МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«МИРЭА – Российский технологический университет»**
**РТУ МИРЭА**

# ЛЕКЦИОННЫЕ МАТЕРИАЛЫ

## Технологии хранения в системах кибербезопасности

*(наименование дисциплины (модуля) в соответствии с учебным планом)*

| | |
|---|---|
| Уровень | бакалавриат |
| | *(бакалавриат, магистратура, специалитет)* |
| Форма обучения | очная |
| | *(очная, очно-заочная, заочная)* |
| Направление(-я) подготовки | 10.05.04 Информационно-аналитические системы безопасности |
| | *(код(-ы) и наименование(-я))* |
| Институт | Кибербезопасности и цифровых технологий (ИКБ) |
| | *(полное и краткое наименование)* |
| Кафедра | КБ-2 «Прикладные информационные технологии» |
| | *(полное и краткое наименование кафедры, реализующей дисциплину (модуль))* |
| Лектор | к.т.н., Селин Андрей Александрович |
| | *(сокращенно – ученая степень, ученое звание; полностью – ФИО)* |

Используются в данной редакции с учебного года 2024/2025

*(учебный год цифрами)*

Проверено и согласовано «____» _____2024 г. _____ А.А. Бакаев

*(подпись директора Института/Филиала с расшифровкой)*

Москва 2024 г.

# Технологии хранения в системах кибербезопасности

2024 год

# Лекция 13.
# Apache Spark

# Учебные вопросы лекции:

1. Архитектура Spark
2. Особенности Apache Spark
3. MapReduce и Spark
5. Основные концепции Spark

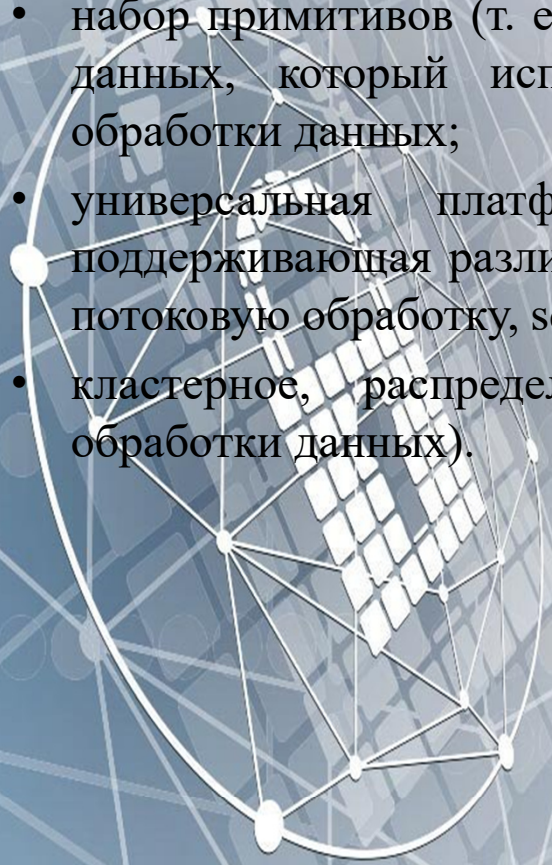
# Введение

Apache Spark – это BigData фреймворк с открытым исходным кодом для распределённой пакетной и потоковой обработки неструктурированных и слабоструктурированных данных, входящий в экосистему проектов Hadoop. Apache Spark известен как быстрый, простой в использовании и универсальный механизм для обработки больших данных, имеющий встроенные модули для потоковой передачи, SQL, машинного обучения (ML) и обработки графов.

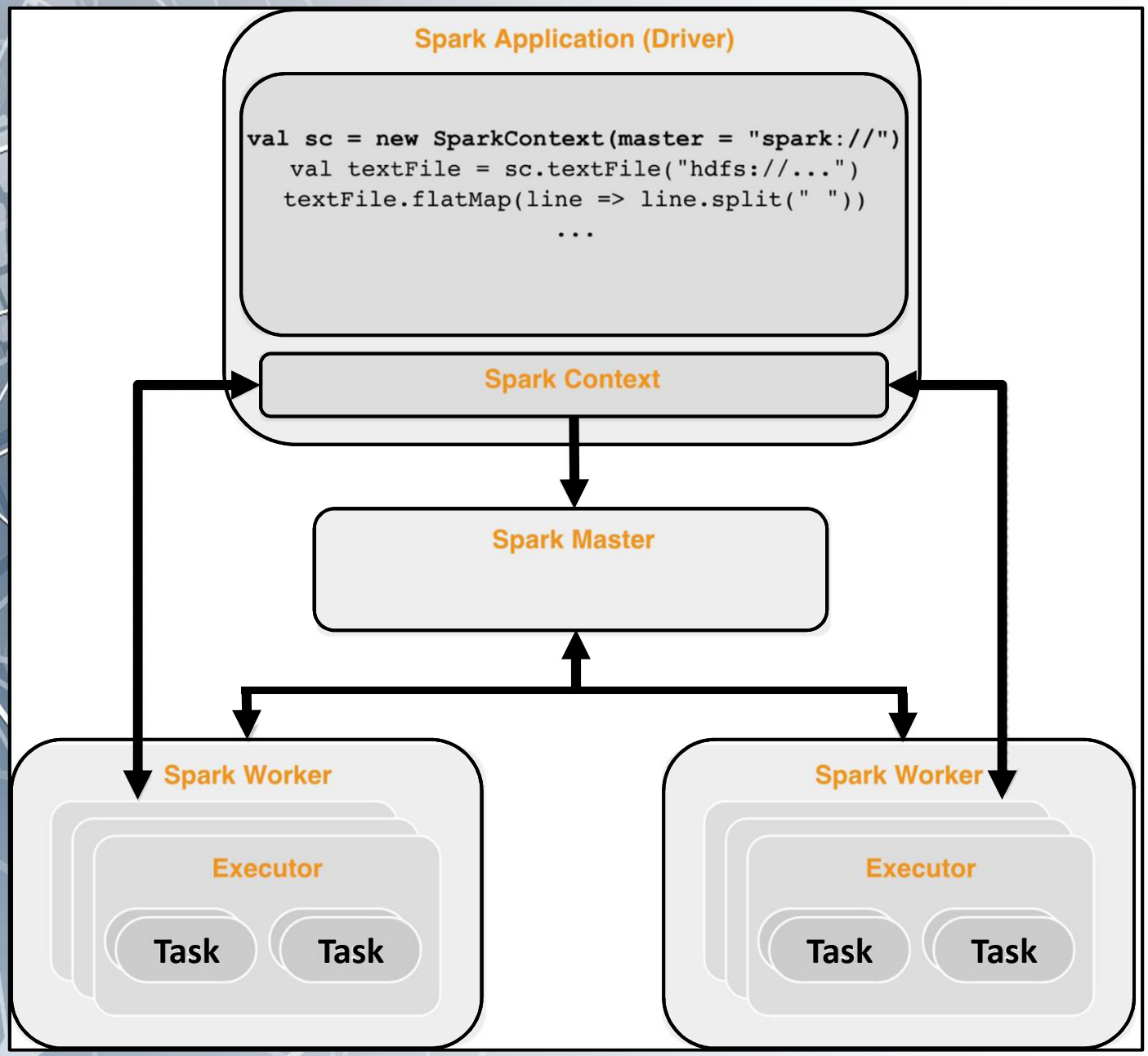
# Архитектура Spark

Apache Spark может рассматриваться как:

- набор примитивов (т. е. фреймворк) для обслуживания всего цикла обработки данных, который используется для написания высокоуровневой логики обработки данных;
- универсальная платформа, предоставляющая различные средства и поддерживающая различные режимы обработки данных: пакетную обработку, потоковую обработку, sql-запросы, обработку графов;
- кластерное, распределенное приложение (в момент непосредственной обработки данных).



