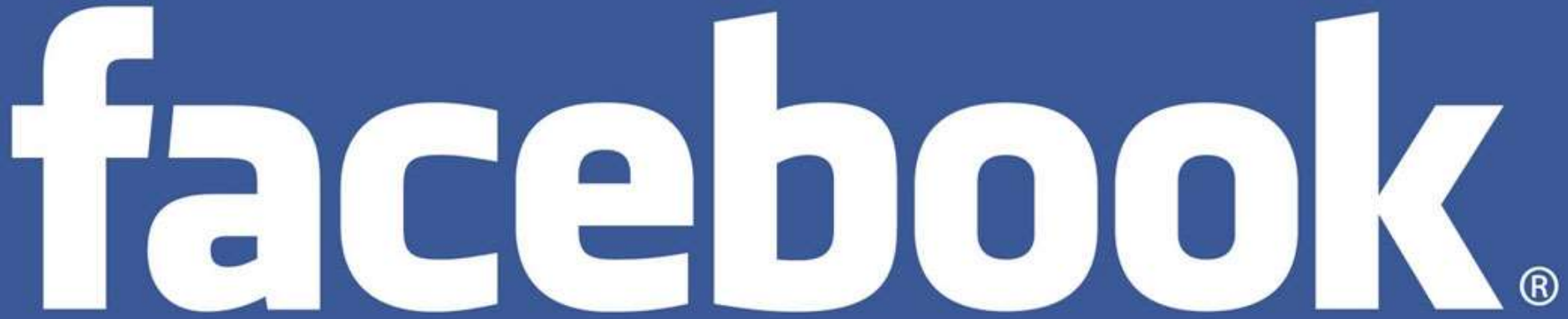
Apache Hive

Prof. Dr. Stephan Trahasch
Hochschule Offenburg



Jeff Hammerbacher, Information Platforms and the Rise of the Data Scientist. In, *Beautiful Data*, O'Reilly, 2009.

> "On the first day of logging the Facebook clickstream, more than 400 gigabytes of data was collected. The load, index, and aggregation processes for this data set really taxed the Oracle data warehouse. Even after significant tuning, we were unable to aggregate a day of clickstream data in less than 24 hours."

Hive and Pig

- Hive: data warehousing applications in Hadoop
  - Query language is HQL, variant of SQL
  - Tables stored on HDFS with different encodings
  - Developed by Facebook, now open source

- Pig: large-scale data processing system
  - Scripts are written in Pig Latin, a dataflow language
  - Programmer focuses on data transformations
  - Developed by Yahoo!, now open source
- Common idea:
  - Provide higher-level language to facilitate large-data processing
  - Higher-level language “compiles down” to Hadoop jobs

Hive: Example

- Hive looks similar to an SQL database
- Relational join on two tables:
  - Table of word counts from Shakespeare collection
  - Table of word counts from the bible

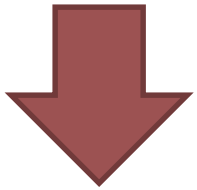
```
SELECT s.word, s.freq, k.freq FROM shakespeare s
  JOIN bible k ON (s.word = k.word) WHERE s.freq >= 1 AND k.freq >= 1
  ORDER BY s.freq DESC LIMIT 10;

the       25848    62394
I         23031    8854
and       19671    38985
to        18038    13526
of        16700    34654
a         14170    8057
you       12702    2720
my        11297    4135
in        10797    12445
is        8882     6884
```

Hive: Behind the Scenes

```
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;
```

(Abstract 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)))
```



(one or more of MapReduce jobs)

Hive: Behind the Scenes

```
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
```

```
Reduce Operator Tree:
        Join Operator
          condition map:
               Inner Join 0 to 1
          condition expressions:
            0 {VALUE._col0} {VALUE._col1}
            1 {VALUE._col0}
          outputColumnNames: _col0, _col1, _col2
          Filter Operator
            predicate:
                expr: ((_col0 >= 1) and (_col2 >= 1))
                type: boolean
            Select Operator
              expressions:
                    expr: _col1
                    type: string
                    expr: _col0
                    type: int
                    expr: _col2
                    type: int
              outputColumnNames: _col0, _col1, _col2
              File Output Operator
                compressed: false
                GlobalTableId: 0
                table:
                    input format:
org.apache.hadoop.mapred.SequenceFileInputFormat
                    output format:
org.apache.hadoop.hive.ql.io.HiveSequenceFileOutputFormat
```

```
Stage: Stage-2
   Map Reduce
     Alias -> Map Operator Tree:
       hdfs://localhost:8022/tmp/hive-training/36421437
           Reduce Output Operator
             key expressions:
                   expr: _col1
                   type: int
             sort order: -
             tag: -1
             value expressions:
                   expr: _col0
                   type: string
                   expr: _col1
                   type: int
                   expr: _col2
                   type: int
     Reduce Operator Tree:
       Extract
         Limit
           File Output Operator
             compressed: false
             GlobalTableId: 0
             table:
                 input format:
org.apache.hadoop.mapred.TextInputFormat
                 output format:
org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputForm


Stage: Stage-0
   Fetch Operator
     limit: 10
```

Hive Usage @Facebook in 2010

Statistics per day:

- 4 TB of compressed new data added per day
- 135 TB of compressed data scanned per day
- 7500+ Hive jobs on per day

Hive simplifies Hadoop:

- ~200 people/month run jobs on Hadoop/Hive
- Analysts (non-engineers) use Hadoop through Hive
- 95% of jobs are Hive Jobs

Outline

- Introduction
- Creating Tables
- Data Storage
- Data Hierarchy
- Loading Data
- Metastore
- Summary

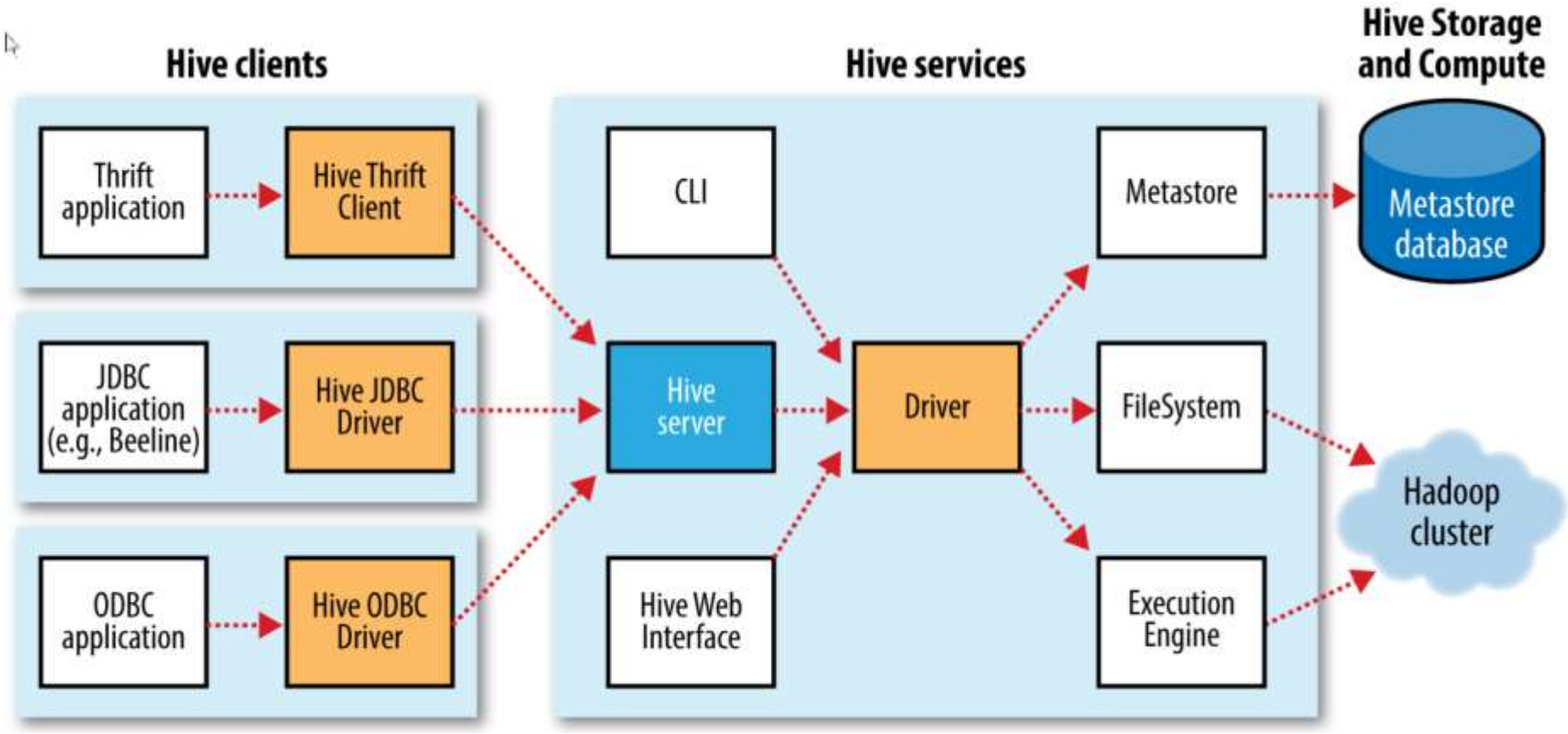
What Is Hive?

- A data warehousing infrastructure based on Hadoop
- Hive is designed and great for batch processing
- Immediately makes data on a cluster available to non-Java programmers via SQL like queries
- Built on HiveQL (HQL), a SQL-like query language
- Interprets HiveQL and generates MapReduce jobs that run on the cluster
- Enables easy data summarization, ad-hoc reporting and querying, and analysis of large volumes of data
- Developed by Facebook and a top-level Apache project

What Hive Is Not

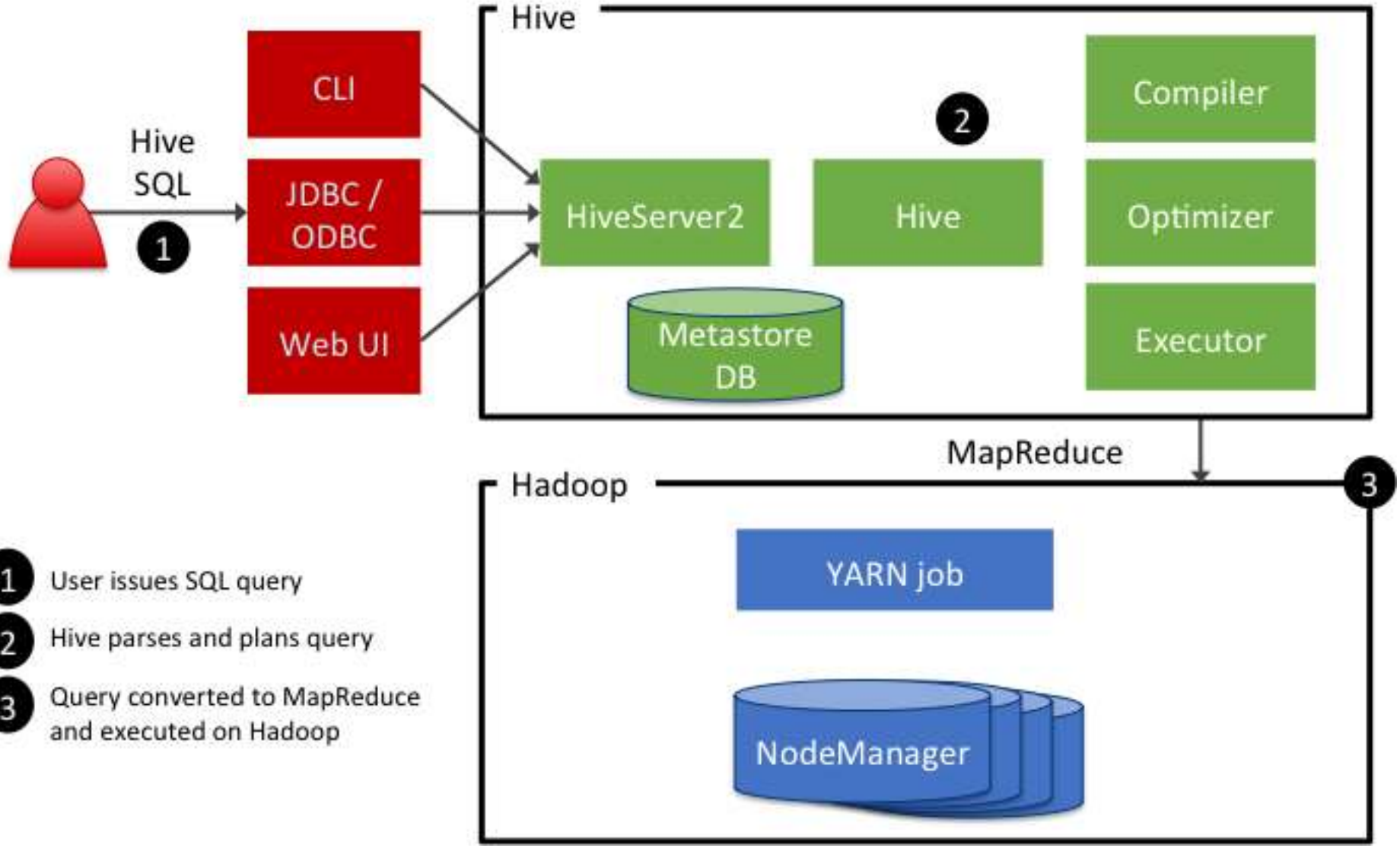
- Hive, like Hadoop, is designed for batch processing of large datasets
- Not an OLTP or real-time system
- Latency and throughput are both high compared to a traditional RDBMS!
- Even when dealing with relatively small data (<100 MB)

Hive Architecture



Source: Hadoop – The definitive Guide

Hive Architecture

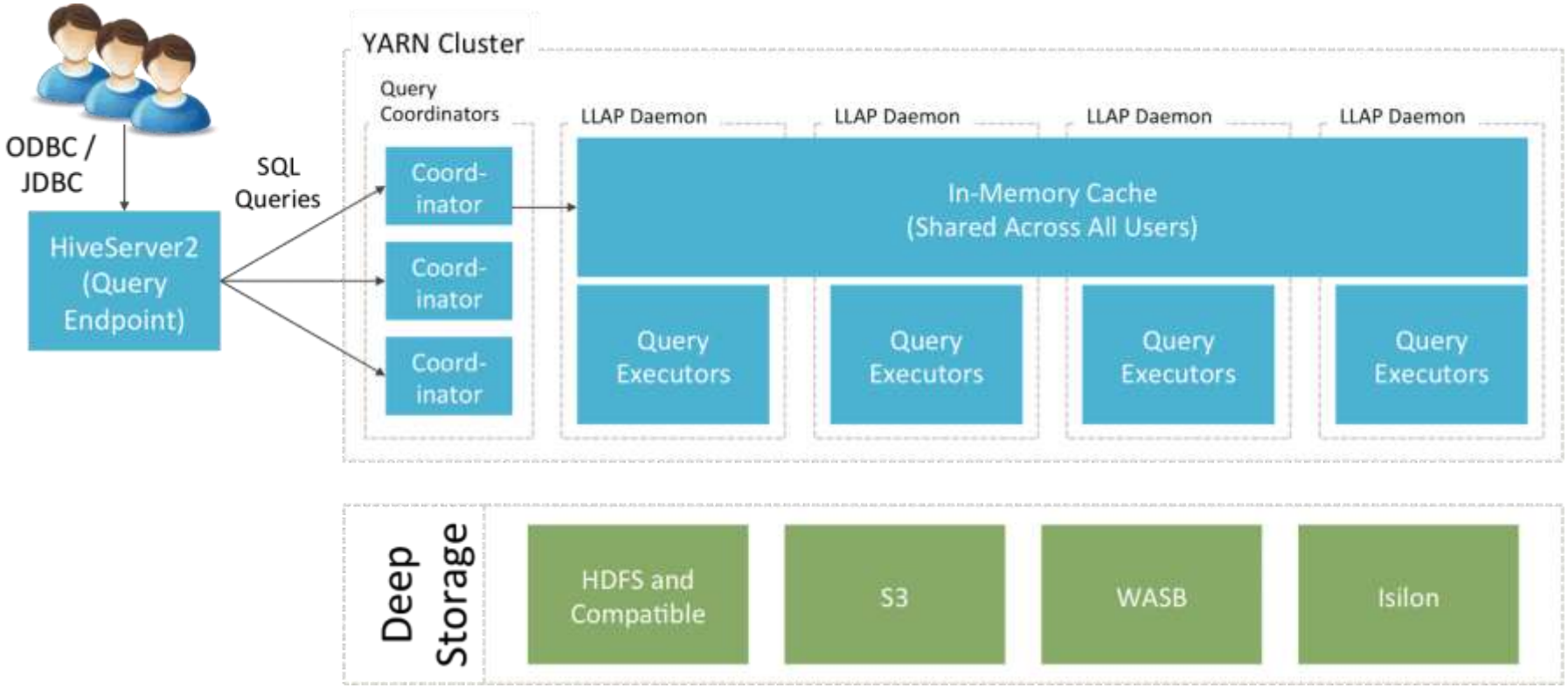


HiveQL / HQL provides the basic SQL-like operations

- Select columns using SELECT
- Filter rows using WHERE
- JOIN between tables
- Evaluate aggregates using GROUP BY
- Store query results into another table
- Download results to a local directory (i.e., export from HDFS)
- Manage tables and queries with CREATE, DROP, and ALTER
- Hive CLI: Traditional Hive client that connects to a HiveServer instance
  - `$ hive`
    `hive>`
- Beeline: A new command-line client that connects to a HiveServer2 instance
  - `$ beeline –u` *`url`* `–n` *`username`* `–p` *`password`*
    `beeline>`

Hive 2

Hive 2 shifts from a disk-centric architecture to a memory-centric architecture through Hive LLAP (Live Long and Process).



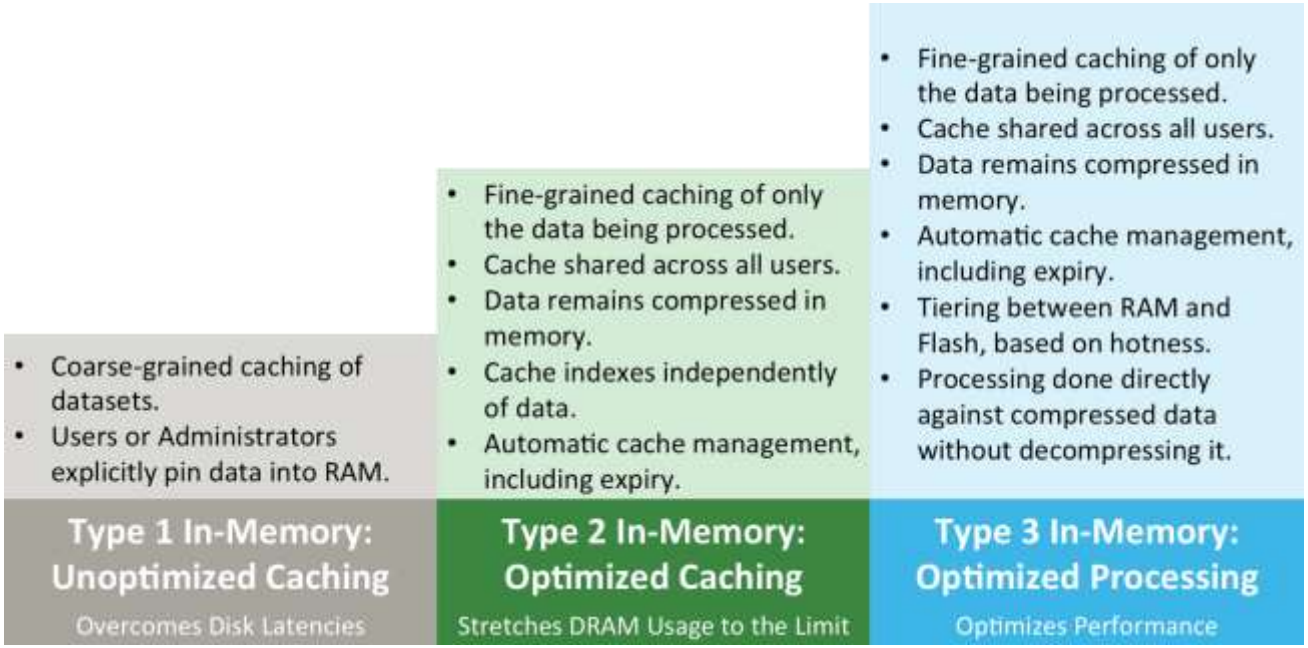
https://de.hortonworks.com/blog/apache-hive-going-memory-computing/

Hive LLAP - Live Long and Process

- LLAP uses persistent query servers to avoid long startup times and deliver fast SQL.
- LLAP shares its in-memory cache among all SQL users, maximizing the use of this scarce resource.
- Pre-fetching and caching of column chunks
- LLAP has fine-grained resource management and preemption, making it great for highly concurrent access across many users.
- Multi-threaded JIT-friendly operator pipelines

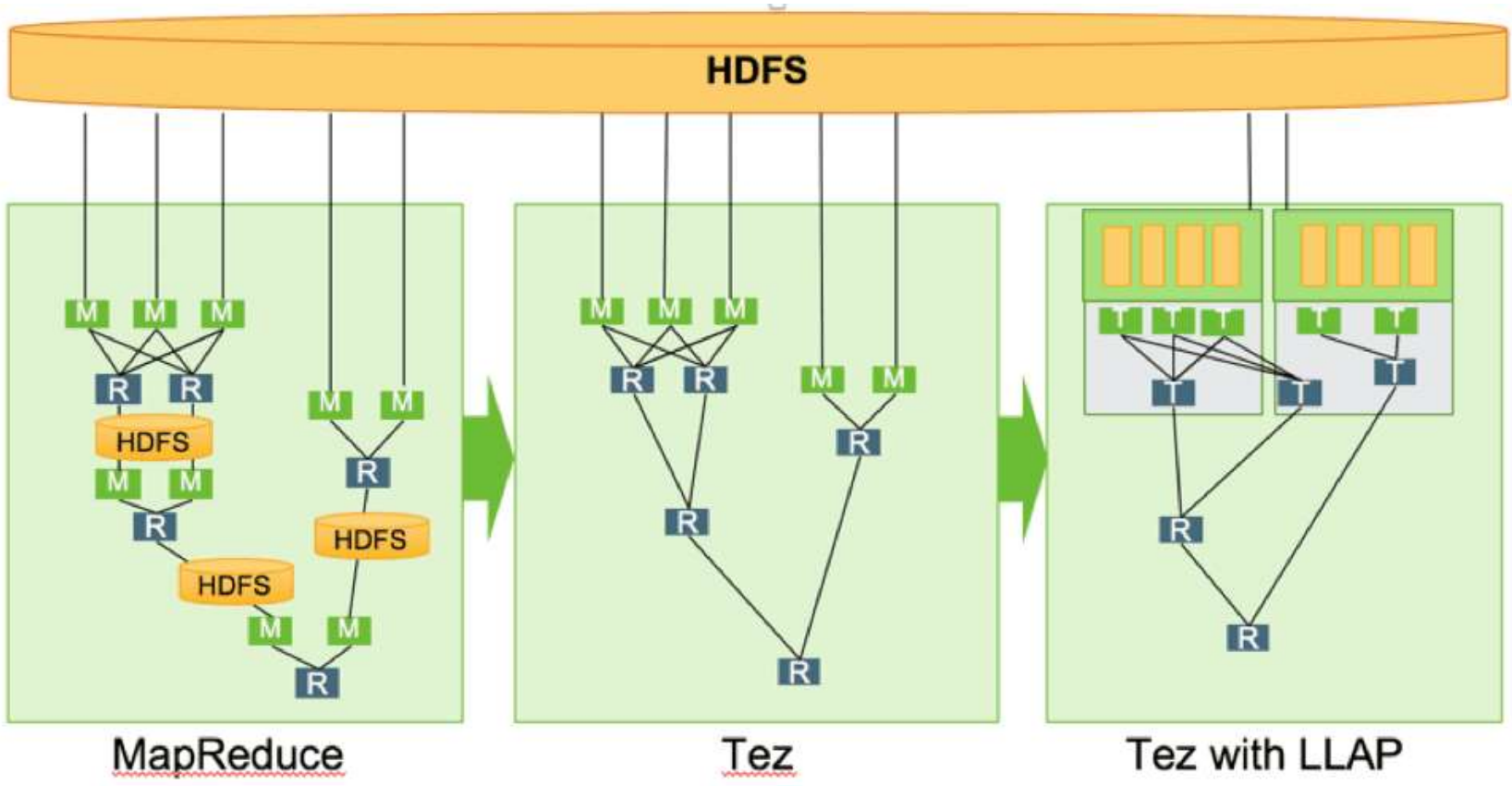
Caching

- You must cache only the specific data needed for processing queries.
- The cache needs to be shared across all users who need the data.
- The data needs to stay compressed in RAM to offset the 100x higher cost of RAM.



https://de.hortonworks.com/blog/apache-hive-going-memory-computing/

Hive 2: From MapReduce to Tez and LLAP



LLAP provides a hybrid execution model.

- LLAP consists of a long-lived daemon which replaces direct interactions with the HDFS DataNode, and a tightly integrated DAG-based framework.
- Functionality such as caching, pre-fetching, some query processing and access control are moved into the daemon.
