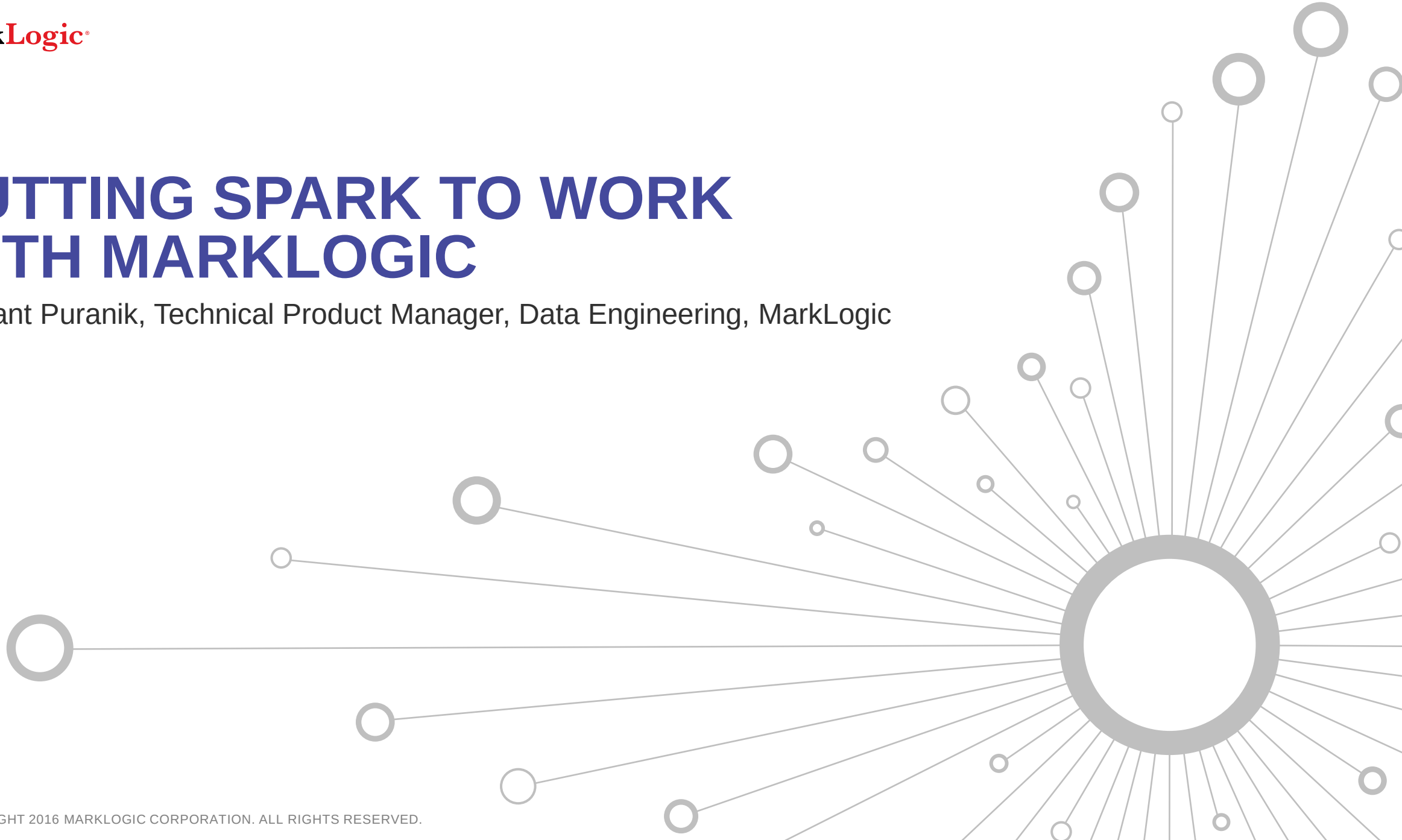
MarkLogic

PUTTING SPARK TO WORK WITH MARKLOGIC

Hemant Puranik, Technical Product Manager, Data Engineering, MarkLogic

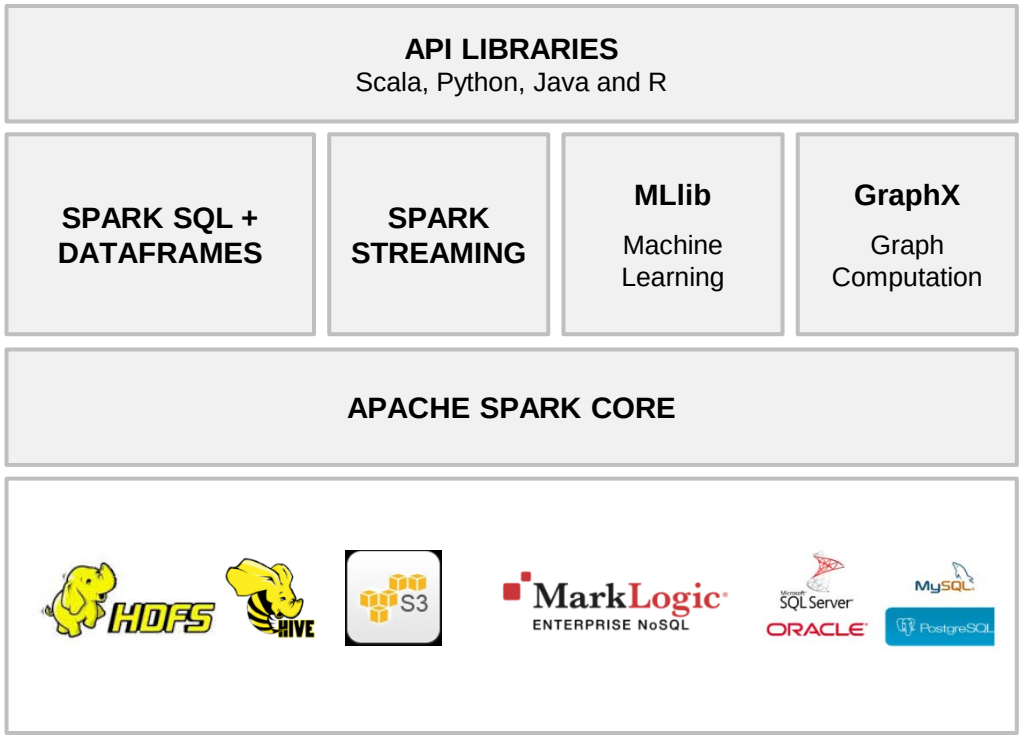


MarkLogic

Agenda

- How Apache Spark Complements MarkLogic
- Spark and MarkLogic Use Cases
- Deep Dive – Using Spark with MarkLogic
- MarkLogic and Spark Integration – What's Next
- Q&A

HOW APACHE SPARK COMPLEMENTS MARKLOGIC



WHAT IS APACHE SPARK?

"For Large-Scale Data Processing"…

- **Open Source Cluster Computing Framework**
  - Faster than Hadoop MapReduce
  - Built to deliver sophisticated analytics
  - Easy to use for manipulating large datasets
- **API abstractions over SQL and NoSQL data** – Analytics over Hadoop, RDBMS, MarkLogic etc.
- **Unified Engine for Advanced Analytics** – Streaming, Machine Learning, Graph etc.

