YCBS 257 - Data at Scale (Winter 2019)

Instructor: Khaled Tannir

McGill | School of Continuing Studies

mcgill.ca
/continuingstudies

School of Continuing Studies

YCBS 257-256 / 257 - Data at Scale (BIG DATA)

Course 6

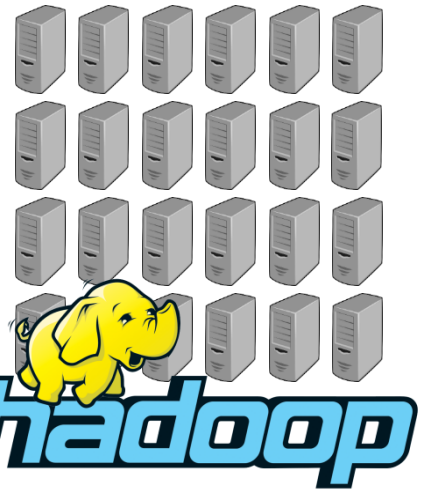
Data Storage in Hadoop

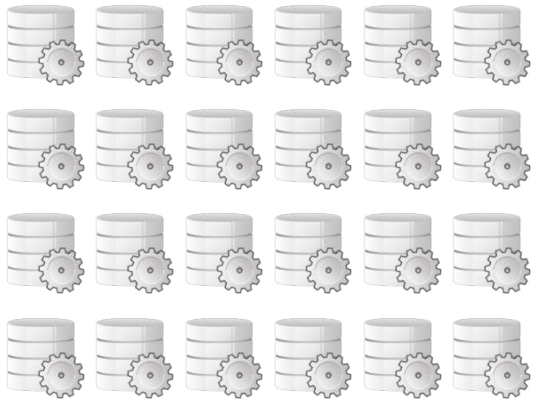
Khaled Tannir

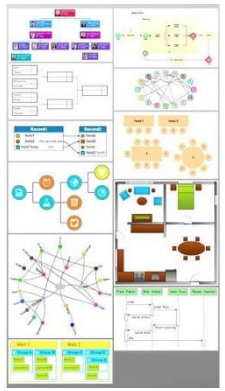
McGill

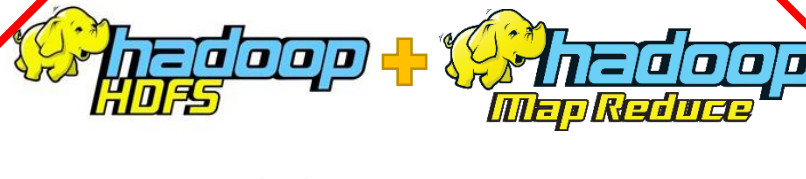
Montréal – Fall 2018

Machine Learning at Scale









Analyze

Ingest

Machine Learning at Scale

Volume

Low Velocity — High Velocity

Distributed Storage

Volume

On Disk — In-Memory

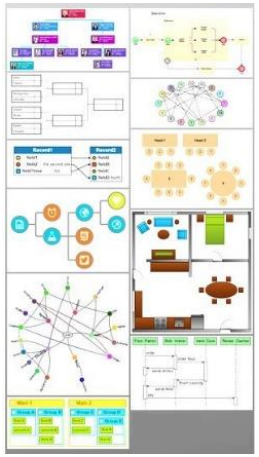
Distributed Processing

Velocity

Low Velocity — High Velocity

Variety

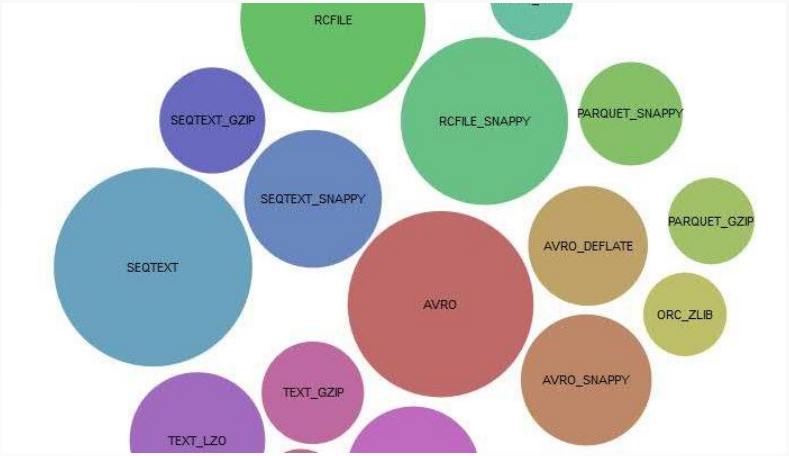
Unstructured — Structured



Theme of this Course

Data Storage in Hadoop

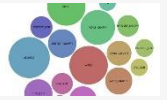
- *Hadoop File Format*
  - *Avro*
  - *Parquet*
- *NoSQL Databases*
  - *Core Concepts, NoSQL databases classifications*
- *Apache HBase*
  - *Core Concepts, Using HBase*



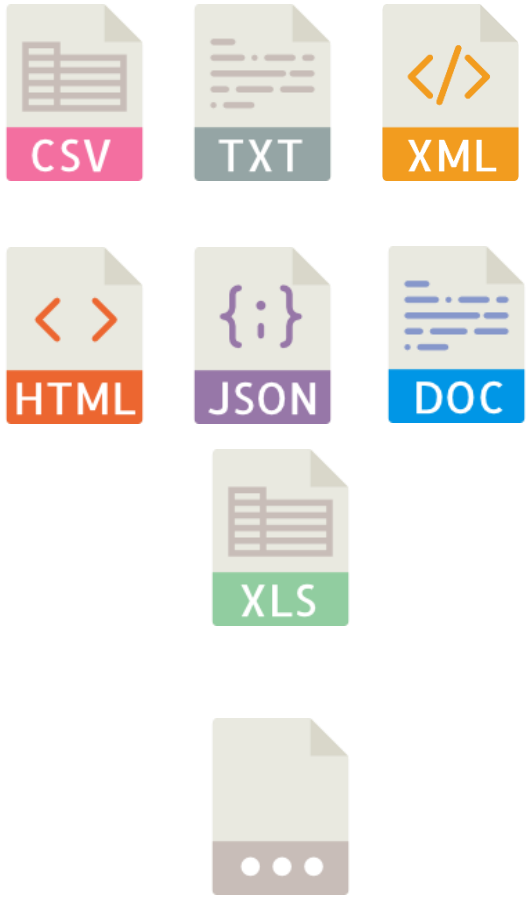
Hadoop File Formats

Concepts and comparisons

Many File Formats



- CSV
- Text
- JSON
- Protobuff
- Thrift
- Sequence File
- Avro

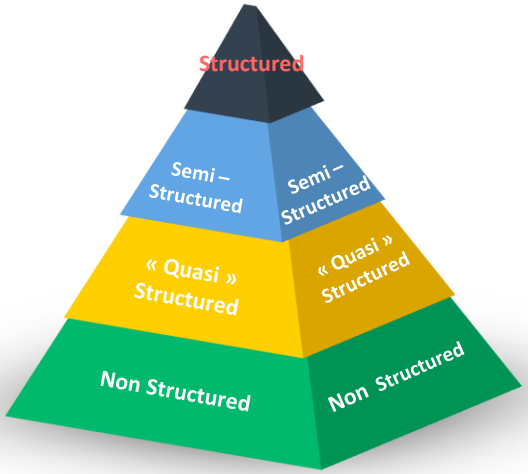


- Parquet
- ORC
- …

What is a 'Good' File Format?

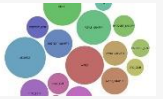
- Well-Defined
- Expressive
- Simple
- Optimized Binary encoding
- Compressed
- Splittable (compatible with HDSF)

- Structured, semi/non-structured
- General usage
- Classified regarding their performances



We should consider also file analysis not only storage

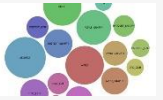
A ‘Good’ File Format (1)



- **Well-Defined**
  - **Reading or analyzing a file should not be interrupted by a missing or miscoded value**

```
movieId,title
55269,Darjeeling Limited, The
```

A 'Good' File Format (2)

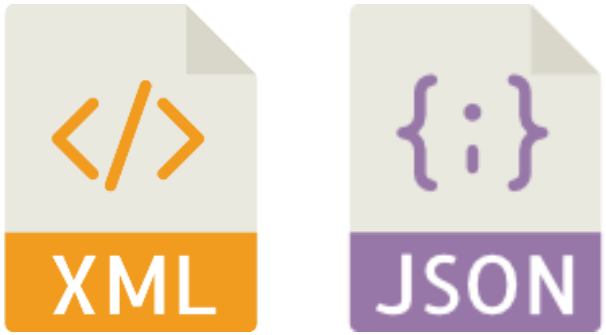


- **Expressive**
  - **Reading data should be as easy as possible**

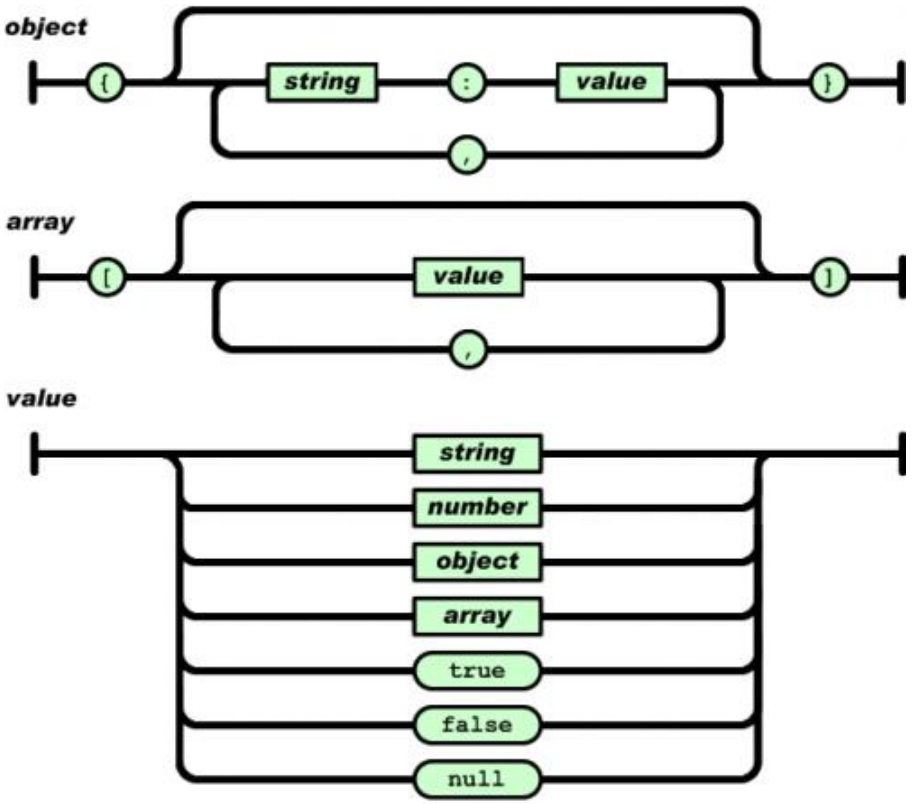
```
movieId,title,genres
94959,Moonrise Kingdom,Comedy|Drama|Romance


{"movieId": 94959, "title": "Moonrise Kingdom",
 "genres": ["Comedy", "Drama", "Romance"]}
```

A ‘Good’ File Format (3)

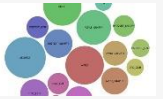
- **Simple**
  - **XML: Well-Formed but not simple (definitions, namespaces, tags, …)**
  - **JSON: is a Well-Formed and Simple**



JSON Definition



A ‘Good’ File Format (4)

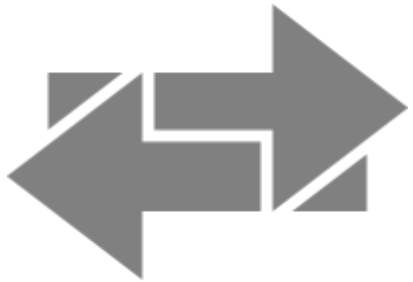


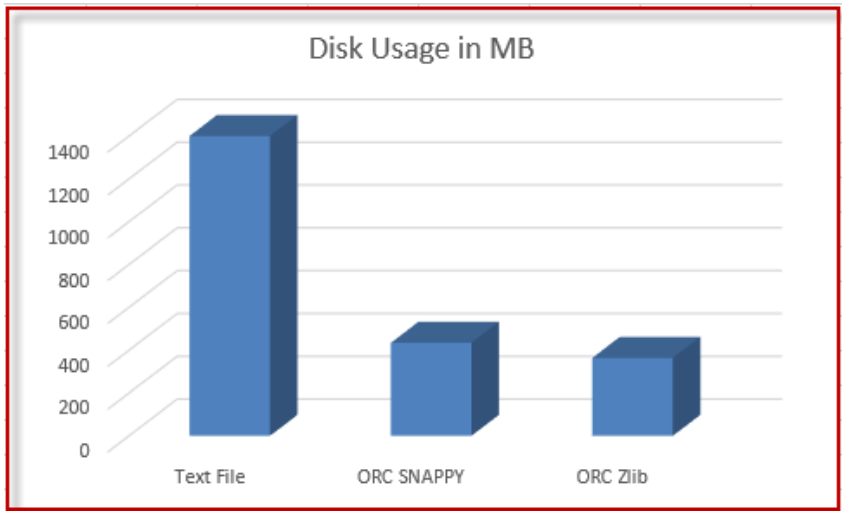
- **Optimized Binary encoding**
  - **Everything should be encoded and written in byte to reduce storage size. (readable by machine not human)**

