Среда Spark

Spark RDD / Spark SQL

Лекция 2

Толоконов Илья

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.

Кратко про прошлую лекцию

3 основных вопроса в Big Data

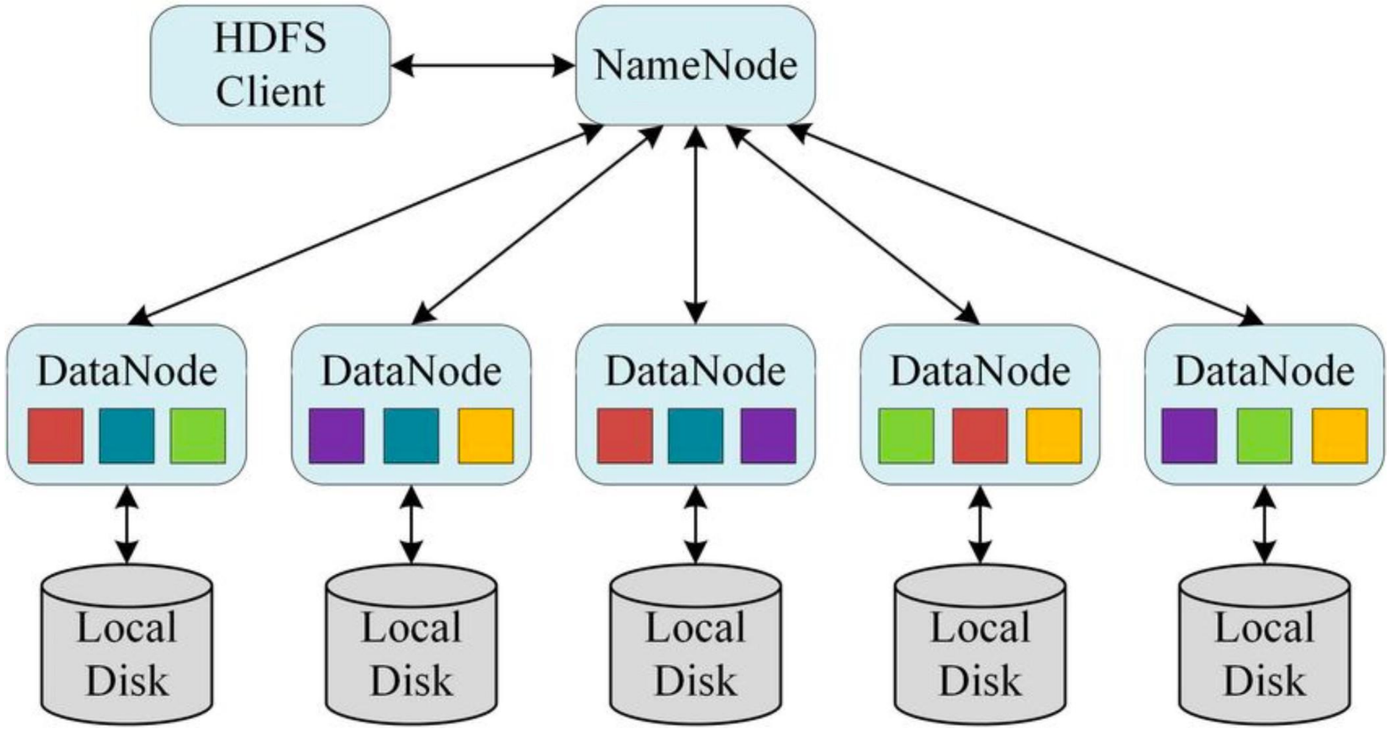
1. Как хранить большие данные? (Storage Layer)
2. Как обрабатывать большие данные? (Processing layer)
3. Как управлять ресурсами кластера? (Resource management layer)

Кратко про прошлую лекцию

Как хранить большие данные?

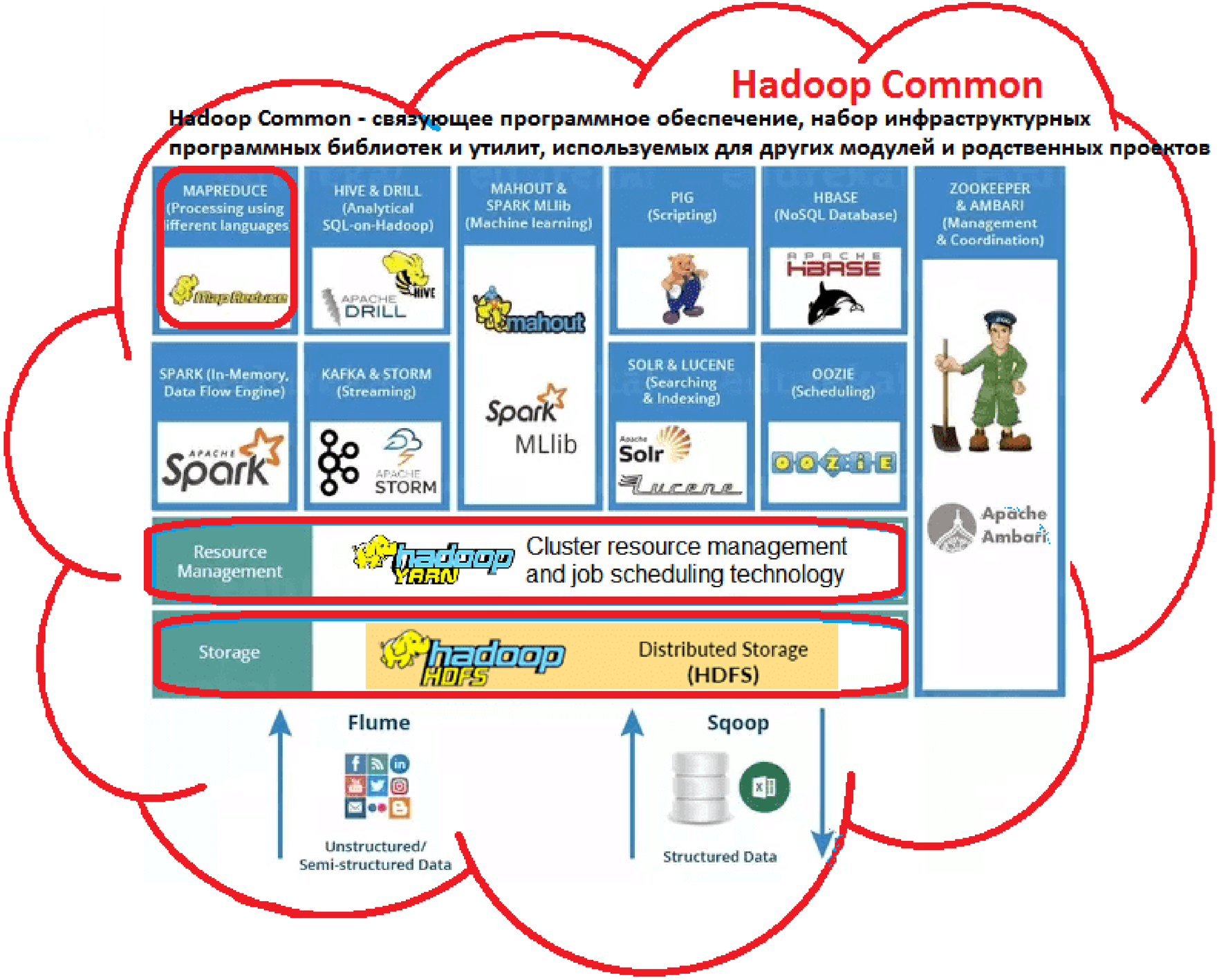
HDFS – распределенная файловая система, разработанная для работы с большими объемами данных в кластерах

