Chapter 4
NoSQL

Bal Krishna Nyaupane
Assistant Professor
Department of Electronics and Computer Engineering
Institute of Engineering, Tribhuvan University
bkn@wrc.edu.np

Structured Data

- This is the data which is ***an organized form (i.e. in rows and columns)*** and can be easily used by computer program. Relationships exist between entities of data, such as classes and their objects.
- Structured data is highly organized and easily understood by machine language.
- Those working within relational databases can input, search, and manipulate structured data relatively quickly.
- ***Examples of structured data include*** names, dates, addresses, credit card numbers, stock information, geo-location, and more.
- ***Data stored in databases is an*** example of structured data.

Semi-structured Data

- Semi-structured data ***is a form of structured data*** that does not conform with the formal structure of data models associated with relational databases or other forms of data tables, ***but nonetheless contain tags or other markers to separate semantic elements*** and enforce hierarchies of records and fields within the data. Therefore, ***it is also known as self-describing structure.***
- ***Examples of semi-structured data include*** JSON and XML are forms of semi-structured data.
- The reason that this third category exists (between structured and unstructured data) is because ***semi-structured data is considerably easier to analyze than unstructured data.***
- Many ***Big Data solutions and tools have the ability*** to 'read' and process either JSON or XML.

Unstructured Data

- Unstructured data is information that ***either does not have a predefined data model*** or is ***not organized in a pre-defined manner***.
- Unstructured information is typically text-heavy, but may contain data such as dates, numbers, and facts as well.
- ***Common examples of unstructured data include*** text, video, audio, mobile activity, social media activity, satellite imagery, surveillance imagery, body of emails – the list goes on and on.
- Unstructured data ***is difficult to deconstruct*** because it has ***no pre-defined model***, meaning it cannot be organized in relational databases.
- Instead, non-relational, or NoSQL databases, are best fit for managing unstructured data.

Unstructured data types

Text files and documents

Server, website and application logs

Sensor data

Images

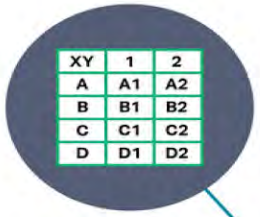
Video files

Audio files

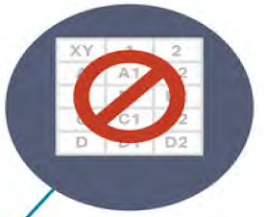
Emails

Social media data

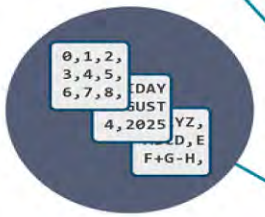
Structured Data vs Unstructured Data



Can be displayed in rows, columns and relational databases



Cannot be displayed in rows, columns and relational databases



Numbers, dates and strings

Images, audio, video, word processing files, e-mails, spreadsheets

Estimated 20% of enterprise data *(Gartner)*

Estimated 80% of enterprise data *(Gartner)*

Requires less storage

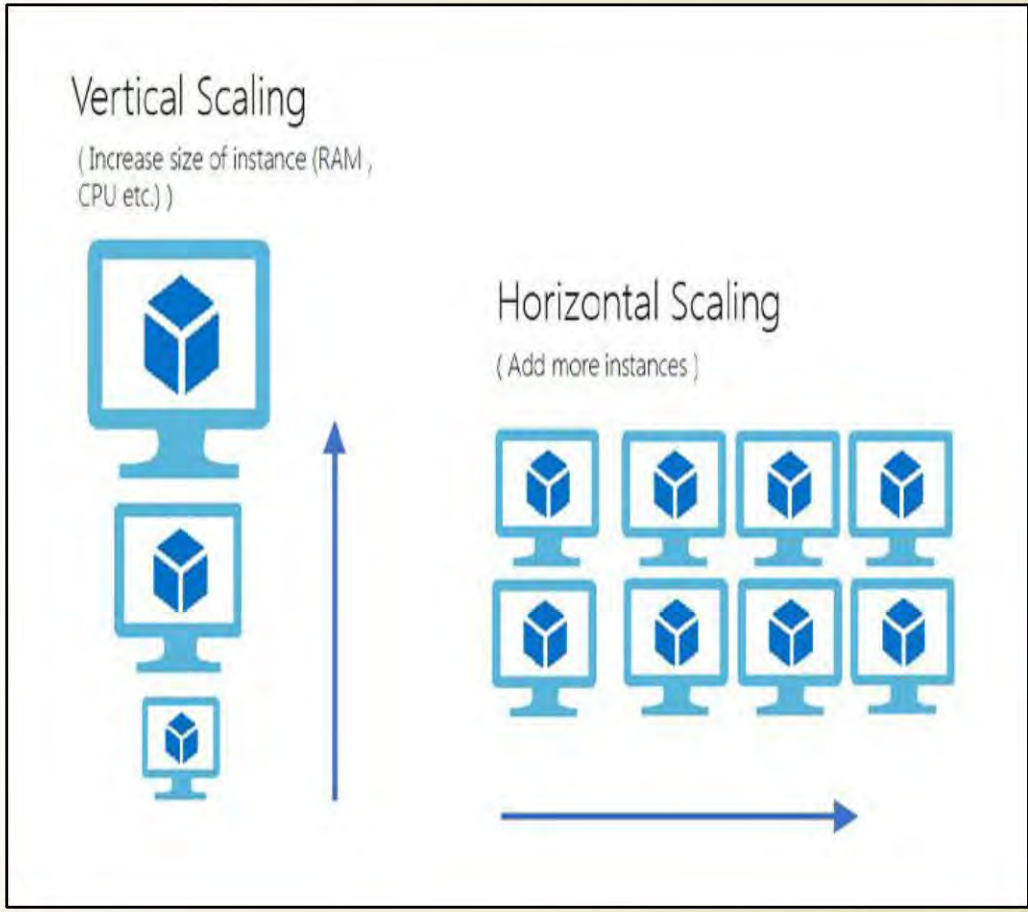
Requires more storage

Easier to manage and protect with legacy solutions

More difficult to manage and protect with legacy solutions

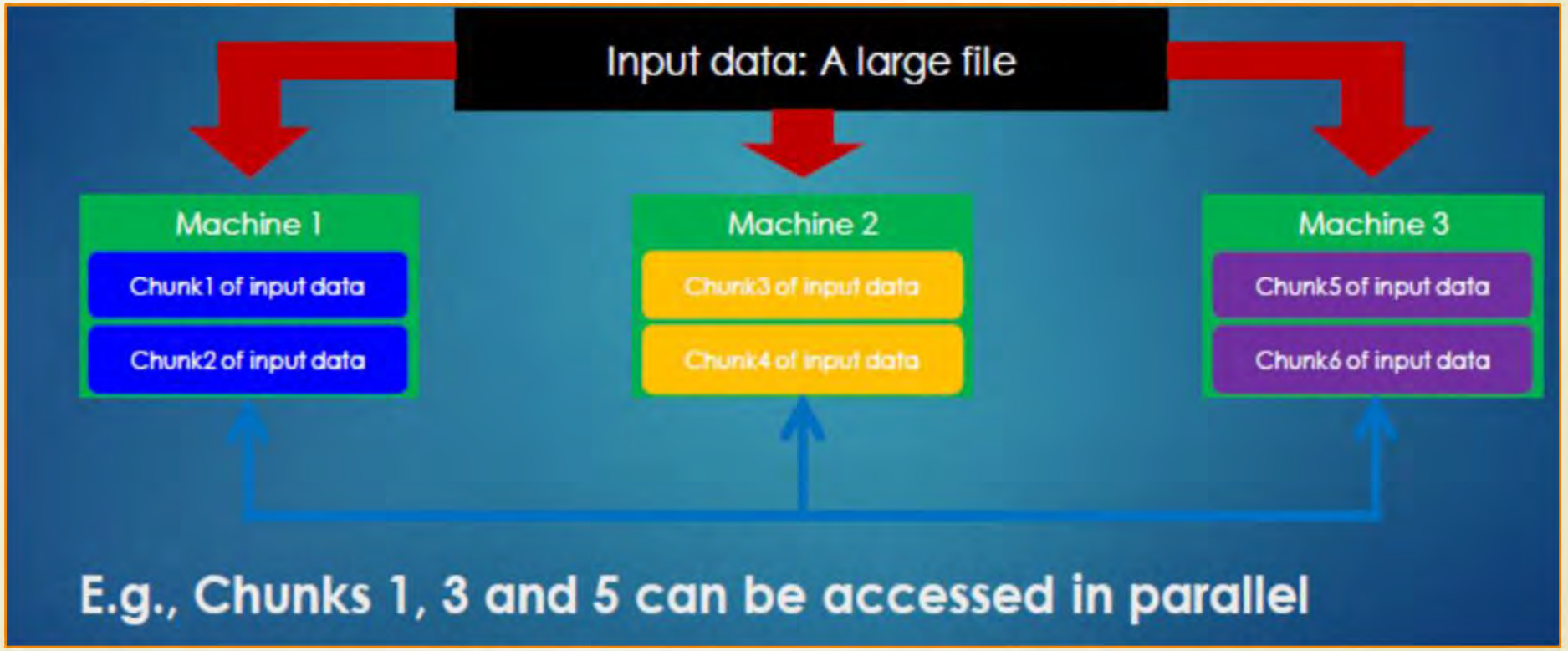
Scaling traditional database

- Traditional RDBMSs can be either scaled:
- ***Vertically (or Up)***
  - Can be achieved by hardware upgrades (e.g., faster CPU, more memory, or larger disk)
  - Limited by the amount of CPU, RAM and disk that can be configured on a single machine
- ***Horizontally (or Out)***
  - Can be achieved by adding more machines
  - Requires database sharding and probably replication
  - Limited by the Read-to-Write ratio and communication overhead



Why Sharding Data?

- Sharding is a ***type of database partitioning*** that separates ***very large databases the into smaller, faster, more easily managed parts*** called data shards. ***The word shard means a small part of a whole.***
- Data is typically sharded (or striped) to allow for concurrent/parallel accesses.



Why Replicating Data?

- Replicating data across servers helps in:
  - ***Avoiding single point of failures***
  - ***It creases the reliability of a system.***
    - If one replica is unavailable or crashes, use another
    - Protect against corrupted data
  - ***It improves the performance of a system.***
    - Scale with size of the distributed system (replicated Web servers)
    - Scale in geographically distributed systems (Web proxies)
- The ***key issue*** is the need to maintain ***consistency of replicated*** data.
  - If one copy is modified, others become inconsistent.

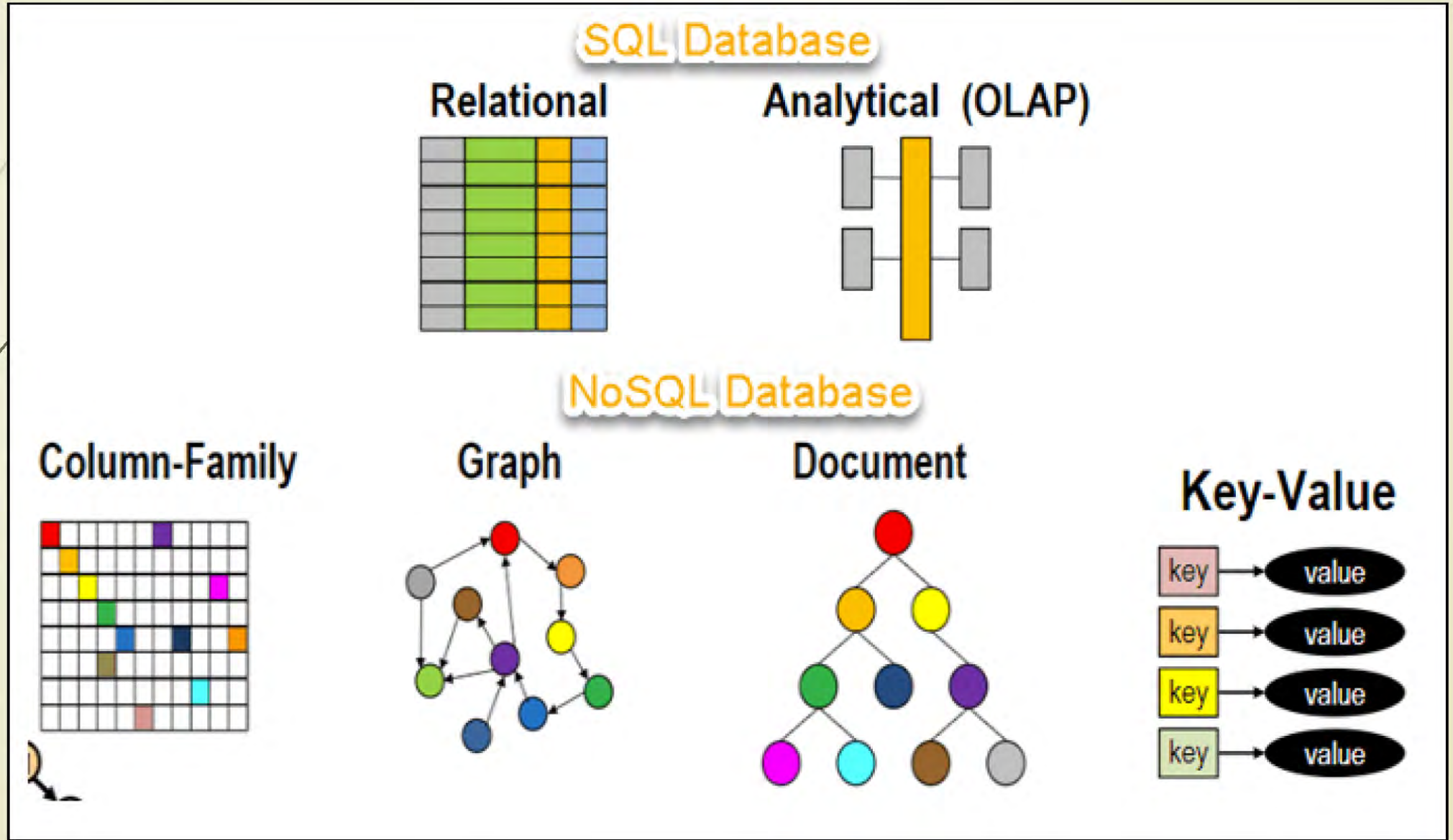
What is NoSQL?

