

Large Scale Machine Learning (LSML)

О курсе

Лекции

1. Введение в Apache Spark
2. Онлайн-обучение и линейные модели
3. Градиентный бустинг для больших данных
4. Рекомендательные системы
5. Трюки с хэшированием (LSH и не только)
6. Кластеризация больших данных
7. ETL пайплайны
8. Экосистема Hadoop



Практика

- Рекомендательная система на Apache Spark
- Упражнения после семинаров

Формула оценки

Результирующая оценка по дисциплине рассчитывается по формуле

$$\mathbf{0.5 \ ДЗ + 0.2 \ Проверочные + 0.3 \ Экзамен}$$

Предусмотрены автоматы (с 6-7 баллов)

Облачо от Azure

У каждого студента:

- Своя подписка в Azure (заполните форму!)
- 500 баксов на счету



Экзамен

- Письменный экзамен
- Задачи на листочке

Важное

- Репозиторий https://github.com/ZEMUSHKA/lsmI_hse
- Канал в телеграме

LSML #1

Введение в Apache Spark

Задача Word Count

- У нас есть огромный текстовый файл
- Хотим посчитать частоты слов в тексте

Как считать?

1. Разделим на кусочки
2. Обработаем каждый на отдельном ядре
3. Сложим счетчики слов

Парадигма Map-Reduce на примере Word Count

Шаг Map:

$(K_1, V_1) \rightarrow \text{List}(K_2, V_2)$

$(\# строки, "Deer Bear") \rightarrow [("Deer", 1), ("Bear", 1)]$

Шаг Shuffle (или Sort):

Shuffle делит данные по $\text{hash}(key) \% N$ на N частей

Sort сортирует данные по key и делит на N частей (по границам key)

Шаг Reduce:

$(K_1, (V_1, V_2, \dots)) \rightarrow \text{List}(K_3, V_3)$

$("Bear", (1, 1)) \rightarrow [("Bear", 2)]$

Hadoop

- **Hadoop** – проект Apache для распределенных вычислений
- **Hadoop MapReduce** – система для вычислений в парадигме Map-Reduce
- **Hadoop YARN** – планировщик задач и система для управления ресурсами кластера

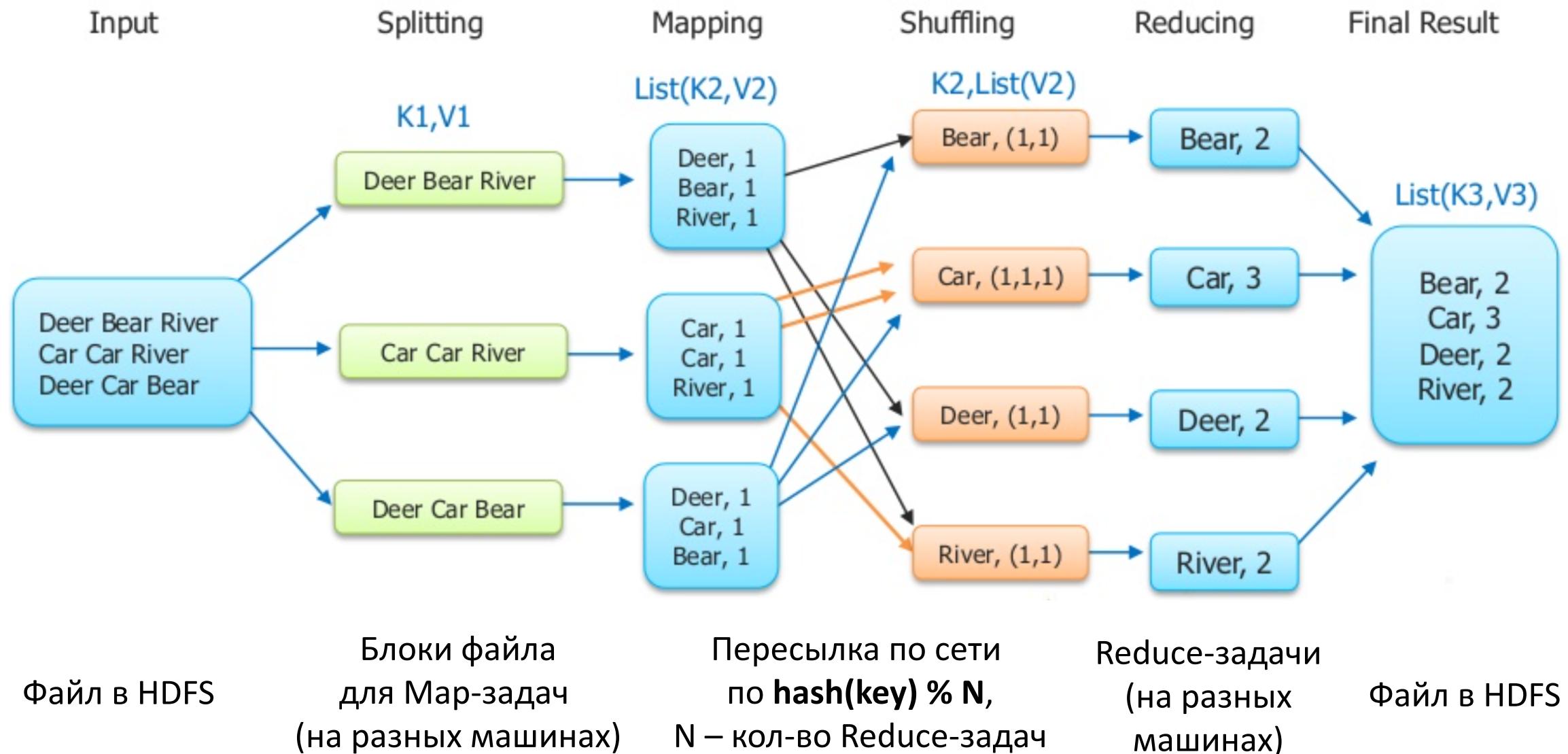
HDFS: распределенная файловая система Hadoop