# Архитектура Spark



# Архитектура Spark

**Driver** – компонент, ответственный за:

- отслеживание состояния обработки, генерацию задач и их перезапуск в случае отказов;
- оркестровку вычислений – назначение задач на вычислительные ресурсы (занимаемые с помощью executor'ов) с учетом их использования данных;
- контроль за вычислительными ресурсами, включая их выделение и возвращение менеджеру ресурсов;

**Executor** – компонент, ответственный за выполнение вычислений над данными на ресурсах вычислительного узла, где он располагается. В его задачи входит:

- контроль за расходом вычислительных ресурсов;
- хранение данных и предоставление доступа к ним, включая загрузку данных с диска или их десериализацию, для выполнения обработки;
- запуск выполнения задач и сохранение их результатов.

Driver обращается к своим executor'ам каждый раз, когда ему нужно запустить или перезапустить задачи, а также принимает и обрабатывает регулярные периодические отчеты от своих executor'ов, реализуемые через архитектурный паттерн heartbeat. Поэтому не рекомендуется размещать driver на узле вне кластера, который имеет малую скорость соединения с кластером, например, на узле, отделенном от кластера сетью Интернет.

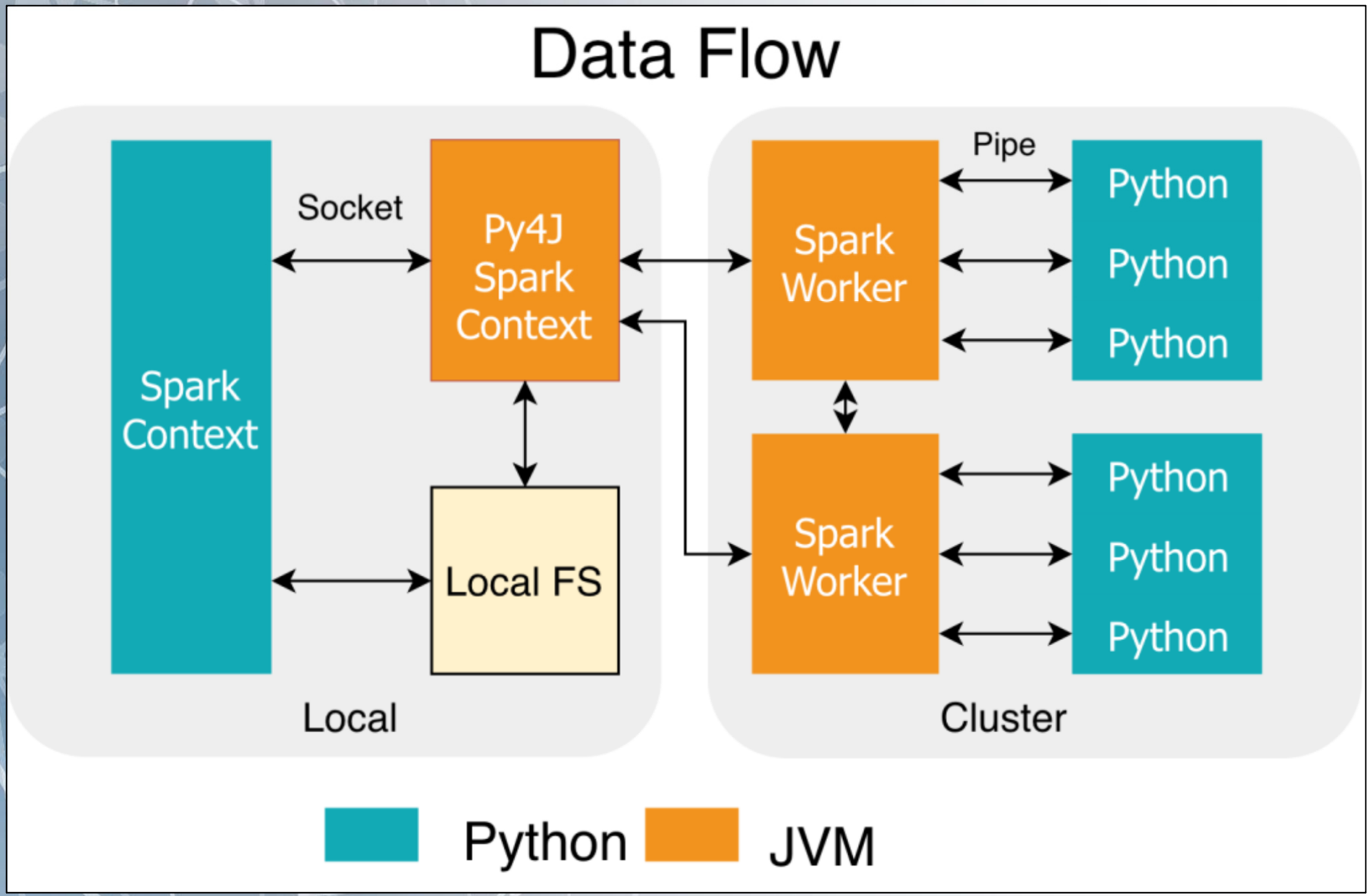
# Архитектура Spark

Современная обработка данных предполагает совместное использование кластера несколькими приложениями, в том числе несколькими разными приложениями Spark. Отдельные приложения Spark имеют абсолютно независимые наборы executor'ов, и executor не может переходить от одного драйвера к другому. Но executor'ы разных приложений могут располагаться на одних и тех же узлах кластера, если ресурсы узлов это позволяют.

И driver, и executor требуют для своего исполнения JVM, однако driver может становится частью других приложений, используясь как программная библиотека – в этом случае запуск приложения Spark происходит программно с помощью создания объекта специального класса SparkContext или SparkSession. Такой способ будет более подробно рассмотрен в следующем разделе. Executor всегда является отдельным JVM процессом, запущенным на узле кластера или в контейнере на узле кластера.

В случае, когда приложение Spark является самостоятельным и имеет свой собственный jar файл с определенным Main классом, его можно запустить с помощью специального скрипта spark-submit из стандартного дистрибутива Spark.

# Архитектура Spark



# Особенности Apache Spark

**Spark – всё-в-одном для работы с большими данными**. Spark создан для того, чтобы помогать решать широкий круг задач по анализу данных, начиная с простой загрузки данных и SQL-запросов и заканчивая машинным обучением и потоковыми вычислениями, при помощи одного и того же вычислительного инструмента с неизменным набором API. Главный инсайт этой программной многозадачности в том, что задачи по анализу данных в реальном мире—будь они интерактивной аналитикой в таком инструменте, как Jupyter Notebook, или же обычным программированием для выпуска приложений – имеют тенденцию требовать сочетания множества разных типов обработки и библиотек. Целостная природа Spark делает решение этих заданий проще и эффективнее.

Например, если вы загружаете данные при помощи SQL-запроса и потом оцениваете модель машинного обучения при помощи библиотеки Spark ML, движок может объединить все эти шаги в один проход по данным. Более того, для исследователей данных может быть выгодно применять объединённый набор библиотек (например, Python или R) при моделировании, а веб-разработчикам пригодятся унифицированные фреймворки, такие как Node.js или Django.

