Использование вероятностных структур данных для оптимизации ETL - процессов

Вертлиб Дмитрий

d.vertlib@crpt.ru

План доклада:

1. Проблематика
2. Использование регулярных структур
3. Что такое вероятностные структуры данных
4. Использование фильтров Блума
5. Ленивый сегментный фильтр
6. Выводы

Обновление данных

Таблица назначения	Входные данные
Apache Iceberg	Apache Iceberg
Apache Parquet	Apache Parquet
ZSTD	ZSTD
600 млрд, строк	1,6 (4) млрд, строк
100 Тбайт	160(400) Гбайт

SLA на обновление 4 часа

Обновление данных

```
merge into dst
 using  src
  on dst.id = src.id
 when matched then update set *
 when not matched  then insert *
```

Merge-On-Read

Обновление данных

```
== Physical Plan ==
WriteDelta (16)
+- Sort (15)
   +- Exchange (14)
      +- MergeRows (13)
         +- Project (12)
            +- SortMergeJoin RightOuter (11)
               :- * Sort (6)
               :  +- Exchange (5)
               :     +- * Filter (4)
               :        +- * Project (3)
               :           +- * ColumnarToRow (2)
               :              +- BatchScan dst (1)
               +- Sort (10)
                  +- Exchange (9)
                     +- Project (8)
                        +- BatchScan src (7)
```

Merge-On-Read

Обновление данных

```
== Physical Plan ==
WriteDelta (16)
+- Sort (15)
   +- Exchange (14)
      +- MergeRows (13)
         +- Project (12)
            +- SortMergeJoin RightOuter (11)
               :- * Sort (6)
               :  +- Exchange (5)
               :     +- * Filter (4)
               :        +- * Project (3)
               :           +- * ColumnarToRow (2)
               :              +- BatchScan dst (1)
               +- Sort (10)
                  +- Exchange (9)
                     +- Project (8)
                        +- BatchScan src (7)
```

RightOuter (SortMergeJoin)

Merge-On-Read

Обновление данных

```
== Physical Plan ==
WriteDelta (16)
+- Sort (15)
   +- Exchange (14)
      +- MergeRows (13)
         +- Project (12)
            +- SortMergeJoin RightOuter (11)
               :- * Sort (6)
               :  +- Exchange (5)
               :     +- * Filter (4)
               :        +- * Project (3)
               :           +- * ColumnarToRow (2)
               :              +- BatchScan dst (1)
               +- Sort (10)
                  +- Exchange (9)
                     +- Project (8)
                        +- BatchScan src (7)
```

Merge
удаления и добавления

Merge-On-Read

Обновление данных

```
== Physical Plan ==
WriteDelta (16)
+- Sort (15)
   +- Exchange (14)
      +- MergeRows (13)
         +- Project (12)
            +- SortMergeJoin RightOuter (11)
               :- * Sort (6)
               :  +- Exchange (5)
               :     +- * Filter (4)
               :        +- * Project (3)
               :           +- * ColumnarToRow (2)
               :              +- BatchScan dst (1)
               +- Sort (10)
                  +- Exchange (9)
                     +- Project (8)
                        +- BatchScan src (7)
```

Чтение
100 ТБ

Merge-On-Read

Обновление данных

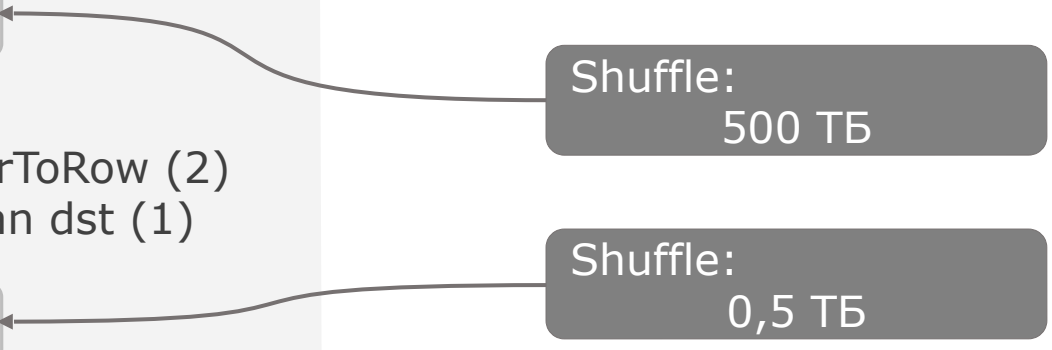
```
== Physical Plan ==
WriteDelta (16)
+- Sort (15)
   +- Exchange (14)
      +- MergeRows (13)
         +- Project (12)
            +- SortMergeJoin RightOuter (11)
               :- * Sort (6)
               :  +- Exchange (5)
               :     +- * Filter (4)
               :        +- * Project (3)
               :           +- * ColumnarToRow (2)
               :              +- BatchScan dst (1)
               +- Sort (10)
                  +- Exchange (9)
                     +- Project (8)
                        +- BatchScan src (7)
```

Фильтрация
I/O 1000 ТБ

Merge-On-Read

Обновление данных

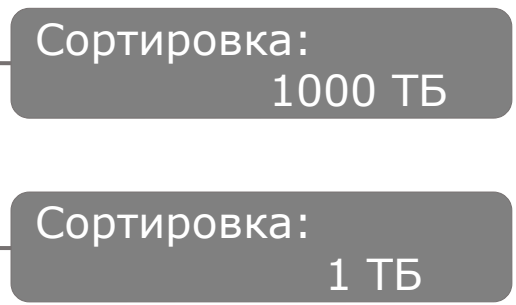
```
== Physical Plan ==
WriteDelta (15)
+- Exchange (14)
   +- MergeRows (13)
      +- * Project (12)
         +- * SortMergeJoin RightOuter (11)
            :- * Sort (6)
            :  +- Exchange (5)
            :     +- * Project (4)
            :        +- * Filter (3)
            :           +- * ColumnarToRow (2)
            :              +- BatchScan dst (1)
            +- * Sort (10)
               +- Exchange (9)
                  +- * ColumnarToRow (8)
                     +- BatchScan src (7)
```



Merge-On-Read

Обновление данных

```
== Physical Plan ==
WriteDelta (15)
+- Exchange (14)
   +- MergeRows (13)
      +- * Project (12)
         +- * SortMergeJoin RightOuter (11)
            :- * Sort (6)
            :  +- Exchange (5)
            :     +- * Project (4)
            :        +- * Filter (3)
            :           +- * ColumnarToRow (2)
            :              +- BatchScan dst (1)
            +- * Sort (10)
               +- Exchange (9)
                  +- * ColumnarToRow (8)
                     +- BatchScan src (7)
```



Merge-On-Read

Обновление данных

```
== Physical Plan ==
WriteDelta (16)
+- Sort (15)
   +- Exchange (14)
      +- MergeRows (13)
         +- Project (12)
            +- SortMergeJoin RightOuter (11)
               :- * Sort (6)
               :  +- Exchange (5)
               :     +- * Filter (4)
               :        +- * Project (3)
               :           +- * ColumnarToRow (2)
               :              +- BatchScan dst (1)
               +- Sort (10)
                  +- Exchange (9)
                     +- Project (8)
                        +- BatchScan src (7)
```

JOIN

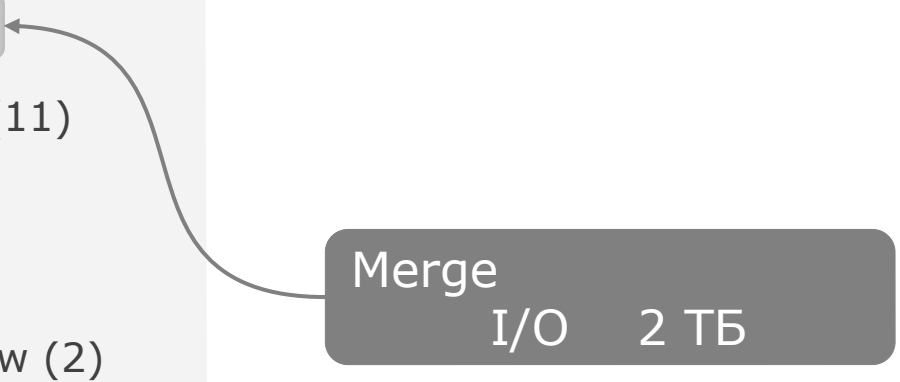
I - 1000 ТБ

O - 2 ТБ

Merge-On-Read

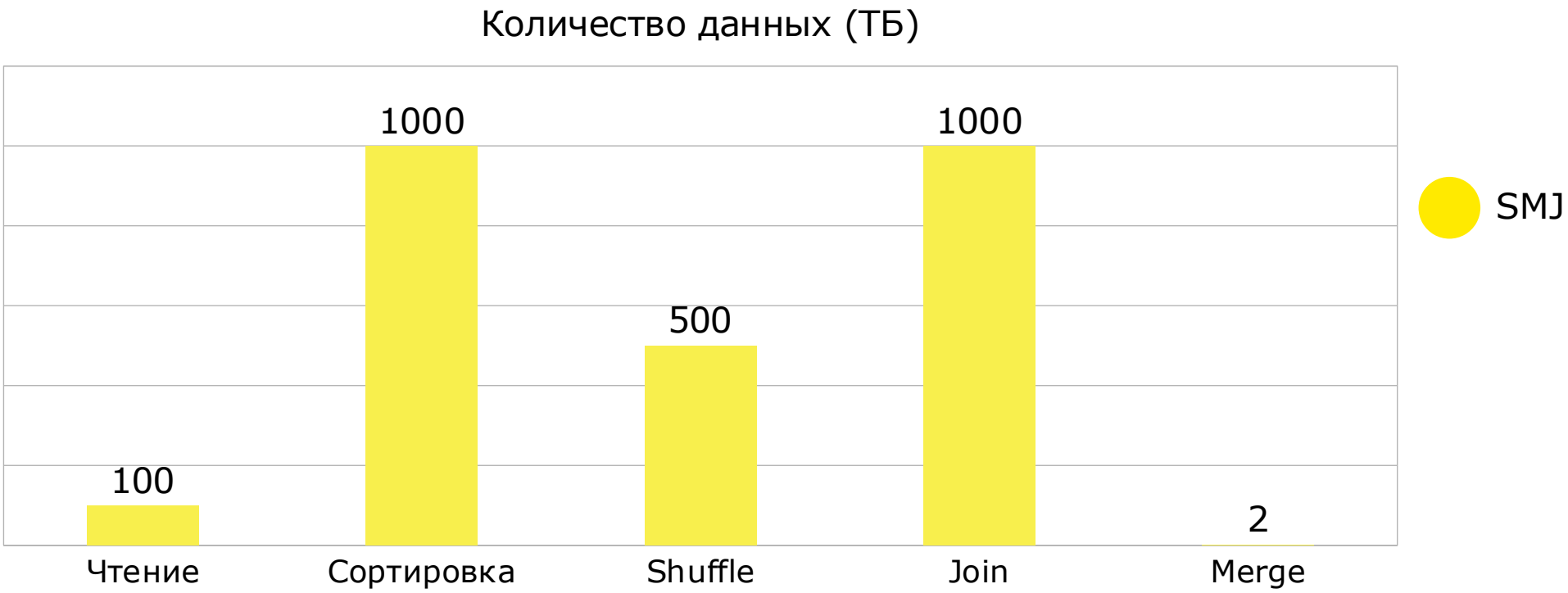
Обновление данных

```
== Physical Plan ==
WriteDelta (16)
+- Sort (15)
   +- Exchange (14)
      +- MergeRows (13)
         +- Project (12)
            +- SortMergeJoin RightOuter (11)
               :- * Sort (6)
               :  +- Exchange (5)
               :     +- * Filter (4)
               :        +- * Project (3)
               :           +- * ColumnarToRow (2)
               :              +- BatchScan dst (1)
               +- Sort (10)
                  +- Exchange (9)
                     +- Project (8)
                        +- BatchScan src (7)
```