- Файлы хранятся **блоками** на разных машинах
- Каждый **блок дублируется** на нескольких разных машинах (replication factor)
- Информация о соответствии **путь файла → его блоки** хранится на специальной машине **Name Node** в RAM

Зачем нужна HDFS?

- Чтобы MapReduce был эффективным

Парадигма Map-Reduce на примере Word Count



Apache Spark



- Фреймворк для выполнения распределенных задач
- API на нескольких языках: Scala, Java, Python
- Будем рассматривать на примере Python API – PySpark
- Включает полезные модули:
 - MLlib: алгоритмы машинного обучения
 - Spark SQL: выполнение SQL запросов на кластере
 - Алгоритмы на графах, потоковая обработка данных, ...

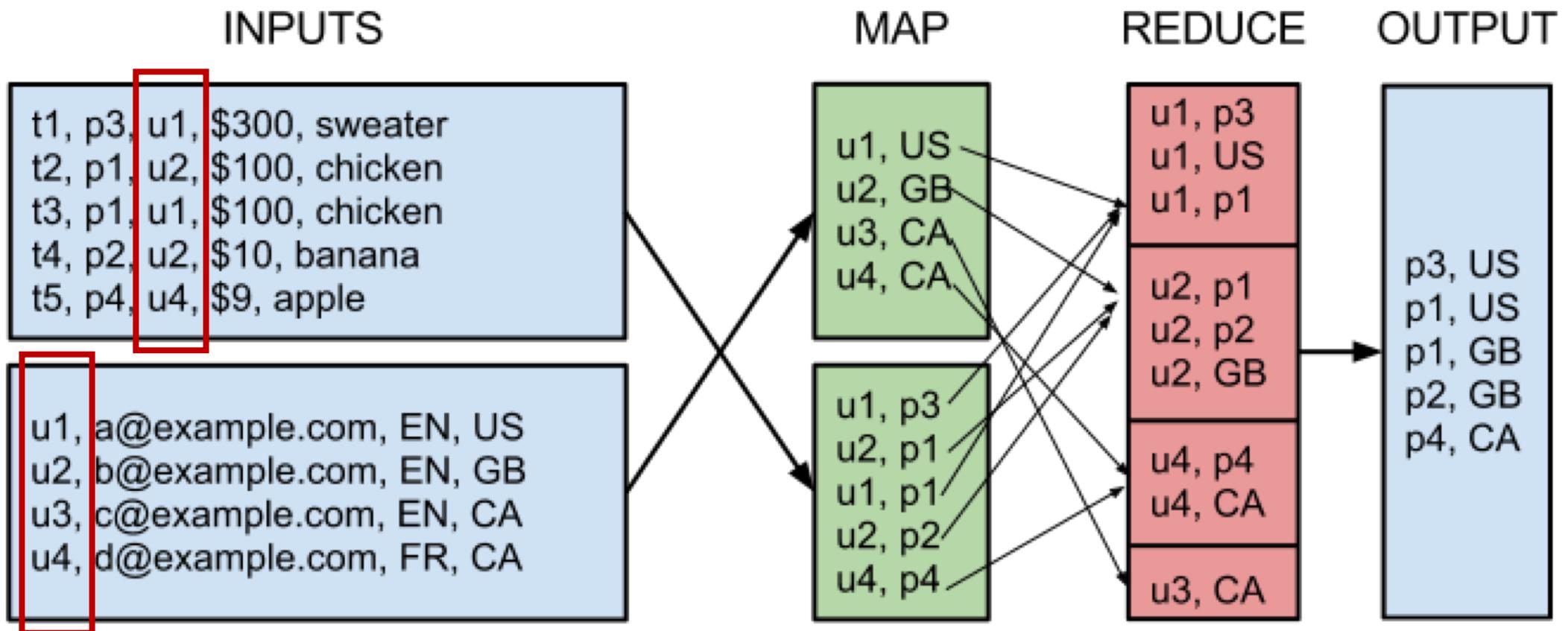
Join на SQL

- Таблица **a** – покупки пользователей (**user**, **product**, ...)
- Таблица **b** – информация о пользователях (**user**, **state**, ...)
- Хотим получить покупки продуктов по штатам
- Нужно сделать join по **user**

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

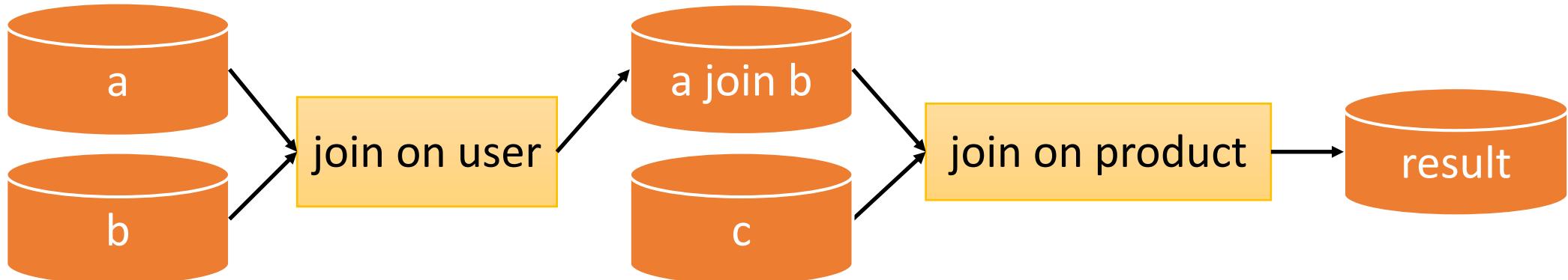
Join на MapReduce

- Этот же запрос на MapReduce:



MapReduce: еще один join

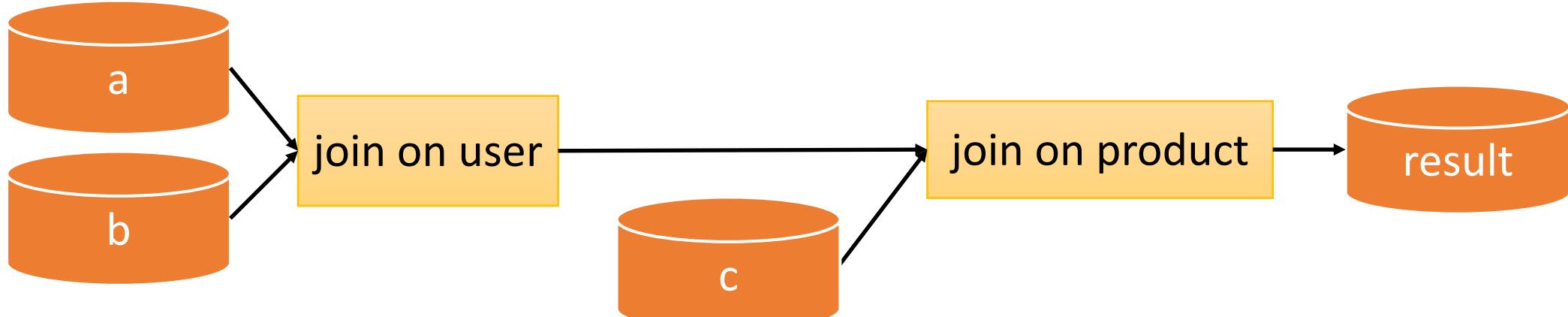
- В таблице **c** лежит информация о продуктах (**product**, **type**, ...)
- Решение на **Hadoop MapReduce**:



- Промежуточные результаты пишутся на диск и читаются с диска
- Это медленно и не всегда необходимо!

Spark: еще один join

- Решение на **Spark**:



- Результаты вычислений передаются по возможности в памяти **без сохранения на диск**
- Промежуточные результаты можно **закэшировать** в памяти и выполнять над ними несколько операций (если хватает суммарной памяти машин)
- Вычисления можно представить в виде графа (**DAG** – направленный граф без циклов)

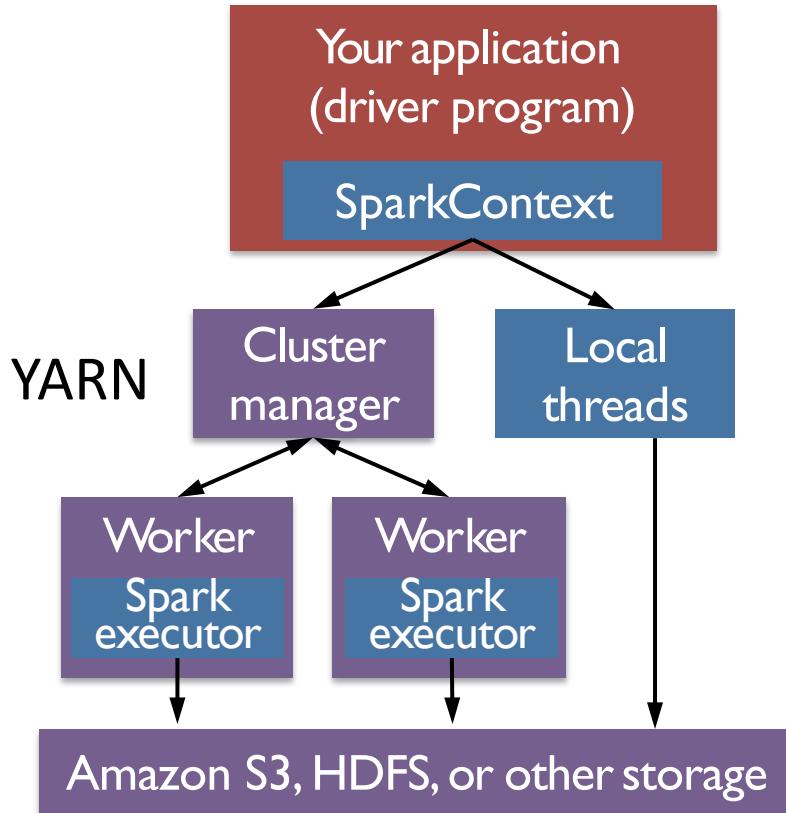
RDD в Spark

- **RDD** (resilient distributed dataset) – восстанавливаемый распределенный набор данных
 - Набор данных хранится распределенно на разных машинах
 - Потерянные части могут быть восстановлены (известна цепочка вычислений – DAG)
- Как сделать RDD?
 - Из файла (например, в HDFS)
 - **Распараллелив** Python коллекцию (список, итератор, ...)
 - **Трансформацией** из другого RDD

