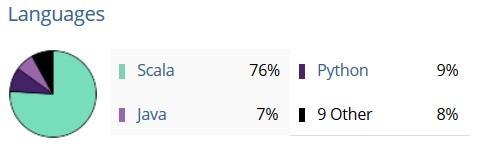
APACHE SPARK

* Spark is a computing engine similar to Map Reduce and TEZ.
* But spark has some unique advantages which makes it 100 times faster than MR while using its memory and 10 times faster while using its hard disk.

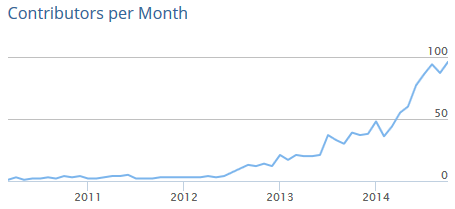
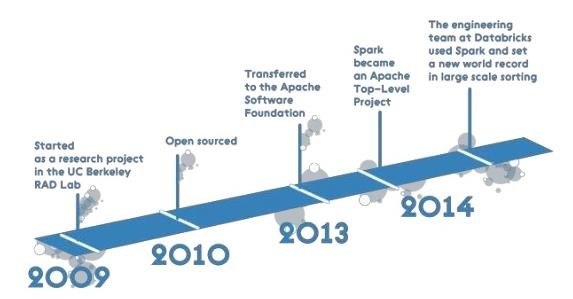
##### Introduction to Apache Spark

* **Apache Spark™ is a fast and general unified in-memory engine** for large-scale data processing framework for massive distributed & parallel computing (cluster) **using the power of cheap memory.**
* Harnessing power of cheap memory
* Written using Scala, Java and Python languages
* High-level APIs support in Scala, Java and Python
* Has had [36,338 commits](https://www.openhub.net/p/apache-spark/commits/summary) made by [1,319 contributors](https://www.openhub.net/p/apache-spark/contributors/summary)

representing [lakhs of lines of code](https://www.openhub.net/p/apache-spark/analyses/latest/languages_summary) with proper comments

* Developers from 50+ companies includes UC Berkley, Cloudera

, Yahoo, Databricks, Intel, Groupon etc.,



##### Brief History

##### The key difference between Hadoop MapReduce and Spark

To make the comparison fair, here we will contrast Spark with Hadoop MapReduce, as both are responsible for data processing.

In fact, the key difference between them lies in the approach to processing:

* Spark can do it in-memory, while Hadoop MapReduce has to read from and write to a disk.
* As a result, the speed of processing differs significantly
* Spark may be up to 100 times faster. However, the volume of data processed also differs: Hadoop

MapReduce is able to work with far larger data sets than Spark.

##### Tasks Hadoop MapReduce is good for:

Linear processing of huge data sets

* Hadoop MapReduce allows parallel processing of huge amounts of data.
* It breaks a large chunk into smaller ones to be processed separately on different data nodes and automatically gathers the results across the multiple nodes to return a single result. In case the resulting dataset is larger than available RAM, Hadoop MapReduce may outperform Spark.

Economical solution, if no immediate results are expected.

* Hadoop team considers MapReduce a good solution if the speed of processing is not critical. For instance, if data processing can be done during night hours (batch process), it makes sense to consider using Hadoop MapReduce.

##### Tasks Spark is good for:

Fast data processing.

In-memory processing makes Spark faster than Hadoop MapReduce – up to 100 times for data in RAM and up to 10 times for data in storage.

Iterative processing.

If the task is to process data again and again – Spark defeats Hadoop MapReduce.

Spark’s Resilient Distributed Datasets (RDDs) enable multiple map operations in memory, while Hadoop MapReduce has to write interim results to a disk.

Near real-time processing.

If a business needs immediate insights, then they should opt for Spark and its in-memory processing.

Graph processing.

Spark’s computational model is good for iterative computations that are typical in graph processing. And Apache Spark has GraphX – an API for graph computation.

Machine learning.

Spark has MLlib – a built-in machine learning library, while Hadoop needs a third-party to provide it.

MLlib has out-of-the-box algorithms that also run in memory. But if required, our Spark specialists will tune and adjust them to tailor to your needs.

Joining datasets.

Due to its speed, Spark can create all combinations faster, though Hadoop may be better if joining of very large data sets that requires a lot of shuffling and sorting is needed.

##### Examples of practical applications

We analyzed several examples of practical applications and made a conclusion that Spark is likely to outperform MapReduce in all applications below, thanks to fast or even near real-time processing. Let’s look at the examples.

Customer segmentation. Analyzing customer behavior and identifying segments of customers that demonstrate similar behavior patterns will help businesses to understand customer preferences and create a unique customer experience.

Risk management. Forecasting different possible scenarios can help managers to make right decisions by choosing non-risky options.

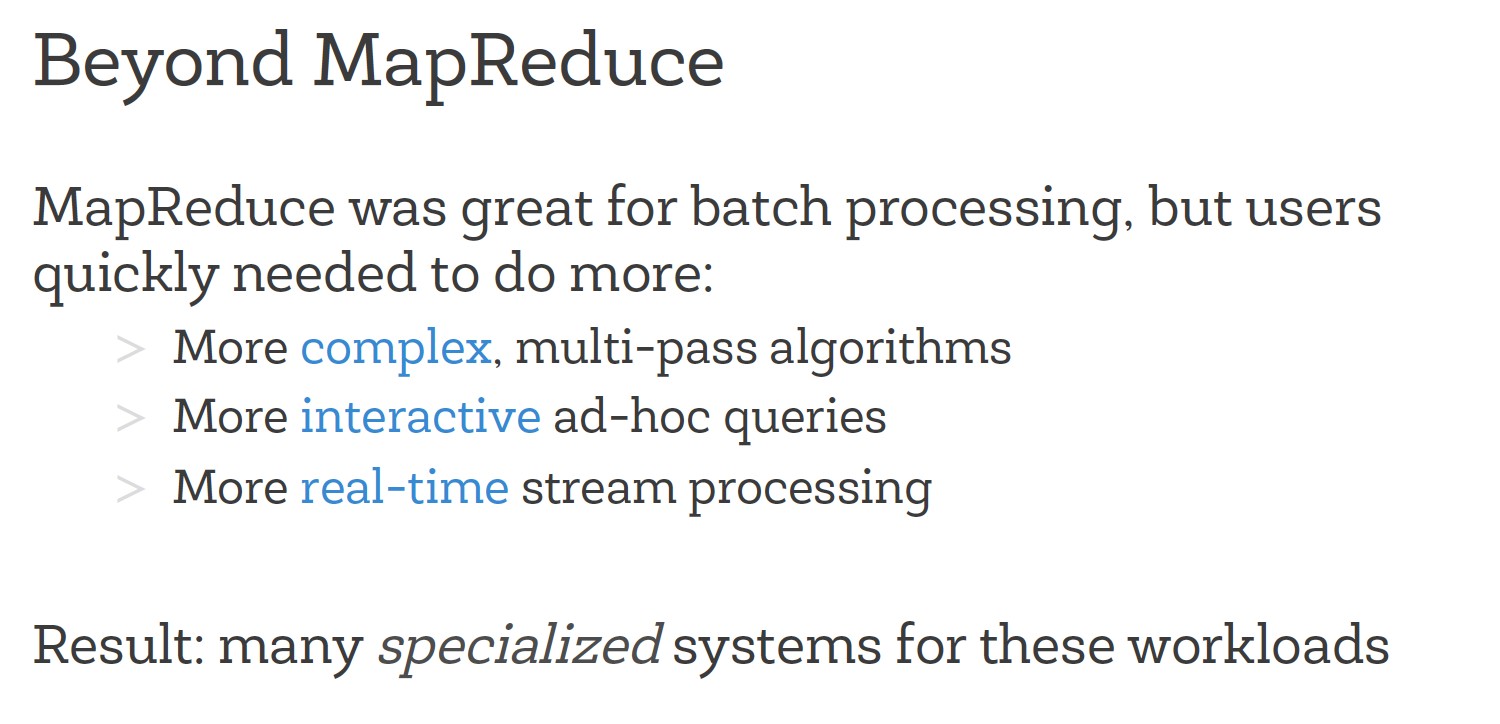
Real-time fraud detection. After the system is trained on historical data with the help of machine-learning algorithms, it can use these findings to identify or predict an anomaly in real time that may signal of a possible fraud.

