Introduction to Spark in the context of a Distributed Pipeline

Maria Dominguez, Xavier Tordoir

Scala.io 2019

LUNATECH
SIMPLIFY YOUR IT

JVM development

Devops

ML & Big Data

96
Employees

26
Nationalities

Lots
Open source

We welcome you to one of our offices

LUNATECH

Outline: covered concepts

Spark

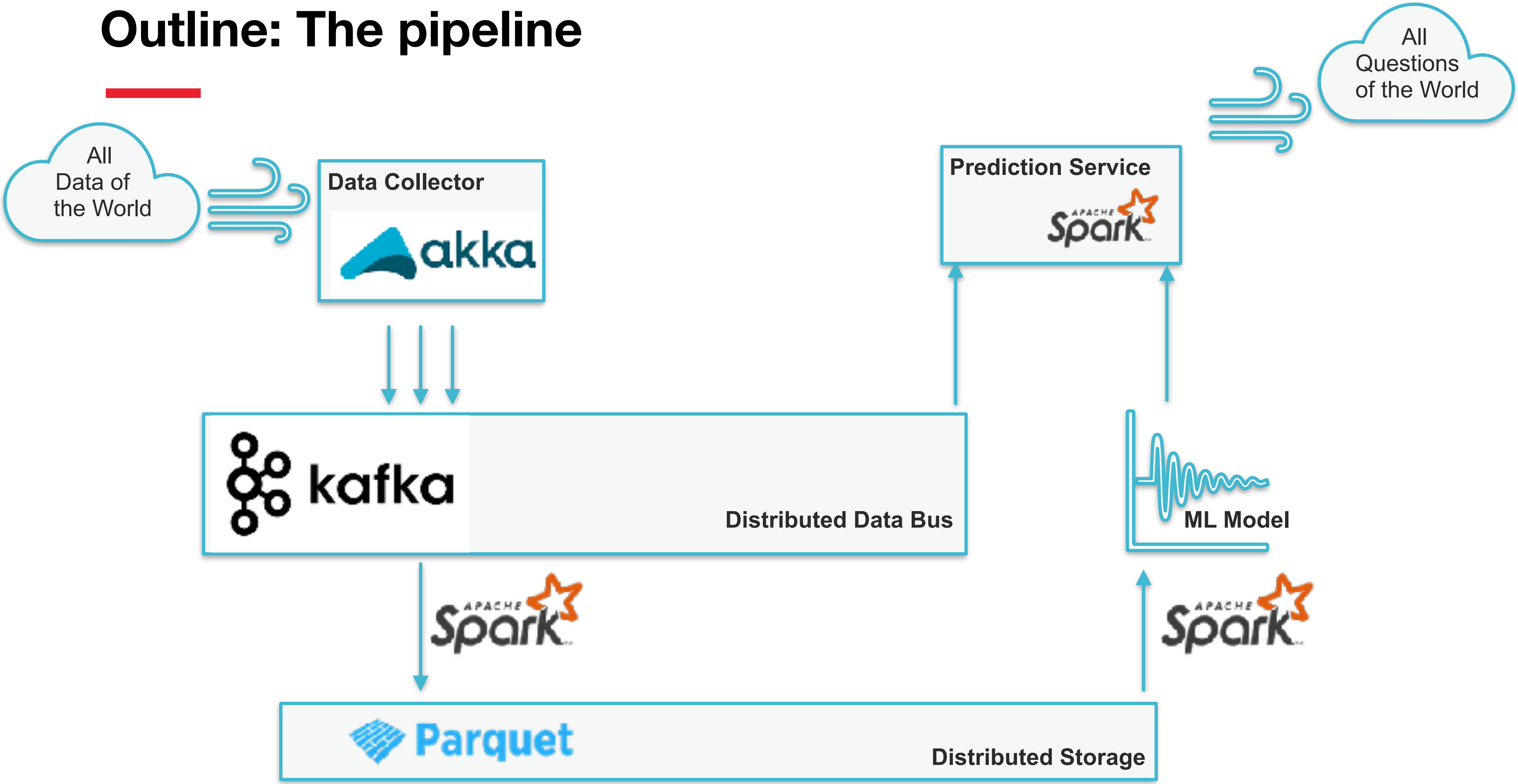
DataFrames & Datasets

Spark notebooks

ML concepts

Streaming

Outline: The pipeline



Hand-on set-up

VirtualBox installed

Download appliance: https://xtordoirtmp.s3-eu-west-1.amazonaws.com/sparkintro.ova

Import Appliance in virtualBox:

- Menu “File” -> “Import Appliance”

Start VM “sparkintro”

Open http://localhost:9000 in browser

Troubleshooting: Memory limits, network adapter 2 disconnect, use local ip

Notebooks, how? why?

Data Science implies:

- **knowledge** of the data, including its corner cases

- **Exploration** of how to guide the modelling, choosing the right methods for the data

- **Trials and errors**, no possibility to implement functional specs, only model validation

=> Need for **interactive programming**
=> **Notebooks**

Notebooks are good **educational** tools as well

Reference scala API

Notebooks, how? why?

jupyter

https://jupyter.org/

- Python environment
- Kernels to support different languages:

    https://almond.sh/

Zeppelin

https://zeppelin.apache.org/

Spark-notebook

http://spark-notebook.io/

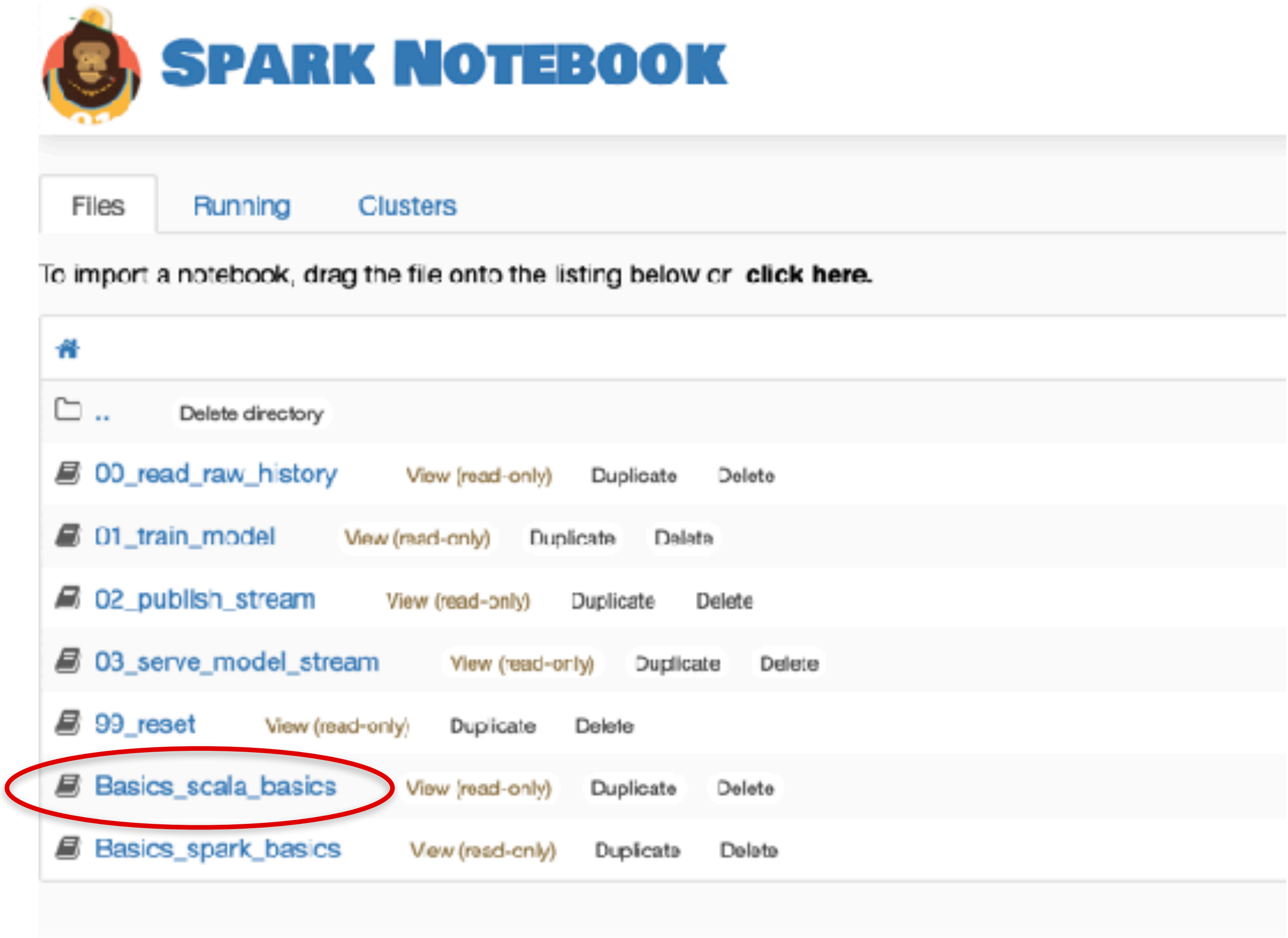
Scala notebooks

Notebooks: Take control of the environment

Understand the environment:

- Open the Reset notebook
- Look for markup cells, code cells
- Run cell with imports (ctrl-enter or shift-enter), close the tab
- Re-open the notebook, add a cell, check if imported calls work
- How can that be?
- Save, shutdown kernel
- Restart Kernel, what happens to imported calls?

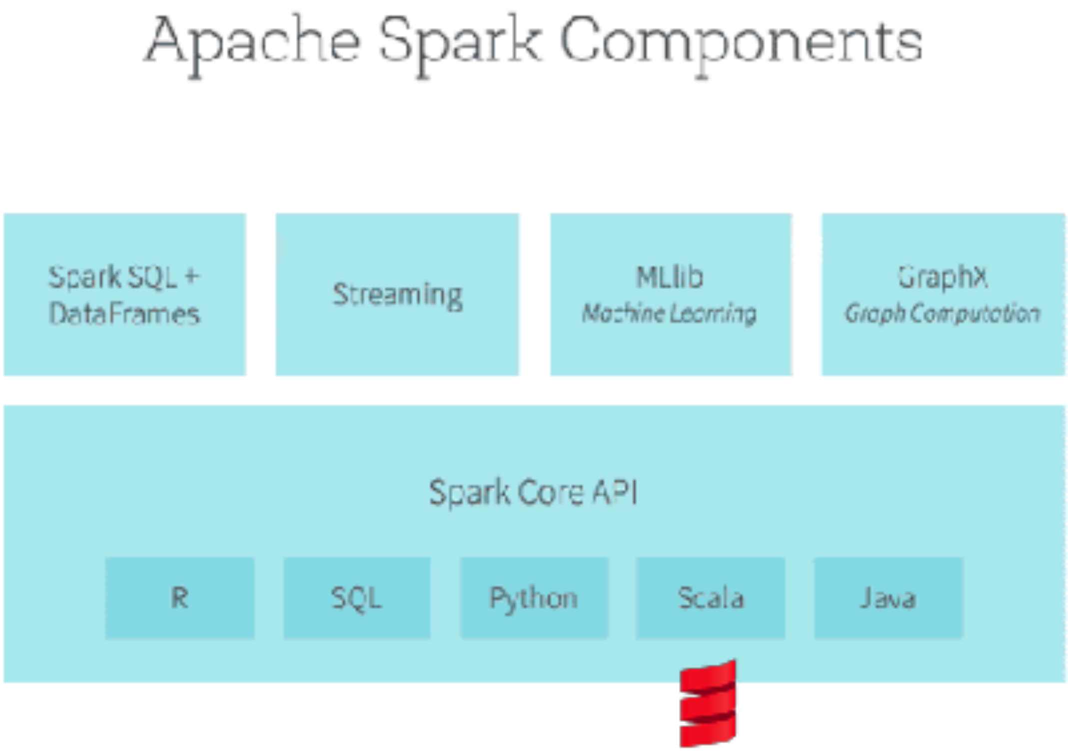
Start with a little bit of Scala: Collection API

