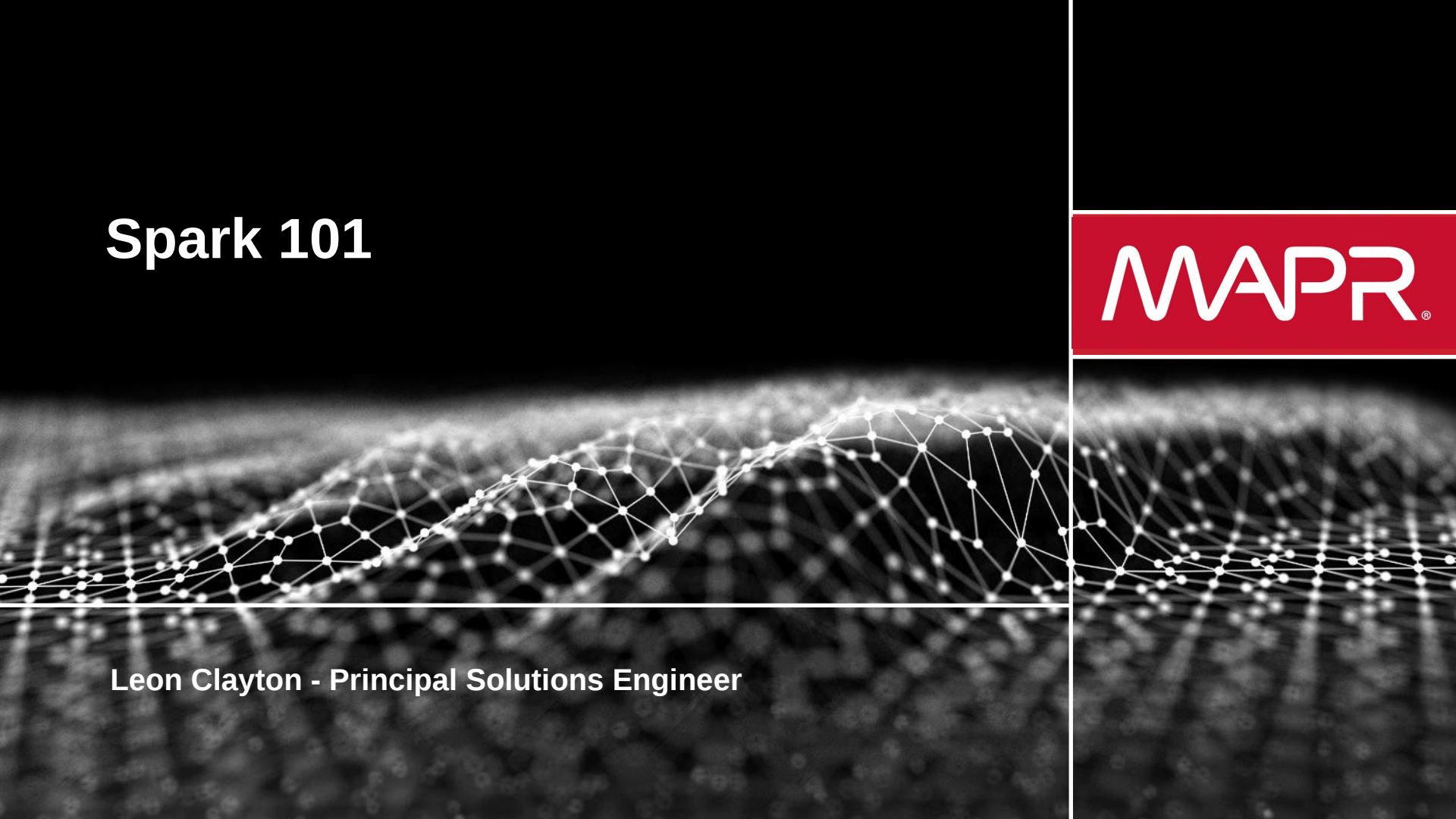
# Spark 101



Leon Clayton - Principal Solutions Engineer

# {"about" : "me"}

**Leon Clayton**

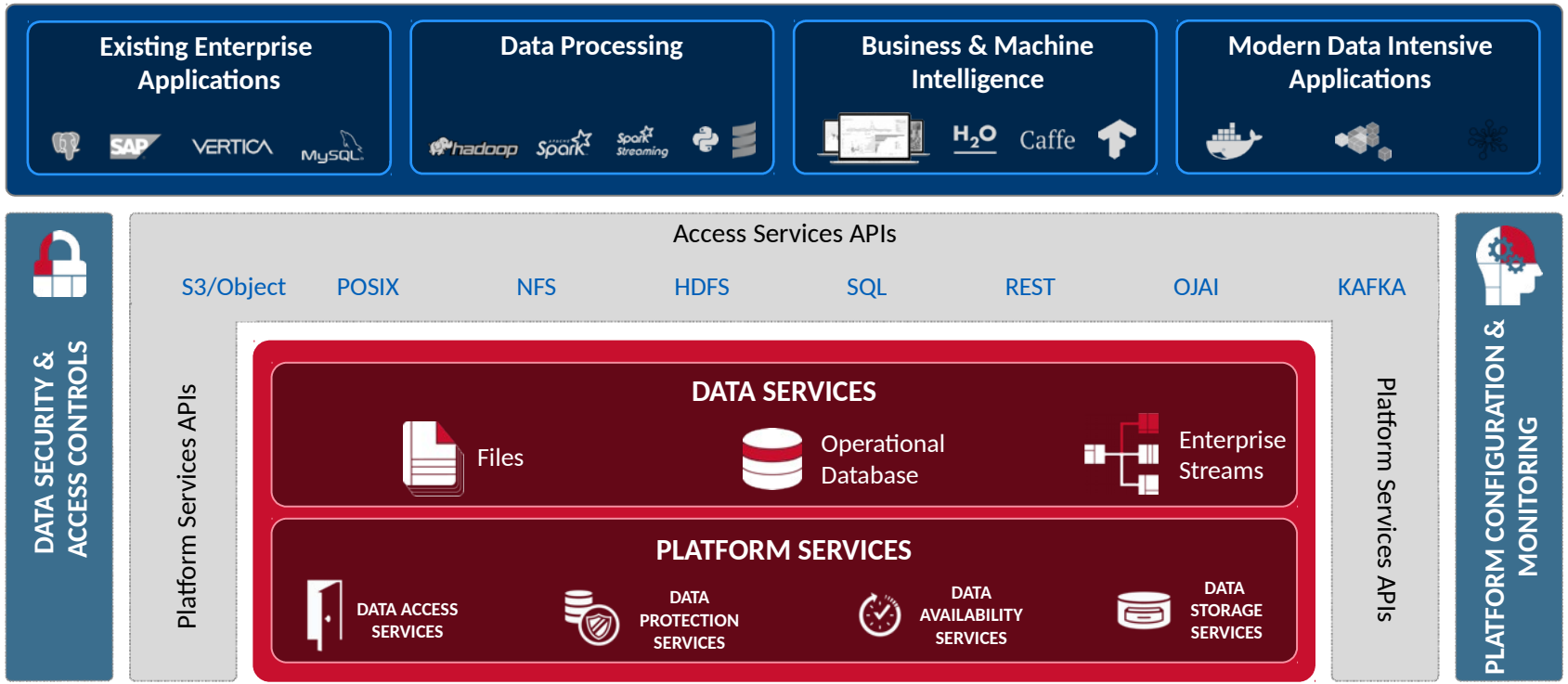
- MapR
  - Solution Architect
- EMC
  - EMEA Isilon Lead
- NetApp
  - Performance Team Lead
- Philips
  - HPC team lead
- Royal Navy
  - Type 42 – Real time systems engineer


- lclayton@mapr.com


MAPR

# Foundation For A Global Data Fabric

## MapR Converged Data Platform



# Streaming Use Cases

# Use Cases

## Financial Services

- Fraudulent Transactions
- Surveillance
- Anomaly Detection
- Mobile Notifications of Transactions

## Retail

- Recommendation Engines
- Supply Chain Optimisation
- Clickstream Analysis
- Personalised Customer Experience
- Ad Targeting

## Smart Cities

- Traffic Congestion Rerouting
- Rail, Tube, Buses, Telemetry data.
- Emergency Incidents

