

Занятие 5

Распределенные файловые системы. Hadoop, Spark

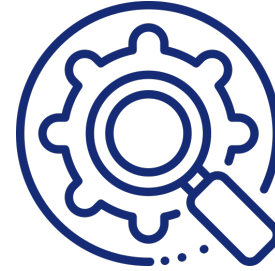
Бояр Владислав

Занятие состоит из:



Теория:

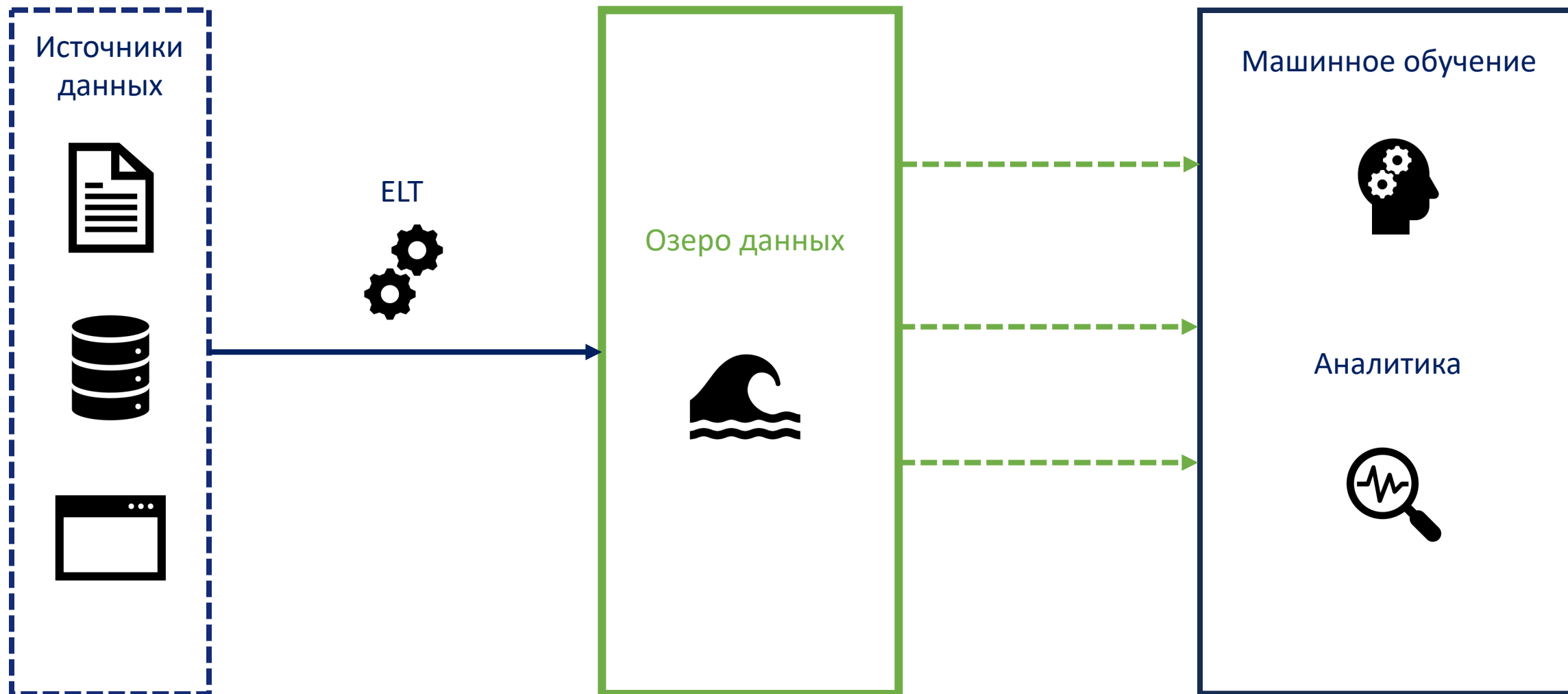
- Hadoop;
- HDFS;
- Spark.



Практика:

- PySpark.

Вспомним что такое Data Lake





Что такое Hadoop?

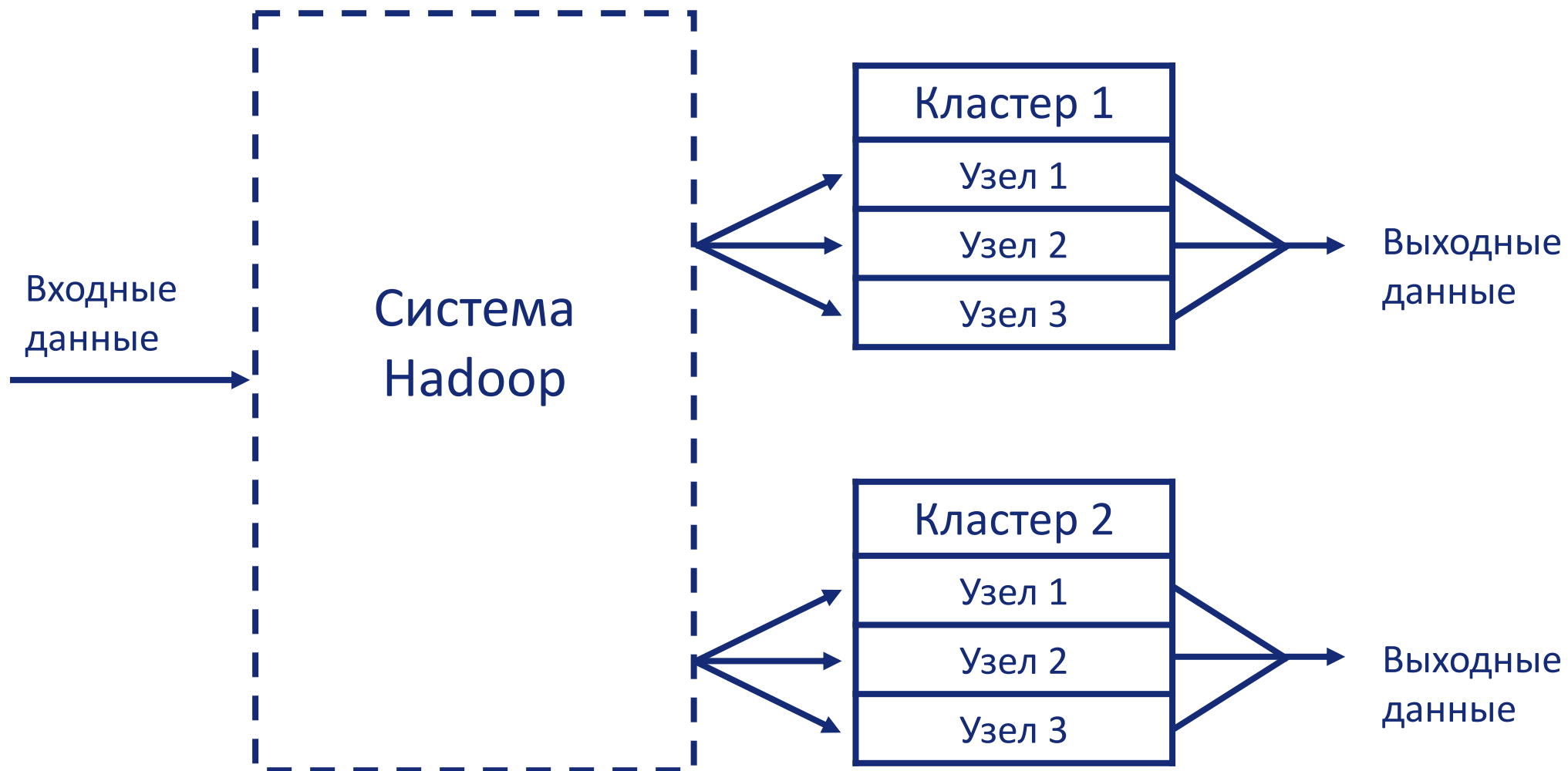
- Экосистема для распределенного хранения и обработки данных;
- Основная технология обработки больших данных;
- На его основе строятся Data Lakes;
- Open-source.

Архитектурно состоит из кластеров

- Кластеры используются как единый ресурс;
- Каждая задача разделяется на несколько более простых подзадач;
- Подзадачи выполняются параллельно на разных узлах кластера.