- Small/short queries are largely processed by this daemon directly, while any heavy lifting will be performed in standard YARN containers.
- Similar to the DataNode, LLAP daemons can be used by other applications as well.

Persistent Daemon runs on the worker nodes

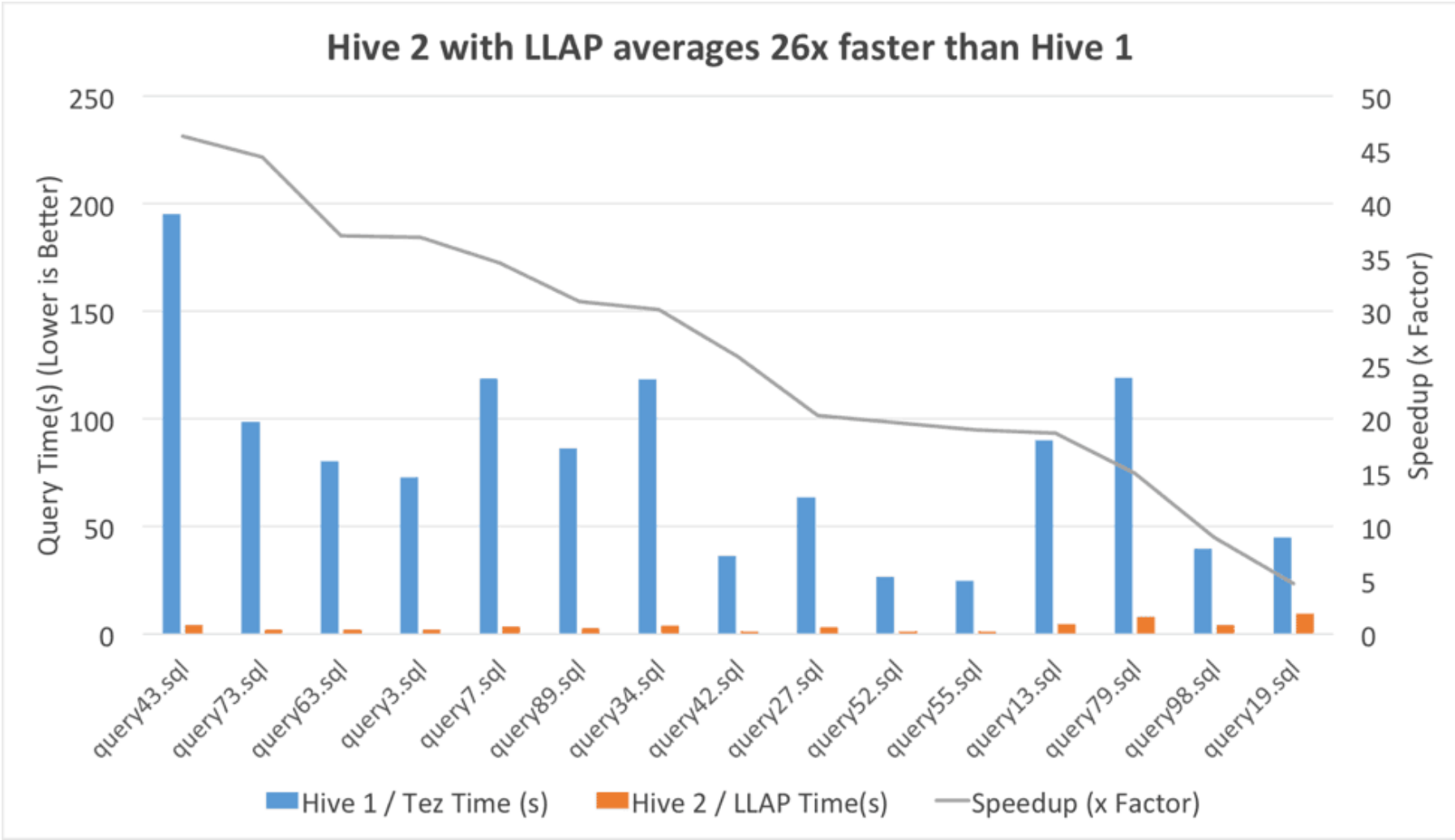
- To facilitate caching and JIT optimization, and to eliminate most of the startup costs. It handles I/O, caching, and query fragment execution.
- These nodes are stateless. Any request to an LLAP node contains the data location and metadata. It processes local and remote locations; locality is the caller's responsibility (YARN).
- Recovery/resiliency
  Failure and recovery is simplified because any data node can still be used to process any fragment of the input data. The Tez AM can thus simply rerun failed fragments on the cluster.
- Communication between nodes
  LLAP nodes are able to share data (e.g., fetching partitions, broadcasting fragments). This is realized with the same mechanisms used in Tez.

Execution Engine does not replace the existing execution model but rather enhances it.

The daemons are optional. Hive can work without them and also is able to bypass them even if they are deployed and operational. Feature parity with regard to language features is maintained.

- External orchestration and execution engines.
  LLAP is not an execution engine like MapReduce or Tez. Execution is scheduled and monitored by an existing Hive execution engine (such as Tez) transparently over both LLAP nodes, as well as regular containers.
- Partial execution.
  The result of the work performed by an LLAP daemon can either form part of the result of a Hive query, or be passed on to external Hive tasks, depending on the query.
- Resource Management.
  YARN remains responsible for the management and allocation of resources. The YARN container delegation model is used to allow the transfer of allocated resources to LLAP.

Hive 1 with Tez versus Hive 2 with LLAP



Outline

- Introduction
- Creating Tables
- Data Storage
- Data Hierarchy
- Loading Data
- Metastore
- Summary

How Hive Load and Store Data (1)

- Queries operate on tables, just like in an RDBMS
- A table is simply an HDFS directory containing one or more tables
  Default path: `/user/hive/warehouse/<table_name>`
- Supports many formats for data storage and retrieval

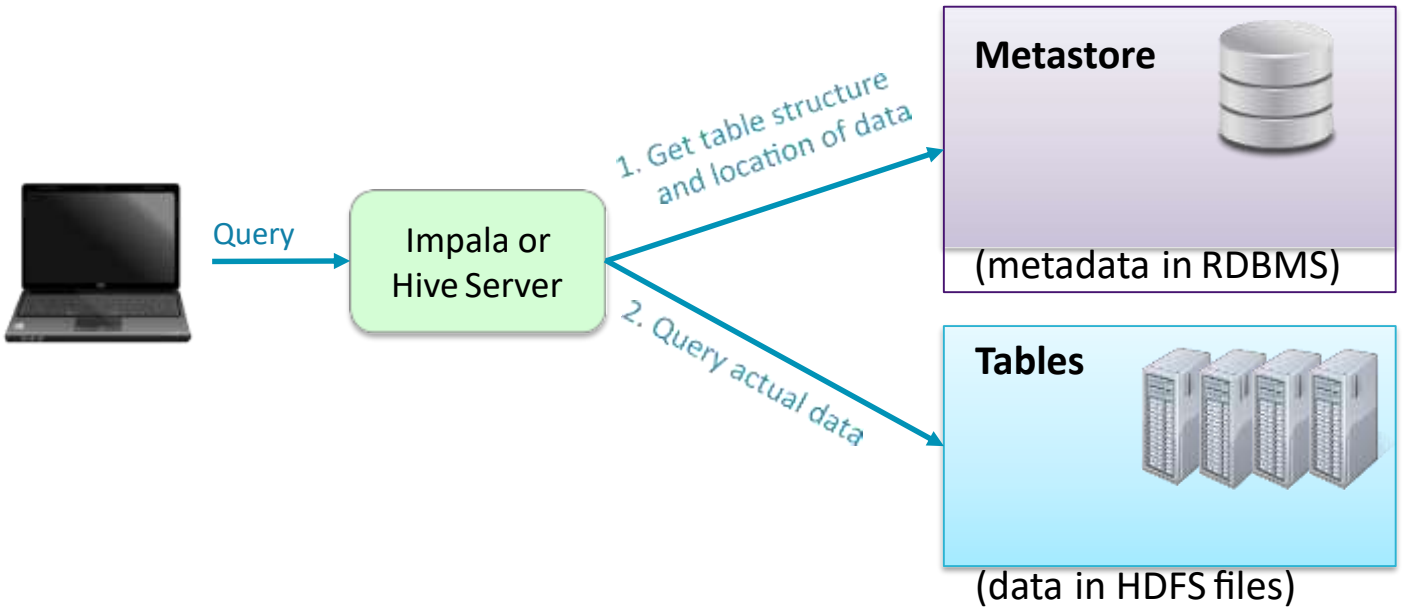
What is the structure and location of tables?

- These are specified when tables are created
- This metadata is stored in the Metastore (RDBMS like MySQL)
  → Tables in HDFS, metadata in the Metastore

How Hive Load and Store Data (2)

Hive and Impala use the Metastore to determine data format and location

The query itself operates on data stored in HDFS



Source: Cloudera

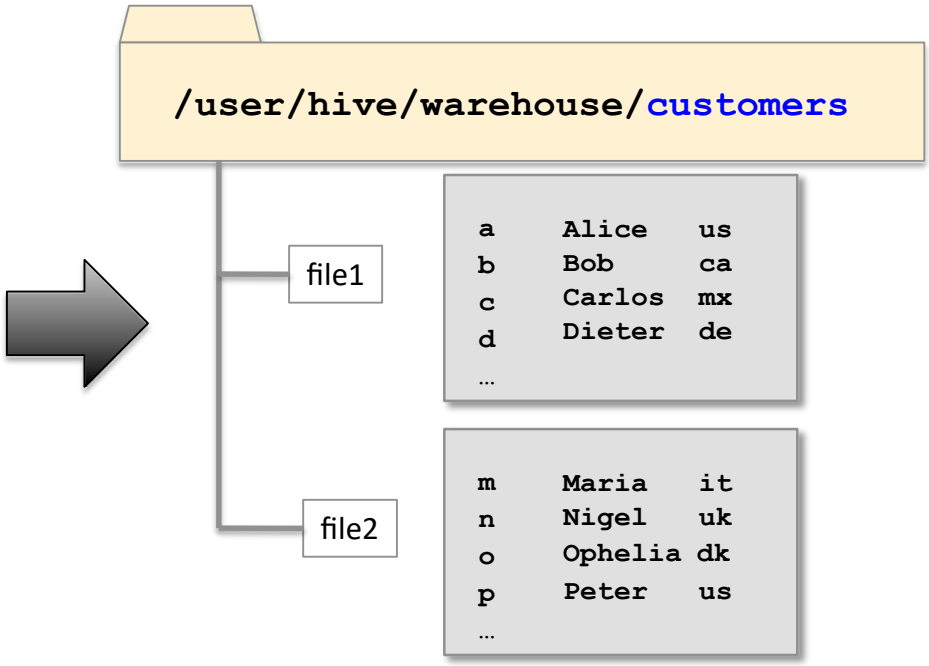
The Data Warehouse Directory

By default, data is stored in the HDFS directory `/user/hive/warehouse`

Each table is a subdirectory containing any number of files

customers table

cust_id	name	country
001	Alice	us
002	Bob	ca
003	Carlos	mx
…	…	…
392	Maria	it
393	Nigel	uk
394	Ophelia	dk
…	…	…



Source: Cloudera

Creating a Database

Databases and tables are created and managed using the DDL of HiveQL or Impala SQL. Very similar to standard SQL DDL.

To create a new database

