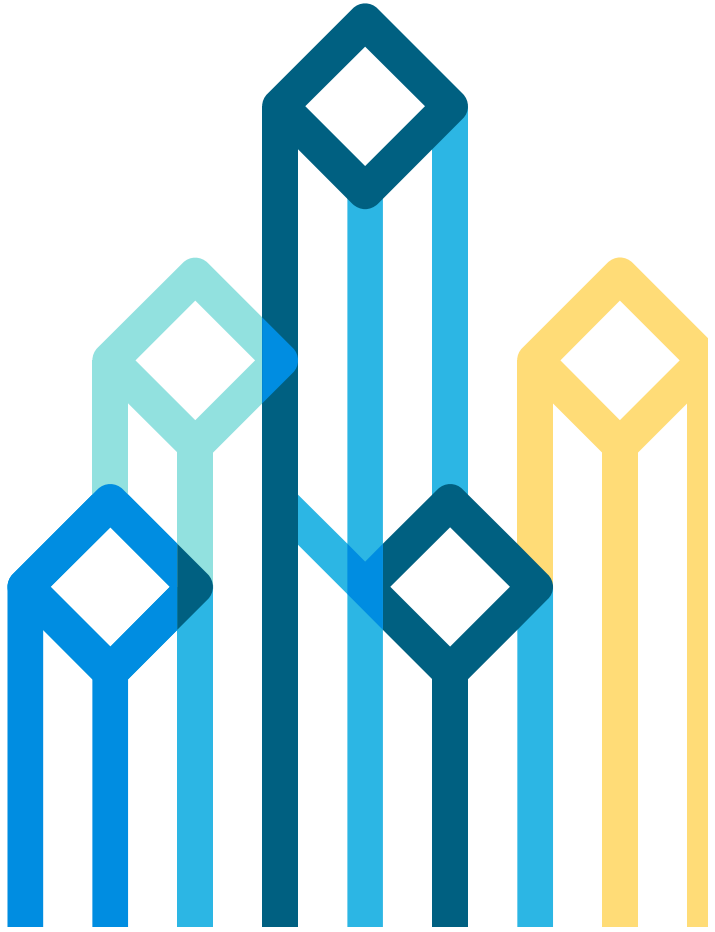
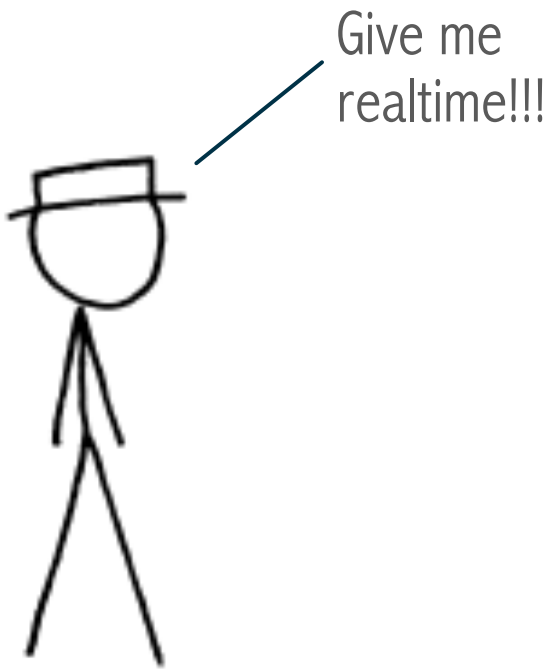
cloudera

Fast Analytics with Apache Kudu (incubating)



Ryan Bosshart//Systems Engineer

bosshart@cloudera.com

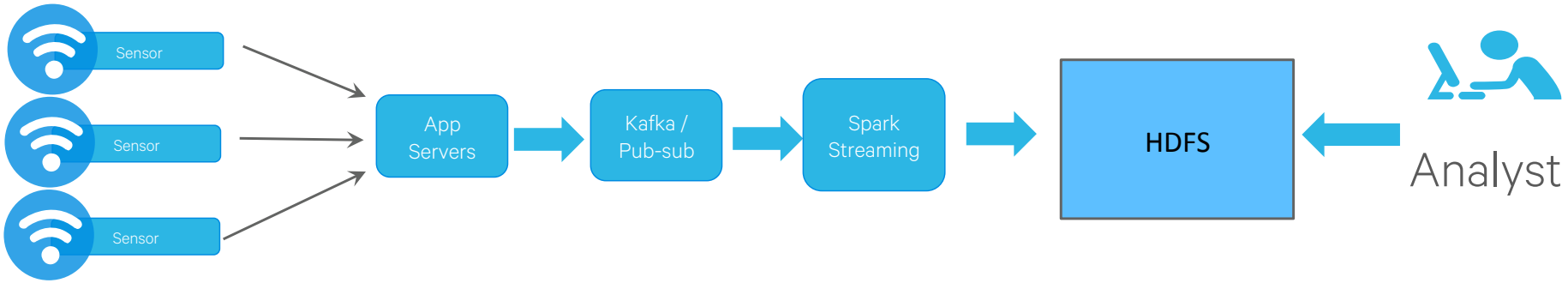


Business

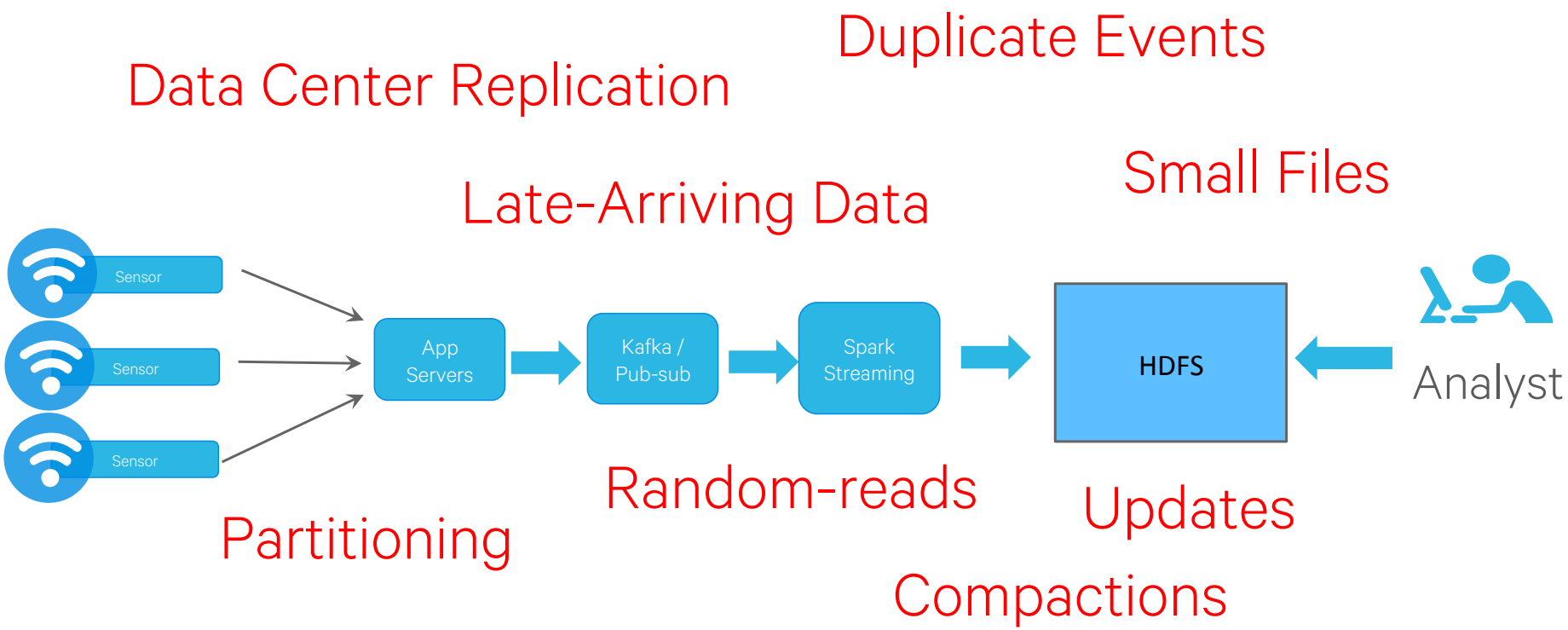
Hadoop Architect

cloudera

How would we build an IOT Analytics System Today?

