Николай Рудопас | Т-Банк

Кровь, пот и Kafka

Request/reply не больно (почти)

Николай Рудопас

Техлид команды в Sales Tech, Т-Банк

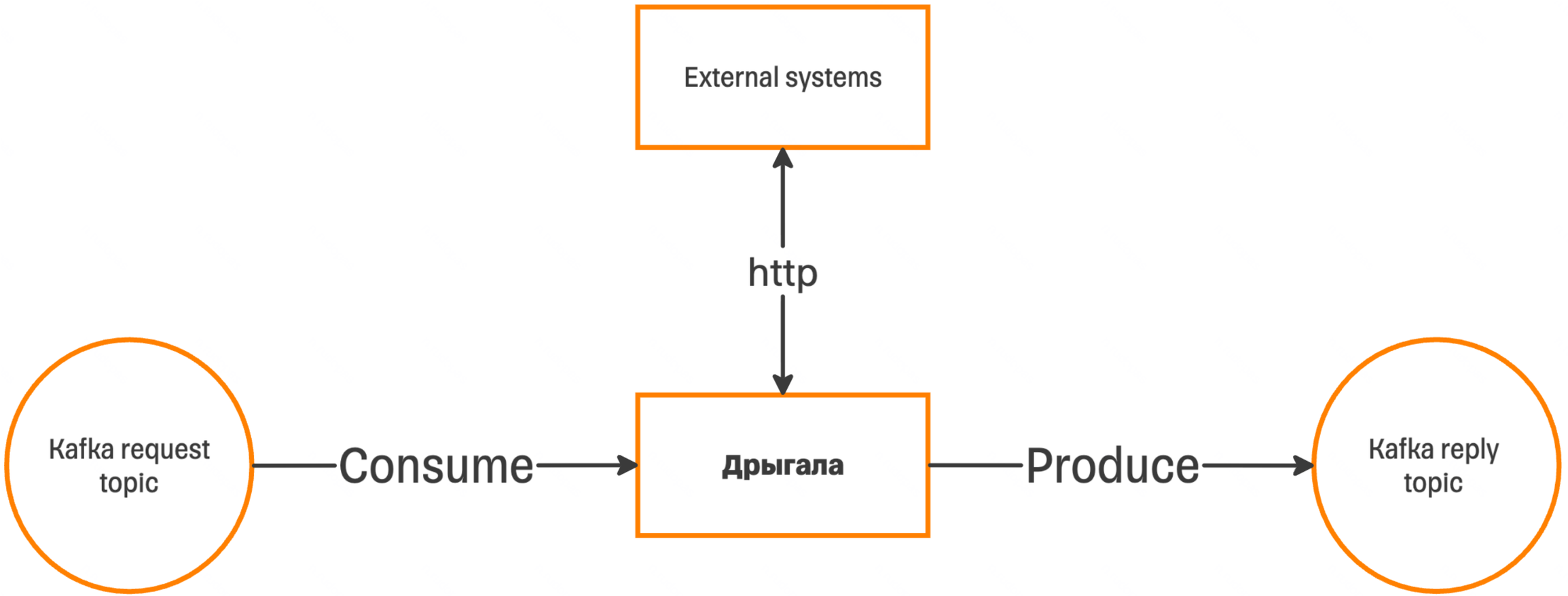
- 10+ лет разрабатываю всякое
- 5+ лет разрабатываю всякое на Java/Kotlin и Spring/Spring Boot
- Реализую бизнес-процессы на стриминге в Т-Банк

«Не суй в прод код из интернетов»

Unknown The Internet User

Дрыгалу хочу!

Рисунок на салфетке



Что-то еще?

Хотелки

Ответ на каждый запрос

Никогда и ни при каких обстоятельствах не терять сообщения

Хотелки

Ответ на каждый запрос

Никогда и ни при каких обстоятельствах не терять сообщения

Сохранять порядок

Стараться не менять порядок запросов и ответов

Хотелки

Ответ на каждый запрос

Никогда и ни при каких обстоятельствах не терять сообщения

Сохранять порядок

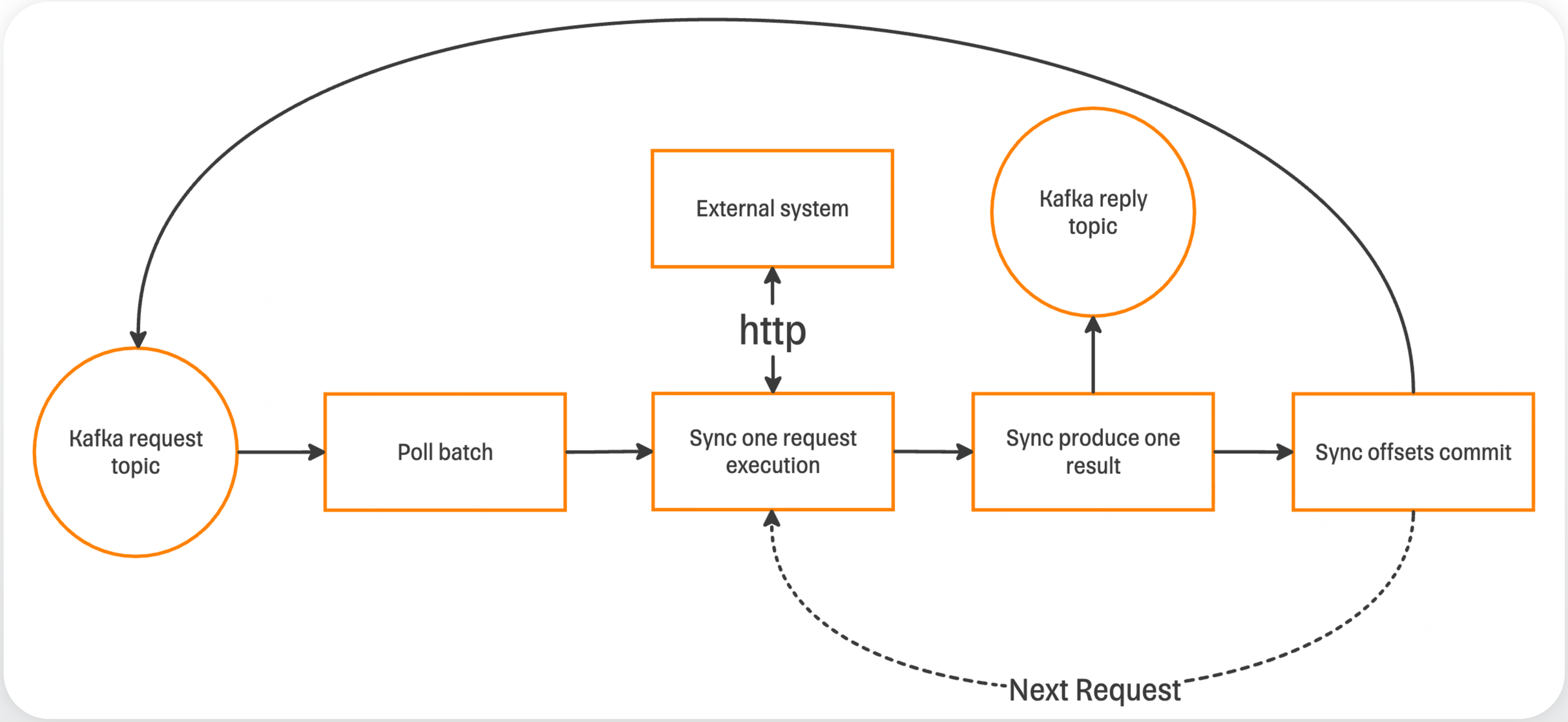
Стараться не менять порядок запросов и ответов

Не дублировать ответы

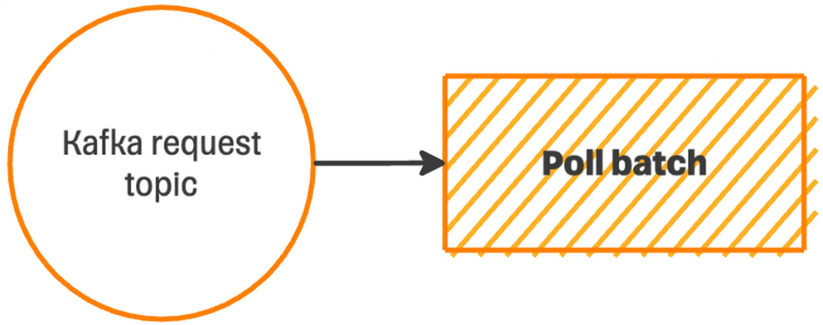
По возможности избегать дублирования ответа по одному запросу

Недолго думаем

Рисунок ключа

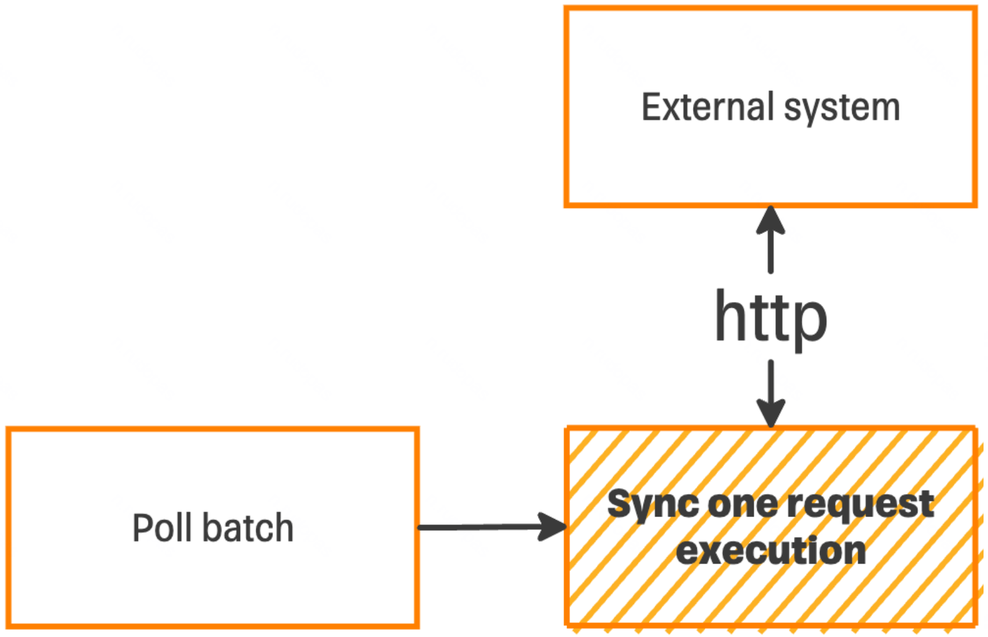


Полим



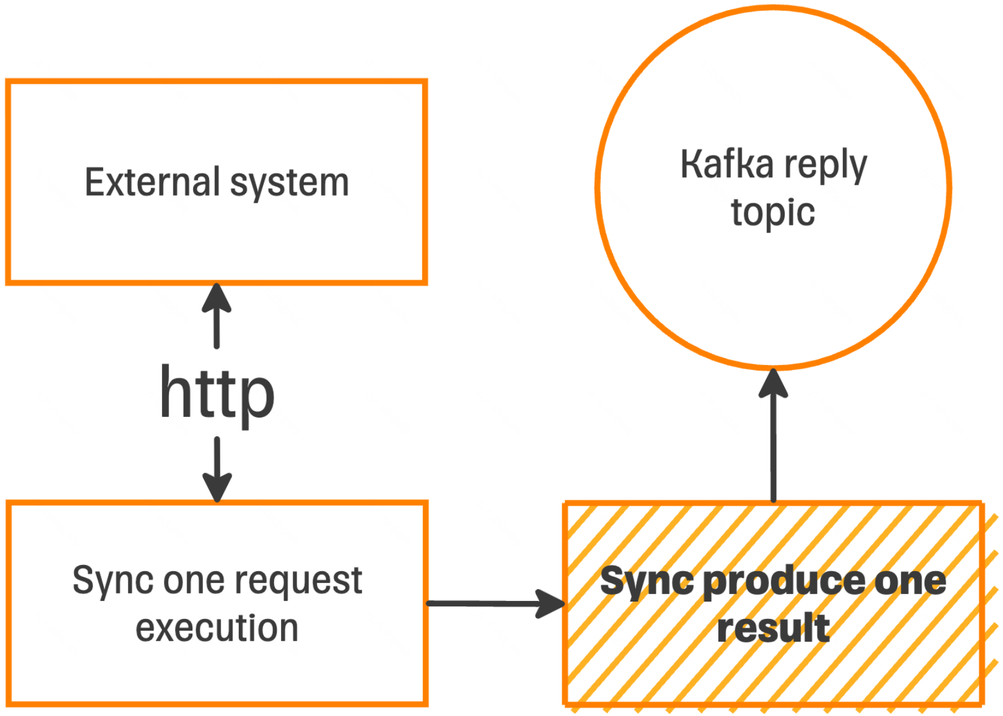

```
7   fun startProcessing(
8       consumer: KafkaConsumer<String, Request>,
9       requestTopic: String,
10      pollTimeout: Duration
11  ) {
12      // Подписсываемся на топик
13      consumer.subscribe(listOf(requestTopic))
14      while (true) {
15          // Полим батч
16          val requests = consumer.poll(pollTimeout)
17      }
18  }
```

Исполняем



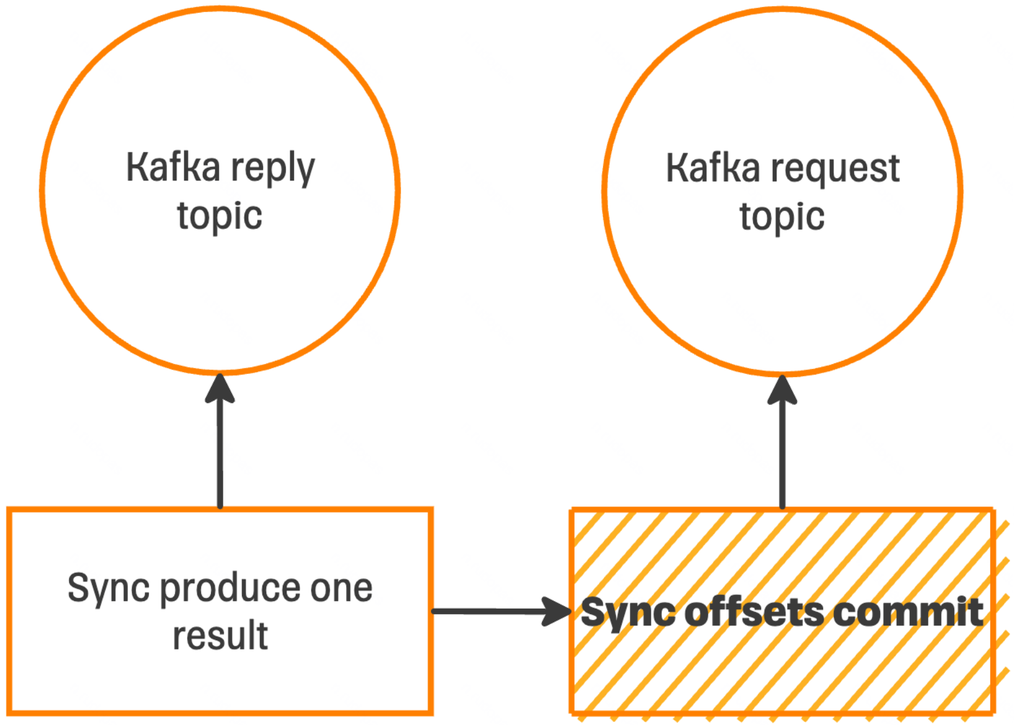

```
8   fun startProcessing(
9       consumer: KafkaConsumer<String, Request>,
10      requestTopic: String,
11      pollTimeout: Duration
12  ) {
13      // Подписываемся на топик
14      consumer.subscribe(listOf(requestTopic))
15      while (true) {
16          // Полим батч
17          val requests = consumer.poll(pollTimeout)
18          requests.forEach { record ->
19              // Делаем свои бизнес-дела
20              val response = getResponse(record.value())
21          }
22      }
23  }
```

Настойчиво отправляем




```kotlin
fun startProcessing(
    consumer: KafkaConsumer<String, Request>,
    requestTopic: String,
    pollTimeout: Duration,
    producer: KafkaProducer<String, Response>,
    responseTopic: String,
) {
    // Подписываемся на топик
    consumer.subscribe(listOf(requestTopic))
    while (true) {
        // Полим батч
        val requests = consumer.poll(pollTimeout)
        requests.forEach { record ->
            // Делаем свои бизнес-дела
            val response = getResponse(record.value())
            // Настойчиво отправляем
            producer
                .send(ProducerRecord(responseTopic, record.key(), response))
                .get()
        }
    }
}
```

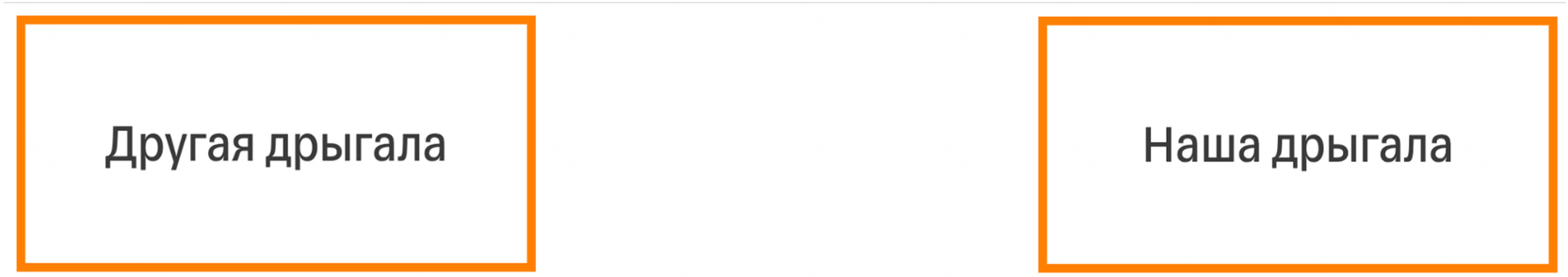
Настойчиво сдвигаем оффсеты



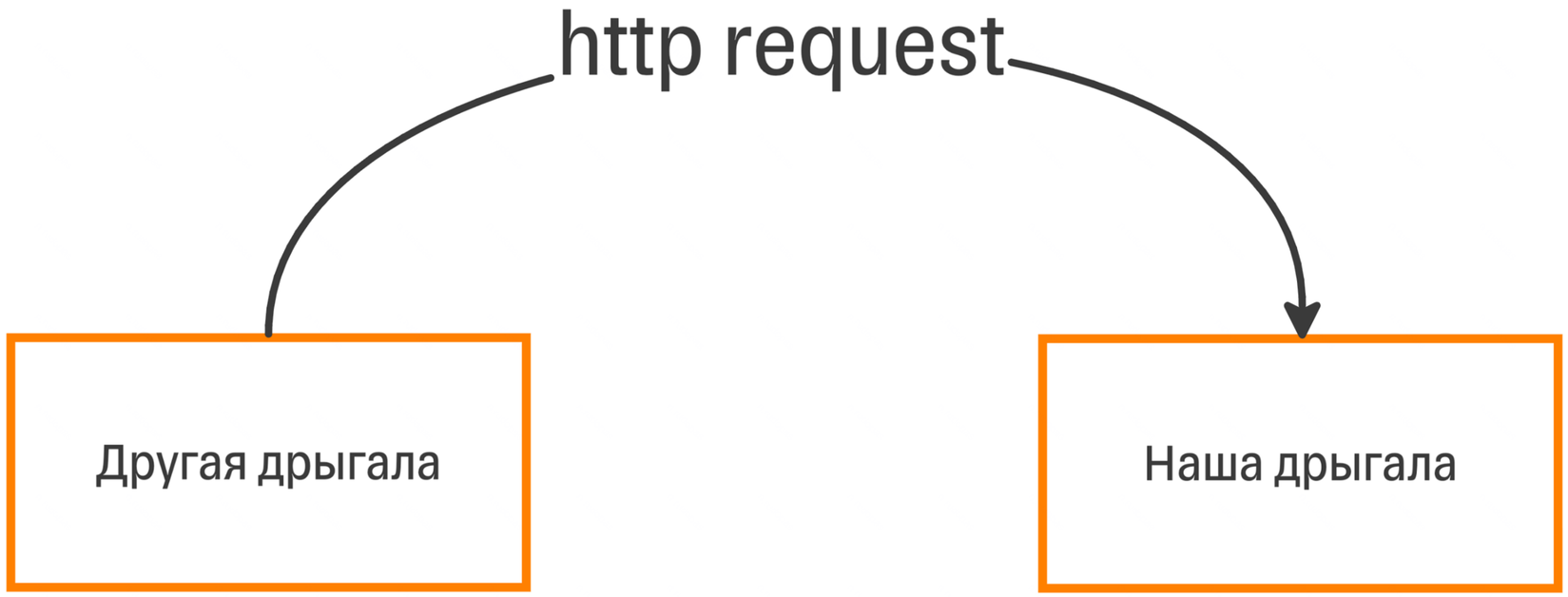

```kotlin
// Подписываемся на топик
consumer.subscribe(listOf(requestTopic))
while (true) {
    // Полим батч
    val requests = consumer.poll(pollTimeout)
    requests.forEach { record ->
        // Делаем свои бизнес-дела
        val response = getResponse(record.value())
        // Настойчиво отправляем
        producer
            .send(ProducerRecord(responseTopic, record.key(), response))
            .get()
        // Собираем оффсеты чтоб подвигать
        val offsets: Map<TopicPartition, OffsetAndMetadata> = mapOf(
            TopicPartition(requestTopic, record.partition())
                to OffsetAndMetadata( offset: record.offset() + 1)
        )
        // Настойчиво сдвигаем оффсеты
        consumer.commitSync(offsets)
    }
}
```

