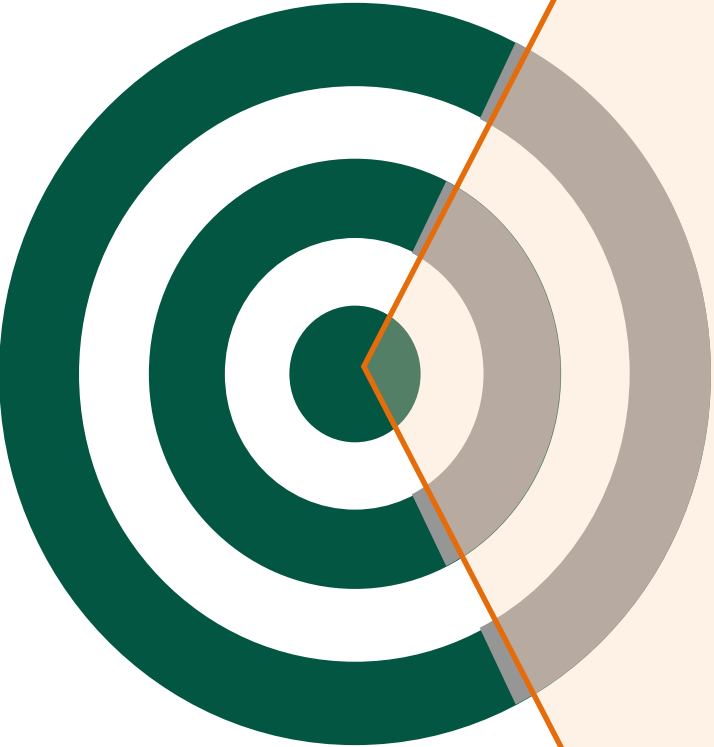
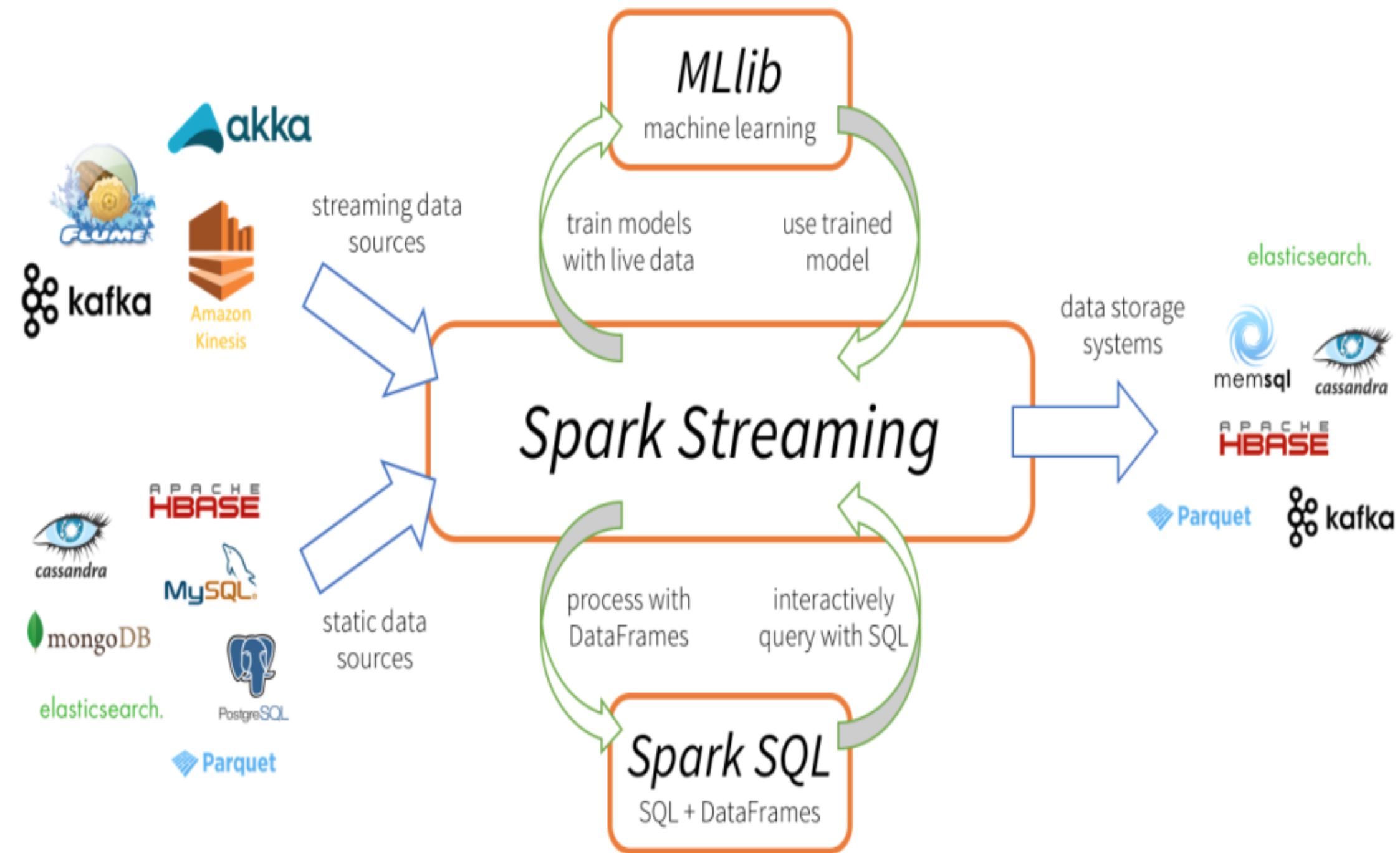




## In this session, you will learn about:

- 
- Directed Acyclic Graph (DAG)
  - Data Lineage
  - Lazy Evaluation
  - Spark Installation
  - Data Ingestion in Spark
  - Spark Mllib
  - Implementing the Machine learning Algorithms with Scala
  - Spark Streaming
  - Spark SQL

# Spark Components



# Directed Acyclic Graph (DAG)

Directed Acyclic Graph (DAG) is a tool that depicts the structure of basic blocks, helps to see the flow of values flowing among the basic blocks, and offers optimization too. DAG provides easy transformation on basic blocks. DAG can be understood as:

- Leaf nodes represent identifiers, names or constants.
- Interior nodes represent operators.
- Interior nodes also represent the results of expressions or the identifiers/name where the values are to be stored or assigned.