```
CREATE DATABASE loudacre;
```

Adds the database definition to the metastore

Creates a storage directory in HDFS

`/user/hive/warehouse/loudacre.db`

To avoid error in case database already exists (useful for scripting)

```
CREATE DATABASE IF NOT EXISTS loudacre;
```

Creating a Table

Basic syntax for creating a table:

```
CREATE TABLE tablename (colname DATATYPE, ...)
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY char
  STORED AS {TEXTFILE|SEQUENCEFILE|…};
```

Creates a subdirectory in the database's warehouse directory in HDFS

Default database:

`/user/hive/warehouse/tablename`

Named database:

`/user/hive/warehouse/dbname.db/tablename`

Use LIKE to create a new table based on an existing table definition

```
CREATE TABLE jobs_archived LIKE jobs;
```

Example Table Definition

The following example creates a new table named jobs

Data stored as text with four comma-separated fields per line

```
CREATE TABLE jobs (
    id INT,
    title STRING,
    salary INT,
    posted TIMESTAMP
  )
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ',';
```

Example of corresponding record for the table above

```
1,Data Analyst,100000,2013-06-21 15:52:03
```

Source: Cloudera

Creating Tables Based On Existing Data

Create a table based on a SELECT statement

Often know as ‘Create Table As Select’ (CTAS)

```
CREATE TABLE ny_customers AS SELECT
  cust_id, fname, lname
  FROM customers
  WHERE state = 'NY';
```

Source: Cloudera

Primitive Data Types

Type	Comments
TINYINT, SMALLINT, INT, BIGINT	1, 2, 4 and 8-byte integers
BOOLEAN	TRUE/FALSE
FLOAT, DOUBLE	Single and double precision real numbers
STRING	Character string
TIMESTAMP	Unix-epoch offset *or* datetime string
DECIMAL	Arbitrary-precision decimal
BINARY	Opaque; ignore these bytes

Complex Data Types

Type	Comments
STRUCT	A collection of elements If S is of type STRUCT {a INT, b INT}: S.a returns element a
MAP	Key-value tuple If M is a map from 'group' to GID: M['group'] returns value of GID
ARRAY	Indexed list If A is an array of elements ['a','b','c']: A[0] returns 'a'

Removing a Database

Removing a database is similar to creating it

```
DROP DATABASE IF EXISTS loudacre;
```

These commands will fail if the database contains tables

In Hive: Add the CASCADE keyword to force removal

Caution: this command might remove data in HDFS!


