# CIML Summer Institute
# Scalable Machine Learning


SDSC SAN DIEGO SUPERCOMPUTER CENTER

UC San Diego

# Scalable Machine Learning Agenda

8:00 - 8:30 -- Machine Learning Overview
8:30 - 9:15 -- R on HPC
9:15 - 9:30 -- Break
9:30 - 10:45 -- Spark
10:45 - 11:45 -- Lunch
11:45 - 12:30 -- Intro to Neural Networks / CNNs
12:30 - 12:45 -- Break
12:45 - 1:30 -- Deep Learning Layers & Models
1:30 - 2:00 -- Deep Learning Tutorial

# Spark

Mai H. Nguyen, Ph.D.

# SPARK


- **Computing platform for distributed computing**
  - Built-in parallelism & fault-tolerance on commodity cluster
  - Provides interactive querying, iterative analytics, streaming processing
  - Goals:  speed, ease of use, generality, unified platform
- **History**
  - Research project began in 2009 at UC Berkeley's AMPlab
  - Paper published in 2010
  - Contributed to Apache Software Foundation in 2013
  - Commercial version by Databricks

# SPARK

- Goals: **speed**, ease of use, generality, unified platform
- In-memory processing
  - Exploits distributed memory to cache data
  - Intermediate results written to memory whenever possible
- How does Spark manage data in distributed system?

# RESILIENT DISTRIBUTED DATASETS (RDDs)

- Spark central concept
  - Abstraction of data as distributed collection of objects

- Resilient Distributed Datasets (RDDs)
  - Data abstraction
  - Programming construct for storing data
  - Spark uses RDDs to distribute data and computations across nodes in cluster

# RDD

- Resilient Distributed **Dataset**
  - Collection of data
    - From files in local filesystem (text, JSON, etc.)
    - From data store (HDFS, RDBMS, NoSQL, etc.)
    - Created from another RDD
- Resilient **Distributed** Dataset
  - Data is divided into partitions
  - Partitions are distributed across nodes in cluster
- **Resilient** Distributed Dataset
  - Provides resilience (e.g., fault tolerance) to failures
  - History of operations performed on each partition is tracked to provide lineage-based fault tolerance
- All provided automatically by Spark engine

# SPARK CONTEXT

- Spark Context
  - Entry point to Spark engine
  - Provides way to create RDDs

```
from pyspark import SparkContext, SparkConf

conf = SparkConf()\
   .setAppName("RDD Example") \
   .config("config.option","config.value")
sc = SparkContext(conf=conf)
```

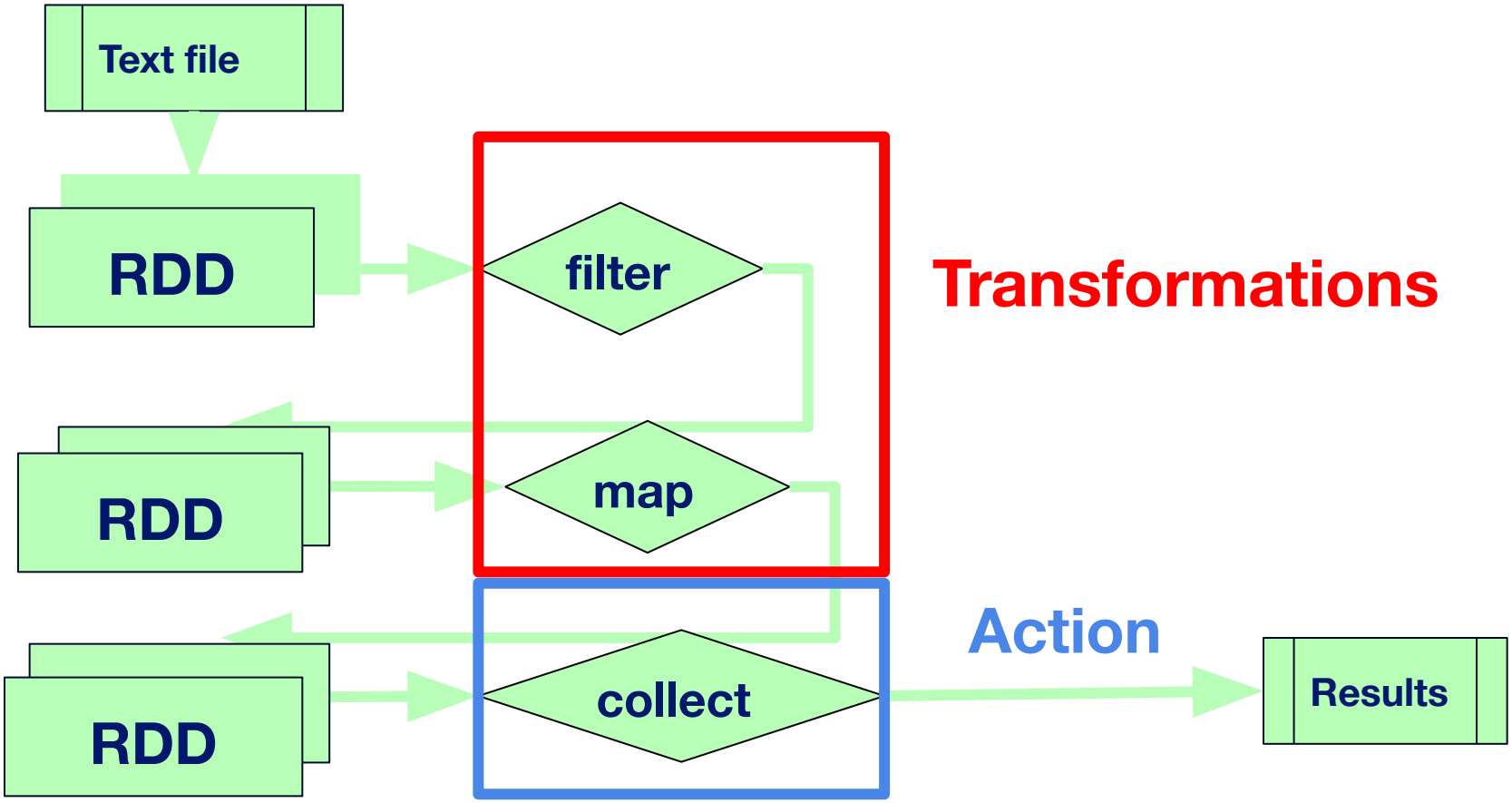
- SparkContext: connection to Spark engine
- SparkConf: configuration parameters for application

# CREATING RDDs

- Read data from files in local filesystem (text, JSON, etc.)
  - lines = sc.textFile(“inputfile.txt”)
- Data read in from data store (HDFS, RDBMS, NoSQL, etc.)
  - lines = sc.textFile(“hdfs://<path>/inputfile.txt”)
- Generate data
  - numbers = sc.parallelize(range(100),3)
    - Divide data into 3 partitions
- Created by transforming another RDD
  - newLines = lines.filter(lambda s: “Spark” in s)
- Note:  RDDs are **immutable**
  - To “change” RDD, create another RDD to hold changed data