11034	ASCII - string size – 5 bytes
1a 2b 00 00	Big-endian Int encoding - 4 bytes
9a 56	zig-zag encoding - 2 bytes

A ‘Good’ File Format (5)

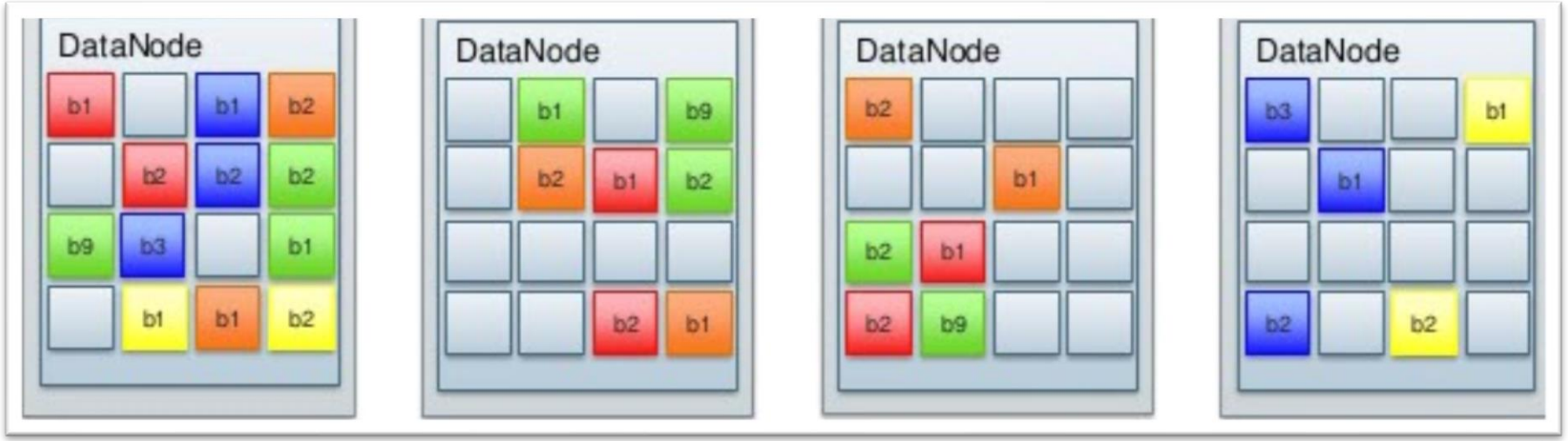
- **Compression**
  - **Reduce the storage size,**
  - **should be also ‘Splitable’ to be Hadoop HDFS compliant**





What is a 'Splittable' Format?

- Capability to process **independently** different chunks of a file
  - A chunk (split) should fit into a HDFS bloc size (usually 64MB or 128MB)
- Allows processing **parallelization**



Best File Format for Hadoop

- **Best file format for Hadoop should be:**

1. **Well-Defined**
2. **Expressive**
3. **Simple**
4. **Optimized Binary enconding**
5. **Compressed**
6. **Splittable**

Parquet

ORC

Files Formats Summary

	Well-Defined	Expressive	Simple	Binary	Compressed	Splittable
CSV	✓	✗	✓	✗	✗	✗
TEXT	✓–	✗	✓	✗	✗	✓
JSON	✓	✓	✓	✗	✗	✗
PROTOBUF	✓	✓–	✓	✓	✓	✗
THRIFT	✓	✓–	✓	✓	✓	✗
SEQ FILES	✗	–	✗	✓	✓	✓
AVRO	✓	✓	✓	✓	✓	✓
Parquet	✓	✓	✗	✓+	✓	✓

AVRO File

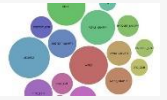
	Well-Defined	Expressive	Simple	Binary	Compressed	Splittable
AVRO	✓	✓	✓	✓	✓	✓

- A good file format for Hadoop
  - Row Oriented
  - Need a Schema
  - Flexible object moel


```
{
  "namespace": "customer.avro",
  "type": "record",
  "name": "customer",
  "fields": [
     {"name": "name",    "type": "string"},
     {"name": "age",     "type": "int"},
     {"name": "address", "type": "string"},
     {"name": "phone",   "type": "string"},
     {"name": "email",   "type": "string"},
  ]
}
```

Avro schema example

AVRO Benefits

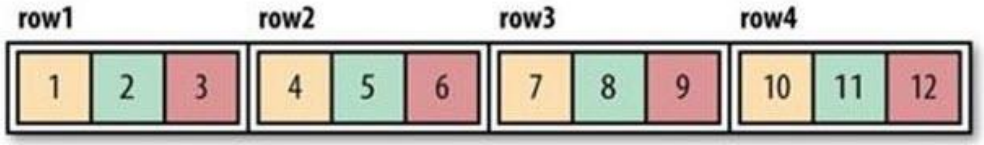


- Write and forget
- Low memory usage
- Ensure that the data is written (especially in case of failure)

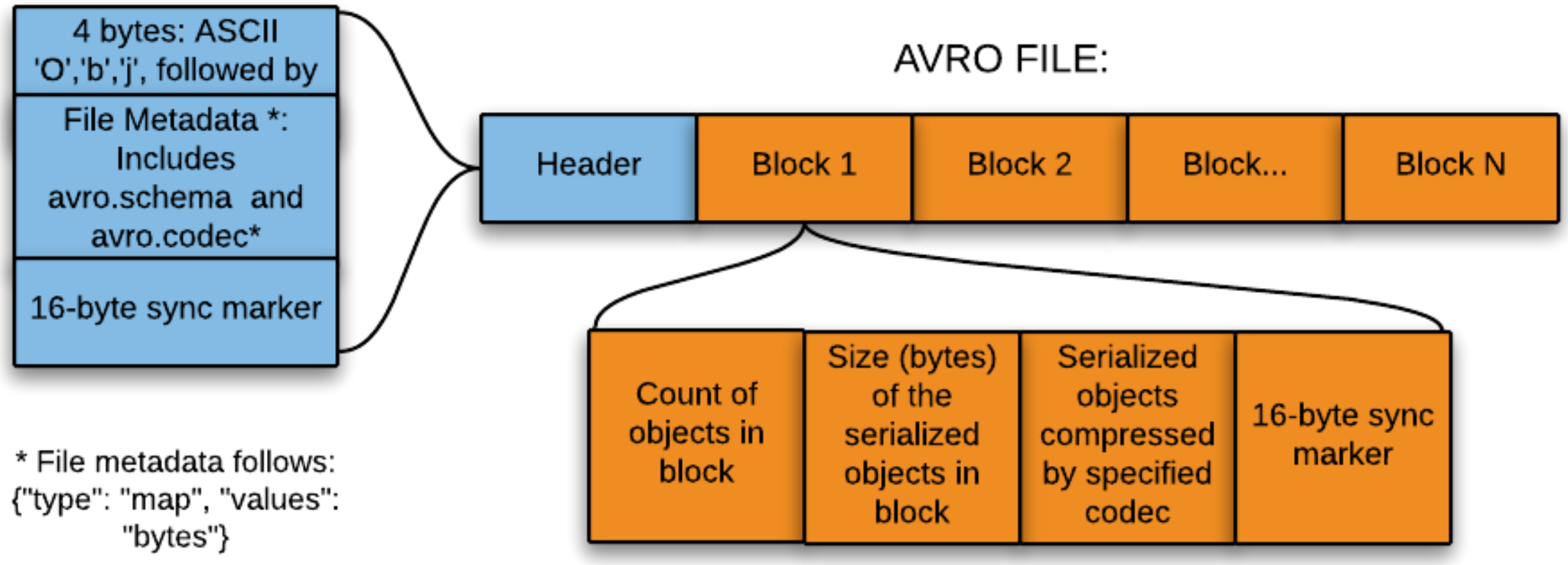
Logical table

	col1	col2	col3
row1	1	2	3
row2	4	5	6
row3	7	8	9
row4	10	11	12

Row-oriented layout (SequenceFile)



AVRO file Structure



AVRO Schema

- **Needed** to decode data
- Stored in the file header
- Schema is in JSON format and support different types:
  - Primitives
    (*boolean, int, long double, string, fixed, binary*)
  - Unions
    (with ***null*** for optional data)
  - Records
    (for complexes objects)
  - Maps and Lists

```
{
  "type" : "record",
  "name" : "tweets",
  "fields" : [ {
    "name" : "username",
    "type" : "string",
}, {
    "name" : "tweet",
    "type" : "string",
}, {
    "name" : "timestamp",
    "type" : "long",
} ],
  "doc:" : "schema for storing tweets"
}
```

AVRO Toolbox

- avro-tools (Apache)
- kite-sdk (Cloudera)
- …

```
[cloudera@quickstart ml-20m]$ avro-tools
Version 1.7.6-cdh5.8.0 of Apache Avro
Copyright 2010 The Apache Software Foundation

C JSON parsing provided by Jansson and
written by Petri Lehtinen. The original software is
available from http://www.digip.org/jansson/.
----------------
Available tools:
          cat  extracts samples from files
      compile  Generates Java code for the given schema.
       concat  Concatenates avro files without re-compressing.
   fragtojson  Renders a binary-encoded Avro datum as JSON.
     fromjson  Reads JSON records and writes an Avro data file.
     fromtext  Imports a text file into an avro data file.
      getmeta  Prints out the metadata of an Avro data file.
    getschema  Prints out schema of an Avro data file.
          idl  Generates a JSON schema from an Avro IDL file
 idl2schemata  Extract JSON schemata of the types from an Avro IDL file
       induce  Induce schema/protocol from Java class/interface via reflection.
   jsontofrag  Renders a JSON-encoded Avro datum as binary.
       random  Creates a file with randomly generated instances of a schema.
      recodec  Alters the codec of a data file.
       repair  Recovers data from a corrupt Avro Data file
  rpcprotocol  Output the protocol of a RPC service
   rpcreceive  Opens an RPC Server and listens for one message.
      rpcsend  Sends a single RPC message.
       tether  Run a tethered mapreduce job.
       tojson  Dumps an Avro data file as JSON, record per line or pretty.
       totext  Converts an Avro data file to a text file.
     totrevni  Converts an Avro data file to a Trevni file.
  trevni_meta  Dumps a Trevni file's metadata as JSON.
trevni_random  Create a Trevni file filled with random instances of a schema.
trevni_tojson  Dumps a Trevni file as JSON.
[cloudera@quickstart ml-20m]$
```

Parquet File

	Well-Defined	Expressive	Simple	Binary	Compressed	Splittable
Parquet	✓	✓	✗	✓+	✓	✓

- A good File Format for Hadoop
  - Column oriented
  - Encode a group of values rather than one value (simplicity ☹)
  - Unlike ORC, Parquet can be used outside Hive

Column Oriented File

- Need to ‘wait’ all the rows to finalize a column
- Memorize rows into groups
- High memory usage
- In case of failure data is not guarantied to be written

Parquet

Table

A	B	C
A1	B1	C1
A2	B2	C2
A3	B3	C3

column-oriented layout

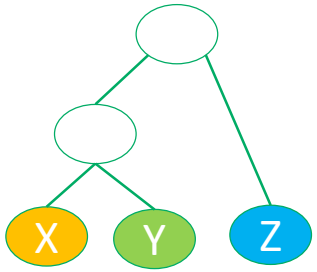
A1	A2	A3	B1	B2	B3	C1	C2	C3

Parquet Benefits

- Very fast reading
  - Read selective columns
- Small encoding data size (reduce storage volume)
- Better data compression

