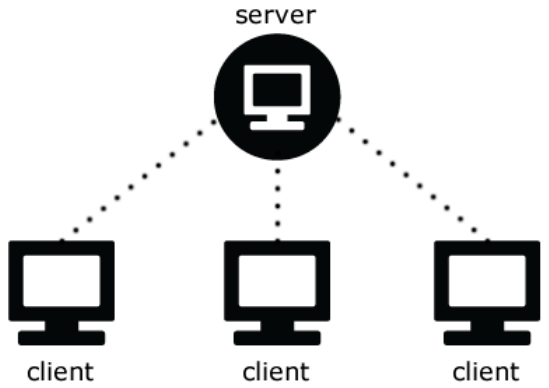

{JSON}


cassandra


{ JSON }

What is JSON?




```
{" WHAT IS JSON ? ":[
{"num":1", "expr":" {"JSON is a syntax for storing and
exchanging data. "},
{"num":2", "expr":" {"JSON is an easier-to-use
alternative to XML. "},
{"num":3", "expr":" {"JSON is lightweight format for
exchanging data b/w the client and server."},
{"num":4", "expr":" {"JSON is not a Programing
Language"},
{"num":5", "expr":" {" JSON is Subset of JavaScript "},
]}
```

{JSON}

History of JSON?

{JSON}

- Douglas Crockford in 2001
- 2002 JSON.org
- 2005 Ajax
- 2006 RFC 4627

What we use JSON?

```
{" WHAT WE USE JSON ? ":[
{"num":1", "expr":" {"Straightforward Syntax"},
{"num":2", "expr":" {"Easy To Create And Manipulate
"},
{"num":3", "expr":" {"Can Be Natively Parsed In
Javascript Using Eval() "},
{"num":4", "expr":" {"Supported By All Major
Javascript Frameworks "},
{"num":4", "expr":" {" Supported By Most Backend
Technologies "},
]}
```

{JSON}

JSON is NOT...

JSON is NOT...

- JSON is not a document format.
- JSON is not a markup language.
- JSON is not a general serialization format.
  - No cyclical/recurring structures.
  - No invisible structures.
  - No functions.

{JSON}

JSON OBJECTS

A JSON object is an unordered set of name/value pairs inserted between {} (curly braces)

```
{ "artistname" : "Deep Purple" }
```

{JSON}

```
{ }

{ "artistname" : "Deep Purple" }

{
  "artistname" : "Deep Purple",
  "formed" : "1968"
}

{
  "artistname" : "Deep Purple",
  "formed" : "1968",
  "origin" : "Hertford, United Kingdom"
}
```

In JSON, a name is a string. Its value can be an object, array, number, string, true, false, or null.

JSON ARRAYS

{JSON}

```
{
  "artists" : [
    {
      "artistname" : "Deep Purple",
      "formed" : "1968"
      },
    {
      "artistname" : "Joe Satriani",
      "born" : "1956"
      },
    {
      "artistname" : "Maroon 5",
      "formed" : "1994"
      }
  ]
}
```

NESTED DATA

{JSON}

```
{
  "artists" : [
    {
      "artistname" : "Deep Purple",
      "formed" : "1968",
      "albums" : [
        {
          "albumname" : "Machine Head",
          "year" : "1972",
          "genre" : "Rock"
        },
        {
          "albumname" : "Stormbringer",
          "year" : "1974",
          "genre" : "Rock"
        }
      ]
    }
  ]
}
```

JSON Data Types

{JSON}

- String
- Number
- Boolean
- Null
- Object
- Array

JSON Data Types

{JSON}

Data Type	Description
String	Any sequence of Unicode code points, inserted between " and " (double quotes). Some characters may need to be escaped
Number	Represented in base 10 with no superfluous leading zero. Can include digits between 0 and 9. It can be a negative number (e.g. -10. It can be a fraction (e.g. .5). It can also have an exponent of 10, prefixed by e, E, +, or -.
Boolean	This can be either true or false.
Null	Empty.

