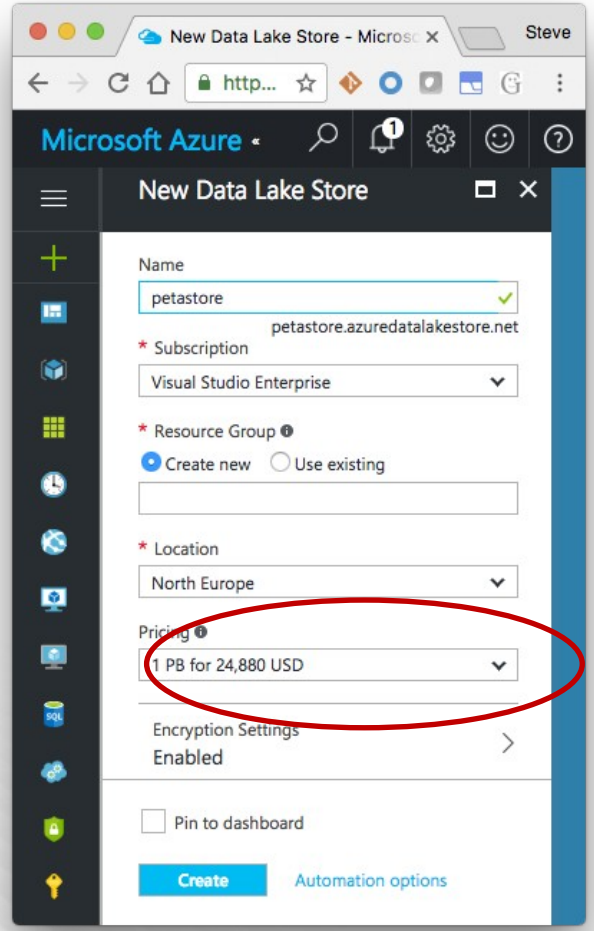
S3A and S3Guard

August 2017



Why Cloud?

- No upfront hardware costs – Pay as you use
- Elasticity
- Often lower TCO
- Natural for Data ingress for IoT, mobile apps, ..
- Business agility

HORTONWORKS

Shared Data and Cloud Storage

- Cloud Storage is the Shared Data Lake
  - –For both Hadoop and Cloud-native (non-Hadoop) Apps
  - –Lower Cost
    - •HDFS via EBS can get expensive
    - •HDFS's role changes
  - –Built-in geo-distribution and DR

- Challenges
  - –Cloud storage designed for scale, low cost and geo-distribution
  - –Performance is slower – was not designed for data-intensive apps
  - –Cloud storage segregated from compute
  - –API and semantics not like a FS – especially wrt. consistency

HORTONWORKS

Making Object Stores work for Big Data Apps

- Focus Areas
  - –Address cloud storage consistency
  - –Performance (changes in connectors and frameworks)
  - –Caching in memory and local storage

- Other issues not covered in this talk
  - –Shared Metastore, Common Governance, Security across multiple clusters
  - –Columnar access control to Tabular data

  See Hortonworks cloud offering

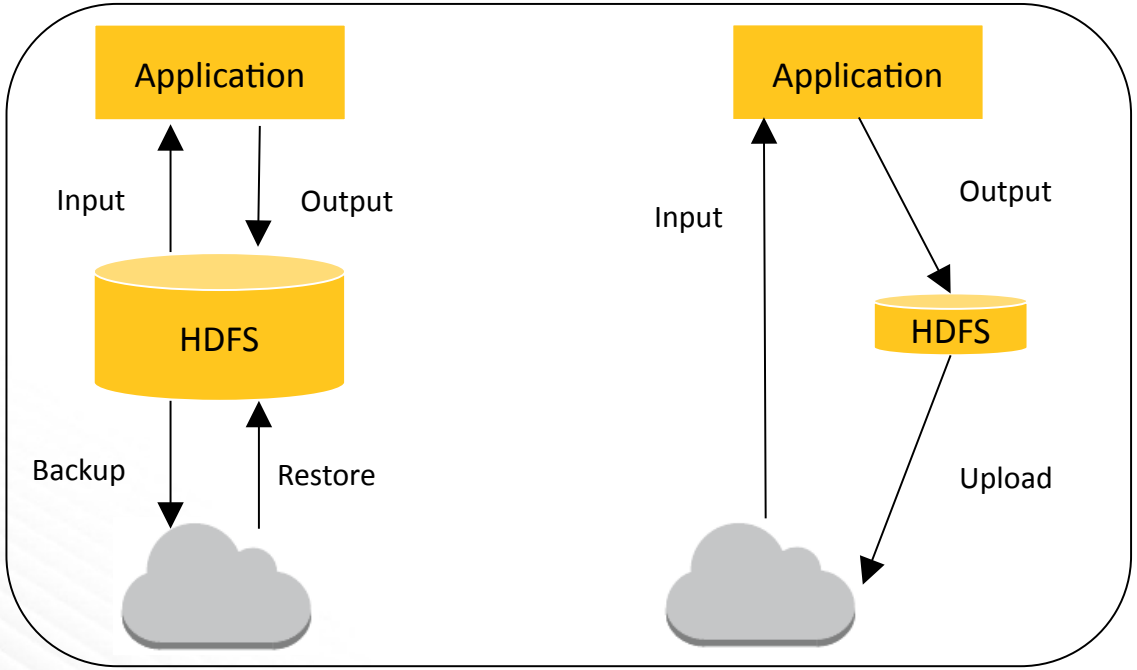
Also Attend 4pm Talk today "Cloudy with a chance of Hadoop…"

HORTONWORKS

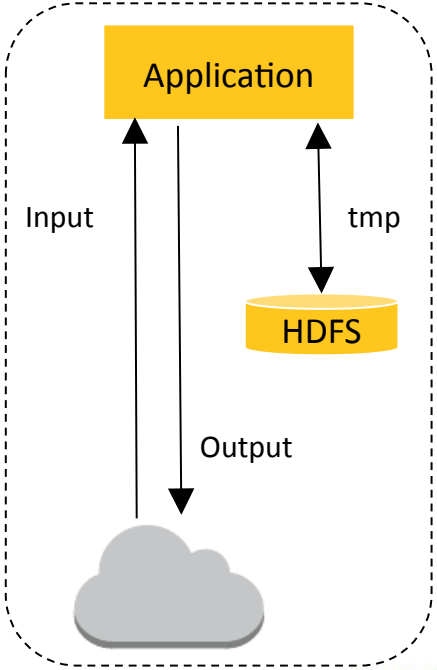
Cloud Storage Integration: Evolution for Agility

