

# Промышленное машинное обучение на Spark



# Содержание курса

- 01      Оборачиваем модель в сервис. Первая часть: Docker. Flask
- 02      Оборачиваем модель в сервис. Вторая часть: Requests. REST API
- 03**     Распределенные вычисления. HDFS. MapReduce. Spark DataFrame
- 04      Погружение в среду Spark. RDD, SQL, Pandas API.
- 05      Генерация признаков. Spark feature engineering
- 06      Распределенное обучение моделей. Spark ML
- 07      Обработка и хранение текстовых данных и картинок.  
Spark image processing. Spark NLP.
- 08      Обработка потоковых данных. Spark Streaming.

# 3. Распределенные вычисления. HDFS. MapReduce. Spark DataFrame

План:

## I. Что такое большие данные и откуда они берутся?

- Сфера производящие большие данные. Data explosion.
- Большие данные - где начало. 3 основных принципа.
- Как компании справляются с большими данными IaaS/PaaS/SaaS.

## II. Как хранить большие данные?

- Устройство файловой системы Linux.
- Отказоустойчивость.
- HDFS. Ее устройство и основные свойства.

## III. Как обрабатывать большие данные?

- Сортировка во внешней памяти. Плюсы и минусы распределенных систем.
- Предпосылки к созданию MapReduce.
- Задача подсчета слов. Map. Shuffle. Reduce.

## IV. Spark

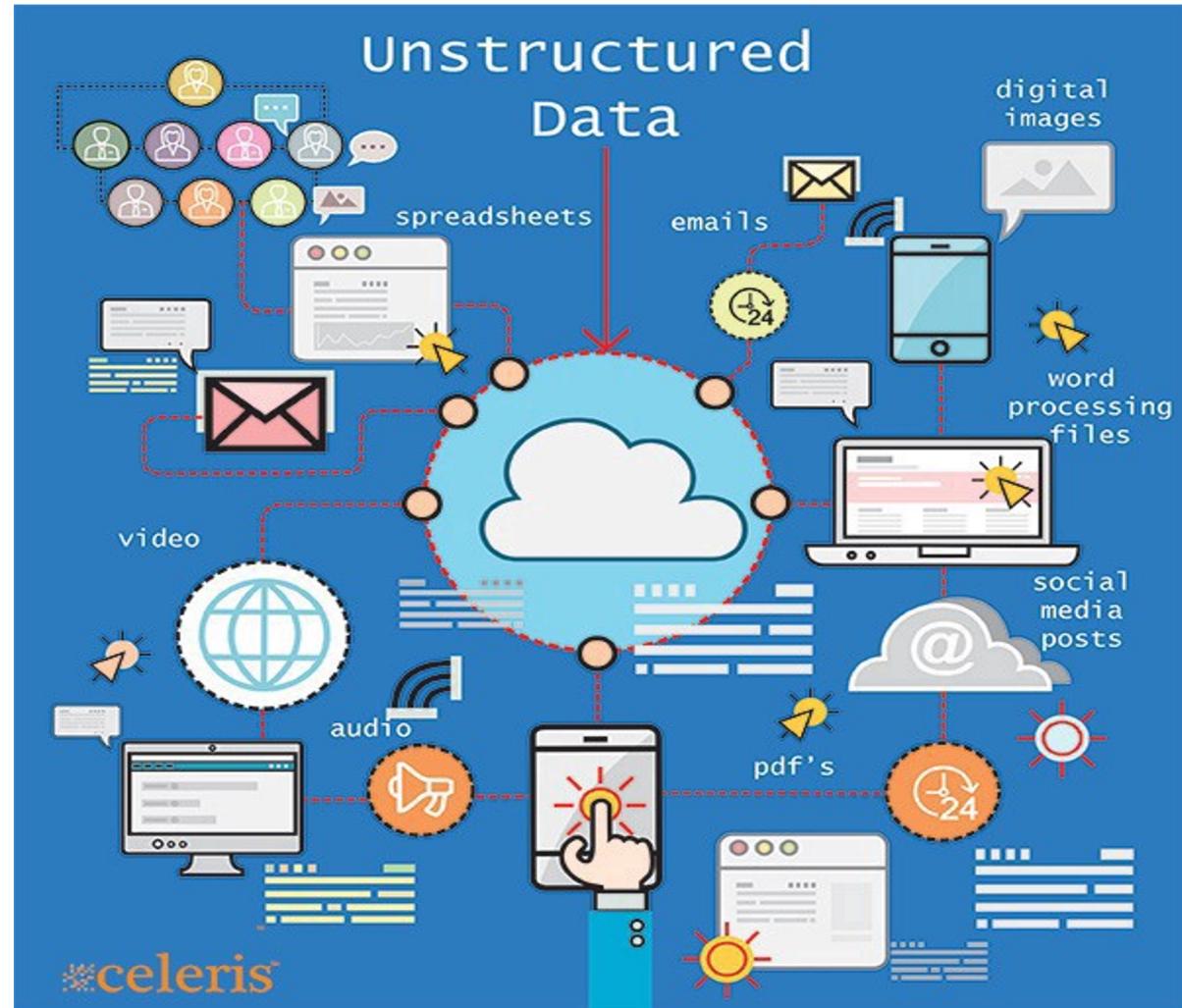
- Spark — основные абстракции и объекты.
- Действия и преобразования.
- Data locality. СерIALIZАЦИЯ

Сфера производящие большие данные.  
Data explosion.