JSON Data Types

{JSON}

Data Type	Description
Object	A JSON object is an unordered set of name/value pairs inserted between {} (curly braces). An object can contain zero or more name/value pairs. Multiple name/value pairs are separated by a , (comma).
Array	A JSON array is an ordered collection of values. It allows you to provide a list of values. A JSON array begins with [(left bracket) and ends with] (right bracket). Its values are separated by , (comma).

Escape Characters

{JSON}

Character	Unicode Name/Code Point	Use this to escape it
"	Quotation mark (U+0022)	\"
\	Reverse solidus (U+005C)	\\
/	Solidus (U+002F)	\/
	Backspace (U+0008)	\b
	Form feed (U+000C)	\f
	Line feed (U+000A)	\n
	Carriage return (U+000D)	\r
	Horizontal tab (U+0009)	\t

{JSON}

```
{
  "artists" : [
    {
      "artistname" : "Deep Purple",
      "formed" : "1968"
      },
    {
      "artistname" : "Joe Satriani",
      "born" : "1956-07-15"
      },
    {
      "artistname" : "Maroon 5",
      "formed" : "1994"
      }
  ]
}
```

JSON has no rule to say that certain objects must use a certain data type, or even contain the same fields. They don't even need to contain the same *number* of fields.

```
"born" : "1956"
"born" : 1956
"born" : "July 15, 1956"
"born" : "1956-07-15"
"born" : "07/15/1956"
"born" : "15/07/1956"
"born" : "I like oranges!"
"born" : [
      {
        "albumname" : "Flying in a Blue
Dream",
        "year" : "1989",
        "genre" : "Instrumental Rock"
      },
      {
        "albumname" : "Shockwave
Supernova",
        "year" : "2015",
        "genre" : "Instrumental Rock"
      }
    ]
"born" : "Oooops!!!"
```

Creating a Schema

- You can apply rules to your JSON files by creating a schema. A JSON Schema allows you to specify what type of data can go into your JSON files.
- You can use a schema to validate the JSON file to ensure it contains only the correct type of data.

```
{ "type": "string" }
```

To Declare a JSON Schema

- The JSON Schema is in fact, JSON itself. So it's not always easy to tell whether you're looking at a JSON Schema or just an ordinary JSON document.

To declare a JSON Schema, use the $schema keyword.

```
{ "$schema": "http://json-schema.org/schema#" }
```

{JSON}

Arrays vs Objects

- Use objects when the key names are arbitrary strings.
- Use arrays when the key names are sequential integers.

JSON	XML
JSON object are	type XML data is typeless
JSON types: string, number, array, boolean	XML data are all string
Data is readily accessible as JSON objects	XML data needs to be parsed
Retrieving value is easy	Retrieving value is difficult
JSON is supported by all browsers	Cross browser XML parsing can be tricky
Simple API	Complex API
Supported by many Ajax toolkit	Not fully supported by Ajax toolkit
Fast object de-serialization in JavaScript	Slower de-serialization in JavaScript
Fully automated way of deserializing/ serializing JavaScript objects	Developers have to write JavaScript code to serialize/de-serialize to/from XML

Pros	Cons
Fast to parse	No namespace support, hence poor extensibility
Good support for all browsers	Limited development tools support
Supported by many languages	No support for formal grammar definition
Concise format: name/value pair - based approach	Retrieving value is difficult
Easy to read	Cross browser XML parsing can be tricky
Easy to write	Complex API

