Big Data
- Latest Technologies

Dr. Qing “Matt” Zhang
ITU

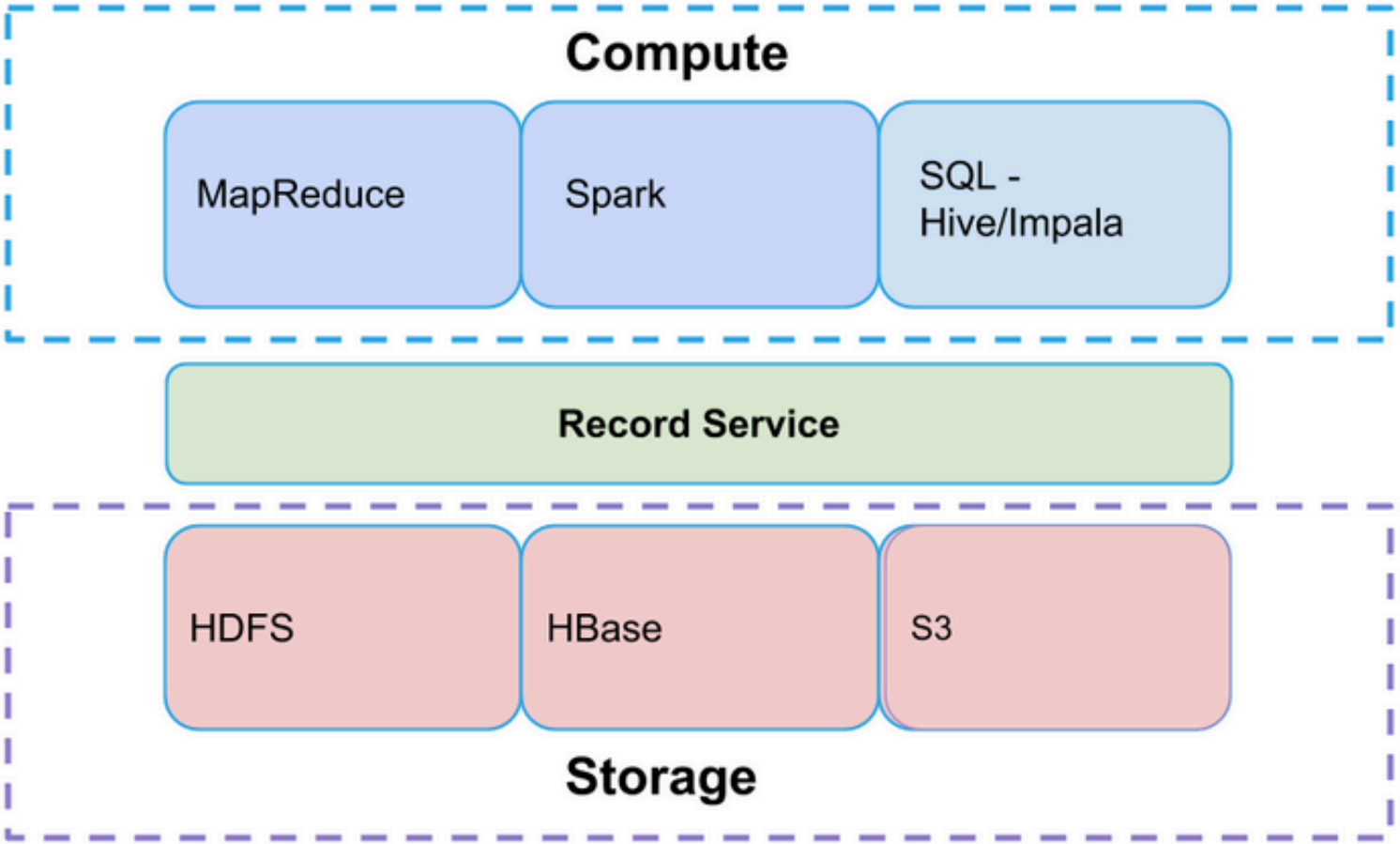
Overview

- **Cloudera Record Service**
- Cloudera Kudu
  - Both announced on 09/28/2015
- Apache Arrow
  - Apache ‘Top Level Project’ from 02/17/2016

Cloudera Record Service

- Motivation
  - Decoupling storage managers (HDFS, HBase) and compute frameworks (MapReduce, Impala, Apache Spark)
    - Greater flexibility to pick the framework that best solves your problem
    - Leads to more complexity to ensure everything works together seamlessly
  - A new core security layer for Hadoop
    - Sits between the storage managers and compute frameworks
    - Provide a unified data access path.