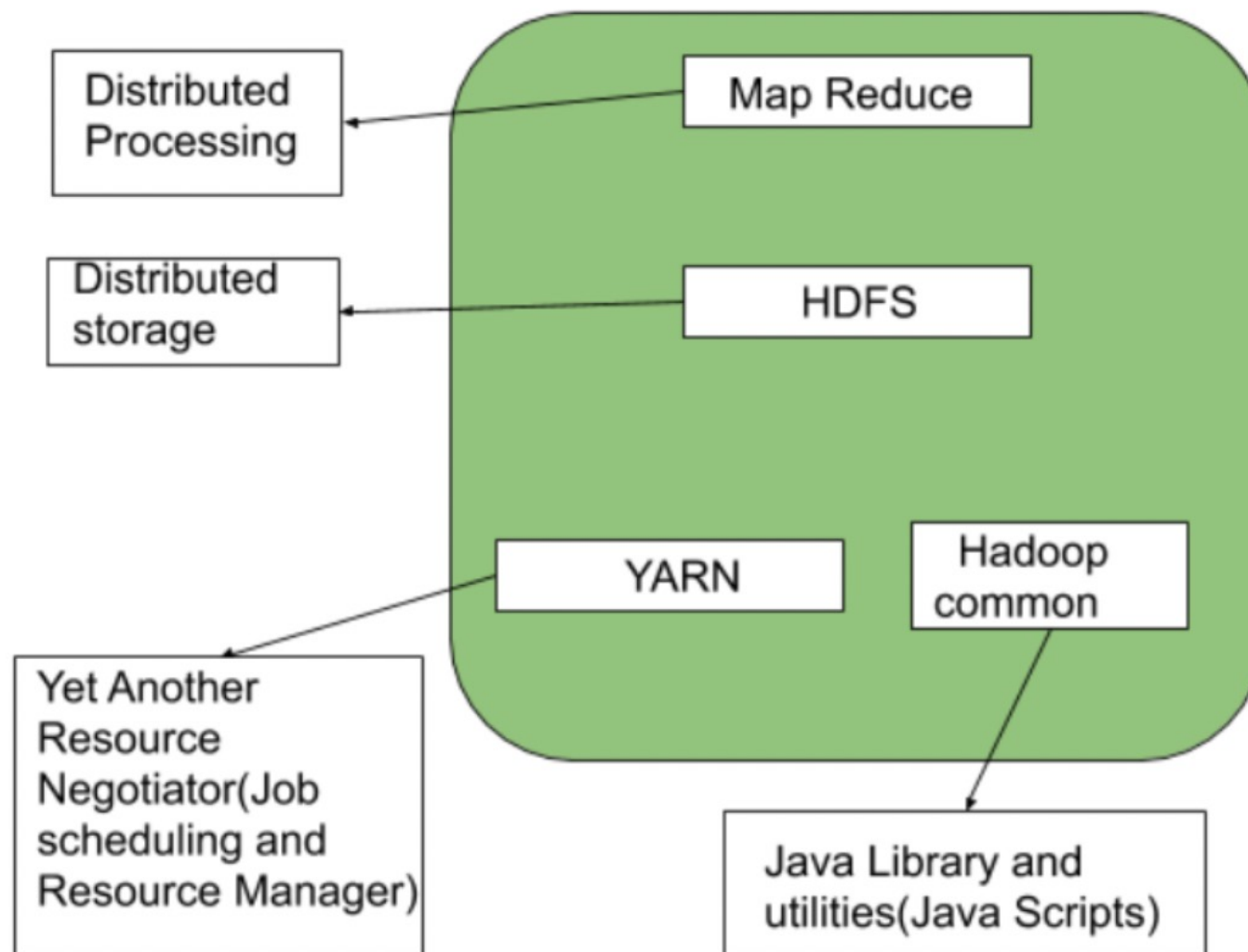
Что такое Hadoop?



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, как правило, 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).

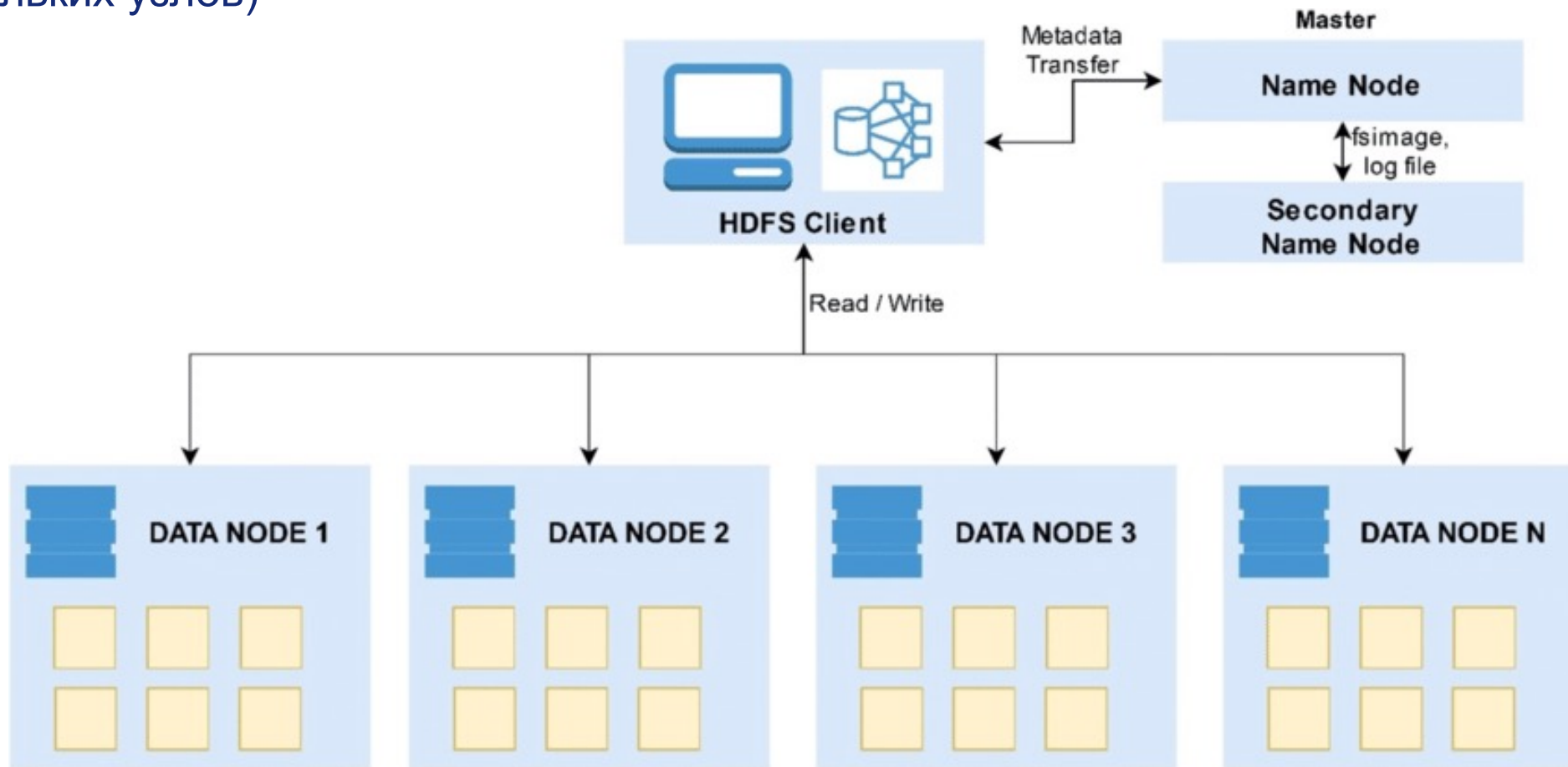
Минусы:

- 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)

