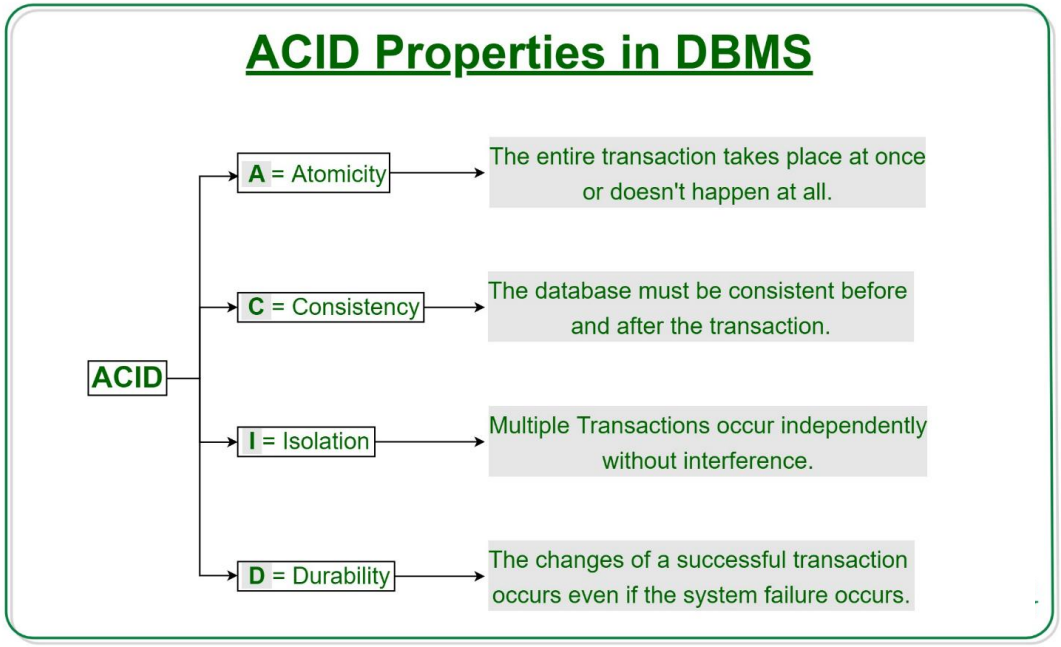
NoSQL Databases

Sonali Agarwal
IIIT Allahabad

BASE Transactions

- Acronym contrived to be the opposite of ACID
  - Basically Available,
  - Soft state,
  - Eventually Consistent
- Characteristics
  - Weak consistency – stale data OK
  - Availability first
  - Best effort
  - Approximate answers OK
  - Aggressive (optimistic)
  - Simpler and faster



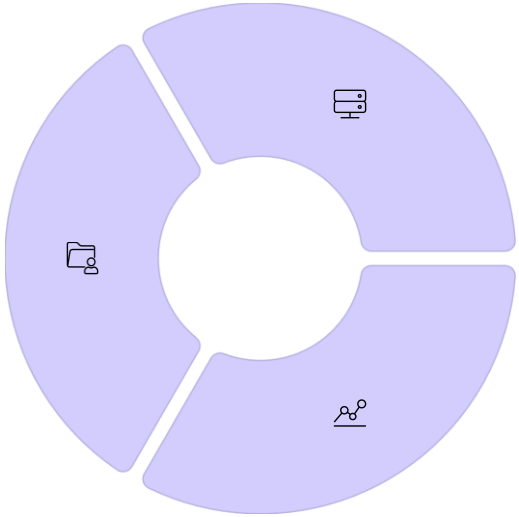
The three pillars of CAP Theorem

Concept	Definition	Analogy
Consistency (C)	Every read receives the most recent write or an error.	All users see the same version of data at the same time.
Availability (A)	Every request receives a (non-error) response — even if it might not have the latest data.	The system is always up and responds, even if it's not perfectly up to date.
Partition Tolerance (P)	The system continues to function despite network failures or message loss between nodes.	Even if one part of the system is cut off due to a network problem, the rest continues working.

Understanding the CAP Theorem

Consistency

All nodes see the same data at the same time. Every read receives the most recent write or an error.



Availability

Every request receives a response, even if some nodes have failed or are unreachable.

Partition Tolerance

The system continues operating despite network partitions that prevent some nodes from communicating.

Brewer's CAP Theorem

You can have at most **two** of these three properties for any distributed system. When network partitions occur (which they inevitably do), you must choose between consistency and availability.

Three Categories of Systems

Type	Trade-off	Behavior	Example Systems
CP (Consistency + Partition Tolerance)	Sacrifices **Availability**	When a network partition occurs, the system chooses to be consistent, but some nodes may not respond.	**MongoDB (default), HBase, Zookeeper**
AP (Availability + Partition Tolerance)	Sacrifices **Consistency**	The system stays available even during network partitions, but some reads may see stale data until synchronization.	**Cassandra, CouchDB, DynamoDB**
CA (Consistency + Availability)	Sacrifices **Partition Tolerance**	Works well in a single-node or local network without partitions; cannot guarantee both when network failure happens.	**Traditional RDBMS (MySQL, PostgreSQL)**

Example

E-Commerce Inventory System:

Suppose an online store uses a distributed database across two data centers, one in Delhi and one in Mumbai.

A user in Delhi buys the last available iPhone.

Simultaneously, a user in Mumbai also tries to buy it.

Case 1: CP System (e.g., MongoDB)
The database ensures **only one of them** successfully buys it (Consistency).
But if the Delhi–Mumbai connection is down (Partition), the system may temporarily **reject** one transaction (less Availability).

Example

Case 2: AP System (e.g., Cassandra)

- Both users' transactions are **accepted immediately** (Availability).
- Later, when the network is restored, the system detects the conflict and resolves it (Eventual Consistency).

Case 3: CA System (e.g., MySQL in a single data center)

- Both reads and writes are **always consistent** and **available**, but if network failure splits the data centers, the system simply stops serving one side (no Partition Tolerance).

In real distributed systems, Partition Tolerance is mandatory, so the real trade-off is between Consistency and Availability.
That's why NoSQL databases are often labeled as either:

- CP systems (MongoDB, HBase)
- AP systems (Cassandra, DynamoDB)

Eventual Consistency Explained

Eventual consistency is a consistency model used in distributed computing to achieve high availability. It guarantees that, if no new updates are made to a data item, eventually all accesses to that item will return the same value.

Update Propagation

When data is updated on one node, the change propagates to other nodes over time through replication mechanisms

Temporary Inconsistency

During propagation, different nodes may return different values for the same data—this is the "soft state" period

Convergence

Once propagation completes and no new updates occur, all nodes converge to the same consistent state

Node Failure Handling

If a node cannot be reached, it's eventually removed from service, ensuring the update reaches all active nodes

This model is known as **BASE** (**B**asically **A**vailable, **S**oft state, **E**ventually consistent) as opposed to ACID. The soft state means copies of data may be temporarily inconsistent, but they become consistent at some later time if no more updates occur.

