Apache Spark Engine

- *Lightning-fast cluster computing!*
- Разработан в UC Berkeley’s AMPLab в 2009 году
- Проект Apache верхнего уровня

  http://spark.apache.org/

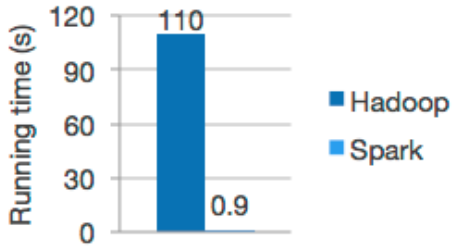
Apache Spark

Скорость

– Работает быстрее чем Hadoop MapReduce в 100 раз

Легкость использования

– Просто писать приложения на Java, Scala и Python



Logistic regression in Hadoop and Spark

```
file = spark.textFile("hdfs://...")

file.flatMap(lambda line: line.split())
    .map(lambda word: (word, 1))
    .reduceByKey(lambda a, b: a+b)
```

Word count in Spark's Python API

Apache Spark

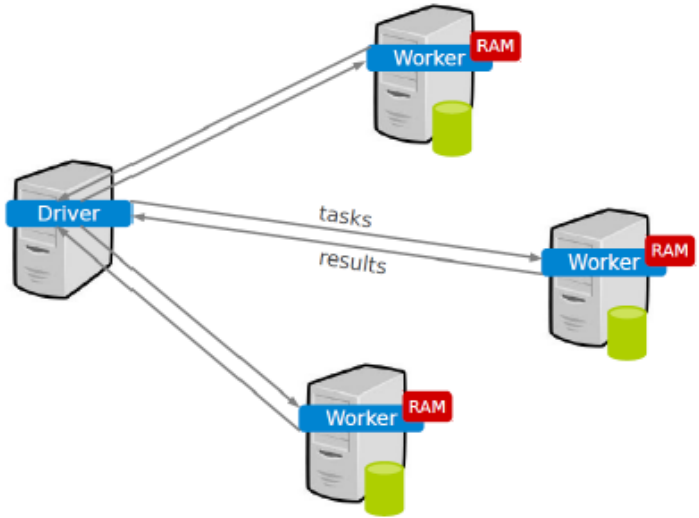
Обобщенность

- Комбинирование SQL, streaming и комплексной аналитики в рамках одного приложения
- Spark SQL, Mlib, GraphX и Spark Streaming

Работает везде

- Hadoop, Mesos, standalone или в облаке
- Доступ к данных из различных источников (HDFS, Cassandra, HBase, S3)

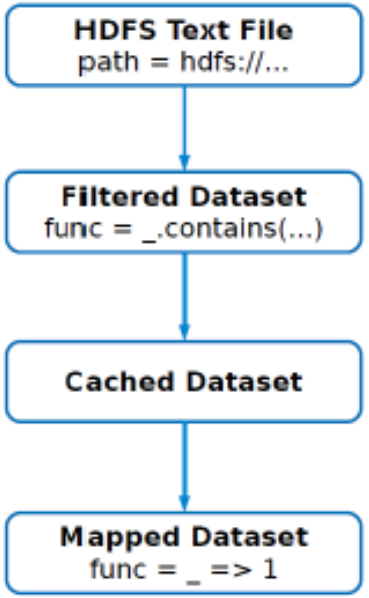
Программный интерфейс Spark



Lineage

- **Lineage:** это transformations, используемые для построения RDD
- **RDD** сохраняются как цепочка объектов, охватывающих весь **lineage** каждого RDD

```
val file = sc.textFile("hdfs://...")
val mail = file.filter(_.contains("MAIL"))
val cached = mail.cache()
val ones = cached.map(_ => 1)
val count = ones.reduce(_+_)
```



