Chapter

Apache Hive

Motivation For Hive

Back To Facebook In ~2008

Their reality

- Hadoop used for many business-critical features
- Many analyst excellent at SQL and RDBMS
- Many BI and dashboarding tools integrate with SQL

Their conclusion

- Smooth migration from SQL to Hadoop is needed

SQL For Data Analysis

- **SQL is the lingua franca for data analysts**
  - Everyone knows SQL from school or learns it quickly
- **SQL is powerful at data analysis and exploration**
  - Intuitive to think in terms of tables, rows and columns
  - Convenient syntax

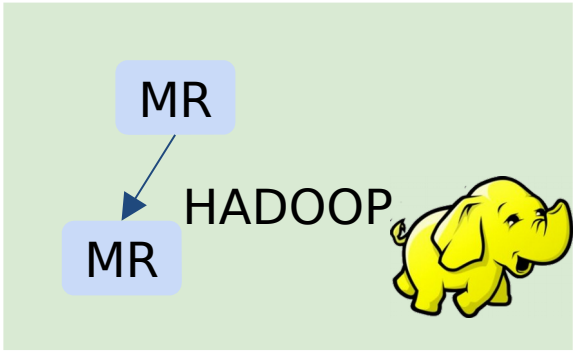
e.g. `JOIN`, `ORDER BY`, `GROUP BY`

  - Many optimizations under the hood

SQL on Hadoop

- **Run SQL-like query to process data on Hadoop**

Results




```
SELECT name,
COUNT(*)
FROM users
GROUP BY name
ORDER BY name;
```

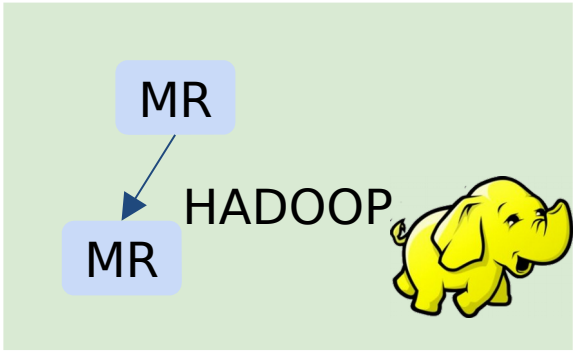
SOME MAGIC

1. Parses query
2. Plans execution
3. Submits MR jobs
4. Monitors the execution

SQL on Hadoop

- **Run SQL-like query to process data on Hadoop**

Results




```
SELECT name,
COUNT(*)
FROM users
GROUP BY name
ORDER BY name;
```

APACHE HIVE

1. Parses query
2. Plans execution
3. Submits MR jobs
4. Monitors the execution

Hive Overview

Basic HiveQL Queries

- **HiveQL is very similar to SQL**
  - Some differences exists, though

```
SELECT state,gender,COUNT(*) AS cnt
FROM uuser
GROUP BY state,gender
ORDER BY cnt DESC;
```

Basic HiveQL Queries

- **HiveQL is very similar to SQL**
  - Some differences exists, though

```
SELECT state,gender,COUNT(*) AS cnt
FROM uuser
GROUP BY state,gender
ORDER BY cnt DESC;
```

This is a Hive table that contains data stored in HDFS

Executing Hive Queries

- **Hive Shell**

```
$ hive
hive> SELECT COUNT(*) FROM uuser;
```

- **Launch queries from files**

```
$ hive -f query.hql
```

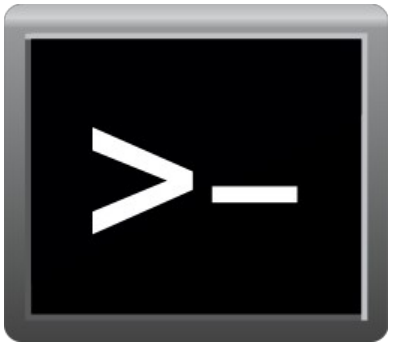
- **One shot command**

```
$ hive -e "SELECT COUNT(*) FROM uuser;"
```

- **Support for multiple other options**
  - e.g. passing parameters

Demo By Instructor

First Queries In Hive Using Hive CLI



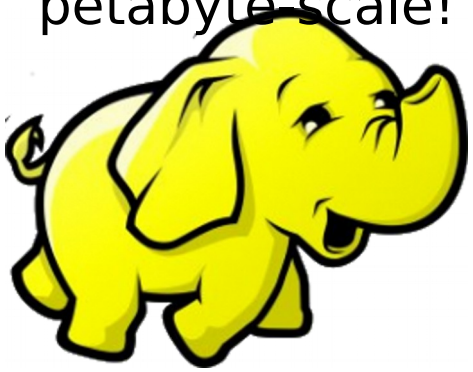
Chapter

Data Analysis In Hive

Common Operators

- SELECT
- WHERE
- GROUP BY
- HAVING
- ORDER BY
- LIMIT
- UNION ALL
- **and more…**

All of them executed on Hadoop at petabyte-scale!



HiveQL Functions

- **Hive comes with plenty of built-in functions**
  - e.g. `COUNT`, `AVG`, `SUM`, `MIN`, `MAX`, `SUBSTRING`
- **Custom UDFs can be implemented**
  - Only in Java

```
SELECT
concat(fname,'',lname),
has_birthday_today(bday),
  if(gender='m','beer','flowers')
FROM uuser
LIMIT 10;
```

Filtering Data

- **Is Elvis a registered user?**
  - Keywords are not case-sensitive, but STRING columns are
- **Use LIKE for String comparison**
  - _ to match a single character
  - % to match a series of characters

```
SELECT fname,lname,state
FROM uuser
WHERE lower(fname) LIKE 'el_is'
AND year(birthdate) = 1935;
```

Subqueries

- **Find tracks with at least 10 streams**

```
SELECT trackCnt.trackId, trackCnt.cnt
FROM ( SELECT trackId, COUNT(*) AS cnt
        FROM stream
        GROUP BY trackId ) trackCnt
        WHERE trackCnt.cnt >= 10;
```

A subquery must be given an alias (name)

Subqueries

- **Find tracks with at least 10 streams**

```
SELECT trackCnt.trackId, trackCnt.cnt
FROM ( SELECT trackId, COUNT(*) AS cnt
        FROM stream
        GROUP BY trackId ) trackCnt
        WHERE trackCnt.cnt >= 10;
```

Having Clause

- **Find tracks with at least 10 streams**

```
SELECT trackId,COUNT(*) AS cnt
FROM stream
GROUP BY trackId
HAVING cnt >= 10;
```

Correlated Subqueries

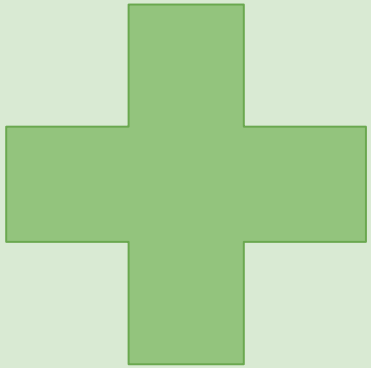
- **Correlated subqueries in `WHERE` clause are NOT supported**
  - e.g "All users who are older than the average age in their state"
  - Workaround is to use JOINs

```
SELECT fname,age
FROM uuser AS u
WHERE age > (
   SELECT avg(age)
  FROM uuser
  WHERE state = u.state
);
```

Workaround With JOIN

- **Correlated subqueries in `WHERE` clause are NOT supported**
  - e.g "All users who are older than the average age in their state"

```
SELECT fname, lname, age
FROM uuser AS u
JOIN (
  SELECT state, AVG(age) as avgage
  FROM uuser
  GROUP BY state
) aa ON u.state = aa.state
WHERE u.age > aa.avgage
```



Numeric Types

Type	Size	Range	Example
TINYINT	1-byte	-128 to 127	13Y
SMALLINT	2-byte	-32,768 to 32,767	100S
INT	4-byte	-2,147,483,648 to 2,147,483,647	200
BIGINT	8-byte	~ $-9*10^{18}$ to $9*10^{18}$	300L
FLOAT	4-byte	Single precision	2.718
DOUBLE	8-byte	Double precision	2.71828182846

Simple Types

Type	Description	Example
TIMESTAMP	Precise time	1354183921
DATE	Date	2014-03-10
STRING	Chain of characters	'JeffKowalsky'
VARCHAR	Fixed-length string	'abcd'
BOOLEAN	True or False	TRUE
BINARY	Raw data	N/A

Collection Types

- **Benefits**
  - Might reflect your data better
  - Make your SQL code and column names cleaner
  - Help you avoid expensive JOINs

Type	Description	Example
ARRAY	list of values	devices[0]
MAP	key-value pairs	version['mobile']
STRUCT	named fields	address.zipcode

Joins in Hive

- **Joining datasets is a common operation in Hive**
  - Also, one of the most expensives
- **Hive supports few types of JOINs**
  - INNER JOIN
  - OUTER JOIN (LEFT, RIGHT and FULL)
  - CROSS JOINS
- **Only equi-joins are supported**

Joins in Hive

- **JOIN syntax (INNER JOIN)**

```
SELECT t.artistname,count(*) as cnt
FROM stream s
JOIN track t ON s.trackId = t.id
GROUP BY t.artistname;
```

  - eg. LEFT OUTER JOIN

Getting Top Results

- **Possible with `ORDER BY` and `LIMIT`**

```
SELECT t.artistname,count(*) as cnt
FROM stream s
JOIN track tON s.trackId = t.id
GROUP BY t.artistname
ORDER BY cntDESC
LIMIT 10;
```

Ugly HiveQL

```
SELECT us.state,COUNT(*) AS total
FROM (
SELECT *
FROM uuseru
JOIN stream s
ON u.id = s.userid
WHERE
DATEDIFF(to_date(from_unixtime(unix_timestamp())),
to_date(u.registrationdate))<= 30
)us
WHERE us.gender= 'M'
GROUP BY us.state;
```

What does this query return?

