

# **Среда Spark**

## **Spark RDD / Spark SQL**

### **Лекция 2**

**Кирилл Сысоев**

# Обо мне

5+ лет в Big Data

HSE University

Senior Data Engineer

OneFactor/UZUM Data

Hadoop, Spark, ClickHouse, Kafka, Docker

Python/Scala, SQL



[t.me/KRSysoev](https://t.me/KRSysoev)  
[ksysoev@hse.ru](mailto:ksysoev@hse.ru)

# Содержание курса

1. Введение в Big Data: как работают и где находятся большие данные;
- 2. Среда Spark. Spark RDD / Spark SQL;**
3. Advanced SQL;
4. Spark ML / Spark TimeSeries;
5. Advanced ML и проверка результатов качества моделей;
6. Spark GraphX /Spark Streaming;
7. Экосистема Spark (MLFlow, AirFlow,H2O AutoML);
8. Spark в архитектуре проекта / Spark CI/CD.

# **Взаимодействие**

## **Общение:**

Мой telegram – личные вопросы/консультации/рекомендации

## **Лекции + ДЗ:**

Telegram-чат «DS-15 Промышленное машинное обучение Spark» – после лекций буду туда публиковать материалы лекций и описание ДЗ с дедлайном

## **Сдача ДЗ:**

Почта – в установленный дедлайн буду ждать письмо с вложением

# Кратко про прошлую лекцию

## 3 основных вопроса в Big Data

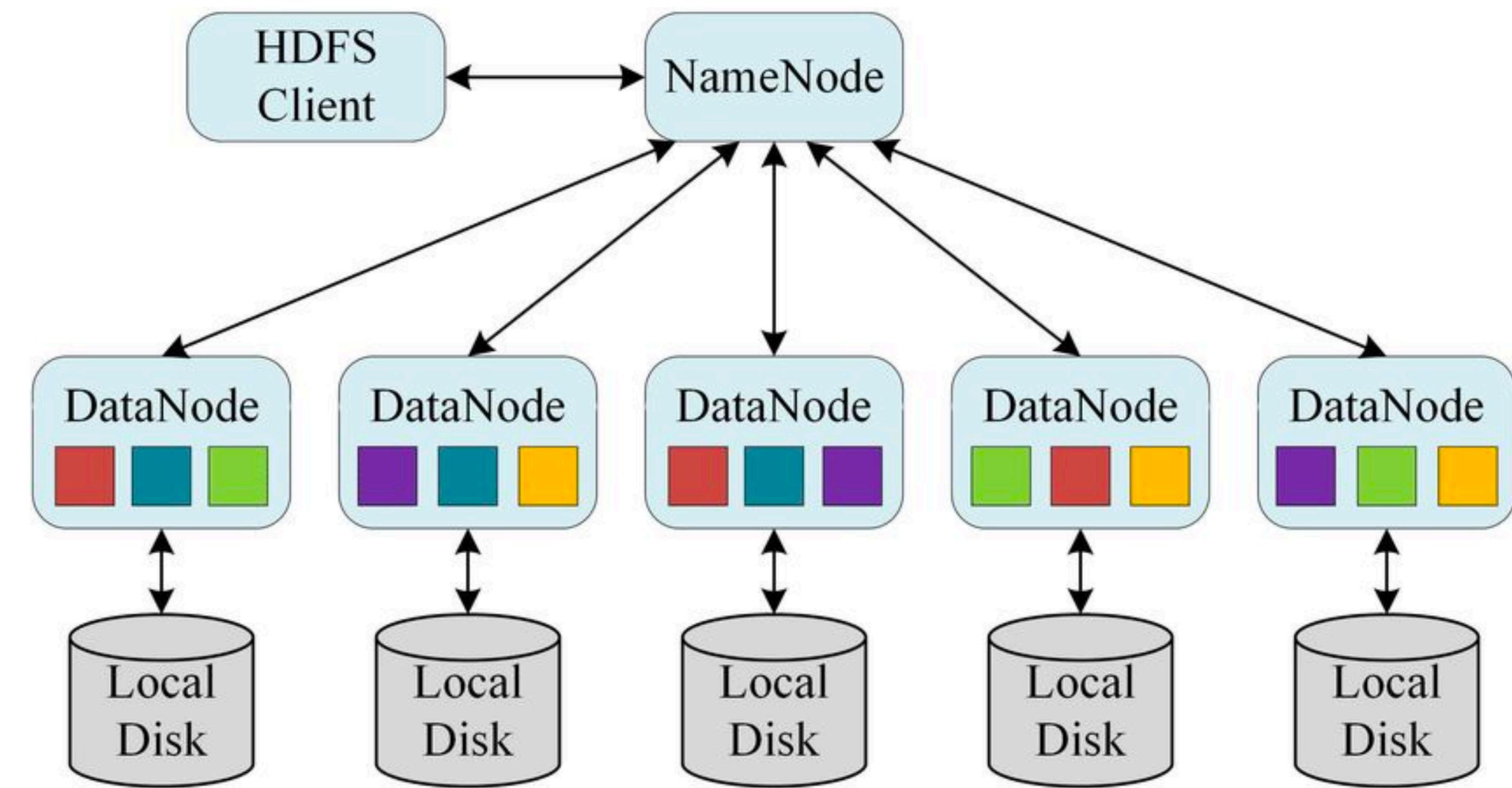
- 1. Как хранить большие данные? (Storage Layer)**
- 2. Как обрабатывать большие данные? (Processing layer)**
- 3. Как управлять ресурсами кластера? (Resource management layer)**

# Кратко про прошлую лекцию

## Как хранить большие данные?

HDFS – распределенная файловая система, разработанная для работы с большими объемами данных в кластерах

1. Распределенное хранение данных
2. Отказоустойчивость
3. Ограниченнaя поддержка изменений в файлах
4. Совместимость с Hadoop
5. Высокая масштабируемость
6. Экономичность