Evolution towards cloud storage as the persistent Data Lake

Goal



AWS –*today* ->>>>>



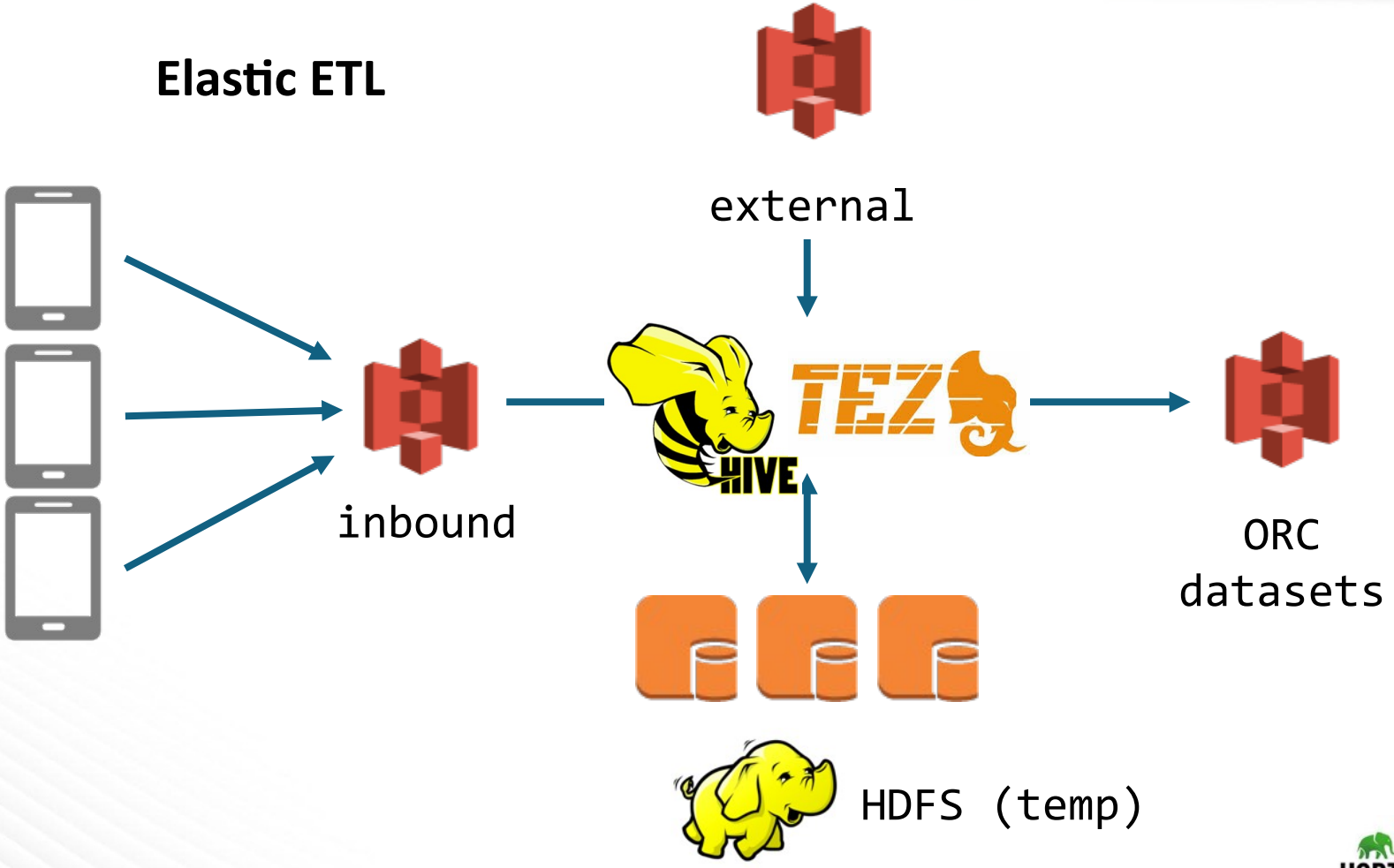
Azure

HORTONWORKS

Some Use Cases

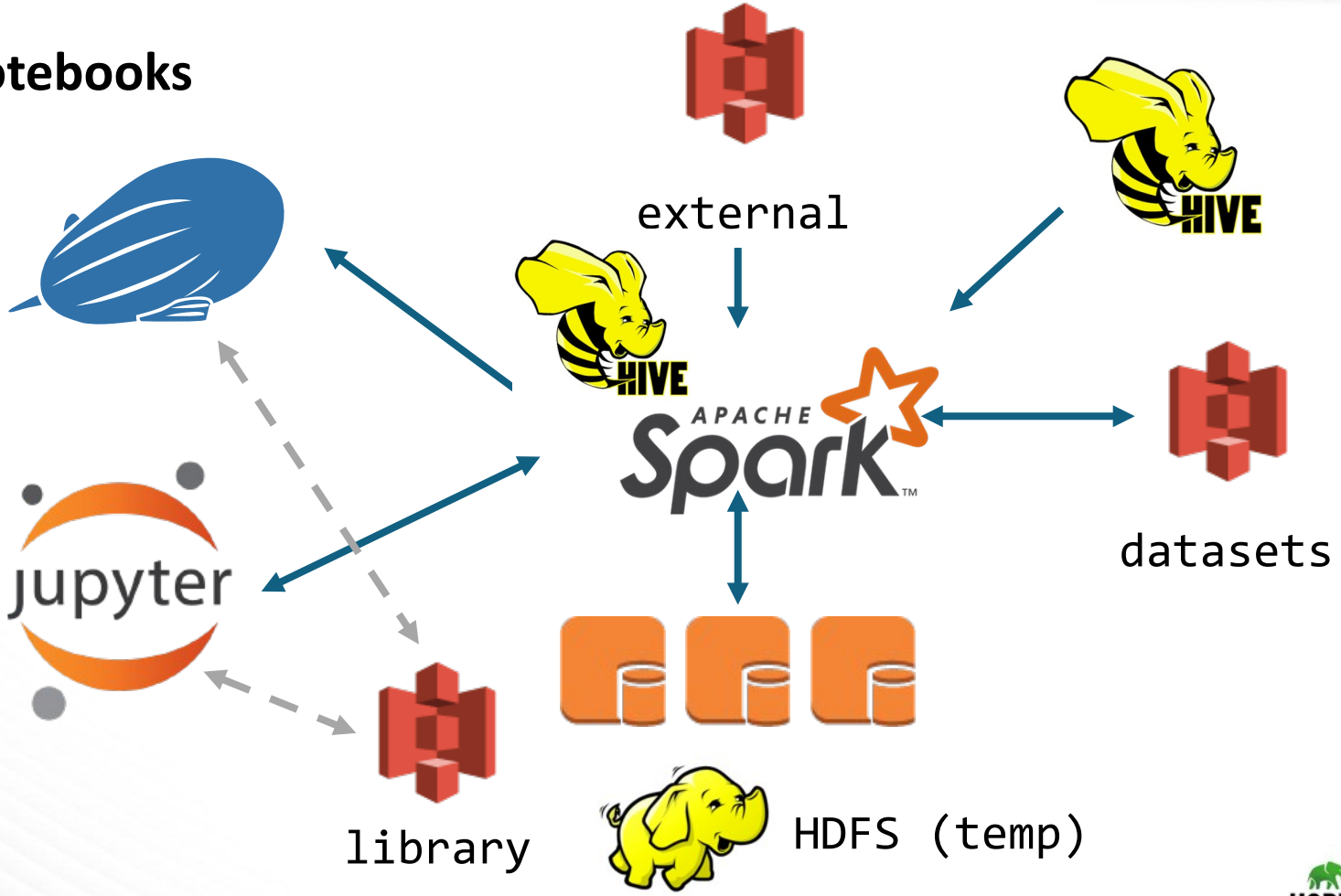
HORTONWORKS®
POWERING THE FUTURE OF DATA™

Elastic ETL



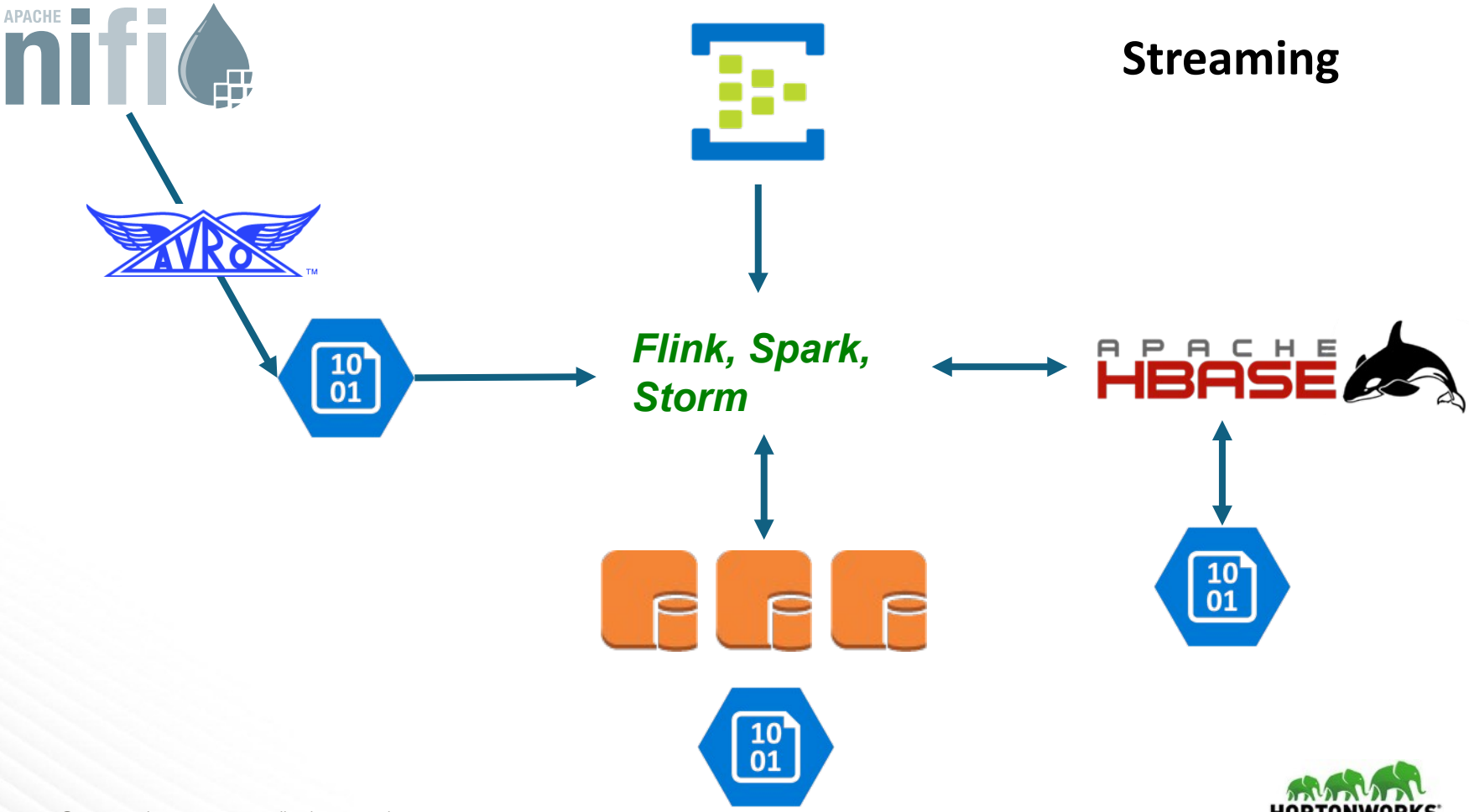