Industrial big data analysis. It’s also about detecting and predicting anomalies, but in this case, these anomalies are related to machinery breakdowns. A properly configured system collects the data from sensors to detect pre-failure conditions.

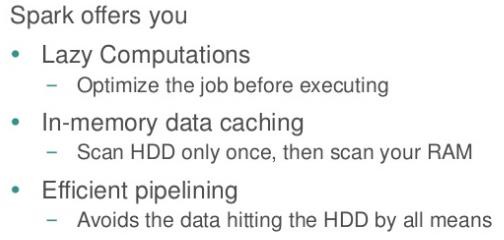
Which framework to choose?

It’s your particular business needs that should determine the choice of a framework. Linear processing of huge datasets is the advantage of Hadoop MapReduce, while Spark delivers fast performance, iterative processing, real-time analytics, graph processing, machine learning and more. In many cases Spark may outperform Hadoop MapReduce. The great news is the Spark is fully compatible with the Hadoop eco-system and works smoothly with Hadoop Distributed FileSystem, Apache Hive, etc.

**Spark Execution Model**

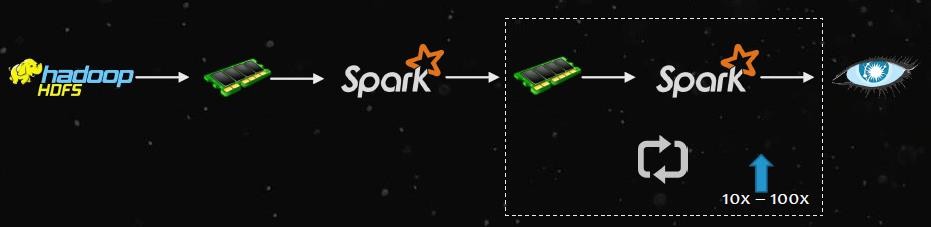


##### Beyond MapReduce





Batch, Huge I/O



Realtime, In-Memory

**Lazy computation:**

A = load data “xyz”

B = a.foreach.filter by some scenario

C = C.println

Here when A = load data “xyz” is executed in the program, its not exactly executed. Its just defined. Similarly B also is just defined. Once C is executed, it’s the actual execution.

After C is executed then only it requests data from B. B requests data from A. then A gets data from the source file. This is called Lazy computation.

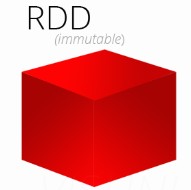
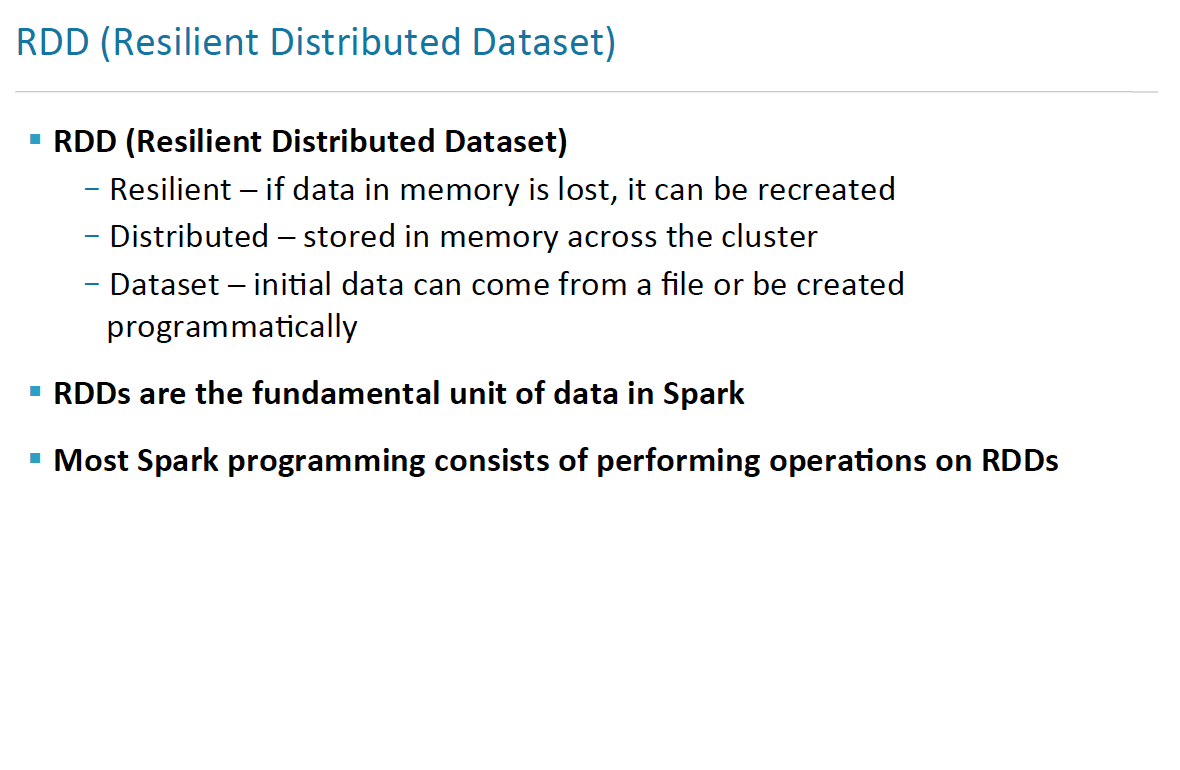
**In memory data caching:**

When the flow of execution happens from A to C, the intermediate results (result of A and B) are stored in-memory. Where as in MR its stored in local hard disk.

**Effective pipelining:**

Since the results of intermediate process are stored in memory, pipelining is efficient.

##### Pillars of Spark



##### All about RDDs

* Core Spark abstraction
* **RDD** is the core abstraction in [Apache Spark](https://data-flair.training/blogs/apache-spark-for-beginners/). It is an immutable, [fault-tolerant](https://data-flair.training/blogs/fault-tolerance-in-apache-spark/)  
  distributed collection of statically typed objects that are usually stored [in-memory](https://data-flair.training/blogs/apache-spark-in-memory-computing/).
* RDD API offers simple operations such as map, reduce, and filter that can be

composed in arbitrary ways.

* Lazily evaluated
* **lazy evaluation** in Spark means that the execution will not start until an

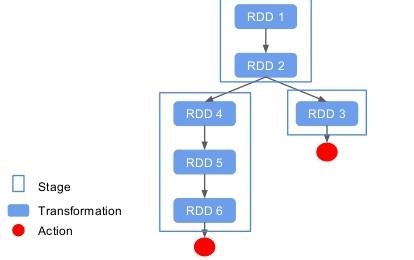
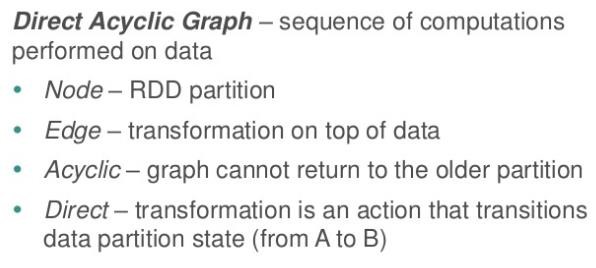
action is triggered.

