

Apache Cassandra



Column Stores

- Usage: read/write extensions
- Popular DBs: HBase, Cassandra



Document Store

- Usage: working with occasionally changing/consistent data
- Popular DBs: Couchbase, MongoDB



Graph Databases

- Usage: spatial data storage
- Popular DBs: Neo4J, Bigdata

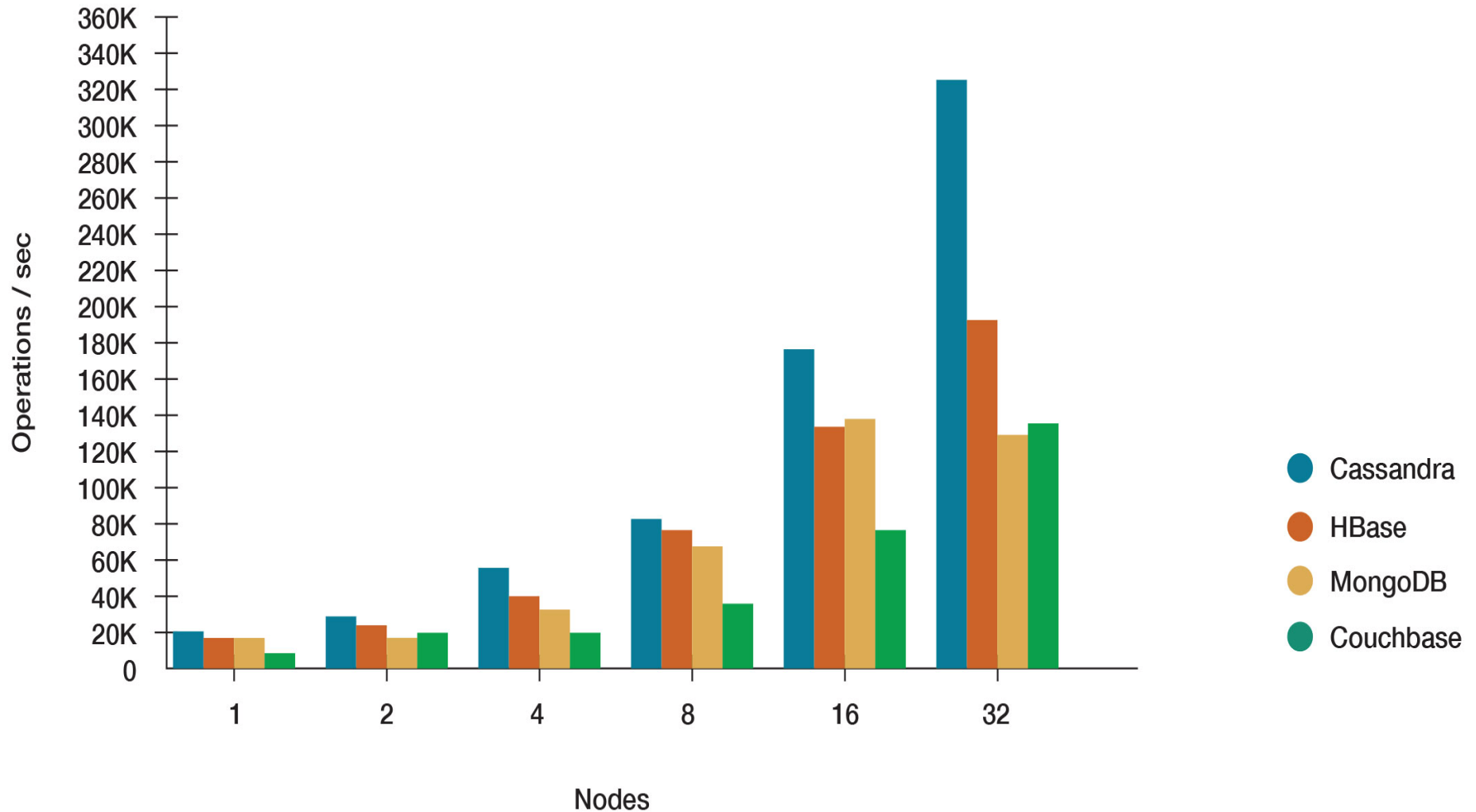


Key/Value

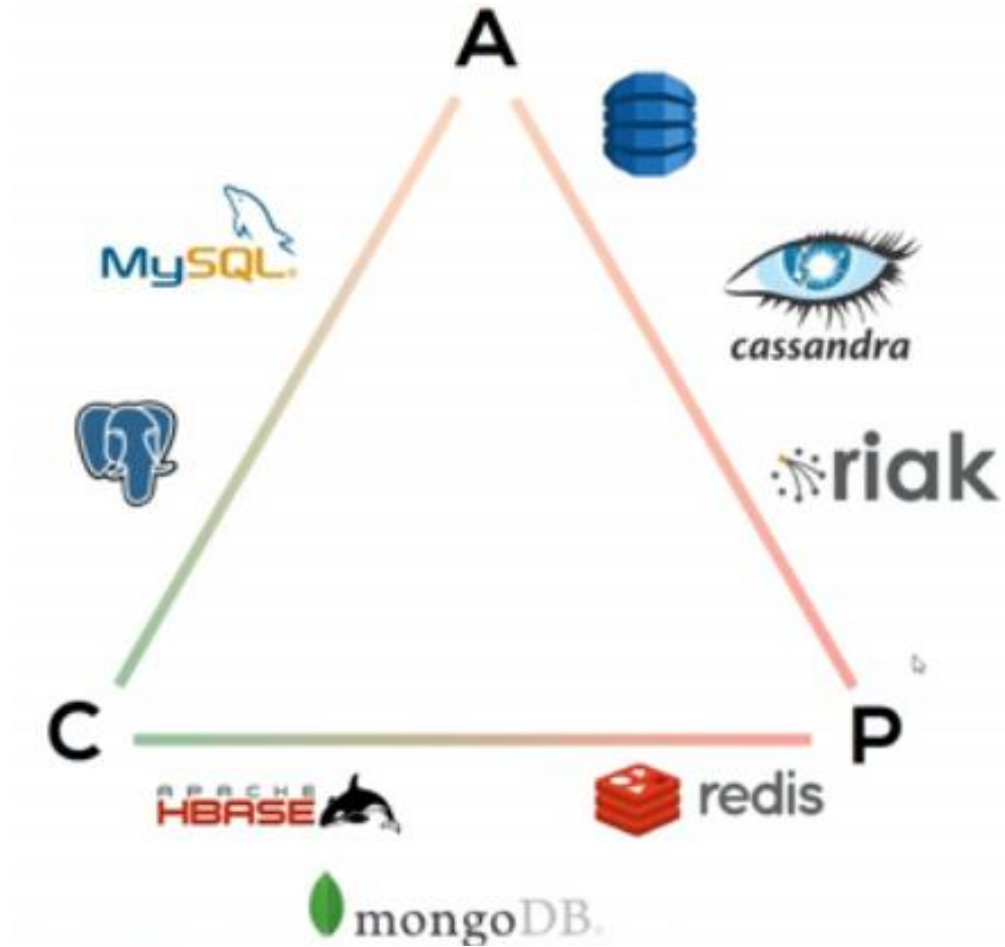
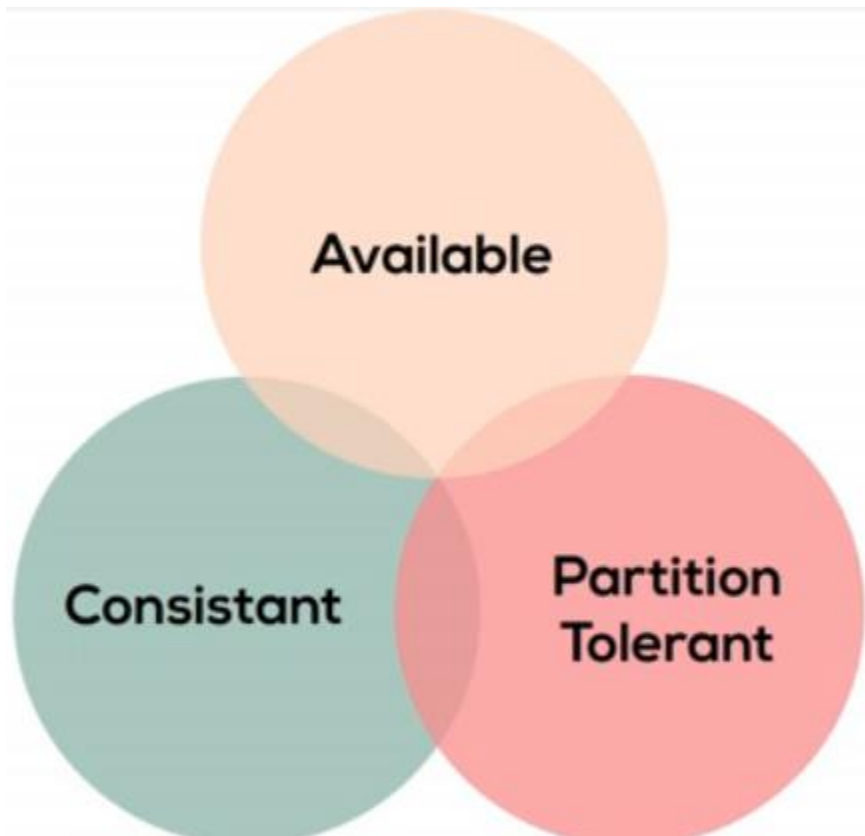
- Usage: briskly changing data and high availability
- Popular DBs: Riak, Redis

NoSQL Databases

Comparison



CAP's Theorem



