Databases for Big Data – part 2

Valentina Ivanova

IDA, Linköping University

LINKÖPING UNIVERSITY

Outline

- Dynamo
- HBase
- Hive
- Shark

DB rankings – September 2016

Rank			DBMS	Database Model	Score		
Sep 2016	Aug 2016	Sep 2015			Sep 2016	Aug 2016	Sep 2015
1.	1.	1.	Oracle	Relational DBMS	1425.56	-2.16	-37.81
2.	2.	2.	MySQL	Relational DBMS	1354.03	-3.01	+76.28
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1211.55	+6.51	+113.72
4.	↑ 5.	↑ 5.	PostgreSQL	Relational DBMS	316.35	+1.10	+30.18
5.	↓ 4.	↓ 4.	MongoDB	Document store	316.00	-2.49	+15.43
6.	6.	6.	DB2	Relational DBMS	181.19	-4.70	-27.95
7.	7.	↑ 8.	Cassandra	Wide column store	130.49	+0.26	+2.89
8.	8.	↓ 7.	Microsoft Access	Relational DBMS	123.31	-0.74	-22.68
9.	9.	9.	SQLite	Relational DBMS	108.62	-1.24	+0.97
10.	10.	10.	Redis	Key-value store	107.79	+0.47	+7.14
11.	11.	↑ 14.	Elasticsearch	Search engine	96.48	+3.99	+24.93
12.	12.	↑ 13.	Teradata	Relational DBMS	73.06	-0.57	-1.20
13.	13.	↓ 11.	SAP Adaptive Server	Relational DBMS	69.16	-1.88	-17.36
14.	14.	↓ 12.	Solr	Search engine	66.96	+1.19	-14.98
15.	15.	15.	HBase	Wide column store	57.81	+2.30	-1.22
16.	16.	↑ 17.	FileMaker	Relational DBMS	55.35	+0.34	+4.35
17.	17.	↑ 18.	Splunk	Search engine	51.29	+2.38	+9.06
18.	18.	↓ 16.	Hive	Relational DBMS	48.82	+1.01	-4.71
19.	19.	19.	SAP HANA	Relational DBMS	43.42	+0.68	+5.22
20.	20.	↑ 25.	MariaDB	Relational DBMS	38.53	+1.65	+14.31
21.	21.	21.	Neo4j	Graph DBMS	36.37	+0.80	+2.83
22.	↑ 24.	↑ 24.	Couchbase	Document store	28.54	+1.14	+2.28
23.	23.	↓ 22.	Memcached	Key-value store	28.43	+0.74	-3.99
24.	↓ 22.	↓ 20.	Informix	Relational DBMS	28.19	-0.86	-9.76
25.	25.	↑ 28.	Amazon DynamoDB	Document store	27.42	+0.82	+7.43

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

RDBMS

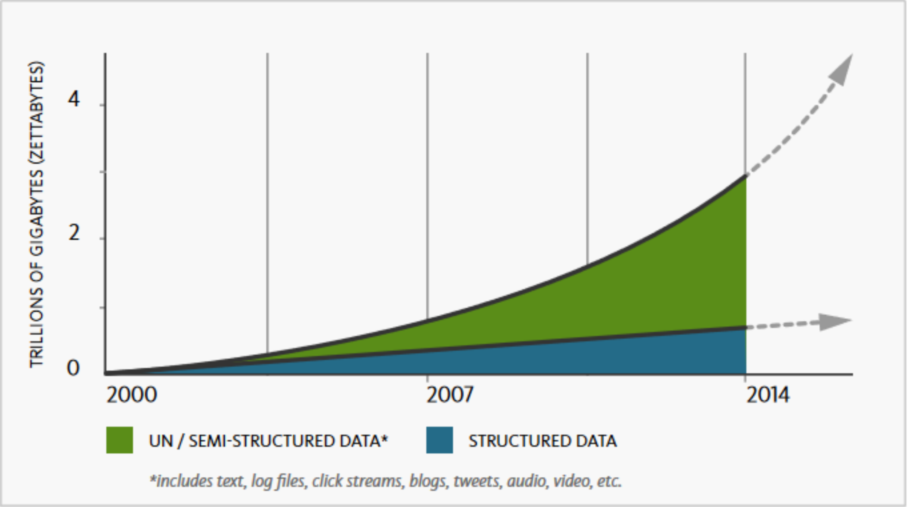
- Established technology
- Transactions support & ACID properties
- Powerful query language - SQL
- Experiences administrators
- Many vendors

Table: Item

item id	name	color	size
45	skirt	white	L
65	dress	red	M

But … – One Size Does Not Fit All[1]

- Requirements have changed:
  - Frequent schema changes, management of unstructured and semi-structured data
  - Huge datasets
  - RDBMSs are not designed to be
    - distributed
    - continuously available
  - High read and write scalability
  - Different applications have different requirements[1]



[1] "One Size Fits All": An Idea Whose Time Has Come and Gone https://cs.brown.edu/~ugur/fits_all.pdf
Figure from: http://www.couchbase.com/sites/default/files/uploads/all/whitepapers/NoSQL-Whitepaper.pdf

NoSQL (not-only-SQL)

- A broad category of disparate solutions
- Simple and flexible non-relational data models
  - schema-on-read vs schema-on-write
- High availability & relax data consistency requirement (CAP theorem)
  - BASE vs ACID
- Easy to distribute – horizontal scalability
  - data are replicated to multiple nodes
- Cheap & easy (or not) to implement (open source)

Distributed (Data Management) Systems

- Number of processing nodes interconnected by a computer network
- Data is stored, replicated, updated and processed across the nodes
- Networks failures are given, not an exception
  - Network is partitioned
  - Communication between nodes is an issue

