Inspire…Educate…Transform.

Big Data

Session 5: No SQL, Scripting (Hive, Pig)

Suryaprakash Kompalli
Senior Mentor, INSOFE

Review of Last Lecture

- Class 1:
  - Different architectures
  - Transition from Databases to data warehouses and data lakes
  - Thinking of Large Jobs as Task Decompositions
  - How BigData is changing IT and business operations
- Class 2:
  - Hadoop: Storage

Review of Last Lecture

- Class 3:
  - Map-Reduce, YARN
  - Spark

Agenda

- NoSQL:
  - Hbase
- Scripting in Big Data:
  - Hive / Pig
  - Other tools
    - Impala
    - Spark SQL
    - Apache Drill

Agenda

- No-SQL
- HBase

There is a misnomer with this name.

SQL refers to “Structured Query Langauge”, an ANSI standard. It was originally also referred to as “Structured English Query Language”, or SEQUEL.

RDBMS evolved to manage data, and allow execution of SQL queries.

But these systems were not suitable to serve the types of queries that can (or need to) be executed on unstructured data

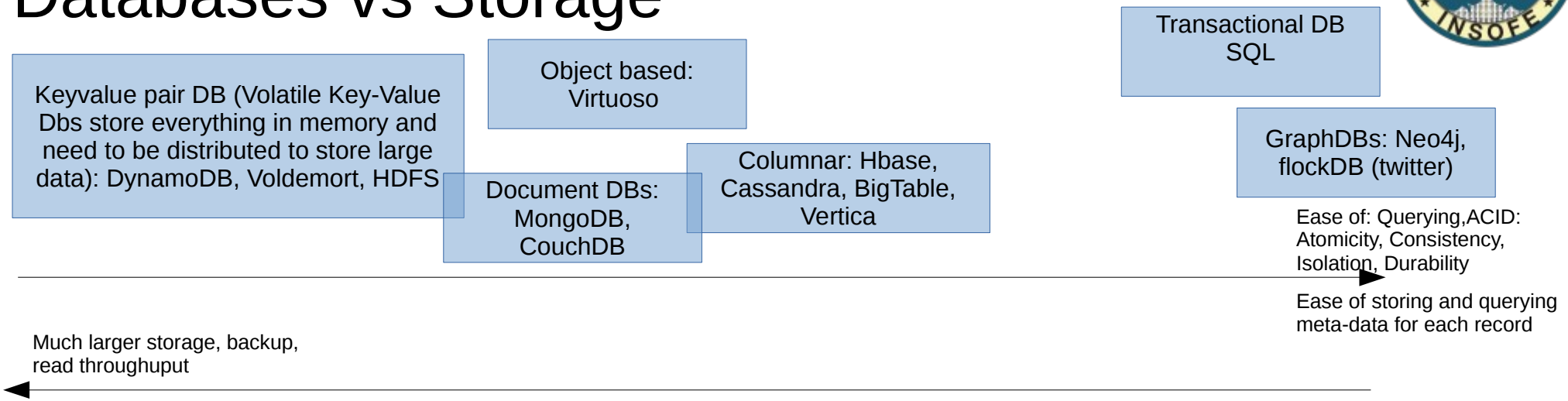
NoSQL is meant to represent Not-Only-SQL

But most of us confuse it with Not-SQL :)

Story does not end there; many columnar and Key-value pair DBs provide SQL like querying: Apache Drill is a SQL query engine that can talk to multiple data sources, including HADOOP

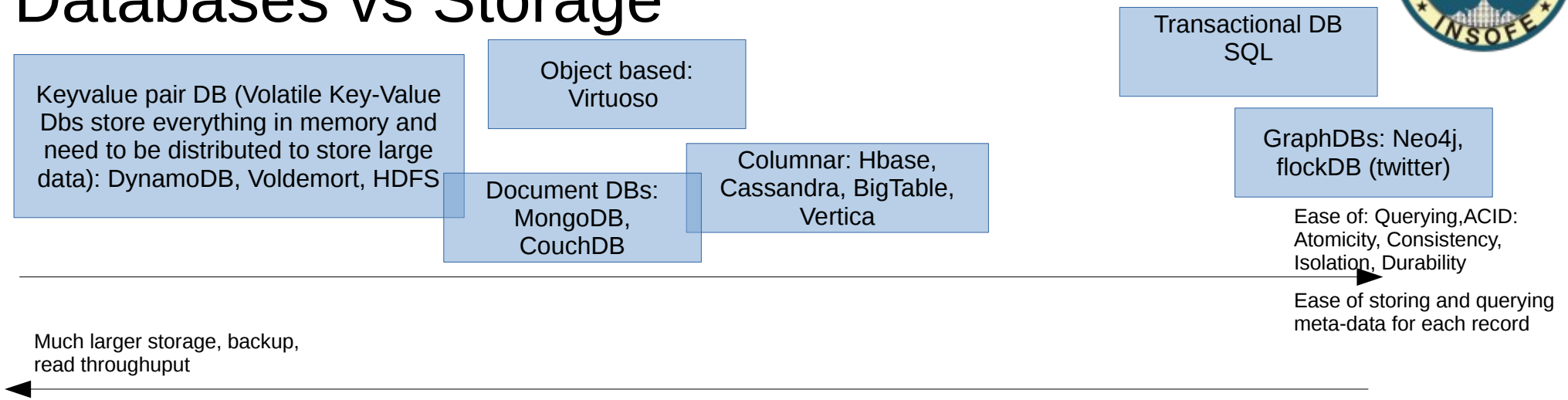
Some Not-Only-SQL examples: queries on flat file contents, GPS coordinates, image and video meta-tags

Databases vs Storage



- Key value pair storage provides great write throughput, but querying can be bad
  - Key-Value pair DBs were first to come up. These were followed by many other db's; we can consider them to be heirarchical key-value Dbs.
- For example, Columnar DBs have depth 3 (HBase):
  - All data is divided into rows,
  - Each row made of multiple columns, different rows may have different columns
  - Each column has multiple versions
  - Example: In a crawler, Each web server/top domain corresponds to one row. Each column is a page in the web server. Data at different time stamps are stored within the columns as different versions
    - A crawler may choose to keep old+new versions of the page, in which case they will be added as new records in a particular column/row combination
    - Another system may choose to keep only the newest version, in which case the DB will have many rows with variable columns

Databases vs Storage



- Many of these DBs have been purpose built, but industry has found new uses for them
  - For Ex. Document DBs were developed as a cross between Object/Key-value and Columnar DBs:
    - They tend to have a structure like document header, summary (abstract), sections, sub-sections etc.
    - The structure is implicitly stored in Json files
- Many DBs use RDBMS constructs behind the scene; for example neo4j also uses an RDBMS design

NoSql: Example #1: Web log analysis

Each record: UserID, URL, timestamp, additional-info

Separate records: UserID, name, age, gender, …

Task: Find average age of user accessing given URL

SQL like querying

No SQL: Example #2: Social-network graph

Each record: $UserID_1$, $UserID_2$

Separate records: UserID, name, age, gender, …

Task: Find all friends of friends of friends of … friends of given user

No SQL: Example #3: Wikipedia

Large collection of documents

Combination of structured and unstructured data

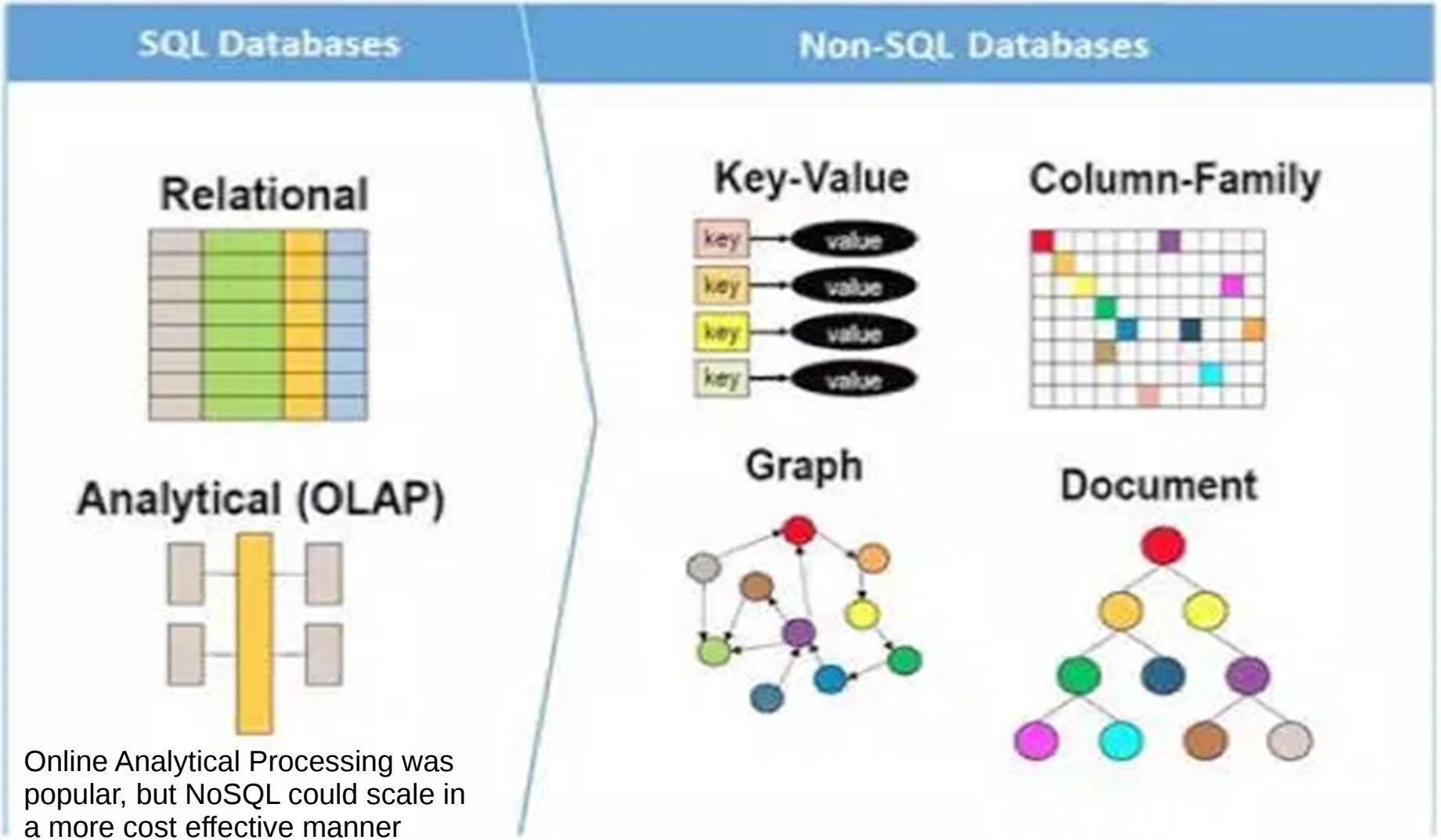
Task: Retrieve introductory paragraph of all pages about U.S. presidents before 1900

Querying files using document structure

NoSQL Taxonomy

- Key/Value
- Document
- Column
- Graph
- Others
  - Geospatial
  - File System
  - Object

A Visual Comparison



Online Analytical Processing was popular, but NoSQL could scale in a more cost effective manner

http://nosql-database.org/
Lists >225 No-SQL databases today.

Popular NoSQL DBs

Graph	Column	Document	Persistent Key/Value	Volatile Key/Value
neo4j	BigTable (Google)	MongoDB (~BigTable)	Dynamo (Amazon)	memcached
FlockDB (Twitter)	HBase (BigTable)	CouchDB	Voldemort (Dynamo)	Hazelcast
InfiniteGraph	Cassandra (Dynamo + BigTable)	Riak (Dynamo)	Redis	
	Hypertable (BigTable)		Membase (memcached)	
	SimpleDB (AmazonAWS)		Tokyo Cabinet	

The NoSQL movement

- Three major papers were the seeds of the NoSQL movement:
  - BigTable (Google)
  - Dynamo (Amazon)
    - Gossip protocol (discovery and error detection)
    - Distributed key-value data store
    - Eventual consistency
  - CAP Theorem (Consistency, Availability, Partition tolerance)

SQL – ACID Properties

- **A**tomic – All of the work in a transaction completes (commit) or none of it completes
- **C**onsistent – A transaction transforms the database from one consistent state to another consistent state. Consistency is defined in terms of constraints. (Usually enforced by the DB Schema)
- **I**solated – The results of any changes made during a transaction are not visible until the transaction has committed.
- **D**urable – The results of a committed transaction survive failures

NoSQL Theory

- Usually do not require a fixed table schema nor do they use the concept of joins
- All NoSQL offerings relax one or more of the ACID properties: Atomicity, consistency, isolation, durability
  - BASE (**B**asically **A**vailable, **S**oft state, **E**ventual consistency)

One overhead of join is that there should be well-defined schemas for each table or data structure and common columns across tables. Both are relaxed in NoSQL

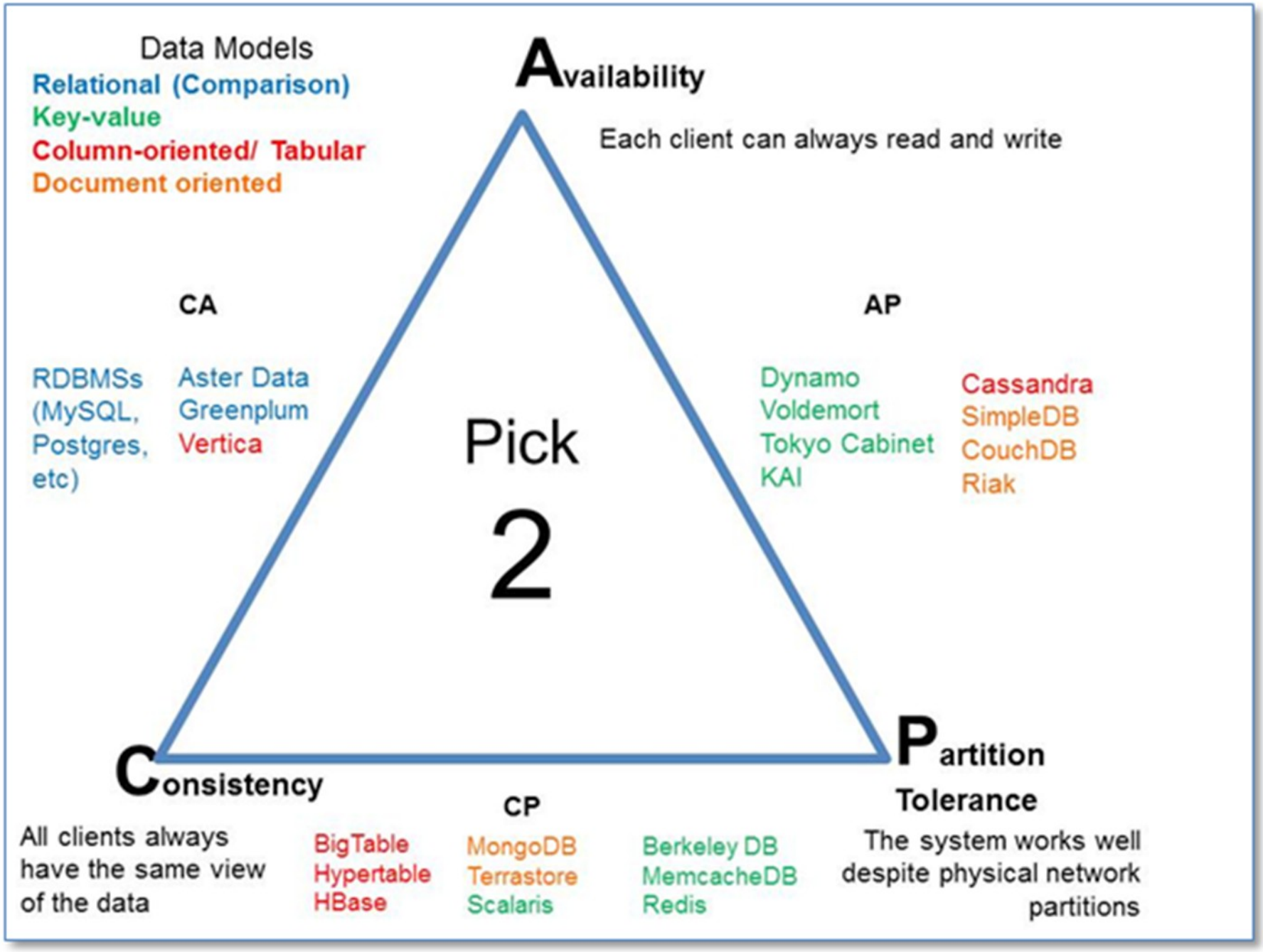
CAP Theorem

It is impossible for a distributed computer system to simultaneously provide all three of the following guarantees:

- ***Consistency****:* all nodes see the same data at the same time
- ***Availability****:* a guarantee that every request receives a response on whether it was successful or failed
- ***Partition tolerance****:* the system continues to operate despite arbitrary message loss or failure of part of the system

According to the theorem, a distributed system can satisfy any two of these guarantees at the same time, but not all three.

We will see how consistency is converted to eventual consistency in HBase


Data Models
Relational (Comparison)
Key-value
Column-oriented/ Tabular
Document oriented
Availability
Each client can always read and write
CA
RDBMSs (MySQL, Postgres, etc)
Aster Data
Greenplum
Vertica
AP
Dynamo
Voldemort
Tokyo Cabinet
KAI
Cassandra
SimpleDB
CouchDB
Riak
Pick
2
Consistency
All clients always have the same view of the data
CP
BigTable
Hypertable
HBase
MongoDB
Terrastore
Scalaris
Berkeley DB
MemcacheDB
Redis
Partition Tolerance
The system works well despite physical network partitions

Popularity Ranking

299 systems in ranking, March 2016

Rank Mar 2016	Rank Feb 2016	Rank Mar 2015	DBMS	Database Model	Score Mar 2016	Score Feb 2016	Score Mar 2015
1.	1.	1.	Oracle	Relational DBMS	1472.01	-4.13	+2.93
2.	2.	2.	MySQL	Relational DBMS	1347.71	+26.59	+86.62
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1136.49	-13.73	-28.31
4.	4.	4.	MongoDB	Document store	305.33	-0.27	+30.32
5.	5.	5.	PostgreSQL	Relational DBMS	299.62	+10.97	+35.19
6.	6.	6.	DB2	Relational DBMS	187.94	-6.55	-10.91
7.	7.	7.	Microsoft Access	Relational DBMS	135.03	+1.95	-6.66
8.	8.	8.	Cassandra	Wide column store	130.33	-1.43	+23.02
9.	↑ 10.	↑ 10.	Redis	Key-value store	106.22	+4.14	+9.17
10.	↓ 9.	↓ 9.	SQLite	Relational DBMS	105.77	-1.01	+4.06
11.	↑ 12.	↑ 15.	Elasticsearch	Search engine	80.17	+2.33	+21.24
12.	↓ 11.	↓ 11.	SAP Adaptive Server	Relational DBMS	76.64	-3.39	-8.72
13.	13.	13.	Teradata	Relational DBMS	74.07	+0.69	+1.29
14.	14.	↓ 12.	Solr	Search engine	69.37	-2.91	-12.52
15.	↑ 16.	↓ 14.	HBase	Wide column store	52.41	+0.39	-8.32
16.	↓ 15.	↑ 17.	Hive	Relational DBMS	50.51	-2.26	+11.18
17.	17.	↓ 16.	FileMaker	Relational DBMS	47.93	+0.90	-4.41
18.	18.	↑ 19.	Splunk	Search engine	43.73	+0.90	+8.01
19.	19.	↑ 21.	SAP HANA	Relational DBMS	39.99	+1.91	+7.82
20.	↑ 21.	↑ 23.	Neo4j	Graph DBMS	32.36	+0.07	+4.73

http://db-engines.com/en/ranking

No-SQL

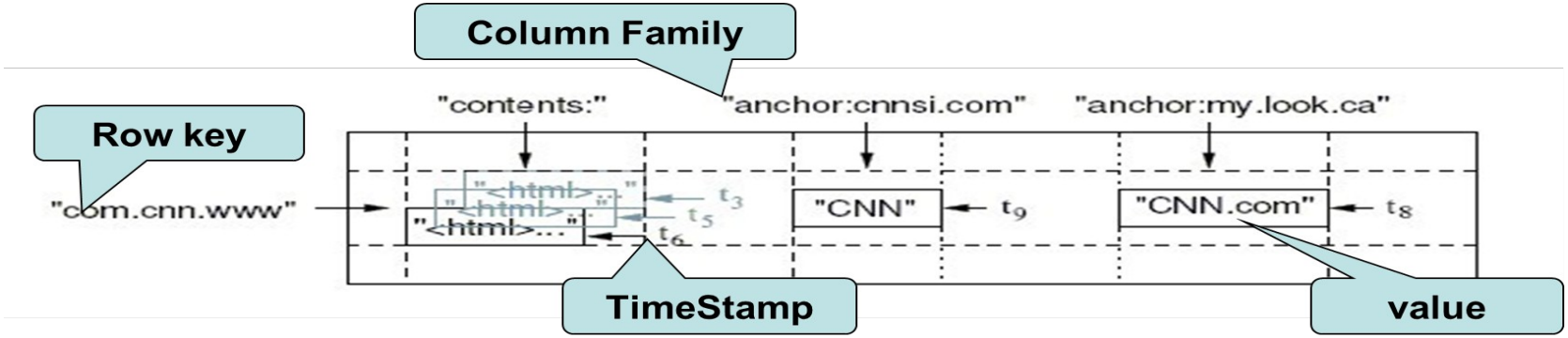
Google

An **open-source**, **distributed**, **column-oriented** database built on top of HDFS based on **BigTable**!