Cassandra - Introduction

- Born at Facebook (Avinash Lakshman and Prashant Malik).
- After Facebook open sourced the code in 2008, it became an Apache Incubator project in 2009 and subsequently became a top-level Apache project in 2010.
- Built on Amazon's dynamo(Distributed Storage and replication techniques) and Google's BigTable(Data and storage engine model).
- Column-oriented database designed to support peer-to-peer symmetric nodes instead of master-slave architecture

Google

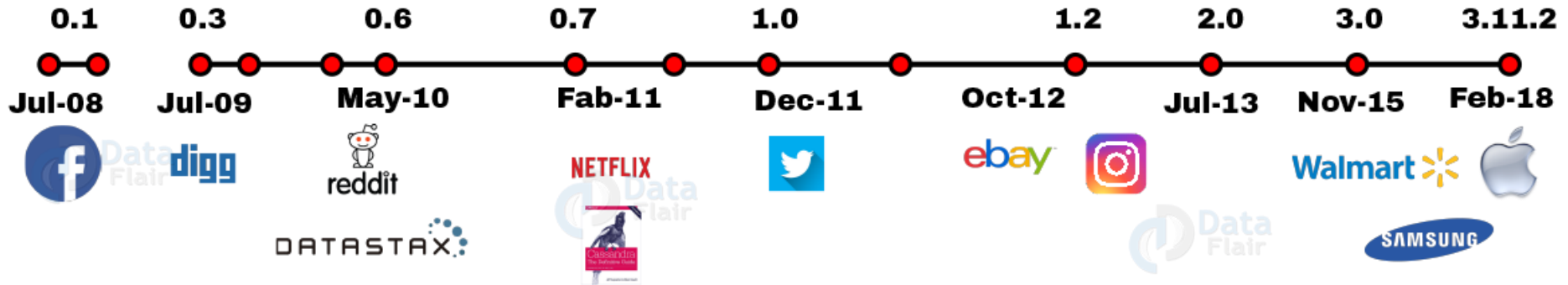
Bigtable, 2006

amazon.com

Dynamo, 2007

facebook.