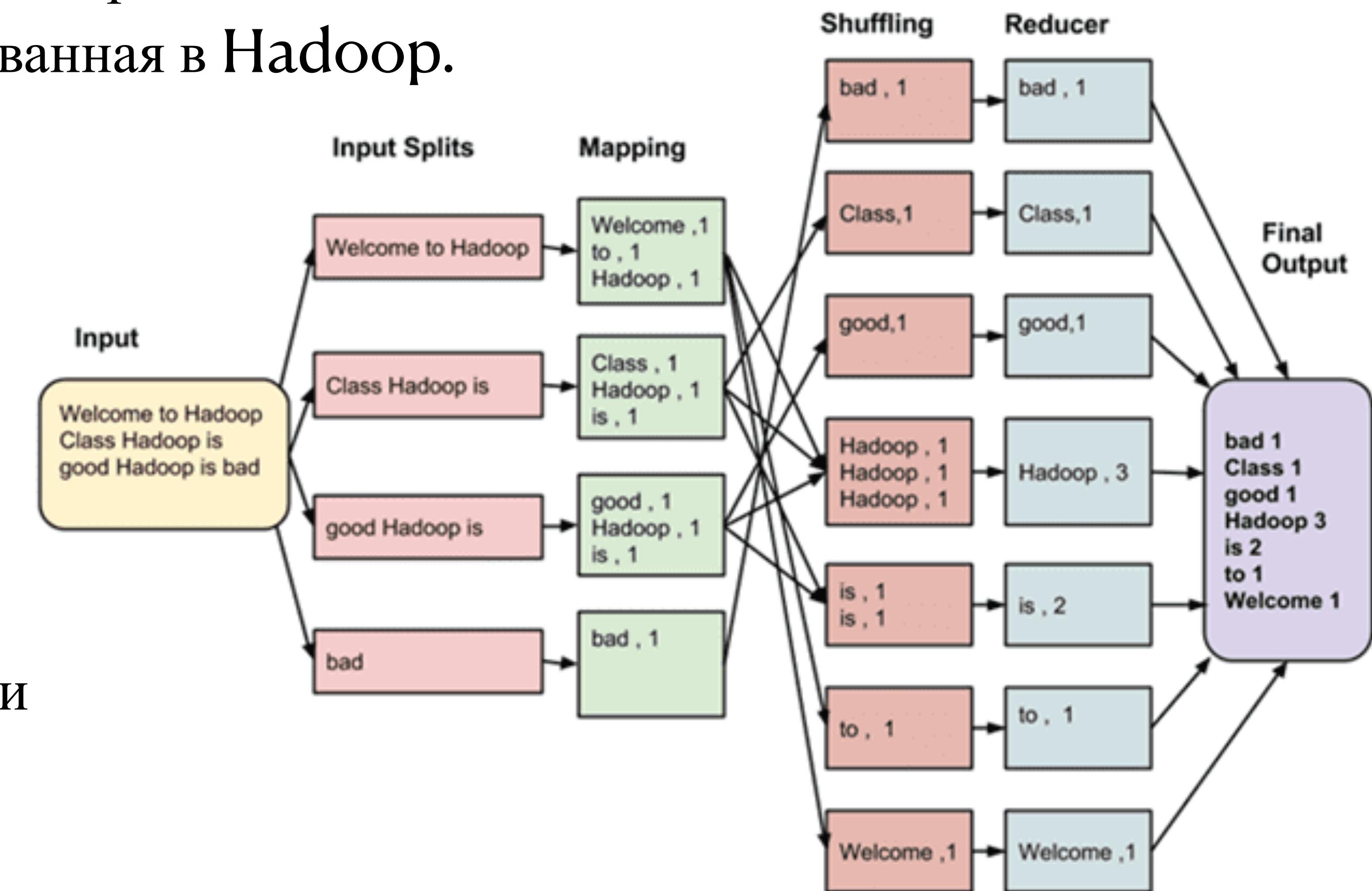


# Кратко про прошлую лекцию

## Как обрабатывать большие данные?

MapReduce — это модель распределенной обработки больших данных, разработанная Google и реализованная в Hadoop.

1. Распределенная обработка данных
2. Двухэтажная модель: Map и Reduce
3. Масштабируемость
4. Отказоустойчивость
5. Интеграция с Hadoop
6. Поддержка только дисковой обработки

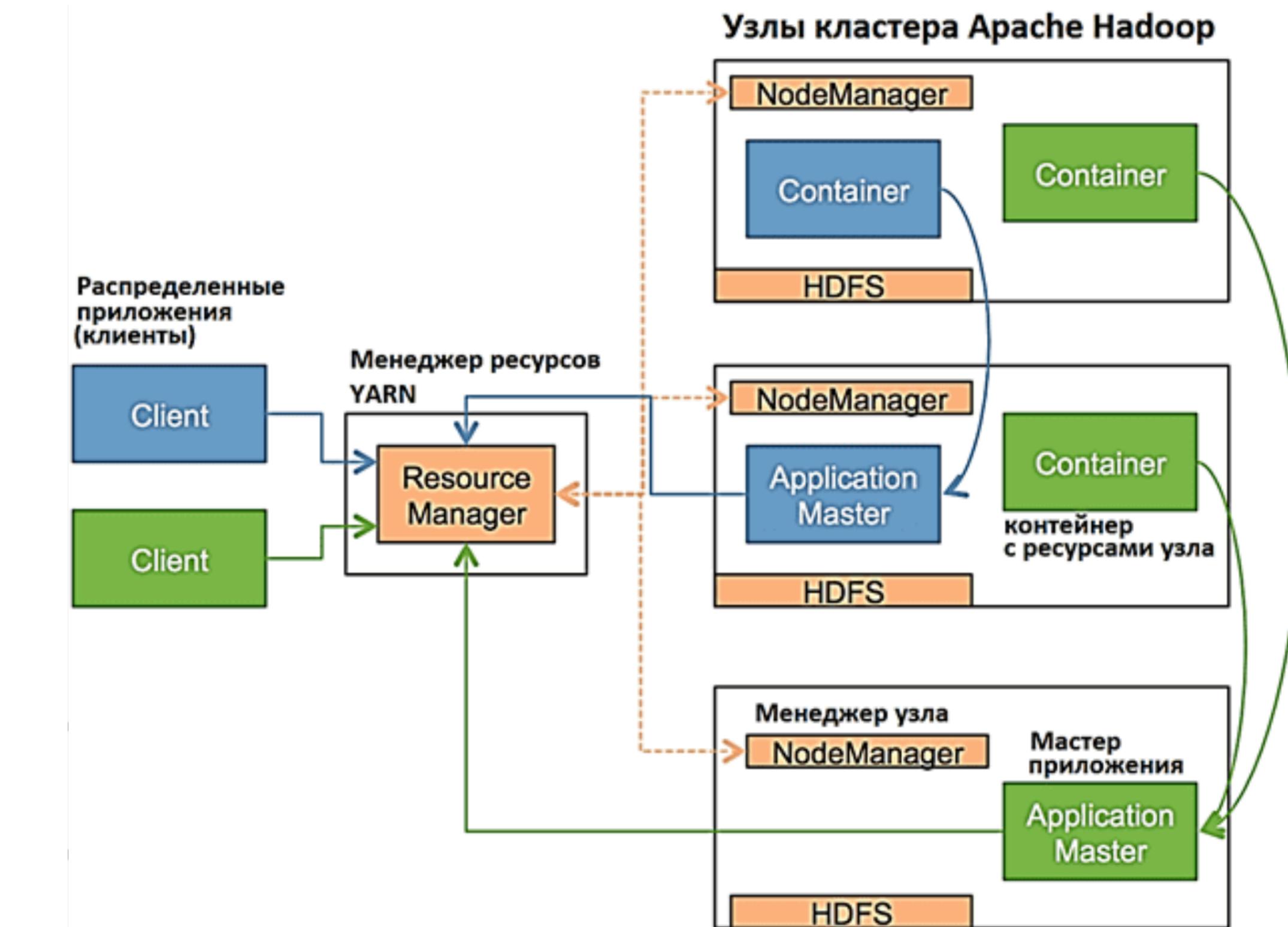


# Кратко про прошлую лекцию

## Как управлять ресурсами кластера?

YARN – ресурсный менеджер в экосистеме Hadoop, который отвечает за распределение вычислительных ресурсов между различными приложениями