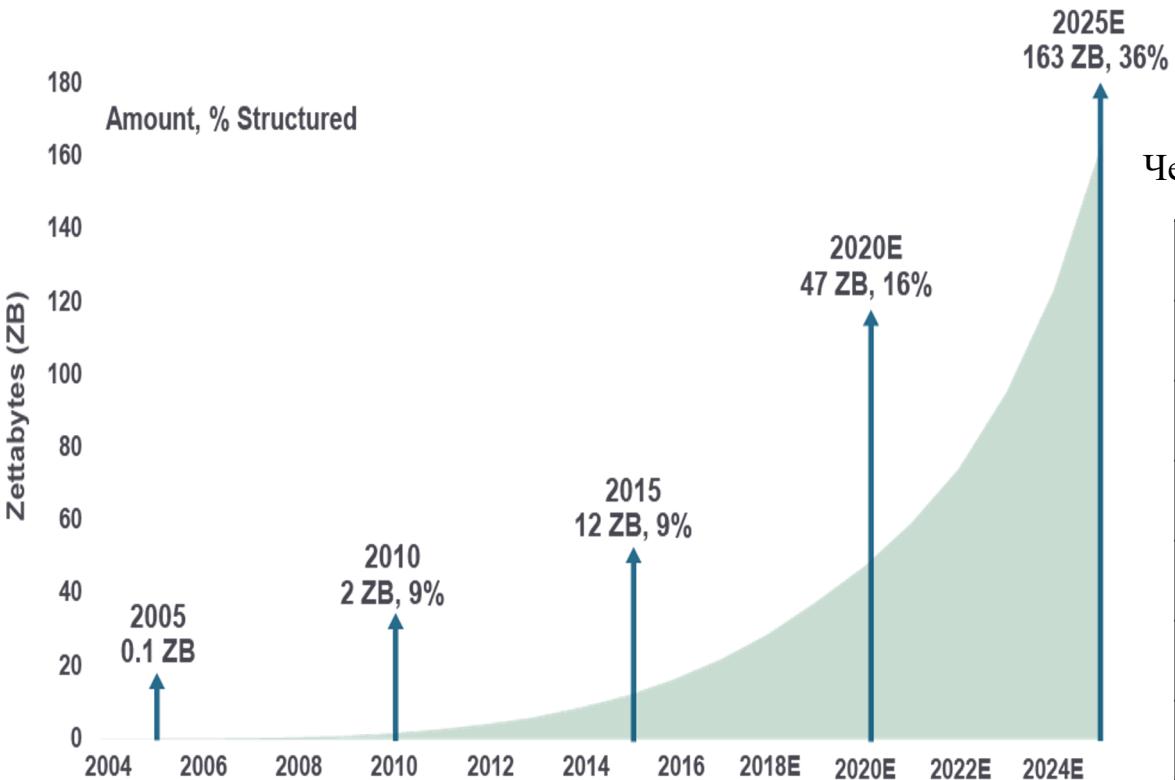
## Откуда пришла Big Data

Сфера:

- Телеком
- Банки
- Социальные сети
- Медиа
- Промышленность
- Биоинформатика
- Интернет вещей



# Data explosion.

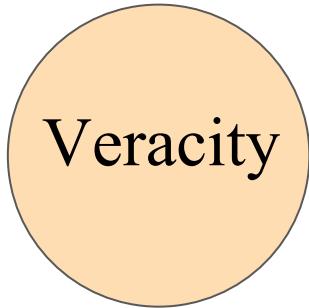


Чему равен зеттабайт? - триллиону гигабайт

1 kilobyte	1 000
1 megabyte	1 000 000
1 gigabyte	1 000 000 000
1 terabyte	1 000 000 000 000
1 petabyte	1 000 000 000 000 000
1 exabyte	1 000 000 000 000 000 000
1 zettabyte	1 000 000 000 000 000 000 000

Большие данные, где начало. 34 основных принципа.

# Большие данные, где начало. 3 основных принципа.



Как компании справляются с большими данными?

### **Infrastructure as a Service**

Applications

Data

Runtime

Middleware

O/S

Virtualization

Servers

Storage

Networking

### **Platform as a Service**

Applications

Data

Runtime

Middleware

O/S

Virtualization

Servers

Storage

Networking

### **Software as a Service**

Applications

Data

Runtime

Middleware

O/S

Virtualization

Servers

Storage

Networking

You Manage

Other Manages

- **IaaS — это Infrastructure as a Service.** Инфраструктура как услуга. К инфраструктуре относят вычислительные ресурсы: виртуальные серверы, хранилища, сети. ([Google Compute Engine](#), [DigitalOcean](#), [Amazon Web Services \(AWS\)](#), and [Cisco Metacloud](#)).
  - Перенос IT-систем в облако.
  - Экономия на инфраструктуре.
  - Быстрый запуск бизнеса.
  - Расширение инфраструктуры.
  - Инфраструктура для компаний со скачками спроса.
  - Разработка и тестирование.
- **PaaS - это Platform as a Service**, платформа как услуга. ([Windows Azure](#), [OpenShift](#), [Heroku](#), and [Google App Engine](#)).
  - Базы данных.
  - Разработка приложений в контейнерах.
  - Аналитика больших данных.
  - Машинное обучение.
- **SaaS — это Software as a Service**, программное обеспечение как сервис ([Google App Engine](#), [Dropbox](#), [JIRA](#), and [others](#)).
  - электронная почта
  - CRM-системы
  - планировщики задач
  - веб-конструкторы для создания сайтов

Первый ключевой вопрос: как хранить большие данные?

# Файловая система

Основными функциями файловой системы являются:

- размещение и упорядочивание на носителе данных в виде файлов;
- определение максимально поддерживаемого объема данных на носителе информации;
- создание, чтение и удаление файлов;
- назначение и изменение атрибутов файлов (размер, время создания и изменения, владелец и создатель файла, доступен только для чтения, скрытый файл, временный файл, архивный, исполняемый, максимальная длина имени файла и т. п.);
- определение структуры файла;
- поиск файлов;
- организация каталогов для логической организации файлов;
- защита файлов при системном сбое;
- защита файлов от несанкционированного доступа и изменения их содержимого

# Linux File System Directories

/bin: Where Linux core commands reside like ls, mv.

/boot: Where boot loader and boot files are located.

/dev: Where all physical drives are mounted like USBs DVDs.

/etc: Contains configurations for the installed packages.

/home: Where every user will have a personal folder to put his folders with his name like /home/likegeeks.

/lib: Where the libraries of the installed packages located since libraries shared among all packages, unlike Windows, you may find duplicates in different folders.

/media: Here are the external devices like DVDs and USB sticks that are mounted, and you can access their files from here.

/mnt: Where you mount other things Network locations and some distros, you may find your mounted USB or DVD.

/opt: Some optional packages are located here and managed by the package manager.

/proc: Because everything on Linux is a file, this folder for processes running on the system, and you can access them and see much info about the current processes.

