Spark Streaming: брать или не брать?

Евгений Ненахов

Tech Lead – центр BigData MTC Digital, к.ф.-м.н

@neltari

MTC DIGITAL

О чём поговорим

- Задача потоковой обработки данных

О чём поговорим

- Задача потоковой обработки данных
- Преимущества и недостатки Spark Streaming

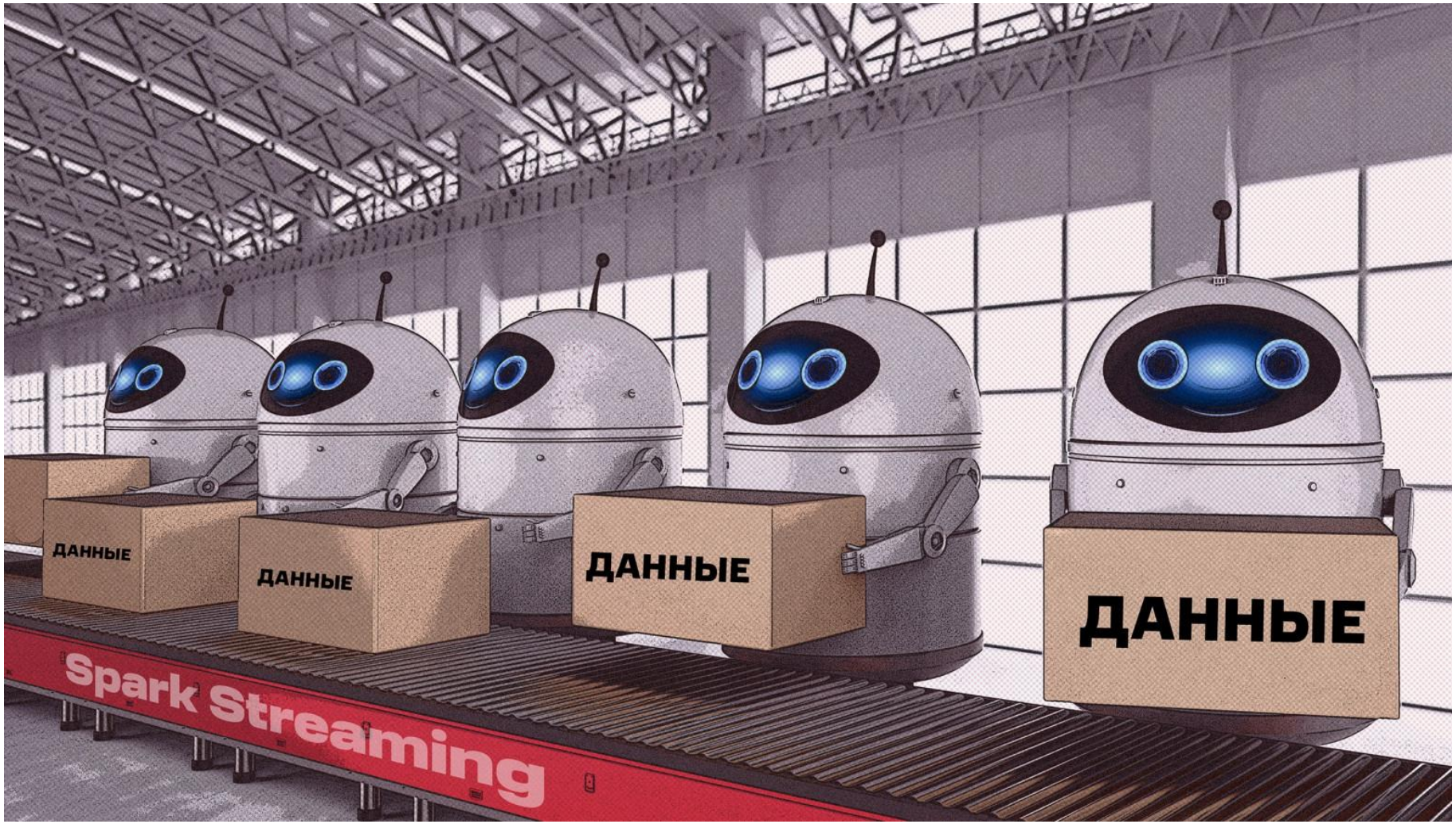
О чём поговорим

- Задача потоковой обработки данных
- Преимущества и недостатки Spark Streaming
- Типовые задачи (не) для Spark Streaming

О чём поговорим

- Задача потоковой обработки данных
- Преимущества и недостатки Spark Streaming
- Типовые задачи (не) для Spark Streaming
- Чек-лист по применению Spark Streaming

Задача потоковой обработки данных



Задача потоковой обработки данных

- Когда начинается потоковая обработка данных?

Задача потоковой обработки данных

- Когда начинается потоковая обработка данных?

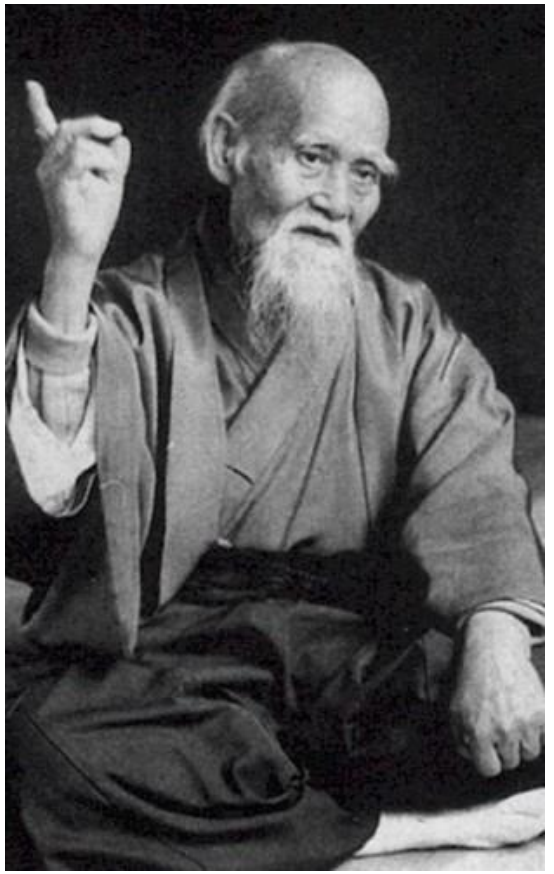
Задача потоковой обработки данных

- Когда начинается потоковая обработка данных?

Задача потоковой обработки данных

- Когда начинается потоковая обработка данных?

Баланс между точностью и скоростью обработки небольших порций данных



Инструменты

- Apache Spark Streaming
- Apache Flink
- Apache NiFi
- Apache Kafka (KSQL)
- Akka(Pekko), Vert.x
- …

Инструменты

- Apache Spark Streaming
- Apache Flink
- Apache NiFi
- Apache Kafka (KSQL)
- Akka(Pekko), Vert.x
- …

Преимущества Spark Streaming

Функционально из коробки:

Преимущества Spark Streaming

Функционально из коробки:

