# Apache Spark™ MLlib 2.x: How to Productionize your Machine Learning Models

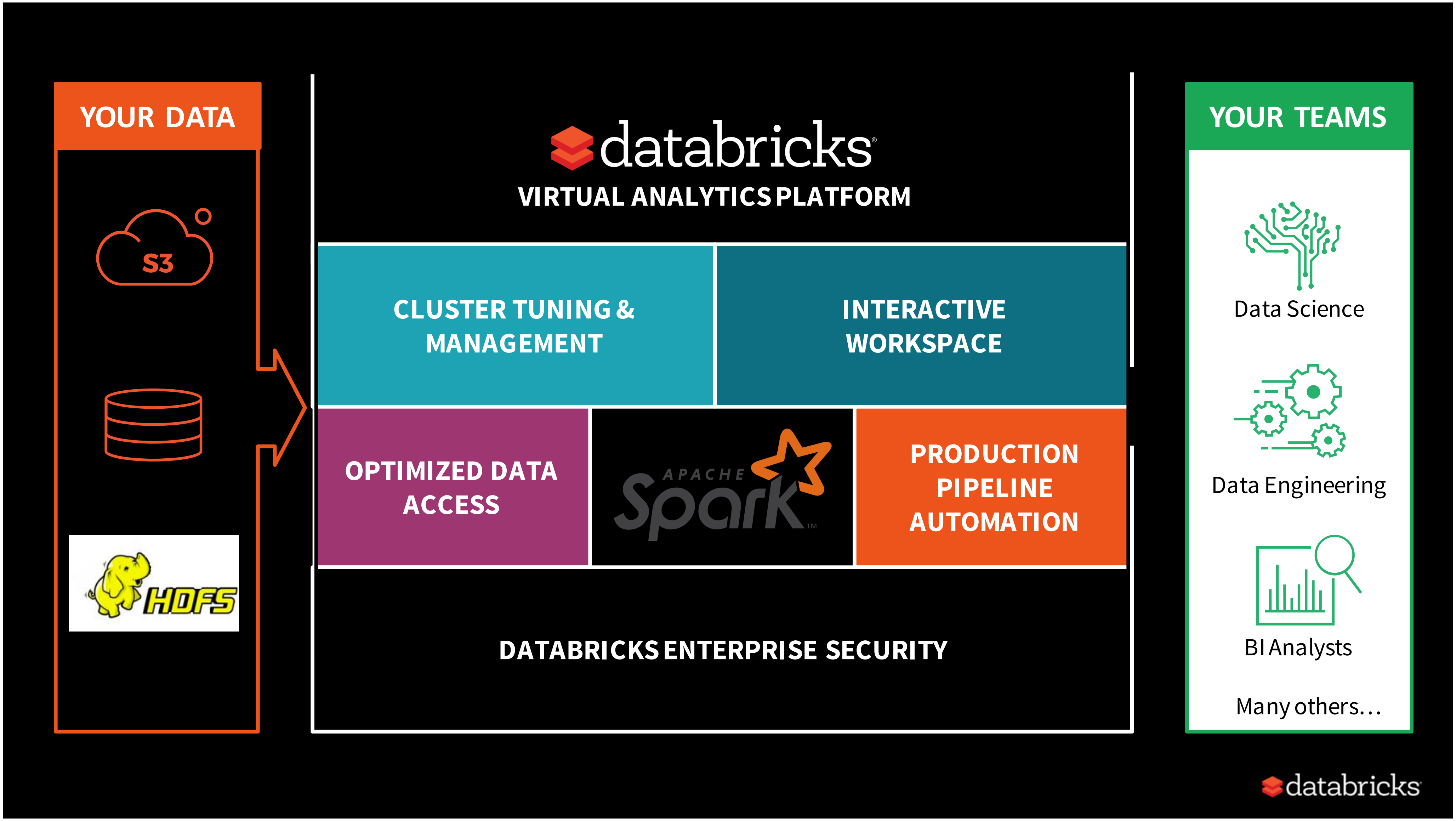
Richard Garris (Principal Solutions Architect)

databricks

**VISION** Empower anyone to innovate faster with big data.

**PRODUCT** A fully-managed data processing platform for the enterprise, powered by Apache Spark.

**WHO WE ARE** Founded by the creators of Apache Spark. Contributes **75%** of the open source code, **10x** more than any other company.



YOUR DATA
S3
HDFS
databricks
VIRTUAL ANALYTICS PLATFORM
CLUSTER TUNING & MANAGEMENT
INTERACTIVE WORKSPACE
OPTIMIZED DATA ACCESS
APACHE Spark
PRODUCTION PIPELINE AUTOMATION
DATABRICKS ENTERPRISE SECURITY
YOUR TEAMS
Data Science
Data Engineering
BI Analysts
Many others…
databricks

# About Me

- Richard L Garris
  - rlgarris@databricks.com
  - Twitter @rlgarris
- Principal Data Solutions Architect @ Databricks
- 12+ years designing Enterprise Data Solutions for everyone from startups to Global 2000
- Prior Work Experience PwC, Google and Skytree – the Machine Learning Company
- Ohio State Buckeye and Masters from CMU

