

# Анализ больших данных с Apache Spark

## Лекция 2. Распределенные вычисления в памяти

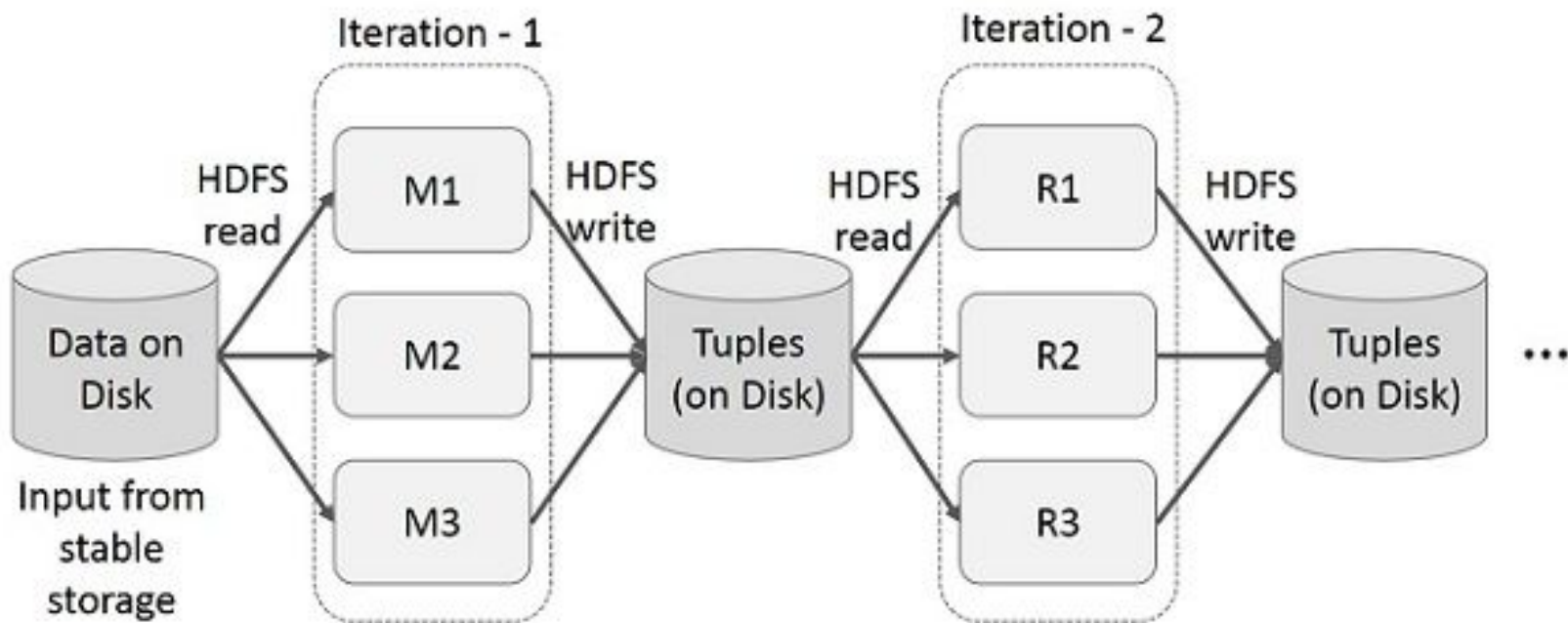
Мурашкин Вячеслав  
2017

<https://github.com/a4tunado/lectures-hse-spark/tree/master/002>

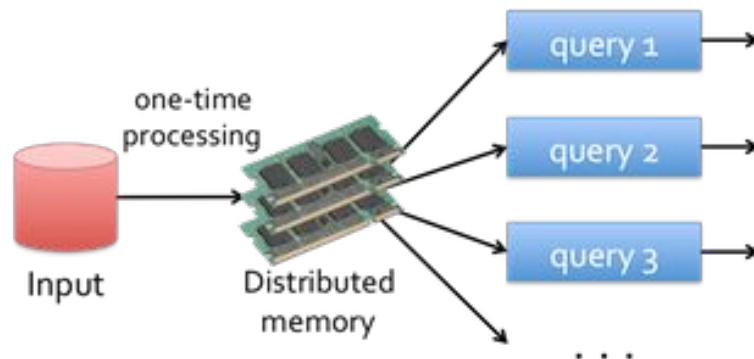
# Лекция 2. Распределенные вычисления в памяти