HORTONWORKS

Notebooks



HORTONWORKS

Streaming



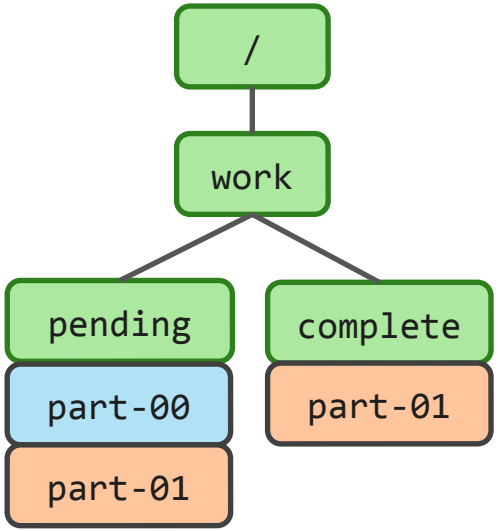
HORTONWORKS

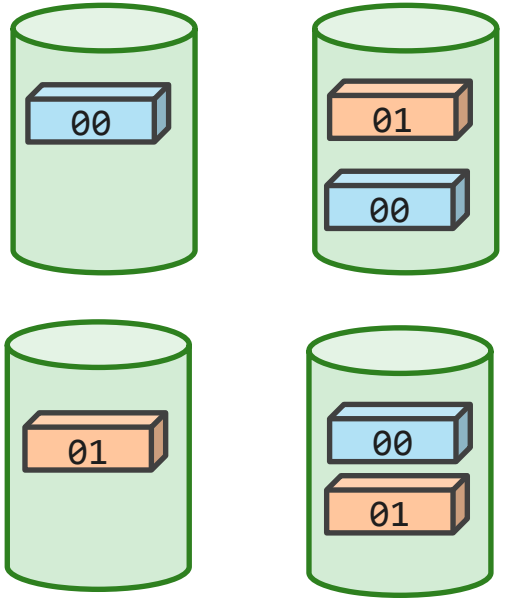
Danger:
Object stores are not filesystems

Cost & Geo-distribution over Consistency and Performance

HORTONWORKS®
POWERING THE FUTURE OF DATA™