Parquet

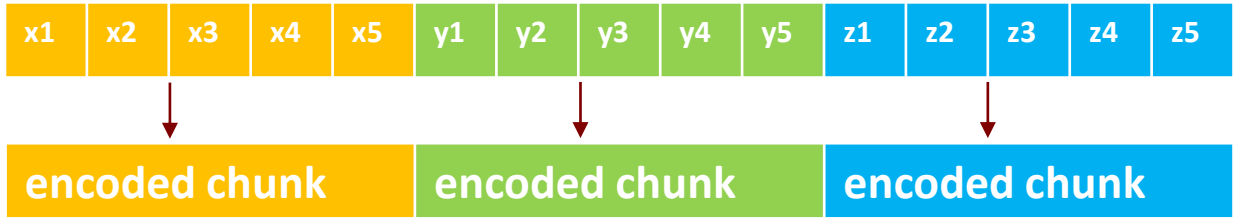
Embedded Schema



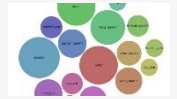
Table

X	Y	Z
x1	y1	z1
x2	y2	z2
x3	y3	z3
x4	y4	z4
x5	Y5	z5

Column Format

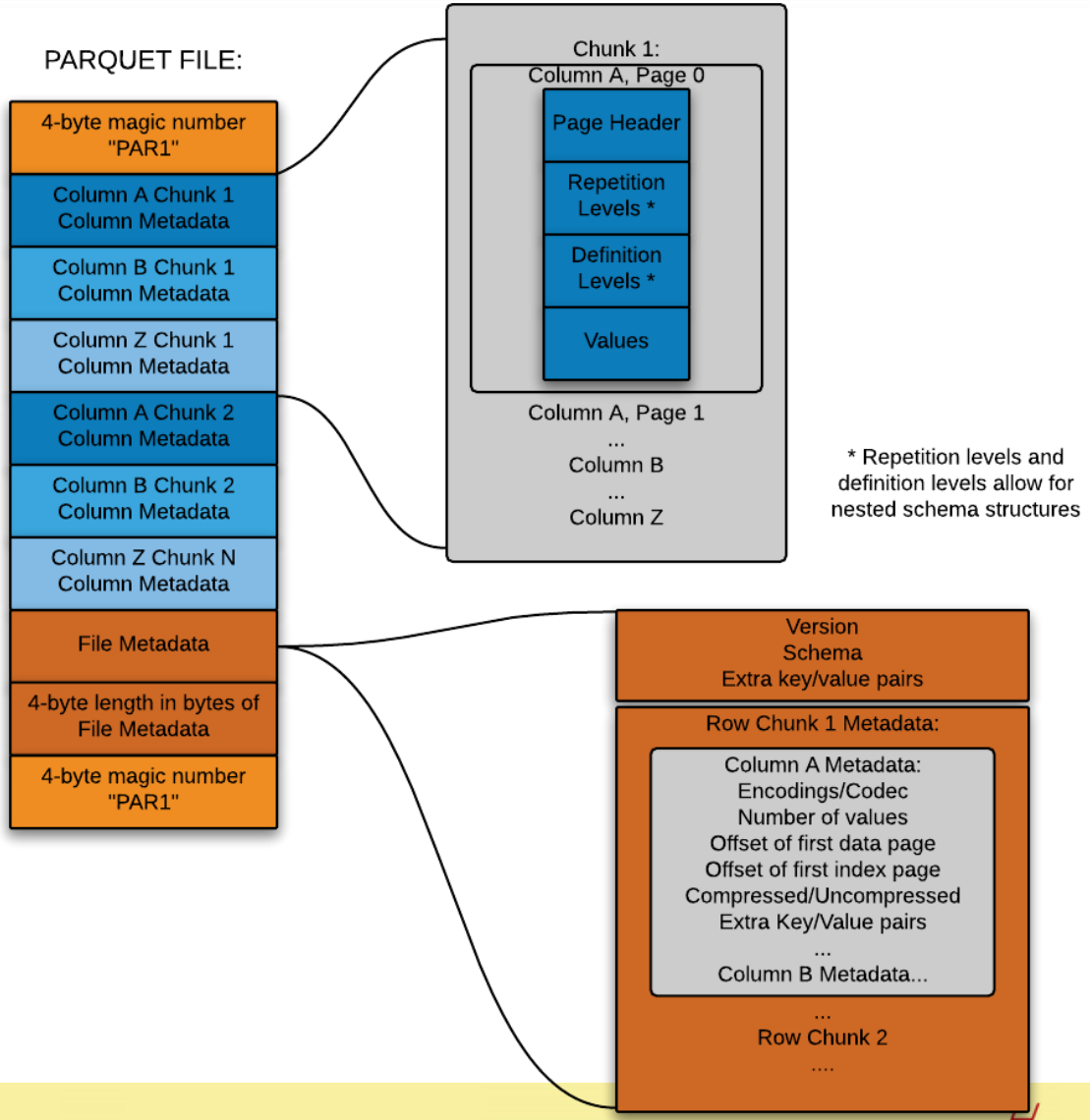


Parquet File Structure



- Data is splitted into blocs (chunks)
- Bloc size is usually based on the HDFS bloc size
- Accumulates encoded data
- No file markers as in Avro

  No wasted space



Parquet Toolbox

- kite-sdk (cloudera)
- parquet-tools (Apache)
- …

```
[cloudera@quickstart ~]$ parquet-tools -h
usage: parquet-tools cat [option...] <input>
where option is one of:
       --debug     Enable debug output
    -h,--help      Show this help string
    -j,--json      Show records in JSON format.
       --no-color  Disable color output even if supported
where <input> is the parquet file to print to stdout

usage: parquet-tools head [option...] <input>
where option is one of:
       --debug          Enable debug output
    -h,--help           Show this help string
    -n,--records <arg>  The number of records to show (default: 5)
       --no-color       Disable color output even if supported
where <input> is the parquet file to print to stdout

usage: parquet-tools schema [option...] <input>
where option is one of:
    -d,--detailed  Show detailed information about the schema.
       --debug     Enable debug output
    -h,--help      Show this help string
       --no-color  Disable color output even if supported
where <input> is the parquet file containing the schema to show

usage: parquet-tools meta [option...] <input>
where option is one of:
       --debug     Enable debug output
    -h,--help      Show this help string
       --no-color  Disable color output even if supported
where <input> is the parquet file to print to stdout

usage: parquet-tools dump [option...] <input>
where option is one of:
    -c,--column <arg>  Dump only the given column, can be specified more than
                       once
    -d,--disable-data  Do not dump column data
       --debug         Enable debug output
    -h,--help          Show this help string
    -m,--disable-meta  Do not dump row group and page metadata
       --no-color      Disable color output even if supported
where <input> is the parquet file to print to stdout
[cloudera@quickstart ~]$
```

NoSQL Databases

Not Only SQL Databases

McGill

FALL 2018

What Is NoSQL

- Stands for Not Only SQL.
- Class of non-relational data storage systems.
- Do not require a fixed table schema nor do they use the concept of joins.
- Do not use the SQL Language to query data.
- Massive read/write performance; availability via horizontal scaling

RDBMS vs NoSQL

Not Only SQL

RDBMS

- Structured and organized data
- Structured Query Language (SQL)
- Data and its relationships stored in separate tables.
- Data Manipulation Language, Data Definition Language
- Tight Consistency
- BASE Transaction

NoSQL

- No declarative query language
- No predefined schema
- Different storage types
- Eventual consistency rather ACID property
- Unstructured and unpredictable data
- CAP Theorem
- Prioritize high performance, high availability and scalability

NoSQL vs. MySQL

Not Only SQL

> 50 GB Data

- Writes Average: ~300 ms
- Reads Average: ~350 ms

> 50 GB Data

- Writes Average: 0.12 ms
- Reads Average: 15 ms

SQL vs NoSQL Queries

Not Only SQL

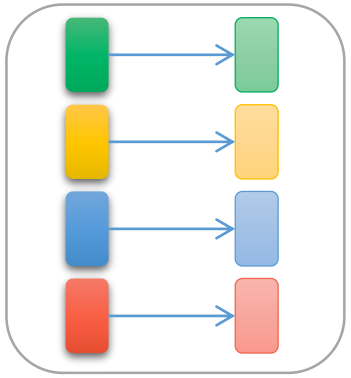
SQL Query:

```
SELECT _id, name, address   <-- projection
FROM   users                <-- table
WHERE  age > 18             <-- select criteria
LIMIT  5                    <-- cursor modifier
```

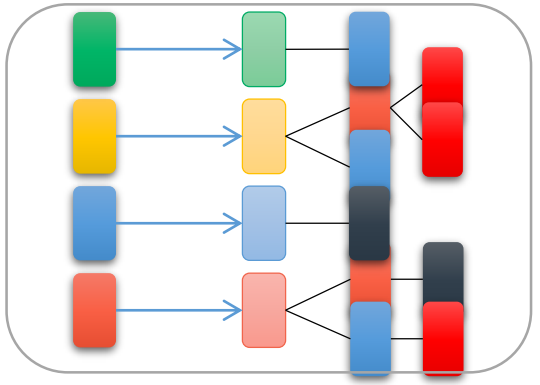
NoSQL Query:

```
db.users.find(
    { age: { $gt: 18 } },   <-- query criteria
    { name: 1, address: 1 } <-- projection
).limit(5)                  <-- cursor modifier
```

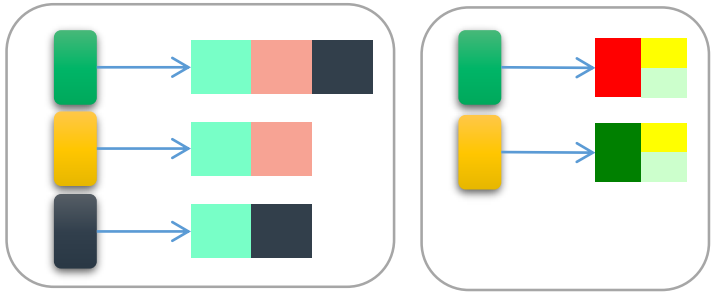
NoSQL Types



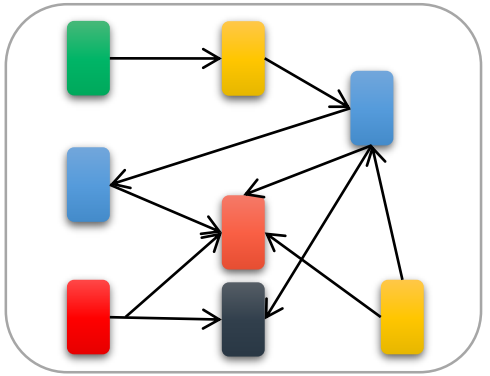
Key/Value Databases



Document Databases



Column-Family Databases



Graph Databases

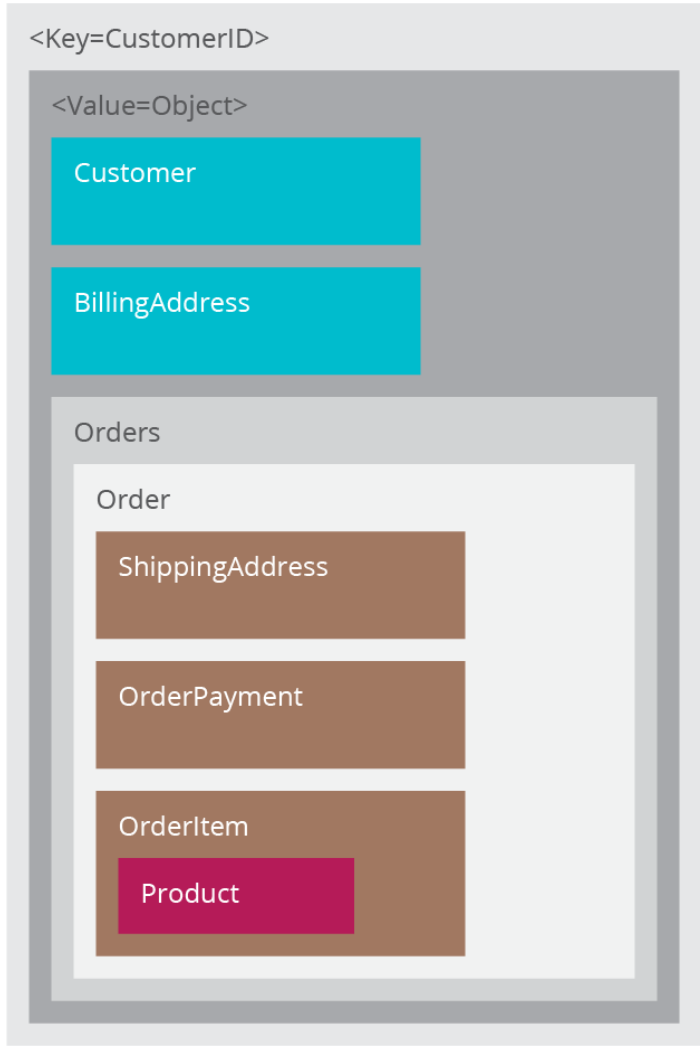
Key Value Pair Based

- **Designed for processing data as a dictionary.**

  Dictionaries contain a collection of records having fields containing data.