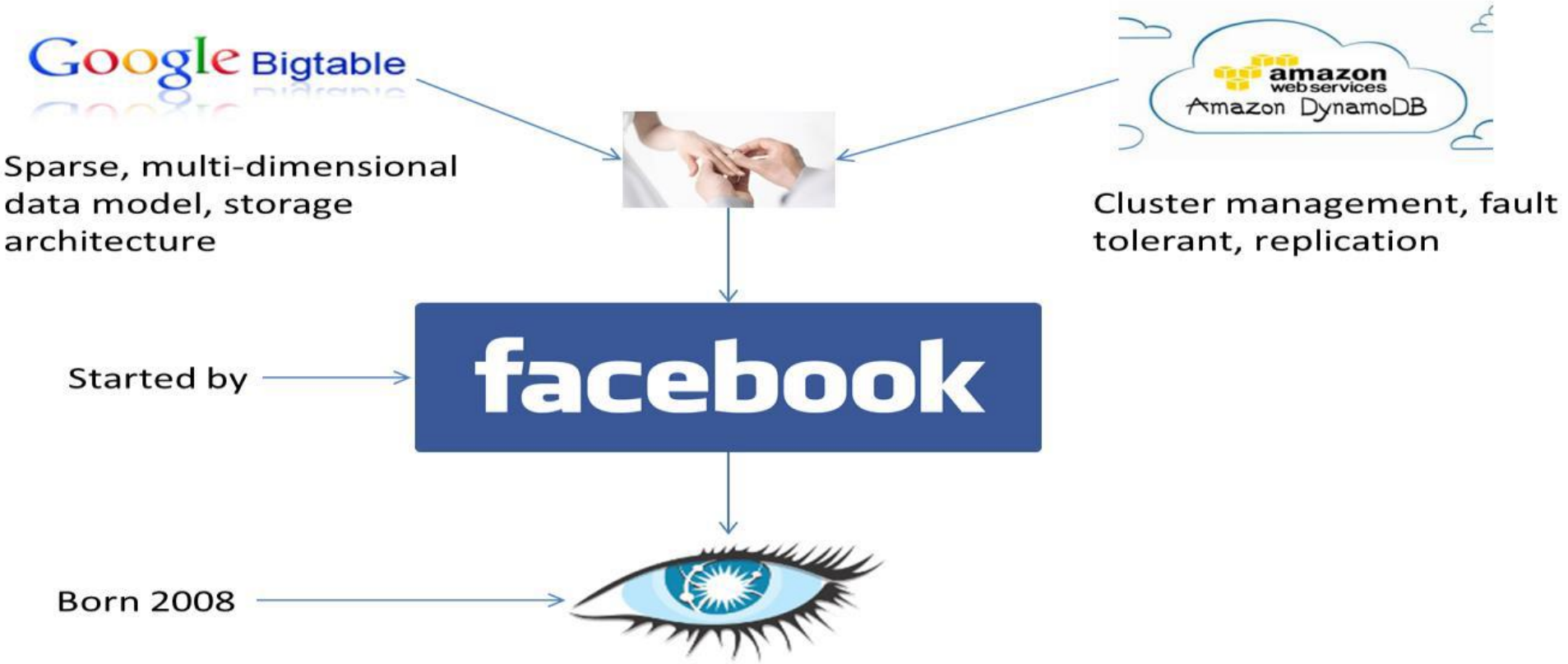

cassandra

The relational model was brought by E.F. Codd's 1970 paper which made data modeling and application programming much easier.

After the arrival of web 2.0 applications, data stores need to large scale OLTP /OLAP application loads. In this circumstance, the relational model does not provide good results. This is the gap which NoSQL systems attempt to fill, by providing a more scalable solution with high availability.