View Example

```
CREATE VIEW users_from_last_month AS
SELECT *
FROM uuseru
JOIN stream s ON u.id = s.userid WHERE
DATEDIFF(to_date(from_unixtime(unix_timestamp()),
  to_date(u.registrationdate))<= 30;
```

```
SELECT state,COUNT(*) AS total
FROM users_from_last_month
WHERE gender= 'M'
GROUP BY state;
```

Benefits of Views

1. **Simpler queries**
   - Divide query in smaller, more manageable pieces
   - Can be shared by users to construct more complex queries from reusable parts
2. **Save your query and treat it like an input table**
   - It is logical construct - doesn't store any data
   - You can't use it as destination in `INSERT OVERWRITE`
3. **Can serve as an access control mechanisms**
   - Can hide columns and/or specific records

Data Management In Hive

Creating A Table

```
CREATE TABLE IF NOT EXISTS short_stream (
trackid int,
userid int,
duration int
)
COMMENT 'Streams that are shorter than 30 sec'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

Create Table With Complex Types

```
CREATE TABLE IF NOT EXISTS song (
…
genre       ARRAY<STRING>,
lyrics      STRUCT<written:DATE,language:STRING>,
version     MAP<STRING,STRING>
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
COLLECTION ITEMS TERMINATED BY ','
MAP KEYS TERMINATED BY '#'
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

Hive File Formats

- **Multiple file formats are supported**
  - Text
  - SequenceFile

A binary format storing key-value pairs

  - Avro

A binary row-oriented format

  - Parquet, ORC

Binary column-oriented formats

  - You can implement own library for reading your custom file format
- **Files can be compressed using popular compression codec**
- **We will learn more about file formats later**

```
hive> describe formatted track;
OK
# col_name              data_type               comment

id                      int
title                   string
artistname              string

# Detailed Table Information
Database:               default
Owner:                  hdfs
CreateTime:             Sun Apr 10 08:13:45 EDT 2016
LastAccessTime:         UNKNOWN
Protect Mode:           None
Retention:              0
Location:               hdfs://ip-172-31-26-17.eu-west-1.compute.internal:8020/training/data/track
Table Type:             EXTERNAL_TABLE
Table Parameters:
        COLUMN_STATS_ACCURATE   false
        EXTERNAL                TRUE
        numFiles                1
        numRows                 -1
        rawDataSize             -1
        totalSize               238792
        transient_lastDdlTime   1460290425

# Storage Information
SerDe Library:          org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
InputFormat:            org.apache.hadoop.mapred.TextInputFormat
OutputFormat:           org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
Compressed:             No
Num Buckets:            -1
Bucket Columns:         []
Sort Columns:           []
Storage Desc Params:
        field.delim             \t
        serialization.format    \t
Time taken: 0.377 seconds, Fetched: 35 row(s)
```

```
hive> describe formatted track;
OK
# col_name              data_type               comment

id                      int
title                   string
artistname              string

# Detailed Table Information
Database:               default
Owner:                  hdfs
CreateTime:             Sun Apr 10 08:13:45 EDT 2016
LastAccessTime:         UNKNOWN
Protect Mode:           None
Retention:              0
Location:               hdfs://ip-172-31-26-17.eu-west-1.compute.internal:8020/training/data/track
Table Type:             EXTERNAL_TABLE
Table Parameters:
        COLUMN_STATS_ACCURATE   false
        EXTERNAL                TRUE
        numFiles                1
        numRows                 -1
        rawDataSize             -1
        totalSize               238792
        transient_lastDdlTime   1460290425

# Storage Information
SerDe Library:          org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
InputFormat:            org.apache.hadoop.mapred.TextInputFormat
OutputFormat:           org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
Compressed:             No
Num Buckets:            -1
Bucket Columns:         []
Sort Columns:           []
Storage Desc Params:
        field.delim             \t
        serialization.format    \t
Time taken: 0.377 seconds, Fetched: 35 row(s)
```

```
hive> describe formatted track;
OK
# col_name              data_type               comment

id                      int
title                   string
artistname              string

# Detailed Table Information
Database:               default
Owner:                  hdfs
CreateTime:             Sun Apr 10 08:13:45 EDT 2016
LastAccessTime:         UNKNOWN
Protect Mode:           None
Retention:              0
Location:               hdfs://ip-172-31-26-17.eu-west-1.compute.internal:8020/training/data/track
Table Type:             EXTERNAL_TABLE
Table Parameters:
        COLUMN_STATS_ACCURATE   false
        EXTERNAL                TRUE
        numFiles                1
        numRows                 -1
        rawDataSize             -1
        totalSize               238792
        transient_lastDdlTime   1460290425

# Storage Information
SerDe Library:          org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
InputFormat:            org.apache.hadoop.mapred.TextInputFormat
OutputFormat:           org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
Compressed:             No
Num Buckets:            -1
Bucket Columns:         []
Sort Columns:           []
Storage Desc Params:
        field.delim             \t
        serialization.format    \t
Time taken: 0.377 seconds, Fetched: 35 row(s)
```

Schema

Ownership and datetimes

```
hive> describe formatted track;
OK
# col_name              data_type               comment

id                      int
title                   string
artistname              string

# Detailed Table Information
Database:               default
Owner:                  hdfs
CreateTime:             Sun Apr 10 08:13:45 EDT 2016
LastAccessTime:         UNKNOWN
Protect Mode:           None
Retention:              0
Location:               hdfs://ip-172-31-26-17.eu-west-1.compute.internal:8020/training/data/track
Table Type:             EXTERNAL_TABLE
Table Parameters:
        COLUMN_STATS_ACCURATE   false
        EXTERNAL                TRUE
        numFiles                1
        numRows                 -1
        rawDataSize             -1
        totalSize               238792
        transient_lastDdlTime   1460290425

# Storage Information
SerDe Library:          org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
InputFormat:            org.apache.hadoop.mapred.TextInputFormat
OutputFormat:           org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
Compressed:             No
Num Buckets:            -1
Bucket Columns:         []
Sort Columns:           []
Storage Desc Params:
        field.delim             \t
        serialization.format    \t
Time taken: 0.377 seconds, Fetched: 35 row(s)
```

```
hive> describe formatted track;
OK
# col_name              data_type               comment

id                      int
title                   string
artistname              string

# Detailed Table Information
Database:               default
Owner:                  hdfs
CreateTime:             Sun Apr 10 08:13:45 EDT 2016
LastAccessTime:         UNKNOWN
Protect Mode:           None
Retention:              0
Location:               hdfs://ip-172-31-26-17.eu-west-1.compute.internal:8020/training/data/track
Table Type:             EXTERNAL_TABLE
Table Parameters:
        COLUMN_STATS_ACCURATE   false
        EXTERNAL                TRUE
        numFiles                1
        numRows                 -1
        rawDataSize             -1
        totalSize               238792
        transient_lastDdlTime   1460290425

# Storage Information
SerDe Library:          org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
InputFormat:            org.apache.hadoop.mapred.TextInputFormat
OutputFormat:           org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
Compressed:             No
Num Buckets:            -1
Bucket Columns:         []
Sort Columns:           []
Storage Desc Params:
        field.delim             \t
        serialization.format    \t
Time taken: 0.377 seconds, Fetched: 35 row(s)
```

Stats about data in HDFS

Format of data in HDFS

Hive Tables

- **Data for Hive tables is usually stored in HDFS**
  - Typically, a table corresponds to a single HDFS directory
  - The directory can contain multiple files
- **Metadata gives context to this data**
  - Raw data is mapped to rows and columns
  - Columns are named and have specific type
- **Tables are grouped into Hive databases**

Managed Table

- **Hive controls life cycle of the data inside the managed table**
  - `DROP` deletes both metadata and actual data
  - Less convenient for sharing this data with other tools

If Hive analyst drops this table, others won't be able to use it

External Table

- **Hive doesn’t own the data in the external table**
  - DROP removes only table metadata from Hive
- **Easier sharing of data with other tools**
- **External tables are recommended**

Create External Table Example

```
CREATE EXTERNAL TABLE IF NOT EXISTS stream (
    userid  INT COMMENT 'user id',
    trackid    INT COMMENT 'song id',
    ts      TIMESTAMP COMMENT 'time of play'
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LOCATION '/training/data/stream ;
```

Inserting Data Into Table

```
INSERT [OVERWRITE] TABLE short_stream
  SELECT trackid,userid,duration
FROM stream
WHERE duration < 30;
```

Inserting Data Into Directory

```
INSERT [OVERWRITE] DIRECTORY '/data/short_stream '
SELECT trackid,userid,duration
FROM stream
WHERE duration < 30;
```

One-Hop Multi-Table Insert

```
FROM uuser
   INSERT OVERWRITE TABLE maleuser
      SELECT fname,lname,state
WHERE gender= 'M'
   INSERT OVERWRITE TABLE femaleuser
      SELECT fname,lname,state
WHERE gender= 'F';
```

Create Table As Select (CTAS)

- It's possible to create a table dynamically, based on the output of a query

```
CREATE TABLE topTenTrackIds AS
SELECT trackid, count(*) AS cnt
FROM stream
GROUP BY trackid
ORDER BY cnt DESC
LIMIT 10;
```

Authorization

- **A set of privileges can be granted e.g.**
  - `SELECT` – Read access to an object
  - `INSERT` – Add data to an object (table)
  - `DELETE` – Delete data in an object (table)
  - `CREATE` – Create new objects
  - `ALTER` – Modify objects
  - `DROP` – Remove objects
  - `ALL PRIVILEGES` – gives all privileges
- **The privileges might apply to databases, table and views**

Typical Authorization Patterns

- **Users are grouped into groups**
  - Alice belongs to the engineers group
- **Roles are assigned to groups (or a user)**
  - etl_engineer role is assigned to the engineers group
- **Roles have required privileges**
  - etl_engineer is granted the SELECT and INSERT privileges on the etl database
- **It can be achieved natively in Hive, or through tools like Sentry and Ranger**
  - Sentry (CDH) or Ranger (HDP) are preferred

