Потоковая обработка с Kafka в условиях Big Data

Евгений Ненахов

MTC DIGITAL

Евгений Ненахов

- TechLead – Центр Big Data MTC Digital
- К.ф.-м.н, доцент кафедры «Вычислительная математика и программирование» МАИ
- Один из авторов образовательного курса для «Data Engineer» Яндекс Практикум
- Спикер Joker, JPoint, SmartData, Highload++

Что вас ждёт?

- Покажу проблемы, с которыми вы можете столкнуться

Что вас ждёт?

- Покажу проблемы, с которыми вы можете столкнуться
  - Производительность, когда не хватает параллельности за счёт партиций
  - Утилизация ресурсов при равномерном распределении партиций

Что вас ждёт?

- Покажу проблемы, с которыми вы можете столкнуться
  - Производительность, когда не хватает параллельности за счёт партиций
  - Утилизация ресурсов при равномерном распределении партиций
- Покажу и разберу решения этих проблем

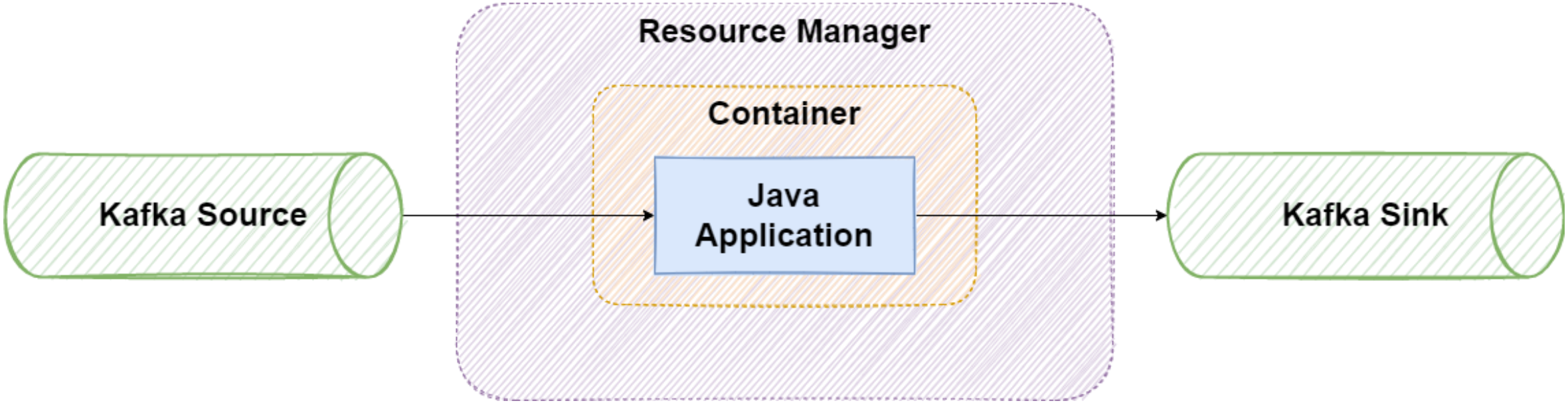
Что вас ждёт?

- Покажу проблемы, с которыми вы можете столкнуться
  - Производительность, когда не хватает параллельности за счёт партиций
  - Утилизация ресурсов при равномерном распределении партиций
- Покажу и разберу решения этих проблем
- Расскажу, что значат конкретные Exception от Consumer и Producer и как с ними бороться

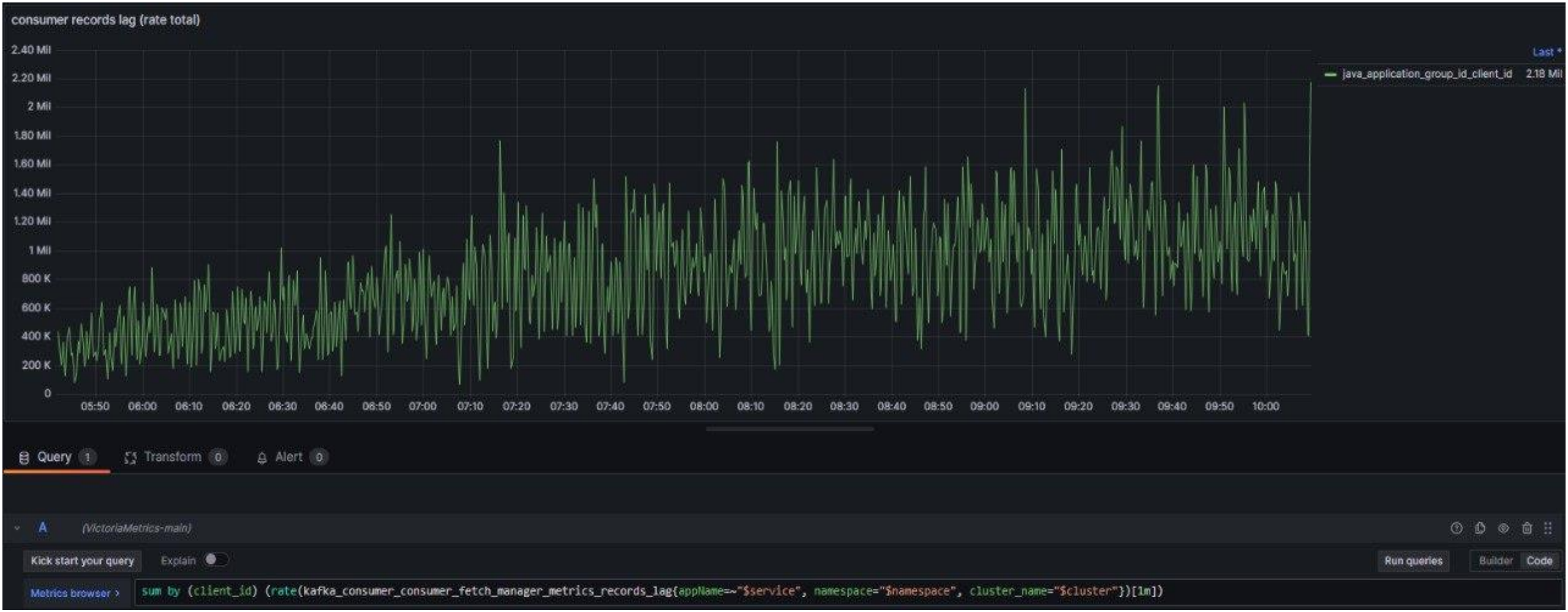
Классика Kafka to Kafka

Простая реализация

Запускаем в единственном экземпляре

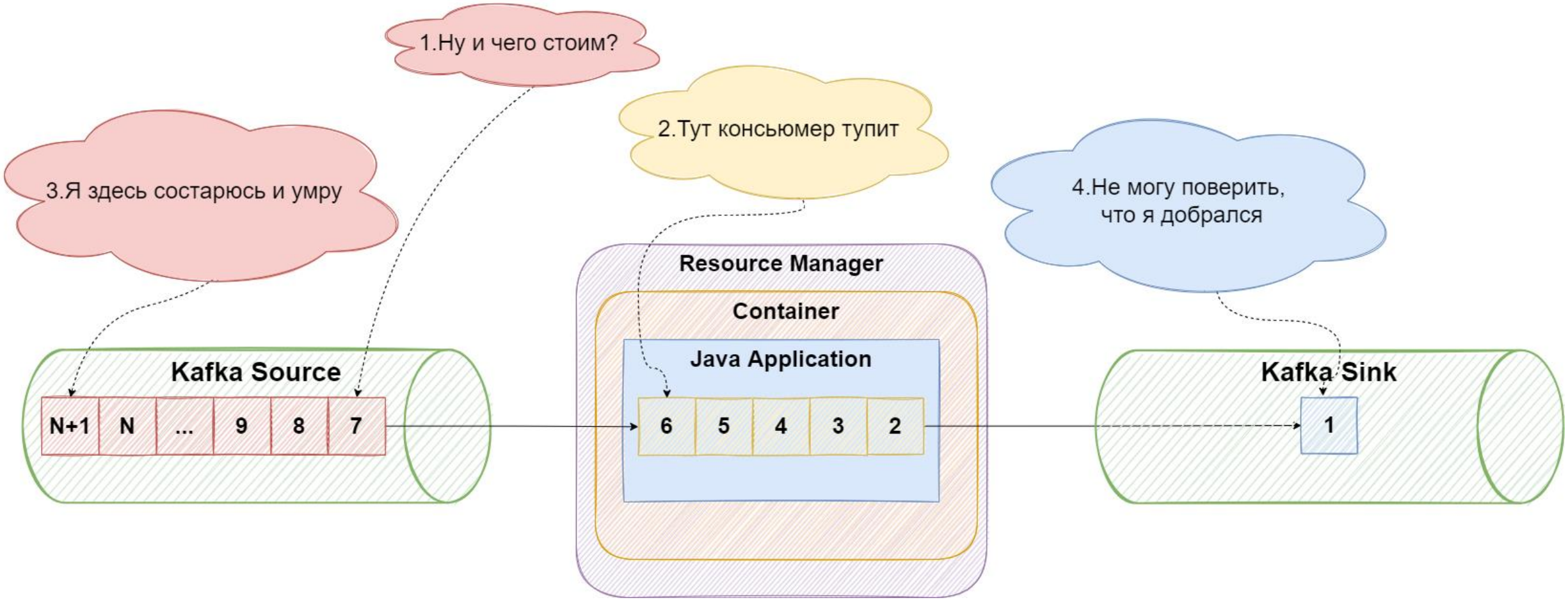


LAG по оффсетам у консьюмера

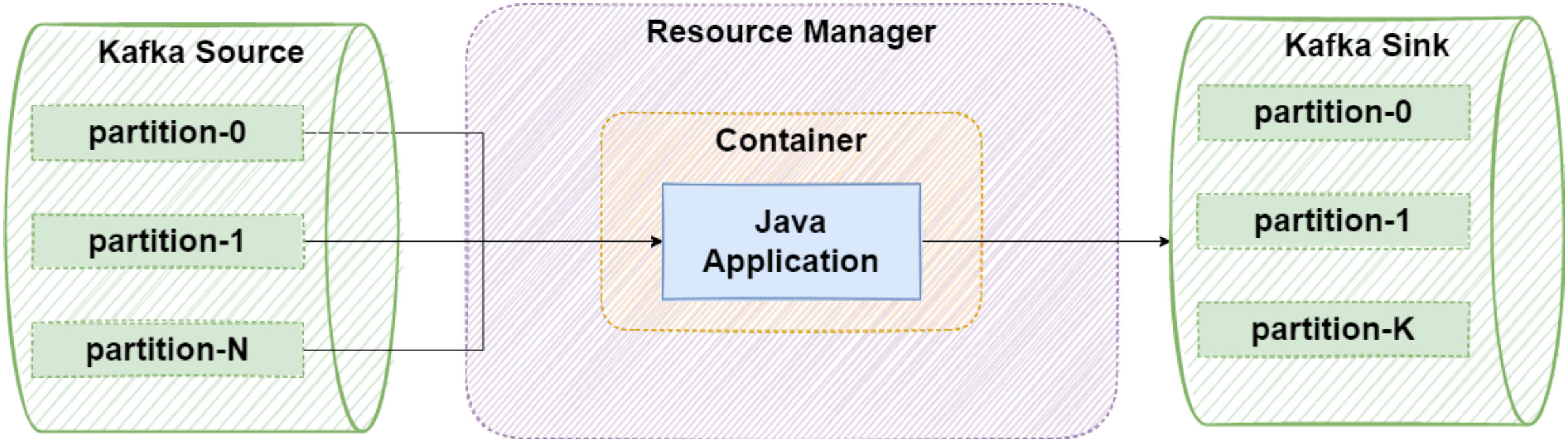


LAG по оффсетам у консьюмера

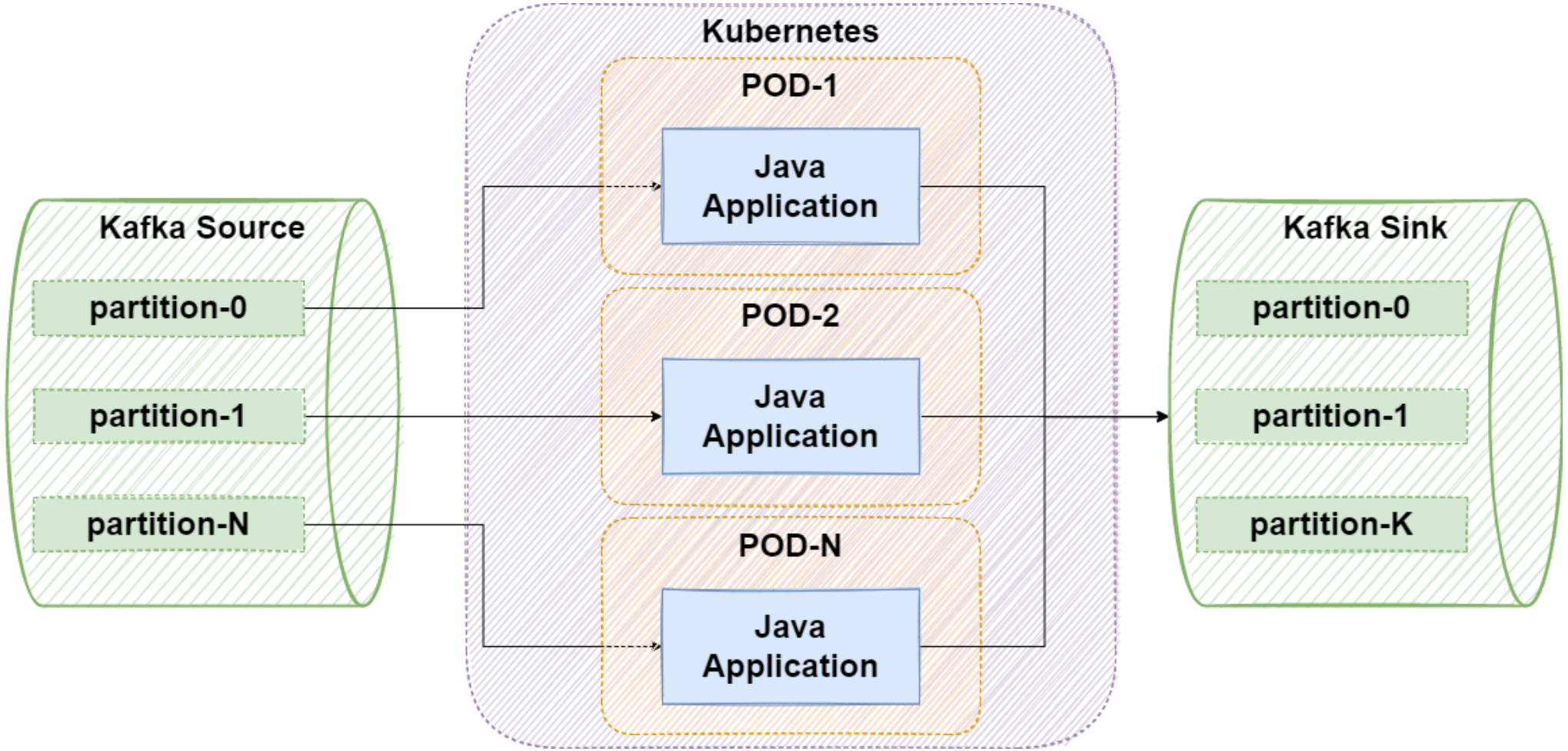
Консьюмер не успевает обрабатывать



Консьюмер читает все партиции

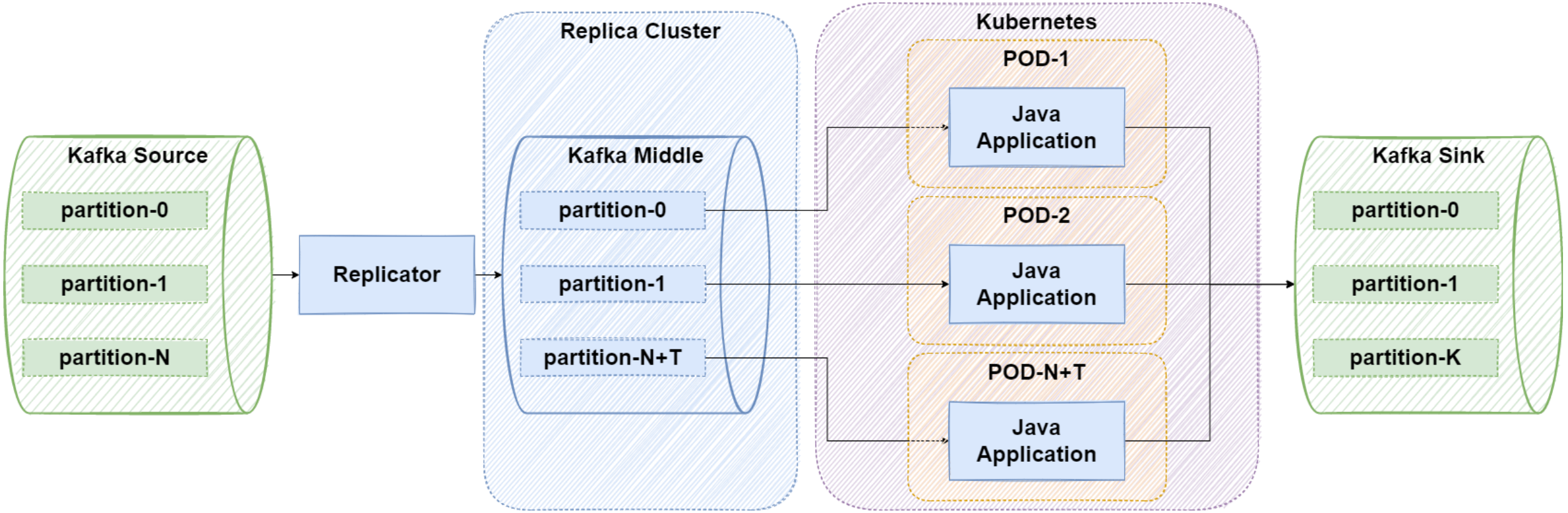


Повышаем пропускную способность

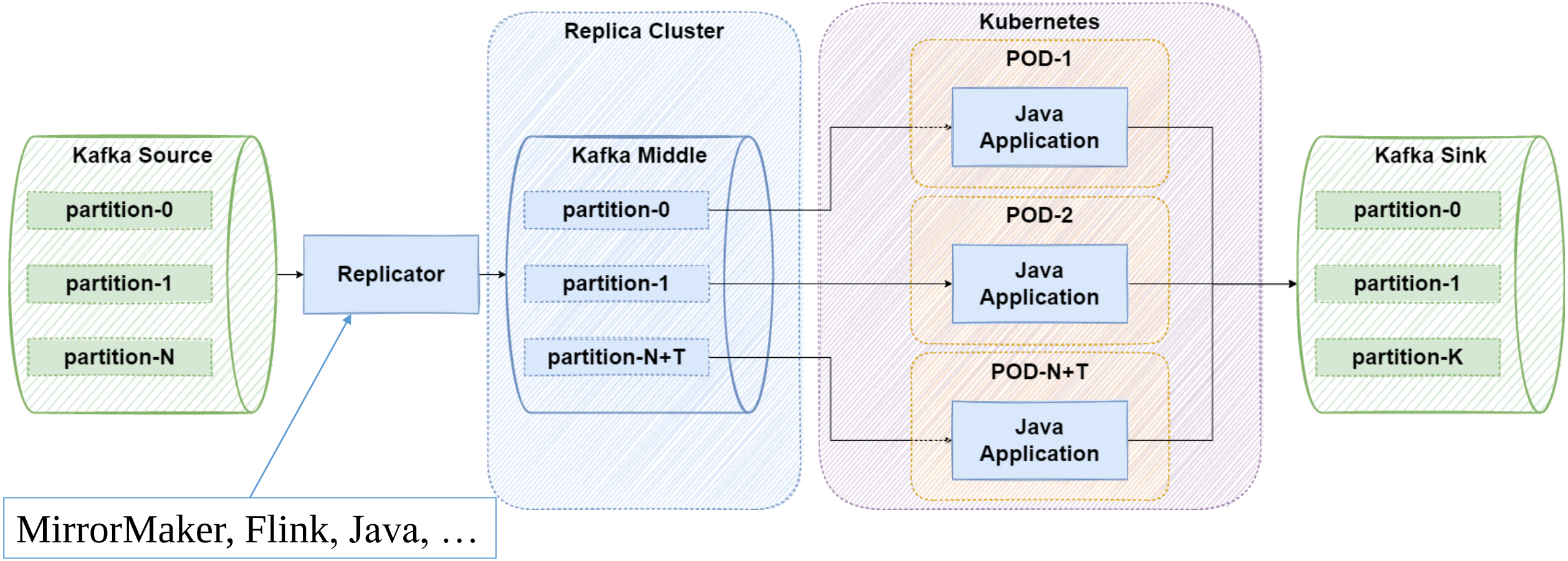


LAG всё равно растёт

Самое «простое» решение



Самое «простое» решение



Самое «простое» решение

Недостатки:

Самое «простое» решение

Недостатки:

- Дополнительные вычислительные ресурсы

Самое «простое» решение

Недостатки:

- Дополнительные вычислительные ресурсы
- Поддержка и эксплуатация собственного кластера (SLA)

Самое «простое» решение

Недостатки:

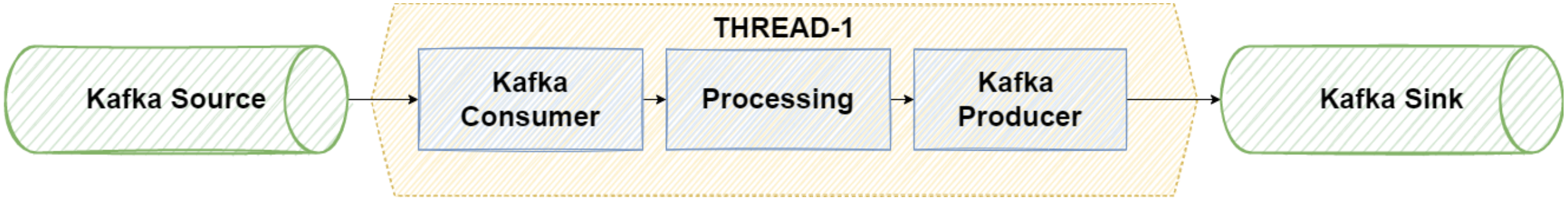
- Дополнительные вычислительные ресурсы
- Поддержка и эксплуатация собственного кластера (SLA)
- Дополнительные требования к хранению данных от системы источника

Самое «простое» решение

Недостатки:

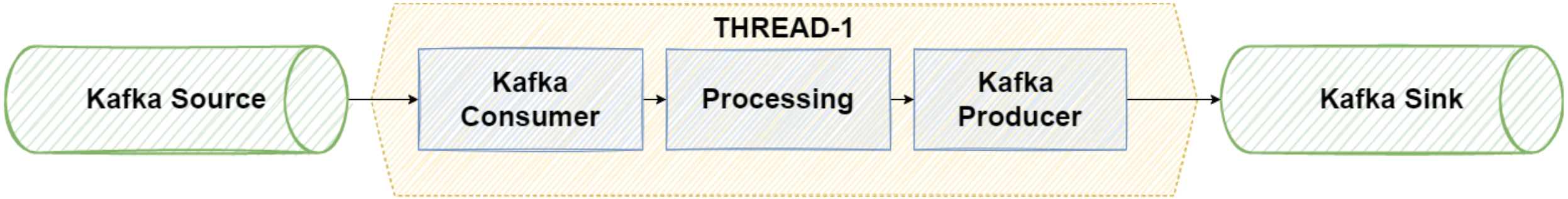
- Дополнительные вычислительные ресурсы
- Поддержка и эксплуатация собственного кластера (SLA)
- Дополнительные требования к хранению данных от системы источника
- Дополнительная точка отказа

Однопоточная обработка данных

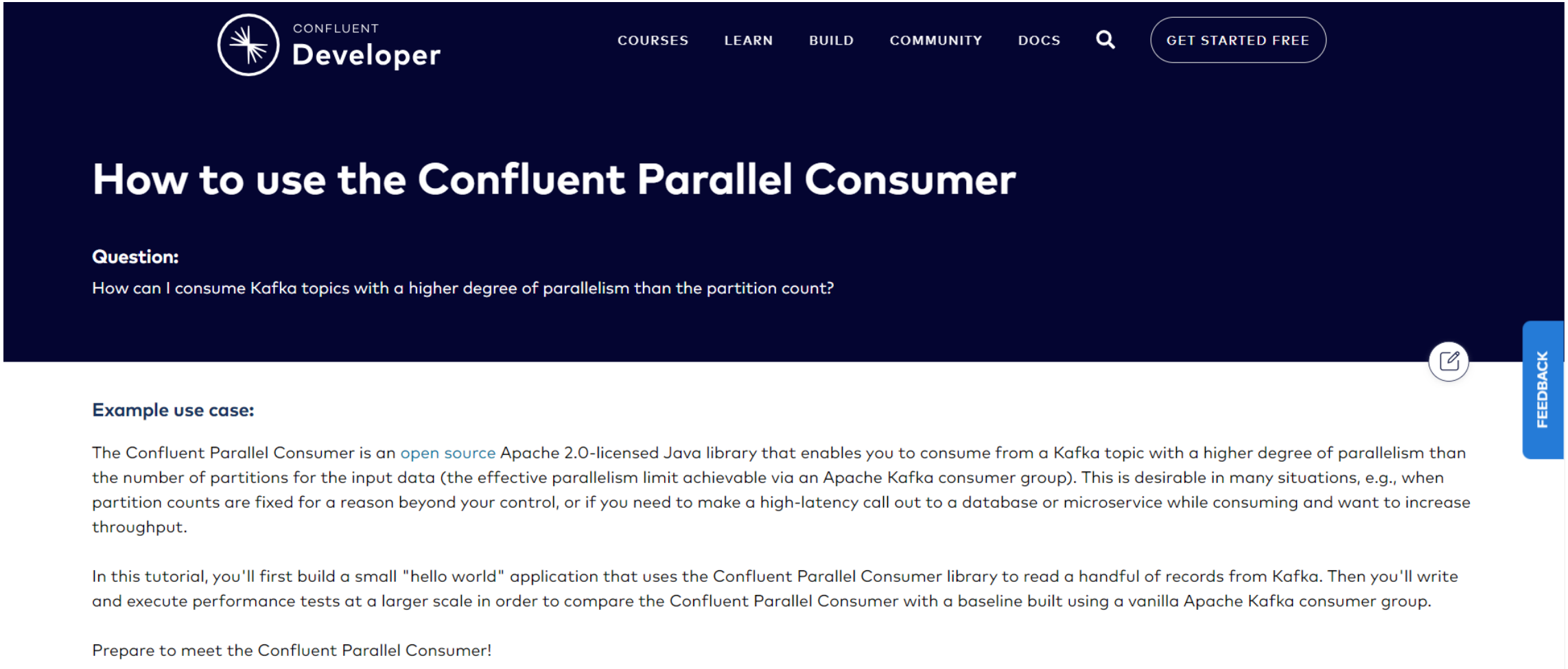


Однопоточная обработка данных

А можно ли распараллелить этапы обработки данных?



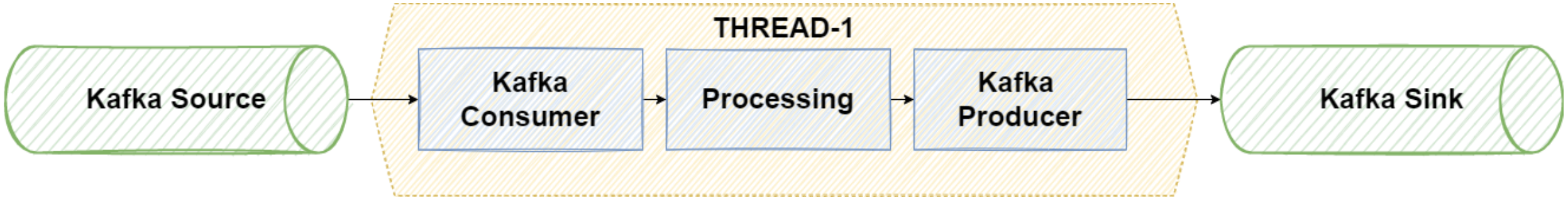
Confluent Parallel Consumer



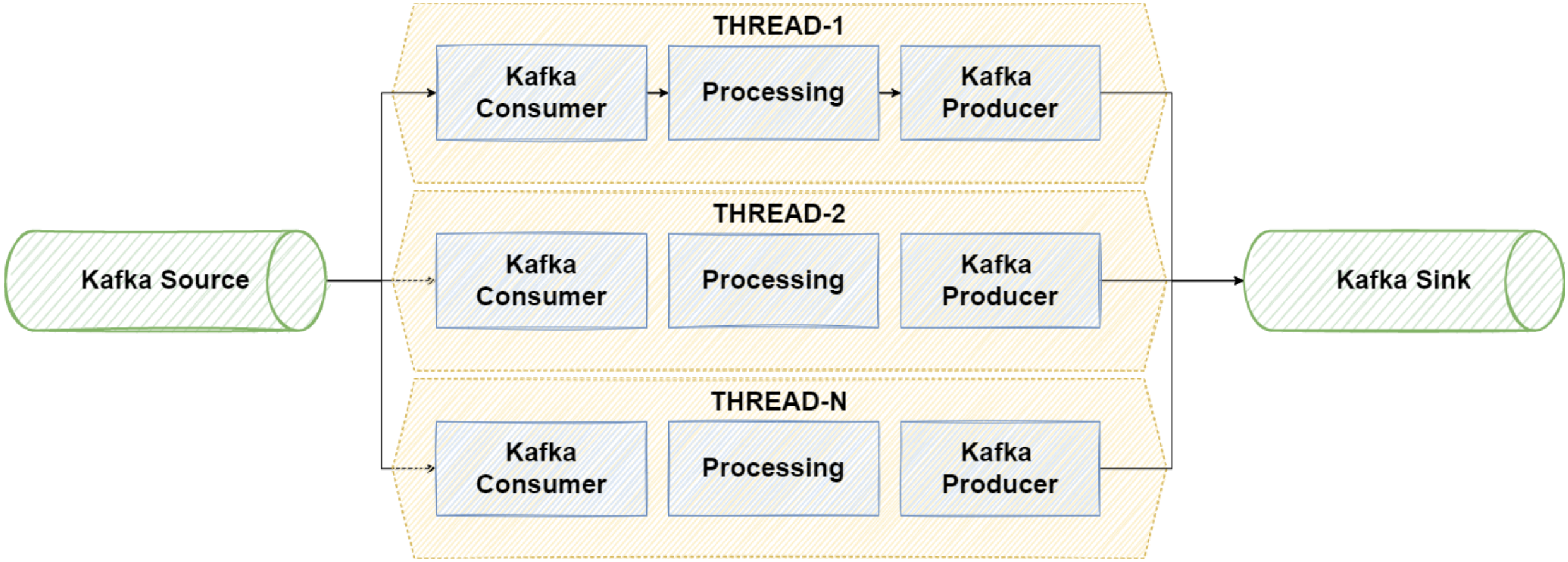
Confluent Parallel Consumer

Сайт с tutorial:
https://developer.confluent.io/tutorials/confluent-parallel-consumer/kafka.html

Репликация инстансов или потоков

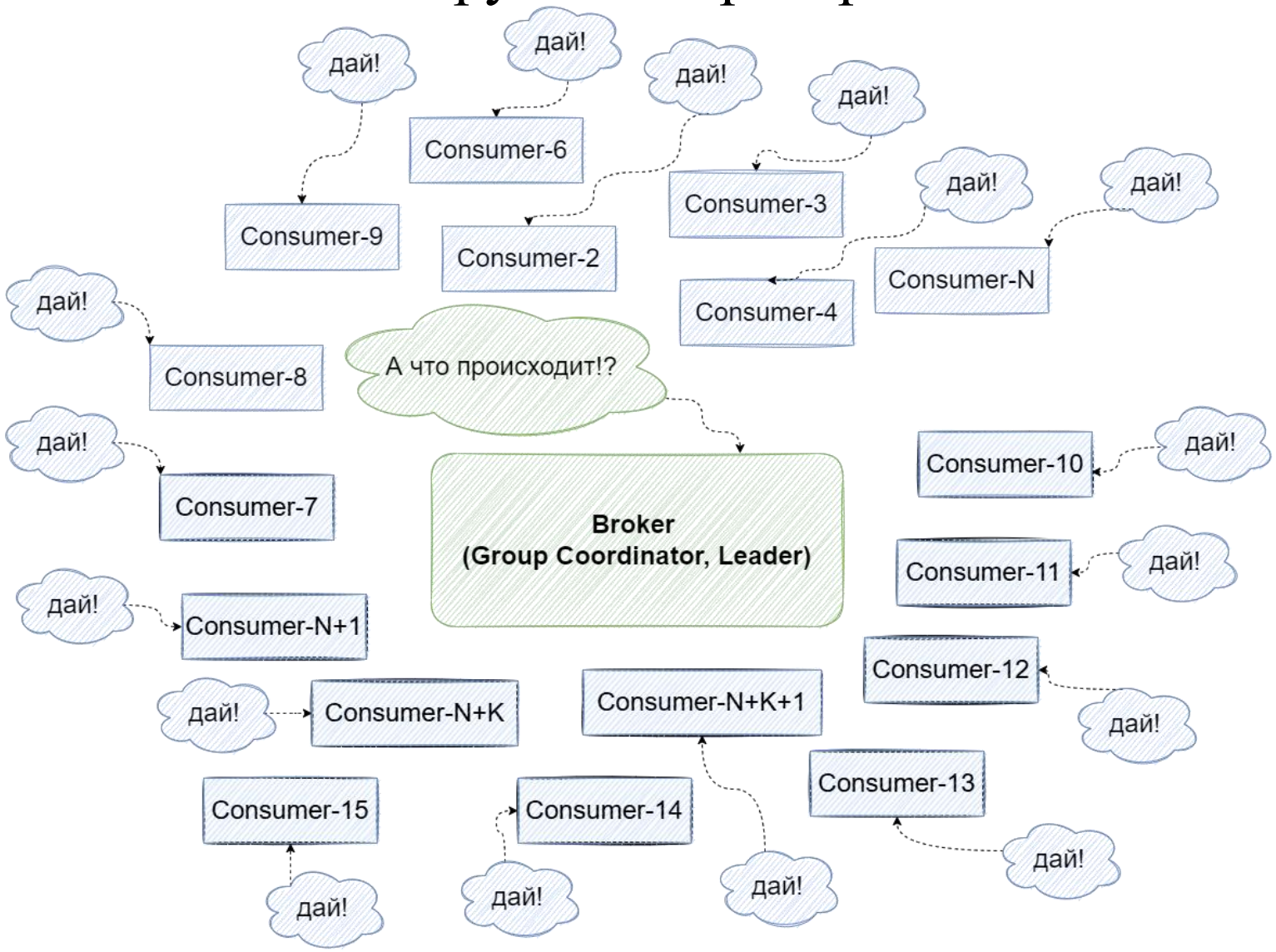


Репликация инстансов или потоков



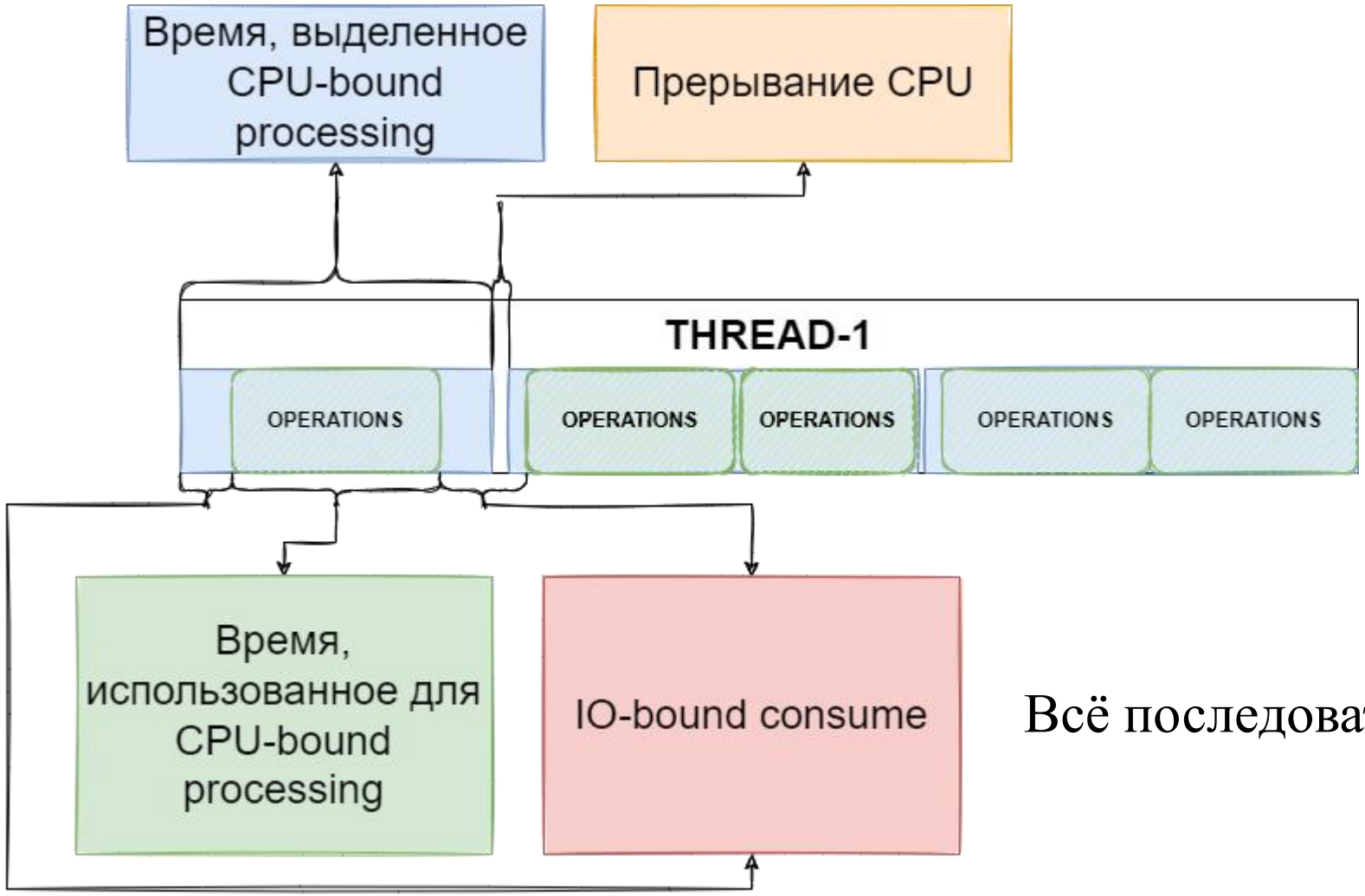
Недостатки параллелизации за счёт реплик

