

PA220: Database systems for data analytics

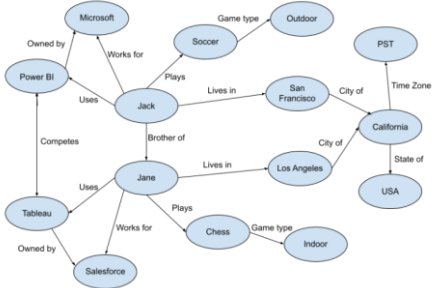
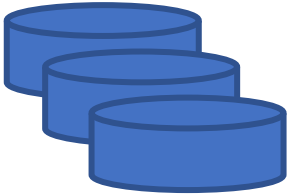
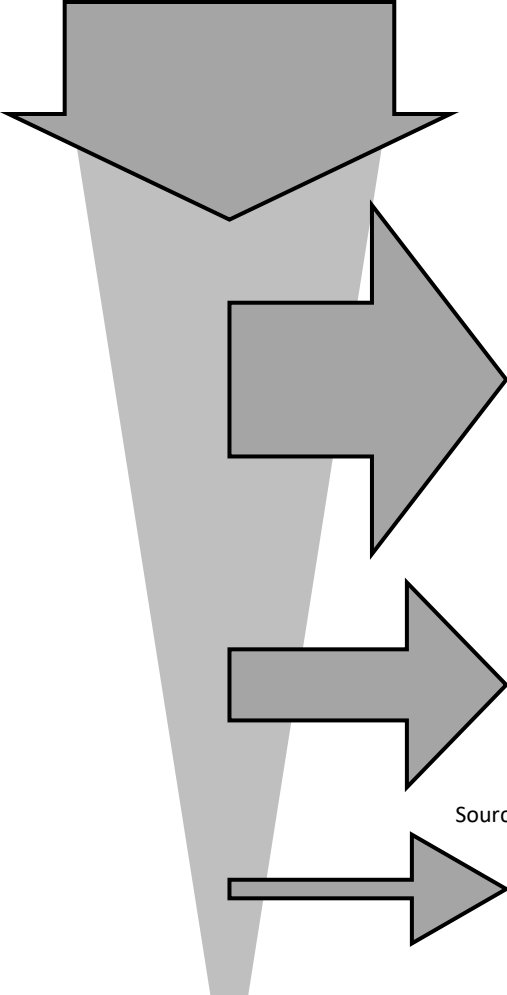
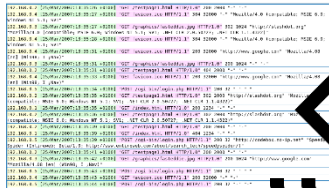
# Big Data Analytics

# Contents

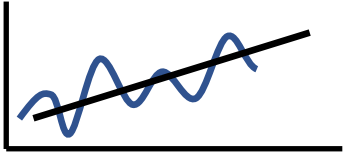
- properties of current data
- architecture of data processing and analytics systems
- challenges in Big Data processing
- distributed data warehouse

# Motivation

- Data production
  - Information systems
  - Monitoring services
  - Sensors, GPS tracking
  - Social networks
- Data processing
  - Storage & archiving
  - Summarization
  - Reporting
  - Visualization
  - Insights
  - Predictions



Source: <https://www.biconnector.com/blog/graph-analytics-for-big-data-complete-guide/>



# Data Production

<b>Data Set/Domain</b>	<b>Description</b>
Social Media	12+ Terabytes ( $10^{12}$ ) of tweets every day and growing.
British Library UK Website Crawler	~110 Terabytes ( $10^{12}$ ) per domain crawl to be delivered
LHC - Large Hadron Collider (CERN)	13-15 Petabytes ( $10^{15}$ ) in 2010
Internet Communications (Cisco)	667 Exabytes ( $10^{18}$ ) in 2013
Digital Universe	7.9 Zettabytes ( $10^{21}$ ) (2015)

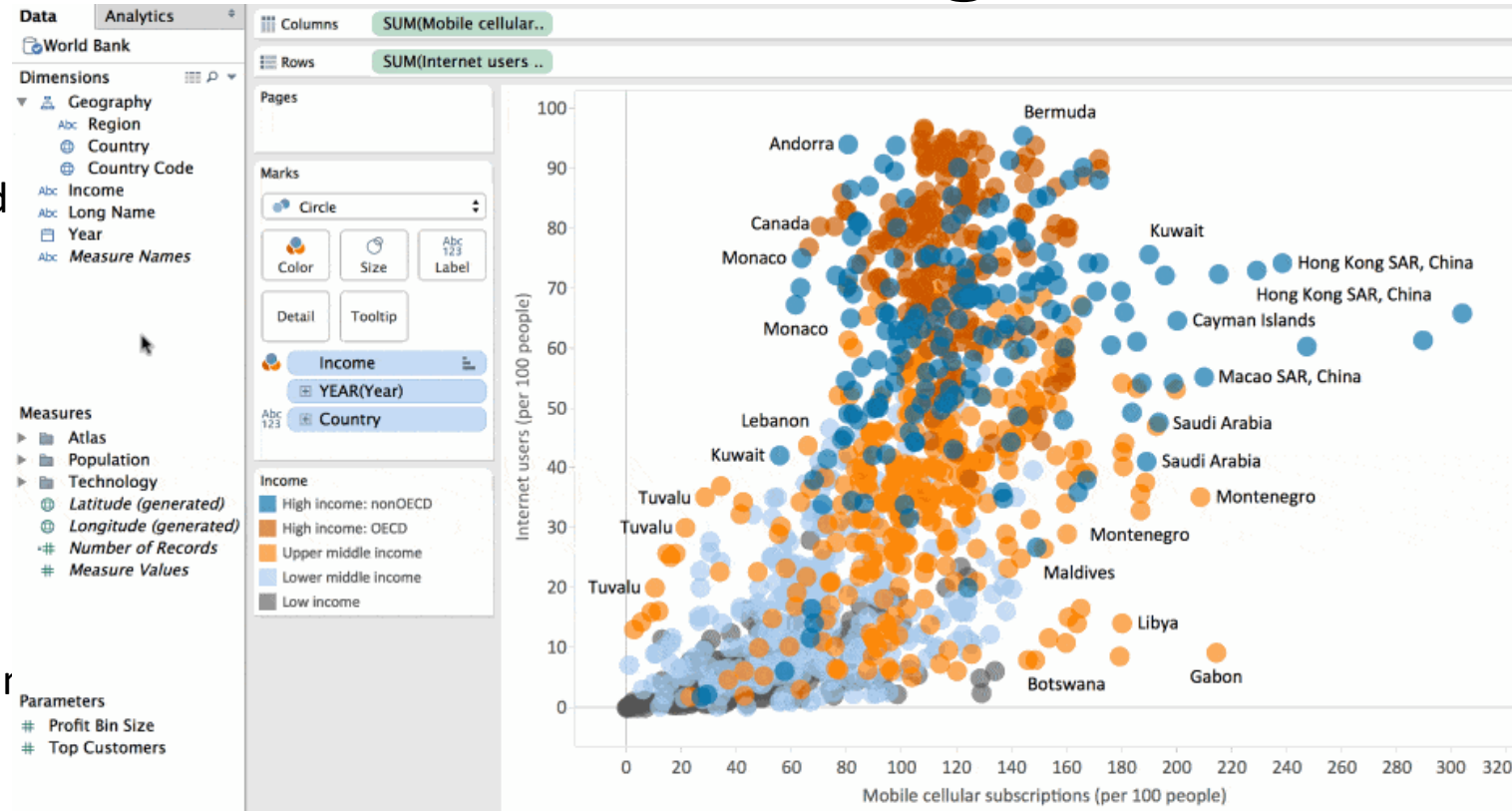
Source:

S. Kaisler, F. Armour, J. A. Espinosa, and W. Money, "Big Data: Issues and Challenges Moving Forward," 46th Hawaii Int. Conf. Syst. Sci., pp. 995–1004, 2013.

