Big Data Storage and Processing

MSc in Data Analytics

CCT College Dublin

Apache HBase

Week 6

Lecturer: Dr. Muhammad Iqbal*

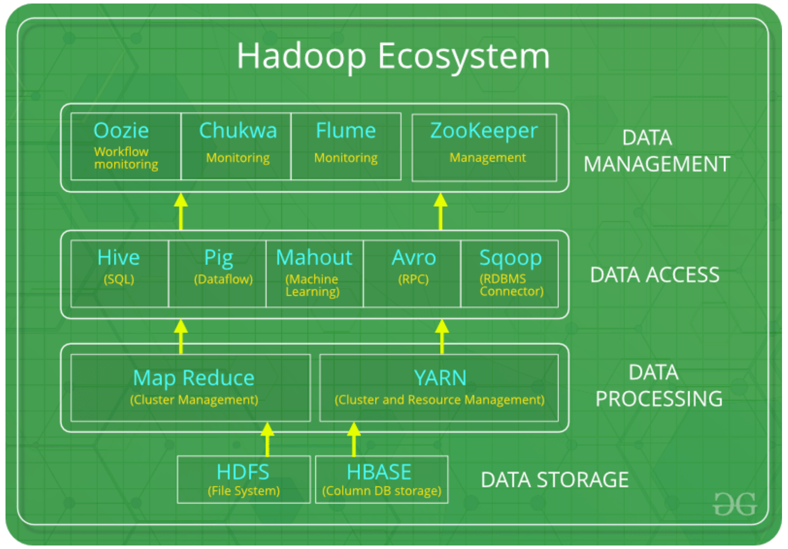
Email: miqbal@cct.ie

Agenda

- Hadoop Ecosystem
- Introduction to Column-oriented DBs
- Relational Databases: Transaction Support
- Relational & Non-relational Databases
- Classification of Non-relational Databases
- HBase & Architecture
- Main components of HBase Architecture

Hadoop Ecosystem

- **Hadoop Ecosystem** is a platform which provides various services to solve the Big data problems. It includes Apache projects and various commercial tools and solutions.
- **New Approach**
  - Google
  - Amazon
- Scalable Storage and Processing platforms based on commodity hardware
- Implemented as an open-source Apache Hadoop project



The Hadoop Ecosystem

- **HDFS:** Hadoop Distributed File System
- **YARN:** Yet Another Resource Negotiator
- **MapReduce:** Programming based Data Processing
- **Spark:** In-Memory data processing
- **PIG, HIVE:** Query based processing of data services
- **HBase:** NoSQL Database
- **Mahout, Spark MLLib:** Machine Learning algorithm libraries
- **Zookeeper:** Managing cluster
- **Oozie:** Job Scheduling

Introduction to
Column-oriented DBs

- **Recent Approaches**
  - **Access to data via**
    - Column-oriented databases
    - Massively Parallel Processing databases
  - **Column-Oriented Databases**
    - Save data grouped by columns
    - Subsequent columns stored contiguously on disk
      - – Good for analytical processing
      - – Reduced **I/O**
      - – Compression opportunities

Id	Fld1	Fld2	Fld3
1675765	<sdhj>hgasd**	154354UU	*&^jjjjGHF
7868733	`<jljlj><>m,j	83899HHY	))((JNNNk
9877388	<iuiuoun<>>	()*8829829	YIUHbh+_'

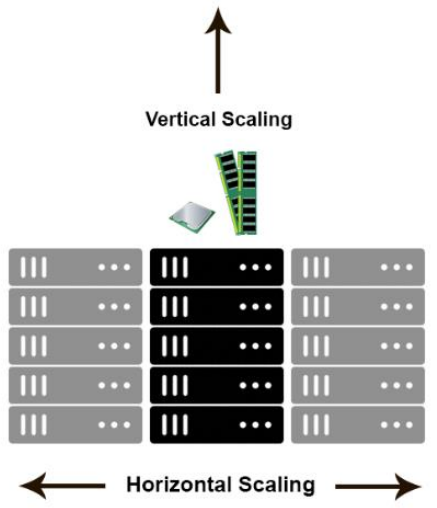
Id	1675765	7868733	9877388
Fld1	<sdhj>hgasd**	`<jljlj><>m,j	<iuiuoun<>>
Fld2	154354UU	()*8829829	()*8829829
Fld3	*&^jjjjGHF	))((JNNNk	YIUHbh+_'

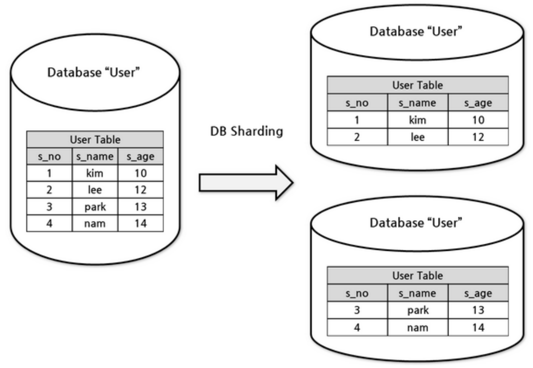
HBase
- Uses an on-disk column storage format
- Provides key-based access to a specific cell of data or a sequential range of cells

Relational Databases

Issues

- Difficulties faced to fulfil the demand for recent storage requirements
- Relational databases typically normalised (Strict structure)
- Transaction semantics: **ACID** properties provide ***strong consistency***
  - **Slave databases** can be added to cope with initial increases in demand for data access services
  - System may be **scaled vertically**
  - If **SQL JOINs** suffer performance degradation, denormalisation may be entertained
  - Stored Procedures/ Secondary Indexes may be abandoned
  - Data may be sharded





