A Perfect Hive Query For A Perfect Meeting

Adam Kawa

Data Engineer @ Spotify

A deal was made!

~5:00 AM, June 16th, 2013

~5:00 AM, June 16th, 2013

- End of our Summer Jam party
- Organized by Martin Lorentzon

Martin Lorentzon

- Co-founder and Chairman of Spotify

Adam Kawa

- Data Engineer at Spotify

The Deal

Martin will invite Adam
and Timbuktu, my favourite Swedish artist,
for a beer or coke or whatever to drink *

* If Adam proves somehow that
he is the biggest fan of Timbuktu in his home country

Timbuktu

- Actually, Jason Michael Bosak Diakité
- A top Swedish rap and reggae artist
  - My favourite song - http://bit.ly/1htAKhM
- Does not know anything about the deal!

The Rules

Question

Who is the biggest fan of Timbuktu in Poland?

The Rules

Question

Who is the biggest fan of Timbuktu in Poland?

Answers

A person who has streamed Timbuktu's songs at Spotify the most frequently!

Data will tell the truth!

Hive Query

- Follow tips and best practices related to
  - Testing
  - Sampling
  - Troubleshooting
  - Optimizing
  - Executing

Additional Requirement

Hive query
should be faster
than my potential meeting with Timbuktu

Why?

Re-run the query during our meeting

Friends

- YARN on 690 nodes
- Hive
  - MRv2
  - Join optimizations
  - ORC
  - Tez
  - Vectorization
  - Table statistics

Introduction

Datasets

TRACK

json_data	...	date	track_id
{"album":{"artistname":"Coldplay"}}	...	20140415	t1
{"album":{"artistname":"Timbuktu"}}	...	20140415	t2
{"album":{"artistname":"Timbuktu"}}	...	20140415	t3

USER

user_id	country	...	date
u1	PL	...	20140415
u2	PL	...	20140415
u3	SE	...	20140415

STREAM

track_id	...	user_id	...	date
t1	...	u1	...	20130311
t3	...	u1	...	20130622
t3	...	u2	...	20131209
t2	...	u2	...	20140319
t1	...	u3	...	20140415

STREAM

- Information about songs streamed by users
- 54 columns
- 25.10 TB of compressed data since Feb 12th, 2013

track_id	...	...	user_id	date
t1	...	...	u1	20130311
t3	...	...	u1	20130622
t3	...	...	u2	20131209
t2	...	...	u2	20140319
t1	...	...	u3	20140415

USER

- Information about users
- 40 columns
- 10.5 GB of compressed data (2014-04-15)

user_id	country	...	date
u1	PL	...	20140415
u2	PL	...	20140415
u3	SE	...	20140415

TRACK

- Metadata information about tracks
- 5 columns
  - But `json_data` contains dozen or so fields
- 15.4 GB of data (2014-04-15)

track_id	json_data	...	date
t1	{"album":{"artistname":"Coldplay"}}	...	20140415
t2	{"album":{"artistname":"Timbuktu"}}	...	20140415
t3	{"album":{"artistname":"Timbuktu"}}	...	20140415

Tables' Properties

✓ Avro and a custom serde
✓ `DEFLATE` as a compression codec
✓ Partitioned by day and/or hour

✗ Bucketed
✗ Indexed

HiveQL

Query

Query

```
FROM   stream s
```

Query

```
FROM    stream s
        JOIN track t   ON t.track_id = s.track_id
```

Query

```
FROM    stream s
        JOIN track t   ON t.track_id = s.track_id
        JOIN user u        ON u.user_id = s.user_id
```

Query

```
FROM    stream s
        JOIN track t   ON t.track_id = s.track_id
        JOIN user u        ON u.user_id = s.user_id

WHERE   lower(get_json_object(t.json_data, '$.album.artistname'))
            = 'timbuktu'
```

Query

```
FROM    stream s
        JOIN track t   ON t.track_id = s.track_id
        JOIN user u        ON u.user_id = s.user_id

WHERE   lower(get_json_object(t.json_data, '$.album.artistname'))
            = 'timbuktu'
        AND u.country = 'PL'
```

Query

```
FROM   stream s
       JOIN track t   ON t.track_id = s.track_id
       JOIN user u       ON u.user_id = s.user_id

WHERE  lower(get_json_object(t.json_data, '$.album.artistname'))
           = 'timbuktu'
       AND u.country = 'PL'
       AND s.date BETWEEN 20130212 AND 20140415
```

Query

```
FROM    stream s
        JOIN track t   ON t.track_id = s.track_id
        JOIN user u        ON u.user_id = s.user_id

WHERE   lower(get_json_object(t.json_data, '$.album.artistname'))
            = 'timbuktu'
        AND u.country = 'PL'
        AND s.date BETWEEN 20130212 AND 20140415
        AND u.date = 20140415 AND t.date = 20140415
```

Query

```
SELECT s.user_id, COUNT(*) AS count

FROM   stream s
       JOIN track t   ON t.track_id = s.track_id
       JOIN user u       ON u.user_id = s.user_id

WHERE  lower(get_json_object(t.json_data, '$.album.artistname'))
           = 'timbuktu'
       AND u.country = 'PL'
       AND s.date BETWEEN 20130212 AND 20140415
       AND u.date = 20140415 AND t.date = 20140415

GROUP  BY s.user_id
```

Query

```
SELECT s.user_id, COUNT(*) AS count

FROM   stream s
       JOIN track t   ON t.track_id = s.track_id
       JOIN user u       ON u.user_id = s.user_id

WHERE  lower(get_json_object(t.json_data, '$.album.artistname'))
           = 'timbuktu'
       AND u.country = 'PL'
       AND s.date BETWEEN 20130212 AND 20140415
       AND u.date = 20140415 AND t.date = 20140415

GROUP  BY s.user_id
ORDER  BY count DESC
LIMIT  100;
```

Query

```
SELECT s.user_id, CO

FROM   stream s
       JOIN track t
       JOIN user u

WHERE  lower(get_json_object(t.json_data, '$.album.artistname'))
           = 'timbuktu'
       AND u.country = 'PL'
       AND s.date BETWEEN 20130212 AND 20140415
       AND u.date = 20140415 AND t.date = 20140415

GROUP  BY s.user_id
ORDER  BY count DESC
LIMIT  100;
```

A line where
I may have a bug ?!

HiveQL

Unit Testing

hive_test

```java
public void testExecute() throws Exception {

  /* Use the Hadoop filesystem API to create a
  data file */
  Path p = new Path(this.ROOT_DIR, "afile");
  FSDataOutputStream o = this.getFileSystem().create(p);
  BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(o));
  bw.write("1\n");
  bw.write("2\n");
  bw.close();

  /* ServiceHive is a component that connections
  to an embedded or network HiveService based
  on the constructor used */
  ServiceHive sh = new ServiceHive();

  /* We can now interact through the HiveService
  and assert on results */
  sh.client.execute("create table atest (num int)");
  sh.client.execute("load data local inpath '"
    + p.toString() + "' into table atest");
  sh.client.execute("select count(1) as cnt from atest");
  String row = sh.client.fetchOne();
  assertEquals("2", row);
  sh.client.execute("drop table atest");

}
```

Verbose and complex Java code

Testing Hive Queries

- Hive analyst usually is not Java developer
- Hive queries are often ad-hoc
  - No big incentives to unit test

Beetest

- Easy unit testing of Hive queries locally
  - Implemented by me
  - Available at github.com/kawaa/Beetest

Beetest

- A unit test consists of
  - `select.hql` - a query to test

Beetest

- A unit test consists of
  - `select.hql` - a query to test
  - `table.ddl` - schemas of input tables

Beetest

- A unit test consists of
  - `select.hql` - a query to test
  - `table.ddl` - schemas of input tables
  - text files with input data

Beetest

- A unit test consists of
  - `select.hql` - a query to test
  - `table.ddl` - schemas of input tables
  - text files with input data
  - `expected.txt` - expected output

Beetest

- A unit test consists of
  - `select.hql` - a query to test
  - `table.ddl` - schemas of input tables
  - text files with input data
  - `expected.txt` - expected output
  - (optional) `setup.hql` - any initialization query

table.ddl

```
stream(track_id String, user_id STRING, date BIGINT)
user(user_id STRING, country STRING, date BIGINT)
track(track_id STRING, json_data STRING, date BIGINT)
```

table.ddl

```
stream(track_id String, user_id STRING, date BIGINT)
user(user_id STRING, country STRING, date BIGINT)
track(track_id STRING, json_data STRING, date BIGINT)
```

For Each Line

1. Creates a *table* with a given schema
   - In local Hive database called `beetest`

table.ddl

```
stream(track_id String, user_id STRING, date BIGINT)
user(user_id STRING, country STRING, date BIGINT)
track(track_id STRING, json_data STRING, date BIGINT)
```

For Each Line

1. Creates a *table* with a given schema
   - In local Hive database called `beetest`
2. Loads *table.txt* file into the table
   - A list of files can be also explicitly specified

Input Files

track.txt

```
t1 {"album":{"artistname":"Coldplay"}}  20140415
t2 {"album":{"artistname":"Timbuktu"}}  20140415
t3 {"album":{"artistname":"Timbuktu"}}  20140415
```

Input Files

track.txt

```
t1 {"album":{"artistname":"Coldplay"}}  20140415
t2 {"album":{"artistname":"Timbuktu"}}  20140415
t3 {"album":{"artistname":"Timbuktu"}}  20140415
```

```
u1 PL 20140415
u2 PL 20140415
u3 SE 20140415
```

user.txt

Input Files

track.txt

```
t1 {"album":{"artistname":"Coldplay"}}  20140415
t2 {"album":{"artistname":"Timbuktu"}}  20140415
t3 {"album":{"artistname":"Timbuktu"}}  20140415
```