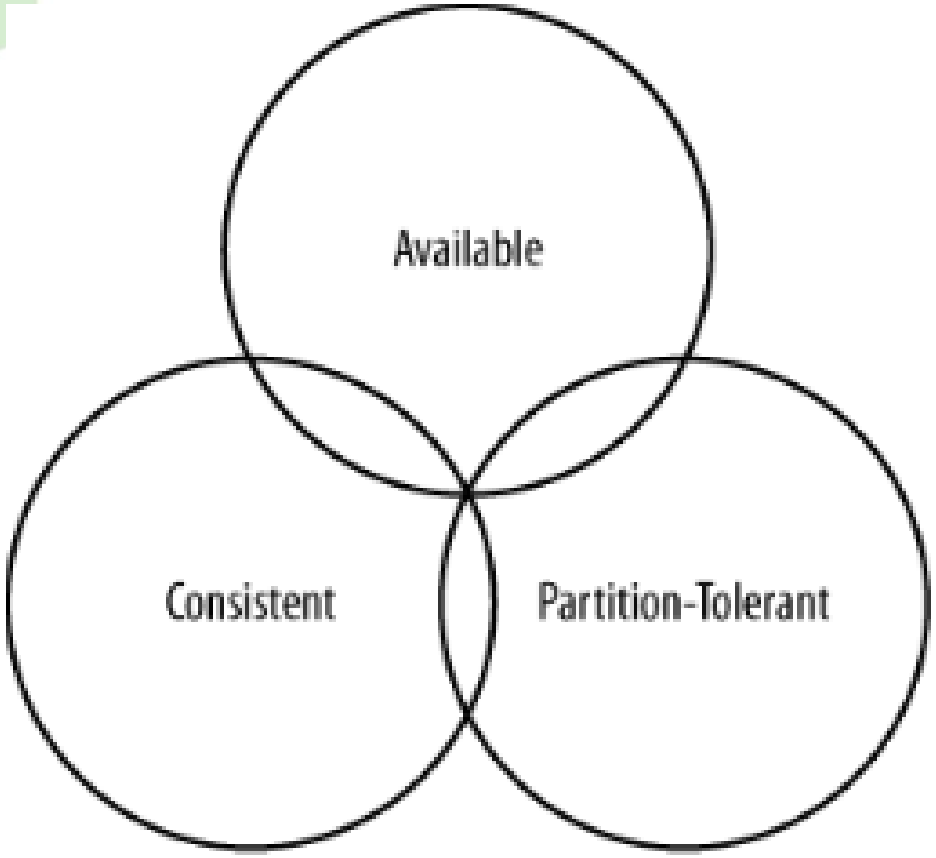
History of CASSANDRA?

History of CASSANDRA?

