FLINK-KUDU CONNECTOR:

An open-source contribution to develop Kappa Architectures

Rubén Casado Tejedor

ruben.casado.tejedor@accenture.com

ruben_casado

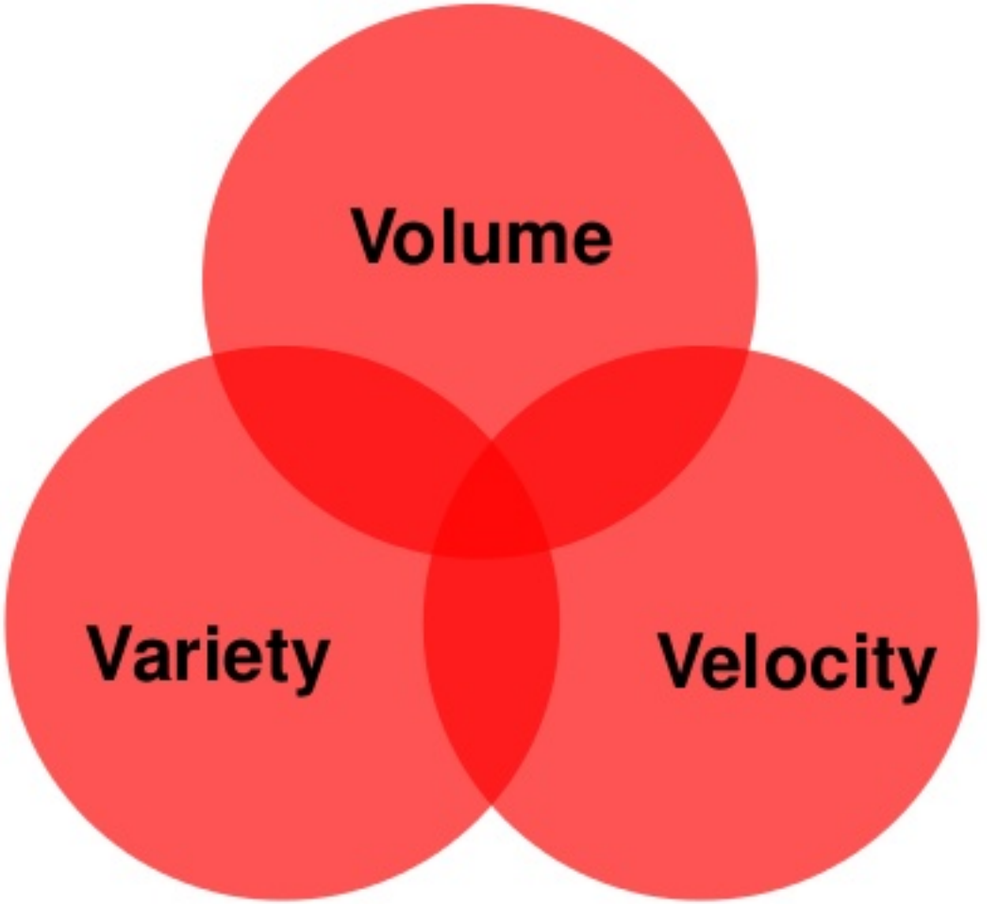
accenture

AGENDA

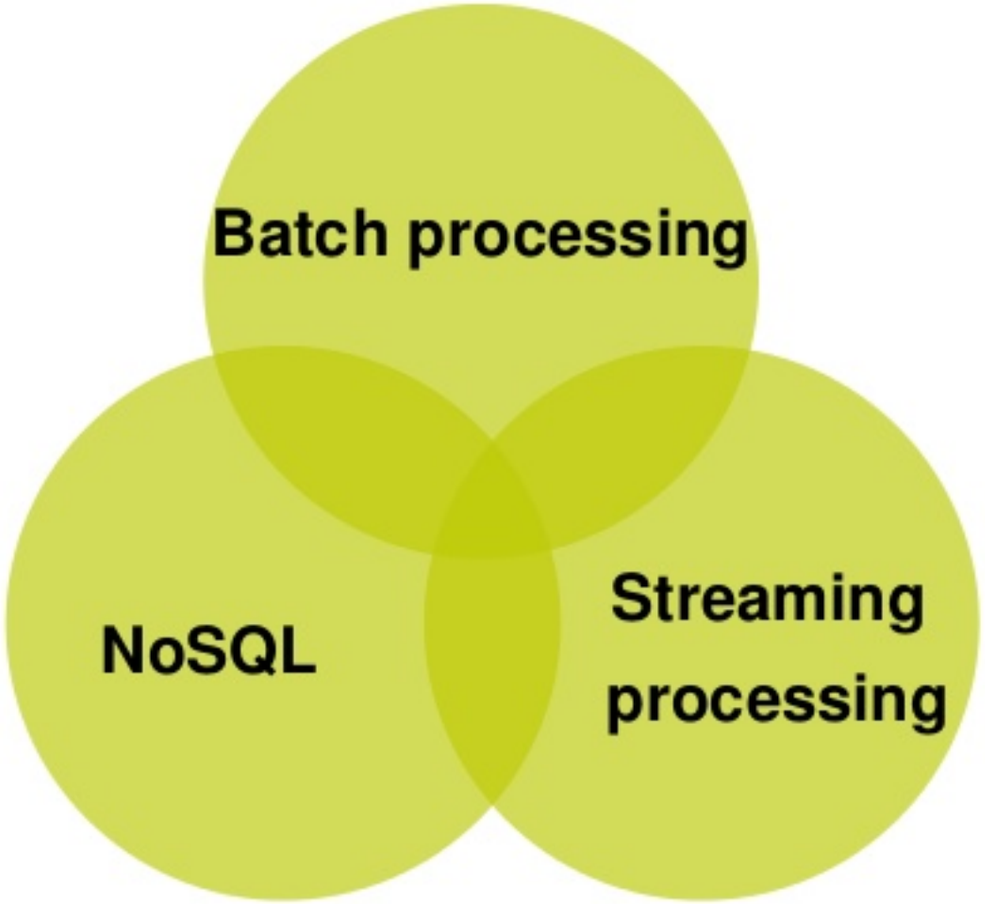
1. **Introduction**
   - Motivation
   - Kappa Architecture
   - Kudu
   - Uses cases
2. **Implementation**
   - Batch Source: KuduInputformat
   - Batch Sink: KuduOutputFormat
   - Stream Sink: KuduSink
3. **Examples**
   - Demo
4. **Contribution to open source**
   - Github
   - Apache Bahir