## Manufacturing

- Predictive / Proactive Asset Maintenance
- Real-Time Dashboards

# Use Case - Database Change Capture For Smart Credit Card Processing

**Business Results** -

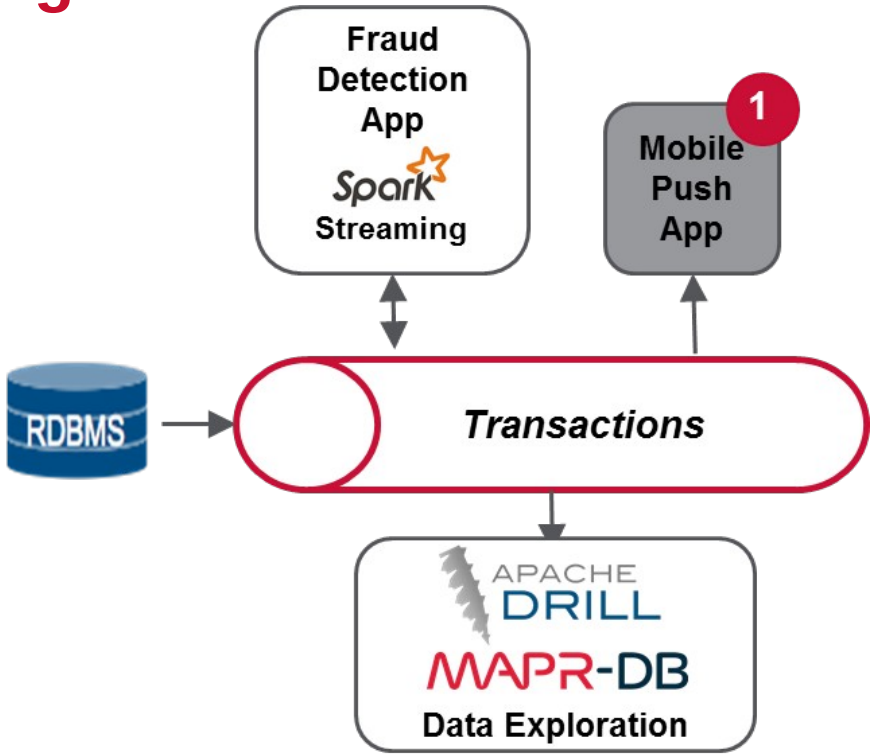
- Improved user satisfaction with real-time mobile notifications of purchases.
- More fraud detected in real-time.
- More productive staff with data exploration.

**Why Streams** -

- Seamless, real-time connection between mainframe RDBMS and ETL/processing.

**Why MapR** -

- Utility-grade reliability means no transactions are lost.
- Converged platform security gives unified authentication, authorisation, encryption.



# Use Case - Application/Infrastructure Monitoring


**Business Results**

- Real-time detection and alerting on failures, security breaches
- Global dashboards on utilisation, availability, performance

**Why Streaming**

- Real-time delivery from apps/infra to ETL & processing systems.
- Reliable buffering of data in case of slow/failing systems.

**Why MapR**

- Converged platform brings all components together.
- Global event replication enables centralised monitoring.
- Secure multi-tenancy allows cluster sharing.

# Use Case - Stream System of Record for Finance

**Business Results** -

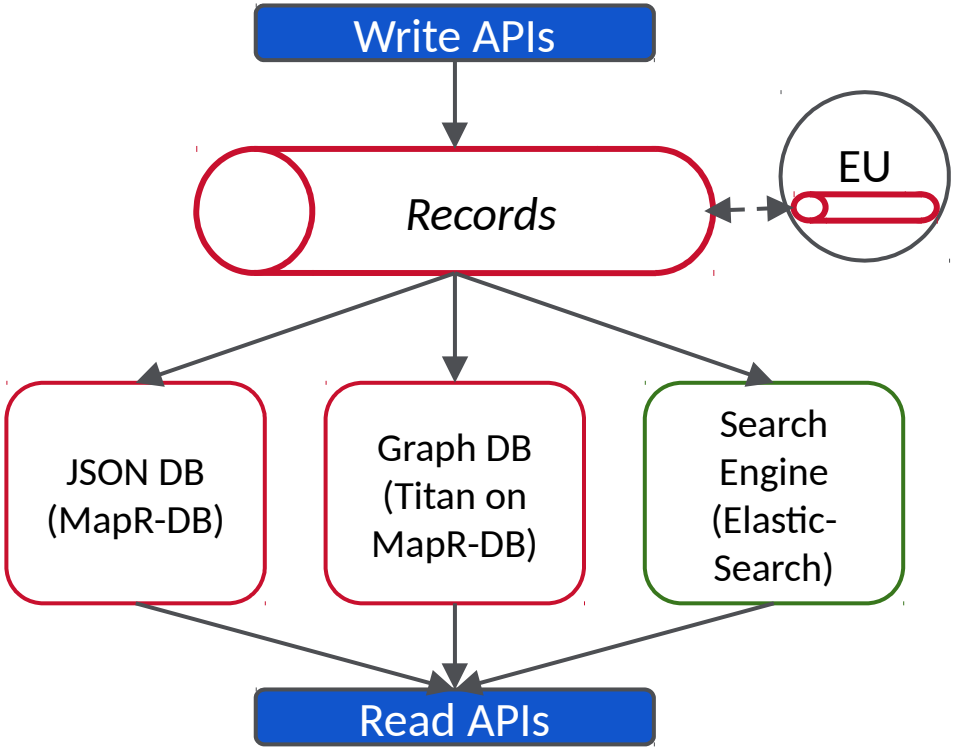
- Data agility - up-to-date views in JSON, Graph, Search formats.
- HIPAA, PCI Compliance.
- Compliance with in-country data regulations.

**Why Streams** -

- Streams are immutable, re-windable, and auditable data structures.
- Pub-sub allows real-time replication to JSON-DB, Graph DB, ElasticSearch.

**Why MapR** -

- Converged platform gives single cluster, single security model for data in motion and at rest.
- Selective, reliable global replication for DR.



# Use Case - IoT Data Transport & Processing

**Business Results** -

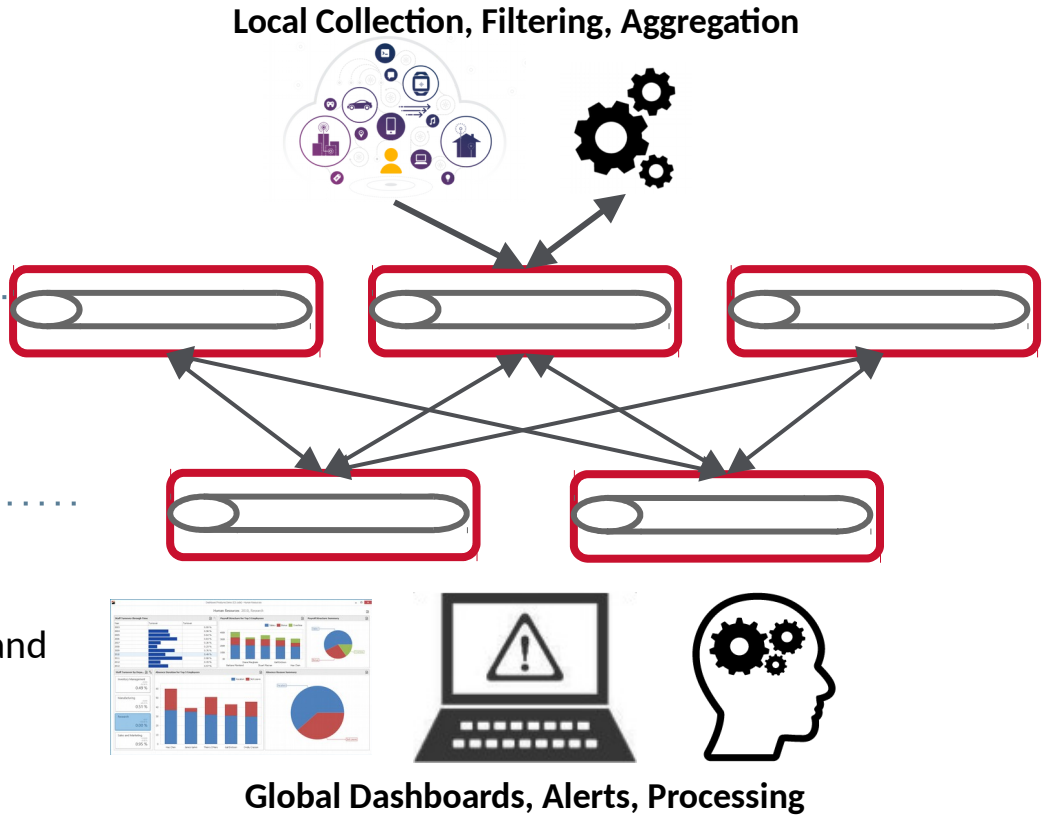
- New revenue streams from collecting and processing data from “things”.
- Low response times by placing collection and processing near users.

**Why Streams** -

- IoT is event-based, and needs an event streaming architecture.

**Why MapR** -

- Converged platform gives single cluster, single security model for data in motion and at rest.
- Reliable global replication for distributed collection, analysis, and DR.