1. Дополнительная нагрузка на брокеры



Недостатки параллелизации за счёт реплик

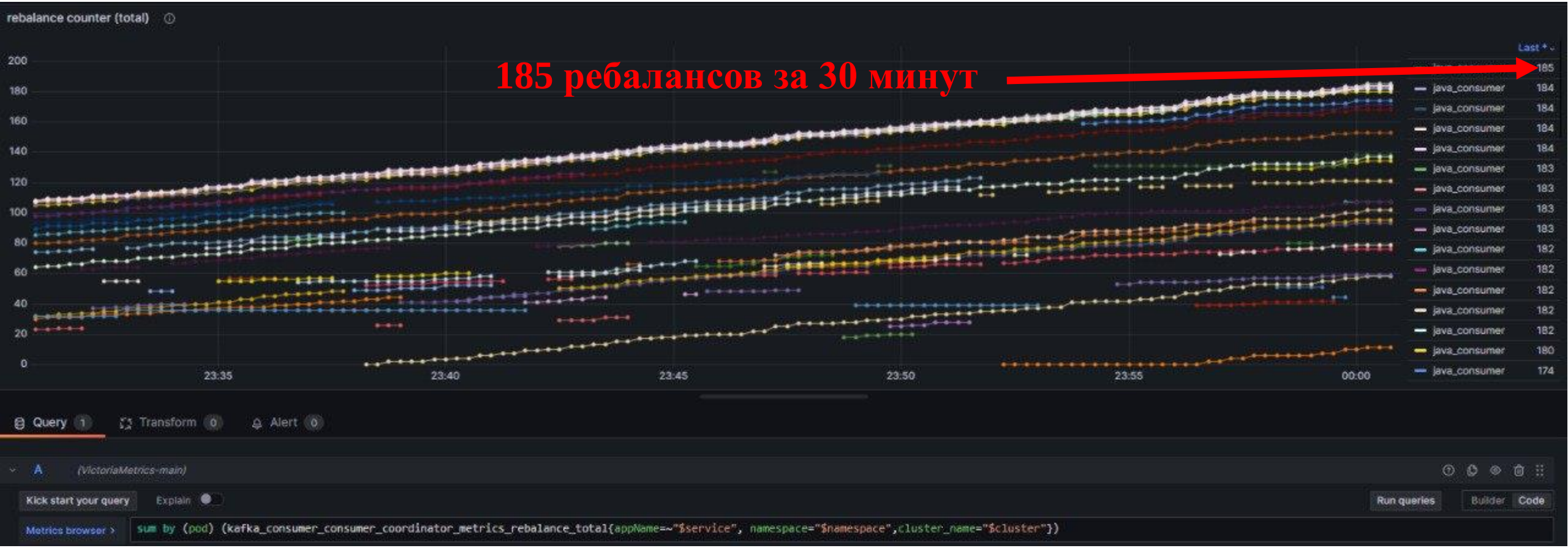
2. Неэффективное использование ресурса CPU



Всё последовательно в потоке

Недостатки параллелизации за счёт реплик

3. Высокая вероятность лавинного ребаланса консьюмеров (~185 за 30 мин)



Недостатки параллелизации за счёт реплик

3. Высокая вероятность лавинного ребаланса консьюмеров (< 1 min)

Недостатки параллелизации за счёт реплик

1. Дополнительная нагрузка на брокеры
2. Неэффективное использование ресурса CPU
3. Высокая вероятность лавинного ребаланса консьюмеров

Parallel Consumer

- Использует «под капотом» Vert.x

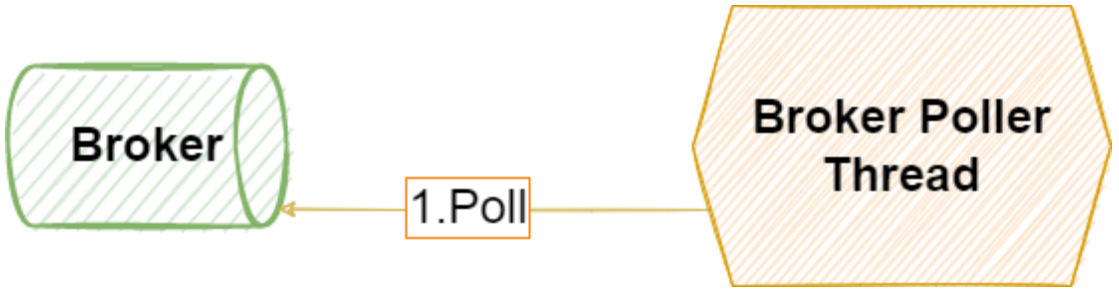
Владимир Красильщик —
Vert.x: руководство по эксплуатации

Parallel Consumer

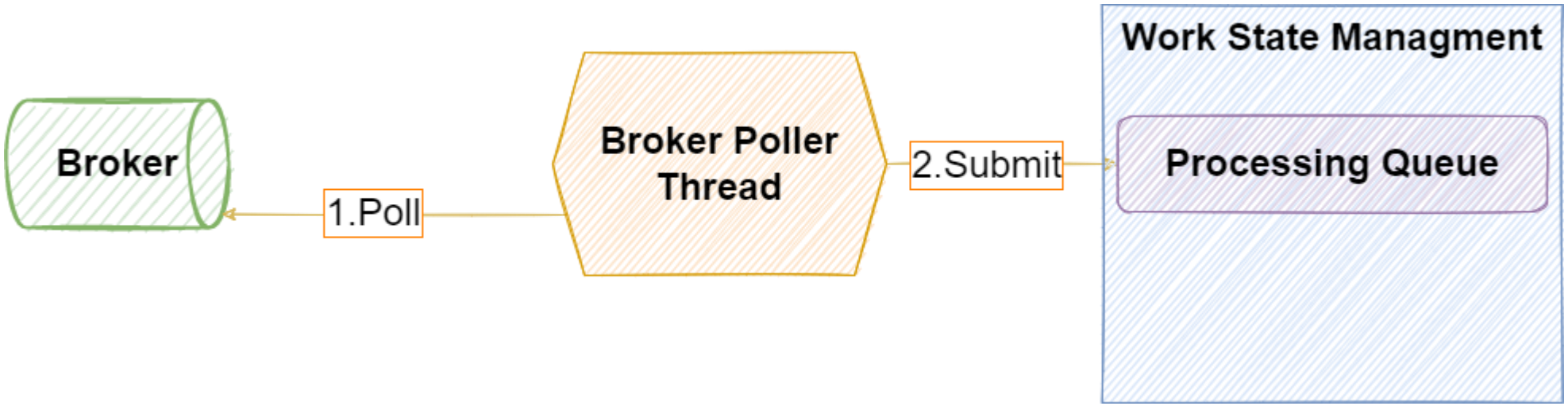
- Использует «под капотом» Vert.x

Даниил Солодухин —
Поваренная книга программиста: Vert.x

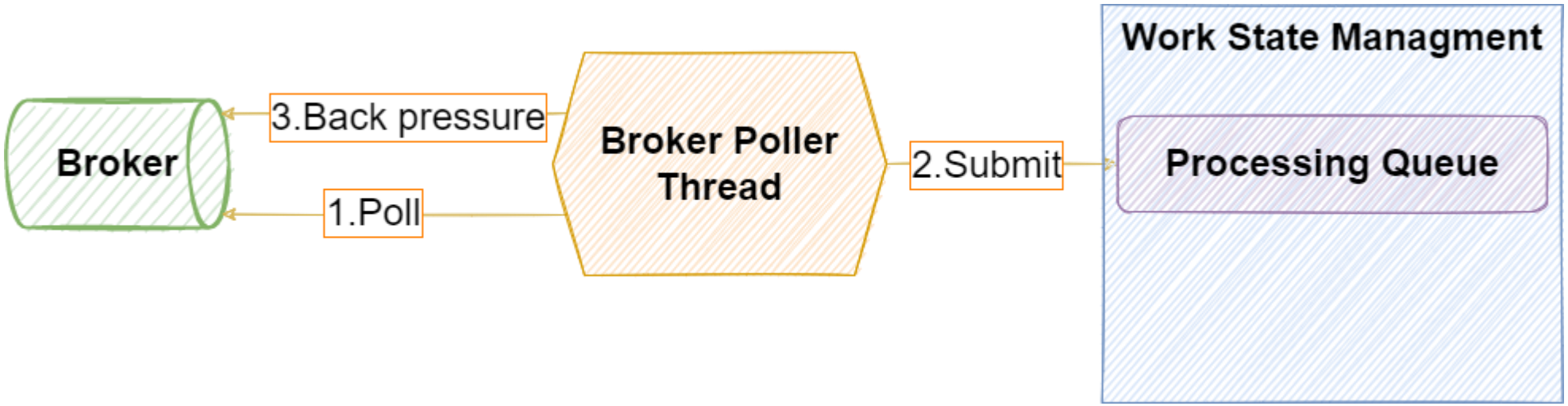
Архитектура Parallel Consumer



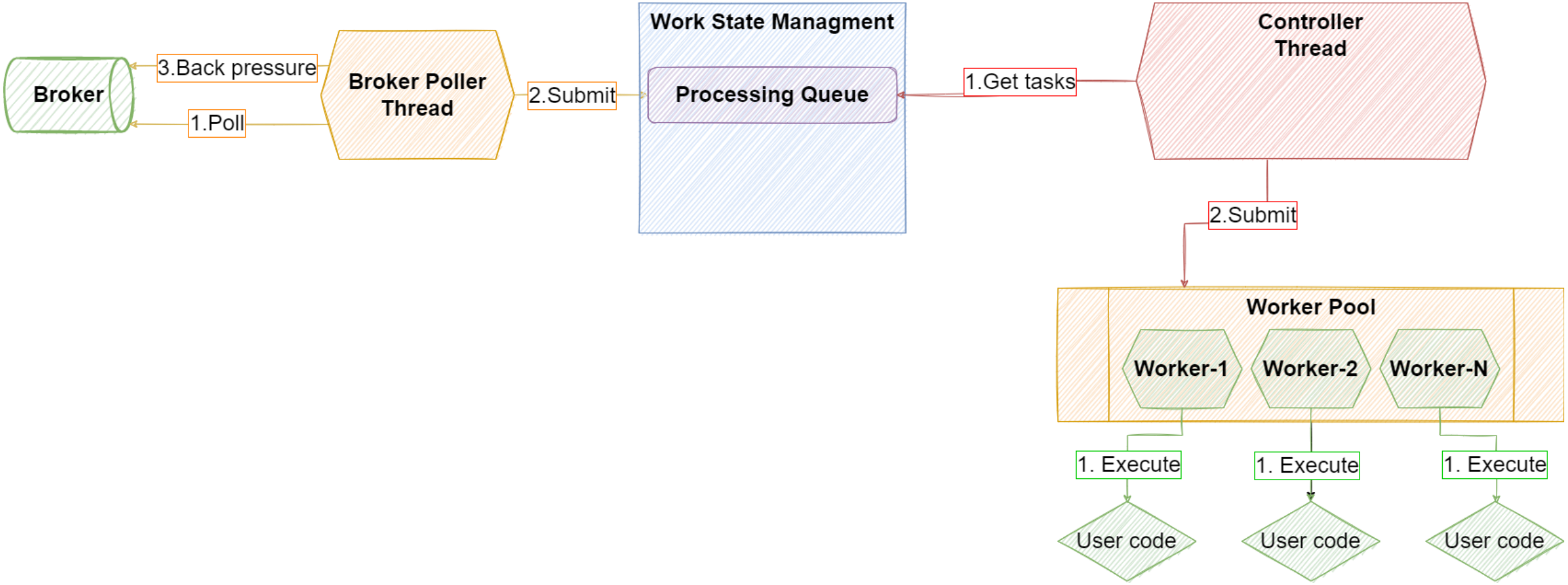
Архитектура Parallel Consumer



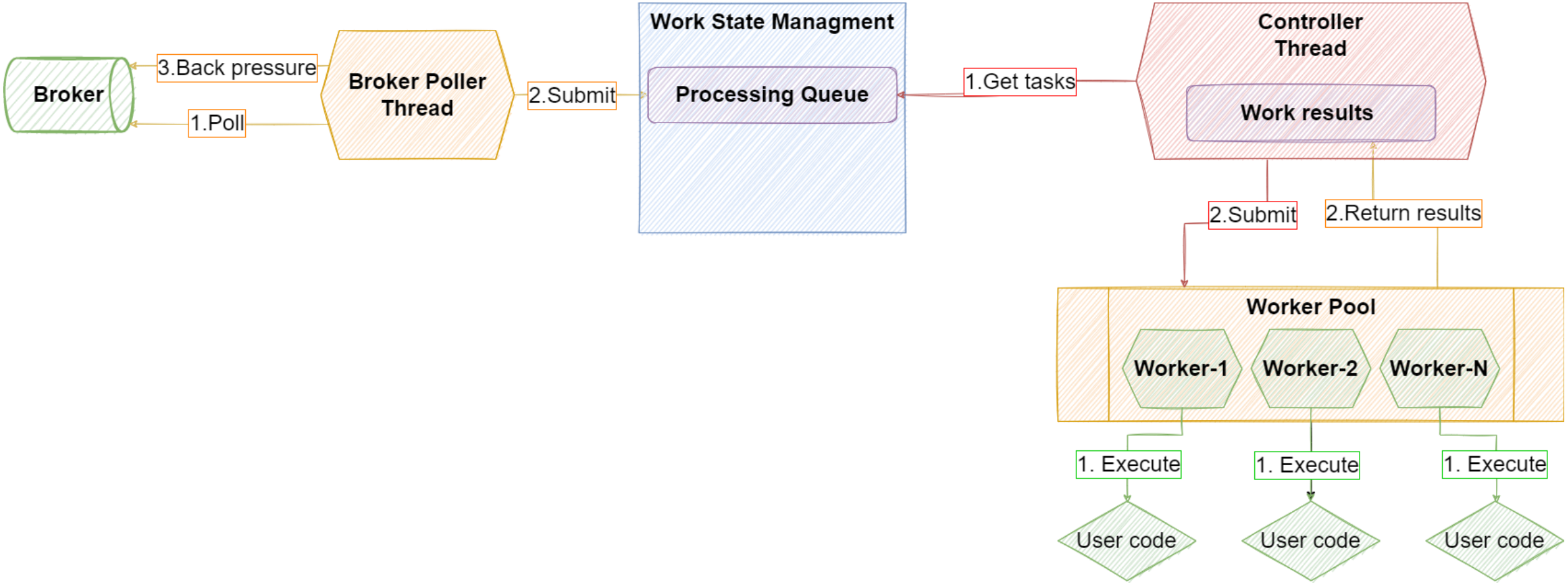
Архитектура Parallel Consumer



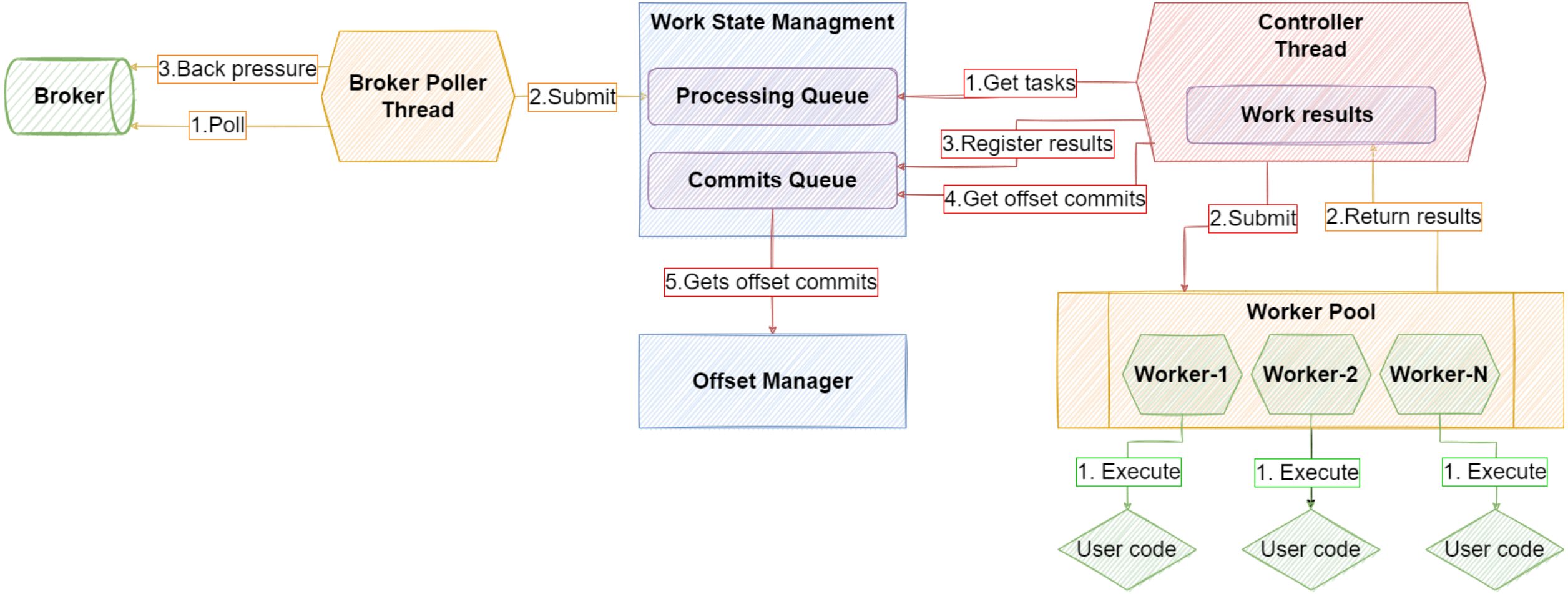
Архитектура Parallel Consumer



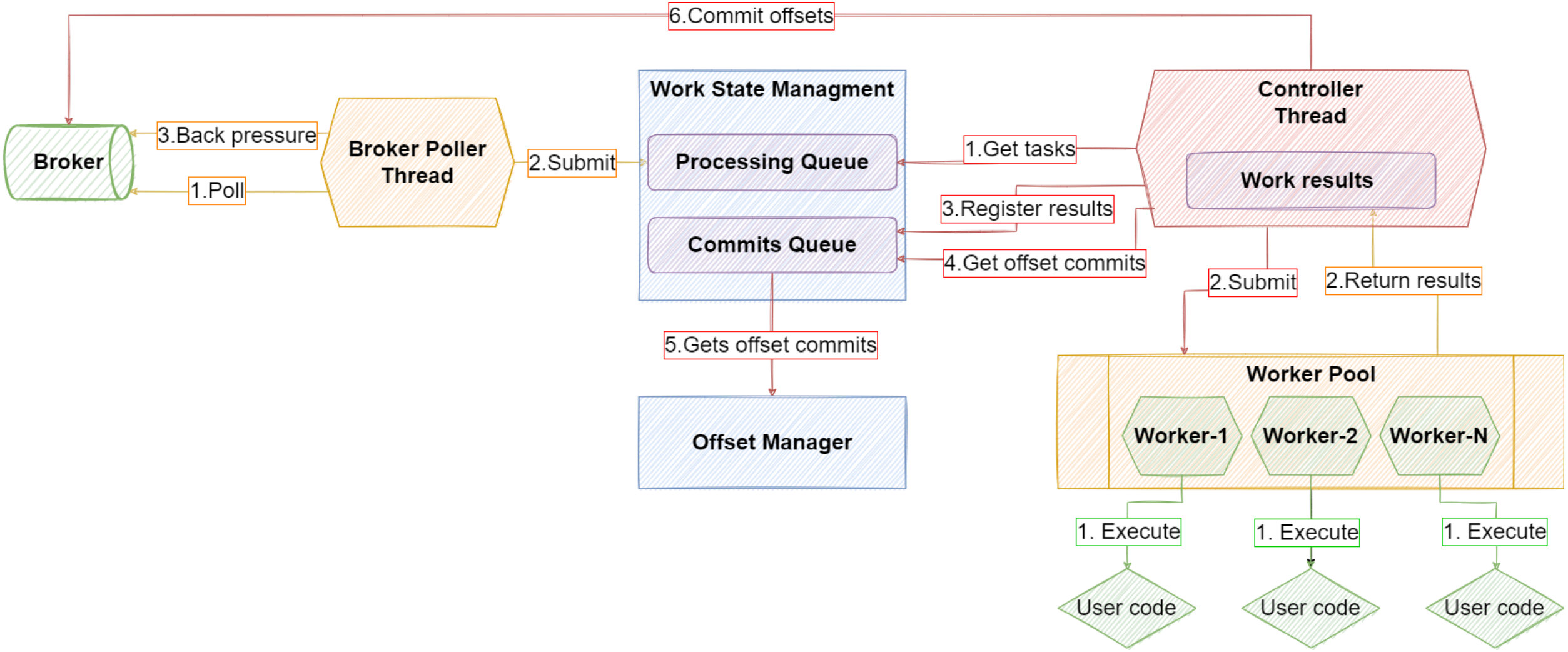
Архитектура Parallel Consumer



Архитектура Parallel Consumer



Архитектура Parallel Consumer



Пишем код с Parallel consumer

```
var kafkaConsumer = getKafkaConsumer();
var kafkaProducer = getKafkaProducer();
var options = ParallelConsumerOptions.<K, V>builder()
        .ordering(UNORDERED) // PARTITION, KEY
        .maxConcurrency(1000) // Default 16
        .batchSize(100) // Default 1
        .commitMode(PERIODIC_CONSUMER_ASYNCHRONOUS) //
PERIODIC_CONSUMER_SYNC, PERIODIC_TRANSACTIONAL_PRODUCER
        .consumer(kafkaConsumer)
        .producer(kafkaProducer)
        .build();
```

Пишем код с Parallel consumer

```
var kafkaConsumer = getKafkaConsumer();
var kafkaProducer = getKafkaProducer();
var options = ParallelConsumerOptions.<K, V>builder()
        .ordering(UNORDERED) // PARTITION, KEY
        .maxConcurrency(1000) // Default 16
        .batchSize(100) // Default 1
        .commitMode(PERIODIC_CONSUMER_ASYNCHRONOUS) //
PERIODIC_CONSUMER_SYNC, PERIODIC_TRANSACTIONAL_PRODUCER
        .consumer(kafkaConsumer)
        .producer(kafkaProducer)
        .build();
```

Пишем код с Parallel consumer

```
var kafkaConsumer = getKafkaConsumer();
var kafkaProducer = getKafkaProducer();
var options = ParallelConsumerOptions.<K, V>builder()
        .ordering(UNORDERED) // PARTITION, KEY
        .maxConcurrency(1000) // Default 16
        .batchSize(100) // Default 1
        .commitMode(PERIODIC_CONSUMER_ASYNCHRONOUS) //
PERIODIC_CONSUMER_SYNC, PERIODIC_TRANSACTIONAL_PRODUCER
        .consumer(kafkaConsumer)
        .producer(kafkaProducer)
        .build();
```

Пишем код с Parallel consumer

```
var kafkaConsumer = getKafkaConsumer();
var kafkaProducer = getKafkaProducer();
var options = ParallelConsumerOptions.<K, V>builder()
        .ordering(UNORDERED) // PARTITION, KEY
        .maxConcurrency(1000) // Default 16
        .batchSize(100) // Default 1
        .commitMode(PERIODIC_CONSUMER_ASYNCHRONOUS) //
PERIODIC_CONSUMER_SYNC, PERIODIC_TRANSACTIONAL_PRODUCER
        .consumer(kafkaConsumer)
        .producer(kafkaProducer)
        .build();
```

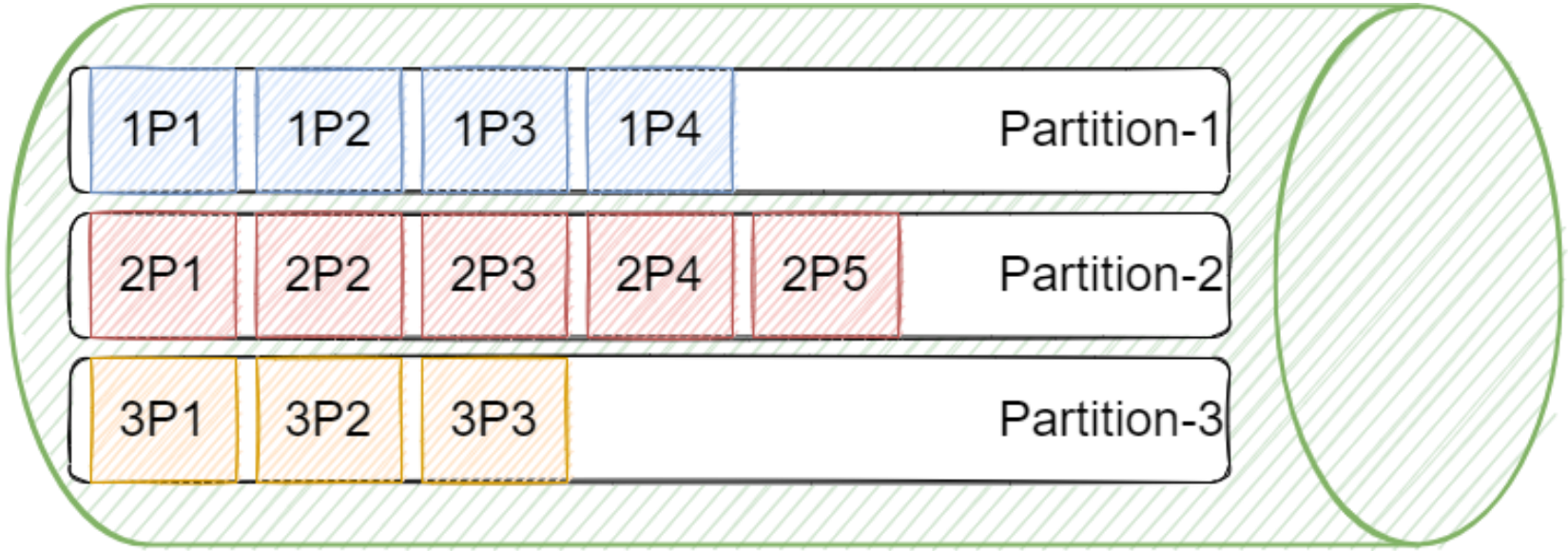
Пишем код с Parallel consumer

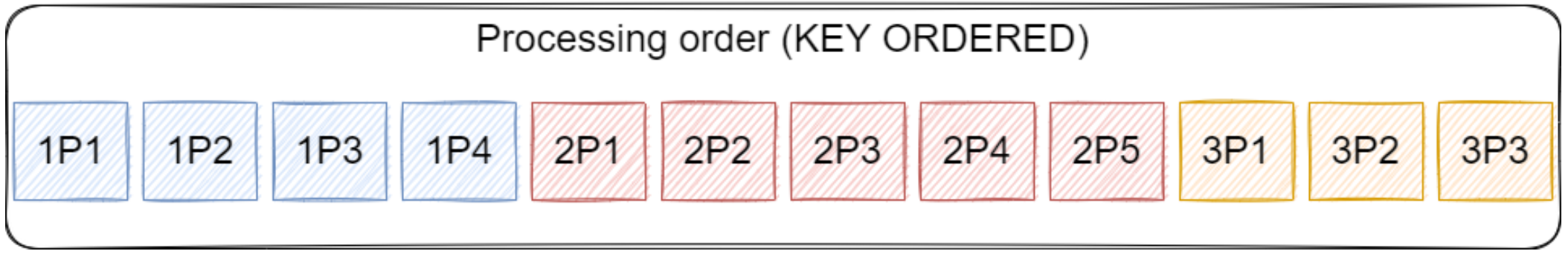
```
var kafkaConsumer = getKafkaConsumer();
var kafkaProducer = getKafkaProducer();
var options = ParallelConsumerOptions.<K, V>builder()
        .ordering(UNORDERED) // PARTITION, KEY
        .maxConcurrency(1000) // Default 16
        .batchSize(100) // Default 1
        .commitMode(PERIODIC_CONSUMER_ASYNCHRONOUS) //
PERIODIC_CONSUMER_SYNC, PERIODIC_TRANSACTIONAL_PRODUCER
        .consumer(kafkaConsumer)
        .producer(kafkaProducer)
        .build();
```

Пишем код с Parallel consumer

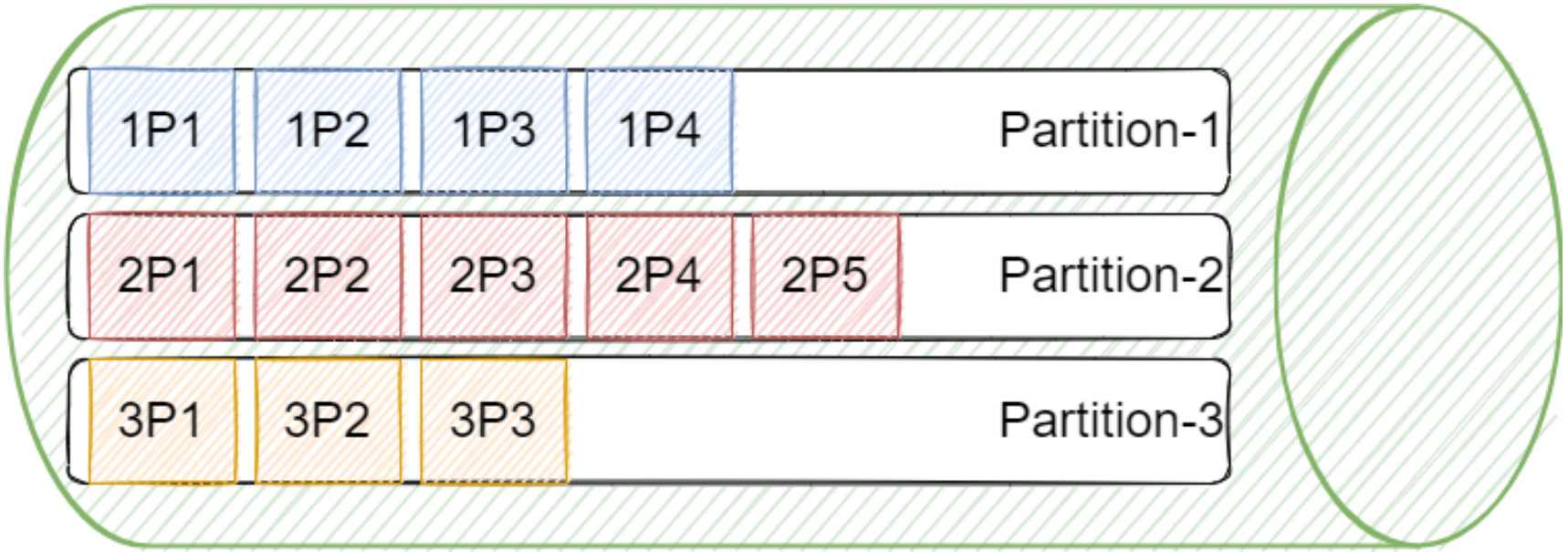
```
var kafkaConsumer = getKafkaConsumer();
var kafkaProducer = getKafkaProducer();
var options = ParallelConsumerOptions.<K, V>builder()
        .ordering(UNORDERED) // PARTITION, KEY
        .maxConcurrency(1000) // Default 16
        .batchSize(100) // Default 1
        .commitMode(PERIODIC_CONSUMER_ASYNCHRONOUS) //
PERIODIC_CONSUMER_SYNC, PERIODIC_TRANSACTIONAL_PRODUCER
        .consumer(kafkaConsumer)
        .producer(kafkaProducer)
        .build();
```

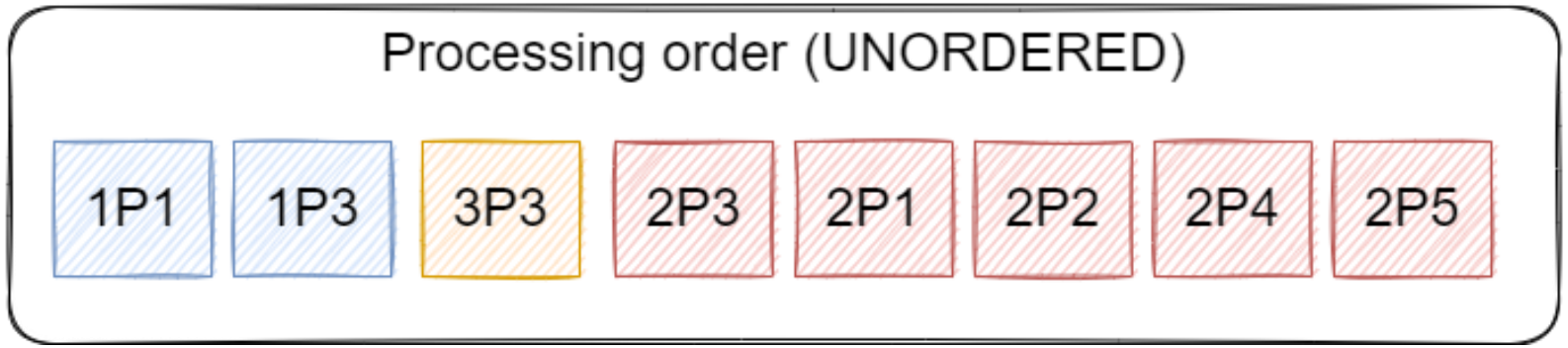
Пишем код с Parallel consumer



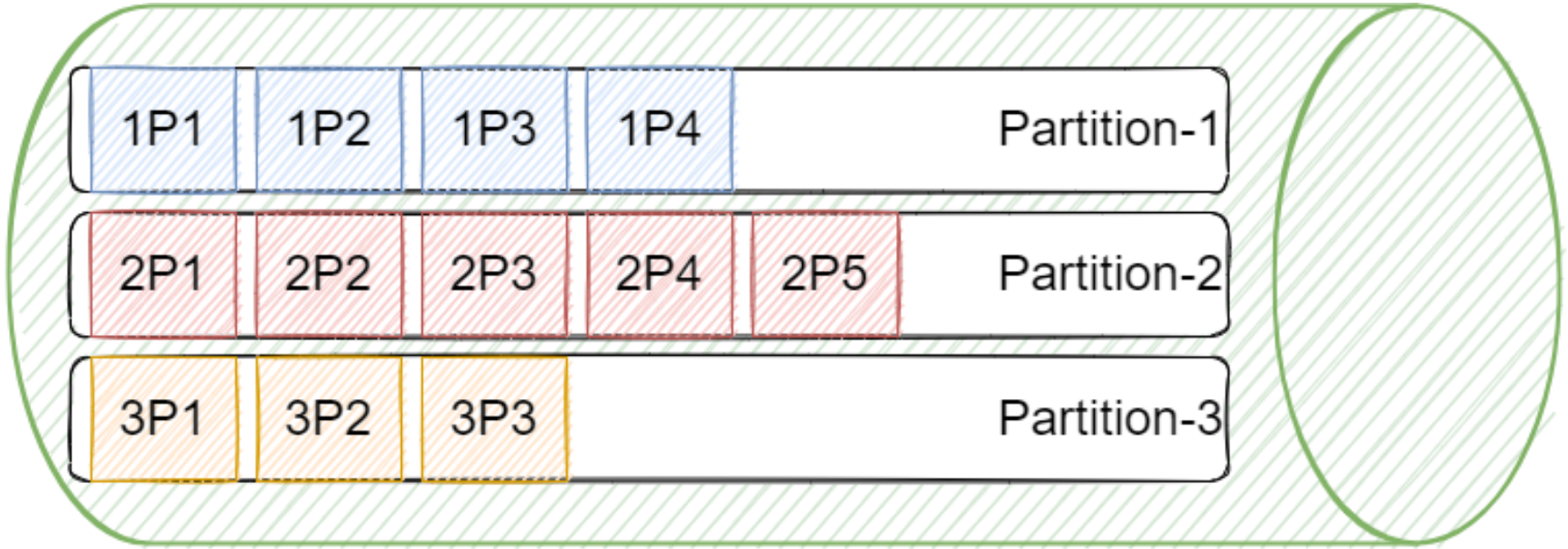


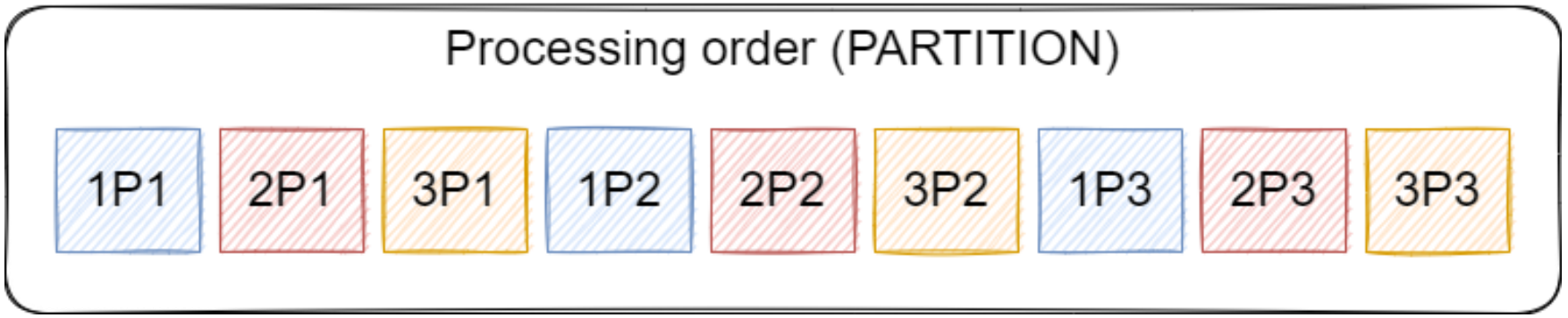
Пишем код с Parallel consumer





Пишем код с Parallel consumer





Пишем код с Parallel consumer

```
var kafkaConsumer = getKafkaConsumer();
var kafkaProducer = getKafkaProducer();
var options = ParallelConsumerOptions.<K, V>builder()
        .ordering(UNORDERED) // PARTITION, KEY
        .maxConcurrency(1000) // Default 16
        .batchSize(100) // Default 1
        .commitMode(PERIODIC_CONSUMER_ASYNCHRONOUS) //
PERIODIC_CONSUMER_SYNC, PERIODIC_TRANSACTIONAL_PRODUCER
        .consumer(kafkaConsumer)
        .producer(kafkaProducer)
        .build();
```