User Table

s_no	s_name	s_age
1	kim	10
2	lee	12
3	park	13
4	nam	14

User Table

s_no	s_name	s_age
1	kim	10
2	lee	12

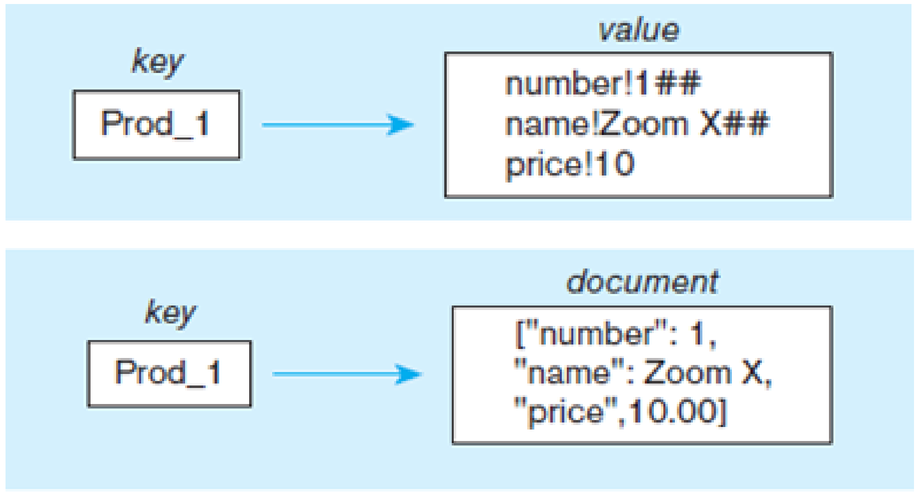
User Table

s_no	s_name	s_age
3	park	13
4	nam	14

NoSQL Databases

Classification

- **Key-Value Stores**
  - Simple data model
  - Map/ Dictionary
  - Clients can put, request (and delete) values by key
  - Amazon DynamoDB, Redis and Riak

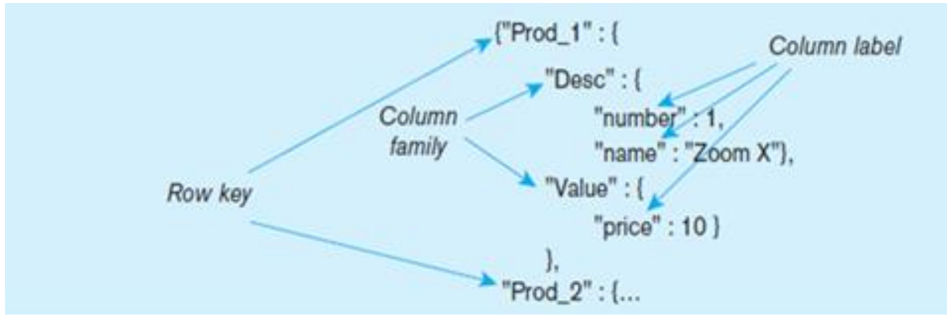


- **Document Oriented Databases**
  - Stores data as ***documents***
  - Documents encapsulate/ encode data
    - Allows encapsulation of key/ value pairs within document.
      - Values may be strings, booleans, numerics, lists etc.
        - XML, JSON, BSON, PDF
      - No strict schema
      - MongoDB is an example

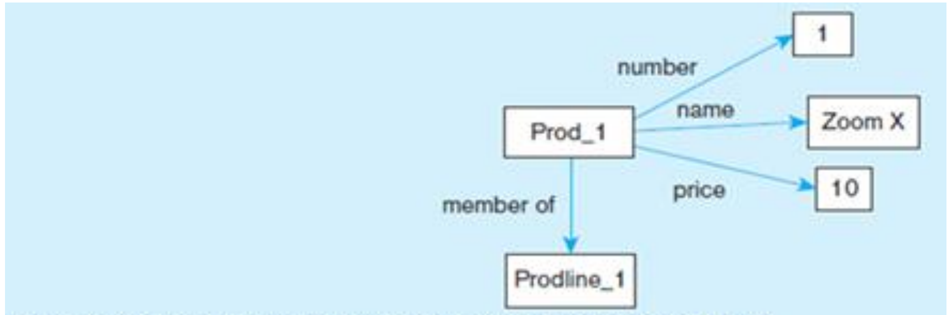
NoSQL Databases

Classification

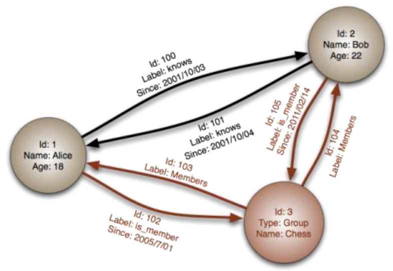
- **Column-oriented Database**
  - Rows and columns, Distribution of data based on both key values (records) and columns, using “column groups/ families”.



- **Graph-oriented Database**
  - Maintain information regarding the relationships between data items. Nodes with properties, Connections between nodes (relationships) can also have properties.



Some of the example structures have been adapted from Kauhanen (2010).