Надо что-то мерить

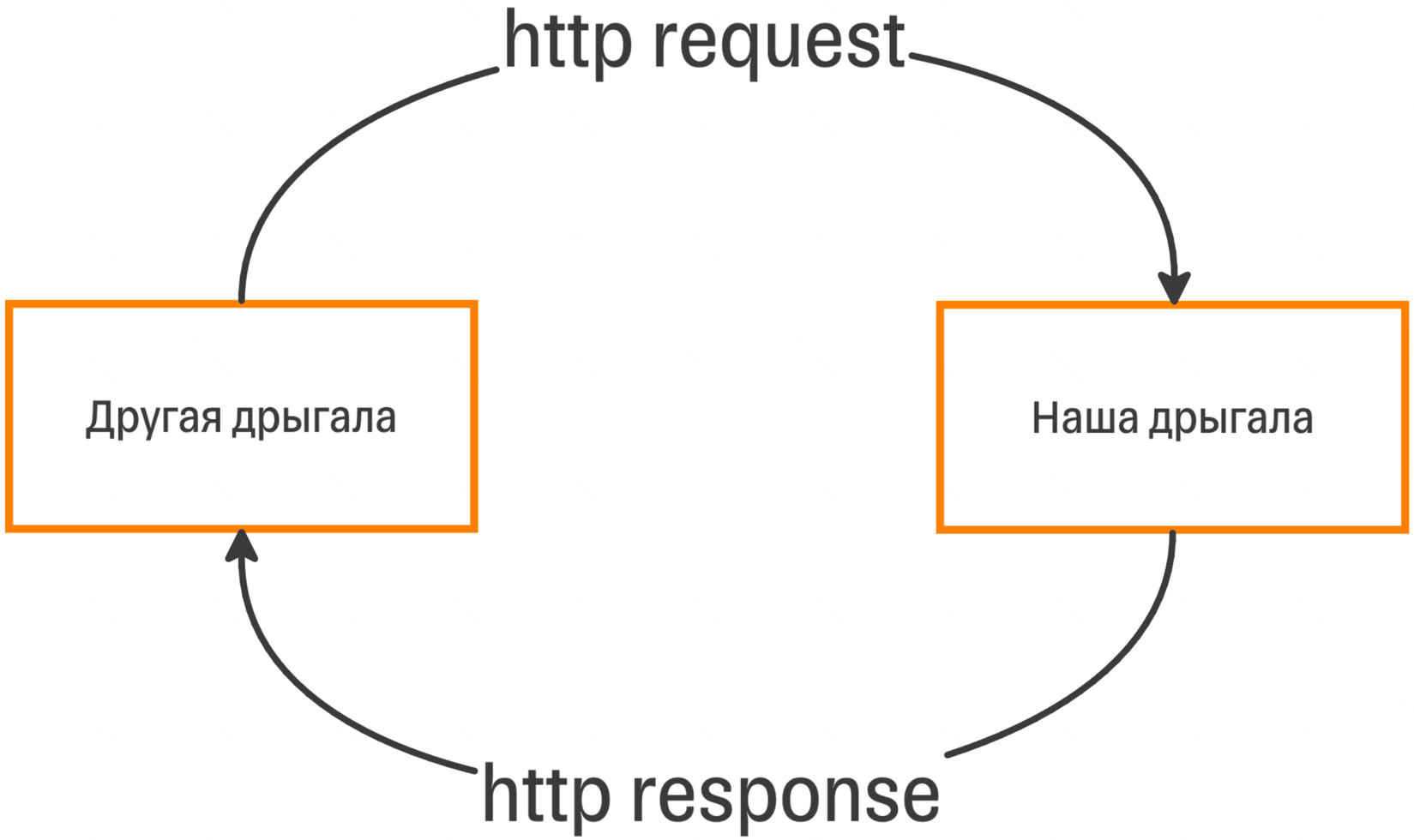
Http Request/Reply



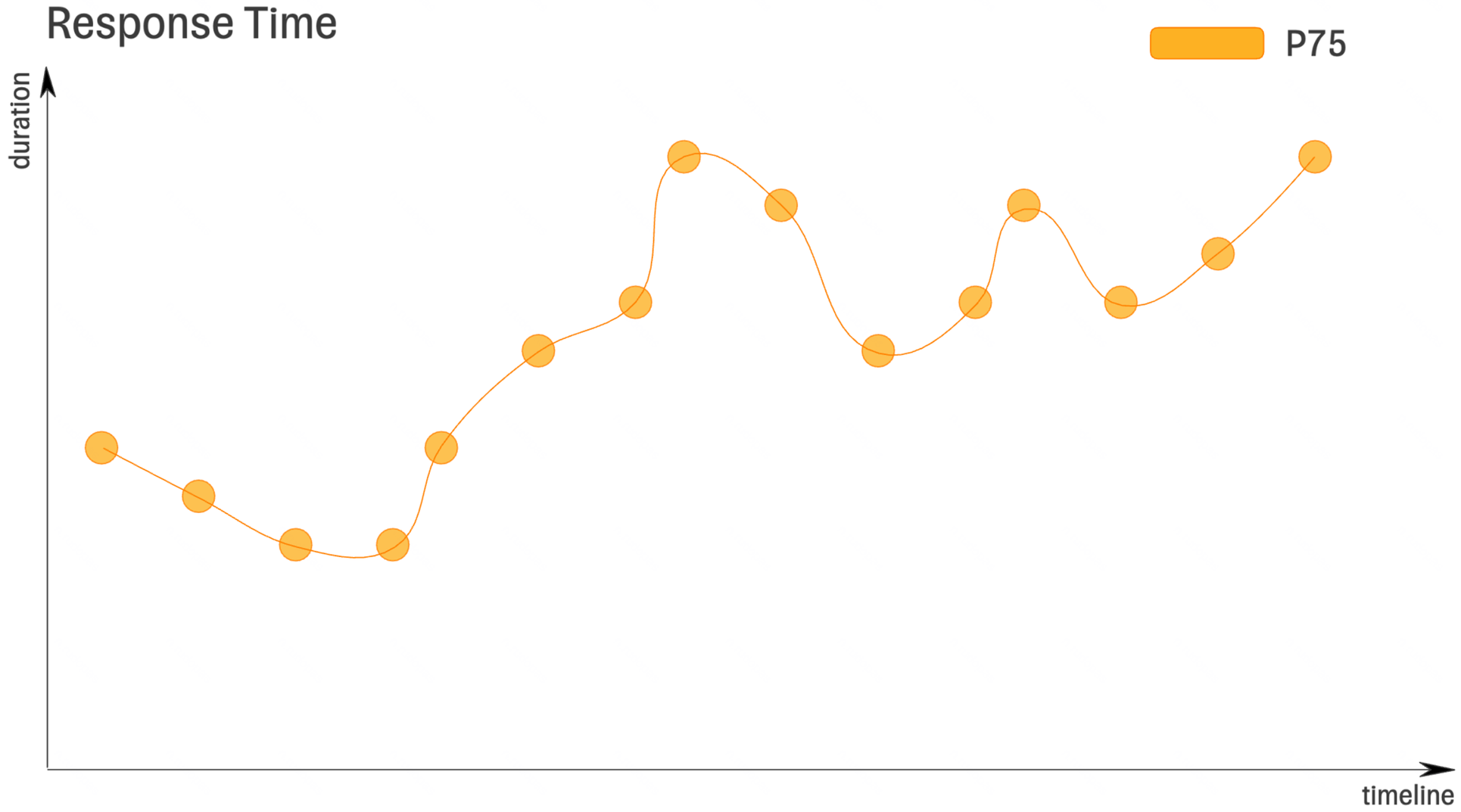
Http Request/Reply



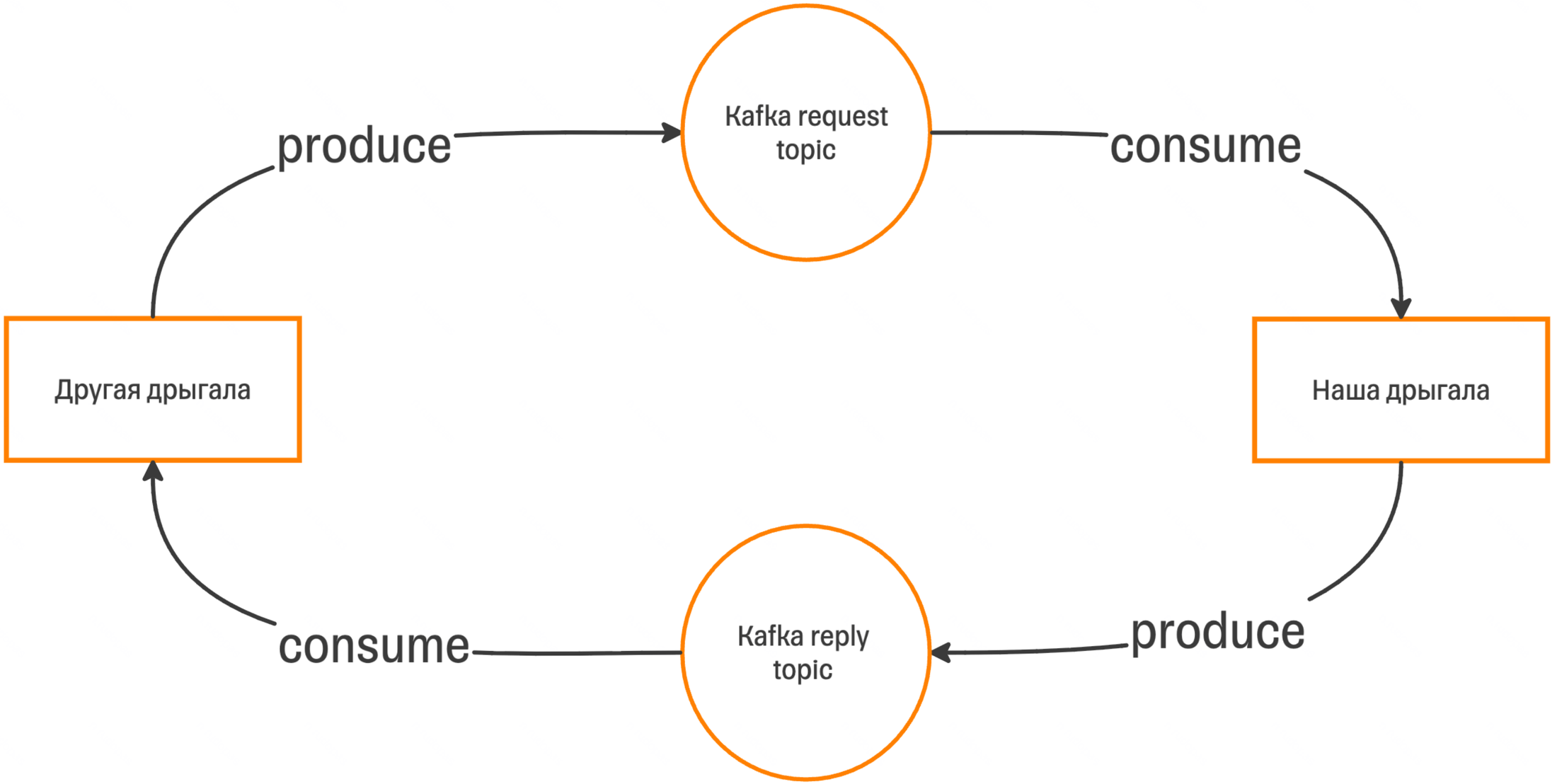
Http Request/Reply



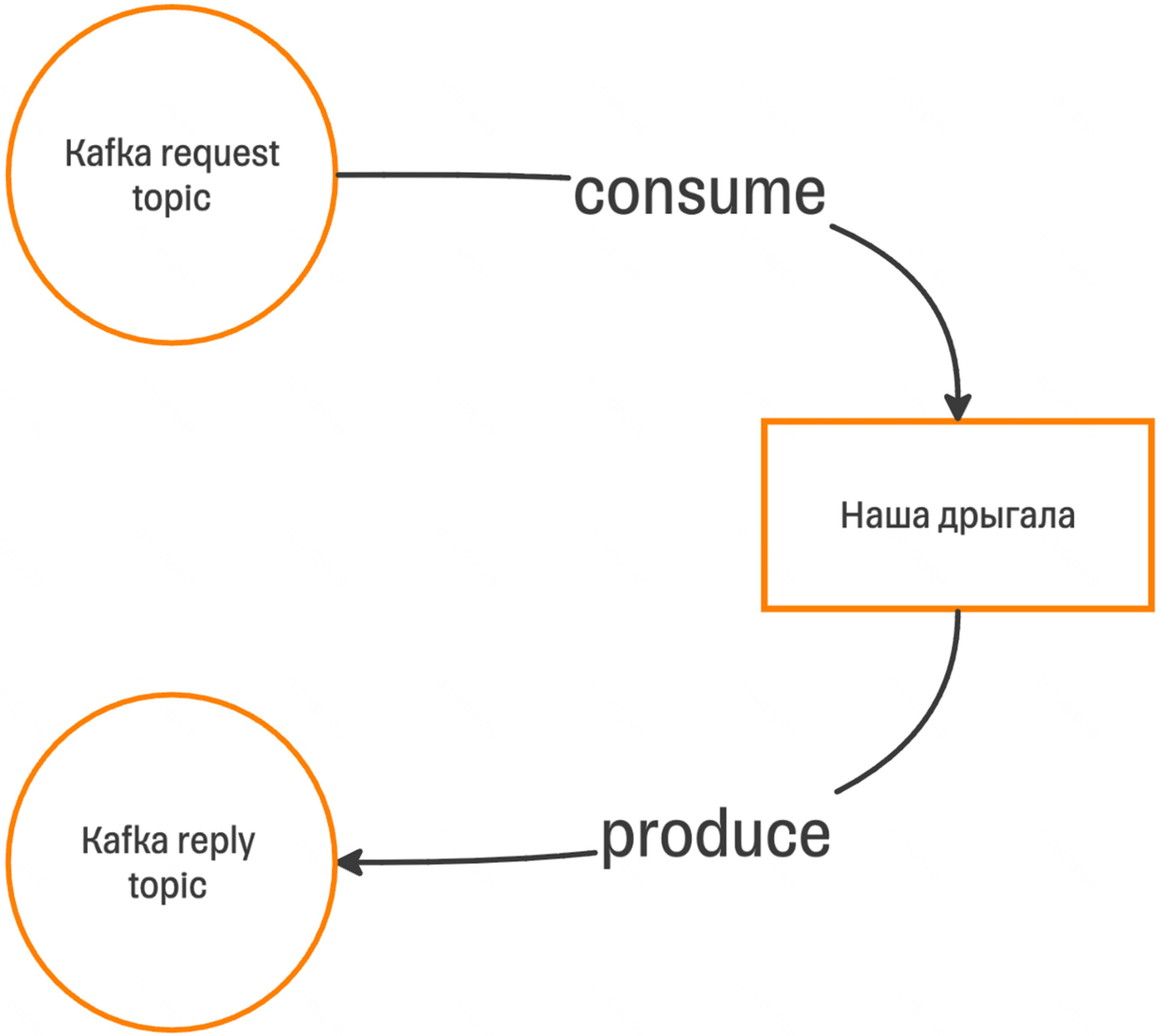
Http Request/Reply



Kafka Request/Reply



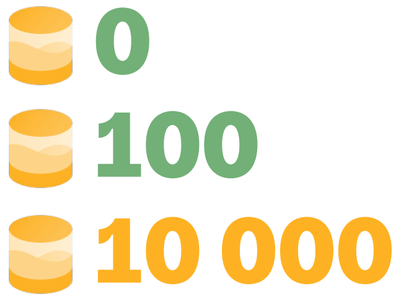
Kafka Request/Reply



Есть же consumer group lag

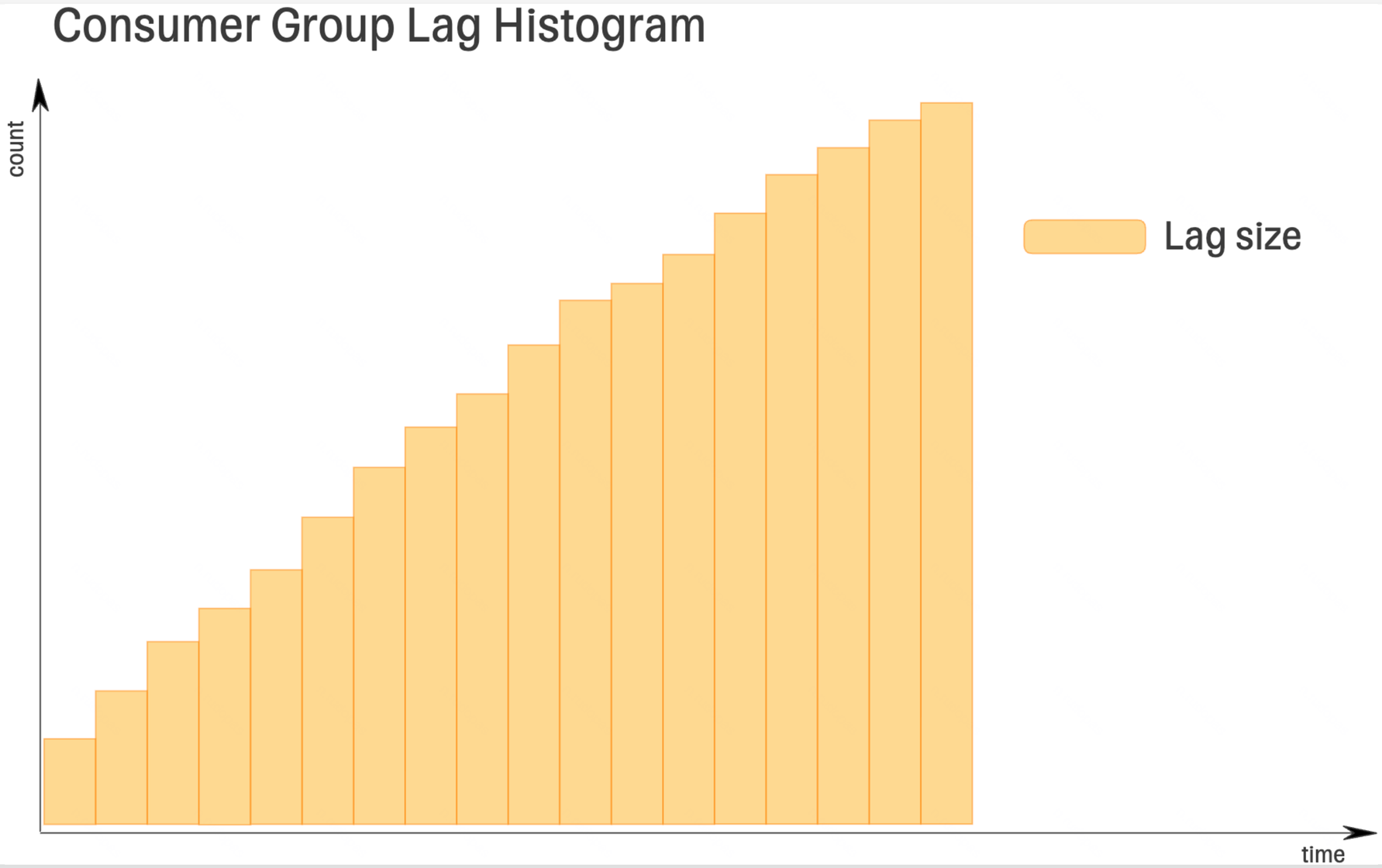
Есть же consumer group lag

Есть же consumer group lag