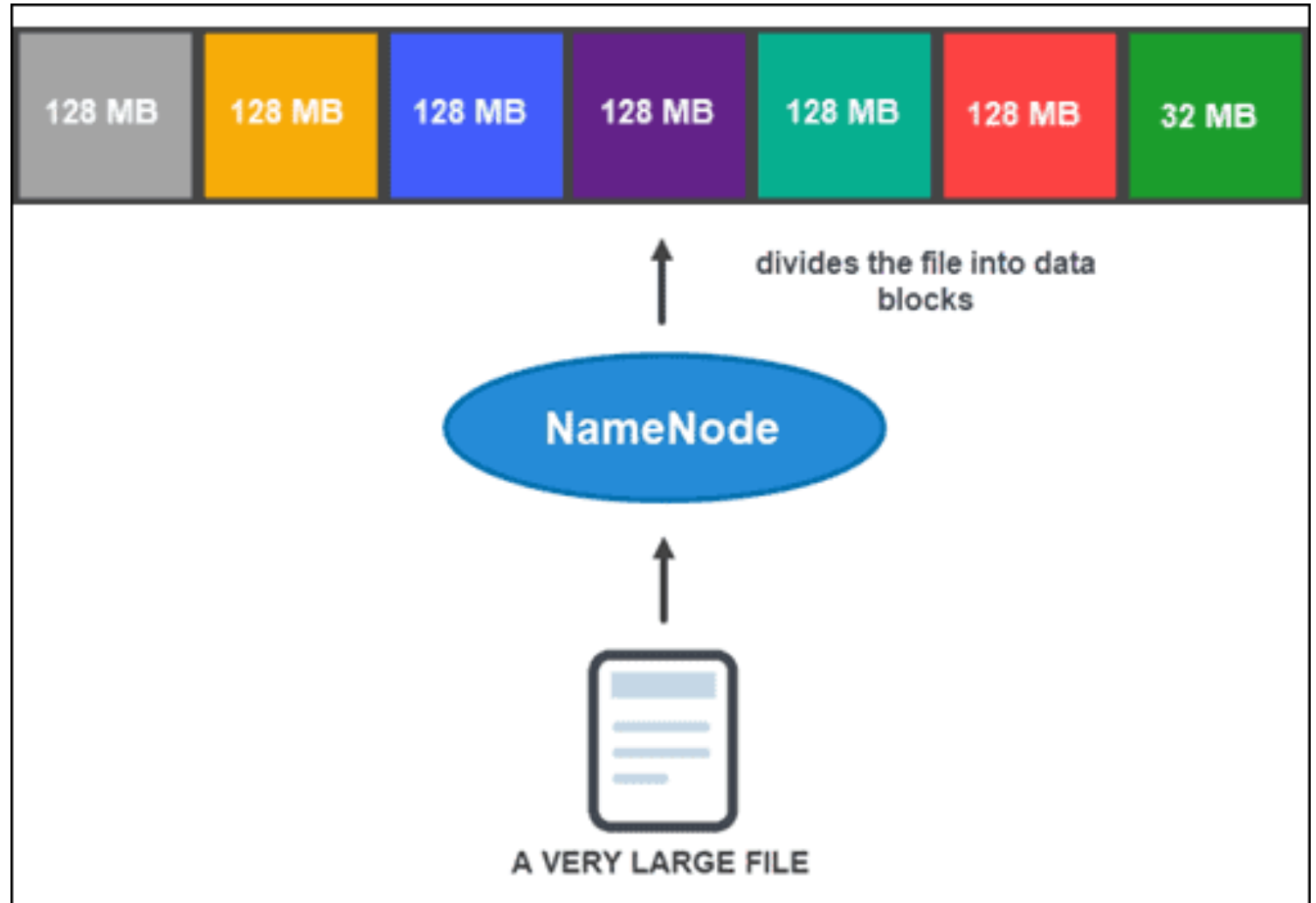


Из чего состоит кластер

- **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.

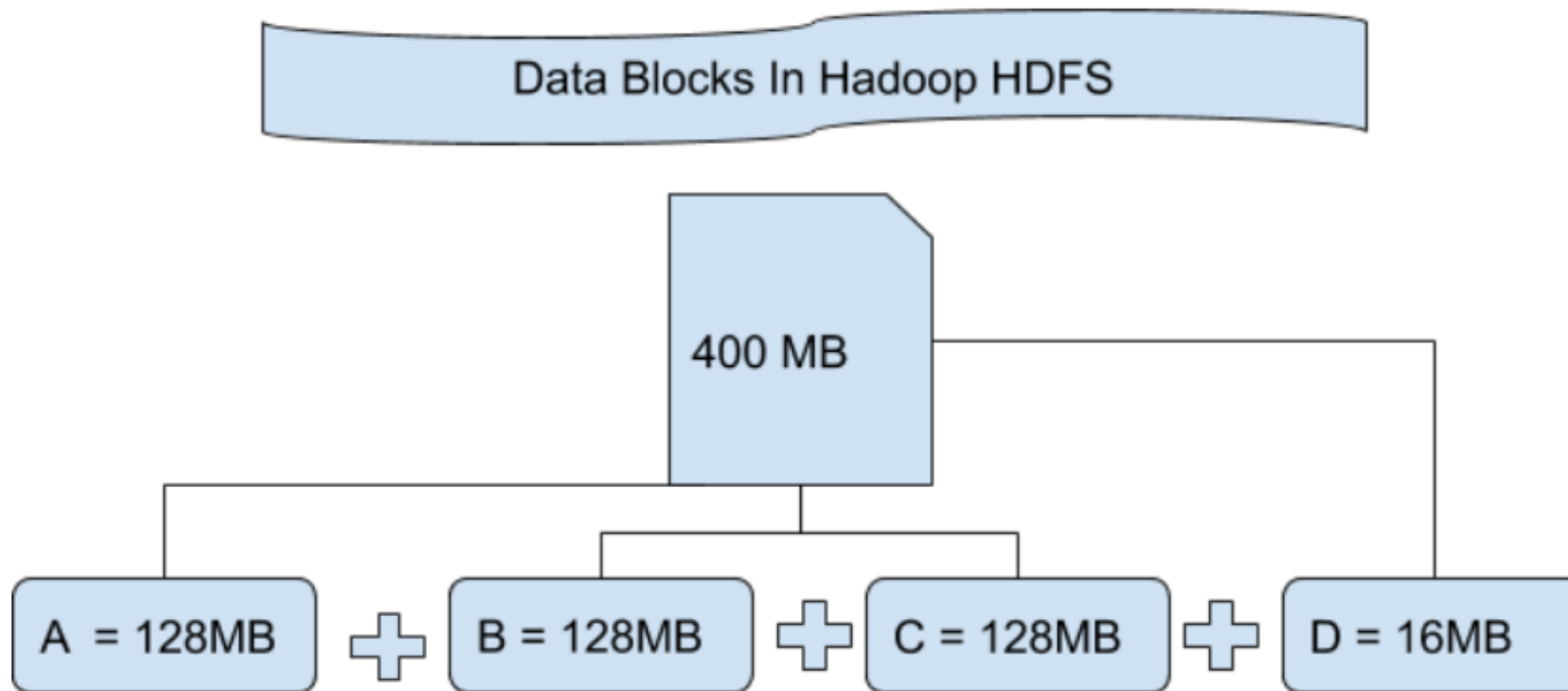
Хранение данных в HDFS

- 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

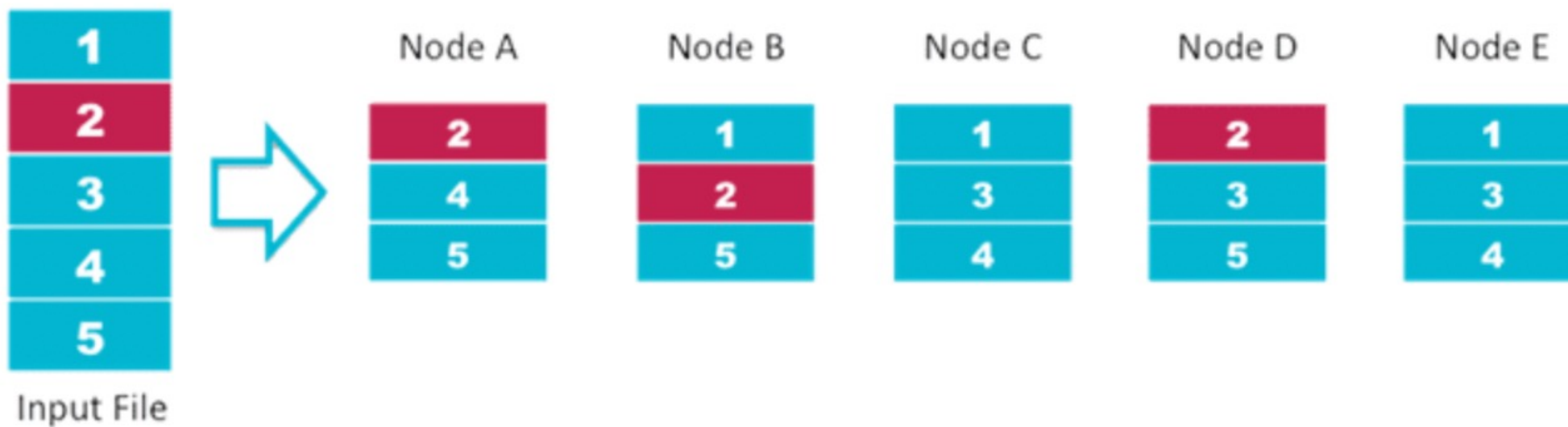
- Файл объемом 400 МБ будет разбит на 3 блока по 128 МБ и один на 16 МБ.