* In Spark, the picture of lazy evaluation comes when Spark transformations occur.
* Partitions can be persisted in-memory or on-disk spread

across multiple nodes

* Immutable
* Once RDD is created, it can’t be modified. But can be reassigned to another RDD.
* Fault tolerant – Via concept of Lineage
* A -> B -> C -> D
* In the above flow, if the result of B which was stored in memory was lost, it can be recreated from A. its called Lineage.+++++++++-

##### Pillars of Spark



##### Creating RDDs

Three ways to create an RDD

* + From a file or set of files & other sources
  + From data in memory
  + From another RDD

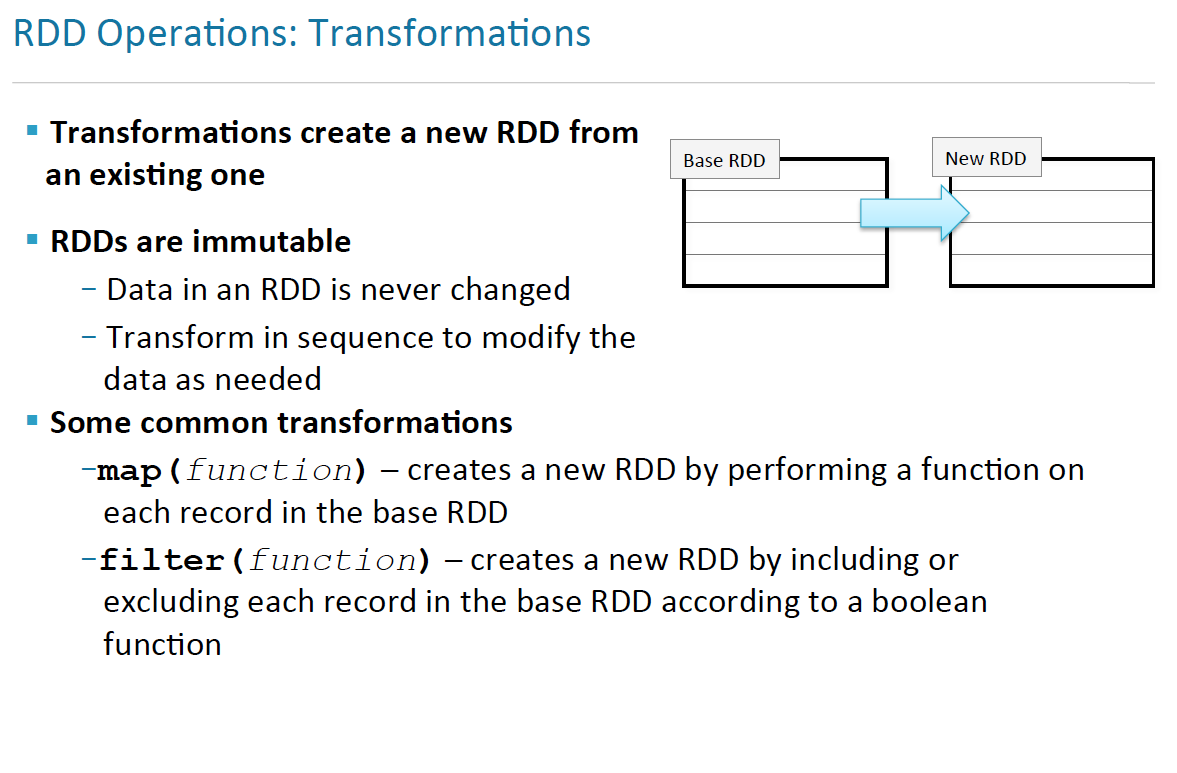
##### Operations in RDD:

##### https://cdn-images-1.medium.com/max/1600/1*1En5bV-PbHNmkv9buFMxlg.pngWhat are the RDD Operations

* Transformations
* Actions

##### Transformation:

* Transformations are kind of operations which will transform your RDD data from one form to another.
* And when you apply this operation on any RDD, you will get a new RDD with transformed data (RDDs in Spark are immutable).
* Operations like **map, filter, flatMap**are transformations.
* Now there is a point to be noted here and that is, when you apply the transformation on any RDD it will not perform the operation immediately.
* It will create a DAG(Directed Acyclic Graph) using the applied operation, source RDD and function used for transformation.
* And it will keep on building this graph using the references till you apply any action operation on the last lined up RDD.
* That is why the transformation in Spark are lazy.



##### There are two types in transformation:

* **Narrow transformation**

In Narrow transformation, all the elements that are required to compute the

records in single partition live in the single partition of parent RDD.

A limited subset of partition is used to calculate the result.

Narrow transformations are the result of map(), filter().

(one to one mapping concept)

* **Wide transformation**

In wide transformation, all the elements that are required to compute

the records in the single partition may live in many partitions of parent RDD.

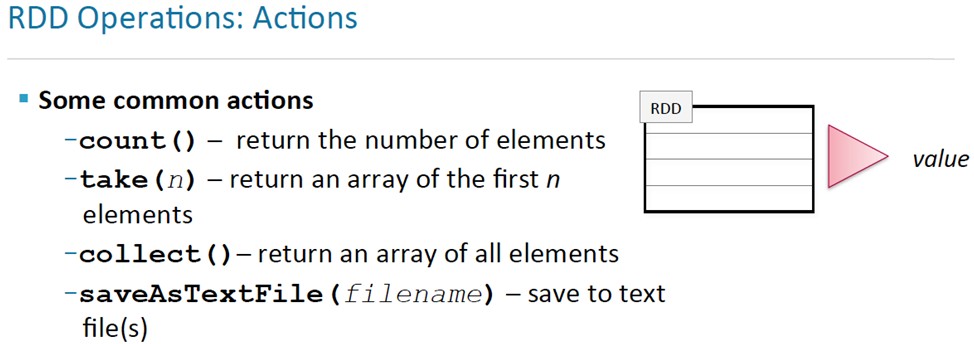
The partition may live in many partitions of parent RDD.