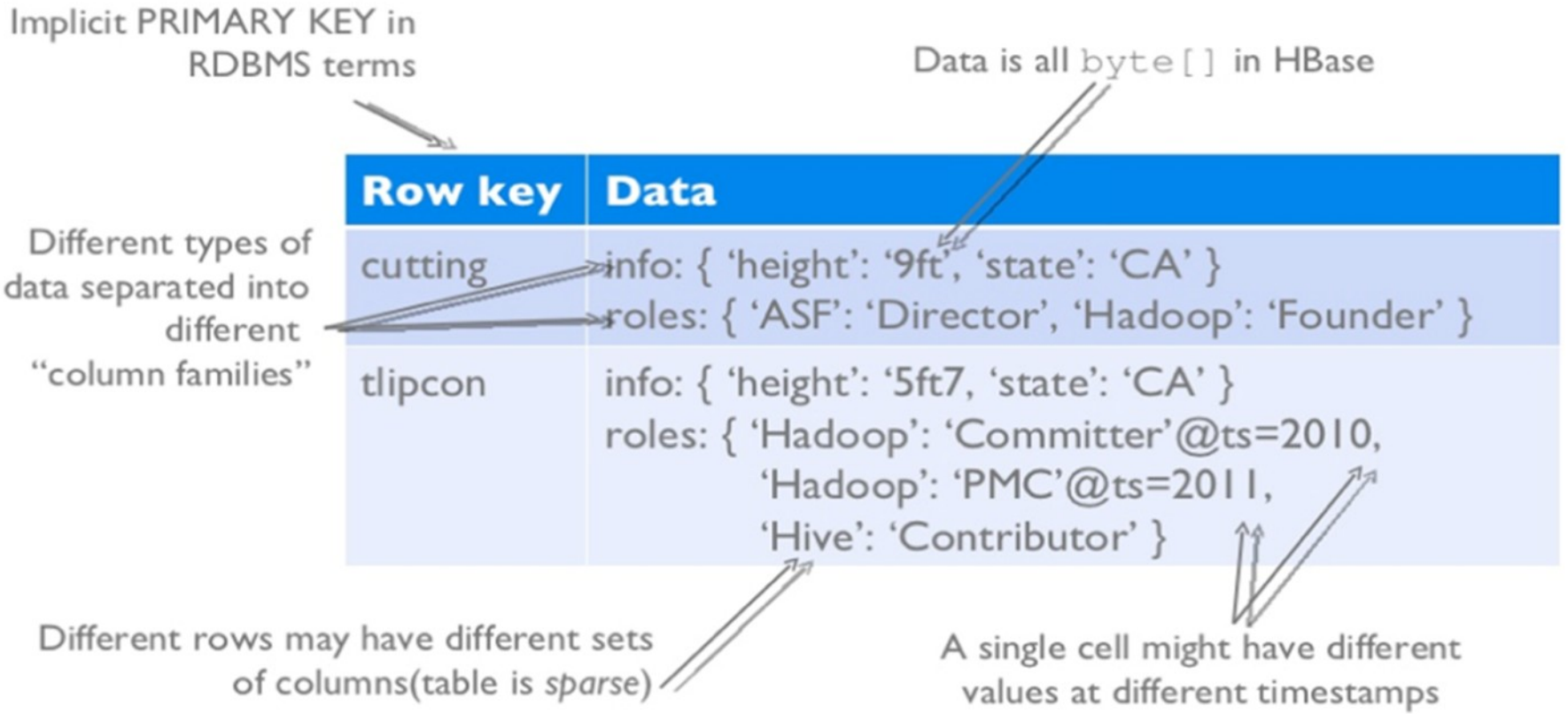
Big Table: Data Model

- Tables are sorted by Row
- Table schema only defines its *column families.*
  - Each family consists of any number of columns
  - Each column consists of any number of versions
  - Columns only exist when inserted, NULLs are free.
  - Columns within a family are sorted and stored together
- Everything except table names are bytearrays:

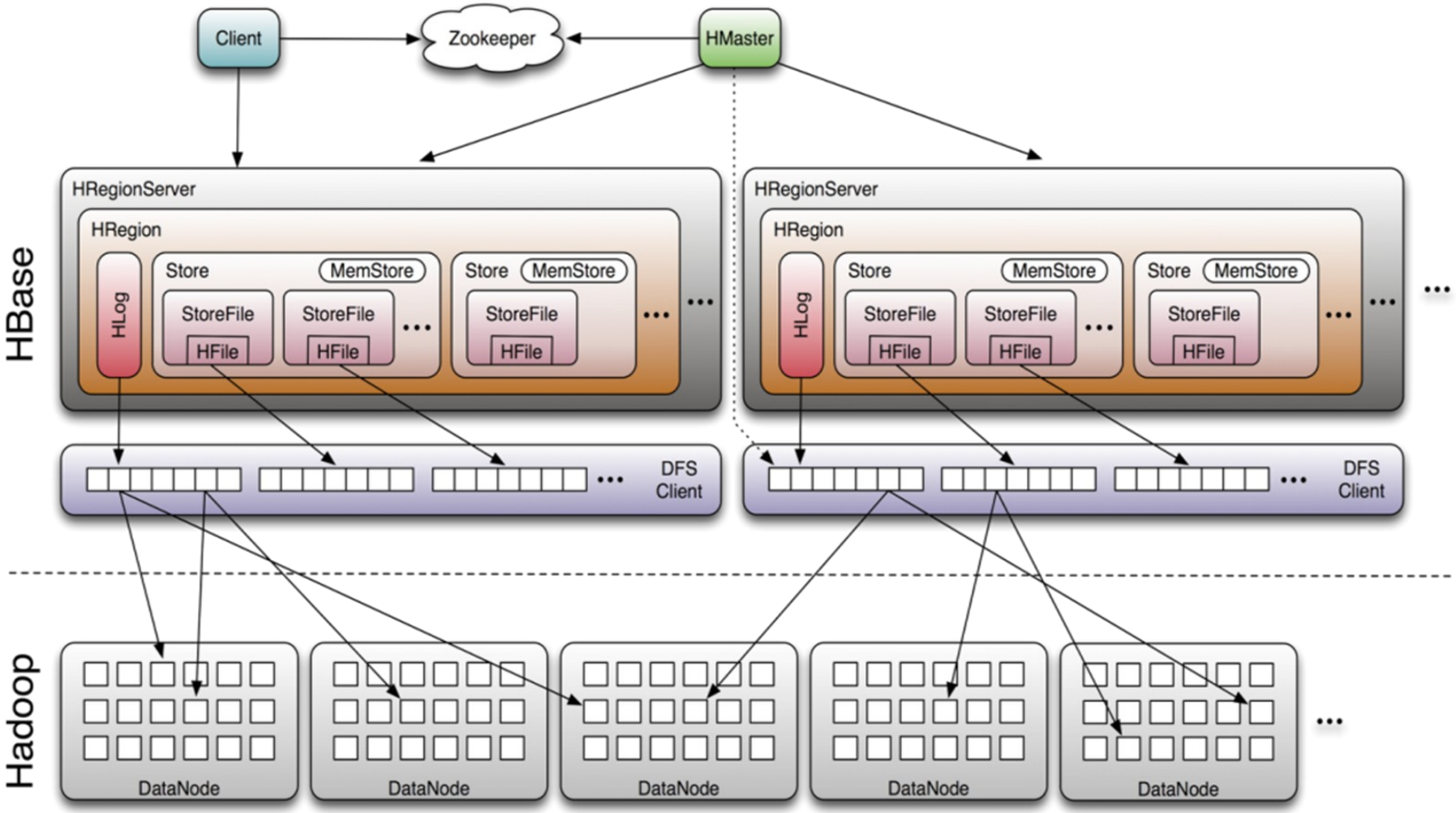
The tuple (Row, Family: Column, Timestamp) will correspond to Value



HBase Logical View

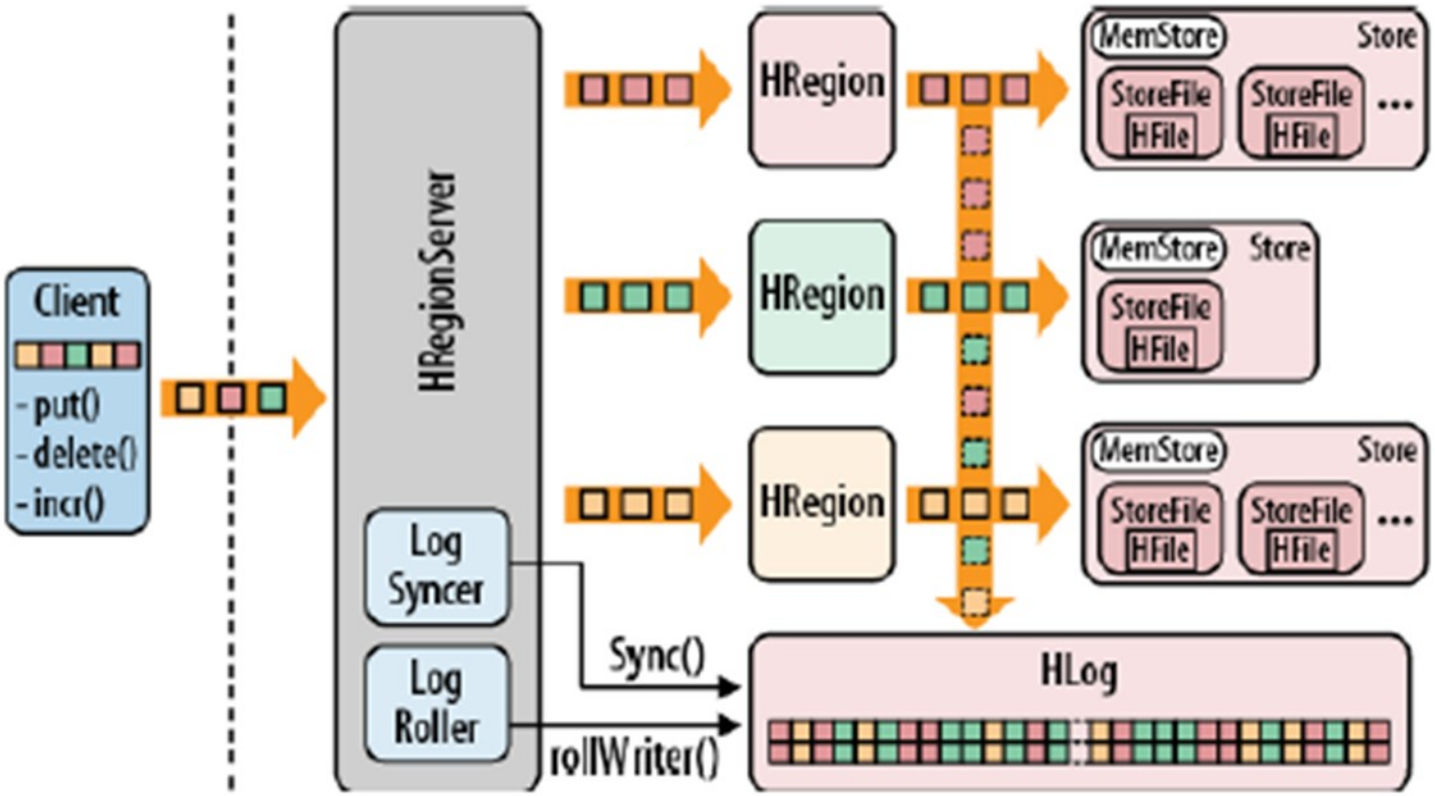


HBase implements BigTable on Hadoop



Hmaster	
HregionServer	A set of row keys are handled by one HRegionServer
HRegion	A set of column families are handled by HRegion
Store	Store handle columns and time stamps

Write path of HBase



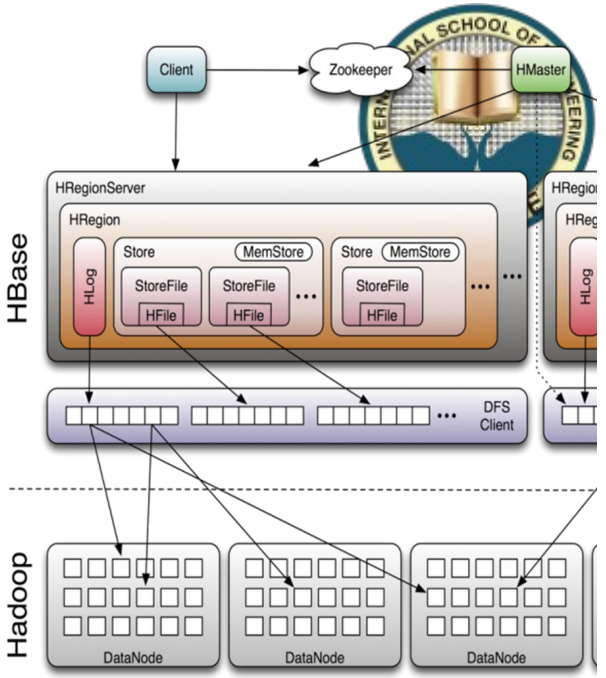
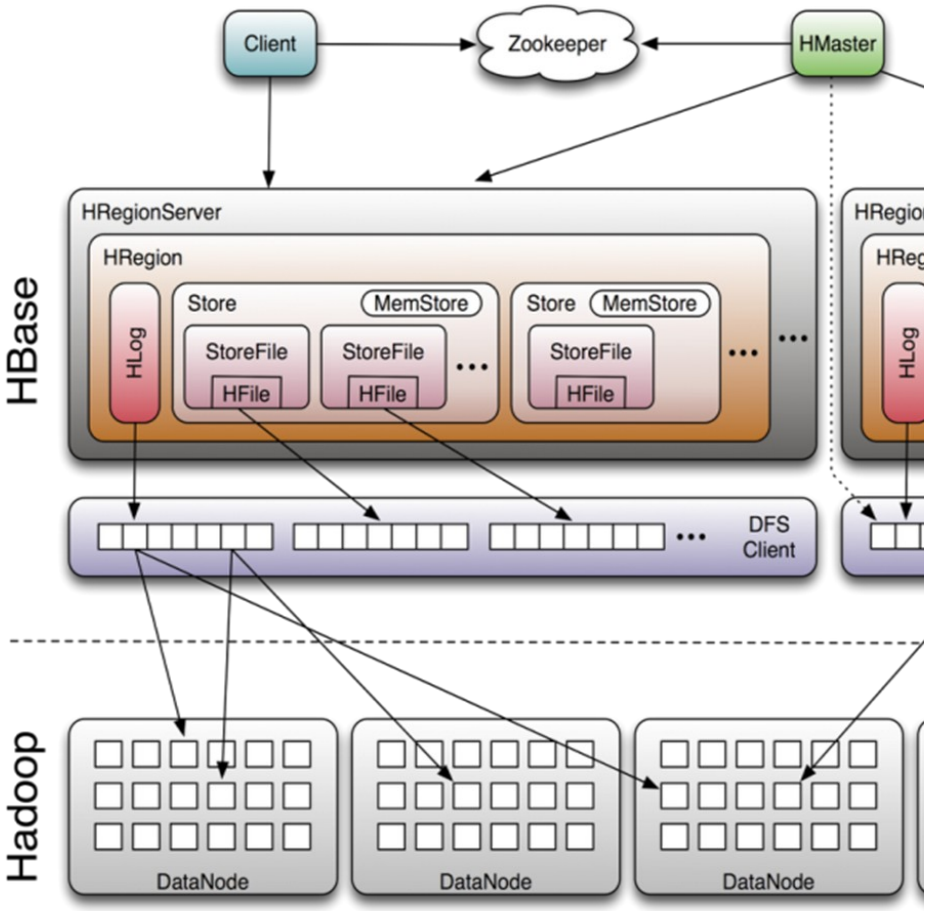


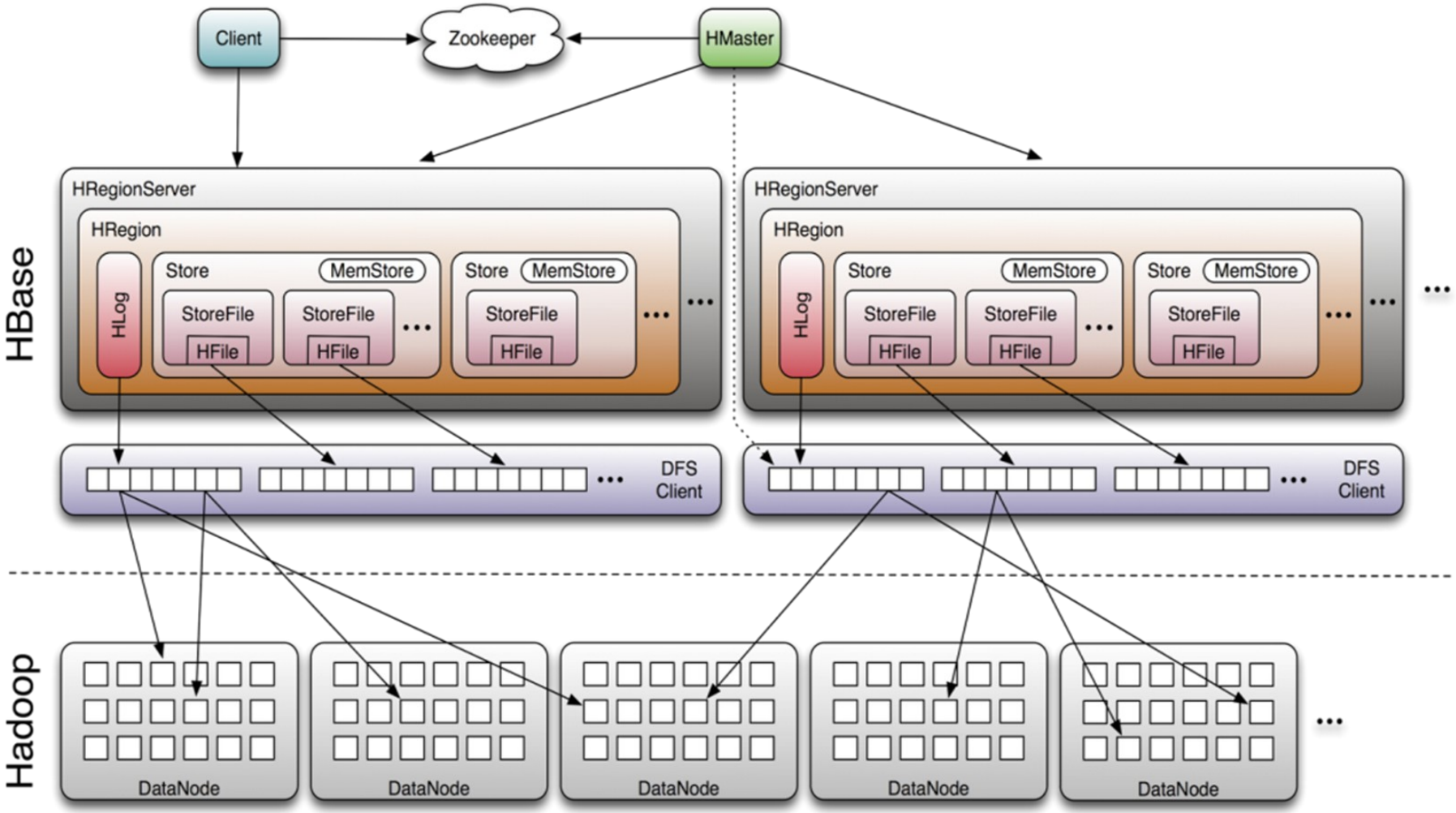
Figure: The write path of HBase

Hbase Response Time



- Let us say a value needs to be updated in HBase
- Before a direct access is made to Store and eventually to HDFS, HRegion will check Hlog
- If a row has been accessed recently to an entry then the new change is made using values from the HLog
  - As a result, StoreFile may have multiple (sometimes stale) versions of data
- But eventually the correct data will be written to HDFS
  - Leading to eventual consistency