CAP Theorem-Availability

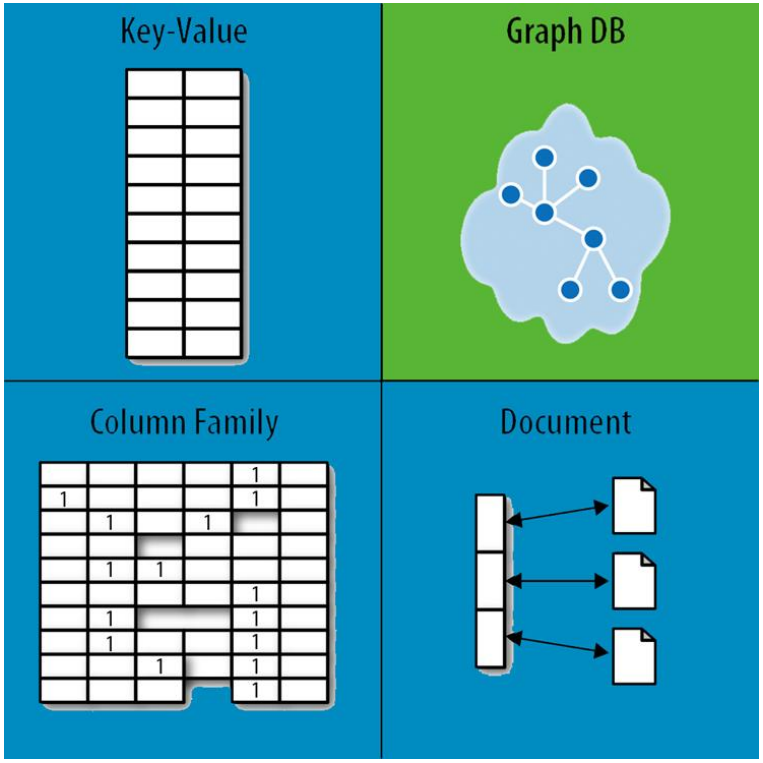
- Traditionally, thought of as the server/process available five 9's (99.999 %).
- However, for large node system, at almost any point in time there's a good chance that a node is either down or there is a network disruption among the nodes.
  - **Want a system that is resilient in the face of network disruption**

CAP Theorem-Partition

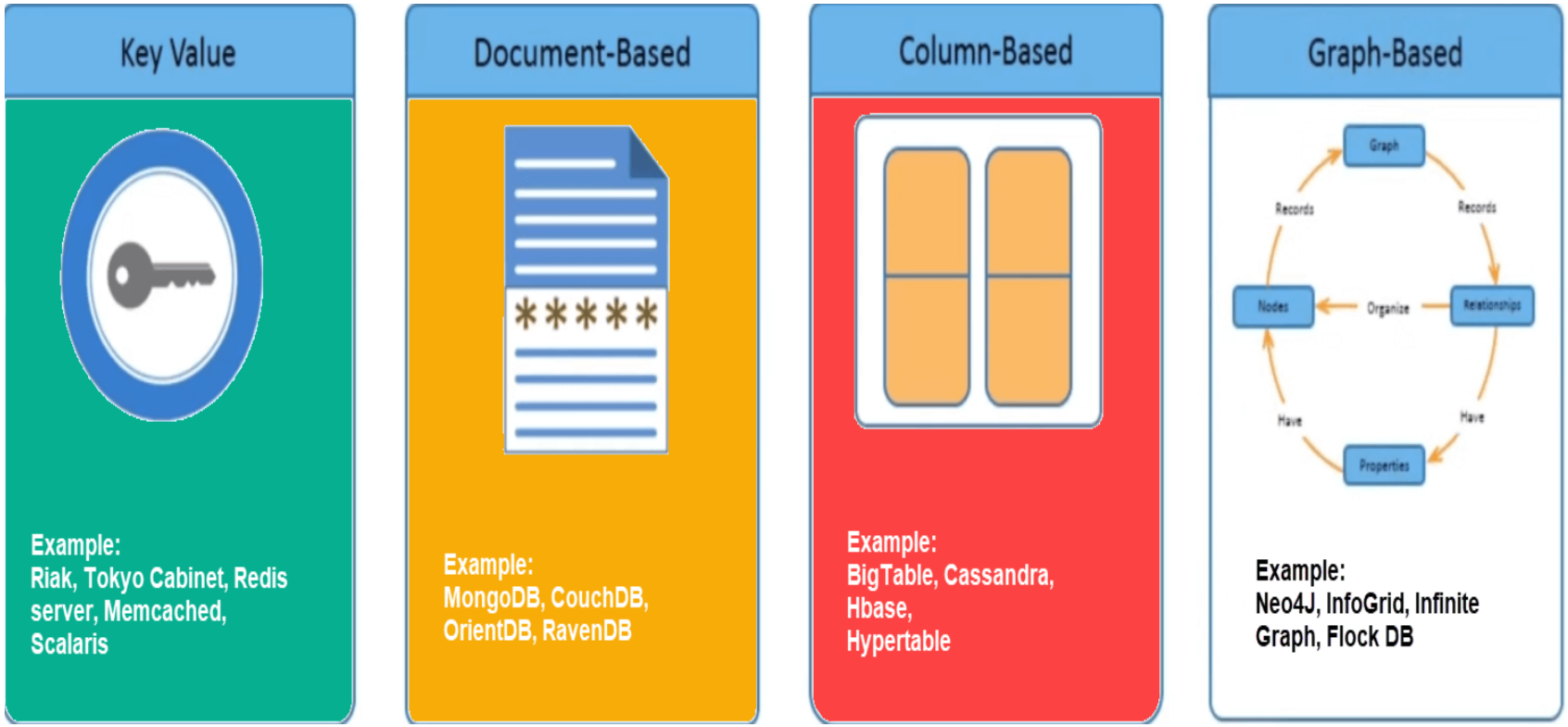
- Very large systems will partition at some point
- Choose one of consistency or availability
- Traditional database choose consistency
- Most Web applications choose availability

Types of NoSQL Databases

- **Key-Value Store:** are the simplest NoSQL databases. Every single item in the database is stored as an attribute name (or 'key'), together with its value.
- **Document databases:** pair each key with a complex data structure known as a document.
- **Sorted ordered Column Store:** Optimized for queries over large datasets, and store columns of data together, instead of rows
- **Graph Databases:** are used to store information about networks of data, such as social connections.

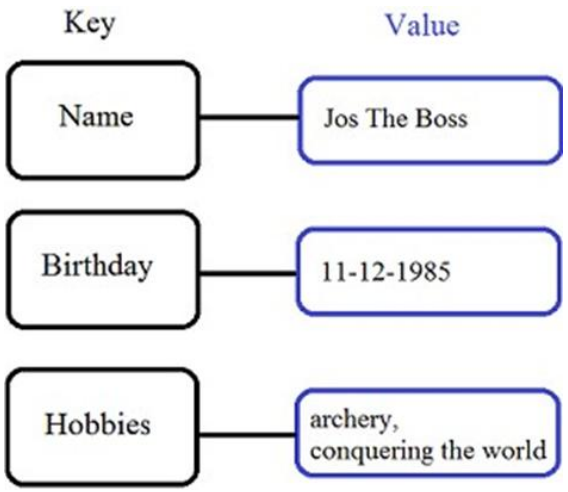


Types of NoSQL Databases



Key/Value stores

- Data is stored in key/value pairs. It is designed in such a way to handle lots of data and heavy load.
- Key-value pair storage databases store data as a hash table where each key is unique, and the value can be a JSON, BLOB(Binary Large Objects), string, etc.
- For example, a key-value pair may contain a key like “Website” associated with a value like “iiita.ac.in”.



Key features of the key-value store:

Simplicity.
Scalability.
Speed.

Key/Value stores

- It is one of the most basic NoSQL database example.
- This kind of NoSQL database is used as a collection, dictionaries, associative arrays, etc. Key value stores help the developer to store schema-less data.
- They work best for shopping cart contents.
- Redis, Dynamo, Riak are some NoSQL examples of key-value store DataBases.
- They are all based on Amazon's Dynamo paper.

Document Databases (Document Store)

- The central concept is the notion of a "document“ which corresponds to a row in RDBMS.
- A document comes in some standard formats like JSON.
- Documents are addressed in the database via a unique key that represents that document.
- The database offers an API or query language that retrieves documents based on their contents.
- Documents are schema free, i.e., different documents can have structures and schema that differ from one another.

Document Databases (Document Store)

- Relational Vs. Document

Col1	Col2	Col3	Col4
Data	Data	Data	Data
Data	Data	Data	Data
Data	Data	Data	Data

Document 1

```
{
  "prop1": data,
  "prop2": data,
  "prop3": data,
  "prop4": data
}
```

Document 2

```
{
  "prop1": data,
  "prop2": data,
  "prop3": data,
  "prop4": data
}
```

Document 3

```
{
  "prop1": data,
  "prop2": data,
  "prop3": data,
  "prop4": data
}
```

In this diagram on your left you can see we have rows and columns, and in the right, we have a document database which has a similar structure to JSON. Now for the relational database, you have to know what columns you have and so on. However, for a document database, you have data store like JSON object. You do not require to define which make it flexible.