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



Hadoop Common

Hadoop Common - связующее программное обеспечение, набор инфраструктурных программных библиотек и утилит, используемых для других модулей и родственных проектов

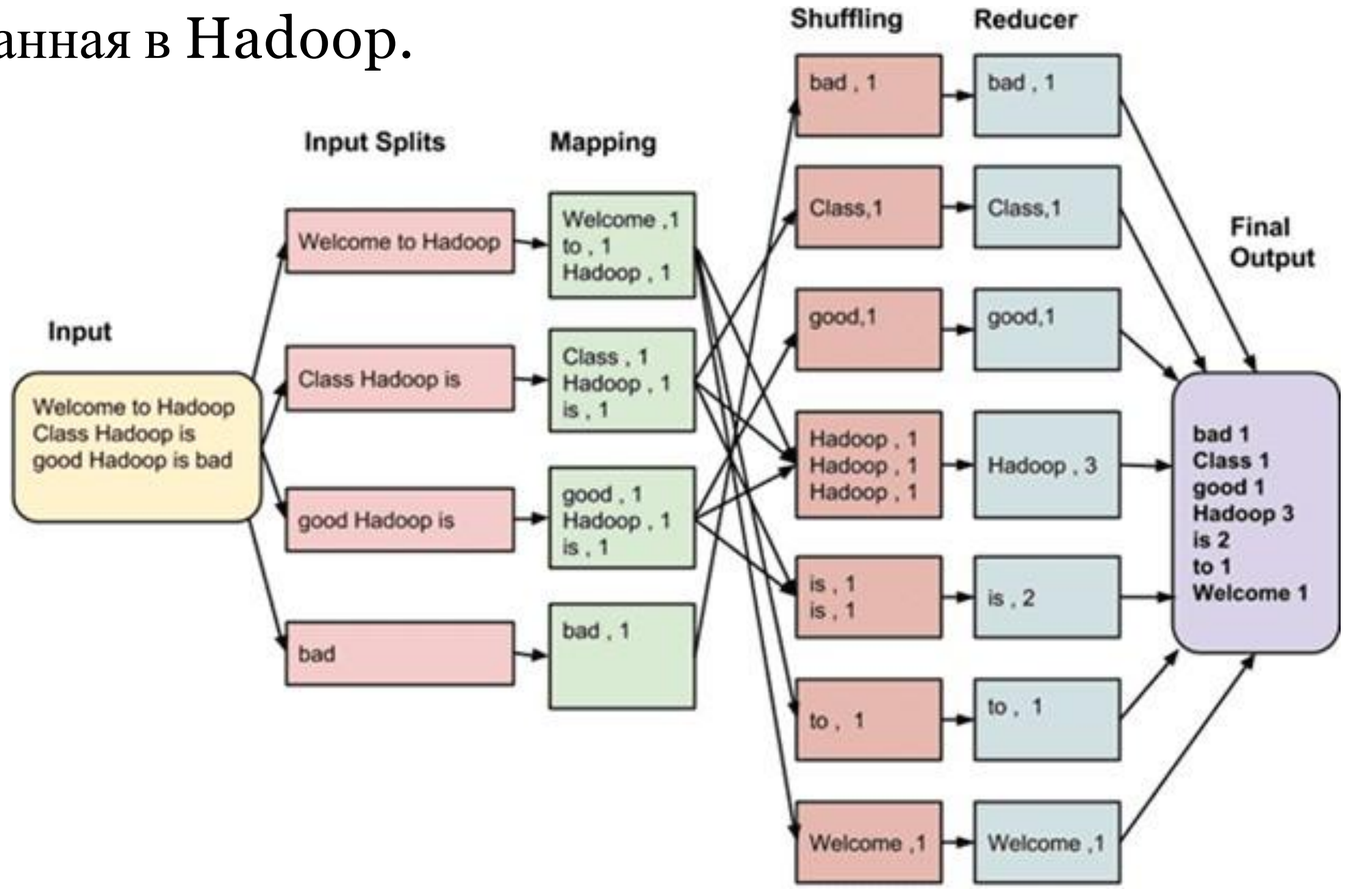


Кратко про прошлую лекцию

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

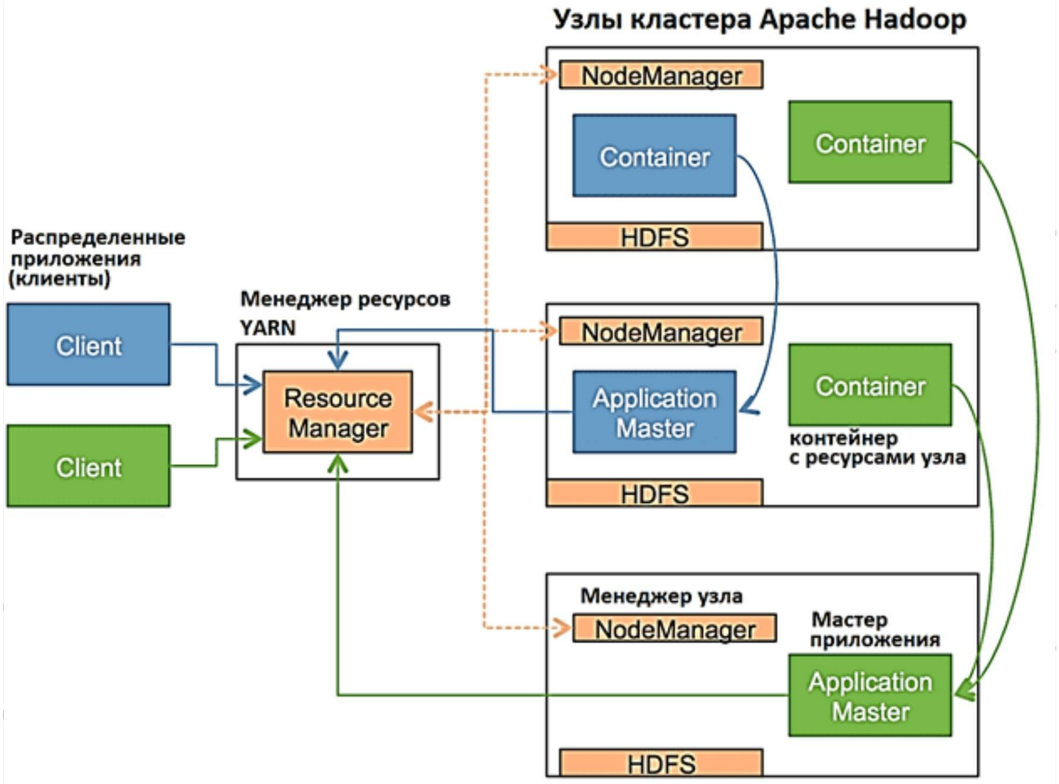


Кратко про прошлую лекцию

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

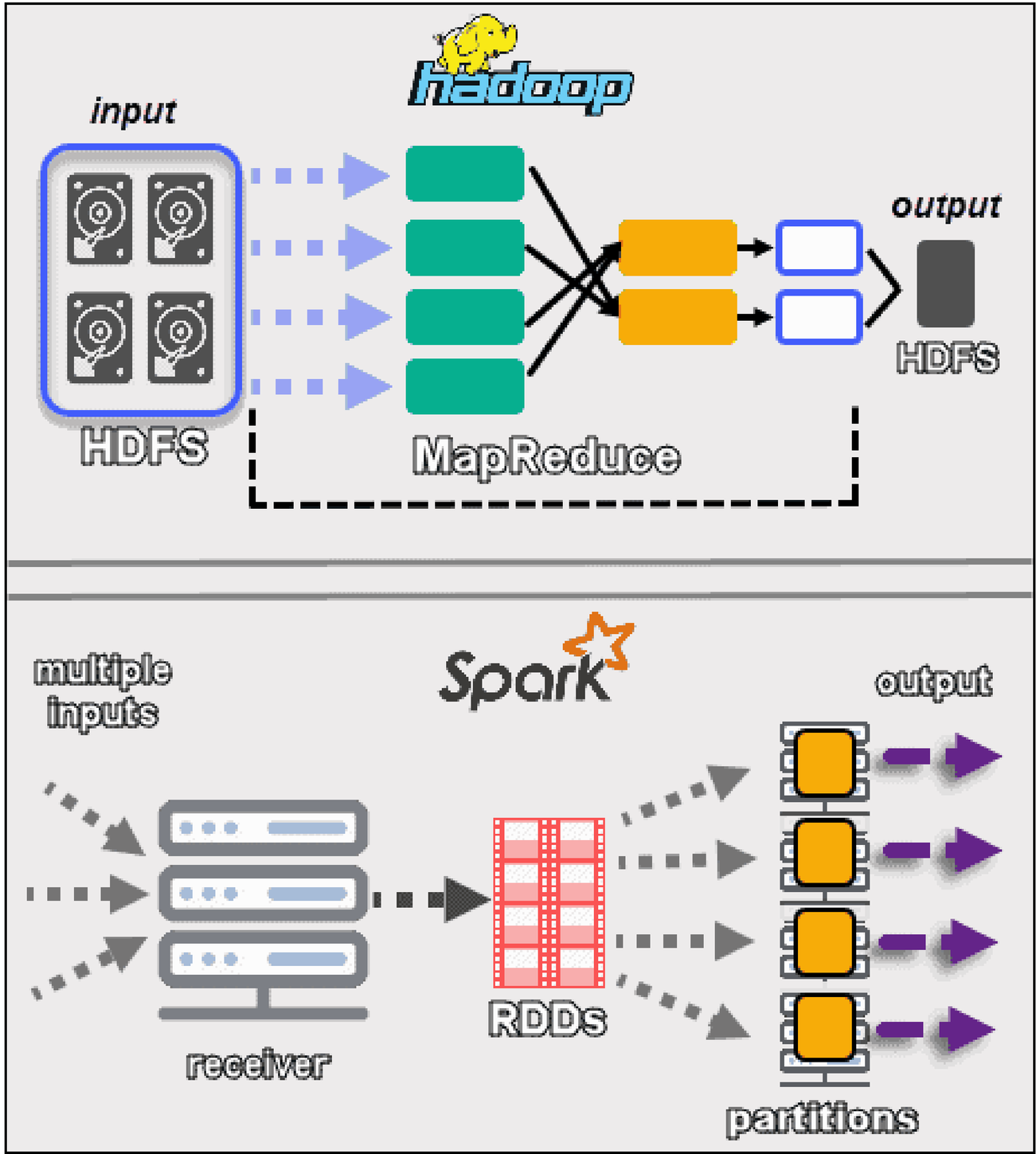


MapReduce vs Spark

1. Производительность в 100 раз выше

MapReduce записывает данные на диске между шагами

Spark хранит промежуточные вычисления в оперативной памяти (in-memory) + встроенная оптимизация