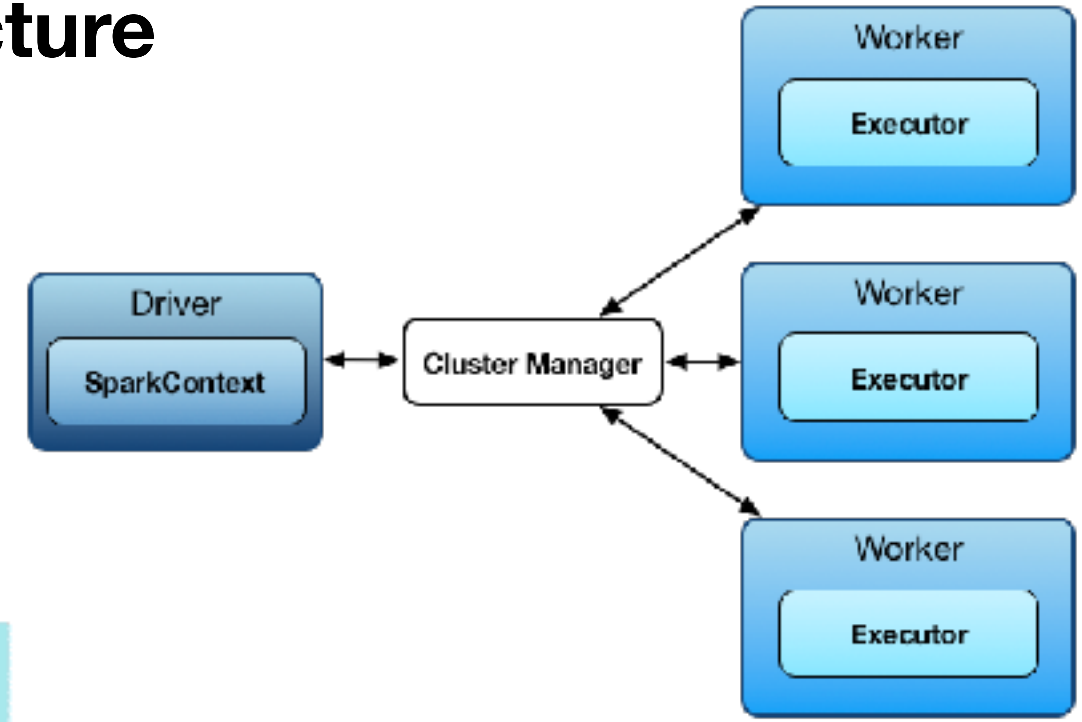


Spark history

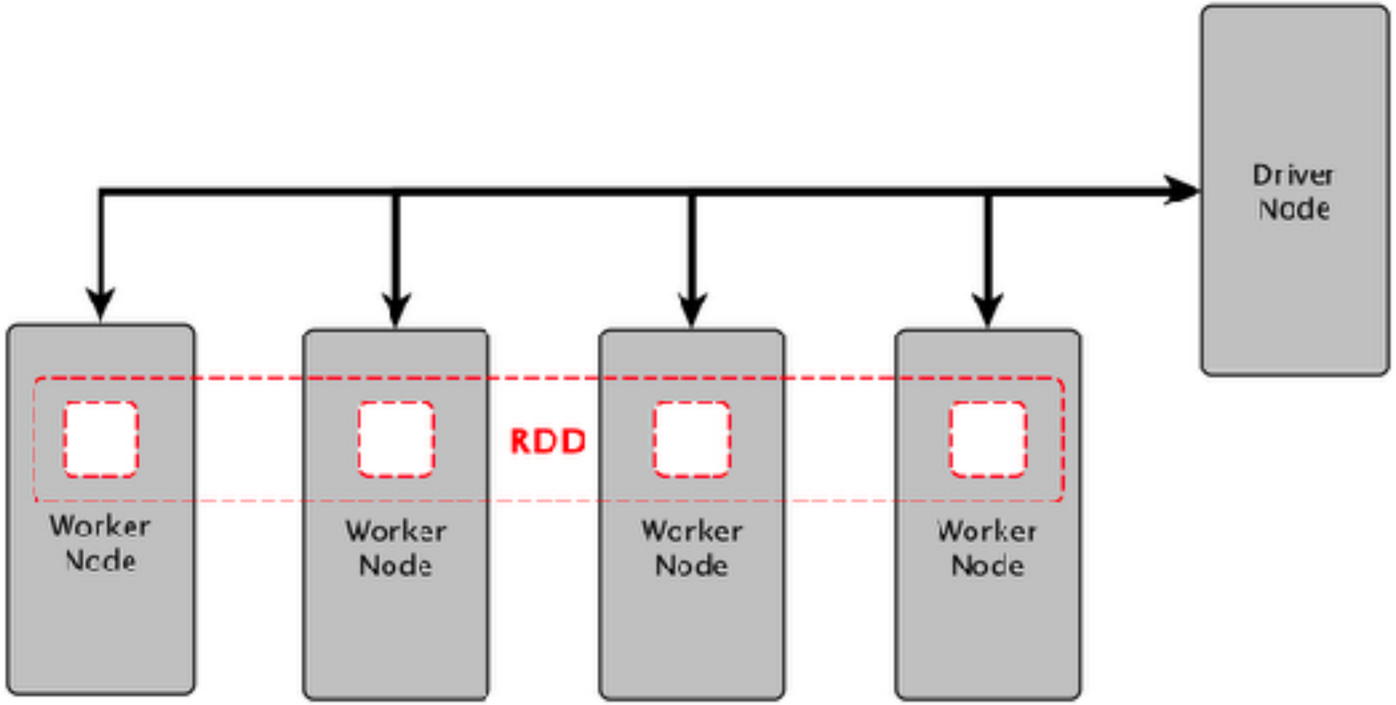
2006	2009
Batch processing	Batch and stream processing
Trivial operations are difficult (filter, join)	Trivial operations are easy (filter, join)
Writing to disk	In memory computation

Spark components & architecture

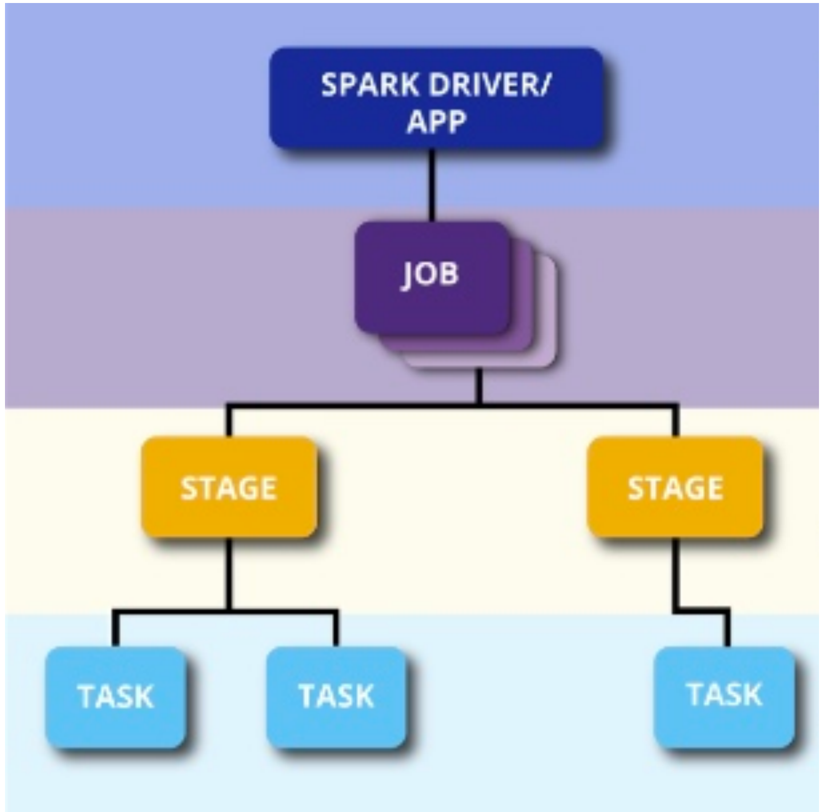




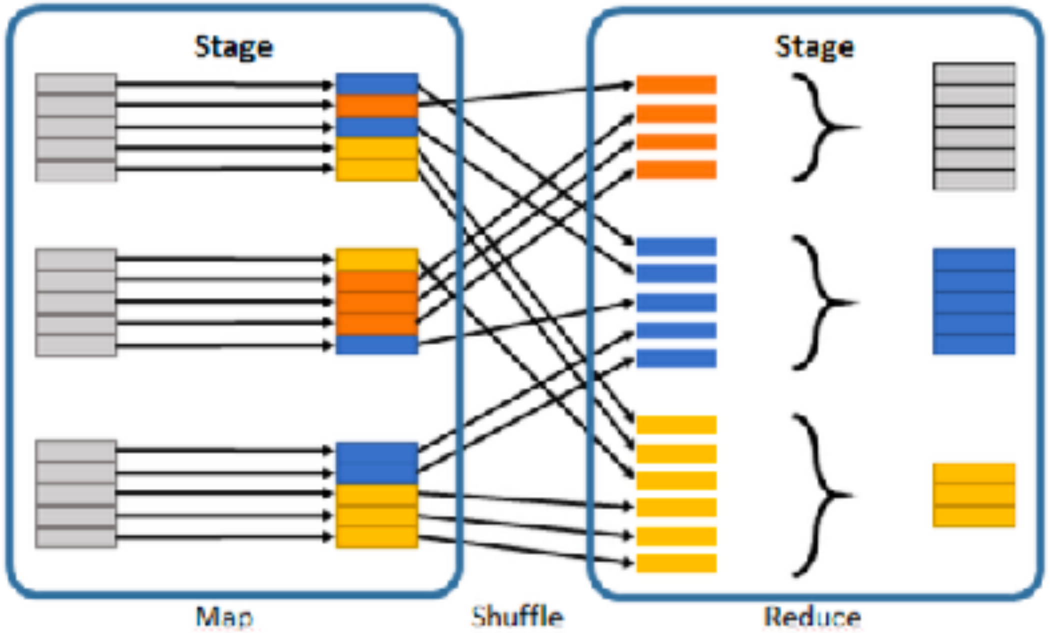
Spark: Partitions



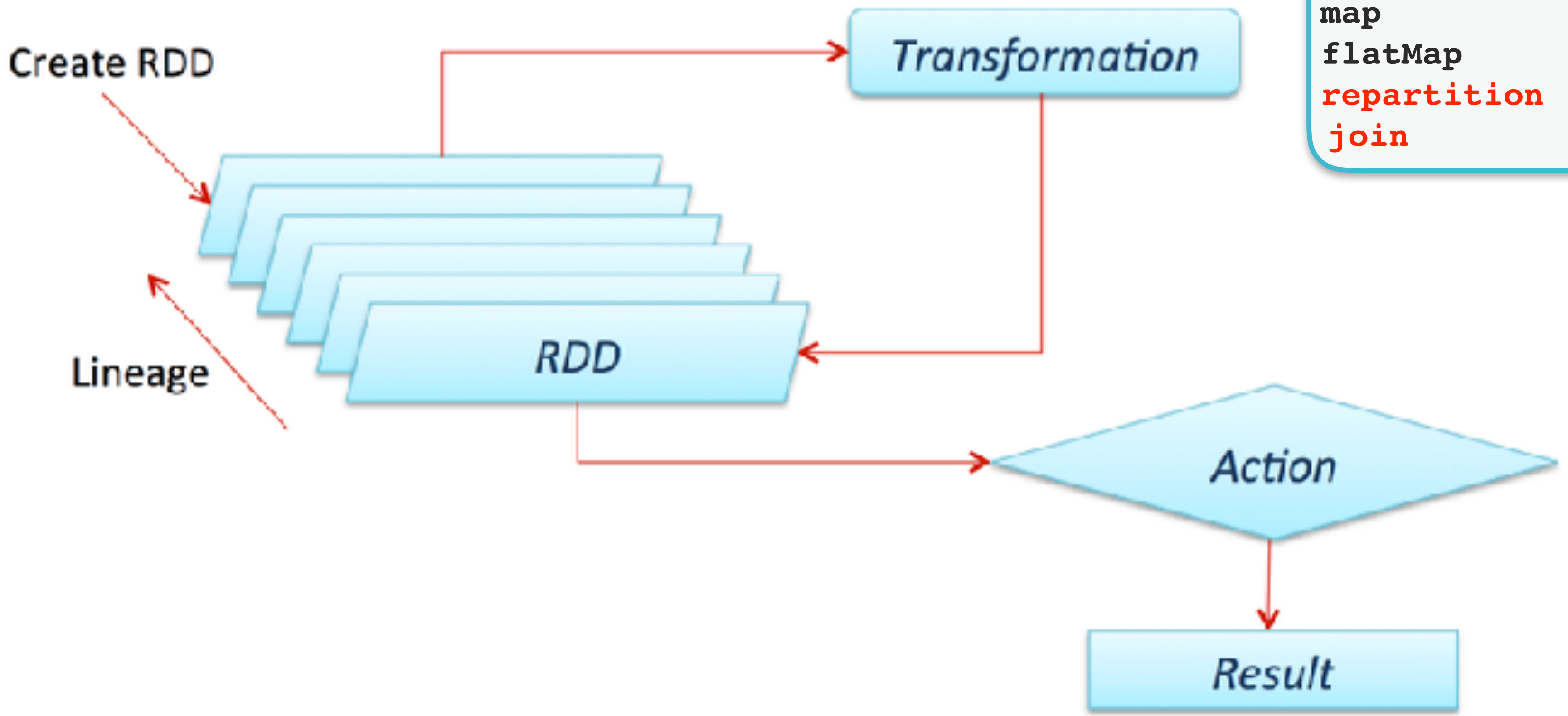
Spark: Jobs, Stages, Tasks




```
spark.sparkContext
  .textFile( path = "README.md")
  .flatMap(line => line.split( regex = " "))
  .map(word => (word, 1))
  .reduceByKey(_ + _)
```



Spark: Actions & Transformations




```
filter
map
flatMap
repartition
join
```

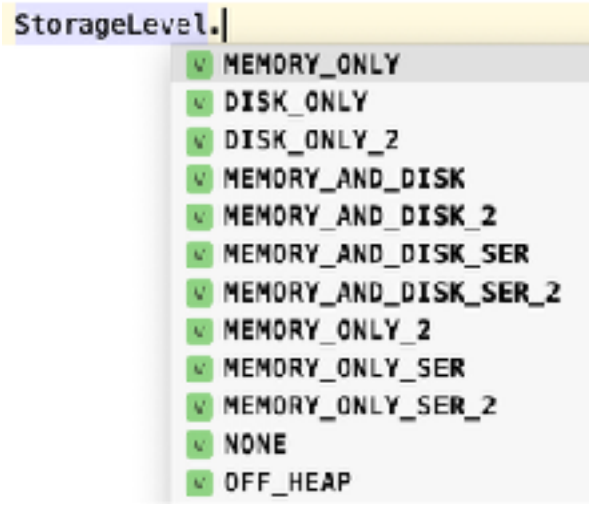
```
collect
count
foreachPartition
take
show
```

Spark: Persistance

Cache vs Persist

useful when data is accessed repeatedly
avoid re-evaluation

```
ds.cache [ds.persist(StorageLevel.MEMORY_ONLY)]
ds.persist(StorageLevel)
ds.unpersist()
```

