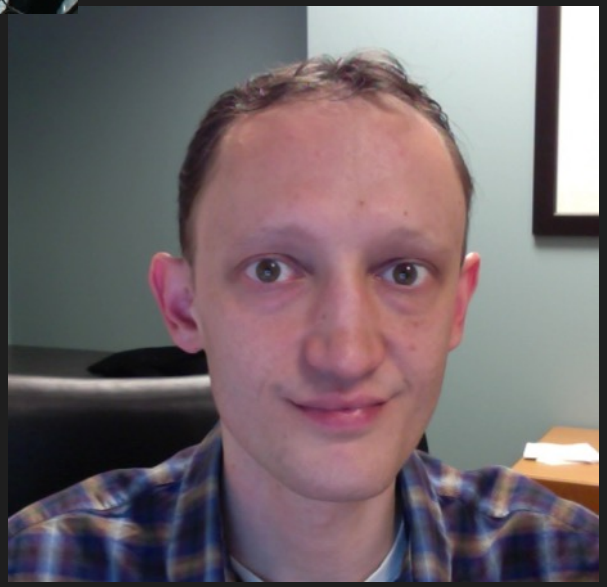
Hadoop, Hive, Spark and Object Stores

Steve Loughran
stevel@hortonworks.com
@steveloughran

November 2016

Steve Loughran,
Hadoop committer, PMC member,
ASF Member

Rajesh Balamohan
Tez Committer, PMC Member



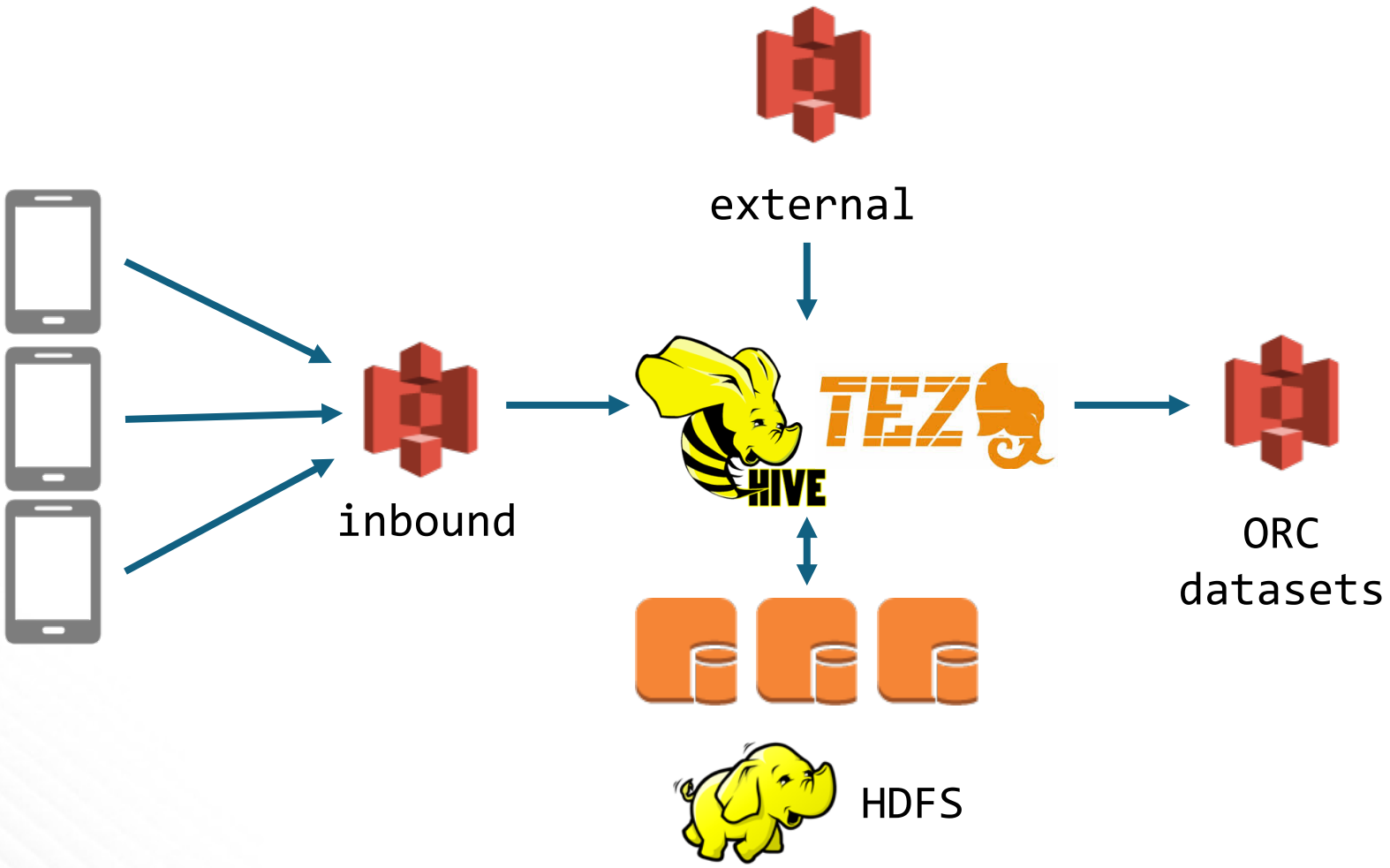
Chris Nauroth,
Apache Hadoop committer & PMC; ASF member