  → Data consistency vs Availability

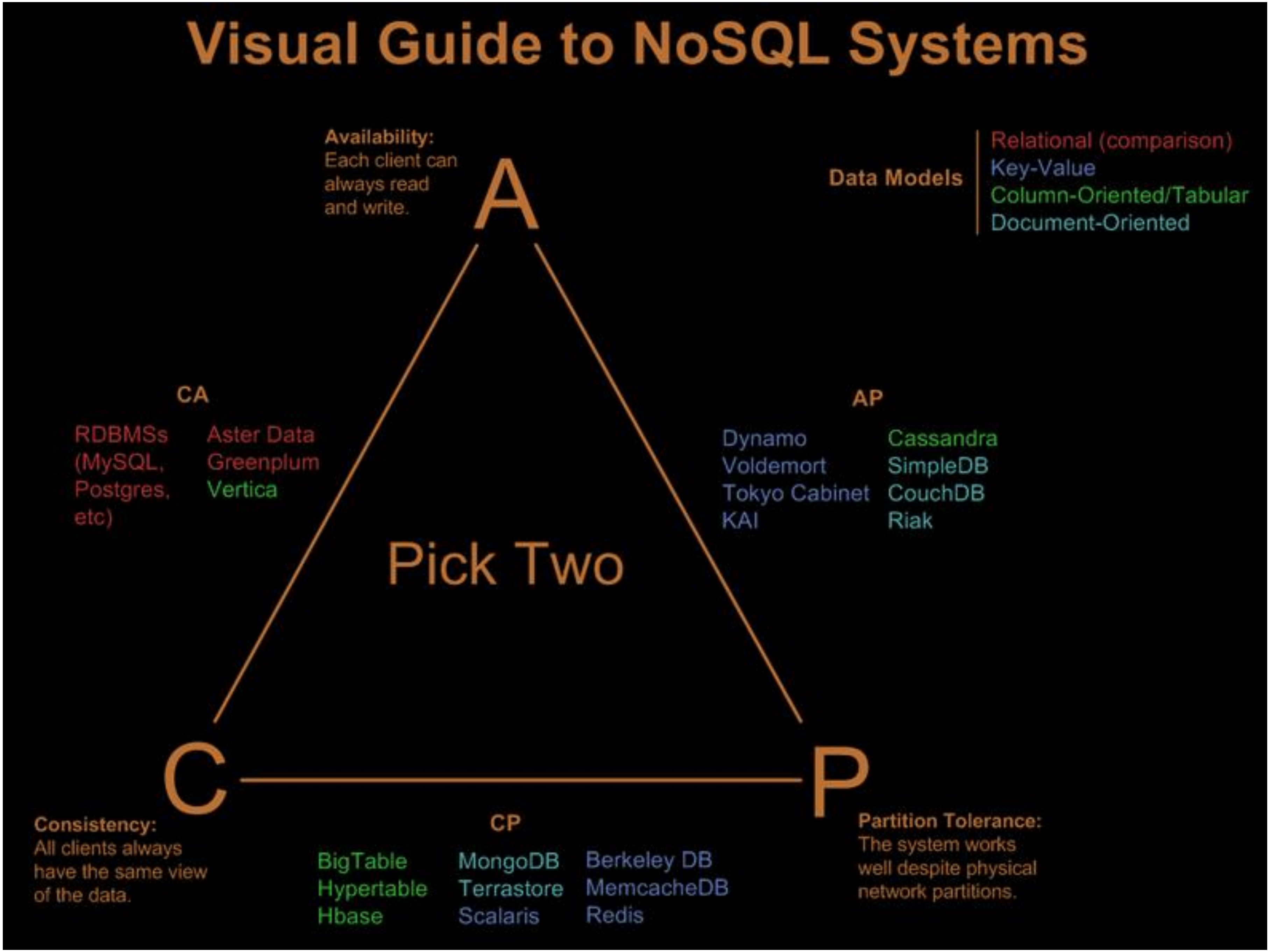


figure from http://blog.nahurst.com/visual-guide-to-nosql-systems

Big Data Analytics Stack

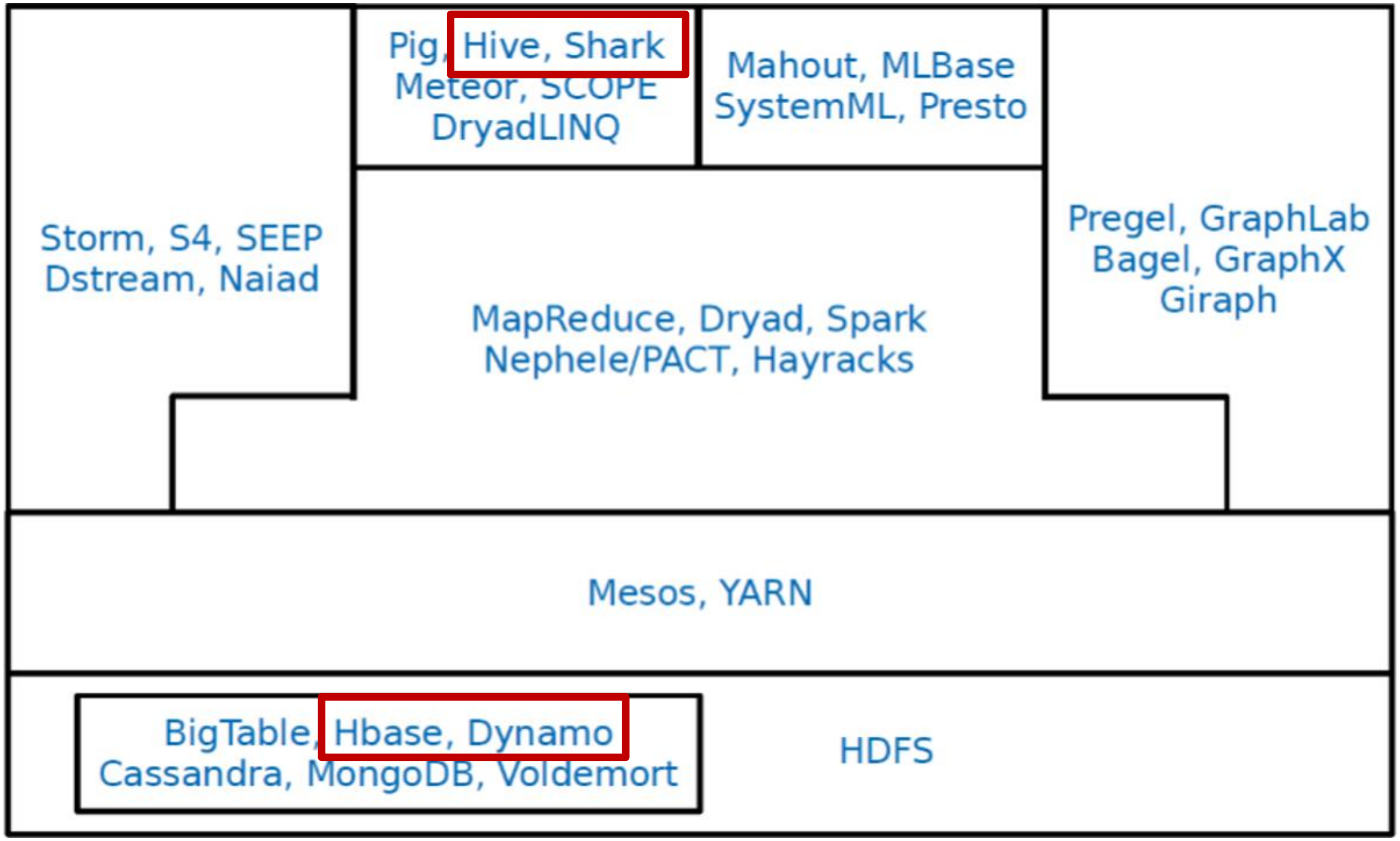
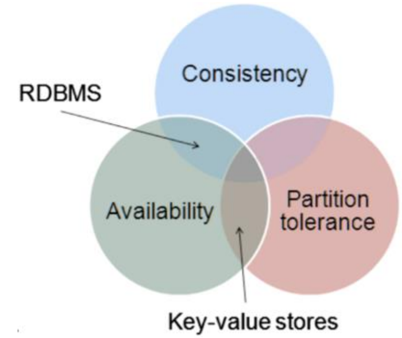


figure from: https://www.sics.se/~amir/dic.htm

Dynamo[Dynamo]

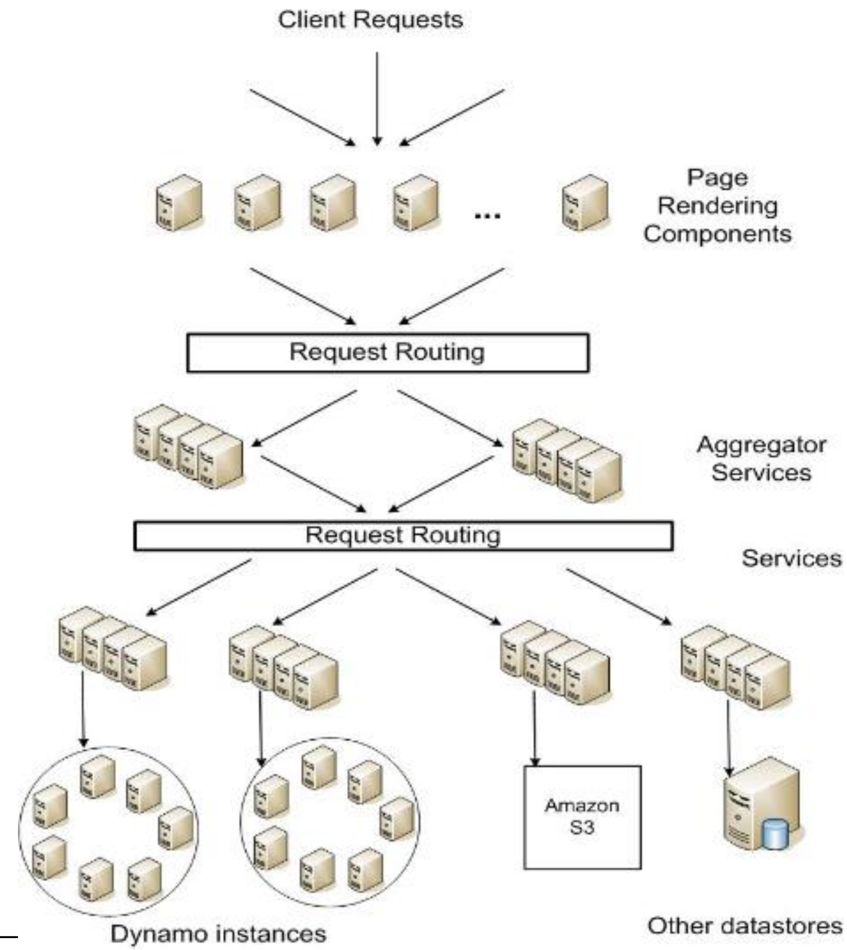
Dynamo



- Highly-available key-value store
- CAP: Availability and Partition Tolerance
- Use case: customer should be able to view and add to the shopping cart during various failure scenarios
  - always serve writes and reads
- Many Amazon services only need primary-key access
  - Best seller lists
  - Customer preferences
  - Product catalog

Amazon's Service Oriented Architecture

- Example: a single page is rendered employing the responses from over 150 services



Dynamo - Techniques

- Consistent hashing
- Object versioning & vector clocks
- Quorum-like techniques
- Gossip-based protocols

Why not RDBMS?

- Amazon's services often store and retrieve data only by key
  - thus do not need complex querying and managing functionalities
- Replication technologies usually favor consistency, not availability
- Cannot scale out easily

Dynamo[Dynamo]

- Storage system requirements:
  - Query model
    - put and get operations to items identified by key
    - binary objects, usually < 1MB
  - ACID-compliant systems have poor availability but Dynamo applications
    - does not require isolation guarantees
    - permits only single key updates

Dynamo[Dynamo]

- System requirements:
  - Efficiency
    - Runs on commodity hardware with Amazon's services having stringent latency requirements
  - No security related requirements

Dynamo[Dynamo]

- Design considerations
  - When to resolve conflicting updates
    - Reads or writes – never reject writes
  - Who to resolve conflicting updates
    - Data store or application
  - Incremental scalability
  - Symmetry
  - Decentralization
  - Heterogeneity

Dynamo - Techniques

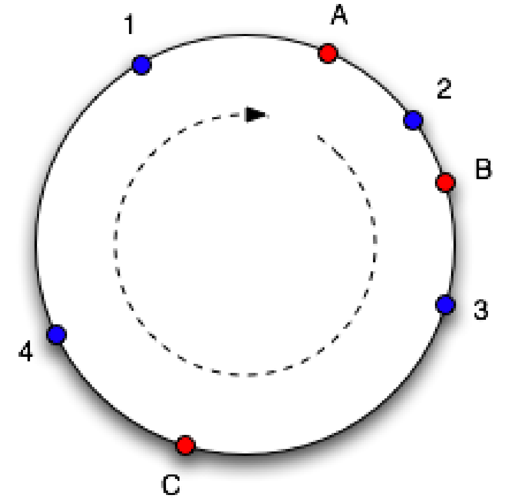
- Consistent hashing
- Quorum-like techniques
- Object versioning & vector clocks

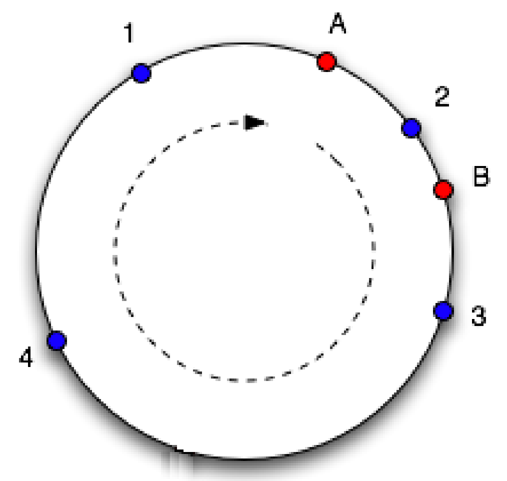
NoSQL: Techniques – Consistent Hashing [Karger]

Basic idea:

- arrange the nodes in a ring
- include hash values of all nodes in hash structure
- calculate hash value of the key to be added/retrieved
- choose node which occurs next clockwise in the ring
- if node is dropped or gets lost, missing data is redistributed to adjacent nodes
- if a new node is added, its hash value is added to the
- the hash realm is repartitioned, and hash data will be transferred to new neighbor

→ no need to update remaining nodes!





Dynamo[Dynamo]

- 128-bit identifier is generated by hashing the key to identify storage node
- Challenges in the basic algorithm
  - Non-uniform data and load distribution
  - Heterogeneity is not accounted for
- Virtual nodes
  - Looks like a single node in the system, but each node can be responsible for more than one virtual node.

Dynamo[Dynamo]

- Each data item is replicated on N hosts
- Each key is assigned to a coordinator node
  - Handles read or write operations

- Preference list contains > N nodes
  - List of nodes responsible for storing the value for a particular key, known by every node
  - Constructed by skipping positions in the ring
  - Nodes in different data centers

Dynamo[Dynamo]

- System architecture
  - get(key) and put(key, context, object)
    - Context stores the object version
  - Quorum protocol – N, W, R
    - N – number of nodes that store replicas
    - R – number of nodes for a successful read
    - W – number of nodes for a successful write
    - R + W > N strong consistency
      - Latency of get (or put) depends on the slowest node
    - R + W ≤ N eventual consistency – better latency

Dynamo[Dynamo]

- get(key) and put(key, context, object)
  - Context stores the object version
- Coordinator node handles reads and writes
  - put() - generates a vector clock and sends to N nodes
  - get() - requests all existing version and returns all causality **unrelated** to the client
    - The divergent versions are then reconciled and the reconciled version superseding the current versions is written back.