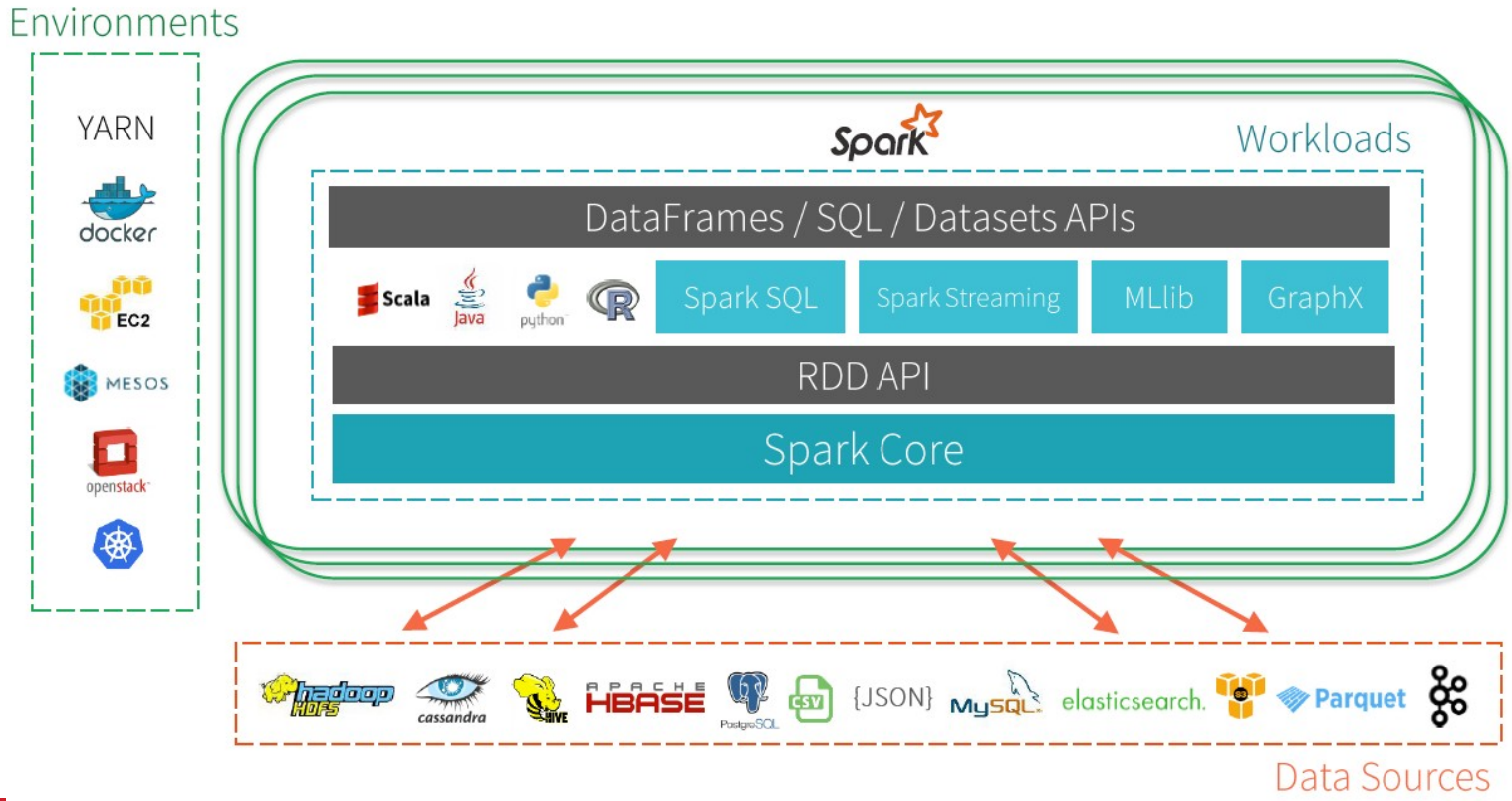
What is Apache Spark?
Why Apache Spark?

# What is Apache Spark?


- Unified Data Processing Engine across Data Sources, Workloads and Environments.
- **Data Sources**: File System, Databases, Streams.
- **Workloads**: Batch, Interactive, Iterative, Streaming.
- **Environments**: YARN, Mesos, Docker.

- Run computations in memory

# Apache Spark Architecture



# Why Apache Spark?

## Fast

- 10x faster on disk
- 100x in memory

## Ease of Development

- Write programs quickly
- More operators
- Interactive shell
- Less code

## Multi-language support

- Scala
- Python
- Java
- SparkR

## Unified Stack

Builds applications combining different processing models

- Batch
- Streaming
- Interactive Analytics

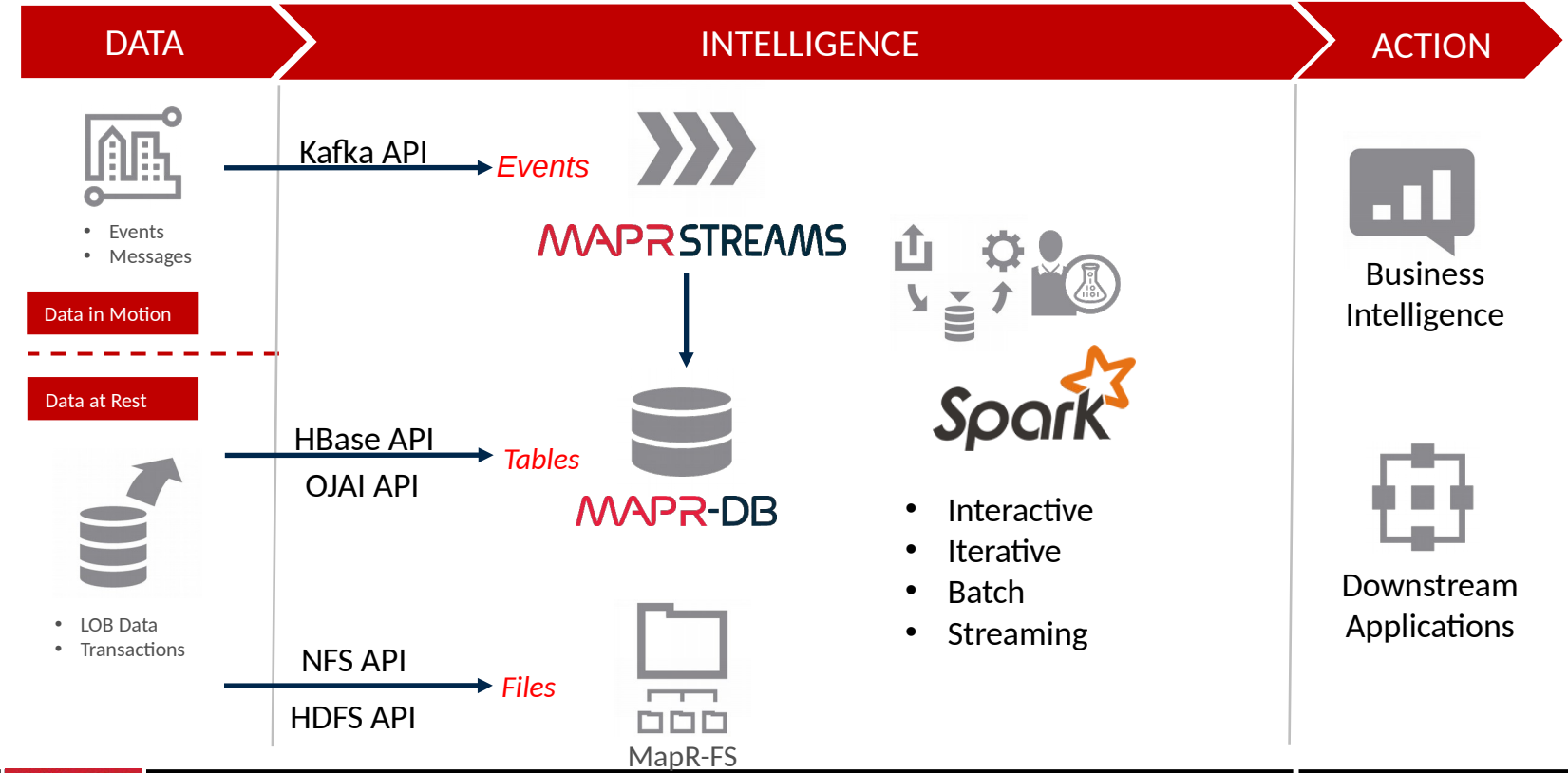
## Deployment Flexibility

- Deployment
  - Mesos, YARN
  - Standalone
  - Local
- Storage
  - HDFS, S3

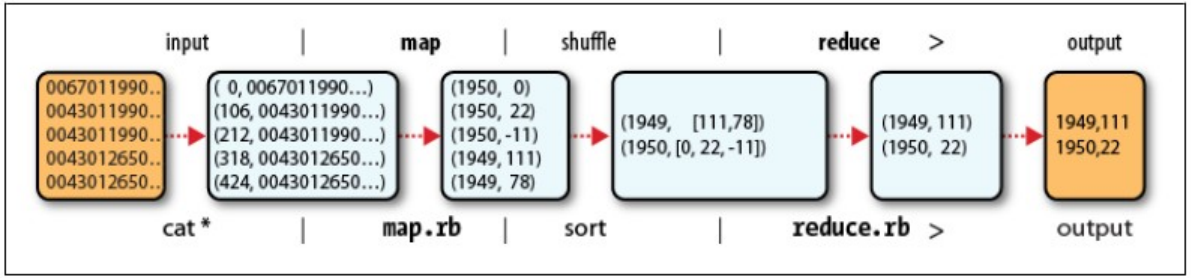
# Simplification


| | | | |
|---|---|---|---|
| Execution Engine | Microsoft Dryad | TEZ, CASCADING | **Spark Core** |
| Batch Processing | Google Dremel | APACHE DRILL | **Spark SQL** |
| Stream Processing | Google MillWheel | STORM, samza | **Spark Streaming** |
| Machine Learning | SAMOA | Open for Innovation KNIME, mahout | **MLlib** |
| Graph Processing | Google Pregel | APACHE GIRAPH, Dato GraphLab | **GraphX** |

# Simplified Data Architecture



# MapReduce vs Spark




- Write Java Code:
  - Mapper Code, Reducer Code
  - Driver Code
  - Serialisations for Input / Output formats.

