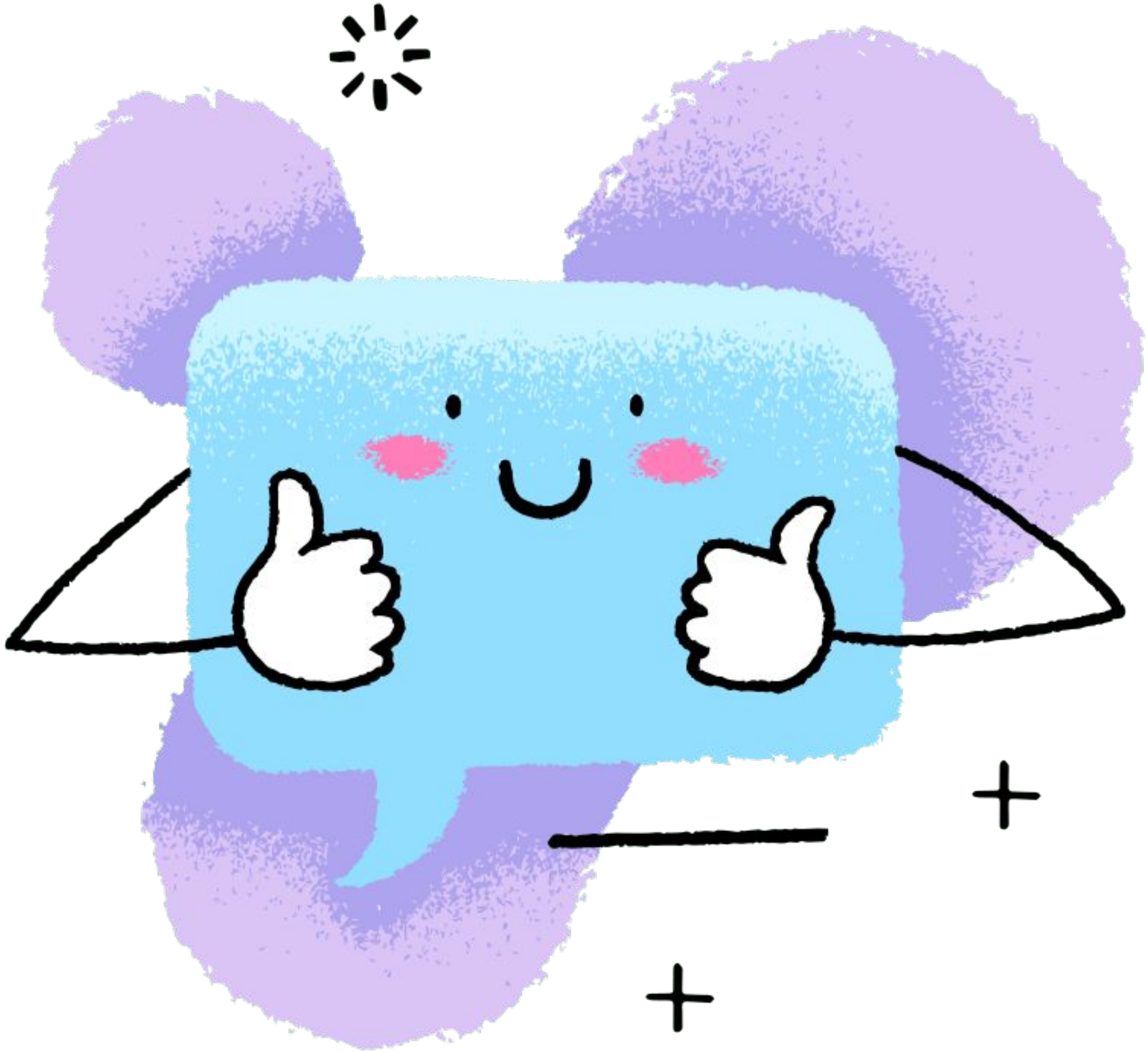
PySpark



GeekBrains

PySpark DataFrame API

Apache Spark

Что будет на уроке

1. RDD: Resilent Data Distribution. Класс DataFrame.
2. План запроса. Перемешивание данных (шаффл). Сохранение данных.
3. groupBy, join
4. Получение плана выполнения запроса.
5. Способы оптимизации: push down фильтрация, стратегия исполнения join.
6. Работа со Spark UI
7. Сложные случаи: перекос в объеме данных между экзекьюторами. Функции repartition, coalesce
8. Агрегирующие функции.
9. Агрегирование по синтетическому ключу.
10. Pivot

RDD

низкоуровневый API для работы с распределенными данными

```python
# In Python
# Create an RDD of tuples (name, age)
dataRDD = sc.parallelize([("Brooke", 20), ("Denny", 31), ("Jules", 30),
("TD", 35), ("Brooke", 25)])
# Use map and reduceByKey transformations with their lambda
# expressions to aggregate and then compute average
agesRDD = (dataRDD
.map(lambda x: (x[0], (x[1], 1)))
.reduceByKey(lambda x, y: (x[0] + y[0], x[1] + y[1]))
.map(lambda x: (x[0], x[1][0]/x[1][1])))
```

