SparkSQL Data Source Extension Practice in Huawei

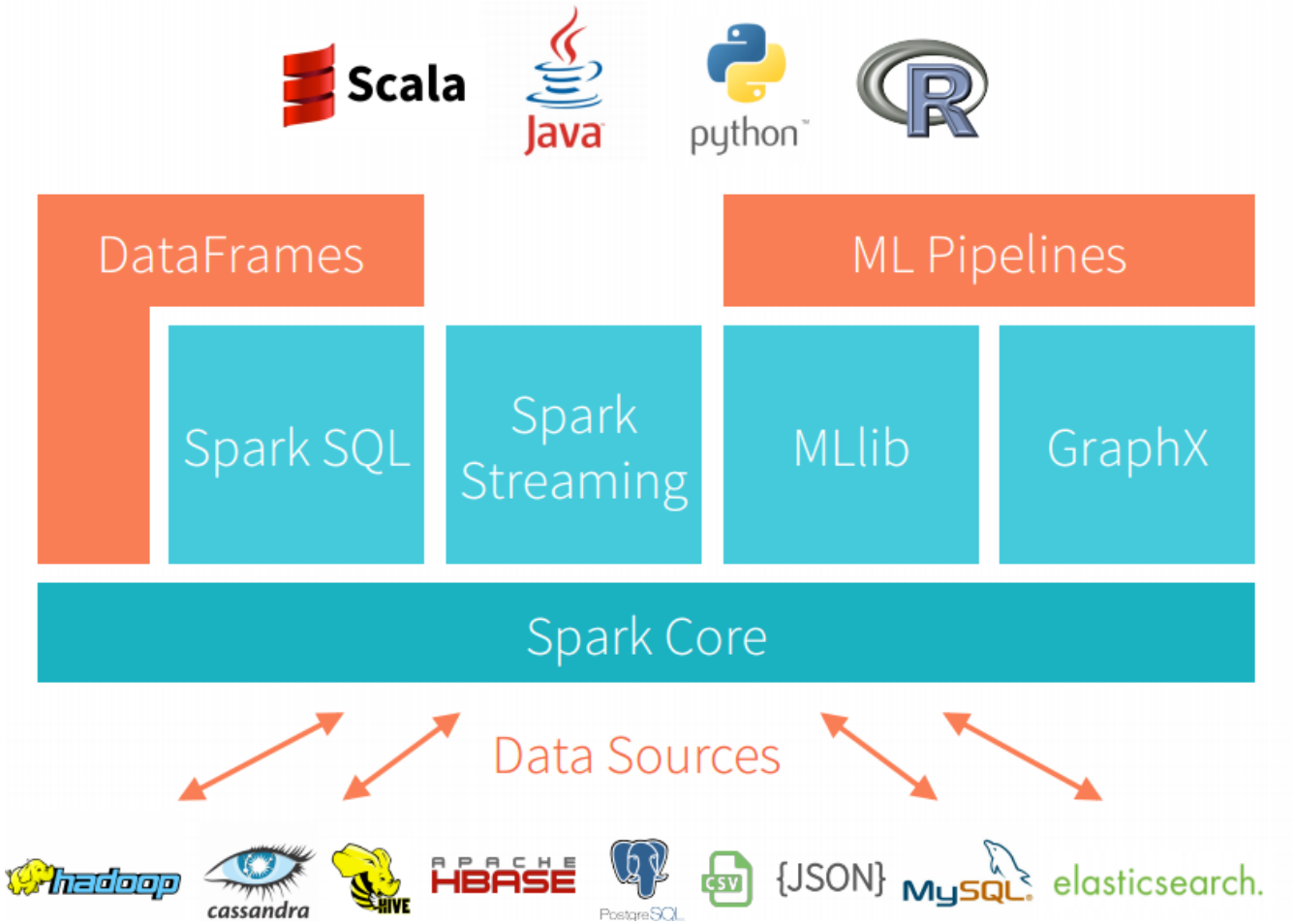
Jacky Li
jacky.likun@huawei.com
2015-7-20 (OSCON)

HUAWEI TECHNOLOGIES CO., LTD.

Agenda

- SparkSQL and Data Source API
- Write your own Data Source Lib
- Big Data in Huawei
- SparkSQL extension in Huawei
  - Astro: SparkSQL on HBase
  - Carbon: SparkSQL on Cube

SparkSQL Overview



SparkSQL:

A module for structured data processing

- DataFrame API: write less code
- DataSource API: read less data
- Catalyst：let optimizer do the hard work

DataFrame API：write less code

Using MapReduce API:

```
public static class WordCountMapClass extends MapReduceBase
  implements Mapper<LongWritable, Text, Text, IntWritable> {

  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(LongWritable key, Text value,
                  OutputCollector<Text, IntWritable> output,
                  Reporter reporter) throws IOException {
    String line = value.toString();
    StringTokenizer itr = new StringTokenizer(line);
    while (itr.hasMoreTokens()) {
      word.set(itr.nextToken());
      output.collect(word, one);
    }
  }
}

public static class WorkdCountReduce extends MapReduceBase
  implements Reducer<Text, IntWritable, Text, IntWritable> {

  public void reduce(Text key, Iterator<IntWritable> values,
                     OutputCollector<Text, IntWritable> output,
                     Reporter reporter) throws IOException {
    int sum = 0;
    while (values.hasNext()) {
      sum += values.next().get();
    }
    output.collect(key, new IntWritable(sum));
  }
}
```

Using Spark RDD API:

```
pdata.map(lambda x: (x.dept, [x.age, 1])) \
  .reduceByKey(lambda x, y: [x[0] + y[0], x[1] + y[1]]) \
  .map(lambda x: [x[0], x[1][0] / x[1][1]]) \
  .collect()
```

Using Spark DataFrame API:

```
data.groupBy("name") \
  .agg(avg("age"))
```

From data engineer to data scientist, who is not familiar with functional programming

DataSource API : connect to more data

- Easy to use API for your to loading/saving DataFrames
- Work together with SparkSQL query optimizer to allow efficient improvement.
  e.g. avoid reading unnecessary data by filter pushed down

built-in

external

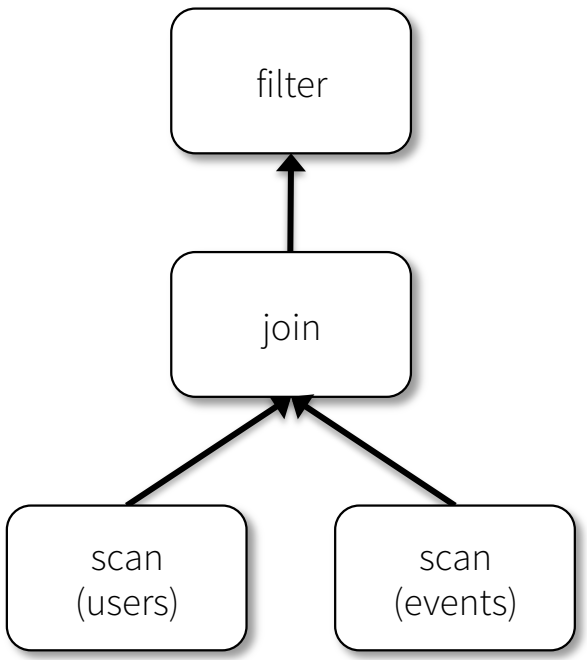
and more …

Source：databricks

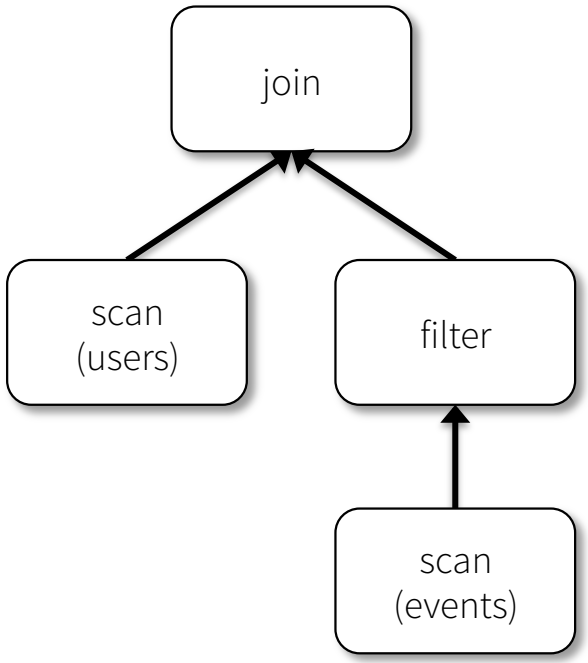
Catalyst: Query optimization

```
joined = users.join(events, users.id == events.uid)
filtered = joined.filter(events.date > ”2015-01-01”)
```