NoSQL Databases

Comparison

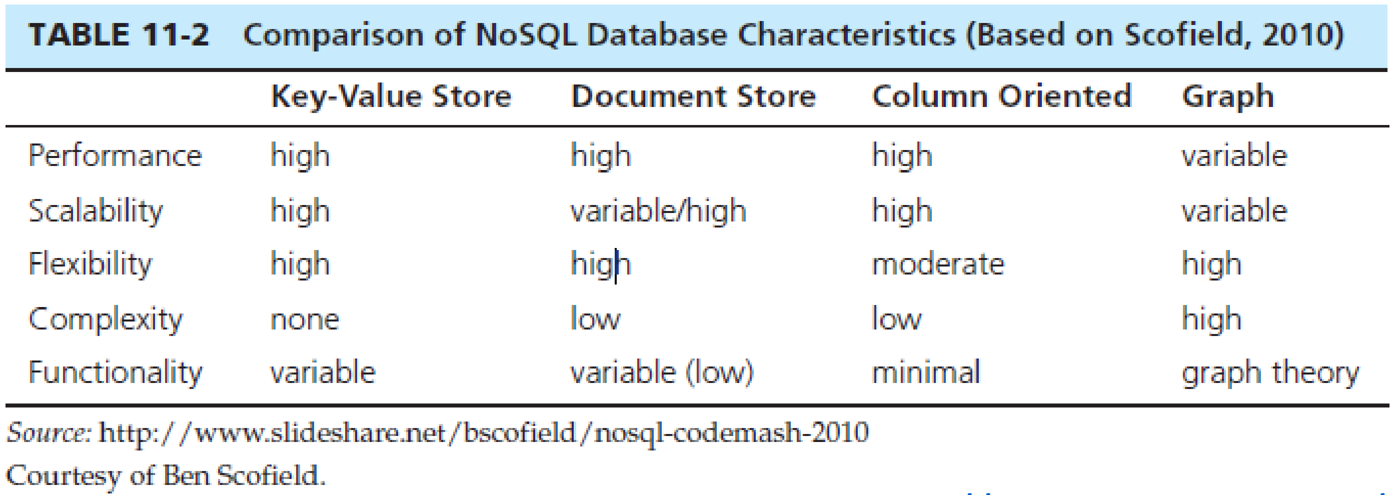


TABLE 11-2 Comparison of NoSQL Database Characteristics (Based on Scofield, 2010)

	Key-Value Store	Document Store	Column Oriented	Graph
Performance	high	high	high	variable
Scalability	high	variable/high	high	variable
Flexibility	high	high	moderate	high
Complexity	none	low	low	high
Functionality	variable	variable (low)	minimal	graph theory

Source: http://www.slideshare.net/bscofield/nosql-codemash-2010
Courtesy of Ben Scofield.

https://nosql-database.org/

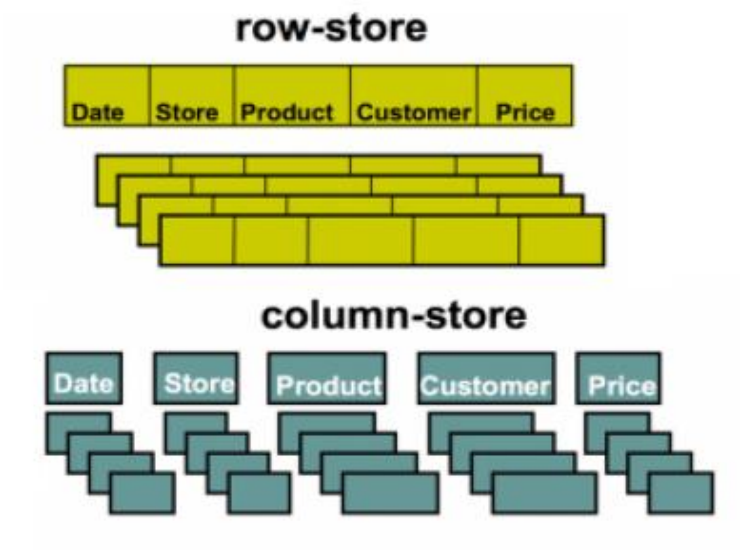
NoSQL Examples

- **Redis** – Key-value store DBMS
- **MongoDB** – Document store DBMS
- **Apache HBase** – Column store DBMS
- **Neo4j** – Graph DBMS

NoSQL Databases

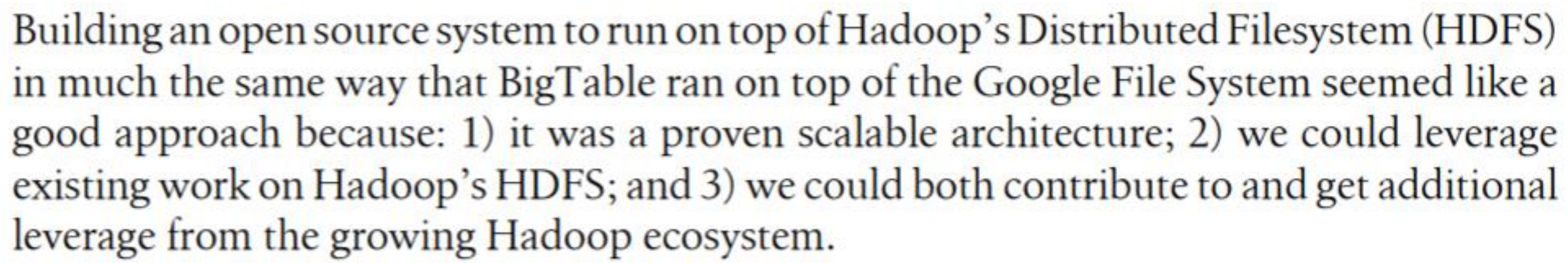
Column Oriented Databases

- Stores data as sections of columns rather than as rows
  - Stores all values of a column together
  - then the values of the next column together
  - and so on
- Good for applications performing analysis of data
- Good compressibility
- Examples: **HBase and Cassandra**
- We will look at HBase as an example



HBase

- A San Francisco-based firm planned to create a Web-based natural language search engine in 2006.
- The generated index size was significantly larger than usual.
- The datastore for accommodating the index intermediaries and webcrawl on top of AWS was breaking under the load.
- They were searching for a different solutions and HBase was discovered.
- Powerset's head of engineering at the time, Chad Walters, offers the following reflections on the event


Building an open source system to run on top of Hadoop's Distributed Filesystem (HDFS) in much the same way that BigTable ran on top of the Google File System seemed like a good approach because: 1) it was a proven scalable architecture; 2) we could leverage existing work on Hadoop's HDFS; and 3) we could both contribute to and get additional leverage from the growing Hadoop ecosystem.