Merge-On-Read

Обновление данных



Проблематика

Читаем	100 ТБ
Shuffle	500 ТБ
Сортируем	1000 ТБ
Фильтруем	1000 ТБ
Сеть	500 ТБ

ОЗУ для сортировок
ЦПУ для сортировок
Сеть для shuffle
ЦПУ для shuffle
IO для shuffle

Проблематика

Начало	
Executors	150
RAM	64 ГБ
Vcore	6
RAMS	9600 ГБ
Vcores	900
Время	более 12 часов

Эксплуатация	
Executors	100
RAM	32 ГБ
Vcore	10
RAMS	3200 ГБ
Vcores	1000
Время	2 часа

План доклада:

1. Проблематика

2. Использование регулярных структур

3. Что такое вероятностные структуры данных

4. Использование фильтров Блума

5. Ленивый сегментный фильтр

6. Выводы

Фильтрация списками

```
merge into dst
 using  src
  on dst.id = src.id
and dst.id in (2000000,2000001,….)
 when matched then update set *
 when not matched  then insert *
```

Фильтрация списками

Merge-On-Read

Фильтрация списками

```
== Physical Plan ==
WriteDelta (15)
+- Exchange (14)
   +- MergeRows (13)
      +- * Project (12)
         +- * SortMergeJoin RightOuter (11)
            :- * Sort (6)
            :  +- Exchange (5)
            :     +- * Project (4)
            :        +- * Filter (3)
            :           +- * ColumnarToRow (2)
            :              +- BatchScan dst (1)
            +- * Sort (10)
               +- Exchange (9)
                  +- * ColumnarToRow (8)
                     +- BatchScan src (7)
```

Merge-On-Read

Фильтрация списками

```
== Physical Plan ==
WriteDelta (15)
+- Exchange (14)
   +- MergeRows (13)
      +- * Project (12)
         +- * SortMergeJoin RightOuter (11)
            :- * Sort (6)
            :  +- Exchange (5)
            :     +- * Project (4)
            :        +- * Filter (3)
            :           +- * ColumnarToRow (2)
            :              +- BatchScan dst (1)
            +- * Sort (10)
               +- Exchange (9)
                  +- * ColumnarToRow (8)
                     +- BatchScan src (7)
```

(1) BatchScan dst
Output [5]: [id, _file, _pos, _spec_id, _partition]
dst (branch=null)
[**filters**=id IN (2000000, 2000001), groupedBy=]

Merge-On-Read

Фильтрация списками

```
== Physical Plan ==
WriteDelta (15)
+- Exchange (14)
   +- MergeRows (13)
      +- * Project (12)
         +- * SortMergeJoin RightOuter (11)
            :- * Sort (6)
            :  +- Exchange (5)
            :     +- * Project (4)
            :        +- * Filter (3)
            :           +- * ColumnarToRow (2)
            :              +- BatchScan dst (1)
            +- * Sort (10)
               +- Exchange (9)
                  +- * ColumnarToRow (8)
                     +- BatchScan src (7)
```

(3) Filter [codegen id : 1]
Input [5]: [id, _file, _pos, _spec_id, _partition]
Condition : id IN (2000000,2000001)

Merge-On-Read

Проблематика

+	−
Эффективно применяется на стороне DataSource	Ограничение на количество ключей (2000)
Используется оптимизатором	Отдельный запрос для создания списка
Стандартное решение	Генерация SQL на «лету»

Broadcast Hash Join

```
SELECT

  /*+ BROADCAST(src) */
  *
FROM
  dst
  JOIN src ON dst.id = src.id
```

Broadcast Hash Join

```
SELECT
  /*+ BROADCAST(src) */
  *
FROM
  dst
  JOIN src ON dst.id = src.id
```

Hint
Объявить таблицу SRC как широковещательную

Broadcast Hash Join

```
== Physical Plan ==
* HashAggregate (10)
+- Exchange (9)
   +- * HashAggregate (8)
      +- * Project (7)
         +- * BroadcastHashJoin Inner BuildRight (6)
            :- * ColumnarToRow (2)
            :  +- BatchScan iceberg.smart.dst (1)
            +- BroadcastExchange (5)
               +- * ColumnarToRow (4)
                  +- BatchScan iceberg.smart.src (3)
```

Broadcast Hash Join

```
== Physical Plan ==
* HashAggregate (10)
+- Exchange (9)
   +- * HashAggregate (8)
      +- * Project (7)
         +- * BroadcastHashJoin Inner BuildRight (6)
            :- * ColumnarToRow (2)
            :  +- BatchScan iceberg.smart.dst (1)
            +- BroadcastExchange (5)
               +- * ColumnarToRow (4)
                  +- BatchScan iceberg.smart.src (3)
```

(5) BroadcastExchange
Input [1]: [id]
Arguments:
HashedRelationBroadcastMode(List(input[0, bigint, false]),false)

Broadcast Hash Join

```
== Physical Plan ==
* HashAggregate (10)
+- Exchange (9)
   +- * HashAggregate (8)
      +- * Project (7)
         +- * BroadcastHashJoin Inner BuildRight (6)
            :- * ColumnarToRow (2)
            :  +- BatchScan iceberg.smart.dst (1)
            +- BroadcastExchange (5)
               +- * ColumnarToRow (4)
                  +- BatchScan iceberg.smart.src (3)
```

(6) BroadcastHashJoin
Left keys [1]: [id]
Right keys [1]: [id]
Join type: Inner
Join condition: None

Broadcast Hash Join

+	−
Нет сортировки	Размер ограничен 8 Гб (1 млн ключей)
Нет shuffle	Нет возможности использовать в MERGE
Стандартное решение	Повышенный расход ОЗУ

Дополнительная проблематика

IN — высокая эффективность, но очень малая применимость из за ограничения количества ключей

SortMergeJoin — большое потребление ресурсов, низкая эффективность. Нет ограничения на размер

BroadcatHashJoin — более низкая эффективность, но более высокое количество ключей. Не везде можно использовать.

Необходимы структуры которые более плотные, возможно с ошибками при фильтрации.

План доклада:

1. Проблематика

2. Использование регулярных структур

3. Что такое вероятностные структуры данных

4. Использование фильтров Блума

5. Ленивый сегментный фильтр

6. Выводы

Вероятностные структуры

Структуры данных или алгоритмы, результатом которых является не детерминированное «да» или «нет», а вероятностные ответы, например, «точно нет» и «возможно»

FPP — ошибка неправильного положительного ответа

N — Количество уникальных ключей

BloomFilter — классическая реализация,

XorFilter, XorPlusFilter — более современная реализация, не работает на потоковых данных.

https://arxiv.org/pdf/1912.08258

План доклада:

1. **Проблематика**
2. **Использование регулярных структур**
3. **Что такое вероятностные структуры данных**
4. **Использование фильтров Блума**
5. Ленивый сегментный фильтр
6. Выводы

Bloom Filter Join

Spark 3 и выше
Только таблицы
Наличие статистики
Автоматическое создание

spark.sql.optimizer.runtime
bloomFilter.enabled
bloomFilter.maxNumItems
bloomFilter.creationSideThreshold
bloomFilter.maxNumBits

Bloom Filter Join

```
analyze table src compute statistics for columns id

analyze table dst compute statistics for columns id
```

```
Select *
  from dst
  join src
  on dst.id = src.id
where src.id > 0
```

Bloom Filter Join

```
analyze table src compute statistics for columns id

analyze table dst compute statistics for columns id
```

```
Select *
  from dst
  join src
  on dst.id = src.id
where src.id > 0
```

Предикат
Условие срабатывания оптимизации

Bloom Filter Join

```
== Physical Plan ==
* HashAggregate (15)
+- Exchange (14)
   +- * HashAggregate (13)
      +- * Project (12)
         +- * SortMergeJoin Inner (11)
            :- * Sort (5)
            :  +- Exchange (4)
            :     +- * Filter (3)
            :        +- * ColumnarToRow (2)
            :           +- Scan parquet spark_catalog.default.dst (1)
            +- * Sort (10)
               +- Exchange (9)
                  +- * Filter (8)
                     +- * ColumnarToRow (7)
                        +- Scan parquet spark_catalog.default.src (6)
```

Filter

Condition : (((id> 0) AND isnotnull(id)) AND **might_contain**(**Subquery scalar-subquery**, [id=], xxhash64(id, 42)))

Bloom Filter Join

```
Subquery:1 Hosting operator id = 3 Hosting Expression
= Subquery scalar-subquery, [id]
ObjectHashAggregate (21)
+- Exchange (20)
   +- ObjectHashAggregate (19)
      +- * Filter (18)
         +- * ColumnarToRow (17)
            +- Scan parquet spark_catalog.default.src (16)
```

Bloom Filter Join

```
Subquery:1 Hosting operator id = 3 Hosting
Expression = Subquery scalar-subquery, [id]
ObjectHashAggregate (21)
+- Exchange (20)
   +- ObjectHashAggregate (19)
      +- * Filter (18)
         +- * ColumnarToRow (17)
            +- Scan parquet spark_catalog.default.src (16)
```

ObjectHashAggregate
Input [1]: [id]
Keys: []
Functions [1]:
[**partial_bloom_filter_agg**(xx
hash64(id, 42), 1000000,
8388608, 0, 0)]
Aggregate Attributes [1]: [buf]
Results [1]: [buf]

Bloom Filter Join

```
Subquery:1 Hosting operator id = 3 Hosting
Expression = Subquery scalar-subquery, [id]
ObjectHashAggregate (21)
+- Exchange (20)
   +- ObjectHashAggregate (19)
      +- * Filter (18)
         +- * ColumnarToRow (17)
            +- Scan parquet spark_catalog.default.src (16)
```

```
(21) ObjectHashAggregate
Input [1]: [buf]
Keys: []
Functions [1]:
[bloom_filter_agg(xxhash64(id, 42),
1000000, 8388608, 0, 0)]
Aggregate Attributes [1]:
[bloom_filter_agg(xxhash64(id, 42),
1000000, 8388608, 0, 0)]
Results [1]:
[bloom_filter_agg(xxhash64(id, 42),
1000000, 8388608, 0, 0) AS
bloomFilter#2439]
```

