

# Основы Spark



# План на неделю



Сравнение Spark и MapReduce



# План на неделю



Сравнение Spark и MapReduce



Spark RDD API



# План на неделю



Сравнение Spark и MapReduce



Spark RDD API



Spark DataFrame API



# Apache Spark

➔ Фреймворк для распределенных вычислений



[Spark.apache.org](http://Spark.apache.org)

# Apache Spark

- ➔ Фреймворк для распределенных вычислений
- ➔ API на многих языках: Scala, Java, Python (PySpark)



# Apache Spark

- ➔ Фреймворк для распределенных вычислений
- ➔ API на многих языках: Scala, Java, Python (PySpark)
- ➔ Внутри много всего: Spark ML, Spark SQL, Spark Streaming



# **Spark и MapReduce**





# SQL join запрос

➔ Таблица **a** — покупки пользователей (**user**, product, ...)



# SQL join запрос

- ➔ Таблица **a** — покупки пользователей (**user**, product, ...)
- ➔ Таблица **b** — информация о пользователях (**user**, country, ...)



# SQL join запрос

- ➔ Таблица **a** — покупки пользователей (**user**, product, ...)
- ➔ Таблица **b** — информация о пользователях (**user**, country, ...)
- ➔ Хотим получить покупки продуктов по странам

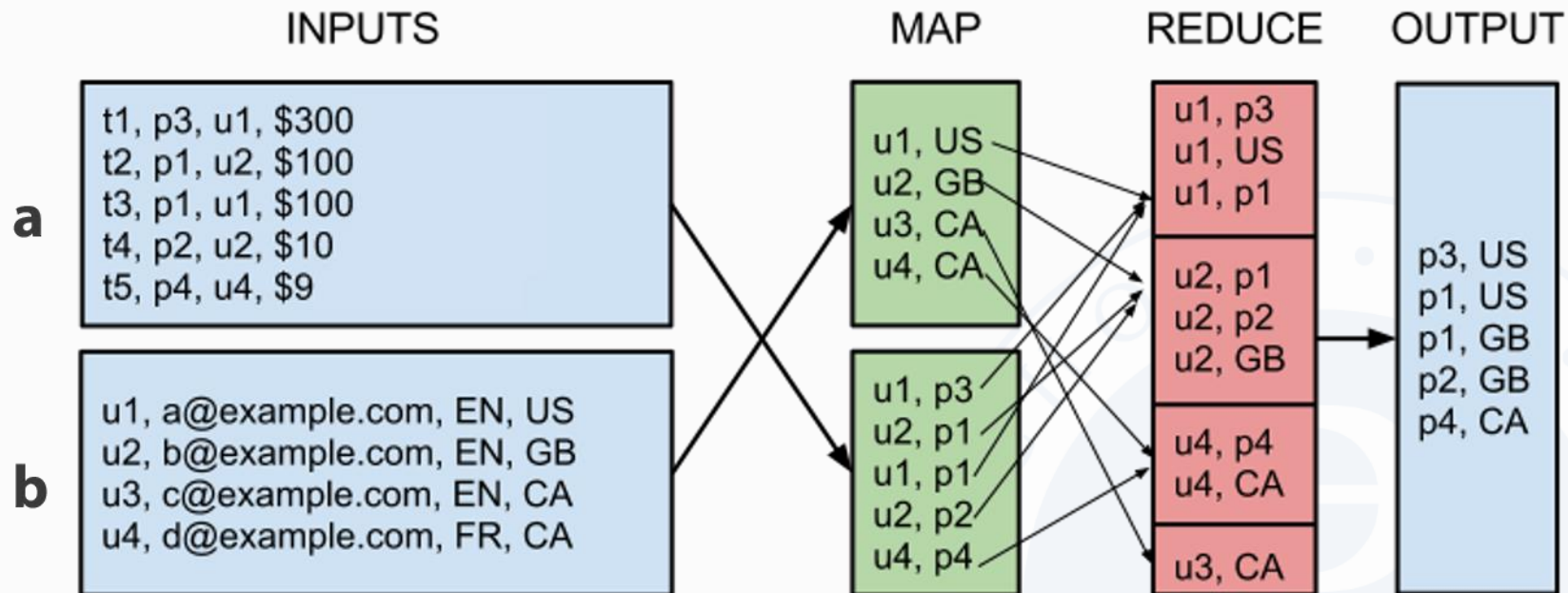


# SQL join запрос

- ➔ Таблица **a** — покупки пользователей (**user**, product, ...)
- ➔ Таблица **b** — информация о пользователях (**user**, country, ...)
- ➔ Хотим получить покупки продуктов по странам
- ➔ Нужно сделать join по **user**

```
select
    a.product,
    b.country
from
    a join b on a.user = b.user
```

# SQL join на MapReduce



# Еще один join на MapReduce

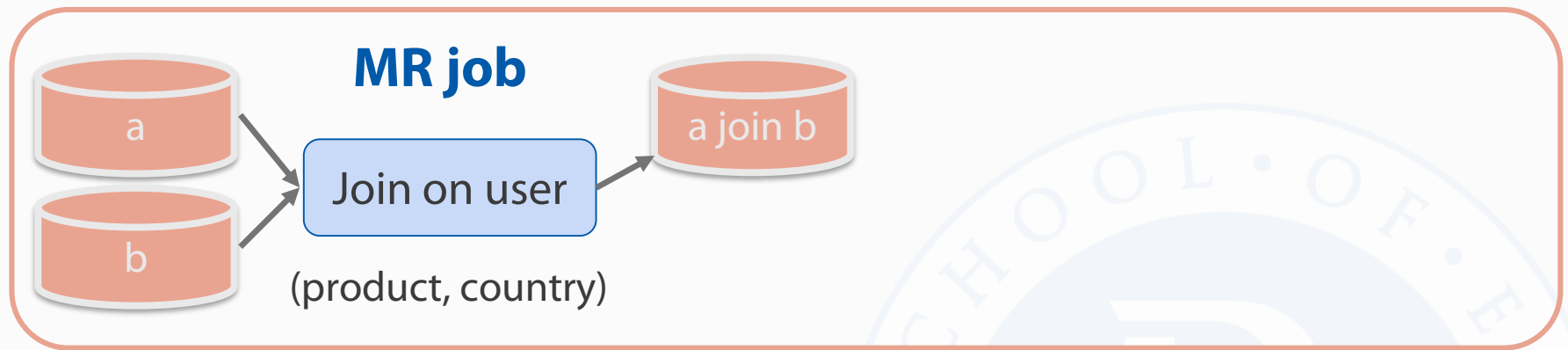
➔ В таблице **c** лежит информация о продуктах (**product**, type)