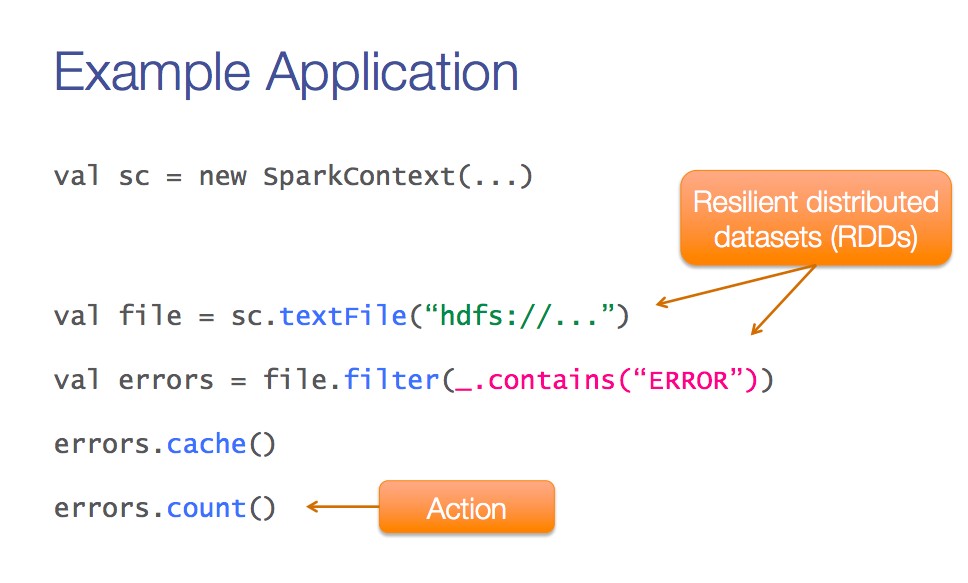
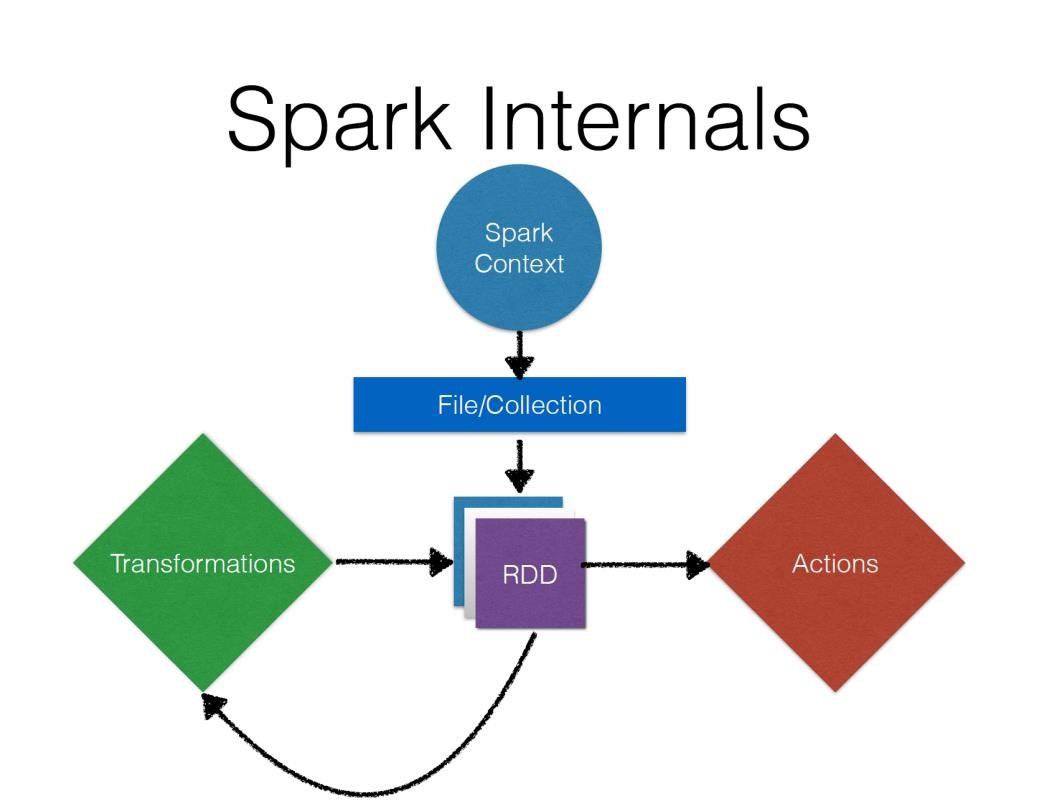
Wide transformations are the result of groupbyKey and reducebyKey.

(many to one concept)

##### Actions

* Transformations create RDDs from each other, but when we want to work with the actual dataset, at that point action is performed.
* When the action is triggered after the result, new RDD is not formed like transformation.
* The values of action are stored to drivers or to the external storage system.
* It brings laziness of RDD into motion.





**A Sample Application**

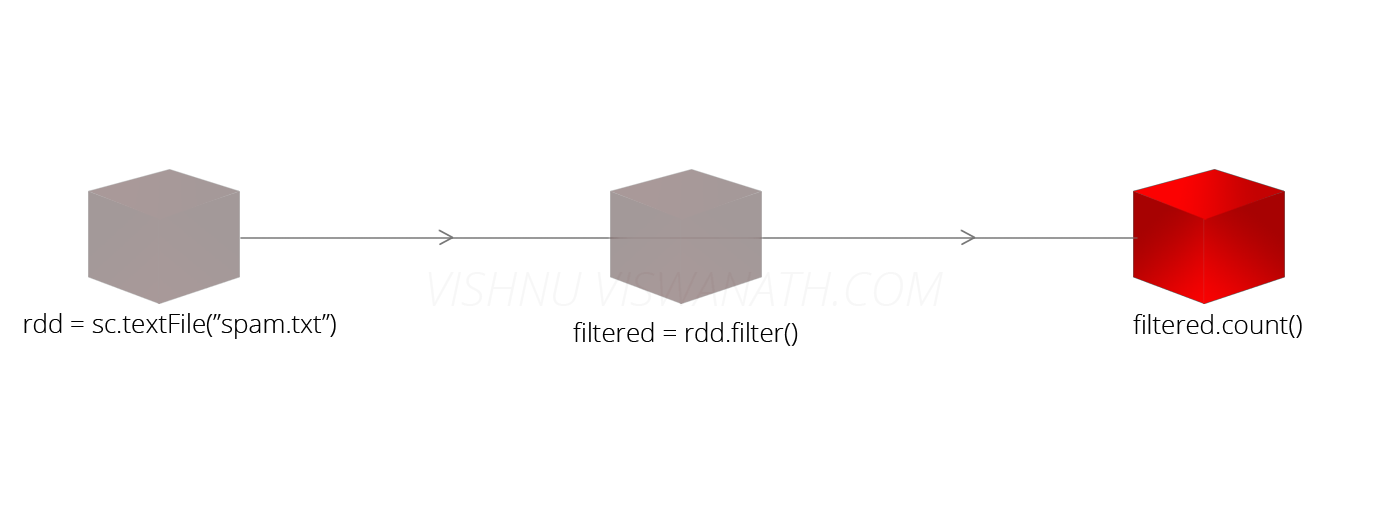
Spark context

Transformation

Transformation

**RDD Lineage**

* + - When you call a transformation, Spark does not execute it immediately, instead it creates a **lineage**.
    - A lineage in the form of DAG stored in Driver, keeps track of what all transformations has to be applied on that RDD.



Lineage Lineage

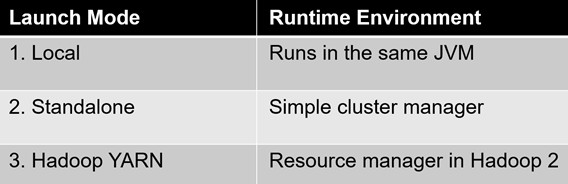
**val rdd = sc.textFile("spam.txt")**

**val filtered = rdd.filter(line => line.contains("money")) filtered.count()**

INCEPTEZ TECHNOLOGIES

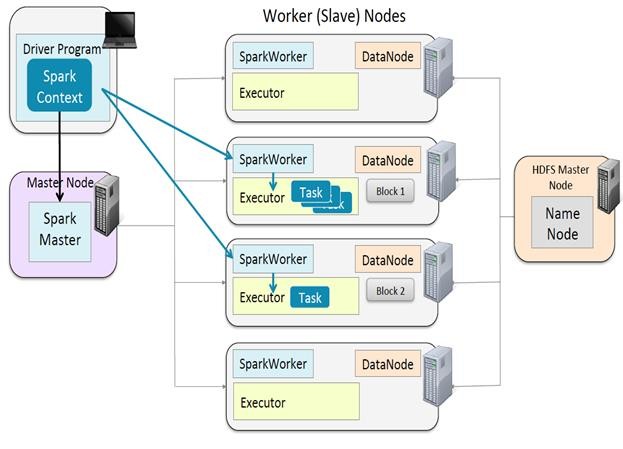
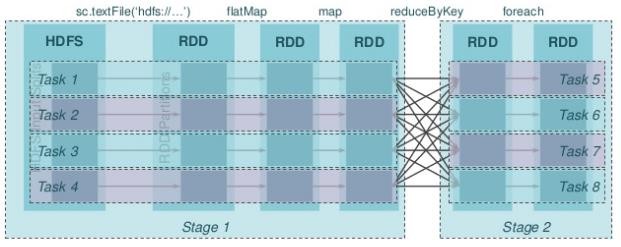
### Terminologies

* **Driver Program**
  + The process to start the execution (main() function)
  + Place where Spark Context is created.
* **Spark Context **
  + Constructor created that tells how to access the cluster.
* **Cluster Manager**
  + An external service to manage resources on the cluster