Bloom Filter Join

```
== Physical Plan ==
* HashAggregate (15)
+- Exchange (14)
   +- * HashAggregate (13)
      +- * Project (12)
         +- * SortMergeJoin Inner (11)
            :- * Sort (5)
            :  +- Exchange (4)
            :     +- * Filter (3)
            :        +- * ColumnarToRow (2)
            :           +- Scan parquet spark_catalog.default.dst (1)
            +- * Sort (10)
               +- Exchange (9)
                  +- * Filter (8)
                     +- * ColumnarToRow (7)
                        +- Scan parquet spark_catalog.default.src (6)
```

Filter

Condition : (((id> 0) AND isnotnull(id)) AND **might_contain(Subquery scalar-subquery**, [id=], xxhash64(id, 42)))

Bloom Filter Join

+	−
Применяется автоматически	Только на таблицах Требуется статистика
Есть параметры для настройки	Необходим предикат по ключу
Входит в поставку	Не работает с Apache Iceberg

План доклада:

1. **Проблематика**
2. **Использование регулярных структур**
3. **Что такое вероятностные структуры данных**
4. **Использование фильтров Блума**
5. Ленивый сегментный фильтр
6. Выводы

Bloom Filter Join

Применять с любыми источниками данных

Использовать в любых операциях

Более универсальные настройки

Комплексные ключи

Фильтрация не только по ключам Join

Bloom Filter Join

Создание BloomFilter

```
val bf = spark.read.table("src")
         .stat.bloomFilter(col("id"), 400000000,0.02)

val bbf = spark.sparkContext.Broadcast(bf)
```

Bloom Filter Join

Создание BloomFilter

метод Spark 2

```
val bf = spark.read.table("src")
        .stat.bloomFilter(col("id"), 400000000,0.02)

val bbf = spark.sparkContext.Broadcast(bf)
```

Bloom Filter Join

Создание BloomFilter

Столбец для ключей

```
val bf = spark.read.table("src")
         .stat.bloomFilter(col("id"), 400000000,0.02)

val bbf = spark.sparkContext.Broadcast(bf)
```

Bloom Filter Join

Создание BloomFilter

```
val bf = spark.read.table("src")
        .stat.bloomFilter(col("id"), 400000000,0.02)

val bbf = spark.sparkContext.Broadcast(bf)
```

Столбец для ключей

Количество уникальных ключей

Bloom Filter Join

Создание BloomFilter

```
val bf = spark.read.table("src")
        .stat.bloomFilter(col("id"), 400000000,0.02)

val bbf = spark.sparkContext.Broadcast(bf)
```

Столбец для ключей

Количество уникальных ключей

Ошибка FPP 2%

Bloom Filter Join

Создание BloomFilter

```
val bf = spark.read.table("src")
        .stat.bloomFilter(col("id"), 400000000, 0.02)

val bbf = spark.sparkContext.Broadcast(bf)
```

Столбец для ключей

Количество уникальных ключей

Ошибка FPP 2%

Распространяем на экзекуторы

Bloom Filter Join

Создание BloomFilter

```
val bf = spark.read.table("src")
        .stat.bloomFilter(col("id"), 400000000,0.02)

val bbf = spark.sparkContext.Broadcast(bf)
```

Создание UserDefinedFunction

```
spark.udf.register("filter", (id: Long) ⇒
bbf.value.mightContainLong(id), BooleanType)
```

Bloom Filter Join

Создание BloomFilter

```
val bf = spark.read.table("src")
        .stat.bloomFilter(col("id"), 400000000,0.02)

val bbf = spark.sparkContext.Broadcast(bf)
```

Создание UserDefinedFunction

```
spark.udf.register("filter", (id: Long) ⇒
bbf.value.mightContainLong(id), BooleanType)
```

Имя функции filter

Bloom Filter Join

Создание BloomFilter

```
val bf = spark.read.table("src")
        .stat.bloomFilter(col("id"), 400000000,0.02)

val bbf = spark.sparkContext.Broadcast(bf)
```

Создание UserDefinedFunction

```
spark.udf.register("filter", (id: Long) ⇒
bbf.value.mightContainLong(id), BooleanType)
```

Имя функции filter

Параметр

Bloom Filter Join

Создание BloomFilter

```
val bf = spark.read.table("src")
        .stat.bloomFilter(col("id"), 400000000,0.02)

val bbf = spark.sparkContext.Broadcast(bf)
```

Создание UserDefinedFunction

```
spark.udf.register("filter", (id: Long) ⇒
bbf.value.mightContainLong(id), BooleanType)
```

Имя функции filter

Параметр

BloomFilter

Bloom Filter Join

Создание BloomFilter

```
val bf = spark.read.table("src")
         .stat.bloomFilter(col("id"), 400000000,0.02)

val bbf = spark.sparkContext.Broadcast(bf)
```

Создание UserDefinedFunction

```
spark.udf.register("filter", (id: Long) ⇒
bbf.value.mightContainLong(id), BooleanType)
```

Имя функции filter

Параметр

BloomFilter

Возвращаемое значение

Bloom Filter Join

Создание BloomFilter

```
val bf = spark.read.table("src")
        .stat.bloomFilter(col("id"), 400000000,0.02)

val bbf = spark.sparkContext.Broadcast(bf)
```

Создание UserDefinedFunction

```
spark.udf.register("filter", (id: Long) ⇒
bbf.value.mightContainLong(id), BooleanType)
```

Bloom Filter Join

```
merge into dst
 using  src
  on dst.id = src.id and filter(dst.id)
 when matched then update set *
 when not matched  then insert *
```

Bloom Filter Join

```
merge into dst
 using  src
  on dst.id = src.id and filter(dst.id)
 when matched then update set *
 when not matched  then insert *
```

Имя функции filter

Bloom Filter Join

```
== Physical Plan ==
WriteDelta (15)
+- Exchange (14)
   +- MergeRows (13)
      +- * Project (12)
         +- * SortMergeJoin RightOuter (11)
            :- * Sort (6)
            :  +- Exchange (5)
            :     +- * Project (4)
            :        +- * Filter (3)
            :           +- * ColumnarToRow (2)
            :              +- BatchScan dst (1)
            +- * Sort (10)
               +- Exchange (9)
                  +- * ColumnarToRow (8)
                     +- BatchScan src (7)
```

Merge-On-Read

Bloom Filter Join

```
== Physical Plan ==
WriteDelta (15)
+- Exchange (14)
   +- MergeRows (13)
      +- * Project (12)
         +- * SortMergeJoin RightOuter (11)
            :- * Sort (6)
            :  +- Exchange (5)
            :     +- * Project (4)
            :        +- * Filter (3)
            :           +- * ColumnarToRow (2)
            :              +- BatchScan dst (1)
            +- * Sort (10)
               +- Exchange (9)
                  +- * ColumnarToRow (8)
                     +- BatchScan src (7)
```

Фильтрация данных
(3) Filter [codegen id : 1]
Input [5]: [id, _file, _pos, _spec_id, _partition]
Condition : **filter**(id)

Merge-On-Read

Bloom Filter Join

```
== Physical Plan ==
WriteDelta (15)
+- Exchange (14)
   +- MergeRows (13)
      +- * Project (12)
         +- * SortMergeJoin RightOuter (11)
            :- * Sort (6)
            :  +- Exchange (5)
            :     +- * Project (4)
            :        +- * Filter (3)
            :           +- * ColumnarToRow (2)
            :              +- BatchScan dst (1)
            +- * Sort (10)
               +- Exchange (9)
                  +- * ColumnarToRow (8)
                     +- BatchScan src (7)
```

Чтение DST: 100 ТБ

Чтение SRC: 0.150 ТБ

Merge-On-Read

Bloom Filter Join

```
== Physical Plan ==
WriteDelta (15)
+- Exchange (14)
   +- MergeRows (13)
      +- * Project (12)
         +- * SortMergeJoin RightOuter (11)
            :- * Sort (6)
            :  +- Exchange (5)
            :     +- * Project (4)
            :        +- * Filter (3)
            :           +- * ColumnarToRow (2)
            :              +- BatchScan dst (1)
            +- * Sort (10)
               +- Exchange (9)
                  +- * ColumnarToRow (8)
                     +- BatchScan src (7)
```

Фильтрация:
Вход 1000 ТБ
Выход 20 ТБ

Merge-On-Read

Bloom Filter Join

```
== Physical Plan ==
WriteDelta (15)
+- Exchange (14)
   +- MergeRows (13)
      +- * Project (12)
         +- * SortMergeJoin RightOuter (11)
            :- * Sort (6)
            :  +- Exchange (5)
            :     +- * Project (4)
            :        +- * Filter (3)
            :           +- * ColumnarToRow (2)
            :              +- BatchScan dst (1)
            +- * Sort (10)
               +- Exchange (9)
                  +- * ColumnarToRow (8)
                     +- BatchScan src (7)
```

Shuffle: 3 ТБ

Shuffle: 0,5 ТБ

Merge-On-Read

Bloom Filter Join

```
== Physical Plan ==
WriteDelta (15)
+- Exchange (14)
   +- MergeRows (13)
      +- * Project (12)
         +- * SortMergeJoin RightOuter (11)
            :- * Sort (6)
            :  +- Exchange (5)
            :     +- * Project (4)
            :        +- * Filter (3)
            :           +- * ColumnarToRow (2)
            :              +- BatchScan dst (1)
            +- * Sort (10)
               +- Exchange (9)
                  +- * ColumnarToRow (8)
                     +- BatchScan src (7)
```

Сортировка: 20 ТБ

Сортировка: 1 ТБ

Merge-On-Read

Bloom Filter Join

```
== Physical Plan ==
WriteDelta (15)
+- Exchange (14)
   +- MergeRows (13)
      +- * Project (12)
         +- * SortMergeJoin RightOuter (11)
            :- * Sort (6)
            :  +- Exchange (5)
            :     +- * Project (4)
            :        +- * Filter (3)
            :           +- * ColumnarToRow (2)
            :              +- BatchScan dst (1)
            +- * Sort (10)
               +- Exchange (9)
                  +- * ColumnarToRow (8)
                     +- BatchScan src (7)
```

Join:

Вход 21 ТБ

Выход 2 ТБ

Merge-On-Read

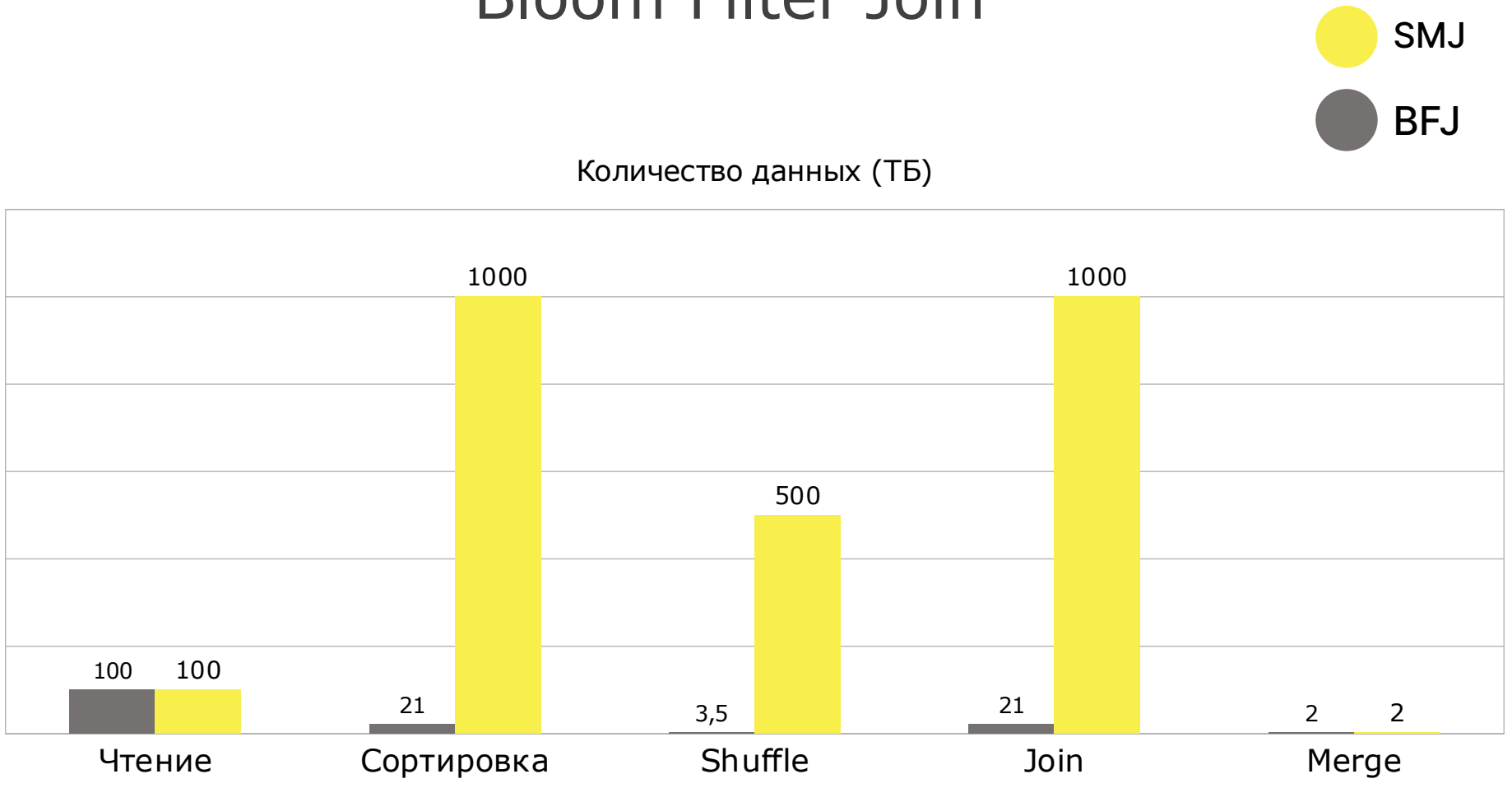
Bloom Filter Join

```
== Physical Plan ==
WriteDelta (15)
+- Exchange (14)
   +- MergeRows (13)
      +- * Project (12)
         +- * SortMergeJoin RightOuter (11)
            :- * Sort (6)
            :  +- Exchange (5)
            :     +- * Project (4)
            :        +- * Filter (3)
            :           +- * ColumnarToRow (2)
            :              +- BatchScan dst (1)
            +- * Sort (10)
               +- Exchange (9)
                  +- * ColumnarToRow (8)
                     +- BatchScan src (7)
```

Merge:
2 ТБ

Merge-On-Read

Bloom Filter Join



Bloom Filter Join

- Кардинальное снижение количества данных
- Универсальность применения
- Базовые механизмы стандартны
- Распределенное создание фильтра
- Уменьшение потребление ресурсов кластера
- Время работы 3-4 часа

- Не только SQL
- Потребление памяти на исполнителях, 1 Гб
- Есть ограничение на оптимальный размер ключей

Bloom Filter Join

- Позволяет кардинально снизить количество обрабатываемых данных.
- !Ключи > 400 млн. Fpp » 100%
- ! При целевом значении ключей в 4000 млн., качество фильтрации значительно ухудшается

План доклада:

1. **Проблематика**
2. **Использование регулярных структур**
3. **Что такое вероятностные структуры данных**
4. **Использование фильтров Блума**
5. **Ленивый сегментный фильтр**
6. Выводы

Ленивый Сегментный фильтр

Использование разных структур

Максимально быстрые структуры

Возможность загружать только часть

Максимально компактные структуры

Хранение на файловой системе

Распределенное построение структур

Ленивый Сегментный фильтр

Array[Filter](Int)

01 --> Filter для сегмента данных 1
02 --> Filter для сегмента данных 2
03 --> Filter для сегмента данных 3
04 --> Filter для сегмента данных 4
N --> Filter для сегмента данных N

Ленивый Сегментный фильтр

Array[Filter](Int)

XFilter(1,1000000)

01 --> Filter для сегмента данных 1

02 --> Filter для сегмента данных 2

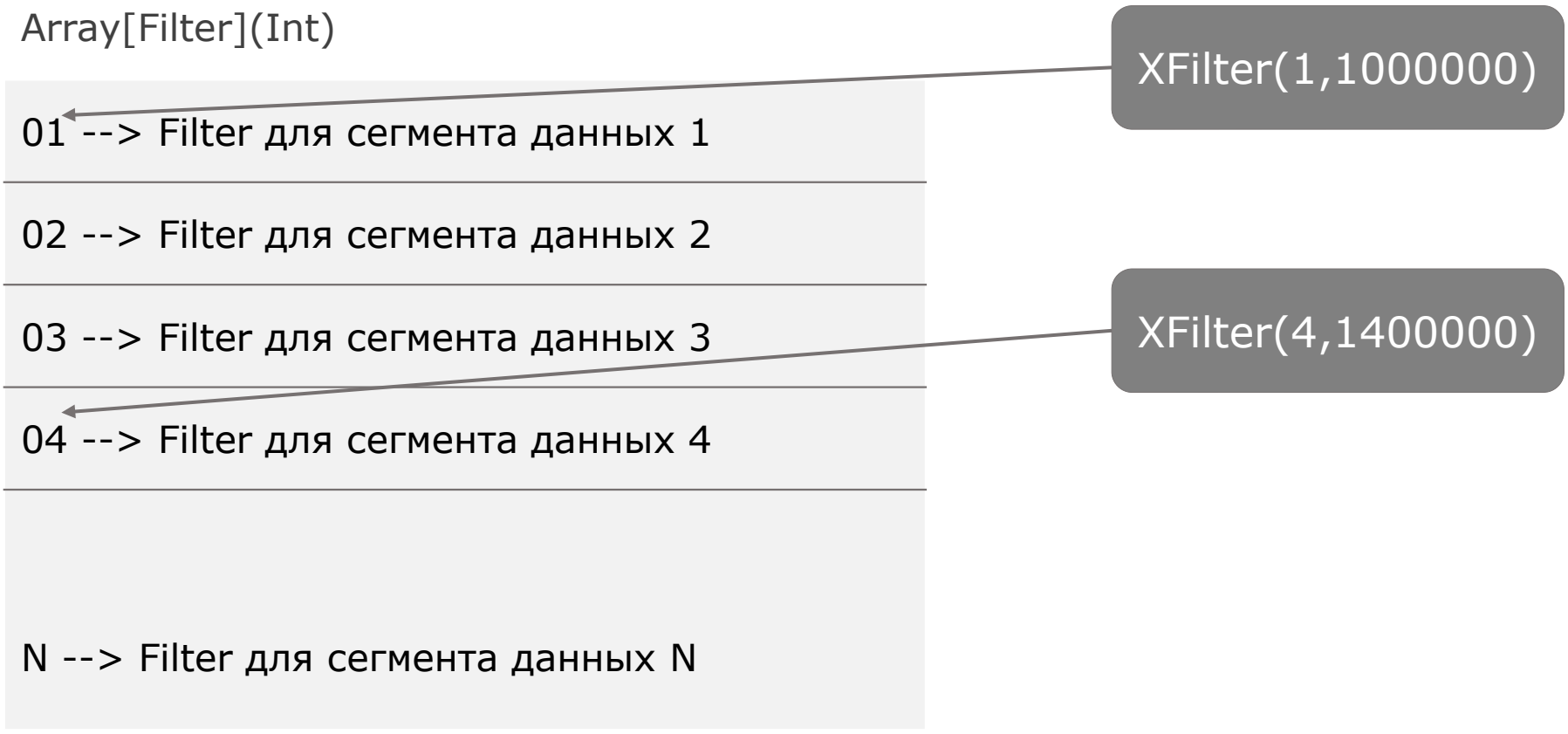
03 --> Filter для сегмента данных 3

04 --> Filter для сегмента данных 4

N --> Filter для сегмента данных N

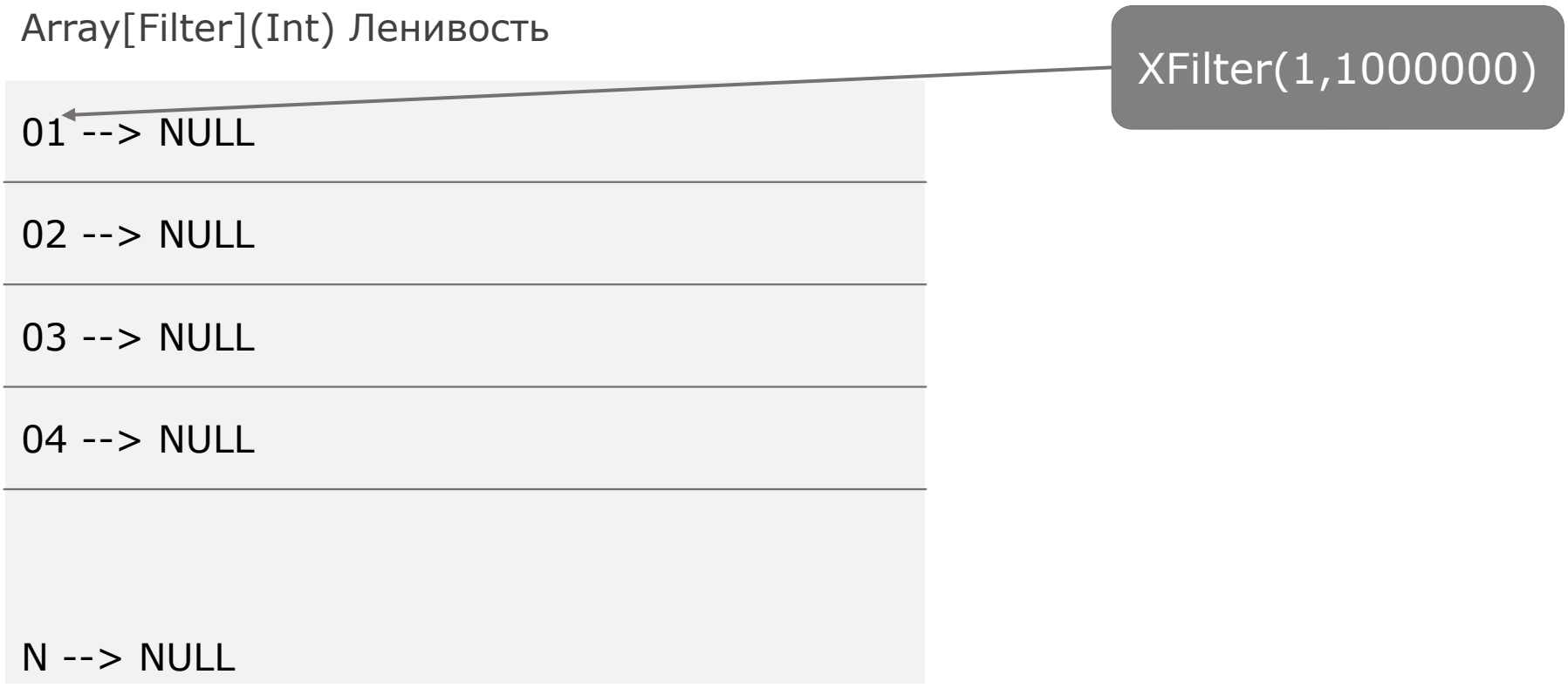
Ленивый Сегментный фильтр

Array[Filter](Int)



