

# Spark. RDD / SQL

SELEZNEV ARTEM  
HEAD OF CVM ANALYTICS @ MAGNIT

# APACHE SPARK



# ФУНКЦИОНАЛЬНАЯ РАЗНИЦА



map()

reduce()

# ФУНКЦИОНАЛЬНАЯ РАЗНИЦА



`map()`

`filter()`

`join()`

`first()`

`reduce()`

`sortBy()`

`groupByKey()`

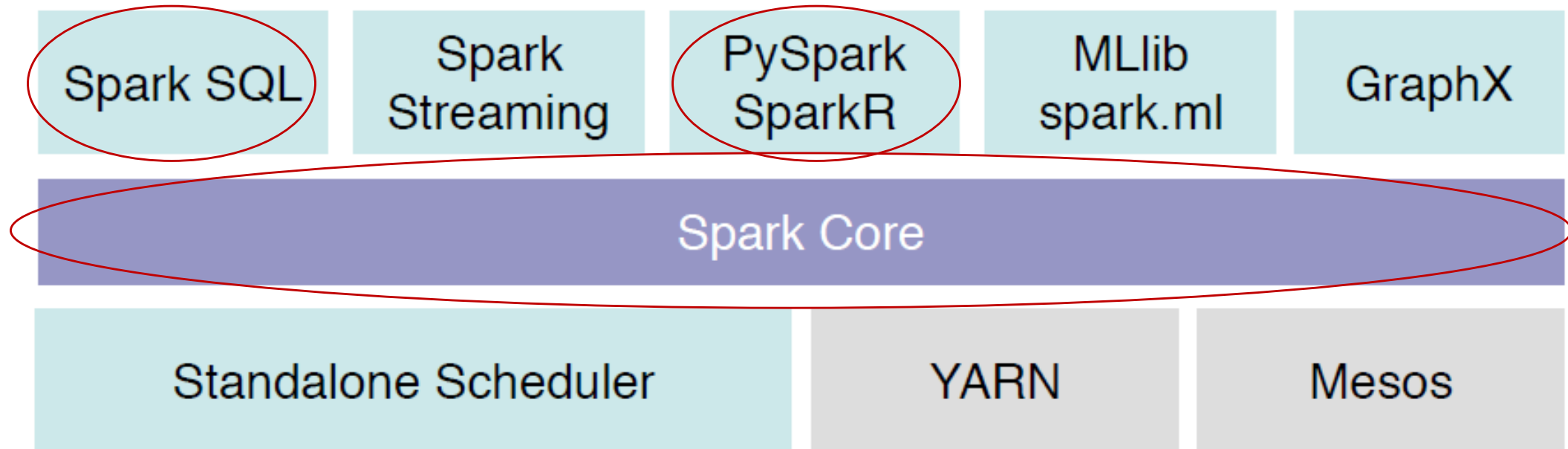
`count()`



`map()`

`reduce()`

# APACHE SPARK



# APACHE SPARK – ЗАВИСИМОСТИ

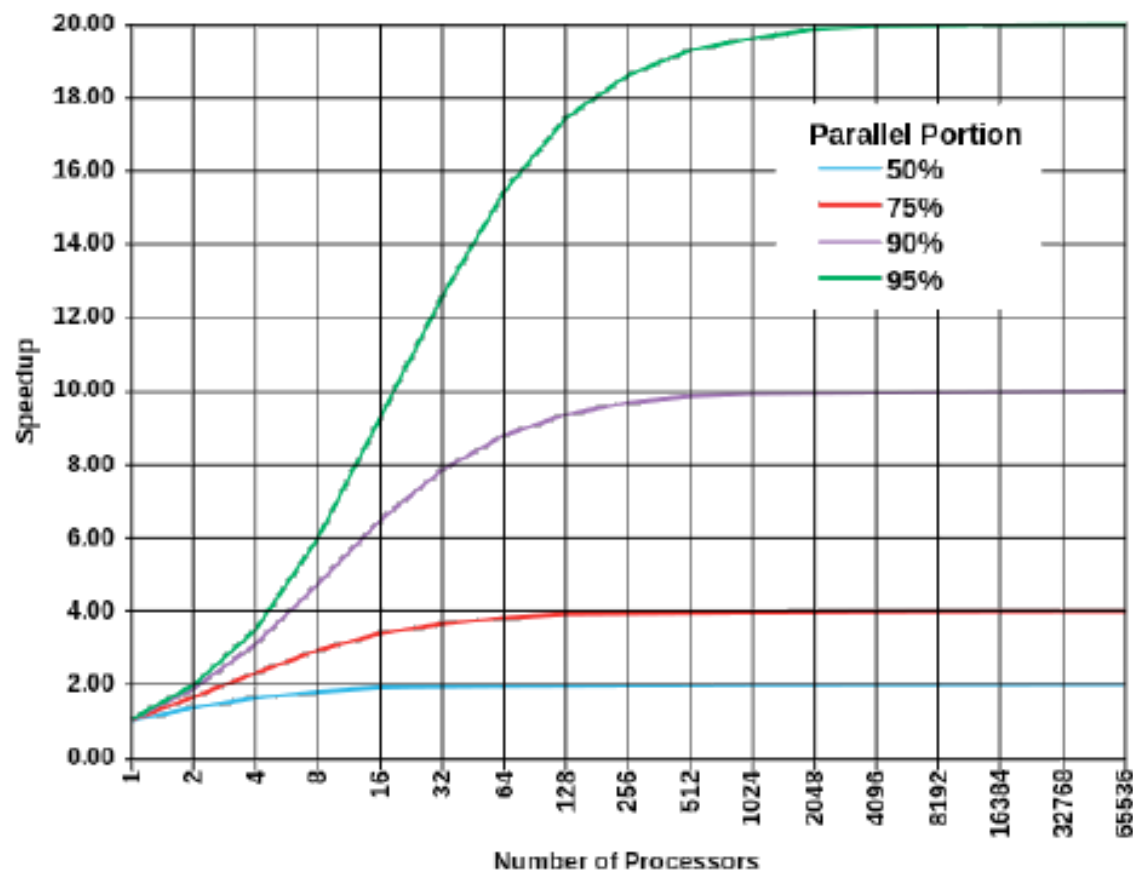
РЕСУРСЫ

ПАРТИЦИОНИРОВАНИЕ

# APACHE SPARK RUN-RESOURCES



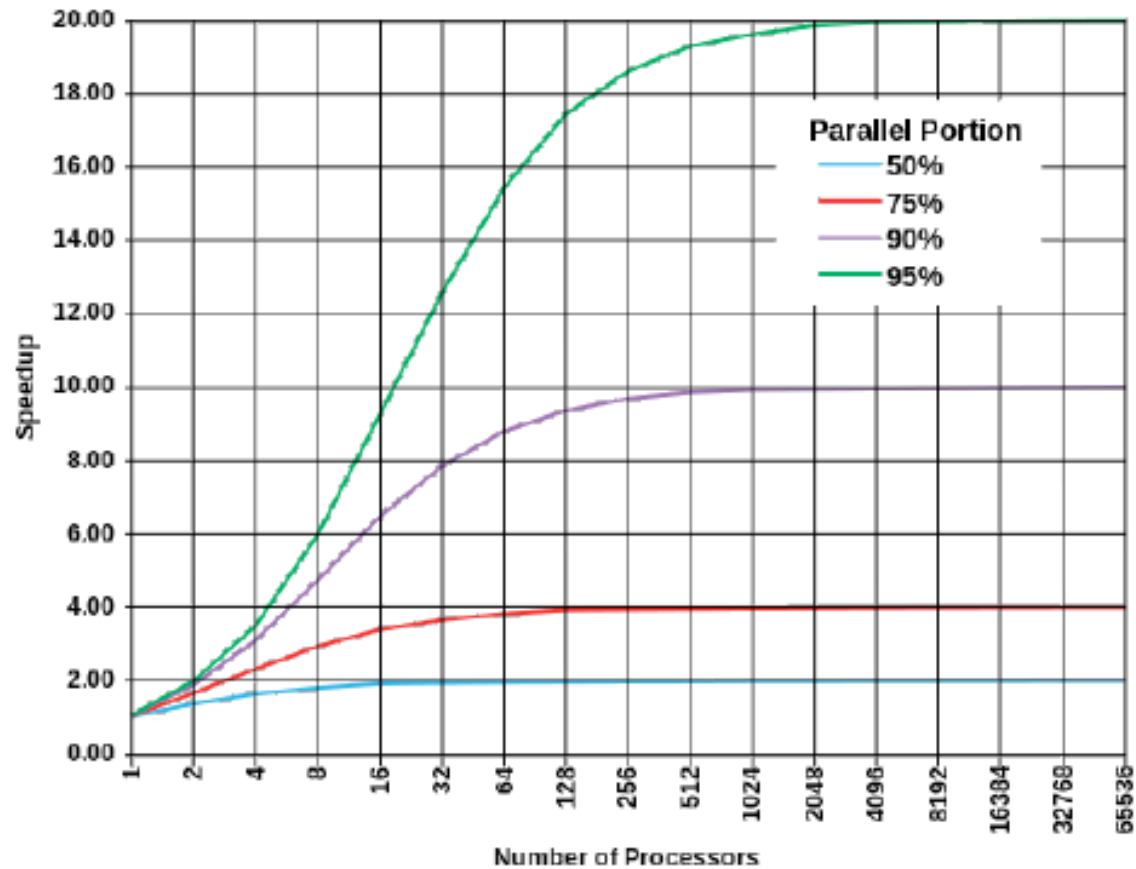
# ЗАКОН АМДАЛА



- Есть ограничение в приросте производительности
- Все будет работать не быстрее самого медленно



# ЗАКОН АМДАЛА



$$Speedup = \frac{1}{(1 - F) + \frac{F}{S}}$$

# ЗАКОН АМДАЛА

$$Speedup = \frac{1}{(1 - F) + \frac{F}{S}}$$

То, что может быть  
распараллелено (%)

То, что НЕ может быть  
распараллелено (%)

Кол-во процессоров

# ЗАКОН АМДАЛА

$$Speedup = \frac{1}{(1 - 0.25) + \frac{0.25}{20}}$$

То, что может быть  
распараллелено (%)

То, что НЕ может быть  
распараллелено (%)

Кол-во процессоров

20 - процессоров  
25% - распараллелено

# ЗАКОН АМДАЛА

$$Speedup = \frac{1}{0.75 + \frac{0.25}{20}} = 1.31$$

То, что может быть  
распараллелено (%)

То, что НЕ может быть  
распараллелено (%)

Кол-во процессоров

20 - процессоров  
25% - распараллелено  
1.31 – быстрее процесс

# APACHE SPARK PARTITIONS



# APACHE SPARK

## (2 СТРУКТУРЫ ДЛЯ PYTHON)

	RDD	DataFrame
Immutability	✓	✓
Schéma	✗	✓
Apache Spark 1	✓	✓
Apache Spark 2	✓	✓ (it does not exist in Java anymore)
Performance optimization	✗	✓
Level	Low	High (built upon RDD)
Typed	✓	✗
Syntax Error	Compile time	Compile time
Analysis Error	Compile time	Runtime

# APACHE SPARK – ТЕРМИНЫ

DRIVER

WORKER

# APACHE SPARK – ТЕРМИНЫ

DRIVER

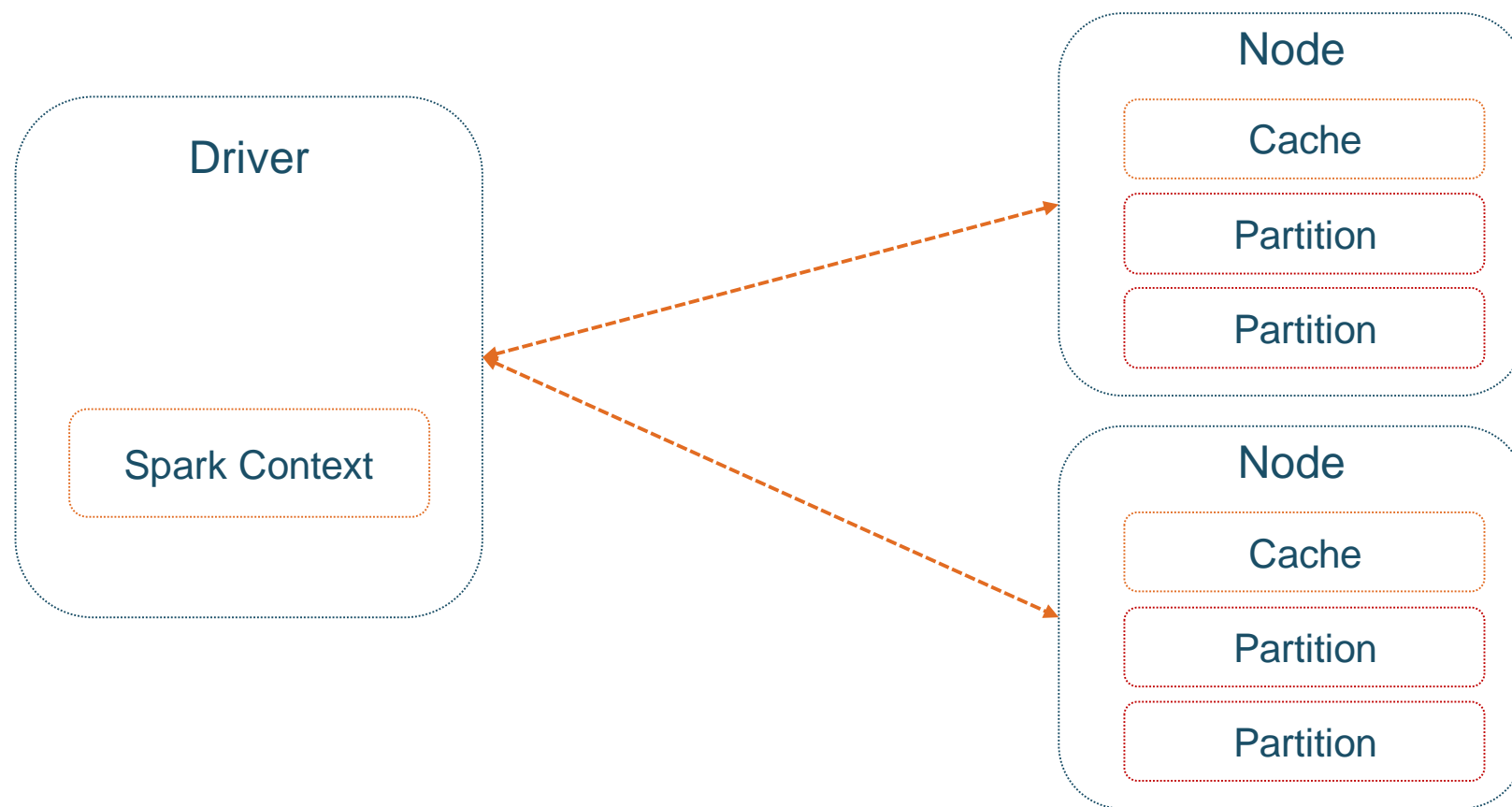
Процесс содержащий Spark Context

WORKER

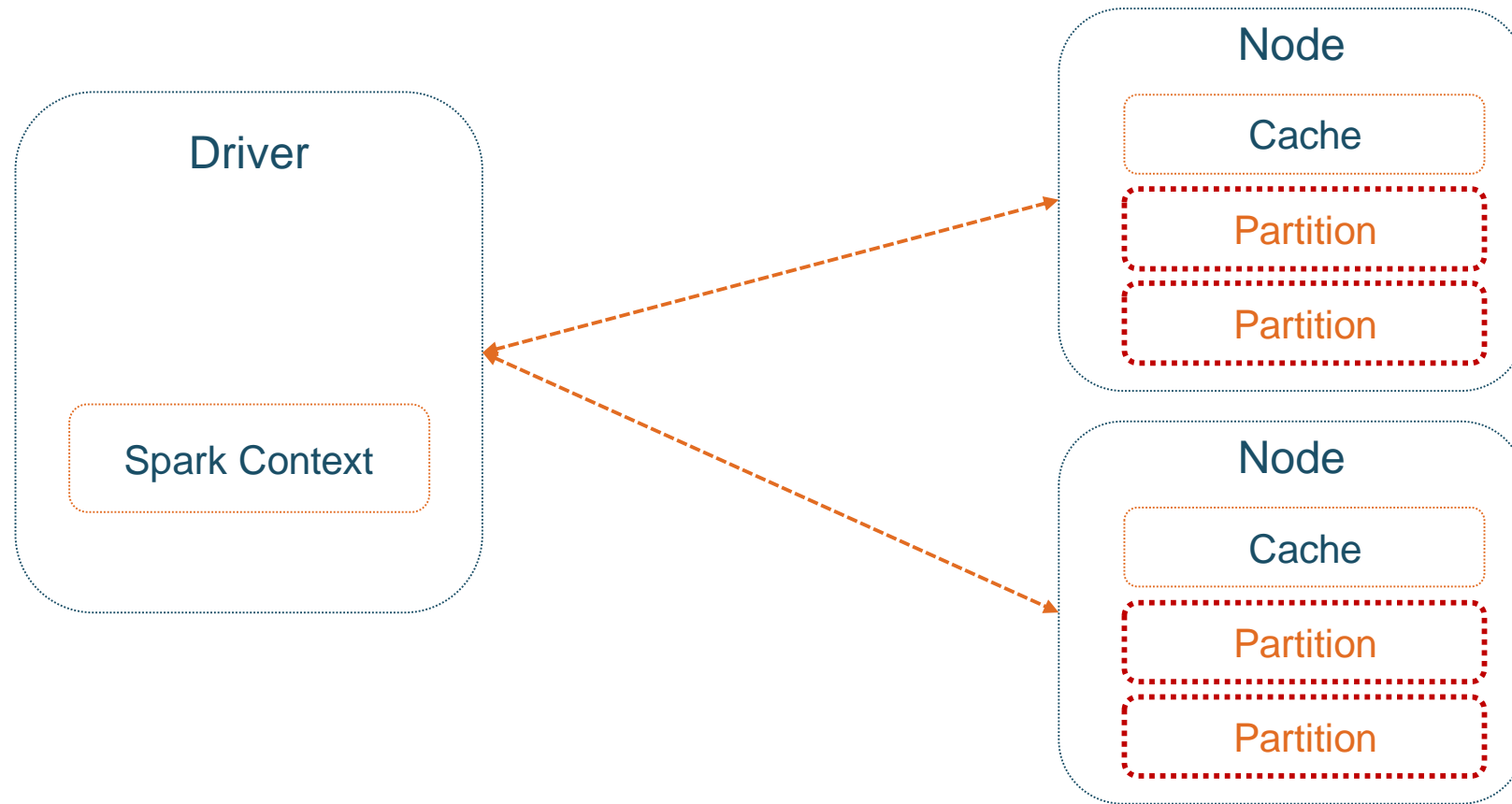
Приложение (node) выполняющее код/команды