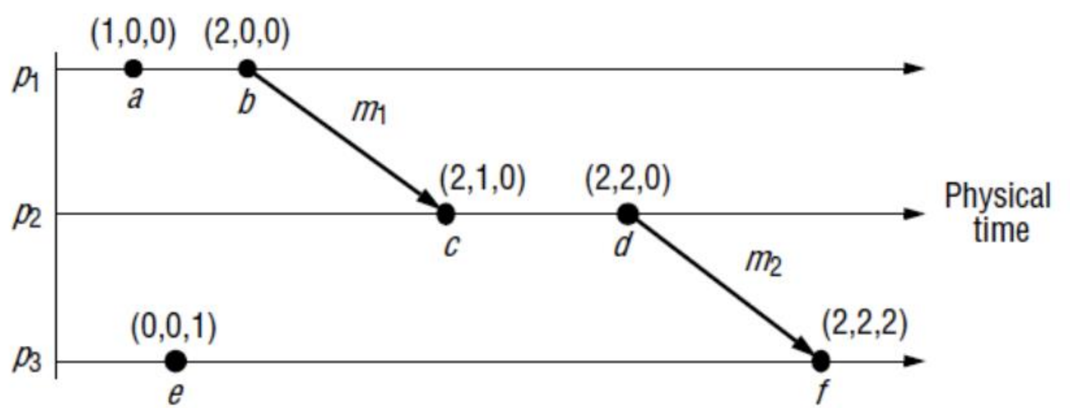
NoSQL: Techniques – Vector Clock[Coulouris]

- A vector clock for a system of N nodes is an array of N integers.
- Each process keeps its own vector clock, V_i , which it uses to timestamp local events.
- Processes piggyback vector timestamps on the messages they send to one another, and there are simple rules for updating the clocks

two events e and e': that $e \rightarrow e' \leftrightarrow V(e) < V(e')$

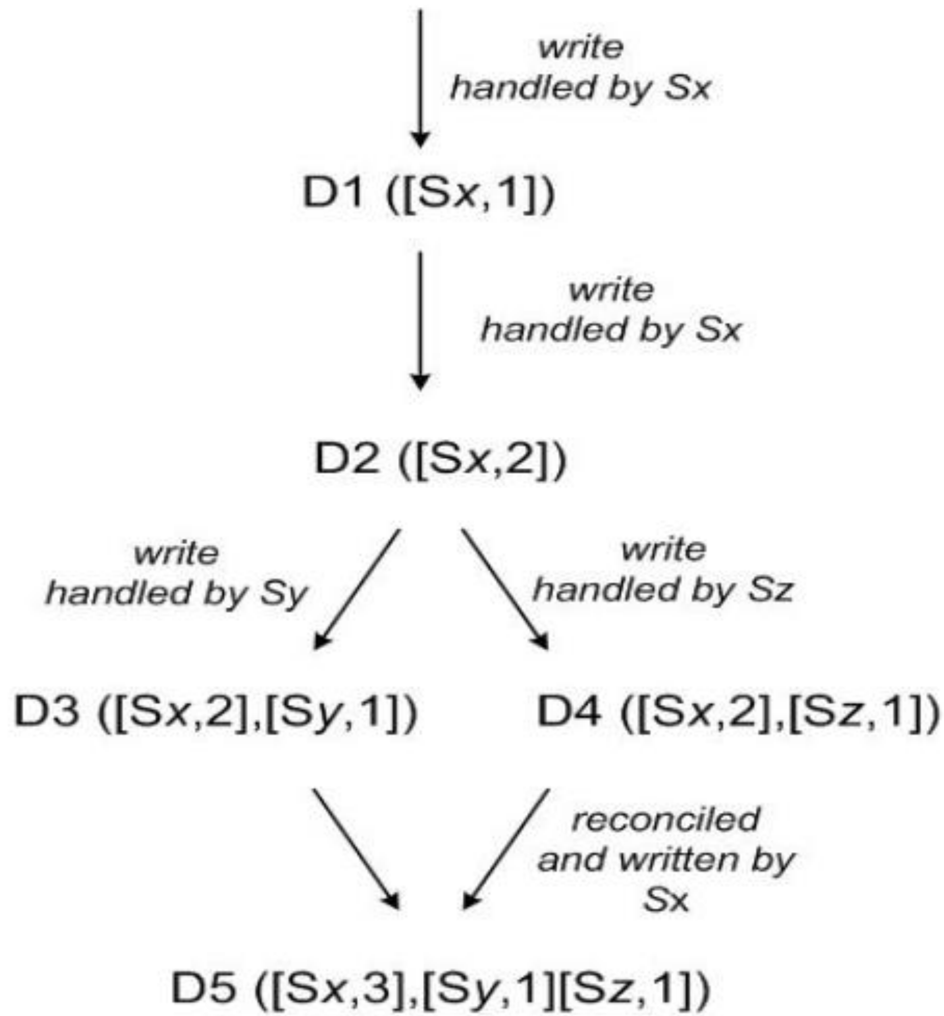
$c \parallel e$ since neither $V(c) \leq V(e)$ nor $V(e) \leq V(c)$

c & e are concurrent



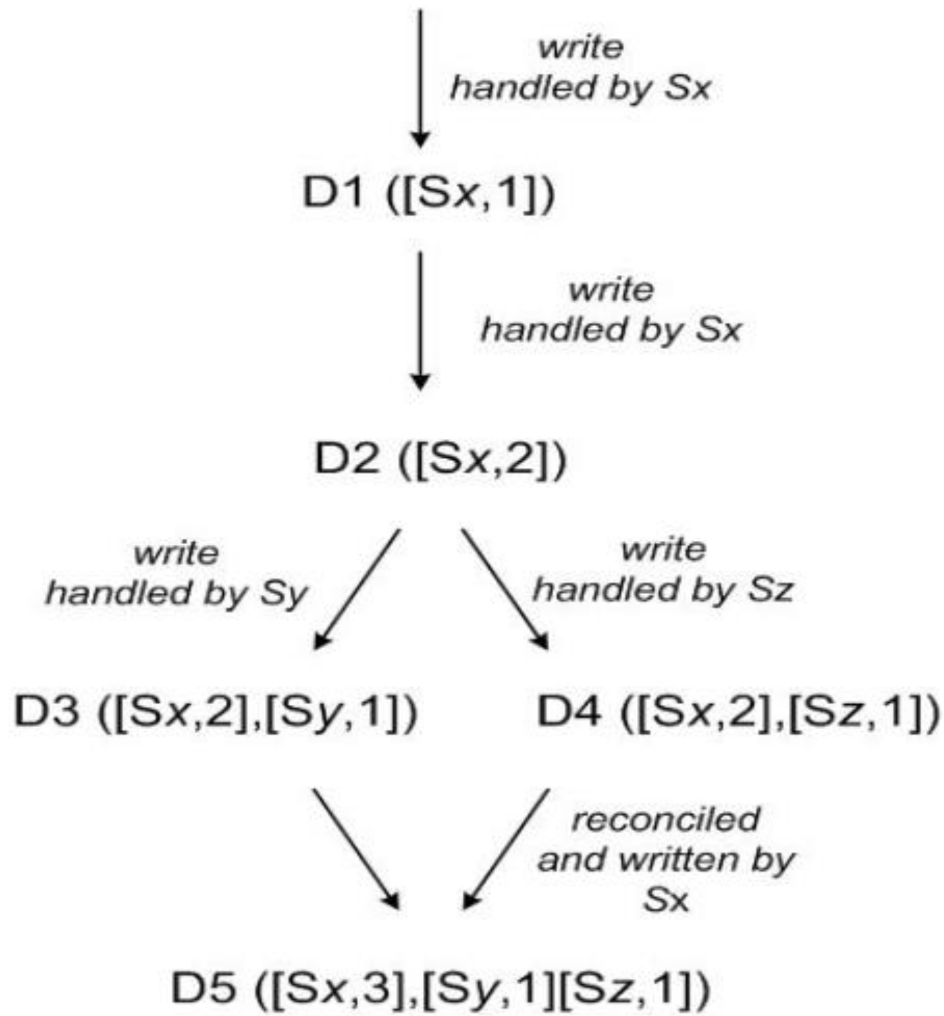
Dynamo - Versioning

- Asynchronous update propagation
- Use case: shopping cart
- Each update is a new, immutable version --> many versions of an object may exist
- Replicas eventually become consistent



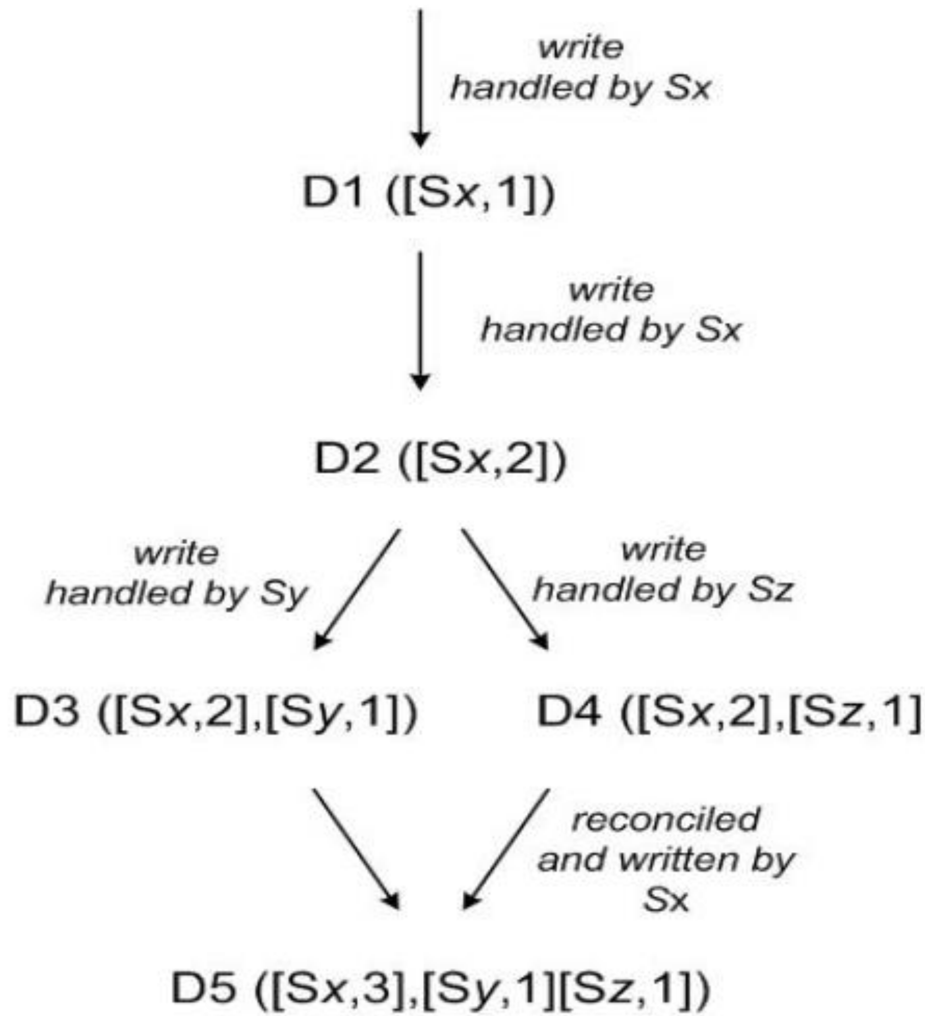
Dynamo - Versioning

- Reconciliation
  - Syntactic
  - Semantic
- Vector clocks
  - Client specifies which version is updating
  - All leave objects are returned if syntactic reconciliation fails



