

Modern Storages and Data Warehousing

Week 8 - Advanced Spark

Попов Илья, i.popov@hse.ru

1 - Homework Q&A

Recap прошлых занятий

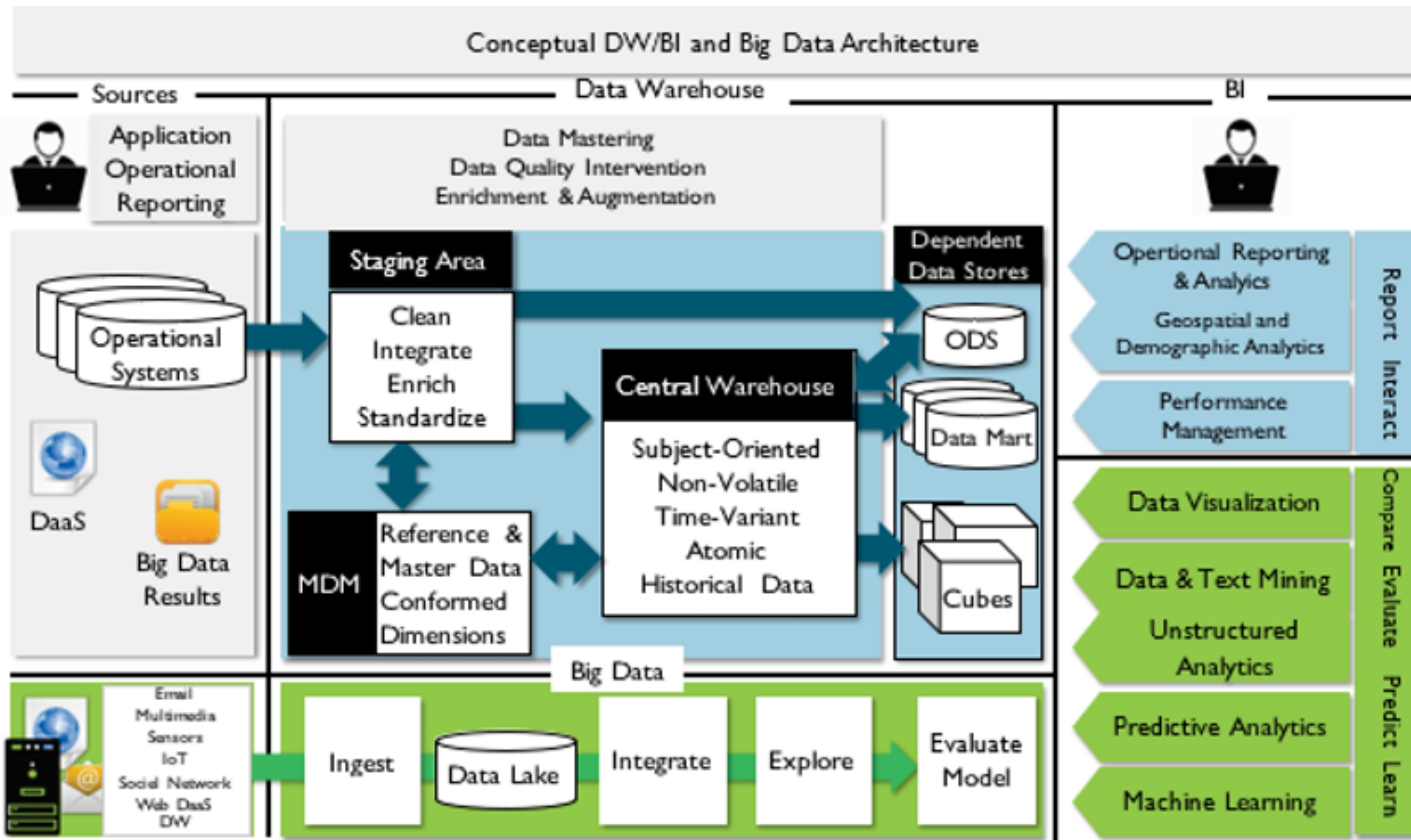


Figure 5: Date Warehouse Concept

2 - Advanced Spark

Мотивация

- › Казалось бы, Spark - штука довольно простая и дружелюбная к DA / DE
- › Spark SQL дает простое API для выполнения аналитических запросов
- › В reference pySpark всегда можно подсмотреть

| А зачем нам тогда
сегодняшнее занятие?
Расходимся?

Мотивация

- › Казалось бы, Spark - штука довольно простая и дружелюбная к DA / DE
- › Spark SQL дает простое API для выполнения аналитических запросов
- › В reference pySpark всегда можно подсмотреть

- › Наши запросы могут хорошо работать на маленьких данных, но на больших они будут работать неэффективно
- › Если мы не знаем, как устроен Spark изнутри, мы не сможем оптимизировать запросы
- › Если мы пишем регулярный процесс, его неоптимальность умножается на количество повторений

2.1 - Построение плана

Пример

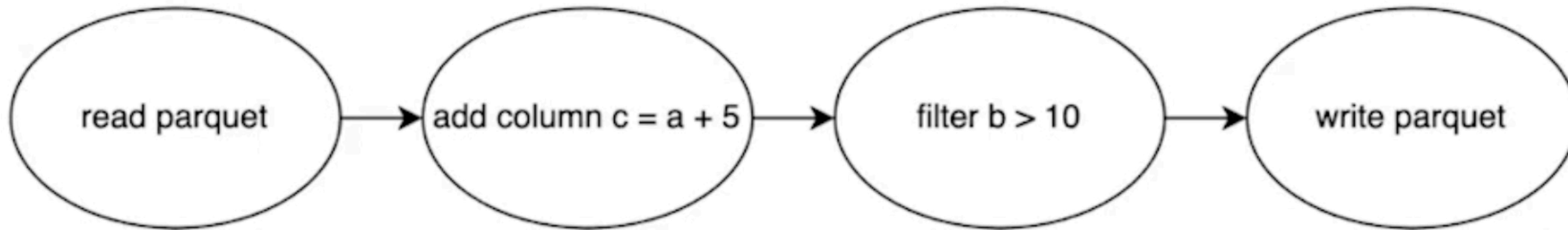
```
spark.read.parquet("/content/example_1") \
    .withColumn("c", f.col("a") + 5) \
    .filter(f.col("b") > 10) \
    .write.parquet("/content/example_1_output")
```

a	b
1	2
2	12
3	6
4	25

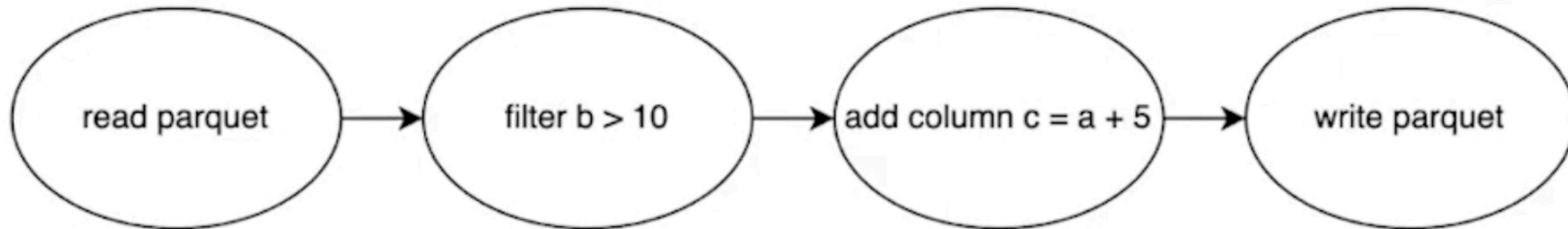
a	b	c
1	2	6
2	12	7
3	6	8
4	25	9

a	b	c
2	12	7
4	25	9

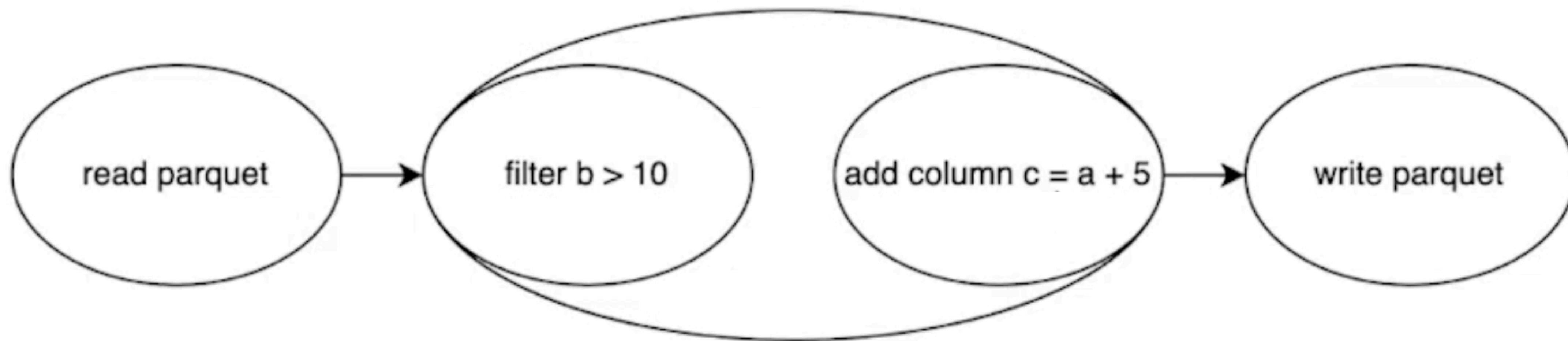
План запроса



План запроса



План запроса



Демо

Ленивые трансформации

- › Если нет write / другого вывода - Spark пристраивает только логический план
- › Оптимизация логического плана в физический происходит на выводе
- › Такие штуки называются ленивыми трансформациями

```
spark.read.parquet("/content/example_1") \
    .withColumn("c", f.col("a") + 5) \
    .filter(f.col("b") > 10)
```

К чему это приводит

```
df = spark.read.parquet("/content/example_1") \n    .withColumn("c", f.col("a") + 5)
```

```
df.filter(f.col("b") > 10).write.parquet("/content/example_2_output_1")
```

```
df.filter(f.col("b") <= 10).write.parquet("/content/example_2_output_2")
```

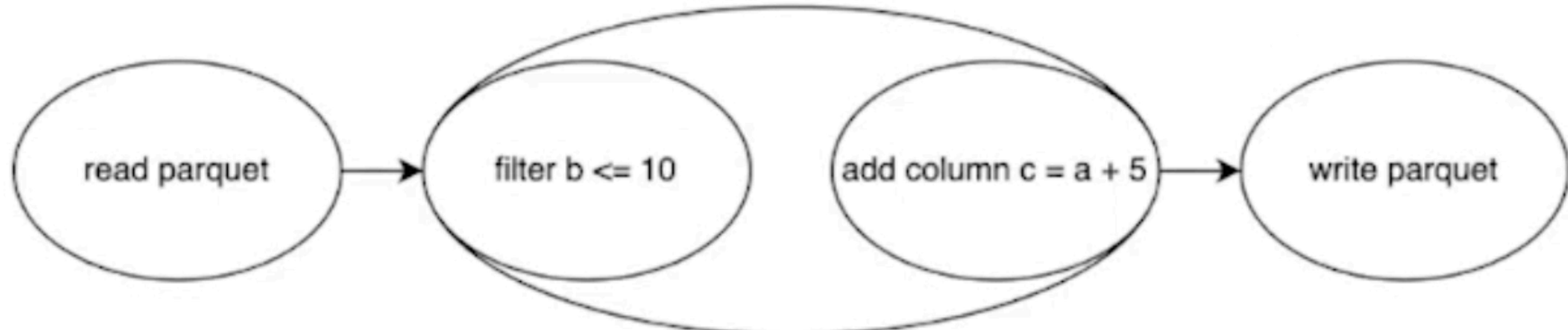
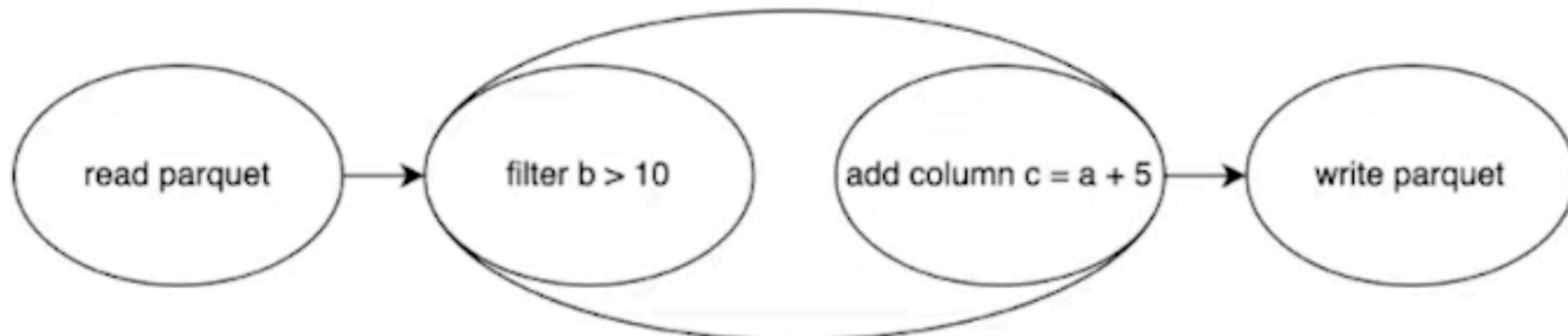
a	b
1	2
2	12
3	6
4	25