OpenSource, 2008



Design Objectives

- Full multi-master database replication
- Global availability at low latency
- Scaling out on commodity hardware
- Linear throughput increase with each additional processor
- Online load balancing and cluster growth
- Partitioned key-oriented queries
- Flexible schema



Open Source

**Elastic
Scalability**

**High Availability
and
Fault Tolerance**

**Peer to
Peer Architecture**



Cassandra

FEATURES

**High
Performance**

**Column
Oriented**

**Tuneable
Consistency**

Schema-Free

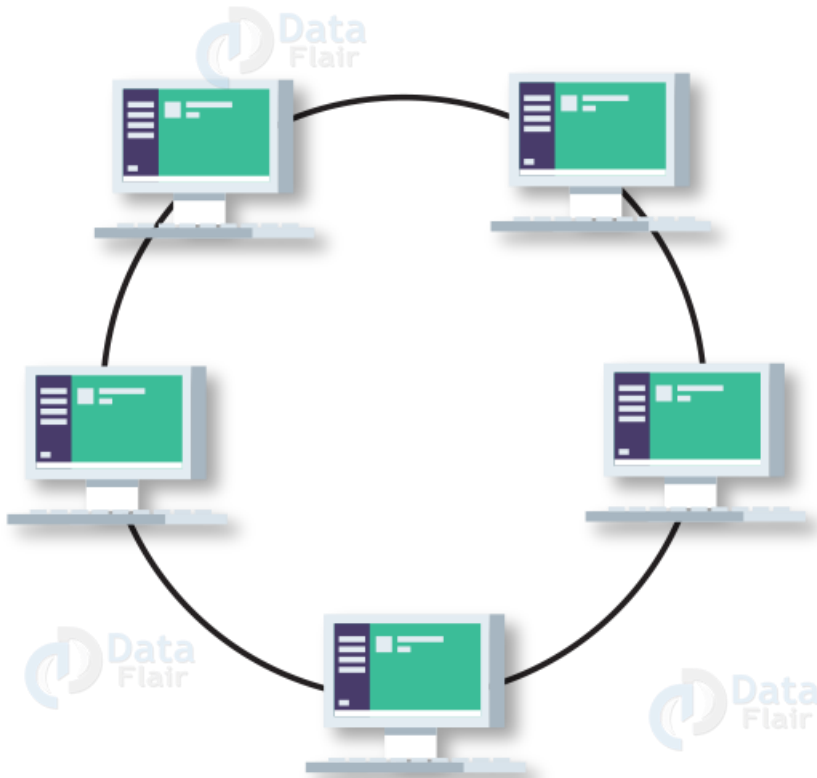
Open Source



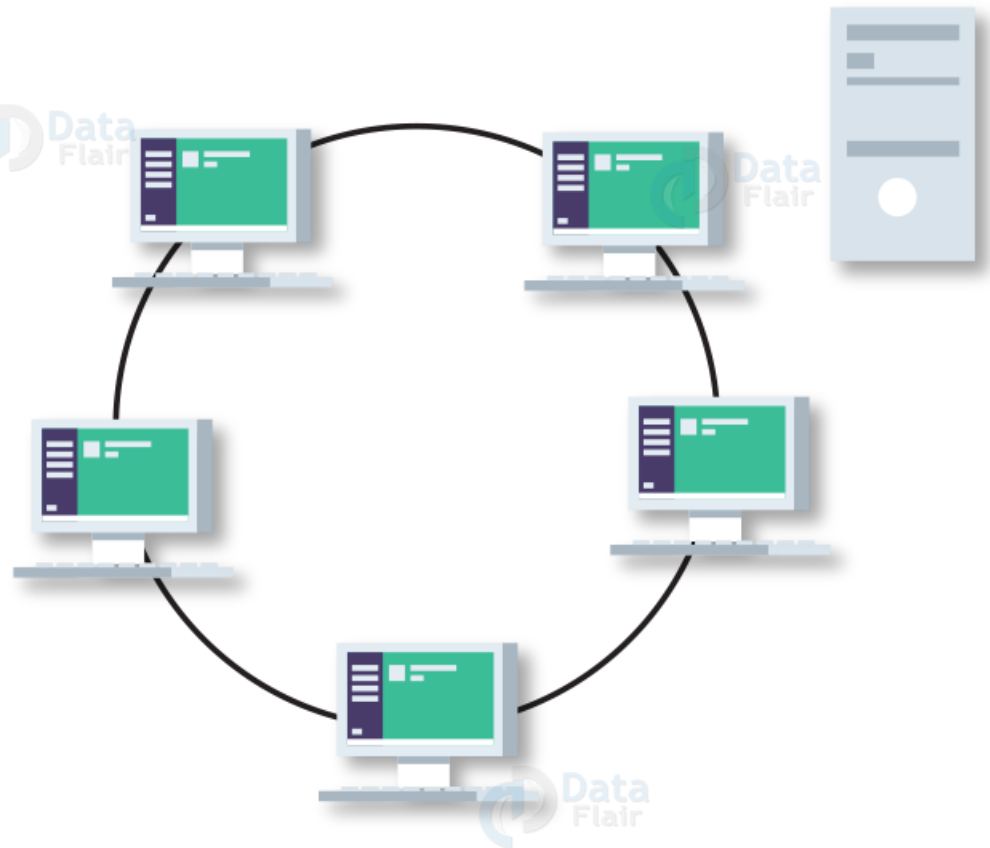
- Cassandra, though it is very powerful and reliable, is FREE!
- It is an open source project by Apache.
- Because of the open source feature, it gave birth to a huge Cassandra Community, where people discuss their queries and views.
- Possibility of integrating Cassandra with other Apache Open-source projects like Hadoop, Apache Hive , Apache pig etc.

