Big Data Analytics
Lecture 4
Hbase and Hive

Yilu Zhou, PhD
Associate Professor
Gabelli School of Business
Fordham University

* Some slides are adopted from Professor Kunpeng Zhang’s Big Data course @UMD

Relational database

- Relational databases have provided a standard persistence model
- SQL has become a de-facto standard model of data manipulation (SQL)
- Relational databases manage concurrency for transactions
- Relational database have lots of tools

What is NoSQL?

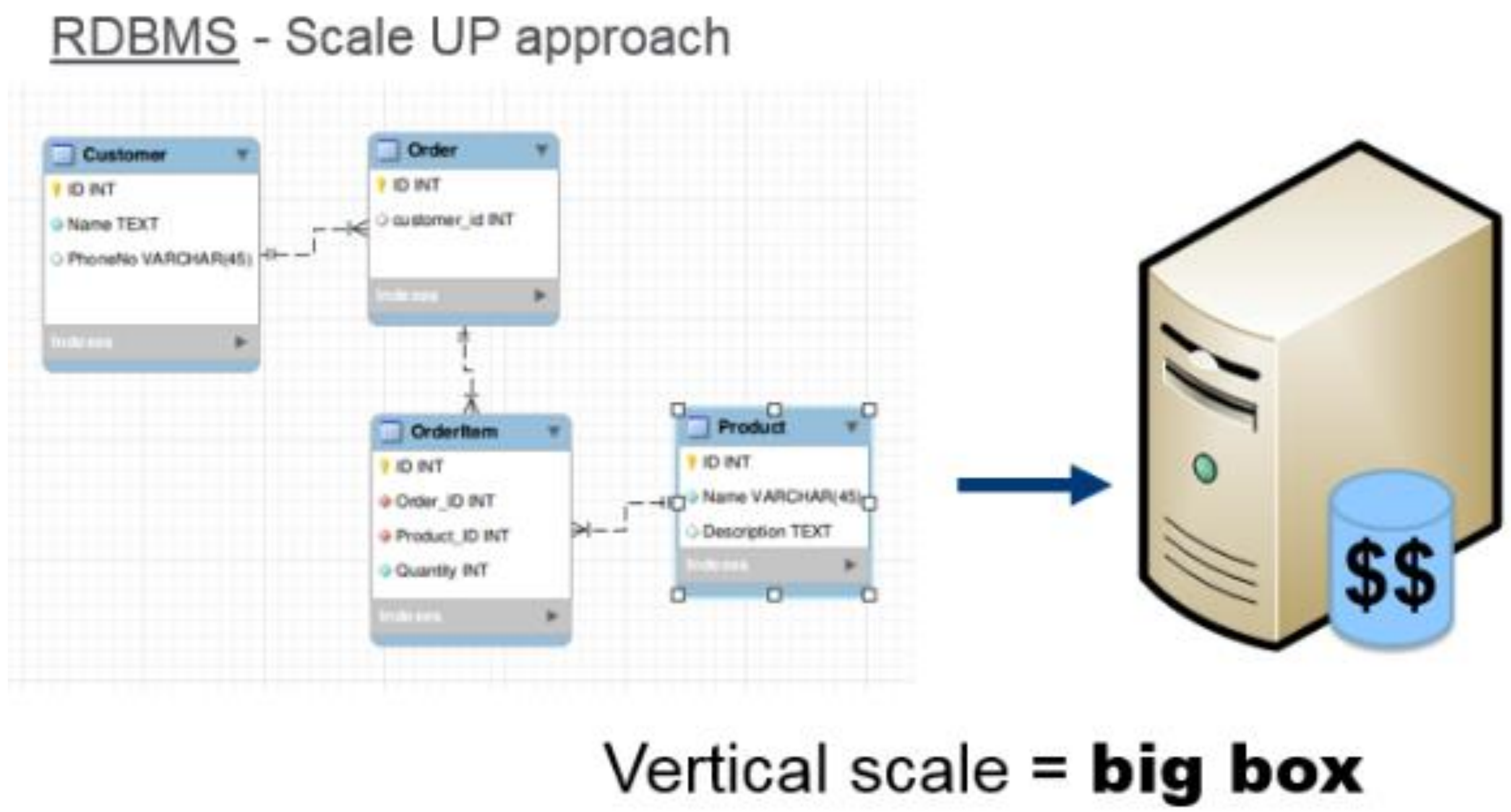
log file

- Stands for **N**ot **O**nly **SQL**
- Class of non-relational data storage systems
- Usually do not require a fixed table schema nor do they use the concept of joins

Why NoSQL

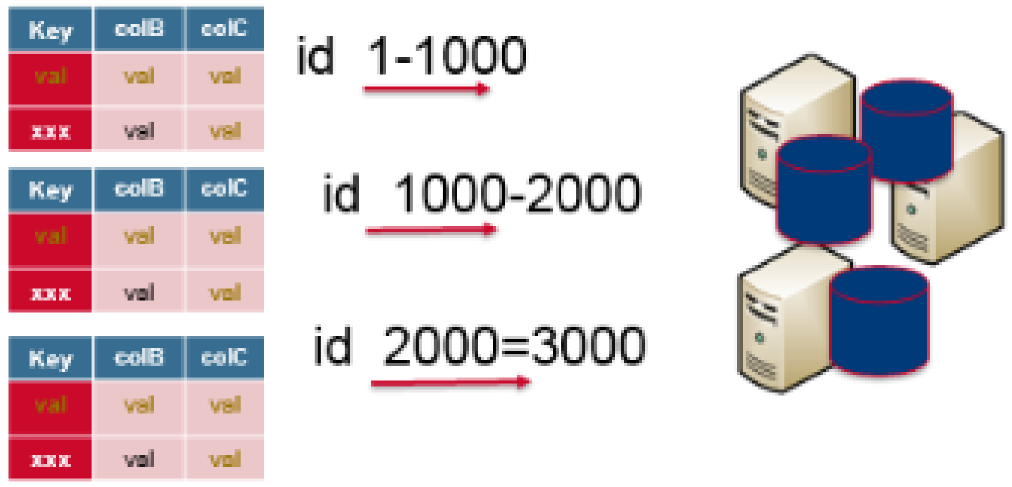
- For data storage, an RDBMS cannot be the be-all/end-all
- Just as there are different programming languages, need to have other data storage tools in the toolbox
- The NoSQL databases are a pragmatic response to growing scale of databases and the falling prices of commodity hardware.
- Most likely, 10 years from now, the majority of data is still stored in RDBMS.

Scale up (vertically)



Scale up (horizontally)

Horizontal scale: split table by rows into partitions across a cluster



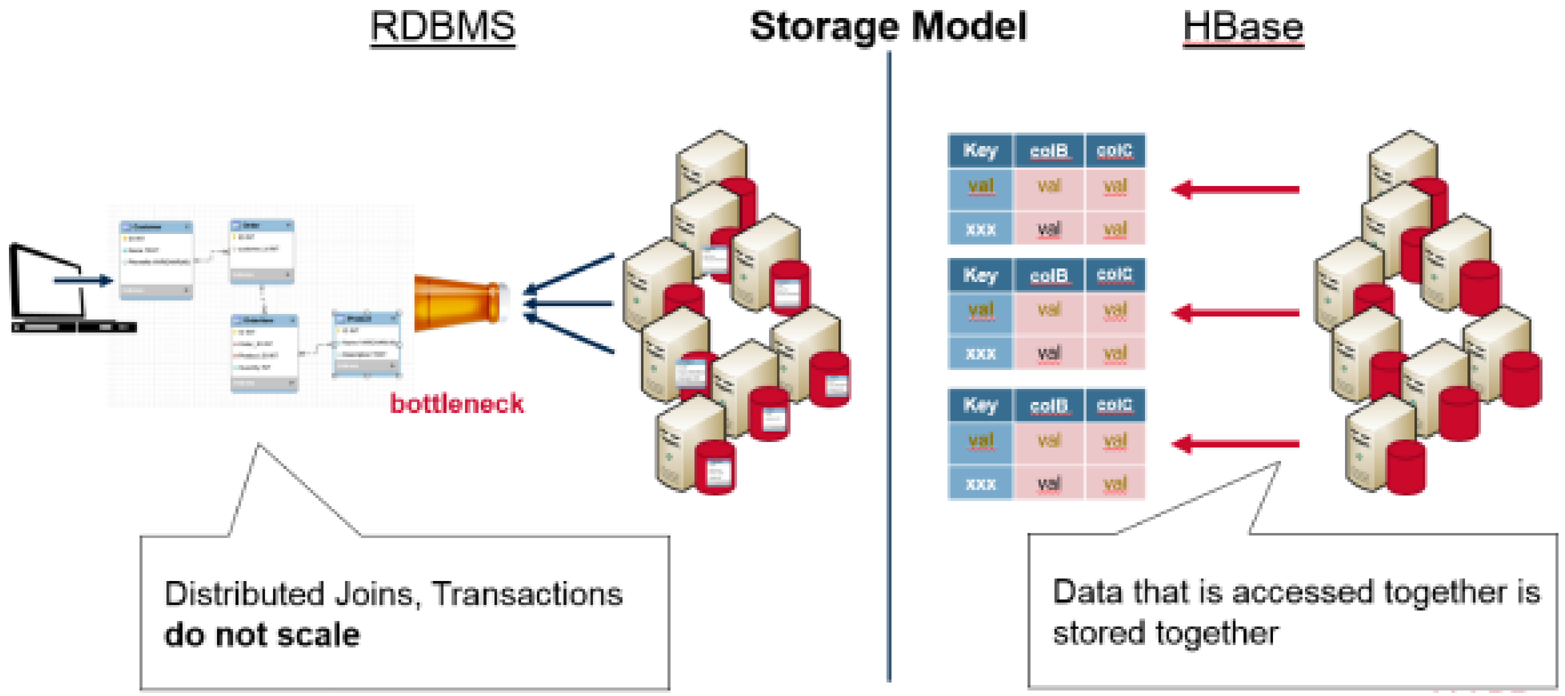
Key	colB	colC
val	val	val
xxx	val	val

Key	colB	colC
val	val	val
xxx	val	val

Key	colB	colC
val	val	val
xxx	val	val

Limitations of a relational model

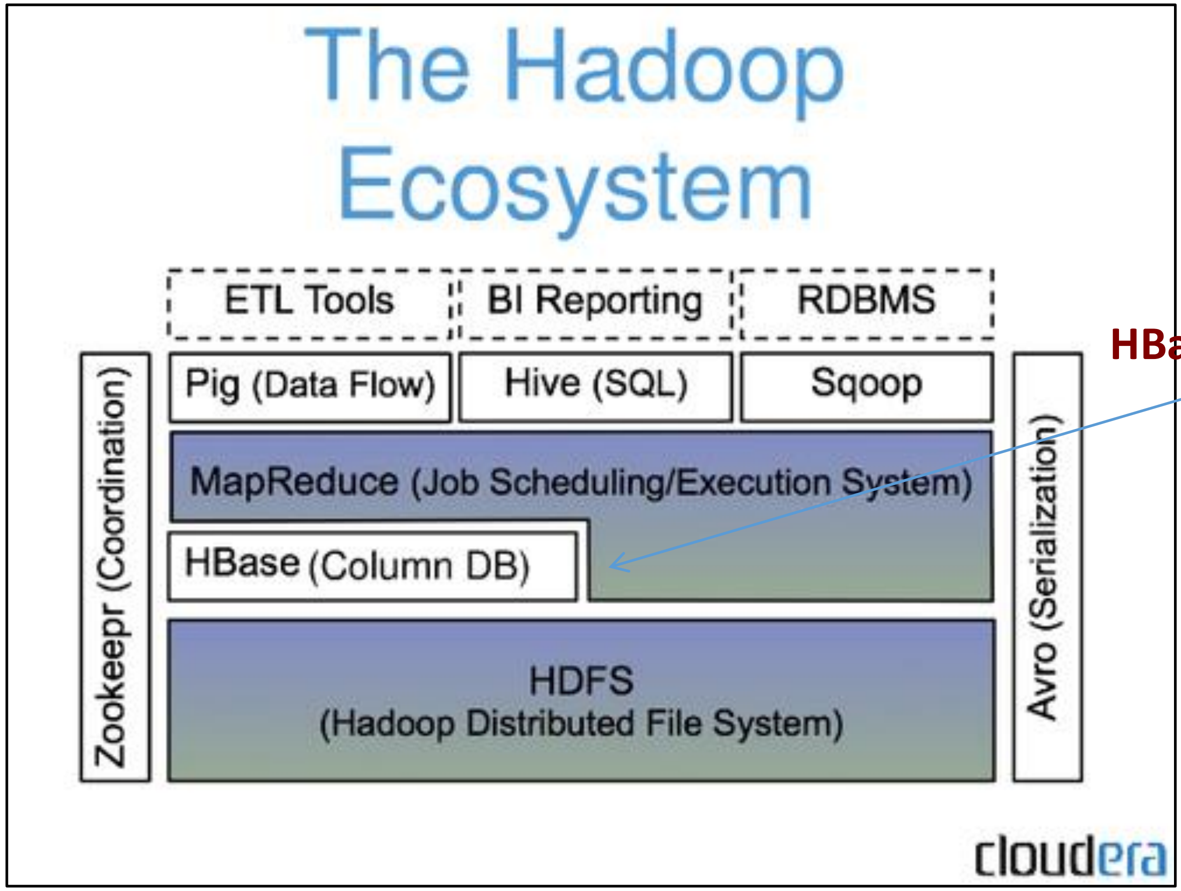
- **Database normalization eliminates redundant data, which makes storage efficient**
- **It causes joins for queries, in order to bring the data back together again.**


