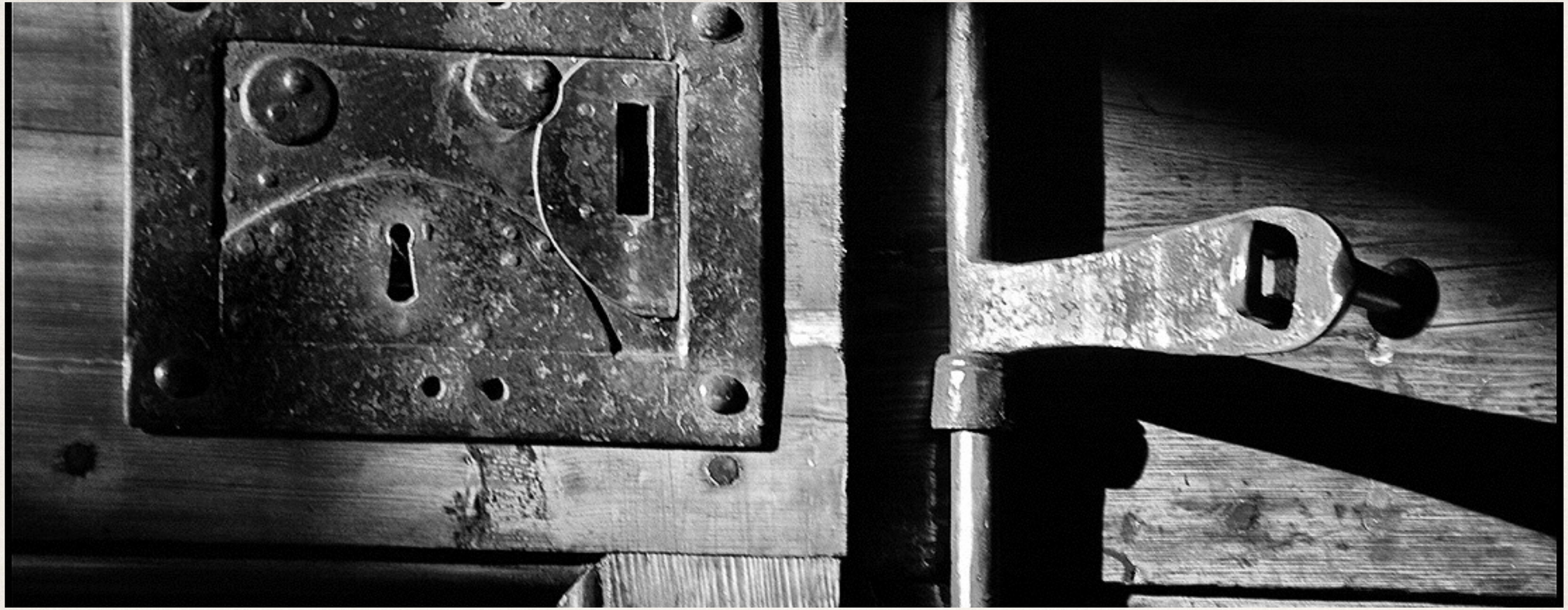


Photo Credit: Sergio Bertolini

Unlocking Our Health Data

Transforming Unstructured Data at Scale

Ola Wiberg, April 24, 2015

Human API

- Launched Summer 2013
- Based in Redwood City, CA
- Building a unified health data platform
- Enable next generation health applications
- Easy integration of health data from “anywhere”

What problems are we solving?

The Old and the New

- Health data lives in silos (both old and new)
  - Old “legacy systems” have a number of technical challenges for data access.
  - New systems are technically easier with modern APIs and JSON data.
  - No universal standards
- *We are enabling developers to work with data from any type of system using one API.*

(Photo credit: Pablo Fernández)

What are the challenges?

Old Systems

- Hospitals, Labs, Clinical trials, Medicare
- Often unstructured & semi-structured data
- Lots of data for points in time
- Too many standards => no standard

New Systems

- Health Tracking (Fitbit, Jawbone, Apple Watch, WellnessFX)
- Structured, but everyone have their own structure
- Continuous streams of data
- No standards

Health IT Standards for Legacy Systems

- Terminology standards
  - Code sets (LOINC, CVX)
  - Classification Systems (ICD-10)
  - Nomenclature (SNOMED, Omaha System)
- Information exchange standards
  - HL7 (CDA, V2, V3, ***FHIR***)

Legacy Data

- Clinical Data
- Medical Records
- Vitals recorded by a nurse
- At best XML over TCP/HTTPS
- *Deep data for specific points in time*

Modern Data

- Personal Health Tracking
- Fitness
- Vitals recorded by a user's device
- JSON over RESTful APIs
- *Continuous streams of data*

The good parts

- Both old and new systems provide a lot of valuable data
- Very useful once computers can make sense of it all
- The new data is filling the gaps of the data being collected in legacy systems
- Both of them together will create a more complete health profile of a person

We are building a system that enables deep learning

from these disparate data sources

How do we build it?