Document Databases, JSON

Example Document (MongoDB):

```
{
  "_id": 101,
  "name": "Sonali Agarwal",
  "department": "IT",
  "courses": ["Big Data", "Software Engineering"]
}
```

- **Best suited for:**
  - Content management systems
  - Blogs, e-commerce catalogs, logs
- **Examples:**
  MongoDB, CouchDB, Firebase Firestore

- Example: CouchDB
  - http://couchdb.apache.org/
  - BBC
- Example: MongoDB
  - http://www.mongodb.org/
  - Foursquare, Shutterfly
- JSON – JavaScript Object Notation

Sorted Ordered Column-Oriented Stores

- Here is an example of a simple database table with 4 columns and 3 rows.

ID	Last	First	Bonus
1	Saravanan	Nandhini	8000
2	Varshini	Usha	4000
3	D	Shanthi	10000

- In a **row-oriented** database system, the data would be stored like this:

  1,Saravanan,Nandhini,8000; 2,Varshini,Usha,4000; 3,D,Shanthi,10000;

- In a **column-oriented** database system, the data would be stored like this:

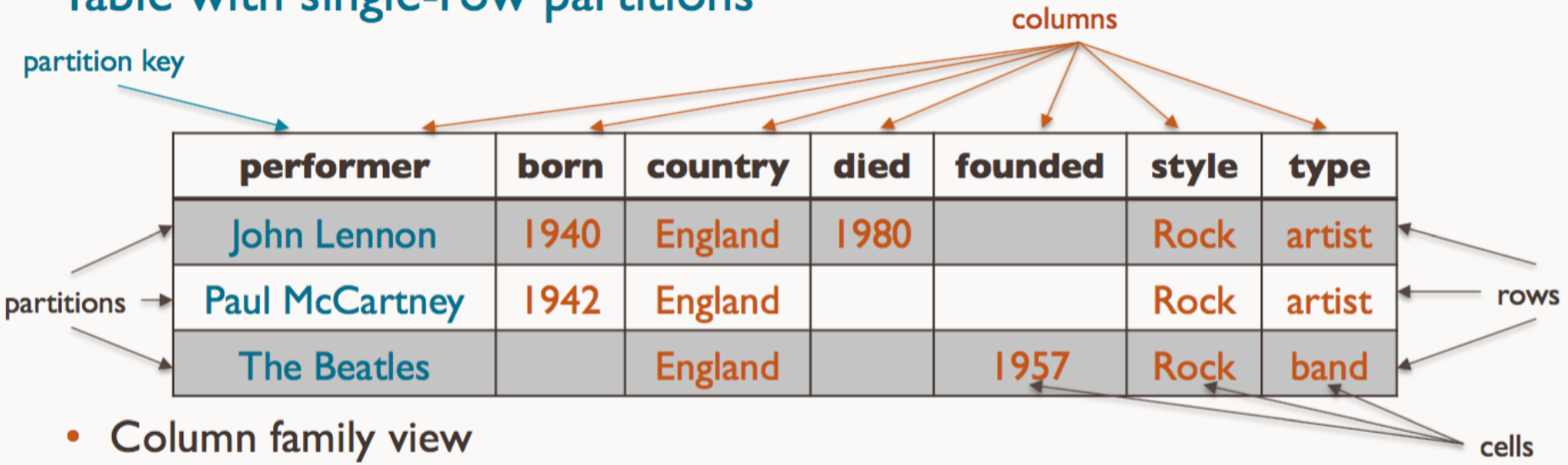
  1,2,3; Saravanan,Varshini,D; Nandhini,Usha,Shanthi; 8000,4000,10000;

Sorted Ordered Column-Oriented Stores

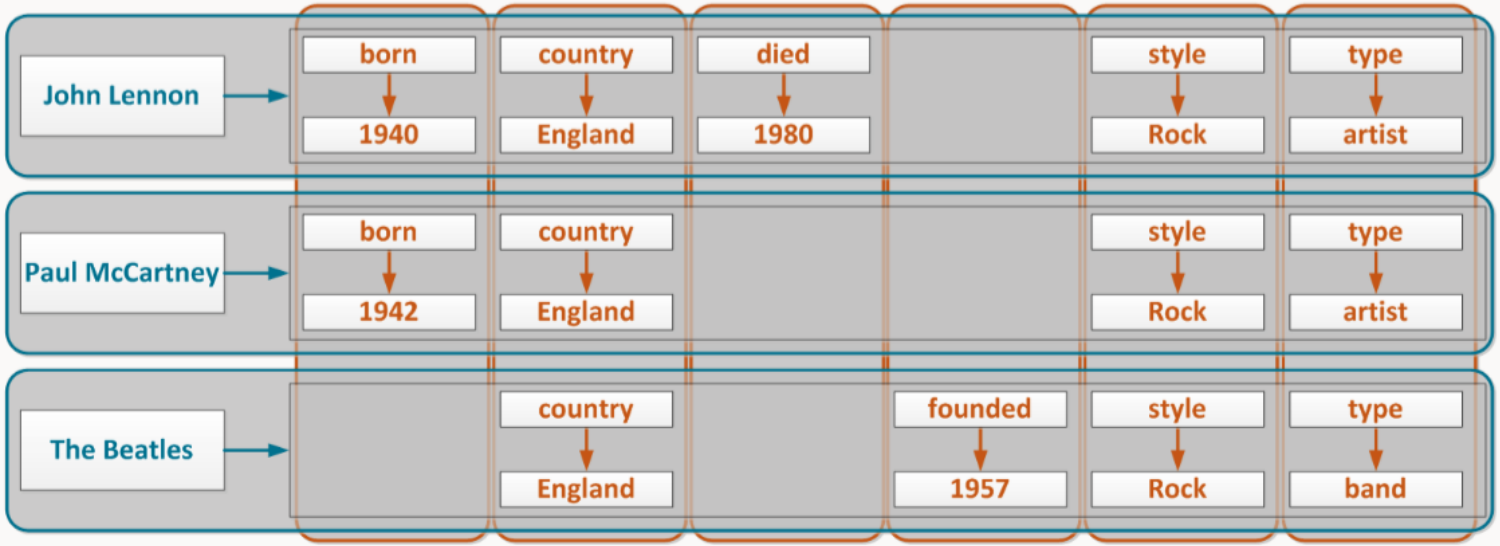
- Data are stored in a column-oriented way
  - Data efficiently stored
  - Avoids consuming space for storing nulls
  - Columns are grouped in column-families
  - Data isn’t stored as a single table but is stored by column families
  - Unit of data is a set of key/value pairs
  - Identified by “row-key”
  - Ordered and sorted based on row-key
- Notable for:
  - Google's Bigtable (used in all Google's services)
  - HBase (Facebook, Yahoo!, ...)