# Nature of Current Data and Processing

## Big Data

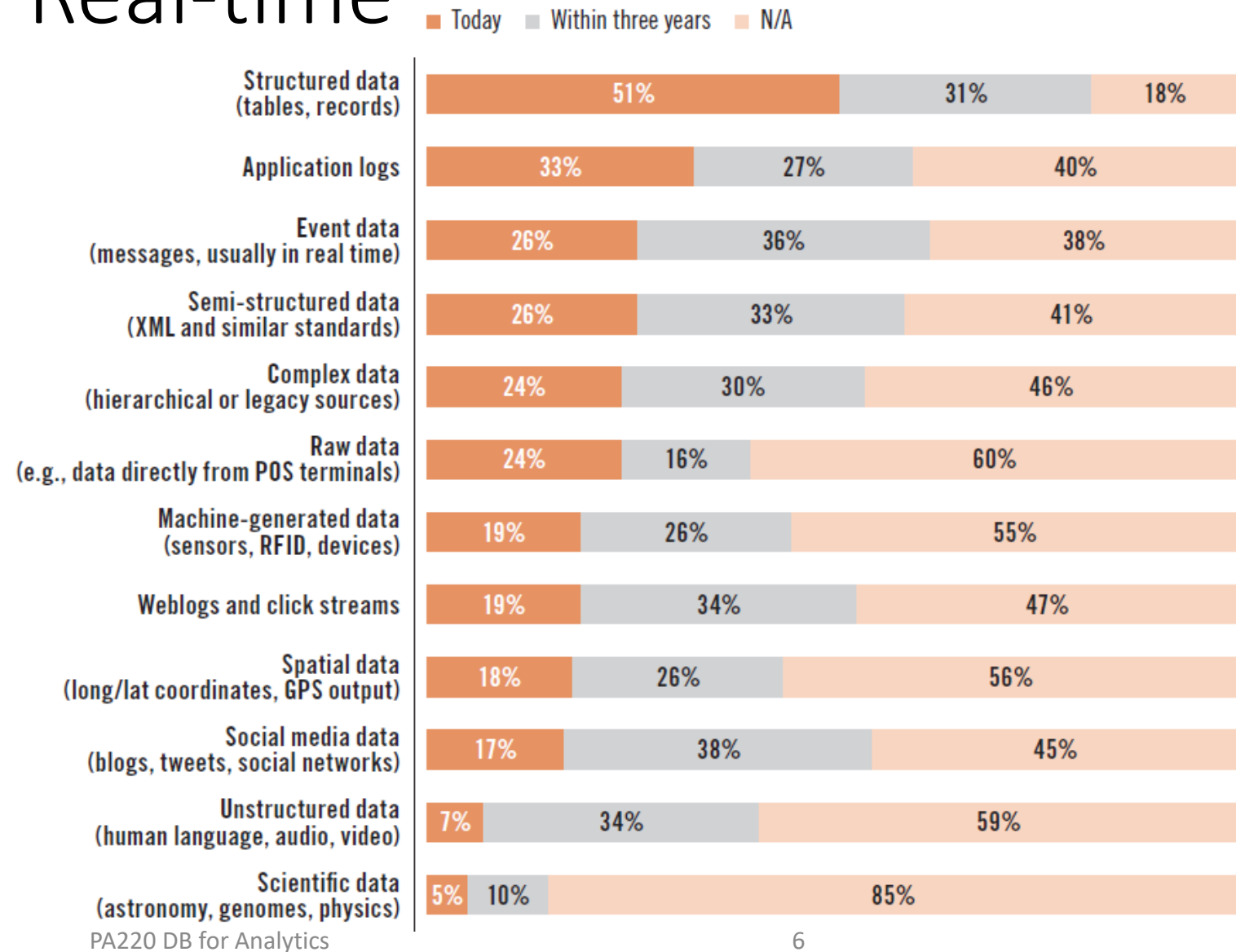
- Volume
  - the amount of data increases tenfold every five years
- Variety
  - varying data structure, text, multimedia, ...
- Velocity
  - continuous data flow from sensors, social networks, ...
- Veracity
  - with different data sources, it is getting more difficult to maintain data certainty
- Value
  - ability to transform data into business
- Real-time processing



Source: tableau.com

# Data Processed in Real-time

- TDWI report, Q4 2014
  - 105 companies over 500 emp.



# Necessities for Big Data Analytics

- infrastructure for big data
  - processing
    - batch
    - iterative
    - stream (real-time)
  - storage
    - key-value stores
    - column stores
- algorithms for big data
  - data integration
  - data reporting
  - analytic functions
  - machine learning

# Computational & Storage Opportunities

- horizontal scaling instead of vertical scaling
- new platforms
  - HDFS & MapReduce (e.g., Hadoop)
  - distributed stream processing (e.g., Storm)
  - column storage (e.g., Vertica)
  - NoSQL platforms (e.g., HBase)
  - in-memory DBMSs (e.g., VoltDB)



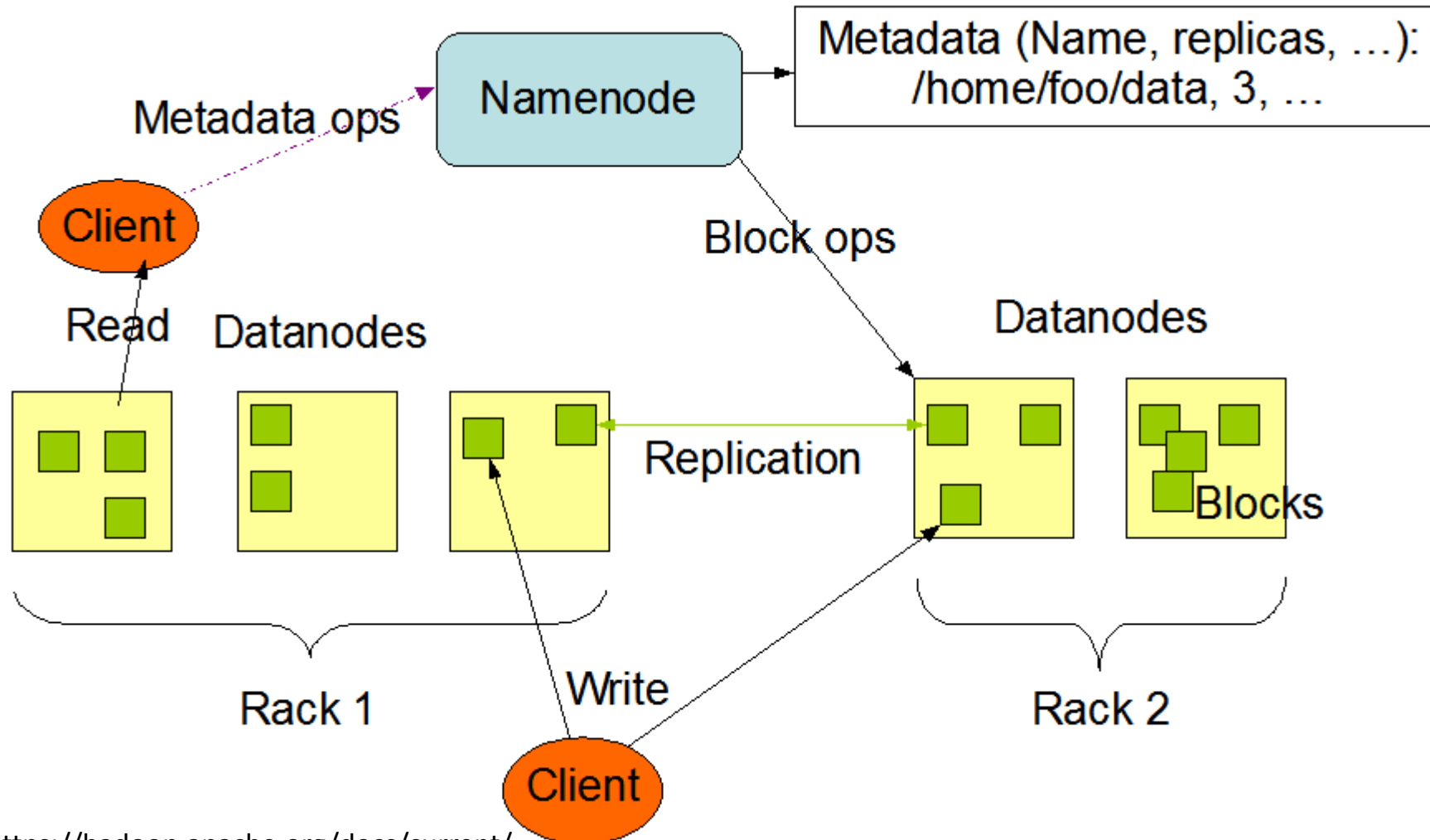
# Hadoop Platform

- SW library for distributed processing of large data sets
  - across clusters of computers
- high-availability achieved on application layer by replication
  - tasks run / data stored on unreliable HW
- HDFS – distributed high-throughput file system
  - designed for mostly immutable files
  - concurrent write not supported
  - cooperation with MapReduce – data & computation locality
- MapReduce – programming model for large scale data processing
  - Map() – filtering and sorting, outputs “key,value” pairs Map(k1,v1) → list(k2,v2)
  - Reduce() – summarizing Map() results by their keys Reduce(k2, list (v2)) → list(k3,v3)

# HDFS

- Files are divided into blocks (chunks), typically 64 MB
  - The chunks are replicated at three different machines
    - ...in an “intelligent” fashion, e.g., never all on the same computer rack
  - The block size and replication factor are tunable per file.
- One machine is a **name node** (master)
- The others are **data nodes** (chunk servers)
  - The master keeps track of all file metadata
    - mappings from files to chunks and locations of the chunks on data nodes
  - To find a file chunk, the **client** queries the master, and then it contacts the relevant data nodes.
  - The master’s metadata files are also replicated.
- Files in HDFS are write-once (except for appends and truncates)
  - and have strictly one writer at any time.