Make Apache Hadoop at home in the cloud

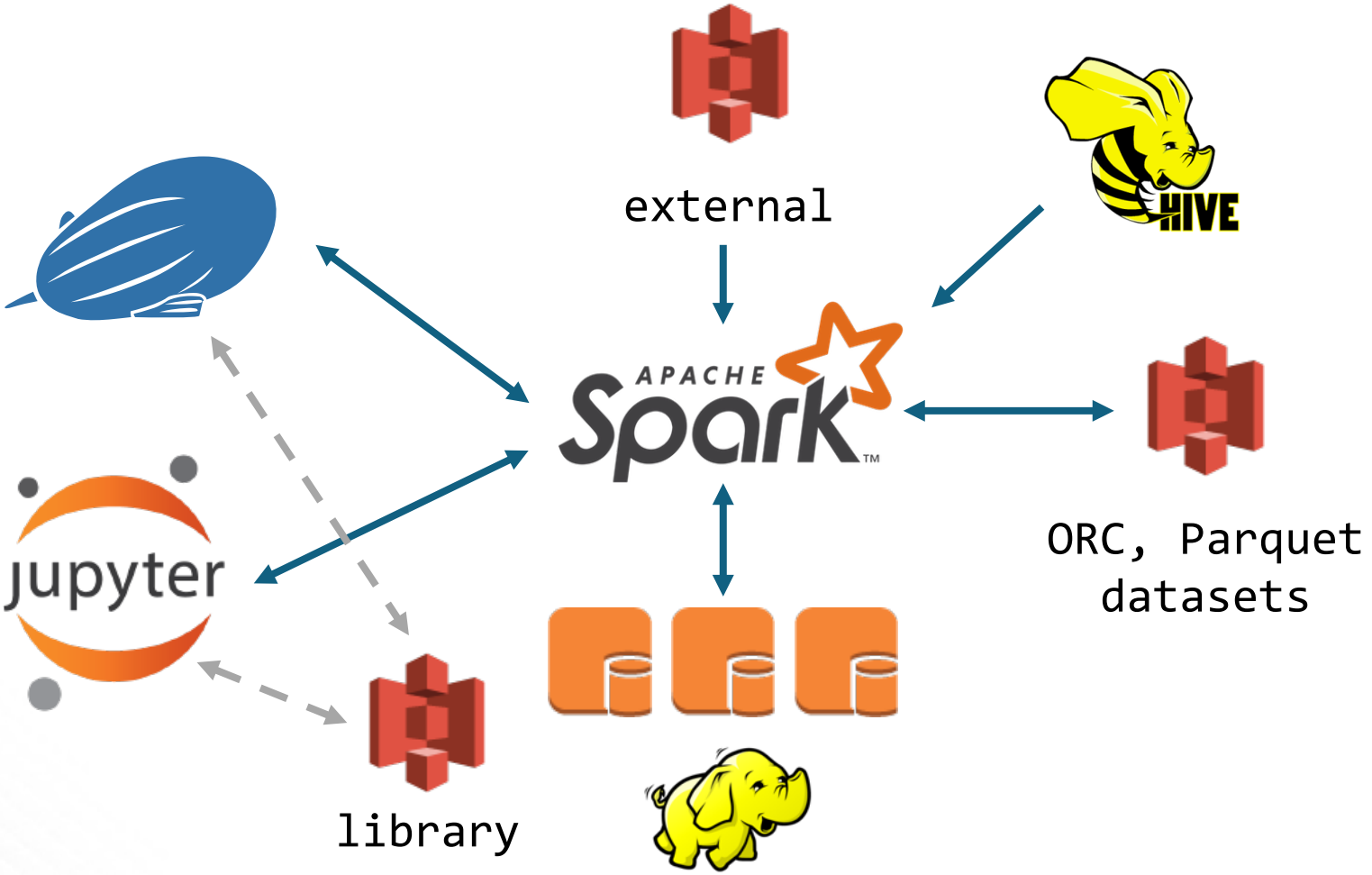
Step 1: Hadoop runs great on Azure
Step 2: Beat EMR on EC2