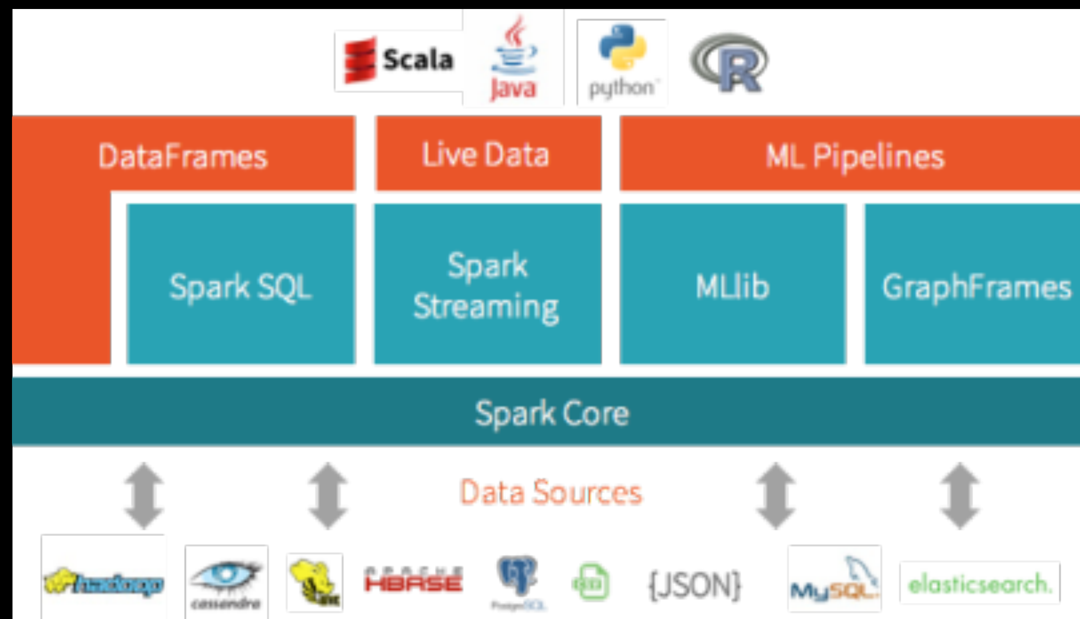
databricks

# Outline

- Spark Mllib 2.X
- Model Serialization
- Model Scoring System Requirements
- Model Scoring Architectures
- Databricks Model Scoring

# About Apache Spark™ MLlib

- Started with Spark 0.8 in the AMPLab in 2014
- Migration to Spark DataFrames started with Spark 1.3 with feature parity within 2.X
- Contributions by 75+ orgs, ~250 individuals
- Distributed algorithms that scale linearly with the data



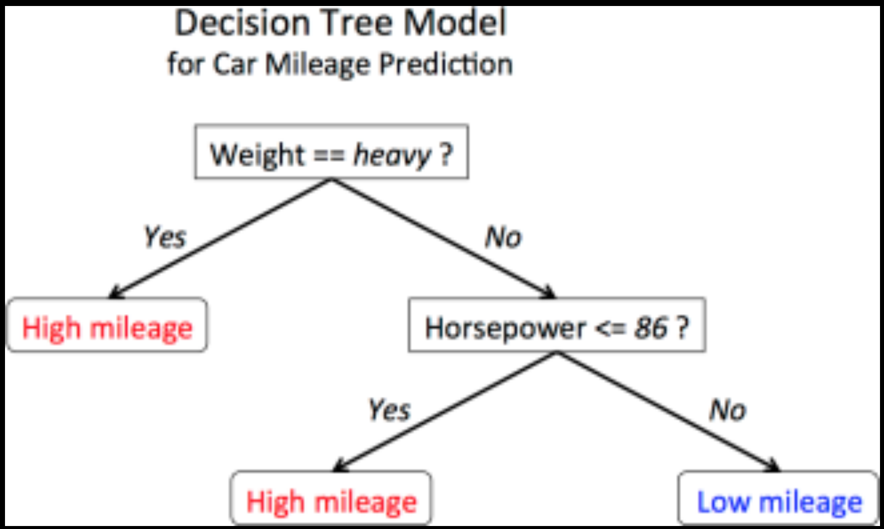

databricks

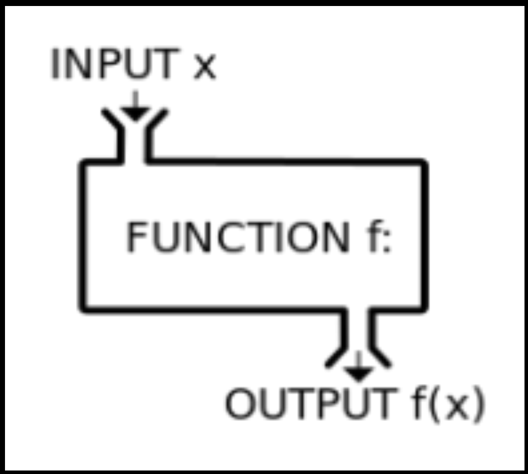
# MLlib's Goals

- General purpose machine learning library optimized for big data
  - Linearly scalable = 2x more machines , runtime theoretically cut in half
  - Fault tolerant = resilient to the failure of nodes
  - Covers the most common algorithms with distributed implementations
- Built around the concept of a Data Science Pipeline (scikit-learn)
- Written entirely using Apache Spark™
- Integrates well with the Agile Modeling Process

# A Model is a Mathematical Function

- A model is a function: $f(x)$
- Linear regression $y = b_0 + b_1 x_1 + b_2 x_2$