# Еще один join на MapReduce

➔ В таблице **c** лежит информация о продуктах (**product**, type)

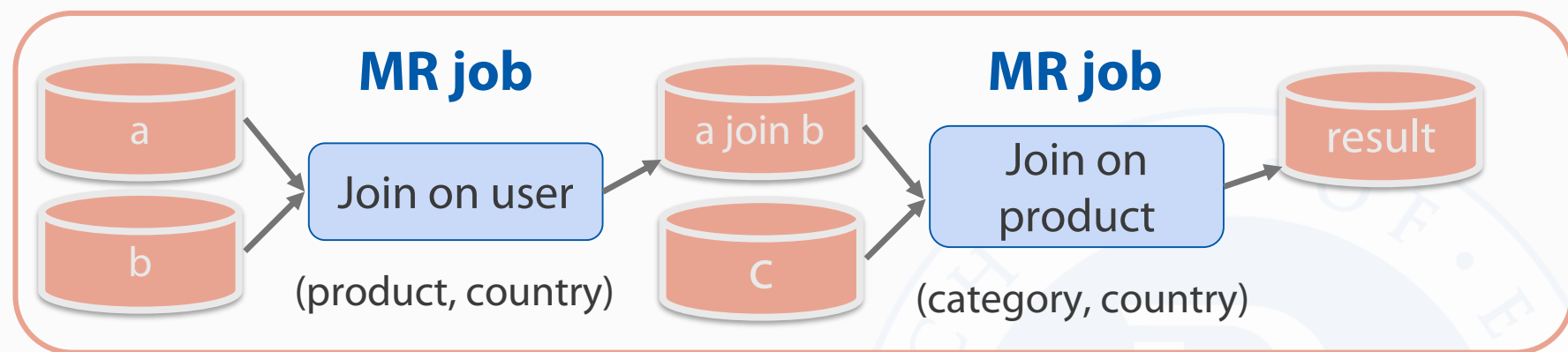
➔ Два join на MapReduce:



# Еще один join на MapReduce

➔ В таблице **c** лежит информация о продуктах (**product**, type)

➔ Два join на MapReduce:

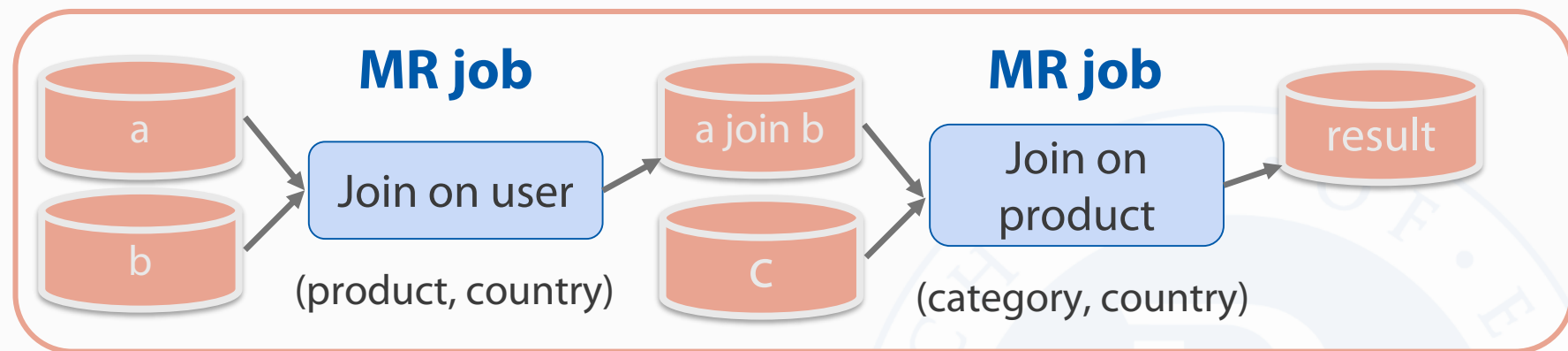




# Еще один join на MapReduce

→ В таблице **c** лежит информация о продуктах (**product**, type)

→ Два join на MapReduce:

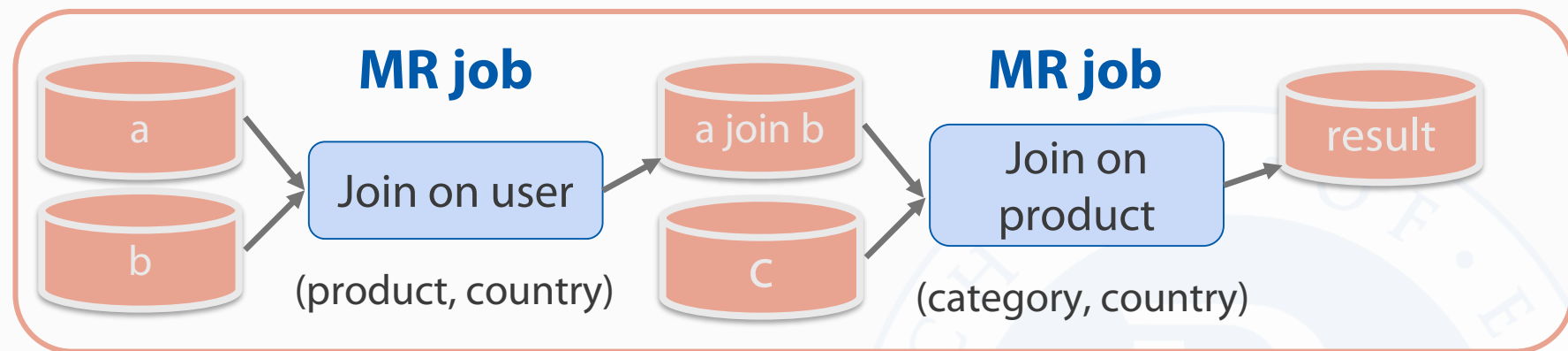


→ MapReduce хранит результаты в HDFS

# Еще один join на MapReduce

→ В таблице **c** лежит информация о продуктах (**product**, type)

→ Два join на MapReduce:

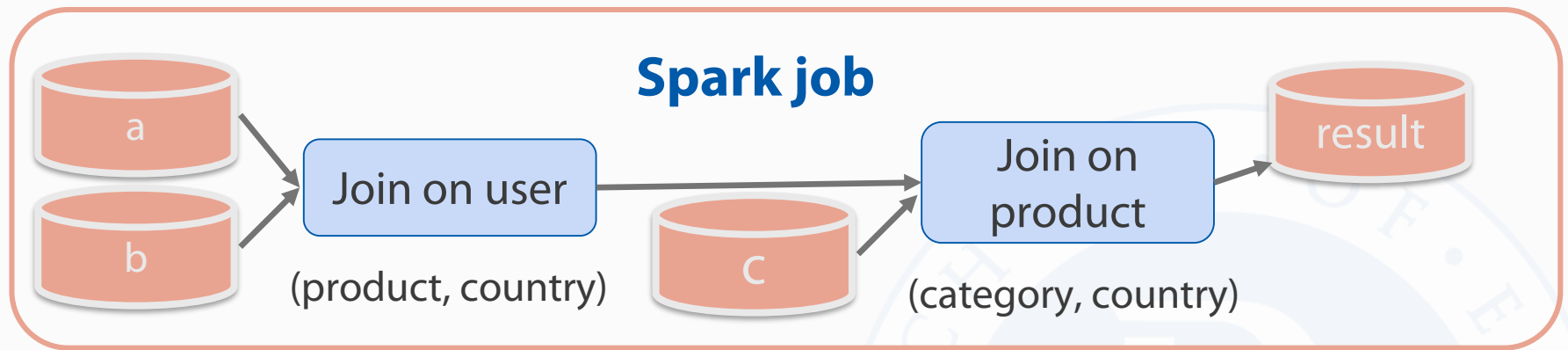


→ MapReduce хранит результаты в HDFS

→ Поэтому для «a join b» мы тратим время на запись в HDFS и тут же читаем эти данные обратно

# Вот тут-то и поможет Spark

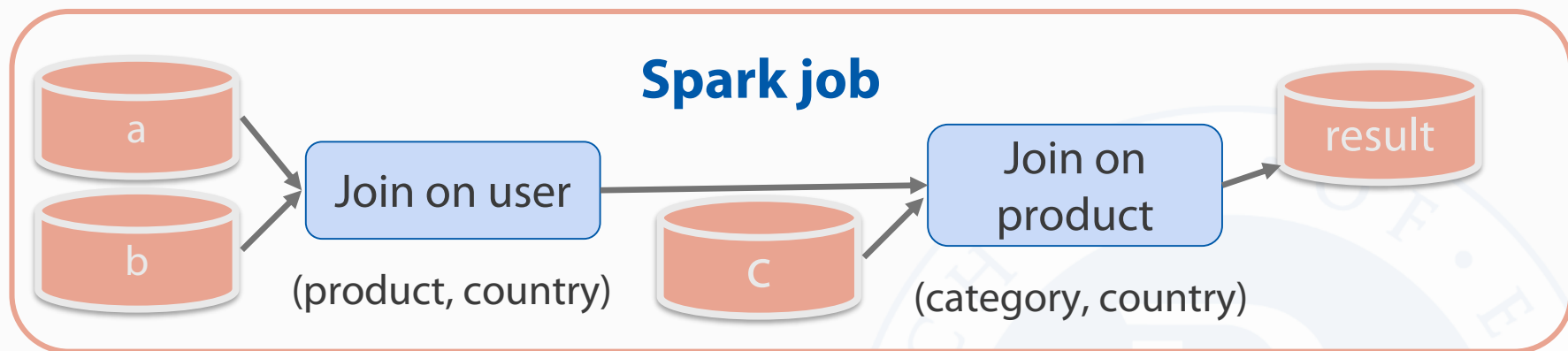
➔ Два join на Spark:



➔ Вычисления описываются как DAG (Directed Acyclical Graph)

# Вот тут-то и поможет Spark

➔ Два join на Spark:

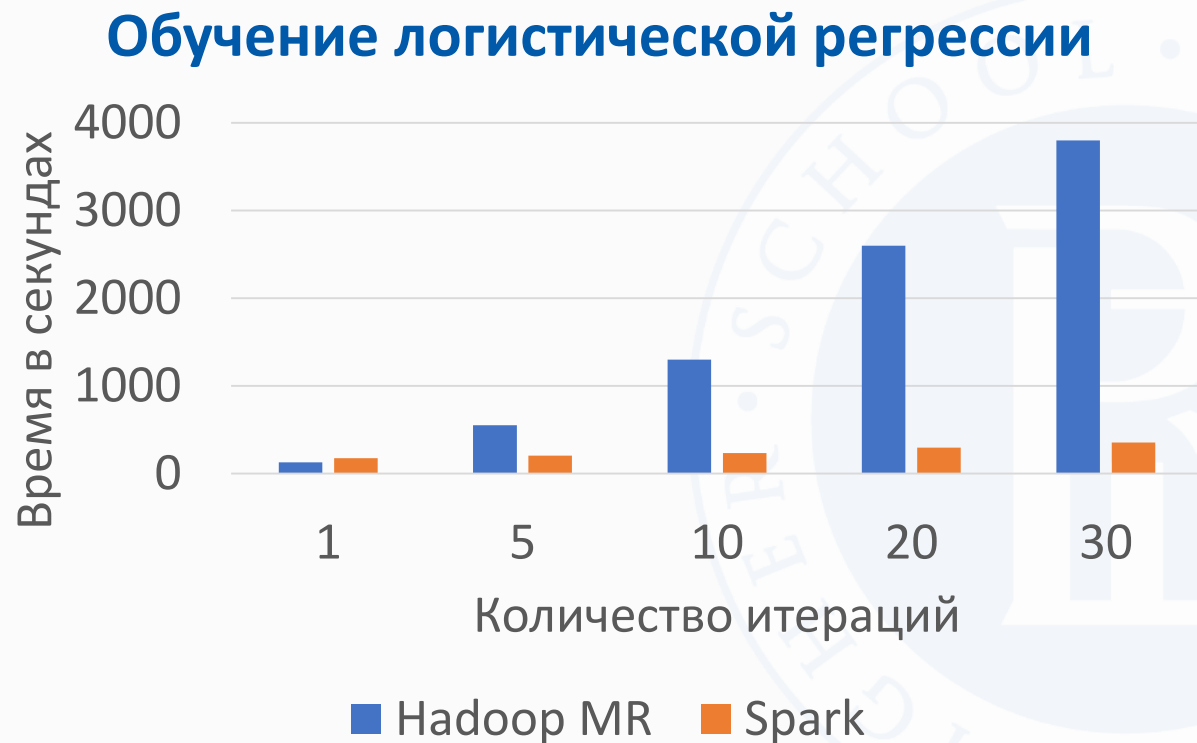


➔ Вычисления описываются как DAG (Directed Acyclical Graph)