# РАБОТА КОНТЕКСТА



# РАБОТА КОНТЕКСТА (?)



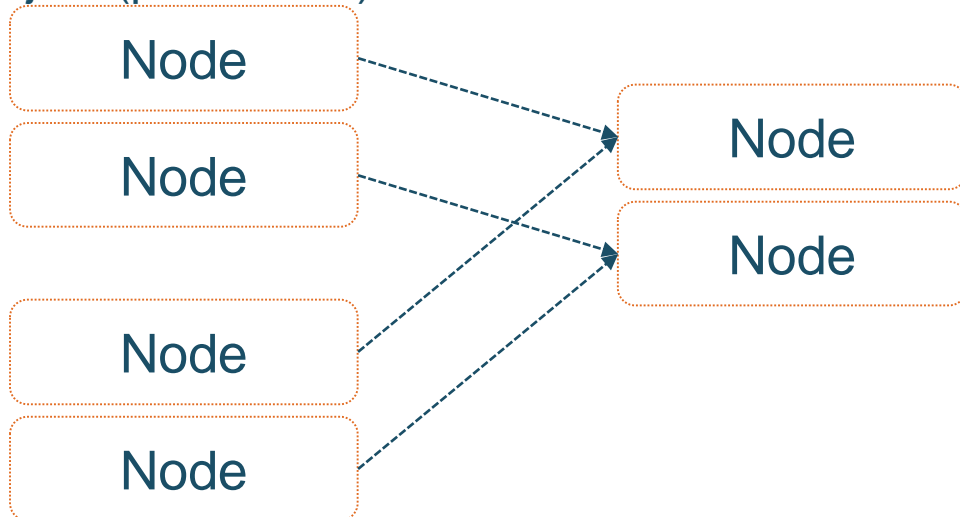
# ЗАВИСИМОСТЬ ОТ ПАРТИЦИЙ

## (НЕ БОЛЬШАЯ)

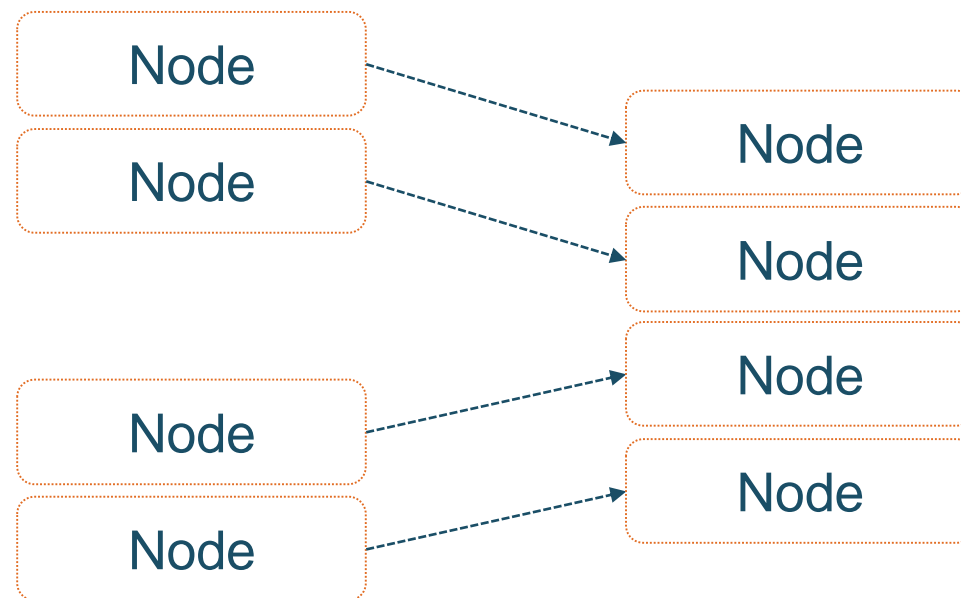
map, filter



join (partitioned)



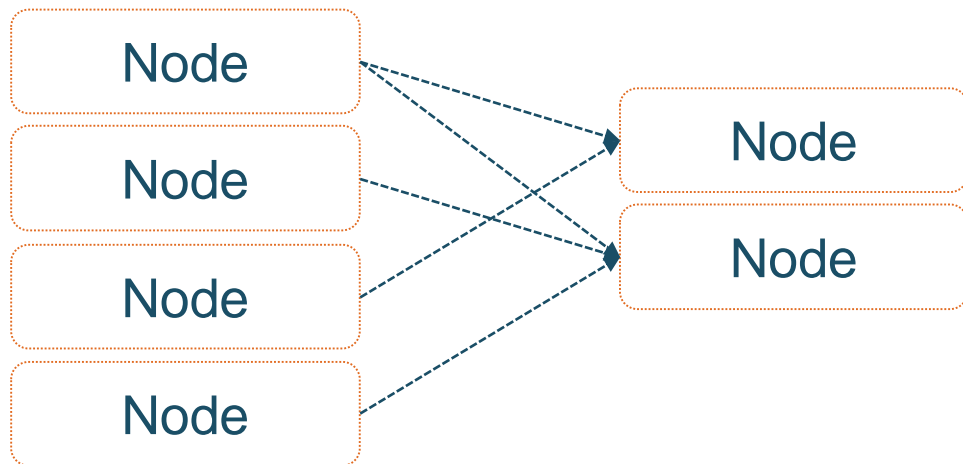
union



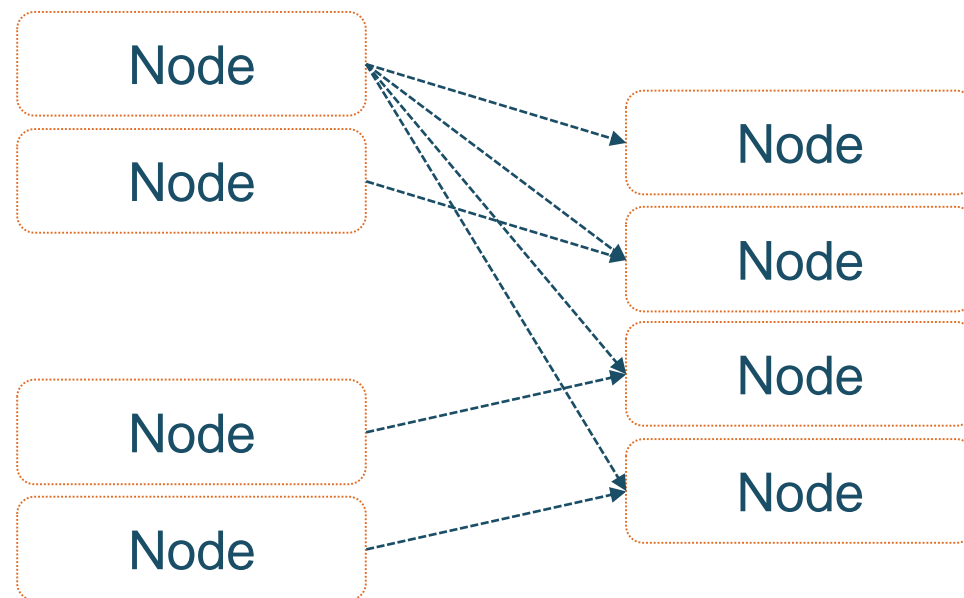
# ЗАВИСИМОСТЬ ОТ ПАРТИЦИЙ

## (БОЛЬШАЯ)

groupByKey



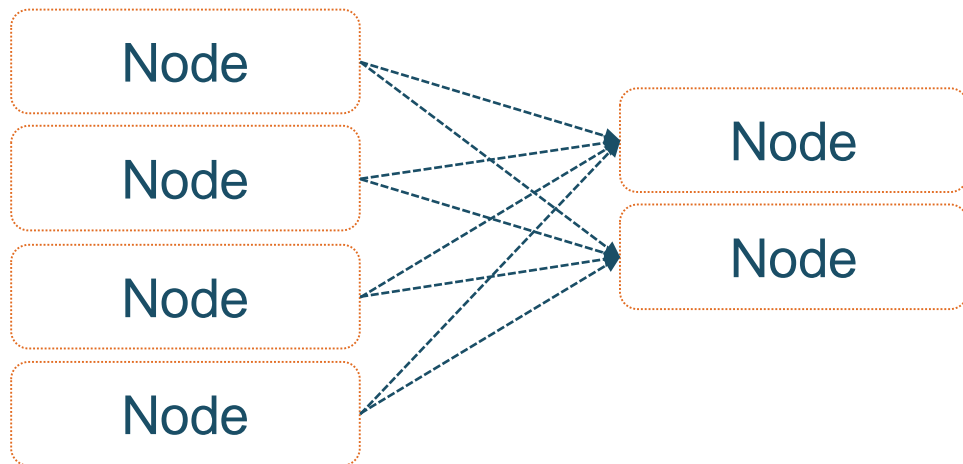
join (not partitioned)



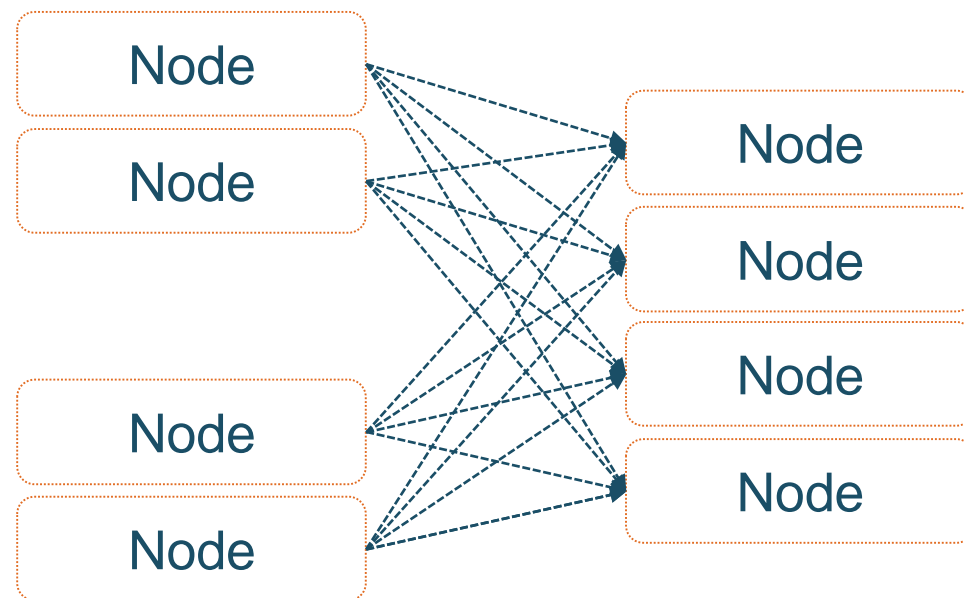
# ЗАВИСИМОСТЬ ОТ ПАРТИЦИЙ

## (БОЛЬШАЯ)

groupByKey

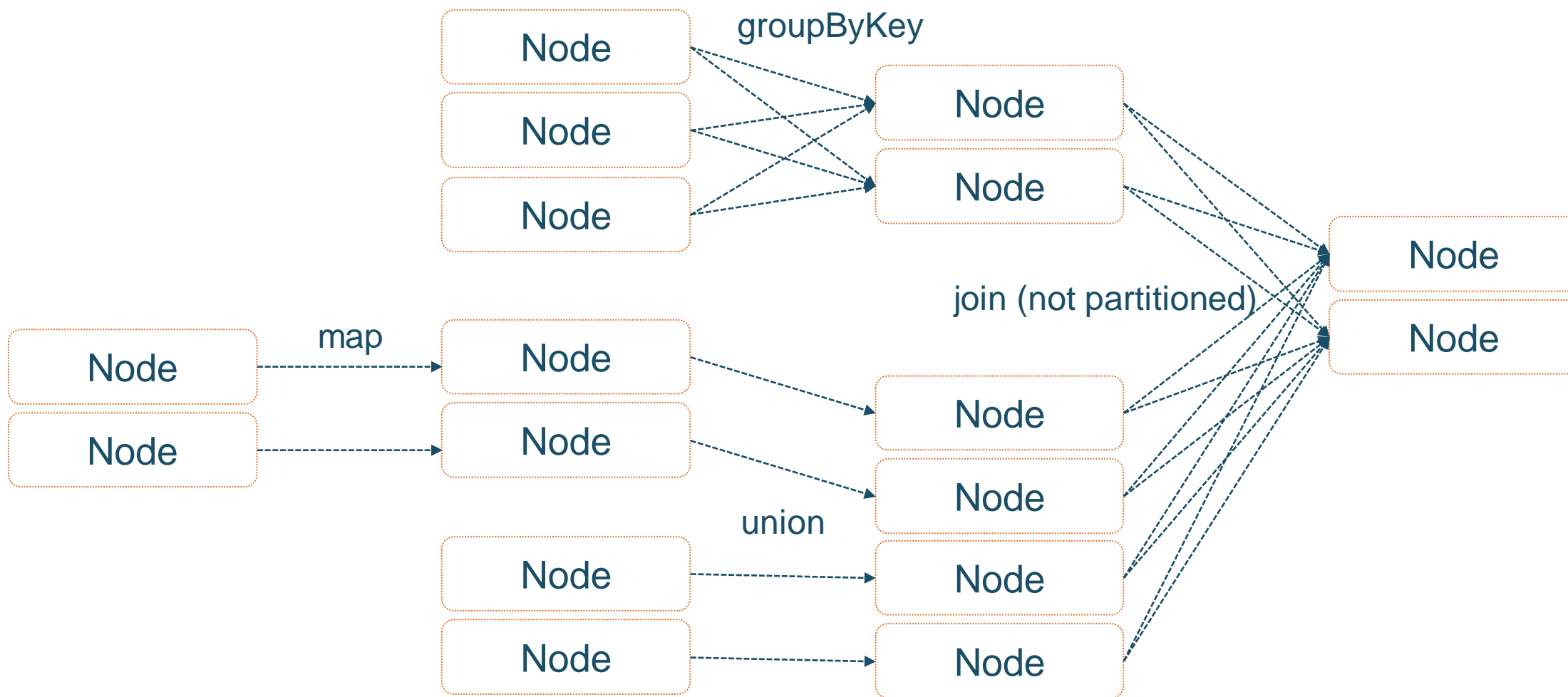


join (not partitioned)



