

# Занятие 3

## Распределенные файловые системы. Hadoop, Spark

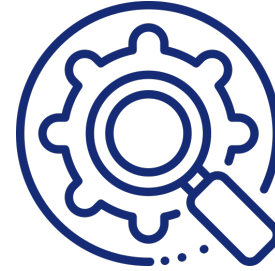
Бояр Владислав

# Занятие состоит из:



## Теория:

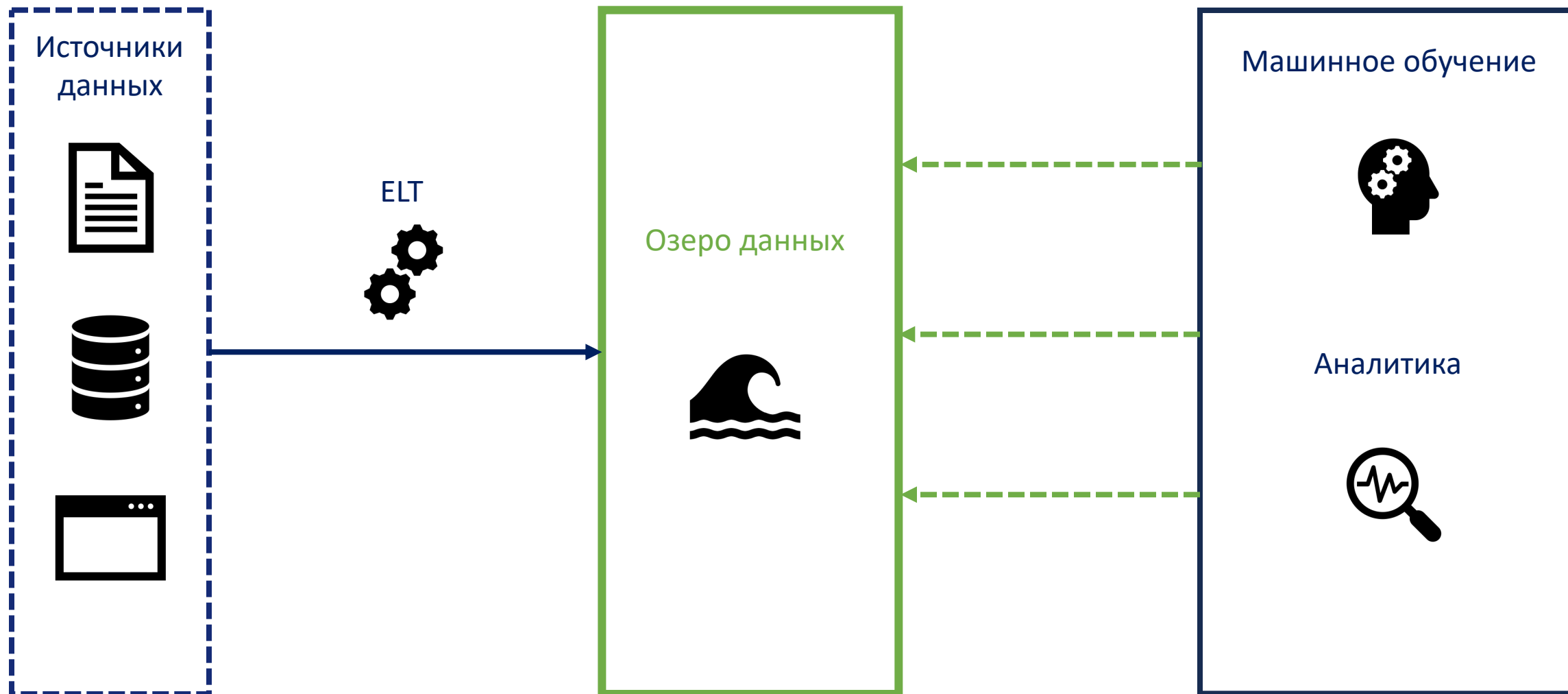
- Hadoop;
- HDFS;
- Spark.



## Практика:

- PySpark.

# Вспомним что такое Data Lake



# Hadoop

# Что такое Hadoop?

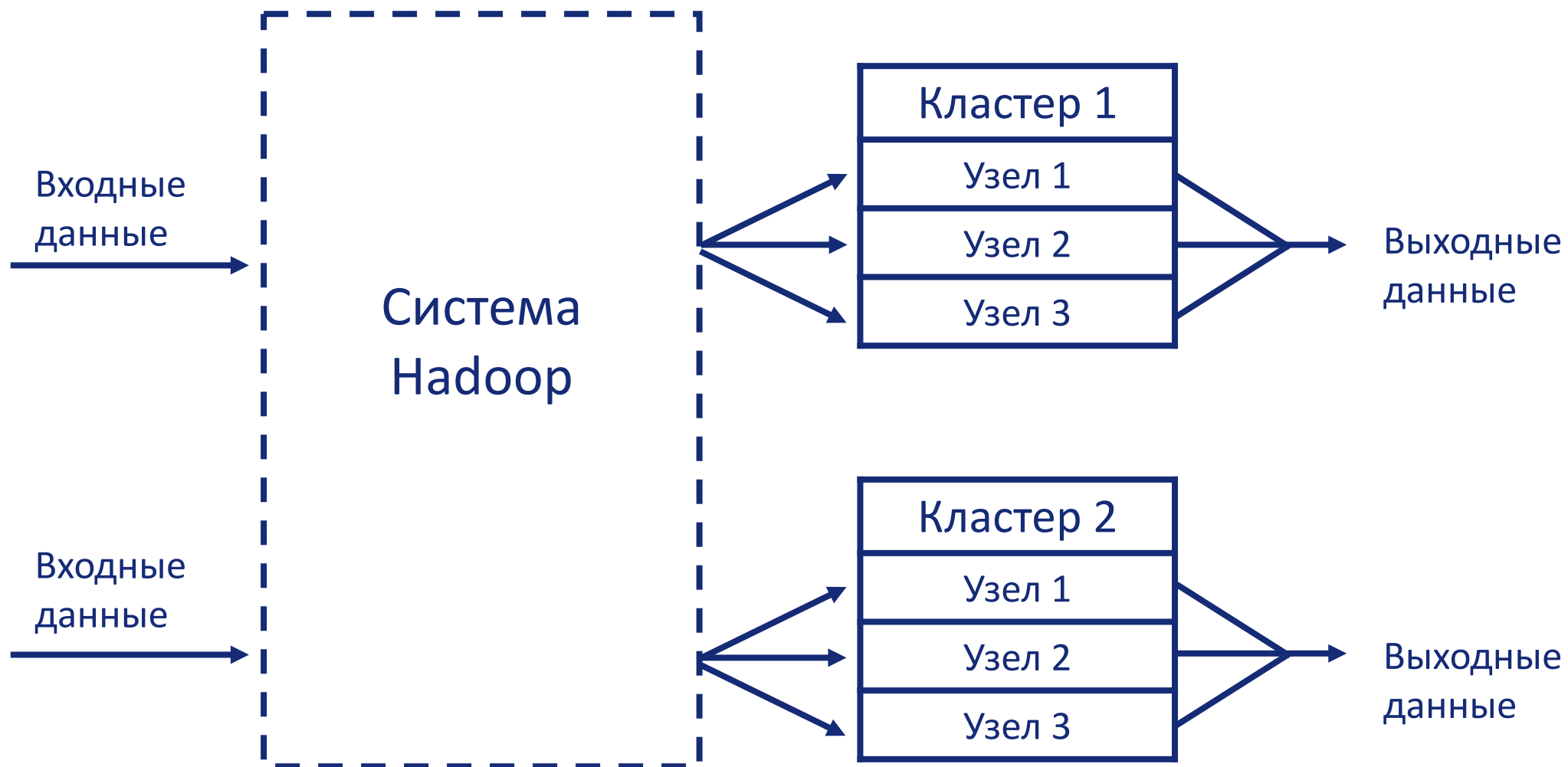
- Экосистема для распределенного хранения и обработки данных
- Основная технология обработки больших данных
- На его основе строятся Data Lakes
- Open-source

## Архитектурно состоит из кластеров

- кластеры используются как единый ресурс
- каждая задача разделяется на несколько более простых подзадач
- подзадачи выполняются параллельно на разных узлах кластера