# Особенности Apache Spark

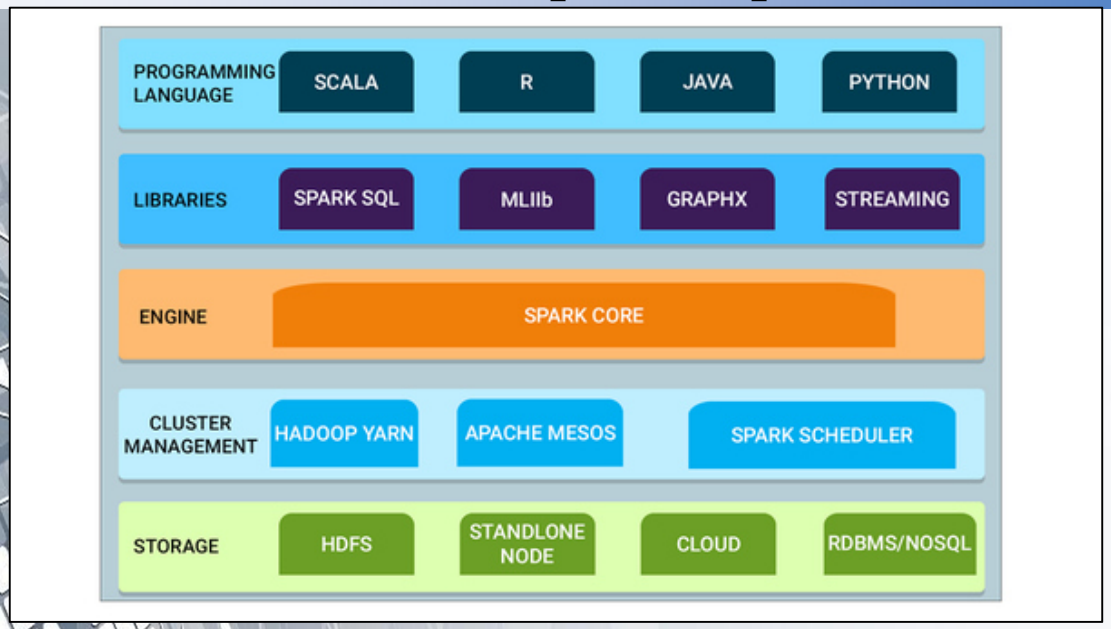
**Spark оптимизирует своё машинное ядро для эффективных вычислений –** “то есть Spark только управляет загрузкой данных из систем хранения и производит вычисления над ними, но сам не является конечным постоянным хранилищем. Со Spark можно работать, когда имеешь дело с широким разнообразием постоянных систем хранения, включая системы облачного типа по примеру Azure Storage и Amazon S3, распределенные файловые системы, такие как Apache Hadoop, пространства для хранения ключей, как Apache Cassandra, и последовательностей сообщений, как Apache Kafka. И всё же, Spark не сохраняет данные сам по себе надолго и не поддерживает ни одну из этих систем. Главная причина здесь в том, что большинство данных уже находится в нескольких системах хранения. Перемещать данные дорого, поэтому Spark только обрабатывает данные при помощи вычислительных операций, не важно, где они при этом находятся.” (из книги Databricks). Сфокусированность Sparks на вычислениях отличает его от более ранних программных платформ по обработке больших данных, например от Apache Hadoop. Это ПО включает в себя и систему хранения (HFS, сделанную для недорогих хранилищ на кластерах продуктовых серверов Defining Spark 4) и вычислительную систему (MapReduce). Между собой они интегрируются достаточно хорошо.

# Особенности Apache Spark

**Библиотеки Spark дарят очень широкую функциональность –** сегодня стандартные библиотеки Spark являются главной частью этого проекта с открытым кодом. Ядро Spark само по себе не слишком сильно изменялось с тех пор, как было выпущено, а вот библиотеки росли, чтобы добавлять ещё больше функциональности. И так Spark превратился в мультифункциональный инструмент анализа данных. В Spark есть библиотеки для SQL и структурированных данных (Spark SQL), машинного обучения (MLlib), потоковой обработки (Spark Streaming и более новый Structured Streaming) и аналитики графов (GraphX). Кроме этих библиотек есть сотни открытых сторонних библиотек, начиная от тех, что работают с коннекторами и до вариантов для различных систем хранения и алгоритмов машинного обучения.

**Поддержка нескольких языков разработки –** Scala, Java, Python и R

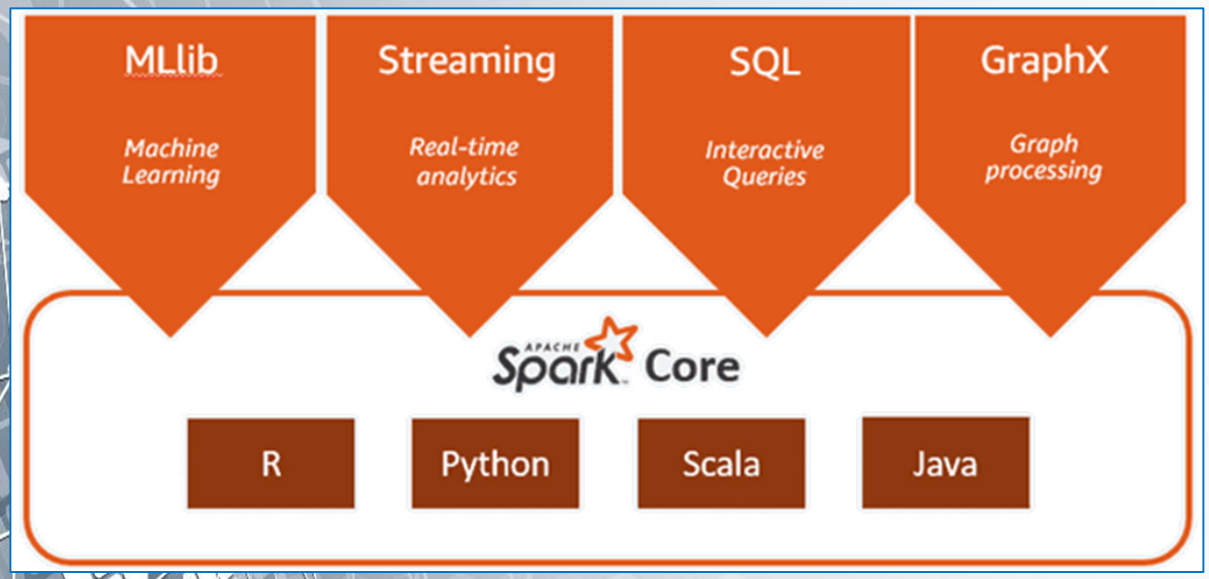
# Особенности Apache Spark




**Spark Core** – это базовый инструмент для крупномасштабной параллельной и распределённой обработки данных. Кроме того, есть дополнительные библиотеки, встроенные поверх ядра. Они позволяют разделить рабочие нагрузки для стриминга, SQL и машинного обучения. Отвечают за управление памятью и восстановление после ошибок, планирование, распределение и мониторинг задач в кластере, а также взаимодействие с системами хранения.