➔ Промежуточные результаты хранятся в памяти или на диске, минуя HDFS

# Итерационные алгоритмы

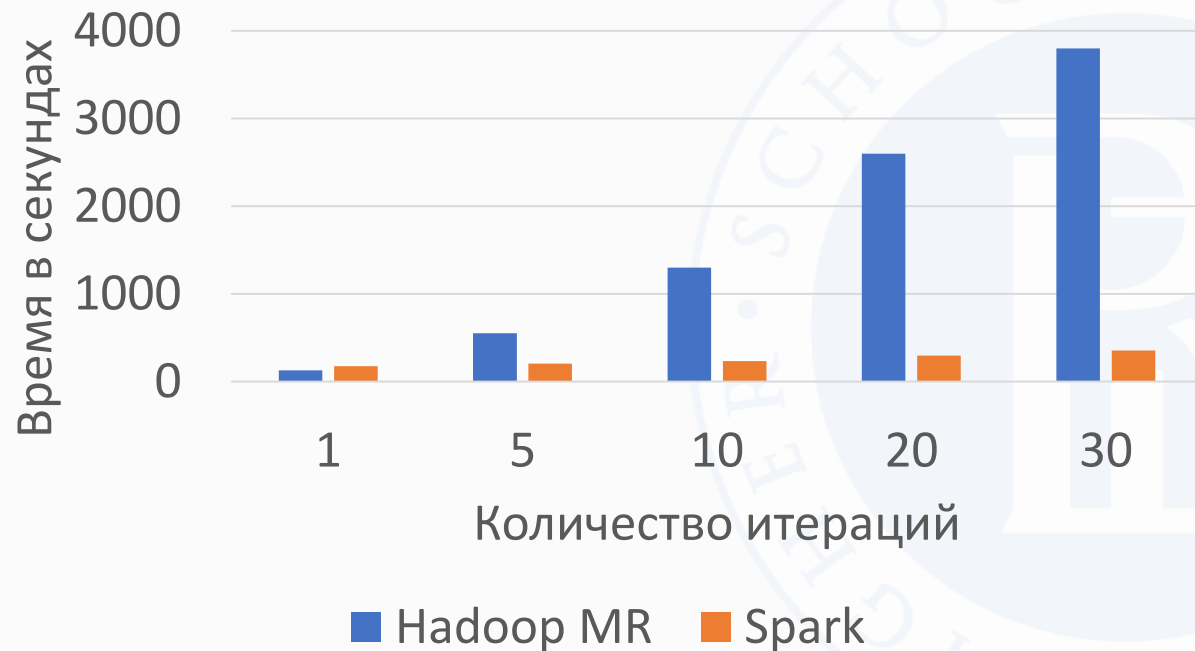
- ➔ В MapReduce есть overhead на запись/чтение в HDFS, запуск каждого шага в YARN



# Итерационные алгоритмы

- ➔ В MapReduce есть overhead на запись/чтение в HDFS, запуск каждого шага в YARN
- ➔ В ML много итерационных алгоритмов, и Spark идеально подходит для таких задач

## Обучение логистической регрессии



# Spark vs MapReduce

## Spark

## MapReduce

Область применения	Итерационные, интерактивные вычисления	Тяжелая пакетная обработка данных
Простота использования	Удобное API на Python	Hadoop streaming с неудобным интерфейсом
Утилизация RAM	Хранит данные в памяти, когда может	Все данные хранятся в HDFS

# Резюме



В реальных задачах требуется несколько MapReduce шагов для решения, промежуточные результаты хранятся в HDFS





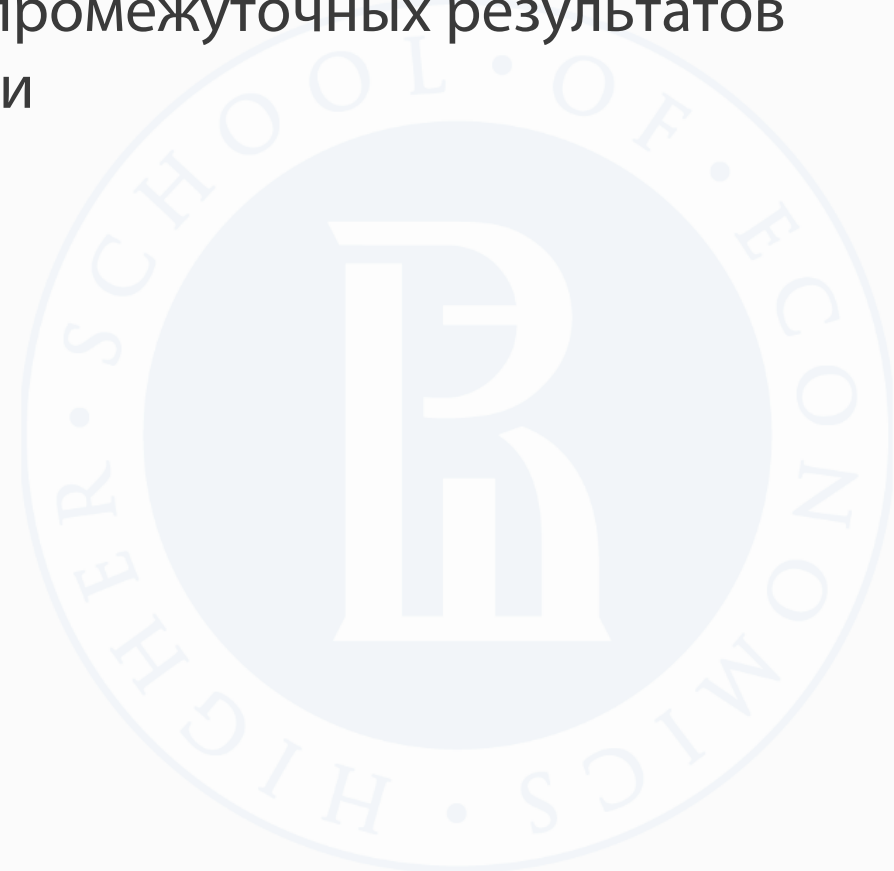
# Резюме



В реальных задачах требуется несколько MapReduce шагов для решения, промежуточные результаты хранятся в HDFS



Spark описывает вычисления в виде графа и может оптимизировать хранение промежуточных результатов вплоть до хранения в памяти