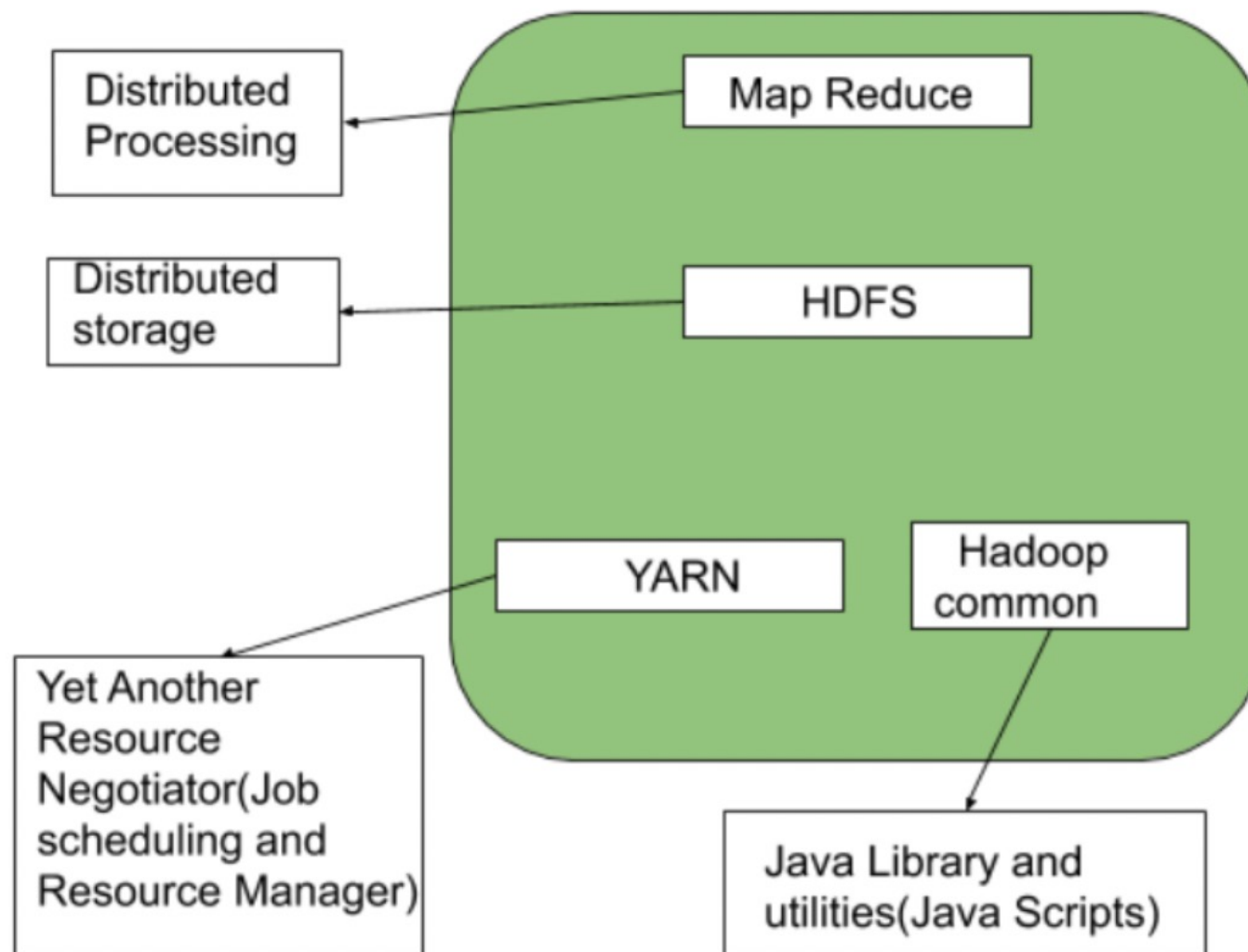
# Что такое Hadoop?



# Основные компоненты Hadoop

- ① **Hadoop Common** – управляющая часть над другими компонентами Hadoop и связующее звено с дополнительными инструментами. Другими словами, это набор инструментов, позволяющих создавать инфраструктуру и работать с файлами.
- ② **HDFS (Hadoop Distributed File System)** – распределённая файловая система; технология хранения файлов на различных серверах данных (узлах, Data Nodes), адреса которых находятся на специальном сервере имен (мастере, Name Node). Обеспечивает сохранность данных от потерь за счет дублирования данных (репликаций). В HDFS хранятся неструктурированные данные.
- ③ **YARN (Yet Another Resource Negotiator)** – система планирования заданий и управления кластером; набор программ, обеспечивающих связь между кластером и приложениями, которые используют его ресурсы для обработки данных.
- ④ **Hadoop MapReduce** – платформа выполнения распределённых MapReduce-вычислений. Позволяет распределять входные данные по узлам кластера.

# Основные компоненты Hadoop





# Hadoop

## Плюсы:

- + эффективная обработка больших данных благодаря модели MapReduce и распараллеливанию вычислений
- + возможность работы с неструктурированными данными
- + масштабируемость
- + гибкость
- + отказоустойчивость (данные реплицированы)

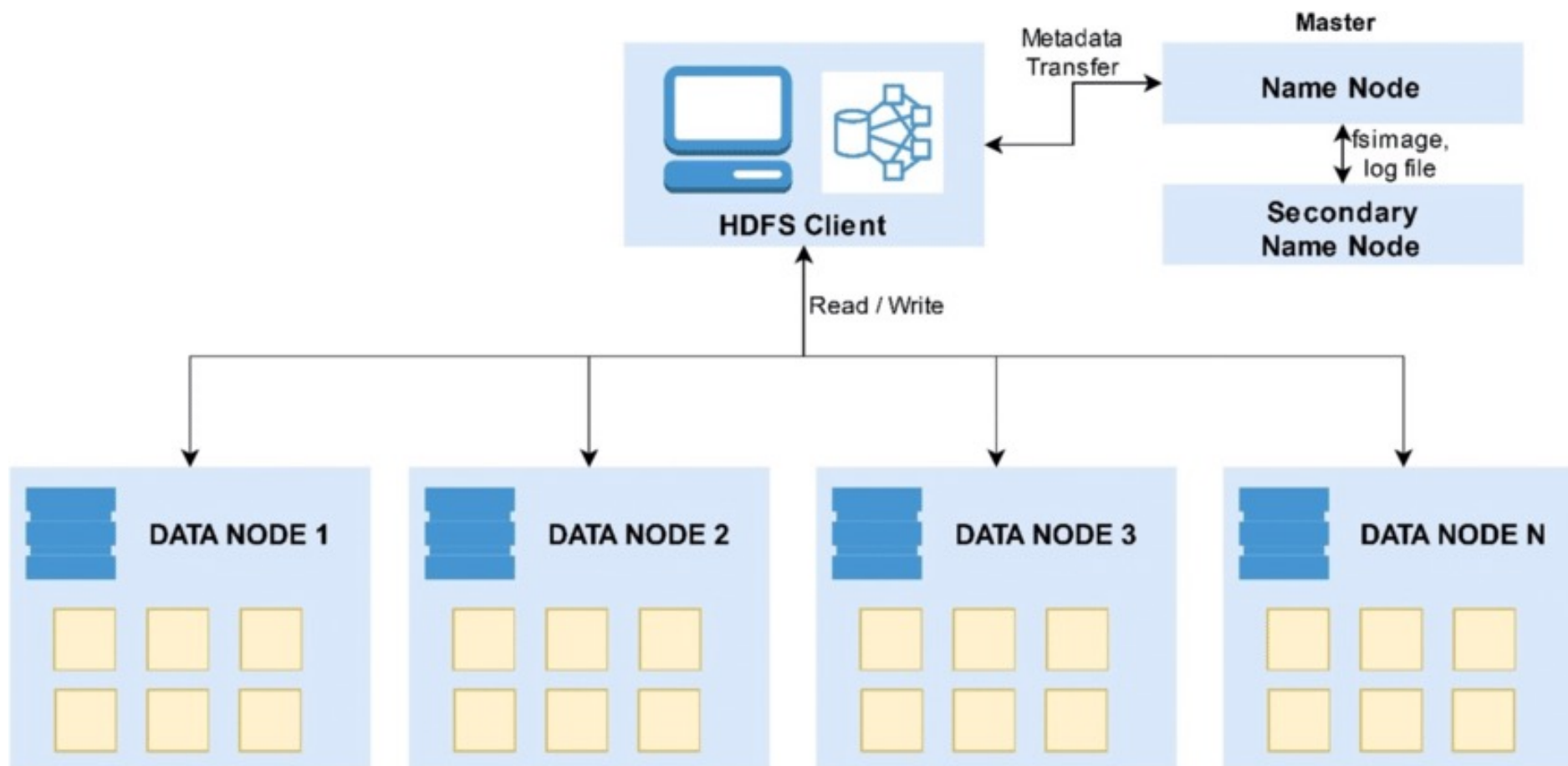
## Минусы:

- нецелесообразно использовать, когда данных мало (минимальный объём блока HDFS 128 МБайт)
- дорого поддерживать инфраструктуру (поскольку open-source)

# Распределительная файловая система Hadoop

# HDFS (Hadoop Distributed File System)

**Распределенная файловая система Hadoop** - файловая система, позволяющая хранить файлы большого объема в распределенной среде (на кластере, состоящем из нескольких узлов)