INTRODUCTION

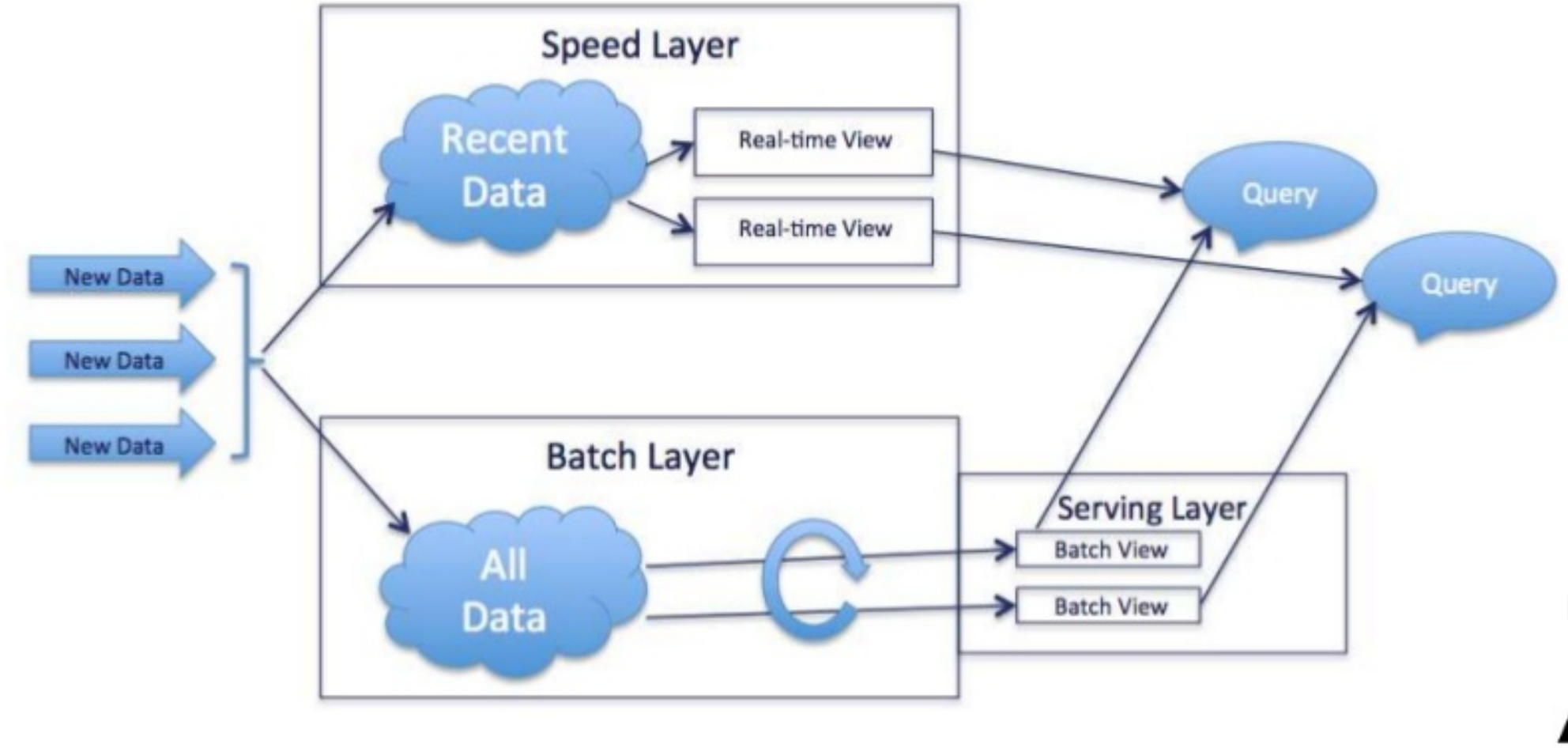
MOTIVATION



MOTIVATION

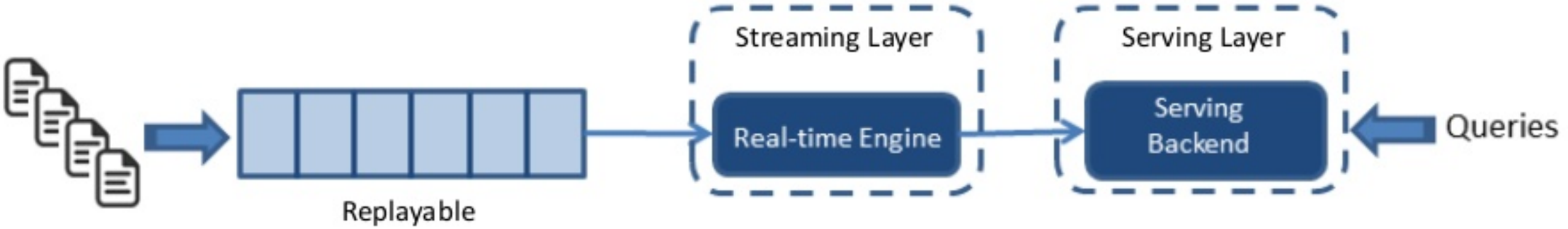