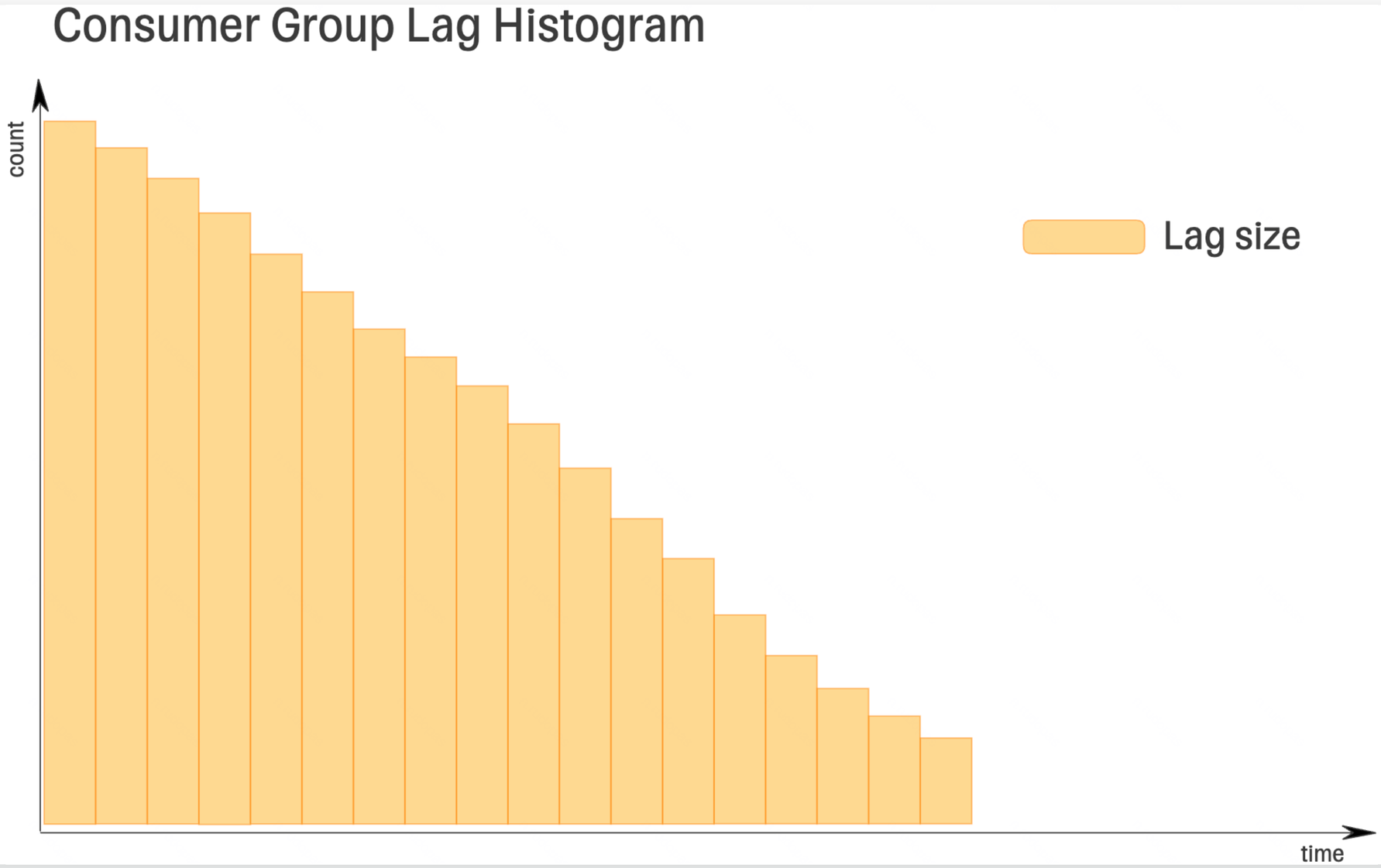


Есть же consumer group lag

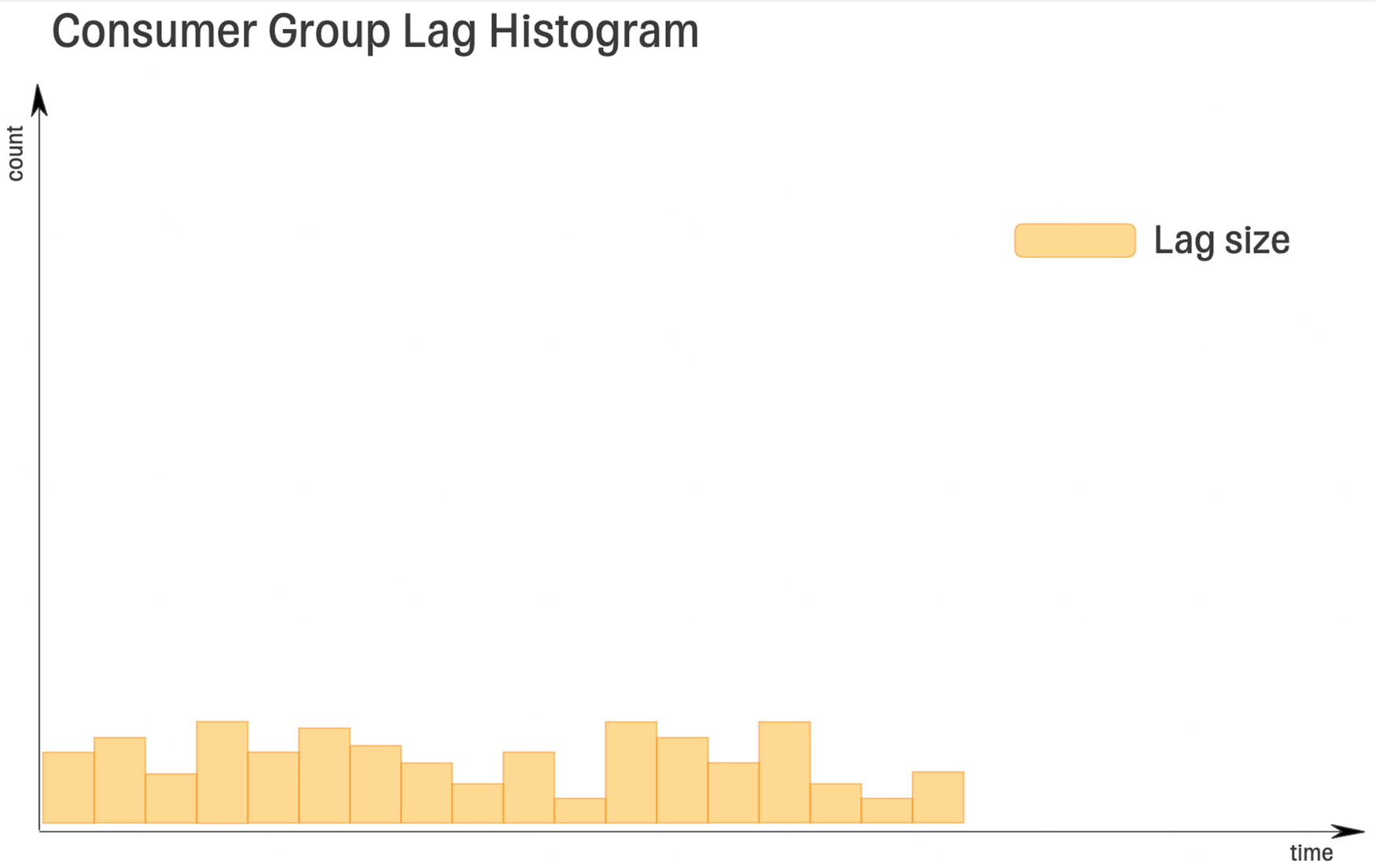
Consumer group lag может расти



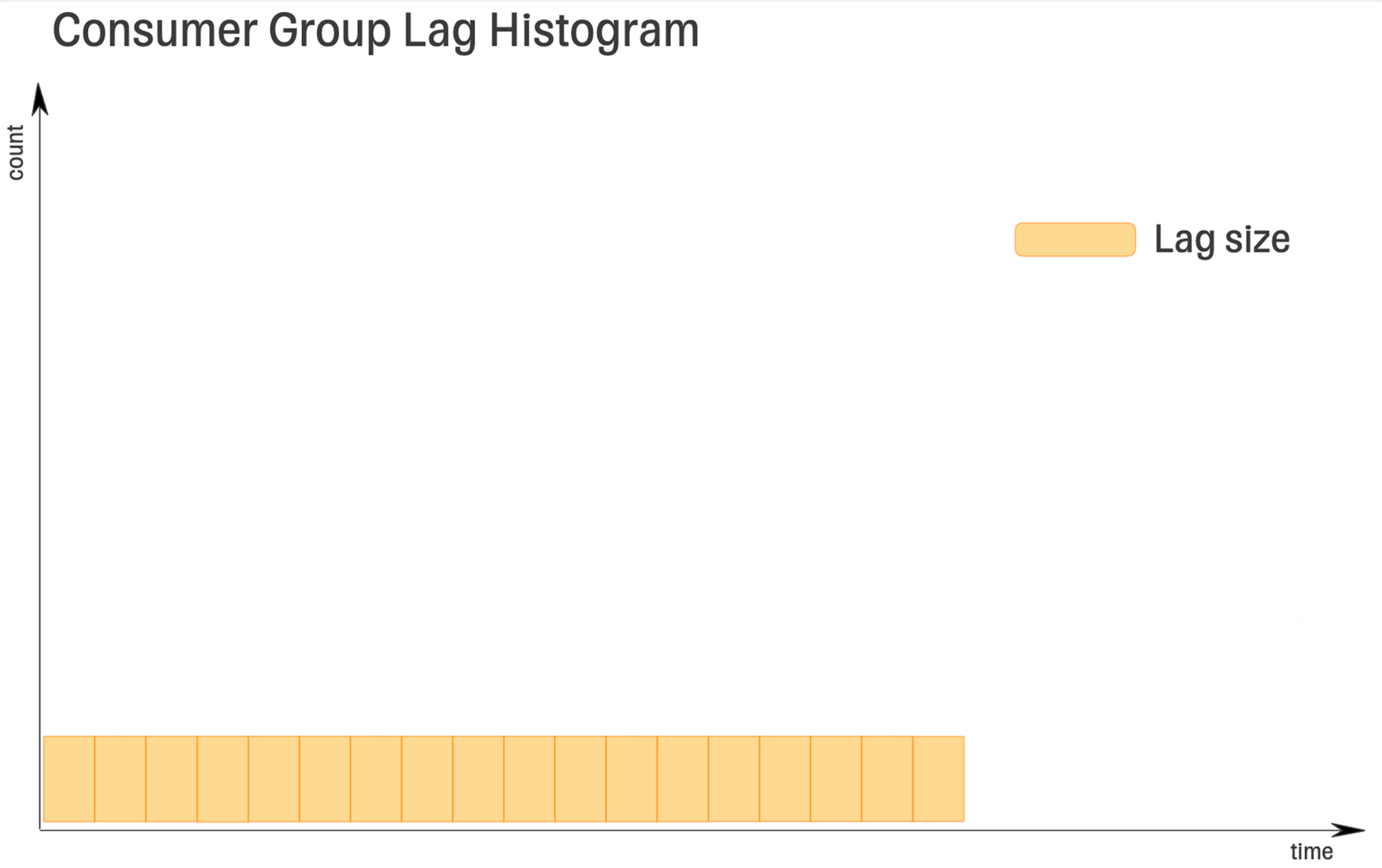
Consumer group lag может падать



Consumer group lag может почти не меняться



Consumer group lag может совсем не меняться



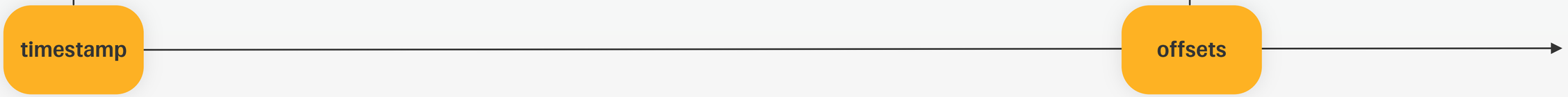
Consistency Lag. От сих до сих

Время создания запроса

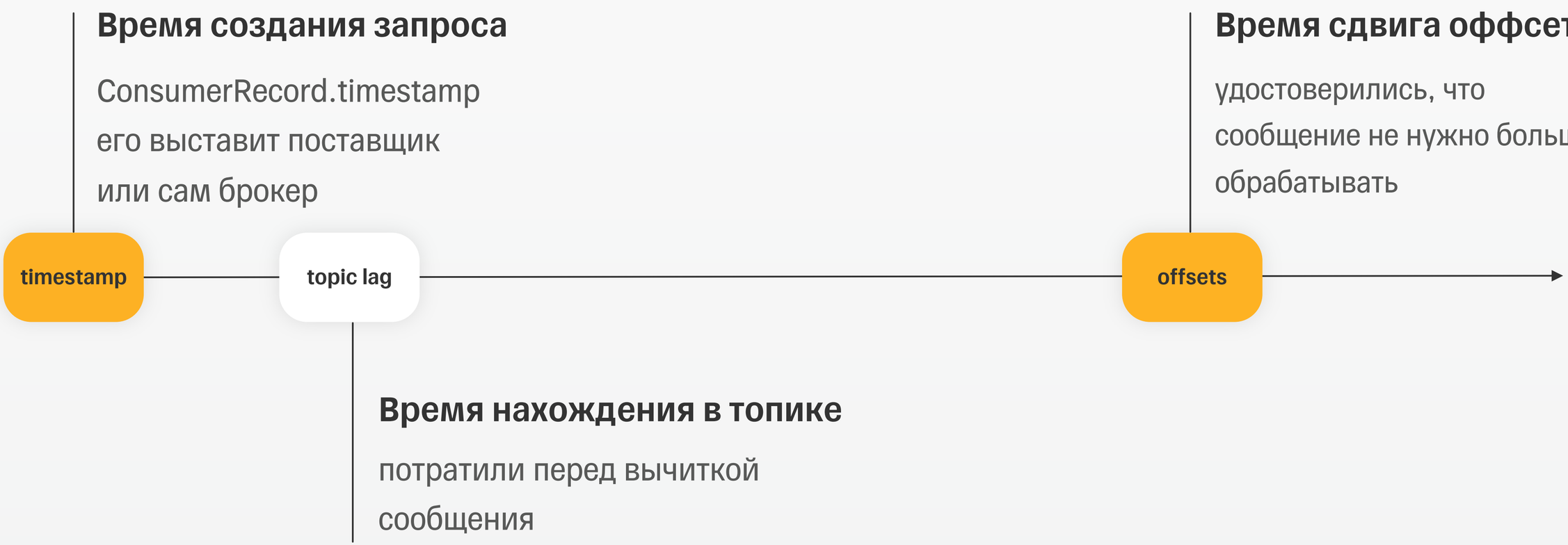
ConsumerRecord.timestamp
его выставит поставщик
или сам брокер