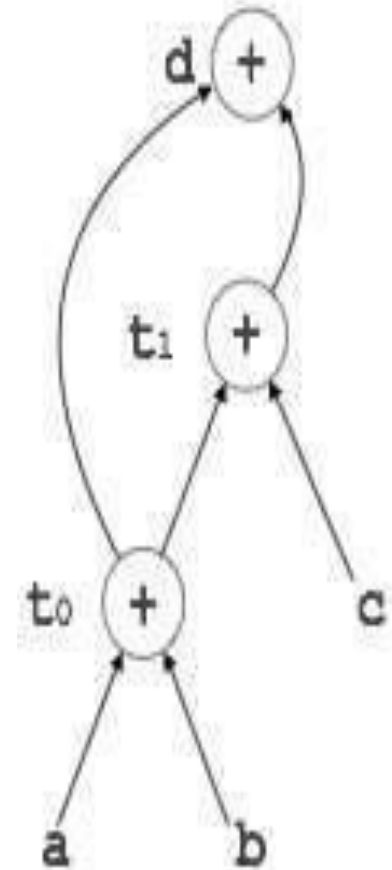
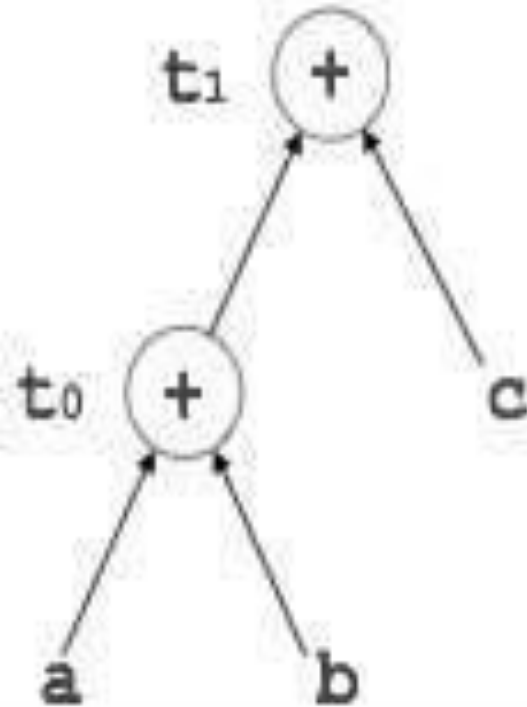
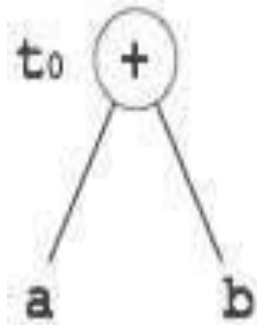
## Example:

$$t_0 = a + b$$

$$t_1 = t_0 + c$$

$$d = t_0 + t_1$$

# Directed Acyclic Graph (DAG)



$$[t_0 = a + b]$$

$$[t_1 = t_0 + c]$$

$$[d = t_0 + t_1]$$

# **Data Lineage**



# Data Lineage – Definition

## Definition

Data lineage is generally defined as a kind of data life cycle that includes the data's origins and where it moves over time.

This term can also describe what happens to data as it goes through diverse processes.

Data lineage can help with efforts to analyze how information is used and to track key bits of information that serve a particular purpose.





# Data Lineage

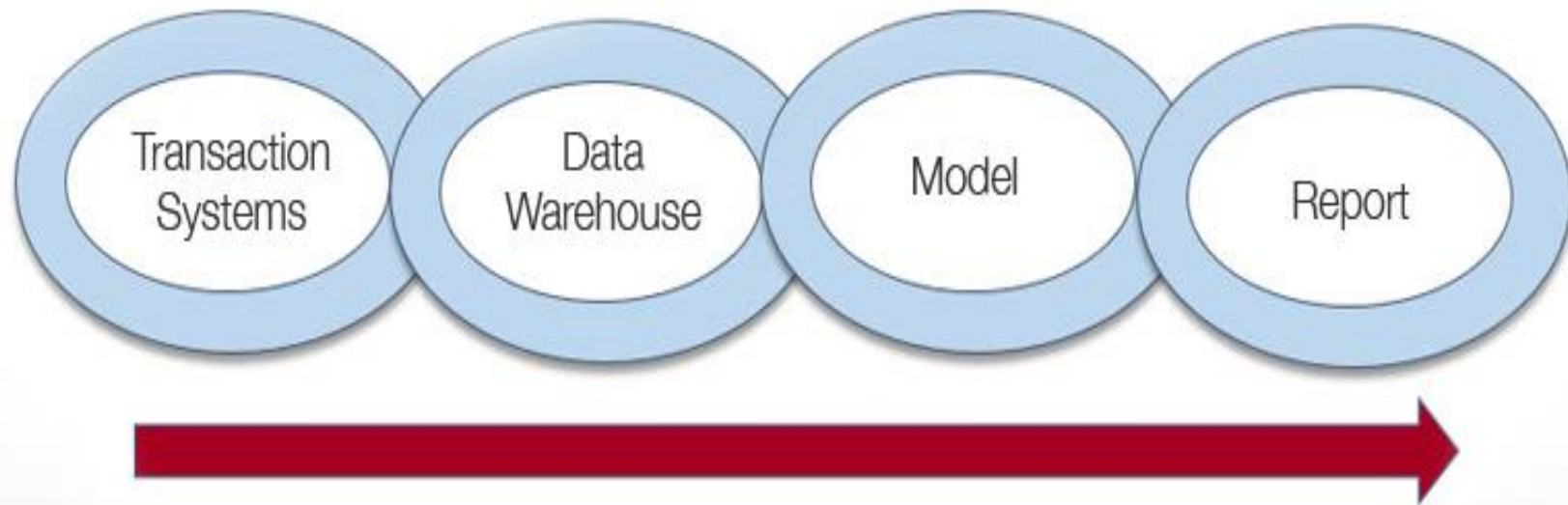
- It provides a visual representation to discover the data flow/movement from its source to destination via various changes and hops on its way in the enterprise environment.
- Data Lineage represents: How the data hops between various data points, how the data gets transformed along the way, how the representation and parameters change, and how the data splits or converges after each hop.
- Easier representation of the Data Lineage can be shown with dots and lines, where dot represents a data container for data point(s) and lines connecting them represents the transformation(s) the data point under goes, between the data containers.