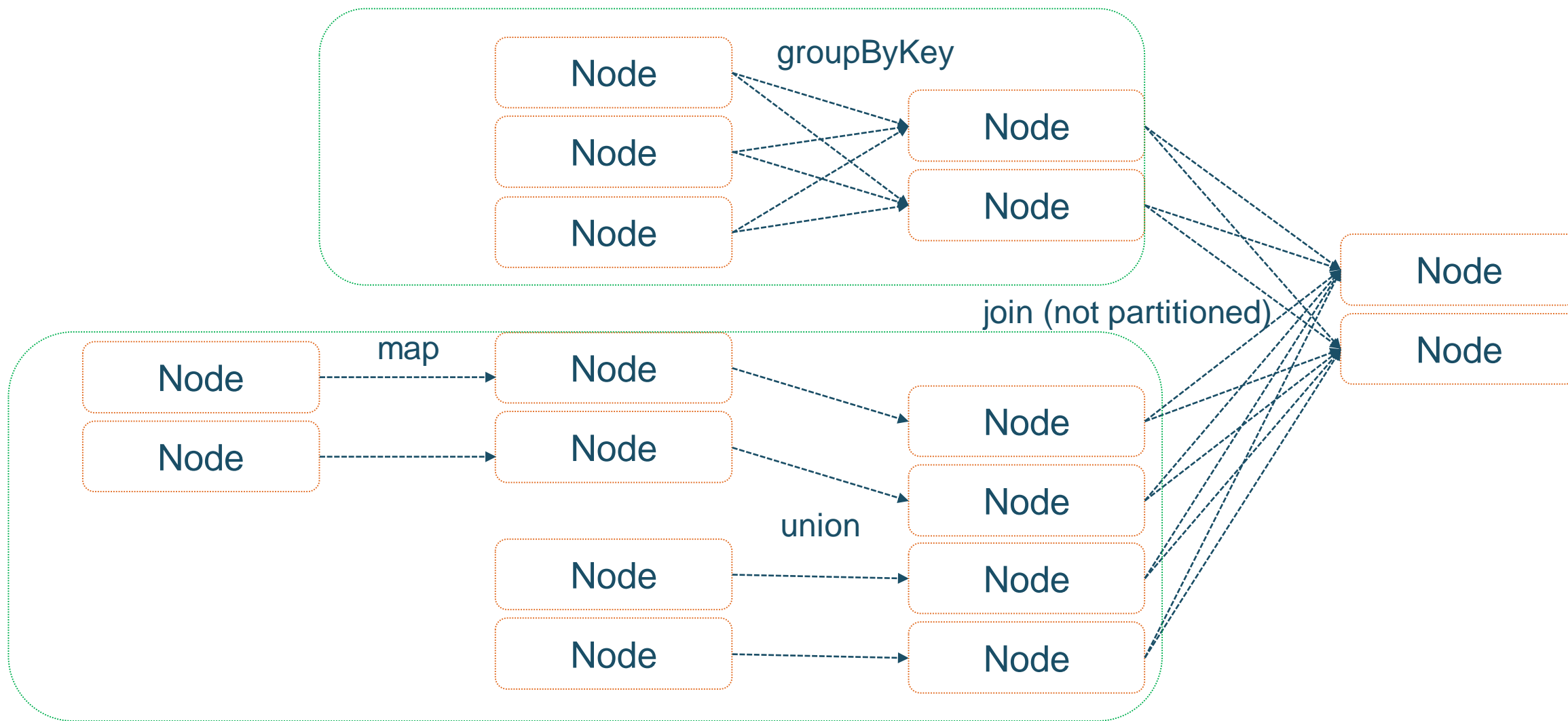
# ЗАВИСИМОСТЬ ОТ ПАРТИЦИЙ

## (В ПРОЦЕССЕ ЗАДАЧИ)



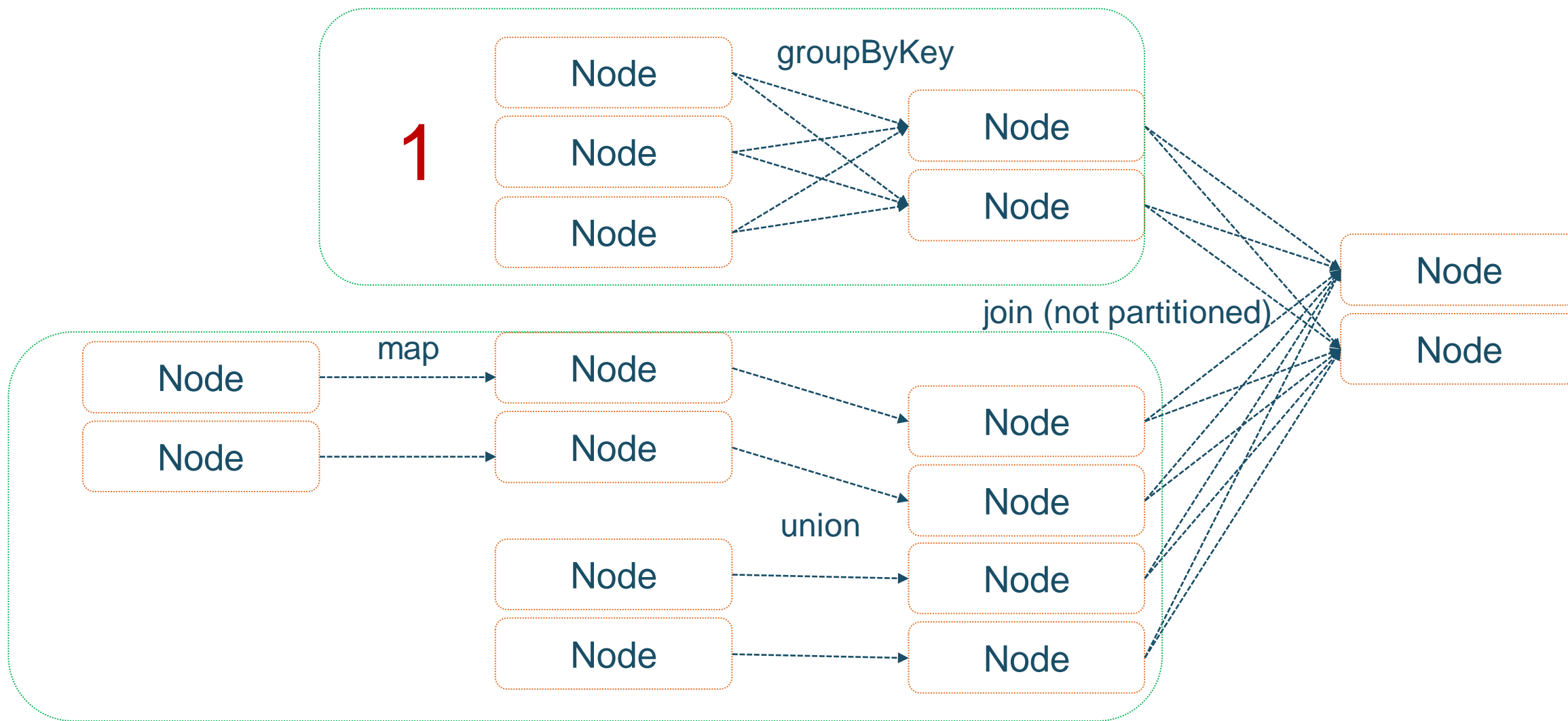
# ЗАВИСИМОСТЬ ОТ ПАРТИЦИЙ

## (В ПРОЦЕССЕ ЗАДАЧИ)



# ЗАВИСИМОСТЬ ОТ ПАРТИЦИЙ

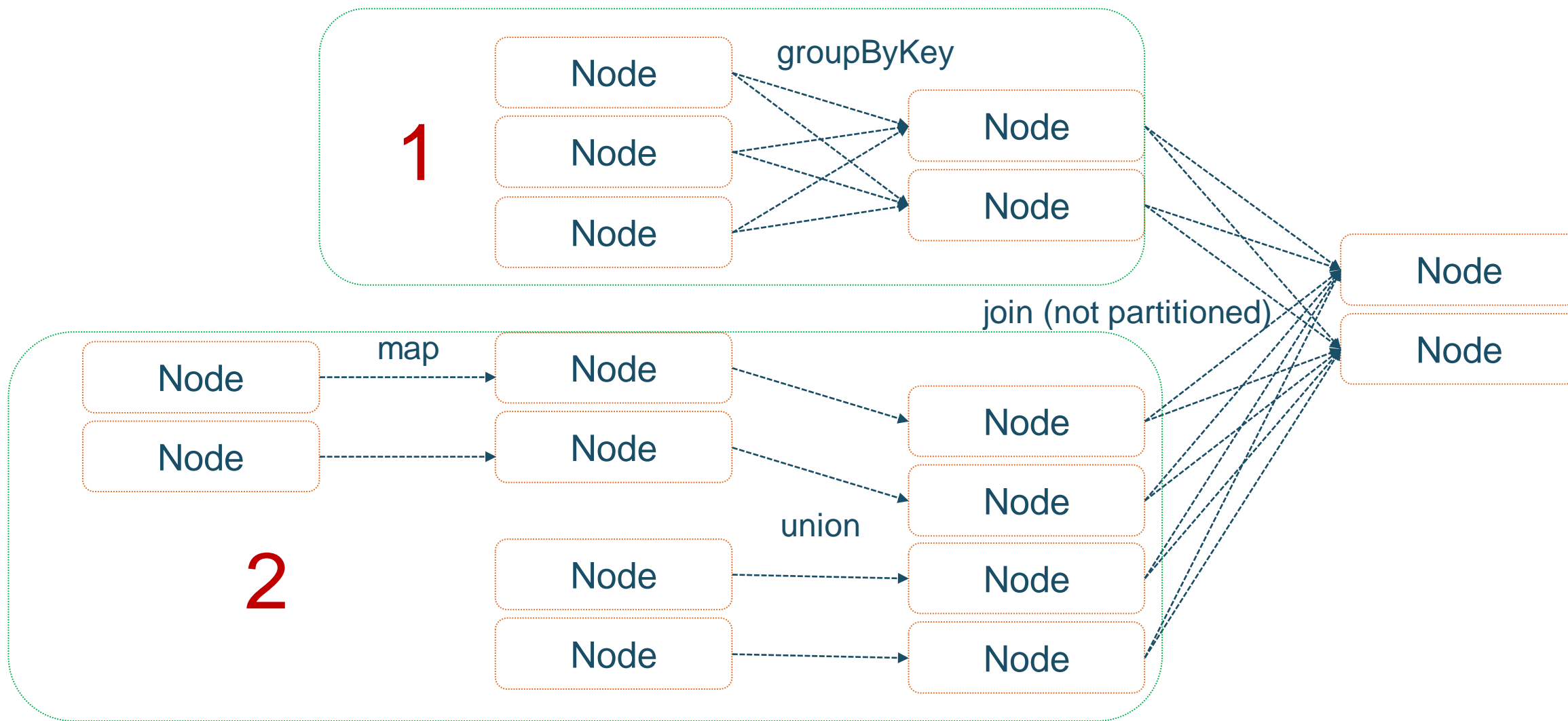
## (В ПРОЦЕССЕ ЗАДАЧИ)





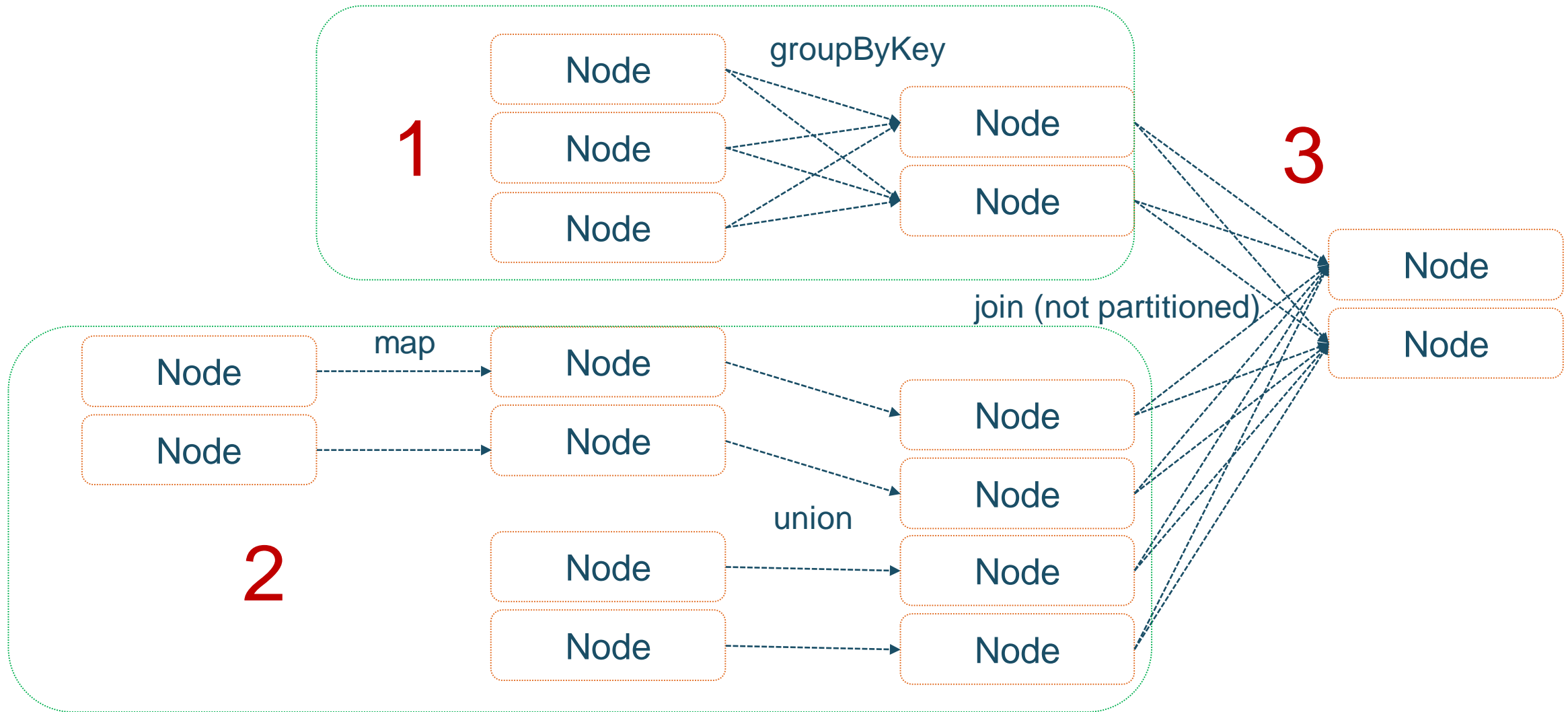
# ЗАВИСИМОСТЬ ОТ ПАРТИЦИЙ

## (В ПРОЦЕССЕ ЗАДАЧИ)



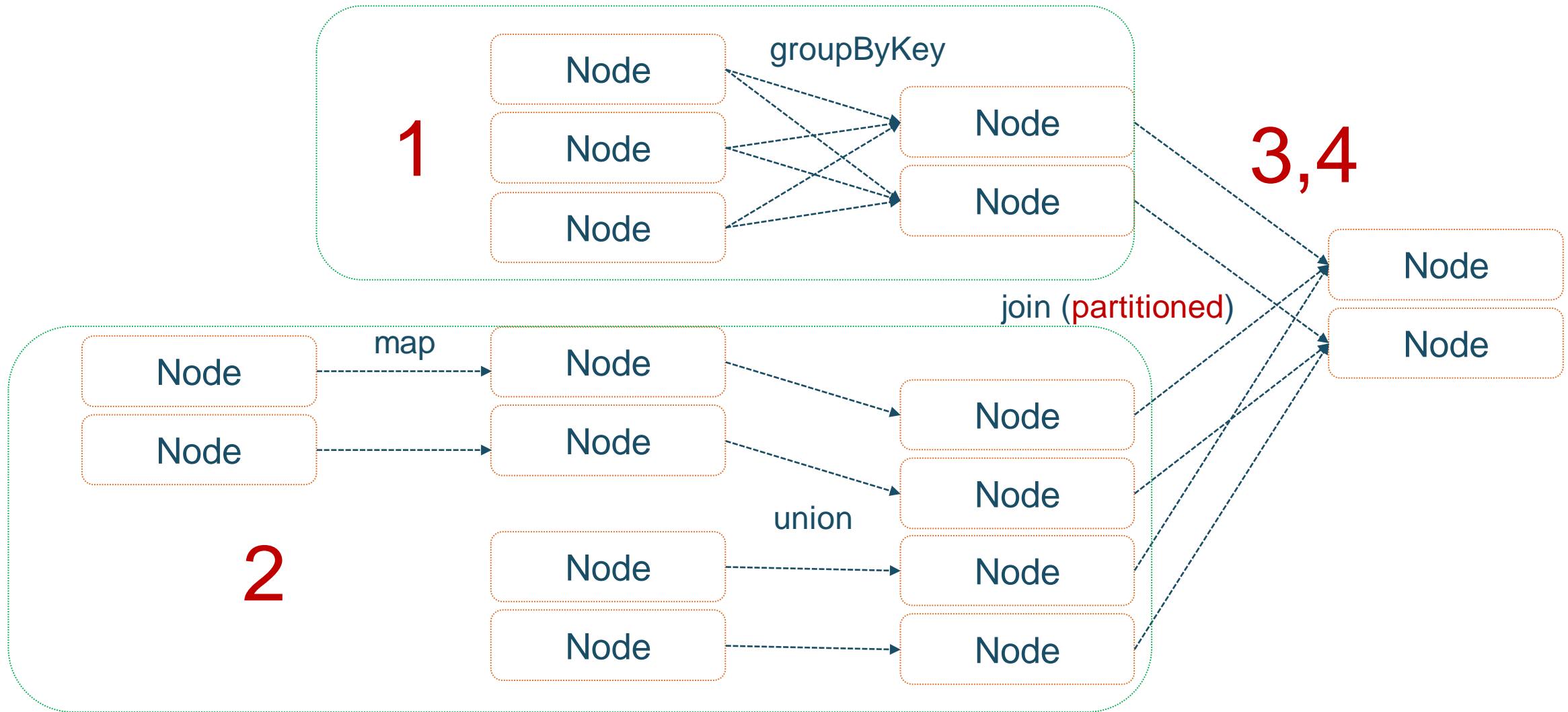
# ЗАВИСИМОСТЬ ОТ ПАРТИЦИЙ

(В ПРОЦЕССЕ ЗАДАЧИ)