Unified Data Access Path



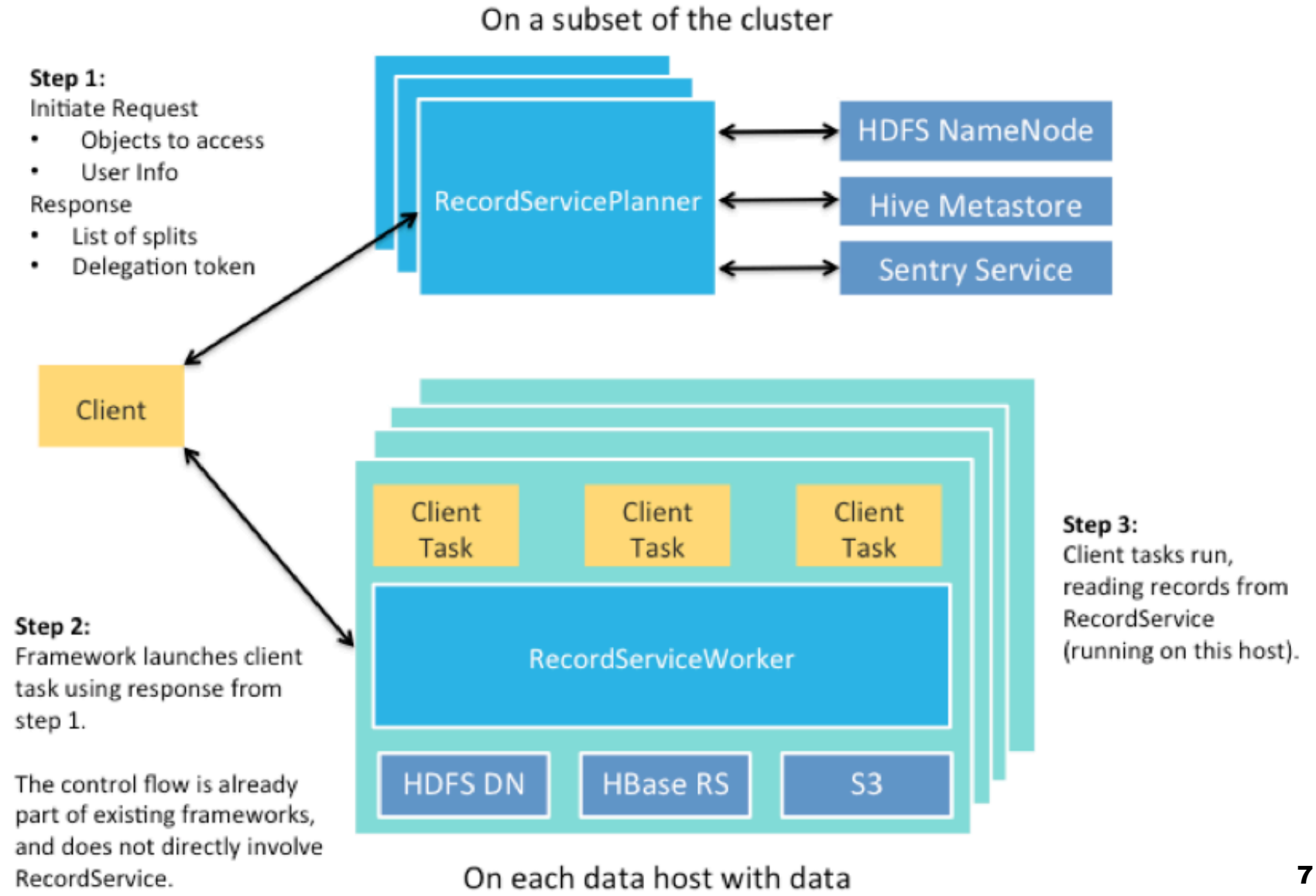
Design

- RecordServicePlanner — Generates tasks, performs authorization checks, handles metadata access.
  - Each task describes a work unit and the preferred locality
  - Called during input split generation.
  - Provides a layer of abstraction over the metadata services (NameNode, Hive Metastore, Sentry server)
- RecordServiceWorker — Executes tasks, reads and writes to the storage layer.
  - Builds on Impala's highly optimized I/O scheduler and file parsers.
  - Returns reconstructed, filtered records in a canonical wire format.
  - Provides a layer of abstraction of the data stores (DataNode)

Design

- Thrift APIs.
- Client Integration Libraries — Allow easy migration to RecordService.
- Planner and Worker share only minimal state via Apache ZooKeeper
  - Scalability
  - Fault tolerance

Architecture



Review: Hadoop Execution

- Simple job execution flow
  - Client contacts the NameNode
    - get the block locations, represented as InputSplits
  - The compute framework (MapReduce/Spark) will launch the map task on the slave nodes.
  - Each slave node read the data (idealy locally) and perform the computation

Hadoop Integration

- Clients talk to the RecordService planner
  - Planners run on a few nodes (default 3)
- Tasks read data from Worker (instead of directly going to the data stores)
  - Workers run on all nodes with data
    - optimize for read locality
- RecordService provides client libraries that implement the common Hadoop InputFormats

Benefits

- Fine-grained data permissions and enforcement across Hadoop ecosystem
  - column-level permissions (projections), row-level permissions (filtering), and data masking
- Performance
  - Sits on the main data access path
  - Scale horizontally
  - Impala IO layer provides low-level optimizations
- Simplicity
  - Provides a higher level, logical abstraction for data
    - Datasets can be specified as logical names (i.e. tables or views)
    - Applications don't need to worry about differences in file formats etc.