DataFrame

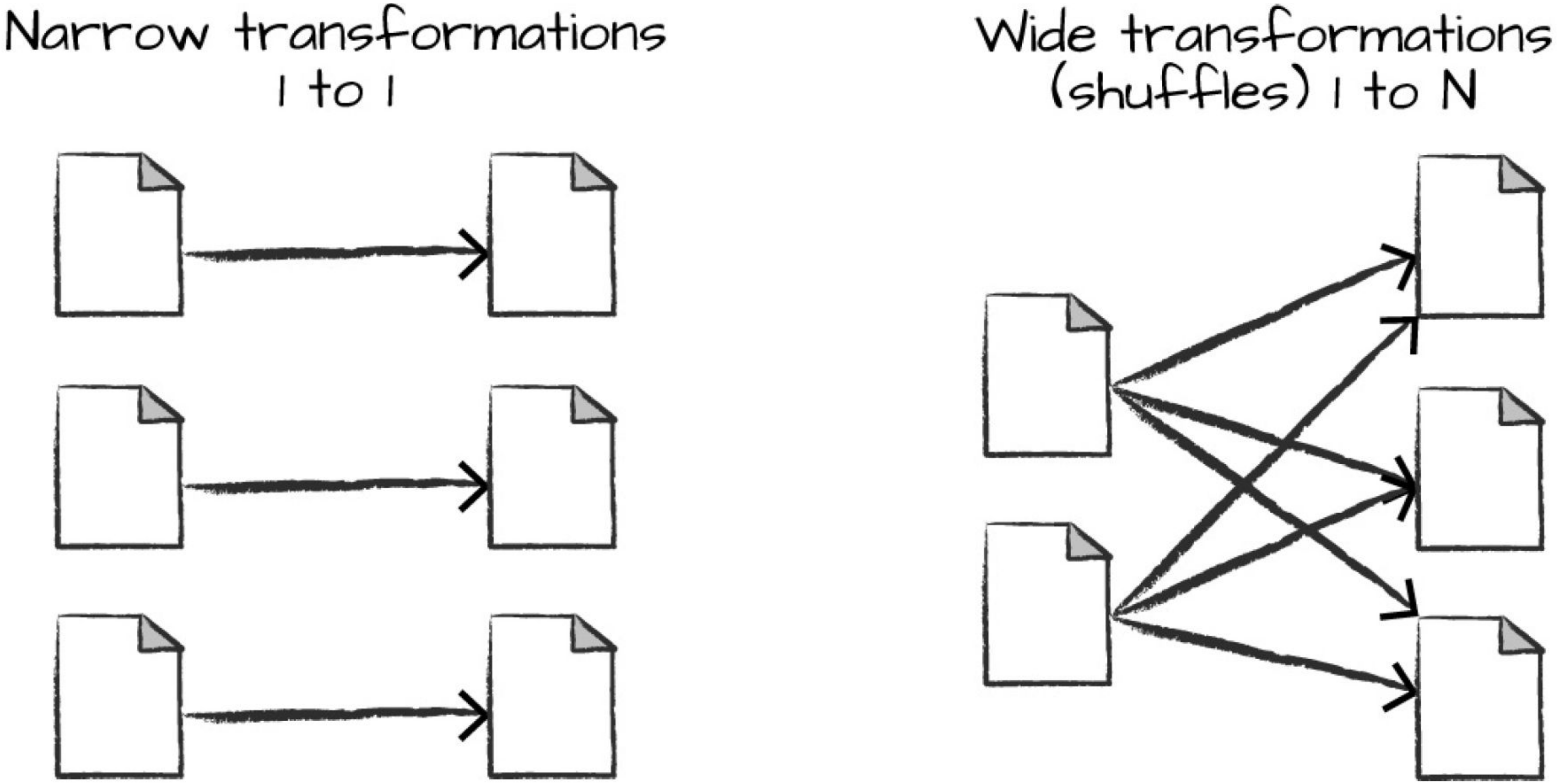
Начиная со Spark 1.6 обобщенный API для работы с RDD

```
+------+--------+
|  name|avg(age)|
+------+--------+
|Brooke|    22.5|
| Jules|    30.0|
|    TD|    35.0|
| Denny|    31.0|
+------+--------+
```

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import avg
# Create a DataFrame using SparkSession
spark = (SparkSession
.builder
.appName("AuthorsAges")
.getOrCreate())
# Create a DataFrame
data_df = spark.createDataFrame([("Brooke", 20), ("Denny", 31), ("Jules", 30),("TD", 35), ("Brooke", 25)], ["name", "age"])
# Group the same names together, aggregate their ages, and compute an average
avg_df = data_df.groupBy("name").agg(avg("age"))
# Show the results of the final execution
avg_df.show()
```

Transformation types

narrow -> in-memory filters
wide -> shuffle, write result on disk



GeekBrains

Transformations and actions as Spark operations

Трансформации копятся до вызова action

Transformations	Actions
orderBy()	show()
groupBy()	take()
filter()	count()
select()	collect()
join()	save()

Spark DataFrame

таблица с типизированными колонками

Id (Int)	First (String)	Last (String)	Url (String)	Published (Date)	Hits (Int)	Campaigns (List[Strings])
1	Jules	Damji	https://tinyurl.1	1/4/2016	4535	[twitter, LinkedIn]
2	Brooke	Wenig	https://tinyurl.2	5/5/2018	8908	[twitter, LinkedIn]
3	Denny	Lee	https://tinyurl.3	6/7/2019	7659	[web, twitter, FB, LinkedIn]
4	Tathagata	Das	https://tinyurl.4	5/12/2018	10568	[twitter, FB]

Основные типы данных Python
значения в колонках

Data type	Value assigned in Python	API to instantiate
ByteType	int	DataTypes.ByteType
ShortType	int	DataTypes.ShortType
IntegerType	int	DataTypes.IntegerType
LongType	int	DataTypes.LongType
FloatType	float	DataTypes.FloatType
DoubleType	float	DataTypes.DoubleType
StringType	str	DataTypes.StringType
BooleanType	bool	DataTypes.BooleanType
DecimalType	decimal.Decimal	DecimalType

Python structured data types in Spark

Data type	Value assigned in Python	API to instantiate
`BinaryType`	`bytearray`	`BinaryType()`
`TimestampType`	`datetime.datetime`	`TimestampType()`
`DateType`	`datetime.date`	`DateType()`
`ArrayType`	List, tuple, or array	`ArrayType(dataType, [nullable])`
`MapType`	`dict`	`MapType(keyType, valueType, [nullable])`
`StructType`	List or tuple	`StructType([fields])`
`StructField`	A value type corresponding to the type of this field	`StructField(name, dataType, [nullable])`

Задание schema
и зачем это нужно

- Избегаем приведение типов
- Не нужно считывать файл только для определения схемы данных (существенно для больших файлов)
- Можно отловить несоответствие данных

```python
from pyspark.sql.types import *
schema = StructType([StructField("author", StringType(), False),
StructField("title", StringType(), False),
StructField("pages", IntegerType(), False)])

#the same schema using DDL
schema = "author STRING, title STRING, pages INT"
```

Создаем статический DataFrame