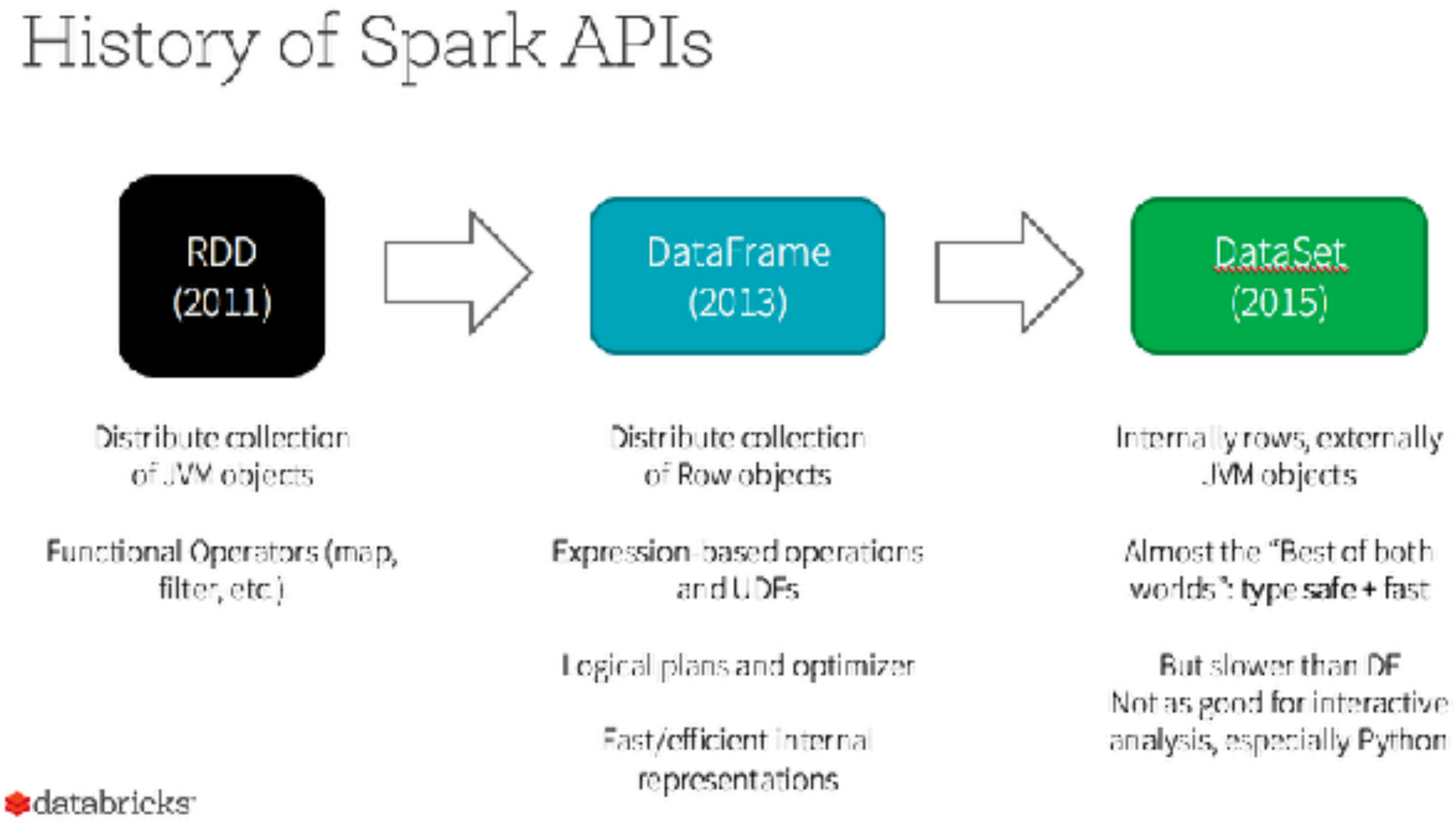


Checkpointing

allows a driver to be restarted on failure with
previously computed state of a distributed computation

```
SparkContext.setCheckpointDir(directory)
ds.checkpoint(eager or lazy)
```

Spark: From RDD to Dataset



Spark processing

Batch processing	Real time processing
Large group of data processed in a single run	Instantaneously data (events) processing
Entire data pre-selected and fed to the application	Stringent constrains in response time
Eg: Training data model	Eg: Prediction making

Spark SQL

Structured data processing

Extra optimisation by Spark: tungsten (memory management) + catalyst (query optimiser)

- SQL API
- Dataset API

Starting point: **`SparkSession`** *(Already available in the notebooks/spark-shell as:* `spark` *)*

```
import org.apache.spark.sql.SparkSession

val spark = SparkSession
  .builder()
  .appName( name = "Word count")
  .config("spark.some.config.option", "some-value")
  .getOrCreate()

// For implicit conversions like converting RDDs to DataFrames
import spark.implicits._
```

Spark Datasets

Distributed collection of data

Strongly typed

A Dataset can be constructed from JVM objects and then manipulated using functional transformations (`map`, `flatMap`, `filter`)

Encoders

API in Scala/Java

Spark Datasets

Spark DataFrames / SQL

```
DataFrame == Dataset[Row]


val df = spark.read.json("people.json")
df.printSchema()

//  DataFrame API
df.select($"name").show()

//  SQL API
df.createOrReplaceTempView("people")
spark.sql("SELECT name FROM people").show()
```

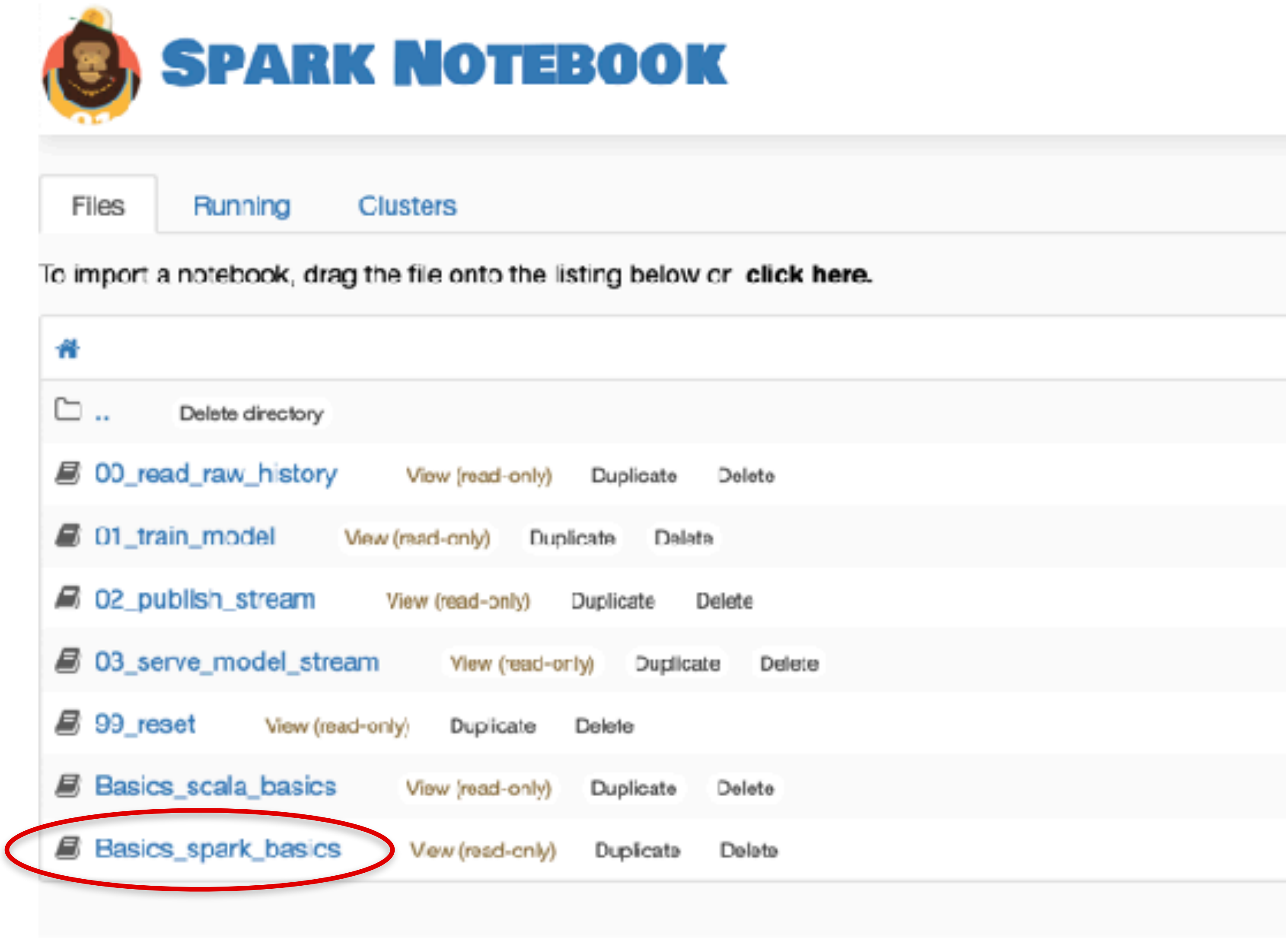
Spark DataFrame

Infer/Programatically define the Schema

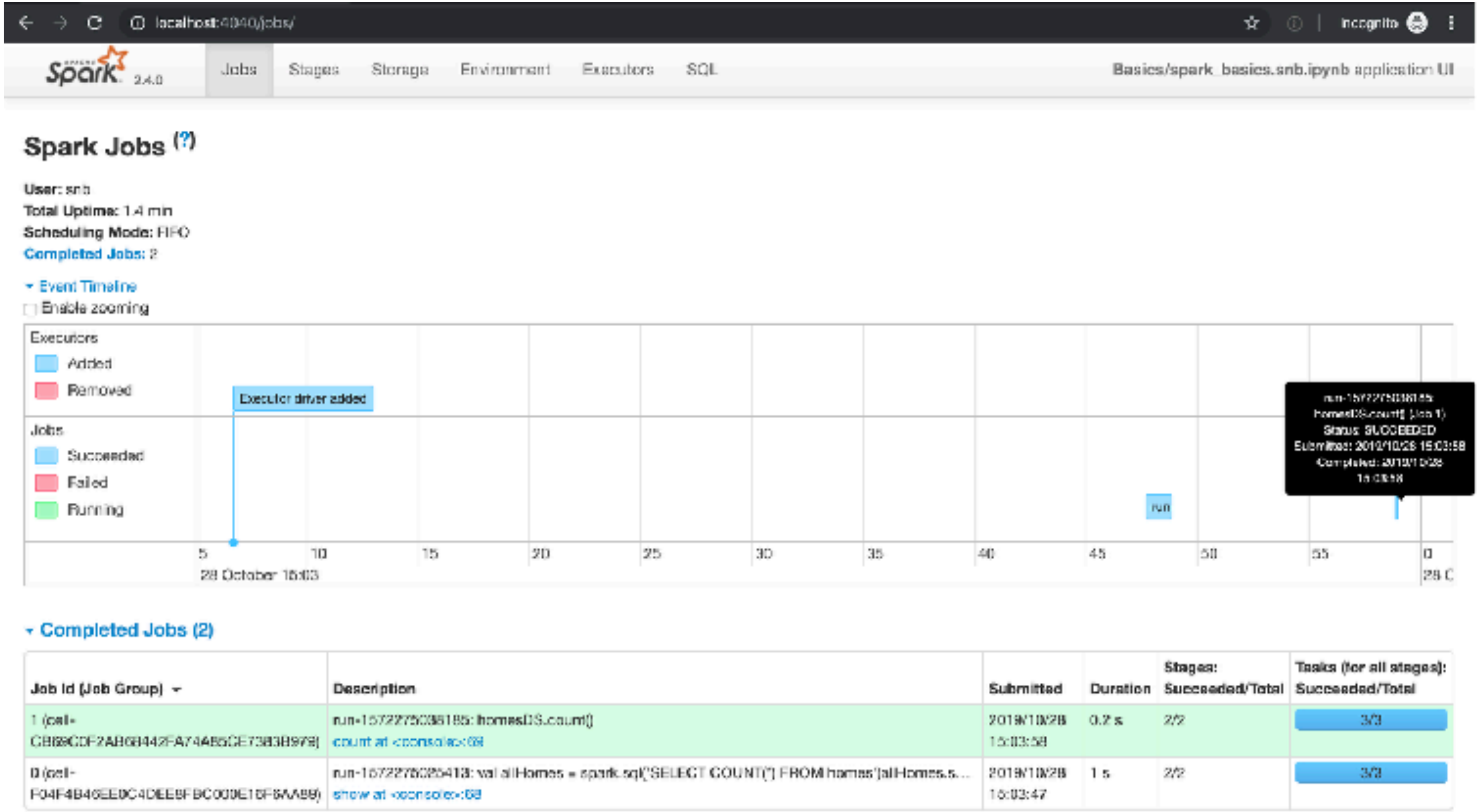
Untyped (Dataset[Row])

	SQL	DataFrames	Datasets
Syntax Errors	Runtime	Compile Time	Compile Time
Analysis Errors	Runtime	Runtime	Compile Time