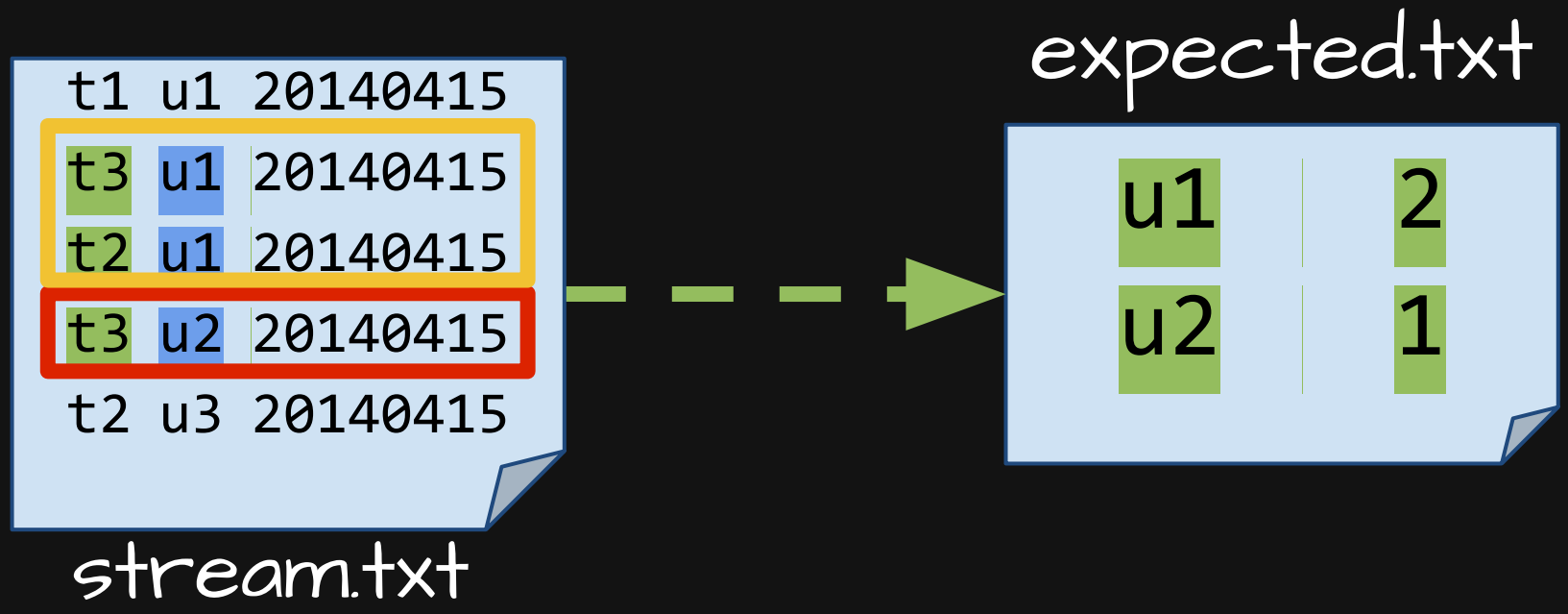
```
u1 PL 20140415
u2 PL 20140415
u3 SE 20140415
```

user.txt

```
t1 u1 20140415
t3 u1 20140415
t2 u1 20140415
t3 u2 20140415
t2 u3 20140415
```

stream.txt

Expected Output File



Running Unit Test

```
$ ./run-test.sh timbuktu
```

Running Unit Test

```
$ ./run-test.sh timbuktu
```

```
INFO: Generated query filename: /tmp/beetest-test-1934426026-query.hql
INFO: Running: hive --config local-config -f /tmp/beetest-test-
1934426026-query.hql
```

Running Unit Test

```
$ ./run-test.sh timbuktu

INFO: Generated query filename: /tmp/beetest-test-1934426026-query.hql
INFO: Running: hive --config local-config -f /tmp/beetest-test-
1934426026-query.hql
INFO: Loading data to table beetest.stream
…
INFO: Total MapReduce jobs = 2
INFO: Job running in-process (local Hadoop)
…
```

Running Unit Test

```
$ ./run-test.sh timbuktu

INFO: Generated query filename: /tmp/beetest-test-1934426026-query.hql
INFO: Running: hive --config local-config -f /tmp/beetest-test-
1934426026-query.hql
INFO: Loading data to table beetest.stream
...
INFO: Total MapReduce jobs = 2
INFO: Job running in-process (local Hadoop)
...
INFO: Execution completed successfully
INFO: Table beetest.output_1934426026 stats: ...
INFO: Time taken: 34.605 seconds
```

Running Unit Test

```
$ ./run-test.sh timbuktu

INFO: Generated query filename: /tmp/beetest-test-1934426026-query.hql
INFO: Running: hive --config local-config -f /tmp/beetest-test-
1934426026-query.hql
INFO: Loading data to table beetest.stream
...
INFO: Total MapReduce jobs = 2
INFO: Job running in-process (local Hadoop)
...
INFO: Execution completed successfully
INFO: Table beetest.output_1934426026 stats: ...
INFO: Time taken: 34.605 seconds
INFO: Asserting: timbuktu/expected.txt and /tmp/beetest-test-1934426026-
output_1934426026/000000_0
INFO: Test passed!
```

Bee test

Be happy !

HiveQL

Sampling

Sampling In Hive

- `TABLESAMPLE` can generate samples of
  - buckets
  - HDFS blocks
  - first N records from each input split

TABLESAMPLE Limitations

✗ Bucket sampling
- Table must be bucketized by a given column

✗ HDFS block sampling
- Only `CombineHiveInputFormat` is supported

Rows TABLESAMPLE

```
SELECT * FROM user TABLESAMPLE(K ROWS)
SELECT * FROM track TABLESAMPLE(L ROWS)
SELECT * FROM stream TABLESAMPLE(M ROWS)

SELECT … stream JOIN track ON … JOIN user ON …
```

✗ No guarantee that `JOIN` returns anything

DataFu And Pig

- DataFu + Pig offer interesting sampling algorithms
  - Weighted Random Sampling
  - Reservoir Sampling
  - Consistent Sampling By Key
- HCatalog can provide integration

Consistent Sampling By Key

```
DEFINE SBK datafu.pig.sampling.SampleByKey('0.01');

   stream_s = FILTER stream BY SBK(user_id);
       user_s = FILTER user BY SBK(user_id);
```

Consistent Sampling By Key

```
DEFINE SBK datafu.pig.sampling.SampleByKey('0.01');

stream_s = FILTER stream BY SBK(user_id);
user_s = FILTER user BY SBK(user_id);
```

Threshold

```
stream_user_s = JOIN user_s BY user_id,
                     stream_s BY user_id;
```

✓ Guarantees that JOIN returns rows after sampling
- Assuming that the threshold is high enough

Consistent Sampling By Key

```
DEFINE SBK datafu.pig.sampling.SampleByKey('0.01');

   stream_s = FILTER stream BY SBK(user_id);
     user_s = FILTER user BY SBK(user_id);
```

Threshold

```
        user_s_pl = FILTER user_s BY country == 'PL';
stream_user_s_pl = JOIN user_s_pl BY user_id,
                        stream_s BY user_id;
```

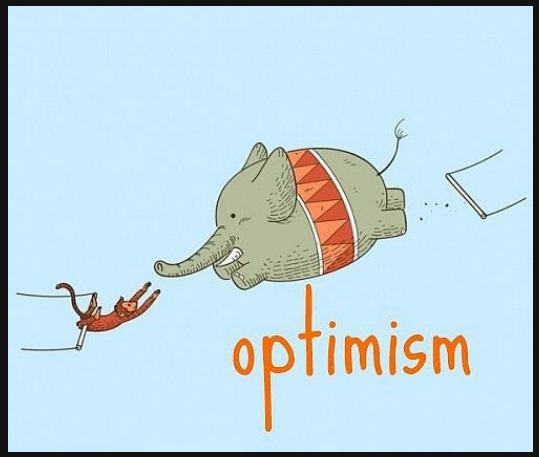
✗ Do not guarantee that Polish users will be included

What's Next?

✗ None of the sampling methods meet all of my requirements

Try and see

- Use `SampleByKey` and 1% of day's worth of data
  - Maybe it will be good enough?



Running Query

1. Finished successfully!
2. Returned 3 Polish users with 4 Timbuktu's tracks
   - My username was not included :(

?

