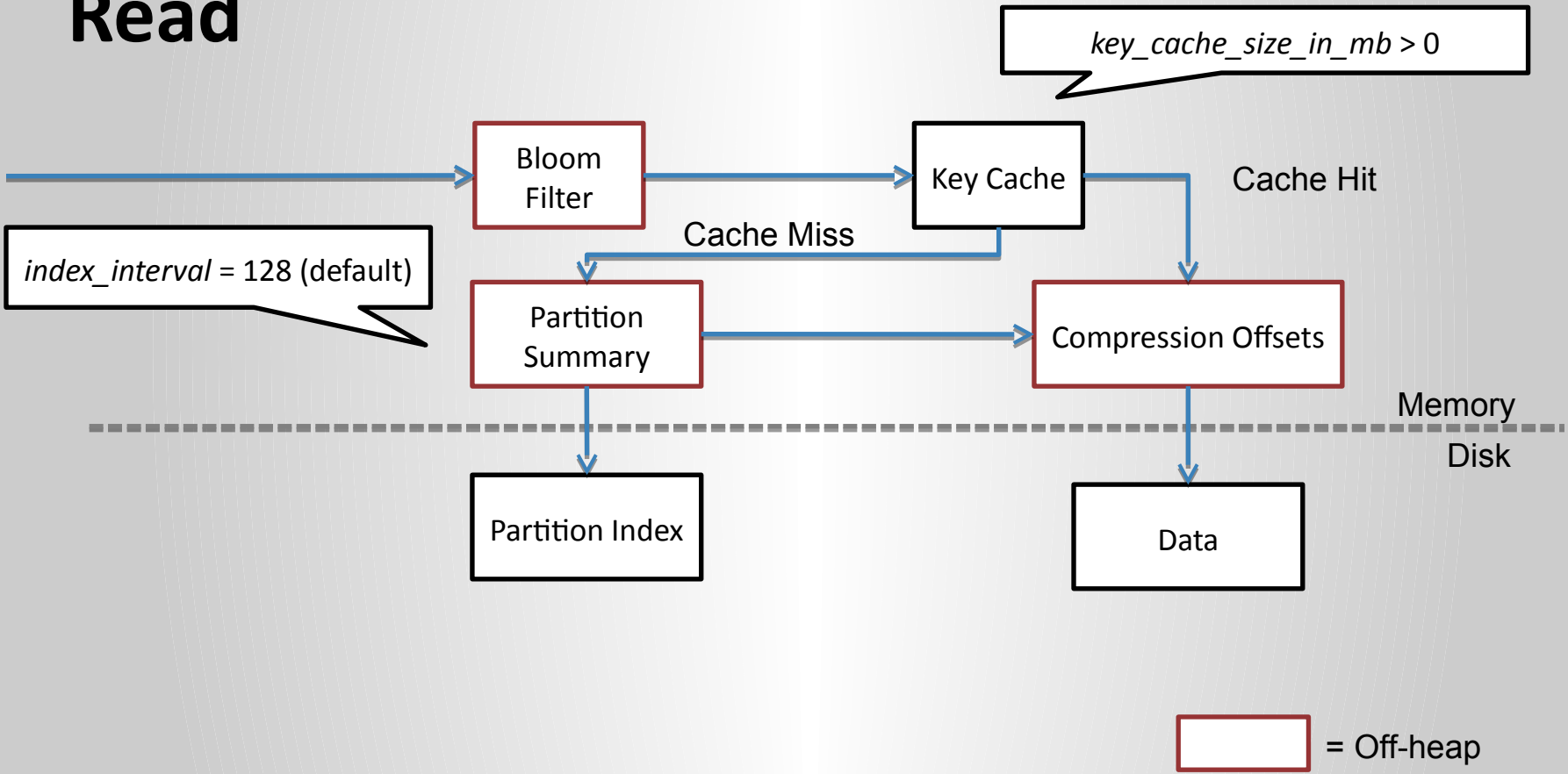
SELECT * FROM table USING CONSISTENCY ONE

Read Operations: Bloom Filters

- Bloom filters provide a fast way of checking if a value is not in a set.



Read


key_cache_size_in_mb > 0
Bloom Filter
Key Cache
Cache Hit
Cache Miss
index_interval = 128 (default)
Partition Summary
Compression Offsets
Memory
Disk
Partition Index
Data
= Off-heap

Cassandra: Conclusion

Cassandra Advantages

- perfect for time-series data
- high performance
- Decentralization
- nearly linear scalability
- replication support
- no single points of failure
- MapReduce support

Cassandra Weaknesses

- no referential integrity
  - no concept of JOIN
- querying options for retrieving data are limited
- sorting data is a design decision
  - no GROUP BY
- no support for atomic operations
  - if operation fails, changes can still occur
- first think about queries, then about data model

Cassandra: Points to Consider

- Cassandra is designed as a distributed database management system
  - use it when you have a lot of data spread across multiple servers
- Cassandra write performance is always excellent, but read performance depends on write patterns
  - it is important to spend enough time to design proper schema around the query pattern
- having a high-level understanding of some internals is a plus
  - ensures a design of a strong application built atop Cassandra

Questions?

References

- Lakshman, Avinash, and Prashant Malik. "Cassandra: a decentralized structured storage system." *ACM SIGOPS Operating Systems Review* 44.2 (2010): 35-40.
- Hewitt, Eben. *Cassandra: the definitive guide*. O'Reilly Media, 2010.
- http://www.datastax.com/documentation/cassandra/2.0/cassandra/architecture/architectureTOC.html
- http://www.slideshare.net/planetcassandra/a-deep-dive-into-understanding-apache-cassandra
- http://www.slideshare.net/DataStax/evaluating-apache-cassandra-as-a-cloud-database
- http://planetcassandra.org/functional-use-cases/
- http://marsmedia.info/en/cassandra-pros-cons-and-model.php
- http://www.slideshare.net/adrianco/migrating-netflix-from-oracle-to-global-cassandra
- http://wiki.apache.org/cassandra/CassandraLimitations