**Cluster management** (управление кластером) – контроль кластера используется для получения кластерных ресурсов, необходимых для решения задач. Spark Core работает на разных кластерных контроллерах, включая Hadoop YARN, Apache Mesos, Amazon EC2 и встроенный кластерный менеджер Spark. Такая служба контролирует распределение ресурсов между приложениями Spark. Кроме того, Spark может получать доступ к данным в HDFS, Cassandra, HBase, Hive, Alluxio и любом хранилище данных Hadoop.

# Особенности Apache Spark



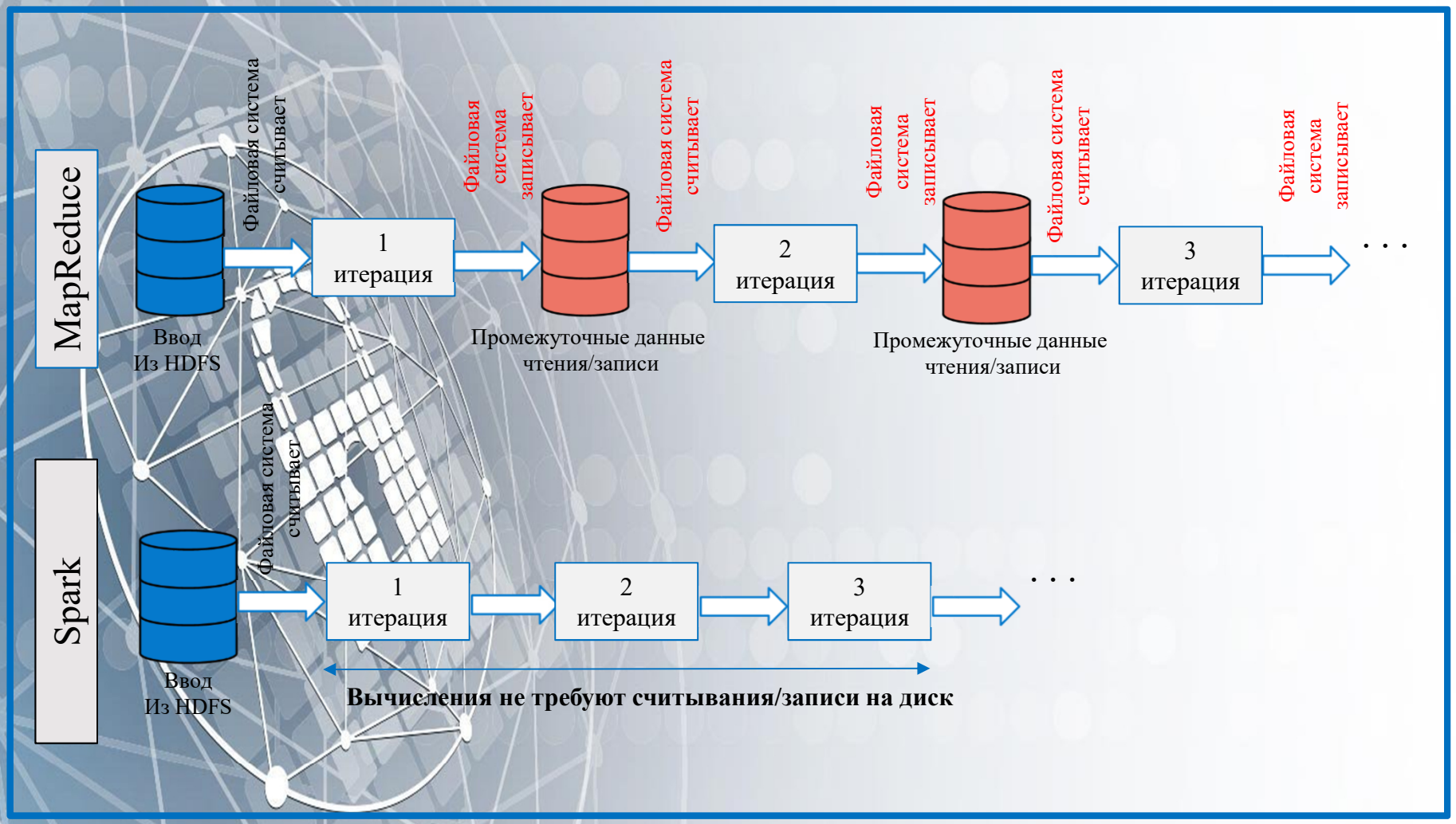

**Spark Streaming** – это компонент Spark, который нужен для обработки потоковых данных в реальном времени.

**Spark SQL** – это новый модуль в Spark. Он интегрирует реляционную обработку с API функционального программирования в Spark. Поддерживает извлечение данных, как через SQL, так и через Hive Query Language. API DataFrame и Dataset в Spark SQL обеспечивают самый высокий уровень абстракции для структурированных данных.

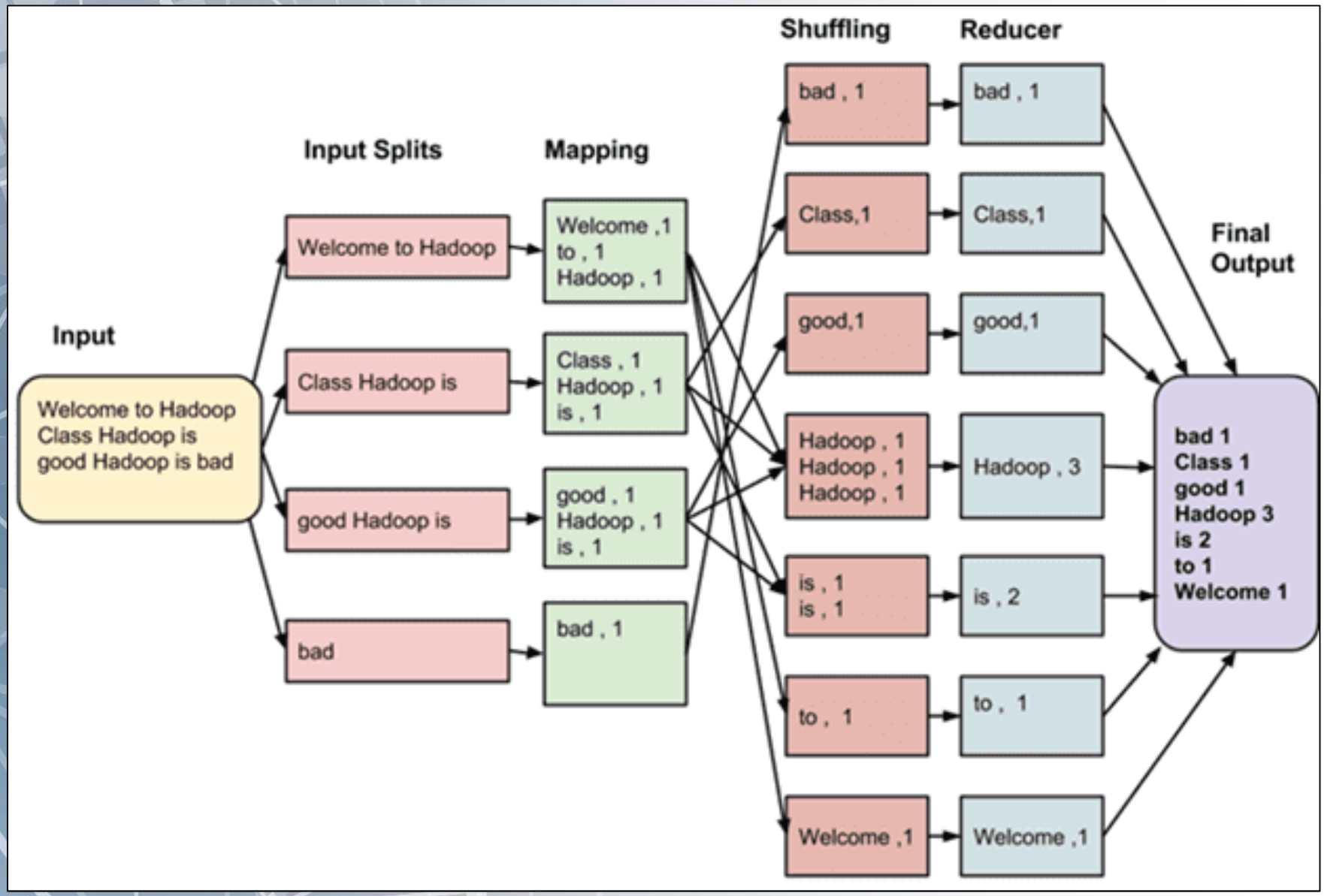
**GraphX** – API Spark для графов и параллельных вычислений с графами. Так что он является расширением Spark RDD с графом устойчивого распределения свойств (Resilient Distributed Property Graph).