MarkLogic vs Spark

MarkLogic ENTERPRISE NoSQL

- Response in Milliseconds
- 10,000 to 100,000 Concurrent Users
- Highly Selective Queries
- Read and Write Access
- Security and ACID Compliance

Spark

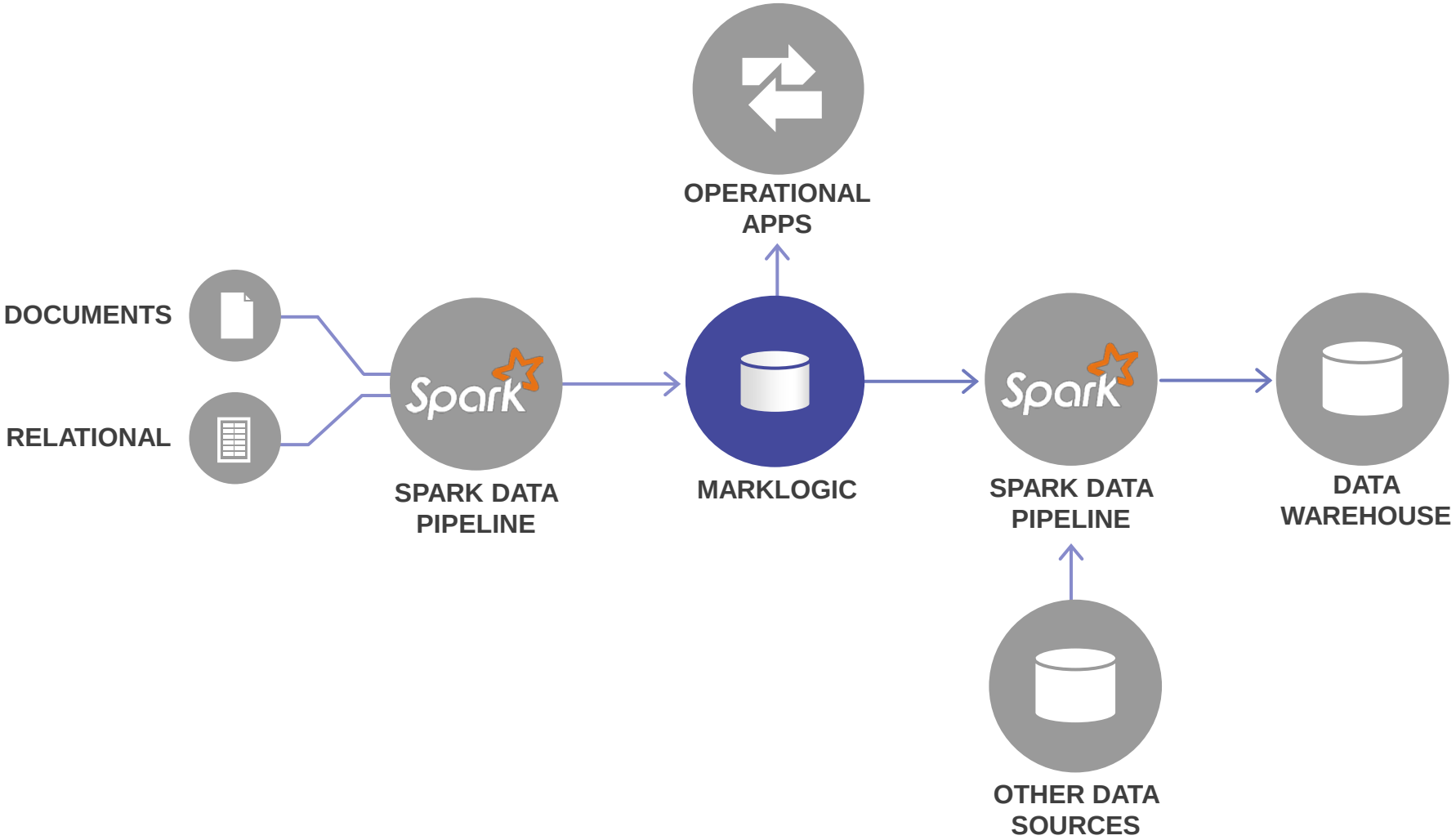
- Response in Seconds and Minutes
- 10s or 100s Concurrent Users
- Non Selective Queries
- Read Only Access
- Parallel Computations on Immutable Data

MarkLogic and Spark?

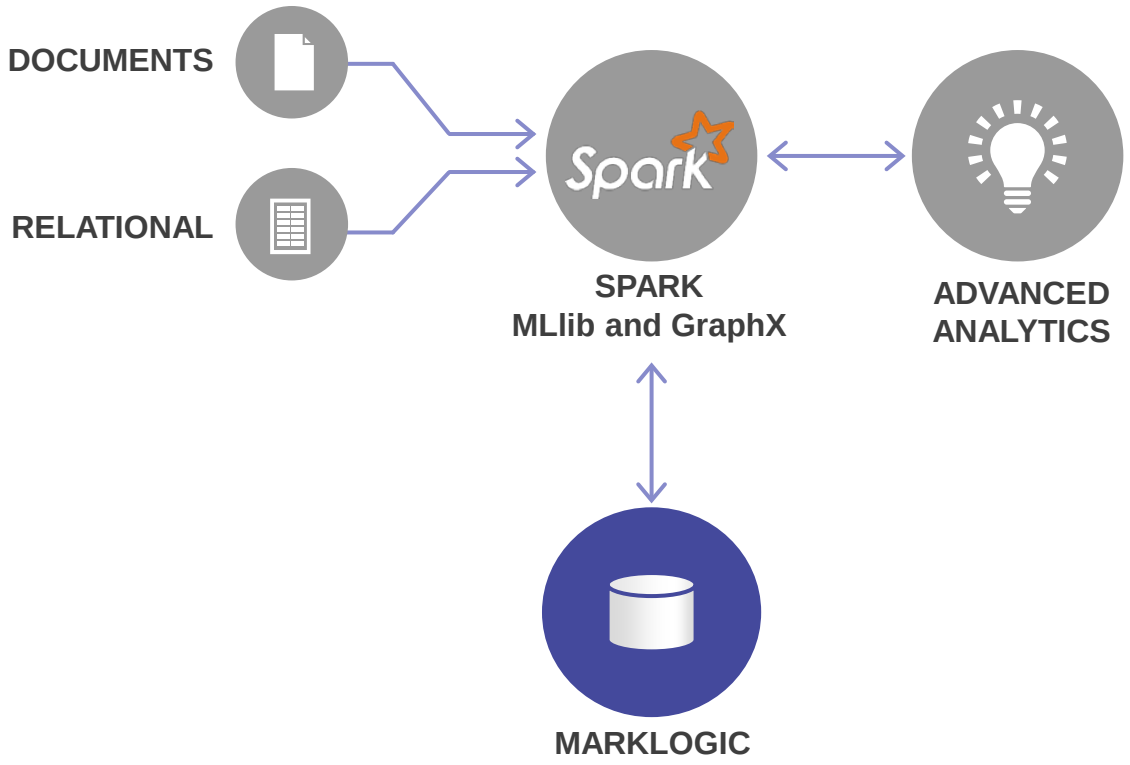
1. Aggregating data that comes in different shapes and source-specific formats — MarkLogic ENTERPRISE NoSQL
2. Highly concurrent transactions and secure query execution over changing data — MarkLogic ENTERPRISE NoSQL
3. Operational BI and Reporting in real time or near real time — MarkLogic ENTERPRISE NoSQL
4. Loading data from external sources into MarkLogic – transforming data on the fly — MarkLogic ENTERPRISE NoSQL + Spark
5. Treat MarkLogic datasets as immutable in order to perform multi-step analytics — MarkLogic ENTERPRISE NoSQL + Spark
6. Looping insights derived from analytical processes into operational applications — MarkLogic ENTERPRISE NoSQL + Spark