1. Универсальная платформа для запуска различных вычислительных фреймворков
2. Гибкое управление ресурсами
3. Высокая масштабируемость
4. Отказоустойчивость
5. Поддержка многопользовательского режима



# Зачем нам нужен еще один инструмент?

- Mapreduce на десятки сложных операций
- Чтобы сразу со сложными функциями
- SQL или команды как в `pandas`
- Использовать машинное обучение на больших данных
- Обработка потоков данных
- Работа с графами

Также:

- YARN, Mesos, Kubernetes, Amazon
- Чтобы локально можно было гонять данные
- Нужно разные языки использовать
- Данные бывают из текстовых файлов
- Внешние плагины тоже не помешают

# MapReduce

```
(  
    data  
        .map(lambda x: (x[0].split(' '), x[1]))  
        .flatMap(lambda x: [(i, x[1]) for i in x[0]])  
        .saveAsTextFile('/opt/bitnami/spark/output/flatmap')  
)
```

# SQL & DataFrame

```
#Showing the data  
df.show()
```

```
+---+---+  
|Company| Person|Sales|  
+---+---+  
| GOOG| Sam|200.0|  
| GOOG| Charlie|120.0|  
| GOOG| Frank|340.0|  
| MSFT| Tina|600.0|  
| MSFT| Amy|124.0|  
| MSFT| Vanessa|243.0|  
| FB| Carl|870.0|  
| FB| Sarah|350.0|  
| APPL| John|250.0|  
| APPL| Linda|130.0|  
| APPL| Mike|750.0|  
| APPL| Chris|350.0|  
+---+---+
```

```
# Max  
df.groupBy('Company').max().show()
```

```
+-----+  
|Company|max(Sales)|  
+-----+  
| APPL| 750.0|  
| GOOG| 340.0|  
| FB| 870.0|  
| MSFT| 600.0|  
+-----+
```

# ML Pipelines

```
from pyspark.ml.linalg import Vectors
from pyspark.ml.classification import LogisticRegression

# Prepare training data from a list of (label, features) tuples.
training = spark.createDataFrame([
    (1.0, Vectors.dense([0.0, 1.1, 0.1])),
    (0.0, Vectors.dense([2.0, 1.0, -1.0])),
    (0.0, Vectors.dense([2.0, 1.3, 1.0])),
    (1.0, Vectors.dense([0.0, 1.2, -0.5]))], ["label", "features"])

# Create a LogisticRegression instance. This instance is an Estimator.
lr = LogisticRegression(maxIter=10, regParam=0.01)
# Print out the parameters, documentation, and any default values.
print("LogisticRegression parameters:\n" + lr.explainParams() + "\n")

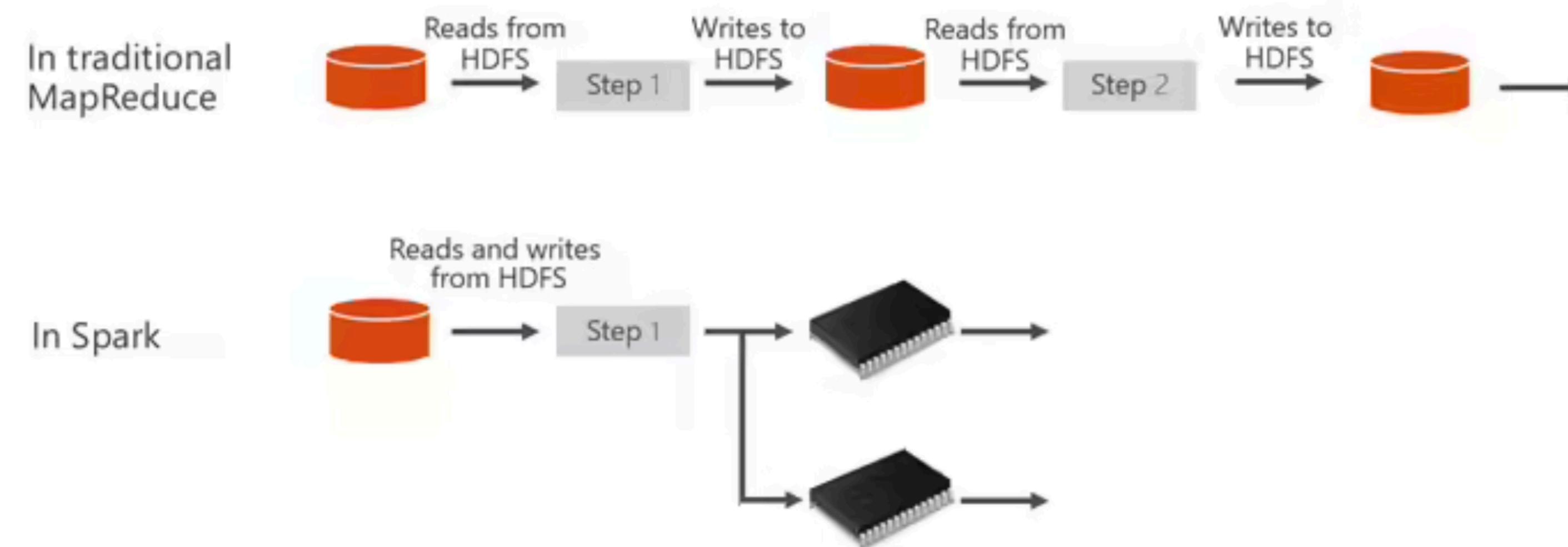
# Learn a LogisticRegression model. This uses the parameters stored in lr.
model1 = lr.fit(training)
```

# Альтернатива Hadoop MapReduce

## Apache Spark

Преимущества Spark:

1. Простота использования – высокогореневый API позволяет концентрироваться на предметной стороне вычислений
2. Высокая скорость работы – возможно создавать интерактивные приложения и использовать сложные алгоритмы
3. Обобщенность – позволяет объединять разнотипные вычисления (запускать SQL-запросы, обрабатывать текст, реализовать ML-алгоритмы)



# Что такое Apache Spark?

Apache Spark – универсальная и высокопроизводительная кластерная вычислительная платформа

Чем Spark лучше MapReduce:

1. Производительность (вычисления в памяти)
2. Удобные абстракции над данными
3. Поддержка нескольких языков программирования: Scala, Java, Python
4. Поддержка большего типа данных
5. Поддержка разных источников данных
  1. Файловые системы: HDFS, Linux File System
  2. Базы данных: HBase, Cassandra, S3
6. Является Open-Source проектом
7. Потоковая обработка (streaming processing)

# Компоненты Apache Spark

## Apache Spark Core

Spark SQL

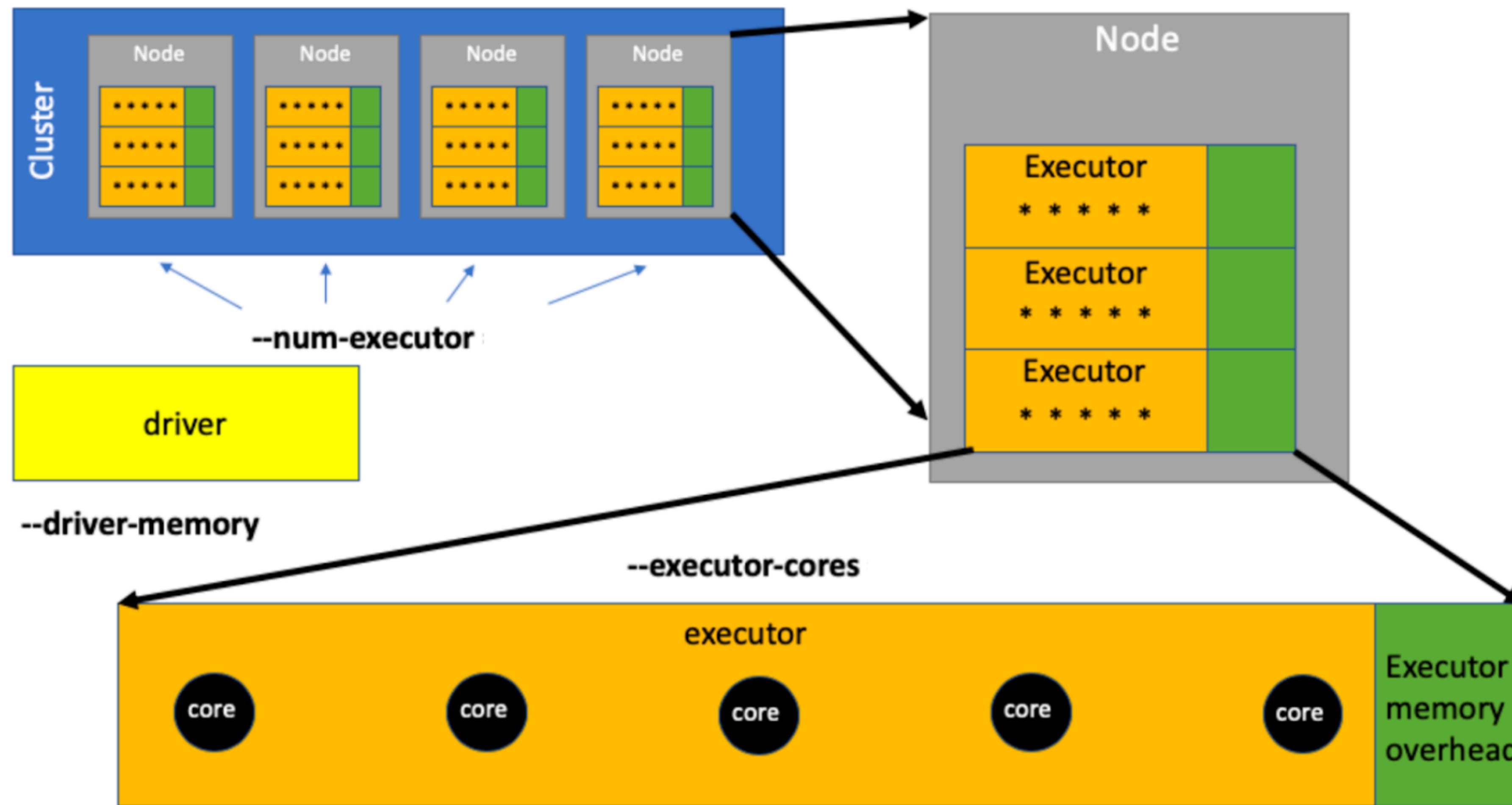
Spark  
Streaming

MLlib  
(Machine  
Learning)

GraphX  
(Graph)

