LLAP: Building Cloud-First BI

Sergey Shelukhin

LLAP: Building Cloud-First BI

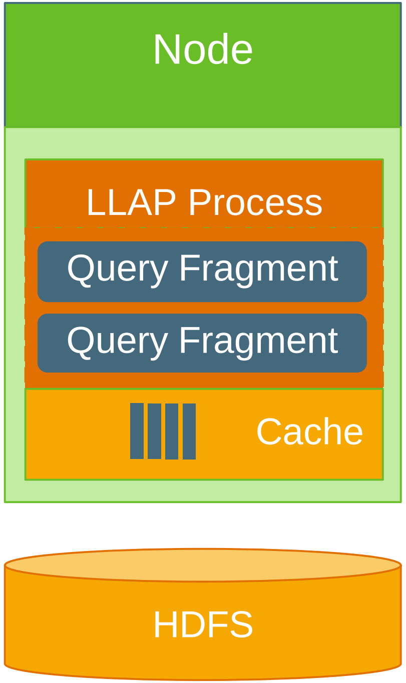
- What is LLAP? Overview
- Cloud-first BI
  - Efficient, scalable multi-user execution and caching
  - Secure
  - Universal (not just for Hive)
  - Production-ready tools
- How to run LLAP

Hortonworks

Overview (AKA the old slides)

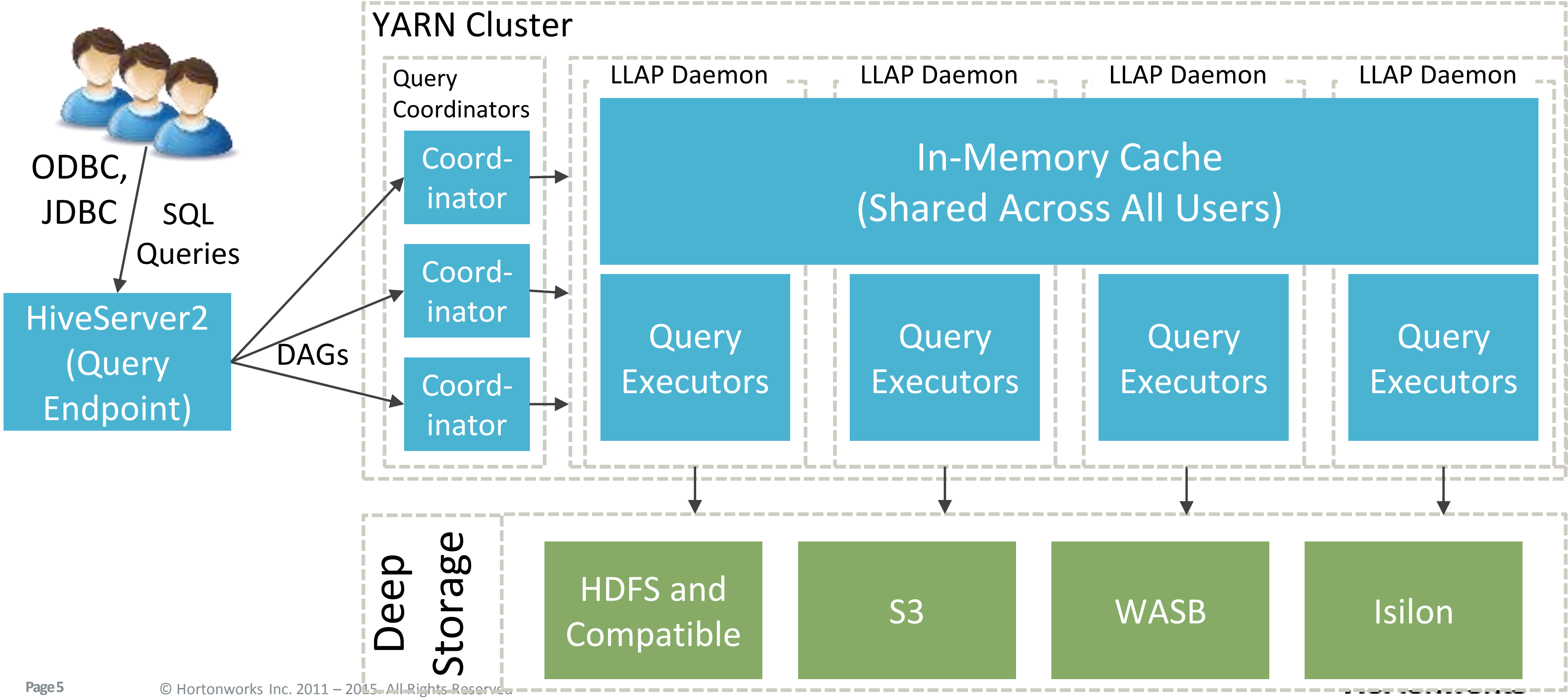
What is LLAP?

- **Hybrid model combining daemons and containers for fast, concurrent execution of analytical workloads (e.g. Hive SQL queries)**
  - Concurrent queries without specialized YARN queue setup
  - Multi-threaded execution of vectorized operator pipelines
- Asynchronous IO and efficient in-memory caching
- Relational view of the data available thru the API
  - High performance scans, code pushdown, centralized security
- Not an "execution engine" (like Tez, MR, Spark)
- Not a storage substrate – reads from HDFS/S3/…



LLAP and Hive

- Transparent to users, BI tools, etc. – HS2/JDBC is the access point



LLAP and Hive

- LLAP is Hive
  - Supports all file formats that Hive does
  - Hive Operators used for processing, same compiler, etc.
    - ⇒ Automatic support for most new optimizations and features
  - HS2 controls the concurrent queries (session pool)
  - Each Query coordinated by a Tez AM; LLAP hosts Tez shuffle
- Can run in parallel with container-based jobs
  - After perf testing, we recommend running most Hive workloads in LLAP
  - In this new model, the cluster capacity is divided proportionally if needed

Hortonworks

LLAP in a BI system - overview

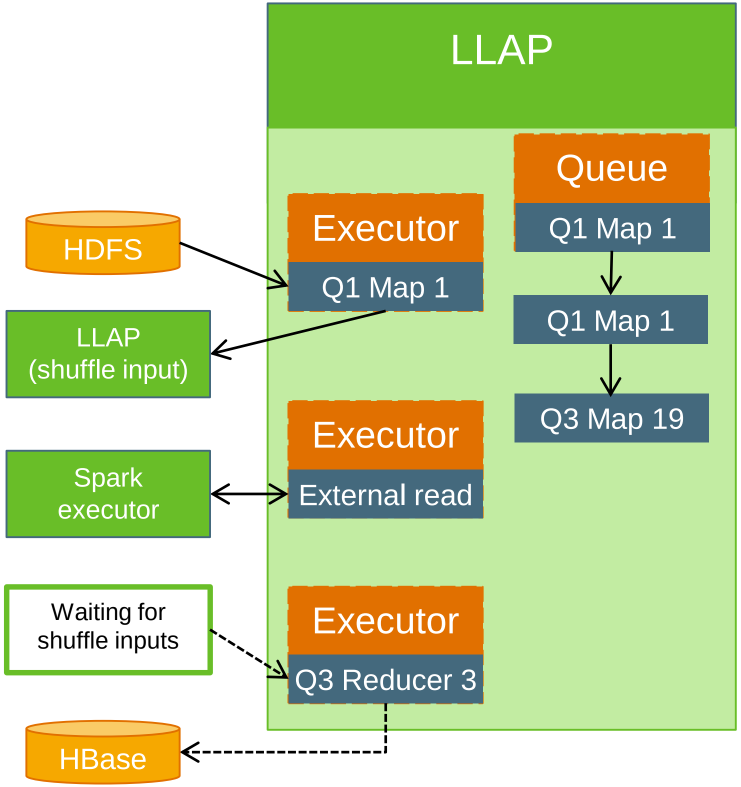
- No split brain – a single platform for ETL…
  - Fault-tolerant components and proven scalability (runs **TPCDS 100 Tb**)
  - ANSI SQL, CB Optimizer, ACID transactions
- …and BI
  - Vectorized engine for fast processing
  - Intelligent scheduling of different workloads running together
  - Caching, incl. on local disk (SSD), to avoid expensive reads
- Zero-ETL analytics with efficient caching of text data
- External access, one metadata store, unified security

Hortonworks

Efficient BI queries (AKA the new slides)

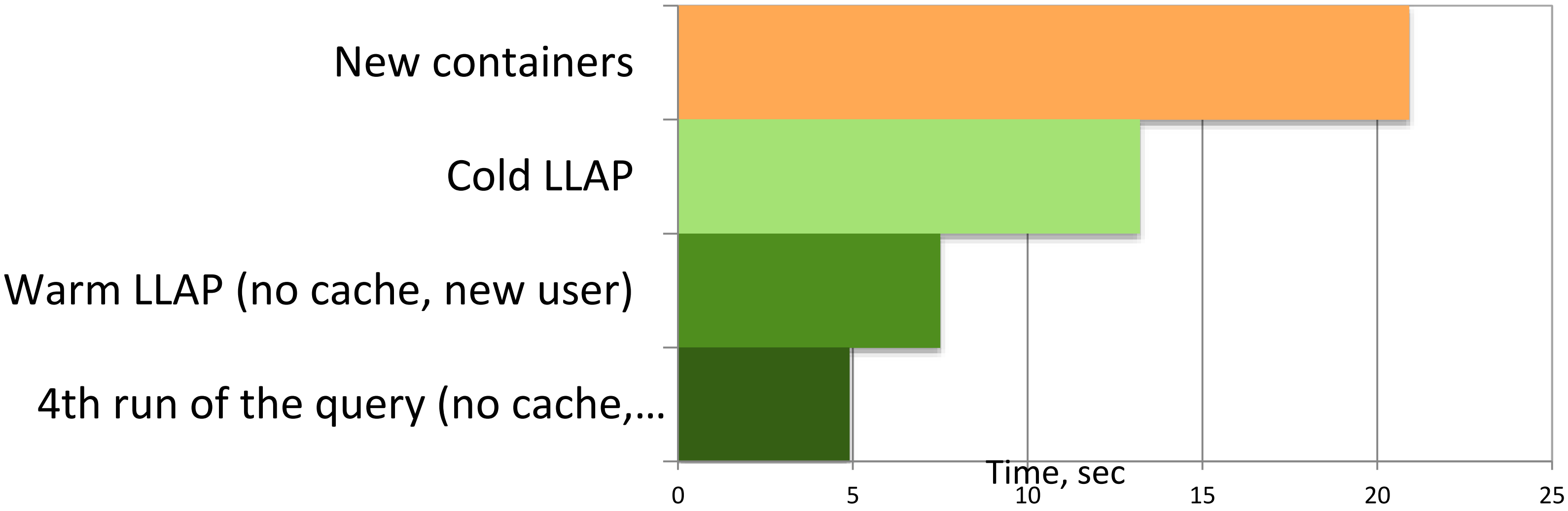
Short overview – execution

- LLAP daemon has a number of executors (think containers) that run "fragments"
- Fragments are parts of multiple parallel workloads (e.g. Hive SQL queries)
- Work queue with pluggable priority
  - Geared towards low latency queries over long-running queries (by default)
- I/O is similar to containers – read/write to HDFS, shuffle, other storages and formats
- Streaming output for data API