HBase implements BigTable on Hadoop



"Facebook likes" run on Hbase: 1.5 billion users, 10 assets per day, 15 billion assets per day. 50 friends may like these contents, let us assume 3 likes per day per asset; This results in 45 billion counters updated per day and displayed
The feature was first implemented on MySQL, then Cassandra (Columnar, but not integrated with HDFS), then settled on HBase

http://highscalability.com/blog/2010/11/16/facebooks-new-real-time-messaging-system-hbase-to-store-135.html
https://www.facebook.com/notes/facebook-engineering/the-underlying-technology-of-messages/454991608919/

NoSQL: Performance Comparison

- Facebook Search

- MySQL > 50 GB Data
  - Writes Average : ~300 ms
  - Reads Average : ~350 ms

- Rewritten with Cassandra > 50 GB Data
  - Writes Average : 0.12 ms
  - Reads Average : 15 ms

HBase vs. HDFS

	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

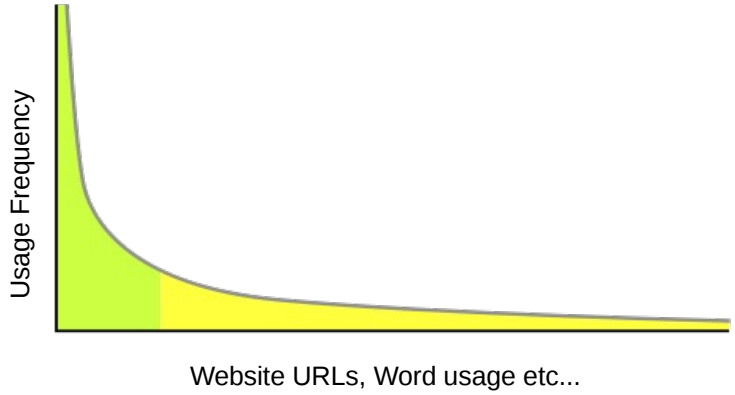
HBase vs. RDBMS

HBase already has SQL-like query language; security is just starting to emerge

	RDBMS	HBase
Data layout	Row-oriented	Column-family-oriented
Transactions	Multi-row ACID	Single row only
Query language	SQL	get/put/scan/etc *
Security	Authentication/Authorization	Work in progress
Indexes	On arbitrary columns	Row-key only
Max data size	TBs	~1PB
Read/write throughput limits	1000s queries/second	Millions of queries/second

When to use HBase

- Random read, write, or both r/w
  - Better to use some other tool when neither is needed
- Thousands of transactions per second on TBs of data
- Your access patterns are well known
  - Power law



https://en.wikipedia.org/wiki/Power_law

A good book on selected NoSQL DBs:
www.christof-strauch.de/nosqldbs.pdf

Agenda

- NoSQL:
  - Hbase
- Scripting in Big Data:
  - Hive / Pig
  - Other tools
    - Impala
    - Spark SQL
    - Apache Drill

```
class Mapper {
  buffer

  map(key, number) {
      buffer.append(number)
      if (buffer.is_full) {
          max = compute_max(buffer)
          emit(1, max)
      }
  }
}


class Reducer {
  reduce(key, list_of_local_max) {
      global_max = 0
      for local_max in list_of_local_max {
          if local_max > global_max {
              global_max = local_max
          }
      }         emit(1, global_max)
  }
}


class Combiner {
  combine(key, list_of_local_max) {
      local_max = maximum(list_of_local_max)
      emit(1, local_max)
  }
}
```

What is this code doing?

Can we do it more easily?

Why cant you have a combiner that sends the list in decreasing order?

Reducer can than pick the largest item coming from combiner

Alternatively, system can provide a function: **max()**, and this generates the underlying map-reduce code

Hive and Pig

- Provide higher-level language to facilitate large-data processing
- Higher-level language “compiles down” to Hadoop jobs
- Hive can be slow, so why do it?
  - Better alternatives are available, but overall concept is used elsewhere
    - Hive on Spark
    - Impala (After releasing Hive, Facebook released Impala, essentially a better Hive)
- Can you write better map reduce scripts than Hive? Ans: Maybe
  - Pig? Ans: If you are very good :)

https://cwiki.apache.org/confluence/display/Hive/Hive+on+Spark

Hive

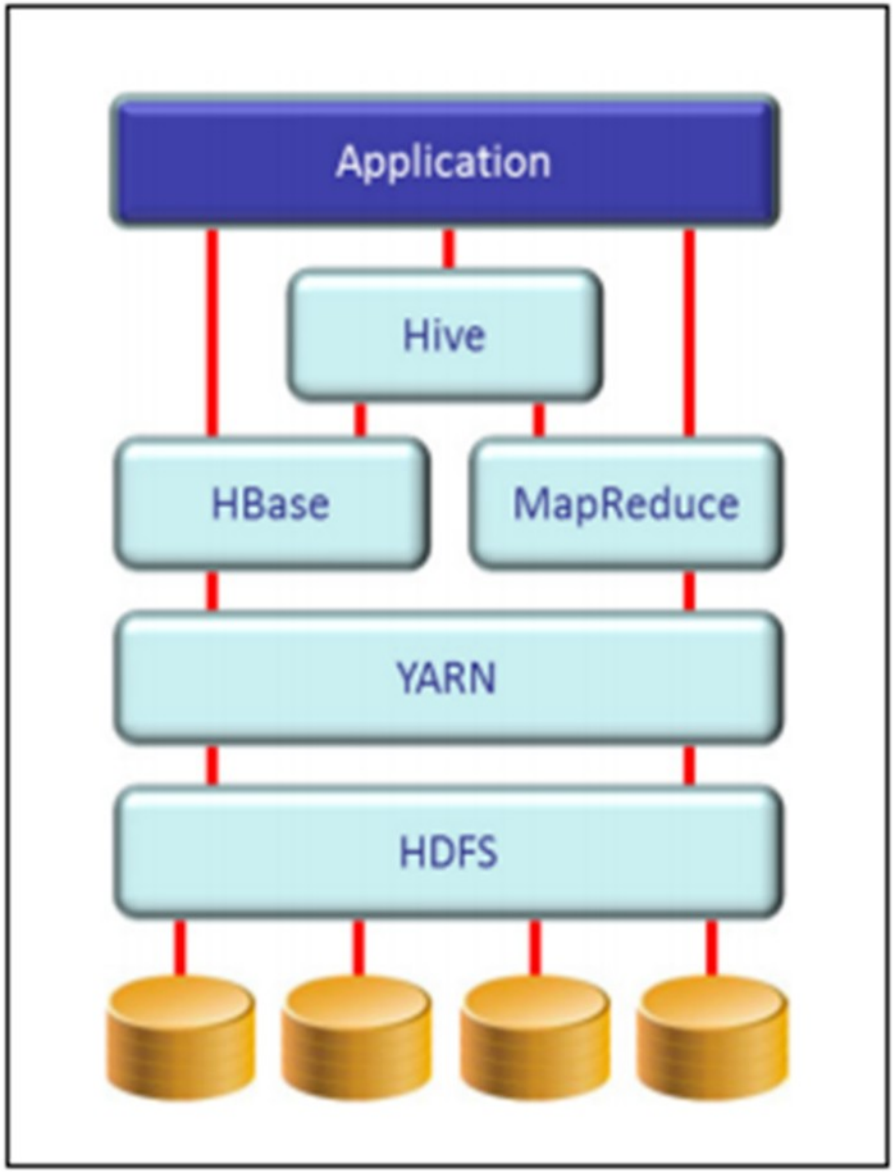
- Hive: data warehousing application in Hadoop
  - Query language is HQL, variant of SQL
  - Tables stored as HDFS flat files

- Developed by Facebook (2007/2008), now open source

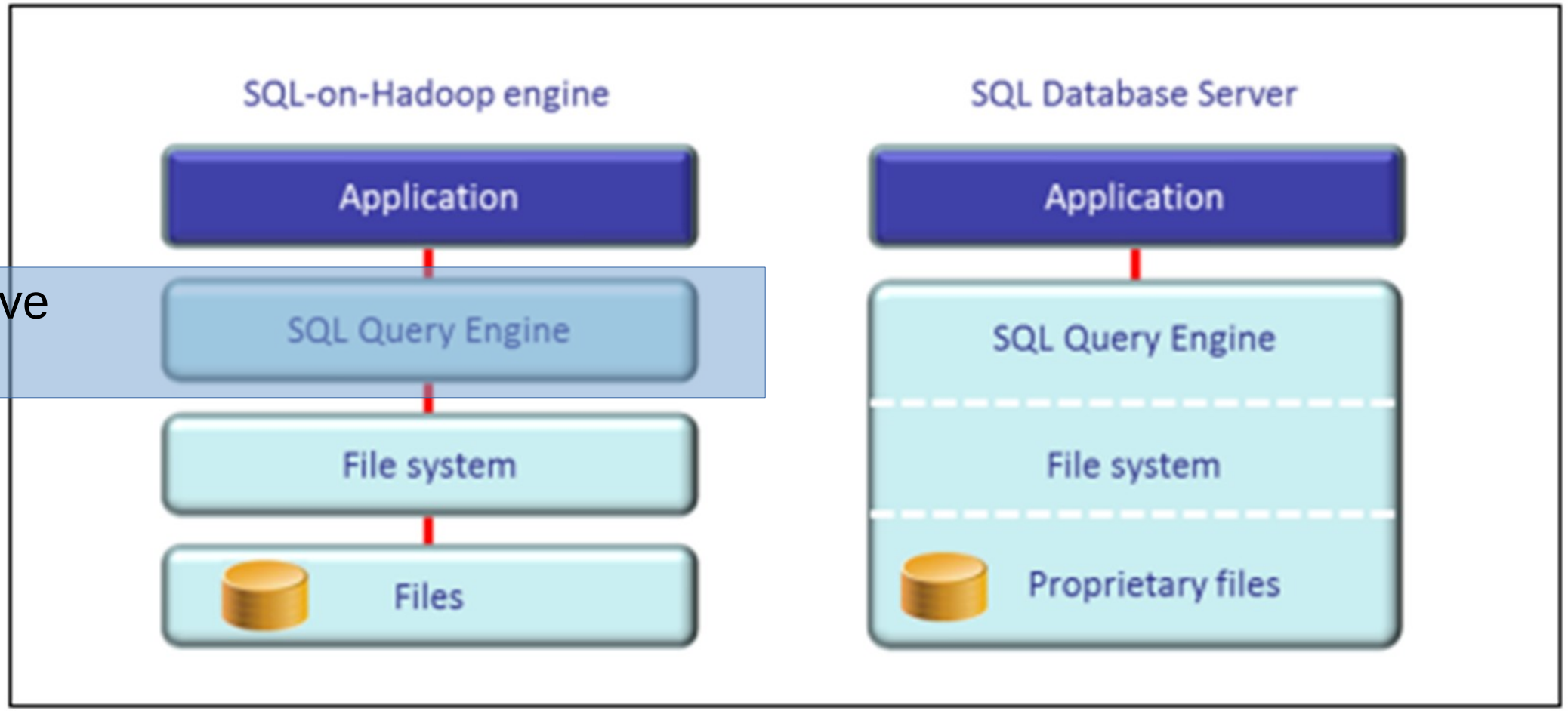
```
-- Delete the tables if they already exist
DROP TABLE myinput;
DROP TABLE wordcount;
CREATE TABLE myinput (line STRING);

-- Load the text from the local (Linux) filesystem.
-- This should be changed to HDFS for any serious usage
LOAD DATA LOCAL INPATH '/user/someperson/mytext.txt' INTO
TABLE myinput;

-- Create a table with the words grouped and counted.
CREATE TABLE wordcount AS
SELECT word, count(1) AS count
GROUP BY word

-- Sort the output by count with the highest counts first
ORDER BY count DESC, word ASC;
```

Above is a sample of Hive code (or HiveQL): Hive will compile this into Map-Reduce code and execute the relevant MR jobs

Hive Architecture

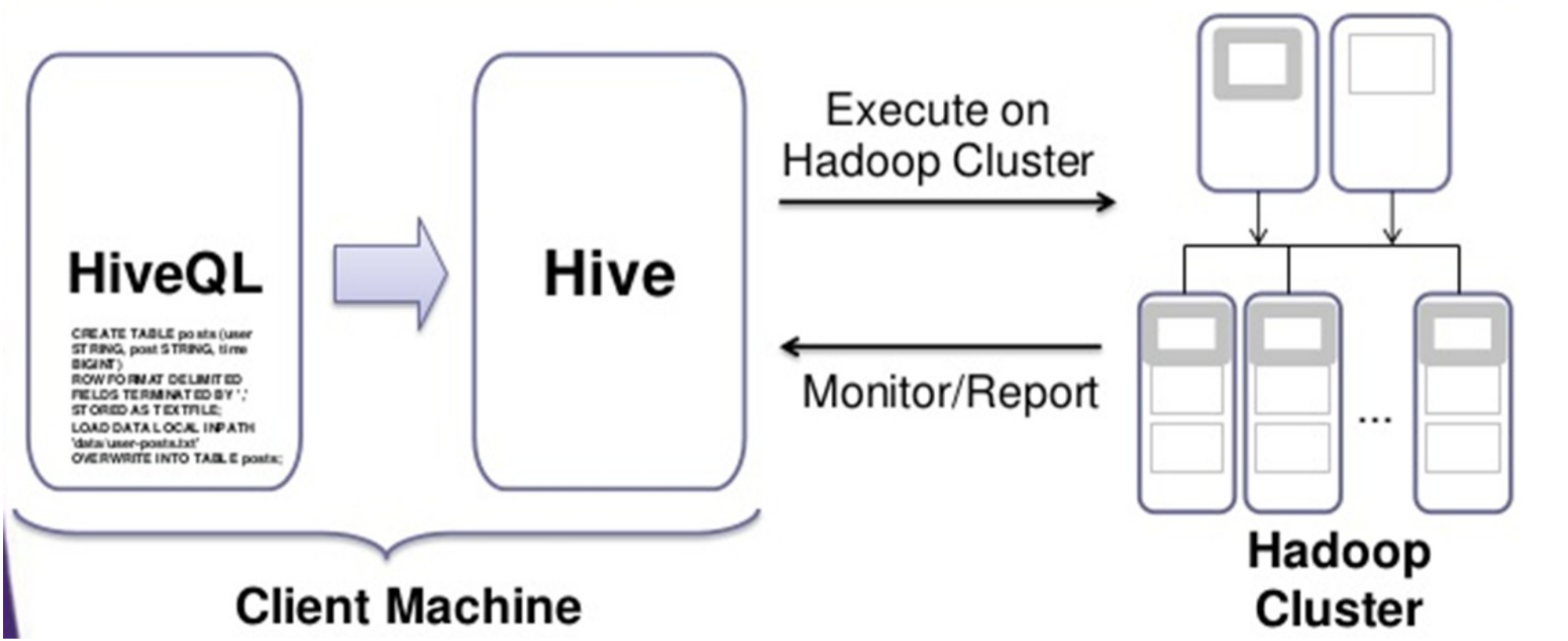


Hive vs traditional SQL Architecture



Hive

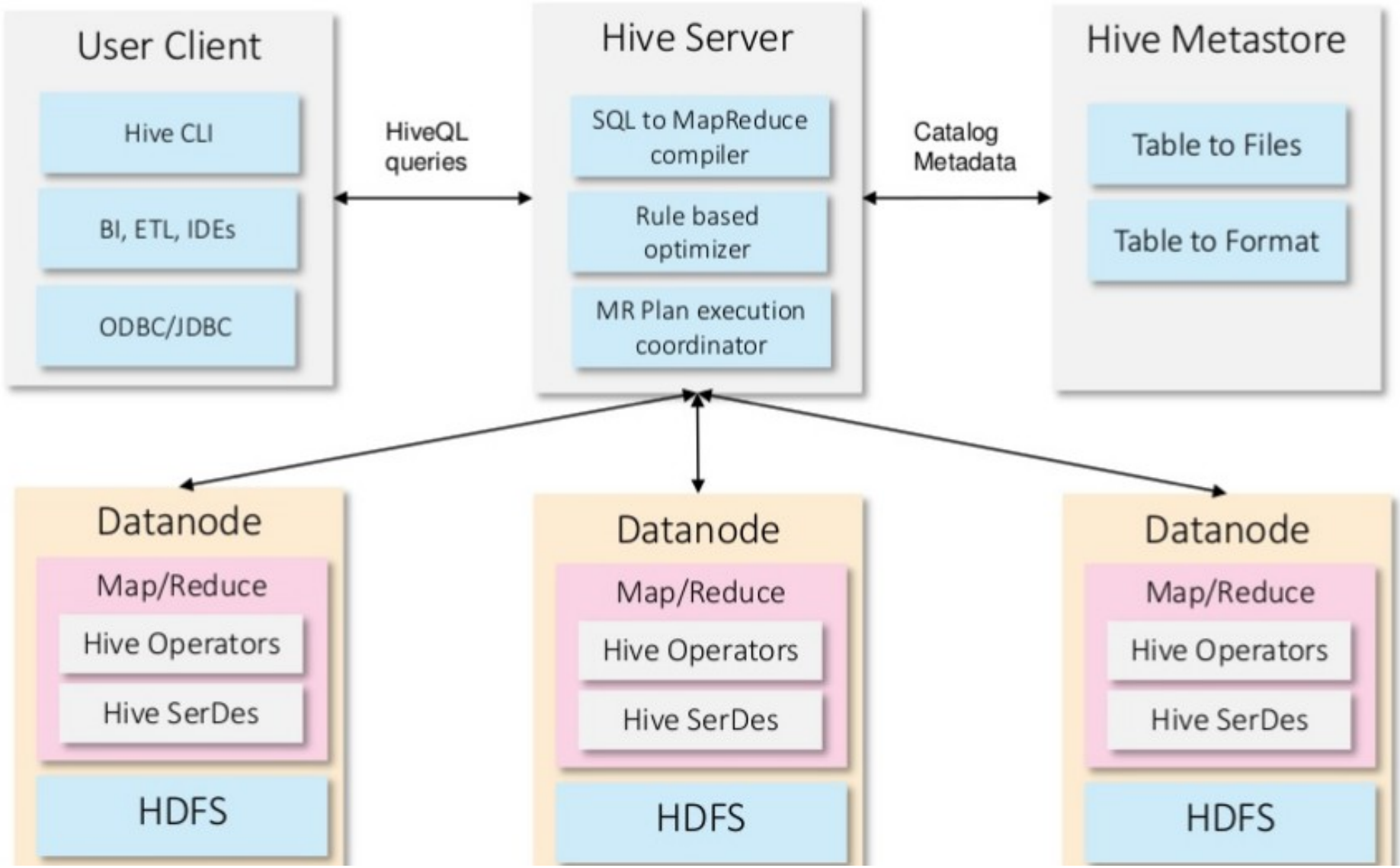
Hive Execution Model: Translate to Map-Reduce



Hive was originally designed to sit on a client machine

List of Hive Commands and built-in UDFs:
https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF

Hive Architecture



PIG

Procedural Queries on Big Data

- Google Sawzall (2003)
  - Rob Pike et al. Interpreting the Data: Parallel Analysis with Sawzall. Special Issue on Grids and Worldwide Computing Programming Models and Infrastructure 2003.