# Apache Spark

## Driver & Executor

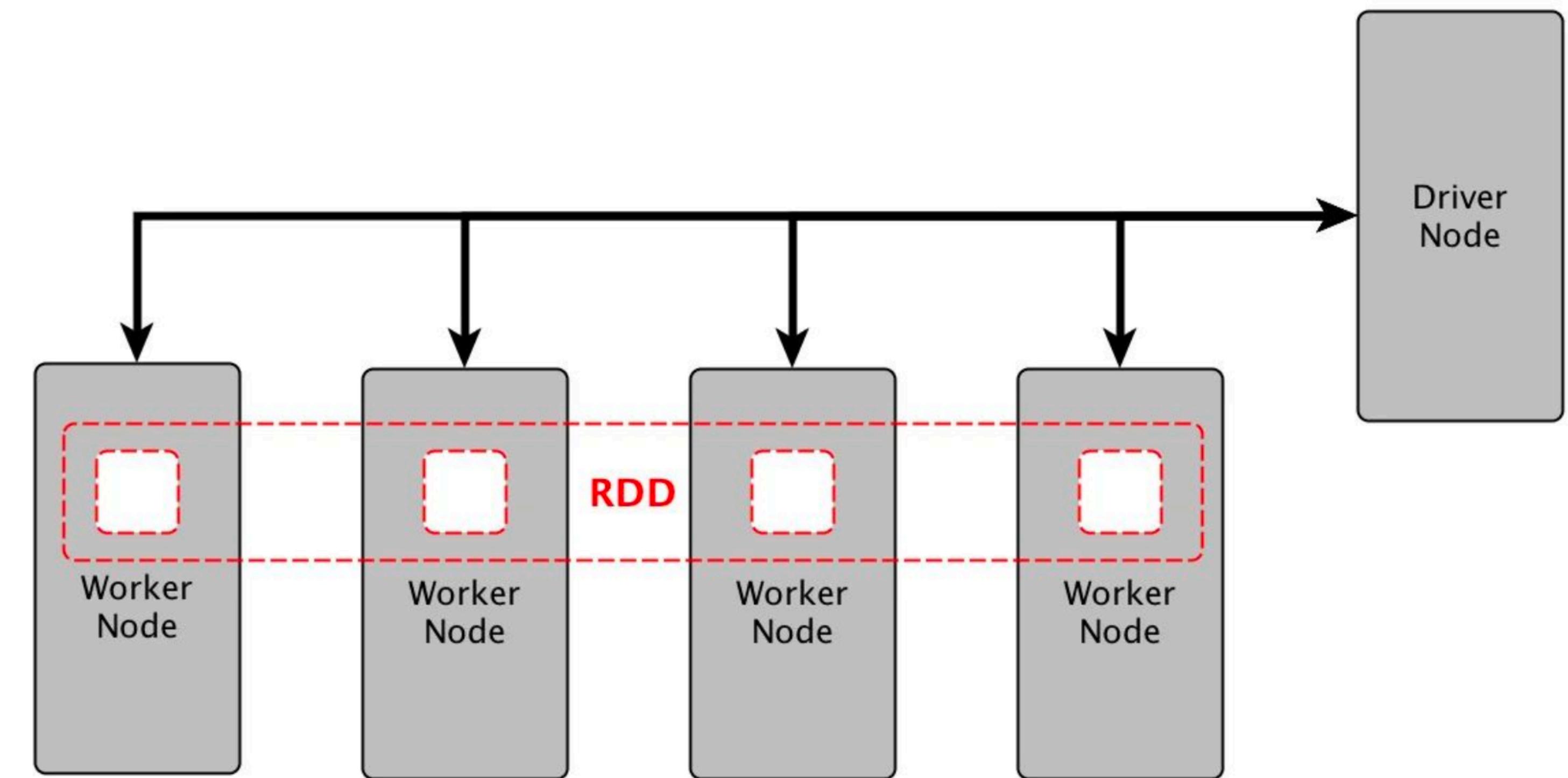


# RDD

**RDD (Resilient Distributed Dataset)** — это фундаментальная структура данных Spark, которая представляет собой неизменяемый набор данных, который вычисляются и располагается на разных узлах кластера. Каждый набор данных в Spark RDD логически разделен на множество серверов, чтобы их можно было вычислить на разных узлах кластера.

```
data = [1, 2, 3]
data_rdd = sc.parallelize(data)

data_rdd.count()
# 3
```



# Основные команды RDD

`sc.textFile` - сформировать RDD из текстового файла

`take(5)` - посмотреть первые 5 элементов результата

`collect` - возвращает результат вычислений в память (!)

`count()` - подсчет числа строк

`map` - построчная обработка (маппер)

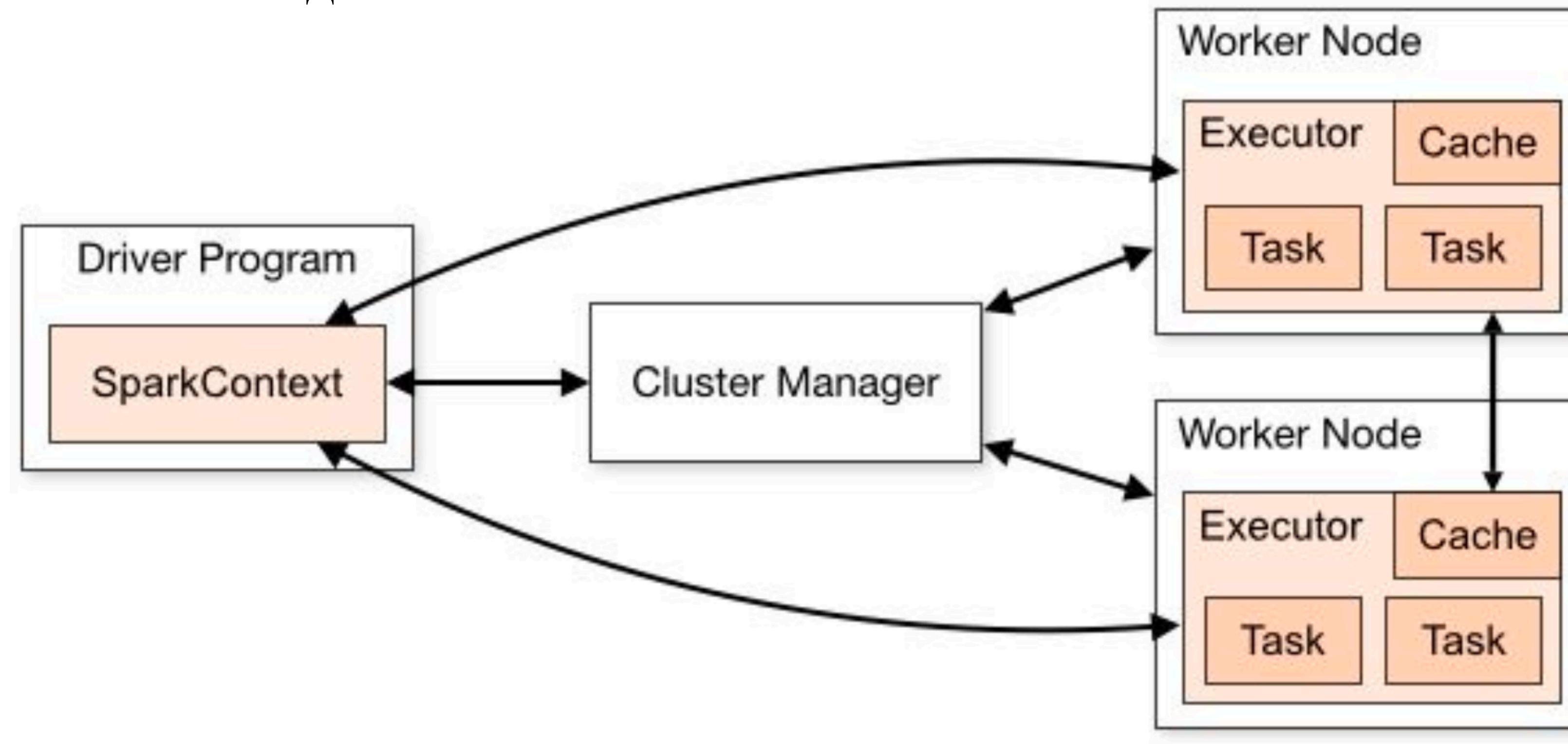
`flatMap` - разворачивание списка в столбец