Пишем код с Parallel consumer

PR на подбор значения для ***maxConcurrency()***
https://github.com/confluentinc/parallel-consumer/pull/22

Пишем код с Parallel consumer

Библиотека Netflix по ограничению параллелизации для максимального ***throughput***
https://github.com/Netflix/concurrency-limits

Пишем код с Parallel consumer

```
var kafkaConsumer = getKafkaConsumer();
var kafkaProducer = getKafkaProducer();
var options = ParallelConsumerOptions.<K, V>builder()
        .ordering(UNORDERED) // PARTITION, KEY
        .maxConcurrency(1000) // Default 16
        .batchSize(100) // Default 1
        .commitMode(PERIODIC_CONSUMER_ASYNCHRONOUS) //
PERIODIC_CONSUMER_SYNC, PERIODIC_TRANSACTIONAL_PRODUCER
        .consumer(kafkaConsumer)
        .producer(kafkaProducer)
        .build();
```

Пишем код с Parallel consumer

```
var kafkaConsumer = getKafkaConsumer();
var kafkaProducer = getKafkaProducer();
var options = ParallelConsumerOptions.<K, V>builder()
        .ordering(UNORDERED) // PARTITION, KEY
        .maxConcurrency(1000) // Default 16
        .batchSize(100) // Default 1
        .commitMode(PERIODIC_CONSUMER_ASYNCHRONOUS) //
PERIODIC_CONSUMER_SYNC, PERIODIC_TRANSACTIONAL_PRODUCER
        .consumer(kafkaConsumer)
        .producer(kafkaProducer)
        .build();
```

Движки Parallel Processor

Движки Parallel Processor

- ParallelStreamProcessor – *`poll`* и *`pollAndProduce`*

Движки Parallel Processor

- ParallelStreamProcessor – *`poll`* и *`pollAndProduce`*
- VertxParallelStreamProcessor – неблокирующие вызовы API

Движки Parallel Processor

- ParallelStreamProcessor – *`poll`* и *`pollAndProduce`*
- VertxParallelStreamProcessor – неблокирующие вызовы API
- ReactorProcessor – *`Mono`* и *`Flux`*

Движки Parallel Processor

- ParallelStreamProcessor – *poll* и *pollAndProduce*
- VertxParallelStreamProcessor – неблокирующие вызовы API
- ReactorProcessor – *Mono* и *Flux*
- JStreamParallelStreamProcessor – *poll* и *pollAndProduce* + Stream API
- JStreamVertxStreamProcessor – неблокирующие вызовы API + Stream API

ParallelStreamProcessor

```
try (var eosStreamProcessor = ParallelStreamProcessor.createEosStreamProcessor(options)) {

    eosStreamProcessor.subscribe(Collections.singleton(inputTopic));

    eosStreamProcessor.pollAndProduce(context -> {
                var consumerRecord = context.getSingleRecord().getConsumerRecord();
                var result = userProcessor.apply(consumerRecord);
                return new ProducerRecord<>(outputTopic, consumerRecord.key(), result);
            }, consumeProduceResult -> {
                log.debug("Message {} saved to broker at offset {}",
                        consumeProduceResult.getOut(),
                        consumeProduceResult.getMeta().offset());
            }
    );
}
```

ParallelStreamProcessor

```
try (var eosStreamProcessor = ParallelStreamProcessor.createEosStreamProcessor(options)) {

    eosStreamProcessor.subscribe(Collections.singleton(inputTopic));

    eosStreamProcessor.pollAndProduce(context -> {
                var consumerRecord = context.getSingleRecord().getConsumerRecord();
                var result = userProcessor.apply(consumerRecord);
                return new ProducerRecord<>(outputTopic, consumerRecord.key(), result);
            }, consumeProduceResult -> {
                log.debug("Message {} saved to broker at offset {}",
                        consumeProduceResult.getOut(),
                        consumeProduceResult.getMeta().offset());
            }
    );
}
```

ParallelStreamProcessor

```
try (var eosStreamProcessor = ParallelStreamProcessor.createEosStreamProcessor(options)) {

    eosStreamProcessor.subscribe(Collections.singleton(inputTopic));

    eosStreamProcessor.pollAndProduce(context -> {
                var consumerRecord = context.getSingleRecord().getConsumerRecord();
                var result = userProcessor.apply(consumerRecord);
                return new ProducerRecord<>(outputTopic, consumerRecord.key(), result);
            }, consumeProduceResult -> {
                log.debug("Message {} saved to broker at offset {}",
                        consumeProduceResult.getOut(),
                        consumeProduceResult.getMeta().offset());
            }
    );
}
```

ParallelStreamProcessor

```
try (var eosStreamProcessor = ParallelStreamProcessor.createEosStreamProcessor(options)) {

    eosStreamProcessor.subscribe(Collections.singleton(inputTopic));

    eosStreamProcessor.pollAndProduce(context -> {
                var consumerRecord = context.getSingleRecord().getConsumerRecord();
                var result = userProcessor.apply(consumerRecord);
                return new ProducerRecord<>(outputTopic, consumerRecord.key(), result);
            }, consumeProduceResult -> {
                log.debug("Message {} saved to broker at offset {}",
                        consumeProduceResult.getOut(),
                        consumeProduceResult.getMeta().offset());
            }
    );
}
```

ParallelStreamProcessor

```
try (var eosStreamProcessor = ParallelStreamProcessor.createEosStreamProcessor(options)) {

    eosStreamProcessor.subscribe(Collections.singleton(inputTopic));

    eosStreamProcessor.pollAndProduce(context -> {
                var consumerRecord = context.getSingleRecord().getConsumerRecord();
                var result = userProcessor.apply(consumerRecord);
                return new ProducerRecord<>(outputTopic, consumerRecord.key(), result);
            }, consumeProduceResult -> {
                log.debug("Message {} saved to broker at offset {}",
                        consumeProduceResult.getOut(),
                        consumeProduceResult.getMeta().offset());
            }
    );
}
```

ParallelStreamProcessor

```
try (var eosStreamProcessor = ParallelStreamProcessor.createEosStreamProcessor(options)) {

    eosStreamProcessor.subscribe(Collections.singleton(inputTopic));

    eosStreamProcessor.pollAndProduce(context -> {
                var consumerRecord = context.getSingleRecord().getConsumerRecord();
                var result = userProcessor.apply(consumerRecord);
                return new ProducerRecord<>(outputTopic, consumerRecord.key(), result);
            }, consumeProduceResult -> {
                log.debug("Message {} saved to broker at offset {}",
                        consumeProduceResult.getOut(),
                        consumeProduceResult.getMeta().offset());
            }
    );
}
```

JStreamVertxParallelStreamProcessor

```
try (var vertxProcessor = JStreamVertxParallelStreamProcessor.createEosStreamProcessor(options)) {

    vertxProcessor.subscribe(Collections.singleton(inputTopic));

    var resultStream = vertxProcessor.vertxHttpReqInfoStream(context -> {
        var consumerRecord = context.getSingleConsumerRecord();
        log.info("Concurrently constructing and returning RequestInfo from record: {}", consumerRecord);
        var params = Map.of("recordKey", consumerRecord.key(), "payload", consumerRecord.value());
        return new VertxParallelEoSStreamProcessor.RequestInfo("localhost", port, "/api", params);
    });

    resultStream.forEach(x -> log.info("From result stream: {}", x));
}
```

JStreamVertxParallelStreamProcessor

```
try (var vertxProcessor = JStreamVertxParallelStreamProcessor.createEosStreamProcessor(options)) {

    vertxProcessor.subscribe(Collections.singleton(inputTopic));

    var resultStream = vertxProcessor.vertxHttpReqInfoStream(context -> {
        var consumerRecord = context.getSingleConsumerRecord();
        log.info("Concurrently constructing and returning RequestInfo from record: {}", consumerRecord);
        var params = Map.of("recordKey", consumerRecord.key(), "payload", consumerRecord.value());
        return new VertxParallelEoSStreamProcessor.RequestInfo("localhost", port, "/api", params);
    });

    resultStream.forEach(x -> log.info("From result stream: {}", x));
}
```

JStreamVertxParallelStreamProcessor

```
try (var vertxProcessor = JStreamVertxParallelStreamProcessor.createEosStreamProcessor(options)) {

    vertxProcessor.subscribe(Collections.singleton(inputTopic));

    var resultStream = vertxProcessor.vertxHttpReqInfoStream(context -> {
        var consumerRecord = context.getSingleConsumerRecord();
        log.info("Concurrently constructing and returning RequestInfo from record: {}", consumerRecord);
        var params = Map.of("recordKey", consumerRecord.key(), "payload", consumerRecord.value());
        return new VertxParallelEoSStreamProcessor.RequestInfo("localhost", port, "/api", params);
    });

    resultStream.forEach(x -> log.info("From result stream: {}", x));
}
```

JStreamVertxParallelStreamProcessor

```
try (var vertxProcessor = JStreamVertxParallelStreamProcessor.createEosStreamProcessor(options)) {

    vertxProcessor.subscribe(Collections.singleton(inputTopic));

    var resultStream = vertxProcessor.vertxHttpReqInfoStream(context -> {
        var consumerRecord = context.getSingleConsumerRecord();
        log.info("Concurrently constructing and returning RequestInfo from record: {}", consumerRecord);
        var params = Map.of("recordKey", consumerRecord.key(), "payload", consumerRecord.value());
        return new VertxParallelEoSStreamProcessor.RequestInfo("localhost", port, "/api", params);
    });

    resultStream.forEach(x -> log.info("From result stream: {}", x));
}
```

ReactorProcessor

```
new ReactorProcessor<>(options).react(context -> {

    var consumerRecord = context.getSingleRecord().getConsumerRecord();
    log.info("Concurrently constructing and returning RequestInfo from record: {}", consumerRecord);
    return Mono.just("something todo");
});
```

ReactorProcessor

```
new ReactorProcessor<>(options).react(context -> {

    var consumerRecord = context.getSingleRecord().getConsumerRecord();
    log.info("Concurrently constructing and returning RequestInfo from record: {}", consumerRecord);
    return Mono.just("something todo");
});
```

Примеры с кодом parallel consumer

https://github.com/confluentinc/parallel-consumer/tree/master/parallel-consumer-examples

Недостатки Parallel Consumer

Недостатки Parallel Consumer

- Ограниченная функциональность консьюмера

1. https://www.instaclustr.com/blog/kafka-parallel-consumer-part-1/
2. https://www.instaclustr.com/blog/improving-apache-kafka-performance-and-scalability-with-the-parallel-consumer-part-2/

Недостатки Parallel Consumer

- Нестабильная работа, нестабильная версия

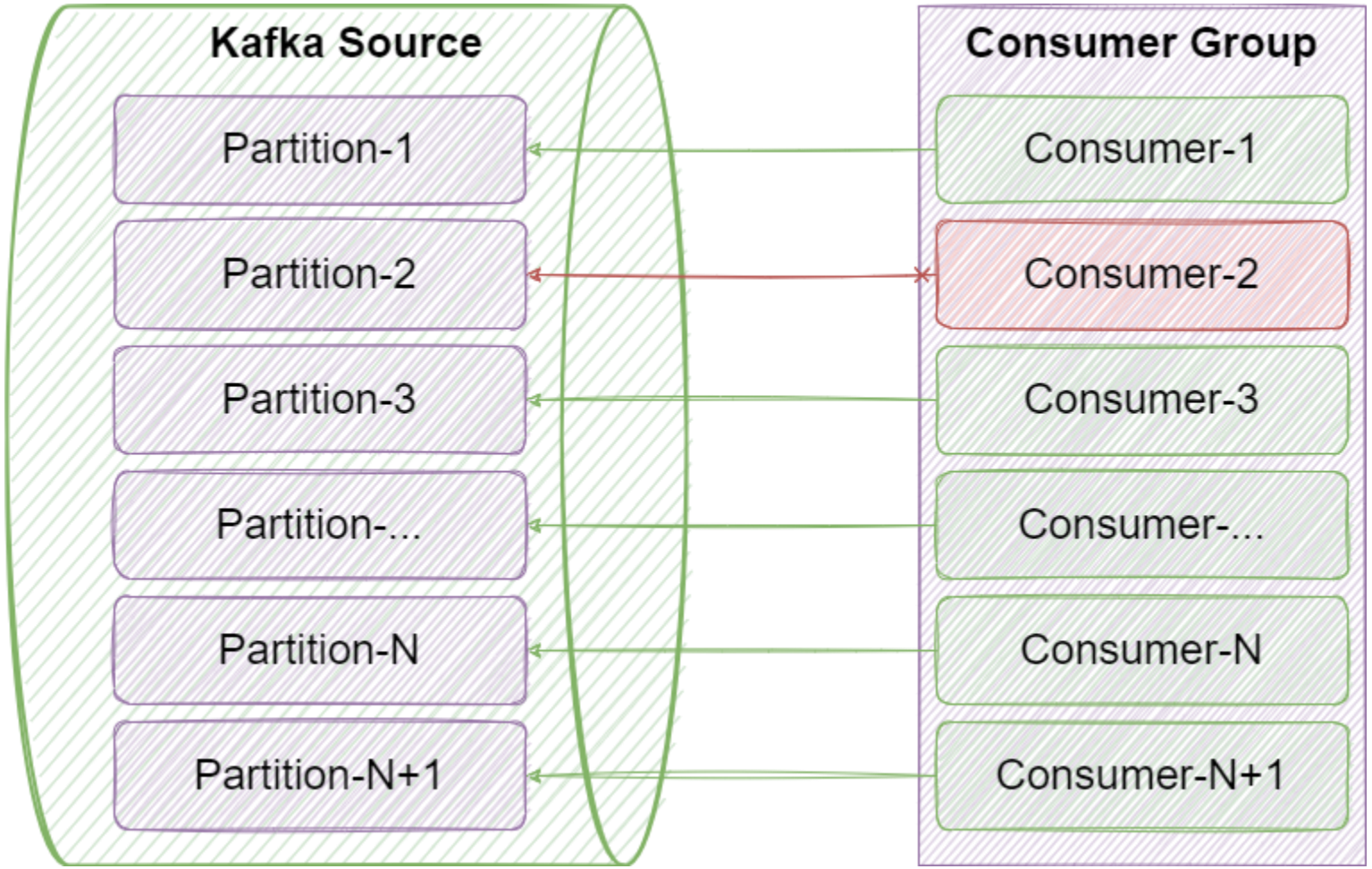
https://github.com/confluentinc/parallel-consumer/issues

Недостатки Parallel Consumer

- Большие консьюмерные группы – это боль

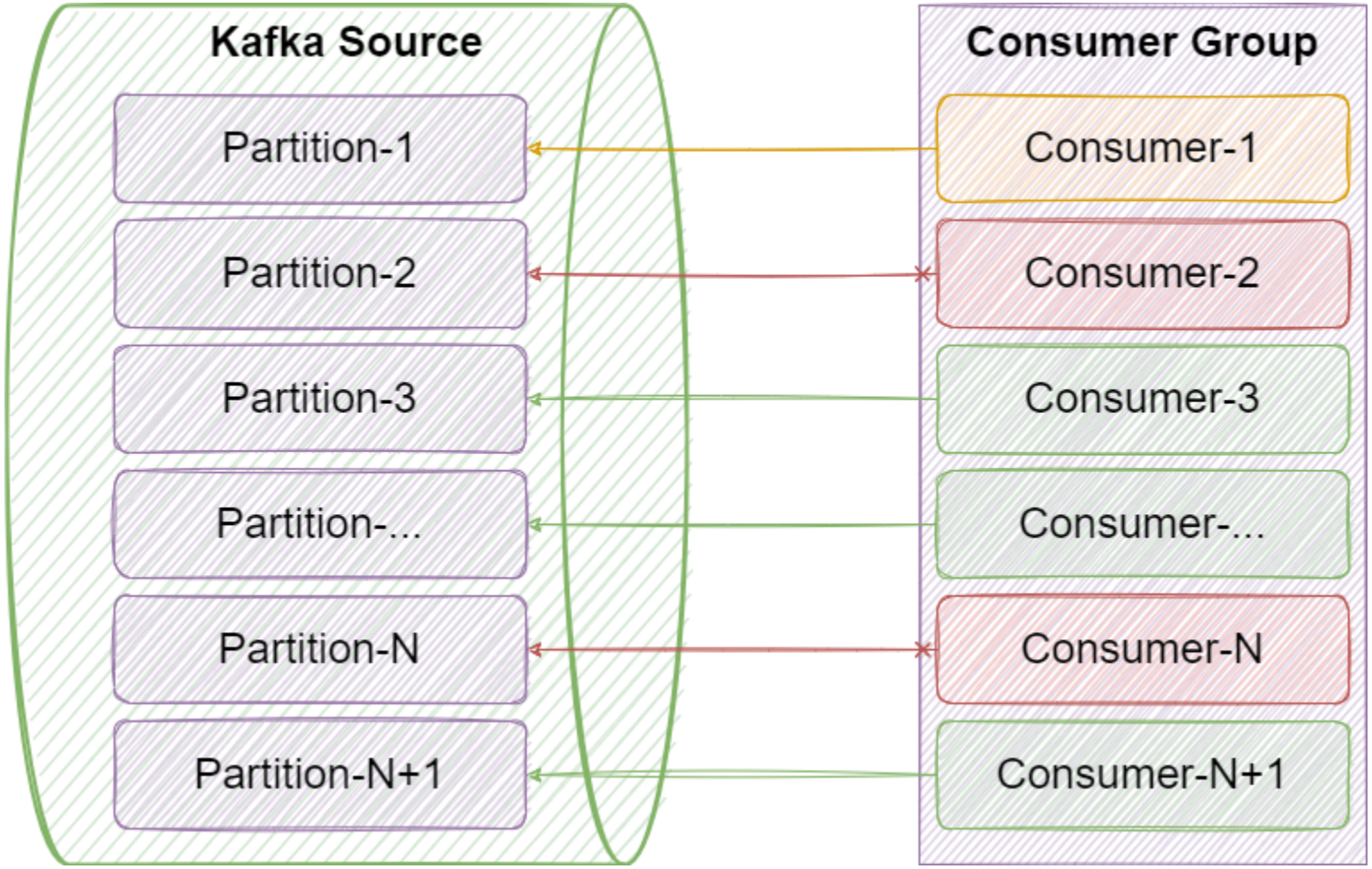
Недостатки Parallel Consumer

- Большие консьюмерные группы – это боль



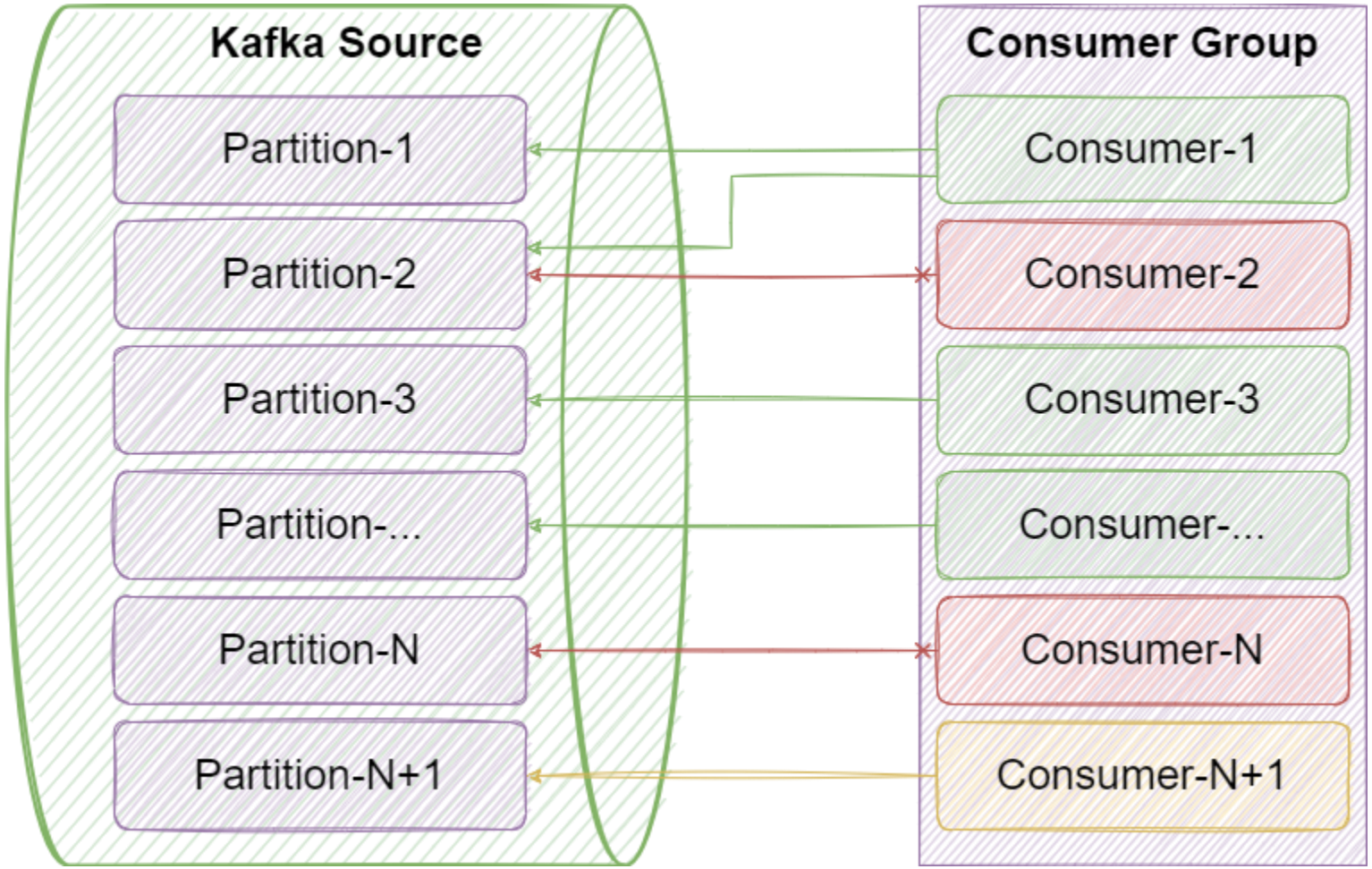
Недостатки Parallel Consumer

- Большие консьюмерные группы – это боль



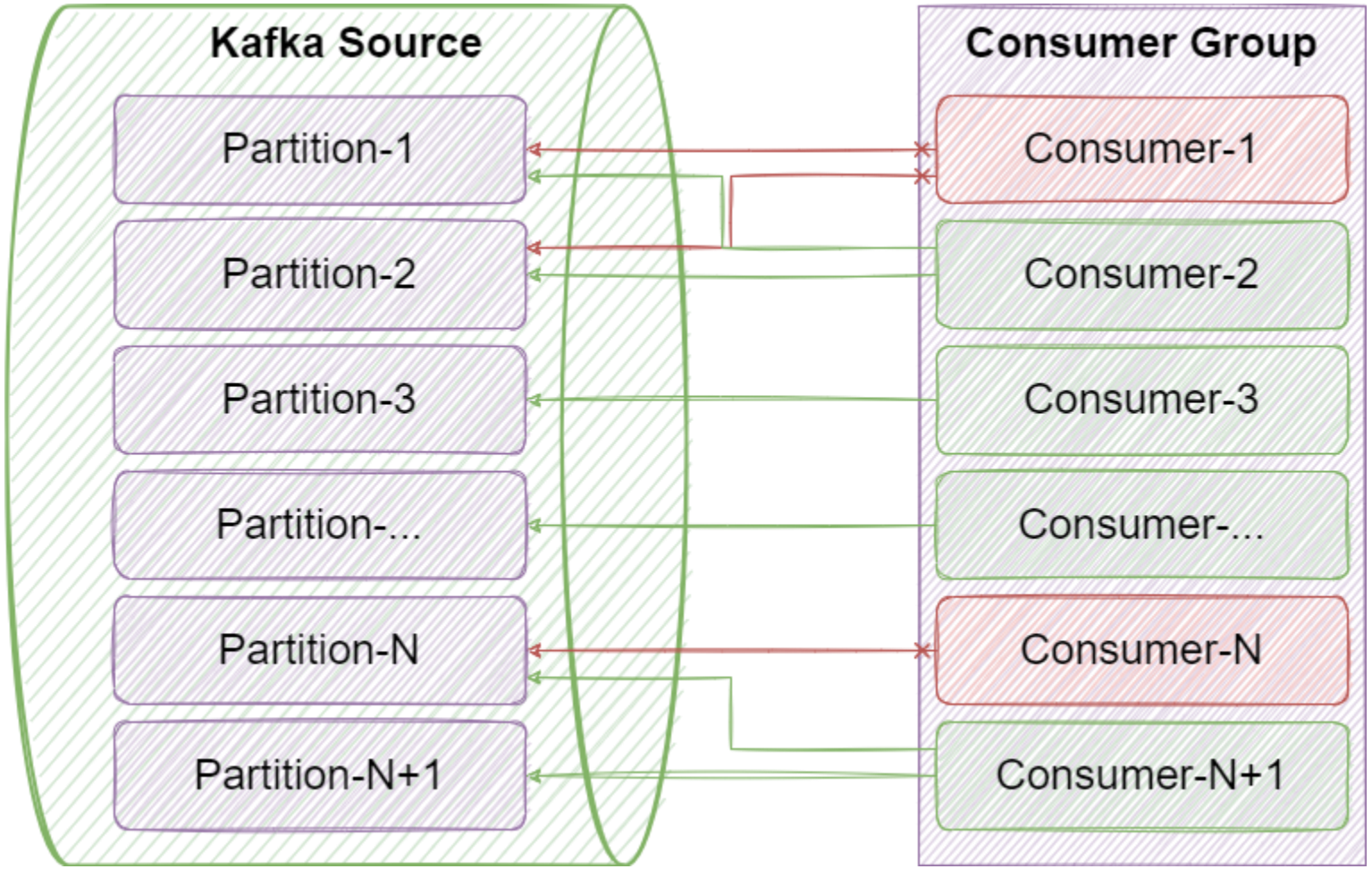
Недостатки Parallel Consumer

- Большие консьюмерные группы – это боль



Недостатки Parallel Consumer

- Большие консьюмерные группы – это боль



Недостатки Parallel Consumer

- Большие консьюмерные группы – это боль

Григорий Кошелев — Когда всё пошло по Кафке

Kafka:

Producer:

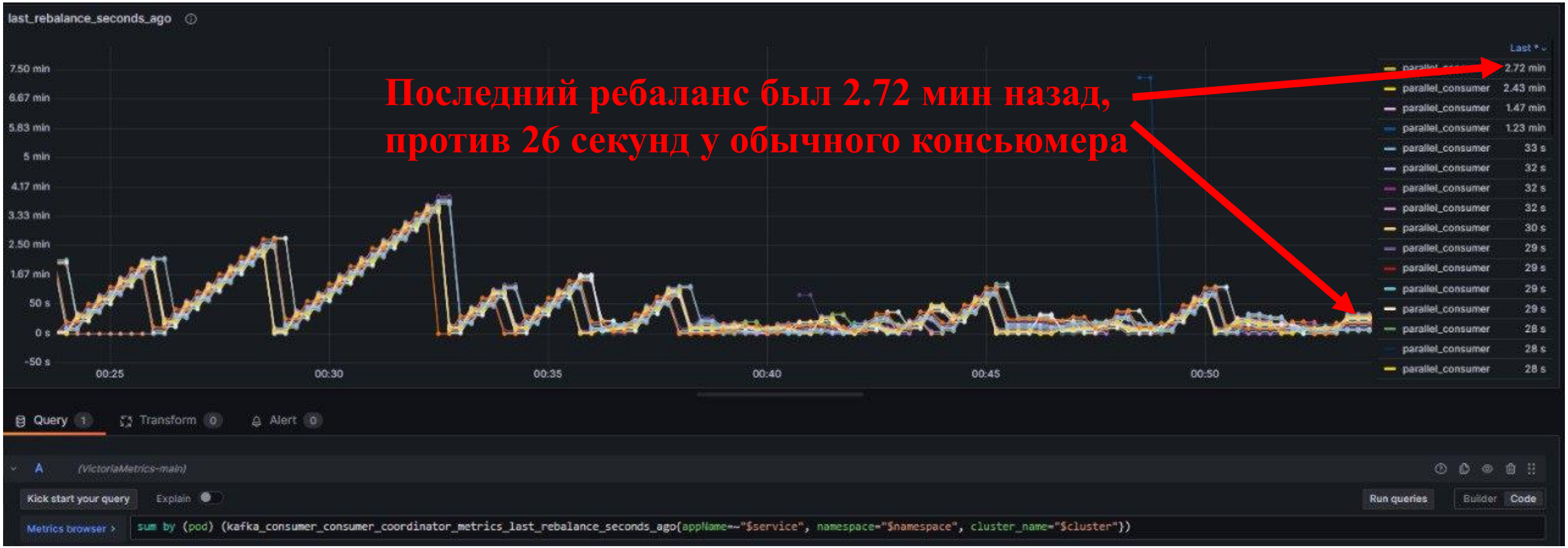
Consumer:

Недостатки Parallel Consumer

- Большие консьюмерные группы – это боль

Недостатки Parallel Consumer

- Большие консьюмерные группы – это боль



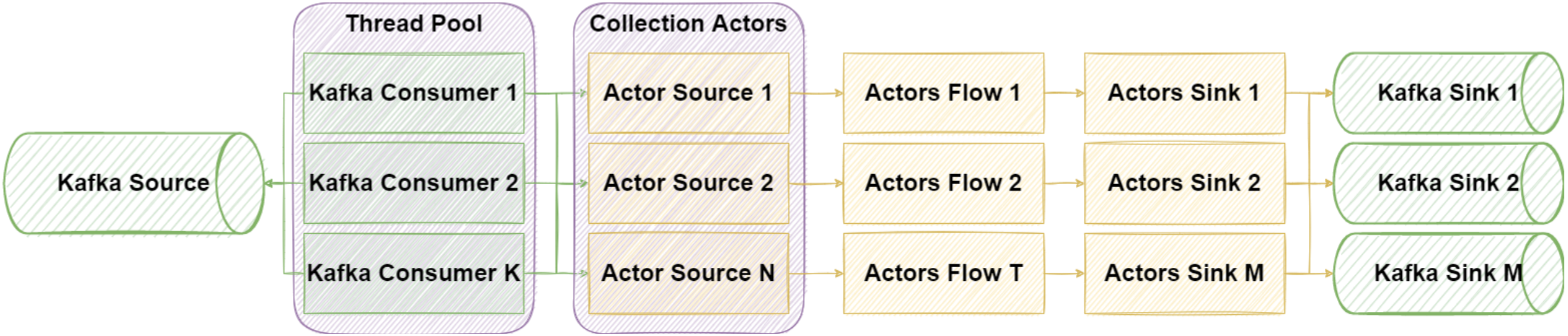
Недостатки Parallel Consumer – итог

- Это неполноценная замена консьюмеру по функциональности
- Использование в production – дополнительный риск
- При длительной работе показывает нестабильное поведение
- Нужно тонко подбирать ключевые параметры (`ordering`, `maxConcurrency`, `batchSize`, `commitMode`) с параметрами обычного консьюмера

Параллелизация с помощью реактивных фреймворков

- Reactive Stream
- RxJava
- Project Reactor
- Akka Streams (Apache Pekko)
- Vert.x
- Micronaut (RxJava, Reactor)
- …

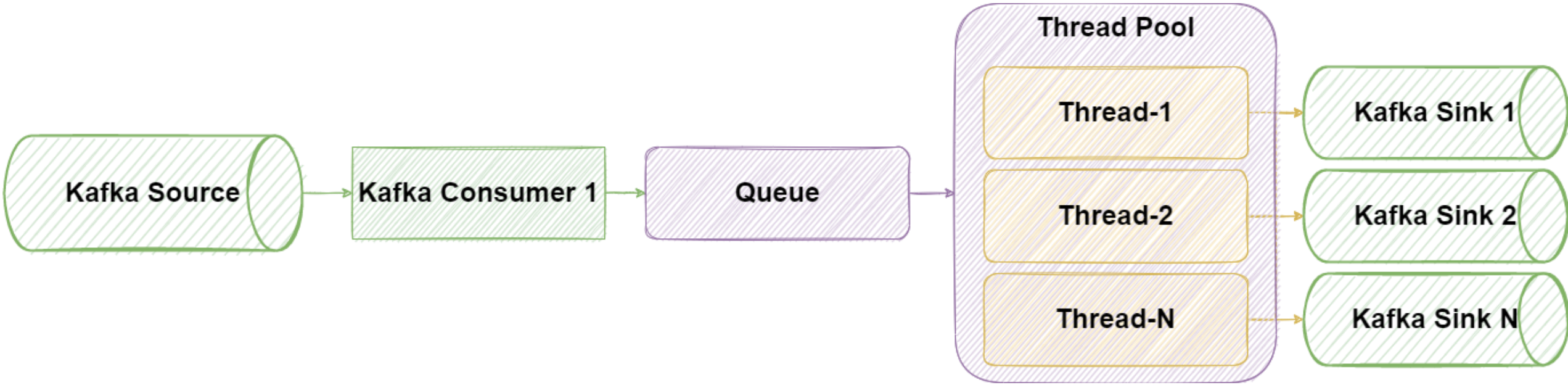
Консьюмер + Akka Streams



Консьюмер + Akka Streams

Евгений Ненахов —
Akka Streams в реальных задачах

Консьюмер + Thread Pool



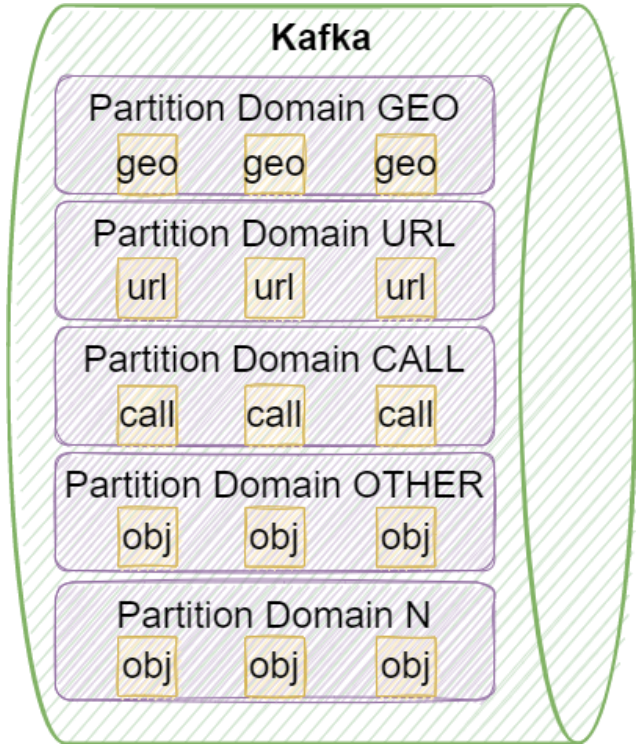
Консьюмер + Thread Pool

https://github.com/inovatrend/mtc-demo/blob/master/src/main/java/com/inovatrend/mtcdemo/MultithreadedKafkaConsumer.java