Peer-to-Peer Network

Architecture



peer to peer

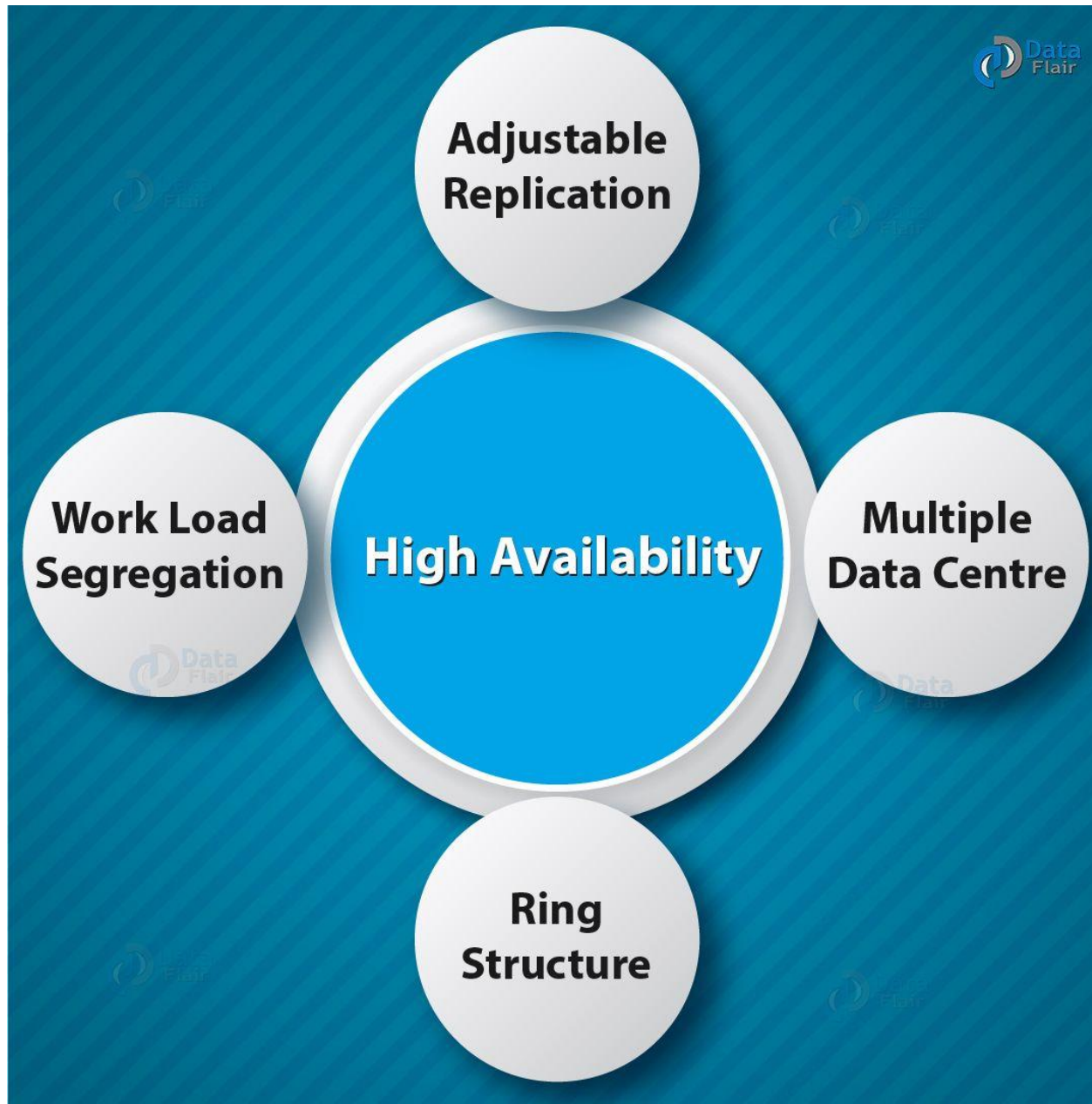


client - server

Elastic Scalability

- Can easily scale-up or scale-down the cluster in Cassandra.
- Flexibility for adding or deleting any number of nodes from the cluster without disturbances - no need of restarting the cluster while scaling up or scaling down.
- Because of this, Cassandra has a very high throughput for the highest number of nodes.
- Moreover, there is zero downtime or any pause during scaling. Hence read and write throughput increases simultaneously without delay.

High Availability and Fault Tolerance



Replication Factor

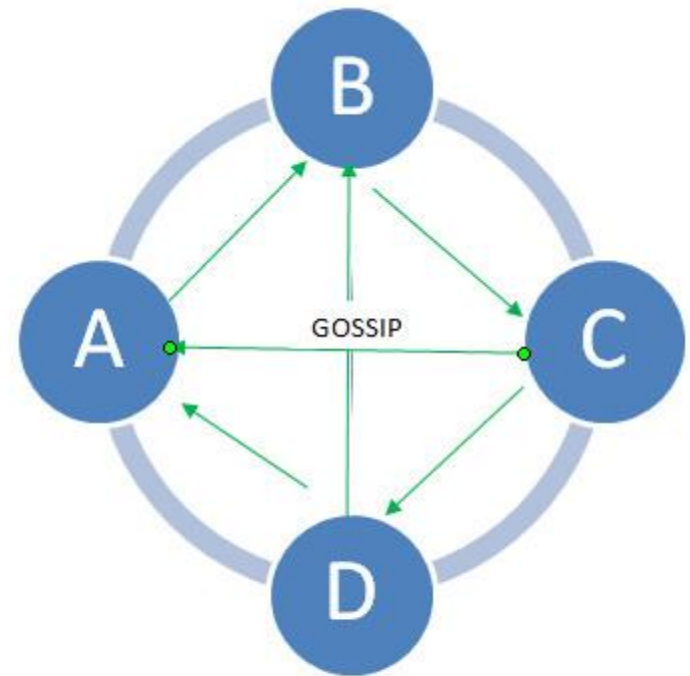
- Determines no. of copies of data across the cluster
- Ideally, it should be more than one and less than the no. of nodes in cluster
- Two replication strategies:
 - SimpleStrategy
 - NetworkTopologyStrategy

