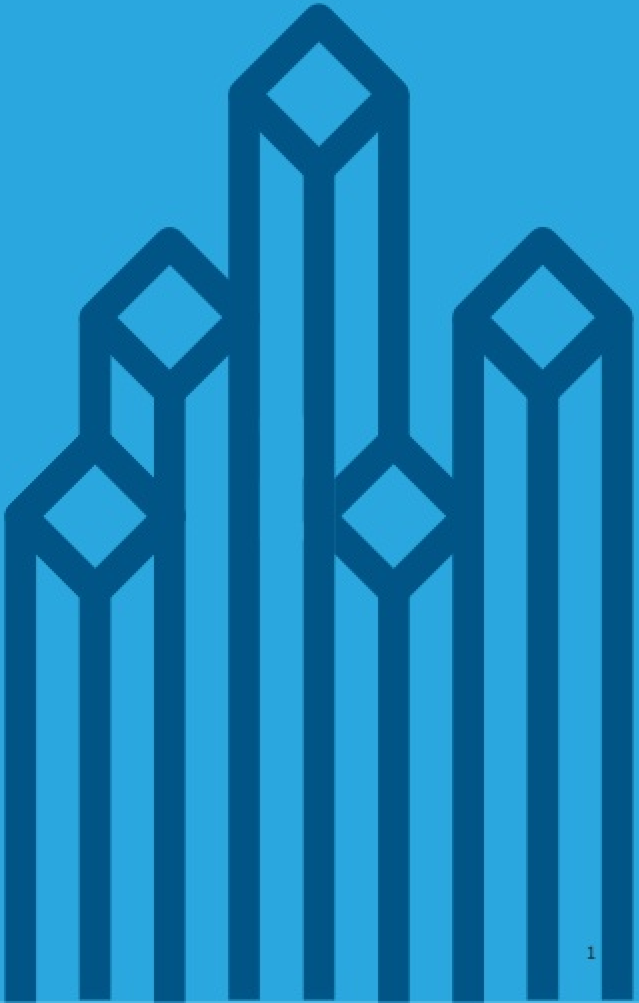
cloudera

Apache Kudu*: Fast Analytics on Fast Data

Todd Lipcon (Kudu team lead) – todd@cloudera.com
@tlipcon

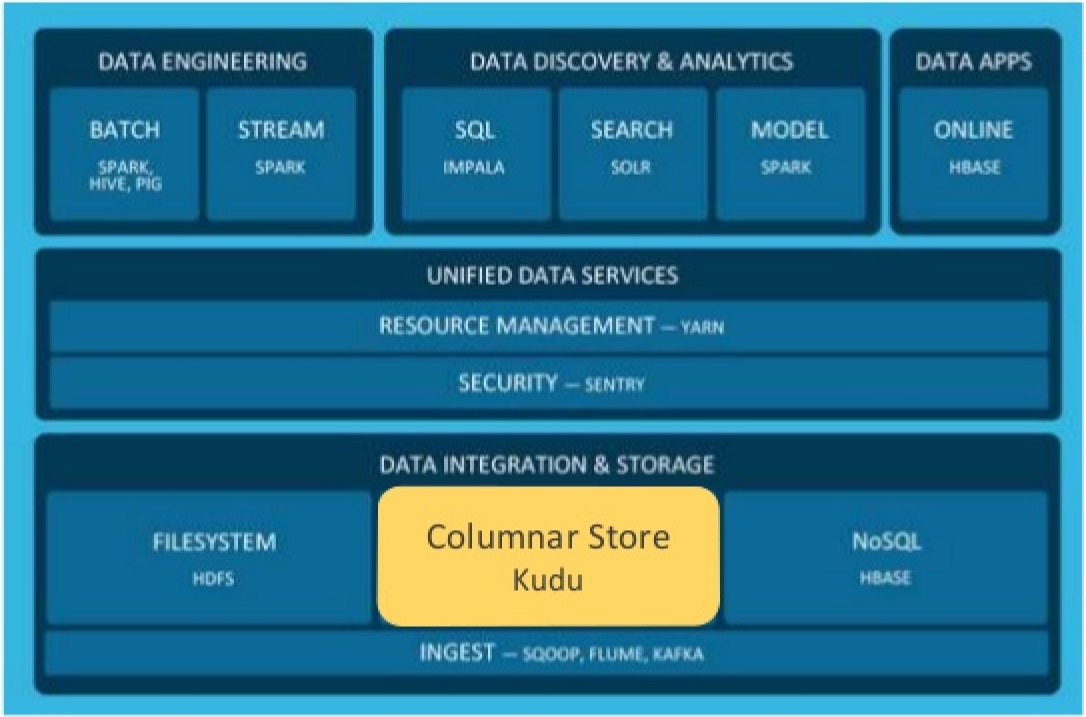
Tweet about this talk: @apachekudu or #kudu

* Incubating at the Apache Software Foundation



Apache Kudu

Storage for Fast Analytics on Fast Data

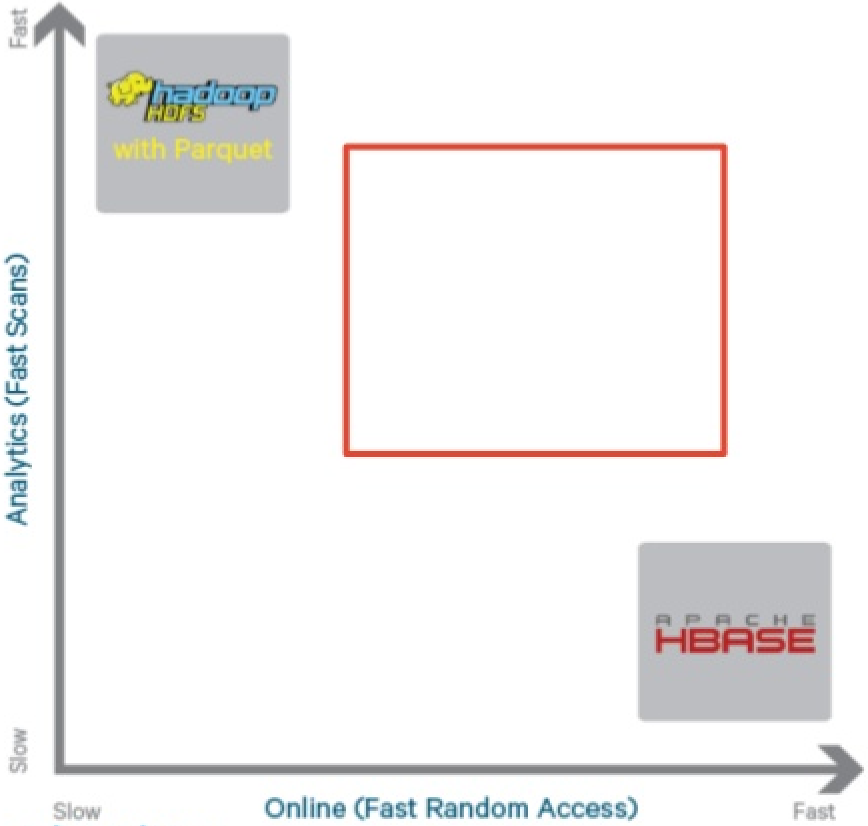


- New updatable column store for Hadoop
- Apache-licensed open source
- Beta now available

cloudera

Why Kudu?

Current Storage Landscape in Hadoop Ecosystem



HDFS (GFS) excels at:

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

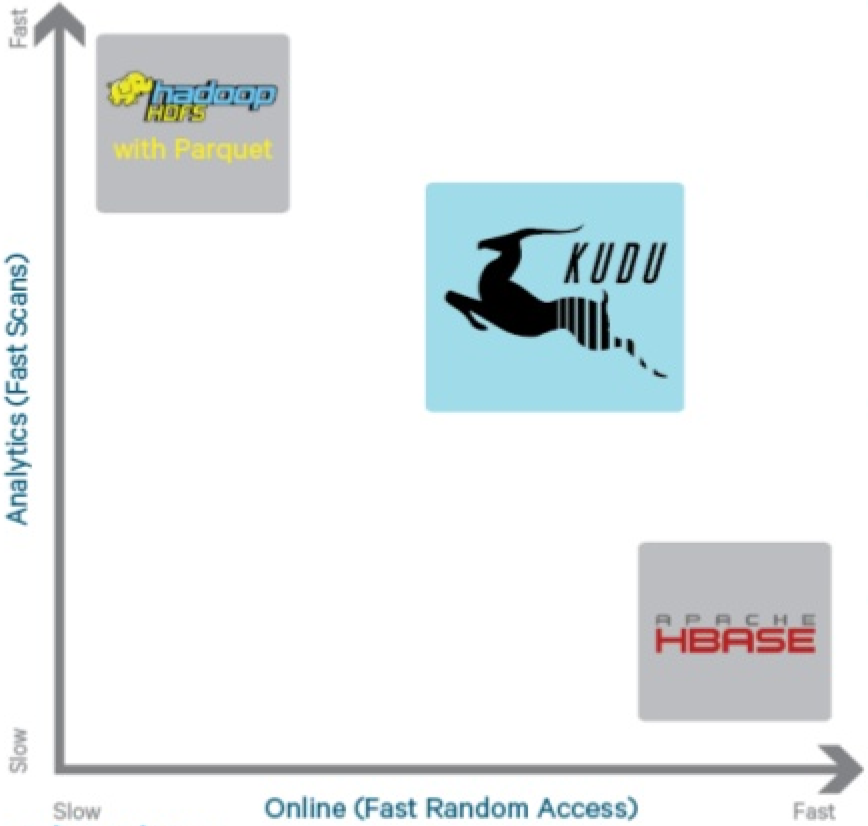
HBase (BigTable) excels at:

- Efficiently finding and writing individual rows
- Making data mutable

Gaps exist when these properties are needed *simultaneously*

cloudera

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)

cloudera

Changing Hardware landscape

- **Spinning disk** -> **solid state storage**
  - **NAND flash**: Up to 450k read 250k write iops, about 2GB/sec read and 1.5GB/sec write throughput, at a price of less than $3/GB and dropping
  - **3D XPoint memory** (1000x faster than NAND, cheaper than RAM)

- **RAM** is cheaper and more abundant:
  - 64->128->256GB over last few years

- ***Takeaway*:** The **next bottleneck is CPU**, and current storage systems weren't designed with CPU efficiency in mind.