Время сдвига оффсетов

удостоверились, что
сообщение не нужно больше
обрабатывать

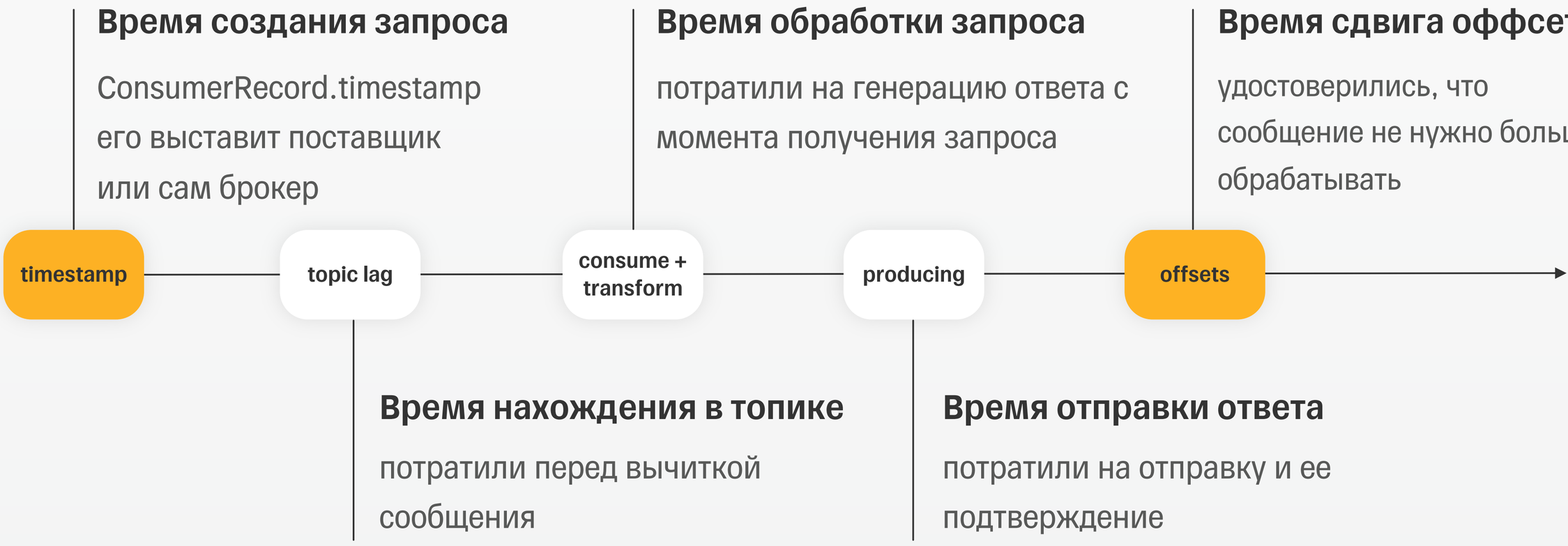


Consistency Lag. Пока лежало

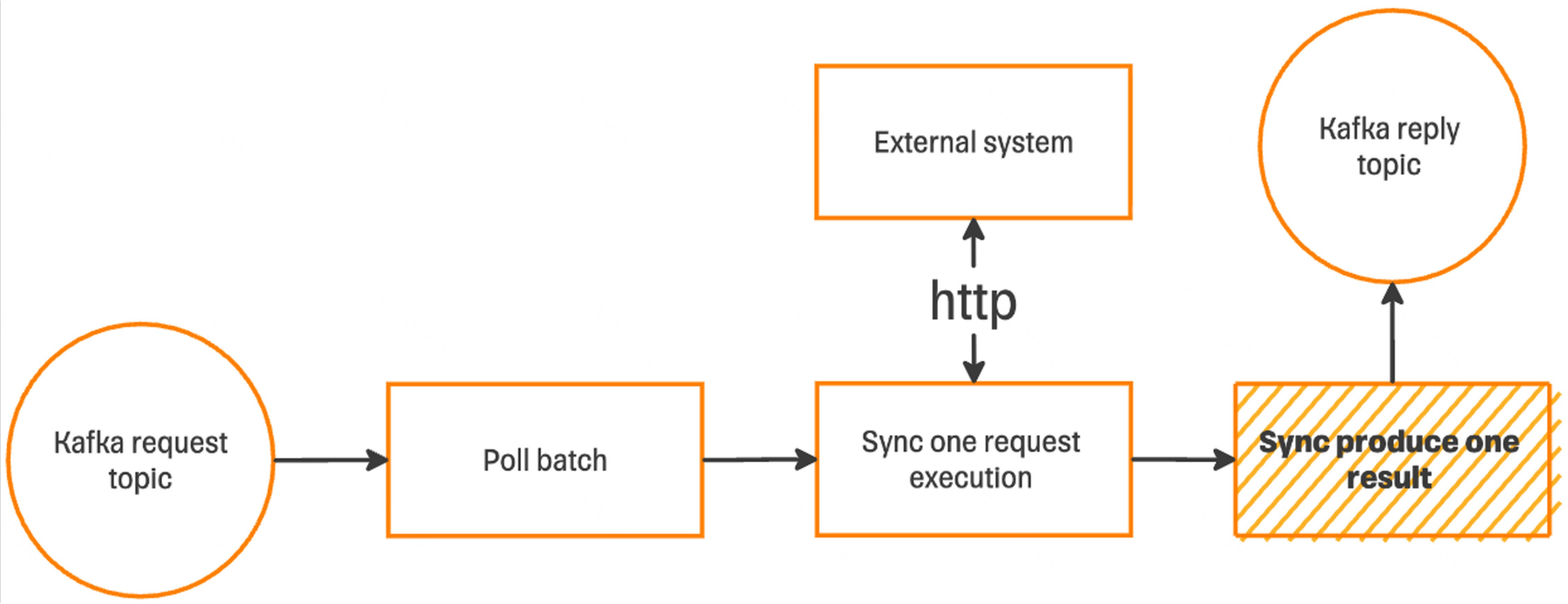


Consistency Lag. Пока обрабатывали

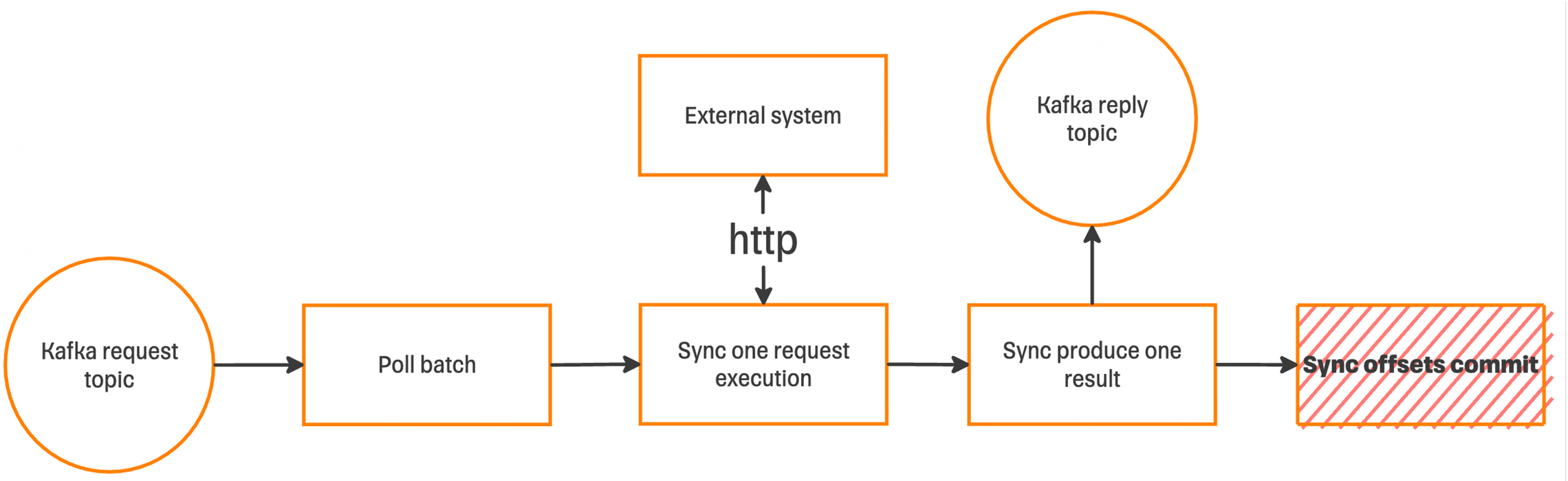
Consistency Lag. Пока отвечали



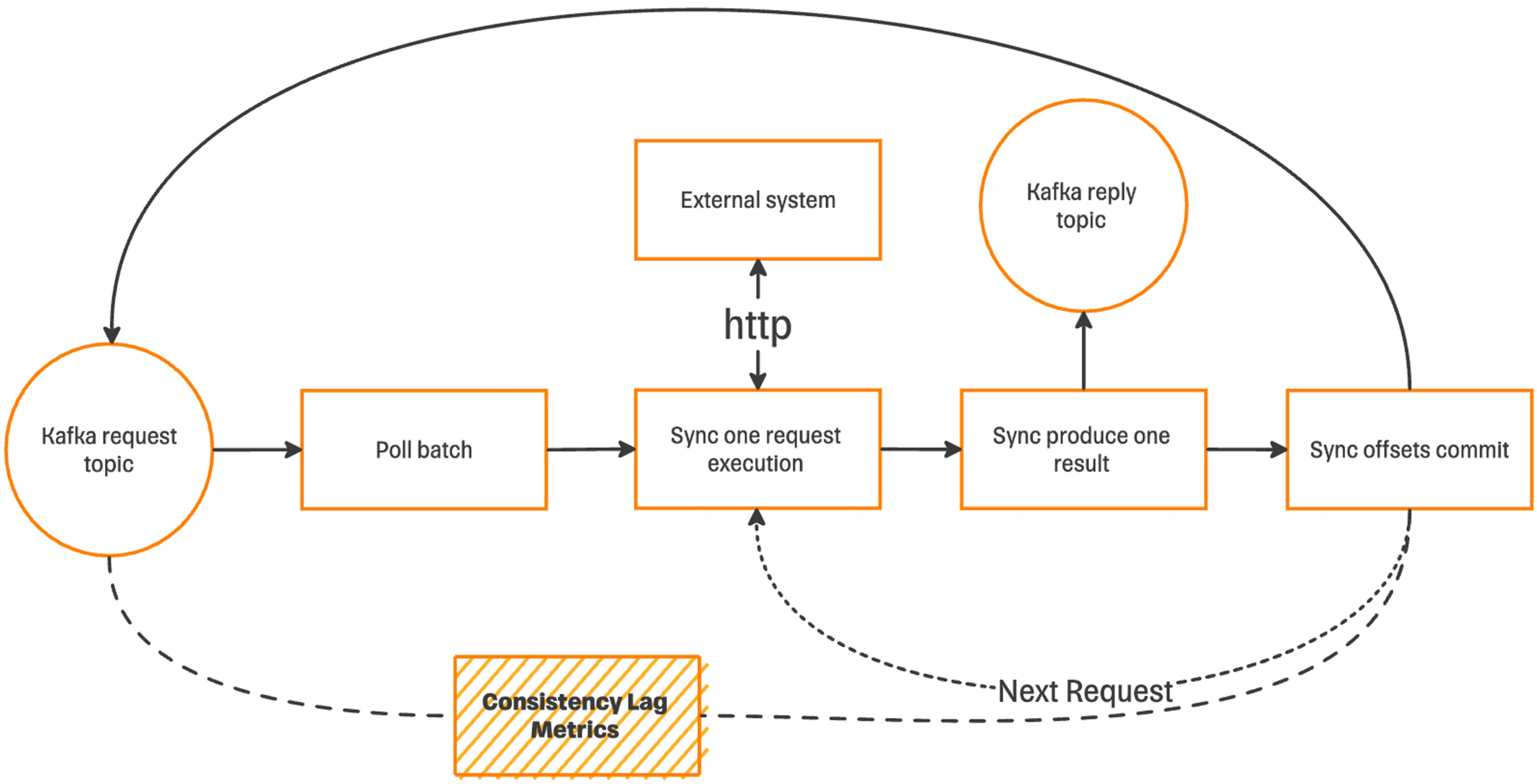
Почему до сдвига оффсетов



Отправим повторно



Финальный штрих



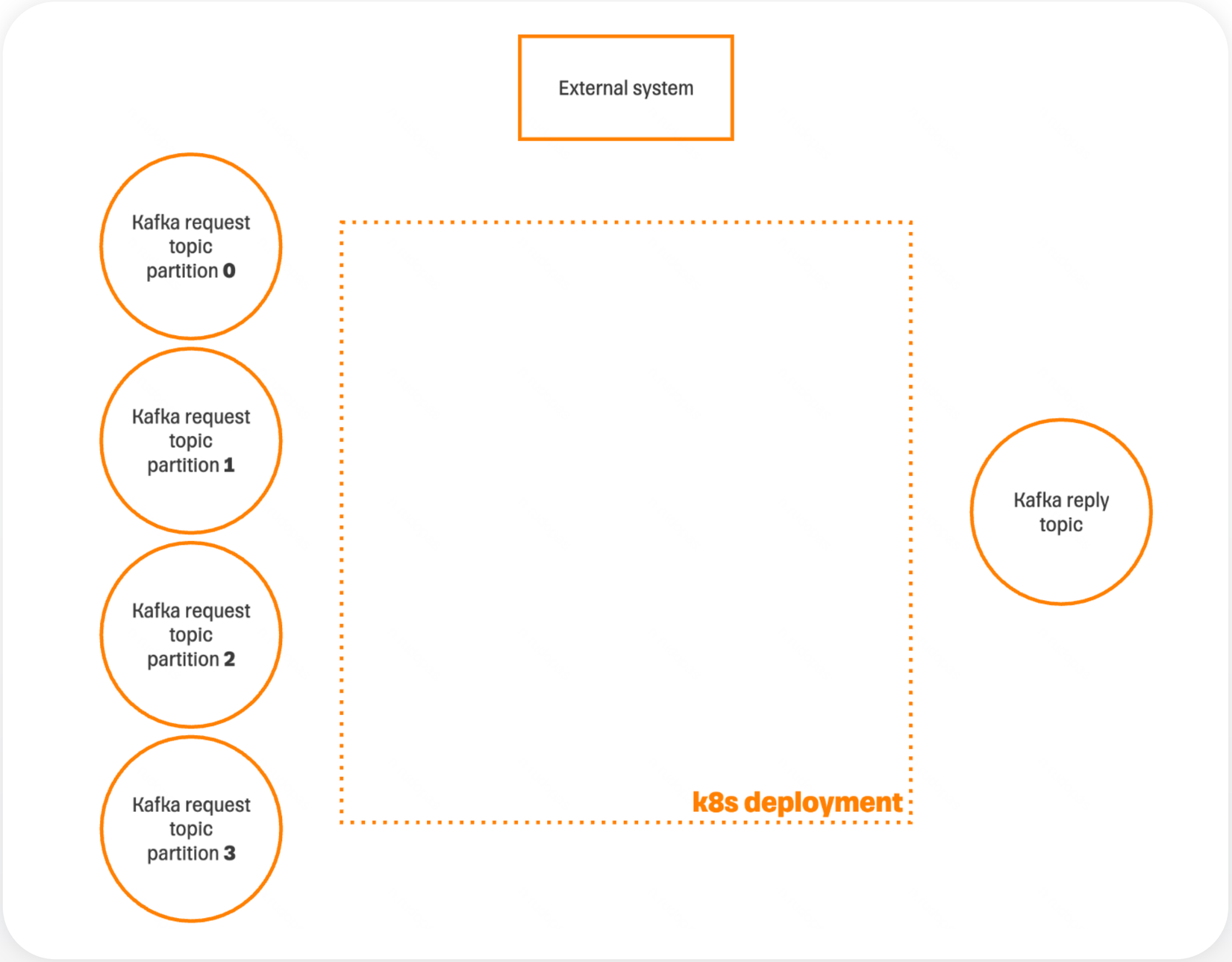
Считаем метрику

Без мам, пап и frameworks

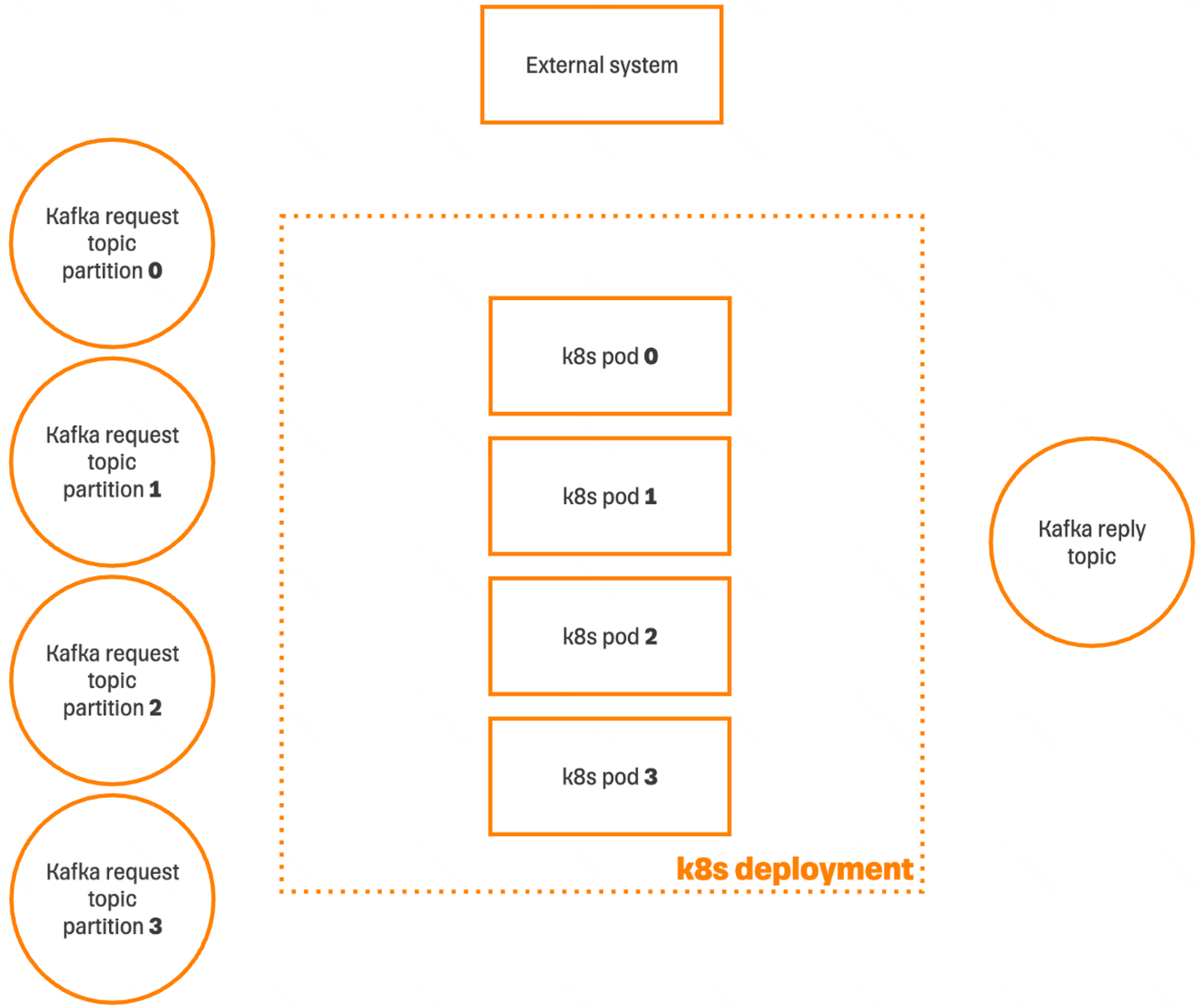
```
fun startProcessing(
    consumer: KafkaConsumer<String, Request>,
    requestTopic: String,
    pollTimeout: Duration,
    producer: KafkaProducer<String, Response>,
    responseTopic: String,
) {
    // Подписываемся на топик
    consumer.subscribe(listOf(requestTopic))
    while (true) {
        // Полим батч
        val requests = consumer.poll(pollTimeout)
        requests.forEach { record ->
            // Делаем свои бизнес-дела
            val response = getResponse(record.value())
            // Настойчиво отправляем
            producer
                .send(ProducerRecord(responseTopic, record.key(), response))
                .get()
            // Собираем оффсеты чтоб подвигать
            val offsets: Map<TopicPartition, OffsetAndMetadata> = mapOf(
                TopicPartition(requestTopic, record.partition())
                    to OffsetAndMetadata( offset: record.offset() + 1)
            )
            // Настойчиво сдвигаем оффсеты
            consumer.commitSync(offsets)
            // Считаем метрики
            recordConsistencyLagMetrics(
                Duration.ofMillis(
                    millis: Instant.now().toEpochMilli() - record.timestamp()
                )
            )
        }
    }
}
```

Собираем и деплоим

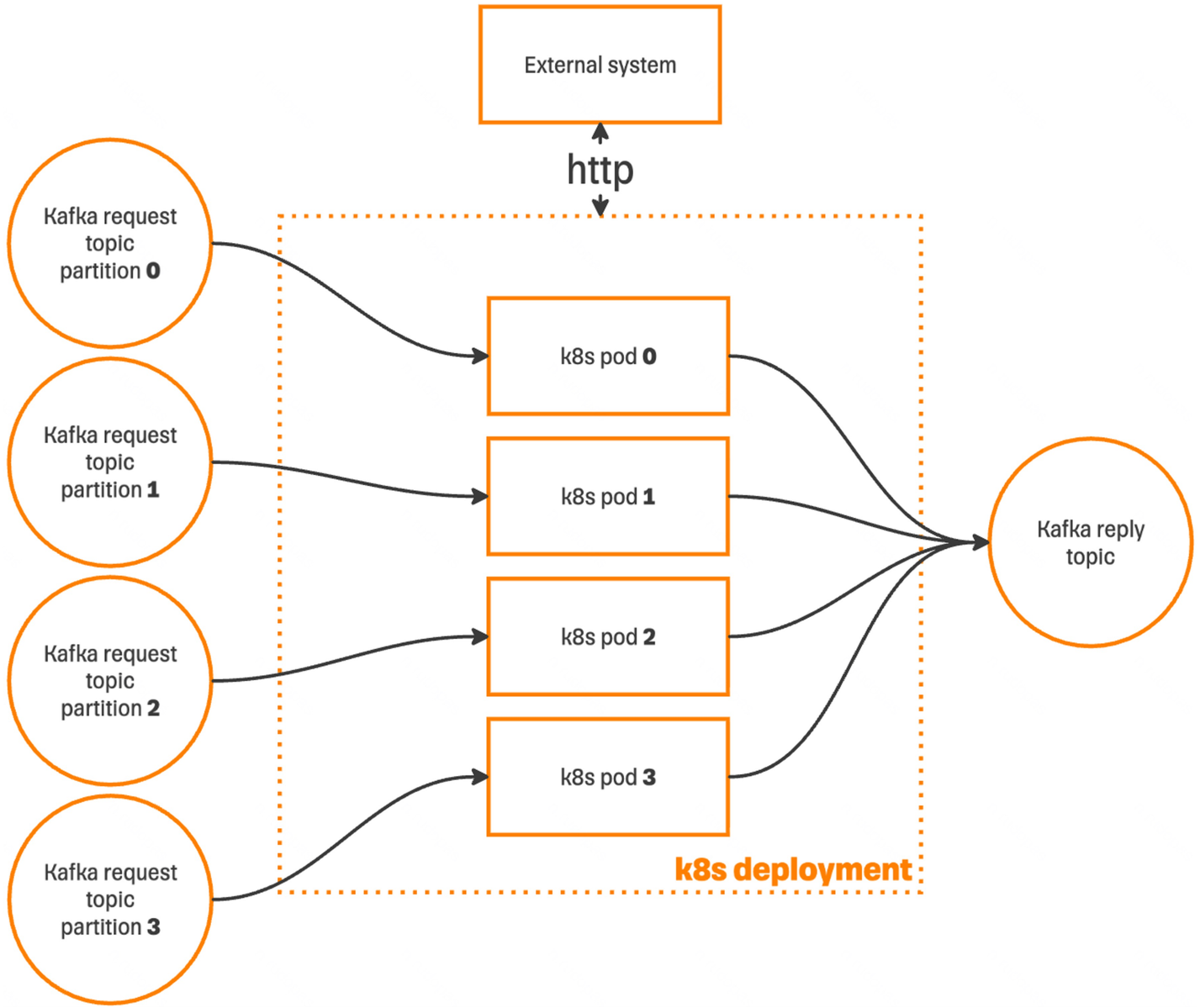
Деплоим


External system
Kafka request topic partition 0
Kafka request topic partition 1
Kafka request topic partition 2
Kafka request topic partition 3
k8s deployment
Kafka reply topic

Понеслась

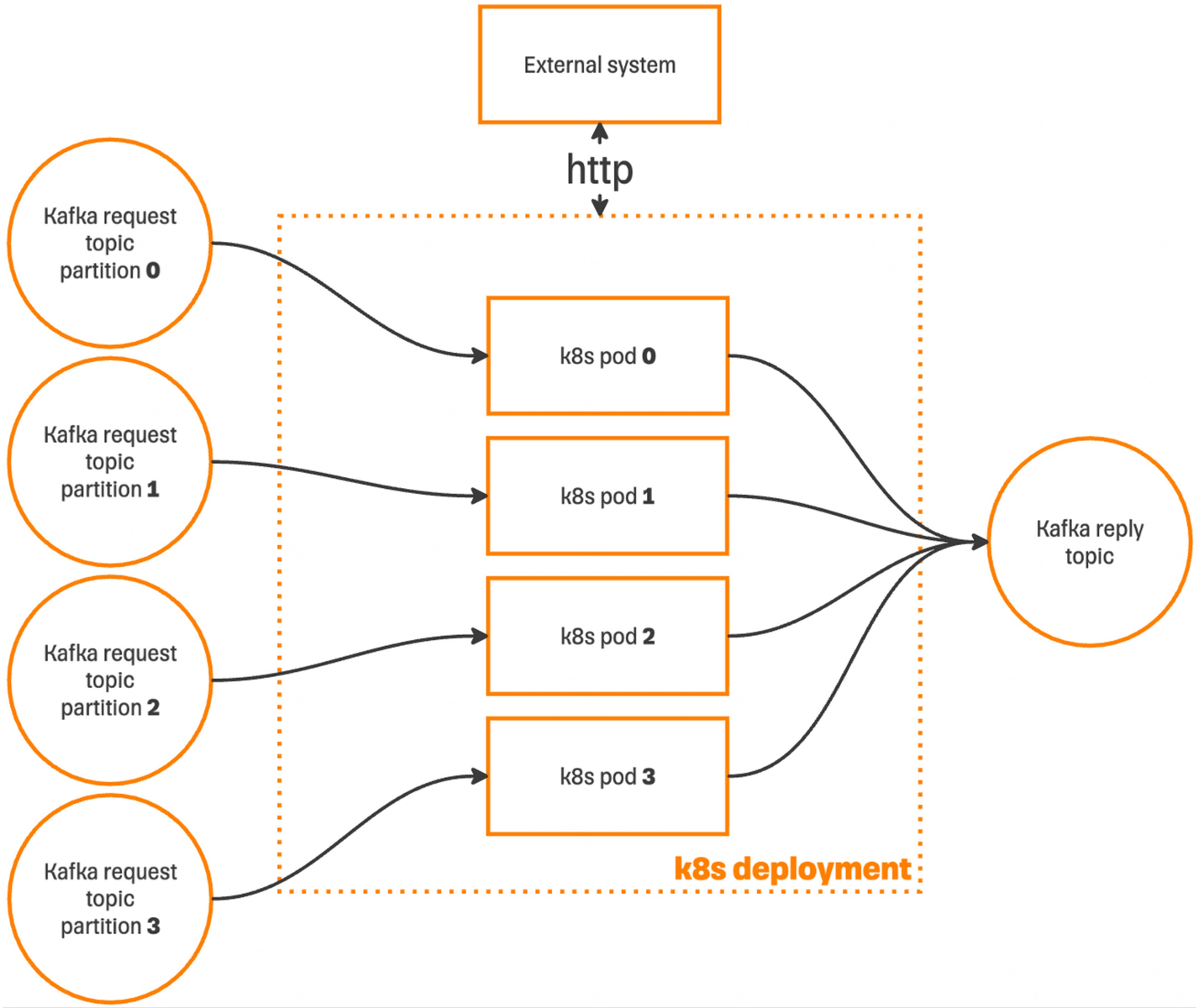

External system
Kafka request topic partition 0
Kafka request topic partition 1
Kafka request topic partition 2
Kafka request topic partition 3
k8s pod 0
k8s pod 1
k8s pod 2
k8s pod 3
k8s deployment
Kafka reply topic

Понеслась


External system
http
Kafka request topic partition 0
Kafka request topic partition 1
Kafka request topic partition 2
Kafka request topic partition 3
k8s pod 0
k8s pod 1
k8s pod 2
k8s pod 3
Kafka reply topic
k8s deployment