HiveQL

Understanding The Plan

EXPLAIN

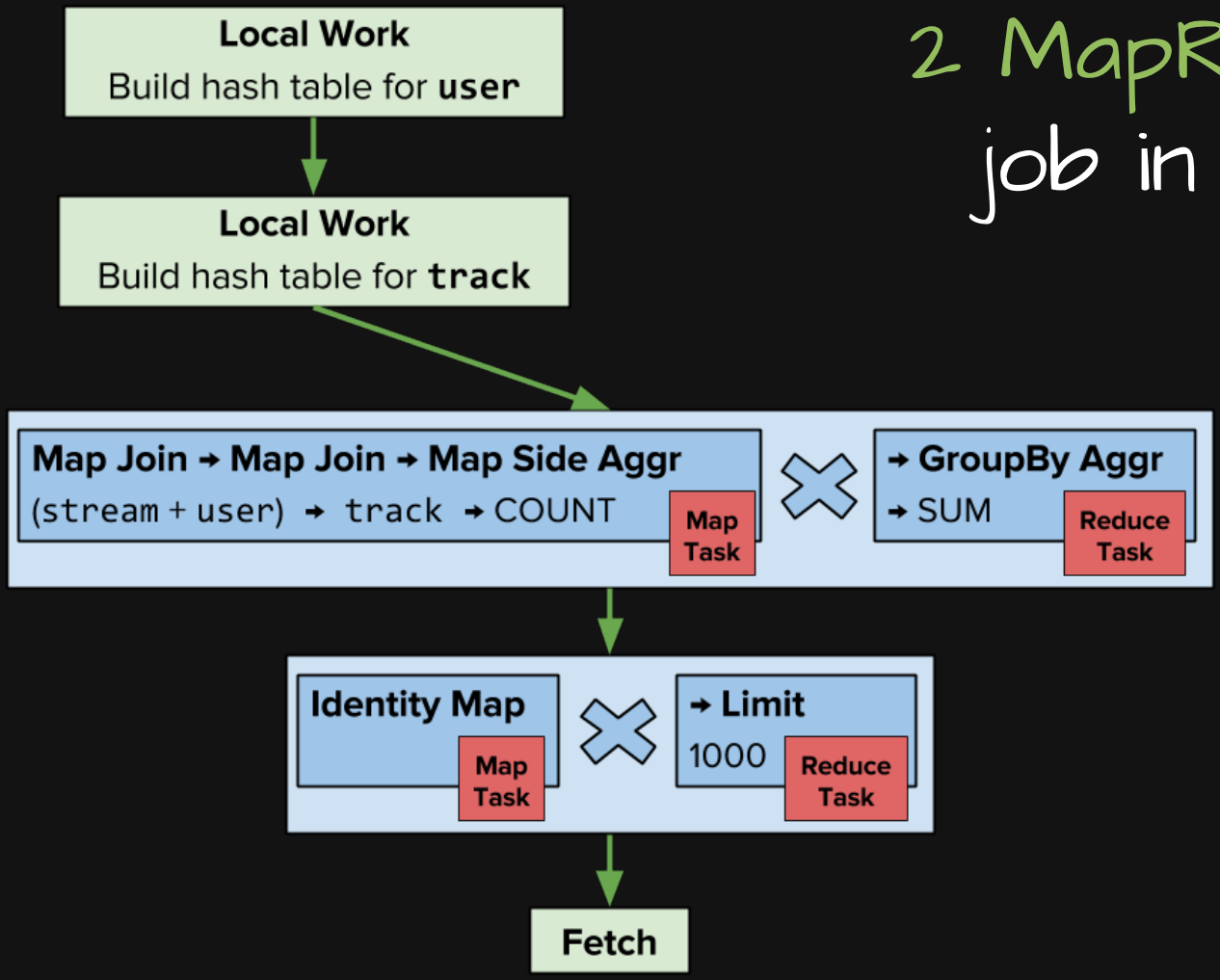
- Shows how Hive translates queries into MR jobs
  - Useful, but tricky to understand

```
STAGE DEPENDENCIES:
  Stage-12 is a root stage , consists of Stage-1
  Stage-1
  Stage-9 depends on stages: Stage-1 , consists of Stage-13, Stage-14, Stage-2
  Stage-13 has a backup stage: Stage-2
  Stage-7 depends on stages: Stage-13
  Stage-3 depends on stages: Stage-2, Stage-7, Stage-8
  Stage-4 depends on stages: Stage-3
  Stage-14 has a backup stage: Stage-2
  Stage-8 depends on stages: Stage-14
  Stage-2
  Stage-0 is a root stage

STAGE PLANS:
  Stage: Stage-12
    Conditional Operator

  Stage: Stage-1
    Map Reduce
      Alias -> Map Operator Tree:
        e
          TableScan
            alias: e
            Filter Operator
              predicate:
                  expr: ((20130212 <= dt) and (dt <= 20140415))
                  type: boolean
              Reduce Output Operator
                key expressions:
                      expr: trackid
                      type: string
                sort order: +
                Map-reduce partition columns:
                      expr: trackid
                      type: string
                tag: 0
                value expressions:
                      expr: username
                      type: string
        m
          TableScan
            alias: m
            Filter Operator

Stage: Stage-9
  Conditional Operator

Stage: Stage-13
  Map Reduce Local Work
    Alias -> Map Local Tables:
      mu
        Fetch Operator
          limit: -1
    Alias -> Map Local Operator Tree:
      mu
        TableScan
          alias: mu
          Filter Operator
            predicate:
                expr: ((country = 'PL') and (dt = 20140415))
                type: boolean
            HashTable Sink Operator
              condition expressions:
                0
                1 {username}
              handleSkewJoin: false
              keys:
                0 [Column[_col5]]
                1 [Column[username]]
              Position of Big Table: 0

Stage: Stage-7
  Map Reduce
    Alias -> Map Operator Tree:
      $INTNAME
          Map Join Operator
            condition map:
                 Inner Join 0 to 1
            condition expressions:
              0
              1 {username}
            handleSkewJoin: false
            keys:
              0 [Column[_col5]]
              1 [Column[username]]
            outputColumnNames: _col66
            Position of Big Table: 0
            Select Operator
              expressions:
```

`EXPLAIN` The Query

- 330 lines
- 11 stages
  - Conditional operators and backup stages
  - MapReduce jobs, Map-Only jobs
  - MapReduce Local Work
- 4 MapReduce jobs
  - Two of them needed for `JOIN`s

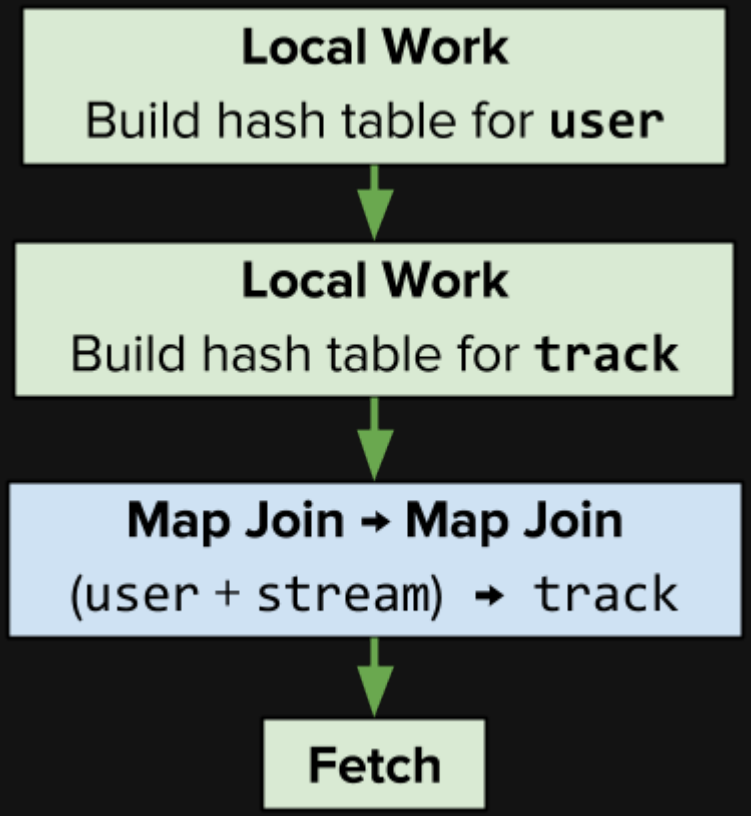
Optimized Query



2 MapReduce job in total

No Conditional Map Join

```
hive.auto.convert.join.noconditionaltask.size
    > filesize(user) + filesize(track)
```



Runs many Map joins in a Map-Only job [HIVE-3784]

Map Join Procedure

1. Local Work
   - Fetch records from small table(s) from HDFS
   - Build hash table
   - If not enough memory, then abort

Map Join Procedure

1. Local Work
   - Fetch records from small table(s) from HDFS
   - Build hash table
   - If not enough memory, then abort
2. MapReduce Driver
   - Add hash table to the distributed cache

Map Join Procedure

1. Local Work
   - Fetch records from small table(s) from HDFS
   - Build hash table
   - If not enough memory, then abort
2. MapReduce Driver
   - Add hash table to the distributed cache
3. Map Task
   - Read hash table from the distributed cache
   - Stream through, supposedly, the largest table
   - Match records and write to output