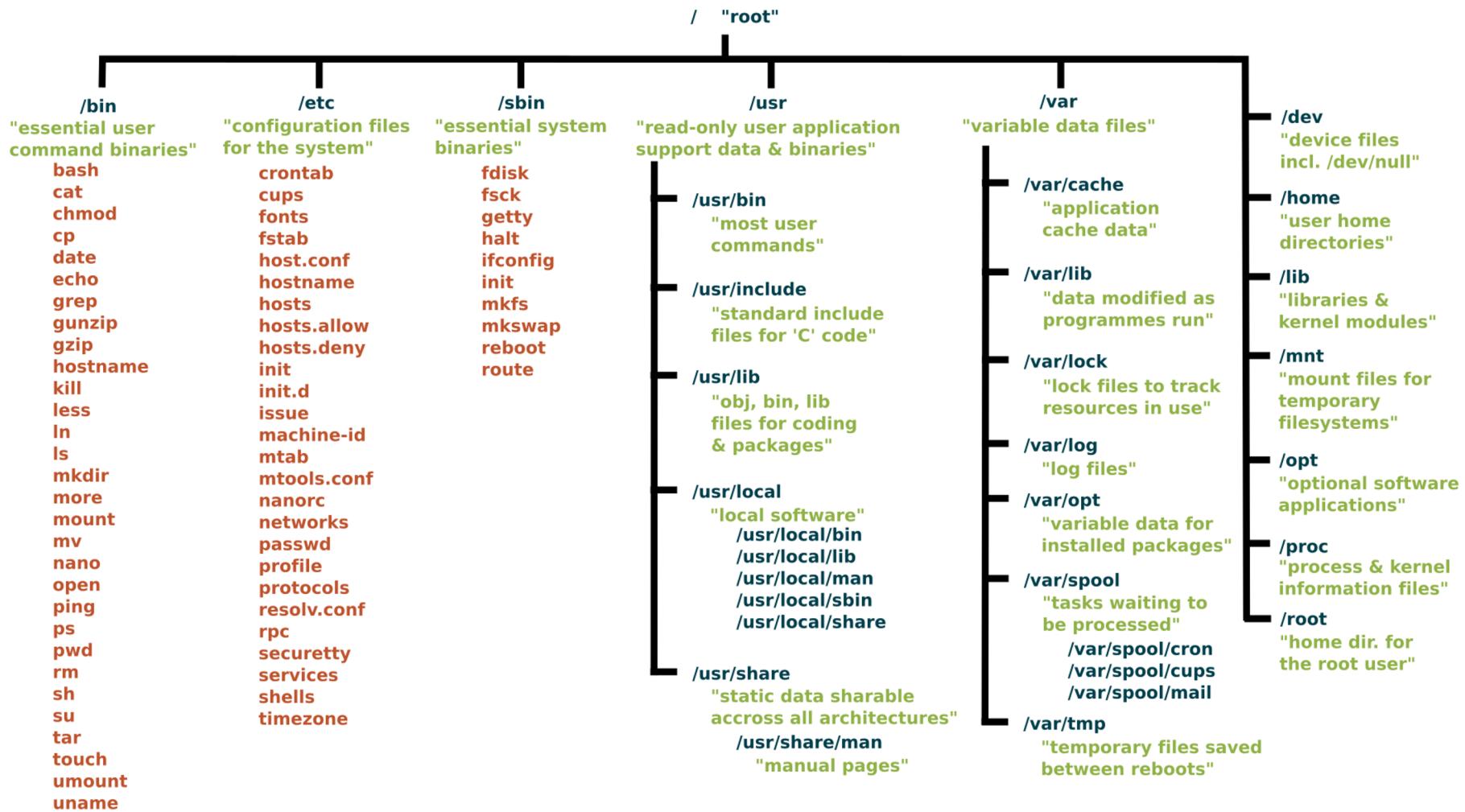
/root: The home folder for the root user.

/sbin: Like /bin, but binaries here are for root user only.

/tmp: Contains the temporary files.

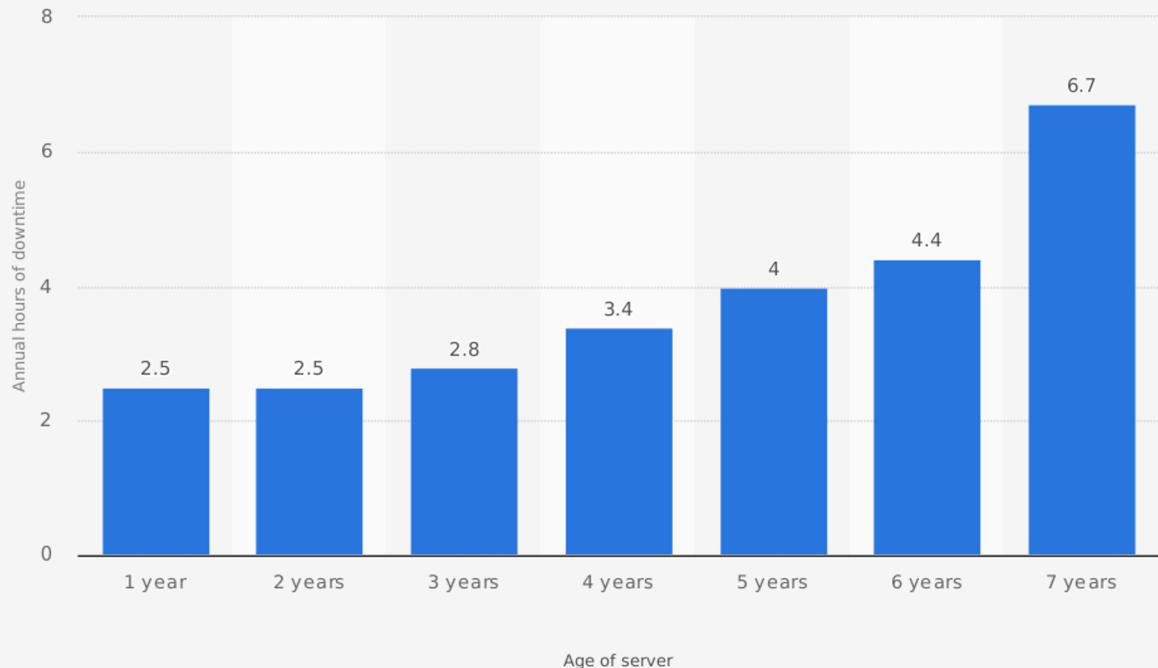
/usr: Where the utilities and files shared between **users on Linux**.