Hive Architecture

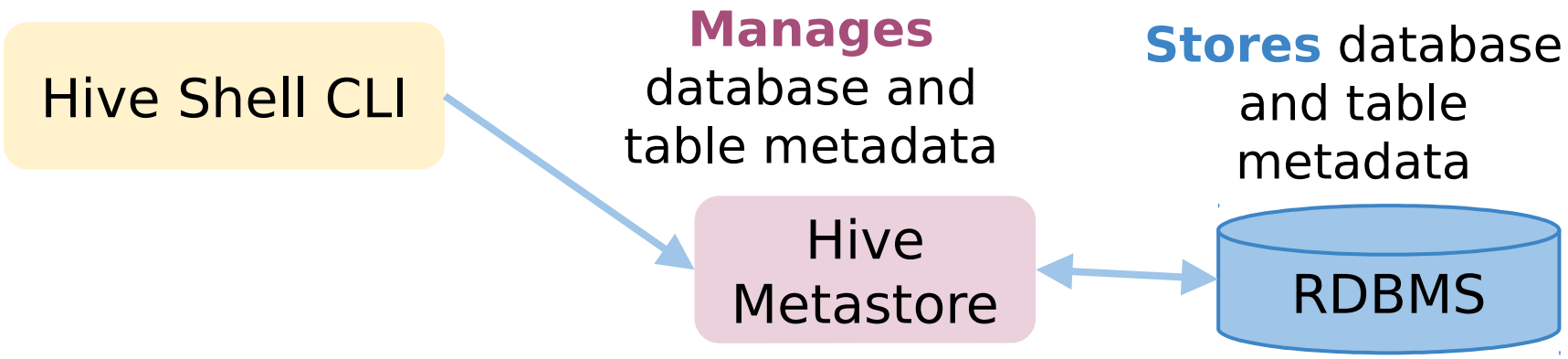
Hive Metadata

- **When processing a table, Hive needs to know**
  - What is the schema (columns and their types)
  - Where data for a table is located in HDFS
  - What is the format of the data (e.g. binary, text)
- **Where is this information available?**

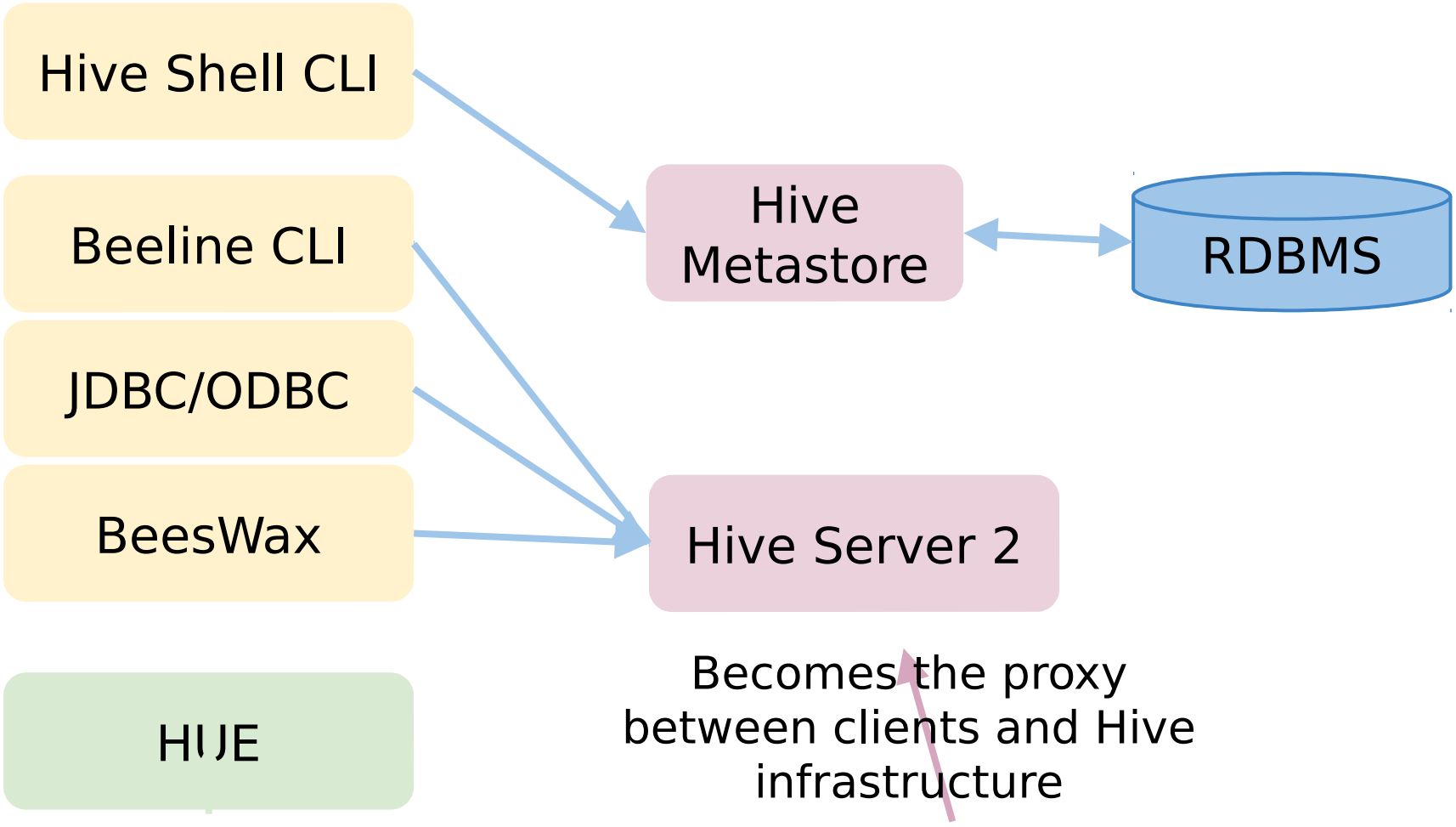
Hive Metastore

- **The master daemon in Hive**
- **Manages metadata about databases and tables in Hive**
- **Stores the metadata in RDBMS e.g. MySQL, Postgres, Oracle**
  - Metadata is relatively small
- **Enables the work on a table abstraction**
  - Knows how to map a table to a dataset in HDFS

Hive Metastore



Hive Server 2



More About Hive Server 2

- **A proxy between clients and Hive infrastructure**
  - HUE
  - Beeline CLI
  - BI tools like Tableau, Pentaho, QlikView
- **Does the heavy work**
  - Talks to Hive Metastore
  - Submits the queries
  - Contains Hive libraries, connectors etc.
- **Increases security**
  - Integrated with Kerberos
  - A single point of credentials to Hive metastore

Exercise

Create a Hive Database and Table
Run First Queries

Hive vs. RDBMS

Hive And RDBMs At Spotify

- **Hive**
  - Calculation of KPIs
  - Large-scale ad-hoc analysis for business
  - Integration with BI tool for data exploration
- **RDBMS**
  - Powering home-grown dashboarding solutions
  - Integrated into process of buying Spotify gift card
- **DWH**
  - No commercial data-warehouses at all…

Hive vs RDBMS

	Hive	RDBMS
Dialect	HiveQL	SQL
Execution Engine	MapReduce, Tez, Spark	Developed during last decades
Record Level CRUD	Experimental	Yes
Latency	High	Low
Transactions	At row-level	Yes
Indexes	Limited support	Supported
Scale	Petabytes	Terabytes

Basic Text Functions

Return type	Name	Returns
String	concat(String A, String B …)	Concatenated strings
Int	length(String A)	Length of the string
String	lower(String A)	All characters to lowercase
String	printf(String format, Obj… args)	Input formatted according to printf-style format strings
String	substr(String A, int start, int len)	Substring of Input
String	trim (String A)	Input stripped of leading and trailing spaces
String	upper(String A)	All characters to uppercase

Other Useful Functions

Return type	Name	Returns
Int	year|month|day(String A)	Year, month or day part of timestamp string
Bigint	unix_timestamp()	Current unix timestamp in seconds
String	from_unixtime(Bigint time, String format)	String representing the timestamp
String	to_date(String timestamp)	Date part of timestamp
Type	cast(expr as <type>)	Expr converted to <type> or Null
String	if(boolean test, T v1, T v2)	v1 when test is True, v2 otherwise
Array	split(string str, string pat)	Split str around pat (part is a regular expression)

Chapter

Advanced Hive

Hive Execution Plans

Reading Data As It Is

- How is this query executed by map and reduce tasks?

```
SELECT * FROM uuser LIMIT 10;
```

Reading Data As It Is

- **No need for any map and reduce tasks!**

```
SELECT * FROM uuser LIMIT 10;
```

Aggregating Data

- How many James who use do live in each state?

```
SELECT state,COUNT(*) FROM uuser
WHERE upper(fname) LIKE 'JAMES'
GROUP BY state;
```

- How many MapReduce jobs are run as a result of this query?