# Резюме



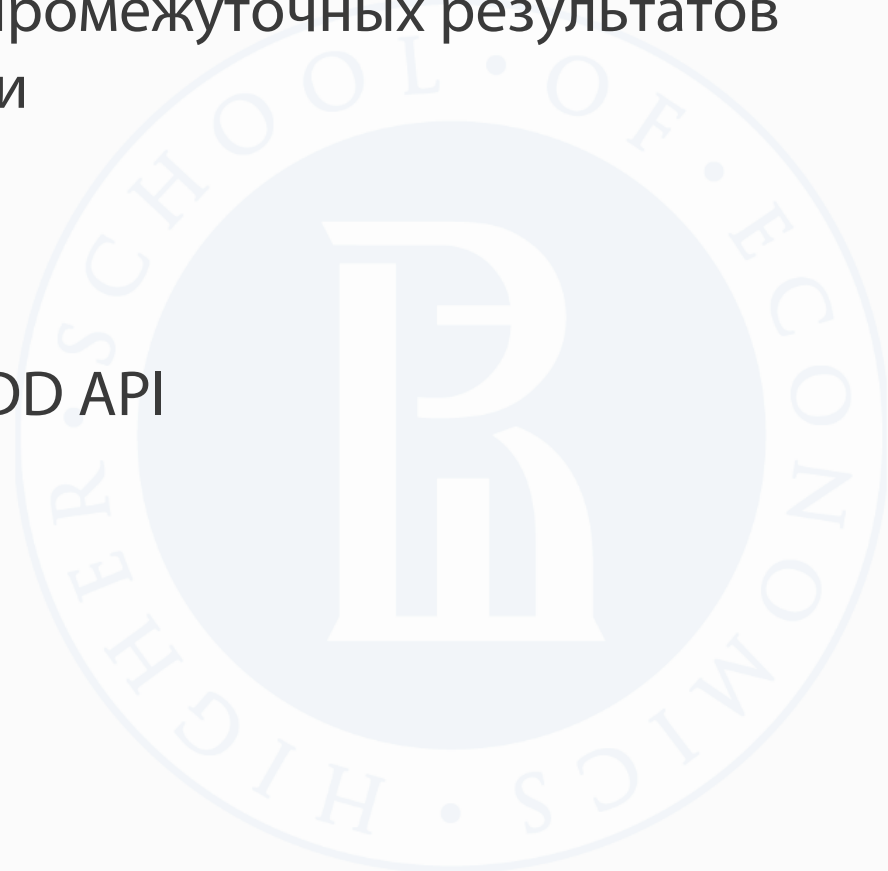
В реальных задачах требуется несколько MapReduce шагов для решения, промежуточные результаты хранятся в HDFS



Spark описывает вычисления в виде графа и может оптимизировать хранение промежуточных результатов вплоть до хранения в памяти



Далее: поговорим о Spark RDD API



# Spark RDD API



# Spark RDD

➔ В Spark вычисления описываются операциями над RDD



# Spark RDD

Абстракция RDD (Resilient Distributed Dataset):

➔ Восстанавливаемый распределенный набор данных



# Spark RDD

Абстракция RDD (Resilient Distributed Dataset):

- ➔ Восстанавливаемый распределенный набор данных
- ➔ Входы операций должны быть RDD



# Spark RDD

Абстракция RDD (Resilient Distributed Dataset):

- ➔ Восстанавливаемый распределенный набор данных
- ➔ Входы операций должны быть RDD
- ➔ Все промежуточные результаты будут RDD, так как известна цепочка вычислений (DAG) и потерянные части легко восстановить из входных данных



# Spark RDD

Как сделать RDD:

➔ Файл из HDFS (уже восстанавливаемый и распределенный)





# Spark RDD

Как сделать RDD:

- ➔ Файл из HDFS (уже восстанавливаемый и распределенный)
- ➔ Распараллелив Python коллекцию (список, итератор, ...)



# Spark RDD

Как сделать RDD:

- ➔ Файл из HDFS (уже восстанавливаемый и распределенный)
- ➔ Распараллелив Python коллекцию (список, итератор, ...)
- ➔ Трансформацией из другого RDD



# Операции над RDD

## ➔ Трансформации (RDD → RDD):

Трансформации ленивые (вычисляются, когда будут нужны)

Пример: `map` применяет преобразование к каждому элементу RDD и возвращает новый RDD с результатом  
Еще примеры: `reduceByKey`, `join`



# Операции над RDD

## → Трансформации (RDD → RDD):

Трансформации ленивые (вычисляются, когда будут нужны)

Пример: `map` применяет преобразование к каждому элементу RDD и возвращает новый RDD с результатом

Еще примеры: `reduceByKey`, `join`

## → Действия:

Действия приводят к запуску DAG для расчета RDD

Примеры: `saveAsTextFile`, `collect`, `count`

# Операции над RDD

## → Трансформации (RDD → RDD):

Трансформации ленивые (вычисляются, когда будут нужны)

Пример: `map` применяет преобразование к каждому элементу RDD и возвращает новый RDD с результатом

Еще примеры: `reduceByKey`, `join`

## → Действия:

Действия приводят к запуску DAG для расчета RDD

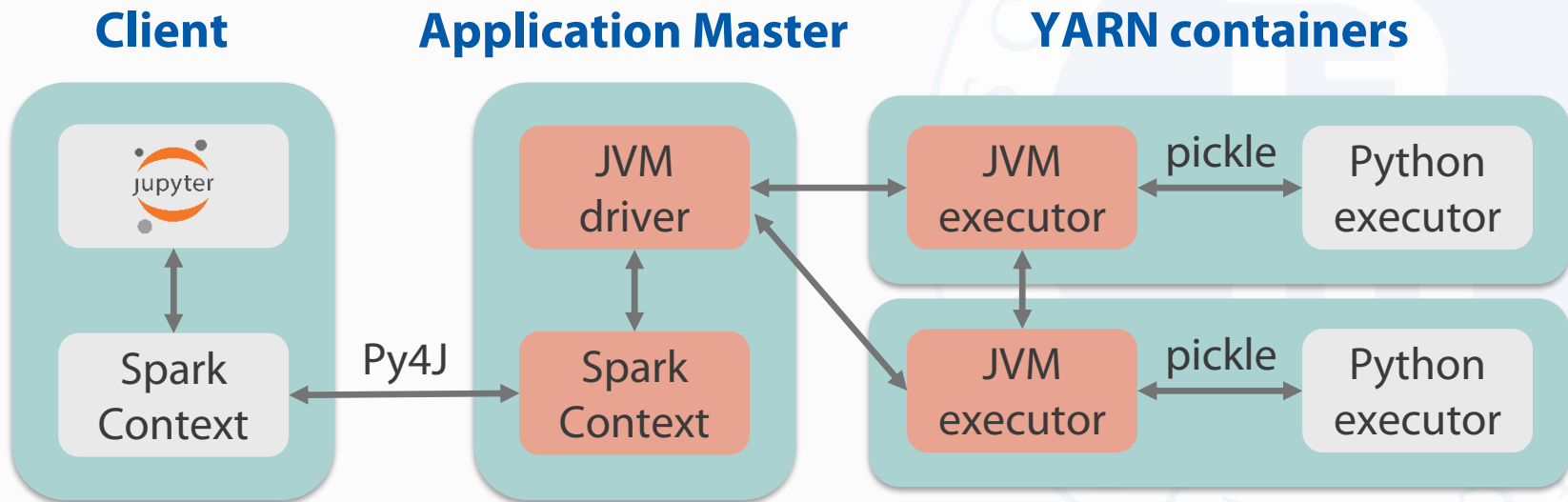
Примеры: `saveAsTextFile`, `collect`, `count`

