ЧТО МЫ ДЕЛАЕМ В NAVIO


SPARK –
ВСЁ!

ИСТОРИЯ SPARK

До Spark был Hadoop

- **Обычные SQL-СУБД не вывозили резко возросшие объёмы данных**
- **В крупных компаниях уже использовали Hadoop MapReduce**

- **Основные проблемы Hadoop MapReduce:**
  - Чтение-запись в распределёную FS на каждой стадии обработки
  - Сложность разработки алгоритмов
  - Тонны бойлерплейт кода

Spark в 2009: MapReduce, только лучше

- **Spark появился в AMPLab в Беркли в рамках магистерской работы Matei Zaharia**
- **Представил концепцию RDD, которая решала проблемы MapReduce:**
  - RDD API – более высокоуровневая и удобная абстракция над MapReduce
  - Код на Scala поддерживал лямбда-функции
  - Можно стыковать стадии обработки последовательно в любом количестве и порядке
  - Результат стадии – в RAM выполнившей её ноды – InMemory!
  - Чтение результата как входа следующей стадии - по сети между нодами

MapReduce vs Spark RDD

```java
public class WordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```



```scala
object WordCountRDD {
  def main(args: Array[String]): Unit = {
    val sc = new SparkContext("local[*]",
"WordCount")

    val textFile = sc.textFile("input.txt")

    val counts = textFile.flatMap(line =>
line.split(" "))
                         .map(word => (word, 1))
                         .reduceByKey(_ + _)

    counts.saveAsTextFile("output")
  }
}
```

Spark в 2010-2024: рост и развитие

- С версии 1.0 (2014 год) InMemory успешно складывается на диск, чтобы запросы не падали по OOM

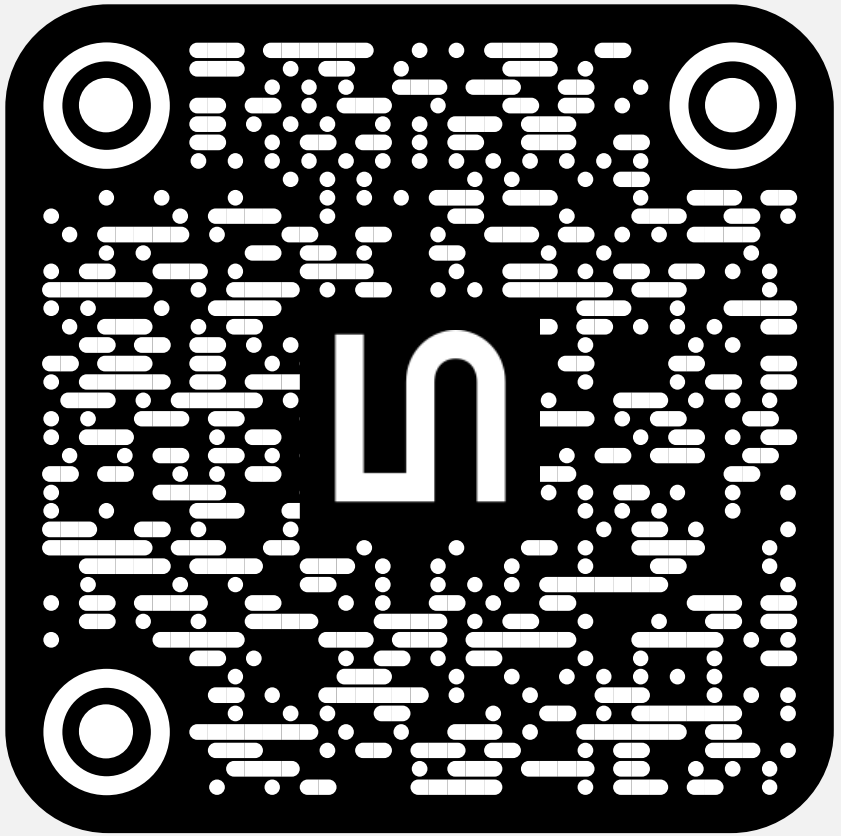
- Spark SQL - более 400 SQL-функций (теперь с улучшенной поддержкой ANSI)
- Dataframe API – полностью эквивалентный SQL
- Единый оптимизирующий SQL-движок
- Spark Streaming
- User-Defined Functions, в том числе векторизованные (Pandas UDF)
- Постепенное расширение поддержки Python

Spark в наше время: эволюция

- **PySpark Arrow UDF**
- **Python DataSource API**
- **Spark Connect API, PySpark-Client**
  - Про них поговорим позже

Apache Arrow

- Фреймворк для колоночных данных
  - Хранение в памяти
  - Обработка
  - Передача между процессами
- Содержит вычислительный движок
  - Низкоуровневый, для использования в других движках
- Реализован на 10 языках программирования
- Унифицирует «общение» систем
  - Данные передаются без конвертации



Май 2025: https://phdays.com/ru/forum/program/?talk-id=1735

PySpark Arrow UDF (3.5)

```
import pyarrow

df = spark.createDataFrame([(1, 21), (2, 30)],
("id", "age"))
def filter_func(iterator):
    for batch in iterator:
        pdf = batch.to_pandas()
        yield \
pyarrow.RecordBatch.from_pandas(pdf[pdf.id == 1])

df.mapInArrow(filter_func, df.schema).show()
```

Ужас! Прямо в доке (*)

- **До версии 4.0 так и работали Arrow UDF — через Pandas UDF API**

* https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.DataFrame.mapInArrow.html

PySpark Arrow UDF (4.0+)

```python
import pyarrow
import pyarrow.compute as pc
df = spark.createDataFrame([(1, 21), (2, 30)],
("id", "age"))
def filter_func(iterator):
    for batch in iterator:
        yield batch.filter(pc.field("id") == 1)

df.mapInArrow(filter_func, df.schema).show()
```

У Arrow есть встроенные compute-функции

- С версии 4.0 наоборот, Pandas UDF работают через Arrow
- Наконец-то можно передавать сложные структурные типы

Python DataSource API (4.0+)

- Позволяет реализовать чтение и запись
- В Batch и Streaming режиме
- Определив несколько классов и методов
- Поддерживает Arrow Batch (но можно и без него)

- Оптимальный способ подключить Spark к чему угодно
  - Никакой компиляции
  - Быстрая разработка
  - Быстрое выполнение (на PyArrow)

```python
from pyspark.sql.datasource import DataSource, DataSourceReader, InputPartition
from pyspark.sql import SparkSession
import pyarrow as pa

# Define the ArrowBatchDataSource
class ArrowBatchDataSource(DataSource):
    """
    A Data Source for testing Arrow Batch Serialization
    """

    @classmethod
    def name(cls):
        return "arrowbatch"

    def schema(self):
        return "key int, value string"

    def reader(self, schema: str):
        return ArrowBatchDataSourceReader(schema, self.options)

# Define the ArrowBatchDataSourceReader
class ArrowBatchDataSourceReader(DataSourceReader):
    def __init__(self, schema, options):
        self.schema: str = schema
        self.options = options

    def read(self, partition):
        # Create Arrow Record Batch
        keys = pa.array([1, 2, 3, 4, 5], type=pa.int32())
        values = pa.array(["one", "two", "three", "four", "five"], type=pa.string())
        schema = pa.schema([("key", pa.int32()), ("value", pa.string())])
        record_batch = pa.RecordBatch.from_arrays([keys, values], schema=schema)
        yield record_batch

    def partitions(self):
        # Define the number of partitions
        num_part = 1
        return [InputPartition(i) for i in range(num_part)]