LAMBDA ARCHITECTURE



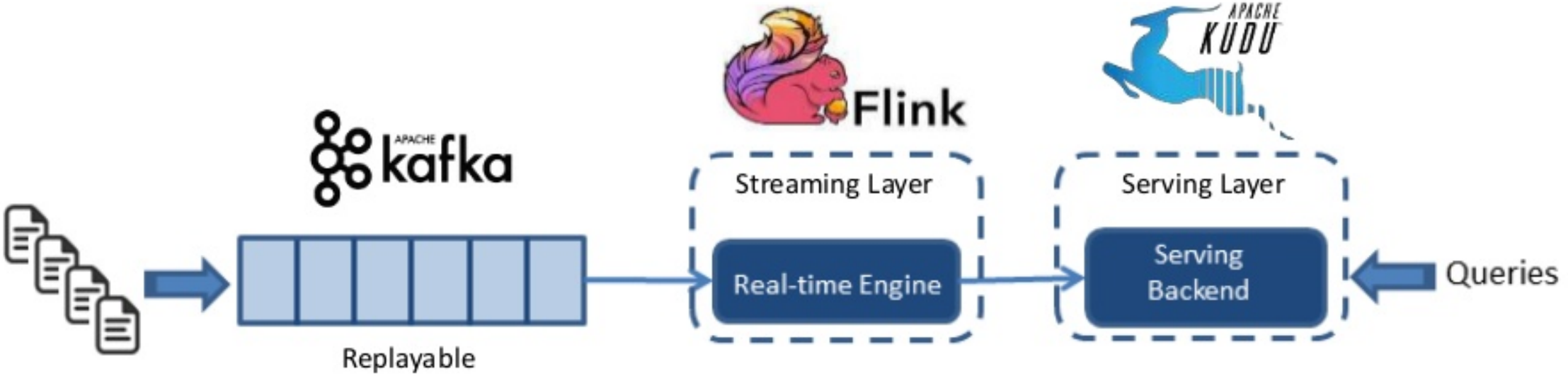
λ

KAPPA ARCHITECTURE



K