- Недостатки MapReduce
- Распределенные вычисления в памяти
- RDD (Resilient Distributed Datasets)
- Архитектура Spark
- Концепция вычислений на Spark
- DataFrame и SQL запросы
- Задание: реализовать обучение модели логистической регрессии на Spark

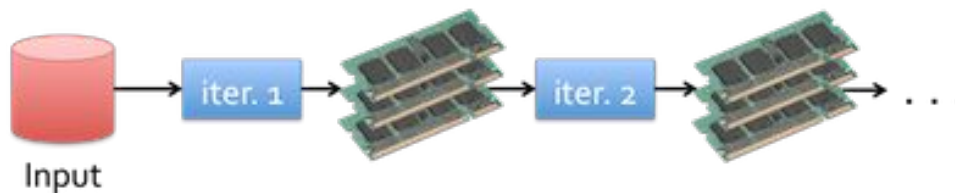
# Недостатки MapReduce



# Распределенные вычисления в памяти



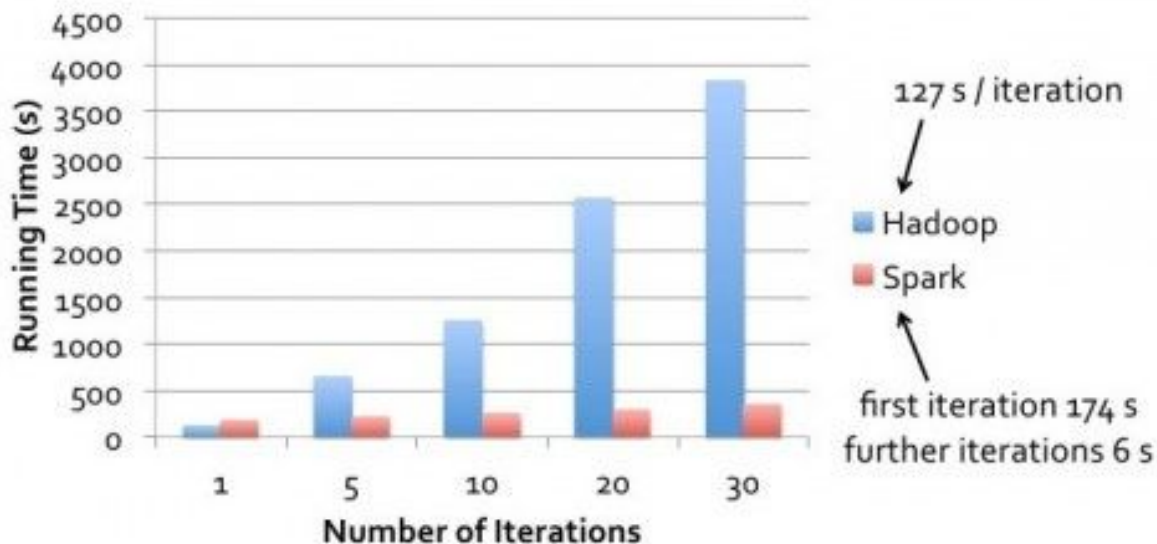
(a) Low-latency computations (queries)