# Initialize the Spark Session
spark = SparkSession.builder.appName("ArrowBatchExample").getOrCreate()

# Register the ArrowBatchDataSource
spark.dataSource.register(ArrowBatchDataSource)

# Load data using the custom data source
df = spark.read.format("arrowbatch").load()

df.show()
```

ПЛЮСЫ И МИНУСЫ SPARK

Почему DE и компании используют Spark?

- Самый удобный API (максимально приближенный к SQL)
- Лопатит любые объёмы данных (после небольшого тюнинга)
- Читает любые форматы данных (встроенные + DataSource API)
- Огромный выбор встроенных функций и классный UDF API
- Расширяемый (плагины, кастомные операторы и форматы)

Почему возникает желание отказаться от Spark? (1)

- **Spark-сессия поднимается 20 секунд**
- **Очень медленный (особенно при обработке SmallData)**
- **На Scala/Java нет SOTA-библиотек для актуальных задач**
  - С 2015 года, со времён Tensorflow, практически всё уже на Python

Почему возникает желание отказаться от Spark? (2)

- **Большинство Spark-jobs – RAM-Bound**
  - Можно было бы ускорить расчёты за счёт увеличения параллелизма
  - Если бы не приходилось выделять столько оперативки
- **Кондовое управление ресурсами**
  - У всех контейнеров одинаковые ресурсы
  - В K8S у контейнера RAM Request = Limit
- **X5 потребление оперативки**
  - Большую часть времени выделенная RAM не используется
  - Но если оперативки не хватит – таск падает
  - Поэтому приходится выделять много

ЧЕМ УЖЕ ПЫТАЛИСЬ ЗАМЕНИТЬ SPARK?

SmallData: обрабатываем на одной ноде

https://knowyourmeme.com/memes/ok-so-basically-im-very-smol

Polars

- Фреймворк локальной обработки данных на Rust+Python
- Многие уже используют (и мы тоже - *)
- Под капотом – Arrow2 (не Apache)
- DataFrame+LazyFrame API – что-то среднее между Pandas и Spark
- Быстро работает, но на одной ноде (распределённый – только как сервис в AWS)
- Есть SQL-API (но он неправильно работает)
- Мощный механизм Rust-расширений
- Расширение polars-st для геоданных (отдельная либа)

* Сегодня: https://smartdataconf.ru/talks/783ee4bdb1694d1fa8a2314430673770/

DuckDB

- Локальная OLAP SQL-СУБД на C++
- Работает даже быстрее Polars
- Но тоже на одной ноде
- И нет DataFrame API

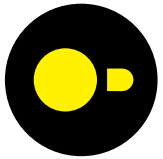
Май 2025: https://phdays.com/ru/forum/program/?talk-id=1736

Для кого оптимально

- Начинаете с нуля?
- Данные не влезают в Excel?
- Pandas достал тормозить и падать по памяти?

Вам хватает чистого SQL?

ДА



DuckDB

НЕТ

Polars

Проблемы подхода

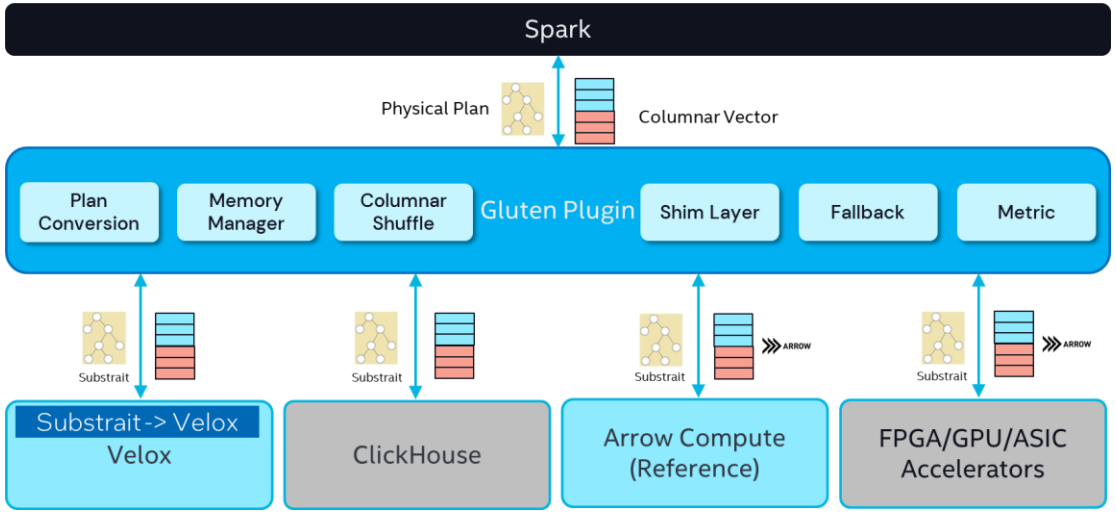
- Сначала всё придётся переписать со Spark
- Нет работы с Hive Metastore (зато есть с Iceberg)
- Перестали влезать на одну ноду – придётся костылить распределённый режим

BIGDATA: ВЫПОЛНИМ ПЛАН SPARK'А НА НАТИВНОМ ДВИЖКЕ!

Gluten on Velox/Clickhouse

- Плагин для Spark
- Проброс вычислений в нативные движки
