

Spark + ClickHouse

не борьба, а симбиоз

Крашенинников Александр · 5 сентября 2019



О чём доклад

- Откуда потребность в интеграции Spark и ClickHouse
- Сравнение двух гигантов
- Особенности взаимодействия
(tips and tricks for performance)

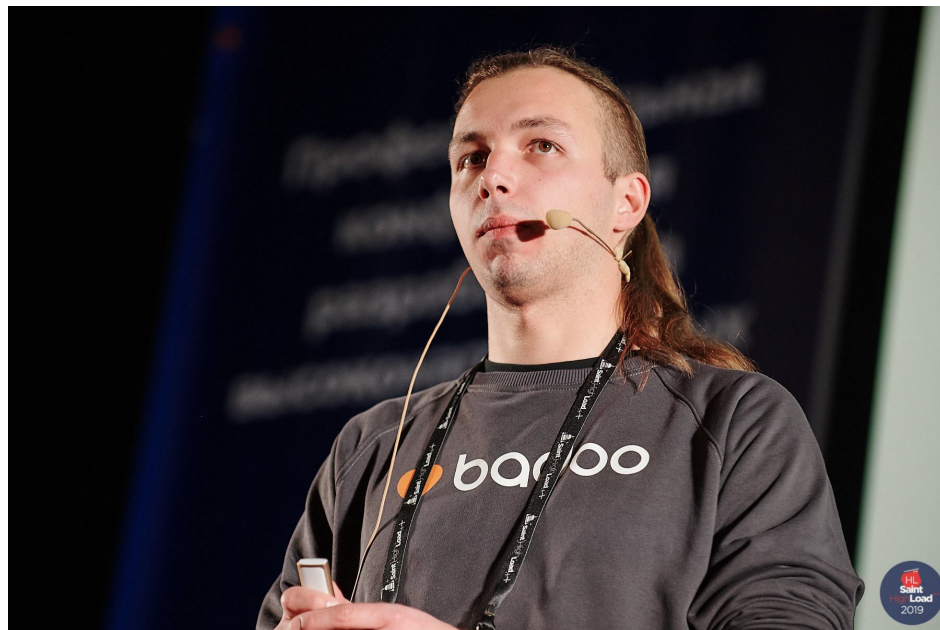
Обо мне

Badoo, Head of Data Engineering

Строю мир на distributed open-source решениях: ClickHouse, Spark, Hadoop

С ClickHouse работаю более 2 лет, успешно «продал» его в несколько проектов внутри компании

Занёс в ClickHouse несколько функций, делаю PR на документацию



MagicLab[★]

 badoo

 bumble

Lumen

 CHAPPY

Откуда возникла потребность в интеграции?

Большому проекту — большие данные!

- В проекте много разнообразных фич
 - Encounters (знакомства со случайными людьми)
 - People Nearby (люди рядом со мной)
 - Общение между пользователями
 - Streaming видеоконтента
 - И многое другое

Большому проекту — большие данные!

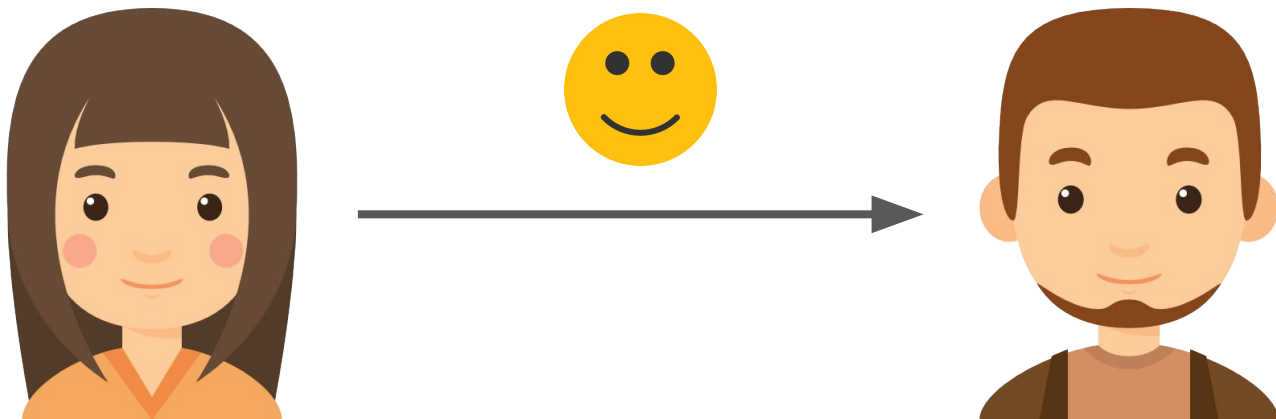
- В проекте много разнообразных фич
 - Encounters (знакомства со случайными людьми)
 - People Nearby (люди рядом со мной)
 - Общение между пользователями
 - Streaming видеоконтента
 - И многое другое
- Фичи обязательно покрыты продуктовыми метриками

Большому проекту — большие данные!

- В проекте много разнообразных фич
 - Encounters (знакомства со случайными людьми)
 - People Nearby (люди рядом со мной)
 - Общение между пользователями
 - Streaming видеоконтента
 - И многое другое
- Фичи обязательно покрыты продуктовыми метриками
- Метрики предоставляют инженеры
 - вычисляют
 - мониторят

Вычисление метрик

- Возникает бизнес-событие (голос)



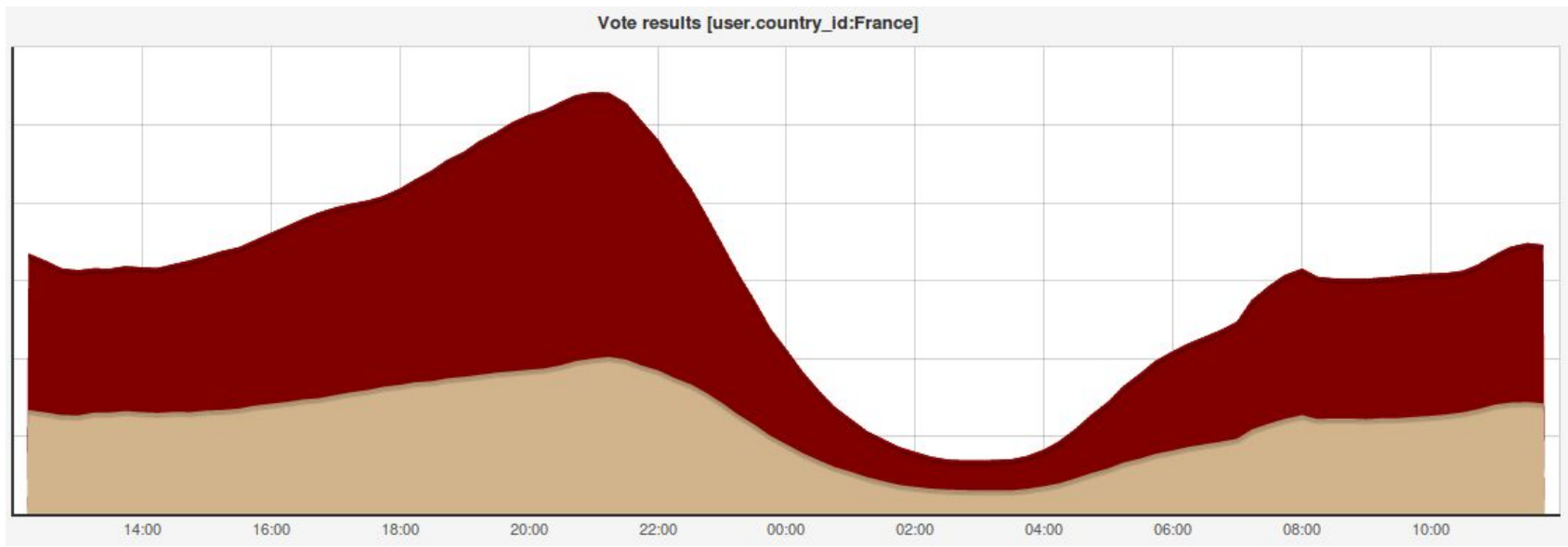
Вычисление метрик

- Возникает бизнес-событие (голос)
- В соответствие ставится статистическое событие

```
message VoteProfileEvent {  
    required int64 active_user_id      = 1;  
    required int64 active_user_country = 2;  
    required int64 passive_user_id     = 3;  
    required int64 passive_user_country = 4;  
    required VoteResult vote_result    = 5;  
}
```

Вычисление метрик

- Возникает бизнес-событие (голос)
- В соответствие ставится статистическое событие
- События доставляются в ClickHouse
- Рисуем графики



Мониторинг метрик

- Метрика без мониторинга – бесполезная метрика ©

Мониторинг метрик

- Метрика без мониторинга – бесполезная метрика ©
- Варианты мониторинговых правил
 - Число регистраций/голосов/платежей не нулевое
 - Число загрузок фотографий во Франции более X/сек

Edge-cases мониторинга

- Метрики иерархичны
 - Обкладываем мониторингом только top-level
 - Не видим просадку более детализированных

Edge-cases мониторинга

- Метрики иерархичны
 - Обкладываем мониторингом только top-level
 - Не видим просадку более детализированных
- Метрики имеют сложное поведение
 - Всплески в одно и то же время, раз в неделю

Edge-cases мониторинга

- Метрики иерархичны
 - Обкладываем мониторингом только top-level
 - Не видим просадку более детализированных
- Метрики имеют сложное поведение
 - Всплески в одно и то же время, раз в неделю
- Традиционный мониторинг не покрывает всех потребностей
- **Применим модный Machine Learning, и сделаем Anomaly Detection!**