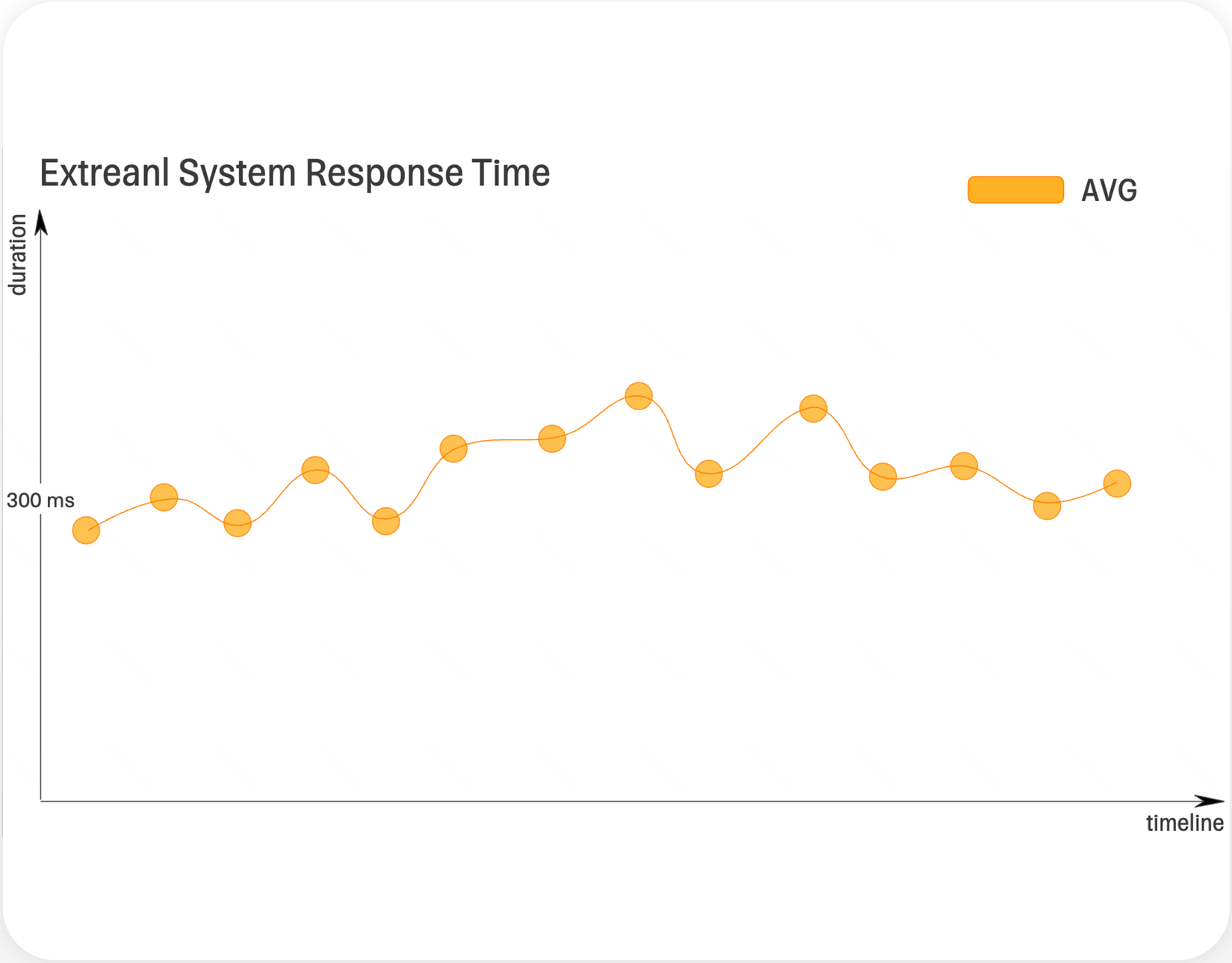
Смотрим. Красивое

- На pod по партиции



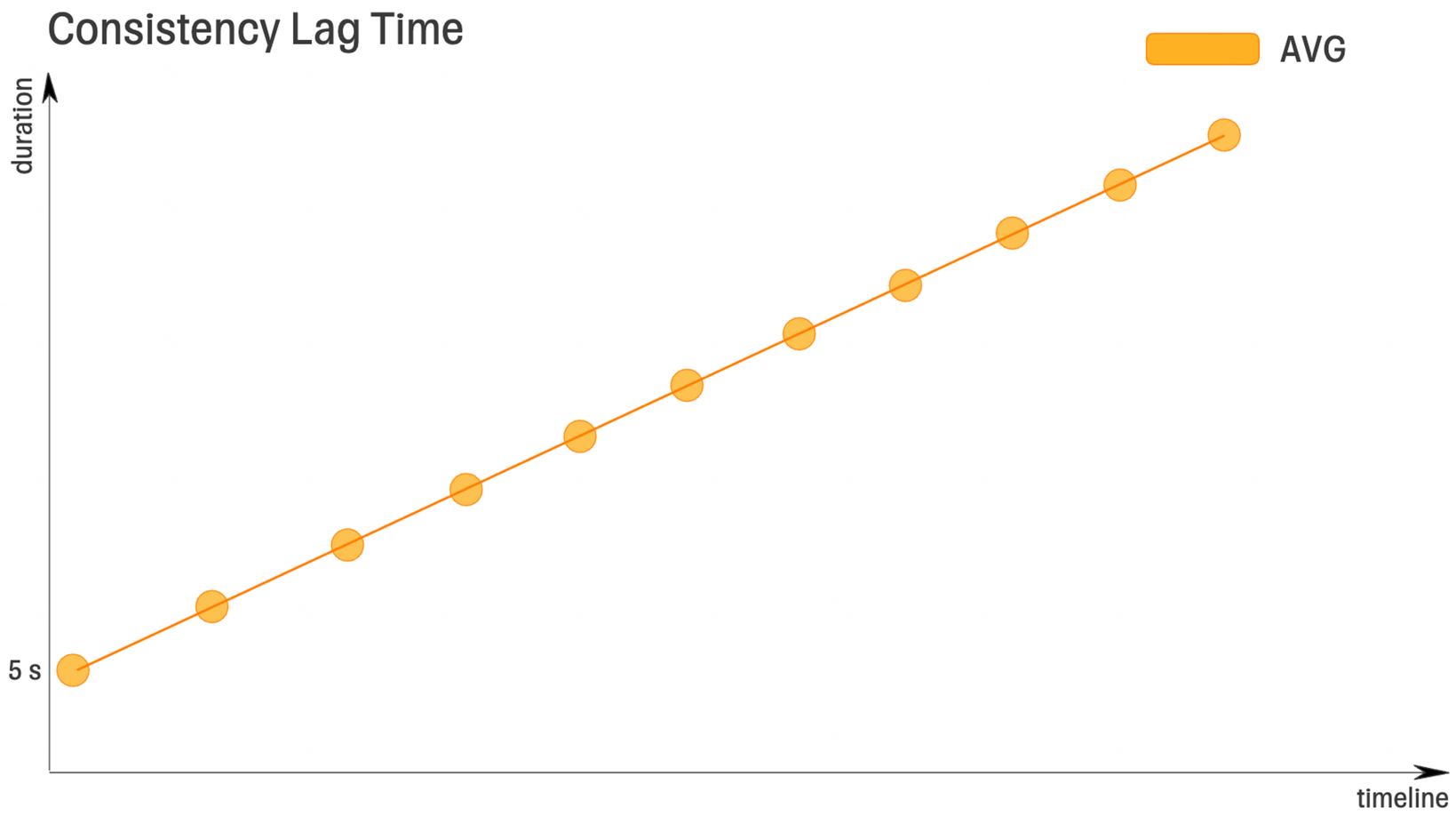
Смотрим. Красивое

- На pod по партиции
- Latency external system avg 300ms



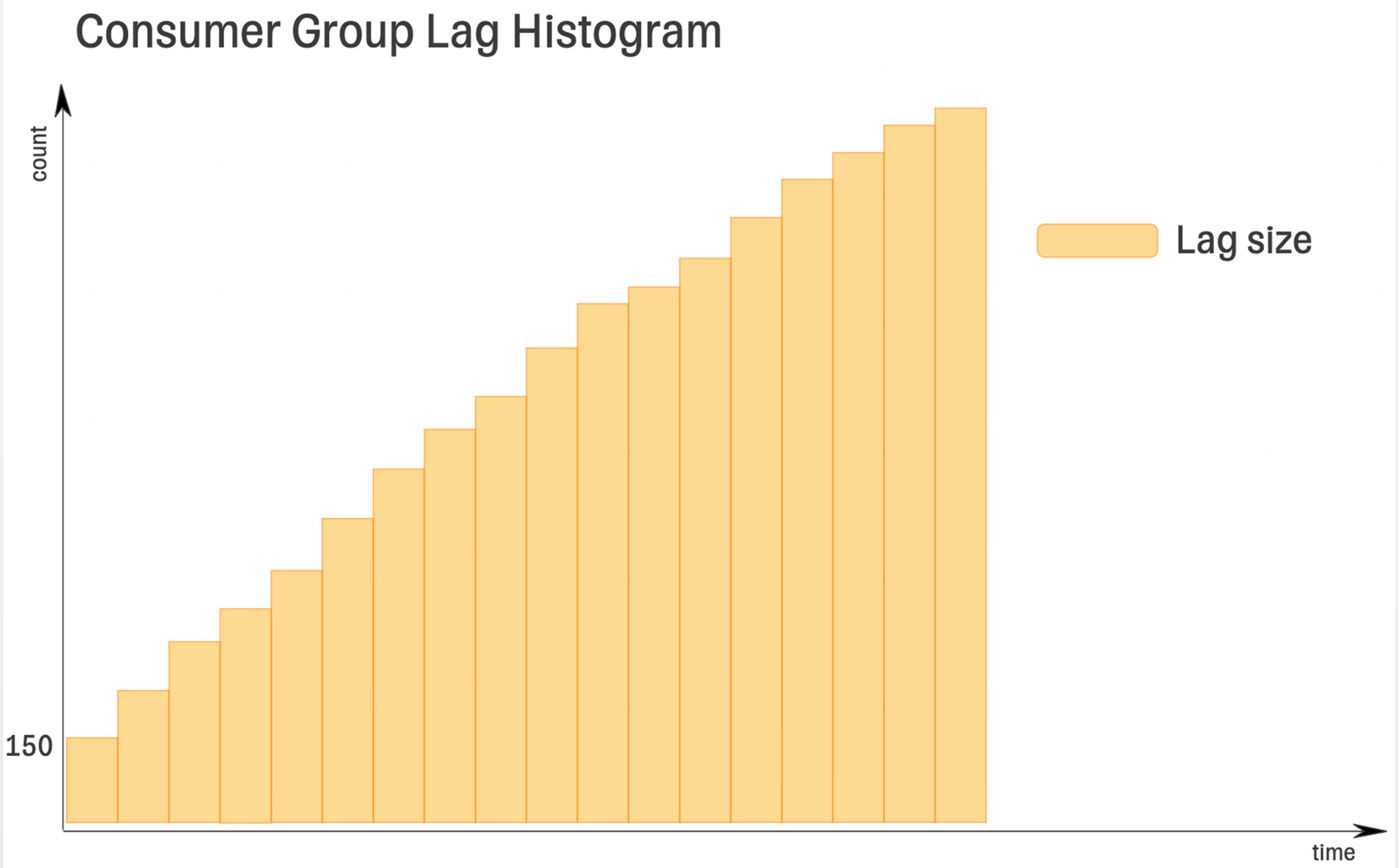
Смотрим. Некрасивое

- На pod по партиции
- Latency external system avg 300ms
- Consistancy lag avg 5sec и растет



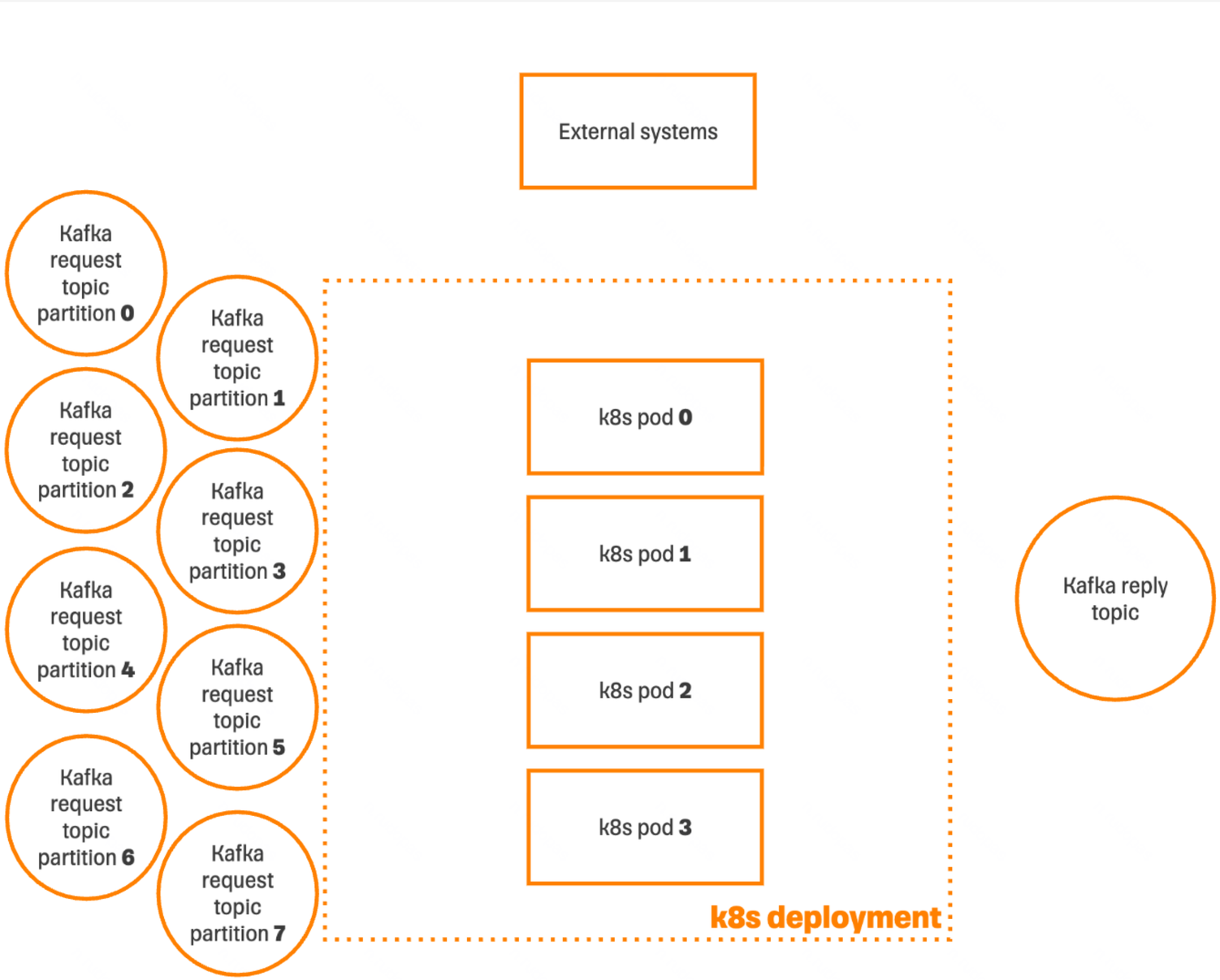
Смотрим. Некрасивое

- На pod по партиции
- Latency external system avg 300ms
- Consistancy lag avg 5sec и растет
- Consumer group lag 150 и растет



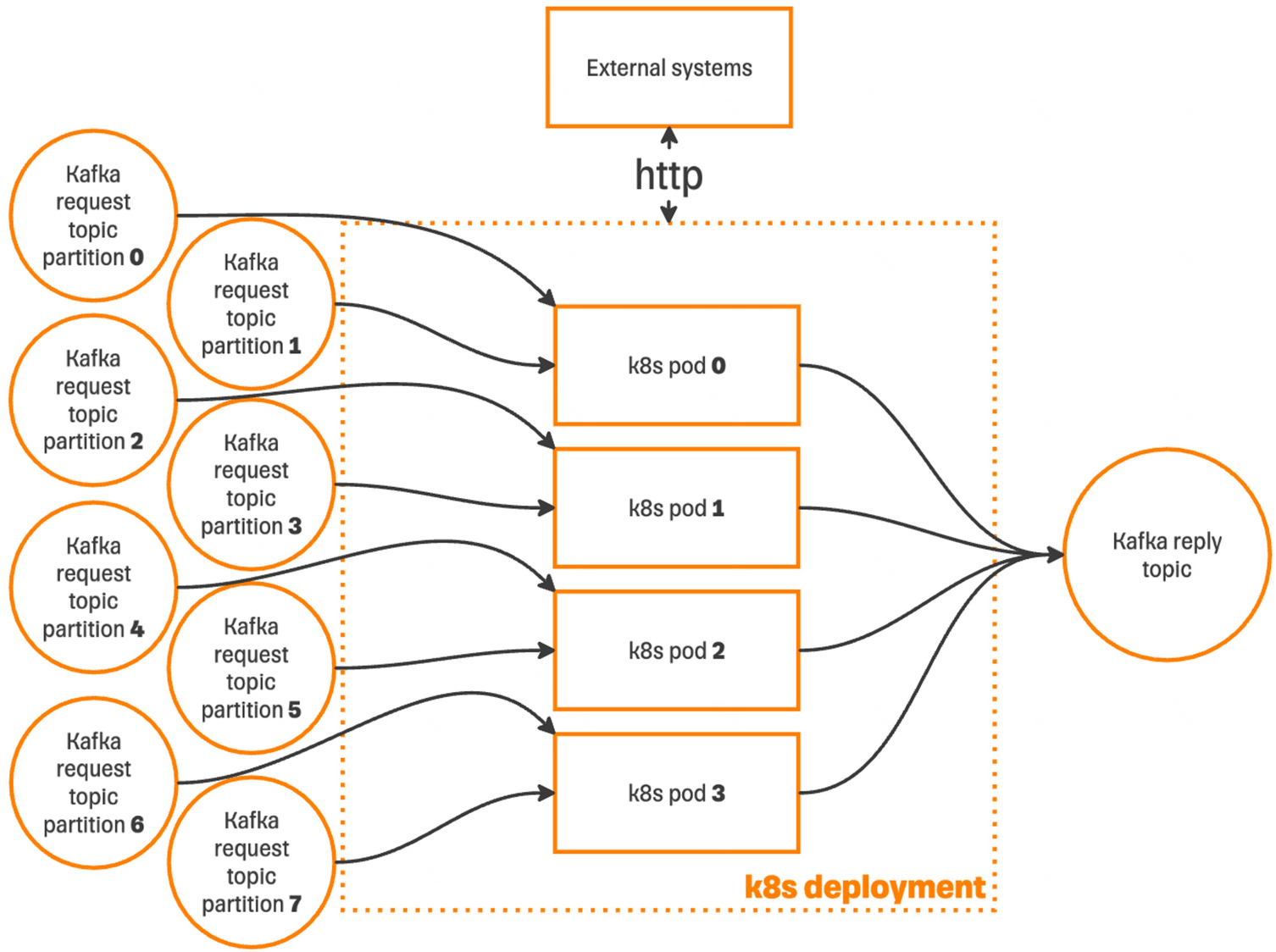
Недолго думаем

Штош


External systems
Kafka request topic partition 0
Kafka request topic partition 1
Kafka request topic partition 2
Kafka request topic partition 3
Kafka request topic partition 4
Kafka request topic partition 5
Kafka request topic partition 6
Kafka request topic partition 7
k8s pod 0
k8s pod 1
k8s pod 2
k8s pod 3
k8s deployment
Kafka reply topic

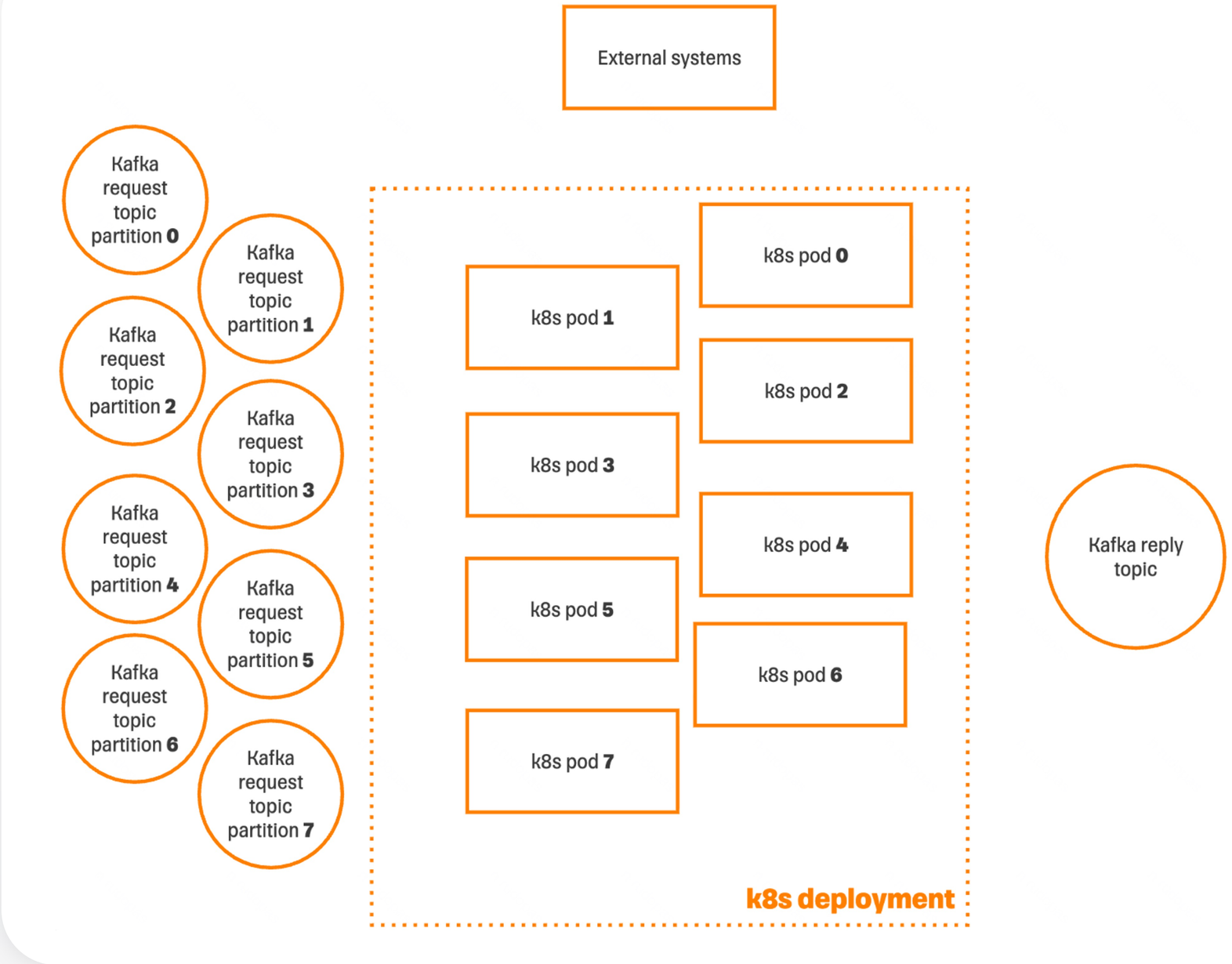
Помогло?

Штош


External systems
http
Kafka request topic partition 0
Kafka request topic partition 1
Kafka request topic partition 2
Kafka request topic partition 3
Kafka request topic partition 4
Kafka request topic partition 5
Kafka request topic partition 6
Kafka request topic partition 7
k8s pod 0
k8s pod 1
k8s pod 2
k8s pod 3
k8s deployment
Kafka reply topic

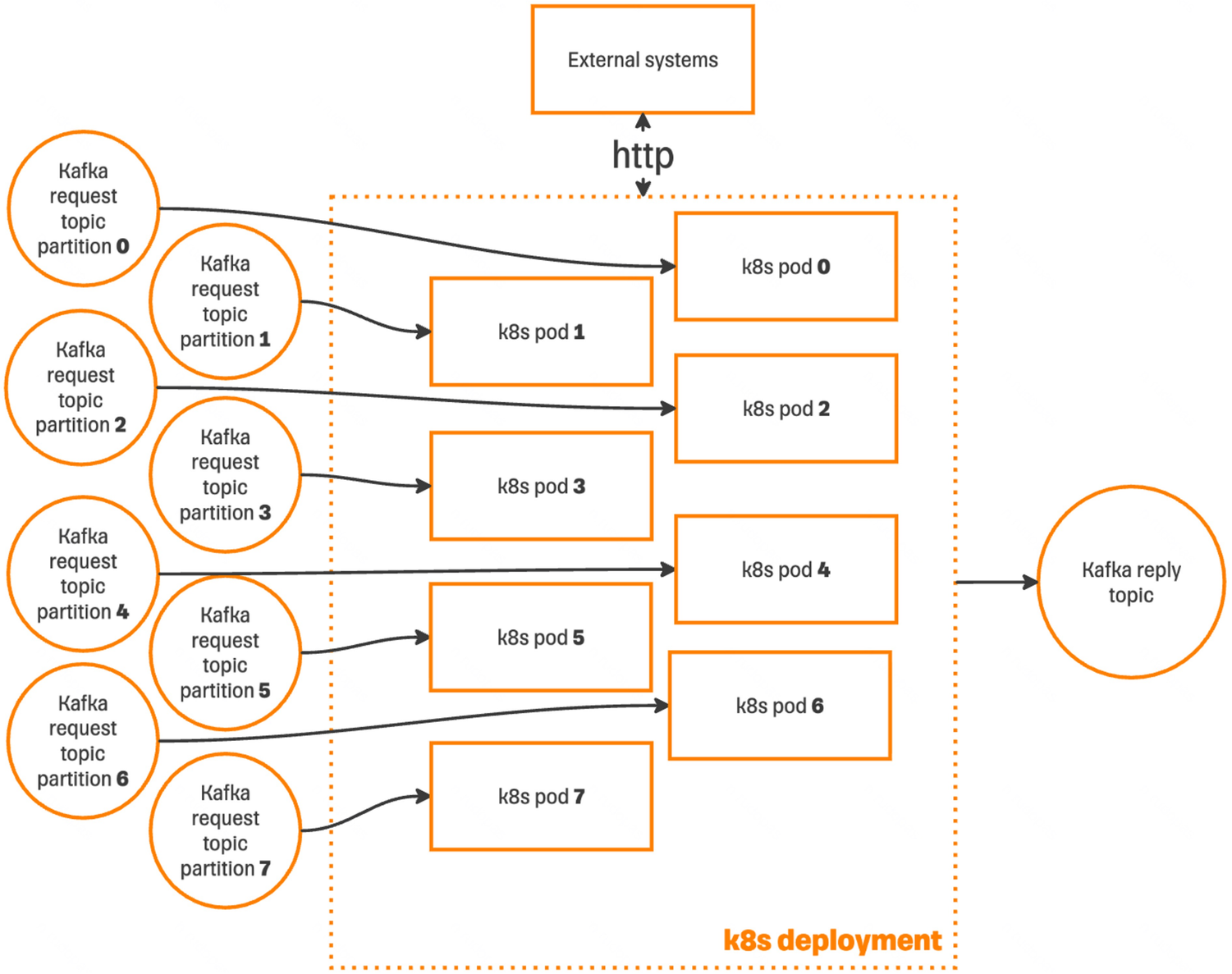
Еще недолго думаем

Еще раз


External systems
Kafka request topic partition 0
Kafka request topic partition 1
Kafka request topic partition 2
Kafka request topic partition 3
Kafka request topic partition 4
Kafka request topic partition 5
Kafka request topic partition 6
Kafka request topic partition 7
k8s pod 0
k8s pod 1
k8s pod 2
k8s pod 3
k8s pod 4
k8s pod 5
k8s pod 6
k8s pod 7
k8s deployment
Kafka reply topic