```
DROP DATABASE loudacre CASCADE;
```

Outline

- Introduction
- Creating Tables
- Data Storage
- Data Hierarchy
- Loading Data
- Metastore
- Summary

Controlling Table Data Location

- By default, table data is stored in the warehouse directory
- This is not always ideal
  Data might be shared by several users
- Use LOCATION to specify the directory where table data resides

```
CREATE TABLE jobs (
    id INT, title STRING, salary INT, posted TIMESTAMP
  )
  ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
  LOCATION '/loudacre/jobs';
```

Source: Cloudera

Externally Managed Tables

- CAUTION: Dropping a table removes its data in HDFS
- Tables are “managed” or “internal” by default
- Using EXTERNAL when creating the table avoids this behavior
- Dropping an external table removes only its metadata

```
CREATE EXTERNAL TABLE adclicks
  ( campaign_id STRING,
    click_time TIMESTAMP,
    keyword STRING,
    site STRING,
    placement STRING,
    was_clicked BOOLEAN,
    cost SMALLINT)
  LOCATION '/loudacre/ad_data';
```

Source: Cloudera

Create external table

CREATE TABLE

- LOAD: file moved into Hive's data warehouse directory
- DROP: both metadata and data deleted

CREATE EXTERNAL TABLE

- LOAD: no files moved
- DROP: only metadata deleted
- Use this when sharing with other Hadoop applications, or when you want to use multiple schemas on the same data

Exploring Tables (1)

The SHOW TABLES command
lists all tables in the current database

```
SHOW TABLES;
+---------------+
|   tab_name    |
+---------------+
| accounts      |
| employees     |
| job           |
| vendors       |
+---------------+
```

The DESCRIBE command lists
the fields in the specified table

```
DESCRIBE jobs;
+--------+-----------+---------+
| name   | type      | comment |
+--------+-----------+---------+
| id     | int       |         |
| title  | string    |         |
| salary | int       |         |
| posted | timestamp |         |
+--------+-----------+---------+
```

Exploring Tables (2)

DESCRIBE FORMATTED also shows table properties

```
DESCRIBE FORMATTED jobs;
+------------------+----------------------------------------+---------+
| name             | type                                   | comment |
+------------------+----------------------------------------+---------+
| # col_name       | data_type                              | comment |
| id               | int                                    | NULL    |
| title            | string                                 | NULL    |
| salary           | int                                    | NULL    |
| posted           | timestamp                              | NULL    |
|                  | NULL                                   | NULL    |
| # Detailed Table | NULL                                   | NULL    |
| Information
| Database:        | default                                | NULL    |
| Owner:           | training                               | NULL    |
| CreateTime:      | Wed Jun 17 09:41:23 PDT 2015           | NULL    |
| LastAccessTime:  | UNKNOWN                                | NULL    |
| Protect Mode:    | None                                   | NULL    |
| Retention:       | 0                                      | NULL    |
| Location:        | hdfs://localhost:8020/loudacre/jobs    | NULL    |
| Table Type:      | MANAGED_TABLE                          | NULL    |
…
```

Browsing Tables And Partitions

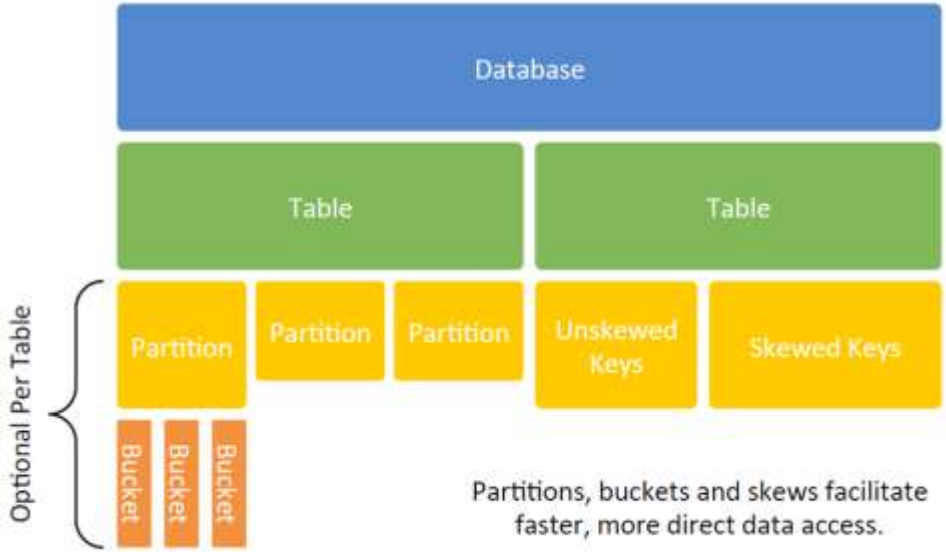
Command	Comments
`SHOW TABLES;`	Show all the tables in the database
`SHOW TABLES 'page.*';`	Show tables matching the specification (uses regex syntax)
`SHOW PARTITIONS page_view;`	Show the partitions of the page_view table
`DESCRIBE page_view;`	List columns of the table
`DESCRIBE EXTENDED page_view;`	More information on columns (useful only for debugging)
`DESCRIBE page_view PARTITION (ds='2008-10-31');`	List information about a partition

Outline

- Introduction
- Creating Tables
- Data Storage
- Data Hierarchy
- Loading Data
- Metastore
- Summary

Hive is organised hierarchically into

- Databases
  namespaces that separate tables and other objects
- Tables
  homogeneous units of data with the same schema
- Partitions
  determine how the data is stored
  - Allow efficient access to subsets of the data
- Buckets/clusters
  - For subsampling within a partition
  - Join optimization



Hive Partitions

Use **partitioned by** clause to define a partition when creating table:

```
create table employees (id int, name string, salary double)
partitioned by (dept string);
```

Subfolders are created based on the partition values:

```
/apps/hive/warehouse/employees
        /dept=hr/
        /dept=support/
        /dept=engineering/
        /dept=training/
```

Can make some queries faster

Divide data based on partition column

Use PARTITION BY clause when creating table

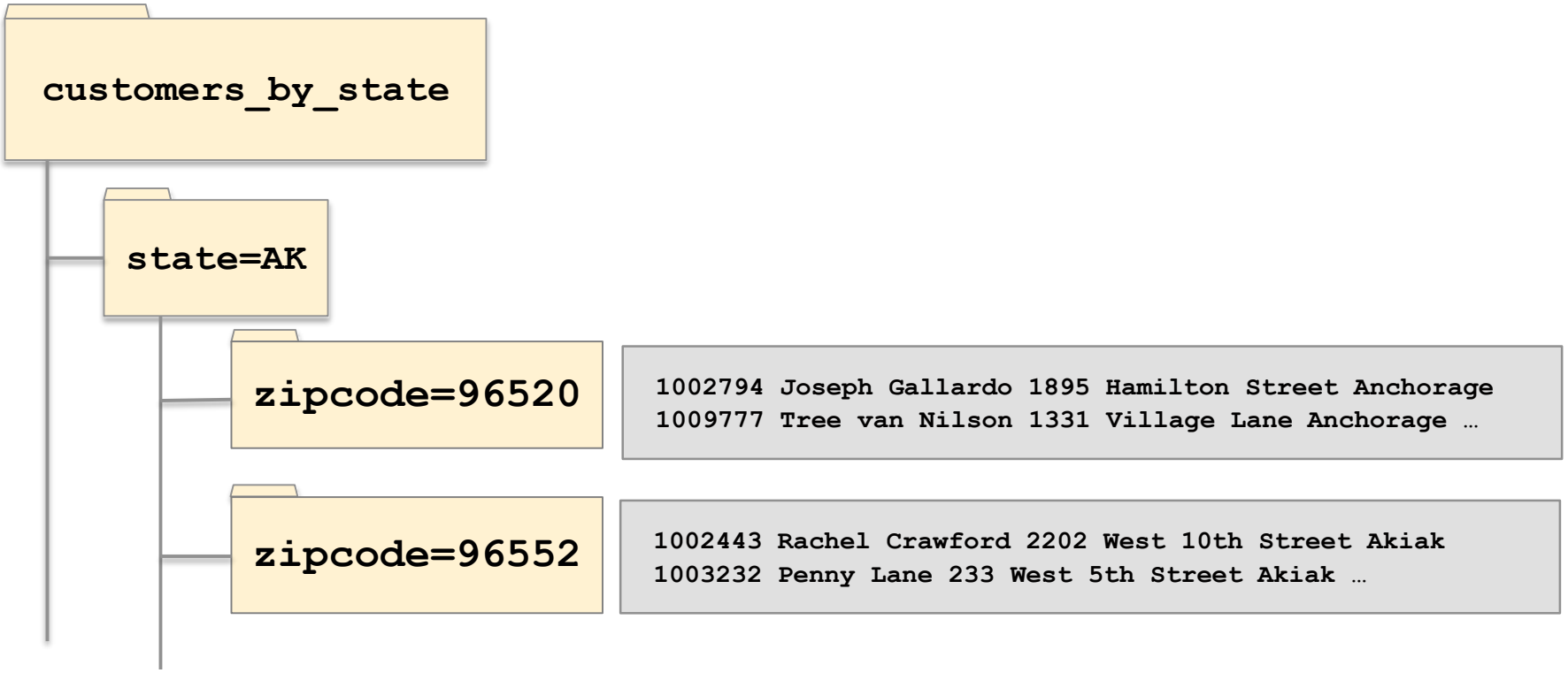
Use PARTITION clause when loading data

SHOW PARTITIONS will show a table's partitions

A table may be partitioned in multiple dimensions.

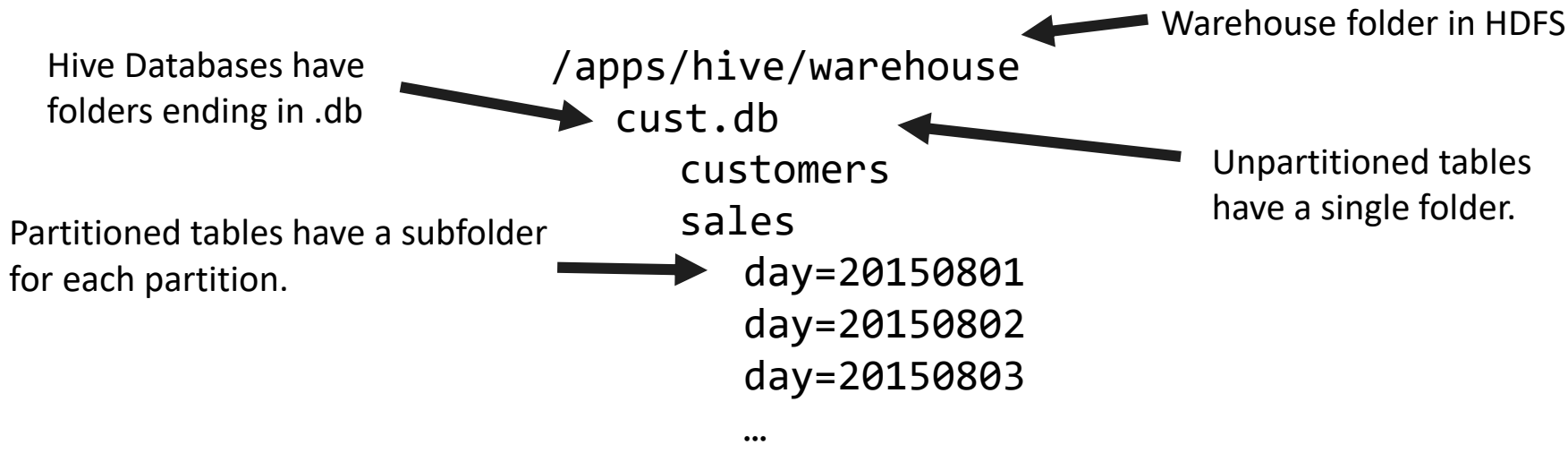
For example, in addition to partitioning customers by state, we might also subpartition each state partition by zipcode to permit efficient queries by zipcode