(standalone manager, YARN, Apache Mesos, Local)

* **Deploy Mode**
  + **cluster** : Driver inside the cluster
  + **client** : Driver outside of Cluster

### Terminologies (Contd.)



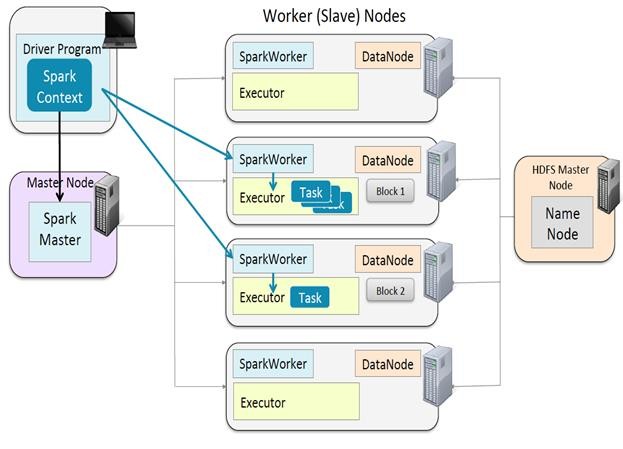
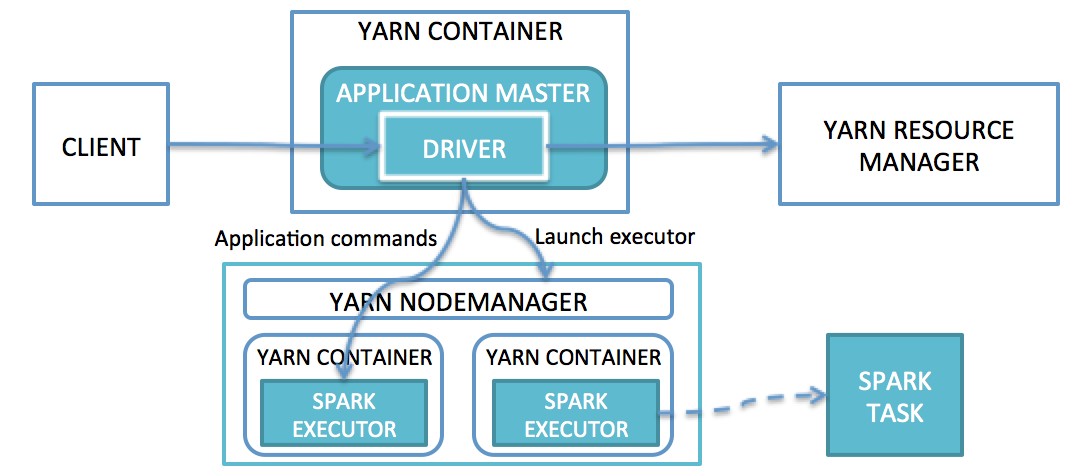
* **Worker Node :** Node that run the application program in cluster
* **Executor**
  + Process launched on a worker node, that runs the Tasks
  + Keep data in memory or disk reading from external sources
* **Job**
  + Consists multiple tasks, Created based on a Action
* **Stage :** Each Job is divided into a smaller set of tasks

called Stages that is sequential and depend on each other

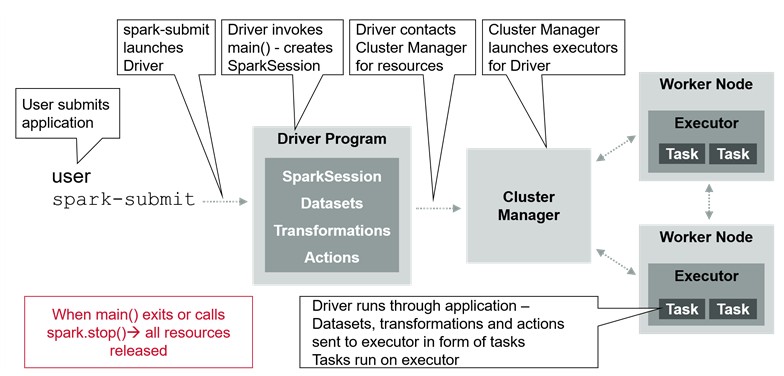
* **Task :** A unit of work that will be sent to executor.
* **Partitions:** Data unit that will be handled parallel, Same as Blocks in HDFS.

### Spark Cluster Deployment

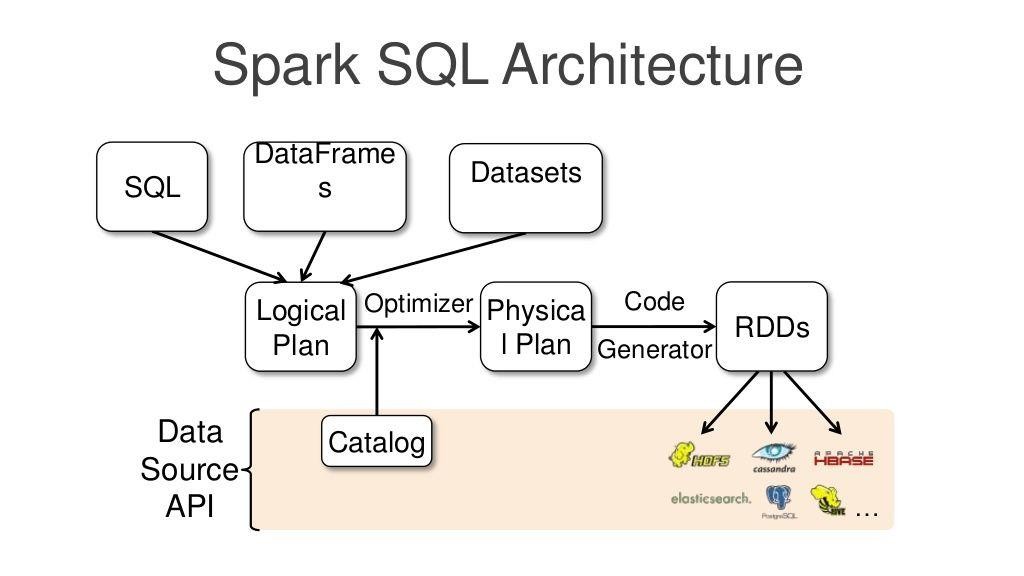
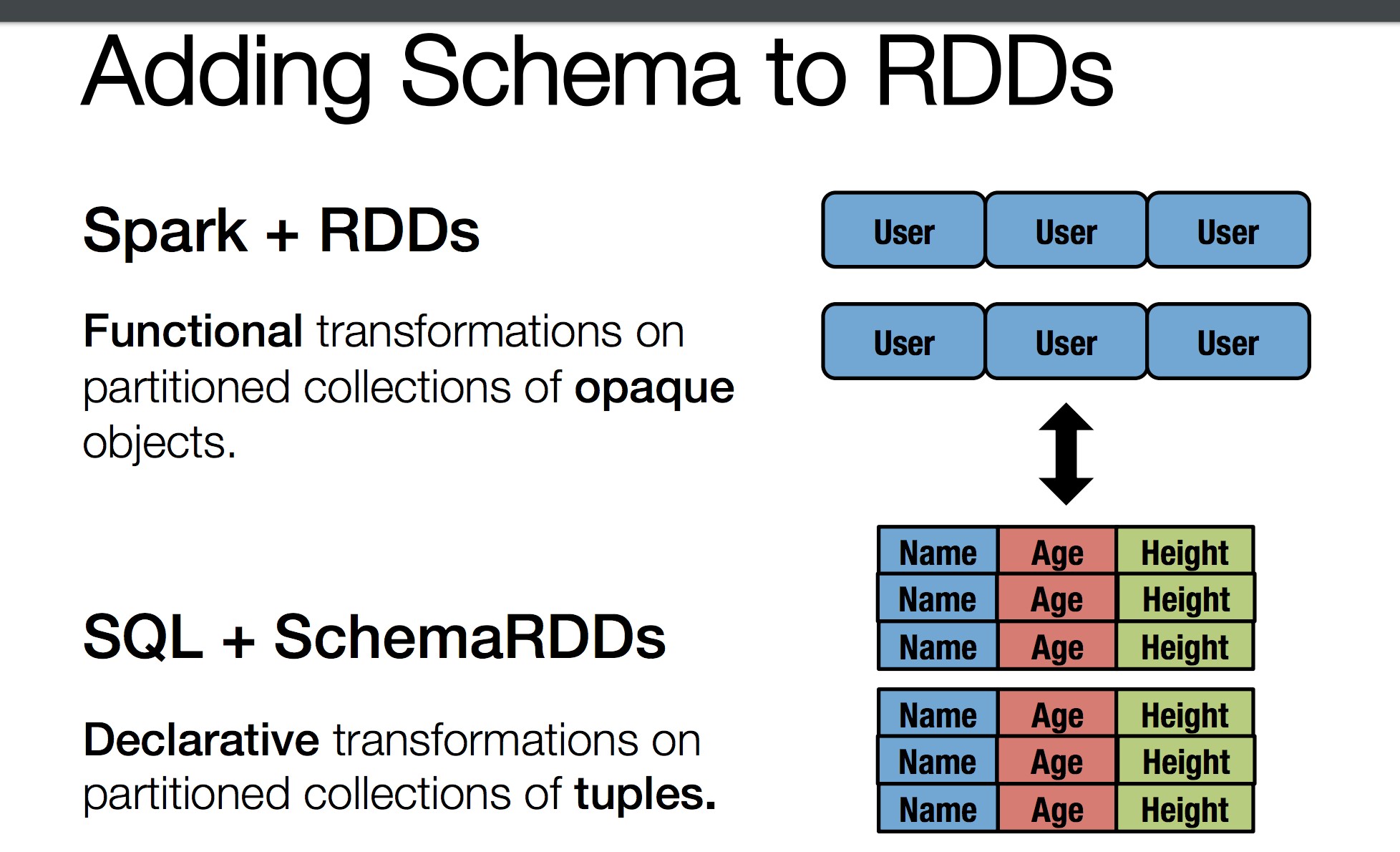
###### Standalone Spark Spark onYARN