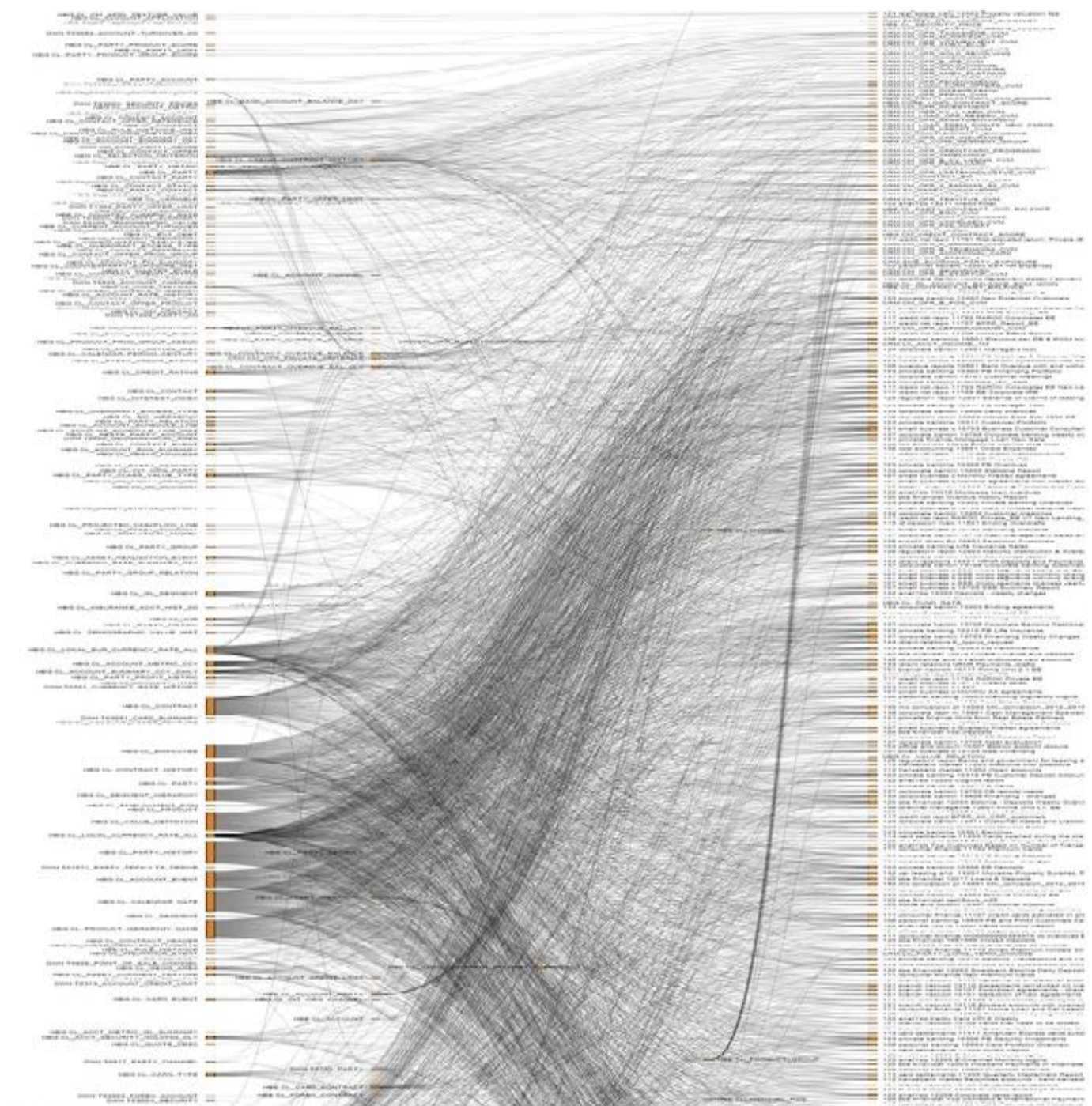


# Data Lineage

Data Lineage



# Data Lineage



# Common Application of Data Lineage

- One common application of data lineage methodologies is in the field of business intelligence, which involves gathering data and building conclusions from that data.
- Data lineage helps to show, for example, how sales information has been collected and what role it could play in new or improved processes that put the data through additional flow charts within a business or organization.
- All of this is part of a more effective use of the information that businesses or other parties have obtained.



# Common Application of Data Lineage



Another use of data lineage, as pointed out by business experts, is in safeguarding data and reducing risk.



By collecting large amounts of data, businesses and organizations are exposing themselves to certain legal or business liabilities.



These relate to any possible security breach and exposure of sensitive data.



Using data lineage techniques can help data managers handle data better and avoid some of the liability associated with not knowing where data is at a given stage in a process.

# **Spark Installation**



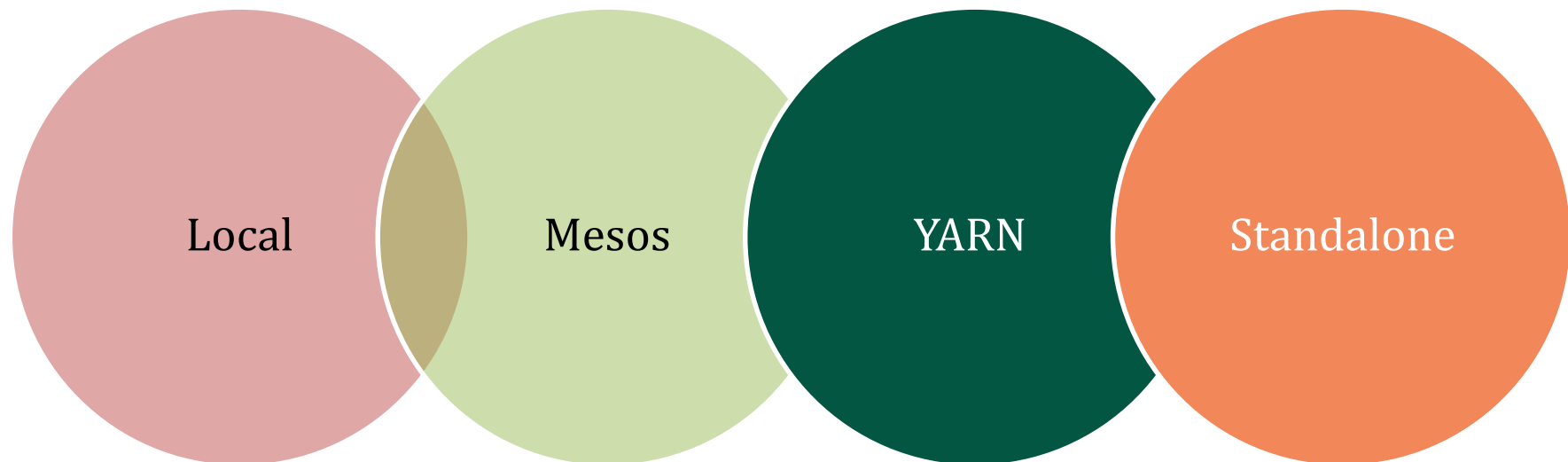


## Installation

# Spark Installation



Cluster Managers:





# Local Mode



- For application development purposes, no cluster required
- Local
  - Run Spark locally with one worker thread (i.e. no parallelism at all).
  - `local[K]`
    - Run Spark locally with K worker threads (ideally, set this to the number of cores on your machine).
- `local[*]`
  - Run Spark locally with as many worker threads as logical cores on your machine.

# Standalone Mode



- Place compiled version of spark at each node
- Deployment scripts
  - `sbin/start-master.sh`
  - `sbin/start-slaves.sh`
  - `sbin/stop-all.sh`
- Various settings, e.g. port, webUI port, memory, cores, java opts
- Drivers use `spark://HOST:PORT` as master
- Only supports a simple FIFO scheduler
  - Application or global config decides how many cores and memory will be assigned to it.
- Resilient to Worker failures, Master single point of failure
- Supports Zookeeper for multiple Masters, leader election and state recovery. Running applications unaffected
- Or local filesystem recovery mode just restarts Master if it goes down. Single node. Better with external monitor

# YARN Mode



- Yet another resource negotiator
- Decouples resource management and scheduler from data processing framework
- Exclusive to Hadoop ecosystem
- Binary distribution of spark built with YARN support
- Uses hadoop configuration HADOOP\_CONF\_DIR or YARN\_CONF\_DIR
- Master is set to either yarn-client or yarn-cluster

# Mesos Mode



Mesos is a cluster operating system



Abstracts CPU, memory, storage and other resources enabling fault tolerant and elastic distributed system



Can run Spark along with other applications (Hadoop, Kafka, ElasticSearch, Jenkins, etc.) and manage resources and scheduling across the whole cluster and all the applications



Mesos master replaces Spark Master as Cluster Manager



Spark binary accessible by Mesos (config)

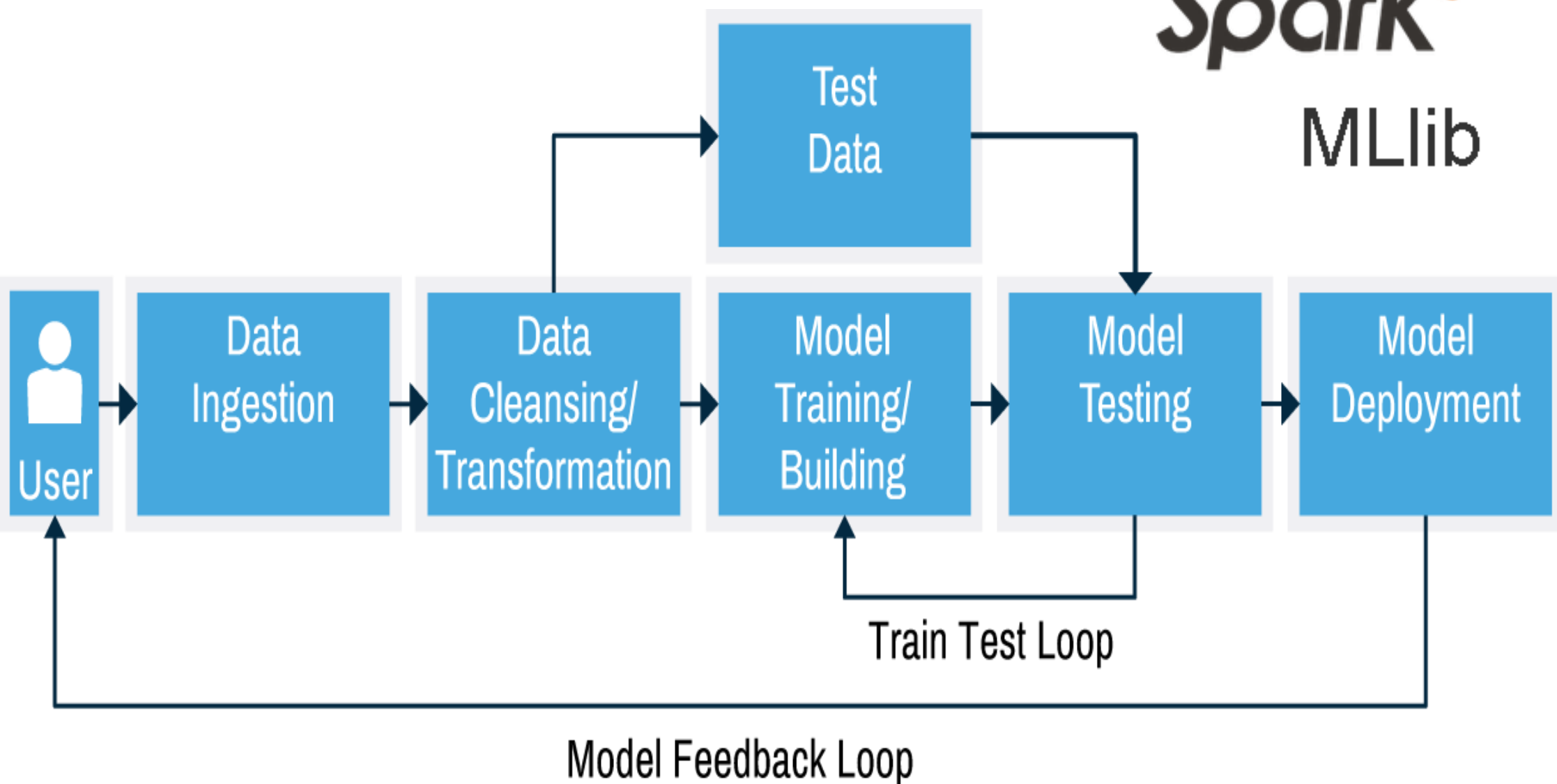
# Mesos Mode



- Mesos://HOST:PORT for single master mesos or mesos://zk://HOST:PORT for multi master mesos using Zookeeper for failover.
- In “fine-grained” mode (default), each Spark task runs as a separate Mesos task.
- This allows multiple instances of Spark (and other frameworks) to share machines at a very fine granularity.
- The “coarse-grained” mode will instead launch only *one* long-running Spark task on each Mesos machine, and dynamically schedule its own “mini-tasks” within it.