- window function – оконные функции для частичной агрегации

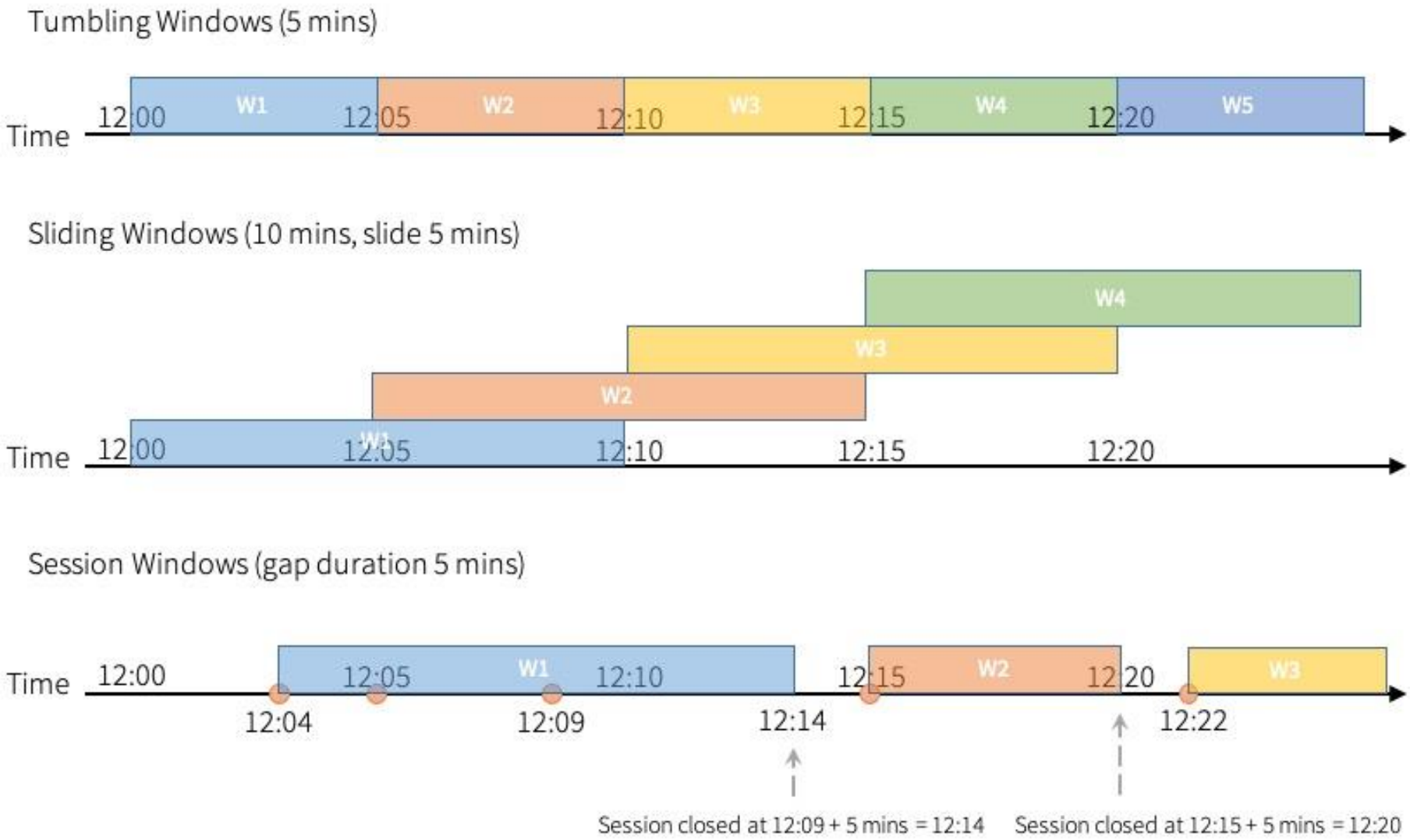
Оконные функции

```
...
lines = spark \
    .readStream \
    .format("socket") \
    .option("host", host) \
    .option("port", port) \
    .option('includeTimestamp', 'true') \
    .load()

words = lines.select(explode(split(lines.value, " ")).alias("word"), \
       lines.timestamp)

wordCounts = words.groupBy(window(words.timestamp, "3 minutes", "1 minutes"), words.word) \
    .count()
...
```

Типы оконных функций

- Classic – классические окна с данными
  - Tumbling – фиксированное
  - Sliding – скользящее перекрывающее
- Session – на основе сессии работы с данными

Типы оконных функций

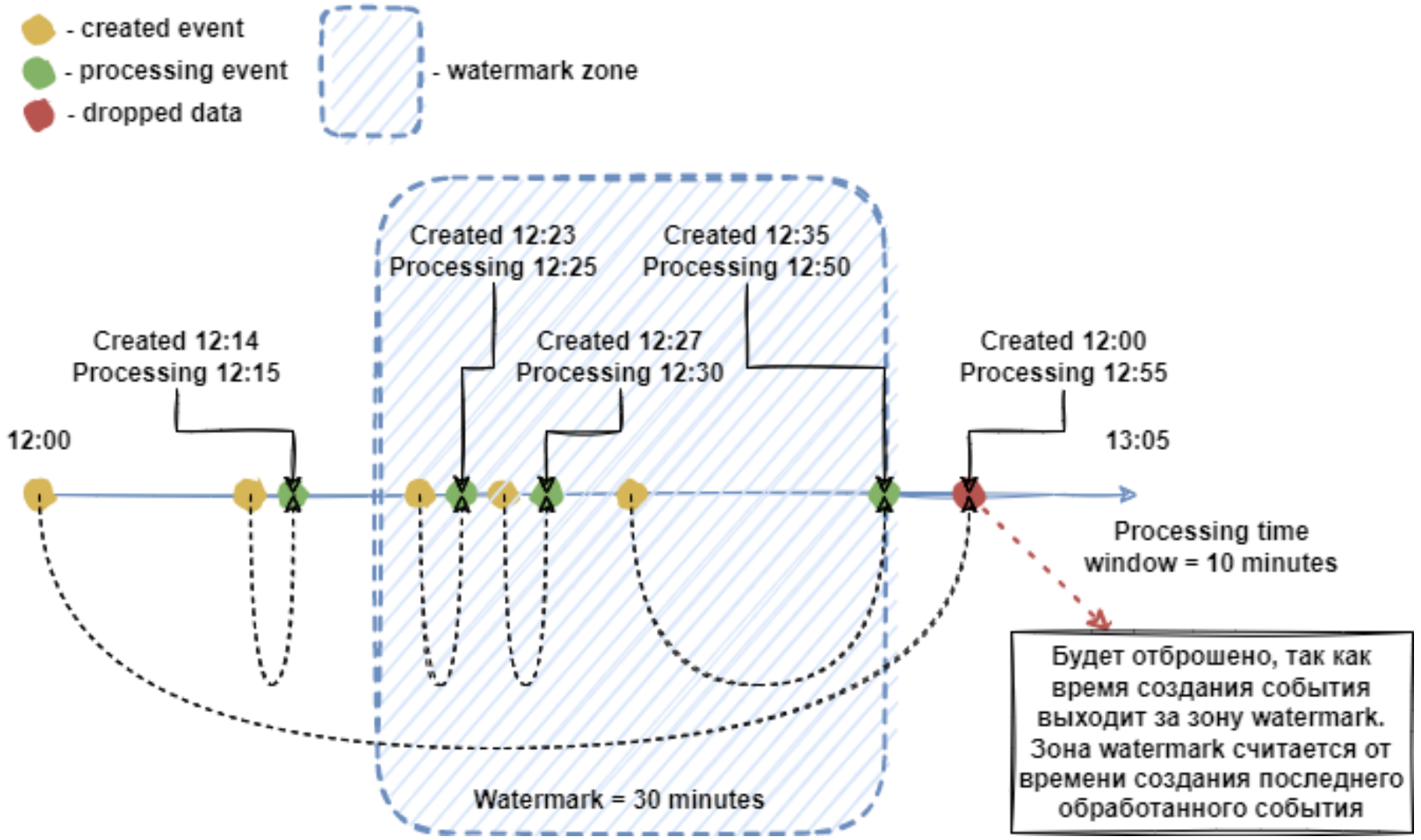


Преимущества Spark Streaming

Функционально из коробки:

- window function – оконные функции для частичной агрегации
- watermark – временные метки для фильтрации неактуальных данных

Watermark


- created event
- processing event
- dropped data
- watermark zone
Created 12:23
Processing 12:25
Created 12:35
Processing 12:50
Created 12:14
Processing 12:15
Created 12:27
Processing 12:30
Created 12:00
Processing 12:55
12:00
13:05
Processing time
window = 10 minutes
Будет отброшено, так как время создания события выходит за зону watermark. Зона watermark считается от времени создания последнего обработанного события
Watermark = 30 minutes