# HDFS Architecture



Source: <https://hadoop.apache.org/docs/current/>

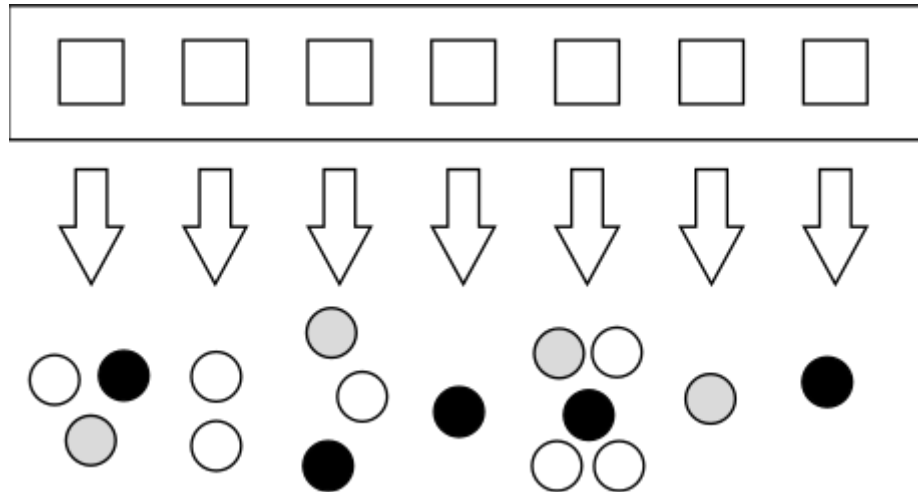
# Distributed Computation Platforms

- batch processing -> MapReduce, Spark, ...
- stream processing -> Storm, Spark Streaming, ...
  
- MapReduce
  - a programming model for distributed data processing
  - cooperates with a distributed file system
  
  - A distributed computational task has three phases:
    - The map phase: data transformation
    - The grouping phase done automatically by the MapReduce Framework
    - The reduce phase: data aggregation
  - The user defines only map & reduce functions.

# MapReduce – Map Function

- **Map function** simplifies the problem in this way:
  - Input: a **single data item** (e.g., line of text) from a data file
  - Output: zero or more **(key, value) pairs**
- The **keys** are **not typical** “primary keys”:
  - They do **not** have to be **unique**
  - A map task can produce **several key-value pairs** with the same key (even from a single input)
- **Map phase** applies the map function to **all** items

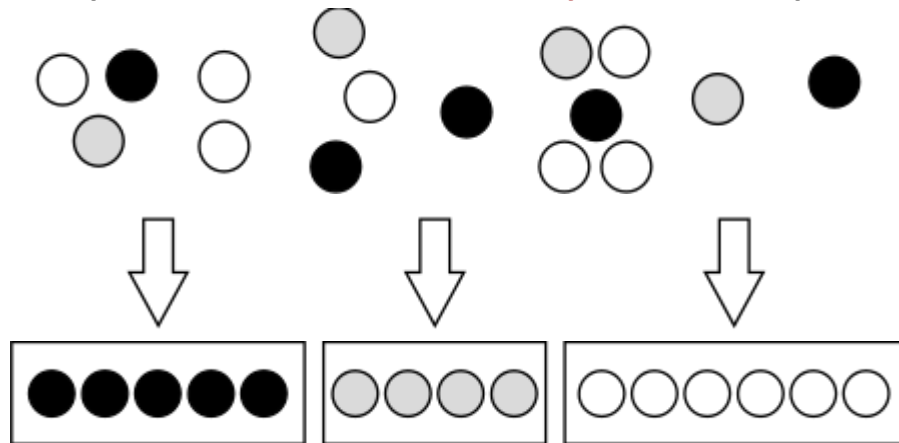
# MapReduce – Map Function



- input data
- map function
- output data  
(color indicates the key value)

# MapReduce – Grouping Phase

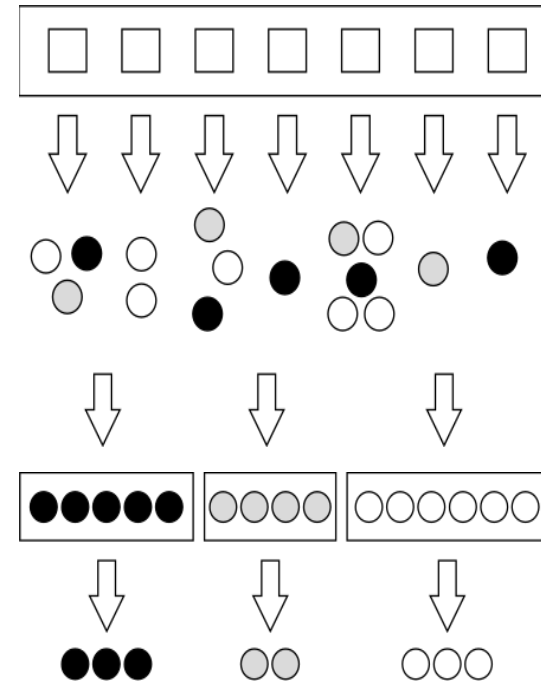
- **Grouping** (Shuffling): The key-value **outputs from the map** phase are grouped by key
  - Values sharing **the same key** are sent to the same reducer
  - These values are **consolidated** into a single list (**key, list**)
    - This is convenient for the reduce function
- This phase is **realized by** the MapReduce **framework**



- intermediate output  
(color indicates the key value)
- grouping phase – shuffle function

# MapReduce – Reduce Function

- Reduce: **combine** the values for each key
  - to achieve the **final result(s)** of the computational task
- **Input:** (key, value-list)
  - value-list contains all values generated for given key in the Map phase
- **Output:** (key, value-list)
  - zero or more **output records**



- input file
- map function
- output data (color indicates the key value)
- shuffle function
- reduce function
- output records