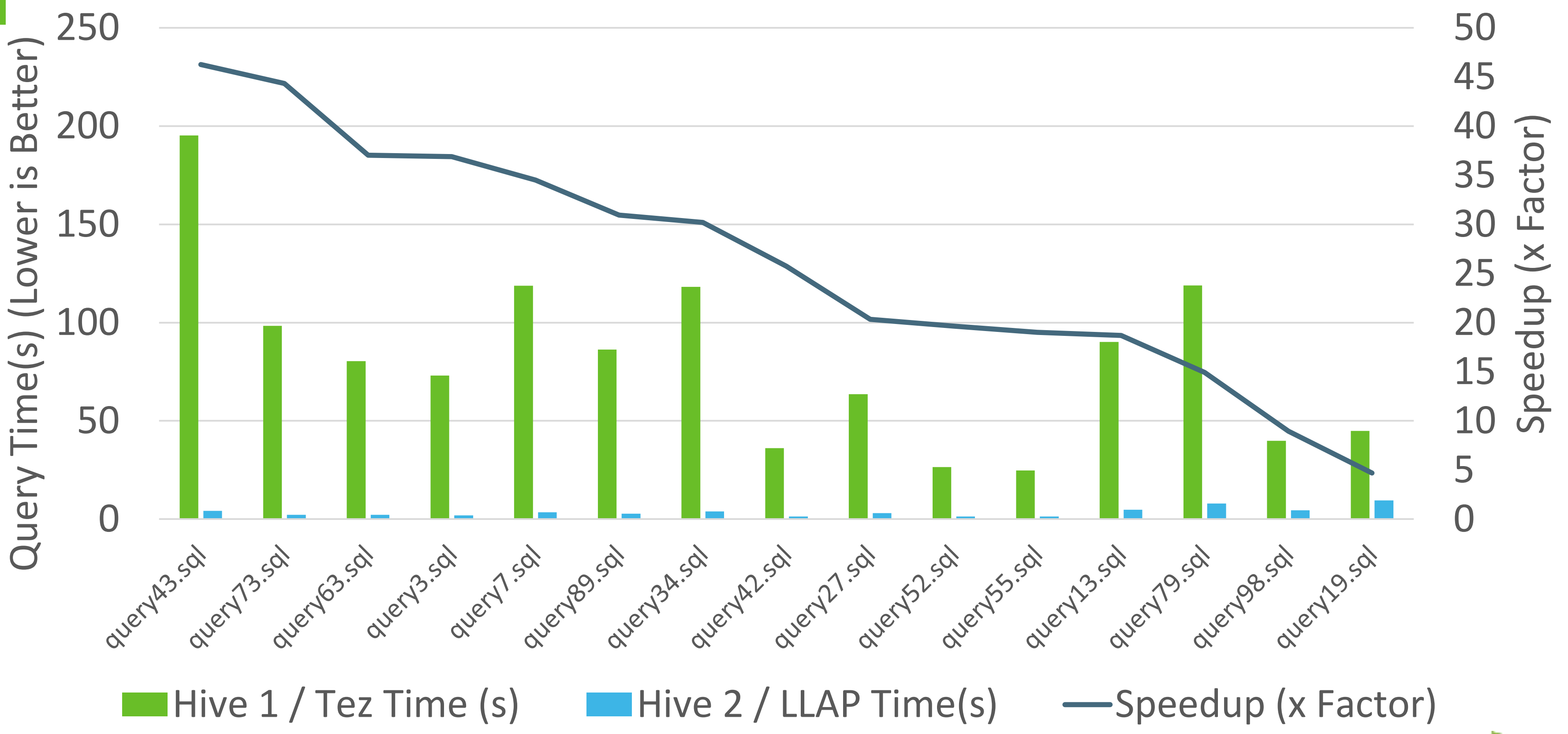


Efficiency for individual queries

- Eliminates container startup costs
- JIT optimizer has a chance to work (esp. for vectorization)
- Data sharing (hash join tables, etc.)



Individual queries – LLAP vs Hive 1 (x26 faster)



Hortonworks

Hive + LLAP vs Impala; TPCDS 10Tb on 10 nodes

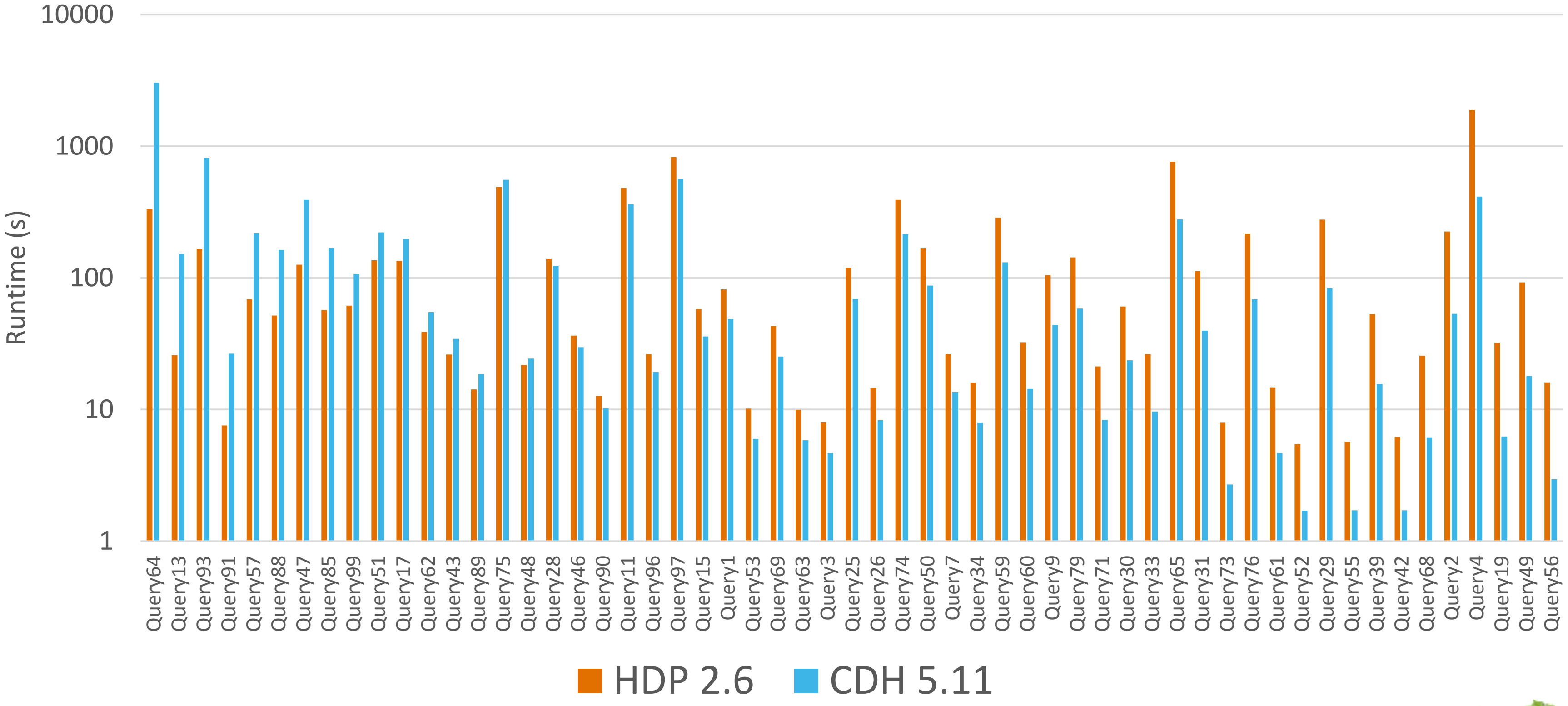
- LLAP - 180 GB memory size, 32 GB cache

2 hours, 24 mins(*) Total Time in HDP 2.6	**2 hours, 32 min(*)** Total Time in CDH 5.11
99 / 99 Unmodified TPC-DS Queries Complete In HDP 2.6 at 10TB	**57 / 99(**)** Unmodified TPC-DS Queries Complete In CDH 5.11 at 10TB

(*) Among queries which complete in both engines.
(**) 3 runtime failures, 39 compile failures

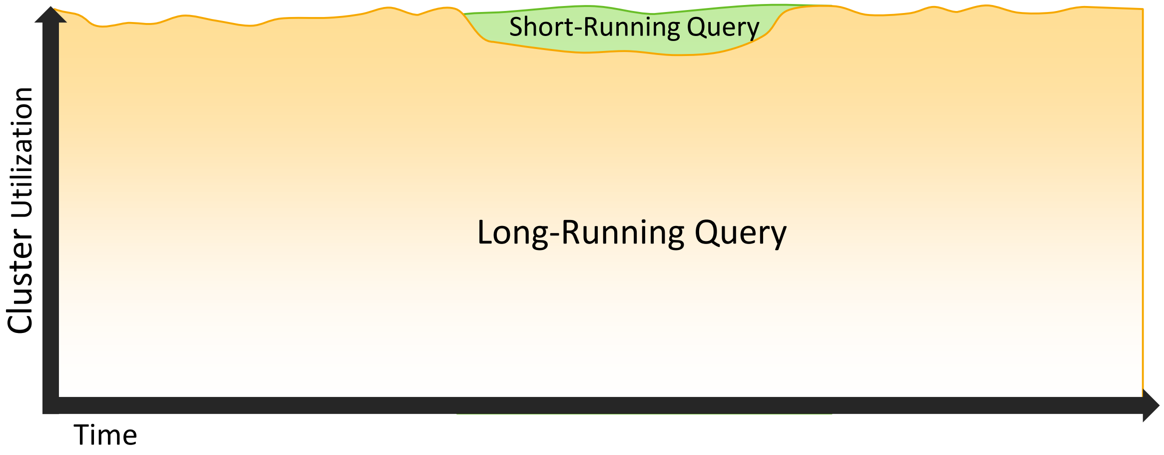
Hortonworks

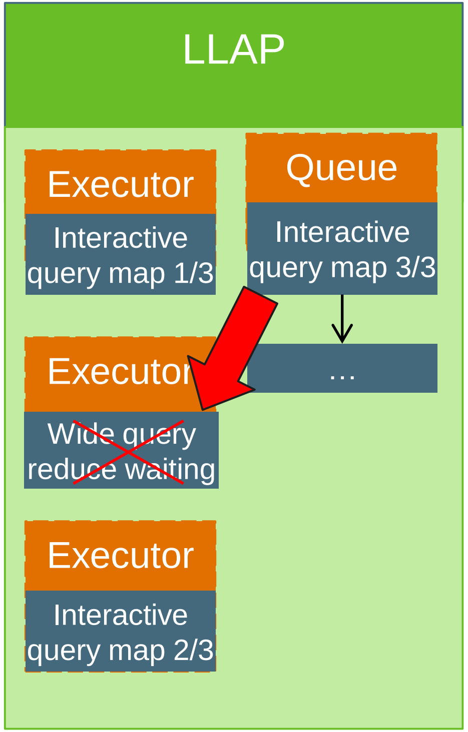
Hive + LLAP vs Impala (log scale; lower is better)



Parallel queries – priorities, preemption

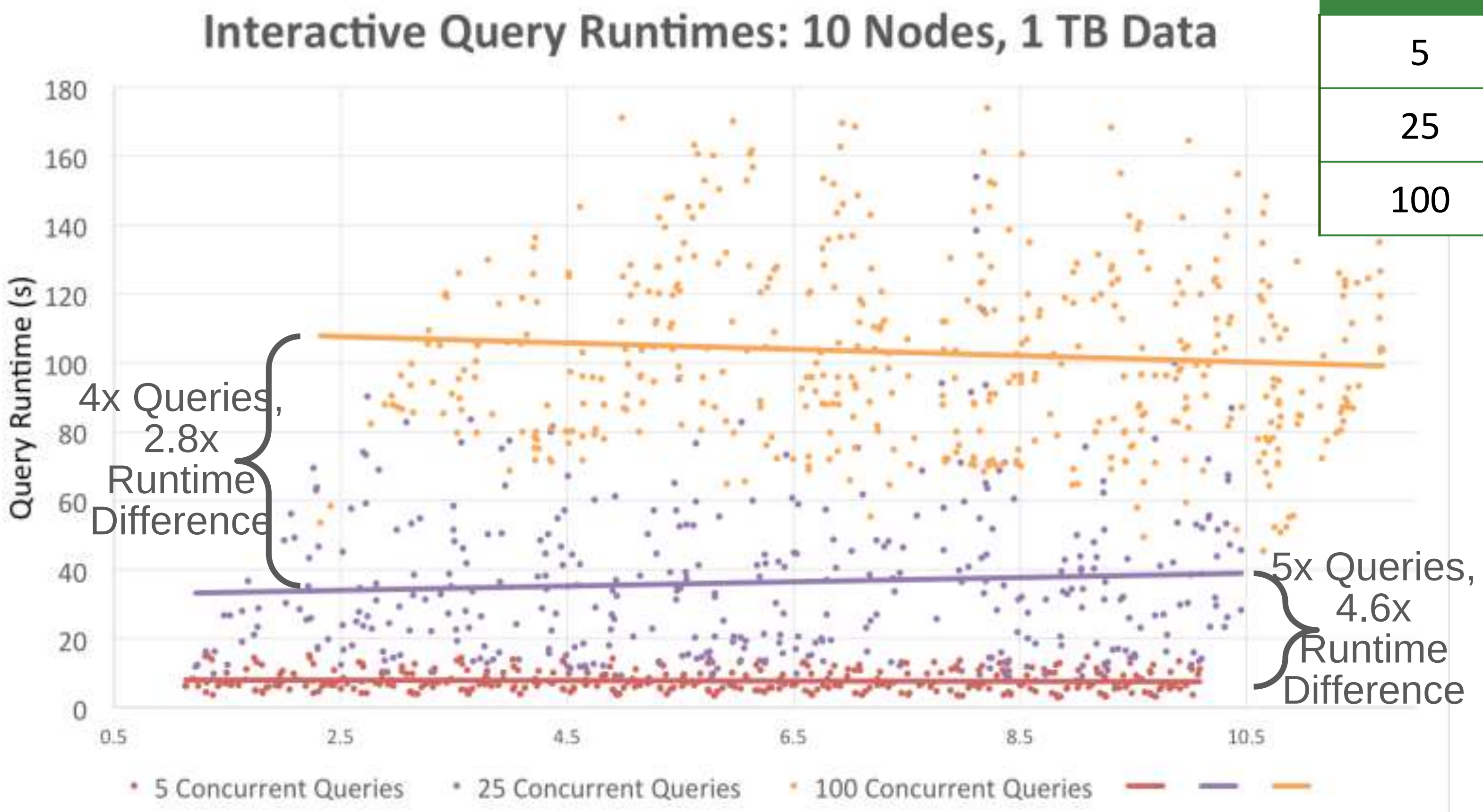
- Lower-priority fragments can be preempted
  - For example, a fragment can start running before its inputs are ready, for better pipelining; such fragments may be preempted
- LLAP work queue examines the DAG parameters to give preference to interactive (BI) queries





Hortonworks

Efficiency for parallel queries



Concurrent Queries	Average Runtime
5	7.76s
25	36.24s
100	102.89s

Hortonworks

Central workload management (WIP)

- Easier query SLAs and priorities
  - Separate ETL and BI queries
- Controls for massively parallel workloads
- Manage resource allocations
  - Rules for different queues and users – guaranteed resources allocations, etc.
  - Multiple "equal" queries – FCFS, equal distribution, ...