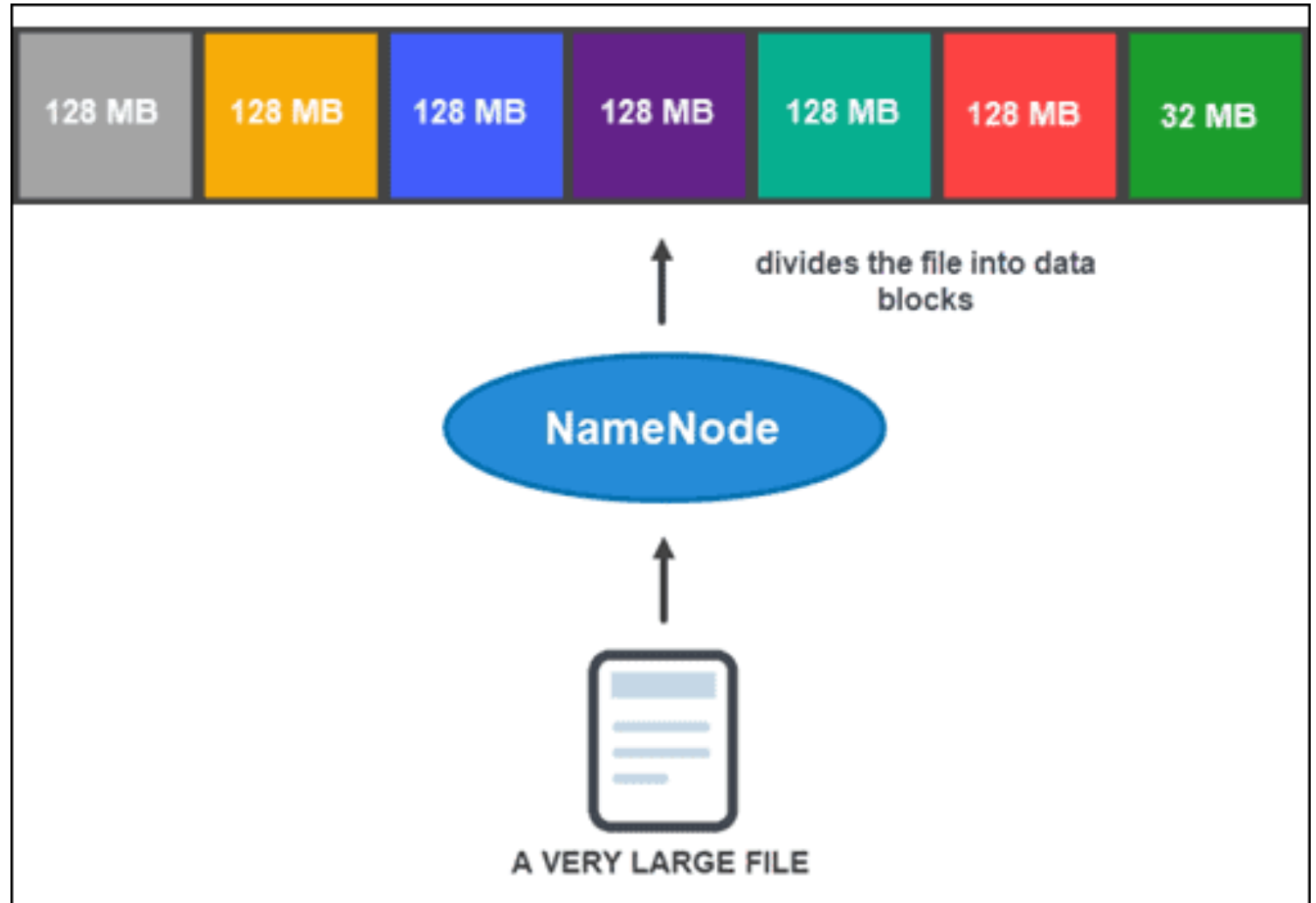


# Из чего состоит кластер

- **Name Node** - хранит метаданные системы: на какой ноде лежат определенные данные. Как правило, одна на кластер.
- **Secondary Name Node** - Необходима для быстрого восстановления Name Node в случае её выхода из строя. Как правило, одна на кластер.
- **Data Node** – хранит блоки файлов. Таких элементов в кластере много, ограничение только в технических ВОЗМОЖНОСТЯХ.

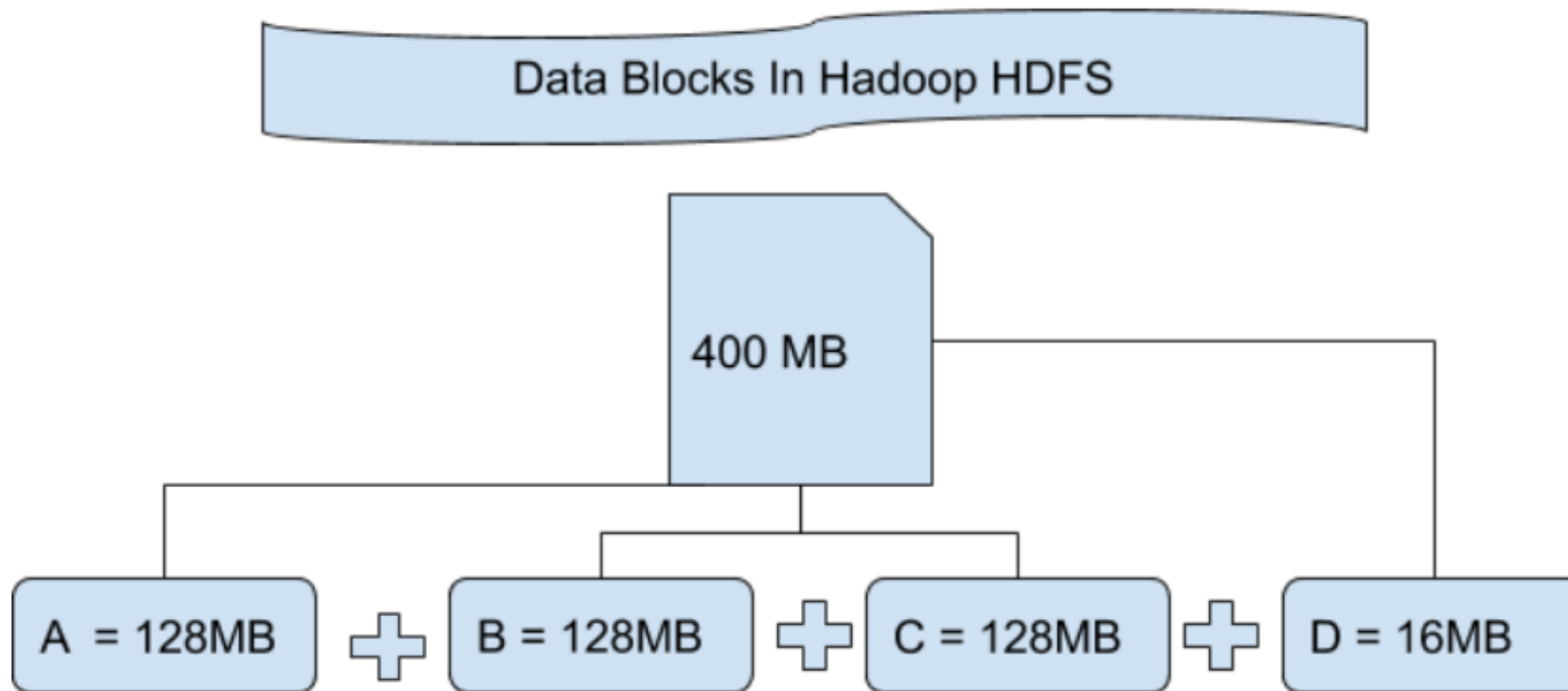
# Хранение данных в HDFS

- все загружаемые в HDFS файлы разбиваются на части (блоки)
- по умолчанию размер блока составляет 128 МБ
- кол-во используемых блоков зависит от размера исходного файла