# **Spark MLlib**







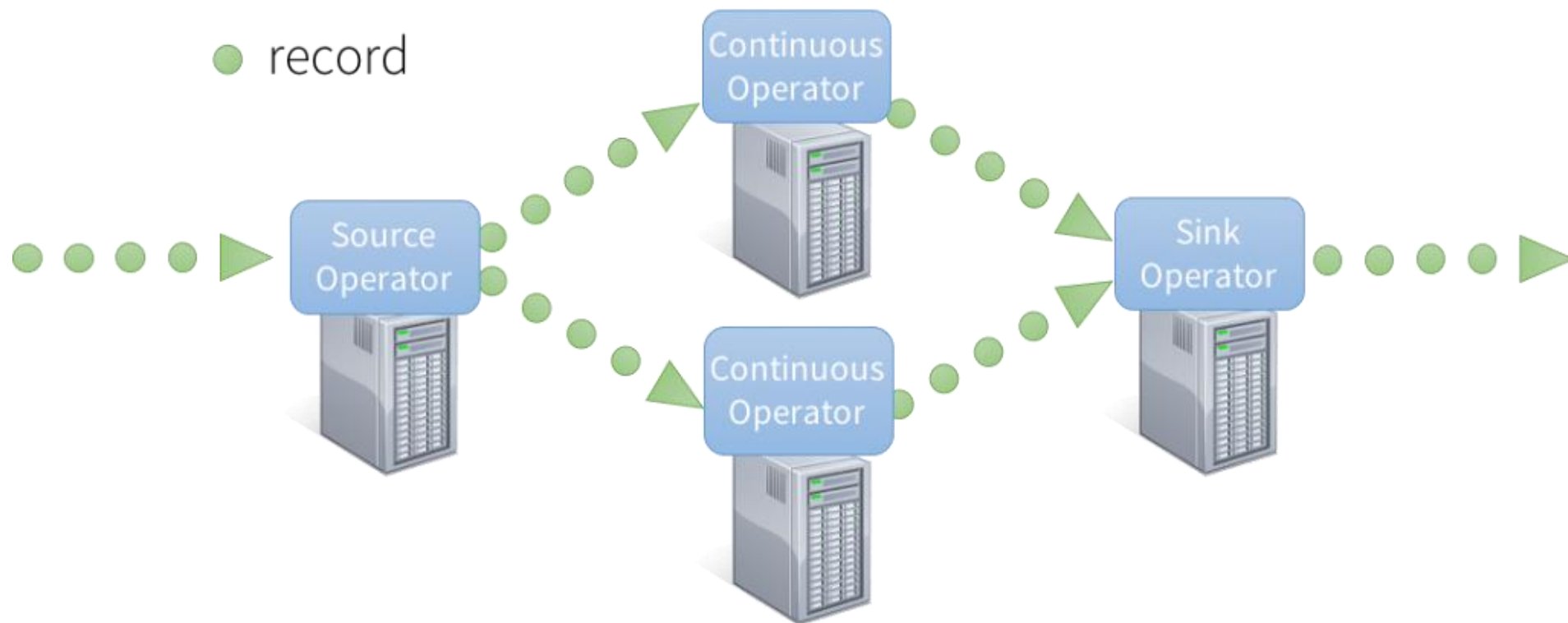
	Discrete	Continuous
Supervised	<b>Classification</b> <ul style="list-style-type: none"><li>• Logistic regression (and regularized variants)</li><li>• Linear SVM</li><li>• Naive Bayes</li><li>• Random Decision Forests (soon)</li></ul>	<b>Regression</b> <ul style="list-style-type: none"><li>• Linear regression (and regularized variants)</li></ul>
Unsupervised	<b>Clustering</b> <ul style="list-style-type: none"><li>• K-means</li></ul>	<b>Dimensionality reduction, matrix factorization</b> <ul style="list-style-type: none"><li>• Principal component analysis / singular value decomposition</li><li>• Alternating least squares</li></ul>