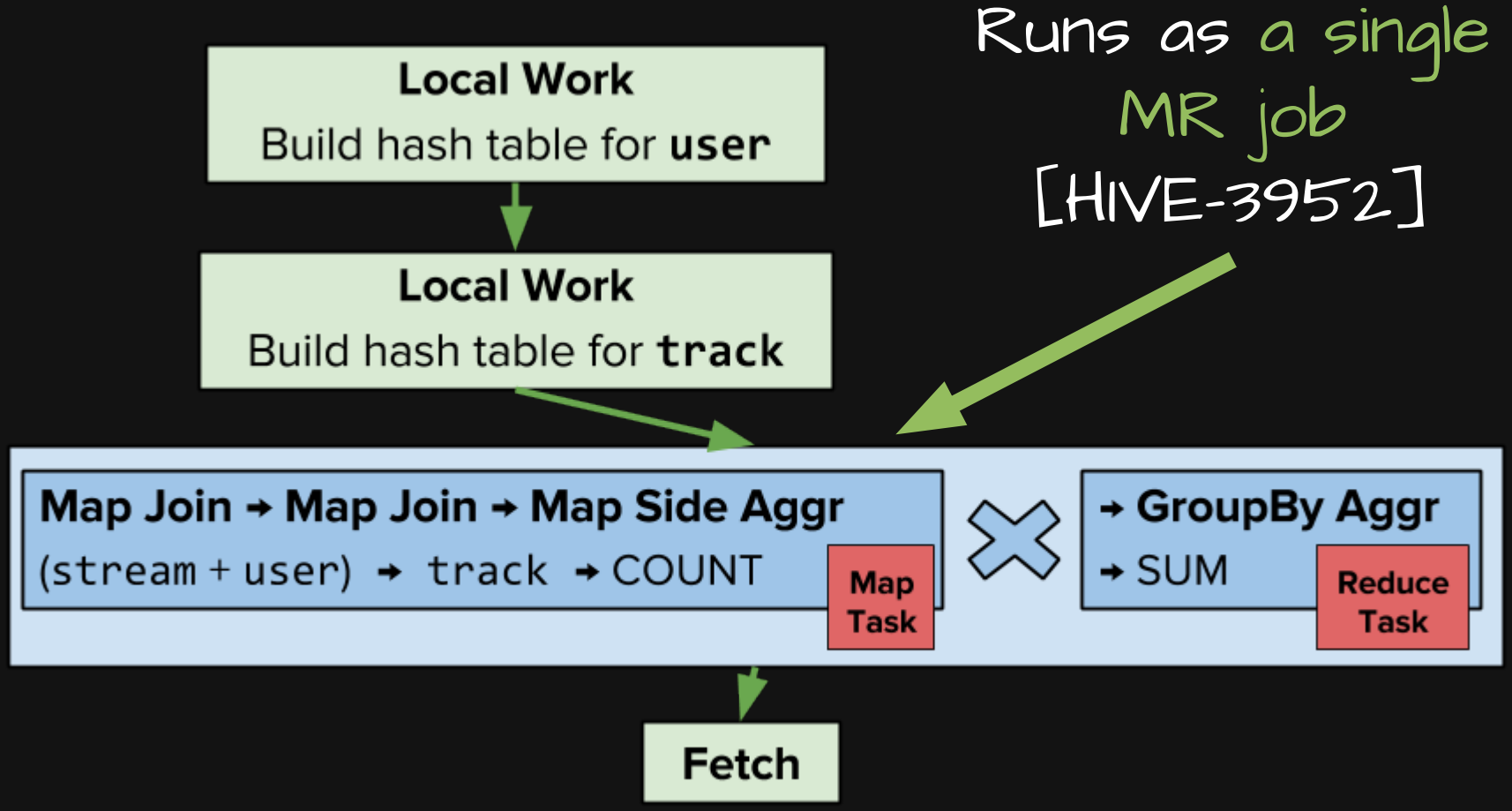
Map Join Procedure

1. Local Work
   - Fetch records from small table(s) from HDFS
   - Build hash table
   - If not enough memory, then abort
2. MapReduce Driver
   - Add hash table to the distributed cache
3. Map Task
   - Read hash table from the distributed cache
   - Stream through, supposedly, the largest table
   - Match records and write to output
4. No Reduce Task

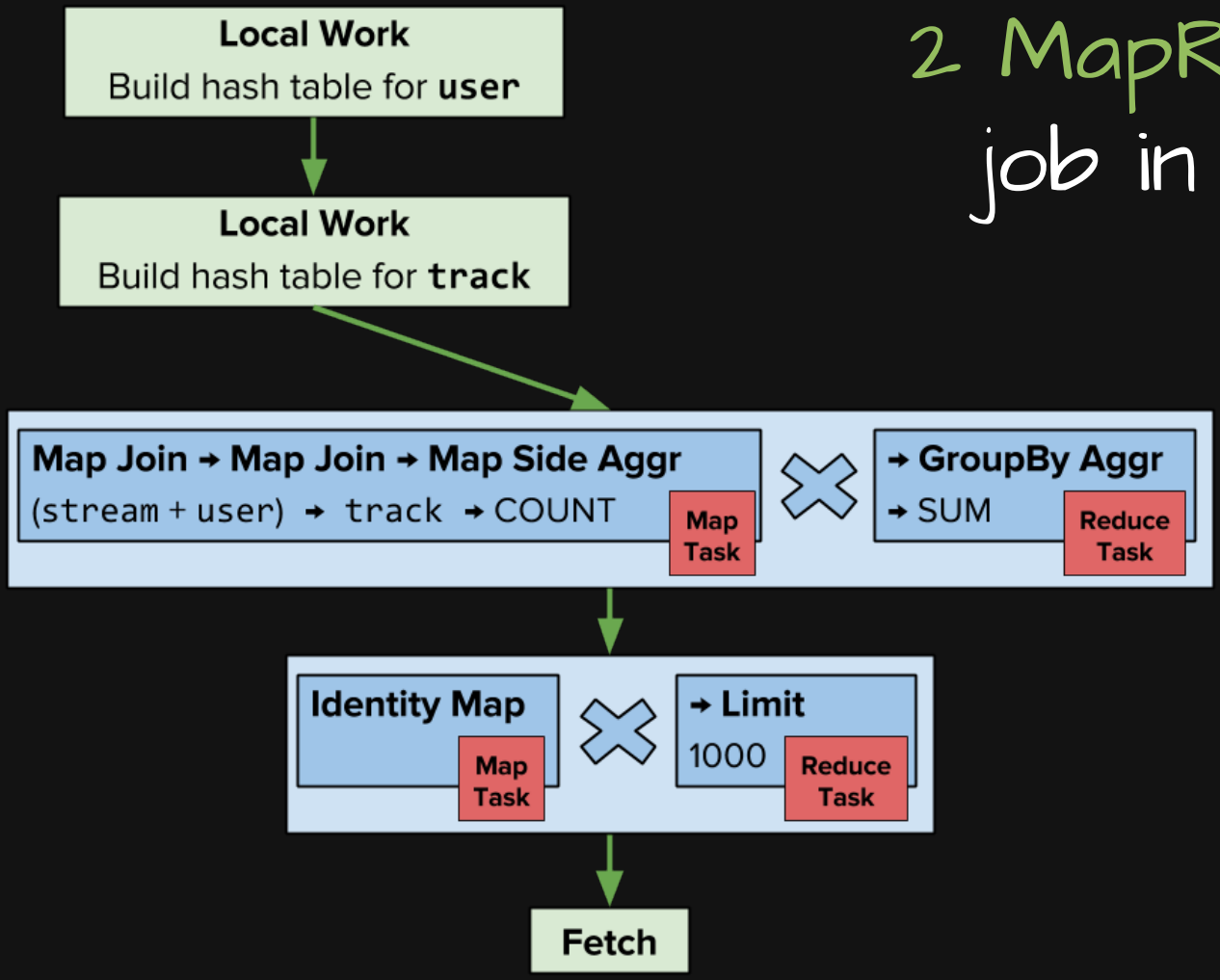
Map Join Cons

- Table has to be downloaded from HDFS
  - Problem if this step takes long
- Hash table has to be uploaded to HDFS
  - Replicated via the distributed cache
  - Fetched by hundred of nodes

Grouping After Joining



Optimized Query



2 MapReduce job in total

HiveQL

Running At Scale

Day's Worth Of Data

```
2014-05-05 21:21:13Starting to launch local task
to process map join;  maximum memory = 2GB

^Ckawaa@sol:~/timbuktu$ date
Mon May  5 21:50:26 UTC 2014
```

Too Big Tables

✗ Dimension tables too big to read fast from HDFS

✓ Add a preprocessing step

- Apply projection and filtering in Map-only job
- Make a script ugly with intermediate tables

Day’s Worth Of Data

✓ 15s to build hash tables!

✗ 8m 23s to run 2 Map-only jobs to prepare tables

- “Investment”

Day's Worth Of Data

- Map tasks

```
FATAL [main] org.apache.hadoop.mapred.YarnChild:
Error running child : java.lang.
OutOfMemoryError: Java heap space
```

Possible Solution

- Increase JVM heap size
- Decrease the size of in-memory map output buffer
  - `mapreduce.task.io.sort.mb`

(Temporary) Solution

- Increase JVM heap size
- Decrease the size of in-memory map output buffer
  - `mapreduce.task.io.sort.mb`

My query generates small amount of intermediate data

- `MAP_OUTPUT_*` counters

Second Challenge

✅ Works!

❌ 2-3x slower than two Regular joins

Heavy Garbage Collection!

Metric		Regular Join	Map Join
The slowest job	Shuffled Bytes	12.5 GB	0.56 MB
	CPU	120 M	331 M
	GC	13 M	258 M
	GC / CPU	0.11	0.78
	Wall-clock	11m 47s	31m 30s

Fixing Heavy GC

- Increase the container and heap sizes
  - `4096` and `-Xmx3584m`

Metric		Regular Join	Map Join
The slowest job	GC / CPU	`0.11`	`0.03`
Query	CPU	`1d 17m`	`23h 8m`
	Wall-clock	`20m 15s`	`14m 42s`

Where To Optimize?

- 1st MapReduce Job
  - Heavy map phase
    - No GC issues any more
    - Tweak input split size
  - Lightweight shuffle
  - Lightweight reduce
    - Use 1-2 reduce tasks

Where To Optimize?

- 1st MapReduce Job
  - Heavy map phase
    - No GC issues any more
    - Tweak input split size
  - Lightweight shuffle
  - Lightweight reduce
    - Use 1-2 reduce tasks
- 2nd MapReduce Job
  - Very small
    - Enable uberization [HIVE-5857]

Current Best Time

2 months of data

50 min 2 sec

10th place

?

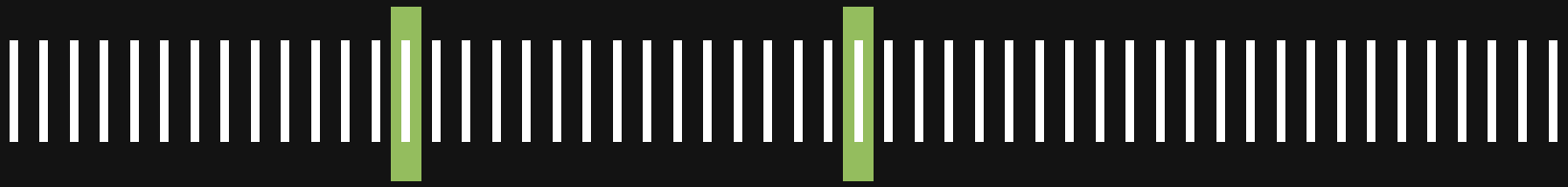
Changes are needed!