```
… PARTITIONED BY (state STRING, zipcode STRING)
```



Partitioning Hive

- Each partition is associated with a folder in HDFS
- All partitions have an entry in the Hive Catalog
- Hive optimizer will parse the query for filter conditions and skip unneeded partitions
- **Usage consideration**
  - Too many partitions can lead to bad performance in Hive Catalog & Optimizer
  - No range partitioning / no continuous values
  - Normally date partitioned by data load

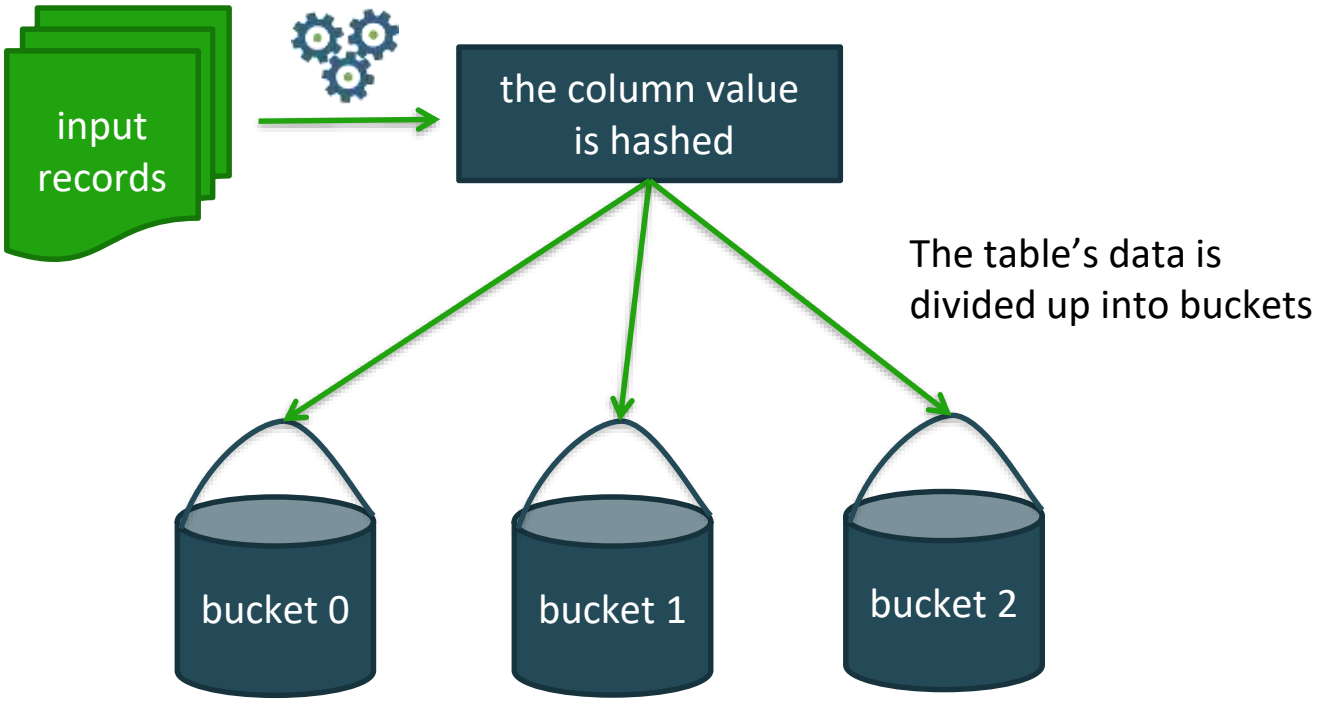


Reasons for organizing your tables (or partitions) into buckets.

1. More efficient queries.
   Bucketing imposes extra structure on the table, which Hive can take advantage of when performing certain queries.
   In particular, a join of two tables that are bucketed on the same columns - which include the join columns - can be efficiently implemented as a map-side join.

2. Make sampling more efficient.
   When working with large datasets, it is very convenient to try out queries on a fraction of your dataset while you are in the process of developing or refining them.

Hive Buckets



- Can speed up queries that involve sampling the data
  Sampling works without bucketing, but Hive has to scan the entire dataset
- Use CLUSTERED BY when creating table
  For sorted buckets, add SORTED BY
- To query a sample of your data, use TABLESAMPLE

CLUSTERED tells Hive that a table shoud be bucketed

```
CREATE TABLE bucketed_users (id INT, name STRING)
CLUSTERED BY (id)
SORTED BY (id ASC)
INTO 4 BUCKETS;
```

- Here we are using the user ID to determine the bucket.
- Hive does this by hashing the value and reducing modulo the number of buckets, so any particular bucket will effectively have a random set of users in it.
- The data within a bucket may additionally be sorted by one or more columns. This allows even more efficient map-side joins, since the join of each bucket becomes an efficient merge sort.

Map-side join case for two tables

```
SELECT sales.*, things.*
FROM sales JOIN things ON (sales.id = things.id);
```

- When the two tables are bucketed in the same way, a mapper processing a bucket of the left table knows that the matching rows in the right table are in its corresponding bucket, so it need only retrieve that bucket (which is a small fraction of all the data stored in the right table) to effect the join.
- This optimization also works when the number of buckets in the two tables are multiples of each other; they do not have to have exactly the same number of buckets.
- If one table is small enough to fit in memory, Hive can load it into memory to perform the join in each of the mappers → map join.

Outline

- Introduction
- Creating Tables
- Data Storage
- Data Hierarchy
- Loading Data
- Metastore
- Summary

Loading Data From HDFS Files

- To load data, simply add files to the table's directory in HDFS
- Can be done directly using the hdfs dfs commands
- This example loads data from HDFS into the sales table

```
$ hdfs dfs -mv \
  /tmp/sales.txt   /user/hive/warehouse/sales/
```

- Alternatively, use the LOAD DATA INPATH command
- This moves data within HDFS, just like the command above
- Source can be either a table or directory

```
LOAD DATA INPATH '/tmp/sales.txt'
   OVERWRITE INTO TABLE sales;
```

Source: Cloudera

Appending Selected Records to a Table

Another way to populate a table is through a query

Use INSERT INTO to add results to an existing Hive table

```
INSERT INTO TABLE accounts_copy
  SELECT * FROM accounts;
```

Specify a WHERE clause to control which records are appended

```
INSERT INTO TABLE loyal_customers
  SELECT * FROM accounts
  WHERE YEAR(acct_create_dt) =
  2008  AND acct_close_dt IS NULL;
```

Loading Data From a Relational Database

Sqoop has built-in support for importing data into Hive

Add the --hive-import option to your Sqoop command

Creates the table in the Hive metastore

Imports data from the RDBMS to the table's directory in HDFS

```
$ sqoop import \
  --connect jdbc:mysql://localhost/loudacre \
  --username training \
  --password training \
  --fields-terminated-by '\t' \
  --table employees \
  --hive-import
```

Loading Data with Dynamic Partitioning