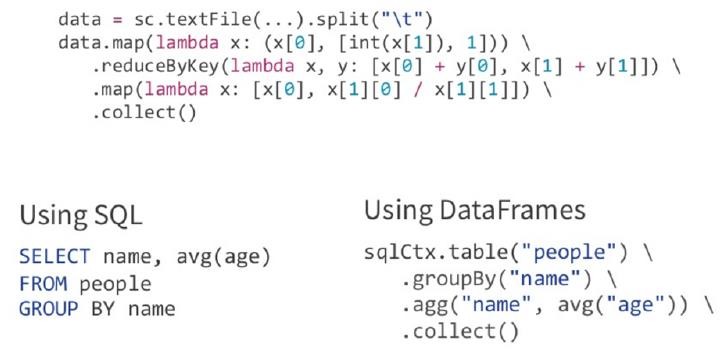
**Lifecycle of a Spark Application**



Spark SQL



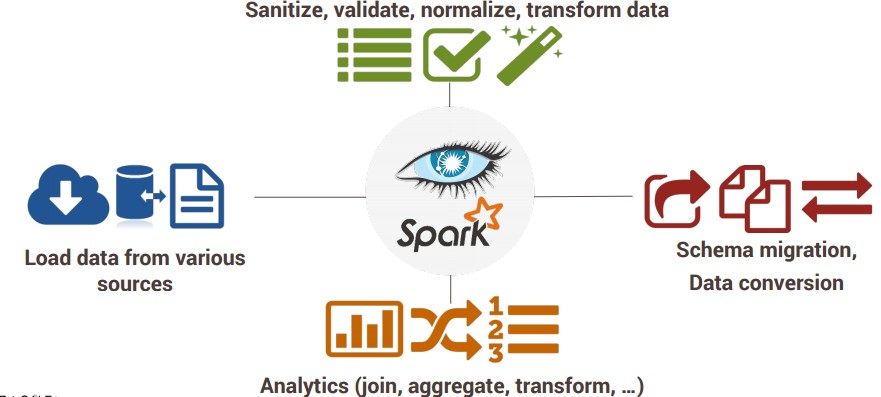
**Why Spark SQL**



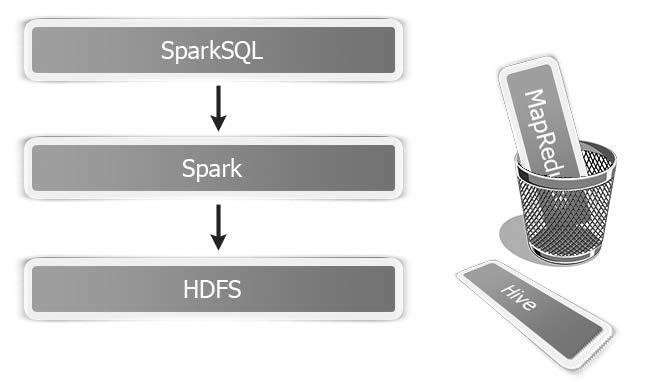
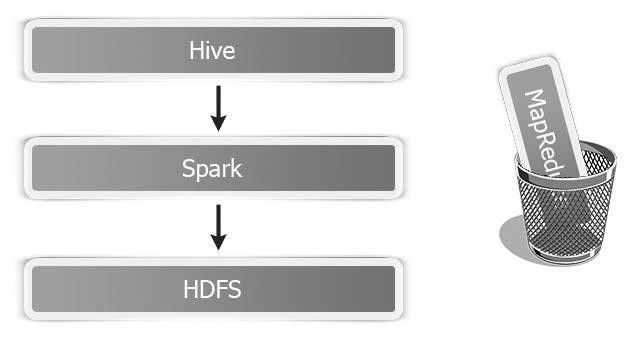
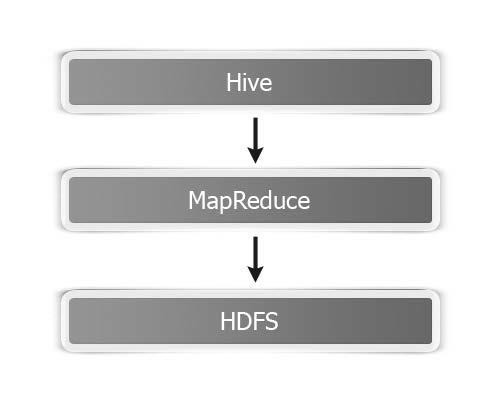
* **Coding Complexity is less**
* **Read less data**
* **Let optimizer do the hard work**
* **Integrated**
  + Seamlessly mix SQL queries with Spark programs.
  + Connect to any data source the same way.
* **Hive Compatibility**
  + Run unmodified Hive queries on existing data.