Hbase

The Dawn of Big Data

- **Systems** like **Hadoop** enable them to gather and process petabytes of data, and process using new machine learning algorithms to meet the business objectives.

- The implementation of scalable storage and processing system outside of Google as part of the open source Hadoop project: **HDFS and MapReduce**.

- Hadoop excels at storing data of arbitrary, semi-, or even unstructured formats as well as also complements existing database systems of almost any kind.

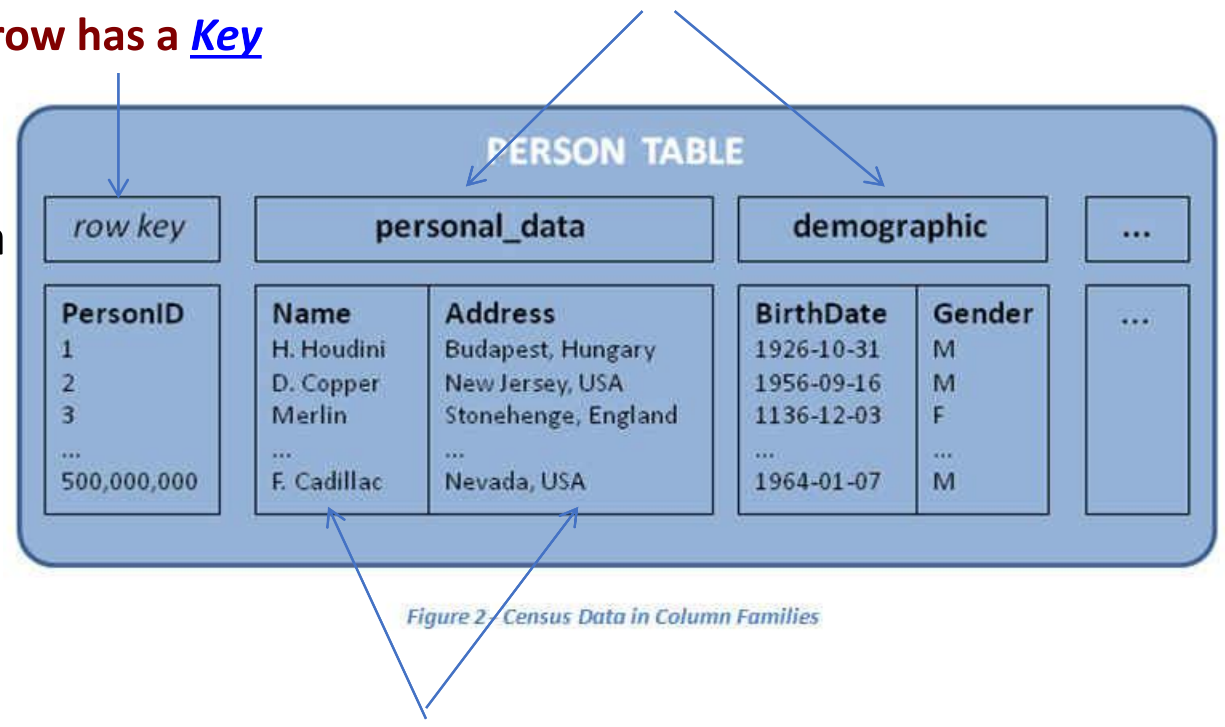
- This makes analysis easy and fast, but the users also need access to the final data, not in batch mode but using random access.

- **RDBMS** are the most prominent, but there are also quite a few specialized variations and implementations, like object-oriented databases. Most **RDBMS** strive to implement Codd's 12 rules, which forces them to comply to very rigid requirements.

HBase Architecture

- A ***column*** is the most basic unit.
- One or more columns form a ***row.*** Each row can have many Columns in each column family.
- Each row is addressed by a ***row key.***
- A number of rows form a ***table.*** Tables have a fixed number of Column Families.
- Each column has a set of values and each with a timestamp.
- Each **rowkey:columnfamily:column:timestamp** combination represents coordinates for a Cell.