# ML Pipelines

Load data

↓

Extract features

↓

Train model

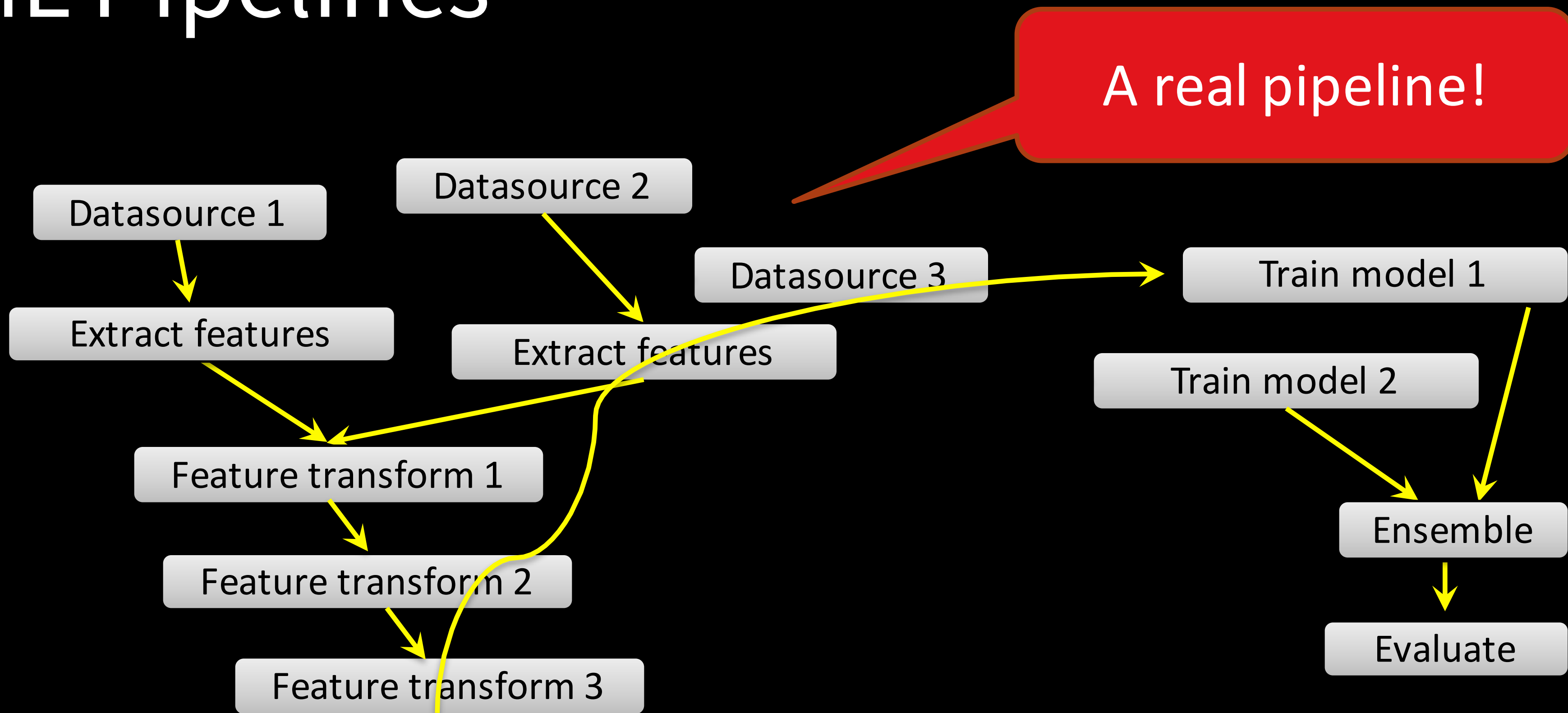
↓

Evaluate

A very simple pipeline

databricks

# ML Pipelines




databricks

# Productionizing Models Today

## Data Science

Develop Prototype Model using Python/R

## Data Engineering

Re-implement model for production (Java)

databricks

# Problems with Productionizing Models

## Data Science

Develop Prototype
Model using Python/R

## Data Engineering

Re-implement model for
production (Java)

- Extra work
- Different code paths
- Data science does not translate to production
- Slow to update models

# MLlib 2.X Model Serialization

## Data Science

Develop Prototype
Model using Python/R

Persist model or Pipeline:

```
model.save("s3n://...")
```

## Data Engineering

Load Pipeline (Scala/Java)

```
Model.load("s3n://…")
```

Deploy in production

databricks

# MLlib 2.X Model Serialization Snippet

## Scala

```scala
val lrModel = lrPipeline.fit(dataset)

// Save the Model
lrModel.write.save("/models/lr")
```

## Python

```python
lrModel = lrPipeline.fit(dataset)

# Save the Model
lrModel.write.save("/models/lr")
```

# Model Serialization Output

## Code

```
// List Contents of the Model Dir
dbutils.fs.ls("/models/lr")
```

## Output

```
+-------------------------+----------+
|path                     |name      |
+-------------------------+----------+
|dbfs:/models/lr/metadata/|metadata/|
|dbfs:/models/lr/stages/  |stages/   |
+-------------------------+----------+
```

Remember this is a pipeline model and these are the stages!

databricks

# Transformer Stage (StringIndexer)

## Code

```
// Cat the contents of the Metadata dir
dbutils.fs.head("/models/lr/stages/00_strIdx_bb9728f85745/metadata/part-00000")

// Display the Parquet File in the Data dir
display(spark.read.parquet("/models/lr/stages/00_strIdx_bb9728f85745/data/"))
```

## Output

Metadata and params


```
{
  "class":"org.apache.spark.ml.feature.StringIndexerModel",
  "timestamp":1488120411719,
  "sparkVersion":"2.1.0",
  "uid":"strIdx_bb9728f85745",
  "paramMap":{
    "outputCol":"workclassIdx",
    "inputCol":"workclass",
    "handleInvalid":"error"
  }
}
```


Data (Hashmap)

| labels |
| --- |
| array |
| 0: Private |
| 1: Self-emp-not-inc |
| 2: Local-gov |
| 3: ? |
| 4: State-gov |
| 5: Self-emp-inc |
| 6: Federal-gov |
| 7: Without-pay |
| 8: Never-worked |

databricks

# Estimator Stage (LogisticRegression)

## Code

```
// Cat the contents of the Metadata dir
dbutils.fs.head(”/models/lr/stages/18_logreg_325fa760f925/metadata/part-00000")

// Display the Parquet File in the Data dir
display(spark.read.parquet("/models/lr/stages/18_logreg_325fa760f925/data/"))
```

## Output

Model params


{"class":"org.apache.spark.ml.classification.LogisticRegressionModel",
"timestamp":1488120446324,
"sparkVersion":"2.1.0",
"uid":"logreg_325fa760f925",
"paramMap":{
"predictionCol":"prediction",
"standardization":true,
"probabilityCol":"probability",
"maxIter":100,
"elasticNetParam":0.0,
"family":"auto",
"regParam":0.0,
"threshold":0.5,
"fitIntercept":true,
"labelCol":"label” }}


| numClasses | numFeatures | interceptVector | coefficientMatrix |
|---|---|---|---|
| 2 | 100 | array<br>0: 1<br>1: 1<br>2: []<br>3:<br>0: -2.0412260644120477 | object<br>numRows: 1<br>numCols: 100<br>values:<br>0: -1.6536519269918135<br>1: -2.1437754918551044<br>2: -1.8372425650595294 |