(b) Iterative computations

# Распределенные вычисления в памяти

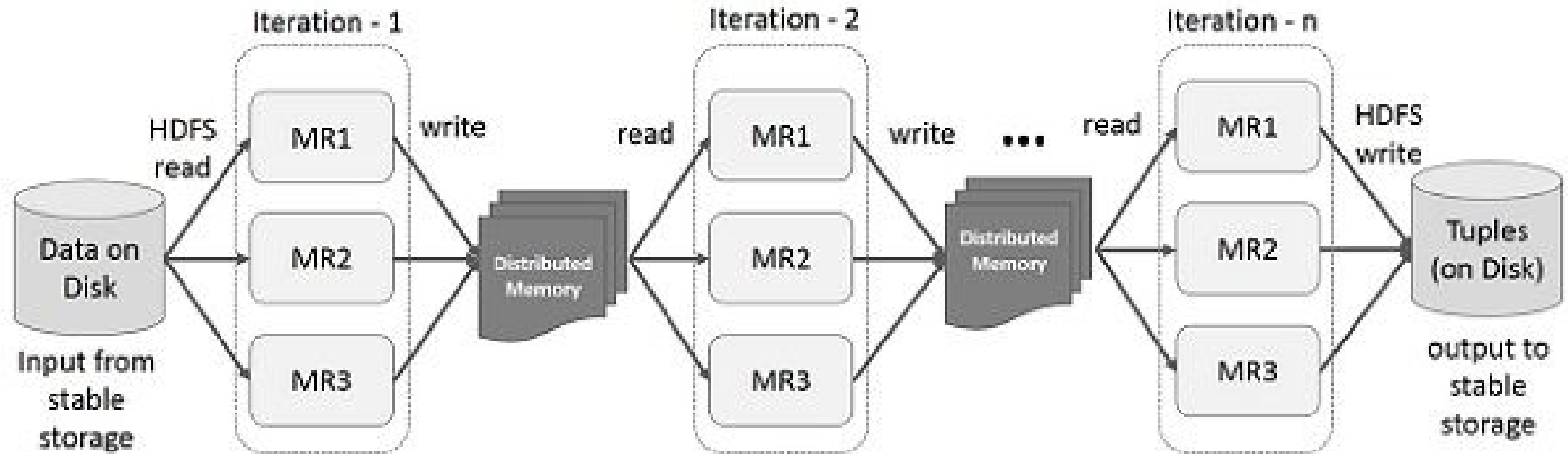
## Logistic Regression Performance



# RDD (Resilient Distributed Datasets)

- Основной способ представления данных в Spark
- Read-only
- Представляет собой набор объектов одного типа
- Каждый набор данных в RDD может быть разбит на логические части (chunk) и размещен на разных машинах кластера
- Может храниться как на жестком диске, так и в оперативной памяти
- Новый RDD можно получить загрузив данные из распределенного хранилища (HDFS), либо путем преобразования другого RDD
- [https://cs.stanford.edu/~matei/papers/2012/nsdi\\_spark.pdf](https://cs.stanford.edu/~matei/papers/2012/nsdi_spark.pdf)

# RDD (Resilient Distributed Datasets)



# RDD (Resilient Distributed Datasets)

- Управление настройками RDD
  - [partitioning](#) (repartition)
    - способ разбиения данных на логические части (chunk)
  - [persist](#) (MEMORY\_ONLY, MEMORY\_AND\_DISK, ...)
    - определяет место хранения данных, полезно если все данные не помещаются в память
  - [unpersist](#)
    - удаляет данные и высвобождает память