- Если метод не реализован – возврат к Scala
- Ускорение x2.8 на TPC-H

https://github.com/apache/incubator-gluten?tab=readme-ov-file#7-performance

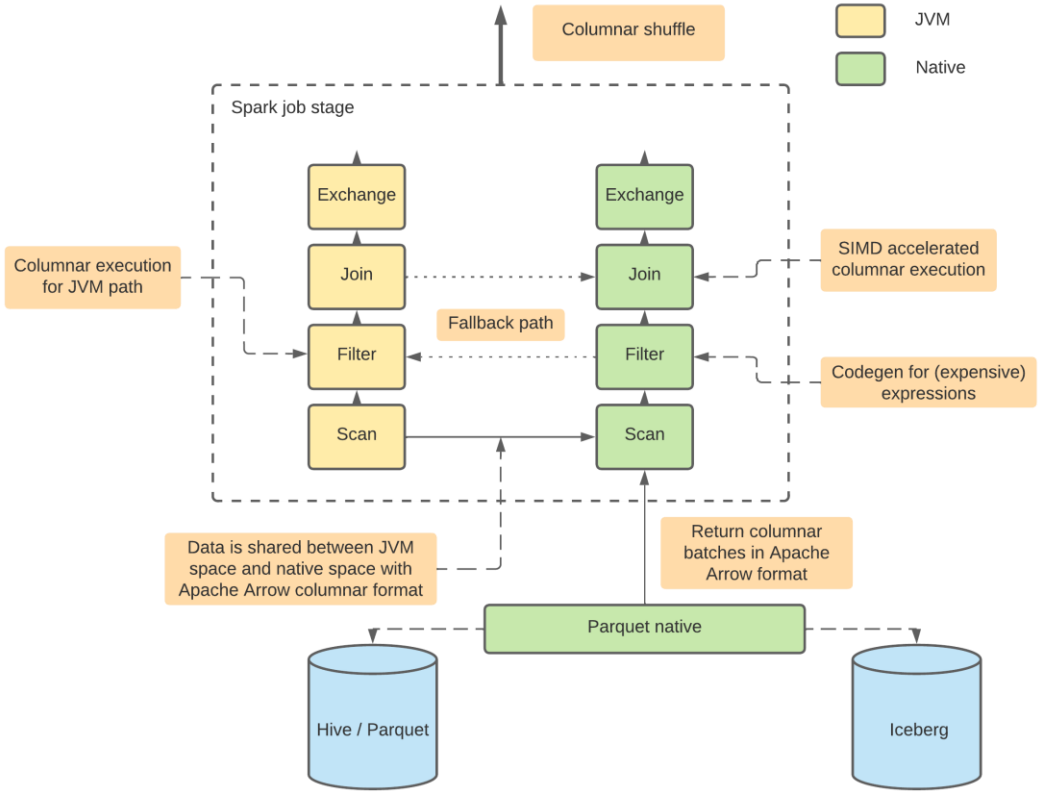


https://github.com/apache/incubator-gluten

Сентябрь 2024: https://smartdataconf.ru/archive/2024/talks/f0f13389f1674c61a3d4e17cea0205ef

Apache Comet

- Плагин для Spark
- Проброс вычислений в Apache DataFusion
- Если метод не реализован – возврат к Scala
- Ускорение x2.4 на TPC-H



https://datafusion.apache.org/comet/overview.html

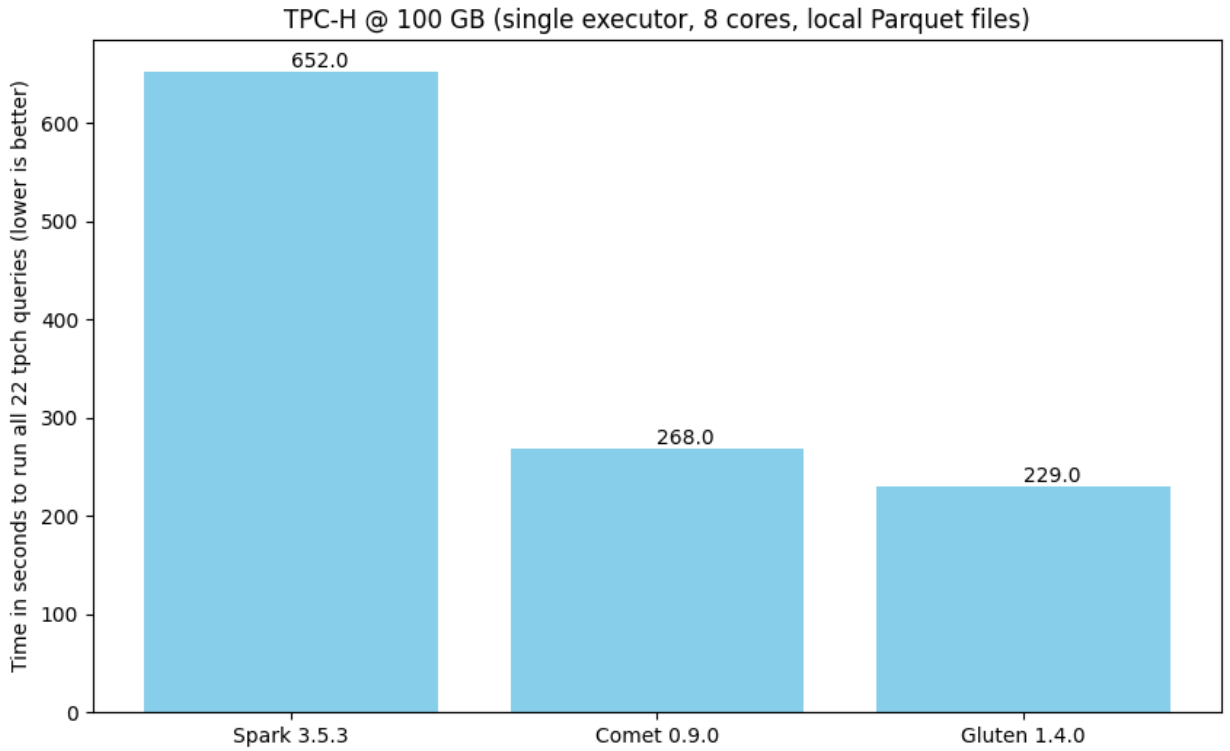
Завтра: https://smartdataconf.ru/talks/8e248260514840208beaf9cf90cc0778

Apache DataFusion

- **Низкоуровневый фреймворк локальной обработки данных**
- **Реализован на Rust, под капотом Apache Arrow-RS**
- **Предоставляет примитивы для создания высокоуровневых движков:**
  - Compute-функции (скалярные, агрегатные, оконные)
  - Организация колоночных вычислений (row-batch, repartition)
  - Оптимизатор логических и физических планов запросов
  - Абстракции для чтения/записи
- **И возможности для расширения на каждом этапе**

Gluten vs Comet

- Производительность сопоставимая
- Вопросы к реализации – тоже *



https://datafusion.apache.org/comet/gluten_comparison.html#performance

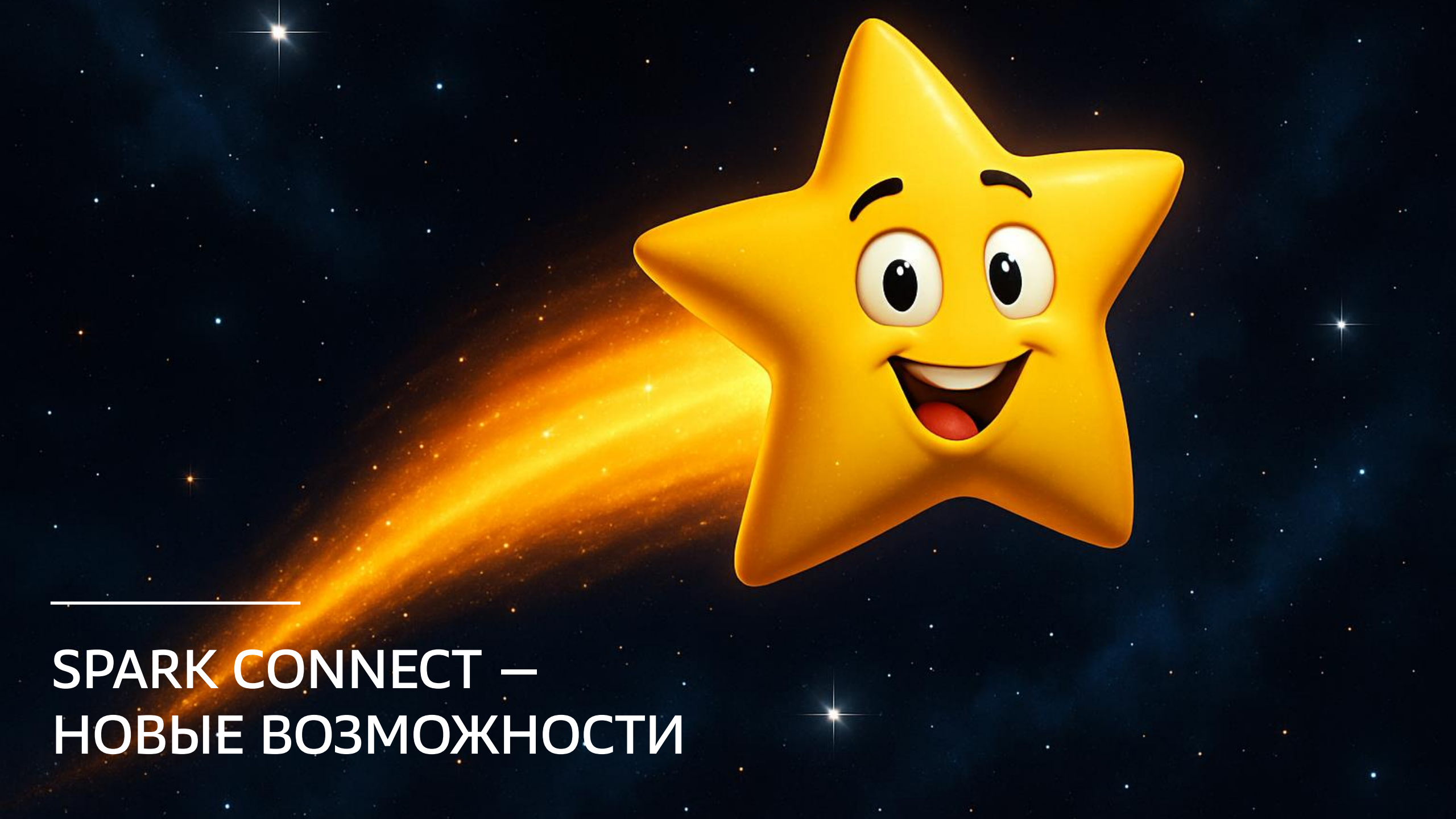
* https://github.com/apache/datafusion-comet/issues/2036

Проблемы подхода

- Ускорение – пока только на графиках TPC-H
- Функции могут отсутствовать или быть реализованы неправильно
- Удвоение проблем версионирования и совместимости
  - Yo dawg, я слышал, вы любите собирать код,
    поэтому если вы собрали код на Java/Scala,
    вы можете собрать ещё код на C++/Rust

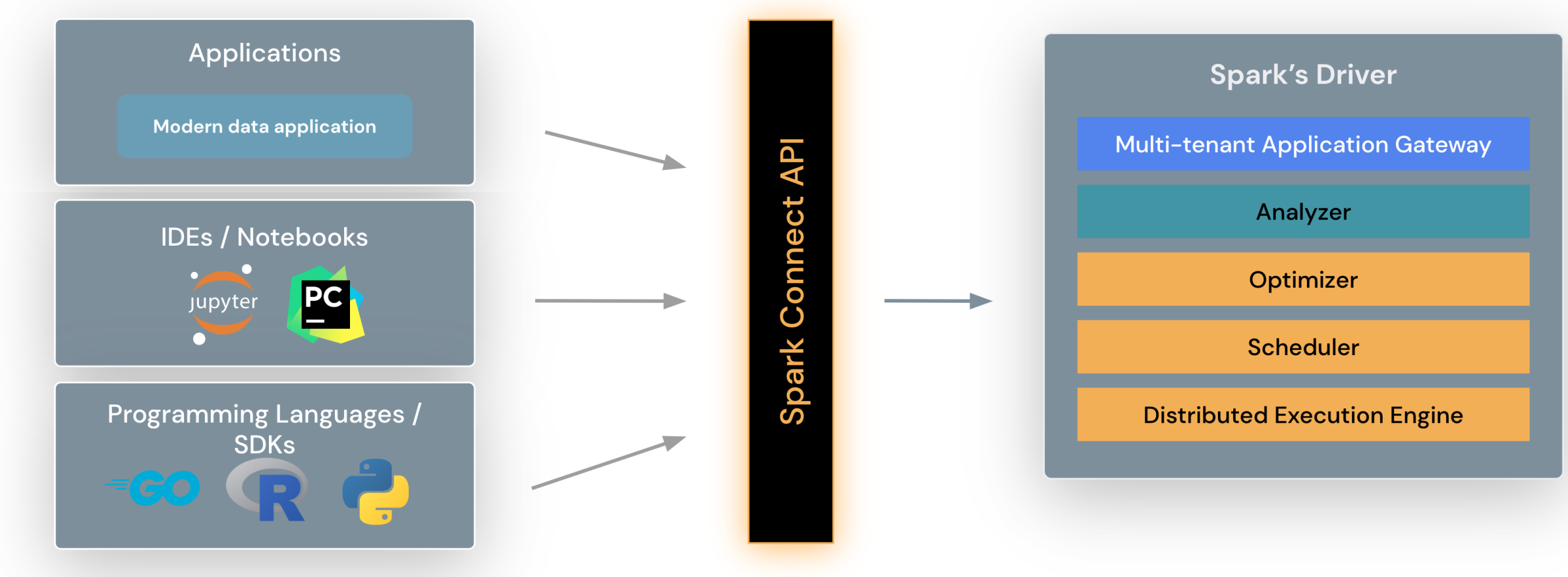
А можно как-то по-другому?

- Можно ли полностью отказаться от JVM?
- И груза решений, которые тянутся с 2009 года
- Реализовать Spark API с нуля?



SPARK CONNECT — НОВЫЕ ВОЗМОЖНОСТИ

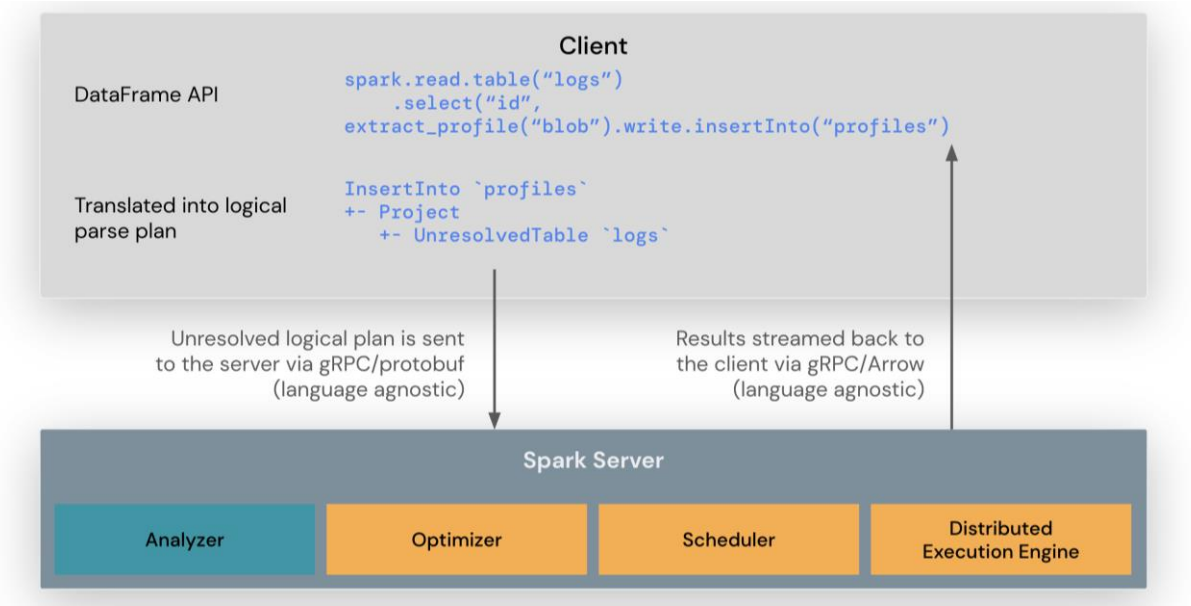
Spark Connect



https://spark.apache.org/docs/latest/spark-connect-overview.html

Spark Connect

- Клиент-серверная архитектура
- От клиента - сырой план запроса
- От сервера – результат запроса
- Табличные данные – в Arrow IPC формате

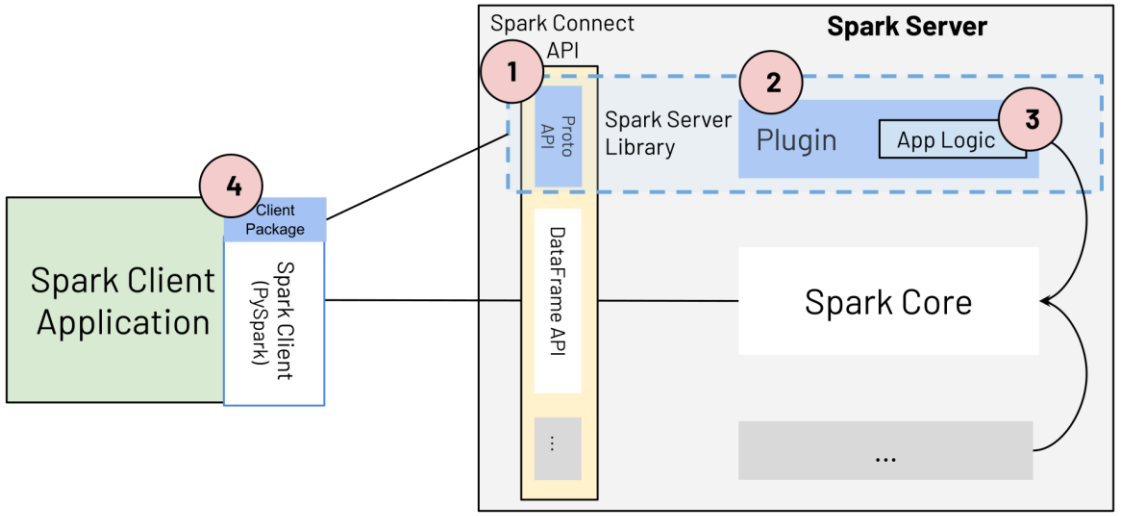


https://spark.apache.org/docs/latest/spark-connect-overview.html

Завтра: https://smartdataconf.ru/talks/70f7ff2fd89344148153f3db5f156f80

PySpark Client

- **pip install pyspark**
  - 450MB
  - Требуется Java
- **pip install pyspark-client**
  - 1.5MB (в 300 раз меньше)
  - Больше никакой Java



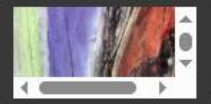
https://spark.apache.org/docs/4.0.0/app-dev-spark-connect.html

Daft

- Фреймворк обработки данных на Rust+Python
- Под капотом – Arrow2 (не Apache)
- Распределённый режим через Ray Cluster
- Оконки, UDF
- Работа с изображениями, ML-методы

- Частично реализует Spark Connect Server