/var: Contains system logs and other variable data.



Отказоустойчивость хранения данных и вычислений.

## Annual number of server downtime hours based on server age, as of 2015



Source  
IDC  
© Statista 2018

Additional Information:  
Worldwide; 2015

Вероятность, что в следующий час случится поломка:

$$\begin{aligned}P &= 2.5 / (24 * 365) \\&= 0.00028\end{aligned}$$

$$\begin{aligned}P(\text{не выйдет из строя}) \\&= (1 - P) = 0.9997\end{aligned}$$

1000 машин в кластере

Вероятность, что один из серверов сломается ближайший час:

$$1 - 0.9997^{1000} = 0.25$$

HDFS. Ее устройство и основные свойства.

# Полезные определения

Кластер - совокупность компьютеров объединенных сетью и выполняющих задачи, посылаемые клиентами.

Нода (Node) - один из компьютеров подключенных к кластеру.

Стойка (Rack) - совокупность нескольких нод, объединенных сетью

Демон - компьютерная программа в системах класса UNIX, запускаемая самой системой и работающая в фоновом режиме без прямого взаимодействия с пользователем

Клиент - это аппаратный или программный компонент вычислительной системы, посылающий запросы серверу.

# Цели:

- Отказ оборудования - это скорее норма, чем исключение.
- Приложения, работающие в HDFS, имеют большие наборы данных. Типичный файл в HDFS имеет размер от гигабайтов до терабайт.
- Write once Read many
- «Перемещение вычислений дешевле, чем перемещение данных» (Data locality)
- HDFS был разработан таким образом, чтобы его можно было легко переносить с одной платформы на другую.

# Основные задачи HDFS

1. Отказоустойчивость в условиях частых сбоев и поломок
2. Параллельная обработка частей данных

## Block Replication

- Управляющий узел, узел имен или сервер имен (NameNode)

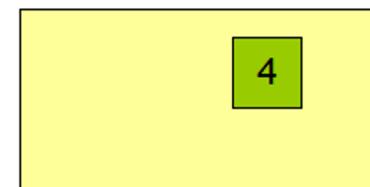
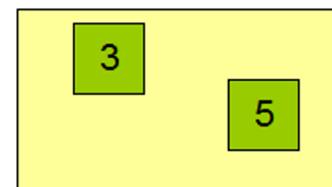
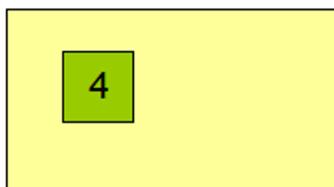
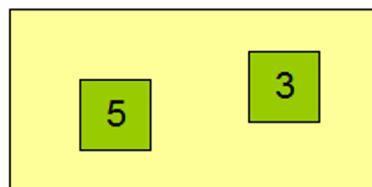
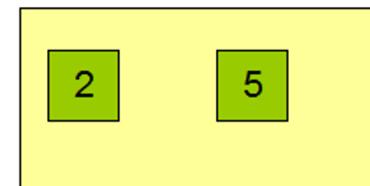
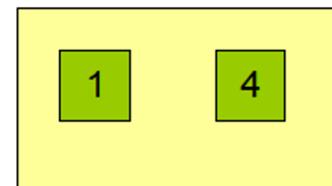
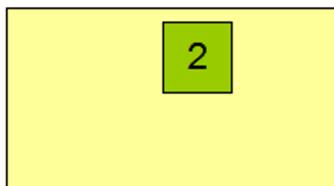
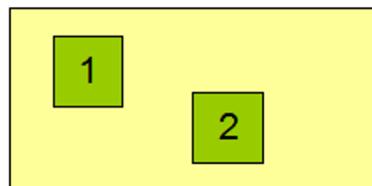
Namenode (Filename, numReplicas, block-ids, ...)

/users/sameerp/data/part-0, r:2, {1,3}, ...

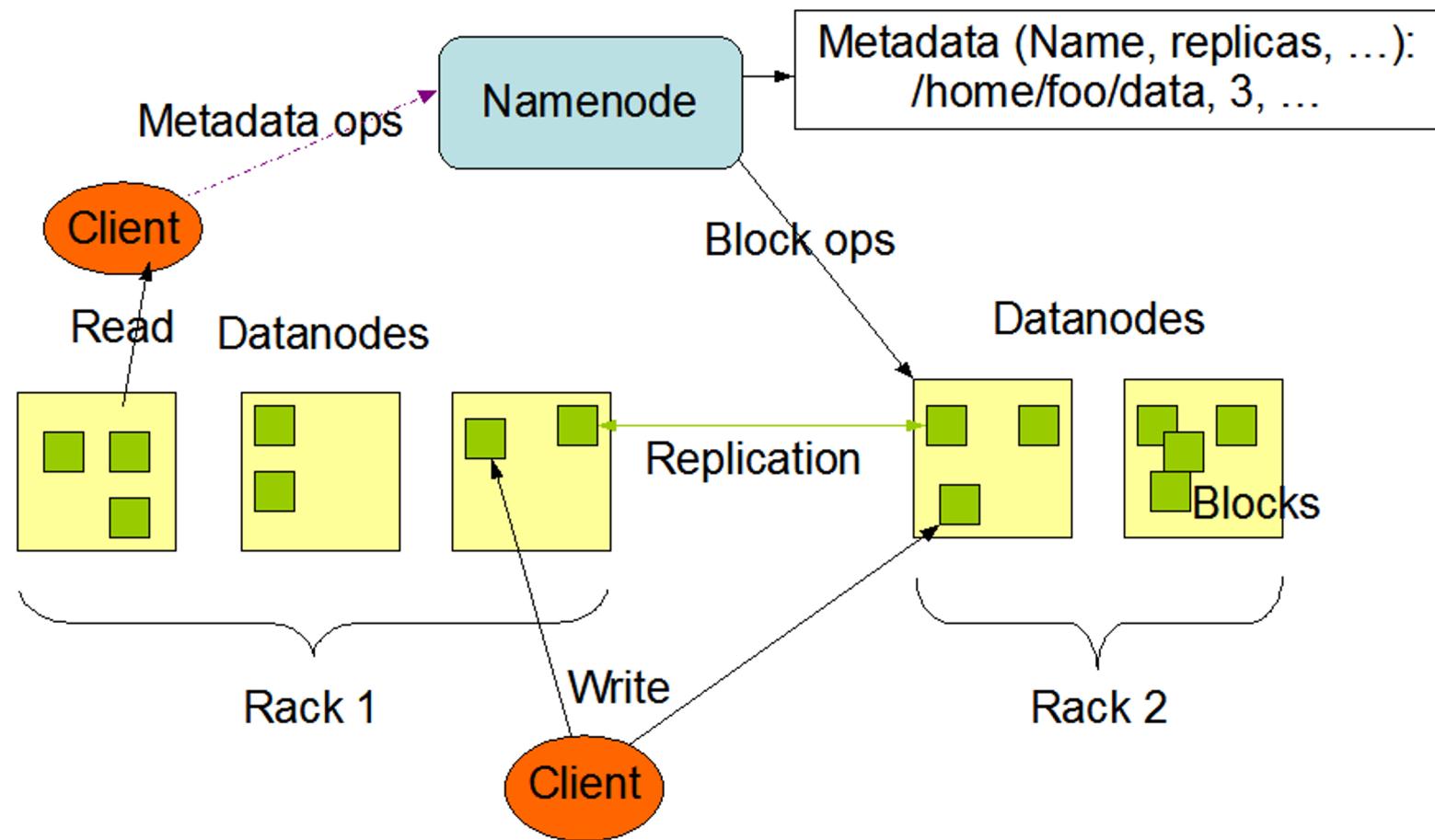
/users/sameerp/data/part-1, r:3, {2,4,5}, ...