RDBMS
Storage Model
HBase
bottleneck
Key
colB
colC
val
xxx
Distributed Joins, Transactions
do not scale
Data that is accessed together is stored together

HBase

- HBase is a distributed column-oriented database built on top of the HDFS. It is an open-source project and horizontally scalable.
- HBase is a data model that is similar to Google's BigTable that designed to provide quick random access to huge amounts of structured data.
- HBase is a part of Hadoop ecosystem that provides real-time read/write access to data in the Hadoop File System.
- **HBase stores its data in HDFS.**

Part of Hadoop's Ecosystem



HBase is built on top of HDFS

HBase files are internally stored in HDFS

HBase vs. HDFS

- Both are distributed systems that scale to hundreds or thousands of nodes

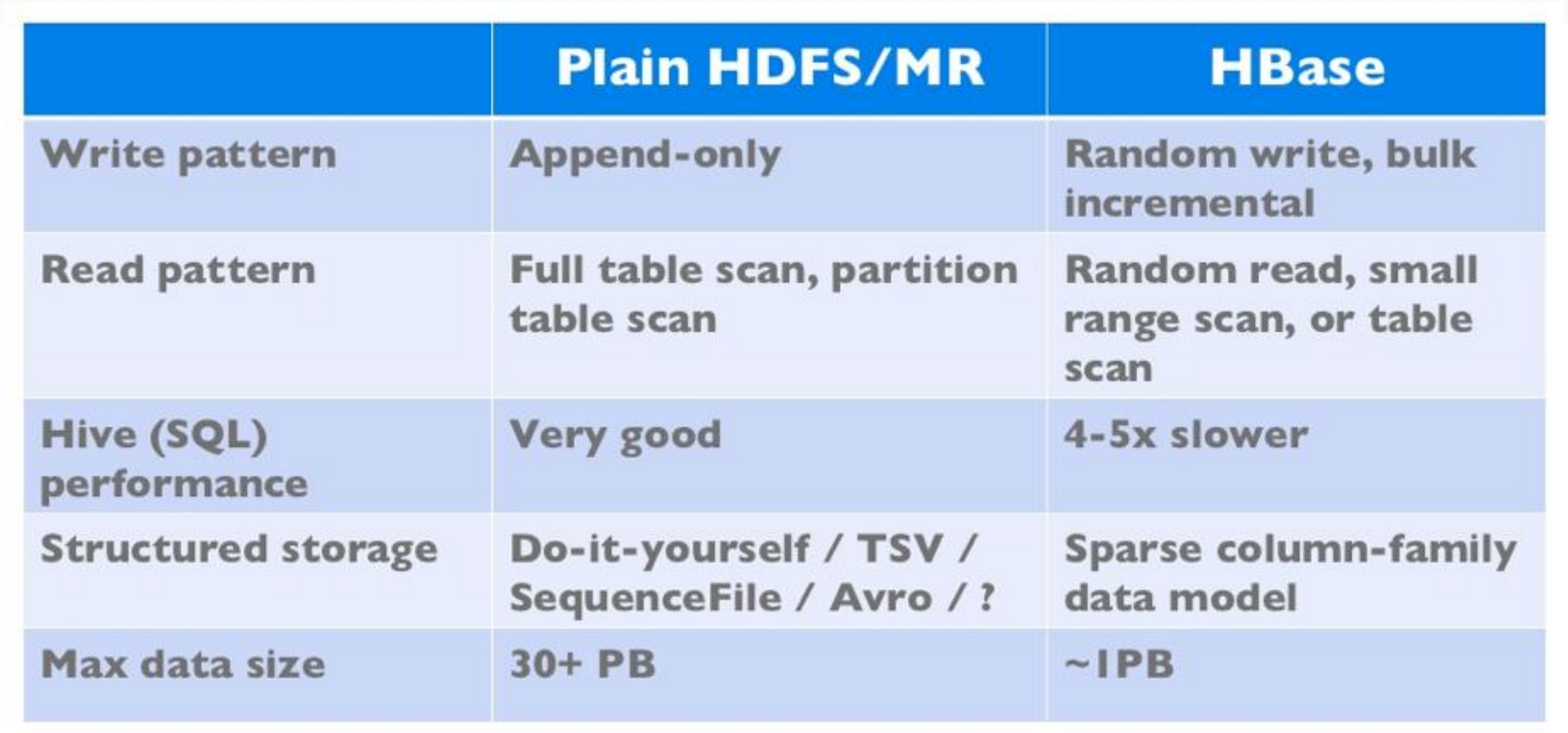
- *HDFS* is good for batch processing (scans over big files)
  - Not good for record lookup
  - Not good for incremental addition of small batches
  - Not good for updates

HBase vs. HDFS (Cont’d)

- *HBase* is designed to efficiently address the above points
  - Fast record lookup
  - Support for record-level insertion
  - Support for updates (not in place)

- HBase updates are done by creating new versions of values

HBase vs. HDFS (Cont'd)



	Plain HDFS/MR	HBase
Write pattern	Append-only	Random write, bulk incremental
Read pattern	Full table scan, partition table scan	Random read, small range scan, or table scan
Hive (SQL) performance	Very good	4-5x slower
Structured storage	Do-it-yourself / TSV / SequenceFile / Avro / ?	Sparse column-family data model
Max data size	30+ PB	~1PB

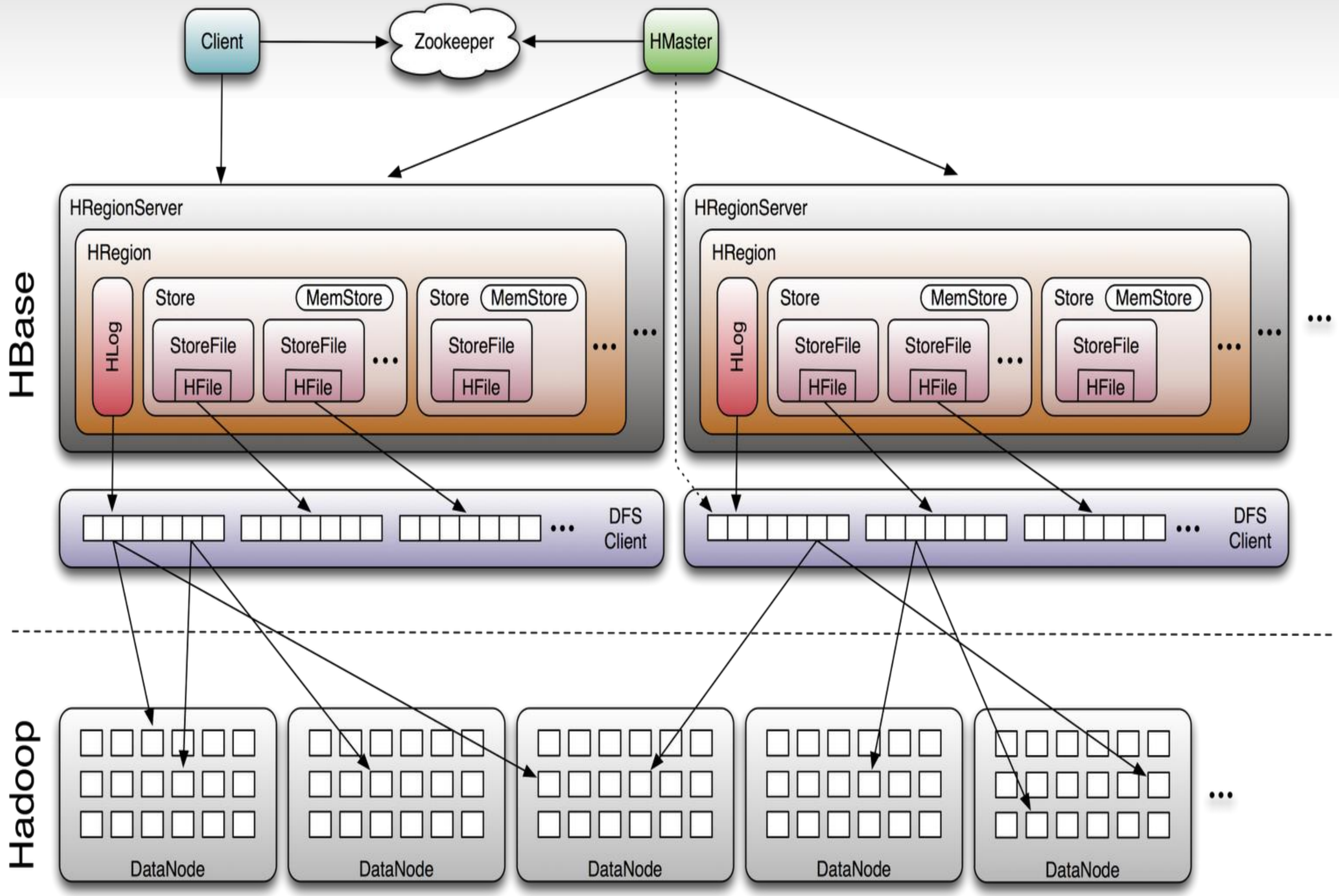
If application has neither random reads or writes ➔ Stick to HDFS

It is well suited for **real-time data processing or random read/write access** to large volumes of **data**.