# PROCESSING RDDs

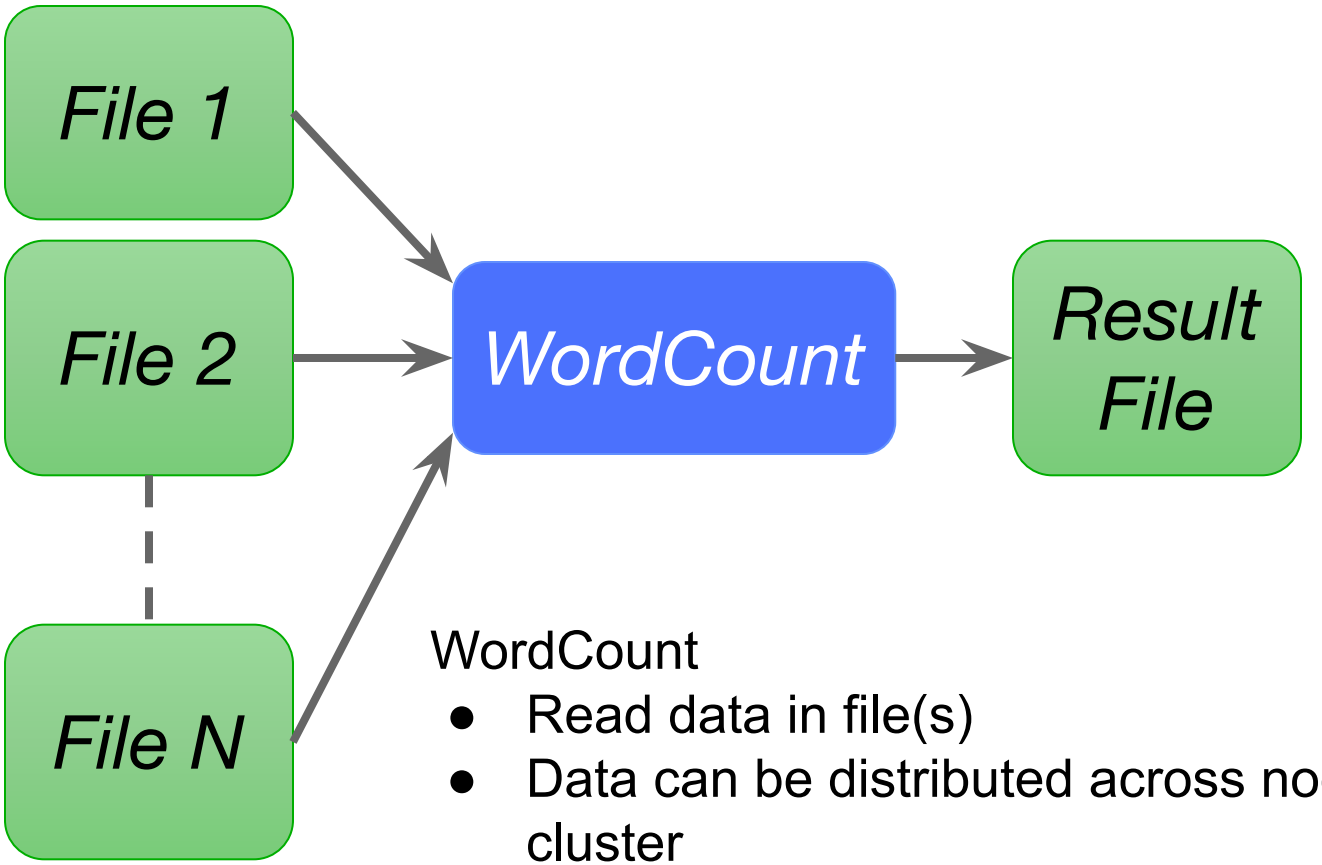
- RDDs can be processed using 2 types of operations
  - **Transformation**: Creates new RDD from existing RDD
  - **Action**: Runs computation(s) on RDD and returns value



# LAZY EVALUATION

- Transformations on RDDs have **lazy evaluation**
  - Transformations are not immediately processed
  - Plan of operations is built
- Operations executed when **action** is performed
  - i.e., actions force computation
- Allows for optimizations in generating physical plan
- Example:
  - filtered = strings.filter(strings.value.contains(“Spark”))
    - Nothing is returned
  - filtered.count()
    - ‘filter’ is performed, and count is returned

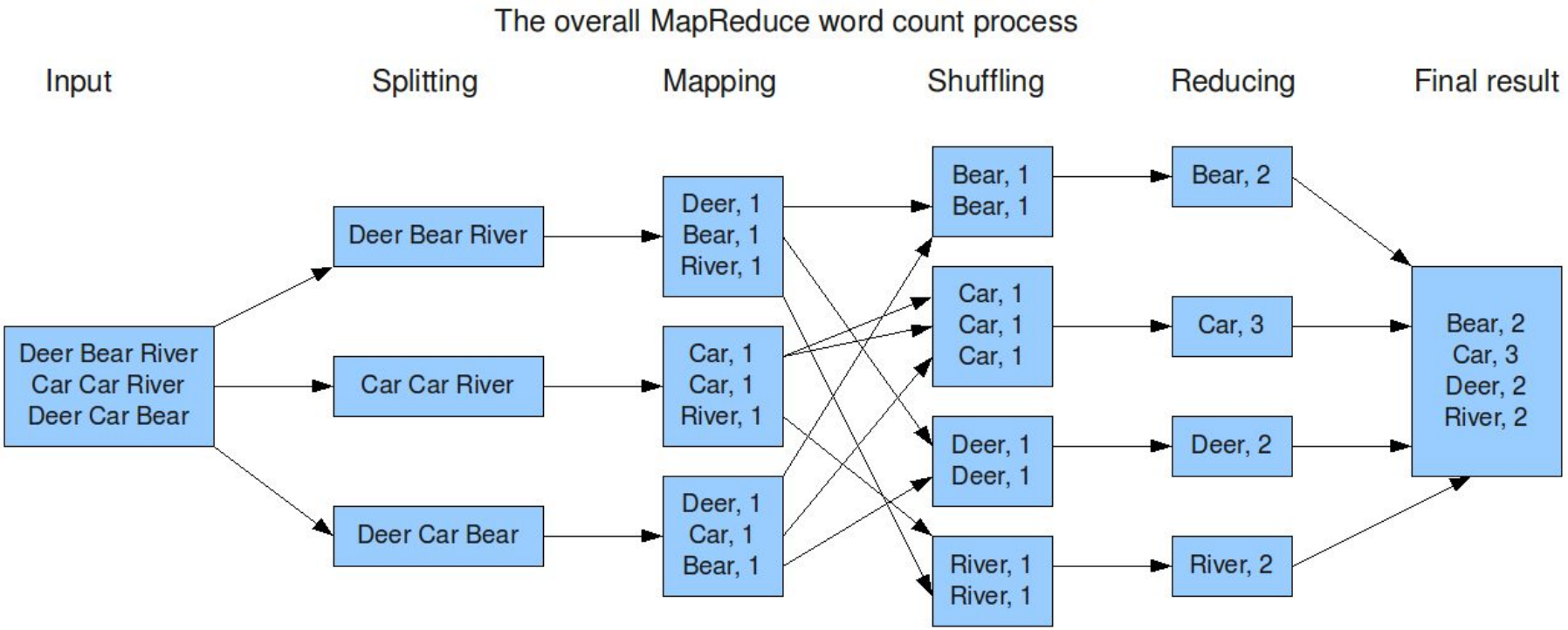
# WordCount




WordCount

- Read data in file(s)
- Data can be distributed across nodes in cluster
- Count number of occurrences of each word

# WordCount




https://www.todaysoftmag.com/article/1358/hadoop-mapreduce-deep-diving-and-tuning