- NoSQL database stands for ***"Not Only SQL" or "Not SQL***." Though a better term would NoREL NoSQL caught on. Carl Strozz introduced ***the NoSQL concept in 1998.***
- NoSQL is a generic term used to refer to any data store that ***does not follow the traditional RDBMS model***—specifically, ***the data is non-relational*** and ***it does not use SQL as the query language.***
- NoSQL is a non-relational DMS, that ***does not require a fixed schema, avoids joins, and is easy to scale.***
- Traditional RDBMS uses SQL syntax to store and retrieve data for further insights. Instead, ***a NoSQL database system encompasses a wide range of database technologies that can store*** structured, semi-structured, unstructured and polymorphic data.

What is NoSQL?

- There is no full agreement but nowadays we can ***summarize NoSQL definition as follows***
  - *Next generation databases addressing some of the points:*
  - *non relational*
  - *schema-free*
  - *no Join*
  - *Distributed*
  - *horizontally scalable with easy replication support*
  - *eventually consistent*
  - *open source*

NoSQL Database



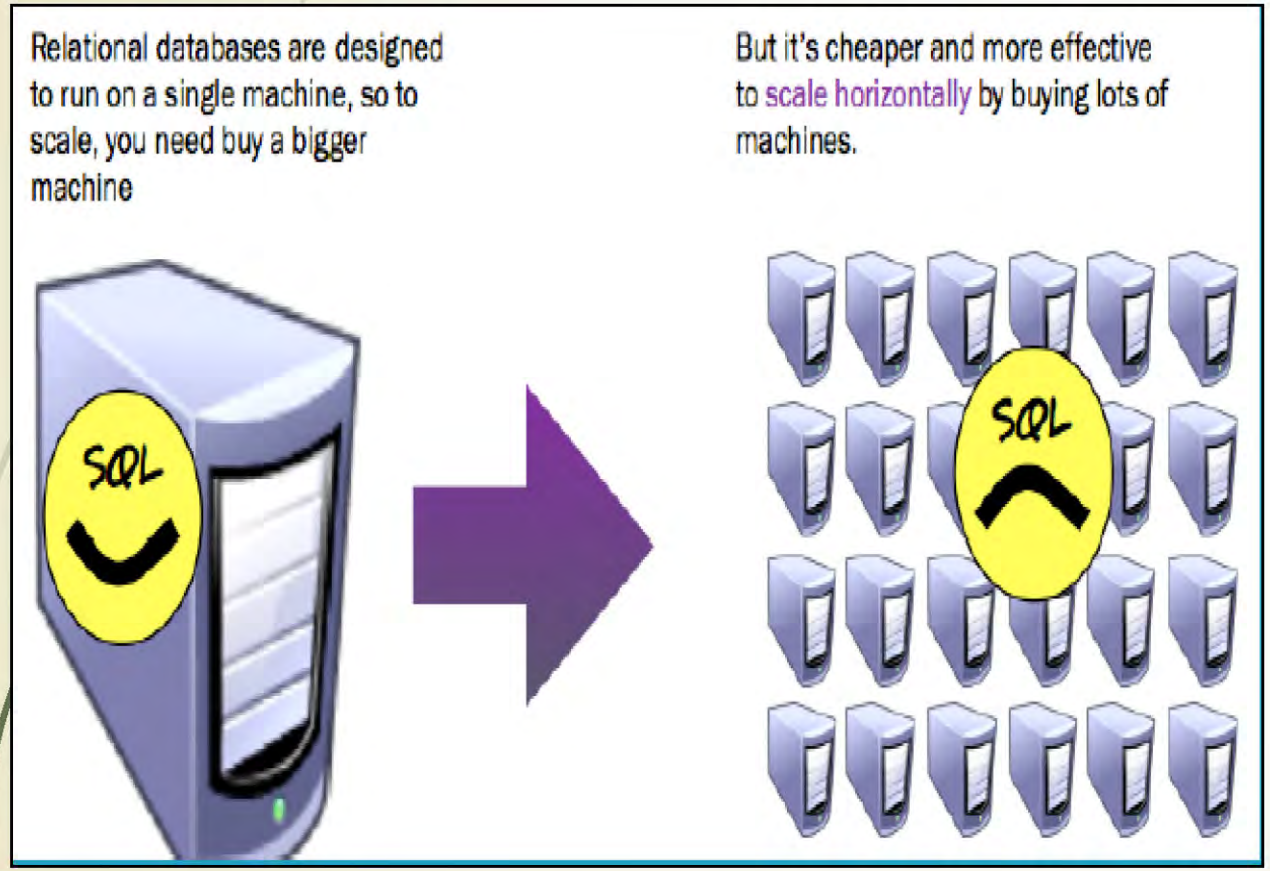
Why NoSQL?

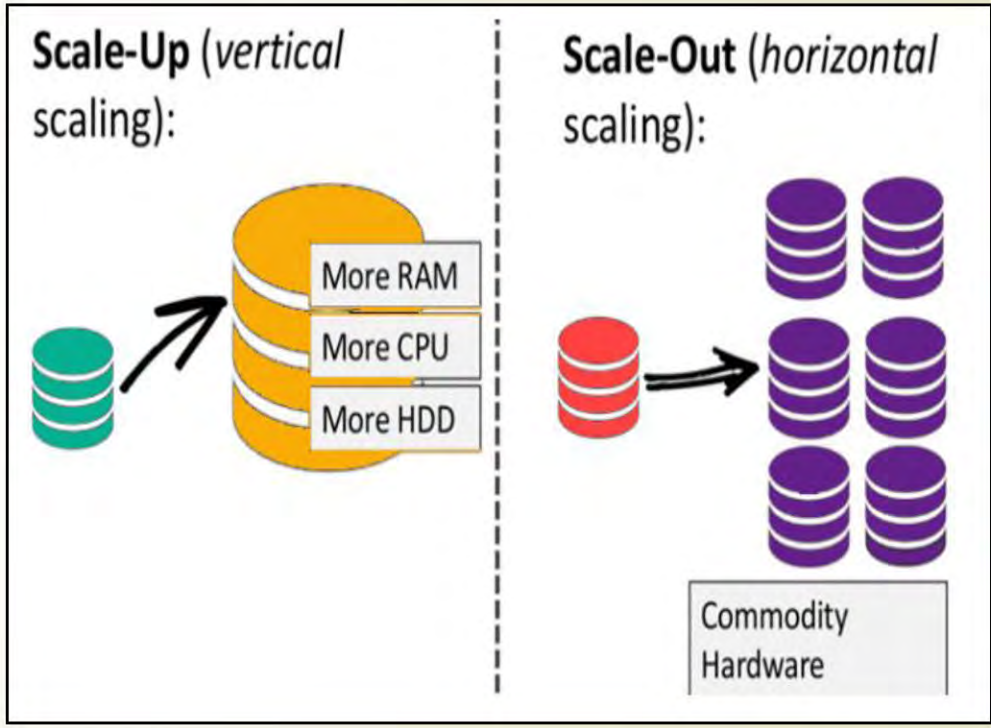
- ***NoSQL is used for Big data and real-time web apps***. For example companies like Twitter, Facebook, Google that collect terabytes of user data every single day. The listed companies didn't start off by rejecting relational technologies. ***They tried them and found that they didn't meet their requirements:***
  - *Huge concurrent transactions volume*
  - *Expectations of low-latency access to massive datasets*
  - *Expectations of nearly perfect service availability while operating in an unreliable environment*
- ***They tried the traditional approach***
  - *Adding more HW*
  - *Upgrading to faster HW as available*

Why NoSQL?

- ***Schema less data representation:*** This means that you don't have to think too far ahead to define a structure and you can continue to evolve over time—including adding new fields or even nesting the data.
- ***Development time:*** It reduced development time because one doesn't have to deal with complex SQL queries.
- ***Speed:*** Even with the small amount of data that you have, if you can deliver in milliseconds rather than hundreds of milliseconds — especially over mobile and other intermittently connected devices.
- ***Plan ahead for scalability:*** your application can be quite elastic — it can handle sudden spikes of load. Of course, you win users over straightaway.

Why NoSQL?


Relational databases are designed to run on a single machine, so to scale, you need buy a bigger machine
But it's cheaper and more effective to scale horizontally by buying lots of machines.
SQL
SQL


Scale-Up (vertical scaling):
More RAM
More CPU
More HDD
Scale-Out (horizontal scaling):
Commodity Hardware

Characteristics of NoSQL

- ***Non-relational***
  - Never provide tables with flat fixed-column records
  - Doesn't require object-relational mapping and data normalization
  - No complex features like query languages, referential integrity, joins, and ACID
- ***Schema-free***
  - Offers heterogeneous structures of data in the same domain
- ***Distributed***
  - Multiple NoSQL databases can be executed in a distributed fashion
  - Offers auto-scaling and fail-over capabilities
  - Mostly no synchronous replication between distributed nodes.
  - Only providing eventual consistency
  - Shared Nothing Architecture. This enables less coordination and higher distribution.

Advantages of NoSQL

- ***Big Data Capability***
- No Single Point of Failure
- ***Easy Replication***
- It provides fast performance and horizontal scalability.
- Handle ***structured, semi-structured, and unstructured*** data with equal effect
- NoSQL ***databases don't need a dedicated high-performance server***
- Simple to implement than using RDBMS
- It ***can serve as the primary data source for online applications***.
- Handles big data ***which manages data velocity, variety, volume, and complexity***
- Excels at distributed database and multi-data center operations
- Offers a flexible schema design which can easily be altered without downtime or service disruption.

Disadvantages of NoSQL

- ***Narrow focus:*** NoSQL databases have very narrow focus as it is mainly designed for storage but it provides very little functionality. Relational databases are a better choice in the field of Transaction Management than NoSQL.
- ***Open-source:*** NoSQL is open-source database. There is no reliable standard for NoSQL yet. In other words two database systems are likely to be unequal.
- ***GUI is not available:*** GUI mode tools to access the database is not flexibly available in the market.
- ***Doesn't work as well with relational data***
- When the volume of data increases it is difficult to maintain unique values as keys become difficult

RDMS Vs. NoSQL

NoSQL	RDMS
• *No Standard Query Language. Each database has owns type of query syntax.*	• *Has SQL.*
• *Non-Relational*	• *Relational*
• *Schema free*	• *Based on Schema*
• *Provides fast performance and horizontal scalability*	• *Vertical Scaling*
• *Can handle structured, semi-structured, and unstructured data with equal effect.*	• *Handle only Structured Data.*
• *NoSQL databases don't need a dedicated high performance server.*	• *Need a dedicated high performance server.*
• *Database administrators are not required.*	• *DBA are required.*
• *Based on BASE and CAP Theorem.*	• Based on ACID properties.

Where NoSQL Is Used?

- Google (BigTable, LevelDB)
- LinkedIn (Voldemort)
- Facebook (Cassandra)
- Twitter (Hadoop/Hbase, FlockDB, Cassandra)
- Netflix (SimpleDB, Hadoop/HBase, Cassandra)
- CERN (CouchDB)

The CAP Theorem

- ***Eric Brewer***, at the University of California, presented the theory in the fall of 1998, and ***it was published in 1999 as the CAP Principle***.
- A distributed system ***can satisfy any two of these guarantees at the same time***, but not all three. ***The three guarantees that cannot be met simultaneously are:***
  - ***Consistency:*** The data within ***the database remains consistent***, even after an operation has been executed. This means once data is written, any future read request should contain that data. For example, after updating the order status, all the clients should be able to see the same data.
  - ***Availability:*** The database ***should always be available and responsive***. It should not have any downtime.

The CAP Theorem

- ***Partition Tolerance:*** Partition Tolerance means that the system ***should continue to function even if the communication among the servers is not stable***. For example, the servers can be ***partitioned into multiple groups*** which may not communicate with each other. Here, ***if part of the database is unavailable***, other parts are always unaffected.
- ***In 2002,*** a formal proof of Brewer's concept was ***published by Nancy Lynch and Seth Gilbert of MIT***, turning it into a "true theorem."

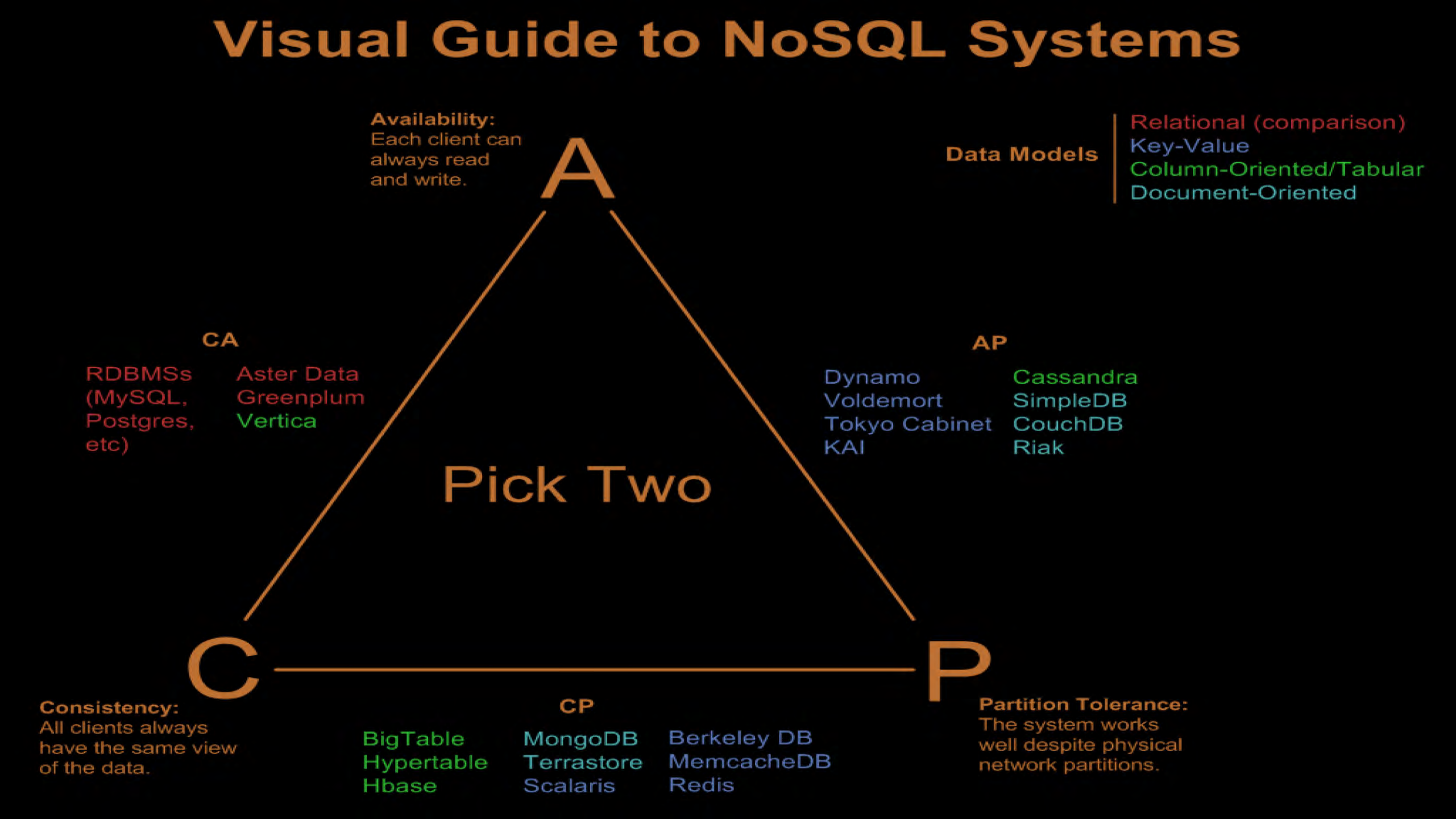
The CAP Theorem

- In other words, ***CAP can be expressed as*** "***If the network is broken, your database won't work***"
  - "won't work" = down OR inconsistent
- In RDBMS ***we do not have P*** (network partitions)
  - Consistency and Availability are achieved
- **In NoSQL we want to *have P***
  - Need to ***select either C or A***
  - ***Drop A*** -> Accept waiting until data is consistent
  - ***Drop C*** -> Accept getting inconsistent data sometimes