Watermark

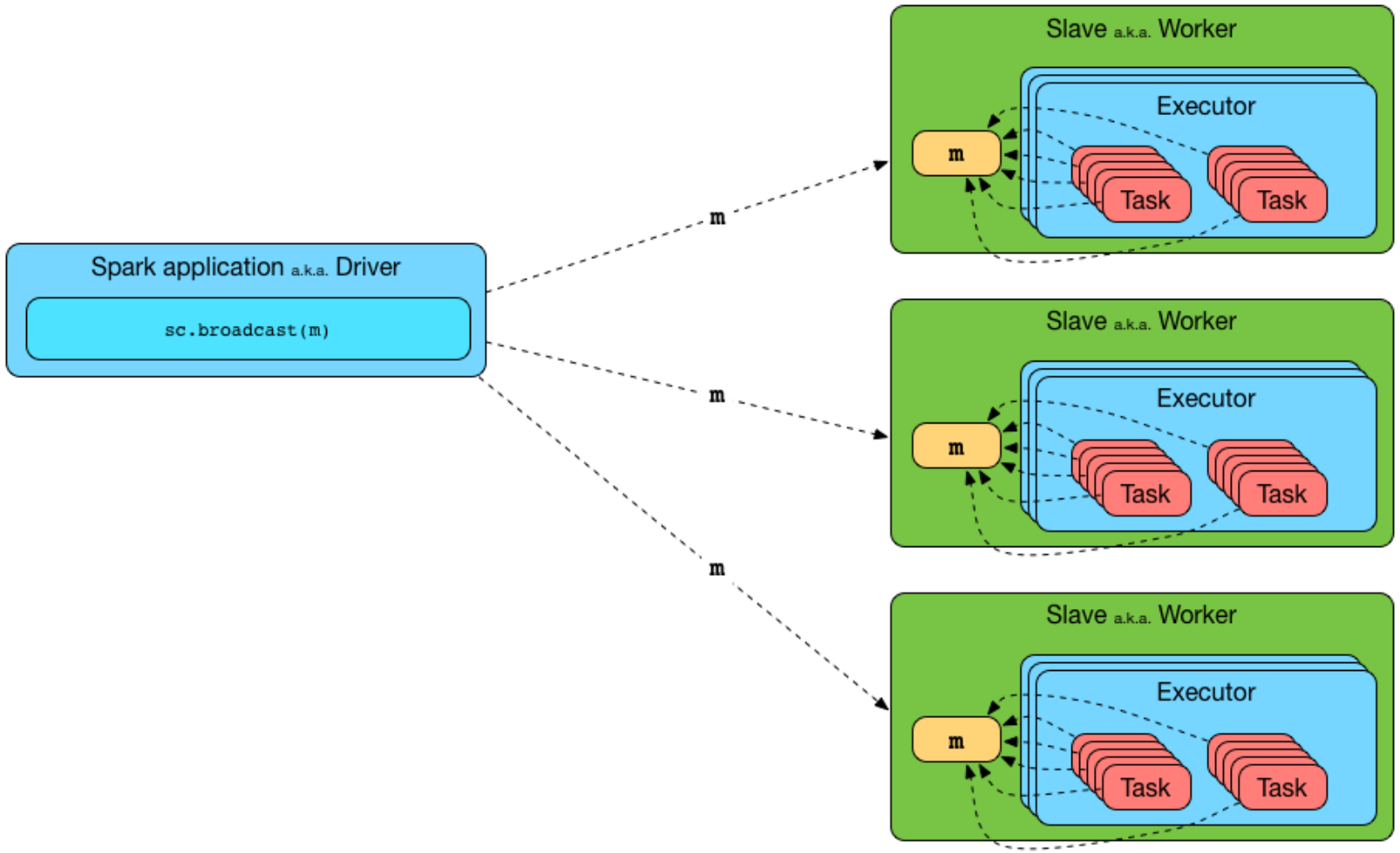
```
...
lines = spark \
    .readStream \
    .format("socket") \
    .option("host", host) \
    .option("port", port) \
    .option('includeTimestamp', 'true') \
    .load()

words = lines.select(explode(split(lines.value, " ")).alias("word"), \
       lines.timestamp)

wordCounts = words.withWatermark("timestamp", "5 minutes") \
    .groupBy(window(words.timestamp, "3 minutes", "1 minutes"), words.word) \
    .count()
...
```

Преимущества Spark Streaming

Функционально из коробки:

- window function – оконные функции для частичной агрегации
- watermark – временные метки для фильтрации неактуальных данных
- broadcast variables – глобальные read-only переменные

Broadcast variable



Преимущества Spark Streaming

Функционально из коробки:

- window function – оконные функции для частичной агрегации
- watermark – временные метки для фильтрации неактуальных данных
- broadcast variables – глобальные read-only переменные
- deduplication – избавления от дублей в потоке

Deduplication

```
...
df
  .withColumn('value', f.col('value').cast(StringType()))
  .withColumn('event', f.from_json(f.col('value'), schema))
            .selectExpr('event.*')
            .withColumn('timestamp',
                        f.from_unixtime(f.col('timestamp'), "yyyy-MM-dd' 'HH:mm:ss.SSS")
                         .cast(TimestampType()))
            .dropDuplicates(["client_id", "timestamp"])
            .withWatermark('timestamp', '10 minutes')
...
```

Преимущества Spark Streaming

Функционально из коробки:

- window function – оконные функции для частичной агрегации
- watermark – временные метки для фильтрации неактуальных данных
- broadcast variables – глобальные read-only переменные
- deduplication – избавления от дублей в потоке
- exactly-once – семантика строго одного обработанного сообщения

Exactly-once

https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#fault-tolerance-semantics

Fault Tolerance Semantics

Delivering end-to-end exactly-once semantics was one of key goals behind the design of Structured Streaming. To achieve that, we have designed the Structured Streaming sources, the sinks and the execution engine to reliably track the exact progress of the processing so that it can handle any kind of failure by restarting and/or reprocessing. Every streaming source is assumed to have offsets (similar to Kafka offsets, or Kinesis sequence numbers) to track the read position in the stream. The engine uses checkpointing and write-ahead logs to record the offset range of the data being processed in each trigger. The streaming sinks are designed to be idempotent for handling reprocessing. Together, using replayable sources and idempotent sinks, Structured Streaming can ensure **end-to-end exactly-once semantics** under any failure.

Exactly-once

https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#fault-tolerance-semantics

Fault Tolerance Semantics

Delivering end-to-end exactly-once semantics was one of key goals behind the design of Structured Streaming. To achieve that, we have designed the Structured Streaming sources, the sinks and the execution engine to reliably track the exact progress of the processing so that it can handle any kind of failure by restarting and/or reprocessing. Every streaming source is assumed to have offsets (similar to Kafka offsets, or Kinesis sequence numbers) to track the read position in the stream. The engine uses checkpointing and write-ahead logs to record the offset range of the data being processed in each trigger. The streaming sinks are designed to be idempotent for handling reprocessing. Together, using replayable sources and idempotent sinks, Structured Streaming can ensure **end-to-end exactly-once semantics** under any failure.

Какой ценой?

Преимущества Spark Streaming

Функционально из коробки:

- window function – оконные функции для частичной агрегации
- watermark – временные метки для фильтрации неактуальных данных
- broadcast variables – глобальные read-only переменные
- exactly-once – семантика строго одного обработанного сообщения
- объединение потоков (join) – stream-stream, stream-static

Join stream

```
impressions = spark.readStream....
clicks = spark.readStream....

impressionsWithWatermark = impressions.withWatermark("impressionTime", "2 hours")
clicksWithWatermark = clicks.withWatermark("clickTime", "3 hours")

impressionsWithWatermark.join(
    clicksWithWatermark,
    expr("""
    clickAdId = impressionAdId AND
    clickTime >= impressionTime AND
    clickTime <= impressionTime + interval 1 hour
    """)
)
```

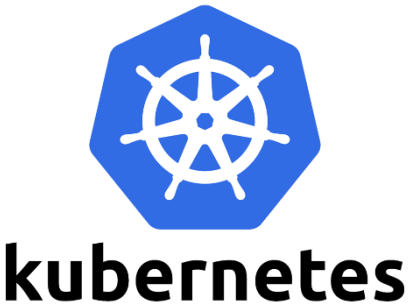
Преимущества Spark Streaming

Нефункционально:

Преимущества Spark Streaming

Нефункционально:

- Менеджер ресурсов – YARN, K8s



Преимущества Spark Streaming

Нефункционально:

- Менеджер ресурсов – YARN, K8s
- Высокая пропускная способность, отказоустойчивость

Преимущества Spark Streaming

Нефункционально:

- Менеджер ресурсов – YARN, K8s
- Высокая пропускная способность, отказоустойчивость
- Одна кодовая база для batch и streaming

```
...
lines = spark \
    .read \
...


...
lines = spark \
    .readStream \
...
```

Преимущества Spark Streaming

Нефункционально:

- Менеджер ресурсов – YARN, K8s
- Высокая пропускная способность, отказоустойчивость
- Одна кодовая база для batch и streaming
- Blacklist и speculation – для отказоустойчивости и надёжности

Преимущества Spark Streaming

Нефункционально:

- Менеджер ресурсов – YARN, K8s
- Высокая пропускная способность, отказоустойчивость
- Одна кодовая база для batch и streaming
- Blacklist и speculation – для отказоустойчивости и надёжности

```
--conf "spark.blacklist.enabled=true"
--conf "spark.executor.heartbeatInterval" ~ 20s
```