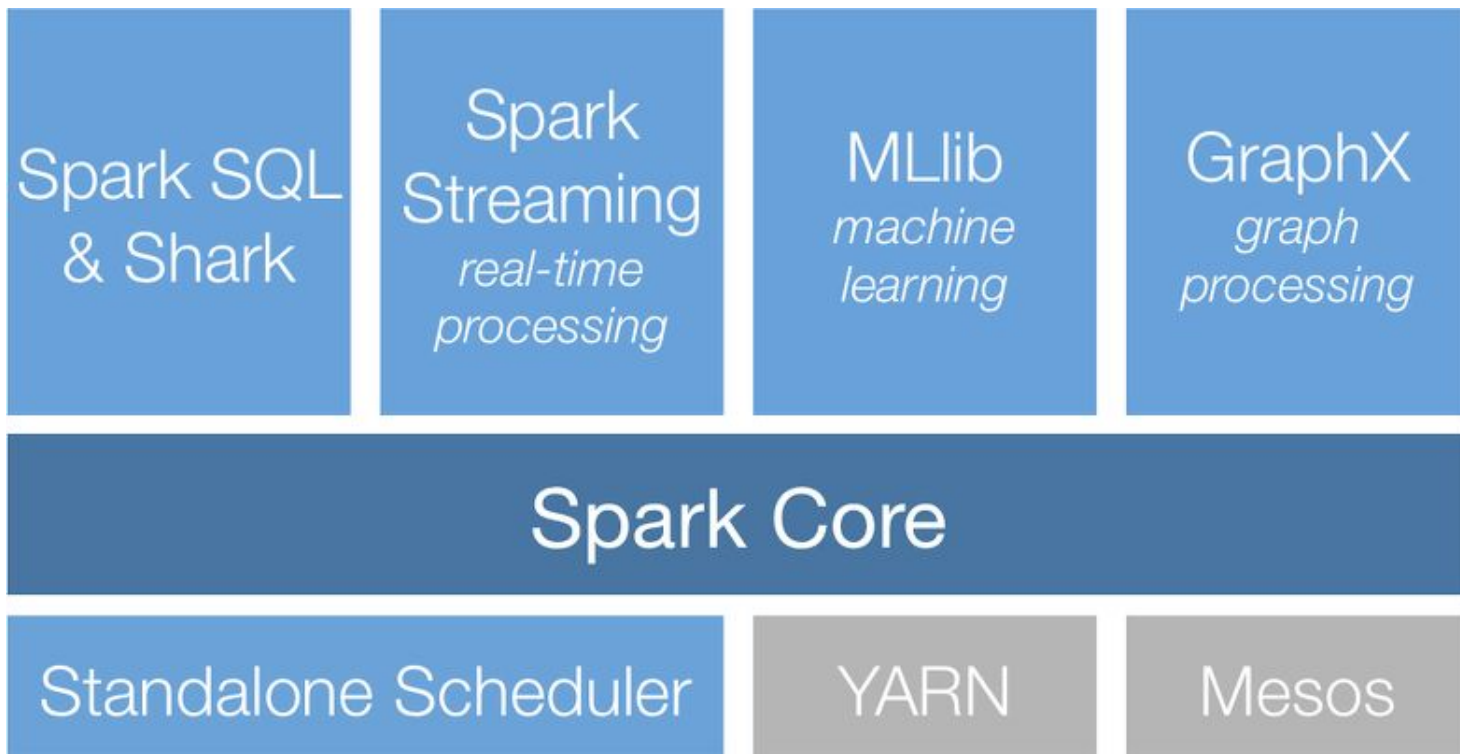
# RDD (Resilient Distributed Datasets)

- Поддерживаются два типа операций:
  - преобразования (map, reduceByKey, join, ...)
    - задают преобразование отдельно для каждой строки
    - возвращают ссылку на новый RDD
    - вычисления по требованию (lazy computations)
    - сохраняется последовательность преобразований для восстановления в случае отказа ноды кластера
  - действия (min, max, count, ...)
    - иницируют расчет и возвращают значения