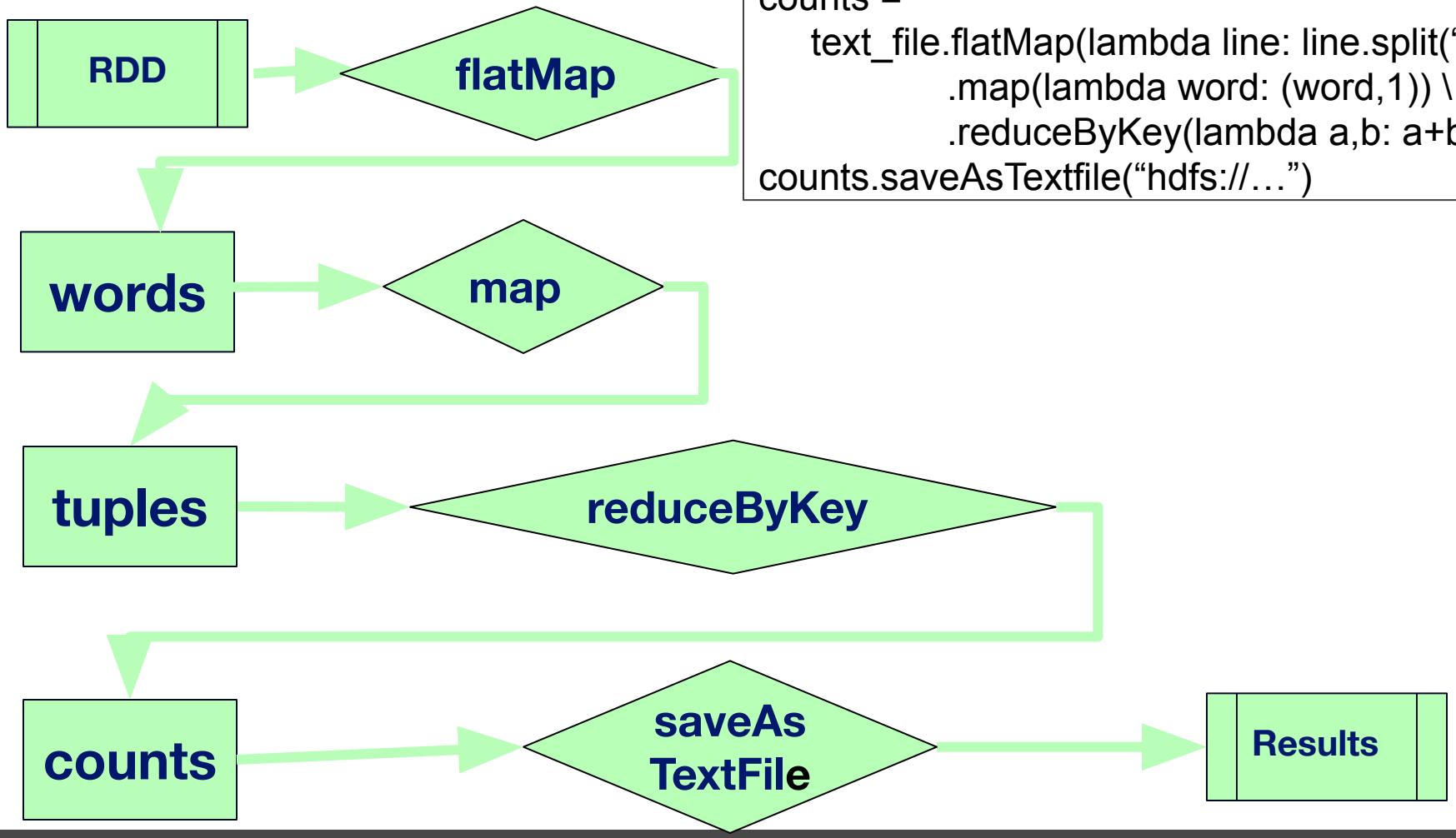
Data is partitioned across nodes

Map generates key-value pairs

Pairs with same key moved to same node

Reduce sums values for each key

# WordCount (RDD)




```
text_file = sc.textFile("hdfs://..")
counts =
    text_file.flatMap(lambda line: line.split(" ")) \
             .map(lambda word: (word,1)) \
             .reduceByKey(lambda a,b: a+b)
counts.saveAsTextfile("hdfs://…")
```

# DATAFRAMES & DATASETS

- **Extensions to RDDs**
  - Higher-level abstractions
  - Improved performance
  - Better scalability
- **Can convert to/from RDDs and use with RDDs**

# DATAFRAMES & DATASETS

**DataFrame**

- Lazy evaluation
- Data organized as collection of Rows
- No static type checking
- APIs in Java, Scala, Python, R

**DataSet**

- Lazy evaluation
- Data organized as collection of Rows
- Provides static type checking
- APIs in Java and Scala

# USING DATAFRAMES

- Spark Session
  - Entry point to Spark engine
  - Note that SparkContext is now **SparkSession**

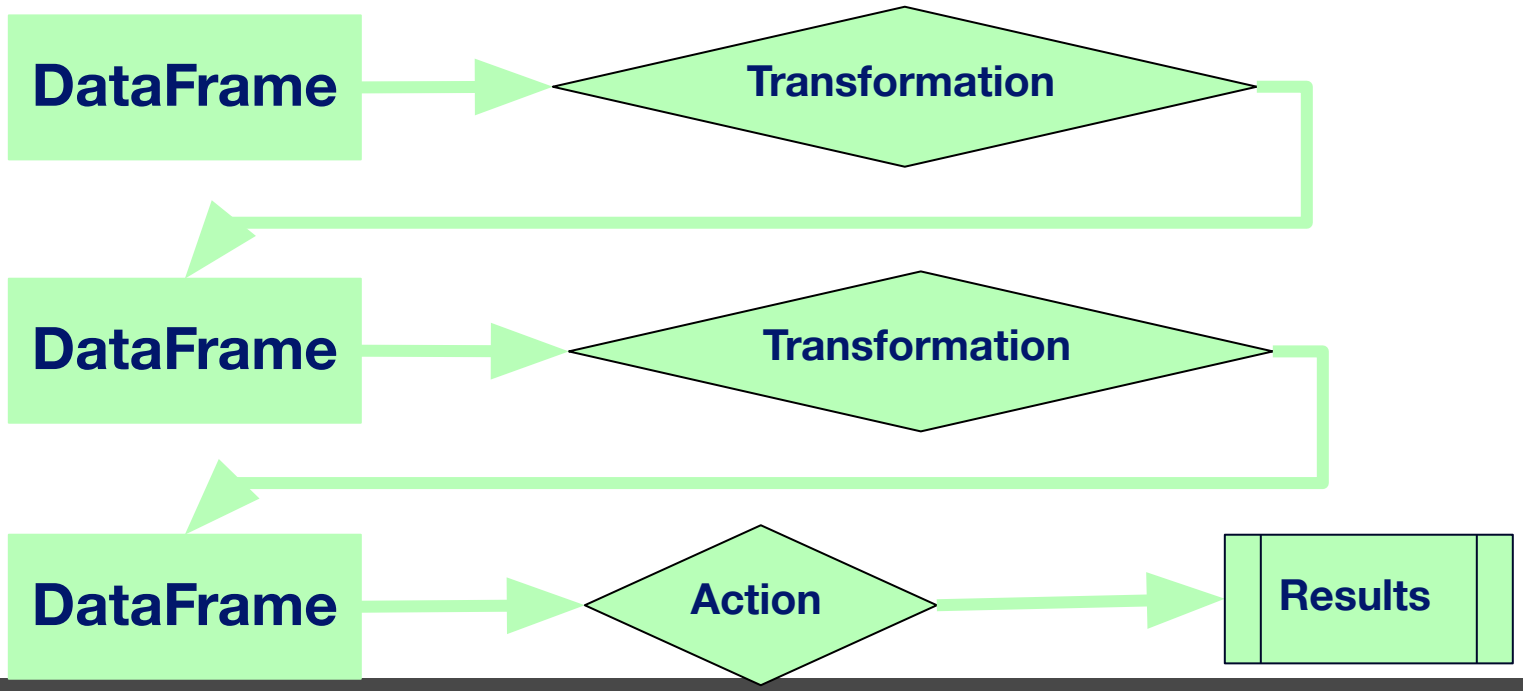
```
from pyspark import SparkSession

conf = SparkSession \
   .builder \
   .appName("DataFrame Example") \
   .config("config.option","config.value")\
   .getOrCreate()

spark =
   SparkSession.builder.config(conf=conf)\
                .getOrCreate()
```