What’s Kudu?

Scalable and Fast Tabular Storage

- **Scalable**
  - Tested up to 275 nodes (~3PB cluster)
  - Designed to scale to **1000s of nodes, tens of PBs**
- **Fast**
  - **Millions** of read/write operations per second across cluster
  - **Multiple GB/second** read throughput per node
- **Tabular**
  - **SQL-like** schema**: finite number** of **typed** columns (unlike HBase/Cassandra)
  - **Fast ALTER TABLE**
  - **"NoSQL"** APIs: Java/C++/Python **or SQL** (Impala/Spark/etc)

Use cases and architectures

Kudu Use Cases

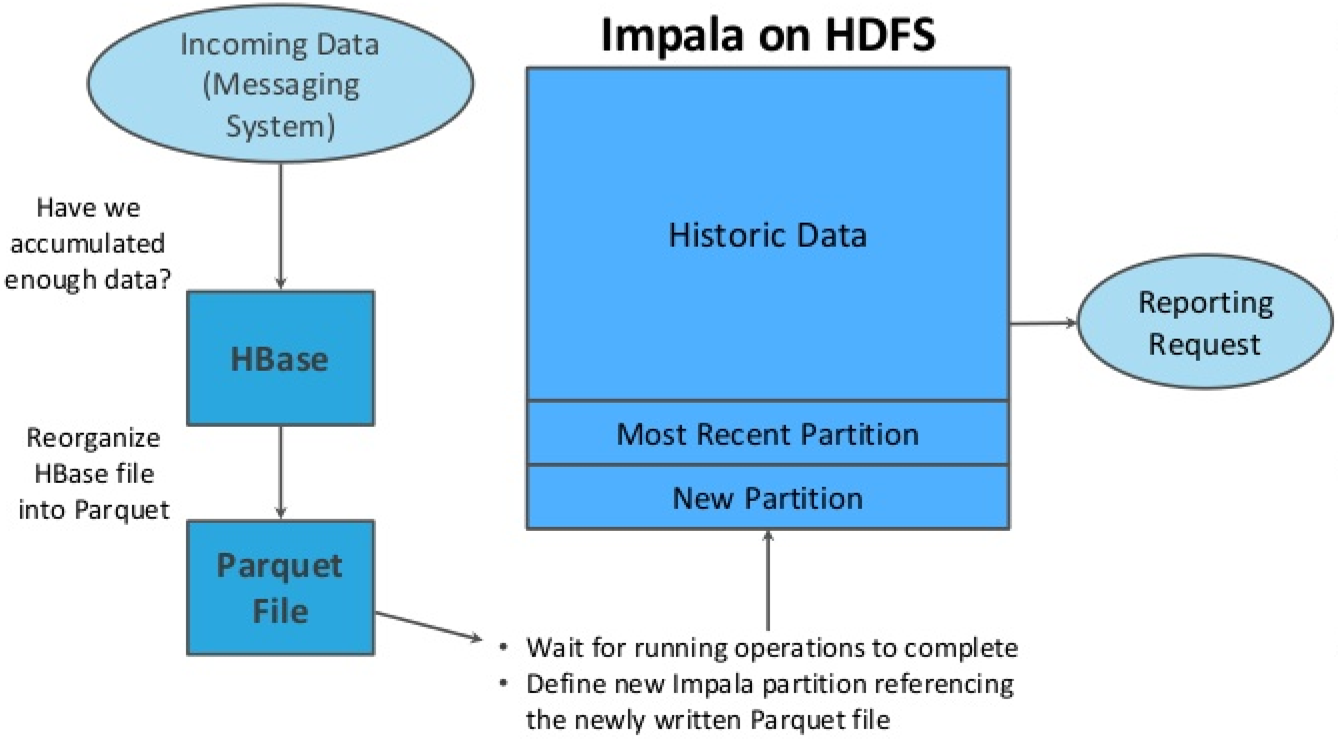
Kudu is best for use cases requiring a simultaneous combination of sequential and random reads and writes

- **Time Series**
  - Examples: Stream market data; fraud detection & prevention; network monitoring
  - Workload: Insert, updates, scans, lookups

- **Online Reporting**
  - Examples: ODS
  - Workload: Inserts, updates, scans, lookups

Real-Time Analytics in Hadoop Today

Fraud Detection in the Real World = Storage Complexity


Incoming Data (Messaging System)
Have we accumulated enough data?
HBase
Reorganize HBase file into Parquet
Parquet File
Impala on HDFS
Historic Data
Most Recent Partition
New Partition
Reporting Request

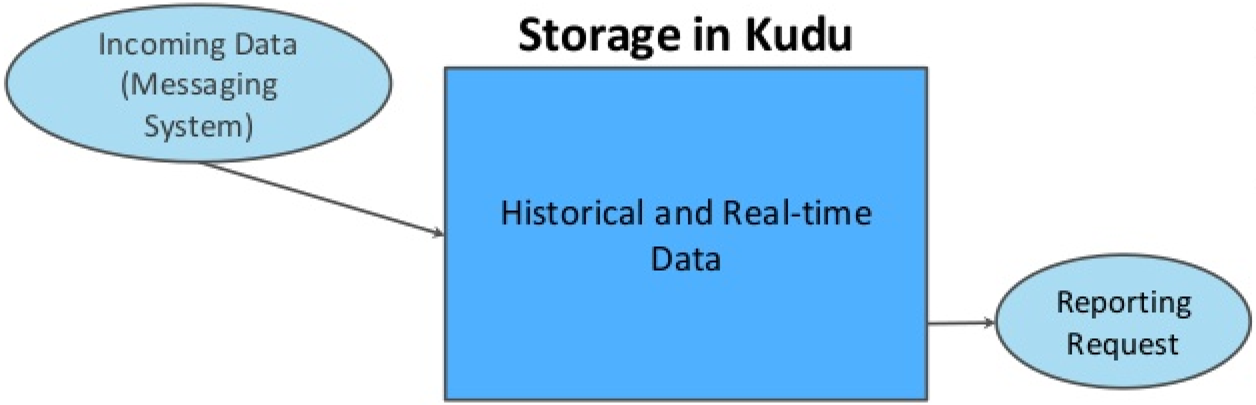
- Wait for running operations to complete
- Define new Impala partition referencing the newly written Parquet file

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?

cloudera

Real-Time Analytics in Hadoop with Kudu


Incoming Data (Messaging System)
Storage in Kudu
Historical and Real-time Data
Reporting Request

Improvements:

- **One system** to operate
- **No cron jobs or background processes**
- **Handle late arrivals or data corrections with ease**
- **New data available immediately for analytics or operations**

cloudera

Xiaomi Use Case

- World's 4th largest smart-phone maker (most popular in China)
- Gather important RPC tracing events from mobile app and backend service.
- Service monitoring & troubleshooting tool.