MARKLOGIC & SPARK USE CASES

Batch Data Movement – Spark Data Pipeline

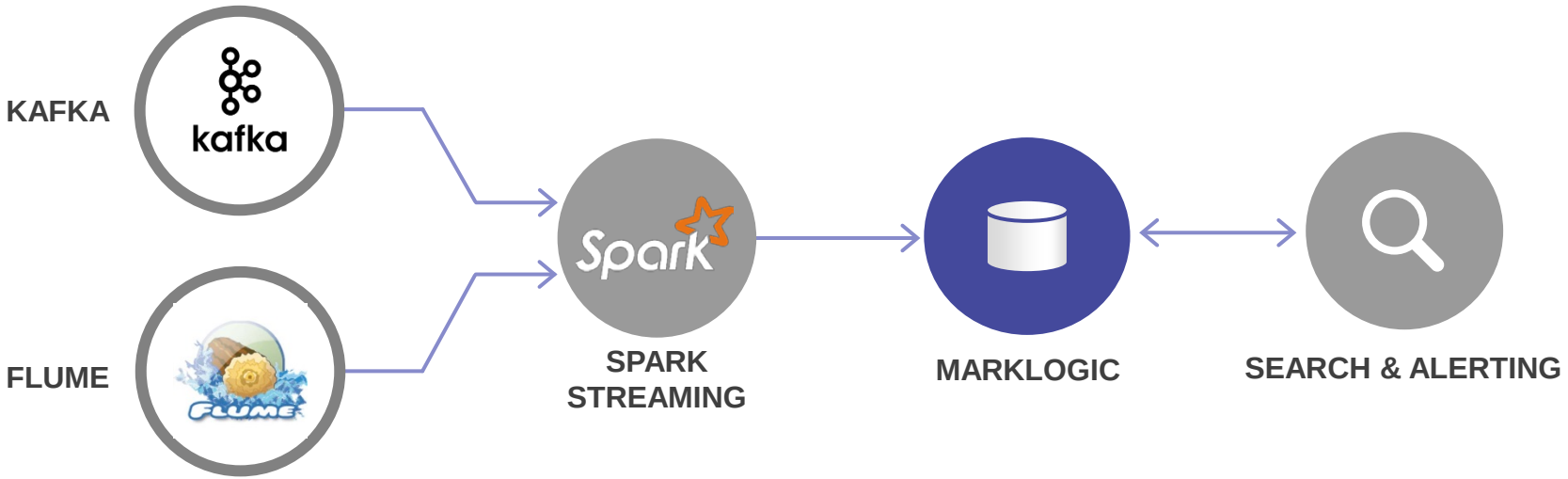


Advanced Analytics – Machine Learning, Graph Analytics



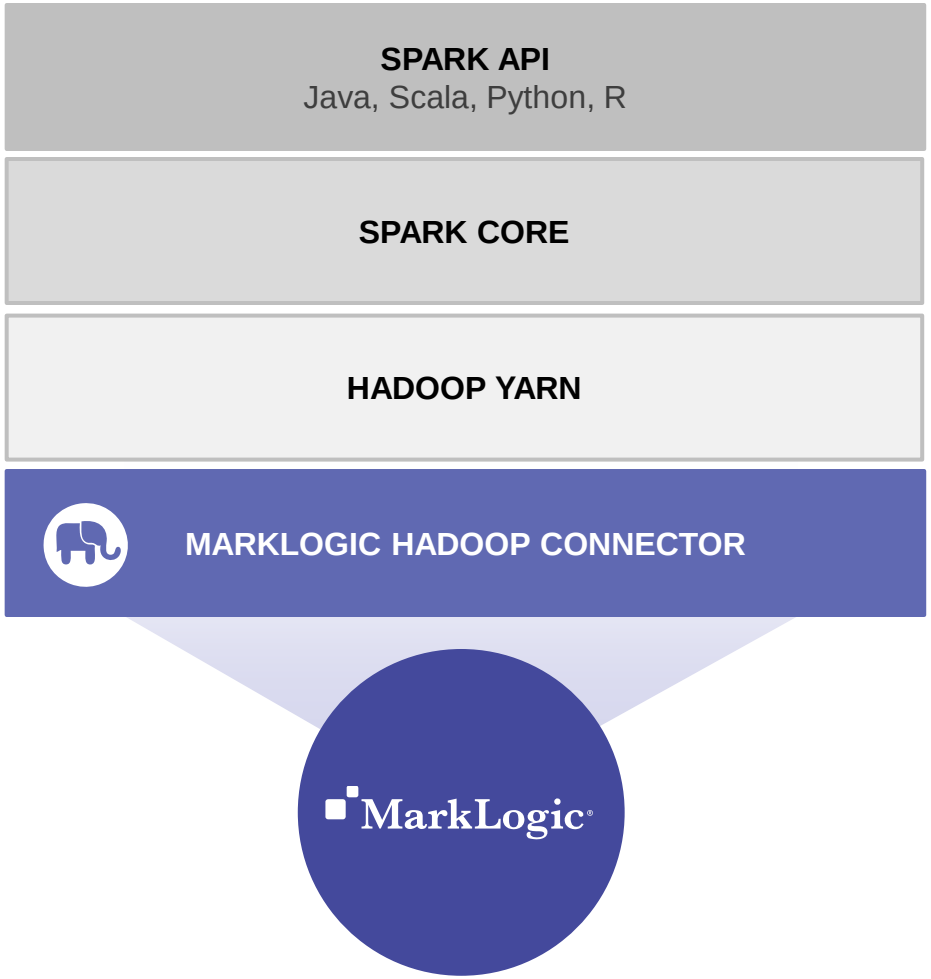
MarkLogic

Streaming Analytics



DEEP DIVE

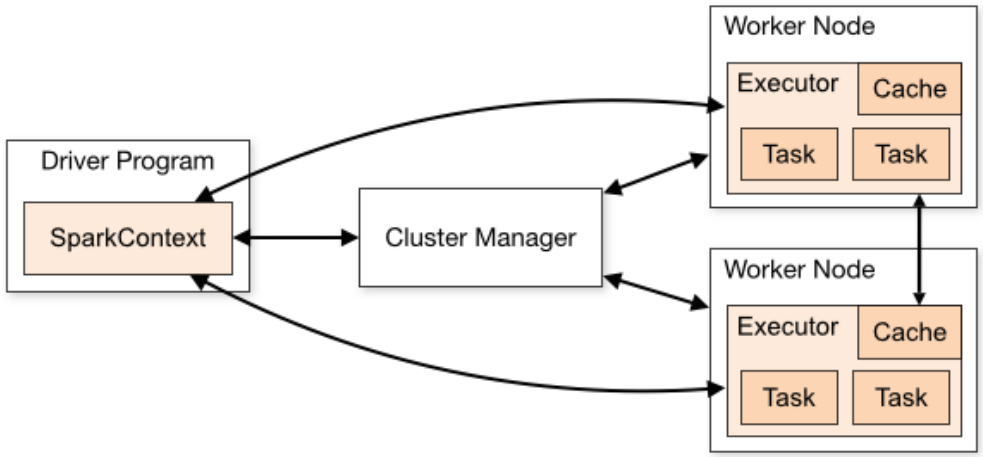
USING SPARK WITH MARKLOGIC



SPARK AND MARKLOGIC

Using the Hadoop Connector

- Spark has built-in support for loading/saving Hadoop data
- MarkLogic Hadoop Connector represents MarkLogic documents in Hadoop Compatible input/output formats
- MarkLogic Hadoop Connector is certified against Hortonworks and Cloudera platforms that ship with Spark