```
df.sort("num_pixels_red", desc=True).collect()
```

path Utf8	image Image[MIXED]	red_mask Python	num_pixels_red Int64
s3://daft-public-data/open-images/validation-images/0040009ad56c2bc2.jpg			193170
s3://daft-public-data/open-images/validation-images/004a9412eb1a83e8.jpg			92483
s3://daft-public-data/open-images/validation-images/004545770f4770c7.jpg			66202
s3://daft-public-data/open-images/validation-images/007f71665b0812a7.jpg			12725
s3://daft-public-data/open-images/validation-images/0022ecd6f681bed6.jpg			11237

МИР, ЕСЛИ БЫ SPARK ПЕРЕПИСАЛИ НА ARROW

Май 2025: https://phdays.com/ru/forum/program/?talk-id=1735

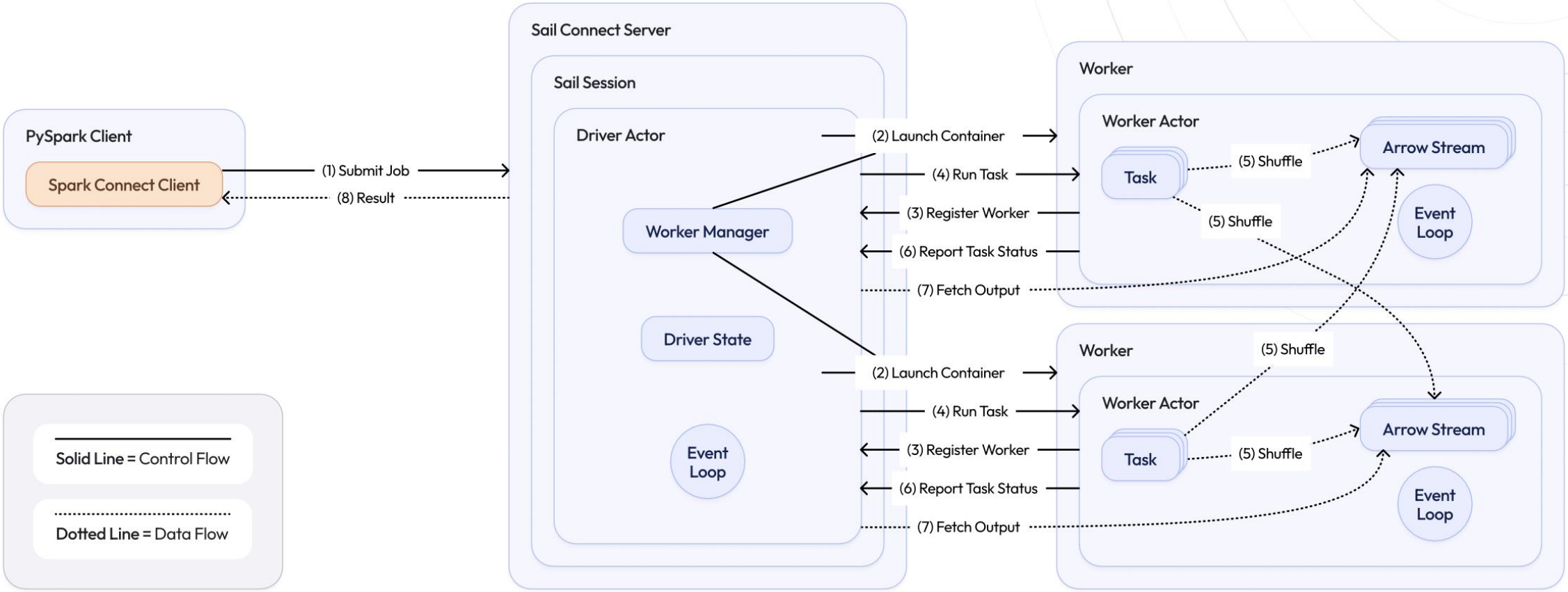

SAIL –
SPARK HA ARROW

Sail

- Фреймворк распределённой обработки данных на Rust+Python
- Распределённый режим – в K8S
- Под капотом – Apache DataFusion и Apache Arrow-rs
- 0% жавки
- Ставит целью полную реализацию Spark Connect Server
- Есть даже Spark Streaming (в каком-то виде)
- Ускорение x4 на TPC-DS *

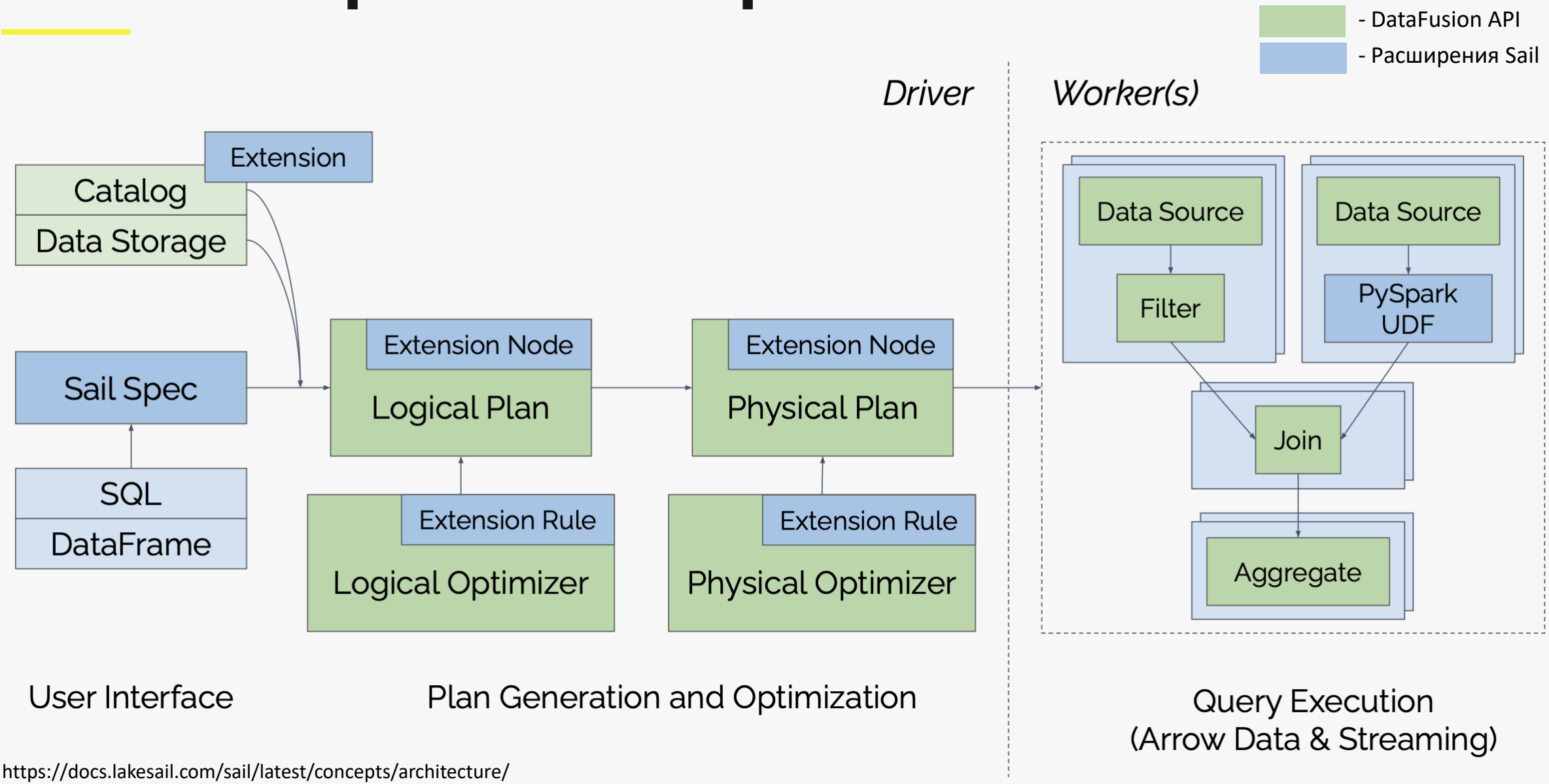
* https://docs.lakesail.com/sail/latest/introduction/benchmark-results/

Sail: архитектура



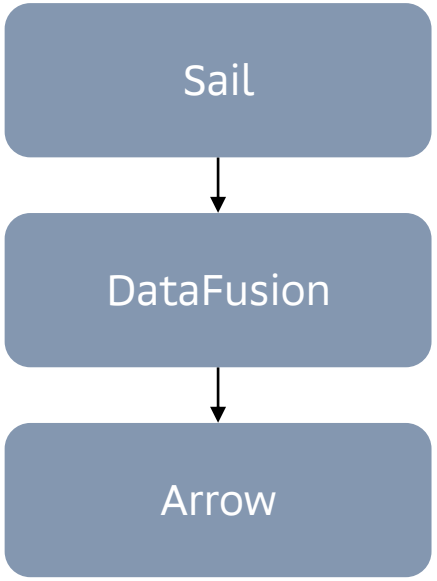
https://docs.lakesail.com/sail/latest/concepts/architecture/

Sail: планирование запроса



https://docs.lakesail.com/sail/latest/concepts/architecture/

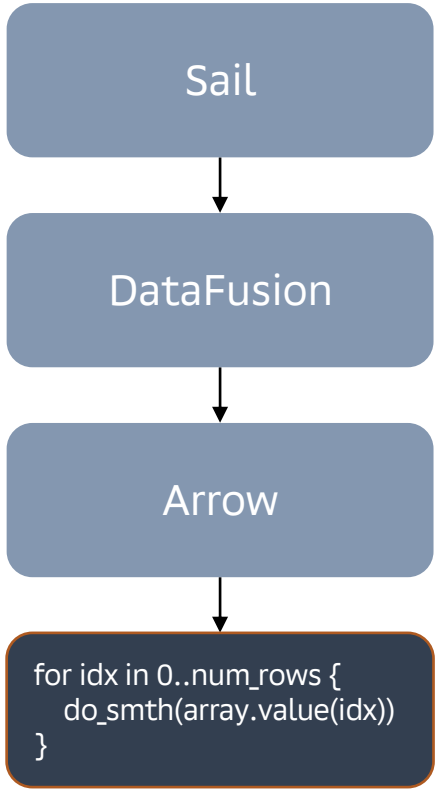
Sail: физическое выполнение



- Переиспользование готовых компонентов
- Vectorized execution на Rust
- Blazingly Fast
- ...

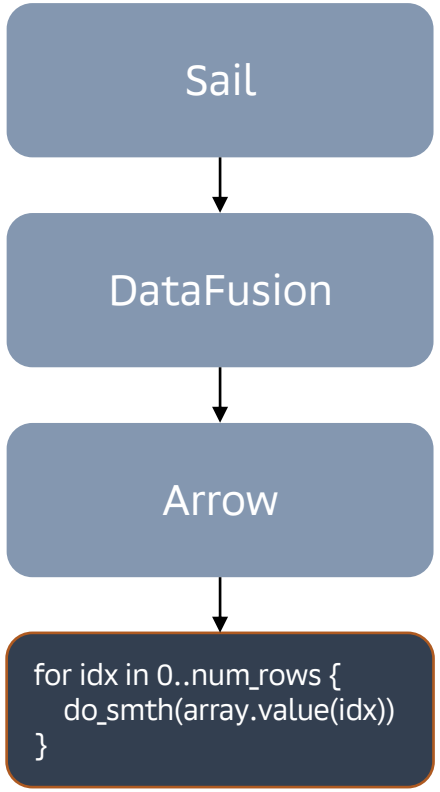
Май 2025: https://phdays.com/ru/forum/program/?talk-id=1735

Sail: физическое выполнение



- На самом деле в большинстве функций примерно так)
- Справедливо для любого execution движка
- Важную роль играет скорость исполнения такого кода

Sail: физическое выполнение



- На самом деле в большинстве функций примерно так)
- Справедливо для любого execution движка
- Важную роль играет скорость исполнения такого кода