**MLlib** (Машинное обучение): MLlib расшифровывается как библиотека машинного обучения.

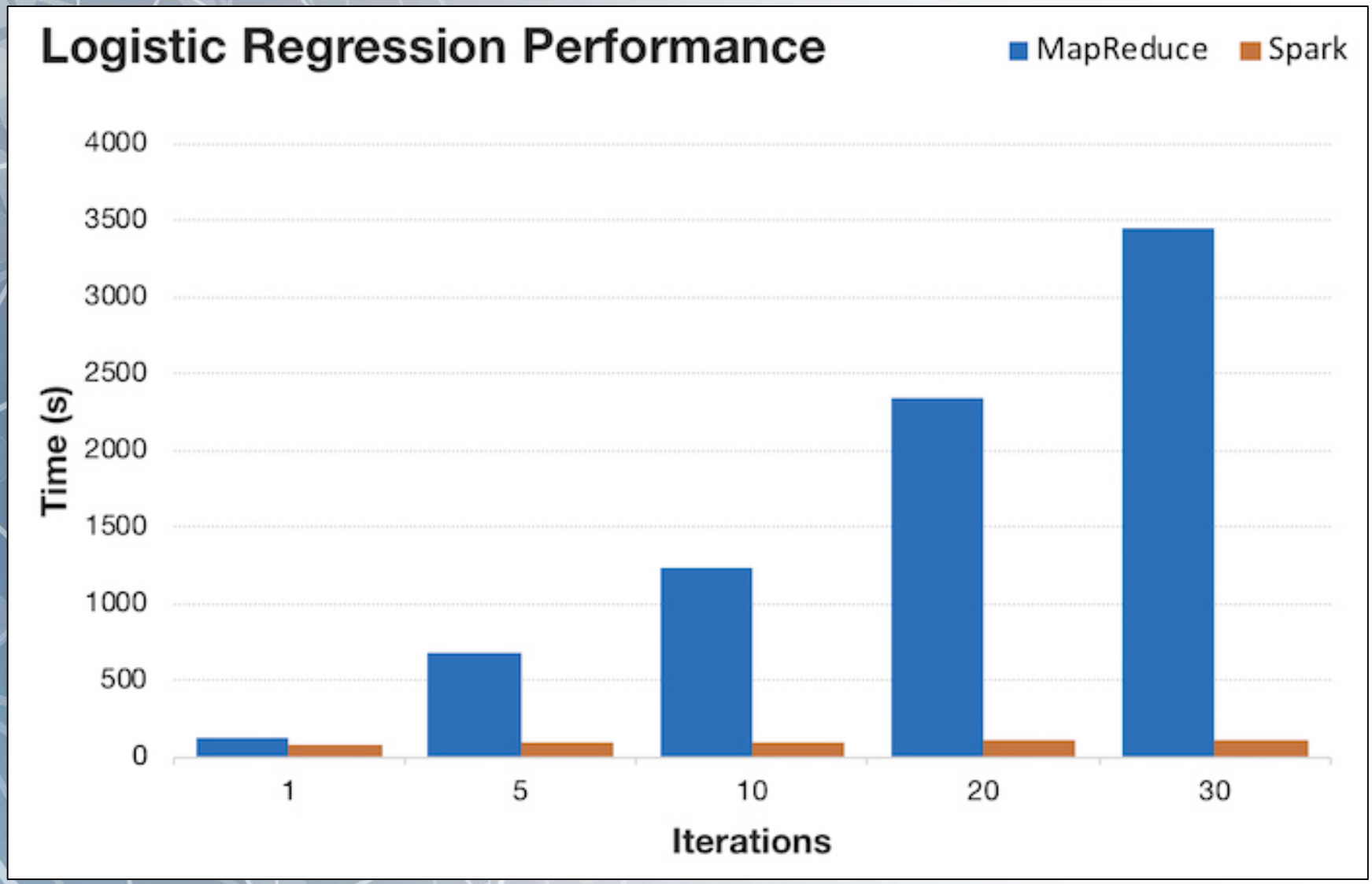
# Отличия MapReduce и Spark



# -=MapReduce для тех, кто забыл (Лекция №7)=-



# Отличия MapReduce и Spark

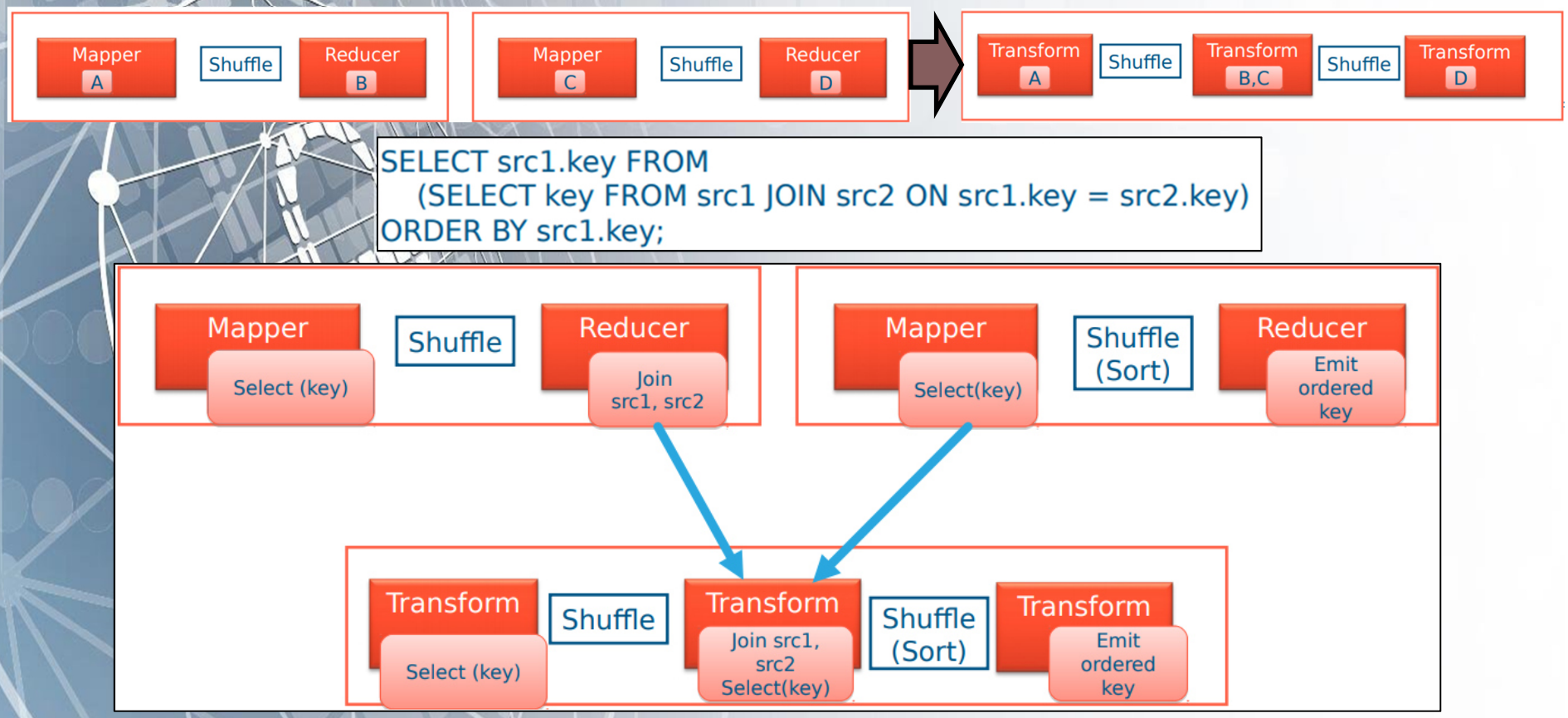


# Отличия MapReduce и Spark

| | MapReduce | Spark |
|---|---|---|
| Данные | Файл | RDD сохраняемые в памяти узлов |
| Программа | Map, Shuffle, Reduce в заданном порядке | Трансформации в любом заданном порядке, нет деления на виды. |
| Жизненный цикл | Задача - Java процессы, которые запускаются и выгружаются для каждого шага | Задача – выполняется на доступных, долгоживущих процессах Executors |

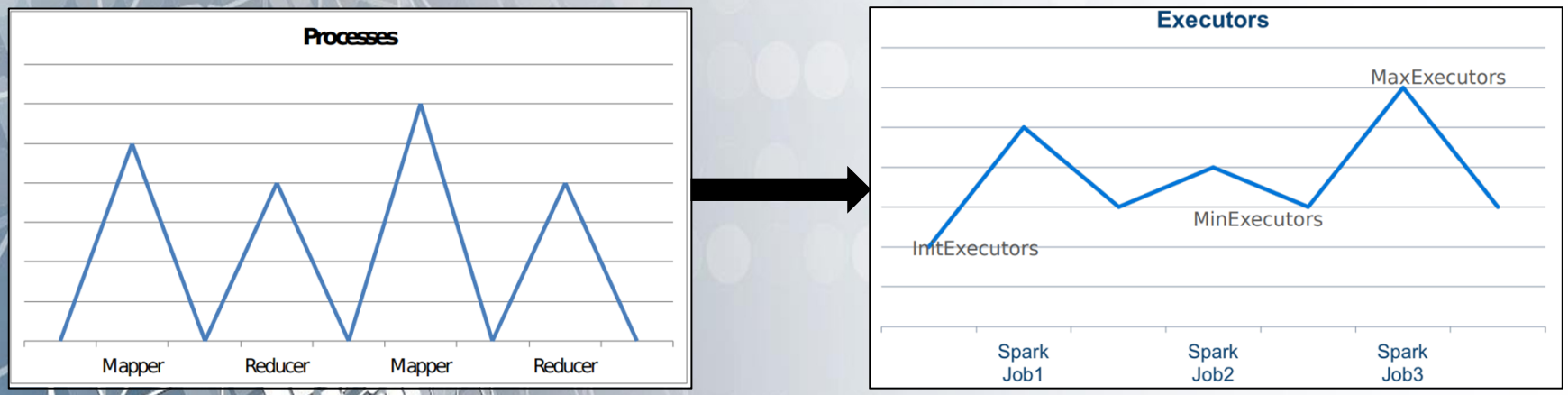
# Отличия MapReduce и Spark

Меньше шагов – Spark job это набор трансформаций (без разделения Mapper - Reducer) разделенных Shuffle.



# Отличия MapReduce и Spark

## Жизненный цикл процессов




**MapReduce** – каждый шаг запускает и удаляет процессы Mapper и Reducer

**Spark** – каждый Executor (исполнитель) является долгоживущим процессом и может в течение жизни исполнять одну или несколько задач последовательно и параллельно (executor cores)