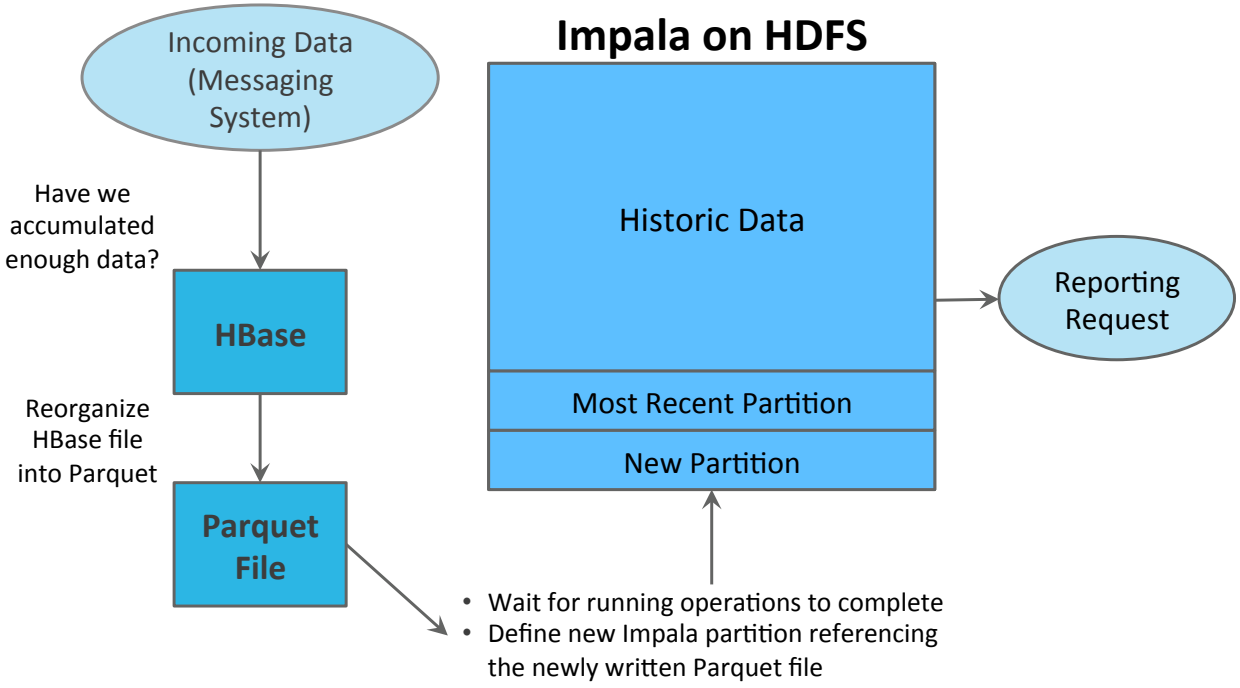


What Makes This Hard?



Real-Time Analytics in Hadoop Today

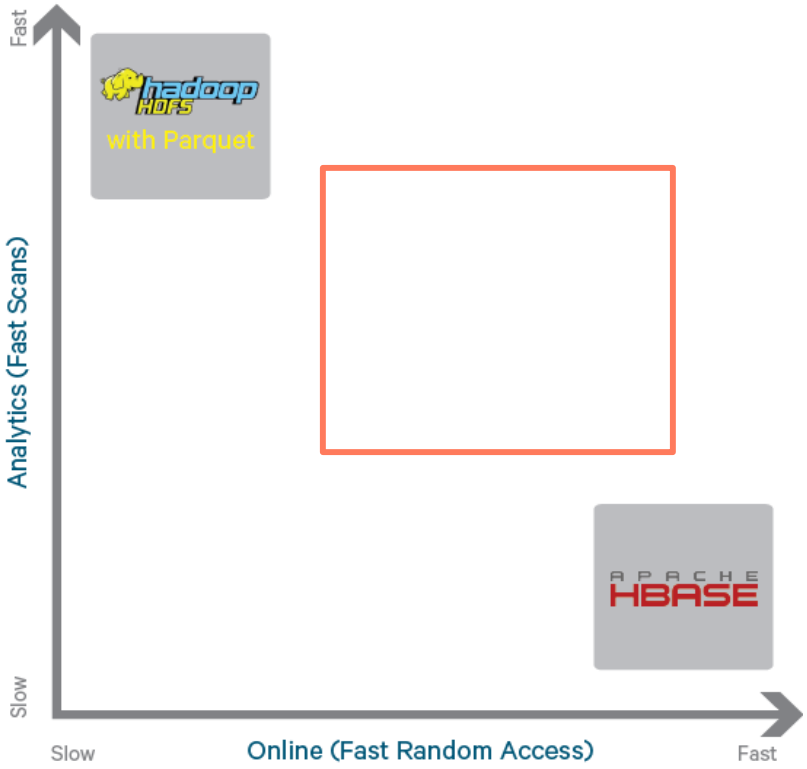
Realtime Analytics in the Real World = Storage Complexity



Considerations:

- How do I handle failure during this process?
- How often do I reorganize data streaming in into a format appropriate for reporting?
- When reporting, how do I see data that has not yet been reorganized?
- How do I ensure that important jobs aren't interrupted by maintenance?

Previous storage landscape of the Hadoop ecosystem



HDFS (GFS) excels at:

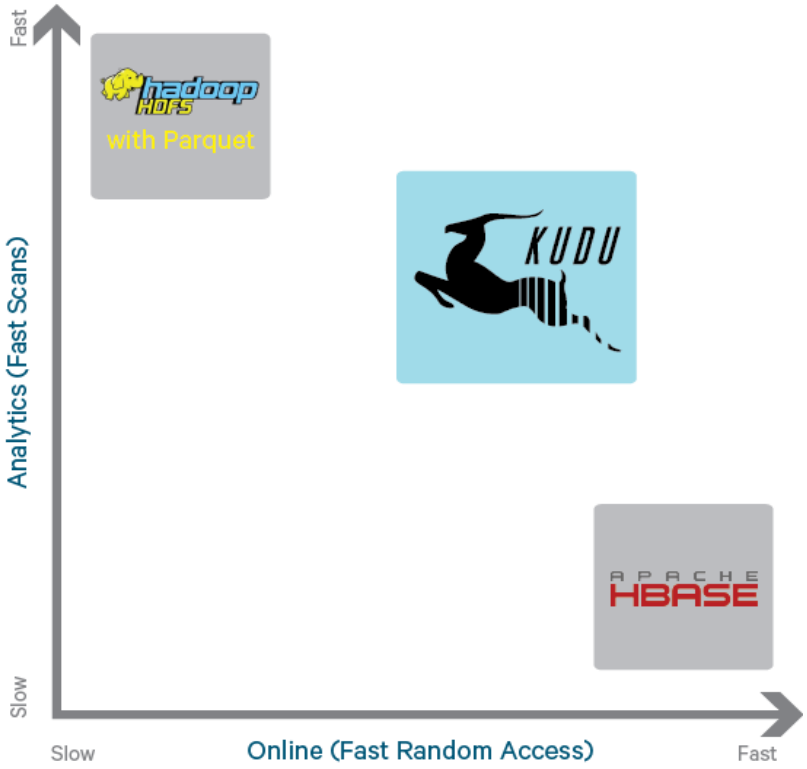
- Batch ingest only (eg hourly)
- Efficiently scanning large amounts of data (analytics)

HBase (BigTable) excels at:

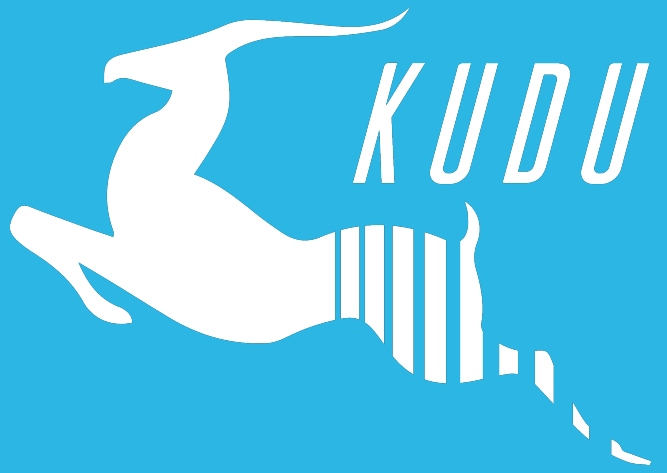
- Efficiently finding and writing individual rows
- Making data mutable

Gaps exist when these properties are needed *simultaneously*

Kudu design goals



- **High throughput** for big scans
  Goal: Within 2x of Parquet
- **Low-latency** for short accesses
  Goal: 1ms read/write on SSD
- **Database-like** semantics (initially single-row ACID)
- **Relational data model**
  - SQL queries are easy
  - "NoSQL" style scan/insert/update (Java/C++ client)



Kudu for Fast Analytics

Why Now

cloudera

Major Changes in Storage Landscape

[2007ish]

All spinning disks
Limited RAM

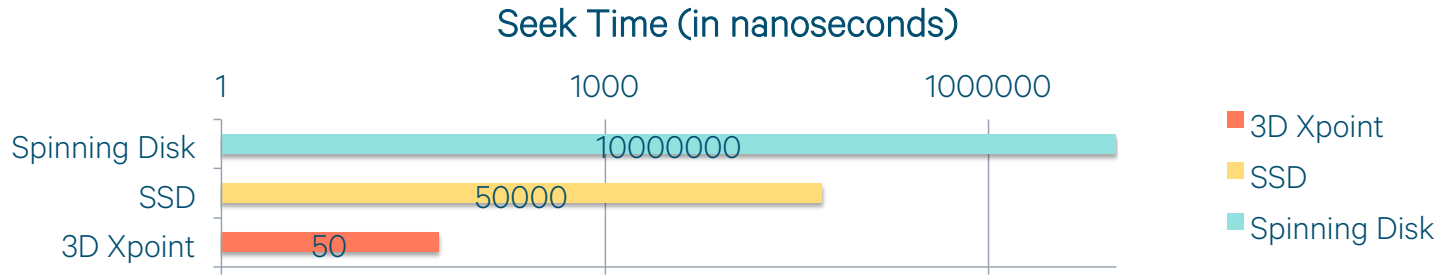
[2013ish]