# CREATING DATAFRAMES

- Read data from files in local filesystem (text, JSON, etc.)
  - df = spark.read.csv("data.csv", header="True")
- Data read in from data store (HDFS, RDBMS, NoSQL, etc.)
  - df = spark.read.csv("hdfs:///<path>/data.csv")
- Generate data
  - empl_0 = Row(id="123", name="John")
  - empl_1 = Row(id="456", name="Mary")
  - employees = [empl_0, empl_1]
  - df = spark.createDataFrame(employees)
- Created by transforming another DataFrame
  - filter_df = df.filter(col("name")=="Mary"))

# DATAFRAME TRANSFORMATIONS & ACTIONS

- Similar to RDDs, DataFrames can be processed using transformations and actions
- Transformations on DataFrames also have lazy evaluation
- Operations executed when action is performed



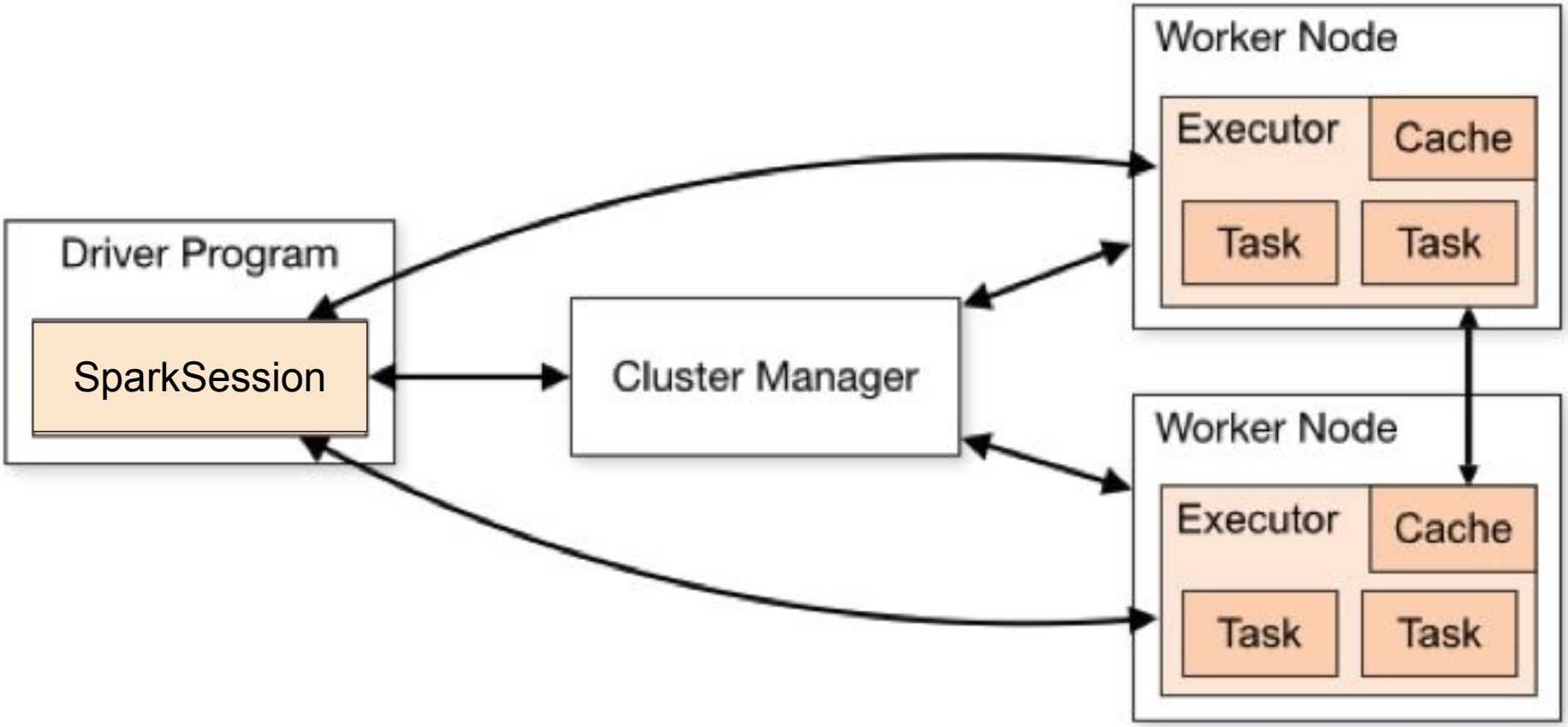
# DATA PERSISTENCE

- Persist data through caching
  - Data is stored in memory to avoid re-computing
- Can specify different storage levels
  - In memory, on disk, serialized in memory, etc.
- Examples
  - df.cache() – MEMORY_ONLY
  - df.persist(MEMORY_ONLY_SER) – Serialized in memory
  - df.unpersist() – Remove from cache

# SPARK PROGRAM STRUCTURE

- **Start Spark session**
  - spark = SparkSession.builder.config(conf=conf).getOrCreate()
- **Create distributed dataset**
  - df = spark.read.csv(“data.csv”,header=”True”)
- **Apply transformations**
  - new_df = df.filter(col(“dept”) == “Sales”)
- **Perform actions**
  - df.collect()
- **Stop Spark session**
  - spark.stop()

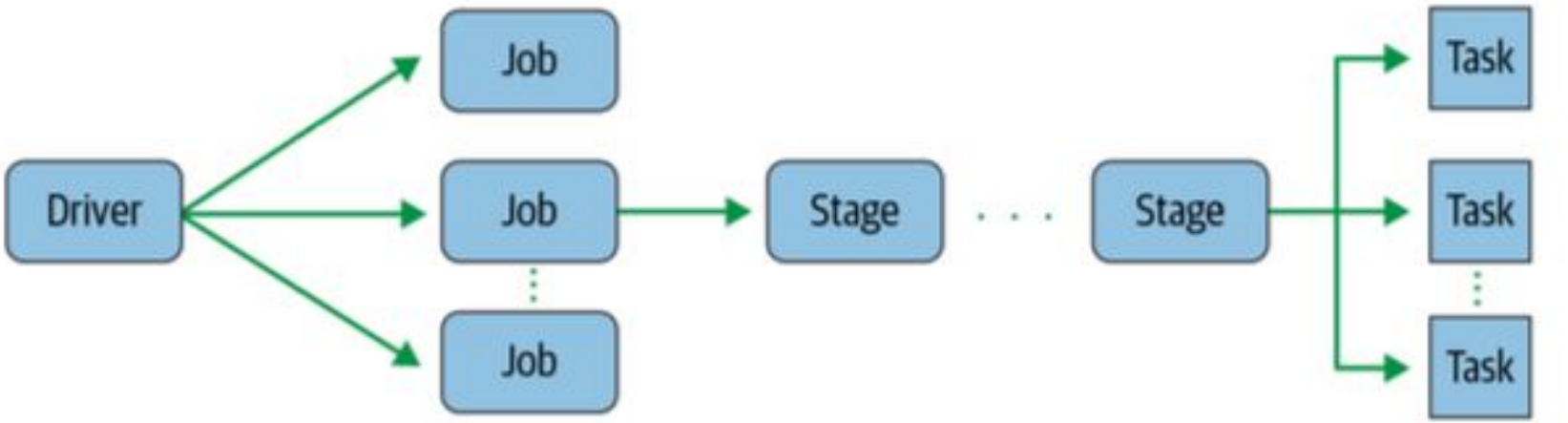
# SPARK ARCHITECTURE



# SPARK PROCESS CONCEPTS

- **Application**
  - User program consists of driver program and executors on cluster
- **SparkSession**
  - Object that provides point of entry to interact with underlying Spark functionality using Spark APIs
- **Job**
  - Parallel computation consisting of multiple operations that are executed in response to Spark actions
- **Stage**
  - Each job gets divided into smaller units called stages
- **Task**
  - Single unit of work or execution sent to executor