High write throughput

- >5 Billion records/day and growing

Query latest data and quick response

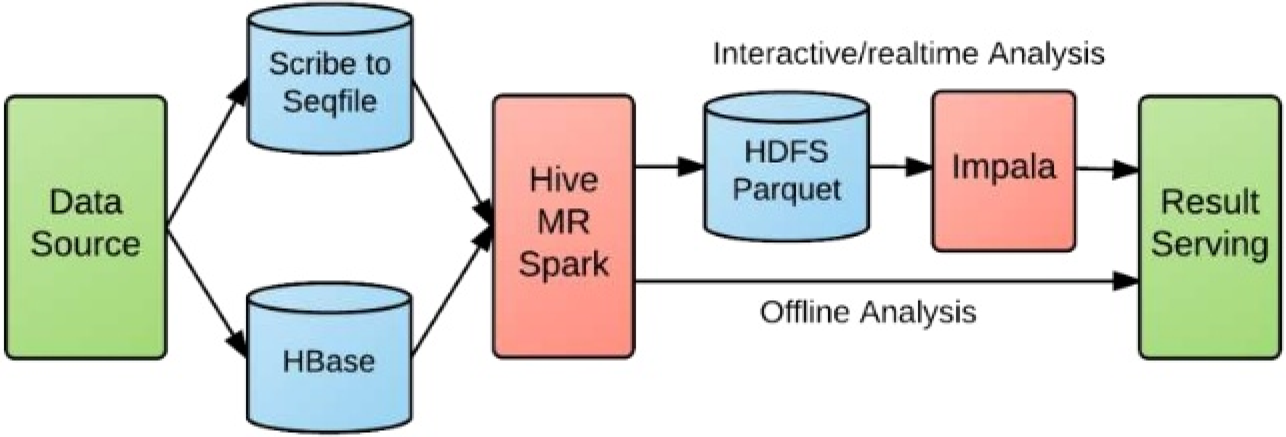
- Identify and resolve issues quickly

Can search for individual records

- Easy for troubleshooting

Xiaomi Big Data Analytics Pipeline

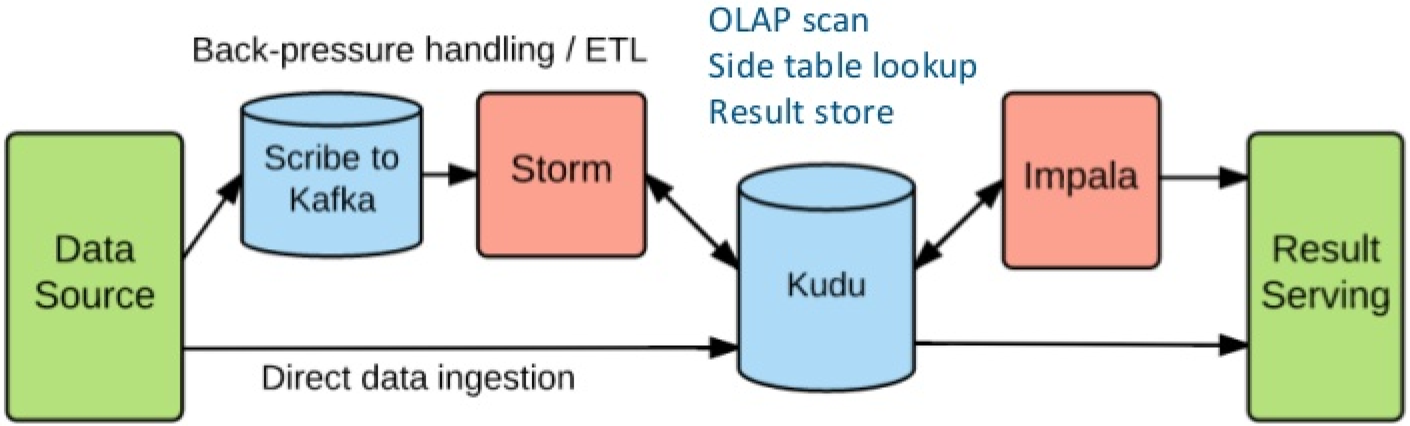
Before Kudu



- **Long pipeline**
  high latency(1 hour ~ 1 day), data conversion pains
- **No ordering**
  Log arrival(storage) order not exactly logical order
  e.g. read 2-3 days of log for data in 1 day

Xiaomi Big Data Analysis Pipeline

Simplified With Kudu



- ETL Pipeline(0~10s latency)
  Apps that need to prevent backpressure or require ETL
- Direct Pipeline(no latency)
  Apps that don't require ETL and no backpressure issues

How it Works

Replication and fault tolerance

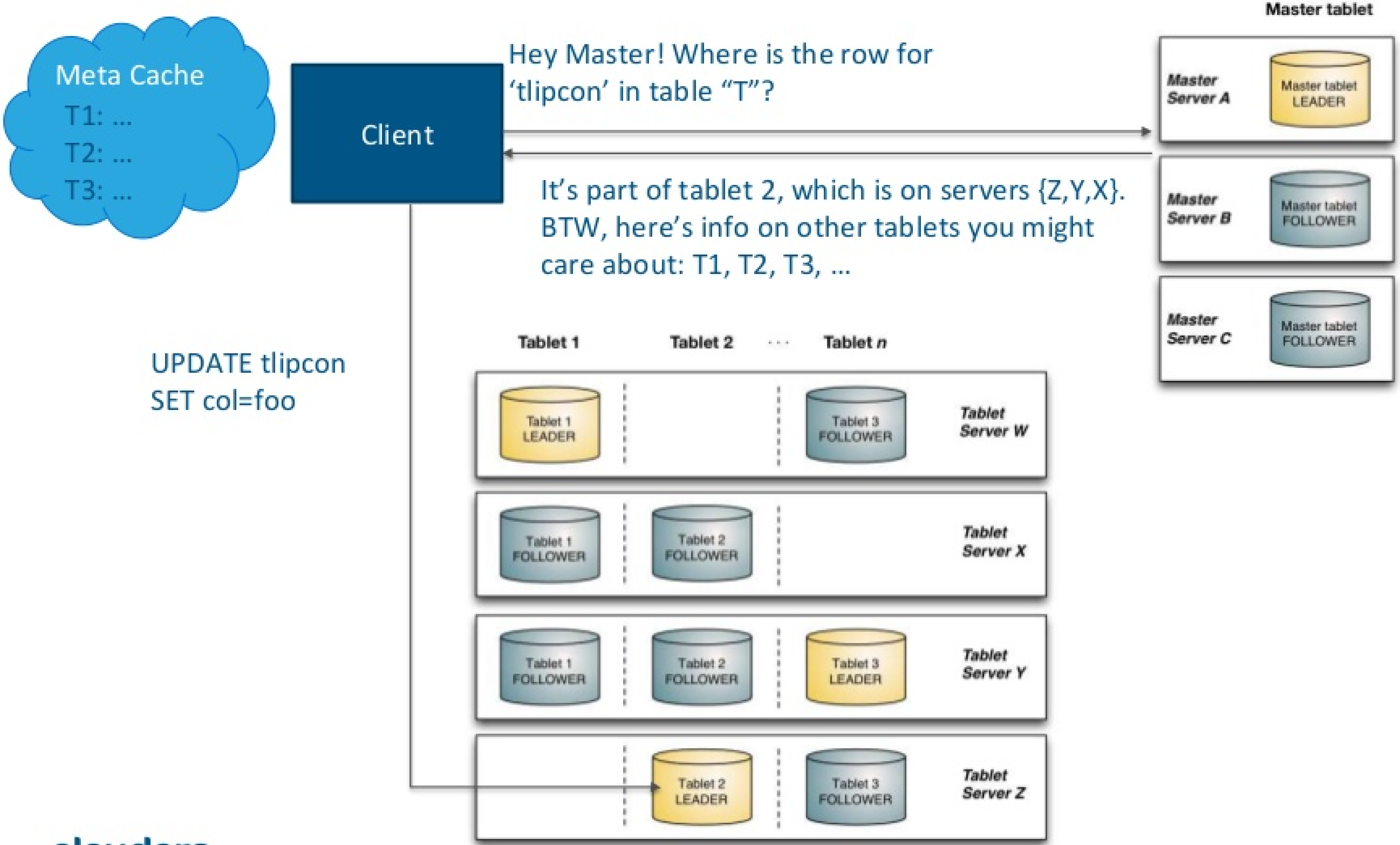
cloudera

Tables, Tablets, and Tablet Servers

- Table is **horizontally partitioned into *tablets***
  - *Range* or *hash* partitioning
  - `PRIMARY KEY (host, metric, timestamp) DISTRIBUTE BY HASH(timestamp) INTO 100 BUCKETS`
    - `bucketNumber = hashCode(row[‘timestamp’]) % 100`
- Each tablet has N **replicas** (3 or 5), with **Raft consensus**
  - Automatic **fault tolerance**
  - MTTR: ~5 seconds
- **Tablet servers** host tablets on local disk drives

Metadata and the Master

- **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)
- **Not a bottleneck**
  - super fast in-memory lookups

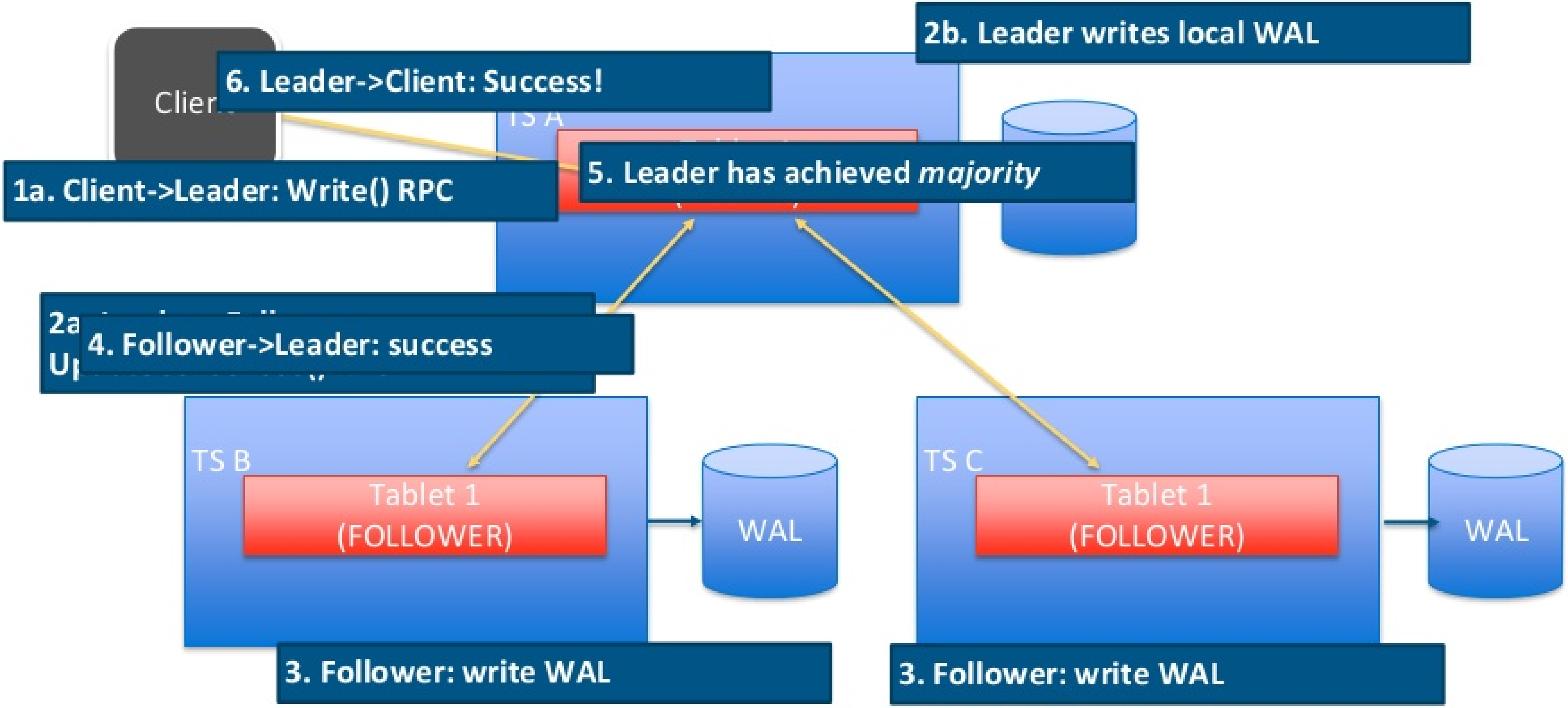


Hey Master! Where is the row for 'tlipcon' in table "T"?