# Пример хранения файла в HDFS

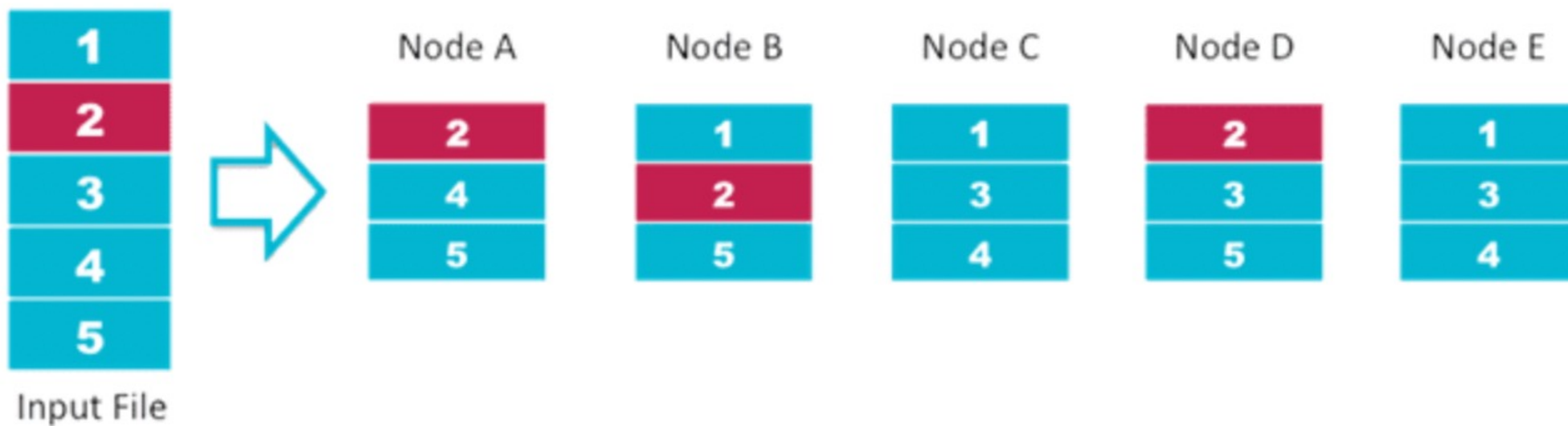
- файл объемом 400 МБ будет разбит на 3 блоков по 128 МБ и один на 16 МБ.



# Репликация данных в HDFS

- репликация (дублирование) информации осуществляется для сохранения данных в случае отказа одного из элементов кластера
- по умолчанию части (блоки) файла реплицируются дважды (создаются две копии) и сохраняются в разные места (на разные ноды)

## HDFS Data Distribution



# Как обратиться к HDFS?

## Основные команды HDFS CLI

- Загрузить файл `hdfs dfs -put data.txt /tmp/data.txt`
- Скачать файл `hdfs dfs -get /tmp/data.txt /download`
- Посмотреть содержимое директории `hdfs dfs -ls /tmp`
- Создать директорию `hdfs dfs -mkdir /tmp/test`
- Удалить директорию `hdfs dfs -rm -R /tmp/test`

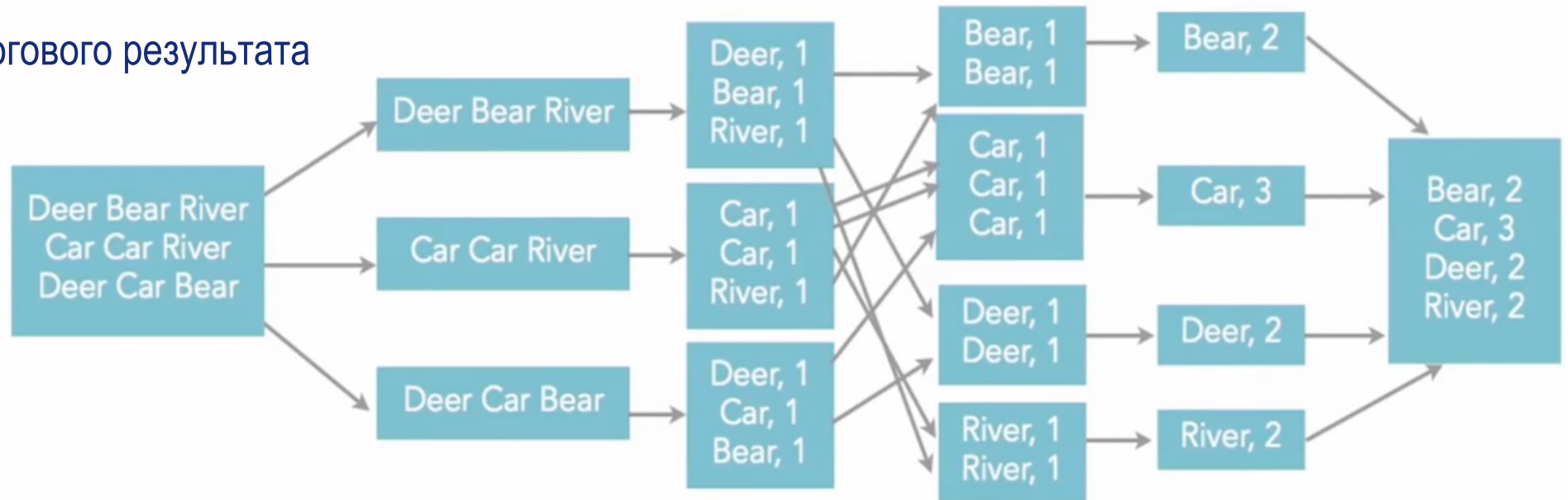
**Недостаток HDFS:** небольшие файлы занимают минимальный размер блока (128Мб)



Map Reduce

# Map Reduce

- Входные данные
- Распределение данных по узлам
- Применение функции-трансформации: **Map** (GroupBy)
- Shuffle (обмен данными между узлами)
- Применение функции-действия: **Reduce** (Count)
- Получение итогового результата



# Map Reduce над списком Python

```
input_list = [1, 2, 3, 4, 5]
```

```
mapped_list = list(map(lambda x: x ** 2, input_list))
```

---

```
mapped_list: [1, 4, 9, 16, 25]
```

```
reduced_value = sum(mapped_list)
```

---

```
reduced_value: 55
```

# Пример интерфейса Hadoop

Hadoop

Overview

Datanodes

Datanode Volume Failures

Snapshot

Startup Progress

Utilities ▾

## Browse Directory

/origin\_data/gmail/log/topic\_log/2020-06-16

Go!



Show 25 ▾ entries

Search:

<input type="checkbox"/>	↕	Permission	↕	Owner	↕	Group	↕	Size	↕	Last Modified	↕	Replication	↕	Block Size	↕	Name	↕
<input type="checkbox"/>		<a href="#">-rw-r--r--</a>		<a href="#">atguigu</a>		<a href="#">supergroup</a>		98 KB		Mar 02 19:38		<a href="#">1</a>		128 MB		<a href="#">log-.1646221072146.lzo</a>	
<input type="checkbox"/>		<a href="#">-rw-r--r--</a>		<a href="#">atguigu</a>		<a href="#">supergroup</a>		38.14 KB		Mar 02 19:38		<a href="#">1</a>		128 MB		<a href="#">log-.1646221086713.lzo</a>	

Showing 1 to 2 of 2 entries

Previous

1

Next

Hadoop, 2019.

# Форматы данных HDFS

## Строковые:

- csv, json, avro;
- быстрые операции записи, медленное чтение

## Колоночные:

- parquet, rc, orc;
  - быстрые операции чтения и фильтрации, медленная запись
- 

# Строковые VS Колоночные форматы

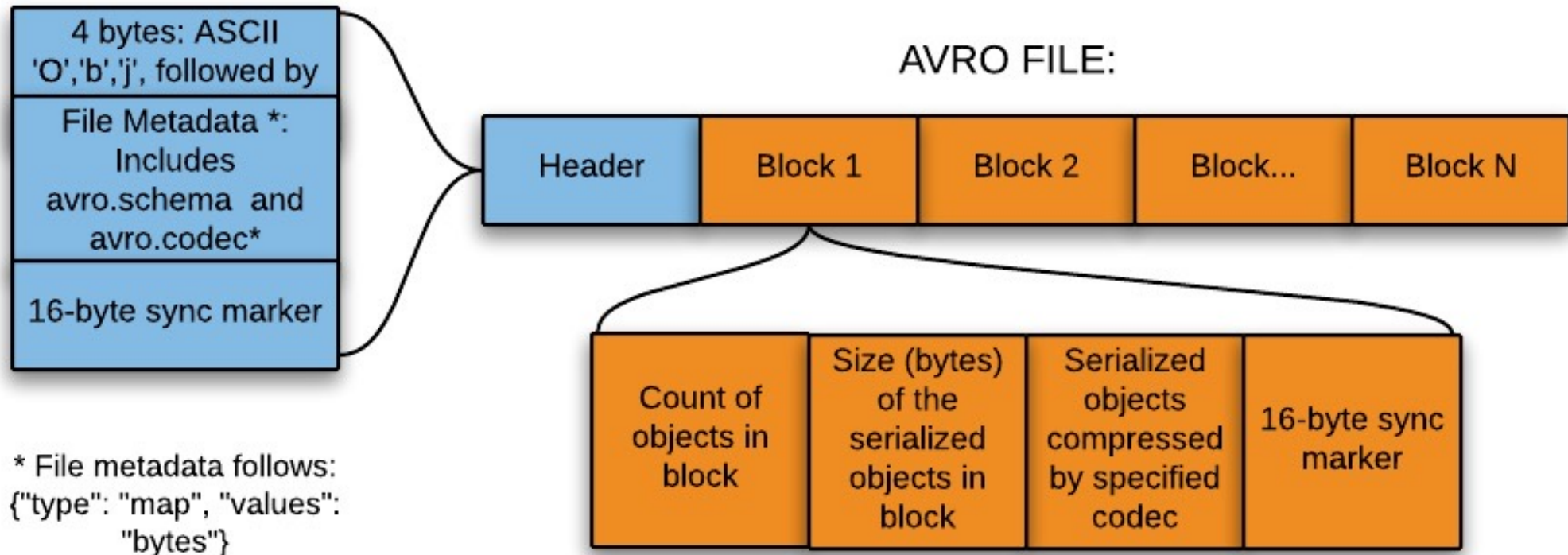
Row Store	
Row 1	India
	Chocolate
	1000
Row 2	India
	Ice-cream
	2000
Row 3	Germany
	Chocolate
	4000
Row 4	US
	Noodle
	500