High Performance

- Cassandra database has one of the best performance as compared to other NoSQL database.
- Developers wanted to utilize the capabilities of many multi-core machines. This is the base of development of Cassandra.
- Cassandra has proven itself to be excellently reliable when it comes to a large set of data.
- Therefore, Cassandra is used by a lot of organizations which deal with huge amount of data on a daily basis. Furthermore, they are ensured about the data, as they cannot afford to lose the data.

Gossip and Failure Detection

- Gossip is used for intra-ring communication
- The gossip process runs every second for every node and exchange state messages with up to three other nodes in the cluster.
- Each node independently will always select one to three peers to gossip with. It will always select a live peer (if any) in the cluster.
- Eases the discovery and sharing of location and state information with other nodes in the cluster



PEER TO PEER DISTRIBUTION
MODEL OF CASSANDRA

Column-Oriented

- Cassandra's data model is column-oriented.
- In other databases, column name contains metadata, whereas, columns in Cassandra also contain actual data.
- In Cassandra, columns are stored based on column names.

- In column oriented database, each column can be stored in separate files on disk.
- So we can just access the columns of our interest unlike in row-oriented databases.

Employee Table				
Id	Name	Age	Gender	Car
1	{Name : Brian}	{Age : 21}	{Gender : M}	{Car : BMW}
2	{Name : John}	{Age : 43}	{Gender : M}	{Car : BMW}
3	{Name : Bob}	{Age : 45}	{Gender : M}	{Car : BMW}
4	{Name : Frank}	{Age : 23}	{Gender : M}	{Car : Audi}
5	{Name : Olivia}	{Age : 35}	{Gender : F}	{Car : Audi}
6	{Name : Emma}	{Age : 32}	{Gender : F}	{Car : Audi}
7	{Name : Sophia}	{Age : 45}	{Gender : F}	
8	{Name : Mia}	{Age : 23}	{Gender : F}	

Id	Name	Age	Gender	Car
1	{Name : Brian}	{Age : 21}	{Gender : M} * 4	{Car : BMW}
2	{Name : John}	{Age : 43}		{Car : BMW}
3	{Name : Bob}	{Age : 45}		{Car : BMW}
4	{Name : Frank}	{Age : 23}		{Car : Audi}
5	{Name : Olivia}	{Age : 35}	{Gender : F} * 4	{Car : Audi}
6	{Name : Emma}	{Age : 32}		{Car : Audi}
7	{Name : Sophia}	{Age : 45}		
8	{Name : Mia}	{Age : 23}		

Schema-Free

- There is a flexibility in Cassandra to create columns within the rows. That is, Cassandra is known as the schema-optional data model.
- Since each row may not have the same set of columns, there is no need to show all the columns needed by the application at the surface.
- Therefore, Schema-less/Schema-free database in a column family is one of the most important Cassandra features.

Cassandra consistency level

- The **Cassandra consistency level** is defined as the minimum number of **Cassandra** nodes that must acknowledge a read or write operation before the operation can be considered successful.
- How many nodes will be read before responding to the client is based on the consistency level specified by the client. If client-specified consistency level is not met, there is possibility that few nodes may respond with out-of-date copies, in which case read operation blocks.

Tunable Consistency



- Two types of consistency in Cassandra:
 - Eventual consistency and Strong Consistency.
- Eventual consistency makes sure that the client is acknowledged as soon as a part of the cluster acknowledges the write. (Used when performance matters)
- Whereas, Strong consistency make sure that any update is broadcasted to all the nodes or machines where the particular data is suited.
- Cassandra can cash in on any of them depending on requirements.