Intercept + Coefficients

databricks

# Estimator Stage (DecisionTree)

## Code

```
// Display the Parquet File in the Data dir
display(spark.read.parquet(”/models/dt/stages/18_dtc_3d614bcb3ff825/data/"))

// Re-save as JSON
spark.read.parquet("/models/dt/stages/18_dtc_3d614bcb3ff825/data/").json((”/models/json/dt").
```

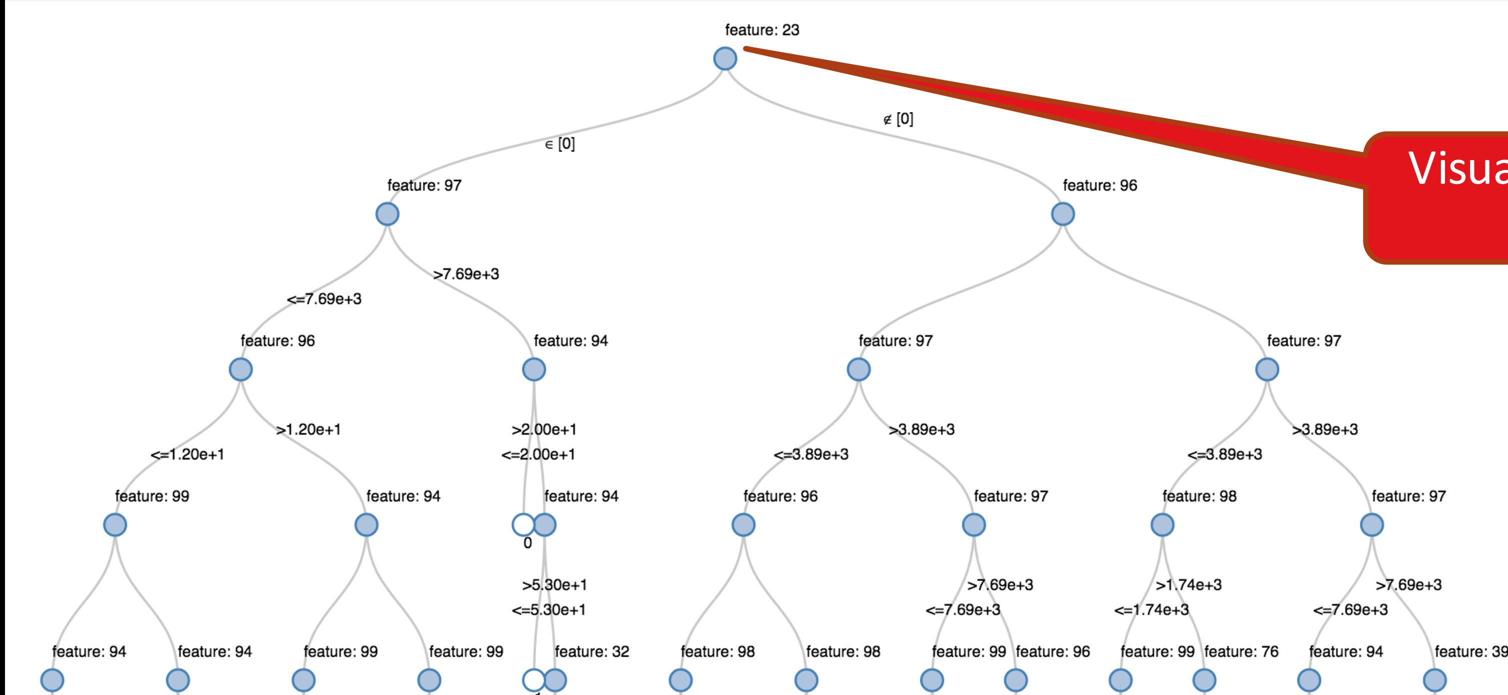
## Output

| id | prediction | impurity | impurityStats | gain | leftChild | rightChild | split |
|---|---|---|---|---|---|---|---|
| 99 | 1 | 0 | [0,2] | -1 | -1 | -1 | {"featureIndex":-1,"leftCategoriesOrThreshold":[],"numCategories":-1} |
| 100 | 1 | 0.005154604757994008 | [1,386] | 0.00013864488567297793 | 101 | 102 | {"featureIndex":94,"leftCategoriesOrThreshold":[66],"numCategories":-1} |
| 101 | 1 | 0 | [0,353] | -1 | -1 | -1 | {"featureIndex":-1,"leftCategoriesOrThreshold":[],"numCategories":-1} |

Decision Tree Splits

# Visualize Stage (DecisionTree)

```
display(dtModel.stages.last.asInstanceOf[DecisionTreeClassificationModel])
```




Visualization of the Tree
In Databricks

databricks

# What are the Requirements for a Robust Model Deployment System?

# Model Scoring Environment Examples

- In Web Applications / Ecommerce Portals
- Mainframe / Batch Processing Systems
- Real-Time Processing Systems / Middleware
- Via API / Microservice
- Embedded in Devices (Mobile Phones, Medical Devices, Autos)