Anomaly Detection: concept

- Собираем историю метрики (обучающая выборка)

Anomaly Detection: concept

- Собираем историю метрики (обучающая выборка)
- Предсказываем ожидаемое значение метрики
и доверительный интервал

Anomaly Detection: concept

- Собираем историю метрики (обучающая выборка)
- Предсказываем ожидаемое значение метрики и доверительный интервал
- Соединяем с актуальным значением метрики и сохраняем в хранилище
- При помощи SQL-запроса находим самые значительные отклонения

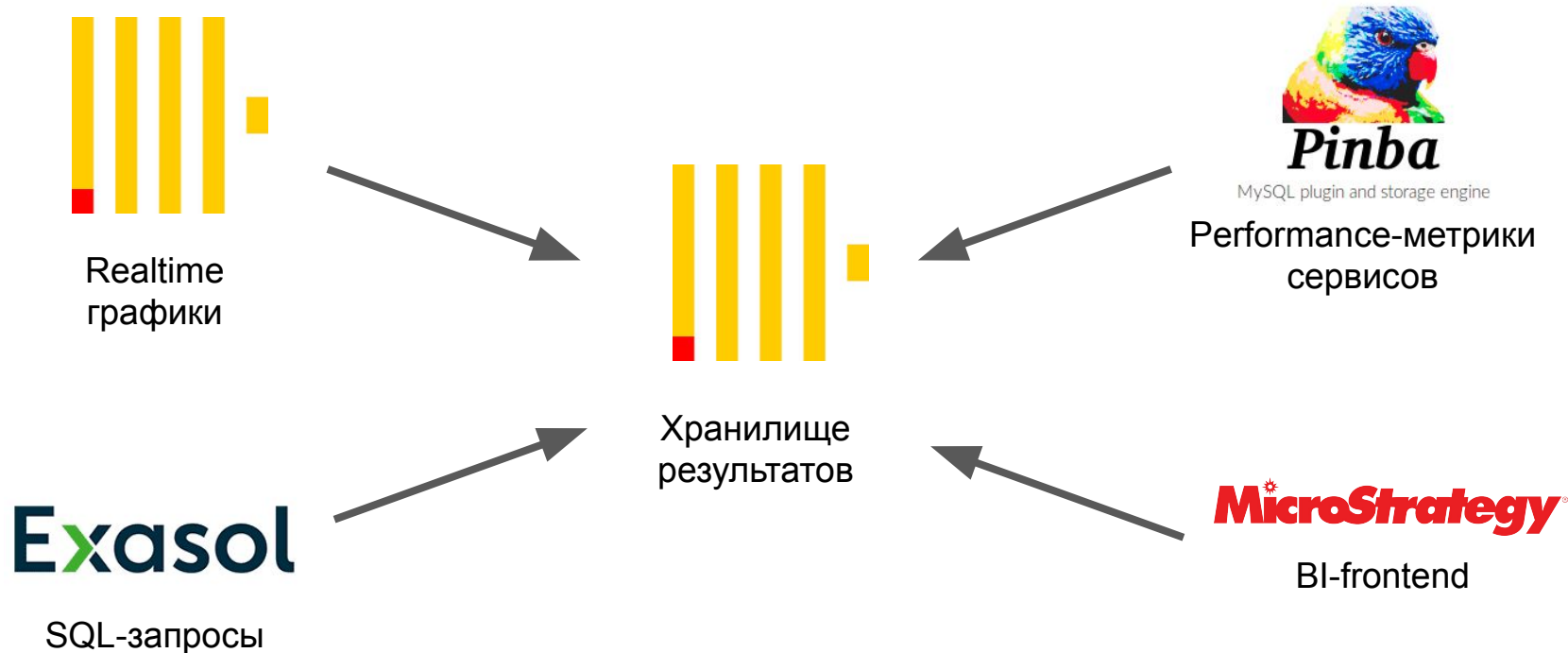
Anomaly Detection: вопросы

- Хранение результатов, выполнение запросов
- Получение данных из разных источников
- Сборка pipeline
 - обучающая история
 - свежие данные из источников
 - предсказание
 - вставка в хранилище

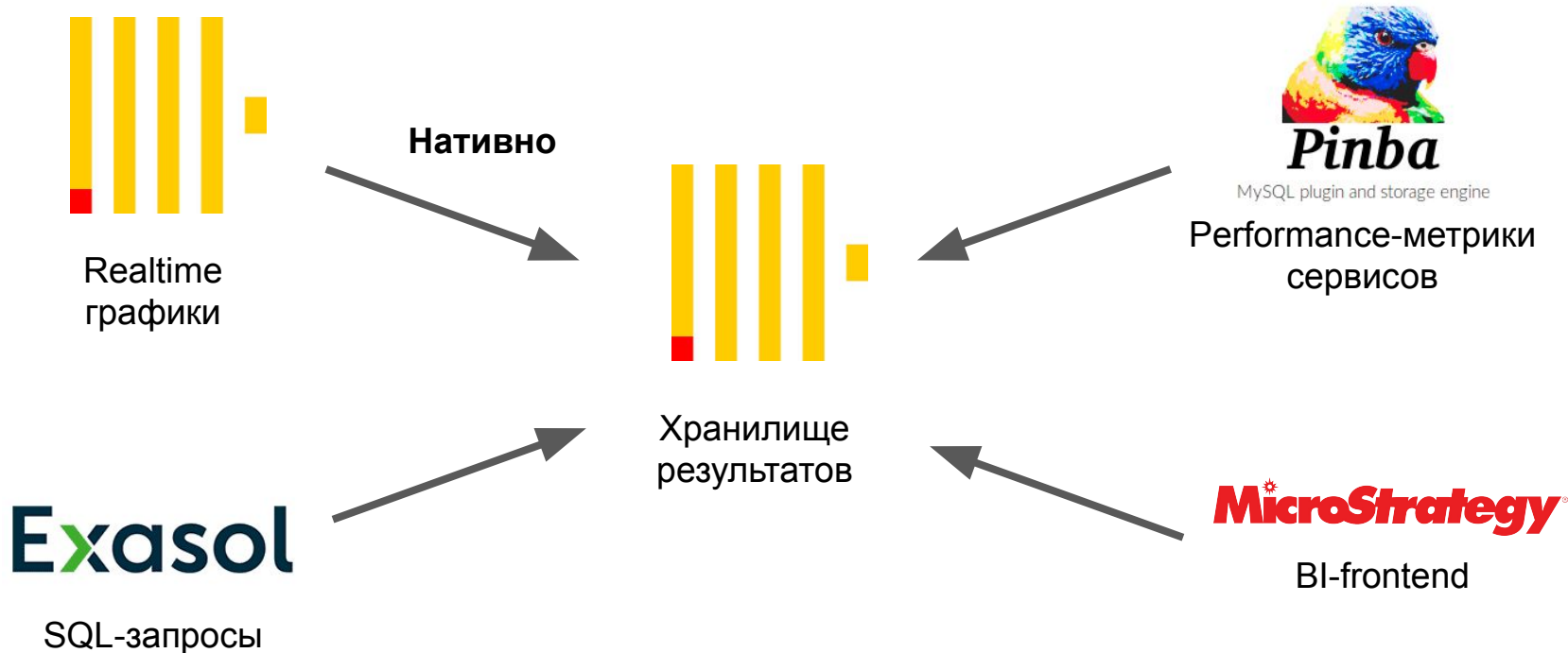
Хранение результатов

- Много данных
 - 15 миллионов метрик
 - 5 миллиардов строк
- Требуется SQL-доступ
- Ну, вы поняли...
 - ClickHouse!