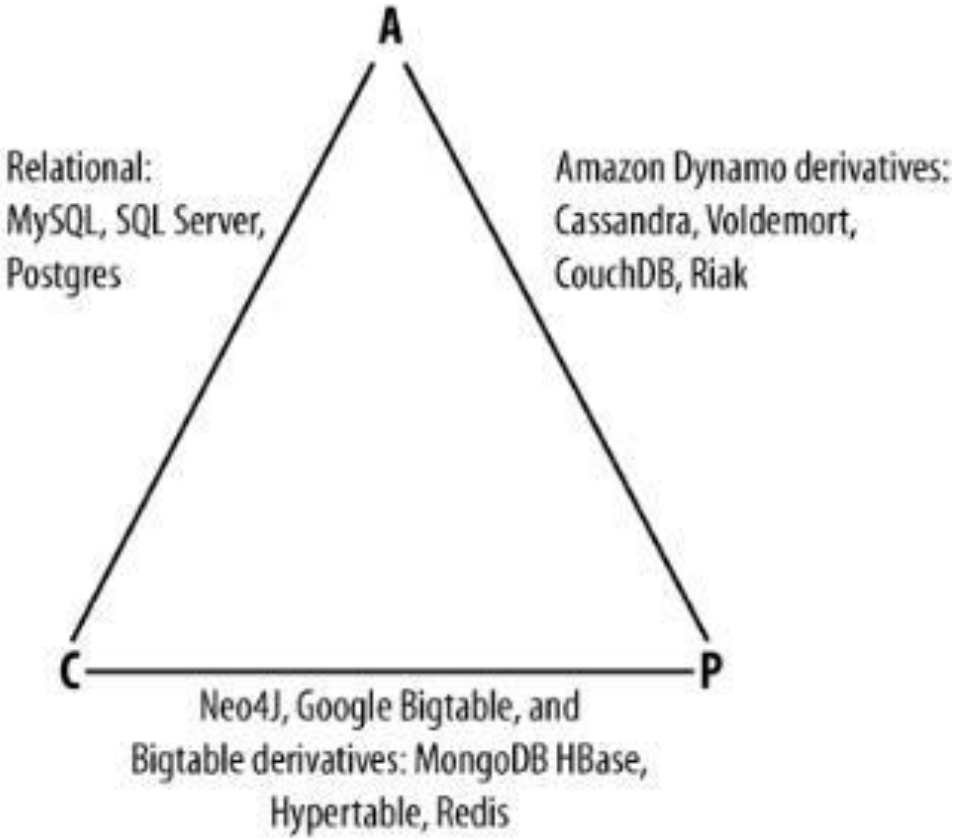


WHAT IS EVENTUALLY CONSISTENT?

Building reliable distributed systems on a global scale demands trade-offs between consistency and availability .Consistency in a nutshell means that when something is written, it is expected that all reads after the write will have access to that written data. In Cassandra, due to its distributed nature, there are no such hard guarantees. However, we can say that it eventually reaches a consistent state because all data is eventually replicated across the distributed data store.



The CAP theorem visualized as a Venn diagram.



NOSQL systems and CAP Model

USERS

- **Twitter** is using Cassandra for analytics.
- **Mahalo** uses it for its primary near-time data store.
- **Facebook** still uses it for inbox search, though they are using a proprietary fork.
- **Digg** uses it for its primary near-time data store.
- **Rackspace** uses it for its cloud service, monitoring, and logging.
- **Reddit** uses it as a persistent cache.
- **Cloudkick** uses it for monitoring statistics and analytics.
- **Ooyala** uses it to store and serve near real-time video analytics data.
- **SimpleGeo** uses it as the main data store for its real-time location infrastructure.
- **Onespot** uses it for a subset of its main data store.

ARCHITECTURE

Cassandra was developed as a peer to peer distributed system where all nodes serve the same functions, meaning there is **no single point of failure**.

One of Cassandra's greatest strength is its availability and scaling, it achieves this through a fully distributed system where data is replicated across multiple nodes according to user settings.

1. Client Writes to any Cassandra Node
2. Coordinator Node replicates to nodes and Zones
3. Nodes return ack to coordinator
4. Coordinator returns ack to client
5. Data written to internal commit log disk

If a node goes offline, hinted handoff completes the write when the node comes back up.

Requests can choose to wait for one node, a quorum, or all nodes to ack the write

SSTable disk writes and compactions occur asynchronously

Cassandra write cycle as shown by Netflix's Cassandra production rollout team.

Strengths of Cassandra

- Gigabyte to Petabyte scalability
- Flexible Schema Design
- Linear Performance of gains through node addition
- **C**assandra **Q**uery **L**anguage (like SQL)
- Data Compression
- No need for separate caching layer
- Tunable data consistency

Data Model

Relational Model	Cassandra Model
Database	Keyspace
Table	Column Family (CF)
Primary key	Row key
Column name	Column name/key
Column value	Column value

CLUSTER

It is not probably the best choice if you only have one node in Cassandra. Usually a user creates several nodes that will form a ring and uses a peer-to-peer protocol for data replication. So in case of one node going down the node having the data replica responds to the query. The ring is the outer most structure in Cassandra. They call this ring cluster.

KEYSPACES

The key space is the outermost container for data in Cassandra. It resembles the database in relational database. A cluster can hold several key spaces. In Cassandra when you create key space there are set of basic attributes you can set for it as follow:

a) **Replication Factor:** The number of nodes holding the replica of data

b) **Replica Placement Strategy:** The strategy the replication should happen

c) **Column Families:** Like a database having several tables a Keyspace can define many column families

COLUMN

The basic building block in Cassandra is a column. The column is constructed of three parts: column name, column value and a timestamp. The column name and value are simply saved as Java byte arrays. The timestamp is implementation of Cassandra IClock interface. The timestamp is used for internal Cassandra use and is not accessible or editable by the user.

```
{
"name": "email",
"value: "me@example.com",
"timestamp": 1274654183103300
}
```