File Format

ORC

Observation

- The biggest table has 52 “real” columns
  - Only 2 of them are needed
  - 2 columns are “partition” columns



My Benefits Of ORC

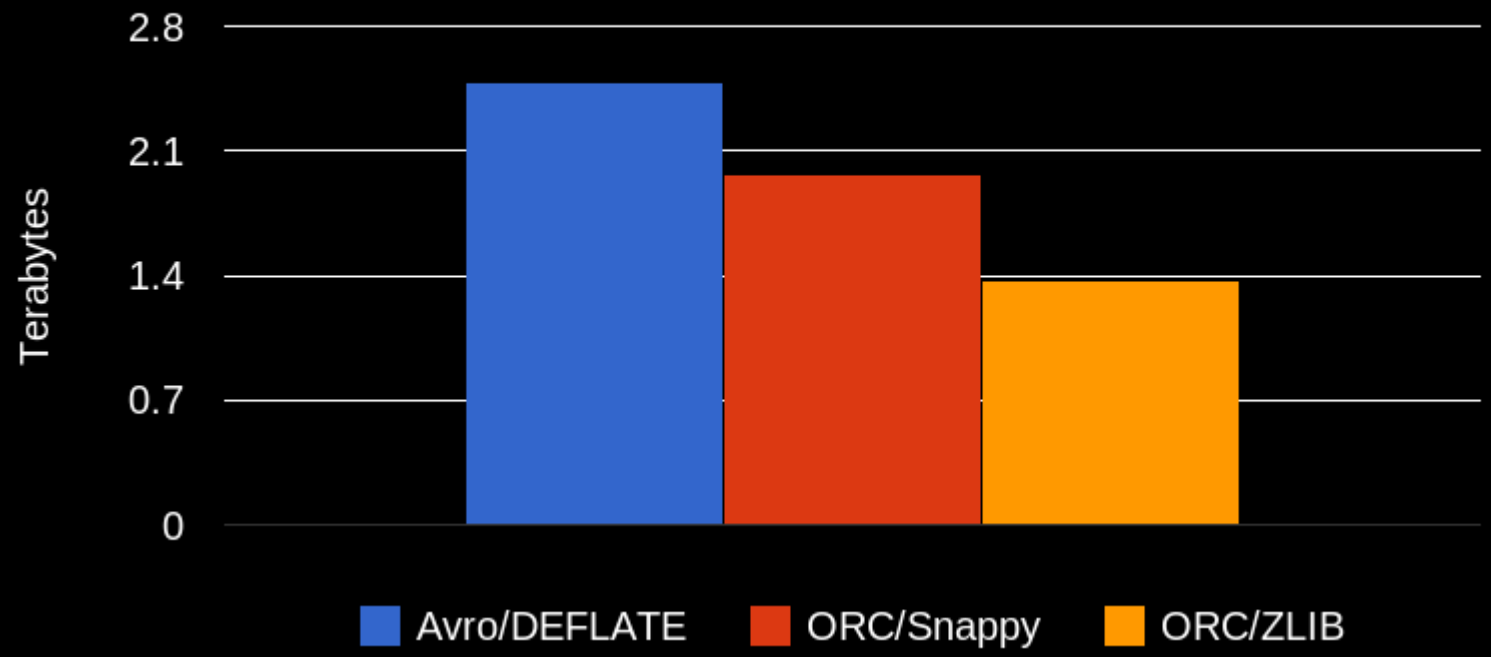
✅ Improve speed of query
- Only read required columns
  - Especially important when reading remotely