SSD/NAND cost effective
RAM much cheaper

[2017+]

Intel 3Dxpoint
256GB, 512GB RAM common

.



cloudera

The next bottleneck is CPU

IOT, Real-time, and Reporting Use-Cases

There are more use cases requiring a simultaneous combination of sequential and random reads and writes

- Machine data analytics
  - Example: IOT, Connected Cars, Network threat detection
  - Workload: Inserts, scans, lookups
- Time series
  - Examples: Streaming market data, fraud detection / prevention, risk monitoring
  - Workload: Insert, updates, scans, lookups
- Online reporting
  - Example: Operational data store (ODS)
  - Workload: Inserts, updates, scans, lookups

IOT Use-Cases

- Analytical
  - R&D wants to know part performance over time.
  - Train predictive models on machine or part failure.
- Real-time
  - Machine Service – e.g. grab an up-to-date “diagnosis bundle” before or during service.
  - Rolled out a software update – need to find out performance ASAP!

IOT Use-Cases

- Analytical
  - R&D wants to know optimal part per
  - Tra achine or part failure.
- Real-time
  - Machine Service – e.g. grab an up-to-date "dia during ser
  - Rol – need to find out performance ASAP!

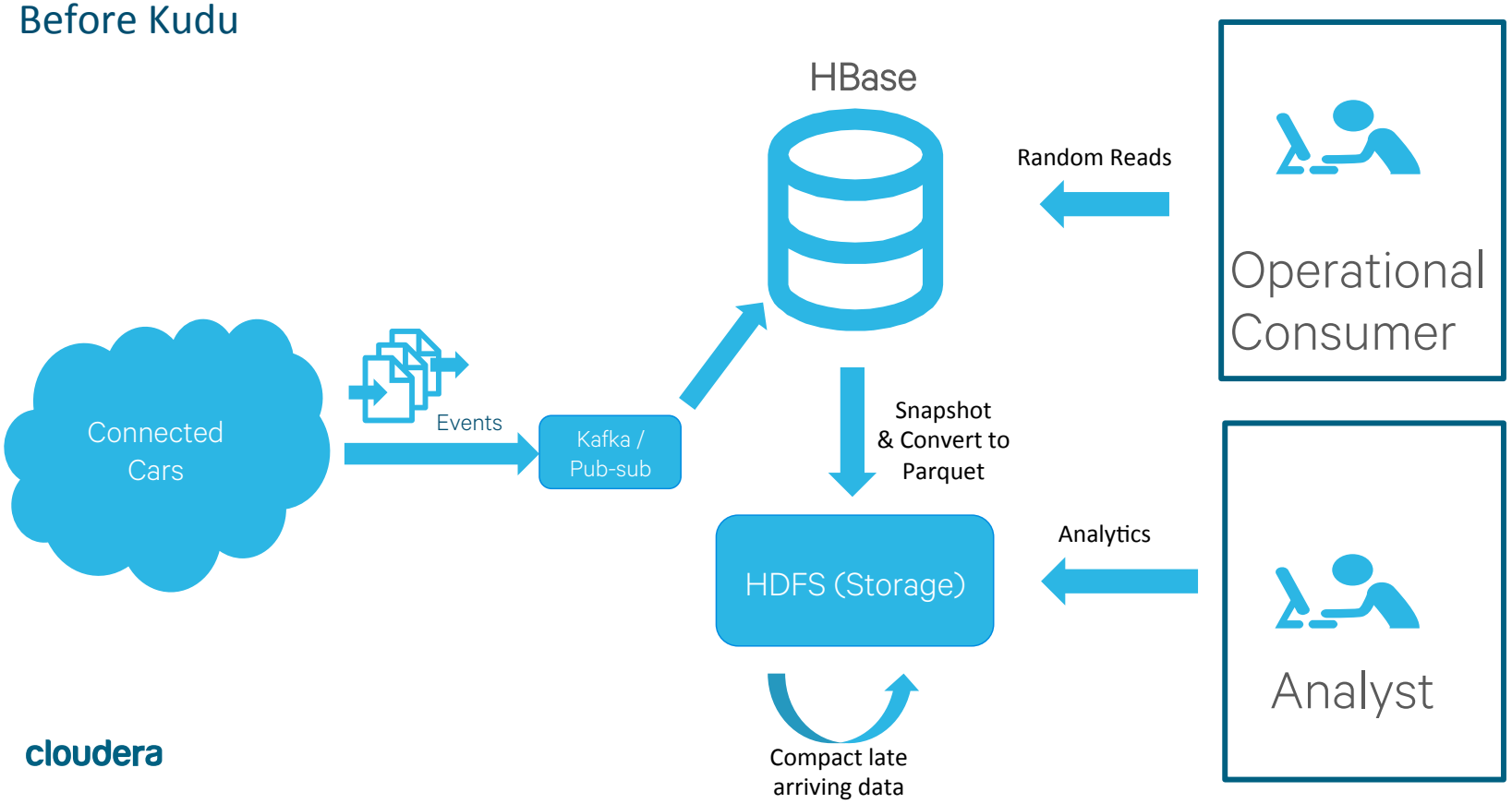
fast, efficient scans
= HDFS

fast inserts/lookups
= HBase

cloudera

Hybrid big data analytics pipeline

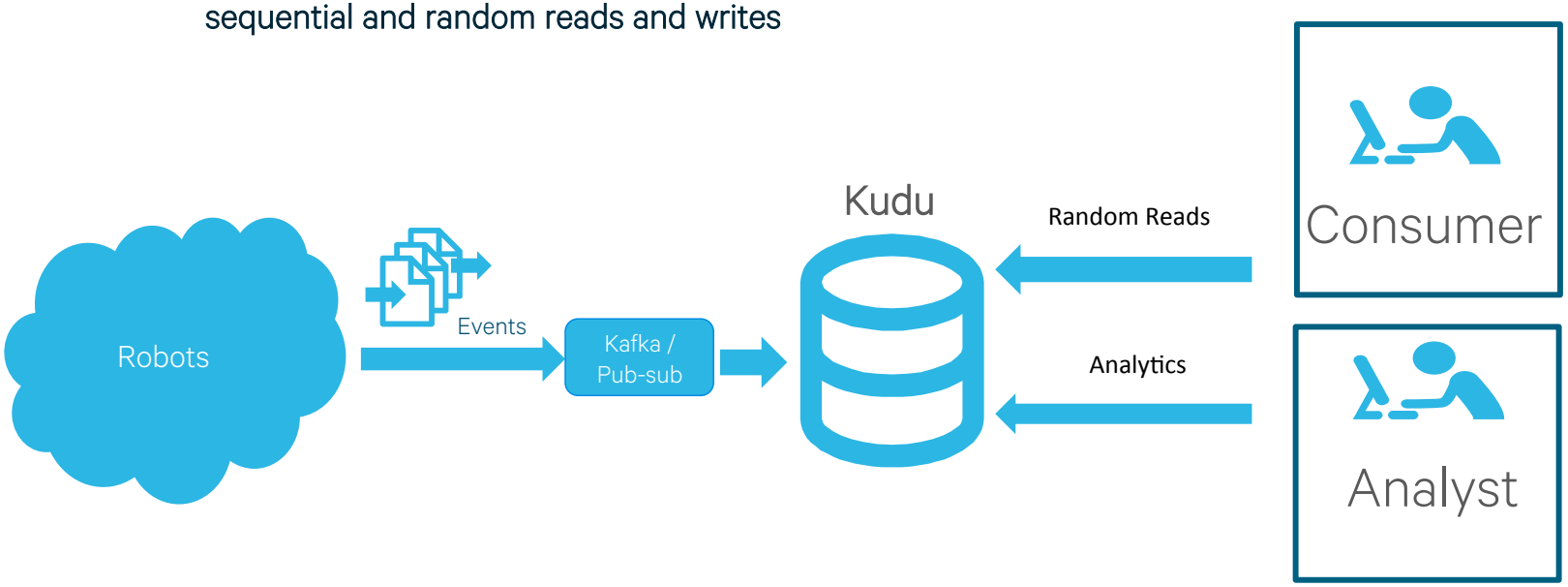
Before Kudu

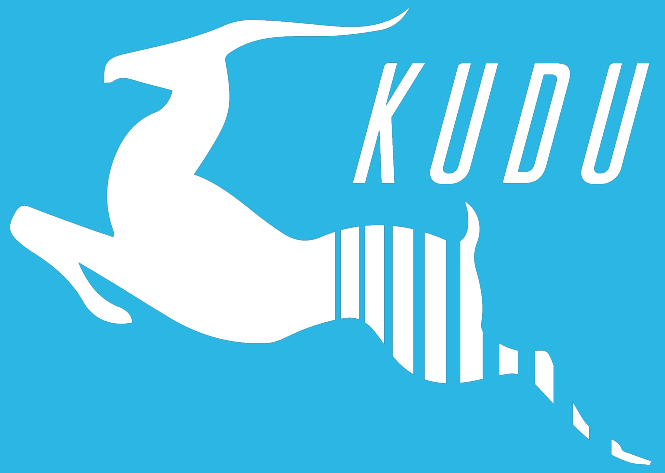


cloudera

Kudu-Based Analytics Pipeline

Kudu supports simultaneous combination of sequential and random reads and writes





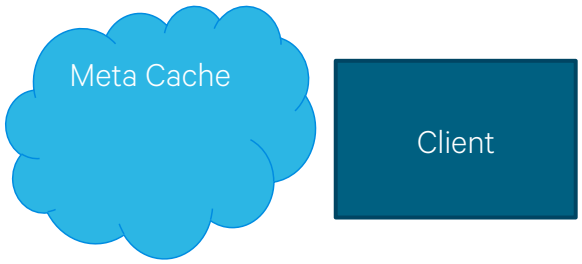
How it works

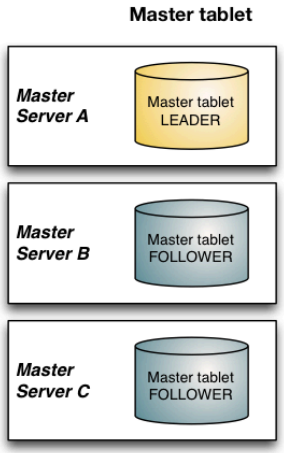
Replication and fault tolerance

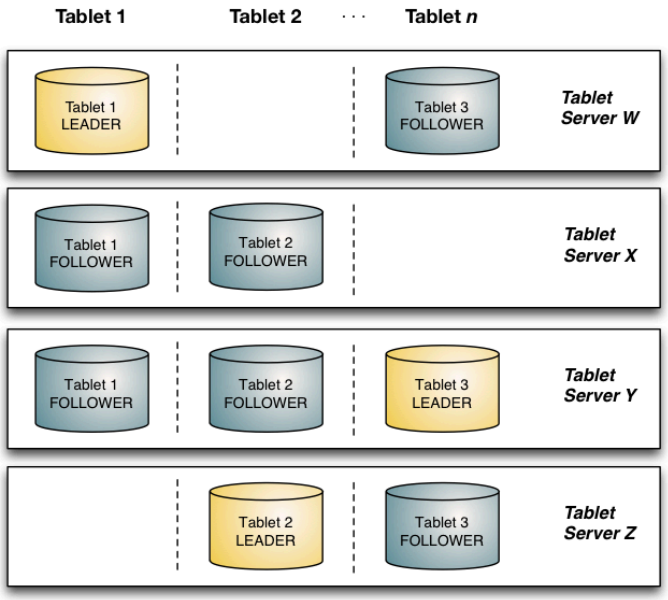
Kudu Basic Design

- Basic Construct: Tables
  - Tables broken down into Tablets (roughly equivalent to regions or partitions)
- Typed storage
- Maintains consistency via:
  - Multi-Version Concurrency Control (MVCC)
  - Raft Consensus[1] to replicate *operations*
- Architecture supports geographically disparate, active/active systems
  - Not in the initial implementation

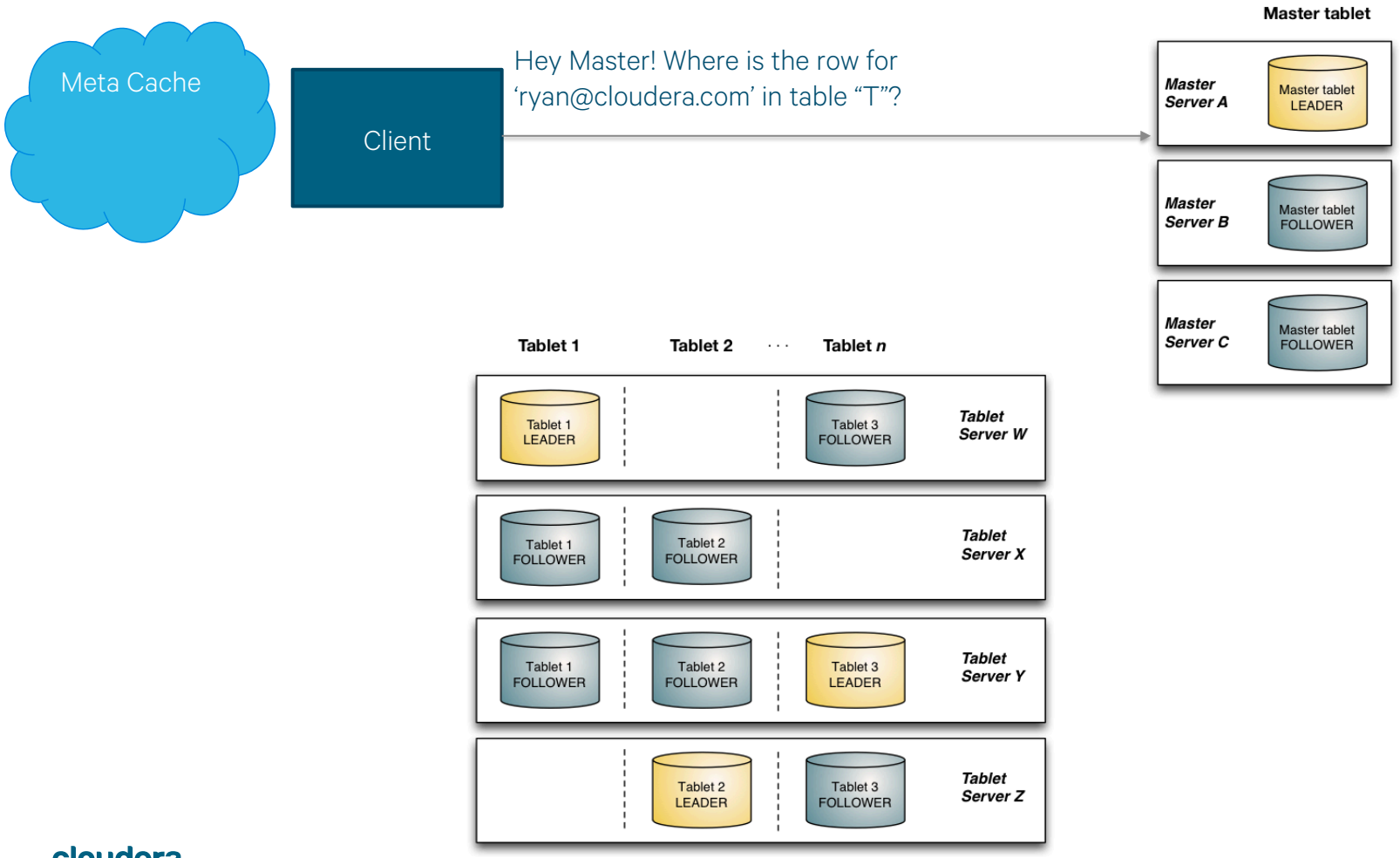
[1] http://thesecretlivesofdata.com/raft/