Tables, Rows, Columns, and Cells


Each record is divided into Column Families
Each row has a Key
PERSON TABLE
Each column family consists of one or more Columns

row key	personal_data		demographic		...
PersonID	**Name**	**Address**	**BirthDate**	**Gender**	...
1	H. Houdini	Budapest, Hungary	1926-10-31	M	
2	D. Copper	New Jersey, USA	1956-09-16	M	
3	Merlin	Stonehenge, England	1136-12-03	F	
...	...	...	...	...	
500,000,000	F. Cadillac	Nevada, USA	1964-01-07	M	

Figure 2: Census Data in Column Families

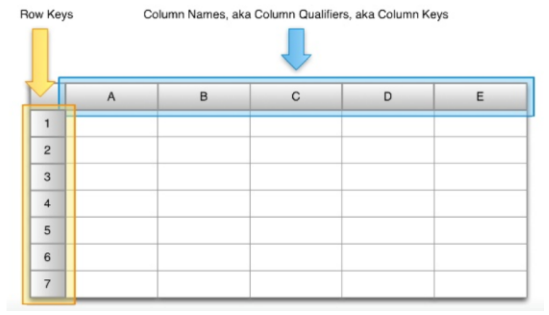
HBase Architecture

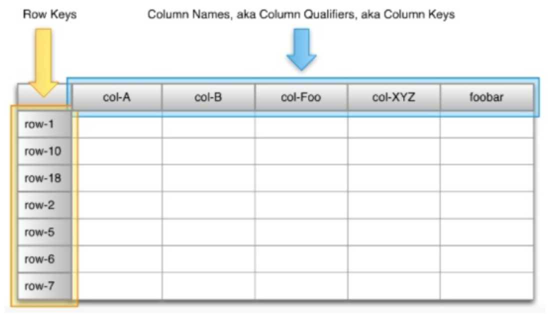
Tables, Rows, Columns, and Cells

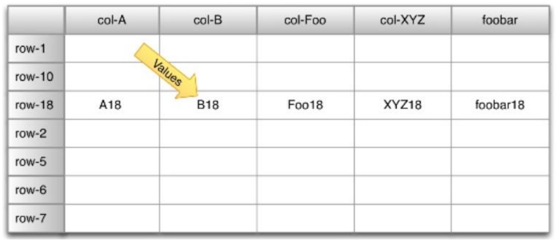
- Rows are sorted lexicographically by their row-key
  - Row-keys are compared at a binary level, byte by byte, from left to right
  - Provides something similar to a traditional primary key
- Row-keys are any array of bytes
- Columns are grouped into ***column families***
  - Provide semantic boundaries between different sets of columns
  - Can be used to control storage features (e.g., compression)
  - All columns in a column family are stored together in the same low-level storage file called a ***Hfile***
  - Columns are referenced as ***family:qualifier***

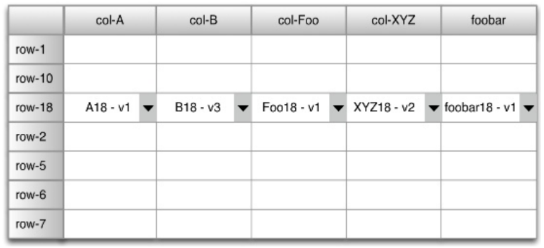
Defined at create time – should not be changed

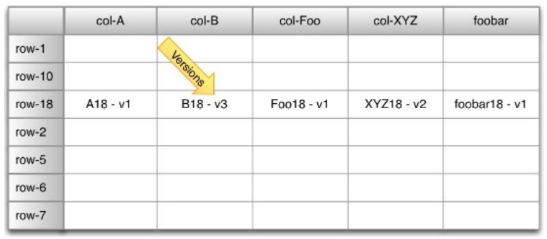
HBase Architecture

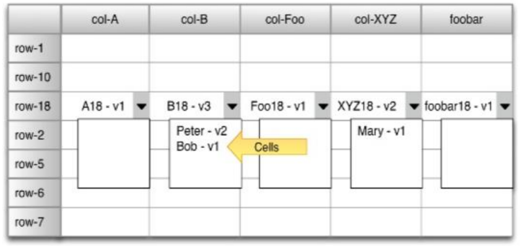




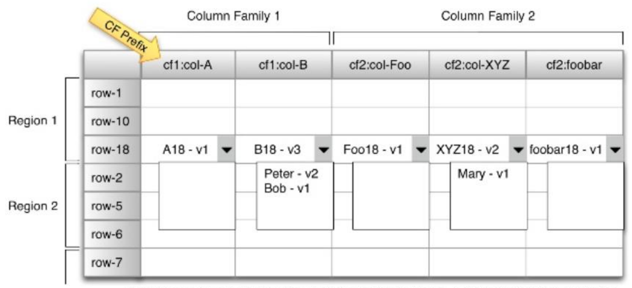




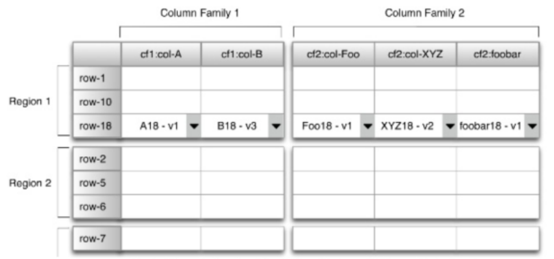




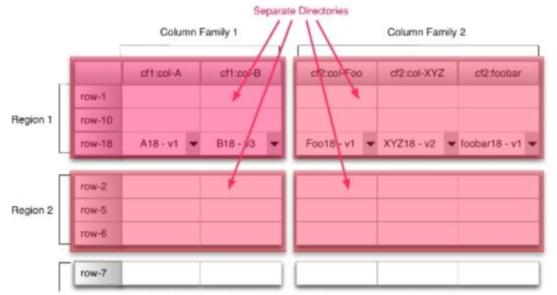
Coordinates for a Cell: *Row Key* → *Column Name* → *Version*



Coordinates for a Cell: *Row Key* → *Column Family Name* → *Column Qualifier* → *Version*



Physical Coordinates for a Cell: *Region Directory* → *Column Family Directory* → *Row Key* → *Column Family Name* → *Column Qualifier* → *Version*