```
fn arrays_zip_generic<O: OffsetSizeTrait>(args: &[ArrayRef]) -> Result<ArrayRef> {
    …
    for row_idx in 0..num_rows {
        if validity_mask_opt
            .as_ref()
            .is_some_and(|mask| !mask.is_valid(row_idx))
        {
            offsets.push(last_offset);
            continue;
        }

        let arrays_one_row = lists
            .iter()
            .map(|arg| arg.value(row_idx))
            .collect::<Vec<_>>();
    …
```

https://github.com/lakehq/sail/blob/9dd269b6a95dc5cb17e6e6baa19e3668cc958fc2/crates/sail-plan/src/extension/function/array/arrays_zip.rs

BLAZINGLY FAST
ИЛИ НЕ ОСОБО?

Чем будем мерить? TPC-DS?

- **TPC-DS – индустриальный стандарт бенчмаркинга СУБД**
- **Но есть нюансы:**
  - Каждый считает по-своему, на своём железе
  - Подгонка движков под запросы
  - Запросы отличаются от стандартной аналитической нагрузки
    - *А некоторые вообще странные (например, запрос 6)*
  - Нет единой большой сводной таблицы с результатами

ClickBench

- **Как известно, ClickHouse не тормозит**
- **Чтобы это доказать — реализовали собственный бенчмарк**
- **Главные особенности:**
  - Open Source, любой может добавить реализацию для конкретного фреймворка
  - Уже добавлены десятки фреймворков и много конфигураций железа (инстансы в облаке)
  - Результаты прогонов всех фреймворков на одинаковых инстансах
  - Есть все упомянутые фреймворки *
  - Запросы похожи на аналитические
  - Не все ещё подогнали свои движки

https://benchmark.clickhouse.com/

* https://github.com/ClickHouse/ClickBench/pull/402 (403, 557, 575)

ClickBench: что будем сравнивать?

- Чтение из Parquet (без InMemory, индексов и т.д.)
- Результаты «холодного» прогона (никакого кеширования)
- Июльский прогон на одной ноде c6a.x4large (16 vcore, 32GB ram)

- Метрика – среднее геометрическое:

$$\sqrt[N]{\frac{B_1}{A_1} * \frac{B_2}{A_2} * \frac{B_3}{A_3} * \ \dots \ * \frac{B_N}{A_N}}$$

Пример:

Запрос	Время A	Время B	(B/A)
1	1.5	2.1	2.1 / 1.5 = 1.4
2	1.0	1.0	1.0 / 1.0 = 1.0
3	3.0	1.5	1.5 / 3.0 = 0.5

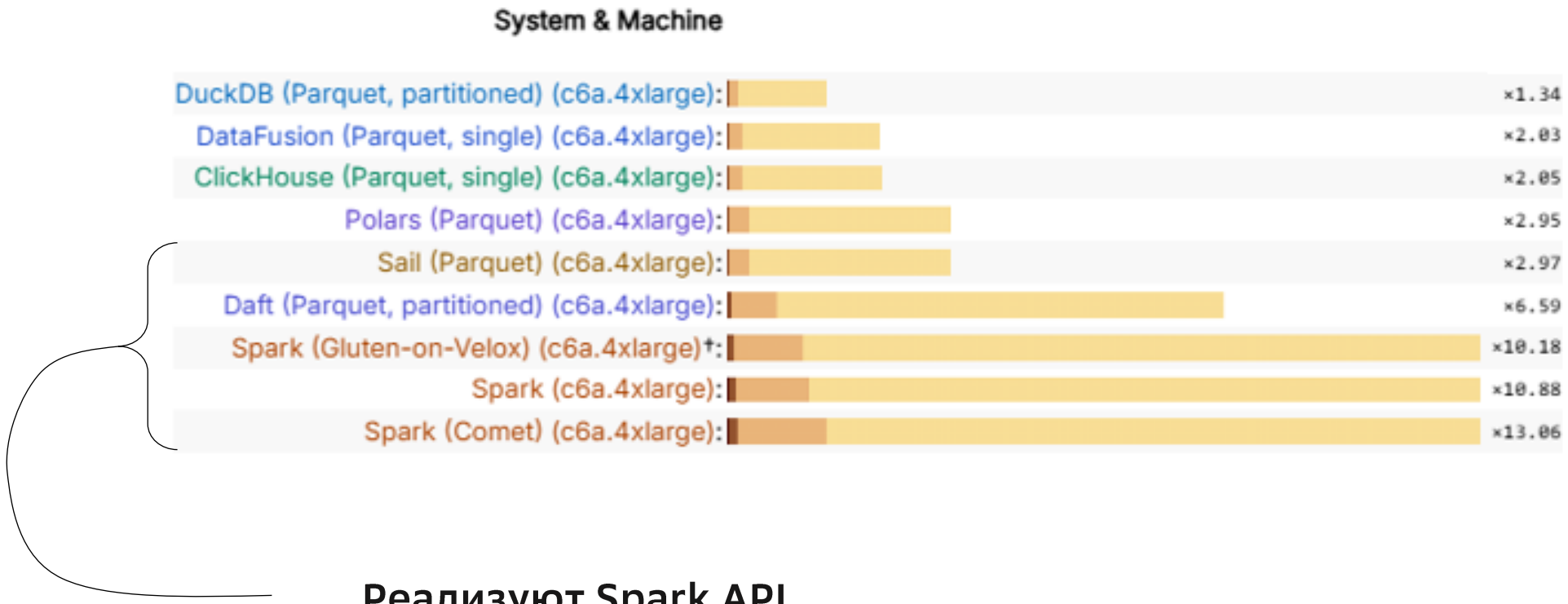
$1.4 * 1.0 * 0.5 = 0.7$

$\sqrt[3]{0.7} \approx 0.887$

$0.887 < 1$ => A медленнее B

$\frac{1}{0.887} \approx 1.127$ => B быстрее A на 12.7%

ClickBench: результаты (общие)

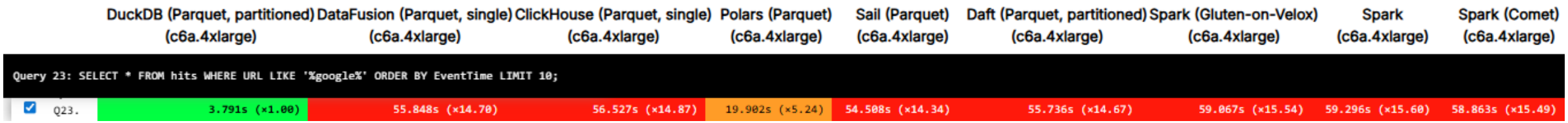


Реализуют Spark API

https://benchmark.clickhouse.com/

ПОЧЕМУ %SQLENGINENAME% ТОРМОЗИТ? И КАК ЭТО ИСПРАВИТЬ?

Почему так медленно? (или быстро?)



	DuckDB (Parquet, partitioned) (c6a.4xlarge)	DataFusion (Parquet, single) (c6a.4xlarge)	ClickHouse (Parquet, single) (c6a.4xlarge)	Polars (Parquet) (c6a.4xlarge)	Sail (Parquet) (c6a.4xlarge)	Daft (Parquet, partitioned) (c6a.4xlarge)	Spark (Gluten-on-Velox) (c6a.4xlarge)	Spark (c6a.4xlarge)	Spark (Comet) (c6a.4xlarge)
Query 23: SELECT * FROM hits WHERE URL LIKE '%google%' ORDER BY EventTime LIMIT 10;									
Q23.	3.791s (×1.00)	55.848s (×14.70)	56.527s (×14.87)	19.902s (×5.24)	54.508s (×14.34)	55.736s (×14.67)	59.067s (×15.54)	59.296s (×15.60)	58.863s (×15.49)

- **Викторина)**
- **Простой запрос:**