HORTONWORKS®
POWERING THE FUTURE OF DATA™

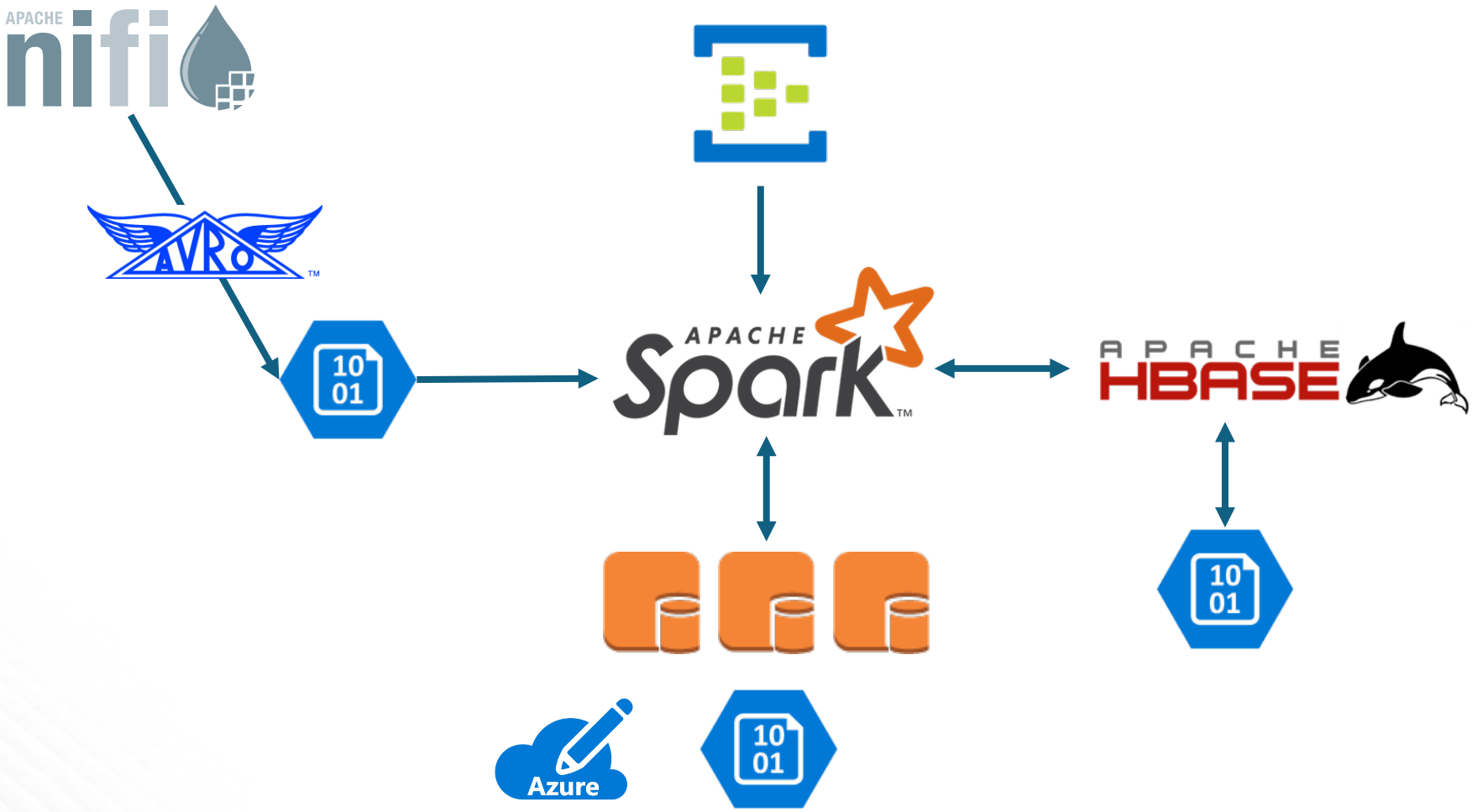
Elastic ETL



Notebooks

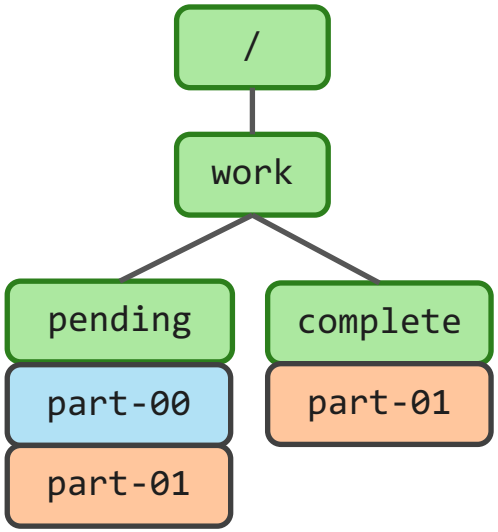


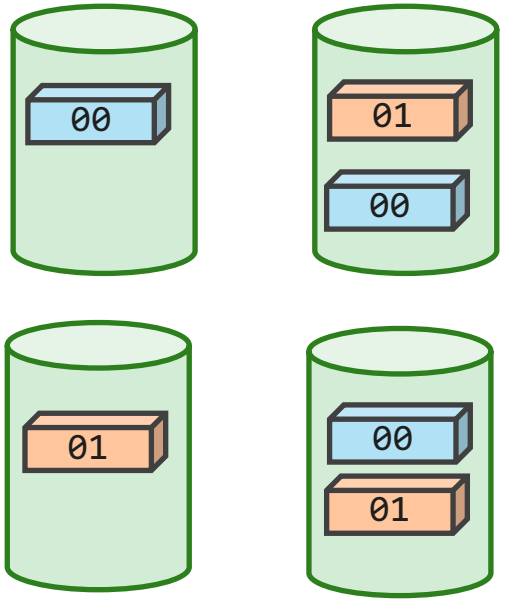
Streaming



HORTONWORKS

A Filesystem: Directories, Files → Data

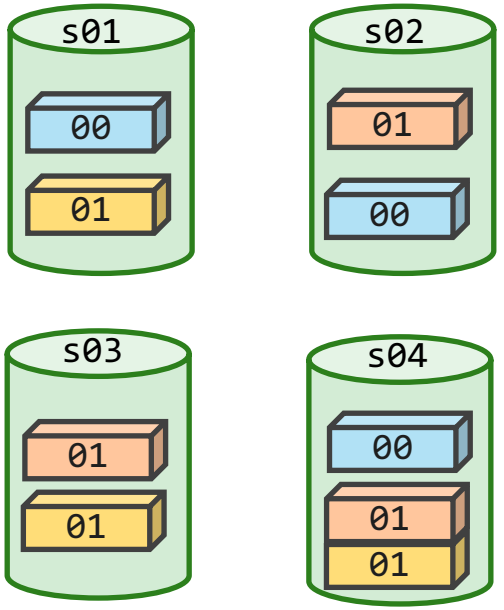





```
rename("/work/pending/part-01", "/work/complete")
```

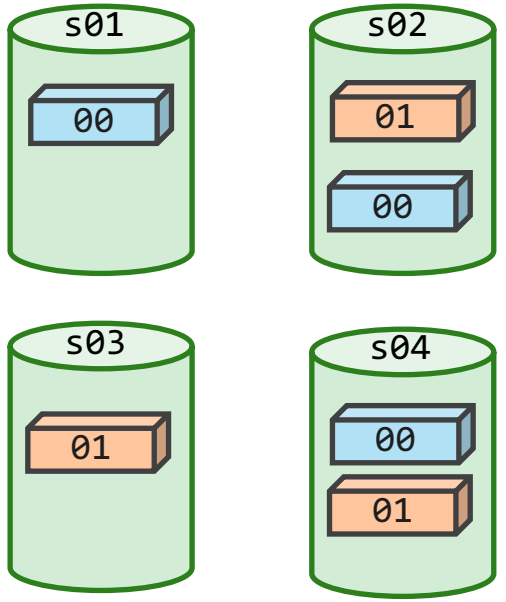
Object Store: hash(name)->blob

```
hash("/work/pending/part-00")
 ["s01", "s02", "s04"]

hash("/work/pending/part-01")
 ["s02", "s03", "s04"]

copy("/work/pending/part-01",
     "/work/complete/part01")


delete("/work/pending/part-01")
```