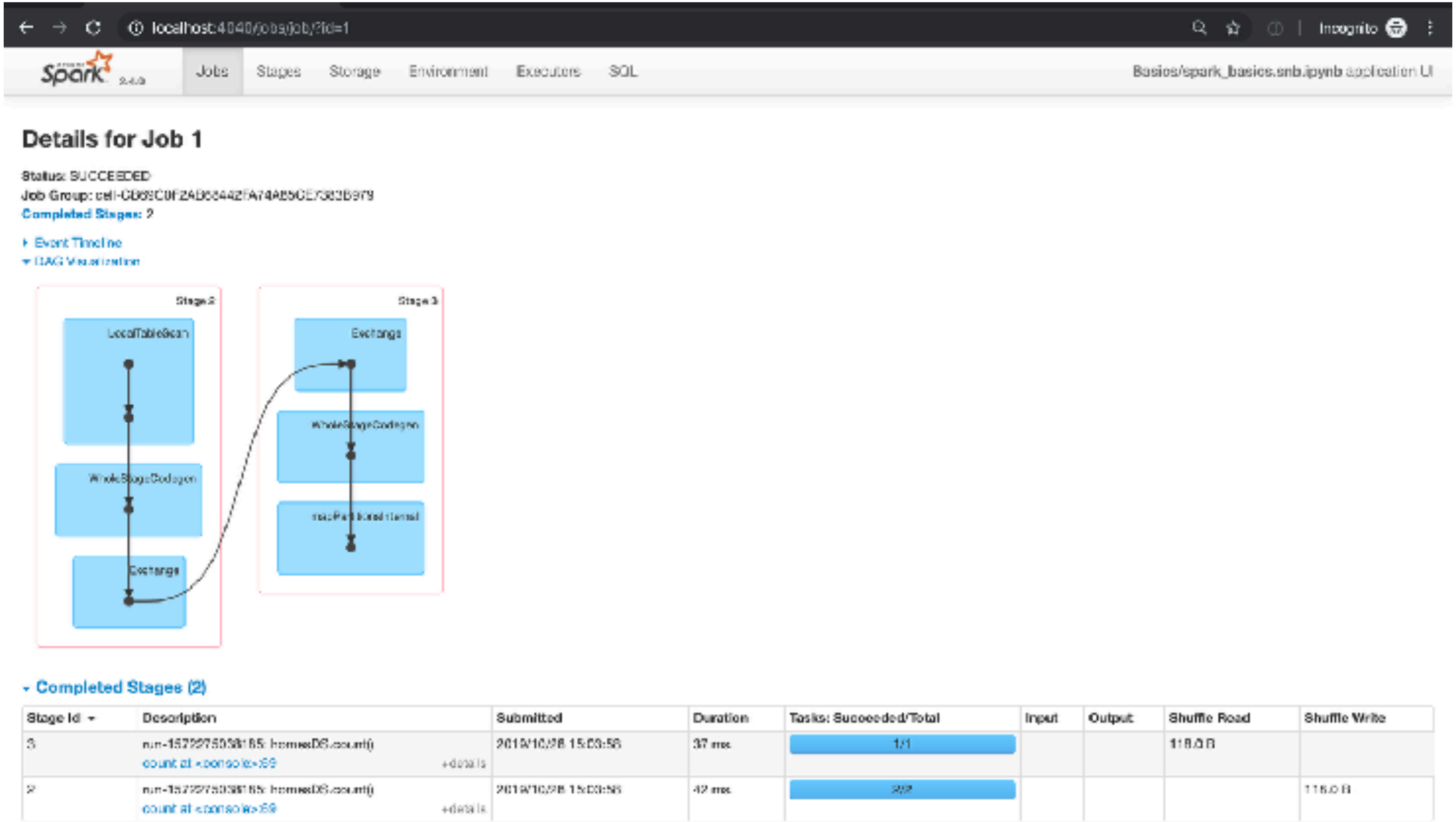
Phew…let's recap with a hands-on



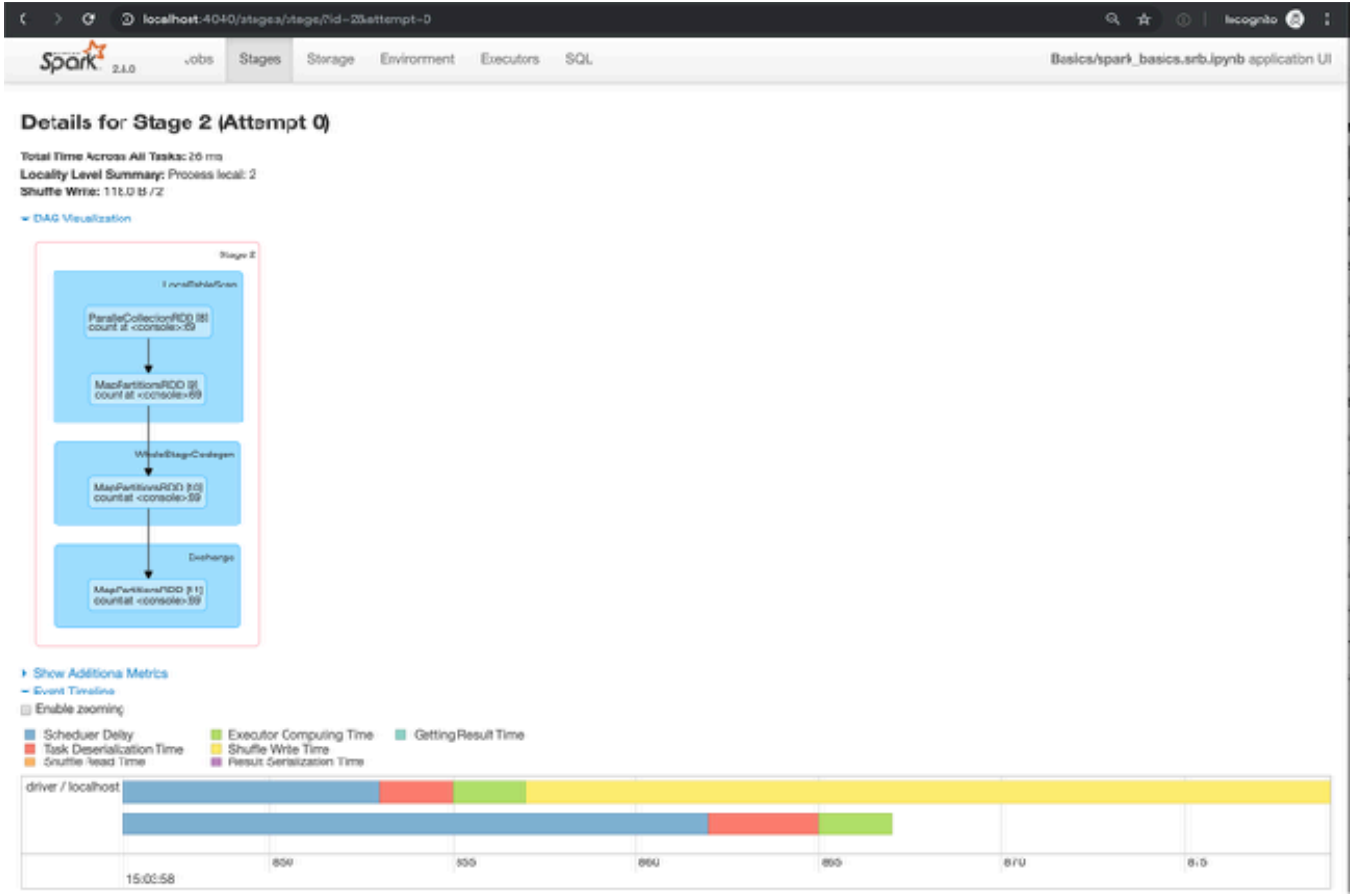
Spark: Monitoring



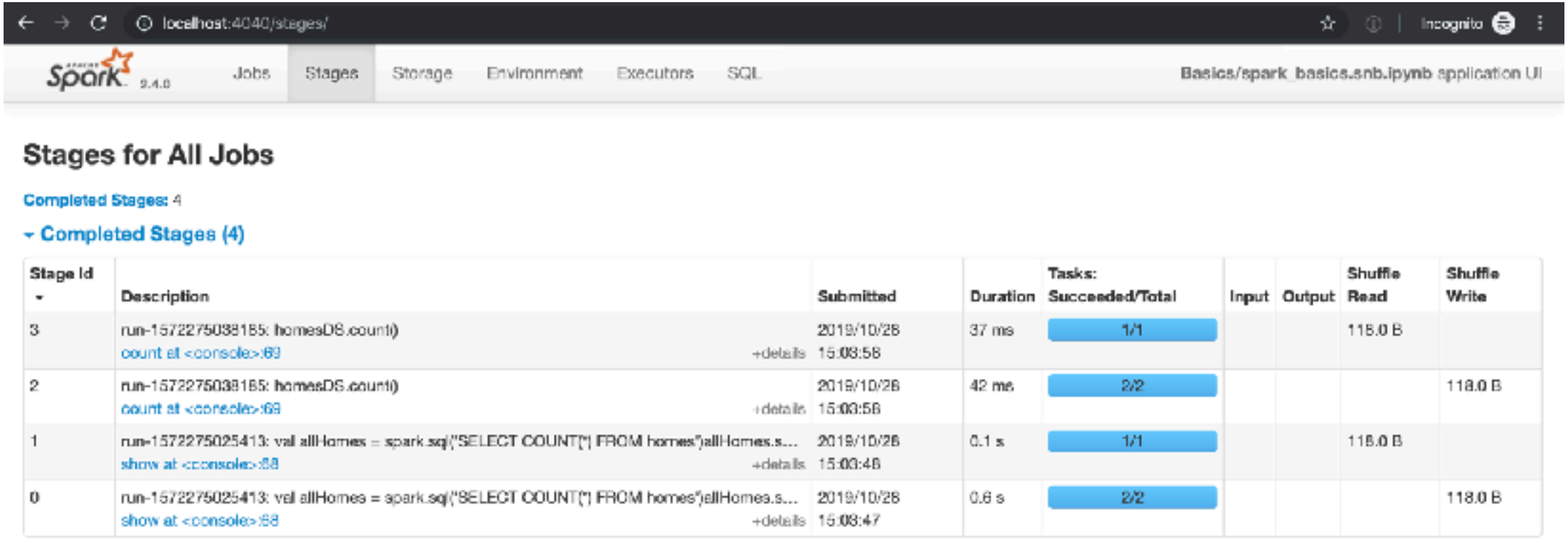
Spark: Monitoring



Spark: Monitoring



Spark: Monitoring



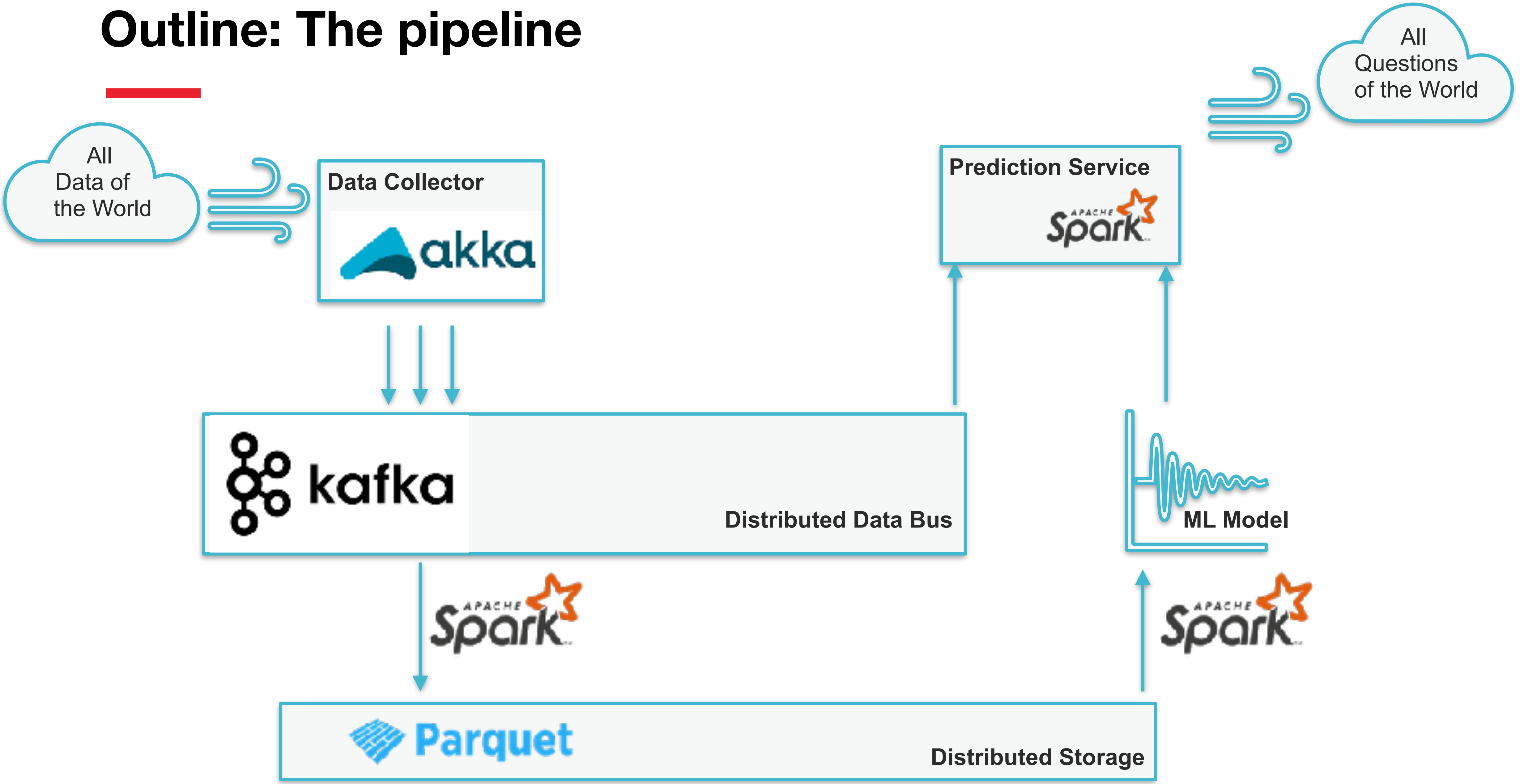
Stages for All Jobs

Completed Stages: 4

Completed Stages (4)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
3	run-1572275038155: homesDS.count() count at <console>:69 +details	2019/10/28 15:03:58	37 ms	1/1			118.0 B	
2	run-1572275038155: homesDS.count() count at <console>:69 +details	2019/10/28 15:03:58	42 ms	2/2				118.0 B
1	run-1572275025413: val allHomes = spark.sql("SELECT COUNT(*) FROM homes")allHomes.s... show at <console>:68 +details	2019/10/28 15:03:48	0.1 s	1/1			118.0 B	
0	run-1572275025413: val allHomes = spark.sql("SELECT COUNT(*) FROM homes")allHomes.s... show at <console>:68 +details	2019/10/28 15:03:47	0.6 s	2/2				118.0 B