# ЗАВИСИМОСТЬ ОТ ПАРТИЦИЙ

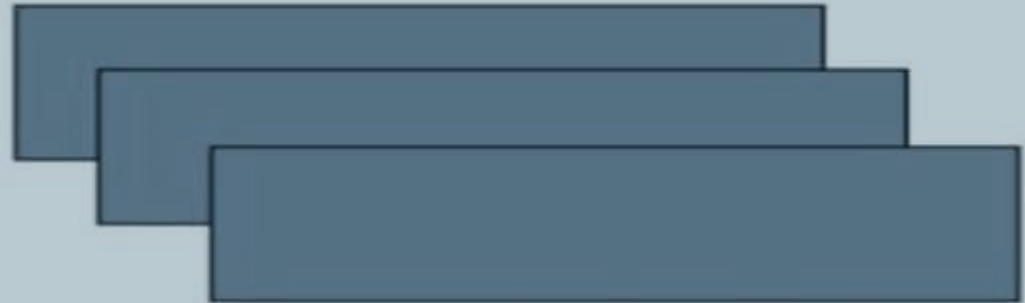
(В ПРОЦЕССЕ ЗАДАЧИ)



# ФУНКЦИИ УПРАВЛЕНИЯ ПАРТИЦИЯМИ

<code>.partitions()</code>	Список объектов партиций
<code>.dependencies()</code>	Список объектов зависимостей
<code>.compute(p, parent)</code>	Кол-во элементов в партиции Р в родительском объекте
<code>.practitioner()</code>	Метадата по партиции
<code>preferredLocations(p)</code>	Список нод, где партиция Р расположена

SPARK RDD



# RDD ПРИЧИНЫ ИСПОЛЬЗОВАНИЯ

НУЖНЫ НЕ ИЗМЕНЯЕМЫЕ ОБЪЕКТЫ

НУЖНА ТИПИЗАЦИЯ `RDD[int]`,  
`RDD[string]`

FAULT TOLERANCE

ПРОВЕСТИ КРУПНЫЕ (ГРУБЫЕ) ИЗМЕНЕНИЯ ПО  
ВСЕМУ НАБОРУ ДАННЫХ

ВАЖНО ПАРТИЦИОНИРОВАНИЕ, РАСПРЕДЕЛЕНИЕ ПО НОДАМ  
(ОПРЕДЕЛЕННОЕ)

ЕСТЬ ОГРАНИЧЕНИЯ ПО  
РЕСУРСАМ

КОГДА НУЖНО ИСПОЛЬЗОВАТЬ ДРУГИЕ  
ОПТИМИЗАТОРЫ (НЕ CATALYST)

# RDD ПРИЧИНЫ ИСПОЛЬЗОВАНИЯ

ИСПОЛЬЗОВАТЬ BROADCAST

```
spark.sparkContext.broadcast(VARIABLE)
```

ИСПОЛЬЗОВАТЬ ACCUMULATOR

```
spark.sparkContext.accumulator(Type, Func())
```

# ACCUMULATOR

```
accum = sc.accumulator(0)
```

```
from collections import Counter
```

```
class CounterAccumulatorParam(ps.accumulators.AccumulatorParam):  
    def zero(self, initialValue):  
        return Counter()  
  
    def addInPlace(self, v1, v2):  
        v1 += v2  
        return v1
```

```
accum = sc.accumulator(Counter(), CounterAccumulatorParam())
```

```
def count_null(record):  
    global accum  
  
    c = Counter()  
  
    for key, value in record.items():  
        if value == '':  
            c[key] += 1  
  
    accum.add(c)
```

```
rdd_dict.foreach(count_null)
```

```
accum.value
```



# ACCUMULATOR

```
accum = sc.accumulator(0)
```

```
from collections import Counter
```

```
class CounterAccumulatorParam(ps.accumulators.AccumulatorParam):  
    def zero(self, initialValue):  
        return Counter()  
  
    def addInPlace(self, v1, v2):  
        v1 += v2  
        return v1
```

```
accum = sc.accumulator(Counter(), CounterAccumulatorParam())
```

```
def count_null(record):  
    global accum  
  
    c = Counter()  
  
    for key, value in record.items():  
        if value == '':  
            c[key] += 1  
  
    accum.add(c)
```

```
rdd_dict.foreach(count_null)
```

```
accum.value
```

# RDD НЕ ИЗБАВЛЯЕТ ОТ ПРОБЛЕМ

НЕТ ОПТИМИЗАТОРА (В DATAFRAME / DATASET ИСПОЛЬЗУЕТСЯ CATALYST)

НУЖНО СЛЕДИТЬ ЗА ТИПАМИ ДАННЫХ

ДЕГРАДАЦИЯ ДАННЫХ ПРИ МАЛОМ КОЛ-ВЕ ОЗУ (КОГДА IN-MEMORY)

НУЖНО ИСПОЛЬЗОВАТЬ GARBAGE COLLECTION

# RDD (KEY – VALUE) | MAP – REDUCE

```
pets = sc.parallelize([("cat", 1), ("dog", 1), ("cat", 2)])
```

```
pets.reduceByKey(lambda x, y: x + y) # => {(cat, 3), (dog, 1)}
```

```
pets.groupByKey() # => {(cat, [1, 2]), (dog, [1])}
```

```
pets.sortByKey() # => {(cat, 1), (cat, 2), (dog, 1)}
```

MAPPER

- `str`(обычный файл\тс вашими\тданными)
- `list(list(str(обычный файл),  
          str(с вашими),  
          str(данными)  
          ))`
- `function(object) <- list(str)`
- `return: key – value`

# RDD (KEY – VALUE) | MAP – REDUCE

```
pets = sc.parallelize([("cat", 1), ("dog", 1), ("cat", 2)])
```

```
pets.reduceByKey(lambda x, y: x + y) # => {(cat, 3), (dog, 1)}
```

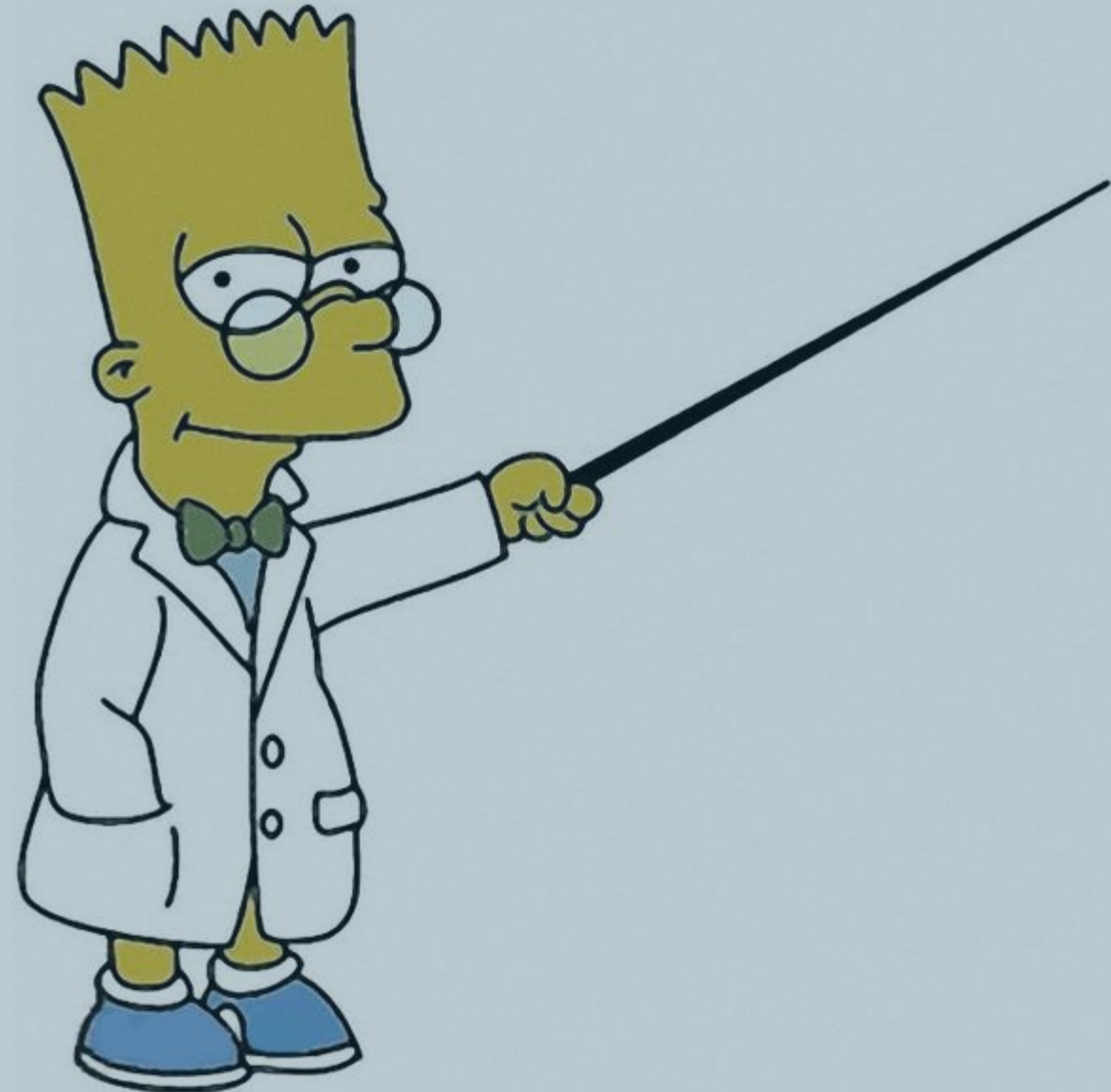
```
pets.groupByKey() # => {(cat, [1, 2]), (dog, [1])}
```

```
pets.sortByKey() # => {(cat, 1), (cat, 2), (dog, 1)}
```

MAPPER

- `str`(обычный файл\тс вашими\тданными)
- `list(list(str(обычный файл),  
str(с вашими),  
str(данными)  
))`
- `function(object) <- list(str)`
- `return: key – value`

RDD НА ПРИМЕРЕ



# RDD | WORDCOUNT

```
file_rdd.map(line => line.split(" "))  
  .map(split => (split(0), split(1).toInt))  
  .groupByKey()  
  .mapValues(iter => iter.reduce(_ + _)).collect()
```

**dummy.txt**

```
jon 2  
mary 3  
anna 1  
jon 1  
jesse 3  
mary 5
```

# RDD | PLAN

```
sc.textFile('file:///dummy.txt')
```



```
map(line => line.split(" "))
```



```
map(split => (split(0), split(1).toInt))
```



```
groupByKey()
```

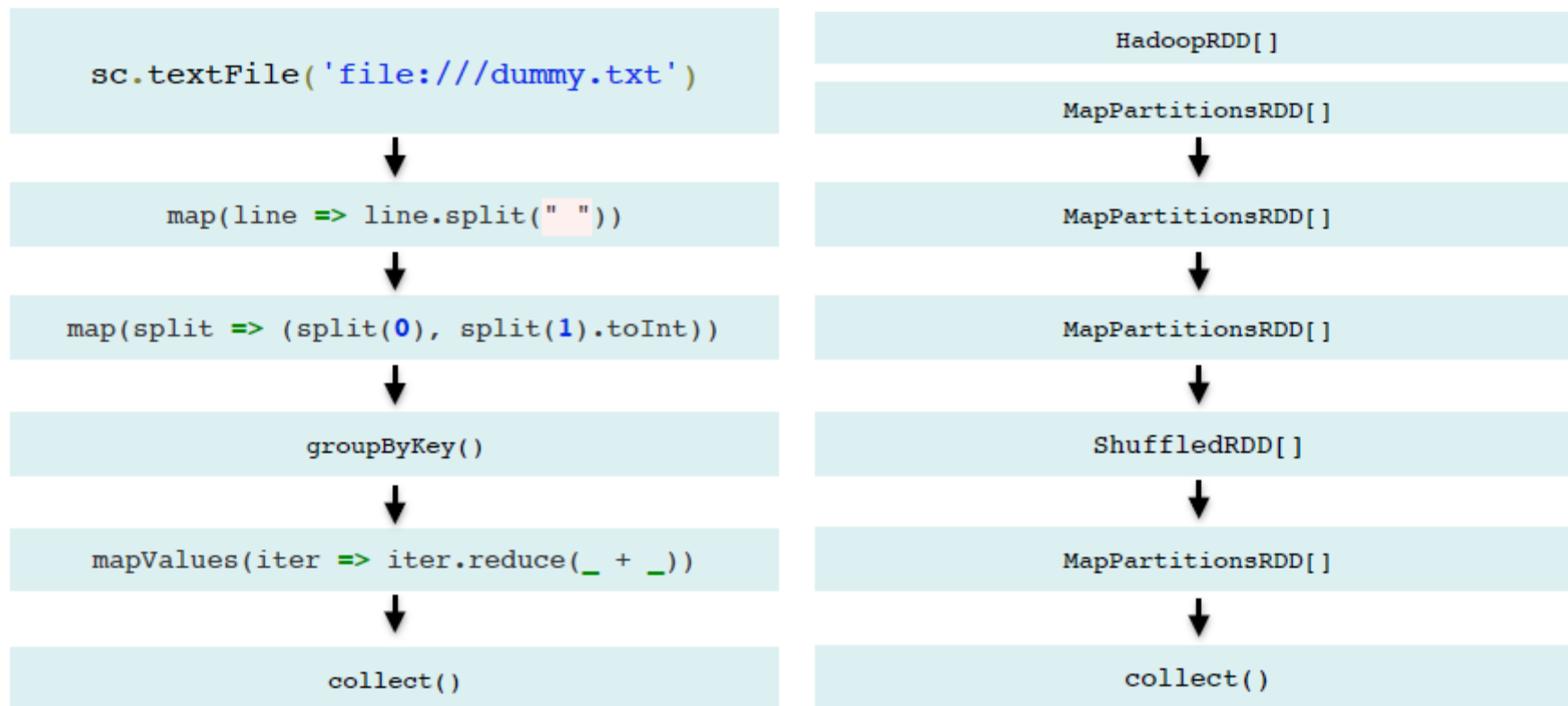


```
mapValues(iter => iter.reduce(_ + _))
```



```
collect()
```