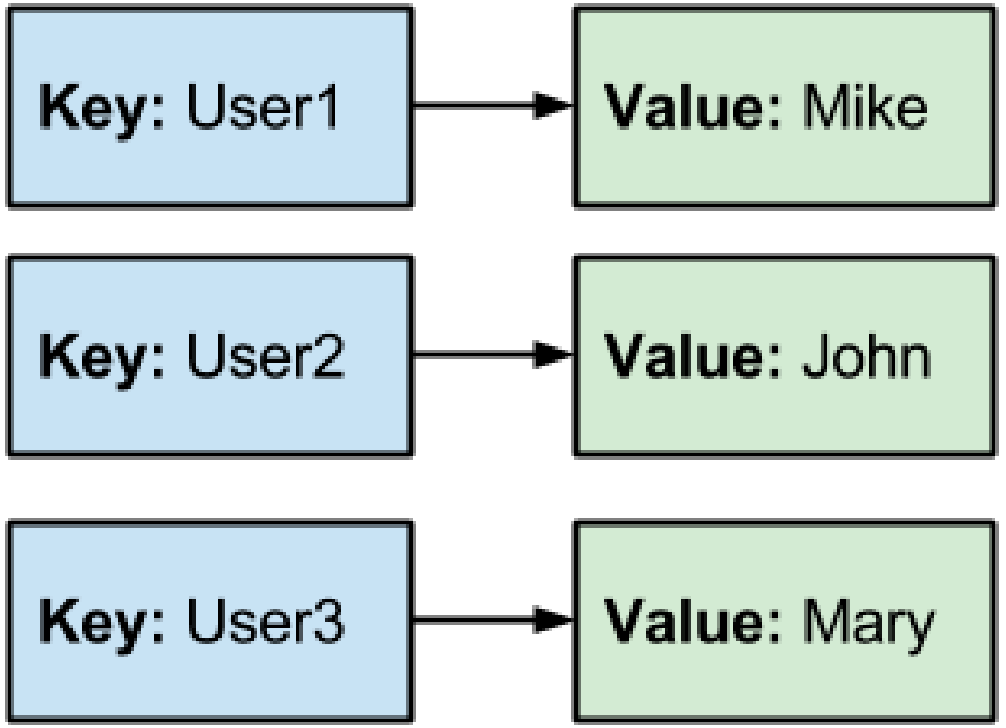
- **Records are stored and retrieved using a Key that uniquely identifies the record,**

- **The Key is used to quickly find the data within the database**



Key-Value Pair Example

- Key-value stores are most basic types of NoSQL databases.
- Designed to handle huge amounts of data.
- Key value stores allow developer to store schema-less data.
- In the key-value storage, database stores data as hash table where each key is unique and the value can be string, JSON, BLOB (basic large object) etc.
- A key may be strings, hashes, lists, sets, sorted sets and values are stored against these keys.
- Key-Value stores can be used as collections, dictionaries, associative arrays etc.



Document Based

- **The database stores and retrieves documents. It stores documents in the value part of the key-value store.**
- **Self-describing, hierarchical tree data structures consisting of maps, collections, and scalar values.**
- **Documents are schema free.**

Different documents can have structures and schema that differ from one another.

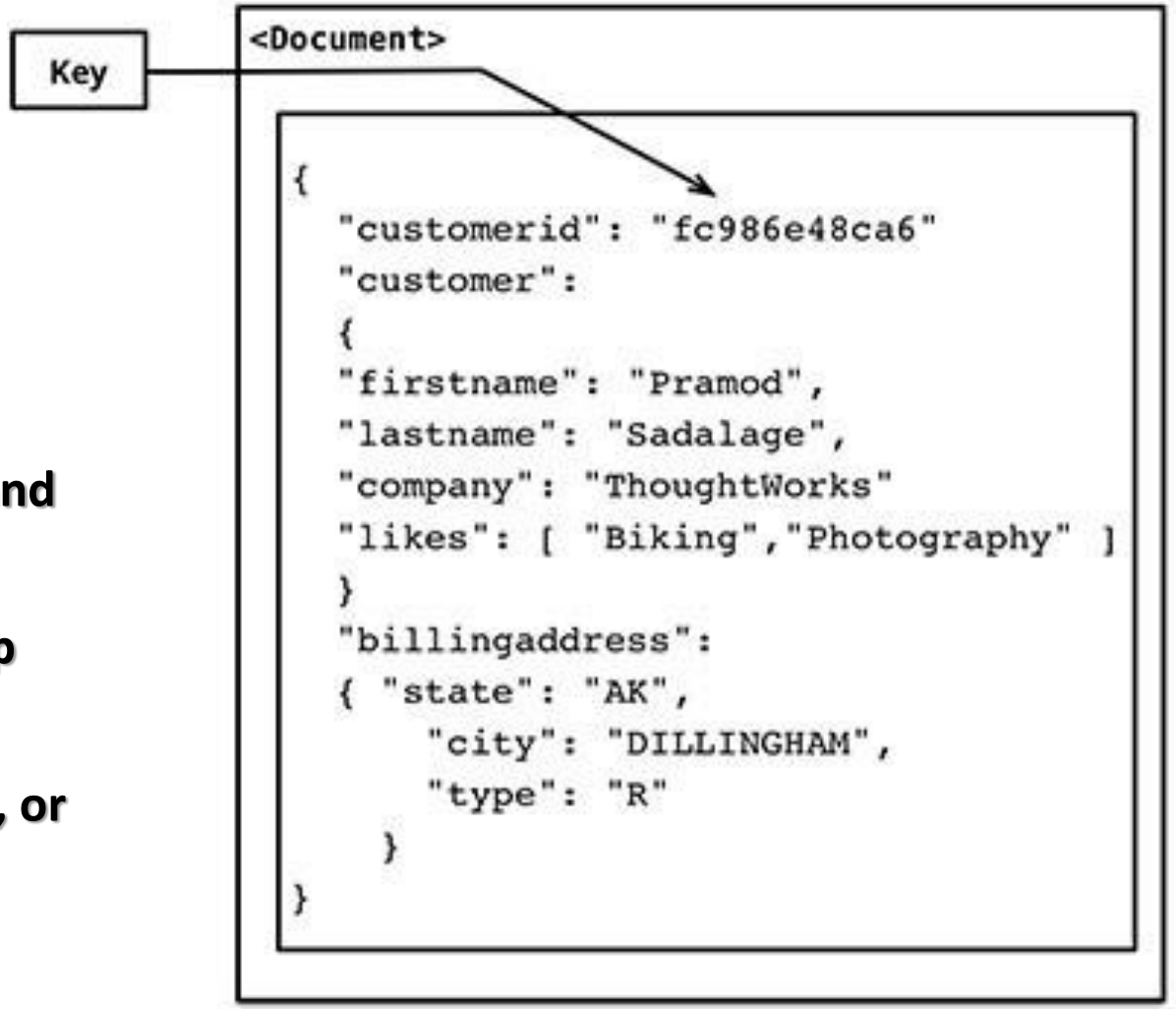
<Key=CustomerID>

```
{
   "customerid": "fc986e48ca6"
   "customer":
   {
   "firstname": "Pramod",
   "lastname": "Sadalage",
   "company": "ThoughtWorks",
   "likes": [ "Biking","Photography" ]
   }
   "billingaddress":
   { "state": "AK",
      "city": "DILLINGHAM",
      "type": "R"
    }
}
```

Key

Documents Based Example

- A collection of documents
- Data in this model is stored inside documents.
- A document is a key value collection where the key allows access to its value.
- Documents are not typically forced to have a schema and therefore are flexible and easy to change.
- Documents are stored into collections in order to group different kinds of data.
- Documents can contain many different key-value pairs, or key-array pairs, or even nested documents.

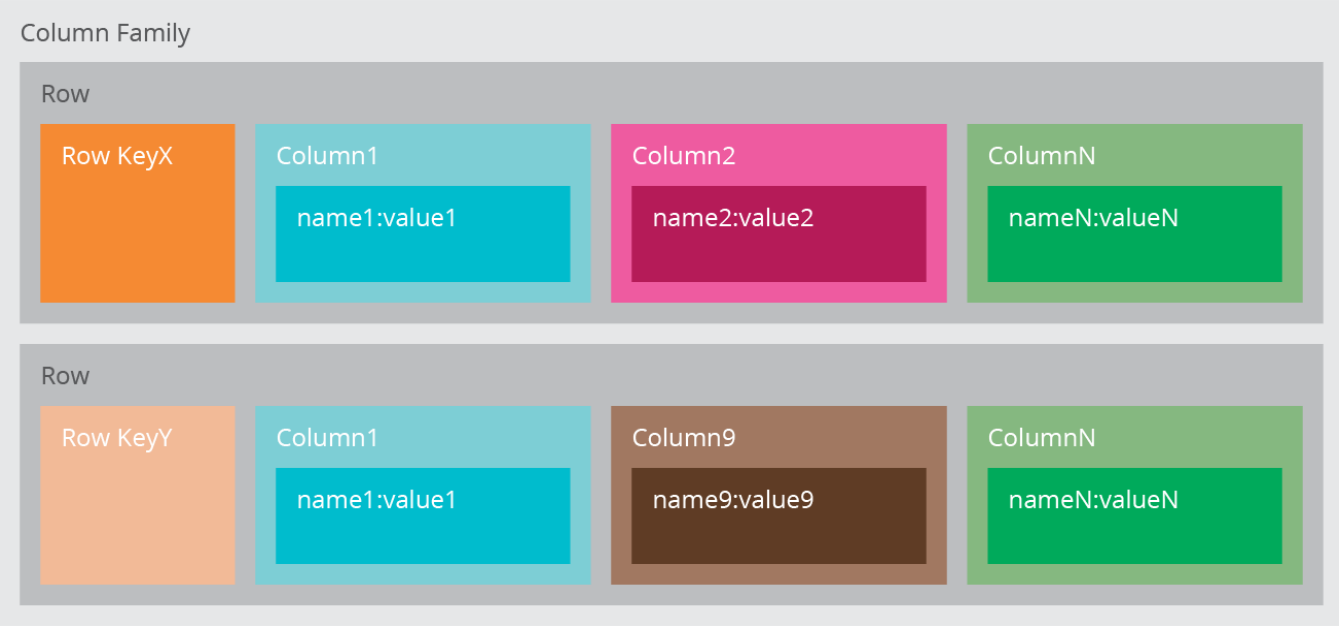


Column Based

- **It store data as Column families containing rows that have many columns associated with a rowkey.**

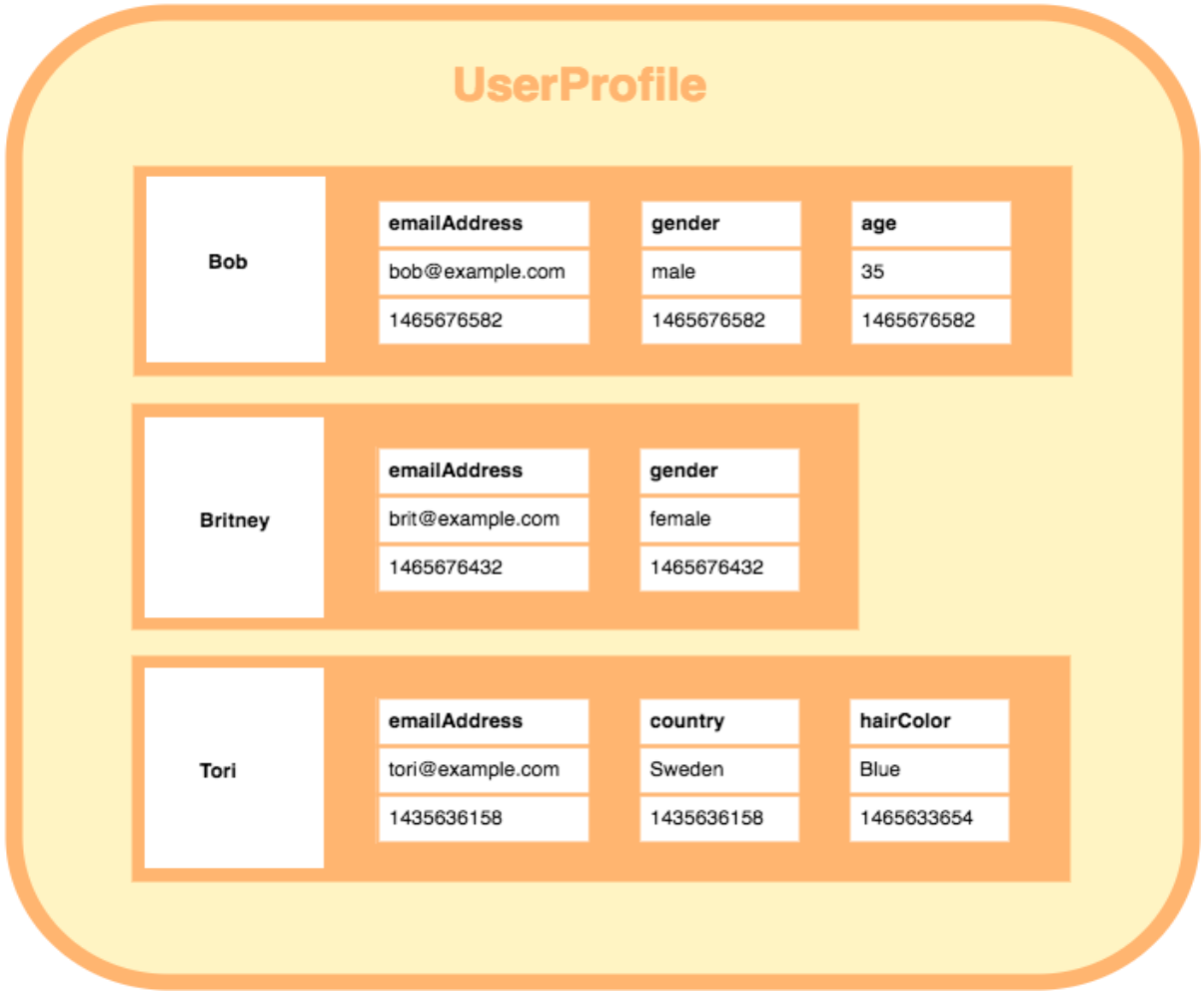
  Each row can have different columns.