`filter` - фильтрация строк функцией

`reduce` - попарные действия с элементами

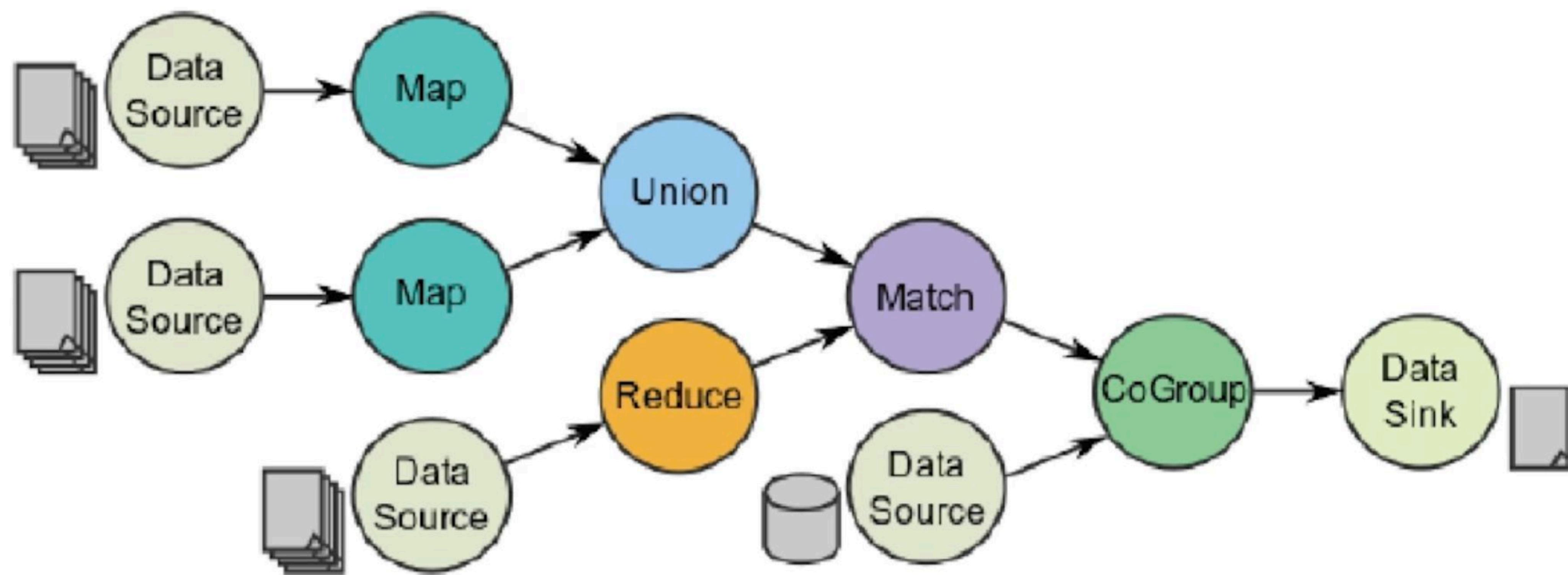
# Spark Context

**SparkContext** — это точка входа для всех операций **Spark** и средство, с помощью которого приложение подключается к ресурсам кластера Spark. Через этот специальный объект Spark происходит конфигурирование ресурсов кластера, обращение к данным и постановка различных задач на исполнение.



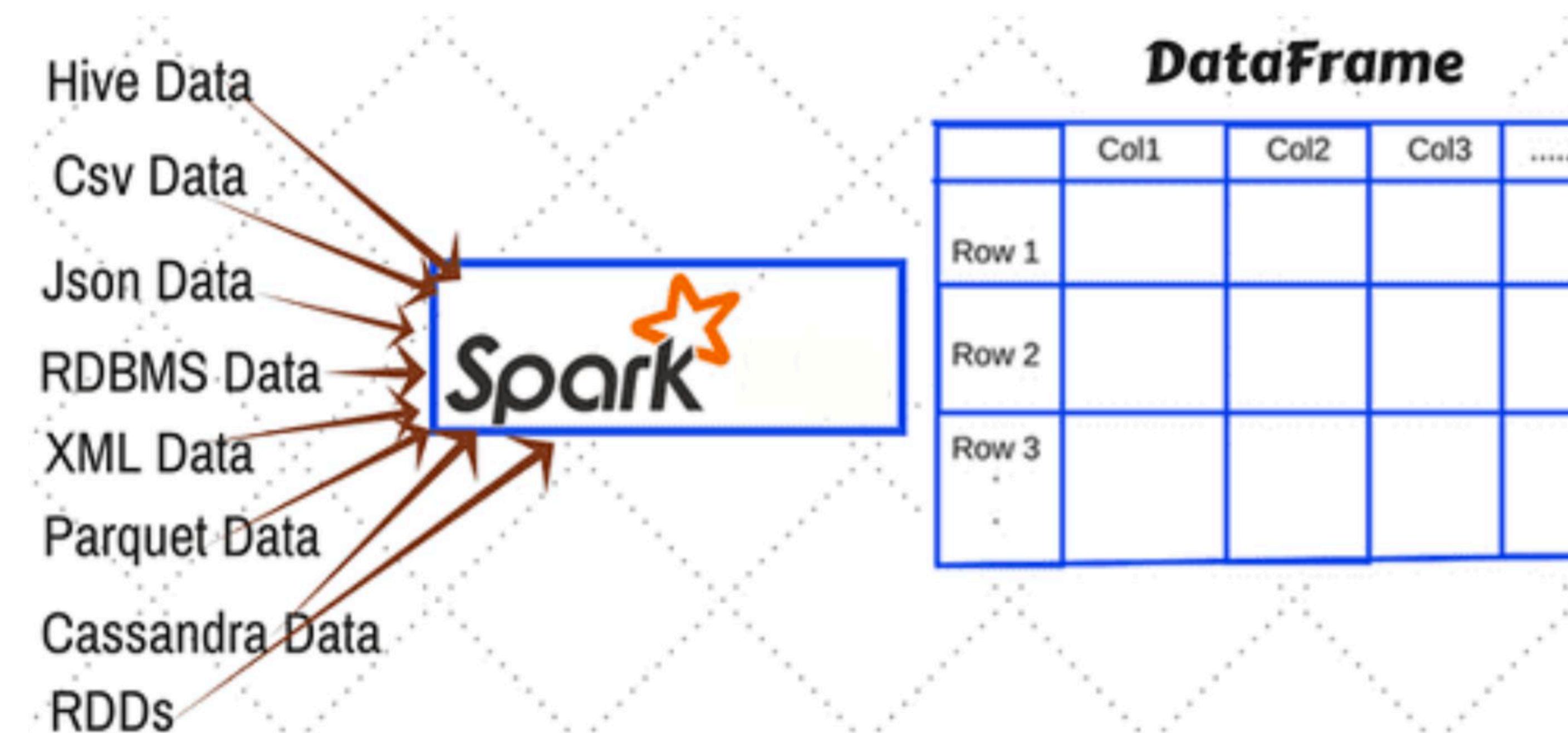
# DAG

**DAG (Направленный ациклический граф)** — это набор вершин и ребер, где вершины представляют RDD, а ребра представляют операцию, которая будет применяться к RDD. В Spark DAG каждое ребро направляет от более раннего к более позднему в последовательности.



# Spark DataFrame

**Spark DataFrame** — это набор данных, организованный в виде таблицы. Концептуально он эквивалентен таблице базе данных или фрейму данных в Python, но с более обширной внутренней оптимизацией. Spark DataFrames могут быть созданы из различных источников, таких как файлы структурированных данных, таблицы в Hive, внешние базы данных или существующие RDD.