The CAP Theorem

- A ***consistency model determines rules for visibility*** and ***apparent order of updates***. For example:
  - Row X is replicated on nodes M and N
  - Client A writes row X to node N
  - Some period of time t elapses.
  - Client B reads row X from node M
  - ***Does client B see the write from client A?***
  - Consistency is a continuum with tradeoffs
  - ***For NoSQL***, ***the answer would be: maybe***
- ***CAP Theorem states:*** Strict ***Consistency can't be achieved*** at the same time as availability and partition-tolerance.

Visual Guide to NoSQL Systems



Eventual Consistency

- Although ***applications must deal with instantaneous consistency***, NoSQL systems ensure that ***at some future point in time the data assumes a consistent state.*** In contrast to ACID systems that ***enforce consistency at transaction commit, NoSQL guarantees consistency only at some undefined future time.***
- The term "***eventual consistency***" means to have ***copies of data on multiple machines*** to get high availability and scalability. Thus, ***changes made to any data item*** on one machine has to ***be propagated to other replicas***.
- ***Data replication may not be instantaneous*** as some copies will be updated immediately ***while others in due course of time***. These copies may be mutually, but in due course of time, ***they become consistent***. Hence***, the name eventual consistency.***
- ***Known as BASE (Basically Available, Soft state, Eventual consistency), as opposed to ACID.***

BASE

- ***Basic availability***
  - ***Each request is guaranteed a response*** - successful or failed execution. It means ***DB is available all the time as per CAP theorem***.
  - The availability ***of BASE is achieved through supporting partial failures*** without total system failure.
- ***Soft state***
  - Soft state indicates that the ***state of the system may change over time***, even without input (for eventual consistency).
- ***Eventual consistency***
  - The database may ***be momentarily inconsistent*** but will ***be consistent eventually.***

NoSQL Databases

- The NoSQL databases are ***categorized on the basis of how the data is stored.***
- Because of the need to provide collected information from large volumes, generally in near real-time***, NoSQL mostly follows a horizontal structure***. They are ***optimized for insert and retrieve operations*** on a large scale with built-in capabilities for replication and clustering.
- ***There are Four basic types of NoSQL databases:***
  1. ***Column-oriented***
  2. ***Document Store***
  3. ***Key Value Store***
  4. ***Graph***

NoSQL Databases

Document	Key-Value	XML	Column	Graph
MongoDB	Redis	BaseX	BigTable	Neo4J
CouchDB	Membase	eXist	Hadoop / HBase	FlockDB
RavenDB	Voldemort		Cassandra	InfiniteGraph
Terrastore	MemcacheDB		SimpleDB	
			Cloudera	

Column-oriented databases

- In a typical column-oriented store, you ***predefine a column-family***.
- A ***column-family is a set of columns grouped together*** into a bundle. Columns in ***a column-family are logically related to each other***, although this not a requirement.
- Column-family members ***are physically stored together*** and typically a user benefits by ***clubbing together columns with similar access characteristics*** into the same family.
- In a column database, ***a column-family is analogous to a column in an RDBMS***. Both are typically defined before data is stored in tables and are fairly static in nature.
- ***Columns in RDBMS define the type of data they can store***. Column-families have no such limitation; they can contain any number of columns, which can store any type of data, ***as far as they can be persisted as an array of bytes***.

Column-oriented databases

- ***Each row of a column-oriented*** database table stores data values in only those ***columns for which it has valid values***.
- Column databases are designed to scale and can easily accommodate millions of columns and billions of rows. ***Therefore, a single table often spans multiple machines.*** A row-key uniquely identifies a row in a column database. Rows are ordered and split into bundles, containing contiguous values, as data grows.
- In comparison, most relational DBMS store data in rows, ***the benefit of storing data in columns, is fast search/ access and data aggregation.***
- Relational databases ***store a single row as a continuous disk entry***. Different rows are stored in different places on disk ***while Columnar databases store all the cells corresponding to a column as a continuous disk entry*** thus makes the search/access faster.

Column-oriented databases

Emp_no	Dept_id	Hire_date	Emp_ln	Emp_fn
1	1	2001-01-01	Smith	Bob
2	1	2002-02-01	Jones	Jim
3	1	2002-05-01	Young	Sue
4	2	2003-02-01	Stemle	Bill
5	2	1999-06-15	Aurora	Jack
6	3	2000-08-15	Jung	Laura

Row-Oriented Database

1	1	2001-01-01	Smith	Bob
2	1	2002-02-01	Jones	Jim
3	1	2002-05-01	Young	Sue

Column-Oriented Database

1	2	3	4	5
1	1	1	2	2
2001-01-01	2002-02-01	2002-02-01	2002-02-01	2002-02-01

Column-oriented databases

Row-oriented

ID	Name	Grade	GPA
001	John	Senior	4.00
002	Karen	Freshman	3.67
003	Bill	Junior	3.33

Column-oriented

Name	ID
John	001
Karen	002
Bill	003

Grade	ID
Senior	001
Freshman	002
Junior	003

GPA	ID
4.00	001
3.67	002
3.33	003

Column-oriented databases

- For example: To query ***the titles from a bunch of a million articles*** will be a ***painstaking task*** while using relational databases as it will go over each location to get item titles.
  - On the other hand, with just one disk access, title of all the items can be obtained.
- ***Some of the databases*** that fall under this category include:
  - Oracle RDBMS Columnar Expression
  - Microsoft SQL Server 2012 Enterprise Edition
  - ***Apache Cassandra***
  - ***Hbase***
  - Google Big Table

Advantages of Column-oriented databases

- ***Computing maxima, minima, averages and sums***, specifically on large datasets, is where these column-oriented data stores outshine in performance.
- ***When new values are applied for either all rows at once*** or with same column filters, these databases will allow partial data access ***without touching unrelated columns*** and be ***much faster in execution***.
- Since columns will be of uniform type and mostly (except in cases of variable-length strings) of the same length, ***there are possibilities of efficient storage in terms of size***. Such as a column with the same values across rows (for example, the department of a user profile or whether a user's profile is public or private or even a user's age), ***the same or similar adjacent values can be compressed efficiently.***

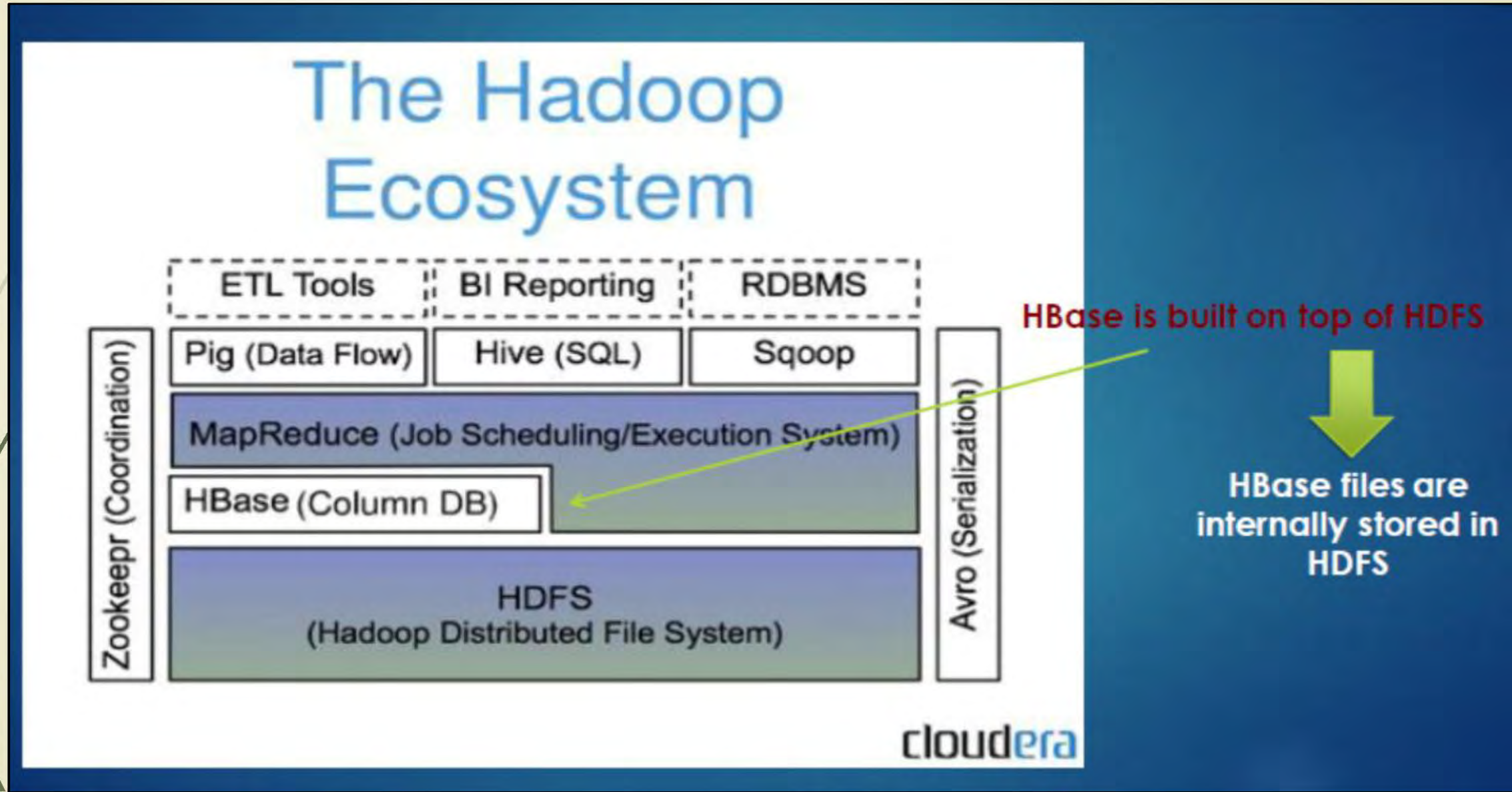
HBase

- HBase is a distributed ***column-oriented database*** built on top of the ***Hadoop file system***. It is an open-source project and is horizontally scalable.
- HBase is a data model that ***is similar to Google's big table*** designed to provide quick random access to huge amounts of structured data. It leverages the ***fault tolerance provided by the Hadoop File System*** (HDFS).
- It is a ***part of the Hadoop ecosystem*** that provides random real-time read/write ***access to data in the Hadoop File System***.
- One can store the ***data in HDFS either directly or through HBase***. Data consumer reads/accesses the data in HDFS randomly using HBase. HBase sits on top of the Hadoop File System ***and provides read and write access.***

HBase

- ***Features***
  - HBase is linearly scalable.
  - It has automatic failure support.
  - It provides consistent read and writes.
  - It integrates with Hadoop, both as a source and a destination.
  - It has easy java API for client.
  - It provides data replication across clusters.
- ***Uses***
  - It is used whenever there is a need to write heavy applications.
  - HBase is used whenever we need to provide fast random access to available data.
  - Companies such as Facebook, Twitter, Yahoo, and Adobe use HBase internally.

HBase: Part of Hadoop's Ecosystem



HBase Vs. HDFS

HDFS	HBase
HDFS is a distributed file system suitable for storing large files.	HBase is a database built on top of the HDFS.
HDFS does not support fast individual record lookups.	HBase provides fast lookups for larger tables.
It provides high latency batch processing; no concept of batch processing.	It provides low latency access to single rows from billions of records (Random access).
It provides only sequential access of data.	HBase internally uses Hash tables and provides random access, and it stores the data in indexed HDFS files for faster lookups.