Table			
	Country	Product	Sales
Row 1	India	Chocolate	1000
Row 2	India	Ice-cream	2000
Row 3	Germany	Chocolate	4000
Row 4	US	Noodle	500

Column Store	
Country	India
	India
	Germany
	US
Product	Chocolate
	Ice-cream
	Chocolate
	Noodle
Sales	1000
	2000
	4000
	500

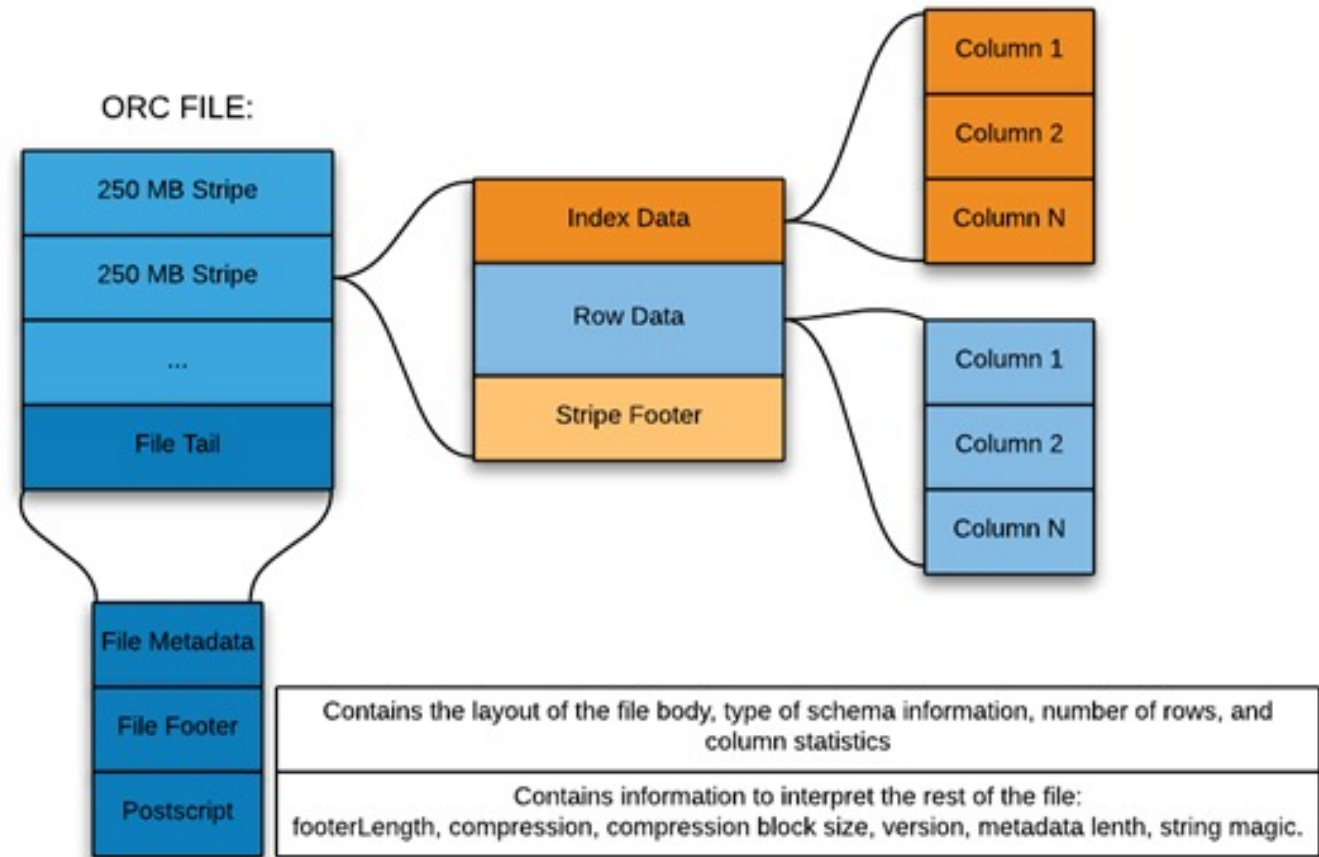
# AVRO

- отдельно хранит схему данных в формате JSON



# ORC (Optimized Row Columnar File)

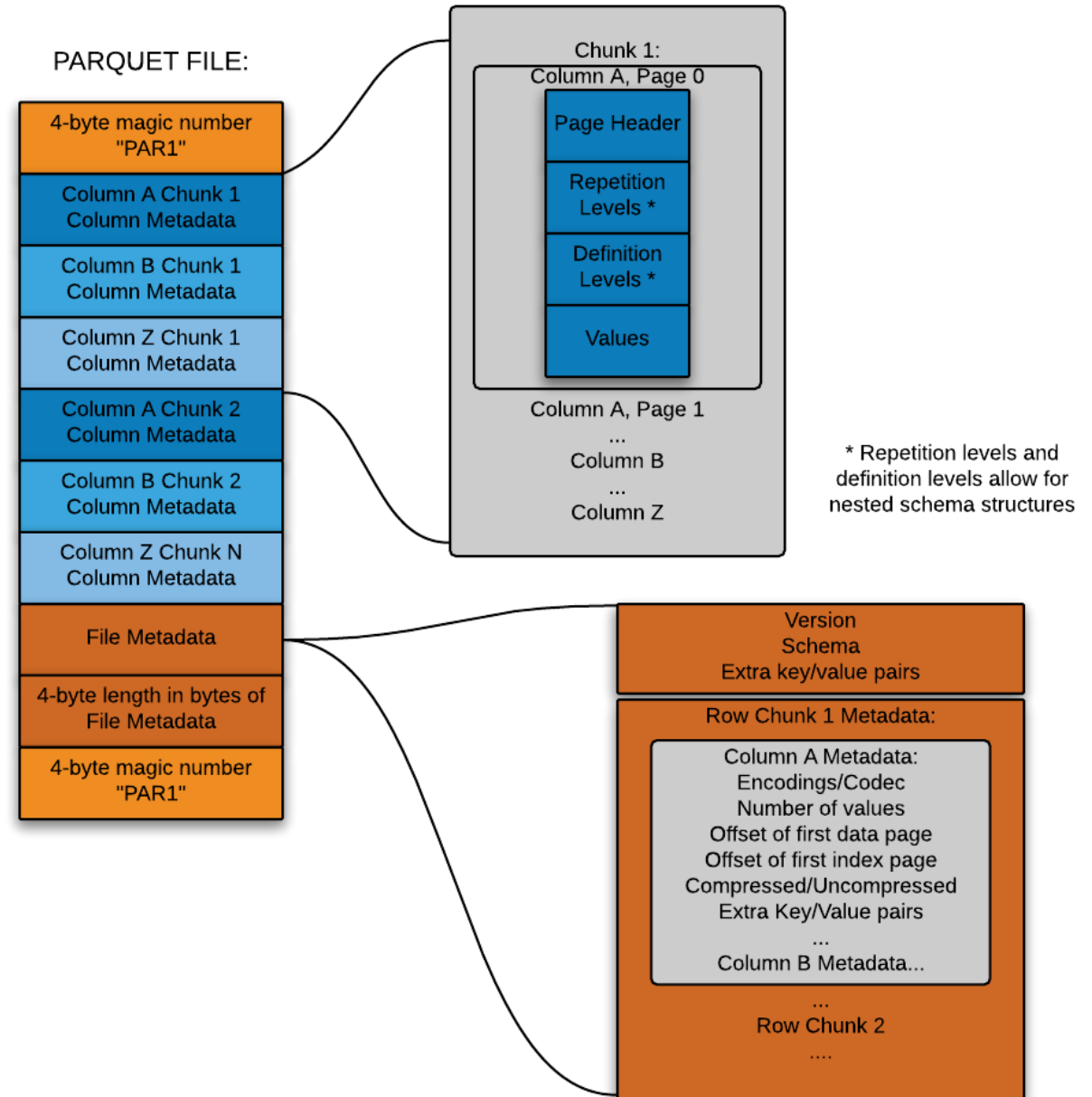
- оптимизированный строково-колоночный формат файлов
- данные разделены на полосы по 250 МБ
- колонки в полосах разделены друг от друга, что позволяет считывать данные избирательно





# Parquet

- колоночно-ориентированный формат данных
- наиболее популярный формат для работы с помощью Spark
- позволяет хранить данные с вложенными структурами и быстро считывать их



# Структура Parquet

Имеет 3 уровня:

- ① **Группа строк (Row group)** – логическое горизонтальное разбиение данных на строки, состоящие из фрагментов каждой колонки в наборе данных.
- ② **Фрагмент колонки (Column chunk)** – фрагмент конкретной колонки. Эти фрагменты хранятся в определенной группе строк и гарантированно будут смежными в файле.
- ③ **Страница (Page)** – фрагменты колонок делятся на страницы, записанные друг за другом. У страниц общий заголовок, позволяющий пропускать ненужные при чтении.