COLUM FAMILY

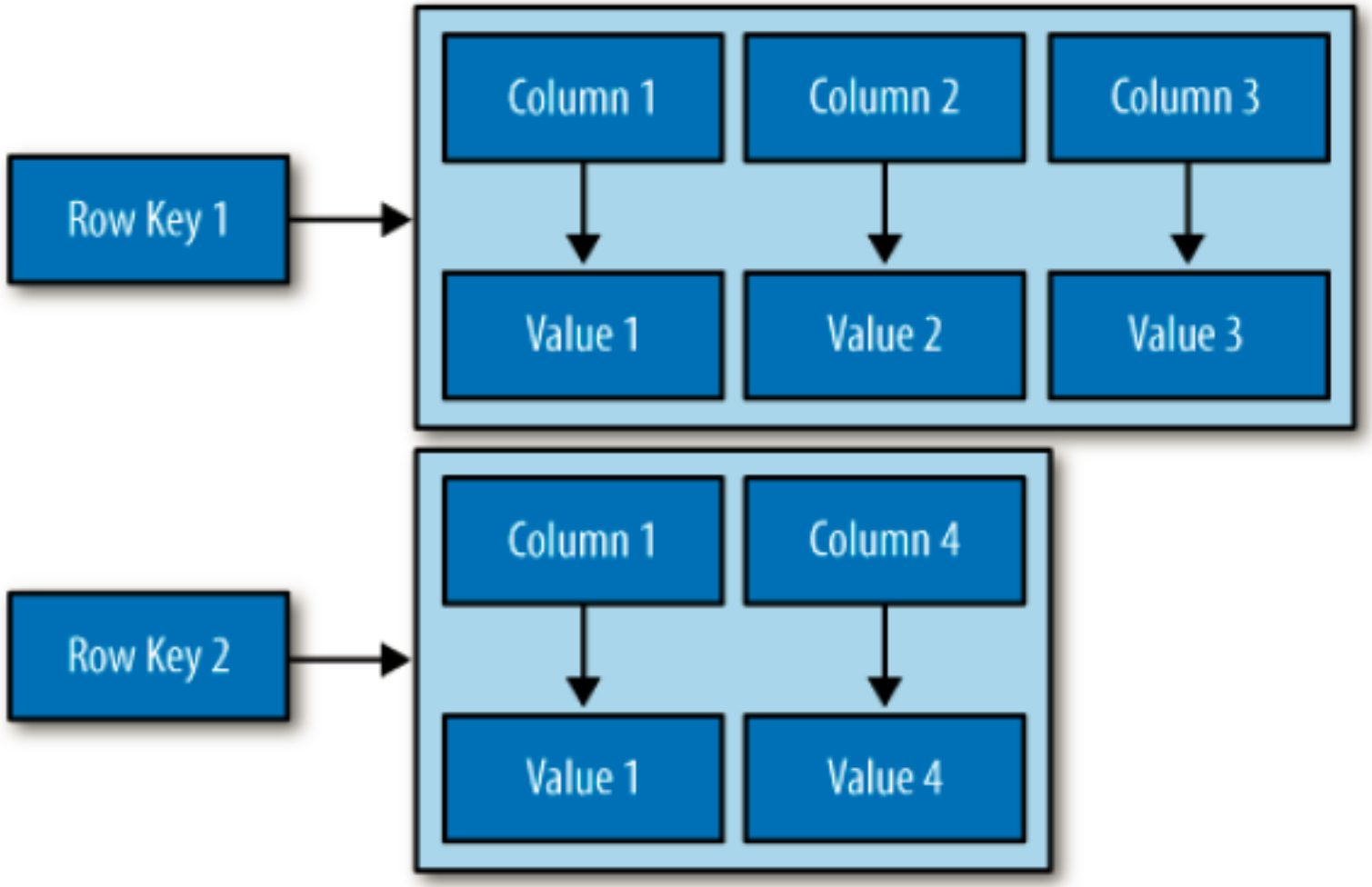
The column family resembles the table in a relational database.