Hortonworks

Central workload management (WIP)

- The scheduling can be more flexible than e.g. YARN
  - Scheduler is aware of query characteristics, fragment types, etc.
  - Fragments easy to pre-empt compared to containers
- Runaway query prevention, dynamic policy-based priorities
  - Is it BI or ETL? We can guess from query runtime, data consumed, etc.
  - Ability to "pause" long queries, only allowing leftover resources
  - Ability to size a BI query allocation to e.g. always run in one wave
- Intelligent utilization of slack
  - Queries get guaranteed fractions of the cluster, but can use empty space

Hortonworks

Caching

Caching for BI workloads - basics

- Fine-grained (columnar), compact (dictionary, RLE encoded)
  - Important due to projections over many wide EDW tables
- Prioritized – indexes are cached with higher priority
  - Important to make use of PPD for BI query filters
- Off-heap (no extra GC), supports SSD
  - Saves on cloud reads
- LRFU replacement policy avoids the damage from large scans
  - Since cache is most critical for BI queries, esp. on the cloud

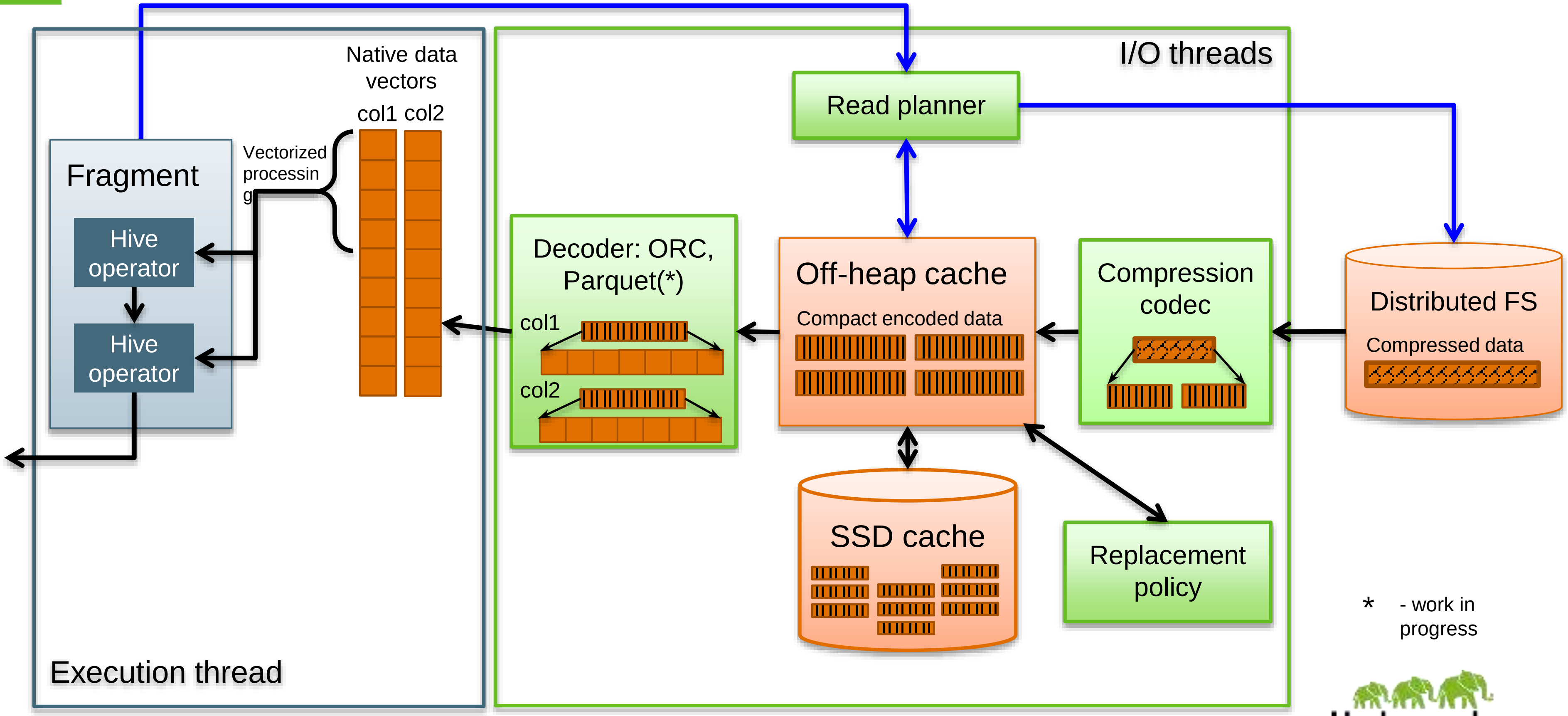
Hortonworks

Caching for BI workloads - basics

- On-prem, does not make <u>large</u> queries <u>on ORC</u> much faster
- On S3/Azure though, reads are much more slow
  - Even disk cache is still much faster than FS reads
  - Especially if text is involved

Hortonworks

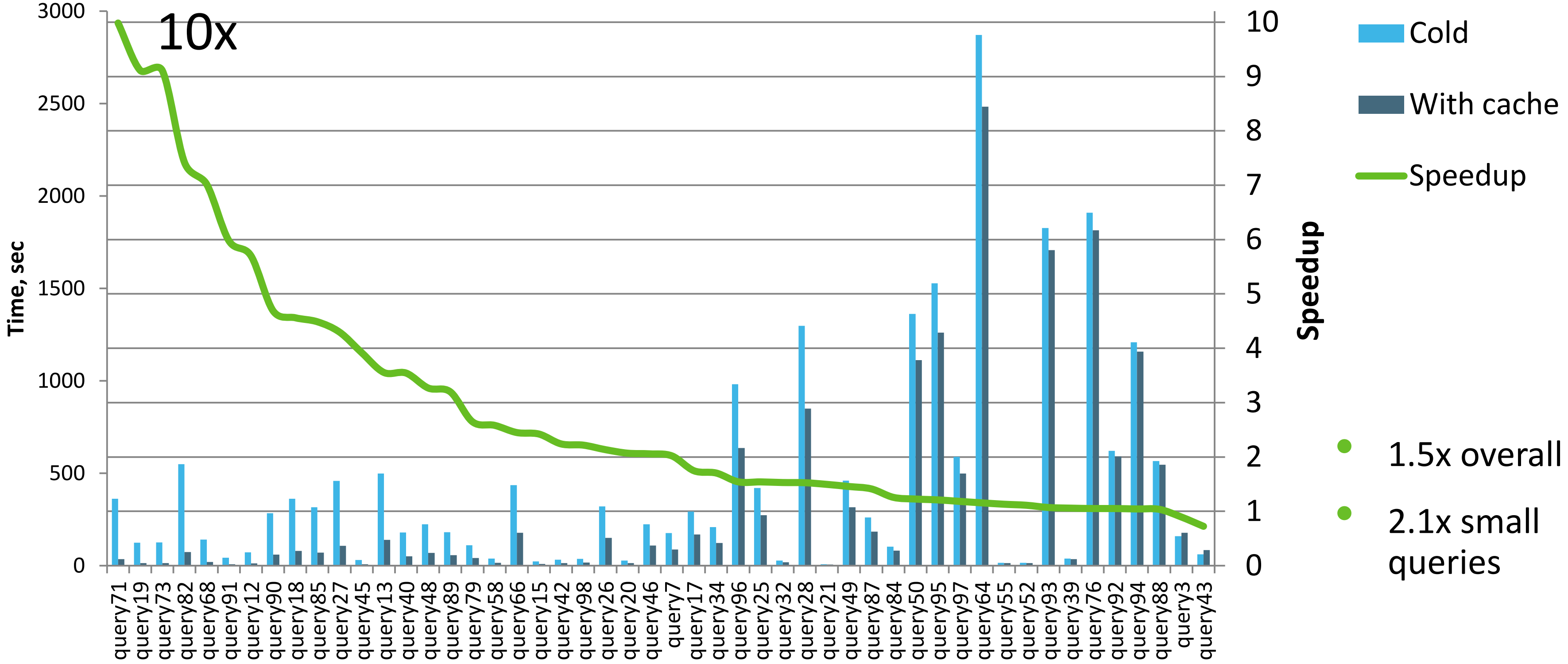
In-memory processing – columnar



* - work in progress

Hortonworks

Up to 10x speed up with a cloud FS (100 Tb dataset, SSD)