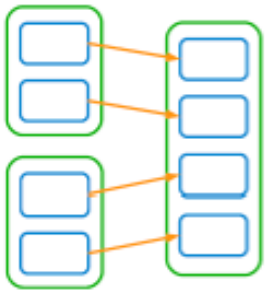
Dependencies RDD

- Два типа зависимостей между RDD
  - **Narrow**
  - **Wide**

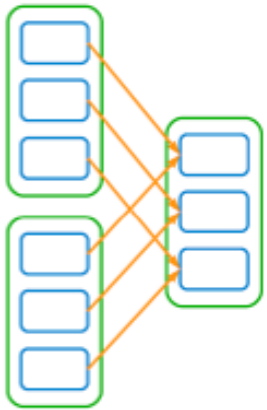
Dependencies RDD: **Narrow**

- **Narrow:** каждая партиция родительского RDD используется максимум в одной дочерней партиции RDD
- *Narrow dependencies* позволяют выполнять **pipelined execution** на одной ноде кластера:
  - Например, фильтр следуемый за Map

map, filter



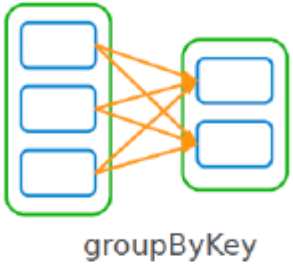
union

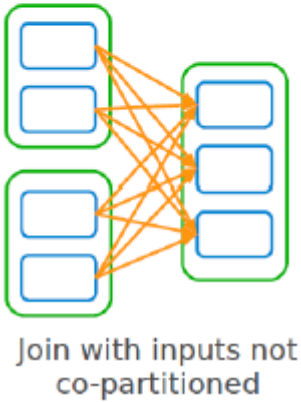


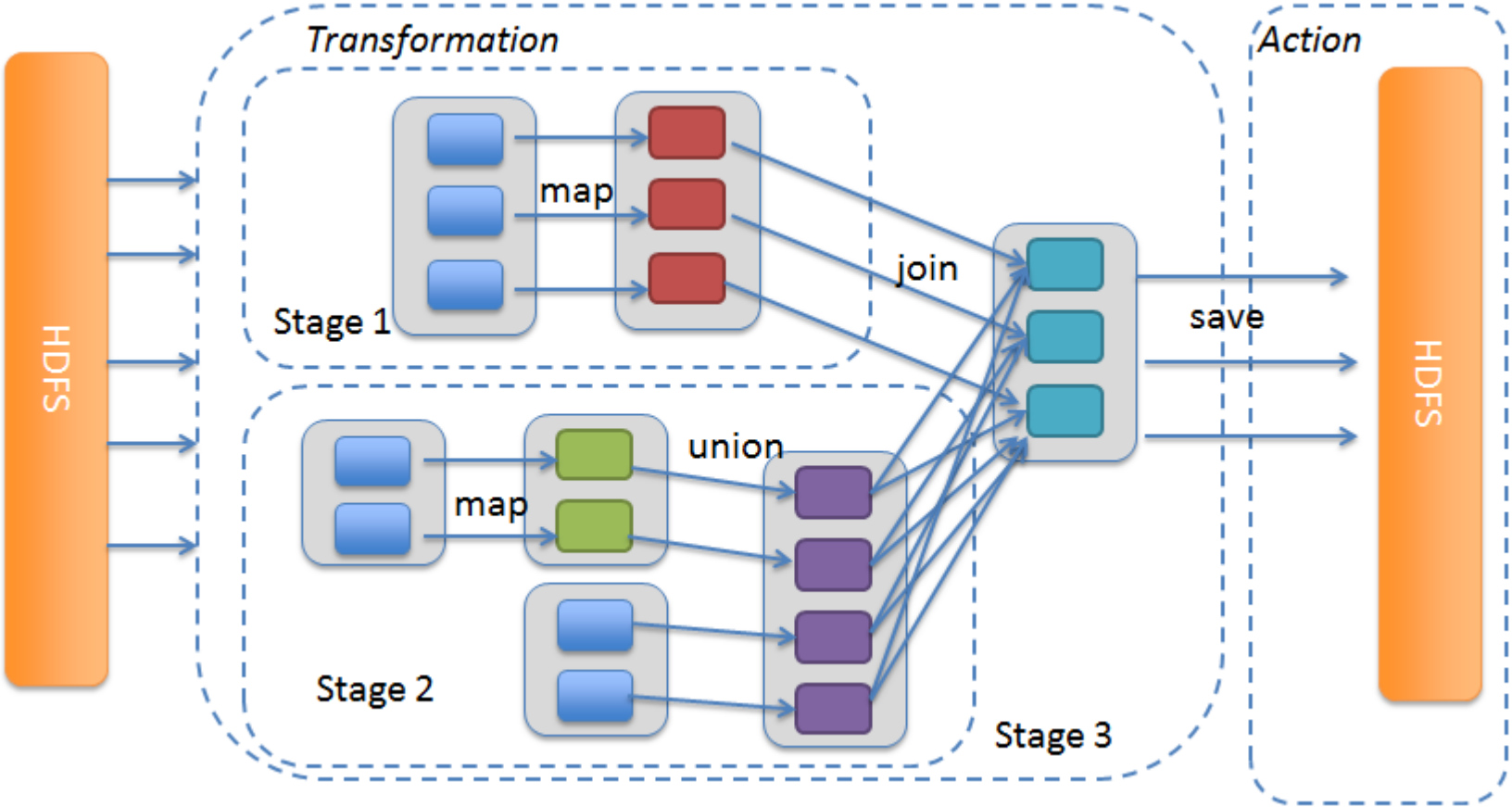
join with inputs
co-partitioned

Dependencies RDD: **Wide**

- **Wide:** каждая партиция родительского RDD используется во множестве дочерних партиций RDD






Transformation
Action
HDFS
map
Stage 1
join
save
HDFS
union
map
Stage 2
Stage 3

RDD Fault Tolerance

- RDD поддерживает информацию о ***lineage***, которая может быть использована для **восстановления** потерянных партиций
- **Отсутствие репликации**
- Пересчет только **потерянных партиций** RDD
- Промежуточные результаты из *wide dependencies* по возможности сохраняются

RDD Fault Tolerance

- Восстановление может быть **затратным по времени** для RDD с длинными цепочками *lineage* и wide dependencies
- Может быть полезным сохранять состояния некоторых RDD в надежное хранилище

Memory Management

- Если недостаточно памяти для ***новых*** партиций RDD, то будет использоваться механизм вытеснения ***LRU*** (*least recently used*)
- Spark предоставляет 3 опции для хранения *RDD*
  - В ***memory storage*** в виде ***deserialized Java objects***
  - В ***memory storage*** в виде ***serialized Java objects***
  - На ***disk storage***
- Функция *cache()* явным образом указывает, что RDD нужно хранить в памяти

Примеры: Text Search (Scala)

```
val file = spark.textFile("hdfs://...")
val errors = file.filter(line => line.contains("ERROR"))

// Count all the errors
 errors.count()

// Count errors mentioning MySQL
 errors.filter(line => line.contains("MySQL")).count()

// Fetch the MySQL errors as an array of strings