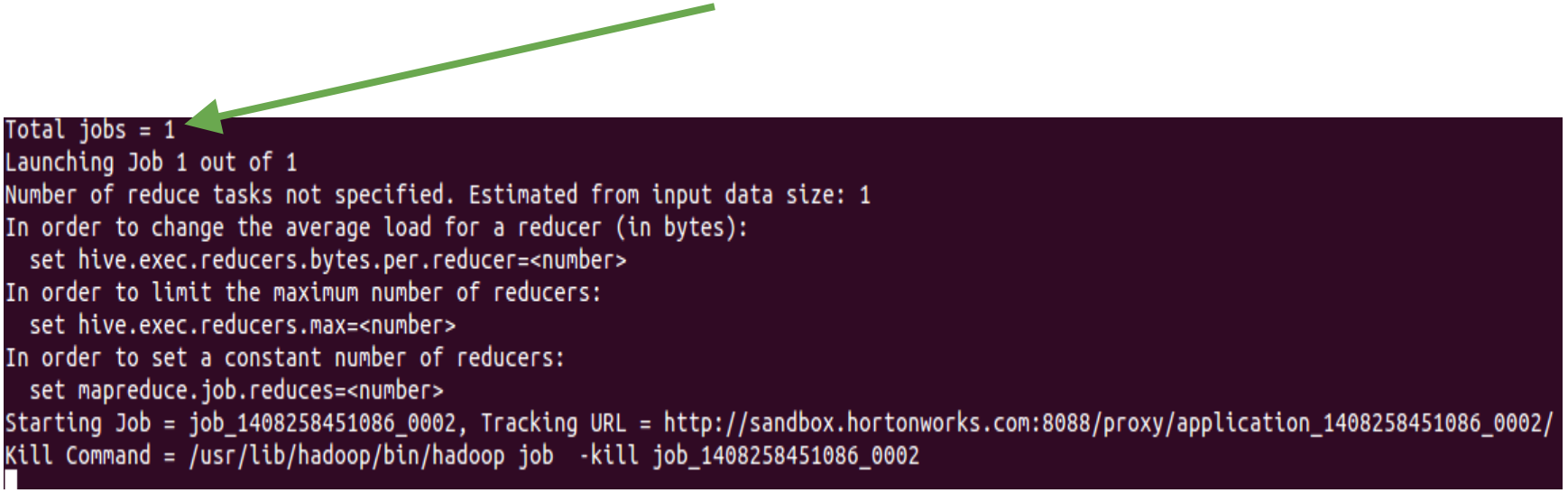
1st Method - Guessing

- **Projecting**
  - Pick the state column
- **Filtering**
  - Retain only users who are JAMES
- **Grouping**
  - Group by state
- **Aggregating**
  - Sum all JAMES for each state

```
SELECT state,COUNT(*)
FROM uusers
WHERE
 upper(fname) LIKE 'JAMES'
GROUP BY state;
```

2nd Method - Running And Checking

One MapReduce job



3rd Method - Using EXPLAIN

```
EXPLAIN SELECT state,COUNT(*) FROM uusers
WHERE upper(fname) LIKE 'JAMES'
GROUP BY state;
```

Three Main Sections In Explain

```
ABSTRACT SYNTAX TREE:
  (TOK_QUERY (TOK_FROM (TOK_TABREF (TOK_TABNAME
users))) ...

STAGE DEPENDENCIES:
  Stage-1 is a root stage
  Stage-0 is a root stage

STAGE PLANS:
  Stage: Stage-1
    ... (next slide) ... .
```

Stage Plan

```
Stage:Stage-1
  Map Reduce
   Alias -> Map Operator Tree:
    users
     TableScan
      alias:uuser
       Filter Operator
        predicate:
          expr:(upper(fname) like 'JAMES')
          type:boolean
         Select Operator
          expressions:
              expr:state
              type:string
            outputColumnNames:state
    Reduce Operator Tree:
... (next slide)...
```

Reduce Operator

Reducing (shuffle) is often triggered by phases like

- GROUP BY
- JOIN
- ORDER BY

```
… (previous slide)…

Reduce Operator Tree:
  Group By Operator
    aggregations:
      expr: count(VALUE._col0)

… (next slide)…
```

Fetch Stage Plan

- **Stage-0 is Fetch**
- **Fetch query results and display them on console**
  - -1 means no limit

```
... (previous slide)...

  Stage: Stage-0
    Fetch Operator
      limit: -1
```

Partitioning Tables

Back in

- **Daily reports for artists**

Why does it run so slooooooow ??

```
SELECT t.artistname,COUNT(*)
FROM stream s
JOIN track t ON s.trackid = t.id
WHERE
to_date(s.ts)= to_date('2013-12-02 00:00:00')
GROUP BY t.artistname;
```

Scanning Data

- **By default Hive reads all files from the table directory in HDFS, regardless of filters in `WHERE` clause**

```
/app/hive/warehouse/stream s/stream 01.tsv
                    stream 02.tsv
                    stream 03.tsv
                    stream 04.tsv
                    …
```

All files are read from disk each time and filtered later on

Partitioning

Recommended solutions

- **Split table into independent parts (partitions)**
  - Each part corresponds to subset of our data

e.g. daily partitions

- **Avoid unnecessary read of irrelevant data from HDFS**
  - Read only data that corresponds to your partition

Partitioned Table

```
CREATE TABLE stream_daily (
ts        TIMESTAMP,
host         STRING,
userid    INT,
trackid   INT,
duration  INT
)
COMMENT 'streamed songs'
PARTITIONED BY (dt STRING)
ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\t';
```

Populating A Single Partition

- **Every new partition ends up as new directory on HDFS**

```
INSERT OVERWRITE TABLE stream_daily PARTITION (dt="2013-01-01")
SELECT
ts,
host,
userid,
trackid,
duration
FROM stream
WHERE from_unixtime(unix_timestamp(ts),"yyyy-MM-dd")= "2013-01-01"
```

Dynamic Partitioning

- **Creates partition dynamically based on the value of the last column(s) in a query**

```
SET hive.exec.dynamic.partition.mode=nonstrict;
INSERT OVERWRITE TABLE stream_daily PARTITION (dt)
SELECT
ts,
host,
userid,
trackid,
duration,
from_unixtime(unix_timestamp(ts),"yyyy-MM-dd")
FROM stream
```

Showing Partitions

```
$ hive -e "show partitions stream_daily"
dt=2013-01-07
dt=2013-01-10
dt=2013-01-12
…
```

```
$ hdfs dfs -ls /apps/hive/warehouse/stream_daily

drwxrwxrwx   -tigerhdfs      0 2016-04-12 10:00
/apps/hive/warehouse/stream_daily/dt=2013-01-07
drwxrwxrwx   -tigerhdfs      0 2016-04-12 10:00
/apps/hive/warehouse/stream_daily/dt=2013-01-10
drwxrwxrwx   -tigerhdfs      0 2016-04-12 10:00
/apps/hive/warehouse/stream_daily/dt=2013-01-12
```

Querying Partitioned Table

- **Partition keys behave like regular columns**

```
SELECT artistname,COUNT(*)
FROM stream_daily sd
JOIN track t ON sd.trackid = t.id
WHERE dt='2013-01-12'
GROUP BY artistname;
```

Dynamic Partitioning Settings

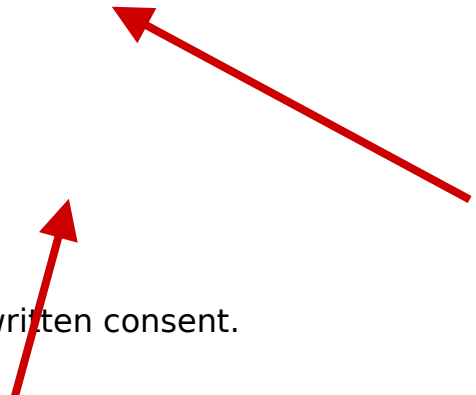
- **Couple of properties that safeguard partitioning process**

Property	Default	Description
hive.exec.dynamic.partition	false	Turn on/off dynamic partitioning
hive.exec.dynamic.partition.mode	strict	nonstrict allows for all dynamic partitions
hive.exec.max.dynamic.partitions.pernode	100	Max number of dynamic partitions that can be created by single mapper or reducer
hive.exec.max.dynamic.partitions	1000	Max total number of dynamic partitions
hive.exec.max.created.files	100000	Max number of files created globally

Loading Data Into Partition

```
LOAD DATA INPATH 'files/recent_streams.tsv
INTO TABLE stream_daily
PARTITION(dt='2013-12-01');
```

You need to ensure that loaded data matches the partition



Adding Partitions

```
ALTER TABLE stream_daily ADD PARTITION (dt='2013-12-01')
LOCATION '/data/plays/2013-12-01';
```

- Directory doesn't have to exist
- Data can be shared between multiple applications

Renaming And Dropping Partitions

```
ALTER TABLE stream_daily PARTITION(dt='2013-12-01')
RENAME TO PARTITION(dt='2013-12-02');
```

```
ALTER TABLE stream_daily DROP IF EXISTS
PARTITION(dt='2013-12-01');
```

Strict Mode

Meet your Hive cluster guard

- **Prohibits querying the partitioned tables without WHERE clause that filters on partitions**

```
hive> set hive.mapred.mode=strict;
```

```
FAILED: SemanticException [Error 10041]: No partition predicate
found for Alias "stream_daily" Table "stream_daily"
```

Potentially Inefficient Query

- **What can be dangerous with this query?**

```
SELECT ts,userid
FROM stream_daily
WHERE date > "2013-12-10"
ORDER BY ts;
```

Other STRICT Mode Safeguards

- **Forbids queries with ORDER BY, but without LIMIT**

```
SELECT ts,userid
FROM stream_daily
WHERE date > "20131210"
ORDER BY ts;
```

FAILED: SemanticException 4:9 **In strict mode, if ORDER BY is specified, LIMIT must also be specified.** Error encountered near token 'time'