# RDD | PLAN RDD

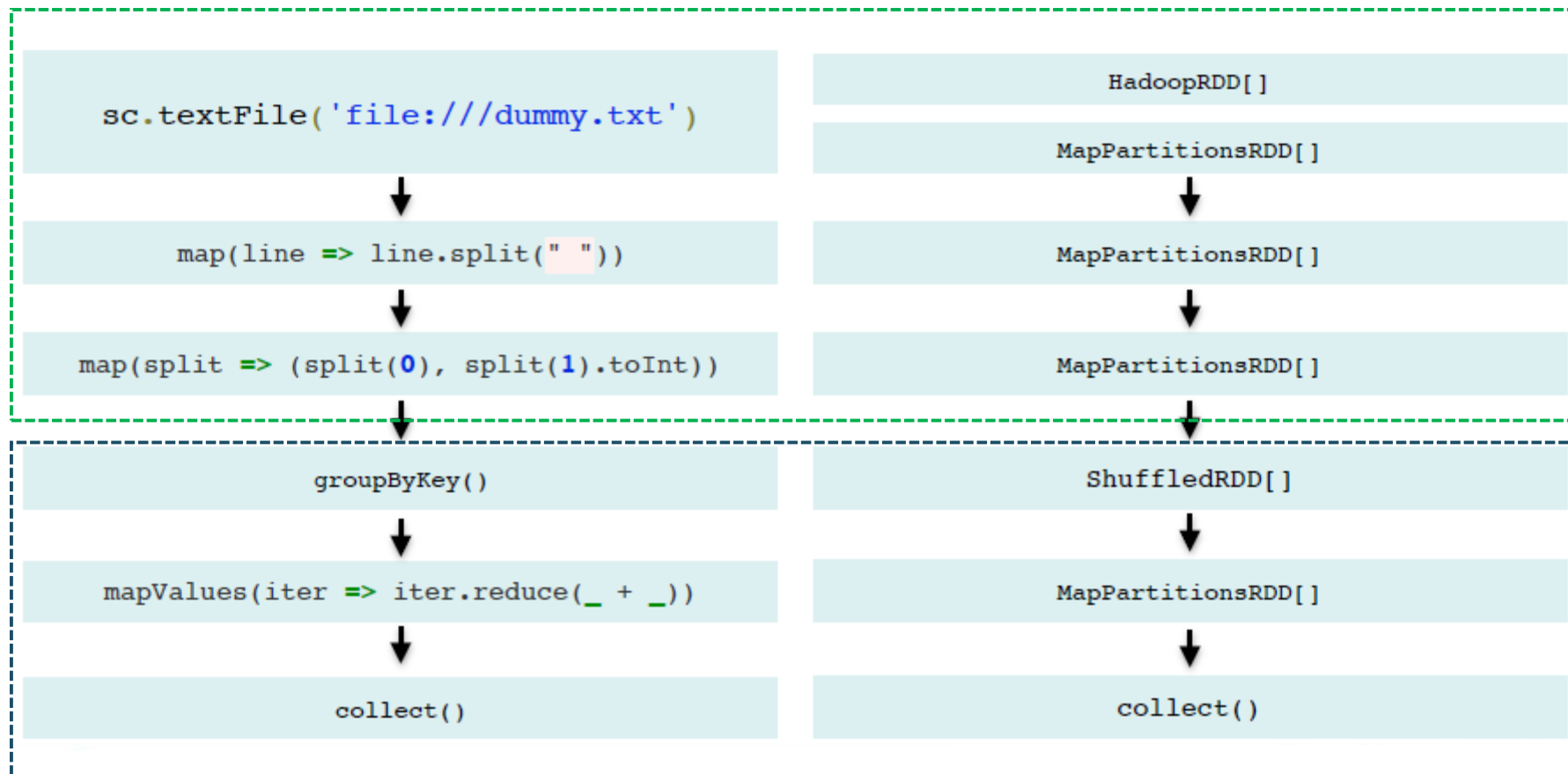




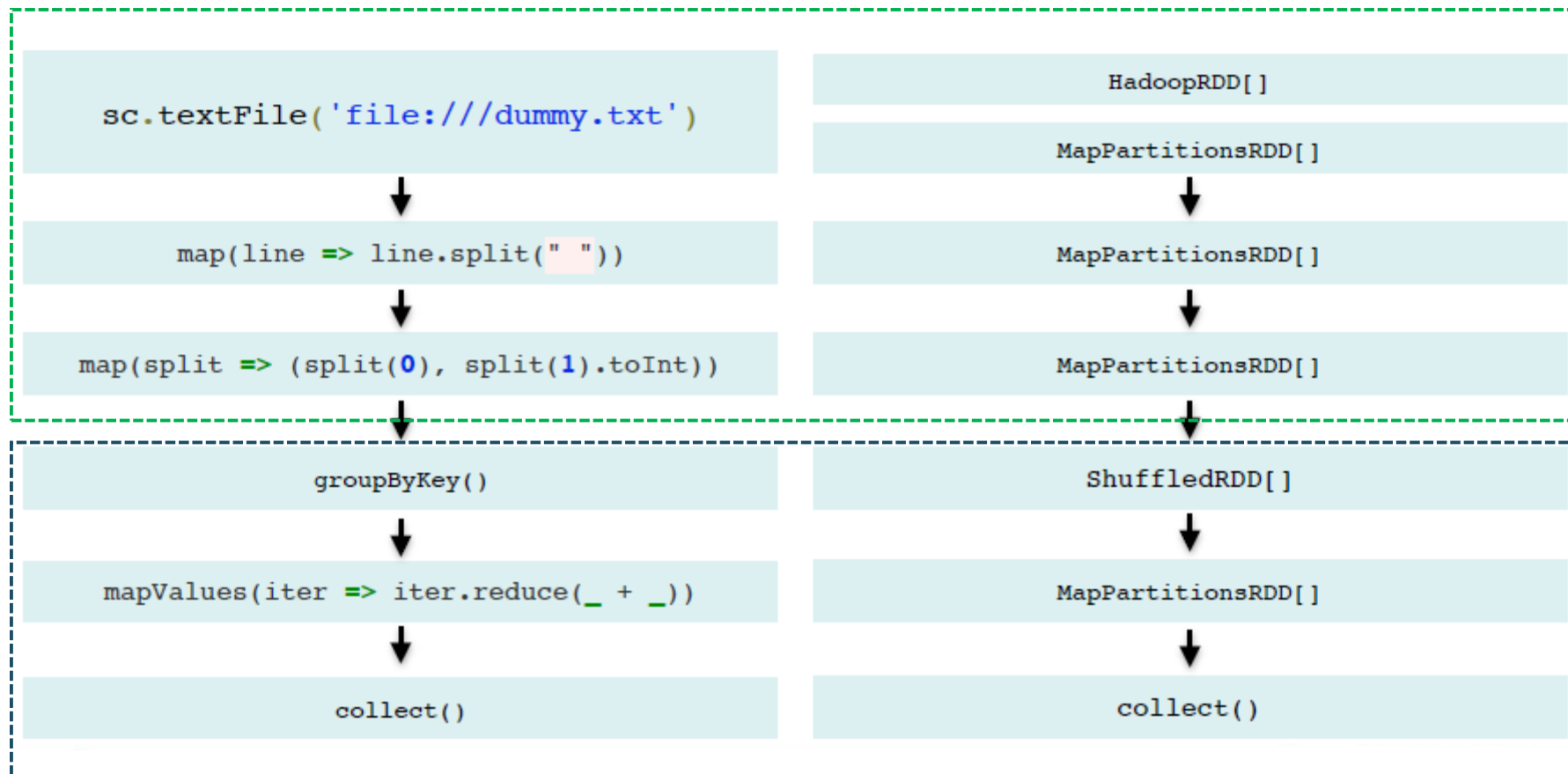
# RDD | БАРЬЕР



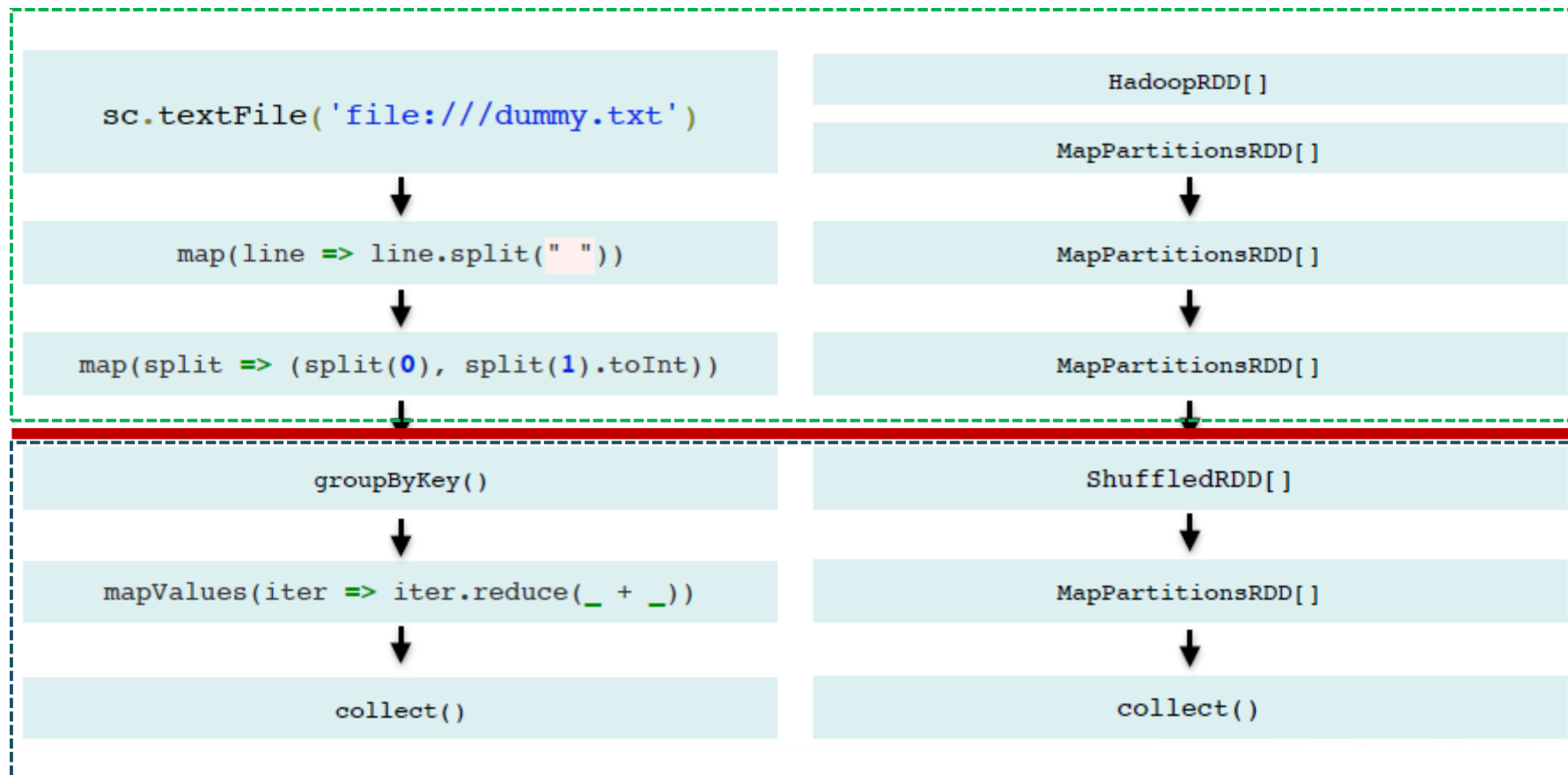
# RDD | БАРЬЕР



# RDD | БАРЬЕР

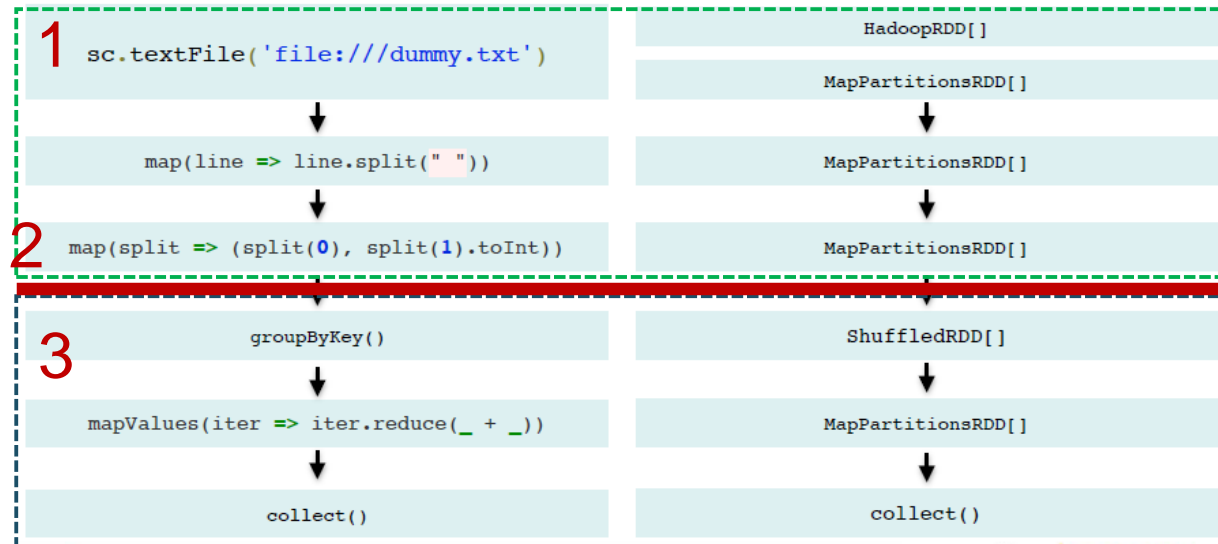


# RDD | БАРЬЕР



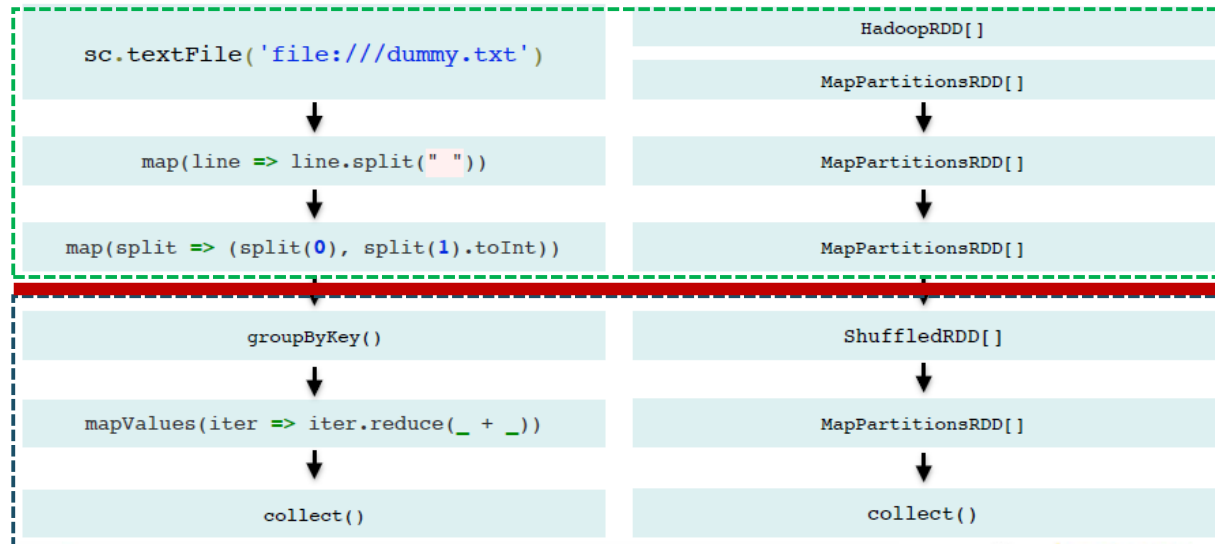
Shuffle  
барьер

# RDD | ВЫПОЛНЕНИЕ



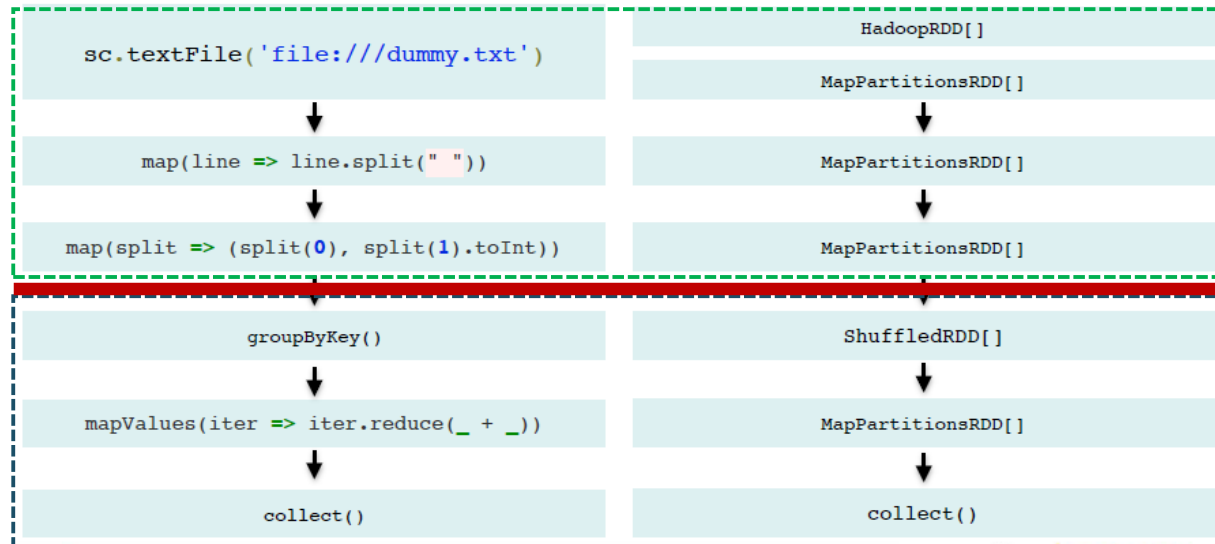
- Разбиение уровней на задачи для Executor's