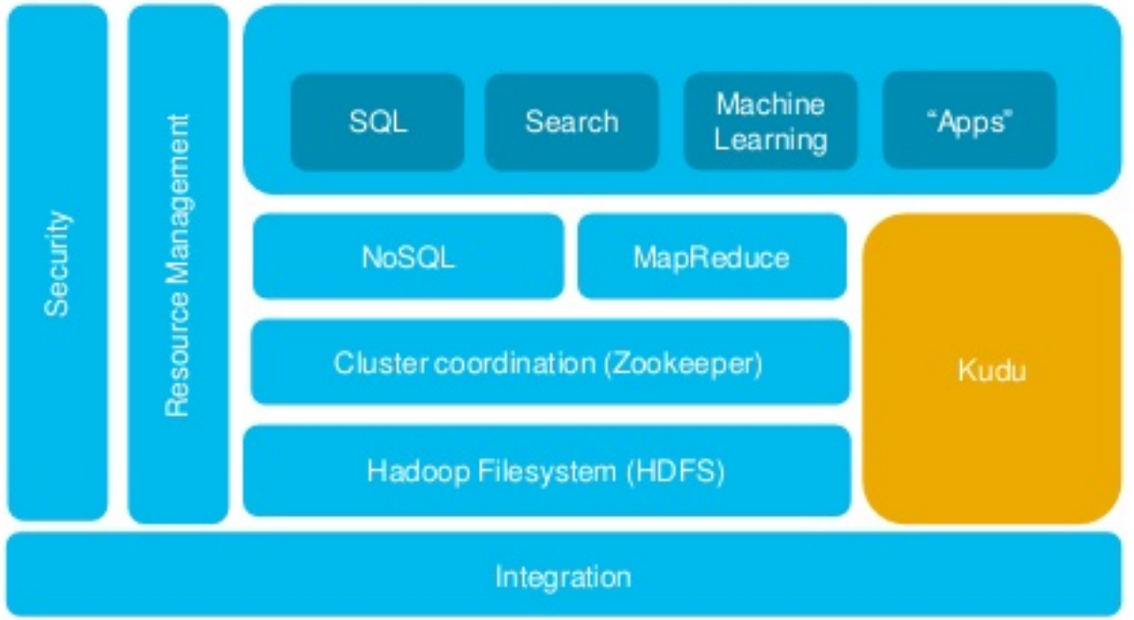
KAPPA ARCHITECTURE



K

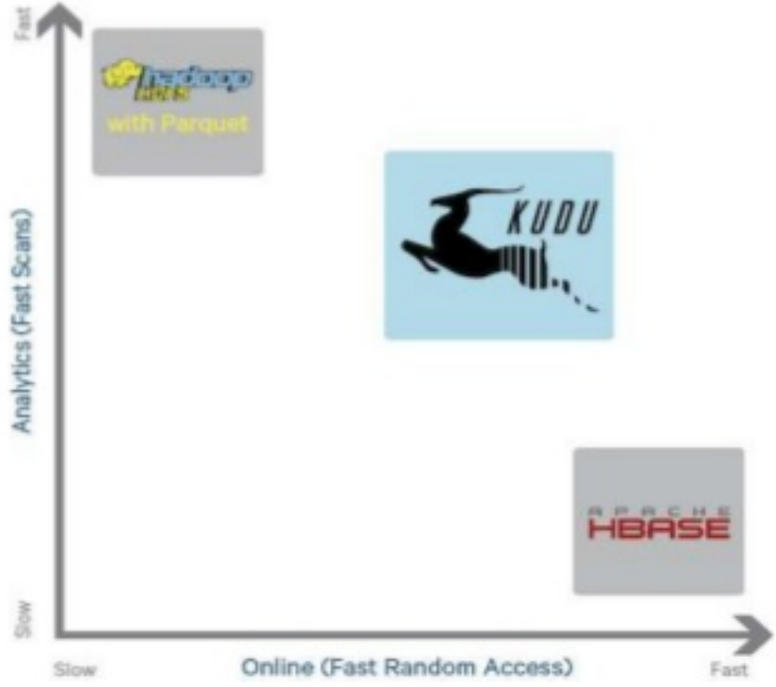
WHAT IS APACHE KUDU?