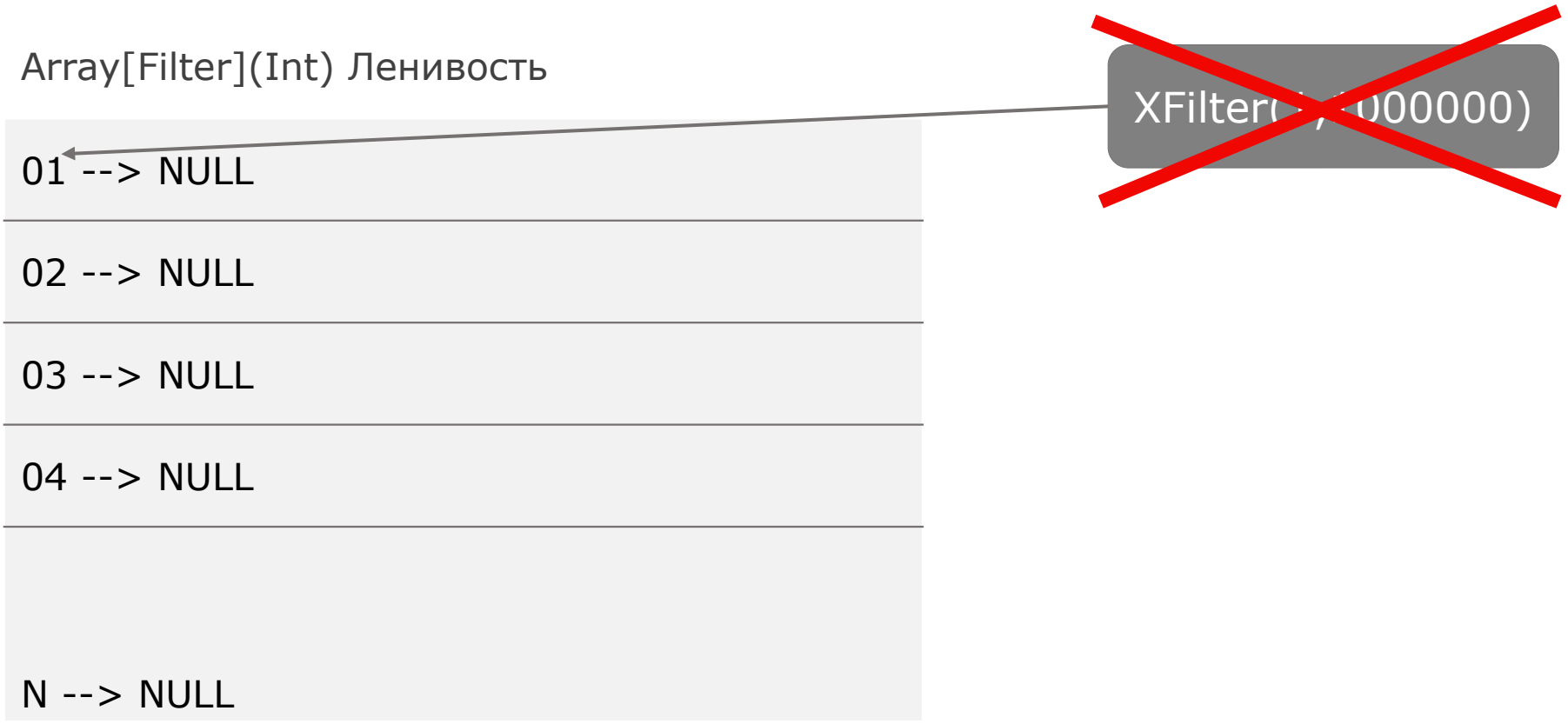
Ленивый Сегментный фильтр

Array[Filter](Int) Ленивость



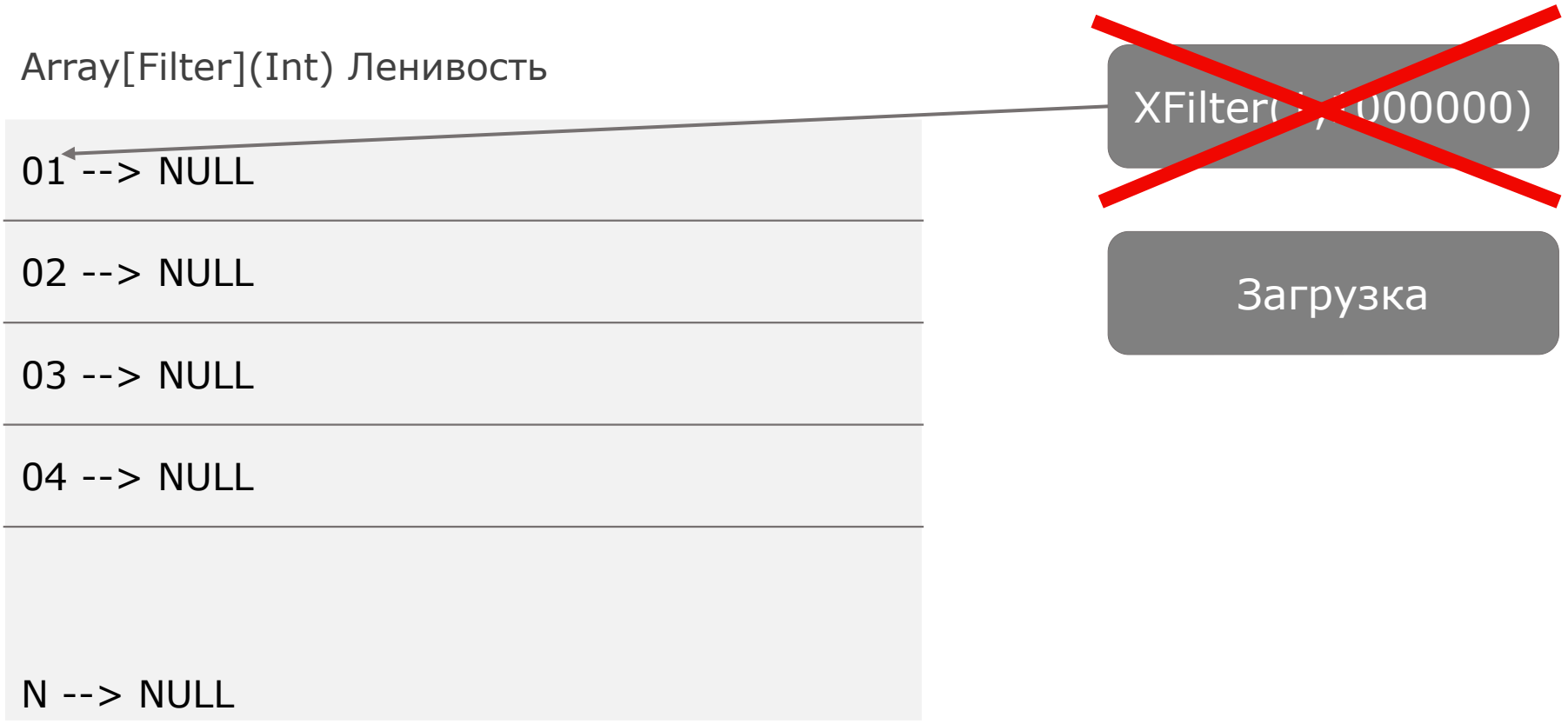
Ленивый Сегментный фильтр

Array[Filter](Int) Ленивость



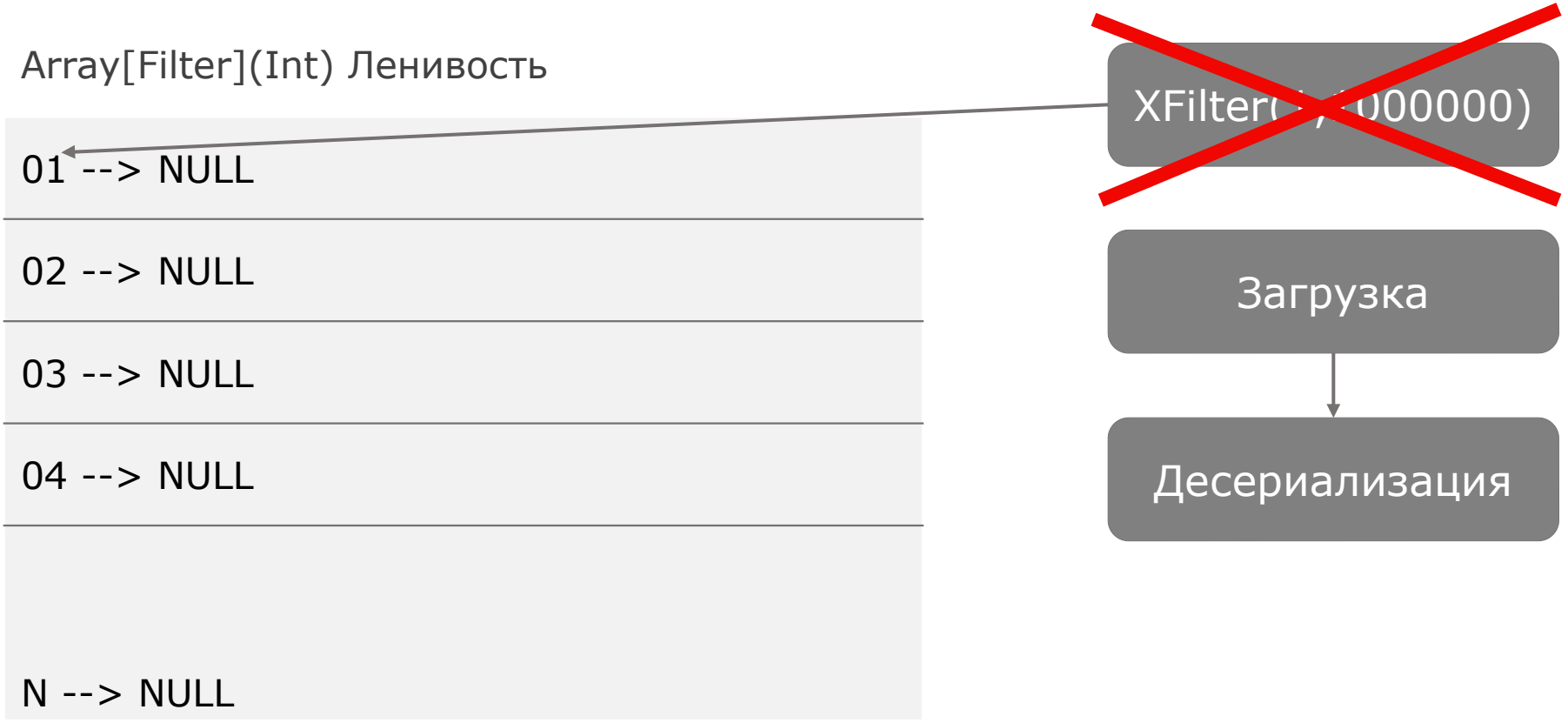
Ленивый Сегментный фильтр

Array[Filter](Int) Ленивость



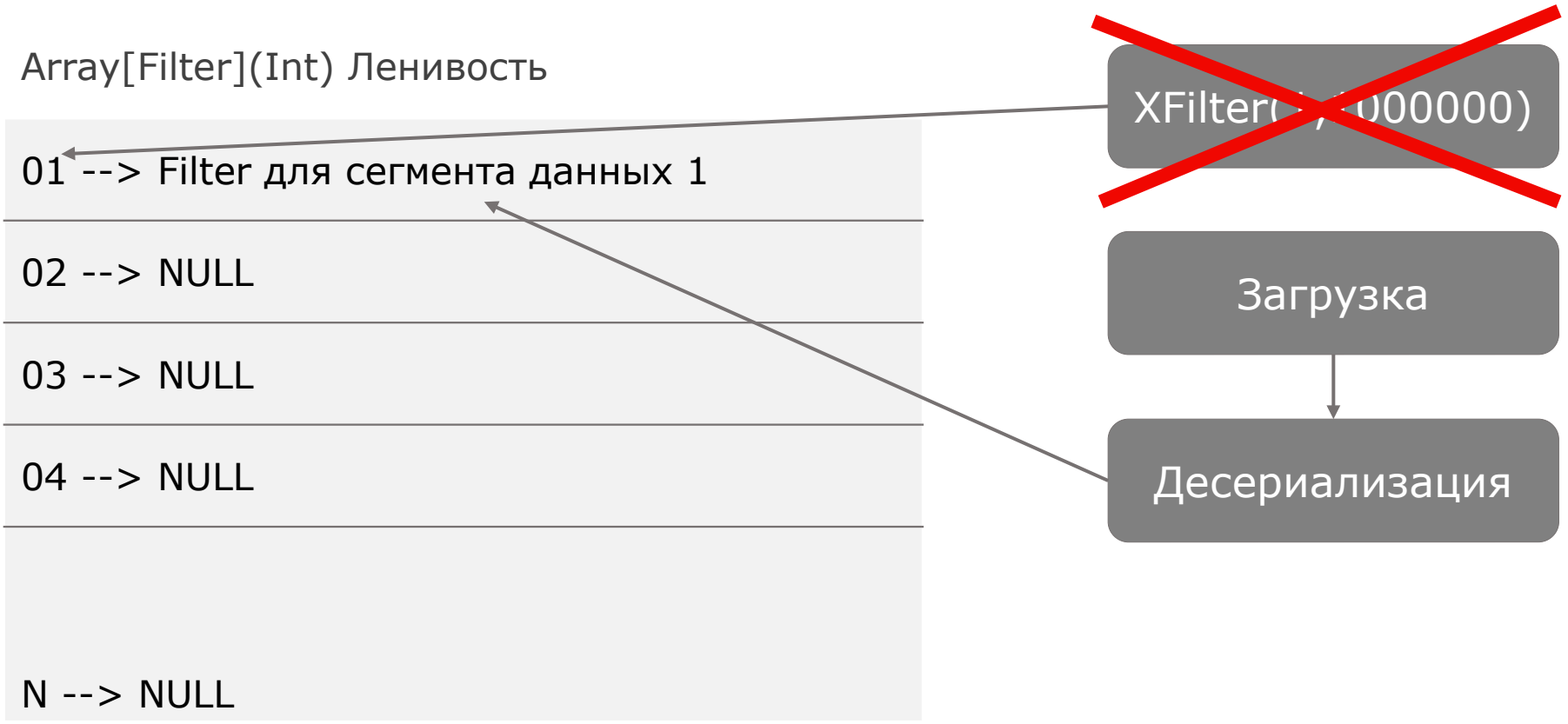
Ленивый Сегментный фильтр

Array[Filter](Int) Ленивость



Ленивый Сегментный фильтр

Array[Filter](Int) Ленивость

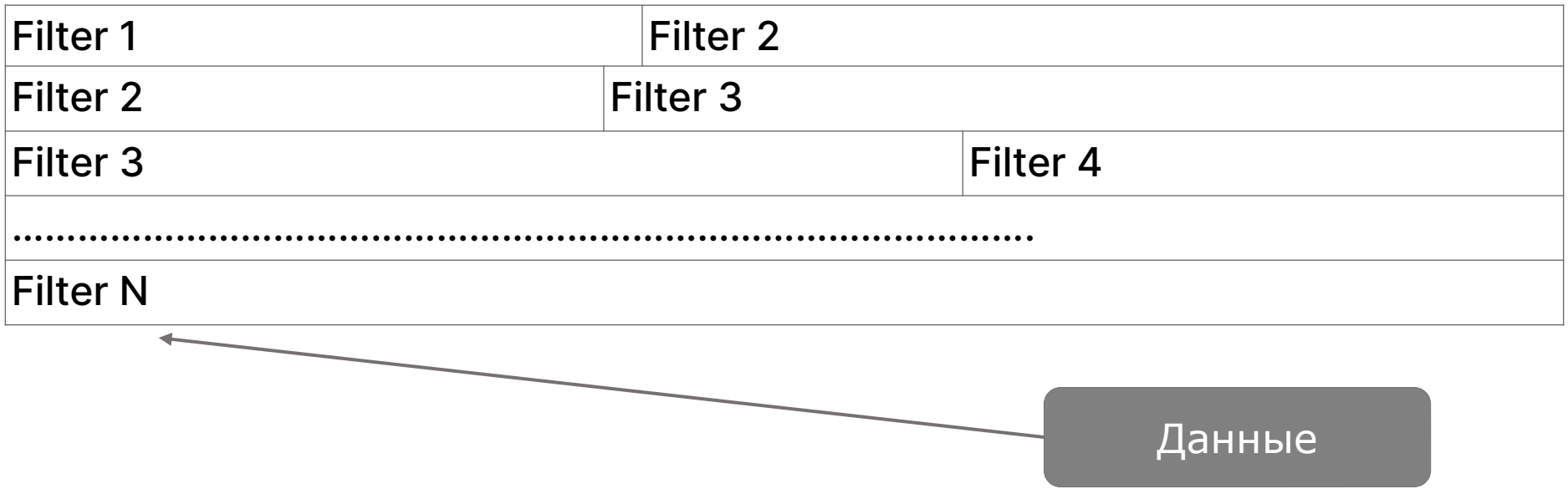


Ленивый Сегментный фильтр

Array[Filter](Int) Ленивость

Ленивый Сегментный фильтр

Хранение



Ленивый Сегментный фильтр

Хранение

Filter 1	Filter 2
Filter 2	Filter 3
Filter 3	Filter 4
.....	
Filter N	

1	Смещение	Размер	hash64	2	Смещение	Размер	hash64	3	Смещение
...									
...						N	Смещение	Размер	hash64

Метаданные

Ленивый Сегментный фильтр

Хранение

Filter 1 Filter 2
Filter 2 Filter 3
Filter 3 Filter 4
.....
Filter N

1	Смещение	Размер	hash64	2	Смещение	Размер	hash64	3	Смещение
...									
... Размер метаданных						N	Смещение	Размер	hash64

Метаданные

Ленивый Сегментный фильтр

Хранение

Filter 1	Filter 2
Filter 2	Filter 3
Filter 3	Filter 4
.....	
Filter N	

1	Смещение	Размер	hash64	2	Смещение	Размер	hash64	3	Смещение
...									
... Размер метаданных						N	Смещение	Размер	hash64

Ленивый Сегментный фильтр

Хранение

Filter 1	
Filter 2	Filter 3
Filter 3	
.....	
Filter N	

1	Смещение	Размер	hash64	2	Смещени
...					
...					
Размер метаданных					

Открываем Stream

Ленивый Сегментный фильтр

Хранение

Filter 1	
Filter 2	Filter 3
Filter 3	
.....	
Filter N	

1	Смещение	Размер	hash64	2	Смещени
...					
...					

