



# Hive & Performance

Toronto Hadoop User Group

July 23 2013



Presenter:

Adam Muise – Hortonworks

[amuise@hortonworks.com](mailto:amuise@hortonworks.com)



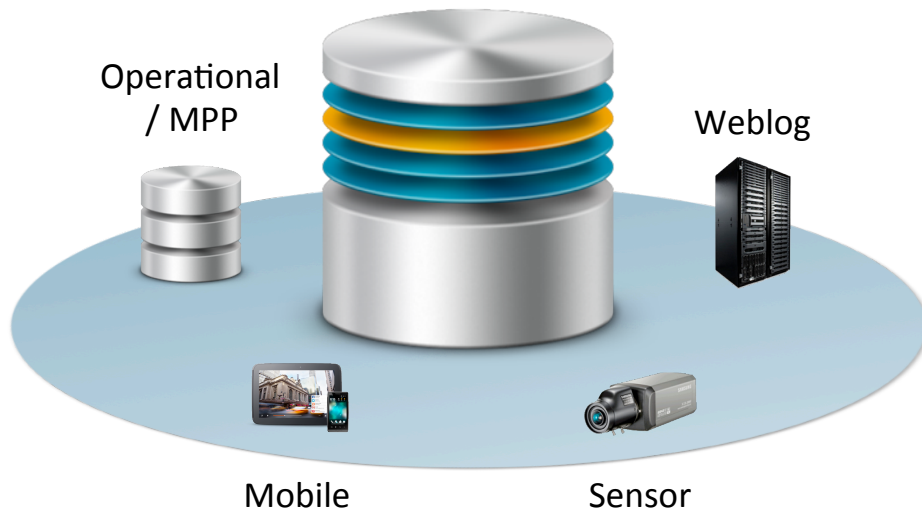
# Agenda

---

- **Hive – What Is It Good For?**
- Hive's Architecture and SQL Compatibility
- Turning Hive Performance to 11
- Get Data In and Out of Hive
- Hive Security
- Project Stinger – Making Hive 100x Faster
- Connecting to Hive From Popular Tools

# Hive – SQL Analytics For Any Data Size

**Store and Query all  
Data in Hive**



**Use Existing SQL Tools  
and Existing SQL Processes**



# Hive's Focus

---

- **Scalable SQL processing over data in Hadoop**
- **Scales to 100PB+**
- **Structured and Unstructured data**

# Comparing Hive with RDBMS

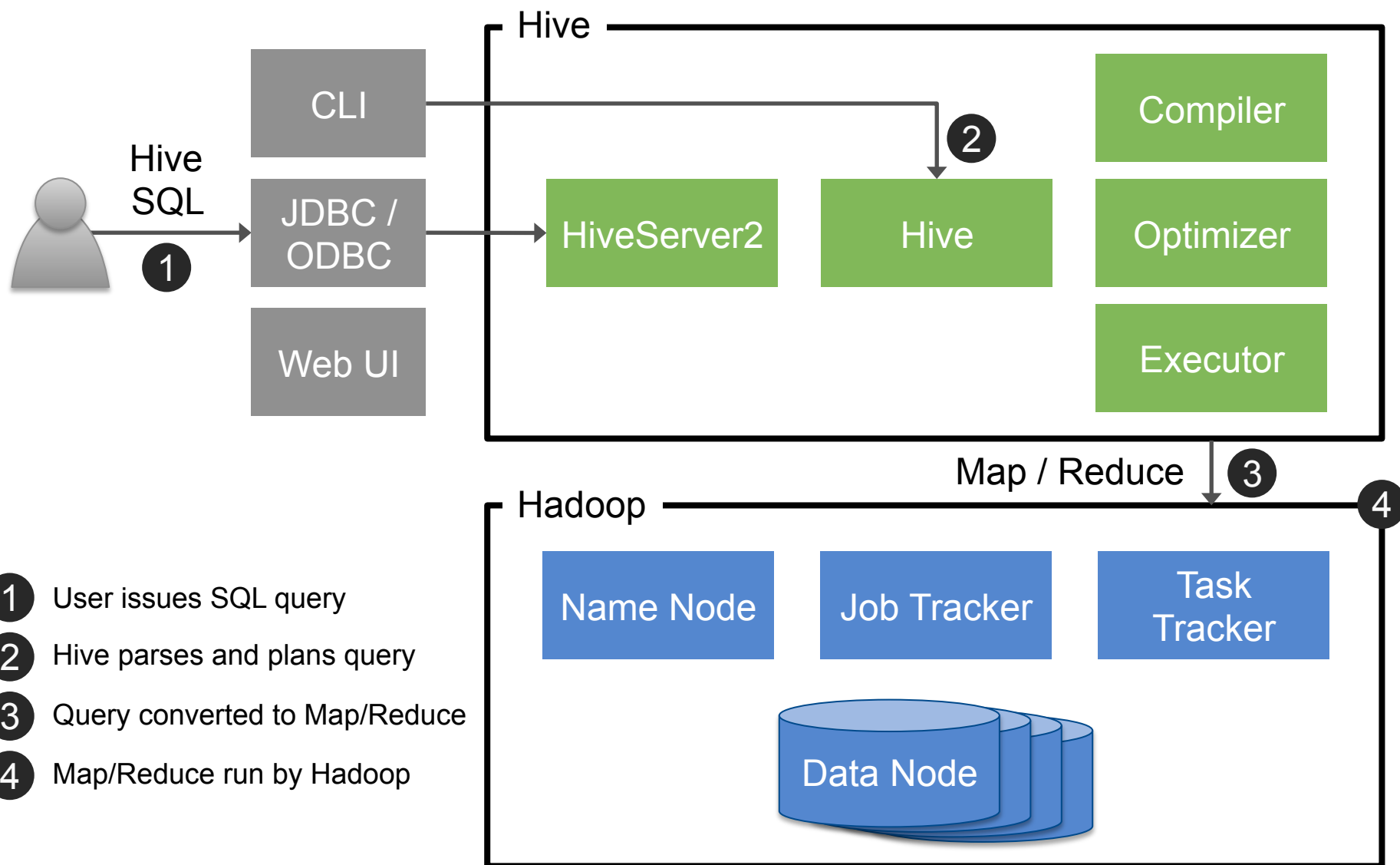
Hive	RDBMS
SQL Interface.	SQL Interface.
Focus on analytics.	May focus on online or analytics.
No transactions.	Transactions usually supported.
Partition adds, no random INSERTs. In-Place updates not natively supported (but are possible).	Random INSERT and UPDATE supported.
Distributed processing via map/reduce.	Distributed processing varies by vendor (if available).
Scales to hundreds of nodes.	Seldom scale beyond 20 nodes.
Built for commodity hardware.	Often built on proprietary hardware (especially when scaling out).
Low cost per petabyte.	What's a petabyte?

# Agenda

---

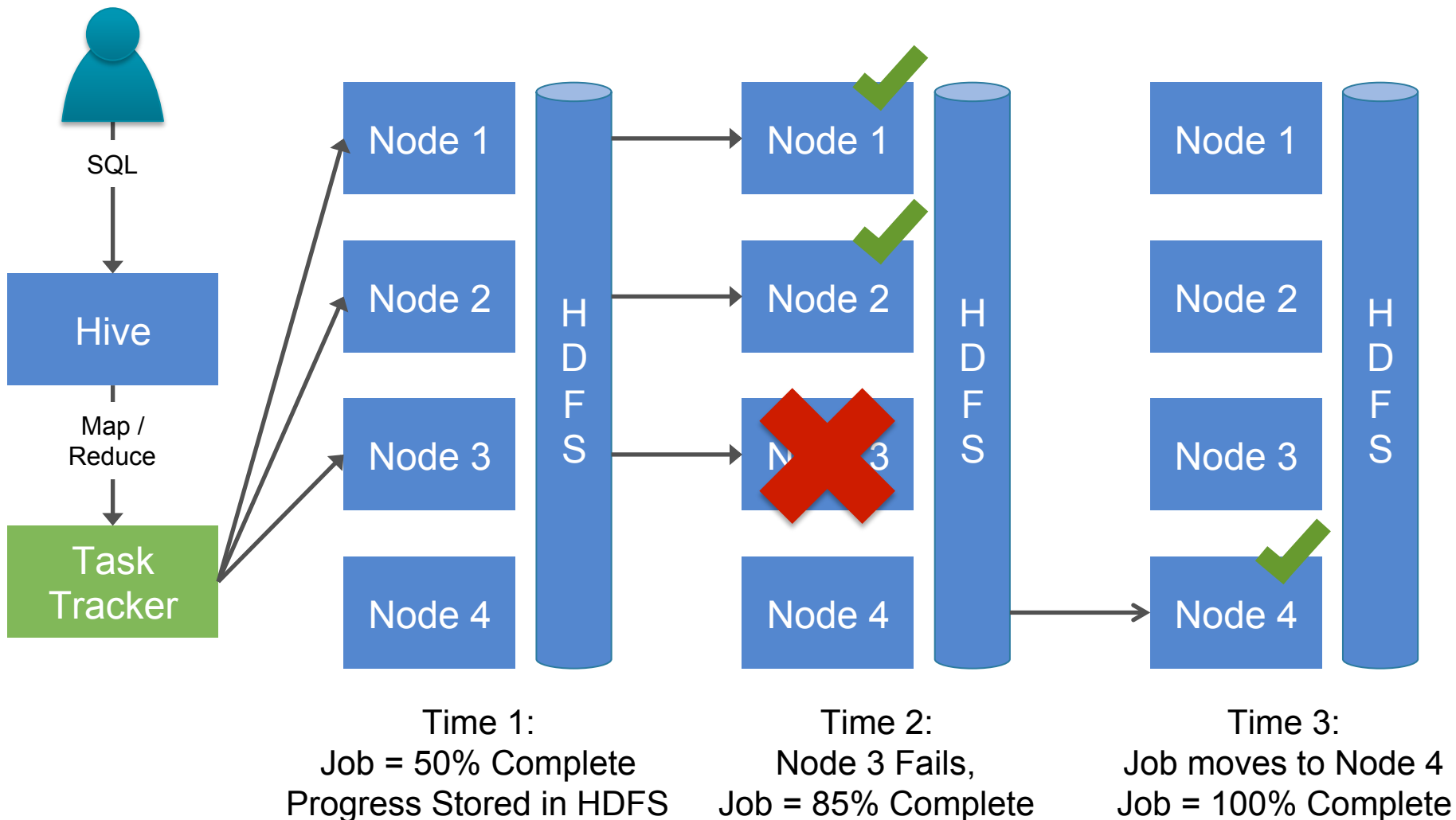
- Hive – What Is It Good For?
- **Hive's Architecture and SQL Compatibility**
- Turning Hive Performance to 11
- Get Data In and Out of Hive
- Hive Security
- Project Stinger – Making Hive 100x Faster
- Connecting to Hive From Popular Tools

# Hive: The SQL Interface to Hadoop



- 1 User issues SQL query
- 2 Hive parses and plans query
- 3 Query converted to Map/Reduce
- 4 Map/Reduce run by Hadoop

# Hive: Reliable SQL Processing at Scale





# SQL Coverage: SQL 92 with Extensions




## SQL Datatypes

INT
TINYINT/SMALLINT/BIGINT
BOOLEAN
FLOAT
DOUBLE
STRING
BINARY
TIMESTAMP
ARRAY, MAP, STRUCT, UNION
DECIMAL
CHAR
VARCHAR
DATE

## SQL Semantics

SELECT, LOAD, INSERT from query
Expressions in WHERE and HAVING
GROUP BY, ORDER BY, SORT BY
CLUSTER BY, DISTRIBUTE BY
Sub-queries in FROM clause
GROUP BY, ORDER BY
ROLLUP and CUBE
UNION
LEFT, RIGHT and FULL INNER/OUTER JOIN
CROSS JOIN, LEFT SEMI JOIN
Windowing functions (OVER, RANK, etc.)
Sub-queries for IN/NOT IN, HAVING
EXISTS / NOT EXISTS
INTERSECT, EXCEPT