Overview

- Cloudera Record Service
- **Cloudera Kudu**
  - Both announced on 09/28/2015
- Apache Arrow
  - Apache ‘Top Level Project’ from 02/17/2016

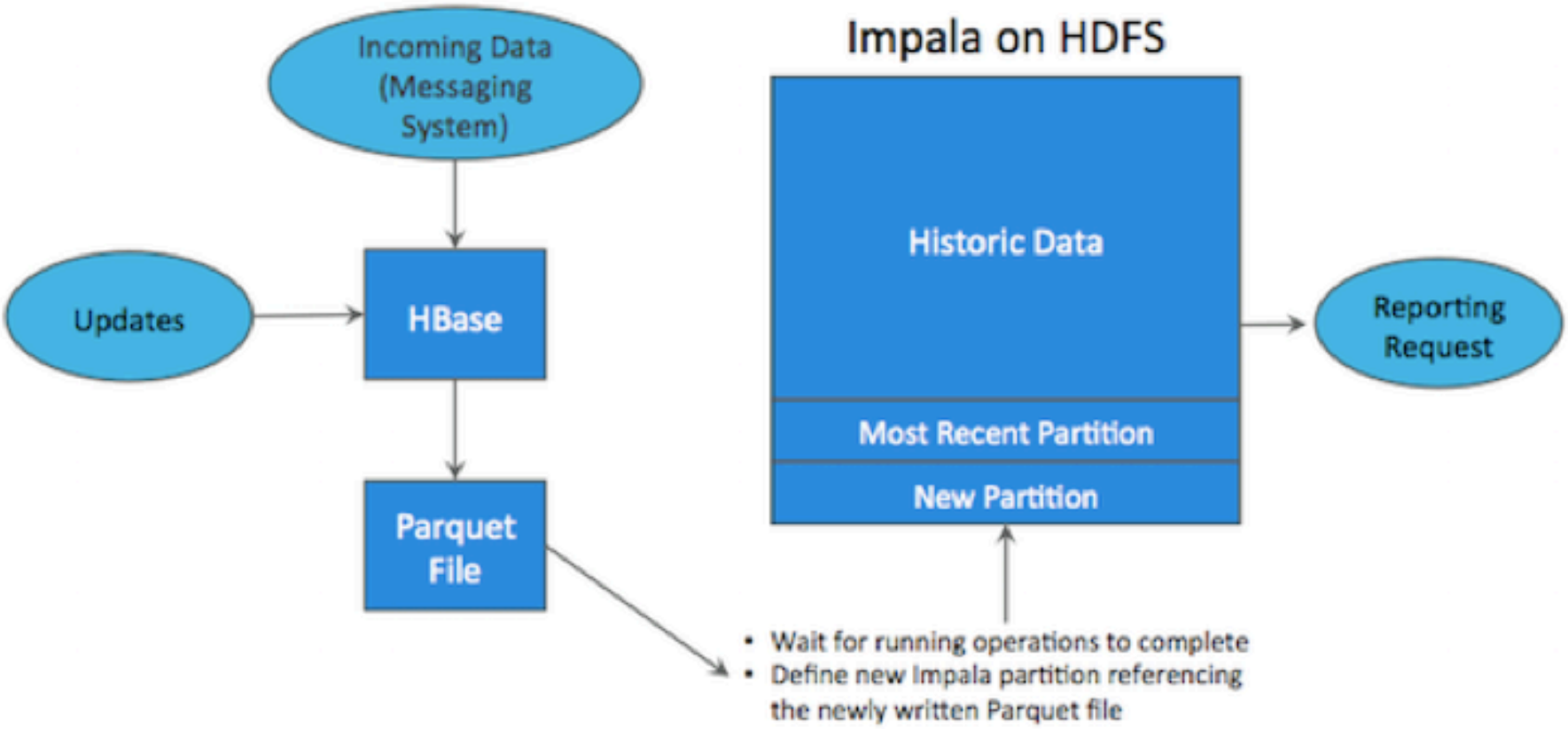
Motivation

- No update on HDFS data
- In practice, may require constant change of fact or dimension tables
- HBase or Cassandra solutions
  - Periodic roll-over to HDFS
  - Complicated workflows
  - Problem with latency and data freshness

Gap in Hadoop Capabilities

- Data storage and processing technologies that define the Apache Hadoop ecosystem are expansive and ever-improving
- Hybrid architectures
  - Enterprise use cases require the simultaneous availability of capabilities
    - Stitch multiple tools together
      - ingest and update data in one storage system
      - reorganize this data to optimize for an analytical reporting use-case served from another