## → Другие операции:

Примеры: `persist`, `cache` заставляют Spark сохранить RDD в памяти для последующего быстрого доступа

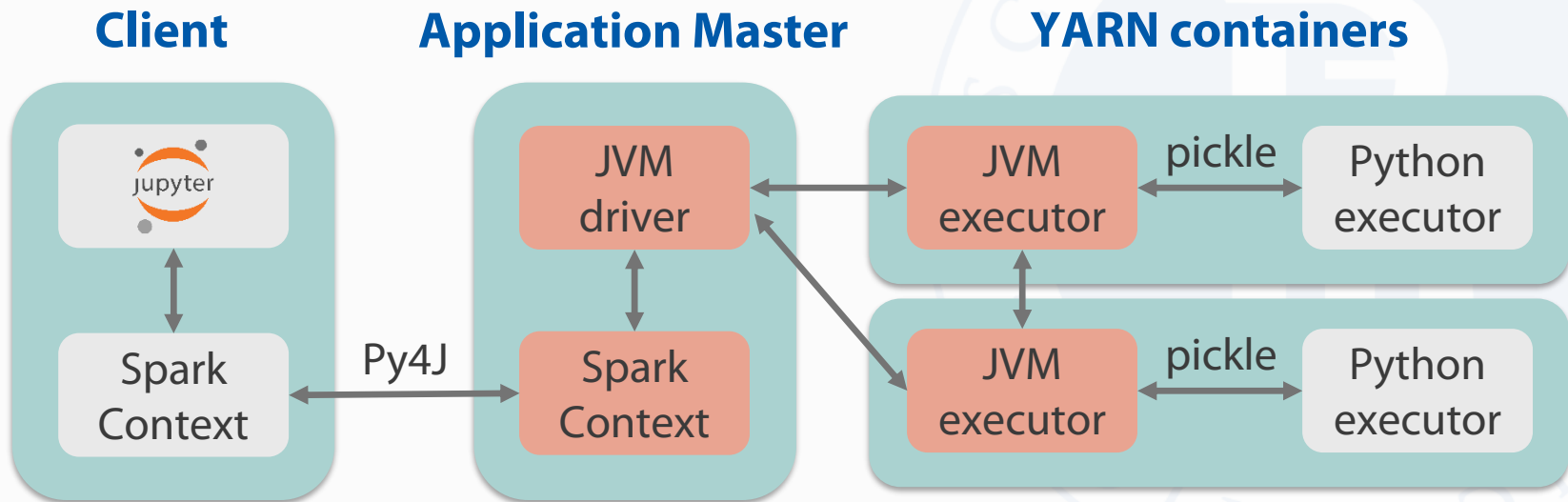
# Как устроена программа на PySpark (yarn)

➔ При создании в Python `SparkContext` запускается YARN приложение



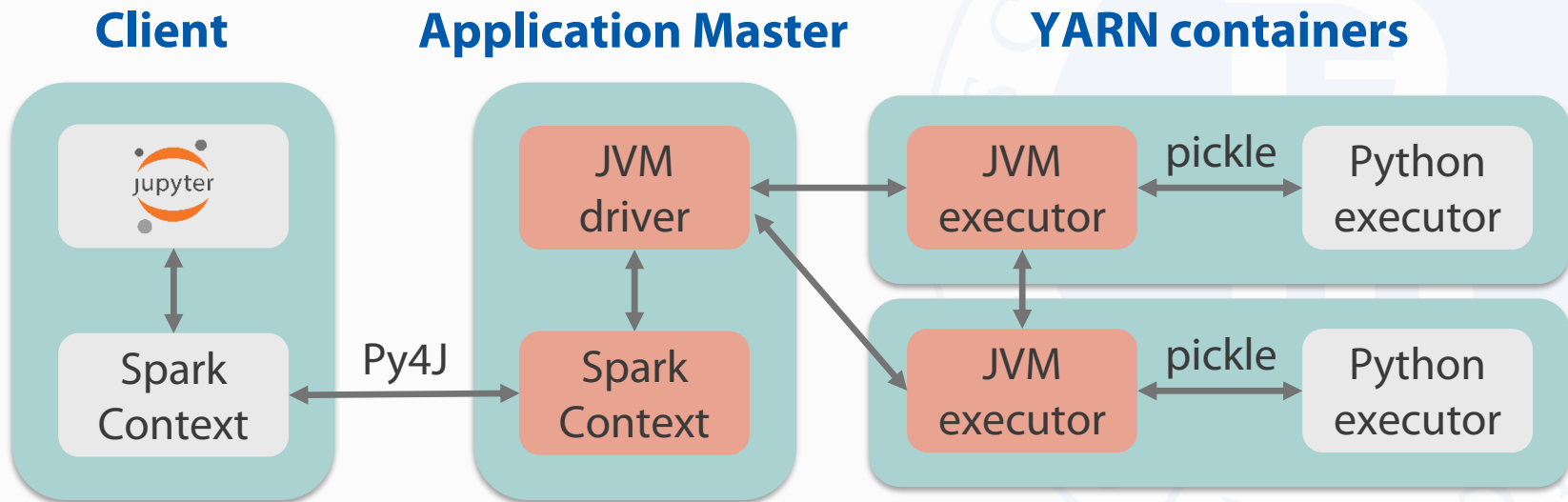
# Как устроена программа на PySpark (yarn)

- ➔ При создании в Python `SparkContext` запускается YARN приложение
- ➔ В Application Master запускается `driver`, который создает JVM версию `SparkContext`



# Как устроена программа на PySpark (yarn)

- ➔ При создании в Python `SparkContext` запускается YARN приложение
- ➔ В Application Master запускается `driver`, который создает JVM версию `SparkContext`
- ➔ `Driver` координирует работу `Executors` (вычисляют RDD)