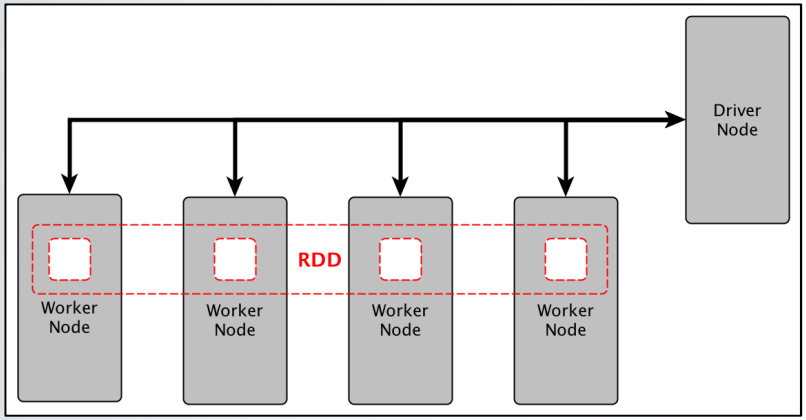
# Основные концепции Spark

## RDD и граф преобразований

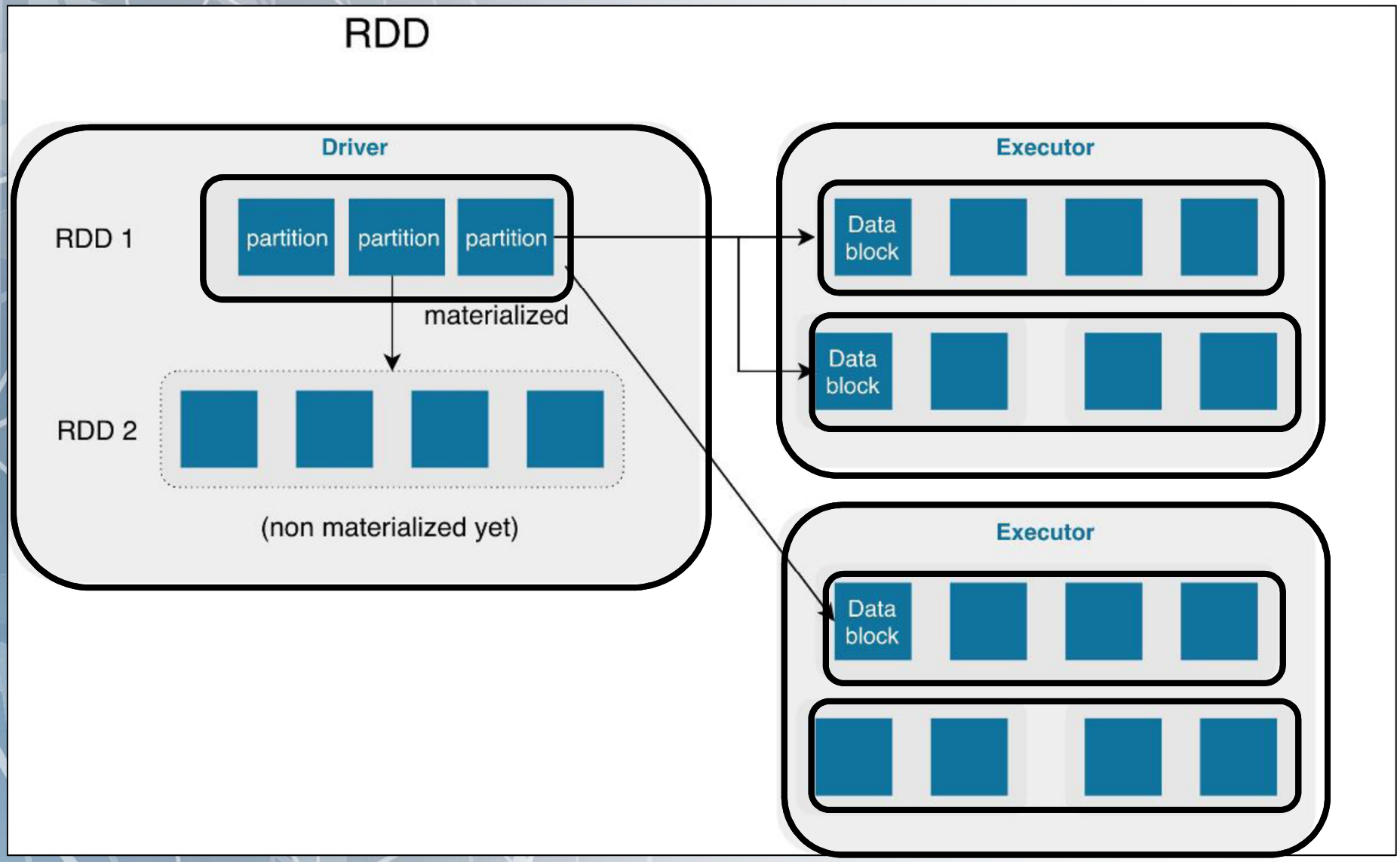
Для представления данных Spark использует концепцию RDD (Resilient Distributed Datasets). RDD – это абстракция набора данных, состоящего из записей одного произвольного типа и разделенного на части, которые размещены по всему кластеру. RDD содержит метаинформацию о наборе данных, но не сами данные, и хранится в driver приложения Spark во время его выполнения и только во время его выполнения.

Непосредственно данные, разбитые на блоки, хранятся на executor'ах или во внешнем хранилище (если датасет не был еще загружен для обработки, т.е. так могут быть представлены, условно говоря, входные данные приложения Spark).

- Отказоустойчивая – для RDD ведется Lineage – Spark всегда знает как восстановить RDD в случае сбоя
- Внутри RDD разбита на партиции – это минимальный объем RDD, который будет обработан каждым рабочим узлом.
- RDD распределена по узлам Executors