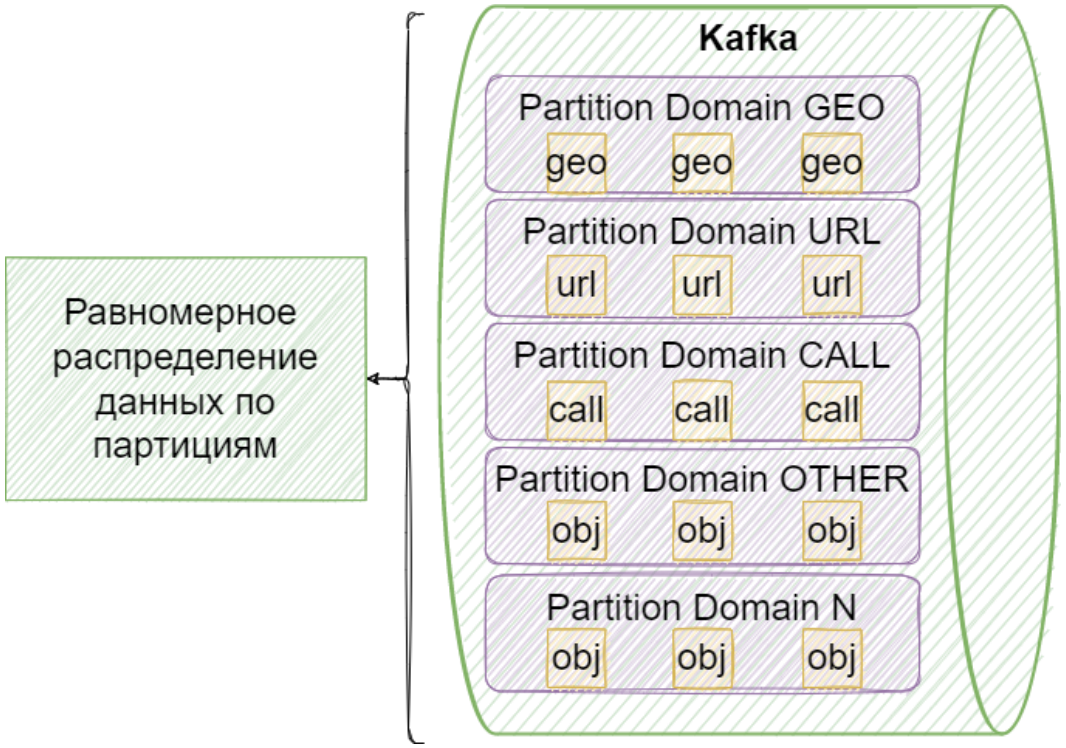
Решения проблем производительности

1. Собственный кластер Kafka
2. Parallel consumer
3. Handmade: Consumer + Reactive Framework
4. Handmade: Consumer + ThreadPool

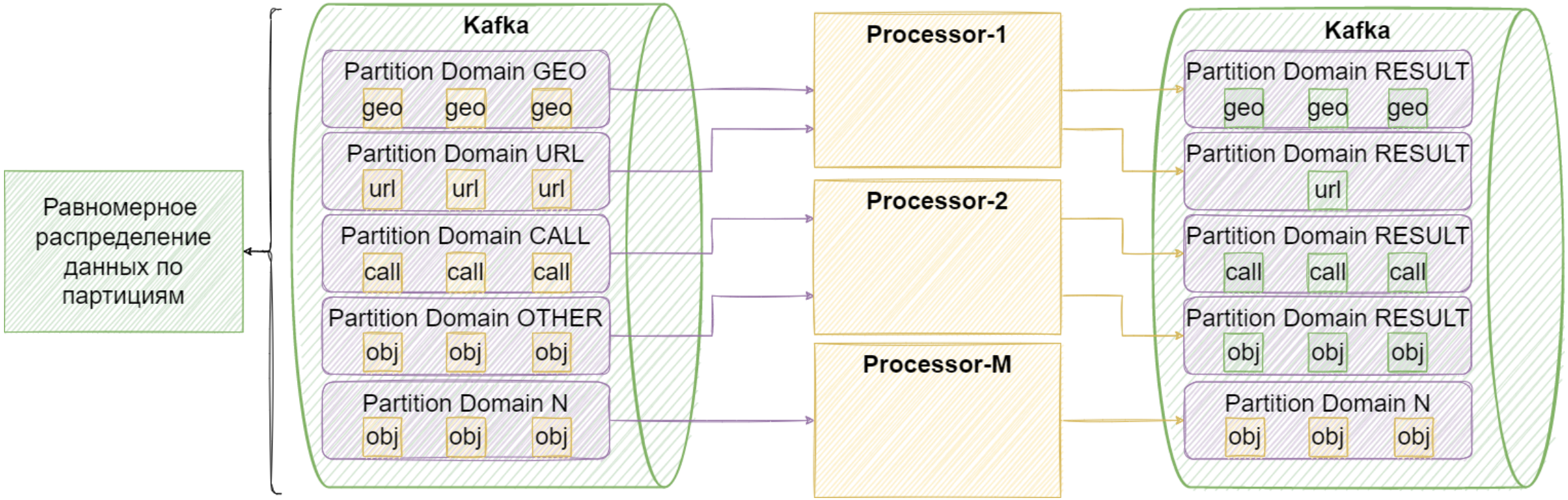
Проблема распределения партиций



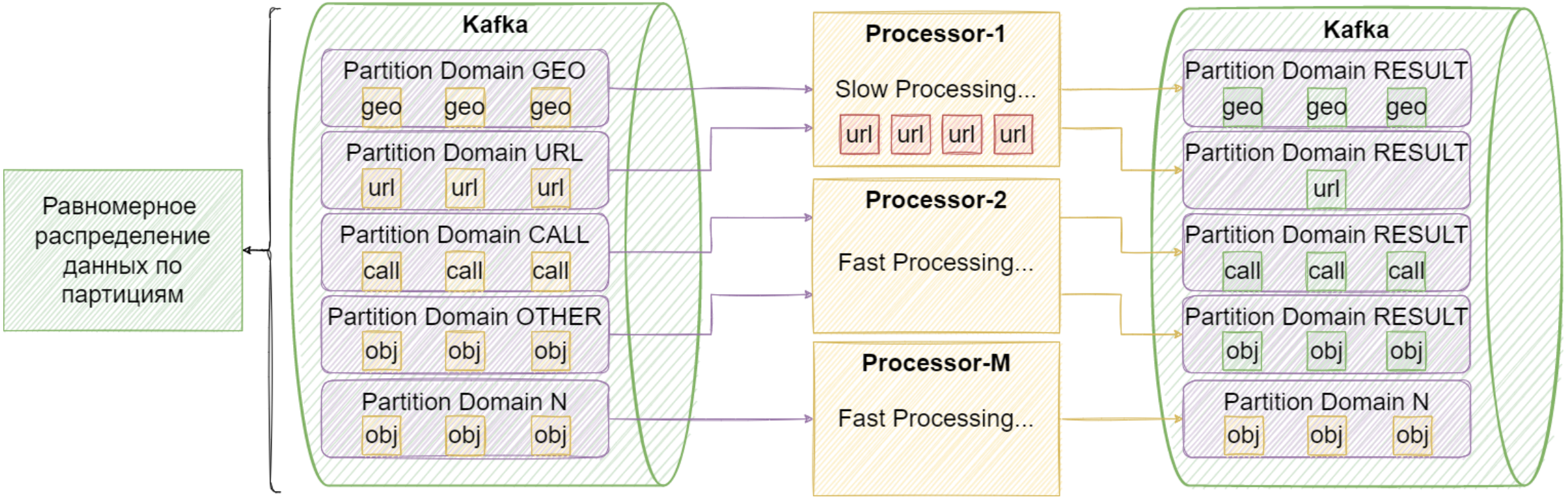
Проблема распределения партиций



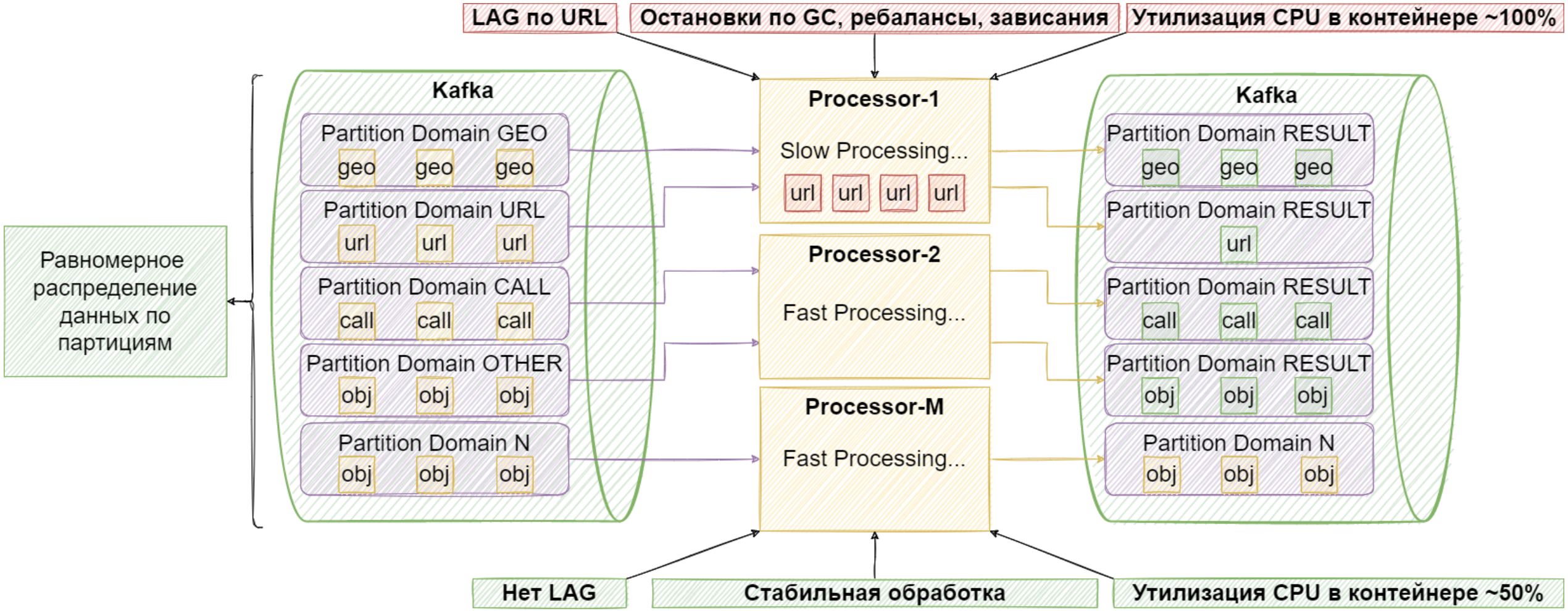
Проблема распределения партиций



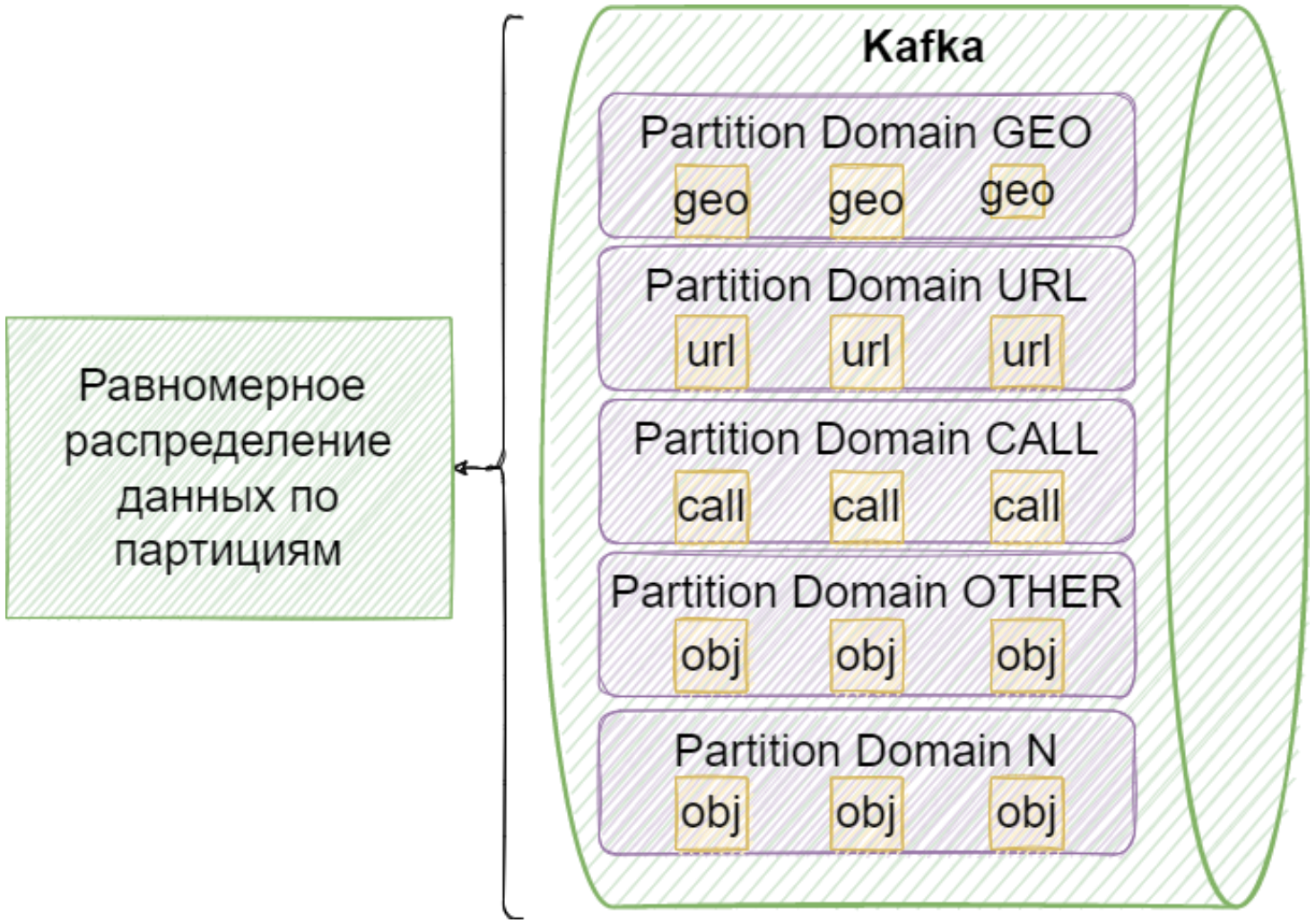
Проблема распределения партиций

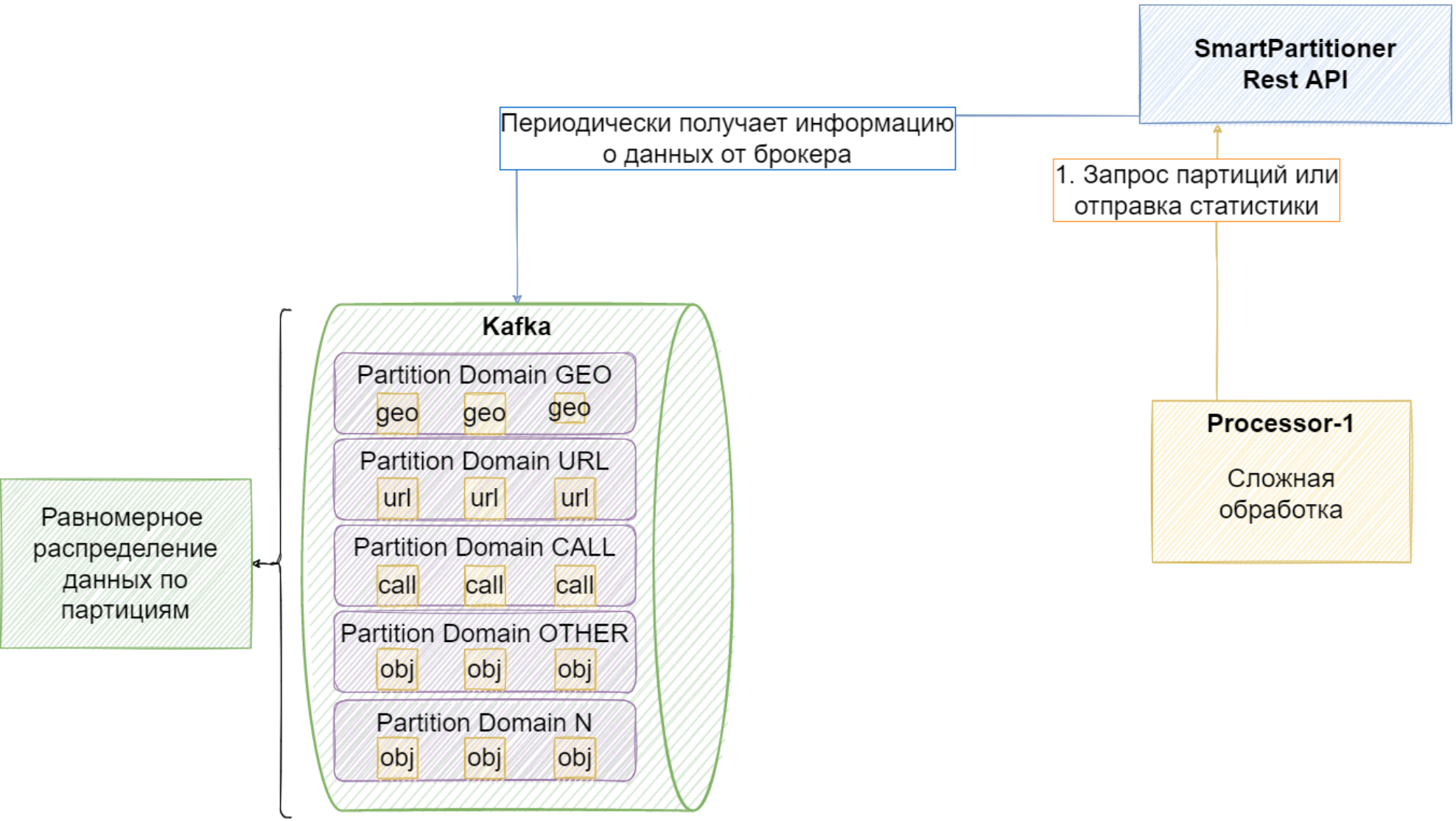


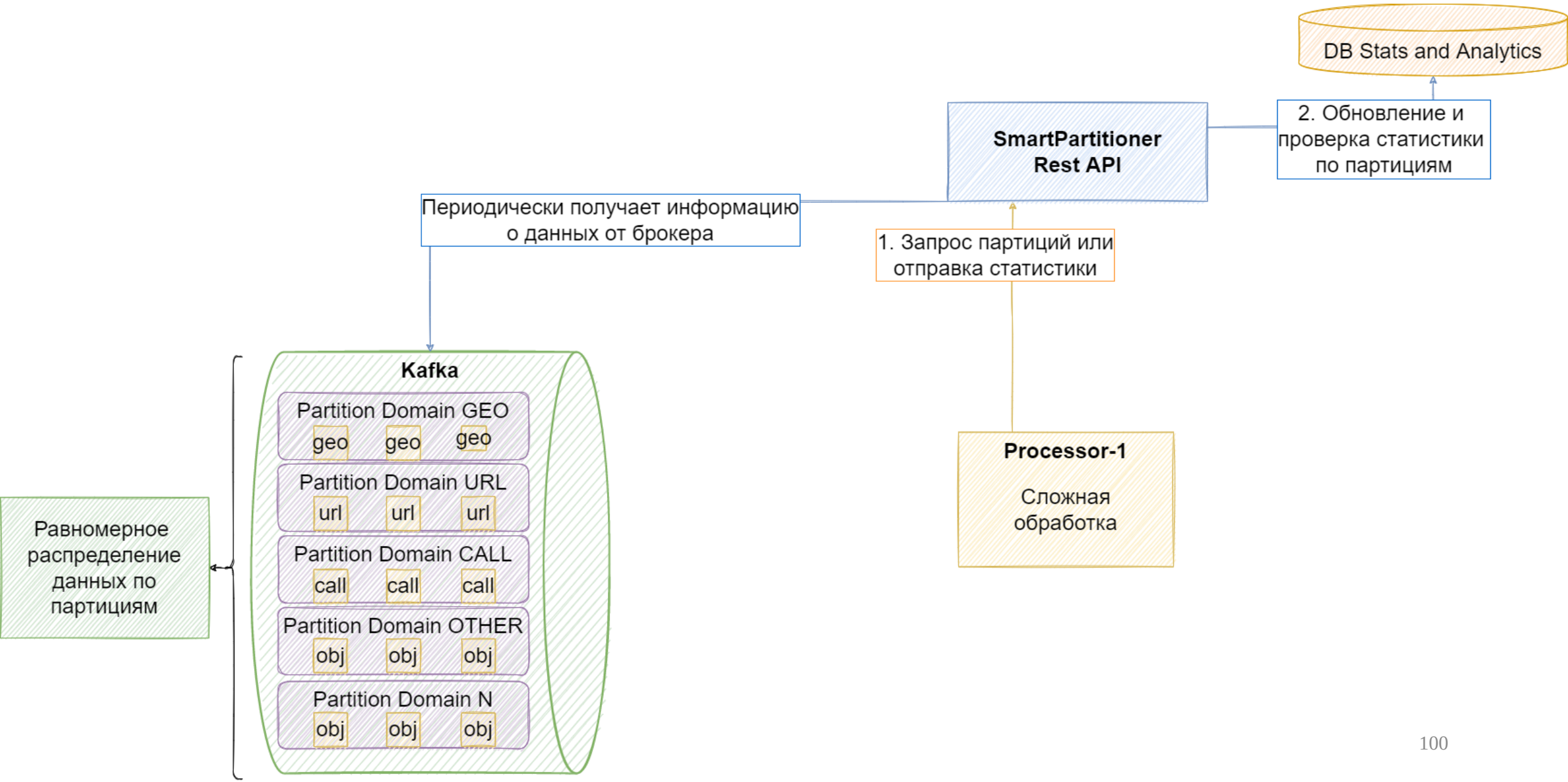
Проблема распределения партиций



Вариант решения проблемы

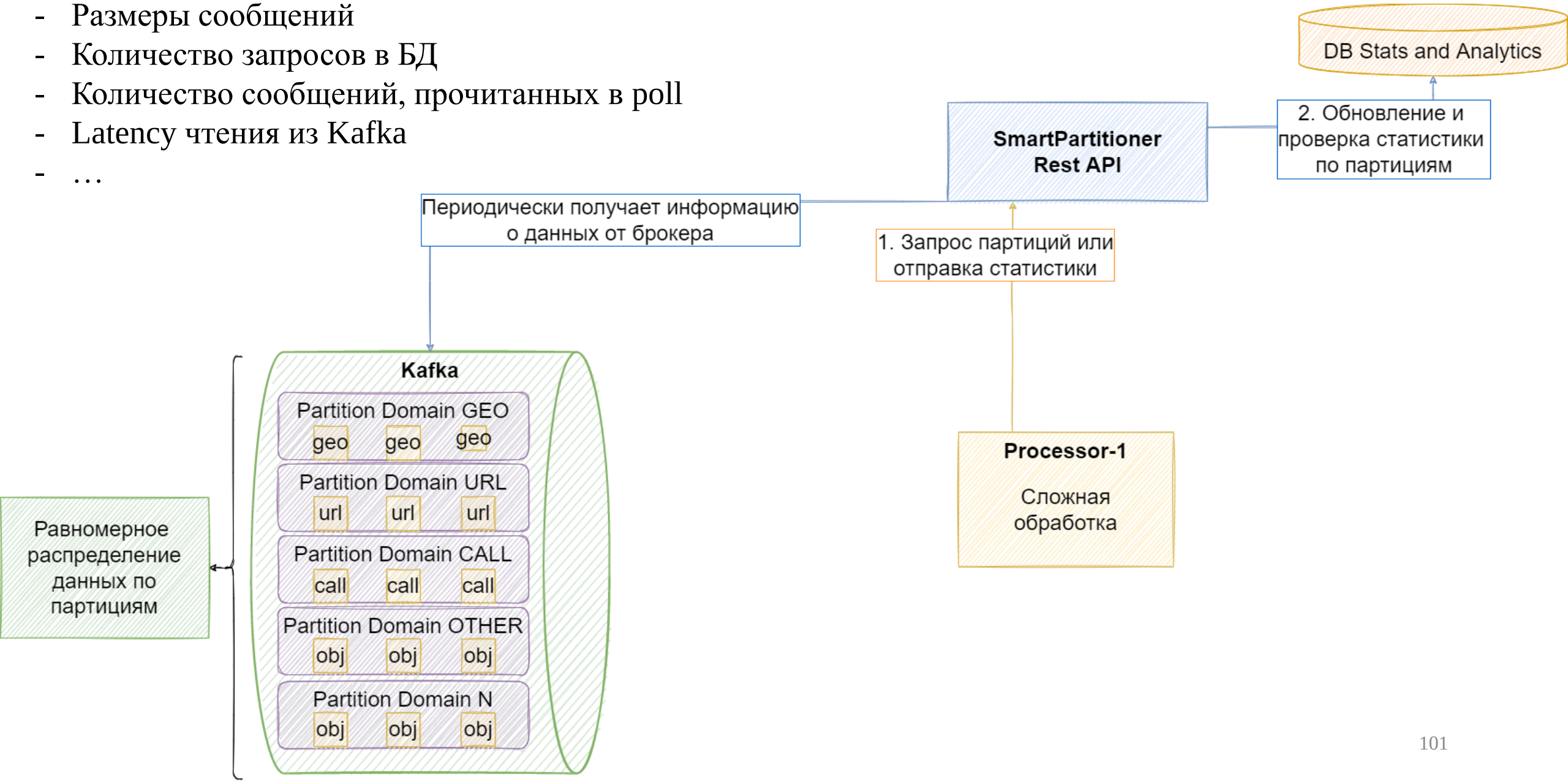


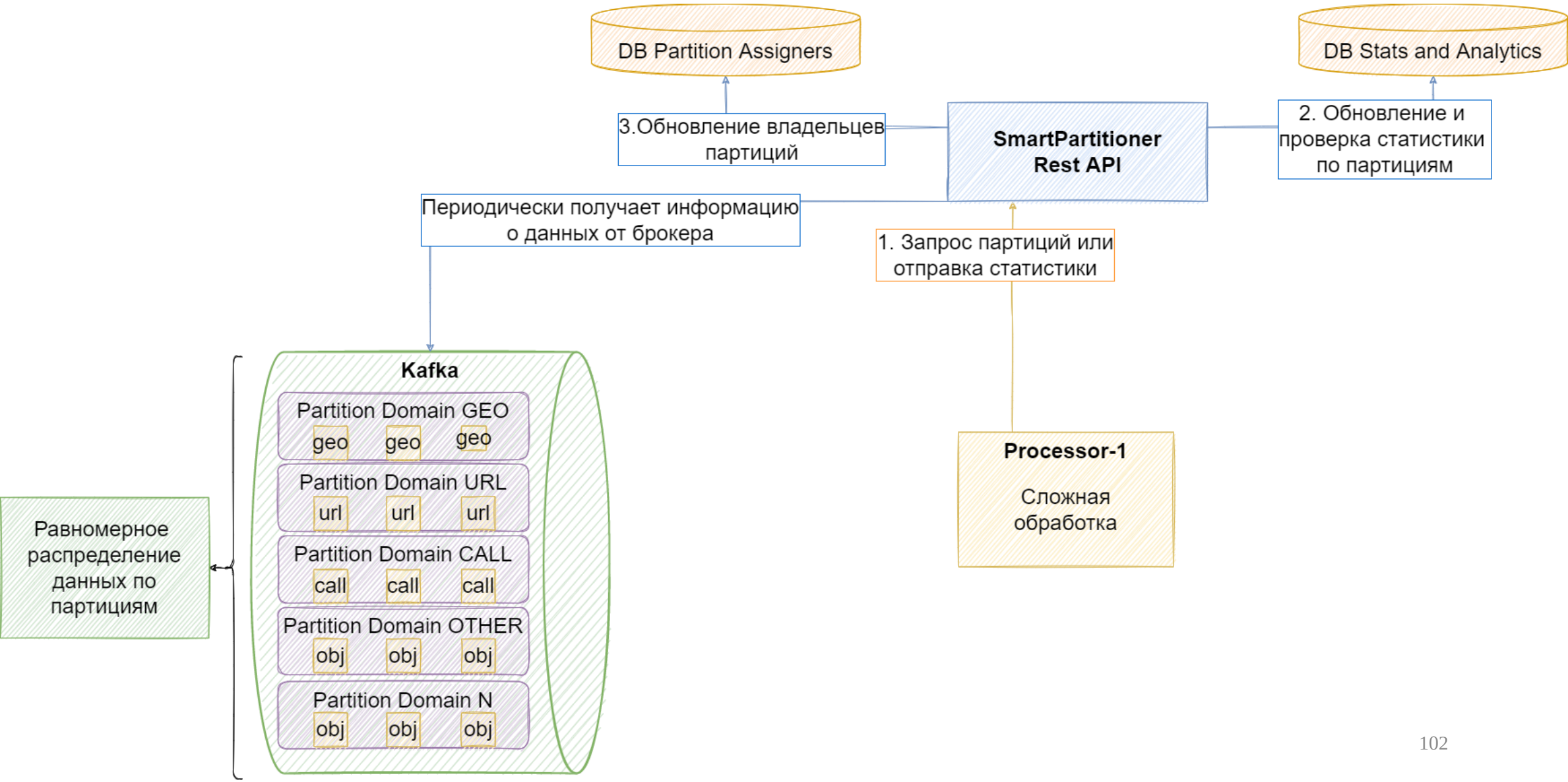

SmartPartitioner
Rest API
Периодически получает информацию
о данных от брокера
1. Запрос партиций или
отправка статистики
Kafka
Partition Domain GEO
geo geo geo
Partition Domain URL
url url url
Partition Domain CALL
call call call
Partition Domain OTHER
obj obj obj
Partition Domain N
obj obj obj
Processor-1
Сложная
обработка
Равномерное
распределение
данных по
партициям

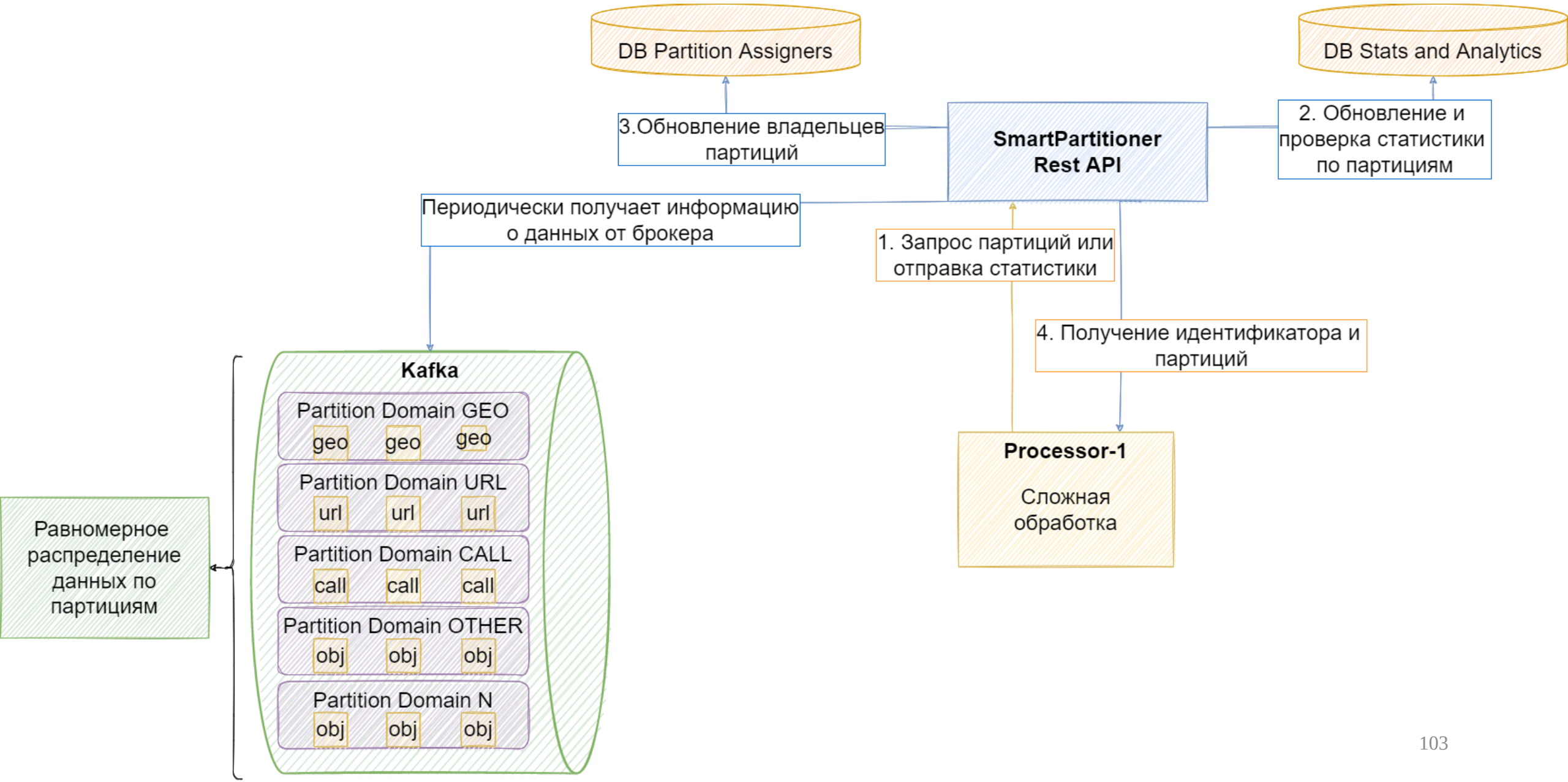

DB Stats and Analytics
SmartPartitioner
Rest API
2. Обновление и проверка статистики по партициям
Периодически получает информацию о данных от брокера
1. Запрос партиций или отправка статистики
Kafka
Partition Domain GEO
geo geo geo
Partition Domain URL
url url url
Partition Domain CALL
call call call
Partition Domain OTHER
obj obj obj
Partition Domain N
obj obj obj
Равномерное распределение данных по партициям
Processor-1
Сложная обработка

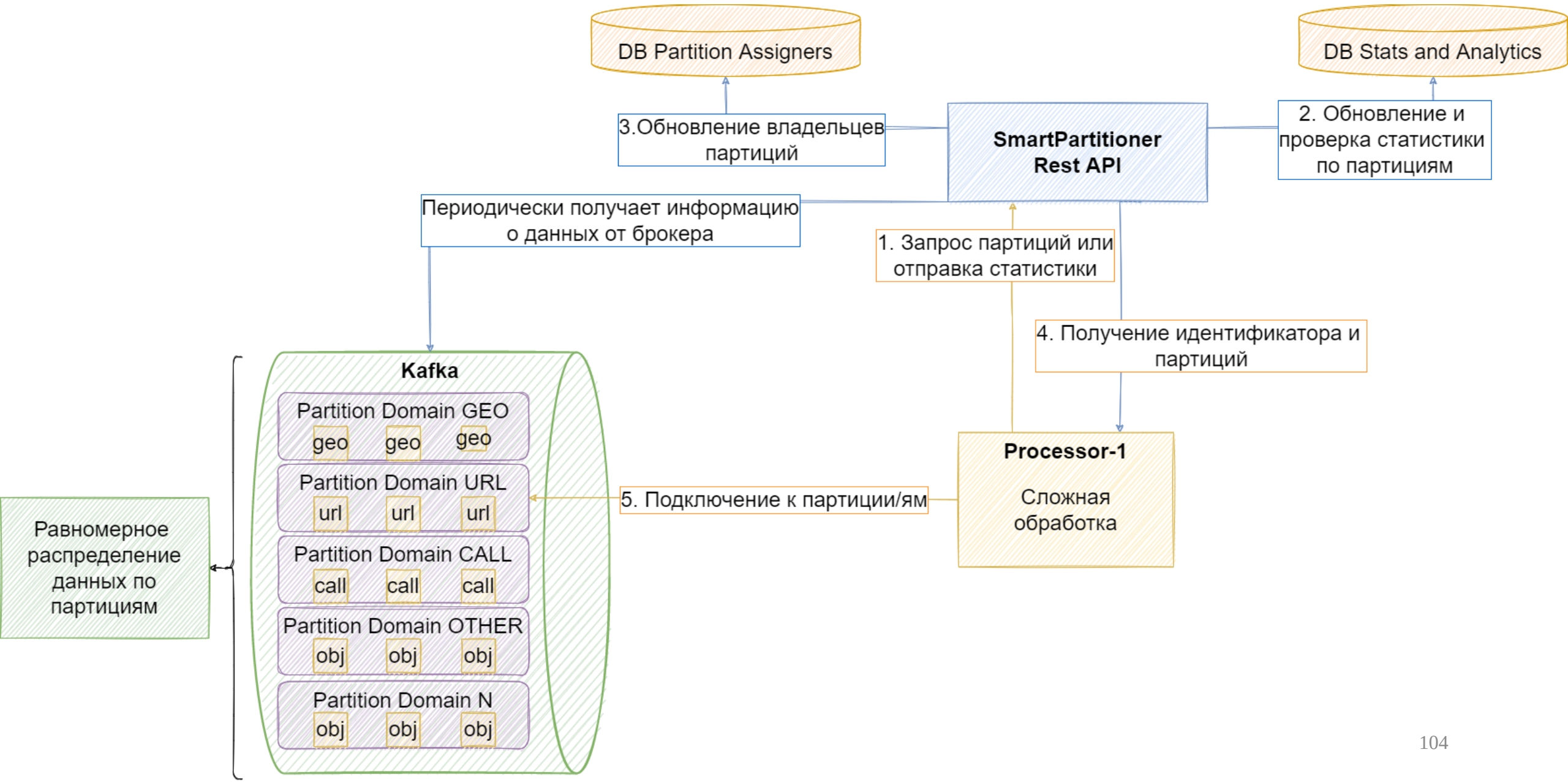
Какие данные отправлять в статистику:

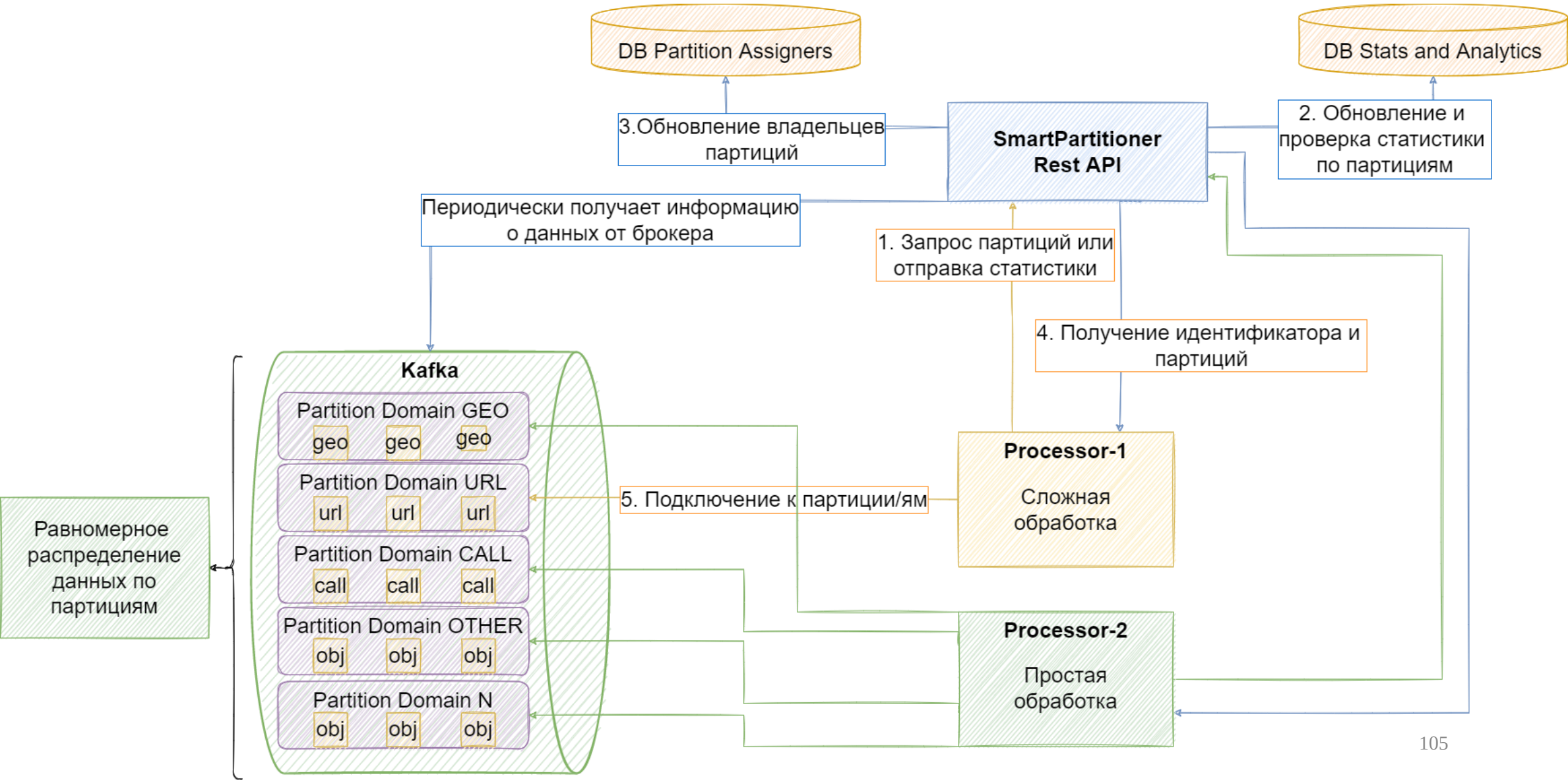
- Скорость обработки сообщений
- Размеры сообщений
- Количество запросов в БД
- Количество сообщений, прочитанных в poll
- Latency чтения из Kafka
- …

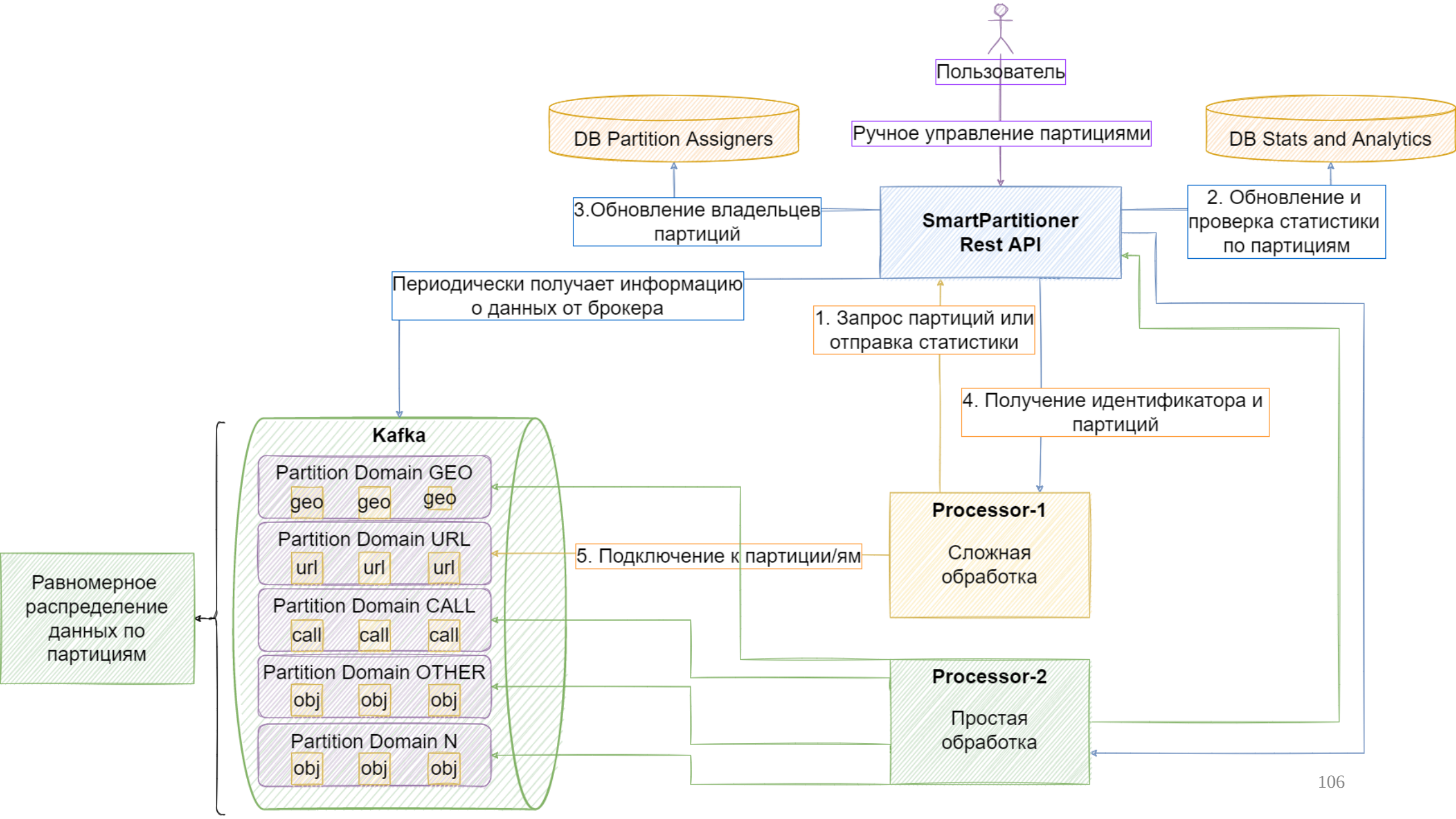



DB Partition Assigners
DB Stats and Analytics
3.Обновление владельцев партиций
SmartPartitioner Rest API
2. Обновление и проверка статистики по партициям
Периодически получает информацию о данных от брокера
1. Запрос партиций или отправка статистики
Kafka
Partition Domain GEO
geo geo geo
Partition Domain URL
url url url
Partition Domain CALL
call call call
Partition Domain OTHER
obj obj obj
Partition Domain N
obj obj obj
Равномерное распределение данных по партициям
Processor-1
Сложная обработка


DB Partition Assigners
DB Stats and Analytics
3.Обновление владельцев партиций
SmartPartitioner Rest API
2. Обновление и проверка статистики по партициям
Периодически получает информацию о данных от брокера
1. Запрос партиций или отправка статистики
4. Получение идентификатора и партиций
Kafka
Partition Domain GEO
geo geo geo
Partition Domain URL
url url url
Partition Domain CALL
call call call
Partition Domain OTHER
obj obj obj
Partition Domain N
obj obj obj
Processor-1
Сложная обработка
Равномерное распределение данных по партициям


DB Partition Assigners
DB Stats and Analytics
3.Обновление владельцев партиций
SmartPartitioner Rest API
2. Обновление и проверка статистики по партициям
Периодически получает информацию о данных от брокера
1. Запрос партиций или отправка статистики
4. Получение идентификатора и партиций
Kafka
Partition Domain GEO
geo geo geo
Partition Domain URL
url url url
Partition Domain CALL
call call call
Partition Domain OTHER
obj obj obj
Partition Domain N
obj obj obj
5. Подключение к партиции/ям
Processor-1
Сложная обработка
Равномерное распределение данных по партициям


DB Partition Assigners
DB Stats and Analytics
3.Обновление владельцев партиций
SmartPartitioner Rest API
2. Обновление и проверка статистики по партициям
Периодически получает информацию о данных от брокера
1. Запрос партиций или отправка статистики
4. Получение идентификатора и партиций
Kafka
Partition Domain GEO
geo geo geo
Partition Domain URL
url url url
Partition Domain CALL
call call call
Partition Domain OTHER
obj obj obj
Partition Domain N
obj obj obj
5. Подключение к партиции/ям
Processor-1
Сложная обработка
Processor-2
Простая обработка
Равномерное распределение данных по партициям


Пользователь
DB Partition Assigners
Ручное управление партициями
DB Stats and Analytics
3.Обновление владельцев партиций
SmartPartitioner Rest API
2. Обновление и проверка статистики по партициям
Периодически получает информацию о данных от брокера
1. Запрос партиций или отправка статистики
4. Получение идентификатора и партиций
Kafka
Partition Domain GEO
geo geo geo
Partition Domain URL
url url url
Partition Domain CALL
call call call
Partition Domain OTHER
obj obj obj
Partition Domain N
obj obj obj
5. Подключение к партиции/ям
Processor-1
Сложная обработка
Processor-2
Простая обработка
Равномерное распределение данных по партициям

Что даёт «умное» распределение партиций

- Эффективное распределение нагрузки
- Снижение количества ребалансов консьюмеров
- Управление чтением партиций
- Управление параметрами консьюмера без изменения кода

Промежуточные выводы

Промежуточные выводы

- Parallel Consumer способен повысить пропускную способность обработки данных, но он ещё «сырой» и использование его в production несёт риски

Промежуточные выводы

- Parallel Consumer способен повысить пропускную способность обработки данных, но он ещё «сырой» и использование его в production несёт риски
- Можно сделать собственное решение в вариациях Consumer + Reactive Framework, Consumer + Thread Pool. Что использовать, зависит от компетенций команды и решаемых задач.

Промежуточные выводы

- Parallel Consumer способен повысить пропускную способность обработки данных, но он ещё «сырой» и использование его в production несёт риски
- Можно сделать собственное решение в вариациях Consumer + Reactive Framework, Consumer + Thread Pool. Что использовать, зависит от компетенций команды и решаемых задач.
- Для эффективного распределения нагрузки можно использовать подход «умного» распределения партиций

Частые проблемы и их решения

Частые проблемы и их решения

О чём говорит:

```
TimeoutException: Expiring XXX record(s) for XXX due
to XXX ms has passed since batch creation plus
linger time
```

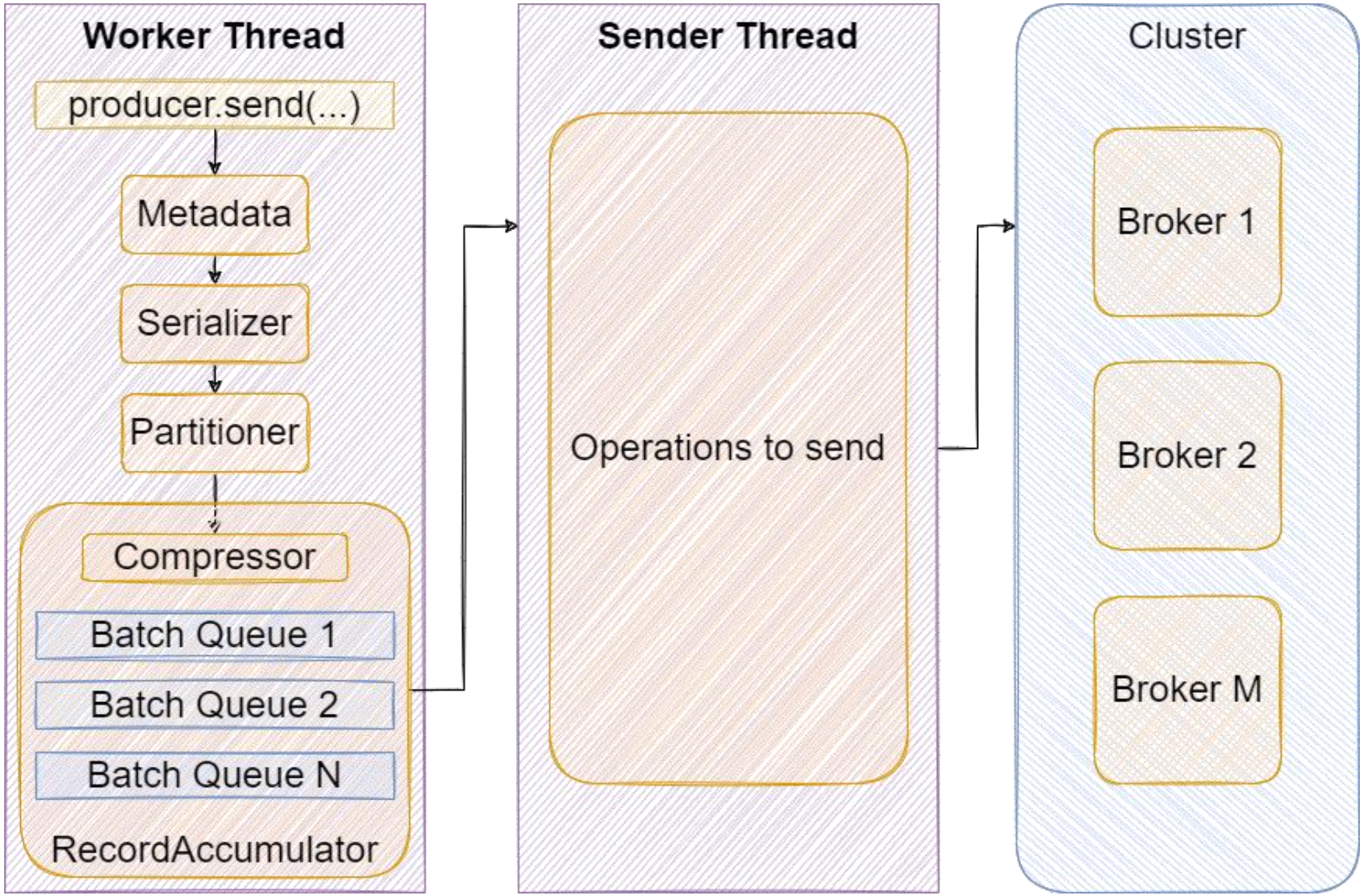
Частые проблемы и их решения

О чём говорит:

```
TimeoutException: Expiring XXX record(s) for XXX due
to XXX ms has passed since batch creation plus
linger time
```

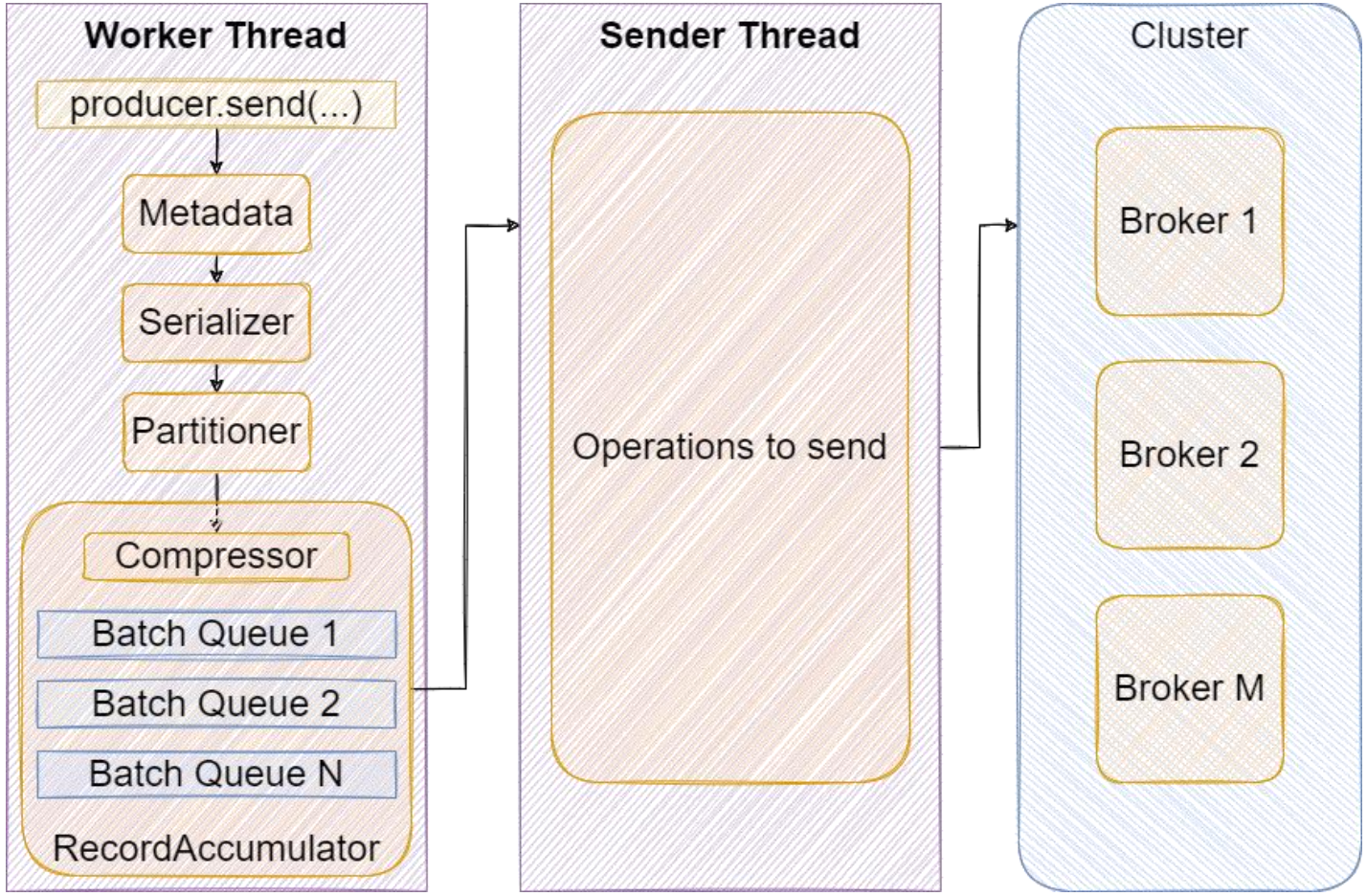
Григорий Кошелев — Когда всё пошло по Кафке: Producer

TimeoutException: Expiring XXX record(s) for XXX due to XXX ms has passed since batch creation plus linger time



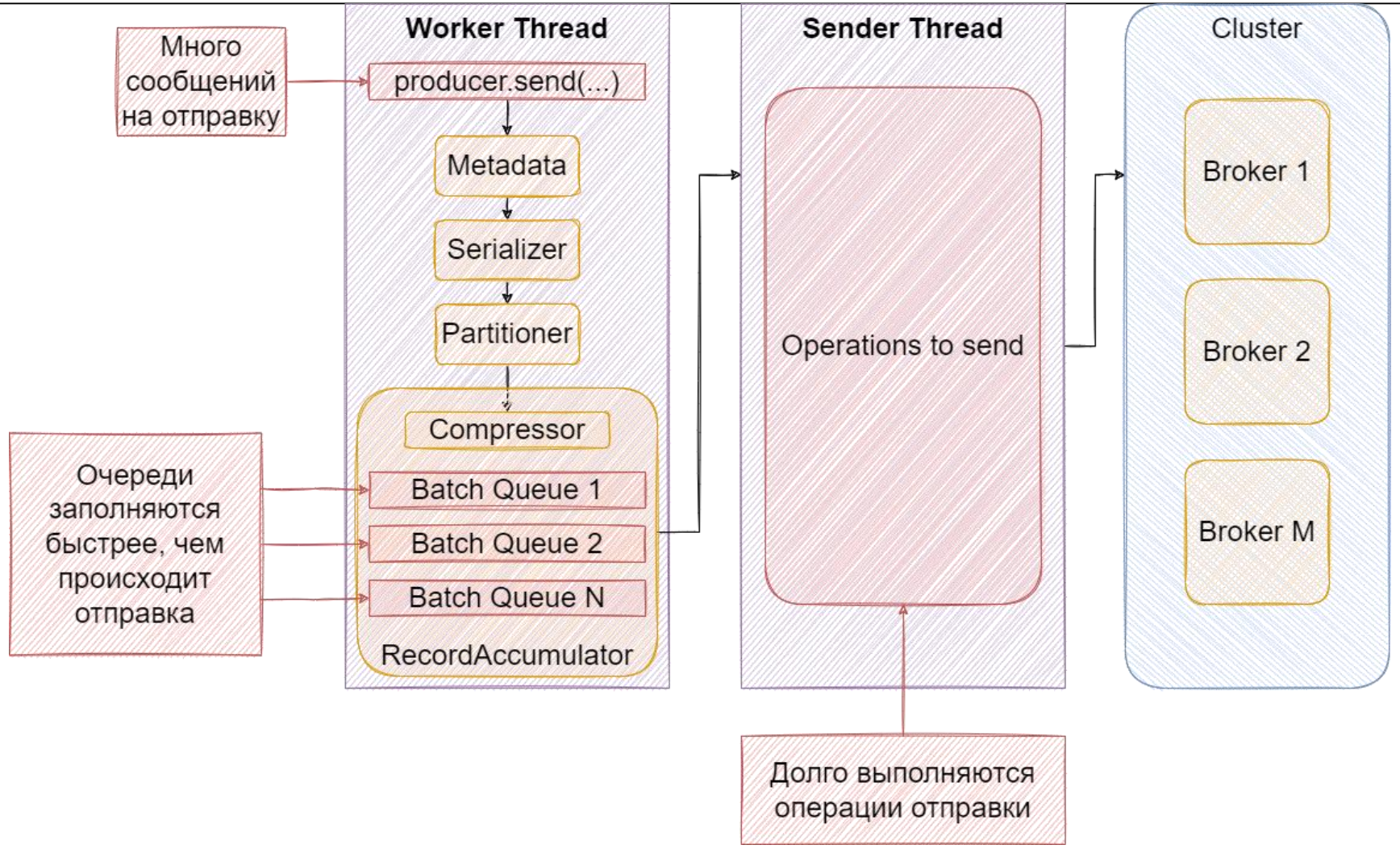
TimeoutException: Expiring XXX record(s) for XXX due to XXX ms has passed since batch creation plus linger time

delivery.timeout.ms >= (request.timeout.ms + linger.ms)



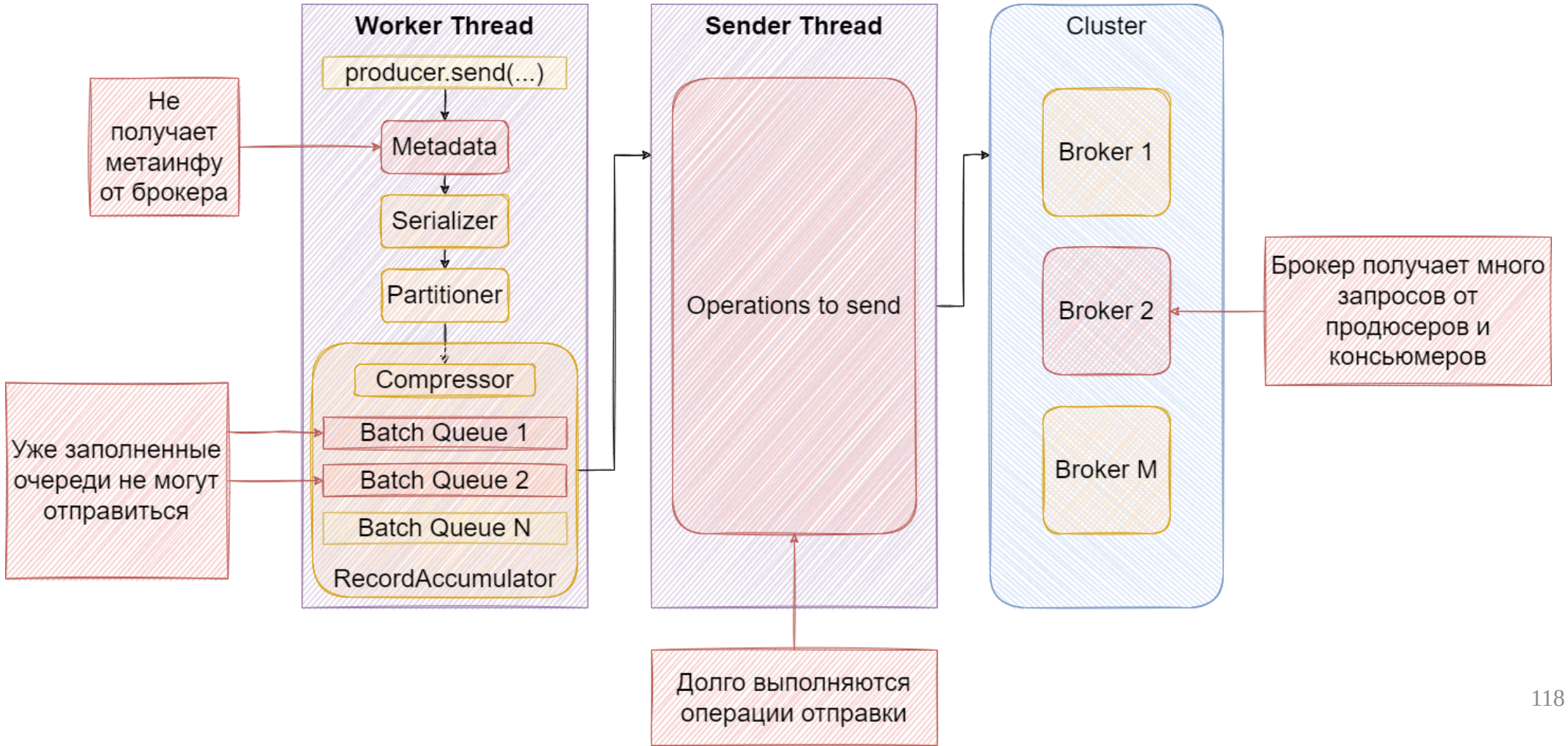
TimeoutException: Expiring XXX record(s) for XXX due to XXX ms has passed since batch creation plus linger time

1. Очень много сообщений на отправку



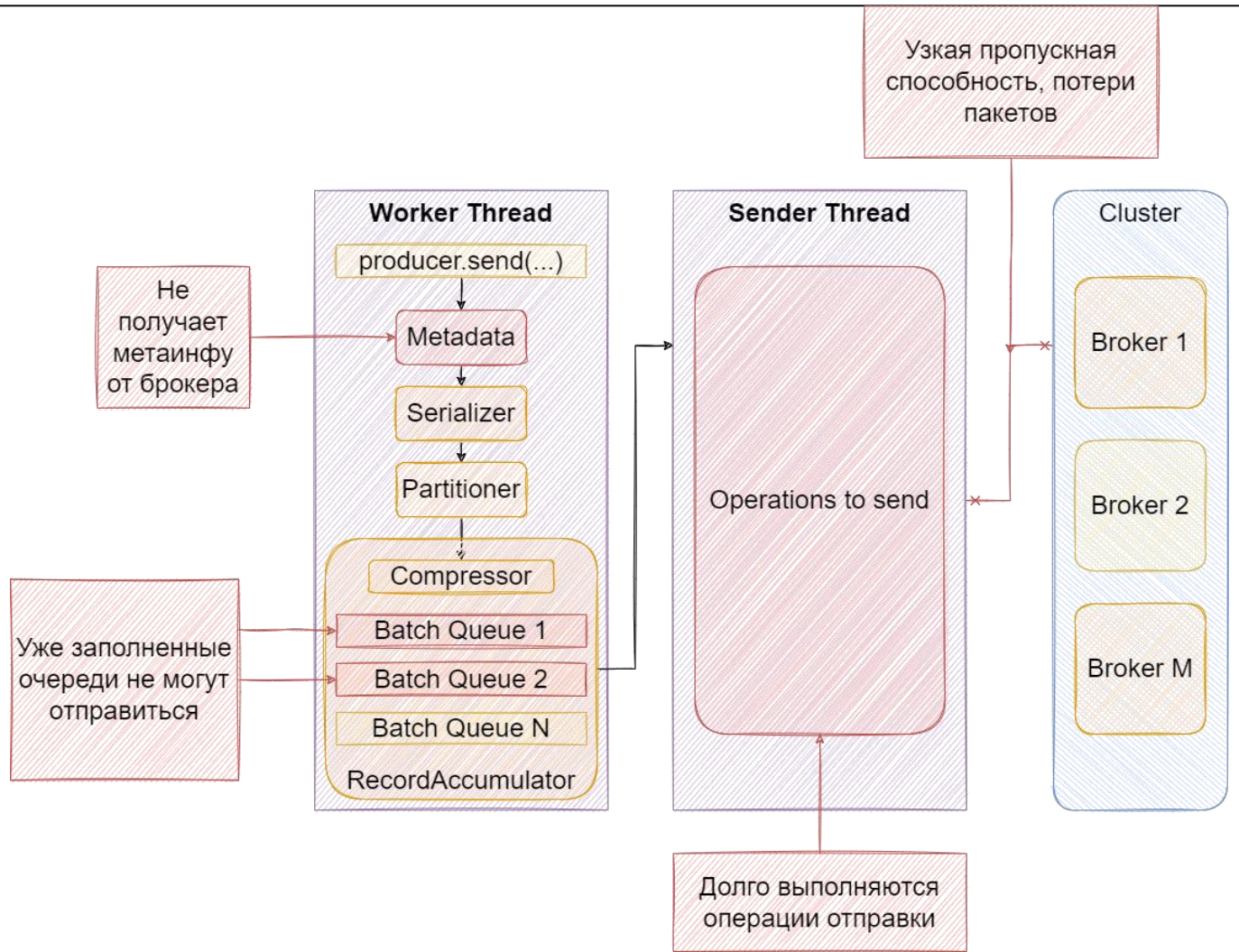
TimeoutException: Expiring XXX record(s) for XXX due to XXX ms has passed since batch creation plus linger time

2. Брокеру плохо от количества запросов от consumer и producer



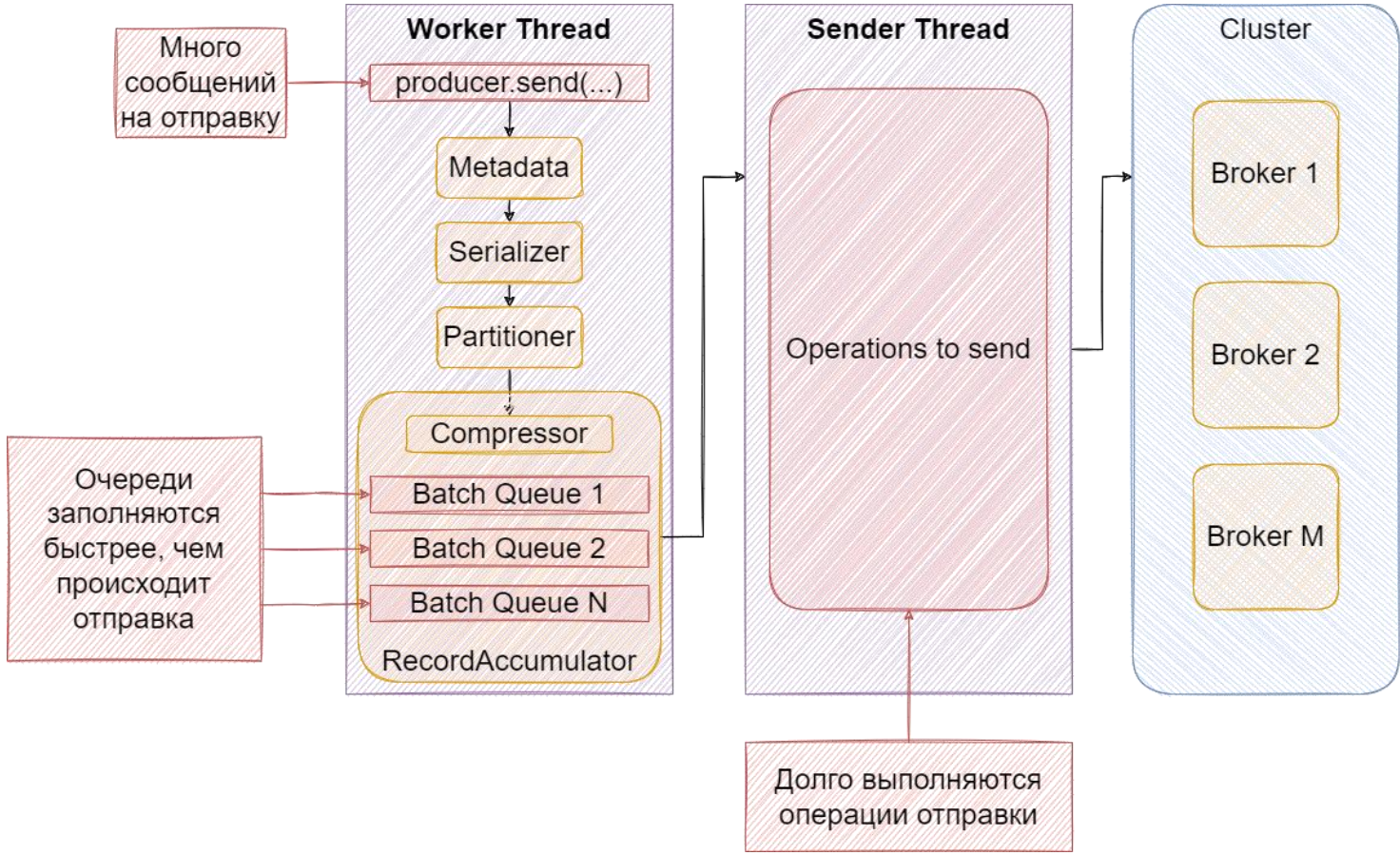
TimeoutException: Expiring XXX record(s) for XXX due to XXX ms has passed since batch creation plus linger time

3. Сетевая проблема – узкая пропускная способность, потери пакетов и т.п.