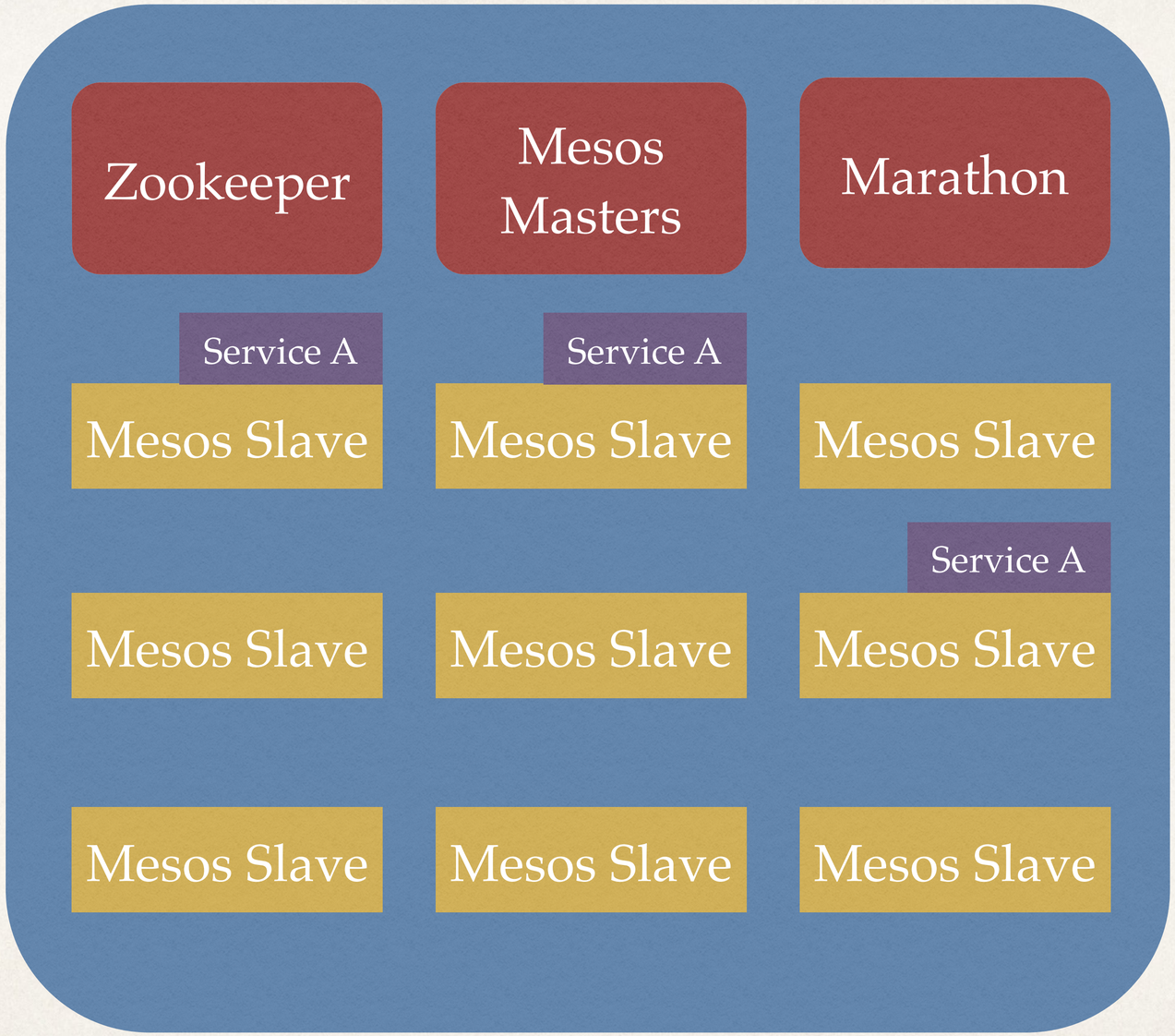
System Overview

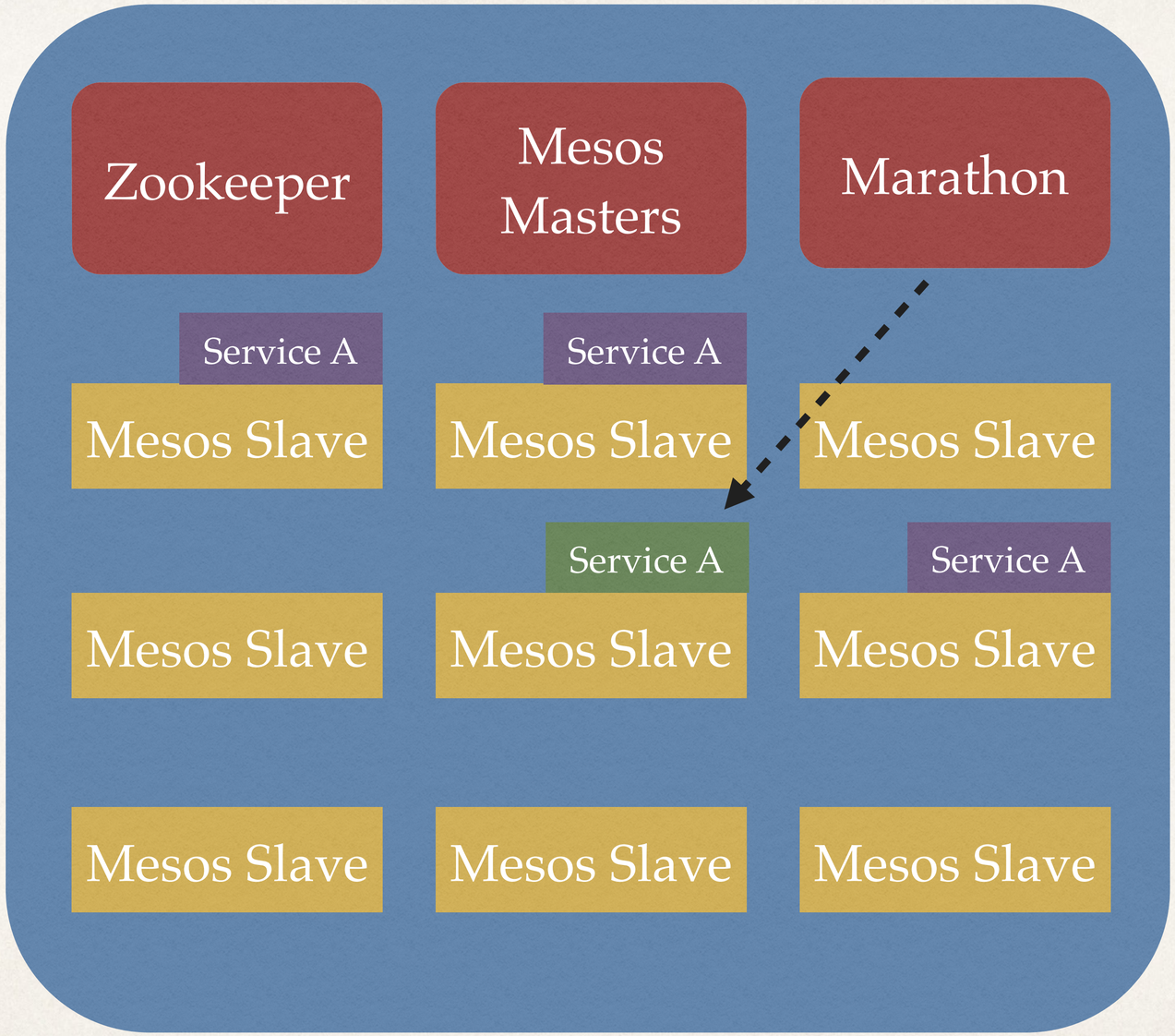
- Many incoming data sources (push and pull)
- Many different data types (standards/models)
- Process data in near real-time (from the time data enters our system)
- Flexibility and speed of development are essential