- Apache Pig (2006)
  - Christopher Olston et al. Pig Latin: A Not-So-Foreign Language for Data Processing. SIGMOD 2008.
  - https://pig.apache.org/

Example Data Analysis Task

Find the top 10 most visited pages in each category

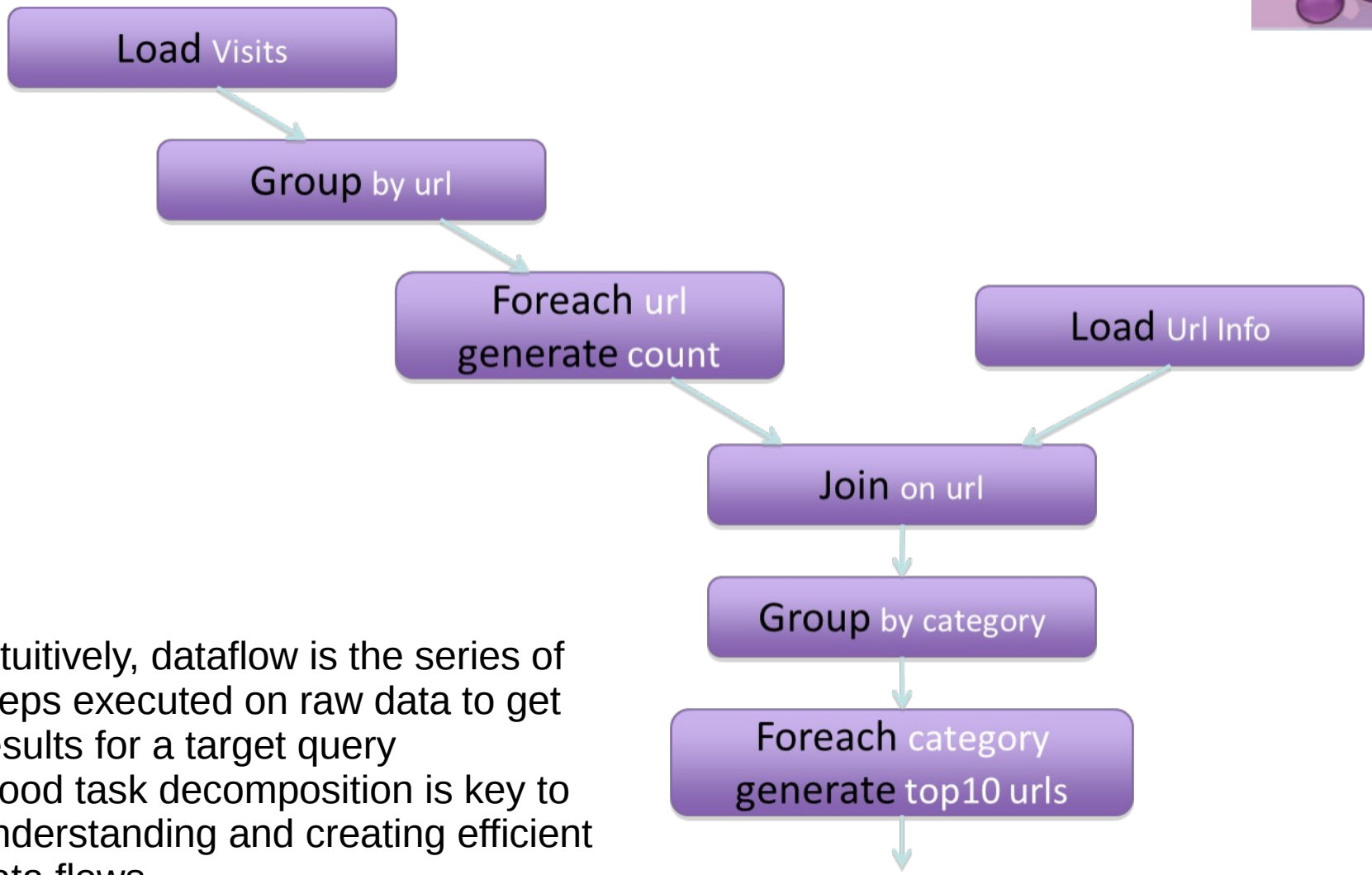
Visits

User	Url	Time
Amy	cnn.com	8:00
Amy	bbc.com	10:00
Amy	flickr.com	10:05
Fred	cnn.com	12:00

URLs

Url	Category	PageRank
cnn.com	News	0.9
bbc.com	News	0.8
flickr.com	Photos	0.7
espn.com	Sports	0.9

Data Flow



Intuitively, dataflow is the series of steps executed on raw data to get results for a target query
Good task decomposition is key to understanding and creating efficient data flows

Quick Start and Interoperability

```
visits       = load '/data/visits' as (user, url, time);
gVisits      = group visits by url;
visitCounts  = foreach gVisits generate url, count(visits);

urlInfo      = load '/data/urlInfo' as (url, category, pRank);
visitCounts  = join visitCounts by url, urlInfo by url;

gCategories = group visitCounts by category;
topUrls = foreach gCategories generate top(visitCounts,10);

store topUrls into '/data/topUrls';
```

Everything in orange is a PIG command, for instance "top" is a User-Defined Function (UDF)

How does PIG work?

- Submit a script directly
- Grunt, the pig shell
- PigServer class, a JDBC like interface
- PigPen, an eclipse plugin
  - Allows text and UI based scripting
  - Samples data and shows example data flows

- Start a terminal and run

```
$ cd /usr/share/cloudera/pig/
$ bin/pig -x local
```

- Should see a prompt like:

```
grunt>
```

How Does Pig Work

Pig Latin

```
A = LOAD 'myfile'
    AS (x, y, z);
B = FILTER A by x > 0;
C = GROUP B BY x;
D = FOREACH A GENERATE
    x, COUNT(B);
STORE D INTO 'output';
```

pig.jar:

- parses
- checks
- optimizes
- plans execution
- submits jar to Hadoop
- monitors job progress

Execution Plan

Map:
Filter

Reduce:
Count

UI to enter PIG Latin and the PIG compiler runs on client

Execution happens on cluster

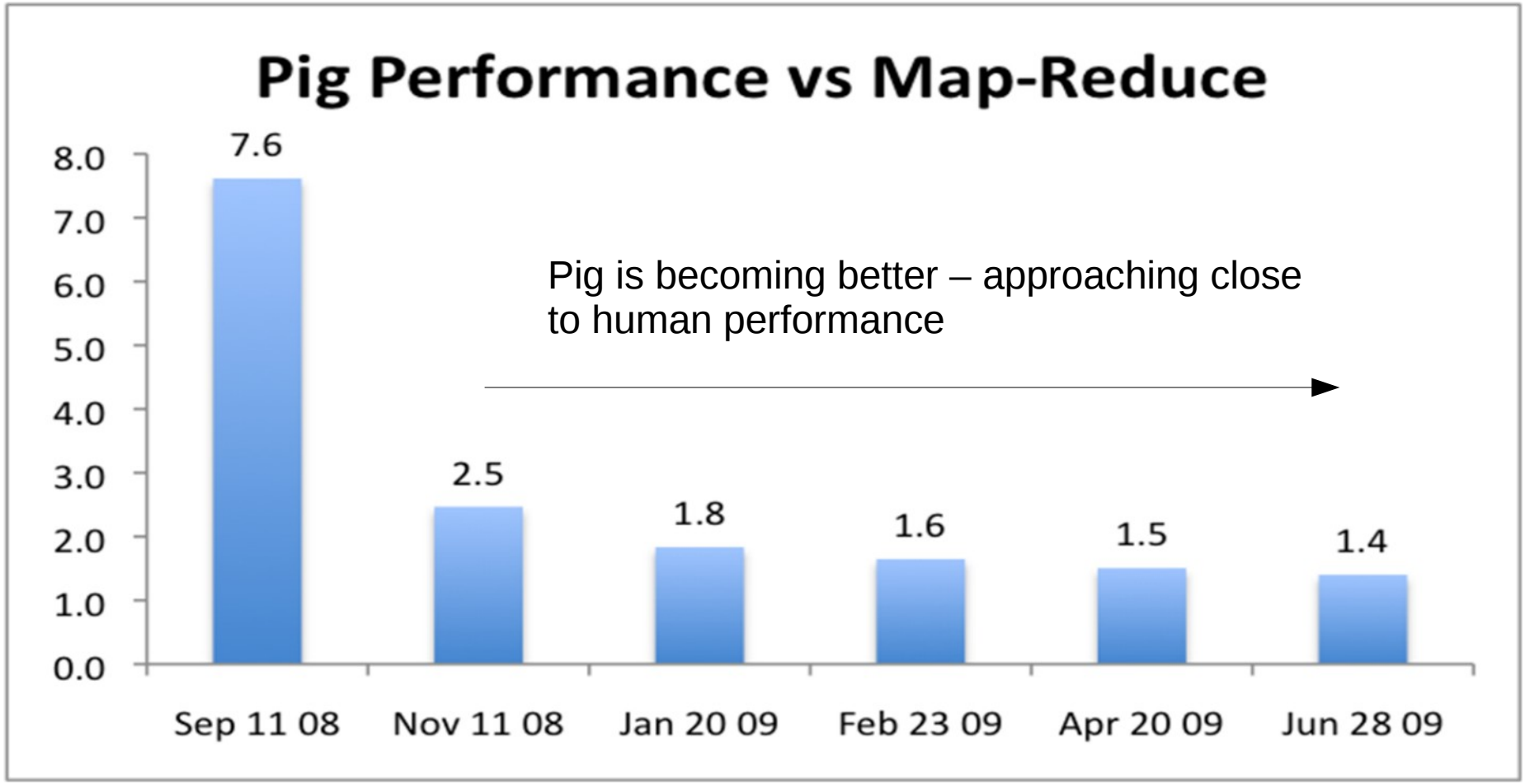
Pig vs MapReduce

Pig Latin

```
A = LOAD 'myfile'
    AS (x, y, z);
B = FILTER A by x > 0;
C = GROUP B BY x;
D = FOREACH A GENERATE
    x, COUNT(B);
STORE D INTO 'output';
```

The MapReduce code is 240 lines compared to five lines of Pig code

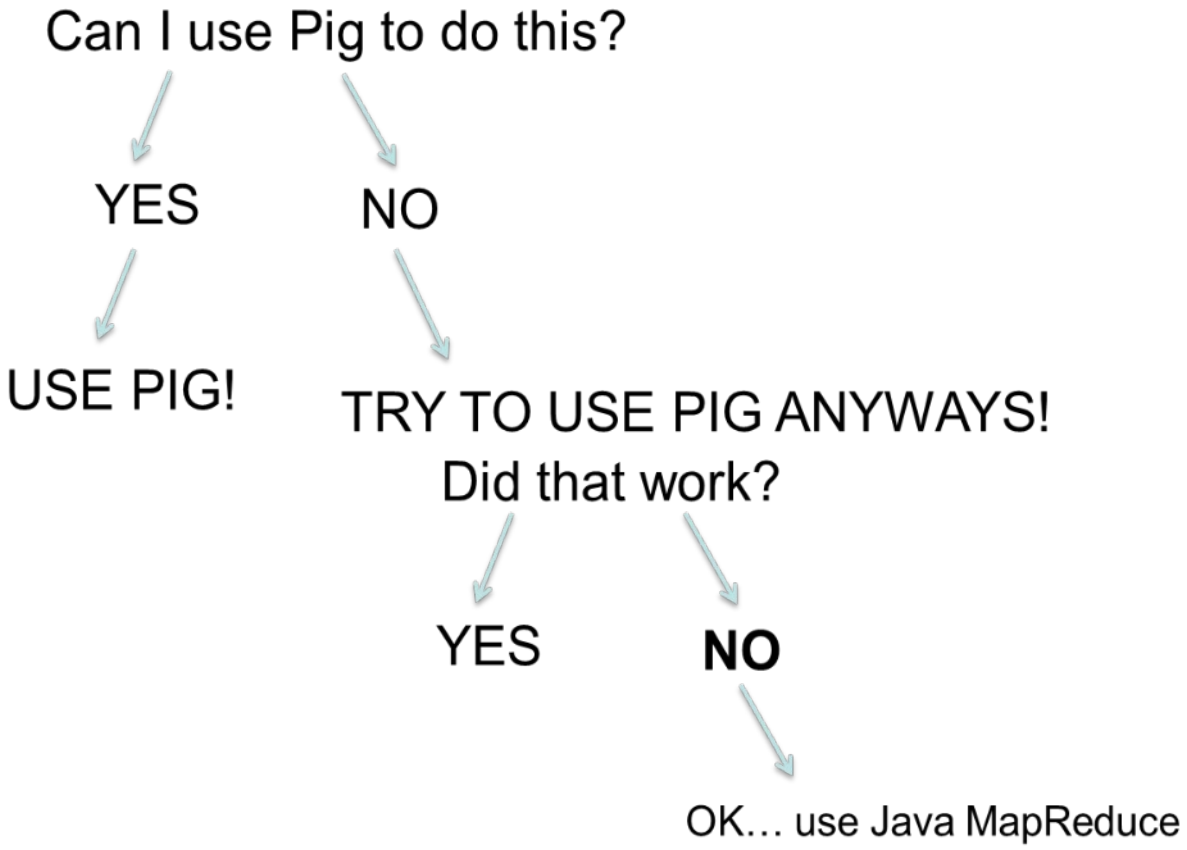
Pig vs MapReduce



A set of queries used to test pig performance: https://cwiki.apache.org/confluence/display/PIG/PigMix

When to use Pig and MapReduce?

Donald Miner (August 2013)



Key Points in Pig

- Nothing happens till a Dump or Store is performed
- Use filter and foreach early to remove unnecessary columns or rows and reduce intermediate temporary results
- Use Parallel keyword on group to run more reduce tasks

http://pig.apache.org/docs/r0.7.0/cookbook.html#Use+the+PARALLEL+Clause

VS

Pig vs Hive

- Pig directly operates on flat files and does not need a table format (Hive assumed tabular data structure)
  - Files are assumed to have delimiters; you can say “delimited by”
- Schema is optional: We have not told if user is string or int or etc
  - Lazy type evaluation; till I ask Pig to perform an evaluation like abc/2, variable abc will not be converted to int.
    - After using a variable as string, if you use the same var as int, pig will complain
- Pig is a compiler, it does not interpret each line and waits till it looks at a “store” command
  - Code is not converted till Pig can see a bunch of code between a load and store construct
  - Can lead to more efficient map-reduce (In general, compilers are known to generate more efficient code in comparison with interpreters)

Pig procedural v/s SQL declarative

Pig

- Users = load 'users' as (name, age, ipaddr);
- Clicks = load 'clicks' as (user, url, value);
- ValuableClicks = filter Clicks by value > 0;
- UserClicks = join Users by name, ValuableClicks by user;
- Geoinfo = load 'geoinfo' as (ipaddr, dma);
- UserGeo = join UserClicks by ipaddr, Geoinfo by ipaddr; ByDMA = group UserGeo by dma;
- ValuableClicksPerDMA = foreach ByDMA generate group, COUNT(UserGeo);
- store ValuableClicksPerDMA into 'ValuableClicksPerDMA';

Pig commands appear as DataFlow

SQL

```
insert into
    ValuableClicksPerDMA
    select dma, count(*) from
    geoinfo join ( select
    name, ipaddr from users
    join clicks on (users.name
    = clicks.user) where
    value > 0; ) using ipaddr
    group by dma;
```

SQL statements are designed in a declarative fashion where multiple queries are overlaid within each other

Pig procedural v/s SQL declarative

Pig

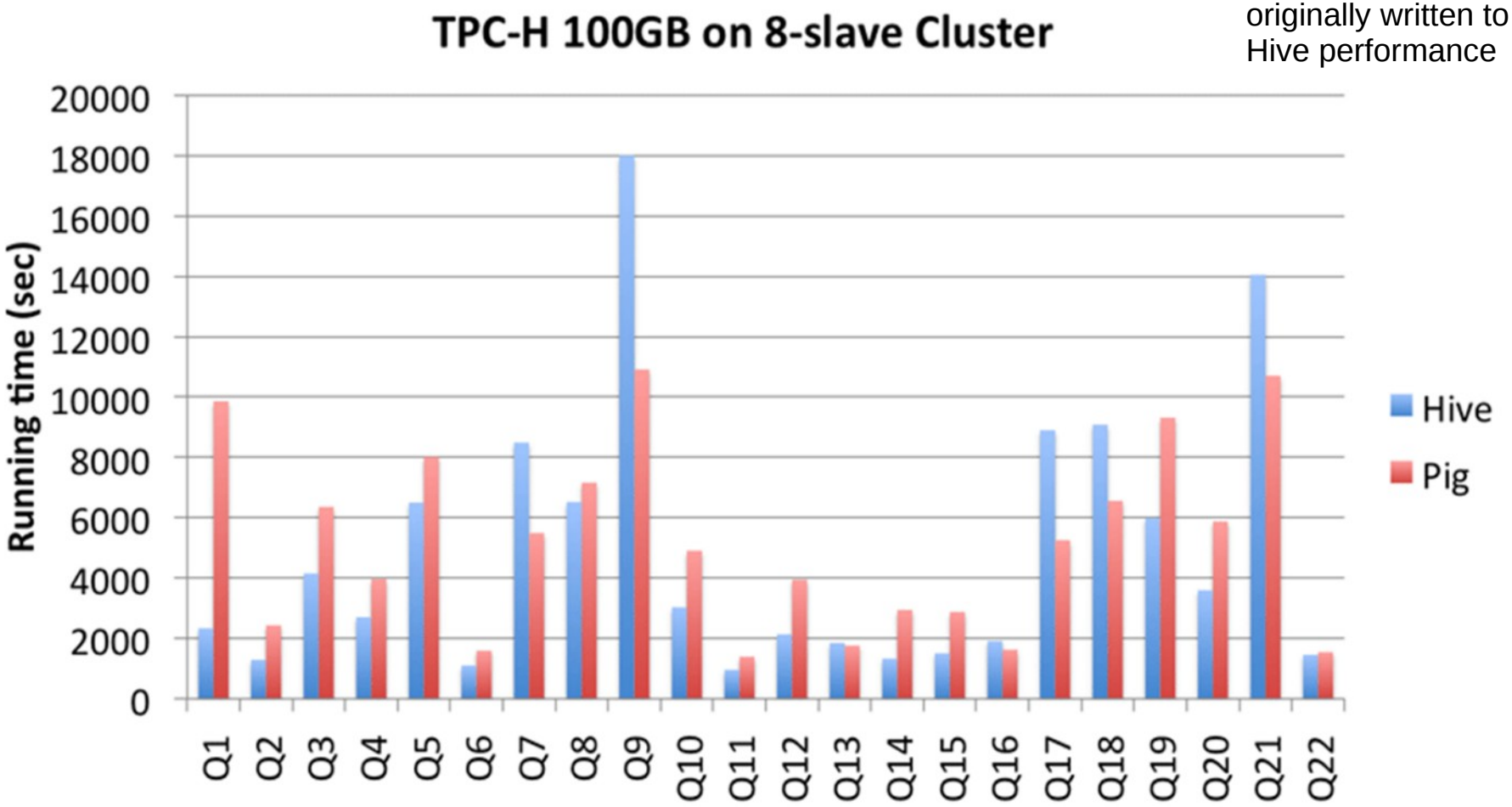
- Pig is procedural
- Nested relational data model (No constraints on Data Types)
- Schema is optional
- Scan-centric analytic workloads (No Random reads or writes; do something with the entire flat file)
  - Better for analytical queries, not transactional queries
- Limited query optimization
  - Code optimization is better

SQL / HQL

- SQL is declarative
- Flat relational data model (Data is tied to a specific Data Type)
- Schema is required
- OLTP + OLAP workloads (Perform tasks on specific transactions or rows)
- Significant opportunity for query optimization

Pig vs Hive performance comparison

TPC-H is a set of queries originally written to check Hive performance



http://hortonworks.com/blog/pig-performance-and-optimization-analysis/ (Sep 2012)
http://www.ibm.com/developerworks/library/ba-pigvhive/pighivebenchmarking.pdf

Pig vs Hive

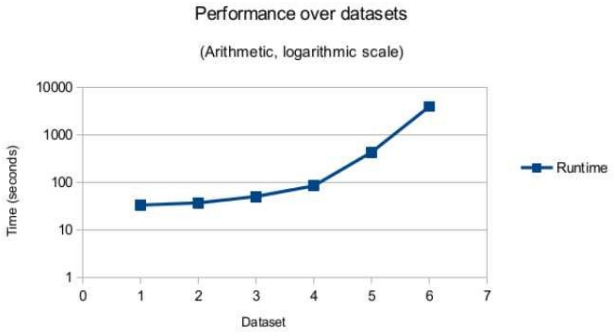


Figure 1: Pig runtime plotted in logarithmic scale

Runtime efficiency will depend on the data

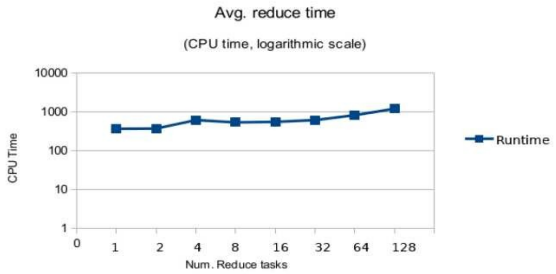


Figure 2: **Hive** reduce time (CPU time) plotted in logarithmic scale.