# RDD | ВЫПОЛНЕНИЕ



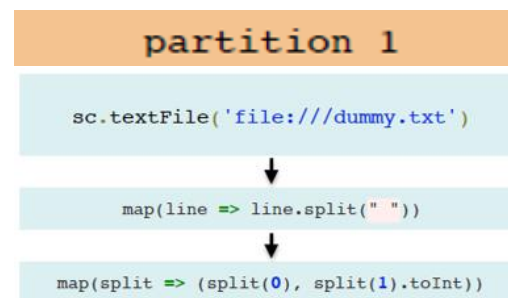
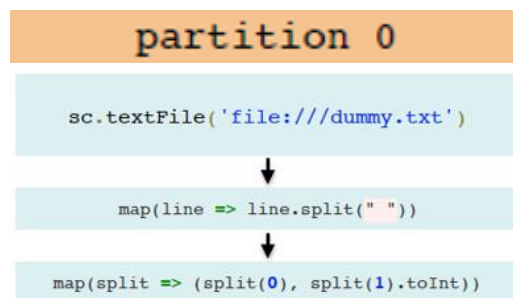
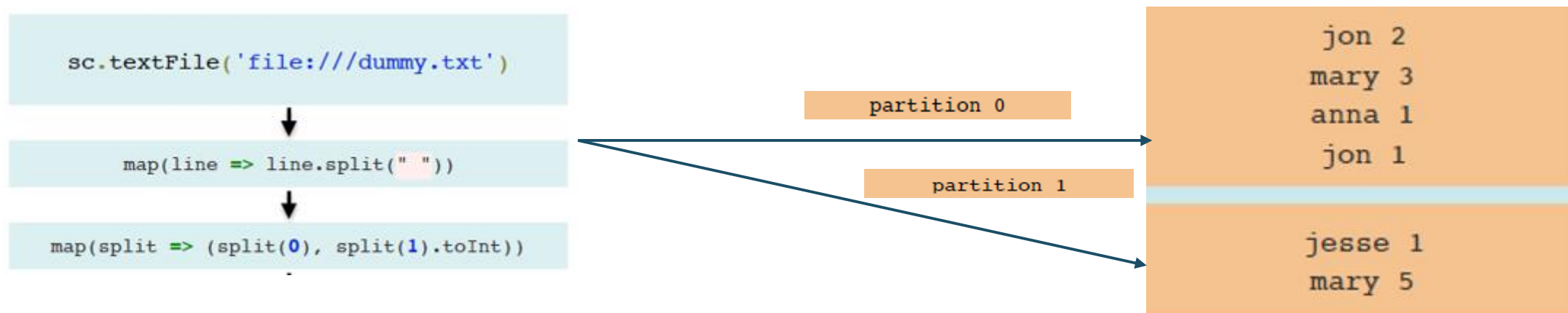
- Разбиение уровней на задачи для Executor's
- Задача – это процесс партиционирования данных и вычисления

# RDD | ВЫПОЛНЕНИЕ



- Разбиение уровней на задачи для Executor's
- Задача – это процесс партиционирования данных и вычисления
- Выполнение каждой задачи

# RDD | ПАРТИЦИОНИРОВАНИЕ ЗАДАЧ





# RDD | SHUFFLE

jon 2  
mary 3  
anna 1  
jon 1

jesse 1  
mary 5

`groupByKey()`



`mapValues(iter => iter.reduce(_ + _))`

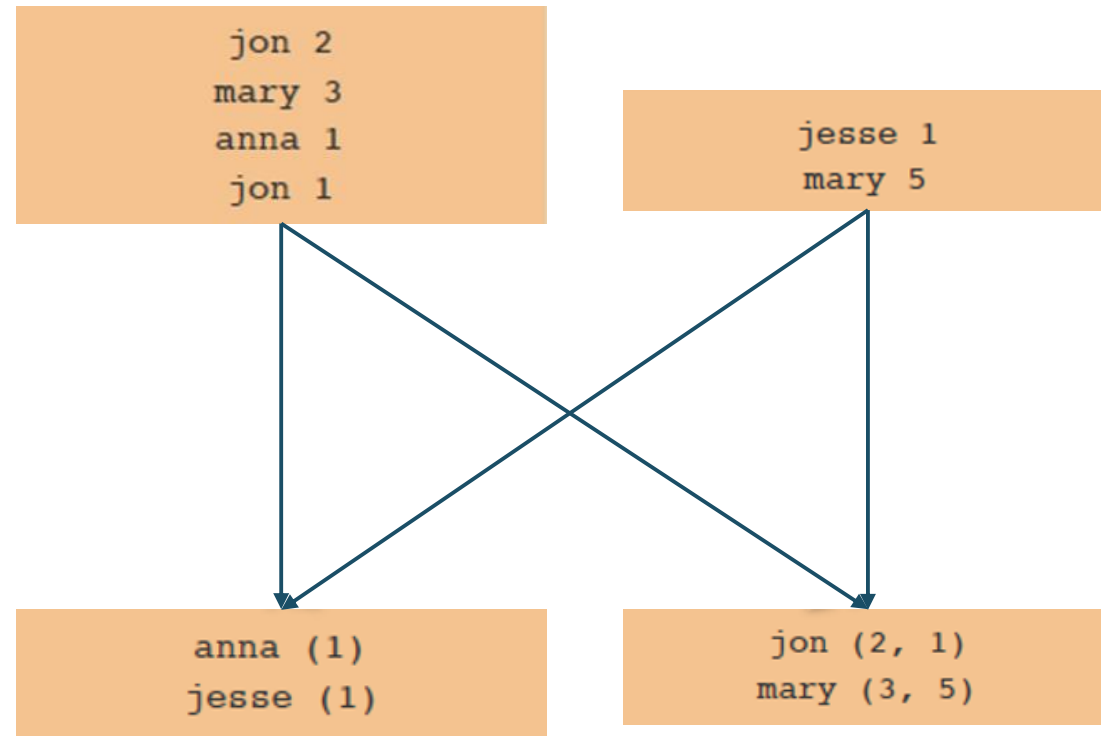
# RDD | SHUFFLE

```
groupByKey()
```



```
mapValues(iter => iter.reduce(_ + _))
```

- Перераспределение данных по партициям
- Hash key для создания бакетов
- Выполнение процесса с записью на диск temp файлов (как Hadoop)



# RDD TRANSFORMATIONS

## RDD ACTIONS



# RDD ТЕРМИНЫ

TRANSFORMATION

«Ленивое» вычисление. Return – **новый RDD**

ACTION

Запускает **выполнение** вычислений над данными.  
Return – **финальное значение (на драйвер)**

# RDD TRANSFORMATION

1  
`nums = sc.parallelize([1,2,3])`

2  
`squared = nums.map(lambda x: x*x) # => {1, 4, 9}`

3  
`even = squared.filter(lambda x: x % 2 == 0) # => [4]`

4  
`nums.flatMap(lambda x: range(x)) # => {0, 0, 1, 0, 1, 2}`

# RDD TRANSFORMATION

1

```
nums = sc.parallelize([1,2,3])
```

2

```
squared = nums.map(lambda x: x*x) # => {1, 4, 9}
```

3

```
even = squared.filter(lambda x: x % 2 == 0) # => [4]
```

4

```
nums.flatMap(lambda x: range(x)) # => {0, 0, 1, 0, 1, 2}
```

Количество вычислений = 1!

# RDD ACTION

1

```
nums = sc.parallelize([1, 2, 3])
```

2

```
nums.collect() # => [1, 2, 3]
```

3

```
nums.take(2) # => [1, 2]
```

4

```
nums.count() # => 3
```

5

```
nums.reduce(lambda: x, y: x + y) # => 6
```

6

```
nums.saveAsTextFile("hdfs://file.txt")
```

# RDD ACTION

1

```
nums = sc.parallelize([1, 2, 3])
```

2

```
nums.collect() # => [1, 2, 3]
```

3

```
nums.take(2) # => [1, 2]
```

4

```
nums.count() # => 3
```

5

```
nums.reduce(lambda: x, y: x + y) # => 6
```

6

```
nums.saveAsTextFile("hdfs://file.txt")
```

Количество  
вычислений = 6



# RDD SCRIPT

```
B [1]: import random

flips = 100000

coins = range(1, flips + 1)

heads = (
    sc.parallelize(coins)
    .map(lambda i: random.random())
    .filter(lambda r: r < 0.51)
    .count()
)
```

Generator

Создаем RDD

Action

Transformations

# RDD SCRIPT

```
B [1]: import random

flips = 100000

coins = range(1, flips + 1)

heads = (
    sc.parallelize(coins)
    .map(lambda i: random.random())
    .filter(lambda r: r < 0.51)
    .count()
)
```

- Создаем функцию
- Применяем её к объекту

# RDD SCRIPT

```
B [1]: import random

flips = 100000

coins = range(1, flips + 1)

heads = (
    sc.parallelize(coins)
    .map(lambda i: random.random())
    .filter(lambda r: r < 0.51)
    .count()
)
```

# RDD SCRIPT

```
[1]: import random

flips = 100000

coins = range(1, flips + 1)

heads = (
    sc.parallelize(coins)
    .map(lambda i: random.random())
    .filter(lambda r: r < 0.51)
    .count()
)
```

==

```
[2]: import random

flips = 100000

coins = range(1, flips + 1)

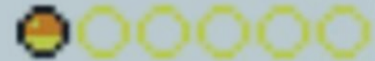
rdd = sc.parallelize(coins)

flips_rdd = rdd.map(lambda i: random.random())
heads_rdd = flips_rdd.filter(lambda r: r < 0.51)

heads = heads_rdd.count()
```

# DATAFRAMES

	name	age	state	num_children	num_pets
0	john	23	iowa	2	0
1	mary	78	dc	2	4
2	peter	22	california	0	0
3	jeff	19	texas	1	5
4	bill	45	washington	2	0
5	lisa	33	dc	1	0



wild DATAFRAME appeared!

# ПРИНЦИП РАБОТЫ С DF

Создать DataFrame из ресурсов

Применить трансформаторы к DataFrame  
(select, filter, etc.)

Применить экшены к DataFrame  
(show, saveAs...., etc.)

**КАК С RDD?**

# НОВАЯ СТРУКТУРА!

TABLE

Состоит из **строк** и **колонок**

ROW

Из объекта **rows** – namedtuple - rdd

# НОВАЯ МОДЕЛЬ РАЗРАБОТКИ

```
: df = sqlContext.read.text("")  
onlyComments = df.filter("status == 'comments'")
```

«Ленивые» вычисления



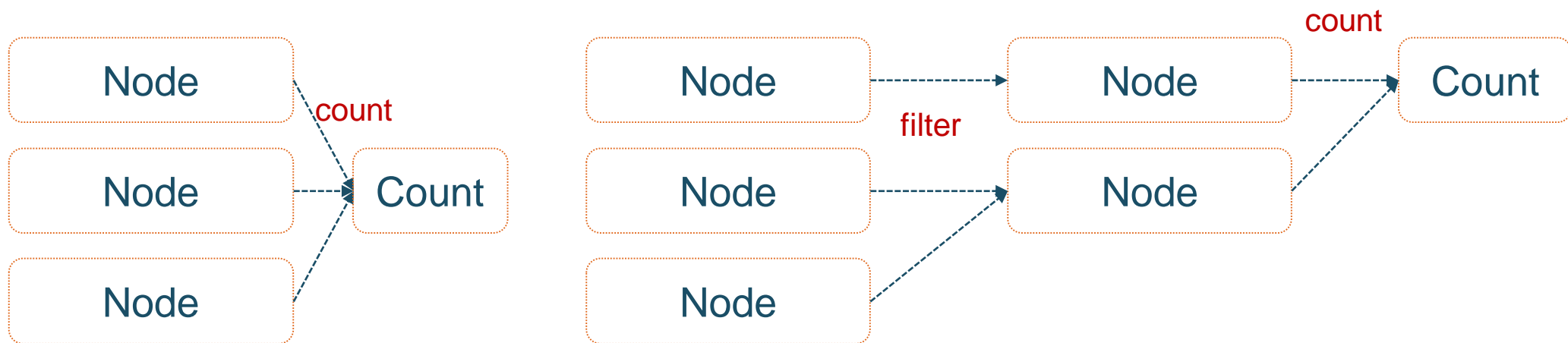
# НОВАЯ МОДЕЛЬ РАЗРАБОТКИ

```
: df = sqlContext.read.text("")  
onlyComments = df.filter("status == 'comments'")  
onlyComments.count() / df.count()
```

?

# НОВАЯ МОДЕЛЬ РАЗРАБОТКИ

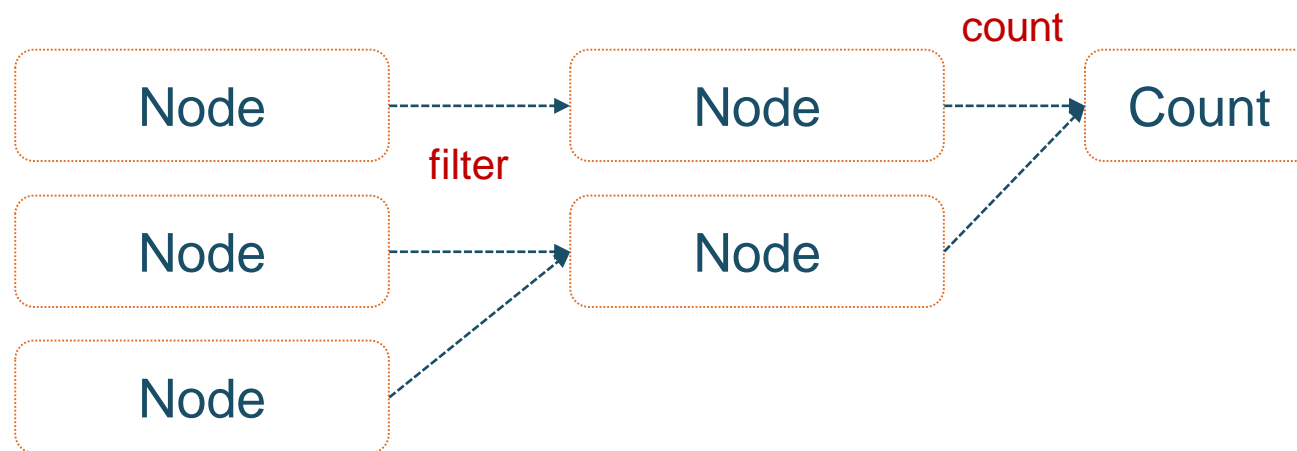
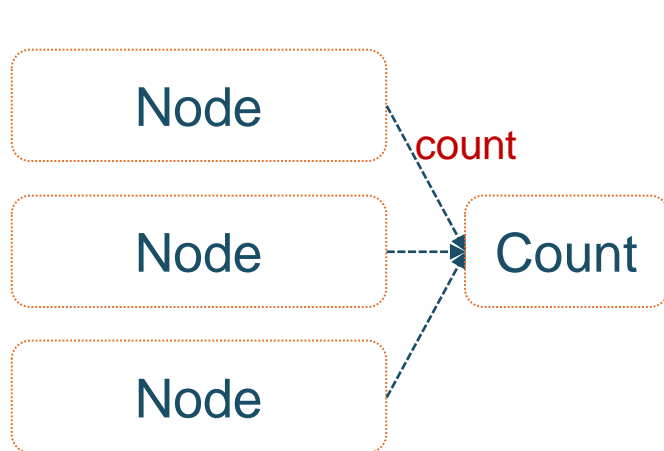
```
: df = sqlContext.read.text("")  
onlyComments = df.filter("status == 'comments'")  
onlyComments.count() / df.count()
```