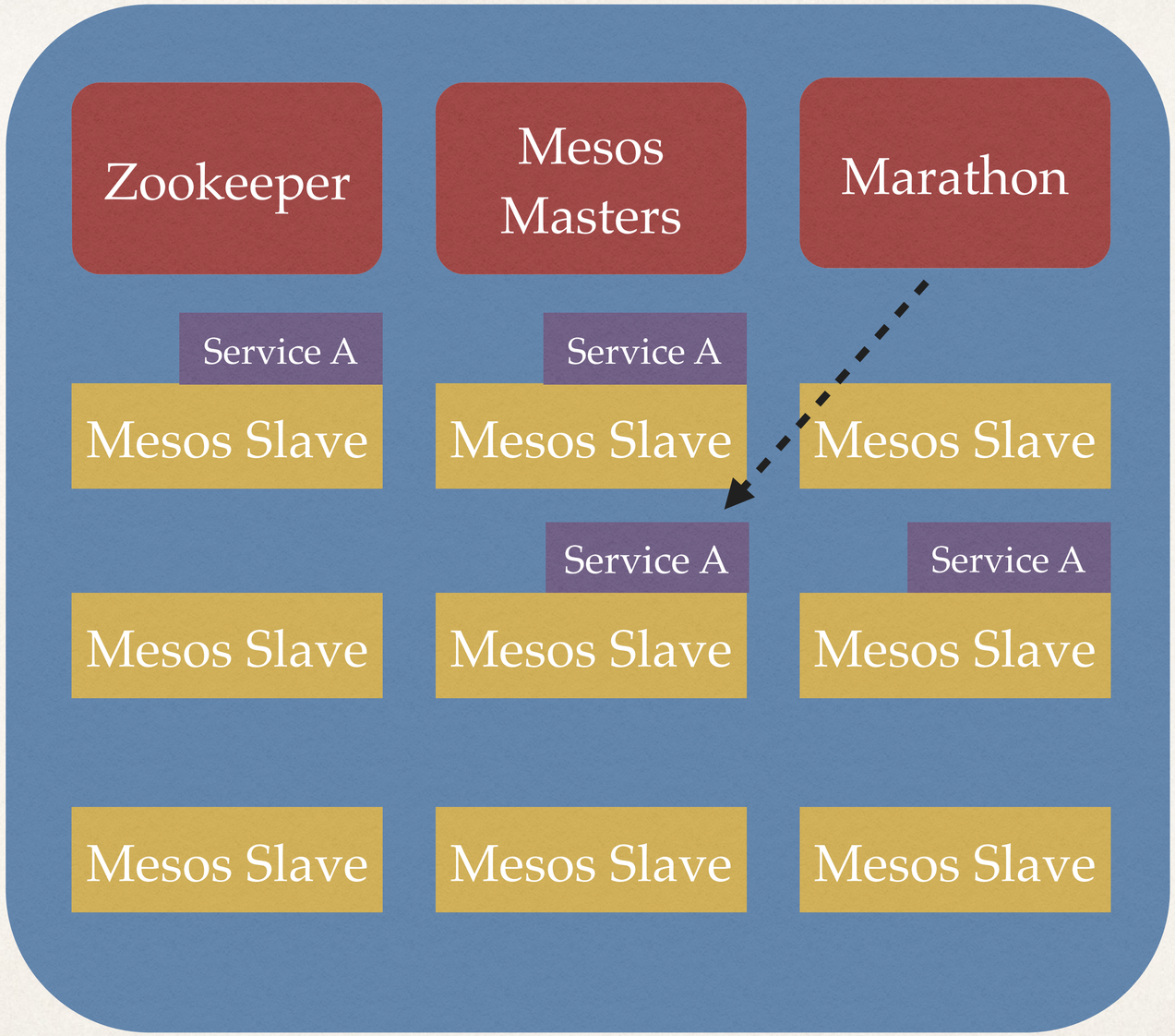
Our infrastructure

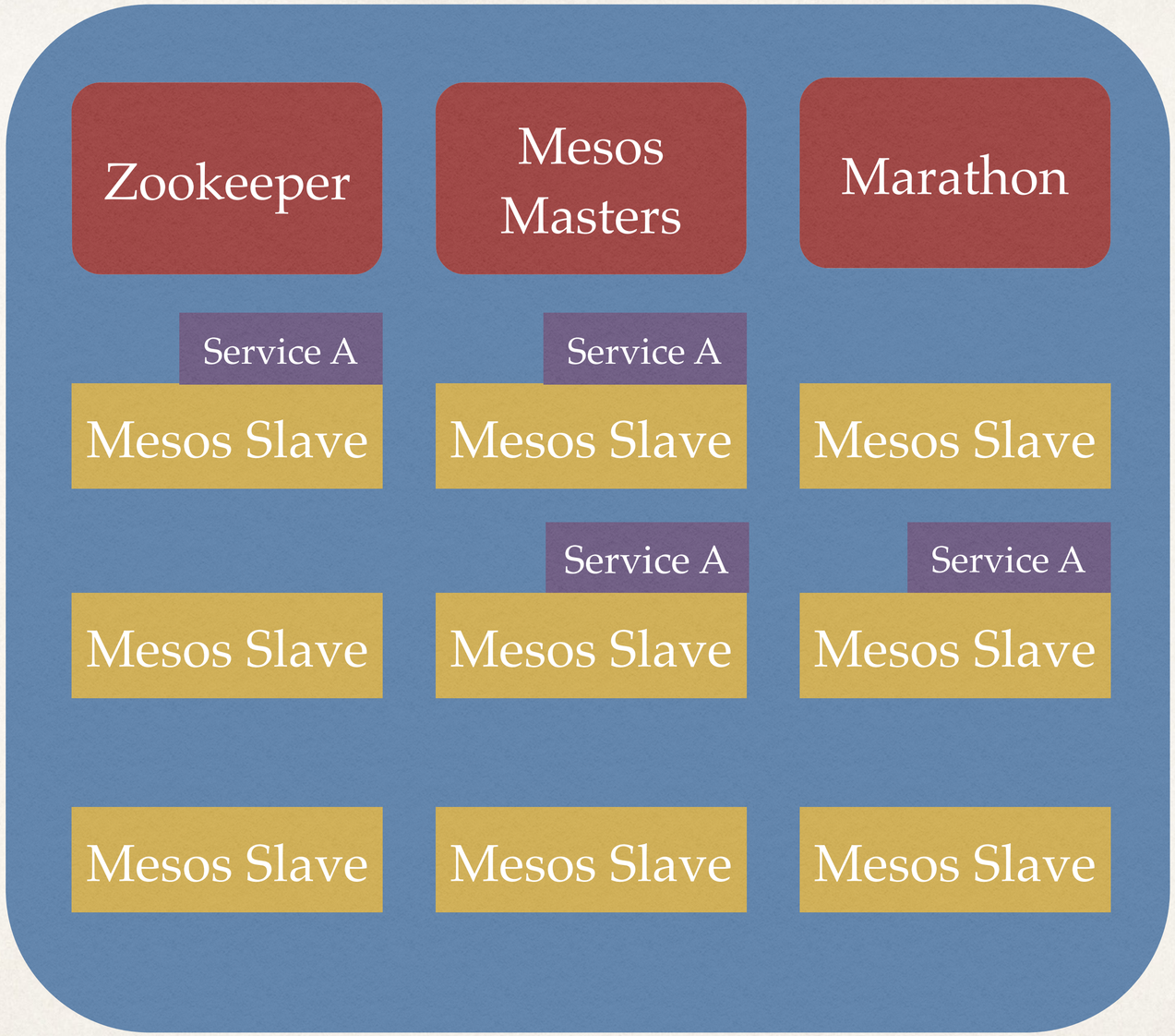
- We build micro-services
- Containerize them using Docker
- Deploy them using Mesos/Marathon
- Push the data through a Kafka pipeline
- And use Spark to work with the data