Bucketing

Bucketing

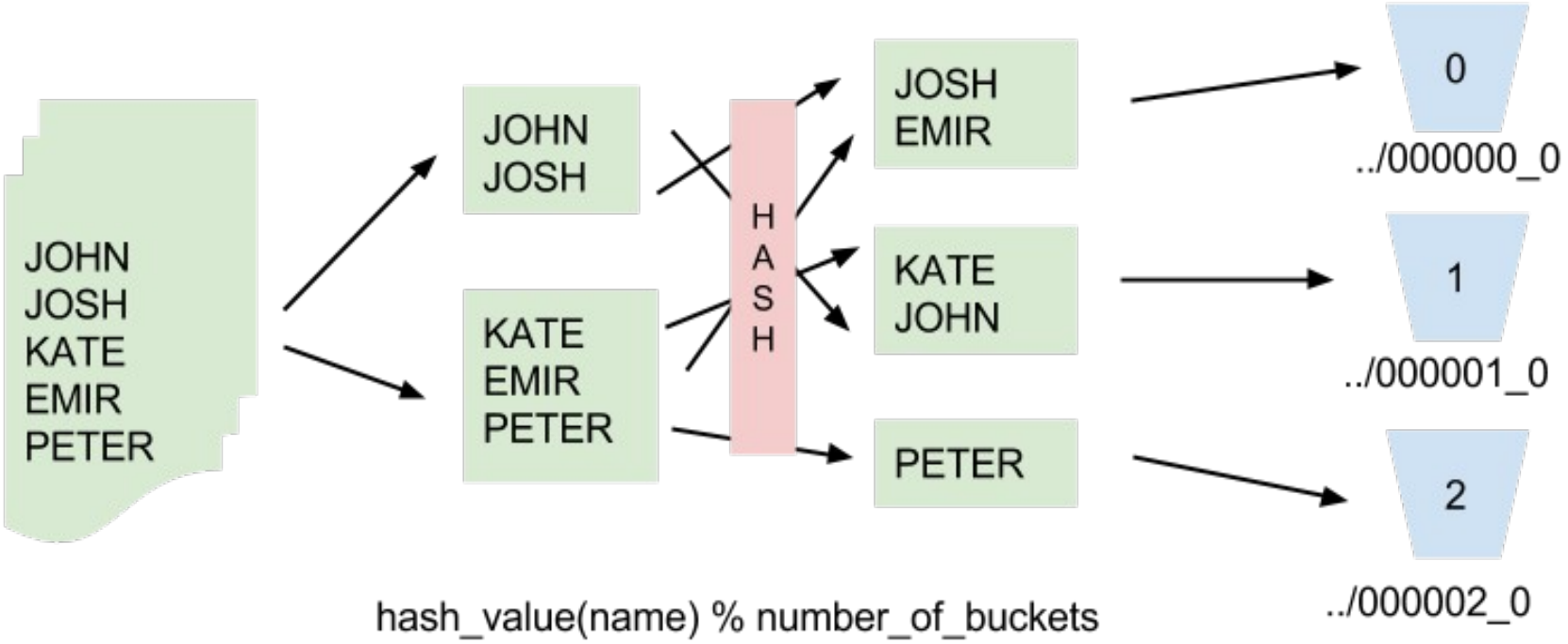
- **One more way of subdividing data**
- **Spreads data into a fixed number of buckets**
  1. Calculate the hash code for the value of the bucketed column
  2. Assign the bucket based on the hash code

Creating A Bucketed Table

```
SET hive.enforce.bucketing = true;

CREATE TABLE rock.stream_bucketed (
timestamp STRING,
host STRING,
userId INT,
…
)
PARTITIONED BY (date STRING)
CLUSTERED BY (userId) INTO 16 BUCKETS;
```

Bucketing Process



Benefits Of Bucketing

- **Easy “sampling”**
  - Just process subset of data from particular bucket(s)
- **Improves JOINs if all tables are bucketed on the join column**
  - Matches can be found in the corresponding buckets

Bucket Sampling

- **Example**
  - The stream_bucketed table is divided into 16 buckets
- **Return the 3rd bucket**

```
SELECT *
FROM stream_bucketed TABLESAMPLE(BUCKET 3 OUT OF 16 ON
userid)
```

- **Return 3rd and 11th bucket**

```
SELECT *
FROM stream_bucketed TABLESAMPLE(BUCKET 3 OUT OF 8 ON userid)
```

- **Return a half of the 3rd bucket**

```
SELECT *
FROM stream_bucketed TABLESAMPLE(BUCKET 3 OUT OF 32 ON userid)
```

Table Sampling

- **In some scenarios, TABLESAMPLE scans the whole table**

1. When table is not bucketed

```
SELECT *
FROM stream TABLESAMPLE(BUCKET 3 OUT OF 16 ON userid);
```

2. When non-buckled column is used

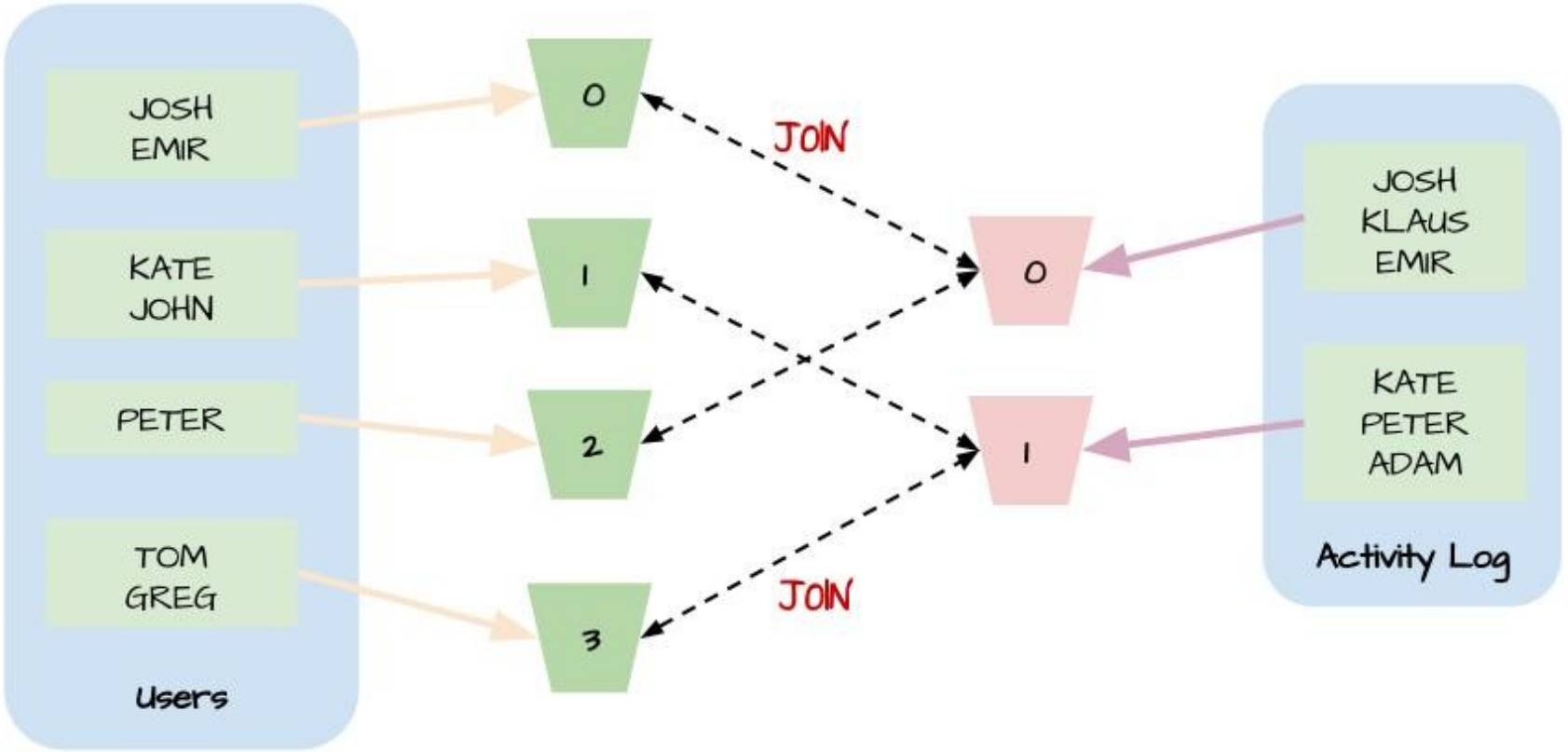
```
SELECT *
FROM stream_bucketed TABLESAMPLE(BUCKET 3 OUT OF 16 ON
gender);
```

3. Or just take random sample

```
SELECT *
FROM stream_bucketed TABLESAMPLE(BUCKET 3 OUT OF 16 ON
rand());
```

Bucketing With JOINs

- When all tables are be bucketed on the join column



Optimizations

Map-Side Join Optimization

- **Can be used when one table is small enough**
  - Size of the “small” table is defined by `hive.mapjoin.smalltable.filesize`

Defaults to ~24 MB

- **Small table is broadcasted to each task and cached in memory**
- **Map-side join is more efficient than reduce-side join**
  - But they have a scalability bottleneck
- **Map-side joins are enabled by setting `hive.auto.convert.join` to `true`**

Join Optimizations

- **Use a common join key when joining three or more tables**
  - Hive will join all sets in a single MapReduce job

```
SELECT sp.user_id, s.title, l.lyrics
FROM songs s
JOIN song_plays sp ON s.id = sp.song_id
JOIN lyrics l ON s.id = l.song_id
WHERE s.artist = 'Metallica';
```

Number of Mappers and Reducers

- **Number of Mappers depends on**
  - The size of the data (the number of HDFS blocks)
  - The InputFormat used
  - Compression

e.g. a text file compressed by gzip must be processed by a single task

  - A few configuration settings e.g. split size
- **Number of Reducers depends on**
  - Operation being performed on the data
  - Size of the input data
- **Number of Reducers can be tuned manually**
  - hive.exec.reducers.bytes.per.reducer (defaults to 1GB)

Number of Mappers and Reducers

- **Balance is required**
  - Too many tasks impose too much overhead
  - Too few tasks do not utilize parallelism of the cluster
- **Analyze intermediate data**
  - Cross join can yield a lot of intermediate data
  - Filtering or sampling can yield little intermediate data
- **Experiment with different number of tasks**
  - Benchmarking is complicated though

Hive SerDe

- **SerDe (SerializerDeserializer) are used to translate records into rows in Hive tables and the other way around**
- **Hive comes with a lot of predefined SerDes**
  - RegexSerDe
  - CSVSerDe, TSVSerDe
  - JSONSerDe
  - AvroSerDe