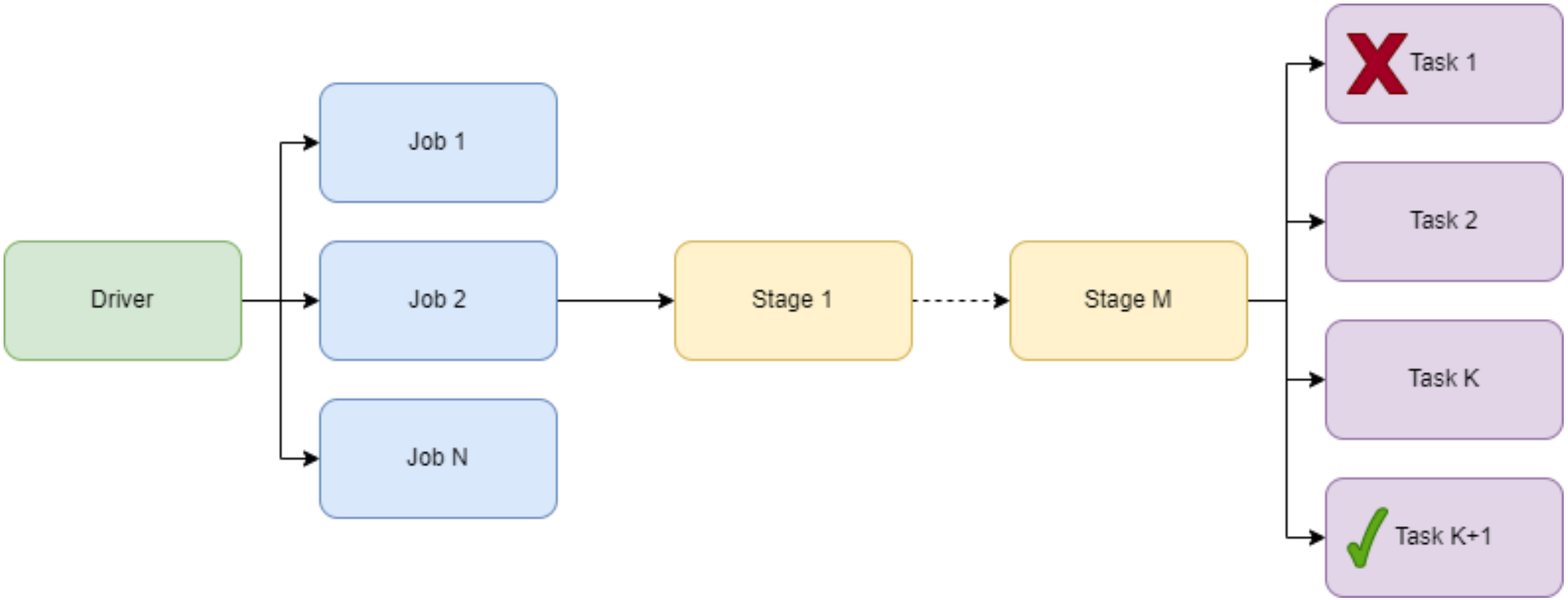
Преимущества Spark Streaming

Нефункционально:

- Менеджер ресурсов – YARN, K8s
- Высокая пропускная способность, отказоустойчивость
- Одна кодовая база для batch и streaming
- Blacklist и speculation – для отказоустойчивости и надёжности

```
--conf "spark.speculation=true"
--conf "spark.speculation.multiplier=3"
--conf "spark.speculation.quantile=0.90"
```

Speculation

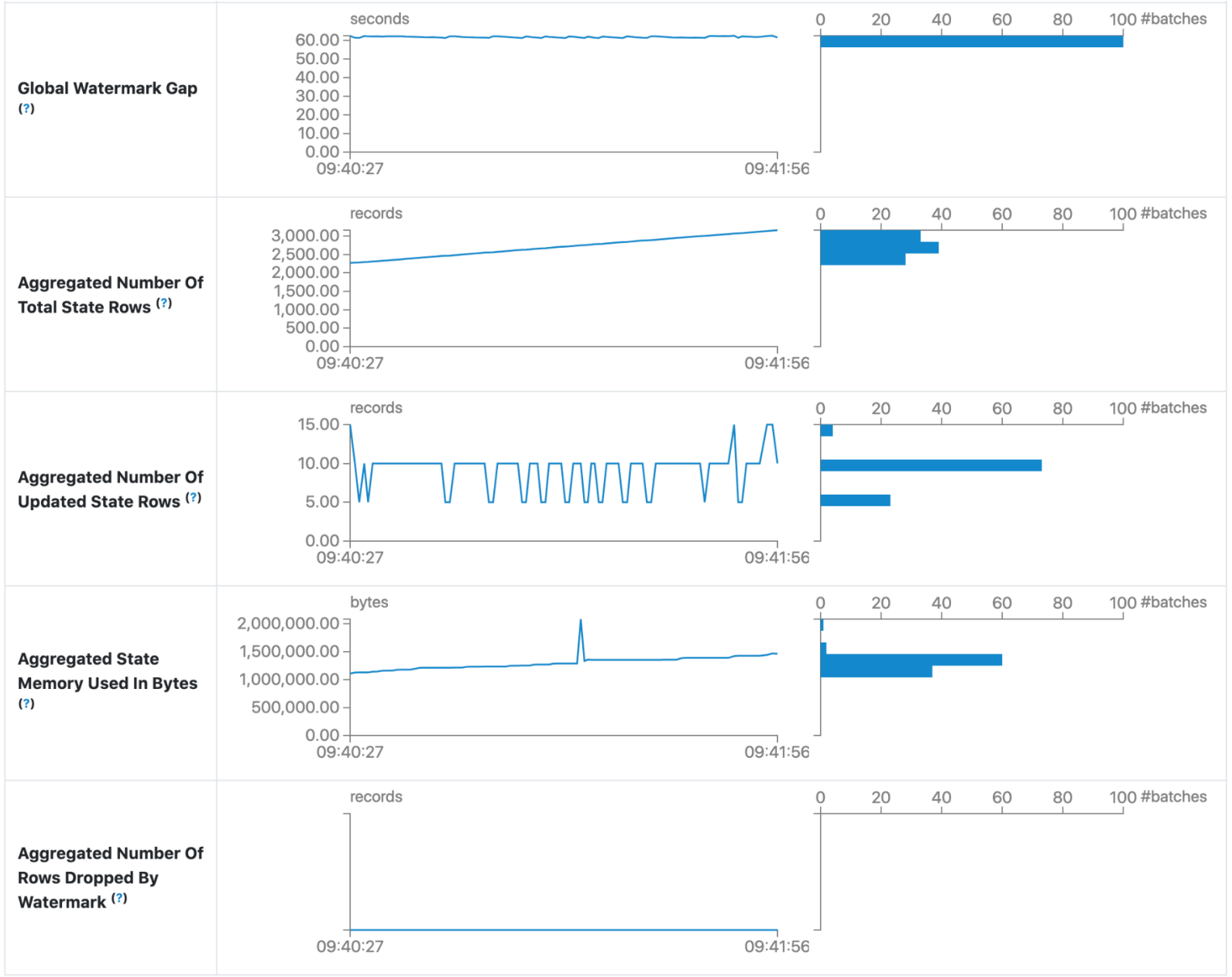


Преимущества Spark Streaming

Нефункционально:

- Менеджер ресурсов – YARN, K8s
- Высокая пропускная способность, отказоустойчивость
- Одна кодовая база для batch и streaming
- Blacklist и speculation – для отказоустойчивости и надёжности
- UI – для мониторинга и анализа Spark-приложения

Spark UI


Global Watermark Gap (?)
seconds
60.00
50.00
40.00
30.00
20.00
10.00
0.00
09:40:27
09:41:56
0 20 40 60 80 100 #batches
Aggregated Number Of Total State Rows (?)
records
3,000.00
2,500.00
2,000.00
1,500.00
1,000.00
500.00
0.00
09:40:27
09:41:56
0 20 40 60 80 100 #batches
Aggregated Number Of Updated State Rows (?)
records
15.00
10.00
5.00
0.00
09:40:27
09:41:56
0 20 40 60 80 100 #batches
Aggregated State Memory Used In Bytes (?)
bytes
2,000,000.00
1,500,000.00
1,000,000.00
500,000.00
0.00
09:40:27
09:41:56
0 20 40 60 80 100 #batches
Aggregated Number Of Rows Dropped By Watermark (?)
records
09:40:27
09:41:56
0 20 40 60 80 100 #batches

Недостатки Spark Streaming

Недостатки Spark Streaming

- Функционально:
  - True (?) streaming (continues processing experiment)

https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#continuous-processing

Continuous Processing

[Experimental]

Continuous processing is a new, experimental streaming execution mode introduced in Spark 2.3 that enables low (~1 ms) end-to-end latency with at-least-once fault-tolerance guarantees. Compare this with the default *micro-batch processing* engine which can achieve exactly-once guarantees but achieve latencies of ~100ms at best. For some types of queries (discussed below), you can choose which mode to execute them in without modifying the application logic (i.e. without changing the DataFrame/Dataset operations).