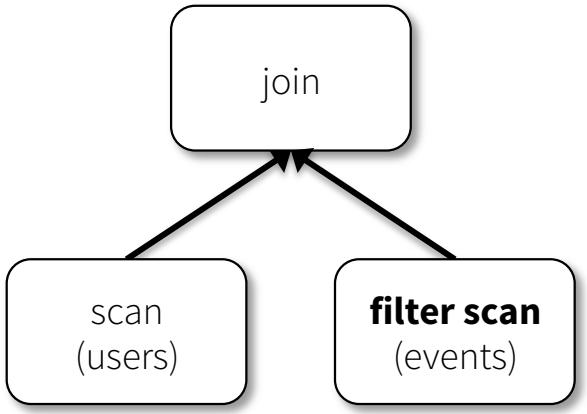
logical plan



optimized plan

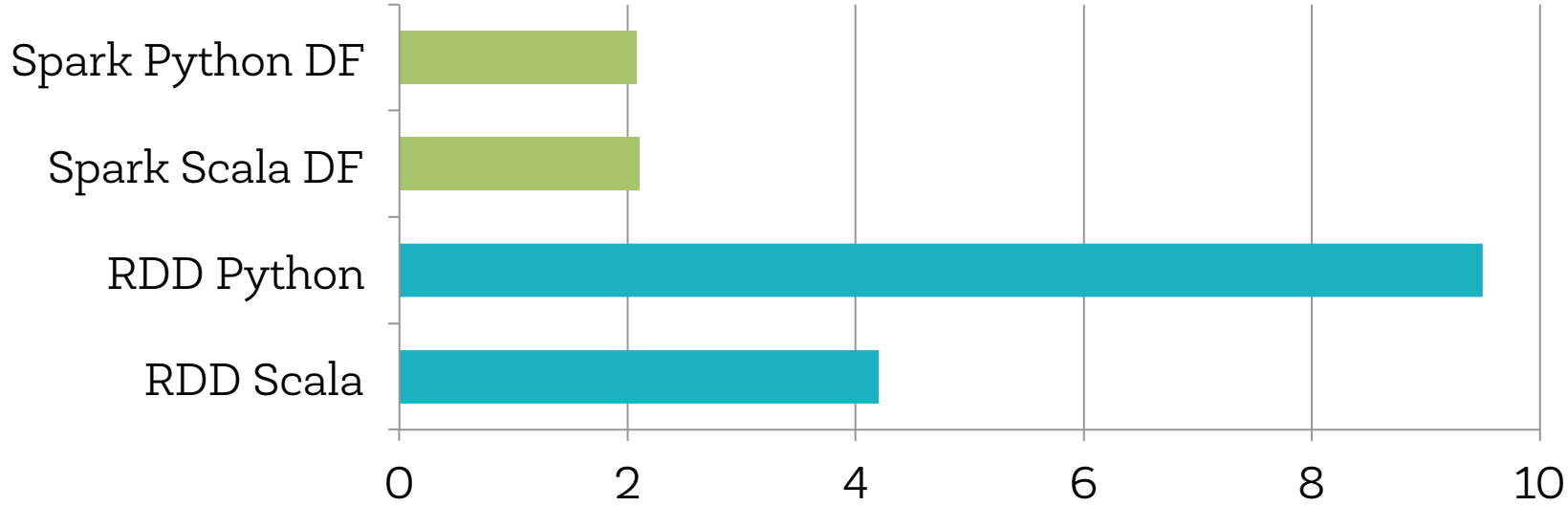


optimized plan
with intelligent data sources



Source：databricks

Performance



Runtime performance of aggregating 10 million int pairs (secs)

Source：databricks

- 2X performance improvement
- All language achieve the **same performance**!

Learn more

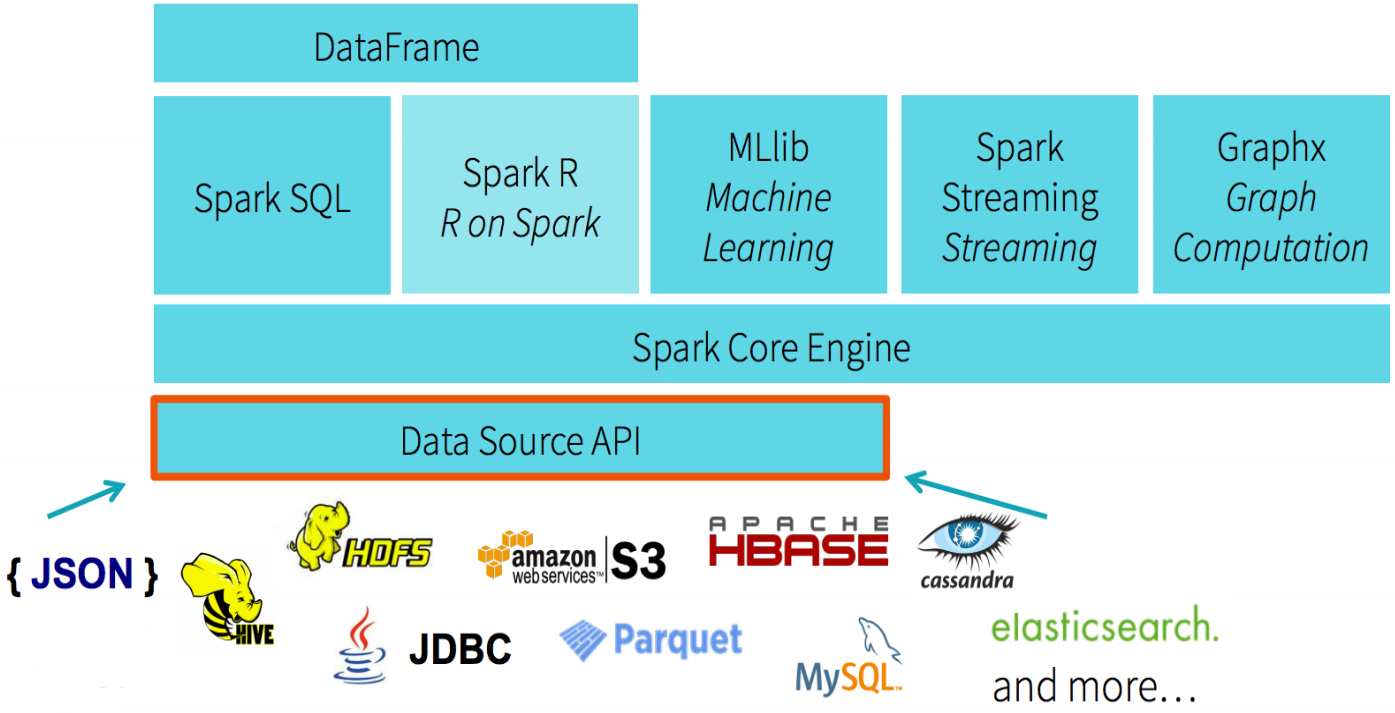
Programming Guide:

http://spark.apache.org/docs/latest/sql-programming-guide.html

Spark Meetup: http://www.meetup.com/spark-users/

Write your own Data Source Lib

Why use Data Source API



Data source API

- Uniform way to Access Data sources
- Pluggable data sources
  http://spark-packages.org/
- API is still young

Use case

- leverage spark ecosystem on legacy data source
- improve scalability and performance of standalone data source
- unified access point to all data

How to write a Data Source Library

Implement 3 interfaces:

- **BaseRelation: base class to extend, every relation is associated with a schema definition**
- **RelationProvider: factory for creating the concrete relation**
- **Scan: multiple types of TableScan interface are designed, choose one to implement**

support full record scan

support column pruning

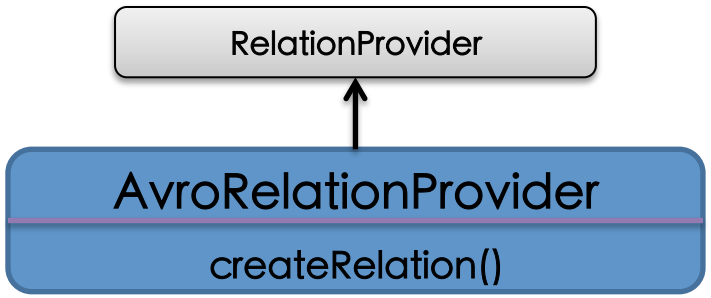
supports column and filter pruning

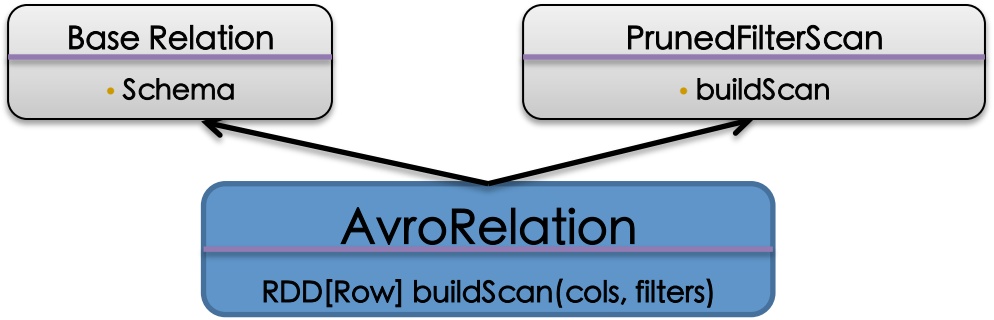
Example: Apache Avro

```
CREATE  TABLE episodes USING
com.databricks.spark.avro
OPTIONS (path "episodes.avro")
```


```
SELECT episodes  from … where  …
```