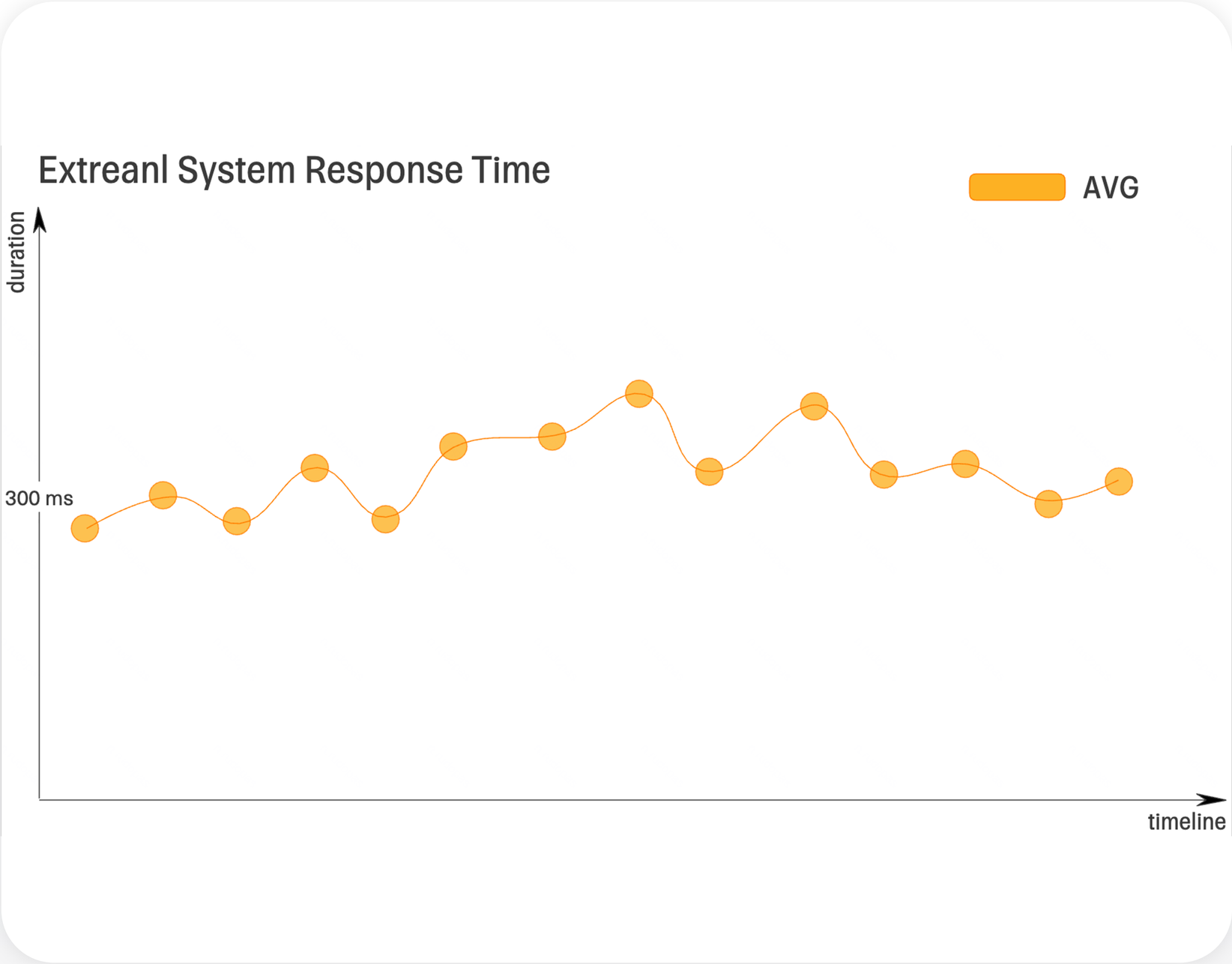
Штош. Красивое

- На pod по партиции



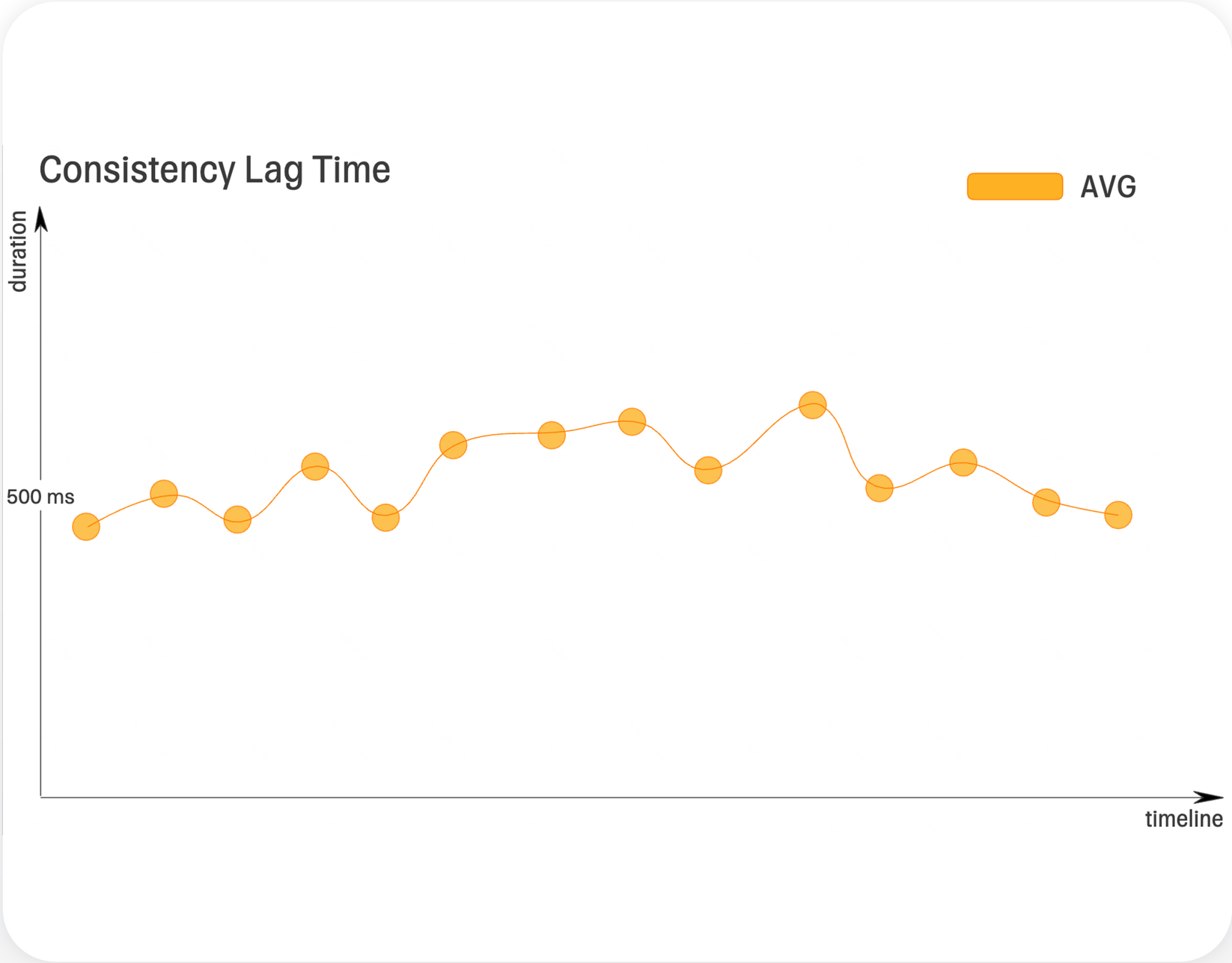
Штош. Красивое

- На pod по партиции
- Latency external system avg 300ms



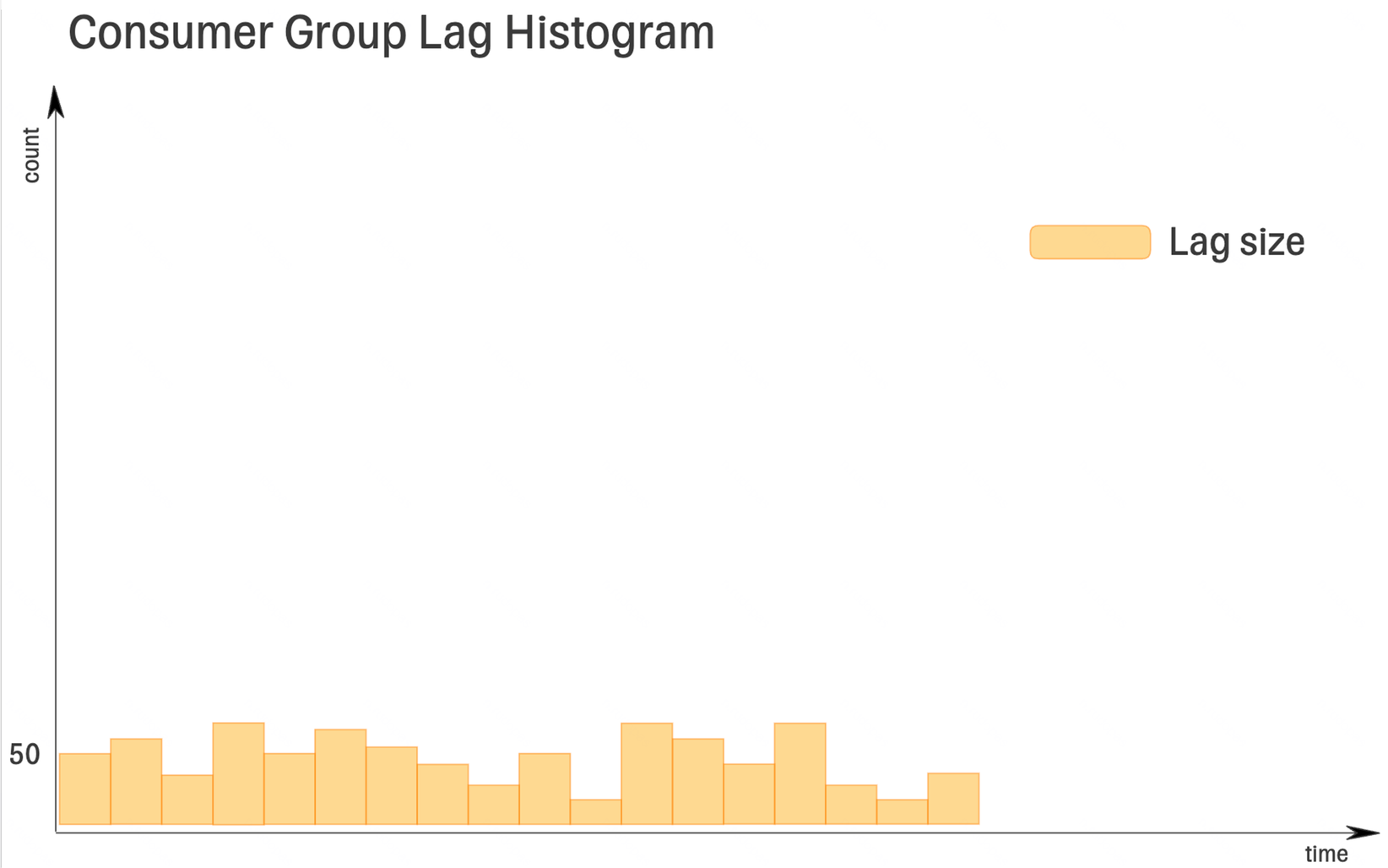
Штош. Красивое

- На pod по партиции
- Latency external system avg 300ms
- Consistancy lag avg 500ms



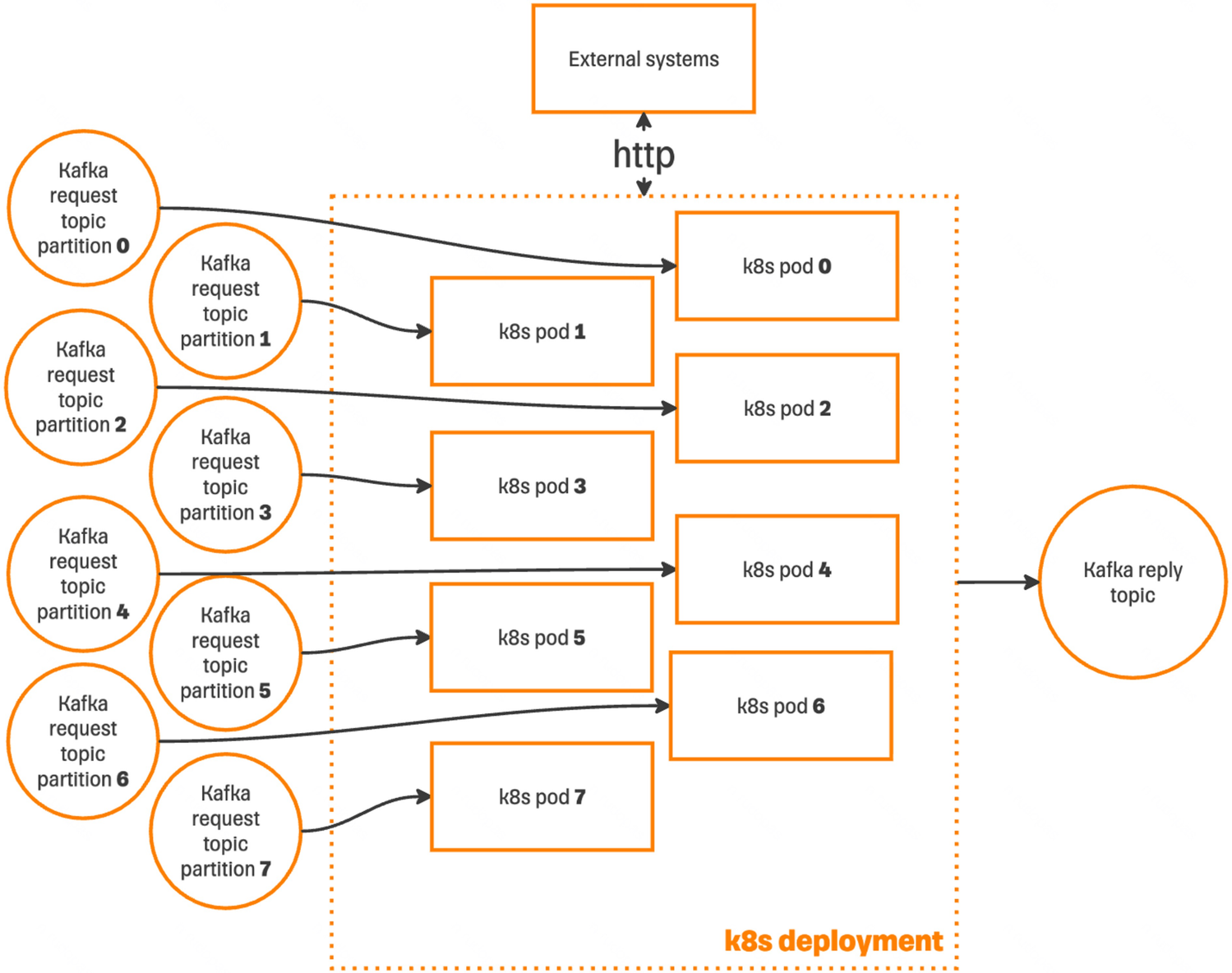
Штош. Красивое

- На pod по партиции
- Latency external system avg 300ms
- Consistancy lag avg 500ms
- Consumer group lag 50



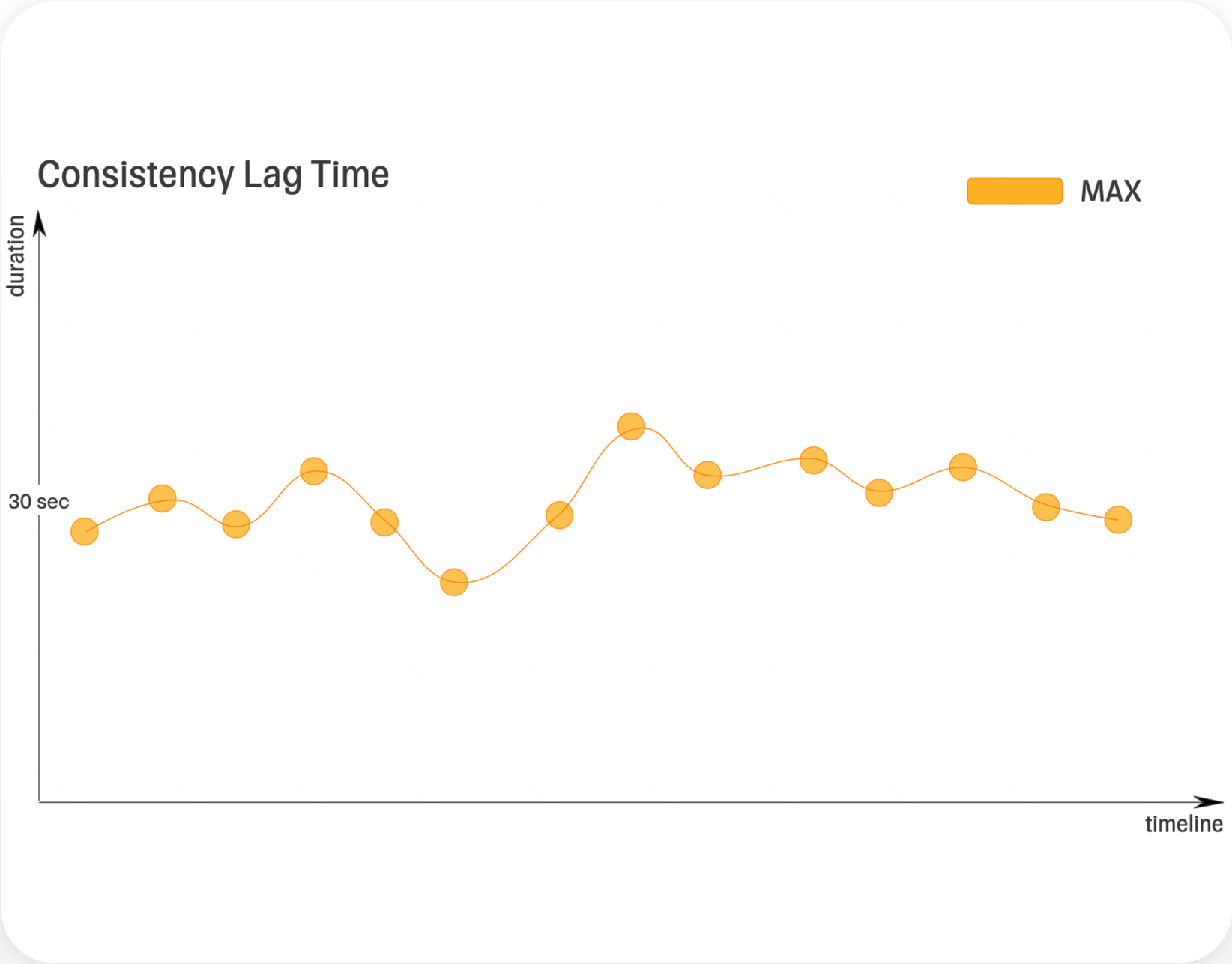
Это победа?

AVG-то хорошо



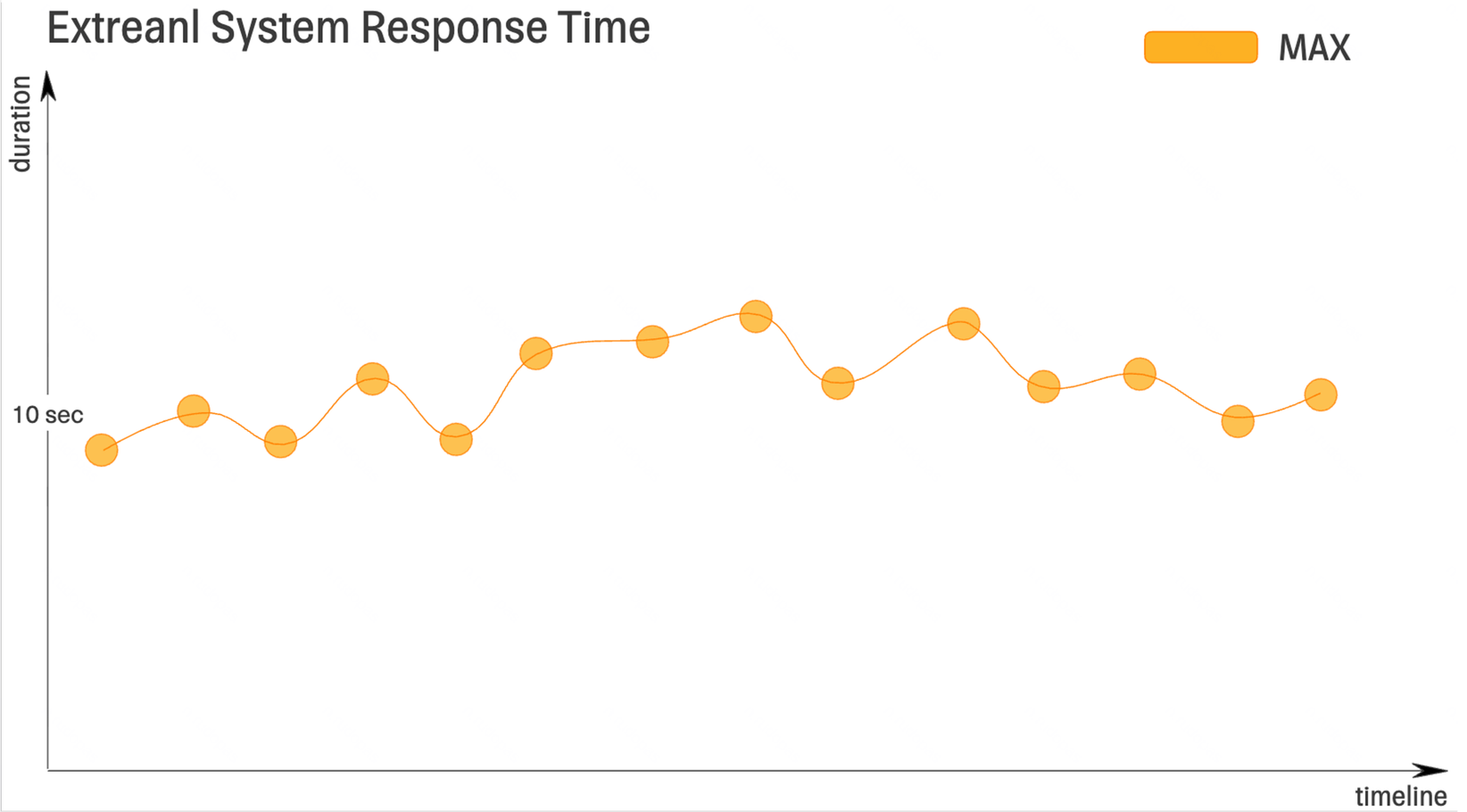
AVG-то хорошо, но что-то не так

- Consistancy lag max 30 sec



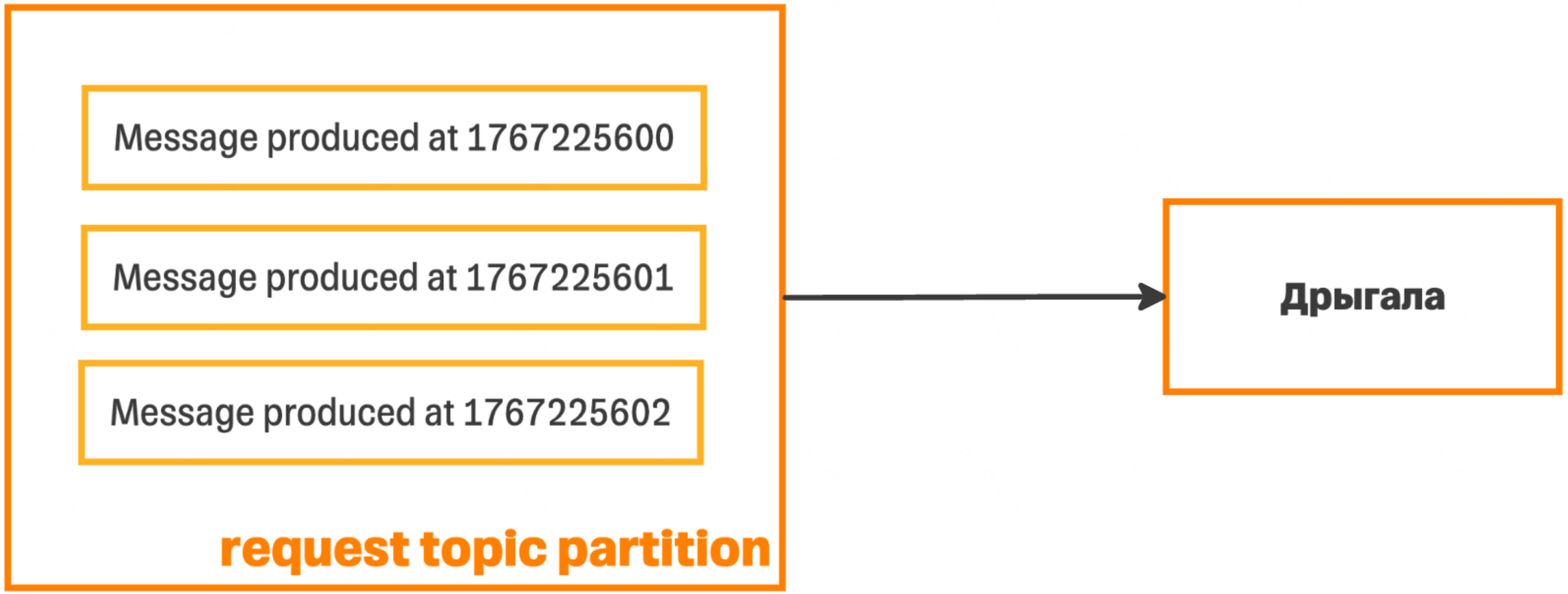
AVG хорошо, но совсем не так

- Consistancy lag max 30 sec
- Latency external system max 10sec

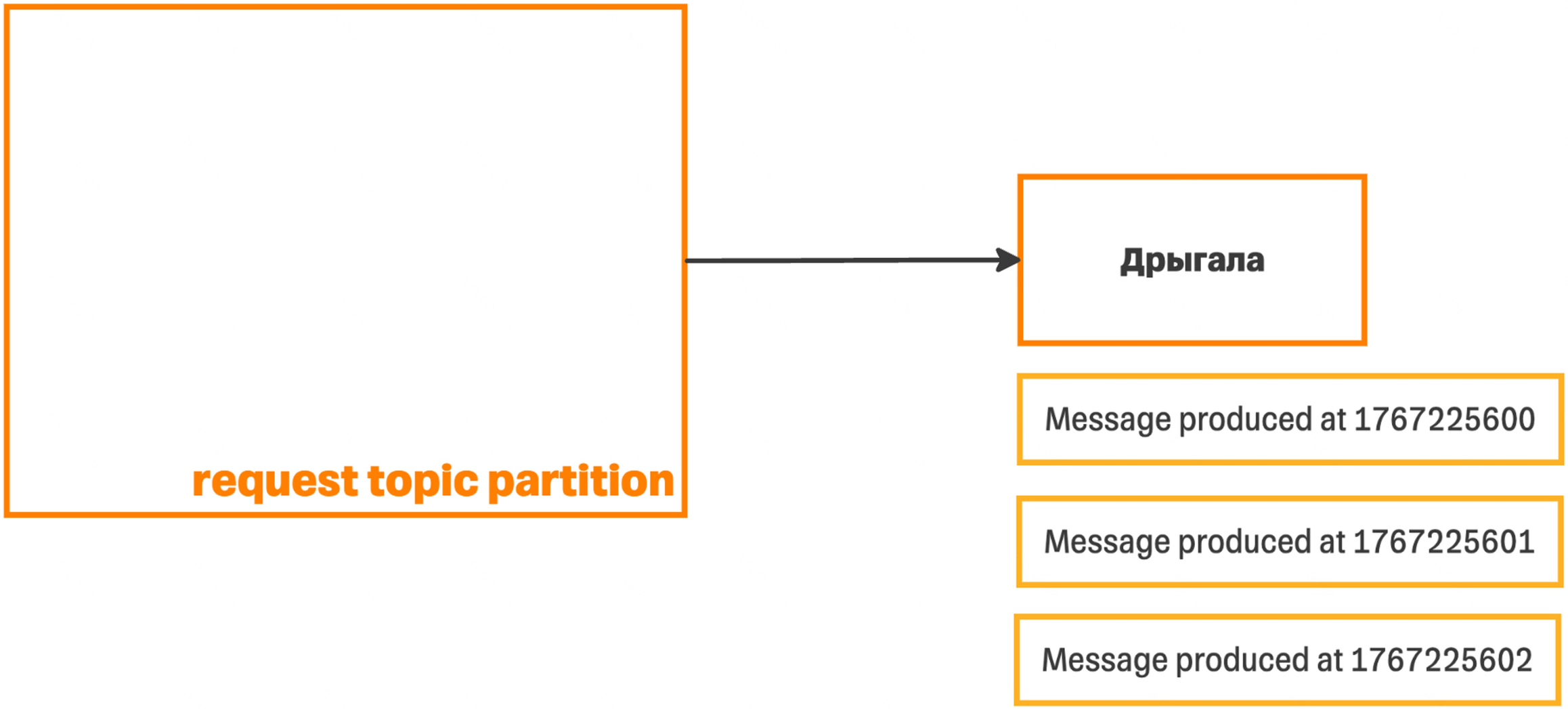


Что происходит?!