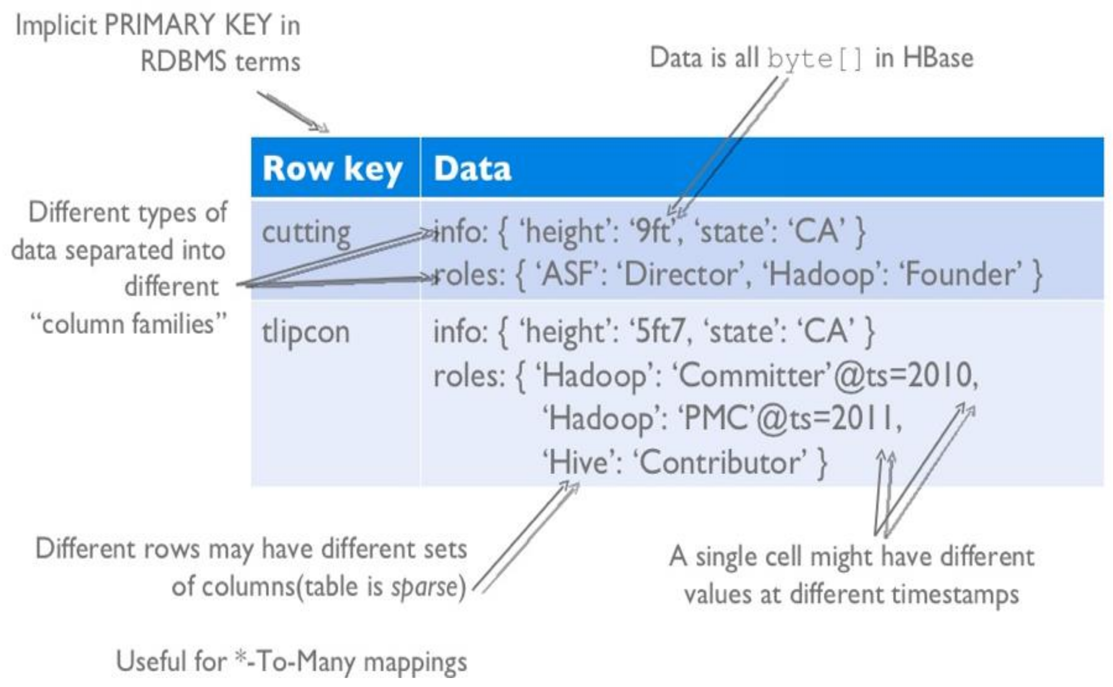


Physical Coordinates for a Cell: *Region Directory* → *Column Family Directory* → *Row Key* → *Column Family Name* → *Column Qualifier* → *Version*

HBase Architecture

Tables, Rows, Columns, and Cells

- A ***cell*** is either implicitly timestamped by the system or a timestamp can also be provided by the user
- Different versions of a cell are stored in decreasing timestamp order
  - Generally, most recent data required first
- Number of versions or timespans for which to maintain versions can be configured
- Access to row data is ***atomic***
  - No guarantee or transactional features that span multiple rows or multiple accesses across tables



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' }

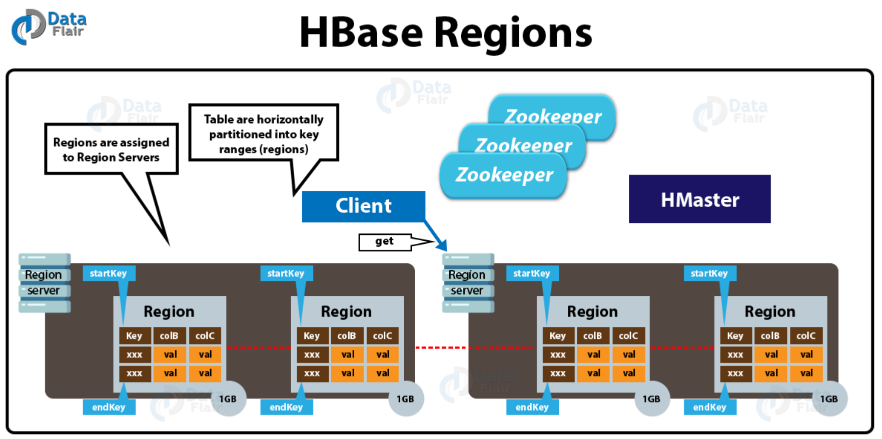
Implicit PRIMARY KEY in RDBMS terms
Data is all byte[] in HBase
Different types of data separated into different "column families"
Different rows may have different sets of columns(table is *sparse*)
A single cell might have different values at different timestamps
Useful for *-To-Many mappings

- A **Column Family** is a group of related columns
- All **columns** must be in a column family
- Each row can have a completely different set of **columns** for a **column family**