# Hidden Technical Debt in ML Systems

**"Hidden Technical Debt in Machine Learning Systems ", Google NIPS 2015**

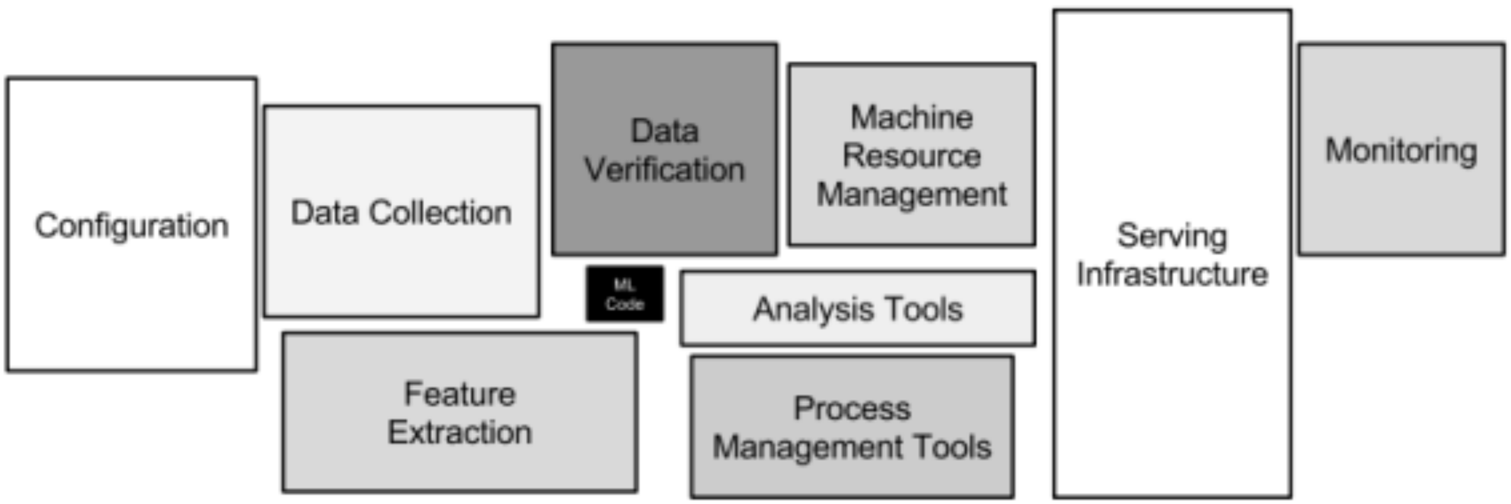
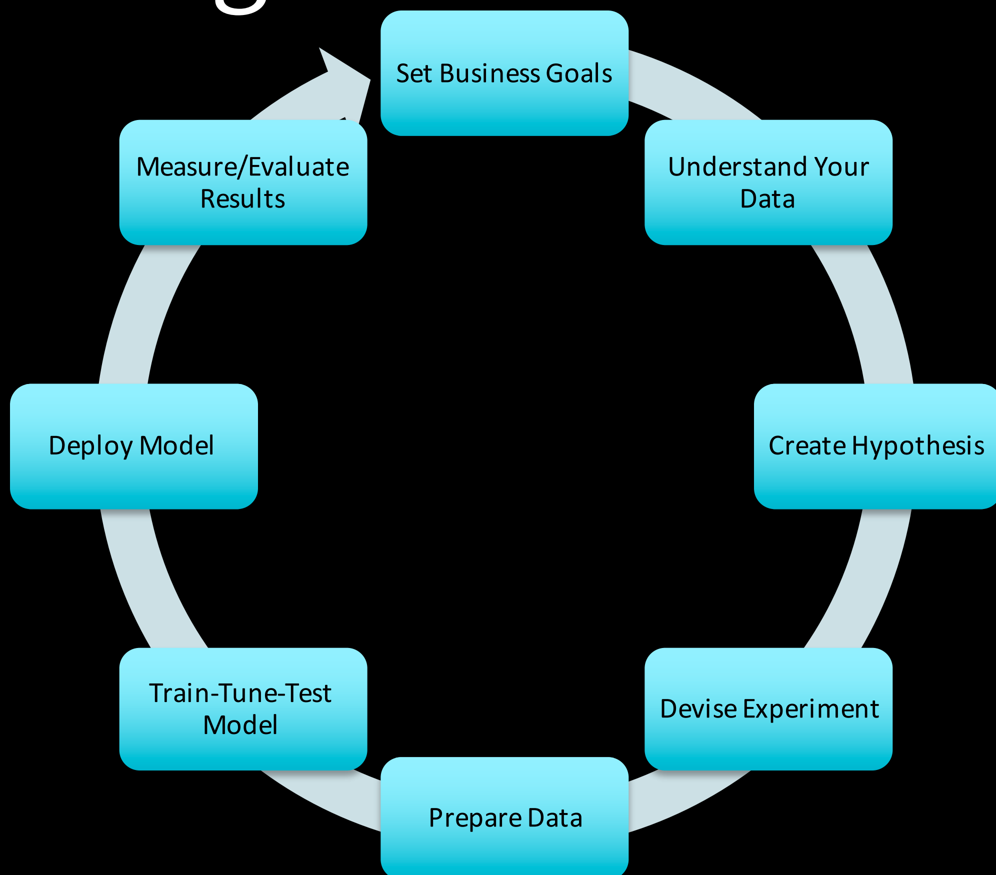