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

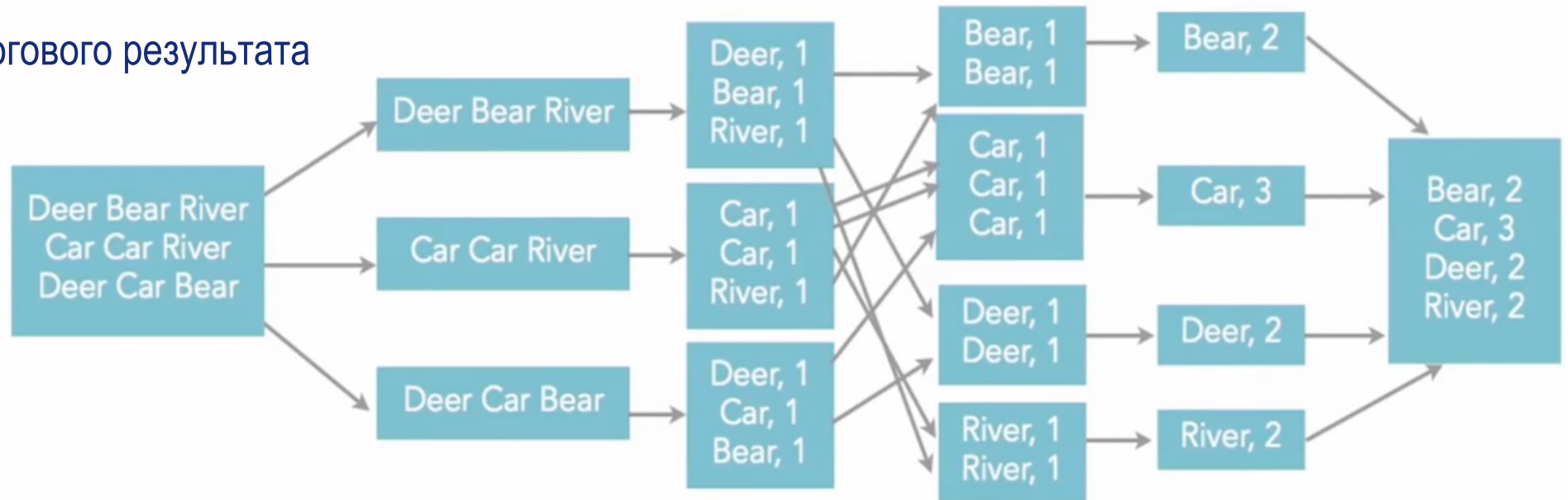
- Загрузить файл `hdfs dfs -put data.txt /tmp/data.txt`
- Скачать файл `hdfs dfs -get /tmp/data.txt /download`
- Посмотреть содержимое директории `hdfs dfs -ls /tmp`
- Создать директорию `hdfs dfs -mkdir /tmp/test`
- Удалить директорию `hdfs dfs -rm -R /tmp/test`

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"/>		-rw-r--r--		atguigu		supergroup		98 KB		Mar 02 19:38		1		128 MB		log-.1646221072146.lzo	
<input type="checkbox"/>		-rw-r--r--		atguigu		supergroup		38.14 KB		Mar 02 19:38		1		128 MB		log-.1646221086713.lzo	

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

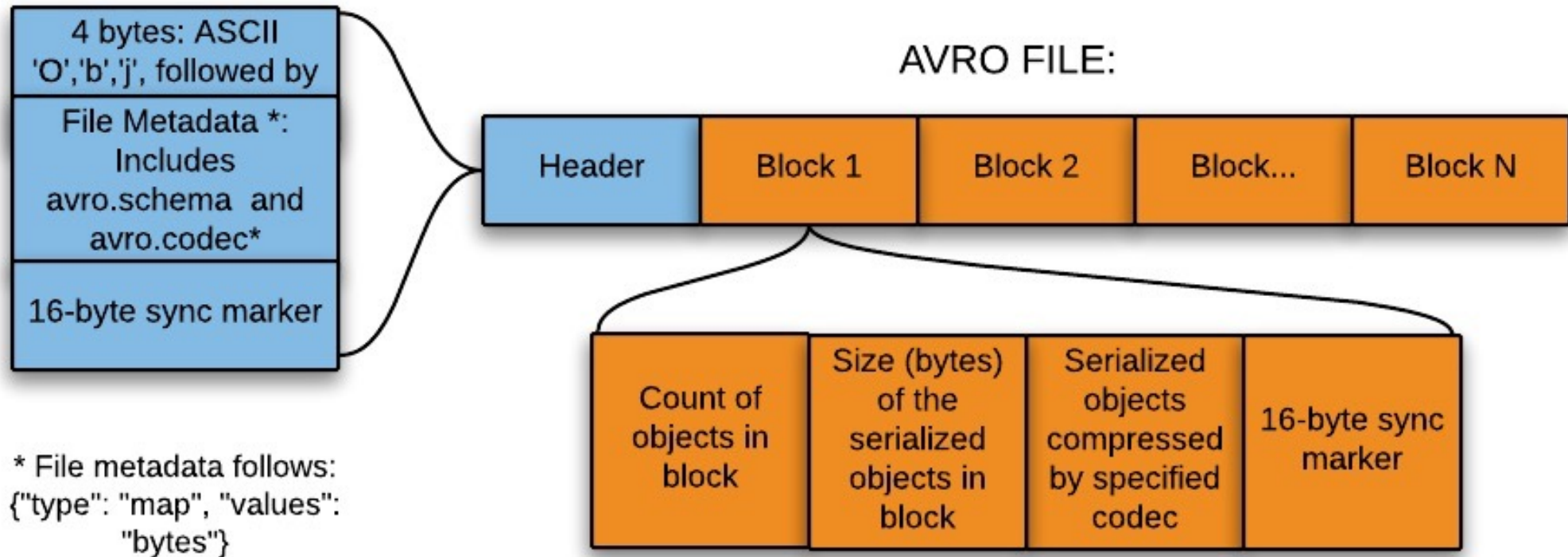
Row Store	
Row 1	India
	Chocolate
	1000
Row 2	India
	Ice-cream
	2000
Row 3	Germany
	Chocolate
	4000
Row 4	US
	Noodle
	500