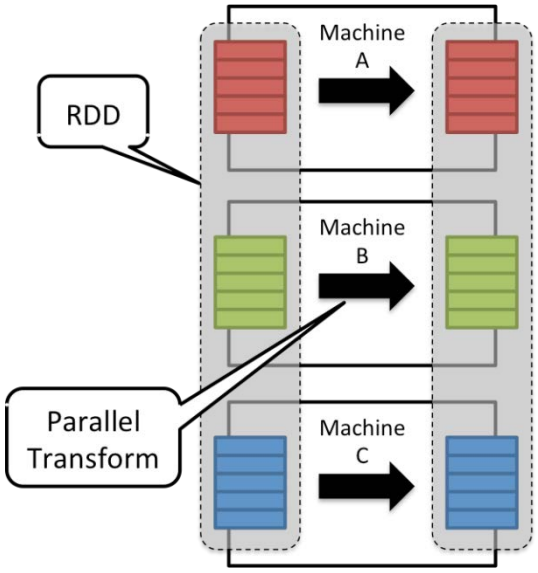


OVERVIEW OF SPARK

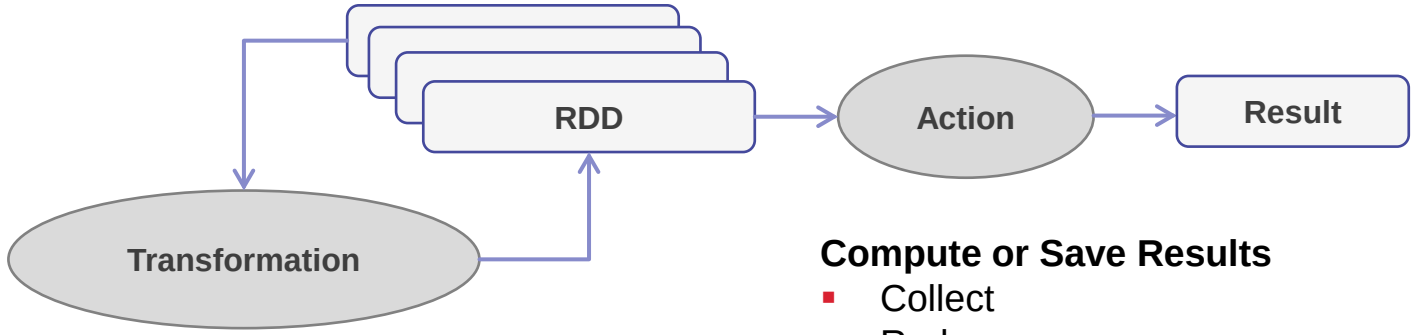
Cluster Computing Framework

- Each driver applications has its own executor processes in the cluster
- Resource Management via Cluster Manager
  - Standalone
  - Yarn (Using MarkLogic Hadoop Connector)
  - Mesos

Key Concept – Resilient Distributed Dataset (RDD)



- Collections of objects physically partitioned across cluster
- Stored in RAM or on Disk or Mixed
- RDD Operations - Transformations and Actions



Produce New RDD

- Map / FlatMap
- Filter
- SortByKey
- GroupByKey
- …

Compute or Save Results

- Collect
- Reduce
- Count
- Save
- …

Loading MarkLogic Data Into Spark RDD

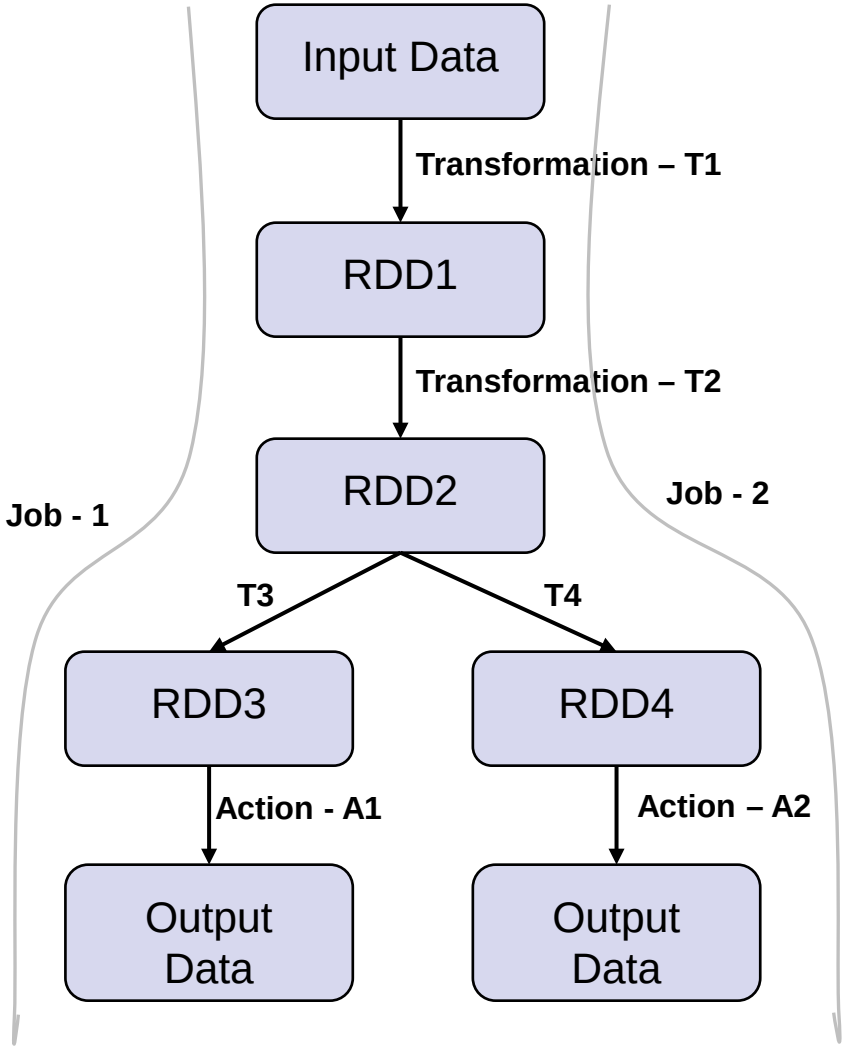
```
//first you create the spark context within java
SparkConf conf = new SparkConf().setAppName("com.marklogic.spark.examples").setMaster("local");
JavaSparkContext context = new JavaSparkContext(conf);

//Create configuration object and load the MarkLogic specific properties from the configuration file
Configuration hdConf = new Configuration();
FileInputStream ipStream =  new FileInputStream(configFilePath);
hdConf.addResource(ipStream);

//Create RDD based on documents within MarkLogic database.
//Load documents as DocumentURI, MarkLogicNode pairs.
JavaPairRDD<DocumentURI, MarkLogicNode> mlRDD = context.newAPIHadoopRDD(
                                                hdConf,                         //Configuration
                                                DocumentInputFormat.class,      //InputFormat
                                                DocumentURI.class,              //Key Class
                                                MarkLogicNode.class             //Value Class
);
```

For more details refer to [How to use MarkLogic in Apache Spark applications]