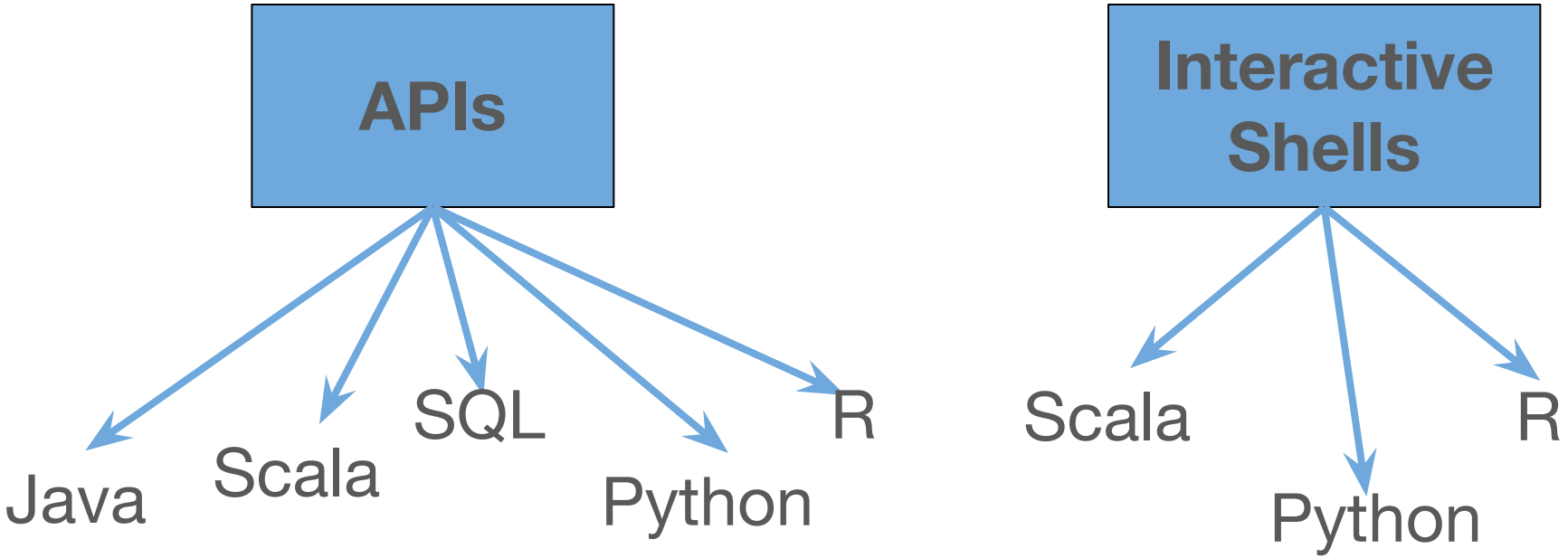
# SPARK APPLICATION




- Driver creates SparkSession object
- Driver converts Spark application into one or more jobs
- An action creates a job
- DAG (directed acyclic graph) of instructions built for each job
- Each node in DAG is single or multiple Spark stages
- Each stage is broken down into tasks
- Tasks are distributed to executors

# SPARK INTERFACE

Goals: speed, **ease of use**, generality, unified platform



# RDD WORDCOUNT EXAMPLE IN SPARK

Spark RDD API available in Python, Scala, Java, and R

```
text_file = sc.textFile("hdfs://...")
counts = text_file.flatMap(lambda line: line.split(" ")) \
             .map(lambda word: (word, 1)) \
             .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://...")
```

```
val textFile = sc.textFile("hdfs://...")
val counts = textFile.flatMap(line => line.split(" "))
                 .map(word => (word, 1))
                 .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```

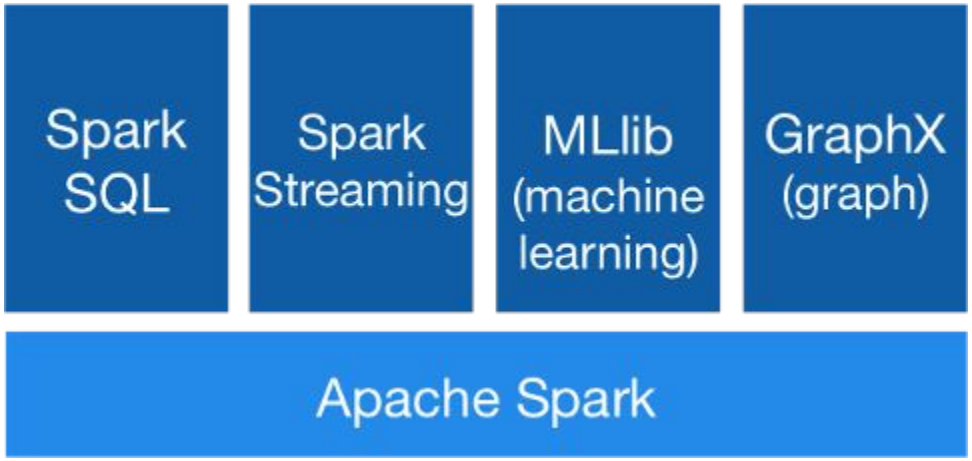
```
JavaRDD<String> textFile = sc.textFile("hdfs://...");
JavaPairRDD<String, Integer> counts = textFile
    .flatMap(s -> Arrays.asList(s.split(" ")).iterator())
    .mapToPair(word -> new Tuple2<>(word, 1))
    .reduceByKey((a, b) -> a + b);
counts.saveAsTextFile("hdfs://...");
```

# SPARK - GENERALITY

- Goals:  speed, ease of use, **generality,** unified platform
- Support for several data sources
  - Local file systems, HDFS, RDBMSs, MongoDB, Kafka, AWS S3, etc.
- Can run on various platforms
  - Hadoop, Kubernetes, cloud, standalone
- Support for multiple workloads
  - batch, streaming
  - machine learning, SQL, graph processing

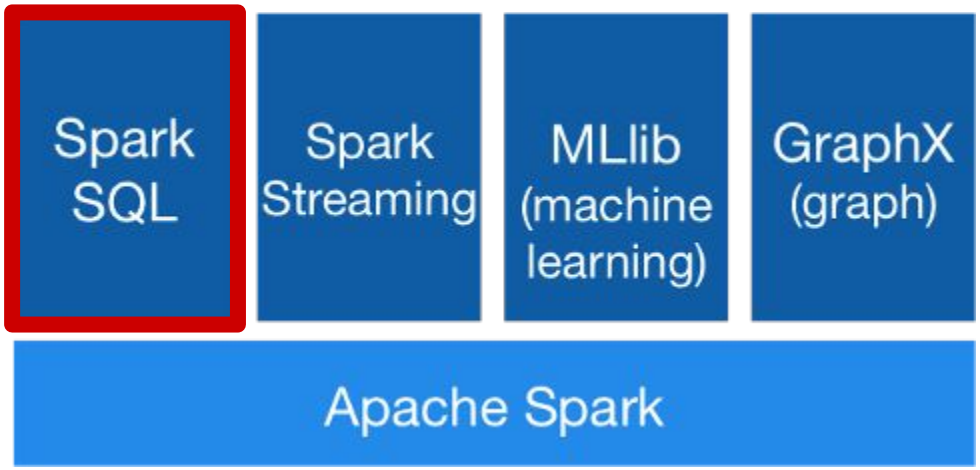
# SPARK - UNIFIED PLATFORM

- Goals: speed, ease of use, generality**, unified platform**




- Provides unified platform for various analytics processing
- **Spark engine** provides core capabilities for distributed processing
- **Spark libraries** provide additional higher-level functionality for diverse workloads

# SPARK SQL




- **Structured Data Processing**
  - Provides support for SQL and query processing
  - Structure of data and computations allow for efficient query plan can be constructed
  - Has APIs for SQL, Scala, Java, Python, and R
  - Generated underlying code is identical

# SPARK SQL

- Execute SQL queries
  - SQL

    spark.sql(“SELECT max(count)
    FROM flight_data”).take(1)

  - PySpark

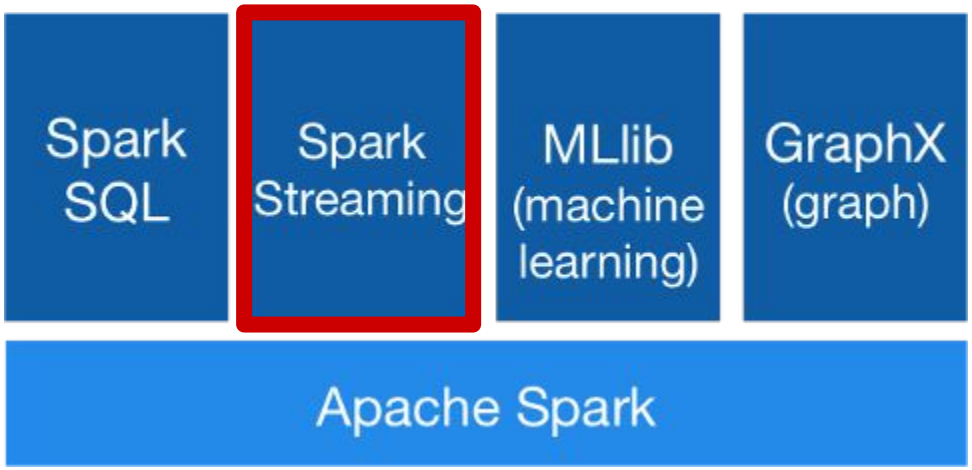
    from pyspark.sql.functions import max
    flight_data.select(max(“count”)).take(1)

# SPARK SQL

- Integrate SQL queries with Spark commands