A complex hybrid architecture designed to cover gaps in storage system capabilities



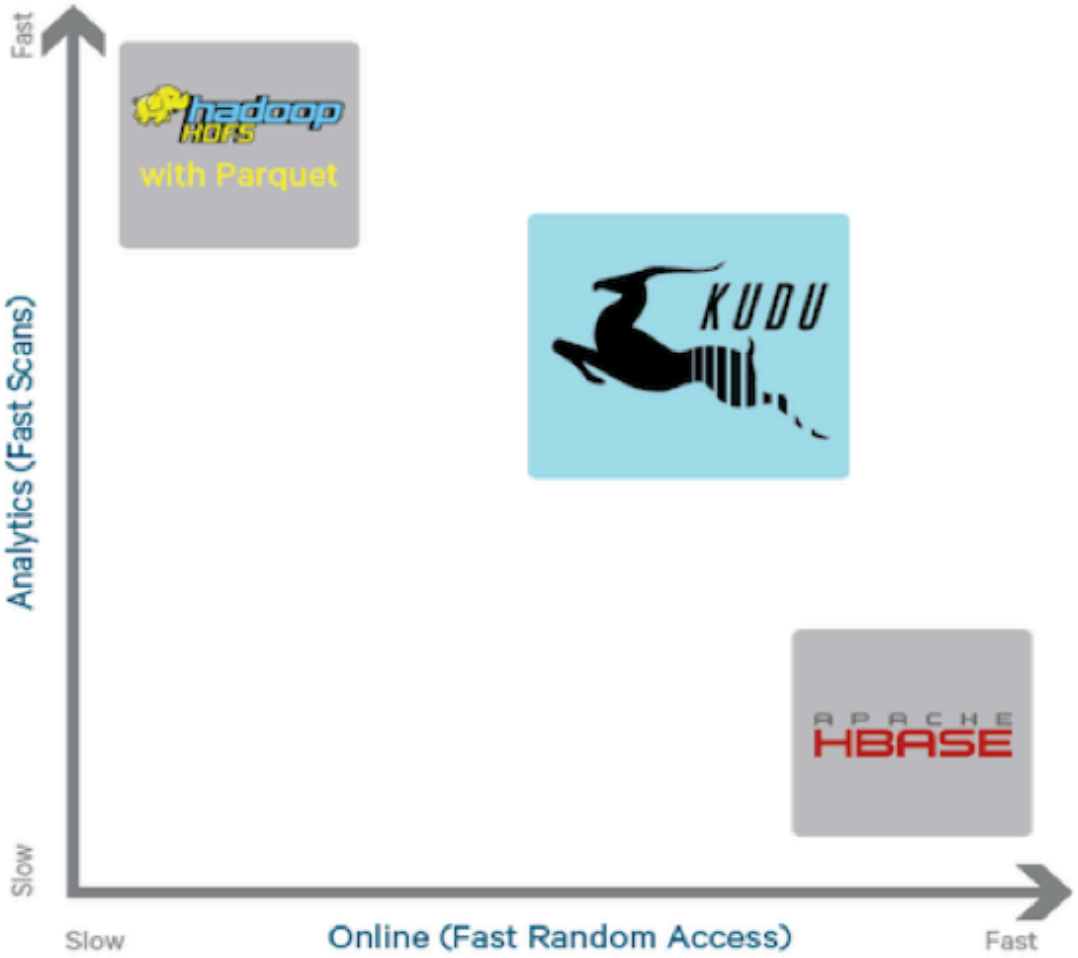
Kudu Properties

- Provide fast analytical and real-time capabilities
- Efficient utilization of modern CPU and I/O resources
- Do updates in place
- Simple and evolvable data model

Benefits

- Fast Data access
  - Nearly as fast as raw HDFS for scans
  - Nearly as fast as HBase for random access
- High CPU performance
  - Single-column scan rate 10-100x faster than HBase
  - Needed to take advantage of RAM and Flash
- High IO efficiency
  - True column store with type-specific encodings
  - Allow efficient analytics when only certain columns are accessed

Fast Data Access



Basic Design

- A storage system for tables of structured data in user view
  - well-defined schema
  - predefined number of typed columns
  - primary key composed of one or more of its columns
- Tables are composed of tablets/partitions
  - Replicated
  - Tablets are typically tens of gigabytes, and an individual node typically holds 10-100 Tablets.