Dynamo - Versioning

- Sx, Sy, Sz – nodes
- D1, D2, D3, D4, D5 – versions of data items
- [Sx, 1] vector clock at Sx
- Divergent versions are rare
  - One version: 99.94%
  - Four versions: 0.00009%



Dynamo[Dynamo]

- Handling failure – hinted handoff
- Sloppy quorum - all read and write operations are performed on the first N healthy nodes from the preference list
  - If a node is temporary down the replica is sent to another
  - The replica will have a hint in its metadata for its intended location
  - After the node recovers it will receive the replica

Dynamo - Summary

- Highly-available key-value store
- CAP: Sacrifices consistency for availability in the pretense of network partitions
- Every node has the same responsibilities
- Consistent hashing
- Vector clocks for replicas reconciliation
- Quorum-like and decentralized replica synchronization protocol

HBase[HBase][Hadoop]

Column-oriented Databases

- Saved data grouped by columns
- Not all values are needed for some queries/applications
  - Analytical databases

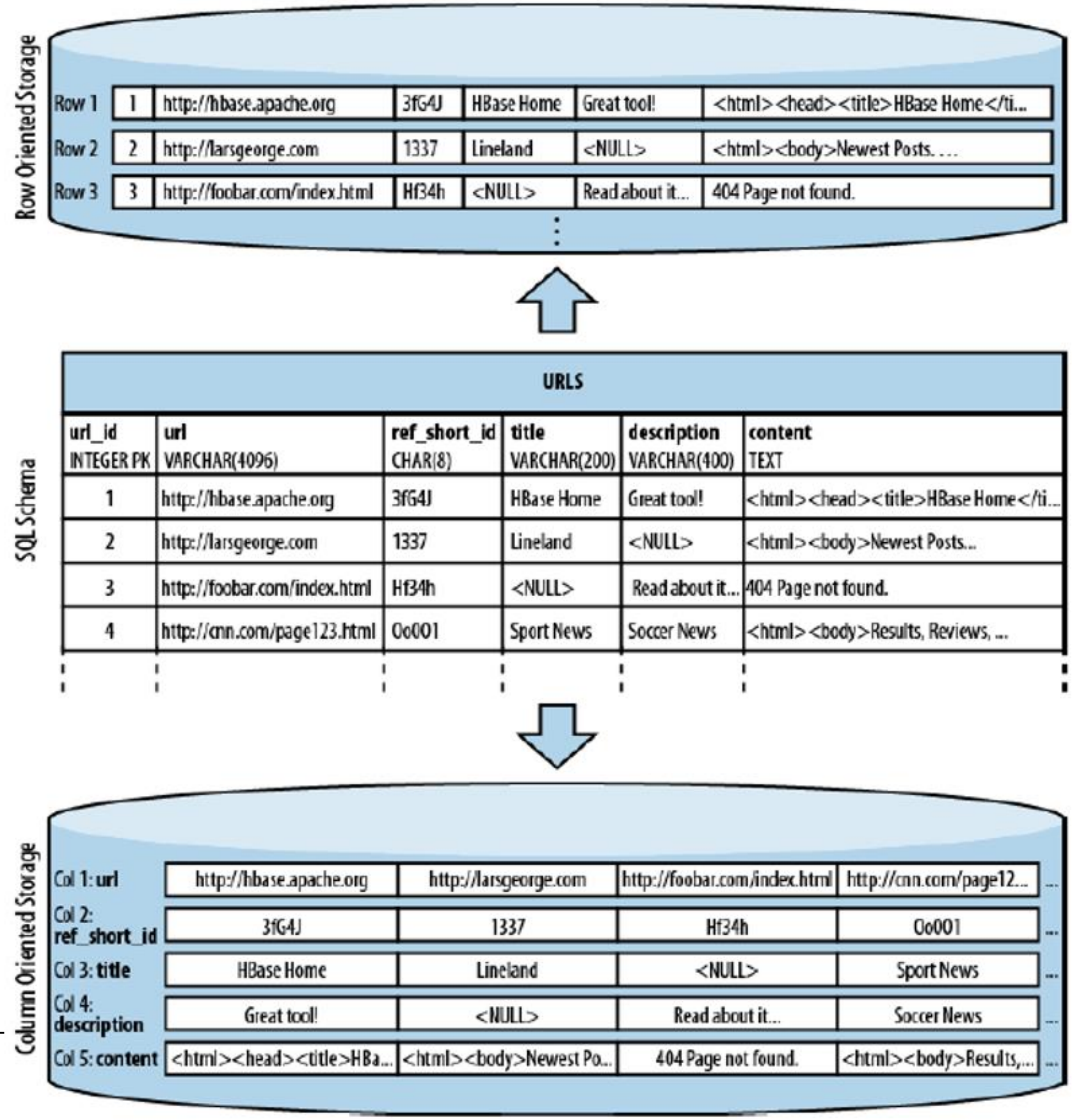
- Leads to
  - Reduced I/O
  - Better compression due to similar values

Column-oriented Model[HBase]

Row-oriented storage

SQL Schema

Column-oriented storage (HBase)



HBase – a Column-family Database

- Hosts very large sparse tables
- Based on Google BigTable and built on top of HDFS
- Provide l*ow-latency* real-time read/write random access on a (sequence of) cell level
- Scales linearly on commodity hardware
- Atomic access to row data
- CAP: provides strong consistency and partition tolerance --> sacrifices availability
- Started at the end of 2006, in May 2010 became Apache Top Level Project

HBase[HBase] Canonical Example – *webtable*

Row key	**Time stamp**	**Family `content`**		**Family `outgoing links`**		**Family `inbound links`**	
		html	**png**	**cnnsi.com**	**my.look.ca**	**news.bbc.com**	**theguardian.com**
com.cnn.www	t9			CNN	cnn.com	cnn.com	cnn.com
	t8		logo.png				
	t6	contents:html = "<html>…"	logo1.png				
	t5	contents:html = "<html>…"					
	t3	contents:html = "<html>…"					

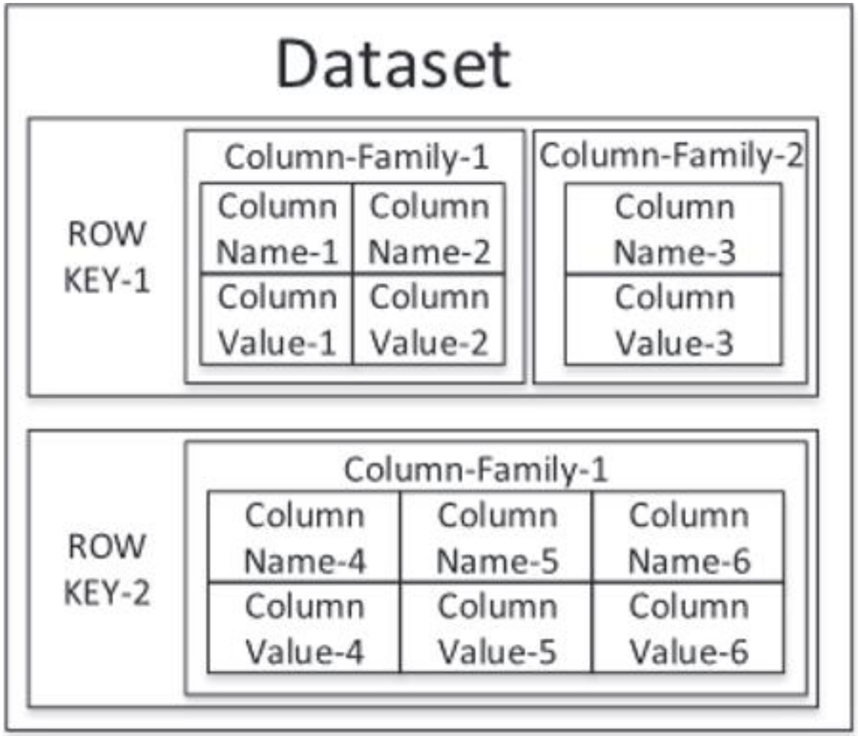
HBase in Facebook[HBaseInFacebook]

- Facebook applications that use HBase with HBase enhancements performed internally in Facebook
  - Facebook Messaging - High write throughput
  - Facebook Insights – Real-time analytics
  - Facebook Metric System - Fast reads of Rrecent data and table scans

- Others: Adobe, StumbleUpon, Twitter, and groups at Yahoo!

HBase[HBase]

- Terminology overlaps, but misleading:
  - *Most basic unit* Column
    - versions
  - Row
  - Table
  - Cell

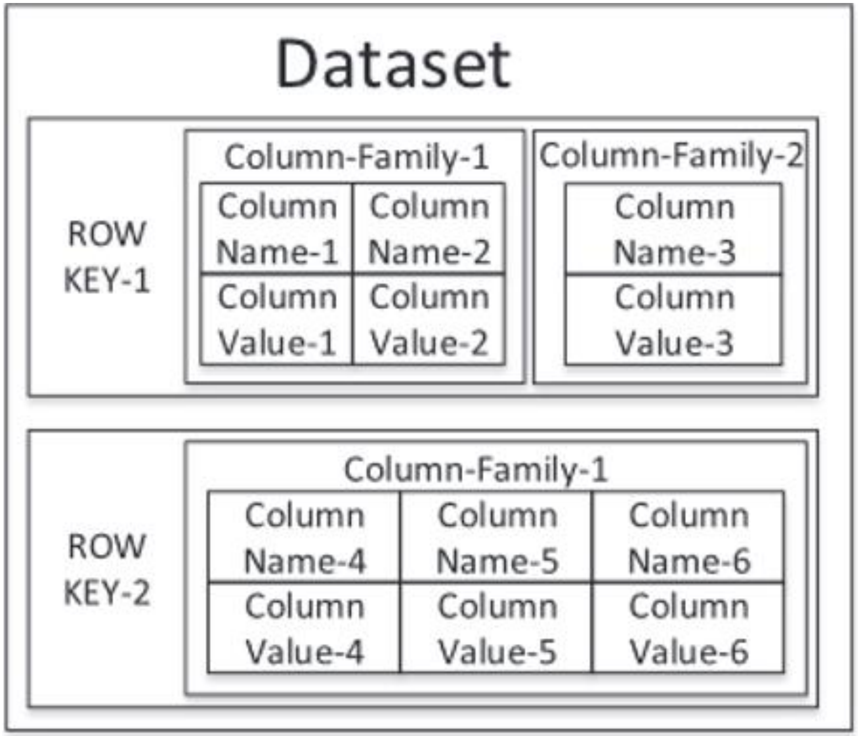


Column-Family Store

HBase[HBase]

- A **table** consists of multiple rows
  - primary key access
- A **row** has a key and column families:
  - Atomic access to row data
  - Sorted lexically:

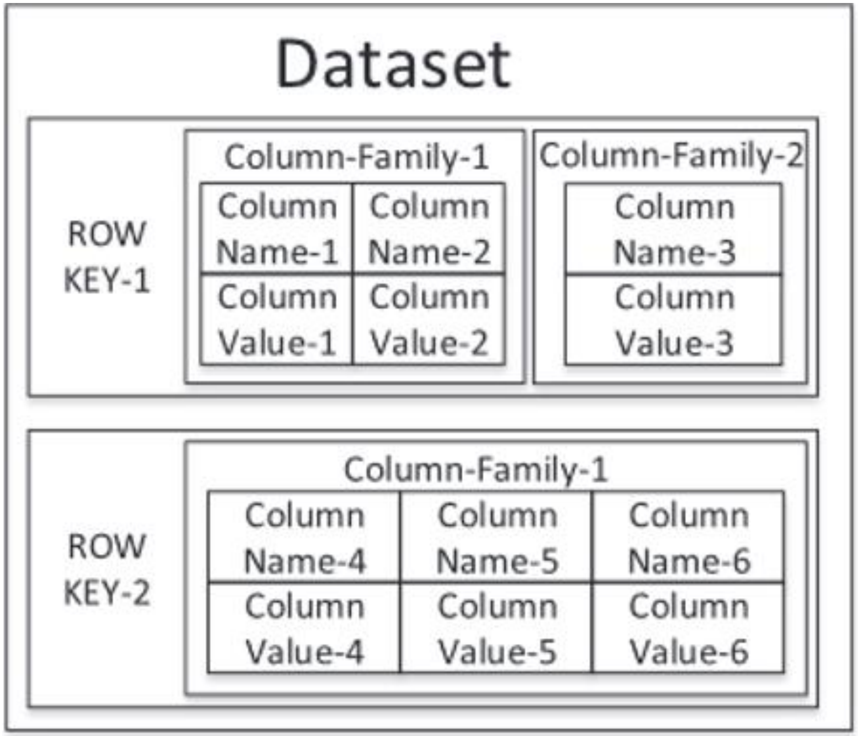
    r1
    r10
    r11
    r2



Column-Family Store

HBase[HBase]

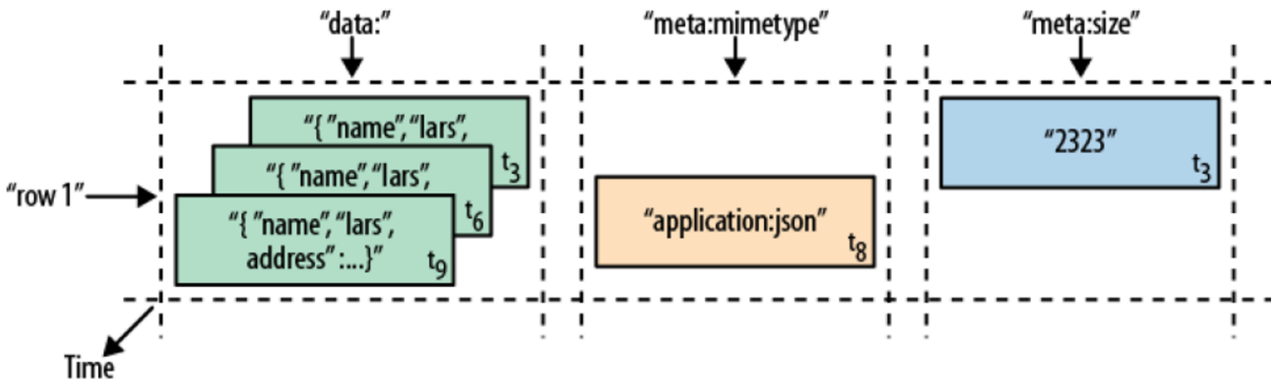
- Columns Families: `content`
- Columns: *family:qualifier* `content:pdf` `content:html`
- All columns in a column family stored together in HFile



Column-Family Store

HBase – Cell [HBase]

- Cell contains value and timestamp
  - `(Table, RowKey, Family, Column, Timestamp) → Value`



Row Key	Time Stamp	Column "data:"	Column "meta:" "mimetype"	"size"	Column "counters:" "updates"
"row1"	t_3	"{ "name" : "lars", "address" : ...}"		"2323"	"1"
	t_6	"{ "name" : "lars", "address" : ...}"			"2"
	t_8		"application/json"		
	t_9	"{ "name" : "lars", "address" : ...}"			"3"

HBase[HBase]

- Canonical example – *webtable*

Row key	**Timestamp**	**Family `content`**		**Family `outgoing links`**		**Family `inbound links`**	
		html	**png**	**cnnsi.com**	**my.look.ca**	**news.bbc.com**	**theguardian.com**
com.cnn.europe	t9			CNN	cnn.com	cnn.com	cnn.com
	t8		logo.png				
	t6	contents:html = "<html>..."	logo1.png				
	t5	contents:html = "<html>..."					
com.cnn.asia	t8	contents:html = "<html>..."					

LINKÖPING UNIVERSITY

HBase - Summary

- Column-oriented data store
  - Hosts very large sparse tables on commodity hardware
  - Column values are timestamped
- Low-latency real-time random access on HDFS!
  - blog.cloudera.com/blog/2012/06/hbase-io-hfile-input-output/
- Row are sorted & stored lexicographically
  - Atomic access to row data
    - But no transactional features across multiple rows
  - No real indexes & high write throughput
- Canonical application - webtable

Big Data Analytics Stack

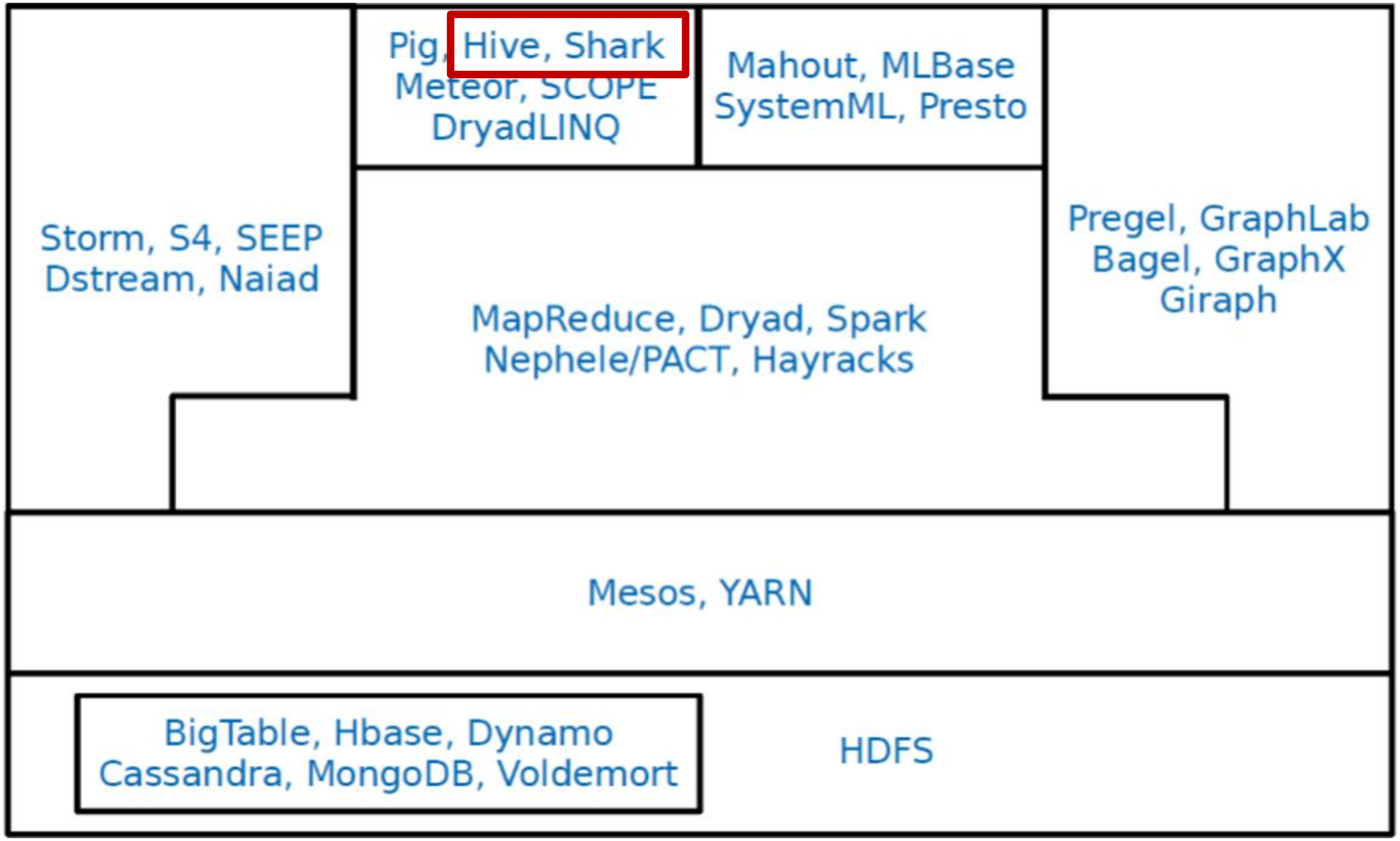


figure from: https://www.sics.se/~amir/dic.htm

Hive[Hive]

Motivation

- MapReduce programming model is low level
- Hadoop/Spark lacks expressiveness
  - end users need to write code even for simplest aggregations, hard to maintain and reuse
- Many experienced SQL developers
- Business intelligence tools already provide SQL interfaces

Hive[Hive]

- Scalable data warehouse
- Built on top of Hadoop
  - translates a query into MapReduce tasks
  - Intermediate results materialized on HDFS
- HiveQL - SQL-like declarative language + UDFs
- Data analytics at Facebook
- Open source since August 2008

DBMS applications – OLTP vs OLAP

Table: Cart

order	Item	quantity
1	45	1
1	55	1
1	65	2
2	65	1

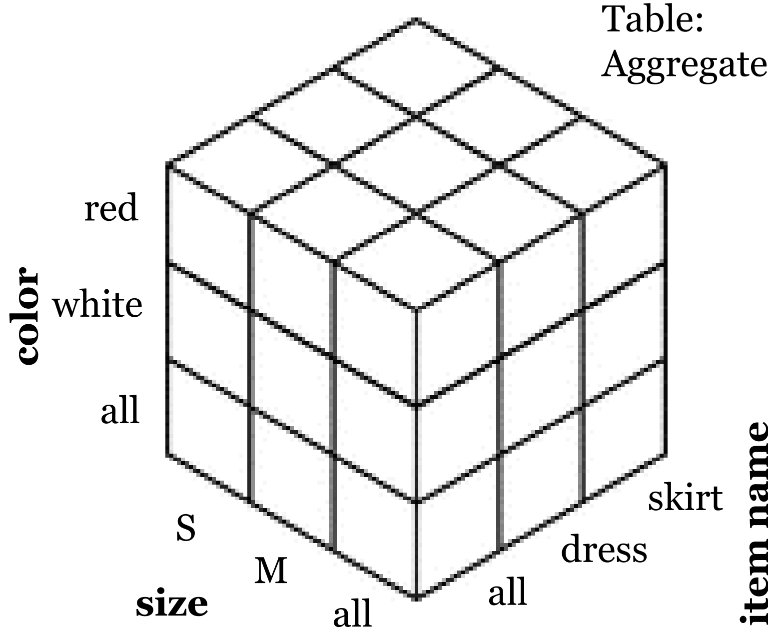
Table: Order

order	customer
1	22
2	33

Table: Item

item	name	color	size
45	skirt	white	L
65	dress	red	M

Table:
Aggregated Sales



Hive[Hive + Hadoop]