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, …

UPDATE tlipcon
SET col=foo

Raft Consensus



1a. Client->Leader: Write() RPC

2b. Leader writes local WAL

2a

Up

3. Follower: write WAL

3. Follower: write WAL

4. Follower->Leader: success

5. Leader has achieved *majority*

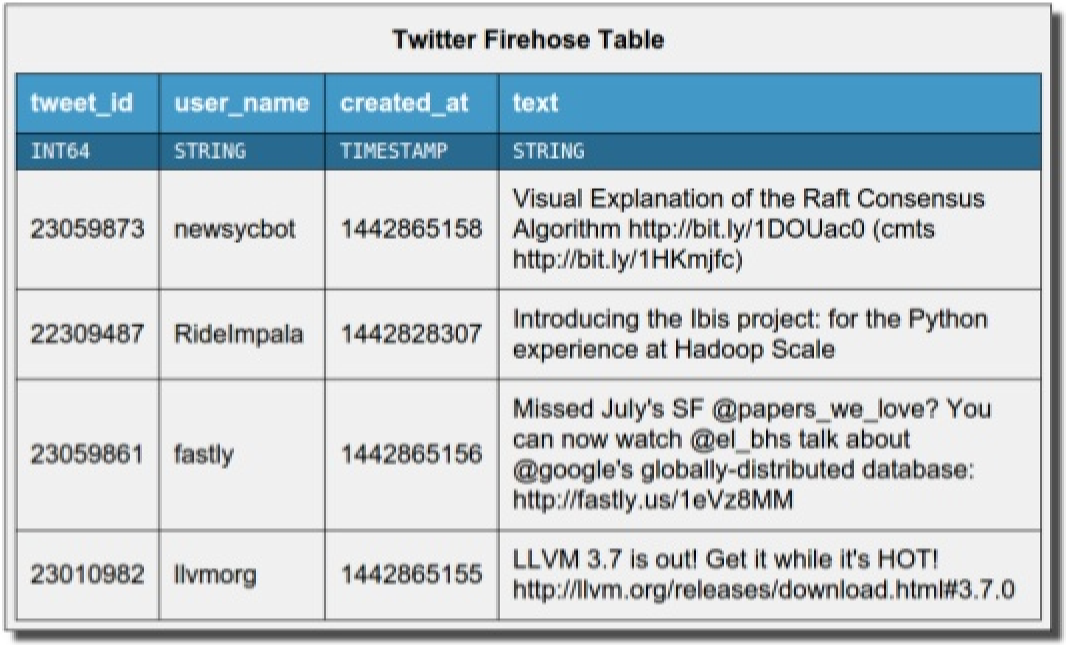
6. Leader->Client: Success!

How it Works

Columnar storage

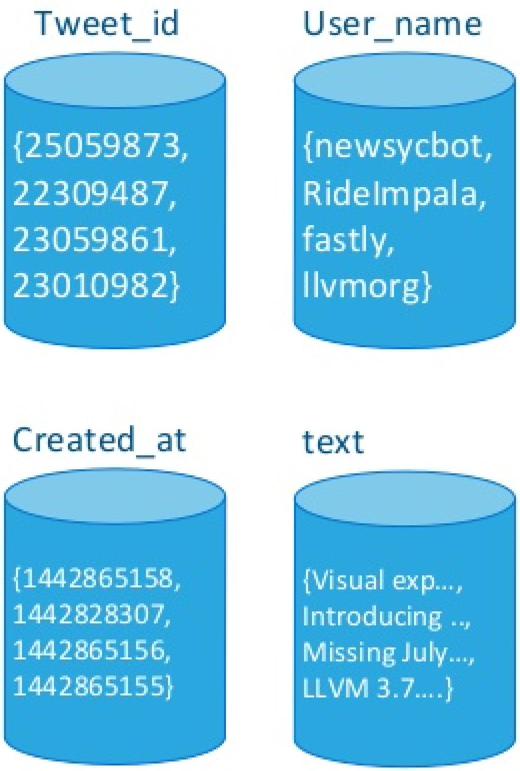
cloudera

Columnar Storage



Twitter Firehose Table

tweet_id INT64	user_name STRING	created_at TIMESTAMP	text 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



Tweet_id

{25059873, 22309487, 23059861, 23010982}

User_name

{newsycbot, RideImpala, fastly, llvmorg}

Created_at

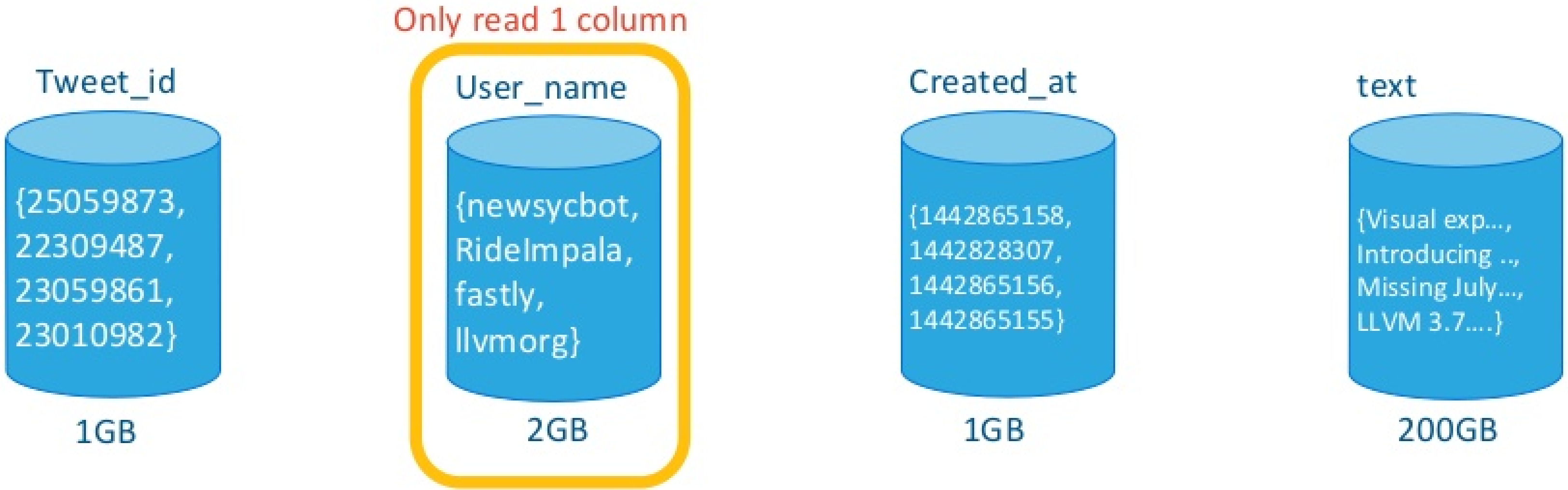
{1442865158, 1442828307, 1442865156, 1442865155}

text

{Visual exp..., Introducing .., Missing July..., LLVM 3.7....}

cloudera

Columnar Storage



Only read 1 column

Tweet_id

{25059873, 22309487, 23059861, 23010982}

1GB

User_name

{newsycbot, RideImpala, fastly, llvmorg}

2GB

Created_at

{1442865158, 1442828307, 1442865156, 1442865155}

1GB

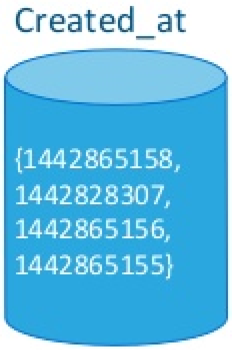
text

{Visual exp…, Introducing .., Missing July…, LLVM 3.7….}

200GB

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

Columnar Compression



Created_at

{1442865158, 1442828307, 1442865156, 1442865155}

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
  - available in other slide decks online, or come to office hours to learn more!