The foundation for the data infrastructure is Mesos and Marathon

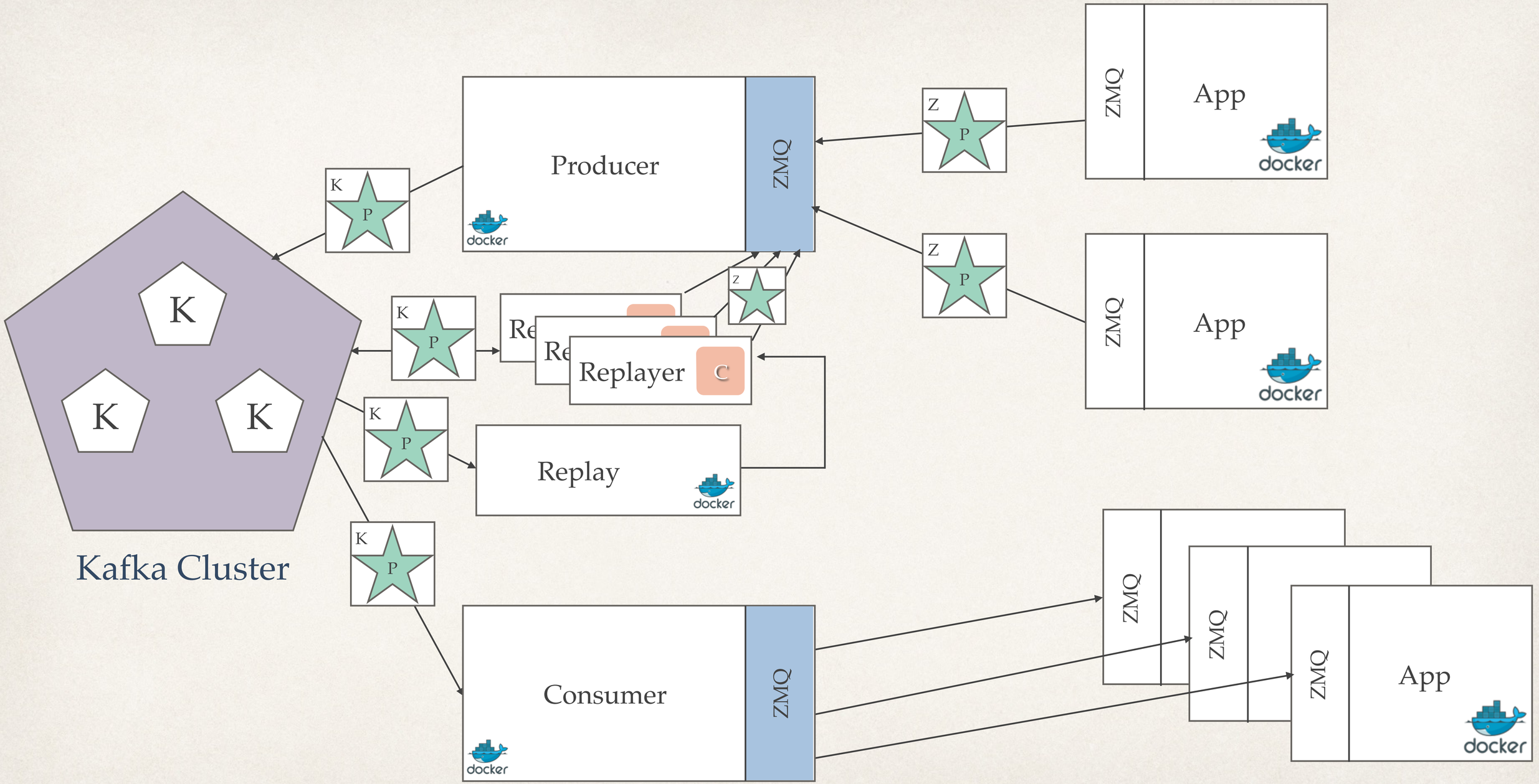

Zookeeper
Mesos Masters
Marathon
Service A
Service A
Mesos Slave
Mesos Slave
Mesos Slave
Service A
Mesos Slave
Mesos Slave
Mesos Slave
Mesos Slave
Mesos Slave
Mesos Slave

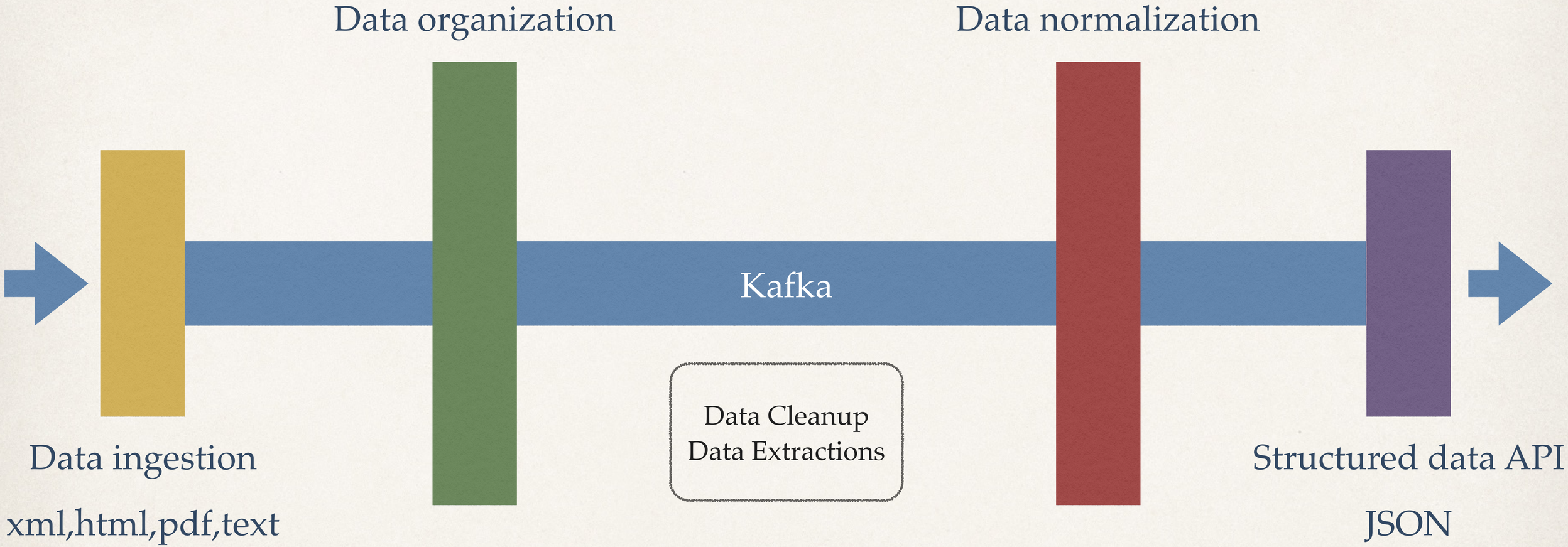

Zookeeper
Mesos Masters
Marathon
Service A
Service A
Mesos Slave
Mesos Slave
Mesos Slave
Service A
Service A
Mesos Slave
Mesos Slave
Mesos Slave
Mesos Slave
Mesos Slave
Mesos Slave