Решение проблемы: 1. Очень много сообщений на отправку

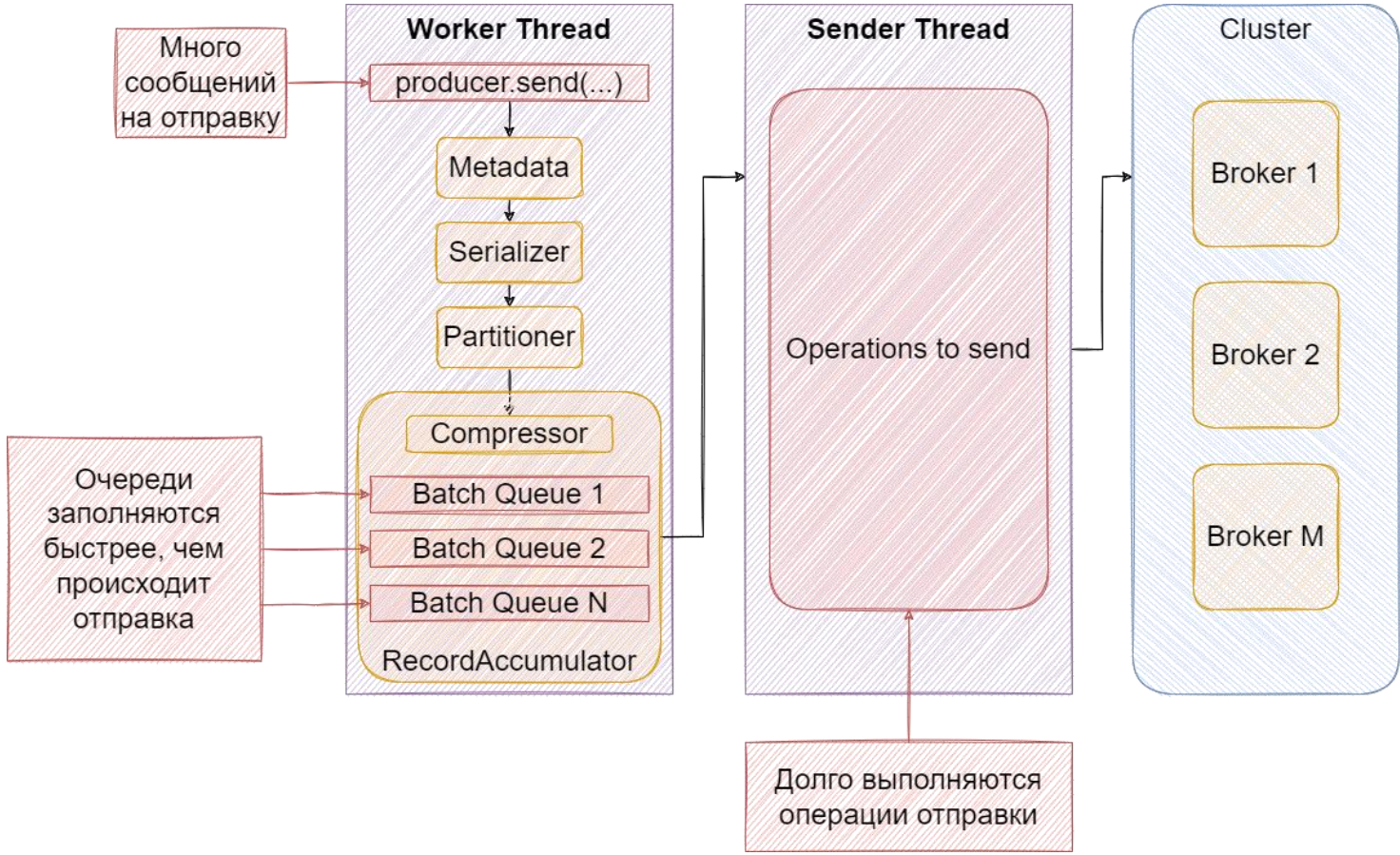
1. batch.size – увеличиваем (эмпирически)



Решение проблемы: 1. Очень много сообщений на отправку

1. batch.size – увеличиваем (эмпирически)

2. request.timeout.ms – увеличиваем (эмпирически)

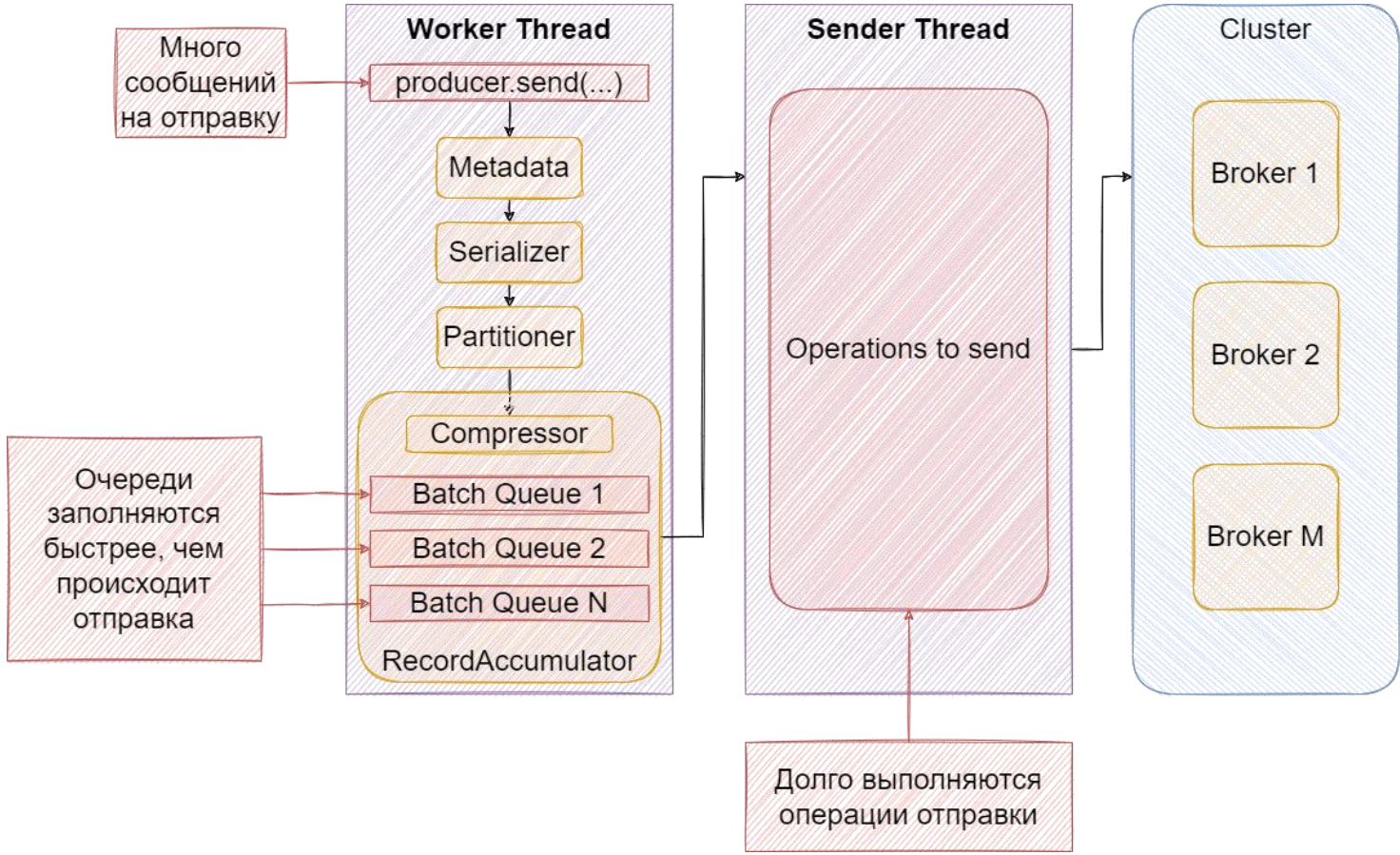


Решение проблемы: 1. Очень много сообщений на отправку

1. batch.size – увеличиваем (эмпирически)

2. request.timeout.ms – увеличиваем (эмпирически)

3. linger.ms = 0



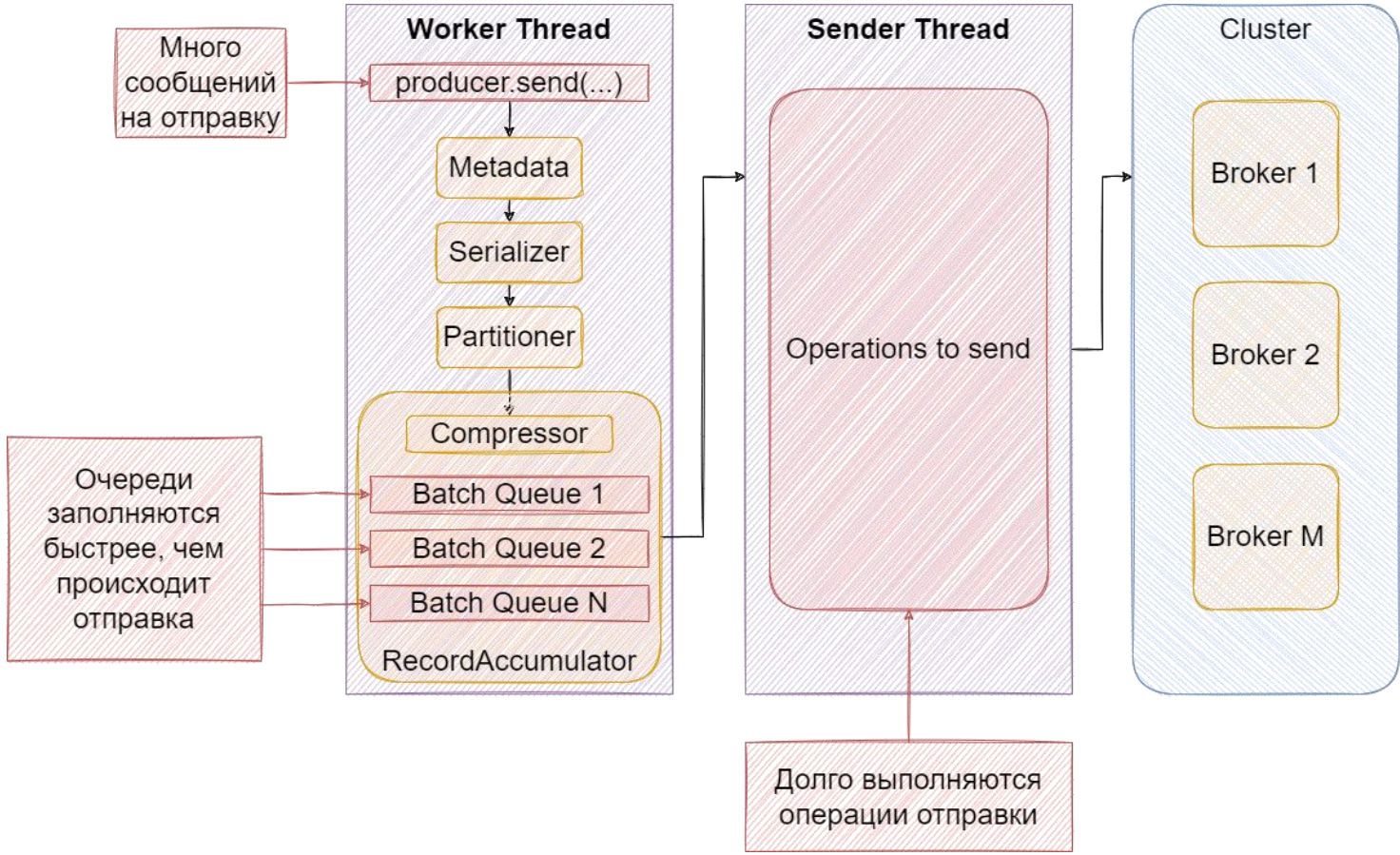
Решение проблемы: 1. Очень много сообщений на отправку

1. batch.size – увеличиваем (эмпирически)

2. request.timeout.ms – увеличиваем (эмпирически)

3. linger.ms = 0

4. acks = 0,1 (не all, если возможно)



Решение проблемы: 1. Очень много сообщений на отправку

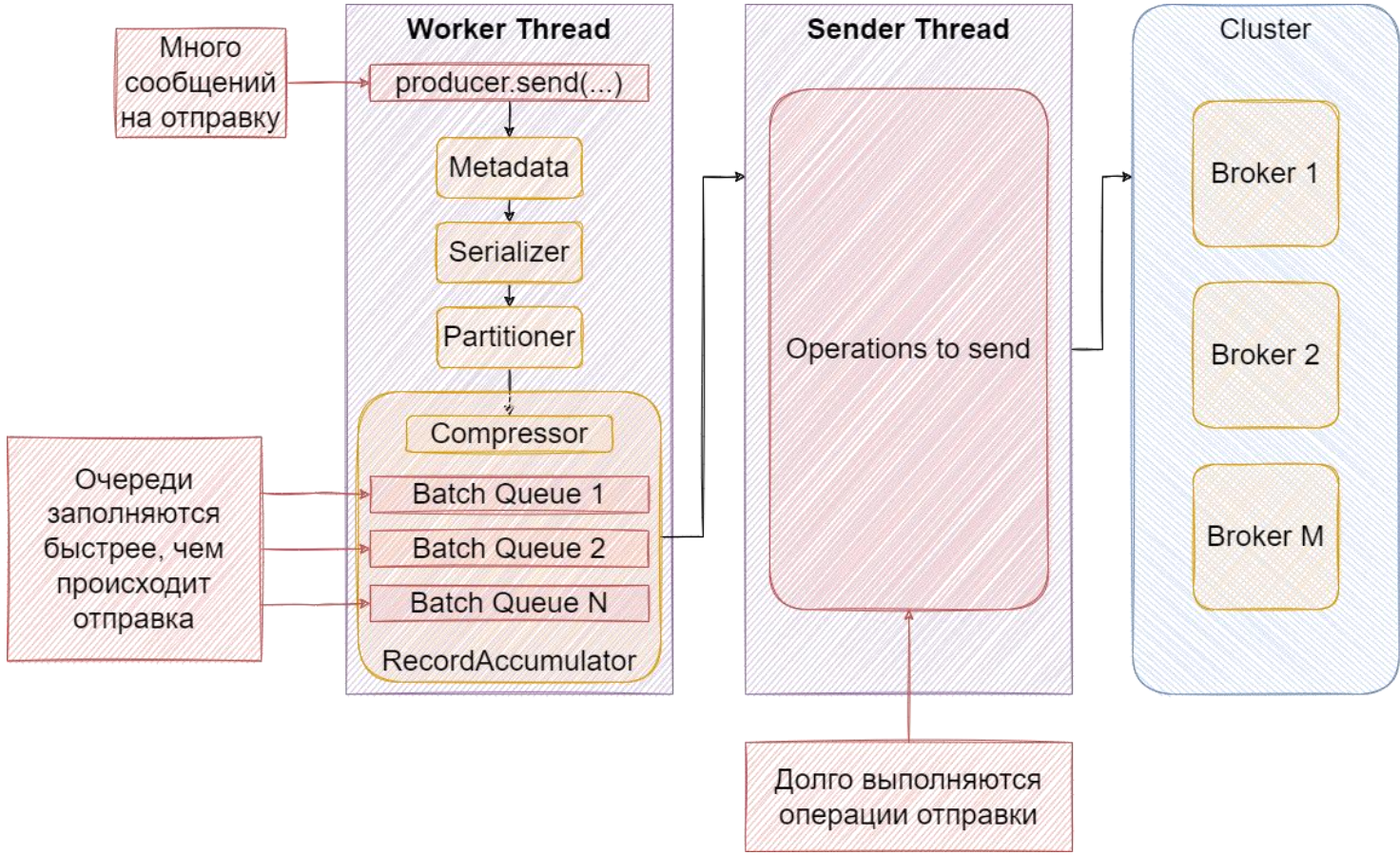
1. batch.size – увеличиваем (эмпирически)

2. request.timeout.ms – увеличиваем (эмпирически)

3. linger.ms = 0

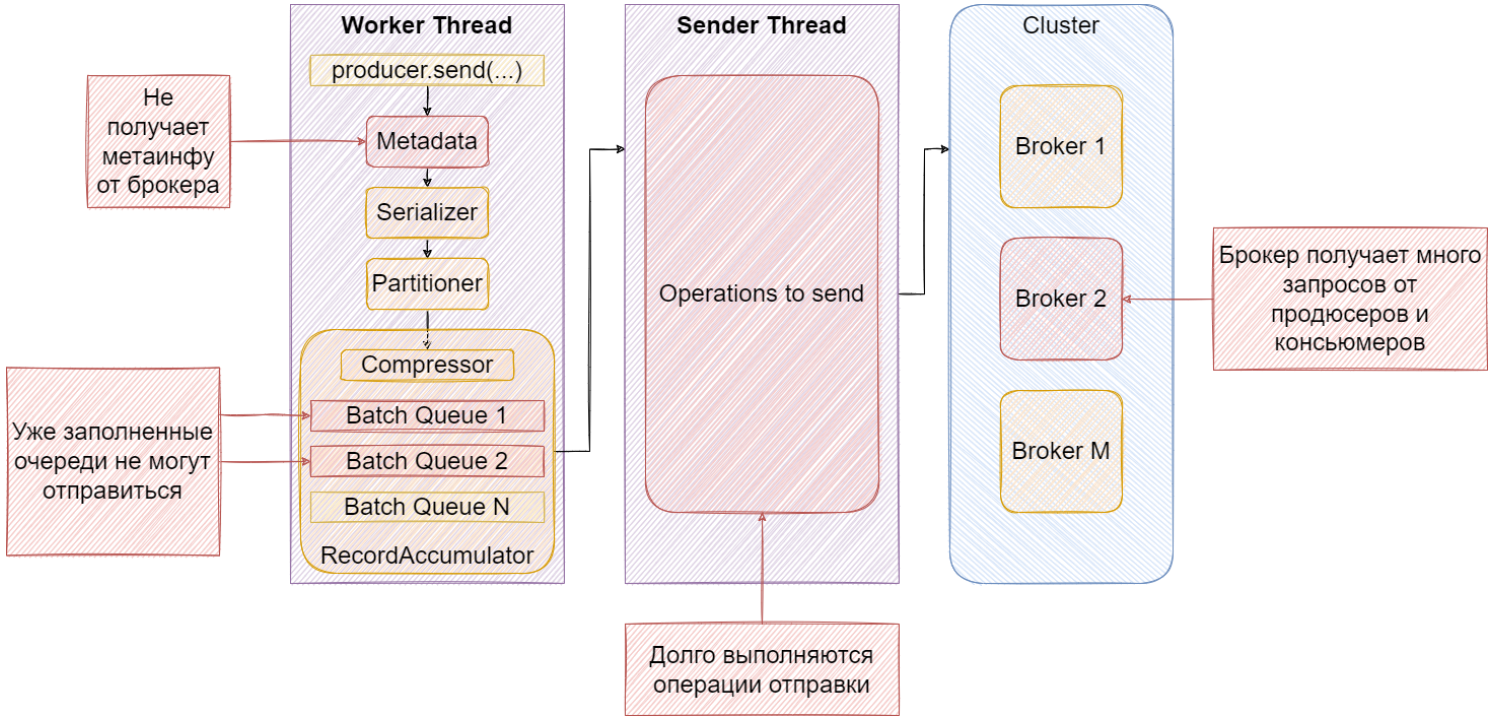
4. acks = 0,1 (не all, если возможно)

5. compression.type = snappy или lz4 (меньше тратить CPU)



Решение проблемы: 2. Брокеру плохо от количества запросов от consumer и producer

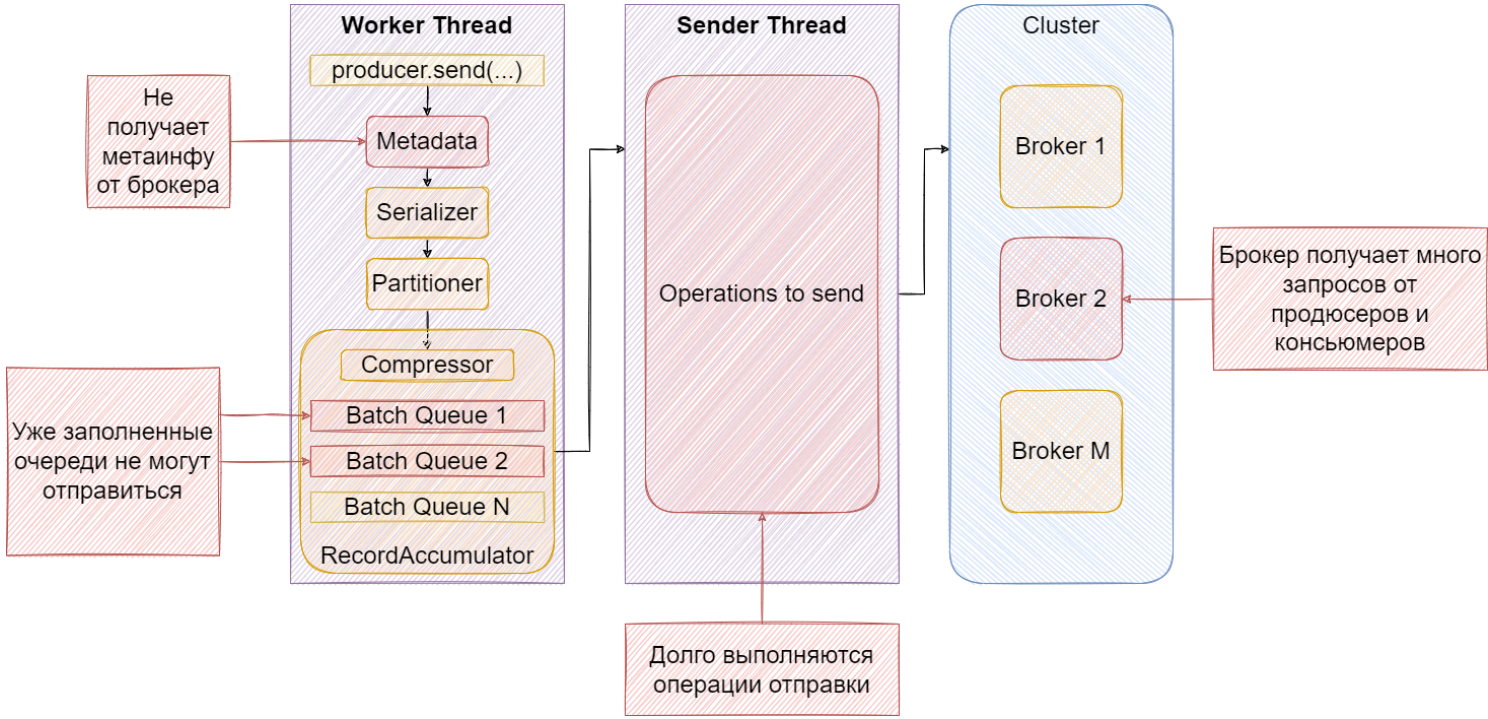
1. batch.size – увеличиваем (эмпирически)



Решение проблемы: 2. Брокеру плохо от количества запросов от consumer и producer

1. batch.size – увеличиваем (эмпирически)

2. request.timeout.ms – увеличиваем (эмпирически)

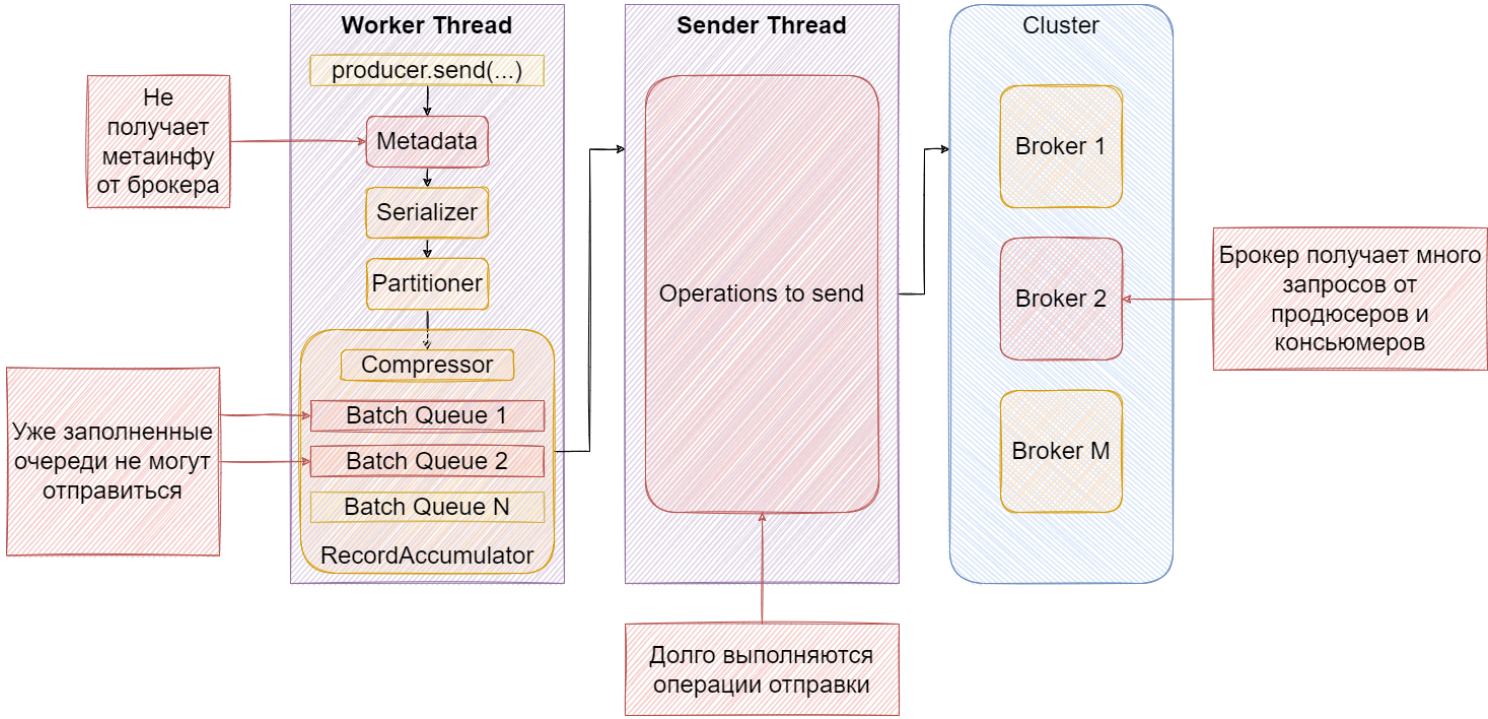


Решение проблемы: 2. Брокеру плохо от количества запросов от consumer и producer

1. batch.size – увеличиваем (эмпирически)

2. request.timeout.ms – увеличиваем (эмпирически)

3. buffer.memory – увеличиваем (эмпирически)



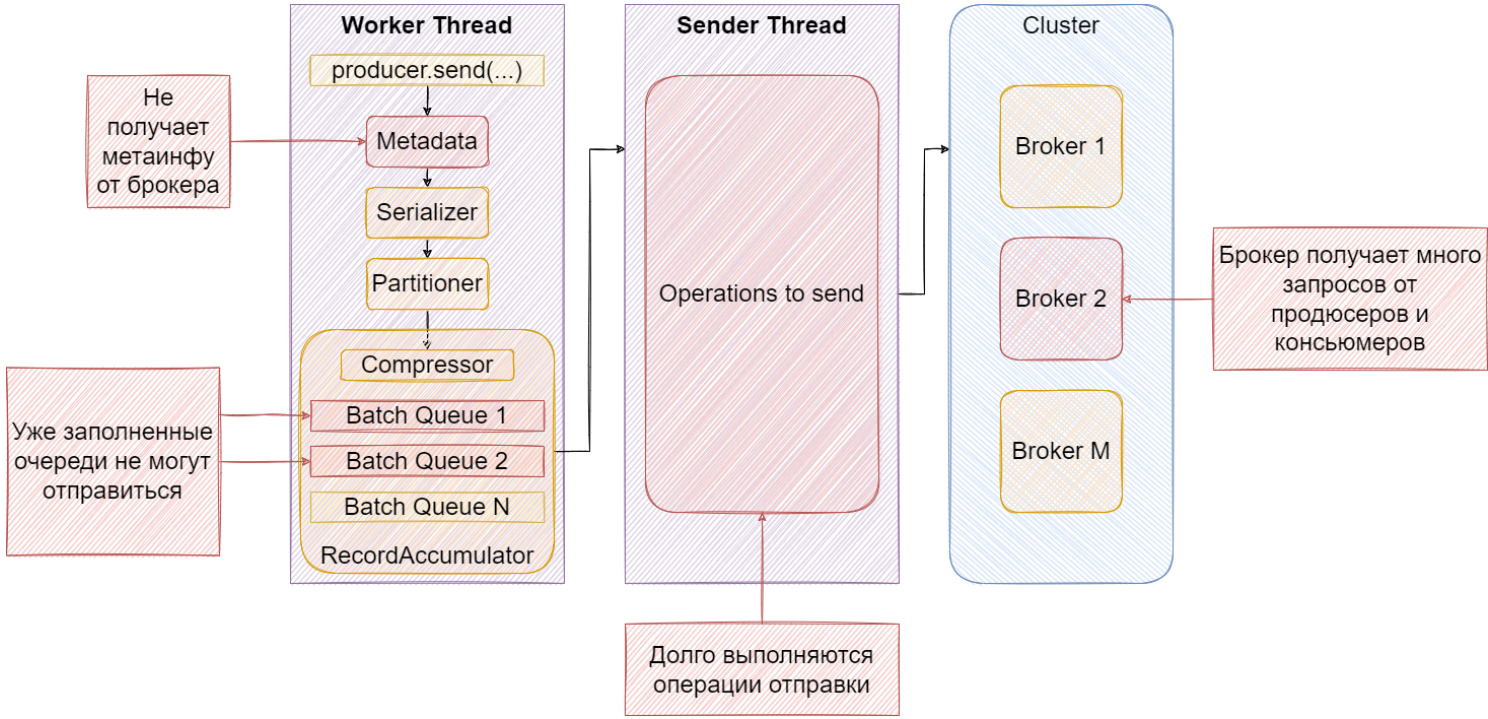
Решение проблемы: 2. Брокеру плохо от количества запросов от consumer и producer

1. batch.size – увеличиваем (эмпирически)

2. request.timeout.ms – увеличиваем (эмпирически)

3. buffer.memory – увеличиваем (эмпирически)

4. linger.ms – увеличиваем (эмпирически)



Решение проблемы: 2. Брокеру плохо от количества запросов от consumer и producer

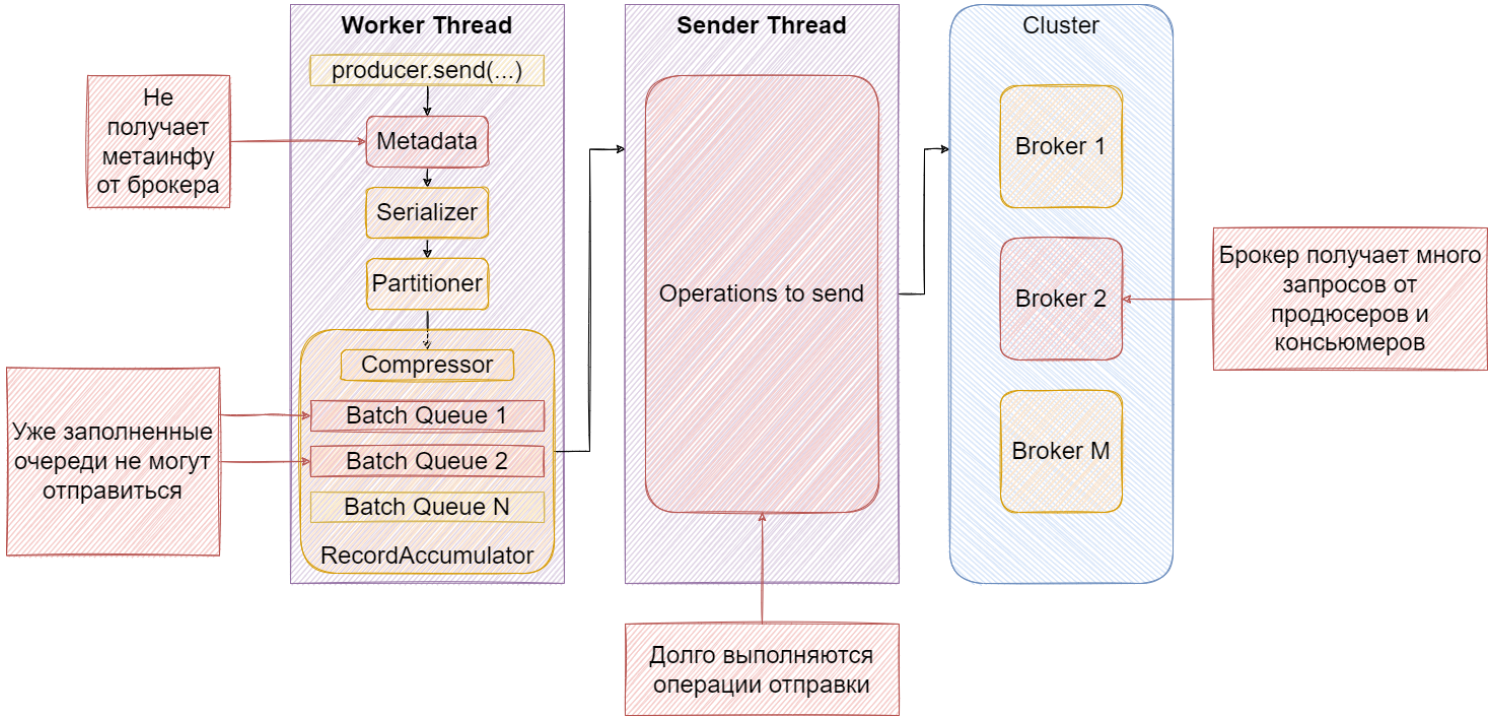
1. batch.size – увеличиваем (эмпирически)

2. request.timeout.ms – увеличиваем (эмпирически)

3. buffer.memory – увеличиваем (эмпирически)

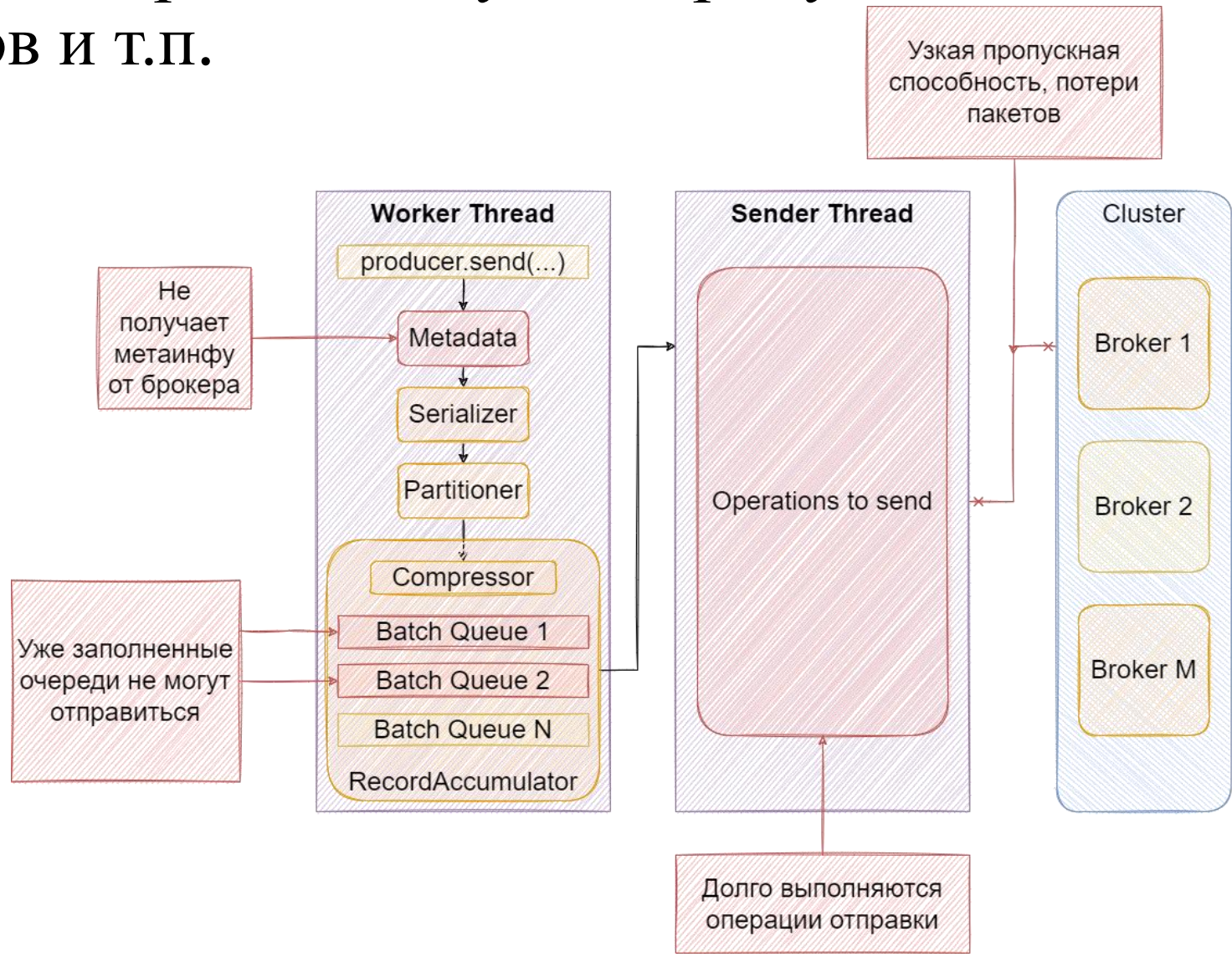
4. linger.ms – увеличиваем (эмпирически)

5. acks = 0,1 (не all, если возможно)



Решение проблемы: 3. Сетевая проблема – узкая пропускная способность, потери пакетов и т.п.