## Legend

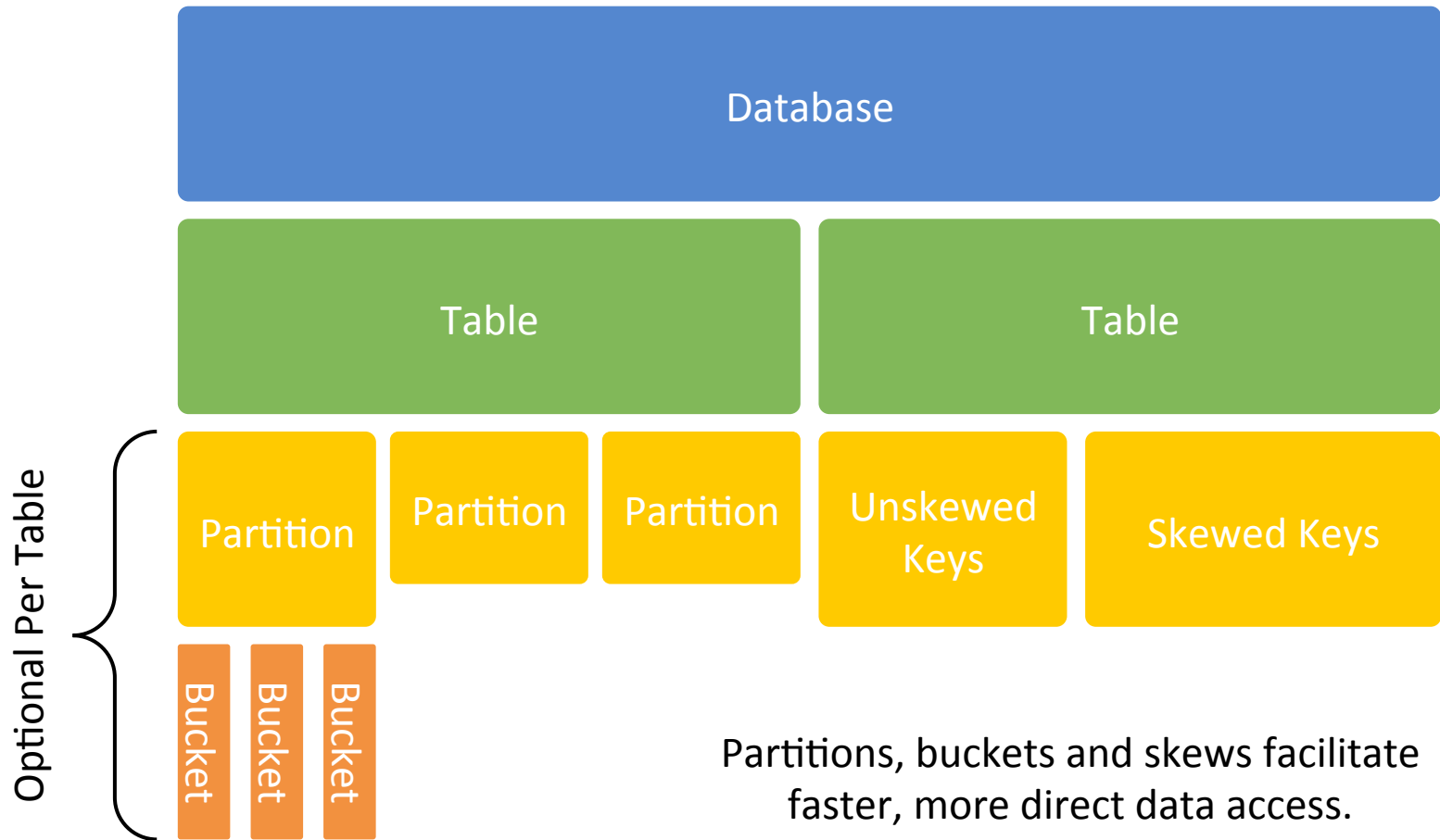
	Available
	New in Hive 0.11
	Roadmap

# Agenda

---

- Hive – What Is It Good For?
- Hive’s Architecture and SQL Compatibility
- **Turning Hive Performance to 11**
- Get Data In and Out of Hive
- Hive Security
- Project Stinger – Making Hive 100x Faster
- Connecting to Hive From Popular Tools

# Data Abstractions in Hive



# “I heard you should avoid joins...”

---

- **“Joins are evil” – Cal Henderson**
  - Joins should be avoided in online systems.
- **Joins are unavoidable in analytics.**
  - Making joins fast is the key design point.

# Quick Refresher on Joins

customer			order		
first	last	id	cid	price	quantity
Nick	Toner	11911	4150	10.50	3
Jessie	Simonds	11912	11914	12.25	27
Kasi	Lamers	11913	3491	5.99	5
Rodger	Clayton	11914	2934	39.99	22
Verona	Hollen	11915	11914	40.50	10

```
SELECT * FROM customer join order ON customer.id = order.cid;
```

Joins match values from one table against values in another table.

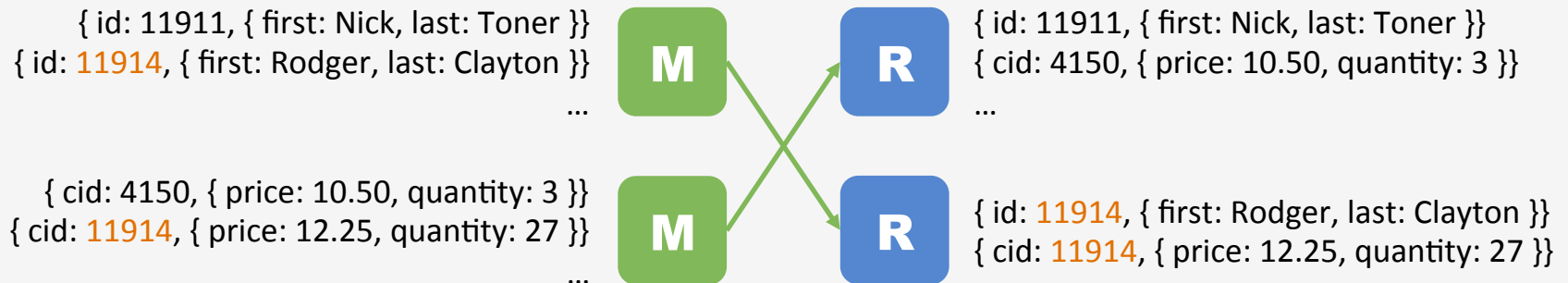
# Hive Join Strategies

Type	Approach	Pros	Cons
<b>Shuffle Join</b>	Join keys are shuffled using map/reduce and joins performed join side.	Works regardless of data size or layout.	Most resource-intensive and slowest join type.
<b>Broadcast Join</b>	Small tables are loaded into memory in all nodes, mapper scans through the large table and joins.	Very fast, single scan through largest table.	All but one table must be small enough to fit in RAM.
<b>Sort-Merge-Bucket Join</b>	Mappers take advantage of co-location of keys to do efficient joins.	Very fast for tables of any size.	Data must be sorted and bucketed ahead of time.

# Shuffle Joins in Map Reduce

customer			order		
first	last	id	cid	price	quantity
Nick	Toner	11911	4150	10.50	3
Jessie	Simonds	11912	11914	12.25	27
Kasi	Lamers	11913	3491	5.99	5
Rodger	Clayton	11914	2934	39.99	22
Verona	Hollen	11915	11914	40.50	10

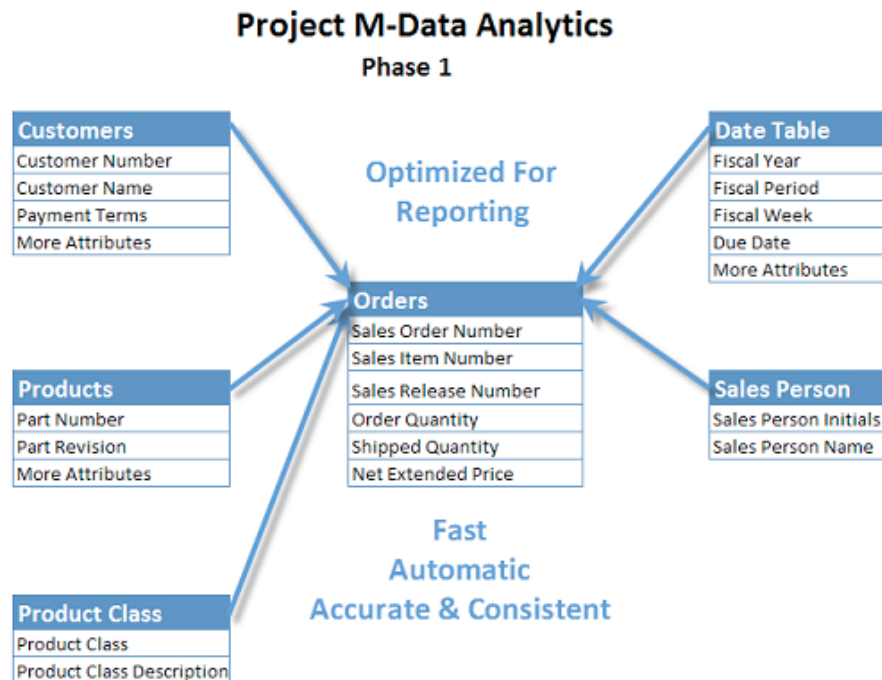
```
SELECT * FROM customer join order ON customer.id = order.cid;
```



Identical keys shuffled to the same reducer. Join done reduce-side.  
Expensive from a network utilization standpoint.

# Broadcast Join

- Star schemas use dimension tables small enough to fit in RAM.
- Small tables held in memory by all nodes.
- Single pass through the large table.
- Used for star-schema type joins common in DW.





# When both are too large for memory:

customer			order		
first	last	id	cid	price	quantity
Nick	Toner	11911	4150	10.50	3
Jessie	Simonds	11912	11914	12.25	27
Kasi	Lamers	11913	11914	40.50	10
Rodger	Clayton	11914	12337	39.99	22
Verona	Hollen	11915	15912	40.50	10

```
SELECT * FROM customer join order ON customer.id = order.cid;
```


**Cluster and sort by the most common join key.**

```
CREATE TABLE order (cid int, price float, quantity int)  
CLUSTERED BY(cid) SORTED BY(cid) INTO 32 BUCKETS;
```

```
CREATE TABLE customer (id int, first string, last string)  
CLUSTERED BY(id) SORTED BY(id) INTO 32 BUCKETS;
```

# Hive's Clustering and Sorting

customer			order		
first	last	id	cid	price	quantity
Nick	Toner	11911	4150	10.50	3
Jessie	Simonds	11912	11914	12.25	27
Kasi	Lamers	11913	11914	40.50	10
Rodger	Clayton	11914	12337	39.99	22
Verona	Hollen	11915	15912	40.50	10




```
SELECT * FROM customer join order ON customer.id = order.cid;
```

Observation 1:  
Sorting by the join key makes joins easy.  
All possible matches reside in the same area on disk.

# Hive's Clustering and Sorting

customer			order		
first	last	id	cid	price	quantity
Nick	Toner	11911	4150	10.50	3
Jessie	Simonds	11912	11914	12.25	27
Kasi	Lamers	11913	11914	40.50	10
Rodger	Clayton	11914	12337	39.99	22
Verona	Hollen	11915	15912	40.50	10



```
SELECT * FROM customer join order ON customer.id = order.cid;
```

## Observation 2:

Hash bucketing a join key ensures all matching values reside on the same node.  
Equi-joins can then run with no shuffle.