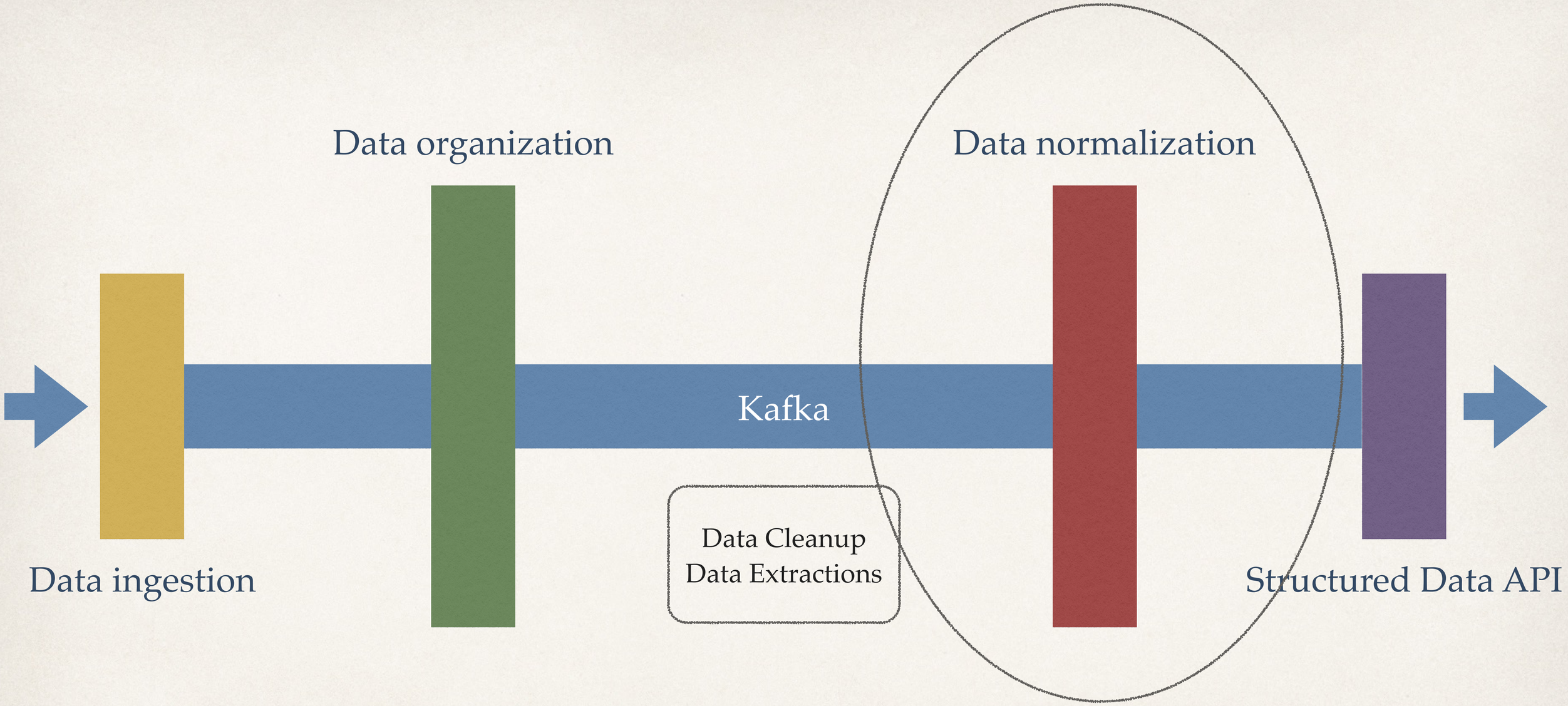

Zookeeper
Mesos Masters
Marathon
Service A
Service A
Mesos Slave
Mesos Slave
Mesos Slave
Service A
Service A
Mesos Slave
Mesos Slave
Mesos Slave
Mesos Slave
Mesos Slave
Mesos Slave


Zookeeper
Mesos Masters
Marathon
Service A
Service A
Mesos Slave
Mesos Slave
Mesos Slave
Service A
Service A
Mesos Slave
Mesos Slave
Mesos Slave
Mesos Slave
Mesos Slave
Mesos Slave

The foundation for the data pipeline is Kafka


Producer
ZMQ
K
P
Z
P
App
docker
Kafka Cluster
K
K
K
Replayer
C
Replay