# **Implementing the Machine Learning Algorithms With Scala**



# Spark Streaming

## Traditional stream processing systems *continuous operator model*

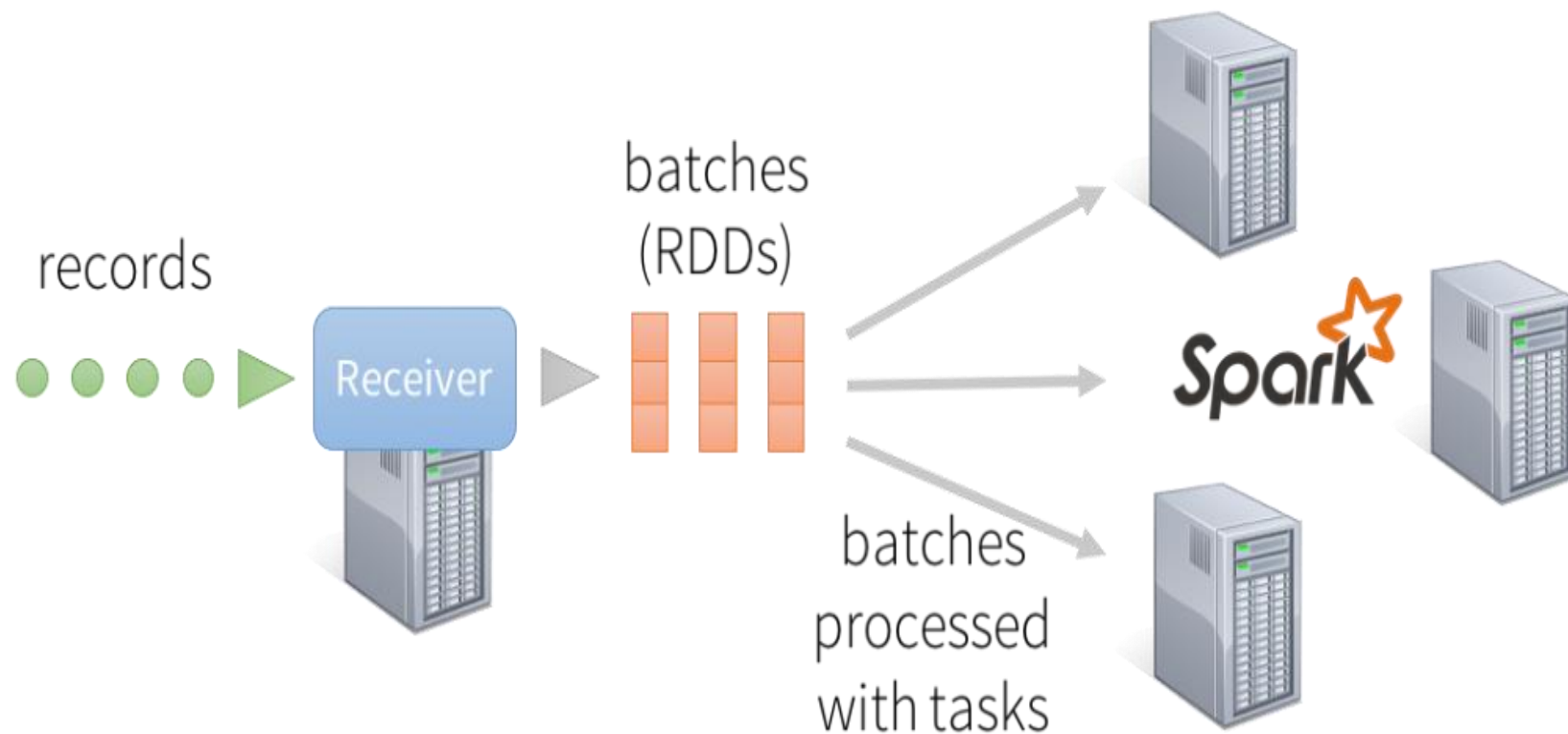


records processed one at a time

# Spark Streaming

## Spark<sup>★</sup> Streaming

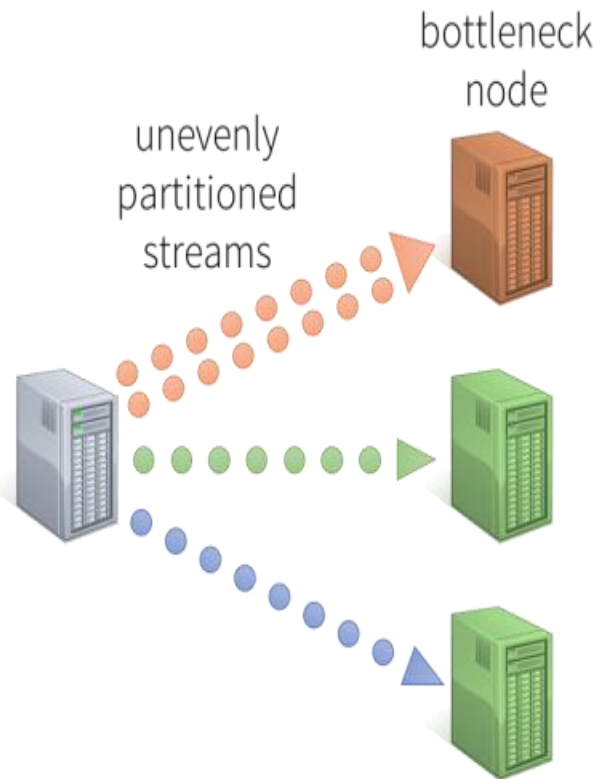
*discretized stream processing*



records processed in batches with short tasks  
each batch is a RDD (partitioned dataset)