- **Column families are groups of related data that is accessed together.**



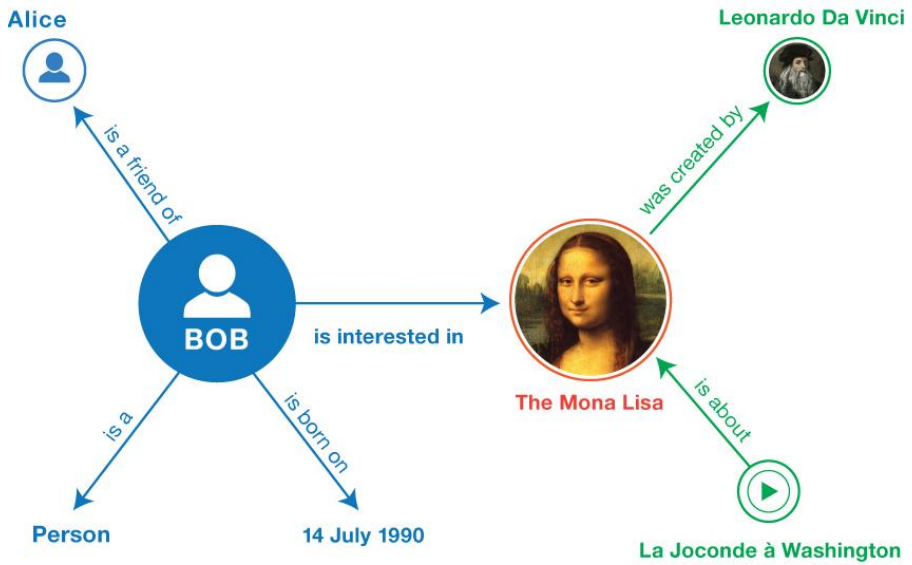
Columns Based Example

- Column-oriented databases primarily work on columns and every column is treated individually.
- Values of a single column are stored contiguously.
- Column stores data in column specific files.
- All data within each column datafile have the same type which makes it ideal for compression.
- Column stores can improve the performance of queries as it can access specific column data.
- High performance on aggregation queries (e.g. COUNT, SUM, AVG, MIN, MAX).



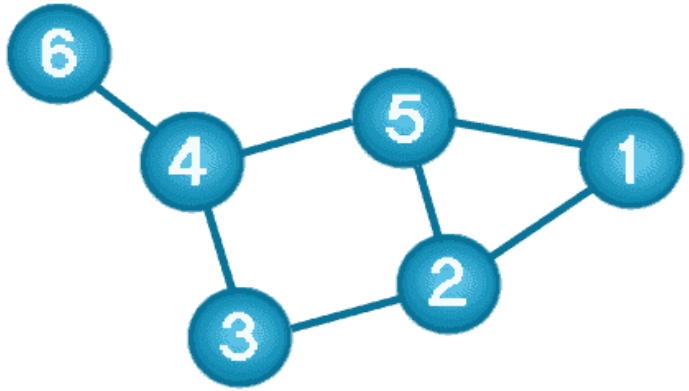
Graph Based

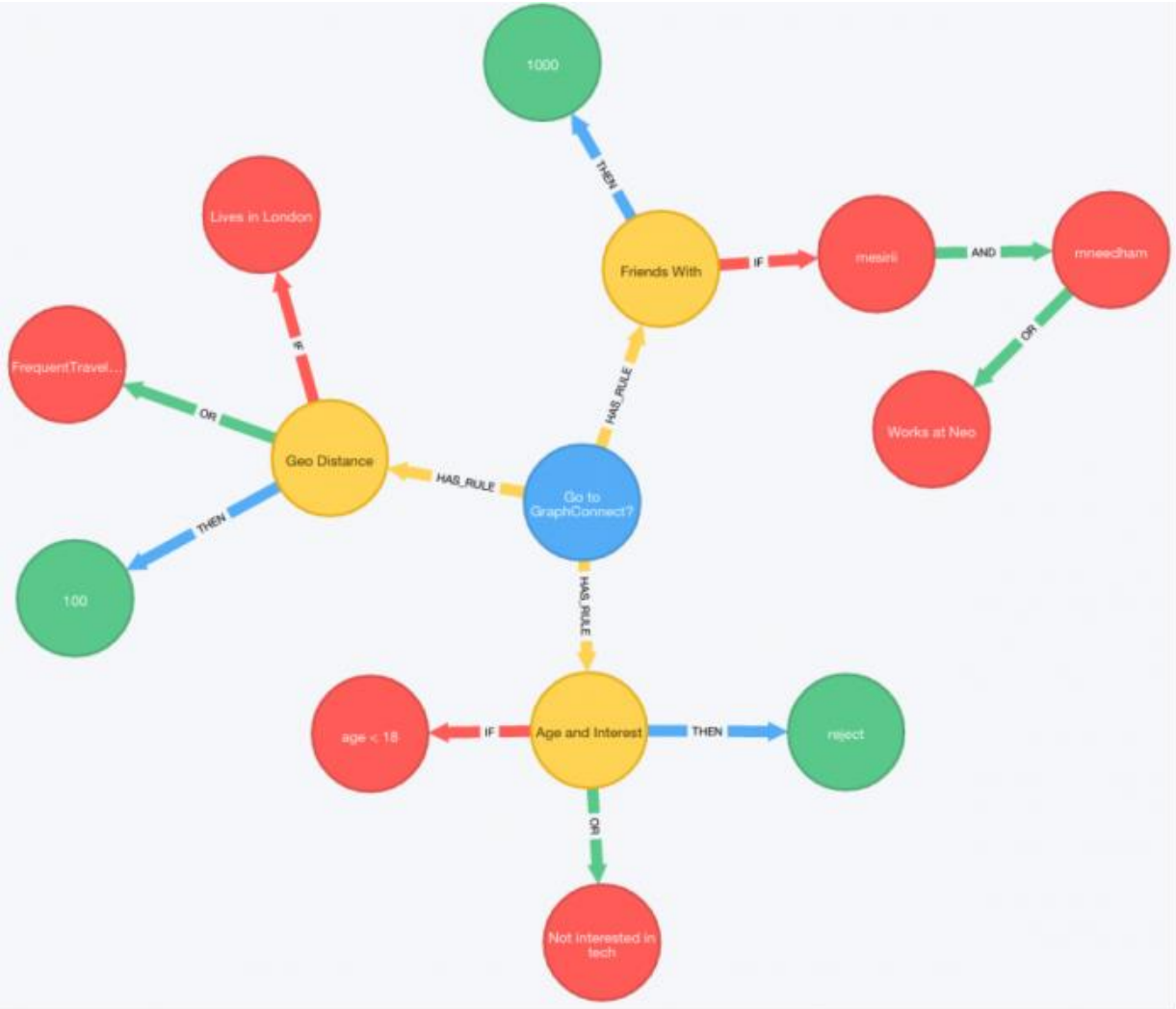
- Store entities and relationships between these entities as nodes and edges of a graph respectively. Entities have properties.
- Traversing the relationships is very fast as relationship between nodes is not calculated at query time but is actually persisted as a relationship.



Graph databases

- A graph data structure consists of a finite set of ordered pairs, called **edges** or **arcs**, of certain entities called **nodes** or **vertices**.
- Following illustration presents a labelled graph of 6 vertices and 7 edges.





NoSQL types Comparison

Not Only SQL

	Document	Graph	Column	Key:Value
Performance	high	variable	high	high
Scalability	high	variable	high	high
Complexity	low	high	low	none
Flexibility	high	high	so-so	high

CAP Theorem

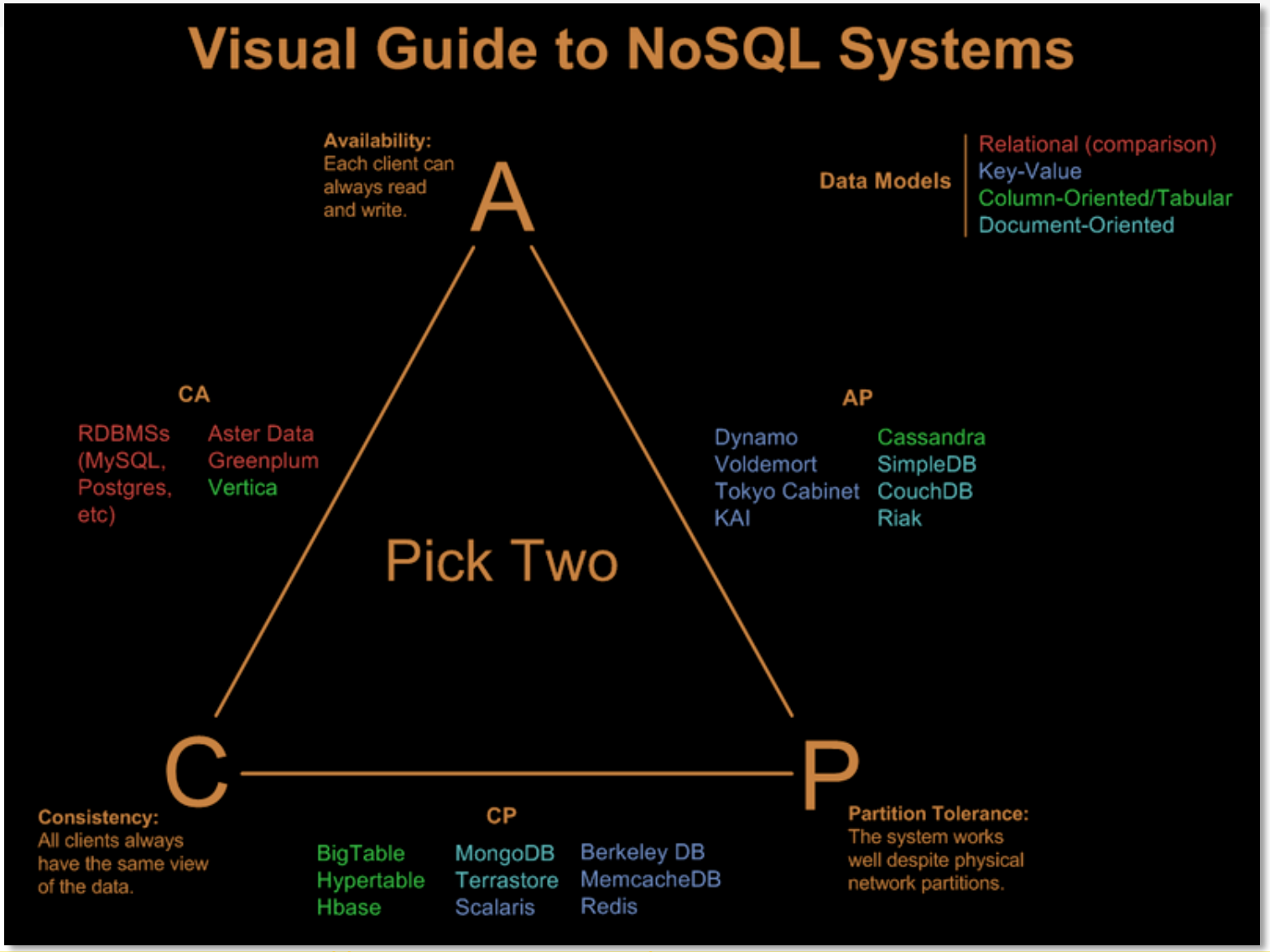
NOT ONLY SQL

- **According to Eric Brewer a distributed system has 3 properties :**
  - **Consistency**
    every read (to any node) gets a response that reflects the most recent version of the data
  - **Availability**
    every request (to a living node) gets an answer: set succeeds, get retunes a value
  - **Partition tolerance**
    service continues to function on network failures

- **We can have at most two of these three properties for any shared-data system**

- **To scale out, we have to Partition. It leaves a choice between Consistency and Availability. (In almost all cases, we would choose Availability over Consistency)**