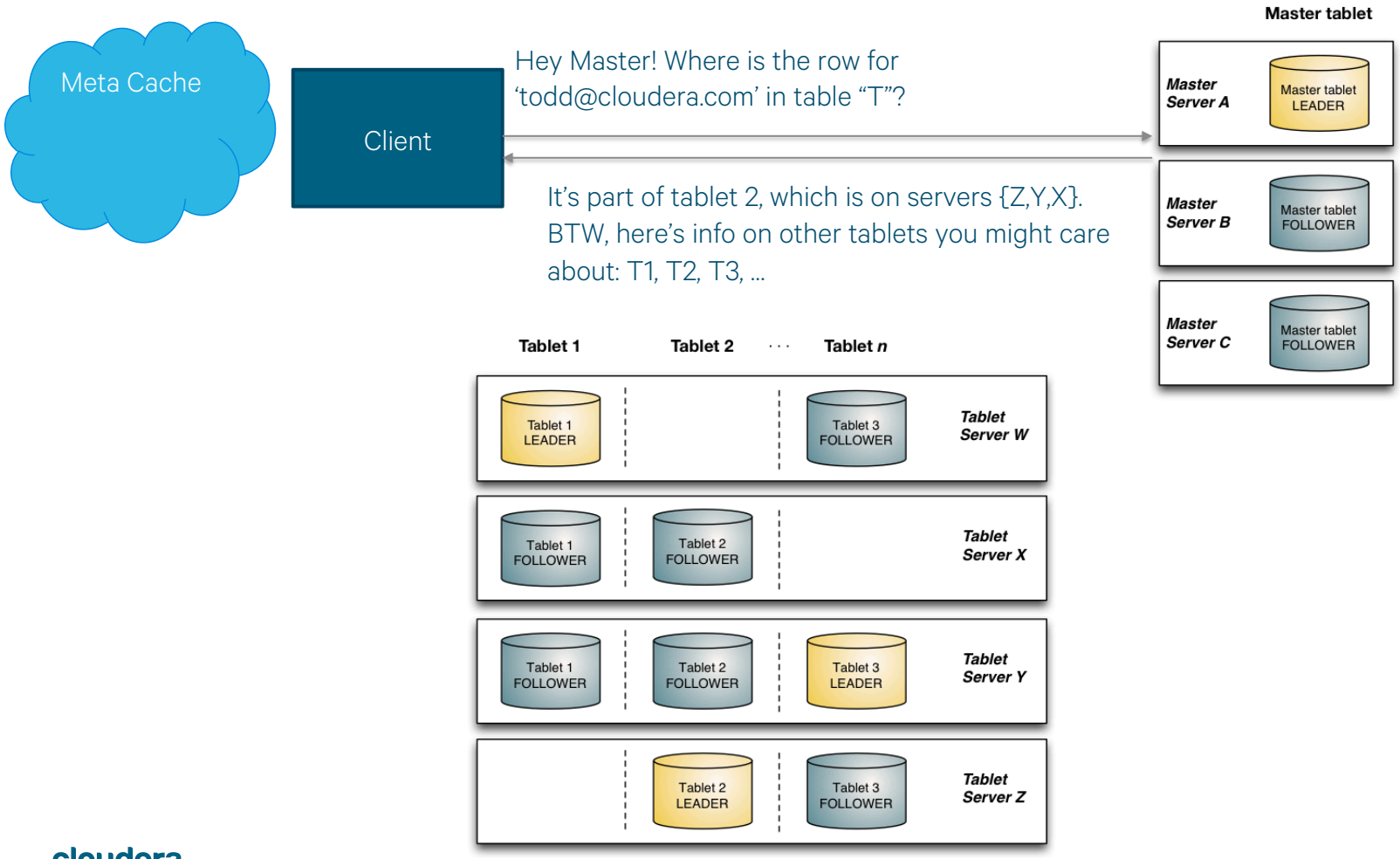

Meta Cache
Client

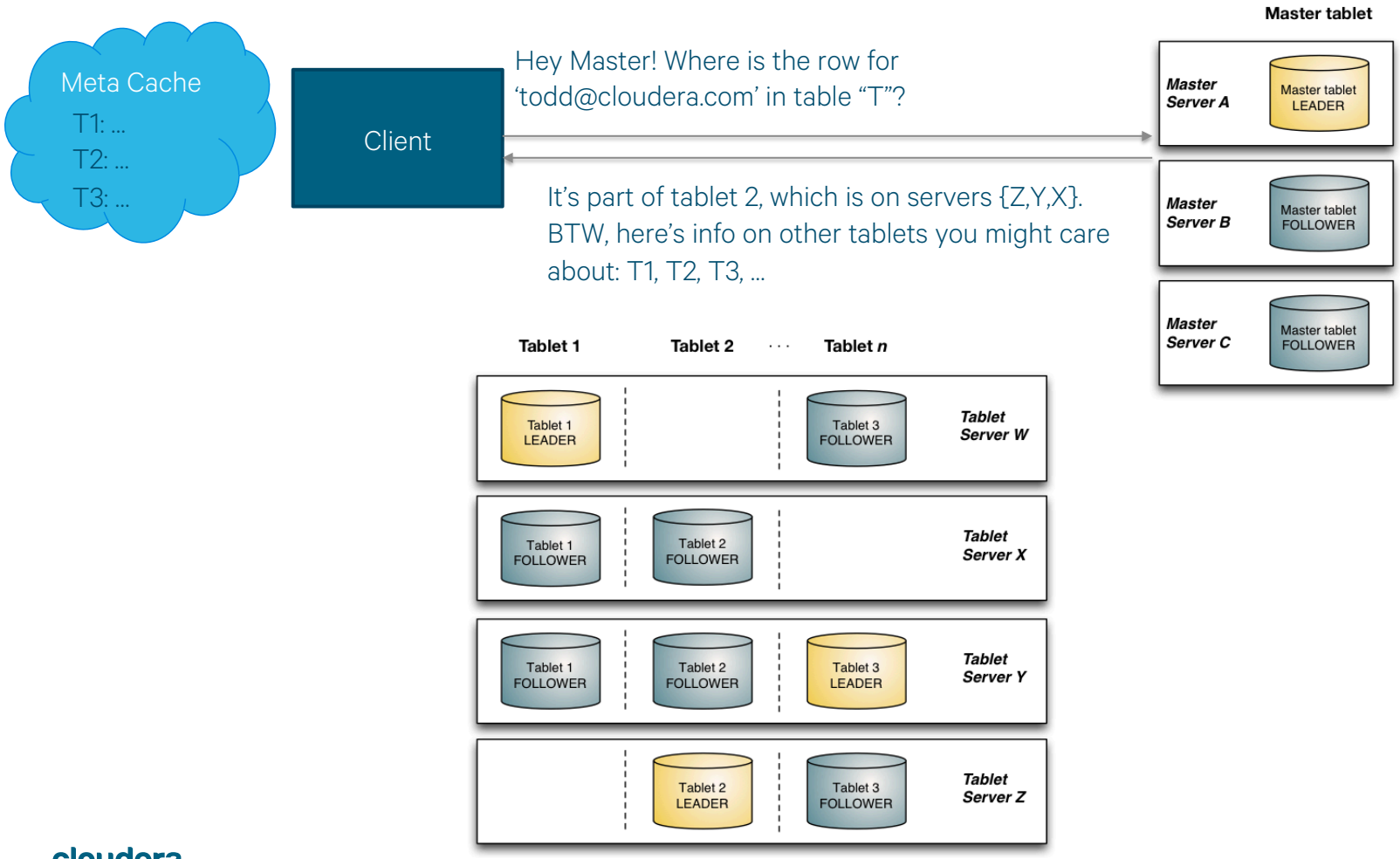

Master tablet
Master Server A
Master tablet LEADER
Master Server B
Master tablet FOLLOWER
Master Server C
Master tablet FOLLOWER

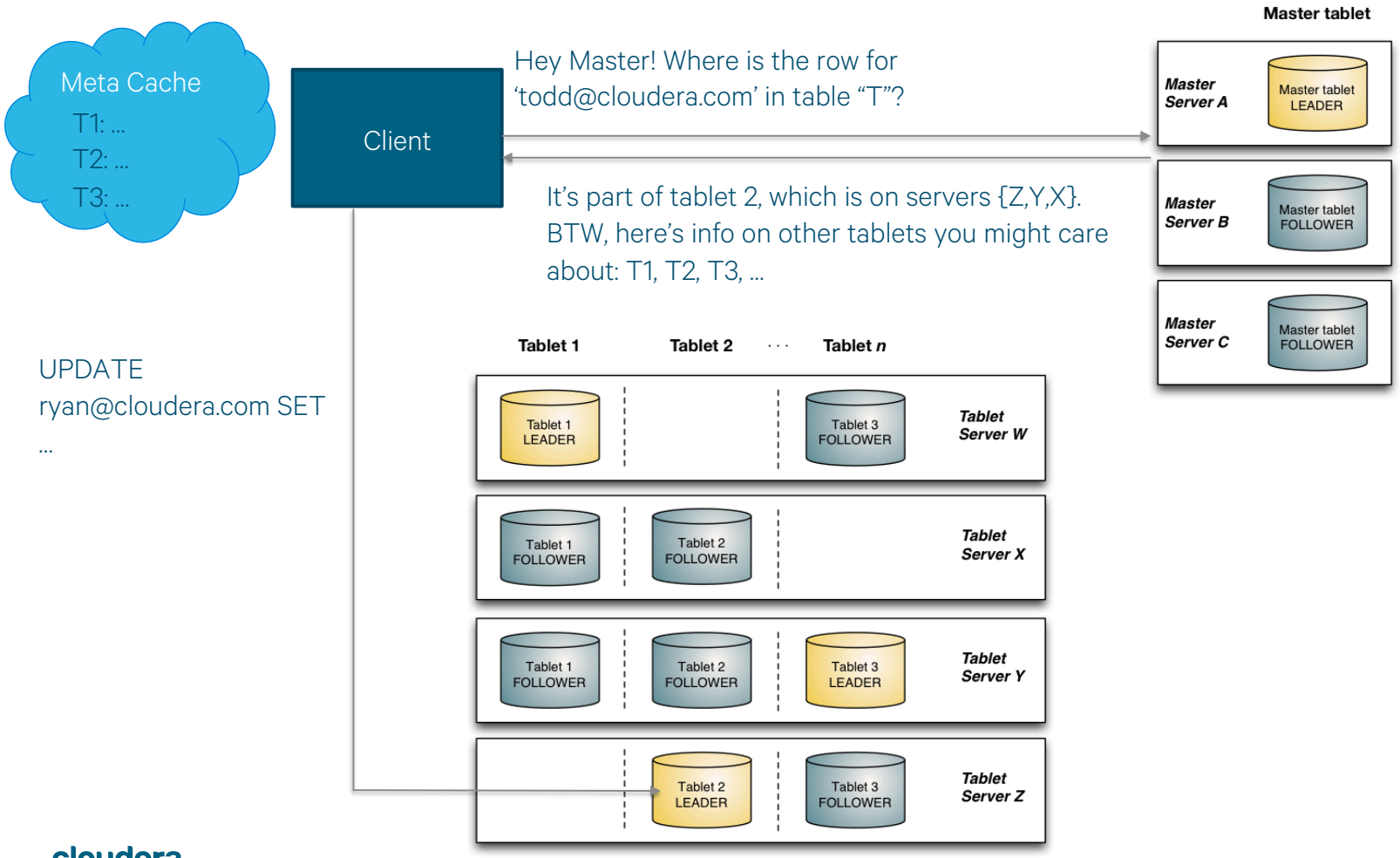

Tablet 1
Tablet 2
Tablet n
Tablet 1 LEADER
Tablet 3 FOLLOWER
Tablet Server W
Tablet 1 FOLLOWER
Tablet 2 FOLLOWER
Tablet Server X
Tablet 1 FOLLOWER
Tablet 2 FOLLOWER
Tablet 3 LEADER
Tablet Server Y
Tablet 2 LEADER
Tablet 3 FOLLOWER
Tablet Server Z