ORC Storage Consumption

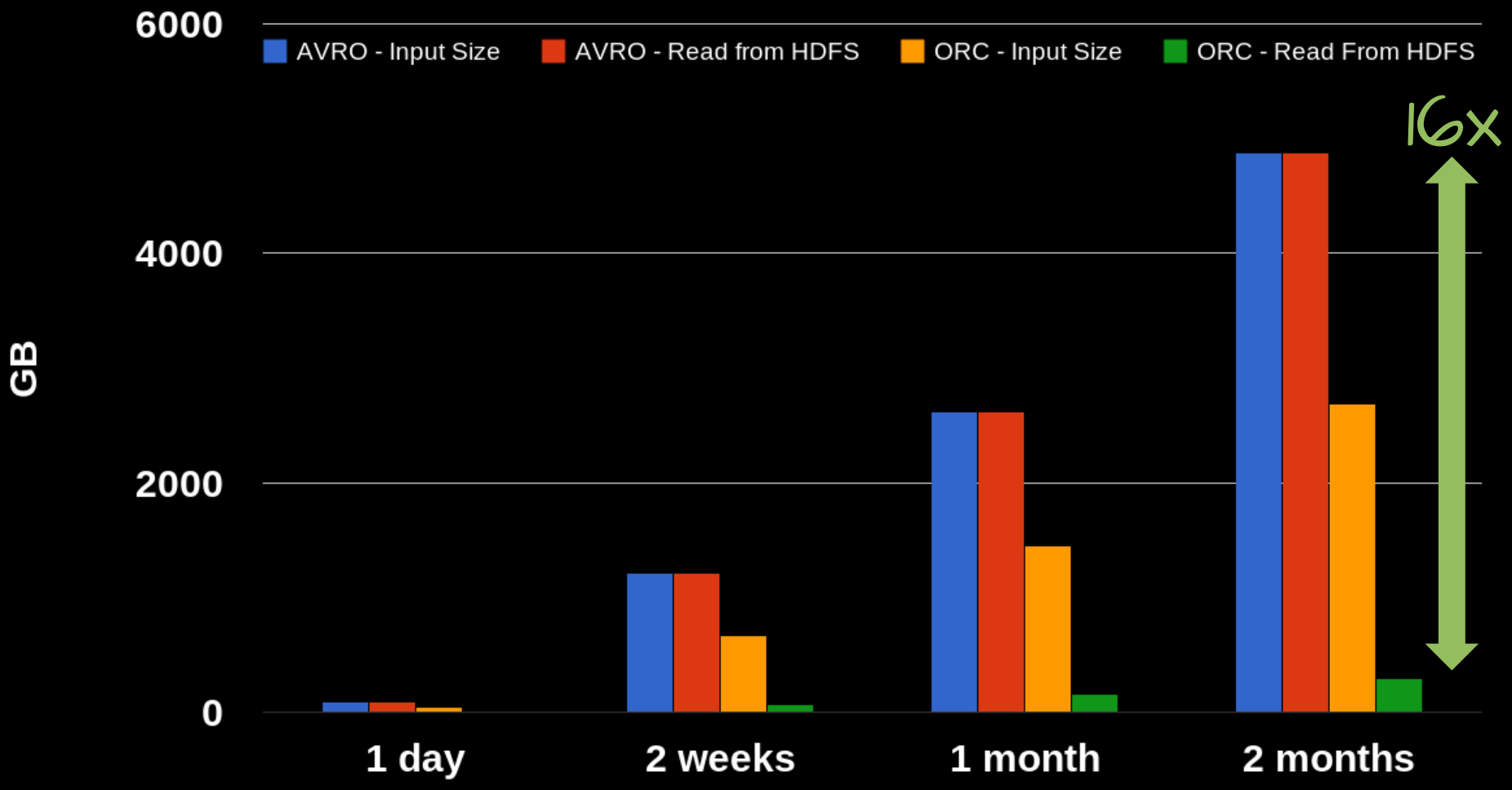
✗ Need to convert Avro to ORC

✓ ORC consumes less space

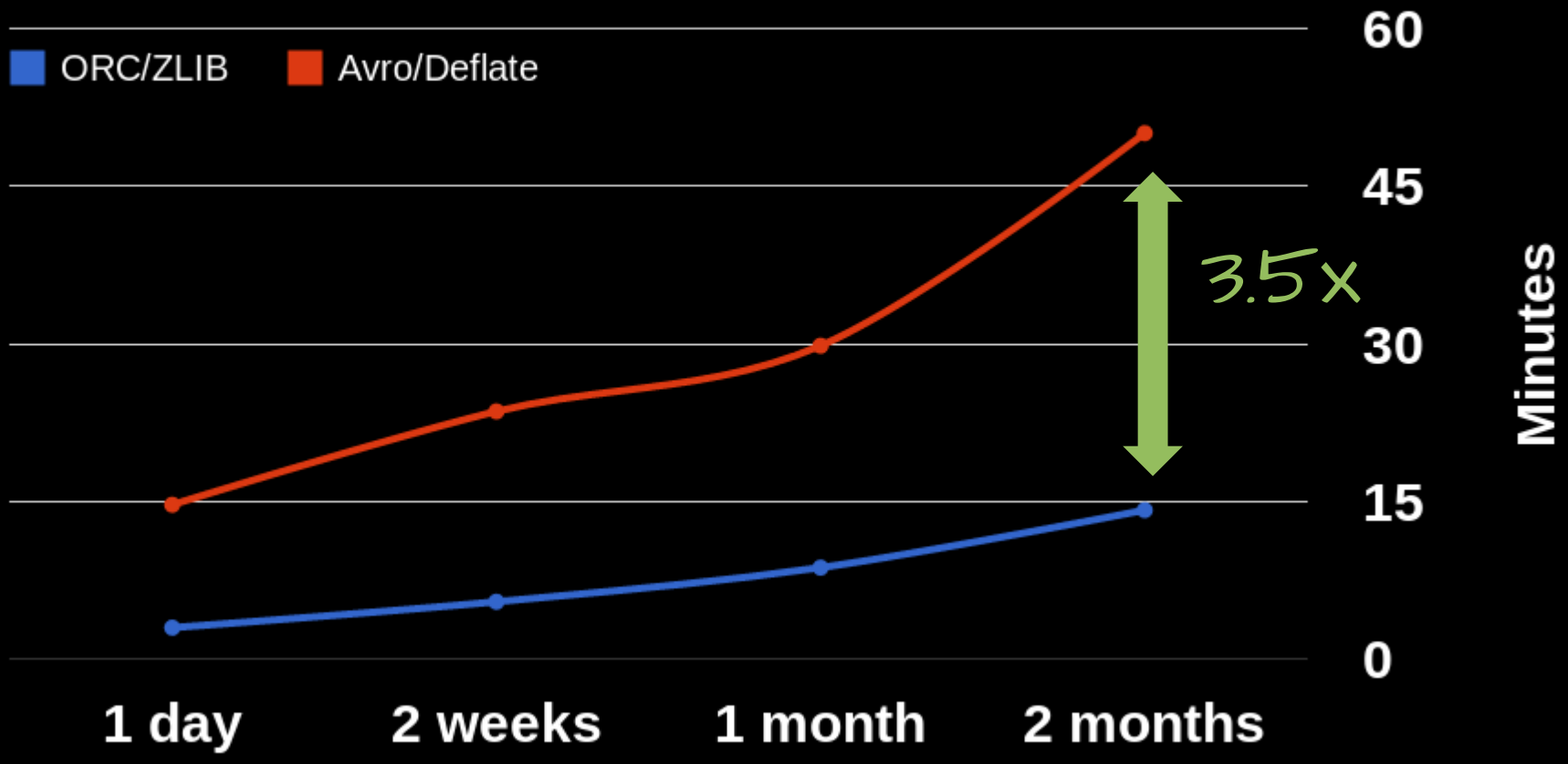
- Efficient encoding and compression



Storage And Access



Wall Clock Time



CPU Time


ORC/ZLIB
Avro/Deflate
1200
900
600
300
0
CPU Hours
32x
1 day
2 weeks
1 month
2 months

Possible Improvements

- Use Snappy compression
  - ZLIB was used to optimize more for storage than processing time

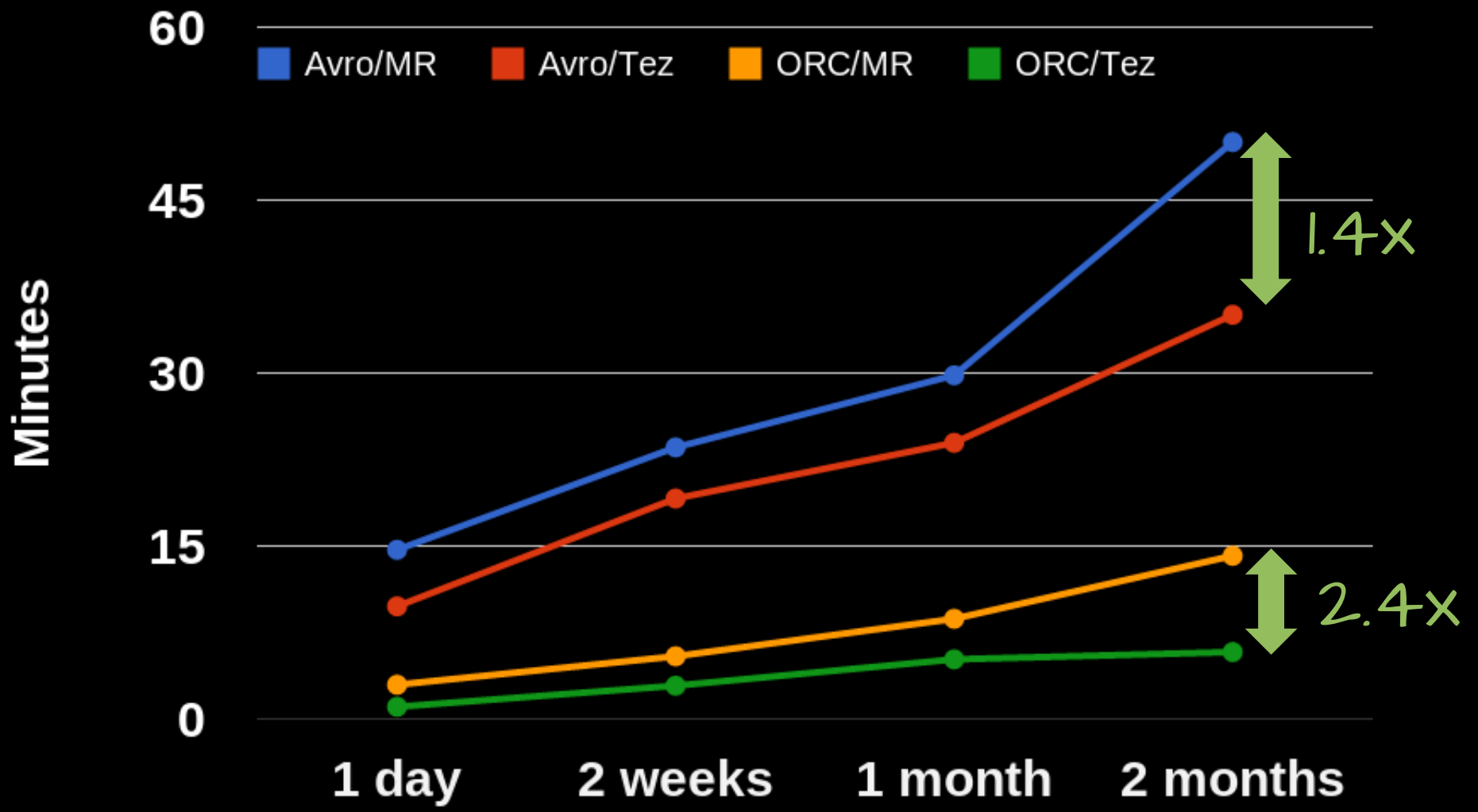
Computation

Apache Tez

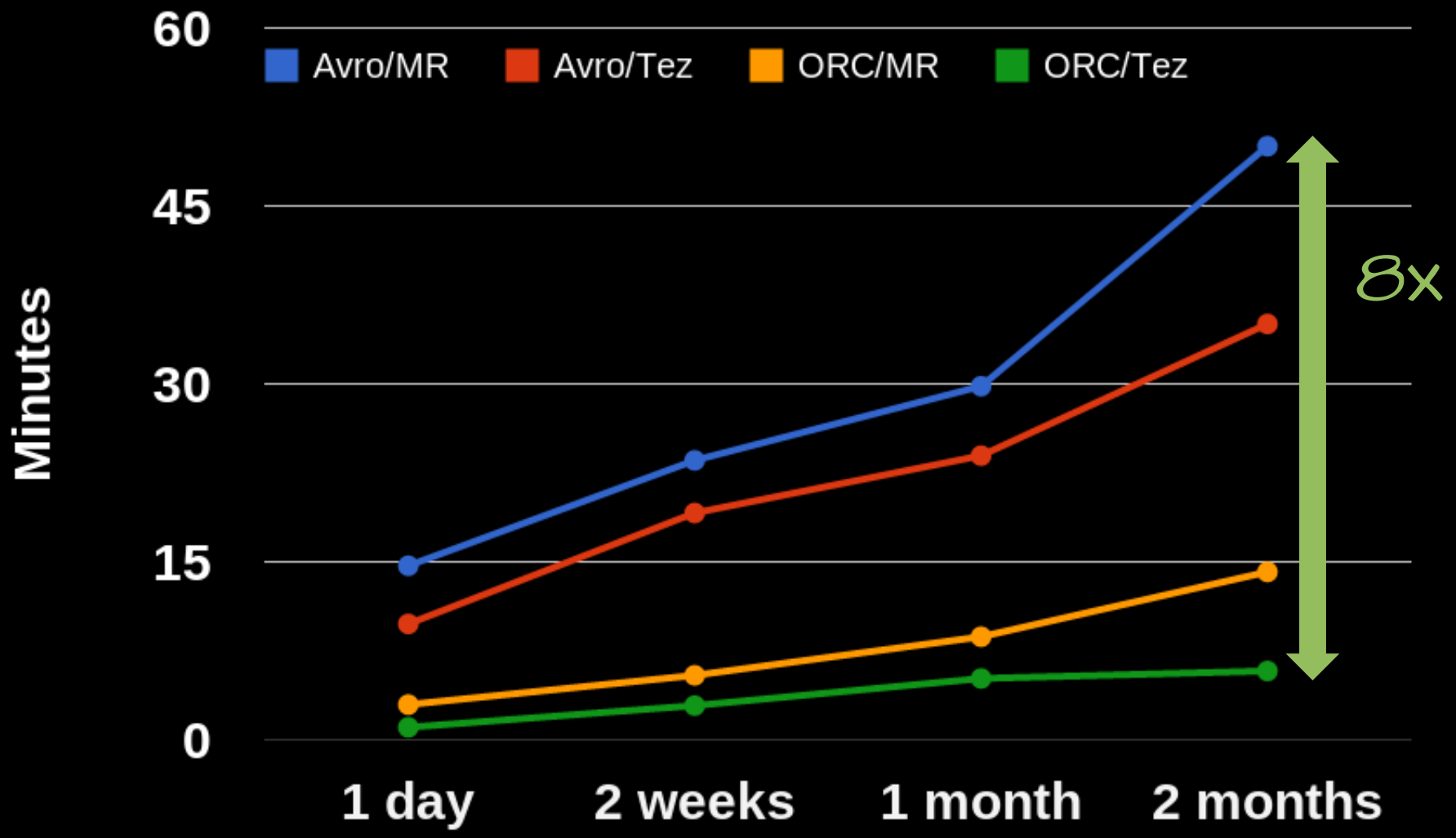
Thoughts?