# НОВАЯ МОДЕЛЬ РАЗРАБОТКИ

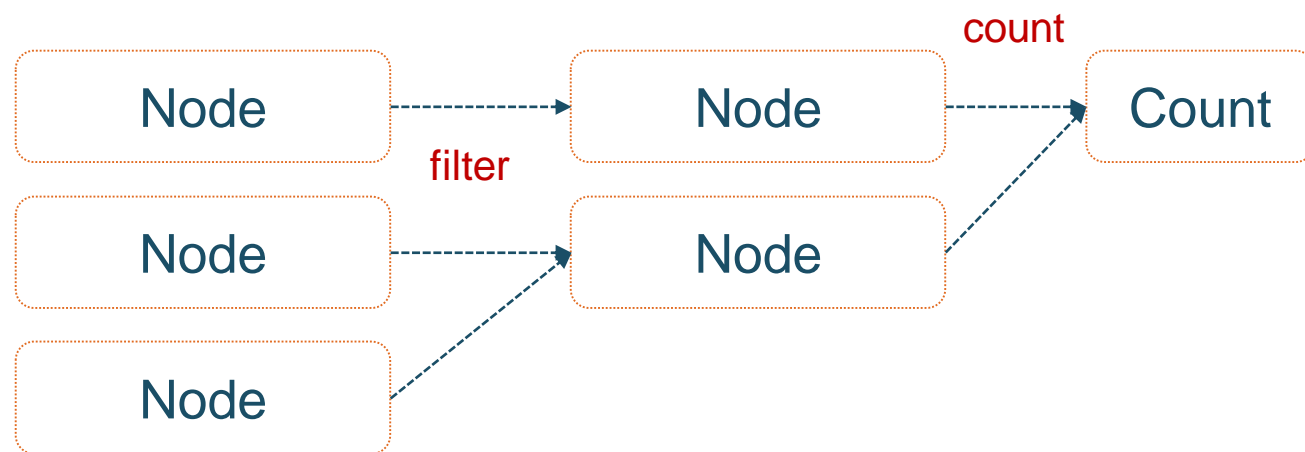
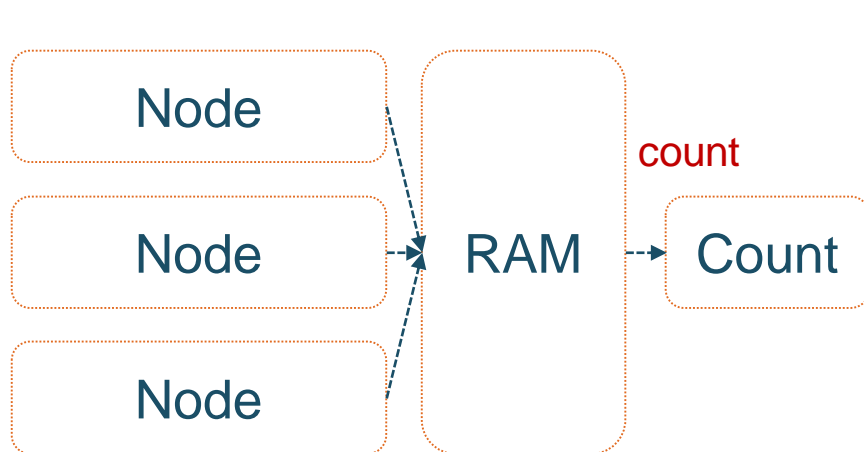
```
: df = sqlContext.read.text("")  
onlyComments = df.filter("status == 'comments'")  
onlyComments.count() / df.count()
```

1. Чтение данных (2 раза)
2. Подсчет результата по патрициям (2 раза)
3. Фильтр
4. Соединение результата на драйвере



# НОВАЯ МОДЕЛЬ РАЗРАБОТКИ

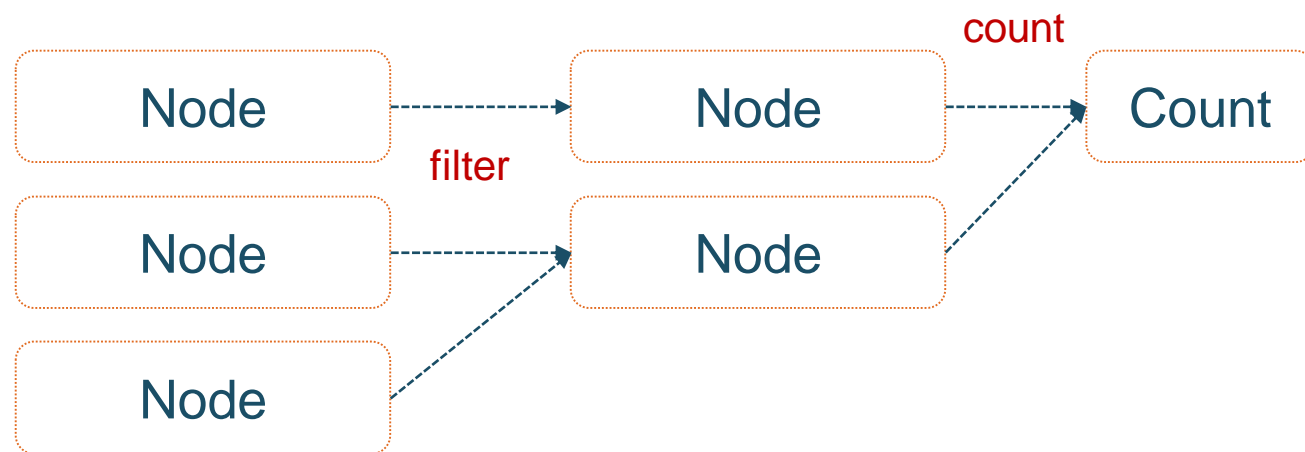
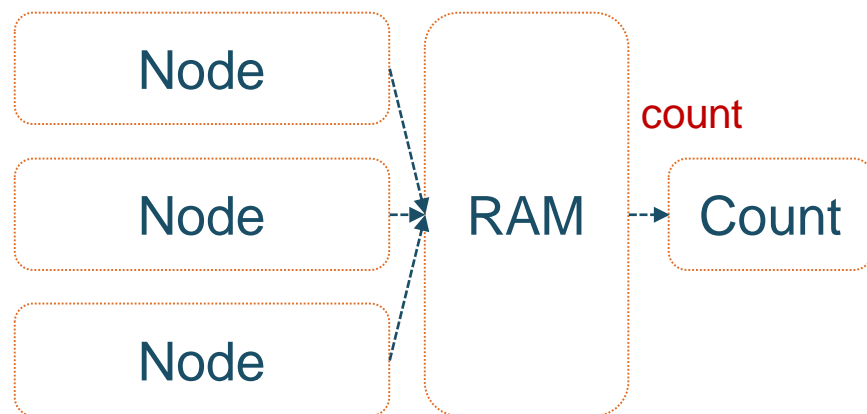
```
: df = sqlContext.read.text("")  
df.cache()  # cached!  
  
onlyComments = df.filter("status == 'comments'")  
  
onlyComments.count() / df.count()
```



# НОВАЯ МОДЕЛЬ РАЗРАБОТКИ

```
: df = sqlContext.read.text("")  
df.cache() # cached!  
onlyComments = df.filter("status == 'comments'")  
onlyComments.count() / df.count()
```

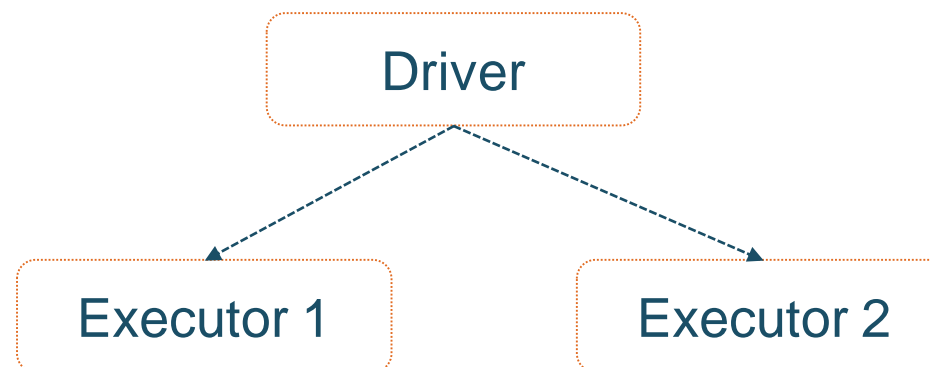
1. Чтение данных
2. Подсчет результата по патрициям (2 раза)
3. Фильтр
4. Соединение результата на драйвере



# НОВАЯ МОДЕЛЬ РАЗРАБОТКИ

- Python код работает на driver
- Transformations на executor'ax
- Actions на executor'ax и driver

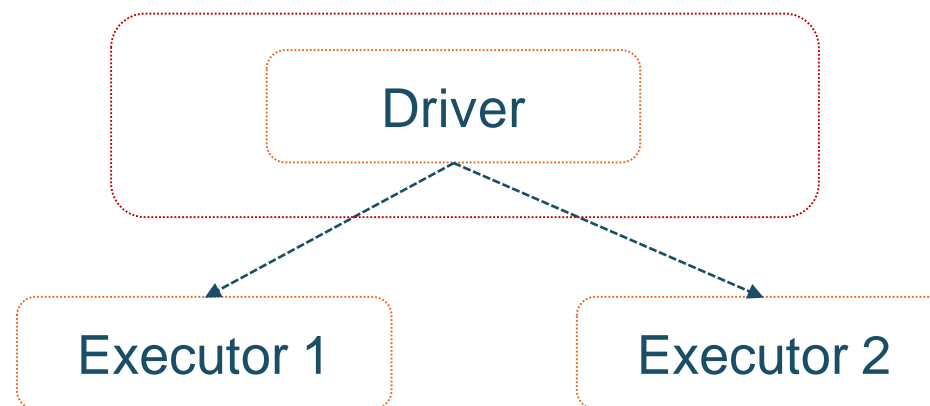
```
: df = sqlContext.read.text("")  
  
onlyComments = df.filter("status == 'comments'")  
  
onlyComments.count() / df.count()
```



# НОВАЯ МОДЕЛЬ РАЗРАБОТКИ

- Python код работает на driver
- Transformations на executor'ax
- Actions на executor'ax и driver

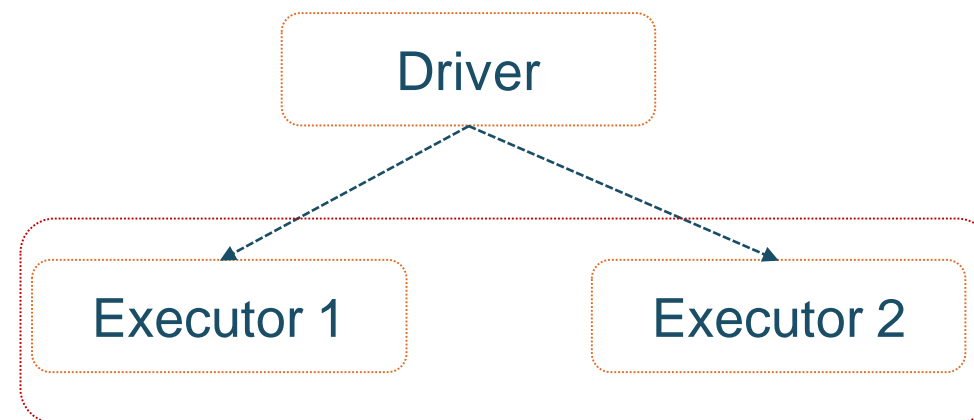
```
: df = sqlContext.read.text("")  
onlyComments = df.filter("status == 'comments'")  
onlyComments.count() / df.count()
```



# НОВАЯ МОДЕЛЬ РАЗРАБОТКИ

- Python код работает на driver
- Transformations на executor'ах  
(а ещё, во время чтения)
- Actions на executor'ах и driver

```
: df = sqlContext.read.text("")  
onlyComments = df.filter("status == 'comments'")  
onlyComments.count() / df.count()
```

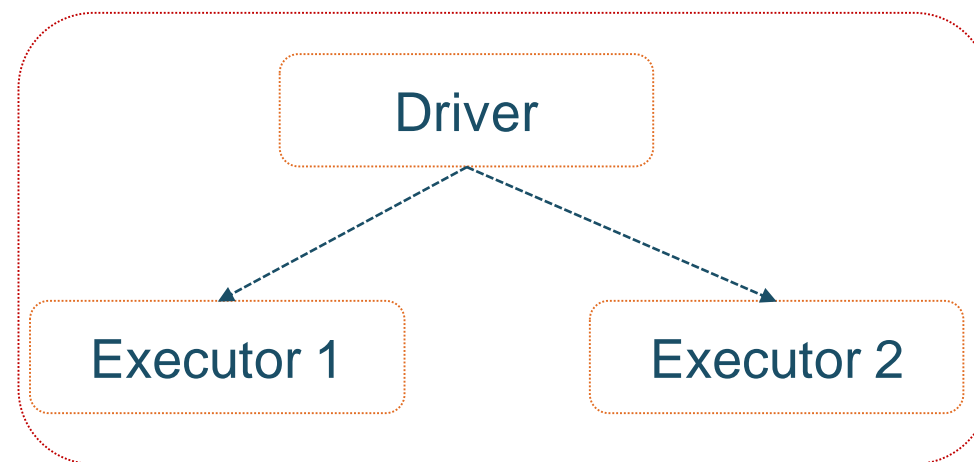




# НОВАЯ МОДЕЛЬ РАЗРАБОТКИ

- Python код работает на driver
- Transformations на executor'ax
- Actions на executor'ax и driver

```
: df = sqlContext.read.text("")  
onlyComments = df.filter("status == 'comments'")  
onlyComments.count() / df.count()
```



# НОВАЯ МОДЕЛЬ РАЗРАБОТКИ

```
context = ps.HiveContext(sc)

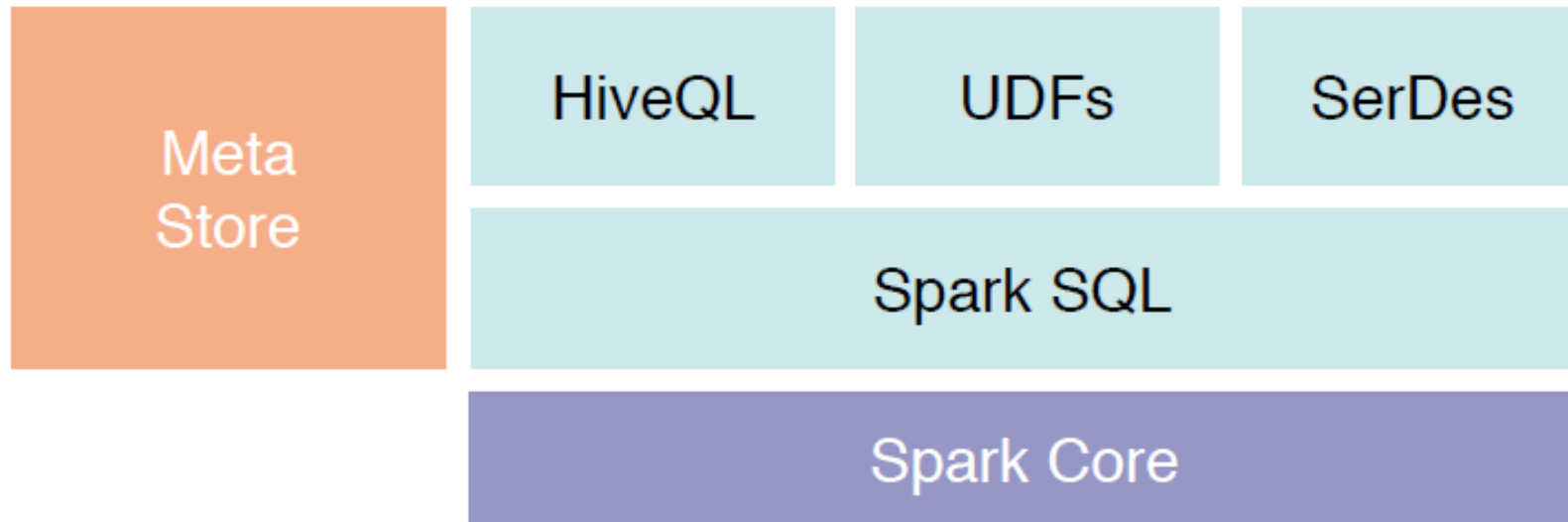
# query with SQL
results = context.sql(
    "SELECT * FROM people")

# apply Python transformation
names = results.map(lambda p: p.name)
```

Spark SQL

Spark Core

# ... IN HIVE CONTEXT



# DATAFRAME IN ACTION

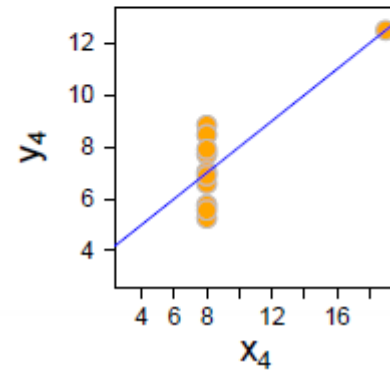
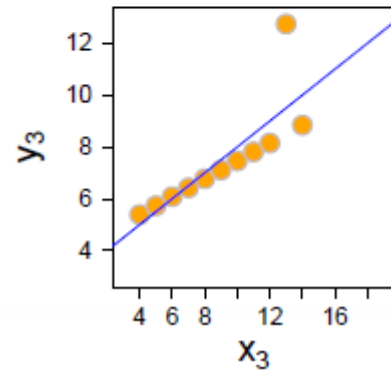
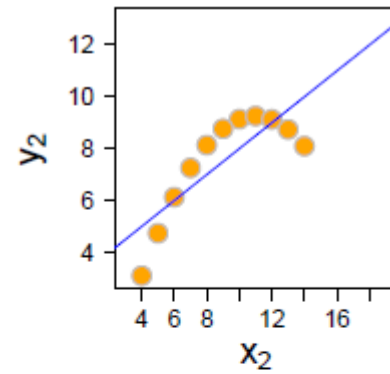
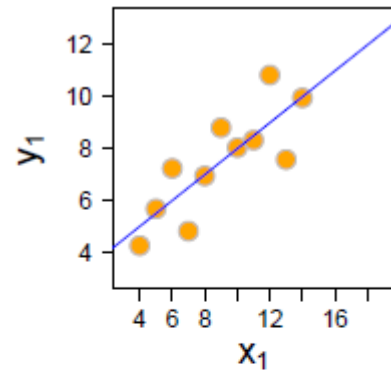


...