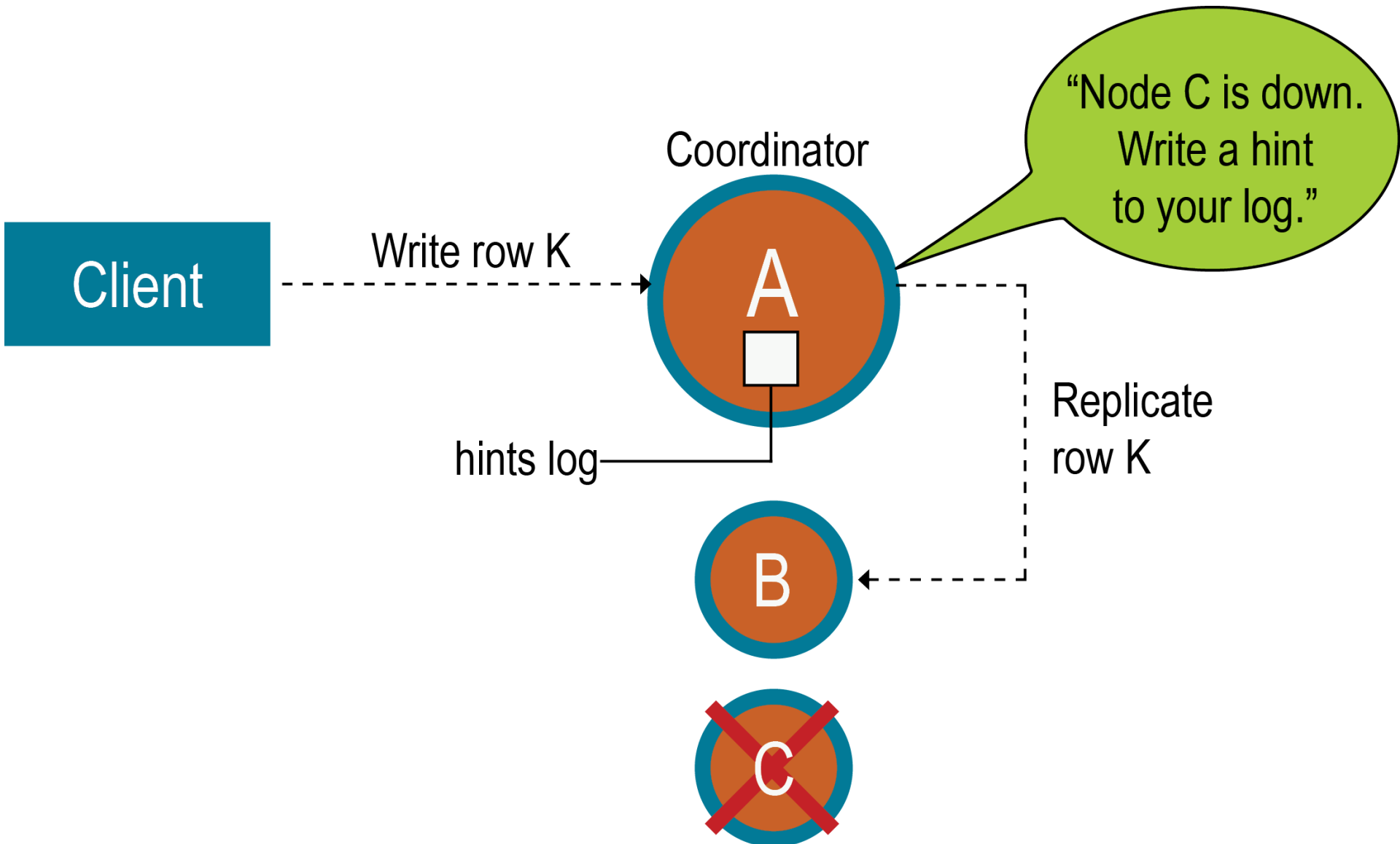
Anti-Entropy and Read Repair

- For repairing unread data, Cassandra uses an anti-entropy version of the gossip protocol.
- Anti-entropy implies comparing all the replicas of each piece of data and updating each replica to the newest version.
- The read repair operation is performed either before or after returning the value to the client as per the specified consistency level.

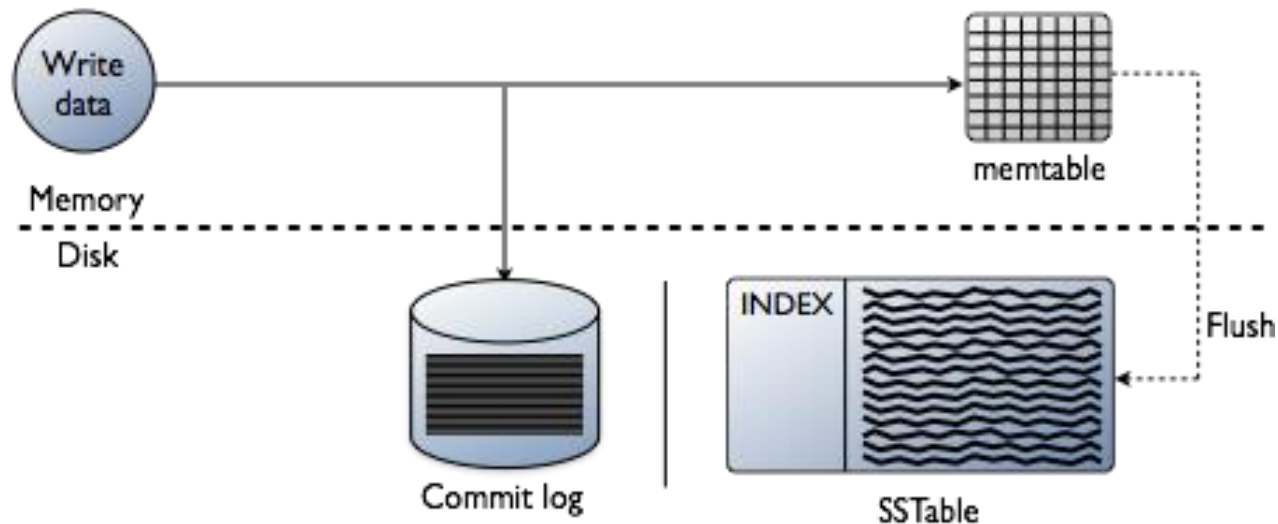
Hinted Handoffs

- Cassandra works on philosophy that it should be always available for writes.
- This is achieved by hinted handoffs.
- Assume
 - A cluster of three nodes – Node A, Node B and Node C.
 - Node C is down for some reason
 - Replication factor is 2
- Hint contains following information:
 - Location of the node on which the replica is to be placed
 - Version metadata
 - The actual data

Hinted Handoffs



Writes in Cassandra



- A write operation once completed is first written to commit log.
- Next it is pushed to a memory resident data structure called Memtable, which has a predefined threshold value.
- When no. of objects reaches a threshold, then the contents of Memtable are flushed to the disk in a file called SSTable(Sorted String Table)
- Flushing is a non-blocking operation.