a	b	c
1	2	6
2	12	7
3	6	8
4	25	9

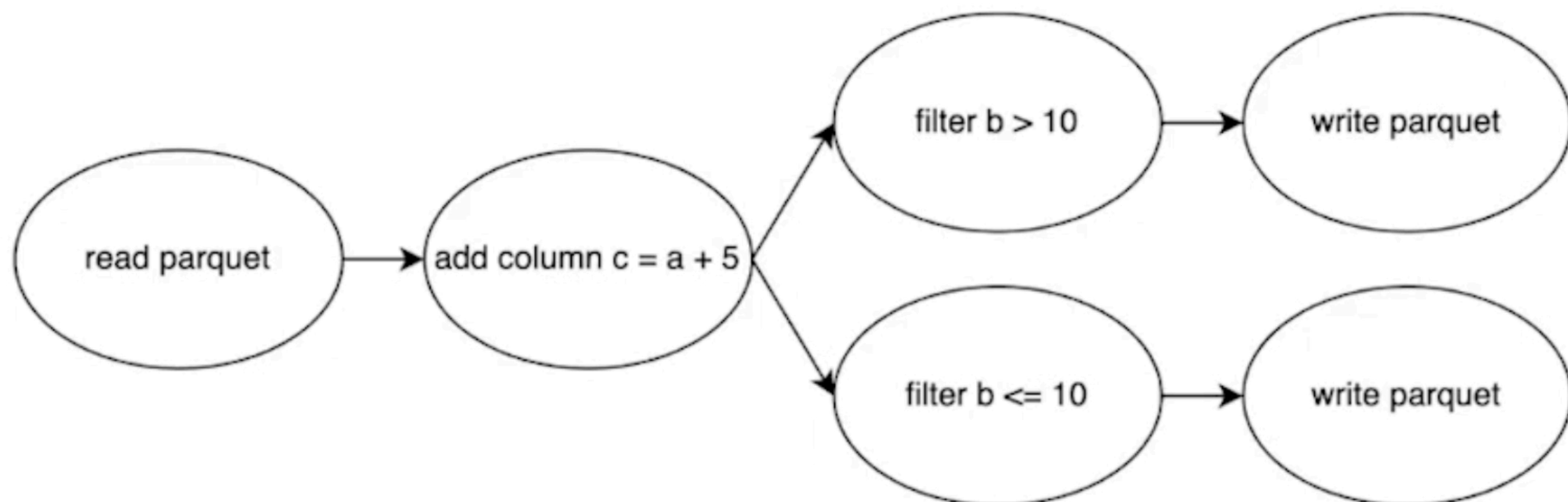
a	b	c
2	12	7
4	25	9

a	b	c
1	2	6
3	6	8

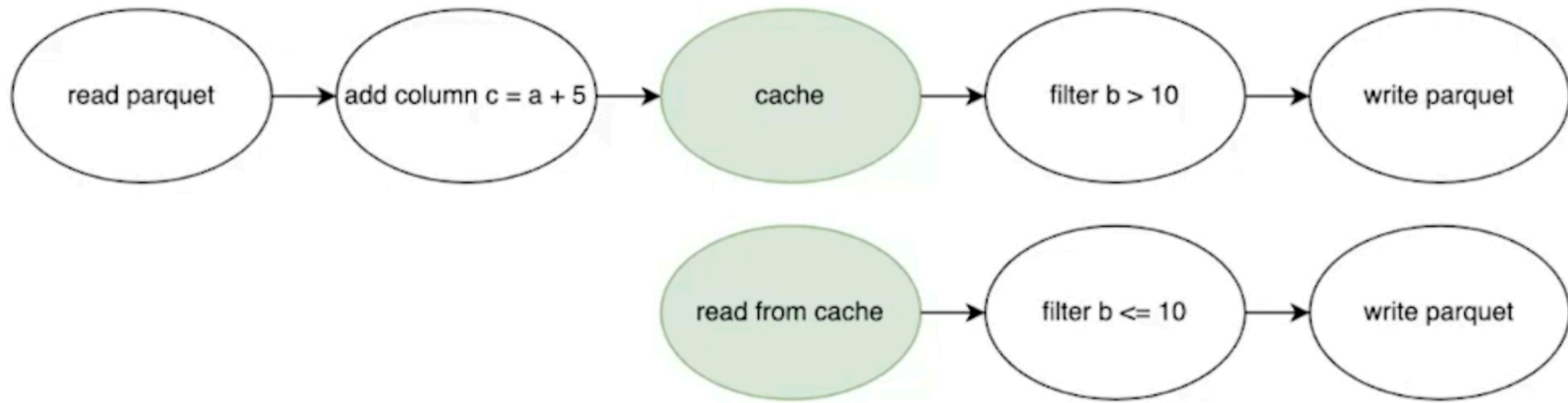
К чему это приводит



А что хочется видеть



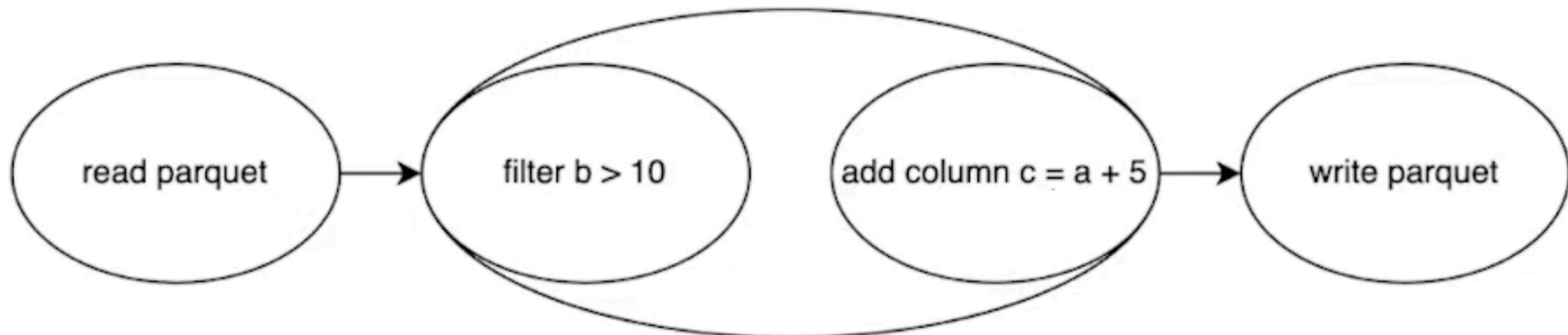
Как заставить его это сделать



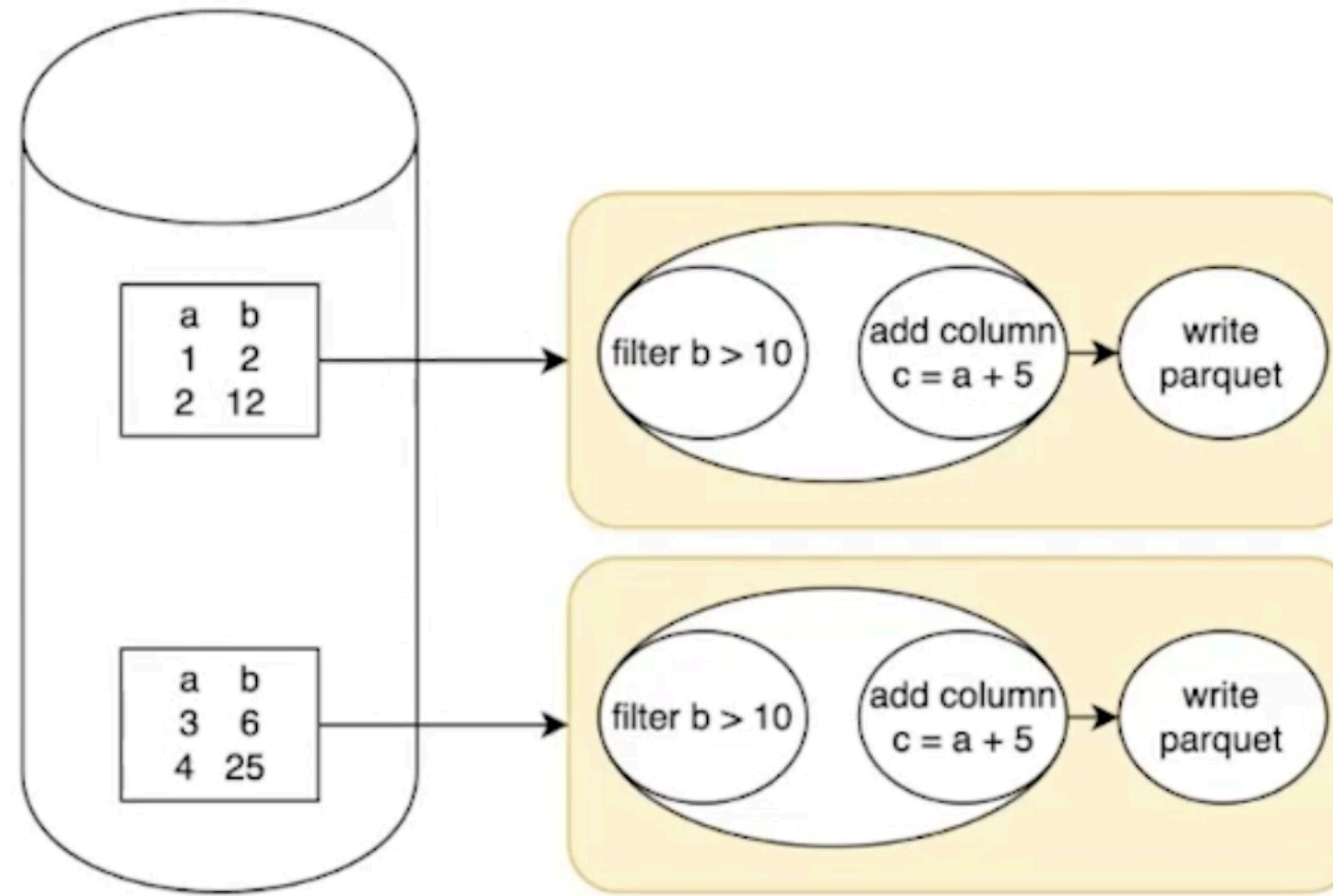
2.2 - Executors

Мотивация

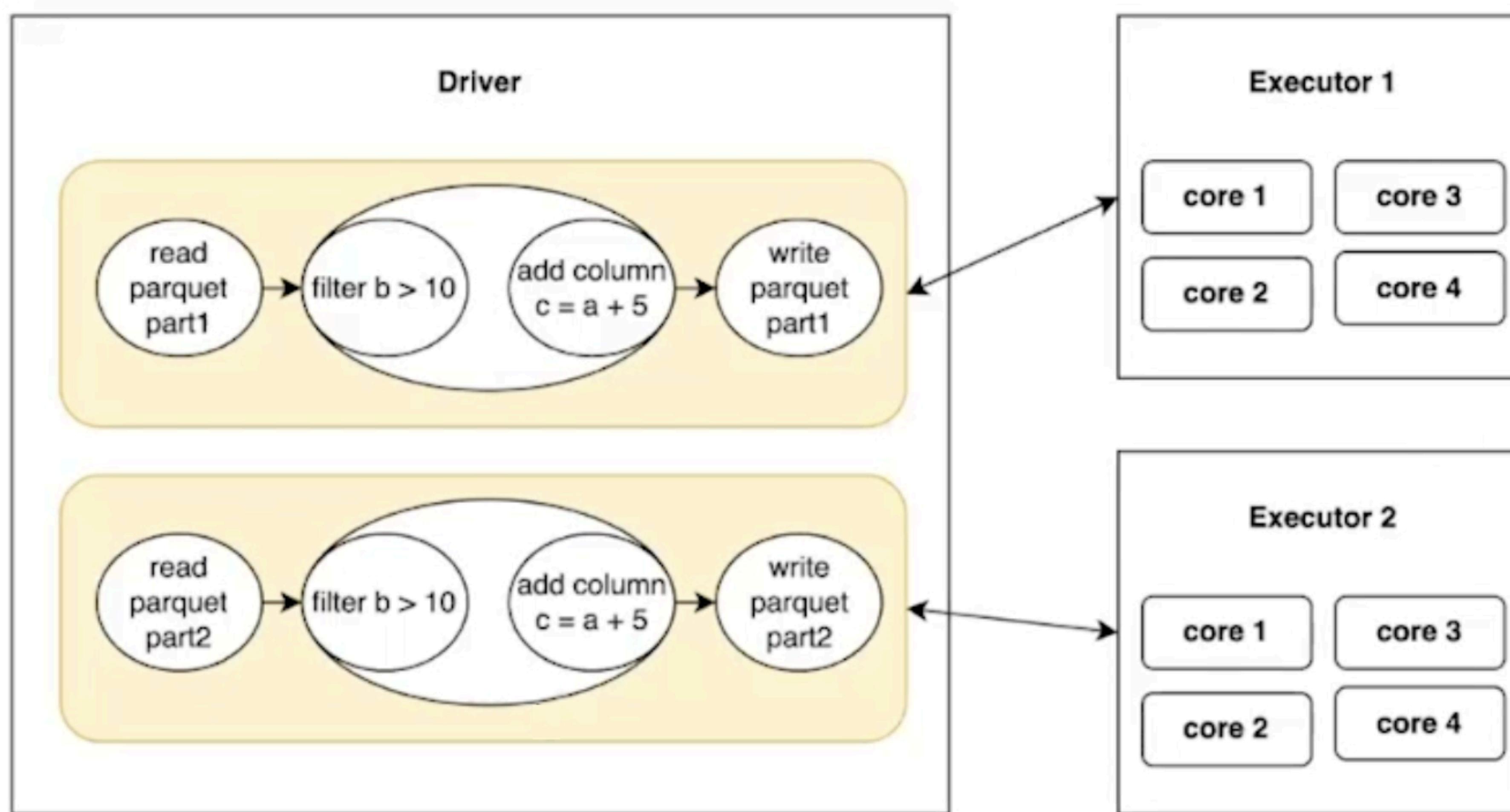
- › Все, что мы делали с вами, мы делали на Driver
- › Мы делали оптимизации не смотря на сами данные, нам достаточно было знать только их природу



Исполнение запроса



Исполнение запроса



Демо

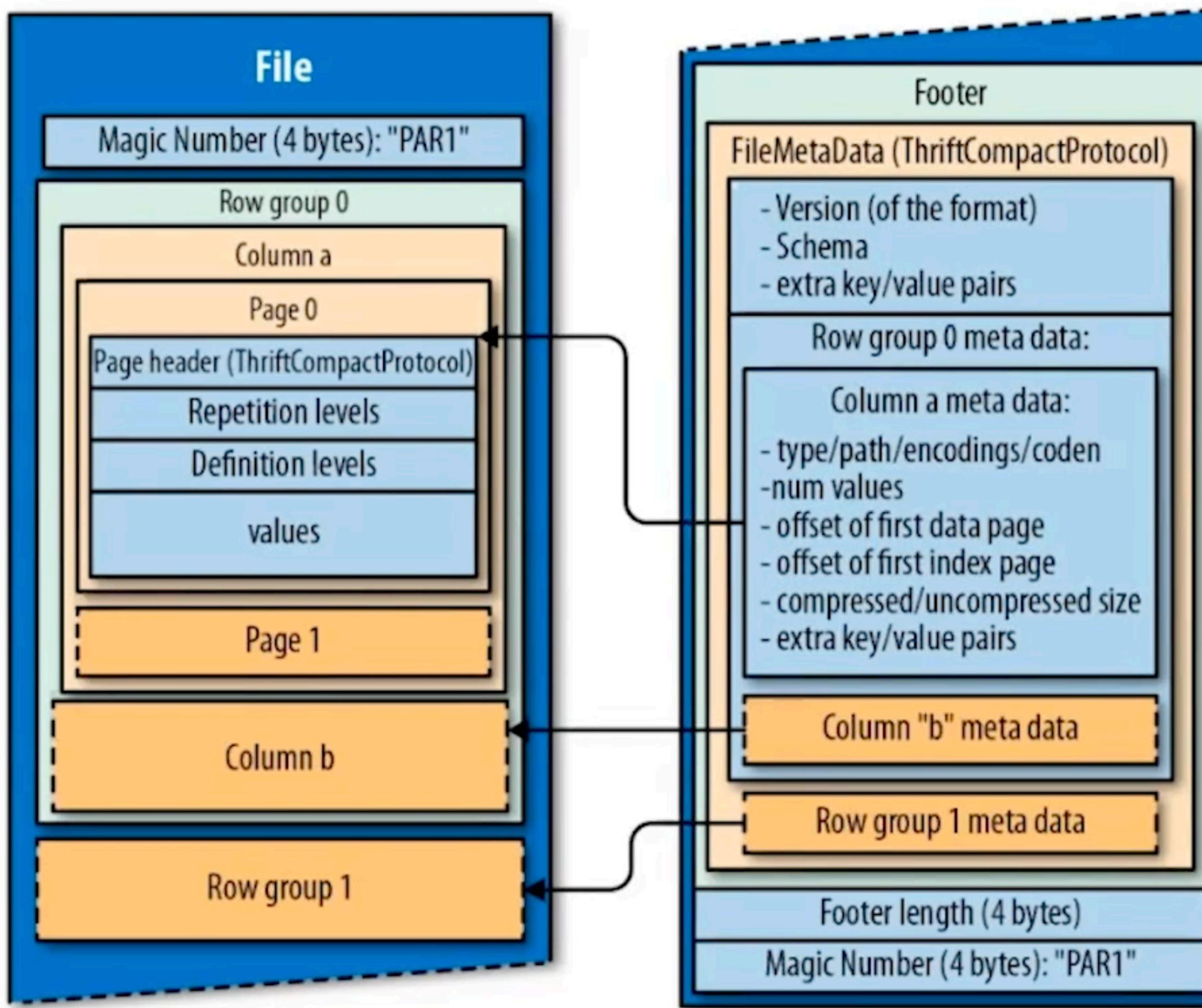
2.3 - Чтение parquet

Схема данных

```
1 spark.read.parquet('/content/example_1.parquet')
```

```
DataFrame[a: bigint, b: bigint]
```

Схема данных



a1	b1	c1	d1
a2	b2	c2	d2
a3	b3	c3	d3
a4	b4	c4	d4
a5	b5	c5	d5
a6	b6	c6	d6

Чтение плана

- › Driver запомнил location и inMemoryFileIndex
Location: InMemoryFileIndex(1 paths)[file:/content/example_1.parquet]
- › Driver запомнил ReadSchema для колонок, которые мы будем читать
ReadSchema: struct<a:bigint,b:bigint>

Чтение плана

- › Driver запомнил location и inMemoryFileIndex
Location: InMemoryFileIndex(1 paths)[file:/content/example_1.parquet]
- › Driver запомнил ReadSchema для колонок, которые мы будем читать
ReadSchema: struct<a:bigint,b:bigint>

Когда запрос прошел, добавились еще интересности:

- › PushedFilters: [IsNotNull(b), GreaterThan(b,10)]