A Filesystem: Directories, Files → Data



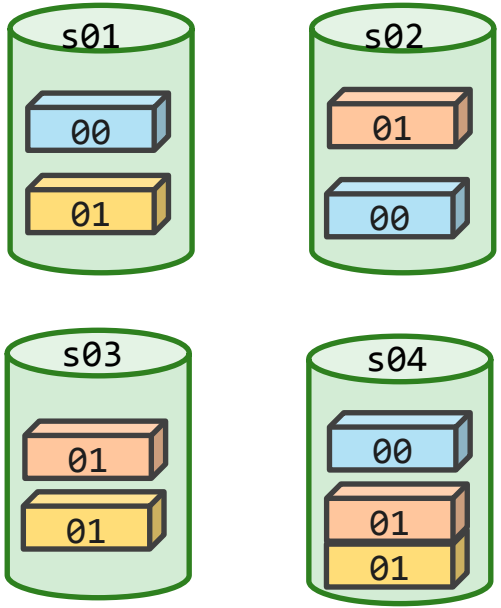



```
rename("/work/pending/part-01", "/work/complete")
```


HORTONWORKS

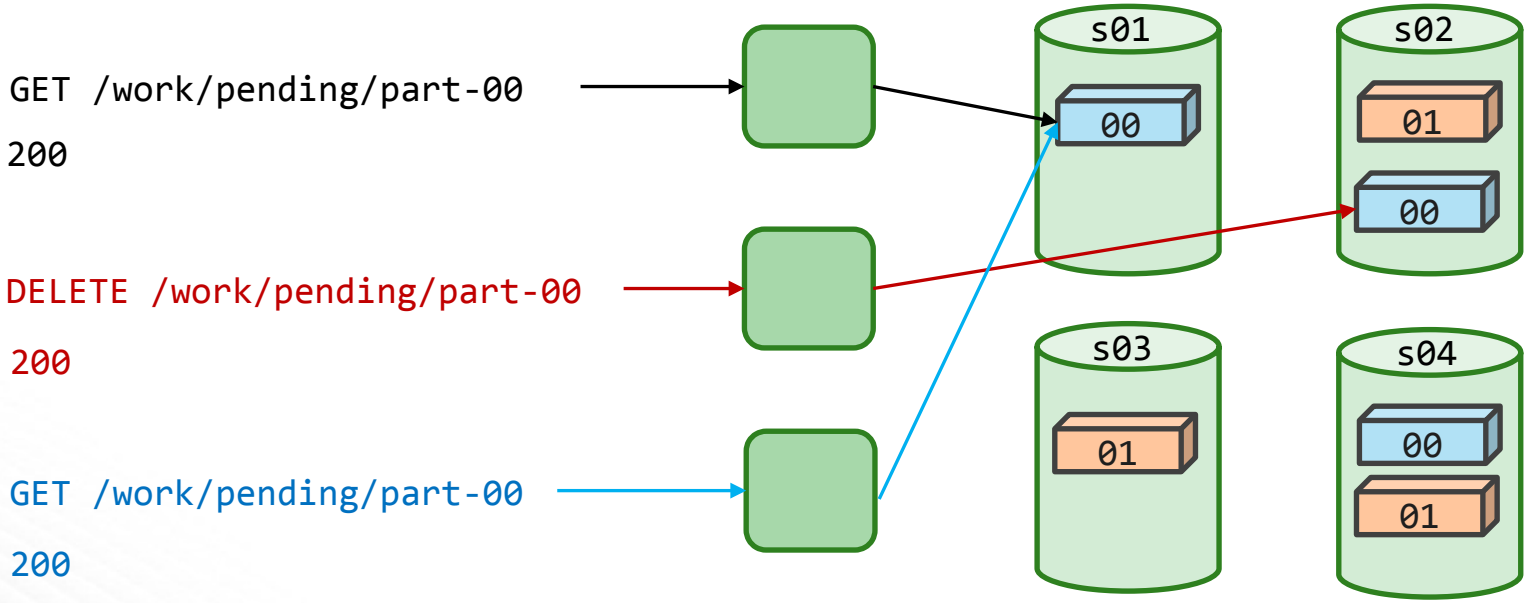
Object Store: hash(name)⇒data

```
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")
```