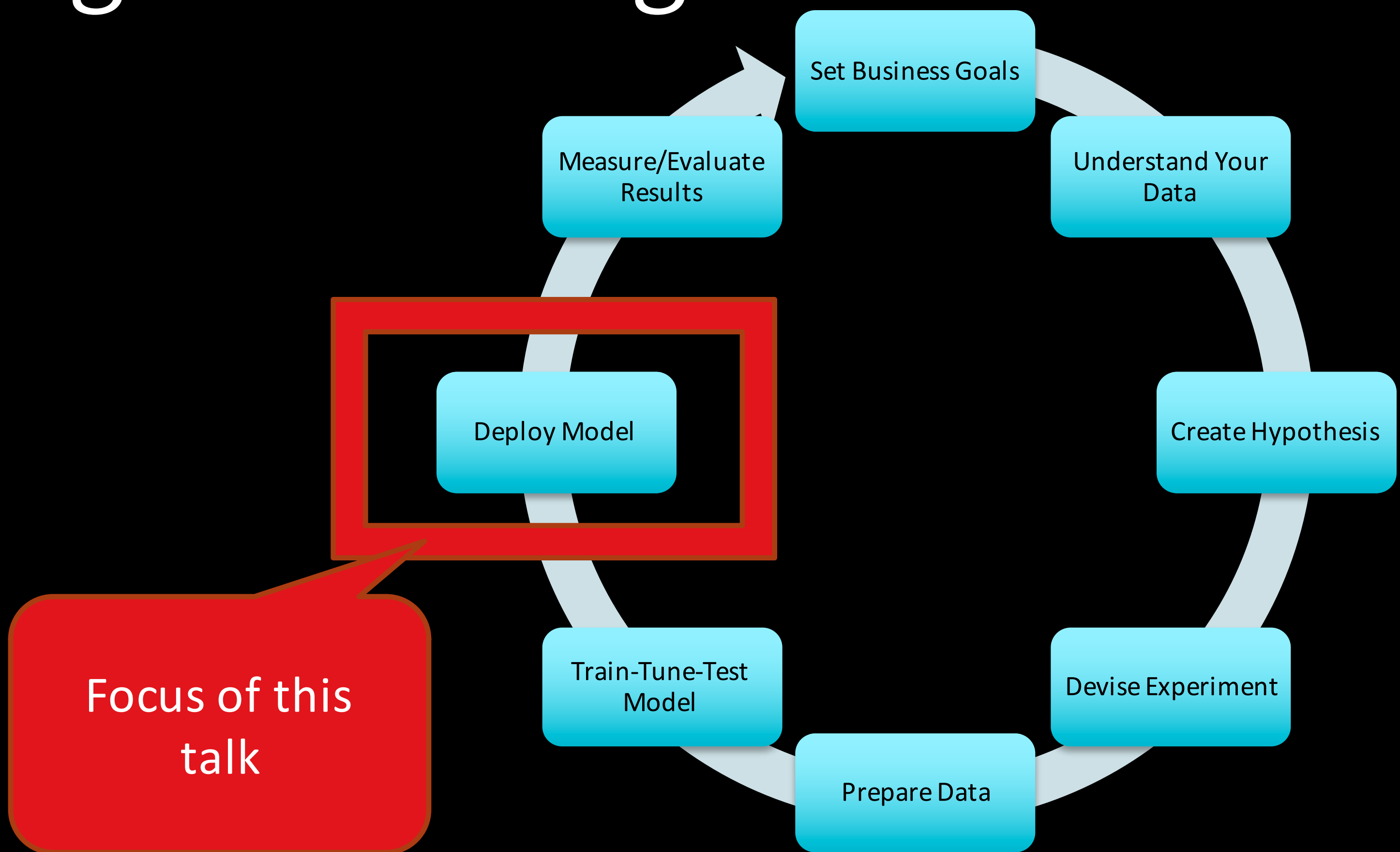

Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

# Agile Modeling Process



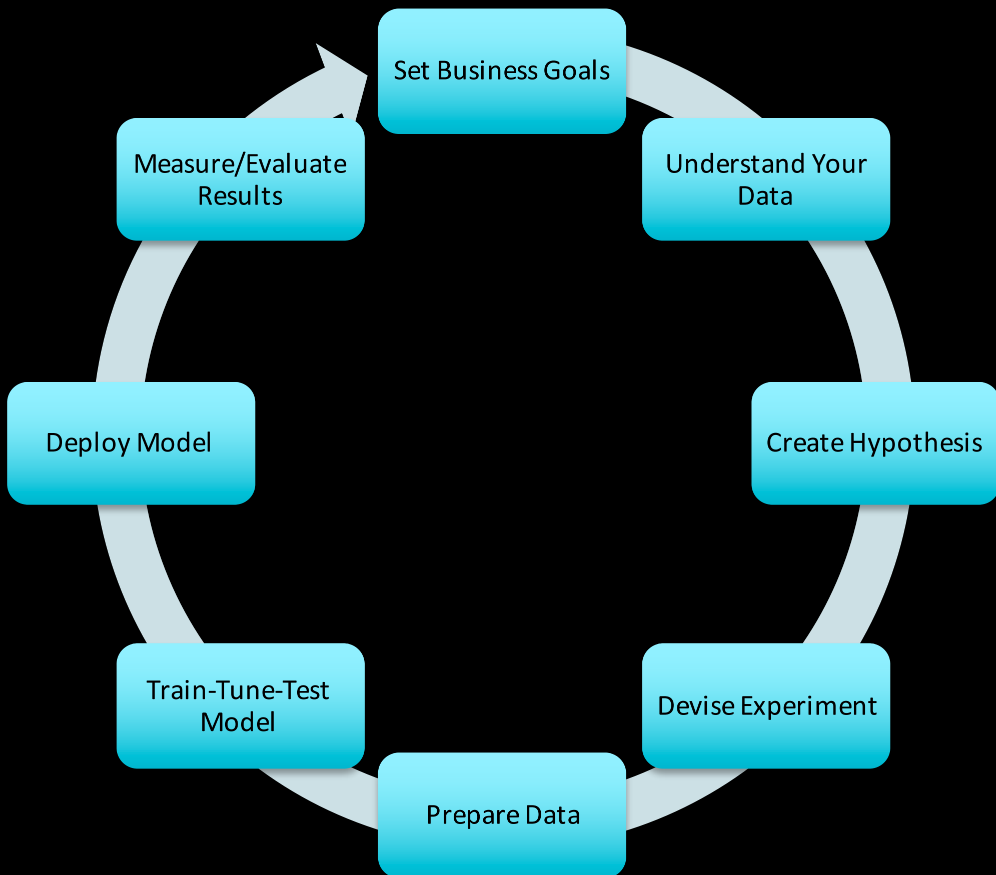

databricks

# Agile Modeling Process




databricks

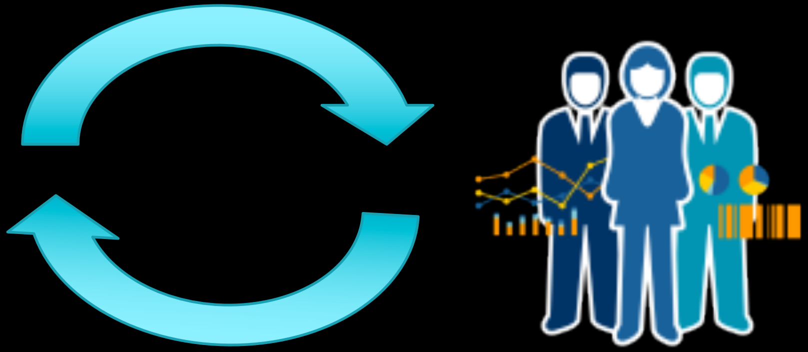
# Deployment Should be Agile




- Deployment needs to support A/B testing and experiments
- Deployment should support measuring and evaluating model performance
- Deployment should be fast and adaptive to business needs

# Model A/B Testing, Monitoring, Updates

- A/B testing – comparing two versions to see what performs better
- Monitoring is the process of observing the model's performance, logging it's behavior and alerting when the model degrades
- Logging should log exactly the data feed into the model at the time of scoring
- Model update process
  - Benchmark (or Shadow Models)
  - Phase-In (20% traffic)
  - Avoid Big Bang



databricks

# Consider the Scoring Environment

**Customer SLAs**

- Response time
- Throughput (predictions per second)
- Uptime / Reliability

**Tech Stack**

- C / C++
- Legacy (mainframe)
- Java

# Scoring in Batch vs Real-Time

## Batch

- Asynchronous
- Internal Use
- Triggers can be event based on time based
- Used for Email Campaigns, Notifications

## Real-Time

- Synchronous
- Could be Seconds:
  - Customer is waiting (human real-time)
- Subsecond:
  - High Frequency Trading
  - Fraud Detection on the Swipe

databricks

# Online Learning and Open / Closed Loop

## Open / Closed Loop

Open Loop – human being involved

Closed Loop – no human involved

- Model Scoring – almost always closed loop, some open loop e.g. alert agents or customer service
- Model Training – usually open loop with a data scientist in the loop to update the model

## Online Learning

- Online is closed loop, entirely machine driven but modeling is risky
- need to have proper model monitoring and safeguards to prevent abuse / sensitivity to noise
- MLlib supports online through streaming models (k-means, logistic regression support online)
- Alternative – use a more complex model to better fit new data rather than using online learning