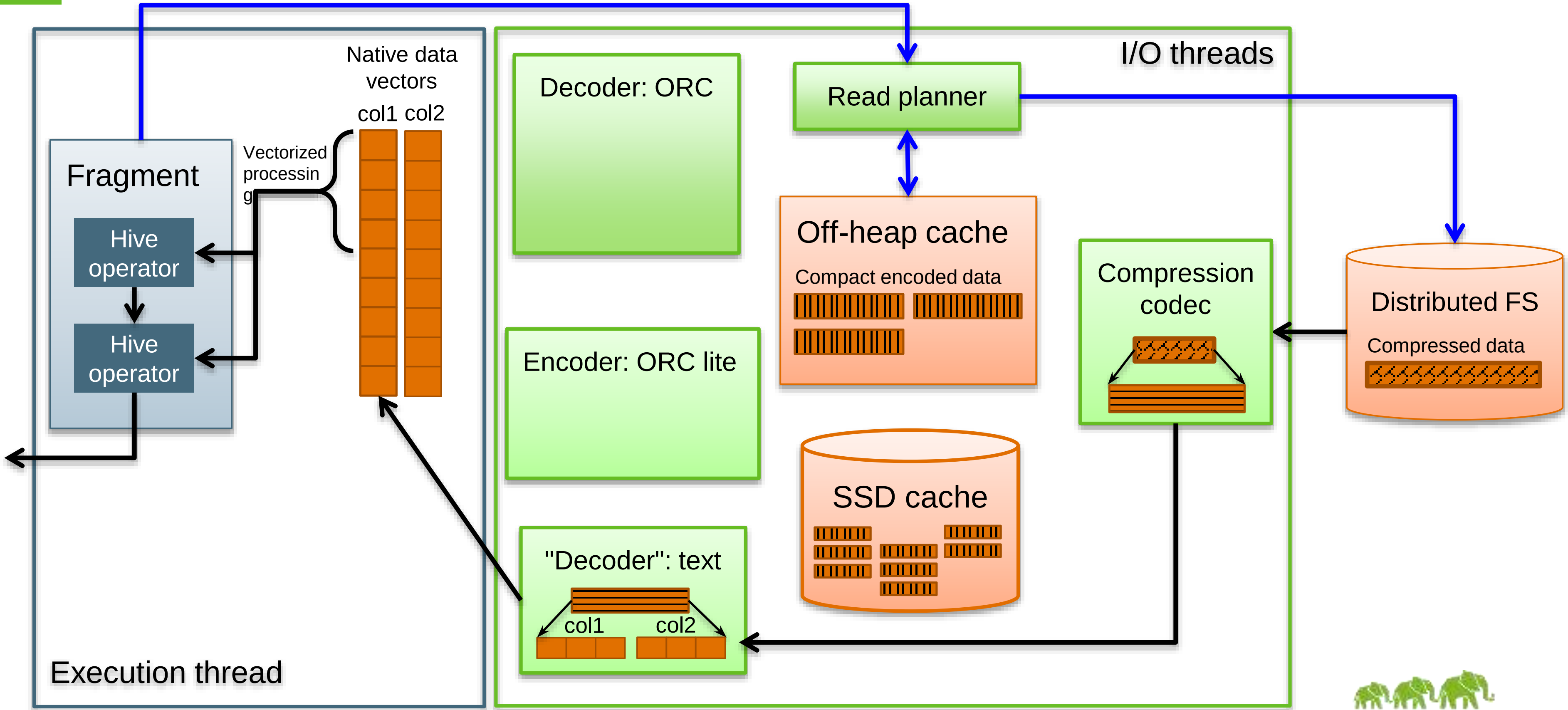
- 1.5x overall
- 2.1x small queries

Hortonworks

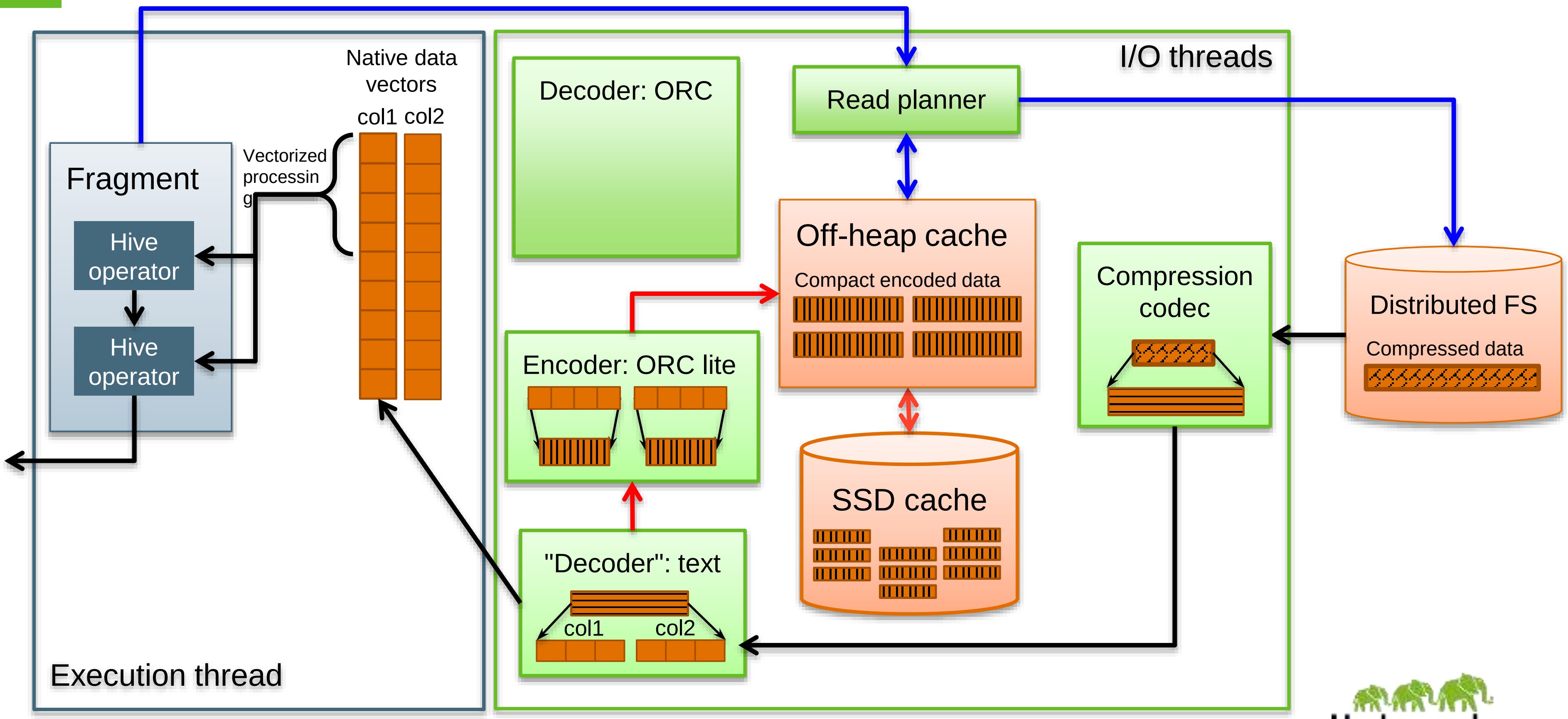
Caching for BI workloads – zero-ETL

- Cache supports columnar data and text
  - ORC is cached natively
  - Parquet runs in LLAP without cache; caching support WIP
- Zero-ETL analytics on CSV and JSON data with text caching
  - Text is efficiently encoded in background; once cached, queries speed up

In-memory processing – text – the first query

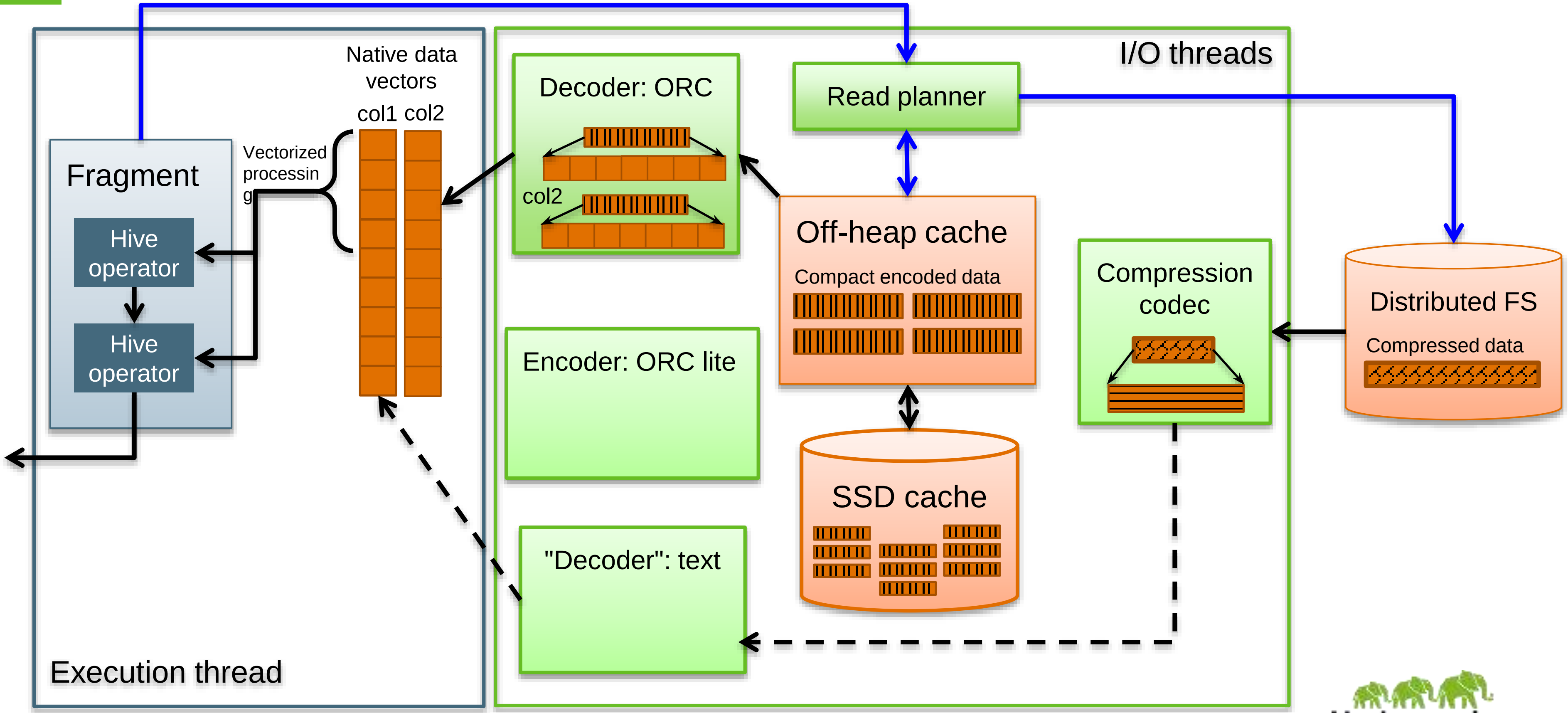


In-memory processing – text – the first query

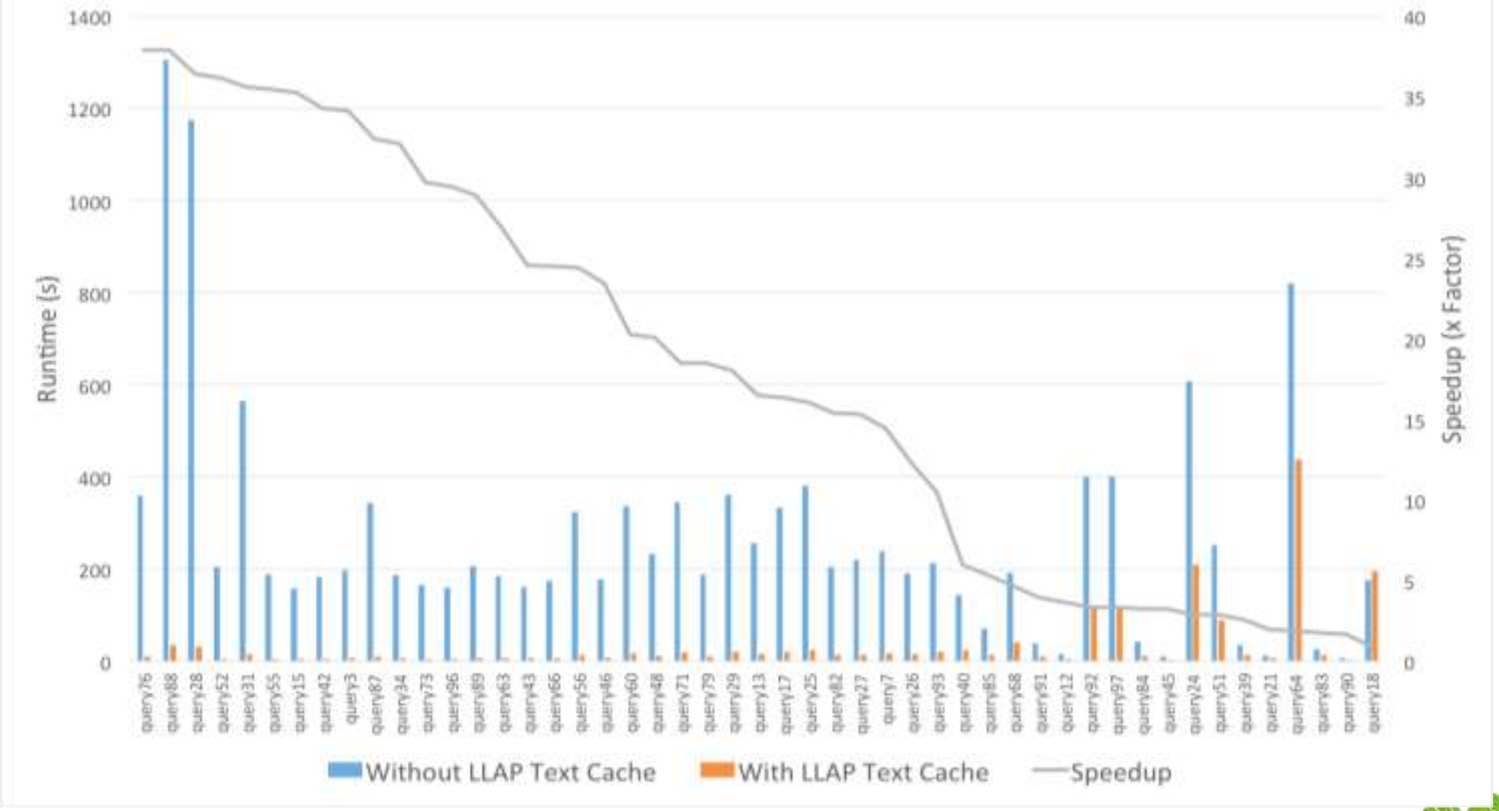


Hortonworks

In-memory processing – text – the second query



Up to 38x speed up with text cache + cloud FS



Hortonworks

Caching for BI workloads – usage patterns

- Usually, the data is loaded periodically (15-60 minutes)
  - Hive ACID is a good fit for such data loads
  - The most recent data is the most accessed
- Avoiding hotspots – locality is configurable
  - Non-strict locality "auto"-creates more replicas
  - Strict cache locality is also possible (for filly in memory/on-SSD data)
  - Locality is based on consistent hashing with some fault tolerance
- Automatically coherent cache – unique-file based
  - Better ACID support (WIP) – updates without invalidating old base files

Hortonworks

External access

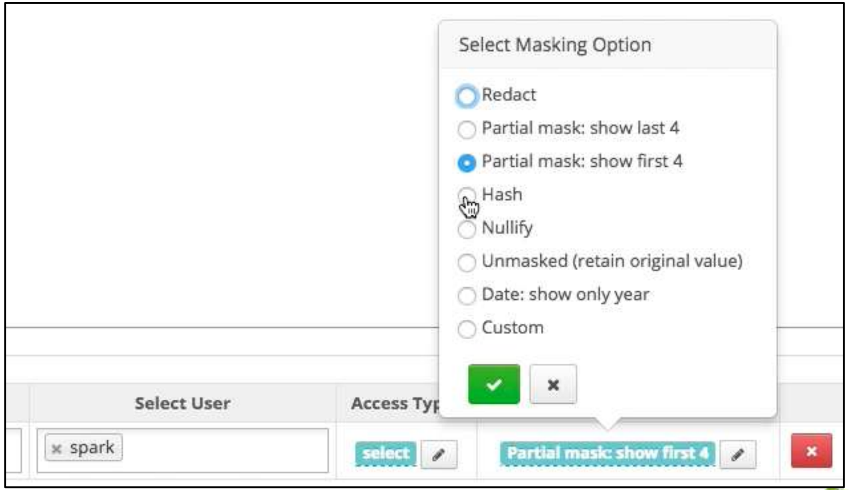
External access – relational view for everyone