# Word Count: MapReduce vs Spark

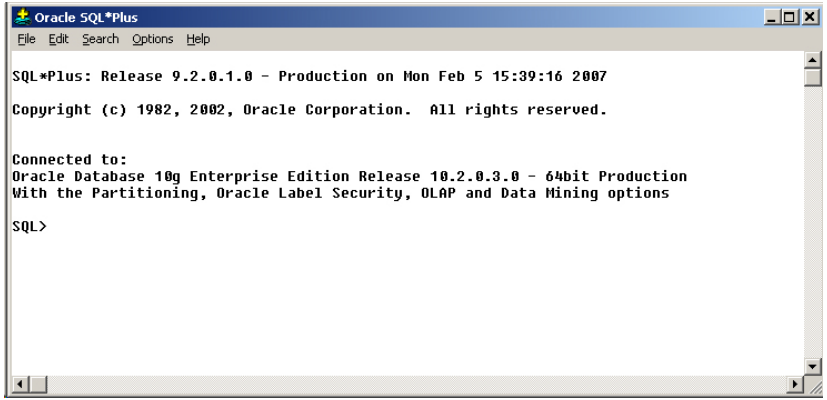
```
wordCounts = textFile.flatMap(line => line.split(" "))
                     .map(word => (word, 1))
                     .reduceByKey(_ + _)
```

```
public class WordCount {

   public static class TokenizerMapper extends
      Mapper<Object, Text, Text, IntWritable> {

      private final static IntWritable one = new
IntWritable(1);
      private Text word = new Text();
      public void map(Object key, Text value,
Context context)
         throws IOException,
InterruptedException {
         StringTokenizer itr = new
StringTokenizer(value.toString());
         while (itr.hasMoreTokens()) {
            word.set(itr.nextToken());
            context.write(word, one);
         }
      }
   }
```

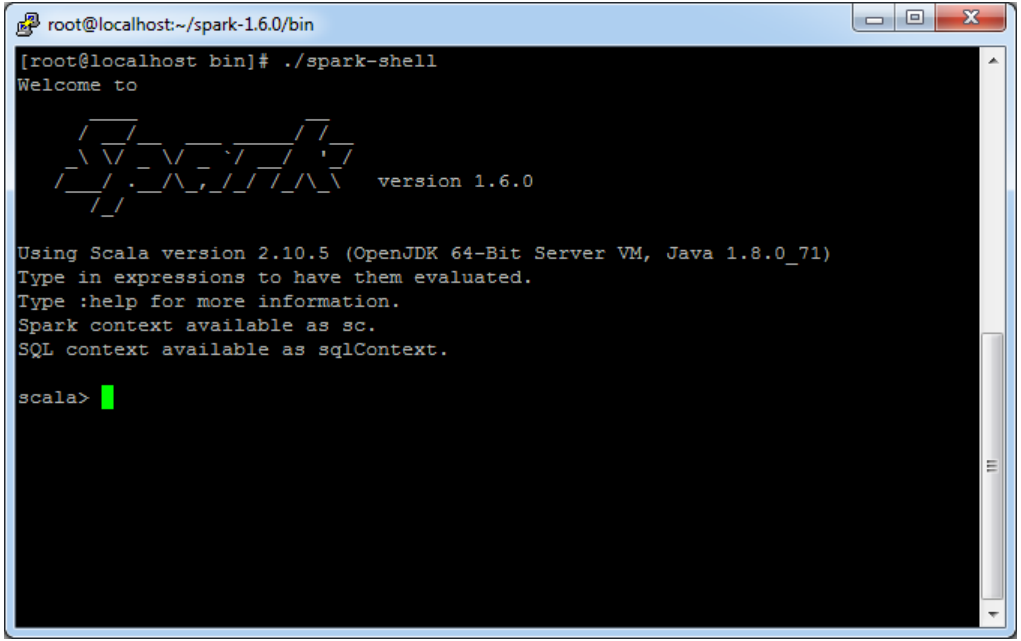
```
 public static class IntSumReducer extends
      Reducer<Text, IntWritable, Text, IntWritable> {

      private IntWritable result = new IntWritable();
      public void reduce(Text key,
Iterable<IntWritable> values,
      Context context) throws IOException,
InterruptedException {
         int sum = 0;
         for (IntWritable val : values) {
            sum += val.get();
         }
         result.set(sum);
         context.write(key, result);
      }
   }
```

```
 public static void main(String[] args) throws Exception {
      Configuration conf = new Configuration();
      String[] otherArgs = new GenericOptionsParser(conf,
args)
         .getRemainingArgs();
      Job job = new Job(conf, "word count");
      job.setJarByClass(WordCount.class);
      job.setMapperClass(TokenizerMapper.class);
      job.setCombinerClass(IntSumReducer.class);
      job.setReducerClass(IntSumReducer.class);
      job.setOutputKeyClass(Text.class);
      job.setOutputValueClass(IntWritable.class);
       FileInputFormat.addInputPath(job, new
Path(otherArgs[0]));
      FileOutputFormat.setOutputPath(job, new
Path(otherArgs[1]));

      System.exit(job.waitForCompletion(true) ? 0 : 1);
   }
}
```