HBase Architecture

Regions

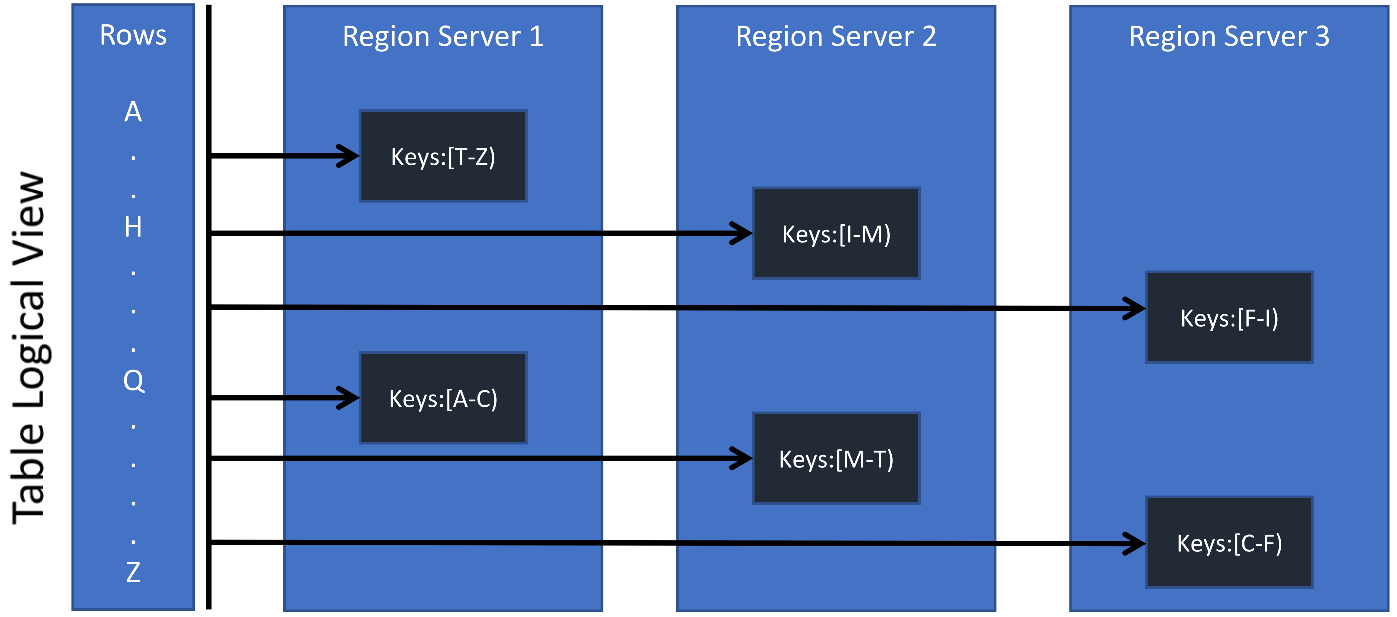
- The basic unit of **scalability** and **load balancing** in HBase is a ***region***
- A **region** is essentially **<u>contiguous ranges of rows</u>** stored together
    - The system will dynamically split them when they exceed a certain threshold size limit - ***autosharding***
    - **Regions** may also be merged
        - Reduce number of files
- **Regions** may be spread across many physical servers
    - Distributes load and increases scalability
- Each **region** is served by exactly **one region server**
- A **region server** can serve more than **one region**



HBase Architecture

Regions

- Initially there is only one region for a table – however, when a limit is reached the region is split into two at the ***middle-key.***
- The default size of the region is 256 MB but it can be adjusted according to the requirements.
- Typically 10 to 1000 regions per server with each region between 1GB and 2GB.



HBase Architecture

Storage API

- **Create / Delete** tables and column families
- Modify metadata
- **CRUD** functionality
- ***scan*** API
- Single row atomic transactions (read-modify-write)
- Integration with MapReduce framework
- No specific domain language, such as **SQL**
- Imperative programming

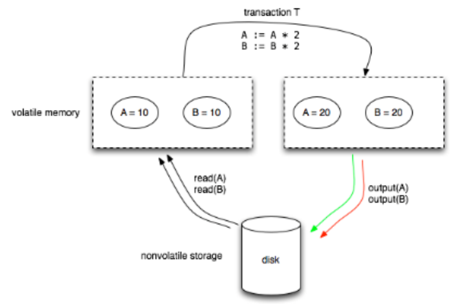
CRUD

- get: retrieve an entire, or partial row (R)
- put: create and update a row (CU)
- delete: delete a cell, column, columns, or row (D)
- Result get (Get get) throws IOException;
- void put(Put put) throws IOException;
- void delete(Delete delete) throws IOException;

HBase Architecture

File Storage

- Data is stored in ***store files*** called ***HFiles***
  - Persistent
  - Internally, files are sequences of blocks with a block index stored at the end
    - Index loaded when HFile is opened and kept in memory
    - Default block size is **64KB**
    - Lookups can be performed with a single disk seek
      - Binary search of index followed by block read from disk
- Store files typically stored in **HDFS**
  - Guarantees durability of data by writing changes across a configurable number of physical servers
- **Write-Ahead Logging** is used
  - **Commit-Log** then memstore
  - **Memstore** flushed when full



HBase Architecture

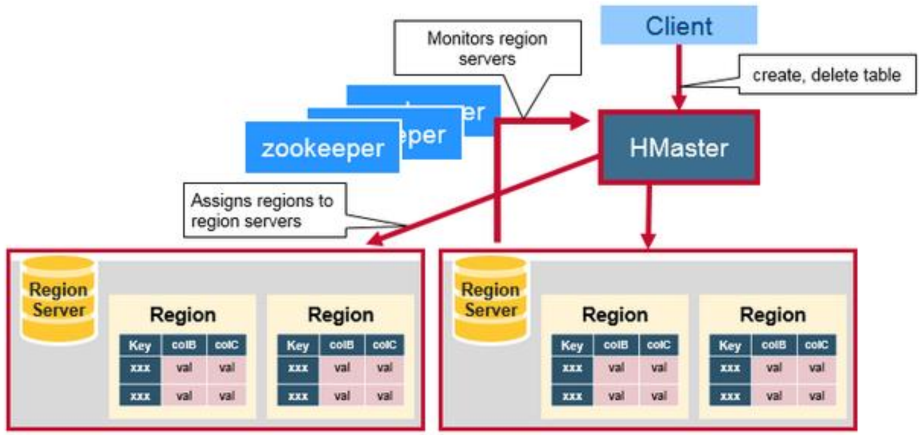
File Storage

- Store Files are **immutable**
  - When deleting **key-value** pairs, a ***delete marker*** is used to mask the actual values from reading clients.
- Reading data involves a merge of what is contained in **<u>memstores</u>** and the data on **<u>disk</u>**.
- Once the data in memory has exceeded a given maximum value, it is flushed as an **HFile** to disk.
- Flushing causes more **HFiles** to be created
  - **HBase** uses ***compaction***
  - ***Minor Compaction -*** triggered each time a memstore is flushed
  - ***Major Compaction -*** run about every 24 hours and merge together all store files into one