\* В **футере** хранятся координаты каждой колонки, для быстрого чтения.

# Apache Spark

# Apache Spark



- Фреймворк для обработки больших данных (распределенных/кластерных вычислений)
- Работает в несколько раз быстрее MapReduce
- Поддержка языков программирования (Scala, Python, Java, R)
- Поддержка SQL запросов

## Распределенные вычисления

- вычисления производятся на кластере;
- кластер - несколько компьютеров, объединенных в одну сеть;
- обрабатываемый файл делится на несколько частей;
- каждая часть обрабатывается параллельно с другими частями;
- каждая часть обрабатывается отдельно на своей части кластера.

# Основные области применения Spark



Аналитика больших данных



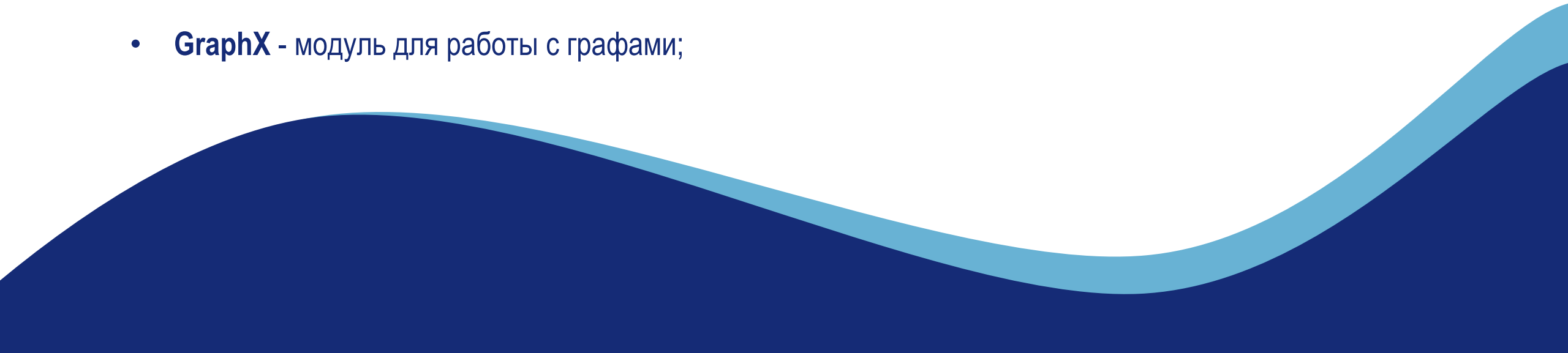
Машинное обучение

# Spark VS Hadoop Map Reduce

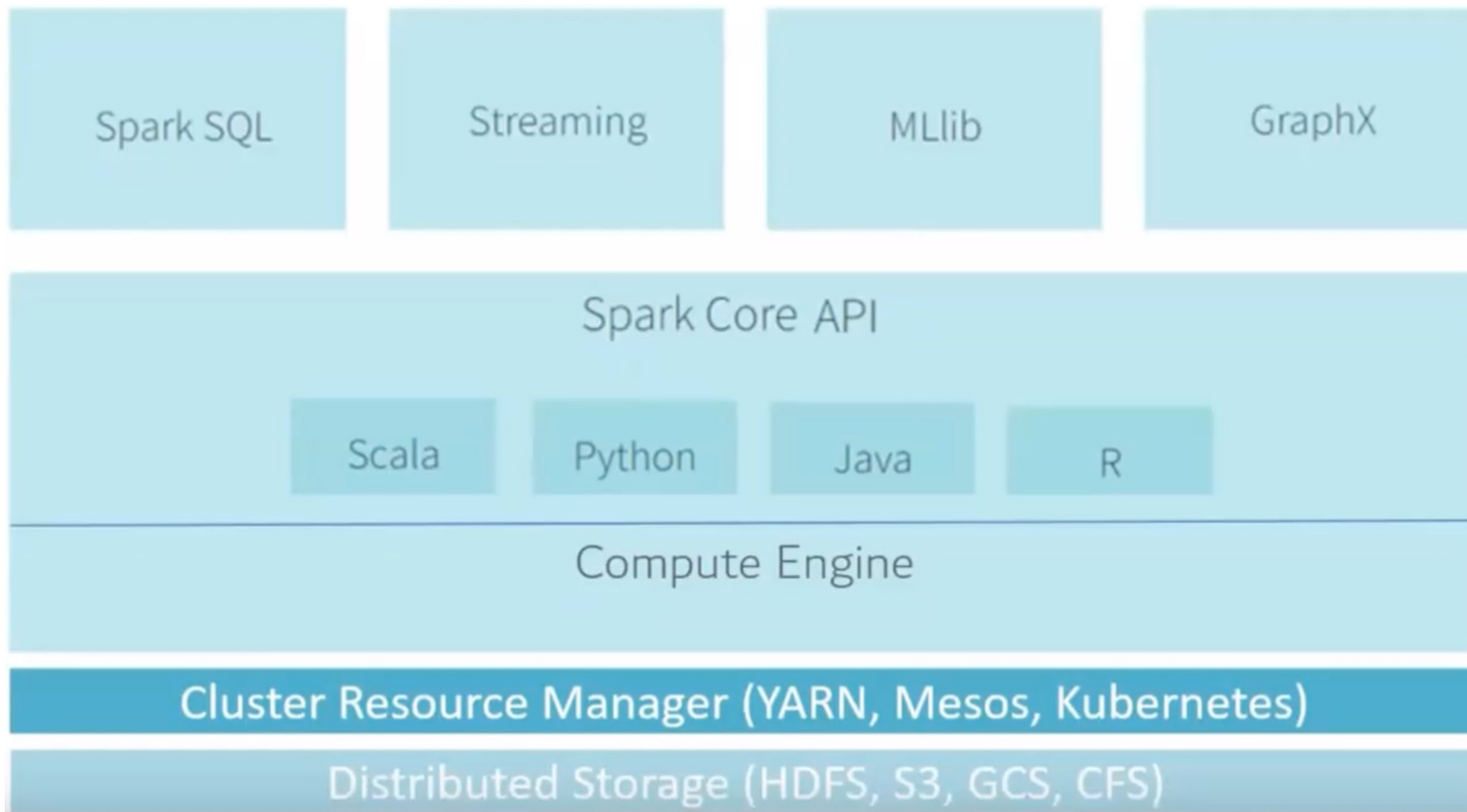
Spark быстрее ~ в 100 раз, чем Map Reduce

Spark	Hadoop Map Reduce
Обработка данных в реальном времени (Real-time processing)	Пакетная обработка данных (Batch processing)
При обработке данных задействует оперативную память	При обработке данных использует ресурсы диска
Написан на Scala	Написан на Java