# MapReduce vs Spark (read–eval–print loop (REPL))




Oracle SQL Plus,
Unix Shell

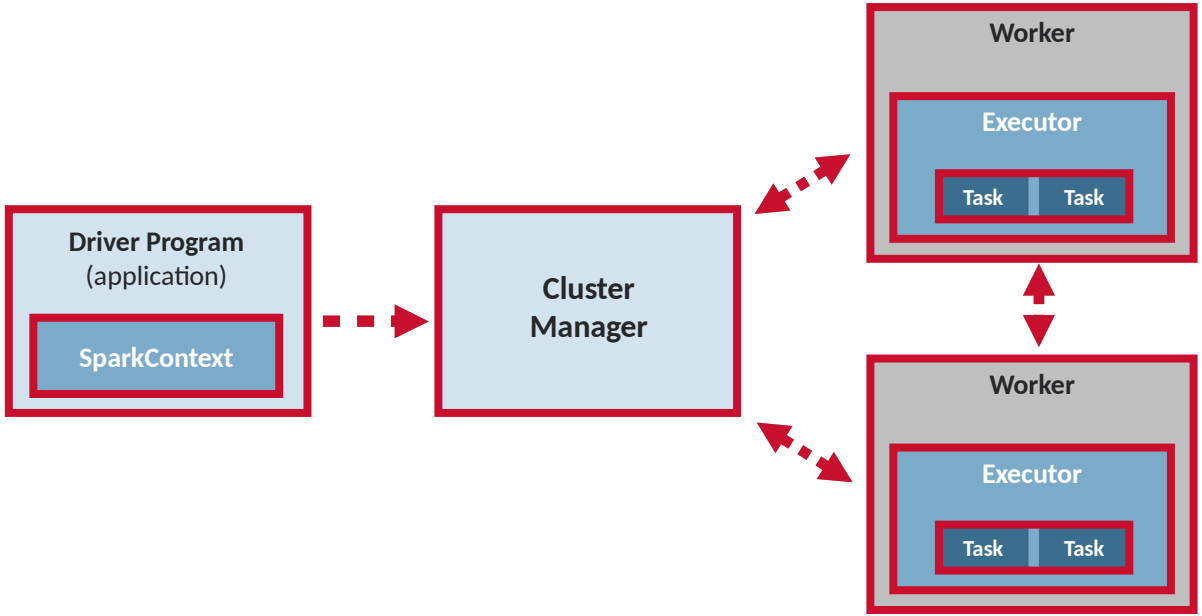



Spark Interactive Shell

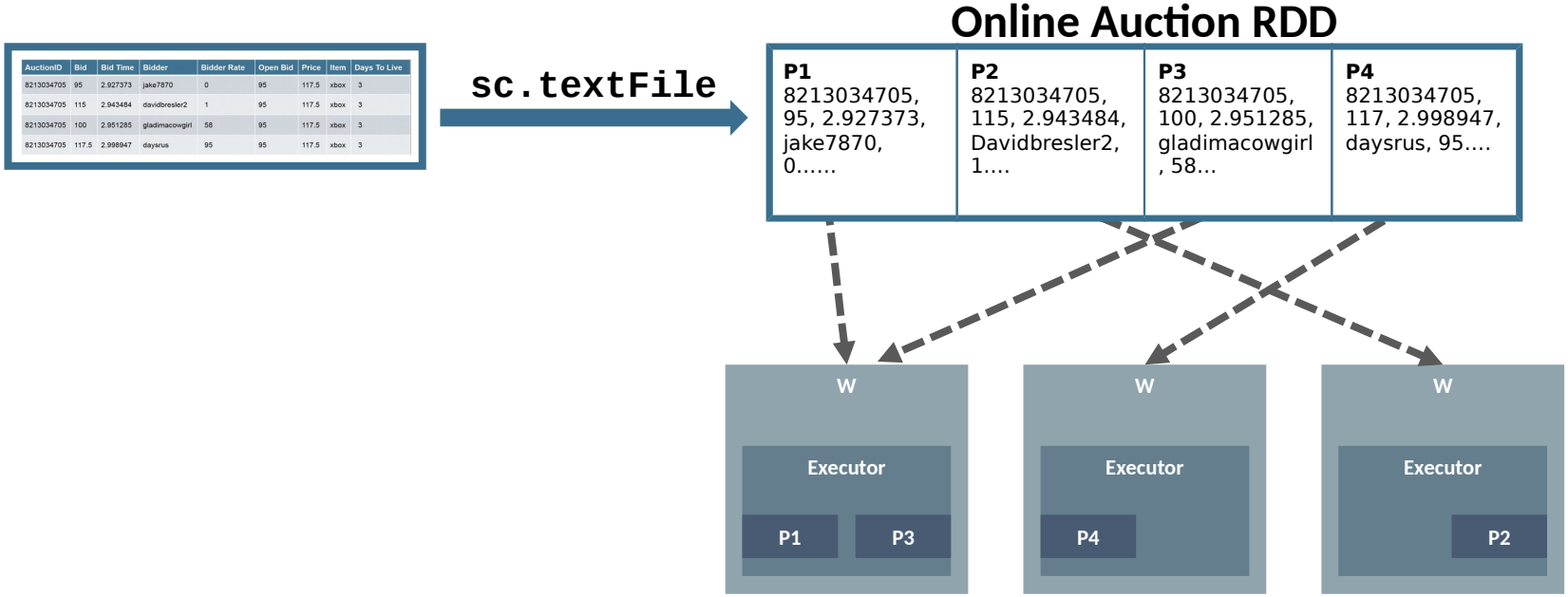
# Spark on Hadoop

| Hadoop | + | Spark | = | Hadoop & Spark |
|---|---|---|---|---|
| • Unlimited scale<br>• Multi-tenant<br>• Reliable<br>• Range of applications<br>• Files, semi-structured data, databases | | • Parallel in-memory processing across platform<br>• Multiple data sources, applications, users | | • Analytics over operational Hadoop applications<br>• Reliable unlimited scale<br>• Analytics over operational applications |

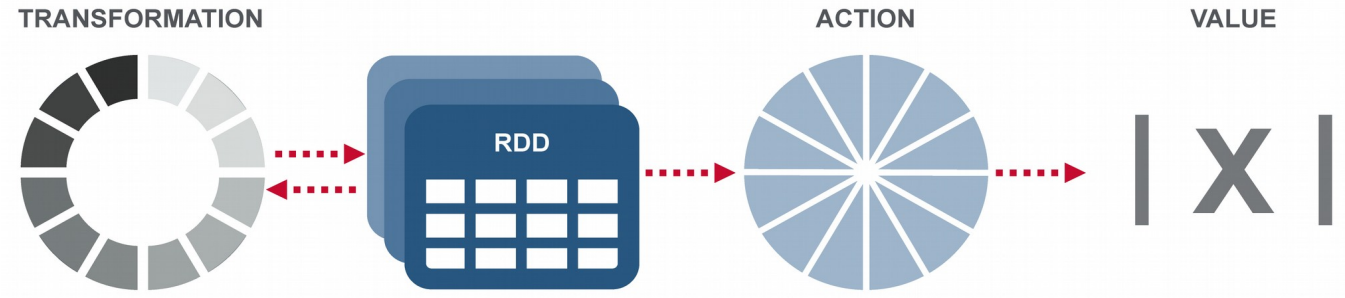
# Spark Components



# Spark Resilient Distributed Datasets



# Spark Resilient Distributed Datasets



# Dataset Operations

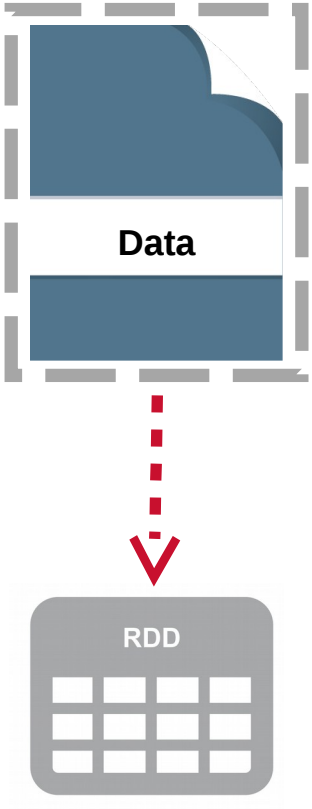
## Transformations

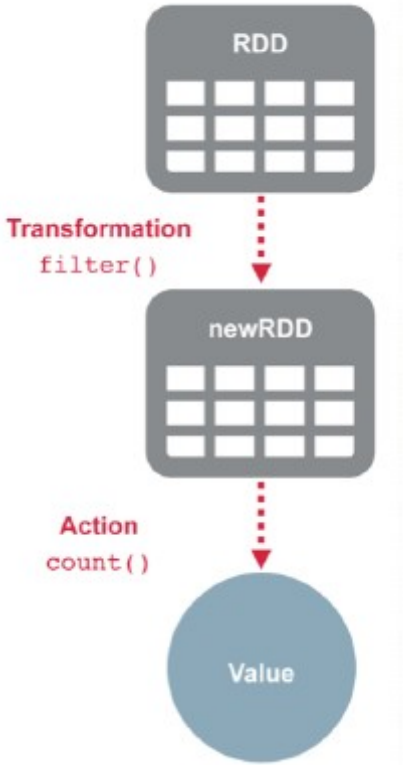
- Creates new dataset from existing one
- Transformed RDD executed only when action runs on it
- Examples: filter(), map(), flatMap()

## Actions

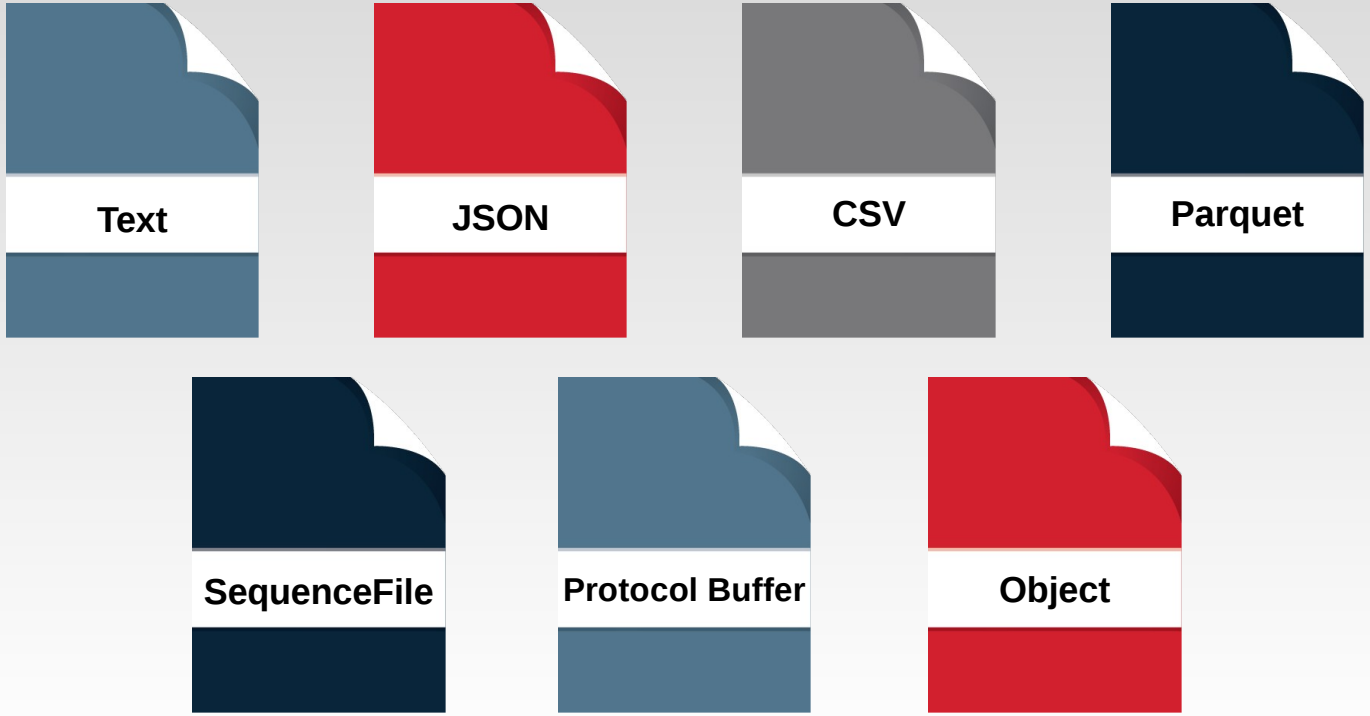
- Return a value to driver program after computation on dataset
- Examples: count(), reduce(), take(), collect()