```python
from pyspark.sql import SparkSession
# Define schema for our data using DDL
schema = "`Id` INT, `First` STRING, `Last` STRING, `Url` STRING,
`Published` STRING, `Hits` INT, `Campaigns` ARRAY<STRING>"
# Create our static data
data = [
[1, "Jules", "Damji", "https://tinyurl.1", "1/4/2016", 4535, ["twitter","LinkedIn"]],
[2, "Brooke","Wenig", "https://tinyurl.2", "5/5/2018", 8908, ["twitter","LinkedIn"]]
]

# Create a DataFrame using the schema defined above
blogs_df = spark.createDataFrame(data, schema)
# Show the DataFrame; it should reflect our table above
blogs_df.show()
# Print the schema used by Spark to process the DataFrame
print(blogs_df.printSchema())
```

Count, countDistinct

агрегирующие функции могут вызываться по всему набору данных

```
from pyspark.sql.functions import count
df.select(count("StockCode")).show()

from pyspark.sql.functions import countDistinct
df.select(countDistinct("StockCode")).show()

from pyspark.sql.functions import approx_count_distinct
df.select(approx_count_distinct("StockCode", 0.1)).show()
```

Column alias

меняем название колонки

```
from pyspark.sql.functions import sum, count, avg, expr
df.select(
  count("Quantity").alias("total_transactions"),
  sum("Quantity").alias("total_purchases"),
  avg("Quantity").alias("avg_purchases"),
  expr("mean(Quantity)").alias("mean_purchases"))\
  .selectExpr(
  "total_purchases/total_transactions",
  "avg_purchases",
  "mean_purchases")\
.show()
```

Считываем данные
для дальнейших упражнений

```python
# in Python
df = spark.read.format("csv")\
.option("header", "true")\
.option("inferSchema", "true")\
.load("/data/retail-data/by-day/2010-12-01.csv")
df.printSchema()
df.createOrReplaceTempView("dfTable")
```

Booleans

Фильтрация по значению в колонке

```python
# in Python
from pyspark.sql.functions import col
df.where(col("InvoiceNo") != 536365)\
.select("InvoiceNo", "Description")\
.show(5, False)
```

```python
from pyspark.sql.functions import instr

priceFilter = col("UnitPrice") > 600
descripFilter = instr(df.Description, "POSTAGE") >= 1

df.where(df.StockCode.isin("DOT"))\
.where(priceFilter | descripFilter).show()
```

SQL для создания колонок и логических выражений через expr

```
from pyspark.sql.functions import expr
df.withColumn("isExpensive", expr("NOT UnitPrice <= 250"))\
.where("isExpensive")\
.select("Description", "UnitPrice").show(5)
```

Численные значения

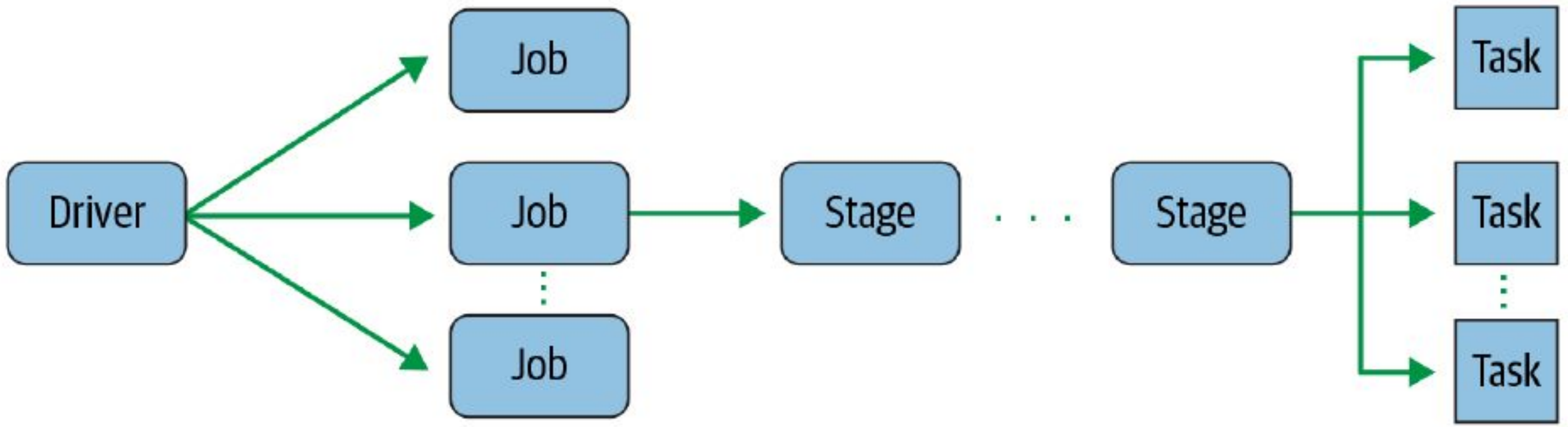
```
from pyspark.sql.functions import expr, pow
fabricatedQuantity = pow(col("Quantity") * col("UnitPrice"), 2) + 5

df.select(expr("CustomerId"), \
          fabricatedQuantity.alias("realQuantity"))\
.show(2)