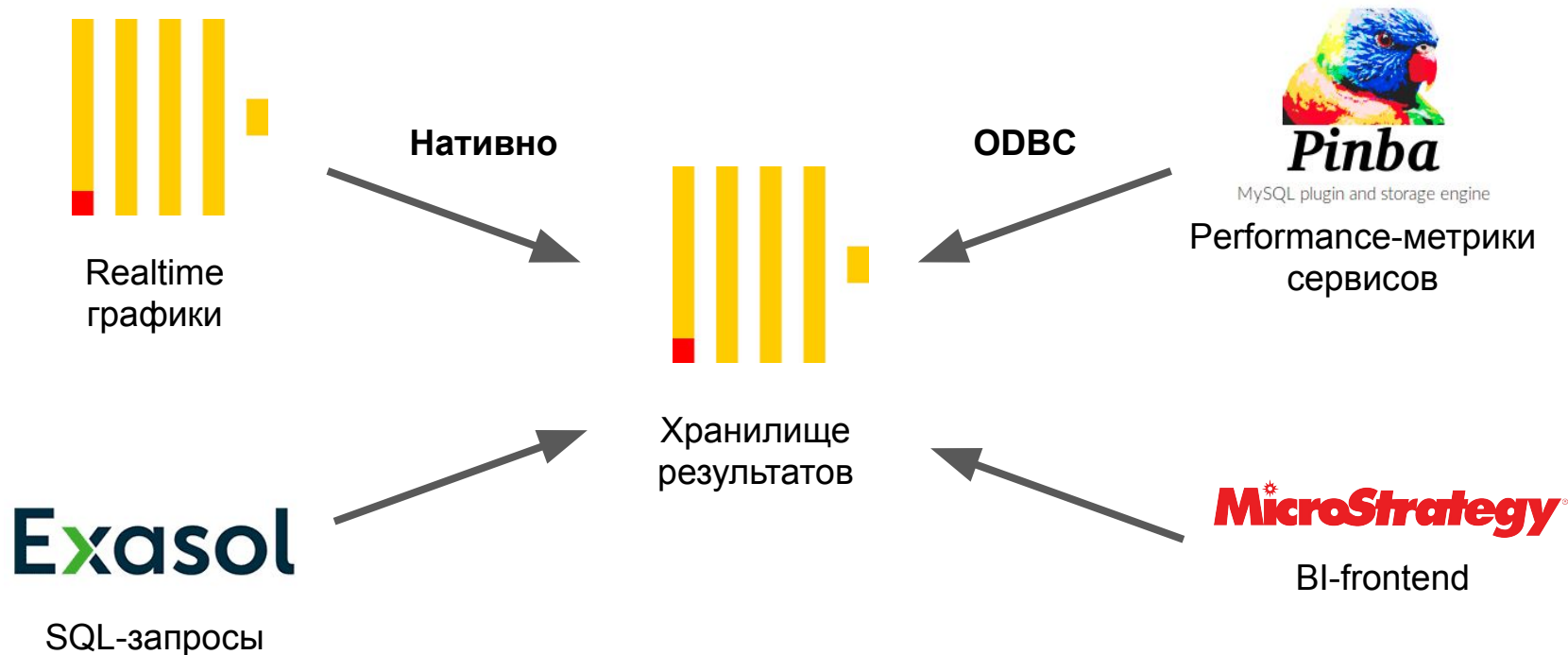
Разные источники метрик



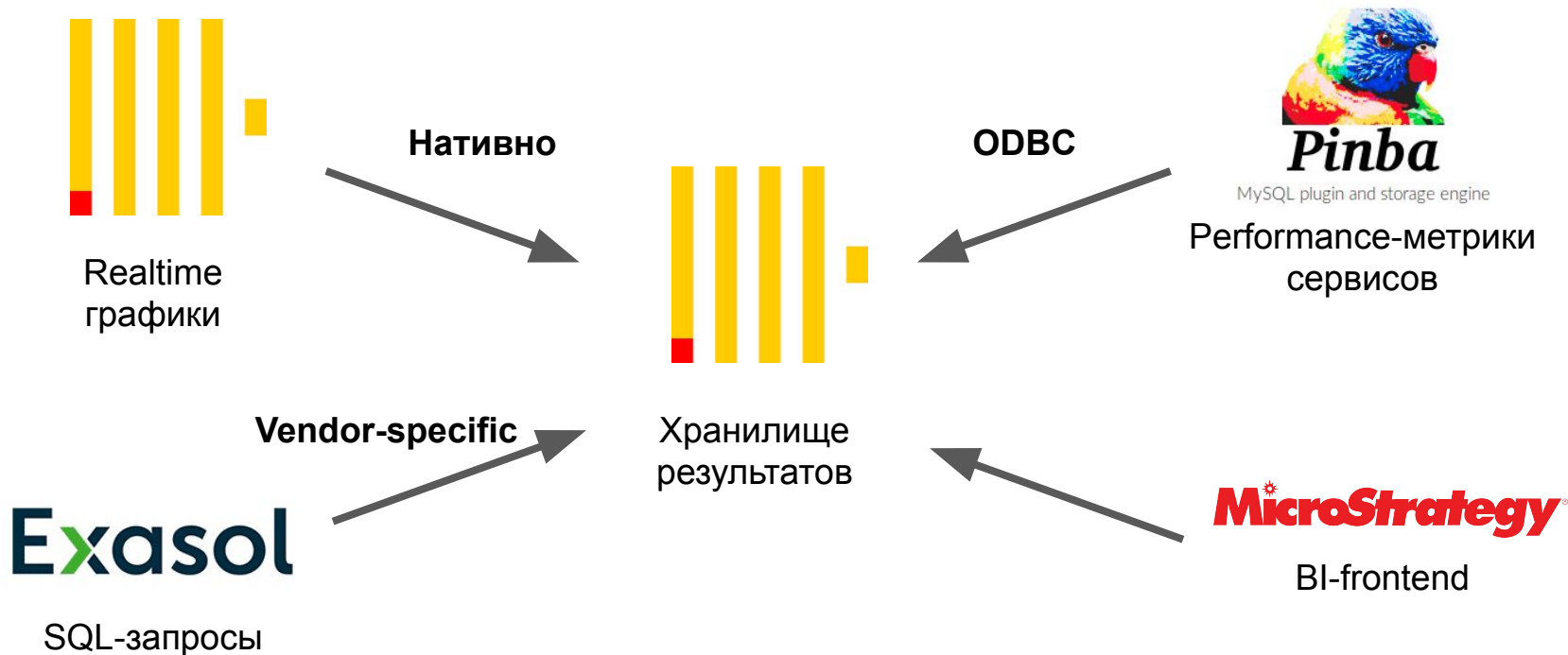
Разные источники метрик



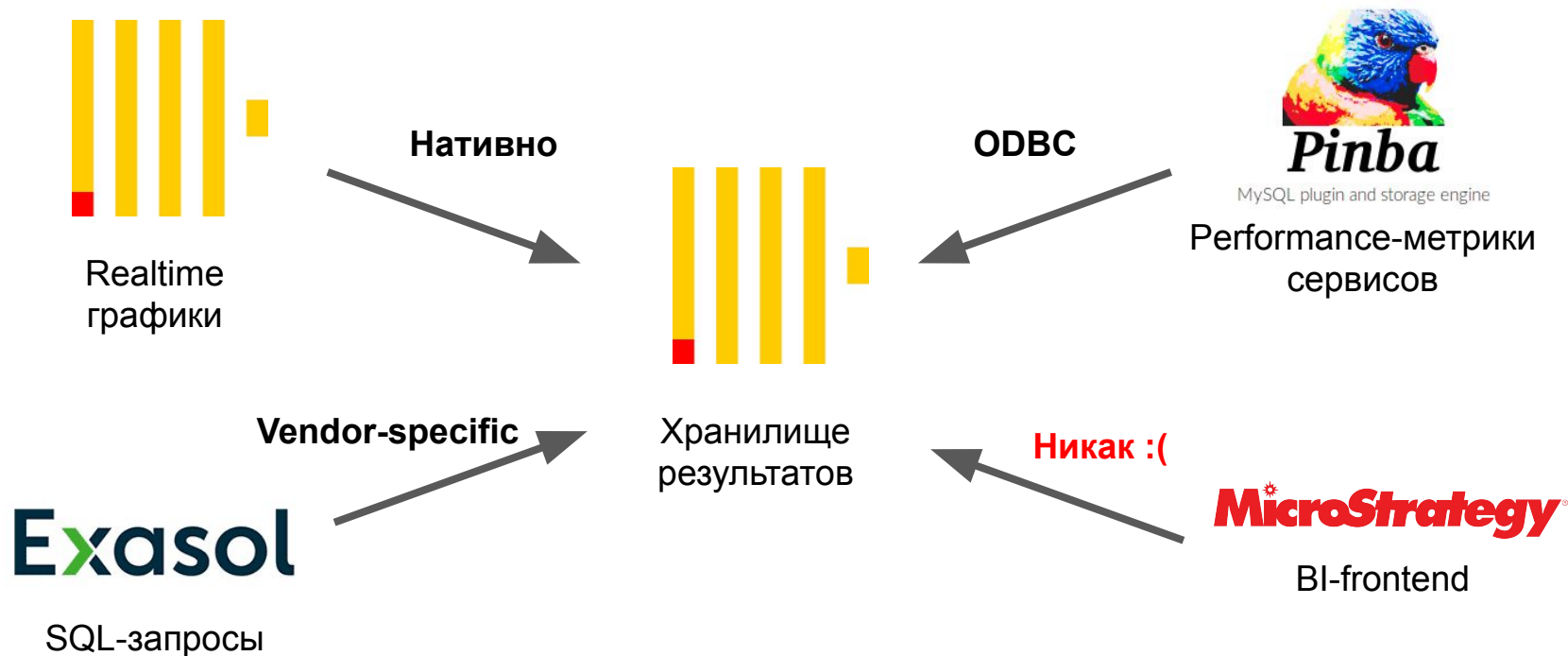
Разные источники метрик



Разные источники метрик



Разные источники метрик



**Итак, что же
у нас было?**



У нас было...

- Несколько хранилищ
- Произвольные протоколы доступа
- Соединение данных из разных источников
- Математический аппарат предсказаний
- Потребность всё это собрать воедино

У нас было...

- Несколько хранилищ
- Произвольные протоколы доступа
- Соединение данных из разных источников
- Математический аппарат предсказаний
- Потребность всё это собрать воедино



 +  ClickHouse = ?