HORTONWORKS

Often: Eventually Consistent



HORTONWORKS

Eventual Consistency problems

- When listing a directory
  - Newly created files may not yet be visible, deleted ones still present
- After updating a file
  - Opening and reading the file may still return the previous data
- After deleting a file
  - Opening the file may succeed, returning the data
- While reading an object
  - If object is updated or deleted during the process

HORTONWORKS

The dangers of Eventual Consistency and Atomicity

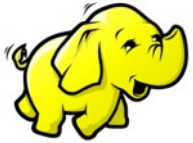
- Temp Data leftovers
  - Annoying Garbage or Worse if direct output committer is used
- List inconsistency means new data may not be visible
- Lack of atomic `rename()` can leave output directories inconsistent

You can get bad or missing data and not even notice

Especially if only a portion of your large data is missing

HORTONWORKS

`org.apache.hadoop.fs.FileSystem`



`hdfs`

`wasb`

`s3a`

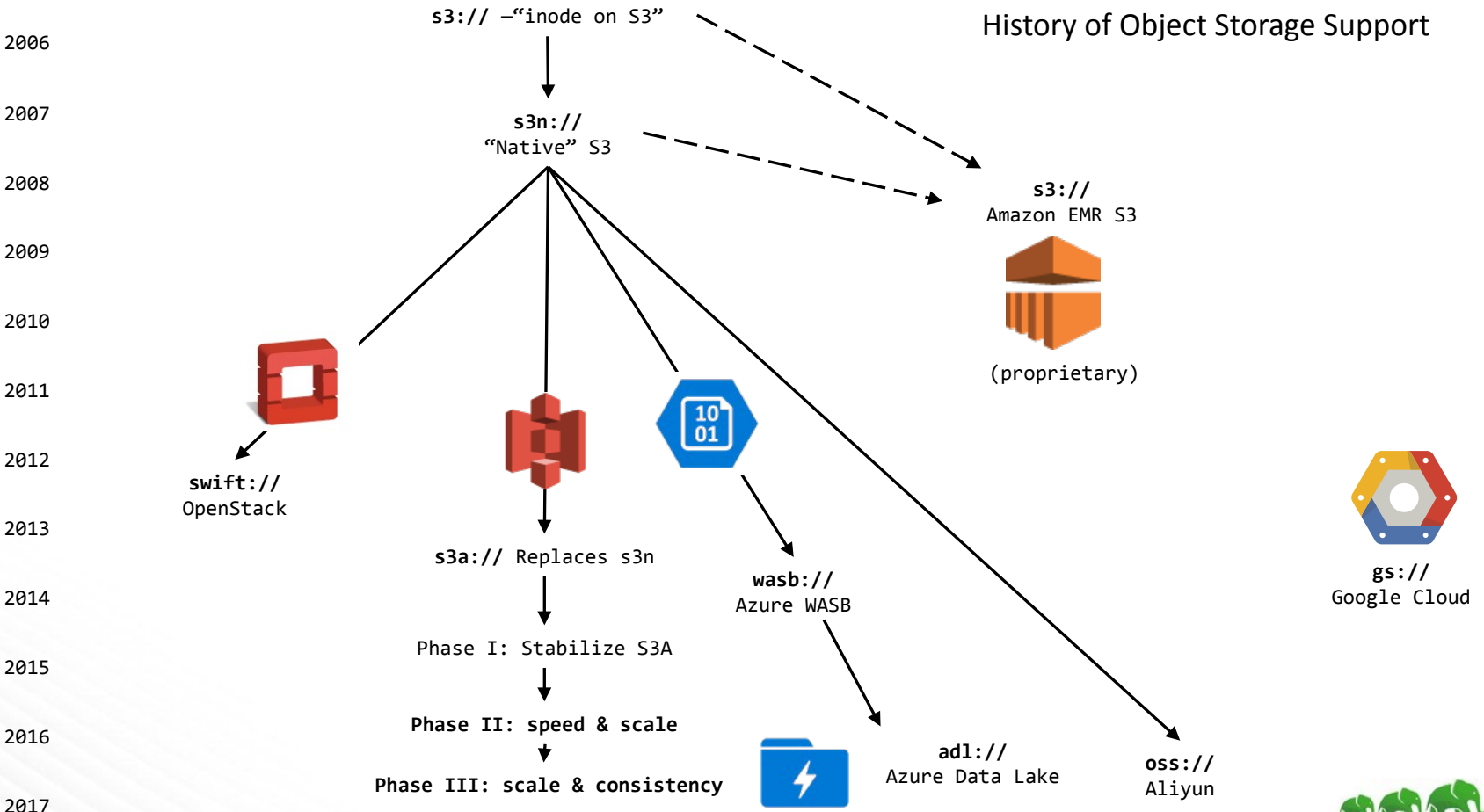
`swift`

`adl`

`gs`

HORTONWORKS

History of Object Storage Support



HORTONWORKS

Cloud Storage Connectors

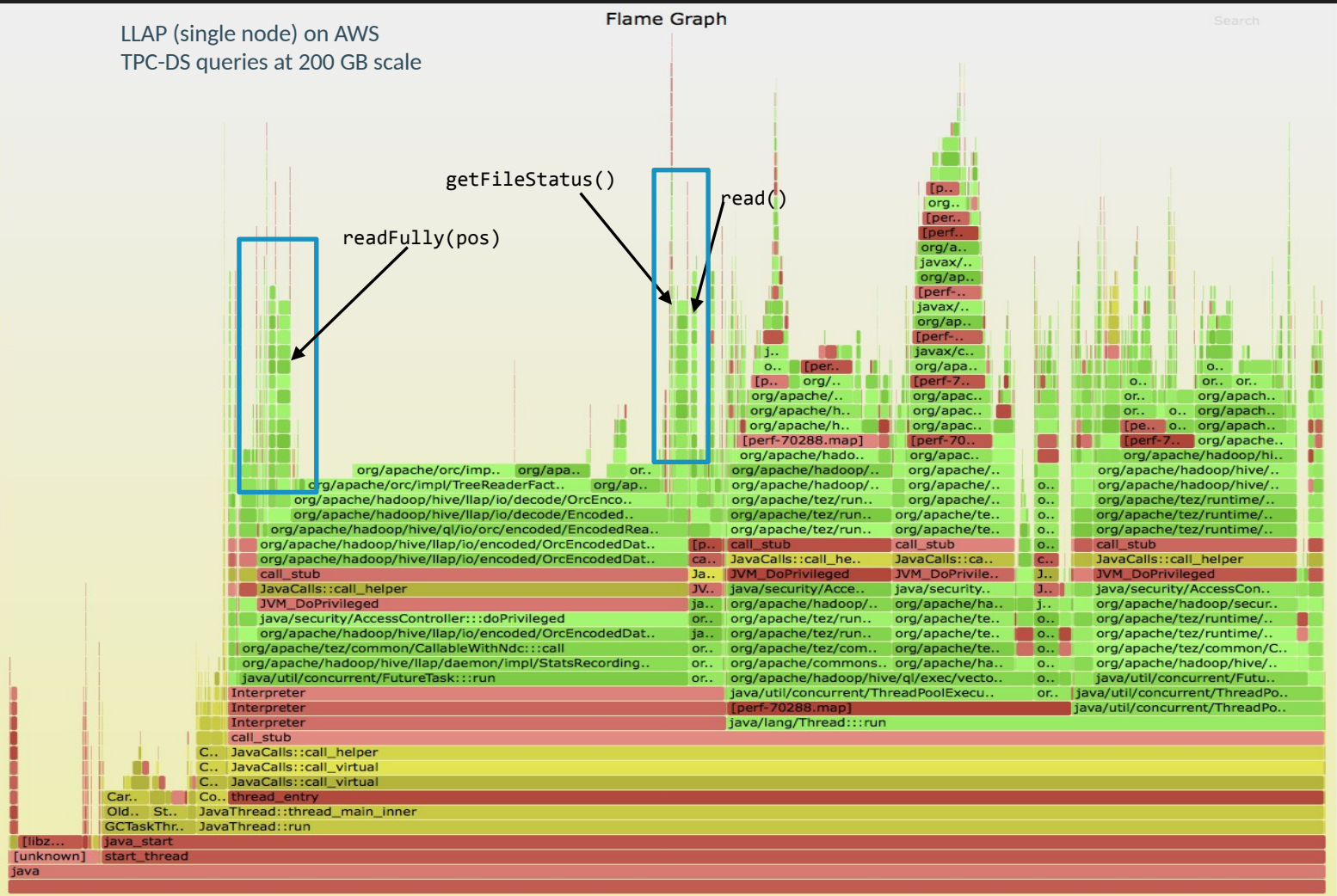
Azure	WASB	• Strongly consistent • Good performance • Well-tested on applications (incl. HBase)
	ADL	• Strongly consistent • Tuned for big data analytics workloads
Amazon Web Services	S3A	• Eventually consistent - consistency work in progress by Hortonworks • Performance improvements recently and in progress • Active development in Apache
	EMRFS	• Proprietary connector used in EMR • Optional strong consistency for a cost