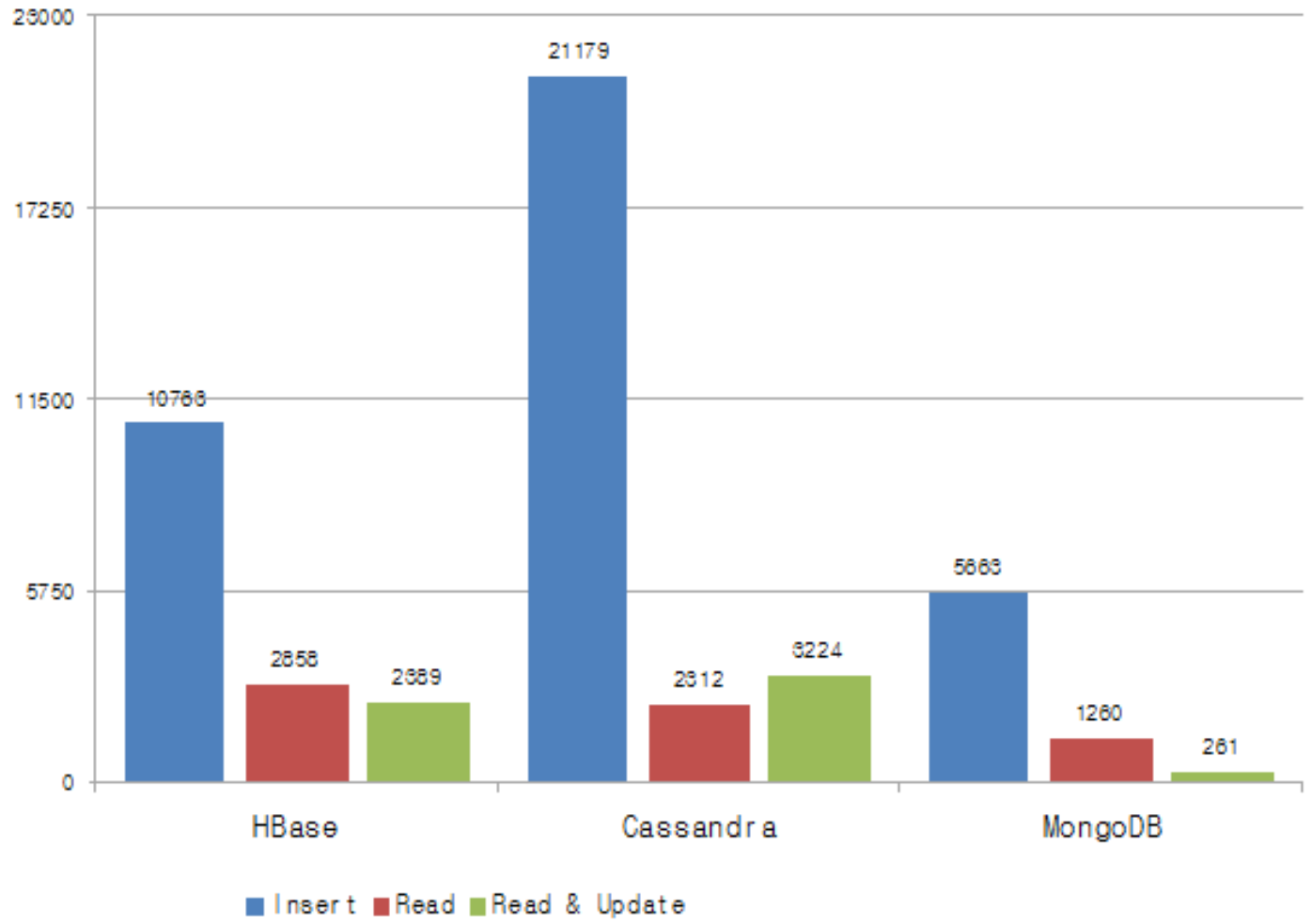
- Cassandra is considered schema free.
- A column family has two attributes: a **name** and a **comparator**.