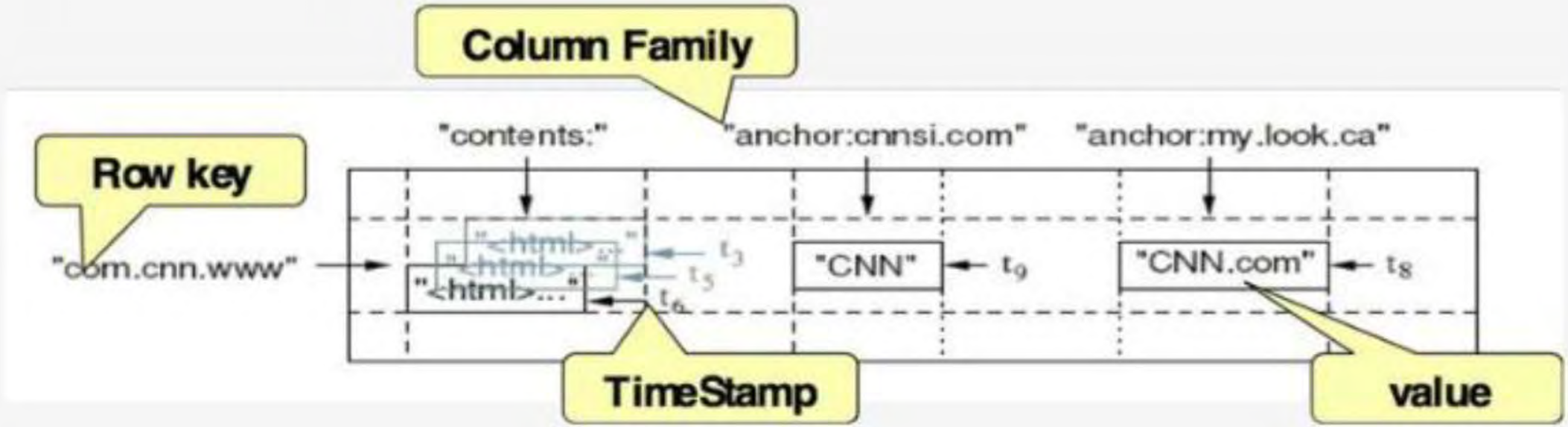
HBase Data Model

- HBase is a column-oriented database and the tables in it are sorted by row.
- The ***table schema defines only column families***, which are the key value pairs. A table have multiple column families and each ***column family can have any number of columns***.
- Subsequent ***column values are stored contiguously on the disk***. Each cell value of the table has a timestamp. In short, in an HBase:
  - Table is a collection of rows.
  - Row is a collection of column families.
  - Column family is a collection of columns.
  - Column is a collection of key value pairs.

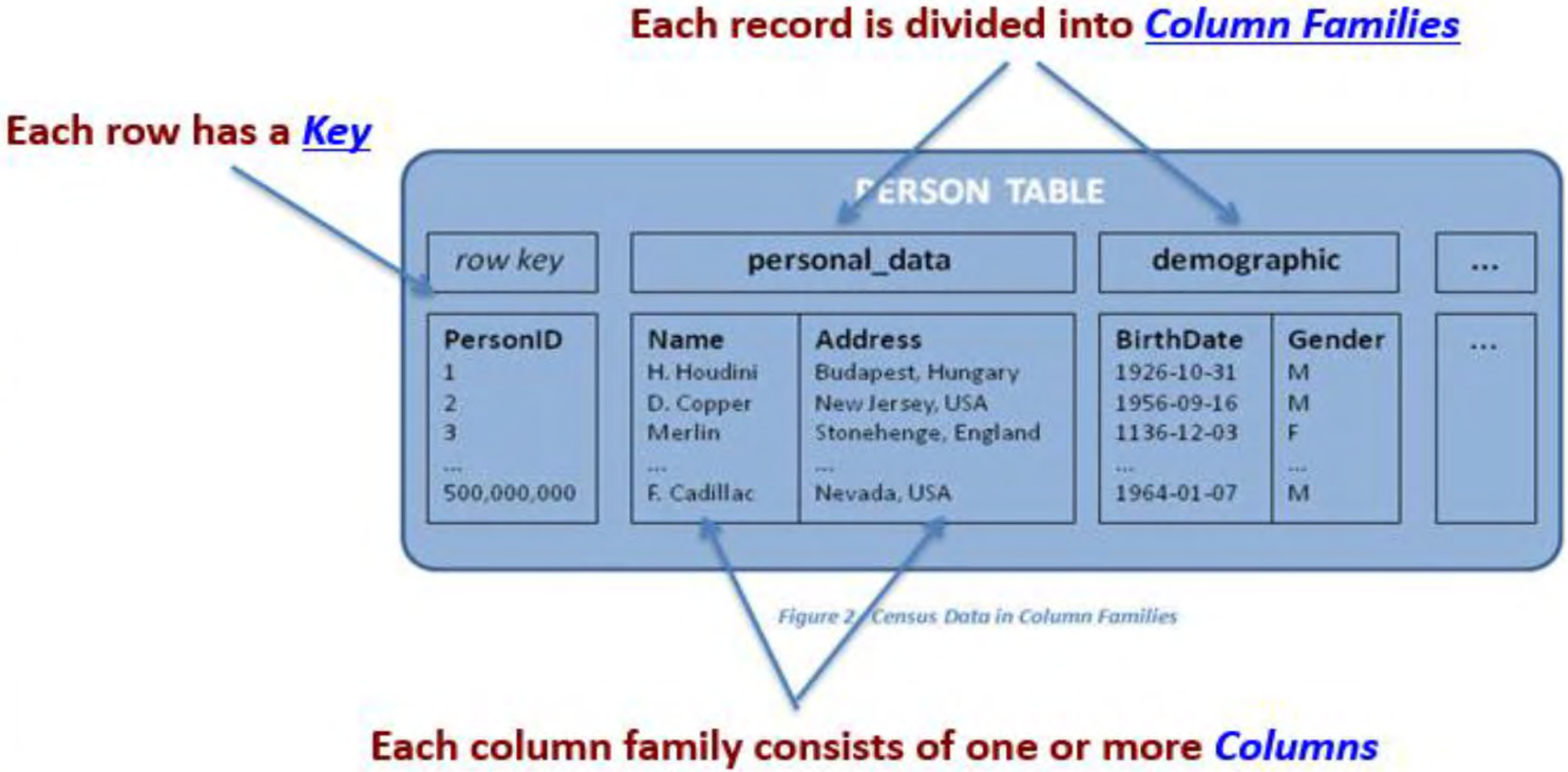
HBase Data Model

HBase is based on Google's Bigtable model

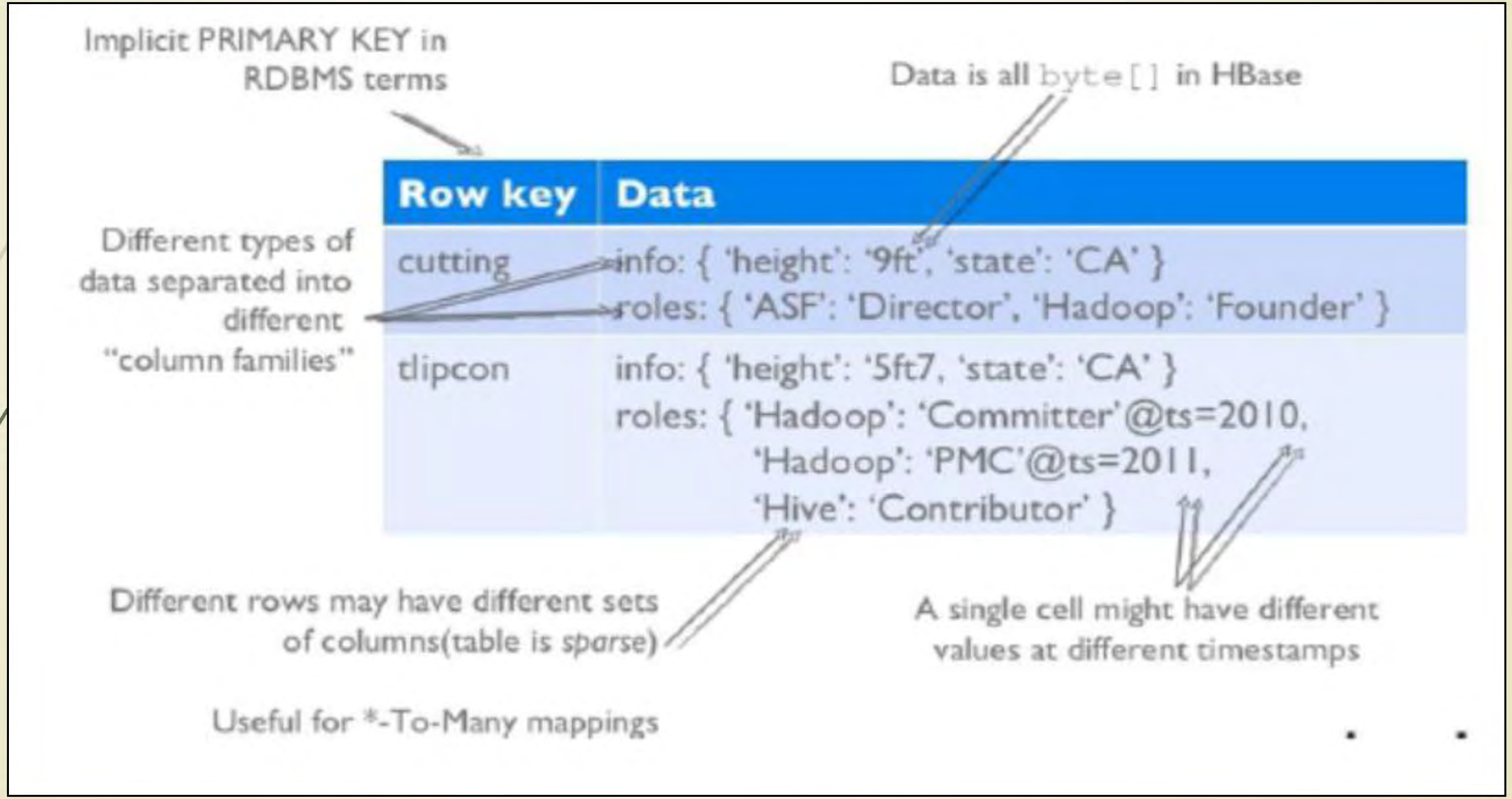
- Key-Value pairs



HBase: Keys and Column Families



HBase Data Model: Logical View



- **Key**
  - Byte array
  - Serves as the primary key for the table
  - Indexed far fast lookup
- **Column Family**
  - Has a name (string)
  - Contains one or more related columns
- **Column**
  - Belongs to one column family
  - Included inside the row
    - *familyName:columnName*

Column family named "Contents"

Column family named "anchor"

Column named "apache.com"

Row key	Time Stamp	Column "contents:"	Column "anchor:"	
"com.apache.www"	t12	"<html>…"		
	t11	"<html>…"		
	t10		"anchor:apache.com"	"APACHE"
"com.cnn.www"	t15		"anchor:cnnsi.com"	"CNN"
	t13		"anchor:my.look.ca"	"CNN.com"
	t6	"<html>…"		
	t5	"<html>…"		
	t3	"<html>…"		

- **Version Number**
  - Unique within each key
  - By default→ System's timestamp
  - Data type is Long
- **Value (Cell)**
  - Byte array

Version number for each row

Row key	Time Stamp	Column "contents:"	Column "anchor:"	
				value
"com.apache.www"	t12	"<html> …"		
	t11	"<html> …"		
	t10		"anchor:apache.com"	"APACHE"
"com.cnn.www"	t15		"anchor:cnnsi.com"	"CNN"
	t13		"anchor:my.look.ca"	"CNN.com"
	t6	"<html> …"		
	t5	"<html> …"		
	t3	"<html> …"		

Notes on Data Model

- HBase schema consists of several ***Tables***
- Each table consists of a set of ***Column Families***
  - Columns are not part of the schema
- HBase has ***Dynamic Columns***
  - Because column names are encoded inside the cells
  - Different cells can have different columns

"Roles" column family has different columns in different cells

Row key	Data
cutting	info: { 'height': '9ft', 'state': 'CA' } roles: { 'ASF': 'Director', 'Hadoop': 'Founder' }
tlipcon	info: { 'height': '5ft7, 'state': 'CA' } roles: { 'Hadoop': 'Committer'@ts=2010, 'Hadoop': 'PMC'@ts=2011, 'Hive': 'Contributor' }

Notes on Data Model

- The ***version number*** can be user-supplied
  - Even does not have to be inserted in increasing order
  - Version number are unique within each key
- Table can be very sparse
  - Many cells are empty
- ***Keys*** are indexed as the primary key

Has two columns
[cnnsi.com & my.look.ca]

Row Key	Time Stamp	ColumnFamily `contents`	ColumnFamily `anchor`
"com.cnn.www"	t9		`anchor:cnnsi.com` = "CNN"
"com.cnn.www"	t8		`anchor:my.look.ca` = "CNN.com"
"com.cnn.www"	t6	`contents:html` = "<html>..."	
"com.cnn.www"	t5	`contents:html` = "<html>..."	
"com.cnn.www"	t3	`contents:html` = "<html>..."	

HBase Physical Model

- Each column family is stored in a separate file (called ***HTables***)
- Key & Version numbers are replicated with each column family
- Empty cells are not stored

HBase maintains a multi-level index on values: *<key, column family, column name, timestamp>*

Table 5.3. ColumnFamily `contents`

Row Key	Time Stamp	ColumnFamily "contents:"
"com.cnn.www"	t6	`contents:html` = "<html>..."
"com.cnn.www"	t5	`contents:html` = "<html>..."
"com.cnn.www"	t3	`contents:html` = "<html>..."