# Архитектура Spark

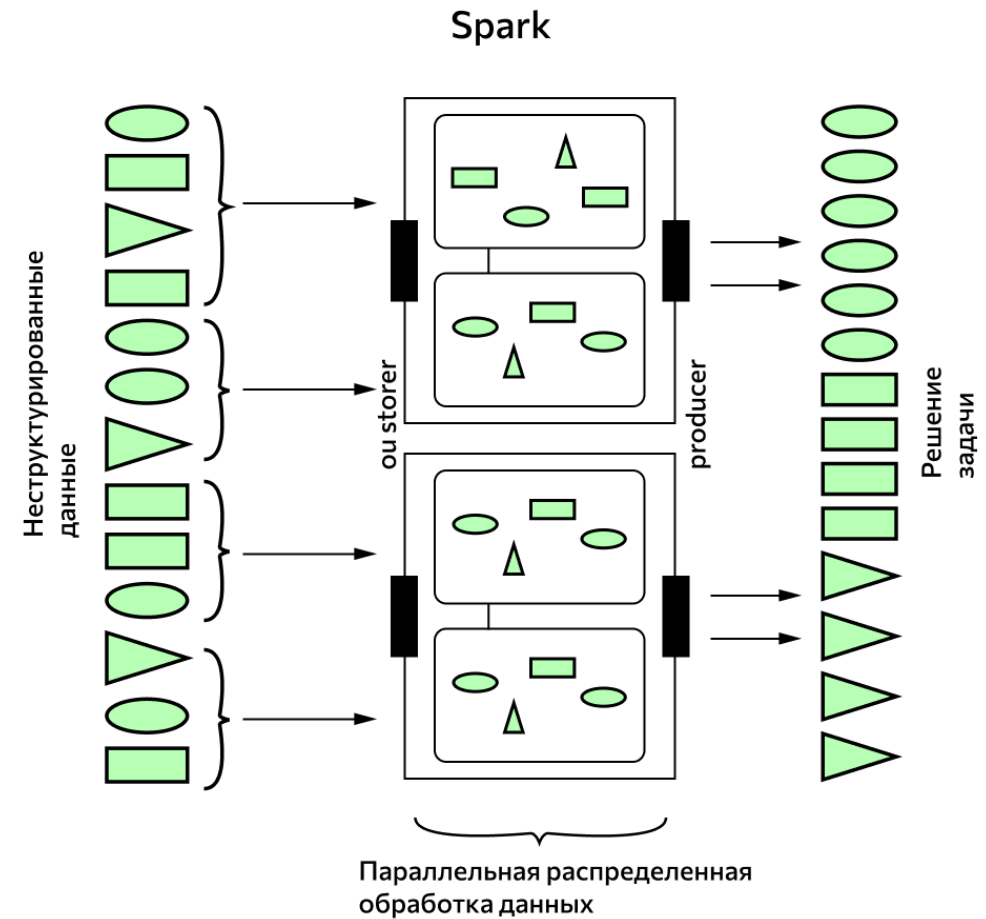
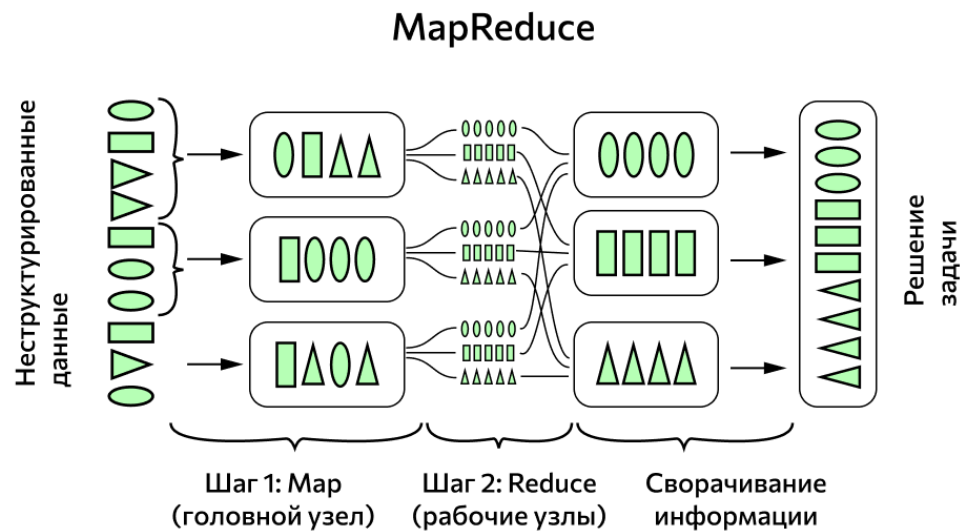
- **Spark Core** - ядро спарка, отвечает за хранение данных, управление памятью, распределение и отслеживание задач в кластере;
  - **Streaming** - средство потоковой обработки в режиме реального времени;
  - **Spark SQL** - компонент, позволяющий делать запросы к данным;
  - **MLlib** - набор библиотек для машинного обучения;
  - **GraphX** - модуль для работы с графами;
- 

# Архитектура Spark





# Spark обрабатывает данные в режиме реального времени небольшими группами

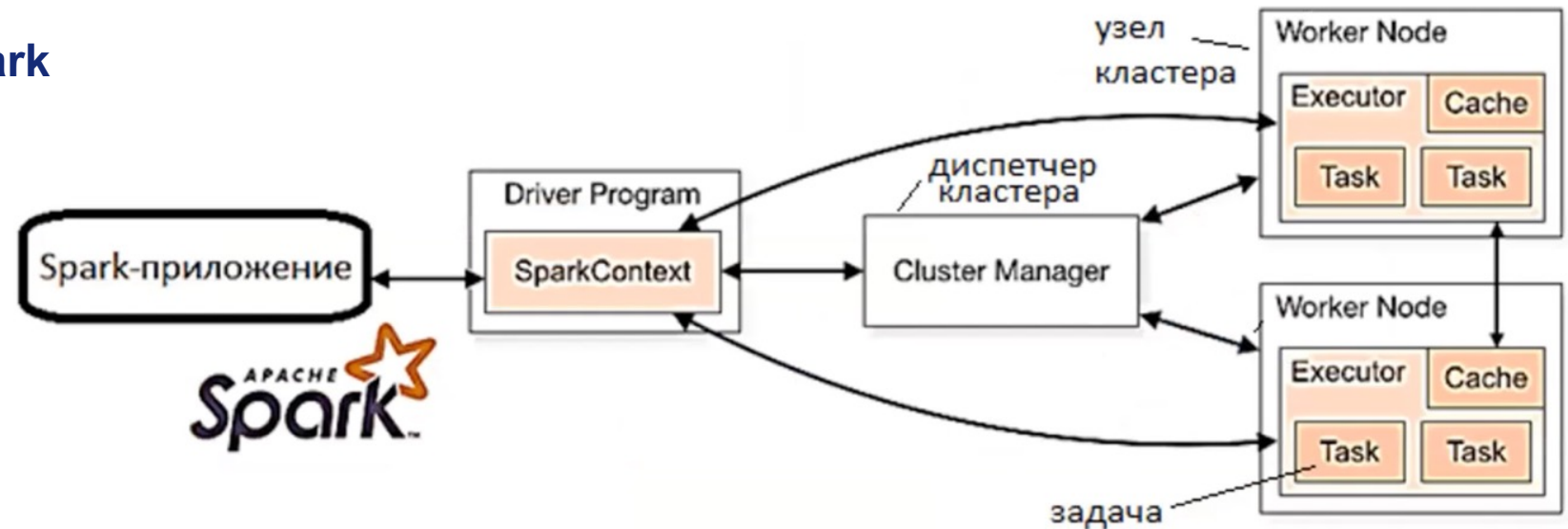


# Схема работы Spark

- При инициализации работы со Spark создается экземпляр класса **SparkContext**;
- **Driver Program** – программа - менеджер, управляющая процессом вычислений: создаёт задания и планирует их выполнение для исполнителей;
- **Cluster Manager** (YARN, Mesos, Kubernetes) – распределяет ресурсы между исполнителями (статическое/динамическое распределение);
- **Executor** (исполнитель) – выполняет отдельную задачу из задания