Unlike relational database systems, **HBase** does not support a structured query language like SQL

Architecture

FORDHAM UNIVERSITY
THE JESUIT UNIVERSITY OF NEW YORK



FORDHAM UNIVERSITY
THE JESUIT UNIVERSITY OF NEW YORK

HBase Data Model

Storage Mechanism in HBase

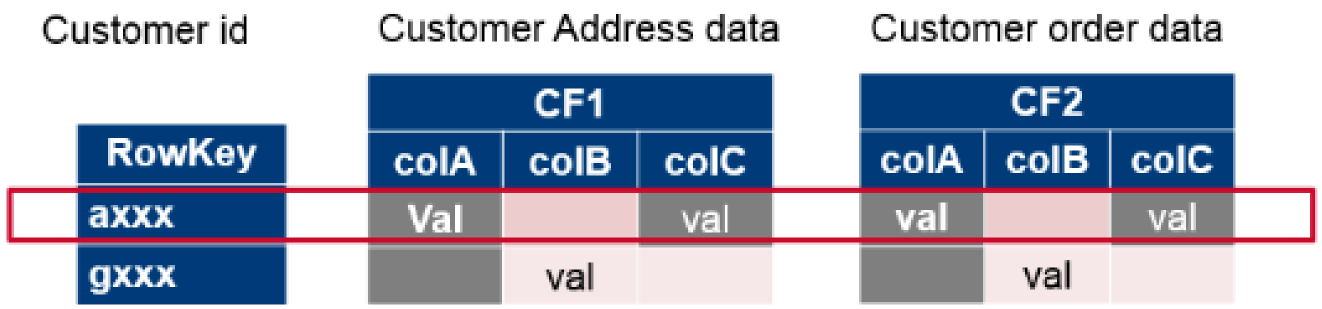
HBase is a **column-oriented database** and the tables in it are sorted by row. The table **schema defines only column families, which are the key value pairs**. A table have multiple column families and each column family can have any number of columns. Subsequent column values are stored contiguously 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

- HBase is referred to as a column family-oriented data store.
- It’s also row-oriented.
  - Each row is indexed by a key that you can use for lookup

column family: attribute in it is related eg, the information of a customer



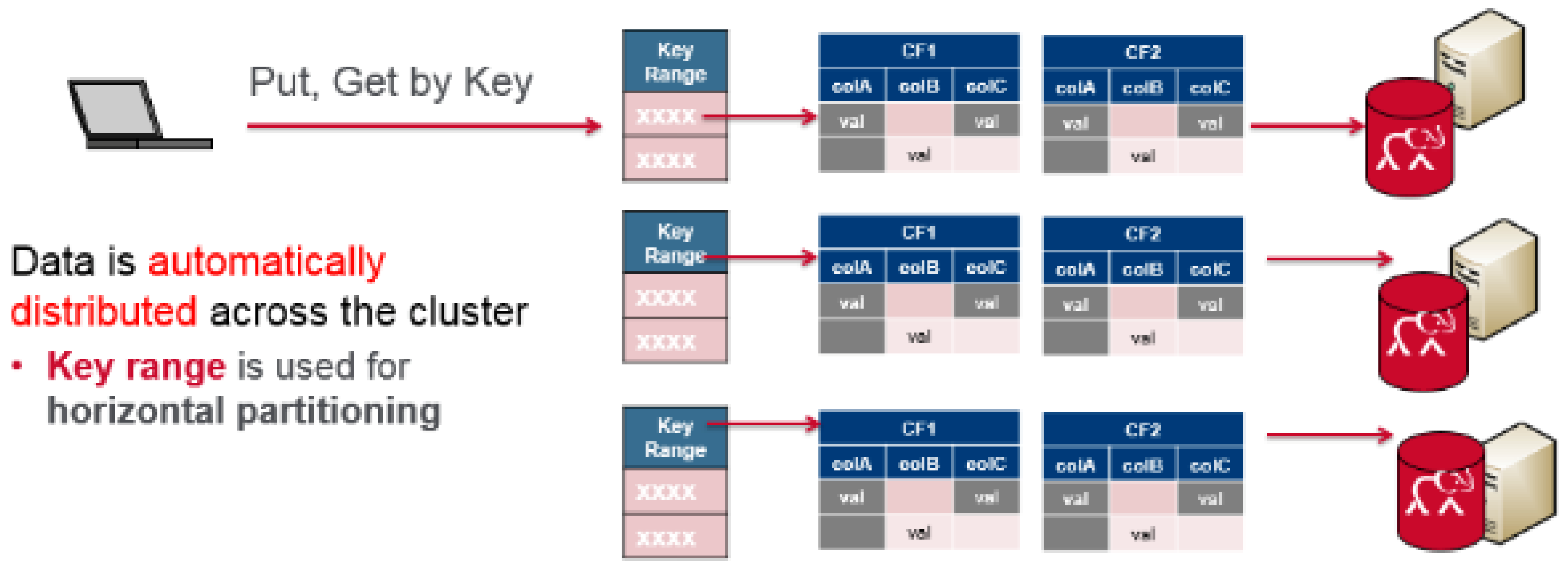
Customer id	Customer Address data			Customer order data		
	CF1			CF2		
RowKey	colA	colB	colC	colA	colB	colC
axxx	Val		val	val		val
gxxx		val			val	

Data is accessed and stored together:

- RowKey is the primary index
- Column Families group similar data by row key

HBase

- **HBase is a distributed database**

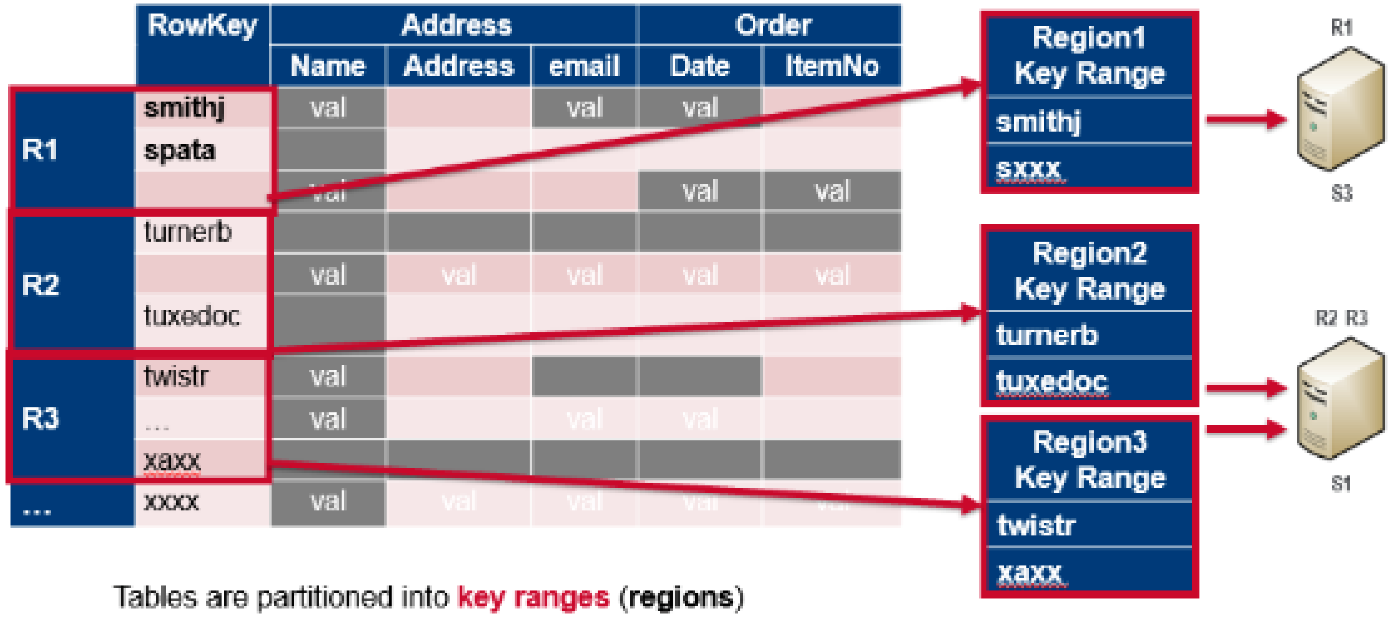


Data is automatically distributed across the cluster

- Key range is used for horizontal partitioning

“Tables” are partitioned into regions

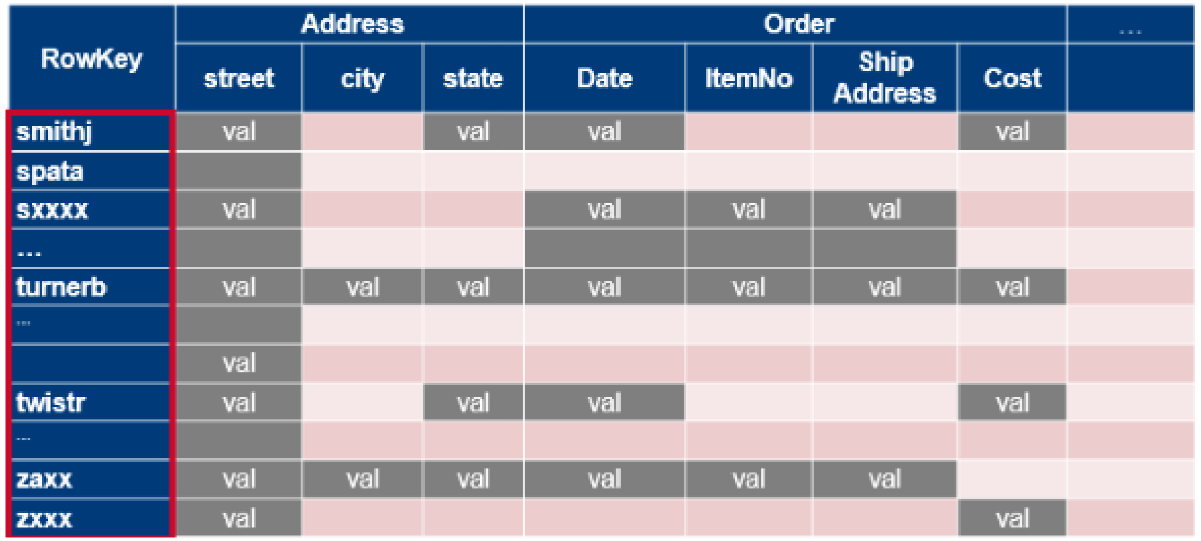
- “Tables” are divided into sequences of rows, by key range, called regions
- These regions are then assigned to the data nodes in the cluster called “RegionServers”
- This is done automatically and is how HBase was designed for horizontal sharding



Tables are partitioned into **key ranges** (**regions**)
Region= served by nodes (Region Servers)
Regions are spread across cluster

Row

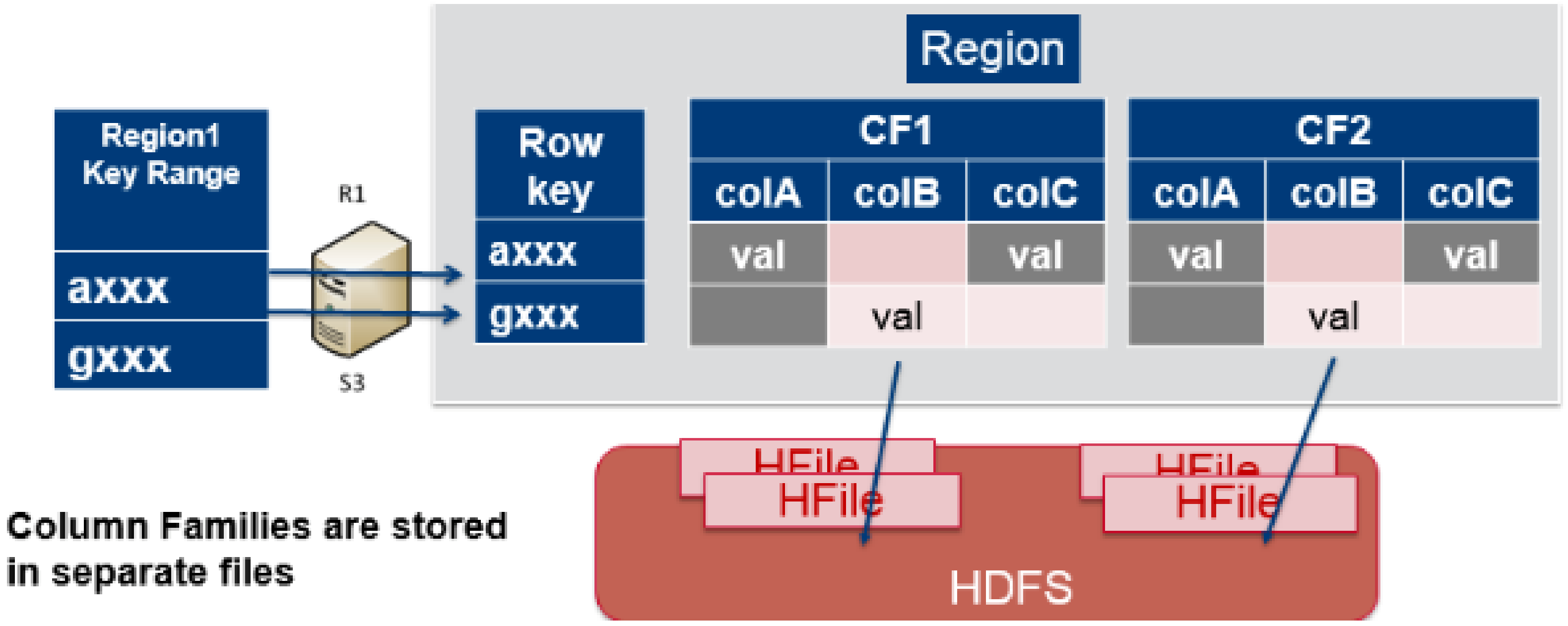
- **A row consists of a row key and one or more columns with values**
- **Records in HBase are stored in sorted order, according to rowkey (related rows are near each other)**