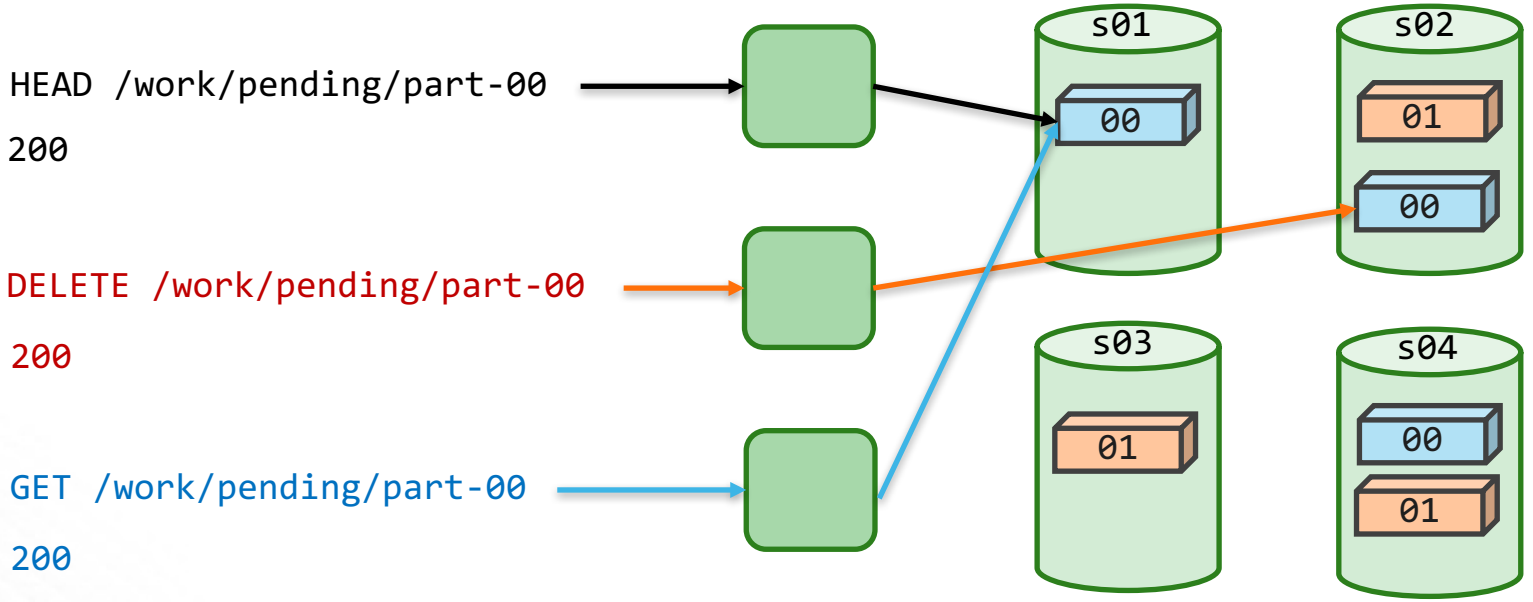


REST APIs

```
PUT /work/pending/part-01
... DATA ...

GET /work/pending/part-01
Content-Length: 1-8192

PUT /work/complete/part01
x-amz-copy-source: /work/pending/part-01

DELETE /work/pending/part-01

HEAD /work/complete/part-01

GET /?prefix=/work&delimiter=/
```

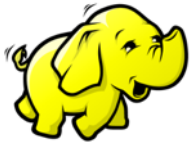


Often Eventually Consistent



Same API


```
org.apache.hadoop.fs.FileSystem
```



hdfs

s3a

wasb

swift

gs

adl

HORTONWORKS

Just a different URL to read

```
val csvdata = spark.read.options(Map(
    "header" -> "true",
    "inferSchema" -> "true",
    "mode" -> "FAILFAST"))
  .csv("s3a://landsat-pds/scene_list.gz")
```

HORTONWORKS

Writing looks the same …

```
val p = "s3a://hwdev-stevel-demo/landsat"
csvData.write.parquet(p)

val o = "s3a://hwdev-stevel-demo/landsatOrc"
csvData.write.orc(o)
```

Hive

```
CREATE EXTERNAL TABLE `scene`(
   `entityid` string,
   `acquisitiondate` timestamp,
   `cloudcover` double,
   `processinglevel` string,
   `path` int,
   `row_id` int,
   `min_lat` double,
   `min_long` double,
   `max_lat` double,
   `max_lon` double,
   `download_url` string) ROW FORMAT DELIMITED
 FIELDS TERMINATED BY ','
 LINES TERMINATED BY '\n'
 STORED AS TEXTFILE
 LOCATION s3a://hwdev-rajesh-new2/scene_list'
 TBLPROPERTIES ('skip.header.line.count'='1');
```

(needed to copy file to R/W object store first)

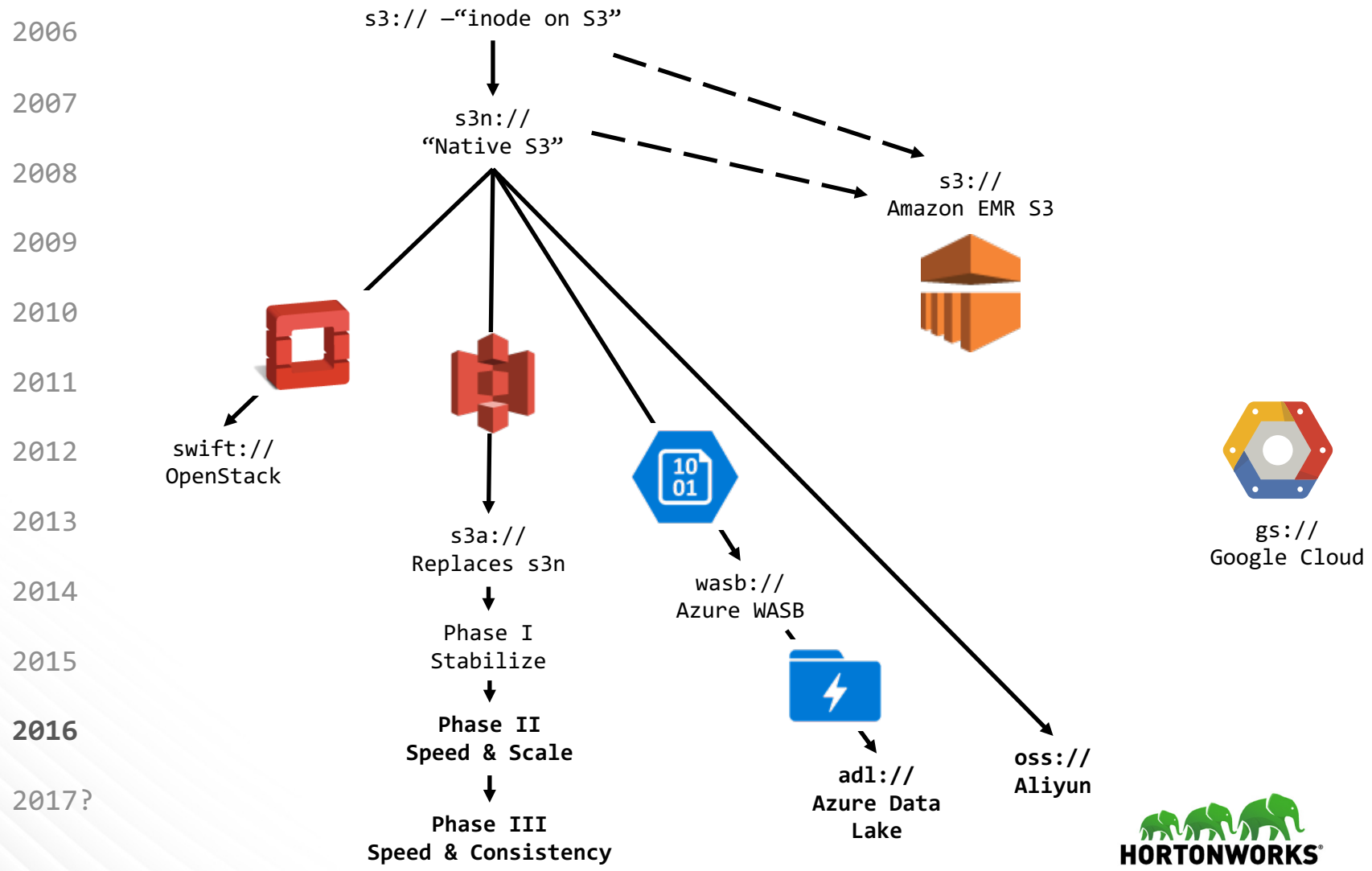
HORTONWORKS

```
> select entityID from scene where cloudCover < 0  limit 10;

+--------------------------+--+
|        entityid          |
+--------------------------+--+
| LT81402112015001LGN00    |
| LT81152012015002LGN00    |
| LT81152022015002LGN00    |
| LT81152032015002LGN00    |
| LT81152042015002LGN00    |
| LT81152052015002LGN00    |
| LT81152062015002LGN00    |
| LT81152072015002LGN00    |
| LT81162012015009LGN00    |
| LT81162052015009LGN00    |
+--------------------------+--+
```