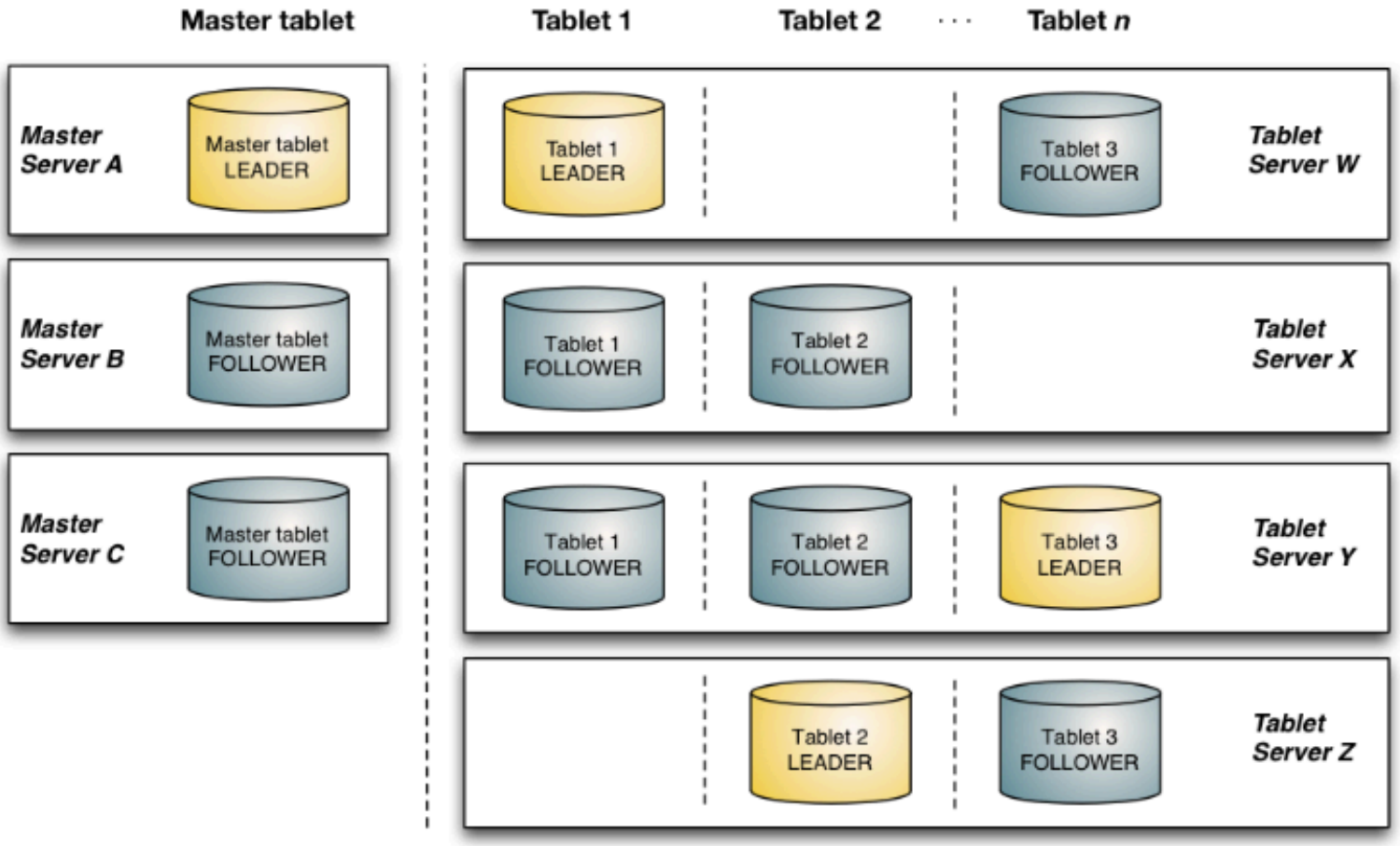
Basic Design (cont'd)

- Master process responsible for managing the metadata
  - Describes the logical structure of the data stored in Tablet Servers
  - Acts as a coordinator when recovering from hardware failure
  - Keeps track of which tablet servers are responsible for hosting replicas of each Tablet.
  - Multiple standby master servers can be defined to provide high availability.

Basic Design (cont'd)

- Log-structured storage
  - Updates, inserts, and deletes are temporarily buffered in memory before being merged into persistent columnar storage
  - Constantly performing small maintenance operations such as compactions
    - Protect against spikes in query latency
- Direct APIs in C++/Java
  - allow for point and batch retrieval of rows
  - writes, deletes, schema changes

Netowork Architecture



Architecture

- Cluster roles
  - Single master server: metadata
    - Replicated for fault tolerance
  - Multiple tablet server: data
- Partitioning
  - Tables are horizontally partitioned into tablets
  - Partition schema is made up of zero or more hash-partitioning rules followed by an optional range-partitioning rule
    - Trade off between query parallelism and query concurrency
- Replication
  - Raft Consensus algorithm
    - Leader replica handles replication details