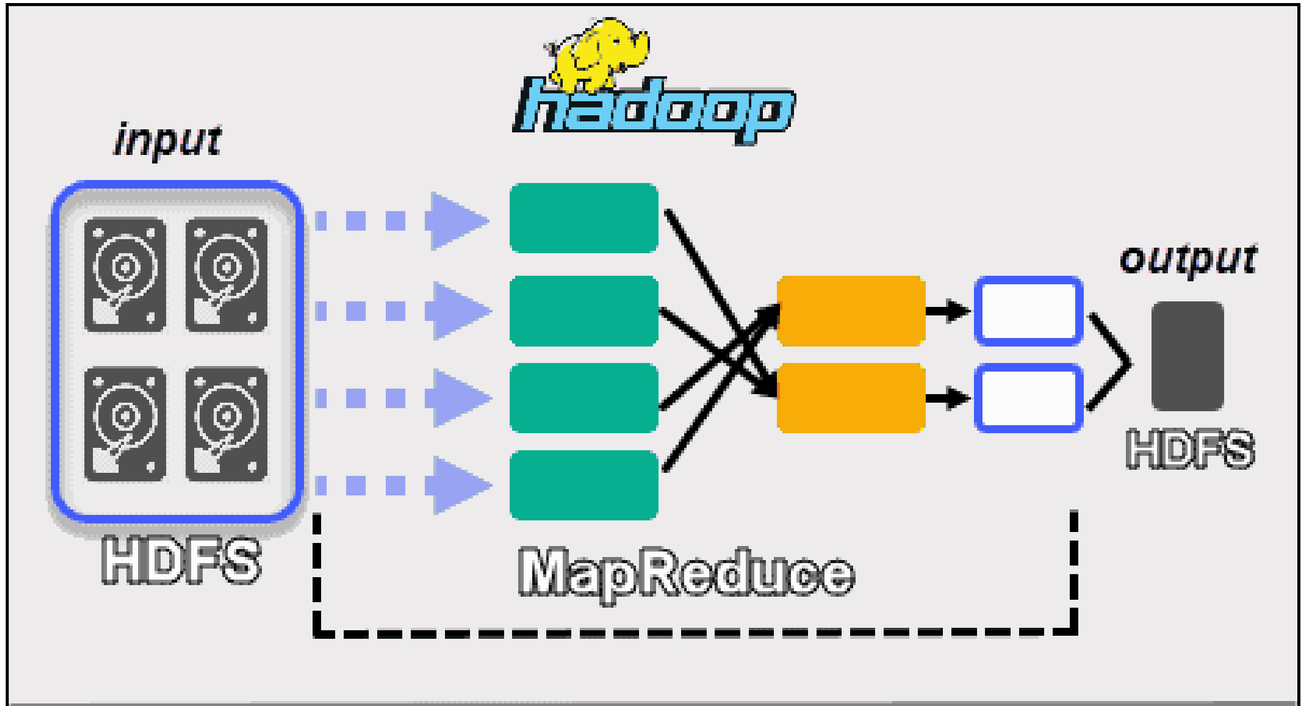


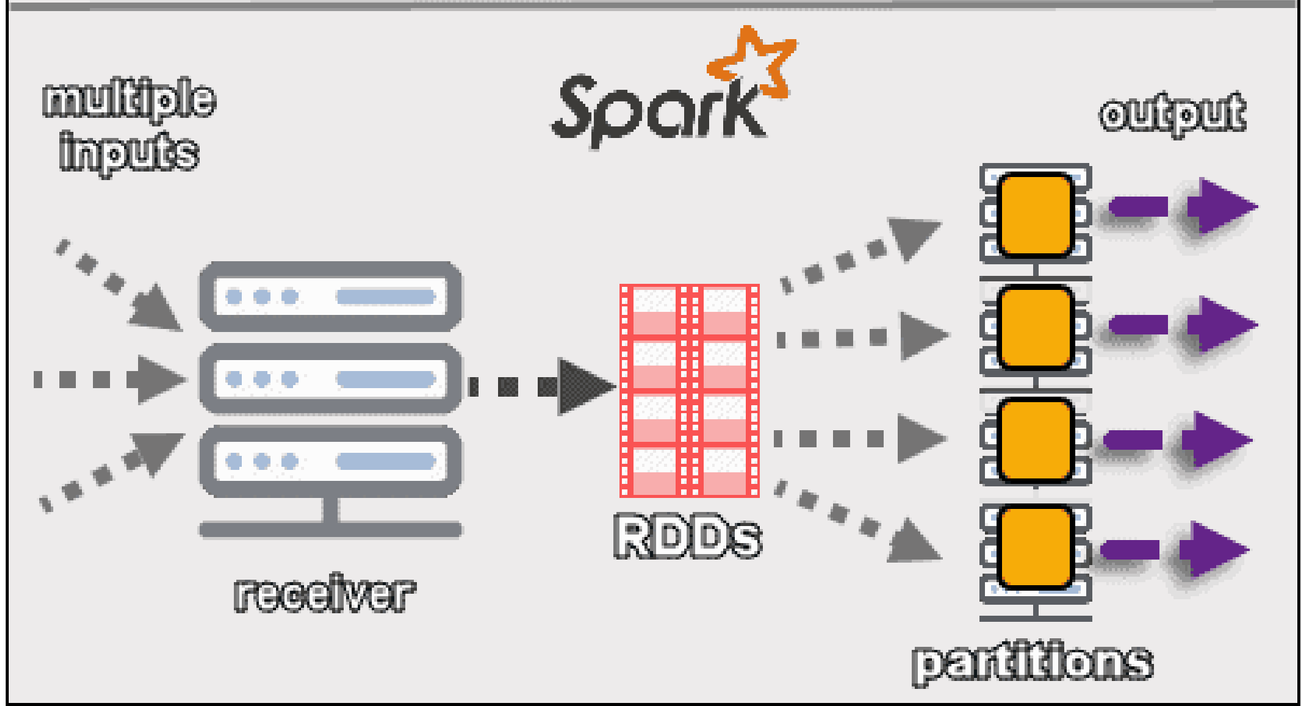
MapReduce vs Spark

2. Гибкость

MapReduce требует более низкоуровневого программирования

Spark API предоставляет использование ЯП: Scala, Python, Java, SQL

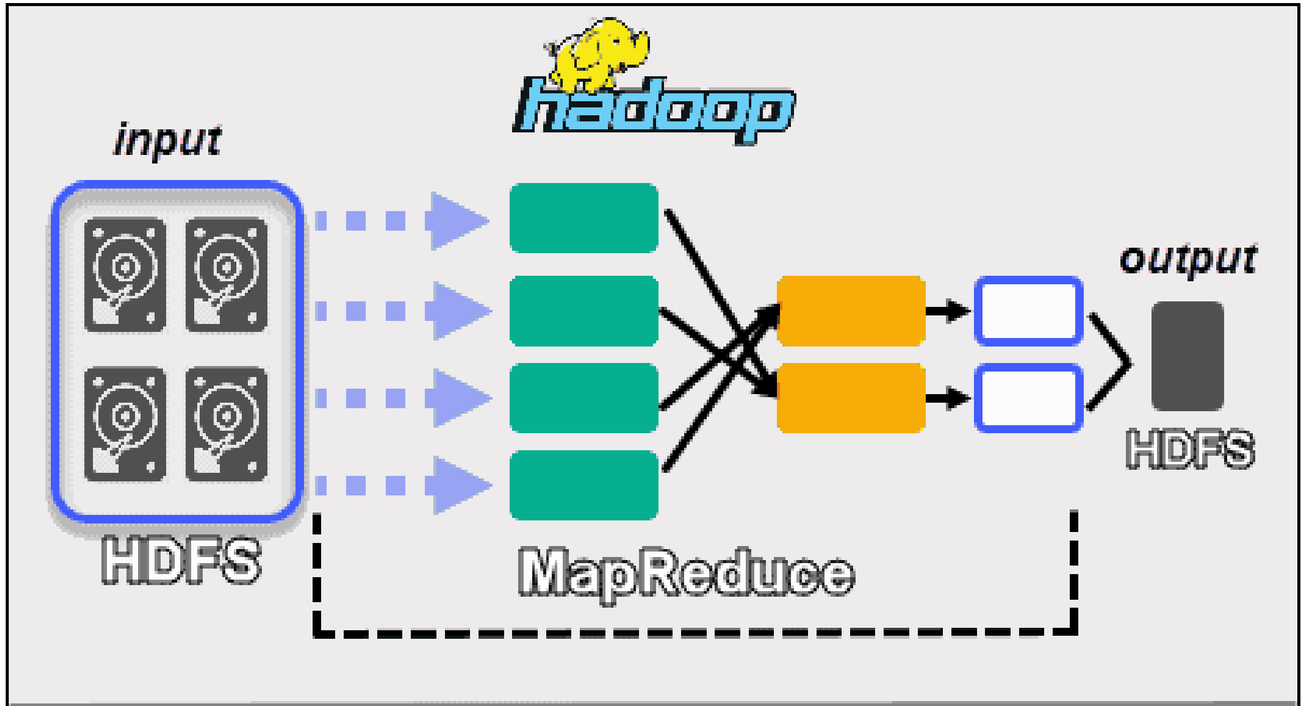