Chapter

Extending Hive

Writing UDF

- **UDFs are custom user defined functions**
  - They are used as other built-in Hive functions
- **Currently you can use only Java to write UDFs**
- **We have three types of UDFs**
  - UDF - regular functions
  - UDAF - aggregate functions
  - UDTF - table generating functions

```java
package com.training.hive;

import org.apache.hadoop.hive.ql.exec.Description;
import org.apache.hadoop.hive.ql.exec.UDF;
import java.util.Calendar;
import java.util.Date;
import java.text.SimpleDateFormat;

@Description(
  name="GetAge UDF",
  value="_FUNC_(date) returns age of a person",
  extended="SELECT _FUNC_(date) from foo limit 1;"
  )
public class GetAgeUDF extends UDF {

    private SimpleDateFormat df;

    public GetAgeUDF() {
        df = new SimpleDateFormat("yyyy-MM-dd");
    }

   public String evaluate(String bday){
       Date date = null;
       try {
           date = df.parse(bday);
       } catch (Exception ex) {
           return null;
       }
       return this.evaluate(date);
   }

```

GetAgeUDF.java

```
32   public String evaluate(Date bday) {
33     if(bday == null) return null;
34     Calendar birthDate = Calendar.getInstance();
35     birthDate.setTime(bday);
36     Calendar today = Calendar.getInstance();
37     Integer age = today.get(Calendar.YEAR) - birthDate.get(Calendar.YEAR);
38     if (today.get(Calendar.MONTH) < birthDate.get(Calendar.MONTH)) {
39           age--;
40     } else if (today.get(Calendar.MONTH) == birthDate.get(Calendar.MONTH)
41                 && today.get(Calendar.DAY_OF_MONTH) < birthDate.get(Calendar.DAY_OF_MONTH)) {
42           age--;
43                 }
44     return age.toString();
45
46   }
47 }
```

Writing UDF

1. **Package class with your UDF to JAR file**
2. **Add a JAR file to classpath**

```
hive> ADD JAR /home/training/hive/GetAgeUDF-1.0.jar;
```

```
hive> CREATE TEMPORARY FUNCTION getAge
    > AS 'com.training.hive.GetAgeUDF';
```

Exercise

Hive Optimizations And Extensions

http://bit.ly/1puQBks

Pages 12 - 19

Quiz

Hive - Advanced Concepts

http://bit.ly/1crLGcU

Q&A

Chapter

Backup

Chapter

Fast SQL

Hive Adoption

- **In 2008, Hive allows us to run SQL-like queries on unimaginable (at that time) amounts of data!**
- **Hive has became the standard for SQL on Hadoop**
- **Due to the design, Hive queries runs minutes or hours**
  - Historically executed as MapReduce jobs
  - Relatively inefficient query optimizers

Ad-Hoc Analysis With Hive (In 2013)

Alternative SQL-like Solutions

- **None of them use MapReduce**
  - Cloudera Impala
  - Facebook Presto
  - Spark SQL
  - Hive on Tez
  - Hive on Spark
  - Others

e.g. Apache Drill

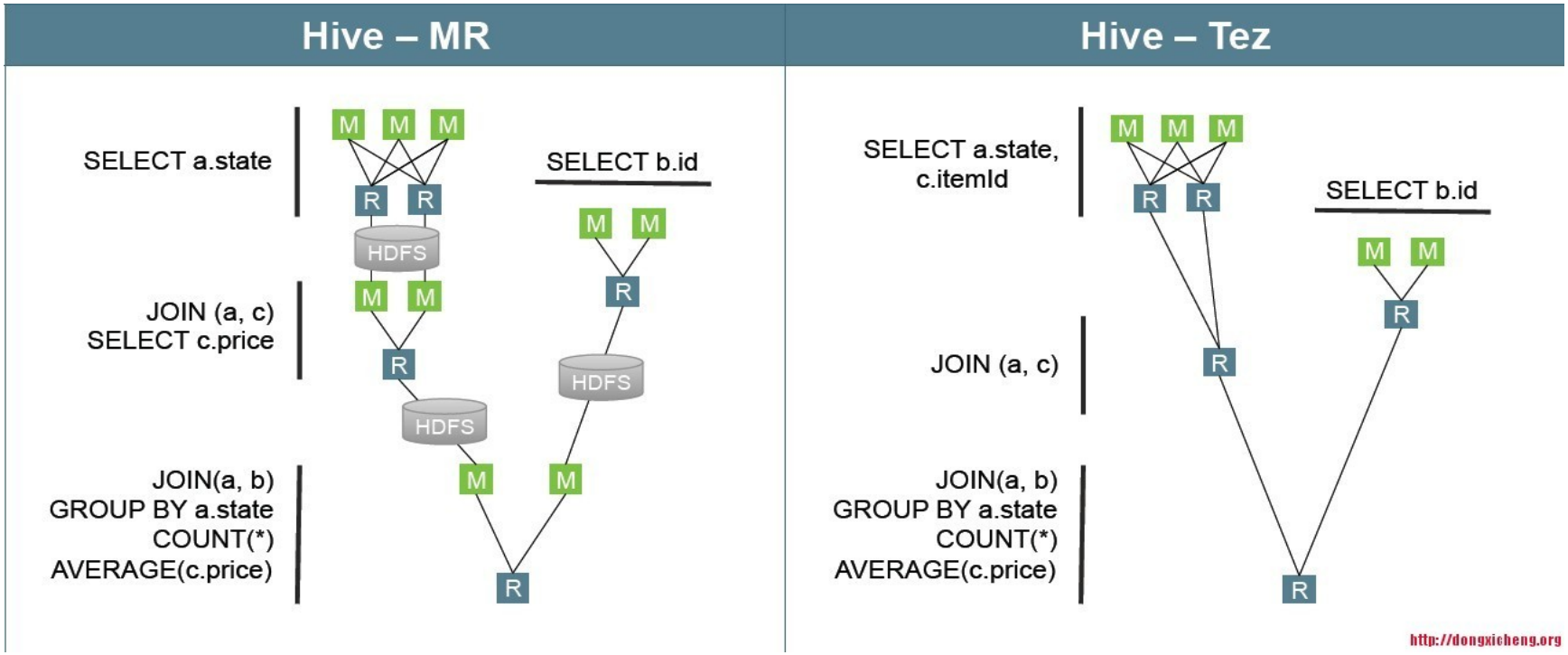
Apache Tez

Tez

- **Efficient execution engine**
  - Faster than MapReduce
- **Can be used by existing frameworks e.g. Hive, Pig, Scalding**
  - SET hive.execution.engine=[tez,mr,spark]
- **Provides low-level operators**
  - You don’t implement own jobs using Tez API
  - The frameworks like Hive use Tez API under the hood

Tez vs. MapReduce

```
SELECT a.state, COUNT(*), AVERAGE(c.price)
FROM a JOIN b ON (a.id = b.id) JOIN c ON (a.itemId = c.itemId)
GROUP BY a.state
```

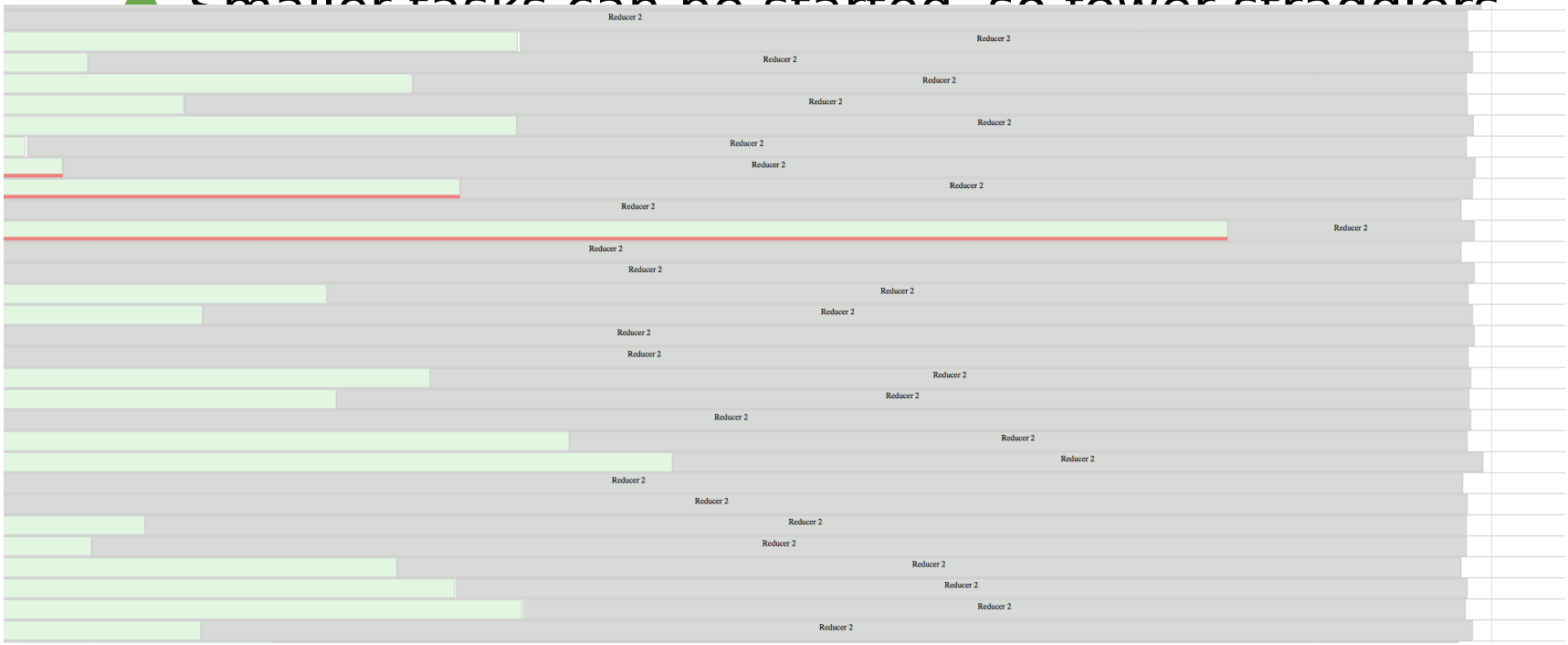


Natural DAGs In Tez

- **No intermediate data written to HDFS (replication 3x)**
- **No need for “empty” map tasks to reshuffle data**
- **No time spent in a queue to start a next MapReduce job**

Reusing Containers

- **Container reuse**
  - Less time spent negotiating with the Resource Manager
  - Smaller tasks can be started, so fewer stragglers