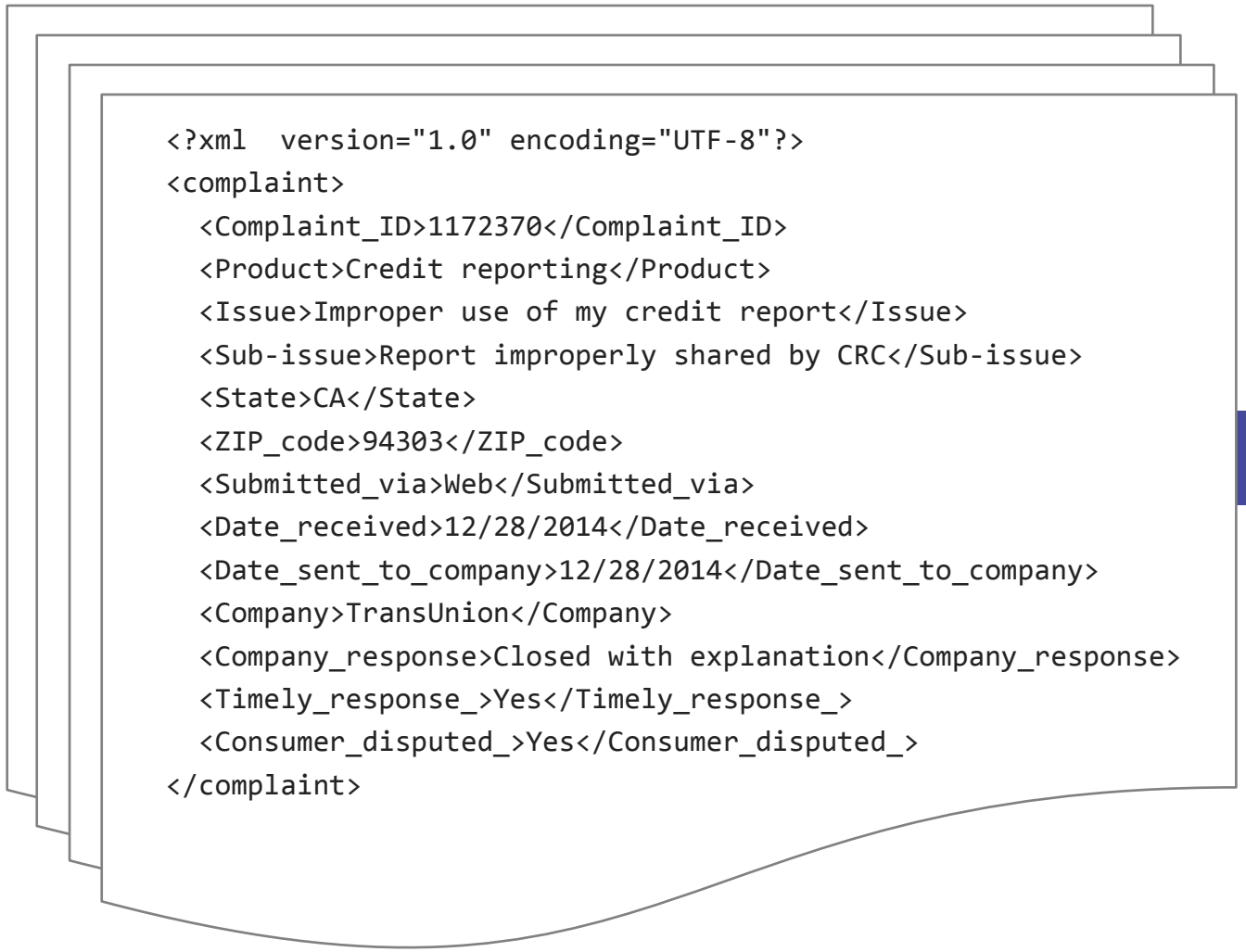
MarkLogic

Spark Data Processing Pipeline

- Built-In Optimizer for Data Processing
  - Lazy Transformations
  - Pipeline Execution
- Transformations and Data Partitioning
  - Narrow Transformations (No data shuffling)
    - Map, FlatMap, Filter, …
  - Wide Transformations (Data Shuffling)
    - SortByKey, ReduceByKey, GroupByKey, ….
- Tracks data lineage for re-computing RDD in case of failure



MarkLogic WordCount Transformation

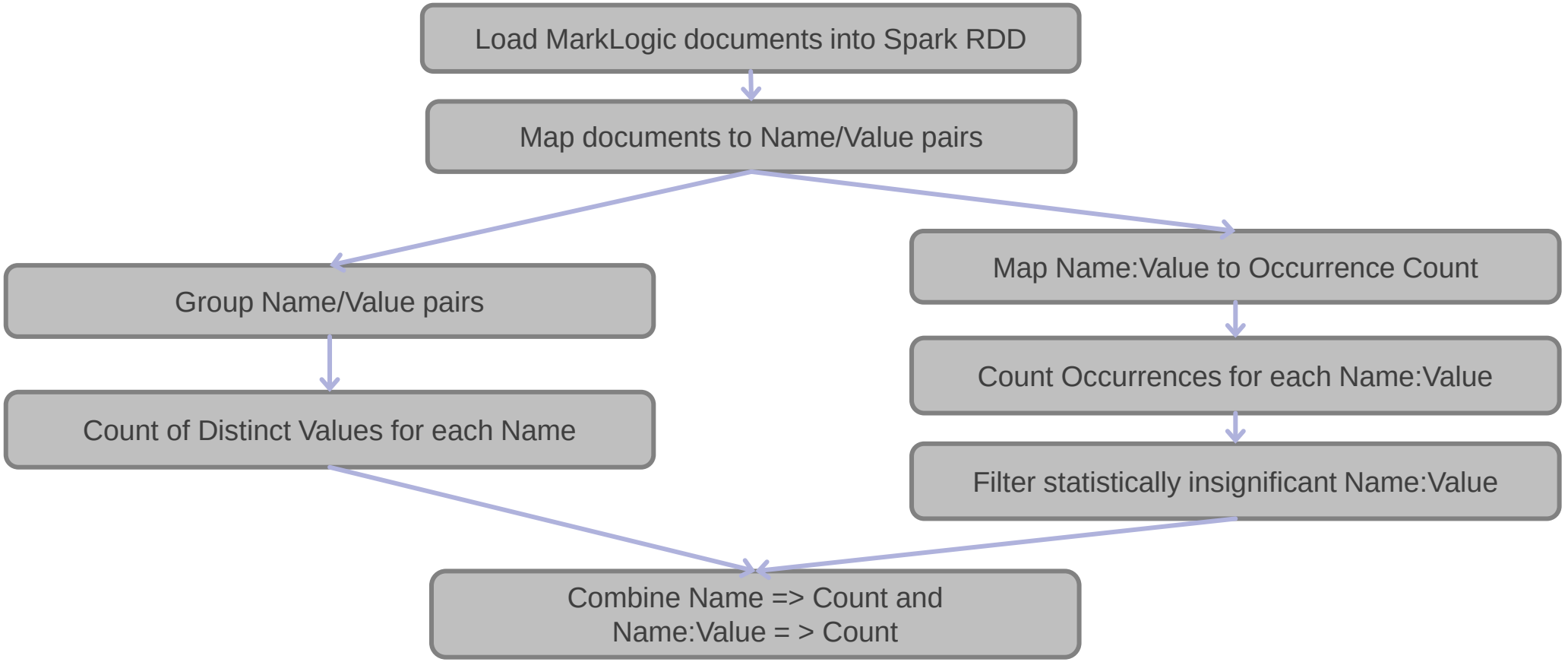


```
<?xml  version="1.0" encoding="UTF-8"?>
<complaint>
  <Complaint_ID>1172370</Complaint_ID>
  <Product>Credit reporting</Product>
  <Issue>Improper use of my credit report</Issue>
  <Sub-issue>Report improperly shared by CRC</Sub-issue>
  <State>CA</State>
  <ZIP_code>94303</ZIP_code>
  <Submitted_via>Web</Submitted_via>
  <Date_received>12/28/2014</Date_received>
  <Date_sent_to_company>12/28/2014</Date_sent_to_company>
  <Company>TransUnion</Company>
  <Company_response>Closed with explanation</Company_response>
  <Timely_response_>Yes</Timely_response_>
  <Consumer_disputed_>Yes</Consumer_disputed_>
</complaint>
```

```
...
...
(Product,11)
(Product:Bank account or service,44671)
(Product:Consumer loan,12683)
(Product:Credit card,48400)
(Product:Credit reporting,54768)
(Product:Debt collection,62662)
(Product:Money transfers,2119)
(Product:Mortgage,143231)
(Product:Other financial service,191)
(Product:Payday loan,2423)
(Product:Prepaid card,626)
(Product:Student loan,11489)
(State,63)
(State:,5360)
(State:AA,10)
(State:AE,141)
(State:AK,465)
...
...
```

MarkLogic

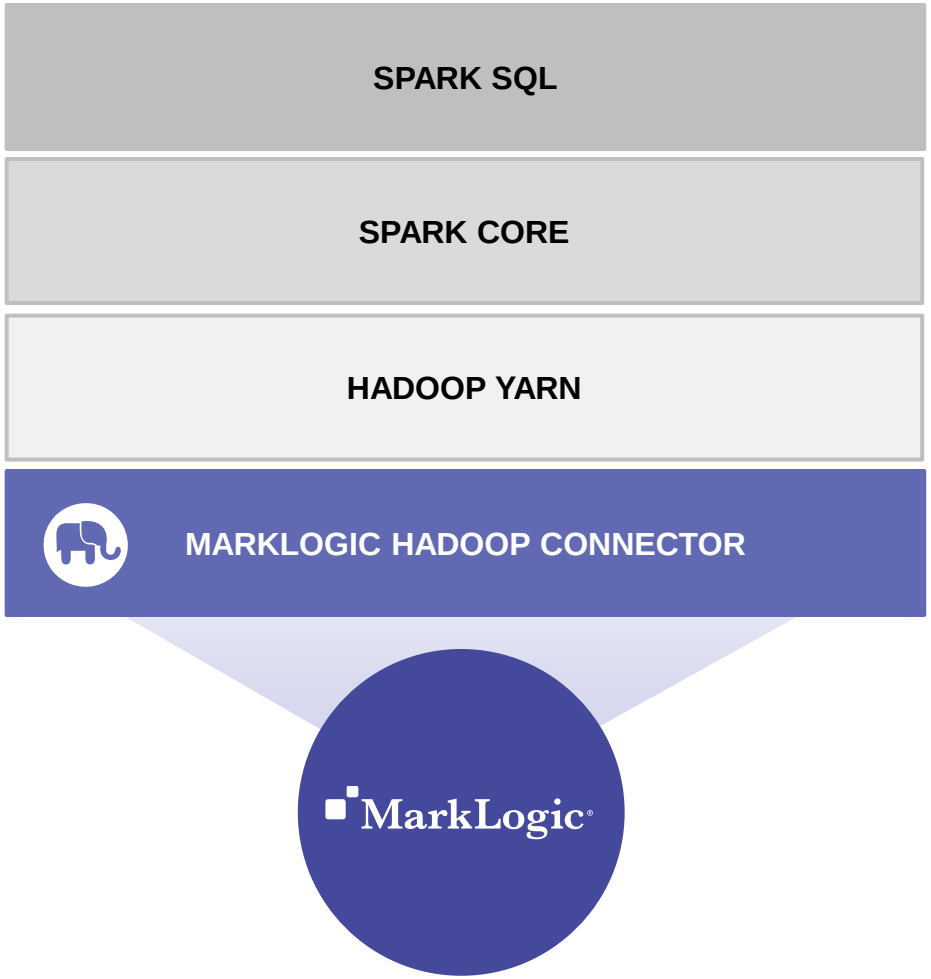
MarkLogic WordCount – Spark Data Pipeline