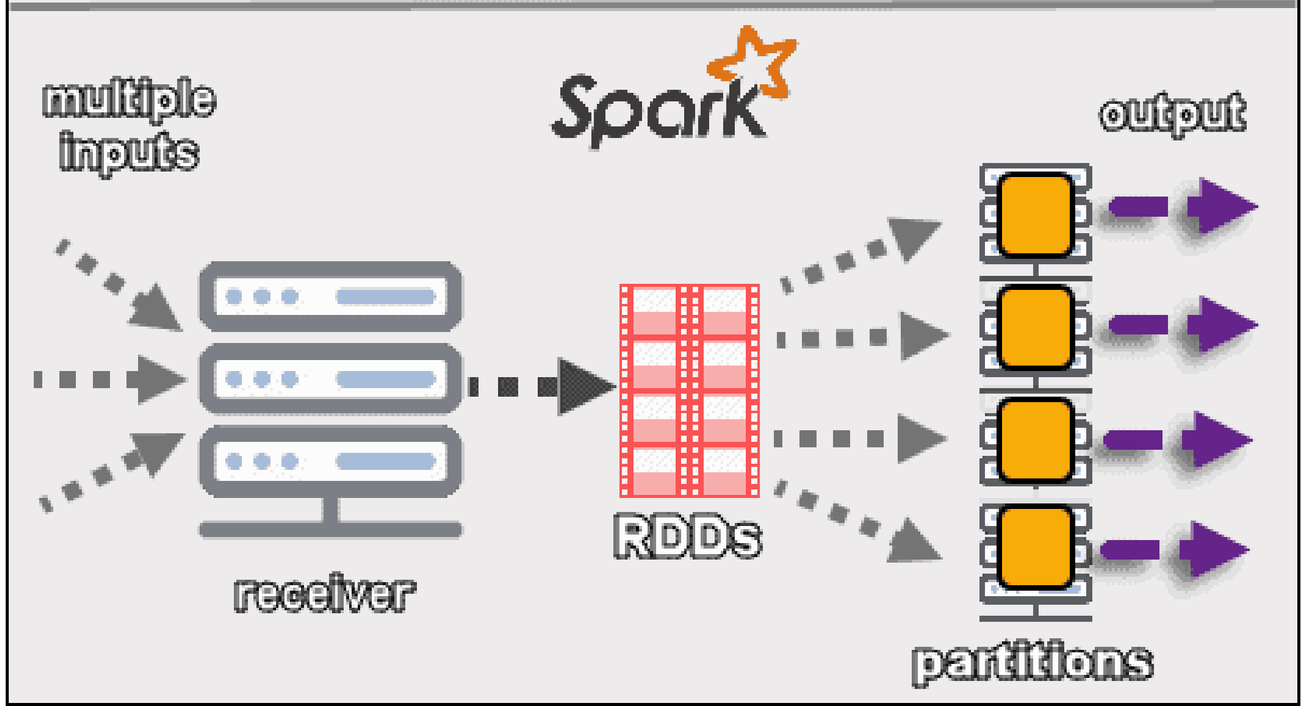


MapReduce vs Spark

3. Удобство разработки

Spark предлагает абстракцию RDD и DataFrame для более высокоуровневой трансформации данных
В **Spark** можно запускать задачи через интерфейсы: Spark Shell или Jupyter Notebooks