Outline: The pipeline



Spark notebooks

Interactive Spark shell in a browser using http://spark-notebook.io/

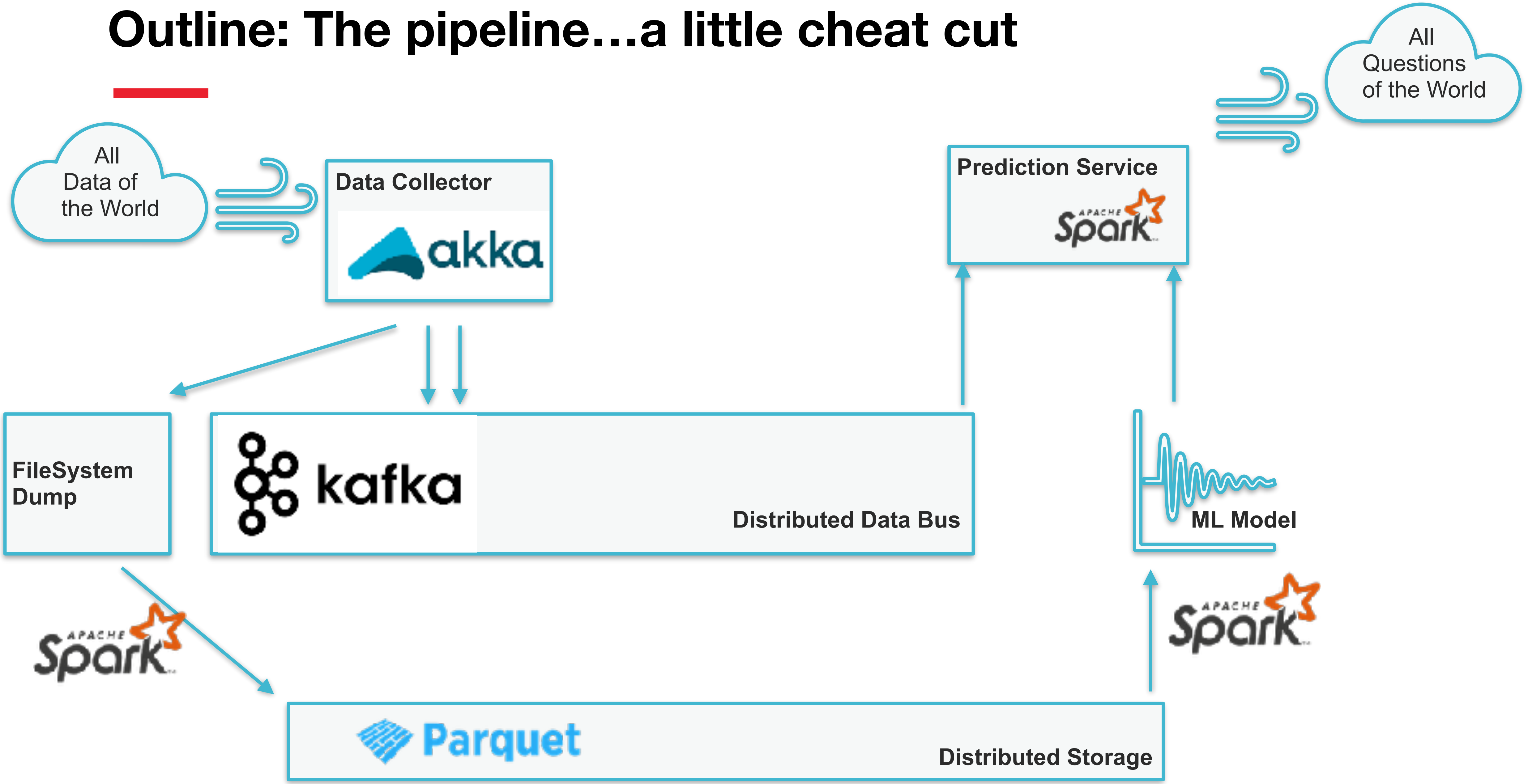
00_read_raw_history: Read historical data, do analysis, preprocessing and save results

01_train_model: Train linear model using preprocessed data

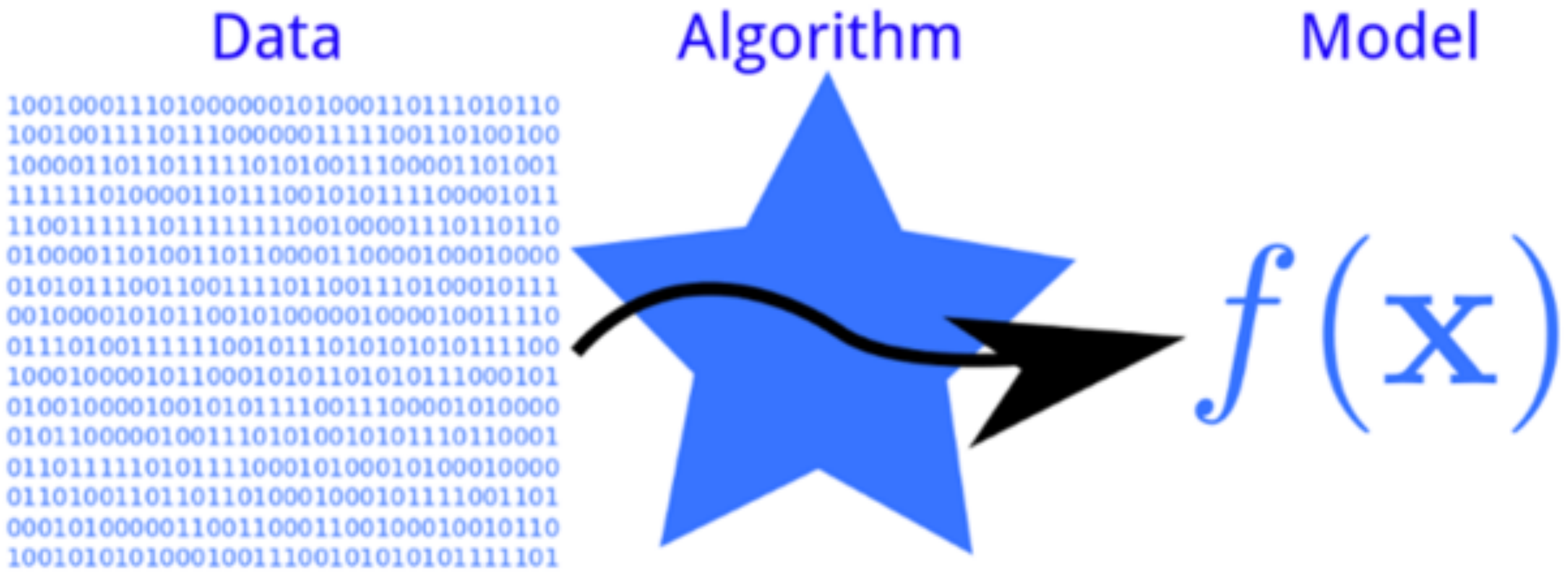
02_publish_stream: Generate a stream of data flowing in Kafka

03_serve_model_stream: Read data from Kafka and make predictions using our model

Outline: The pipeline…a little cheat cut



ML Introduction



ML Introduction

Data as a flat table

```scala
type Feature = Double
type Label   = Double

val dataSet: Seq[ (Vector[Feature], Label) ]
```

Surface	Land	Beds	Sidings
110	896	2	4
120	435	3	2
150	210	4	3
170	713	4	4
80	231	4	4
90	238	3	4
130	118	2	3
146	695	4	4
155	644	4	4

Price
160
189
250
240
179
135
175
169
189

ML Introduction

A model is function a representing a facet of the data

```
val model: Vector[Feature] => Label
```

Surface	Land	Beds	Sidings
110	896	2	4

=>

Price
160

ML Introduction

Learning a Model from Data

```
val train: Seq[ (Vector[Feature], Label)] =>  Vector[Feature] => Label
```

Surface	Land	Beds	Sidings
110	896	2	4
120	435	3	2
150	210	4	3
170	713	4	4
80	231	4	4
90	238	3	4
130	118	2	3
146	695	4	4
155	644	4	4

Price
160
189
250
240
179
135
175
169
189

=>

Surface	Land	Beds	Sidings
110	896	2	4

=>

Price
160

ML Introduction

Training by Minimizing Errors (Loss), e.g. sum of squared errors:

```
val loss = dataSet.map{
    case (x, y) => y - model(x)
    }
    .map(Math.pow(_, 2))
    .reduce( _ + _ )
```

Surface	Land	Beds	Sidings
110	896	2	4
120	435	3	2
150	210	4	3
170	713	4	4
80	231	4	4
90	238	3	4
130	118	2	3
146	695	4	4
155	644	4	4

Price
160
189
250
240
179
135
175
169
189

Price
160
189
250
240
179
135
175
169
189

$$Loss = \sum_i (y_i - \hat{y}_i)^2$$

Missing pieces yet: How a model is built? What is ‘minimizing?

ML Introduction

Models as a vector of parameters

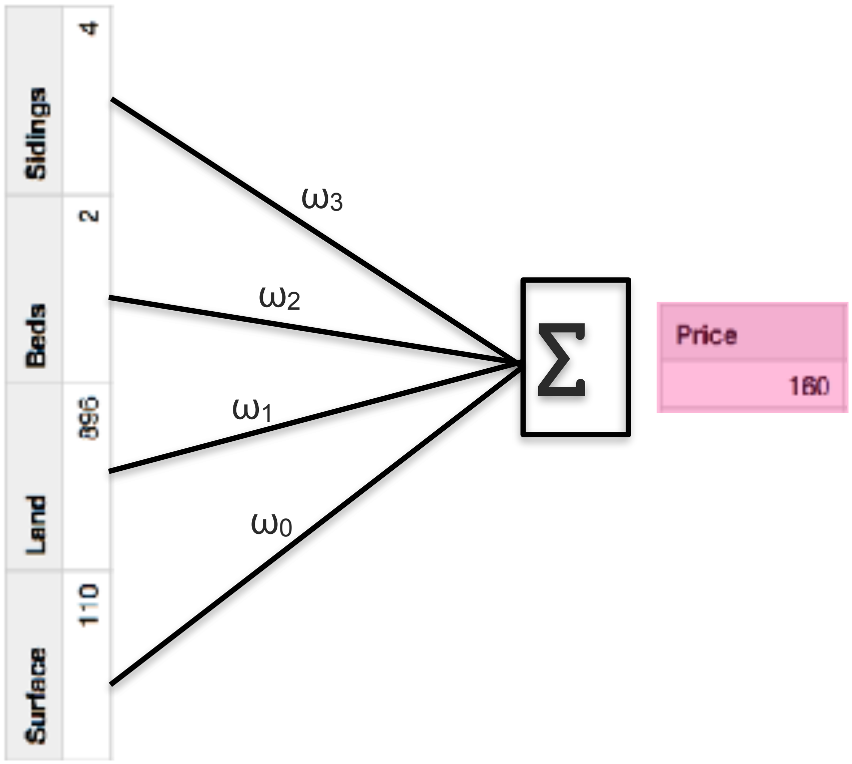
A model is a function, with some parameters, optimisation is finding the best parameters…

Example: A **Linear model** is a linear combination of features:

```
val params: Vector[Double]
val bias: Double

val model = x: Vector[Feature] =>
                   x.zip(params).map( case(wi, xi) => wi * xi )
                                 .reduce( _ + _ )
                 + bias
```

$$\hat{y} = \sum_i x_i \omega_i$$



ML Introduction

Optimisation algorithms

Gradient based methods: How loss varies with each parameters ~ gradient ()

$$\Delta Loss \sim \Delta\omega_i$$

$$\omega_i^* = \omega - \gamma \frac{\Delta Loss}{\Delta\omega_i}$$

Loss and gradient are estimated on a subset of data (a batch) = stochastic gradient based methods

Iterations in batches and epochs (a full dataset pass)

ML Introduction

Metrics

After training: model evaluation

E.g.

Root Mean Squared Error in regression

Accuracy in classification (% correct binary prediction)

Metrics are used for model validation on test data not used in training

Surface	Land	Beds	Sidings	Price
110	896	2	4	160
120	435	3	2	189
150	210	4	3	250
170	713	4	4	240
80	231	4	4	179
90	238	3	4	135
130	118	2	3	175
146	695	4	4	169
155	644	4	4	189

ML Introduction

Data is multidimensional **Arrays** of **Floating** point values

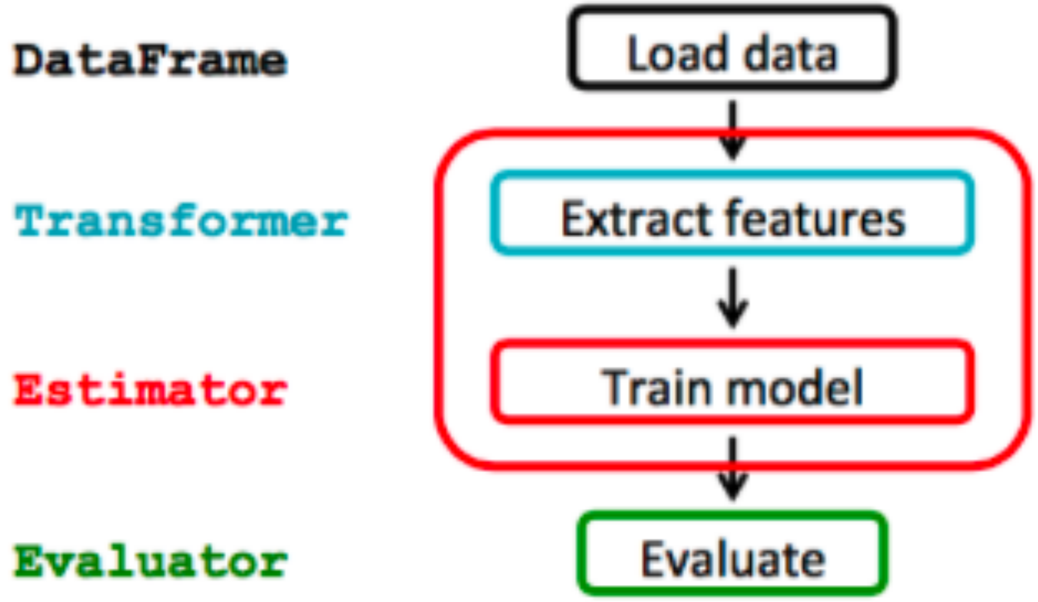
Models are represented as **Arrays** of **Floating** point values and operators

Training, **Evaluating** and **Inference** on models are **operations** on these arrays

Spark ML concepts

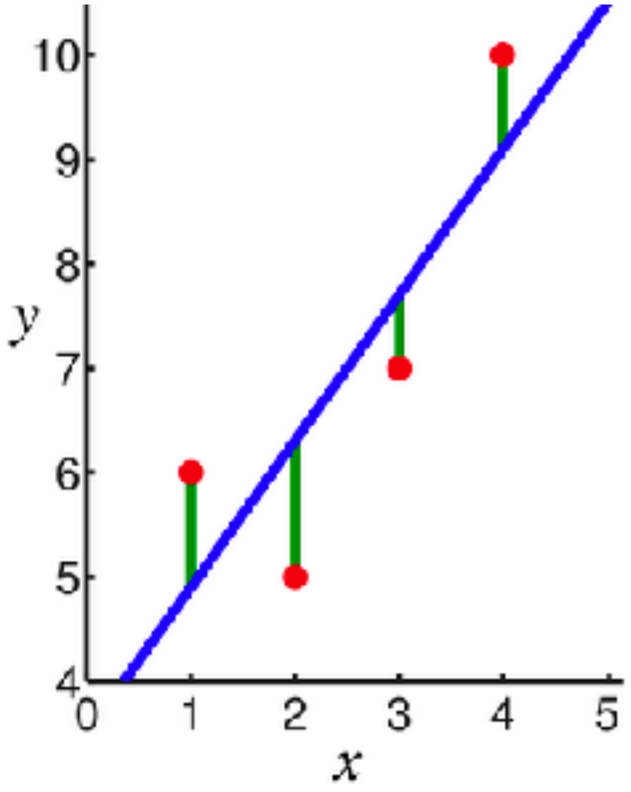
Pipeline (Sequence of `PipelineStages`):

- **Transformers**: Read a DataFrame, select a column, map it into a new column. Output is a new DataFrame with the mapped column appended.
- **Estimators**: Produce a Model from a given DataFrame (Transformer)



Linear Regression

In linear regression, the observations (**red**)
are assumed to be the result of random deviations (**green**)
from an underlying relationship (**blue**)
between a dependent variable (*y*)
and an independent variable (*x*).