- Hive-on-Tez and other DAG executors can use LLAP directly
- LLAP also provides a "relational datanode" view of the data
- Anyone (with access) can push the (approved) code in, from complex query fragments to simple data reads
  - E.g. a Spark DataFrame can be created with LlapInputFormat
- Gives the external services the access to
  - Hive data: centralized, secure data access
  - Ability to read all Hive table types, like ACID transactional tables
  - Hive features: from column-level security, to LLAP columnar cache

Hortonworks

SparkSQL+LLAP example

- Ranger for Hive support cell-level security and masking
  - With SparkSQL utilizing LLAP, this can be used from Spark
  - More at https://hortonworks.com/blog/row-column-level-control-apache-spark/



SparkSQL+LLAP example

- Ranger for Hive support cell-level security and masking
  - With SparkSQL utilizing LLAP, this can be used from Spark
  - More at https://hortonworks.com/blog/row-column-level-control-apache-spark/

```
$ spark-shell --packages
com.hortonworks.spark:spark-llap-assembly_2.11:1.1.1-2.1
--repositories http://repo.hortonworks.com/content/groups/public
--conf spark.sql.hive.llap=true

scala> sql("select * from db_spark.t_customer").show
+------------+------+
|        name|gender|
+------------+------+
|Baraxx xxxxx|     M|
|Donaxx xxxxx|     M|
+------------+------+
```

SparkSQL+LLAP example

- Ranger for Hive support cell-level security and masking
  - With SparkSQL utilizing LLAP, this can be used from Spark
  - More at https://hortonworks.com/blog/row-column-level-control-apache-spark/

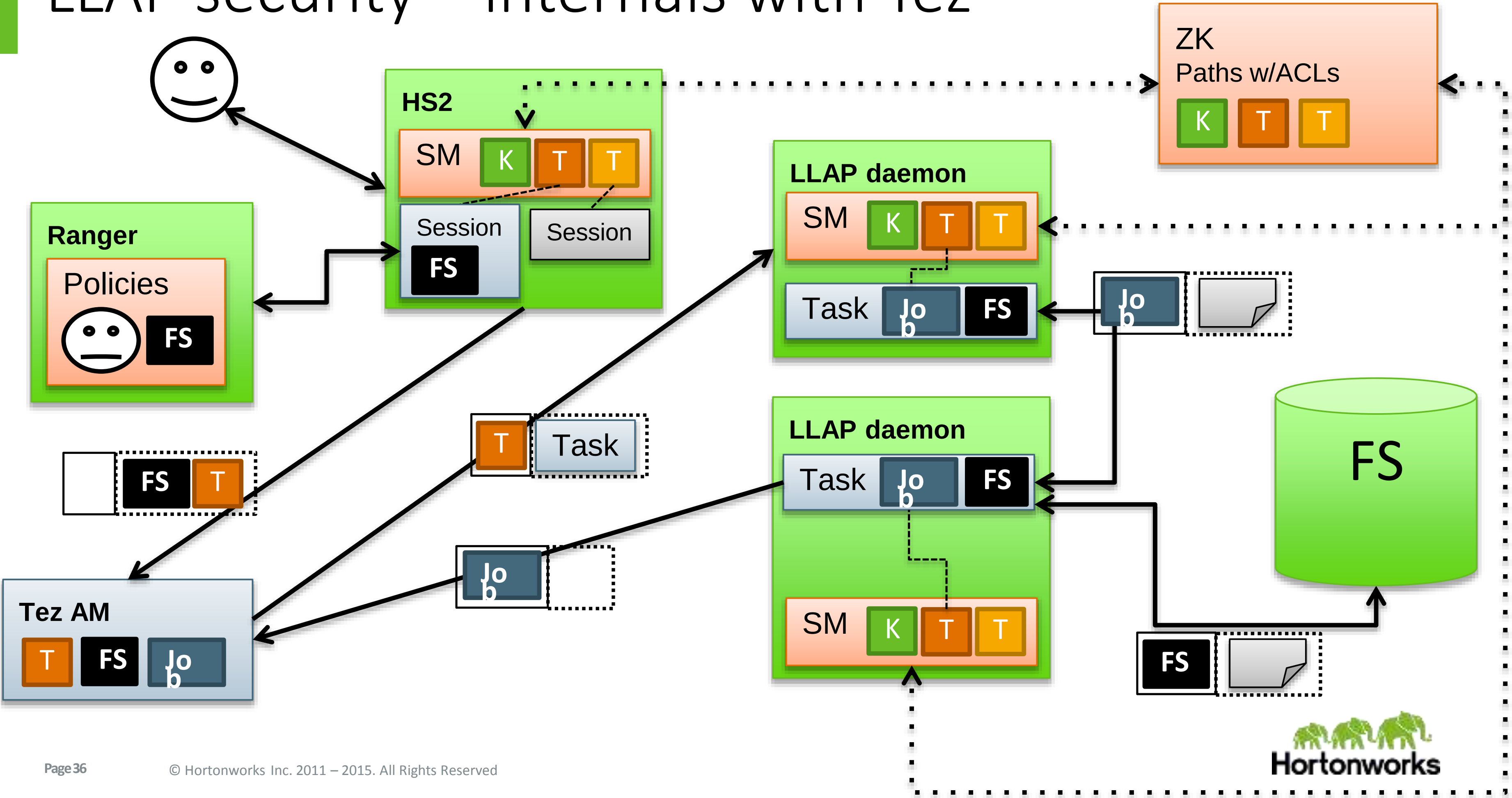
```
$ kinit billing/billing@EXAMPLE.COM

$ spark-shell --packages
com.hortonworks.spark:spark-llap-assembly_2.11:1.1.1-2.1
--repositories http://repo.hortonworks.com/content/groups/public
--conf spark.sql.hive.llap=true

scala> sql("select * from db_spark.t_customer").show
+--------------+------+
|          name|gender|
+--------------+------+
|  Barack Obama|     M|
| Michelle Obama|     F|
|Hillary Clinton|     F|
|  Donald Trump|     M|
+--------------+------+
```

Security

LLAP security – the EDW usage patterns

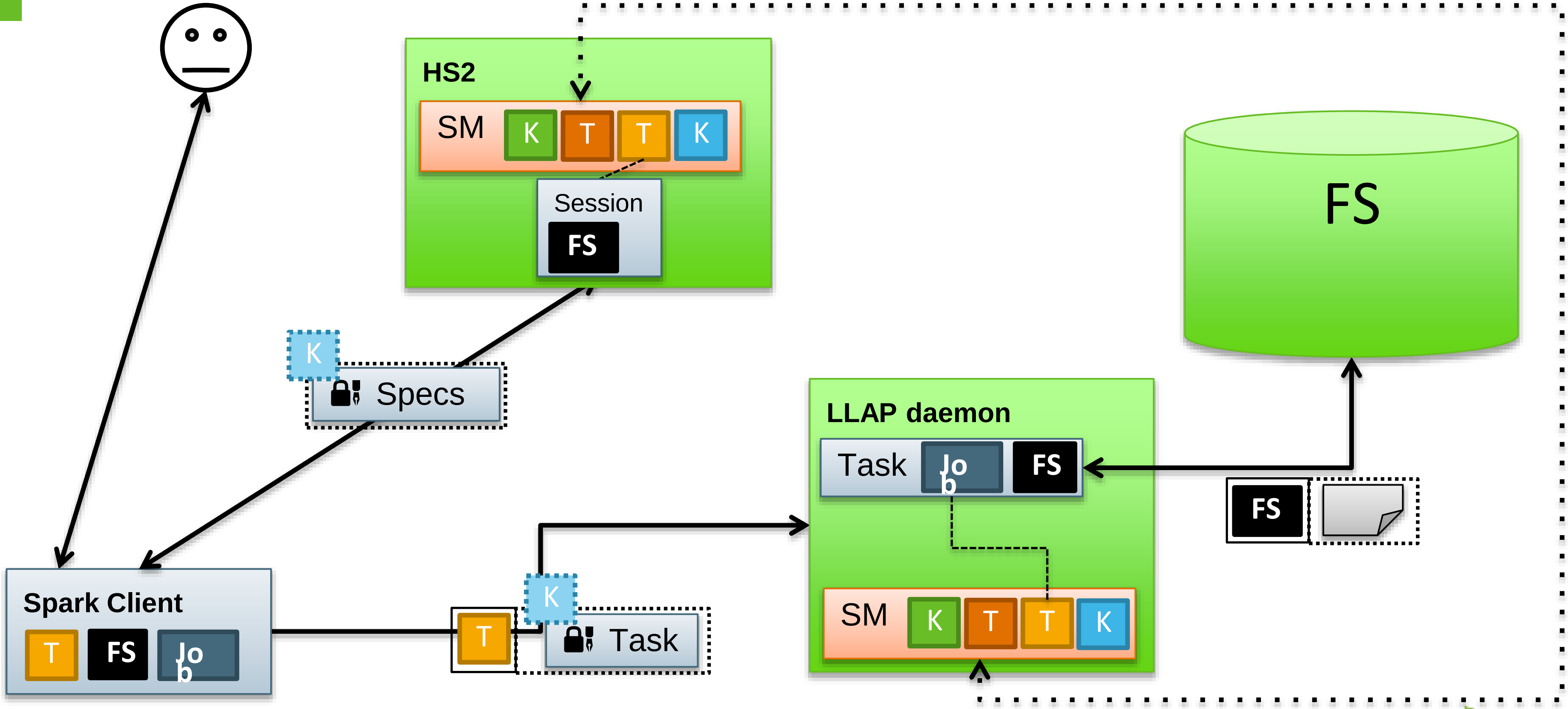
- Single, centrally administered LLAP cluster for all users
  - For now, separate ad hoc clusters cannot use Ambari
- Use Ranger
  - Hive SQL standard auth is an option; doAs is not recommended
  - Hive session AMs and LLAP run as hive superuser; managed by HS2
- HS2 serves as a central coordinator for security
  - Beeline and JDBC access; no CLI (requires client kinit)
  - HS2 checks permissions, enforces Ranger policies
  - Coordinates the usual Hadoop security dance (tokens for tasks, etc.)

LLAP security – internals with Tez



Hortonworks

LLAP security – external (Spark) additions



Hortonworks

Integration and tools

LLAP in Ambari (and HDP)

- Ambari 2.5 + HDP 2.6 = LLAP GA
  - The latest recommended update is HDP 2.6.1.0
  - Do not use Tech Preview versions; use GA
  - Separate version of Hive, "Hive Interactive"
  - Ambari 3.0 – no more separate versions
- Enable "Interactive Query" in Hive tab
  - A default configuration is chosen; more on that later
- No Ambari? Will cover this later

Hortonworks

LLAP on the cloud

- LLAP is in HDC (Hortonworks Data Cloud)
  - For quick, automated cluster deployments on AWS
- Also available on Azure HDInsight
- Details and links at the end!