Consumer


Data organization
Data normalization
Kafka
Data Cleanup
Data Extractions
Data ingestion
xml,html,pdf,text
Structured data API
JSON


Data organization
Data normalization
Kafka
Data Cleanup
Data Extractions
Data ingestion
Structured Data API

Data normalization tasks

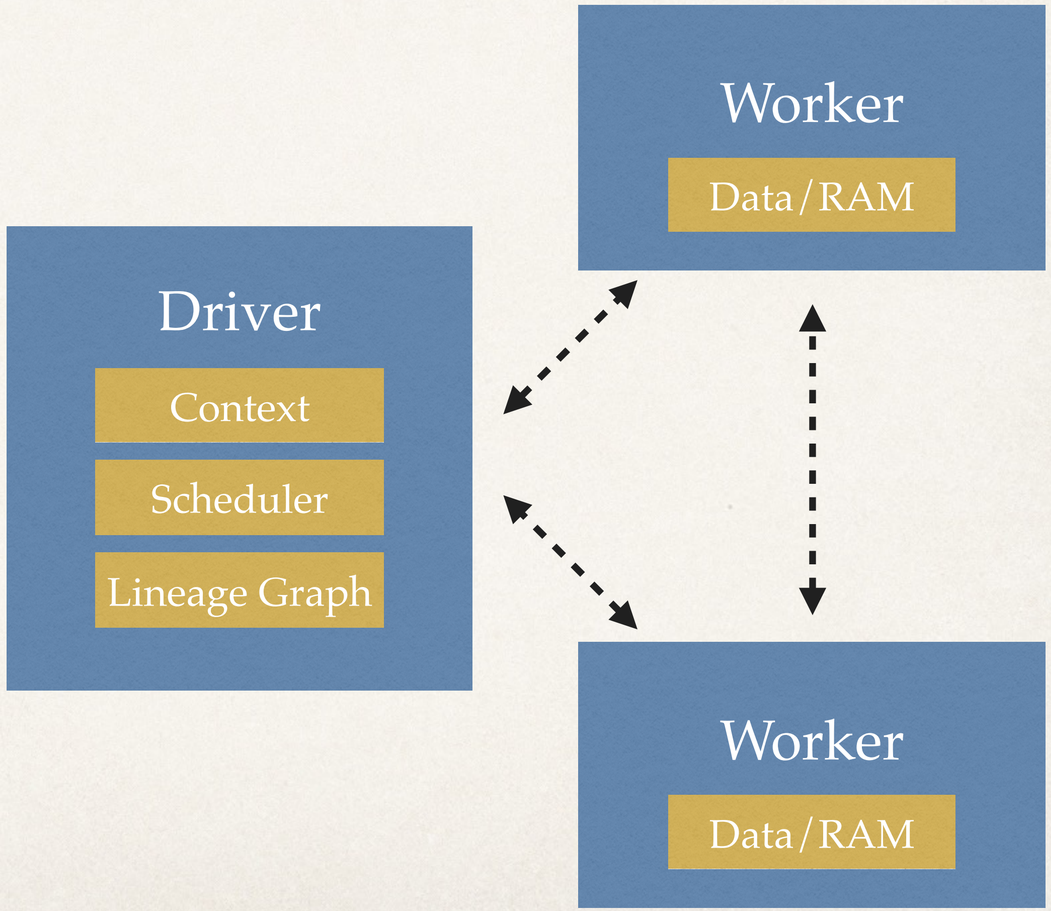
- Using classifiers to normalize test results (i.e. blood tests)
  - Predicting the class - identifying correct name
  - Standardizing on units
  - Normalizing values
- We accomplish this by using Apache Spark