Google Cloud Platform	GCS	• Multiple configurable consistency policies • Currently Google open source • Good performance • Could improve test coverage

HORTONWORKS

Problem: S3 Analytics is too slow/broken

1. Analyze benchmarks and bug-reports
2. Optimize the non-io metadataops (very cheap on HDFS)
3. Fix Read path for Columnar Data
4. Fix Write path
5. Improve query partitioning
6. 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)

listFiles(Path)

rename(source, dest)
```


HORTONWORKS

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

HORTONWORKS

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 2.8/HDP 2.6 transforms I/O performance!

```
// forward seek by skipping stream
fs.s3a.readahead.range=256K

// faster backward seek for Columnar Storage
fs.s3a.experimental.input.fadvise=random

// Write-IO - enhanced data upload (parallel background uploads)
// Additional flags for mem vs disk
fs.s3a.fast.output.enabled=true
fs.s3a.multipart.size=32M
fs.s3a.fast.upload.active.blocks=8

// Additional per-bucket flags
```

—see HADOOP-11694 for lots more!

HORTONWORKS

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

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


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

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

High performance random IO for column data

HORTONWORKS

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

HORTONWORKS

Write Pipeline

- PUT blocks as part of a multipart
- 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=32M
fs.s3a.fast.upload.active.blocks=8
```