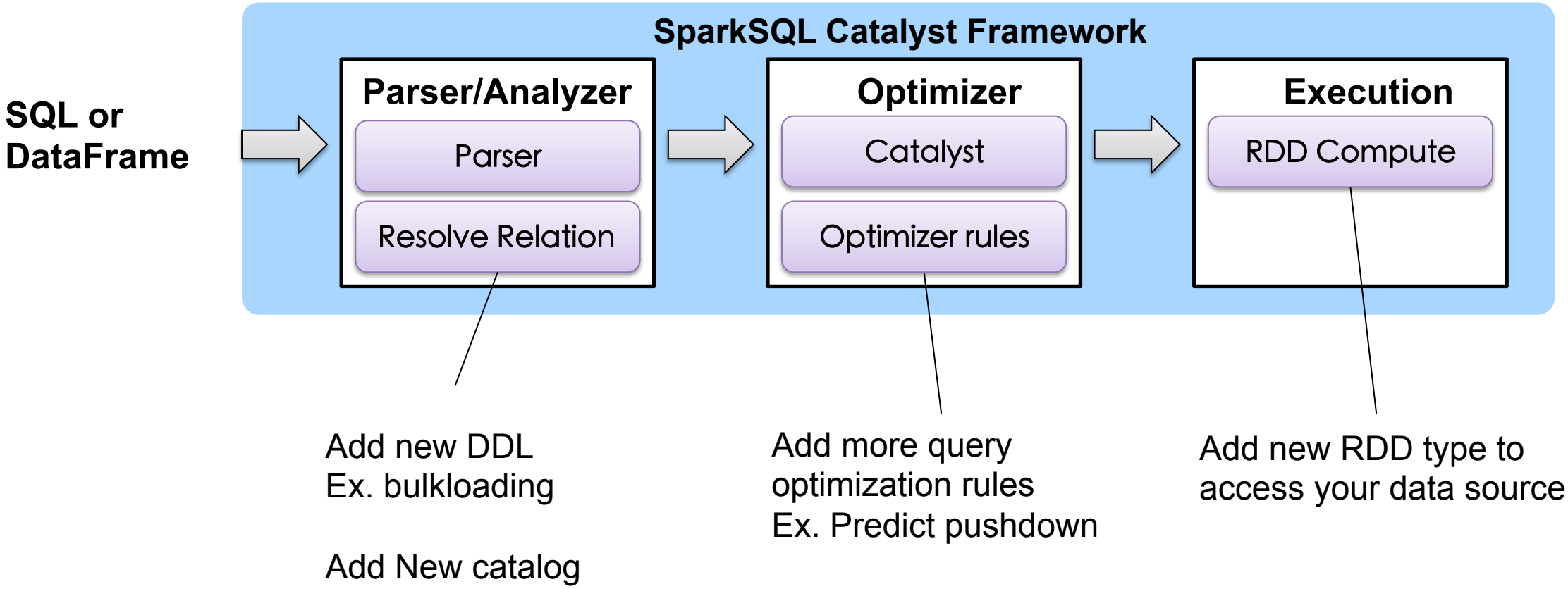
package com.databricks.spark.avro





AvroRelation deriving from PrunedFilterScan, which supports column pruning and filter pruning.

Advanced data source: more syntax and optimization

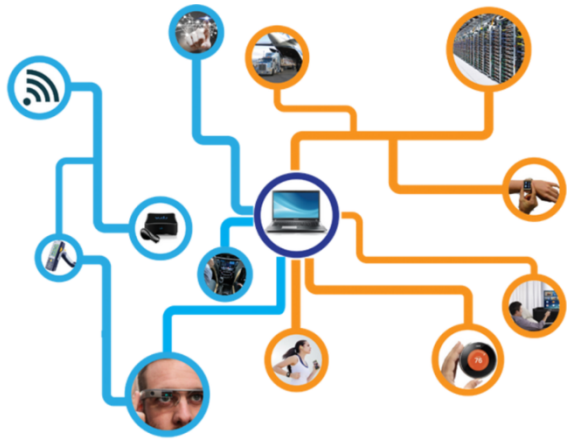


Big Data in Huawei

Big Data in Huawei

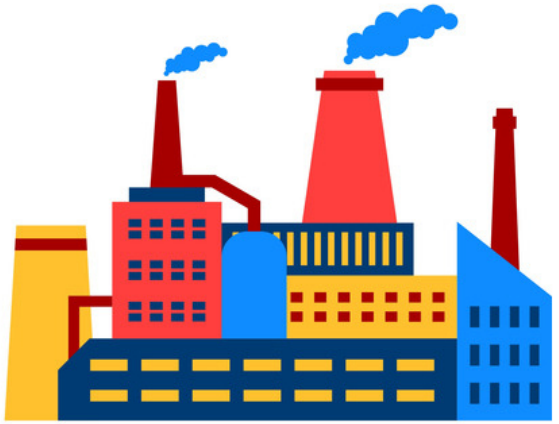
Network Carrier

- Resource Utilization
- Customer Care
- Market Insight
- Data Monetization



Consumer

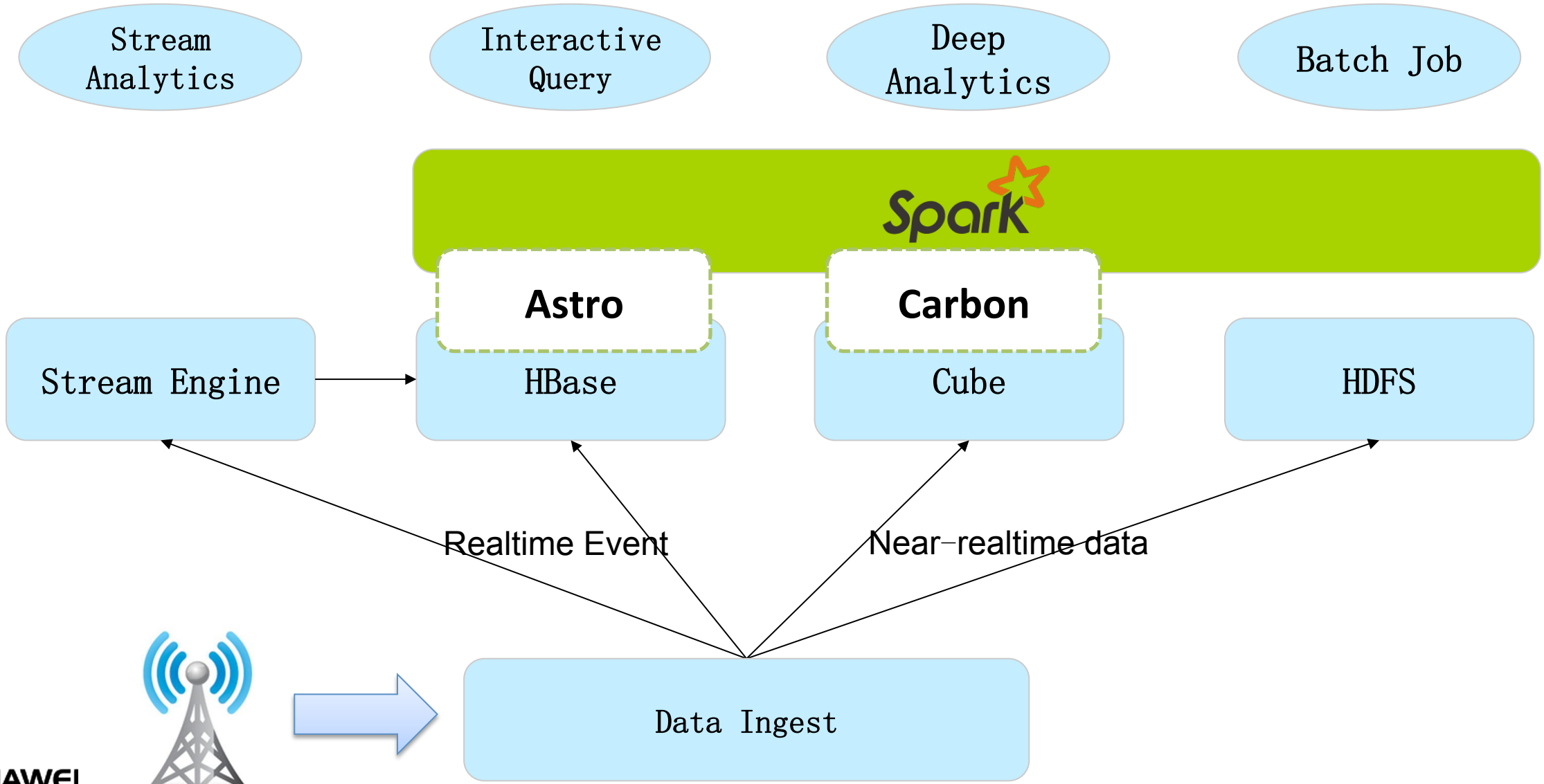
- Campaign
- Realtime Recommendation
- Community Analysis



Enterprise

- Hadoop/Spark Distribution
- Cloud Service

Spark in Huawei



Astro:
SparkSQL on HBase

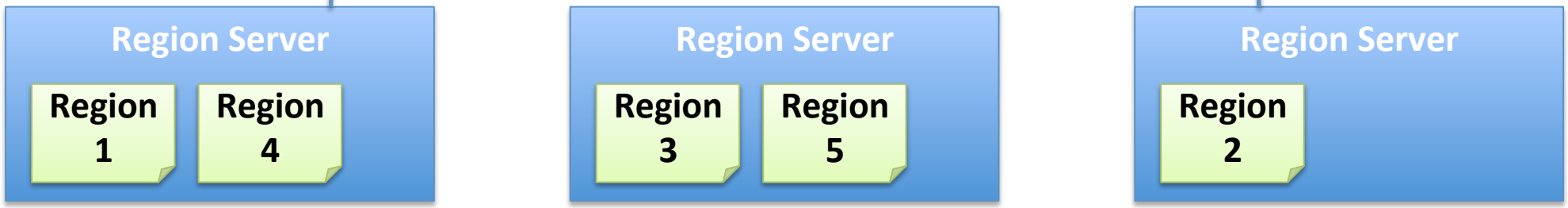
What is HBase

A big sorted table, which is splited into many parts (regions) stored in a HDFS cluster