Table 5.2. ColumnFamily `anchor`

Row Key	Time Stamp	Column Family `anchor`
"com.cnn.www"	t9	`anchor:cnnsi.com` = "CNN"
"com.cnn.www"	t8	`anchor:my.look.ca` = "CNN.com"

HBase Physical Model

Example

Row key	Data
cutting	info: { 'height': '9ft', 'state': 'CA' } roles: { 'ASF': 'Director', 'Hadoop': 'Founder' }
tlipcon	info: { 'height': '5ft7', 'state': 'CA' } roles: { 'Hadoop': 'Committer'@ts=2010, 'Hadoop': 'PMC'@ts=2011, 'Hive': 'Contributor' }

`info` **Column Family**

Row key	Column key	Timestamp	Cell value
cutting	info:height	1273516197868	9ft
cutting	info:state	1043871824184	CA
tlipcon	info:height	1273878447049	5ft7
tlipcon	info:state	1273616297446	CA

`roles` **Column Family**

Row key	Column key	Timestamp	Cell value
cutting	roles:ASF	1273871823022	Director
cutting	roles:Hadoop	1183746289103	Founder
tlipcon	roles:Hadoop	1300062064923	PMC
tlipcon	roles:Hadoop	1293388212294	Committer
tlipcon	roles:Hive	1273616297446	Contributor

Sorted on disk by Row key, Col key, descending timestamp

Milliseconds since unix epoch

cloudera

HBase Region

- Each HTable (column family) is partitioned horizontally into ***regions***
  - Regions are counterpart to HDFS blocks

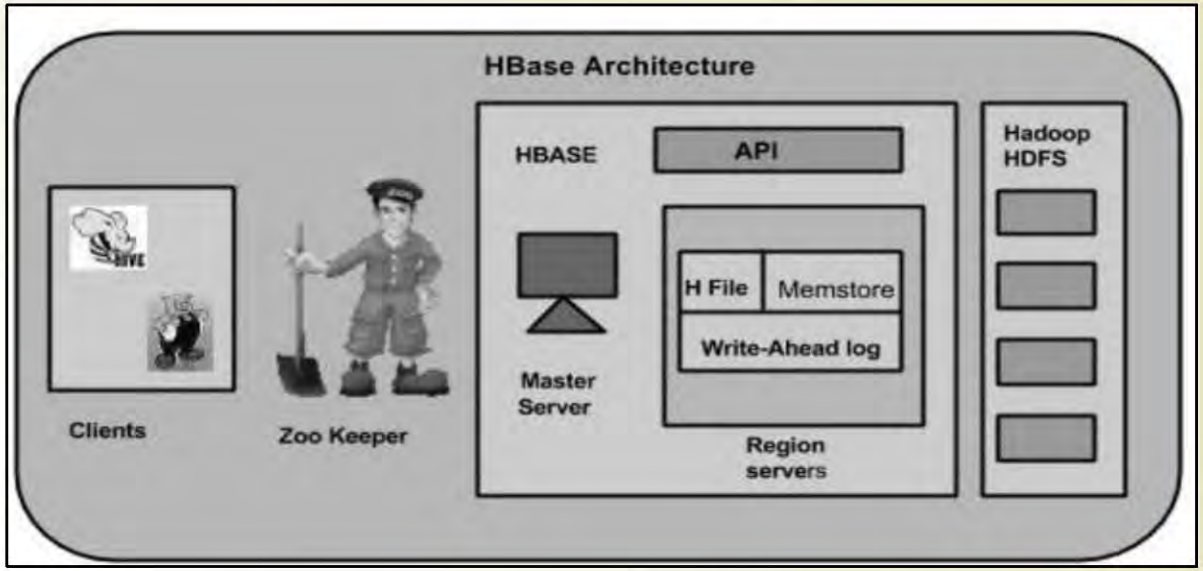
Table 5.3. ColumnFamily `contents`

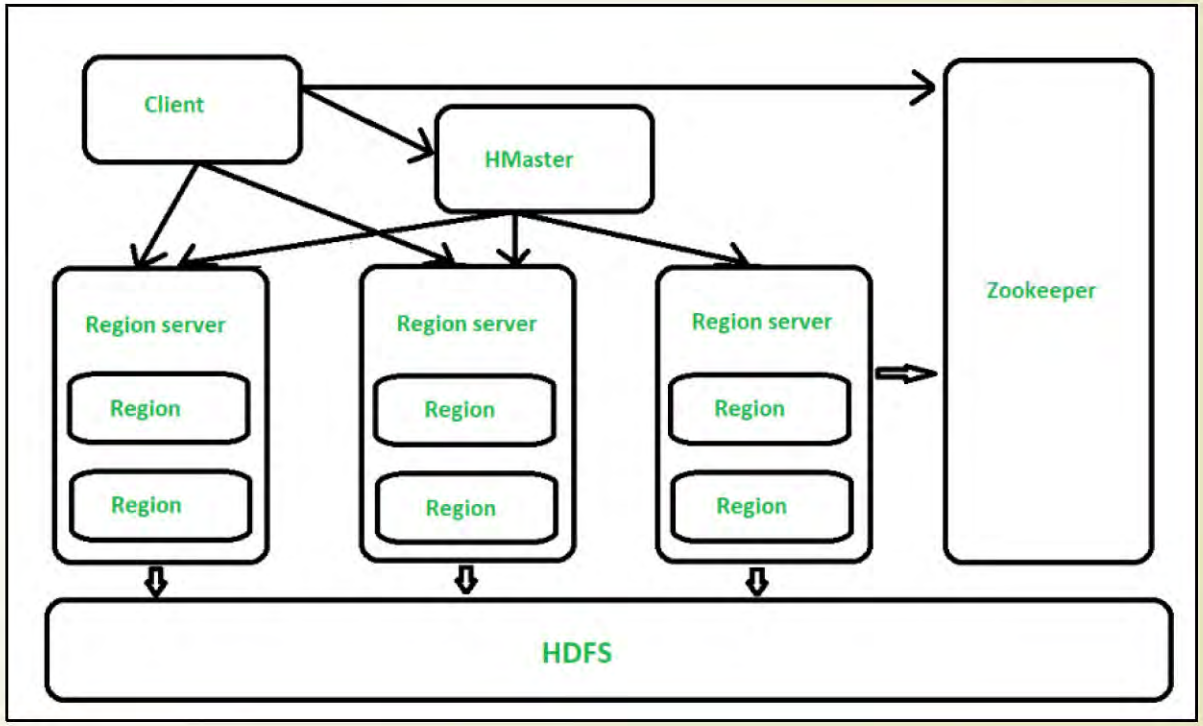
Row Key	Time Stamp	ColumnFamily "contents:"
"com.cnn.www"	t6	`contents:html` = "<html>..."
"com.cnn.www"	t5	`contents:html` = "<html>..."
"com.cnn.www"	t3	`contents:html` = "<html>..."

Each will be one region

HBase Architecture

- *Three Major Components*
  - *The HBaseMaster*
    - One master
  - *The HRegionServer*
    - Many region servers
  - *The HBase client*





HBase Architecture: The master server

- Assigns regions to the region servers and takes the help of Apache ***ZooKeeper*** for this task.
- Handles load balancing of the regions across region servers. It unloads the busy servers and shifts the regions to less occupied servers.
- Maintains the state of the cluster by negotiating the load balancing.
- Is responsible for schema changes and other metadata operations such as creation of tables and column families.
- Responsible for coordinating the slaves
- Assigns regions, detects failures
- Admin functions

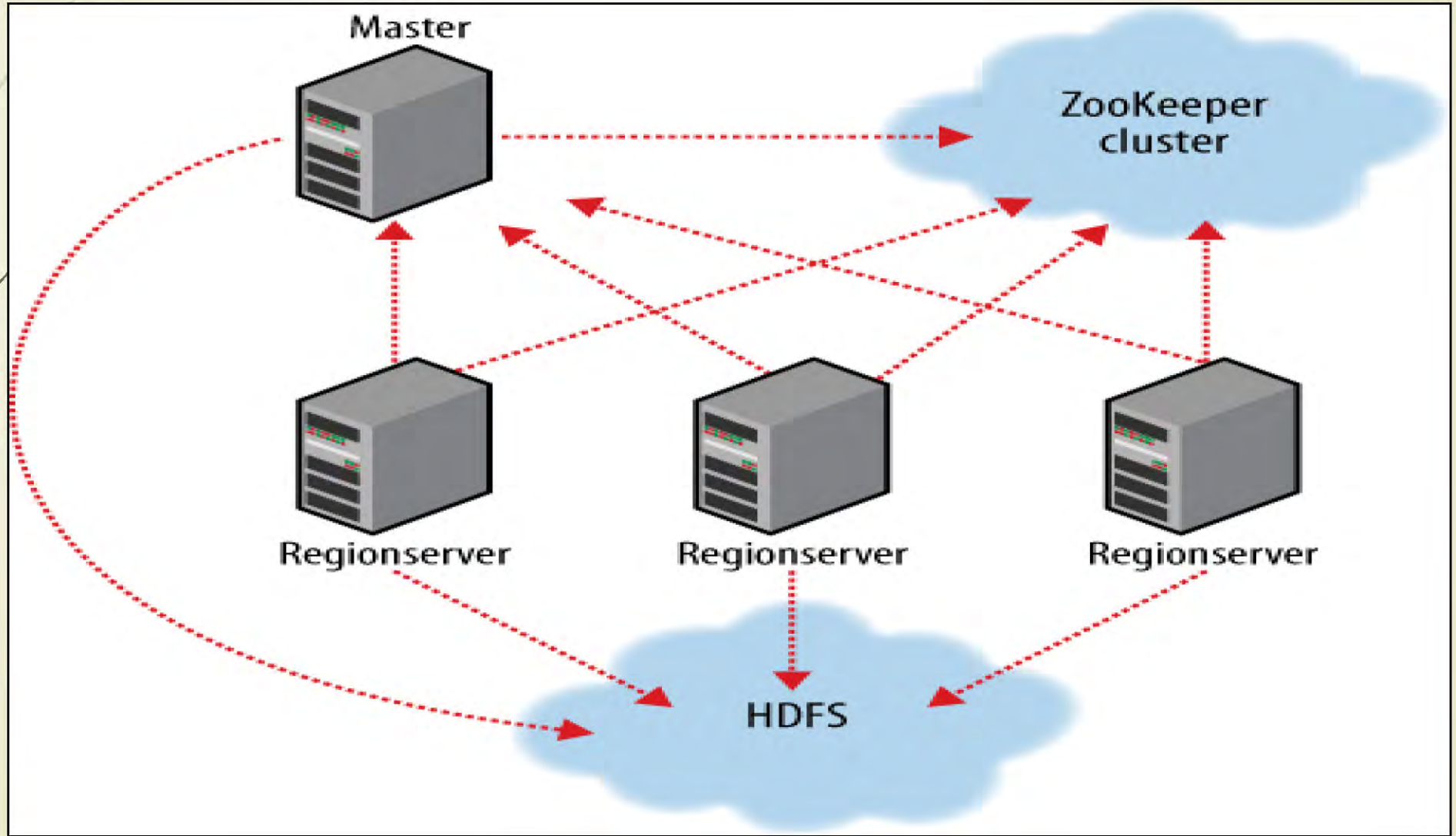
HBase Architecture: Region server

- Regions are nothing but tables that are split up and spread across the region servers.
- ***Region server***
  - The region servers have regions that -
  - Communicate with the client and handle data related operations. Handle read and write
  - requests for all the regions under it.
  - Decide the size of the region by following the
  - region size thresholds.

HBase Architecture: Zookeeper

- HBase depends on ZooKeeper
- Zookeeper is an ***open-source project*** that provides ***services like*** maintaining configuration information, naming, providing distributed synchronization, and providing group services.
- Zookeeper has ephemeral nodes representing different region servers. ***Master servers use these nodes to discover available servers.***
- In addition to availability, the nodes are ***also used to track server failures or network partitions.***
- Clients communicate with ***region servers via zookeeper.***
- ***In pseudo and standalone modes***, HBase itself will take care of zookeeper.
- ***By default HBase manages the ZooKeeper*** instance
  - E.g., starts and stops ZooKeeper
- ***HMaster and HRegionServers register themselves*** with ZooKeeper

HBase Architecture: Zookeeper



HBase: General Commands

- ***status:*** Provides the status of HBase, for example, the number of servers.
- ***version:*** Provides the version of HBase being used.
- ***table_help:*** Provides help for table-reference commands.
- ***whoami:*** Provides information about the user.
- ***create:*** Creates a table.
- ***list:*** Lists all the tables in HBase.
- ***disable:*** Disables a table.
- ***is_disabled:*** Verifies whether a table is disabled.
- ***enable:*** Enables a table.
- ***is_enabled:*** Verifies whether a table is enabled.
- ***describe:*** Provides the description of a table.
- ***alter:*** Alters a table.
- ***exists:*** Verifies whether a table exists.
- ***drop:*** Drops a table from HBase.

HBase: Creating Table

```
HBaseAdmin admin= new HBaseAdmin(config);
HColumnDescriptor []column;
column= new HColumnDescriptor[2];
column[0]=new HColumnDescriptor("columnFamily1:");
column[1]=new HColumnDescriptor("columnFamily2:");
HTableDescriptor desc= new
   HTableDescriptor(Bytes.toBytes("MyTable"));
desc.addFamily(column[0]);
desc.addFamily(column[1]);
admin.createTable(desc);
```

HBase: Operations on Regions- Get()

- Given a key → return corresponding record
- For each value return the highest version

```
Get get = new Get(Bytes.toBytes("row1"));
Result r = htable.get(get);
5.8.1.2. Default Get Example ue(Bytes.toBytes("cf"), Bytes.toBytes("attr"));  // returns current version of value
```

- Can control the number of versions you want

```
Get get = new Get(Bytes.toBytes("row1"));
get.setMaxVersions(3);  // will return last 3 versions of row
Result r = htable.get(get);
byte[] b = r.getValue(Bytes.toBytes("cf"), Bytes.toBytes("attr"));  // returns current version of value
List<KeyValue> kv = r.getColumn(Bytes.toBytes("cf"), Bytes.toBytes("attr"));  // returns all versions of
```

HBase: Operations on Regions- put()

- Insert a new record (with a new key), Or
- Insert a record for an existing key

Implicit version number (timestamp)

```
Put put = new Put(Bytes.toBytes(row));
put.add(Bytes.toBytes("cf"), Bytes.toBytes("attr1"), Bytes.toBytes( data));
htable.put(put);
```

Explicit version number

```
Put put = new Put( Bytes.toBytes(row));
long explicitTimeInMs = 555;  // just an example
put.add(Bytes.toBytes("cf"), Bytes.toBytes("attr1"), explicitTimeInMs, Bytes.toBytes(data));
htable.put(put);
```

Document-oriented Database

- ***Document-oriented database***, or document store, is a computer program designed ***to allow the inserting, retrieving, and manipulating of semi-structured data.*** The ***central concept*** of a document-oriented database is ***the notion of a document.***
- Most of the databases available under this category ***use XML, JSON*** (Java Script Object Notation), ***BSON*** (which is a binary encoding of JSON objects).
- The ***data which is a collection of key value pairs*** is compressed as a document store quite similar to a key-value store, but the only difference is that the ***values stored (referred to as "documents") provide some structure and encoding of the managed data.***
- ***Compared to RDBMS***, the documents themselves ***act as records (or rows),*** however, it is semi-structured as compared to rigid RDBMS.
- ***Even though the documents do not follow a strict schema***, indexes can be created and queried.