2.4 - Group by

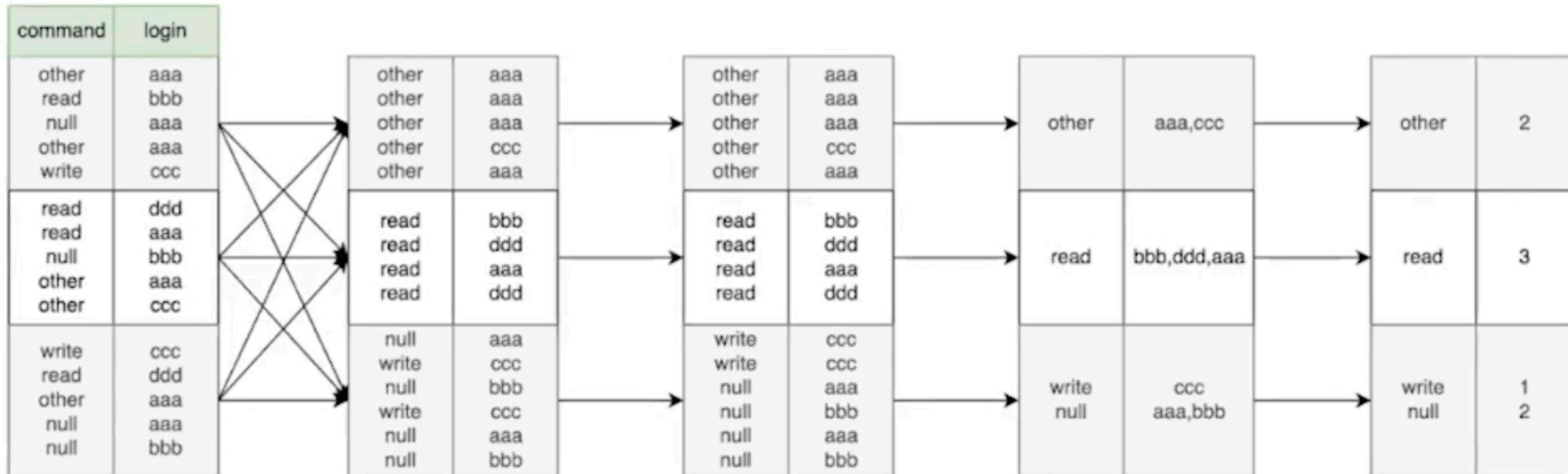
Пример

	command	login
0	read	4b1a5d2b-f1a4-43e4-96c0-1f0b5ad5e8dd
1	write	68aac3a5-4e81-4608-a25c-a32d052dbe2e
2	write	ccc8919c-33a4-4c2a-aafa-5987890e894f
3	write	fa535f76-5580-4c50-b749-86d70416631e
4	write	7eb6cd13-ff9d-49f5-b6cd-35e57d7bdc92
5	read	9a67f06a-6daa-4415-b82c-af79bb7ae8c6
6	read	3dad7cd0-aa8f-4bde-8af3-a9573e2f4fff
7	write	fb3ec781-698d-49c9-af7a-dd4edef7822e
8	read	7eb6cd13-ff9d-49f5-b6cd-35e57d7bdc92
9	other	4b1a5d2b-f1a4-43e4-96c0-1f0b5ad5e8dd

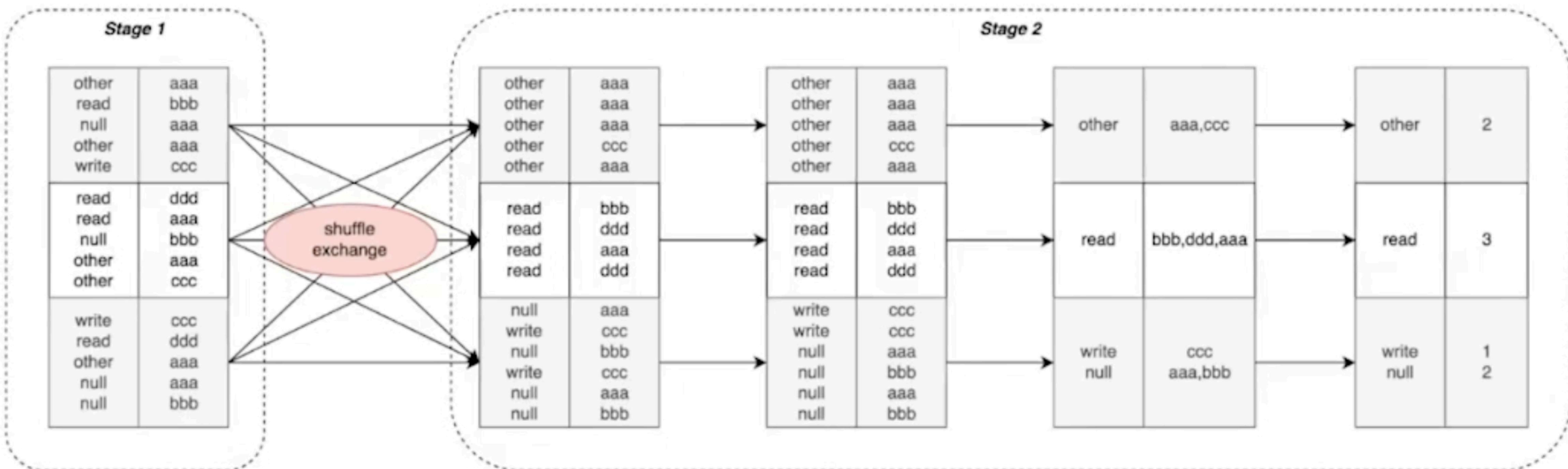
```
1 df = spark.read.parquet('/content/example_4')
2
3 df.groupby('command') \
4   .agg(
5     f.count('*'),
6     f.countDistinct('login')
7   ) \
8   .show()
```

command	count(1)	count(DISTINCT login)
NULL	507	99
read	5051	100
other	1458	100
write	2984	100

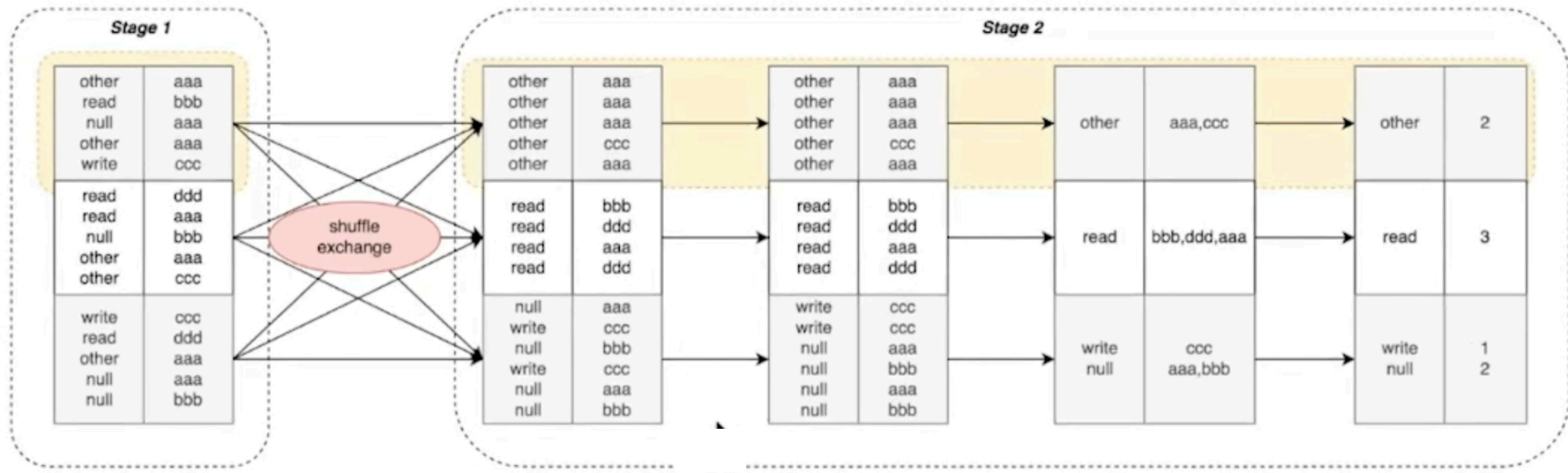
Пример - логика



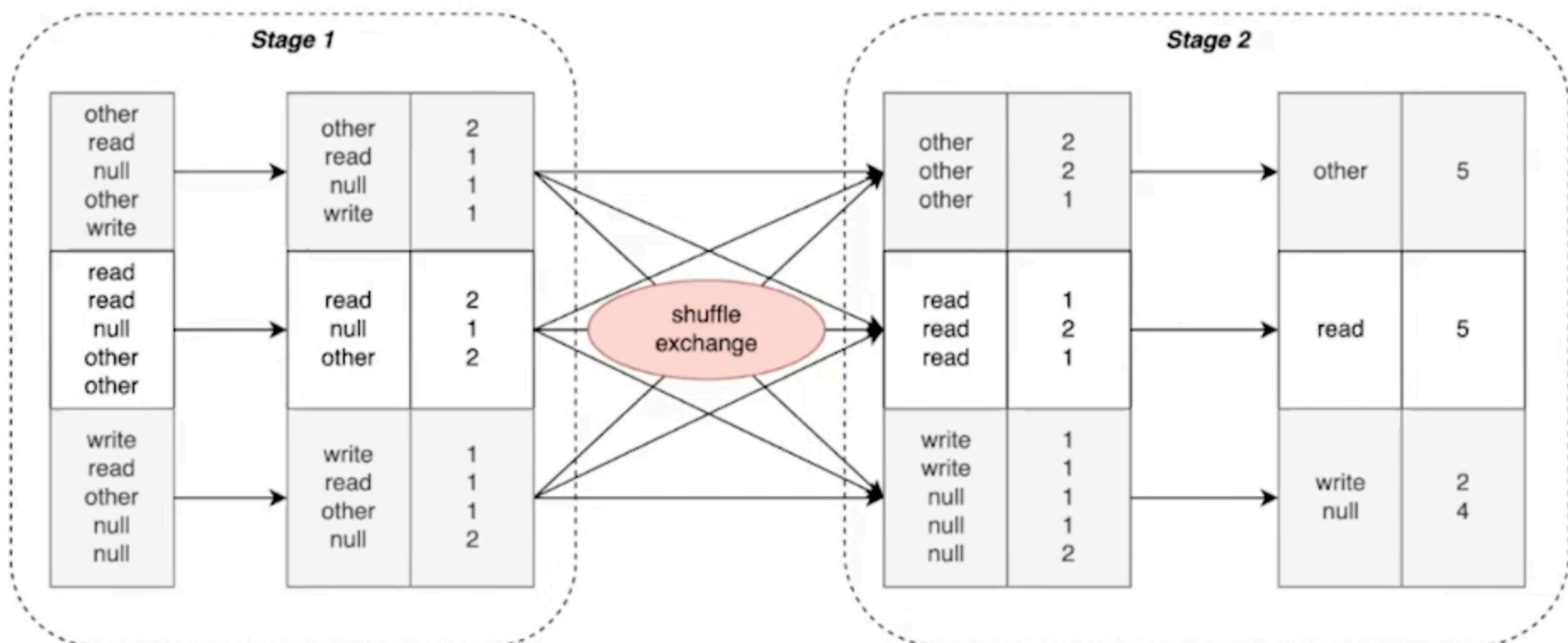
Пример - физика (distinct)



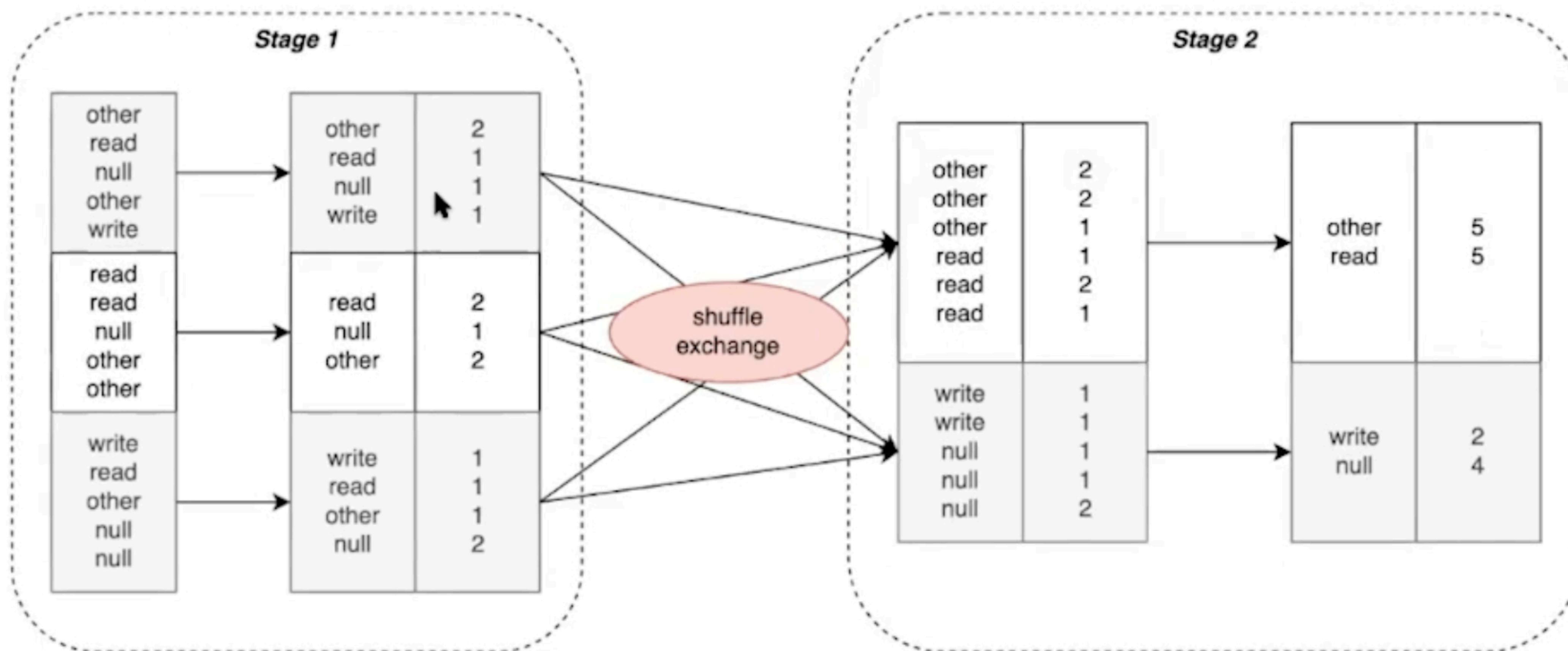
Пример - физика (distinct)



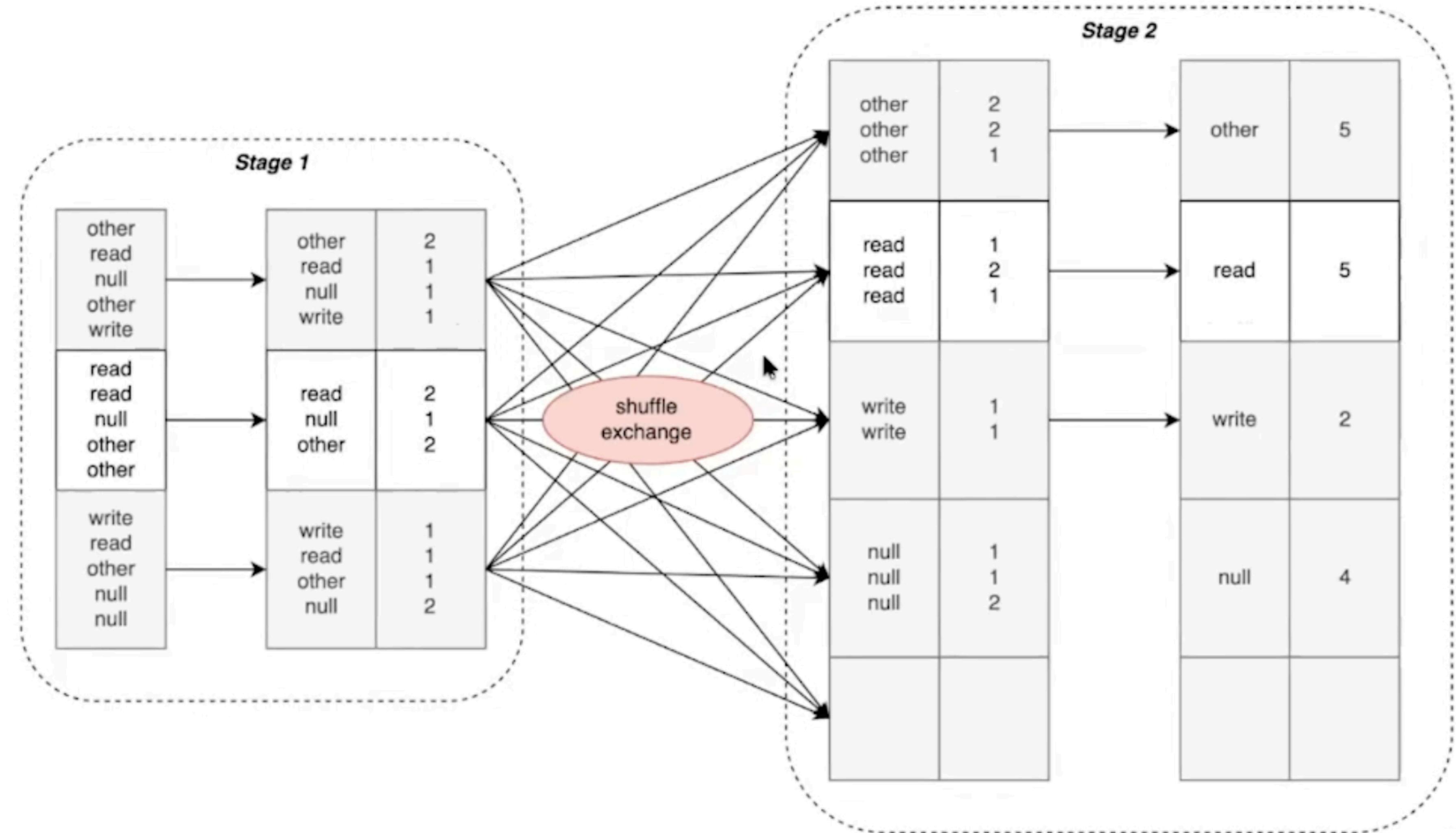
Пример - физика (count)