Row key	Column key	Timestamp	Cell
Row1	info:aaa	1273516197868	valueA
Row1	info:bbb	1273871824184	valueB
Row1	info:bbb	1273871823022	oldValueB
Row1	info:ccc	1273746289103	valueC
Row2	info:hello	1273878447049	i_am_a_value
Row3	info:	1273616297446	another_value

Sorted by Row key and Column

2 Versions of this cell



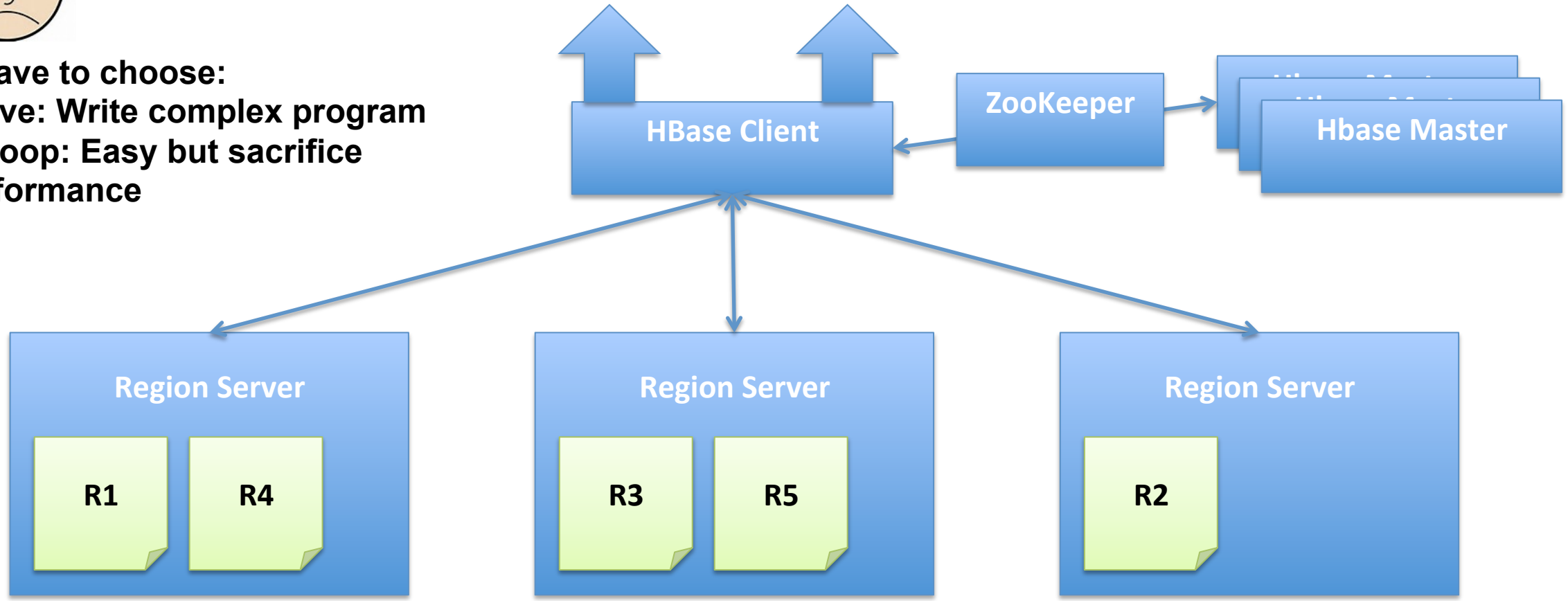
How to access HBase data

Native API: get, put, scan
Filter, coprocessor

Hadoop API: InputFormat,
OutputFormat

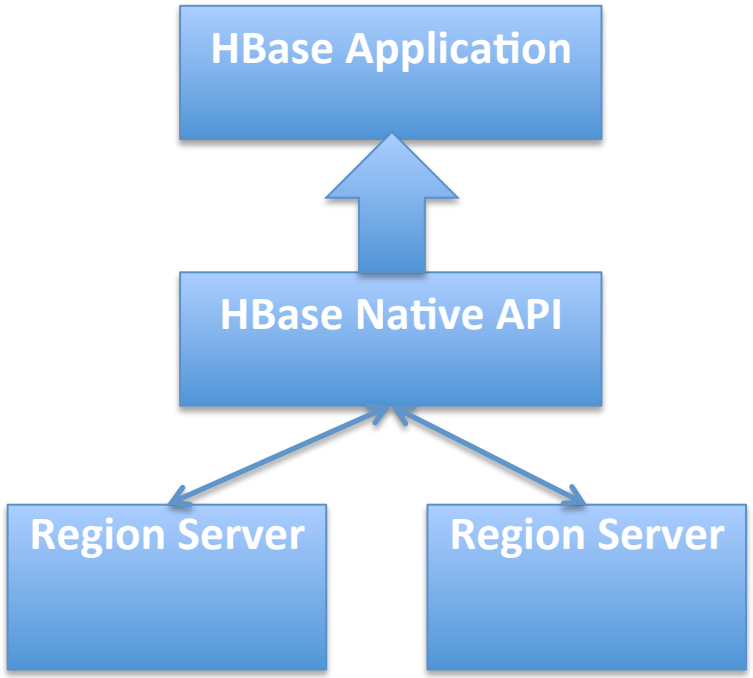
User have to choose:

1. Native: Write complex program
2. Hadoop: Easy but sacrifice performance



Existing Solutions

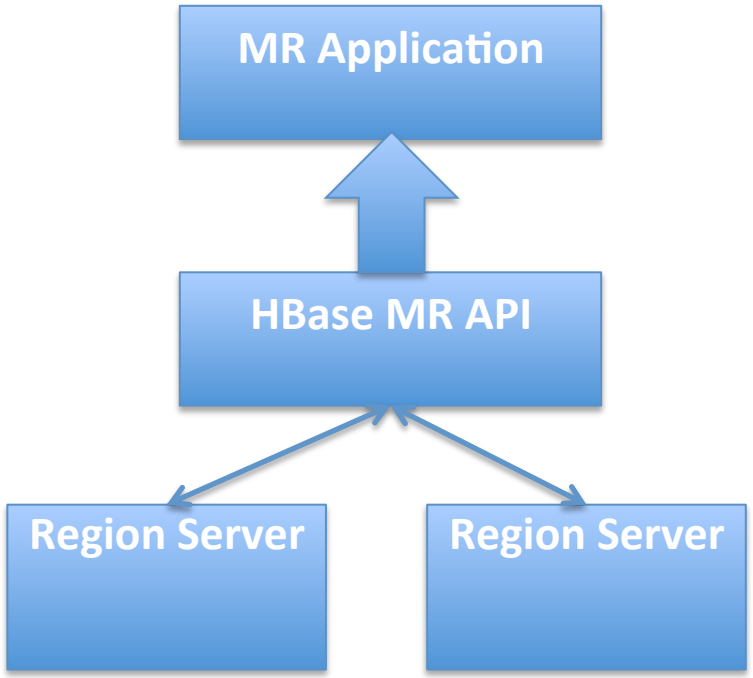
Solution 1: Native Application



Pros:
- Flexible
- High performance if do it right

Cons:
- Low productivity

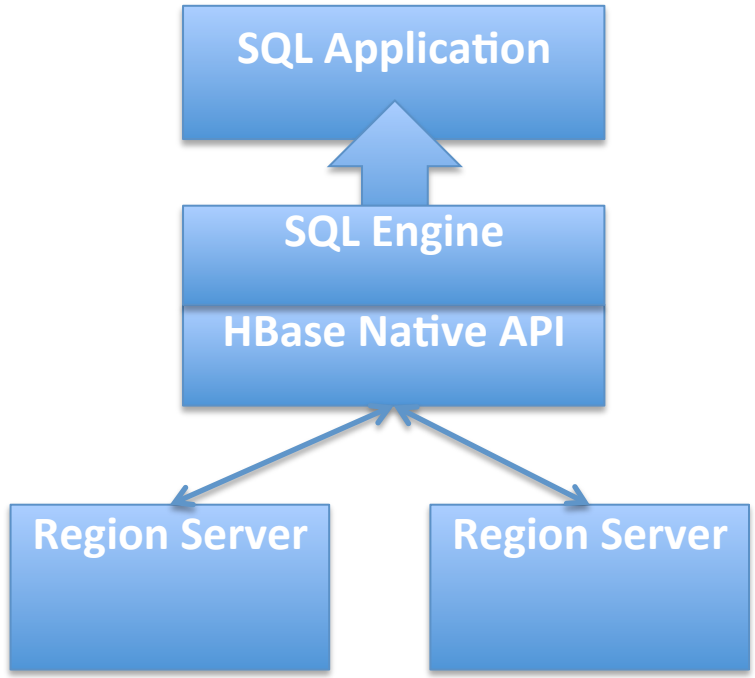
Solution 2: MR Application



Pros:
- Hadoop friendly
- Support SQL using Hive/Impala

Cons:
- Low performance

Solution 3: Purpose-built engine



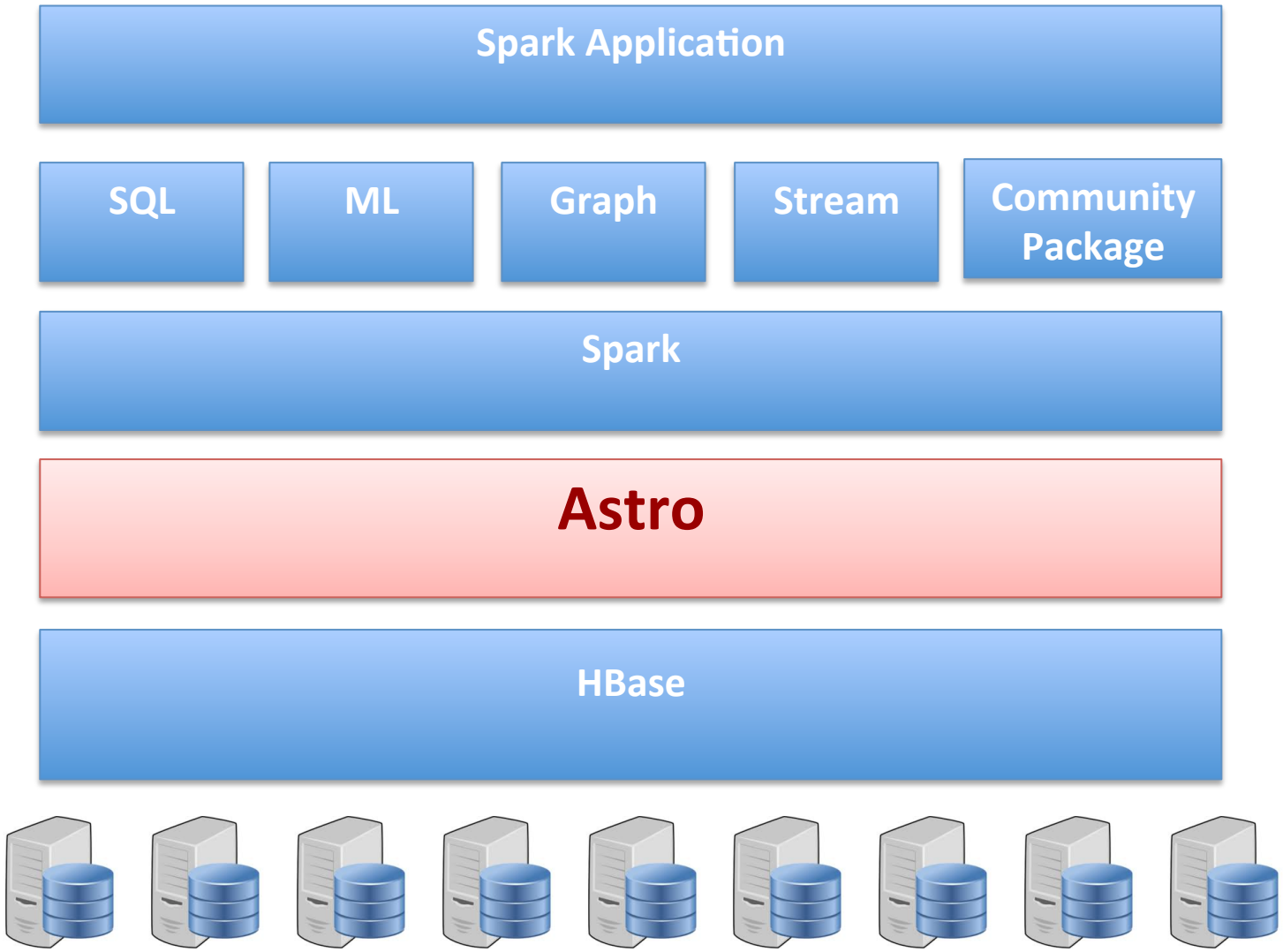
Pros:
- Support SQL
- High performance

Cons:
- Partial distributed

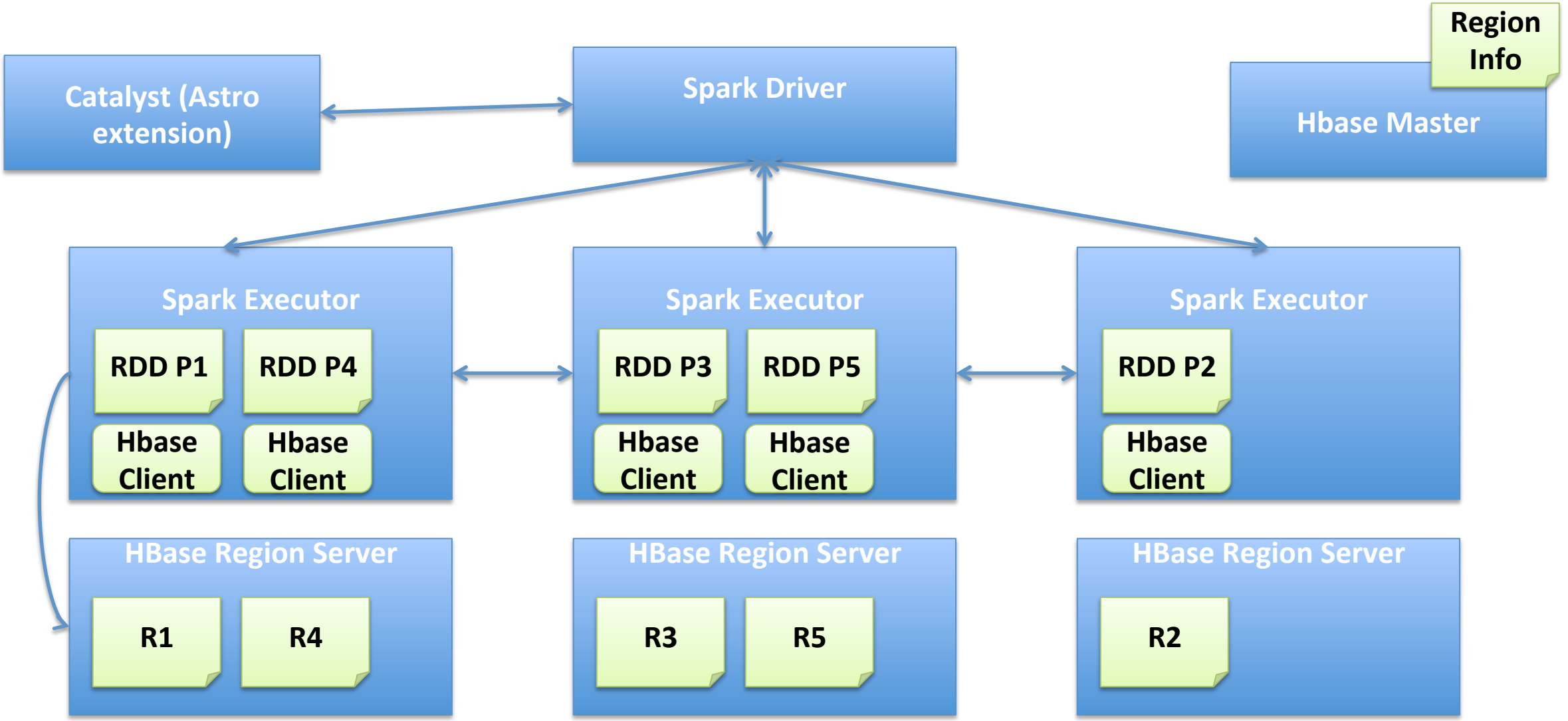
Introducing Astro

Astro = SQL on HBase +
Fully distributed +
Spark Ecosystem

https://github.com/Huawei-Spark/Spark-SQL-on-HBase



Astro Logical Architecture



Features

- Scala/Java/Python multi language support
- SQL and DataFrame API compatible
- Query Optimization: predicate pushdown, aggregation pushdown, region pruning, rowkey jumping, …
- More SQL capabilities: insert, update, bulk load, …
- Join HBase table with other data like Parquet
- CLI tool to execute SQL command

Data Models

CREATE TABLE table_name (col1 TYPE1, col2 TYPE2, …, PRIMARY KEY (col1, col2))
MAPPED BY (hbase_tablename, COLS=[col3=cf1.cq2, col4=cf2.cq1, col5=cf2.cq2])

SparkSQL Table

col1	col2	col3	col4	col5

Hbase Table

Row Key	Column Family cf1			Column Family cf2	
	Qualifier1	Qualifier2	Qualifier3	Qualifier1	Qualifier2
Row Key1	I		like		using
Row Key2		SQL			
Row Key3				on HBase	