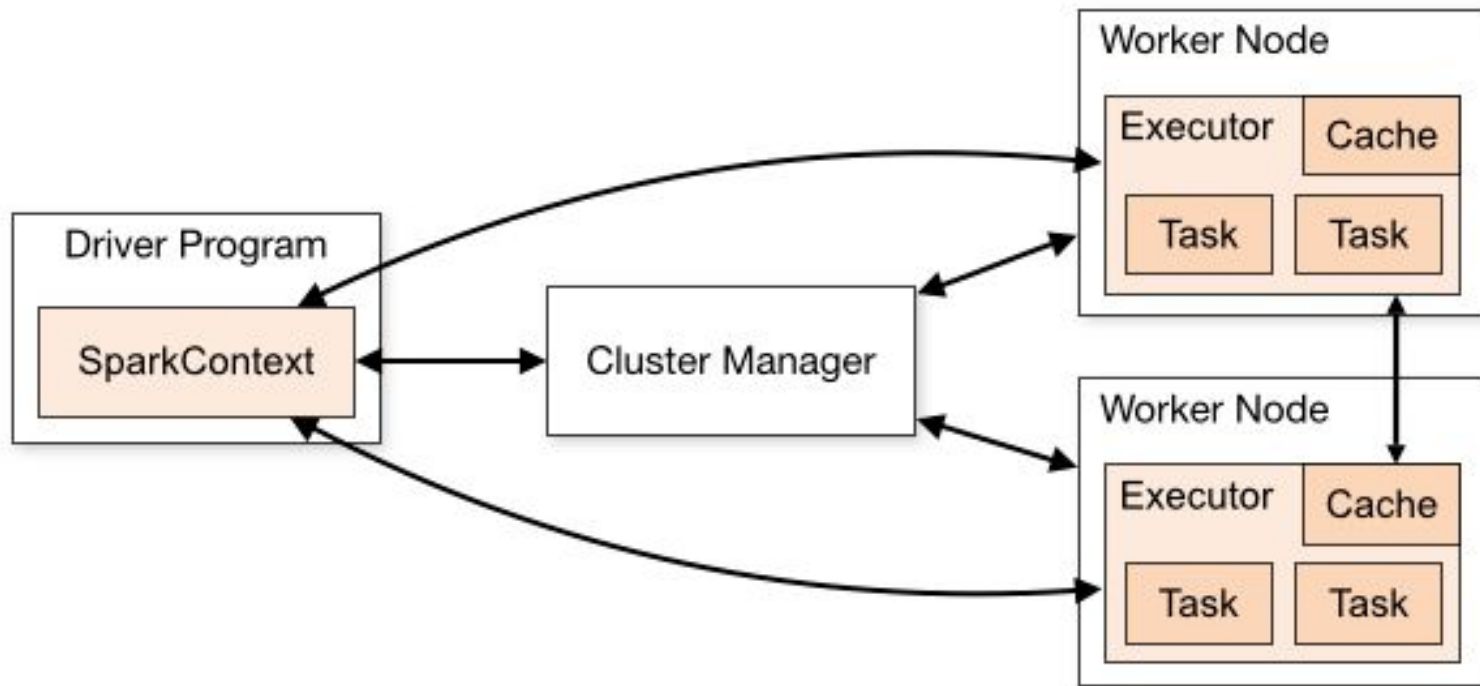
# Apache Spark

- Высокопроизводительная система распределенных вычислений
- Основная часть реализована на *функциональном* языке Scala
- Поддерживает API для Python, Java, Scala и R
- Основные компоненты
  - SparkSQL
  - MLlib
  - Spark Streaming
  - GraphX

# Apache Spark



# Apache Spark



# Apache Spark

- Запуск кластера Spark
  - [Local](#) – запуск на локальной машине, полезно для отладки
  - [Standalone](#) – автономный кластер, входит в состав Spark
  - [Hadoop YARN](#) – поверх кластера Hadoop
  - [Apache Mesos](#) – поверх [Mesos](#) (менеджер ресурсов кластера)
- Запуск задач на кластере
  - [Интерактивная](#) консоль (./bin/spark-shell, ./bin/pyspark)
  - Запуск приложений с помощью утилиты [./bin/spark-submit](#)
  - Запуск приложений через API

# Apache Spark - Standalone Mode

- `$SPARK_HOME/sbin/start-master.sh`
- `$SPARK_HOME/sbin/start-slave.sh spark://<master host>`



Spark Master on ec2-54-234-77-87.compute-1.amazonaws.com:7077

URL: spark://ec2-54-234-77-87.compute-1.amazonaws.com:7077

Workers: 4

Cores: 8 Total, 0 Used

Memory: 62.7 GB Total, 0.0 B Used

Jobs: 0 Running, 0 Completed


## Cluster Summary

ID	Address	State	Cores	Memory
worker-20130205193620-ip-10-157-5-198.ec2.internal-38335	ip-10-157-5-198.ec2.internal-38335	ALIVE	2 (0 Used)	15.7 GB (0.0 B Used)
worker-20130205193620-ip-10-30-143-161.ec2.internal-42928	ip-10-30-143-161.ec2.internal-42928	ALIVE	2 (0 Used)	15.7 GB (0.0 B Used)
worker-20130205193620-ip-10-30-152-34.ec2.internal-36862	ip-10-30-152-34.ec2.internal-36862	ALIVE	2 (0 Used)	15.7 GB (0.0 B Used)
worker-20130205193620-ip-10-30-152-4.ec2.internal-40594	ip-10-30-152-4.ec2.internal-40594	ALIVE	2 (0 Used)	15.7 GB (0.0 B Used)