Сравнение двух гигантов Big Data

Apache Spark

- Фреймворк для распределённых вычислений
- Лямбда-архитектура, MapReduce
- Поддержка SQL
- Произвольные преобразования данных на любом ЯП
- Big-data oriented
- Java, Hadoop, high latency

Yandex ClickHouse

- Вы сами всё знаете

Сравнение

- Одно - фреймворк, другое - база данных
- Для корректности, возьмём в рассмотрение только функционал Spark SQL

Общее сравнение



Тип	фреймворк	база данных
Реализация	Scala	C++
Доступ	JVM-code, SQL, MapReduce	SQL-like
Инсталляция	локально, кластер, on Hadoop	локально, кластер

Персистентные данные



Хранение	HDFS, СУБД	локальный диск
Репликация	на уровне источника	опционально
Шардирование	на уровне источника	опционально
Партиционирование	на уровне источника	per-table

Промежуточные данные



Поддерживает	да	Частично, (GLOBAL IN)
Репликация	опционально	нет
Шардирование	да	нет
Партиционирование	да	нет

SQL



ClickHouse

Приближенные вычисления	+	+
CTE	+	-
Оконные функции	+	-
Промежуточная группировка	ROLLUP, CUBE, GROUPING SETS	ROLLUP, CUBE

```
SELECT count()  
FROM system.functions  
WHERE alias_to = ''
```

<pre>count() 579</pre>

```
grep 'def ' spark/sql/functions.scala \  
| grep -v udf \  
| wc -l
```

340

Типы данных



	Spark	ClickHouse
Скаляры	+	+
Массивы	+	+
Map (key-value)	+	-
Комплексные типы	+	-

Расширяемость



UDF	pluggable	compile
UDAF	pluggable	compile
Внешние источники	любые	xDBC, HTTP, HDFS

Особенности национальной ~~охоты~~ интеграции

В этой серии

- Как взаимодействуют Spark + ClickHouse
- Cluster to cluster взаимодействие
- Проблемы и решения

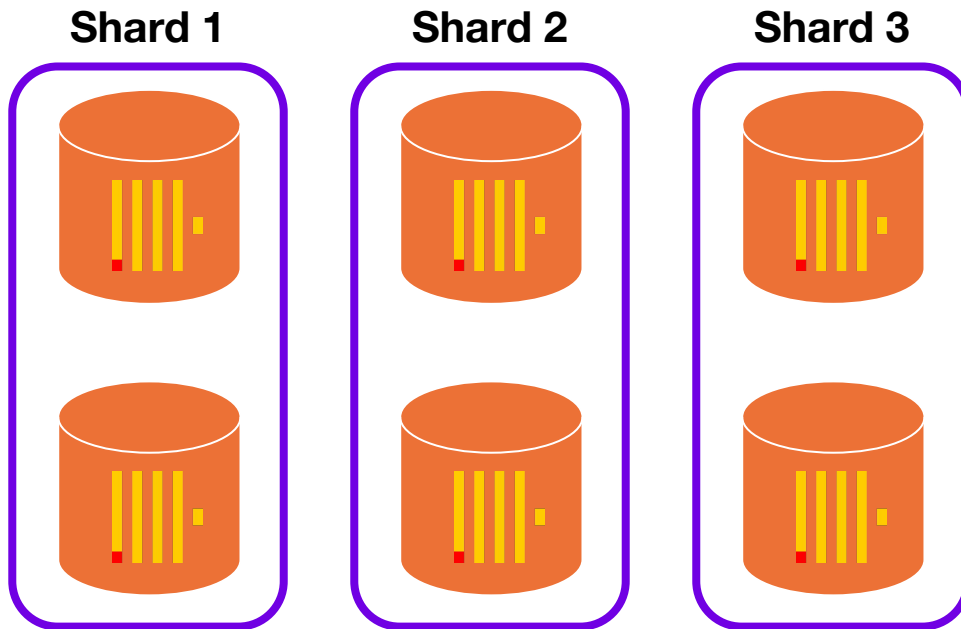
Интеграция

- Сводится к 2 моментам
 - ClickHouse -> Spark чтение
 - Spark -> ClickHouse запись

```
CREATE TABLE metrics_local
(
  `ts` DateTime,
  `metric` String,
  `value` Float32,
  `dt` MATERIALIZED toDate(ts)
)
ENGINE = ReplicatedMergeTree('/metrics')
PARTITION BY dt
ORDER BY (metric, ts)
```

```
CREATE TABLE metrics
AS metrics_local
ENGINE = Distributed(
    'cluster',
    metrics_local
)
```

Инсталляция ClickHouse



Чтение данных в Spark

- File system (HDFS, S3, local fs)
- External databases (JDBC)

Чтение данных в Spark

- File system (HDFS, S3, local fs)
 - External databases (JDBC)
-

```
public static void main(String... argv) throws Exception {  
    Dataset<Row> dataframe = getSparkSession()  
        .read()  
        .jdbc(  
            "jdbc:clickhouse://localhost:8123",  
            "(SELECT * FROM metrics LIMIT 2)",  
            getConnectionProperties()  
        );  
    dataframe.printSchema();  
    dataframe.show(2);  
}
```

```
public static void main(String... argv) throws Exception {  
    Dataset<Row> dataframe = getSparkSession()  
        .read()  
        .jdbc(  
            "jdbc:clickhouse://localhost:8123",  
            "(SELECT * FROM metrics LIMIT 2)",  
            getConnectionProperties()  
        );  
    dataframe.printSchema();  
    dataframe.show(2);  
}
```

API-объект

```
public static void main(String... argv) throws Exception {  
    Dataset<Row> dataframe = getSparkSession()  
        .read()  
        .jdbc(  
            "jdbc:clickhouse://localhost:8123",  
            "(SELECT * FROM metrics LIMIT 2)",  
            getConnectionProperties()  
        );  
    dataframe.printSchema();  
    dataframe.show(2);  
}
```

Чтение из внешнего источника

```
public static void main(String... argv) throws Exception {  
    Dataset<Row> dataframe = getSparkSession()  
        .read()  
        .jdbc(  
            "jdbc:clickhouse://localhost:8123",  
            "(SELECT * FROM metrics LIMIT 2)",  
            getConnectionProperties()  
        );  
    dataframe.printSchema();  
    dataframe.show(2);  
}
```

Через pluggable JDBC драйвер

```
public static void main(String... argv) throws Exception {  
    Dataset<Row> dataframe = getSparkSession()  
        .read()  
        .jdbc(  
            "jdbc:clickhouse://localhost:8123",  
            "(SELECT * FROM metrics LIMIT 2)",  
            getConnectionProperties()  
        );  
    dataframe.printSchema();  
    dataframe.show(2);  
}
```

Из Distributed таблицы

Cxema Spark DataFrame

root

|-- ts: timestamp (nullable = true)

|-- metric: string (nullable = true)

|-- value: float (nullable = true)

Output

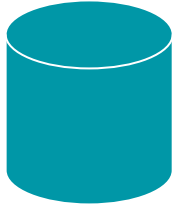
```
+-----+-----+-----+
|          ts|  metric|value|
+-----+-----+-----+
|2019-06-15 12:50:25|metric_1|100.0|
|2019-06-15 12:50:28|metric_2|100.0|
+-----+-----+-----+
```

Исполнение Spark-кода

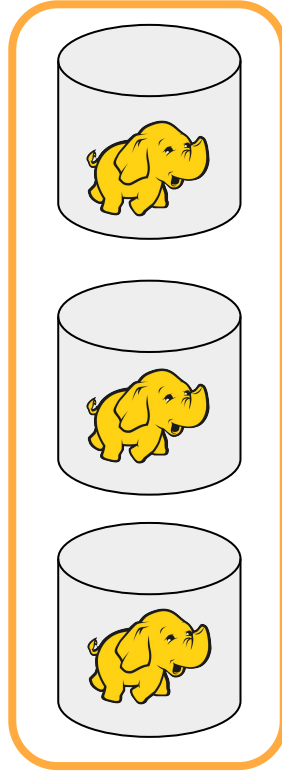
- Управляющая программа (driver)
- Пул исполнителей (executor)
- Driver -> Executor:

“Выполни запрос в ClickHouse и сохрани результат”

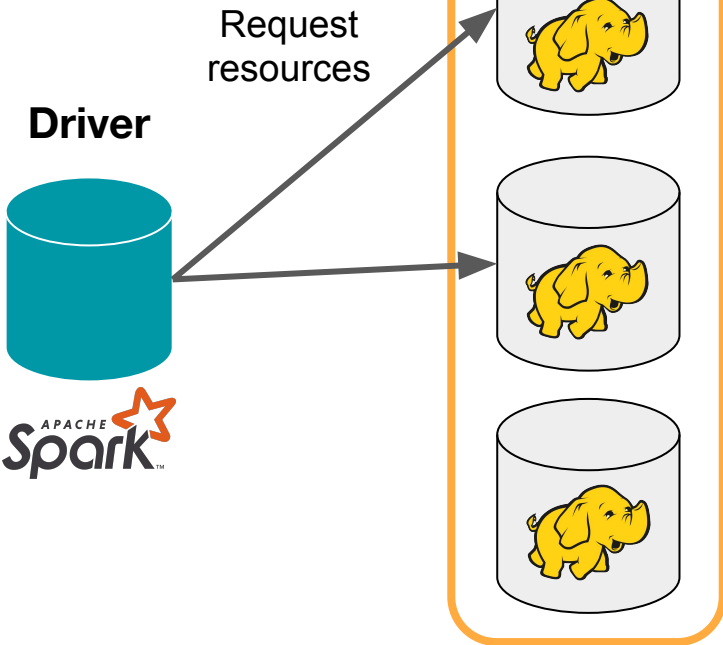
Driver



Hadoop



Hadoop



Driver



Hadoop



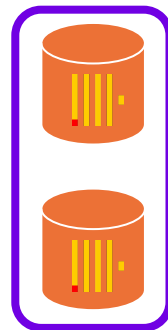
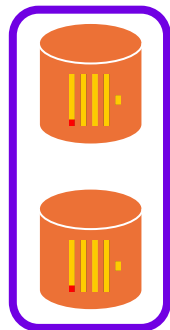
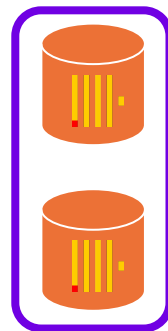
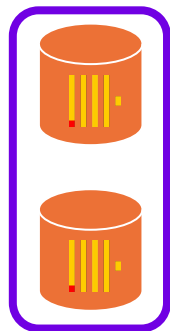
Driver