# Lazy Evaluation

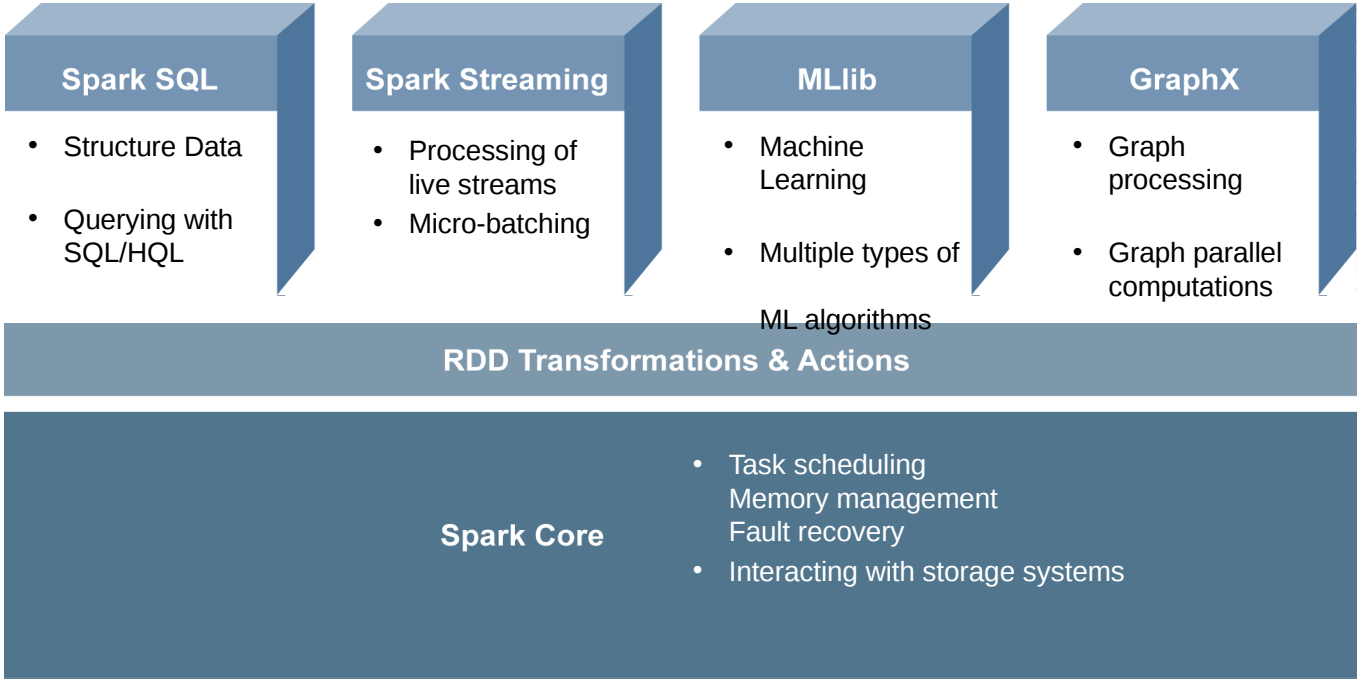


# Data Formats



# OOTB Spark Libraries



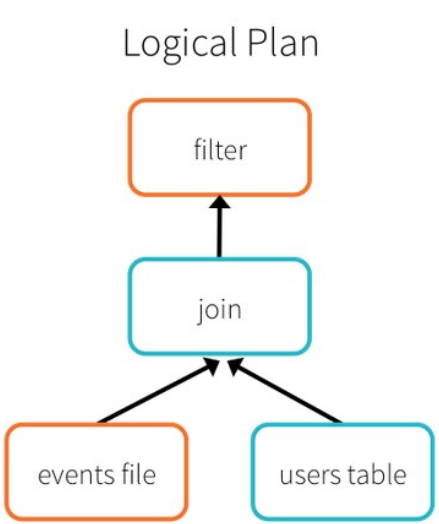
# Spark SQL - DataFrame

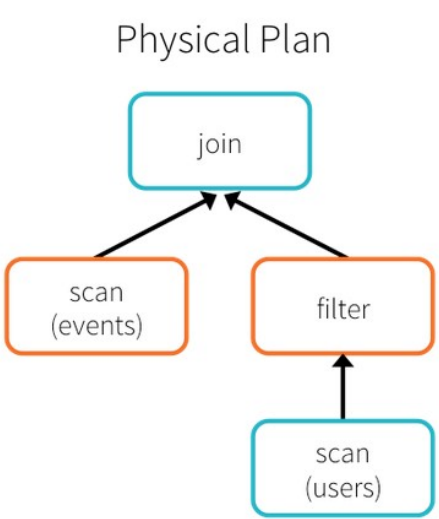
- SparkSQL is a library that runs on top of Apache Spark Core.
- SQL interface via interactive shell, JDBC/ODBC or through DataFrame API.
- Spark DataFrames use a relational optimiser called Catalyst.
- SQL queries are constructed against DataFrames are optimised and executed as sequences of Spark Jobs.
- Distributed collection of data organised into named columns.
- Easily load from/via JDBC, JSON, Parquet, Hive etc.
- Allows SQL-style operations on RDDs
  - Join, groupBy, select etc.

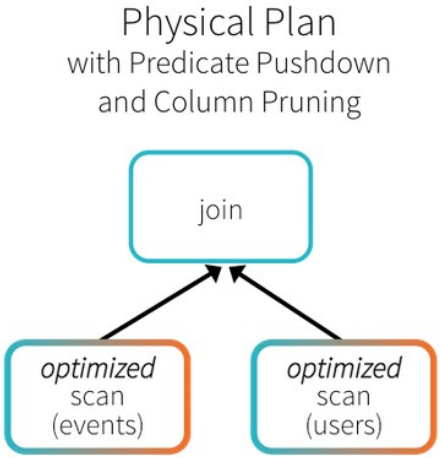
# Spark SQL - DataFrames

```
val sqlContext = new SQLContext(sparkContext)
val user = sqlContext.read.jdbc("jdbc://", "user", properties)
val event = sqlContext.read.parquet("/my/path/event.parquet")

val joinedTable = event
    .join(user, event("user_id") === user("id"))

val dataOfInterest = joinedTable
    .where(joinedTable("city") === "Amsterdam")
    .select("event_time")
```