- Tez will probably not help much, because…
  - Only two MapReduce jobs
  - Heavy work done in map tasks
  - Little intermediate data written to HDFS

Wall Clock Time



Wall Clock Time



Benefits Of Tez

✅ Natural DAG

- No “empty” map task
- No necessity to write intermediate data to HDFS
- No job scheduling overhead

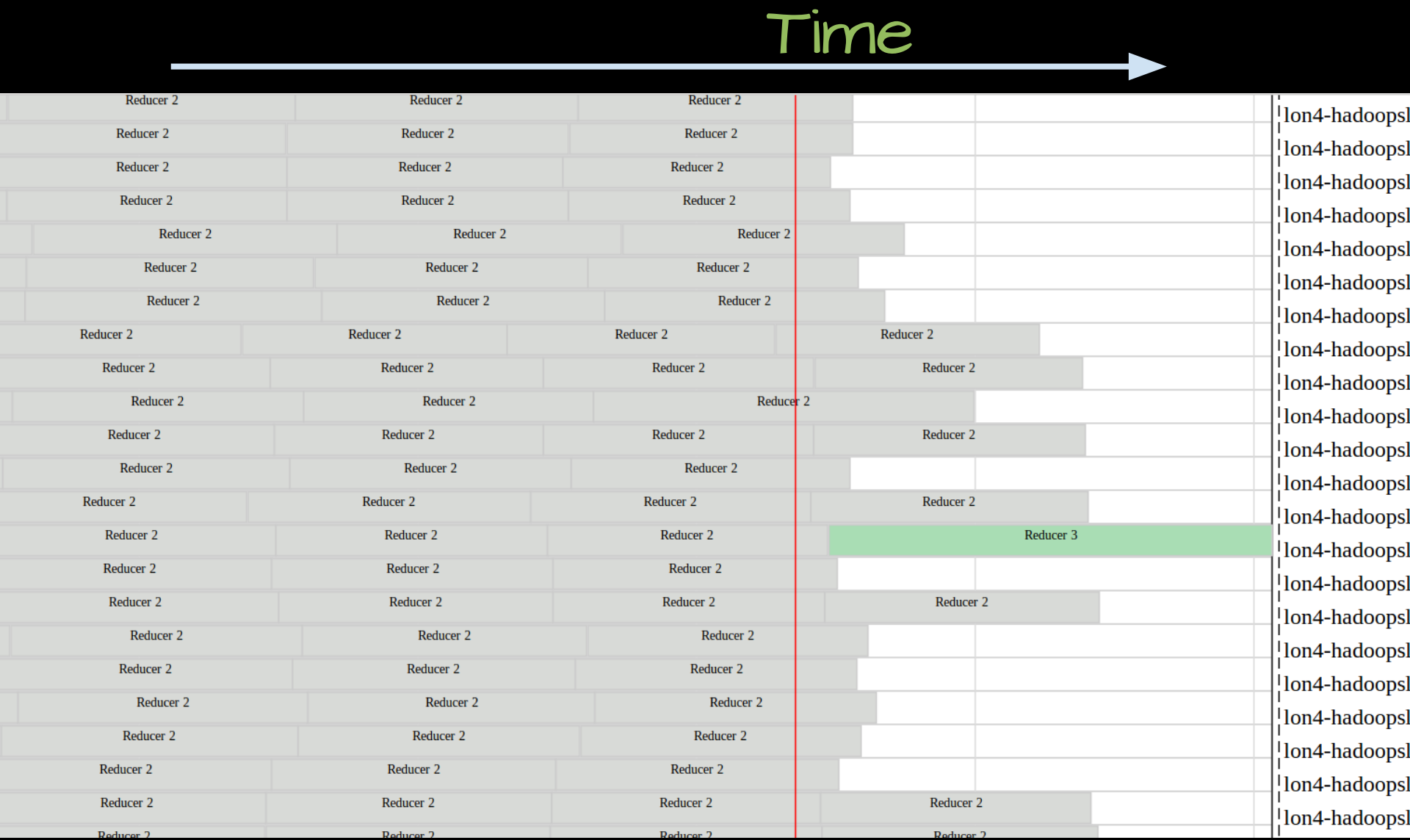
Benefits Of Tez

✅ Dynamic graph reconfiguration
- Decrease reduce task parallelism at runtime
- Pre-launch reduce tasks at runtime

Benefits Of Tez

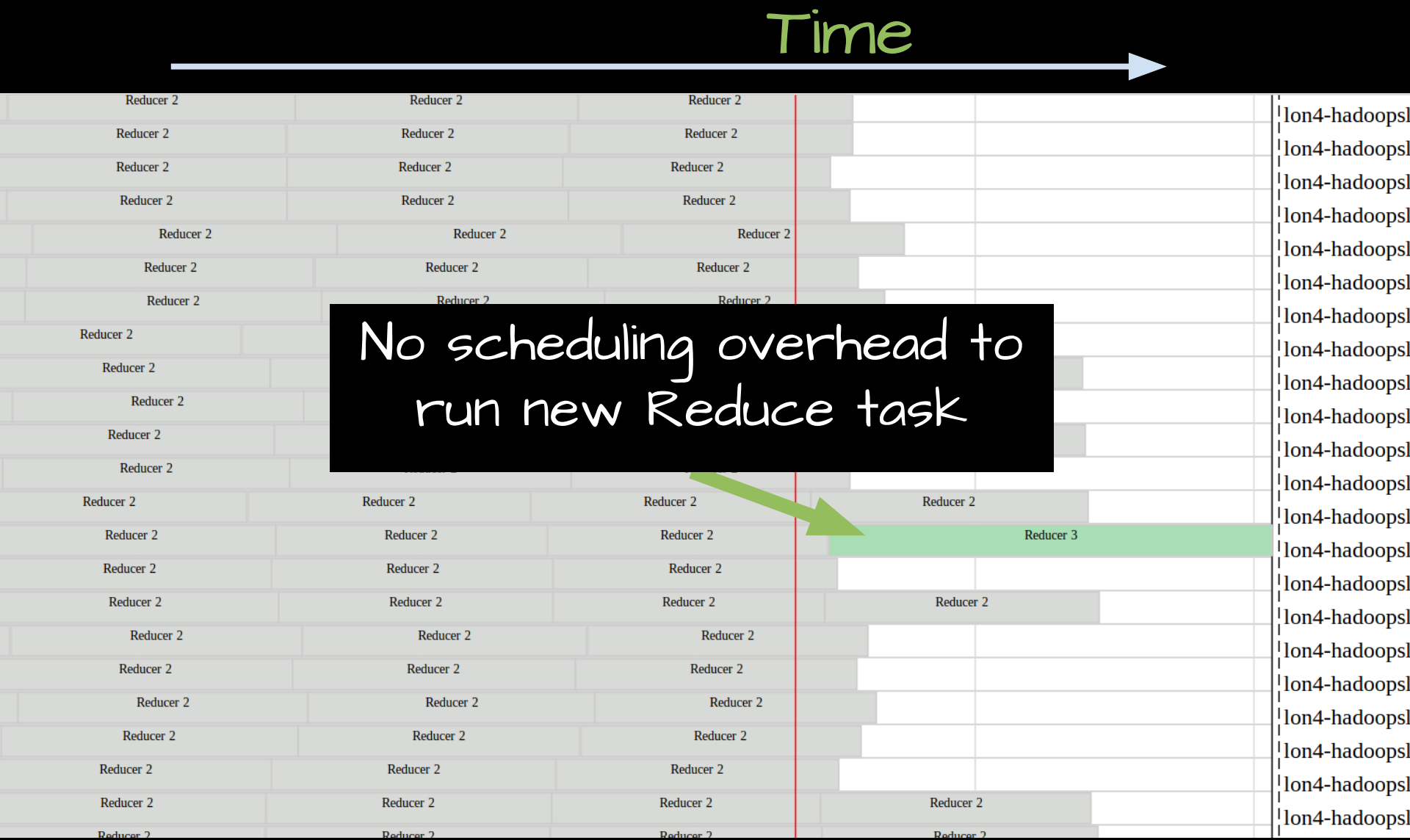
✅ Smart number of map tasks based on e.g.
- Min/max limits input split size
- Capacity utilization