## Running Jobs

JobID	Description	Cores	Memory per Node	Submit Time	User	State	Duration
-------	-------------	-------	-----------------	-------------	------	-------	----------

# Концепция вычислений на Spark

 jupyter sparkwordcount Last Checkpoint: 7 minutes ago (autosaved)

File Edit View Insert Cell Kernel Help

```
In [ ]: from pyspark import SparkContext
        # creating spark context
        sc = SparkContext('local', 'WordCount App')
```

```
In [ ]: # loading text from file
        with open('../data/idiot.txt') as src:
            text = src.readlines()

        # making RDD from text lines
        text_rdd = sc.parallelize(text)

        # counting word entries and storing result as RDD
        wc_rdd = text_rdd.flatMap(lambda line: line.split()) \
            .map(lambda word: (word, 1)) \
            .reduceByKey(lambda x, y: x + y)

        # getting result back to client
        wc = wc_rdd.collect()

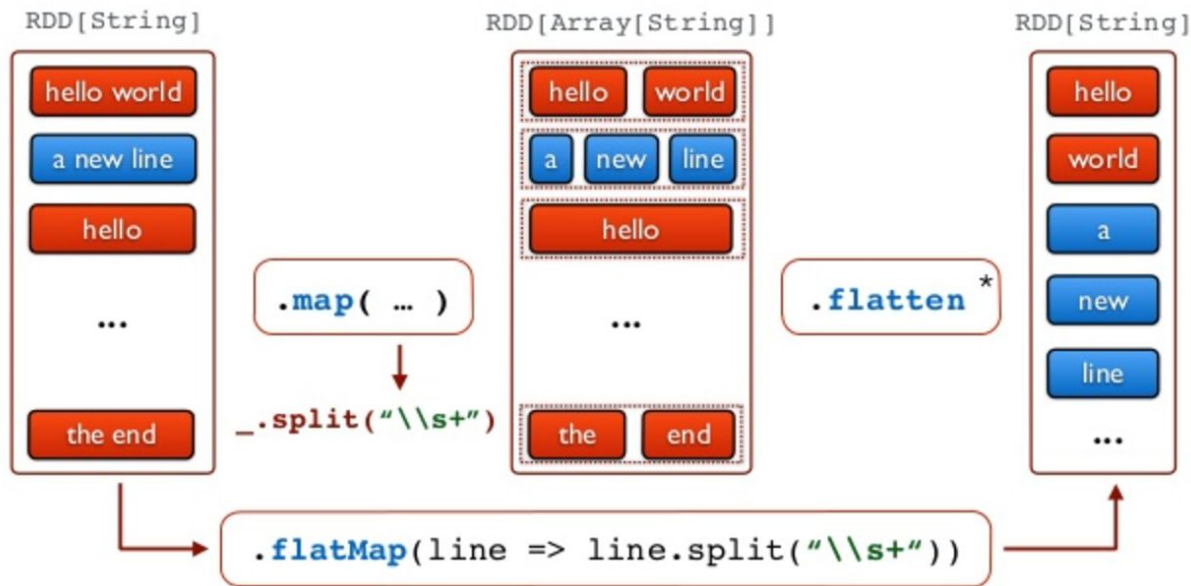
        # output 10 most frequent words
        print sorted(wc, key=lambda e: -e[-1])[ :10]
```

# Концепция вычислений на Spark - SparkContext

- SparkContext
  - один экземпляр на приложение
  - создание RDD и хранение мета информации
    - `sc.parallelize`
    - `sc.textFile`
    - `sc.wholeTextFiles`
  - отвечает за запуск приложений на Spark кластере
  - управление настройками и процессом выполнения приложений
  - получение информации о статусе выполнения приложений