# Spark SQL - DataFrames

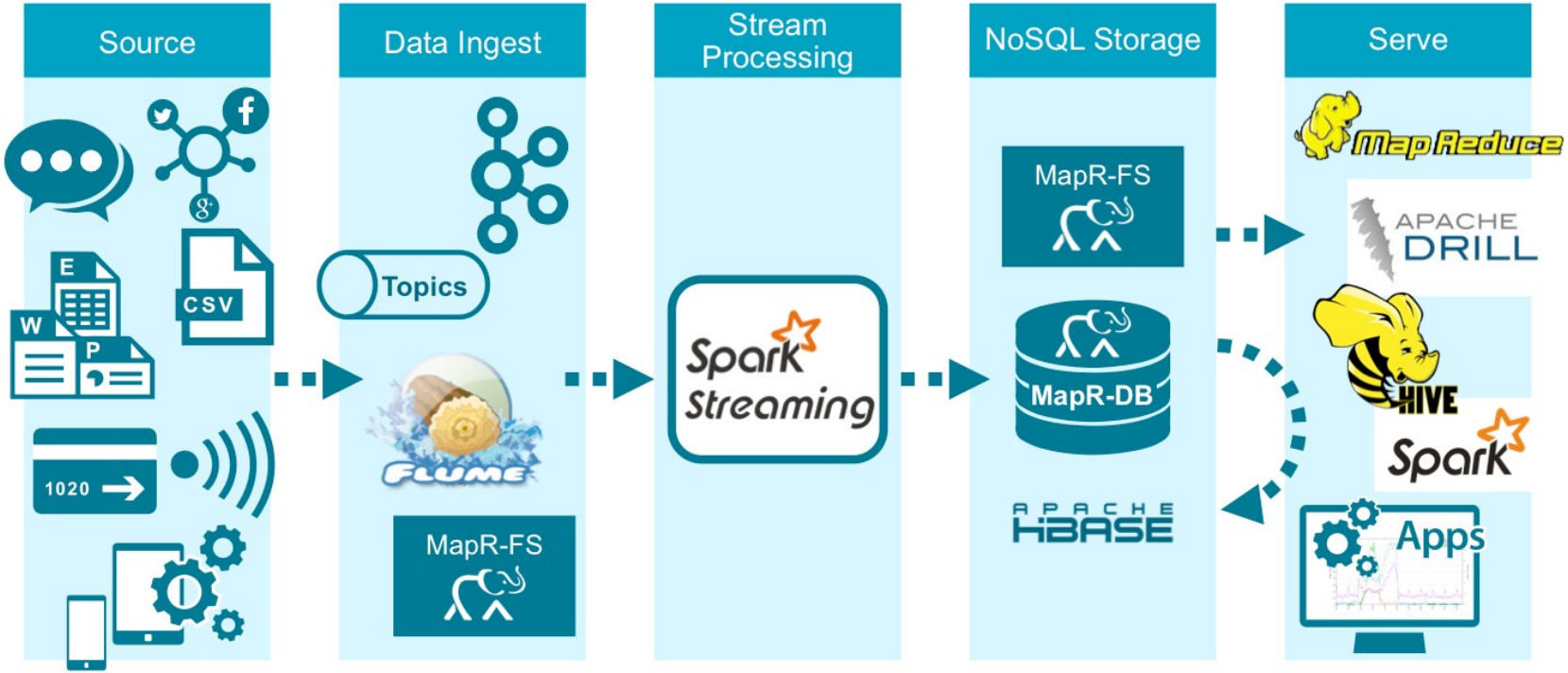


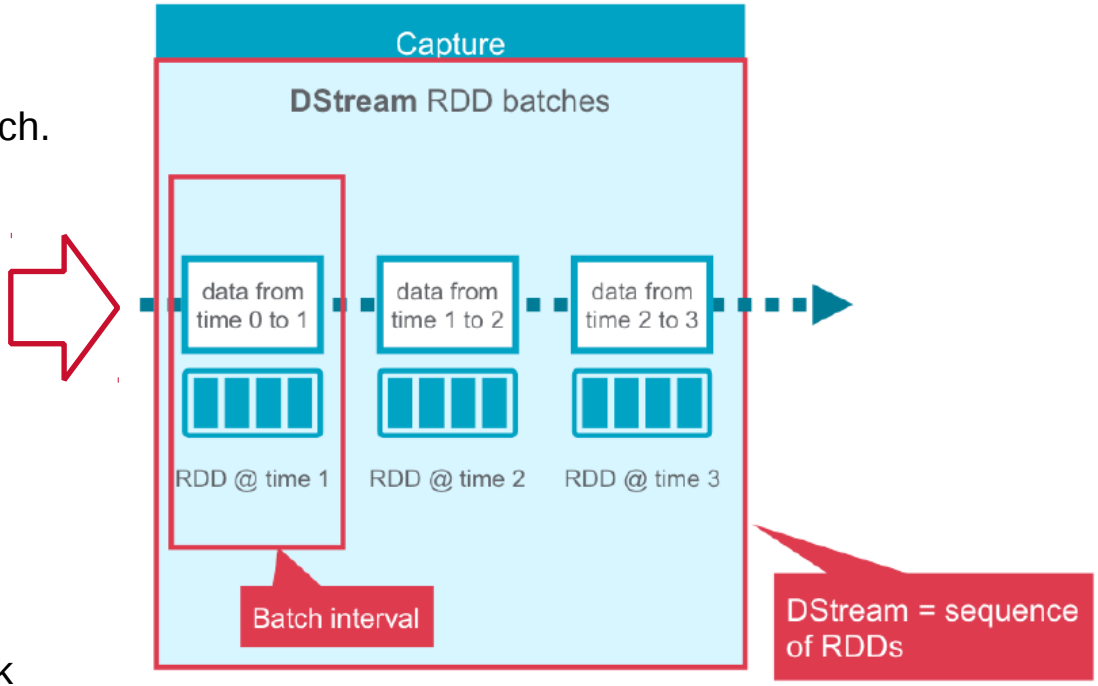
# Stream Processing Architecture

# Stream Processing Architecture



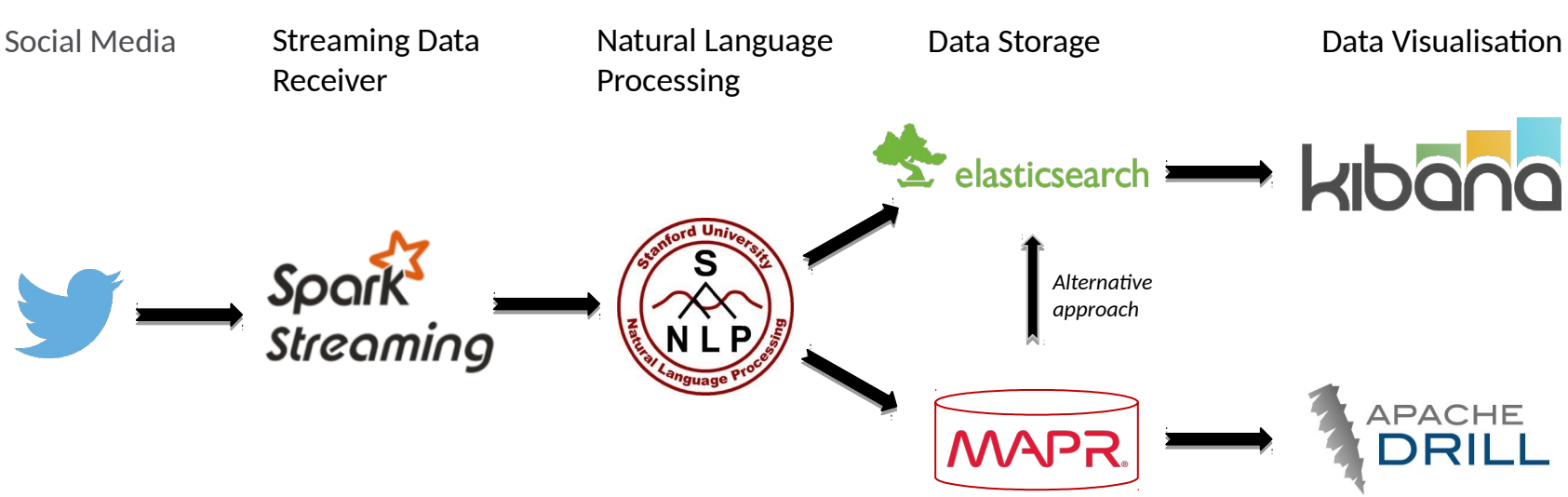
# Spark Streaming

- Same runtime programming model as batch.
- Streaming data is continuous.
- To process data using Spark Streaming it needs to be batched.
- Data stream is divided into batches of X milliseconds called DStreams.
- Each mini batch is represented as a Spark RDD.



# Twitter Sentiment Analysis

# Twitter Sentiment Analysis – Process Flow



# Twitter Sentiment Analysis – Sentiment Analysis Libraries

| | Stanford NLP | NLTK | RapidMiner | Mahout |
|---|---|---|---|---|
| **Language support** | Many<br>incl. Ducth, English, English, German, Chinese, Spanish etc. | Few<br>EN-NL-FR | One<br>EN | Many<br><custom development> |
| **Open-source** | Yes | Yes | Yes & No<br>Free & enterprise version | Yes |
| **Development effort** | Limited | Limited | Limited | Complex |
| **License costs** | Free | Free | Depending on use-case | Free |

# Twitter Sentiment Analysis – Stanford NLP


- Deep Learning - Recursive Neural Tensor Networks (RNTN) to do the sentiment classification.
- Pre-trained models on 11,855 sentences.
- Has labels for around 215,154 unique phrases.

# Twitter Sentiment Analysis – Stanford NLP

**Sentiment Analysis** | Information | Live Demo | **Sentiment Treebank** | Help the Model | Source Code

Source: http://nlp.stanford.edu/sentiment/treebank.html

Displaying **all 9,645 sentences**.

Content ... contains all of the following words

Sentence length ... is between – and – tokens

Sentiments are rated on a scale between 1 and 25, where 1 is the most negative and 25 is the most positive.

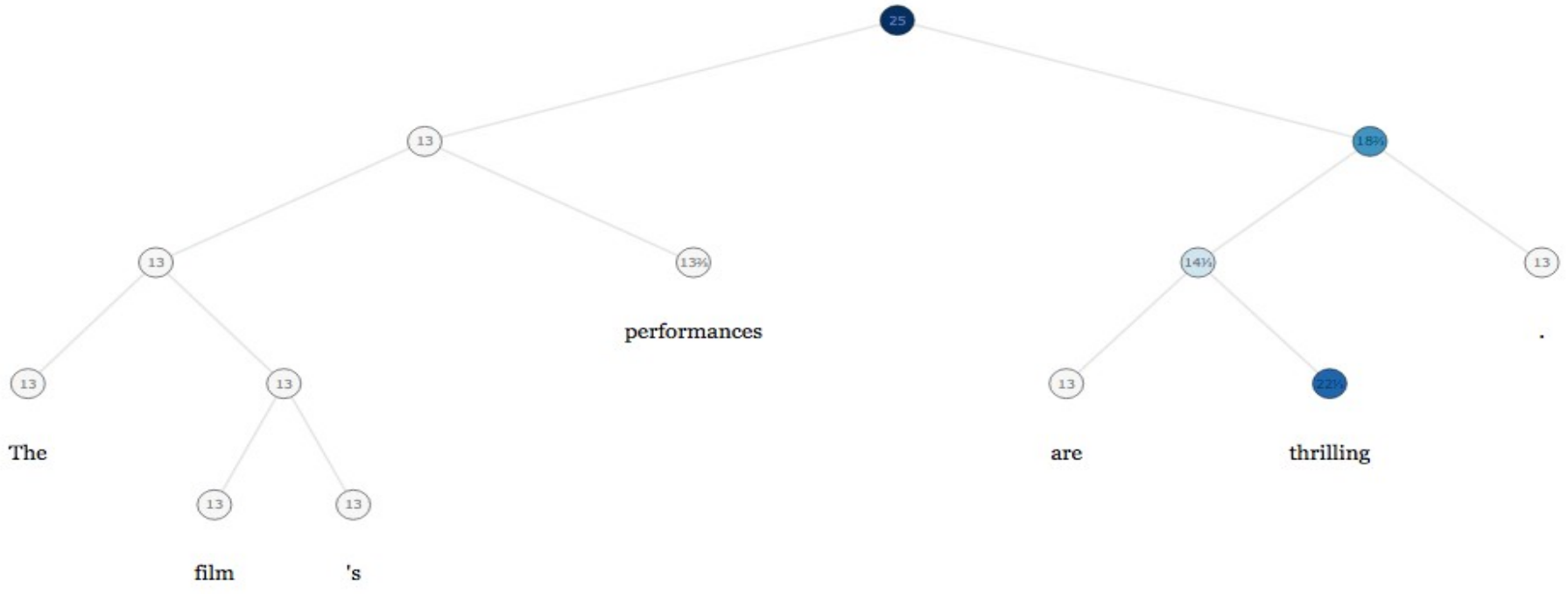
Extreme opinions ... include negative sentiments rated less than – or positive sentiments rated greater than –