```
df = spark.sql (“SELECT * FROM Employees”)
df.show(100)

num_employees =
   df.select(“Age”,”Dept”,”Salary”)
      .groupBy(“Dept”)
      .where(df.Salary > 80000)
      .count()
```

# SPARK STREAMING




- **Streaming Data Processing**
  - Scalable processing for real-time analytics
  - Structured streaming
    - Data stream is divided into micro-batches of data
    - Same operations for static data can be used
  - Has APIs for Scala, Java, and Python

# REAL-TIME ANALYTICS

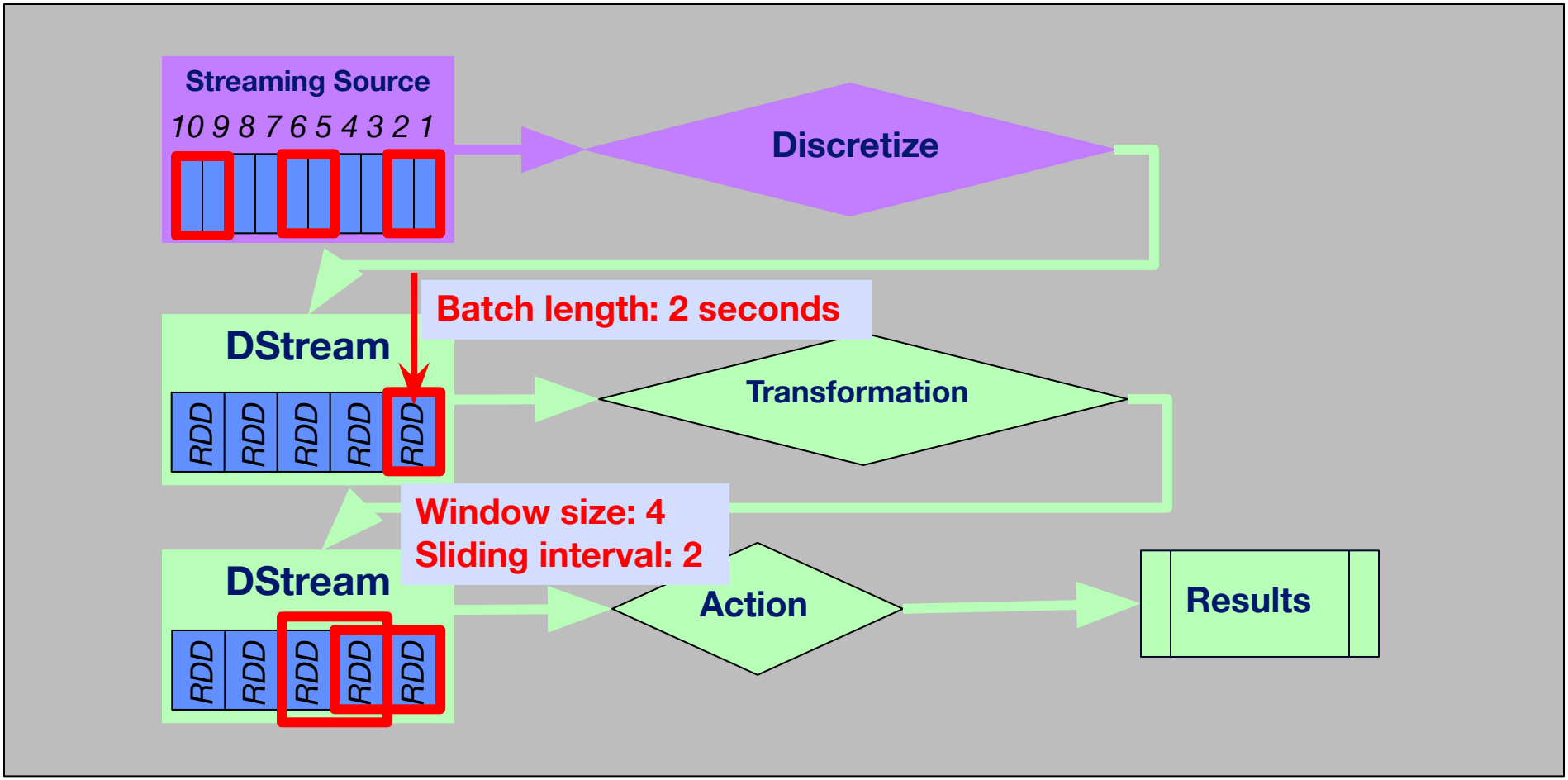
- (Near) Real-Time Analytics
  - Analysis and use of data as it enters system
- Examples
  - Identifying fraudulent credit card transaction at point-of-sale
  - Viewing orders as they happen for up-to-date inventory tracking and trend analysis
  - Understanding trending topics of tweets/news articles/etc.

# SPARK STREAMING

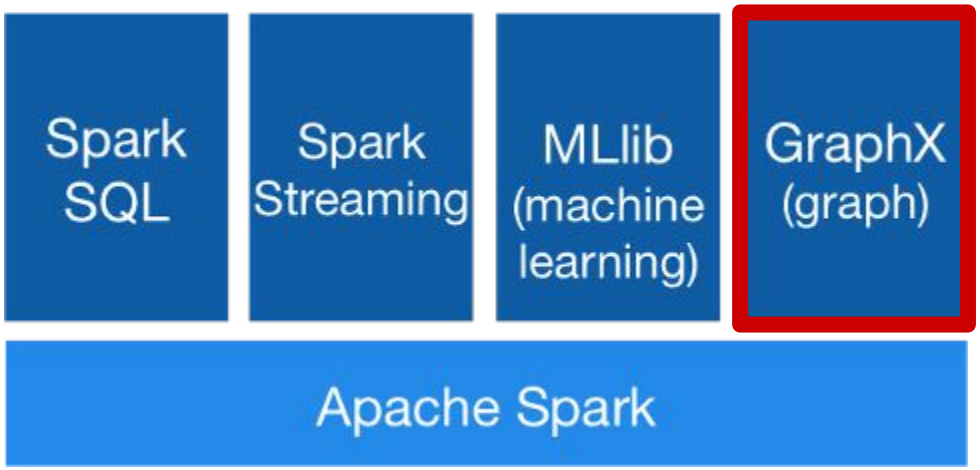
- Input data stream is divided into batches of data that are processed by Spark engine
- DStream:  high-level abstraction
  - Implemented as sequence of RDDs
- Any Spark operation can be applied to DStreams


https://spark.apache.org/docs/latest/streaming-programming-guide.html

# SPARK STREAMING



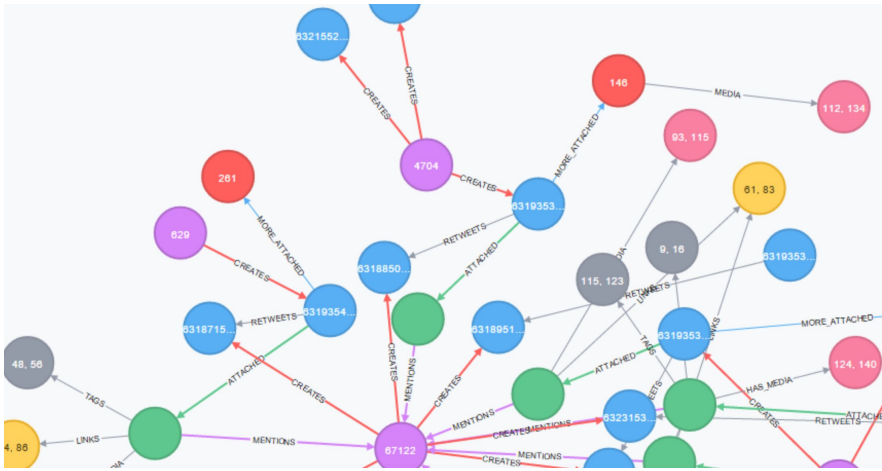
# SPARK GRAPHX / GRAPHFRAMES




- **Graph Computation**
  - Distributed graph processing