Query Optimization

1. Region pruning by analyzing rowkey

   e.g. select … where key<3 or key>10

2. Use Hbase Filter to push down filter while scan, thus data transfer is minimized

   e.g. select … where value > 100

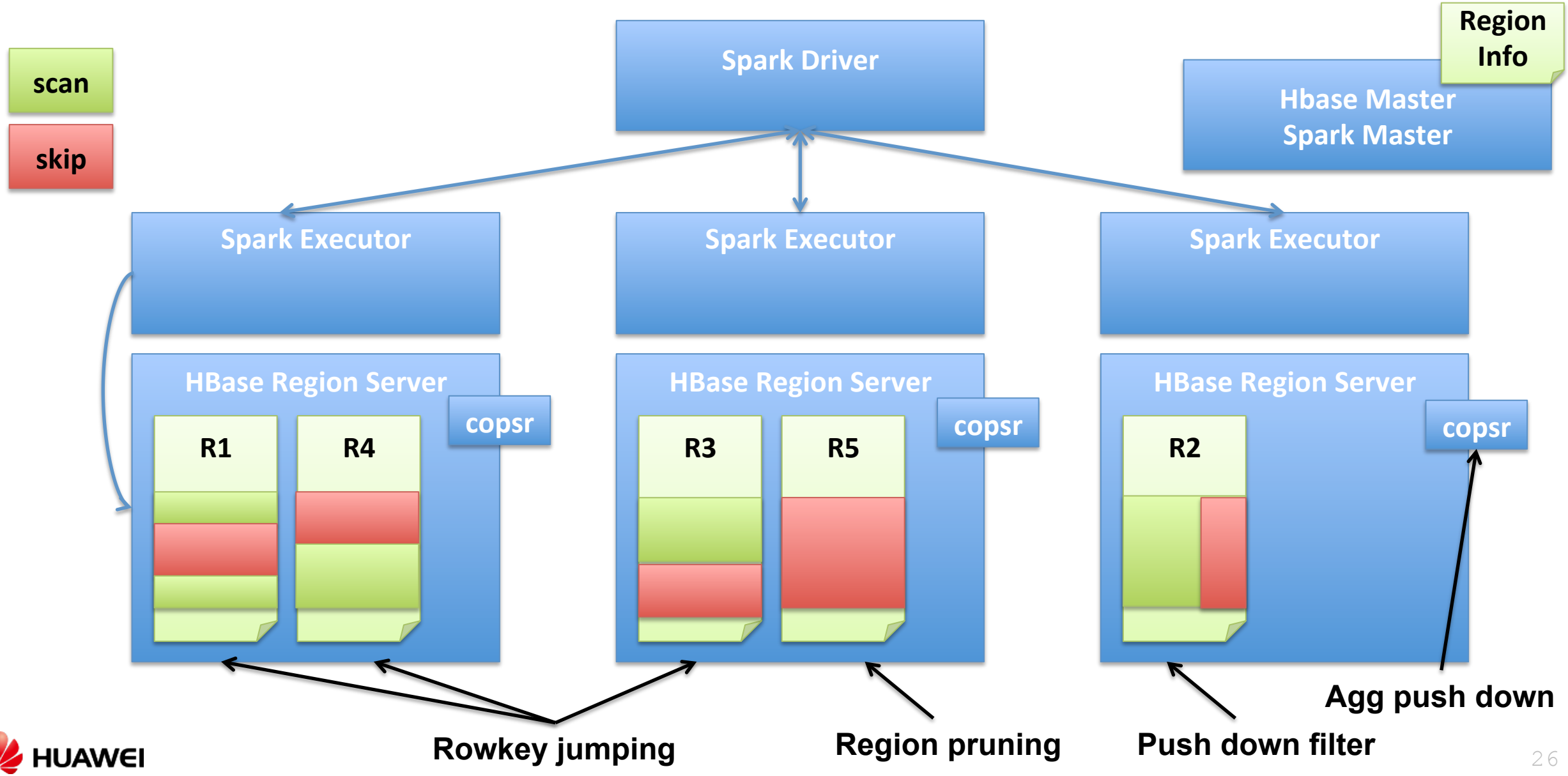
3. Implements Hbase Custom Filter to jump to the required full/partial key directly

   e.g. select … where (key1 < 3 AND key2 > 5) OR (key1 = 8 AND key 2 < 4)

4. Use Hbase coprocessor to push down computation and minimal data transfer

   e.g. select … sum(col) … group by col

Query Optimization Example



Project Info

- Open Source project
- Spark external package (WIP)
- Github Repo: https://github.com/Huawei-Spark/Spark-SQL-on-HBase
  Includes:
  - Design Doc
  - Source Tree
  - Test Cases
  - CLI tool
- Project Lead:
  - Yan Zhou (yan.zhou.sc@huawei.com)
  - Bing Xiao (bing@huawei.com)

Demo

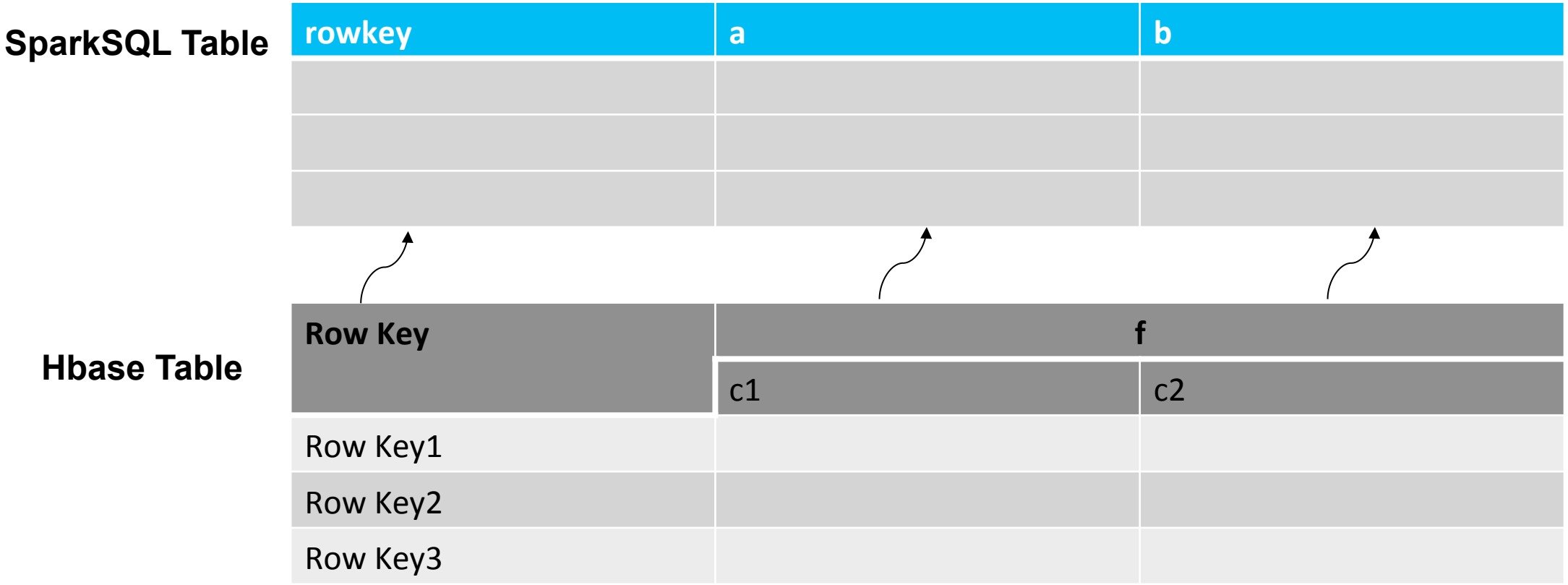
- Demo1: Create and query table with existing HBase table
- Demo2: Create and query table with new HBase table

Code can be found at

https://github.com/Huawei-Spark/Spark-SQL-on-Hbase/tree/master/examples

Demo1: create table with existing HBase table

- Create SparkSQL table map to existing Hbase table
- A single column map to hbase rowkey



Demo2: create table with new HBase table

- Create and query SparkSQL table map to Hbase table
- Multiple columns map to hbase table rowkey
- Bulk data sample data into SparkSQL table, which stores in Hbase table