# либо через SQL
df.selectExpr("CustomerId",
"(POWER((Quantity * UnitPrice), 2.0) + 5) as realQuantity")\
.show(2)
```

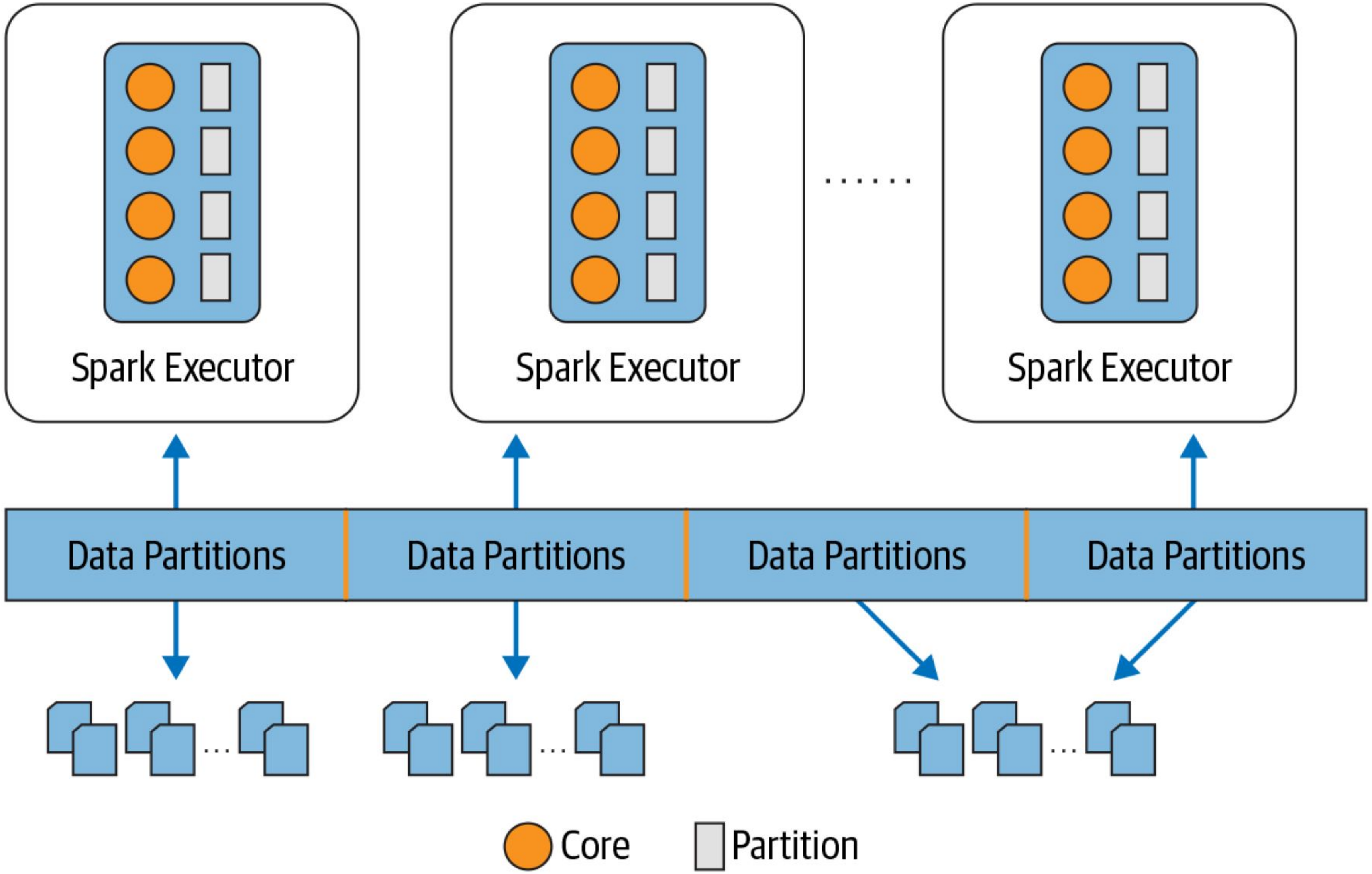
Spark jobs, stage, task

shuffle occurs between every two stages



Spark executors, partitions

Relationship of Spark tasks, cores, partitions



df.cache()

store as many of the partitions in memory across Spark executors as memory allows