# Controlling Data Locality with Hive

- **Bucketing:**
  - Hash partition values into a configurable number of buckets.
  - Usually coupled with sorting.
- **Skews:**
  - Split values out into separate files.
  - Used when certain values are frequently seen.
- **Replication Factor:**
  - Increase replication factor to accelerate reads.
  - Controlled at the HDFS layer.
- **Sorting:**
  - Sort the values within given columns.
  - Greatly accelerates query when used with ORCFile filter pushdown.

# Guidelines for Architecting Hive Data

Table Size	Data Profile	Query Pattern	Recommendation
Small	Hot data	Any	Increase replication factor
Any	Any	Very precise filters	Sort on column most frequently used in precise queries
Large	Any	Joined to another large table	Sort and bucket both tables along the join key
Large	One value >25% of count within high cardinality column	Any	Split the frequent value into a separate skew
Large	Any	Queries tend to have a natural boundary such as date	Partition the data along the natural boundary

# Hive Persistence Formats

---

- **Built-in Formats:**

- ORCFile
- RCFile
- Avro
- Delimited Text
- Regular Expression
- S3 Logfile
- Typed Bytes

- **3<sup>rd</sup>-Party Addons:**

- JSON
- XML

# Hive allows mixed format.

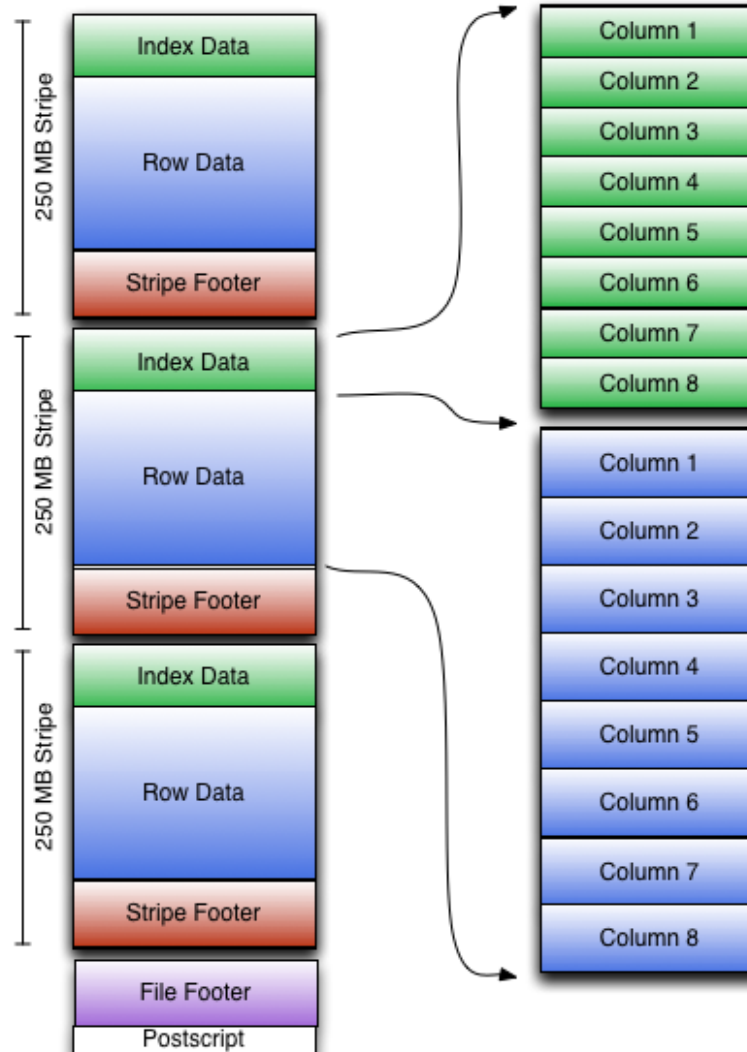
---

- **Use Case:**

- Ingest data in a write-optimized format like JSON or delimited.
- Every night, run a batch job to convert to read-optimized ORCFile.

# ORCFile – Efficient Columnar Layout

Large block size well suited for HDFS.



Columnar format arranges columns adjacent within the file for compression and fast access.



# ORCFile Advantages

---

- **High Compression**

- Many tricks used out-of-the-box to ensure high compression rates.
- RLE, dictionary encoding, etc.

- **High Performance**

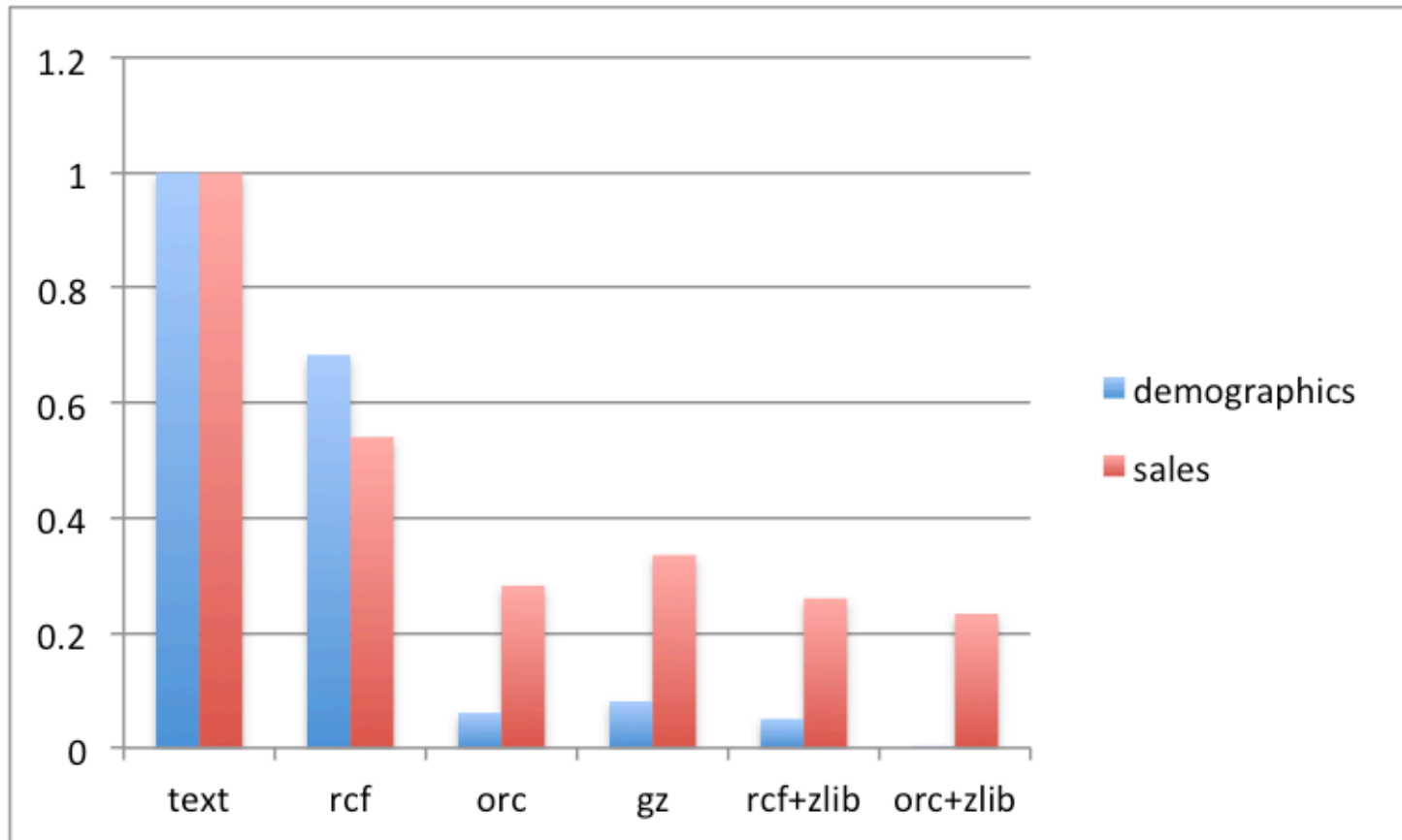
- Inline indexes record value ranges within blocks of ORCFile data.
- Filter pushdown allows efficient scanning during precise queries.

- **Flexible Data Model**

- All Hive types including maps, structs and unions.

# High Compression with ORCFile

- Data set from TPC-DS



# Some ORCFile Samples

sale					
id	timestamp	productsk	storesk	amount	state
10000	2013-06-13T09:03:05	16775	670	\$70.50	CA
10001	2013-06-13T09:03:05	10739	359	\$52.99	IL
10002	2013-06-13T09:03:06	4671	606	\$67.12	MA
10003	2013-06-13T09:03:08	7224	174	\$96.85	CA
10004	2013-06-13T09:03:12	9354	123	\$67.76	CA
10005	2013-06-13T09:03:18	1192	497	\$25.73	IL

```
CREATE TABLE sale (  
    id int, timestamp timestamp,  
    productsk int, storesk int,  
    amount decimal, state string  
) STORED AS orc;
```

# ORCFile Options and Defaults

Key	Default	Notes
orc.compress	ZLIB	High level compression (one of NONE, ZLIB, SNAPPY)
orc.compress.size	262,144 (= 256 KiB)	Number of bytes in each compression chunk
orc.stripe.size	268,435,456 (= 256 MiB)	Number of bytes in each stripe
orc.row.index.stride	10,000	Number of rows between index entries (must be $\geq 1,000$ )
orc.create.index	true	Whether to create row indexes

# No Compression: Faster but Larger

sale					
id	timestamp	productsk	storesk	amount	state
10000	2013-06-13T09:03:05	16775	670	\$70.50	CA
10001	2013-06-13T09:03:05	10739	359	\$52.99	IL
10002	2013-06-13T09:03:06	4671	606	\$67.12	MA
10003	2013-06-13T09:03:08	7224	174	\$96.85	CA
10004	2013-06-13T09:03:12	9354	123	\$67.76	CA
10005	2013-06-13T09:03:18	1192	497	\$25.73	IL

```
CREATE TABLE sale (  
  id int, timestamp timestamp,  
  productsk int, storesk int,  
  amount decimal, state string  
) STORED AS orc tblproperties ("orc.compress"="NONE");
```

# Column Sorting to Facilitate Skipping

sale					
id	timestamp	productsk	storesk	amount	state
10005	2013-06-13T09:03:18	1192	497	\$25.73	IL
10002	2013-06-13T09:03:06	4671	606	\$67.12	MA
10003	2013-06-13T09:03:08	7224	174	\$96.85	CA
10004	2013-06-13T09:03:12	9354	123	\$67.76	CA
10001	2013-06-13T09:03:05	10739	359	\$52.99	IL
10000	2013-06-13T09:03:05	16775	670	\$70.50	CA

```
CREATE TABLE sale (  
  id int, timestamp timestamp,  
  productsk int, storesk int,  
  amount decimal, state string  
) STORED AS orc;  
INSERT INTO sale AS SELECT * FROM staging SORT BY productsk;
```

ORCFile skipping speeds queries like  
WHERE productsk = X, productsk IN (Y, Z); etc.

# Not Your Traditional Database

- **Traditional solution to all RDBMS problems:**
  - Put an index on it!



Your Oracle DBA

- **Doing this in Hadoop = #fail**

# Going Fast in Hadoop

- **Hadoop:**
  - Really good at coordinated sequential scans.
  - No random I/O. Traditional index pretty much useless.
- **Keys to speed in Hadoop:**
  - Sorting and skipping take the place of indexing.
  - Minimizing data shuffle the other key consideration.
- **Skipping data:**
  - Divide data among different files which can be pruned out.
    - Partitions, buckets and skews.
  - Skip records during scans using small embedded indexes.
    - Automatic when you use ORCFile format.
  - Sort data ahead of time.
    - Simplifies joins and skipping becomes more effective.