		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

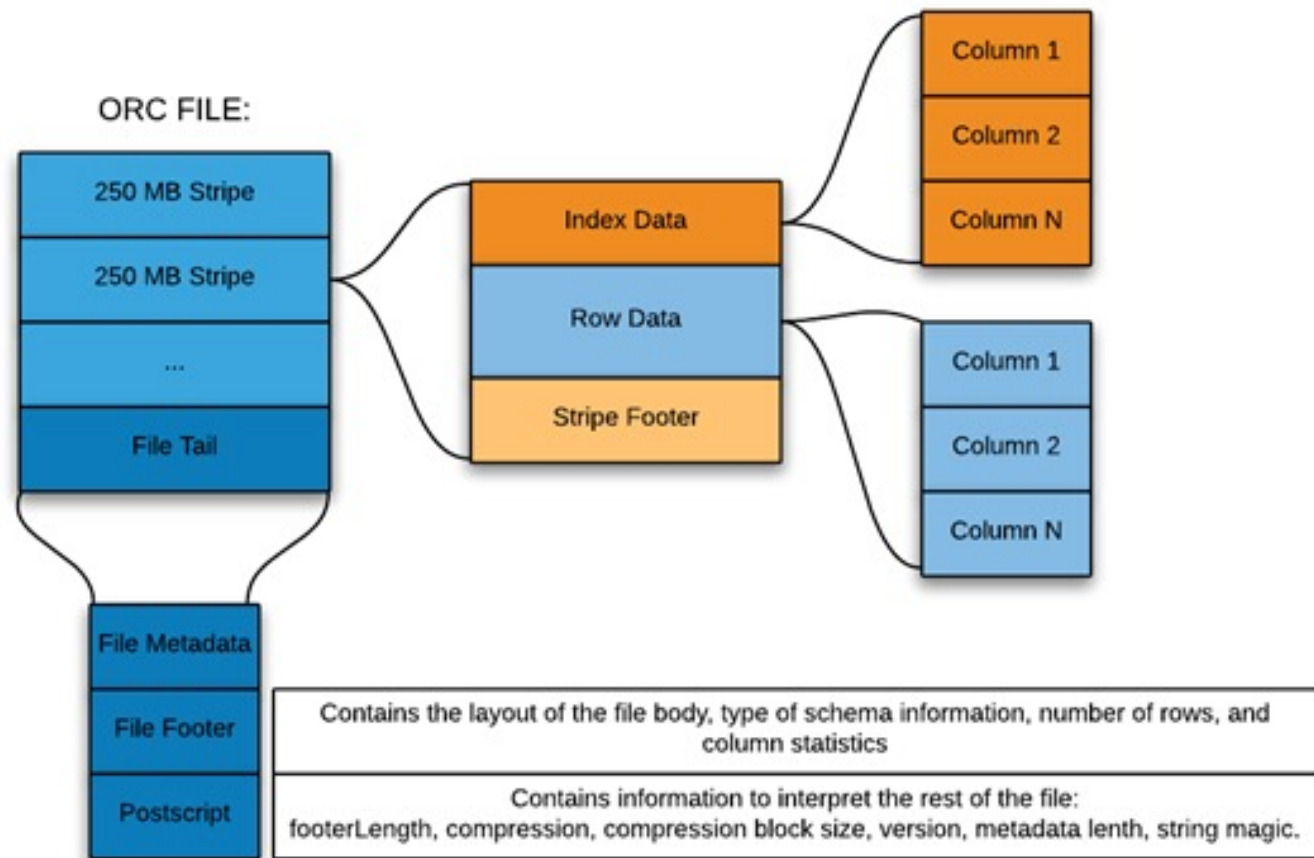
AVRO

- Отдельно хранит схему данных в формате JSON
- + сжимаемость, возможность изменения схемы данных
- - формат отсутствует по умолчанию, для чтения и записи необходимо устанавливать компонент Avro



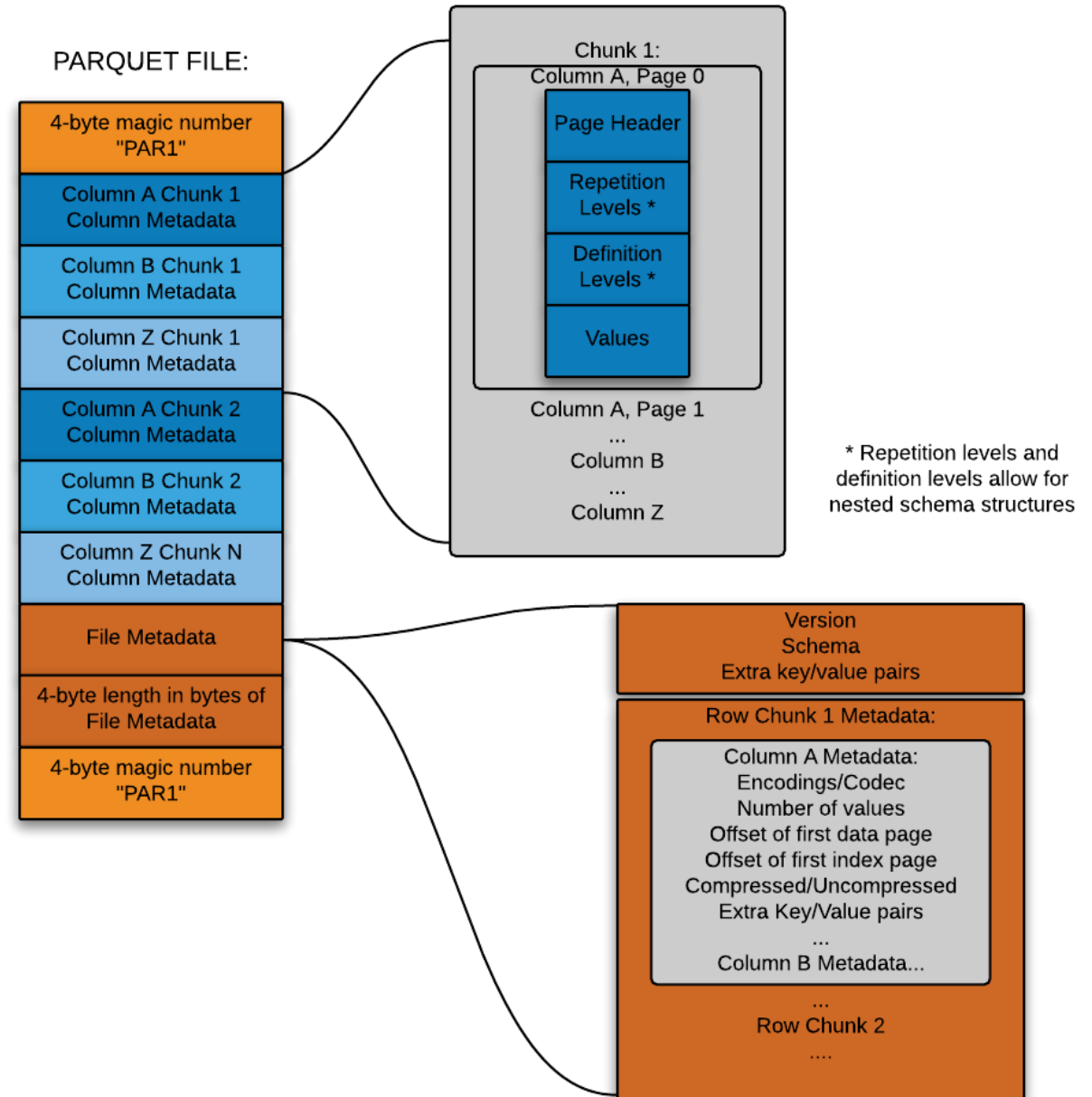
ORC (Optimized Row Columnar File)

- Оптимизированный строково-колоночный формат файлов;
- Данные разделены на полосы по 250 МБ;
- Колонки в полосах разделены друг от друга, что позволяет считывать данные избирательно.
- В футере файла записан список полос в файле, количество строк на полосу и тип данных каждого столбца. Там же записано результирующее значение count, min, max и sum по каждому столбцу.
- Индексные данные включают минимальные и максимальные значения для каждого столбца и позиции строк в каждом столбце. Индексы ORC используются только для выбора полос и групп строк, а не для ответа на запросы.



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.

Структура Parquet

Header

Row group

Row group

.....

Row group

Footer

<Column 1 Chunk 1 + Column Metadata>
<Column 2 Chunk 1 + Column Metadata>
...
<Column N Chunk 1 + Column Metadata>

<Column 1 Chunk 2 + Column Metadata>
<Column 2 Chunk 2 + Column Metadata>
...
<Column N Chunk 2 + Column Metadata>

<Column 1 Chunk M + Column Metadata>
<Column 2 Chunk M + Column Metadata>
...
<Column N Chunk M + Column Metadata>