RowKey	Address			Order				…
	street	city	state	Date	ItemNo	Ship Address	Cost	
smithj	val		val	val			val	
spata								
sxxxx	val			val	val	val		
…								
turnerb	val	val	val	val	val	val	val	
…								
	val							
twistr	val		val	val			val	
…								
zaxx	val	val	val	val	val	val		
zxxx	val						val	

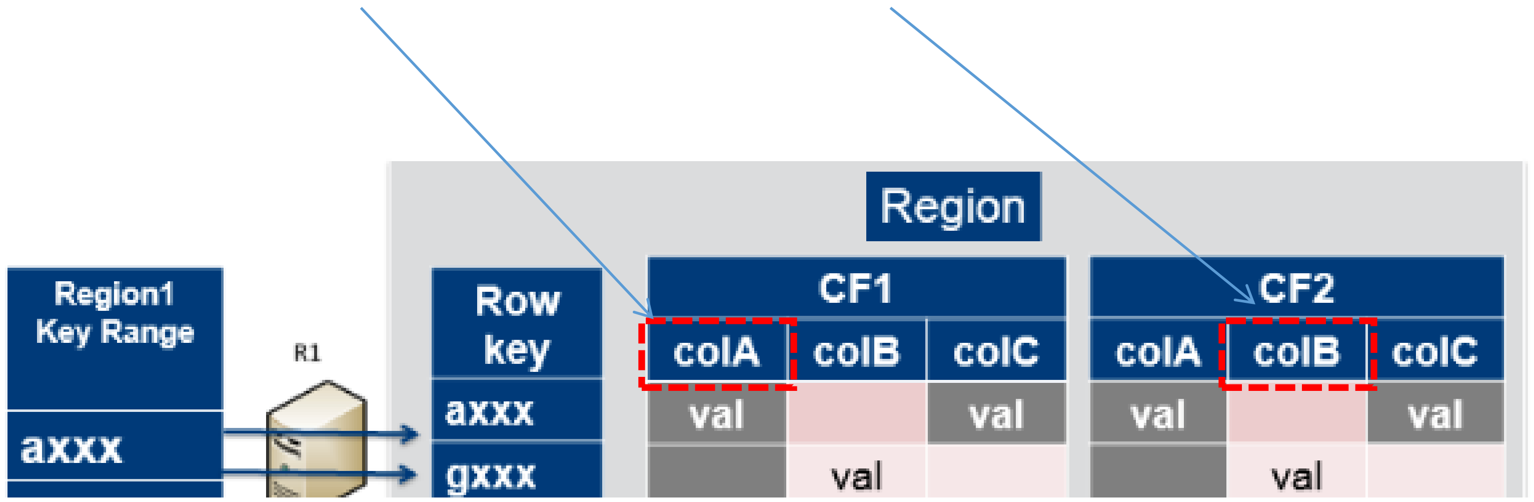
Column families

- Defined by the user and fixed at table creation
- Mapped to storage files
- Stored in separate files, which can also be accessed separately



Column

- **A column in HBase consists of a column family and a column qualifier, which are delimited by a : (colon) character**
  - ❑**E.g., CF1:colA, CF1:colB, CF2:colB**

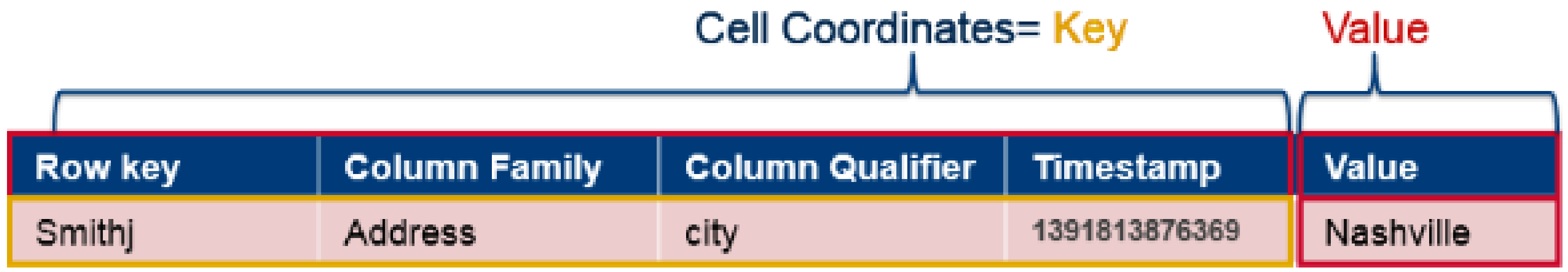


Cell and timestamp

- A cell is a combination of row, column family, and column qualifier, and contains a value and a timestamp (representing the value's version)
- A timestamp is written alongside each value, and is the identifier for a given version of a value.
  - By default, the timestamp represents the time on the RegionServer when the data was written
  - We can specify a different timestamp value when we put data into the cell

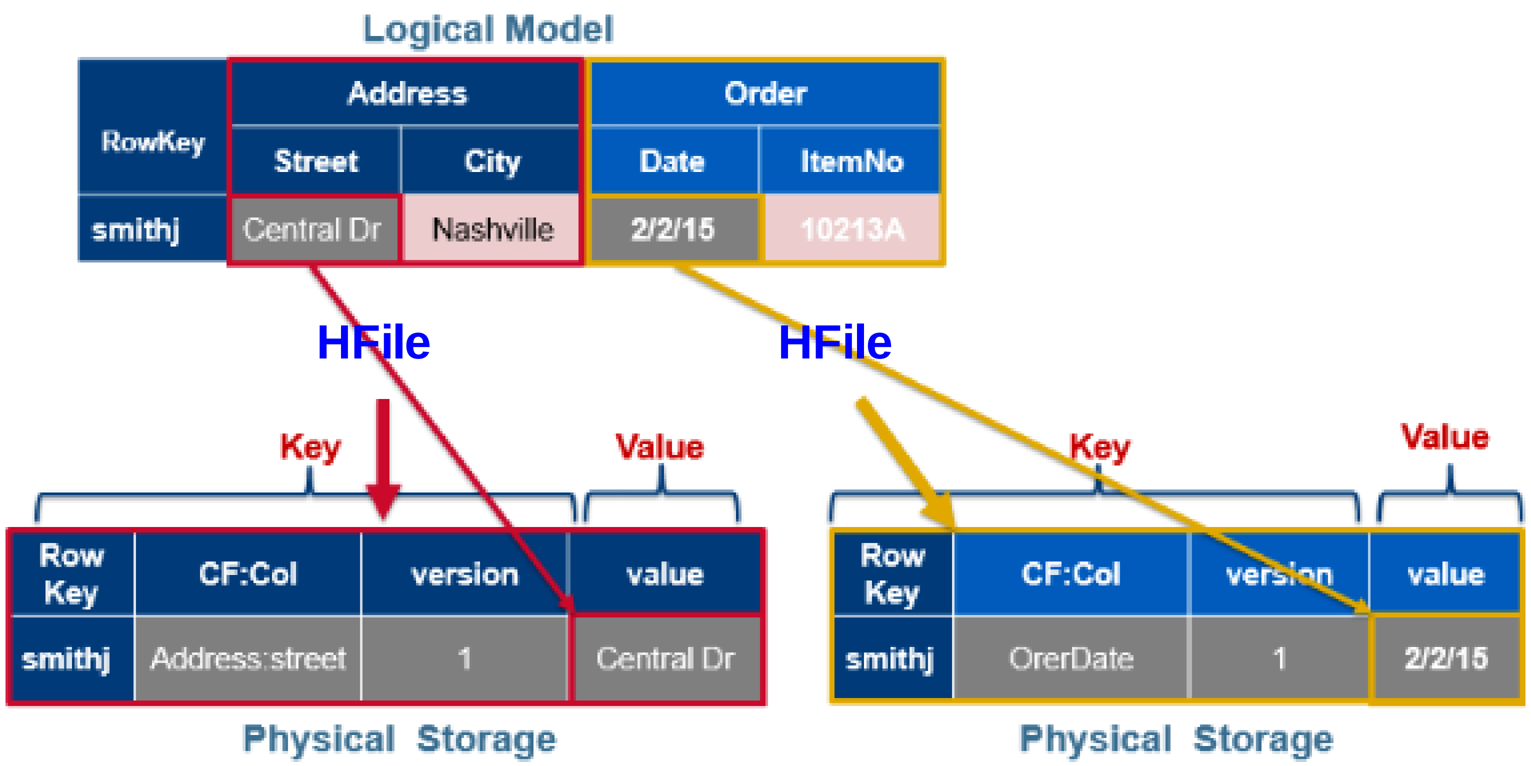
Data in HBase - **logical** view

How are data stored?



Row key	Column Family	Column Qualifier	Timestamp	Value
Smithj	Address	city	1391813876369	Nashville

Data in HBase – physical view



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

info and roles are two column family

"Roles" column family has different columns in different cells

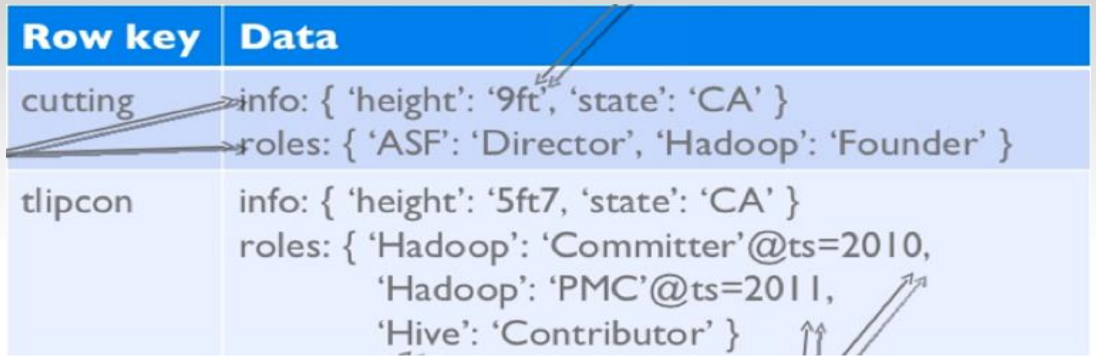
RowKey	**Value**

cutting	Info:{ 'height': '9ft', 'state': 'MD' }
	Roles:{ 'ASF': 'Director', 'Hadoop': 'Founder' }

tlipcon	Info:{ 'height': '5ft7', 'state': 'CA'}
	Roles:{ 'Hadoop': 'Committer',
	'Hadoop': 'Founder',
	'Hive': 'Contributor' }