# Data Layout Considerations for Fast Hive

Skip Reads	Minimize Shuffle	Reduce Latency
Partition Tables and/or Use Skew	Sort and Bucket on Common Join Keys	Increase Replication Factor For Hot Data
Sort Secondary Columns when Using ORCFile	Use Broadcast Joins when Joining Small Tables	Enable Short-Circuit Read
		Take Advantage of Tez + Tez Service (Future)

# Partitioning and Virtual Columns

- **Partitioning makes queries go fast.**
- **You will almost always use some sort of partitioning.**
- **When partitioning you will use 1 or more virtual columns.**

```
# Notice how xdate and state are not "real" column names.  
CREATE TABLE sale (  
    id int, amount decimal, ...  
) partitioned by (xdate string, state string);
```

- **Virtual columns cause directories to be created in HDFS.**
  - Files for that partition are stored within that subdirectory.

# Loading Data with Virtual Columns

- **By default at least one virtual column must be hard-coded.**

```
INSERT INTO sale (xdate='2013-03-01', state='CA')
SELECT * FROM staging_table
WHERE xdate = '2013-03-01' AND state = 'CA';
```

- **You can load all partitions in one shot:**
  - set `hive.exec.dynamic.partition.mode=nonstrict`;
  - Warning: You can easily overwhelm your cluster this way.

```
set hive.exec.dynamic.partition.mode=nonstrict;
INSERT INTO sale (xdate, state)
SELECT * FROM staging_table;
```

# You May Need to Re-Order Columns

- **Virtual columns must be last within the inserted data set.**
- **You can use the `SELECT` statement to re-order.**

```
INSERT INTO sale (xdate, state='CA')
SELECT
    id, amount, other_stuff,
    xdate, state
FROM staging_table
WHERE state = 'CA';
```

# The Most Essential Hive Query Tunings



# Tune Split Size – Always

- **mapred.max.split.size** and **mapred.min.split.size**
- **Hive processes data in chunks subject to these bounds.**
- **min too large -> Too few mappers.**
- **max too small -> Too many mappers.**
- **Tune variables until mappers occupy:**
  - All map slots if you own the cluster.
  - Reasonable number of map slots if you don't.
- **Example:**
  - set `mapred.max.split.size=100000000`;
  - set `mapred.min.split.size=1000000`;
- **Manual today, automatic in future version of Hive.**
- **You will need to set these for most queries.**

# Tune io.sort.mb – Sometimes

- **Hive and Map/Reduce maintain some separate buffers.**
- **If Hive maps need lots of local memory you may need to shrink map/reduce buffers.**
- **If your maps spill, try it out.**
- **Example:**
  - `set io.sort.mb=100;`

# Other Settings You Need

- **All the time:**
  - set `hive.optimize.mapjoin.mapreduce=true`;
  - set `hive.optimize.bucketmapjoin=true`;
  - set `hive.optimize.bucketmapjoin.sortedmerge=true`;
  - set `hive.auto.convert.join=true`;
  - set `hive.auto.convert.sortmerge.join=true`;
  - set `hive.auto.convert.sortmerge.join.noconditionaltask=true`;
- **When bucketing data:**
  - set `hive.enforce.bucketing=true`;
  - set `hive.enforce.sorting=true`;
- **These and more are set by default in HDP 1.3.**
  - Check for them in `hive-site.xml`
  - If not present, set them in your query script



# Check Your Settings

- **In Hive shell:**

- See all settings with “set;”

```
hive> set;  
datanucleus.autoCreateSchema=true  
datanucleus.autoStartMechanismMode=checked  
datanucleus.cache.level2=false  
datanucleus.cache.level2.type=none  
datanucleus.connectionPoolingType=DBCP  
datanucleus.identifierFactory=datanucleus1  
datanucleus.plugin.pluginRegistryBundleCheck=LOG  
datanucleus.storeManagerType=rdbms
```

- See a particular setting by adding it.

```
hive> set hive.auto.convert.join;  
hive.auto.convert.join=true  
hive> █
```

- Change a setting.

```
hive> set hive.auto.convert.join=false;  
hive> █
```

# Example Workflow: Create a Staging table

```
CREATE EXTERNAL TABLE pos_staging (  
    txnid STRING,  
    txntime STRING,  
    givenname STRING,  
    lastname STRING,  
    postalcode STRING,  
    storeid STRING,  
    ind1 STRING,  
    productid STRING,  
    purchaseamount FLOAT,  
    creditcard STRING  
)ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' '  
LOCATION '/user/hdfs/staging_data/pos_staging';
```

*The raw data is the result of initial loading or the output of a mapreduce or pig job. We create an external table over the results of that job as we only intend to use it to load an optimized table.*

# Example Workflow: Choose a partition scheme

```
hive> select distinct concat(year(txntime),month(txntime)) as part_dt
from pos_staging;
...
OK
20121
201210
201211
201212
20122
20123
20124
20125
20126
20127
20128
20129
Time taken: 21.823 seconds, Fetched: 12 row(s)
```

*Execute a query to determine if the partition choice returns a reasonable result. We will use this projection to create partitions for our data set. You want to keep your partitions large enough to be useful in partition pruning and efficient for HDFS storage. Hive has configurable bounds to ensure you do not exceed per node and total partition counts (defaults shown):*

*hive.exec.max.dynamic.partitions=1000*

*hive.exec.max.dynamic.partitions.pernode=100*

# Example Workflow: Define optimized table

```
CREATE TABLE fact_pos
(
  txnid STRING,
  txntime STRING,
  givenname STRING,
  lastname STRING,
  postalcode STRING,
  storeid STRING,
  ind1 STRING,
  productid STRING,
  purchaseamount FLOAT,
  creditcard STRING
) PARTITIONED BY (part_dt STRING)
CLUSTERED BY (txnid)
SORTED BY (txnid)
INTO 24 BUCKETS
STORED AS ORC tblproperties ("orc.compress"="SNAPPY");
```

*The `part_dt` field is defined in the partition by clause and cannot be the same name as any other fields. In this case, we will be performing a modification of `txntime` to generate a partition key. The cluster and sorted clauses contain the only key we intend to join the table on. We have stored as ORCFile with Snappy compression.*

# Example Workflow: Load Data Into Optimized Table

```
set hive.enforce.sorting=true;
set hive.enforce.bucketing=true;
set hive.exec.dynamic.partition=true;
set hive.exec.dynamic.partition.mode=nonstrict;
set mapreduce.reduce.input.limit=-1;

FROM pos_staging
INSERT OVERWRITE TABLE fact_pos
PARTITION (part_dt)
SELECT
    txnid,
    txntime,
    givenname,
    lastname,
    postalcode,
    storeid,
    ind1,
    productid,
    purchaseamount,
    creditcard,
    concat(year(txntime),month(txntime)) as part_dt
SORT BY productid;
```

*We use this command to load data from our staging table into our optimized ORCFile format. Note that we are using dynamic partitioning with the projection of the txntime field. This results in a MapReduce job that will copy the staging data into ORCFile format Hive managed table.*

# Example Workflow: Increase replication factor

---

```
hadoop fs -setrep -R -w 5 /apps/hive/warehouse/fact_pos
```

Increase the replication factor for the high performance table. This increases the chance for data locality. In this case, the increase in replication factor is not for additional resiliency. This is a trade-off of storage for performance.

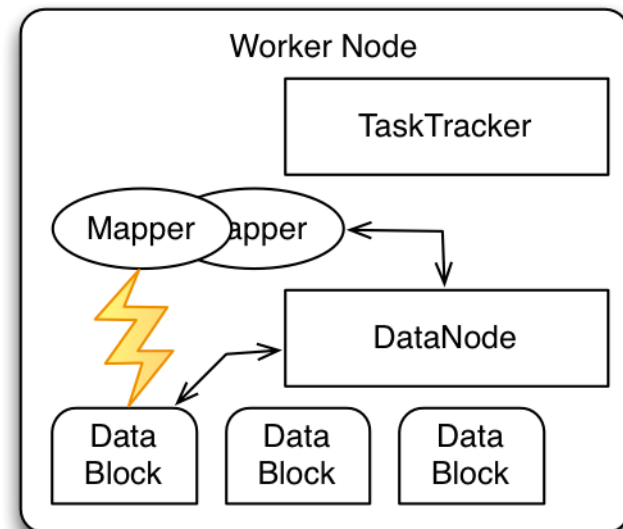
In fact, to conserve space, you may choose to reduce the replication factor for older data sets or even delete them altogether. With the raw data in place and untouched, you can always recreate the ORCFile high performance tables. Most users place the steps in this example workflow into an Oozie job to automate the work.

# Example Workflow: Enabling Short Circuit Read

In `hdfs-site.xml` (or your custom Ambari settings for HDFS, restart service after):

```
dfs.block.local-path-access.user=hdfs
dfs.client.read.shortcircuit=true
dfs.client.read.shortcircuit.skip.checksum=false
```

*Short Circuit reads allow the mappers to bypass the overhead of opening a port to the datanode if the data is local. The permissions for the local block files need to permit hdfs to read them (should be by default already)  
See [HDFS-2246](#) for more details.*



Lightning Bolt = Short Circuit Read  
HDFS-2246

# Example Workflow: Execute your query

```
set hive.mapred.reduce.tasks.speculative.execution=false;
set io.sort.mb=300;
set mapreduce.reduce.input.limit=-1;
```

```
select productid, ROUND(SUM(purchaseamount),2) as total
from fact_pos
where part_dt between '201210' and '201212'
group by productid
order by total desc
limit 100;
```

...

OK

20535 3026.87

39079 2959.69

28970 2869.87

45594 2821.15

...

15649 2242.05

47704 2241.22

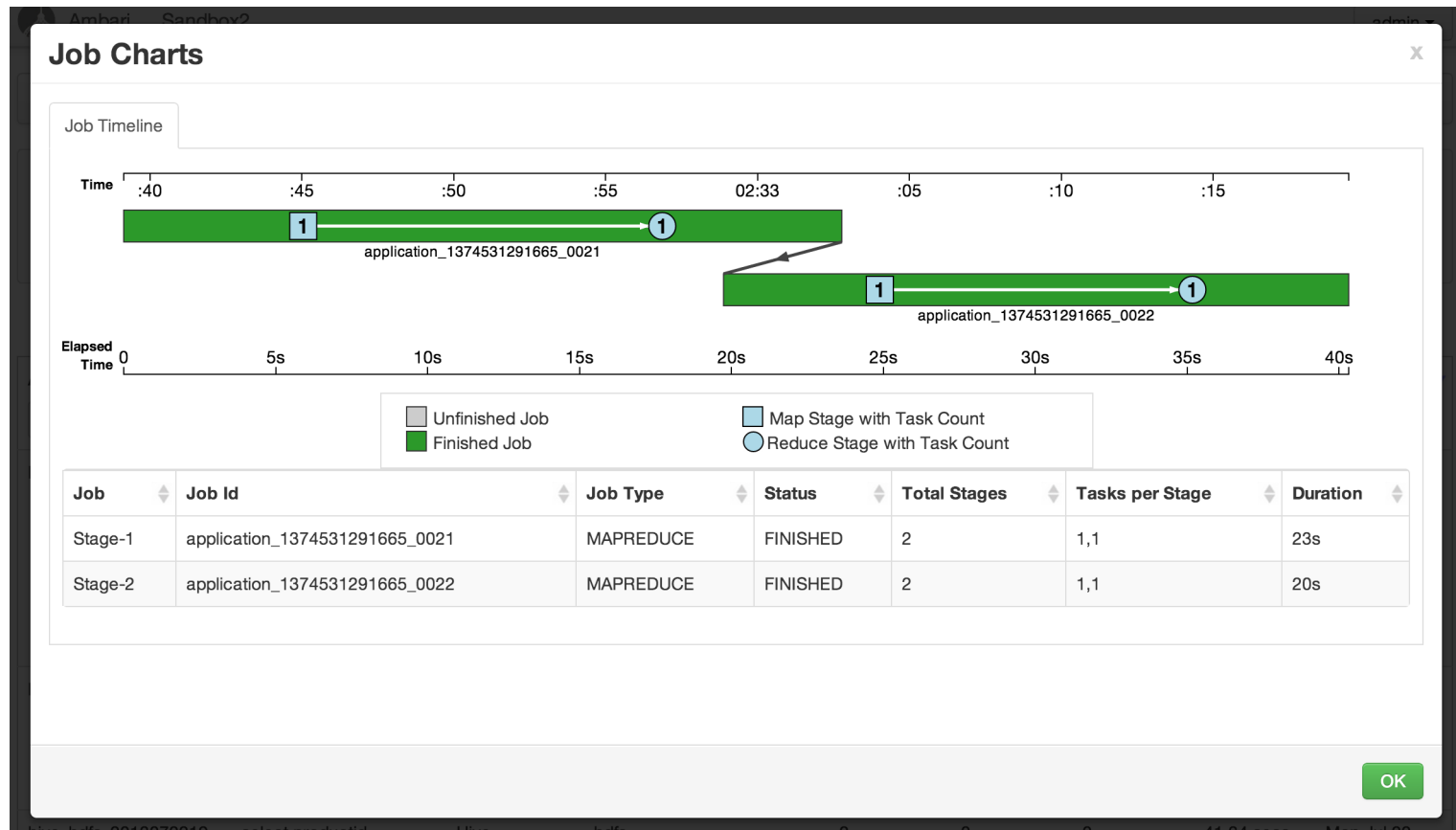
8140 2238.61

Time taken: 40.087 seconds, Fetched: 100 row(s)