Hortonworks

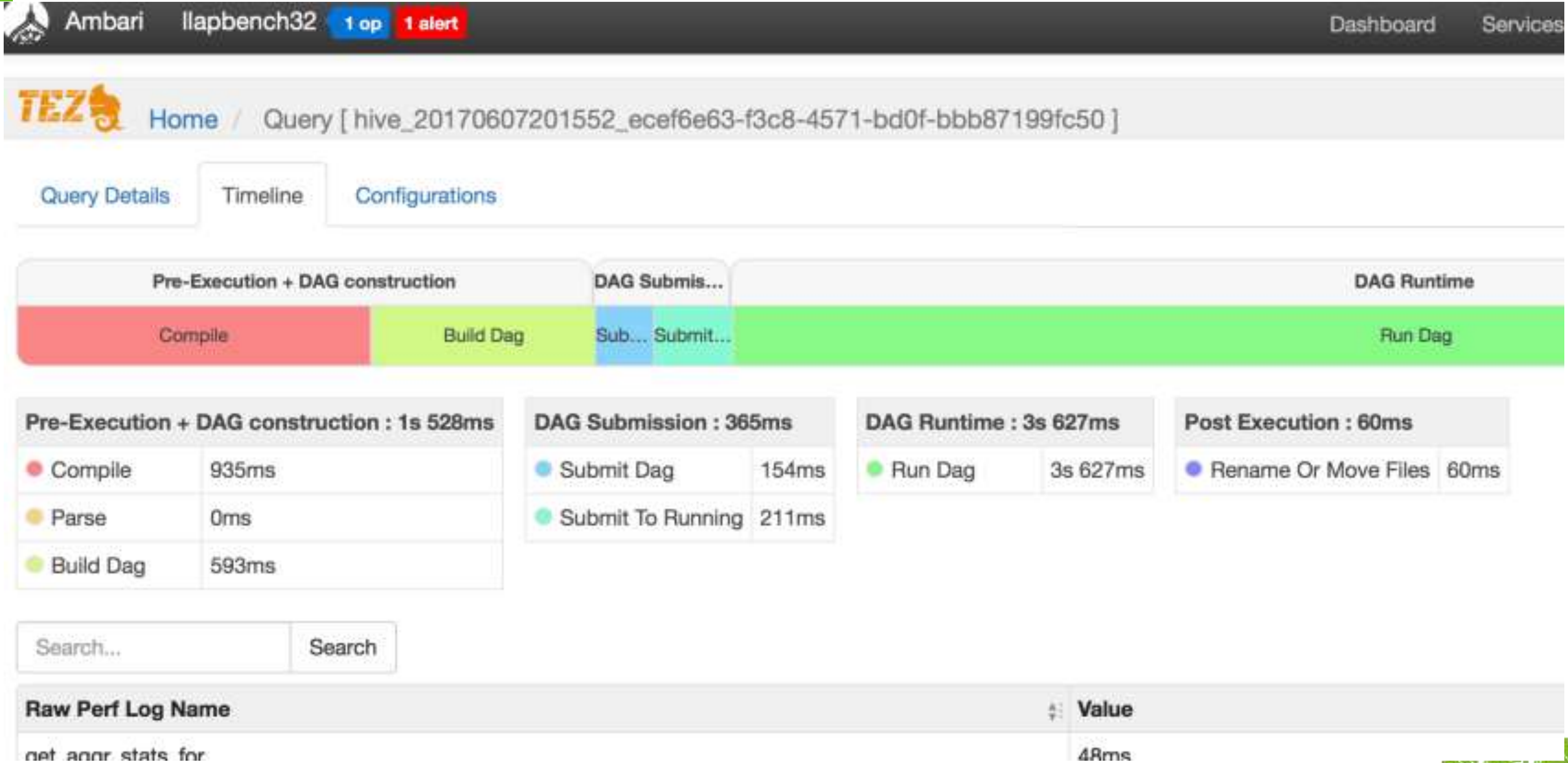
Tez UI – queries and LLAP integration

Query ID: Search... | User: Search... | DAG ID: Search... | Tables Read: Search... | Tables Written: Search... | App ID: Search... | Queue: Search... | Execution Mode: Search...

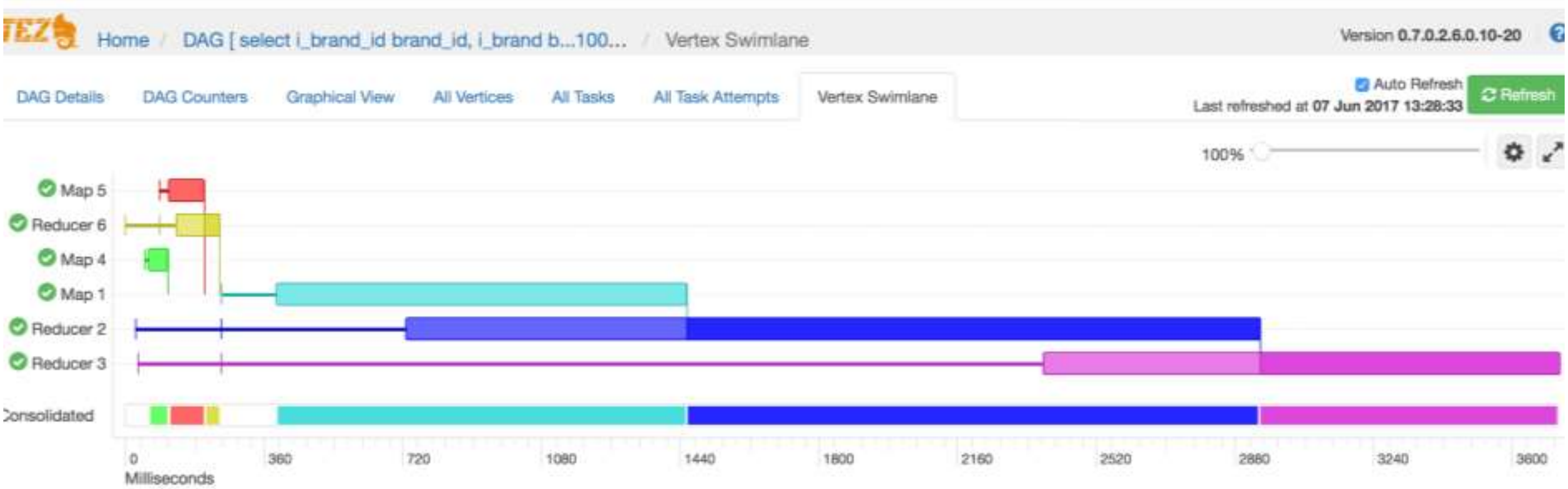
Status	Query	DAG ID	Tables Read	LLAP App ID	Duration	Applic
RUNNING	select i_item_id, s_...	Not Available!	tpcds_orc.store_sal...	application_1496771484228_0022	Not Available!	Not Av
RUNNING	select i_item_id, av...	dag_1496771484228_0020_380	tpcds_orc.promotio...	application_1496771484228_0022	Not Available!	applica
SUCCEEDED	select i_brand_id br...	dag_1496771484228_0019_374	tpcds_orc.date_dim...	application_1496771484228_0022	5s 185ms	applica
RUNNING	select s_store_nam...	dag_1496771484228_0021_374	tpcds_orc.store,tpc...	application_1496771484228_0022	Not Available!	applica
SUCCEEDED	select i_item_id, s_...	dag_1496771484228_0018_373	tpcds_orc.store_sal...	application_1496771484228_0022	9s 14ms	applica
SUCCEEDED	select i_item_id, av...	dag_1496771484228_0020_379	tpcds_orc.promotio...	application_1496771484228_0022	7s 716ms	applica
SUCCEEDED	select i_brand_id br...	dag_1496771484228_0021_373	tpcds_orc.date_dim...	application_1496771484228_0022	6s 668ms	applica
SUCCEEDED	select s_store_nam...	dag_1496771484228_0018_372	tpcds_orc.store,tpc...	application_1496771484228_0022	7s 792ms	applica
SUCCEEDED	select i_item_id, s_...	dag_1496771484228_0019_373	tpcds_orc.store_sal...	application_1496771484228_0022	9s 169ms	applica
SUCCEEDED	select i_item_id, av...	dag_1496771484228_0020_378	tpcds_orc.promotio...	application_1496771484228_0022	7s 322ms	applica