HORTONWORKS

S3guard
Fast, consistent S3 metadata

HADOOP-13445

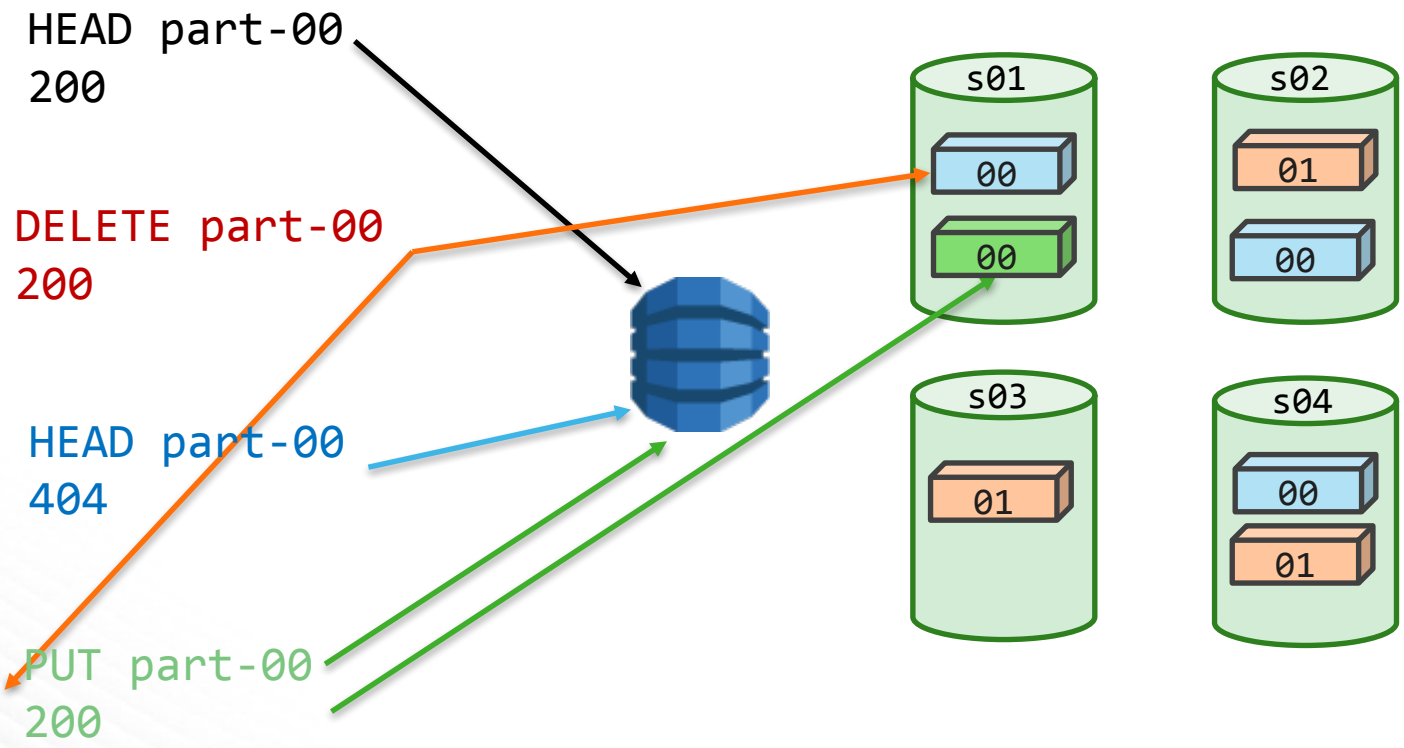
HORTONWORKS
POWERING THE FUTURE OF DATA

S3Guard: Fast And Consistent S3 Metadata

- Goals
  - Provide consistent list and get status operations on S3 objects written with S3Guard enabled
    - `listStatus()` after `put` and `delete`
    - `getFileStatus()` after `put` and `delete`
  - Performance improvements that impact real workloads
  - Provide tools to manage associated metadata and caching policies.
- Again, 100% open source in Apache Hadoop community
  - Hortonworks, Cloudera, Western Digital, Disney …
- Inspired by Apache licensed *S3mper* project from Netflix
- Seamless integration with S3AFileSystem

HORTONWORKS

Use DynamoDB as fast, consistent metadata store



HORTONWORKS

Installation and use

- Provision a DynamoDB table in same region as HDC cluster & buckets (best cost/IO resilience through shared table across >1 bucket)
- Enable S3Guard in clients
- Authoritative mode = fastest —but can we trust it yet?

HORTONWORKS

Settings

```
<property>
  <name>fs.s3a.metadatastore.impl</name>
  <value>org.apache.hadoop.fs.s3a.s3guard.DynamoDBMetadataStore</value>
</property>

<property>
  <name>fs.s3a.s3guard.ddb.table</name>
  <value>ireland-team</value>
</property>

<property>
  <name>fs.s3a.metadatastore.authoritative</name>
  <value>false</value>
</property>
```


HORTONWORKS

hadoop s3guard

```
hadoop s3guard init -meta dynamodb://ireland-team s3a://ireland-1

hadoop s3guard import s3a://ireland-1

hadoop s3guard diff s3a://ireland-1

hadoop s3guard destroy s3a://ireland-1

hadoop s3guard prune -days 7  s3a://ireland-1

 (see also DynamoDB TTL option)
```

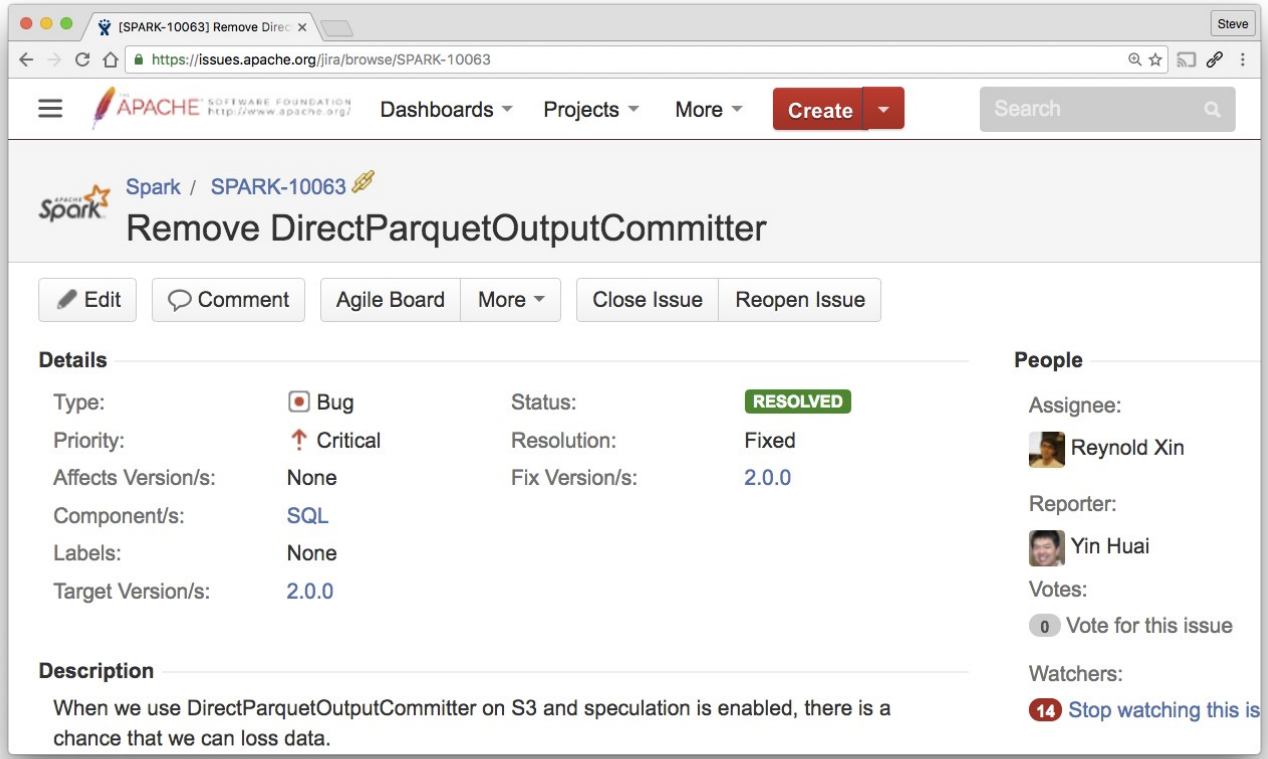
Dynamic table creation

```
<property>
  <name>fs.s3a.s3guard.ddb.table.create</name>
  <value>true</value>
</property>

<property>
  <name>fs.s3a.s3guard.ddb.table.capacity.read</name>
  <value>10</value>
</property>

<property>
  <name>fs.s3a.s3guard.ddb.table.capacity.write</name>
  <value>10</value>
</property>
```