*In the case above, we have a simple query executed to test out our table. We have some example parameters set before our query. The good news is that most of the parameters regarding join and engine optimizations are already set for you in Hive 0.11 (HDP). The `io.sort.mb` is presented as an example of one of the tunable parameters you may want to change for this particular SQL (note this value assumes 2-3GB JVMs for mappers). We are also partition pruning for the holiday shopping season, Oct to Dec.*



# Example Workflow: Check Execution Path in Ambari



*You can check the execution path in Ambari's Job viewer. This gives a high level overview of the stages and particular number of map and reduce tasks. With Tez, it also shows task number and execution order. The counts here are small as this is a sample from a single-node HDP Sandbox. For more detailed analysis, you will need to read the query plan via "explain".*

# Learn To Read The Query Plan

---

- **“explain extended” in front of your query.**
- **Sections:**
  - Abstract syntax tree – you can usually ignore this.
  - Stage dependencies – dependencies and # of stages.
  - Stage plans – important info on how Hive is running the job.

# A sample query plan.

## ABSTRACT SYNTAX TREE:

```
(TOK_QUERY (TOK_FROM (TOK_JOIN (TOK_JOIN (TOK_TABREF (TOK_TABNAME request)) (TOK_TABREF (TOK_TABNAME client)) (= (. (TOK_TABLE_OR_COL request) clientsk) (. (TOK_TABLE_OR_COL client) id))) (TOK_TABREF (TOK_TABNAME path)) (= (. (TOK_TABLE_OR_COL request) pathsk) (. (TOK_TABLE_OR_COL path) id)))) (TOK_INSERT (TOK_DESTINATION (TOK_DIR TOK_TMP_FILE)) (TOK_SELECT (TOK_SELEXPR (TOK_TABLE_OR_COL path)) (TOK_SELEXPR (TOK_TABLE_OR_COL country)) (TOK_SELEXPR (TOK_FUNCTIONSTAR count) cnt)) (TOK_WHERE (= (. (TOK_TABLE_OR_COL client) country) 'us')) (TOK_GROUPBY (TOK_TABLE_OR_COL path) (TOK_TABLE_OR_COL country)) (TOK_ORDERBY (TOK_TABSORTCOLNAMEDESC (TOK_TABLE_OR_COL cnt))) (TOK_LIMIT 5)))
```

## STAGE DEPENDENCIES:

```
Stage-9 is a root stage
Stage-8 depends on stages: Stage-9
Stage-4 depends on stages: Stage-8
Stage-0 is a root stage
```

4 Stages

## STAGE PLANS:

```
Stage: Stage-9
Map Reduce Local Work
Alias -> Map Local Tables:
  client
    Fetch Operator
      limit: -1
  path
    Fetch Operator
      limit: -1
Alias -> Map Local Operator Tree:
  client
    TableScan
      alias: client
      GatherStats: false
      Filter Operator
        isSamplingPred: false
        predicate:
          expr: (country = 'us')
          type: boolean
```

Stage Details

# Use Case: Star Schema Join



```
# Most popular URLs in US.
select path, country, count(*) as cnt
  from request
  join client on request.clientsk = client.id
  join path on request.pathsk = path.id
where
  client.country = 'us'
group by
  path, country
order by cnt desc
limit 5;
```

Ideal =  
Single scan of fact table.

# Case 1: `hive.auto.convert.join=false;`

- **Generates 5 stages.**

## STAGE DEPENDENCIES:

Stage-1 is a root stage

Stage-2 depends on stages: Stage-1

Stage-3 depends on stages: Stage-2

Stage-4 depends on stages: Stage-3

Stage-0 is a root stage

# Case 1: hive.auto.convert.join=false;

- **4 Map Reduces tells you something is not right.**

Stage: Stage-1  
Map Reduce

Stage: Stage-2  
Map Reduce

Stage: Stage-3  
Map Reduce

Stage: Stage-4  
Map Reduce



# Case 1: `hive.auto.convert.join=true;`

---

- **Only 4 stages this time.**

## STAGE DEPENDENCIES:

Stage-9 is a root stage

Stage-8 depends on stages: Stage-9

Stage-4 depends on stages: Stage-8

Stage-0 is a root stage

# Case 1: `hive.auto.convert.join=false;`

---

- **Only 2 Map Reduces**

Stage: Stage-8  
Map Reduce

Stage: Stage-4  
Map Reduce



# Case 1: hive.auto.convert.join=false;

- **Stage-9 is Map Reduce Local Work. What is that?**

```
client
  TableScan
    alias: client
  Filter Operator
    predicate:
      expr: (country = 'us')
...
path
  TableScan
    alias: path
...
```

- **Hive is loading the dimension tables (client and path) directly. Client is filtered by country.**
- **Remaining map/reduce are for join and order by.**

# Hive Fast Query Checklist

- Partitioned data along natural query boundaries (e.g. date).
- Minimized data shuffle by co-locating the most commonly joined data.
- Took advantage of skews for high-frequency values.
- Enabled short-circuit read.
- Used ORCFile.
- Sorted columns to facilitate row skipping for common targeted queries.
- Verified query plan to ensure single scan through largest table.
- Checked the query plan to ensure partition pruning is happening.
- Used at least one ON clause in every JOIN.

# For Even More Hive Performance

- Increased replication factor for frequently accessed data and dimensions.
- Tuned `io.sort.mb` to avoid spilling.
- Tuned `mapred.max.split.size`, `mapred.min.split.size` to ensure 1 mapper wave.
- Tuned `mapred.reduce.tasks` to an appropriate value based on map output.
- Checked jobtracker to ensure “row container” spilling does not occur.
- Gave extra memory for mapjoins like broadcast joins.
- Disabled `orc.compress` (file size will increase) and tuned `orc.row.index.stride`.
- Ensured the job ran in a single wave of mappers.

# Agenda

---

- Hive – What Is It Good For?
- Hive’s Architecture and SQL Compatibility
- Turning Hive Performance to 11
- **Get Data In and Out of Hive**
- Hive Security
- Project Stinger – Making Hive 100x Faster
- Connecting to Hive From Popular Tools

# Loading Data in Hive

---

- **Sqoop**

- Data transfer from external RDBMS to Hive.
- Sqoop can load data directly to/from HCatalog.

- **Hive LOAD**

- Load files from HDFS or local filesystem.
- Format must agree with table format.

- **Insert from query**

- CREATE TABLE AS SELECT or INSERT INTO.

- **WebHDFS + WebHCat**

- Load data via REST APIs.

# Optimized Sqoop Connectors

---

- **Current:**

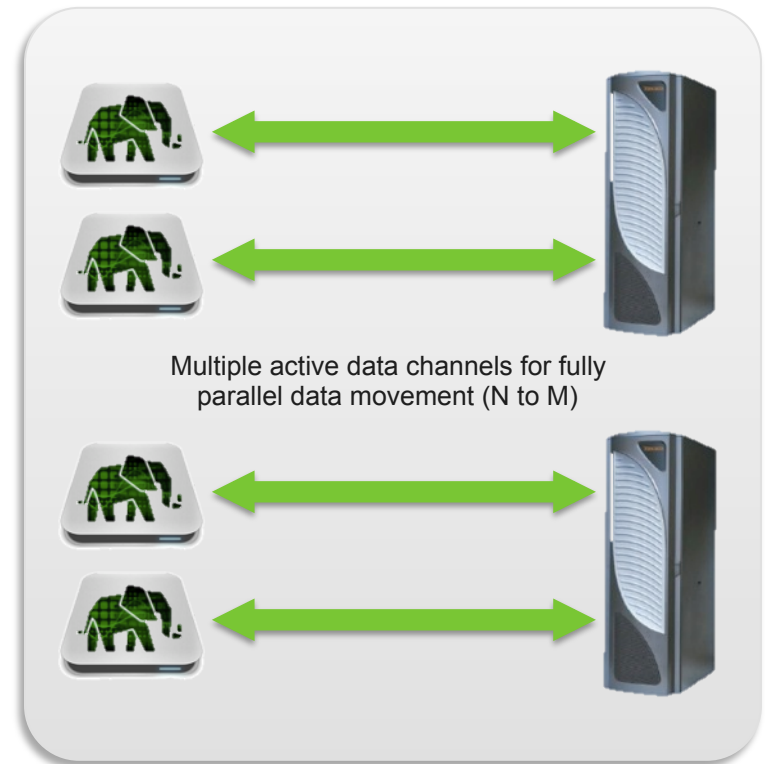
- Teradata
- Oracle
- Netezza
- SQL Server
- MySQL
- Postgres

- **Future:**

- Vertica

# High Performance Teradata Sqoop Connector

- **High performance Sqoop driver.**
- **Fully parallel data load using Enhanced FastLoad.**
- **Multiple readers and writers for efficient, wire-speed transfer.**
- **Copy data between Teradata and HDP Hive, HBase or HDFS.**



**Enhanced FastLoad developed in partnership with Teradata.  
Free to download & use with Hortonworks Data Platform.**

# ACID Properties

- **Data loaded into Hive partition- or table-at-a-time.**
  - No INSERT or UPDATE statement. No transactions.
- **Atomicity:**
  - Partition loads are atomic through directory renames in HDFS.
- **Consistency:**
  - Ensured by HDFS. All nodes see the same partitions at all times.
  - Immutable data = no update or delete consistency issues.
- **Isolation:**
  - Read committed with an exception for partition deletes.
  - Partitions can be deleted during queries. New partitions will not be seen by jobs started before the partition add.
- **Durability:**
  - Data is durable in HDFS before partition exposed to Hive.



# Handling Semi-Structured Data

---

- **Hive supports arrays, maps, structs and unions.**
- **SerDes map JSON, XML and other formats natively into Hive.**

# Agenda

---

- Hive – What Is It Good For?
- Hive’s Architecture and SQL Compatibility
- Turning Hive Performance to 11
- Get Data In and Out of Hive
- **Hive Security**
- Project Stinger – Making Hive 100x Faster
- Connecting to Hive From Popular Tools

# Hive Authorization

---

- **Hive provides Users, Groups, Roles and Privileges**
- **Granular permissions on tables, DDL and DML operations.**
- **Not designed for high security:**
  - On non-kerberized cluster, up to the client to supply their user name.
  - Suitable for preventing accidental data loss.