- Tables, columns, rows, partitions
  - SerDe to read/write table rows in custom format
- Types
  - Primitive & complex – maps, arrays, arbitrarily nested
  - User-defined types
- Schema-on-read *not* schema-on-write
- Updates, Locks, Indexes

Hive – Tables[Hive + Hadoop]

- The data typically is stored in HDFS
  - Tables stored in directories in HDFS

```
CREATE TABLE managed_table (dummy STRING);

LOAD DATA INPATH '/user/tom/data.txt' INTO table
table_name;

DROP TABLE managed_table;
```

- `CREATE TABLE + LOAD DATA` ***move*** the data
- `DROP TABLE` the data and metadata are deleted, the ***data no longer exists***

Hive – External Tables[Hive + Hadoop]

- The data typically is stored in HDFS
  - External tables – when using other tools on the same dataset

```
CREATE EXTERNAL TABLE external_table (dummy STRING)
LOCATION '/user/tom/external_table';
```

- `CREATE` ***`EXTERNAL`*** `TABLE` ***does not move*** the data
- `DROP TABLE` the metadata only are deleted, the ***data continue to exist***

Hive – Partitions and Buckets[Hive + Hadoop]

- The data typically is stored in HDFS
  - Tables stored in directories in HDFS
    - External tables
  - Partitions by a partition column
    - `CREATE TABLE test_part(ds string, hr int) PARTITIONED BY (ds string, hr int)`
    - `SELECT * FROM test_part WHERE ds='2009-02-02' AND hr=11;`
  - Buckets gives extra structure; more efficient queries

Hive – Tables, Partitions and Buckets[Hive + Hadoop]

- Tables stored in directories in HDFS

  hdfs://user/hive/warehouse/table_name

- Partitions are subdirectories

  hdfs://user/hive/warehouse/table_name/partition_name

- Buckets are stored in files

  hdfs://user/hive/warehouse/table_name/bucket_name

  hdfs://user/hive/warehouse/table_name/partition_name/bucket_name

HiveQL vs SQL[Hadoop]

Feature	HiveQL	SQL
Updates	`UPDATE, INSERT, DELETE`	`UPDATE, INSERT, DELETE`
Transactions	Limited support	Supported
Indexes	Supported	Supported
Data types	SQL supported + boolean, array, map, struct	Integral, floating point, fixed point, text and binary strings, temporal
Functions	Hundreds of built-in functions	Hundreds of built-in functions
Multiple inserts	Supported	Not supported
`CREATE TABLE AS SELECT`	Supported	Not valid SQL-92, but found in some databases
`SELECT`	SQL-92. `SORT BY` for partial ordering. `LIMIT` to limit number of rows returned.	SQL-92
Joins	SQL-92 or variants (join tables in the FROM clause, join condition in the WHERE clause)	Inner joins, outer joins, semi joins, map joins, cross joins
Subqueries	In the FROM, WHERE, or HAVING clause (uncorrelated queries not supported)	In any clause. Correlated or noncorrelated.
Views	Read-only. Materialized views not supported.	Updatable. Materialized or nonmaterialized.
Extension points	User-defined functions. Map-Reduce scripts.	User-defined functions. Stored procedures.

HiveQL vs SQL[Hive]

- Change the order of the FROM and SELECT/MAP/REDUCE
- Multi inserts

```
FROM t1
INSERT OVERWRITE TABLE t2
   SELECT t3.c2, count(1) FROM t3 WHERE t3.c1 <= 20
   GROUP BY t3.c2
INSERT OVERWRITE DIRECTORY '/output_dir'
   SELECT t3.c2, avg(t3.c1) FROM t3
   WHERE t3.c1 > 20 AND t3.c1 <= 30
   GROUP BY t3.c2
```

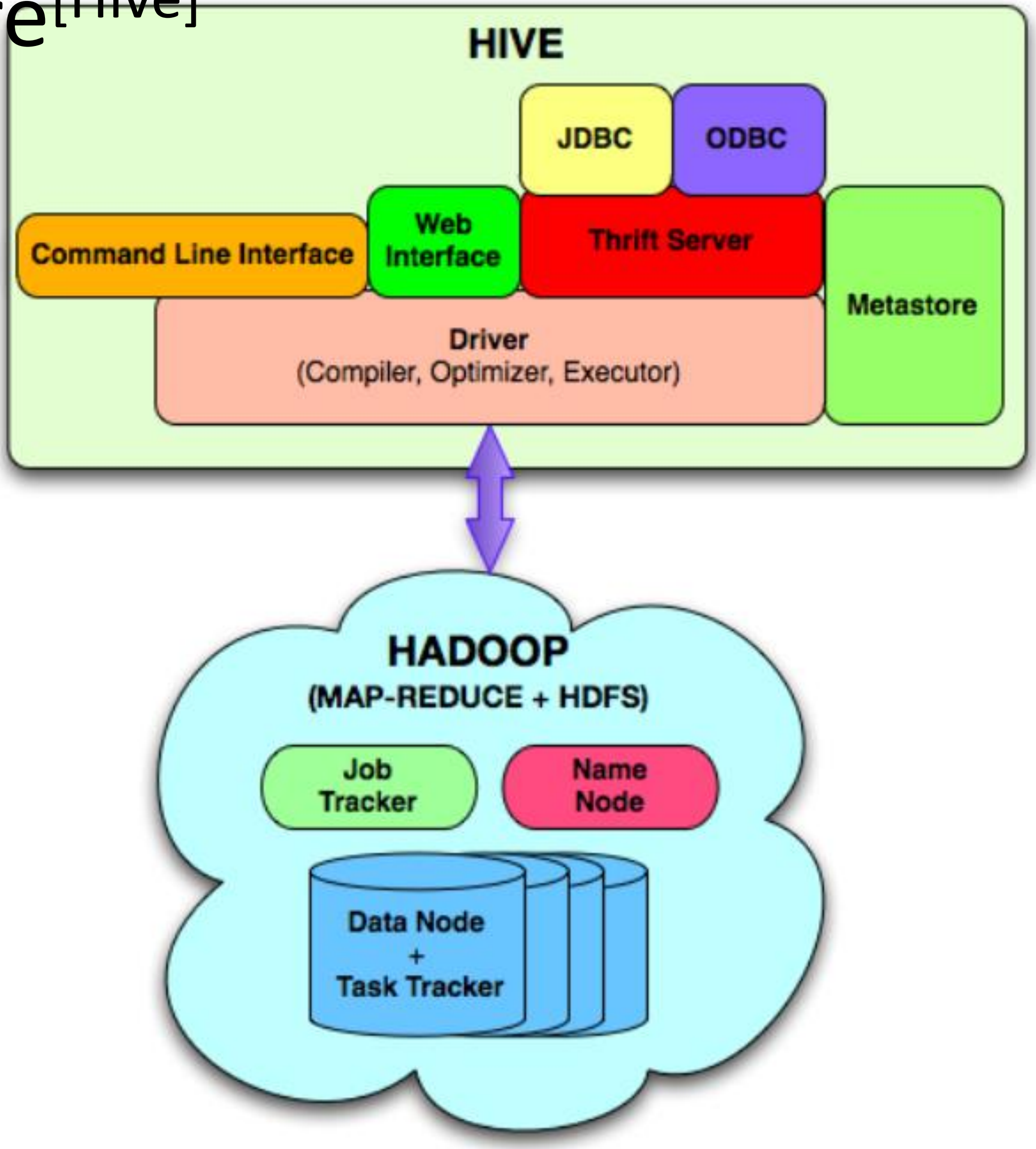
HiveQL vs SQL[Hive]

- Word Count in Hive using custom user program

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

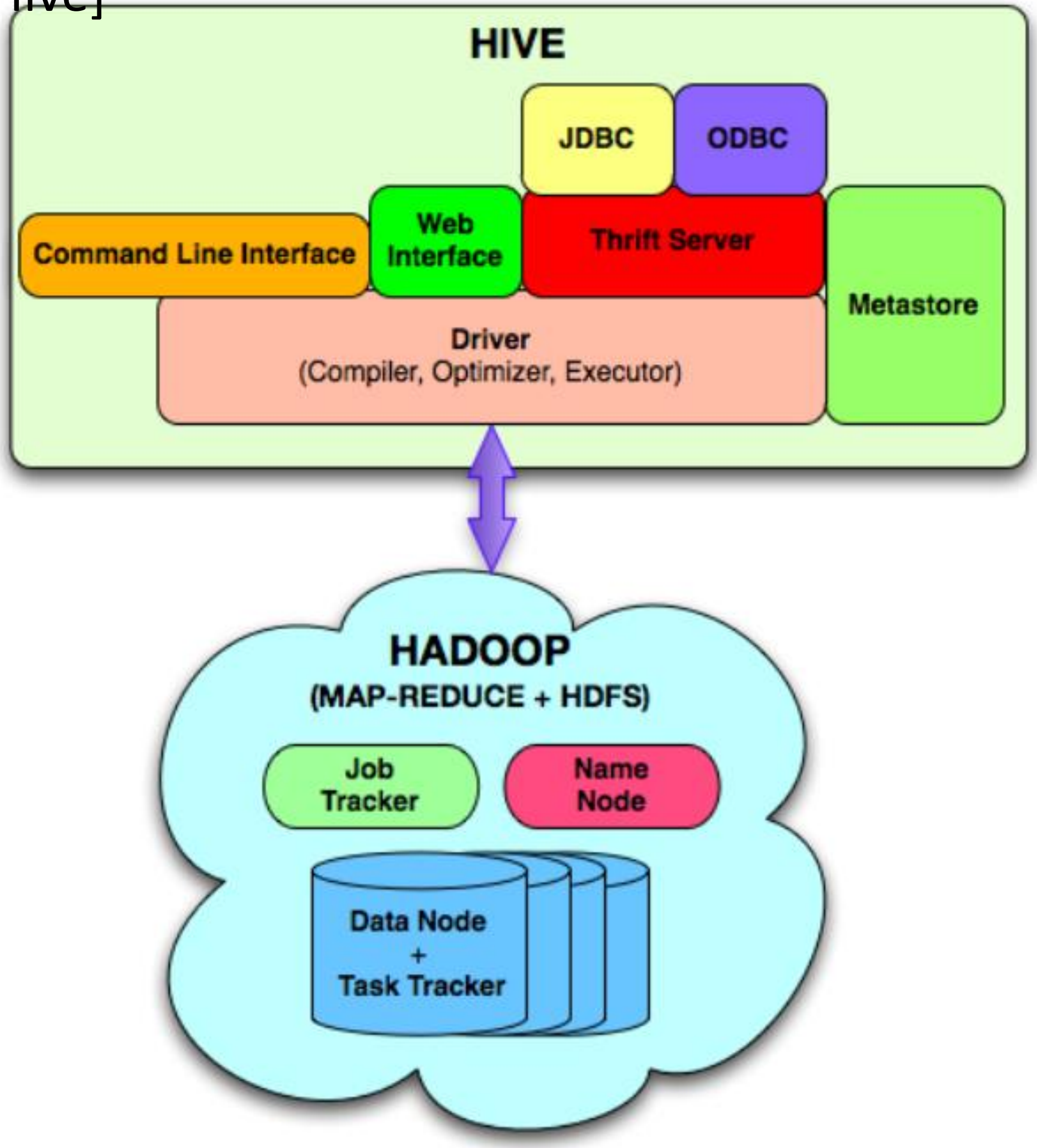
Hive – Architecture[Hive]