 errors.filter(line => line.contains("MySQL")).collect()
```

Примеры: Text Search (Python)

```
file = spark.textFile("hdfs://...")
errors = file.filter(lambda line: "ERROR" in line)

# Count all the errors
errors.count()

# Count errors mentioning MySQL
errors.filter(lambda line: "MySQL" in line).count()

# Fetch the MySQL errors as an array of strings
 errors.filter(lambda line: "MySQL" in line).collect()
```

```
JavaRDD<String> file = spark.textFile("hdfs://...");
JavaRDD<String> errors = file.filter(new Function<String, Boolean>() {
  public Boolean call(String s) { return s.contains("ERROR"); }
 });

errors.count();

errors.filter(new Function<String, Boolean>() {
  public Boolean call(String s) { return s.contains("MySQL"); }
}).count();

errors.filter(new Function<String, Boolean>() {
  public Boolean call(String s) { return s.contains("MySQL"); }
}).collect();
```

Примеры: Word Count (Scala)

```
val file = spark.textFile("hdfs://...")
val counts = file.flatMap(line => line.split(" "))
                 .map(word => (word, 1))
                 .reduceByKey(_ + _)
 counts.saveAsTextFile("hdfs://...")
```

Примеры: Word Count (Python)

```
file = spark.textFile("hdfs://...")
counts = file.flatMap(lambda line: line.split(" ")) \
              .map(lambda word: (word, 1)) \
              .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://...")
```

Примеры: Word Count (Java)

```
JavaRDD<String> file = spark.textFile("hdfs://...");
JavaRDD<String> words = file.flatMap(new FlatMapFunction<String, String>() {
  public Iterable<String> call(String s) { return Arrays.asList(s.split(" ")); }
});

JavaPairRDD<String, Integer> pairs = words.mapToPair(new PairFunction<String, String, Integer>() {
  public Tuple2<String, Integer> call(String s) { return new Tuple2<String, Integer>(s, 1); }
});

JavaPairRDD<String, Integer> counts = pairs.reduceByKey(new Function2<Integer, Integer>() {
  public Integer call(Integer a, Integer b) { return a + b; }
});
counts.saveAsTextFile("hdfs://...");
```