Недостатки Spark Streaming

- Функционально:
  - True (?) streaming (continues processing experiment)

```
spark \
    .readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "host1:port,host2:port") \
    .option("subscribe", "topic_source") \
    .load() \
    .selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)") \
    .writeStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "host1:port1,host2:port2") \
    .option("topic", "topic_sink") \
    .trigger(continuous="1 second") \ # .trigger(availableNow=True) \
    .start()
```

Недостатки Spark Streaming

- Функционально:
  - True (?) streaming (continues processing experiment)
  - Мало готовых коннекторов: File, Kafka, Foreach/ForeachBatch

Недостатки Spark Streaming

- Функционально:
  - True (?) streaming (continues processing experiment)
  - Мало готовых коннекторов: File, Kafka, Foreach/ForeachBatch

https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#using-foreach-and-foreachbatch

Note:

- By default, `foreachBatch` provides only at-least-once write guarantees. However, you can use the batchId provided to the function as way to deduplicate the output and get an exactly-once guarantee.
- `foreachBatch` does not work with the continuous processing mode as it fundamentally relies on the micro-batch execution of a streaming query. If you write data in the continuous mode, use `foreach` instead.

Недостатки Spark Streaming

- Функционально:
  - True (?) streaming (continues processing experiment)
  - Мало готовых коннекторов: File, Kafka, Foreach/ForeachBatch
  - DataFrame API ограничен – нет возможности управления коммитом оффсетов, применение функторов map, flatMap и т.п.

Недостатки Spark Streaming

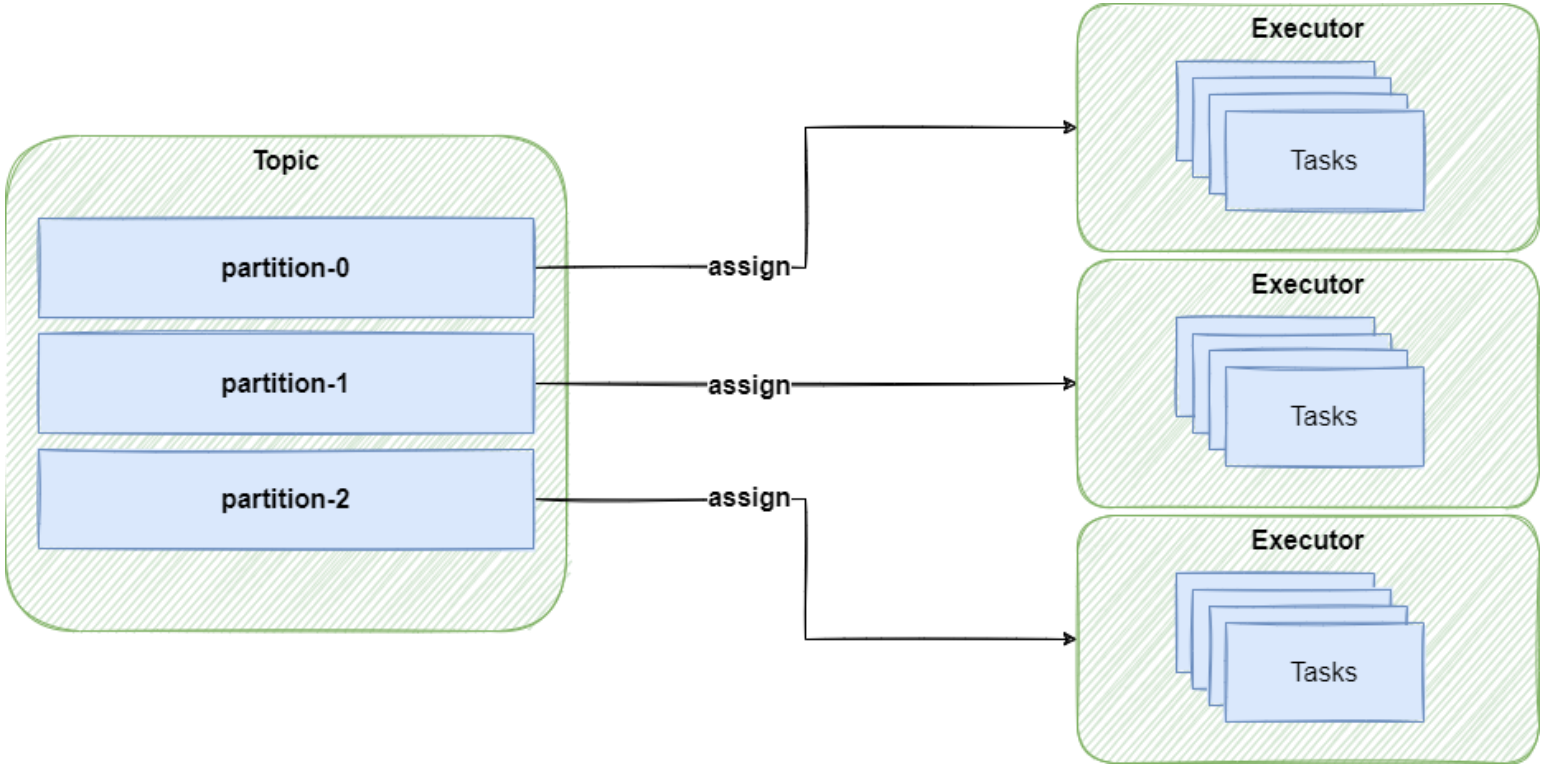
- Нефункционально:
  - Нужна тонкая конфигурация на каждый поток – сразу не поедет

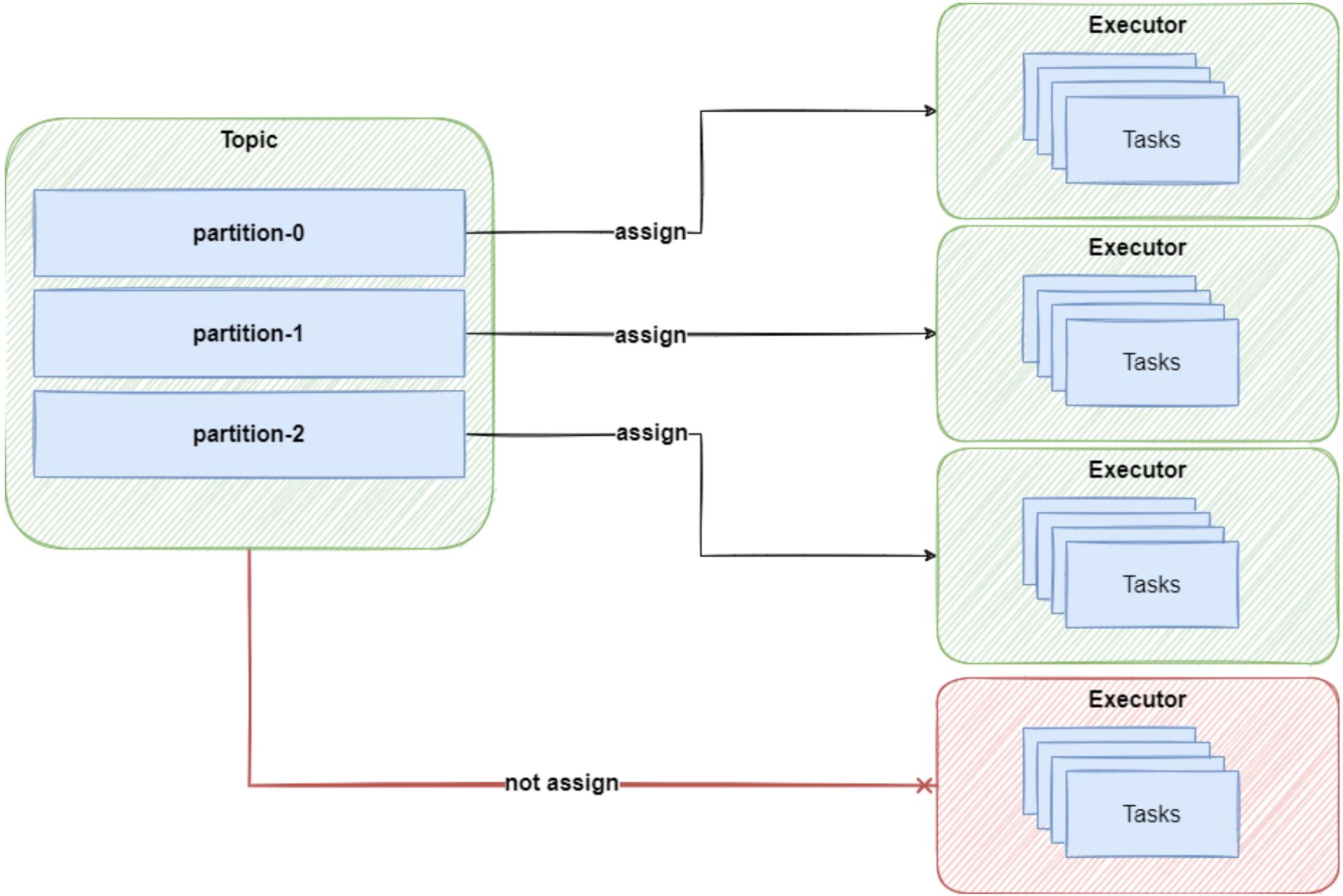
Недостатки Spark Streaming

- Нефункционально:
  - Нужна тонкая конфигурация на каждый поток – сразу не поедет
  - Горизонтальное масштабирование зависит от источника