- Metastore
  - Served by RDBMS
  - Metadata about the tables
  - Specified at table creation time and reused when the table is referenced



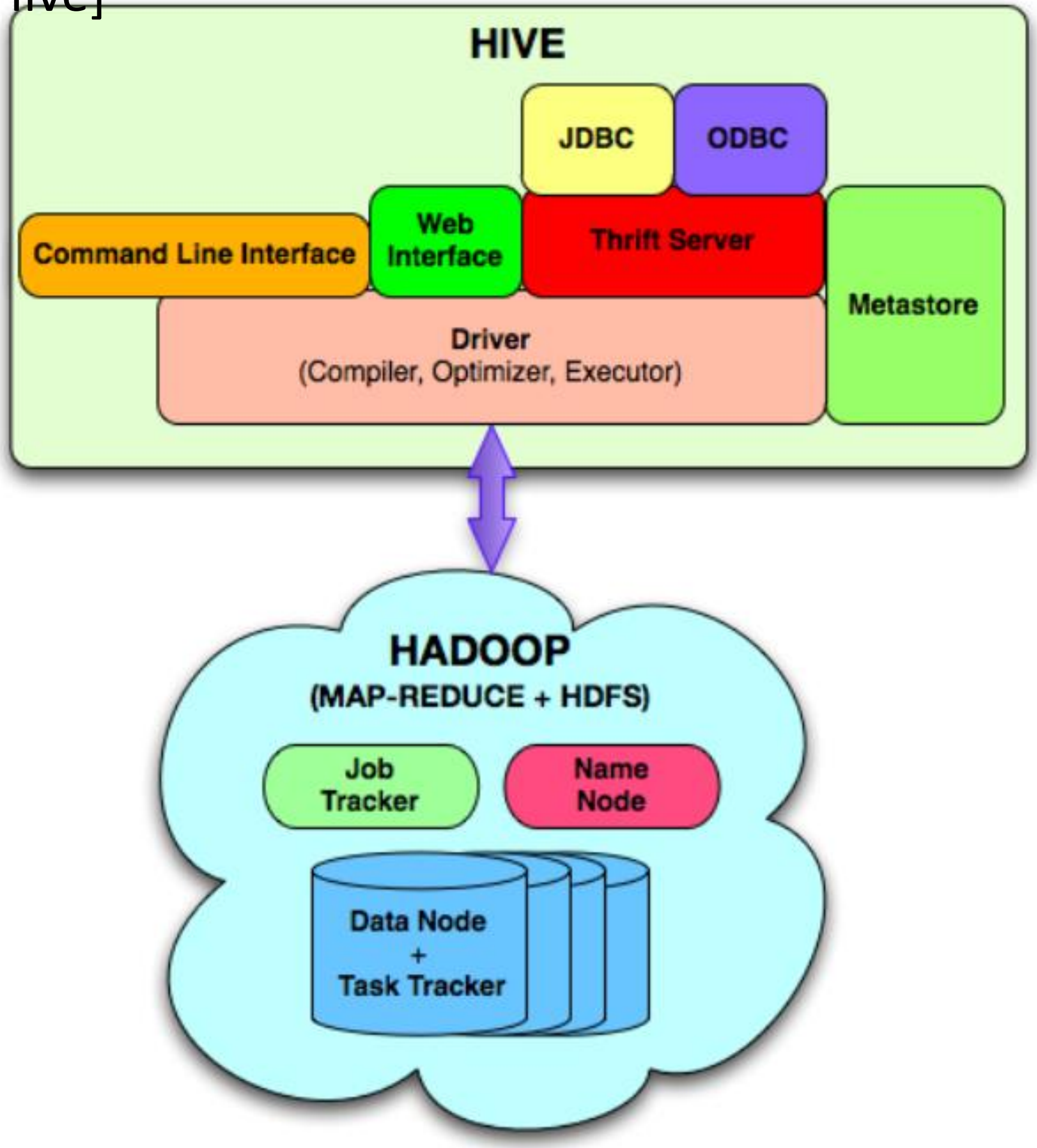
Hive – Architecture[Hive]

- Driver manages the lifecycle of a HiveQL statement:
  - Query Compiler and Optimizer – creates a logical plan from HiveQL query
  - Execution Engine - executes the plan preserving dependencies



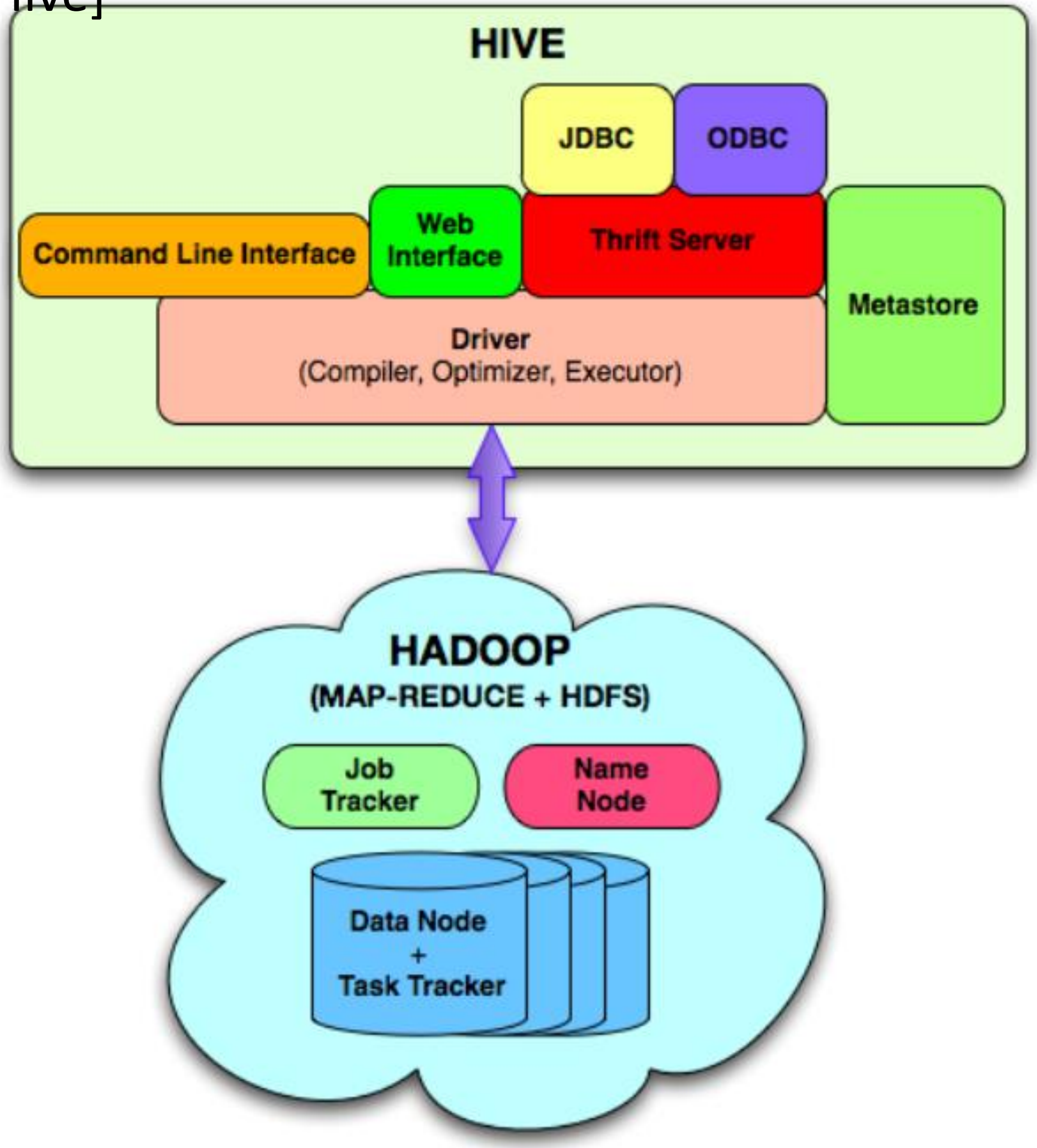
Hive – Architecture[Hive]

- Query Compiler
  - Parsing
  - Logical plan generation
  - Physical plan generation