MarkLogic WordCount – Spark Data Pipeline

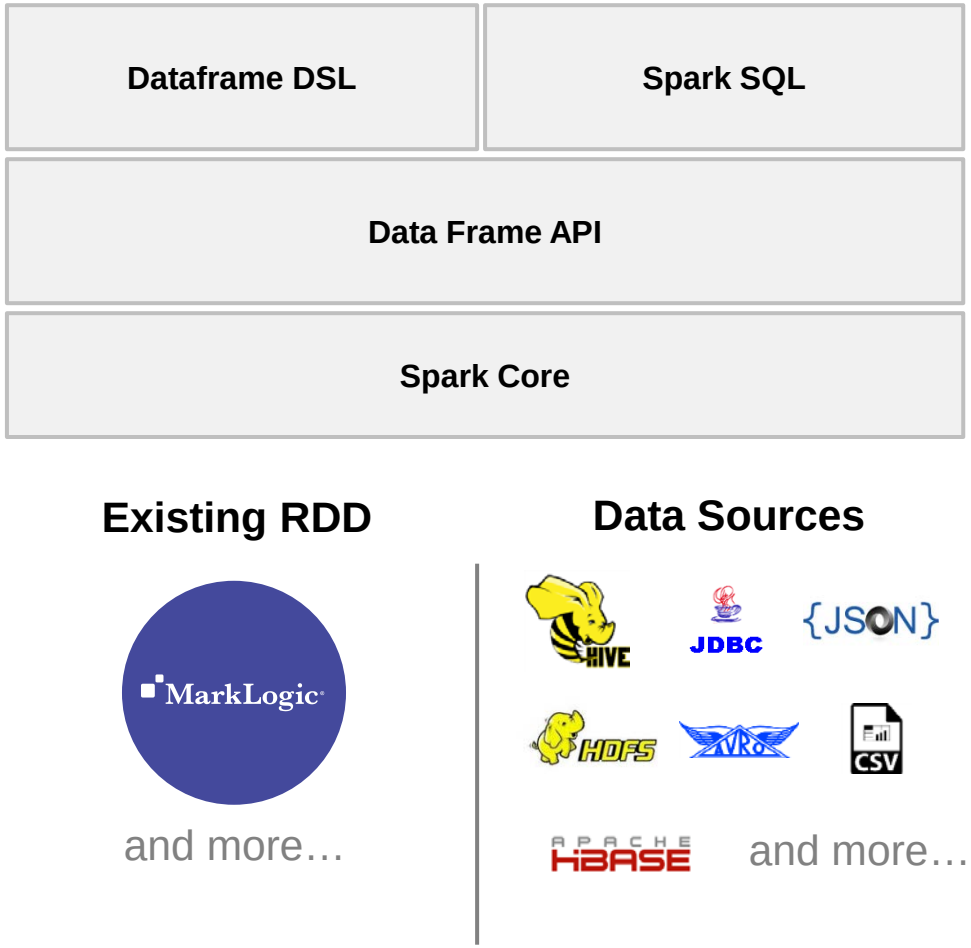
```
//Convert XML elements into name value pairs where element content is value
elementNameValues = mlRDD.flatMapToPair(ELEMENT_NAME_VALUE_PAIR_EXTRACTOR);

//Group element values for the same element name
elementNameValueGroup = elementNameValues.groupByKey();

//Count distinct values for each element name
elementNameDistinctValueCountMap = elementNameValueGroup.mapValues(DISTINCT_VALUE_COUNTER);

//Map the element name value pairs to occurrence count of each name:value pair
elementValueOccurrenceCountMap = elementNameValues.mapToPair(ELEMENT_VALUE_OCCURRENCE_COUNT_MAPPER);

//Aggregate the occurrence count of each distinct name:value pair.
elementValueOccurrenceAggregateCountMap = elementValueOccurrenceCountMap.reduceByKey(VALUE_COUNT_REDUCER);

//Filter out the name:value occurrences that are statistically insignificant
relevantNameValueOccurrences = elementValueOccurrenceAggregateCountMap.filter(ELEMENT_VALUE_COUNT_FILTER);

//Combine the distinct value count for each element and occurrence count for each name:value pair
valueDistribution = elementNameDistinctValueCountMap.union(relevantNameValueOccurrenceCounters);
```

For more details refer to [How to use MarkLogic in Apache Spark applications]



EXAMPLE

Spark SQL & MarkLogic



MARKLOGIC & SPARK

Spark SQL and DataFrames

- DataFrames
  - Abstraction for Structured Data – RDD + Schema
  - Container for Logical Plan
  - Multiple DSLs share same query engine/optimizer
- MarkLogic DataFrame is created based on RDD

1. Load MarkLogic Documents into Spark RDD

2. Create and configure SQLContext within your spark application

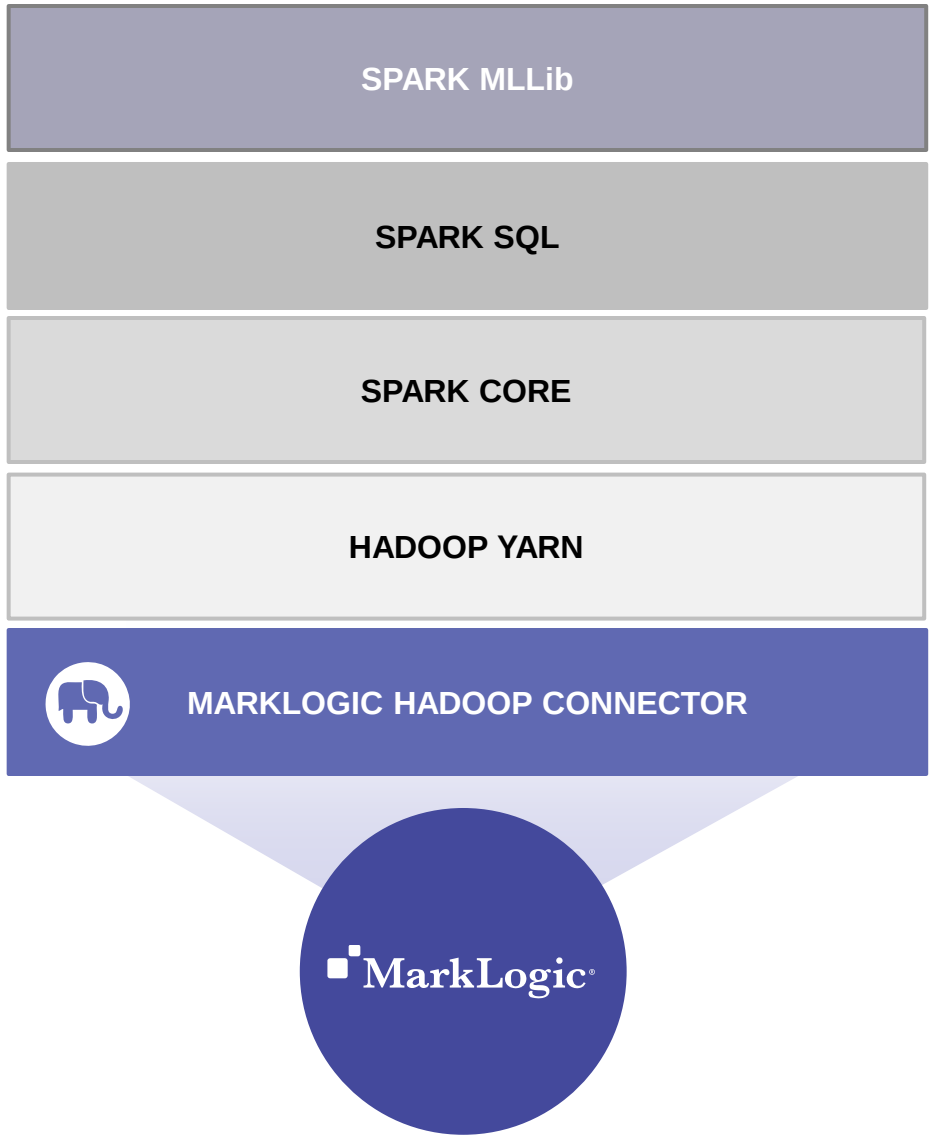
```
SQLContext sqlContext = new SQLContext(context);
sqlContext.setConf("spark.sql.shuffle.partitions", String.valueOf(10));
```