?

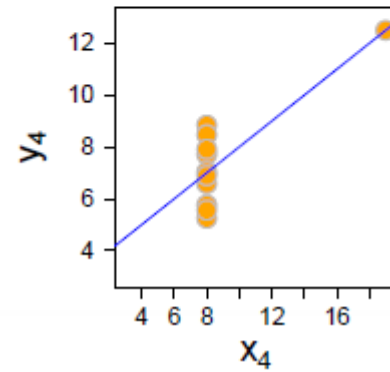
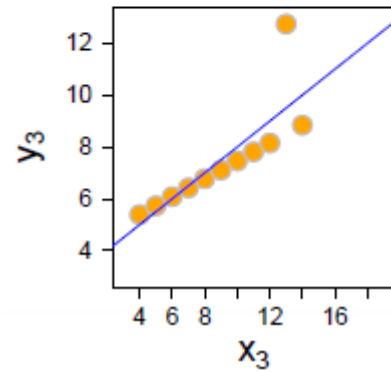
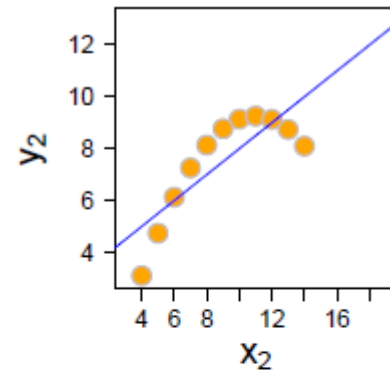
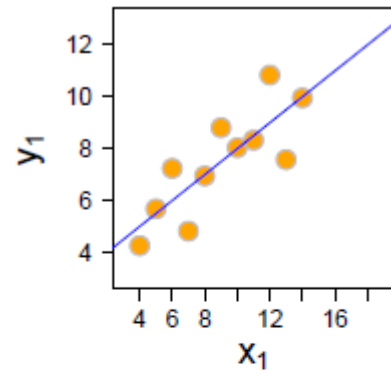
# EDA

## GREAT DESCRIBE



# EDA

## GREAT DESCRIBE



<i>Mean (x)</i>	9
<i>Sample Variance (x)</i>	11
<i>Mean (y)</i>	7.50
<i>Sample Variance (y)</i>	4.127
<i>Correlation</i>	0.816
<i>Linear Regression</i>	$y = 3.00 + 0.500x$

# EDA

- Какой размер данных?
- Как много уникальных значений?
- Как много пропусков (пустых значений)?
- Статистика для числовых значений?
- Распределение по колонке?



# ПРОПУСКИ

- `column.isNull()`
- `dataframe.dropna(column_name)`
- `dataframe.fillna()`
- `dataframe.replace(to_replace, value)`
- `pyspark.accumulators`

# ПРОПУСКИ

- `column.isNull()`
- `dataframe.fillna()`

```
df.filter(df_dates['students_reached'].isNull()).select('students_reached', 'funding_status').collect()
```

```
[Row(students_reached=None, funding_status=u'completed'),  
Row(students_reached=None, funding_status=u'completed'),  
Row(students_reached=None, funding_status=u'expired'),  
Row(students_reached=None, funding_status=u'completed'),  
...]
```

```
df_no_null = df.fillna(0, ['students_reached'])
```

# ПРОПУСКИ

- `column.isNull()`
- `dataframe.fillna()`

```
df.filter(df_dates['students_reached'].isNull()).select('students_reached', 'funding_status').collect()
```

```
[Row(students_reached=None, funding_status=u'completed'),  
Row(students_reached=None, funding_status=u'completed'),  
Row(students_reached=None, funding_status=u'expired'),  
Row(students_reached=None, funding_status=u'completed'),  
...]
```

```
df_no_null = df.fillna(0, ['students_reached'])
```

# ACCUMULATOR

```
accum = sc.accumulator(0)
```

```
from collections import Counter
```

```
class CounterAccumulatorParam(ps.accumulators.AccumulatorParam):  
    def zero(self, initialValue):  
        return Counter()  
  
    def addInPlace(self, v1, v2):  
        v1 += v2  
        return v1
```

```
accum = sc.accumulator(Counter(), CounterAccumulatorParam())
```

```
def count_null(record):  
    global accum  
  
    c = Counter()  
  
    for key, value in record.items():  
        if value == '':  
            c[key] += 1  
  
    accum.add(c)
```

```
rdd_dict.foreach(count_null)
```

```
accum.value
```

# ОБЩАЯ СТАТИСТИКА

- `dataframe.describe(column_name)`

```
df.select('total_donations', 'num_donors', 'students_reached', \
         df_dates['total_price_excluding_optional_support'].alias('p_exclude'), \
         df_dates['total_price_including_optional_support'].alias('p_include')) \
.describe().show()
```

summary	total_donations	num_donors	students_reached	p_exclude	p_include
count	771929	771929	771779	771929	771929
mean	370.85023398481707	4.264279486843997	96.71620114048193	569.6223687446723	676.180708551764
stddev	733.4647726421459	6.132976060232441	2118.592960253374	11763.955807309705	14344.347534777195
min	0.0	0.0	0.0	0.0	0.0
max	244778.0	521.0	999999.0	1.0250017E7	1.2500021E7

# ВИЗУАЛИЗАЦИЯ РАСПРЕДЕЛЕНИЙ

## **RDD**

- `rdd.histogram()`
- `rdd.stats()`

## **DataFrame**

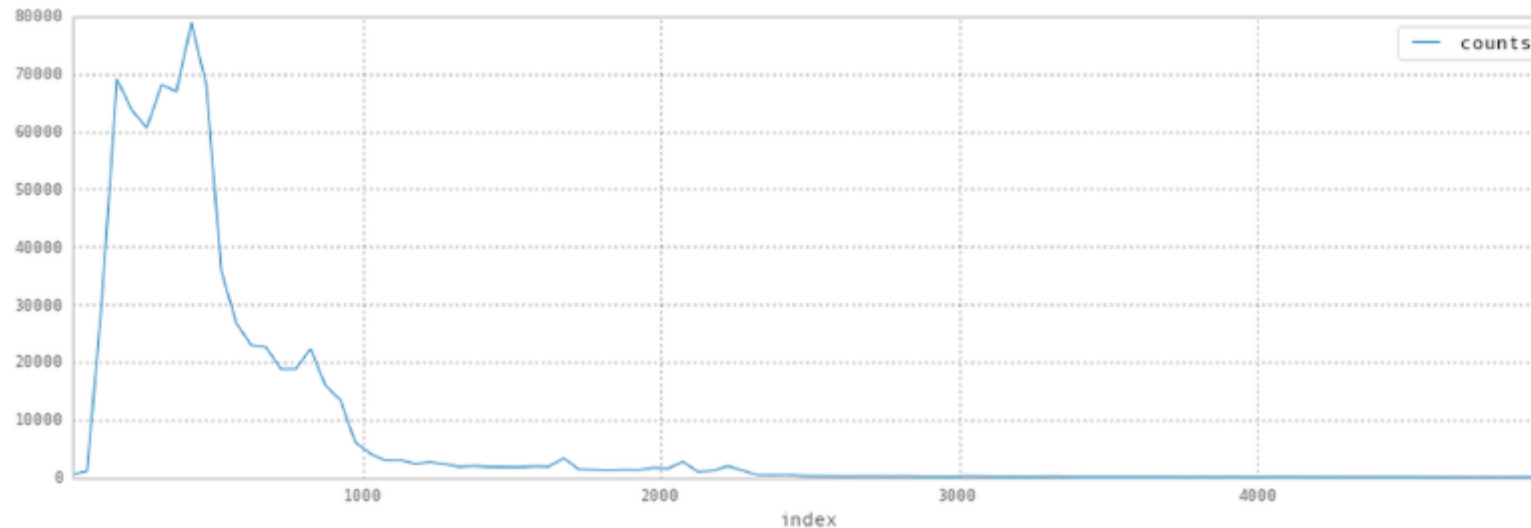
- `dataframe.groupby('column_name').count()`
- `dataframe.describe('column_name')`

# ВИЗУАЛИЗАЦИЯ РАСПРЕДЕЛЕНИЙ

```
price_rdd = df_no_null.select('total_price_excluding_optional_support').rdd.map(lambda r: r.asDict().values()[0])
```

```
def plot_rdd_hist(hist):  
    idx = []  
  
    for i in range(len(hist[0]) - 1):  
        idx.append((hist[0][i] + hist[0][i+1]) / 2)  
  
    pd.DataFrame({'counts': hist[1], 'index': idx}).set_index('index').plot(figsize=(16,5))
```

```
plot_rdd_hist(price_rdd.filter(lambda x: x < 5000).histogram(100))
```

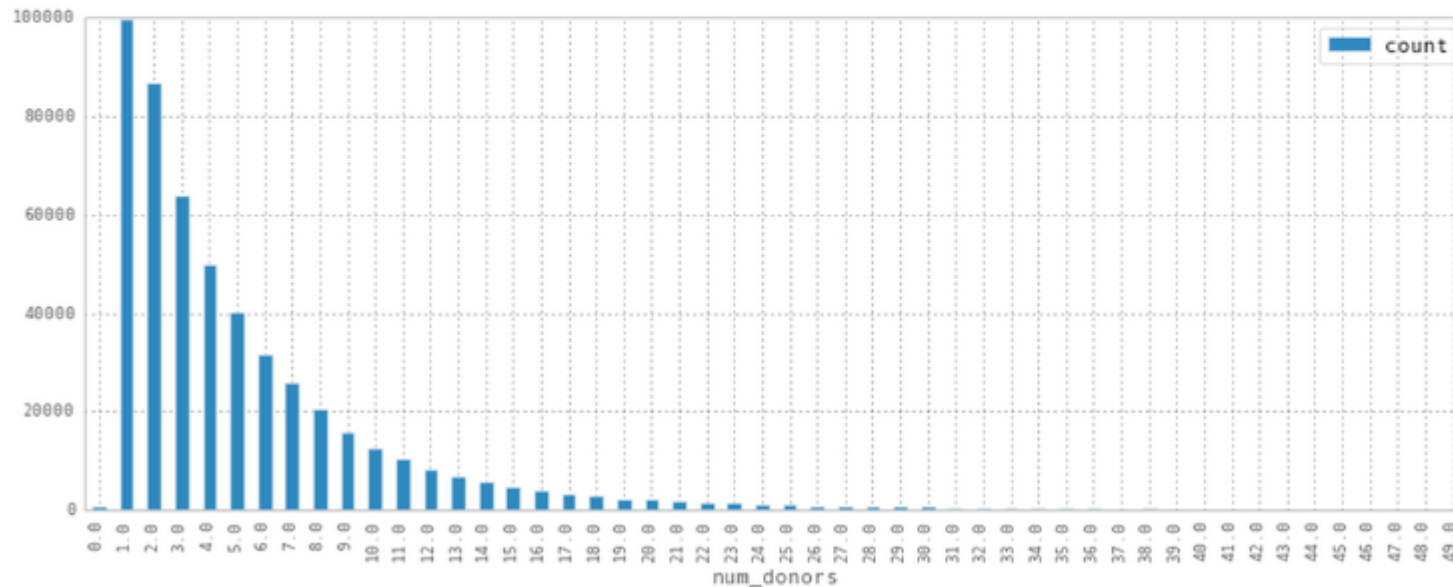


# ВИЗУАЛИЗАЦИЯ РАСПРЕДЕЛЕНИЙ

```
def spark_histogram(df, column):  
    donor_counts = df.groupby(column).count()  
    donor_df = donor_counts.toPandas()  
    donor_df[column] = donor_df.num_donors.astype(float)  
    return donor_df.sort(column).set_index(column).iloc[:50,:].plot(kind='bar', figsize=(14,5))
```

```
spark_histogram(df_complete, 'num_donors')
```

<matplotlib.axes.\_subplots.AxesSubplot at 0x113b2be50>





# ВЗАИМОДЕЙСТВИЕ ДАННЫХ

- `dataframe.crosstab()`
- `dataframe.corr()`

```
df_no_null.stat.corr('total_price_excluding_optional_support', 'num_donors')  
0.007004254706419042
```

```
df_no_null.stat.corr('total_price_excluding_optional_support', 'students_reached')  
0.0006159991686679948
```

```
df_no_null.stat.corr('total_price_excluding_optional_support', 'total_price_including_optional_support')  
0.9999972199123168
```

# ВЗАИМОДЕЙСТВИЕ ДАННЫХ

```
df_dates.crosstab('resource_type', 'funding_status').show()  
df_dates.crosstab('primary_focus_area', 'resource_type').show()
```

resource_type_funding_status	live	completed	reallocated	expired
null	2	28	0	18
Other	4542	54610	747	22550
Books	5982	118810	1527	34554
Visitors	102	806	6	341
Supplies	11939	185870	2602	63406
Trips	347	4381	62	1474
Technology	18957	150500	2256	85510

primary_focus_area_resource_type	Trips	Visitors	Other	Technology	Books	Supplies	null
Literacy & Language	630	228	32795	109605	127282	75924	4
null	0	0	0	0	1	0	41
Applied Learning	1197	104	9429	17869	4863	22596	0
Math & Science	1902	323	16353	75189	11746	89101	3
Music & The Arts	947	441	8305	19289	2883	37804	0
Health & Sports	159	54	4633	3054	432	12970	0
Special Needs	241	32	7636	19359	4112	17151	0
History & Civics	1188	73	3298	12858	9554	8271	0

# PERFORMANCE



# МОНИТОРИМ И ОПТИМИЗИРУЕМ

Данные: применяем сериализацию и кэширование

Следим за структурами данных, кэшем,  
количеством шаффов

Запомним: parallelism + memory + GC

# МОНИТОРИМ И ОПТИМИЗИРУЕМ

Данные: применяем сериализацию и кэширование

Следим за структурами данных, кэшем,  
количеством шаффов

Запомним: parallelism + memory + GC

# МОНИТОРИМ И ОПТИМИЗИРУЕМ

`.coalesce()` — когда очень много партиций

Много задач заканчиваются быстро, но есть  
несколько медленных

Много задач в очереди

<100мс на задачу

# МОНИТОРИМ И ОПТИМИЗИРУЕМ

`.repartition()` – когда очень мало партиций

Не эффекта от параллелизма

Данные очень смещены  
(смотрим за skew в данных)

# МОНИТОРИМ И ОПТИМИЗИРУЕМ





# МОНИТОРИМ И ОПТИМИЗИРУЕМ

Storage - Cache

Environment - Configuration

Executors – Workers

# МОНИТОРИМ И ОПТИМИЗИРУЕМ

Всегда старайтесь уменьшить данные

- `aggregateByKey()`
- `filter()`

# МОНИТОРИМ И ОПТИМИЗИРУЕМ

Остерегайтесь shuffle  
Сделайте заранее партиционирование и .persist()

`partitionBy()`



- `join()`
- `reduceByKey()`
- `sortByKey()`

# МОНИТОРИМ И ОПТИМИЗИРУЕМ

Ищите лучшие варианты

`groupByKey().mapValues()`



`reduceByKey()`

# УРОВНИ КЭША

MEMORY\_ONLY

В ОЗУ, как RDD объект в JVM

MEMORY\_AND\_DISK

В ОЗУ, как RDD объект в JVM  
(но на диск помещается то, что не влезло в ОЗУ)

DISK\_ONLY

RDD партции хранятся на диске, без участия ОЗУ

\*\_SER / \*\_N

Повторяет уровень хранения, но объект становится сериализованный (в байт коде) (более эффективно по памяти, но появляется зависимость процессора /

Повторяет уровень хранения, повторяет (реплицирует) на N количестве нод

# ЗАПОМНИМ

НЕЛЬЗЯ ИЗМЕРИТЬ – НЕЛЬЗЯ УСКОРИТЬ

ДЕЛАЙТЕ РЕВЬЮ СВОЕГО КОДА (ИЗБЕГАЙТЕ ЦИКЛОВ)

НАБОРЫ ДАННЫХ МОГУТ БЫТЬ ОПТИМИЗИРОВАННЫ  
ПАРТИЦИОНИРОВАНИЕМ

САМАЯ ДОРОГАЯ ОПЕРАЦИЯ SHUFFLE

# SPARK ПРОЕКТ



# SPARK ПРОЕКТ

- Цель: Разработать Data Quality «платформу» на Apache Spark

Задача: Выбрать данные

(<https://cseweb.ucsd.edu/~jmcauley/datasets.html>)