  - Special structures for storing vertex and edge information & operations for manipulating graphs
  - GraphX (RDD-based) & GraphFrames (DF-based)
  - Has APIs in Scala, Java, Python (GraphFrames)

# SPARK GRAPHX / GRAPHFRAMES

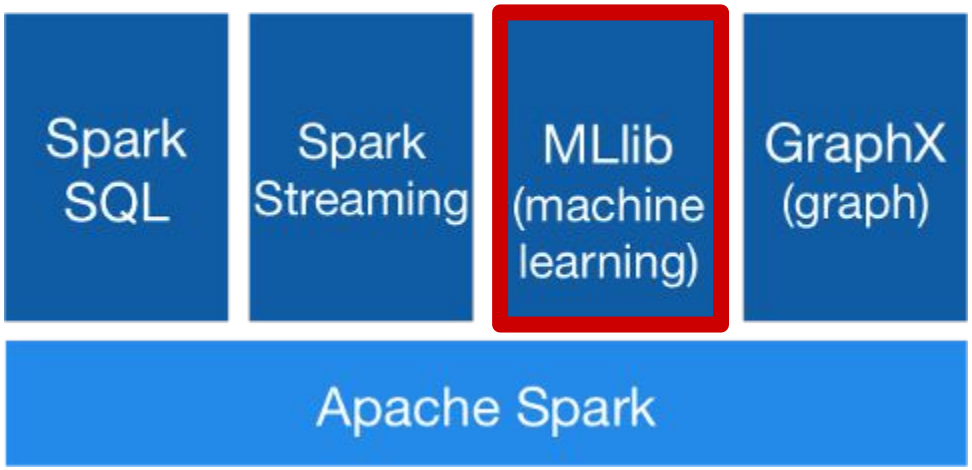
- **Graph analytics**
  - Analysis of relations among entities
- **Data represented as graph**
  - Entities are vertices
  - Relationships are edges
- **Example: Analyzing tweets**
  - Extract conversation threads
  - Find interacting groups
  - Find influencers in community



# SPARK GRAPHX / GRAPHFRAMES

- Graph operators & algorithms
  - Connected Components
  - PageRank
  - Triangle Counting
  - Label Propagation Algorithm
  - Shortest Paths

# SPARK MLLIB




- **Machine Learning**
  - Scalable machine learning library
  - Distributed implementations of machine learning algorithms and utilities
  - Has APIs for Scala, Java, Python, and R

# SPARK MLLIB ALGORITHMS

- **Machine Learning**
  - Classification, regression, clustering, etc.
  - Evaluation metrics
- **Statistics**
  - Summary statistics, sampling, etc.
- **Utilities**
  - Dimensionality reduction, transformation, etc.
- **ML Pipelines**
  - Similar to scikit-learn

# MLLIB EXAMPLE: STATISTICS

```
from pyspark.sql.functions import rand

# Generate random numbers
df = sqlContext.range(0,10)
      .withColumn("rand1", rand(seed=10))
      .withColumn("rand2", rand(seed=27))