Column Store Comments

- Table with single-row partitions



- Column family view



Sorted Ordered Column-Oriented Stores

- More efficient than row (or document) store if:
  - Multiple row/record/documents are inserted at the same time so updates of column blocks can be aggregated
  - Retrievals access only some of the columns in a row/record/document
- Each storage block contains data from only one column
- Example:
  - Hadoop/Hbase
  - http://hadoop.apache.org/
  - Yahoo, Facebook

Sorted Ordered Column-Oriented Stores

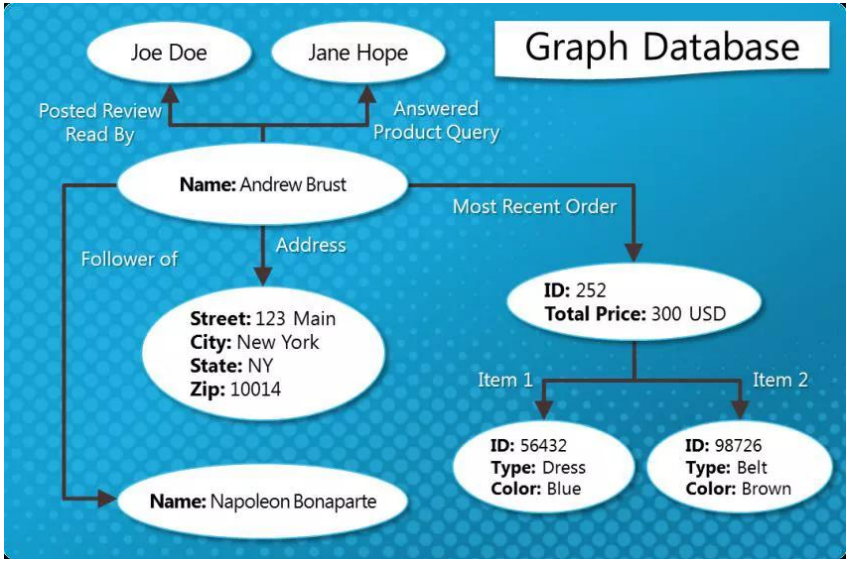
Advantages

- Scalability.
- Compression.
- Very responsive.

- Applications
  - Time-series data
  - Sensor data, analytics
  - Dashboards
  - High write throughput

Graph Databases

- Graph-oriented
- Everything is stored as an edge, a node or an attribute.
- Each node and edge can have any number of attributes.
- Both the nodes and edges can be labelled.
- Labels can be used to narrow searches.



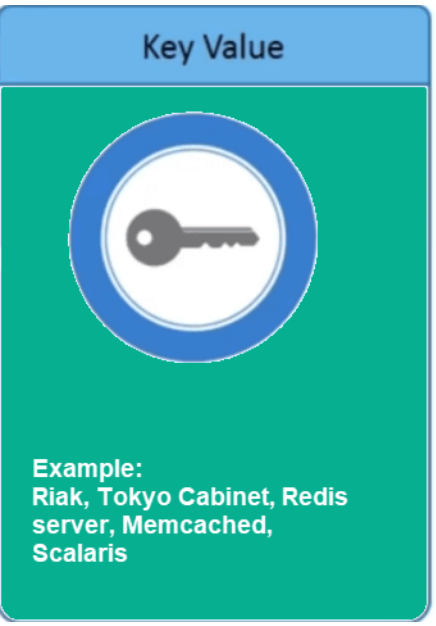
Graph Databases

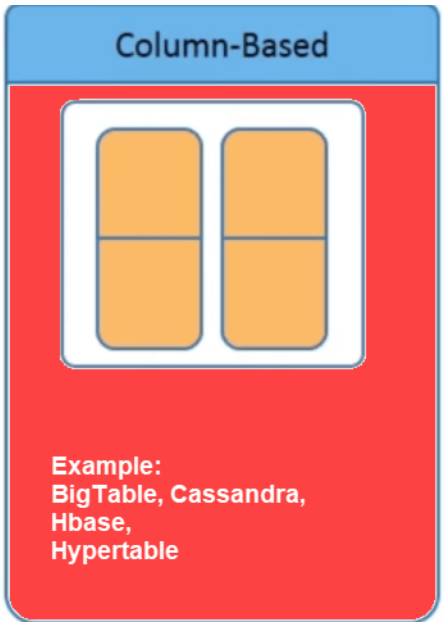
Best suited for:

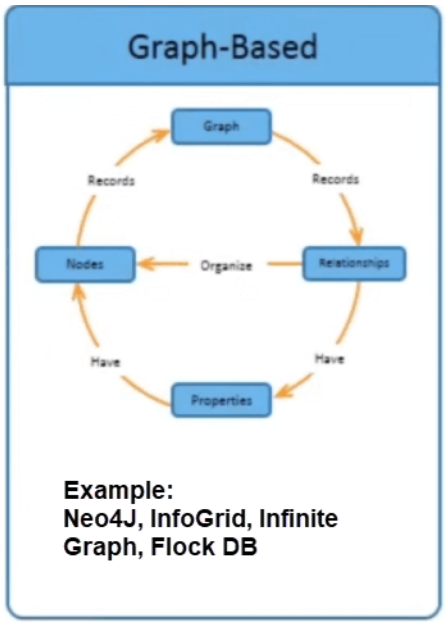
- Social networks
- Recommendation engines
- Fraud detection

Examples:

- Neo4j,
- Amazon Neptune,
- OrientDB







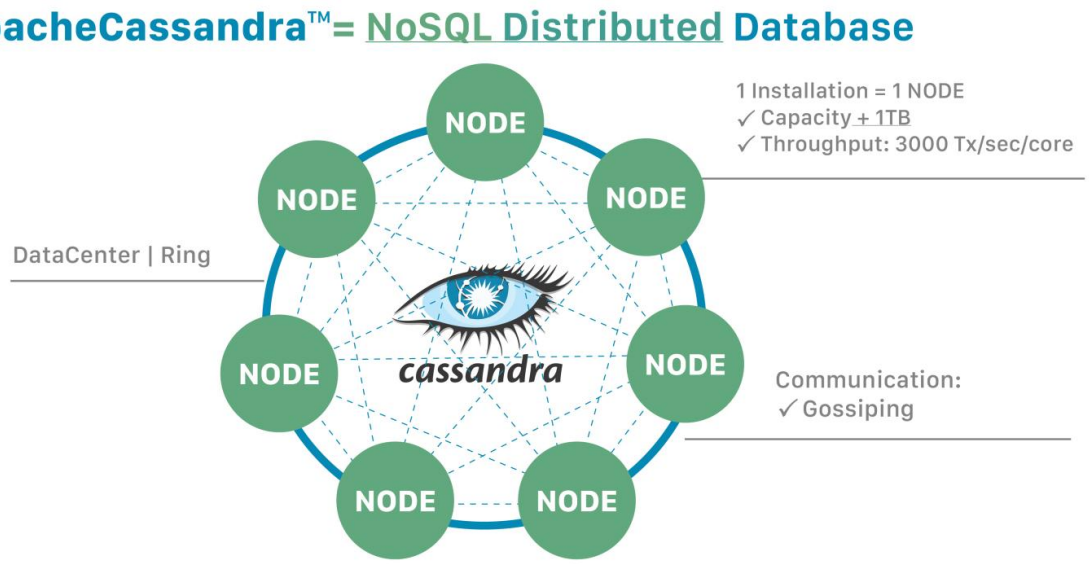
Cassandra - A Decentralized Structured Storage System

What is Apache Cassandra?

- Cassandra is a NoSQL distributed database.
- By design, NoSQL databases are lightweight, open-source, non-relational, and largely distributed.
- Counted among their strengths are horizontal scalability, distributed architectures, and a flexible approach to schema definition.
- Cassandra is among the NoSQL databases that have addressed the constraints of previous data management technologies, such as SQL databases.