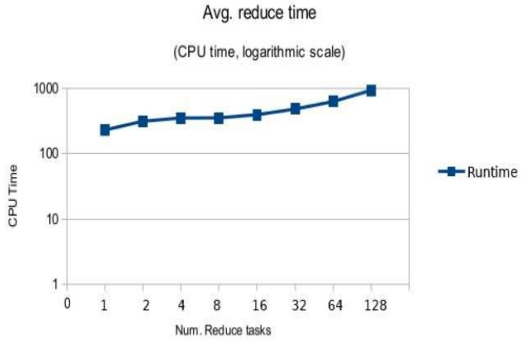
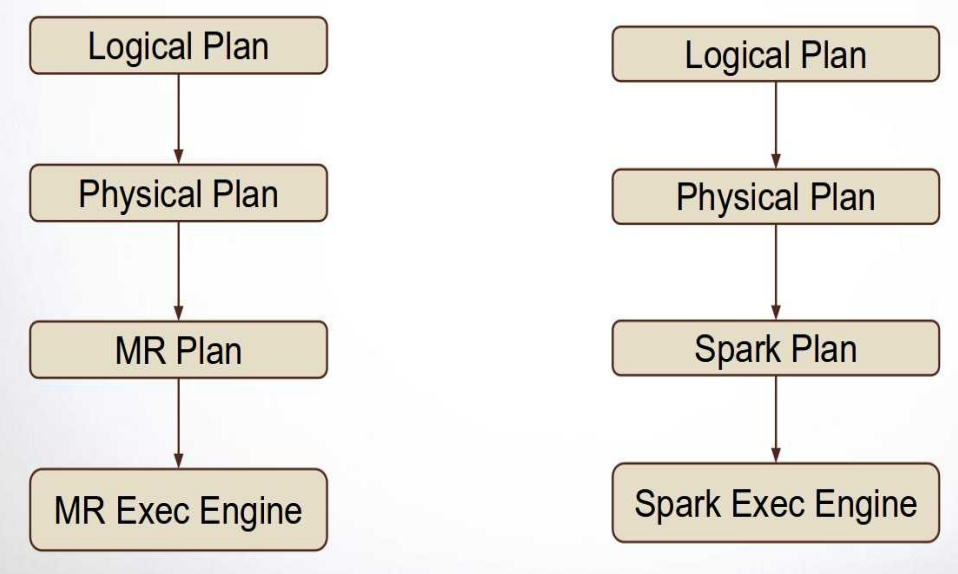


Figure 3: **Pig** reduce time (CPU time) plotted in logarithmic scale.

Increasing number of reducers does not necessarily speed up processing, there will be an overhead of moving data around

Images taken from: http://www.ibm.com/developerworks/library/ba-pigvhive/pighivebenchmarking.pdf

Pig on Spark



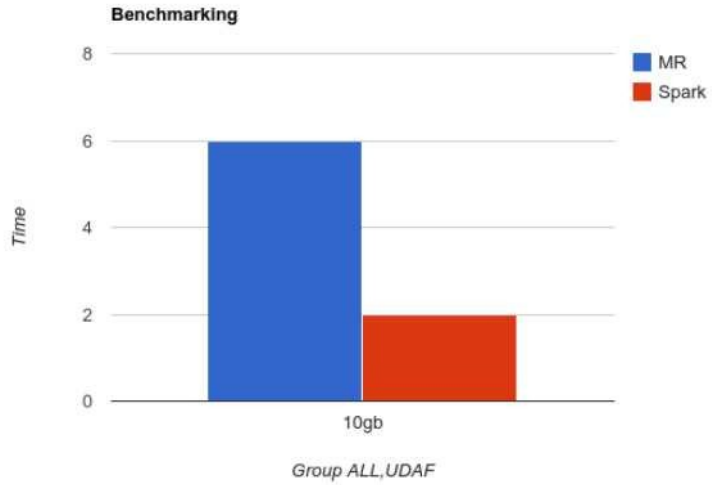
The Pig queries remain same but at the pysical plan stage are converted into a spark plan

Pig Operator	Spark Operator
Load	newAPIHadoopFile
Store	saveAsNewAPIHadoopFile
Filter	filter transformation
GroupBy	groupby & map
Join	CoGroup
ForEach	mapPartitions
Sort	sortByKey + map

Pig Operators have equivalent spark commands

Source: http://events.linuxfoundation.org/sites/events/files/slides/ApacheCon-Pig-on-Spark.pdf

Pig on Spark



Approximately 3X speedup is obtained; this is currently not in-memory spark

Even with the speedup of Pig, or Pig on spark, these systems are not designed for OLAP (**Online** Analytical Processing) or OLTP (**Online** Transactional Processing)

The online part is key – RDBS are tuned to perform better on transactional data

Underlying structures are MR, Hive, HDFS; these are not good for transactional loads that need ACID

Specific transactions that do not need ACID are OK on these systems

Source: http://events.linuxfoundation.org/sites/events/files/slides/ApacheCon-Pig-on-Spark.pdf

SQL is Cool!

Started here

Progression to here

Batch jobs via Map Reduce

Apache Hive

- ✓ Fault Tolerance
- ✓ Scales to Petabytes
- ✓ Schema Flexibility

Transactional Database on Hbase?

Unproven

Real-time query response
On Data Warehouse

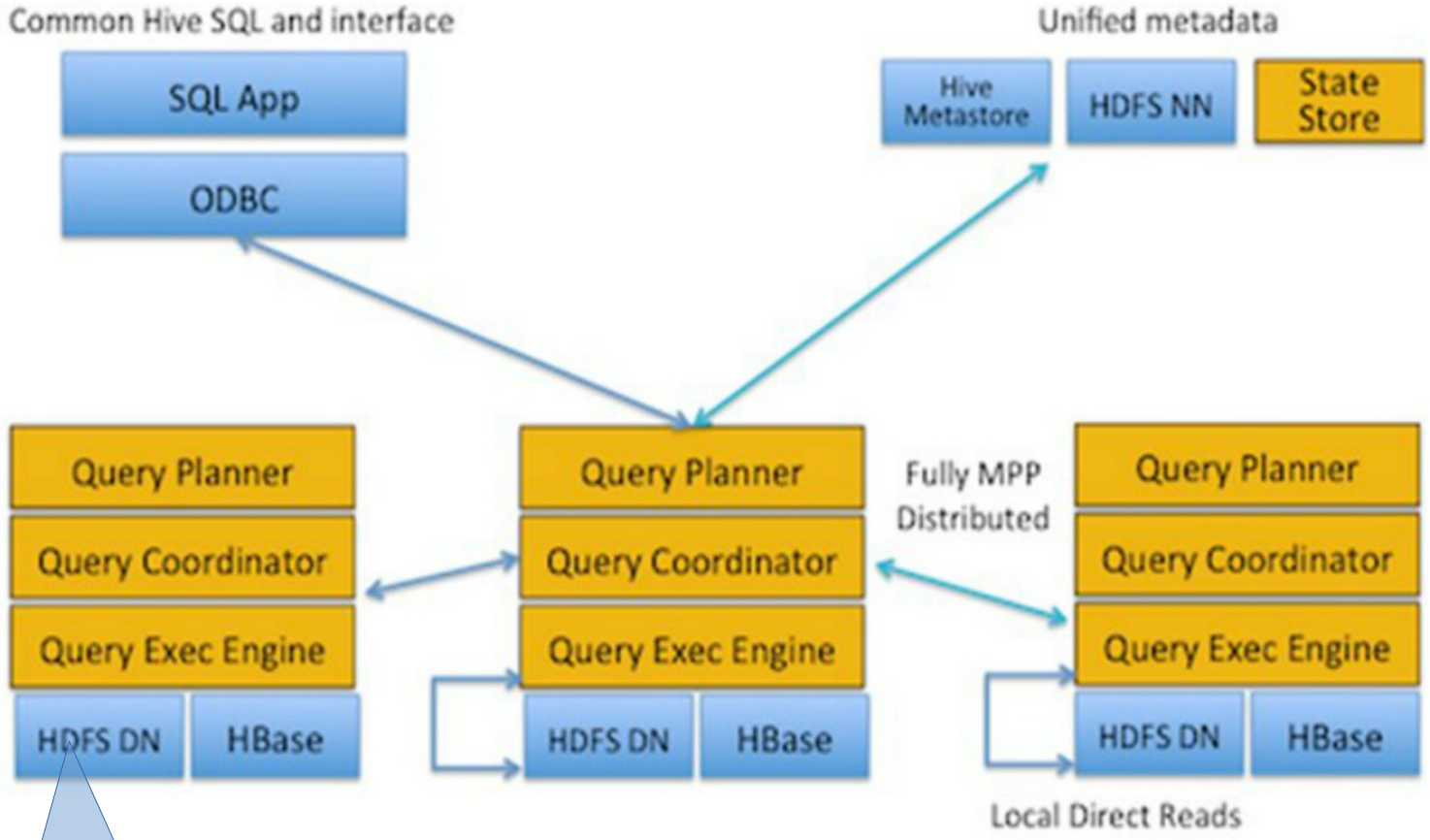
- Cloudera Impala
- Apache
  - Drill (MapR)
  - Presto (Facebook)
  - Shark/Spark (UC Berkeley AMPLab)
  - Stinger initiative and Tez (Hortonworks)
- IBM Big SQL
- Pivotal HAWQ
- NEWSQL

? Fault Tolerance
? Scale to Petabytes
? Schema Flexibility

Agenda

- NoSQL:
  - Hbase
- Scripting in Big Data:
  - Hive / Pig
  - Other tools
    - Impala
    - Spark SQL
    - Apache Drill

Cloudera Impala



These run on the DataNodes and Containers

Stinger is a similar tool to Impala; difference and IP is in the query planning and coordination algorithms. IMPALA does not use MR, but uses YARN and HDFS
http://tinyurl.com/hfd5clx

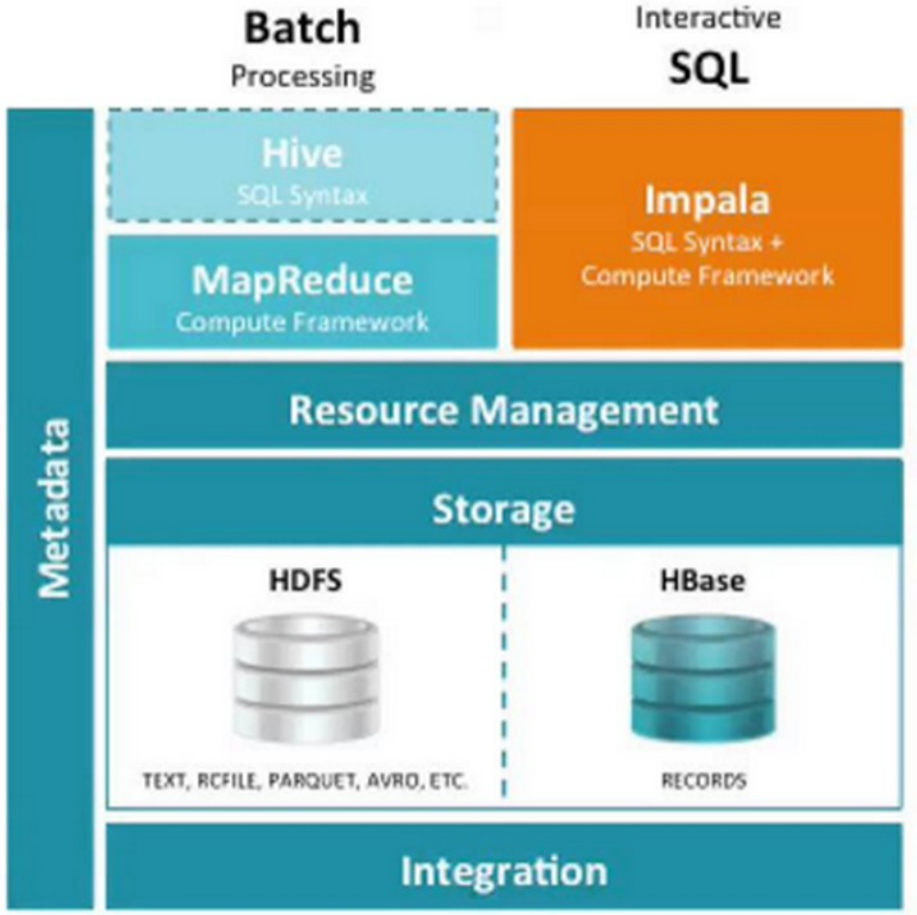
Cloudera Impala vs Hive

Shares Everything Client-Facing

- Metadata (table definitions)
- ODBC/JDBC drivers
- SQL syntax (Hive SQL)
- Flexible file formats
- Machine pool
- Hue GUI

But Built for Different Purposes

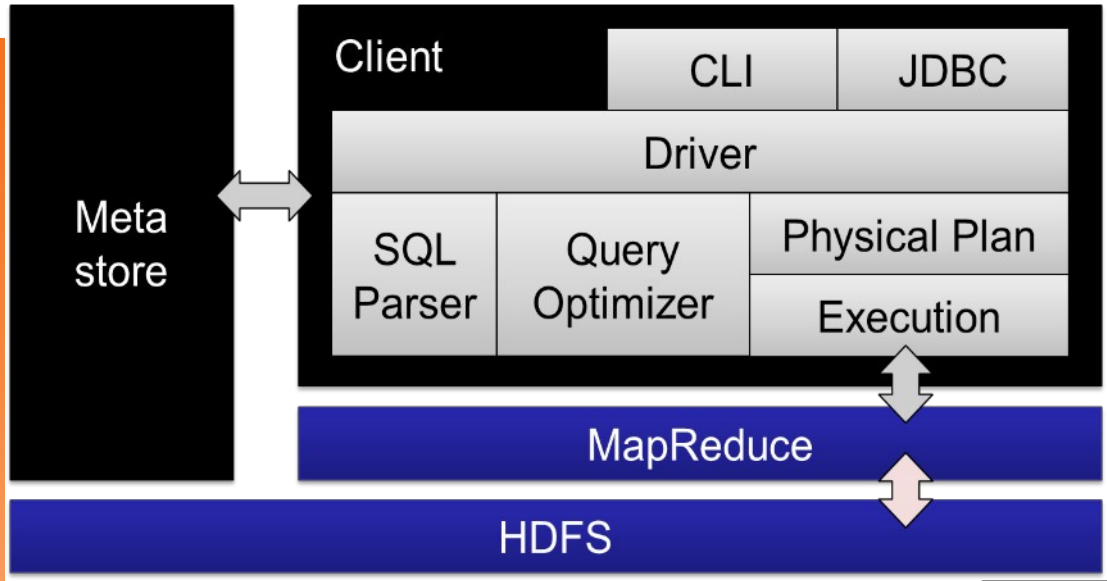
- **Hive:** runs on MapReduce and ideal for batch processing
- **Impala:** native MPP query engine ideal for interactive SQL



Agenda

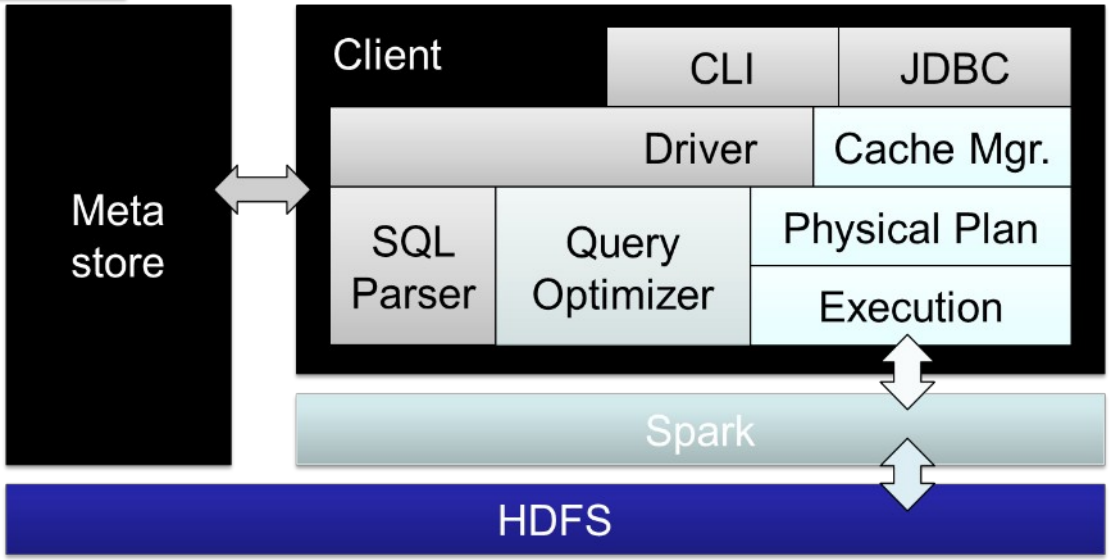
- NoSQL:
  - Hbase
- Scripting in Big Data:
  - Hive / Pig
  - Other tools
    - Impala
    - Spark SQL
    - Apache Drill