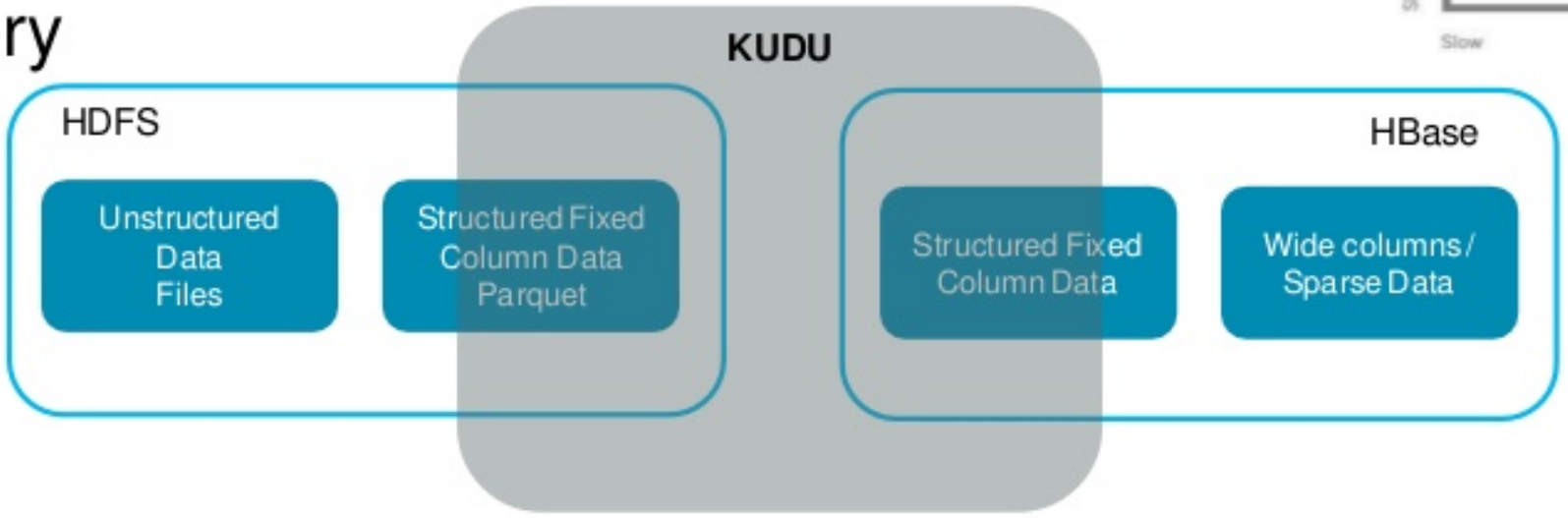
A new **storage system** for **structured data** that provides a combination of **fast inserts/updates** and **efficient columnar scans** to enable multiple real-time analytic workloads across a single layer



WHY KUDU?

HDFS is good for reading/writing **large** amount of data.
HBase is good for **low latency** read and writes.
Kudu complements HDFS and HBase by providing **fast scans** as well as **fast access** to records presenting a **relational data model** with SQL query