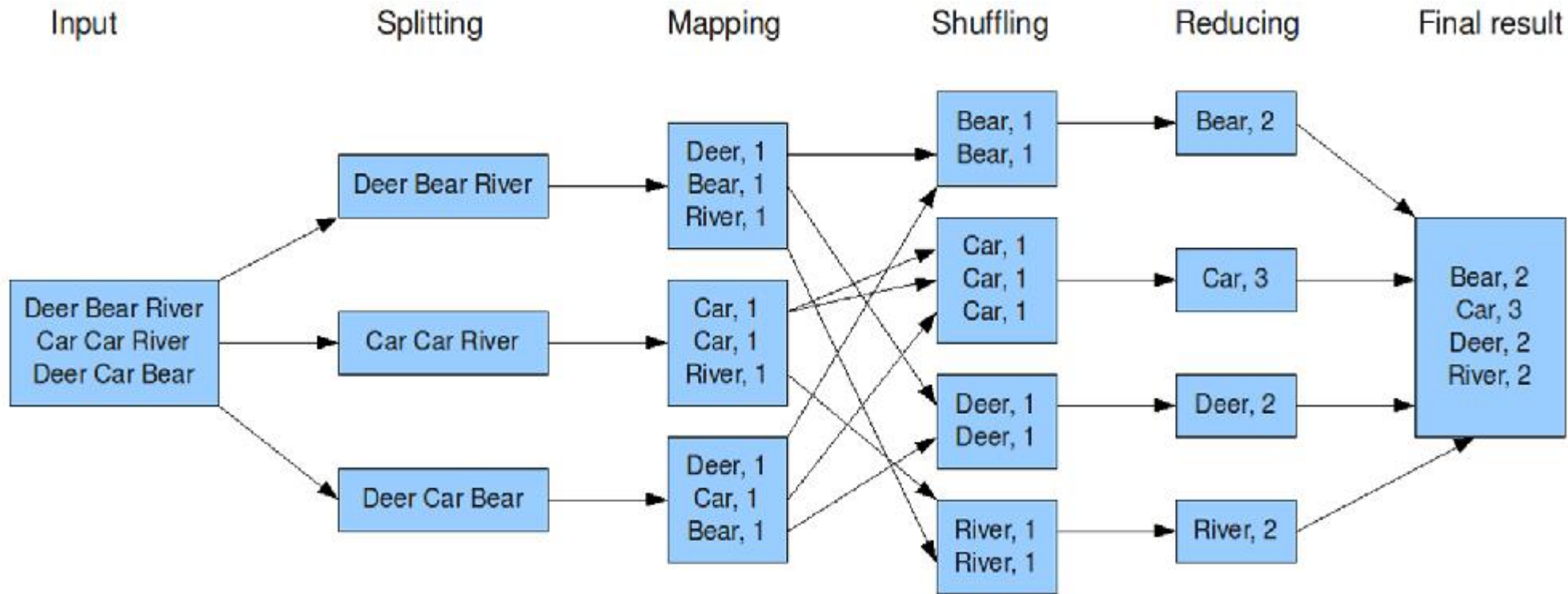
# MapReduce Example: Word Count

- Task: Calculate word frequency in a set of documents

```
map(String key, Text value):  
    // key: document name (ignored)  
    // value: content of document (words)  
foreach word w in value:  
    emitIntermediate(w, 1);
```

```
reduce(String key, Iterator values):  
    // key: a word  
    // values: a list of counts  
int result = 0;  
foreach v in values:  
    result += v;  
emit(key, result);
```

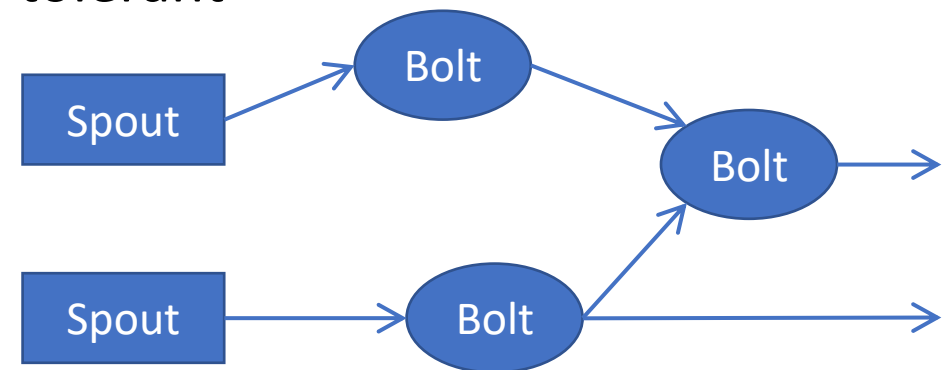
# MapReduce Example: Word Count



Source: <http://www.cs.uml.edu/~jlu1/doc/source/report/MapReduce.html>

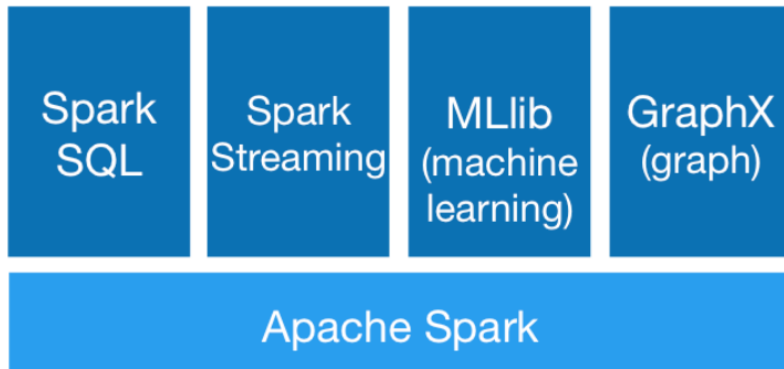
# Distributed Computation Platforms

- batch processing -> MapReduce, Spark, ...
- stream processing -> Storm, Spark Streaming, ...
- Storm
  - real-time computation system, scalable, fault-tolerant
  - algorithm as a directed acyclic graph
    - edges = streams of data tuples
    - spouts = data source
    - bolts = processing node
  - data model = a tuple of named fields
  - mapping to physical workers



# Apache Spark

- a unified analytics engine for large-scale data processing.
- high performance for both batch and streaming data
  - using a state-of-the-art DAG scheduler,
  - a query optimizer, and
  - a physical execution engine.
  - 100x faster than Hadoop



# Distributed Storage Platforms

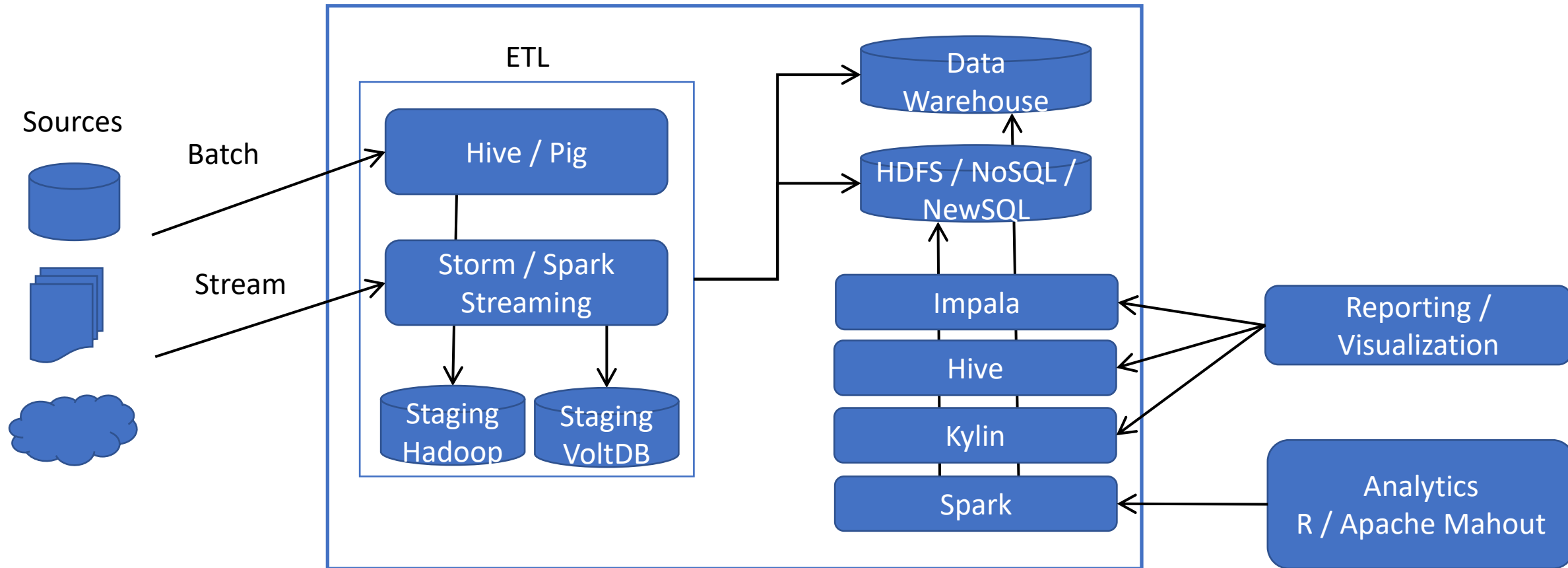
- key-value stores / NoSQL databases (Hbase)
  - structured / tabular data model, but flexible schema
  - horizontal scaling
  - no ACID, no join operation
  - key = identifies a row (typically with timestamp)
  - value = a multidimensional structure
- column stores (C-store)
  - relational data model, values of a column stored continuously
  - read-optimized, high-query throughput DBMS
  - relaxed consistency on reads

```
"aaaaa" : {  
  "A" : {  
    "foo" : "y",  
    "bar" : "d"  
  },  
  "B" : {  
    "" : "w"  
  }  
}
```