SparkSQL Table

grade	class	subject	teacher_name	teacher_age

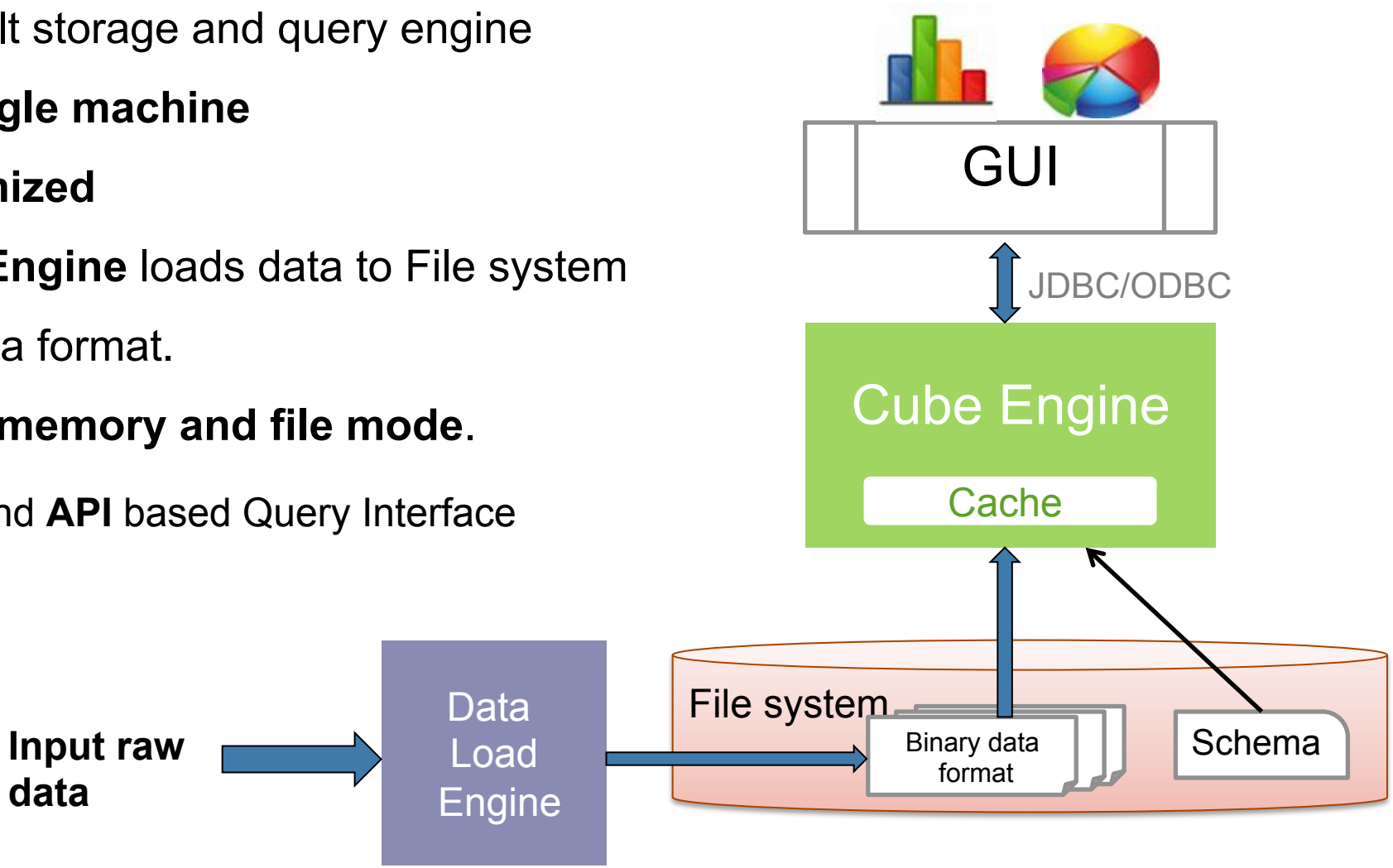
Hbase Table

Row Key	teacher	
	name	age
Row Key1		
Row Key2		
Row Key3		

Carbon:
SparkSQL on Cube

Before Spark: Standalone Cube Engine

- In-house built storage and query engine
- Runs on **single machine**
- Highly **optimized**
- **Data Load Engine** loads data to File system in Binary data format.
- Supports **in memory and file mode**.
- **MDX** , **SQL** and **API** based Query Interface

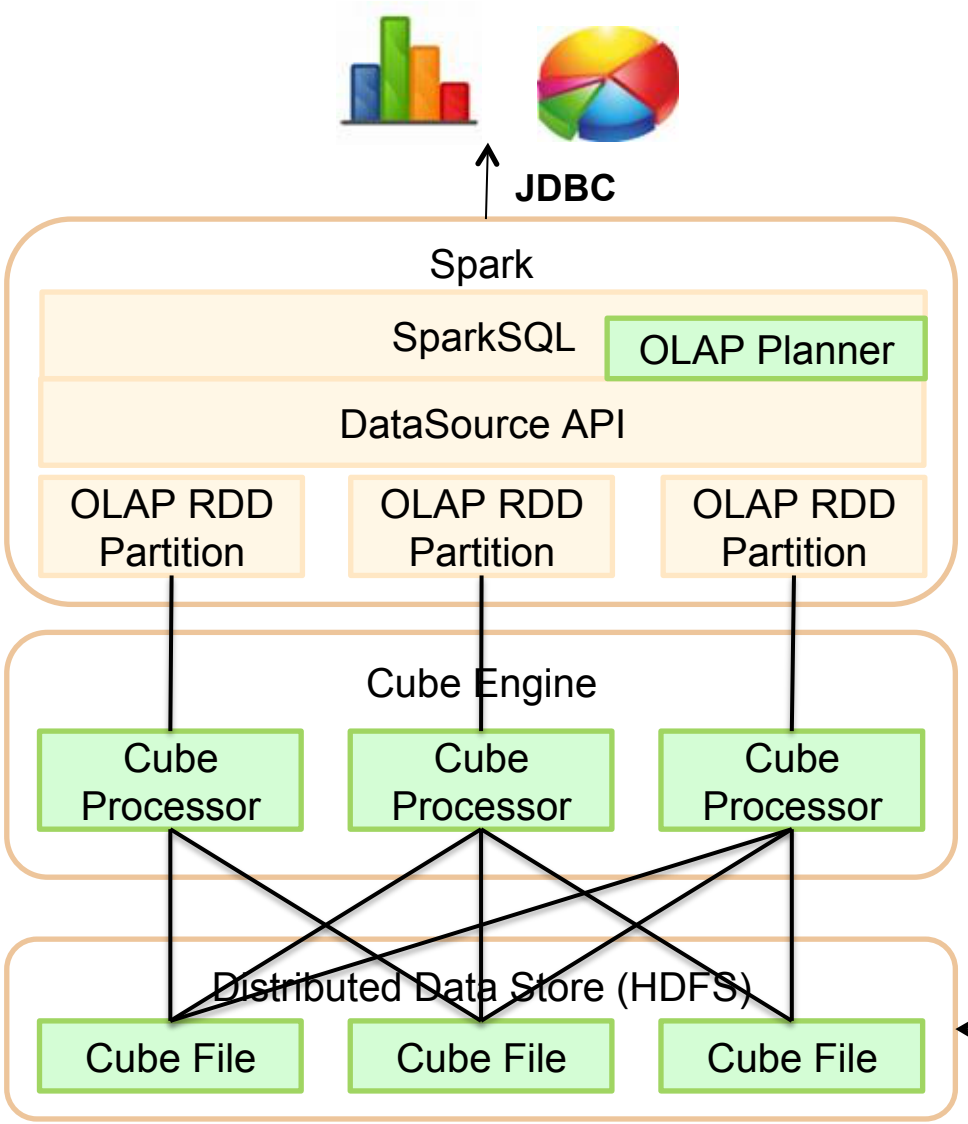


Motivations

- **Functionality**
  - OLAP style analytics over Big Data like slicing, dicing
  - BI tool integration
- **Scalability**
  - Re-use existing High performance engine, but utilize distributed computing framework to scale out
- **Reliability**
  - Utilize industry−proven storage layer - HDFS

Solution: leverage Spark to make it distributed and ecosystem friendly

Carbon Logical Architecture



Language：

- Use SparkSQL
- New DDL：CREATE CUBE, LOAD INTO CUBE

Compute：

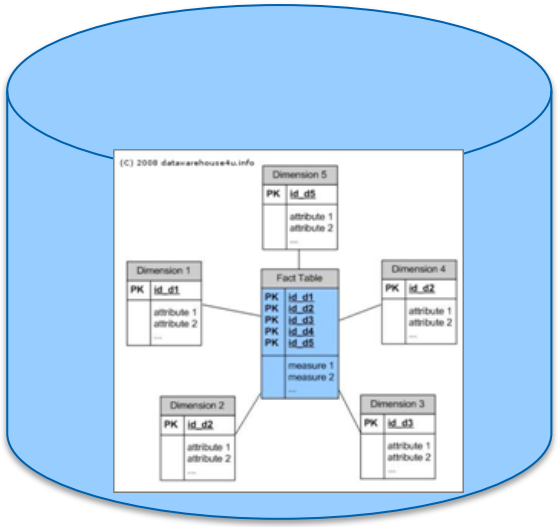
- Customized optimization rules based on Catalyst
- Cube Processor: Cube data scan, jump, agg, etc

Storage：

- Cube File, multidimensional index built-in
- Store Schema in HDFS/Hive Metastore

Data Model

Original data



Star Schema

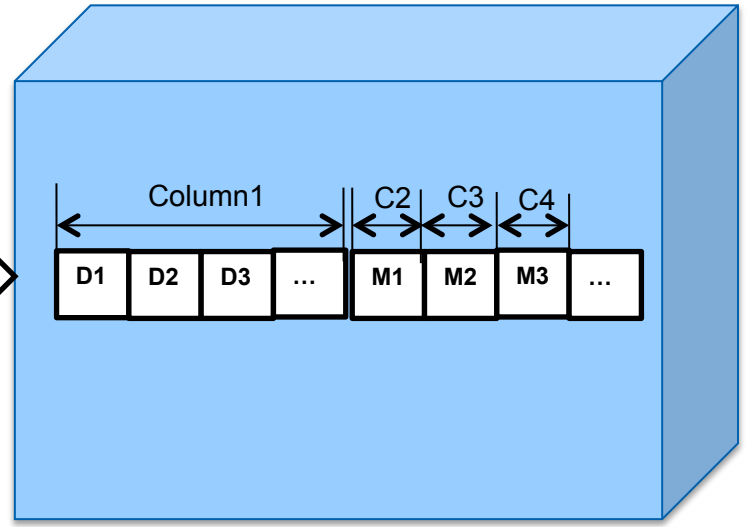
Create Cube and load data

Use SparkSQL to

1. **Create cube …**
2. **Load data….**

Cube Metadata

Cube Store in HDFS



Distributed Cube File

Cube File Format

Parquet, ORC: Store and Query complex nested structure data

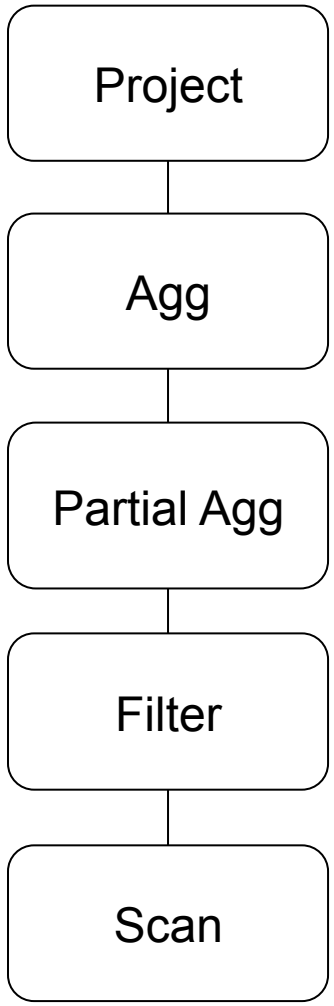
- Columnar format, support nested data structure
- Predicate push down: scan only required column, partition, min/max index
- Schema evolution