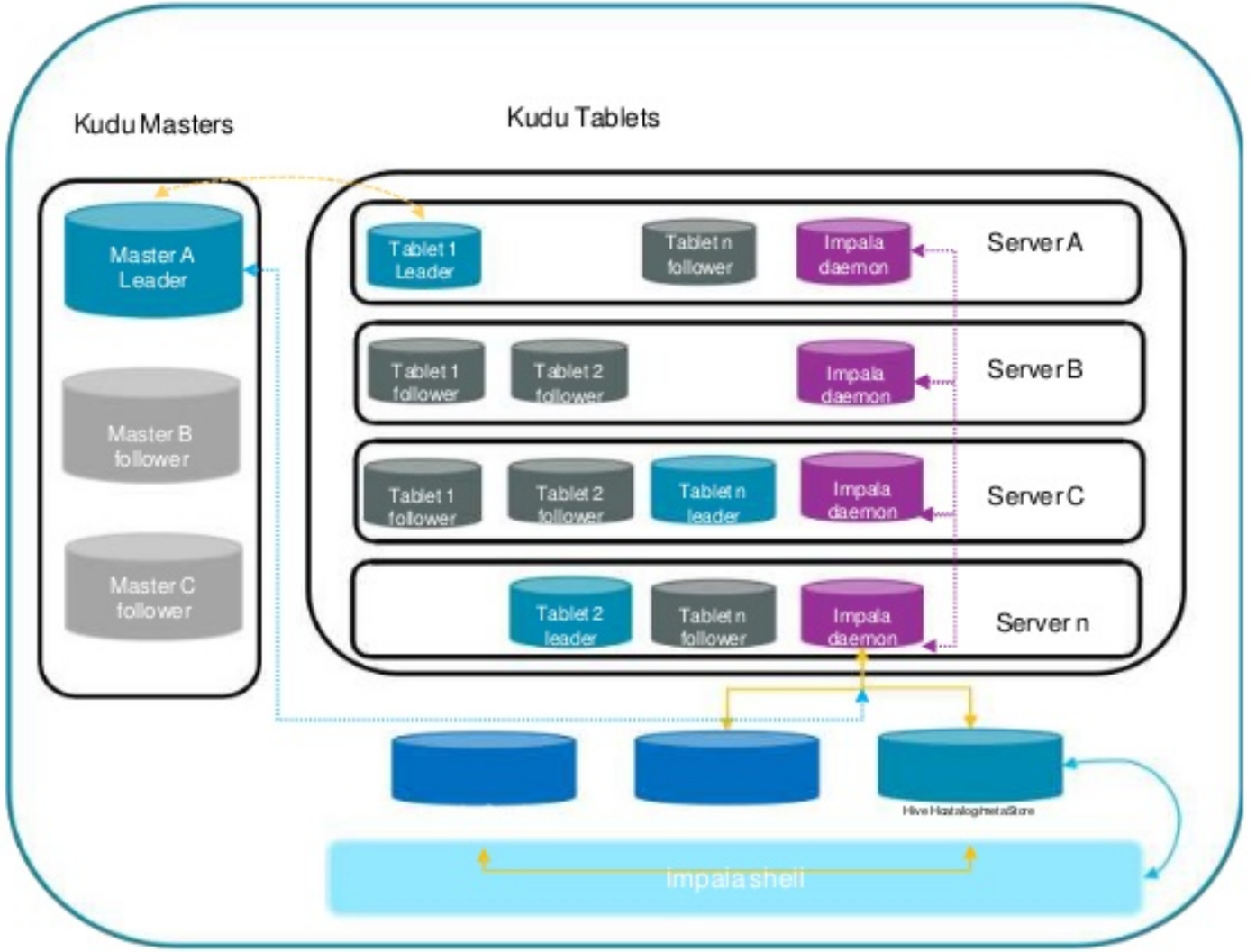
HOW IS KUDU?

Features

- Fixed Schema
- Typed columns
- Allows insert/update/deletes
- Allows table alter

• Architecture

- Kudu Master
- Kudu Table
- Tablets



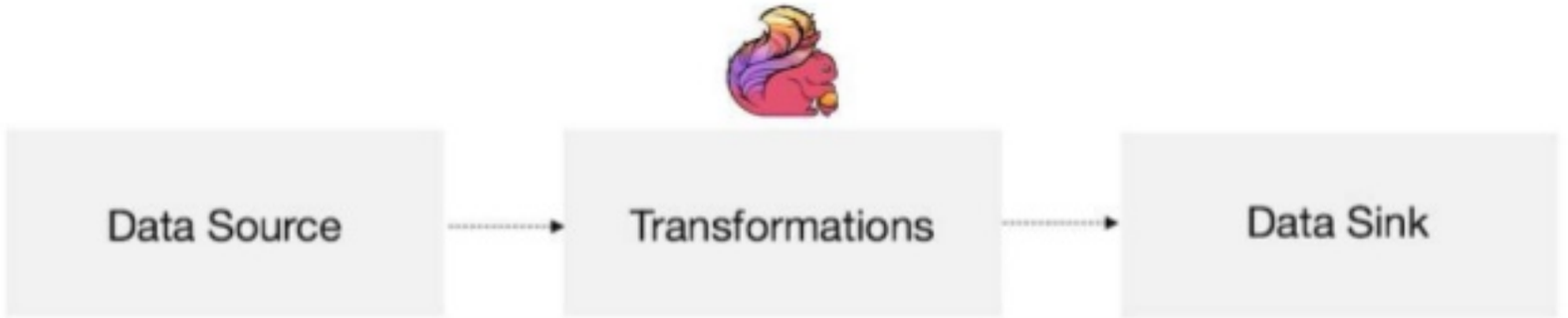
IMPLEMENTATION

SCOPE

Connectors for **Data Source** and **Data Sink** Flink's APIs.