```
CREATE TABLE ORC_SALES
  (clientid INT, dt DATE, rev DOUBLE, profit DOUBLE, comment STRING )
  PARTITIONED BY (country STRING )
  STORED AS ORC;
INSERT INTO TABLE orc_sales PARTITION (country) SELECT * FROM del_sales;
```

Dynamic partition columns need to be the last columns in your dataset. Change order in SELECT list if necessary

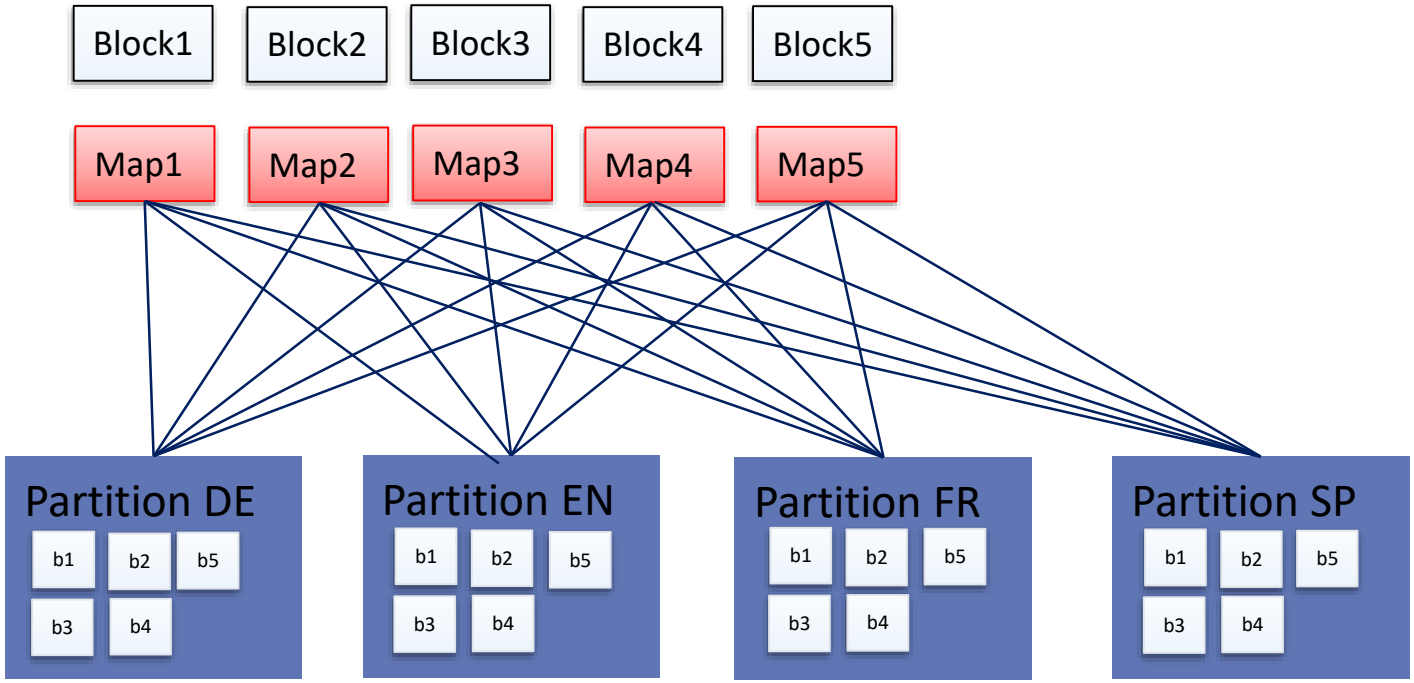
Dynamic partitioning could create millions of partitions for bad partition keys. Parameters exist that restrict the creation of dynamic partitions

```
set hive.exec.dynamic.partition=true;
set hive.exec.dynamic.partition.mode = nonstrict;
set hive.exec.max.dynamic.partitions.pernode=100000;
set hive.exec.max.dynamic.partitions=100000;
set hive.exec.max.created.files=100000;
```

Most of these settings are already enabled with good values

Dynamic Partition Loading

One file per Reducer/Mapper
Standard Load will use Map tasks to write data.
One map task per input block/split.



Small files

- Large number of writers with large number of partitions results in small files
  - Files with 1-10 blocks of data are more efficient for HDFS
  - ORC compression is not very efficient on small files

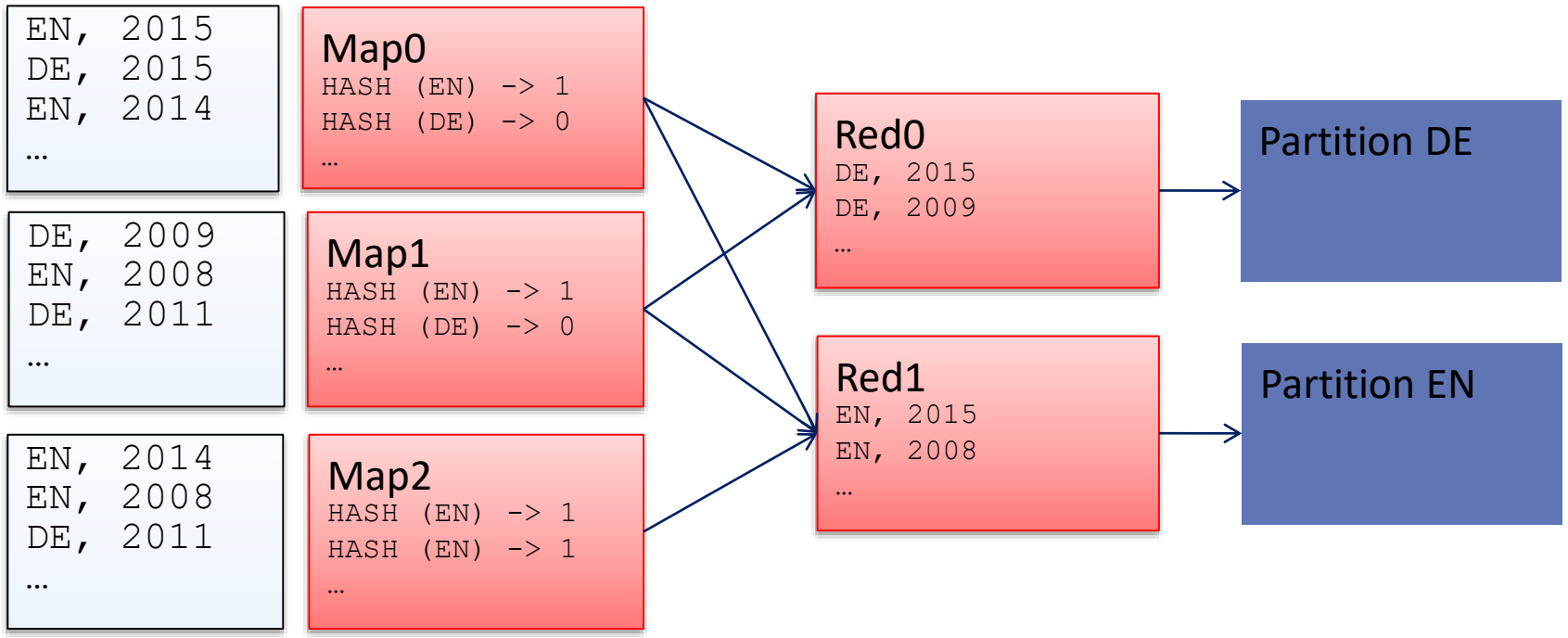
- ORC Writer will keep one Writer object open for each partition he encounters.
  - RAM needed for one stripe in every file / column
  - Too many Writers results in small stripes (down to 5000 rows)

- If you run into memory problems you can increase the task RAM or increase the ORC memory pool percentage
  - `set hive.tez.java.opts="-Xmx3400m";`
  - `set hive.tez.container.size = 4096;`
  - `set hive.exec.orc.memory.pool = 1.0;`

Loading Data Using Distribution

For large number of partitions, load data through reducers.

- One or more reducers associated with a partition through data distribution
- Beware of Hash conflicts (two partitions being mapped to the same reducer by the hash function)



Bucketing

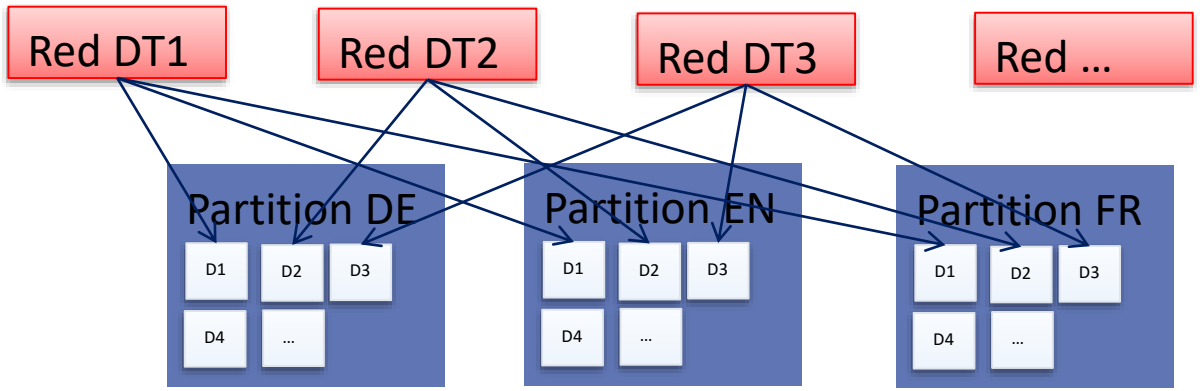
- Hive tables can be bucketed using the CLUSTERED BY keyword
  - One file/reducer per bucket
  - Buckets can be sorted
  - Additional advantages like bucket joins and sampling

- Per default one reducer for each bucket across all partitions
  - Performance problems for large loads with dynamic partitioning
  - ORC Writer memory issues

- Enforce Bucketing and Sorting in Hive
  - Set hive.enforce.sorting=true;
  - set hive.enforce.bucketing=true;

Bucketing Example