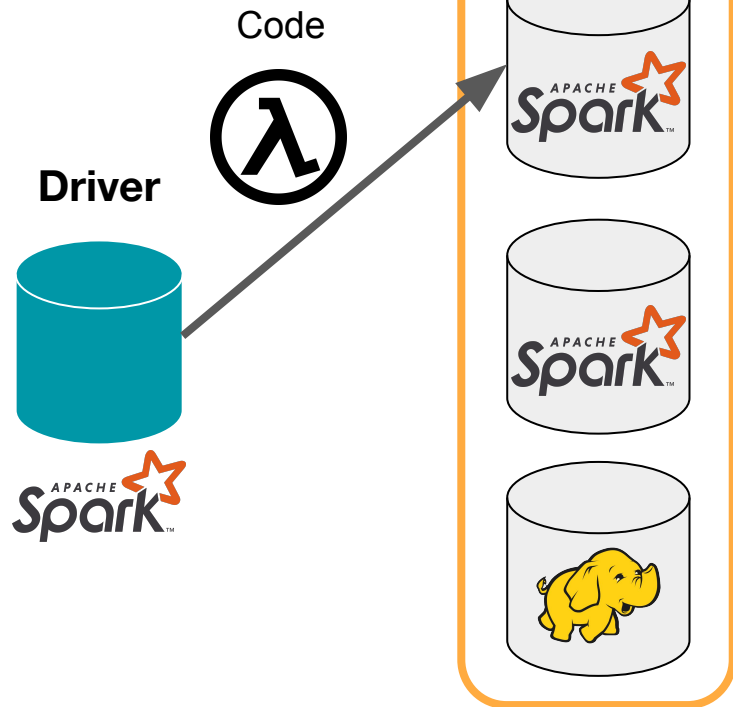
Hadoop



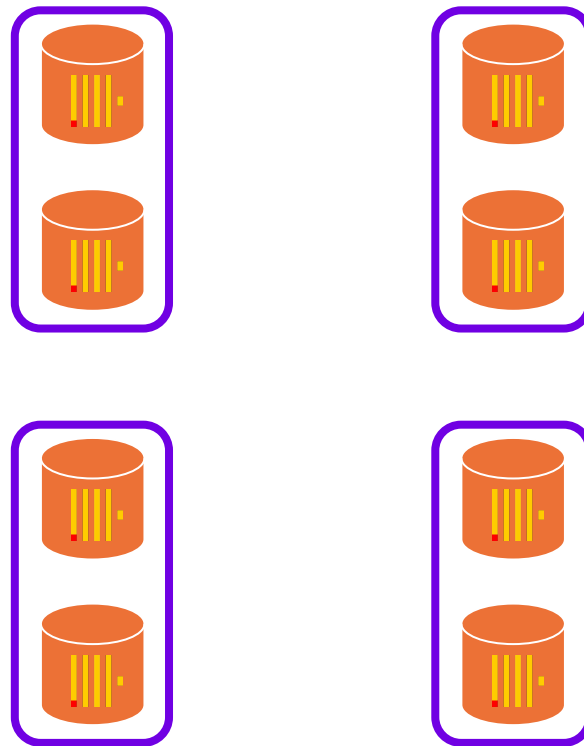
ClickHouse

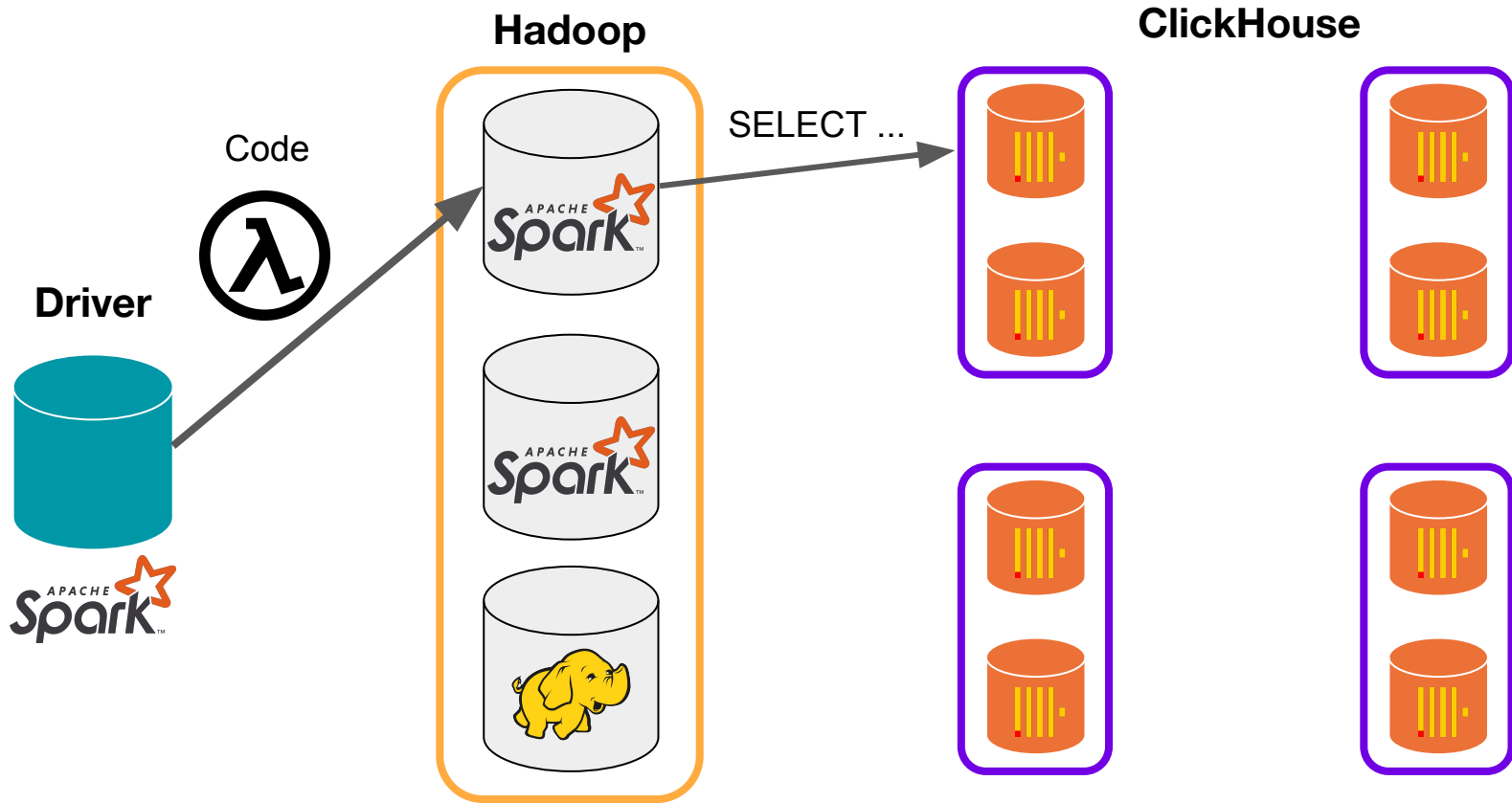


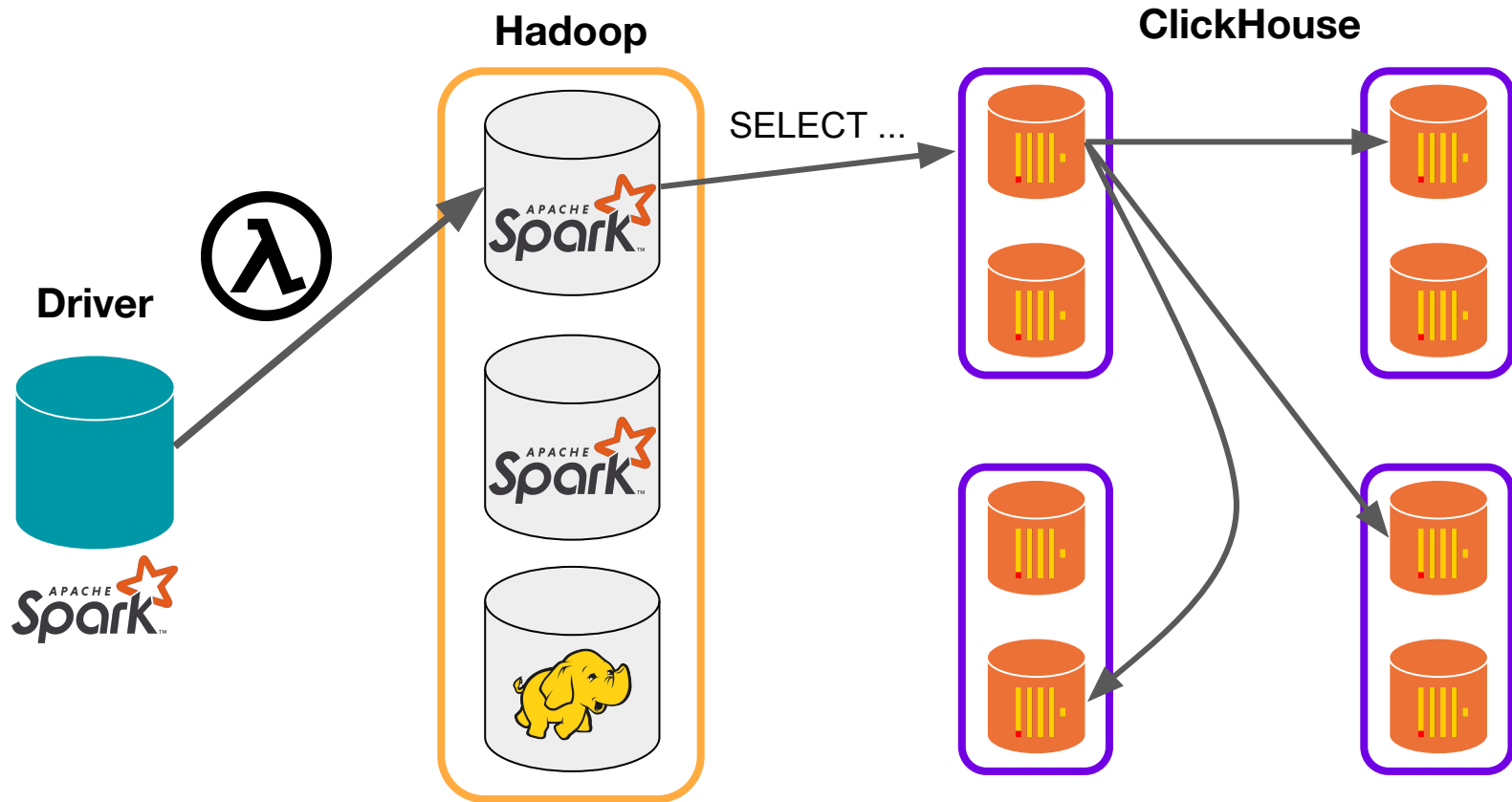
Hadoop

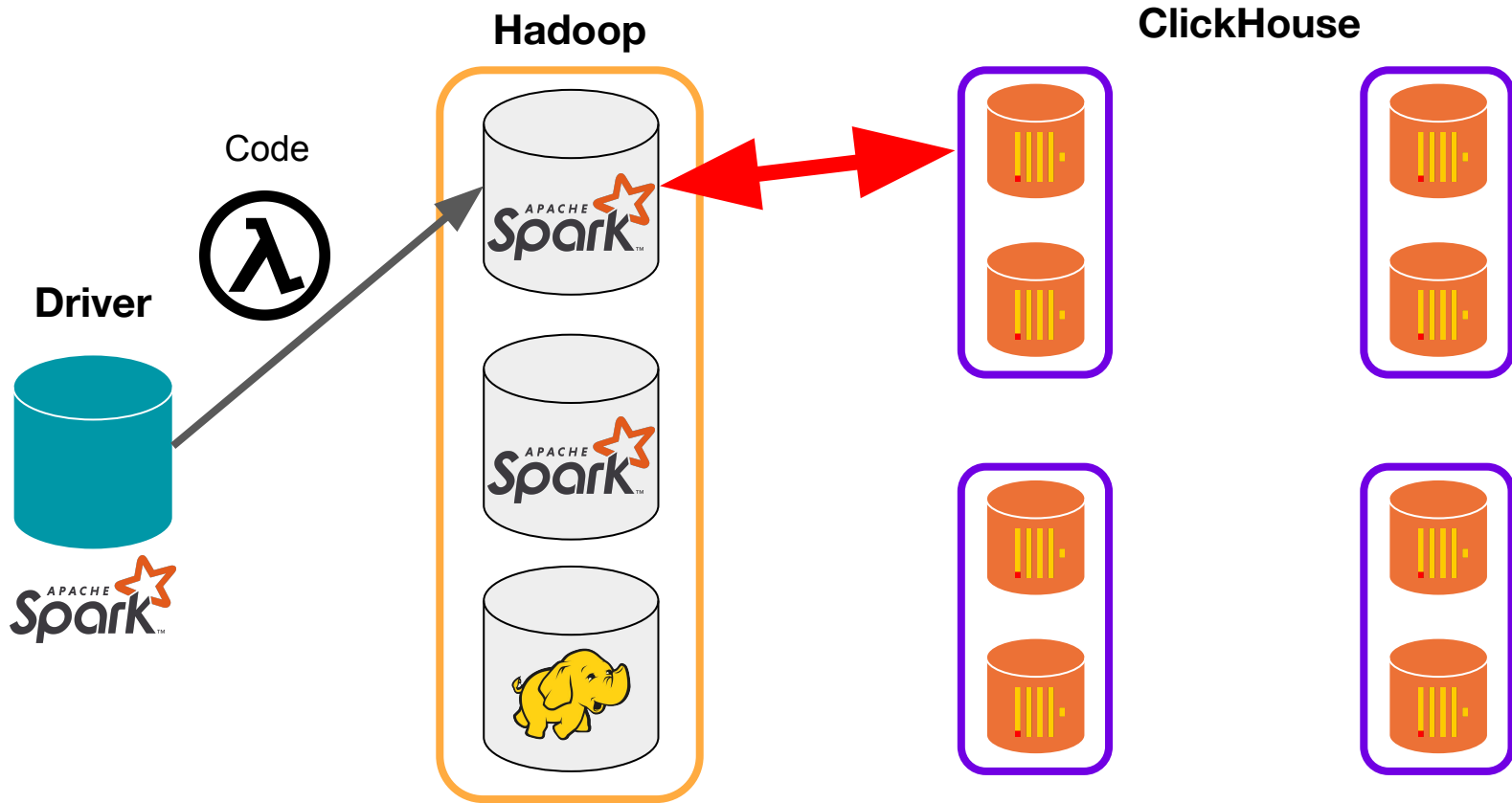


ClickHouse









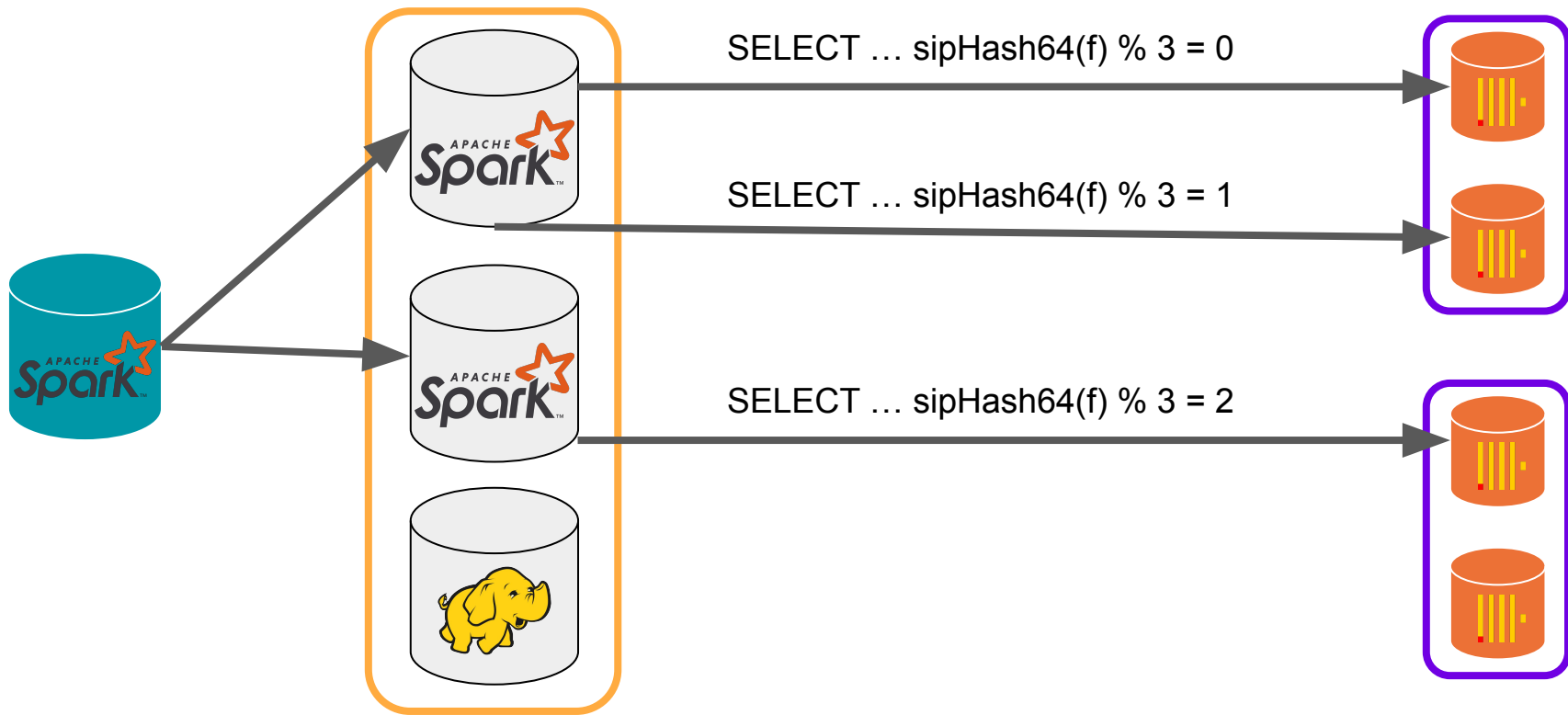
Проблемы

- Весь dataset в один executor
- OutOfMemoryError
- Однопоточная передача данных
- Repartition на стороне Spark

Решение

- Разбить resultset ClickHouse на отдельные запросы
 - `sipHash64(field) % num_queries = YYY`
 - per-shard partitioning
- Запрос к MergeTree в шарде/реплике

Шардированный запрос



```
Dataset<Row> result = sparkSession.sqlContext().emptyDataFrame();  
for (int i = 0; i < NUM_SHARDS; i++) {  
    final String query = "(SELECT * FROM metrics " +  
        "WHERE sipHash64(metric) % " + NUM_SHARDS + " = " + i + ")";  
    result = sparkSession.read().jdbc(  
        nextAvailableHost(), query, getConnectionProperties()  
    ).unionAll(result);  
}
```

```
Dataset<Row> result = sparkSession.sqlContext().emptyDataFrame();  
for (int i = 0; i < NUM_SHARDS; i++) {  
    final String query = "(SELECT * FROM metrics " +  