Hive vs Spark SQL (Shark) Architecture



Hive

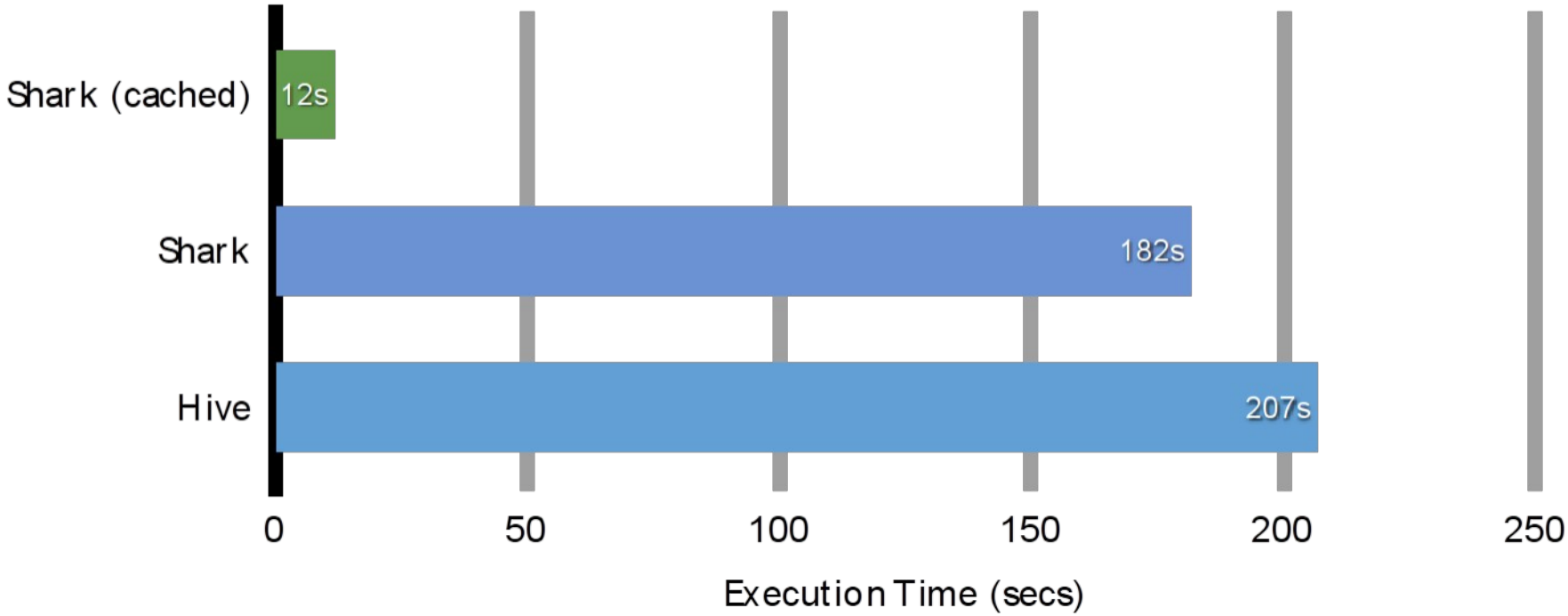
Spark SQL



[Engle et al, SIGMOD 2012]

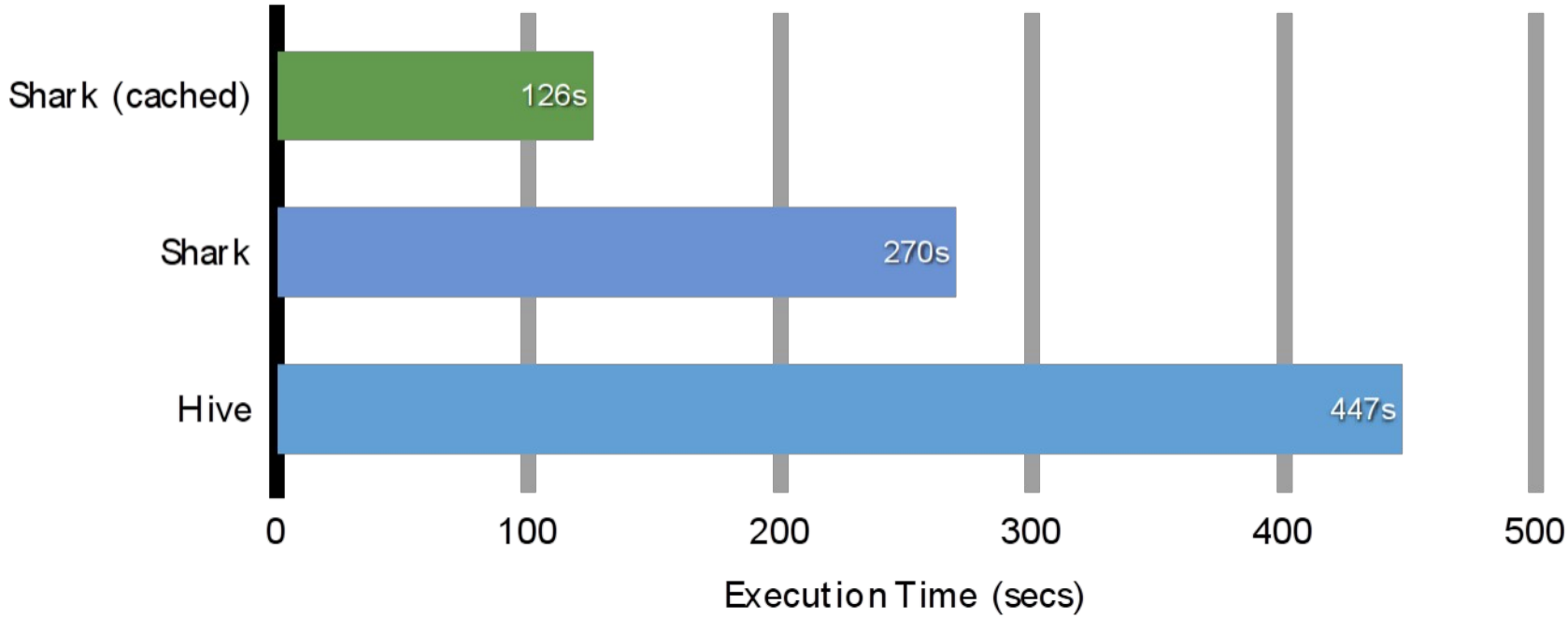
Benchmark Query 1

```
SELECT * FROM grep WHERE field LIKE '%XYZ%';
```

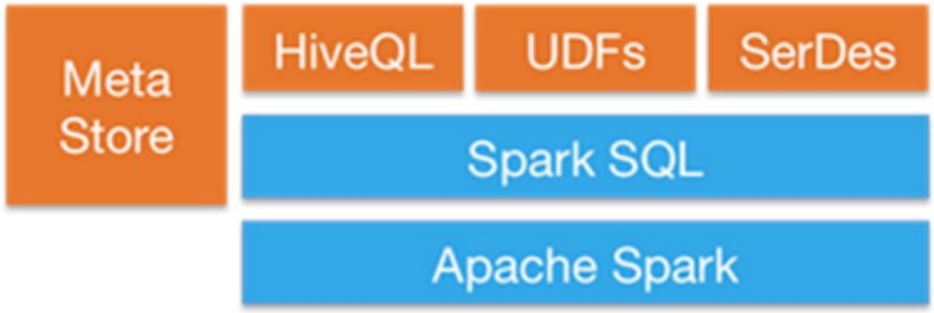


Benchmark Query 2

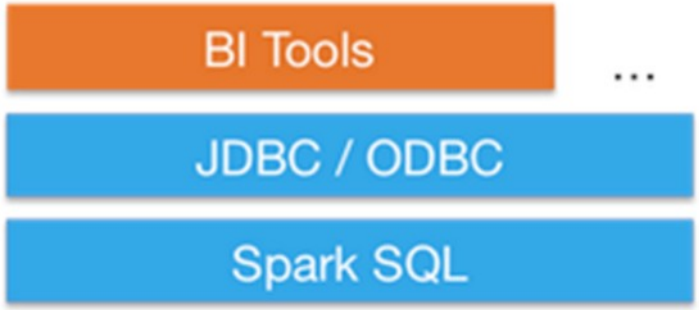
```
SELECT sourceIP, AVG(pageRank), SUM(adRevenue) AS earnings FROM rankings
AS R, userVisits AS V ON R.pageURL = V.destURL WHERE V.visitDate BETWEEN
'1999-01-01' AND '2000-01-01' GROUP BY V.sourceIP ORDER BY earnings DESC
LIMIT 1;
```



Spark SQL (Shark)



Spark SQL can use existing Hive metastores, SerDes, and UDFs.



Use your existing BI tools to query big data.

Agenda

- NoSQL:
  - Hbase
- Scripting in Big Data:
  - Hive / Pig
  - Other tools
    - Impala
    - Spark SQL
    - Apache Drill

User Interfaces

- REST
- CLI (command line)
- Native API
- JDCB / ODBC

Processing Interfaces

- SQL
- DrQL (Drill query language)
- MongoQL
- ...

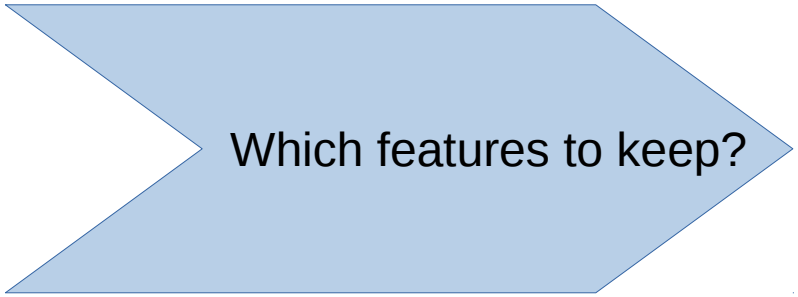
Data Interfaces

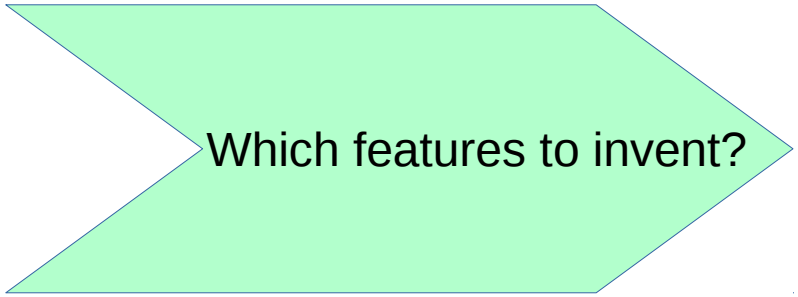
- Cassandra
- Hbase
- Hadoop HDFS
- MongoDB
- RDBMS
- (pluggable data sources)

In early days of NoSQL, the underlying data structure was still kept in somewhat tabular format. Goal in drill is to be able to talk to any data model. Transformations should happen at read/write time: Data should be self-describing or schema less

Some slides from Apache Drill: M. C. Srivas, CTO & Co-Founder, MapR, May 14, 2014

Apache Drill: Rethink SQL for Big Data





- **ANSI SQL**
  - Familiar and ubiquitous
- **Performance**
  - Interactive nature crucial for BI/Analytics
- **One technology**
  - Painful to manage different technologies
- **Enterprise ready**
  - System-of-record, HA, DR, Security, Multi-tenancy, ...

- **Flexible data-model**
  - Allow schemas to evolve rapidly
  - Support semi-structured data types
- **Agility**
  - Self-service possible when developer and DBA is same
- **Scalability**
  - In all dimensions: data, speed, schemas, processes, management

Self-Describing Data

ORACLE TERADATA MySQL

APACHE HBASE mongoDB

Centralized schema

- Static
- Managed by the DBAs
- In a centralized repository

Long, meticulous data preparation process (ETL, create/alter schema, etc.)
– can take 6-18 months

Self-describing, or schema-less, data

- Dynamic/evolving
- Managed by the applications
- Embedded in the data

Less schema, more suitable for data that has higher volume, variety and velocity

Apache Drill: Data Source is in the Query

```
select      timestamp, message
from        dfs1.logs.`AppServerLogs/2014/Jan/p001.parquet`
where       errorLevel > 2
```

This is a *cluster* in Apache Drill
- DFS
- HBase
- Hive meta-store

A *work-space*
- Typically a sub-directory
- HIVE database

A *table*
- pathnames
- Hbase table
- Hive table

Note: This is a select on a file

Combine Data Sources on the fly

- JSON
- CSV
- ORC (ie, all Hive types)
- Parquet
- HBase tables
- … can combine them

```
Select  USERS.name,  PROF.emails.work
from
  dfs.logs.`/data/logs`  LOGS,
  dfs.users.`/profiles.json`  USERS,
where
   LOGS.uid = USERS.uid   and
   errorLevel > 5
order by  count(*);
```

Queries on Entire Directories

```
// On a file
    select      errorLevel, count(*)
    from        dfs.logs.`/AppServerLogs/2014/Jan/part0001.parquet`
    group by    errorLevel;

// On the entire data collection: all years, all months
    select      errorLevel, count(*)
    from        dfs.logs.`/AppServerLogs`
    group by    errorLevel;
```

Direct Queries on Nested Data

```
{  name: classic
   fillings: [
     { name:  sugar cal:  400 }]}

{ name: choco
  fillings: [
     { name: sugar  cal: 400 }
     { name: chocolate cal: 300 }]}

{  name: bostoncreme
   fillings:  [
     { name:  sugar  cal: 400 }
     { name:  cream  cal: 1000 }
     { name:  jelly  cal: 600 }]}
```

// Flattening maps in JSON, parquet and other nested records

```
select name, flatten(fillings) as f
  from   dfs.users.`/donuts.json`
  where  f.cal < 300;
```

// lists the fillings < 300 calories

Several applications may add object layers to JSON; flattening removes those and creates a lean data structure

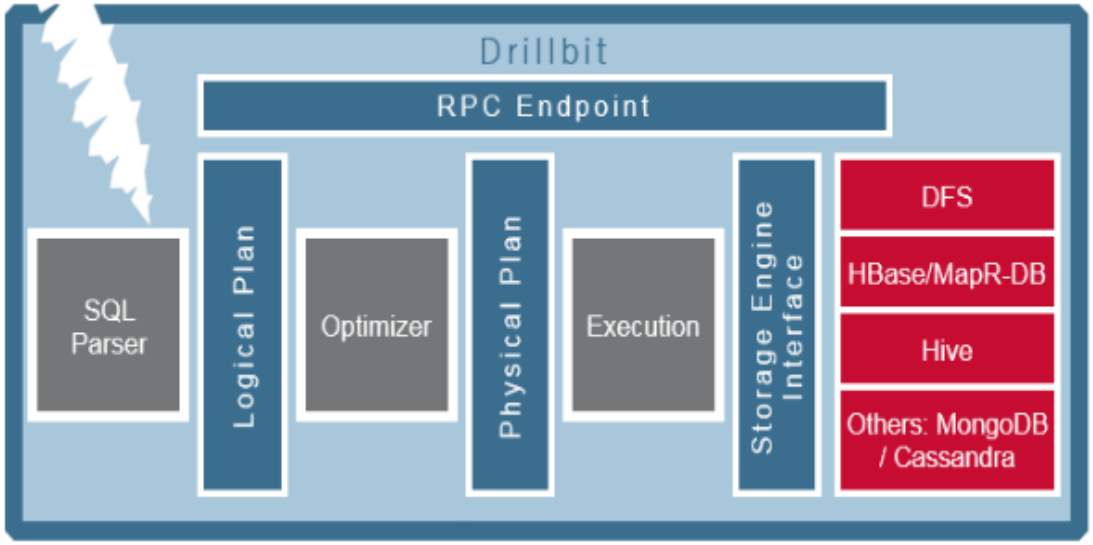
De-centralized Meta Data

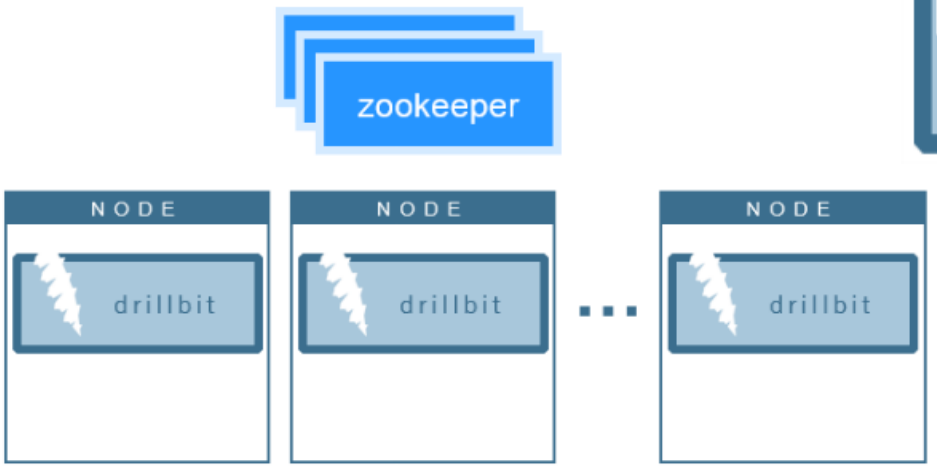
// count the number of tweets per customer, where the customers are in Hive, and their tweets are in HBase. Note that the hbase data has no meta-data information

```
select   c.customerName,  hb.tweets.count
from     hive.CustomersDB.`Customers` c
join     hbase.user.`SocialData` hb
on       c.customerId = convert_from( hb.rowkey, UTF-8);
```

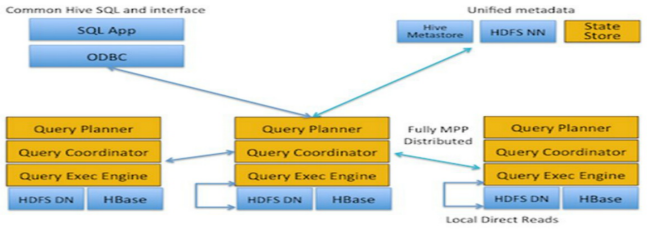
Apache Drill: Basic Execution Process

- ZooKeeper maintains cluster membership information
- Drillbits use ZooKeeper to find other Drillbits in the cluster
- Client uses ZooKeeper to find Drillbits to submit a query



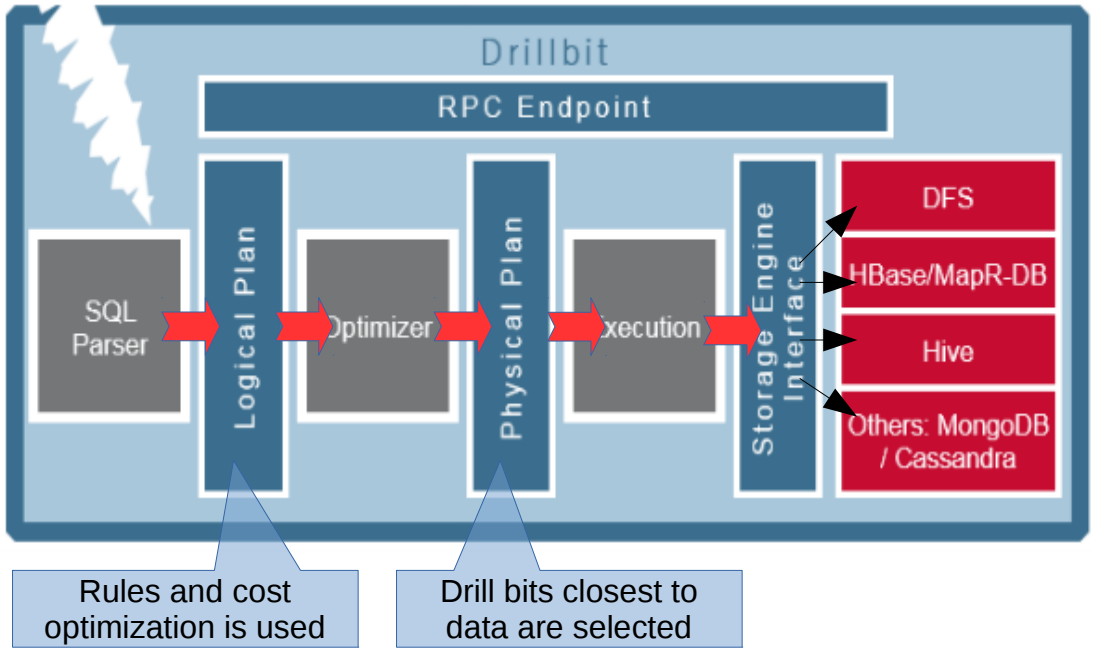


Drillbit is the equivalent of Query Planner, Query Coordinator and Query Exec Engine of Impala



https://www.mapr.com/blog/apache-drill-architecture-ultimate-guide

Planning & Execution



Different parts of a drillbit; Logical flow is from left to right

Drillbit that accepts the initial query becomes the Foreman for the request.