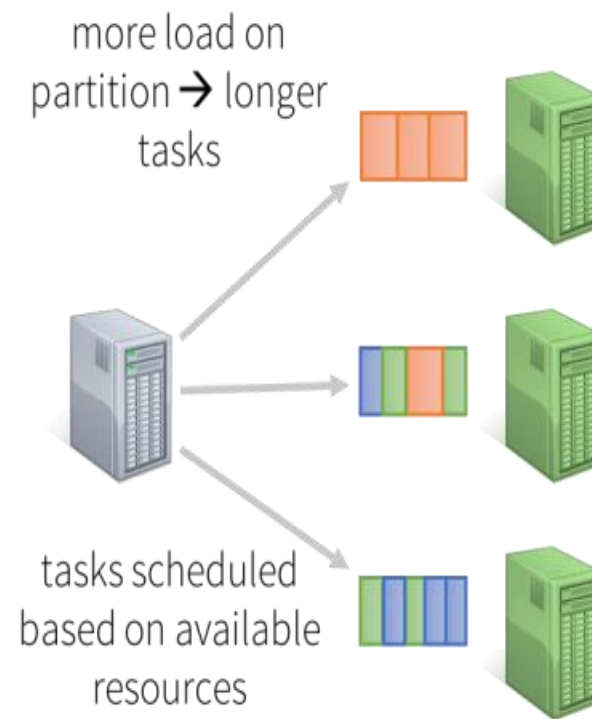
# Spark Streaming

## Traditional systems



static scheduling of continuous operators to nodes can cause bottlenecks

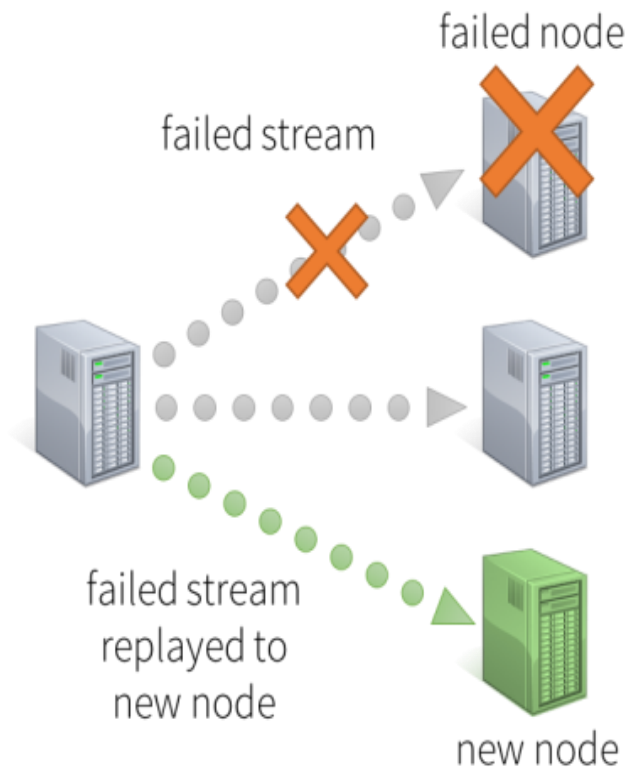
## Spark Streaming



dynamic scheduling of tasks ensures even distribution of load

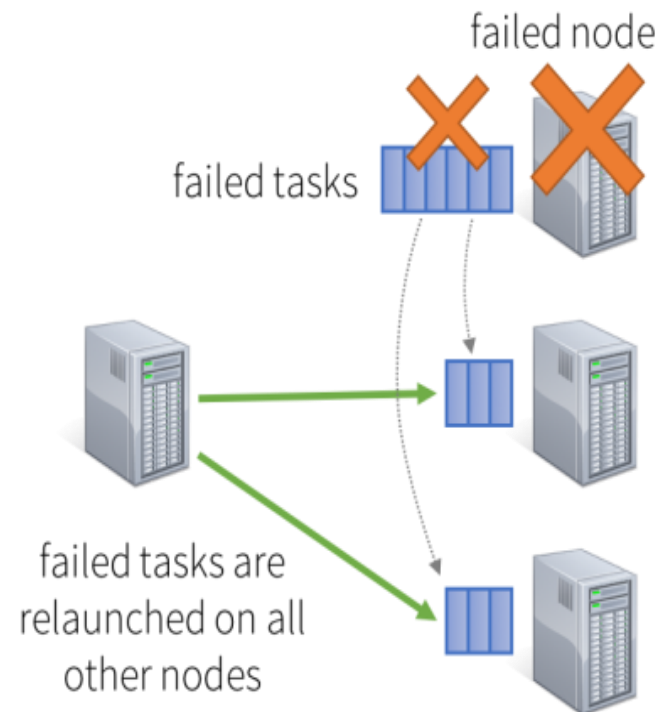
# Spark Streaming

## Traditional systems



slower recovery by using single node for recomputations

## Spark Streaming



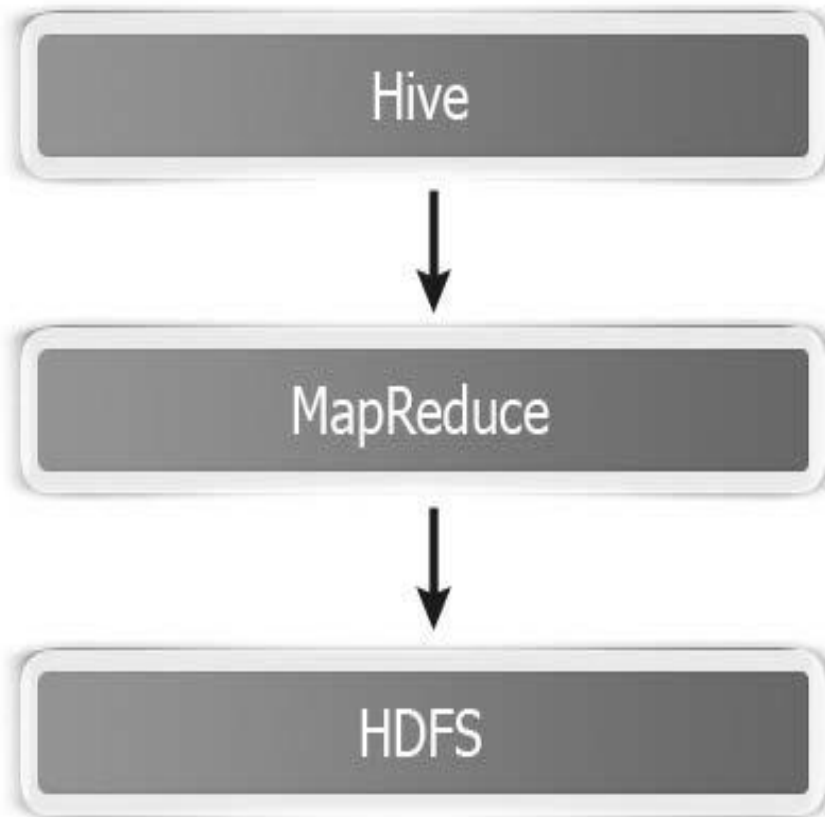
faster recovery by using multiple nodes for recomputations

# Spark SQL





# Spark SQL - Introduction

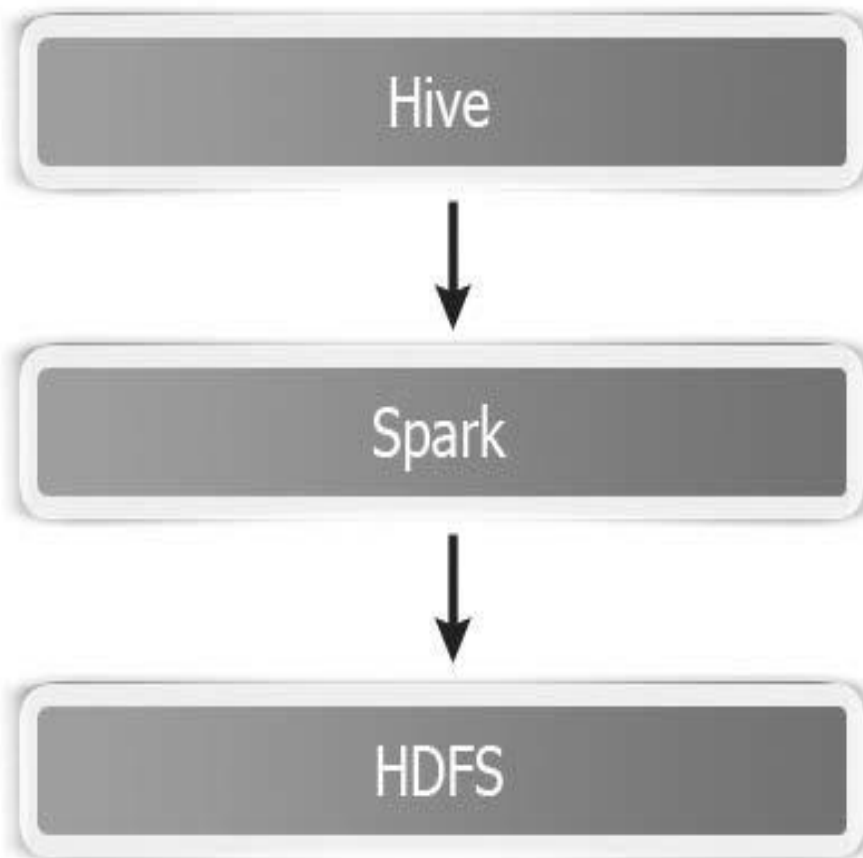


- Spark SQL is a relatively new component in Spark ecosystem, introduced in Spark 1.0 for the first time.
- It incorporates a project named Shark, which was an attempt to make Hive run on Spark.
- Hive is essentially a relational abstraction, which converts SQL queries to MapReduce jobs.