# Distributed Storage Platforms

- real-time databases (e.g., VoltDB (originally H-Store))
  - NewSQL databases
    - scalability of NoSQL, relational data model
    - ACID guarantees
  - row-oriented storage on a distributed shared-nothing cluster
  - main memory db
  - fault-tolerance by node replication

# Data Warehouse for Big Data



# Distributed Data Warehouse

- Hive – data warehouse for large datasets
  - unstructured data in HDFS, structure projected on read
  - manages and queries data using HiveQL
    - converts them to Map-Reduce jobs
  - supports indexing
  - DML operations
    - UPDATE & DELETE at row level
- Kylin – provides OLAP for big data
  - precalculates aggregations – data cubes fully on Hadoop and Spark
  - query engine translation from SQL
    - exploit prepared aggregations
    - low-latency query evaluation (sub-second)
  - integrate with Tableau, Power BI



# Advanced Analytics

- Apache Mahout
  - scalable machine learning library
  - based on Hadoop, Spark
  - aimed at
    - recommendations, collaborative filtering
    - clustering, dimensionality reduction, classification
- Project R
  - platform for statistical computing and visualization
  - integrate to Hadoop

# Advanced Analytics

- data quality is crucial
  - Tamr
    - data unification platform
    - automated integration with machine learning
    - thousands of data sources
- analytic model
  - computed & adjusted off-line
  - deployment
    - in complex analysis
    - in ETL

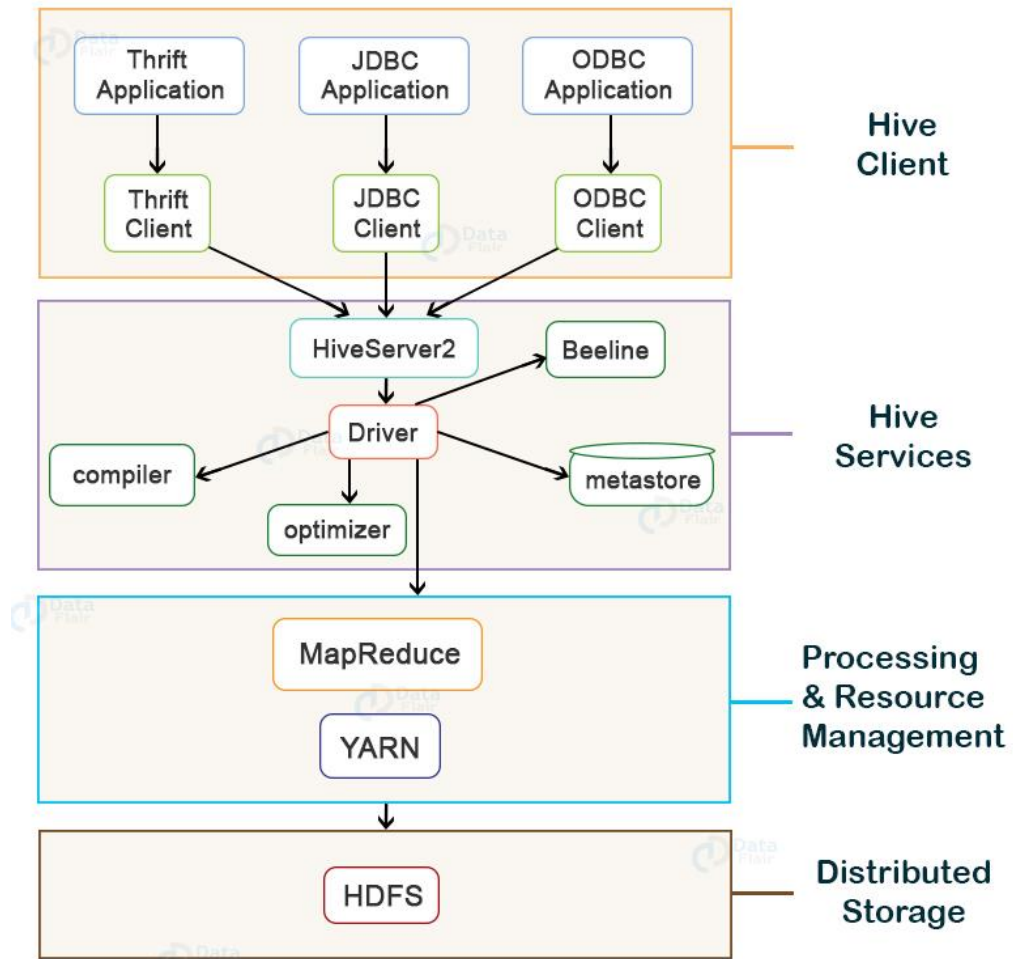
# Advanced Analytics in Real-time

- event processing
  - tracking streams to detect events
  - event = change of state, exceeding a threshold, anomalies, ...
  - deriving conclusions from events
- complex event processing
  - combine multiple sources
  - implement pattern detection, correlation, filtering, aggregation, ...
  - extension to SQL – StreamSQL
    - continuous queries with incremental results
    - windowing & aggregations
    - windowing & joins

# Apache Hive

- A system for querying and managing structured data built on top of Hadoop
  - Uses Map-Reduce for execution
  - HDFS for storage – but any system that implements Hadoop FS API
- Key Building Principles:
  - Structured data with rich data types (structs, lists and maps)
  - Query data directly from different formats (text/binary) and file formats (Flat/Sequence)
  - SQL as a familiar programming tool and for standard analytics
  - Allow embedded scripts for extensibility and for non-standard applications
  - Rich metadata to allow data discovery and for optimization

# Apache Hive – Architecture



Source: <https://data-flair.training/blogs/apache-hive-architecture/>

# Apache Hive – MetaStore

- Stores table/partition properties:
  - Table schema and SerDe library for formatting rows
  - Table location on HDFS
  - Logical partitioning keys and types
  - Partition level metadata
  - Other information

```
» CREATE TABLE mylog (  
    user_id BIGINT,  
    page_url STRING,  
    unix_time INT)  
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';  
  
» CREATE table mylog_rc (  
    user_id BIGINT,  
    page_url STRING,  
    unix_time INT)  
ROW FORMAT SERDE  
    'org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe'  
STORED AS RCFILE;
```

# Apache Hive – Structured Data

- Type system
  - Primitive types (double, float, bigint, int, smallint, tinyint, boolean, string, timestamp)
  - Recursively build up using Composition/Maps/Lists
- ObjectInspector interface for user-defined types
  - To recursively list schema
  - To recursively access fields within a row object
- Generic (De)Serialization Interface (SerDe)
- Serialization families implement interface
  - Thrift DDL based SerDe
  - Delimited text based SerDe
  - You can write your own SerDe (XML, JSON ...)

# Apache Hive – Query Language

- Basic SQL

- From clause subquery
- Equi-join only (also outer)
- Multi-table Insert
- Multi-col group-by
  - Grouping sets, cubes, rollups
- Sampling
- Objects traversal

- Extensibility

- Pluggable Map-reduce scripts using TRANSFORM

```
hive> select * from temperature limit 10;
OK
station NULL NULL NULL NULL flag NULL NULL NULL country name
AQW00061705 1 1 1 26.888888888888893 P -14.3306 -170.7136 3.7 AS PAGO PAGO WSO AP
AQW00061705 1 2 1 26.888888888888893 P -14.3306 -170.7136 3.7 AS PAGO PAGO WSO AP
AQW00061705 1 3 1 26.833333333333332 P -14.3306 -170.7136 3.7 AS PAGO PAGO WSO AP
AQW00061705 1 4 1 26.777777777777782 P -14.3306 -170.7136 3.7 AS PAGO PAGO WSO AP
AQW00061705 1 5 1 26.777777777777782 P -14.3306 -170.7136 3.7 AS PAGO PAGO WSO AP
AQW00061705 1 6 1 NULL -14.3306 -170.7136 3.7 AS PAGO PAGO WSO AP
AQW00061705 1 7 1 NULL -14.3306 -170.7136 3.7 AS PAGO PAGO WSO AP
AQW00061705 1 8 1 NULL -14.3306 -170.7136 3.7 AS PAGO PAGO WSO AP
AQW00061705 1 9 1 NULL -14.3306 -170.7136 3.7 AS PAGO PAGO WSO AP
Time taken: 0.293 seconds, Fetched: 10 row(s)
```

```
hive> describe temperature;
OK
station string
month int
day int
Hour int
temperature double
flag string
latitude double
longitude double
elevation double
country string
name string
Time taken: 0.22 seconds, Fetched: 11 row(s)
```