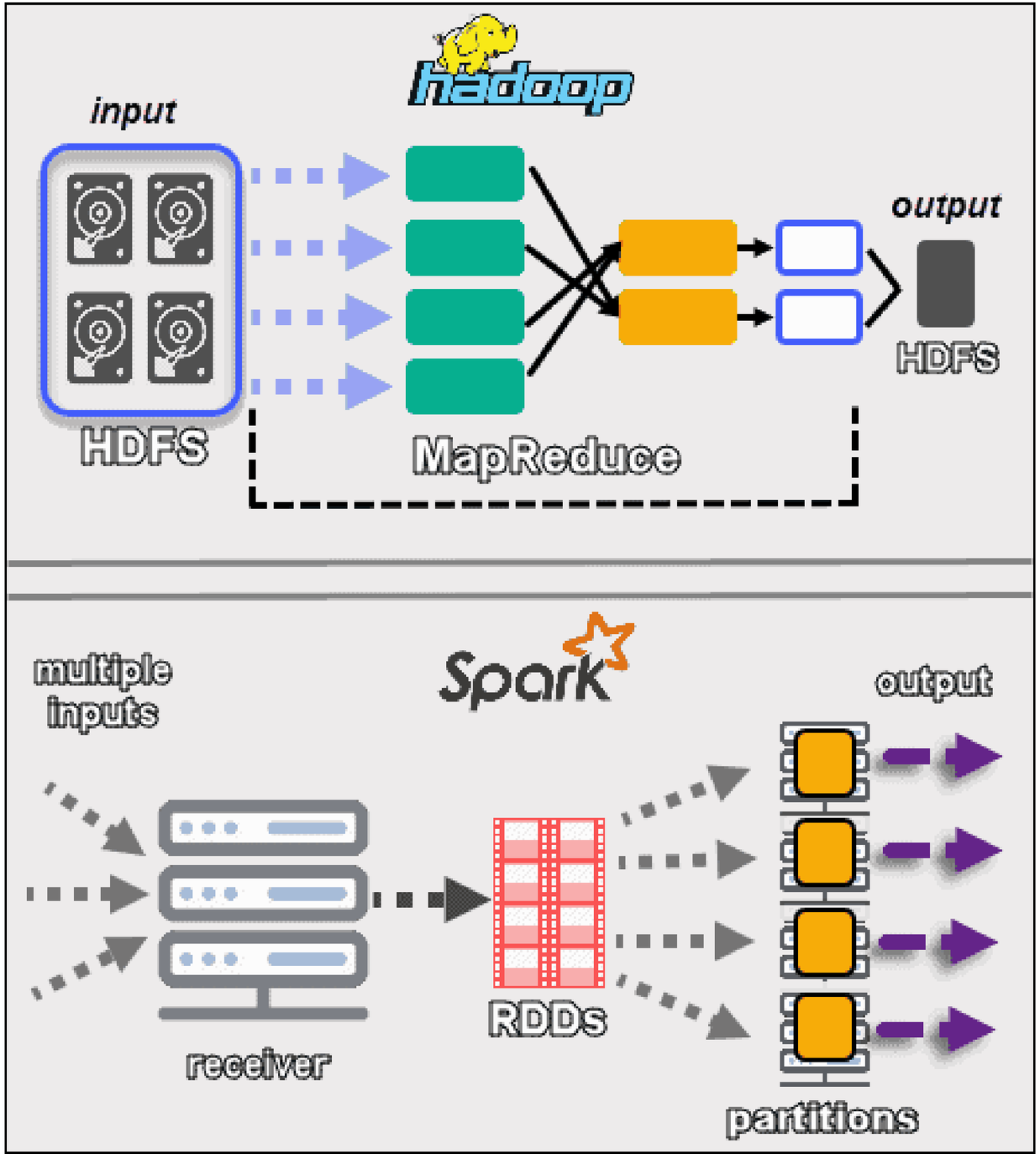




MapReduce vs Spark

4. Экосистема

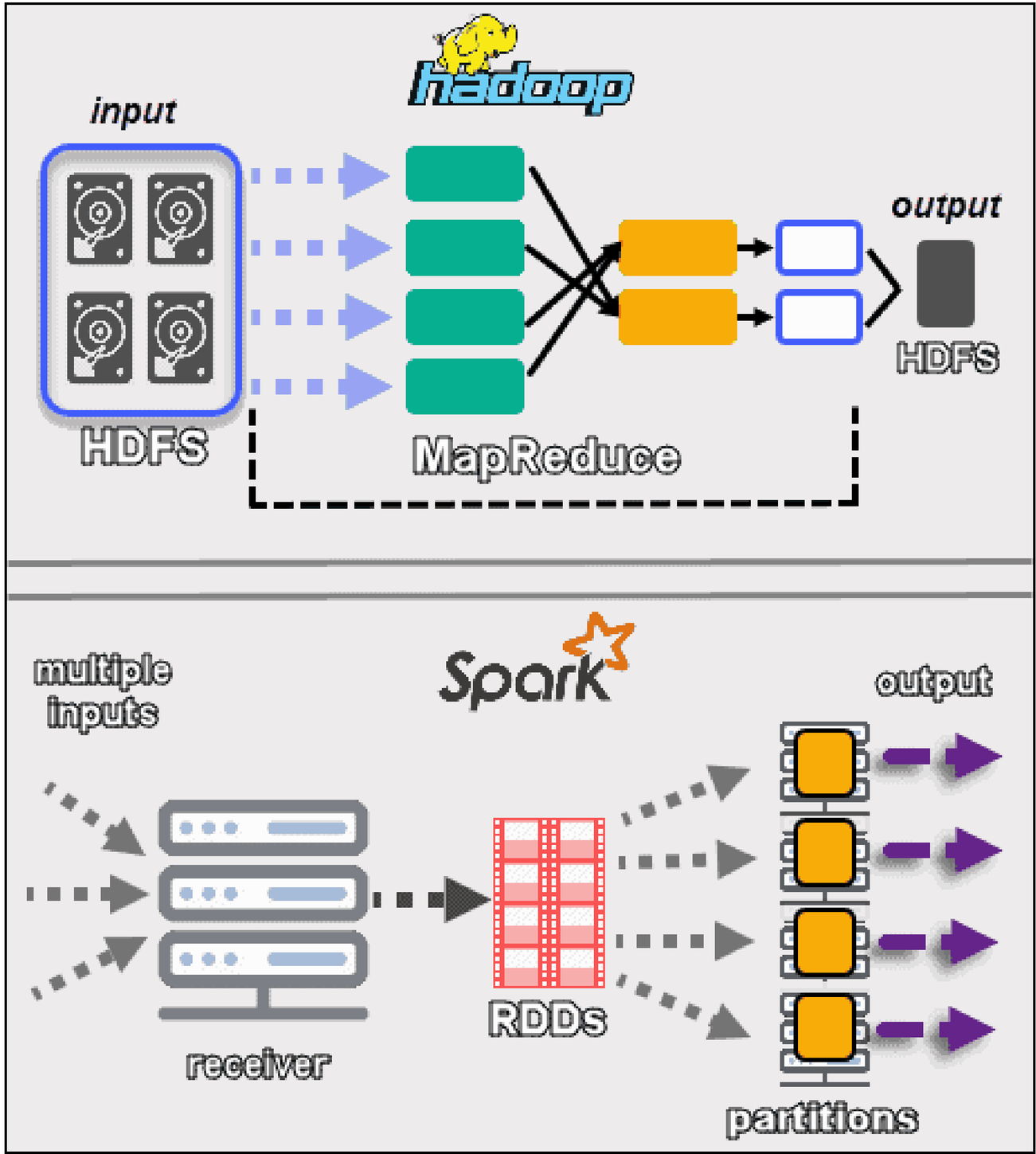
Spark SQL: для работы с данными в стиле реляционных БД.
MLlib: для машинного обучения
GraphX: для работы с графами
Spark Structured Streaming: для более продвинутой потоковой обработки
Web UI: Spark имеет встроенный веб-интерфейс, предоставляющий инфо о текущих и завершенных заданиях, стадиях выполнения, использовании ресурсов.