Three connectors:

- Source
  [Batch] DataSet Source **KuduInputFormat**
- Sink
  [Batch] DataSet Sink **KuduOutputFormat**
  [Streams] DataStream Sink **KuduSink**



BATCH SOURCE: *KUDUINTPUTFORMAT*

1. Create custom class which implements Flink's InputFormat<OT,T>.
2. Implement interface methods:
   - open
   - createInputSplits
   - nextRecord
3. Usage

   ExecutionEnvironment env = …;

   DataSet<RowSerializable> source = ***KuduInputBuilder.build(****env, TABLE_NAME, KUDU_MASTER)*

BATCH SOURCE: *KUDUINTPUTFORMAT*

open(KuduInputSplit split)

```
@Override
public void open(KuduInputSplit split) throws IOException {
    if (table == null) {
        throw new IOException("The Kudu table has not been opened!");
    }
    KuduScanToken.KuduScanTokenBuilder builder =
            client.newScanTokenBuilder(this.table)
            .setProjectedColumnNames(this.projectColumns);

    this.tokens = builder.build();
    endReached = false;
    scannedRows = 0;

    try {
        LOG.info("SPLIT NUMBER " + split.getSplitNumber());
        scanner = tokens.get(split.getSplitNumber())
                .intoScanner(client);
    } catch (Exception e) {
        e.printStackTrace();
    }
    results = scanner.nextRows();
}
```

BATCH SOURCE: *KUDUINTPUTFORMAT*

createInputSplits(int minNumSplits)

```
@Override
public KuduInputSplit[] createInputSplits(final int minNumSplits) {
    LOG.info("3. CREATE SPLITS");

    KuduScanToken.KuduScanTokenBuilder builder = client.newScanTokenBuilder(this.table)
            .setProjectedColumnNames(this.projectColumns);

    this.tokens = builder.build();
    List<KuduInputSplit> splits = new ArrayList<>(minNumSplits);

    for (KuduScanToken token : tokens){
        byte[] startKey = token.getTablet().getPartition().getPartitionKeyStart();
        byte[] endKey = token.getTablet().getPartition().getPartitionKeyEnd();

        List<String> locations =
                new ArrayList<>(token.getTablet().getReplicas().size());
        for (LocatedTablet.Replica replica : token.getTablet().getReplicas()) {
            locations.add(replica.getRpcHost()
                    .concat(":")
                    .concat(replica.getRpcPort().toString()));
        }
        int numSplit = splits.size();
        KuduInputSplit split = new KuduInputSplit(numSplit, (locations.toArray(
                new String[locations.size()])), TABLE_NAME, startKey, endKey);
        splits.add(split);
    }
    LOG.info("Created: " + splits.size() + " splits");
    return splits.toArray(new KuduInputSplit[0]);
}
```

BATCH SOURCE: *KUDUINTPUTFORMAT*

nextRecord (RowSerializable reuse)

```
@Override
public RowSerializable nextRecord(RowSerializable reuse) throws IOException {
    if (scanner == null) {
        throw new IOException("No table scanner provided!");
    }
    if (reuse == null){
        throw new IOException("No row reuse provided");
    }
    if (results.getNumRows()==0){
        throw new IOException("The table is empty");
    }
    try {
        RowResult res = this.results.next();
        RowSerializable resRow= RowResultToRowSerializable(res);
        if (res != null) {
            scannedRows++;
            return resRow;
        }
    } catch (Exception e) {
        endReached = true;
        scanner.close();
        //workaround for timeout on scan
        LOG.warn( message: "Error after scan of " + scannedRows +
                " rows. Retry with a new scanner...", e);
    }
    return null;
}
```

BATCH SINK: *KUDUOUTPUTFORMAT*

1. Create custom class which extends Flink's **RichOutputFormat**.

2. Implement methods of superclass. Most important are:
   - open
   - writeRecord

3. Usage

   DataSet<RowSerializable> result= …..