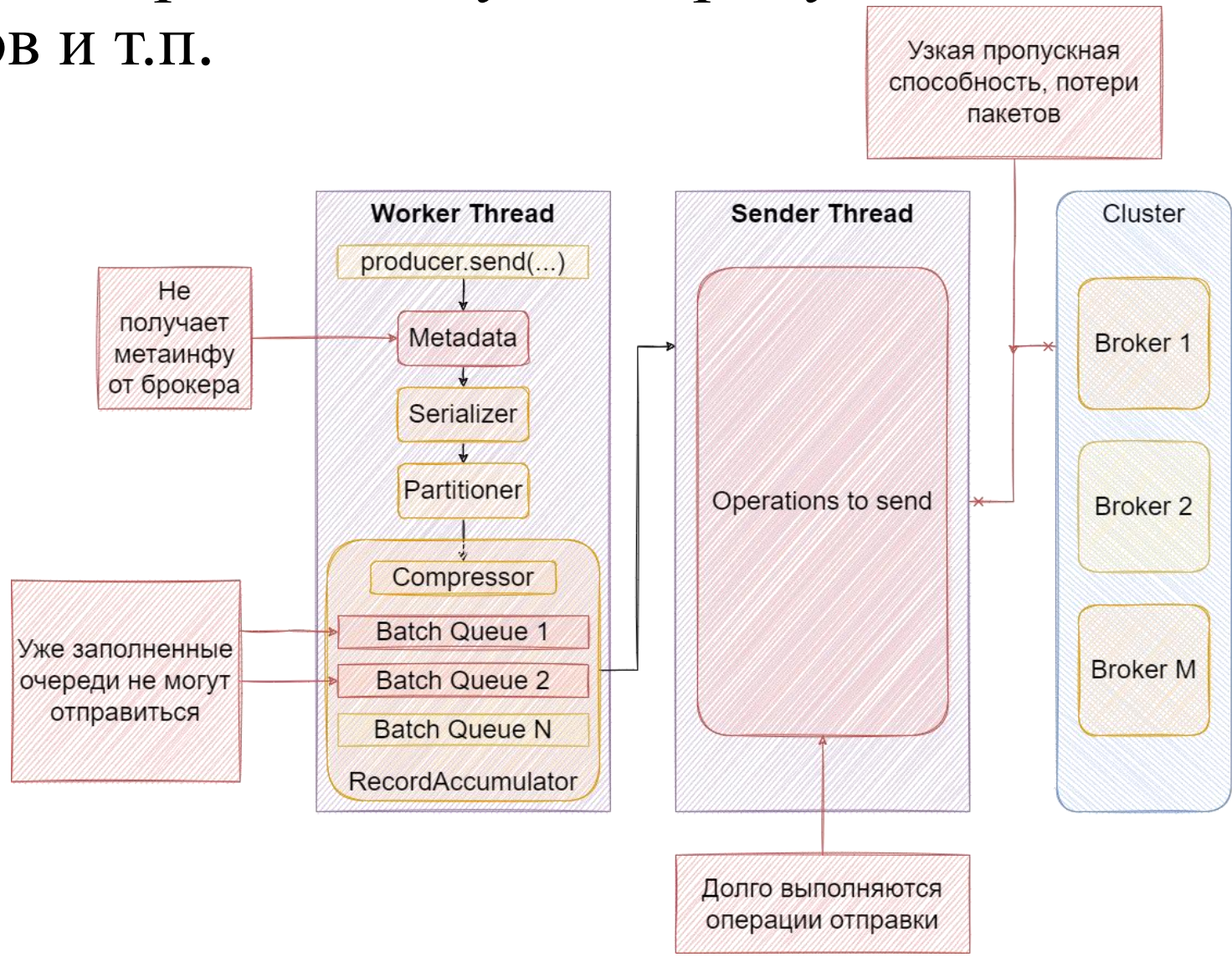
1. batch.size – уменьшаем (эмпирически)



Решение проблемы: 3. Сетевая проблема – узкая пропускная способность, потери пакетов и т.п.

1. batch.size – уменьшаем (эмпирически)

2. compression.type – максимальный тип сжатия

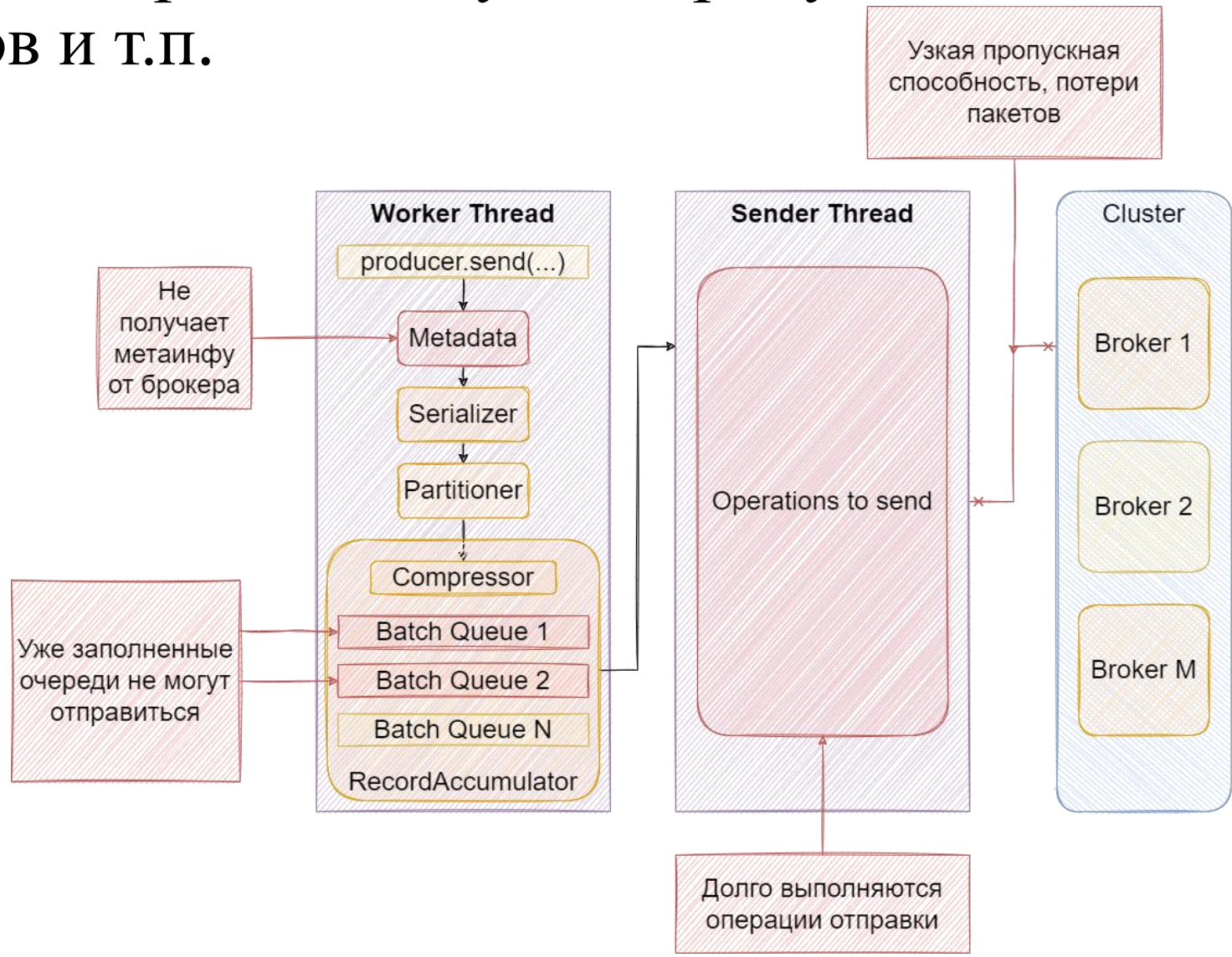


Решение проблемы: 3. Сетевая проблема – узкая пропускная способность, потери пакетов и т.п.

1. batch.size – уменьшаем (эмпирически)

2. compression.type – максимальный тип сжатия

3. Анализ сетевого трафика



О чём говорит:

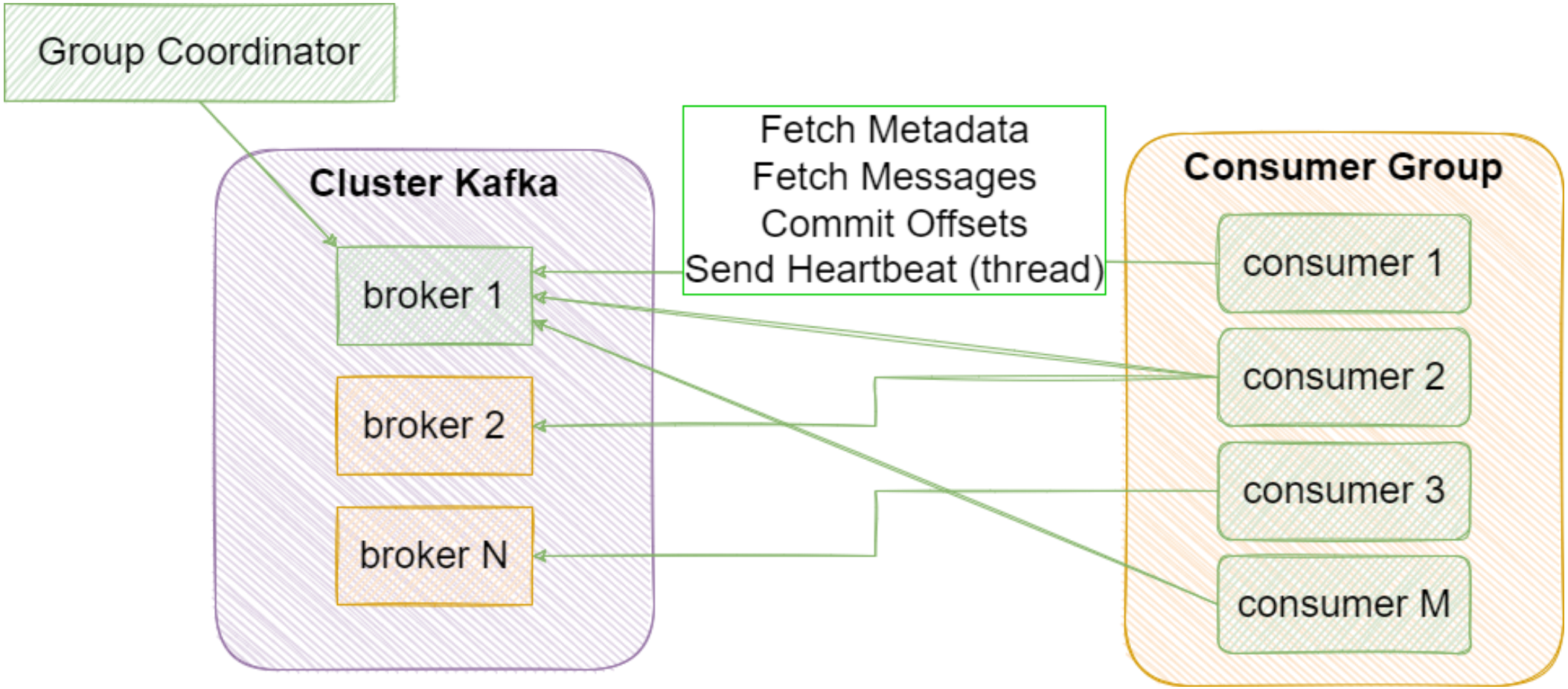
CommitFailedException: Offset commit cannot be completed since the consumer is not part of an active group for auto partition assignment

О чём говорит:

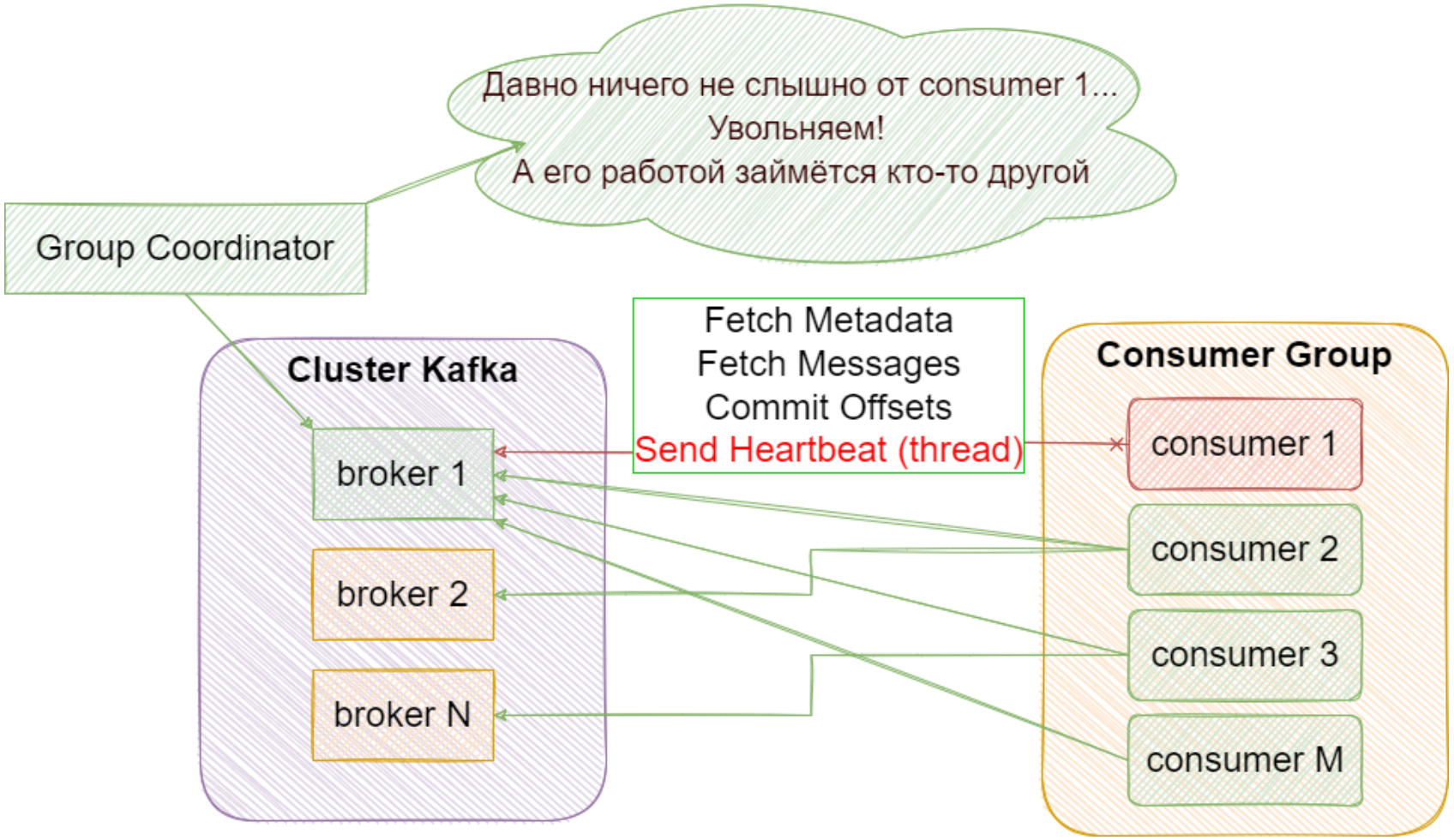
```
CommitFailedException: Offset commit cannot be
completed since the consumer is not part of an
active group for auto partition assignment
```

Григорий Кошелев — Когда всё пошло по Кафке Consumer

CommitFailedException: Offset commit cannot be completed since the consumer is not part of an active group for auto partition assignment

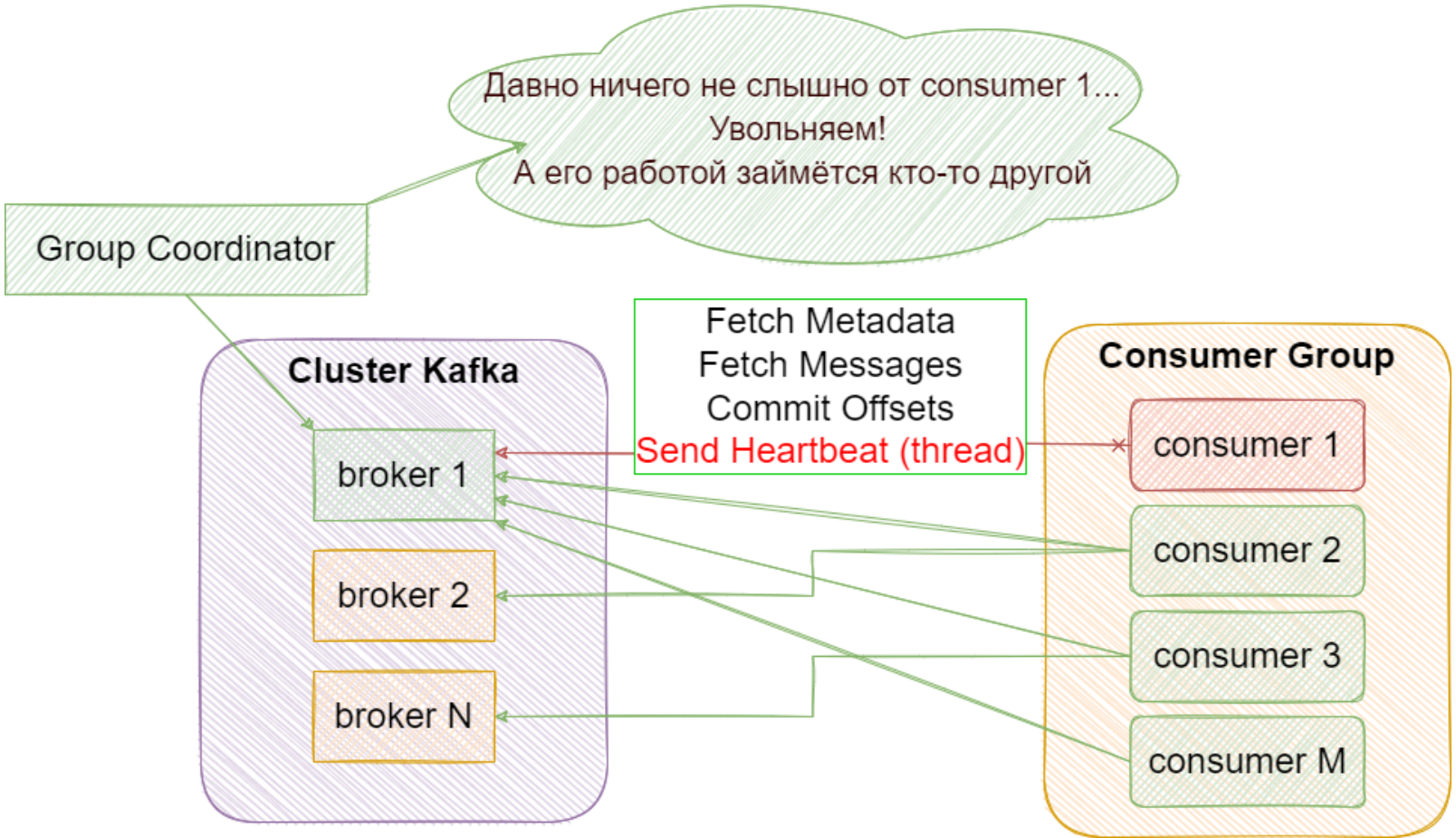


CommitFailedException: Offset commit cannot be completed since the consumer is not part of an active group for auto partition assignment



CommitFailedException: Offset commit cannot be completed since the consumer is not part of an active group for auto partition assignment

1. Долгие паузы на сборку мусора



CommitFailedException: Offset commit cannot be completed since the consumer is not part of an active group for auto partition assignment

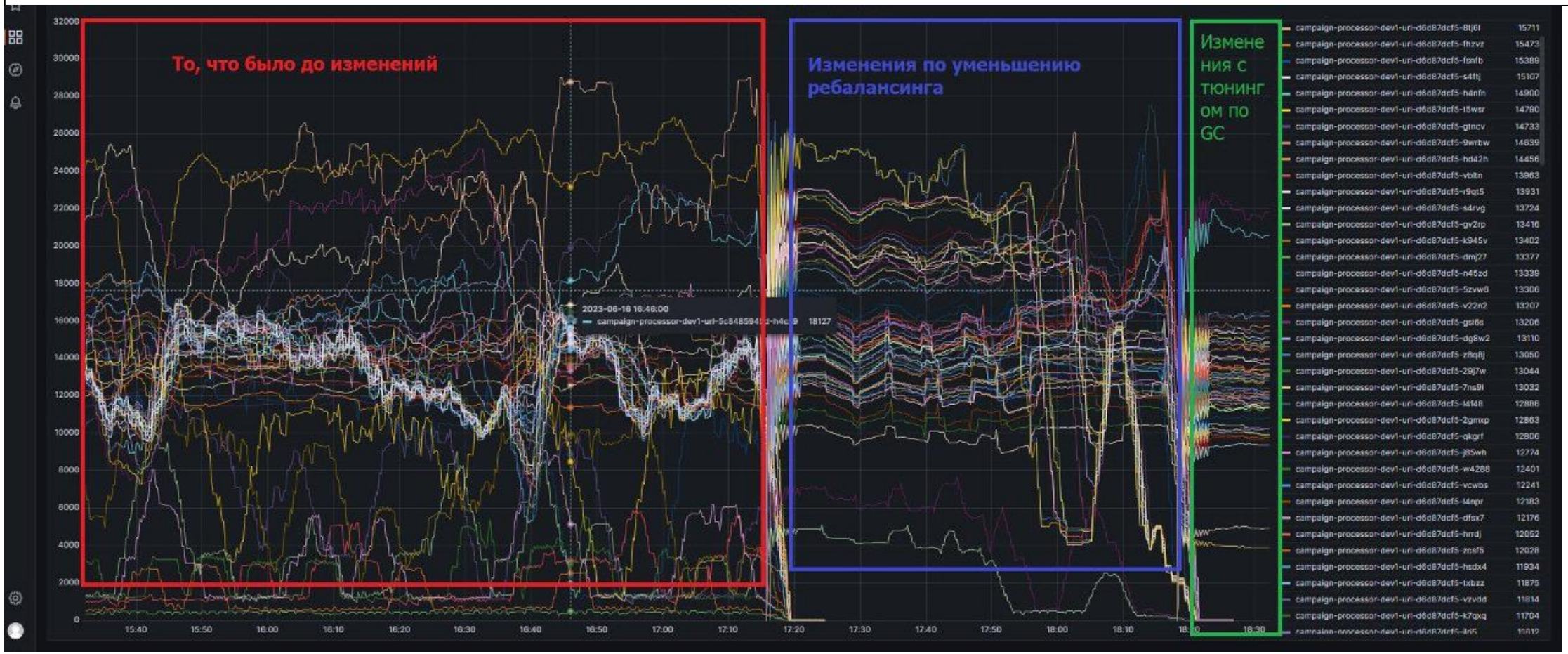
1. Долгие паузы на сборку мусора

Статьи о выборе GC для разных типов задач и их параметры:

- **How to choose the best Java garbage collector**
- **Make sure to choose the right GC for your Java application to achieve maximum performance**

CommitFailedException: Offset commit cannot be completed since the consumer is not part of an active group for auto partition assignment

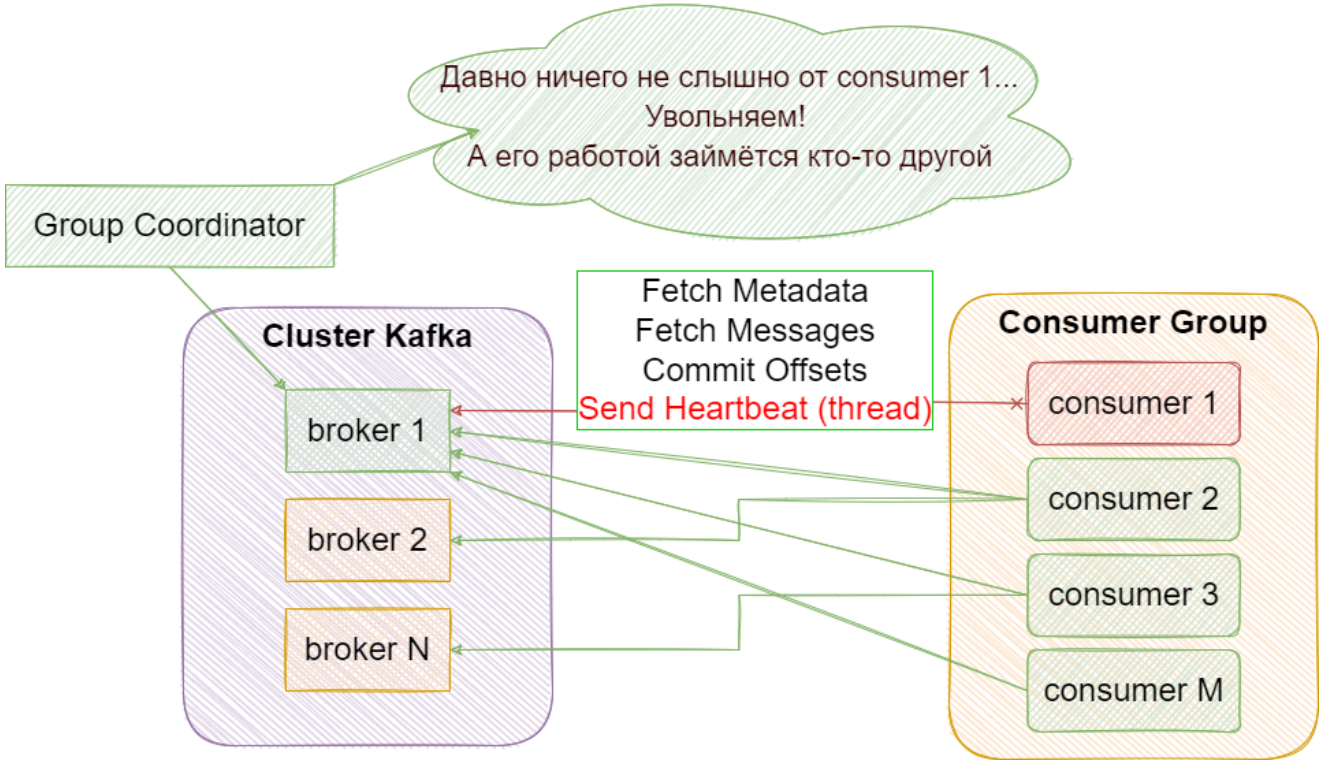
1. Долгие паузы на сборку мусора



CommitFailedException: Offset commit cannot be completed since the consumer is not part of an active group for auto partition assignment

Решение проблемы с долгими паузами на сборку мусора

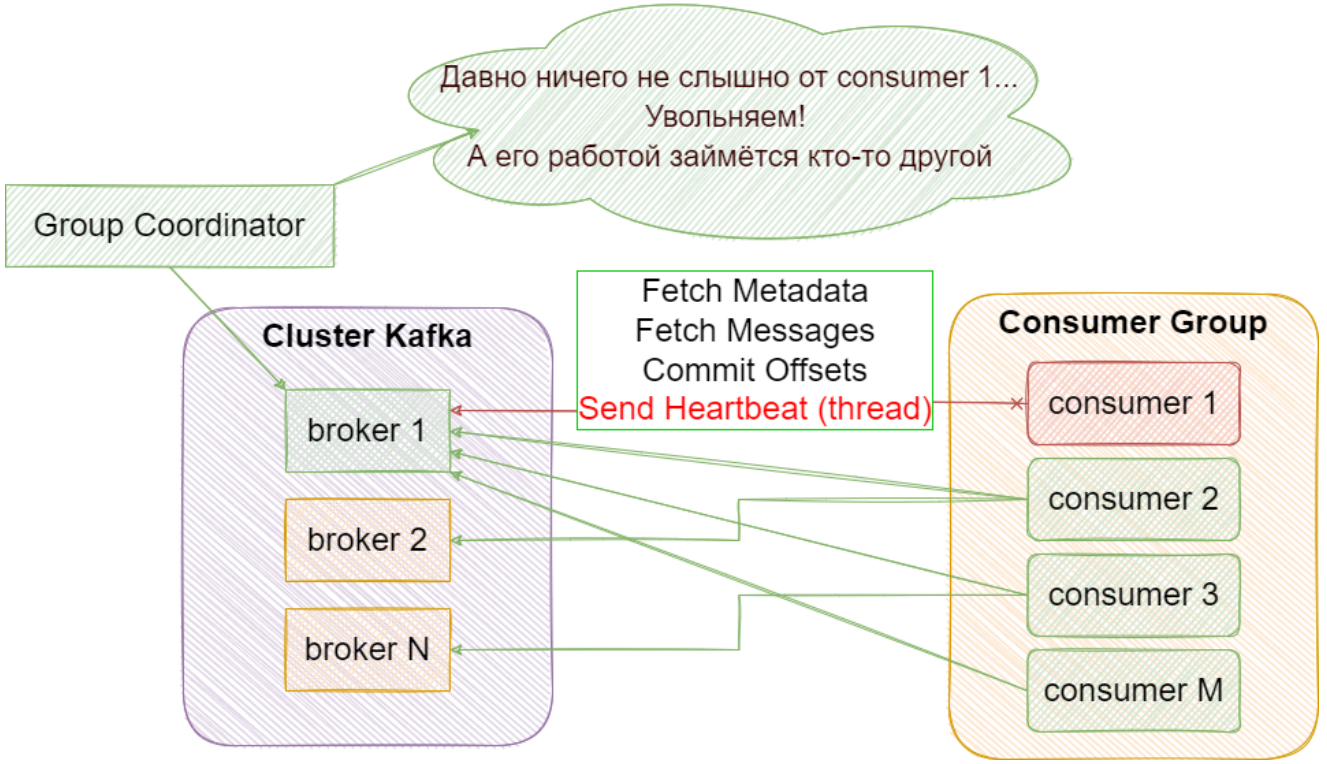
1. Пробуем другие типы GC (ParallelGC, ZGC)



CommitFailedException: Offset commit cannot be completed since the consumer is not part of an active group for auto partition assignment

Решение проблемы с долгими паузами на сборку мусора

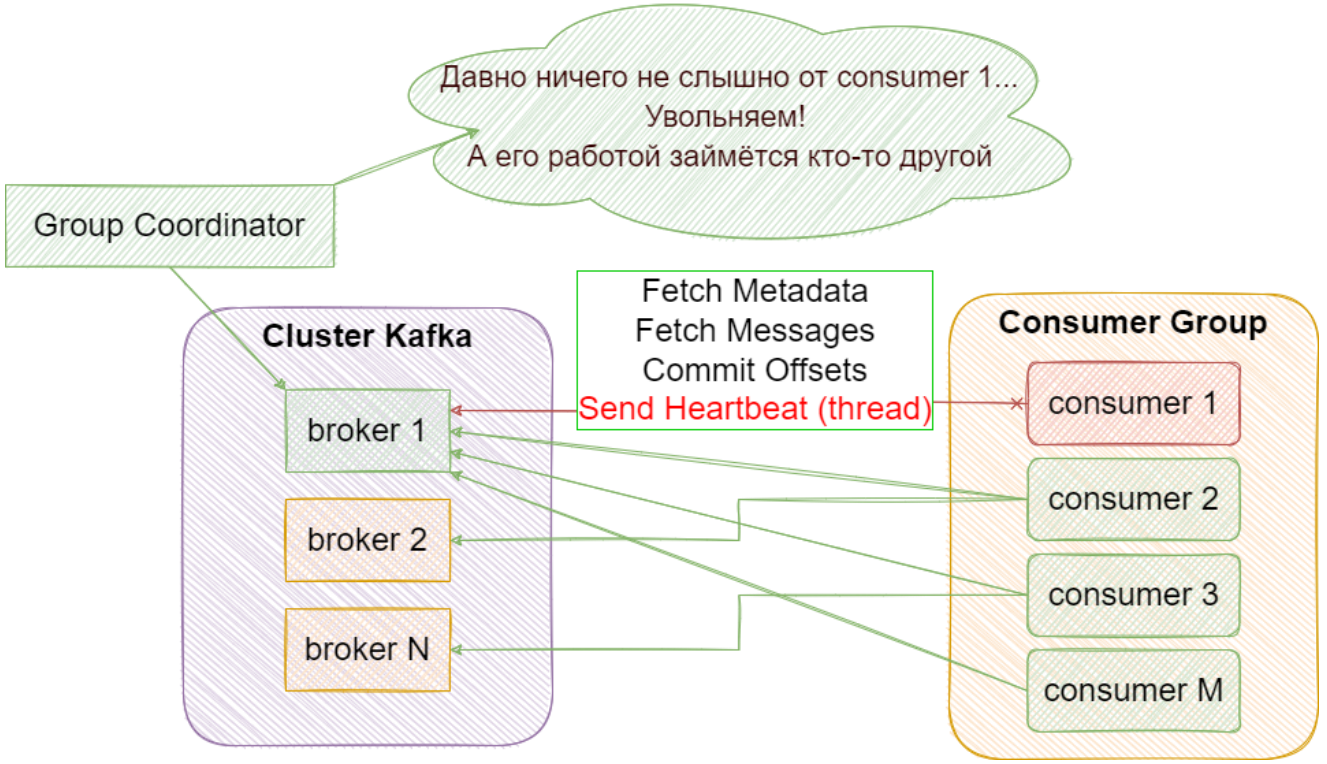
1. Пробуем другие типы GC (ParallelGC, ZGC)
2. max.poll.interval – увеличиваем (эмпирически)



CommitFailedException: Offset commit cannot be completed since the consumer is not part of an active group for auto partition assignment

Решение проблемы с долгими паузами на сборку мусора

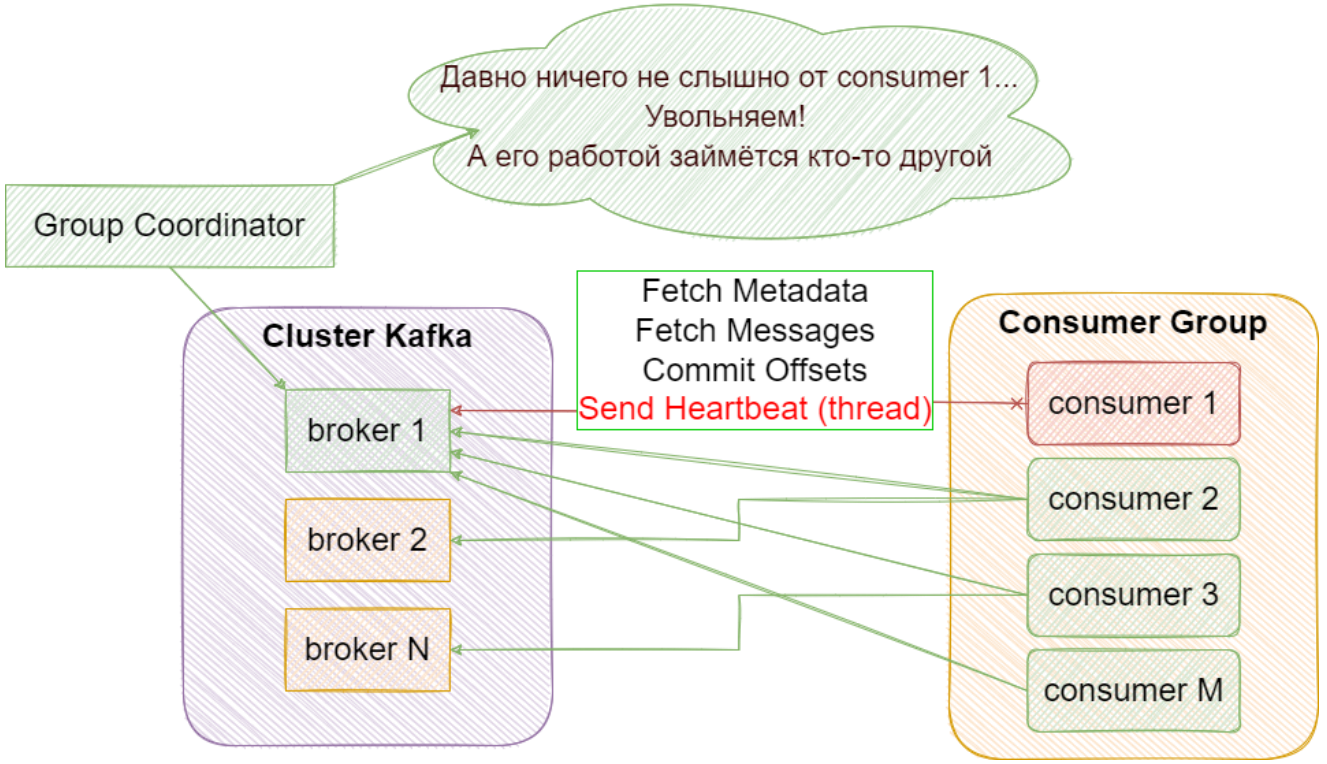
1. Пробуем другие типы GC (ParallelGC, ZGC)
2. max.poll.interval – увеличиваем (эмпирически)
3. session.timeout.ms – увеличиваем (эмпирически)



CommitFailedException: Offset commit cannot be completed since the consumer is not part of an active group for auto partition assignment

Решение проблемы с долгими паузами на сборку мусора

1. Пробуем другие типы GC (ParallelGC, ZGC)
2. max.poll.interval – увеличиваем (эмпирически)
3. session.timeout.ms – увеличиваем (эмпирически)
4. fetch.min.bytes – уменьшаем (эмпирически)