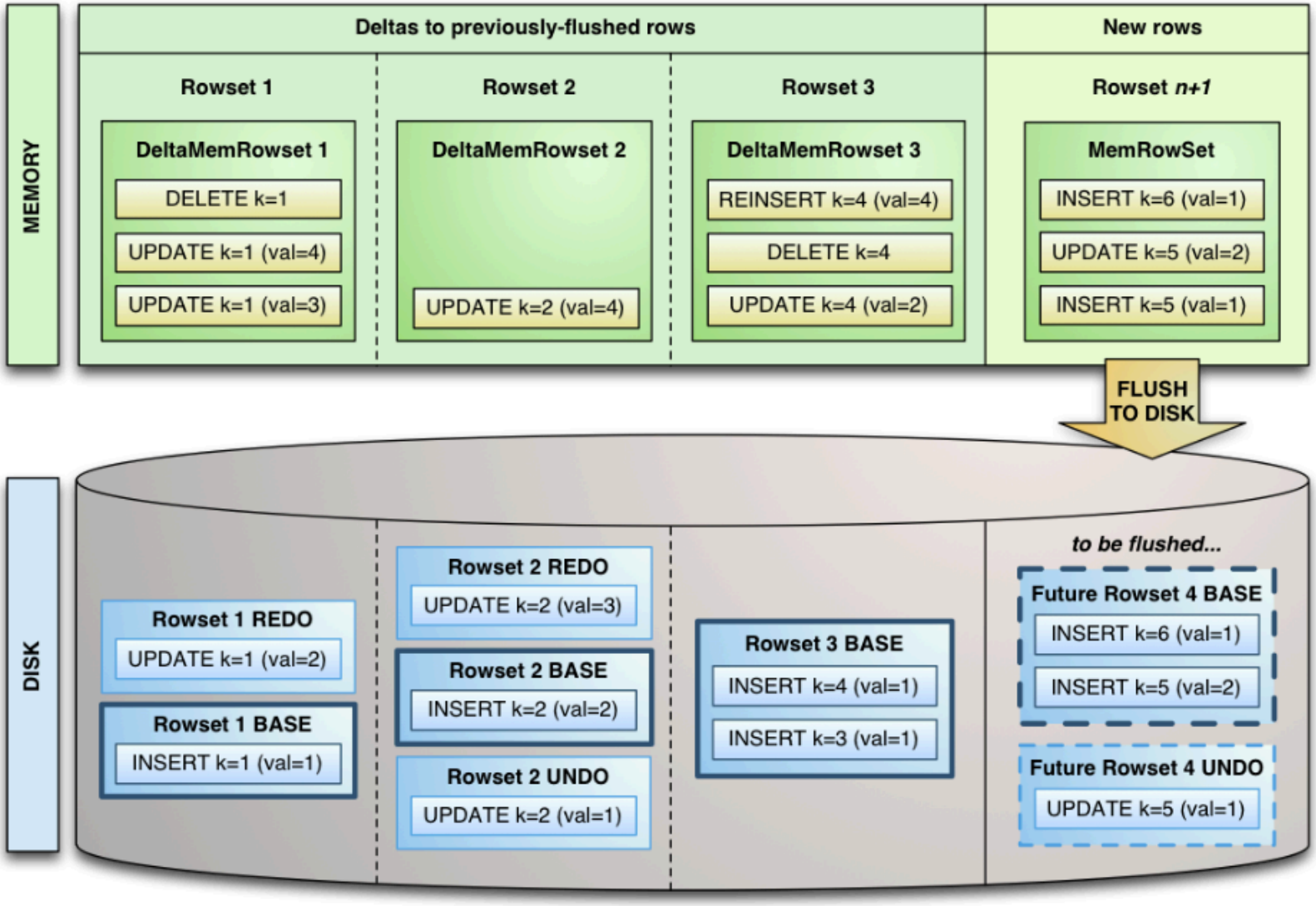
Kudu Master

- Catalog Manager
  - Single tablet table for catalog information
    - Current version of the table schema
    - State of the table (creating, running, deleting etc.)
    - Set of tablets for this table
  - A full write-through cache of the catalog is kept in memory

Cluster Co-ordination

- Each tablet server is configured with a list of host names
- On startup, tablet server register with Master and sends report
  - Future reports are incremental
- Tablet servers are responsible for all replica configuration
  - Raft Leader replica decides replication
  - Master is only observer
    - Suggests configuration change to lead replica of the tablet, on replica count change
- Clients query the Master for tablet location information

Data Model: single-tablet view



Tablet Storage

- Tablets are subdivided into RowSets
  - MemRowSets
    - Stores all recently inserted rows
    - flushed to disk periodically
  - DiskRowSets
    - base data
      - Column representation of rows
      - Immutable once flushed
    - delta stores
      - Updates and deletes
      - In-memory *DeltaMemStores*, or on-disk *DeltaFiles*
    - delta flush
      - Flushing a DeltaMemStore, a new empty store is swapped in while the existing one is written to disk and becomes a DeltaFile.