# Основные концепции Spark

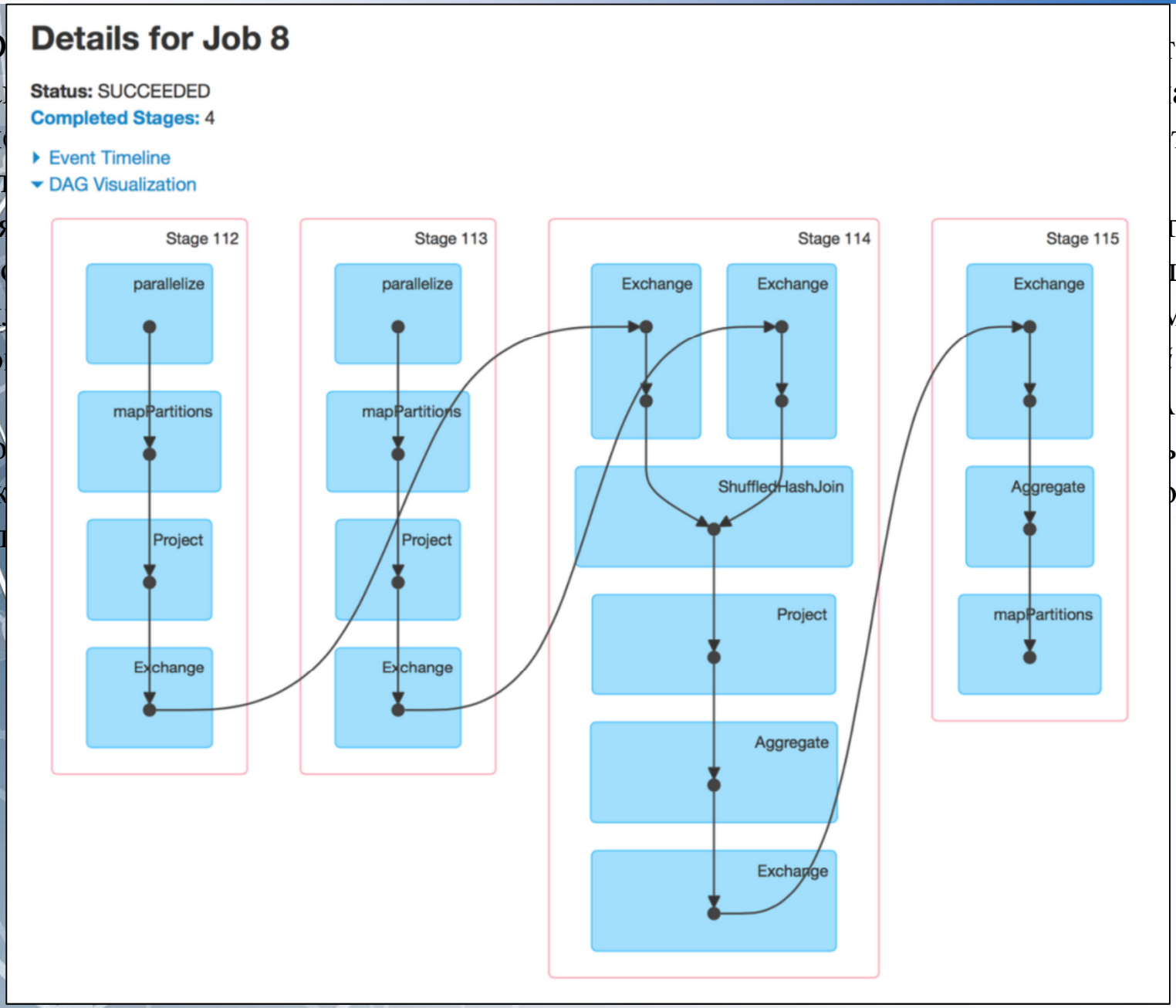


# Основные концепции Spark

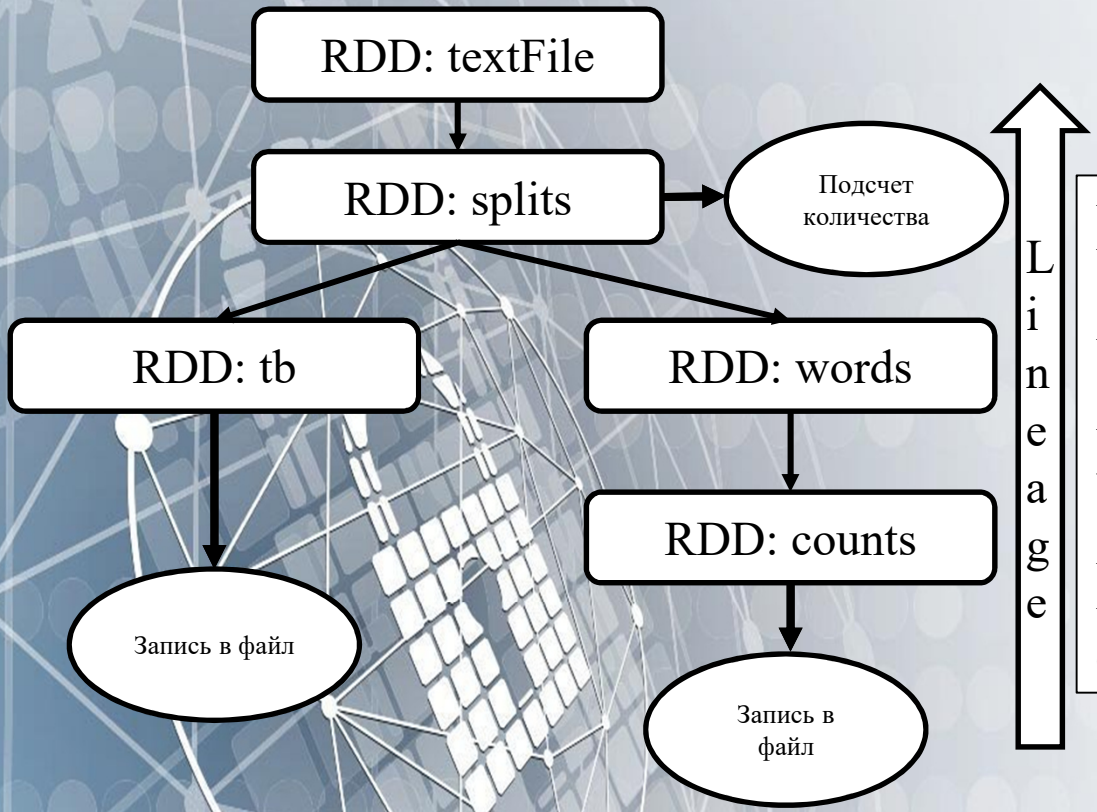
RD
ссыла
За сч
случа
Свя
обраб
храни
много
(граф
счет о
раз ск
при ит

тельно
астер.
торые

т план
шнего
можно
план
G) за
один
олезно



# Lazy Evaluation




Lineage

```
val textFile = sc.textFile("hdfs://...")
val splits = textFile.flatMap(line => line.split(" "))

val allwords = splits.count()

val tb = splits.filter(_.startsWith("b")))
tb.saveAsTextFile("hdfs://...")

val words = splits.map(word => (word, 1))
val counts = words.reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```

**Трансформация**
не приводит к запуску вычислений

**Примеры: map**(*func*), **filter**(*func*), **union**(*otherDataset*), **reduceByKey**(*func*), **join**(*otherDataset*)

**Действие**
запускает цепочку вычислений

**Примеры: collect**(), **count**(), **take**(*n*), **saveAsTextFile**(*path*)

# Преимущества и недостатки Lazy Evaluation

**Преимущества:**

**Удобство написания программ -** улучшает читаемость кода, можно разбивать на небольшие куски, потом все соберется в единый DAG.

**Избежание ненужных вычислений и трафика между драйвером и узлами –** Обрабатываются только те данные, которые реально нужны. Строится единый план выполнения.

**Оптимизация -** построенный план запроса оптимизируется Spark, сдвигая например некоторые фильтры ближе к началу

**Недостатки:**

**Необходимо заботиться о повторном вычислении -** Каждое действие выполняется без оглядки на другое. Необходимо избегать повторных вычислений. cache(), persist().

**Разрастание плана запросов -** особенно в итерационных алгоритмах. Здесь может помочь savepoint(), который сохраняет данные на диск и очищает lineage.

# Преимущества Apache Spark

**Скорость**

- Apache Spark – это вычислительный инструмент, работающий со скоростью света. Благодаря уменьшению количества операций чтения-записи на диск и хранения промежуточных данных в памяти, Spark запускает приложения в 100 раз быстрее в памяти и в 10 раз быстрее на диске, чем Hadoop.
- Hadoop MapReduce – MapReduce читает и записывает на диск, а это снижает скорость обработки и эффективность в целом.

**Простота использования**

- Apache Spark – многие библиотеки Spark облегчают выполнение большого количества основных высокоуровневых операций при помощи RDD (Resilient Distributed Dataset/эластичный распределённый набор данных).
- Hadoop—в MapReduce разработчикам нужно написать вручную каждую операцию, что только усложняет процесс при масштабировании сложных проектов.