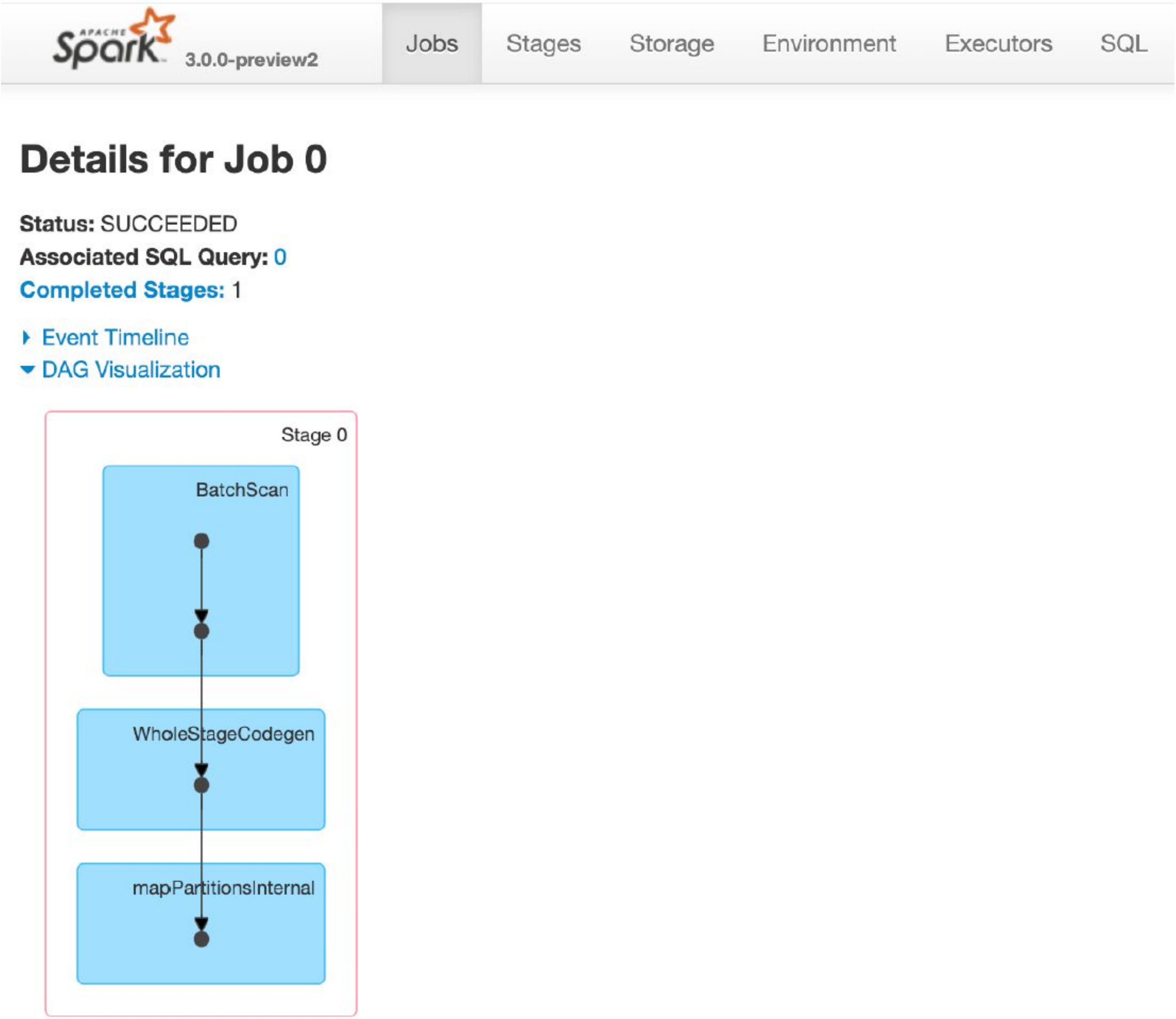
When to Cache and Persist:

- DataFrames commonly used during iterative machine learning training
- DataFrames accessed commonly for doing frequent transformations during ETL or building data pipelines

When Not to Cache and Persist:

- DataFrames that are too big to fit in memory
- An inexpensive transformation on a DataFrame not requiring frequent use, regardless of size