Data access

- Read
  - Seeks the target column to the correct row offset
  - Copies cells from the source column into output row batch
  - Consult the delta stores to see if any later updates have replaced cells with newer versions
- Insert
  - Need to consult all DiskRowSets before save to MemRowSets, to enforce primary key uniqueness constraints
    - Unlike many other NoSQL stores, where INSERT = UPSERT
    - Achieved by bloom filter on keys, and key range partition for each DiskRowSet
- Lazy materialization
  - Read columns which have associated range predicates before reading any other columns.

Maintenance

- Delta compaction
  - Where there're large number of deltas, merge back into the base data columns
- RowSet compaction
  - Merge two or more DiskRowSets into a sorted stream of output rows, written back to a new DiskRowSets
    - Remove deleted
    - Reduce key range overlap
- Scheduling maintenance
  - A pool of maintenance threads within the tablet server, running all the time
  - Run small units of flush/compaction work, adapt to workload

Overview

- Cloudera Record Service
- Cloudera Kudu
  - Both announced on 09/28/2015
- **Apache Arrow**
  - Apache ‘Top Level Project’ from 02/17/2016

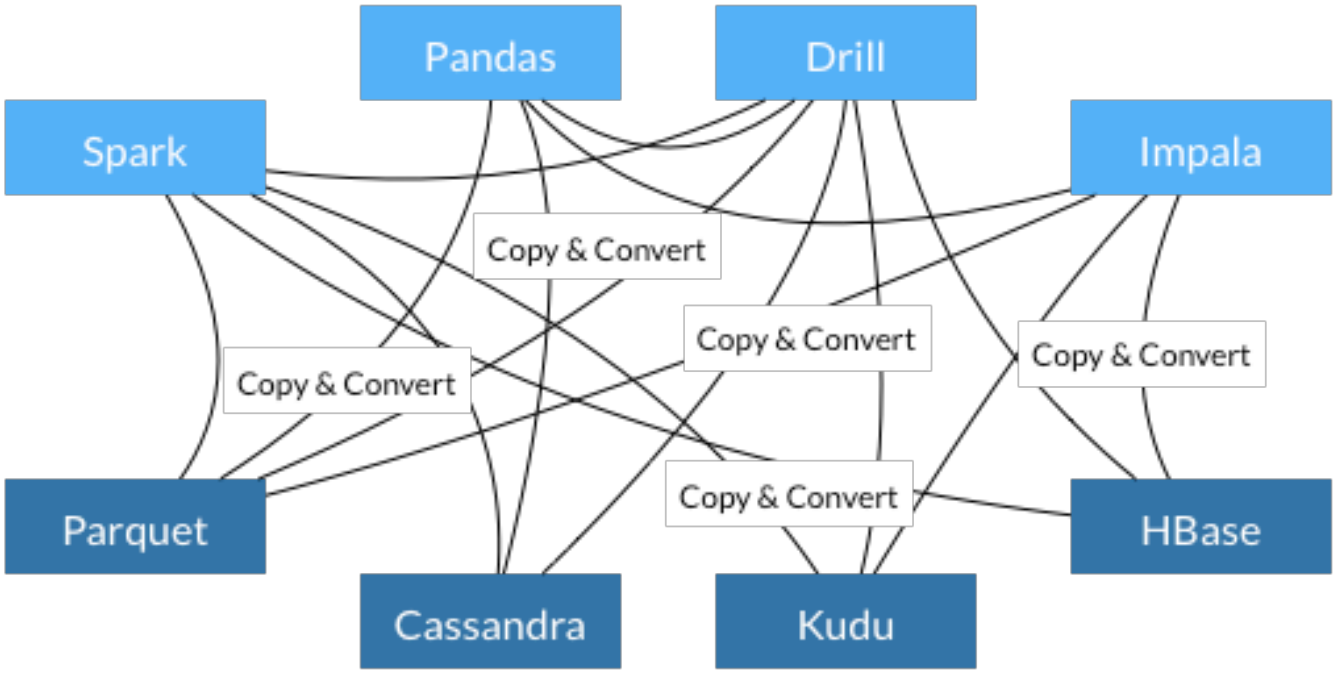
Background

- In-memory data systems
- Columnar storage
- Common approach, varied implementation