```sql
SELECT * FROM hits WHERE URL LIKE '%google%'
ORDER BY EventTime LIMIT 10;
```

- **Скорость DuckDB на голову выше всех остальных**
- **Разница в скорости до 15 раз, как так?**
  1. В DuckDB используют читы
  2. В DuckDB баг / некорректный расчёт
  3. В DuckDB наиболее эффективный алгоритм

Почему так медленно?

```sql
SELECT *
FROM hits
WHERE URL LIKE '%google%'
ORDER BY EventTime
LIMIT 10;
```

- **Без каких-либо оптимизаций:**
  - Происходит FullScan всех колонок (очень медленно)
  - Далее все данные сортируются (ещё медленнее)
  - И в конце выбирается 10 первых строк

SCAN (select *)
FILTER (like)
Все ряды, все колонки

↓

SORT (ключ – EventTime)
SHUFFLE (в distributed mode)

↓

LIMIT (10)

↓

- **Разбираем на примере DataFusion**
  - Как минимум до версии 49 включительно
  - Дальше разобрались и соптимизировали

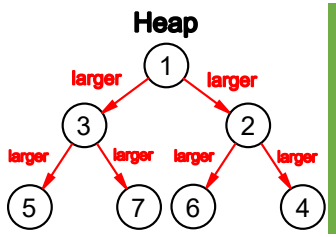
https://datafusion.apache.org/blog/2025/09/10/dynamic-filters/

Как соптимизировать? (1)

```
SELECT *
FROM hits
WHERE URL LIKE '%google%'
ORDER BY EventTime
LIMIT 10;
```

- **TopK оптимизация:**
  - Убираем сортировку и выбор первых 10 рядов
  - Держим в памяти 10 рядов с минимальными значениями
  - В двоичной куче по ключу EventTime

SCAN (select *)
FILTER (like)
Все ряды, все колонки



https://github.com/cirosantilli/media/blob/master/heap.svg

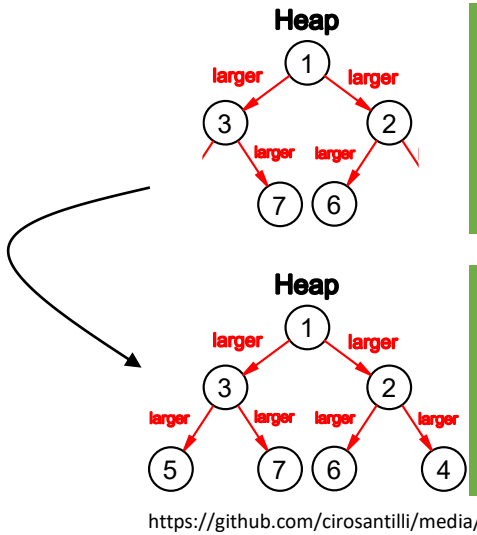
TopK (10)

https://datafusion.apache.org/blog/2025/09/10/dynamic-filters/

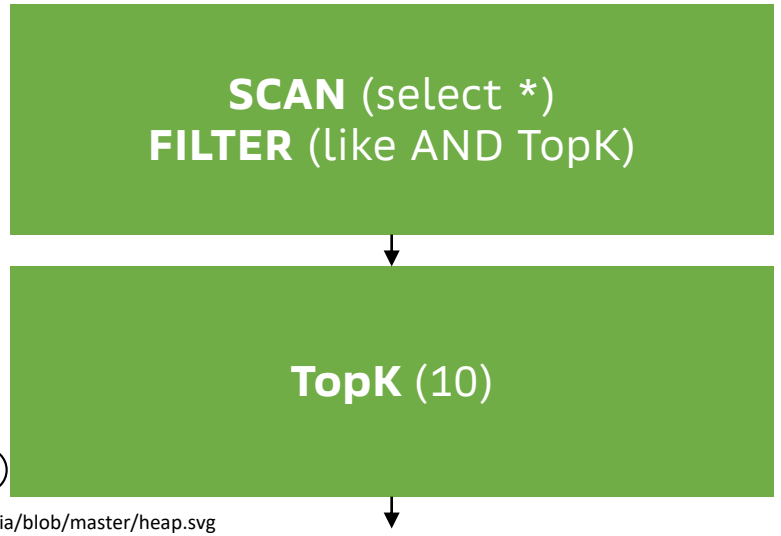
Как соптимизировать? (2)

```
SELECT *
FROM hits
WHERE URL LIKE '%google%'
ORDER BY EventTime
LIMIT 10;
```

- **Поздняя материализация:**
  - Совмещаем чтение и построение кучи
  - Всегда держим только верхние 10 рядов
  - Читаем только колонки URL и EventTime
  - Проверяем 2 условия:
    - *URL LIKE '%google%'*
    - *EventTime < Max(Heap)*
  - Если подходит - обновляем кучу
  - Вычитываем все данные в память и оставляем ряд
  - В распределённом режиме – в конце делаем глобальный TopK



https://github.com/cirosantilli/media/blob/master/heap.svg



https://datafusion.apache.org/blog/2025/09/10/dynamic-filters/

А как в DuckDB?