# Apache Hive – Query Language

- Aggregate queries mapped to MR jobs:

```
hive> select count(*) from temperature;
Query ID = dohnal_20210119140041_1a94796f-172f-4d40-b8c3-10932ea638c3
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
2021-01-19 14:00:41,609 INFO [3093d28d-ea97-4e81-ab63-7c6cdd17fe7d main] client.ConfiguredRMFailoverProxyProvider: Failing over to rm2
Starting Job = job_1605005553005_4269, Tracking URL = https://hador-c1.ics.muni.cz:8090/proxy/application_1605005553005_4269/
Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job_1605005553005_4269
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2021-01-19 14:00:50,401 Stage-1 map = 0%, reduce = 0%
2021-01-19 14:01:01,721 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 6.07 sec
2021-01-19 14:01:13,029 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 8.2 sec
MapReduce Total cumulative CPU time: 8 seconds 200 msec
Ended Job = job_1605005553005_4269
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 8.2 sec HDFS Read: 8456014 HDFS Write: 107 HDFS EC Read: 0 SUCCESS
Total MapReduce CPU Time Spent: 8 seconds 200 msec
OK
4003322
Time taken: 32.929 seconds, Fetched: 1 row(s)
```

# Apache Hive – Query Language

- Custom map/reduce scripts:

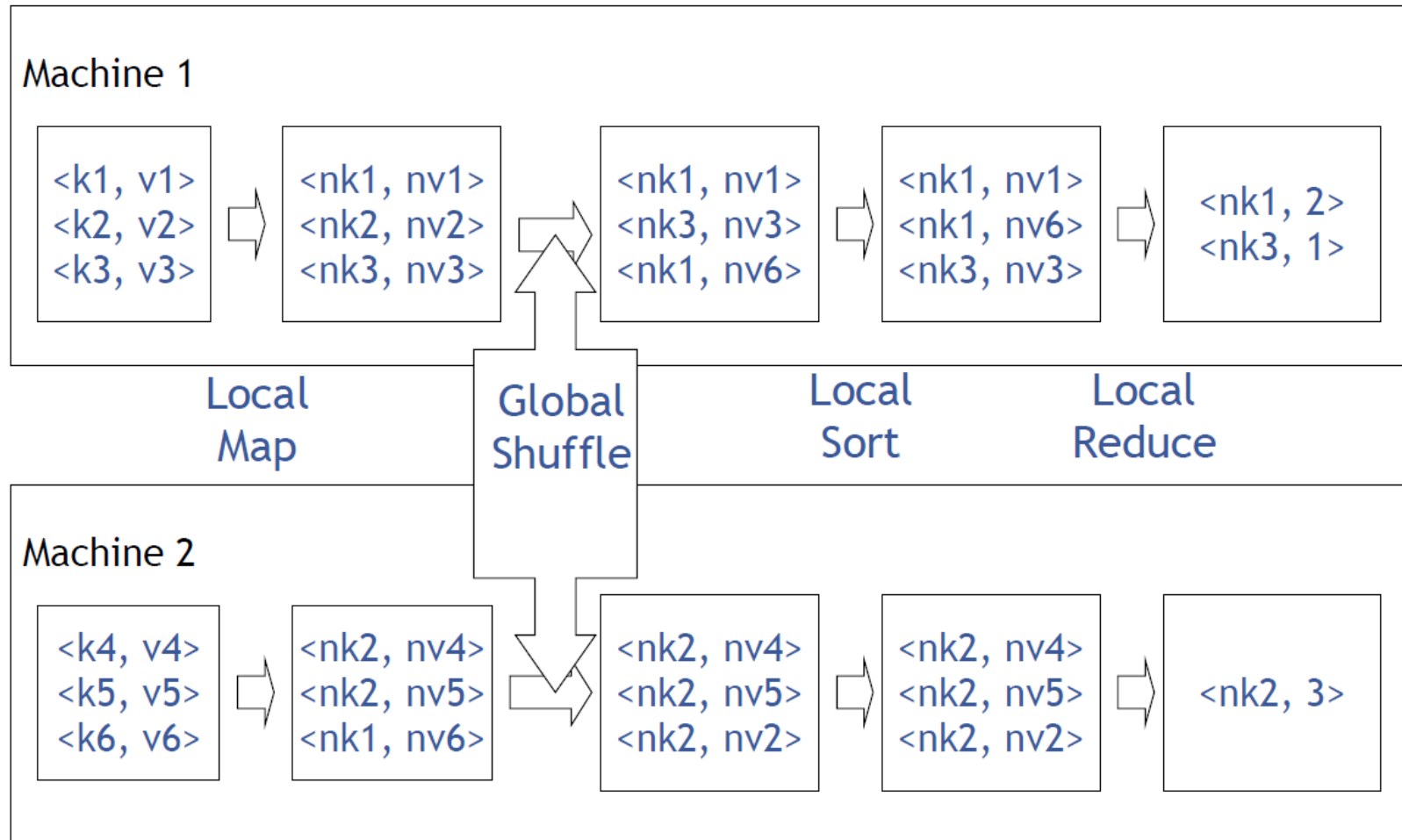
```
FROM (  
  FROM pv_users  
  SELECT TRANSFORM(pv_users.userid, pv_users.date) USING 'map_script' AS (dt, uid)  
  CLUSTER BY(dt)  
) map
```

```
INSERT INTO TABLE pv_users_reduced  
  SELECT TRANSFORM(map.dt, map.uid) USING 'reduce_script' AS (day, count);
```

Sample map\_script.py:

```
import sys  
import datetime  
  
for line in sys.stdin:  
  line = line.strip()  
  userid, unixtime = line.split('\t')  
  weekday = datetime.datetime.fromtimestamp(float(unixtime)).isoweekday()  
  print ','.join([str(weekday), userid])
```

# Apache Hive – MapReduce



# Apache Hive - HiveQL

- Joins – inner, outer

- equi-joins with conjunctions supported, also semi-joins supported

```
INSERT INTO TABLE pv_users
SELECT pv.pageid, u.age
FROM page_view pv JOIN user u ON (pv.userid = u.userid);
```

```
INSERT INTO TABLE pv_users
SELECT pv.*, u.gender, u.age
FROM page_view pv FULL OUTER JOIN user u ON (pv.userid = u.id)
WHERE pv.date = 2008-03-03;
```

- Group by

```
SELECT pageid, age, count(1)
FROM pv_users
GROUP BY pageid, age;
```

```
SELECT pageid, COUNT(DISTINCT userid)
FROM page_view GROUP BY pageid
```

# Apache Hive – Tables and Files

```
FROM pv_users
```

```
INSERT INTO TABLE pv_gender_sum  
SELECT pv_users.gender, count_distinct(pv_users.userid)  
GROUP BY(pv_users.gender)
```