Negations ... include a difference in sentiment rating beyond –

# Twitter Sentiment Analysis – Stanford NLP – Example 1

000199

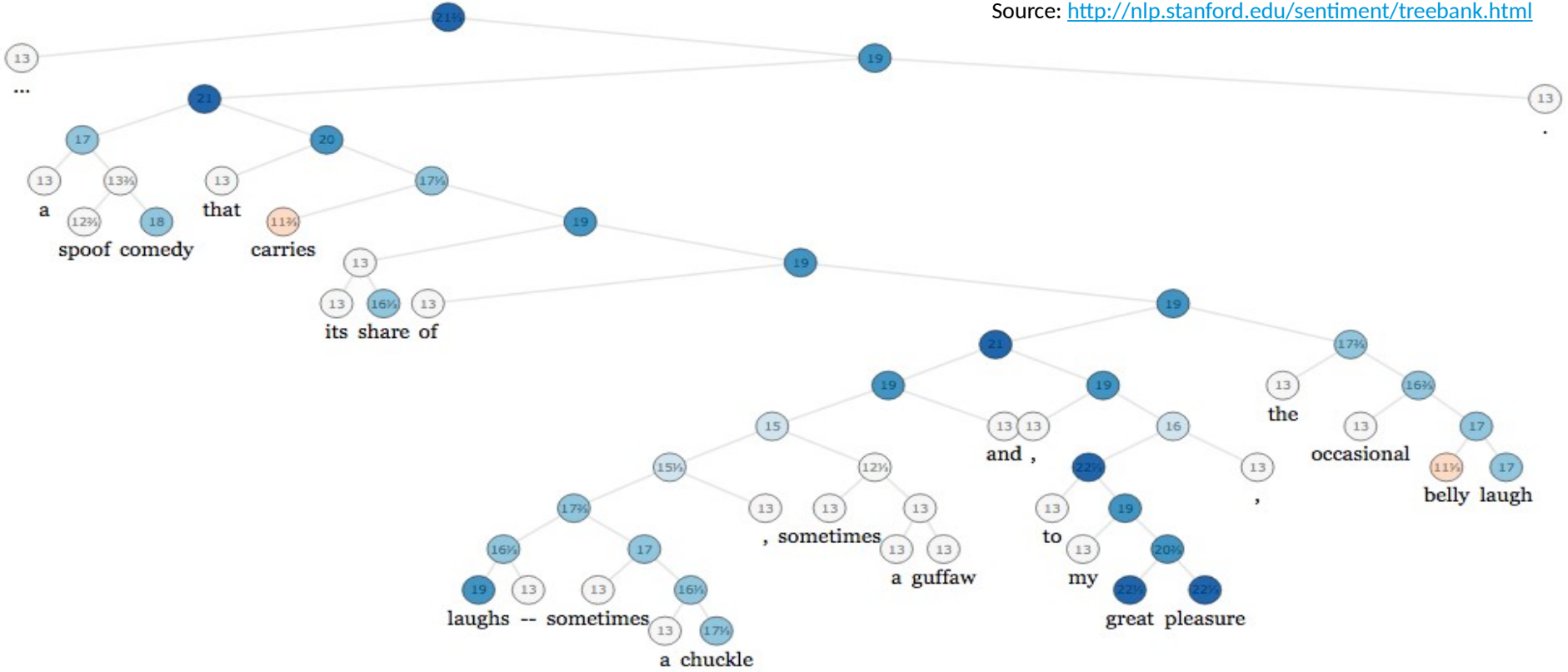
Source: http://nlp.stanford.edu/sentiment/treebank.html



# Twitter Sentiment Analysis – Stanford NLP – Example 2

000173

Source: http://nlp.stanford.edu/sentiment/treebank.html



# Twitter Sentiment Analysis – Code Review – Twitter Stream

```
// Import Twitter Spark Streaming library
import org.apache.spark.streaming.twitter._
```

```
// Set the system properties so that Twitter4j library used by twitter stream
// can use them to generate OAuth cedentials
System.setProperty("twitter4j.oauth.consumerKey", consumerKey)
System.setProperty("twitter4j.oauth.consumerSecret", consumerSecret)
System.setProperty("twitter4j.oauth.accessToken", accessToken)
System.setProperty("twitter4j.oauth.accessTokenSecret", accessTokenSecret)
```

```
val sparkConf = new SparkConf().setAppName("TwitterSentimentAnalysis")
val ssc = new StreamingContext(sparkConf, Seconds(1))
val tweets = TwitterUtils.createStream(ssc, None, filters)
```

Batch Interval

```
ssc.start()
ssc.awaitTermination()
```

# Twitter Sentiment Analysis – Code Review – Twitter Stream

```
// Import Twitter Spark Streaming library
import org.apache.spark.streaming.twitter._
```

```
// Set the system properties so that Twitter4j library used by twitter stream
// can use them to generate OAuth credentials
System.setProperty("twitter4j.oauth.consumerKey", consumerKey)
System.setProperty("twitter4j.oauth.consumerSecret", consumerSecret)
System.setProperty("twitter4j.oauth.accessToken", accessToken)
System.setProperty("twitter4j.oauth.accessTokenSecret", accessTokenSecret)
```

```
val sparkConf = new SparkConf().setAppName("TwitterSentimentAnalysis")
val ssc = new StreamingContext(sparkConf, Seconds(1))
val tweets = TwitterUtils.createStream(ssc, None, filters)
```

Batch Interval

```
ssc.start()
ssc.awaitTermination()
```

# Twitter Sentiment Analysis – Code Review – Write to MapR-DB

- ```
  // Write to MapR-DB
  tweets.foreachRDD {
        (rdd, time) => rdd.map(t => {

        var key = t.getUser.getScreenName + "-" + t.getCreatedAt.toInstant.toString
        var p = new Put(Bytes.toBytes(key))
  ```

- ```
  p.add(cfNameBytes, Bytes.toBytes("user"), Bytes.toBytes(t.getUser.getScreenName))
  p.add(cfNameBytes, Bytes.toBytes("created_at"), Bytes.toBytes(t.getCreatedAt.toInstant.toS
  p.add(cfNameBytes, Bytes.toBytes("location"), Bytes.toBytes(Option(t.getGeoLocation).map(ge      { s"$
  {geo.getLatitude},${geo.getLongitude}" }).toString))
  p.add(cfNameBytes, Bytes.toBytes("text"), Bytes.toBytes(t.getText))
  p.add(cfNameBytes, Bytes.toBytes("hashtags"), Bytes.toBytes(t.getHashtagEntities.ma  _.getText).toString))
  p.add(cfNameBytes, Bytes.toBytes("retweet"), Bytes.toBytes(t.getRetweetCount))
  p.add(cfNameBytes, Bytes.toBytes("language"), Bytes.toBytes(detectLanguage(t.getText)))
  p.add(cfNameBytes, Bytes.toBytes("sentiment"), Bytes.toBytes(detectSentiment(t.getText).toString))
  (new ImmutableBytesWritable, p)
  }).saveAsHadoopDataset(jobConfig)
  }
  ```

Enrichment

# Twitter Sentiment Analysis – Code Review – Write to Elasticsearch

```
tweets.foreachRDD{(rdd, time) =>
rdd.map(t => {
     Map(
          "user"-> t.getUser.getScreenName,
          "created_at" -> t.getCreatedAt.toInstant.toString,
          "location" -> Option(t.getGeoLocation).map(geo => { s"${geo.getLatitude},$
{geo.getLongitude}" }),
          "text" -> t.getText,
          "hashtags" -> t.getHashtagEntities.map(_.getText),
          "retweet" -> t.getRetweetCount,
          "language" -> detectLanguage(t.getText),
          "sentiment" -> detectSentiment(t.getText).toString )
}).saveToEs("twitter/tweet") }
```

# Learn more:

explore.mapr.com/data-fabric/