Spark Streaming on Azure Storage

```
val streamc = new StreamingContext(sparkConf, Seconds(10))
val azure = "wasb://demo@example.blob.core.windows.net/in"
val lines = streamc.textFileStream(azure)
val matches = lines.map(line => {
    println(line)
    line
  })
matches.print()
streamc.start()
```

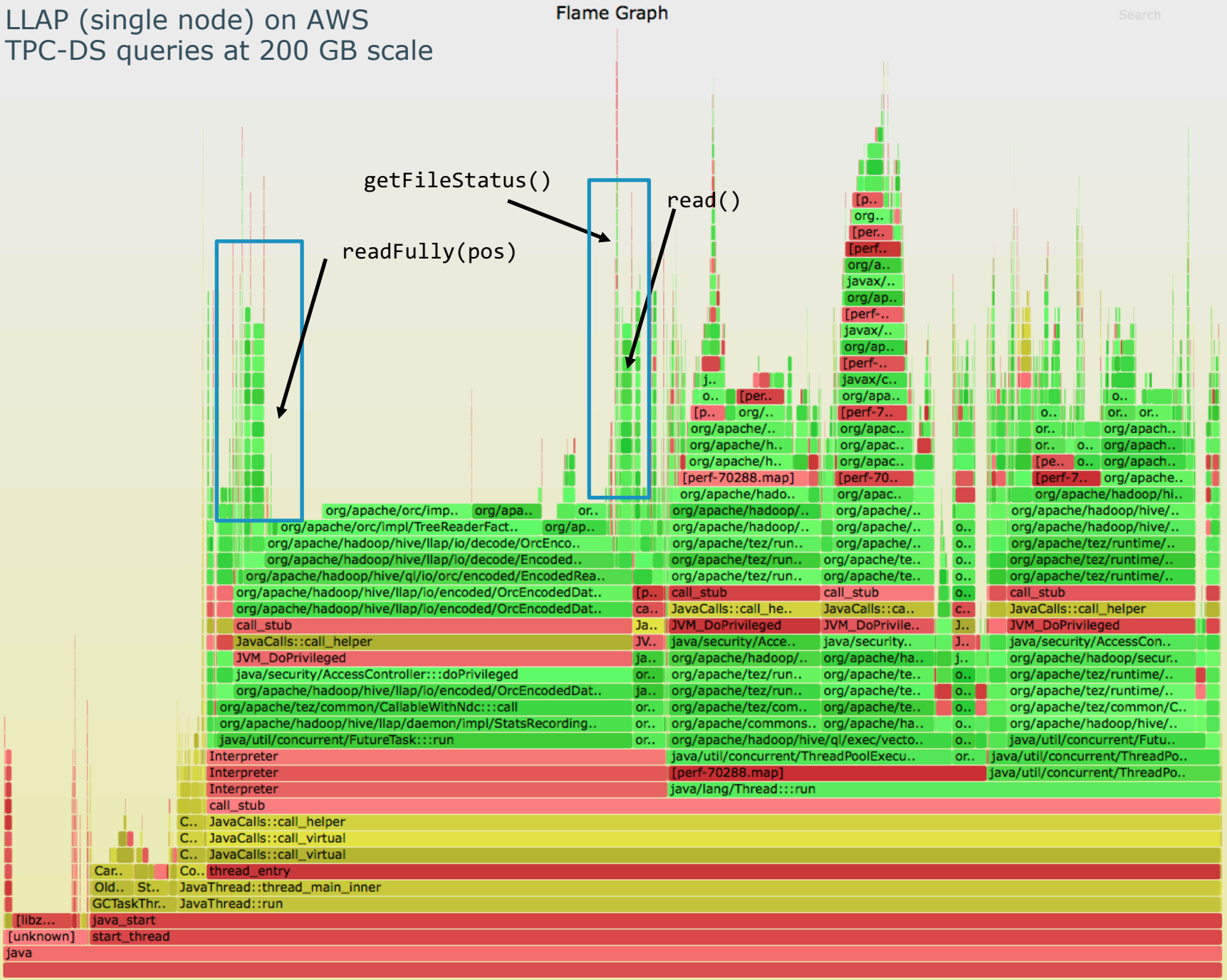
Where did those object store clients come from?