Shuffle partitions



Shuffle partitions

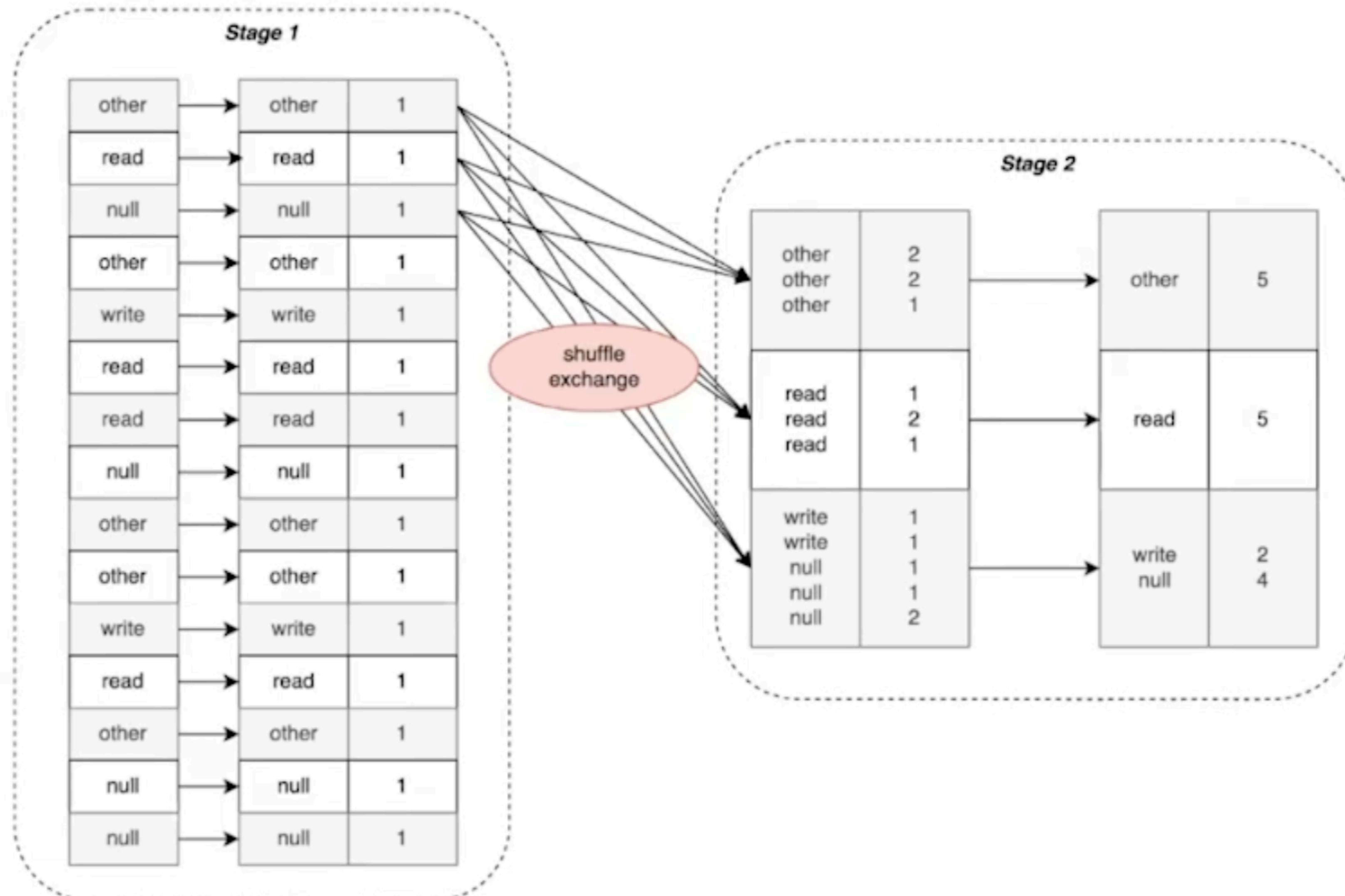


Shuffle partitions

```
1 spark.conf.get('spark.sql.shuffle.partitions')  
'200'
```

- › Много партиций - плохо, не используем бенефиты от предагрегации
- › Мало партиций - тоже плохо, начинается disk spill и skew
- › Нужно мерить пера и держать баланс

Чем нам грозят маленькие партиции

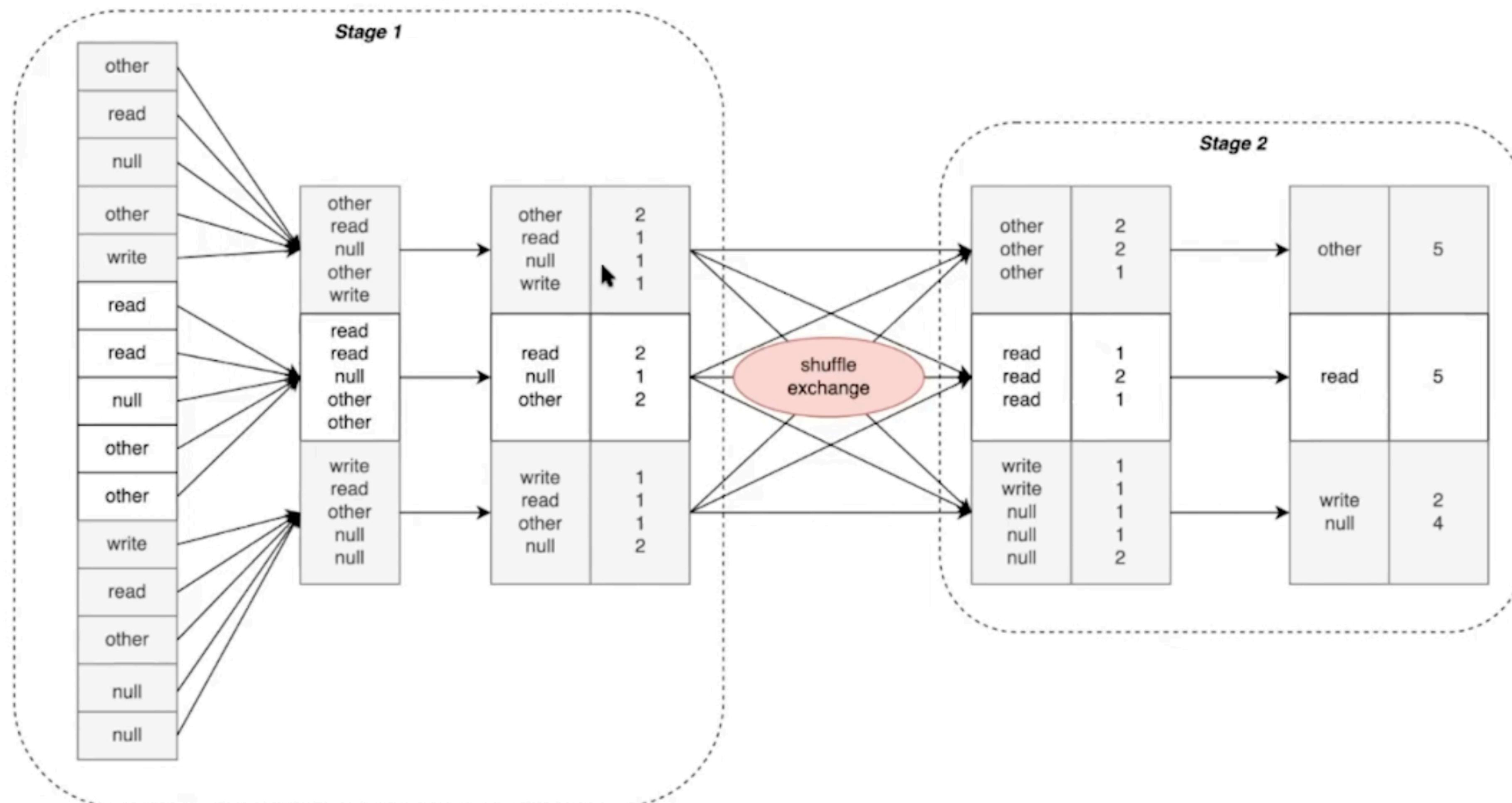


Как это поправить

```
1 df = spark.read.parquet('/content/example_4')
2
3 df.groupby('command') \
4   .coalesce(3) \
5   .agg(
6     f.count('*'),
7     f.countDistinct('login')
8   ) \
9   .show()
```

command	count(1)	count(DISTINCT login)
NULL	507	99
read	5051	100
other	1458	100
write	2984	100

Что получится



Coalesce vs Repartition

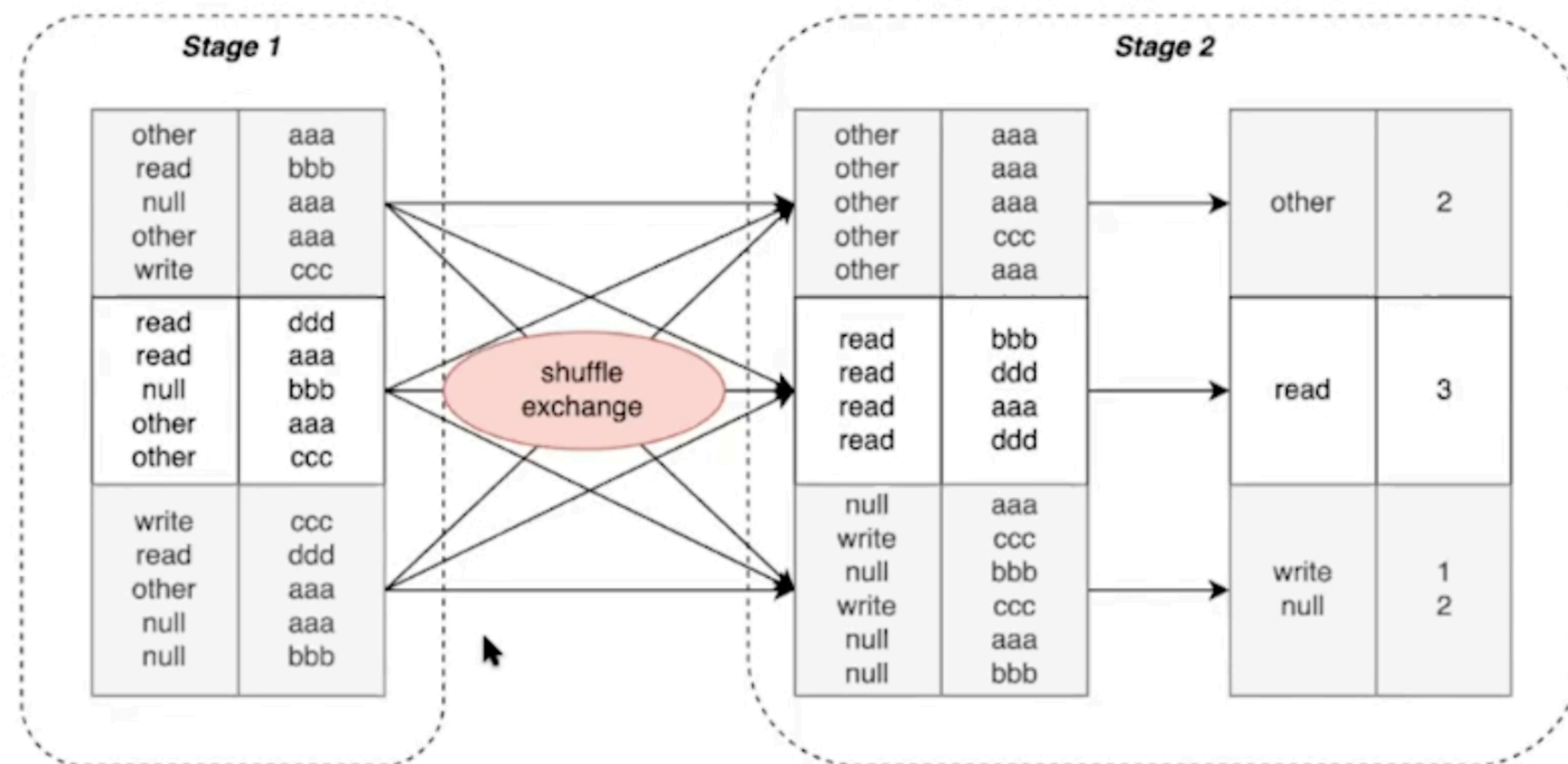
```
1 df = spark.read.parquet('/content/example_4')
2
3 df.groupby('command') \
4     .agg(
5         f.count('*'),
6         f.countDistinct('login')
7     ) \
8     .coalesce(1) \
9     .show()
```

command	count(1)	count(DISTINCT login)
NULL	507	99
read	5051	100
other	1458	100
write	2984	100

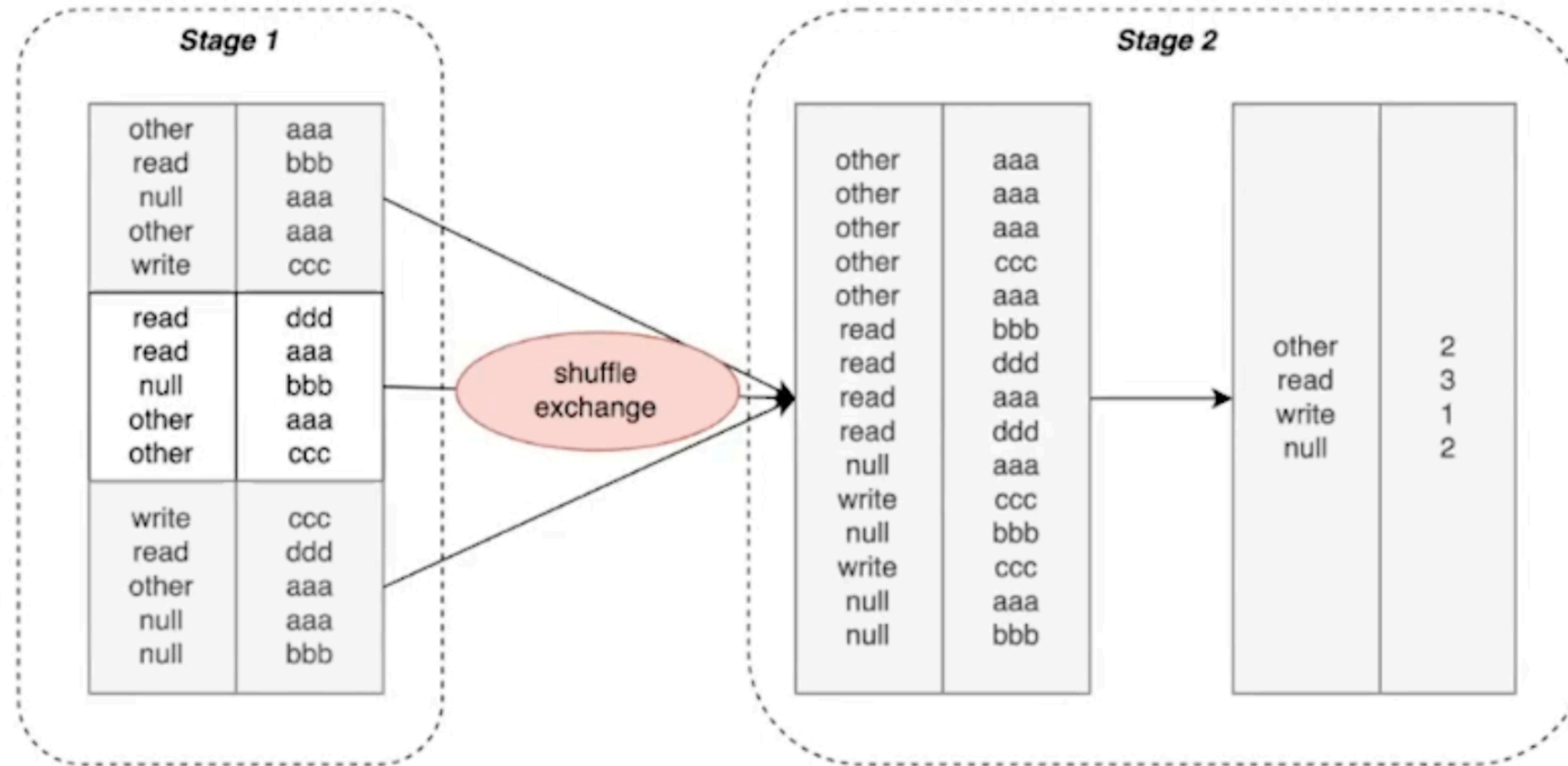
```
1 df = spark.read.parquet('/content/example_4')
2
3 df.groupby('command') \
4     .agg(
5         f.count('*'),
6         f.countDistinct('login')
7     ) \
8     .repartition(1) \
9     .show()
```

command	count(1)	count(DISTINCT login)
NULL	507	99
read	5051	100
other	1458	100
write	2984	100