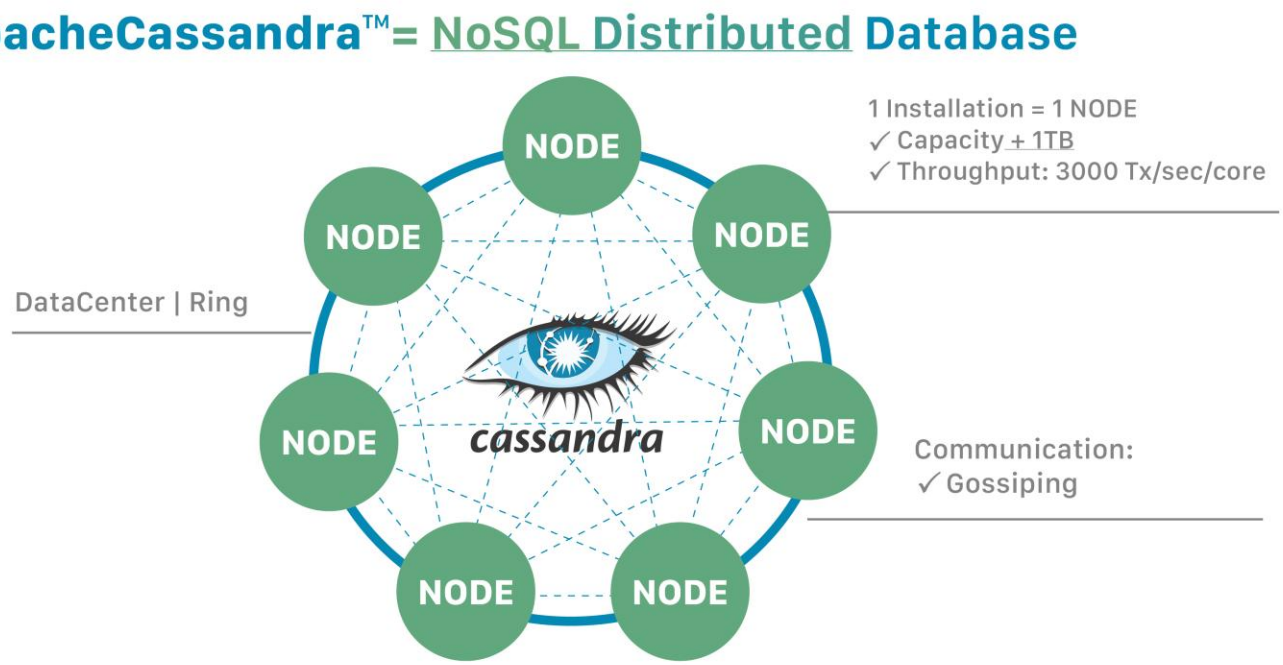
Outline

- Extension of Bigtable with aspects of Dynamo
- Motivations:
  - High Availability
  - High Write Throughput
  - Fail Tolerance



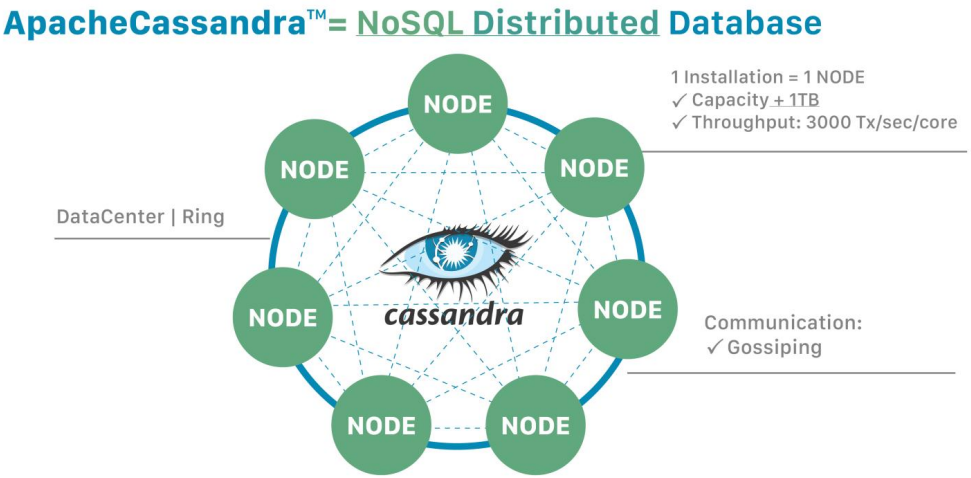
Distribution provides power and resilience

- "Distributed" means that Cassandra can run on multiple machines while appearing to users as a unified whole.
- Cassandsra can also run as a single node.
- But to get the maximum benefit out of Cassandra, you would run it on multiple machines.



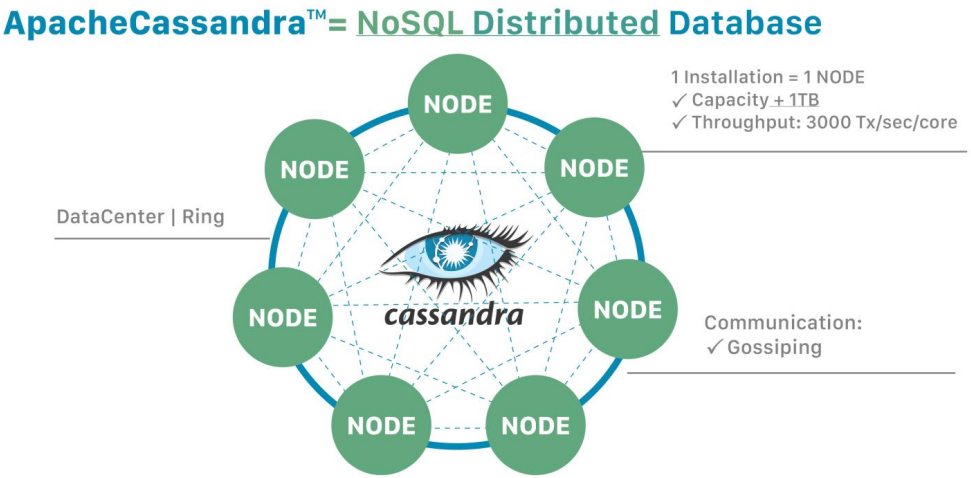
Distribution provides power and resilience

- Since it is a distributed database, Cassandra can (and usually does) have multiple nodes.
- A node represents a single instance of Cassandra.
- These nodes communicate with one another through a protocol called **gossip,** which is a process of computer peer-to-peer communication.



Distribution provides power and resilience

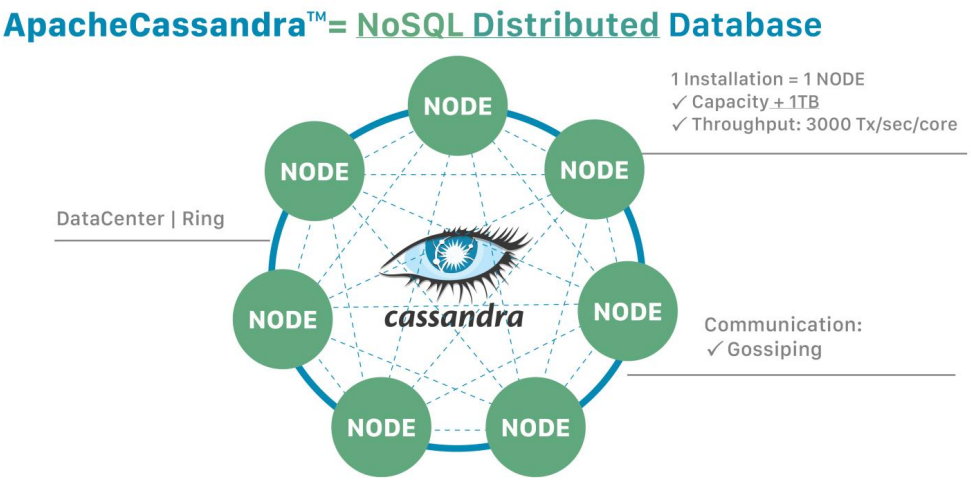
- Cassandra also has a masterless architecture – any node in the database can provide the exact same functionality as any other node – contributing to Cassandra's robustness and resilience.
- Multiple nodes can be organized logically into a cluster, or "ring".
- You can also have multiple datacenters.



Distribution provides power and resilience

- Cassandra databases easily scale when an application is under high stress, and the distribution also prevents data loss from any given datacenter's hardware failure.
- A distributed architecture also brings technical power; for example, a developer can tweak the throughput of read queries or write queries in isolation.

This means that Cassandra allows developers and administrators to tune performance parameters for reads and writes separately, based on the needs of the application.



Understanding Read and Write Consistency in Cassandra

- Cassandra is a **distributed NoSQL database** where data is **replicated** across multiple nodes.
- When you perform **read** or **write** operations, Cassandra gives you control over **how many replicas** must respond before it considers the operation successful.
- This control is set using the **consistency level (CL)**.
- To Achieve strong consistency in Cassandra
  - ***R + W > RF = strong consistency***
- In this equation,
  - R = Read replica count
  - W = Write replica count
  - RF = Replication factor
  - **Replication Factor (RF)** = number of copies of the data stored in the cluster.