Problem: S3 work is too slow

1. Analyze benchmarks and bug-reports
2. Fix Read path
3. Fix Write path
4. Improve query partitioning
5. The Commitment Problem

LLAP (single node) on AWS
TPC-DS queries at 200 GB scale



The Performance Killers

```
getFileStatus(Path) (+ isDirectory(), exists())

  HEAD path          // file?
  HEAD path + "/"    // empty directory?
  LIST path          // path with children?


read(long pos, byte[] b, int idx, int len)


readFully(long pos, byte[] b, int idx, int len)
```

Positioned reads: close + GET, close + GET

```
read(long pos, byte[] b, int idx, int len)
    throws IOException {
  long oldPos = getPos();
  int nread = -1;
  try {
    seek(pos);
    nread = read(b, idx, len);
  } catch (EOFException e) {
  } finally {
    seek(oldPos);
  }
  return nread;
}
```

seek() is the killer, especially the seek() back

HADOOP-12444 Support lazy seek in S3AInputStream

```
public synchronized void seek(long pos)
    throws IOException {
  nextReadPos = targetPos;
}
```

+configurable readhead before open/close()

```
<property>
  <name>fs.s3a.readahead.range</name>
  <value>256K</value>
</property>
```

But: ORC reads were still underperforming

HORTONWORKS

HADOOP-13203: fs.s3a.experimental.input.fadvise

```
// Before
GetObjectRequest req = new GetObjectRequest(bucket, key)
  .withRange(pos, contentLength - 1);


// after
finish = calculateRequestLimit(inputPolicy, pos,
  length, contentLength, readahead);

GetObjectRequest req = new GetObjectRequest(bucket, key)
  .withRange(pos, finish);
```

bad for full file reads

Every HTTP request is precious

- HADOOP-13162: Reduce number of getFileStatus calls in mkdirs()
- HADOOP-13164: Optimize deleteUnnecessaryFakeDirectories()
- HADOOP-13406: Consider reusing filestatus in delete() and mkdirs()
- HADOOP-13145: DistCp to skip getFileStatus when not preserving metadata
- HADOOP-13208: listFiles(recursive=true) to do a bulk listObjects

see HADOOP-11694

Performance Considerations When Running Hive Queries

- Splits Generation
  - File formats like ORC provides threadpool in split generation
- ORC Footer Cache
  - `hive.orc.cache.stripe.details.size > 0`
- Reduce S3A reads in Task side
  - `hive.orc.splits.include.file.footer=true`

HORTONWORKS

Performance Considerations When Running Hive Queries

- Tez Splits Grouping
  - Hive uses Tez as its default execution engine
  - Tez groups splits based on min/max group setting, location details and so on
  - S3A always provides “localhost” as its block location information
  - When all splits-length falls below min group setting, Tez aggressively groups them into single split. This causes issues with S3A as single task ends up doing sequential operations.
  - Fixed in recent releases

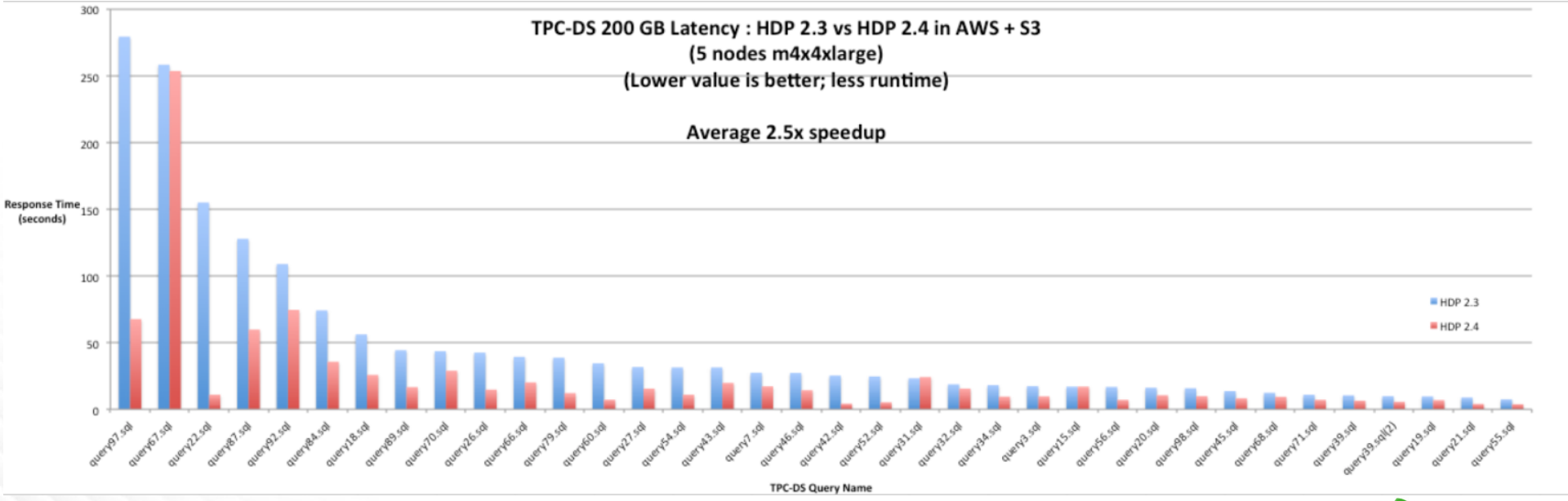
- Container Launches
  - S3A always provides “localhost” for block locations.
  - Good to set “yarn.scheduler.capacity.node-locality-delay=0”