Document-oriented Database

- One document may provide ***an employee whose whole details are not completely known:***

```
{
"EmployeeID": "SM1",
"FirstName" : "Anuj",
"LastName" : "Sharma",
"Age" : 45,
"Salary" : 10000000
}
```

Document-oriented Database

- A second document may ***have complete details about another employee:***

```
{
"EmployeeID": "MM2",
"FirstName" : "Anand",
"Age" : 34,
"Salary" : 5000000,
"Address" : {
        "Line1" : "123, 4th Street",
        "City" : "Bangalore",
        "State" : "Karnataka"
    },
    "Projects" : [
    "nosql-migration",
    "top-secret-007"
 ]
}
```

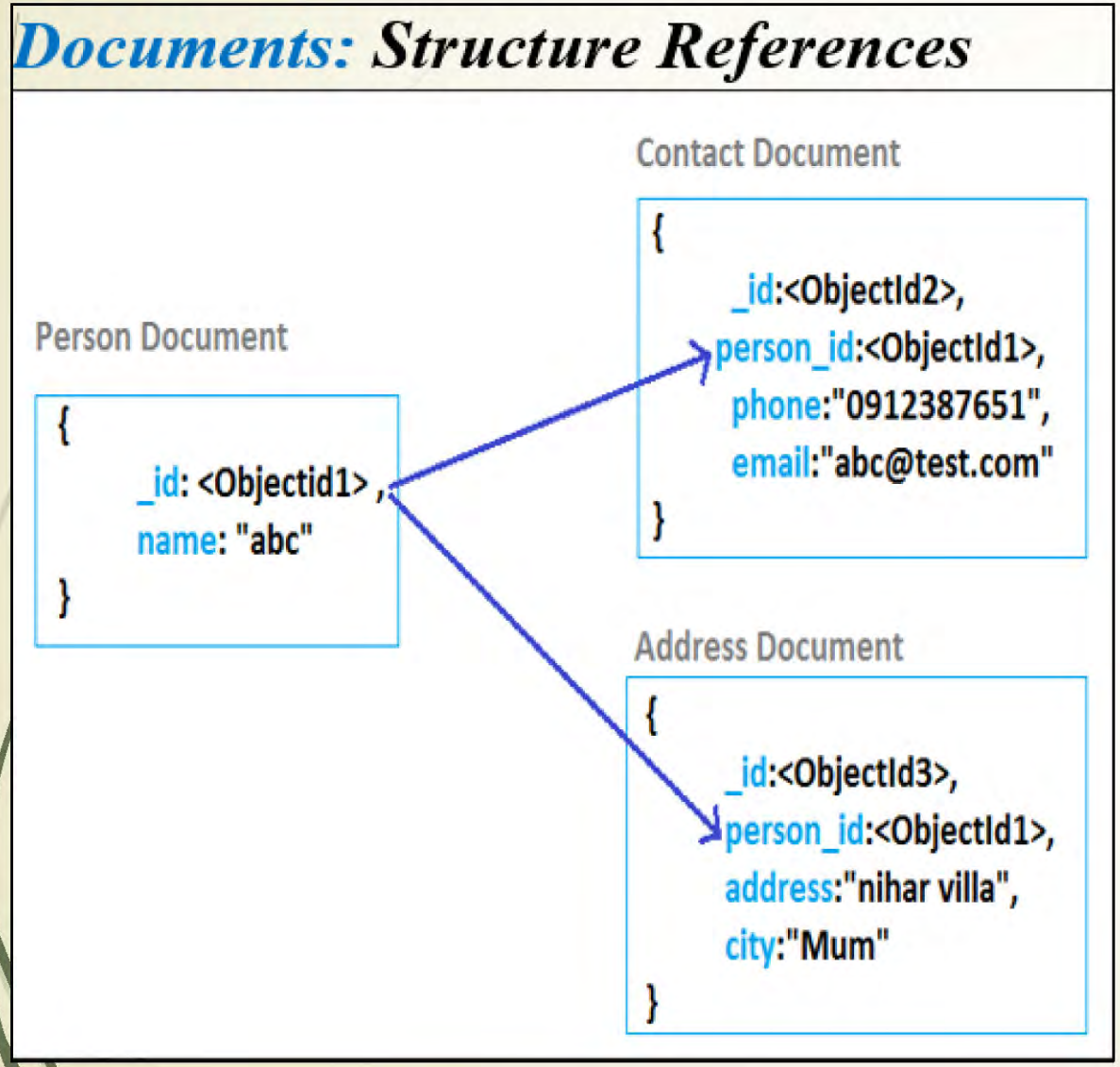
Document-oriented Database

- The ***first two documents are somewhat similar*** with the second document having more details as compared to the first. However, if you ***look at the third document***, the content has ***no correlation to the first two documents*** whatsoever—this is about an office location rather than an employee.
- A third document may have information about one of the office locations:

```
{
"LocationID"         : "Bangalore-SDC-BTP-CVRN",
"RegisteredName"     : "ACME Software Development Ltd"
"RegisteredAddress" : {
    "Line1" : "123, 4th Street",
    "City" : "Bangalore",
    "State" : "Karnataka"
    },
}
```

Document-oriented Database

Documents: Structure References



Document : Structure Embedded

```
{
  _id: <ObjectId1>,
  username: "123xyz",
  contact: {
             phone: "123-456-7890",
             email: "xyz@example.com"
           },
  access: {
             level: 5,
             group: "dev"
           }
}
```

Embedded sub-document

Embedded sub-document

Data Models: Relational to Document

Relational

Person:

Pers_ID	Surname	First_Name	City
0	Miller	Paul	London
1	Ortega	Alvaro	Valencia
2	Huber	Urs	Zurich
3	Blanc	Gaston	Paris
4	Bertolini	Fabrizio	Rom

no relation

Car:

Car_ID	Model	Year	Value	Pers_ID
101	Bentley	1973	100000	0
102	Rolls Royce	1965	330000	0
103	Peugeot	1993	500	3
104	Ferrari	2005	150000	4
105	Renault	1998	2000	3
106	Renault	2001	7000	3
107	Smart	1999	2000	2

MongoDB Document

```
{
  first_name: 'Paul',
  surname: 'Miller'
  city: 'London',
  location: [45.123,47.232],
  cars: [
   { model: 'Bentley',
     year: 1973,
     value: 100000, ... },
   { model: 'Rolls Royce',
     year: 1965,
     value: 330000, ... }
  ]
}
```

Document-oriented Database

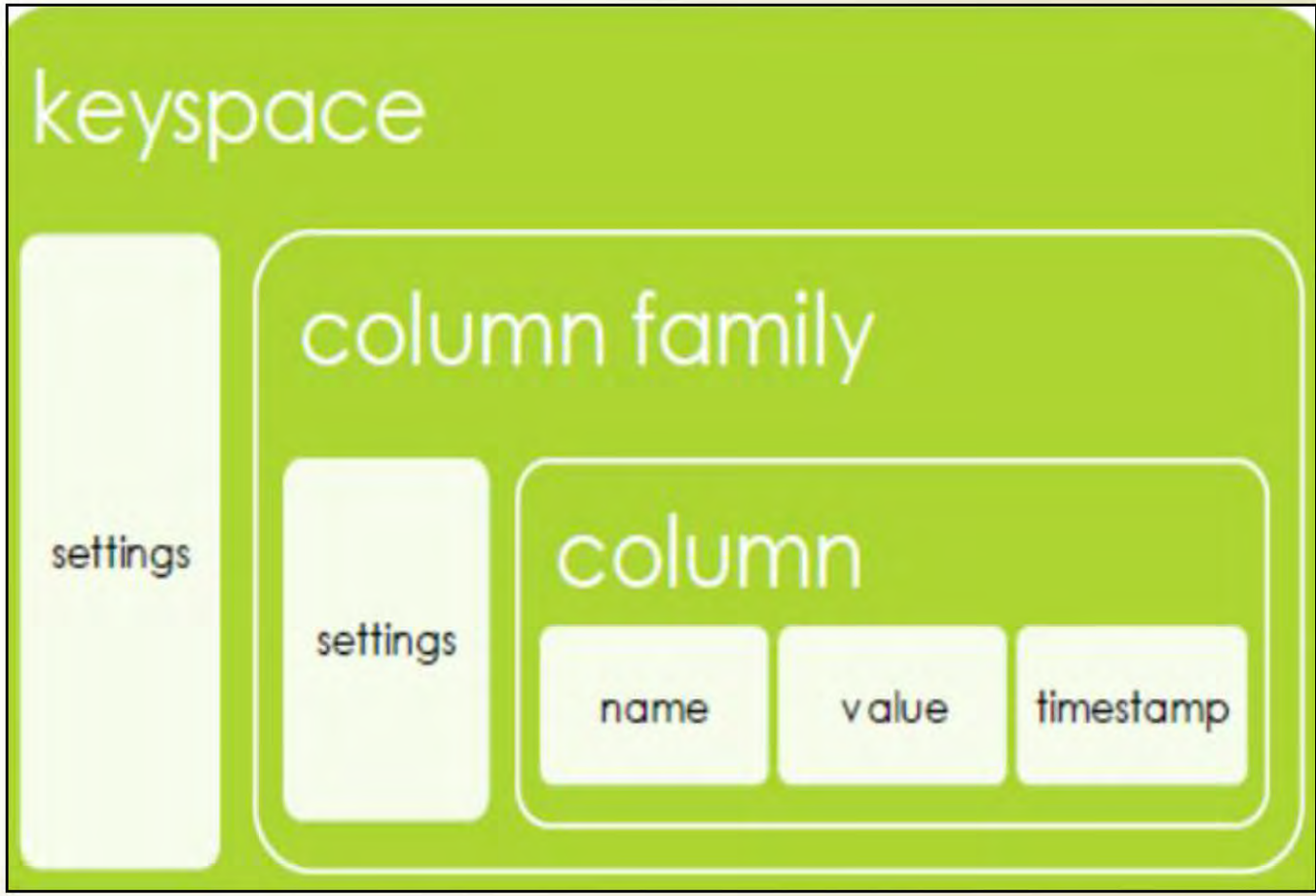
- Document-oriented databases provide flexibility—dynamic or changeable schema or even schema less documents.
- Because of the limitless flexibility provided in this model, this is one of the more popular models implemented and used.
- Some of popular databases that provide document-oriented storage include.
- ***MongoDB***
- CouchDB
- ***Apache Cassandra***
- Terrastore
- Redis
- BaseX

Apache Cassandra

- ***History of Cassandra***
  - Cassandra was created ***to power the 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
- ***What is Apache Cassandra?***
  - Apache Cassandra is an open source ***NoSQL distributed database*** trusted by thousands of companies ***for scalability and high availability without compromising performance.***
  - ***Linear scalability*** and proven ***fault-tolerance on commodity hardware*** or cloud infrastructure make it the ***perfect platform for mission-critical data.***