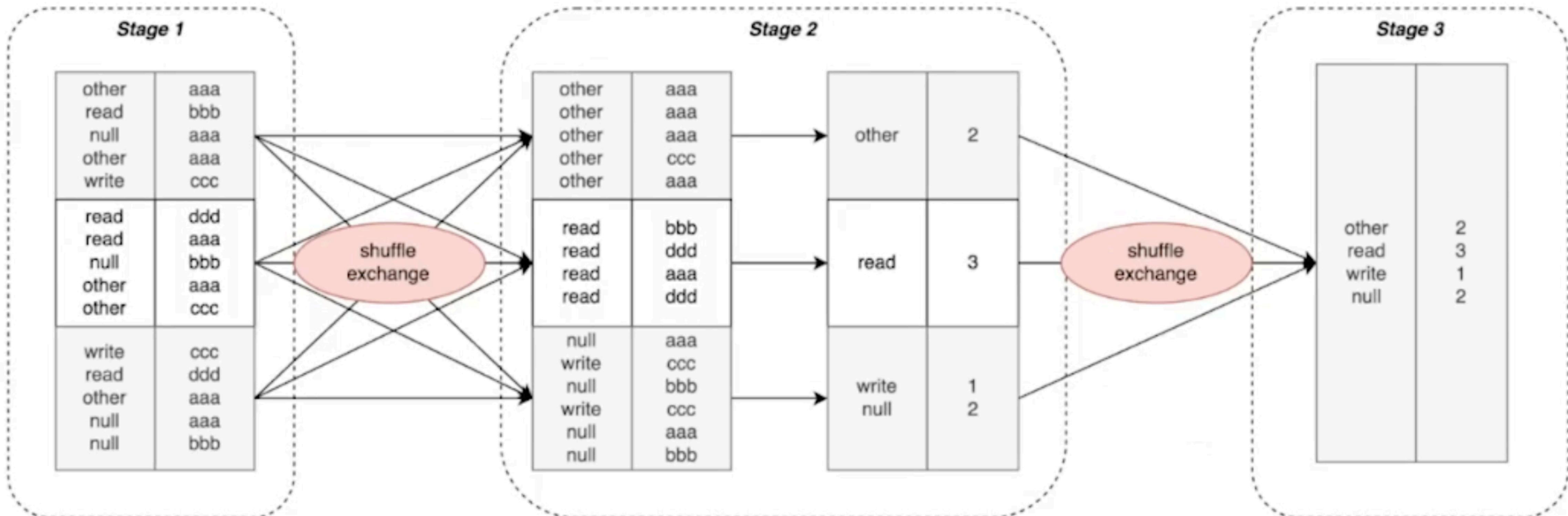
БЫЛО



Coalesce



Repartition

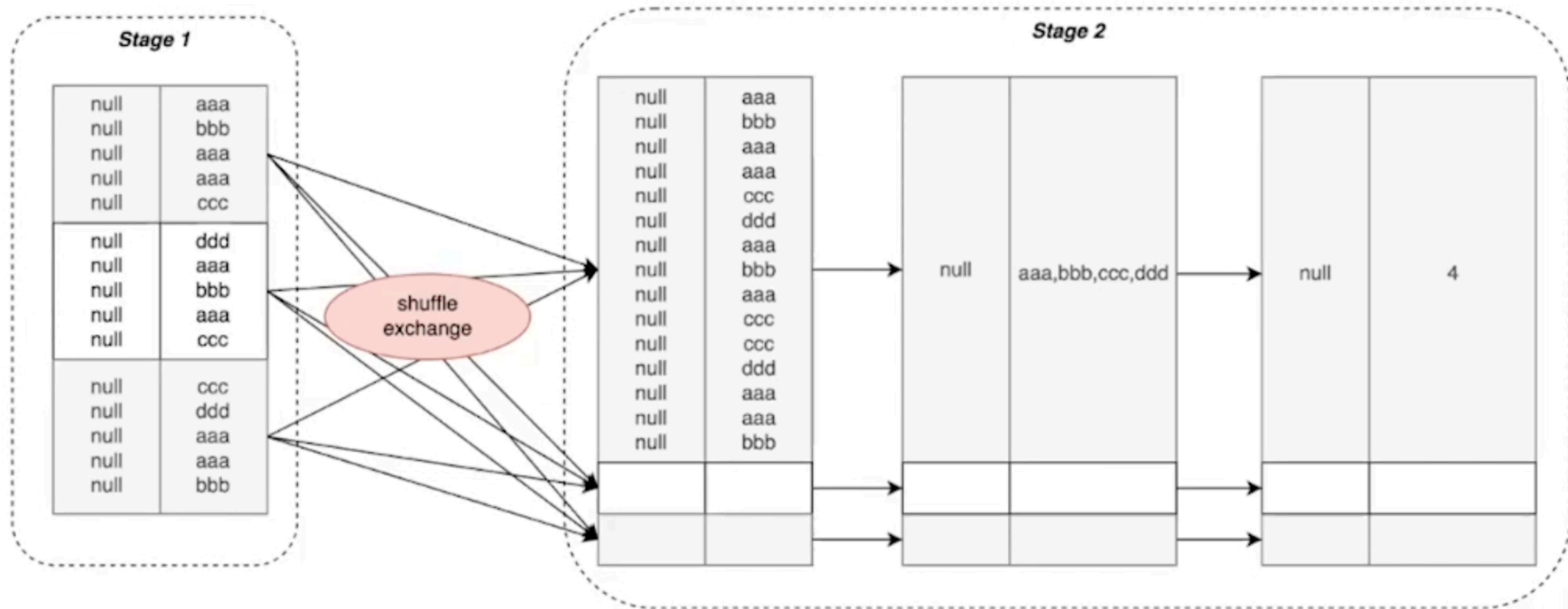


Partition skew

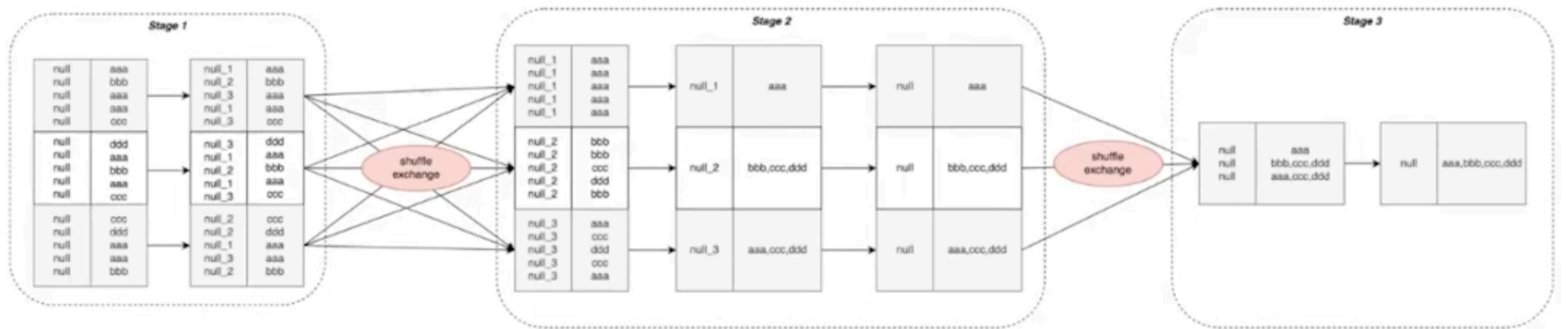
› Представим, что мы хотим выполнить пример, но наши данные выглядят вот так

Stage 1		
null		aaa
null		bbb
null		aaa
null		aaa
null		ccc
null		ddd
null		aaa
null		bbb
null		aaa
null		ccc
null		ccc
null		ddd
null		aaa
null		aaa
null		bbb

Partition skew



Partition skew



2.5 - Join

Пример

```
1 usage = spark.read.parquet('/content/example_4')
2 staff = spark.read.parquet('/content/example_5')
3
4 usage.join(staff, ['login'], 'inner') \
5     .groupby('is_robot_flg') \
6     .agg( f.count('*'), f.countDistinct('command') ) \
7     .repartition(1) \
8     .show()
```

is_robot_flg	count(1)	count(DISTINCT command)
true	5883	3
false	4117	3

Пример - логика

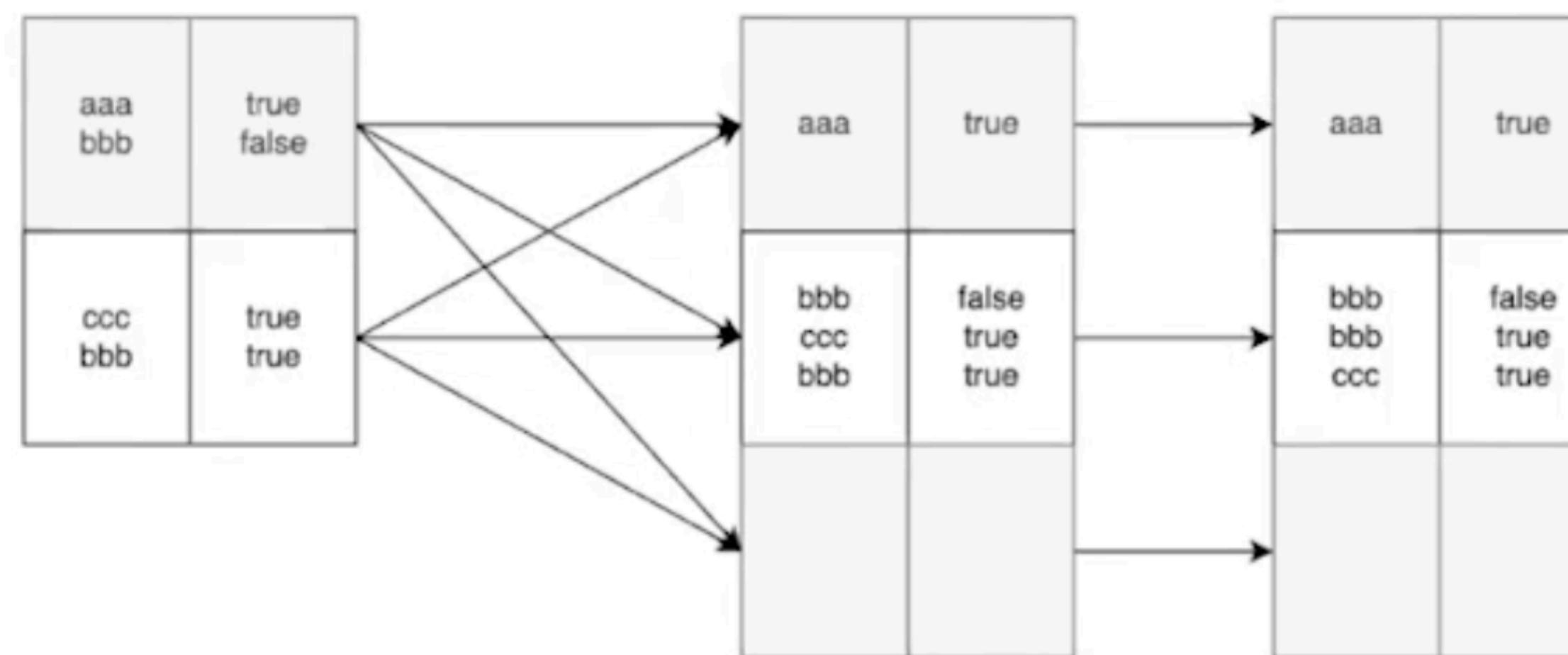
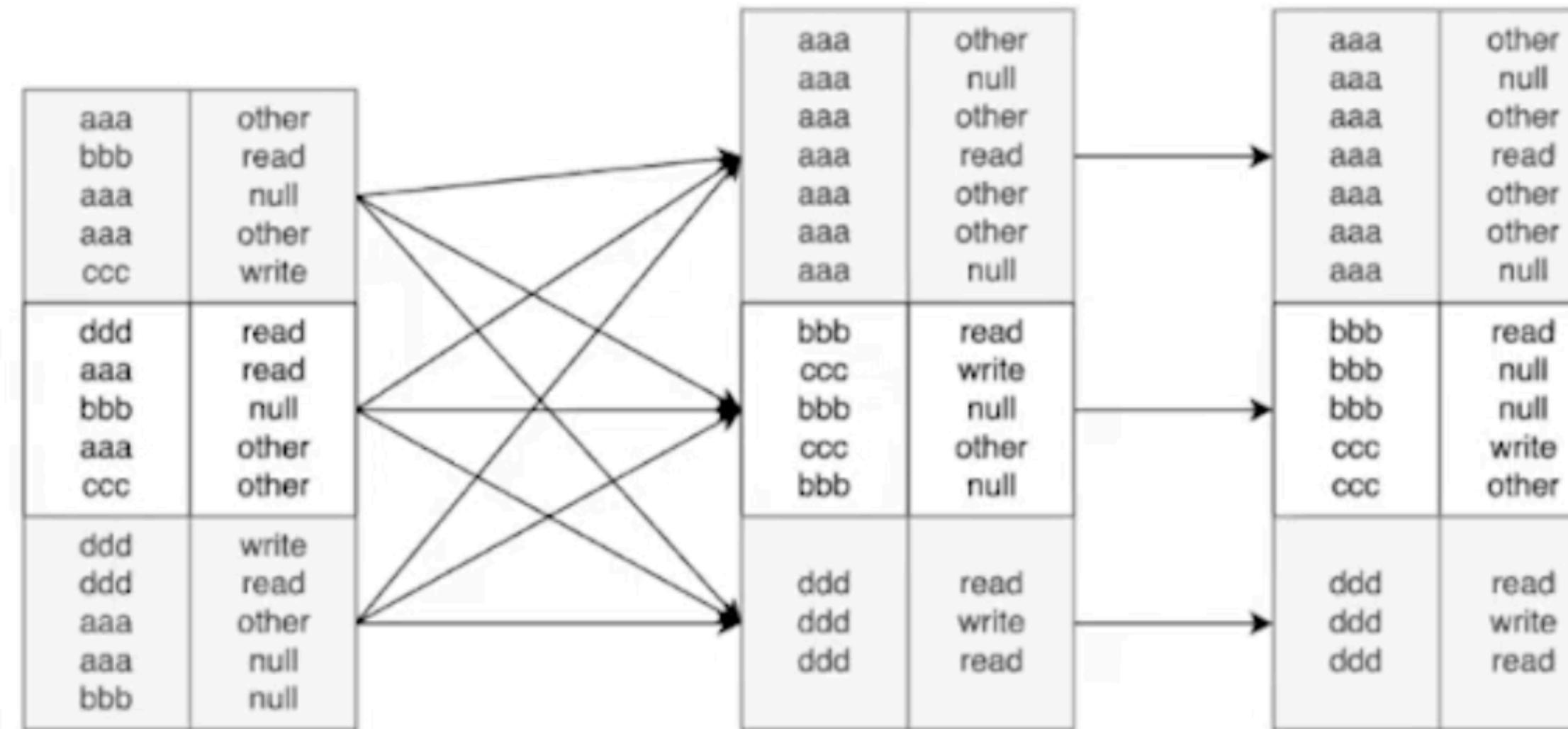
login	command
aaa	other
bbb	read
aaa	null
aaa	other
ccc	write
ddd	read
aaa	read
bbb	null
aaa	other
ccc	other
ddd	write
ddd	read
aaa	other
aaa	null
bbb	null

login	is_robot
aaa	true
bbb	false
bbb	true
aaa	true
aaa	true
ccc	true
bbb	true
ccc	true