## Datanodes

- Узел или сервер данных (DataNode, Node)



# HDFS Architecture



# Особенности:

- большой размер блока
- ориентация на недорогие и, поэтому не самые надежные сервера
- репликация на уровне кластера
- репликация происходит в асинхронном режиме
- клиенты могут считывать и писать файлы HDFS напрямую через программный интерфейс Java;
- файлы пишутся однократно,
- принцип WORM (Write-once and read-many)
- сжатие данных и рациональное использование дискового пространства
- самодиагностика
- все метаданные сервера имен хранятся в оперативной памяти.

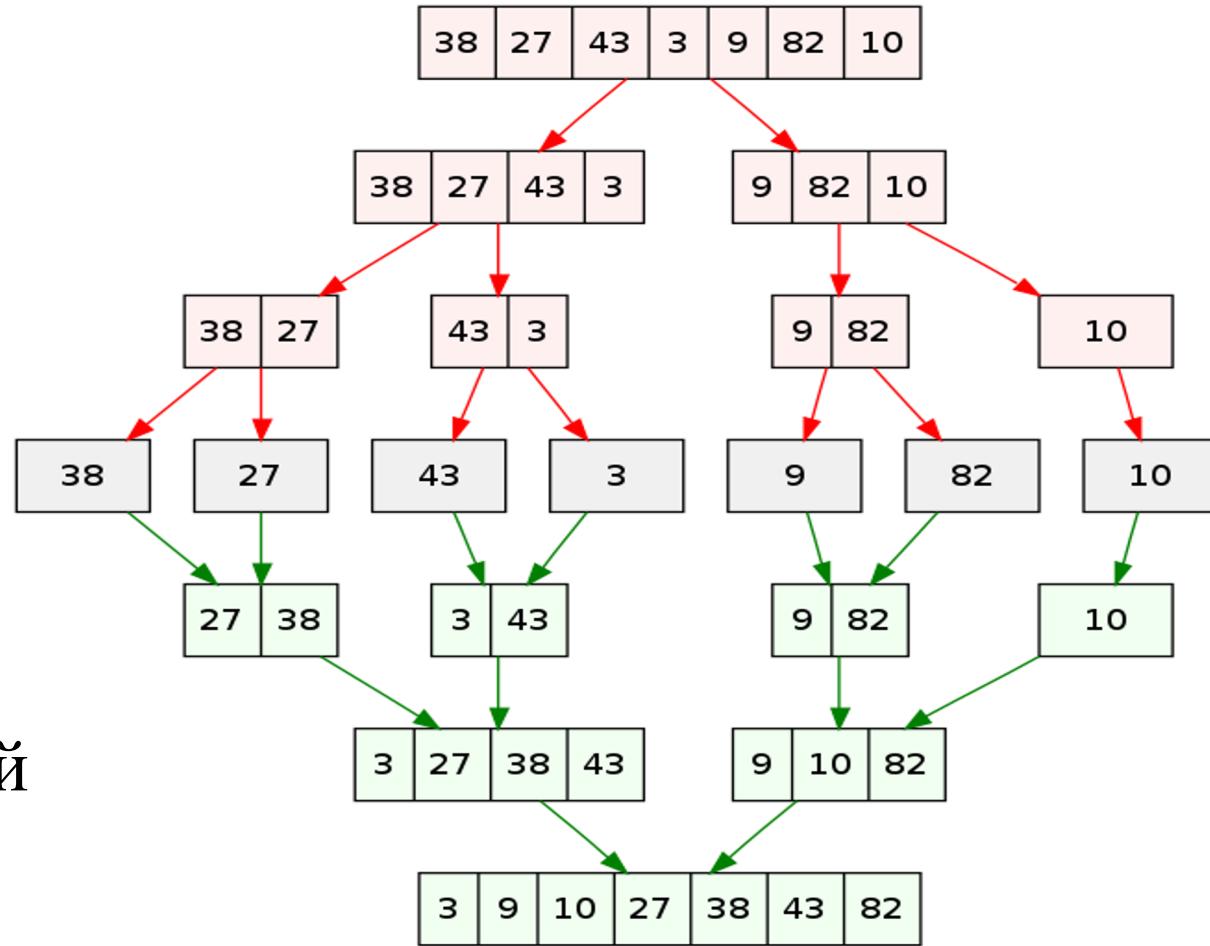
Второй ключевой вопрос: как обрабатывать большие данные?