Integrations

Spark DataSource Integration (WIP)

```
sqlContext.load("org.kududb.spark",
     Map("kudu.table" -> “foo”,
         "kudu.master" -> “master.example.com”))
   .registerTempTable(“mytable”)
df = sqlContext.sql(
   “select col_a, col_b from mytable “ +
   “where col_c = 123”)
```

Available in Kudu 0.7.0, but still being improved

Impala Integration

- `CREATE TABLE … DISTRIBUTE BY HASH(col1) INTO 16 BUCKETS AS SELECT … FROM …`
- `INSERT/UPDATE/DELETE`
- Optimizations like predicate pushdown, scan parallelism, more on the way

- Not an Impala user? Community working on other integrations (Hive, Drill, Presto, Phoenix)

MapReduce Integration

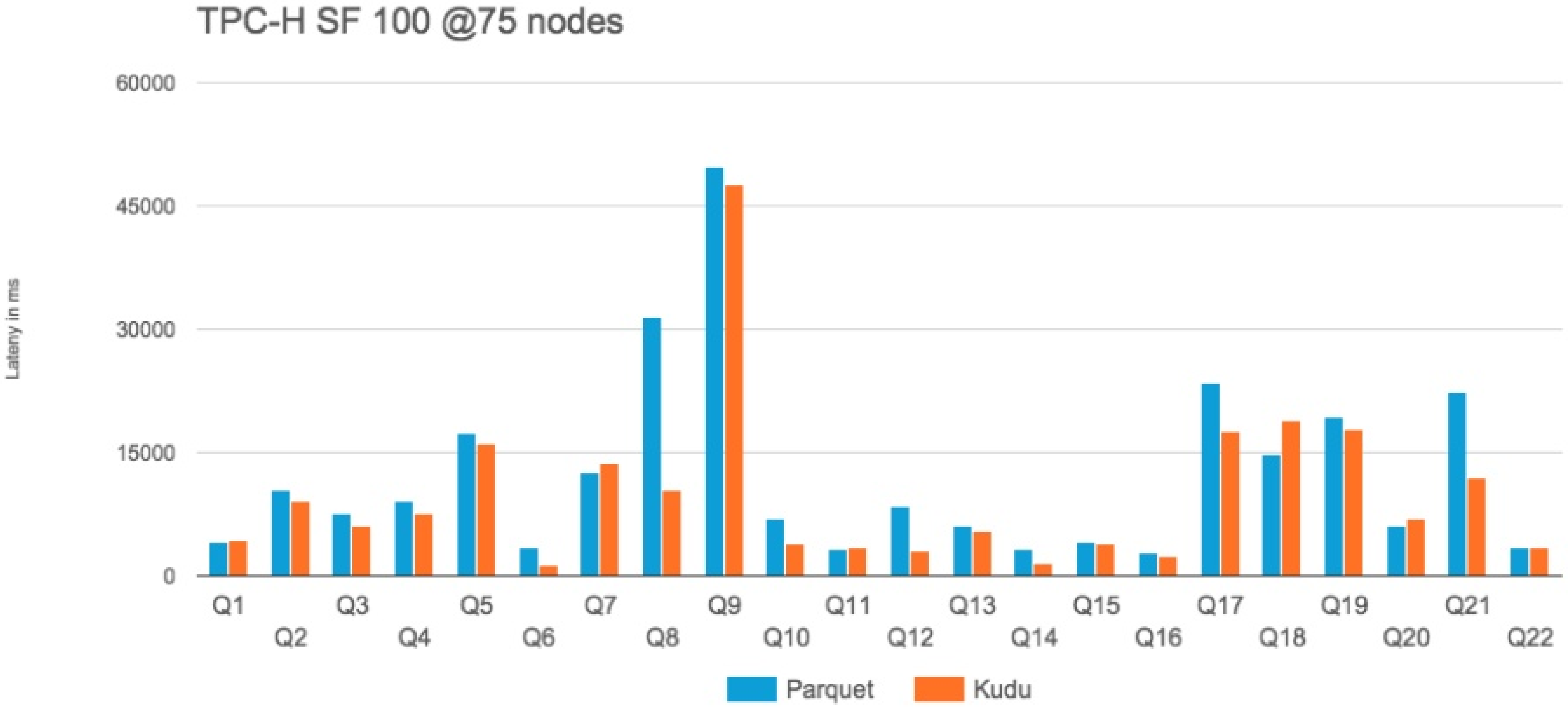
- Multi-framework cluster (MR + HDFS + Kudu on the same disks)
- **KuduTableInputFormat / KuduTableOutputFormat**
  - Support for pushing predicates, column projections, etc

Performance

TPC-H (Analytics benchmark)

- 75 server cluster
  - 12 (spinning) disk 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;
    ```

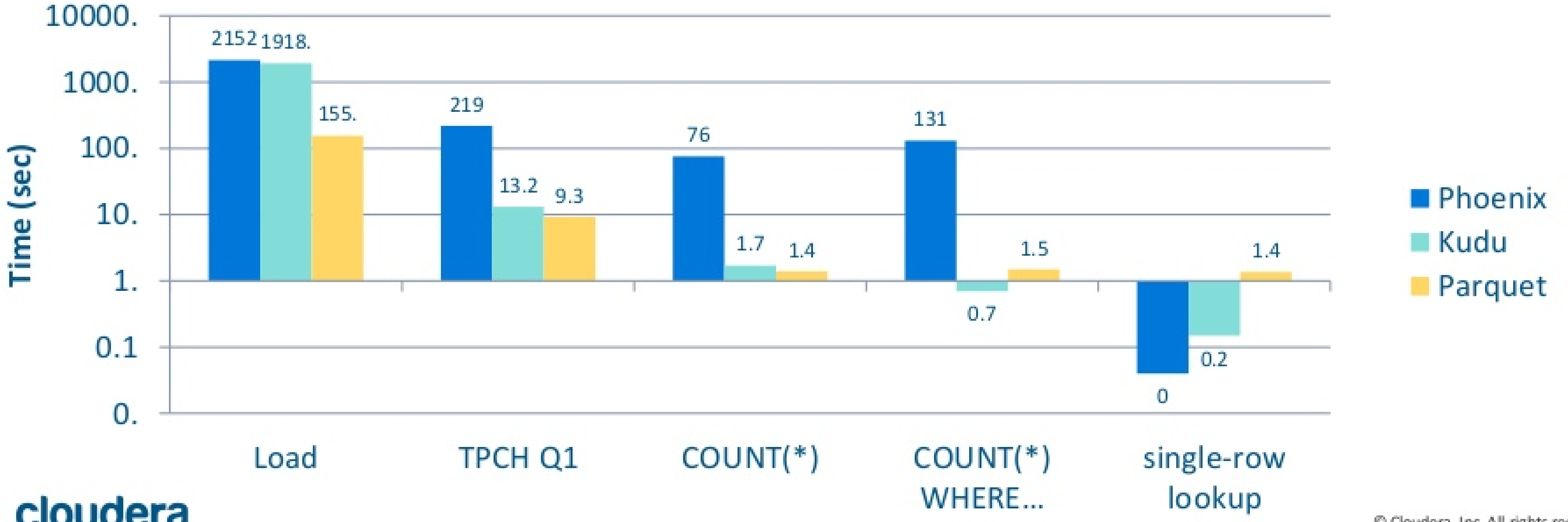


- Kudu outperforms Parquet by 31% (geometric mean) for RAM-resident data

cloudera

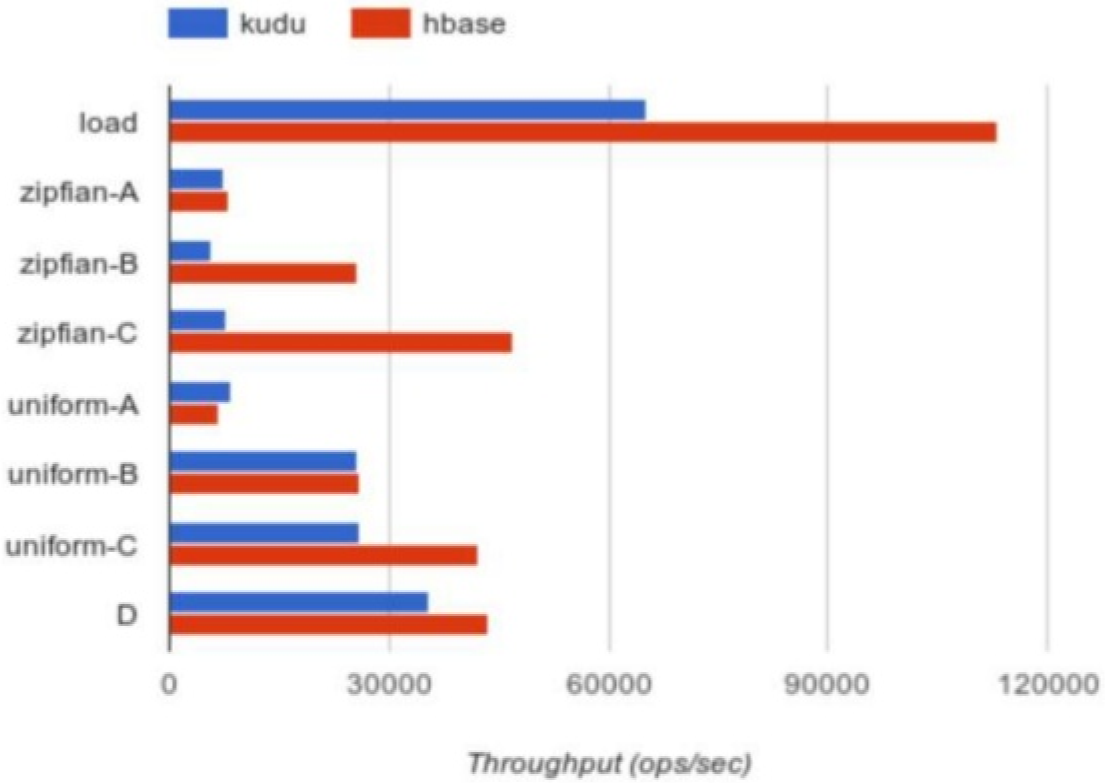
Versus other NoSQL Storage

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



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



Getting started

Project Status

- Open source beta released in September
- Latest release 0.7.1 hot off the presses
  - Usable for many applications
  - Have not experienced unrecoverable data loss, reasonably stable (almost no crashes reported). Users testing up to 200 nodes so far.
  - Still requires some expert assistance, and you'll probably find some bugs
- Part of the **Apache Software Foundation** Incubator
  - Community-driven open source process