Hortonworks

Tez UI – query swimlane

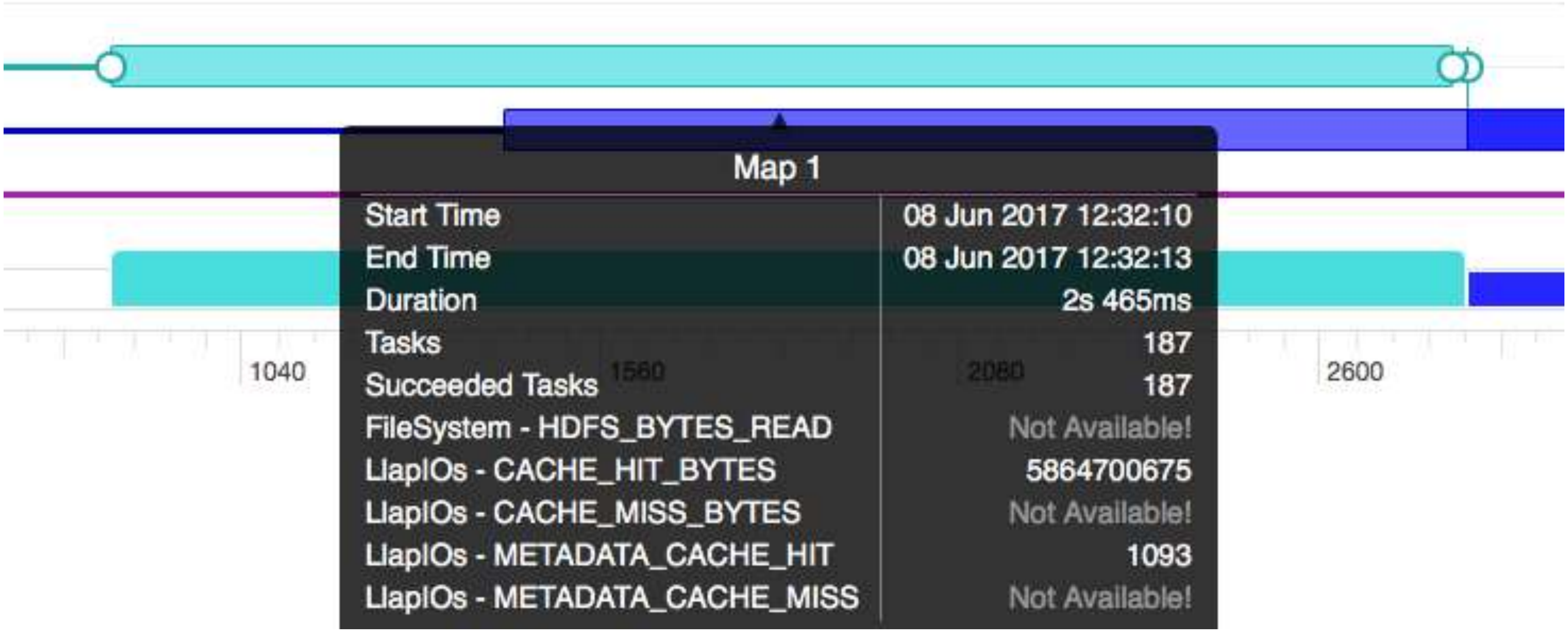


Tez UI – DAG swimlane



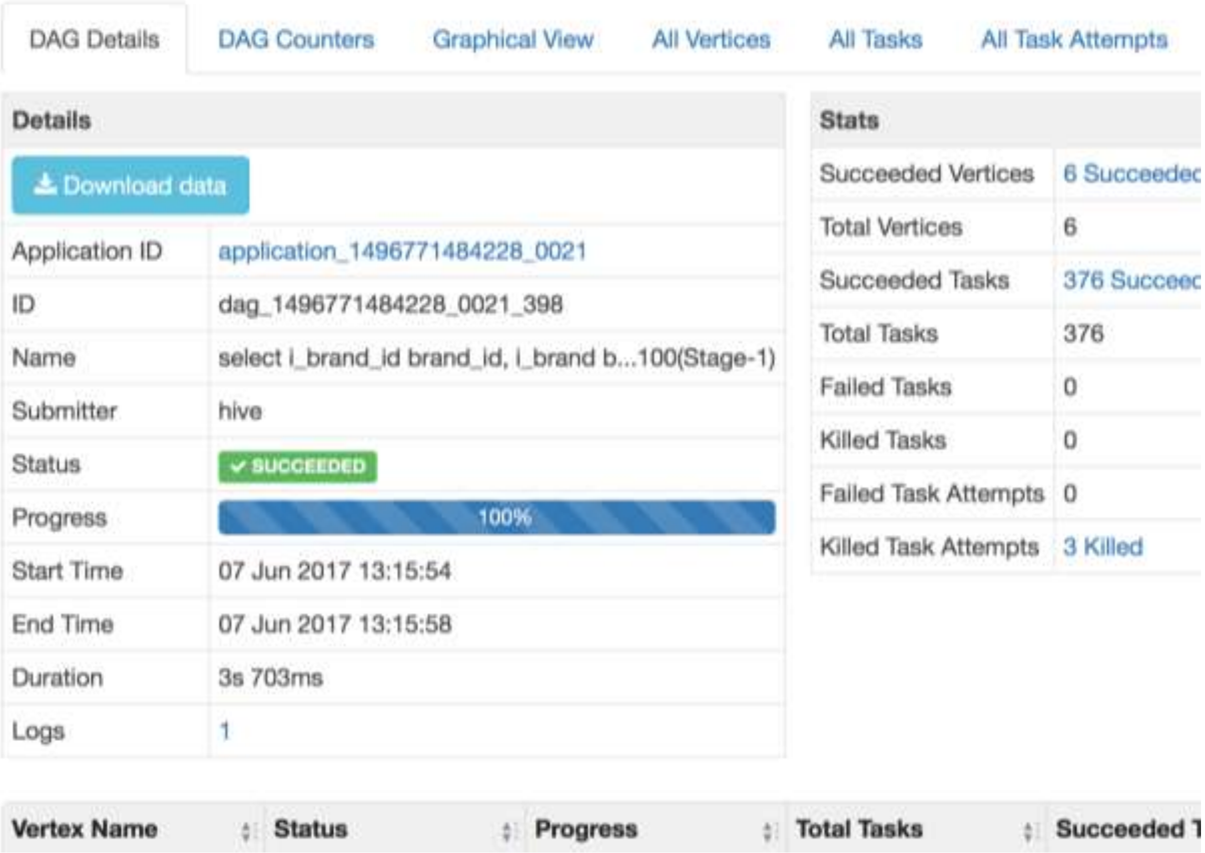
Hortonworks

Tez UI – LLAP counters



Hortonworks

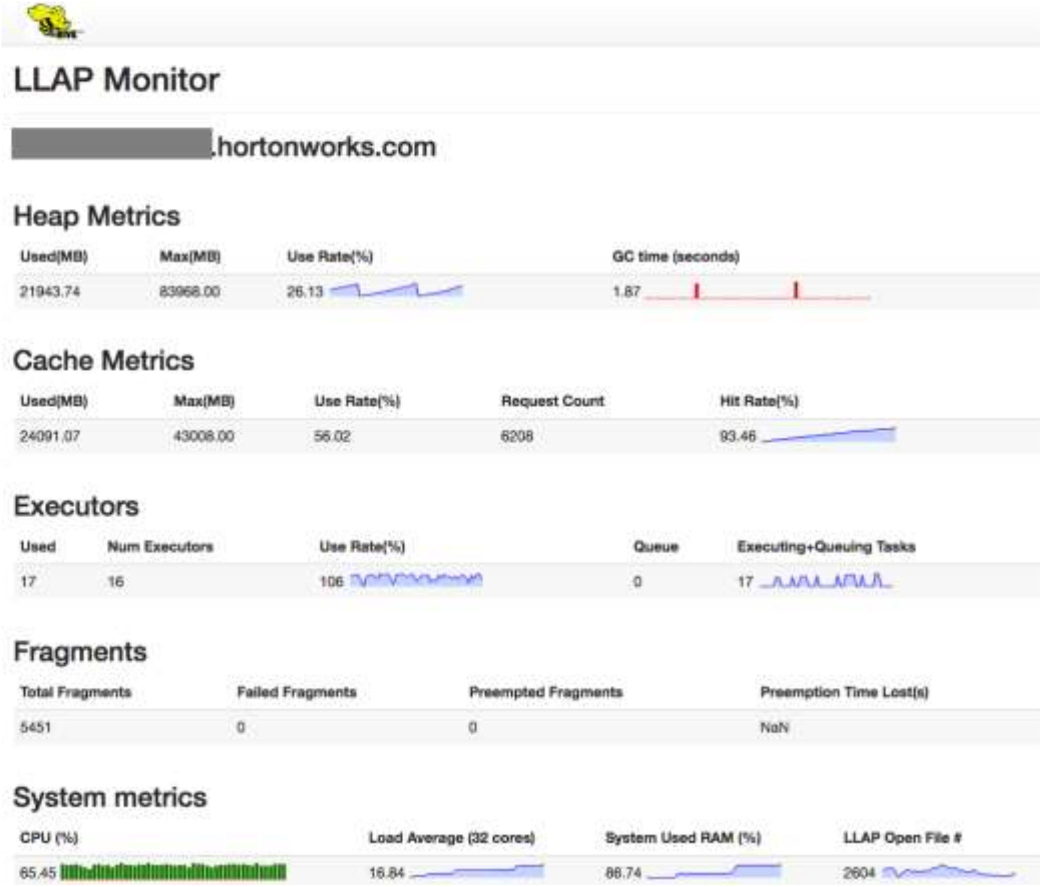
Tez UI – query information and debug data



Hortonworks

Monitoring

- LLAP exposes a UI for monitoring
- Also has jmx endpoint with much more data, logs and jstack endpoints as usual
- Aggregate monitoring UI is work in progress



Hortonworks

Debugging

- JMX view (:15002/jmx) contains many detailed metrics
  - E.g. "ExecutorsState" shows the running tasks and their state
- Log lines for most operations are annotated with task attempt #
- By default, stores separate log files per query
  - Logs can be downloaded (`yarn logs ...`) even for a running LLAP app
  - Log file contains the statements annotated for the query
  - File name contains session application ID and DAG# - "dag_TTTT_MMM_N"
    - e.g. `dag_`**`1490656001509_4602`**`_1` for Tez AM `application_`**`1490656001509_4602`**

Running LLAP

Making the best use of LLAP - summary

- Java 8, G1GC; some kernel configs, esp. for TCP connections
- Ambari comes with reasonable Hive perf configuration
  - May still need tweaking for specific workload; *more so w/o Ambari*
- SSD and text cache require "advanced config" until Ambari 3.0
- LLAP cluster sizing in a nutshell
  - AM per query (+ 1-2), AMs on all nodes; 2Gb RAM per AM, rest to LLAP
  - (Executor+IO thread) per core, 3-4Gb RAM per executor, rest to cache
- Without Ambari, see `hive --service llap` command + Slider

Hortonworks

Making the best use of LLAP – OS and Java

- Java 8 strongly recommended
  - G1 GC recommended (e.g. `--args " -XX:+UseG1GC -XX:+ResizeTLAB -XX:-ResizePLAB"`)
- Kernel settings

```
sysctl -w net.core.somaxconn=16384;
echo "never" > /sys/kernel/mm/transparent_hugepage/enabled
echo "never" > /sys/kernel/mm/transparent_hugepage/defrag
/etc/init.d/nscd restart
```

Making the best use of LLAP – cluster sizing

1. Pick the number of parallel queries (not just sessions)
   - AM per query + some constant slack
2. Pick executors per node, and io.threadpool count (# of cores per node)
3. Determine total memory size for LLAP - subtract the AM(s) from each NM (YARN memory per node)
   - 1 AM = 2 Gb; best to spread the AMs across each node
4. Determine the Xmx for LLAP; one executor (core) == 3-4Gb
5. Determine cache size - from the total, take out Xmx + ~3Gb (if lower, 20%)
   - ~3Gb for Java overhead, shared overhead
6. Tweak based on workload?

Hortonworks

Cluster sizing example

- **6 node cluster – 100Gb RAM each, 24 cores, NM Size = 96Gb, 25 concurrent queries**
- 28 AMs; total AM memory = 28*2 = 56
- (96*6 – 28*2)/6 = 86.66 => "Memory per daemon" = 85Gb
- "LLAP heap size" (Xmx) = 3Gb*24 = 72Gb
- "Cache size" = 85Gb – 72Gb – 3Gb ~= 10Gb

Hortonworks

Making the best use of LLAP – Hive settings

- ***Ambari has a lot of this configured by default***
- The basics - use ORC; enable PPD, configure mapjoin size, etc.
- Enable vectorization and consider new vectorization features
  - `hive.vectorized.execution.*.enabled` – see the configuration file documentation
- Enable LLAP split locality - `hive.llap.client.consistent.splits`
  - `hive.llap.task.scheduler.locality.delay` to tweak strict/relaxed locality
- Consider disabling CBO for interactive queries (test your queries!)
- Use parallel compilation in HS2 - `hive.driver.parallel.compilation`
- Shuffle improvement - `tez.am.am-rm.heartbeat.interval-ms.max=5000`

Making the best use of LLAP – new cache stuff

- Text cache (not turned on in Ambari until 3.0!)
  - `hive.llap.io.encode.enabled`, `hive.vectorized.use.(row|vector).serde.deserialize`
- SSD cache (not turned on in Ambari until 3.0!)
  - `hive.llap.io.allocator.mmap`; `hive.llap.io.allocator.mmap.`**`path`**
  - `hive.llap.io.memory.size` controls the total cache size (on disk)
  - You have to disable YARN memory check for now until YARN 2.8
- Cache on cloud FS
  - `hive.orc.splits.allow.synthetic.fileid`, `hive.llap.cache.allow.synthetic.fileid`

Hortonworks

LLAP without Ambari

- Requires Hive 2.X (2.2-2.3 are coming soon); ZK, YARN, Slider
- **`hive --service llap`** generates a slider package
  - Run this as the correct user – slider paths are user-specific! kinit on secure cluster
  - Specify a name, # of instances; memory, cache size, etc.; see `--help`
- Generates `run.sh` to start the cluster (in `--output`) directory
  - Or use `--startImmediately` in newer versions
- Queries can be run from HS2 and CLI; basic configuration:
  - `hive.execution.mode=llap, hive.llap.execution.mode=all, hive.llap.io.enabled=true, hive.llap.daemon.service.hosts=@<cluster name>`

Summary

LLAP provides a

- unified, managed and secure cloud-ready EDW solution for BI and ETL workloads via a
- fast execution substrate harnessing Hive vectorized SQL engine and
- efficient in-memory caching layer for columnar and text data

Try Hive LLAP Today on-prem or in the Cloud

Hortonworks Data Platform 2.6
Powered by 100% open source Apache Hadoop
http://hortonworks.com/downloads/

Hortonworks Data Cloud
Easy HDP on Amazon Web Services
http://hortonworks.com/products/cloud/aws/

Microsoft Azure HDInsight
A cloud Spark and Hadoop service for your enterprise
http://azure.microsoft.com/en-us/services/hdinsight/

Questions?

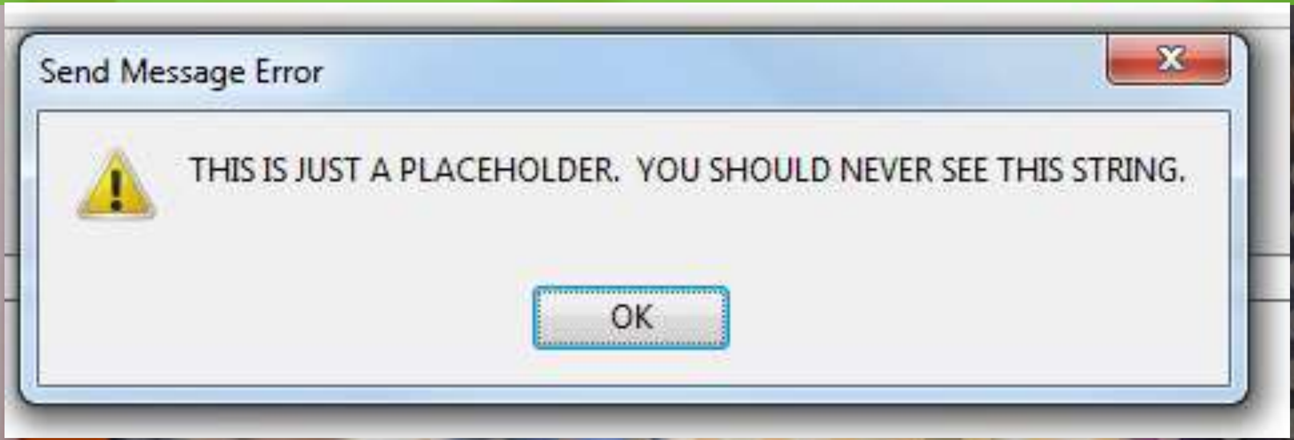
?