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



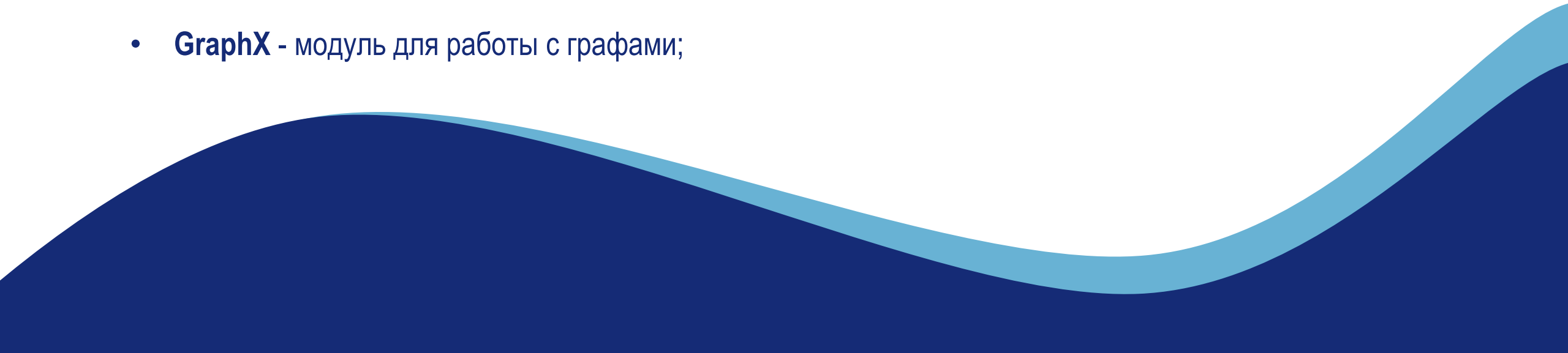
Машинное обучение

Spark VS Hadoop Map Reduce

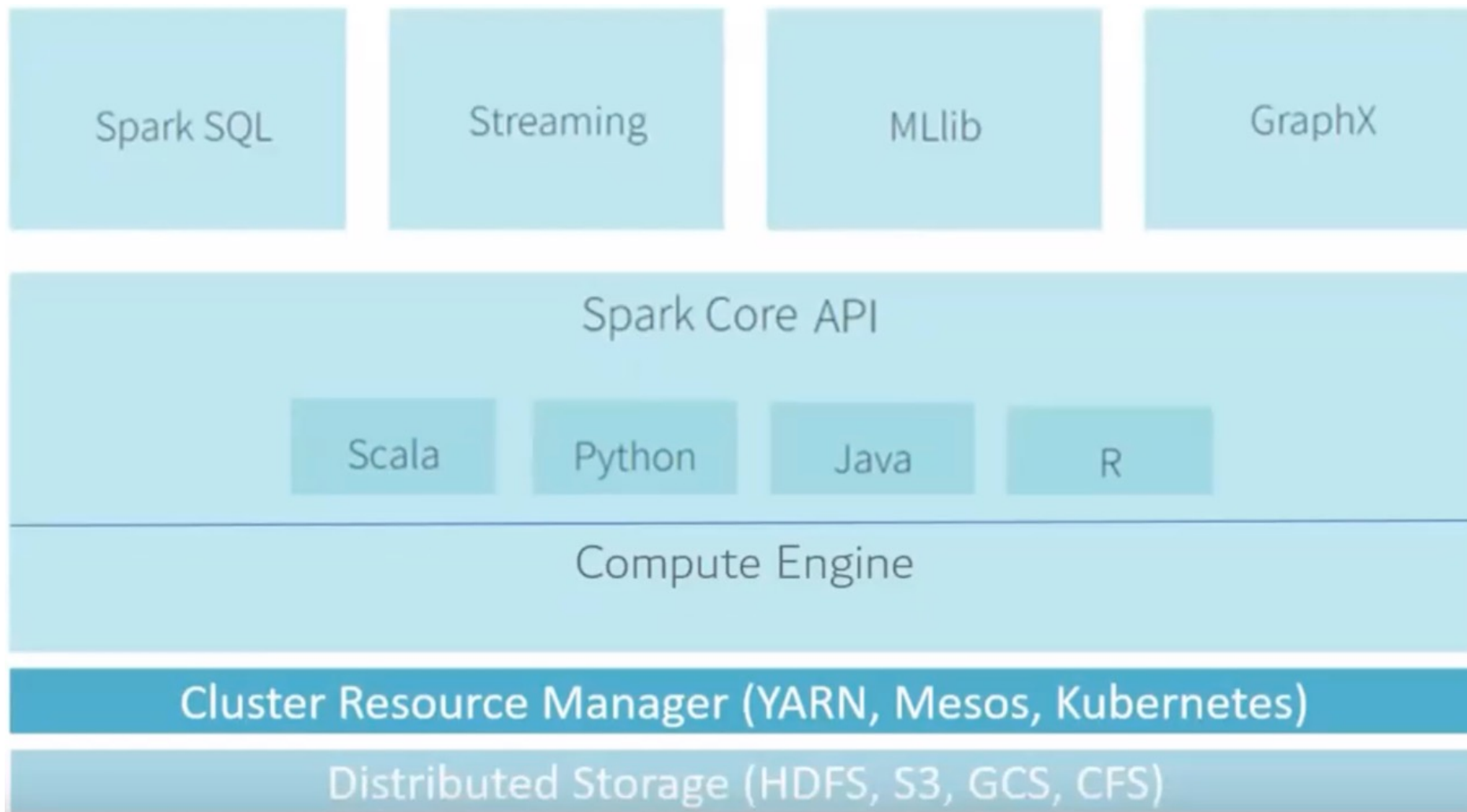
Spark быстрее ~ в 100 раз, чем Map Reduce

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

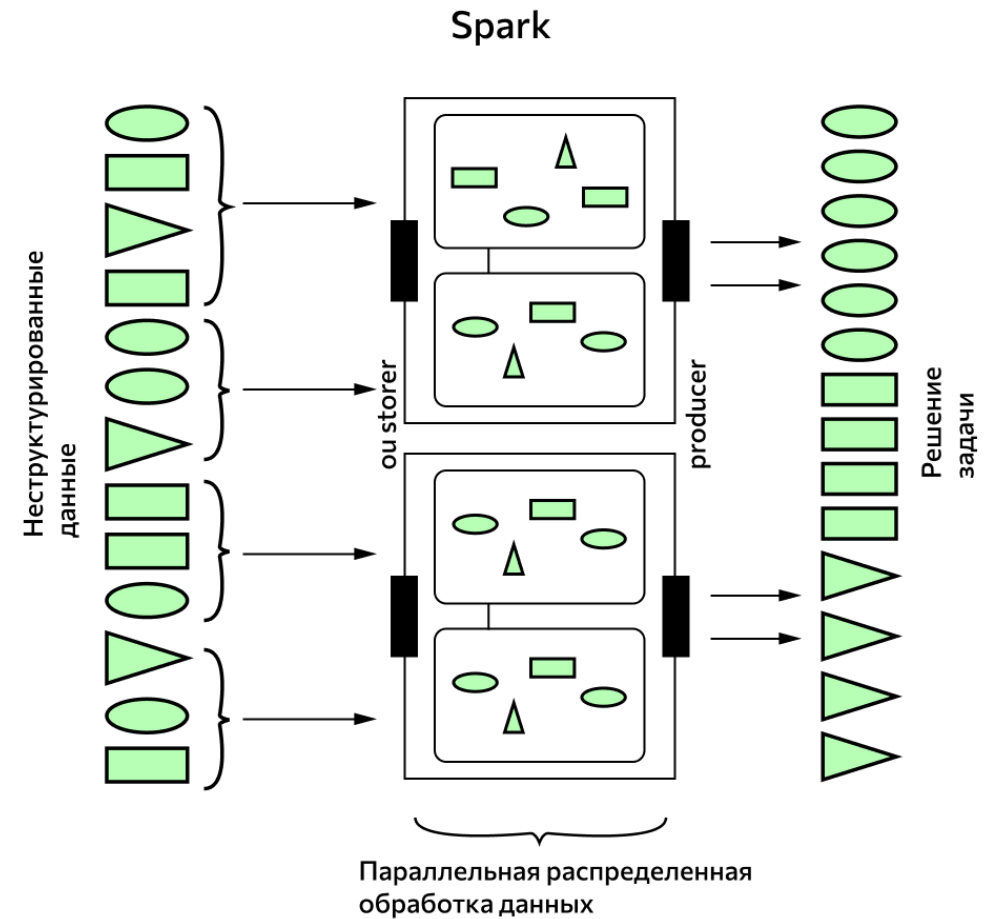
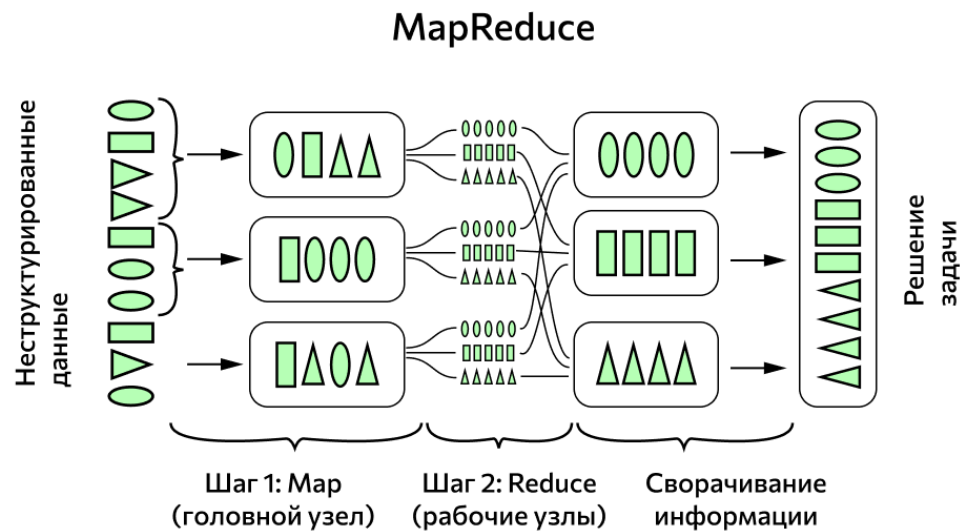
Архитектура Spark