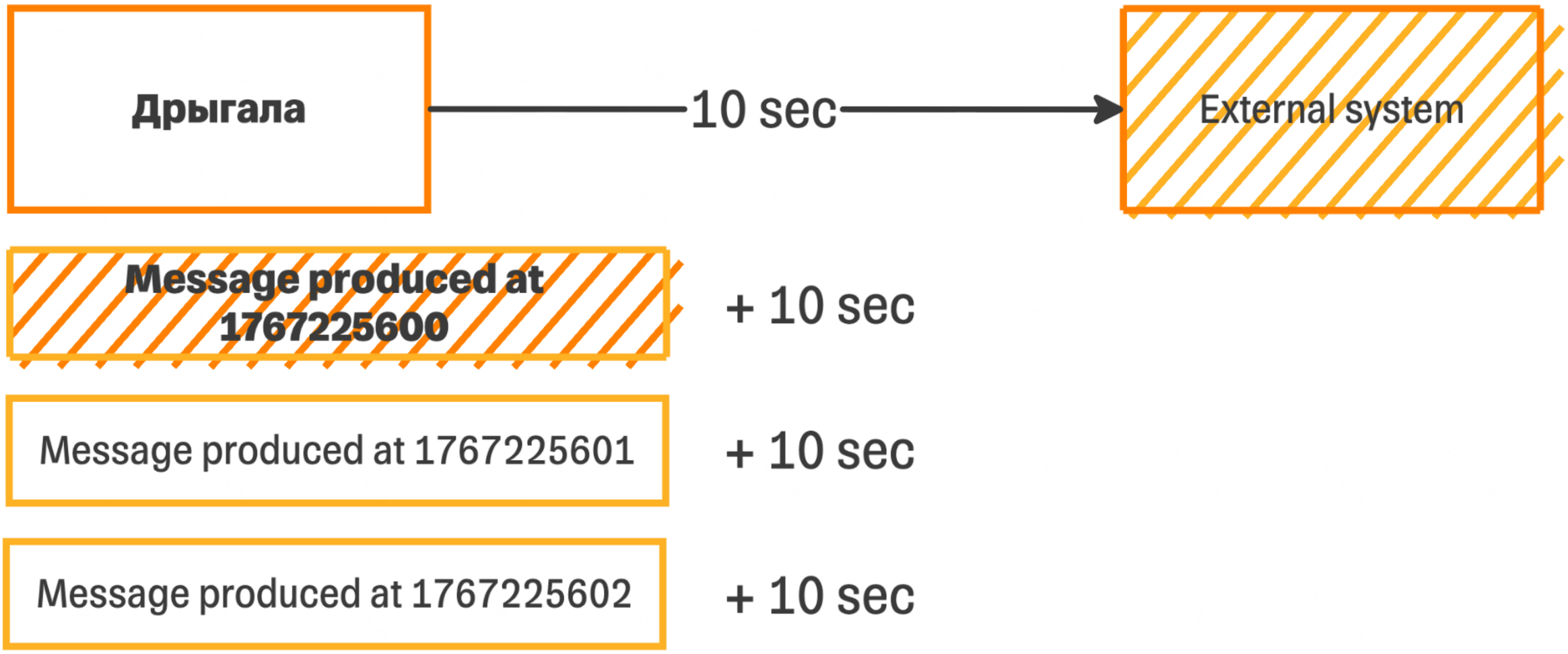
Как копится лаг



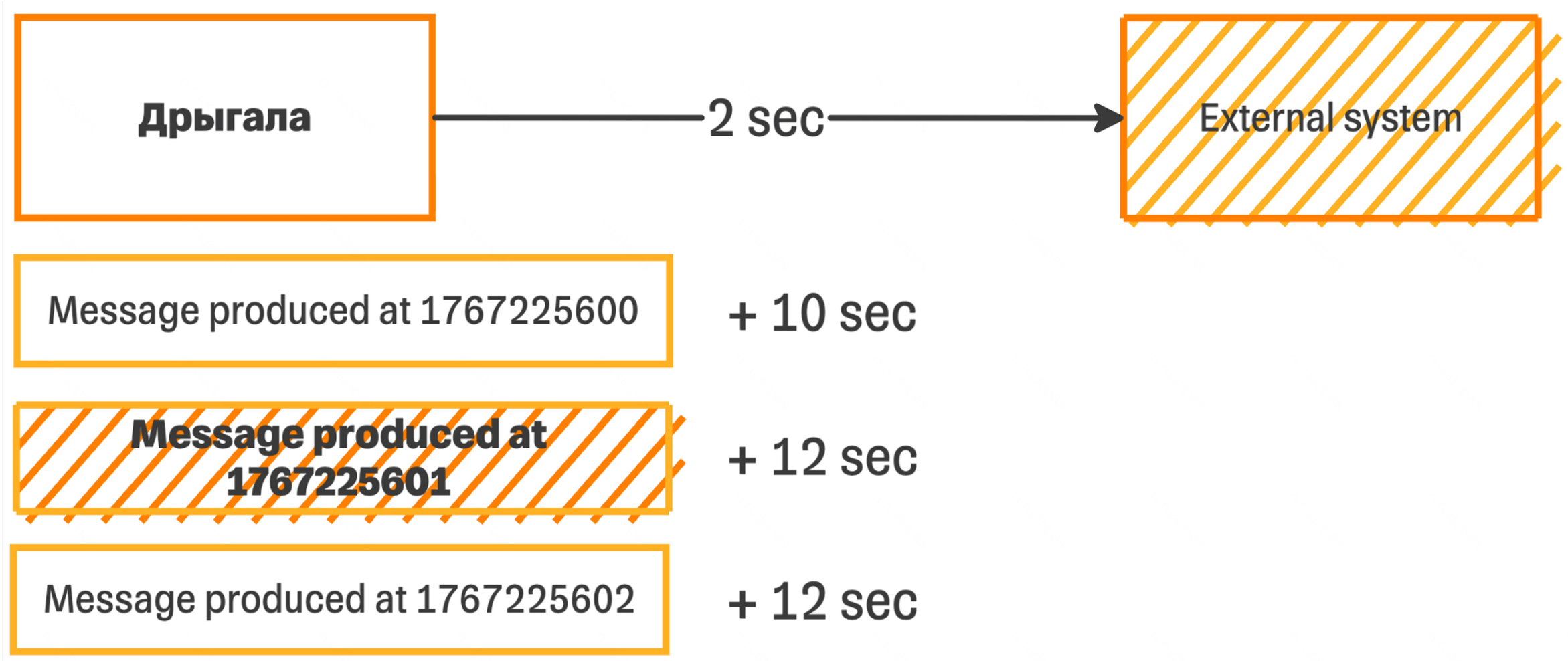
Как копится лаг



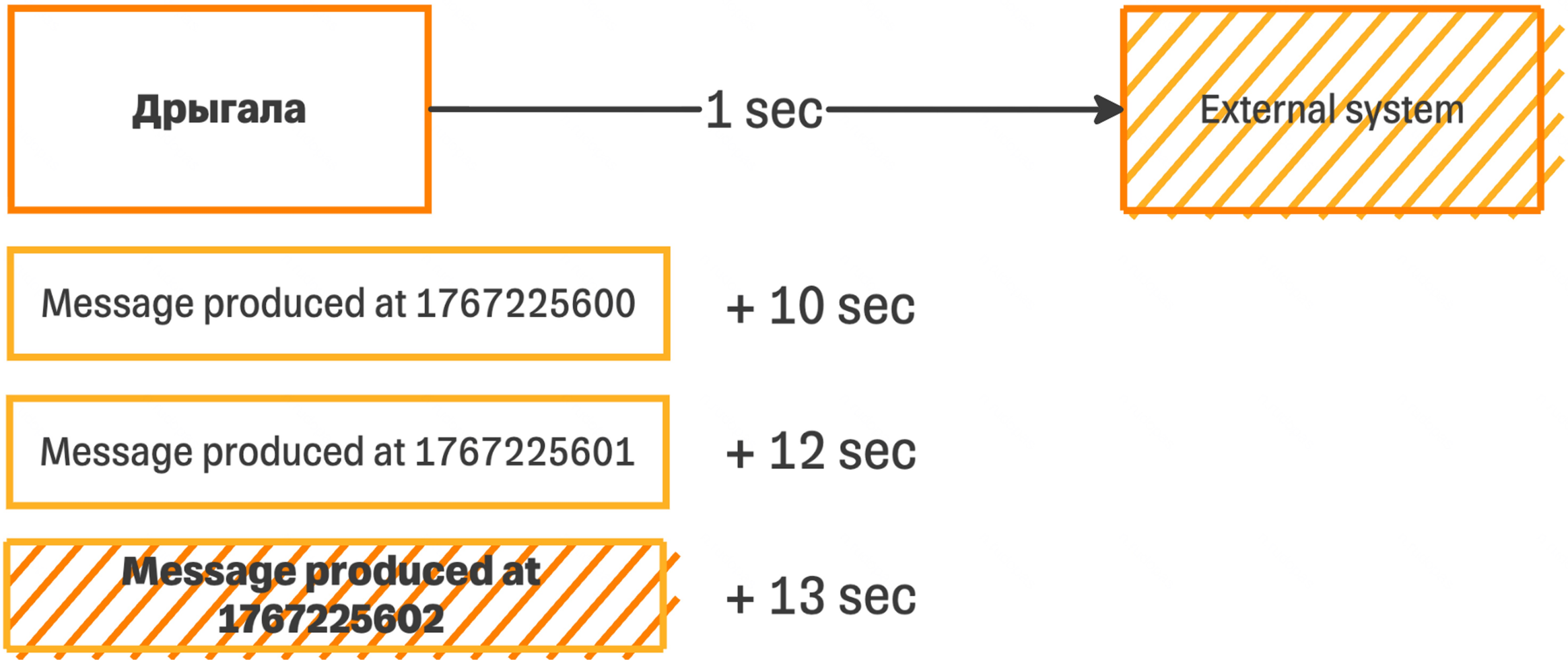
Как копится лаг



Как копится лаг

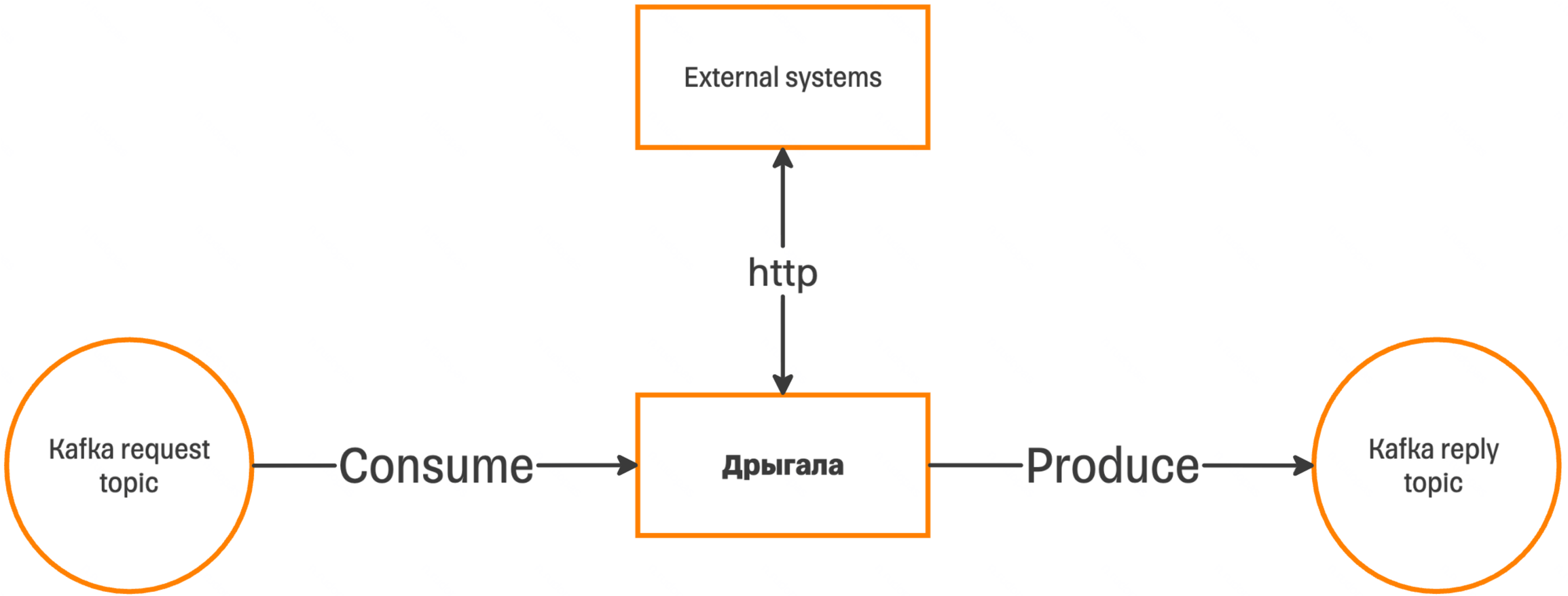


Как копится лаг

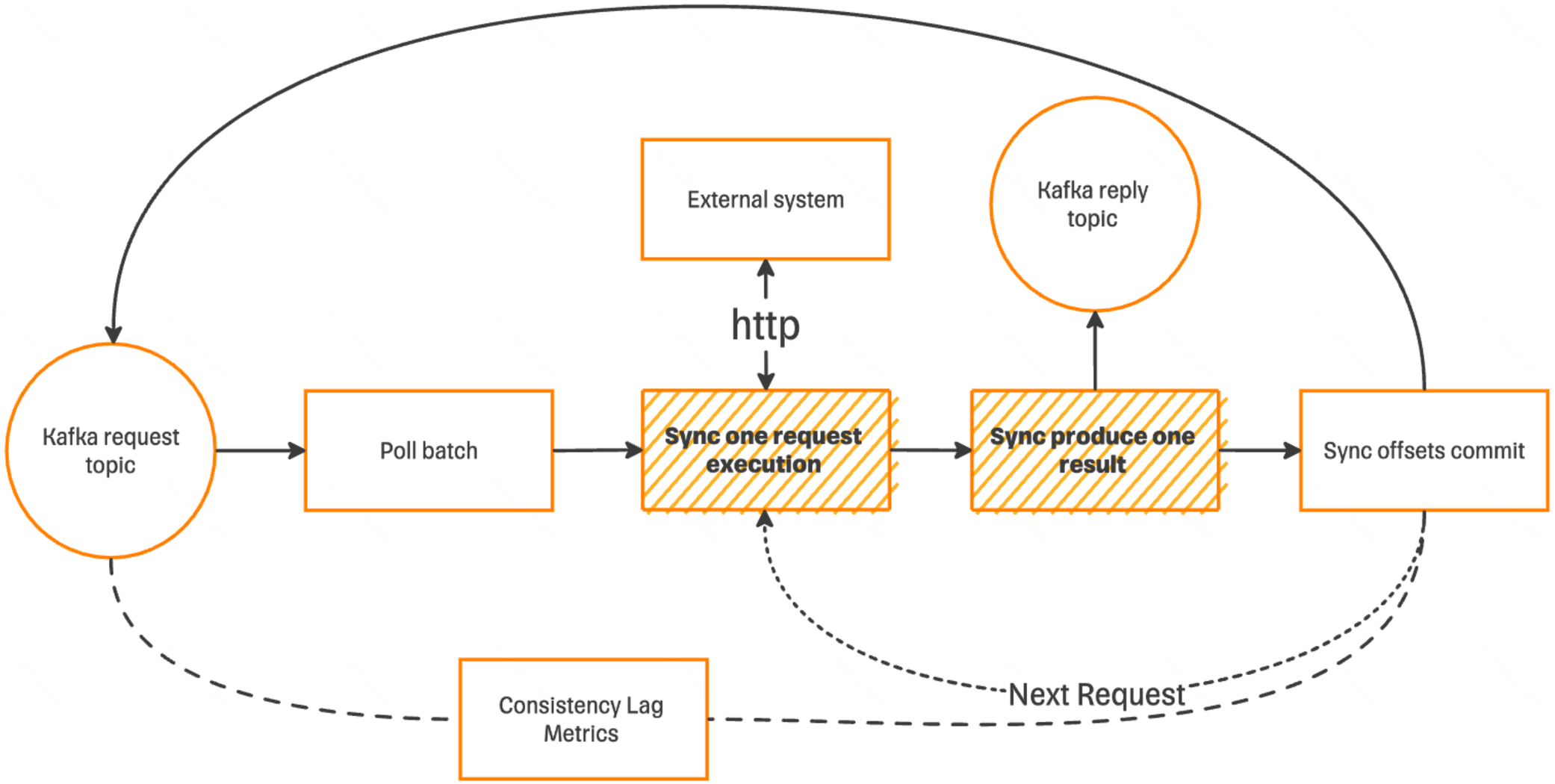


Думаем и смотрим докладики

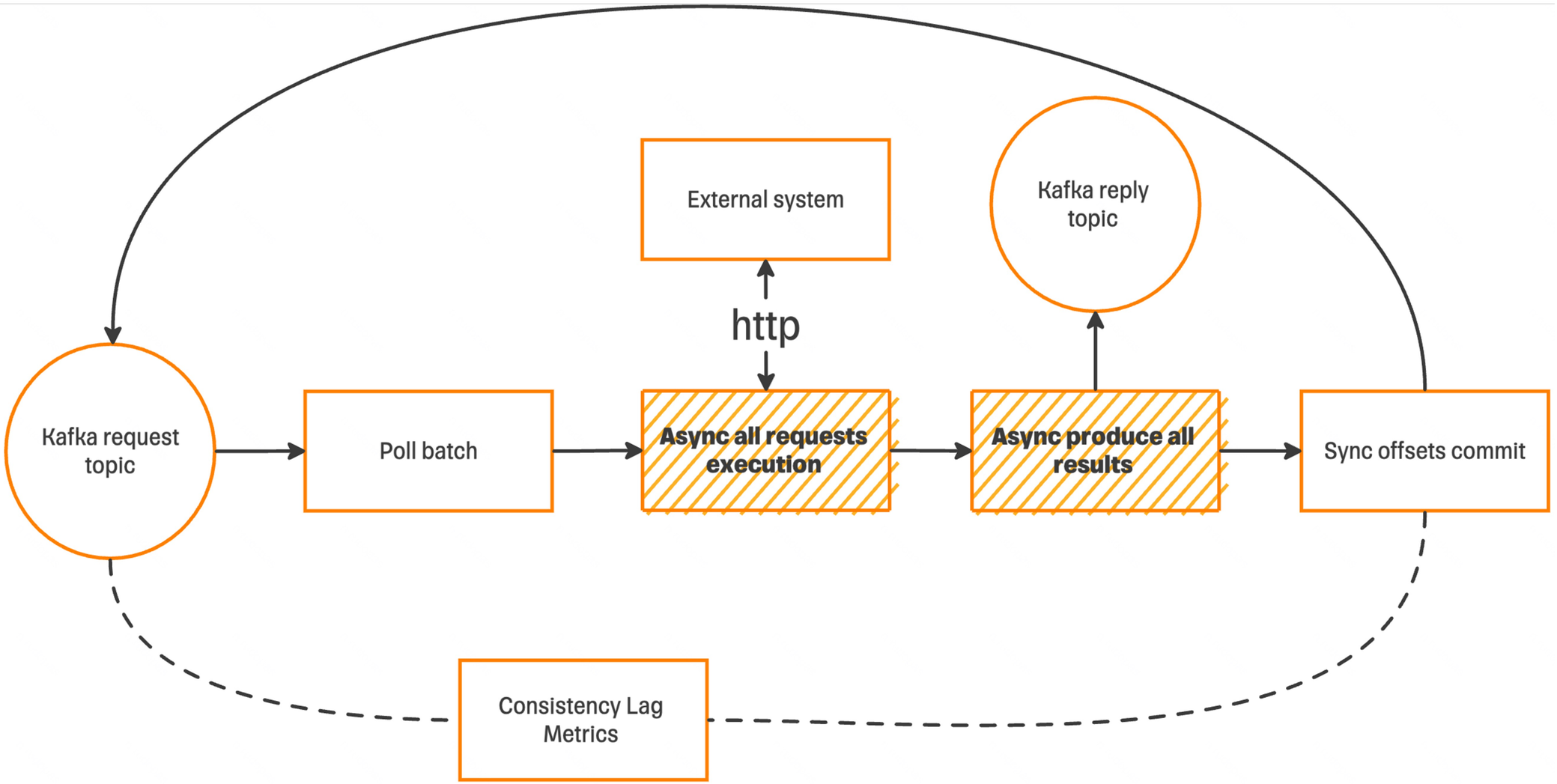
Все еще нужна дрыгала



Было


External system
Kafka reply topic
http
Kafka request topic
Poll batch
Sync one request execution
Sync produce one result
Sync offsets commit
Consistency Lag Metrics
Next Request