NoSQL and CAP Theorem



2010 visual by Nathan Hurst

http://blog.nahurst.com/visual-guide-to-nosql-systems

Apache HBase

The Hadoop NoSQL Database

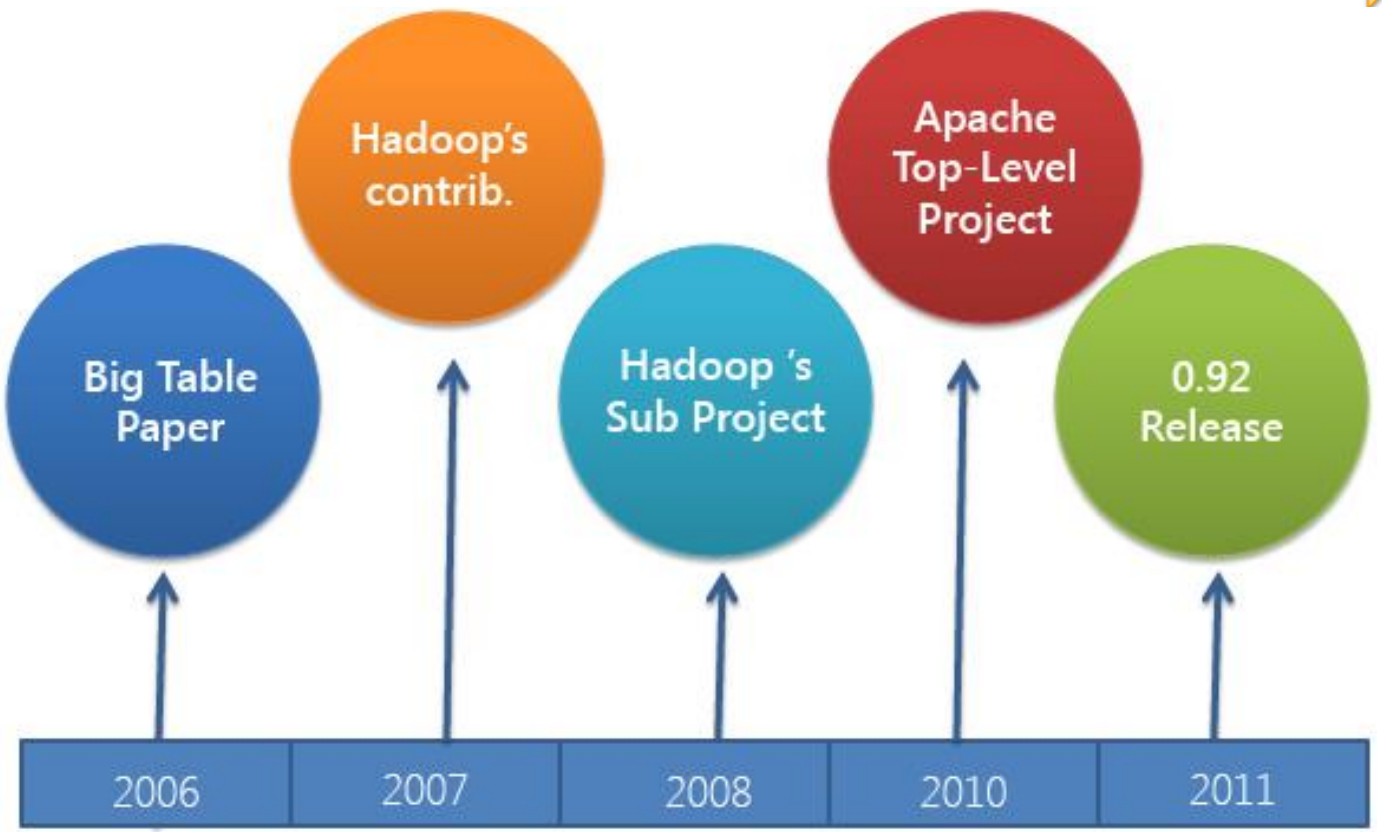
McGill

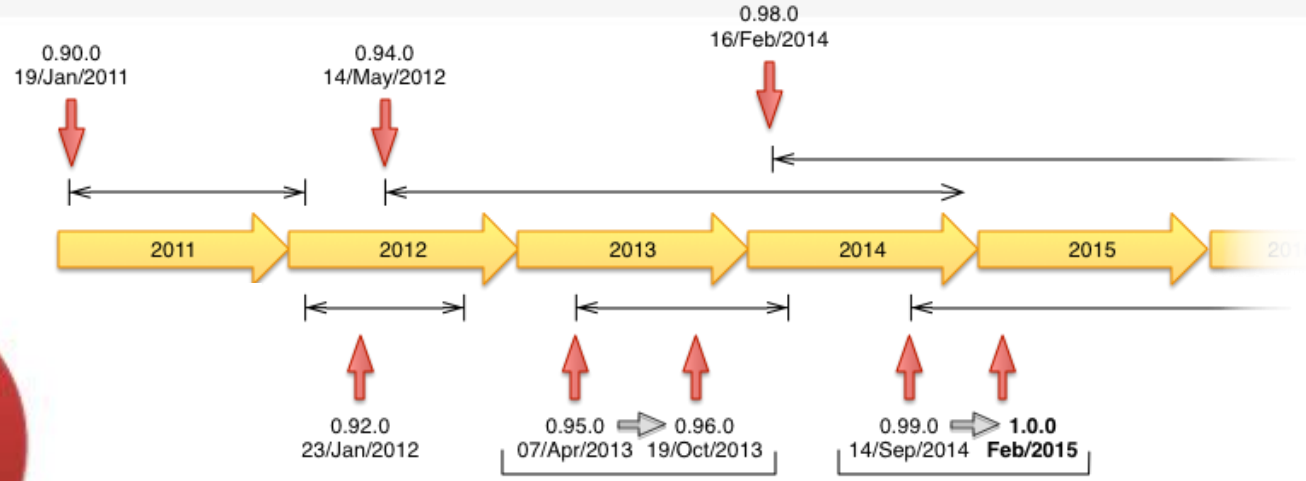
FALL 2018

What is Apache HBase?

- NoSQL Distributed, Column-Oriented database
- Open source Apache project based on BigTable!
- Built on top of HDFS
- Designed for large data volume (terabytes to petabytes).
- Low latency random read / write to HDFS.
- Consistent and Partition tolerant
- Written in Java
- Runs on commodity hardware

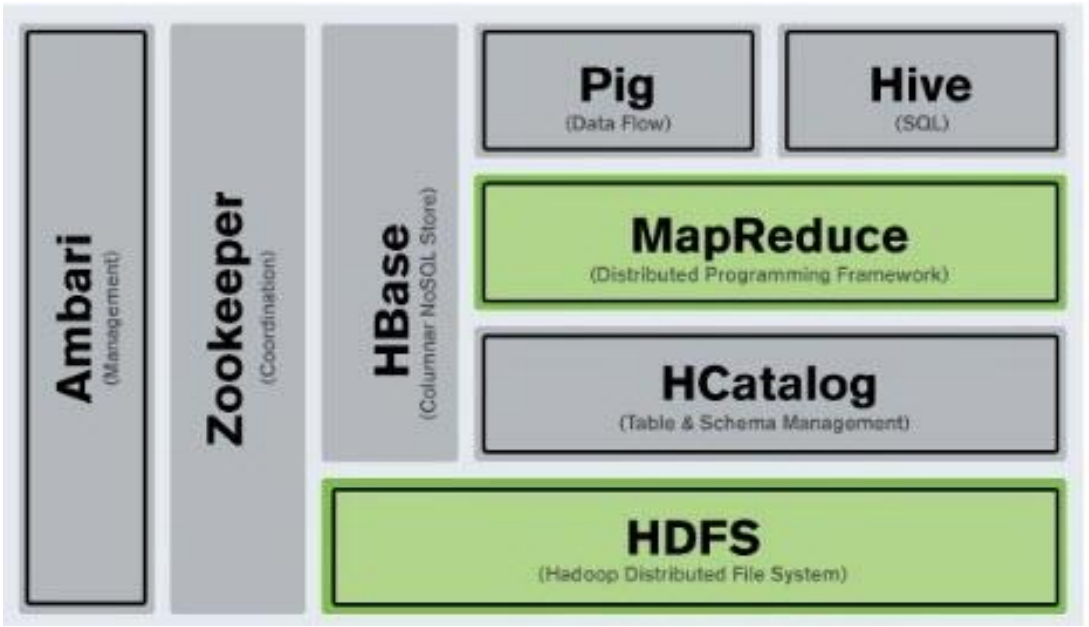
HBase History





HBase main components

HBase = HDFS (Storage) + Processing



Apache HBase

- **Rely on Hadoop:**
  - **Data storage using HDFS**
  - **Parallelize processing using MapReduce**

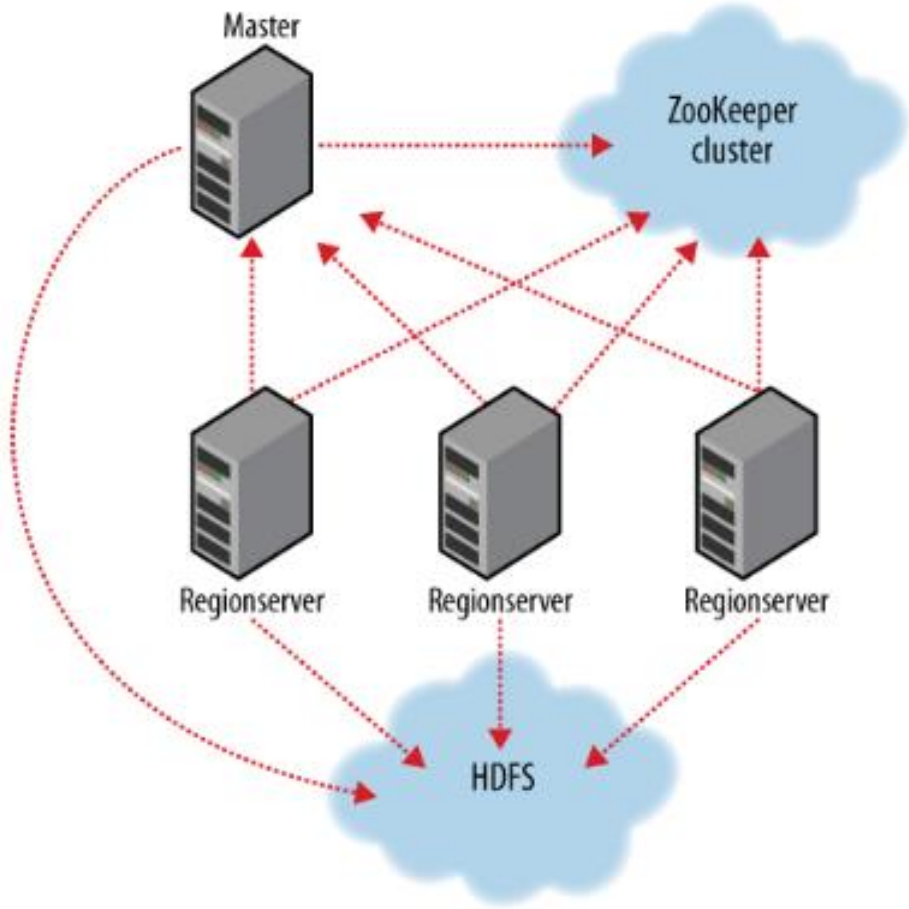
- **Depends on Zookeeper**
  - **Ensure availability**

    synchronization, master server election, server availability checking, …
  - **Manages a ZooKeeper instance as the authority on cluster state**

- **Use *Block Cache* and *Bloom Filters* to optimize queries**

HBase High Level Architecture

- **HBase Master**
  - One per cluster
  - Responsible for monitoring region servers
  - Load balancing for regions
  - Redirect client to correct region servers
  - The current SPOF



- **RegionServer(s)**
  - Many per cluster
  - Serving requests (Write/Read/Scan) of Client
  - Send HeartBeat to Master
  - Throughput and Region numbers are scalable by region servers

HBase – Logical View of Data

RDBMS View

ID (pk)	First Name	Last Name	tweet	Timestamp
1234	John	Smith	hello	20130710
5678	Joe	Brown	xyz	20120825
5678	Joe	Brown	zzz	20130916