# HiveServer2

---

- **HiveServer2 is a gateway / JDBC / ODBC endpoint Hive clients can talk to.**
- **Supports secure and non-secure clusters.**
- **DoAs support allows Hive query to run as the requester.**
- **(Coming Soon) LDAP authentication.**

# Agenda

---

- Hive – What Is It Good For?
- Hive's Architecture and SQL Compatibility
- Turning Hive Performance to 11
- Get Data In and Out of Hive
- Hive Security
- **Project Stinger – Making Hive 100x Faster**
- Connecting to Hive From Popular Tools

# Roadmap to 100x Faster



## Hive

### Base Optimizations

Generate simplified DAGs  
Join Improvements



### Vector Query Engine

Optimized for modern  
processor architectures

### Query Planner

Intelligent Cost-Based  
Optimizer

## Hadoop

-  Phase 1
-  Phase 2
-  Phase 3

### Buffer Caching

Cache accessed data  
Optimized for vector engine

### Tez Service

Pre-warmed Containers  
Low-latency dispatch



### ORCFile

Column Store  
High Compression  
Predicate / Filter Pushdowns

### Tez

Express data processing  
tasks more simply  
Eliminate disk writes

# Phase 1 Improvements

Path to Making Hive 100x Faster

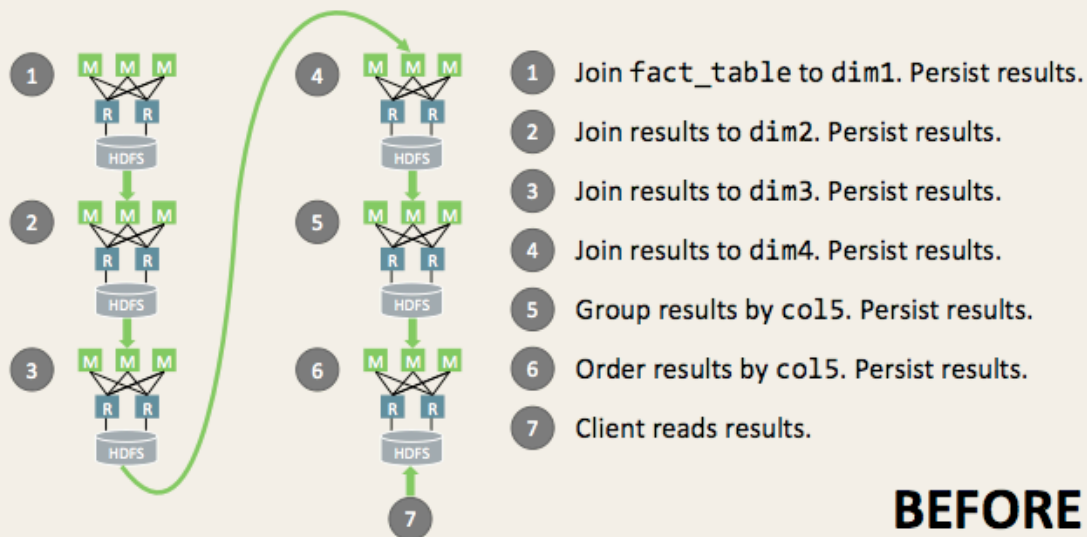
# Join Optimizations

- **Performance Improvements in Hive 0.11:**
- **New Join Types added or improved in Hive 0.11:**
  - In-memory Hash Join: Fast for fact-to-dimension joins.
  - Sort-Merge-Bucket Join: Scalable for large-table to large-table joins.
- **More Efficient Query Plan Generation**
  - Joins done in-memory when possible, saving map-reduce steps.
  - Combine map/reduce jobs when GROUP BY and ORDER BY use the same key.
- **More Than 30x Performance Improvement for Star Schema Join**