Spark notebooks

Interactive Spark shell in a browser using http://spark-notebook.io/

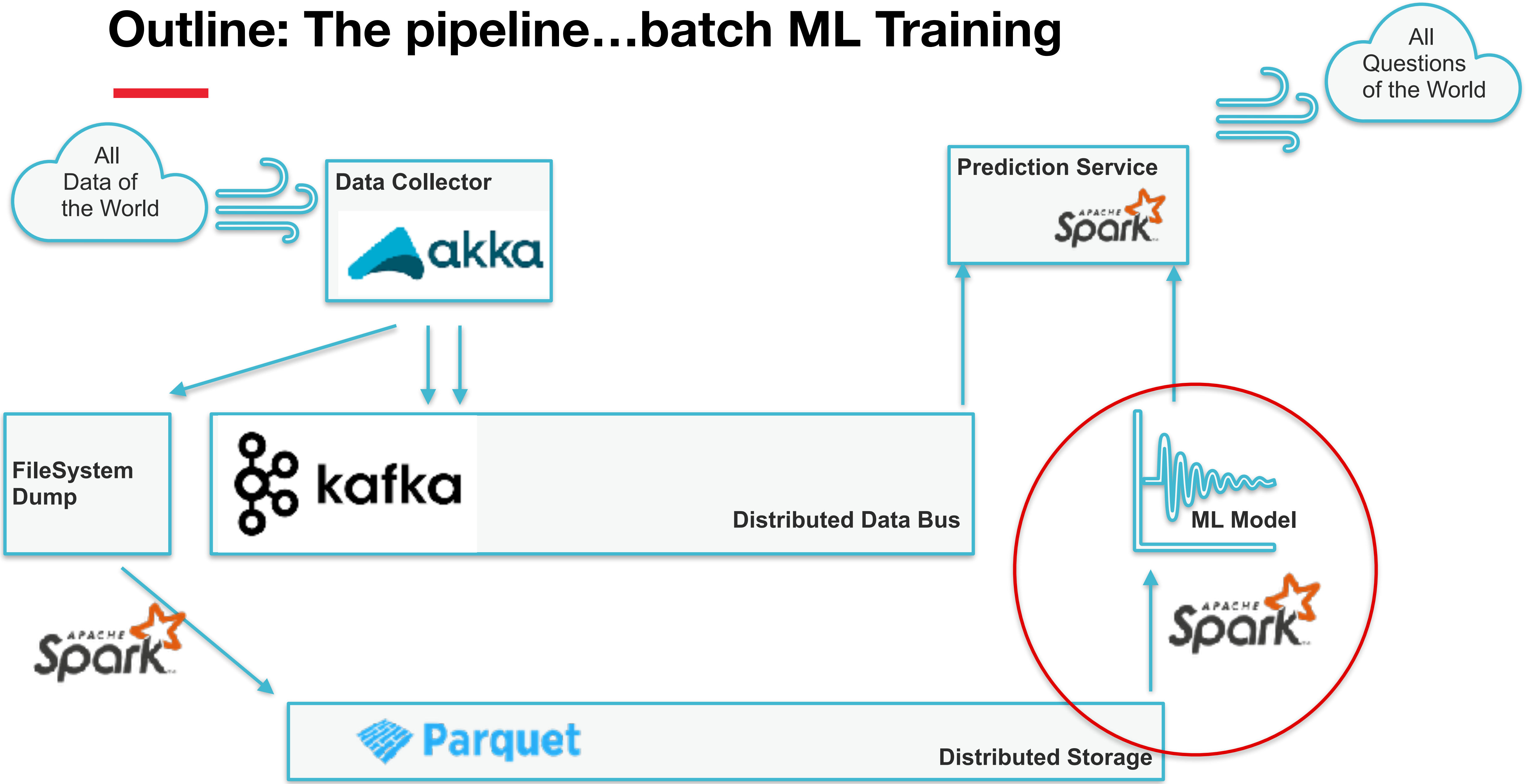
00_read_raw_history: Read historical data, do analysis, preprocessing and save results

01_train_model: Train linear model using preprocessed data

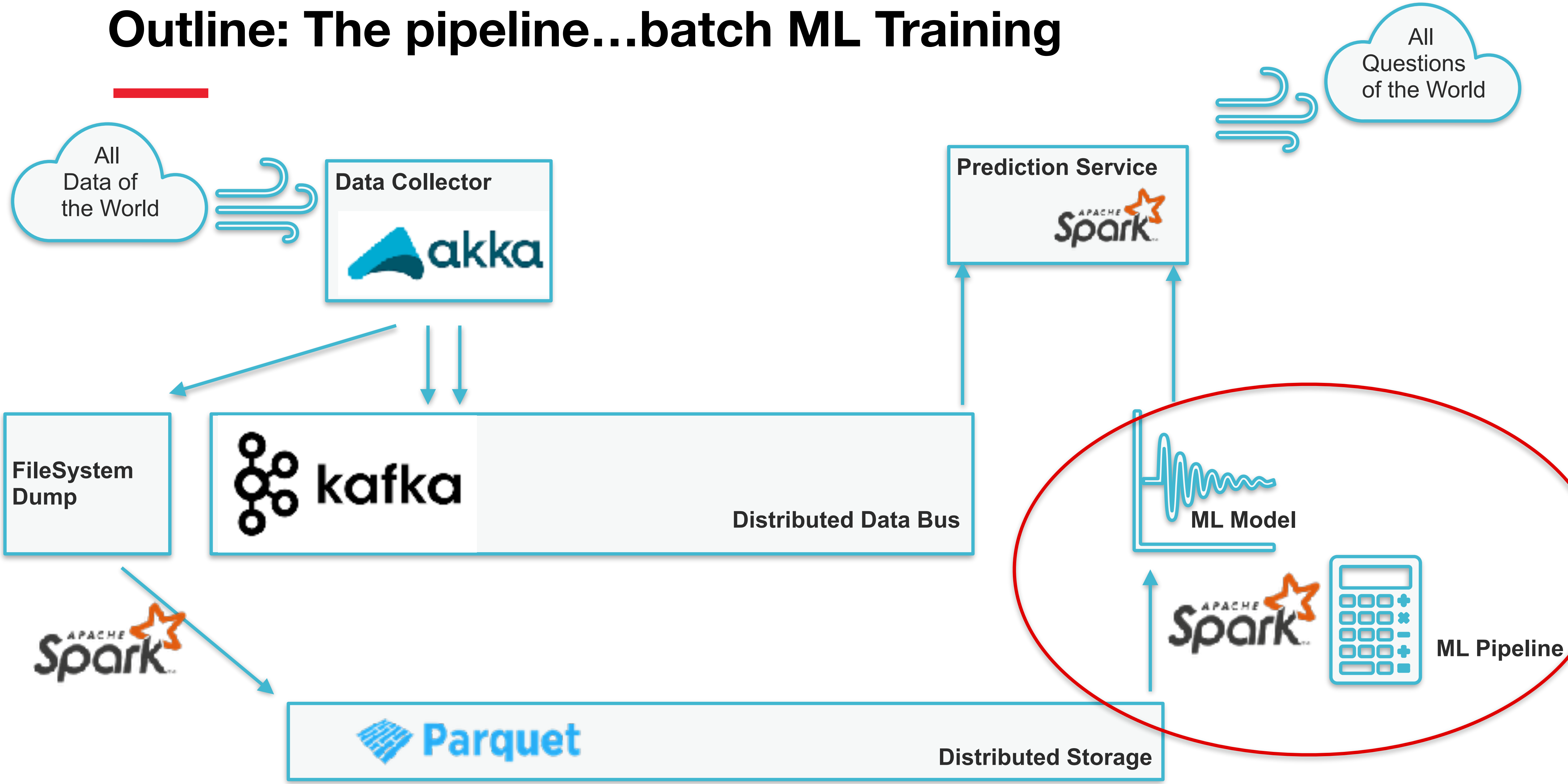
02_publish_stream: Generate a stream of data flowing in Kafka