Операции над RDD

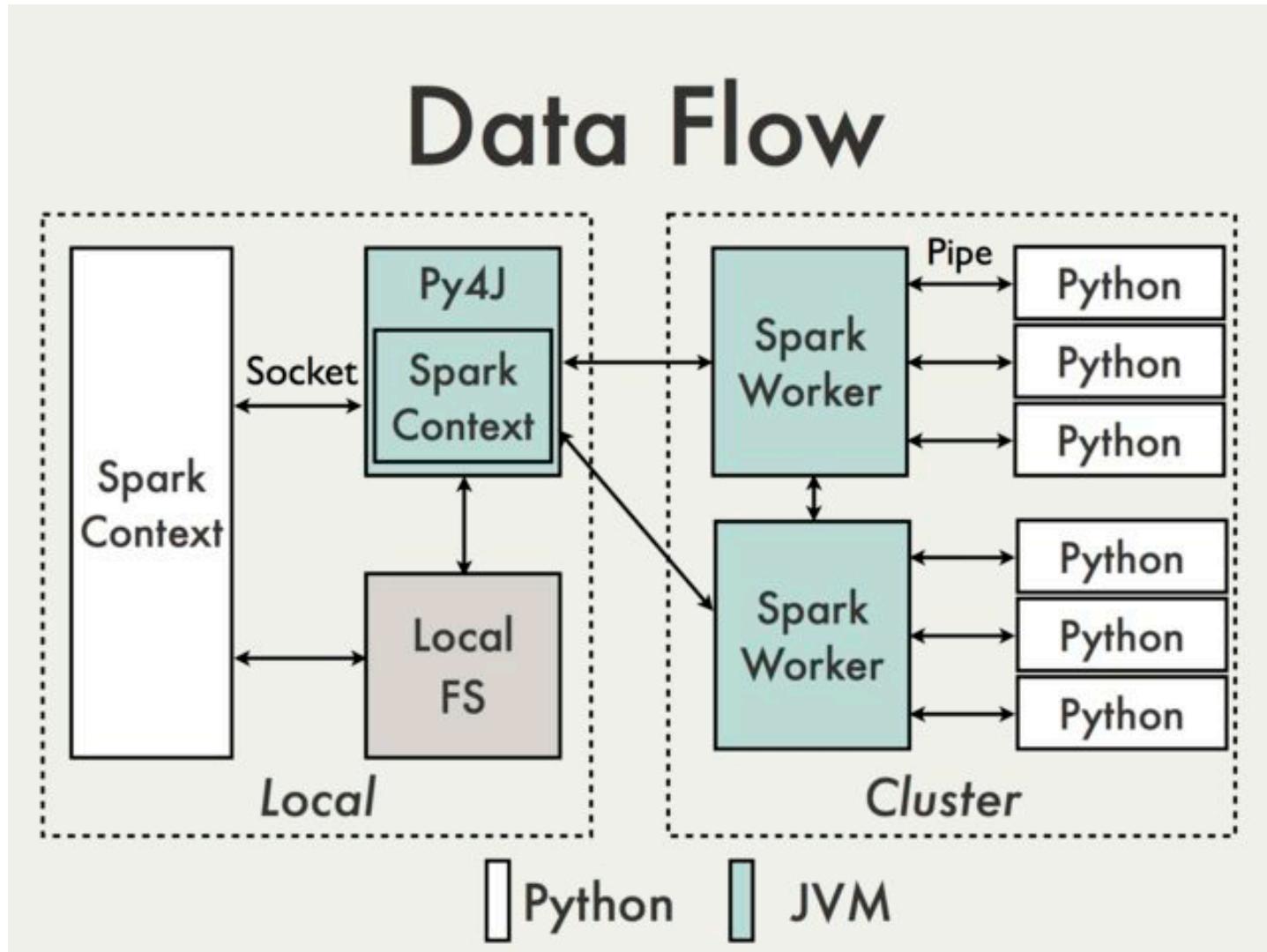
- Программа на Spark пишется в терминах операций над RDD
- **Трансформации:**
 - RDD → RDD
 - Ленивые – не вычисляются сразу
 - Примеры: `map`, `reduceByKey`, `join`
- **Действия:**
 - Приводят к вычислению RDD
 - Примеры: `saveAsTextFile`, `collect`, `count`
- **Другие операции:**
 - `persist`, `cache` – помечают RDD для сохранения в памяти/на диске при первом их вычислении