committer and founder have differer

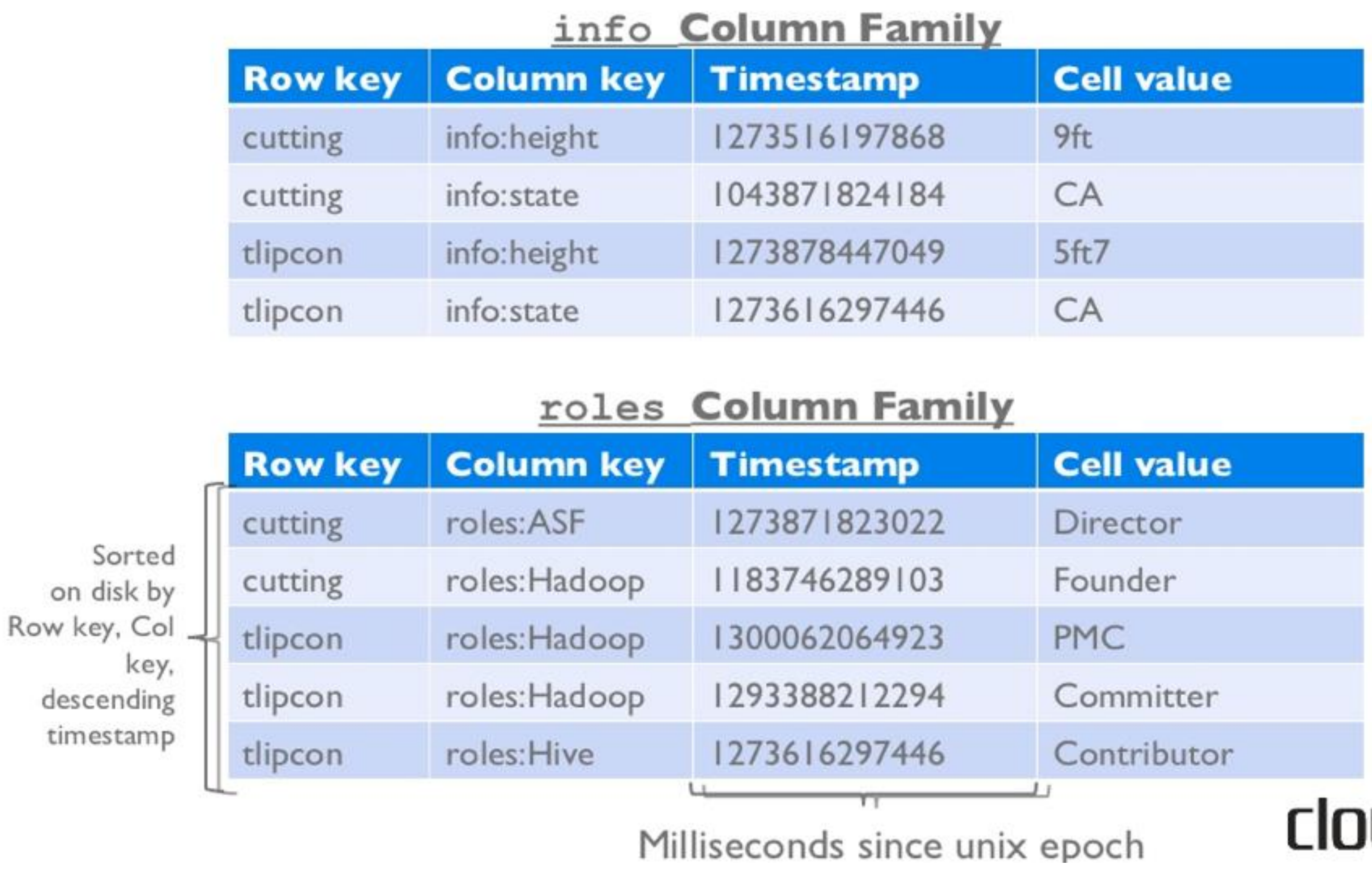
Example



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’ }


Sorted on disk by Row key, Col key, descending timestamp
Milliseconds since unix epoch

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

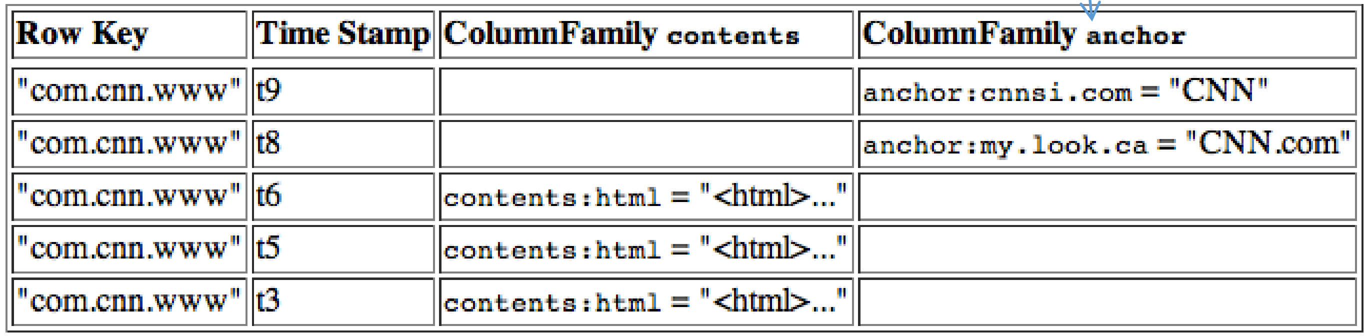
roles Column Family

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

Notes on data model (Cont'd)

- 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>..."	

Summary

Hbase is Not

- Not an SQL Database.
- Not relational.
- No joins.
- No fancy query language and no sophisticated query engine.

Features of HBase

- 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.

When Should I Use HBase?

make sure you have enough data. If you have hundreds of millions or billions of rows, then HBase is a good candidate.

Hive and Impala

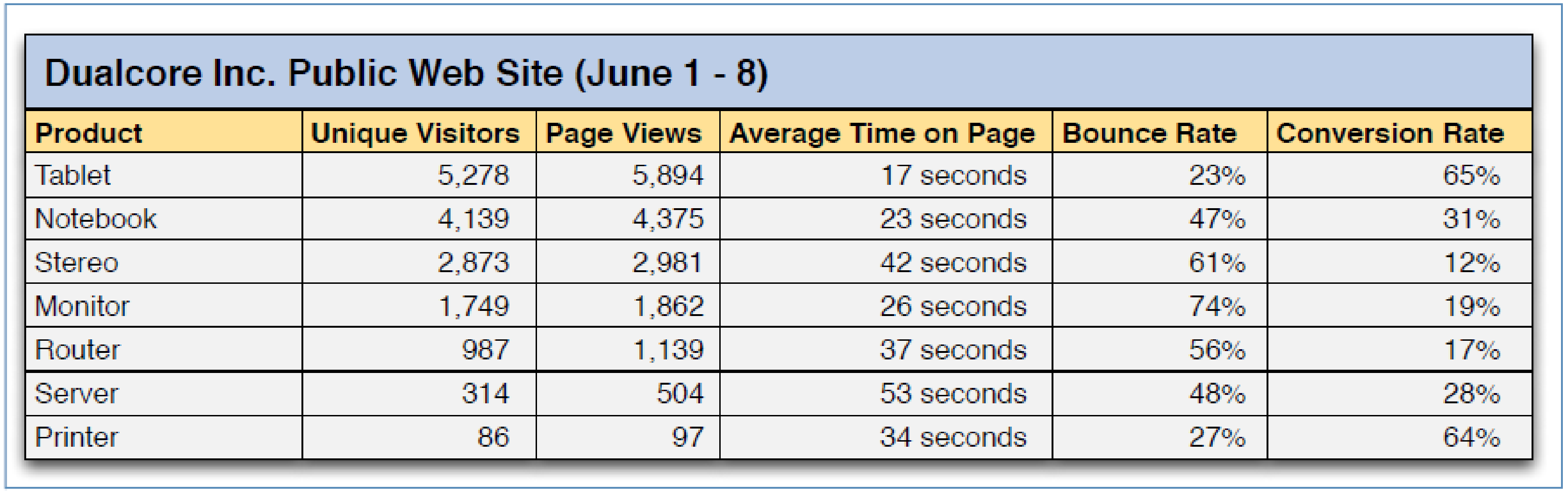
Impala more fast

hive used for query

Simple Use Case: Log Files

Use Case: Log File Analytics

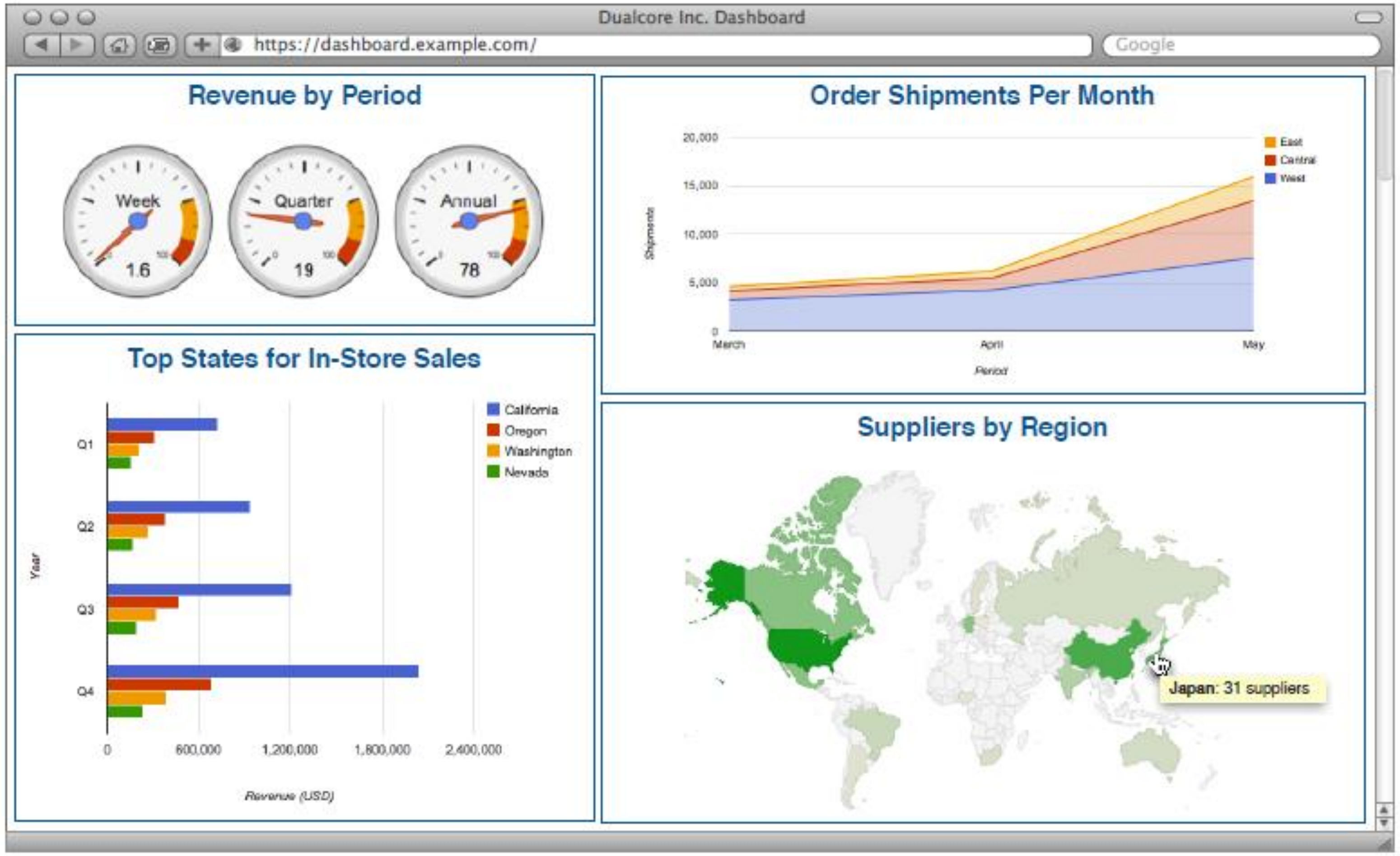
- **Server log files are an important source of data**
- **Hive and Impala allow you to treat a directory of log files like a table**
  - Allows SQL-like queries against raw data



Dualcore Inc. Public Web Site (June 1 - 8)

Product	Unique Visitors	Page Views	Average Time on Page	Bounce Rate	Conversion Rate
Tablet	5,278	5,894	17 seconds	23%	65%
Notebook	4,139	4,375	23 seconds	47%	31%
Stereo	2,873	2,981	42 seconds	61%	12%
Monitor	1,749	1,862	26 seconds	74%	19%
Router	987	1,139	37 seconds	56%	17%
Server	314	504	53 seconds	48%	28%
Printer	86	97	34 seconds	27%	64%