# Преимущества Apache Spark

**Обработка больших наборов данных**

- Apache Spark – так как, Spark оптимизирован относительно скорости и вычислительной эффективности при помощи хранения основного объёма данных в памяти, а не на диске, он может показывать более низкую производительность относительно Hadoop MapReduce в случаях, когда размеры данных становятся такими огромными, что недостаточность RAM становится проблемой.
- Hadoop – Hadoop MapReduce позволяет обрабатывать огромные наборы данных параллельно. Он разбивает большую цепочку на небольшие отрезки, чтобы обрабатывать каждый отдельно на разных узлах данных. Если итоговому датасету необходимо больше, чем имеется в доступе RAM, Hadoop MapReduce может сработать лучше, чем Spark. Поэтому Hadoop стоит выбрать в том случае, когда скорость обработки не критична и решению задач можно отвести ночное время, чтобы утром результаты были готовы. .

# Преимущества Apache Spark

**Функциональность**

- Итеративная обработка. Если по условию задачи нужно обрабатывать данные снова и снова, Spark разгромит Hadoop MapReduce. Spark RDD активирует многие операции в памяти, в то время как Hadoop MapReduce должен записать промежуточные результаты на диск.
- Обработка в почти что реальном времени. Если бизнесу нужны немедленные инсайты, тогда стоит использовать Spark и его обработку прямо в памяти.
- Обработка графов. Вычислительная модель Spark хороша для итеративных вычислений, которые часто нужны при обработке графов. И в Apache Spark есть GraphX – API для расчёта графов.
- Машинное обучение. В Spark есть MLlib – встроенная библиотека машинного обучения, а вот Hadoop нужна третья сторона для такого же функционала. MLlib имеет алгоритмы “out-of-the-box” (возможность подключения устройства сразу после того, как его достали из коробки, без необходимости устанавливать дополнительное ПО, драйверы и т.д.), которые также реализуются в памяти.
- Объединение датасетов. Благодаря скорости Spark может создавать все комбинации быстрее, а вот Hadoop показывает себя лучше в объединении очень больших наборов данных, которым нужно много перемешивания и сортировки.

# Заключение

Массовое распространение больших данных и экспоненциально растущая скорость вычислительных мощностей привели к тому, что инструменты вроде Apache Spark и других программ, анализирующих большие данные, стали незаменимыми в работе исследователей данных, став стандартом в индустрии реализации аналитики больших данных и решении сложных бизнес-задач в реальном времени.

СПАСИБО ЗА ВНИМАНИЕ!