Partitioner

- Partitioner
 - takes a call on how to distribute data on the various nodes in a cluster.
 - also determines the node on which to place the very first copy of data
- It is a hash function to compute the token of the partition key which helps to identify a row uniquely.
- Both the Murmur3Partitioner and RandomPartitioner use tokens(it's hash) to help assign equal portions of data to each node and evenly distribute data from all the tables throughout the ring or other grouping, such as a keyspace.

Relational Tables

Que : How many employee of a company drive BMW?

Employee				
ID	FirstName	Surname	CompanyCarId	Salary
1	Elvis	Presley	1	\$30000
2	David	Bowie	2	\$40000
3	Kylie	Jenner	3	\$60000
4	Elton	John	1	\$20000
5	Mariah	Carey	2	\$30000
6	Justin	Bieber	4	\$30000
7	Selena	Gomez	5	\$50000

Company Car				
ID	Make	Model	Cost	Engine
1	BMW	5 Series	\$50000	1.8
2	Audi	A6	\$55000	1.6
3	Mercedes	C-Class	\$60000	1.6
4	Mercedes	A-Class	\$30000	1.4
5	BMW	3 Series	\$35000	1.6

Joins

- Joins – Joining two tables
- Not possible in distributed databases
- Solution – Query First Approach (Queries are reflected in tables)
- Design tables for queries
- May end up having duplicate data in multiple tables, but that is the only efficient approach for distributed databases, also called denormalization

Query First Approach-Data Model

Employee By Car Make				
Make	EmployeeID	Employee Firstname	Employee Surname	Salary
BMW	1	Elvis	Presley	\$30000
BMW	4	Elton	John	\$20000
BMW	7	Selena	Gomez	\$30000
Audi	2	David	Bowie	\$40000
Audi	5	Mariah	Carey	\$30000
Mercedes	3	Kylie	Jenner	\$60000
Mercedes	6	Justin	Bieber	\$50000

Company Car By ID				
ID	Make	Model	Cost	Engine
1	BMW	5 Series	\$50000	1.8
2	Audi	A6	\$55000	1.6
3	Mercedes	C-Class	\$60000	1.6
4	Mercedes	A-Class	\$30000	1.4
5	BMW	3 Series	\$35000	1.6

Cassandra Tables

Partition Keys

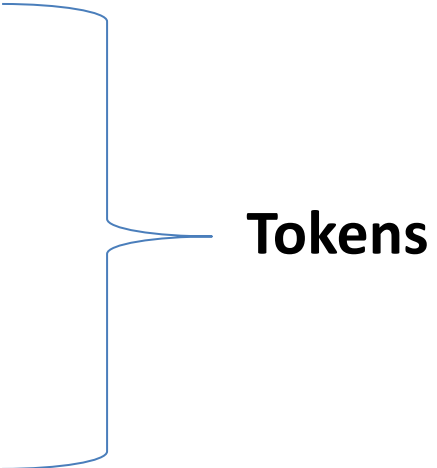
Employee By Car Make				
BMW*3	{Id : 1}	{Firstname : Elvis}	{Surname : Presley}	{Salary : \$30000}
	{Id : 4}	{Firstname : Elton}	{Surname : John}	{Salary : \$20000}
	{Id : 7}	{Firstname : Selena}	{Surname : Gomez}	{Salary : \$30000}
Audi*2	{Id : 2}	{Firstname : David}		{Salary : \$40000}
	{Id : 5}	{Firstname : Mariah}	{Surname : Carey}	{Salary : \$30000}
Mercedes*2	{Id : 3}	{Firstname : Kylie}	{Surname : Jenner}	
	{Id : 6}	{Firstname : Justin}	{Surname : Bieber}	{Salary : \$50000}

Company Car By ID				
1	{Make : BMW}	{Model : 5 Series}	{Cost : \$50000}	{Engine : 1.8}
2	{Make : Audi}	{Model : A6}	{Cost : \$55000}	{Engine : 1.6}
3	{Make : Mercedes}	{Model : C-Class}	{Cost : \$60000}	{Engine : 1.6}
4	{Make : Mercedes}	{Model : A-Class}	{Cost : \$30000}	{Engine : 1.4}
5	{Make : BMW}	{Model : 3 Series}	{Cost : \$35000}	{Engine : 1.6}

Partition keys

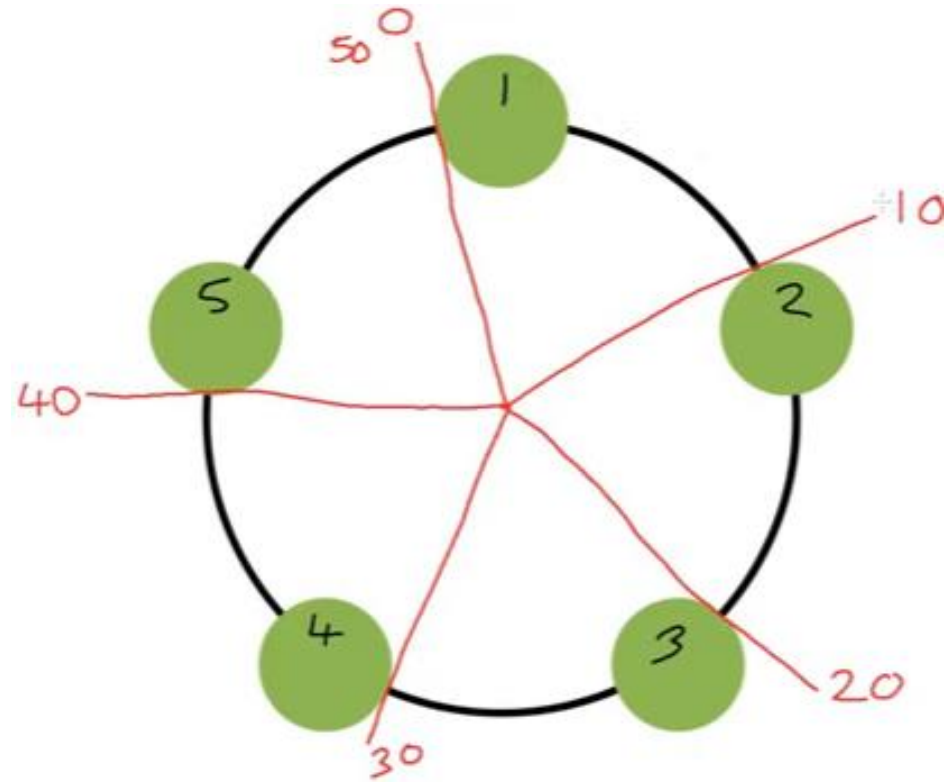
- Data belonging to same partition will be written on same node.
- In cassandra, data is accessed by partition keys and not by primary key though primary key may be part of the table.
- E.g. For Employee Table – Car Make, For Car table – Car ID
- Hash function is used for creating partitioning