databricks

# Model Scoring – Bot Detection

Not All Models Return Boolean – e.g. a Yes / No

<u>Example: Login Bot Detector</u>

Different behavior depending on probability that use is a bot

0.0-0.4 ☞ Allow login

0.4-0.6 ☞ Send Challenge Question

0.6 to 0.75 ☞ Send SMS Code

0.75 to 0.9 ☞ Refer to Agent

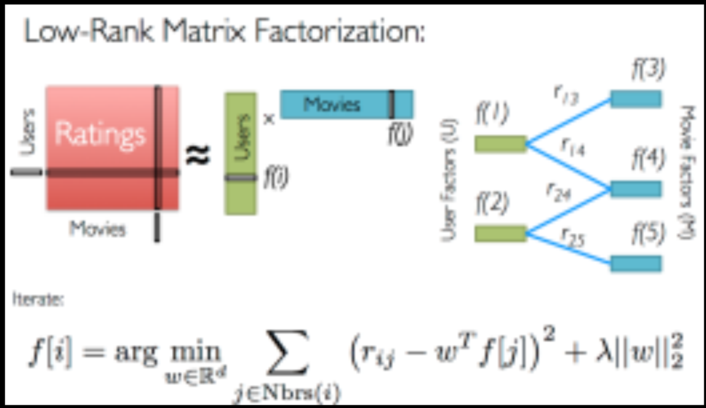
0.9 - 1.0 ☞ Block

# Model Scoring – Recommendations

<u>Output is a ranking of the top n items</u>

API – send user ID + number of items

Return sorted set of items to recommend




Optional –

pass context sensitive information to tailor results

# Model Scoring Architectures

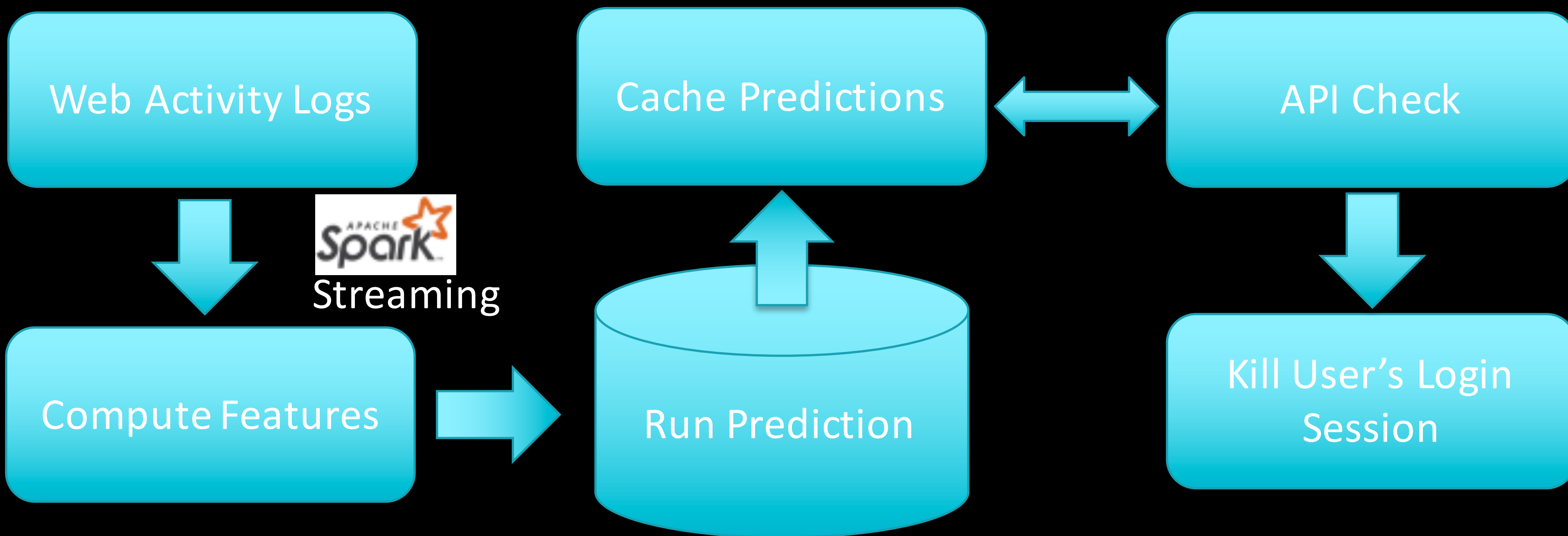
# Architecture Option A

## Precompute Predictions using Spark and Serve from Database


databricks

# Architecture Option B

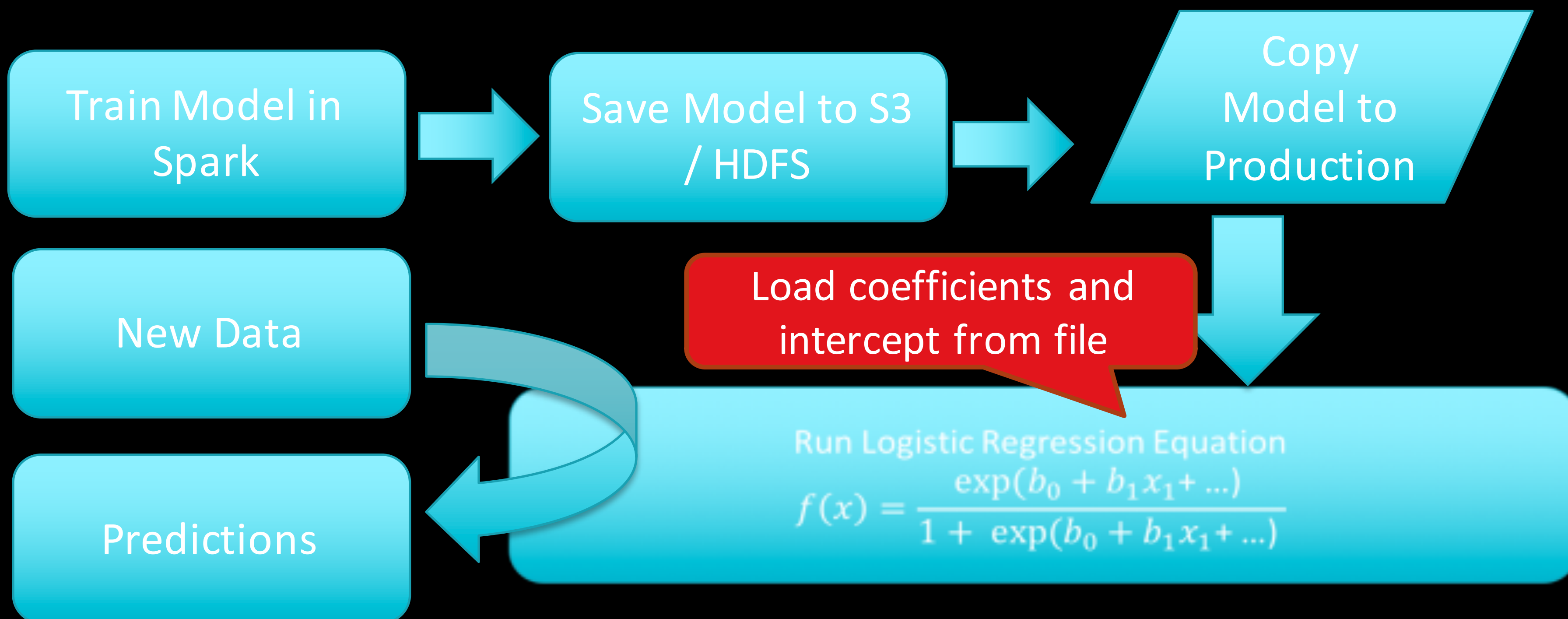
## Spark Stream and Score using an API with Cached Predictions




databricks

# Architecture Option C

## Train with Spark and Score Outside of Spark



# Databricks Model Scoring

databricks

# Databricks Model Scoring

- Based on Architecture Option C
- Goal: Deploy MLlib model outside of Apache Spark and Databricks.
  - Easy to Embed in Existing Environments
  - Low Latency and Complexity
  - Low Overhead

# Databricks Model Scoring

- Train Model in Databricks
  - Call Fit on Pipeline
  - Save Model as JSON
- Deploy model in external system
  - Add dependency on “dbml-local” package (without Spark)
  - Load model from JSON at startup
  - Make predictions in real time

## Code

```
// Fit and Export the Model in Databricks
val lrModel = lrPipeline.fit(dataset)
ModelExporter.export(lrModel, " /models/db ")


// In Your Application (Scala)
import com.databricks.ml.local.ModelImport

val lrModel= ModelImport.import("s3a:/...")
val jsonInput = ...
val jsonOutput= lrModel.transform(jsonInput)
```

# Databricks Model Scoring Private Beta

- Private Beta Available for Databricks Customers
- Available on Databricks using Apache Spark 2.1
- Only logistic regression available now
- Additional Estimators and Transformers in Progress

# Demo Model Scoring

https://community.cloud.databricks.com/?o=1526931011080774
#notebook/1904316851197504

databricks

# Thank You.

Questions?

Happy Sparking

richard@databricks.com