О чём говорит:

```
org.apache.clients.NetworkClient Disconnecting from node
XXX due to request timeout
```

О чём говорит:

`org.apache.clients.NetworkClient Disconnecting from node XXX due to request timeout`

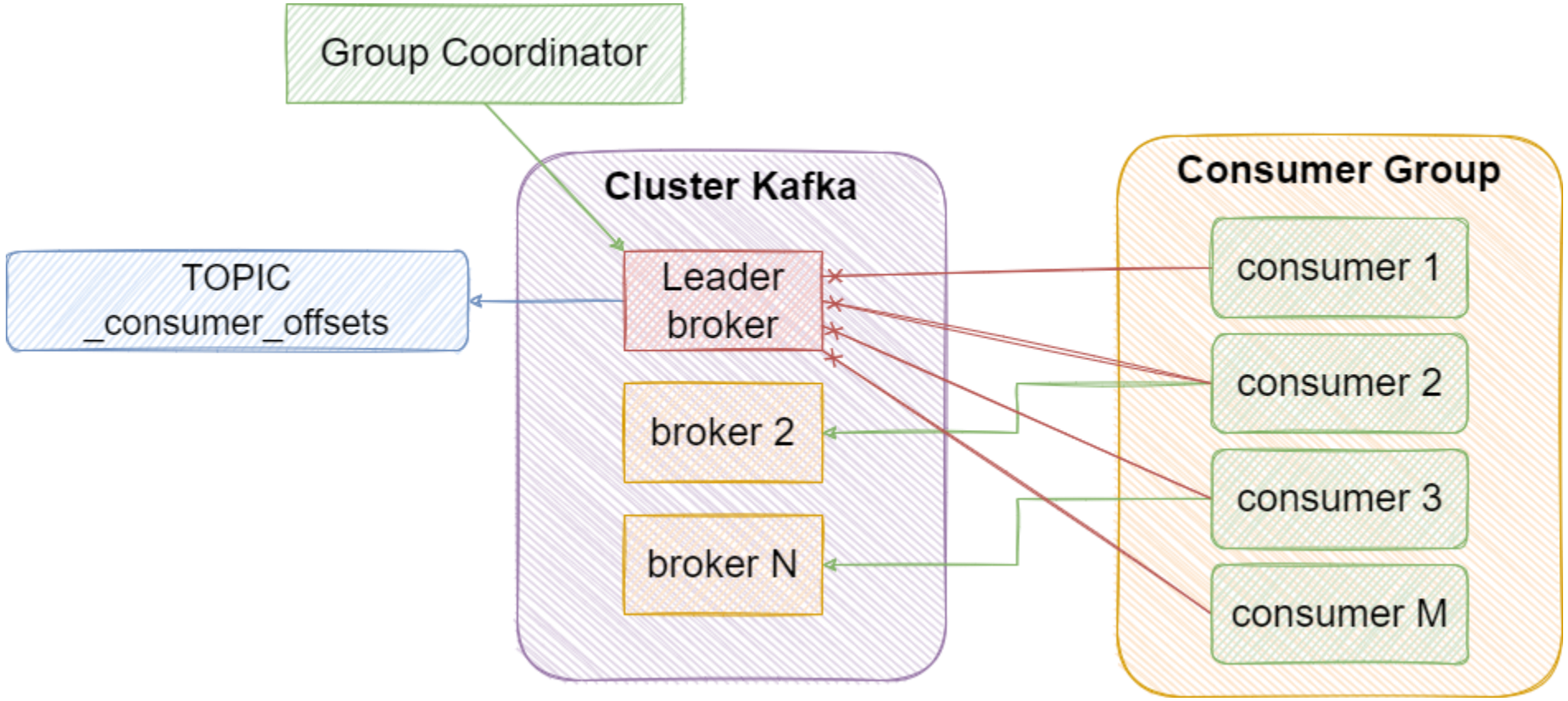
Pull Request - KAFKA-14317: ProduceRequest timeouts are logged as network exceptions

request.timeout.ms – увеличиваем (эмпирически)

О чём говорит:

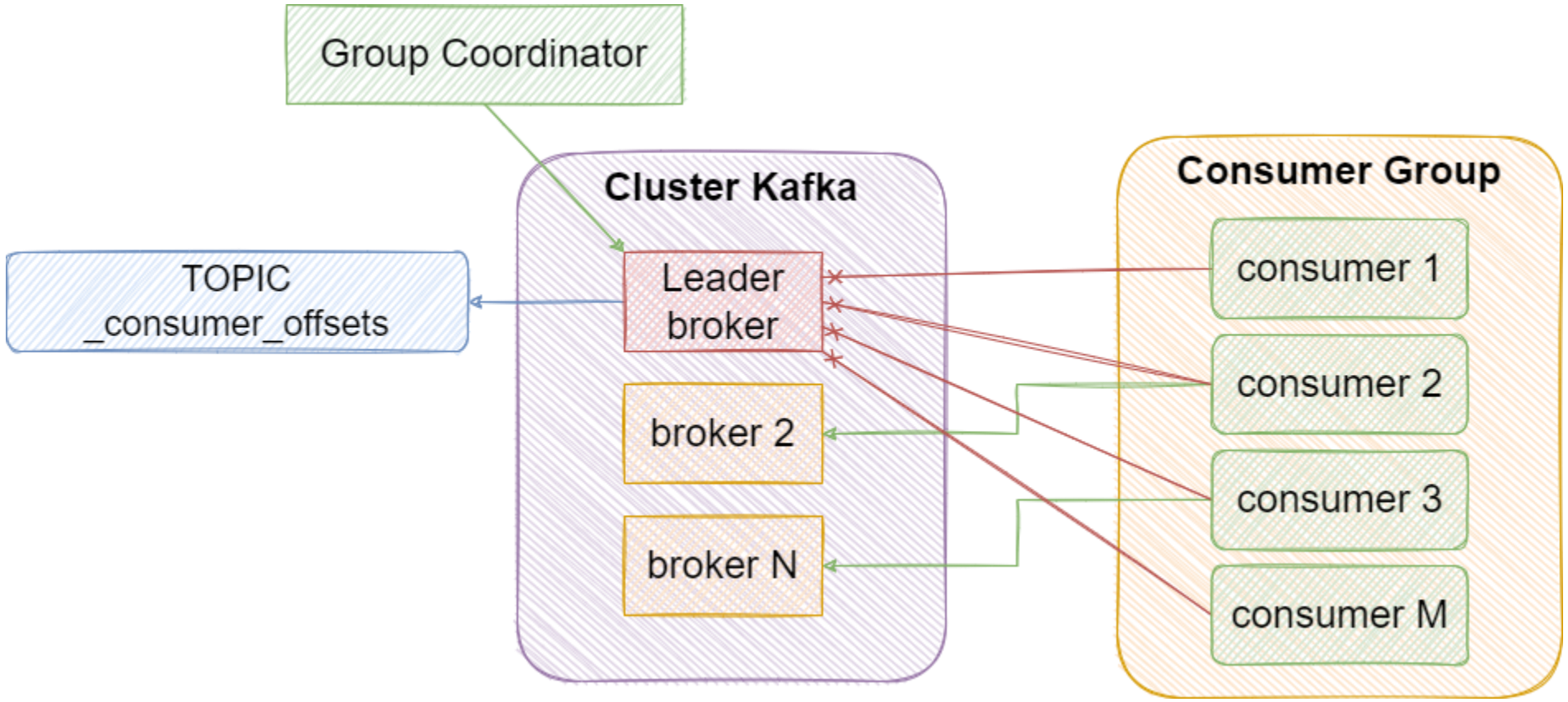
```
org.apache.kafka.common.errors.GroupCoordinatorNotAvailableException: The group coordinator is not available
```

GroupCoordinatorNotAvailableException: The group coordinator is not available



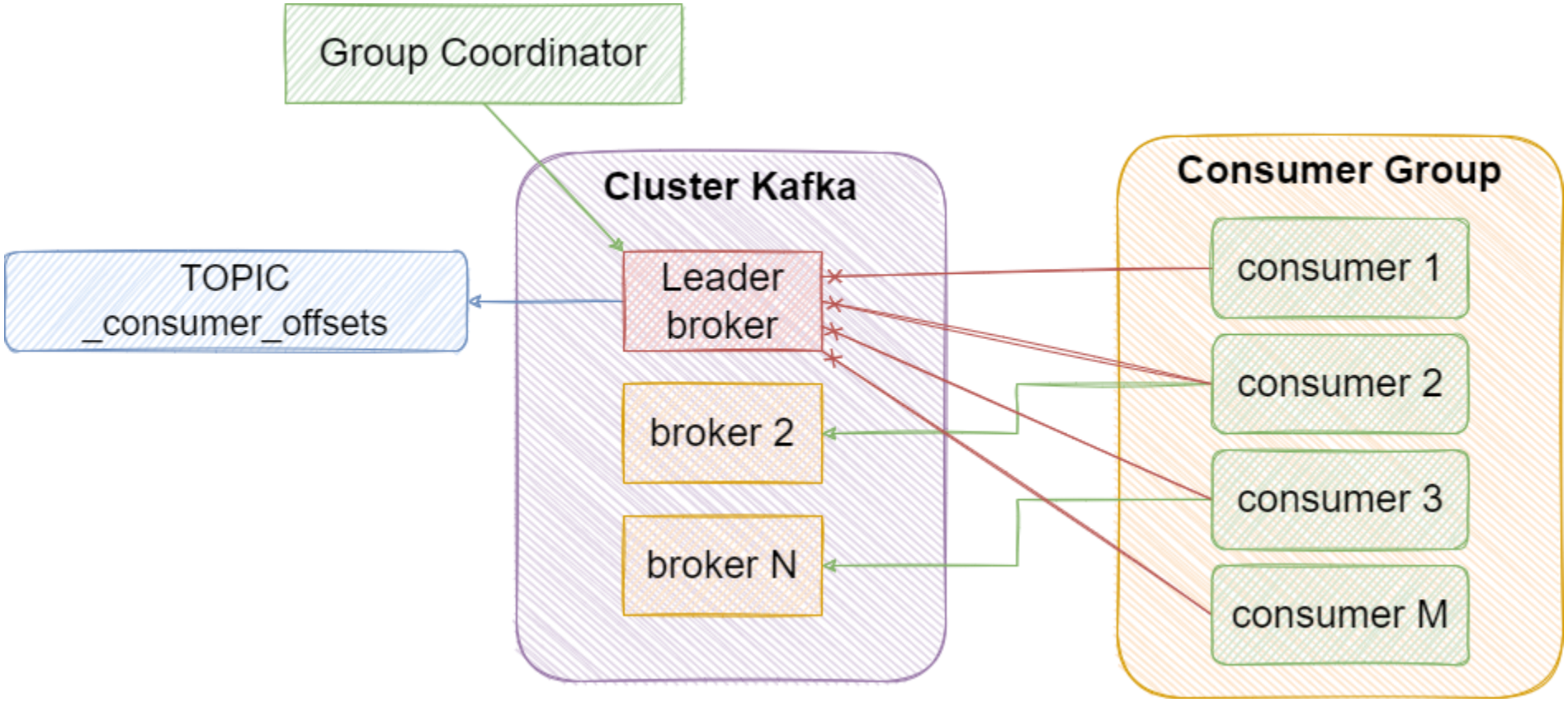
GroupCoordinatorNotAvailableException: The group coordinator is not available

1. Машине, на которой живёт брокер, стало плохо, брокер вовремя не отвечает



GroupCoordinatorNotAvailableException: The group coordinator is not available

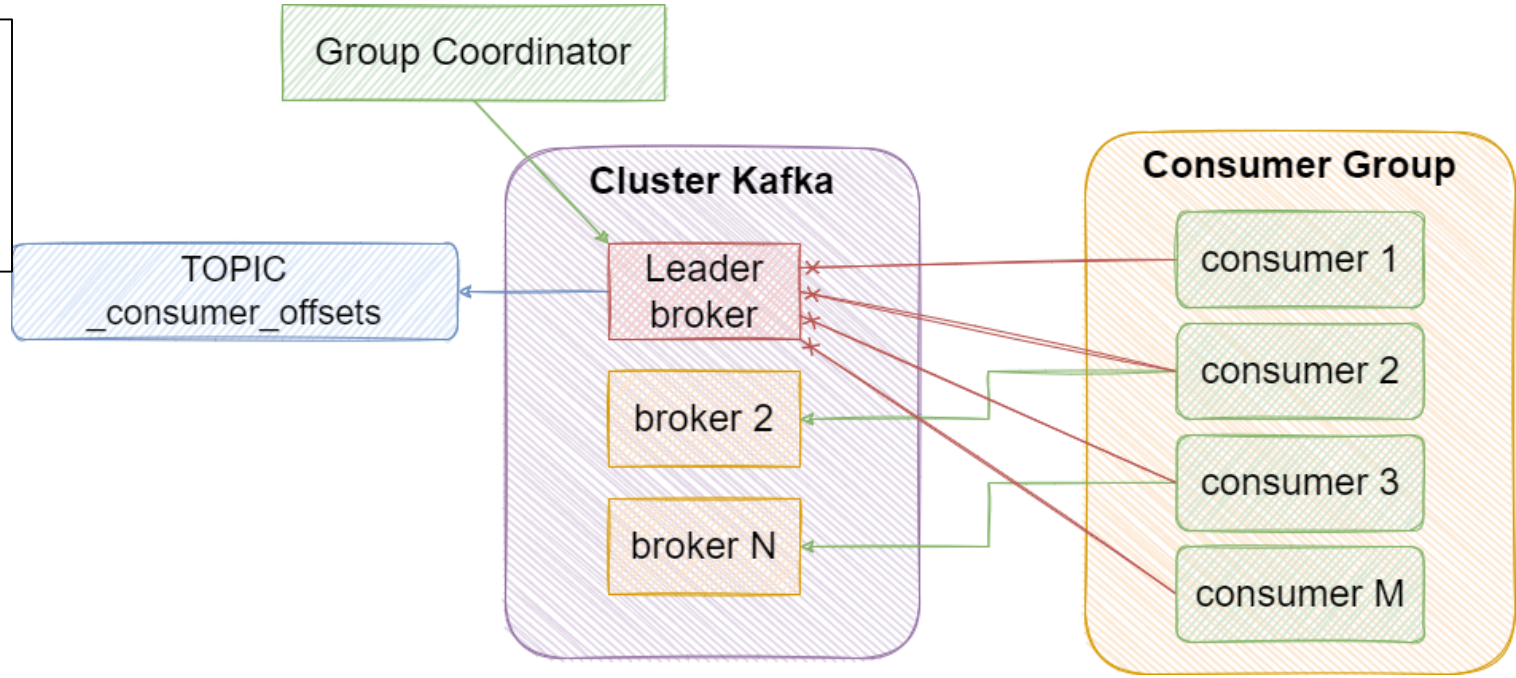
2. Брокера накрыло «цунами» запросов от огромных консьюмерных групп



GroupCoordinatorNotAvailableException: The group coordinator is not available

1. Машине, на которой живёт брокер, стало плохо, брокер вовремя не отвечает

1. Смотрим метрики машины по физическим ресурсам (CPU, RAM, I/O, Disk Usage, Network)

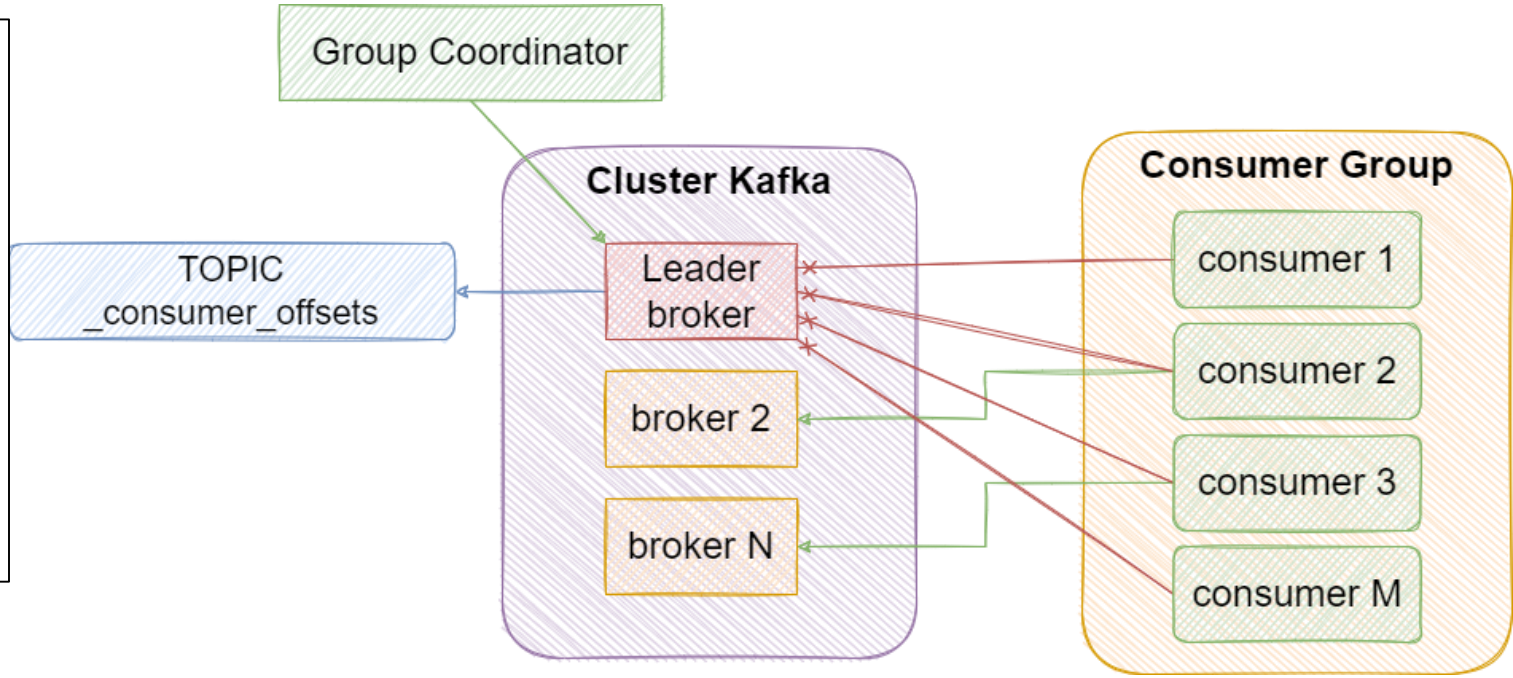


GroupCoordinatorNotAvailableException: The group coordinator is not available

1. Машине, на которой живёт брокер, стало плохо, брокер вовремя не отвечает

1. Смотрим метрики машины по физическим ресурсам (CPU, RAM, I/O, Disk Usage, Network)

2. Исключаем проблему большого количества запросов, оставив 1 консьюмер в группе.



GroupCoordinatorNotAvailableException: The group coordinator is not available

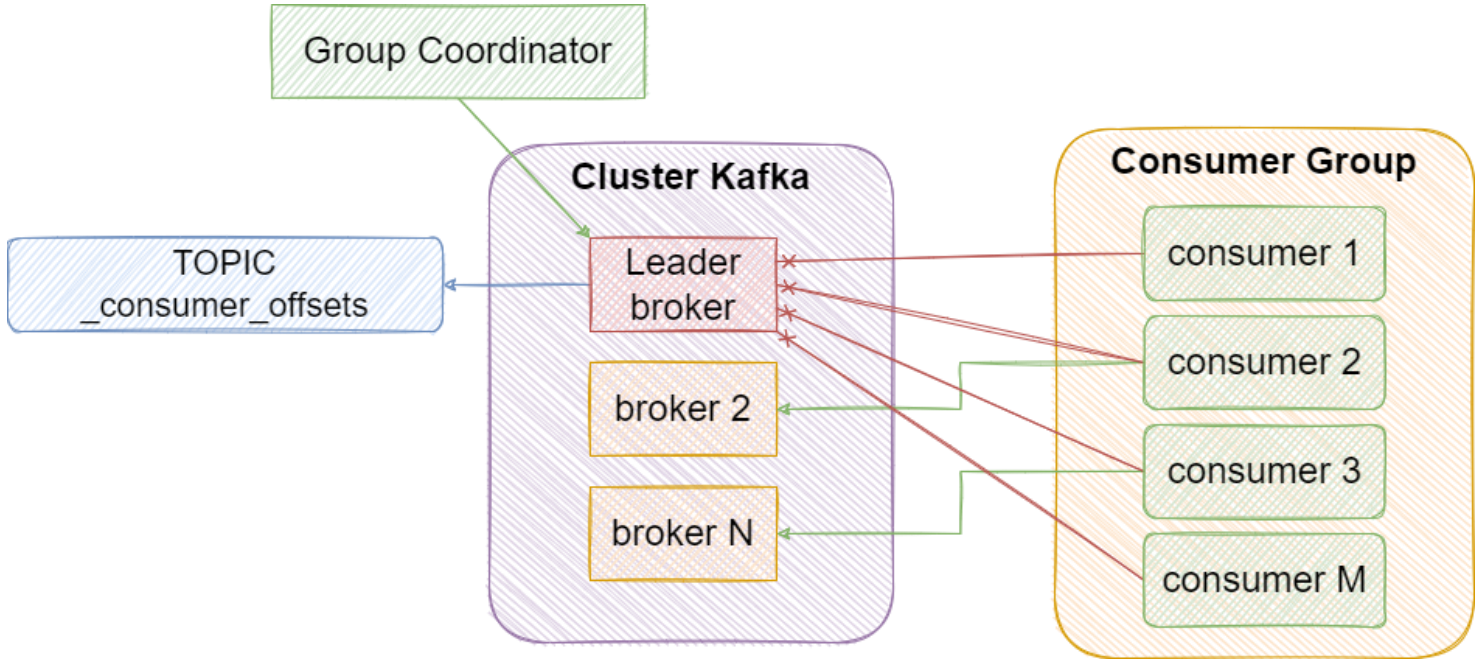
1. Машине, на которой живёт брокер, стало плохо, брокер вовремя не отвечает

https://www.stuckintheloop.com/posts/my_first_kafka_outage/

GroupCoordinatorNotAvailableException: The group coordinator is not available

2. Брокера накрыло «цунами» запросов от огромных консьюмерных групп

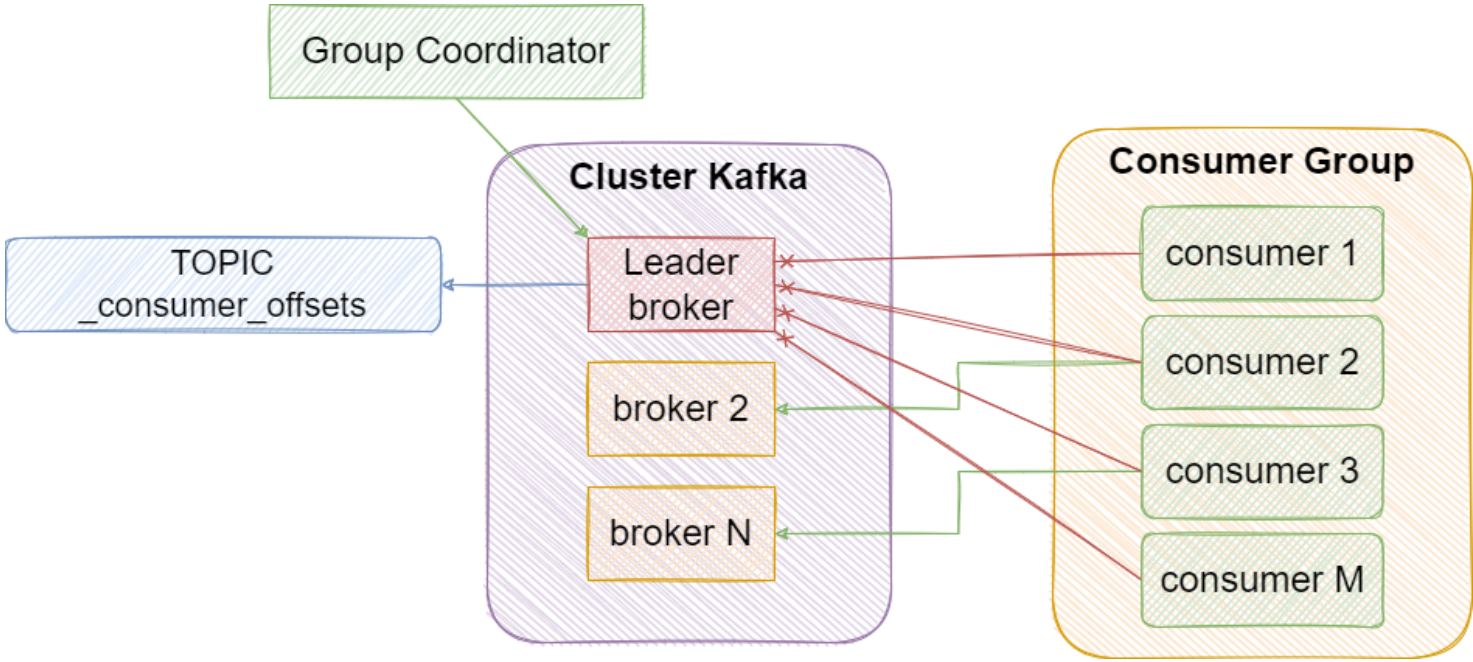
1. request.timeout.ms – увеличиваем (эмпирически)



GroupCoordinatorNotAvailableException: The group coordinator is not available

2. Брокера накрыло «цунами» запросов от огромных консьюмерных групп

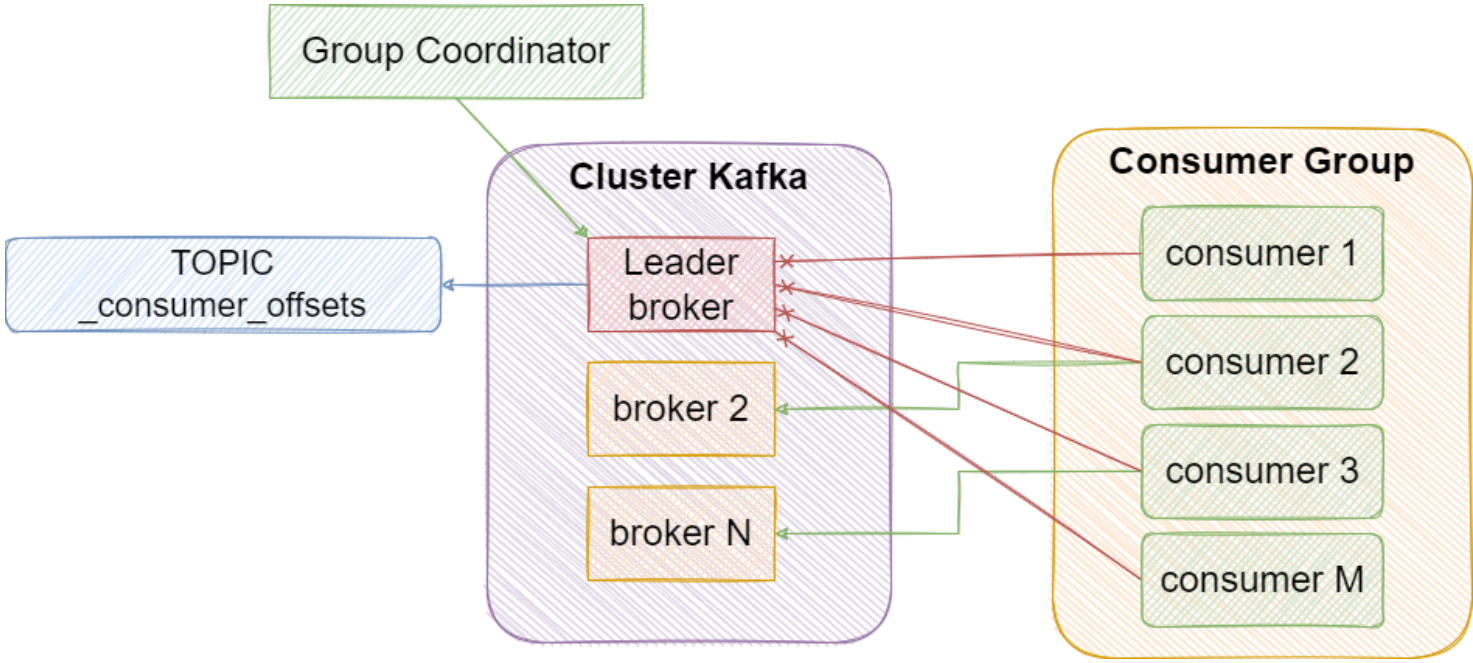
1. request.timeout.ms – увеличиваем (эмпирически)
2. session.timeout.ms – увеличиваем (эмпирически)



GroupCoordinatorNotAvailableException: The group coordinator is not available

2. Брокера накрыло «цунами» запросов от огромных консьюмерных групп

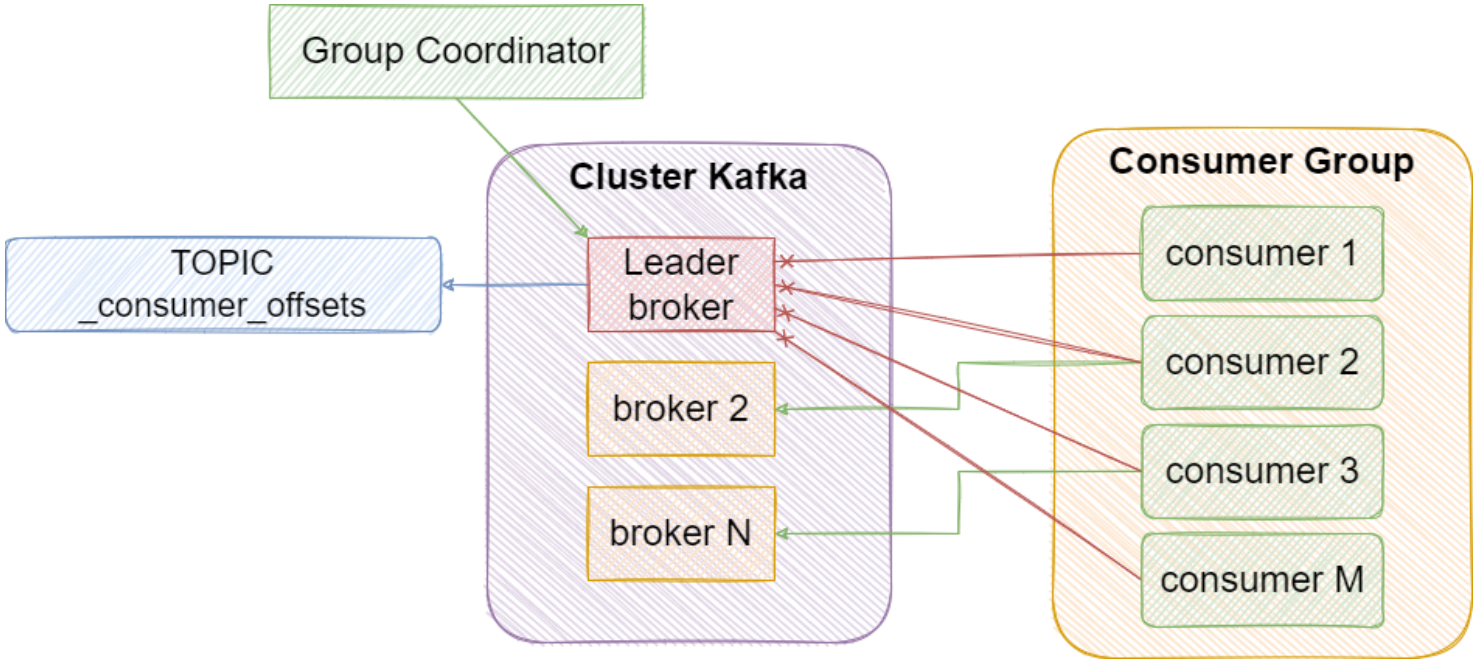
1. request.timeout.ms – увеличиваем (эмпирически)
2. session.timeout.ms – увеличиваем (эмпирически)
3. commit.interval.ms – увеличиваем (эмпирически)



GroupCoordinatorNotAvailableException: The group coordinator is not available

2. Брокера накрыло «цунами» запросов от огромных консьюмерных групп

1. request.timeout.ms – увеличиваем (эмпирически)
2. session.timeout.ms – увеличиваем (эмпирически)
3. commit.interval.ms – увеличиваем (эмпирически)
4. уменьшаем по возможности количество консьюмеров в группе



Итоги

Итоги

- Рассмотрена проблема производительности решения задачи потоковой обработки данных с Java и Kafka, когда нам не хватает параллельности за счёт партиций

Итоги

- Рассмотрена проблема производительности решения задачи потоковой обработки данных с Java и Kafka, когда нам не хватает параллельности за счёт партиций
- Рассмотрена проблема утилизации ресурсов при равномерном распределении партиций на консьюмеры

Итоги

- Рассмотрена проблема производительности решения задачи потоковой обработки данных с Java и Kafka, когда нам не хватает параллельности за счёт партиций
- Рассмотрена проблема утилизации ресурсов при равномерном распределении партиций на консьюмеры
- Рассмотрены конкретные ошибки при работе с Consumer и Producer

Итоги

- Для решения проблем с производительностью есть несколько решений со своими преимуществами и недостатками:
  - Parallel Consumer
  - Consumer + Reactive Framework
  - Consumer + Thread Pool

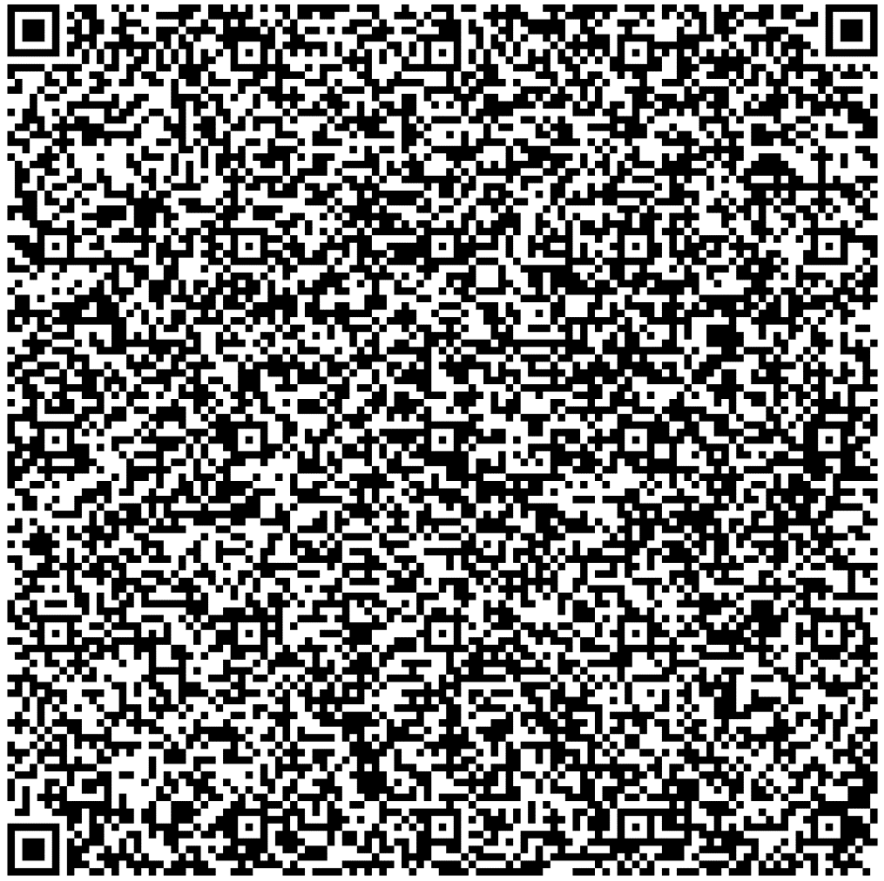
Итоги

- Для решения проблем с производительностью есть несколько решений со своими преимуществами и недостатками:
  - Parallel Consumer
  - Consumer + Reactive Framework
  - Consumer + Thread Pool
- Для решения проблем с утилизацией ресурсов можно воспользоваться подходом «умного» распределения партиций

Итоги

- Для решения проблем с производительностью есть несколько решений со своими преимуществами и недостатками:
  - Parallel Consumer
  - Consumer + Reactive Framework
  - Consumer + Thread Pool
- Для решения проблем с утилизацией ресурсов можно воспользоваться подходом «умного» распределения партиций
- Каждая проблема Consumer и Producer требует детальной диагностики и изменений совокупности параметров

Полезная инфа



1. Github: parallel consumer
2. How Do Kafka Concurrency and Throughput Work? Part 1
3. How Do Kafka Concurrency and Throughput Work? Part 2
4. Github: Netflix concurrency-limits
5. PR Release parallel consumer 1.0
6. How to use the Confluent Parallel Consumer
7. Multi-Threaded Message Consumption with the Apache Kafka Consumer
8. Demo: MultiThreadedKafkaConsumer
9. Introducing the Confluent Parallel Consumer
10. Доклад: Когда всё пошло по Kafka 1 - Kafka
11. Доклад: Когда всё пошло по Kafka 2 - Producer
12. Доклад: Когда всё пошло по Kafka 3 - Consumer
13. Kafka Consumer Issues: Fixing JVM Garbage Collection Problems
14. How to choose the best Java garbage collector
15. Make sure to choose the right GC for your Java application to achieve maximum performance

Евгений Ненахов

 **@neltari**