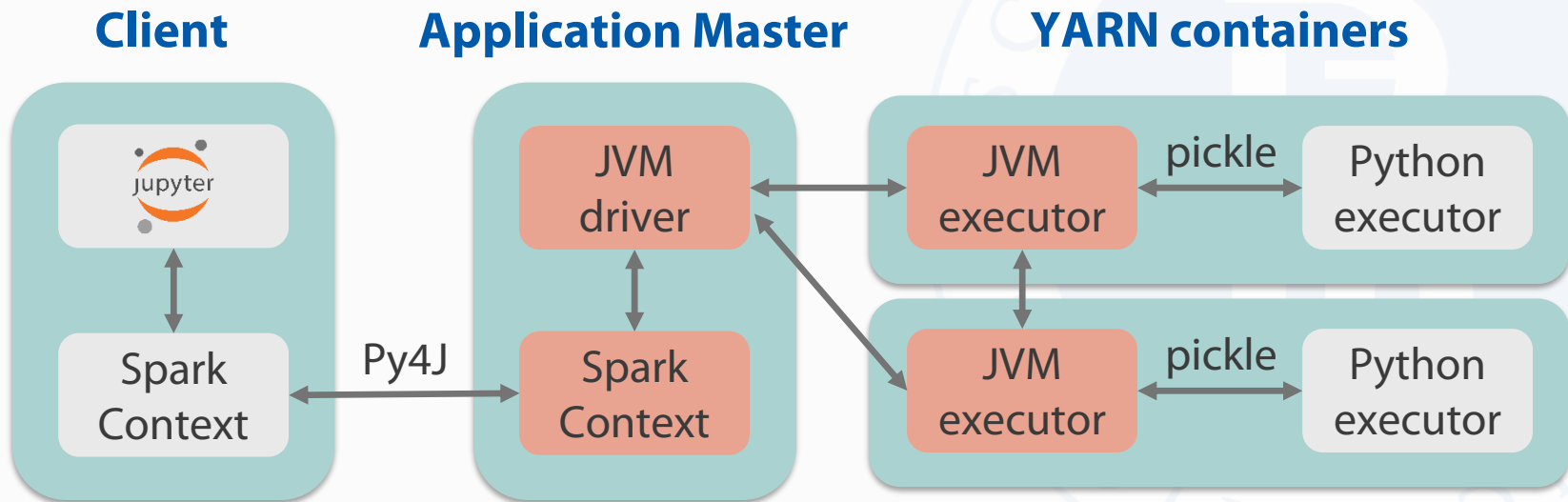




# Как устроена программа на PySpark (yarn)

Spark работает с Python объектами в сериализованном виде (pickle), они будут десериализованы для обработки в Python, поэтому элементы RDD занимают память **дважды**:

- ➔ В сериализованном виде в JVM (например, кэш в RAM)
- ➔ Десериализованный объект в Python



# Простейшая программа на PySpark

```
rdd = (sc                                     # SparkContext
      .parallelize([1, 2, 3, 4])             # создаем RDD
      .map(lambda x: x * 2))                 # трансформируем RDD
print rdd                                     # ленивые вычисления
print rdd.collect()                         # запускаем DAG
```

```
PythonRDD[17] at RDD at PythonRDD.scala:48
[2, 4, 6, 8]
```



# Резюме



Программа на Spark — это набор операций над RDD



# Резюме






Программа на Spark — это набор операций над RDD



Вычисления в Spark ленивые, только действия над RDD приводят к запуску вычислений DAG




# Резюме






-  Программа на Spark — это набор операций над RDD
-  Вычисления в Spark ленивые, только действия над RDD приводят к запуску вычислений DAG
-  SparkContext — точка входа для работы со Spark



# Резюме

-  Программа на Spark — это набор операций над RDD
-  Вычисления в Spark ленивые, только действия над RDD приводят к запуску вычислений DAG
-  SparkContext — точка входа для работы со Spark
-  PySpark не очень эффективен по памяти, зато позволяет работать с произвольными Python объектами через pickle

# Резюме

-  Программа на Spark — это набор операций над RDD
-  Вычисления в Spark ленивые, только действия над RDD приводят к запуску вычислений DAG
-  SparkContext — точка входа для работы со Spark
-  PySpark не очень эффективен по памяти, зато позволяет работать с произвольными Python объектами через pickle
-  Далее: поговорим о Spark SQL

# Spark SQL





# Spark SQL

➔ В Spark кроме RDD API есть еще DataFrame API



[Spark.apache.org](https://spark.apache.org)

# Spark SQL

- ➔ В Spark кроме RDD API есть еще DataFrame API
- ➔ DataFrame хранит табличные данные (как в Pandas)



# Spark SQL

- ➔ В Spark кроме RDD API есть еще DataFrame API
- ➔ DataFrame хранит табличные данные (как в Pandas)
- ➔ DataFrame поддерживает SQL запросы (на кластере)



# Spark SQL

- ➔ В Spark кроме RDD API есть еще DataFrame API
- ➔ DataFrame хранит табличные данные (как в Pandas)
- ➔ DataFrame поддерживает SQL запросы (на кластере)
- ➔ Можно конвертировать из/в Pandas DataFrame



# Spark SQL

- ➔ В Spark кроме RDD API есть еще DataFrame API
- ➔ DataFrame хранит табличные данные (как в Pandas)
- ➔ DataFrame поддерживает SQL запросы (на кластере)
- ➔ Можно конвертировать из/в Pandas DataFrame
- ➔ DataFrame обрабатывается целиком в JVM  
(> 10x быстрее Python)



# Создание DataFrame

→ Создаем Spark SQL сессию:

```
sc = pyspark.SparkContext()  
se = SparkSession(sc)
```