Как устроена программа на Spark



- Программа состоит из **driver** и **workers**.
- **Driver** хранит цепочку вычислений – DAG
- **Workers** запускаются на машинах кластера или как локальные процессы на одной машине с driver
- RDD распределены по **workers**
- **Driver** – это обычная программа на Python, в которой создается **SparkContext (sc)** – объект для работы со Spark

Особенности PySpark



- Все Python объекты сериализуются при помощи cPickle:
 - Любое действие с Python объектом требует десериализации в Python
- При работе с PySpark данные хранятся дважды:
 - Сериализованный RDD в JVM у Spark Worker
 - Десериализованные данные в памяти Python

Пример трансформации

```
rdd = (sc
        .parallelize([1, 2, 3, 4])
        .map(lambda x: x * 2))
print rdd
print rdd.collect()
```

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

Пример трансформации

```
rdd = (sc
        .parallelize([1, 2, 3, 4])
        .flatMap(lambda x: [x, x * 2]))
print rdd
print rdd.collect()
```

```
PythonRDD[19] at RDD at PythonRDD.scala:48
[1, 2, 2, 4, 3, 6, 4, 8]
```

Пример действия

```
rdd = (sc
        .parallelize(np.random.random((1000,))))
        .flatMap(lambda x: [x, x * 2]))
print rdd
print rdd.takeOrdered(2, lambda x: -x)
```

```
PythonRDD[29] at RDD at PythonRDD.scala:48
[1.9987386963918603, 1.997388155520317]
```

MapReduce как две операции в Spark

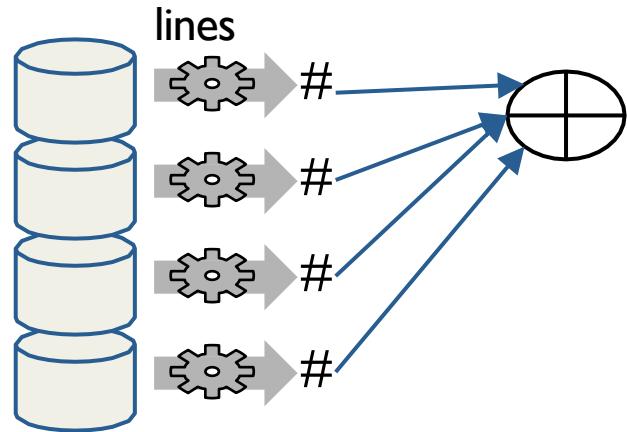
```
rdd = (  
    sc  
.parallelize(["this is text", "text too"])  
.flatMap(lambda x: [(w, 1) for w in x.split()])  
.reduceByKey(lambda a, b: a + b))  
  
print rdd  
print rdd.collect()
```

```
PythonRDD[61] at RDD at PythonRDD.scala:48  
[( 'text', 2 ), ( 'too', 1 ), ( 'is', 1 ), ( 'this', 1 )]
```

Кэширование в RAM

```
lines = sc.textFile("...", 4)
```

```
print lines.count()
```