# Show summary statistics
df.describe().show()

# Compute correlation
df.stat.corr("rand1","rand2")
```

# MLLIB EXAMPLE: CLUSTER ANALYSIS

```
from pyspark.ml.clustering import KMeans

# Read and parse data
data = spark.read.csv("data.csv", header="true")


# k-means model for clustering
kmeans = Kmeans().setK(3).setSeed(123)
model = kmeans.fit (data)
for center in model.clusterCenters()
   print (center)
```

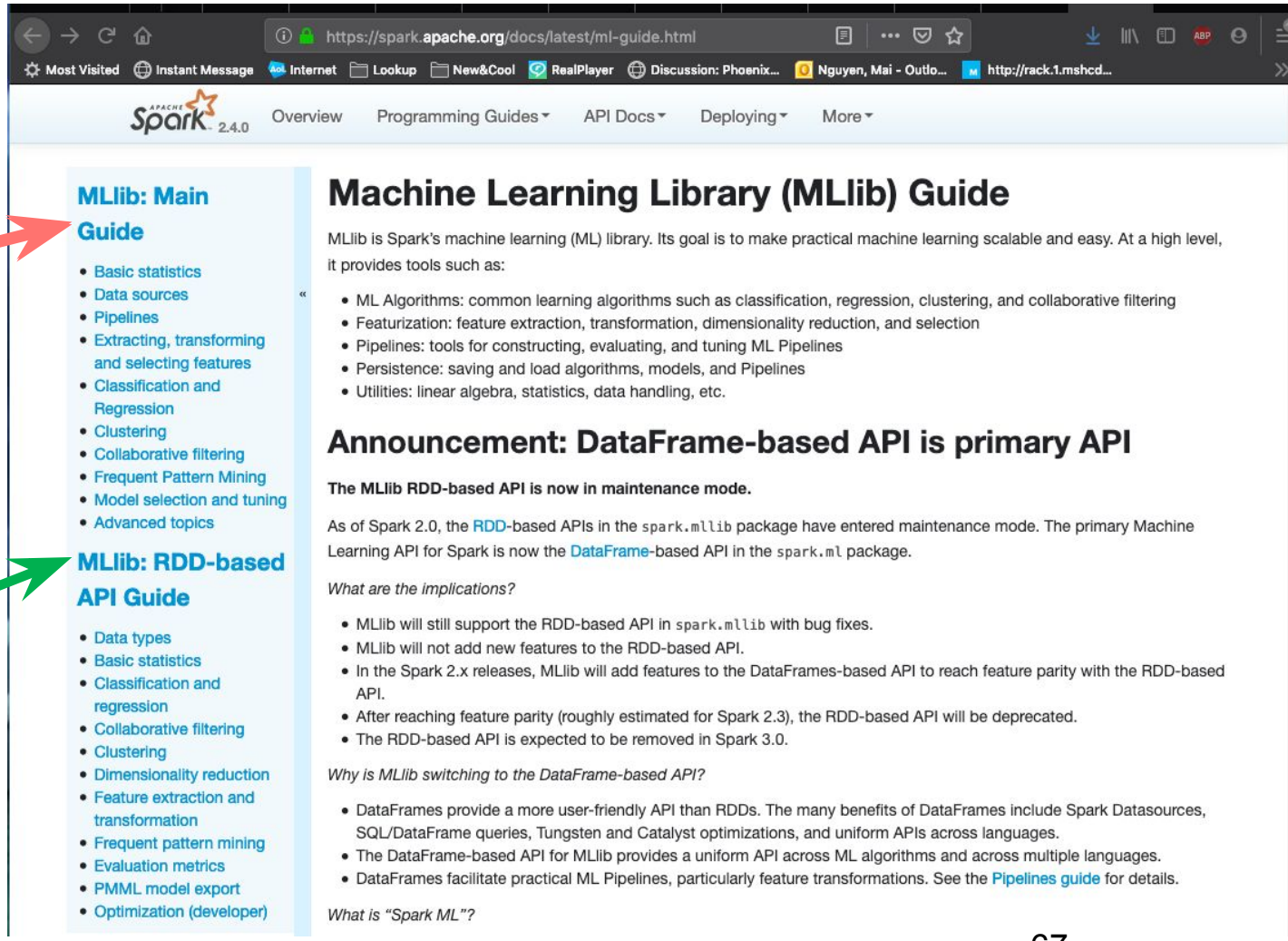
# MLLIB EXAMPLE: CLASSIFICATION

```
from pyspark.ml.classification import DecisionTreeClassifier

# Split data into train & test sets
trainDF,testDF = data.randomSplit([0.7,0.3],seed=123)

# Build model
dt = DecisionTreeClassifier(
          featuresCol='features',
          labelCol='label',
          predictionCol='prediction')
model = dt.fit(trainDF)

# Test model
predictions = model.transform(testDF)
```

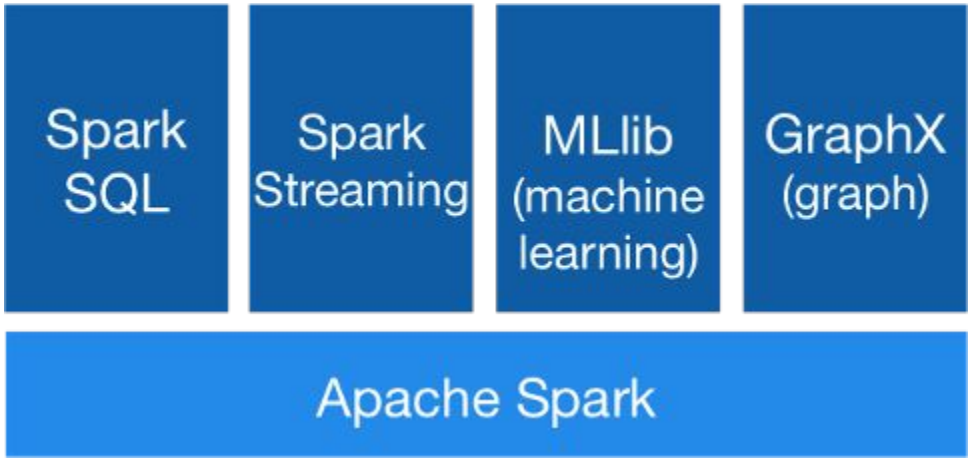
# MLLIB LIBRARIES

**ml**
**(DataFrame-based)**

**mllib**
**(RDD-based)**

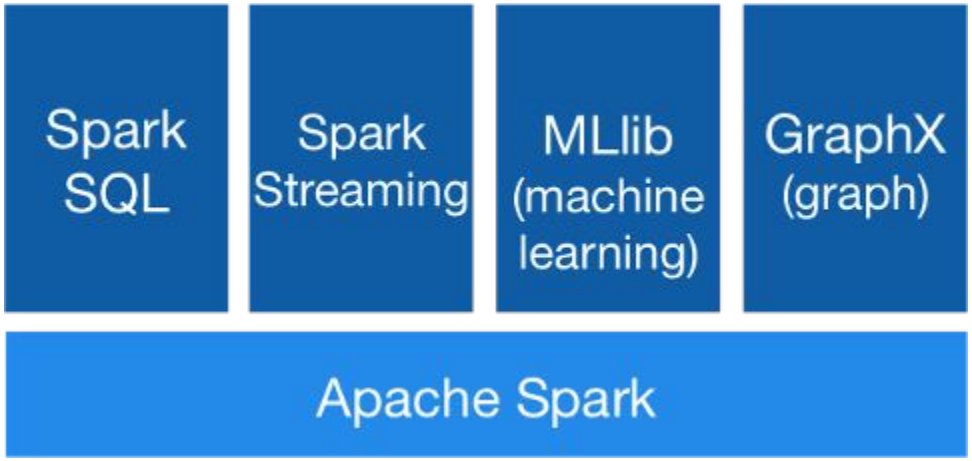


# SPARK LIBRARIES




- **Spark Libraries**
  - Use Spark engine as core
  - Extend functionality to particular applications
  - Third-party packages: https://spark-packages.org

# SPARK




Goals: speed, ease of use, generality, unified platform

# Spark

- Goals:  speed, ease of use, generality, unified platform
- RDDs
    - Transformations & actions
    - Lazy evaluation
- DataFrames & DataSets
- Spark program structure
- Spark architecture
- Spark process concepts
- Spark APIs & interactive shells
- Support for different platforms & workloads
- Spark libraries

# Spark Demo

- **Demo**
  - PySpark
  - To show common DataFrame operations

- **Code**
  - 3.3_spark/pyspark-intro/pyspark-dataframe.ipynb