Недостатки Spark Streaming

- Нефункционально:
  - Нужна тонкая конфигурация на каждый поток – сразу не поедет
  - Горизонтальное масштабирование зависит от источника




Topic
partition-0
partition-1
partition-2
assign
assign
assign
not assign
Executor
Tasks
Executor
Tasks
Executor
Tasks
Executor
Tasks

Недостатки Spark Streaming

- Нефункционально:
  - Нужна тонкая конфигурация на каждый поток – сразу не поедет
  - Горизонтальное масштабирование зависит от источника
  - Изменение интенсивности данных приводит к непредсказуемым последствиям

Недостатки Spark Streaming

- Нефункционально:
  - Нужна тонкая конфигурация на каждый поток – сразу не поедет
  - Горизонтальное масштабирование зависит от источника
  - Изменение интенсивности данных приводит к непредсказуемым последствиям

Доклад:
Валерия Дымбицкая — Автоматический тюнинг Spark-приложений

Недостатки Spark Streaming

- Нефункционально:
  - Нужна тонкая конфигурация на каждый поток – сразу не поедет
  - Горизонтальное масштабирование зависит от источника
  - Изменение интенсивности данных приводит к непредсказуемым последствиям
  - Отсутствует UI для управления потоками данных

Типовые задачи для Spark Streaming

Spark Structured Streaming

Streaming tasks

Типовые задачи для Spark Streaming

- Большой поток данных (миллионы сообщений) из Kafka с записью на HDFS или S3.

Типовые задачи для Spark Streaming

- Большой поток данных (миллионы сообщений) из Kafka с записью на HDFS или S3.

Доклад:
Денис Ефаров — 100 миллиардов сообщений в Kafka: загрузил и забыл

Типовые задачи для Spark Streaming

- Большой поток данных (миллионы сообщений) из Kafka с записью на HDFS или S3.

```
spark \
    .readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "host1:port1,host2:port2") \
    .option("subscribe", "topic_name") \
    .option("maxOffsetsPerTrigger", "1000000") \
    .load() \
    .writeStream \
    .trigger(processingTime = '1 minutes') \ # .trigger(availableNow=True) \
    .format("PARQUET") \
    .option("path", "/path/to/storage/date") \
.start()
```

Типовые задачи для Spark Streaming

- Большой поток данных (миллионы сообщений) с дедубликацией без внешнего хранилища

Типовые задачи для Spark Streaming

- Большой поток данных (миллионы сообщений) с дедубликацией без внешнего хранилища

```
spark.readStream.format('kafka') \
    .option('kafka.bootstrap.servers', 'host:port').option("subscribe", "client.stream") \
    .options(**kafka_security_options) \
    .load() \
    .withColumn('value', f.col('value').cast(StringType())) \
    .withColumn('event', f.from_json(f.col('value'), schema)) \
    .selectExpr('event.*') \
    .withColumn('timestamp',f.from_unixtime(f.col('timestamp'), "yyyy-MM-dd' 'HH:mm:ss.SSS")
                .cast(TimestampType())) \
    .dropDuplicates(["client_id", "timestamp"]).withWatermark('timestamp', '10 minutes') \
    .writeStream \
    .outputMode("append") \
    .format("console") \
    .option("truncate", False) \
    .trigger(availableNow=True) \
    .start()
```

Типовые задачи для Spark Streaming

- Обогащение большого потока данных (миллионы сообщений) справочной информацией и записью в несколько синков

Типовые задачи для Spark Streaming

- Обогащение большого потока данных (миллионы сообщений) справочной информацией и записью в несколько синков

```
incomming_message_schema = ...

kafka_data = spark.readStream.format('kafka').option(…)….load() \
    .select(from_json(col("value").cast("string"), incomming_message_schema))

postgresql_data = spark.read.format('jdbc').option(…)….load()

result_df = kafka_data.join(postgresql_data, col("kafka_join_col") == col("pg_join_col")) \
    .select(...)

result_df.writeStream \
    .foreachBatch(foreach_batch_function) \
    .start() \
    .awaitTermination()
```

Типовые задачи для Spark Streaming

- Обогащение большого потока данных (миллионы сообщений) справочной информацией и записью в несколько синков

```
incomming_message_schema = ...

kafka_data = spark.readStream.format('kafka').option(…)….load() \
    .select(from_json(col("value").cast("string"), incomming_message_schema))

postgresql_data = spark.read.format('jdbc').option(…)….load()

result_df = kafka_data.join(postgresql_data, col("kafka_join_col") == col("pg_join_col")) \
    .select(...)

result_df.writeStream \
    .foreachBatch(foreach_batch_function) \
    .start() \
    .awaitTermination()
```

Типовые задачи для Spark Streaming

- Обогащение большого потока данных (миллионы сообщений) справочной информацией и записью в несколько синков

```
incomming_message_schema = ...

kafka_data = spark.readStream.format('kafka').option(…)….load() \
    .select(from_json(col("value").cast("string"), incomming_message_schema))

postgresql_data = spark.read.format('jdbc').option(…)….load()

result_df = kafka_data.join(postgresql_data, col("kafka_join_col") == col("pg_join_col")) \
    .select(...)

result_df.writeStream \
    .foreachBatch(foreach_batch_function) \
    .start() \
    .awaitTermination()
```

Типовые задачи для Spark Streaming

- Обогащение большого потока данных (миллионы сообщений) справочной информацией и записью в несколько синков

```
def foreach_batch_function(df, epoch_id):
    df.persist()

    df.write \
        .mode("append") \
        .format("jdbc").option(…)….save()

    kafka_df = transform(df)

    kafka_df.write \
        .format("kafka").option(…)….save()

    df.unpersist()
```

Типовые задачи для Spark Streaming

- Обогащение большого потока данных (миллионы сообщений) справочной информацией и записью в несколько синков

```
def foreach_batch_function(df, epoch_id):
    df.persist()

    df.write \
        .mode("append") \
        .format("jdbc").option(…)….save()

    kafka_df = transform(df)

    kafka_df.write \
        .format("kafka").option(…)….save()

    df.unpersist()
```

Типовые задачи для Spark Streaming

- Обогащение большого потока данных (миллионы сообщений) справочной информацией и записью в несколько синков

```
def foreach_batch_function(df, epoch_id):
    df.persist()

    df.write \
        .mode("append") \
        .format("jdbc").option(…)….save()

    kafka_df = transform(df)

    kafka_df.write \
        .format("kafka").option(…)….save()

    df.unpersist()
```

Типовые задачи для Spark Streaming

- Обогащение большого потока данных (миллионы сообщений) справочной информацией и записью в несколько синков

```
def foreach_batch_function(df, epoch_id):
    df.persist()

    df.write \
        .mode("append") \
        .format("jdbc").option(…)….save()

    kafka_df = transform(df)

    kafka_df.write \
        .format("kafka").option(…)….save()

    df.unpersist()
```

Типовые задачи для Spark Streaming

- Обогащение большого потока данных (миллионы сообщений) справочной информацией и записью в несколько синков

```
def foreach_batch_function(df, epoch_id):
    df.persist()

    df.write \
        .mode("append") \
        .format("jdbc").option(…)….save()

    kafka_df = transform(df)

    kafka_df.write \
        .format("kafka").option(…)….save()

    df.unpersist()
```