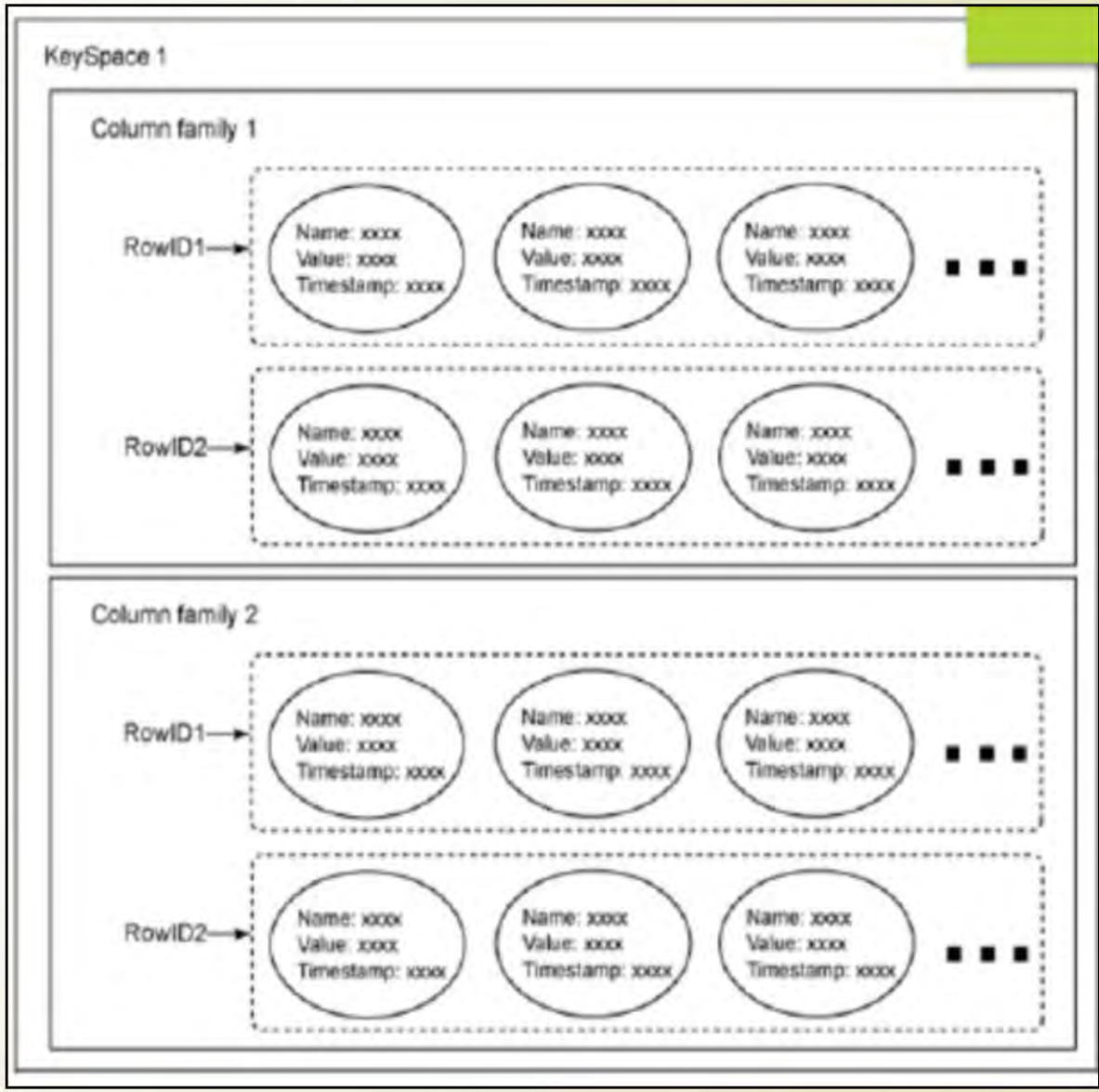
Apache Cassandra: Data Model

- 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



Apache Cassandra: Data Model

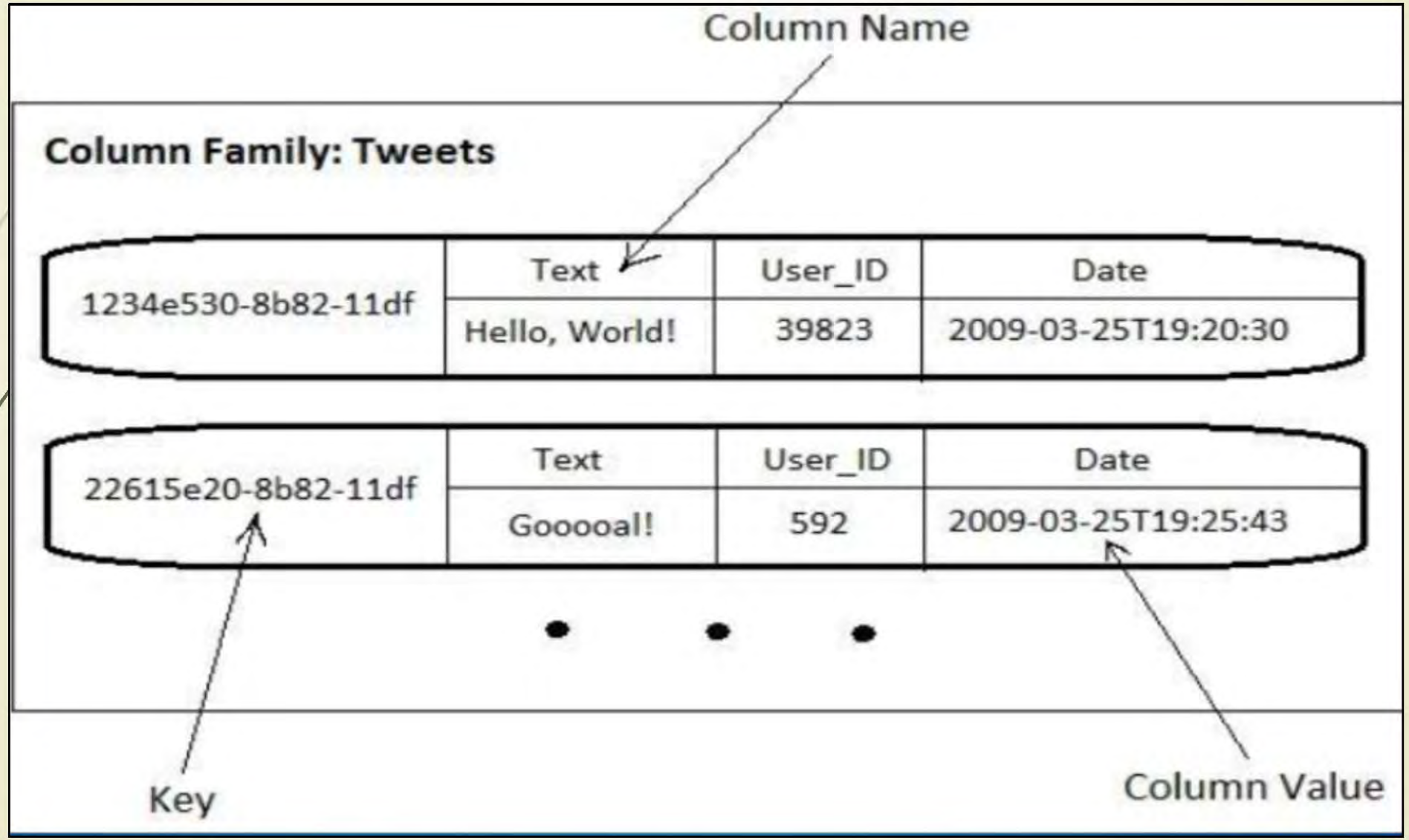
- At the heart of Cassandra ***lies two structures***: ***column family*** and ***cell***. There is a container entity for these entities called keyspace. ***Keyspace is the outermost container for data*** in Cassandra.
- ***Column families:*** sets of key-value pairs
  - ***column family as a table*** and ***key-value pairs as a row*** (using relational database analogy)
- A ***row is a collection of columns*** labeled with a name



Apache Cassandra

- ***Keyspace:*** Keyspace is the ***outermost container for data in Cassandra***. The ***basic attributes of a Keyspace in Cassandra are***:
  - ***Replication factor*** - It is the number of machines in the cluster that will receive copies of the same data.
  - ***Replica placement strategy*** - It is nothing but the strategy to place replicas in the ring. In ***Simple Strategy***, it allows a single integer RF (Replication Factor) to be defined. It determines the number of nodes that should contain a copy of each row. For example, if replication factor is 2, then two different nodes should store a copy of each row. ***Network Topology Strategy***, it allows a replication factor to be specified for each datacenter in the cluster.
  - ***Column families*** - Keyspace is a container for a list of one or more column families. A column family, in turn, is a container of a collection of rows. Each row contains ordered columns. Column families represent the structure of your data. Each keyspace has at least one and often many column families.

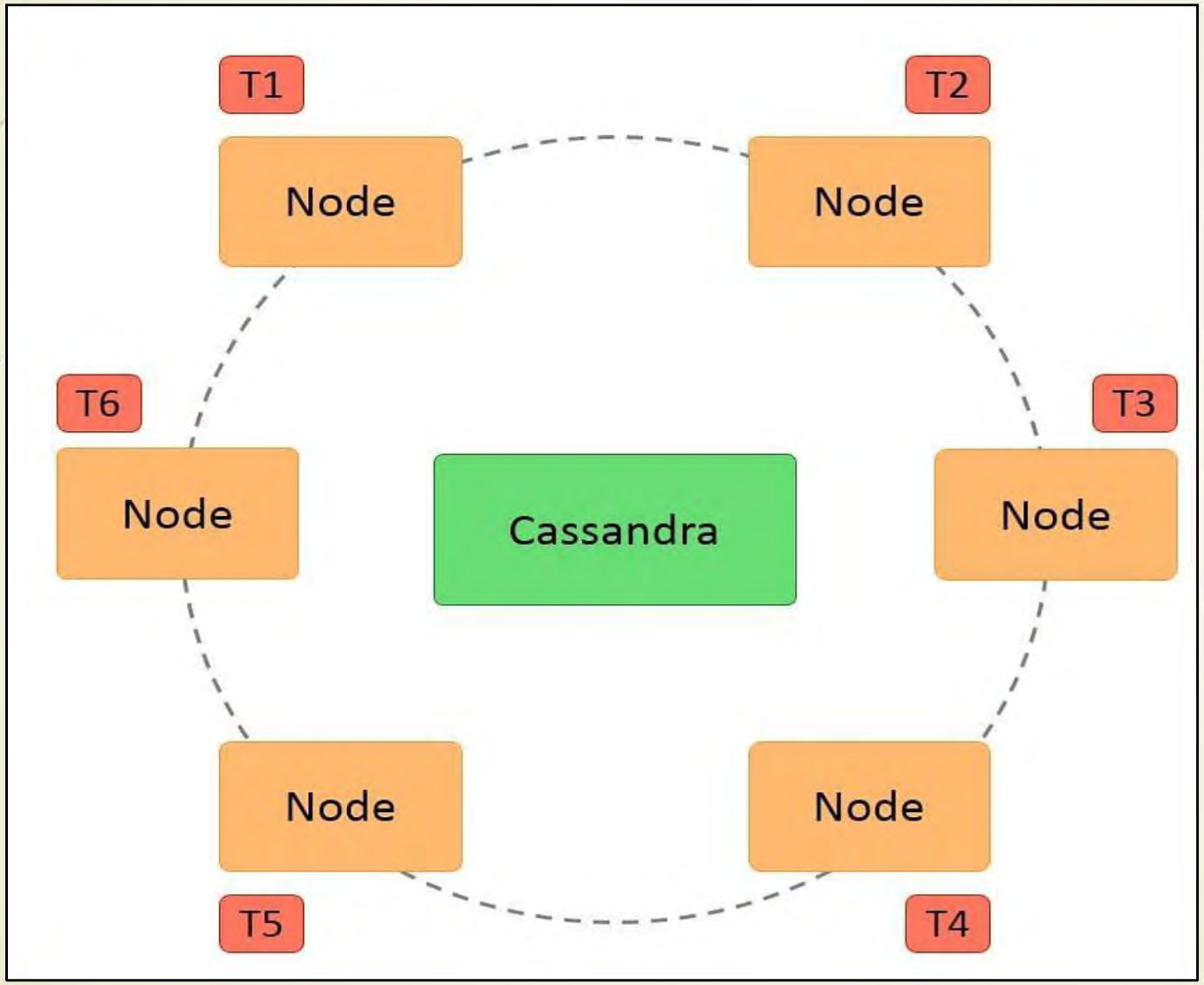
Apache Cassandra: Data Model



Apache Cassandra Architecture

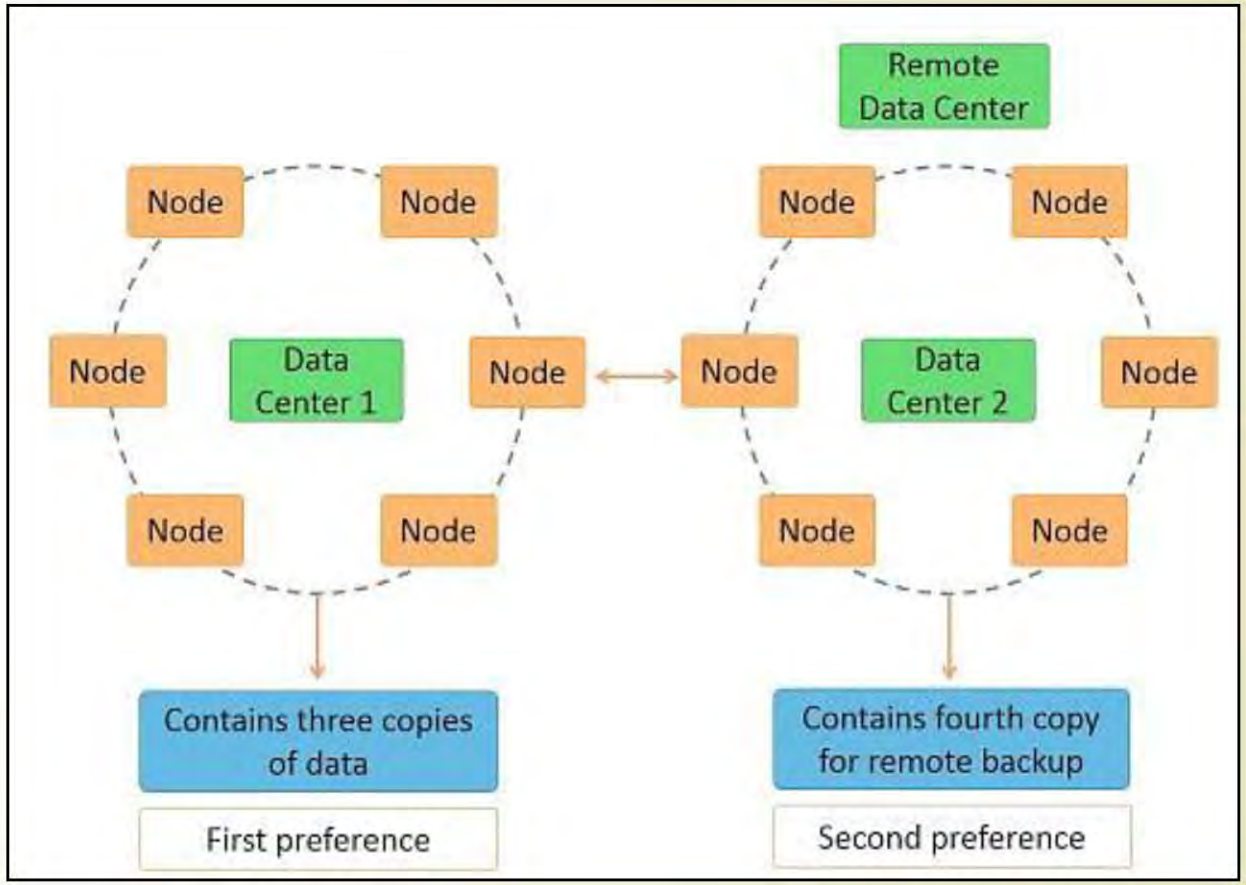
- The ***features of Cassandra architecture*** are as follows:
  - Cassandra is designed such that it ***has no master or slave nodes.***
  - It has a ***ring-type architecture***, that is, its nodes are logically distributed like a ring.
  - Data is ***automatically distributed across all the nodes.***
  - Similar to HDFS, ***data is replicated across the nodes*** for redundancy.
  - ***Data is kept in memory*** and lazily written to the disk.
  - ***Hash values of the keys are used to distribute the data*** among nodes in the cluster.
- A hash value is a number that maps any given key to a numeric value. For example, ***the string 'ABC' may be mapped to 101***, and decimal number ***25.34 may be mapped to 257***. A hash value is generated using an algorithm so that the same value of the key always gives the same hash value. ***In a ring architecture, each node is assigned a token value***, as shown in the image below:

Apache Cassandra Architecture



Apache Cassandra Architecture

- Additional features of Cassandra architecture are:
  - *Cassandra architecture supports multiple data centers.*
  - *Data can be replicated across data centers.*
- You can keep ***three copies of data in one data center*** and ***the fourth copy in a remote data center for remote backup***.
- Data reads prefer a local data center to a remote data center.