Spark UI
DAG, resources

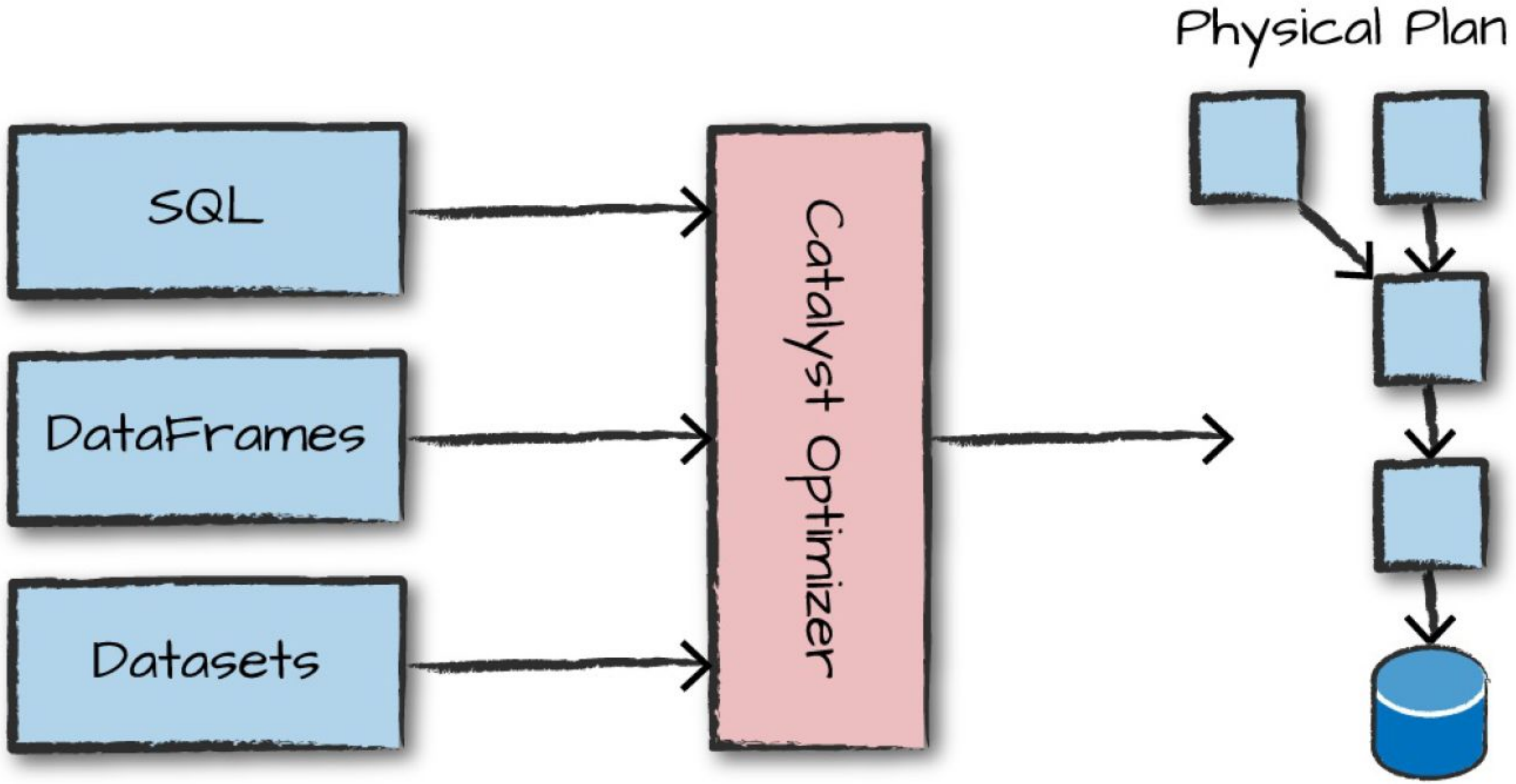


Практика

Zeppelin -> Lecture 2

Оптимизатор запросов

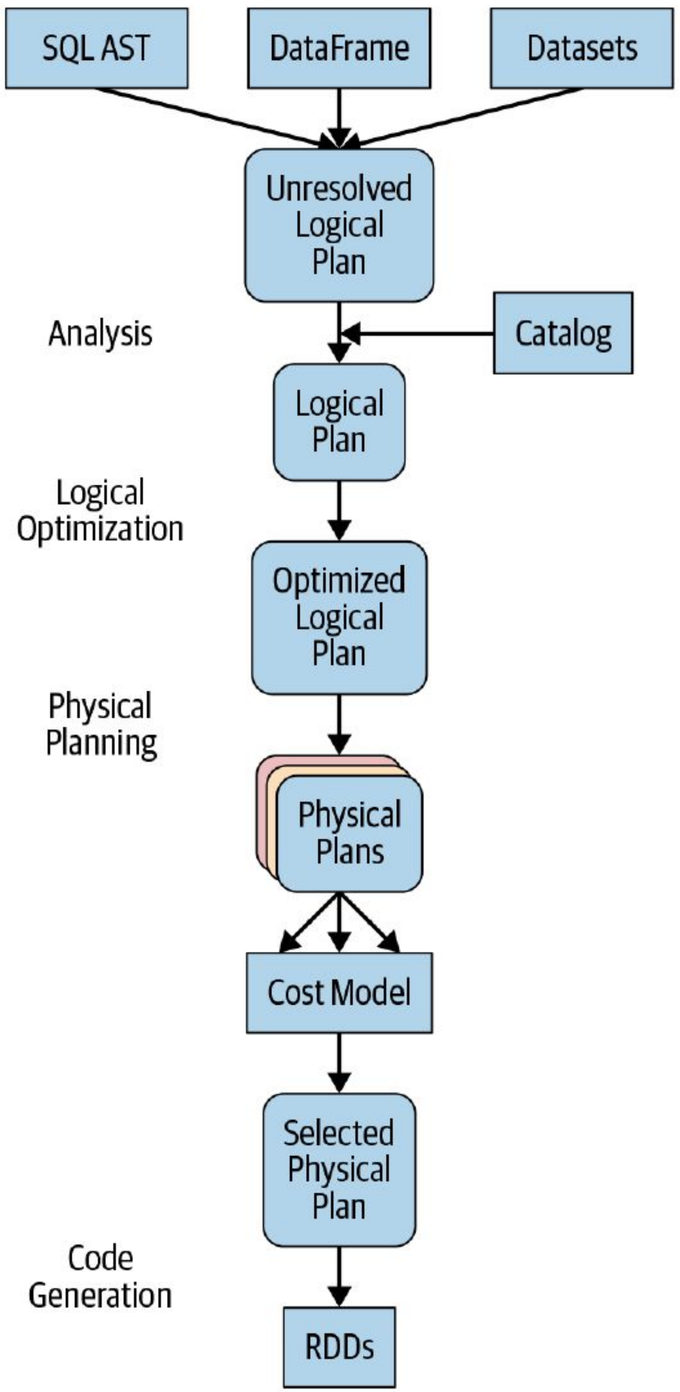
Lazy исполнение трансформаций позволяет оптимизировать план



GeekBrains

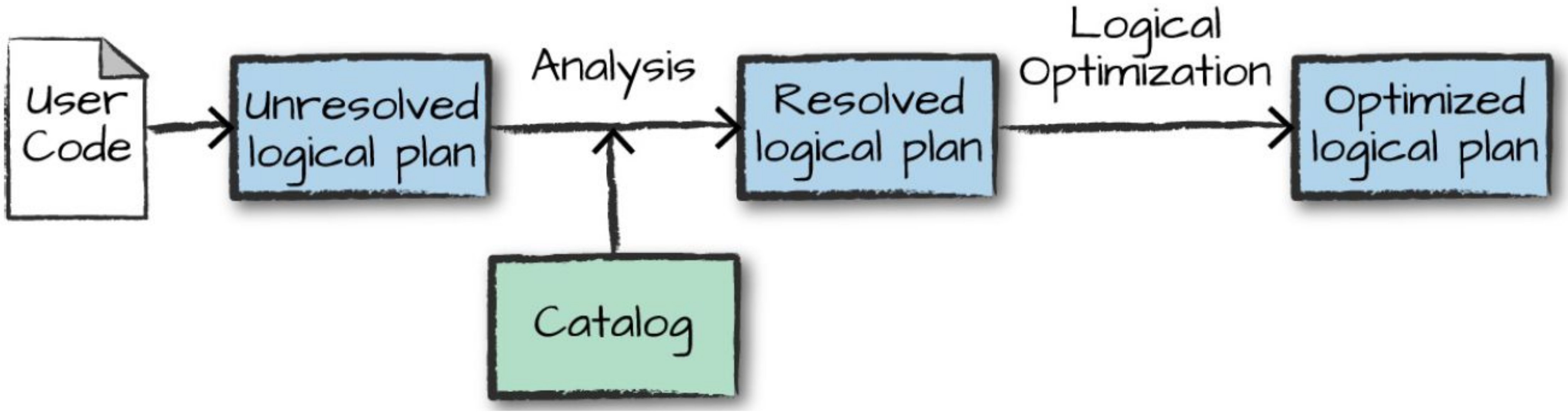
Catalyst Optimizer
transformational phases

1. Analysis
2. Logical optimization
3. Physical planning