Top Three Users

	Hive on MapReduce on Avro	Hive on Tez on Avro
Plan	2 MapReduce jobs	Map => Reduce => Reduce
Wallclock Time (sec)	353	197
Improvement		1.8x

Top Three Users - On A Busy Cluster

	Hive on MapReduce on Avro	Hive on Tez on Avro
Plan	2 MapReduce jobs	Map => Reduce => Reduce
Wallclock Time (sec)	576	183
Improvement		3.14x

The Biggest Polish Fan of Timbuktu

	Hive on MapReduce on ORC ZLIB	Hive on Tez on ORC ZLIB	Hive on Tez on ORC Snappy
Plan	6 MapReduce jobs	Map => Map => Map => Reduce => Reduce	Map => Map => Map => Reduce => Reduce
Wallclock Time (sec)	519	259	209
Improvement		2x	2.5x

The Biggest Polish Fan of Timbuktu

- **~25TB of data on 690-node cluster**
- **Hive on MapReduce - ~6h of computation**
  - Avro
  - Many default settings
- **Hive on Tez - 10 minutes**
  - ORC
  - Join optimizations
  - No JVM Garbage Collection
  - A few minor tricks
- **Find more:**

  A perfect Hive query for a perfect meeting (Hadoop Summit 2014)

Other SQL-on-Hadoop Alternatives

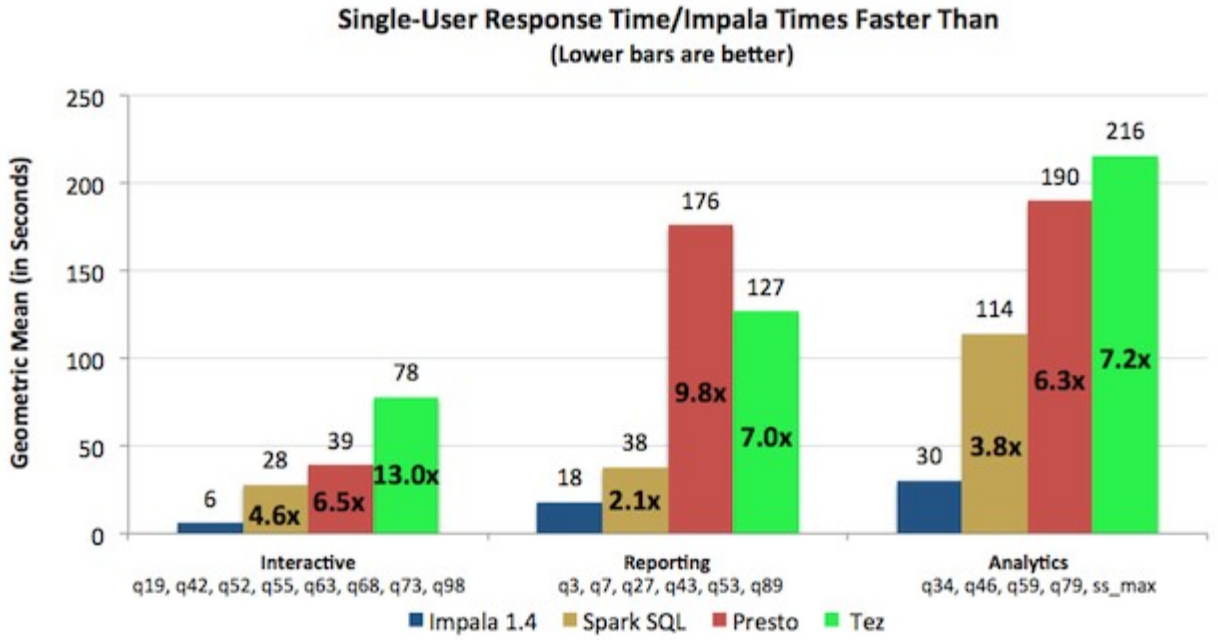
- **Many other tools (e.g. Impala, Presto) follow similar ideas**
  - DAG instead of MapReduce
  - In-memory processing capabilities
- **Because they optimize for performance, sometimes they lack some features**
  - e.g. fault-tolerance, integration with YARN, rich data types, security

SQL-on-Hadoop Benchmarks

- **Conducted by Cloudera**
- **Benchmark claims to be fair and realistic**
  - A realistic set of SQL queries
  - Most optimal file formats across all engines
  - Configuration tuning of each engine
  - Multiple runs of SQL queries for each engine
  - 21-node cluster with standard hardware configuration
  - Possibility to verify by re-running benchmarks on your own

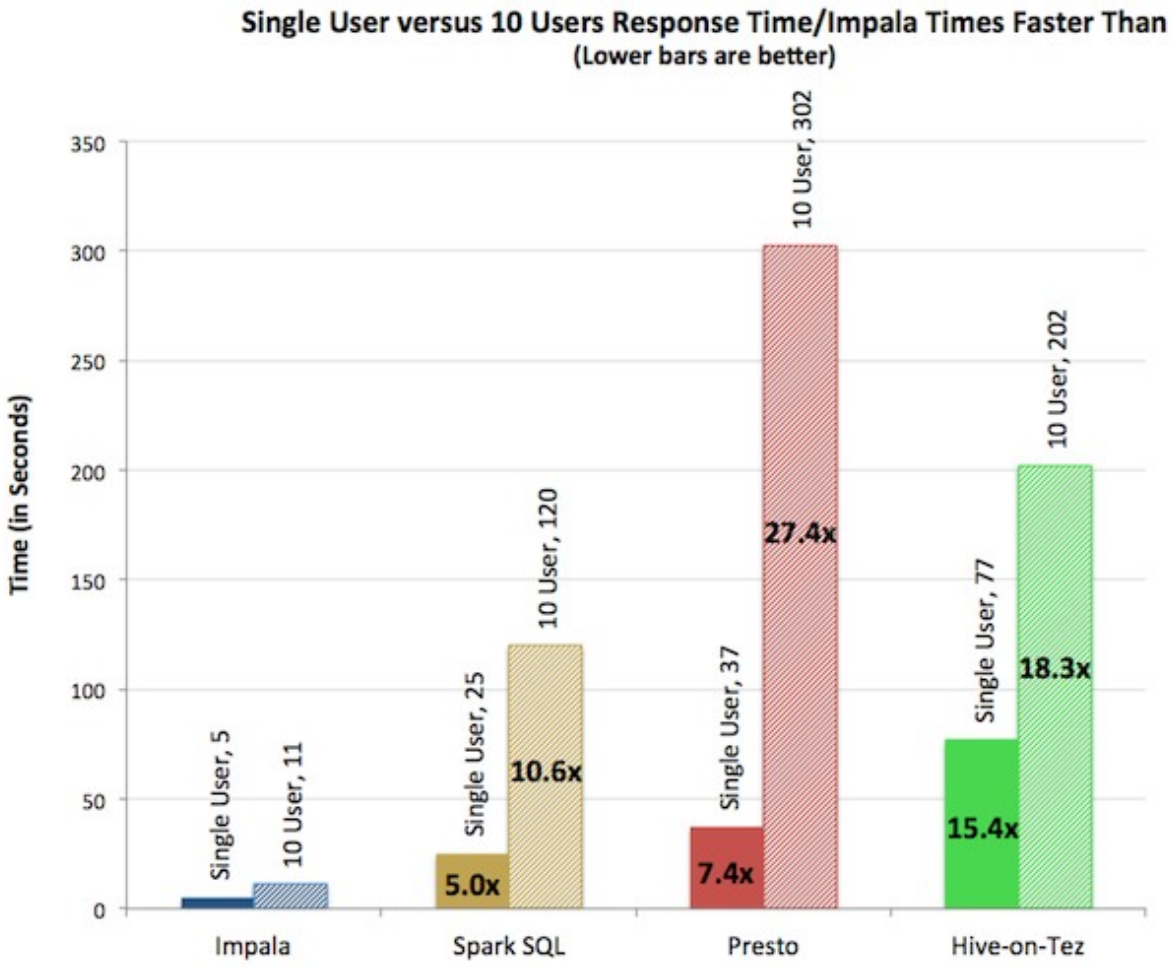
Single-User Benchmarks

- **Impala outperformed all alternatives across all queries run**
- **Impala's performance advantaged ranged from 2.1x to 13.0x**
  - 6.7x faster on average



Multi-User Benchmarks

- **Impala is 18.7x times faster on average**



Which of use?
Hive, Impala or Presto?

Which To Use?

- **Hive is de facto standard and something that you must have**
  - Executes successfully most of the queries
  - Runs on YARN
  - Becomes faster and faster

Can use Tez and Spark execution modes

  - Supported by each vendor
  - Large community that continuously improves it

Which To Use?

- **Presto and Impala are nice additions**
  - Executes some queries, but in a very fast way
  - Don’t integrate with YARN so well
  - Limited support from vendors
  - Smaller (but growing) community

Vectorization

- **Vectorization processes batches of 1024 rows at once instead of single row each time**
  - Operations like scans, aggregations, filters and joins
- **Significantly improves query execution time**
- **It's enabled with two parameters settings**
  - hive.vectorized.execution.enabled
  - hive.vectorized.execution.reduce.enabled

Streaming

- **Works similar to Hadoop Streaming**
  - Opens I/O pipe to an external process
  - Reads from STDIN, writes to STDOUT
- **Is not very efficient**
  - Slow serializing and deserializing data, hard to debug
- **But has advantages**
  - Enables fast prototyping
  - Leverages existing code that is not written in Java
- **Expressed by `MAP()`, `REDUCE()` or `TRANSFORM()` UDFs**

Extending Hive With Scripting Languages

Streaming: Basic Example

```
hive> SELECT TRANSFORM (name,registration)
    > USING '/bin/cut -f1'
  > AS new_user FROM users;

OK
JENSEN
ANTHONY
ENZO
…
```

We've recently received a lot of complaints from users whose accounts had been stolen. It happened because those users had very weak passwords. We would like to send change requests to users who use passwords that are too easy to guess. Let's take advantage of Hive Streaming to prepare a list of such users!