Every Drillbit contains all services and capabilities of Drill, allowing Drill to scale to thousands of users and thousands of nodes.

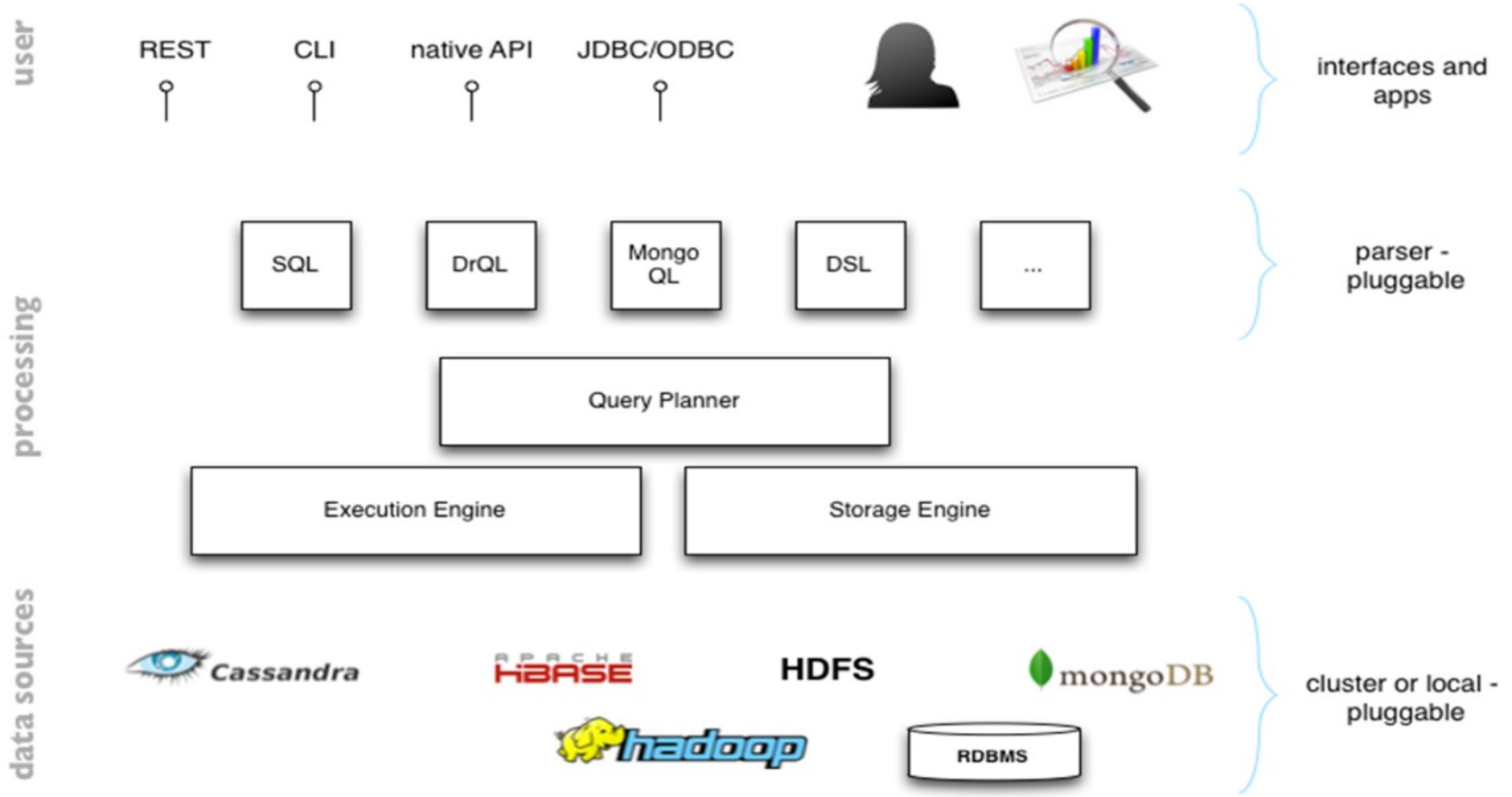
Foreman parses a query into a logical plan made of several fragments.

Foreman gets a list of available Drillbits from ZooKeeper

Foreman selects appropriate nodes to execute various query plan fragments to maximize data locality.

https://www.mapr.com/blog/apache-drill-architecture-ultimate-guide

Apache Drill – Architecture



Inspire...Educate...Transform.

International School of Engineering

Plot 63/A, 1st Floor, Road # 13, Film Nagar, Jubilee Hills, Hyderabad - 500 033

For Individuals:+91-9502334561/63 or 040-65743991

For Corporates:+91-9618483483

Web:http://www.insofe.edu.in

Facebook:https://www.facebook.com/insofe

Twitter:https://twitter.com/Insofeedu

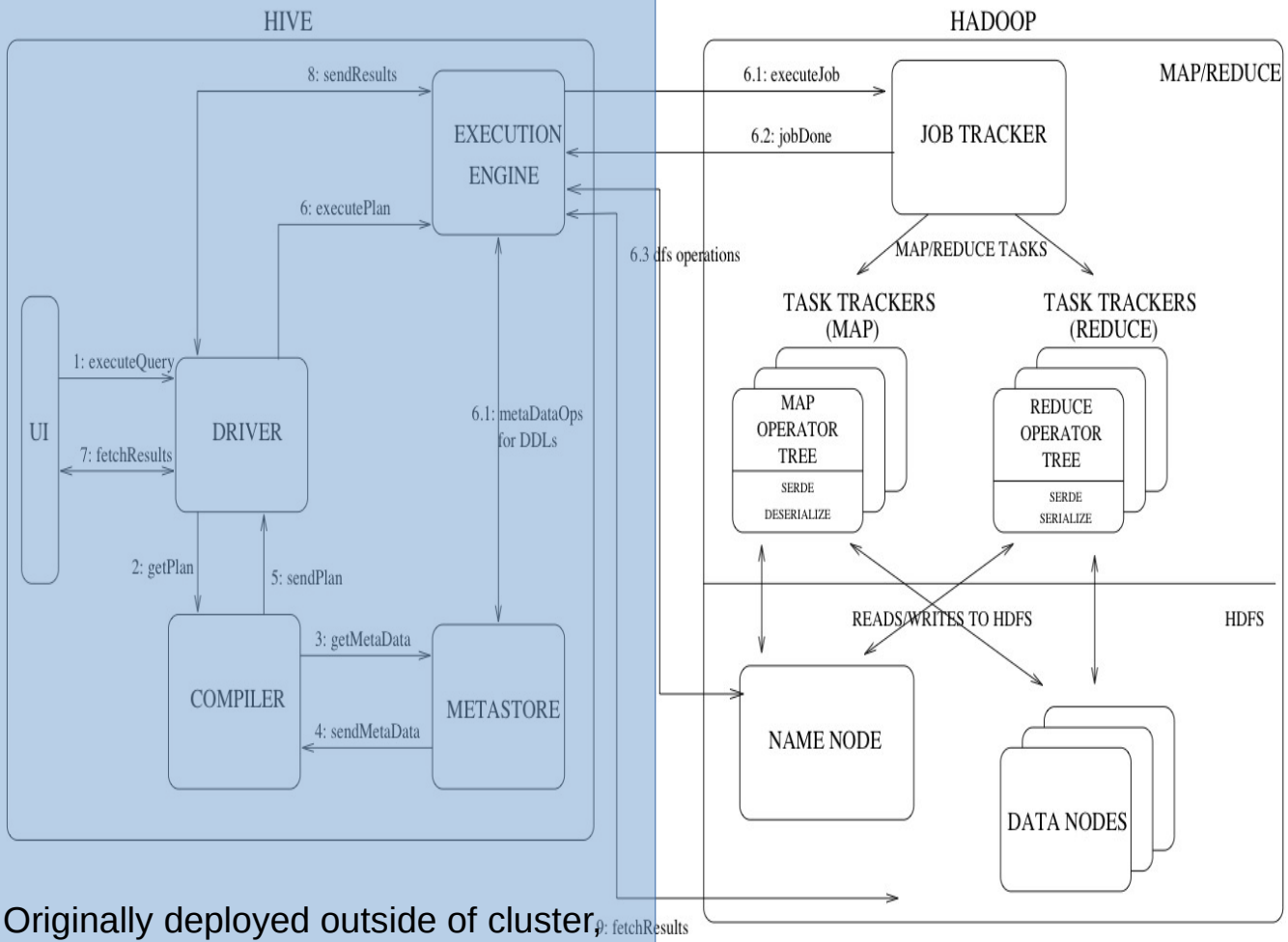
YouTube:http://www.youtube.com/InsofeVideos

SlideShare:http://www.slideshare.net/INSOFE

LinkedIn:http://www.linkedin.com/company/international-school-of-engineering

Additional Material:
Hive

Hive Architecture: Old

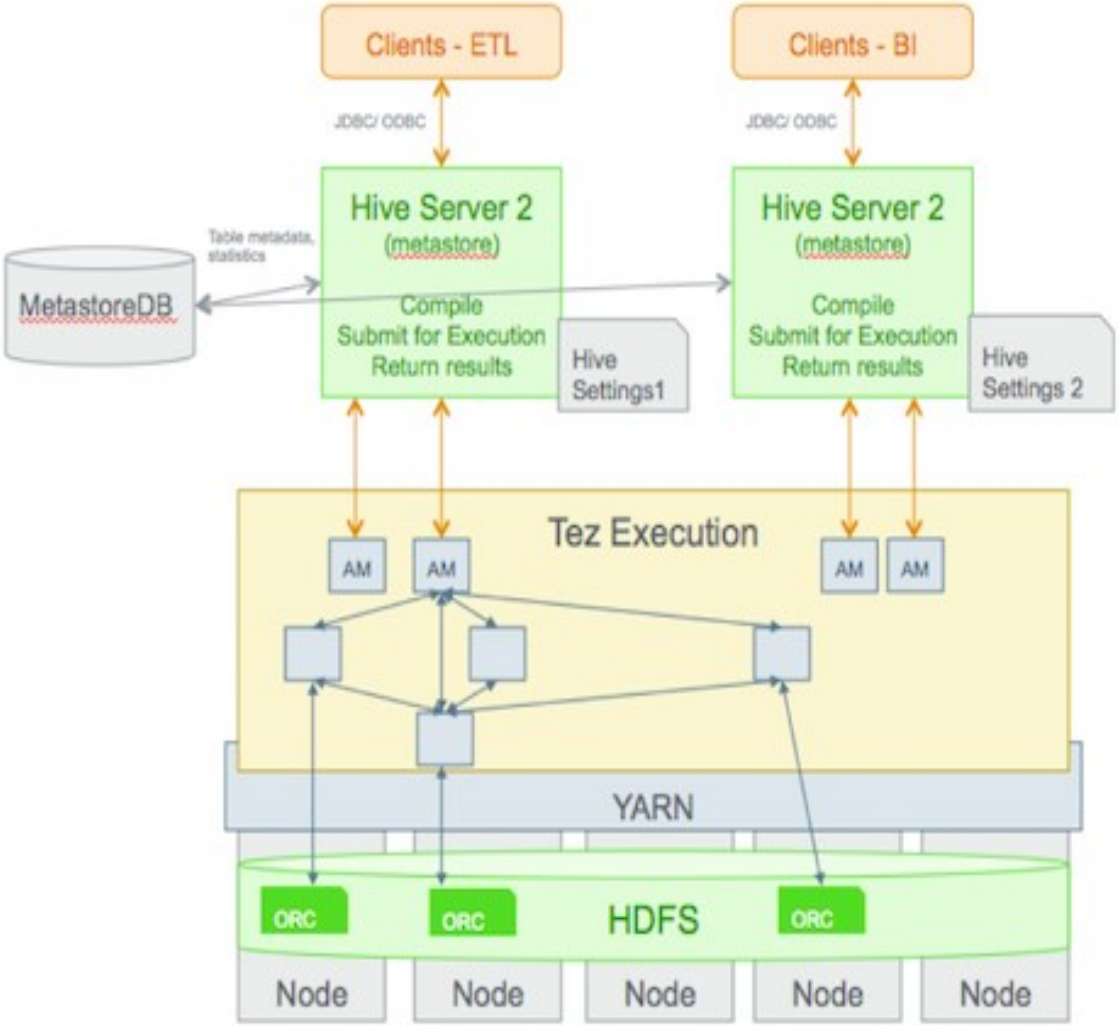


Originally deployed outside of cluster, possibly client machine

- Metastore: stores system catalog
- Driver: manages life cycle of HiveQL query, session handle and session statistics
- Query compiler: Compiles HiveQL into a directed acyclic graph of map/reduce tasks
- Execution engines: Executes the tasks in proper dependency order; interacts with Hadoop
- Client components: CLI, web interface, jdbc/odbc inteface
  - Extensibility interface include Serialize/Deserialize, User Defined Functions (UDFs) and User Defined Aggregate Function.

Image: https://cwiki.apache.org/confluence/display/Hive/Design

Hive Architecture: HiveServer2



Now, separate service on top of Tez+zookeeper

- Metastore: stores system catalog
- HiveServer: Combines drivers and query compilers
  - Driver: manages life cycle of HiveQL query, session handle and session statistics
  - Query compiler: Compiles HiveQL into a directed acyclic graph of map/reduce tasks
- Tez Execution engine: Executes the tasks in proper dependency order; interacts with Hadoop
- Client components: CLI, web interface, jdbc/odbc inteface
- Extensibility interface include SerDe, User Defined Functions (UDFs) and User Defined Aggregate Function.

Image: http://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.3.0/bk_performance_tuning/content/ch_hive_architectural_overview.html

Hive Metastore

- To reduce the time to perform semantic checks during query execution, Hive keeps its metadata in a relational data store.
  - Apache Derby, a light weight embedded SQL database
  - Can be easily replaced with Apace MySQL
  - Example of semantic check: Searching on a column from a table that has no such column, Incorrectly configured database being locked by Hive
- Schema not shared between users – each user has their own instance of Derby.
- Stored in Hive opening directory /metastore_db

Hive Meta Data Model

- Hive structures data into well-understood database concepts such as: tables, rows, cols, partitions
  - It supports primitive types: integers, floats, doubles, and strings
- Hive also supports:
  - associative arrays: map<key-type, value-type>
  - Lists: list<element type>
  - Structs: struct<file name: file type…>
- SerDe: serialize and deserialize API is used to move data in and out of tables

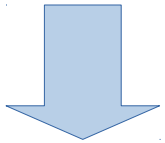
Hive Meta Data Storage

- Tables are logical data units; table metadata associates the data in the table to hdfs directories.
- HDFS namespace: tables (HDFS directory), partition (HDFS subdirectory), buckets (subdirectories within partition)
- E.g., /user/hive/warehouse/test_table is a HDFS directory

Hive: Behind the Scene

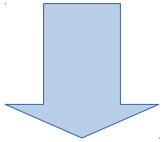
```
SELECT s.word, s.freq, k.freq FROM shakespeare s
  JOIN bible k ON (s.word = k.word) WHERE s.freq >= 1 AND k.freq >= 1

  ORDER BY s.freq DESC LIMIT 10;
```

Query is parsed into a syntax tree



```
(TOK_QUERY (TOK_FROM (TOK_JOIN (TOK_TABREF shakespeare s) (TOK_TABREF
bible k) (= (. (TOK_TABLE_OR_COL s) word) (. (TOK_TABLE_OR_COL k) word))))
(TOK_INSERT (TOK_DESTINATION (TOK_DIR TOK_TMP_FILE)) (TOK_SELECT
(TOK_SELEXPR (. (TOK_TABLE_OR_COL s) word)) (TOK_SELEXPR (.
(TOK_TABLE_OR_COL s) freq)) (TOK_SELEXPR (. (TOK_TABLE_OR_COL k) freq)))
(TOK_WHERE (AND (>= (. (TOK_TABLE_OR_COL s) freq) 1) (>= (.
(TOK_TABLE_OR_COL k) freq) 1))) (TOK_ORDERBY (TOK_TABSORTCOLNAMEDESC (.
(TOK_TABLE_OR_COL s) freq))) (TOK_LIMIT 10)))
```

Hive then converts the syntax trees into MR jobs



Hive: Behind the Scene (2)

```
STAGE DEPENDENCIES:
  Stage-1 is a root stage
  Stage-2 depends on stages: Stage-1
  Stage-0 is a root stage

STAGE PLANS:
  Stage: Stage-1
    Map Reduce
      Alias -> Map Operator Tree:
        s
          TableScan
            alias: s
            Filter Operator
              predicate:
                  expr: (freq >= 1)
                  type: boolean
              Reduce Output Operator
                key expressions:
                      expr: word
                      type: string
                sort order: +
                Map-reduce partition columns:
                      expr: word
                      type: string
                tag: 0
                value expressions:
                      expr: freq
                      type: int
                      expr: word
                      type: string
        k
```