Apache Kudu Community

cloudera
MI
intel®
JD.COM 京东
多·快·好·省
dremio

Getting Started As a User

- http://getkudu.io
- user@kudu.incubator.apache.org
- 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

Getting Started As a Developer

- http://github.com/apache/incubator-kudu
- Code reviews: http://gerrit.cloudera.org
- Public JIRA: http://issues.apache.org/jira/browse/KUDU
  - Includes bugs going back to 2013. Come see our dirty laundry!
- Mailing list: dev@kudu.incubator.apache.org

- Apache 2.0 license open source
- Contributions are welcome and encouraged!

http://getkudu.io/
@getkudu

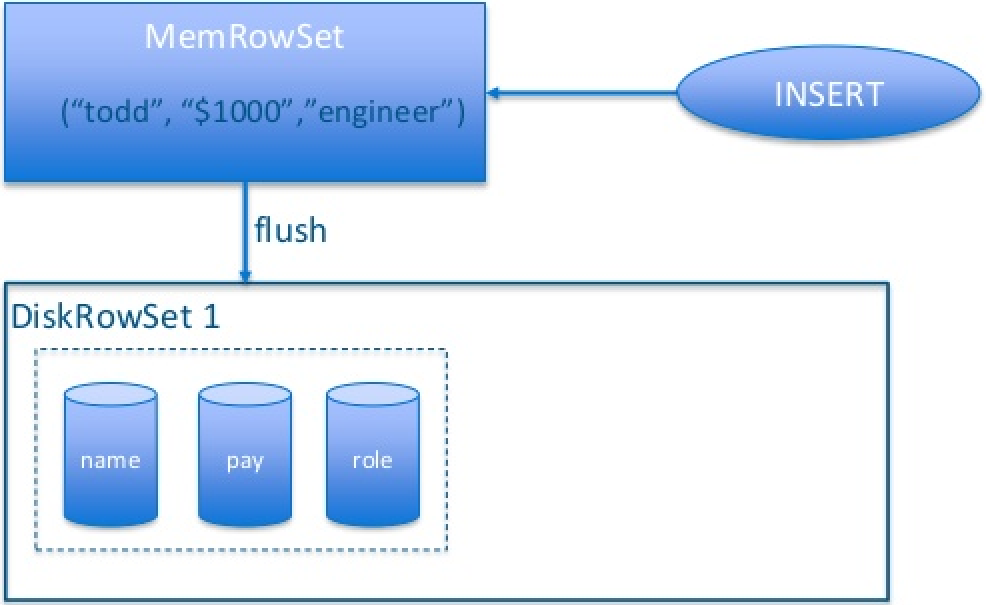
Backup slides

cloudera

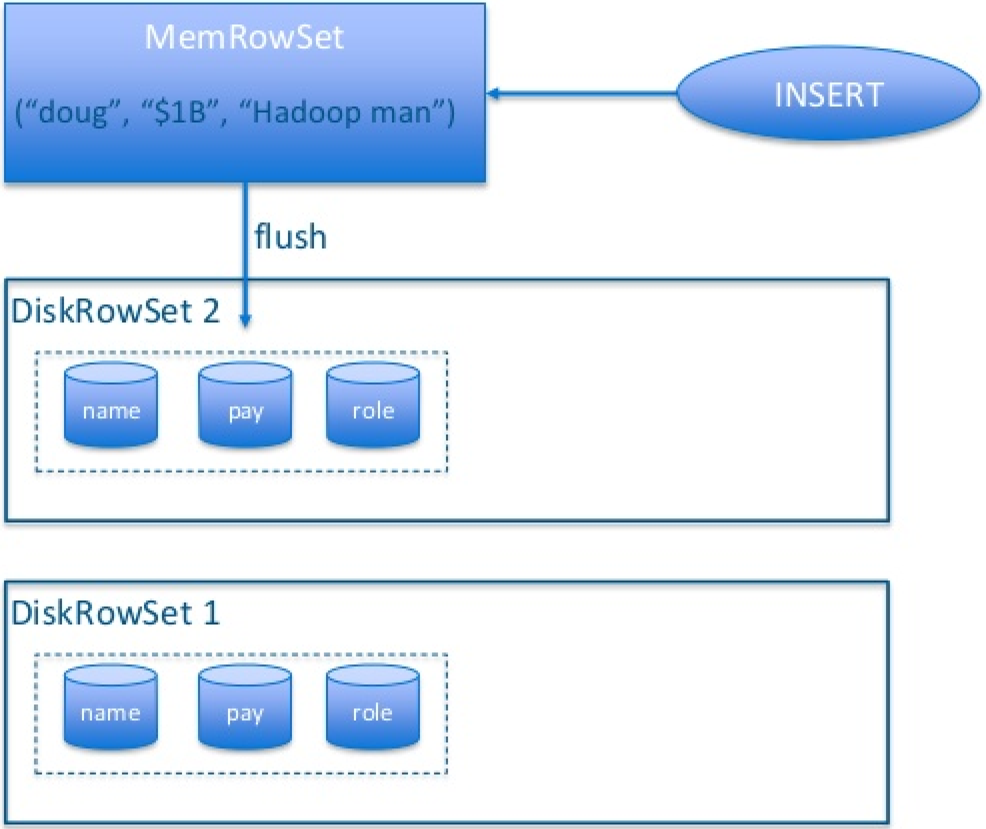
How it works

Write and read paths

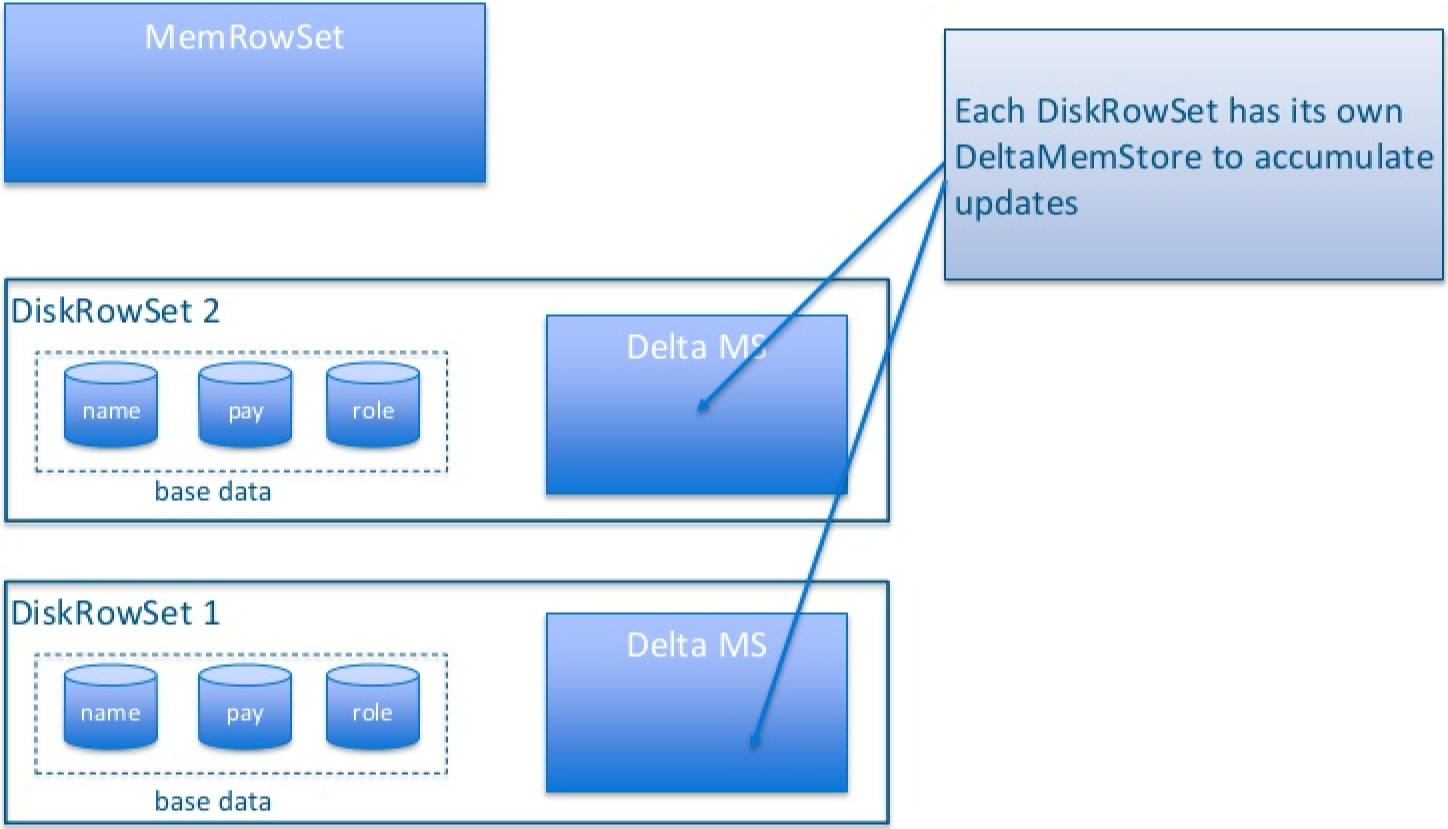