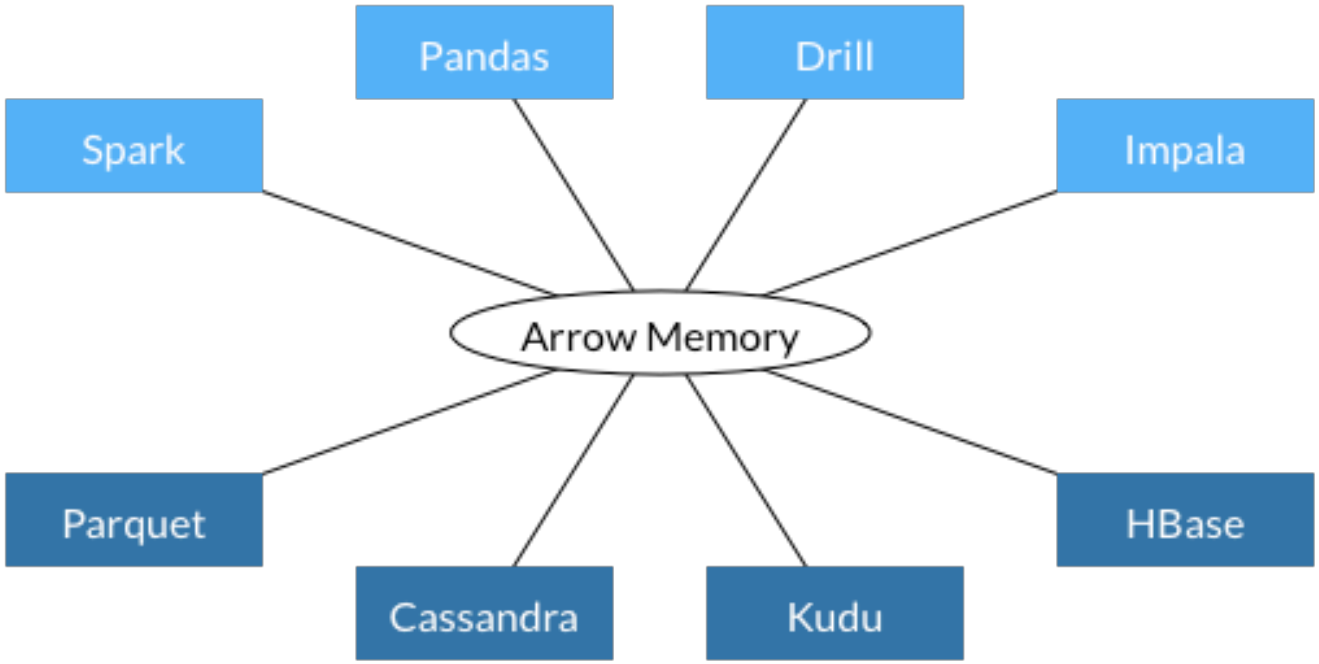
Key Benefits

- Columnar memory-layout, O(1) random access
- Standard data interchange format
  - Sharing of data in a seamless and frictionless fashion between systems and processes.
  - Reduce the large amount of CPU cycles serializing and de-serializing data, to convert data between various formats
  - customers are able to deploy these systems in conjunction with each other without incurring any overhead
- A flexible structured data model supporting complex types like flat tables, json etc

Without Arrow



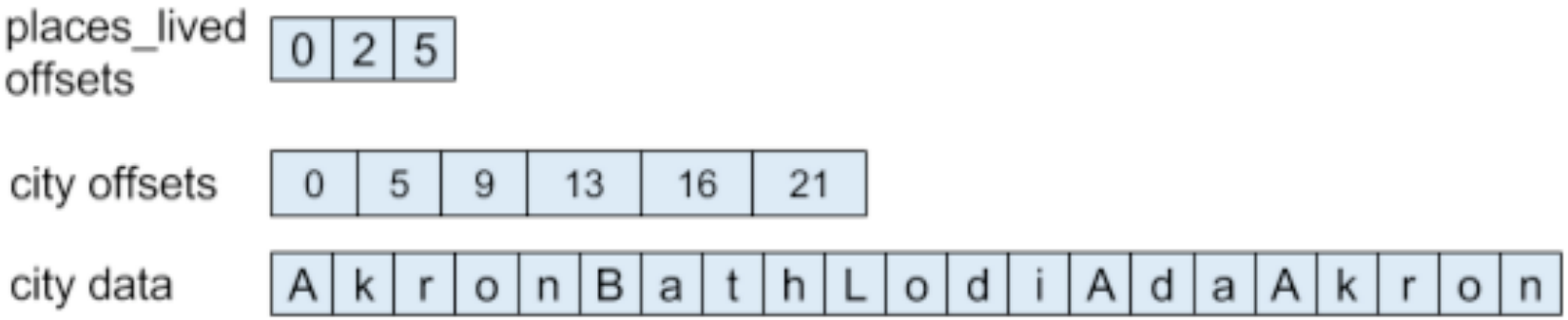
With Arrow


Pandas
Drill
Spark
Impala
Arrow Memory
Parquet
HBase
Cassandra
Kudu

What Arrow array looks like

```
people = [
 {
   name: 'mary', age: 30,
   places_lived: [
     {city: 'Akron', state: 'OH'},
     {city: 'Bath', state: OH'}
   ]
 },
 {
   name: 'mark', age: 33,
   places_lived: [
     {city: 'Lodi', state: 'OH'},
     {city: 'Ada', state: 'OH'},
     {city: 'Akron', state: 'OH}
   ]
 }
]
```

Look at the people.places_lived.city values:



- Thank you!