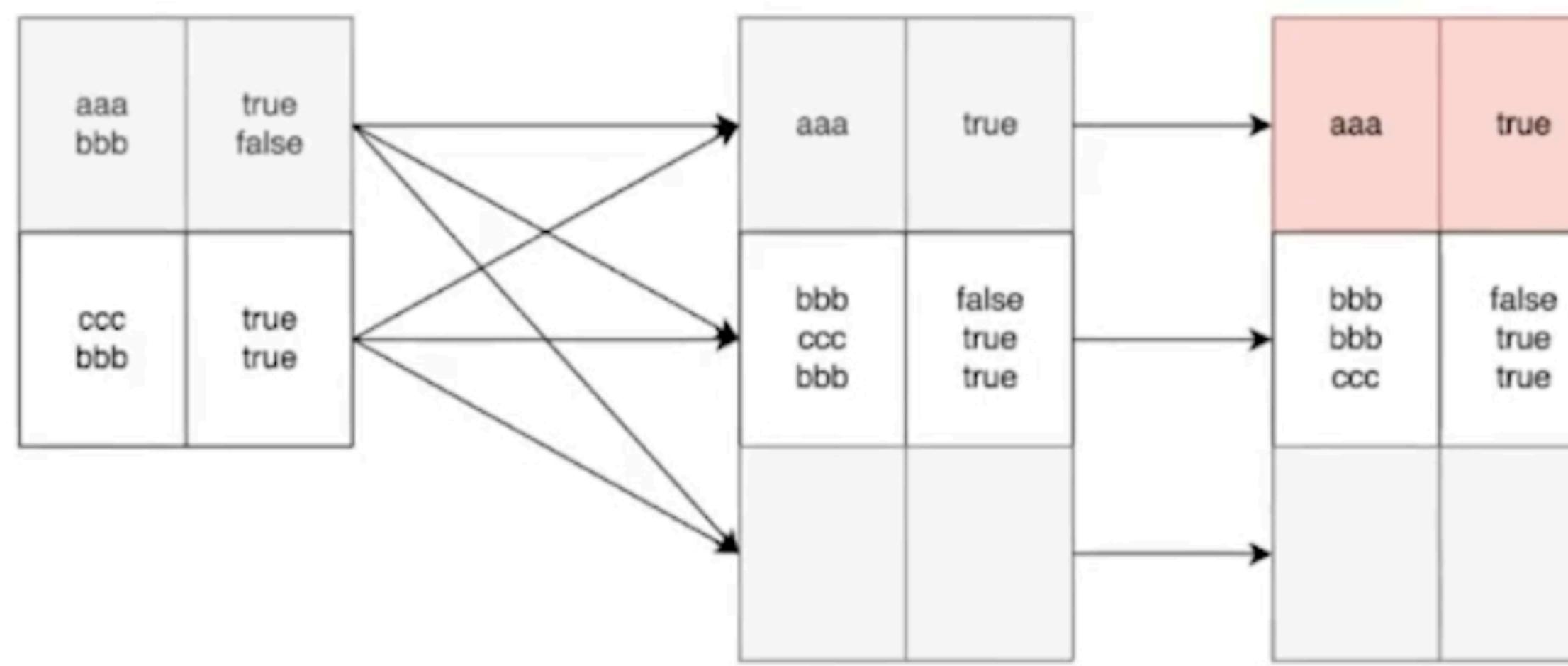
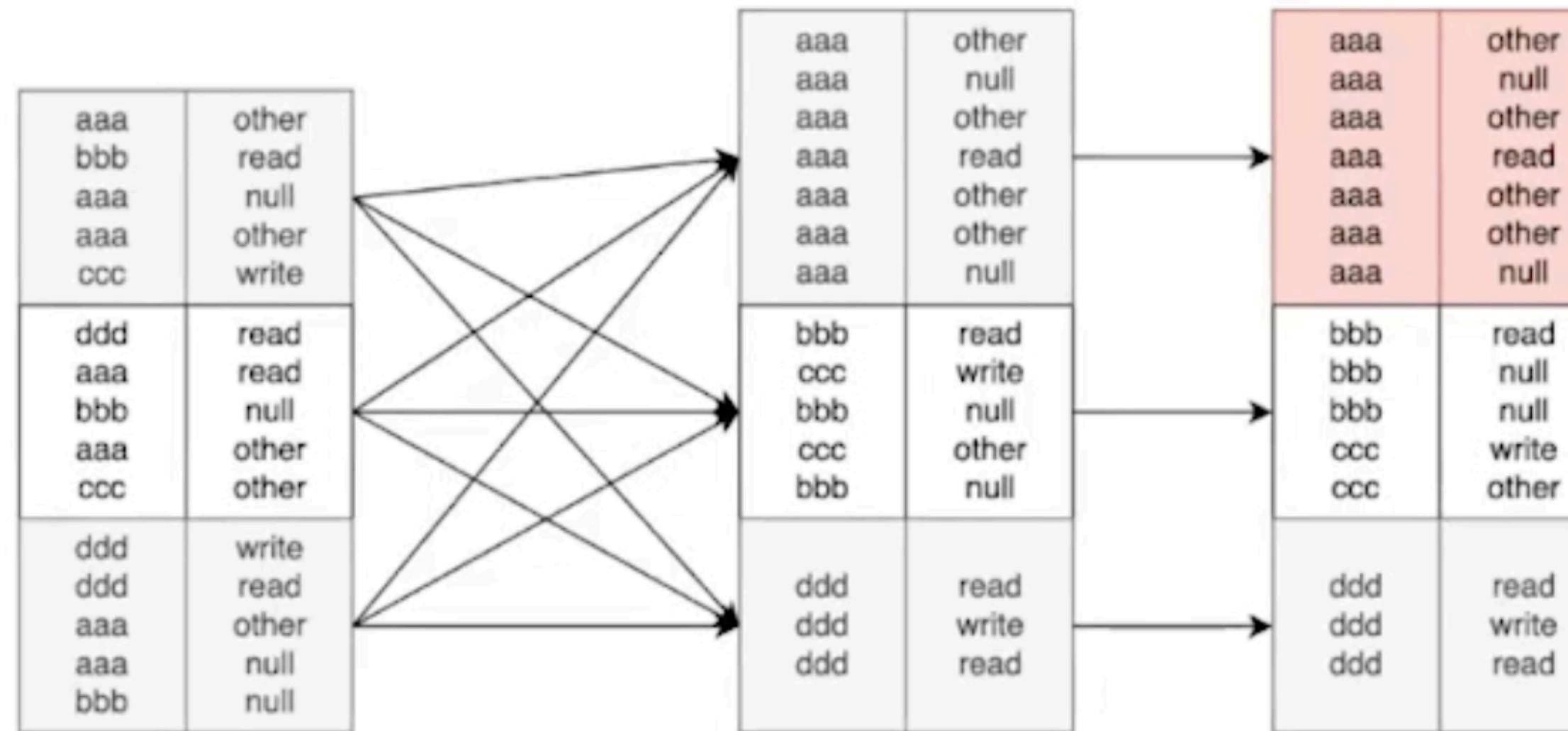


login	is_robot	command
aaa	true	other
bbb	false	read
bbb	true	read
aaa	true	null
aaa	true	other
ccc	true	write
aaa	true	read
bbb	false	null
bbb	true	null
aaa	true	other
ccc	true	other
aaa	true	other
aaa	true	null
bbb	false	null
bbb	true	null

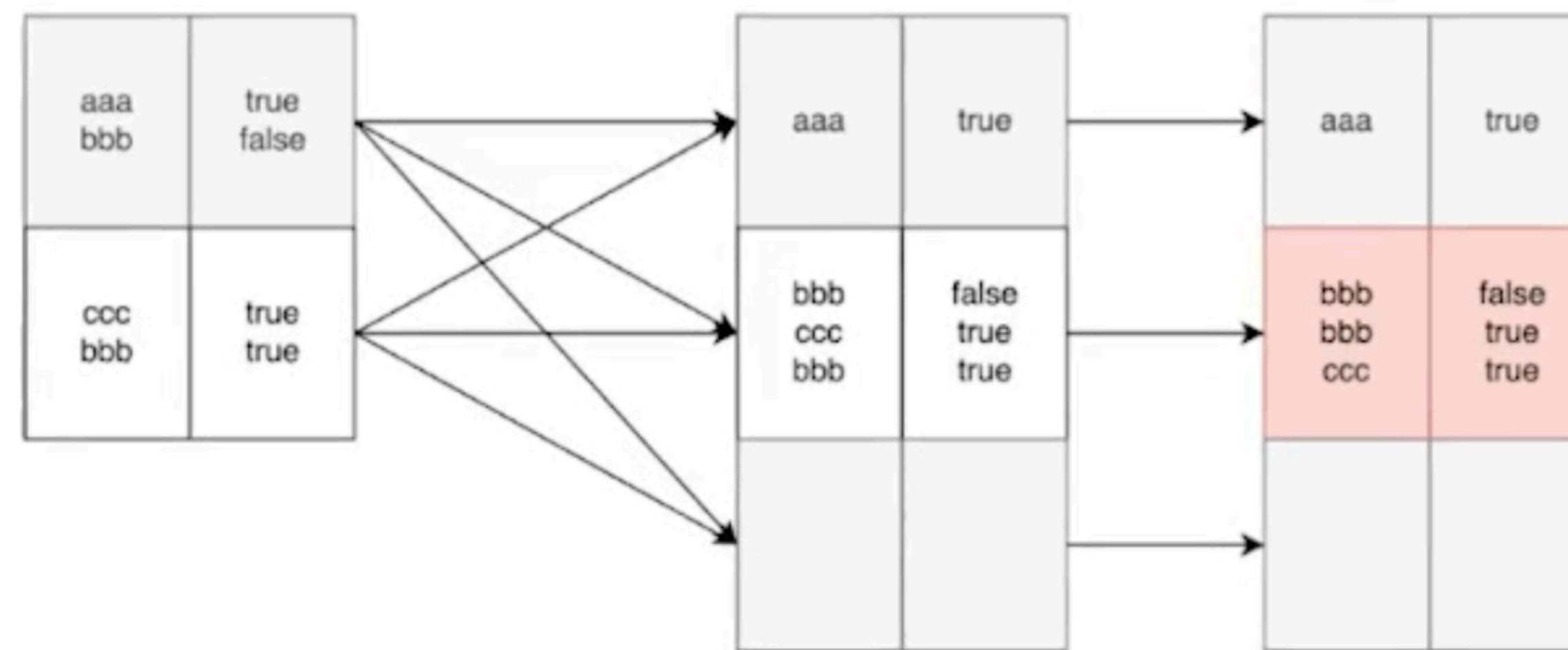
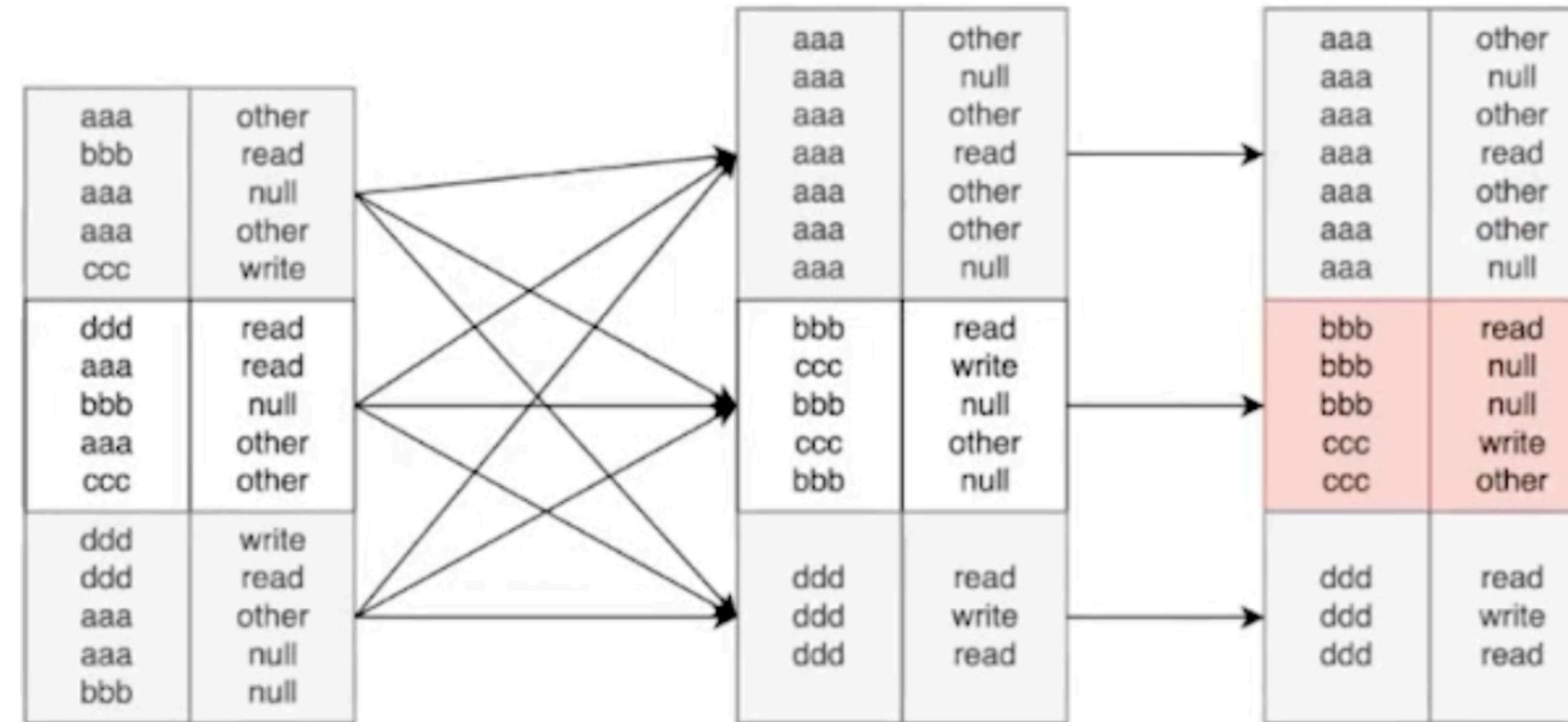
Пример - физика



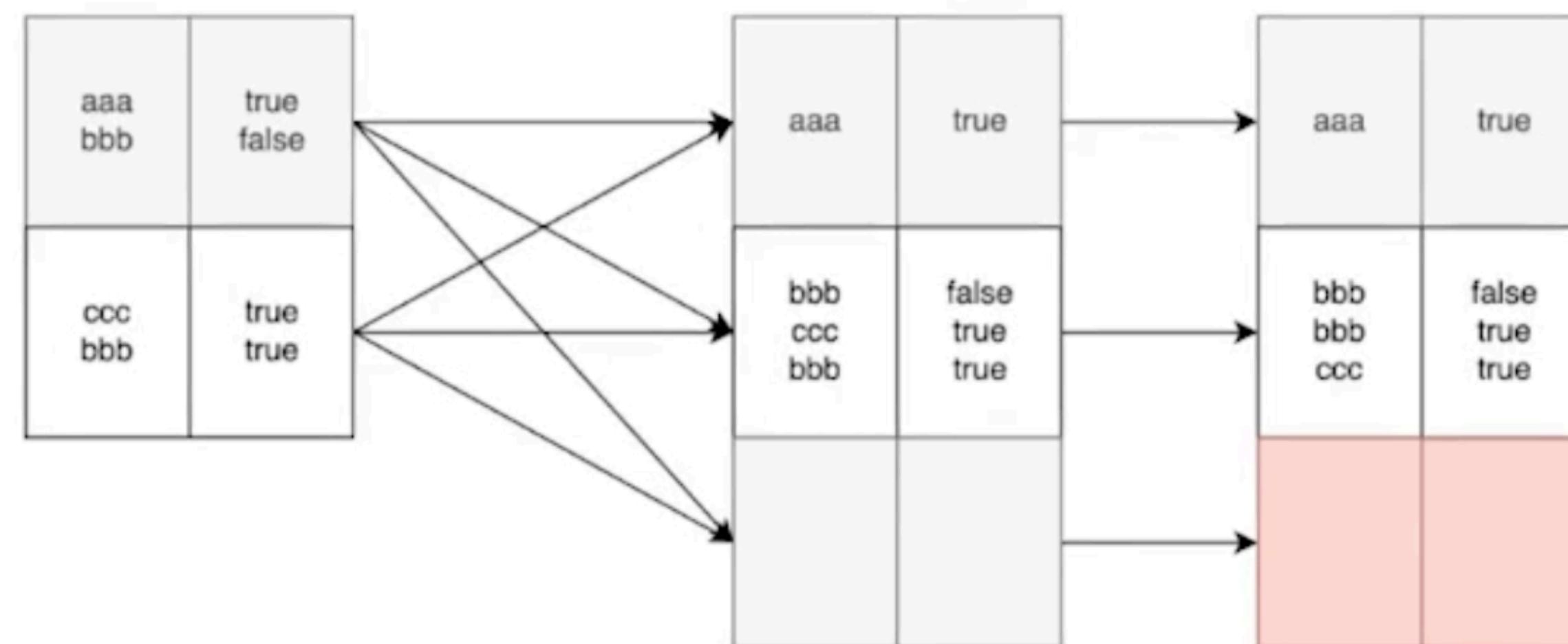
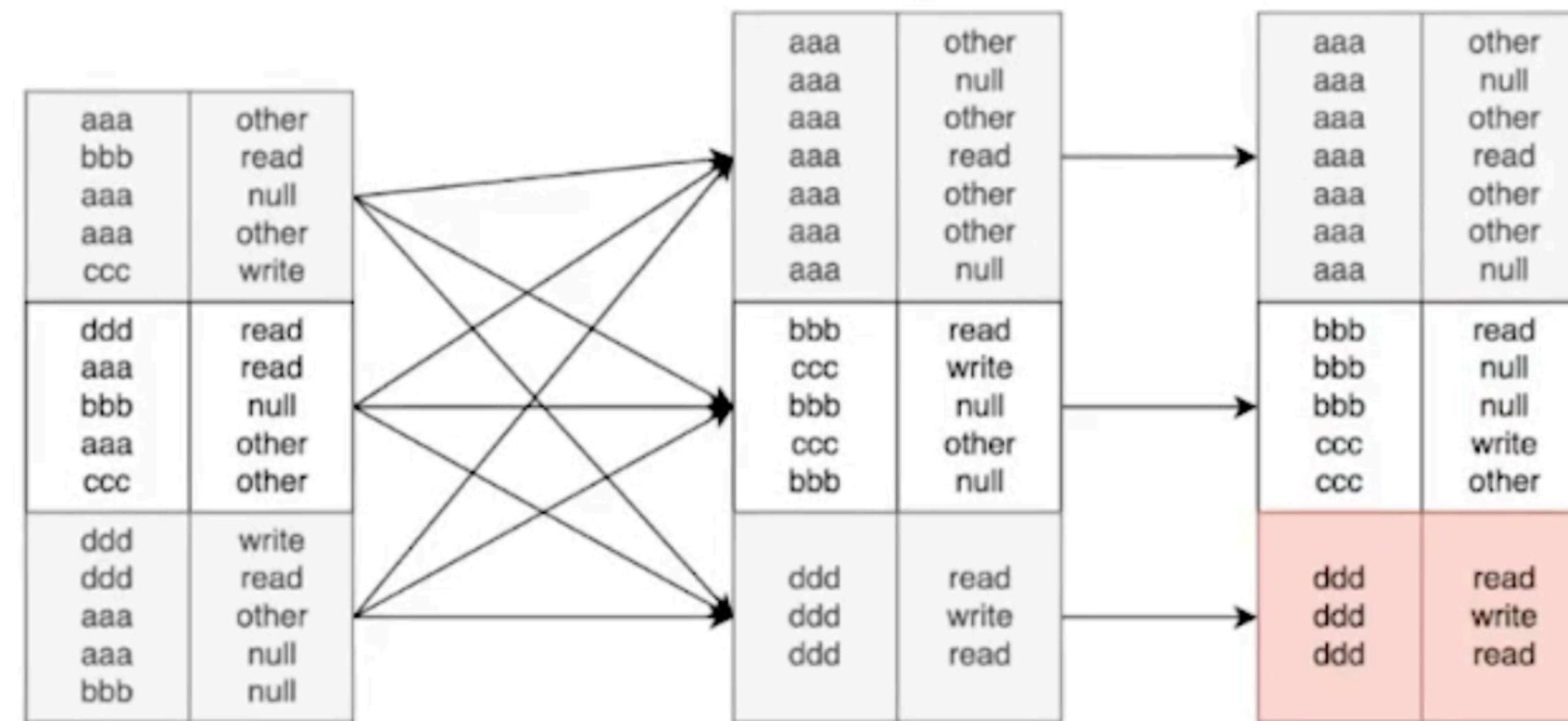
Пример - физика