Use Case: Getting Data for Dashboards

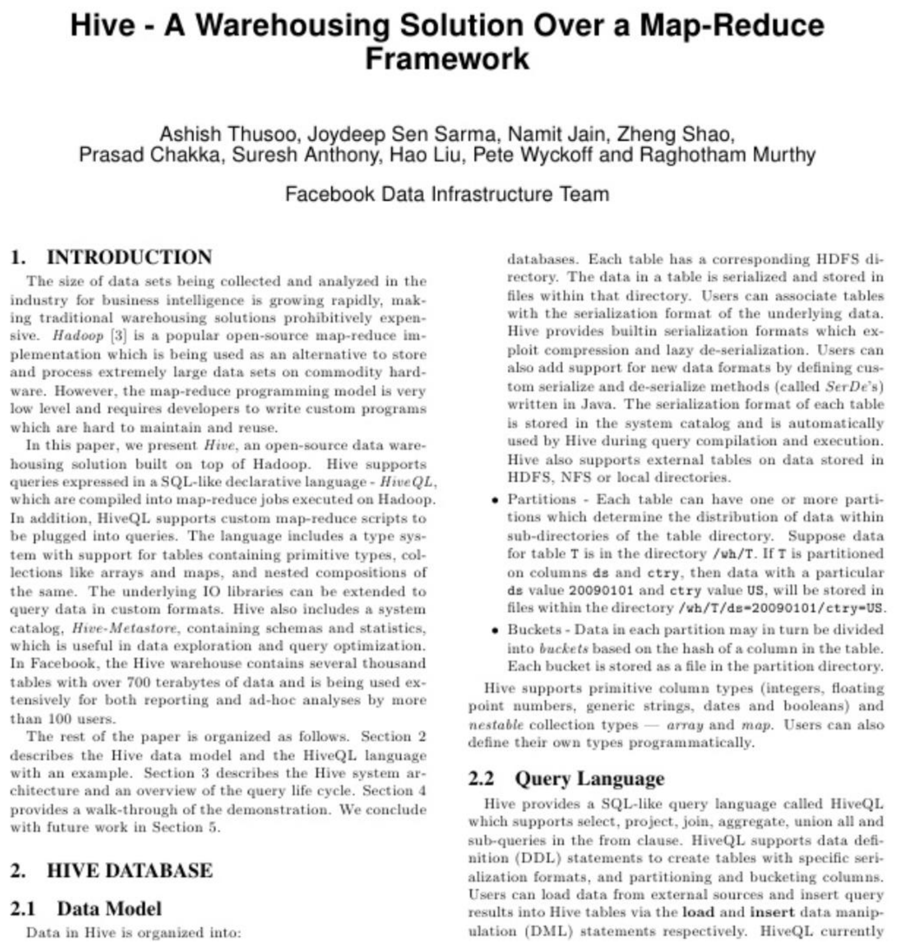


Why Another Data Warehousing System?

- Problem: Data, data and more data
  - 200GB per day in March 2008 back to 1TB compressed per day today
- The Hadoop Experiment
- Problem: Map/Reduce is great but every one is not a Map/Reduce expert
  - I know SQL and I am a python and php expert
- So what do we do: HIVE

Hive: A data warehouse on Hadoop

- Published by Facebook in VLDB conference 2009



Hive - A Warehousing Solution Over a Map-Reduce Framework

Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff and Raghotham Murthy

Facebook Data Infrastructure Team

1. INTRODUCTION

The size of data sets being collected and analyzed in the industry for business intelligence is growing rapidly, making traditional warehousing solutions prohibitively expensive. *Hadoop* [3] is a popular open-source map-reduce implementation which is being used as an alternative to store and process extremely large data sets on commodity hardware. However, the map-reduce programming model is very low level and requires developers to write custom programs which are hard to maintain and reuse.

In this paper, we present *Hive*, an open-source data warehousing solution built on top of Hadoop. Hive supports queries expressed in a SQL-like declarative language - *HiveQL*, which are compiled into map-reduce jobs executed on Hadoop. In addition, HiveQL supports custom map-reduce scripts to be plugged into queries. The language includes a type system with support for tables containing primitive types, collections like arrays and maps, and nested compositions of the same. The underlying IO libraries can be extended to query data in custom formats. Hive also includes a system catalog, *Hive-Metastore*, containing schemas and statistics, which is useful in data exploration and query optimization. In Facebook, the Hive warehouse contains several thousand tables with over 700 terabytes of data and is being used extensively for both reporting and ad-hoc analyses by more than 100 users.

The rest of the paper is organized as follows. Section 2 describes the Hive data model and the HiveQL language with an example. Section 3 describes the Hive system architecture and an overview of the query life cycle. Section 4 provides a walk-through of the demonstration. We conclude with future work in Section 5.

2. HIVE DATABASE

2.1 Data Model

Data in Hive is organized into:

databases. Each table has a corresponding HDFS directory. The data in a table is serialized and stored in files within that directory. Users can associate tables with the serialization format of the underlying data. Hive provides builtin serialization formats which exploit compression and lazy de-serialization. Users can also add support for new data formats by defining custom serialize and de-serialize methods (called *SerDe*'s) written in Java. The serialization format of each table is stored in the system catalog and is automatically used by Hive during query compilation and execution. Hive also supports external tables on data stored in HDFS, NFS or local directories.

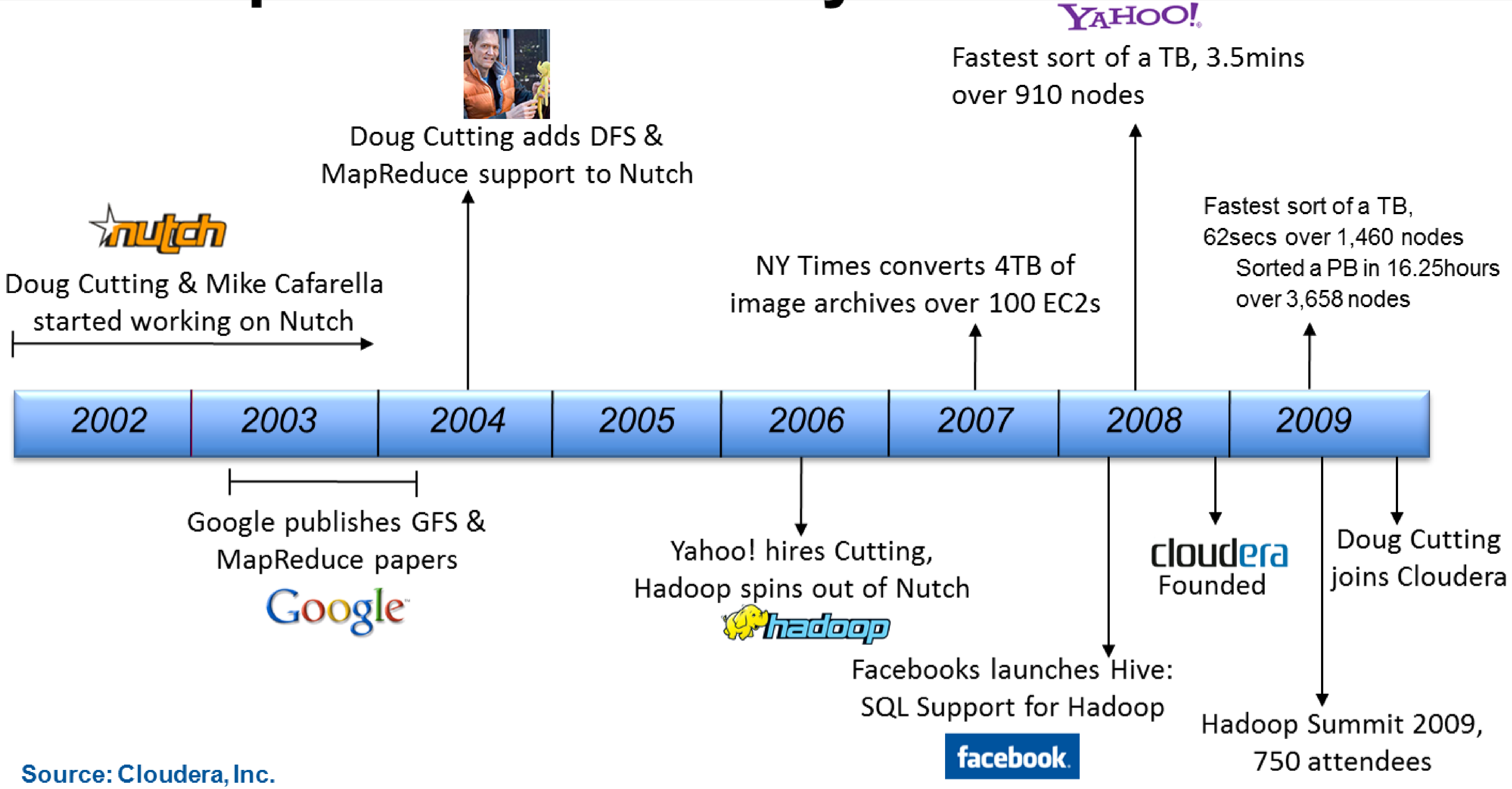
- Partitions - Each table can have one or more partitions which determine the distribution of data within sub-directories of the table directory. Suppose data for table `T` is in the directory `/wh/T`. If `T` is partitioned on columns `ds` and `ctry`, then data with a particular `ds` value `20090101` and `ctry` value `US`, will be stored in files within the directory `/wh/T/ds=20090101/ctry=US`.
- Buckets - Data in each partition may in turn be divided into *buckets* based on the hash of a column in the table. Each bucket is stored as a file in the partition directory.

Hive supports primitive column types (integers, floating point numbers, generic strings, dates and booleans) and *nestable* collection types — *array* and *map*. Users can also define their own types programmatically.

2.2 Query Language

Hive provides a SQL-like query language called HiveQL which supports select, project, join, aggregate, union all and sub-queries in the from clause. HiveQL supports data definition (DDL) statements to create tables with specific serialization formats, and partitioning and bucketing columns. Users can load data from external sources and insert query results into Hive tables via the **load** and **insert** data manipulation (DML) statements respectively. HiveQL currently