Типовые задачи для Spark Streaming

- Большой поток данных (миллионы сообщений) из Kafka с записью на HDFS или S3
- Большой поток данных (миллионы сообщений) с дедубликацией без внешнего хранилища
- Обогащение большого потока данных (миллионы сообщений) справочной информацией и записью в несколько синков

Типовые задачи для Spark Streaming

- Большой поток данных (миллионы сообщений) из Kafka с записью на HDFS или S3
- Большой поток данных (миллионы сообщений) с дедубликацией без внешнего хранилища
- Обогащение большого потока данных (миллионы сообщений) справочной информацией и записью в несколько синков
- Объединение нескольких потоков данных в один (streams join)

Типовые задачи для Spark Streaming

- Большой поток данных (миллионы сообщений) из Kafka с записью на HDFS или S3
- Большой поток данных (миллионы сообщений) с дедубликацией без внешнего хранилища
- Обогащение большого потока данных (миллионы сообщений) справочной информацией и записью в несколько синков
- Объединение нескольких потоков данных в один (streams join)
- Создание Staging или Raw-слоя для DWH

Типовые задачи для Spark Streaming

- Большой поток данных (миллионы сообщений) из Kafka с записью на HDFS или S3
- Большой поток данных (миллионы сообщений) с дедубликацией без внешнего хранилища
- Обогащение большого потока данных (миллионы сообщений) справочной информацией и записью в несколько синков
- Объединение нескольких потоков данных в один (streams join)
- Создание Staging или Raw-слоя для DWH
- Lambda – архитектура (streaming + batch в одной кодовой базе)

Задачи не для Spark Streaming

Streaming tasks

Spark Structured Streaming

Задачи не для Spark Streaming

- Поток данных с минимальным latency (миллисекунды)

Задачи не для Spark Streaming

- Поток данных с минимальным latency (миллисекунды)

Flink

Задачи не для Spark Streaming

- Поток данных с минимальным latency (миллисекунды)

```
source = KafkaSource.builder() \
    .set_bootstrap_servers(brokers) \
    .set_topics("input-topic") \
    .set_group_id("my-group") \
    .set_starting_offsets(KafkaOffsetsInitializer.earliest()) \
    .set_value_only_deserializer(SimpleStringSchema()) \
    .build()

sink = KafkaSink.builder() \
    .set_bootstrap_servers(brokers) \
    .set_record_serializer(
        KafkaRecordSerializationSchema.builder()
            .set_topic("topic-name")
            .set_value_serialization_schema(SimpleStringSchema())
            .build()
    ) \
    .set_delivery_guarantee(DeliveryGuarantee.EXACTLY_ONCE) \
    .build()
```

Задачи не для Spark Streaming

- Поток данных с минимальным latency (миллисекунды)

```
source = KafkaSource.builder() \
    .set_bootstrap_servers(brokers) \
    .set_topics("input-topic") \
    .set_group_id("my-group") \
    .set_starting_offsets(KafkaOffsetsInitializer.earliest()) \
    .set_value_only_deserializer(SimpleStringSchema()) \
    .build()

sink = KafkaSink.builder() \
    .set_bootstrap_servers(brokers) \
    .set_record_serializer(
        KafkaRecordSerializationSchema.builder()
            .set_topic("topic-name")
            .set_value_serialization_schema(SimpleStringSchema())
            .build()
    ) \
    .set_delivery_guarantee(DeliveryGuarantee.EXACTLY_ONCE) \
    .build()
```

Задачи не для Spark Streaming

- Поток данных с минимальным latency (миллисекунды)

```
source = KafkaSource.builder() \
    .set_bootstrap_servers(brokers) \
    .set_topics("input-topic") \
    .set_group_id("my-group") \
    .set_starting_offsets(KafkaOffsetsInitializer.earliest()) \
    .set_value_only_deserializer(SimpleStringSchema()) \
    .build()

sink = KafkaSink.builder() \
    .set_bootstrap_servers(brokers) \
    .set_record_serializer(
        KafkaRecordSerializationSchema.builder()
            .set_topic("topic-name")
            .set_value_serialization_schema(SimpleStringSchema())
            .build()
    ) \
    .set_delivery_guarantee(DeliveryGuarantee.EXACTLY_ONCE) \
    .build()
```

Задачи не для Spark Streaming

- Поток данных с минимальным latency (миллисекунды)

```
source = KafkaSource.builder() \
    .set_bootstrap_servers(brokers) \
    .set_topics("input-topic") \
    .set_group_id("my-group") \
    .set_starting_offsets(KafkaOffsetsInitializer.earliest()) \
    .set_value_only_deserializer(SimpleStringSchema()) \
    .build()

sink = KafkaSink.builder() \
    .set_bootstrap_servers(brokers) \
    .set_record_serializer(
        KafkaRecordSerializationSchema.builder()
            .set_topic("topic-name")
            .set_value_serialization_schema(SimpleStringSchema())
            .build()
    ) \
    .set_delivery_guarantee(DeliveryGuarantee.EXACTLY_ONCE) \
    .build()
```

Задачи не для Spark Streaming

- Поток данных с минимальным latency (миллисекунды)

```
source = env.from_source(source, WatermarkStrategy.no_watermarks(), "Kafka Source")

stream = source…

stream.sink_to(sink)
```

Задачи не для Spark Streaming

- Потоки данных до миллионов событий в секунду с несложной трансформацией или с частичной агрегацией

Задачи не для Spark Streaming

- Потоки данных до миллионов событий в секунду с несложной трансформацией или с частичной агрегацией

Задачи не для Spark Streaming

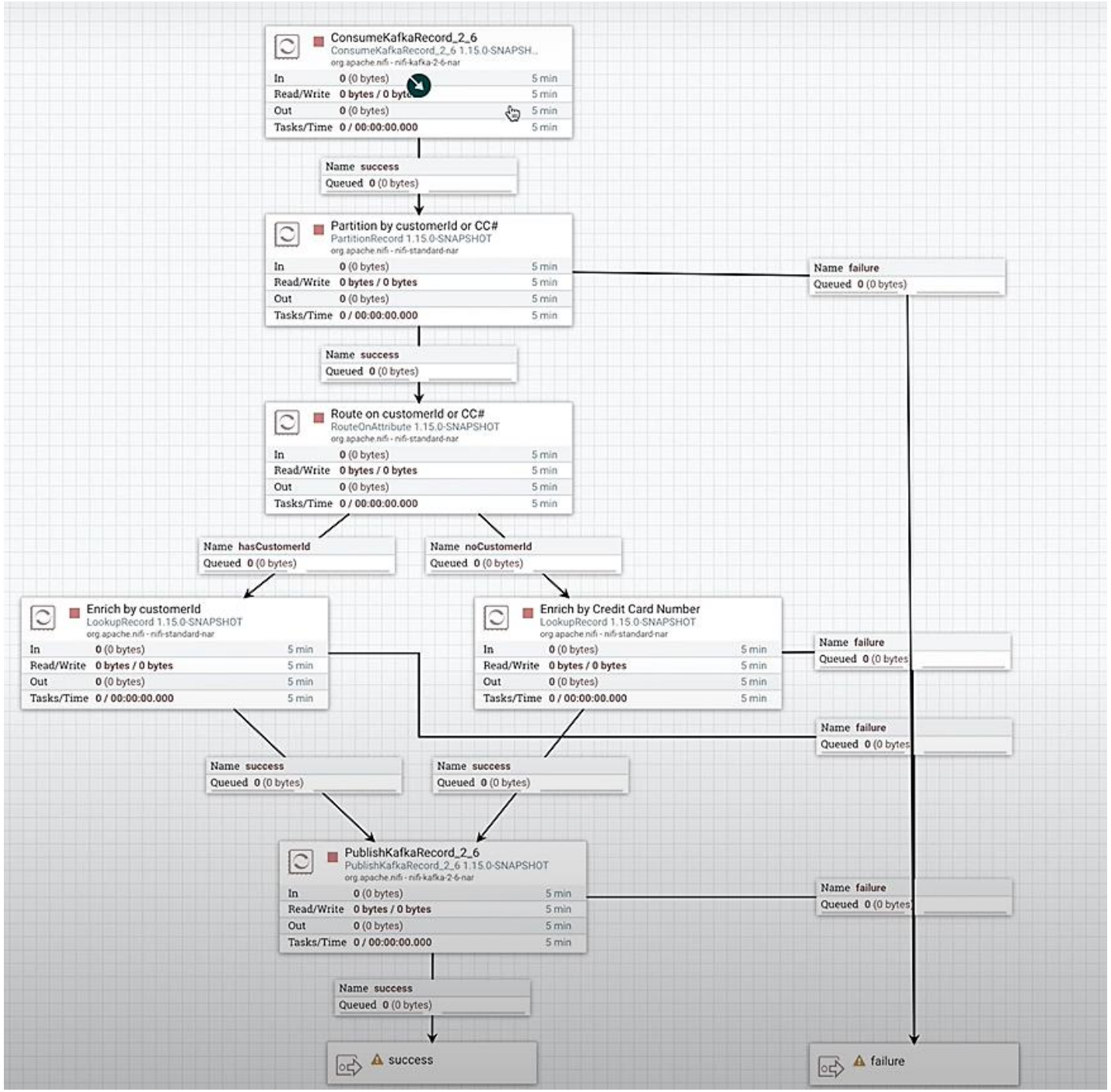
- Потоки данных до миллионов событий в секунду с несложной трансформацией или с частичной агрегацией