MapReduce vs Spark

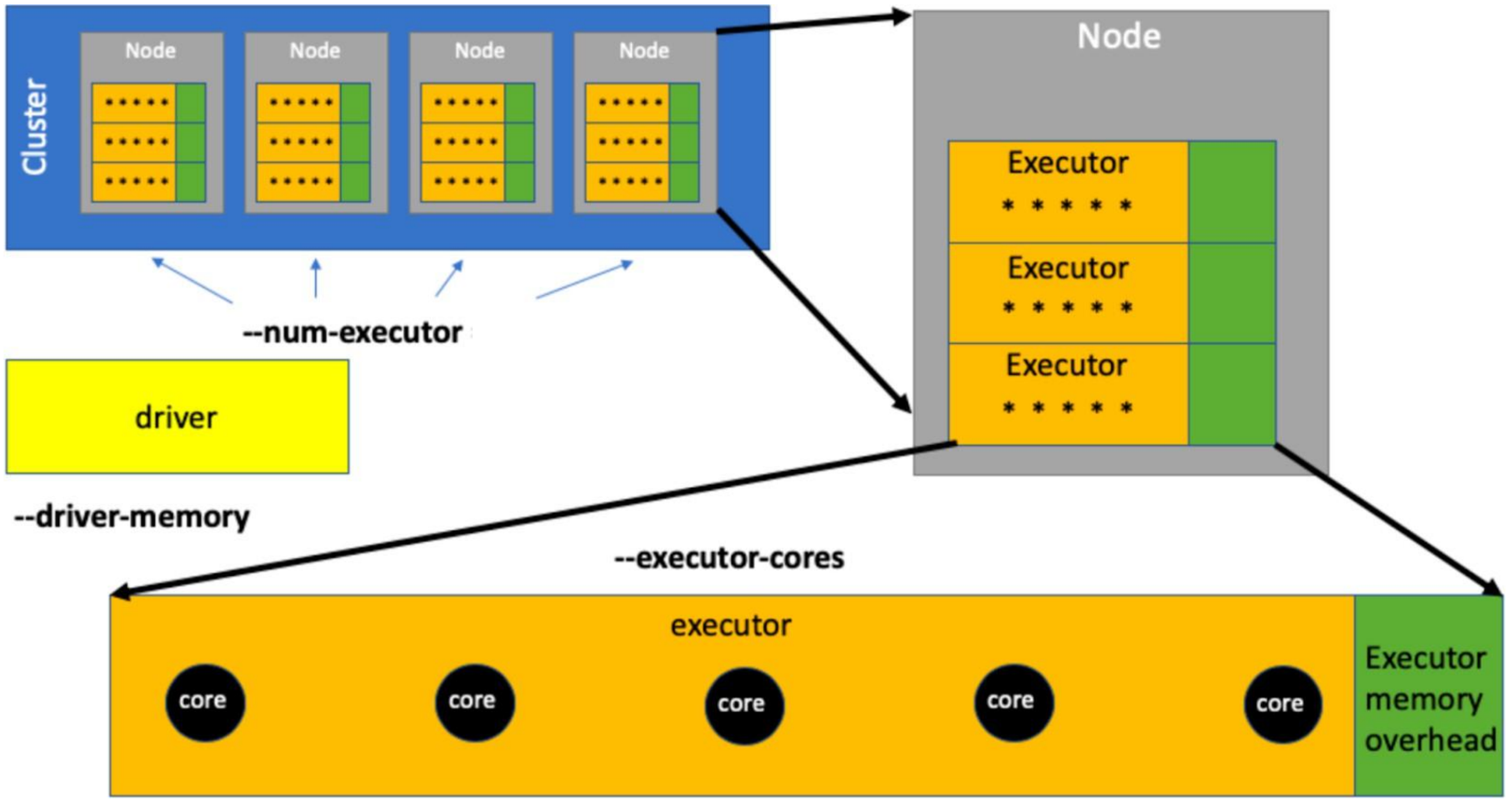
5. Spark является open-source проектом

Благодаря этому была достигнута совместимость как с любыми файловыми системами, так и разными базами данных



Apache Spark

Driver & Executor



Apache Spark

Основные определения:

Application – приложение, поверх Spark API, которое состоит из **Driver** и **Executors** на кластере

SparkSession – точка входа, позволяющая взаимодействовать со Spark с помощью API

Job – параллельное вычисление, состоящее из нескольких задач, порождаемых в ответ на действие

Stage – каждая job делится на более мелкие наборы задач, которые зависят друг от друга

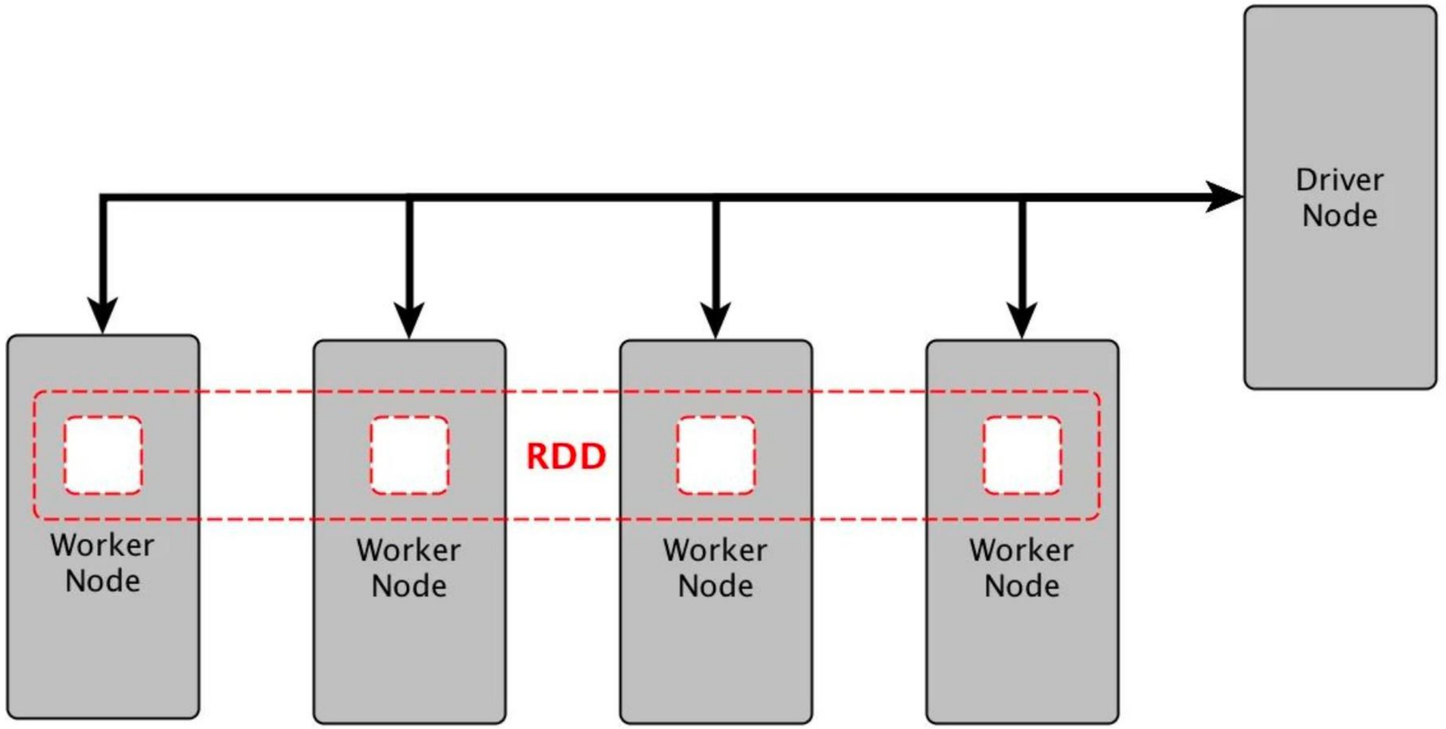
Task – unit of work, которая будет отправлена на **executor**