Hadoop Creation History



Doug Cutting & Mike Cafarella started working on Nutch

Google publishes GFS & MapReduce papers

Doug Cutting adds DFS & MapReduce support to Nutch

Yahoo! hires Cutting, Hadoop spins out of Nutch

NY Times converts 4TB of image archives over 100 EC2s

Facebooks launches Hive: SQL Support for Hadoop

Fastest sort of a TB, 3.5mins over 910 nodes

Founded

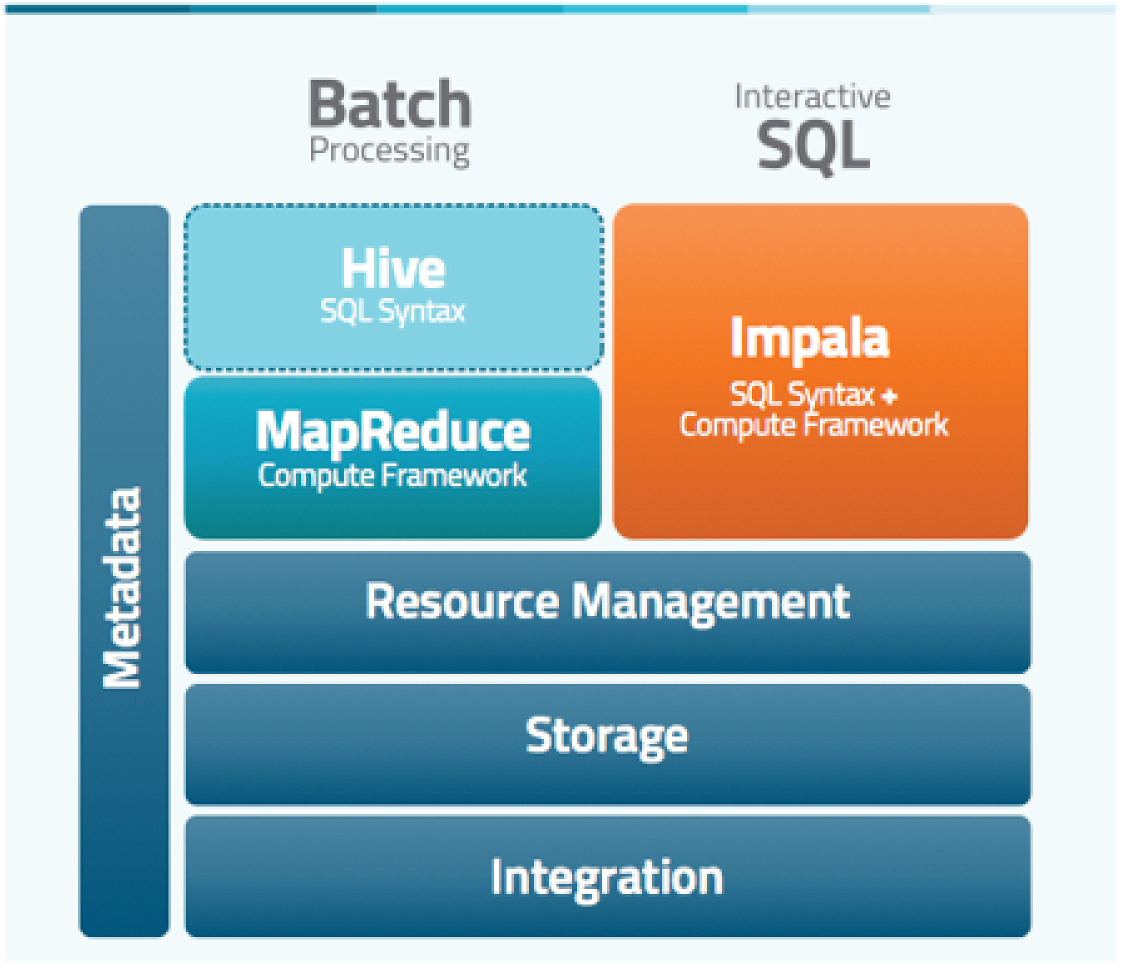
Fastest sort of a TB, 62secs over 1,460 nodes
Sorted a PB in 16.25hours over 3,658 nodes

Hadoop Summit 2009, 750 attendees

Doug Cutting joins Cloudera

Source: Cloudera, Inc.

Where do Impala and Hive sit?



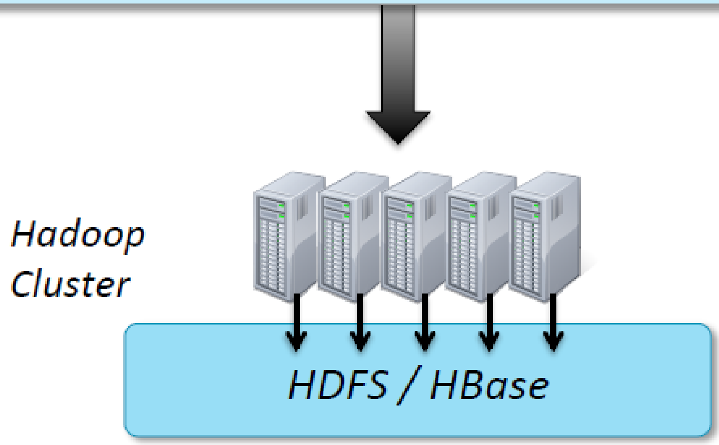
Hive processes data via MapReduce, Impala is a stand-alone MPP framework

What are Impala and Hive?

Introduction to Impala and Hive (1)

- **Impala and Hive are both tools that provide SQL querying of data stored in HDFS / HBase**

```
SELECT zipcode, SUM(cost) AS total
FROM customers
JOIN orders
ON (customers.cust_id = orders.cust_id)
WHERE zipcode LIKE '63%'
GROUP BY zipcode
ORDER BY total DESC;
```



FORDHAM UNIVERSITY
THE JESUIT UNIVERSITY OF NEW YORK

Introduction to Impala and Hive (2)

- **Apache Hive is a high-level abstraction on top of MapReduce**
  - Uses HiveQL
  - Generates MapReduce or Spark* jobs that run on the Hadoop cluster
  - Originally developed at Facebook around 2007
    - Now an open-source Apache project
- **Cloudera Impala is a high-performance dedicated SQL engine**
  - Uses Impala SQL
  - Inspired by Google's Dremel project
  - Query latency measured in milliseconds
  - Developed at Cloudera in 2012
    - Open-source with an Apache license

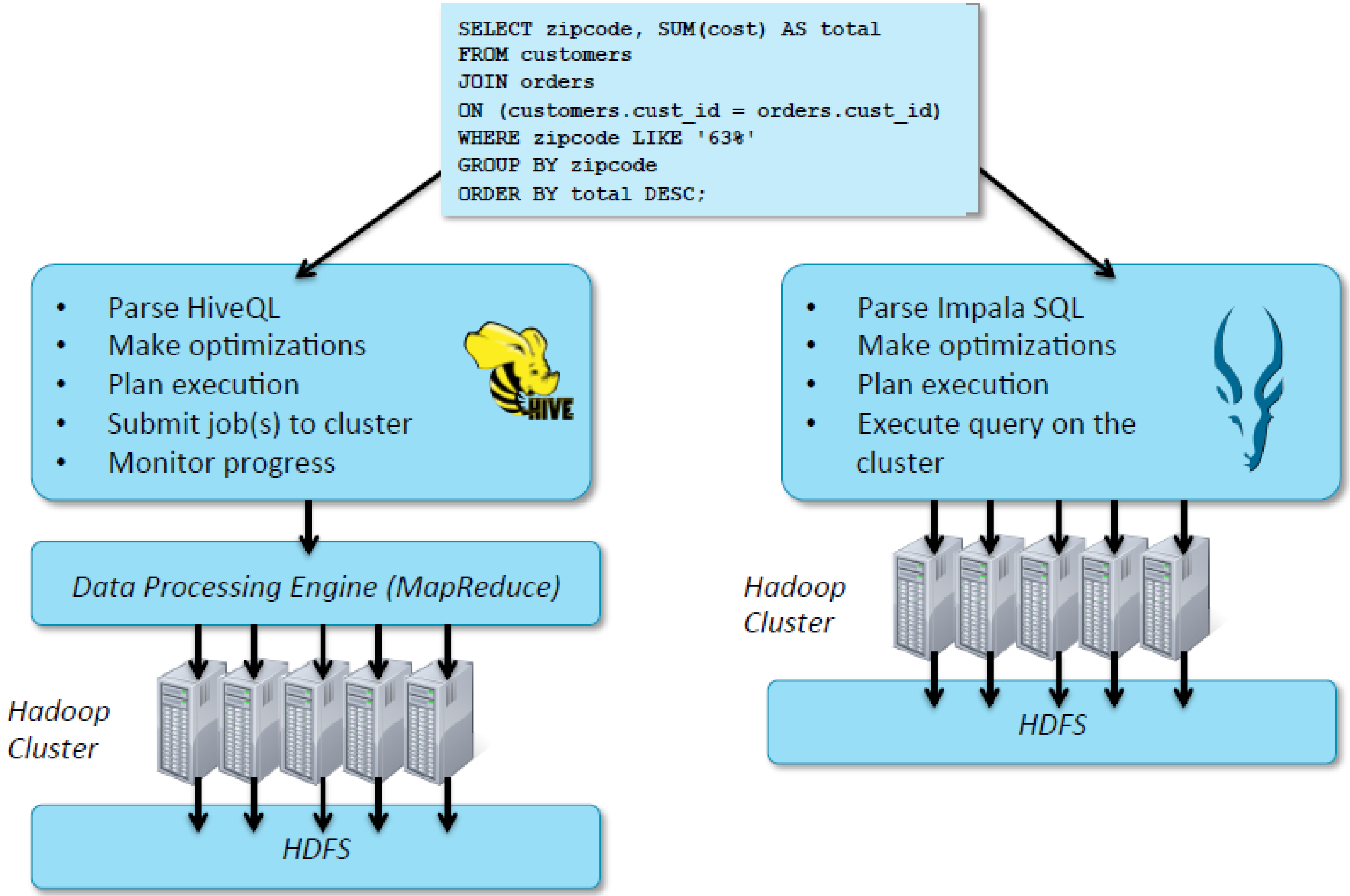
* Hive-on-Spark is currently in beta testing

Differences between Hive and Impala

- **Hive has more features**
  - E.g. Complex data types (arrays, maps) and full support for windowing analytics
  - Highly extensible
  - Commonly used for batch processing
- **Impala is much faster**
  - Specialized SQL engine offers 5x to 50x better performance
  - Ideal for interactive queries and data analysis
  - More features being added over time

Note that Impala does not process the data using MapReduce or Spark, instead it executes the query on the cluster. This feature makes Impala faster than Hive.

High-Level Overview



```
SELECT zipcode, SUM(cost) AS total
FROM customers
JOIN orders
ON (customers.cust_id = orders.cust_id)
WHERE zipcode LIKE '63%'
GROUP BY zipcode
ORDER BY total DESC;
```

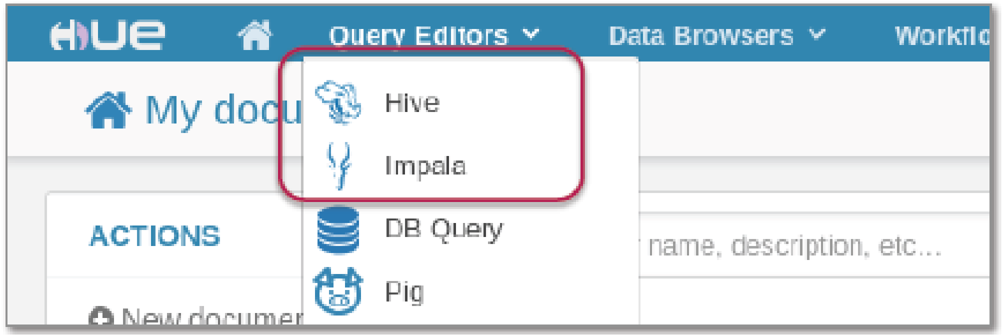
Why Use Hive and Impala?

- **Brings large-scale data analysis to a broader audience**
  - No software development experience required
  - Leverage existing knowledge of SQL
- **More productive than writing MapReduce or Spark directly**
  - Five lines of HiveQL/Impala SQL might be equivalent to 200 lines or more of Java
- **Offers interoperability with other systems**
  - Extensible through Java and external scripts
  - Many business intelligence (BI) tools support Hive and/or Impala

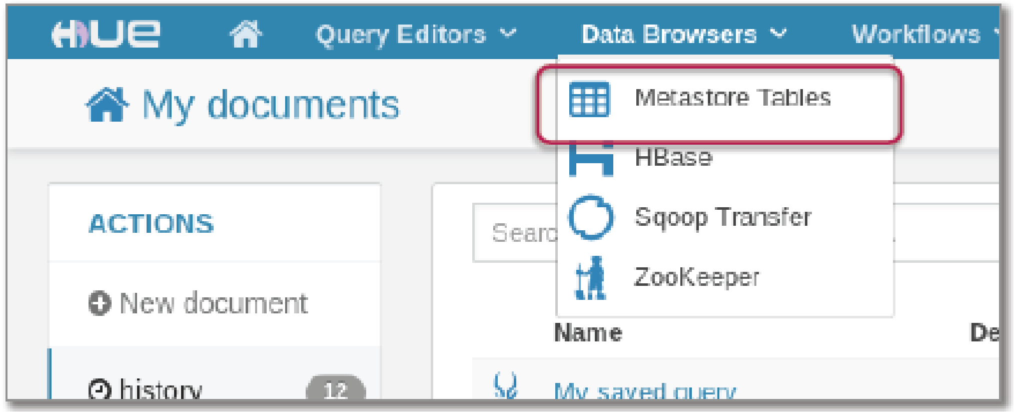
Using HUE with Hive and Impala

You can use Hue to...