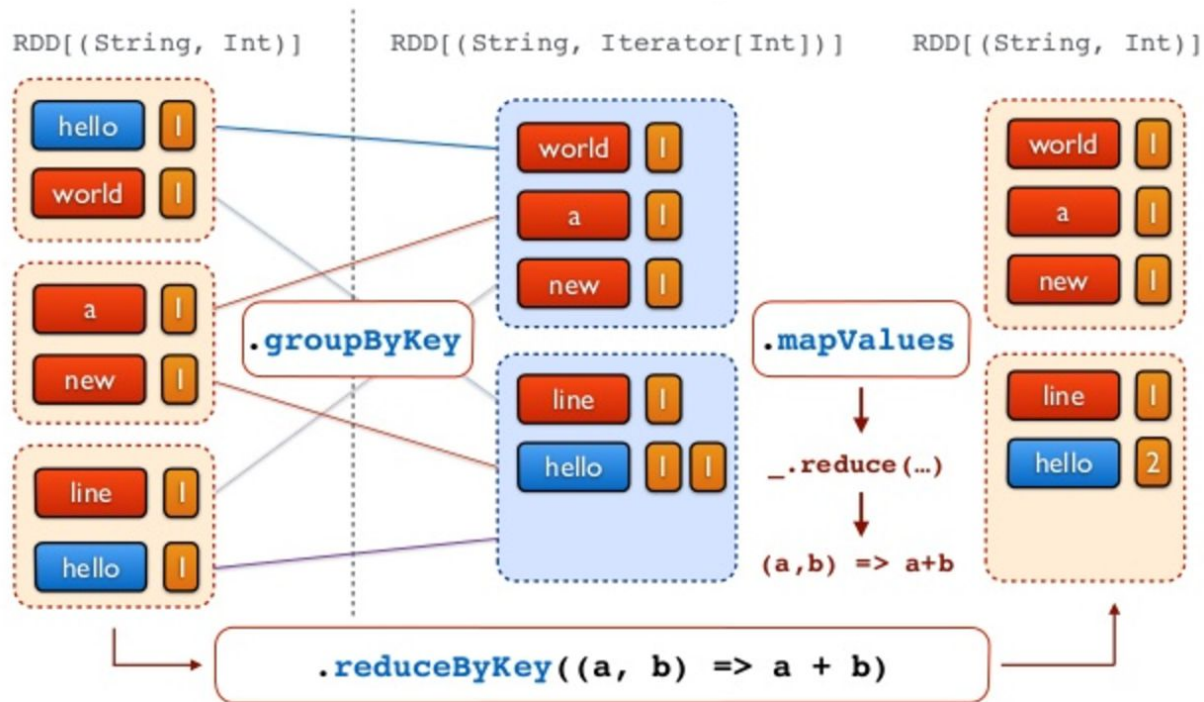
# Концепция вычислений на Spark - flatMap



```
// Step 3 - Split lines into words  
val words = lower.flatMap(line => line.split("\\s+"))
```


Note: `flatten()` not available in spark, only `flatMap`

# Концепция вычислений на Spark - reduceByKey













```
// Step 5 - Count all words  
val freq = counts.reduceByKey(_ + _)
```

# DataFrame и SQL запросы

 jupyter sparksql Last Checkpoint: 2 minutes ago (autosaved)