Tunable consistency of Cassandra

Operation Type	Consistency Level Example	Description
Write	QUORUM, ALL, or ONE	Decide how many nodes must confirm a write before it's considered successful.
Read	LOCAL_QUORUM, ONE, TWO, etc.	Decide how many replicas must respond to a read request.

QUORUM means a **majority** of replica nodes must acknowledge the read or write operation before it is considered successful.

Mathematically:

$$\text{QUORUM} = \left\lfloor \frac{\textit{Replication Factor}}{2} \right\rfloor + 1$$

Where:

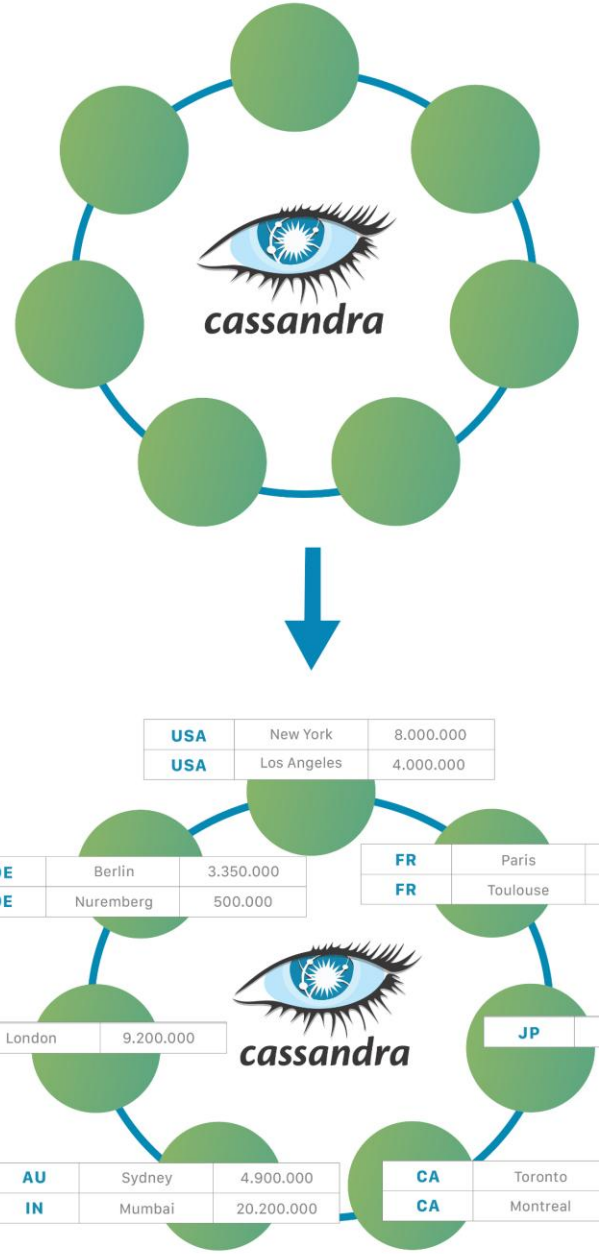
•**Replication Factor (RF)** = number of copies of the data stored in the cluster.

So if RF = 3, **Cassandra waits for responses from 2 replicas** to ensure consistency.

Replication Factor (RF)	QUORUM Value	Meaning
1	1	Only one node (trivial case)
2	2	Both replicas must respond
3	2	Any two replicas must respond
5	3	Any three replicas must respond

Want more power? Add more nodes

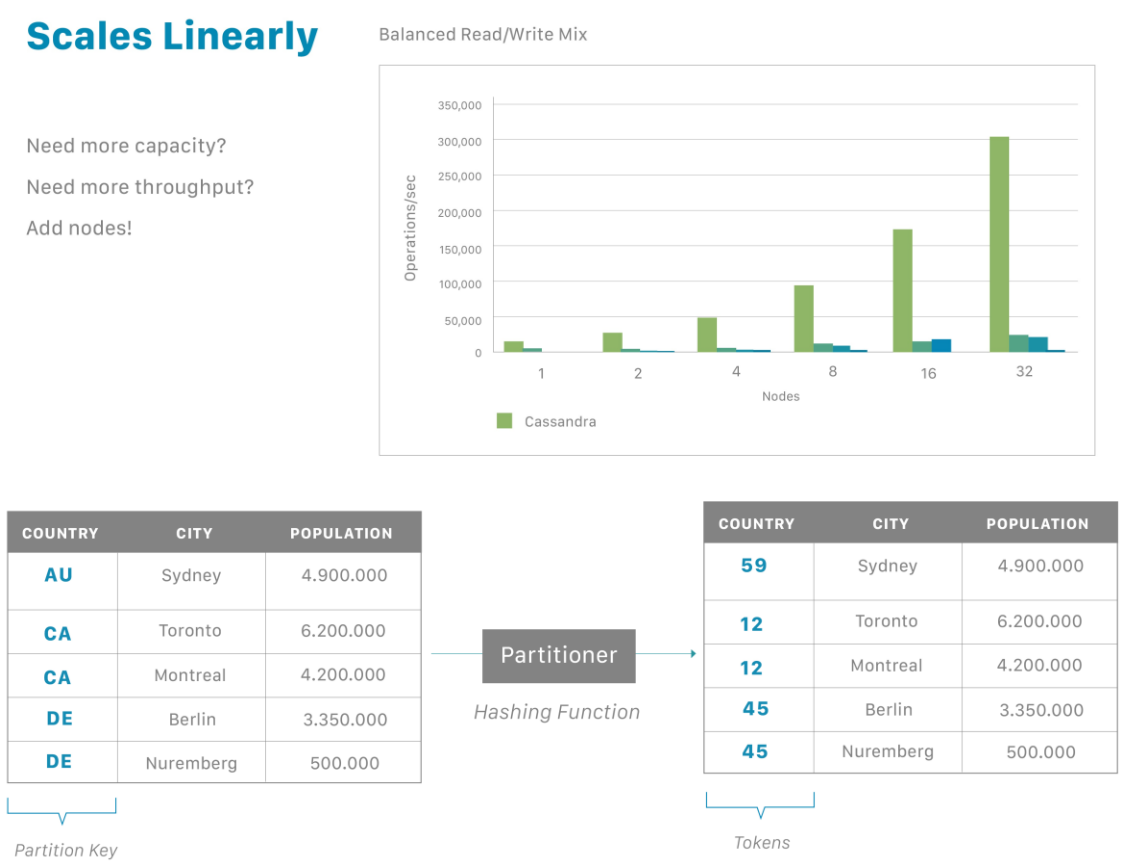
- One reason for Cassandra's popularity is that it enables developers to scale their databases dynamically, using off-the-shelf hardware, with no downtime.
- You can expand when you need to – and also shrink, if the application requirements suggest that path



COUNTRY	CITY	POPULATION
USA	New York	8.000.000
USA	Los Angeles	4.000.000
FR	Paris	2.230.000
DE	Berlin	3.350.000
UK	London	9.200.000
AU	Sydney	4.900.000
DE	Nuremberg	500.000
CA	Toronto	6.200.000
CA	Montreal	4.200.000
FR	Toulouse	1.100.000
JP	Tokyo	37.430.000
IN	Mumbai	20.200.000