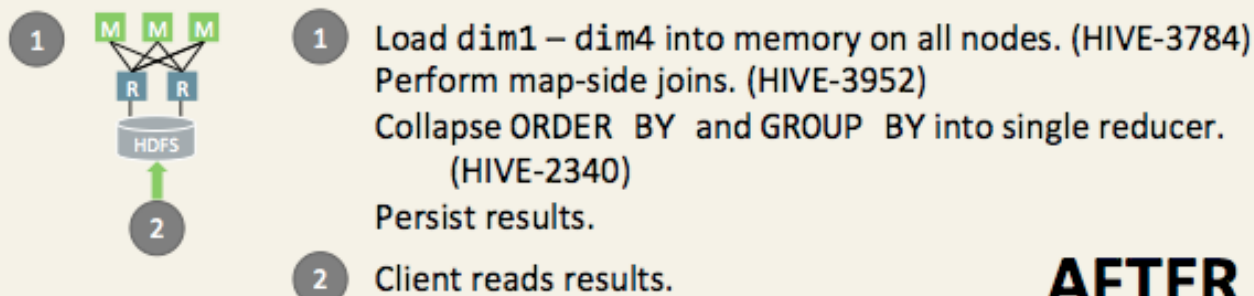
# Star Schema Join Improvements in 0.11

## Star Schema Join: Hive 0.10 without hints.



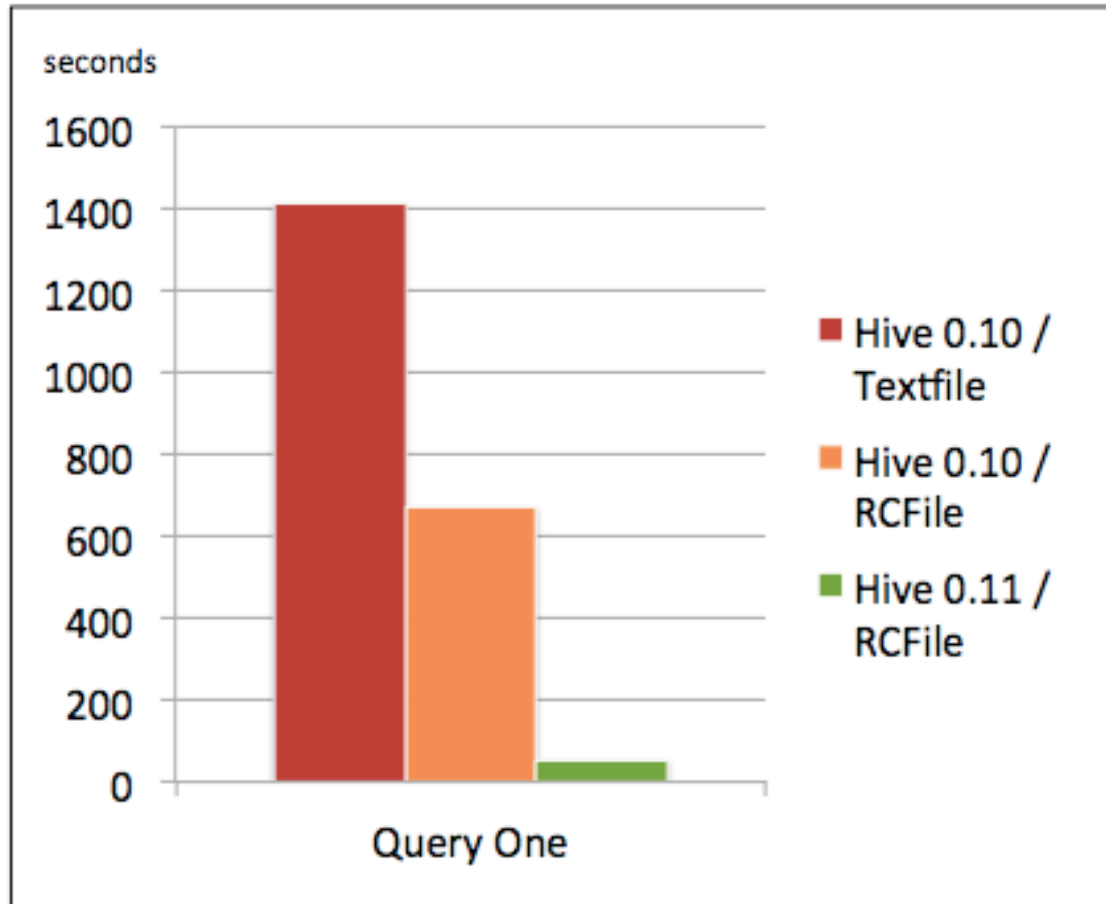
**BEFORE**

## Star Schema Join: Hive 0.11 without hints.



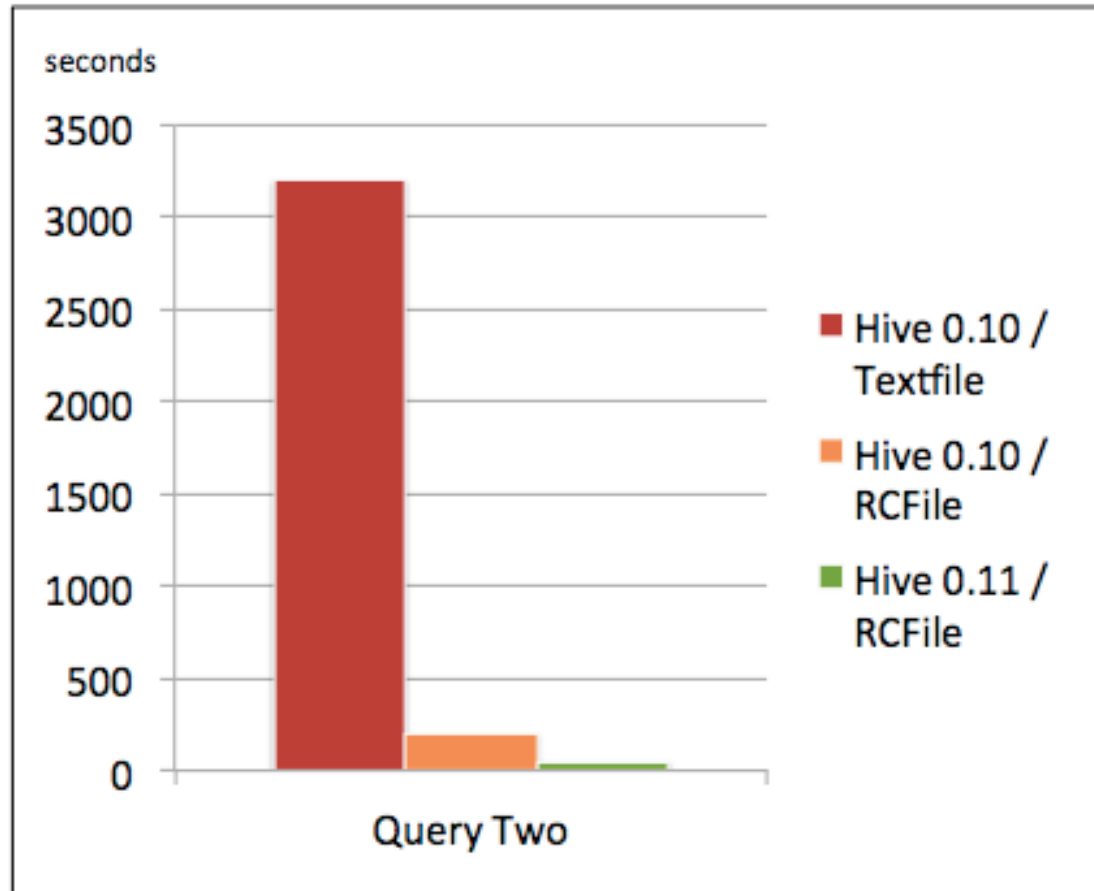
**AFTER**

# Hive 0.11 Star Schema Join Performance



In-Memory Hash Join

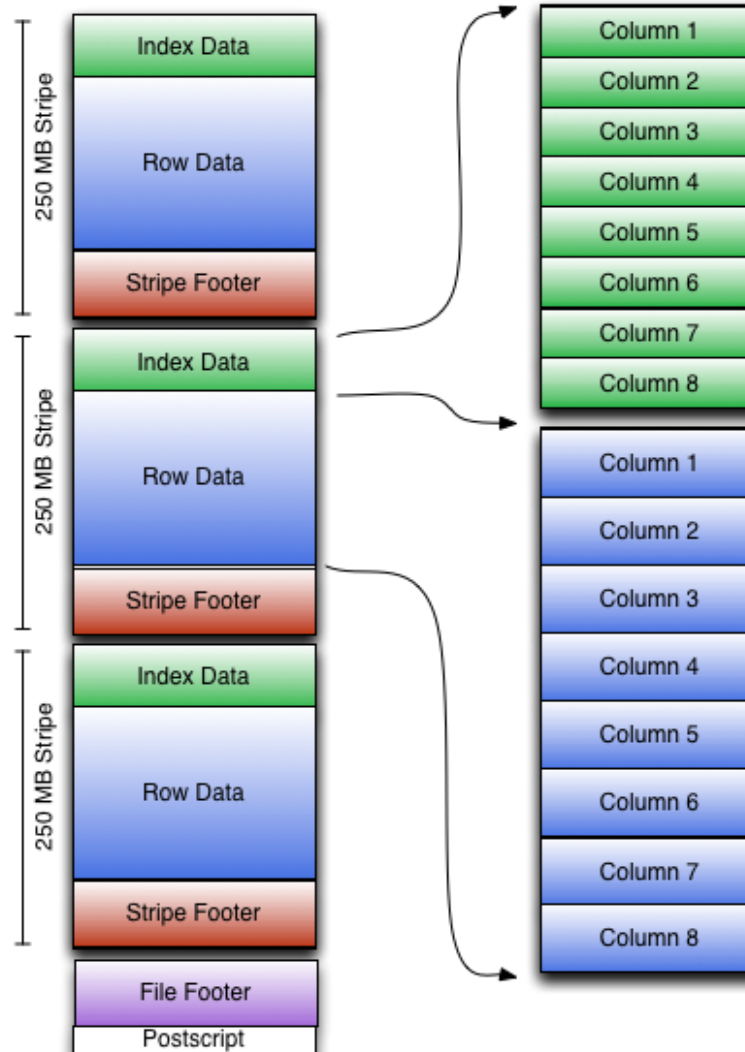
# Hive 0.11 SMB Join Improvements



Sort-Merge-Bucket Join

# ORCFile – Efficient Columnar Layout

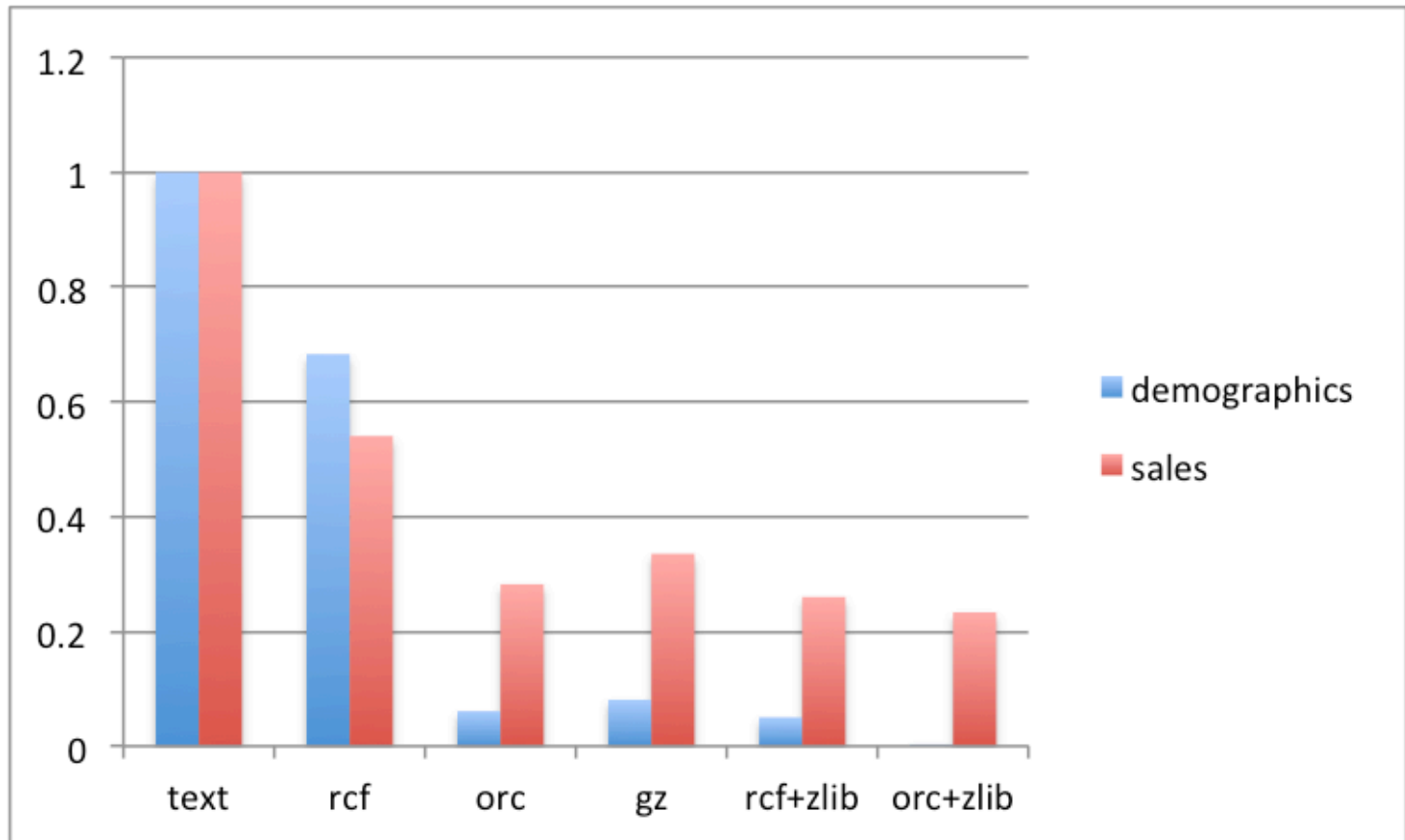
Large block size well suited for HDFS.



Columnar format arranges columns adjacent within the file for compression and fast access.

# ORCFile Provides High Compression

- Data set from TPC-DS



# Phase 2 & 3 Improvements

Path to Making Hive 100x Faster

# Vectorization

---

- **Designed for Modern Processor Architectures**
  - Make the most use of L1 and L2 cache.
  - Avoid branching whenever possible.
- **How It Works**
  - Process records in batches of 1,000 to maximize cache use.
  - Generate code on-the-fly to minimize branching.
- **What It Gives**
  - 30x+ improvement in number of rows processed per second.

# Other Runtime Optimizations

---

- **Optimized Query Planner**
  - Automatically determine optimal execution parameters.
- **Buffering**
  - Cache hot data in memory.



# Hadoop 2.0 - YARN

- **Re-architected Hadoop framework**

- Separation of Resource Management from MapReduce

- **Focus on scale and innovation**

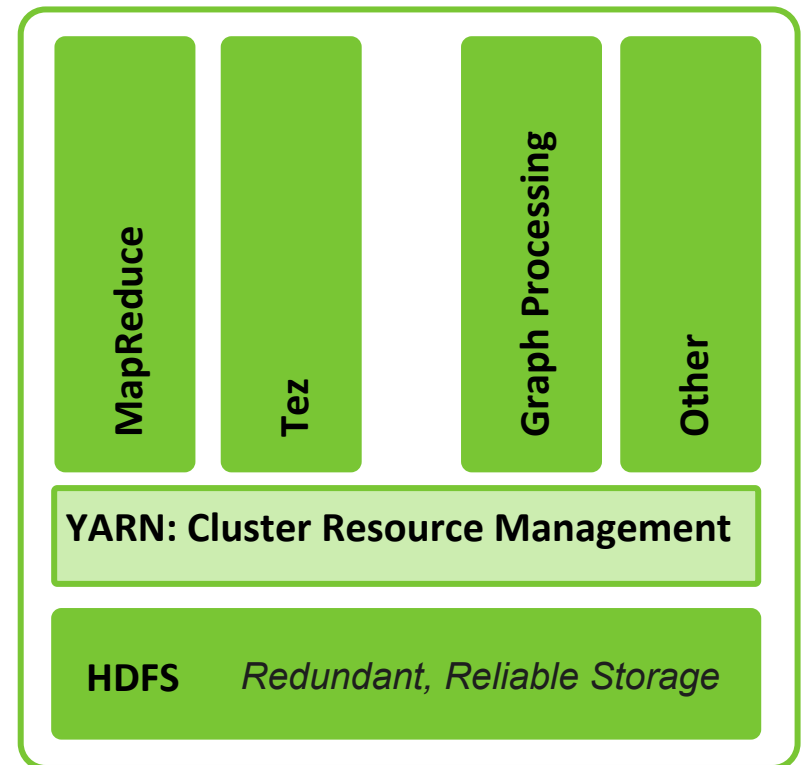
- Support 10,000+ computer clusters
- Extensible to encourage innovation

- **Next generation execution**

- Improves MapReduce performance

- **Supports new frameworks beyond MapReduce**

- Low latency, streaming, etc
- Do more with a single Hadoop cluster



# Tez (“Speed”)

---

- **What is it?**

- A data processing framework as an *alternative* to MapReduce
- A new incubation project in the ASF

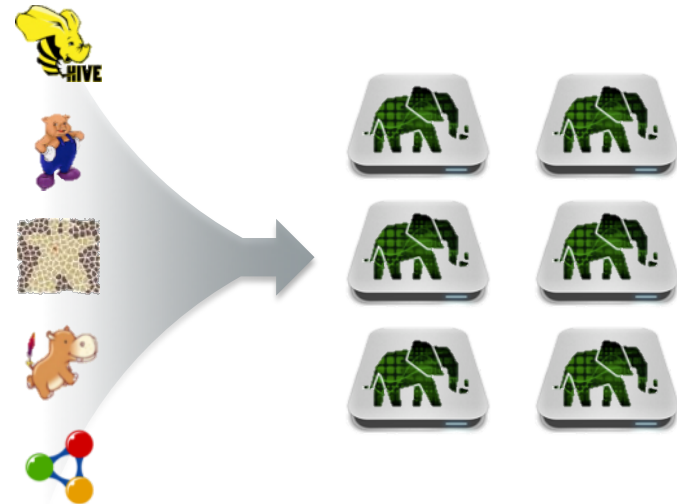
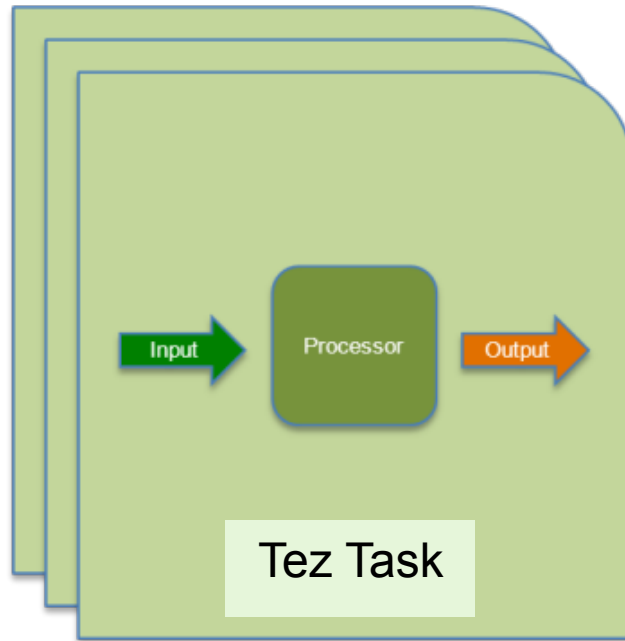
- **Who else is involved?**