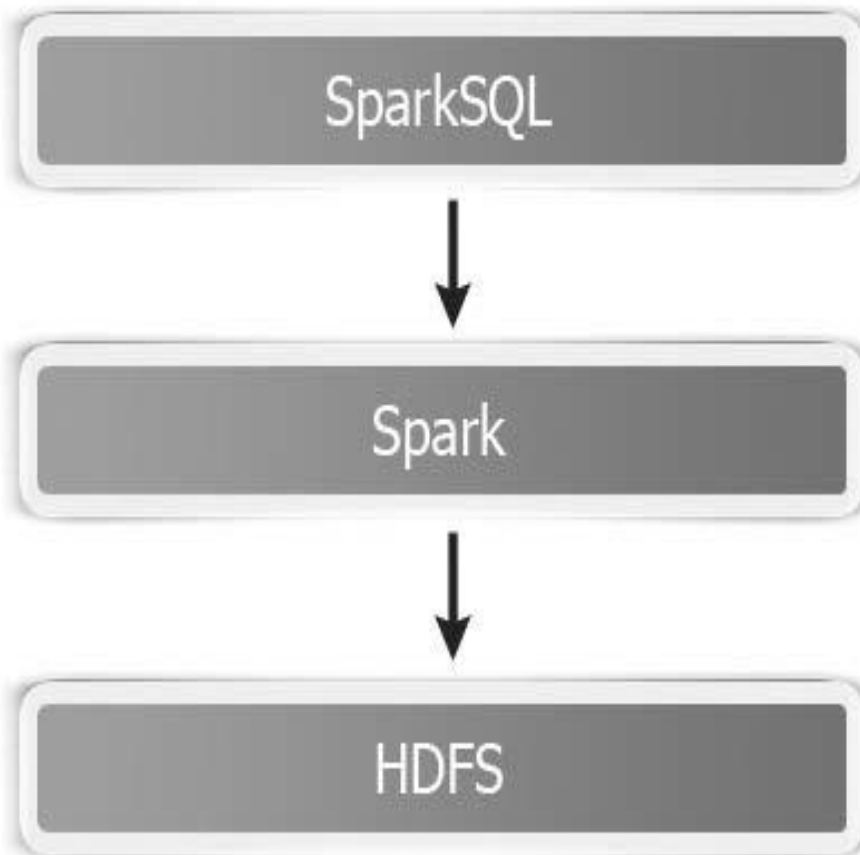
# Spark SQL

Shark replaced the MapReduce part with Spark while retaining most of the code base.



# Spark SQL

- Spark developers hit roadblocks and could not optimize it any further.
- Finally, they decided to write the SQL Engine from scratch and that gave birth to Spark SQL.



# DataFrame - Introduction

Spark SQL uses a programming abstraction called DataFrame.

It is a distributed collection of data organized in named columns.

The DataFrame API also ensures that Spark's performance is consistent across different language bindings.

DataFrame is equivalent to a database table, but provides much finer level of optimization.

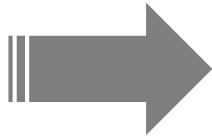
# DataFrame

- An RDD is an opaque collection of objects with no idea about the format of the underlying data.
- In contrast, DataFrames have schema associated with them.
- You can also look at DataFrames as RDDs with schema added to them.
- In fact, until Spark 1.2, there was an artifact called **SchemaRDD**, which has now evolved into DataFrame.
- They provide much richer functionality than SchemaRDDs.
- DataFrames also transparently load from various data sources, such as Hive tables, Parquet files, JSON files, and external databases using JDBC.
- DataFrames can be viewed as RDDs of row objects, allowing users to call the procedural Spark APIs such as map.



# Features of DataFrame

Here is a set of few characteristic features of DataFrame –



---

Ability to process the data in the size of Kilobytes to Petabytes on a single node cluster to large cluster.

---

Supports different data formats (Avro, csv, elastic search, and Cassandra) and storage systems (HDFS, HIVE tables, mysql, etc.).

---

State of art optimization and code generation through the Spark SQL Catalyst optimizer (tree transformation framework).

---

Can be easily integrated with all Big Data tools and frameworks via Spark-Core.

---

Provides API for Python, Java, Scala, and R Programming.

---

# SQL Context

- The other component of SparkSQL is SQLContext.
- The SQLContext class or any of its descendants acts like the entry point into all functionalities.
- You just need a SparkContext to build a basic SQLContext. you can also build a HiveContext for availing the benefit of a superset of the basic SQLContext functionality.
- It also provides more features such as the writing ability for queries by using the more comprehensive HiveSQL parser.
- Other features are accessing Hive UDFs and the read data ability from Hive tables.
- The entire data sources that are available to an SQLContext still exist.
- Therefore, you do not require an existing Hive setup for using a HiveContext.





# SQL Context

HiveContext is just packaged separately for avoiding the dependencies of Hive in the default Spark build.

For your applications, if these dependencies are not a concern, then HiveContext is recommended for Spark 1.3.

You can also use the `spark.sql.dialect` option to select the specific variant of SQL, which is used for parsing queries.

To change this parameter, you can use the `SET key=value` command in SQL or the `setConf` method on an `SQLContext`.

The future releases will be focused on getting `SQLContext` up to feature parity with a `HiveContext`.

# SQL Context



The only dialect available for an SQLContext is “sql” that makes use of a simple SQL parser.



In a HiveContext, the default is “hiveql”, which is much more comprehensive.



It is recommended for most use cases. On an SQLContext, the sql function allows applications to programmatically run SQL queries and then return a DataFrame as a result.

**Thank  
You  
For Your  
Attention**

**&**

**A**