cloudera


Meta Cache
Client
Hey Master! Where is the row for 'ryan@cloudera.com' in table "T"?
Master tablet
Master Server A
Master tablet LEADER
Master Server B
Master tablet FOLLOWER
Master Server C
Master tablet FOLLOWER
Tablet 1
Tablet 2
Tablet n
Tablet 1 LEADER
Tablet 3 FOLLOWER
Tablet Server W
Tablet 1 FOLLOWER
Tablet 2 FOLLOWER
Tablet Server X
Tablet 1 FOLLOWER
Tablet 2 FOLLOWER
Tablet 3 LEADER
Tablet Server Y
Tablet 2 LEADER
Tablet 3 FOLLOWER
Tablet Server Z


Meta Cache
Client
Hey Master! Where is the row for 'todd@cloudera.com' in table "T"?
It's part of tablet 2, which is on servers {Z,Y,X}. BTW, here's info on other tablets you might care about: T1, T2, T3, ...
Master tablet
Master Server A
Master tablet LEADER
Master Server B
Master tablet FOLLOWER
Master Server C
Master tablet FOLLOWER
Tablet 1
Tablet 2
Tablet n
Tablet 1 LEADER
Tablet 3 FOLLOWER
Tablet Server W
Tablet 1 FOLLOWER
Tablet 2 FOLLOWER
Tablet Server X
Tablet 1 FOLLOWER
Tablet 2 FOLLOWER
Tablet 3 LEADER
Tablet Server Y
Tablet 2 LEADER
Tablet 3 FOLLOWER
Tablet Server Z


Meta Cache
T1: ...
T2: ...
T3: ...
Client
Hey Master! Where is the row for 'todd@cloudera.com' in table "T"?
It's part of tablet 2, which is on servers {Z,Y,X}. BTW, here's info on other tablets you might care about: T1, T2, T3, ...
Master tablet
Master Server A
Master tablet LEADER
Master Server B
Master tablet FOLLOWER
Master Server C
Master tablet FOLLOWER
Tablet 1
Tablet 2
Tablet n
Tablet 1 LEADER
Tablet 3 FOLLOWER
Tablet Server W
Tablet 1 FOLLOWER
Tablet 2 FOLLOWER
Tablet Server X
Tablet 1 FOLLOWER
Tablet 2 FOLLOWER
Tablet 3 LEADER
Tablet Server Y
Tablet 2 LEADER
Tablet 3 FOLLOWER
Tablet Server Z



UPDATE
ryan@cloudera.com SET
...

cloudera

Metadata

- Replicated master
  - Acts as a tablet directory
  - Acts as a catalog (which tables exist, etc)
  - Acts as a load balancer (tracks TS liveness, re-replicates under-replicated tablets)
- Caches all *metadata* in RAM for high performance
- Client configured with master addresses
  - Asks master for tablet locations as needed and caches them

Fault tolerance

- **Operations replicated** using **Raft** consensus
  - Strict quorum algorithm. See Raft paper for details
- **Transient failures:**
  - **Follower failure**: Leader can still achieve majority
  - **Leader failure**: automatic leader election (~5 seconds)
  - Restart dead TS within 5 min and it will rejoin transparently
- **Permanent failures**
  - After 5 minutes, automatically creates a new follower replica and copies data
- **N replicas** can tolerate maximum of **(N-1)/2 failures**

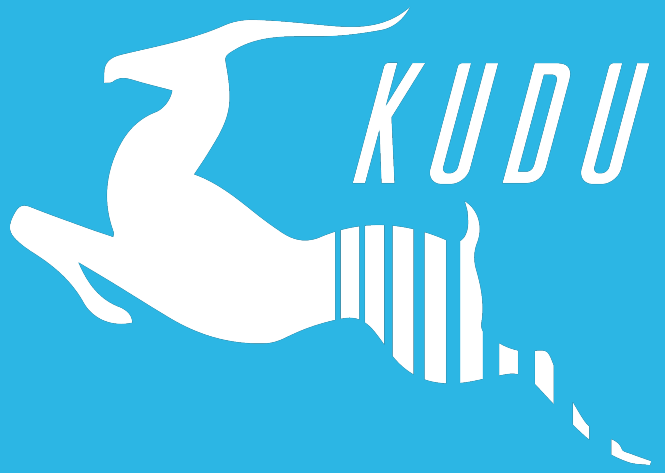
What Kudu is *NOT*

- Not a SQL interface itself
  - It's just the storage layer
- Not an application that runs on HDFS
  - It's an alternative, native Hadoop storage engine
- Not a replacement for HDFS or HBase
  - Select the right storage for the right use case
  - Cloudera will continue to support and invest in all three

Kudu Trade-Offs (vs Hbase)

- Random updates will be slower
  - HBase model allows random updates without incurring a disk seek
  - Kudu requires a key lookup before update, Bloom lookup before insert

- Single-row reads may be slower
  - Columnar design is optimized for scans
  - Future: may introduce “column groups” for applications where single-row access is more important