Why Spark?

- Complete toolset to work with large distributed data sets
- Great stack for doing stream processing
- Built in tools for Machine Learning (MLlib)
- Easy deployment to Mesos or Standalone
- It is FAST (Lazy loading of data into RDDs)
- Native Scala/Python interaction and includes a shell (developer friendly)

Spark Basics

- Spark Driver
  - Define and launch RDDs and has the reference to the SparkContext
- Spark Workers
  - Read/Write data to/from RDDs/HDFS
  - Transforms RDD partitions (filter(), map(), union() etc.)
- Actions
  - collect(), broadcast(), count(), reduce(), take(n)



Spark MLlib

- It’s a framework not a library
- Contains all components you need to do ML at scale.
- Leveraging RDDs to scale across multiple workers
- Includes common algorithms for that are optimized for parallelized environments.

Classification and Regression

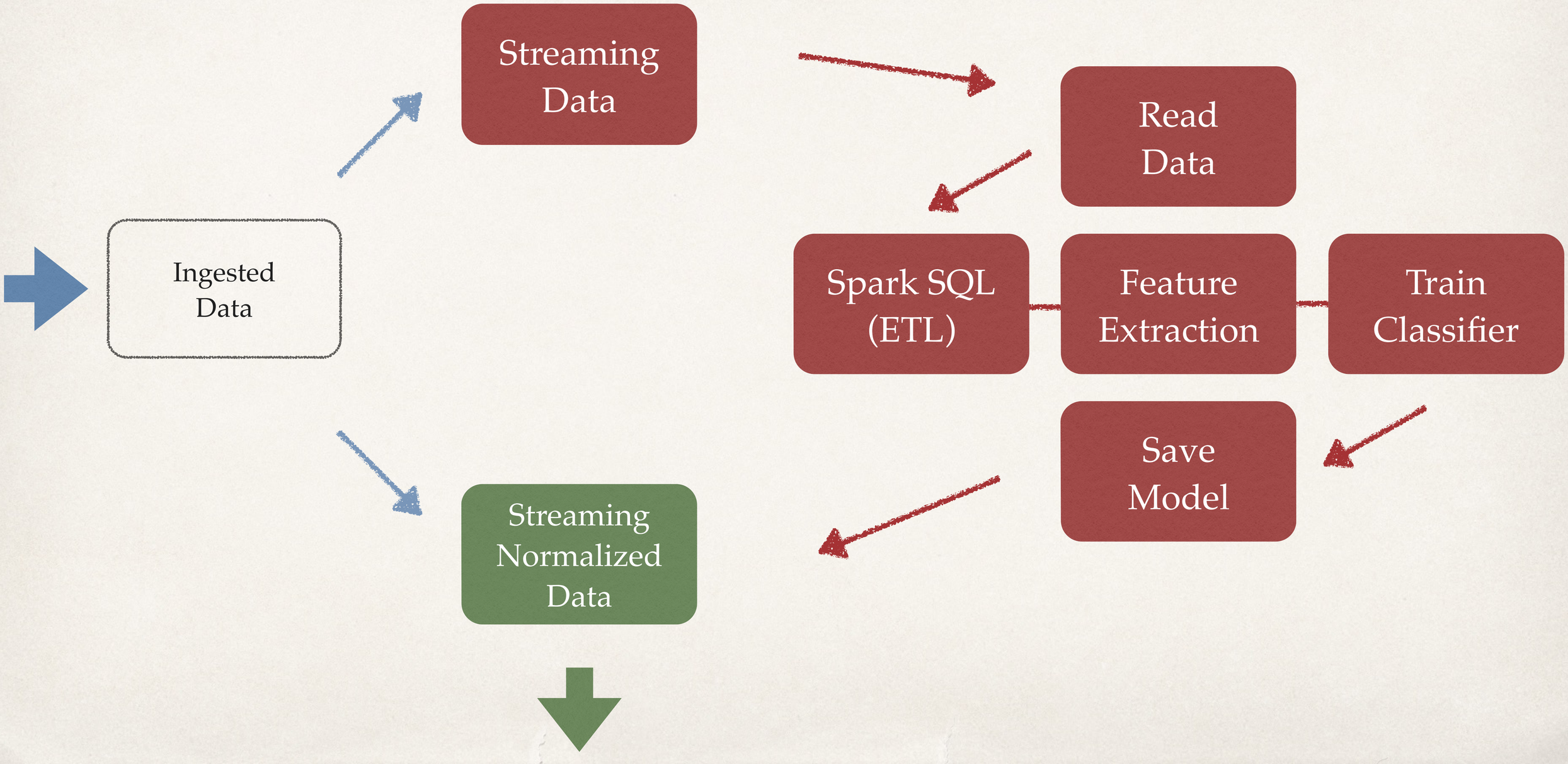
- Libraries for
  - binary classification,
  - multi-class classification
  - and regression analysis.
- We'll look at an example using LogisticRegressionWithLBFGS to classify clinical test results