Back in *StreamRock™*

Streaming: Input Data

```
hive> SELECT * FROM user_pass;
OK
345 rock
13  robot123
67  mammy
98  password
67  Hks67fHIKy45
```

Streaming: Transform With Script

- **Script file has to be added to distributed cache first!**

```
ADD FILE passCheck.pl
```

```
hive> SELECT TRANSFORM(user_id,password)
 > USING 'perl passCheck.pl'
   > AS user_id,protection
   > FROM users;
```

Streaming: Output

INPUT

```
OK
345      rock
13     robot123
67     mammy
98     password
67     Hks67fHIKy45
```

OUTPUT

```
OK
345      LOW
13     HIGH
67     LOW
98     MEDIUM
67     HIGH
```

Local Mode

- **Run MapReduce jobs locally on user's workstation**
- **Enable running local mode automatically**
  - `Sethive.exec.mode.localauto=true;`

Parallel Execution

- **By default Hive executes all stages one at a time**
- **Independent stages can be executed simultaneously**
  - it can decrease overall execution time
- **Parallel execution is disabled by default**

```
SET hive.exec.parallel=true;
```

Indexes in Hive

Indexes

- **You can build an index on columns to speed up some queries**
- **Alternative to Partitioning**
  - When logical partitions are too numerous
- **Can prune some HDFS blocks from a table as input for MapReduce job**

Indexes

- **Stored in another Hive table**
  - Requires extra disk space
- **Customizable with plug-in Java code**
- **Requires careful evaluation**
  - Has significant processing cost to build and use it

Create Index

```
CREATE INDEX users_index
ON TABLE users (state, name)
AS 'org.apache.hadoop.hive.ql.index
   .compact.CompactIndexHandler'
WITH DEFERRED REBUILD
IDXPROPERTIES ('creator' = 'jeff')
IN TABLE users_index_table
   COMMENT 'users partitioned on state and name';
```

Operations On Index

- **Rebuilding the Index**

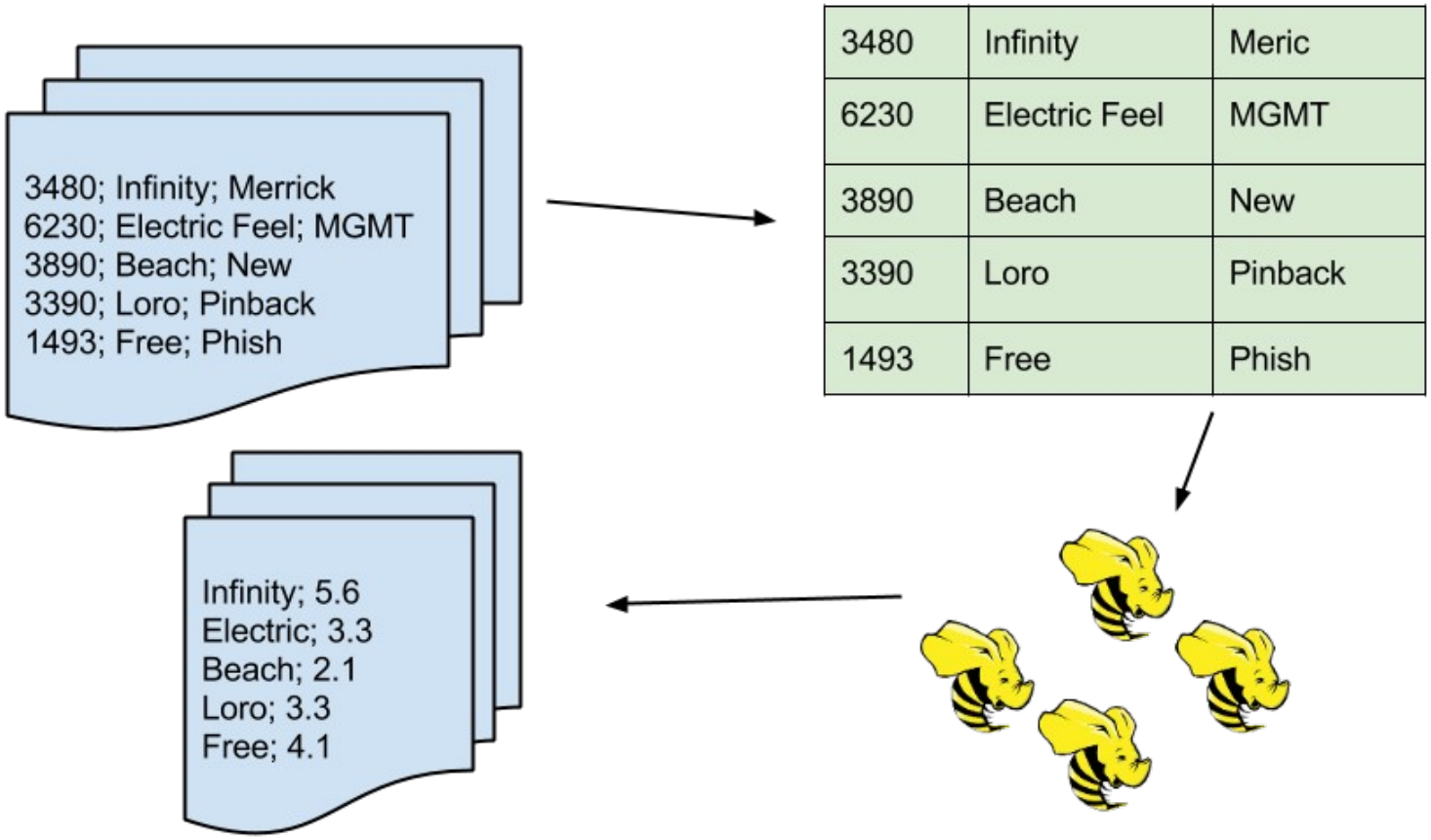
```
ALTER INDEX users_index ON users REBUILD;
```

- **Dropping the Index**

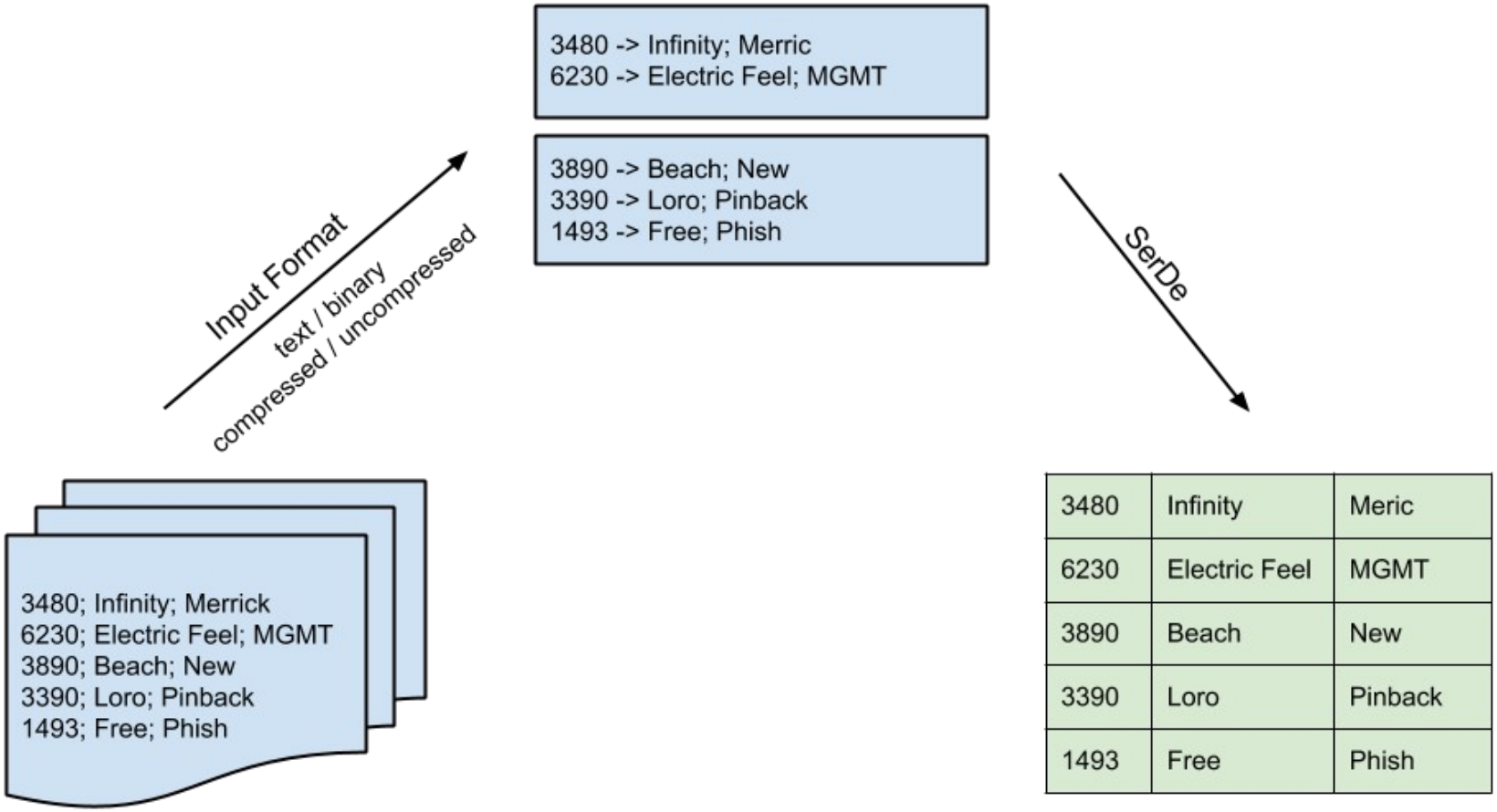
```
SHOW FORMATTED INDEX ON users;
```

```
DROP INDEX IF EXISTS users_index ON users;
```

Format Conversion



Raw Data To Hive Table



Hive Table to Raw Data