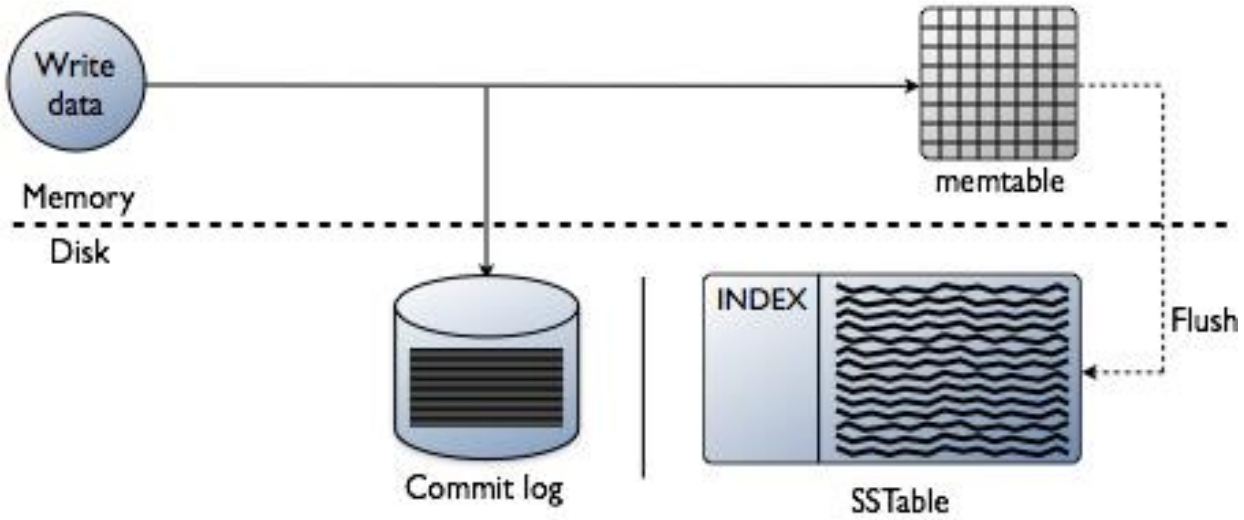
The comparator value indicates how columns will be sorted when they are returned to you in a query–according to long, byte, UTF8, or other ordering.

A collection of these columns constitutes a row in Cassandra. The primary key or unique identifier of each row will be called **row-key**. So a column family will be number of rows and each row can have variable length of columns. In Cassandra the design of many aspects including the data model concentrates on performance so anything that could help better performance has been considered such as variable length of columns.

A COLUMN FAMILY IN CASSANDRA




23000
17250
11500
5750
0
10766
2858
2389
21179
2312
3224
5663
1260
261
HBase
Cassandra
MongoDB
Insert
Read
Read & Update



Pros	Cons
Write Speed	No Ad-Hoc Queries
Multi-DC Replication	No Aggregations
Tunable Consistency	Unpredictable Performance:
JVM Based:	JVM Based
CQL	CQL

REFERENCES

- https://www.json.org/json-en.html
- http://www.json.org/json.pdf
- http://www.csc.kth.se/utbildning/kth/kurser/DD143X/dkand11/Group2Mads/victor.hallberg.malin.eriksson.report.pdf
- http://tinman.cs.gsu.edu/~bjayakumar2/Final/report.pdf
- http://www.oracle.com/technetwork/products/nosqldb/documentation/nosql-vs-cassandra-1961717.pdf
- https://www.datastax.com/wp-content/uploads/2012/08/WP-IntrotoCassandra.pdf
- http://ftp.esrf.eu/pub/cs/tango/meetings/2015_may_solaris/Reynald_Bourtembourg_HDB++_-_High_availability_with_Cassandra.pdf