```sql
SELECT *
FROM hits
WHERE URL LIKE '%google%'
ORDER BY EventTime
LIMIT 10;
```

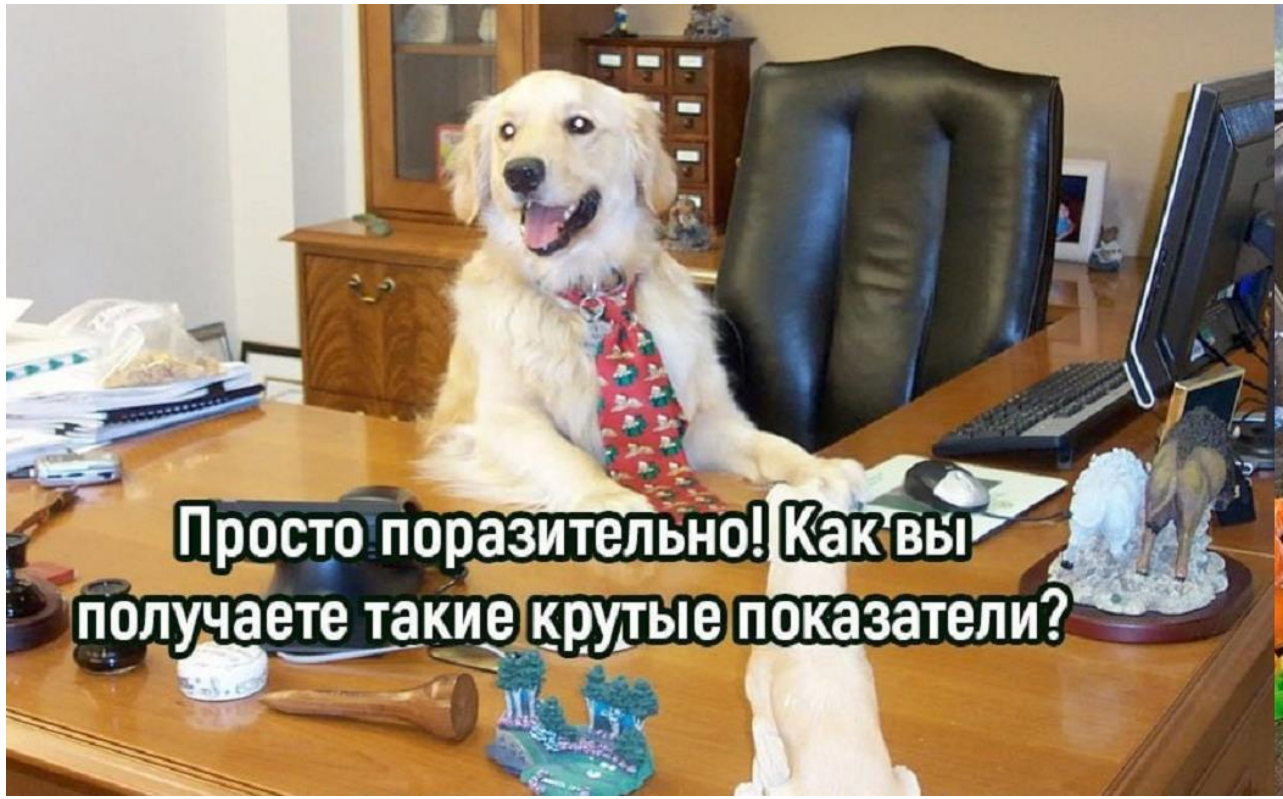
https://www.meme-arsenal.com/create/chose?tag=мозг

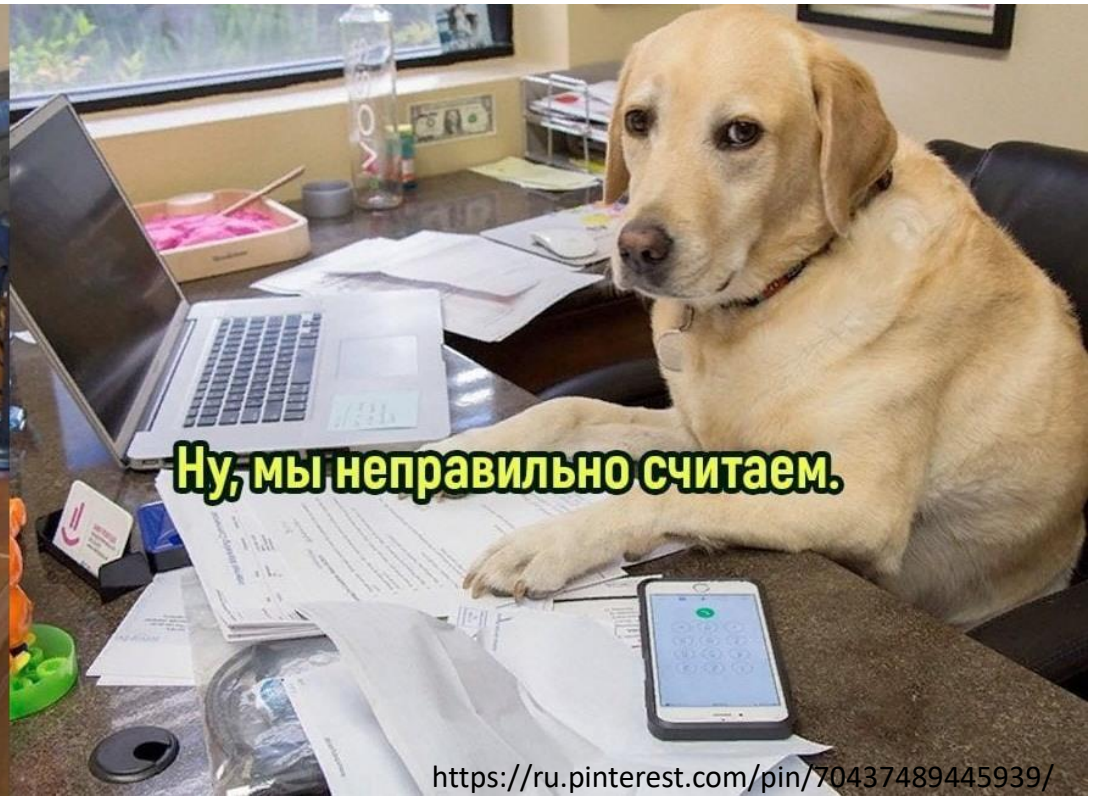
```sql
WITH (
SELECT
    filename() as file,
    row_number_in_file() as rn
FROM hits
WHERE URL LIKE '%google%'
ORDER BY EventTime
LIMIT 10
) AS filter

SELECT *
FROM hits
JOIN filter
ON hits.file = filter.file
AND hits.rn = hits.rn;
```

- **Запрос внутри движка разбивается на 2:**
  1. Чтение 2 колонок, фильтрация, вычисление топ-10
     Получение названия файла и номера строки в файле
  2. Чтение всех колонок, но только
     из нужных файлов по нужным строкам
  3. Никаких читов, просто мощная оптимизация
     Для колоночных форматов данных

КОРРЕКТНОСТЬ РЕАЛИЗАЦИИ SPARK API

Spark API работает. А корректно ли?





https://ru.pinterest.com/pin/70437489445939/

Spark API работает. А корректно ли?

- Как проверить, что реализация spark-функции работает правильно?

- Написать тесты?
- А может есть уже готовые?

Spark API работает. А корректно ли?

- Как проверить, что реализация spark-функции работает правильно?

- Написать тесты?
- А может есть уже готовые?

- Есть!
- DocTest-ы самого Spark)
- 2 в 1: пример работы и тест

API Reference › ··· › Column › pyspark.sql.Column.startswith

pyspark.sql.Column.startswith

`Column.startswith(other)` [source]

String starts with. Returns a boolean `Column` based on a string match.

Changed in version 3.4.0: Supports Spark Connect.

Parameters:

other : *Column or str*

string at start of line (do not use a regex ^)

Examples

```
>>> df = spark.createDataFrame(
...      [(2, "Alice"), (5, "Bob")], ["age", "name"])
>>> df.filter(df.name.startswith('Al')).collect()
[Row(age=2, name='Alice')]
>>> df.filter(df.name.startswith('^Al')).collect()
[]
```

https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.Column.startswith.html#pyspark.sql.Column.startswith

Spark API работает. А корректно ли?

- **В Sail реализована система прогона Spark Tests в CI-пайплайнах**
- **При открытии merge request в Sail:**
  - Прогоняются доктесты Spark
  - Выводится отчёт со всеми ошибками
  - И Diff
- **Сразу видно, что починилось**
  - Или наоборот сломалось)

https://github.com/lakehq/sail/pull/914#issuecomment-3325755810

Spark API работает. А корректно ли?

- Допустим, все доктесты пройдены
- Значит ли это, что всё ОК?

```
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column PASSED [  2%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.__getattr__ PASSED [  5%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.__getitem__ PASSED [  8%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.alias PASSED [ 11%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.asc PASSED [ 14%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.asc_nulls_first PASSED [ 17%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.asc_nulls_last PASSED [ 20%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.between PASSED [ 22%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.bitwiseAND PASSED [ 25%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.bitwiseOR PASSED [ 28%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.bitwiseXOR PASSED [ 31%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.cast PASSED [ 34%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.contains PASSED [ 37%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.desc PASSED [ 40%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.desc_nulls_first PASSED [ 42%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.desc_nulls_last PASSED [ 45%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.dropFields PASSED [ 48%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.endswith PASSED [ 51%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.eqNullSafe PASSED [ 54%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.getField PASSED [ 57%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.getItem PASSED [ 60%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.ilike PASSED [ 62%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.isNaN PASSED [ 65%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.isNotNull PASSED [ 68%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.isNull PASSED [ 71%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.isin PASSED [ 74%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.like PASSED [ 77%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.otherwise PASSED [ 80%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.over PASSED [ 82%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.rlike PASSED [ 85%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.startswith PASSED [ 88%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.substr PASSED [ 91%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.try_cast PASSED [ 94%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.when PASSED [ 97%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.withField PASSED [100%]

---------- generated report log file: /home/x/sail/tmp/spark-tests/latest/doctest-column.jsonl ----------
====================================== 35 passed, 10 warnings in 4.26s =======================================
```

Spark API работает. А корректно ли?

- Допустим, все доктесты пройдены
- Значит ли это, что всё ОК?

```
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column PASSED [  2%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.__getattr__ PASSED [  5%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.__getitem__ PASSED [  8%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.alias PASSED [ 11%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.asc PASSED [ 14%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.asc_nulls_first PASSED [ 17%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.asc_nulls_last PASSED [ 20%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.between PASSED [ 22%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.bitwiseAND PASSED [ 25%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.bitwiseOR PASSED [ 28%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.bitwiseXOR PASSED [ 31%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.cast PASSED [ 34%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.contains PASSED [ 37%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.desc PASSED [ 40%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.desc_nulls_first PASSED [ 42%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.desc_nulls_last PASSED [ 45%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.dropFields PASSED [ 48%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.endswith PASSED [ 51%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.eqNullSafe PASSED [ 54%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.getField PASSED [ 57%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.getItem PASSED [ 60%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.ilike PASSED [ 62%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.isNaN PASSED [ 65%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.isNotNull PASSED [ 68%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.isNull PASSED [ 71%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.isin PASSED [ 74%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.like PASSED [ 77%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.otherwise PASSED [ 80%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.over PASSED [ 82%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.rlike PASSED [ 85%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.startswith PASSED [ 88%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.substr PASSED [ 91%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.try_cast PASSED [ 94%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.when PASSED [ 97%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.withField PASSED [100%]

---------- generated report log file: /home/x/sail/tmp/spark-tests/latest/doctest-column.jsonl ----------
========================================= 35 passed, 10 warnings in 4.26s ==========================================
```

Spark API работает. А корректно ли?

- Допустим, все доктесты пройдены
- Значит ли это, что всё ОК?