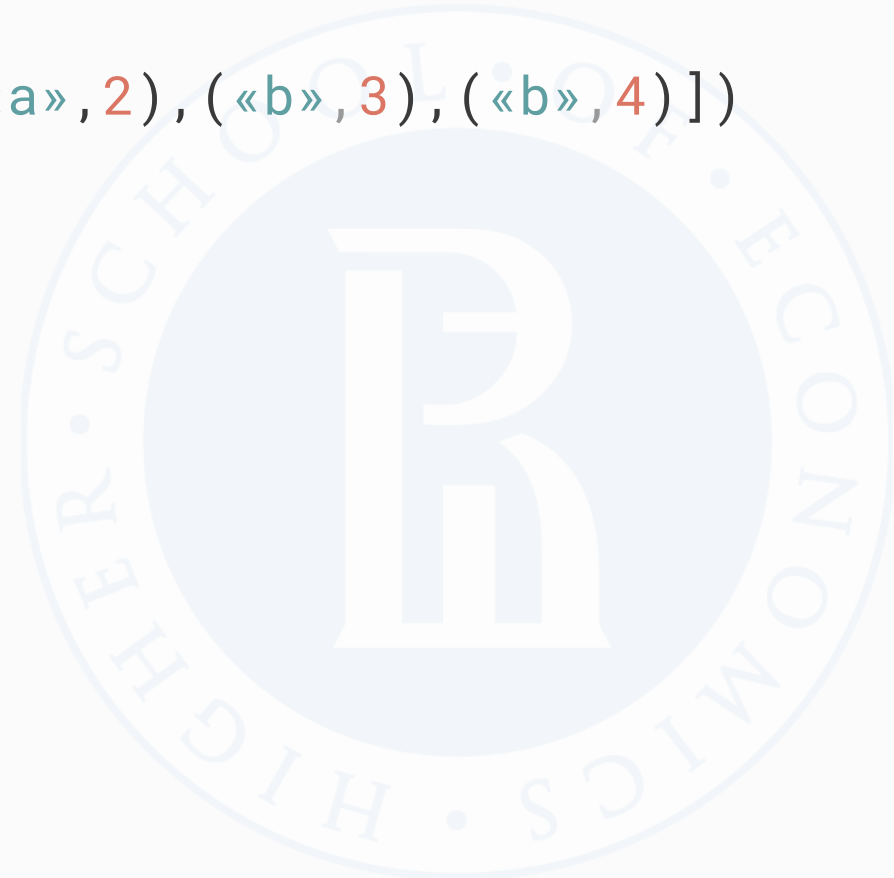
# Создание DataFrame

→ Создаем Spark SQL сессию:

```
sc = pyspark.SparkContext()  
se = SparkSession(sc)
```

→ Создаем RDD:

```
rdd =  
sc.parallelize([(«a», 1), («a», 2), («b», 3), («b», 4)])
```



# Создание DataFrame

➔ Создаем Spark SQL сессию:

```
sc = pyspark.SparkContext()  
se = SparkSession(sc)
```

➔ Создаем RDD:

```
rdd =  
sc.parallelize([(«a», 1), («a», 2), («b», 3), («b», 4)])
```

➔ Конвертируем в DataFrame (вывод типов через Py4J):

```
df = se.createDataFrame(rdd)  
df.printSchema()  
root  
|-- _1: string (nullable = true)  
|-- _2: long (nullable = true)
```



# Запросы к DataFrame

➔ Присваиваем DataFrame имя для запросов:

```
df.registerTempTable("table")
```

➔ Эквивалентные запросы (выполняются в JVM):

```
se.sql("select key from table where value > 1")
```

```
df.select("key").where("value > 1")
```

```
df.select(df.key).where(df.value > 1)
```

key	value
a	1
a	2
b	3
b	4



key
a
b
b

# DataFrame API

Плюсы:

➔ Запросы выполняются в JVM (быстрее, чем в Python)



# DataFrame API

Плюсы:

- ➔ Запросы выполняются в JVM (быстрее, чем в Python)
- ➔ Spark оптимизирует запросы (превращает запрос в оптимальный план выполнения)



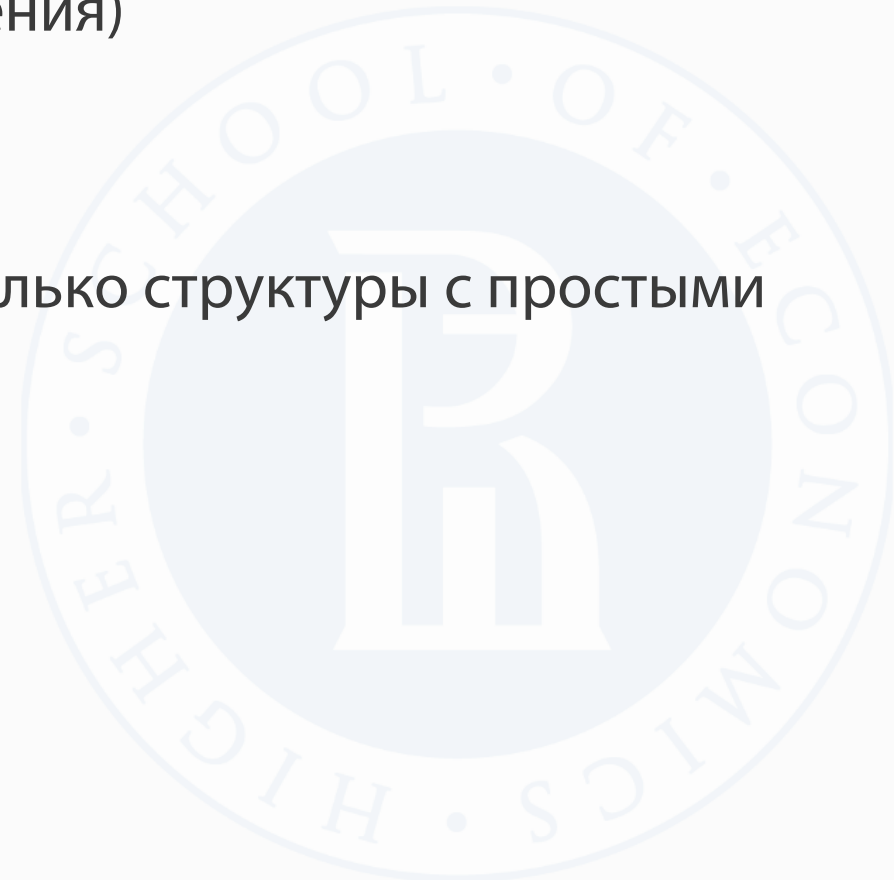
# DataFrame API

## Плюсы:

- ➔ Запросы выполняются в JVM (быстрее, чем в Python)
- ➔ Spark оптимизирует запросы (превращает запрос в оптимальный план выполнения)

## Минусы:

- ➔ В колонках можно хранить только структуры с простыми типами (int, str, float, ...)



# DataFrame API

## Плюсы:

- ➔ Запросы выполняются в JVM (быстрее, чем в Python)
- ➔ Spark оптимизирует запросы (превращает запрос в оптимальный план выполнения)

## Минусы:

- ➔ В колонках можно хранить только структуры с простыми типами (int, str, float, ...)
- ➔ Не все хорошо описывается SQL запросом (например, токенизация текста)

# Заключение



Spark описывает вычисления в виде графа (DAG) и может оптимизировать хранение промежуточных результатов



# Заключение






Spark описывает вычисления в виде графа (DAG) и может оптимизировать хранение промежуточных результатов



Программа на Spark — это набор операций над RDD



# Заключение

-  Spark описывает вычисления в виде графа (DAG) и может оптимизировать хранение промежуточных результатов
-  Программа на Spark — это набор операций над RDD
-  Вычисления в Spark ленивые, только действия над RDD приводят к запуску вычислений DAG





# Заключение

-  Spark описывает вычисления в виде графа (DAG) и может оптимизировать хранение промежуточных результатов
-  Программа на Spark — это набор операций над RDD
-  Вычисления в Spark ленивые, только действия над RDD приводят к запуску вычислений DAG
-  DataFrame API позволяет быстрее исполнять запросы к табличным данным, но не такое гибкое как RDD API