Задача:

Отсортировать массив:

38	27	43	3	9	82	10
----	----	----	---	---	----	----

Разделяй и  
властвуй



Классический  
merge sort

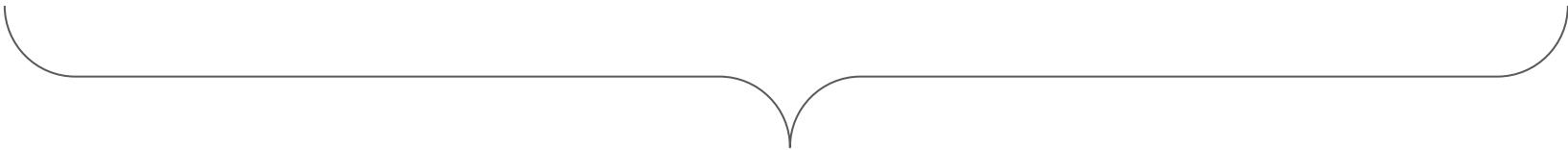
## Задача:

Отсортировать массив, который не помещается в оперативную память:

38	27	43	3	9	82	10
----	----	----	---	---	----	----

...

38	27	43	3	9	82	10
----	----	----	---	---	----	----



1 Тб

38	27	43	3	9	82	10
----	----	----	---	---	----	----

. . .

38	27	43	3	9	82	10
----	----	----	---	---	----	----

Сортировка

3	9	10	27	38	43	82
---	---	----	----	----	----	----

. . .

3	9	10	27	38	43	82
---	---	----	----	----	----	----

Слияние

3	3	3	3	3	3	3
---	---	---	---	---	---	---

. . .

82	82	82	82	82	82	82
----	----	----	----	----	----	----

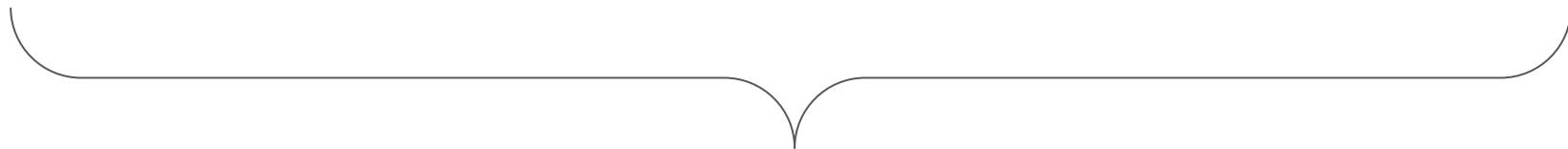
## Задача:

Отсортировать массив, который не помещается на доступный жесткий диск:

38	27	43	3	9	82	10
----	----	----	---	---	----	----

...

38	27	43	3	9	82	10
----	----	----	---	---	----	----

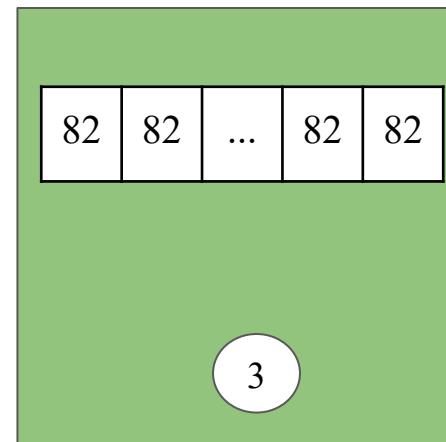
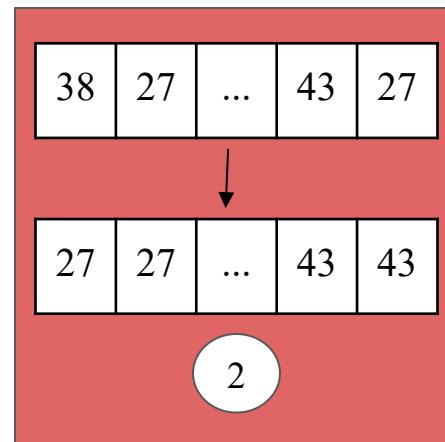
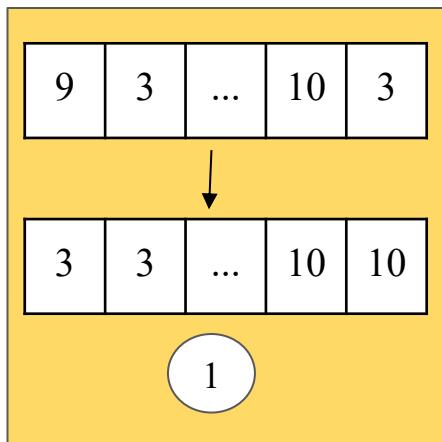


500 Тб

38	27	43	3	9	82	10
----	----	----	---	---	----	----

...

38	27	43	3	9	82	10
----	----	----	---	---	----	----



Задача подсчета слов. Map. Shuffle. Reduce.