        "WHERE sipHash64(metric) % " + NUM_SHARDS + " = " + i + ")";  
    result = sparkSession.read().jdbc(  
        nextAvailableHost(), query, getConnectionProperties()  
    ).unionAll(result);  
}
```

Разбиваем запрос на части

```
Dataset<Row> result = sparkSession.sqlContext().emptyDataFrame();  
for (int i = 0; i < NUM_SHARDS; i++) {  
    final String query = "(SELECT * FROM metrics " +  
        "WHERE sipHash64(metric) % " + NUM_SHARDS + " = " + i + ")";  
    result = sparkSession.read().jdbc(  
        nextAvailableHost(), query, getConnectionProperties()  
    ).unionAll(result);  
}
```

Шардируем


```
Dataset<Row> result = sparkSession.sqlContext().emptyDataFrame();  
for (int i = 0; i < NUM_SHARDS; i++) {  
    final String query = "(SELECT * FROM metrics " +  
        "WHERE sipHash64(metric) % " + NUM_SHARDS + " = " + i + ")";  
    result = sparkSession.read().jdbc(  
        nextAvailableHost(), query, getConnectionProperties()  
    ).unionAll(result);  
}
```

Объединяем

Profit

- Node to node взаимодействие
- Полная утилизация ресурсов
 - Hadoop
 - ClickHouse

А почему медленно?