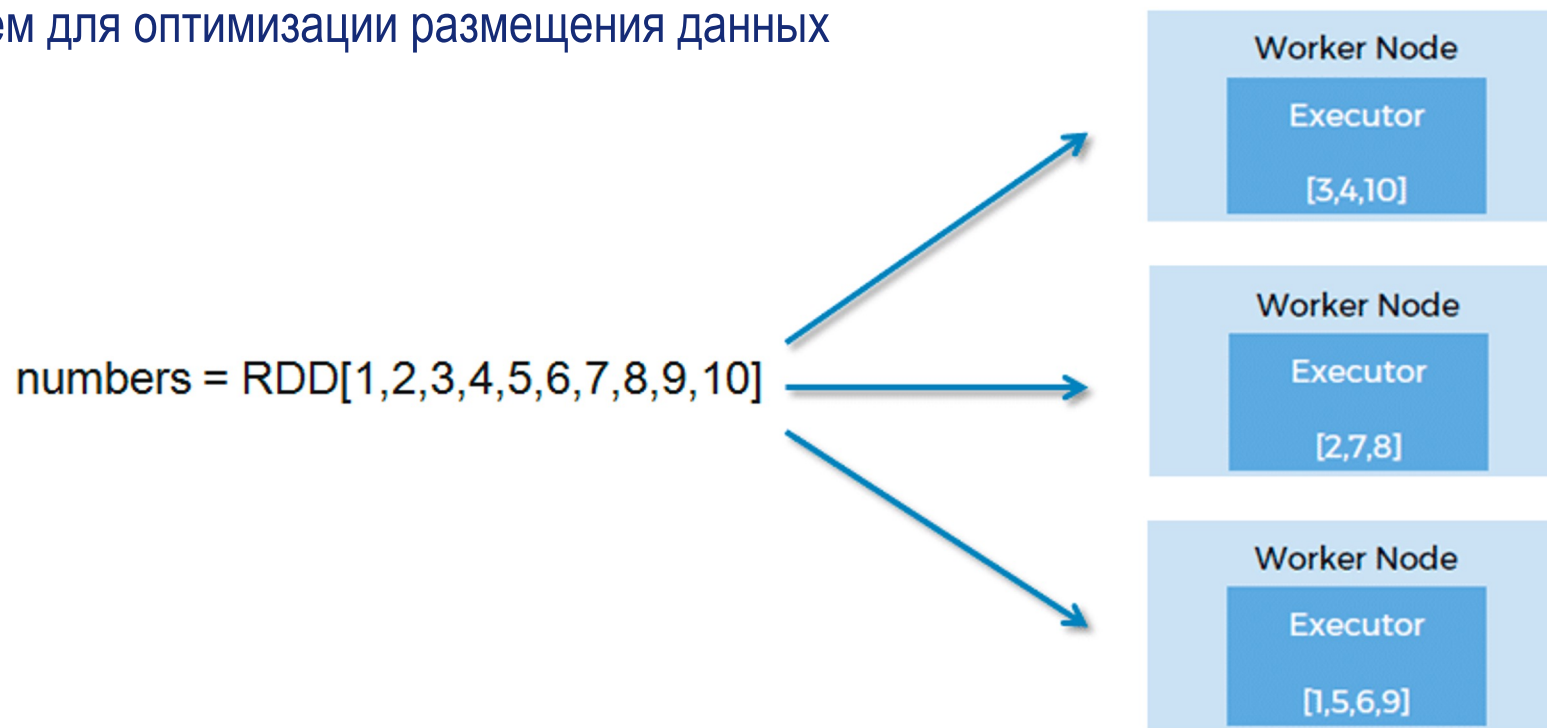
## Форматы данных Spark

- RDD
- DataFrame

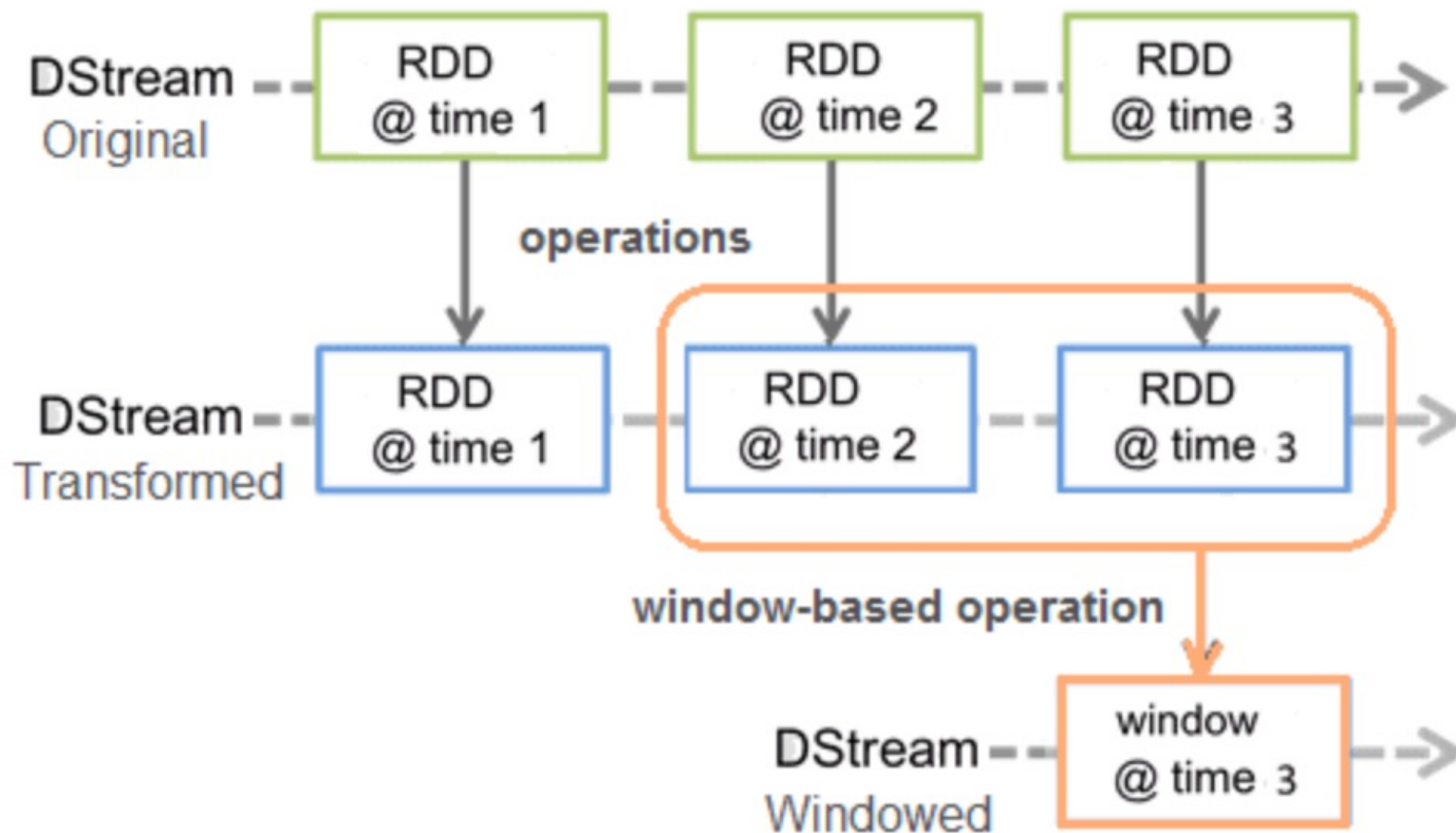


# RDD (Resilient Distributed Dataset)


- отказоустойчивый распределённый датасет, над которым можно производить параллельные вычисления и преобразования с помощью встроенных и пользовательских функций
- позволяет пользователям явно сохранять промежуточные результаты в памяти и управлять их разбиением для оптимизации размещения данных



# Как происходит потоковая обработка RDD



# DataFrame


- В отличие от RDD данные хранятся в именованных столбцах (как в Pandas DataFrame или реляционных БД);
  - Позволяет работать с данными в упрощенном формате (примерно, как в Pandas, но намного быстрее)
  - Как правило, используется для реляционных преобразований, а также для создания временного представления (таблицы), которое позволяет применять к данным SQL-запросы.
- 
- The bottom of the slide features a decorative graphic consisting of two overlapping wavy lines. The lower line is a dark blue, and the upper line is a lighter, medium blue, creating a layered, wave-like effect across the width of the slide.

# Операции Spark: Трансформации и действия

## Трансформации:

- определяют последовательность операций для вычислений;
- filter, union, distinct, join, group, sort.

## Действия:


- возвращают значения / генерируют наборы данных;
  - count, aggregate, saveAsTextFile.
- 

# Spark UI

Spark Master at spa x

10.0.10.175:8080

Apps opencredo envoy cassandra

 2.1.0

Spark Master at spark://10.0.10.175:7077

URL: spark://10.0.10.175:7077

REST URL: spark://10.0.10.175:6066 (cluster mode)

Alive Workers: 3

Cores in use: 6 Total, 6 Used

Memory in use: 19.2 GB Total, 6.0 GB Used

Applications: 3 Running, 0 Completed

Drivers: 0 Running, 0 Completed

Status: ALIVE

### Workers

Worker Id	Address	State	Cores	Memory
<a href="#">worker-20170410154728-10.0.10.73-39180</a>	10.0.10.73:39180	ALIVE	2 (2 Used)	6.4 GB (2.0 GB Used)
<a href="#">worker-20170410154739-10.0.11.65-47952</a>	10.0.11.65:47952	ALIVE	2 (2 Used)	6.4 GB (2.0 GB Used)
<a href="#">worker-20170410154744-10.0.12.48-46280</a>	10.0.12.48:46280	ALIVE	2 (2 Used)	6.4 GB (2.0 GB Used)

### Running Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
<a href="#">app-20170410162323-0002</a> (kill)	<a href="#">Calculate Balance</a>	0	1024.0 MB	2017/04/10 16:23:23	centos	WAITING	2 s
<a href="#">app-20170410162322-0001</a> (kill)	<a href="#">Find Suspicious Transactions</a>	3	1024.0 MB	2017/04/10 16:23:22	centos	RUNNING	3 s
<a href="#">app-20170410162321-0000</a> (kill)	<a href="#">Calculate Average Spending Per County</a>	3	1024.0 MB	2017/04/10 16:23:21	centos	RUNNING	5 s

### Completed Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
----------------	------	-------	-----------------	----------------	------	-------	----------



# Практика

Написание запросов в Google Colab на PySpark