HORTONWORKS

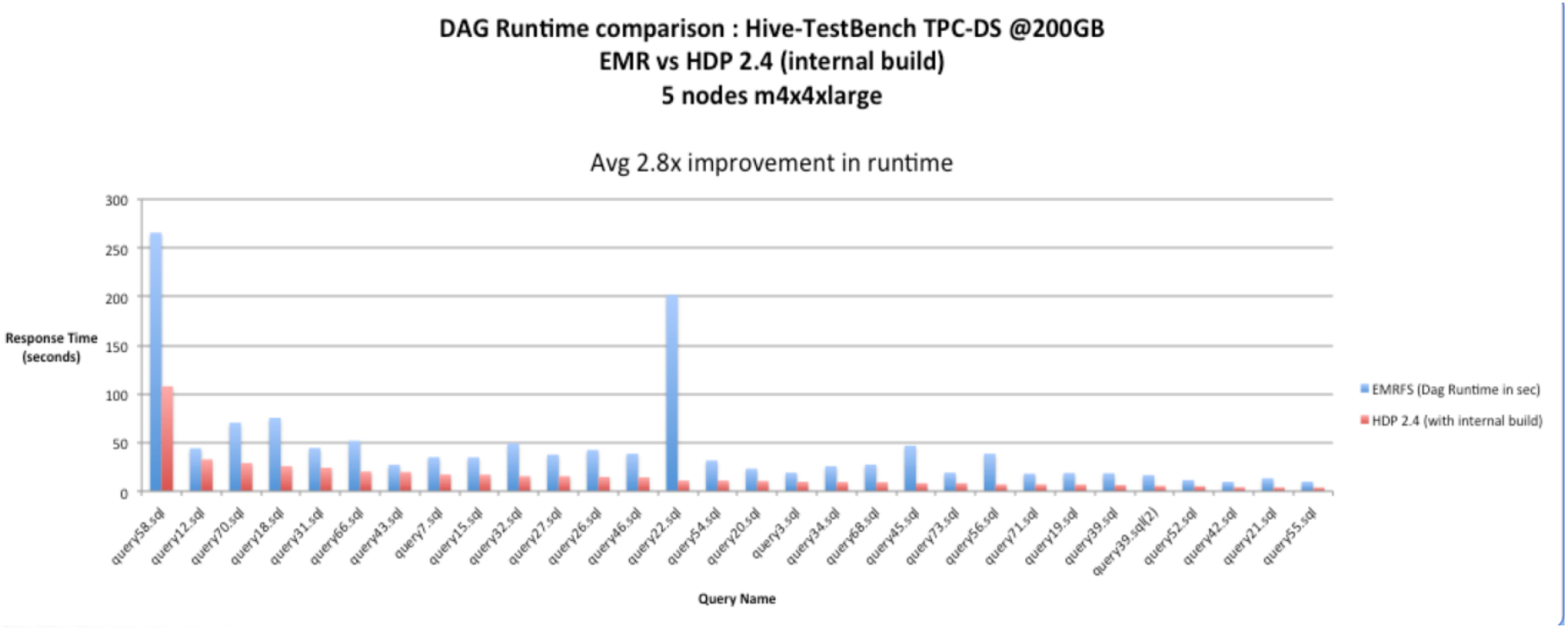

benchmarks !=
your queries
your data
…but we think we've made a good start
HORTONWORKS
POWERING THE FUTURE OF DATA

Hive-TestBench Benchmark shows average 2.5x speedup

- TPC-DS @ 200 GB Scale in S3 (https://github.com/hortonworks/hive-testbench)
- m4x4x large - 5 nodes
- “HDP 2.3 + S3 in cloud” vs “HDP 2.4 + enhancements + S3 in cloud
- Queries like 15,17, 25, 73,75 etc did not run in HDP 2.3 (AWS timeouts)



And EMR? average 2.8x, in our TCP-DS benchmarks



*Queries 40, 50,60,67,72,75,76,79 etc do not complete in EMR.

What about Spark?

object store work applies
needs tuning
SPARK-7481 patch handles JARs

Spark 1.6/2.0 Classpath running with Hadoop 2.7

hadoop-aws-2.7.x.jar
hadoop-azure-2.7.x.jar

aws-java-sdk-1.7.4.jar
joda-time-2.9.3.jar
azure-storage-2.2.0.jar

spark-default.conf

```
spark.sql.parquet.filterPushdown true
spark.sql.parquet.mergeSchema false
spark.hadoop.parquet.enable.summary-metadata false

spark.sql.orc.filterPushdown true
spark.sql.orc.splits.include.file.footer true
spark.sql.orc.cache.stripe.details.size 10000

spark.sql.hive.metastorePartitionPruning true

spark.hadoop.fs.s3a.readahead.range 157810688
spark.hadoop.fs.s3a.experimental.input.fadvise random
```