# Задача подсчета слов

Кошка Мышь Собака

Собака Собака Кошка

Собака Кошка Утка

Кошка Мышь  
Собака

Собака Собака  
Кошка

Собака Кошка  
Утка

1

2

3

# Мар:

Кошка  
Мышь  
Собака

Кошка, 1  
Мышь, 1  
Собака, 1

1

Собака  
Собака  
Кошка

Собака, 1  
Собака, 1  
Кошка, 1

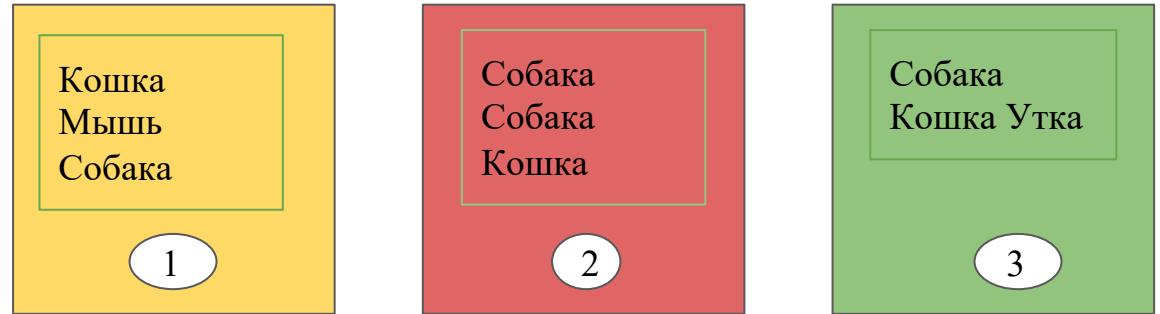
2

Собака  
Кошка  
Утка

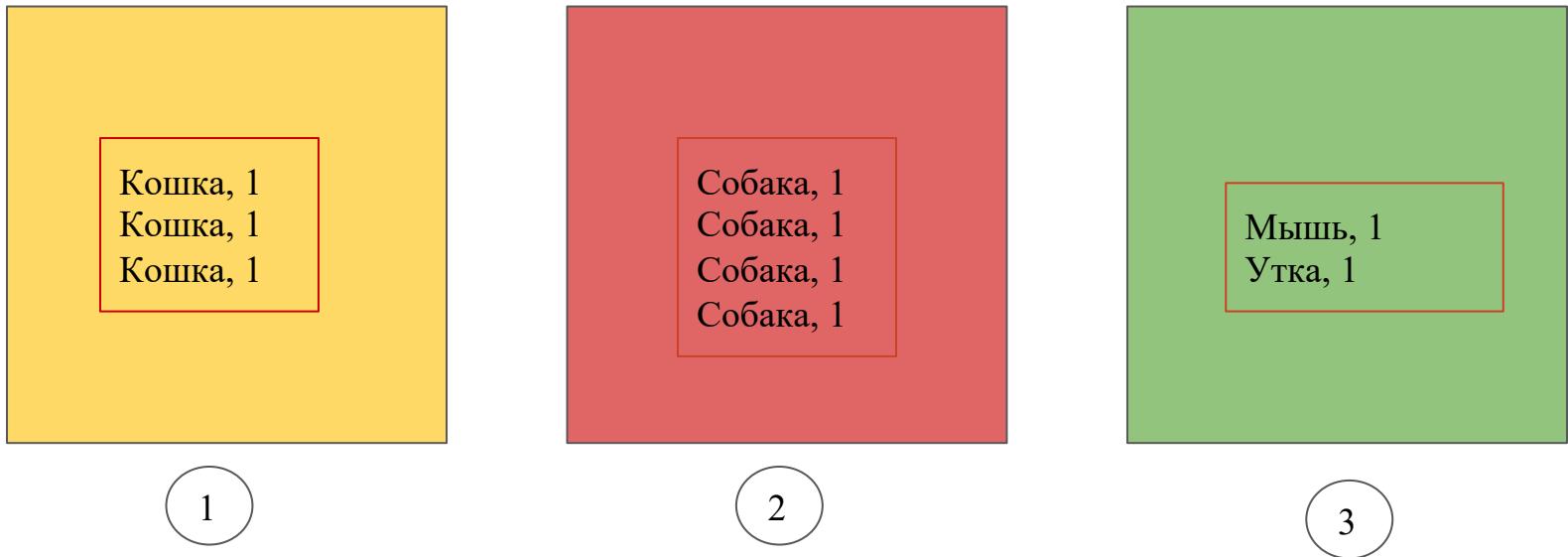
Собака, 1  
Кошка, 1  
Утка, 1

3

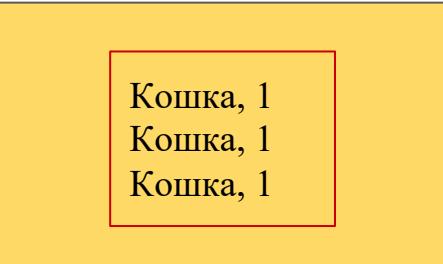
# Shuffle:



По сути - сортировка:



Reduce:



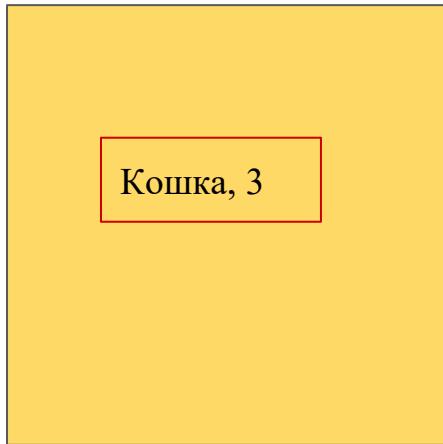
1



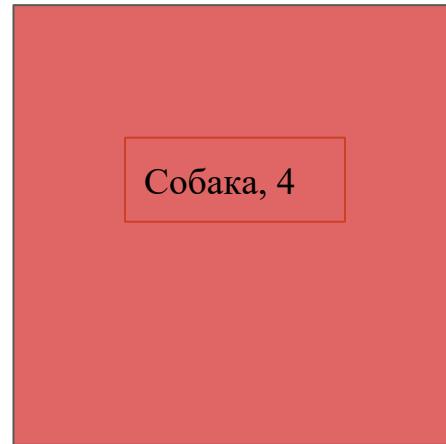
2



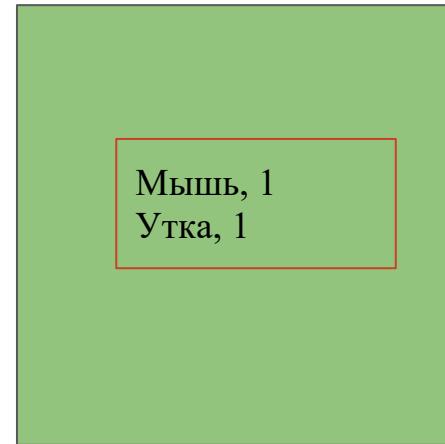
3



1



2



3

Spark - основные абстракции и объекты.

# Основные концепции Spark

- RDD
- SparkContext (sc)
- Directed acyclic graphs (DAG)
- Actions and transformations
- Spark DataFrames

Resilient distributed datasets (RDD).