Размер метаданных

Открываем Stream

Читаем размер

Ленивый Сегментный фильтр

Хранение

Filter 1
Filter 2 Filter 3
Filter 3
.....
Filter N

1	Смещение	Размер	hash64	2	Смещени
...					
...					
Размер метаданных					

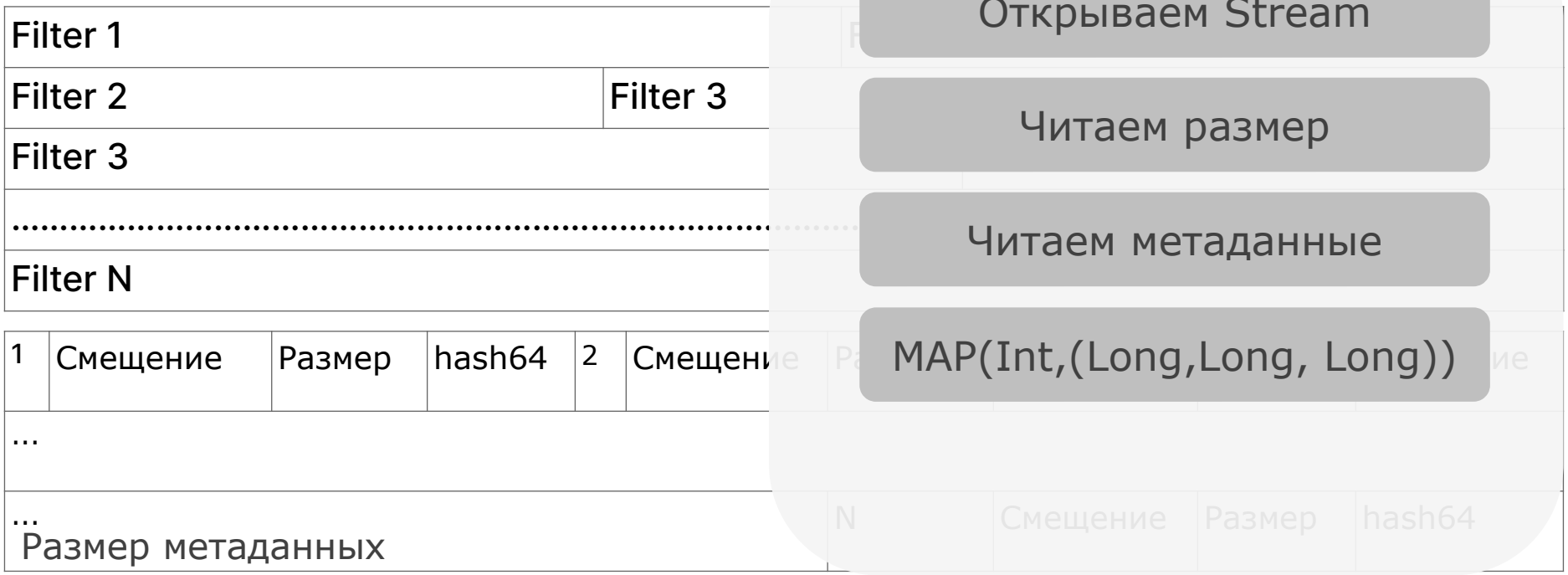
Открываем Stream

Читаем размер

Читаем метаданные

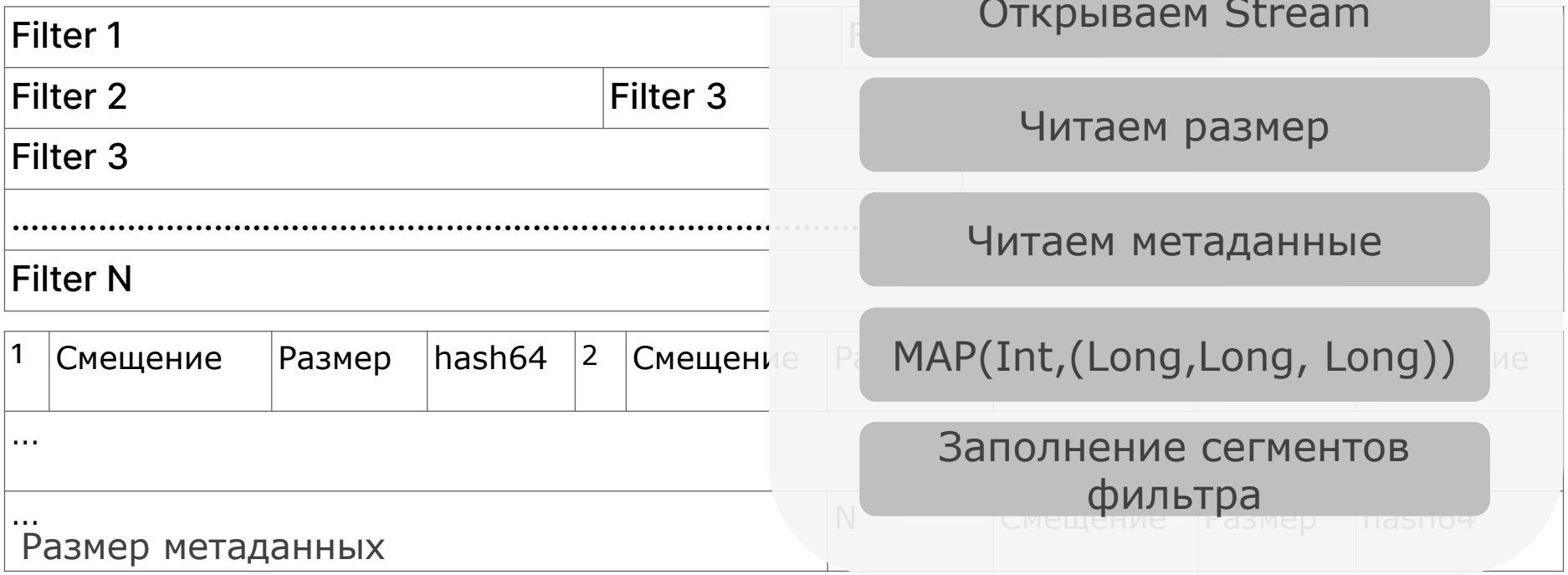
Ленивый Сегментный фильтр

Хранение



Ленивый Сегментный фильтр

Хранение



Открываем Stream

Читаем размер

Читаем метаданные

MAP(Int,(Long,Long, Long))

Заполнение сегментов фильтра

Ленивый Сегментный фильтр

Создание

Запрос для создания набора данных

```
val request =
      "select id, iceberg.bucket(100,id)
         from iceberg.smart.src"

val file = "/tmp/smart_filter.bin"

val segments = 100

XFilter.collect(request, file, segments)
```

Ленивый Сегментный фильтр

Создание

Ключ фильтра

```
val request =
      "select id, iceberg.bucket(100,id)
         from iceberg.smart.src"

val file = "/tmp/smart_filter.bin"

val segments = 100

XFilter.collect(request, file, segments)
```

Ленивый Сегментный фильтр

Создание

Номер сегмента

```
val request =
      "select id, iceberg.bucket(100,id)
         from iceberg.smart.src"

val file = "/tmp/smart_filter.bin"

val segments = 100

XFilter.collect(request, file, segments)
```

Ленивый Сегментный фильтр

Создание

```
val request =
      "select id, iceberg.bucket(100,id)
         from iceberg.smart.src"

val file = "/tmp/smart_filter.bin"

val segments = 100

XFilter.collect(request, file, segments)
```

Ленивый Сегментный фильтр

Создание

```
val request =
      "select id, iceberg.bucket(100,id)
         from iceberg.smart.src"

val file = "/tmp/smart_filter.bin"

val segments = 100

XFilter.collect(request, file, segments)
```

Количество сегментов

Ленивый Сегментный фильтр

Создание

```
val request =
      "select id, iceberg.bucket(100,id)
         from iceberg.smart.src"

val file = "/tmp/smart_filter.bin"

val segments = 100

XFilter.collect(request, file, segments)
```

Создание ЛСФ

Ленивый Сегментный фильтр

Использование

```
XFilter.load(file, segments, "smartFilter")"
```

Загрузка метаданных и формирование UDF

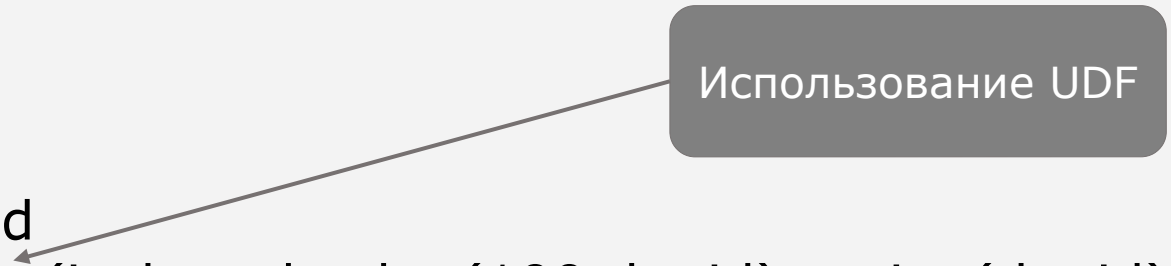
```
merge into dst
 using  src
  on dst.id = src.id
  and  smartFilter(iceberg.bucket(100,dst.id), string(dst.id))
 when matched then update set *
 when not matched  then insert *
```

Ленивый Сегментный фильтр

Использование

```
XFilter.load(file, segments, "smartFilter")"
```

```
merge into dst
 using  src
  on dst.id = src.id
  and  smartFilter(iceberg.bucket(100,dst.id), string(dst.id))
 when matched then update set *
 when not matched  then insert *
```



Ленивый Сегментный фильтр

Использование

```
XFilter.load(file, segments, "smartFilter")"
```

```
merge into dst
 using  src
  on dst.id = src.id
  and  smartFilter(iceberg.bucket(100,dst.id), string(dst.id))
 when matched then update set *
 when not matched  then insert *
```

Функция формирования номеров сегментов

Ленивый Сегментный фильтр

```
== Physical Plan ==
WriteDelta (15)
+- Exchange (14)
   +- MergeRows (13)
      +- * Project (12)
         +- * SortMergeJoin RightOuter (11)
            :- * Sort (6)
            :  +- Exchange (5)
            :     +- * Project (4)
            :        +- * Filter (3)
            :           +- * ColumnarToRow (2)
            :              +- BatchScan dst (1)
            +- * Sort (10)
               +- Exchange (9)
                  +- * ColumnarToRow (8)
                     +- BatchScan src (7)
```

Merge-On-Read

Ленивый Сегментный фильтр

```
== Physical Plan ==
WriteDelta (15)
+- Exchange (14)
   +- MergeRows (13)
      +- * Project (12)
         +- * SortMergeJoin RightOuter (11)
            :- * Sort (6)
            :  +- Exchange (5)
            :     +- * Project (4)
            :        +- * Filter (3)
            :           +- * ColumnarToRow (2)
            :              +- BatchScan dst (1)
            +- * Sort (10)
               +- Exchange (9)
                  +- * ColumnarToRow (8)
                     +- BatchScan src (7)
```

Filter [codegen id : 1]
Input [5]: [id, _file, _pos, _spec_id, _partition]
Condition : **smartFilter**(staticinvoke(class org.apache.iceberg.spark.functions.BucketFunction$BucketLong, IntegerType, invoke, 100, id, IntegerType, LongType, false, true, true), cast(id as string))

Merge-On-Read

Ленивый Сегментный фильтр

```
== Physical Plan ==
WriteDelta (15)
+- Exchange (14)
   +- MergeRows (13)
      +- * Project (12)
         +- * SortMergeJoin RightOuter (11)
            :- * Sort (6)
            :  +- Exchange (5)
            :     +- * Project (4)
            :        +- * Filter (3)
            :           +- * ColumnarToRow (2)
            :              +- BatchScan dst (1)
            +- * Sort (10)
               +- Exchange (9)
                  +- * ColumnarToRow (8)
                     +- BatchScan src (7)
```

Чтение DST: 100 ТБ

Чтение SRC: 0.150 ТБ

Merge-On-Read

Ленивый Сегментный фильтр

```
== Physical Plan ==
WriteDelta (15)
+- Exchange (14)
   +- MergeRows (13)
      +- * Project (12)
         +- * SortMergeJoin RightOuter (11)
            :- * Sort (6)
            :  +- Exchange (5)
            :     +- * Project (4)
            :        +- * Filter (3)
            :           +- * ColumnarToRow (2)
            :              +- BatchScan dst (1)
            +- * Sort (10)
               +- Exchange (9)
                  +- * ColumnarToRow (8)
                     +- BatchScan src (7)
```