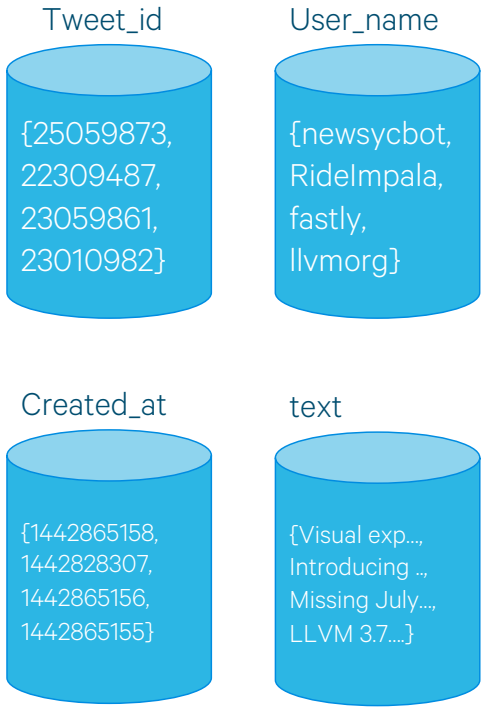
How it works

Replication and fault tolerance

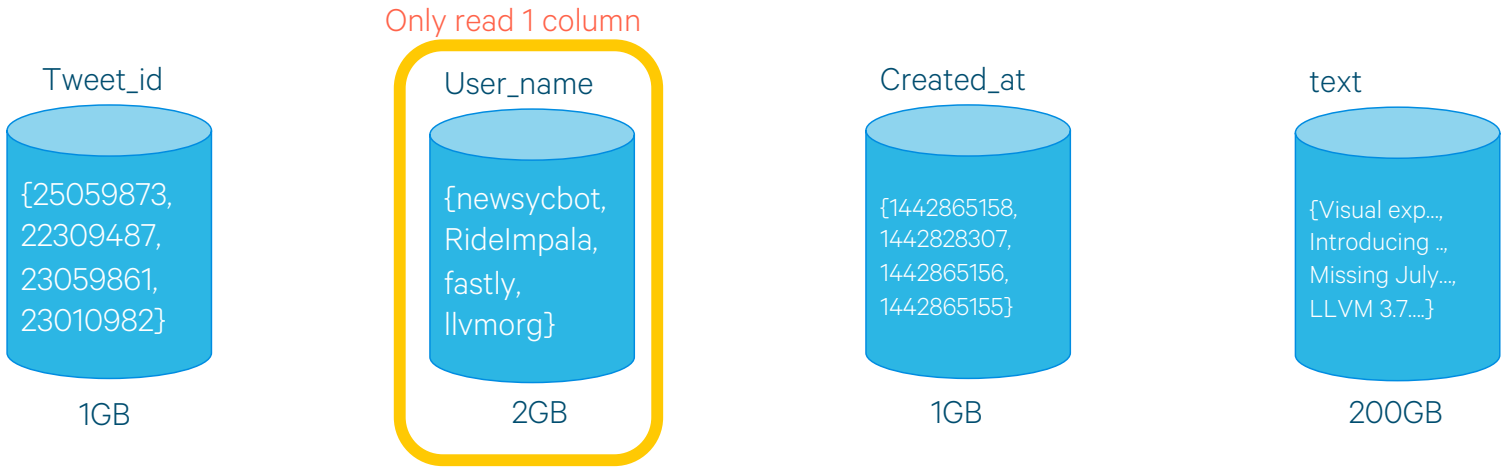
Columnar storage

Twitter Firehose Table

tweet_id	user_name	created_at	text
INT64	STRING	TIMESTAMP	STRING
23059873	newsycbot	1442865158	Visual Explanation of the Raft Consensus Algorithm http://bit.ly/1DOUac0 (cmts http://bit.ly/1HKmjfc)
22309487	RideImpala	1442828307	Introducing the Ibis project: for the Python experience at Hadoop Scale
23059861	fastly	1442865156	Missed July's SF @papers_we_love? You can now watch @el_bhs talk about @google's globally-distributed database: http://fastly.us/1eVz8MM
23010982	llvmorg	1442865155	LLVM 3.7 is out! Get it while it's HOT! http://llvm.org/releases/download.html#3.7.0



Columnar storage



SELECT COUNT(*) FROM tweets WHERE user_name = ‘newsycbot’;

Columnar compression

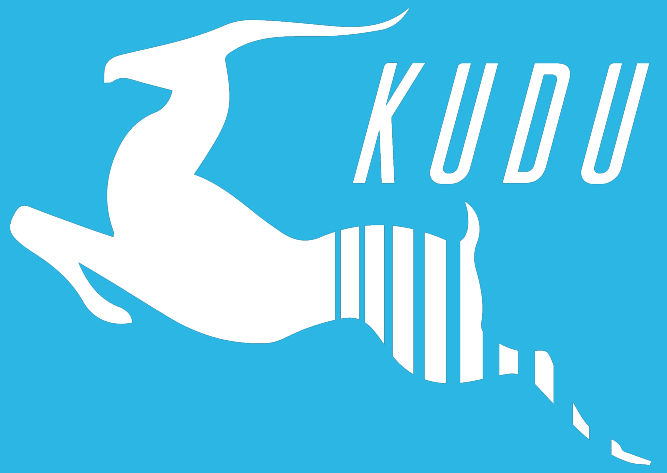
Created_at	Diff(created_at)
1442865158	n/a
1442828307	-36851
1442865156	36849
1442865155	-1
64 bits each	17 bits each

- Many columns can compress to a few bits per row!
- Especially:
  - Timestamps
  - Time series values
  - Low-cardinality strings
- Massive space savings and throughput increase!

Handling inserts and updates

- Inserts go to an in-memory row store (MemRowSet)
  - Durable due to write-ahead logging
  - Later flush to columnar format on disk
- Updates go to in-memory "delta store"
  - Later flush to "delta files" on disk
  - Eventually "compact" into the previously-written columnar data files

- Details elided here due to time constraints
  - Read the Kudu whitepaper at http://getkudu.io/kudu.pdf to learn more!



Integrations

Spark Integration (WIP, available in 0.9)

```
val df = sqlContext.read.options(kuduOptions)
      .format("org.kududb.spark.kudu").load

val changedDF = df.limit(1)
      .withColumn("key", df("key").plus(100))
      .withColumn("c2_s", lit("abc"))

changedDF.write.options(kuduOptions)
      .mode("append")
      .format("org.kududb.spark.kudu").save
```

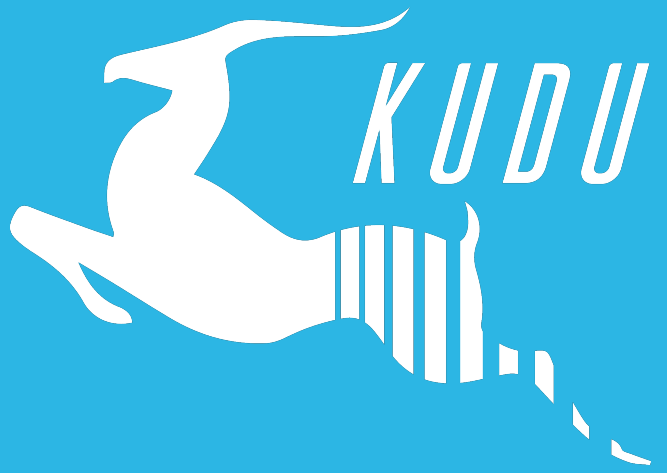
Impala integration

- `CREATE TABLE … DISTRIBUTE BY HASH(vehicle_id) INTO 16 BUCKETS AS SELECT … FROM …`
- `INSERT/UPDATE/DELETE`

- Optimizations like predicate pushdown, scan parallelism, plans for more on the way

MapReduce integration

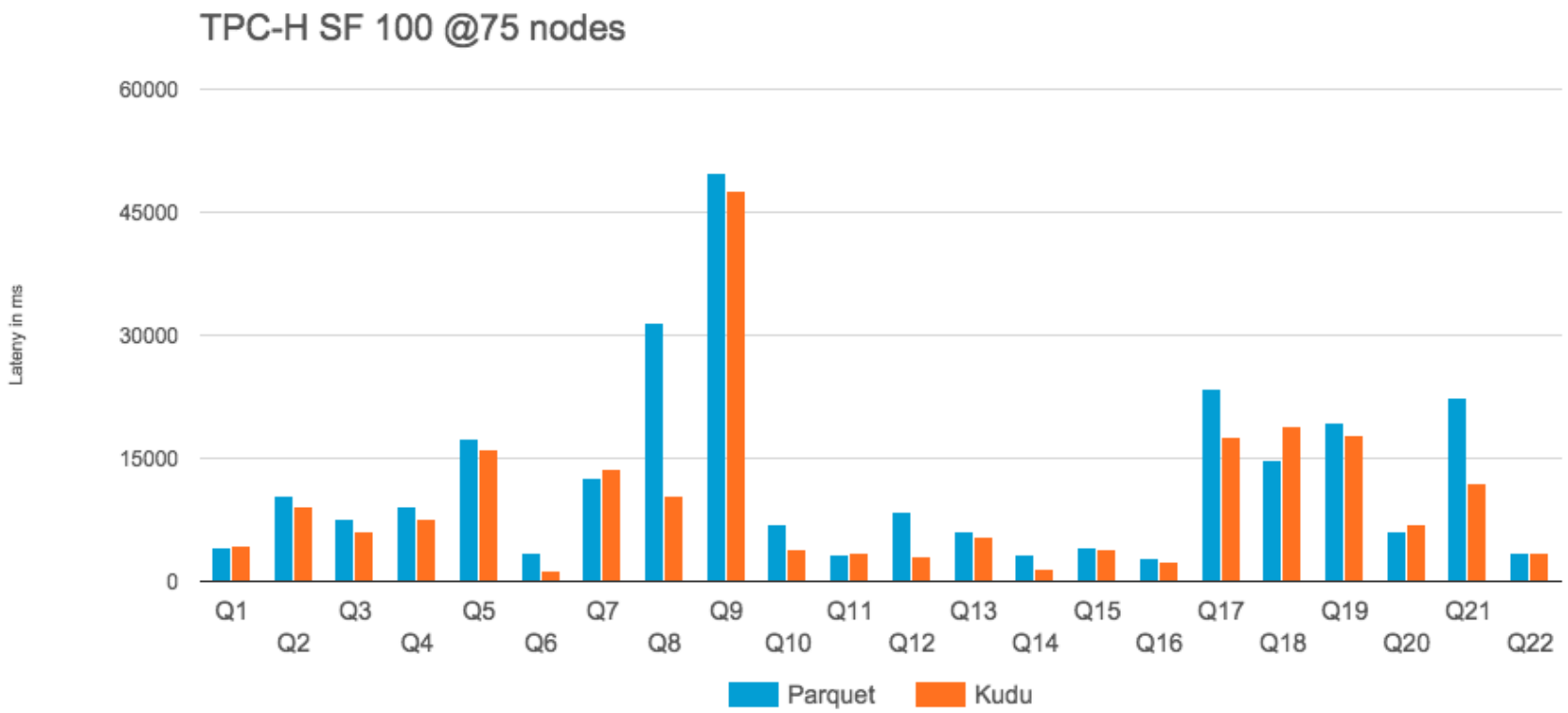
- Most Kudu integration/correctness testing via MapReduce
- Multi-framework cluster (MR + HDFS + Kudu on the same disks)
- KuduTableInputFormat / KuduTableOutputFormat
  - Support for pushing down predicates, column projections, etc.