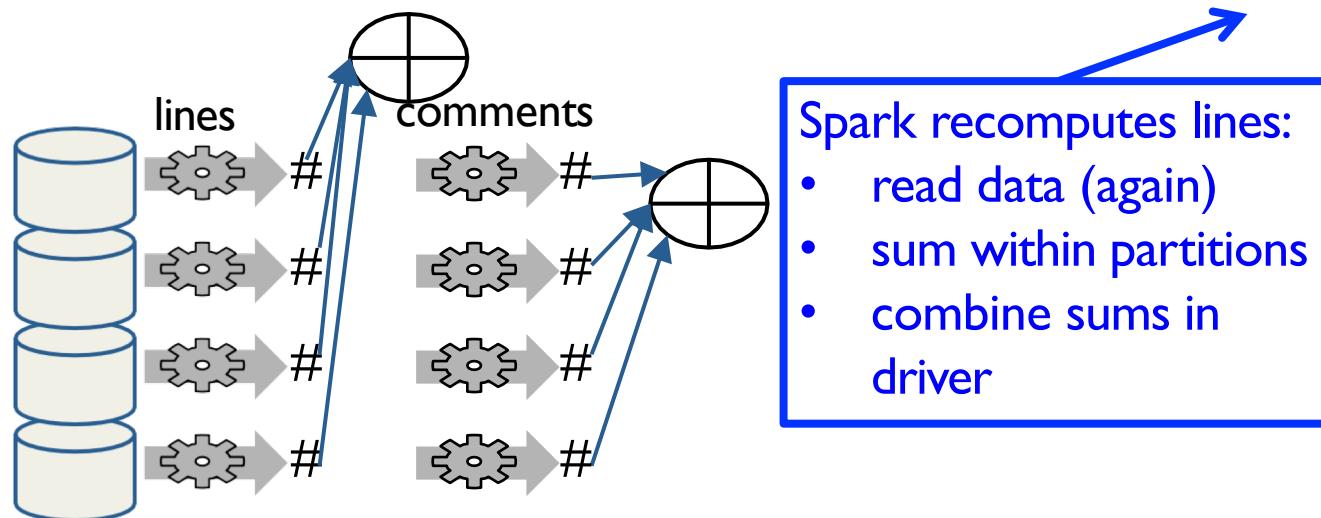


count() causes Spark to:

- read data
- sum within partitions
- combine sums in driver

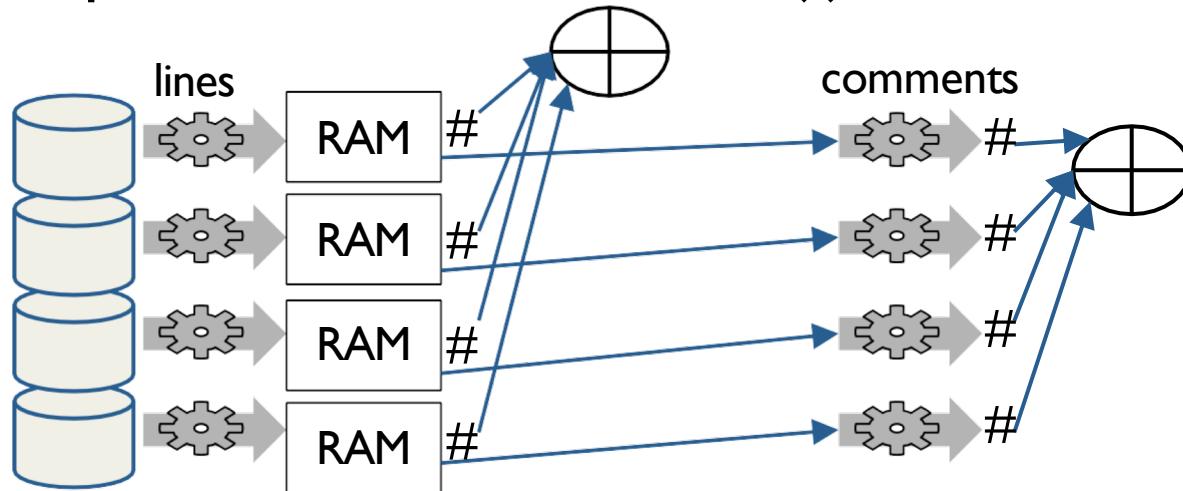
Кэширование в RAM

```
lines = sc.textFile("...", 4)
comments = lines.filter(isComment)
print lines.count(), comments.count()
```



Кэширование в RAM

```
lines = sc.textFile("...", 4)
Lines.cache() # save, don't recompute!
comments = lines.filter(isComment)
print lines.count(), comments.count()
```



Ссылки

- <http://spark.apache.org/docs/latest/programming-guide.html>
- <http://spark.apache.org/docs/latest/api/python/index.html>
- <http://spark.apache.org/docs/latest/sql-programming-guide.html>
- <https://0x0fff.com/wp-content/uploads/2015/11/Spark-Architecture-JD-Kiev-v04.pdf>
- <https://spark-summit.org/2014/wp-content/uploads/2014/07/A-Deeper-Understanding-of-Spark-Internals-Aaron-Davidson.pdf>