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

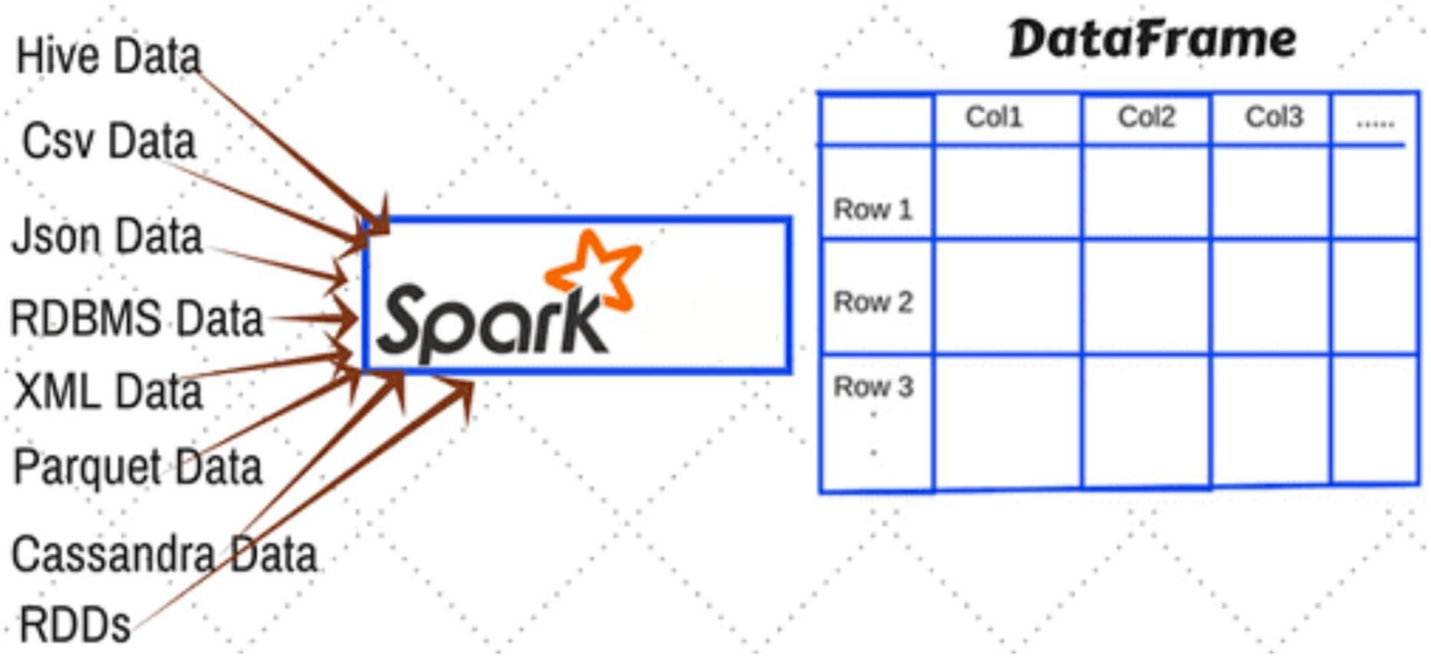
sc.textFile – сформировать RDD из текстового файла
take(5) – посмотреть первые 5 элементов результата
collect – возвращает результат вычислений в память

count() – подсчет числа строк
map – построчная обработка (маппер)
flatMap – разворачивание списка в столбец

filter – фильтрация строк
reduce – попарные действия с элементами

Spark DataFrame

Spark DataFrame — это набор данных, организованный в виде таблицы. Spark DataFrames могут быть созданы из различных источников, таких как файлы структурированных данных, таблицы в Hive, внешние базы данных или существующие RDD.



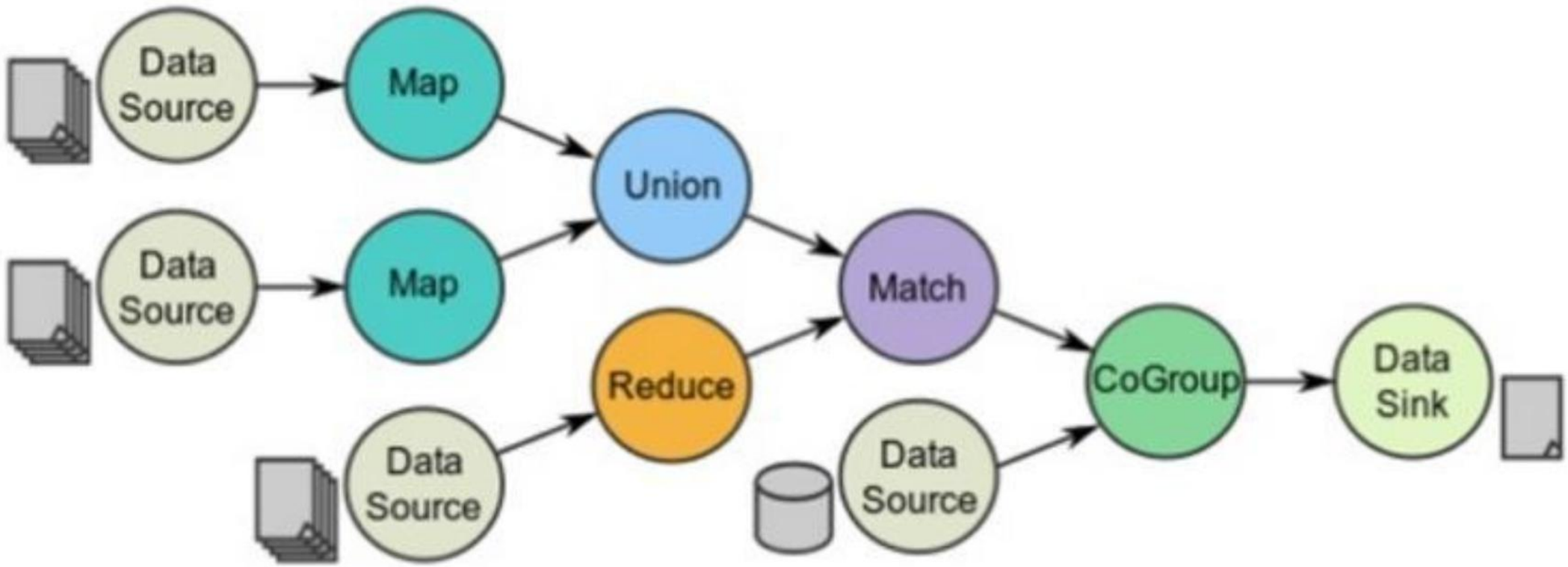
1) spark.read.table()
2) spark.read.parquet()
3) spark.read.csv()
4) spark.sql() >> DataFrame