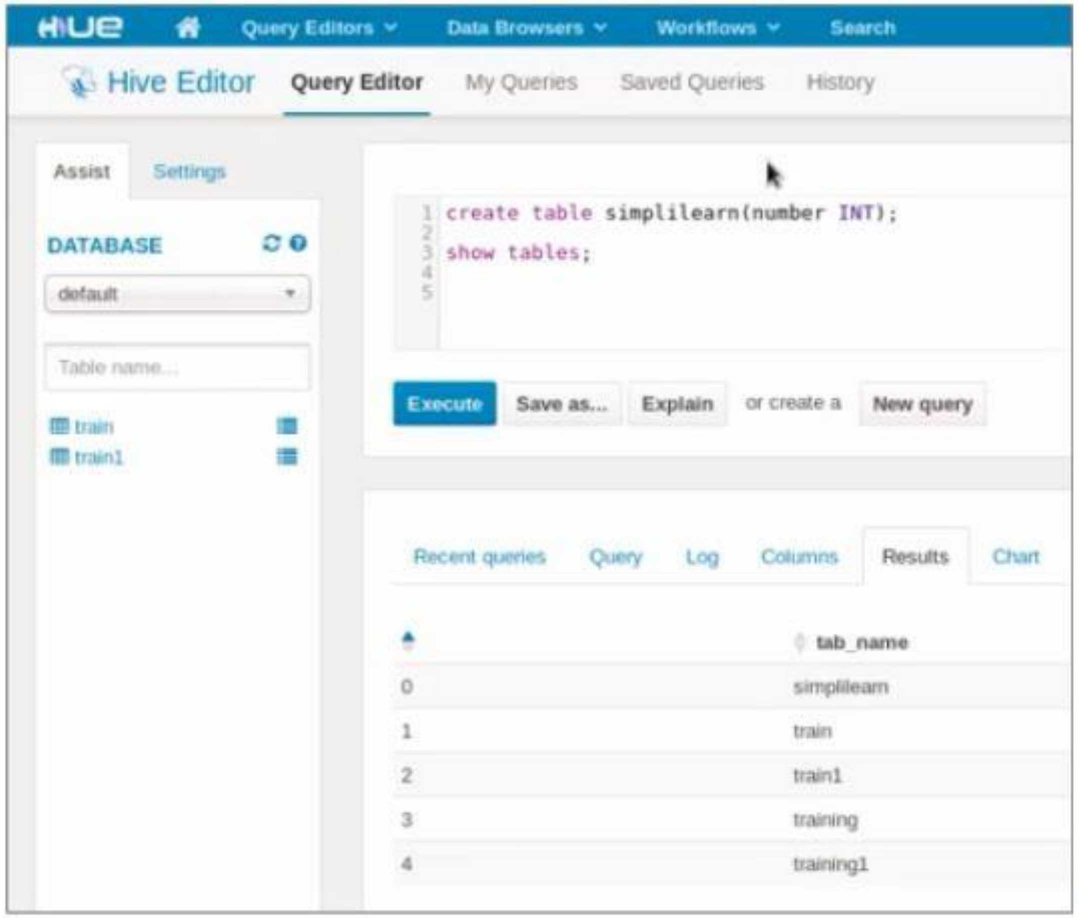
Query data with Hive or Impala



View and manage the Metastore



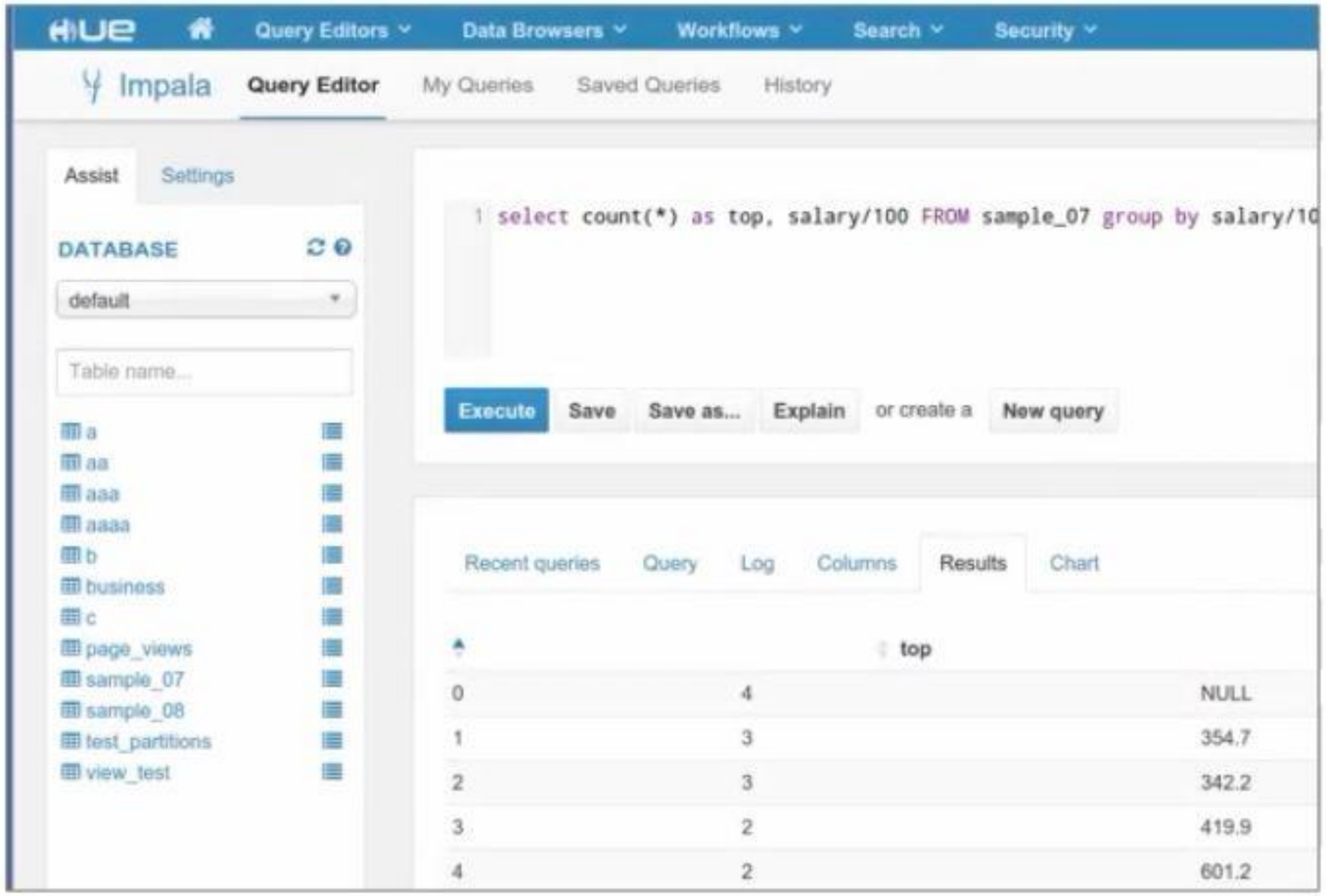
Both are available on Cloudera



```
create table simplilearn(number INT);

show tables;
```

	tab_name
0	simplilearn
1	train
2	train1
3	training
4	training1



```
select count(*) as top, salary/100 FROM sample_07 group by salary/10
```

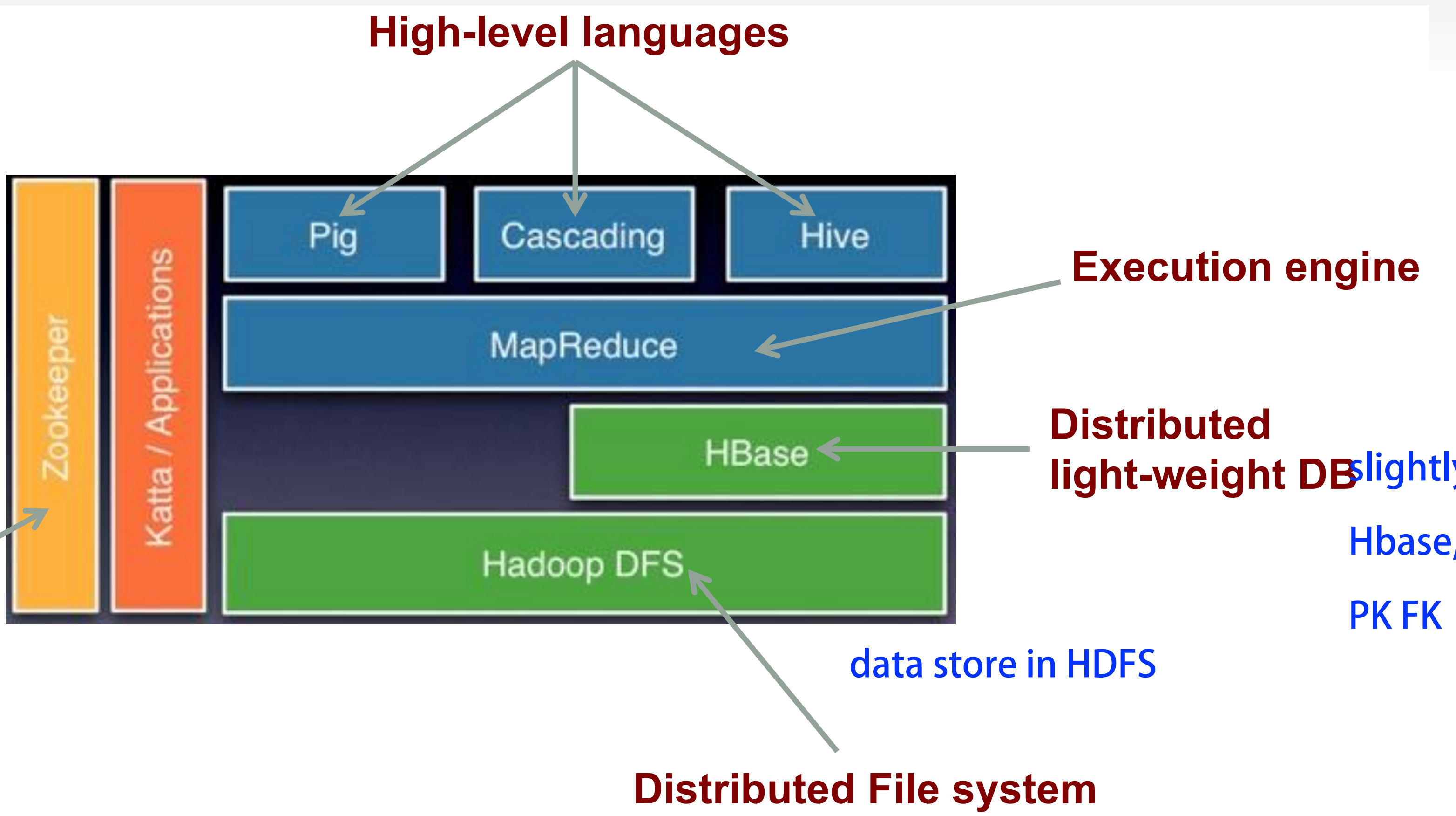
	top	
0	4	NULL
1	3	354.7
2	3	342.2
3	2	419.9
4	2	601.2

The cluster is not a DBMS…

write sql on Hbase, have to use java first

	Relational Database	Hive	Impala
Query language	SQL (full)	SQL (subset)	SQL (subset)
Update individual records	Yes	No	No
Delete individual records	Yes	No	No
Transactions	Yes	No	No
Index support	Extensive	Limited	No
Latency	Very low	High	Low
Data size	Terabytes	Petabytes	Petabytes

Recall: Hadoop big picture



slightly data are stored in Hbase, one table with no PK FK

data store in HDFS

HDFS + MapReduce are enough to have things working

HQL – See Tutorial

```
CREATE EXTERNAL TABLE webpage
   (page_id SMALLINT,
    name STRING,
    assoc_files STRING)
   ROW FORMAT DELIMITED
      FIELDS TERMINATED BY '\t'
      LOCATION '/loudacre/webpage'
```

```
SELECT * FROM webpage WHERE name LIKE "ifruit%"
```

Questions?