- Не включили сжатие
 - Настройки JDBC-соединения

А почему медленно?

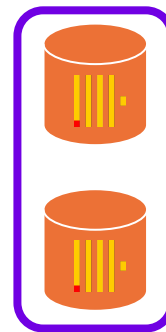
- Не включили сжатие
 - Настройки JDBC-соединения
- Накладные расходы Spark
 - Тюним по мануалу

А почему медленно?

- Не включили сжатие
 - Настройки JDBC-соединения
- Накладные расходы Spark
 - Тюним по мануалу
- Медленный парсинг ответа от ClickHouse
 - Идём курить сорцы JDBC-драйвера



`SELECT ... sipHash(metric) % 3 = 0`



Запрос
от Spark + JDBC

```
SELECT
    ts,
    metric,
    value
FROM
(
    SELECT *
    FROM metrics
    WHERE sipHash64(metric) % 3 = 0
)
FORMAT TabSeparatedWithNamesAndTypes;
```

Пользовательский запрос

```
SELECT
```

```
    ts,  
    metric,  
    value
```

```
FROM
```

```
(
```

```
    SELECT *
```

```
    FROM metrics
```

```
    WHERE sipHash64(metric) % 3 = 0
```

```
)
```

```
FORMAT TabSeparatedWithNamesAndTypes;
```



```
SELECT
```

```
    ts,  
    metric,  
    value
```

```
FROM
```

```
(
```

```
    SELECT *
```

```
    FROM metrics
```

```
    WHERE sipHash64(metric) % 3 = 0
```

```
)
```

```
FORMAT TabSeparatedWithNamesAndTypes;
```

Spark schema
inferring

```
SELECT
    ts,
    metric,
    value
FROM
(
    SELECT *
    FROM metrics
    WHERE sipHash64(metric) % 3 = 0
)
FORMAT TabSeparatedWithNamesAndTypes;
```

TabSeparated

- Мы очень любим его у себя в BI
- Но даёт большой overhead
- 40% времени - парсинг timestamp



Нормальные герои идут в обход

- Проприетарный метод в `clickhouse-jdbc`
`executeQueryClickhouseRowBinaryStream(String sql)`
- Звучит многообещающе!

```
public static void main(String[] args) throws Exception {  
    ClickHouseConnection connection = getConnection();  
    ClickHouseRowBinaryInputStream stream = connection  
        .createStatement()  
        .executeQueryClickhouseRowBinaryStream(  
            "SELECT * FROM metric"  
        );  
    Timestamp ts = stream.readDateTime();  
  
    ts.getTime();  
}
```

```
public static void main(String[] args) throws Exception {  
    ClickHouseConnection connection = getConnection();  
    ClickHouseRowBinaryInputStream stream = connection  
        .createStatement()  
        .executeQueryClickhouseRowBinaryStream(  
            "SELECT * FROM metric"  
        );  
    Timestamp ts = stream.readDateTime();  
  
    ts.getTime();  
}
```

```
public static void main(String[] args) throws Exception {  
    ClickHouseConnection connection = getConnection();  
    ClickHouseRowBinaryInputStream stream = connection  
        .createStatement()  
        .executeQueryClickhouseRowBinaryStream(  
            "SELECT * FROM metric"  
        );  
    Timestamp ts = stream.readDateTime();  
  
    ts.getTime();  
}
```

Быстрое чтение Timestamp

```
public Timestamp readDateTime() throws IOException {  
    long value = readUInt32();  
    return new Timestamp(TimeUnit.SECONDS.toMillis(value));  
}
```


String -> sipHash64(String)

```
public UnsignedLong readUInt64AsUnsignedLong() throws IOException {  
    return UnsignedLong.fromLongBits(in.readLong());  
}
```

Пример реализации

<https://github.com/alex-krash/spark-clickhouse-example>

Оптимизация

- Заменяли TabSeparated -> RowBinary
- Отказались от String в пользу sipHash64(String)
- Скорость передачи ClickHouse -> Spark

выросла в **10 раз**

Запись Spark -> ClickHouse

Представление Spark Dataset



x	y
10	20
30	40

Partition 1

x	y
50	60
70	80

Partition 2

x	y
90	100
110	120

Partition 3

x	y
130	140
150	160

Partition 4



x	y
10	20
30	40

Partition 5

x	y
50	60
70	80

Partition 6

x	y
90	100
110	120

Partition 7

x	y
130	140
150	160

Partition 8

Ситуация

- Представление данных - Spark RDD
- Resilient Distributed Dataset
- На каждом узле - партиция данных
- Необходимо сохранить данные в ClickHouse

Many 2 many. Too many?



x	y
10	20
30	40

Partition 1

x	y
50	60
70	80

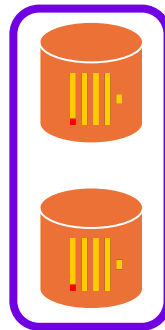
Partition 2

x	y
90	100
110	120

Partition 3

x	y
130	140
150	160

Partition 4



x	y
10	20
30	40

Partition 5

x	y
50	60
70	80

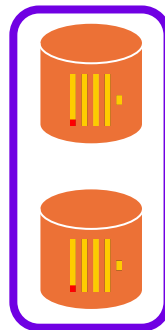
Partition 6

x	y
90	100
110	120

Partition 7

x	y
130	140
150	160

Partition 8



Варианты записи

- Слить все данные в один Spark worker, сделать INSERT

Repartitioning

Worker #1



x	y
10	20
30	40

Partition 1

x	y
50	60
70	80

Partition 2

Worker #2



x	y
10	20
30	40

Partition 5

x	y
50	60
70	80

Partition 6

Worker #3



x	y
10	20
30	40
10	20
30	40
50	60
50	60
70	80
70	80

Запись одного куска

- Если мало данных
 - Идеально, если меньше *max_insert_block_size*
- Много данных -> OutOfMemoryError
- Shuffle данных между воркерами

Варианты записи

- Слить все данные в один Spark worker, сделать INSERT
- Каждый кусок данных вставить независимо

Many 2 many

Worker #1



x	y
10	20
30	40

x	y
50	60
70	80

x	y
90	100
110	120

x	y
130	140
150	160

Worker #2

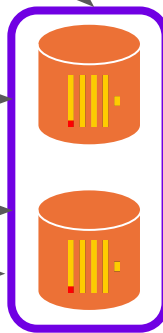
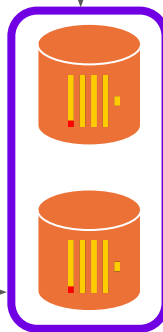


x	y
10	20
30	40

x	y
50	60
70	80

x	y
90	100
110	120

x	y
130	140
150	160



Запись many 2 many

- Spark партиций сильно больше машин ClickHouse
 - Too many parts exception
- Отсутствие транзакций со стороны ClickHouse
 - Надо думать об идемпотентности процесса записи
 - После записи - проверка, что число строк сходится

Запись many 2 many

- Партии Spark -> ClickHouse
 - Пишем в Distributed, repartition на стороне ClickHouse
 - Spark repartition, пишем в локальный MergeTree

Performance hack

- Запись Spark -> ClickHouse тормозит
 - Опять JDBC
 - Опять надо использовать проприетарный метод `sendRowBinaryStream()`

Tips and tricks

Обнаружение топологии

- Приложение на Spark должно знать про инстансы ClickHouse
- Инстансы имеют свойство становиться недоступными

В конфиге - все хосты кластера

- Надо поддерживать актуальный список

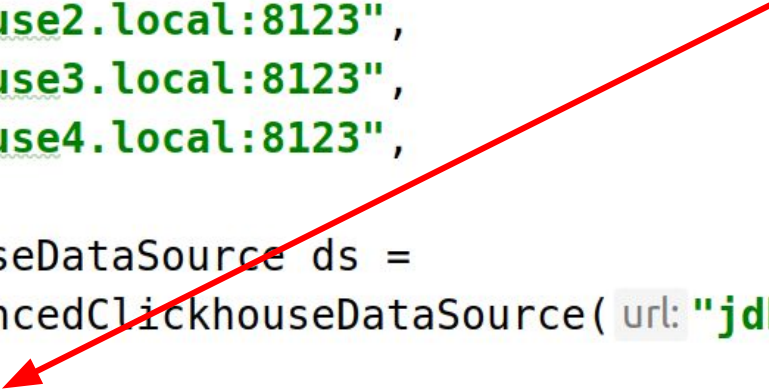
```
String hosts = StringUtils.join(new String[]{  
    "clickhouse1.local:8123",  
    "clickhouse2.local:8123",  
    "clickhouse3.local:8123",  
    "clickhouse4.local:8123",  
});  
BalancedClickhouseDataSource ds =  
    new BalancedClickhouseDataSource(url: "jdbc:clickhouse://" + hosts);  
ds.actualize();  
List<String> active = ds.getEnabledClickHouseUrls();
```