03_serve_model_stream: Read data from Kafka and make predictions using our model

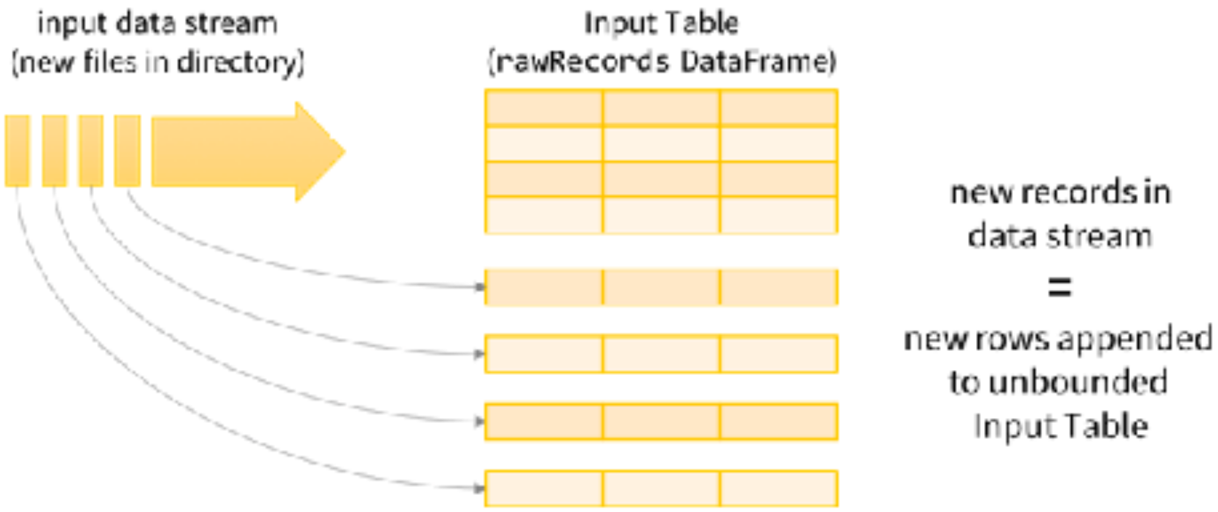
Outline: The pipeline…batch ML Training



Outline: The pipeline…batch ML Training



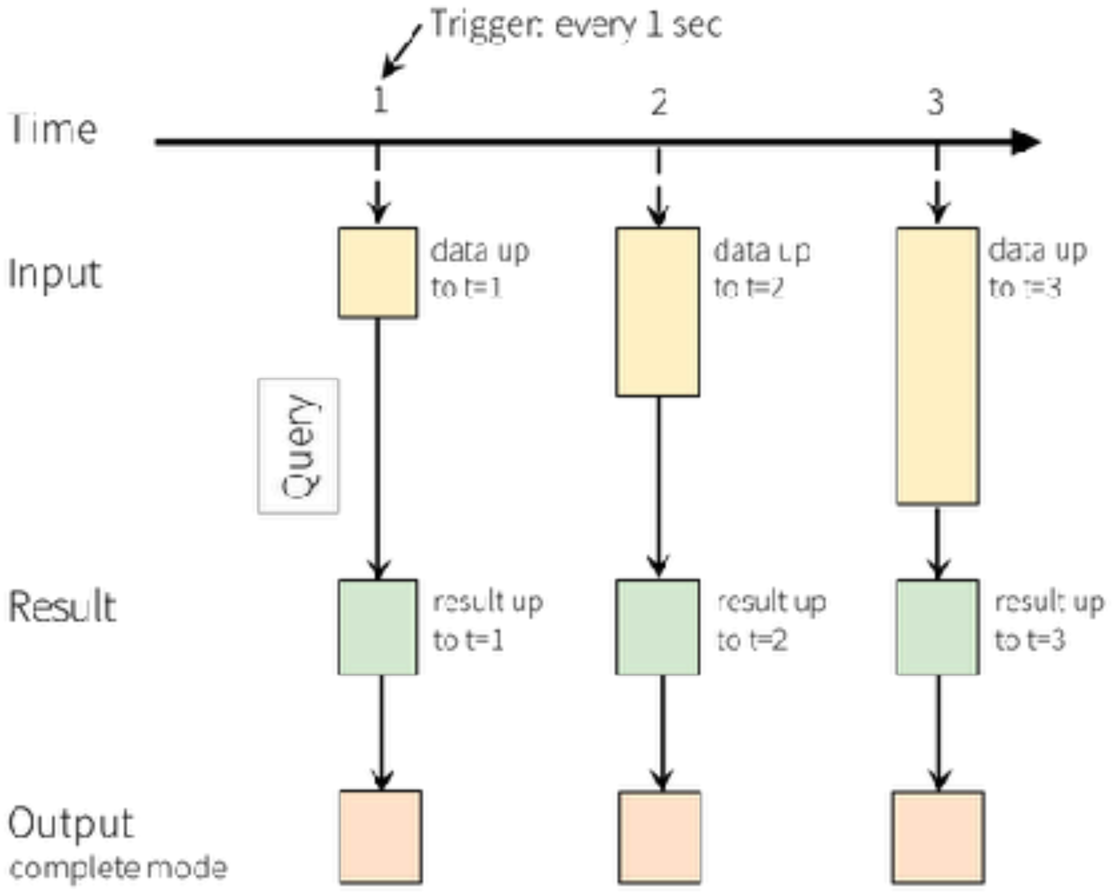
Spark Structured Streaming



Structured Streaming Model

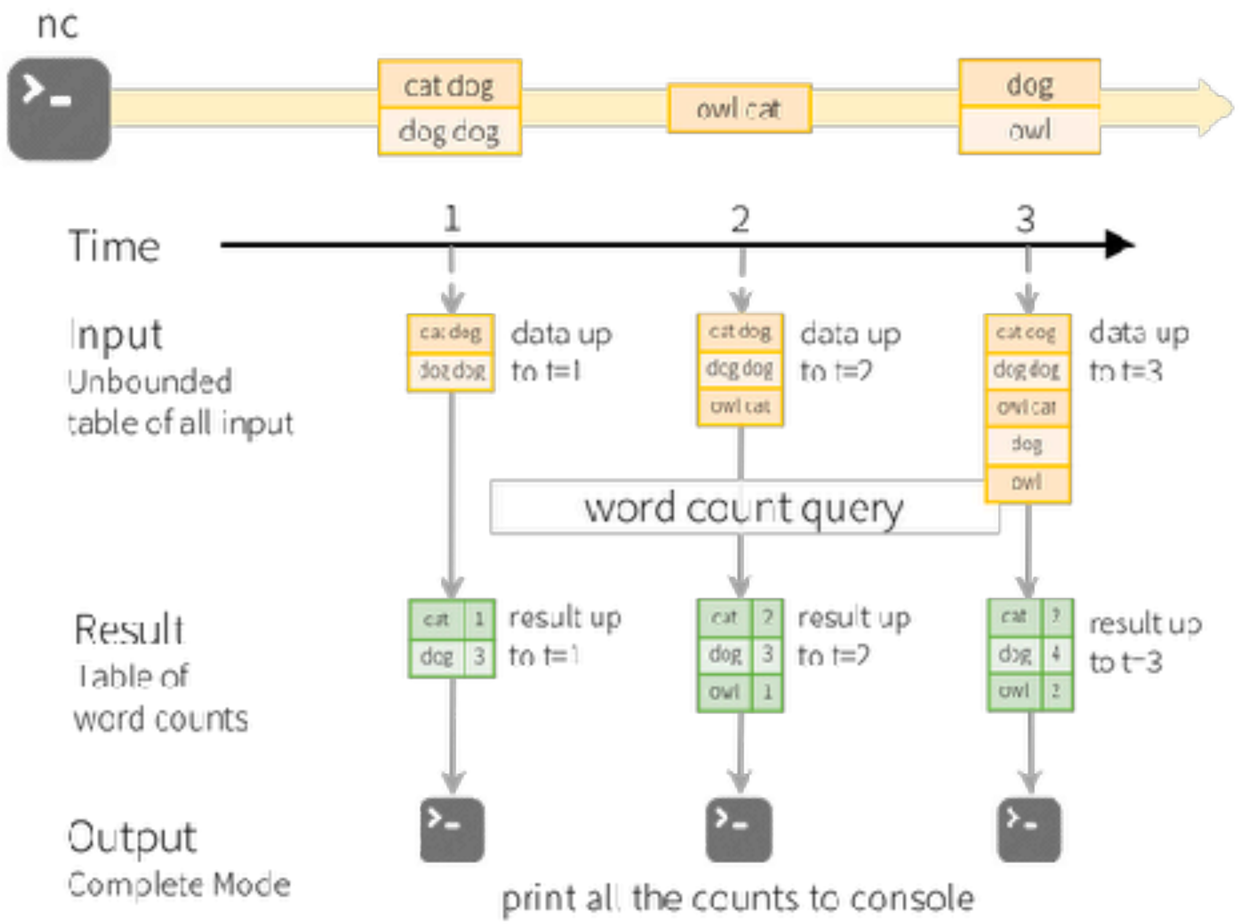
treat data streams as unbounded tables

Spark Structured Streaming



Programming Model for Structured Streaming

Spark Structured Streaming



Spark Structured Streaming

```scala
val df = spark.readStream
  .format( source = "kafka")
  .option("kafka.bootstrap.servers", "localhost:9092")
  .option("subscribe", "topic1")
  .load()

val processedDF: DataFrame = ???

processedDF.writeStream
  .queryName( queryName = "predictions")
  .outputMode( outputMode = "append")
  .format( source = "memory")
  .start()
```

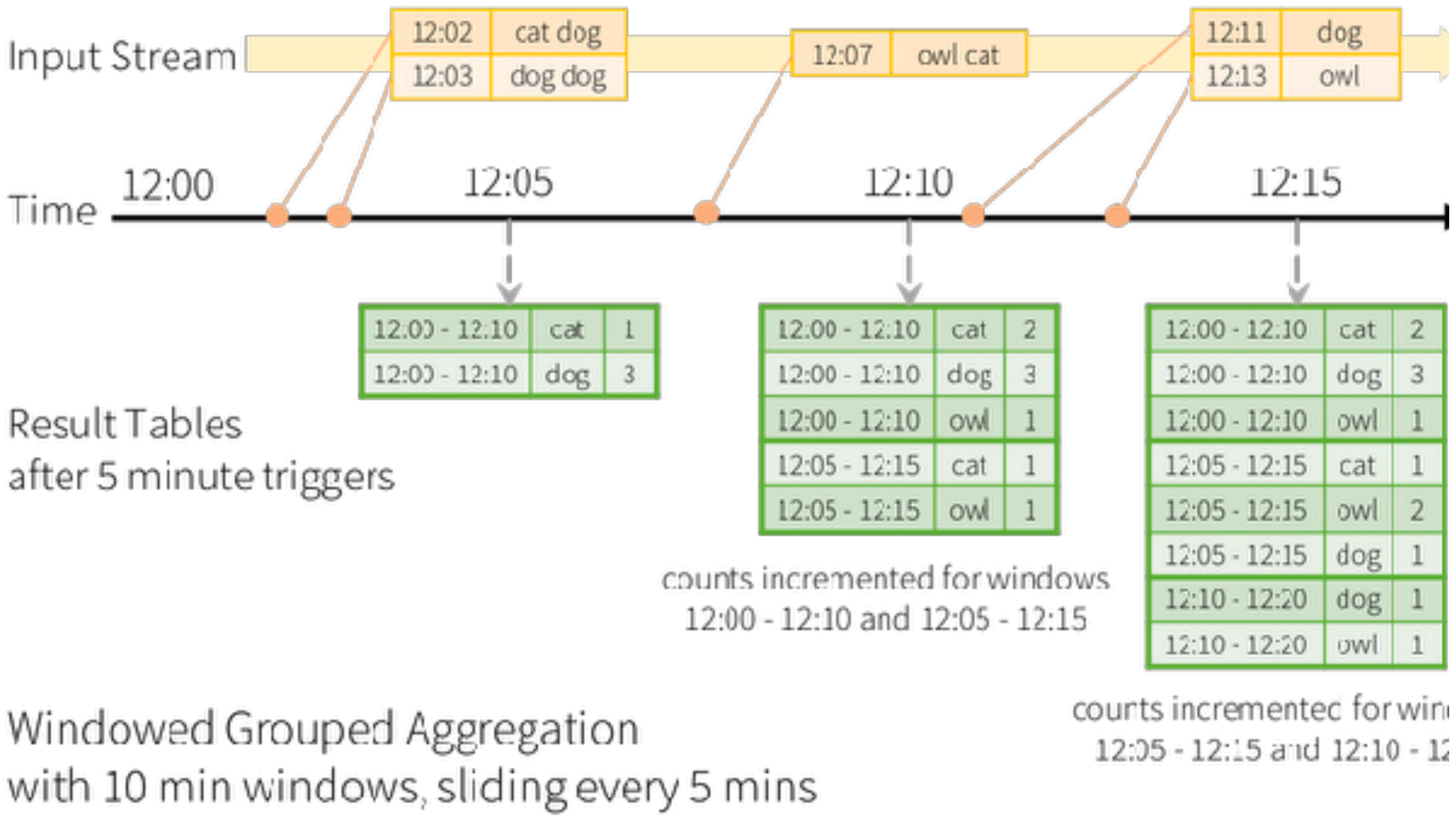
Spark Structured Streaming

Event time vs reception time

Analyse the event based on when it was generated (instead of when it arrived to the system). Extra column with the event time.

Window-based aggregations => grouping and aggregation on the event-time column

Spark Structured Streaming



Spark Structured Streaming

Late data arrival - Watermarking

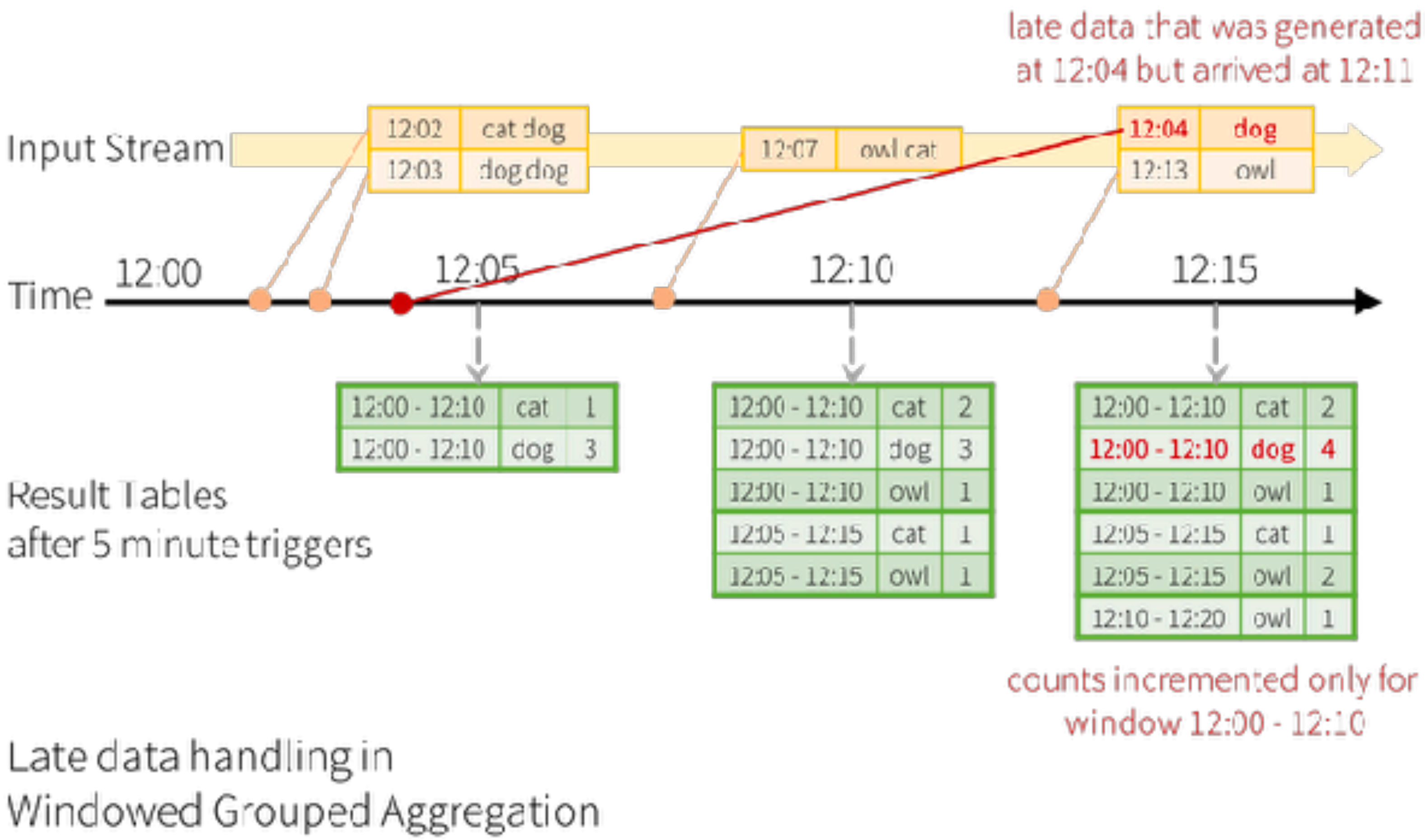
Processing is based on event time.

Spark allows processing events arriving late.

Set a limit with the watermark

```
val windowedCounts = words
  .withWatermark("timestamp", "10 minutes")
  .groupBy(
    window($"timestamp", "10 minutes", "5 minutes"),
    $"word")
  .count()
```

Spark Structured Streaming



Spark notebooks

Interactive Spark shell in a browser using http://spark-notebook.io/

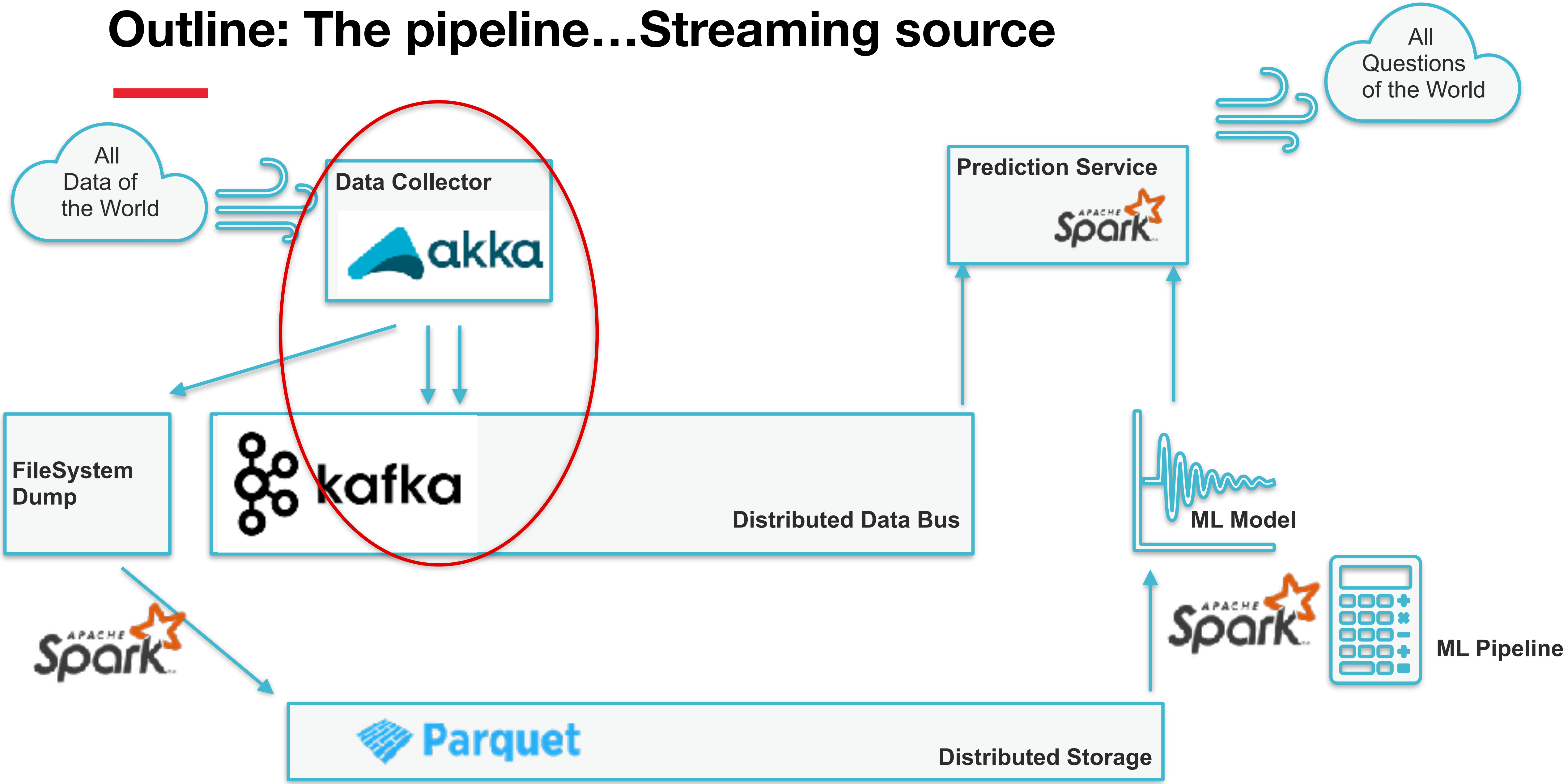
00_read_raw_history: Read historical data, do analysis, preprocessing and save results