Interested? Stop by the Hortonworks booth to learn more

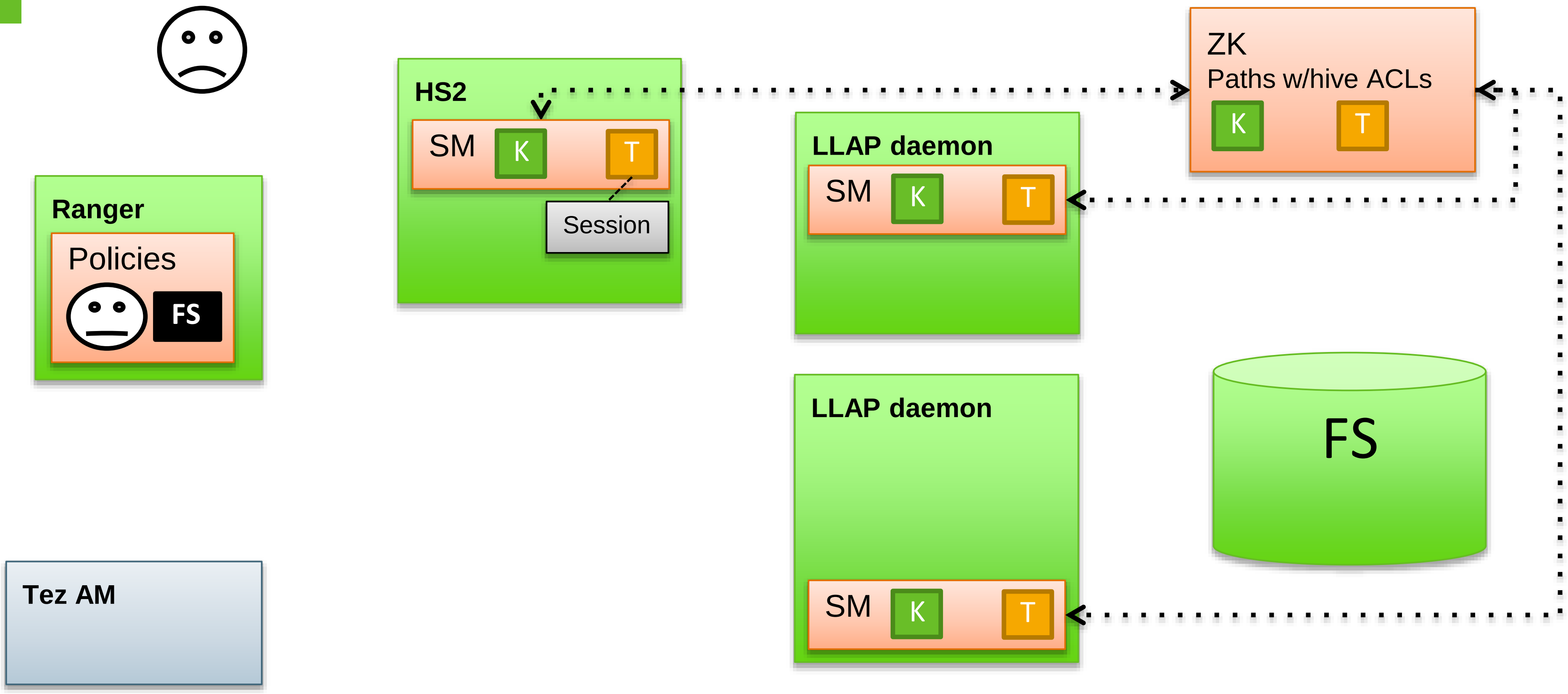
Hortonworks

Hortonworks

Backup slides

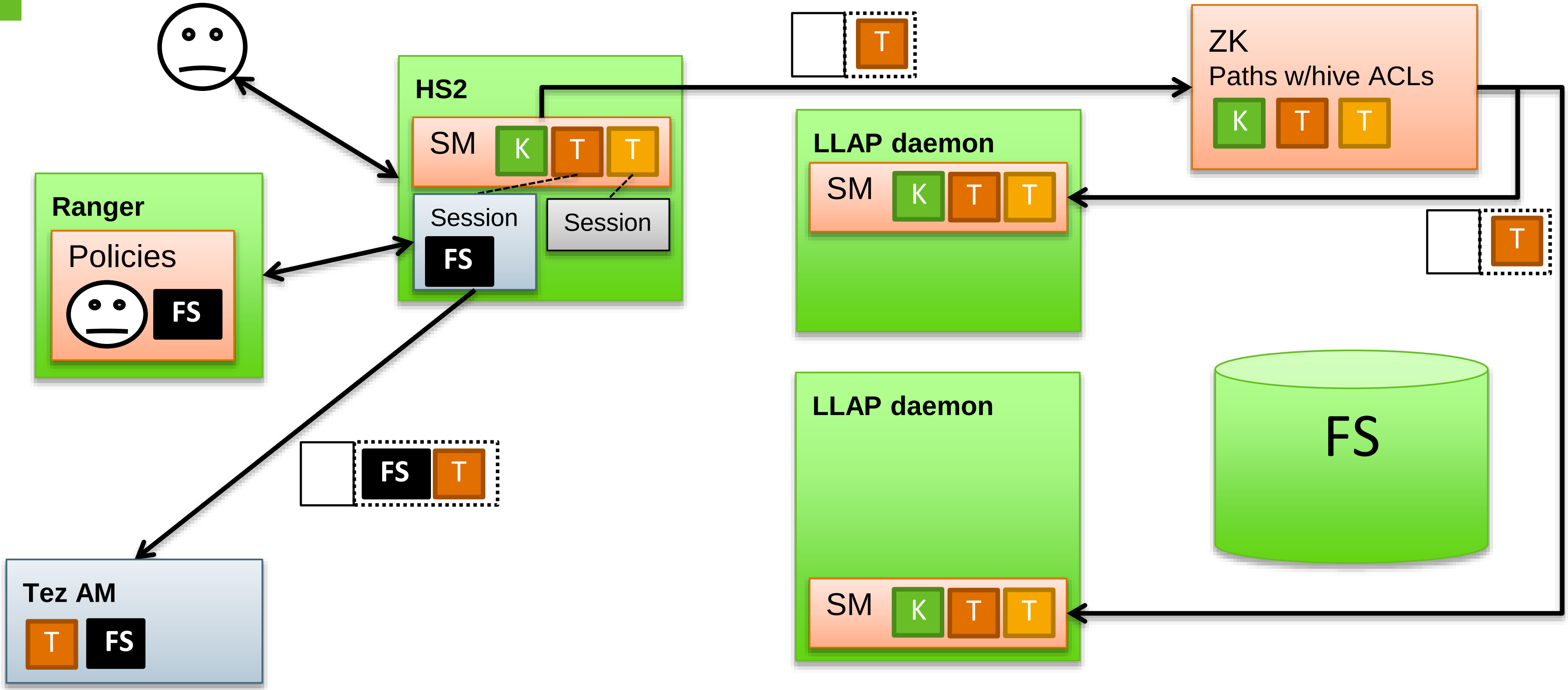


LLAP security – internals with Tez

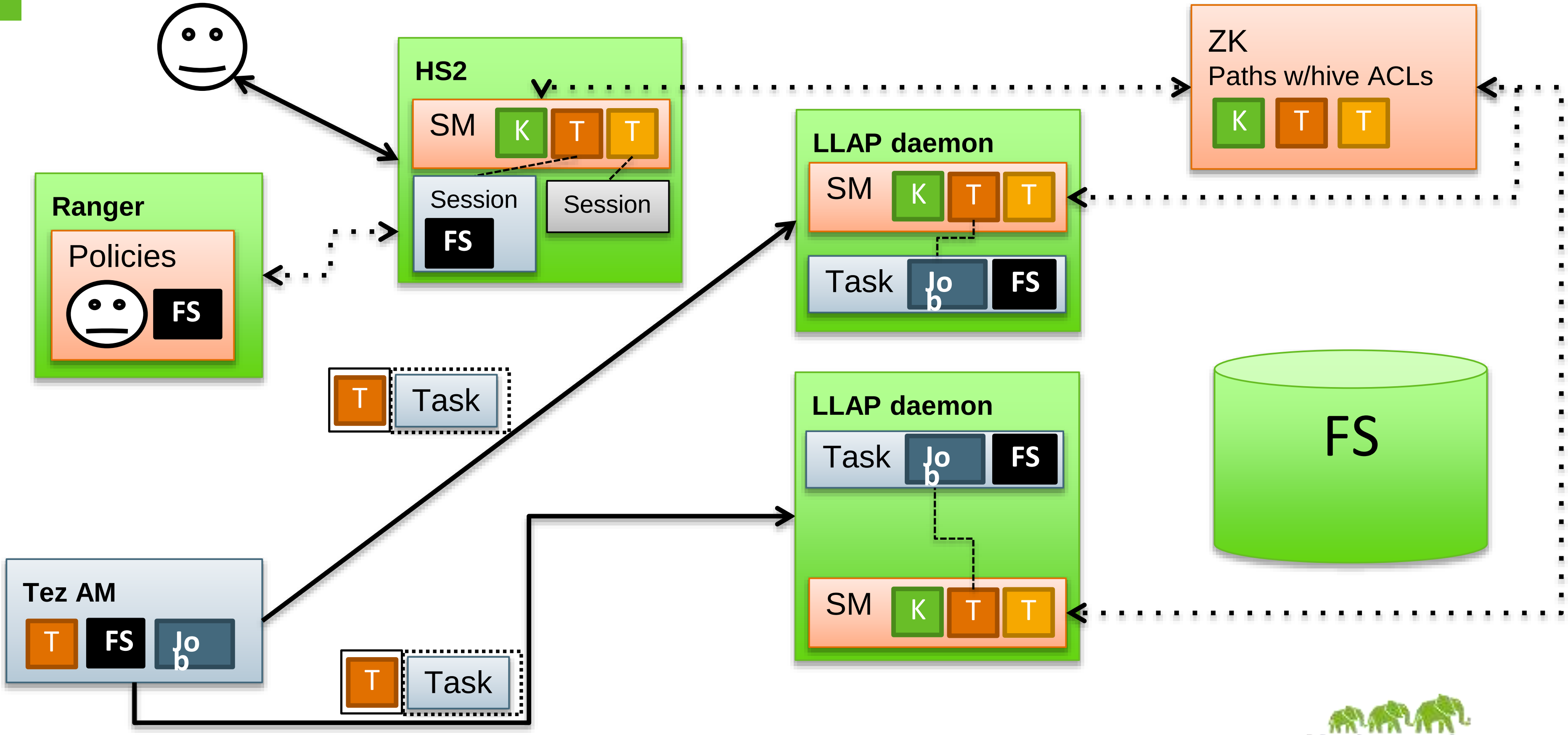


Hortonworks

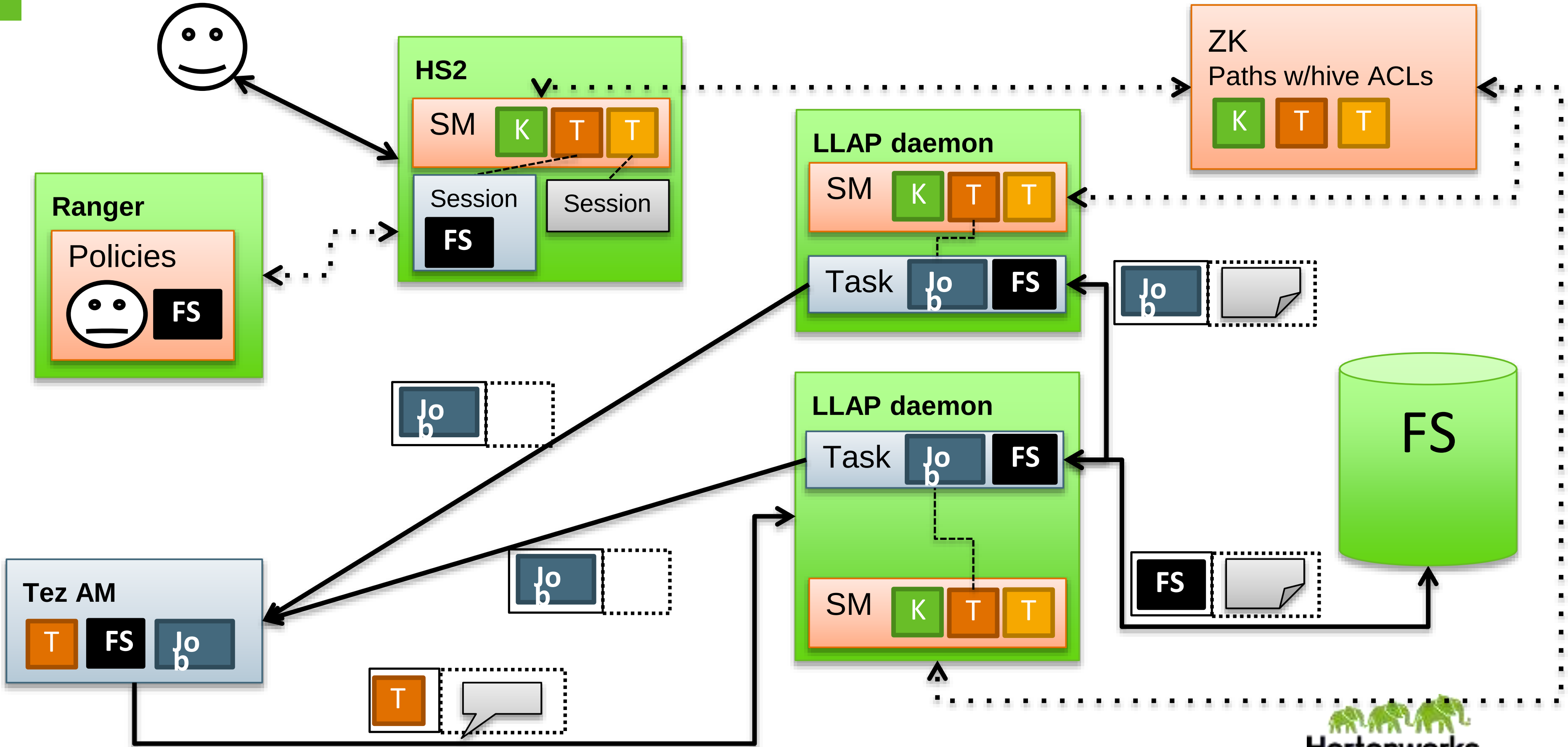
LLAP security – internals with Tez



LLAP security – internals with Tez



LLAP security – internals with Tez



Hortonworks

Future work

- Centralized workload management
- Better tool support for cluster management, SSD cache, etc.
- Improvements to Spark-LLAP integration
- Parquet cache support
- Memory management, cache improvements, faster cluster startup time, other performance features