Logical HBase View

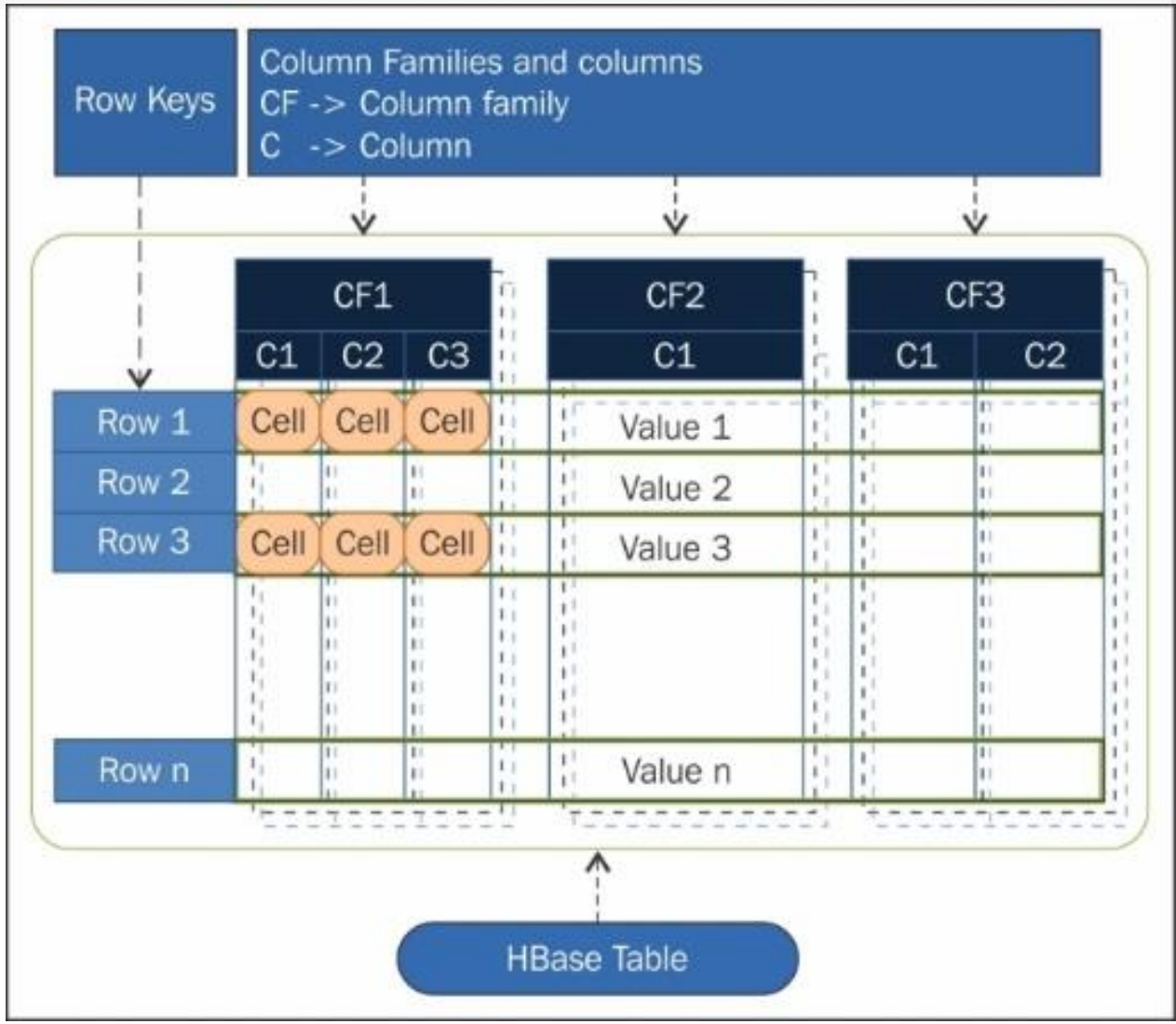
Row key	Value (Column Family, Qualifier, Version)
1234	Info{'lastName': 'Smith', 'firstName':'John'} pwd{'tweet':'hello' @ts 20130710}
5678	Info{'lastName': 'Brown', 'firstName':'Joe'} pwd{'tweet':'xyz' @ts 20120825, 'tweet':'zzz' @ts 20130916}

HBase Data Model

- **No Schema**
- **Table**
  - **Row-key must be unique**
  - **Rows are formed by one or more columns**
  - **Columns are grouped into Column Families**
  - **Column Families must be defined at table creation time**
  - **Any number of Columns per column family**
  - **Columns can be added on the fly**
  - **Columns can be NULL**
    - NULL columns are NOT stored (free of cost)
    - Column only exist when inserted (Sparse)
- **Cell**

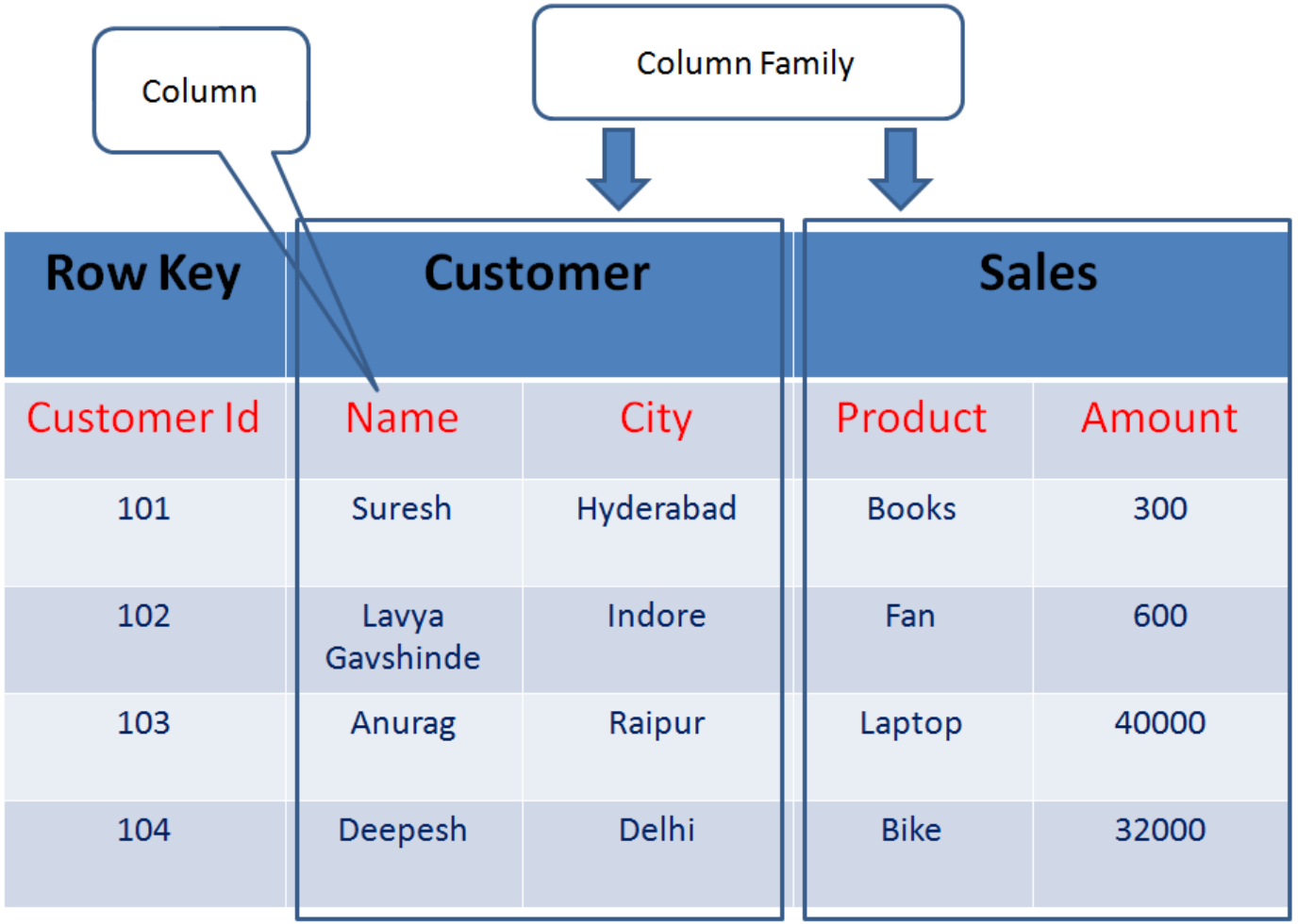
  Row Key, Column Family, Qualifier , Timestamp / Version
- **Data represented in byte array**

  Table name, Column Family name, Column name



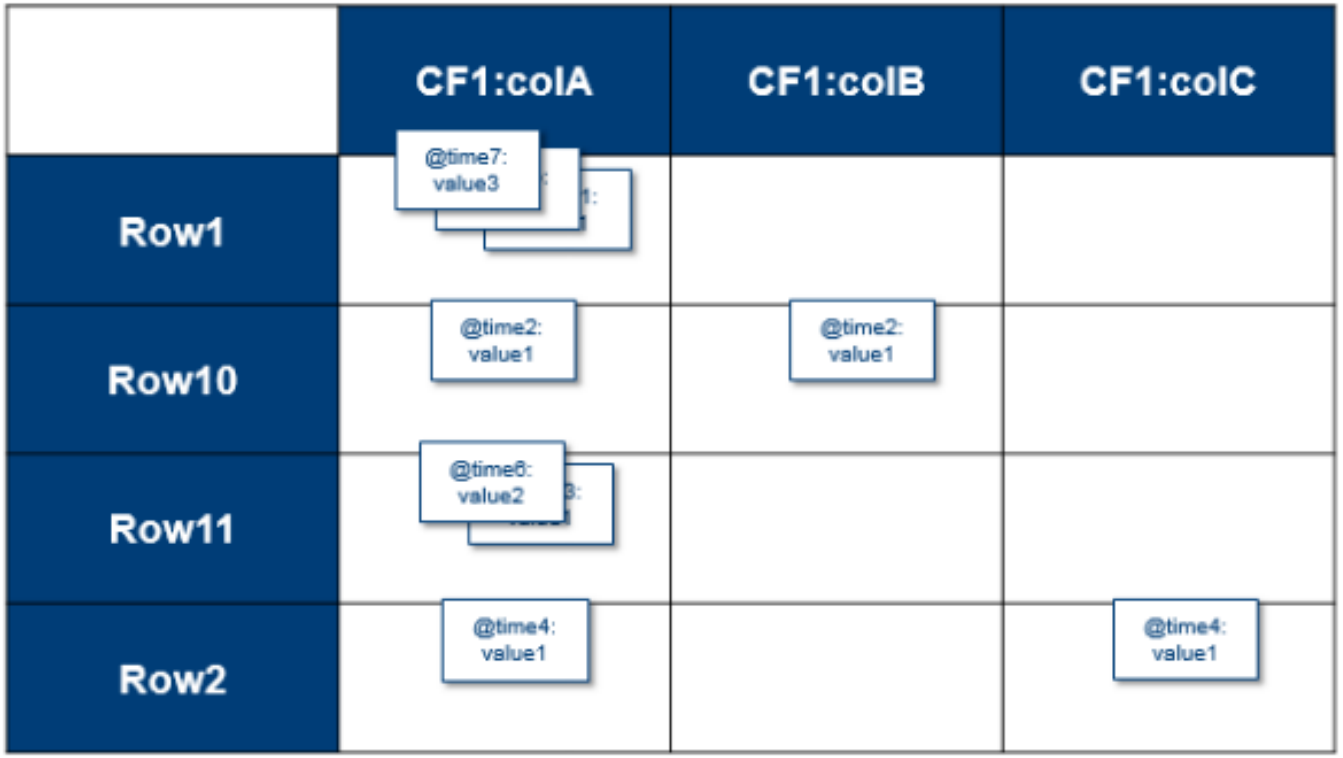
HBase: Keys and Column Families

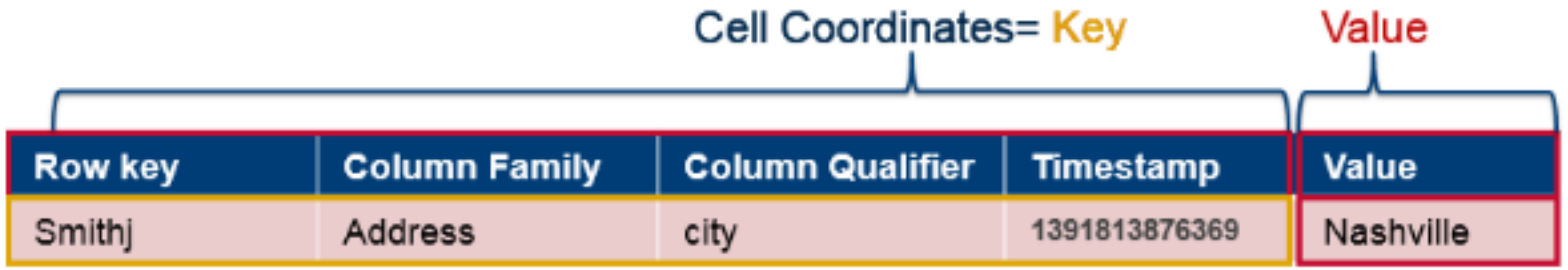
- **RowKey**
  - Byte array
  - Act as a **Primary Key**
  - Natively indexed
- **Column Family**
  - Should have a name (string)
  - Has one or more column(s)
- **Column**
  - Belongs to a Column Family
  - Imbedded into a Row
    - ***familyName:columnName***



HBase: Keys and Column Families

- **Version Number**
  - Unique for each RowKey
  - By Default → System's timestamp
  - Data type : Long
- **Value (Cell)**
  - Byte array



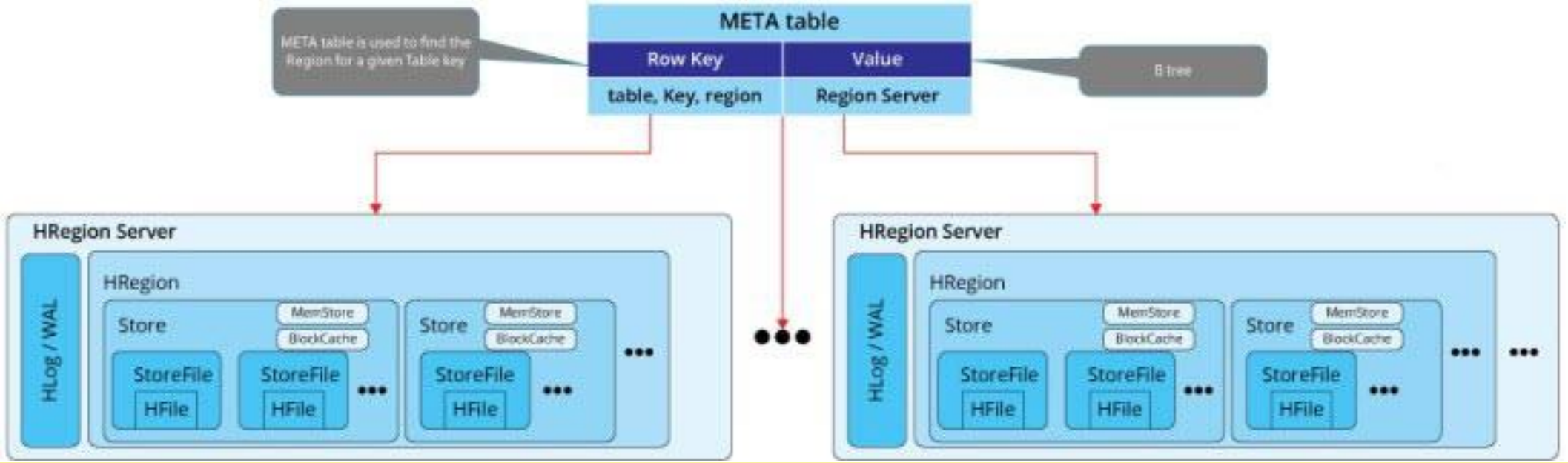


HBase : Zookeeper Role

- **Monitors the state of the cluster and report to the Master Server**
- **Stores critical information of the cluster *(-ROOT- and .Meta. tables)***

-ROOT- table: contains .META. Tables list and their locations.