Performance

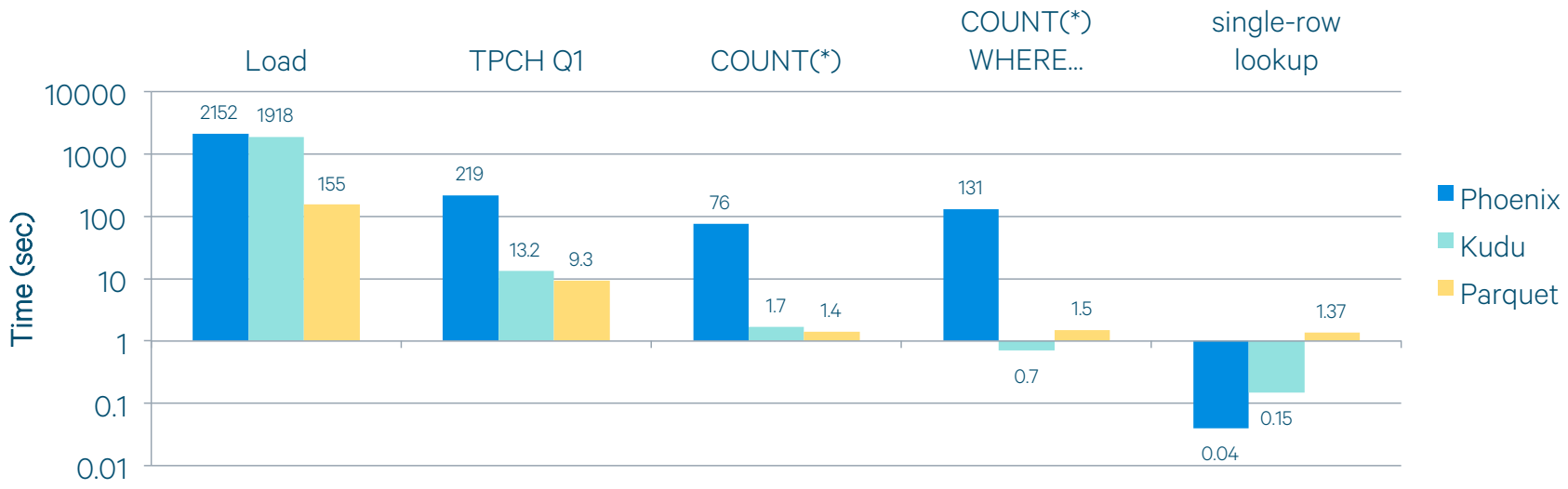
TPC-H (analytics benchmark)

- 75 server cluster
  - 12 (spinning) disks each, enough RAM to fit dataset
  - TPC-H Scale Factor 100 (100GB)
- Example query:
  - ```
    SELECT n_name, sum(l_extendedprice * (1 - l_discount)) as revenue FROM customer, orders,
    lineitem, supplier, nation, region WHERE c_custkey = o_custkey AND l_orderkey =
    o_orderkey AND l_suppkey = s_suppkey AND c_nationkey = s_nationkey AND s_nationkey =
    n_nationkey AND n_regionkey = r_regionkey AND r_name = 'ASIA' AND o_orderdate >= date
    '1994-01-01' AND o_orderdate < '1995-01-01’ GROUP BY n_name ORDER BY revenue desc;
    ```
    ```


TPC-H SF 100 @75 nodes
Latency in ms
60000
45000
30000
15000
0
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q8
Q9
Q10
Q11
Q12
Q13
Q14
Q15
Q16
Q17
Q18
Q19
Q20
Q21
Q22
Parquet
Kudu

Versus other NoSQL storage

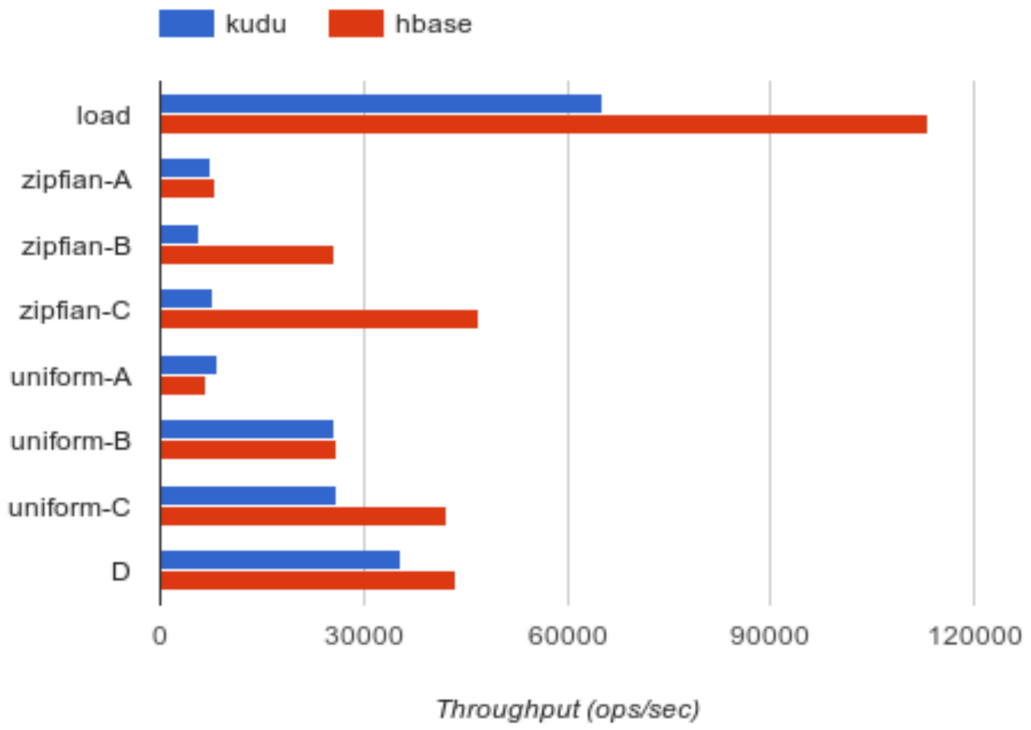
- Apache Phoenix: OLTP SQL engine built on HBase
- 10 node cluster (9 worker, 1 master)
- TPC-H LINEITEM table only (6B rows)



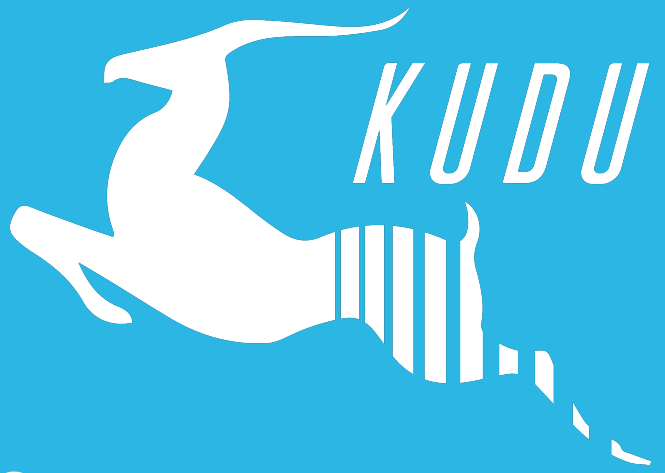
cloudera

What about NoSQL-style random access? (YCSB)

- YCSB 0.5.0-snapshot
- 10 node cluster (9 worker, 1 master)
- 100M row data set
- 10M operations each workload



cloudera

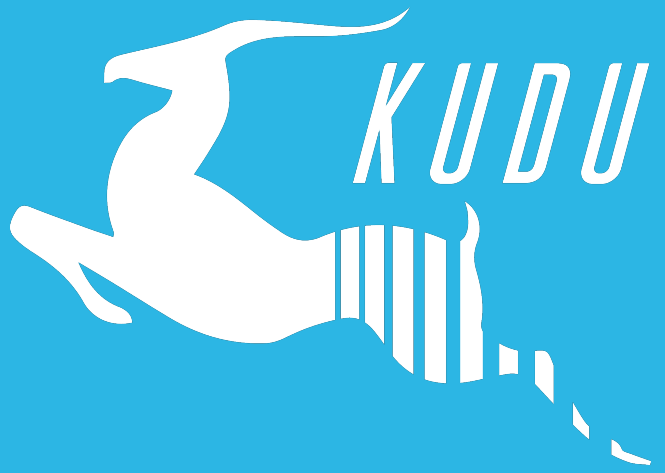


Getting started with Kudu

Getting started as a user

- http://getkudu.io
- kudu-user@googlegroups.com
- http://getkudu-slack.herokuapp.com/

- Quickstart VM
  - Easiest way to get started
  - Impala and Kudu in an easy-to-install VM
- CSD and Parcels
  - For installation on a Cloudera Manager-managed cluster



Questions?

http://getkudu.io

bosshart@cloudera.com

BETA SAFE HARBOR WARNING

- Kudu is BETA (DO NOT PUT IT IN PRODUCTION)
- Please play with it, and let us know your feedback
- Please consider this when building out architectures for second half of 2016
- Why?
  - Storage is important and needs to be stable
    - (That said: we have not experienced data loss. Kudu is reasonably stable, almost no crashes reported)
  - Still requires some expert assistance, and you'll probably find some bugs