4. Code generation



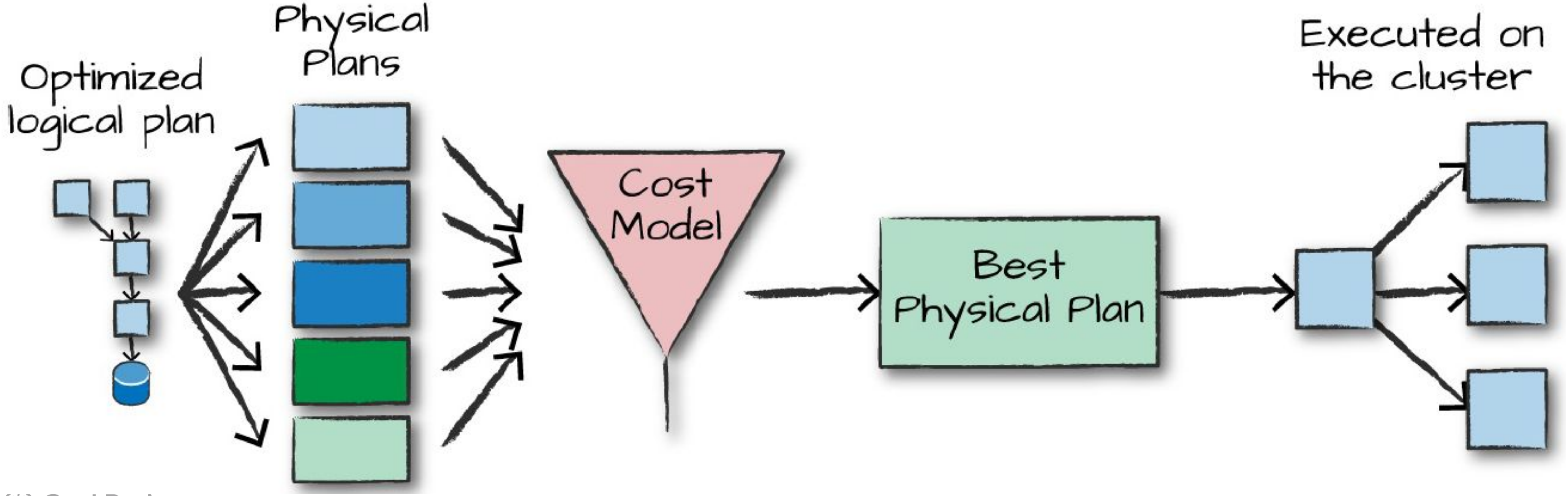
Logical plan

1) проверка наличия таблиц, колонок
2) pushing down predicates or selections

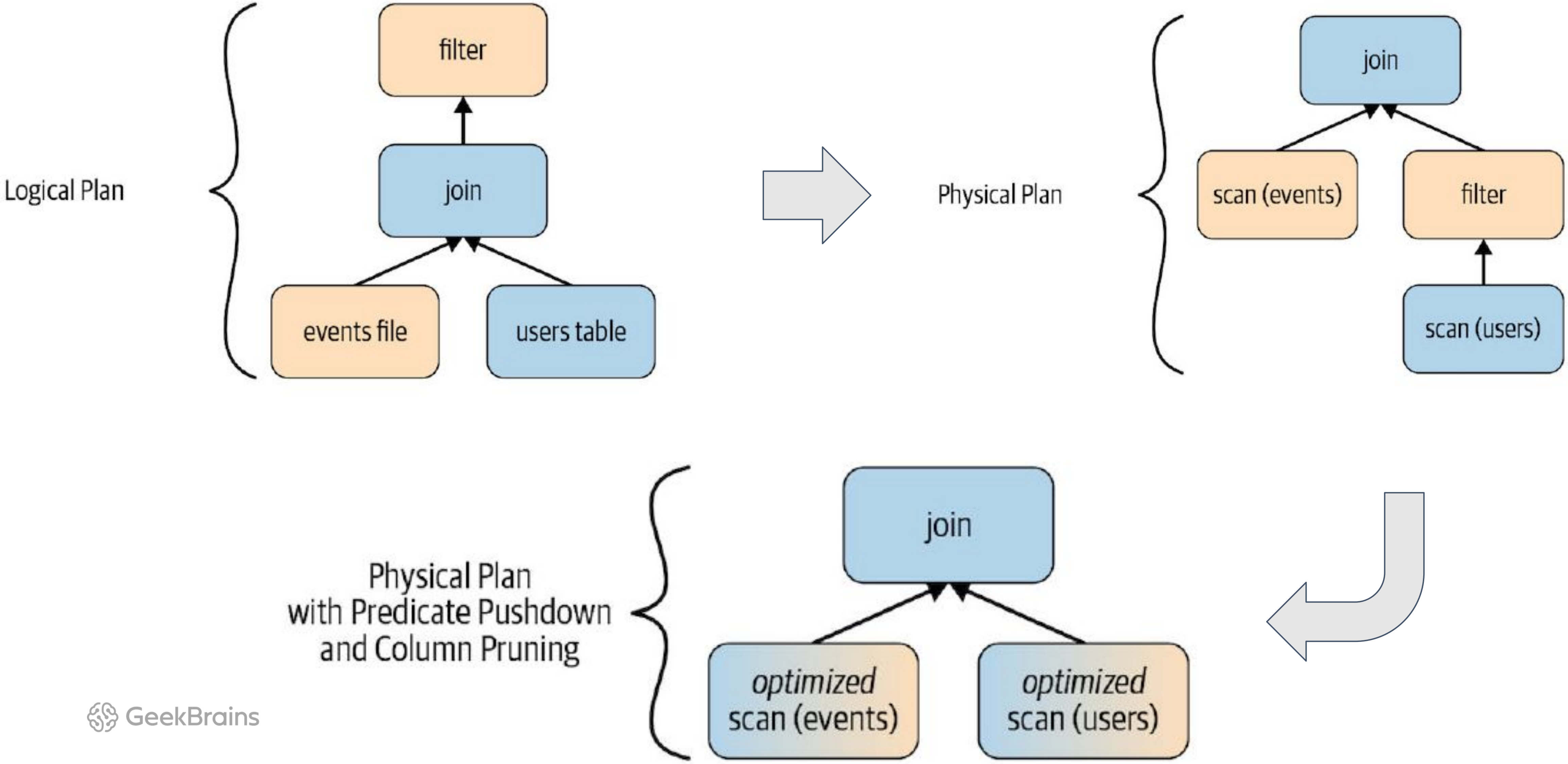


Physical Planning

Использует знание о размере и распределении партиций

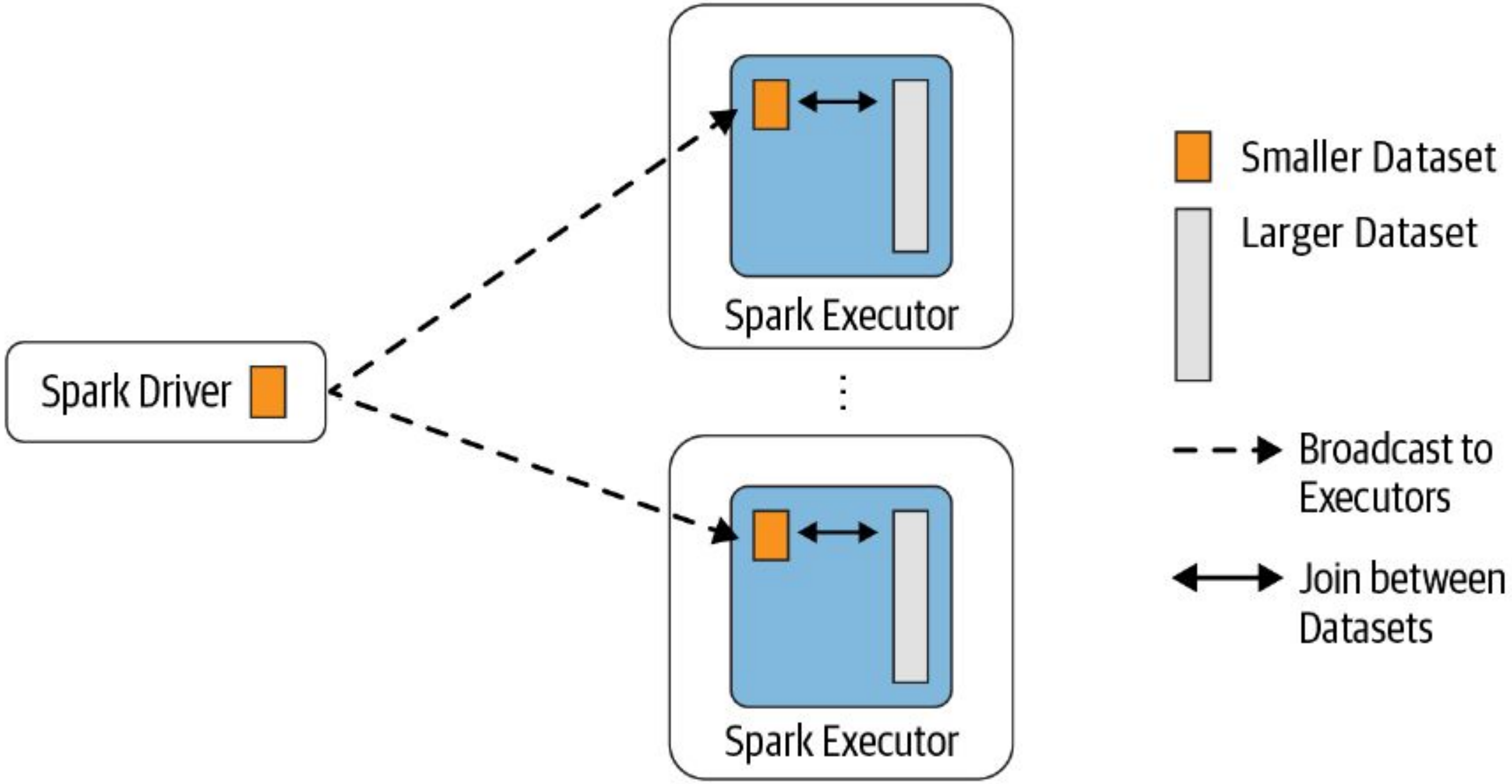


Планы запроса из примера

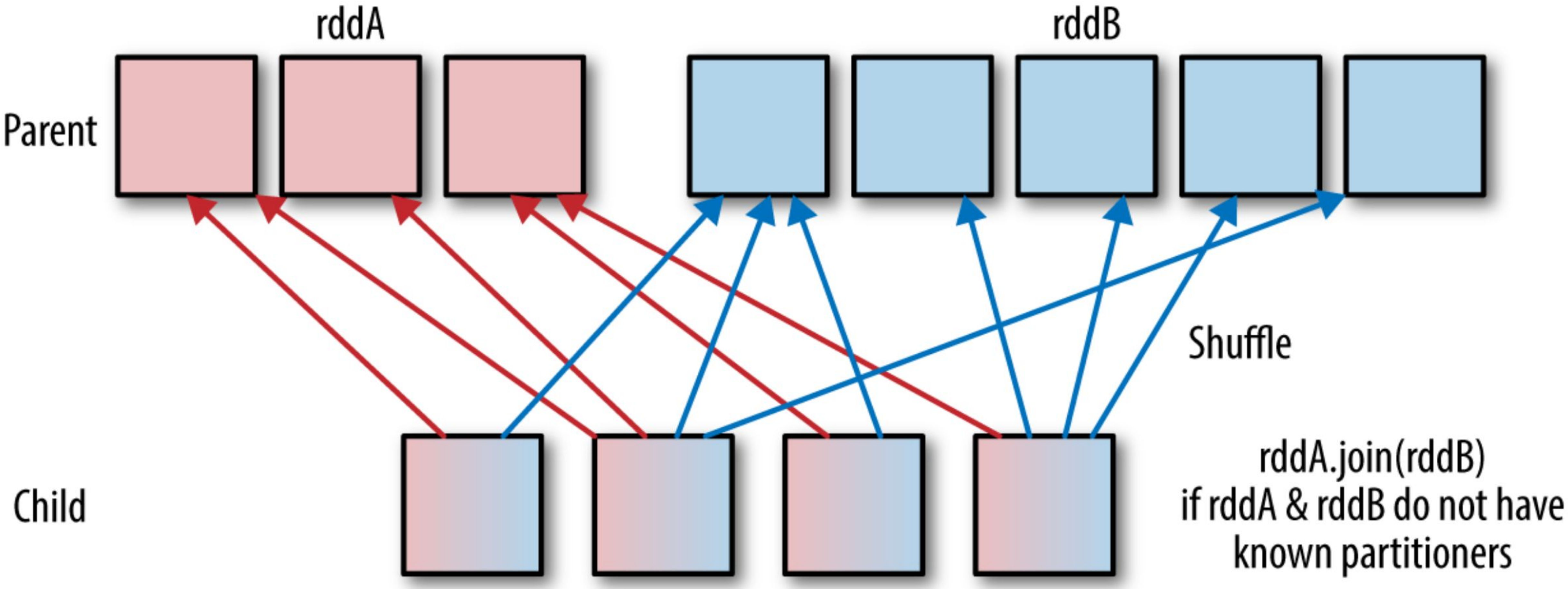


GeekBrains

Broadcast Hash Join



Shuffle Sort Merge Join



Ссылки

catalyst-optimizer

deep-dive-into-spark-sqls-catalyst-optimizer

high-performance-spark/ch04.html

towardsdatascience.com/the-art-of-joining-in-spark

https://itnext.io/handling-data-skew-in-apache-spark-9f56343e58e8

https://blog.clairvoyantsoft.com/optimize-the-skew-in-spark-e523c6ee18ac

Спасибо!

Каждый день вы становитесь лучше :)