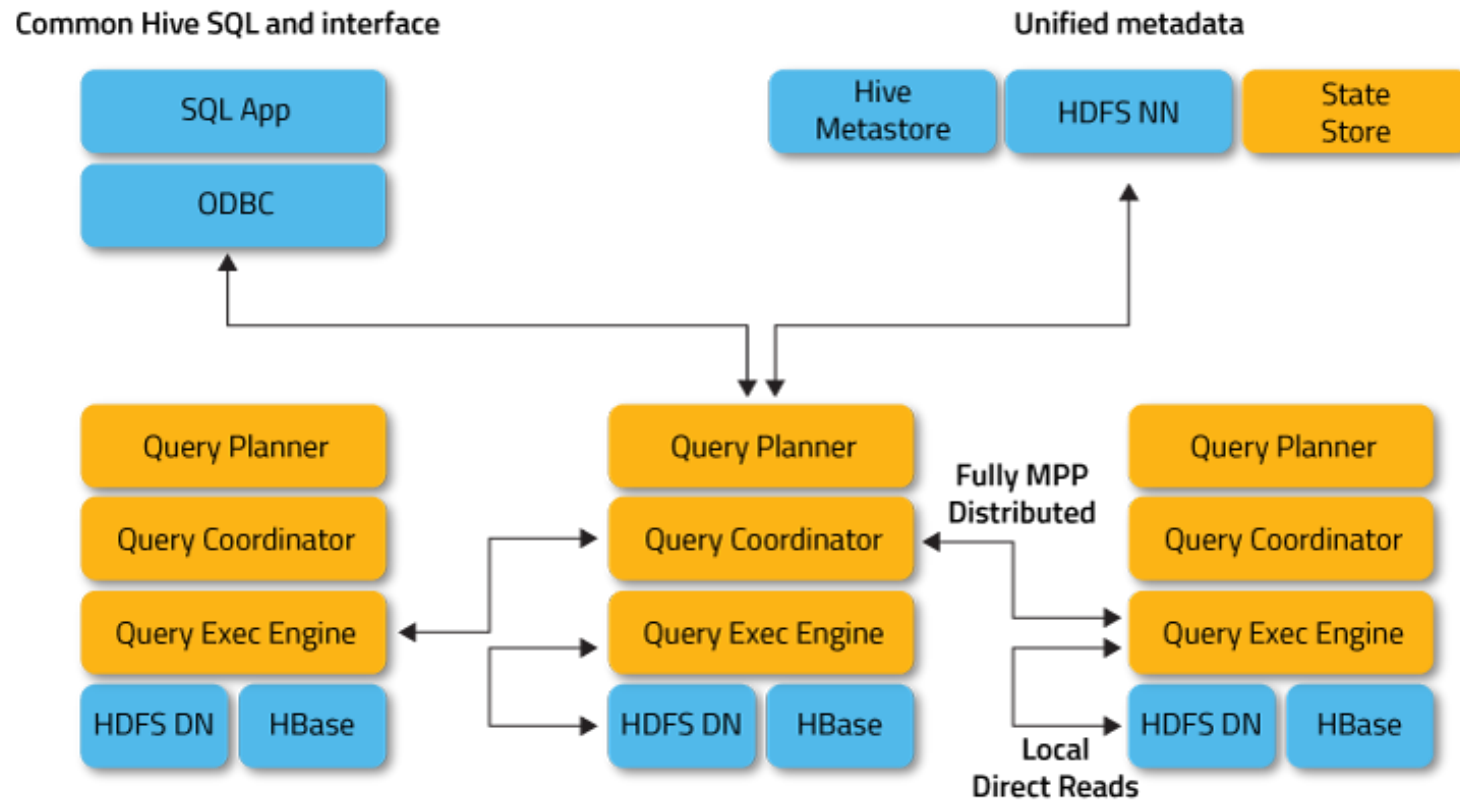
```
INSERT INTO DIRECTORY '/user/facebook/tmp/pv_age_sum.dir'  
SELECT pv_users.age, count_distinct(pv_users.userid)  
GROUP BY(pv_users.age)
```

```
INSERT INTO LOCAL DIRECTORY '/home/me/pv_age_sum.dir'  
FIELDS TERMINATED BY ',' LINES TERMINATED BY \013  
SELECT pv_users.age, count_distinct(pv_users.userid)  
GROUP BY(pv_users.age);
```

# Apache Impala

- a query engine that runs on Apache Hadoop
  - circumvents MapReduce to access the data directly
  - a specialized distributed query engine like commercial parallel RDBMSes
    - in C++, not Java; runtime code generation
- low-latency SQL queries to data stored in HDFS and Apache Hbase
  - an order-of-magnitude faster performance than Hive
- uses the same metadata, SQL syntax (HiveQL), ODBC driver, and user interface as Apache Hive
- supported storage formats
  - (compressed) text file, sequence file, RCFile, Avro, Parquet, HBase

# Apache Impala – Architecture



Source: <http://impala.apache.org/overview.html>

# Apache Impala – Query Language

- SQL support:
  - essentially SQL-92, minus correlated subqueries
  - only equi-joins; no non-equi joins, no cross products
  - Order By requires Limit
  - (Limited) DDL support
  - SQL-style authorization via Apache Sentry (incubating)
  - UDFs and UDAFs are supported
- Join Limitation
  - The smaller table has to fit in aggregate memory of all executing nodes.

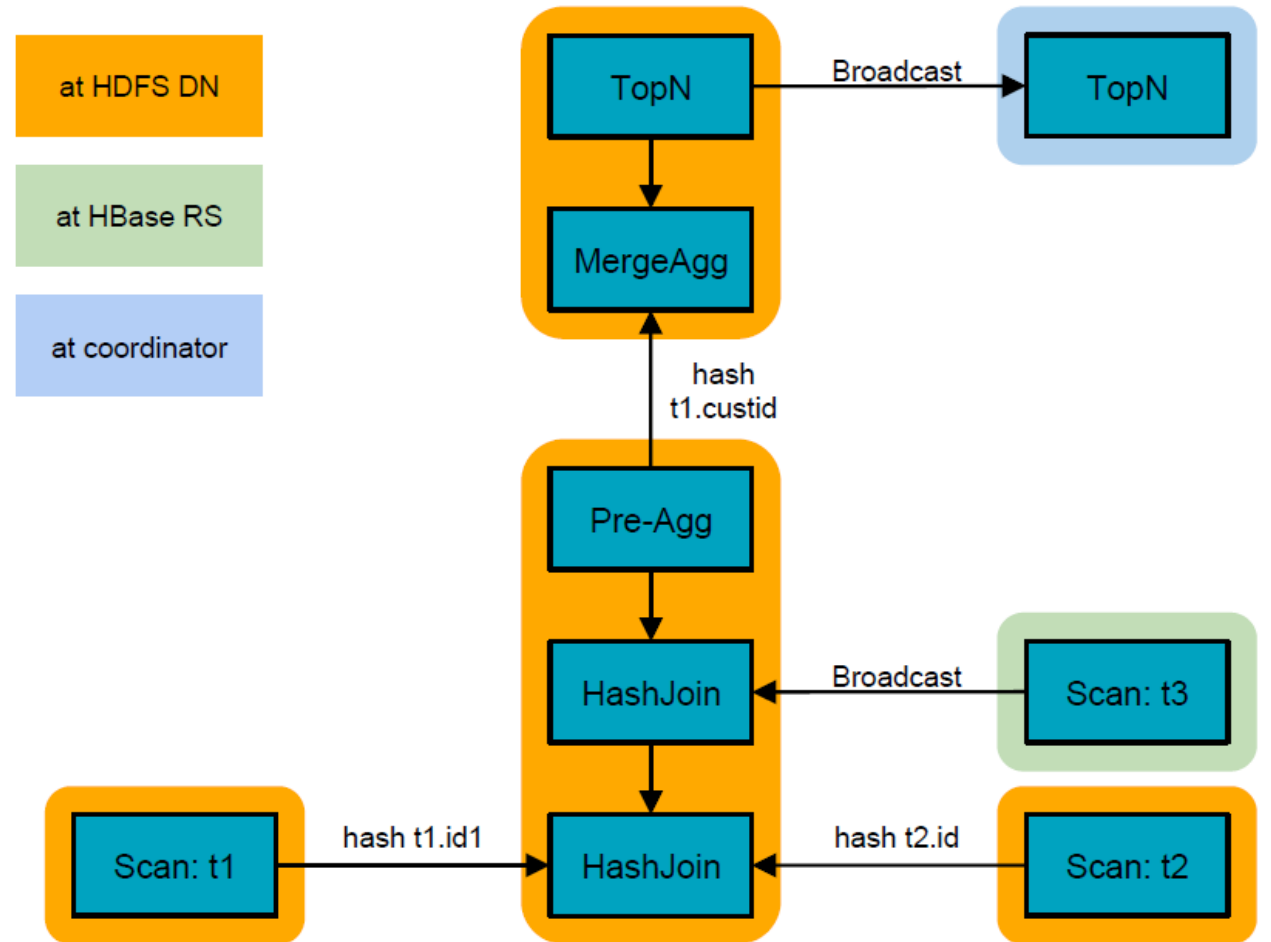


# Apache Impala – Query Planning

- 2-phase planning process:
  - single-node plan: left-deep tree of plan operators
  - plan partitioning: partition single-node plan to maximize scan locality, minimize data movement
- Parallelization of operators:
  - All query operators are fully distributed.
- Plan operators:
  - Scan, HashJoin, HashAggregation, Union, TopN, Exchange

# Apache Impala – Query Planning

```
SELECT t1.custid, SUM(t2.revenue) AS revenue
FROM LargeHdfsTable t1
  JOIN LargeHdfsTable t2 ON (t1.id1 = t2.id)
  JOIN SmallHbaseTable t3 ON (t1.id2 = t3.id)
WHERE t3.category = 'Online'
GROUP BY t1.custid
ORDER BY revenue DESC
LIMIT 10;
```