Kudu storage – Inserts and Flushes



Kudu storage – Inserts and Flushes

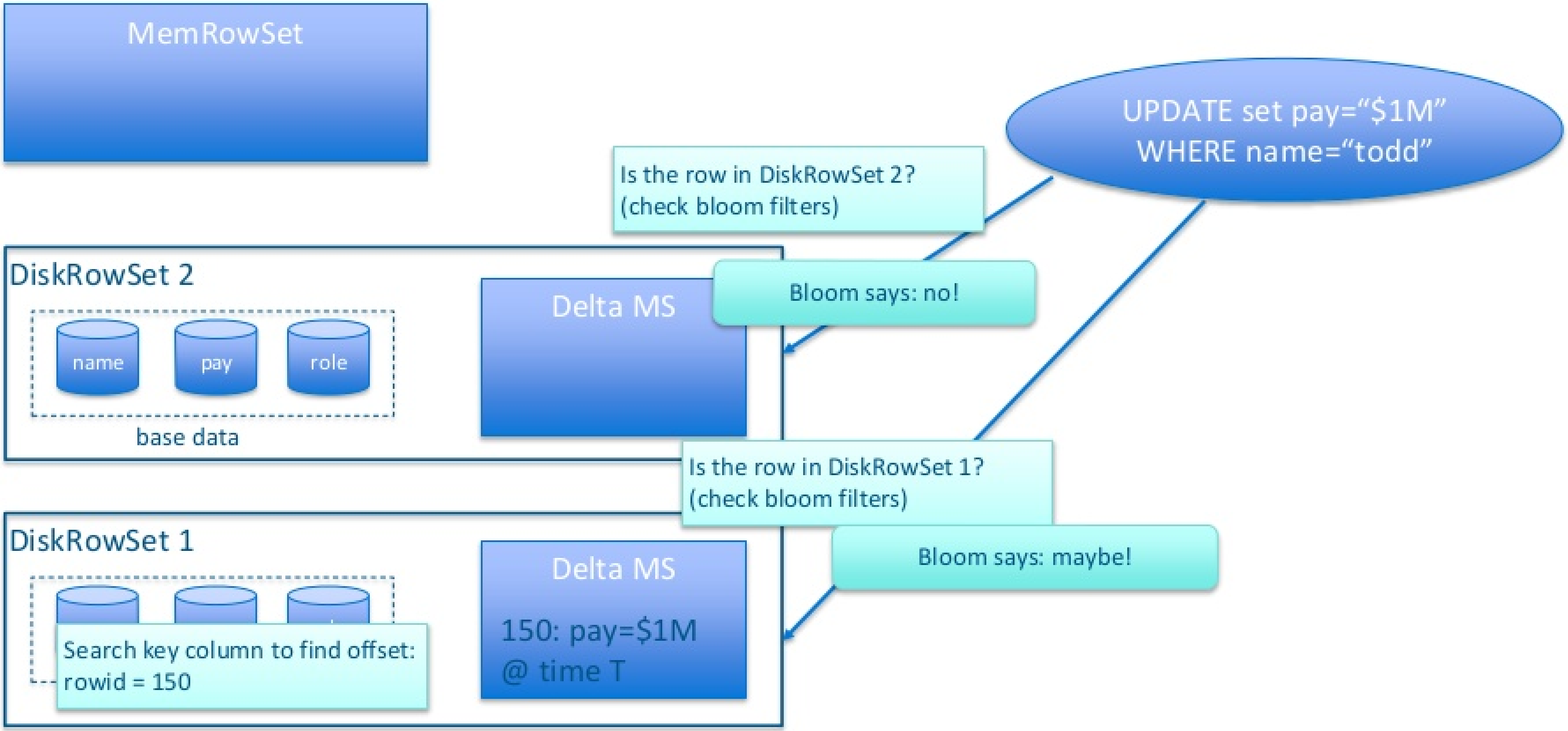


Kudu storage - Updates

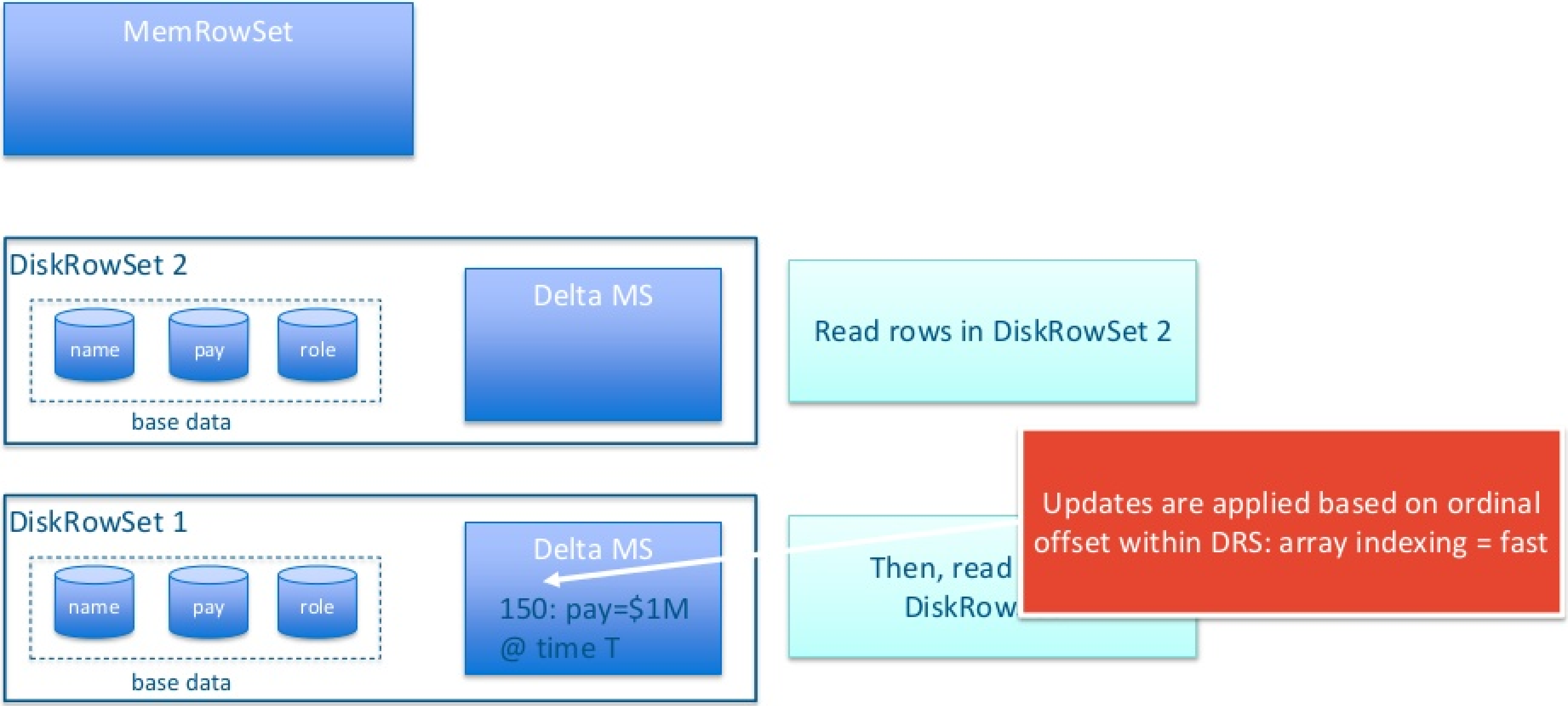


Each DiskRowSet has its own DeltaMemStore to accumulate updates

Kudu storage - Updates



Kudu storage – Read path

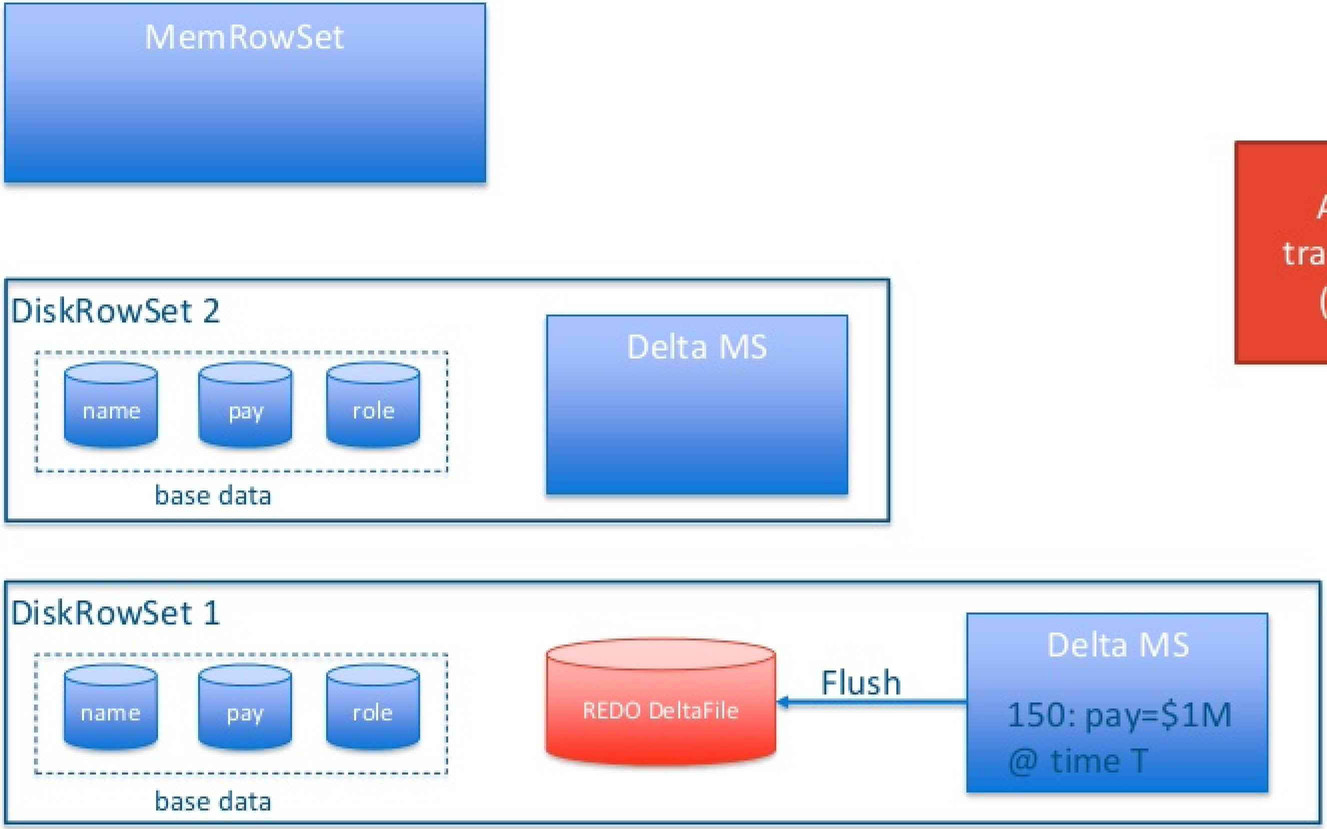


Read rows in DiskRowSet 2

Then, read DiskRow

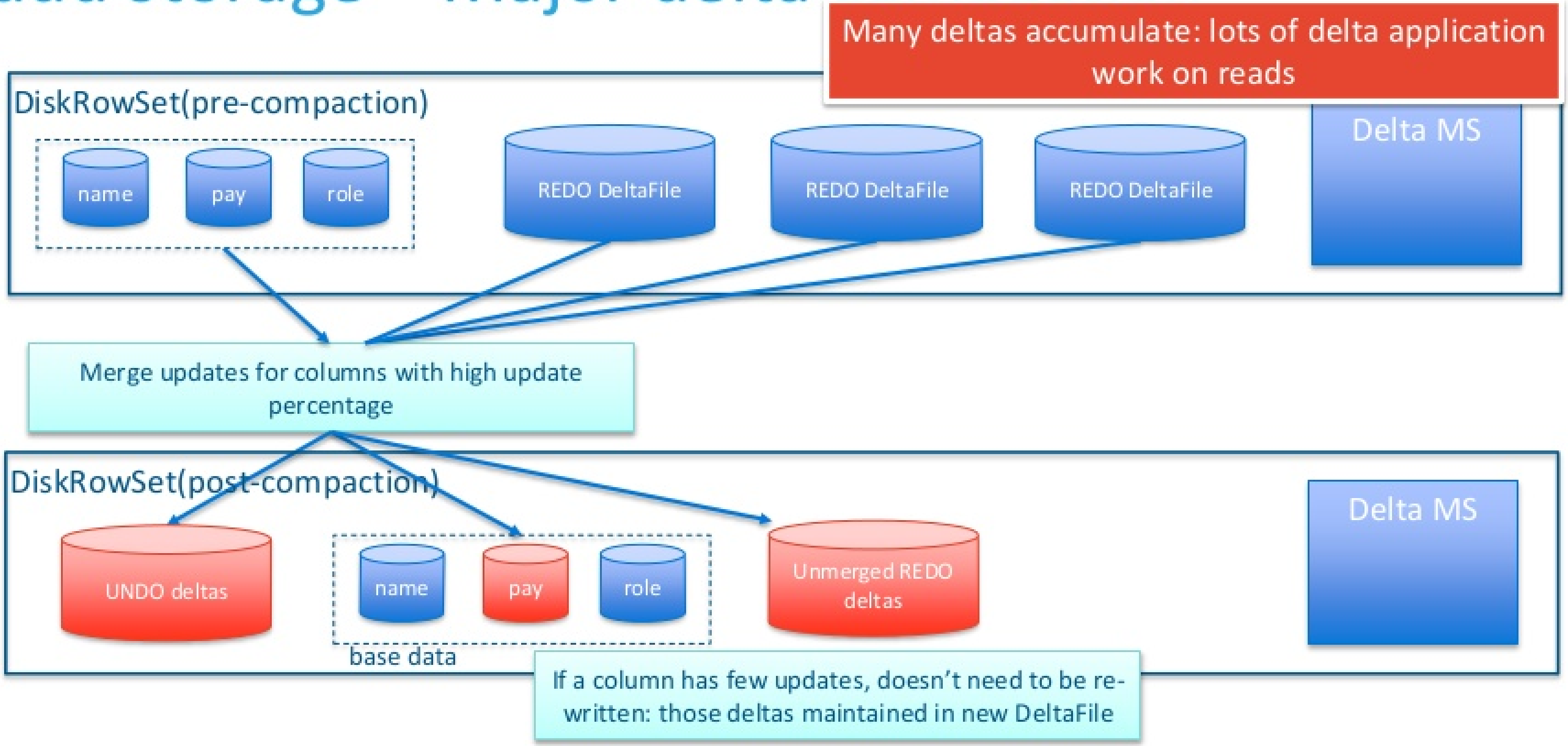
Updates are applied based on ordinal offset within DRS: array indexing = fast

Kudu storage – Delta flushes

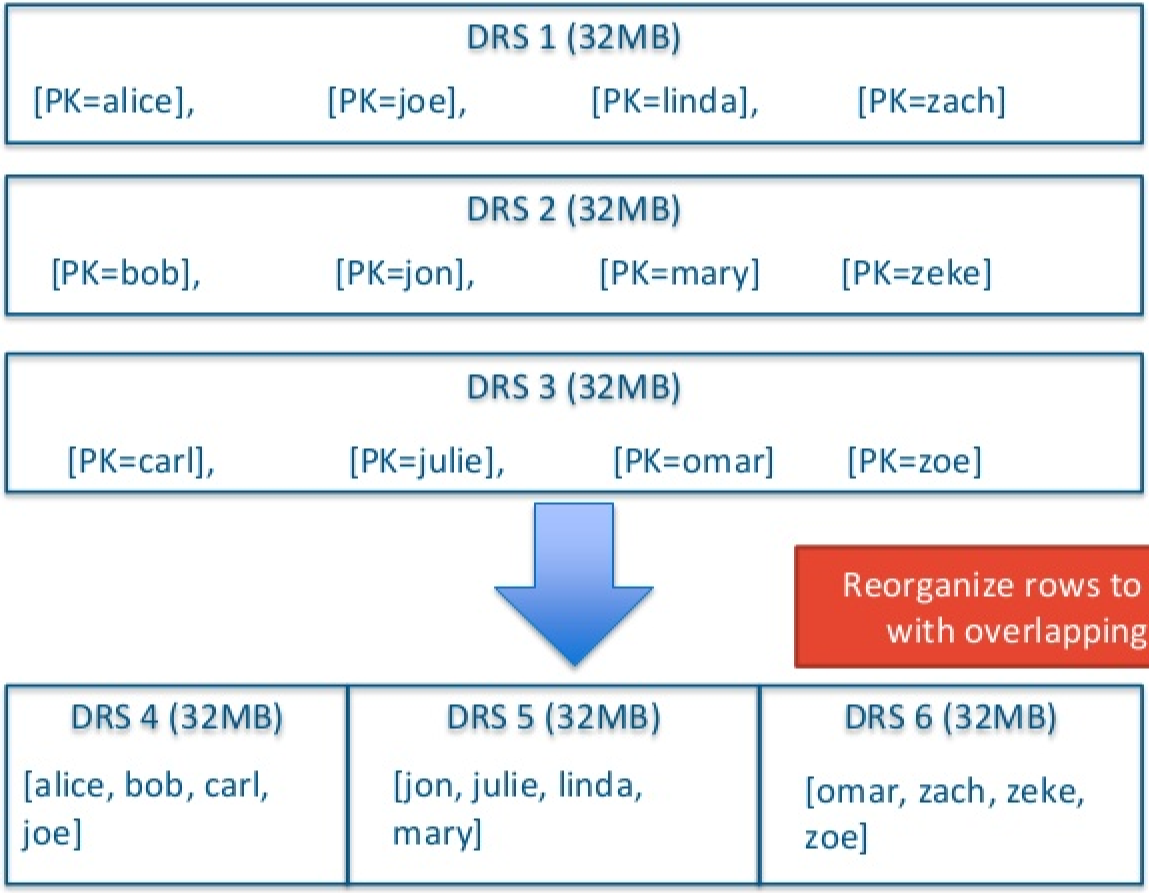


A **REDO** delta indicates how to transform between the 'base data' (columnar) and a later version

Kudu storage – Major delta compaction


Many deltas accumulate: lots of delta application work on reads
DiskRowSet(pre-compaction)
name
pay
role
REDO DeltaFile
REDO DeltaFile
REDO DeltaFile
Delta MS
Merge updates for columns with high update percentage
DiskRowSet(post-compaction)
UNDO deltas
name
pay
role
base data
Unmerged REDO deltas
Delta MS
If a column has few updates, doesn't need to be re-written: those deltas maintained in new DeltaFile

Kudu storage – RowSet Compactions



DRS 1 (32MB)

[PK=alice], [PK=joe], [PK=linda], [PK=zach]

DRS 2 (32MB)

[PK=bob], [PK=jon], [PK=mary] [PK=zeke]

DRS 3 (32MB)

[PK=carl], [PK=julie], [PK=omar] [PK=zoe]

DRS 4 (32MB)

[alice, bob, carl, joe]

DRS 5 (32MB)

[jon, julie, linda, mary]

DRS 6 (32MB)

[omar, zach, zeke, zoe]

Reorganize rows to avoid rowsets with overlapping key ranges