INCEPTEZ TECHNOLOGIES

**Applications of Spark SQL**



INCEPTEZ TECHNOLOGIES



###### Evolution of Spark SQL

Traditional Shark Spark SQL

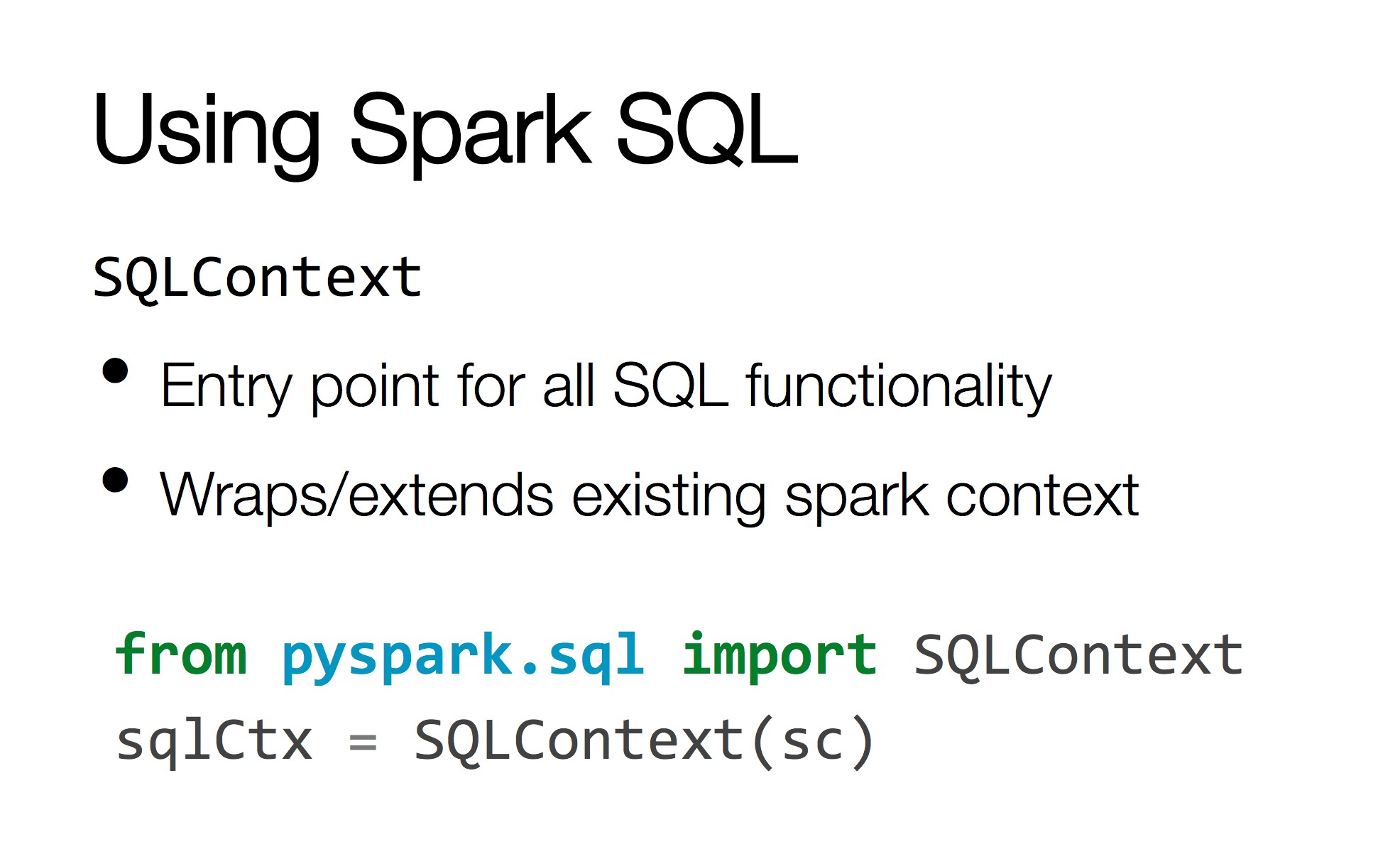
INCEPTEZ TECHNOLOGIES

## Using Spark SQL

import org.apache.spark.sql.SQLContext import spark.sql

val sqlc=new SQLContext(sc)

sqlc.sql("CREATE TABLE IF NOT EXISTS src (NAME string, AGE INT) ")



**Spark Session:**

* **New entry point for Dataset (strongly typed collection) and Dataframes (Row array of Any types which**

**looses its datatype).**

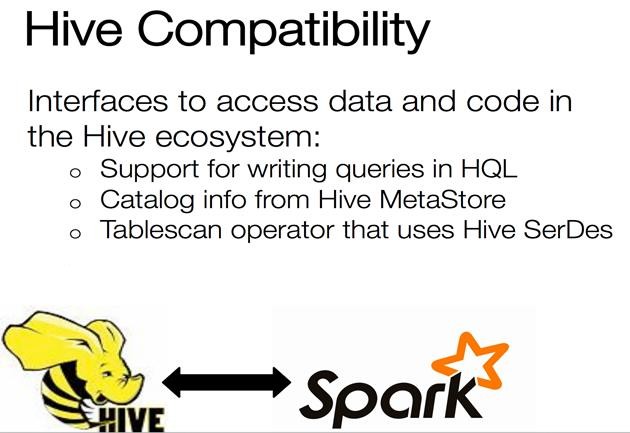
* **SparkSession is essentially combination of SQLContext, HiveContext and future StreamingContext.**

**import org.apache.spark.sql.SparkSession**

**val spark = SparkSession.builder().getOrCreate()**

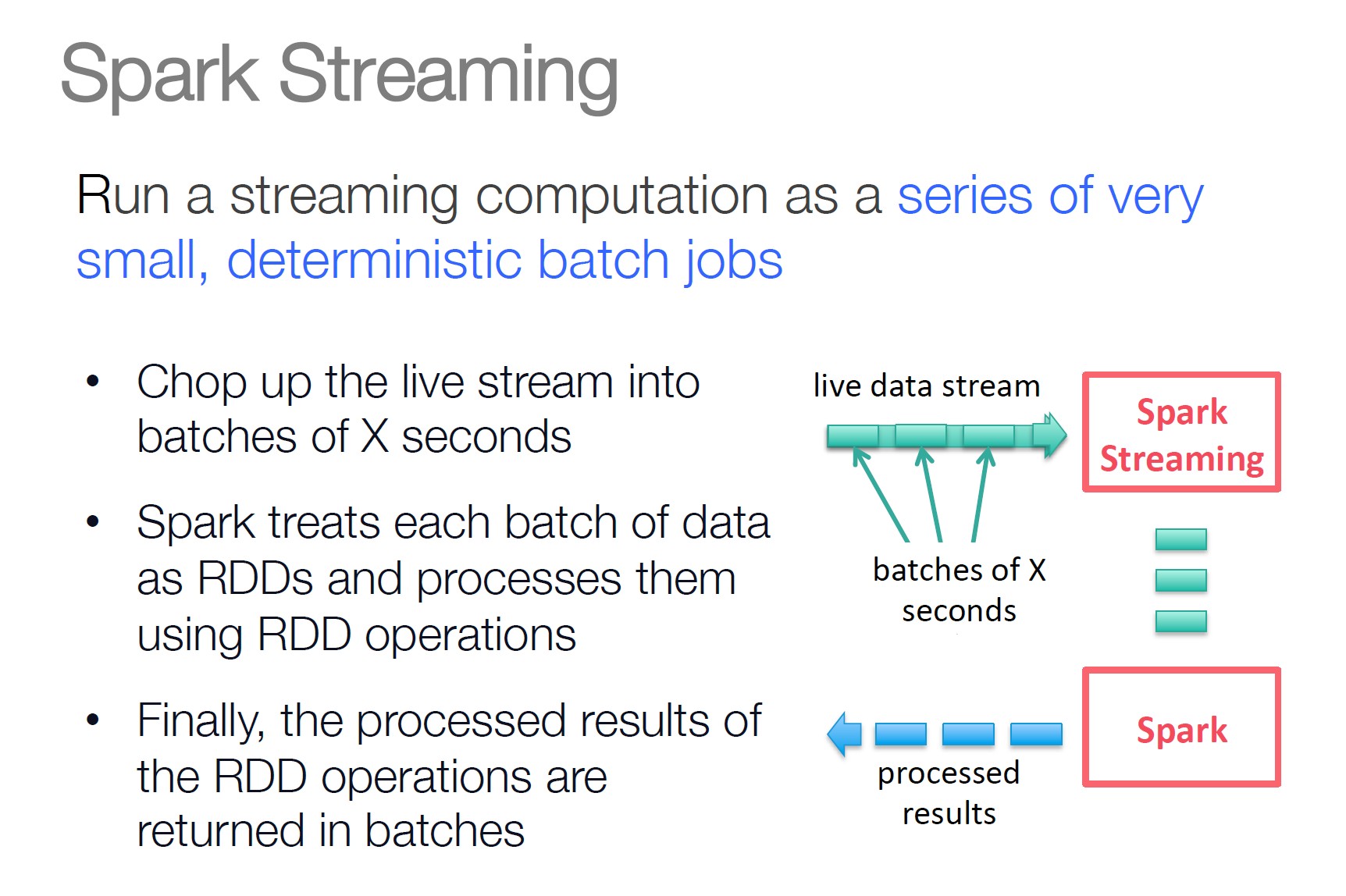
**val tbl = spark("CREATE TABLE IF NOT EXISTS src (NAME string, AGE INT) ")**