HORTONWORKS

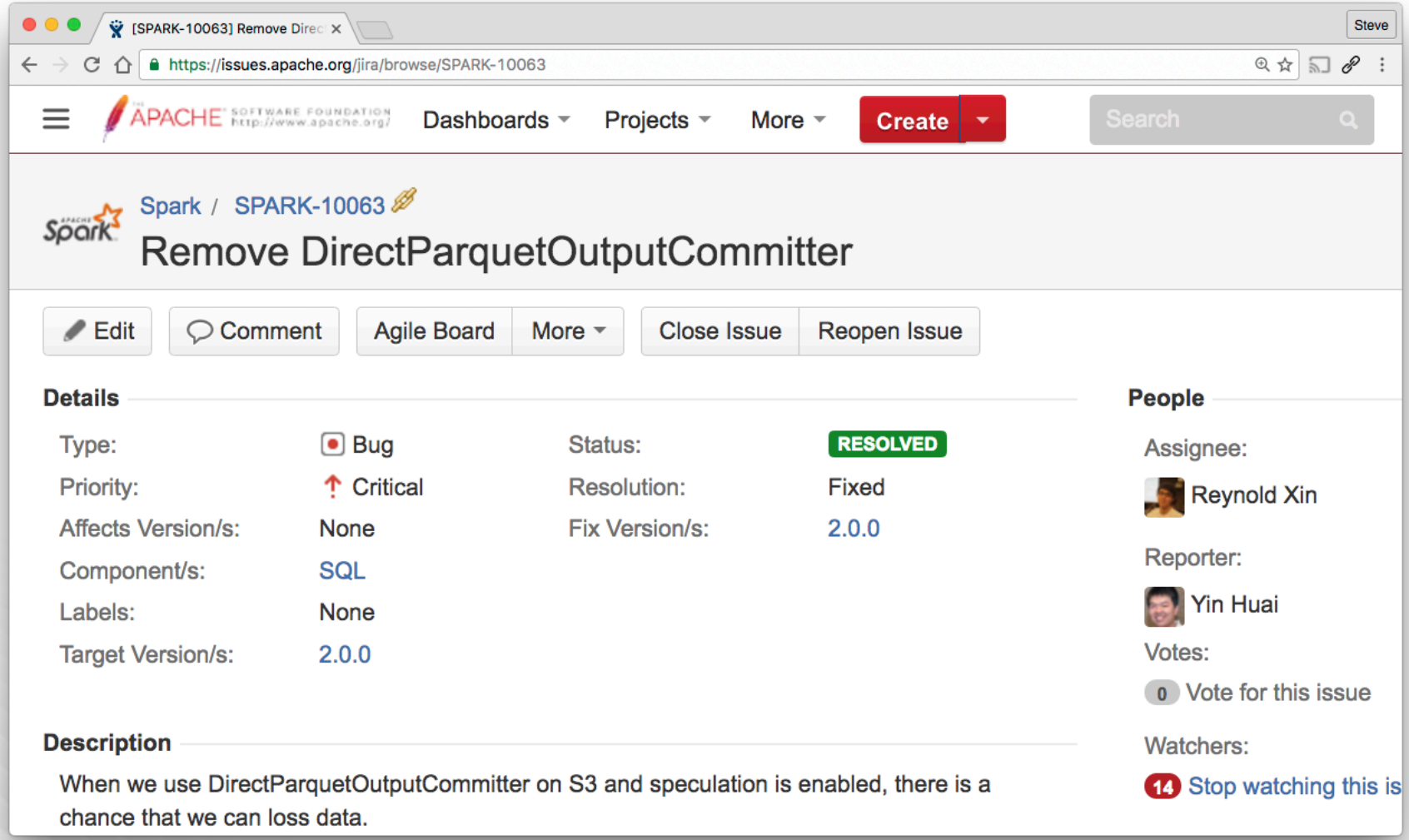
The Commitment Problem

- `rename()` used for atomic commitment transaction
- Time to copy() + delete() proportional to data * files
- S3: 6+ MB/s
- Azure: a lot faster —*usually*

```
spark.speculation false
spark.hadoop.mapreduce.fileoutputcommitter.algorithm.version 2
spark.hadoop.mapreduce.fileoutputcommitter.cleanup.skipped true
```

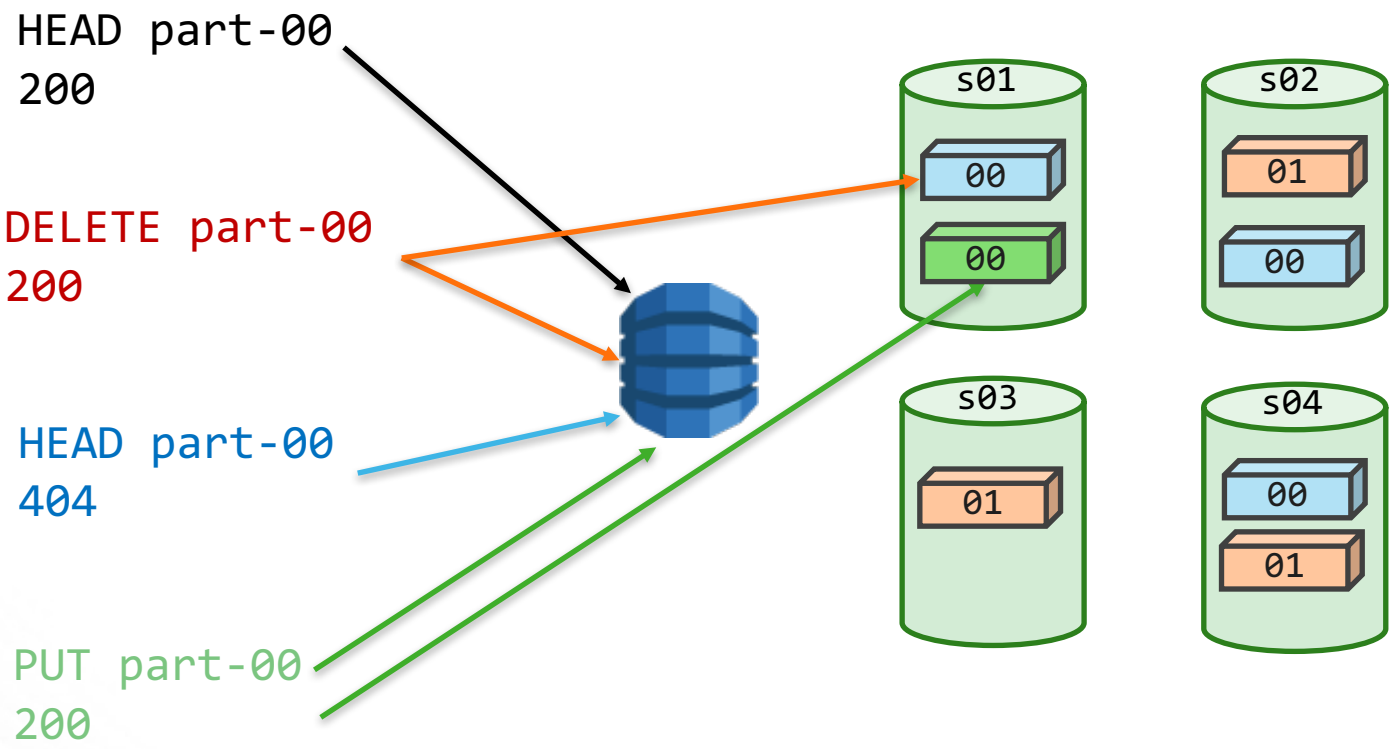
HORTONWORKS

What about Direct Output Committers?



s3guard:
fast, consistent S3 metadata

DynamoDB becomes the consistent metadata store



How do I get hold of these features?

- Read improvements in HDP 2.5
- Read + Write in Hortonwork Data Cloud
- Read + Write in Apache Hadoop 2.8 (soon!)
- s3Guard: No timetable

You can make your own code work better here too!

😢 Reduce `getFileStatus()`, `exists()`, `isDir()`, `isFile()` calls

😢 Avoid `globStatus()`

😢 Reduce `listStatus()` & `listFiles()` calls

😭 *Really* avoid `rename()`

😀 Prefer forward seek,

😀 Prefer `listStatus(path, recursive=true)`

😀 list/delete/rename in separate threads

😀 test against object stores

HORTONWORKS

Questions?

Backup Slides

Write Pipeline

- PUT blocks as part of a multipart, as soon as size is reached
- Parallel uploads during data creation
- Buffer to disk (default), heap or byte buffers
- Great for distcp

```
fs.s3a.fast.upload=true
fs.s3a.multipart.size=16M
fs.s3a.fast.upload.active.blocks=8

// tip:
fs.s3a.block.size=${fs.s3a.multipart.size}
```

Parallel rename (Work in Progress)

- Goal: faster commit by rename
- Parallel threads to perform the COPY operation
- `listFiles(path, true).sort().parallelize(copy)`
- Time from `sum(data)/copy-bandwidth` to more `size(largest-file)/copy-bandwidth`
- Thread pool size will limit parallelism
- Best speedup with a few large files rather than many small ones
- wasb expected to stay faster & has leases for atomic commits

HORTONWORKS