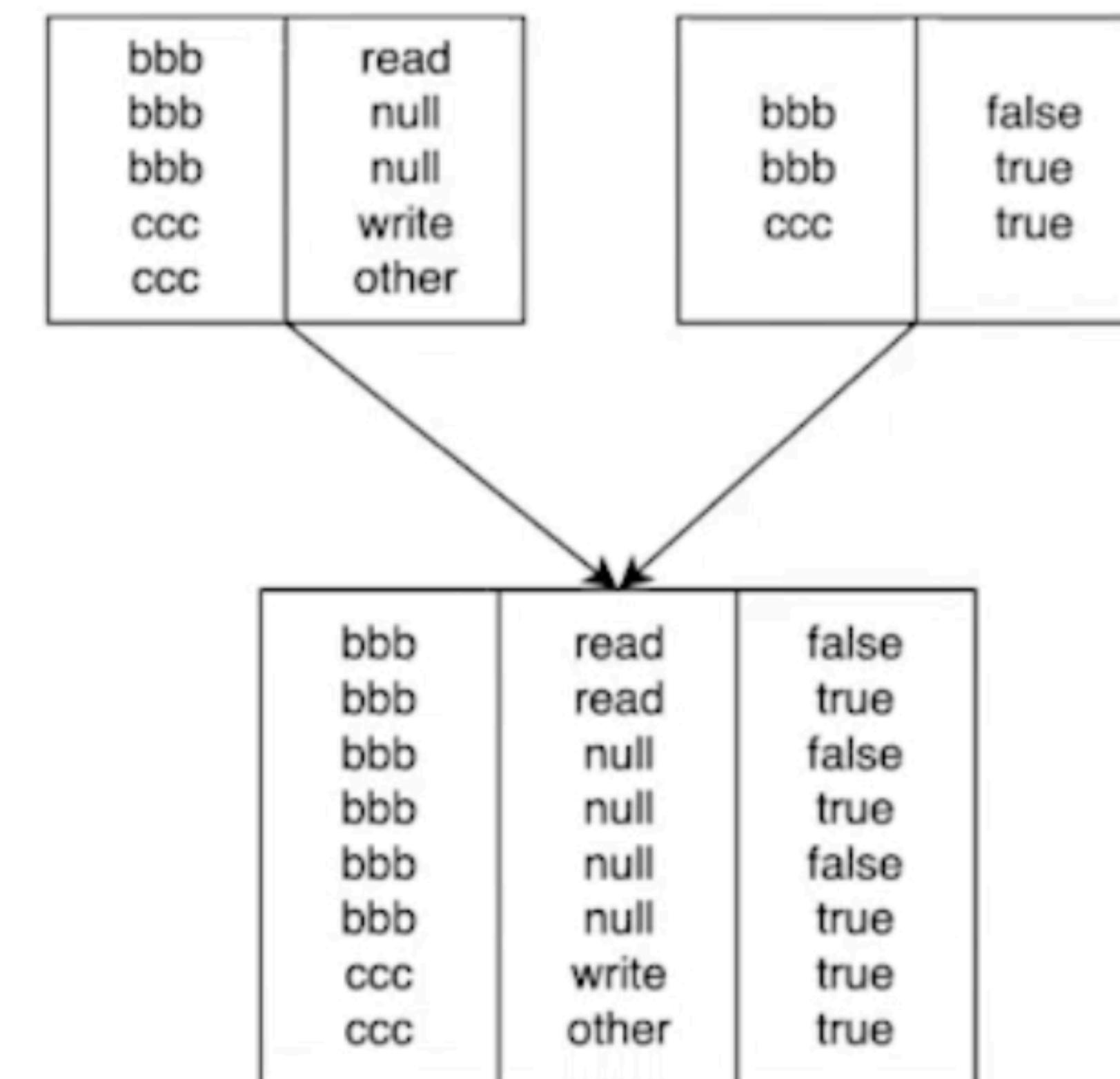
Пример - физика



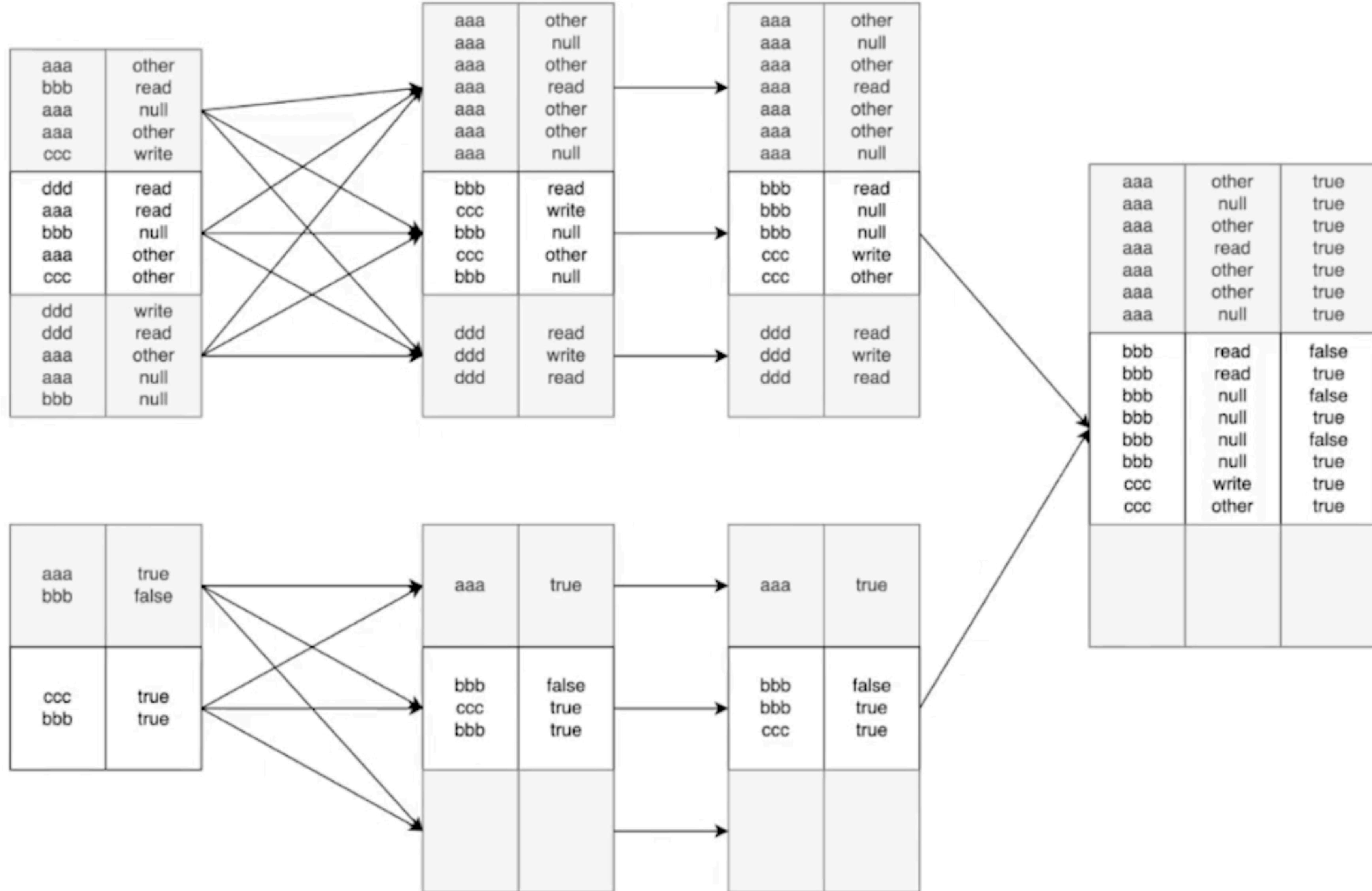
Пример - физика



Sort Merge Join



Пример - физика

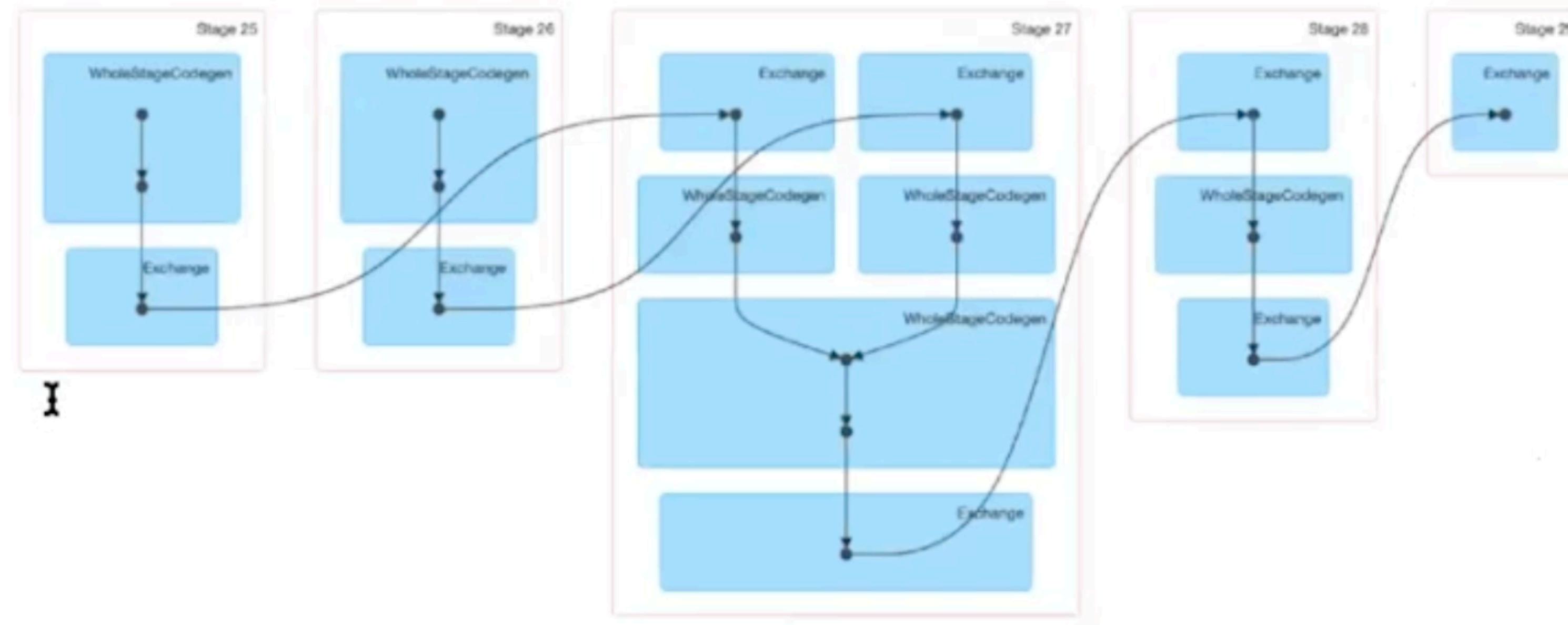


Пример - физика

Details for Job 17

Status: SUCCEEDED
Completed Stages: 5

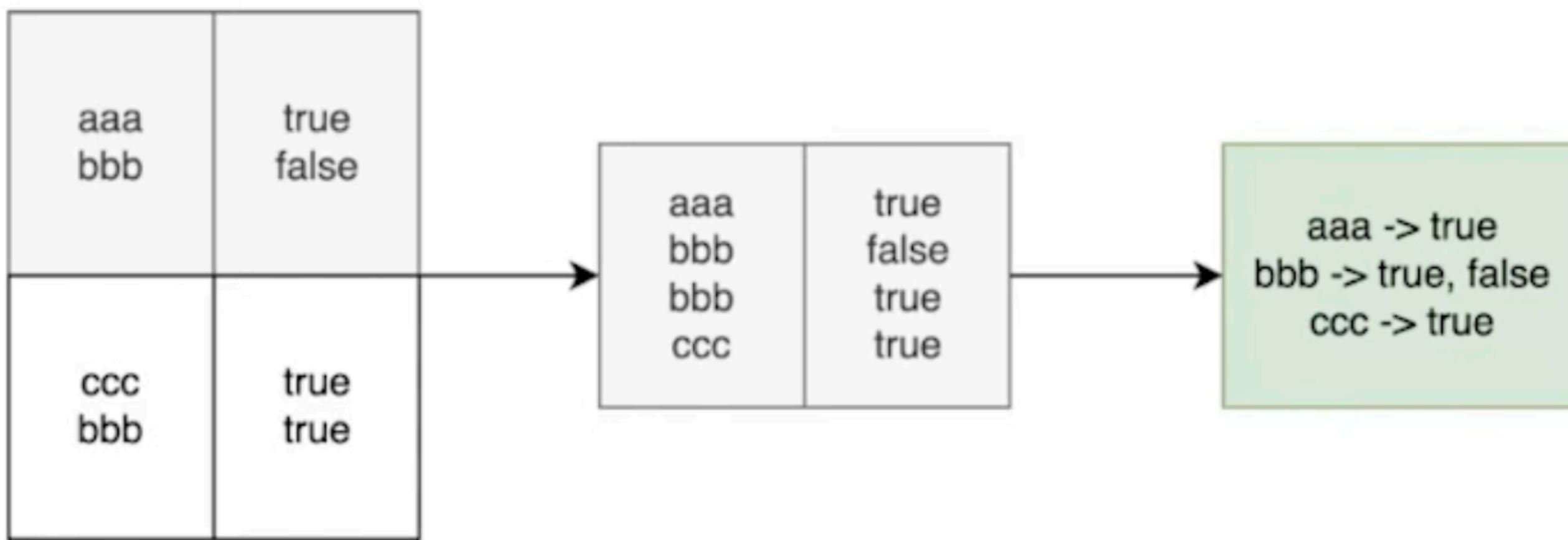
- Event Timeline
- DAG Visualization



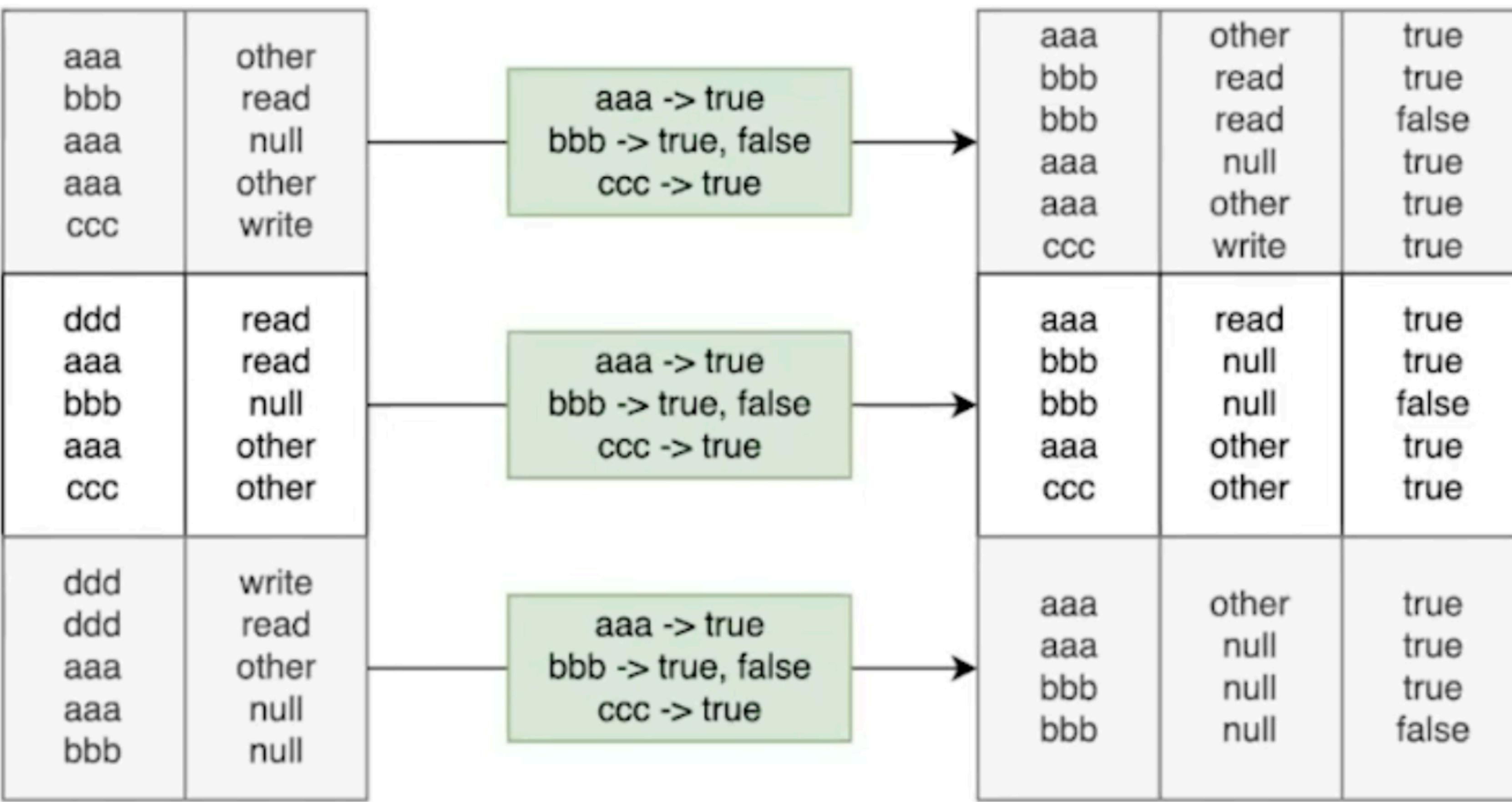
Completed Stages (5)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
29	parquet at NativeMethodAccessorImpl.java:0	+details 2020/11/03 18:21:18	0.5 s	1/1		620.0 B	122.0 B	
28	parquet at NativeMethodAccessorImpl.java:0	+details 2020/11/03 18:21:15	3 s	200/200		8.1 KB	122.0 B	
27	parquet at NativeMethodAccessorImpl.java:0	+details 2020/11/03 18:21:10	5 s	200/200		2.2 MB	8.1 KB	
26	parquet at NativeMethodAccessorImpl.java:0	+details 2020/11/03 18:21:00	5 s	1/1	2.3 MB		2.5 MB	
25	parquet at NativeMethodAccessorImpl.java:0	+details 2020/11/03 18:21:00	5 s	1/1	105.1 KB		142.0 KB	

Hash Map Join



Hash Map Join

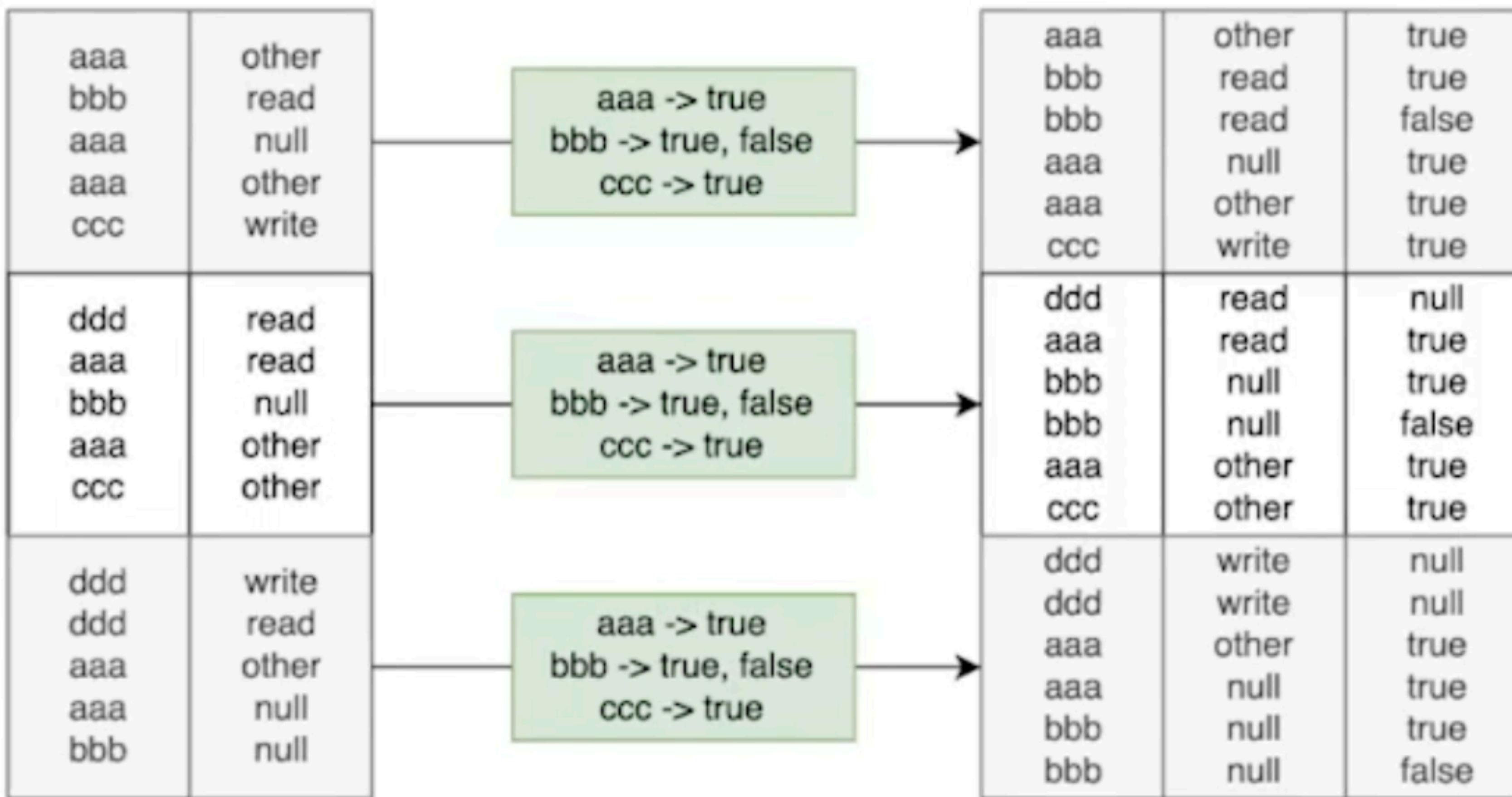


Как это сделать?

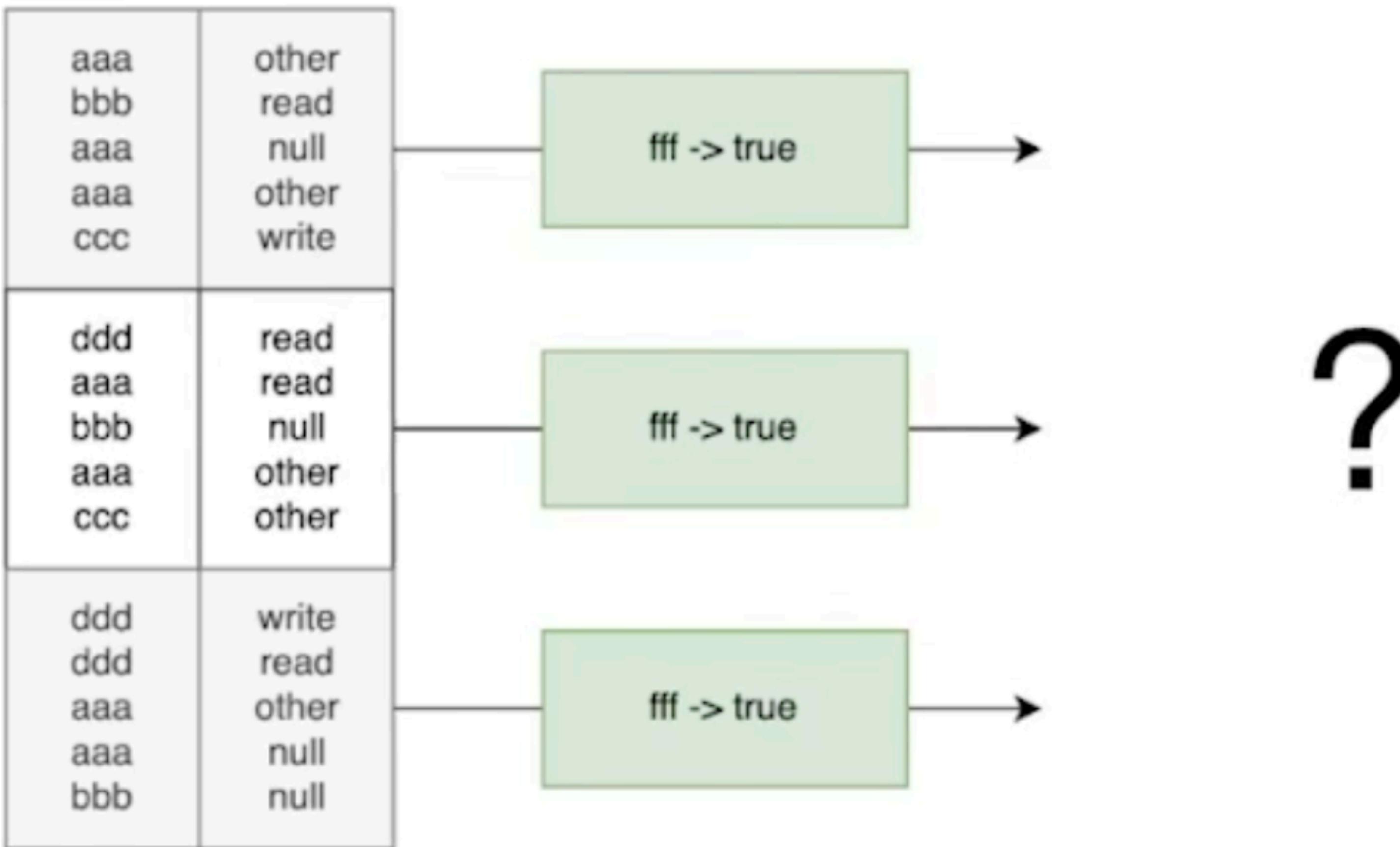