# Apache Impala – Execution Engine

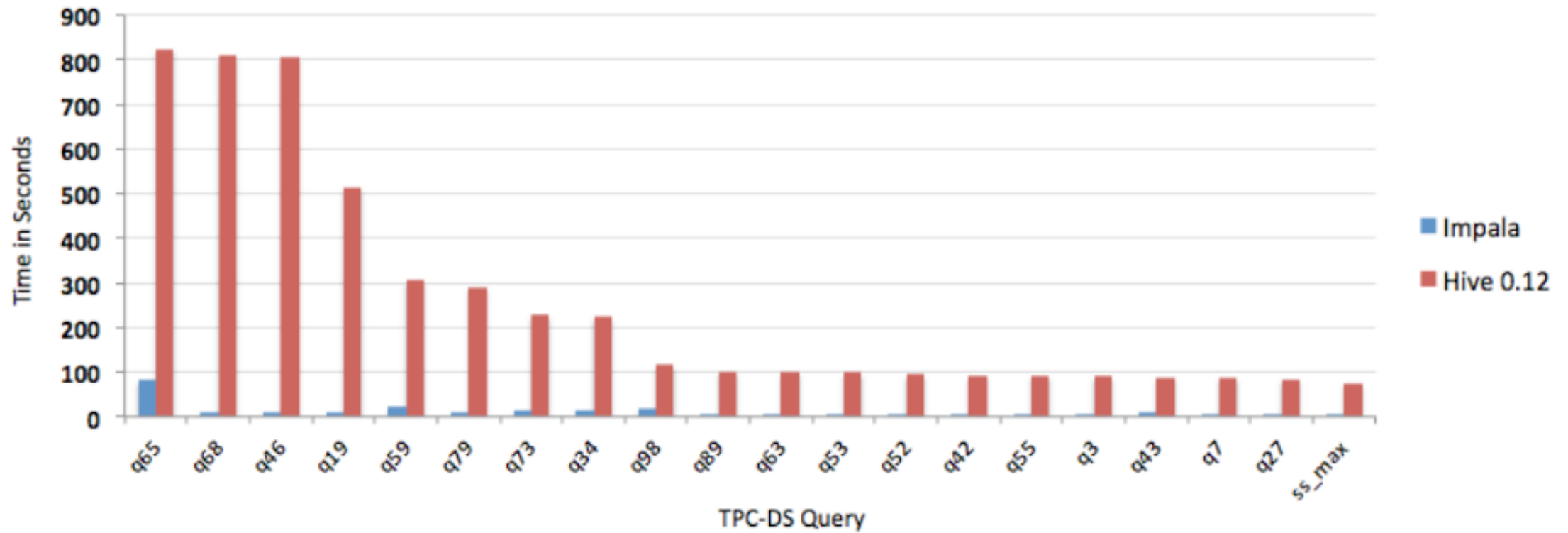
- Written in C++ for minimal execution overhead
- Internal in-memory tuple format
  - puts fixed-width data at fixed offsets
- Uses intrinsics/special cpu instructions
  - for text parsing, crc32 computation, etc.
- Runtime code generation for “big loops”
  - e.g., insert batch of rows into a hash table; unroll a loop that inlines all function calls, contains no dead code, minimizes branches
  - code generated using llvm

# Apache Hive vs. Impala - Performance

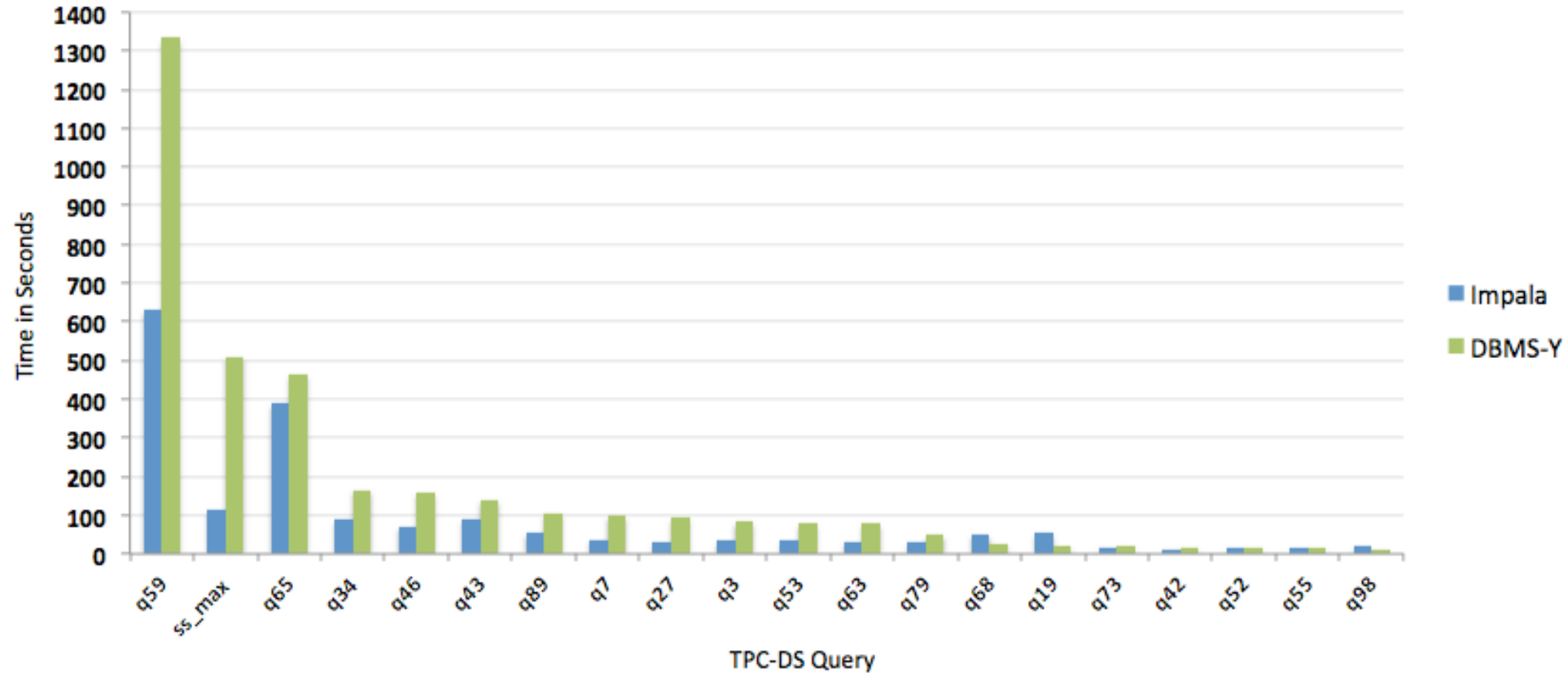
- 20 pre-selected diverse TPC-DS queries
  - modified to remove unsupported language
- Sufficient data scale for realistic comparison (3 TB, 15 TB, and 30 TB)
- Realistic nodes (e.g., 8-core CPU, 96GB RAM, 12x2TB disks)
- Methodology - multiple runs, reviewed fairness for competition, ...
- Results:
  - Impala vs Hive 0.12 (Impala 6--70x faster)
  - Impala vs “DBMS-Y” (Impala average of 2x faster)
  - Impala scalability (Impala achieves linear scale)

Source: <https://blog.cloudera.com/impala-performance-update-now-reaching-dbms-class-speed/>

# Apache Hive vs. Impala - Performance



# Apache Hive vs. Impala - Performance



# Apache Hive vs. Impala

	Hive	Impala
<b>Design</b>	MapReduce jobs	massively parallel processing (MPP)
<b>Use case</b>	long-running ETL jobs	low-latency/interactive queries, also, for multi-user load; interactive BI experience
<b>Complex data types</b>	Yes	No
<b>Query processing</b>	disk-based	in-memory

# Summary

- Big Data changes Data Warehousing to Distributed DWH
- Based on horizontally scalable frameworks
- Transition from batch processing (MR jobs) to stream processing (DAG of tasks)
- Query optimizers – special algorithms, in-memory processing
- Real-time data processing and visualizations



# Credits

- Hive Tutorial
  - <https://cwiki.apache.org/confluence/display/Hive/Tutorial>
- Facebook Data Team - HIVE: Data Warehousing & Analytics on Hadoop
  - <https://slideshare.net/zshao/hive-data-warehousing-analytics-on-hadoop-presentation>
- Mark Grover - Impala: A Modern, Open-Source SQL Engine for Hadoop
  - <https://slideshare.net/markgrover/introduction-to-impala>