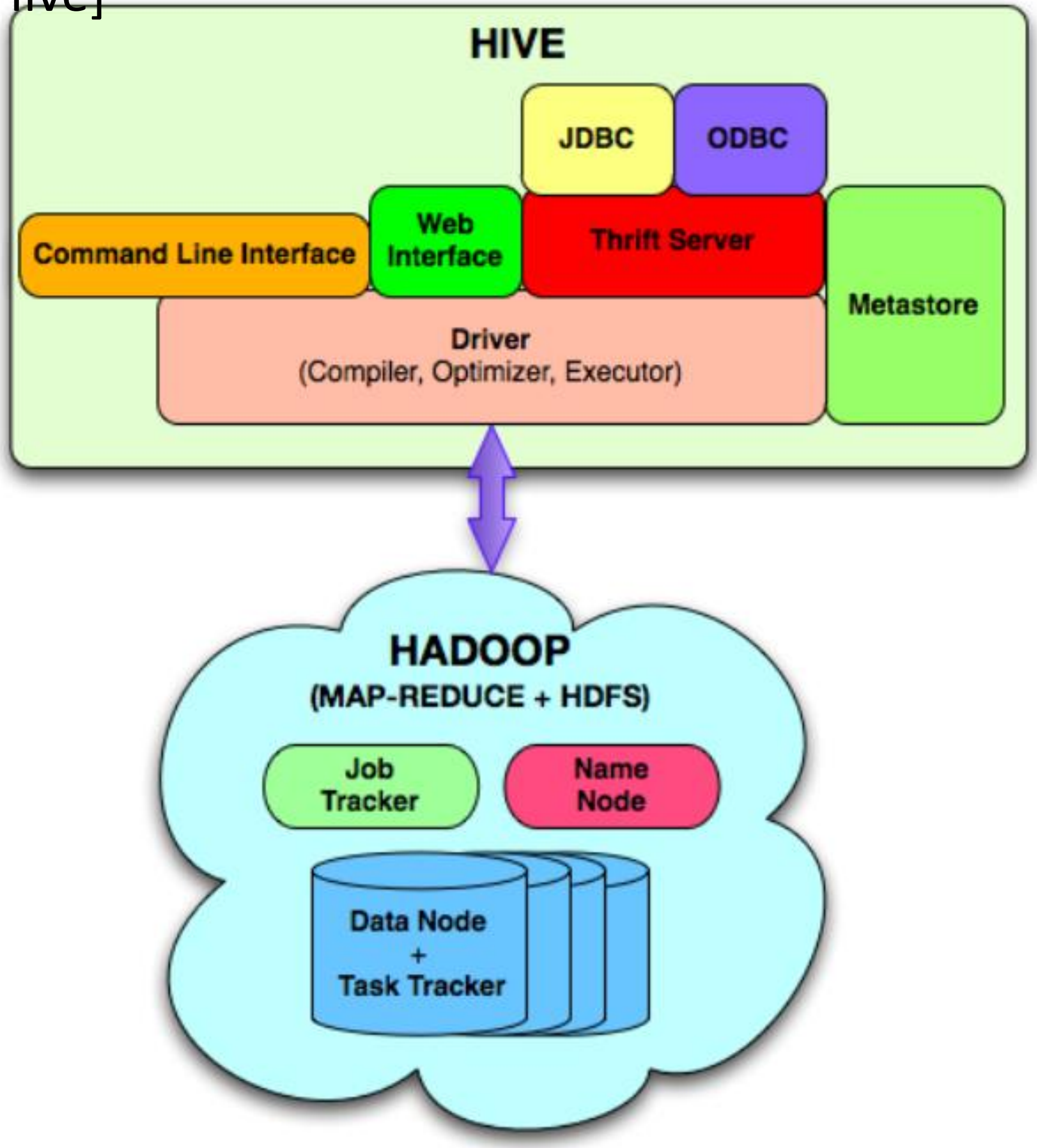
Hive – Architecture[Hive]

- Query Optimizer
  - Column pruning
  - Predicate pushdown
  - Partition pruning
  - Map side joins →no reducer
  - Join reordering - larger tables are streamed



Hive – Architecture[Hive]

- Hive server – enables access from clients written in different languages
- Hive clients
  – CLI
  – JDBC/ODBC
  – webUI



Hive - Summary

- Data warehouse translates SQL queries to MapReduce jobs
- HiveQL is SQL-like language with additional features
- Schema-on-read → no preprocessing
- Table partitions and buckets for more efficient queries
- Column-oriented, row-oriented and text file storage formats

SparkSQL[Shark]

Hive, Shark and SparkSQL [SparkSQLHistory]

- Hive
- Shark project started around 2011
  - built on the Hive codebase
  - swaps Hadoop with Spark
- SparkSQL
  - Shark code base hard to optimize and maintain
  - Shark and Hive compatible
    - Hive's SQL dialects, UDF (user-defined functions) & nested data types

Spark vs MapReduce

- Supports a chain of multiple transformations, not just the two-stage MapReduce topology
- Optimized for low latency
- Provides Resilient Distributed Datasets (RDDs)
  - Written in memory, much faster than the network
  - One copy & the lineage graph
  - RDDs can be rebuilt in parallel in case of failure and slow execution
    - Since RDD are immutable
  - Enables mid-query fault tolerance

Shark[Shark]

- Provides unified engine for running efficiently SQL queries and iterative machine learning algorithms
- In-memory computations
- Benefits from In-memory Resilient Distributed Datasets (RDDs) due to
  - often complex analytic functions are iterative
  - traditional SQL warehouse workloads exhibit strong temporal and spatial locality

Shark – Fault Tolerance[Shark]

- Main-memory databases
  - track fine-grained updates to tables
  - replicate writes across the network
  - expensive on large commodity clusters
- Shark
  - tracks coarse-grained operations, eg, map, join, etc.
  - recovers by tracking the lineage of each dataset and recomputing lost data
  - supports machine learning and graph computations

Shark – Fault Tolerance Properties[Shark]

- Shark can tolerate the loss of any set of worker nodes
  - Also during a query
  - Lost data will be recomputed using the lineage graph
- Lost partitions are rebuilt in parallel
- If a task is slow, it could be run on another node
- Recovery is supported for both SQL and machine learning user defined functions

Shark & Hive

- Query parsing and logical plan generation by the Hive compiler
- Physical plan generation – consists of RDDs transformations

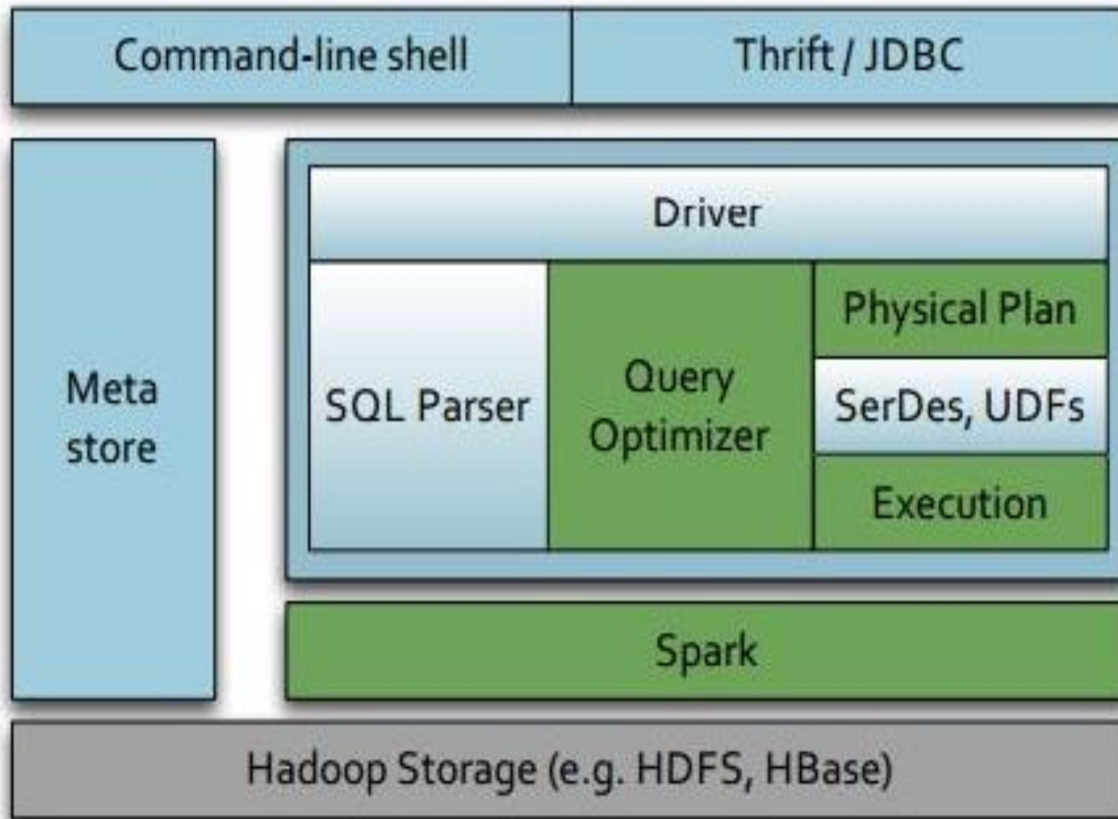


figure from http://www.rosebt.com/blog/spark-shark-and-mesos-data-analytics-stack

Shark – Query Execution[Shark]

- … but how to make it efficient given that:
  - UDF and complex analytic functions
  - Schema-on-read approach, i.e., extract-transform-load (ETL) process has been skipped thus a priory statistics for query optimization are not available

Shark Extensions

- In-memory columnar storage and columnar compression
  – Reduces data size and processing time
- Partial DAG Execution
  – Re-optimize a running query
- Leverage control over data partitioning
  – Allows co-partitioning of frequently joined tables

Shark – Query Execution [Shark]

- In-memory columnar storage – in-memory computation is essential to low-latency query answering
- Shark stores all columns of primitive types as JVM primitive arrays
  - Caching Hive records as JVM objects is inefficient → examples in the paper

Row Storage

13	1000	23
14	2000	27

Column Storage

13	14
1000	2000
23	27

Shark – Query Execution[Shark]

- Partial DAG Execution (PDE)
  - dynamic approach for query optimization
- The query plan is altered based on run-time collected statistics
  - Workers collect global and per partition statistics
  - Workers send them to the master
  - The master dynamically alters the query plan

Shark – Summary

- Data warehouse based on Hive
  - the latest version called SparkSQL
- Efficiently execute complex analytical queries and machine learning algorithms
- Extends Spark execution engine and uses RDDs
- Fault tolerance by tracking the lineage of the RDDs and recomputing in case of failure
  - does not rely on replication
- Tutorials: http://spark.apache.org/docs/latest/sql-programming-guide.html

References

- A comparison between several NoSQL databases with comments and notes by Bogdan George Tudorica, Cristian Bucur
- nosql-databases.org
- Scalable SQL and NoSQL data stores by Rick Cattel
- [Brewer] Towards Robust Distributed Systems @ACM PODC'2000
- [12 years later] CAP Twelve Years Later: How the "Rules" Have Changed, Eric A. Brewer, @Computer Magazine 2012. https://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed
- [Fox et al.] Cluster-Based Scalable Network Services @SOSP'1997
- [Karger et al.] Consistent Hashing and Random Trees @ACM STOC'1997
- [Coulouris et al.] Distributed Systems: Concepts and Design, Chapter: Time & Global States, 5th Edition
- [DataMan] Data Management in cloud environments: NoSQL and NewSQL data stores.

References

- NoSQL Databases - Christof Strauch – University of Stuttgart
- The Beckman Report on Database Research
- [Vogels] Eventually Consistent by Werner Vogels, doi:10.1145/1435417.1435432
- [Hadoop] Hadoop The Definitive Guide, Tom White, 2011
- [Hive] Hive - a petabyte scale data warehouse using Hadoop
- https://github.com/Prokopp/the-free-hive-book
- [Massive] Mining of Massive Datasets
- [HiveManual] https://cwiki.apache.org/confluence/display/Hive/LanguageManual
- [Shark] Shark: SQL and Rich Analytics at Scale
- [SparkSQLHistory] https://databricks.com/blog/2014/07/01/shark-spark-sql-hive-on-spark-and-the-future-of-sql-on-spark.html

References

- [HDFS] The Hadoop Distributed File System
- [Dynamo] Dynamo: Amazon's Highly Available Key-value Store, 2007
- [HBaseInFacebook] Apache hadoop goes realtime at Facebook
- [HBase] HBase The Definitive Guide, 2011
- [HDFSpaper] The Hadoop Distributed File System @MSST2010