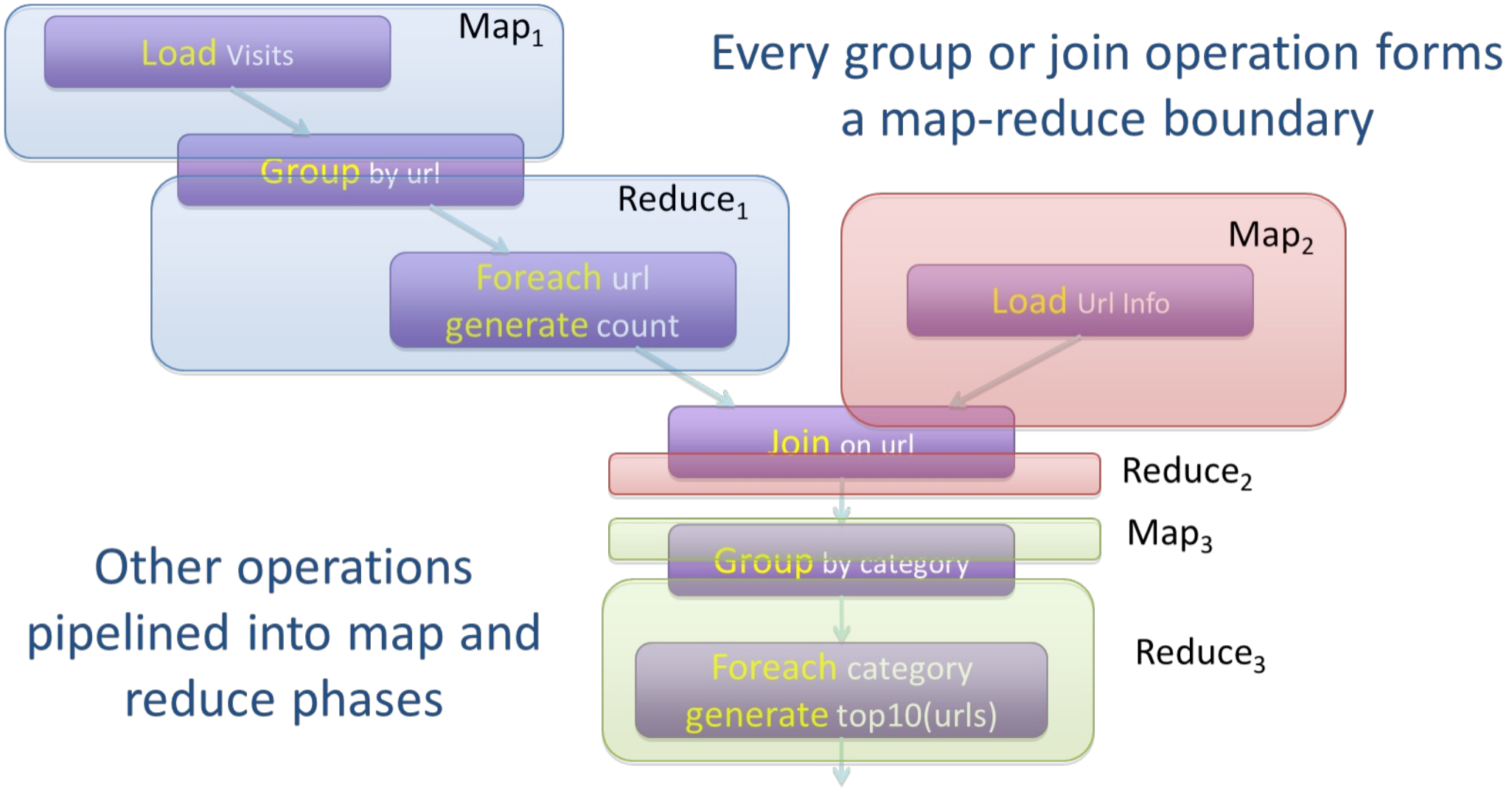
```
TableScan
          alias: k
          Filter Operator
            predicate:
                expr: (freq >= 1)
                type: boolean
            Reduce Output Operator
              key expressions:
                    expr: word
                    type: string
              sort order: +
              Map-reduce partition
columns:
                    expr: word
                    type: string
              tag: 1
              value expressions:
                    expr: freq
                    type: int
```

Word Count in Hive using external UDFs

```
FROM (
MAP doctext USING 'python wc_mapper.py'
AS (word, cnt)
FROM docs
CLUSTER BY word
)
REDUCE word, cnt USING 'pythonwc_reduce.py';
```

Additional Material:
PIG

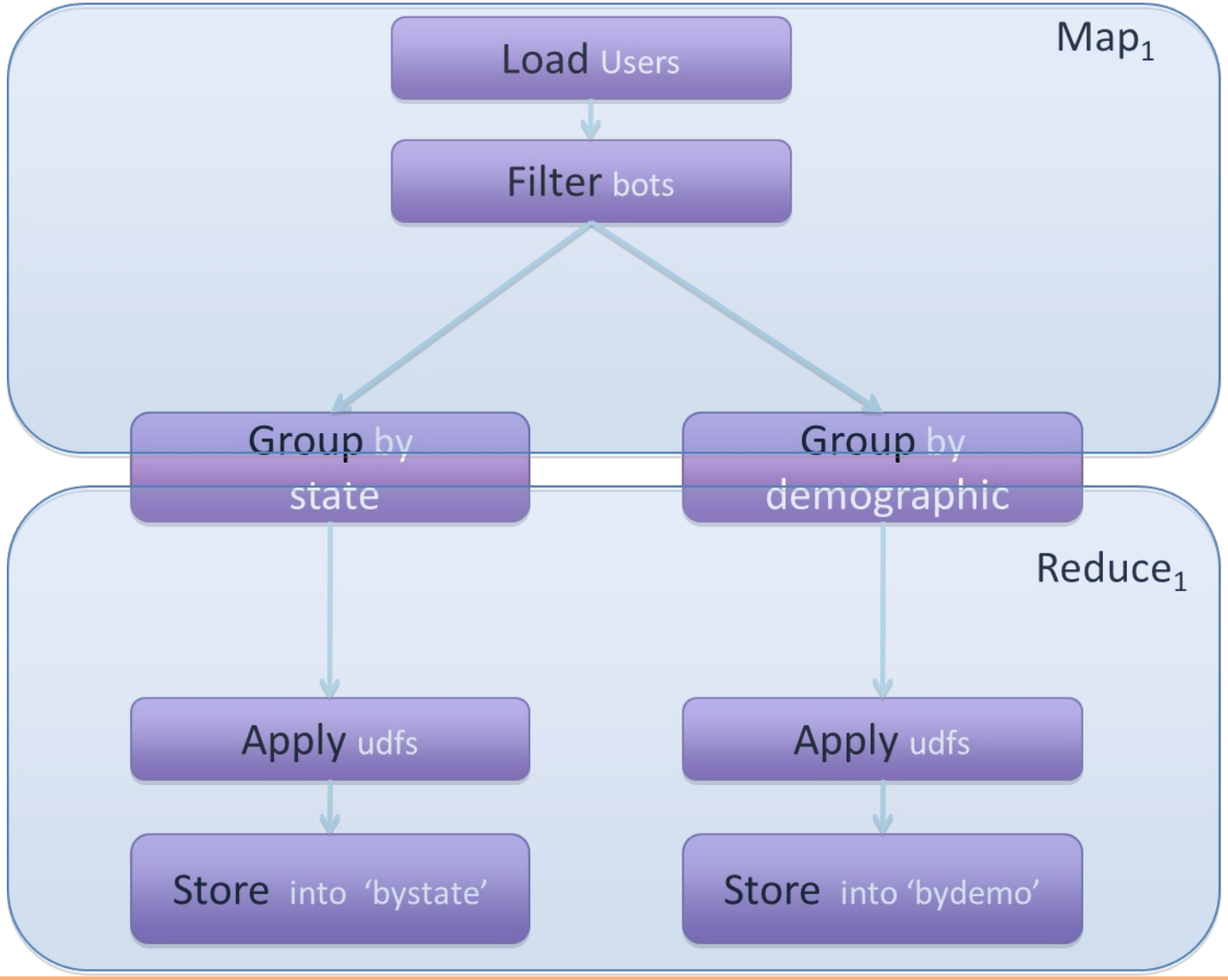
Compilation into MapReduce



Every group or join operation forms a map-reduce boundary

Other operations pipelined into map and reduce phases

Compilation into MapReduce: Multiple DataFlows



Data Model

- Supports four basic types
  - **Atom**: a simple atomic value (*int*, *long*, *double*, *string*)
    - ex: ‘Peter’
  - **Tuple**: a sequence of fields that can be any of the data types
    - ex: (‘Peter’, 14)
  - **Bag**: a collection of tuples of potentially varying structures, can contain duplicates
    - ex: {(‘Peter’), (‘Bob’, (14, 21))}
  - **Map**: an associative array, the key must be a *chararray* but the value can be any type

Nested Data Model

- Pig Latin has a fully-nestable data model with:
  - Atomic values, tuples, bags (lists), and maps

- More natural to programmers than flat tuples
- Avoids expensive joins

Type Casting

- By, default Pig treats data as un-typed
- User can declare types of data at load time

```
log = LOAD 'shakespeare_freq'
          AS (freq:int, word: chararray);
```

- If data type is not declared but script is treating data to be of a certain type, Pig will cast the data to the type and will in all future occurrences, consider the data to be of that type

```
log      = LOAD 'shakespeare_freq' AS (freq, word);
weighted = FOREACH log
           GENERATE freq * 100; --freq cast to int
```

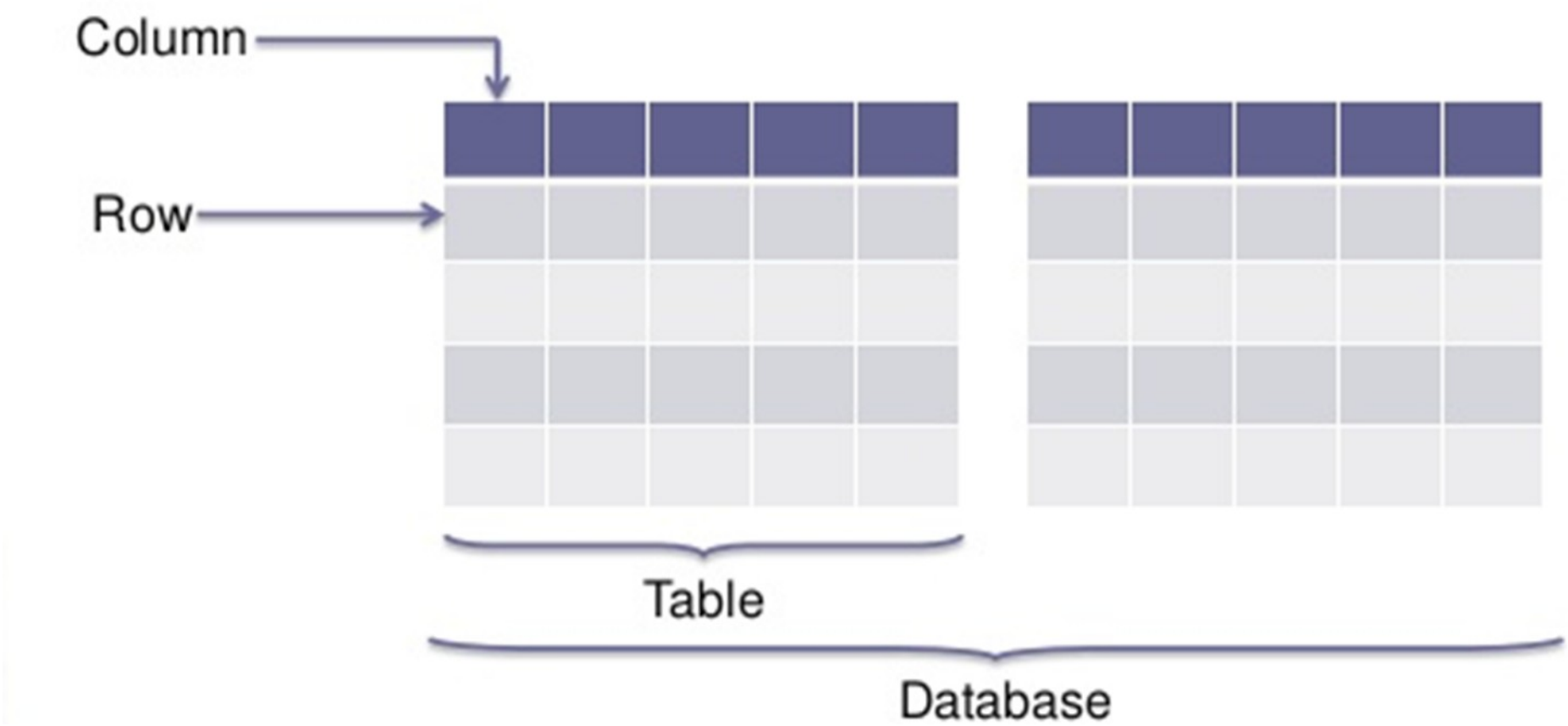
Pig Commands

Pig Command	What it does
load	Read data from file system.
store	Write data to file system.
foreach	Apply expression to each record and output one or more records.
filter	Apply predicate and remove records that do not return true.
group/cogroup	Collect records with the same key from one or more inputs.
join	Join two or more inputs based on a key.
order	Sort records based on a key.
distinct	Remove duplicate records.
union	Merge two data sets.
split	Split data into 2 or more sets, based on filter conditions.
stream	Send all records through a user provided binary.
dump	Write output to stdout.
limit	Limit the number of records.

Additional Material
NoSQL

A Quick Review of SQL

Basic Terminology



Relational Databases - Fundamentals

- Data is organized into tables: rows & columns
- Each row represents an instance of an entity
- Each column represents an attribute of an entity
- Metadata describes each table column
- Relationships between entities are represented by values stored in the columns of the corresponding tables (keys)
- Accessible through Structured Query Language (SQL)

Metadata & Data Table

- Ex: Gene sequence DB for organisms

Name	Type	Max Length	Description
Name	Alphanumeric	100	Organism name
Size	Integer	10	Genome length (bases)
Gc	Float	5	Percent GC
Accession	Alphanumeric	10	Accession number
Release	Date	8	Release date
Center	Alphanumeric	100	Genome center name
Sequence	Alphanumeric	Variable	Sequence

Name	Size	Gc	Accession	Release	Center	Sequence
Escherichia coli K12	4,640,000	50	NC_000913	09/05/1997	Univ. Wisconsin	AGCTTTTCATT…
Streptococcus pneumoniae R6	2,040,000	40	NC_003098	09/07/2001	Eli Lilly and Company	TTGAAAGAAAA…
…						

SQL

- ANSI (American National Standards Institute) standard computer language for accessing and manipulating database systems.
- SQL statements are used to retrieve and update data in a database.
- Includes:

  Data Manipulation Language (DML)

  Data Definition Language (DDL)

  Data Control Language (DCL)

  Transaction Control Language (TCL)

SQL Examples

DDL

```
CREATE DATABASE Microbial;

CREATE TABLE Organism (
        Name varchar(100)
        Size int(10)
        Gc decimal(5)
        Accession varchar(10)
        Release date(8)
        Center varchar(100));

ALTER TABLE Organism ADD Sequence
    varchar;
DROP TABLE Organism;
```

DML

```
SELECT *
FROM Organism, Gene
WHERE
        Organism.Name=“Escherichia
    coli K12” AND
    Organism.Accession=Gene.OAcces
    sion AND Gene.Start>=1,000,000
    AND Gene.End<=2,000,000
```

DDL: Data Definition Language
DML: Data Manipulation Language
DCL: Data Control Langauge
TCL: Transaction Control Language

A Quick Review of No-SQL

Key-Value NoSQL Stores

Typically extremely simple interfaces

- Data model: (key, value) pairs
- Operations: Insert(key,value), Fetch(key), Update(key), Delete(key)
- Some allow (non-uniform) columns within value
- Some allow Fetch on range of keys

Implementation: efficiency, scalability, fault-tolerance

- Records distributed to nodes based on key
- Replication
- Single-record transactions, “eventual consistency”

Key-Value NoSQL: Voldemort

- Voldemort: Created by linkedIn
  - Written in Java
  - Focuses on Availability and Partition Tolerance
  - All data is partitioned (Divided into parts; Each part replicated on multiple servers)
  - Data versioning using Vector Clocks

Document Stores

Like Key-Value Stores except value is document

- Data model: (key, document) pairs
- Document: JSON, XML, other semistructured formats
- Basic operations: Insert(key,document), Fetch(key), Update(key), Delete(key)
- Also Fetch based on document contents

Example systems

- CouchDB, MongoDB (Implemented in C++, commercial version of CouchDB), SimpleDB etc

Document NoSQL: MongoDB

- Development started in 2007
- Commercially supported and developed by 10Gen
- Stores documents using BSON
- Supports AdHoc queries
  - Can query against embedded objects and arrays
- Support multiples types of indexing
- Officially supported drivers available for multiple languages
  - C, C++, Java, Javascript, Perl, PHP, Python and Ruby
- Community supported drivers include:
  - Scala, Node.js, Haskell, Erlang, Smalltalk
- Replication uses a master/slave model
- Scales horizontally via sharding
- Written C++

Community version is CouchDB

MongoDB JSON Example


```
{
  "_id": "guid goes here",
  "_rev": "314159",
  "type": "abstract",
  "author": "Keith W. Hare"
  "title": "SQL Standard and NoSQL Databases",
  "body": "NoSQL databases (either no-SQL or Not Only SQL)
           are currently a hot topic in some parts of
           computing.",
  "creation_timestamp": "2011/05/10 13:30:00 +0004"
}
```

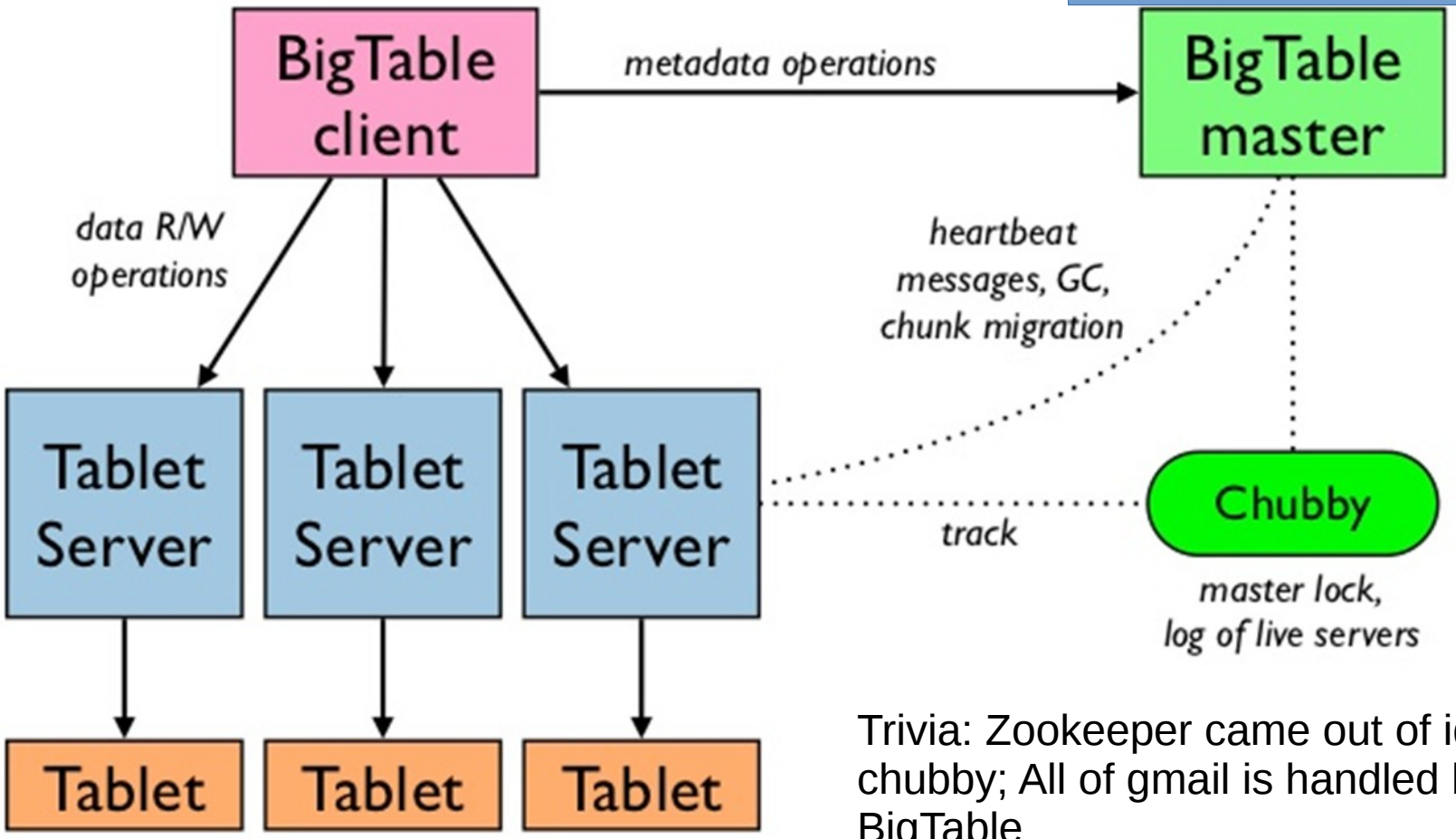
Examples: Web Sites, Online Calendars, Link Sharing, Presentation Sharing

Precursor to HBase

Google Big Table Architecture

A set of row keys are handled by one BigTable client
A set of column families are handled by Tablet Server
Tablets handle columns and time stamps

File system metadata, Access Control Layer, GC, Load balancing



Trivia: Zookeeper came out of ideas in chubby; All of gmail is handled by BigTable