- **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;
- 

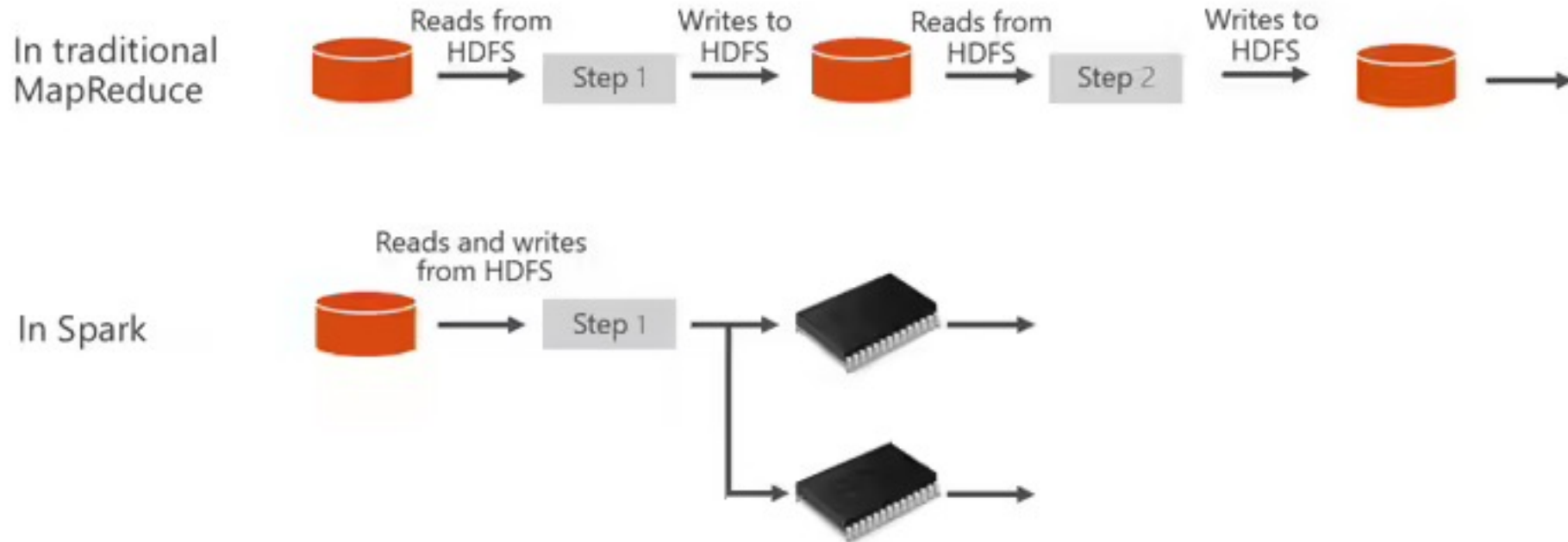
Архитектура Spark



Spark обрабатывает данные в режиме реального времени небольшими группами



Spark vs Map Reduce



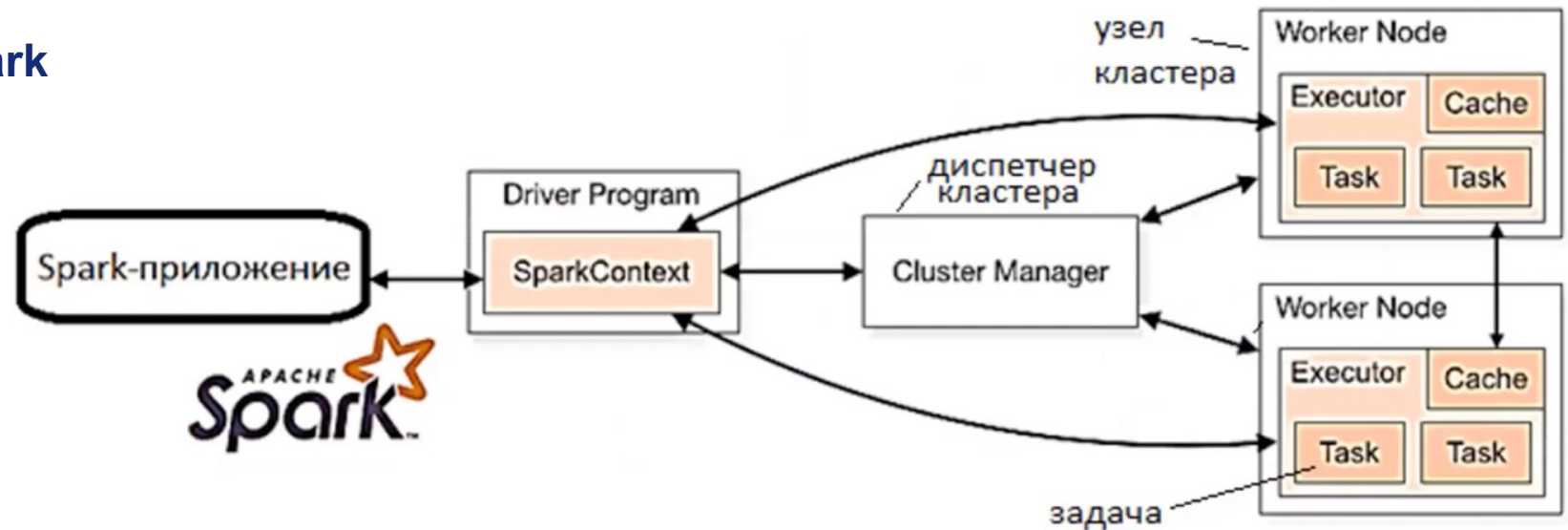
- Spark выполняет обработку данных в памяти и почти не обращается к диску.
- При возникновении ситуации, когда объем обрабатываемых данных превышает объем RAM, Spark сбрасывает часть обрабатываемых данных на диск.
- В Spark включены различные оптимизаторы, позволяющие сокращать количество обращений к диску.