```
1 from pyspark.sql.functions import broadcast
2
3 usage = spark.read.parquet('/content/example_4')
4 staff = spark.read.parquet('/content/example_5')
5
6 usage.join(broadcast(staff), ['login'], 'inner') \
7     .groupby('is_robot_flg') \
8     .agg( f.count('*'), f.countDistinct('command') ) \
9     .repartition(1) \
10    .show()
```

is_robot_flg	count(1)	count(DISTINCT command)
true	5883	3
false	4117	3

Broadcast правой части



Broadcast правой части



Пример

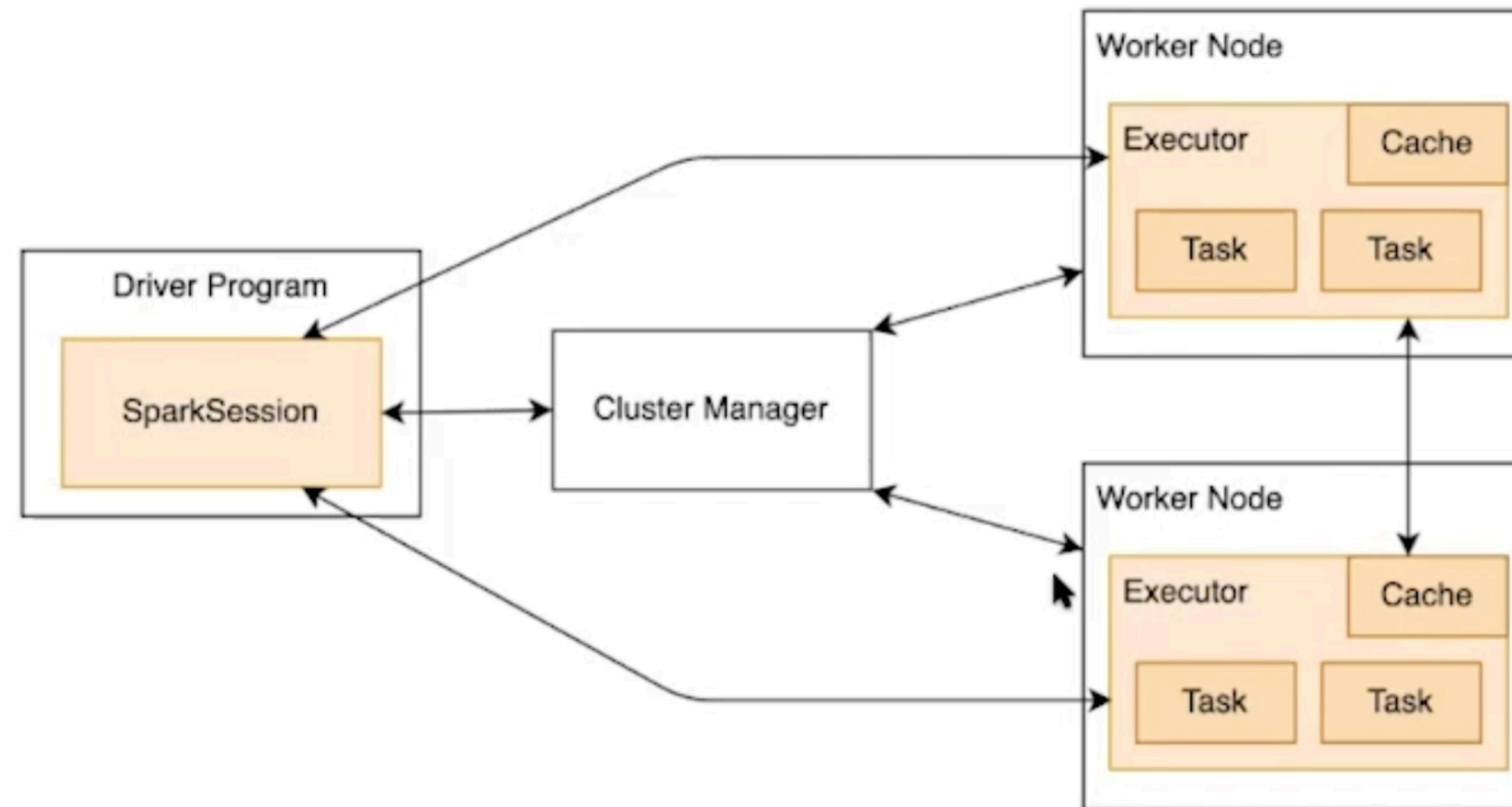
```
1 # before
2 usage.join(broadcast(staff), ['login'], 'right_outer')
3
4 # after
5 tmp = usage.join(broadcast(staff), ['login'], 'inner')
6 staff.join(broadcast(tmp), ['login'], 'left_outer')
```

Пример

```
1 # before
2 usage.join(broadcast(staff), ['login'], 'right_outer')
3
4 # after
5 tmp = usage.join(broadcast(staff), ['login'], 'inner')
6 staff.join(broadcast(tmp), ['login'], 'left_outer')
7
8 # right
9 keys = staff.select('login').distinct()
10 tmp = usage.join(broadcast(keys), ['login'], 'inner')
11 staff.join(broadcast(tmp), ['login'], 'left_outer')
```

2.6 - pySpark

Spark



pySpark