**RDD (Resilient Distributed Dataset)** — это фундаментальная структура данных Apache Spark, которая представляет собой неизменяемую коллекцию объектов, которые вычисляются на разных узлах кластера. Каждый набор данных в Spark RDD логически разделен на множество серверов, чтобы их можно было вычислить на разных узлах кластера.

# SparkContext (sc)

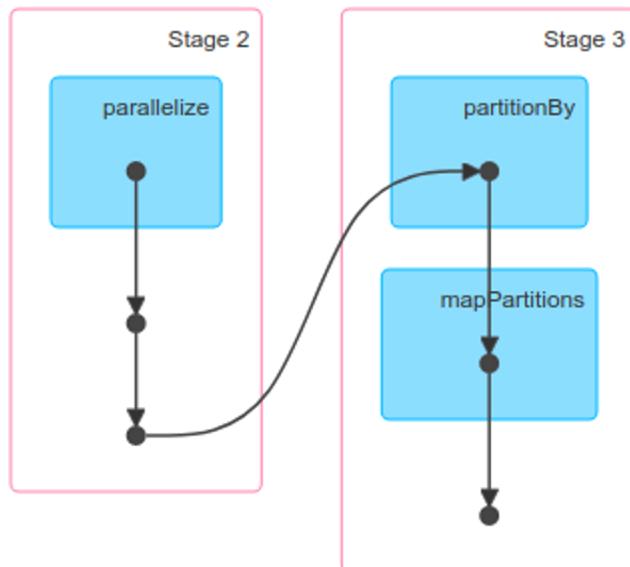
**SparkContext** — это точка входа для всех операций Spark и средство, с помощью которого приложение подключается к ресурсам кластера Spark. Он инициализирует экземпляр Spark и впоследствии может использоваться для выполнения действий, преобразований над RDD, а также извлечения данных и других функций Spark.

SparkContext также инициализирует различные свойства процесса, такие как имя приложения, количество ядер, параметры использования памяти и другие характеристики. В совокупности эти свойства содержатся в объекте SparkConf, который передается в SparkContext в качестве параметра.

Directed Acyclic Graph (DAG).

Пример с прошлой лекции подсчет слов

```
wordCountsCollected = (sparkdata
    .map(mapword)
    .reduceByKey(lambda a,b: a+b)
    .collect())
```



## 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. При вызове действия созданный DAG отправляется планировщику DAG, который далее разбивает граф на этапы и задачи.

Spark DataFrame.

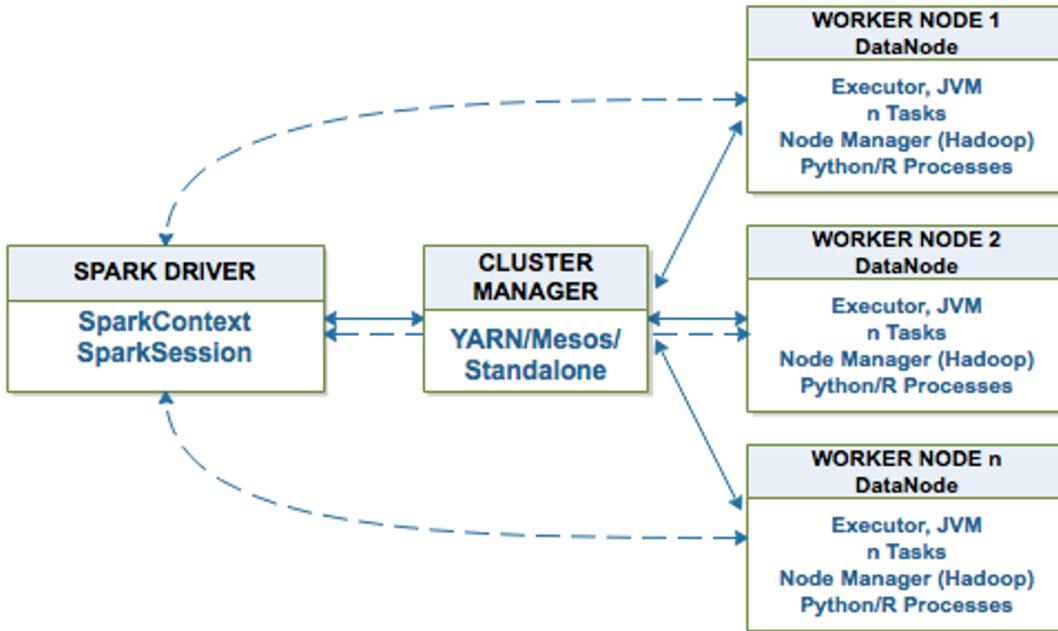
**Spark DataFrame** — это набор данных, организованный в именованные столбцы. Концептуально он эквивалентен таблице в реляционной базе данных или фрейму данных в R / Python, но с более обширной внутренней оптимизацией. DataFrames могут быть созданы из широкого спектра источников, таких как файлы структурированных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

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.



**WorkerNode** — это серверы, на которых размещены приложения Spark. Каждое приложение получает свой собственный уникальный процесс-исполнитель, а именно процессы, которые выполняют фактическое действие и задачи преобразования.

**SparkDriver** отправляет инструкции рабочим узлам для планирования задач.

**Менеджеры кластеров** координируют обмен данными между рабочими узлами, управляемыми узлами (такими как запуск, остановка)

Data locality. Как ее реализовать. Сериализация.  
Пример.

**Локальность данных** — это процесс перемещения вычисления ближе к месту, где находятся фактические данные на узле, вместо перемещения больших данных в вычисления. Это минимизирует перегрузку сети и увеличивает общую пропускную способность системы.

- 1) Данные лежат на том же узле, что и mapper — good
- 2) Данные лежат на соседнем узле в стойке — not so good
- 3) Данные лежат в ноде на другой стойке — bad

## Локальность данных обеспечивается сериализацией кода

**Сериализация** — процесс перевода какой-либо структуры данных в последовательность байтов. Обратной к операции сериализации является операция десериализации (структуризации) — восстановление начального состояния структуры данных из битовой последовательности.

Но не все функции сериализуются - например

```
def func(x):
    f = open('file.txt')
    return x in f.readlines()
```