```
CREATE TABLE ORC_SALES
  (clientID INT, dt DATE, rev DOUBLE, profit DOUBLE, comment STRING)
  PARTITIONED BY (country STRING)
  CLUSTERED BY dt SORT BY (dt) INTO 31 BUCKETS;
INSERT INTO TABLE orc_sales (country) SELECT * FROM del_sales;
```



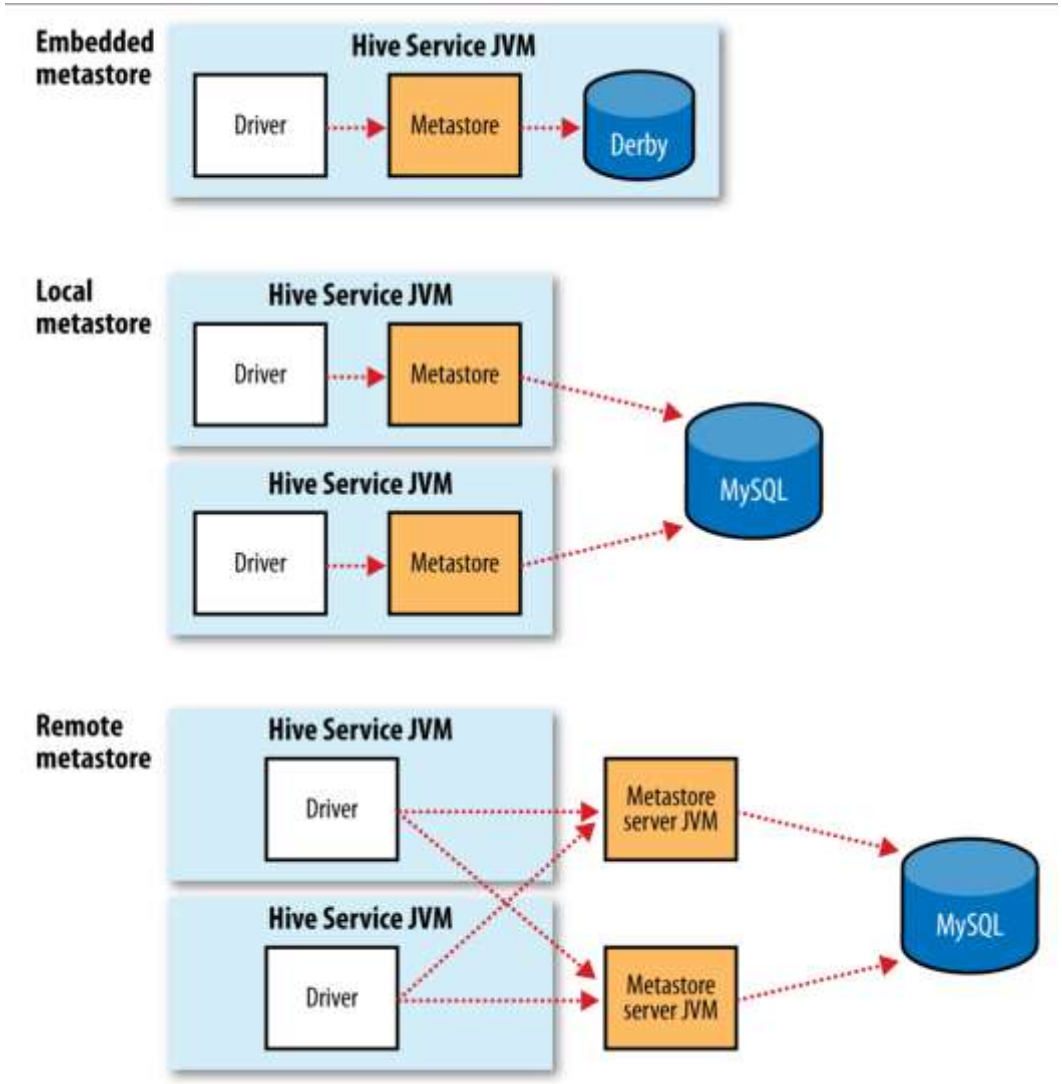
Outline

- Introduction
- Creating Tables
- Data Storage
- Data Hierarchy
- Loading Data
- Metastore
- Summary

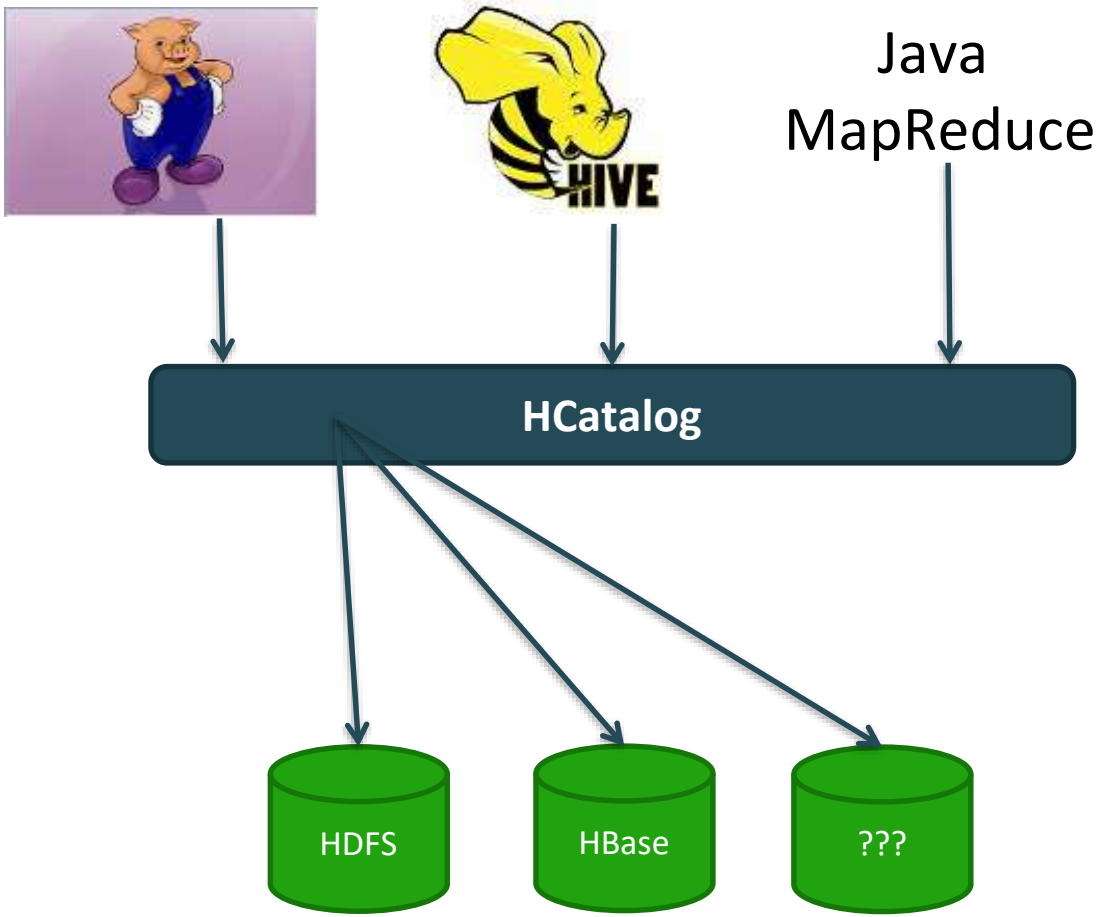
Metastore is the central repository of metadata

- The metastore is the central repository of Hive metadata.
- The metastore is divided into two pieces:
  a service and the backing store for the data.
- By default, the metastore service runs in the same JVM as the Hive service and contains an embedded Derby database instance → embedded metastore configuration
  - one embedded Derby database can access the database files on disk at any one time
  - You can have only one Hive session open at a time that accesses the same metastore!
- The solution to supporting multiple sessions and users is to use a standalone database → local metastore
- Metastore service still runs in the same process as the Hive service but connects to a database running in a separate process

Three different kinds of Metastore



HCatalog in the Ecosystem



HCatalog is a Hive sub-project that provides access to the Metastore

Accessible via command line and REST API

Allows you to define tables using HiveQL DDL syntax

Access those tables from Hive, Impala, MapReduce, Pig, and other tools

Creating Tables in HCatalog

HCatalog uses Hive's DDL syntax

You can specify a single command using the -e option

```
$ hcat -e "CREATE TABLE vendors \
      (id INT, company STRING, email STRING) \
      ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' \
      LOCATION '/dualcore/vendors'"
```

Save longer commands to a text file and use the -f option

```
$ hcat -f createtable.txt
```

Using HCatLoader with Pig

HCatLoader: used to read data from HCatalog managed tables.

```
emp_relation = LOAD 'employees' USING
org.apache.hcatalog.pig.HCatLoader();
```

Using HCatStorer with Pig

HCatStorer: used to write data to HCatalog-managed tables.

```
STORE customer_projection INTO 'customers'
USING
org.apache.hcatalog.pig.HCatStorer();
```

Displaying Metadata in HCatalog

The SHOW TABLES command also shows tables created directly in Hive

```
$ hcat -e 'SHOW TABLES‘

employees
vendors
```

The DESCRIBE command lists the fields in a specified table

Use DESCRIBE FORMATTED instead to see detailed information

```
$ hcat -e 'DESCRIBE vendors'
id       int
company  string
email    string
```

Outline

- Introduction
- Creating Tables
- Data Storage
- Data Hierarchy
- Loading Data
- Metastore
- Summary

Essential Points

- Each table maps to a directory in HDFS
- Table data is stored as one or more files
- Default format: plain text with delimited fields
- The Metastore stores data about the data in an RDBMS
  E.g. Location, column names and types
- Tables are created and managed using the HiveQL Data Definition Language
- HCatalog provides access to the Metastore from tools outside Hive e.g. Pig, MapReduce
- Dynamic Partition loading can lead to problems
  - Normally Optimized Dynamic Sorted Partitioning solves these problems
  - Sometimes manual distribution can be beneficial