В конфиге - все хосты кластера

- Надо поддерживать актуальный список

```
String hosts = StringUtils.join(new String[]{  
    "clickhouse1.local:8123",  
    "clickhouse2.local:8123",  
    "clickhouse3.local:8123",  
    "clickhouse4.local:8123",  
});  
BalancedClickhouseDataSource ds =  
    new BalancedClickhouseDataSource(url: "jdbc:clickhouse://" + hosts);  
ds.actualize();  
List<String> active = ds.getEnabledClickHouseUrls();
```

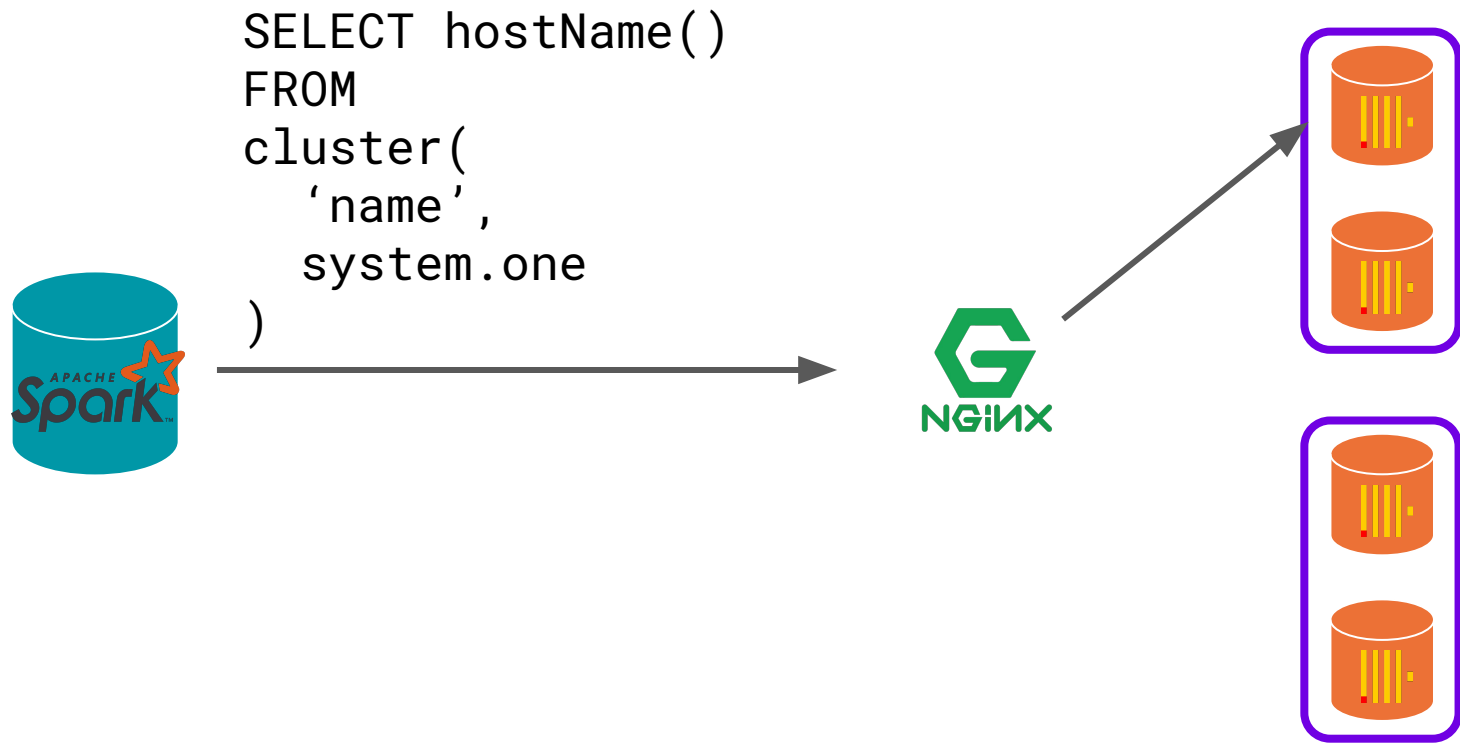
Ping каждого хоста



Proху + прямые соединения

- Nginx как прокси для ClickHouse
- В upstream описываются все хосты
- В приложение записывается адрес Nginx

Nginx as proxy: smoke test

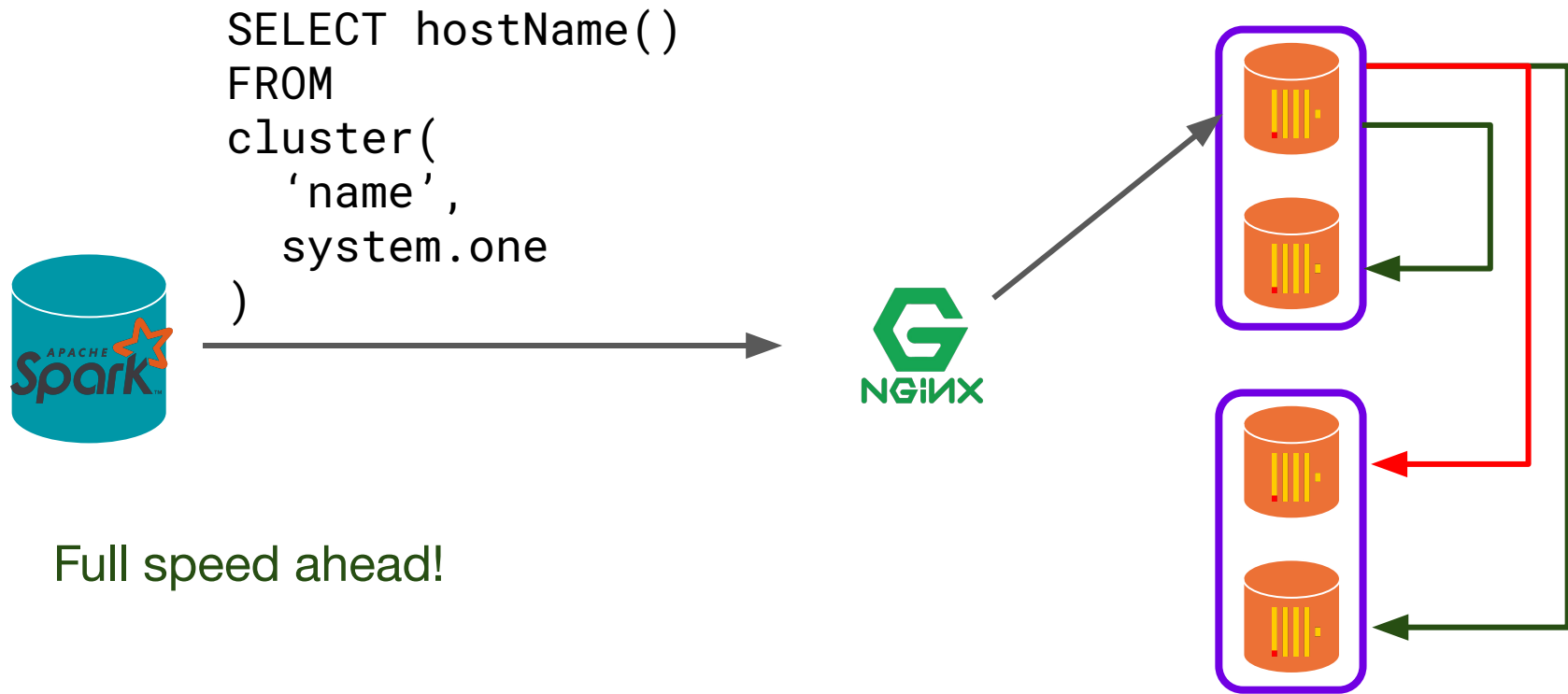


Nginx as proxy: smoke test



Неконсистентное состояние кластера

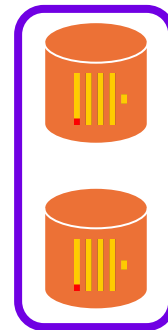
Nginx as proxy : smoke test



Full speed ahead!

Обнаружение доступных хостов

```
SELECT host_name  
FROM  
system.clusters  
WHERE cluster = 'name'
```



Использование доступных хостов

```
String hosts = StringUtils.join(new String[]{  
    "clickhouse1.local:8123",  
    "clickhouse2.local:8123",  
    "clickhouse3.local:8123",  
    "clickhouse4.local:8123",  
});  
BalancedClickhouseDataSource ds =  
    new BalancedClickhouseDataSource(url: "jdbc:clickhouse://" + hosts);  
ds.actualize();  
List<String> active = ds.getEnabledClickHouseUrls();
```

clickhouse-jdbc

- Contributors - welcome!
- Make JDBC great again!
 - Сделать RowBinary форматом общения по-умолчанию

Выводы

- Каждый из инструментов хорош по-своему
- Берём лучшее от каждого, и делаем
- Помним, что распределённые системы - это сложно
- Думаем головой

 +  ClickHouse = ?



Спасибо!