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

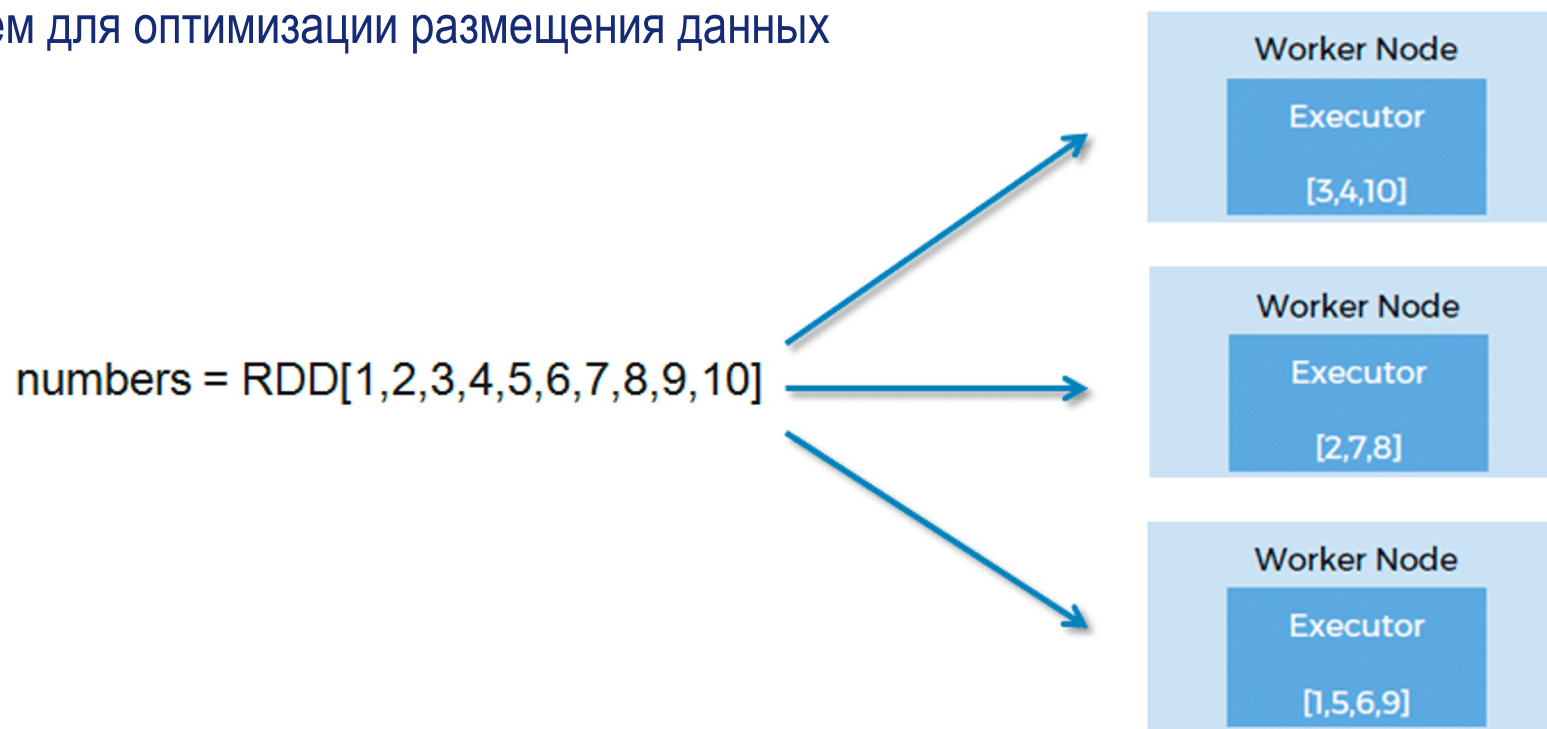
Форматы данных Spark

- 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



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.
- 


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
worker-20170410154728-10.0.10.73-39180	10.0.10.73:39180	ALIVE	2 (2 Used)	6.4 GB (2.0 GB Used)
worker-20170410154739-10.0.11.65-47952	10.0.11.65:47952	ALIVE	2 (2 Used)	6.4 GB (2.0 GB Used)
worker-20170410154744-10.0.12.48-46280	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
app-20170410162323-0002 (kill)	Calculate Balance	0	1024.0 MB	2017/04/10 16:23:23	centos	WAITING	2 s
app-20170410162322-0001 (kill)	Find Suspicious Transactions	3	1024.0 MB	2017/04/10 16:23:22	centos	RUNNING	3 s
app-20170410162321-0000 (kill)	Calculate Average Spending Per County	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
----------------	------	-------	-----------------	----------------	------	-------	----------

YARN UI



Logged in as: dr.who

All Applications

Cluster

About
Nodes
Node Labels
Applications
NEW
NEW SAVING
SUBMITTED
ACCEPTED
RUNNING
FINISHED
FAILED
KILLED

Scheduler

Tools

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved
8	0	0	8	0	0 B	48 GB	0 B	0	16	0

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes	Shutdown Nodes
2	0	0	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	Maximum Cluster Application Priority
Capacity Scheduler	memory-mb (unit=Mi), vcores	<memory:32, vCores:1>	<memory:24576, vCores:8>	100

Show 20 ▾ entries																			Search: <input type="text"/>		
ID ▾	User ▾	Name ▾	Application Type ▾	Queue ▾	Application Priority ▾	StartTime ▾	LaunchTime ▾	FinishTime ▾	State ▾	FinalStatus ▾	Running Containers ▾	Allocated CPU VCores ▾	Allocated Memory MB ▾	Reserved CPU VCores ▾	Reserved Memory MB ▾	% of Queue ▾	% of Cluster ▾	Progress ▾	Tracking UI ▾	Blacklisted Nodes ▾	
application_1659292729026_0009	user6	org.apache.spark.examples.SparkPi	SPARK	default	0	Sun Jul 31 14:48:53 -0400 2022	Sun Jul 31 14:48:53 -0400 2022	Sun Jul 31 14:49:03 -0400 2022	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0	<div></div>	History	0	
application_1659292729026_0008	user5	org.apache.spark.examples.SparkPi	SPARK	default	0	Sun Jul 31 14:48:29 -0400 2022	Sun Jul 31 14:48:29 -0400 2022	Sun Jul 31 14:48:39 -0400 2022	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0	<div></div>	History	0	
application_1659292729026_0007	user4	org.apache.spark.examples.SparkPi	SPARK	data_science	0	Sun Jul 31 14:48:01 -0400 2022	Sun Jul 31 14:48:01 -0400 2022	Sun Jul 31 14:48:12 -0400 2022	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0	<div></div>	History	0	
application_1659292729026_0006	user3	org.apache.spark.examples.SparkPi	SPARK	data_engineering	0	Sun Jul 31 14:47:08 -0400 2022	Sun Jul 31 14:47:08 -0400 2022	Sun Jul 31 14:47:19 -0400 2022	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0	<div></div>	History	0	
application_1659292729026_0004	user2	org.apache.spark.examples.SparkPi	SPARK	adhoc	10	Sun Jul 31 14:45:41 -0400 2022	Sun Jul 31 14:45:41 -0400 2022	Sun Jul 31 14:45:52 -0400 2022	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0	<div></div>	History	0	
application_1659292729026_0003	user1	org.apache.spark.examples.SparkPi	SPARK	adhoc	10	Sun Jul 31 14:44:57 -0400 2022	Sun Jul 31 14:44:57 -0400 2022	Sun Jul 31 14:45:08 -0400 2022	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0	<div></div>	History	0	
application_1659292729026_0002	hadoop	org.apache.spark.examples.SparkPi	SPARK	adhoc	10	Sun Jul 31 14:44:30 -0400 2022	Sun Jul 31 14:44:30 -0400 2022	Sun Jul 31 14:44:42 -0400 2022	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0	<div></div>	History	0	
application_1659292729026_0001	hive	HIVE-38f2a213-4ae9-4a9a-935c-9ecf0ee70291	TEZ	default	0	Sun Jul 31 14:39:44 -0400 2022	Sun Jul 31 14:39:45 -0400 2022	Sun Jul 31 14:44:54 -0400 2022	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0	<div></div>	History	0	

Showing 1 to 8 of 8 entries

First Previous 1 Next Last

HUE UI

Query ▾

Jobs

Editor

Dashboard

Scheduler

Documents

Files

S3

Tables

Indexes

Jobs

Streams

HBase

Security

Importer

default

Tables (5) +

customers

id (int)

name (string)

email_preferences (struct)

addresses (map)

orders (array)

k8s_logs

sample_07

sample_08

web_logs

version (bigint)

app (string)

bytes (smallint)

city (string)

client_ip (string)

code (tinyint)

country_code (string)

country_code3 (string)

country_name (string)

device_family (string)

extension (string)

latitude (float)

longitude (float)

method (string)

os_family (string)

os_major (string)

protocol (string)

Impala

Add a name...

Add a descriptio...

0.92s Database default ▾

```
15 WHERE a.key = 'shipping' and a.zip_code = '76710';
16
17
18
19 -- Compute total amount per order for all customers
20 SELECT
21   c.id AS customer_id,
22   c.name AS customer_name,
23   o.order_id,
24   v.total
25 FROM
26   customers c,
27   c.orders o,
28   (SELECT SUM(price * qty) total FROM o.items) v;
```

Query 034d413ec7474ed5:4249de1000000000 100% Complete 034d413ec7474ed5:4249de1000000000

Query 034d413ec7474ed5:4249de1000000000 100% Complete (1 out of 1)

Query 034d413ec7474ed5:4249de1000000000 100% Complete (1 out of 1)

Query History

Saved Queries

Results (106)

Execution Analysis

	customer_id	customer_name	order_id	total
1	75012	Dorothy Wilk	4056711	918
2	75012	Dorothy Wilk	J882C2	96
3	17254	Martin Johnson	I72T39	18
4	12532	Melvin Garcia	PB6268	68
5	12532	Melvin Garcia	B8623C	2507

Tables Statement 3/3

default.customers

id int

name string

email_preferences struct

addresses map

orders array



Практика

Написание запросов в Google Colab на PySpark