HBase Architecture

Main Components

- The client library
- One **master** server
- Many **region** servers

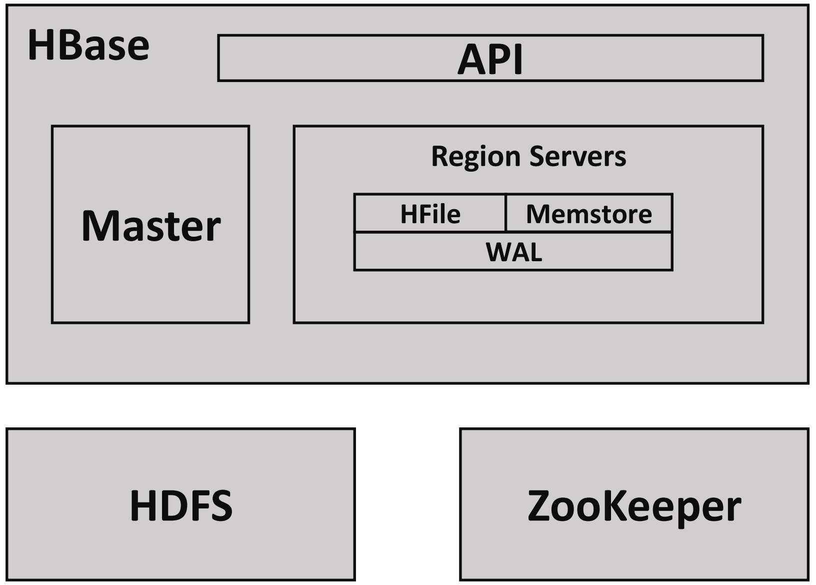


- The **master** is responsible for assigning regions to region servers and uses ***Apache Zookeeper***
  - Reliable, highly available, persistent, distributed coordination service
  - Necessary component for **HBase** operation

HBase Architecture

Main Components

- The **master server** is also responsible for handling load balancing of regions across **region servers**, to unload busy servers and move regions to less occupied ones.
- The master is not part of the actual data storage or retrieval path.
- It negotiates **load balancing** and maintains the state of the cluster, but never provides any data services to either the region servers or the clients, and is therefore lightly loaded in practice.
- In addition, it takes care of schema changes and other metadata operations, such as creation of tables and column families.

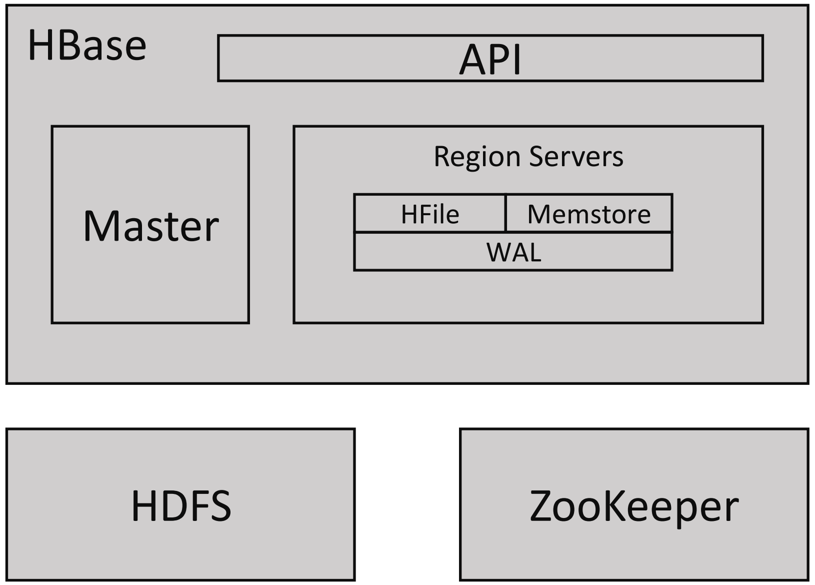


HBase Architecture

Main Components

Region servers

- Region servers are responsible for all ***read*** and ***write*** requests for all regions they serve, and also split regions that have exceeded the configured region size thresholds.
- Clients communicate directly with them to handle all data-related operations.



Resources/ References

- Lars George, 2011, HBase The Definitive Guide, O’Reilly Publishing.
- Apache HBase Primer, Deepak Vohra, Apress, November 2016.
- Hadoop with Python, Zach Radtka; Donald Miner, O'Reilly Media, Inc., 2015.
- Lublinsky B., Smith K. T. and Yakubovich A 2013, Professional Hadoop Solutions, Wrox [ISBN: 13:978-11186]
- Holmes A 2012, Hadoop in Practice, Manning Publications [ISBN: 13:978-16172]
- McKinney W. 2012, Python for Data Analysis, O'Reilly Media [ISBN: 13: 978-14493]
- Some images are used from Google search repository (https://www.google.ie/search) to enhance the level of learning.