   *result.**output**(new **KuduOutputFormat**(KUDU_MASTER, TABLE_NAME, columnNames, MODE));*

BATCH SINK: *KUDUOUTPUTFORMAT*

open (int i, int il)

```java
@Override
public void open(int i, int i1) throws IOException {
    // Establish connection with Kudu
    this.utils = new Utils(host);
    if(this.utils.getClient().tableExists(tableName)){
        logger.info("Mode is CREATE and table already exist. " +
                "Changed mode to APPEND. " +
                "Warning, parallelism may be less efficient");
        tableMode = APPEND;
    }
    // Case APPEND (or OVERRIDE), with builder without column names,
    // because otherwise it throws a NullPointerException
    if(tableMode.equals(APPEND) || tableMode.equals(OVERRIDE)) {
        this.table = utils.useTable(tableName, tableMode);

        if (fieldsNames == null || fieldsNames.length == 0) {
            fieldsNames = utils.getNamesOfColumns(table);
        } else {
            // When column names provided, and table exists,
            // must check if column names match
            utils.checkNamesOfColumns(utils.getNamesOfColumns(this.table),
                    fieldsNames);
        }
    }
}
```

BATCH SINK: *KUDUOUTPUTFORMAT*

writeRecord (RowSerializable `row`)

```
@Override
public void writeRecord(RowSerializable row) throws IOException {
    if(tableMode.equals(CREATE)){
        if (!utils.getClient().tableExists(tableName)) {
            createTable(utils, tableName, fieldsNames, row);
        }else{
            this.table = utils.getClient().openTable(tableName);
        }
    }
    if(table!=null)
        utils.insert(table, row, fieldsNames);
}
```

STREAMS SINK: *KUDUSINK*

1. Create custom class which extends Flink's **RichSinkFunction**.
2. Implement method of superclass
   - open
   - invoke

3. Usage

```
DataStream<String> stream = ....
stream.addSink(new
KuduSink(KUDU_MASTER,tableName,columnNames));
```

STREAMS SINK: *KUDUSINK*

invoke (RowSerializable row)

```
@Override
public void invoke(RowSerializable row) throws IOException {

    // Establish connection with Kudu
    if (this.utils == null)
        this.utils = new Utils(host);

    if (this.table == null)
        this.table = this.utils.useTable(tableName, fieldsNames, row);


    // Make the insert into the table
    utils.insert(table, row, fieldsNames);

    logger.info("Inserted the Row: | " + utils.printRow(row) +
            "at the table \"" + this.tableName + "\"");
}
```

EXAMPLES

USE CASES

Batch processing

- Read data from Kudu table as a Flink DataSet object.
- Write data from Flink DataSet object to Kutu table

Streaming processing

- Write data from Flink DataStream object to Kutu table

EXAMPLES

Batch

- JobSource
- JobBatchSink
- JobBatchInputOutput

Streaming

- JobStreamingSink
- JobStreamingInputOutput

EXAMPLES

BATCH

```
ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();

DataSet < RowSerializable > input = KuduInputBuilder.build(env,TABLE_SOURCE, KUDU_MASTER)

result = input.map(new MyMapFunction())

result.output(new KuduOutputFormat(KUDU_MASTER, TABLE_SINK, columnNames, MODE));

env.execute();
```

EXAMPLES

STREAMING

```
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

DataStream<String> stream = env.addSource(new FlinkKafkaConsumer09<>(
                prop.getProperty("topic"),
                new SimpleStringSchema(),
                prop));

DataStream<RowSerializable> result = stream.map(new MyMapFunction());

result.addSink(new KuduSink(KUDU_MASTER, tableName, columnNames));

env.execute();
```

DEMO

CONTRIBUTION TO OPEN SOURCE

GITHUB & MAVEN

This work was performed by **junior developers**

- Learn Apache Flink
- Learn Apache Kudu
- **Encourage the use and contribution to the open source community**

Contributions are welcome!

https://github.com/rubencasado/Flink-Kudu

```
<!-- https://mvnrepository.com/artifact/es.accenture/flink-kudu-connector -->
<dependency>
    <groupId>es.accenture</groupId>
    <artifactId>flink-kudu-connector</artifactId>
    <version>1.0</version>
</dependency>
```

APACHE BAHIR

Apache Bahir provides extensions to distributed analytics platforms such as Apache Spark and Apache Flink

1) Open Jira ticket

2) Ticket is assigned to you

3) Open a Pull-Request

4) Overhaul the code

5) Done!

CONCLUSIONS

Flink and Kudu are good technology candidates to implement **Kappa Architectures**

We have implemented a first version of the **flink-kudu connector**

Work done by **junior developers** with the main goal of encouraging the use and contribution to the **open source community**

Thanks!!

Rubén Casado Tejedor
ruben.casado.tejedor@accenture.com
ruben_casado