- 22 contributors: Hortonworks (13), Facebook, Twitter, Yahoo, Microsoft

- **Why does it matter?**

- Widens the platform for Hadoop use cases
- Crucial to improving the performance of low-latency applications
- Core to the Stinger initiative
- Evidence of Hortonworks leading the community in the evolution of Enterprise Hadoop

# Tez on YARN: Going Beyond Batch



## Tez Optimizes Execution

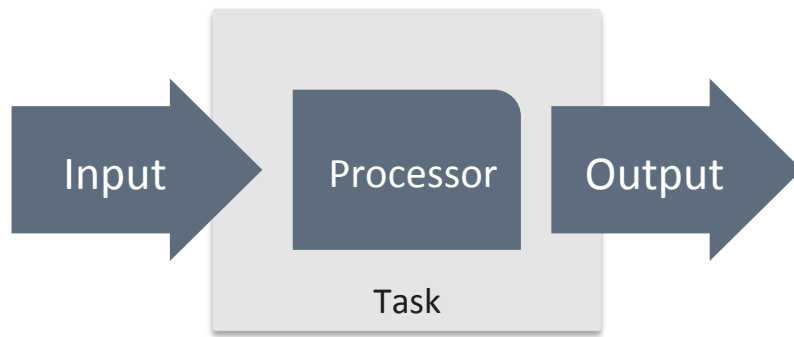
New runtime engine for more efficient data processing

## Always-On Tez Service

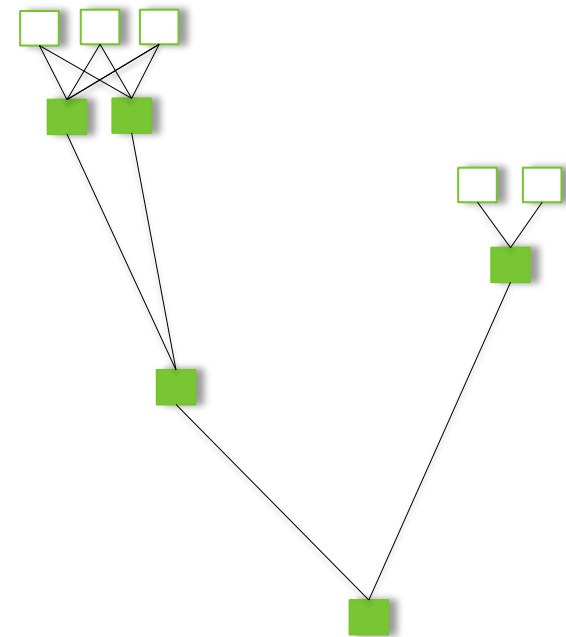
Low latency processing for all Hadoop data processing

# Tez - Core Idea

## Task with pluggable *Input*, *Processor* and *Output*

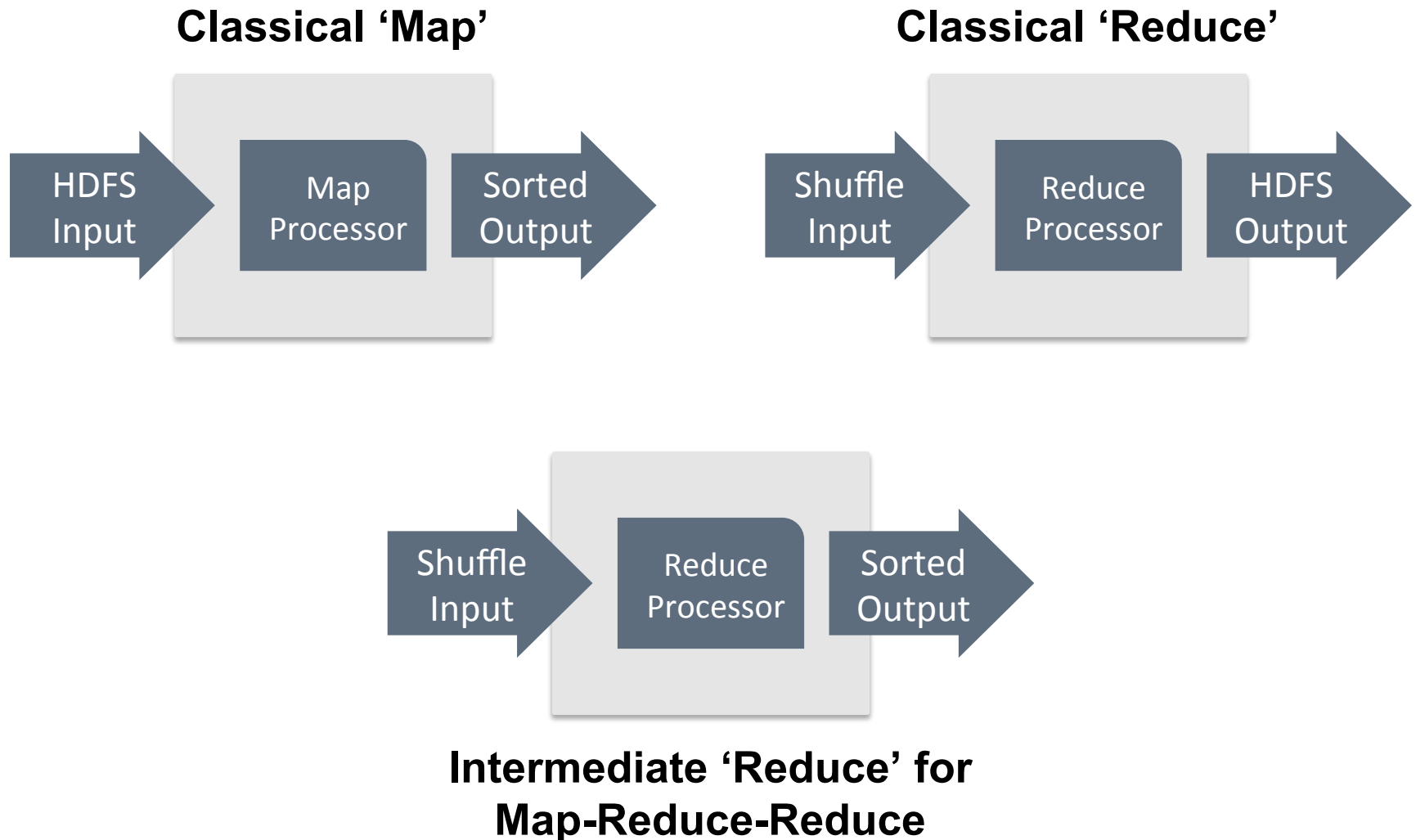


Tez Task -  $\langle$ Input, Processor, Output $\rangle$



## YARN ApplicationMaster to run DAG of Tez Tasks

# Tez: Building blocks for scalable data processing

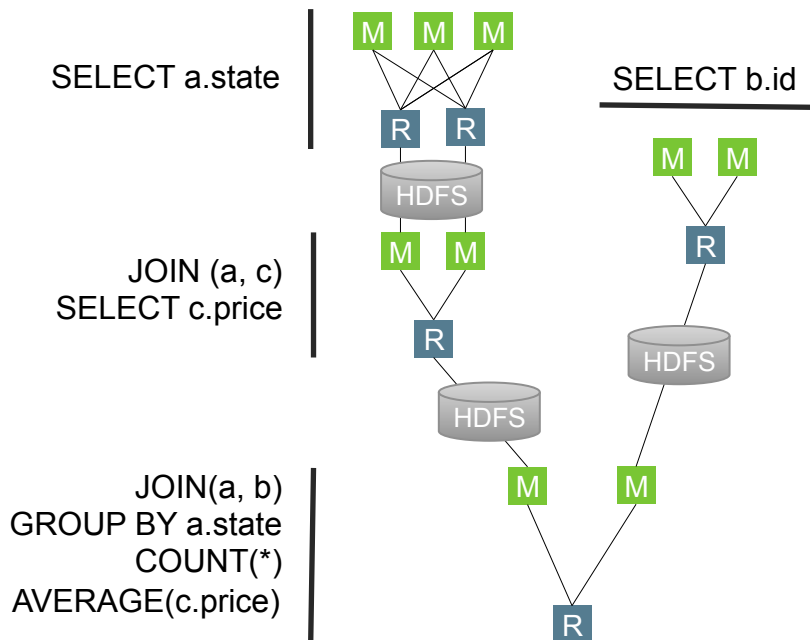


# Hive/MR versus Hive/Tez

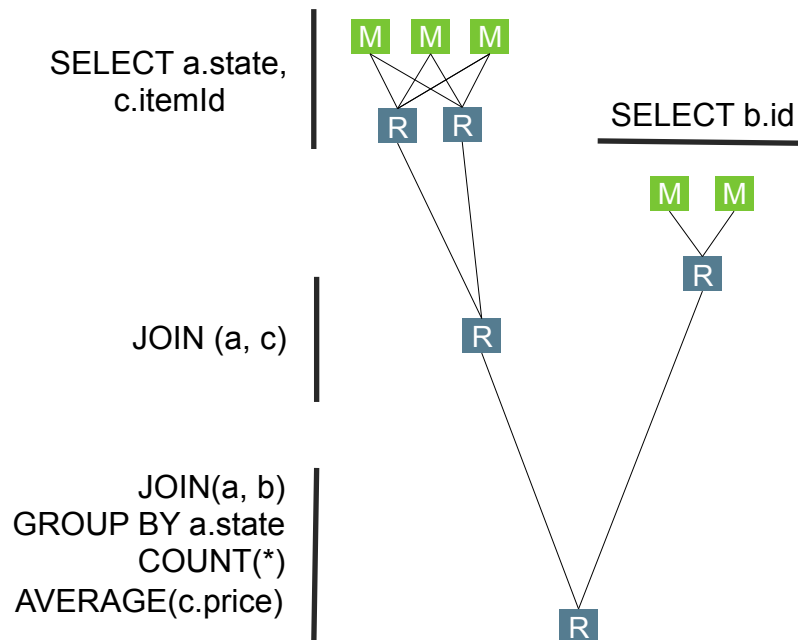
```
SELECT a.state, COUNT(*), AVERAGE(c.price)
  FROM a
 JOIN b ON (a.id = b.id)
 JOIN c ON (a.itemId = c.itemId)
 GROUP BY a.state
```

Tez avoids  
unneded writes to  
HDFS

## Hive – MR



## Hive – Tez



# Tez Service

- **Map/Reduce Query Startup Is Expensive**
- **Solution**
  - Tez Service
    - Hot containers ready for immediate use
    - Removes task and job launch overhead (~5s – 30s)
  - Hive
    - Submits query plan directly to Tez Service
  - ***Native Hadoop service, not ad-hoc***

# Agenda

---

- Hive – What Is It Good For?
- Hive’s Architecture and SQL Compatibility
- Turning Hive Performance to 11
- Get Data In and Out of Hive
- Hive Security
- Project Stinger – Making Hive 100x Faster
- **Connecting to Hive From Popular Tools**



# Hive: The De-Facto SQL Interface for Hadoop



# Connectivity

---

- **JDBC – Included with Hive.**
- **ODBC – Free driver available at [hortonworks.com](http://hortonworks.com).**
- **WebHCat – Run jobs using a simple REST interface.**
- **BI Ecosystem – Most popular BI tools support Hive.**

# Learn More with Hortonworks HDP Sandbox



Go from zero to  
BIG DATA in 15 minutes



Learn to connect to Hadoop with  
popular BI tools.

Built-in tutorials show how to connect  
to Hive with ODBC and Excel.