Advantages vs. Disadvantages

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

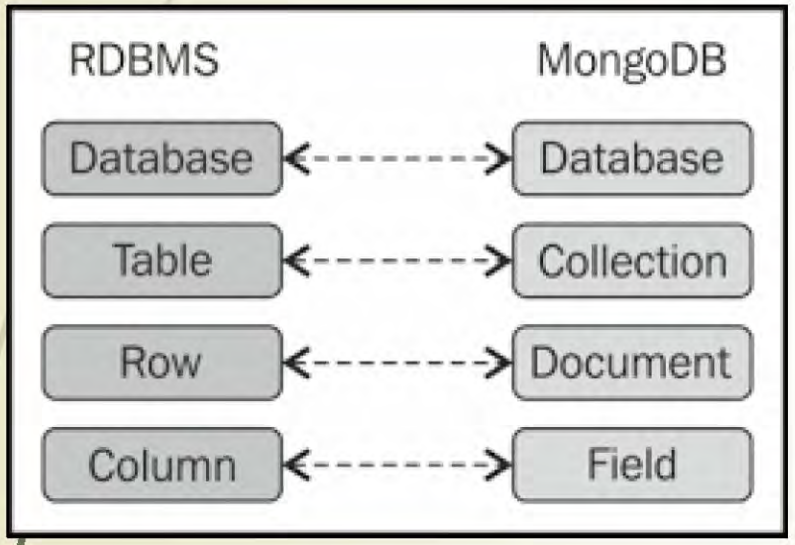
- no referential integrity — no concept of JOIN
- querying options for retrieving data are limited
- sorting data is a design decision — no GROUP BY
- no support for atomic operations — if operation fails, changes can still occur
- first think about queries, then about data model

MongoDB

- MongoDB is based on ***a NoSQL database*** that is used for ***storing data in a key-value pair***. Its ***working is based*** on the ***concept of document and collection.***
- It is also an ***open-source, a document-oriented, cross-platform*** database system that is written using C++.
- A ***record in MongoDB is a document***, which is a data structure ***composed of field and value pairs***. MongoDB documents ***are similar to JSON objects***.
- The ***values of fields may include*** other documents, arrays, and arrays of documents.

```
{
  name: "sue",                          <-- field: value
  age: 26,                              <-- field: value
  status: "A",                          <-- field: value
  groups: [ "news", "sports" ]          <-- field: value
}
```

Mapping RDMS to MongoDB



Relational Database

Student_Id	Student_Name	Age	College
1001	Chaitanya	30	Beginnersbook
1002	Steve	29	Beginnersbook
1003	Negan	28	Beginnersbook

MongoDB

```
{
 "_id": ObjectId("....."),
 "Student_Id": 1001,
 "Student_Name": "Chaitanya",
 "Age": 30,
 "College": "Beginnersbook"
}
{
 "_id": ObjectId("....."),
 "Student_Id": 1002,
 "Student_Name": "Steve",
 "Age": 29,
 "College": "Beginnersbook"
}
{
 "_id": ObjectId("....."),
 "Student_Id": 1003,
 "Student_Name": "Negan",
 "Age": 28,
 "College": "Beginnersbook"
}
```

Why MongoDB?

- ***Document-Oriented data storage***, i.e., data, is stored in a JSON style format, which increases the readability of data as well.
- ***Replication and high availability*** of data.
- MongoDB provides ***Auto-sharding***.
- ***Ad hoc queries are supported by MongoDB***, which helps in searching by range queries, field, or using regex terms.
- Indexing of values can be used to create and improve the overall search performance in MongoDB. ***MongoDB allows any field to be indexed within a document.***
- MongoDB has a ***rich collection of queries.***
- It can ***be integrated with other popular programming languages*** also to handle structured as well as unstructured data within various types of applications.

Key-value store Database

- A ***Key-value store is very closely related to a document store***—it allows the storage of a value against a key.
- Similar to a document store, there is ***no need for a schema*** to be enforced on the value. However, ***there a are few constraints that are enforced by a key-value store:***
  - Unlike a document store that can create a key when a new document is inserted, a ***key-value store requires the key to be specified***.
  - Unlike a document store where the value can be indexed and queried, for a key-value store, ***the value is opaque and as such, the key must be known to retrieve the value.***
- A key-value database, or key-value store, ***is a data storage paradigm designed for storing, retrieving, and managing*** associative arrays, a data structure more commonly known today as a dictionary or hash table.

Key-value store Database

- The key-value part refers to the fact that the database stores data as a collection of key/value pairs. This is a simple method of storing data, and it is known to scale well.
- A few of ***the popular key value stores are***:
  - Redis (in-memory, with dump or command-log persistence)
  - MemcacheDB (built on Memcached)
  - Berkley DB
  - Voldemort (Used in LinkedIn)

Key-value store Database

Key	Value
123456789	APPL, Buy, 100, 84.47
234567890	CERN, Sell, 50, 52.78
345678901	JAZZ, Buy, 235, 145.06
456789012	AVGO, Buy, 300, 124.50

Stock Trading

Key	Value
artist:1:name	AC/DC
artist:1:genre	Hard Rock
artist:2:name	Slim Dusty
artist:2:genre	Country

Artist Info

Phone Directory

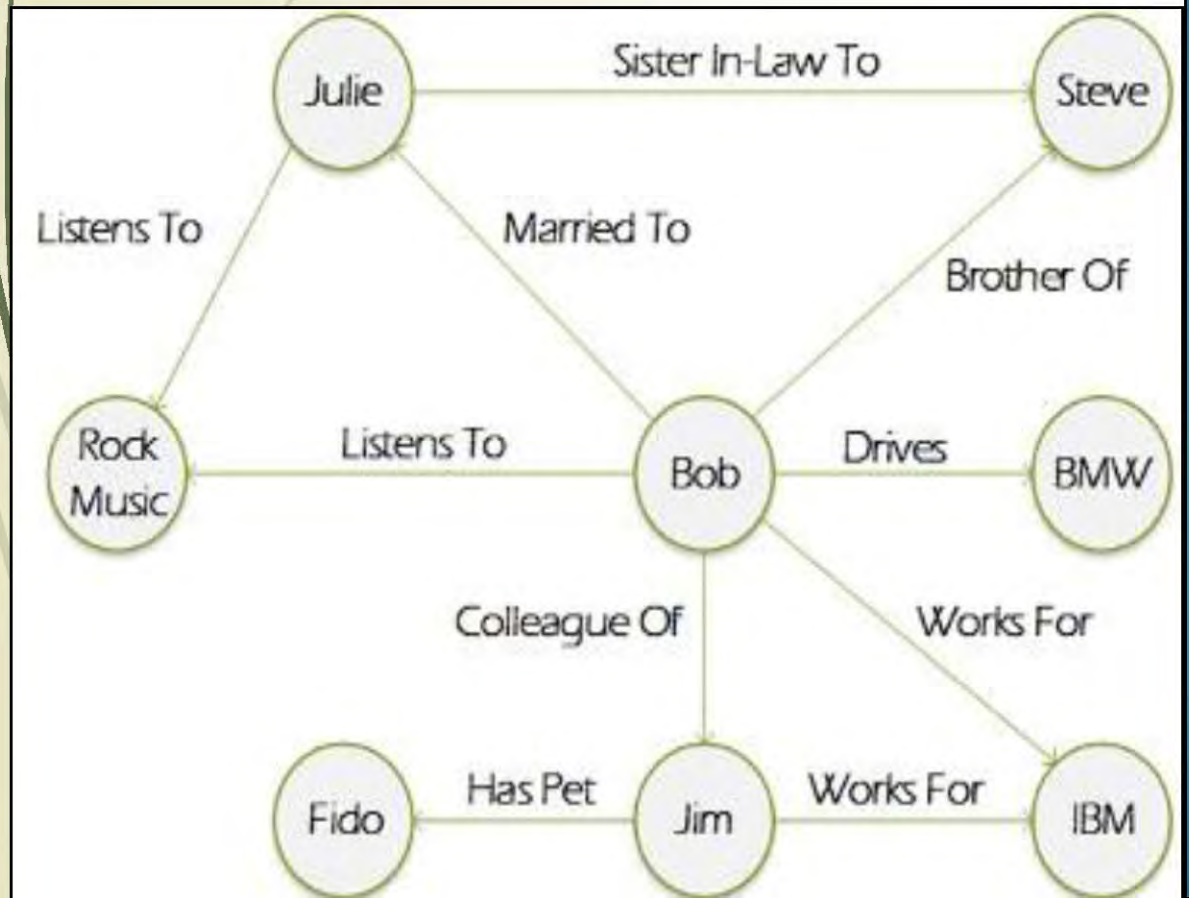
Key	Value
Bob	(123) 456-7890
Jane	(234) 567-8901
Tara	(345) 678-9012
Tiara	(456) 789-0123

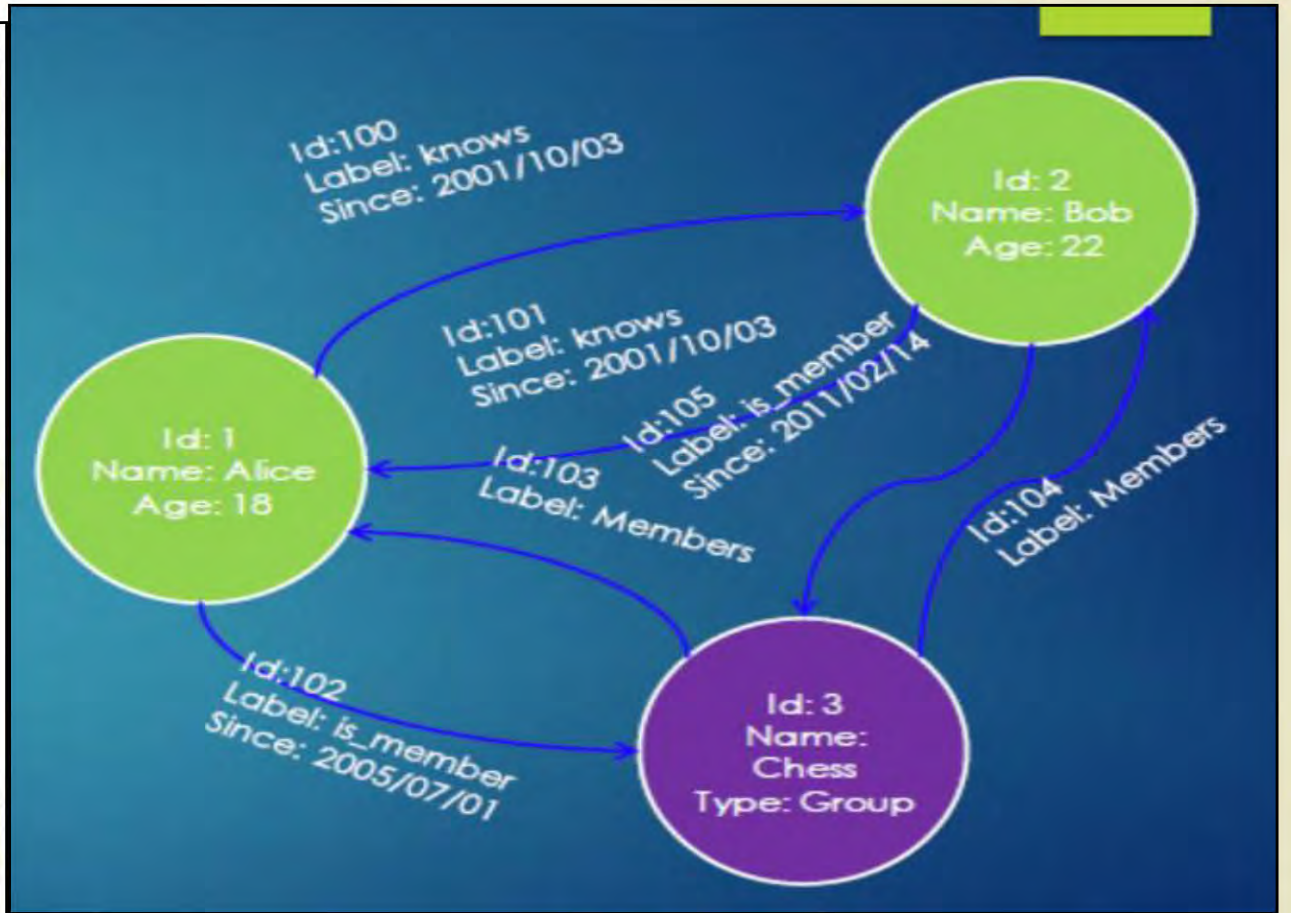
Graph Database

- The concept behind graphing a database is often credited to 18th-century mathematician Leonhard Euler.
- Graph databases represent ***a special category of NoSQL databases*** where relationships are represented as graphs. ***There can be multiple links between two nodes*** in a graph - representing the multiple relationships that the two nodes share.
- The relationships represented may ***include social relationships*** between people, transport links ***between places,*** or network ***topologies*** between connected systems.
- A graph database is essentially a collection of nodes and edges. Each node represents an entity (such as a person or business) and each edge represents a connection or relationship between two nodes.
- ***Every node in a graph database is defined by a unique identifier***, a set of outgoing edges and/or incoming edges and a set of properties expressed as key/value pairs.
- ***Each edge is defined by a unique identifier***, a starting-place and/or ending-place node and a set of properties.

Graph Database

- The mantra of graph database enthusiasts is ***"If you can whiteboard it, you can graph it."*** Example:





SQL Vs. NoSQL

unacademy

SQL	NoSQL
Databases are categorized as RDBMS.	Non-relational or distributed database system.
SQL databases have static or predefined schema.	NoSQL databases have dynamic schema.
SQL databases display data in form of tables so it is known as table-based database.	NoSQL databases display data as collection of key-value pair, documents, graph databases or wide-column stores.
SQL databases are vertically scalable.	NoSQL databases are horizontally scalable.
SQL databases use a powerful language "Structured Query Language" to define and manipulate the data.	In NoSQL databases, collection of documents are used to query the data. It is also called unstructured query language. It varies from database to database.
SQL databases are best suited for complex queries.	NoSQL databases are not so good for complex queries because these are not as powerful as SQL queries.
SQL databases are not best suited for hierarchical data storage.	NoSQL databases are best suited for hierarchical data storage.
MySQL, Oracle, Sqlite, PostgreSQL and MS-SQL etc. are the example of SQL database.	MongoDB, BigTable, Redis, RavenDB, Cassandra, Hbase, Neo4j, CouchDB etc. are the example of nosql database

Thank You

???