Introducing partitions

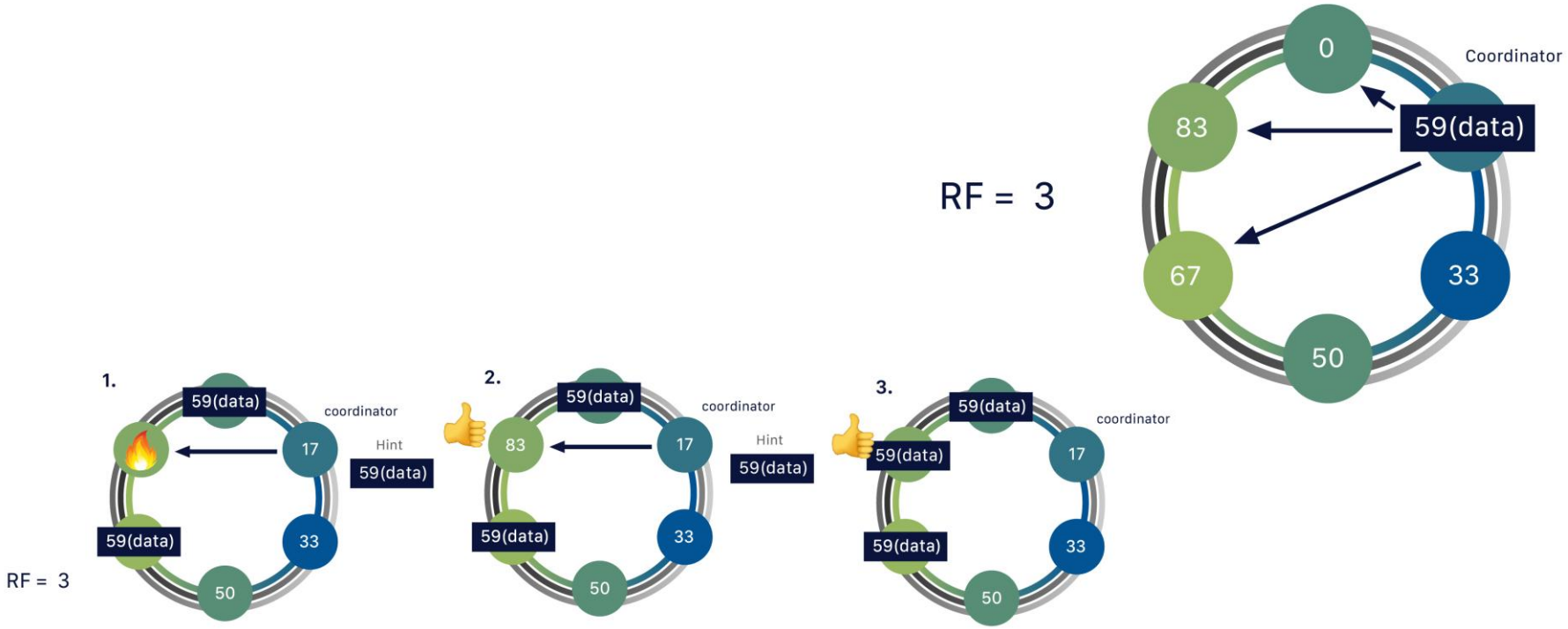
- The partition key is responsible for distributing data among nodes and is important for determining data locality.
- When data is inserted into the cluster, the first step is to apply a hash function to the partition key.
- The output is used to determine what node (based on the token range) will get the data.



The coordinator node isn't a single location; the system would be fragile if it were. It's simply the node that gets the request at that particular moment.
Any node can act as the coordinator.

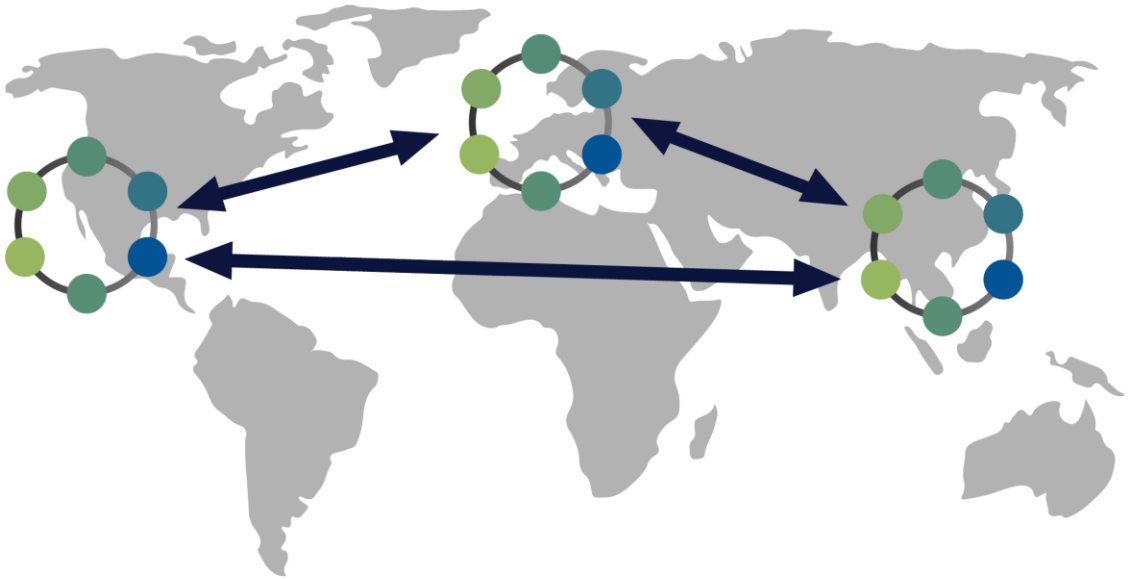
Replication ensures reliability and fault tolerance

- One piece of data can be replicated to multiple (replica) nodes, ensuring reliability and fault tolerance.
- Cassandra supports the notion of a replication factor (RF), which describes how many copies of your data should exist in the database.



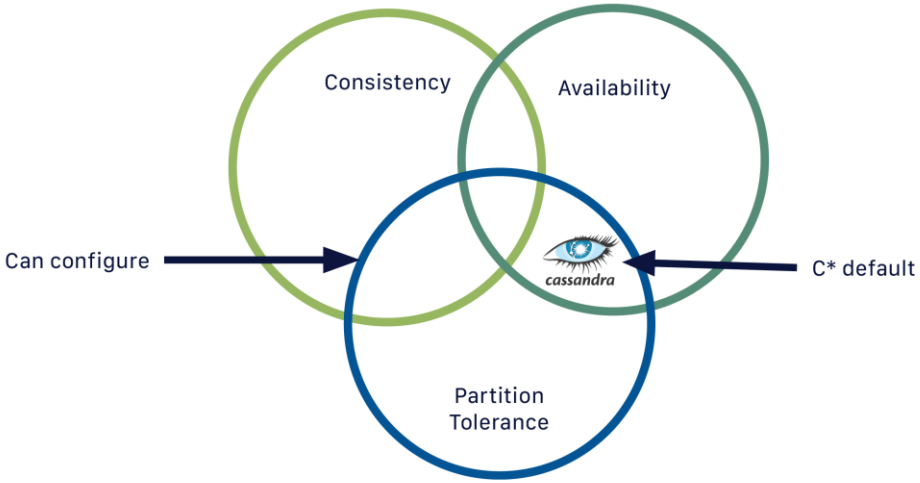
Replication ensures reliability and fault tolerance

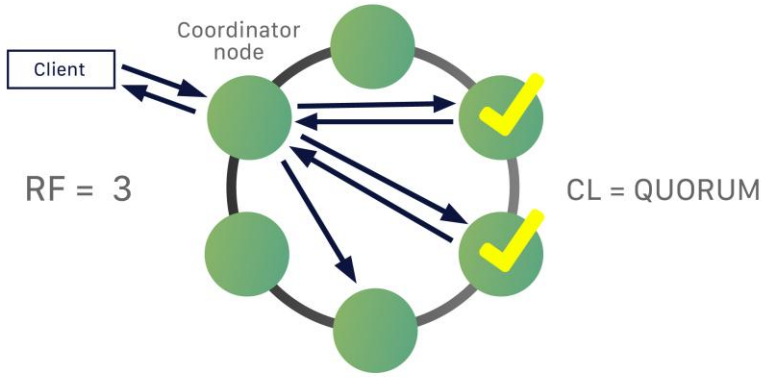
- Cassandra automatically replicates that data around your different data centers. Your application can write data to a Cassandra node on the U.S. west coast, and that data is automatically available in data centers at nodes in Asia and Europe.