Cube File: Store and Query Multi-Dimensional Tabular data (star-schema)

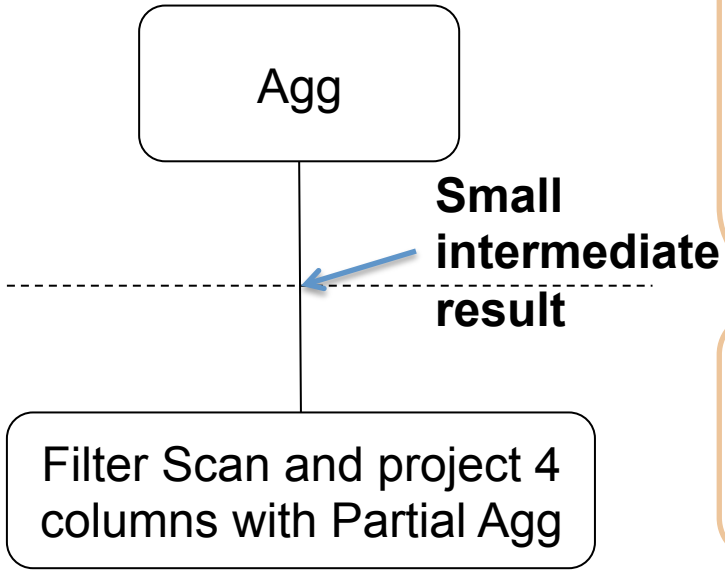
- Columnar format, with native Multi-Dimensional Key support
- More push down: multi dimension filter, group by, distinct count, …
- Trade query time with pre-process time to a more organized data

Push Down Example: Multi-dimension filter and agg

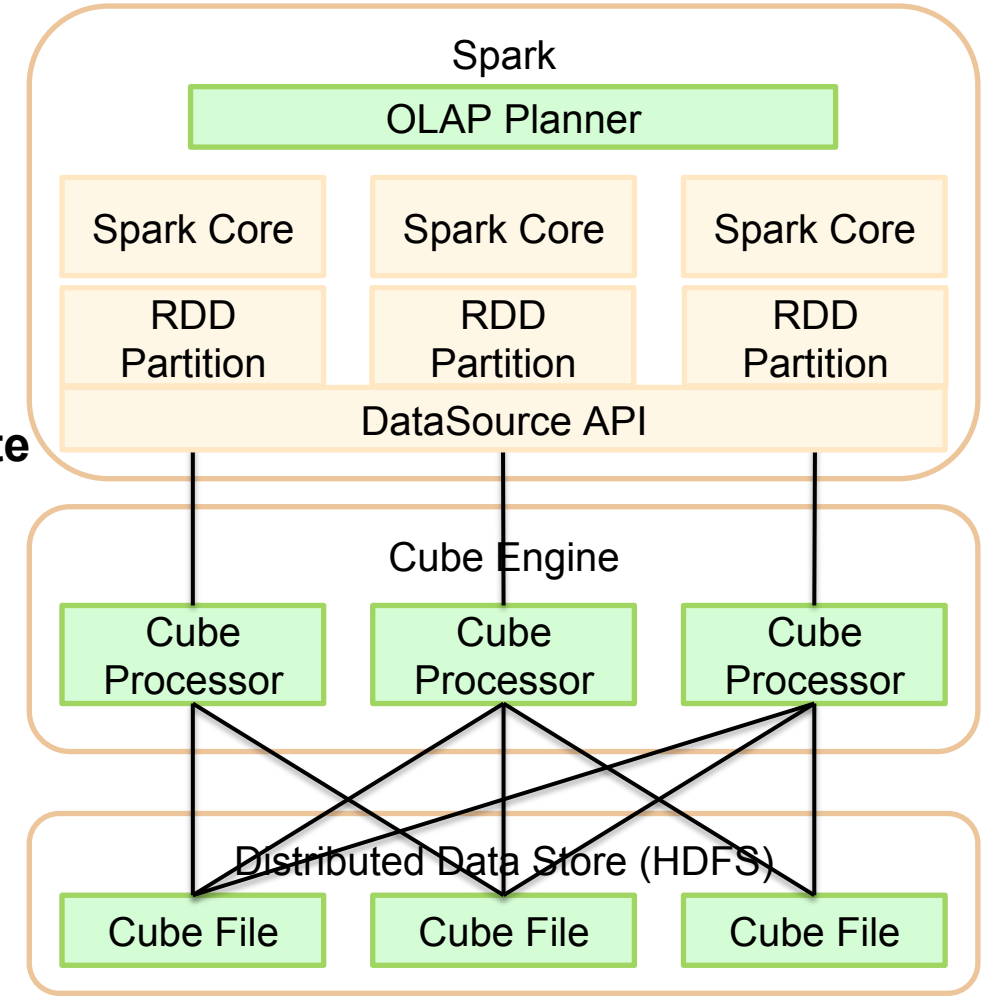
SELECT state, plan, terminal, sum(traffic) FROM user_cube WHERE plan= 4G and state= CA GROUP BY terminal



Before optimization



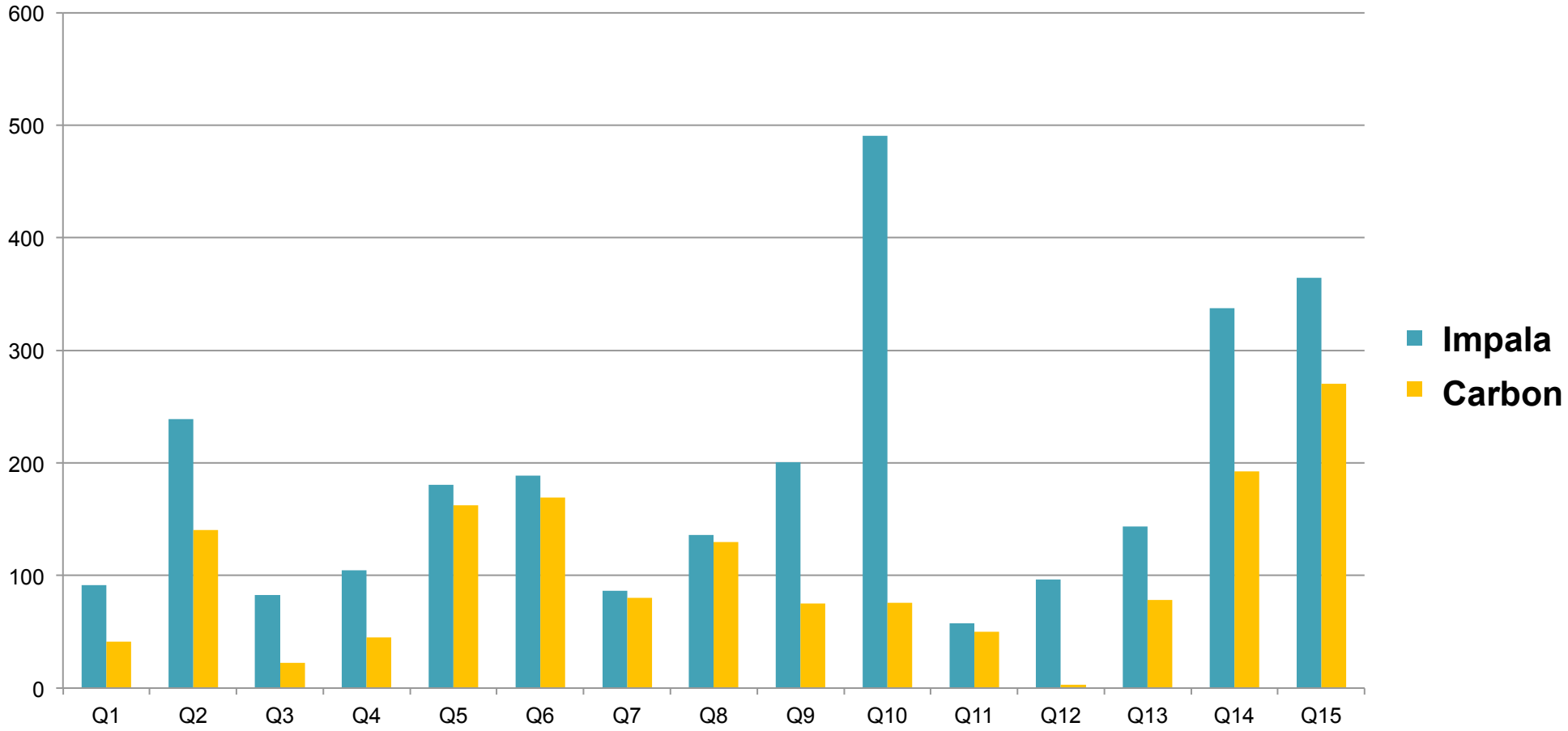
After optimization



Performance

12 Billion records，20 dimension，4 measure，total 1.5TB

- Carbon: Cube file 380GB
- Impala: Parquet file 336GB



Query includes: 1/many dim filter, 1/many dim group by and agg, distinct count

all ! "thanks"

questions.foreach(answer(_))