File Edit View Insert Cell Kernel Help

         Code  CellToolbar

```
In [ ]: from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('SQL App').getOrCreate()
sc = spark.sparkContext

In [ ]: !cat ../data/sales_header.csv

In [ ]: def parse_row(line):
# TODO: parse row values
return line.split(',')

sales_rdd = sc.textFile('../data/sales.csv').map(parse_row)

In [ ]: with open('../data/sales_header.csv') as src:
sales_header = src.read().strip().split(',')

sales_df = sales_rdd.toDF(sales_header)
sales_df.registerTempTable('sales')

sales_by_state_df = spark.sql("SELECT SUM(1) FROM sales")
print sales_by_state_df.rdd.collect()

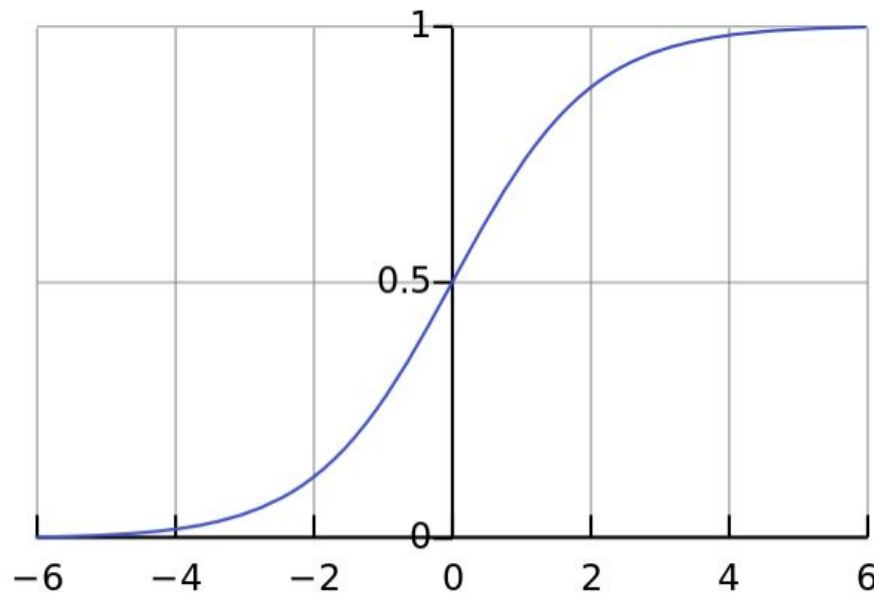
# TODO: print revenue by Country and State
```

# DataFrame и SQL запросы

- `pyspark.sql.`[SparkSession](#)
  - точка входа для работы со SparkSQL
  - содержит ссылку на `SparkContext` (создается при необходимости)
- [DataFrame](#)
  - табличное представление данных
  - набор типизированных записей `pyspark.sql.Row`
  - содержит метаданные об именах и типах полей (схему)
  - является надстройкой над RDD


# Задание: логистическая регрессия

$$y(x) = \frac{1}{1 + \exp(-xw^T)}$$














$$w = w - \lambda \frac{1}{N} \sum_{i=1}^N (y(x_i) - t_i) x_i - \beta w$$

# Задание: логистическая регрессия

 jupyter sparklogit Last Checkpoint: 5 minutes ago (autosaved)

File Edit View Insert Cell Kernel Help

          Code  CellToolbar

```
In [1]: import numpy as np
        from pyspark import SparkContext
        sc = SparkContext('local', 'Logistic Regression App')
```

```
In [2]: def y(x, w):
        # TODO: implement logit function

        def loss(data, w):
            predicted = map(lambda (_, x): max(y(x, w), 1e-32), data)
            return - sum(t * np.log(p) + (1. - t) * np.log(1. - p)
                        for p, (t, x) in zip(predicted, data)) / len(data)

        def parse_row(line):
            values = map(float, line.split(','))
            return values[0], np.array([1.] + values[1:])

        def load_data(path):
            with open(path) as src:
                return map(parse_row, src)
```

```
In [4]: train_data = load_data('../data/logit_train.csv')
        test_data = load_data('../data/logit_test.csv')
```

```
In [ ]: train_data_rdd = sc.parallelize(train_data).persist()
```

# Полезные материалы

- Advanced Analytics with Spark  
<http://shop.oreilly.com/product/0636920035091.do>
- Mastering Apache Spark 2  
<https://www.gitbook.com/book/jaceklaskowski/mastering-apache-spark/details>
- Spark Overview  
<http://spark.apache.org/>
- Cluster Mode Overview  
<http://spark.apache.org/docs/latest/cluster-overview.html>
- Tuning Spark  
<http://spark.apache.org/docs/latest/tuning.html>

# Полезные материалы

- Databricks Blog  
<https://databricks.com/blog>
- Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing  
[https://cs.stanford.edu/~matei/papers/2012/nsdi\\_spark.pdf](https://cs.stanford.edu/~matei/papers/2012/nsdi_spark.pdf)
- Spark Misconceptions  
<https://0x0fff.com/spark-misconceptions/>
- Spark Architecture: Shuffle  
<https://0x0fff.com/spark-architecture-shuffle/>