###### Hive Compatability



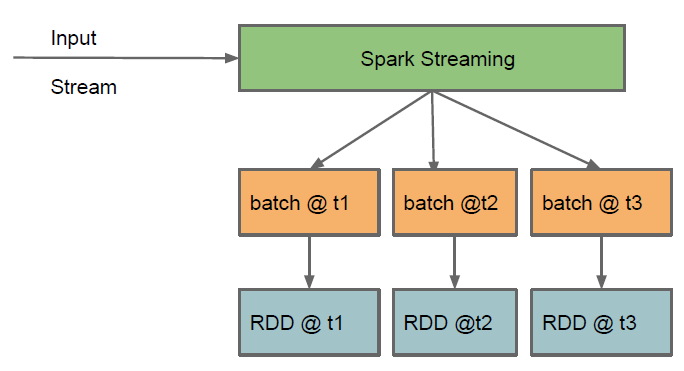
INCEPTEZ TECHNOLOGIES

# SPARK STREAMING

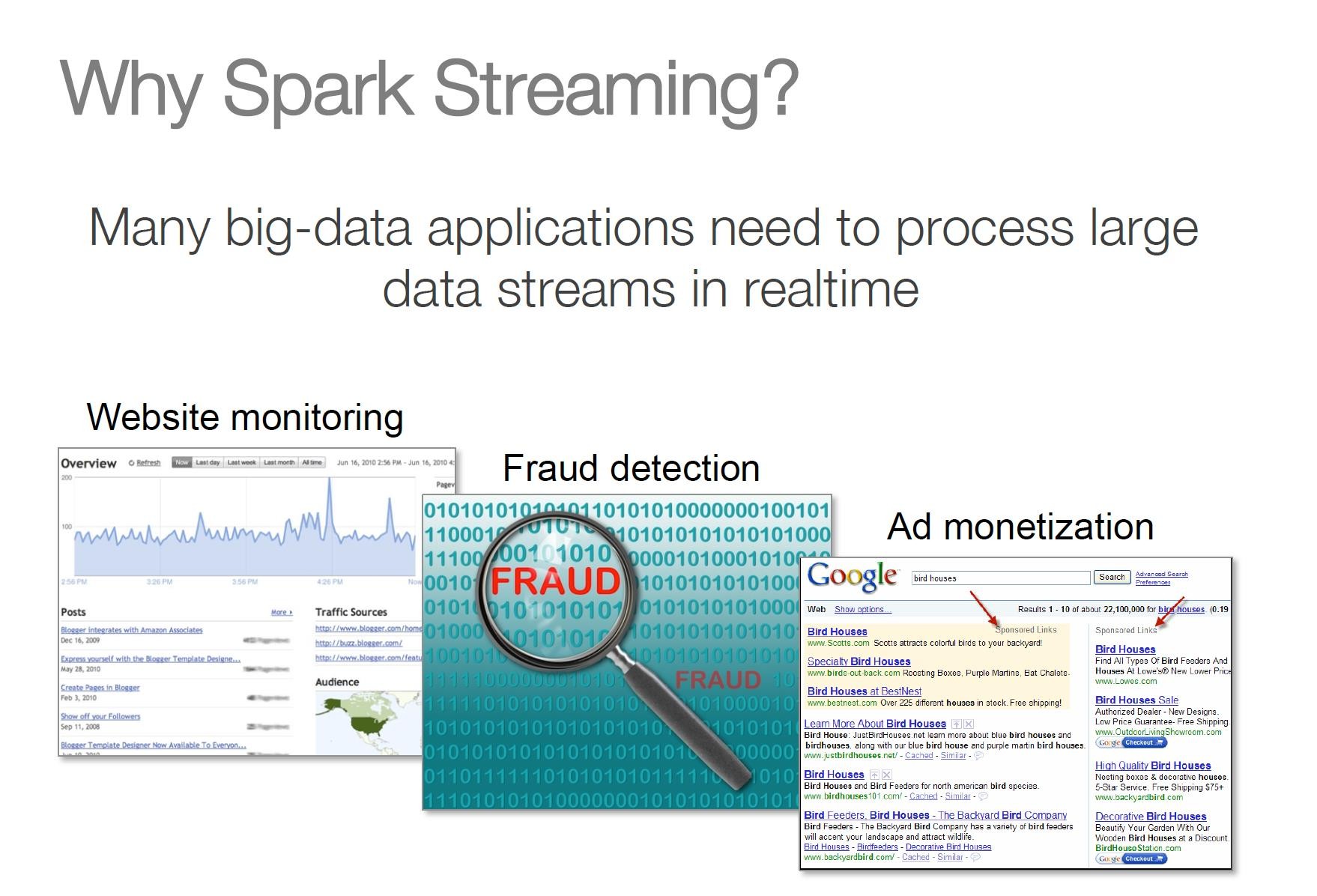


###### Spark Streaming

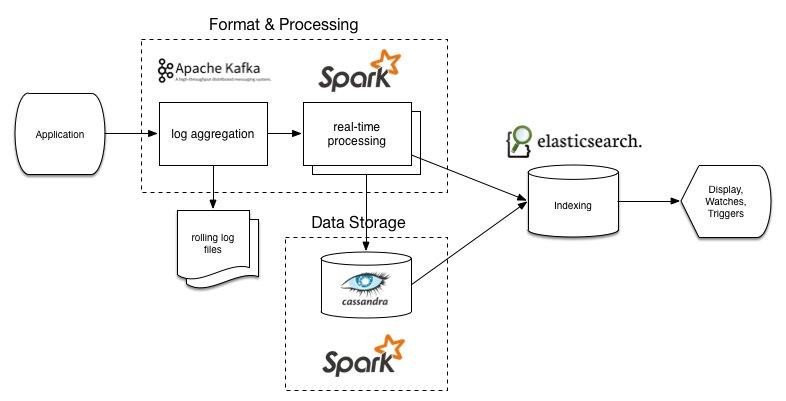
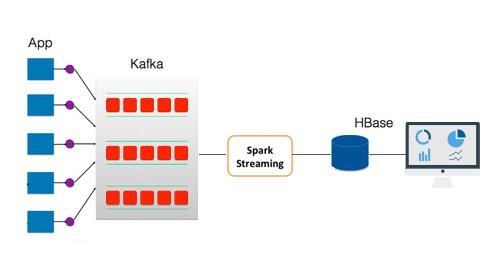
INCEPTEZ TECHNOLOGIES

Key concepts

* + **DStream** – sequence of RDDs representing a stream of data
    - Twitter, HDFS, Kafka, Flume, ZeroMQ, TCP sockets
  + **Transformations** – modify data from one DStream to another RDD
    - Standard RDD operations – map, countByValue, reduceByKey, join, …
  + **Output Operations –** send data to external entity
    - saveAsHadoopFiles – saves to HDFS
    - foreach – do anything with each batch of results



#### Realtime Streaming Usecases



WORKOUTS