1) df.show()
2) df.describe().show()
3) df.printShema()

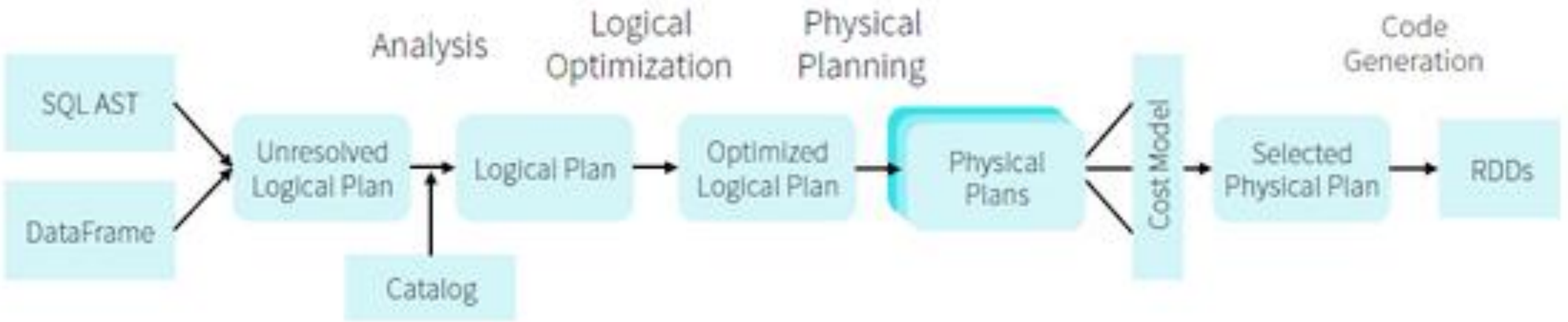
1) df.select()
2) df.filter()
3) df.groupby().agg()

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.



Как spark работает с запросами



Ленивые вычисления

Трансформации:

1. Lazy evaluation – выполняется отложено
2. Immutable – создает новый DataFrame/RDD из существующего

Примеры:

1. orderBy()
2. groupBy()
3. filter()
4. select()
5. join()

Действия:

1. Триггерит выполнение трансформаций
2. Возвращает результат/записывает на диск

Примеры:

1. show()
2. take()
3. collect()
4. count()
5. save()

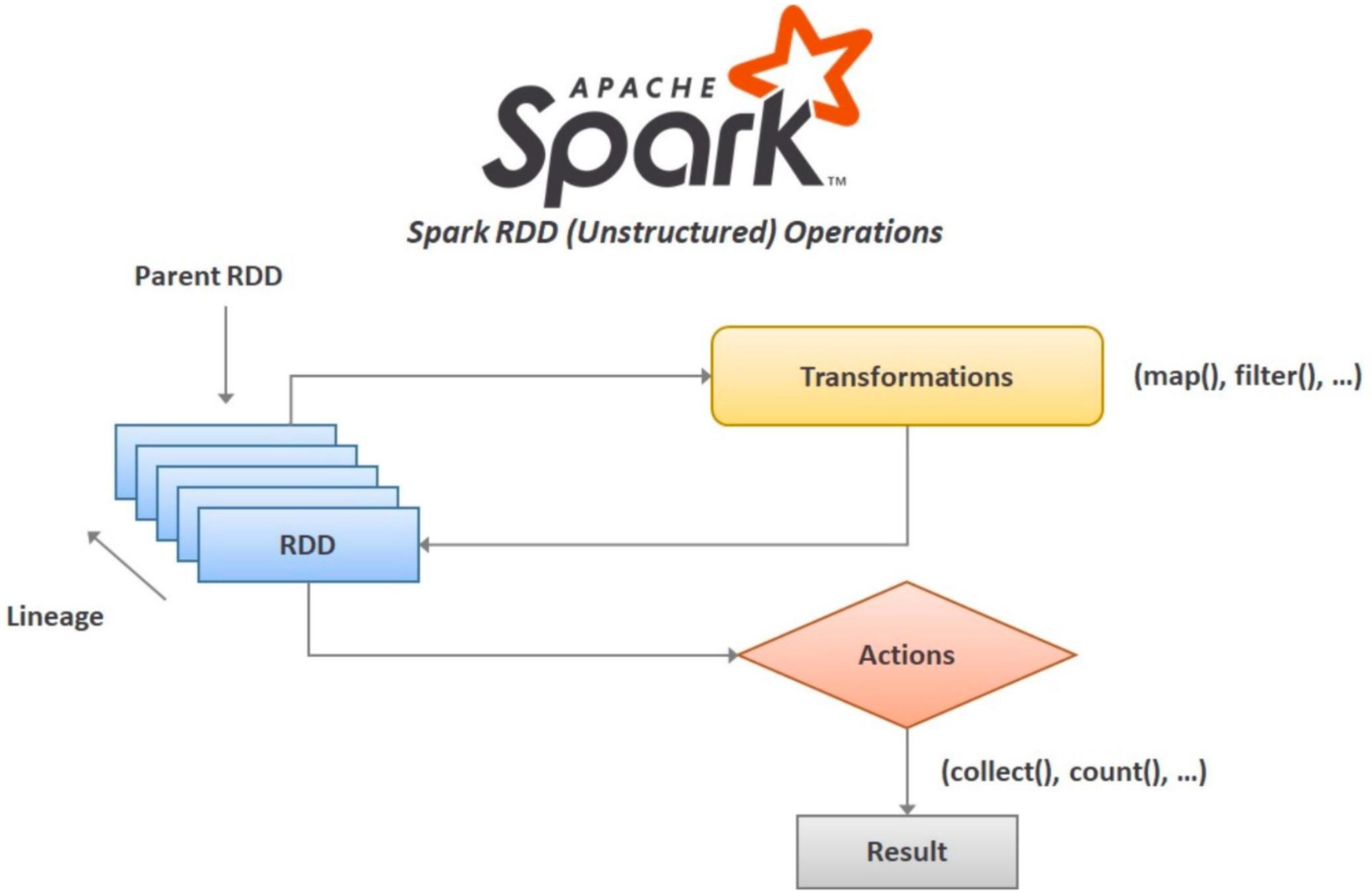
Transformations

map(*func*)	Return a new distributed dataset formed by passing each element of the source through a function *func*.
filter(*func*)	Return a new dataset formed by selecting those elements of the source on which *func* returns true.
mapPartitions(*func*)	Similar to map, but runs separately on each partition (block) of the RDD, so *func* must be of type Iterator<T> => Iterator<U> when running on an RDD of type T.
sortByKey([*ascending*], [*numPartitions*])	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 ascending argument.
repartition(*numPartitions*)	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.
groupByKey([*numPartitions*])	When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs. **Note:** 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. **Note:** 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.
reduceByKey(*func*, [*numPartitions*])	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 *func*, which must be of type (V,V) => V. Like in groupByKey, the number of reduce tasks is configurable through an optional second argument.

Actions

reduce(*func*)	Aggregate the elements of the dataset using a function *func* (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.
collect()	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.
count()	Return the number of elements in the dataset.
first()	Return the first element of the dataset (similar to take(1)).
take(*n*)	Return an array with the first *n* elements of the dataset.
countByKey()	Only available on RDDs of type (K, V). Returns a hashmap of (K, Int) pairs with the count of each key.

Жизненный цикл RDD



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://github.com/tolokonov/hse_pyspark

Спасибо за внимание!