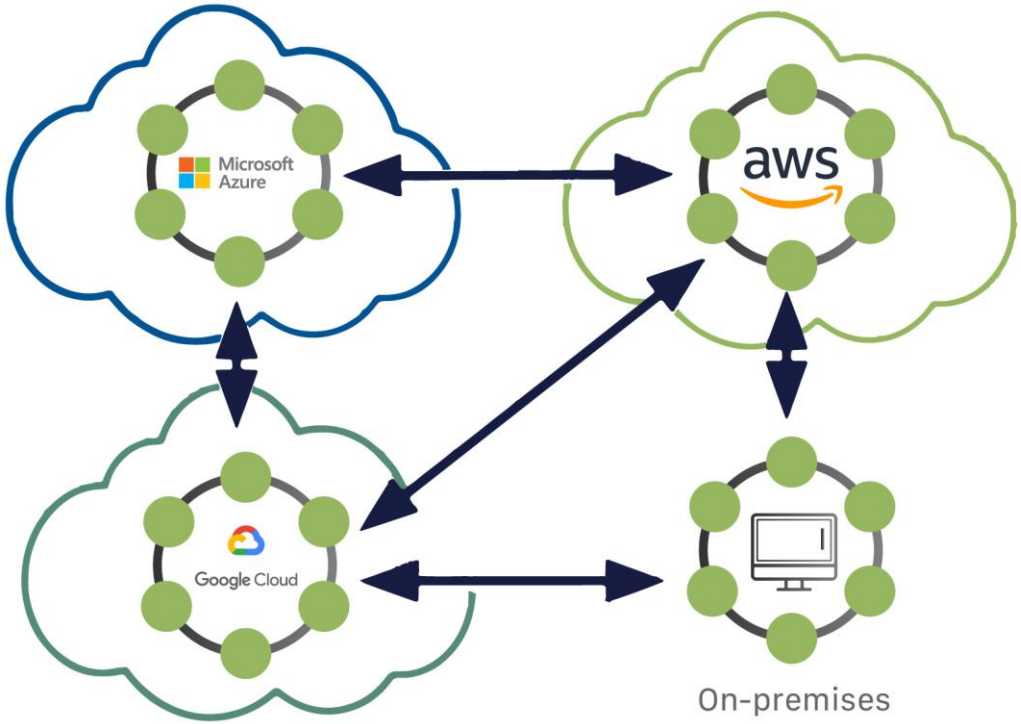
Tuning your consistency

- As per CAP theorem, Cassandra is by default an AP (Available Partition-tolerant) database, hence it is “always on”.
- But you can indeed configure the consistency on a per-query basis.



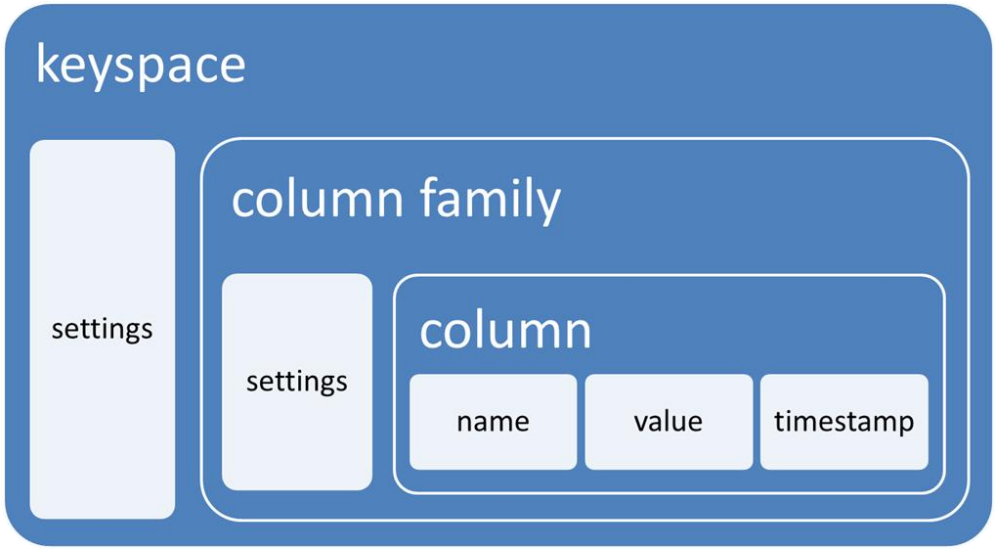


Cassandra is deployment agnostic.



Cassandra Data Model

- Table is a multi dimensional map indexed by key (row key).
- Columns are grouped into Column Families.
- 2 Types of Column Families
  - Simple
  - Super (nested Column Families)
- Each Column has
  - Name
  - Value
  - Timestamp



Cassandra System Architecture

- **Partitioning**

  How data is partitioned across nodes

- **Replication**

  How data is duplicated across nodes

- **Cluster Membership**

  How nodes are added, deleted to the cluster

Cassandra 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.

Cassandra Replication

- **Each data item is replicated at N (replication factor) nodes.**
- **Different Replication Policies**
  - **Rack Unaware –** replicate data at N-1 successive nodes after its coordinator
  - **Rack Aware –** uses ‘Zookeeper’ to choose a leader which tells nodes the range they are replicas for
  - **Datacenter Aware –** similar to Rack Aware but leader is chosen at Datacenter level instead of Rack level.

Gossip Protocols

- Network Communication protocols inspired for real life rumour 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.

References

- "NoSQL -- Your Ultimate Guide to the Non - Relational Universe!" http://nosql-database.org/links.html
- "NoSQL (RDBMS)" http://en.wikipedia.org/wiki/NoSQL
- PODC Keynote, July 19, 2000. *Towards Robust*. *Distributed Systems*. Dr. Eric A. *Brewer*. Professor, UC Berkeley. Co-Founder & Chief Scientist, Inktomi . *www.eecs.berkeley.edu/~**brewer**/cs262b-2004/PODC-keynote.pdf*
- "Brewer's CAP Theorem" posted by Julian Browne, January 11, 2009. http://www.julianbrowne.com/article/viewer/brewers-cap-theorem
- "How to write a CV" Geek & Poke Cartoon http://geekandpoke.typepad.com/geekandpoke/2011/01/nosql.html

References

- "Exploring CouchDB: A document-oriented database for Web applications", Joe Lennon, Software developer, Core International. http://www.ibm.com/developerworks/opensource/library/os-couchdb/index.html
- "Graph Databases, NOSQL and Neo4j" Posted by Peter Neubauer on May 12, 2010 at: http://www.infoq.com/articles/graph-nosql-neo4j
- "Cassandra vs MongoDB vs CouchDB vs Redis vs Riak vs HBase comparison", Kristóf Kovács. http://kkovacs.eu/cassandra-vs-mongodb-vs-couchdb-vs-redis
- "Distinguishing Two Major Types of Column-Stores" Posted by Daniel Abadi onMarch 29, 2010 http://dbmsmusings.blogspot.com/2010/03/distinguishing-two-major-types-of_29.html

References

- “MapReduce: Simplified Data Processing on Large Clusters”, Jeffrey Dean and Sanjay Ghemawat, December 2004. http://labs.google.com/papers/mapreduce.html
- “Scalable SQL”, ACM Queue, Michael Rys, April 19, 2011 http://queue.acm.org/detail.cfm?id=1971597
- “a practical guide to noSQL”, Posted by Denise Miura on March 17, 2011 at http://blogs.marklogic.com/2011/03/17/a-practical-guide-to-nosql/

Books

- "CouchDB *The Definitive Guide*", J. Chris Anderson, Jan Lehnardt and Noah Slater. O'Reilly Media Inc., Sebastopool, CA, USA. 2010
- "Hadoop *The Definitive Guide*", Tom White. O'Reilly Media Inc., Sebastopool, CA, USA. 2011
- "MongoDB *The Definitive Guide*", Kristina Chodorow and Michael Dirolf. O'Reilly Media Inc., Sebastopool, CA, USA. 2010

Thank you