3. Create Spark DataFrame - Map document into tuple & apply schema

```
JavaRDD<ConsumerComplaint> complaints = mlRDD.map(CONSUMER_COMPLAINT_EXTRACTOR);
DataFrame sqlRDD = sqlContext.applySchema(ccComplaints, ConsumerComplaint.class);
```

4. Register a temporary table and execute SQL

```
sqlRDD.registerTempTable(“ConsumerComplaints");
DataFrame resultsRDD = sqlContext.sql("SELECT company, state, COUNT(complaintID) as NumComplaints " +
                                                              "FROM ConsumerComplaints" +
                                                              "GROUP BY company, state " +
                                                              "ORDER BY company, state");
```



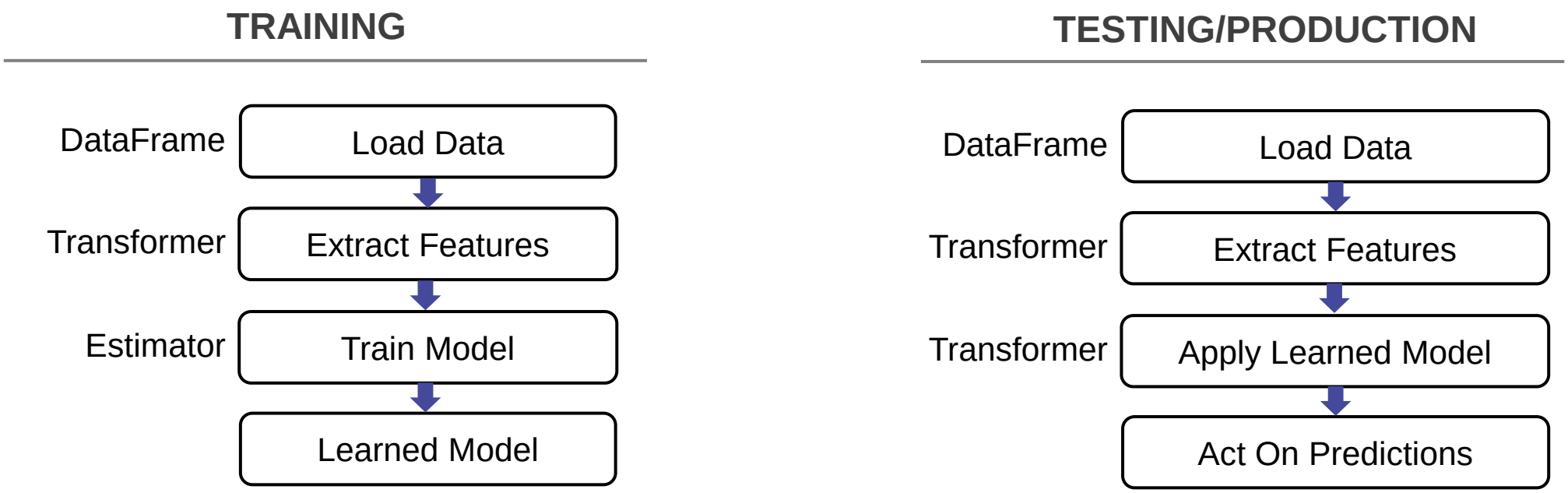
EXAMPLE

Spark Machine Learning & MarkLogic

MarkLogic

Spark Machine Learning – Pipeline Concepts

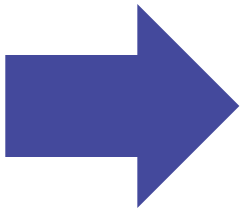
- **Transformer** – Abstraction for feature transformers and learned models
- **Estimator** – Abstraction of a learning algorithm; trains on data



MarkLogic

Credit Risk Assessment using Machine Learning

```
<?xml  version="1.0" encoding="UTF-8"?>
<CreditApplication>
  <AccountBalance> 2000 or Above </AccountBalance>
  <AccountDurationMonths> 12 </AccountDurationMonths>
  <CreditHistory>All credits paid back duly</CreditHistory>
  <CreditPurpose>Computer and Electronics</CreditPurpose>
  <CreditAmount> 1269 </CreditAmount>
  <LengthOfCurrentEmployment> 1-4 Years</LengthOfCurrentEmployment>
  <InstallmentPercentage> 13 </InstallmentPercentage>
  <Gender>female</Gender>
  <MaritalStatus> Married </MaritalStatus>
  <CurrentResidenceDuration> 3 years </CurrentResidenceDuration>
  <ValuableAssets> Real Estate </ValuableAssets>
  <Age> 42 </Age>
  <Housing> Rent </Housing>
  <NumberOfCreditsWithBank> 2 </NumberOfCreditsWithBank>
  <Occupation>Skilled Professional</Occupation>
  <NumberOfDependents> 1 </NumberOfDependents>
  ---
</CreditApplication>
```



```
<?xml  version="1.0" encoding="UTF-8"?>
<CreditApplication>
  <CreditRisk> --- </CreditRisk>
  <AccountBalance> ---
  <AccountDurationMonths> ---
  <CreditHistory> ---
  <CreditPurpose> ---
  <CreditAmount> ---
  <LengthOfCurrentEmployment> ---
  <Gender> ---
  <MaritalStatus> ---
  <CurrentResidenceDuration> ---
  <ValuableAssets> ---
  <Age> ---
  <Housing> ---
  <NumberOfCreditsWithBank> ---
  <Occupation> ---
  <NumberOfDependents> ---
  ---
</CreditApplication>
```

1. Load credit rating training data from MarkLogic into Spark DataFrame
2. Extract and transform credit rating features using VectorAssembler transformer

```
val assembler = new VectorAssembler().setInputCols(featureColumns).setOutputCol("features")
val featureVectors = assembler.transform(trainingData)
```

3. Transform the credit risk labels into ordered indices using StringIndexer

```
val labelIndexer = new StringIndexer().setInputCol(classColumn).setOutputCol("label")
val preparedTrainingSet = labelIndexer.fit(featureVectors).transform(featureVectors)
```

4. Train model using RandomForestClassifier estimator

```
val classifier = new RandomForestClassifier()
                        .setImpurity("gini")
                        .setMaxDepth(3)
                        .setNumTrees(20)
                        .setFeatureSubsetStrategy("auto")
                        .setSeed(5043)
model = classifier.fit(preparedTrainingSet)
```

1. Load new credit applications from MarkLogic into Spark DataFrame

2. Extract and transform credit rating features using VectorAssembler transformer

```
val assembler = new VectorAssembler().setInputCols(featureColumns).setOutputCol("features")
val creditFeatureVectors = assembler.transform(creditApplicationData)
```