Example

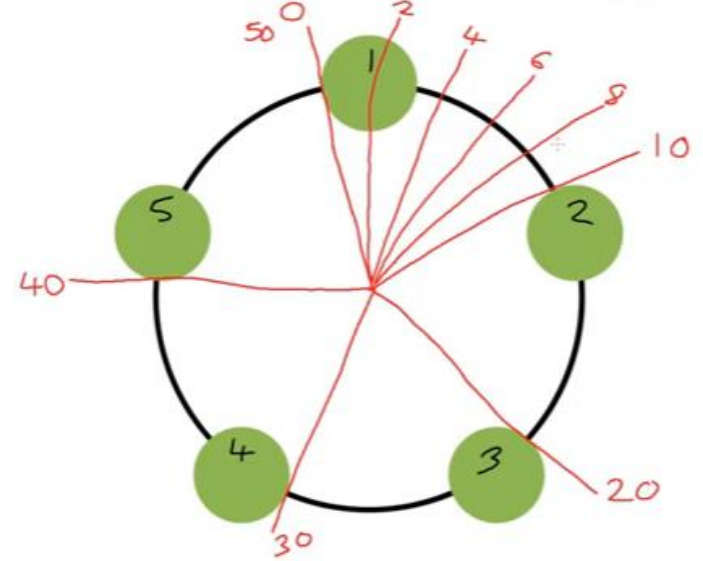
- BMW -> Hash Function -> 15
 - Audi -> Hash Function -> 22
 - Merc -> Hash Function -> 43
 - BMW -> Hash Function -> 15
- 
- Tokens
- Tokens are of 64 bit integers in cassandra.
 - Ranges from -2^{64} to $2^{63} - 1$

Ring

- Ideally, each node is given a token
- In real life, each node can store a range of tokens.
- E.g. node 2 can store tokens >10 and <20 .



Tokens



- In real life, a range assigned to a node can be much larger and Cassandra does not want to do this.
- Instead it assigns multiple token ranges to a node. This concept is called virtual nodes.
- Any token in those ranges still go to that single node, but this gives greater flexibility.

Advantages of Virtual Nodes

- Makes the process of assigning tokens to nodes less manual
- Nodes can be added seamlessly in cassandra
 - Provides more storage
 - Gives high throughput
- Nodes with different capacity can be assigned different number of virtual nodes.
 - E.g. Node having 256 GB of storage can be assigned 8 virtual nodes and node having 64 GB of storage can be assigned 2 virtual nodes.
- Default value in cassandra is 256 virtual nodes

Terms in Cassandra

- A Cluster is a collection of Data Centers.
- A Data Center is a collection of Racks.
- A Rack is a collection of Servers.
- A Server contains 256 virtual nodes (or vnodes) by default.
- A vnode is the data storage layer within a server.
- The hierarchy of elements in Cassandra is:
 - Cluster
 - Data center(s)
 - Rack(s)
 - Server(s)
 - Node (more accurately, a vnode)

References

- <https://data-flair.training/blogs/apache-cassandra-tutorial/>
- <https://docs.datastax.com/en/archived/cassandra/3.0/cassandra/operations/opsRepairNodesHintedHandoff.html>
- <https://docs.datastax.com/en/archived/cassandra/3.0/cassandra/dml/dmlHowDataWritten.html>
- <https://stackoverflow.com/questions/28196440/what-are-the-differences-between-a-node-a-cluster-and-a-datacenter-in-a-cassand>
- <https://www.youtube.com/watch?v=5LTs-G308wY&list=PLalrWAGybpB-L1PGA-NfFu2uiWHEsdscD&index=5>
- <https://www.youtube.com/watch?v=Soaod2WRmlg&list=PLalrWAGybpB-L1PGA-NfFu2uiWHEsdscD&index=6>
- <https://www.youtube.com/watch?v=yyVbDBSMnUw&list=PLalrWAGybpB-L1PGA-NfFu2uiWHEsdscD&index=7>
- <https://www.youtube.com/watch?v=91ZwkO07xHU&list=PLalrWAGybpB-L1PGA-NfFu2uiWHEsdscD&index=8>
- <https://www.linkedin.com/pulse/gossip-protocol-inside-apache-cassandra-soham-saha>
- <https://www.appdynamics.com/blog/engineering/a-newbie-guide-to-databases/>
- <https://www.datastax.com/nosql-databases/benchmarks-cassandra-vs-mongodb-vs-hbase>
- <https://cassandra.apache.org/>

Thank You!