Фильтрация:
Вход 1000 ТБ
Выход 1 ТБ

Merge-On-Read

Ленивый Сегментный фильтр

```
== Physical Plan ==
WriteDelta (15)
+- Exchange (14)
   +- MergeRows (13)
      +- * Project (12)
         +- * SortMergeJoin RightOuter (11)
            :- * Sort (6)
            :  +- Exchange (5)
            :     +- * Project (4)
            :        +- * Filter (3)
            :           +- * ColumnarToRow (2)
            :              +- BatchScan dst (1)
            +- * Sort (10)
               +- Exchange (9)
                  +- * ColumnarToRow (8)
                     +- BatchScan src (7)
```

Shuffle: 0,5 ТБ

Shuffle: 0,5 ТБ

Merge-On-Read

Ленивый Сегментный фильтр

```
== Physical Plan ==
WriteDelta (15)
+- Exchange (14)
   +- MergeRows (13)
      +- * Project (12)
         +- * SortMergeJoin RightOuter (11)
            :- * Sort (6)
            :  +- Exchange (5)
            :     +- * Project (4)
            :        +- * Filter (3)
            :           +- * ColumnarToRow (2)
            :              +- BatchScan dst (1)
            +- * Sort (10)
               +- Exchange (9)
                  +- * ColumnarToRow (8)
                     +- BatchScan src (7)
```

Сортировка: 1 ТБ

Сортировка: 1 ТБ

Merge-On-Read

Ленивый Сегментный фильтр

```
== Physical Plan ==
WriteDelta (15)
+- Exchange (14)
   +- MergeRows (13)
      +- * Project (12)
         +- * SortMergeJoin RightOuter (11)
            :- * Sort (6)
            :  +- Exchange (5)
            :     +- * Project (4)
            :        +- * Filter (3)
            :           +- * ColumnarToRow (2)
            :              +- BatchScan dst (1)
            +- * Sort (10)
               +- Exchange (9)
                  +- * ColumnarToRow (8)
                     +- BatchScan src (7)
```

Join:

Вход 2 ТБ

Выход 2 ТБ

Merge-On-Read

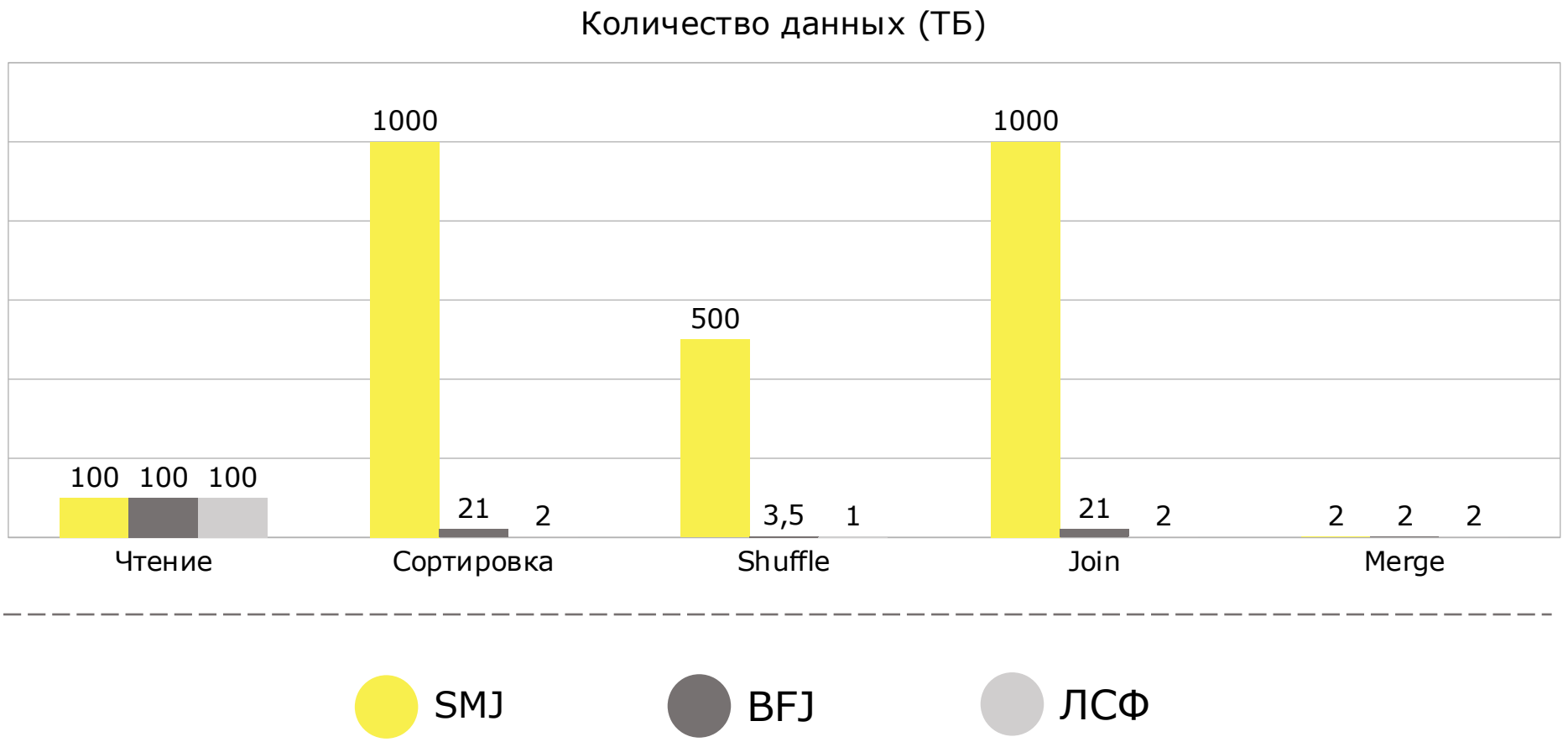
Ленивый Сегментный фильтр

```
== Physical Plan ==
WriteDelta (15)
+- Exchange (14)
   +- MergeRows (13)
      +- * Project (12)
         +- * SortMergeJoin RightOuter (11)
            :- * Sort (6)
            :  +- Exchange (5)
            :     +- * Project (4)
            :        +- * Filter (3)
            :           +- * ColumnarToRow (2)
            :              +- BatchScan dst (1)
            +- * Sort (10)
               +- Exchange (9)
                  +- * ColumnarToRow (8)
                     +- BatchScan src (7)
```

Merge:
2 ТБ

Merge-On-Read

Ленивый Сегментный фильтр



Ленивый Сегментный фильтр

- Кардинальное снижение количества данных
- Универсальность применения
- Уменьшение потребление ресурсов кластера
- Распределенное создание фильтра
- Время работы 1-2 часа
- Нет ограничения на количество ключей

- Не только SQL
- Потребление памяти на исполнителях, до 2 Гб
- Свой механизм создания

План доклада:

1. **Проблематика**
2. **Использование регулярных структур**
3. **Что такое вероятностные структуры данных**
4. **Использование фильтров Блума**
5. **Ленивый сегментный фильтр**
6. **Выводы**

Выводы

Наименование	Стандартный подход	Свой Bloom Filter	Ленивый сегментный фильтр	Результат
Чтение данных, Тб	100	100	100	Паритет
Сортировка данных, Тб	1000	20	2	ЛСФ
Shuffle, Тб	500	2	1	ЛСФ
Join, Тб	1000	20	2	ЛСФ
Время работы, ч	> 12	3 — 4	1,5-2 часа	ЛСФ
Время формирования фильтра, мин.	-	10	16	BF
Размер фильтра, Тб	-	0.001	0.002	BF

Путь

Начало	
Executors	150
RAM	64 ГБ
Vcore	6
RAMS	9600 ГБ
Vcores	900
Время	более 12 часов

Эксплуатация	
Executors	100
RAM	32 ГБ
Vcore	10
RAMS	3200 ГБ
Vcores	1000
Время	2 часа

Выводы

- Высвободили ресурсов по executors 50%
- Сэкономили памяти в 3 раза
- Время выполнения снизили более чем в 6 раз
- Встроенные механизмы не в полной мере подходят к нашим условиям
- Возможно использование встроенных механизмов
- Полученные методы универсальны
- Можно использовать в SQL

Что дальше

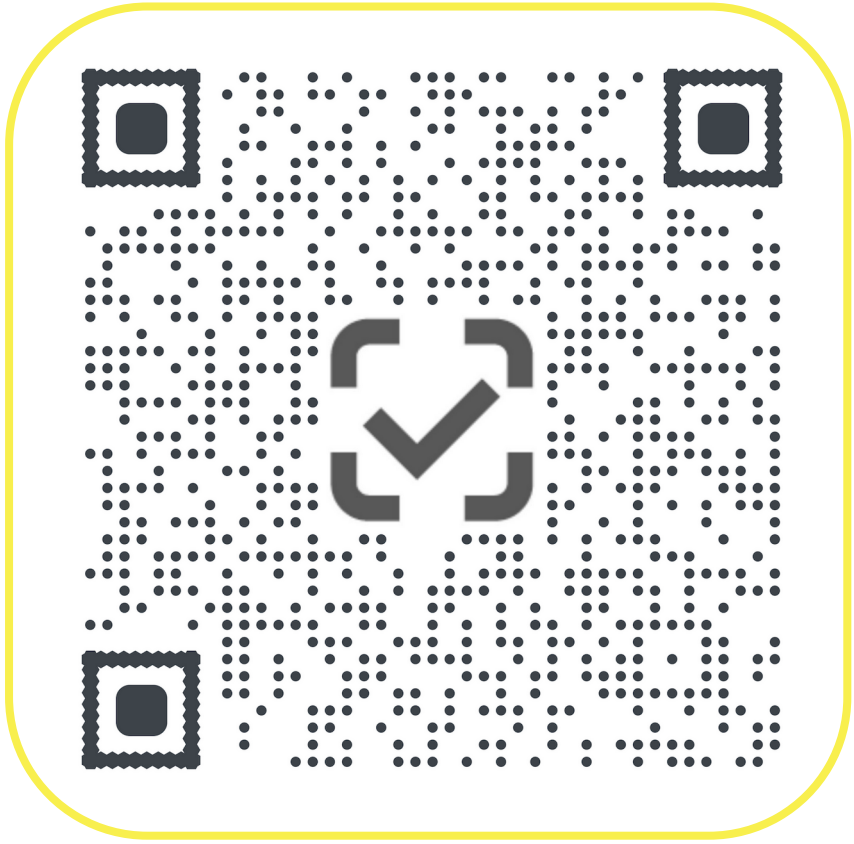
- ЛСФ как как внешнее выражение
- Реализация дополнительных структур в ЛСФ
- Оптимизация дискового хранения ЛСФ
- Стандартизация использования
- Очистка неиспользуемых структур ЛСФ
- Реализация оптимизации для автоматического использования ЛСФ
- При оптимизации запросов использовать сразу несколько подходов и методов

Вопросы

Вертлиб Дмитрий

Ведущий программист разработчик, Честный ЗНАК

d.vertlib@crpt.ru



DE-meetup в СПБ 11/09