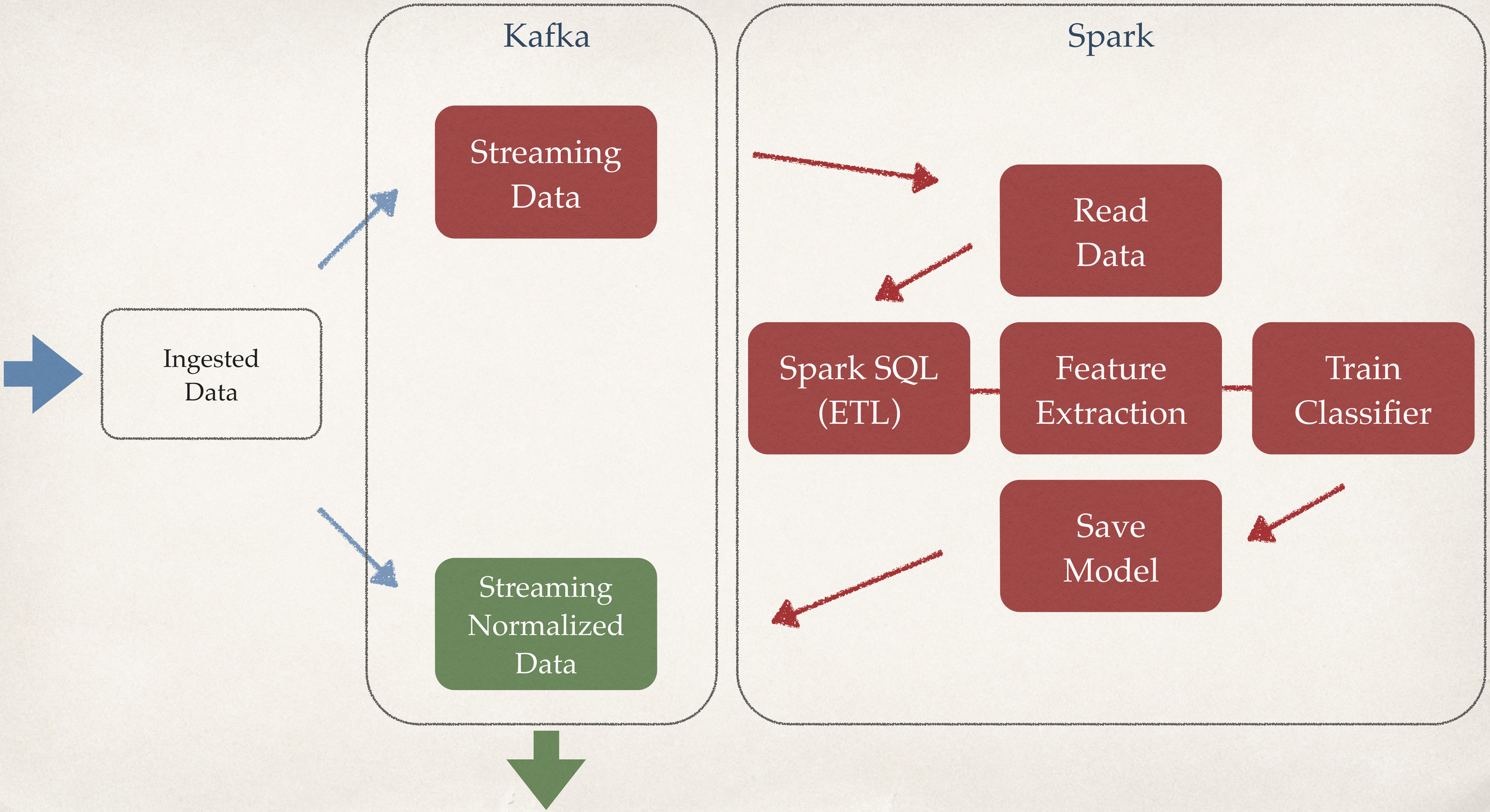
Code example (MLlib /Classification)

Spark Streaming

- Once you have trained a model you can plug it into a streaming Context and get real time predictions / decisions.
- Spark Streaming extends the core API
- Ingest data from 0MQ, Kafka, TCP etc.
- Publish data to dashboards or store in the database etc.

```scala
import org.apache.spark.streaming._
import org.apache.spark.streaming.kafka._

// Initializing Streaming Context
val conf = new SparkConf().setAppName('TestResultClassifier')
val ssc = new StreamingContext(conf, Seconds(5))

// Initializing test restult stream
val kafkaStream = KafkaUtils.createStream(ssc, ...config...)
val testResults = kafkaStream.map(_.getName)

// Load existing model
val model = LogisticRegressionModel.load(sc, "models/test-results/demo")

// Predict using model
def predict(label: String) = {
  val tfVectors = tf.transform(label.toLowerCase().split(" "))
  val predictedVal = model.predict(tfVectors)
}

// Start the streaming
ssc.start()
ssc.awaitTermination()
```


Streaming Data
Read Data
Ingested Data
Spark SQL (ETL)
Feature Extraction
Train Classifier
Save Model
Streaming Normalized Data


Kafka
Spark
Streaming Data
Ingested Data
Read Data
Spark SQL (ETL)
Feature Extraction
Train Classifier
Save Model
Streaming Normalized Data

Whats next…

- This architecture and the tools have given us a lot of flexibility
- Stable platform for the future scale
- Still early stages building out this platform
- Data engineers have ability to explore

Thank you!

Contact info:

ola@humanapi.co

@OlaWiberg

http://humanapi.co