Доклад:
Дмитрий Бугайченко — Stateful streaming: Кейсы, паттерны, реализации

Задачи не для Spark Streaming

- Потоки данных до миллионов событий в секунду с несложной трансформацией или с частичной агрегацией

@nifiusers


ConsumeKafkaRecord_2_6
ConsumeKafkaRecord_2_6 1.15.0-SNAPSH...
org.apache.nifi - nifi-kafka-2-6-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 byte 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
Name success
Queued 0 (0 bytes)
Partition by customerId or CC#
PartitionRecord 1.15.0-SNAPSHOT
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
Name failure
Queued 0 (0 bytes)
Name success
Queued 0 (0 bytes)
Route on customerId or CC#
RouteOnAttribute 1.15.0-SNAPSHOT
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
Name hasCustomerId
Queued 0 (0 bytes)
Name noCustomerId
Queued 0 (0 bytes)
Enrich by customerId
LookupRecord 1.15.0-SNAPSHOT
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
Enrich by Credit Card Number
LookupRecord 1.15.0-SNAPSHOT
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
Name failure
Queued 0 (0 bytes)
Name failure
Queued 0 (0 bytes)
Name success
Queued 0 (0 bytes)
Name success
Queued 0 (0 bytes)
PublishKafkaRecord_2_6
PublishKafkaRecord_2_6 1.15.0-SNAPSHOT
org.apache.nifi - nifi-kafka-2-6-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
Name failure
Queued 0 (0 bytes)
Name success
Queued 0 (0 bytes)
success
failure

Задачи не для Spark Streaming

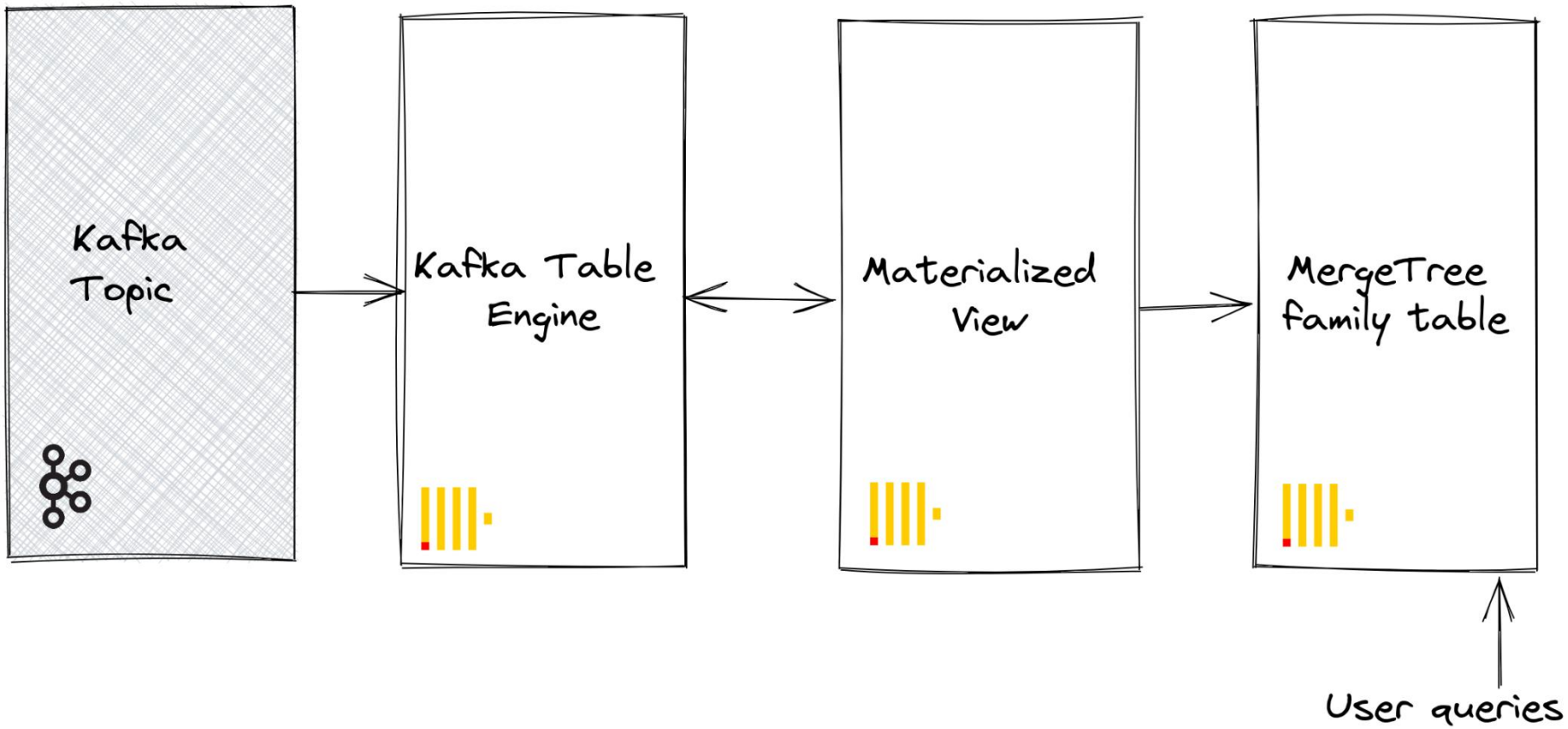
- Потоки данных с пост-анализом – проверка гипотез

Задачи не для Spark Streaming

- Потоки данных с пост-анализом – проверка гипотез

Задачи не для Spark Streaming

- Потоки данных с пост-анализом – проверка гипотез



Задачи не для Spark Streaming

- Контроль на каждом шаге и строгость выполнения операций

Задачи не для Spark Streaming

- Контроль на каждом шаге и строгость выполнения операций (применение функторов map, flatMap и т.п.)

Чек-лист по применению Spark Streaming

Чек-лист по применению Spark Streaming

✓Большие потоки данных (от миллионов событий в секунду)

Чек-лист по применению Spark Streaming

✓Большие потоки данных (от миллионов событий в секунду)

✓Отсутствует необходимость семантики exactly-once

Чек-лист по применению Spark Streaming

- ✓Большие потоки данных (от миллионов событий в секунду)
- ✓Отсутствует необходимость семантики exactly-once
- ✓Не нужно глубокое погружение в обработку данных (функторы)

Чек-лист по применению Spark Streaming

- ✓Большие потоки данных (от миллионов событий в секунду)
- ✓Отсутствует необходимость семантики exactly-once
- ✓Не нужно глубокое погружение в обработку данных (функторы)
- ✓Есть компетенции для поддержки инфраструктуры Spark Streaming

Чек-лист по применению Spark Streaming

- ✓Большие потоки данных (от миллионов событий в секунду)
- ✓Отсутствует необходимость семантики exactly-once
- ✓Не нужно глубокое погружение в обработку данных (функторы)
- ✓Есть компетенции для поддержки инфраструктуры Spark Streaming
- ✓Интенсивность потока не меняется скачкообразно за короткий промежуток времени

Чек-лист по применению Spark Streaming

- ✓Большие потоки данных (от миллионов событий в секунду)
- ✓Отсутствует необходимость семантики exactly-once
- ✓Не нужно глубокое погружение в обработку данных (функторы)
- ✓Есть компетенции для поддержки инфраструктуры Spark Streaming
- ✓Интенсивность потока не меняется скачкообразно за короткий промежуток времени
- ✓Нужно джойнить большие потоки данных с рассинхронном по времени или статику

Сообщества

- Moscow Spark
- Powerful NiFi
- ClickHouse не тормозит
- Инжиниринг данных

Евгений Ненахов

Tech Lead – центр BigData МТС Digital, к.ф.-м.н

@neltari