# Операции над данными

В Spark поддерживается два вида операторов, которые работают над RDD:

- **Transformations** - операторы, которые создают новые RDD на основе имеющихся, при этом самого исполнения операции не происходит
- **Actions** - операторы, которые запускают вычисления и возвращают результат работы в программу или сохраняют его на диск

# Transformations

<b>map(func)</b>	Return a new distributed dataset formed by passing each element of the source through a function <i>func</i> .	<b>groupByKey([numPartitions])</b>	When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs.  <b>Note:</b> If you are grouping in order to perform an aggregation (such as a sum or average) over each key, using reduceByKey or aggregateByKey will yield much better performance.
<b>filter(func)</b>	Return a new dataset formed by selecting those elements of the source on which <i>func</i> returns true.		  <b>Note:</b> By default, the level of parallelism in the output depends on the number of partitions of the parent RDD. You can pass an optional numPartitions argument to set a different number of tasks.
<b>mapPartitions(func)</b>	Similar to map, but runs separately on each partition (block) of the RDD, so <i>func</i> must be of type Iterator<T> => Iterator<U> when running on an RDD of type T.		
<b>sortByKey([ascending], [numPartitions])</b>	When called on a dataset of (K, V) pairs where K implements Ordered, returns a dataset of (K, V) pairs sorted by keys in ascending or descending order, as specified in the boolean <i>ascending</i> argument.	<b>reduceByKey(func, [numPartitions])</b>	When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function <i>func</i> , which must be of type (V,V) => V. Like in groupByKey, the number of reduce tasks is configurable through an optional second argument.
<b>repartition(numPartitions)</b>	Reshuffle the data in the RDD randomly to create either more or fewer partitions and balance it across them. This always shuffles all data over the network.		

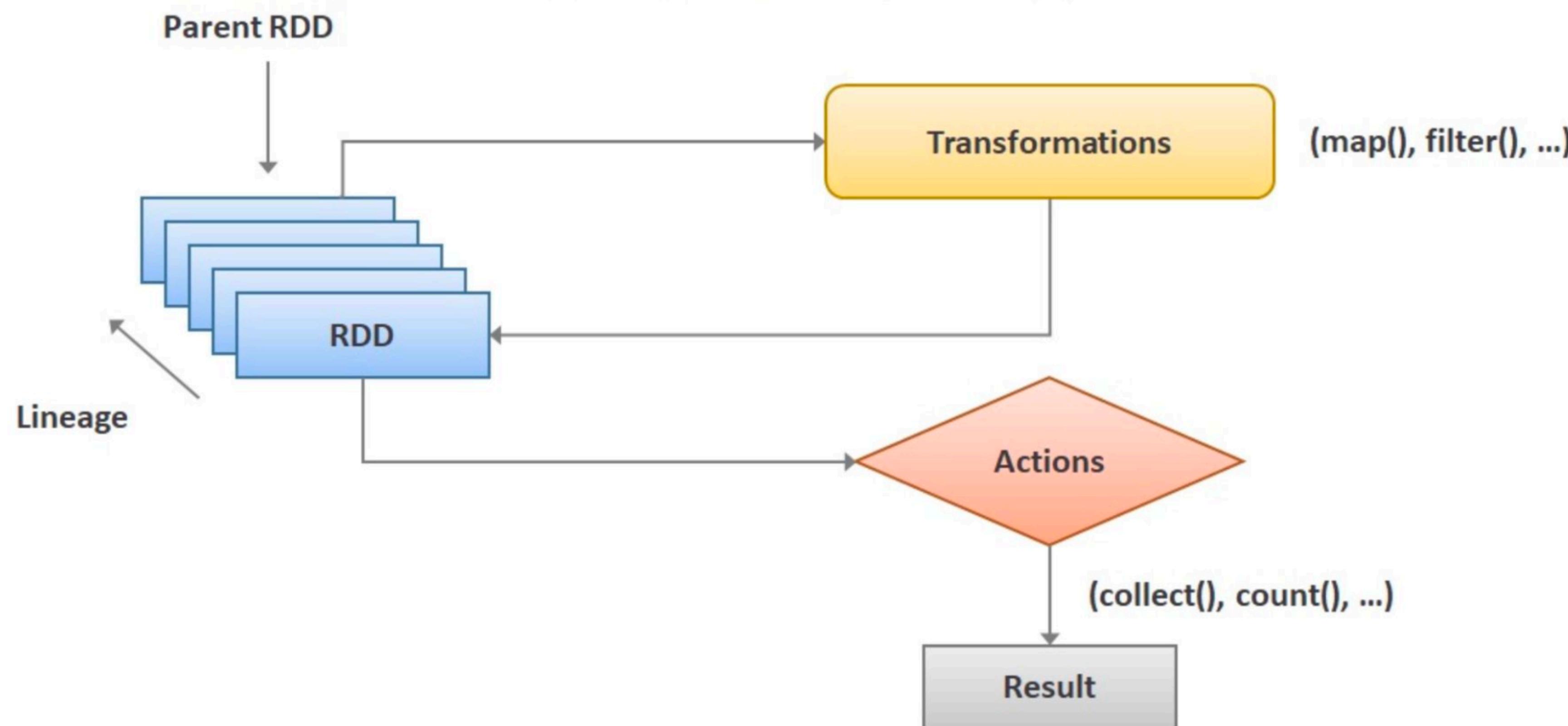
# Actions

<b>reduce(<i>func</i>)</b>	Aggregate the elements of the dataset using a function <i>func</i> (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.
<b>collect()</b>	Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.
<b>count()</b>	Return the number of elements in the dataset.
<b>first()</b>	Return the first element of the dataset (similar to take(1)).
<b>take(<i>n</i>)</b>	Return an array with the first <i>n</i> elements of the dataset.
<b>countByKey()</b>	Only available on RDDs of type (K, V). Returns a hashmap of (K, Int) pairs with the count of each key.