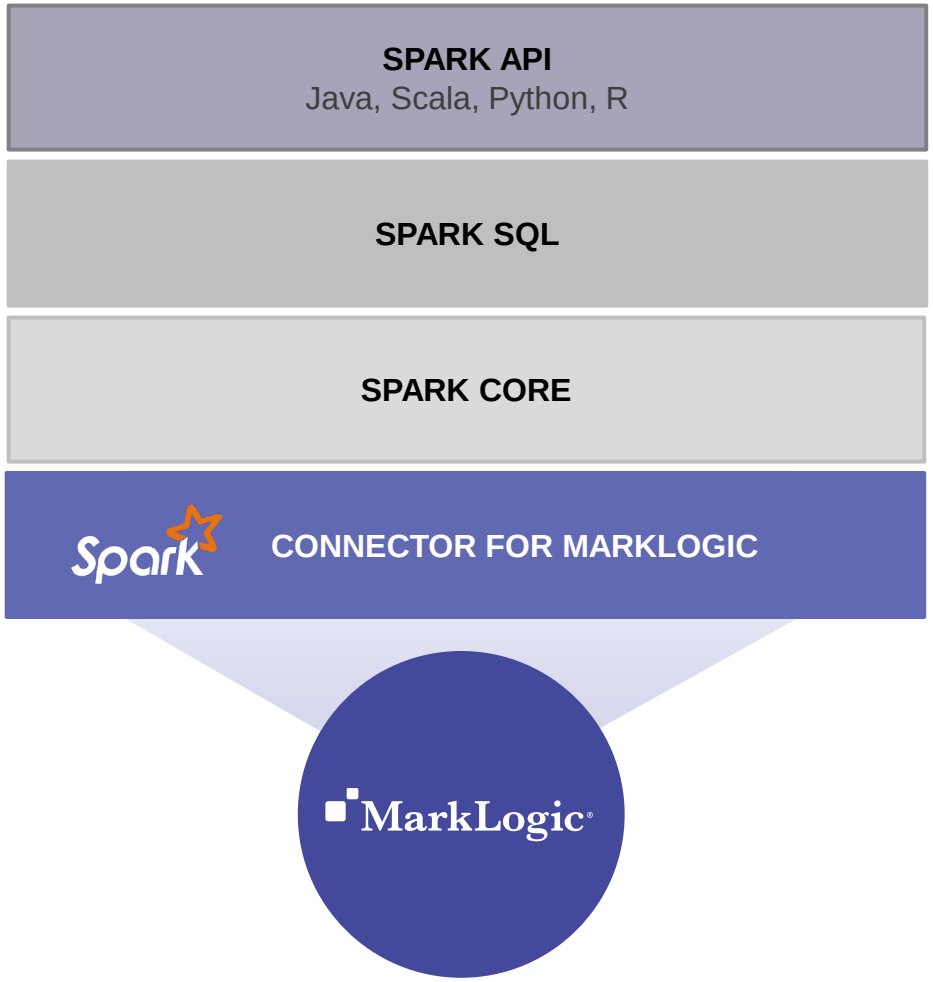
3. Apply the previously learned model to predict credit risk for new application

```
val predictions = model.transform(creditFeatureVectors)
```

4. Update the status of credit application in MarkLogic based on the prediction

WHAT'S NEXT

MARKLOGIC AND SPARK INTEGRATION

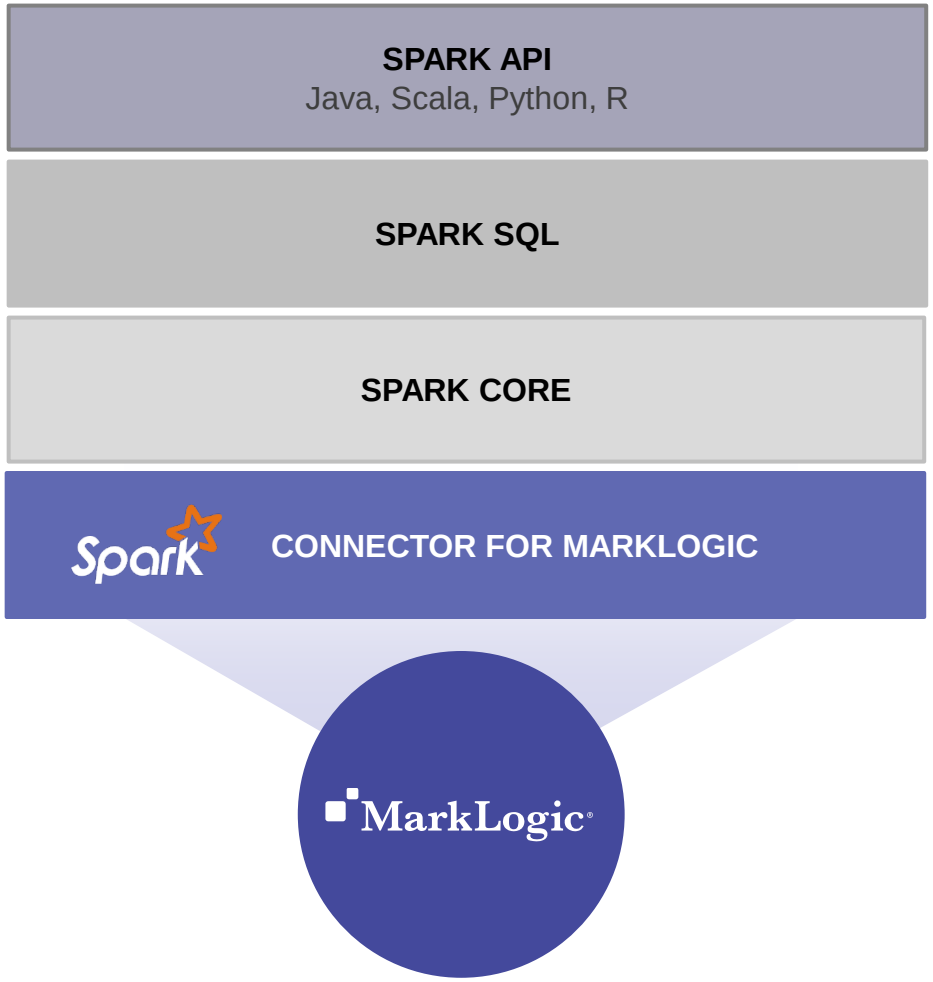


WHAT'S NEXT

Native Spark Connector for MarkLogic

- No runtime dependency on Hadoop / Yarn
- Simplified API for working with Spark RDD

```
//Create RDD based on documents within MarkLogic database.
sparkContext.newMarkLogicRDD(host, port, user, pwd, database, filterQuery);

//Save an arbitrary RDD to MarkLogic database
sparkContext.saveRDDToMarkLogicDatabase(host, port, user, pwd, database, …);
```



WHAT'S NEXT

MarkLogic Database as a SparkSQL Data Source

- Support Spark SQL connectivity via Data Source API
- Simplified API for working with Spark DataFrame

```
//Create DataFrame based on predefined views within MarkLogic database.
Dataframe df = sqlContext.read.MarkLogicView(host, port, …, …, viewName, filter);

//Save an arbitrary DataFrame to MarkLogic database
df.write.MarkLogic(documentURIMapper, [autoCreateView=false]);
```

MarkLogic

Key Takeaways

- Spark is open source big data processing engine (faster than Hadoop/MapReduce)
- MarkLogic's strength is in operational uses cases (i.e. highly concurrent transactional workload)
- MarkLogic and Spark are complementary in 'Operational + Analytical' use cases
- Write your Spark Application leveraging the MarkLogic Hadoop Connector
- Load MarkLogic data in a RDD and/or a DataFrame and use it in Spark apps
- *What's Next* – Native Spark connector for MarkLogic

MarkLogic

More Information

- Blog on Developer Community
  How to use MarkLogic in Apache Spark applications
- GitHub Repository with example application
  https://github.com/HemantPuranik/MarkLogicSparkExamples

Q&A