01_train_model: Train linear model using preprocessed data

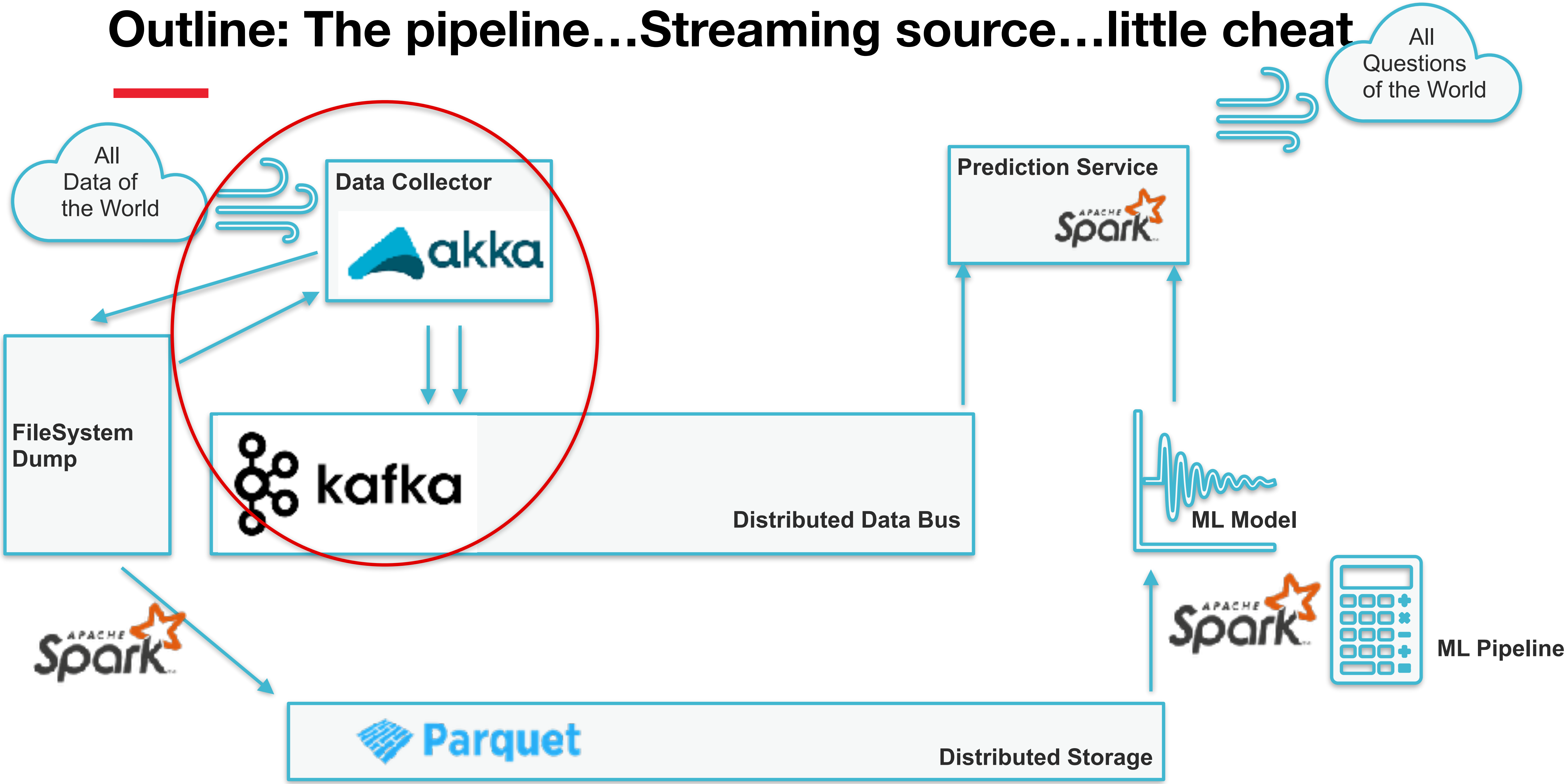
02_publish_stream: Generate a stream of data flowing in Kafka

03_serve_model_stream: Read data from Kafka and make predictions using our model

Outline: The pipeline…Streaming source



Outline: The pipeline…Streaming source…little cheat



Spark notebooks

Interactive Spark shell in a browser using http://spark-notebook.io/

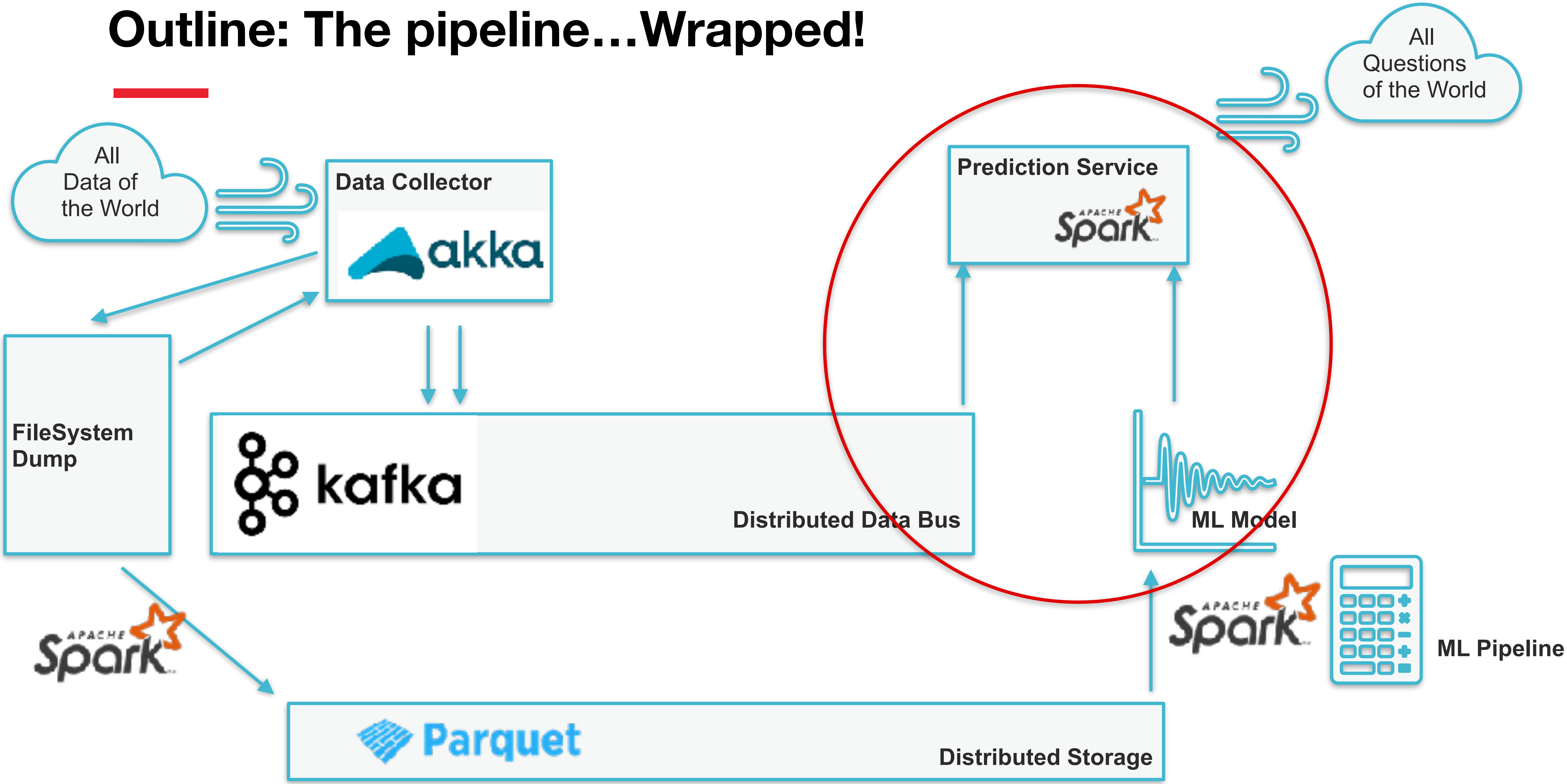
00_read_raw_history: Read historical data, do analysis, preprocessing and save results

01_train_model: Train linear model using preprocessed data

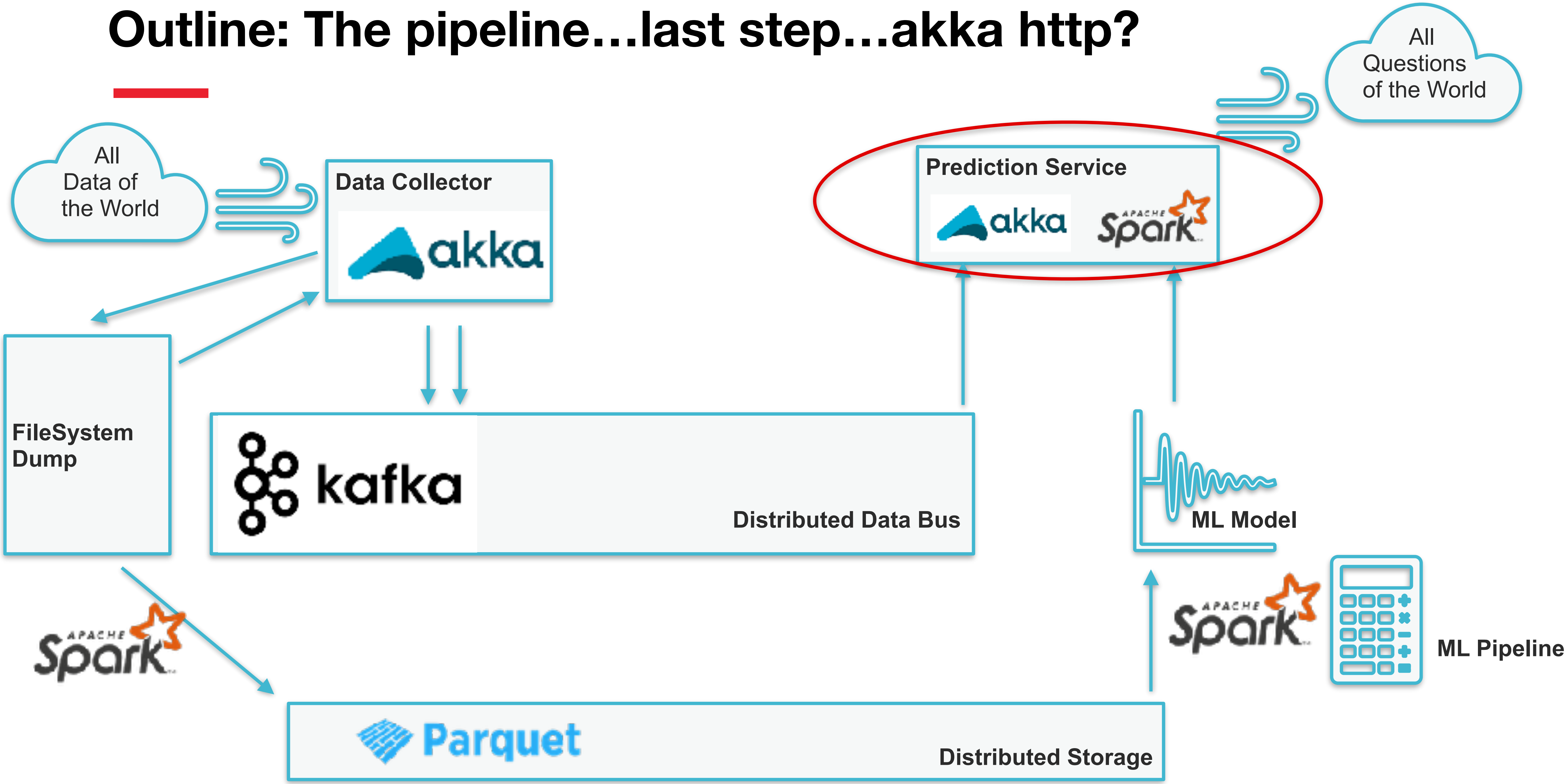
02_publish_stream: Generate a stream of data flowing in Kafka

03_serve_model_stream: Read data from Kafka and make predictions using our model

Outline: The pipeline…Wrapped!



Outline: The pipeline…last step…akka http?



Merci !

xavier.tordoir@lunatech.nl
twitter.com/xtordoir

maria.dominguez@lunatech.nl

www.lunatech.com

LUNATECH
SIMPLIFY YOUR IT