.META. table : contains Regions list and their locations.



Hbase : Finding a Region Server

Zookeeper

ROOT Region

-ROOT-

META Regions

Table name	Start row	End row	Location
TableFoo	rowA	rowF	RegionServer #1
TableFoo	rowG	rowP	RegionServer #2
TableFoo	rowQ	rowZ	RegionServer #1
TableBar	rowA	rowG	RegionServer #3

Table name	Start row	End row	Location
TableBar	rowH	rowS	RegionServer #3
TableBar	rowT	rowZ	RegionServer #2
TableBaz	rowA	rowM	RegionServer #1

Table name	Start row	End row	Location
TableBaz	rowN	rowZ	RegionServer #2

TableFoo Regions

- Rows A-F
- Rows G-P
- Rows Q-Z

TableBar Regions

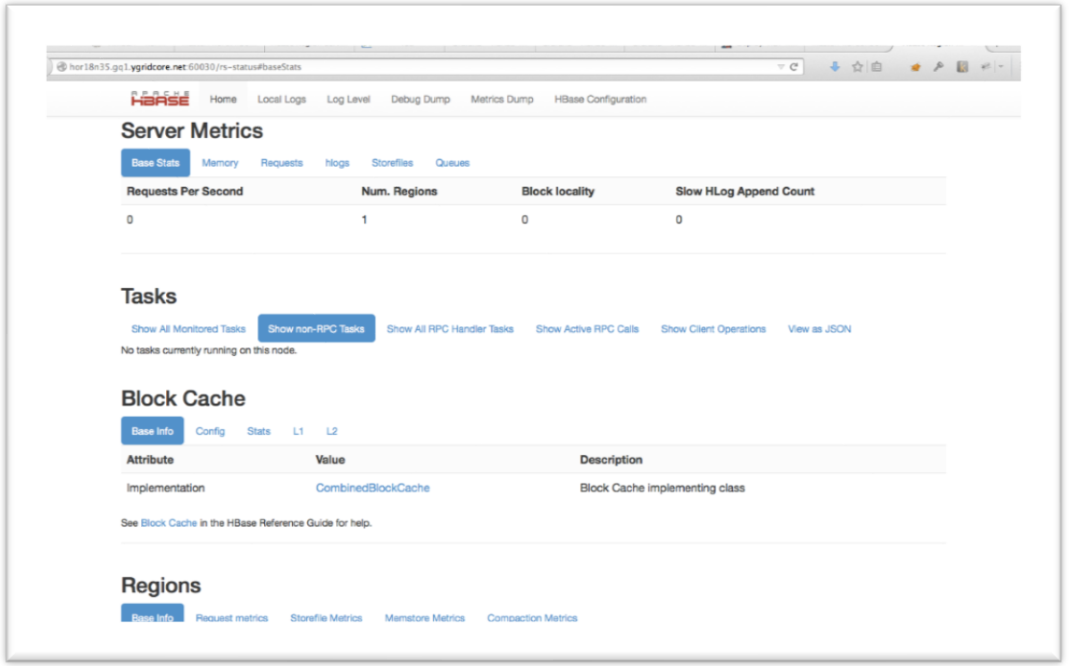
- Rows A-G
- Rows H-S
- Rows T-Z

TableBaz Regions

- Rows A-M
- Rows N-Z

Accessing HBase – *(HBase Clients)*

- **Program / API based clients**
  Java, REST, Thrift, Avro
- **Batch Clients**
  MapReduce (Pig, Hive)
- **Shell: hbase shell**
  Command Line Interface to access HBase
  Supports Client and Administrative operations.
- **Web-based UI**
  HUI (HBase cluster UI)



Master http://localhost:60010

RegionServer http://localhost:60030

HBase shell (basic commands)

Command	Description
list	Shows list of tables
create ‘users’, ‘info’	Creates users table with a single column family name info.
put ‘users’, ‘row1’, ‘info:fn’, ‘John’	Inserts data into users table and column family info.
get ‘users’, ‘row1’	Retrieve a row for a given row key
scan ‘users’	Iterate through table users
disable ‘users’ **drop ‘users’**	Delete a table (requires disabling table)

CRUD explained

CREATE	=	PUT
READ	=	GET
UPDATE	=	PUT
DELETE	=	DELETE

It’s time for a break

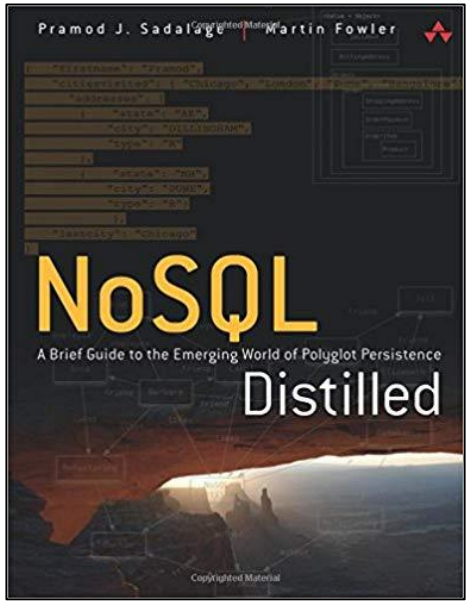
Grab some coffee, We’ll be back in 15min

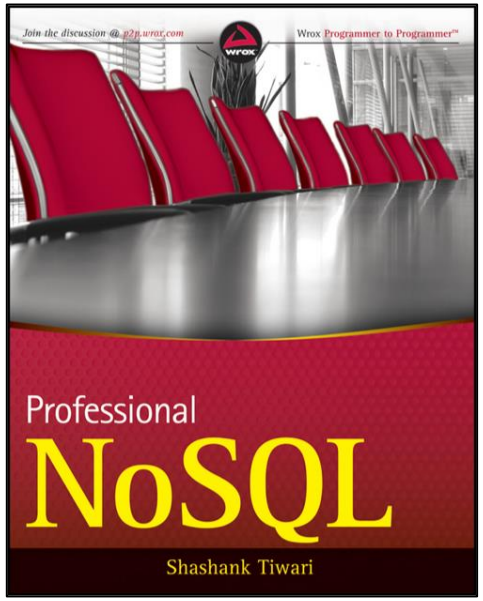
File Format - HBase Workshop

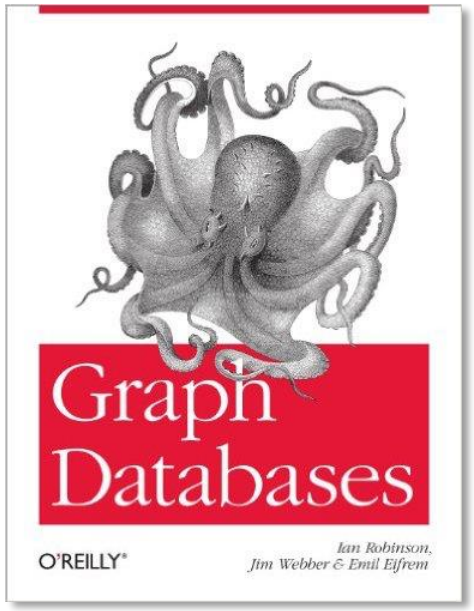
Online Resources

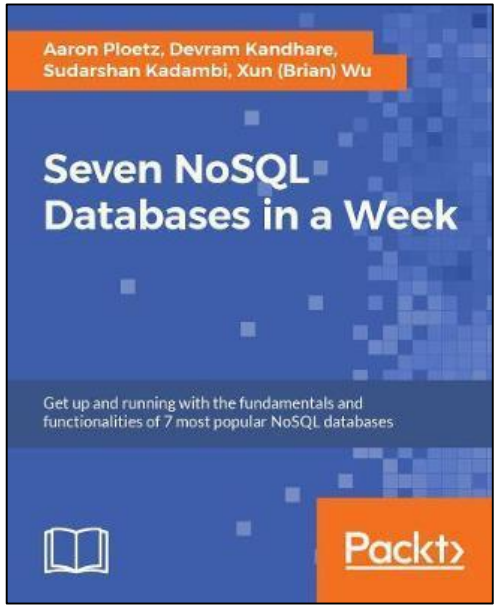
- http://nosql-database.org/
- http://www.ignoredbydinosaurs.com/2013/05/explaining-non-relational-databases-my-mom
- http://en.wikipedia.org/wiki/NoSQL
- http://greendatacenterconference.com/blog/the-five-key-advantages-and-disadvantages-of-nosql
- http://www.tutorialindustry.com/nosql-tutorial-for-beginners
- http://www.techrepublic.com/blog/10-things/10-things-you-should-know-about-nosql-databases
- http://readwrite.com/2011/10/24/oracle-formally-embraces-nosql#awesm=~oCvdI8zKkJmAiZ
- http://www.oracle.com/technetwork/database/database-technologies/nosqldb/overview/index.html
- http://www.neo4j.org/learn/nosql
- http://www.w3resource.com/mongodb/nosql.php
- http://architects.dzone.com/articles/putting-nosql-perspective
- http://en.wikipedia.org/wiki/Shard_%28database_architecture%29
- http://en.wikipedia.org/wiki/Consistent_hashing

Resources

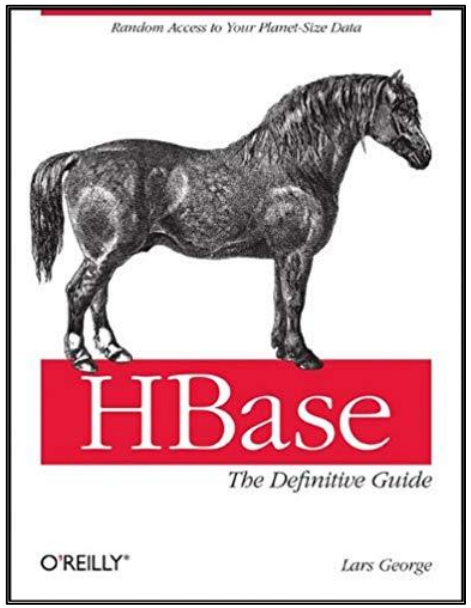

Pramod J. Sadalage
Martin Fowler
NoSQL
A Brief Guide to the Emerging World of Polyglot Persistence
Distilled

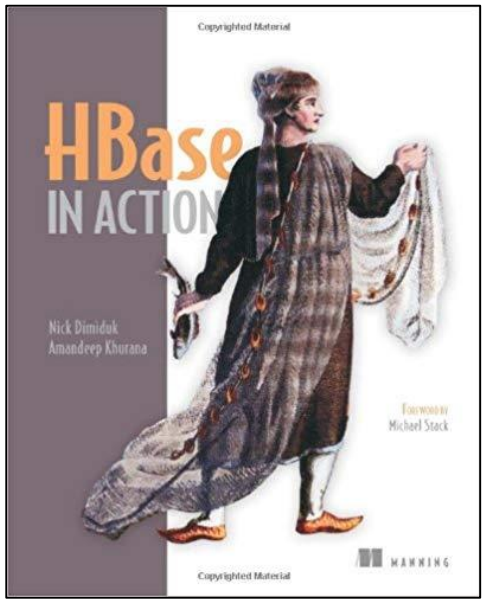

Professional
NoSQL
Shashank Tiwari


Graph
Databases
O'REILLY
Ian Robinson,
Jim Webber & Emil Eifrem


Aaron Ploetz, Devram Kandhare,
Sudarshan Kadambi, Xun (Brian) Wu
Seven NoSQL
Databases in a Week
Get up and running with the fundamentals and
functionalities of 7 most popular NoSQL databases
Packt

Resources





Online Resources

https://hbase.apache.org/

Thank You