```
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column PASSED [  2%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.__getattr__ PASSED [  5%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.__getitem__ PASSED [  8%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.alias PASSED [ 11%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.asc PASSED [ 14%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.asc_nulls_first PASSED [ 17%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.asc_nulls_last PASSED [ 20%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.between PASSED [ 22%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.bitwiseAND PASSED [ 25%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.bitwiseOR PASSED [ 28%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.bitwiseXOR PASSED [ 31%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.cast PASSED [ 34%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.contains PASSED [ 37%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.desc PASSED [ 40%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.desc_nulls_first PASSED [ 42%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.desc_nulls_last PASSED [ 45%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.dropFields PASSED [ 48%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.endswith PASSED [ 51%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.eqNullSafe PASSED [ 54%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.getField PASSED [ 57%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.getItem PASSED [ 60%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.ilike PASSED [ 62%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.isNaN PASSED [ 65%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.isNotNull PASSED [ 68%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.isNull PASSED [ 71%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.isin PASSED [ 74%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.like PASSED [ 77%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.otherwise PASSED [ 80%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.over PASSED [ 82%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.rlike PASSED [ 85%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.startswith PASSED [ 88%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.substr PASSED [ 91%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.try_cast PASSED [ 94%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.when PASSED [ 97%]
.venvs/test-spark.spark-4.0.0/lib/python3.11/site-packages/pyspark/sql/column.py::pyspark.sql.column.Column.withField PASSED [100%]

---------- generated report log file: /home/x/sail/tmp/spark-tests/latest/doctest-column.jsonl ----------
========================================= 35 passed, 10 warnings in 4.26s ==========================================
```

- Доктесты – это очень простые примеры
- А функции Spark очень нетривиальны

Spark API работает. А корректно ли?

- **Доктесты функции map_from_arrays - по одной записи**
- **Нет тестов для случаев:**
  - Несколько рядов (везде по 1)
  - Массивы с дублями ключей (ошибка или дедубликация)
  - Ряды с NULL на месте массива keys и/или values
  - Массивы с разной длиной (должна выдаваться ошибка)
- **В итоге реализация прошла доктесты**
  - Но при этом некорректно работала в последних двух случаях *
  - После выяснения пришлось переделывать **

Example 1: Basic usage of map_from_arrays

```
>>> from pyspark.sql import functions as sf
>>> df = spark.createDataFrame([([2, 5], ['a', 'b'])], ['k', 'v'])
>>> df.select(sf.map_from_arrays(df.k, df.v)).show()
+----------------------+
|map_from_arrays(k, v)|
+----------------------+
|     {2 -> a, 5 -> b}|
+----------------------+
```

Example 2: map_from_arrays with null values

```
>>> from pyspark.sql import functions as sf
>>> df = spark.createDataFrame([([1, 2], ['a', None])], ['k', 'v'])
>>> df.select(sf.map_from_arrays(df.k, df.v)).show()
+----------------------+
|map_from_arrays(k, v)|
+----------------------+
|  {1 -> a, 2 -> NULL}|
+----------------------+
```

Example 3: map_from_arrays with empty arrays

```
>>> from pyspark.sql import functions as sf
>>> from pyspark.sql.types import ArrayType, StringType, IntegerType, StructType, Struc
>>> schema = StructType([
...   StructField('k', ArrayType(IntegerType())),
...   StructField('v', ArrayType(StringType()))
... ])
>>> df = spark.createDataFrame([([], [])], schema=schema)
>>> df.select(sf.map_from_arrays(df.k, df.v)).show()
+----------------------+
|map_from_arrays(k, v)|
+----------------------+
|                    {}|
+----------------------+
```

* https://github.com/lakehq/sail/issues/876
** https://github.com/lakehq/sail/pull/878

Spark API работает. А корректно ли?

- В DataFusion начали переносить спарк функции *
- Казалось бы, Apache Software Foundation
- Но в случае array() и array(NULL) ...
- Возвращается некорректный тип **
  - Int32 вместо NULL
  - NULL конвертируется в любой тип
  - Int32 – не конвертируется например в Struct
  - Это ломает некоторые доктесты

```rust
pub fn make_array_inner(arrays: &[ArrayRef]) -> Result<ArrayRef> {
    let mut data_type = DataType::Null;
    for arg in arrays {
        let arg_data_type = arg.data_type();
        if !arg_data_type.equals_datatype(&DataType::Null) {
            data_type = arg_data_type.clone();
            break;
        }
    }

    match data_type {
        // Either an empty array or all nulls:
        DataType::Null => {
            let length = arrays.iter().map(|a| a.len()).sum();
            // By default Int32
            let array = new_null_array(&DataType::Int32, length);
            Ok(Arc::new(
                SingleRowListArrayBuilder::new(array)
                    .with_nullable(true)
                    .with_field_name(Some(ARRAY_FIELD_DEFAULT_NAME.to_string()))
                    .build_list_array(),
            ))
        }
        DataType::LargeList(..) => array_array::<i64>(arrays, data_type),
        _ => array_array::<i32>(arrays, data_type),
    }
}
```

* https://github.com/apache/datafusion/blob/08e75a9a10c1c859c50b5b54a3664b8c7d1dfde6/datafusion/spark/src/function/array/spark_array.rs#L146
** https://github.com/lakehq/sail/issues/840

Sail: чего не хватает до прода? (1)

- **Чтение и запись в Iceberg**
  - Сейчас можно скрутить через PyIceberg + PyArrow
  - И работа с Delta уже реализована
- **Интеграция с метасторами**
  - В том числе Hive Metastore
- **Запись в классические форматы с поддержкой всех параметров**
  - Механизм insert overwrite partition (но он и в Spark плох)
- **Структурный стриминг**

Sail: чего не хватает до прода? (2)

- **Отказоустойчивость, перезапуск тасков при падении**
- **Полная реализация всего Spark API**
  - Можно отслеживать прогресс по тестам
- **Sub-Interpreters для параллельного исполнения Python UDF**

Sail: кому подойдёт уже сейчас?

- **Пользователям Polars или DuckDB**
  - Если нравится Spark API и не нравится его отсутствие
- **Тем, кто делает RnD, пишет много Ad-Hoc запросов**
  - Никто не мешает сделать pip install pysail pyspark-client
  - И посмотреть, работают конкретно ваши запросы или нет
- **Тем, кто хочет качнуть свой гитхаб**
  - В issues ещё есть довольно простые задачи, которые принесут пользу
  - Разработчики очень быстро отвечают и помогают вкатывать MR
  - https://github.com/lakehq/sail

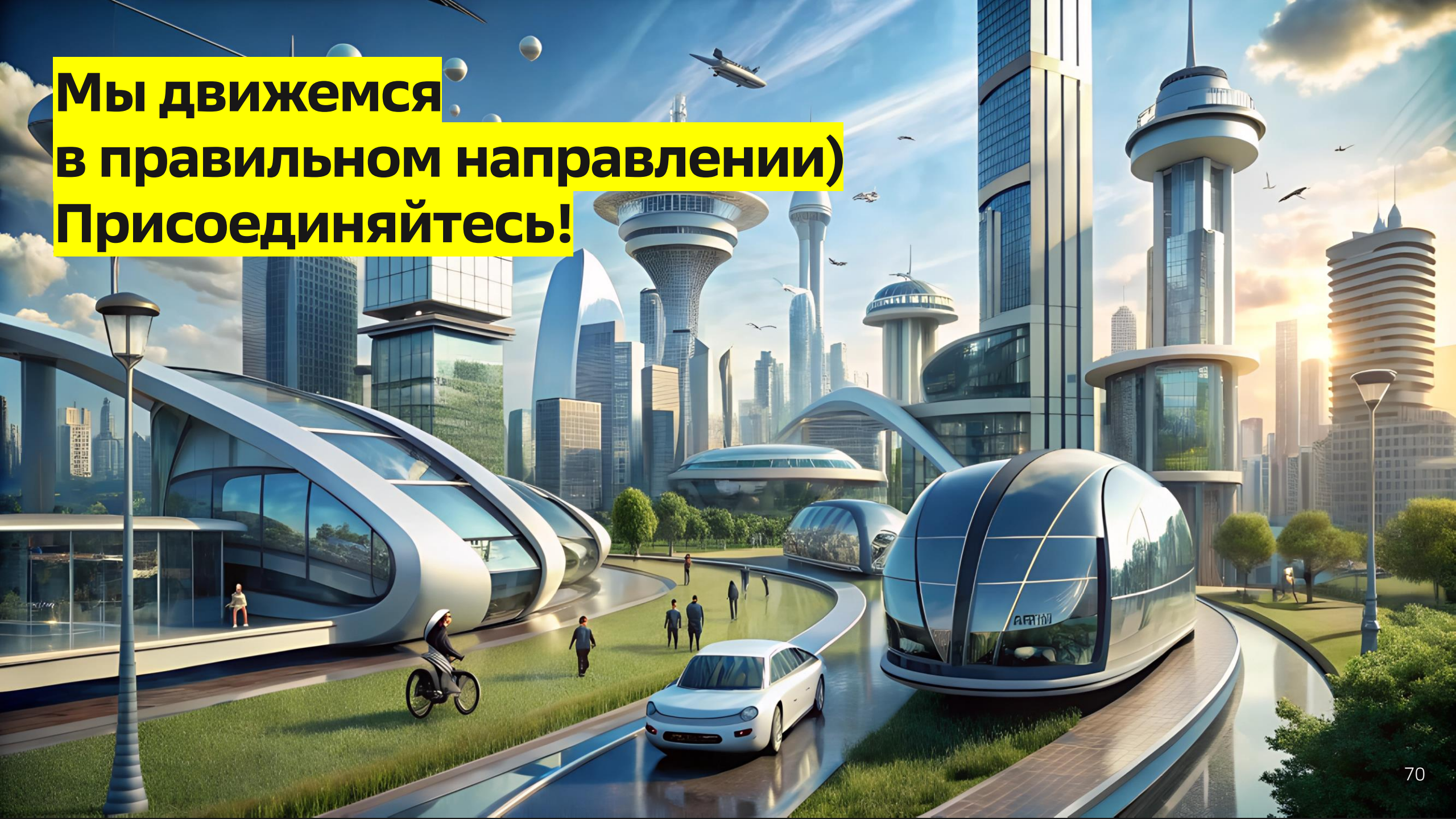
ПОДВЕДЁМ ИТОГИ

Подведём итоги: Spark, бенчмарки, DE

- **Экосистема Spark активно развивается в различных направлениях**
  - Новые функции, новые подходы, попытки ускорить и улучшить
- **Бенчмарки – не просто способ помериться**
  - Но и способ для фреймворков стать лучшей версией себя
  - В результате сравнения появляются идеи, дорабатывается код
- **Российское сообщество DE выходит на новый уровень**
  - Раньше были доклады, что не работает, и с этим ничего не поделать
  - Или про какие-то нишевые решения, созданные с нуля и без широкого распространения
  - А теперь массово включились в доработку и интеграцию мейнстрима

Подведём итоги: Sail

- **Появилась реализация Spark API на Rust**
  - Blazingly fast
  - С хорошей архитектурой
  - С грамотной методикой тестирования
- **Путь Sail до Production-окружений не близок, но виден**
  - А пока можно делать RnD
  - И не забывайте проверять всё, чем пользуетесь
  - Даже Apache компоненты – от багов никто не застрахован


Мы движемся
в правильном направлении)
Присоединяйтесь!