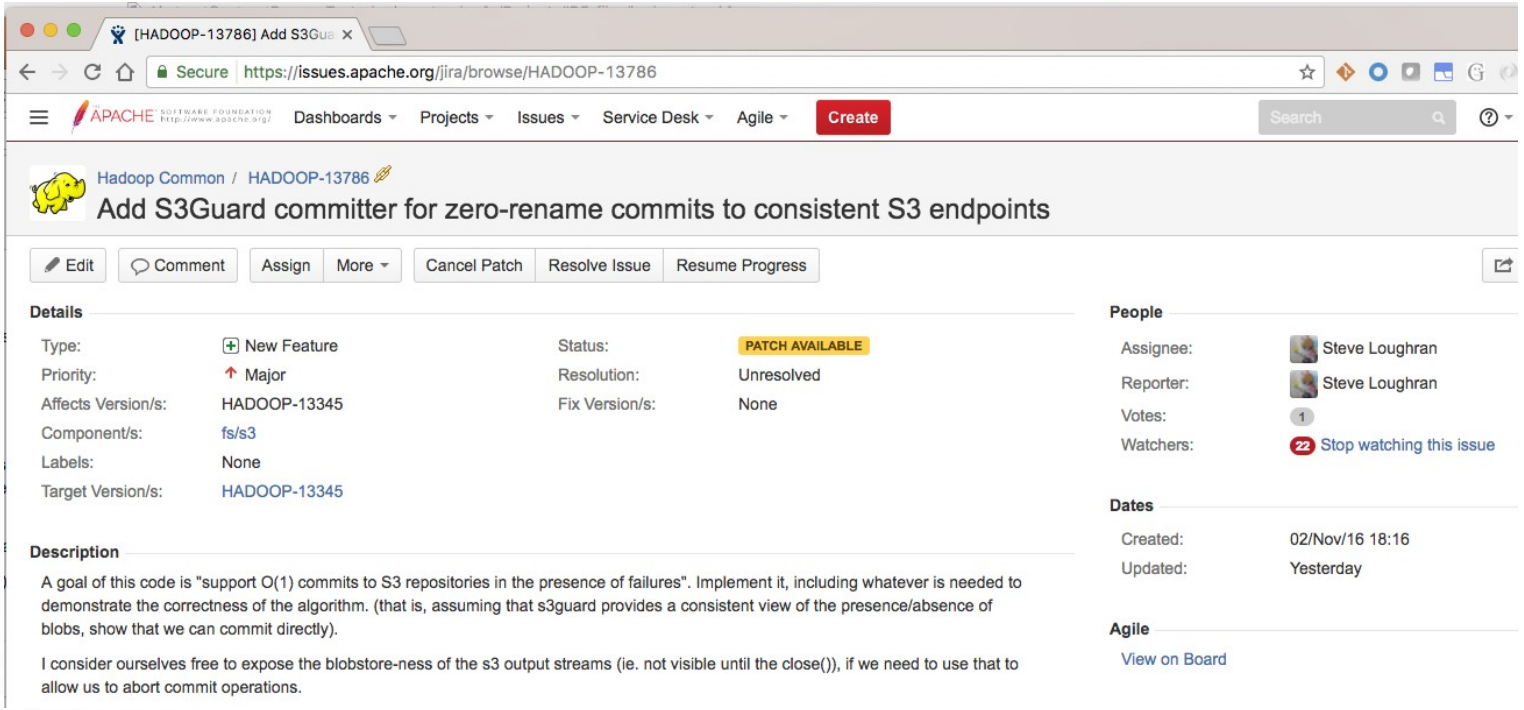
Better to share table, explicitly create, monitor in AWS console

HORTONWORKS

Spark's Direct Output Committer? Risk of Corruption of data



HORTONWORKS



HORTONWORKS

Netflix Staging Committer

1. Saves output to `file://`
2. Task commit: upload to S3A as multipart PUT —*but does not commit the PUT, just saves the information about it to hdfs://*
3. Normal commit protocol manages task and job data promotion in HDFS
4. Final Job committer reads pending information and generates final PUT —possibly from a different host

Outcome:

- No visible overwrite until final job commit: resilience and speculation
- Task commit time = `data/bandwidth`
- Job commit time = POST * `#files`

HORTONWORKS

Availability

- Read + Write in HDP 2.6 and Apache Hadoop 2.8
- S3Guard: preview of DDB caching in HDC
- Commit to ASF trunk/branch-2 this month
  ...then backport
- Zero-rename commit: *work in progress*

Big thanks to:

Rajesh Balamohan

Mingliang Liu

Chris Nauroth

Dominik Bialek

Ram Venkatesh

Everyone in QE, RE

+ everyone who reviewed/tested, patches and added their own, filed bug reports and measured performance

HORTONWORKS®
POWERING THE FUTURE OF DATA™

Questions?

stevel@hortonworks.com @steveloughran
sanjay@hortonworks.com @srr

HORTONWORKS
POWERING THE FUTURE OF DATA