Container Reuse



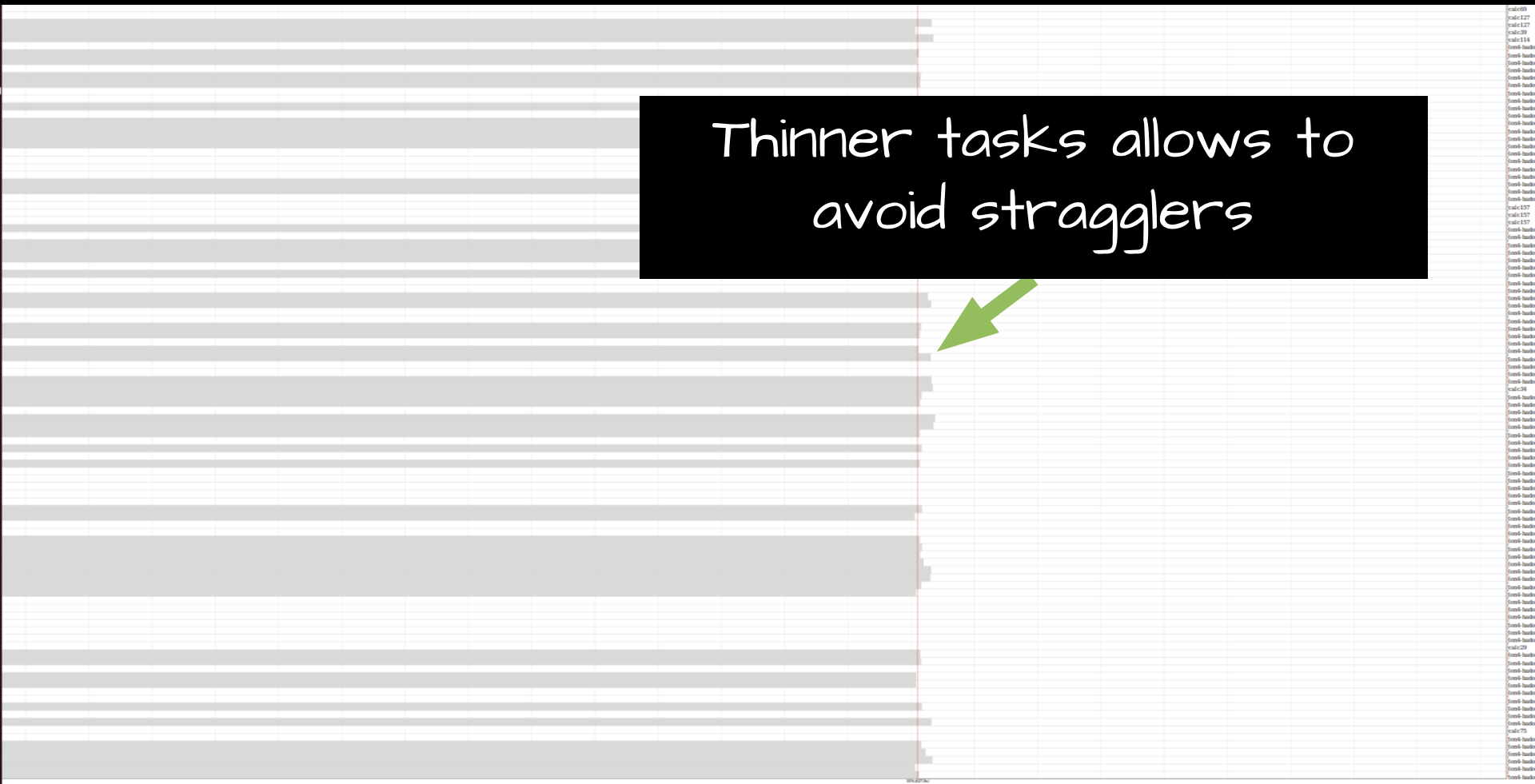
Container Reuse

Container Reuse



Container Reuse

Time



Container + Objects Reuse

Finished within 1,5 sec.

Warm !

EXPLAIN

- Much easier to follow!

```
Map 1 <- Map 4 (BROADCAST_EDGE), Map 5 (BROADCAST_EDGE)
Reducer 2 <- Map 1 (SIMPLE_EDGE)
Reducer 3 <- Reducer 2 (SIMPLE_EDGE)
```

- A dot file for graphviz is also generated

Broadcast Join

- Improved version of Map Join
- Hash table is built in the cluster
  - Parallelism
  - Data locality
  - No local and HDFS write barriers
- Hash table is broadcasted to downstream tasks
  - Tez container can cache them and reuse

Broadcast Join

- Try run the query that was interrupted by ^C
  - On 1 day of data to introduce overhead

Metric	Avro MR	Tez ORC
Wall-clock	UNKNOWN (^C after 30 minutes)	11m 16s

Memory-Related Benefits

✓ Automatically rescales the size of in-memory buffer for intermediate key-value pairs

✓ Reduced in-memory size of map-join hash tables

Lessons Learned

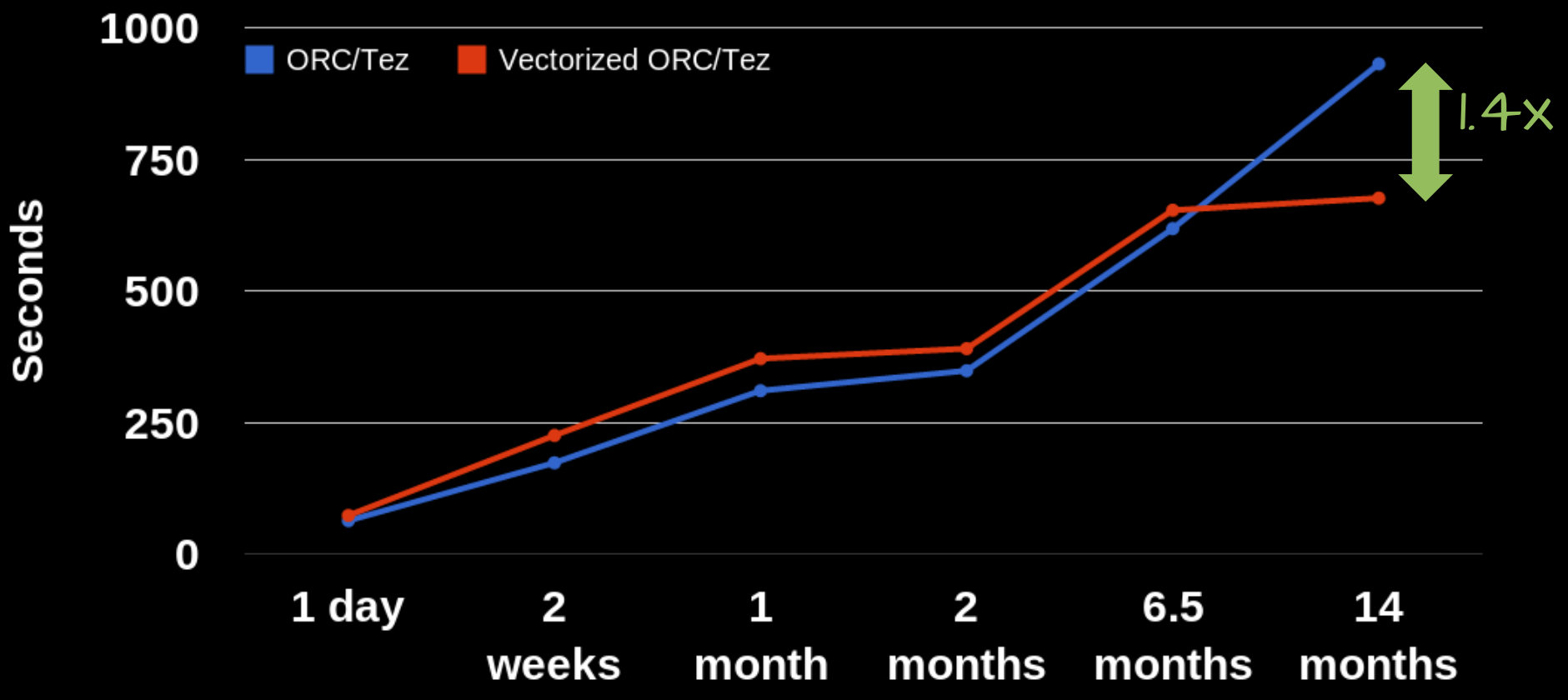
- At scale, Tez Application Master can be busy
  - `DEBUG` log level slows everything down
  - Sometimes needs more RAM
- A couple of tools are missing
  - e.g. counters, Web UI
  - `CLI`, log parser, `jstat` are your friends

Feature Vectorization

Vectorization

- ✓ Process data in a batch of 1024 rows together
  - Instead of processing one row at a time
- ✓ Reduces the CPU usage
- ✓ Scales very well especially with large datasets
  - Kicks in after a number of function calls
- ✓ Supported by ORC

Powered By Vectorization



Vectorized Map Join

✗ Not yet fully supported by vectorization

- ✓ Still less GC
- ✓ Still better use of memory to CPU bandwidth
- ✗ Cost of switching from vectorized to row-mode

✓ It will be revised soon by the community

Lessons Learned

- Ensure that vectorization is fully or partially supported for your query
  - Datatypes
  - Operators
- Run own benchmarks and profilers

Feature

Table Statistics

Table Statistics

✓ Table, partition and column statistics
✓ Possibility to
  - collect when loading data
  - generate for existing tables
  - use to produce optimal query plan
✓ Supported by ORC

Powered By Statistics

Period	ORC/Tez	Vectorized ORC/Tez	Vectorized ORC/Tez + Stats
2 weeks	173 sec	225 sec	129 sec
	1.22K maps	*1.22K maps*	*0.9K maps*
6.5 months	618 sec	653 sec	468 sec
	13.3K maps	*13.3K maps*	*9.4K maps*
14 months	931 sec	676 sec	611 sec
	24.8K maps	*24.8K maps*	*17.9K maps*

Current Best Time

14 months of data

10 min 11 sec

?

Results

Challenge

Results!

- Who is the biggest fan of Timbuktu in Poland?

Results!

- Who is the biggest fan of Timbuktu in Poland?

That's all !

Summary

- Deals at 5 AM might be the best idea ever
- Hive can be a single solution for all SQL queries
  - Both interactive and batch
  - Regardless of input dataset size
- Tez is really MapReduce v2
  - Very flexible and smart
  - Improves what was inefficient with MapReduce
  - Easy to deploy

Special Thanks

- Gopal Vijayaraghavan
  - Technical answers to Tez-related questions
  - Installation scripts
    github.com/t3rmin4t0r/tez-autobuild

- My colleagues for technical review and feedback
  - Piotr Krewski, Rafal Wojdyla, Uldis Barbans, Magnus Runesson

Questions?


Thank you!