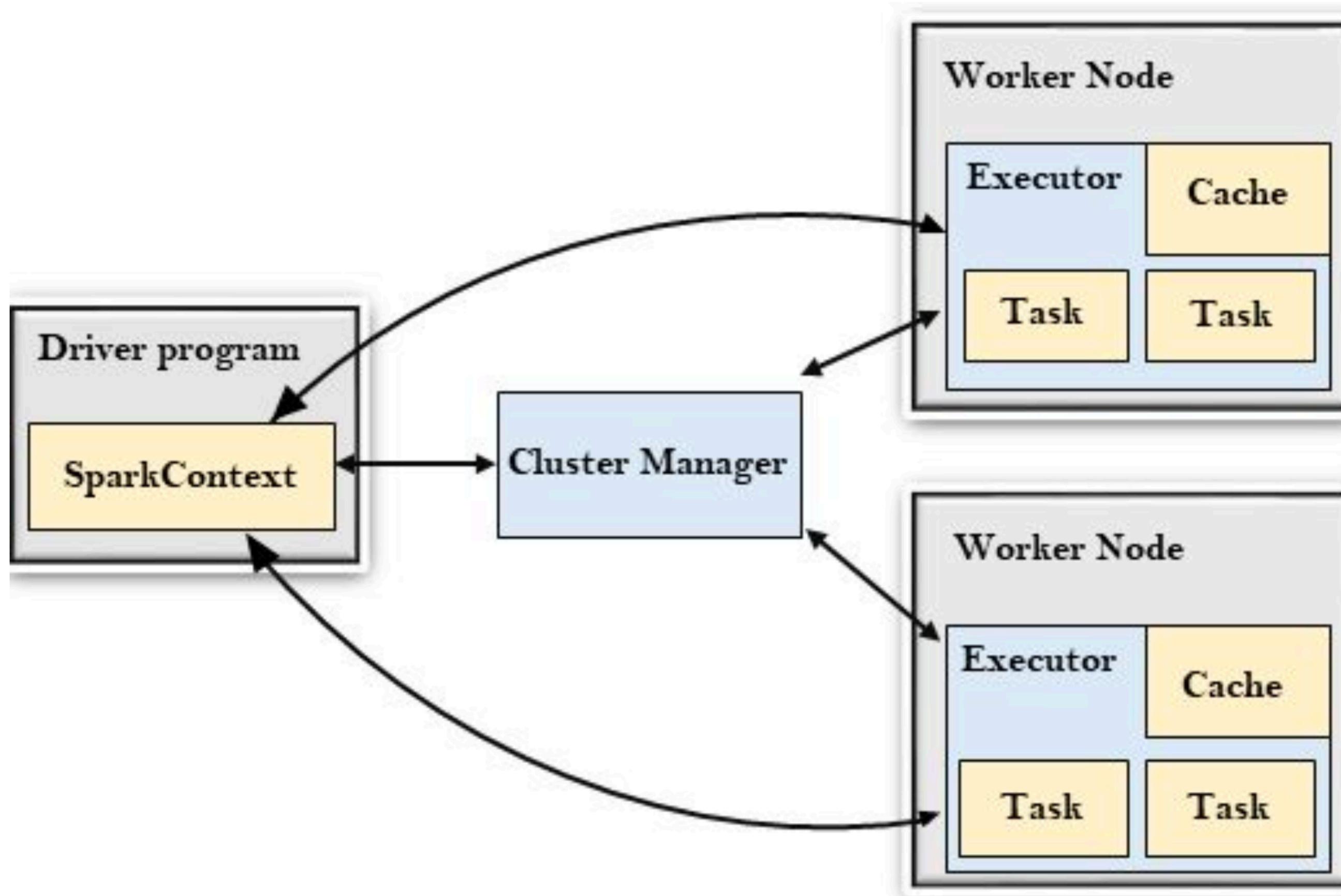
# Жизненный цикл RDD



*Spark RDD (Unstructured) Operations*



# Устройство Spark-кластера



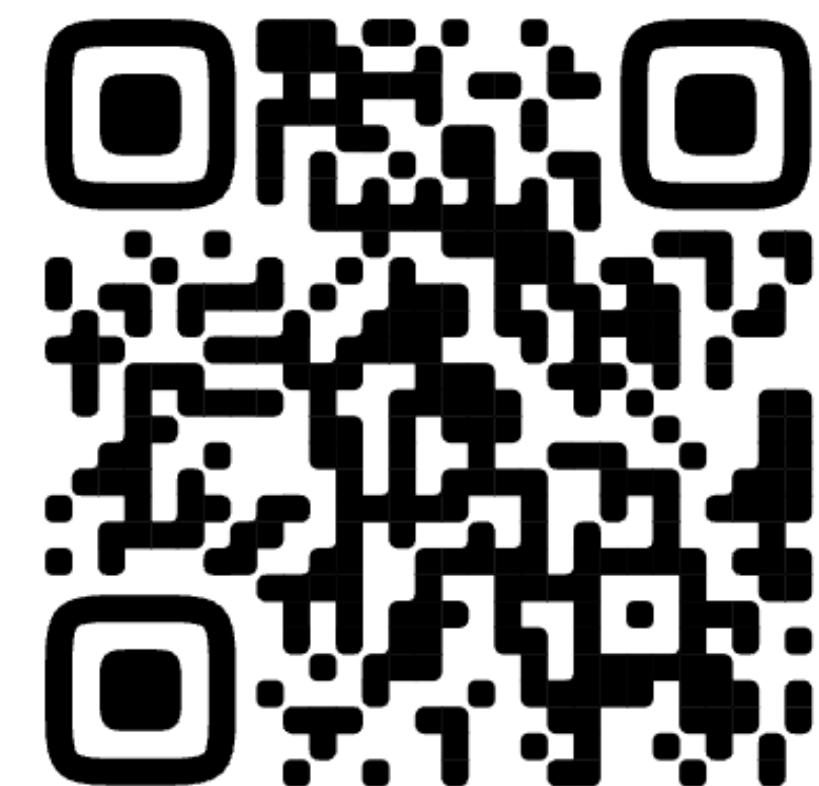
- **Worker Node** - компоненты системы Spark, которые выполняют фактическое действие над данными;
- **Spark Driver** - компонент, ответственный за планирование вычислений и подготовку задач для Worker'ов;
- **Cluster Manager** - компонент, который координирует обмен данными между узлами;

# Практика RDD

Репозиторий на GitHub с python-ноутбуком и файлом с данными  
Рекомендуется его загрузить в Google Collab (вместе с данными)  
и запускать от туда

Дано: файл google\_queries.csv с запросами в google  
В каждой строке указано запрос, количество за день, день запроса в виде «карты,4,2025-01-19» и тд

<https://clck.ru/3Gad5a>



# Data locality

В Spark реализованы механизмы, поддерживающие локальность данных. Под локальностью данных понимается то, что вычисления производятся на том узле, где располагаются необходимые данные. Это позволяет уменьшить нагрузку на сеть и увеличить производительность системы.

Достигается это через передачу Spark-кода на узлы, на которых располагаются данные, а не наоборот.

# Литература для доп погружения в тему

O'REILLY®

## ВЫСОКО- НАГРУЖЕННЫЕ ПРИЛОЖЕНИЯ

Программирование  
масштабирование  
поддержка



ДИТЕР®

Мартин Клеппман

O'REILLY®

## ЭВОЛЮЦИОННАЯ АРХИТЕКТУРА

ПОДДЕРЖКА НЕПРЕРВНЫХ ИЗМЕНЕНИЙ



Нил Форд, Ребекка Парсонс, Патрик Кью

ДИТЕР®

O'REILLY®

## Data Governance The Definitive Guide

People, Processes, and Tools to Operationalize  
Data Trustworthiness



Evren Eryurek, Uri Gilad,  
Valliappa Lakshmanan,  
Anita Kibunguchy-Grant  
& Jessi Ashdown

**Спасибо за внимание!**