Стало


External system
Kafka reply topic
http
Kafka request topic
Poll batch
Async all requests execution
Async produce all results
Sync offsets commit
Consistency Lag Metrics

Обрабатываем

Async all requests execution

```
21  fun processRequests(
22      requests: Iterable<ConsumerRecord<String, Request>>,
23      batchDispatcher: CoroutineDispatcher,
24  ): List<Pair<String, Response>> =
25      runBlocking(context = batchDispatcher) {
26          requests
27              .map {
28                  async {
29                      it.key() to getResponse(it.value())
30                  }
31              }
32              .awaitAll()
33      }
```

```
31  fun startParallelProcessing(
32      consumer: KafkaConsumer<String, Request>,
33      requestTopic: String,
34      pollTimeout: Duration,
35      batchDispatcher: CoroutineDispatcher
36  ) {
37      // Подписсываемся на топик
38      consumer.subscribe(listOf(requestTopic))
39      while (true) {
40          // Полим батч
41          val requests = consumer.poll(pollTimeout)
42
43          val responses = processRequests(
44              requests,
45              batchDispatcher
46          )
47      }
48  }
```

Отправляем

Async produce all results

```kotlin
fun produceResponses(
    responses: List<Pair<String, Response>>,
    responseTopic: String,
    producer: KafkaProducer<String, Response>,
) {
    responses
        .map { response ->
            producer.send(
                ProducerRecord(
                    responseTopic,
                    response.first,
                    response.second
                )
            )
        }.forEach {
            it.get()
        }
}
```

```kotlin
) {
    // Подписсываемся на топик
    consumer.subscribe(listOf(requestTopic))
    while (true) {
        // Полим батч
        val requests = consumer.poll(pollTimeout)

        val responses = processRequests(
            requests,
            batchDispatcher
        )

        produceResponses(
            responses,
            responseTopic,
            producer
        )
    }
}
```

Сдвигаем оффсеты

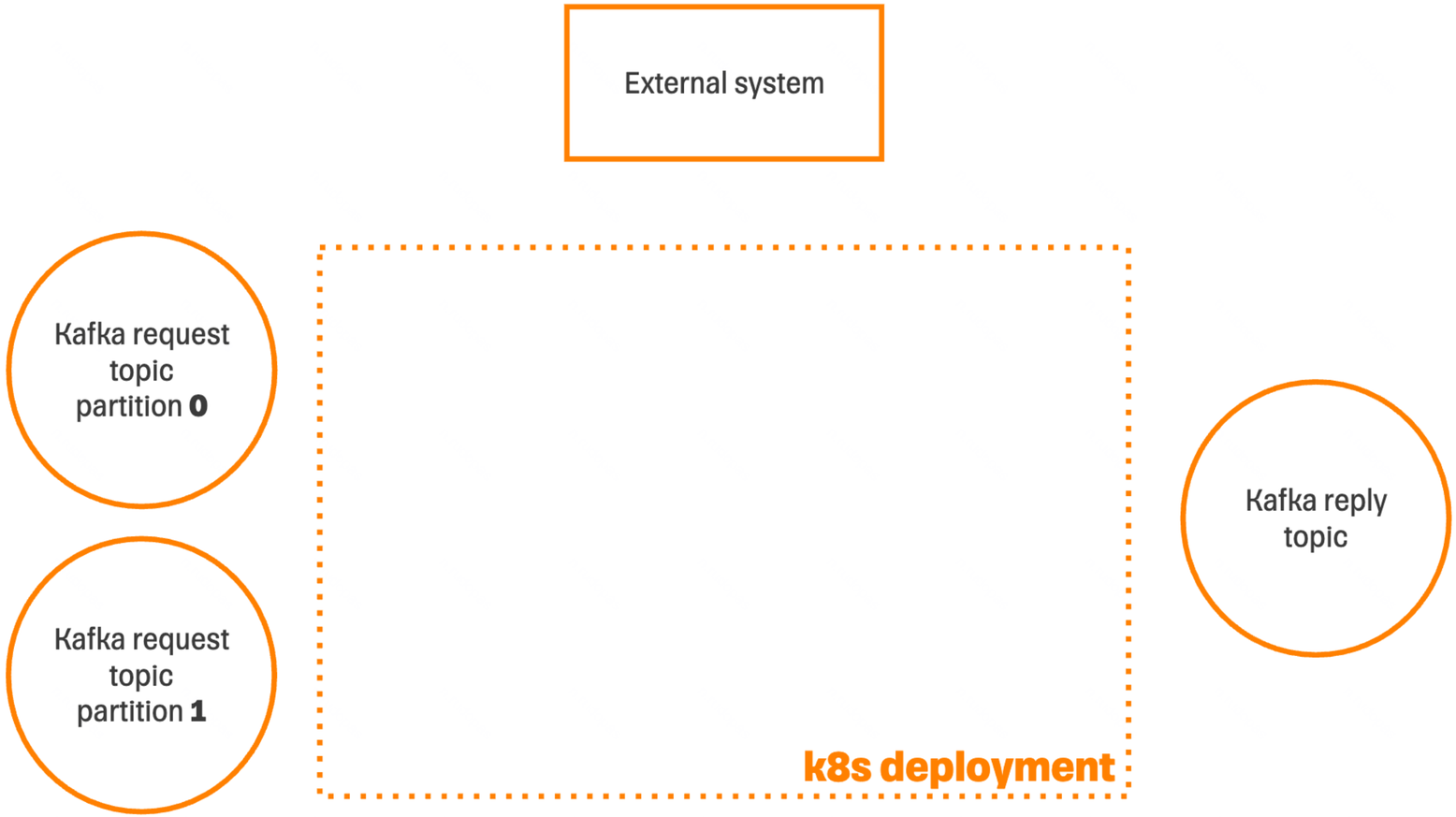
Sync offsets commit

```
48  fun commitOffsets(
49      consumer: KafkaConsumer<String, Request>,
50      consumerRecords: Iterable<ConsumerRecord<String, Request>>
51  ) {
52      val offsets = consumerRecords.groupingBy {
53          TopicPartition(it.topic(), it.partition())
54      }.fold(
55          initialValueSelector = { _, it ->
56              OffsetAndMetadata( offset: it.offset() + 1)
57          },
58          operation = { _, _, it ->
59              OffsetAndMetadata( offset: it.offset() + 1)
60          }
61      )
62      consumer.commitSync(offsets)
63  }
```

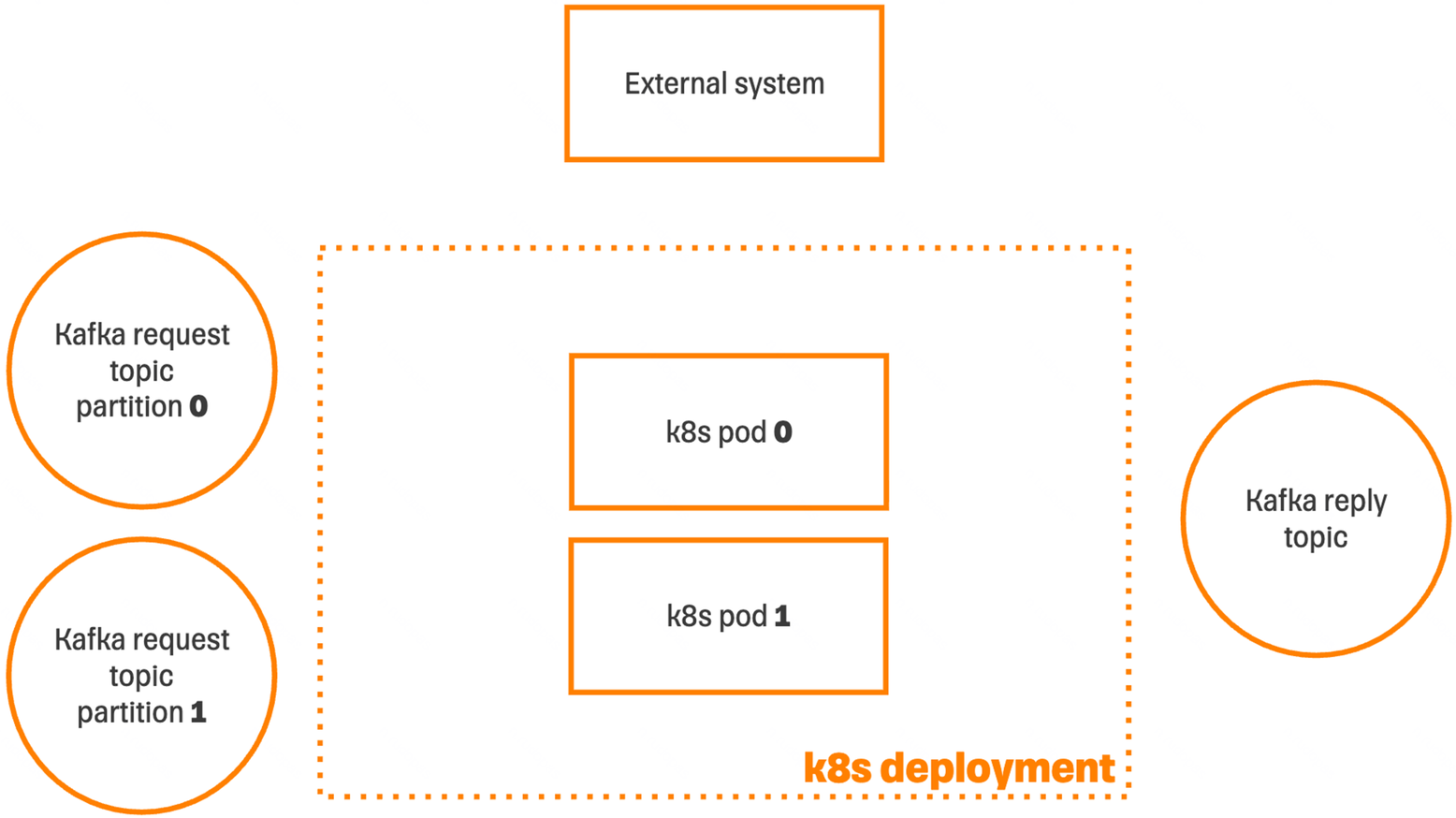
```
85      ) {
86          // Подписываемся на топик
87          consumer.subscribe(listOf(requestTopic))
88          while (true) {
89              // Полим батч
90              val requests = consumer.poll(pollTimeout)
91
92              val responses = processRequests(
93                  requests,
94                  batchDispatcher
95              )
96
97              produceResponses(
98                  responses,
99                  responseTopic,
100                 producer
101             )
102
103             commitOffsets(consumer, requests)
104
105             logMetrics(requests)
```

Собираем и деплоим

Деплоим на QA

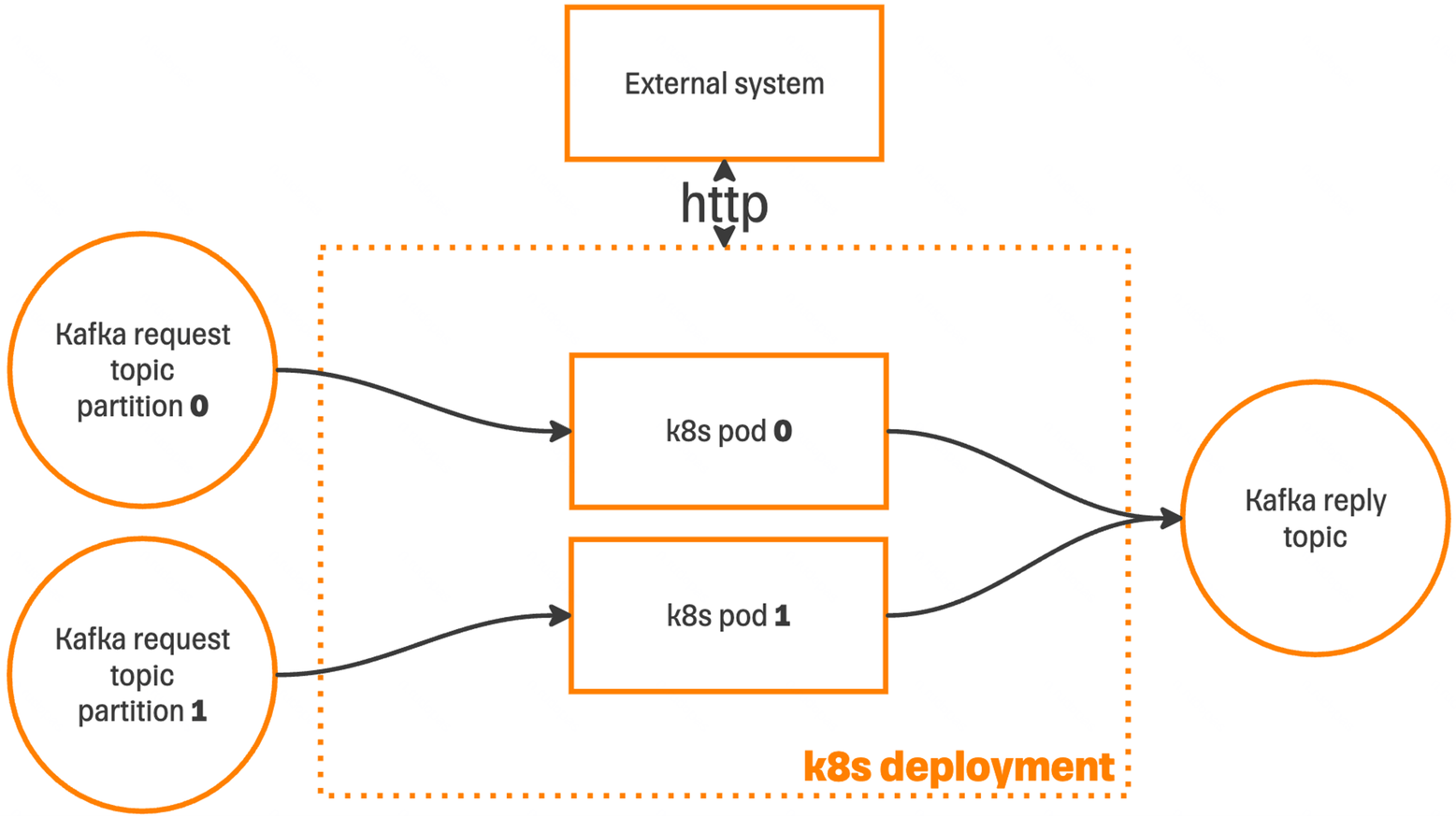


Деплоим на QA



Деплоим на QA

- На pod по партиции

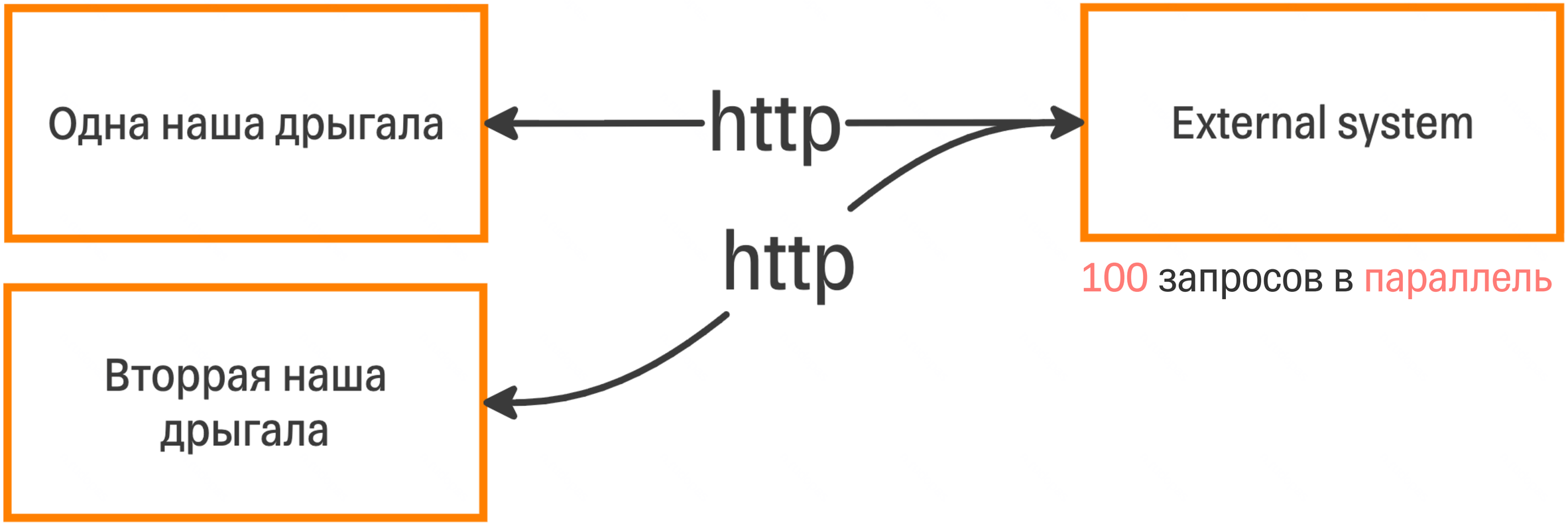


И нас уже ищут

Перестарались

50 запросов в параллель

Перестарались

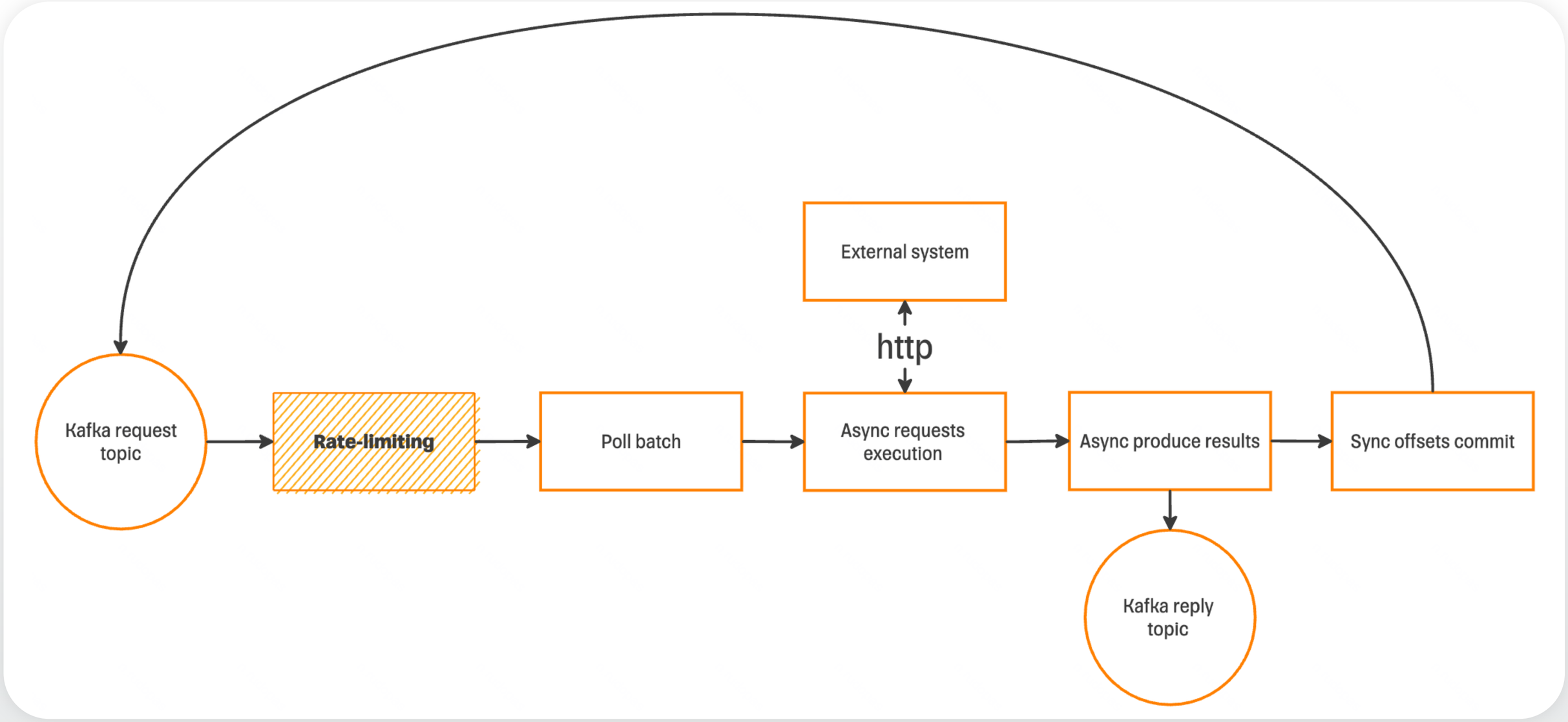


100 запросов в параллель

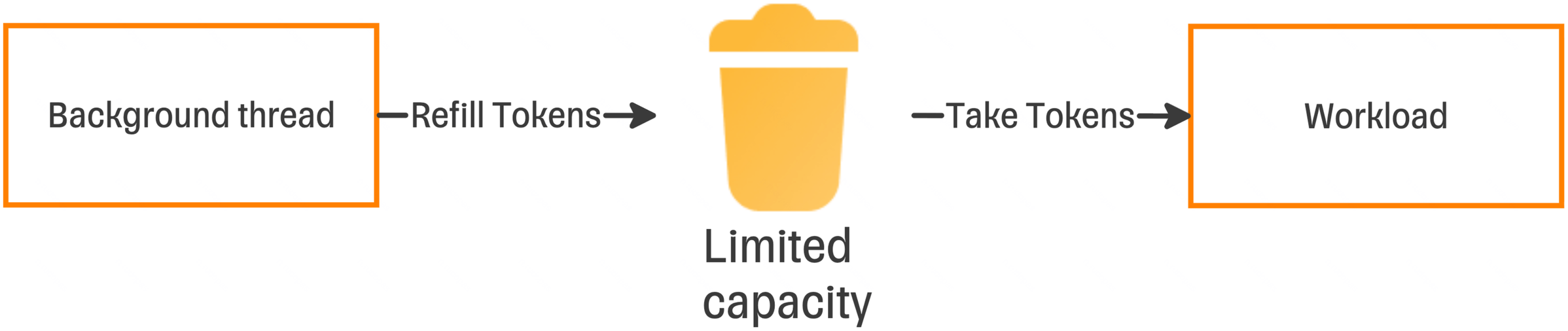
External system прилег

Штош, умерим пыл

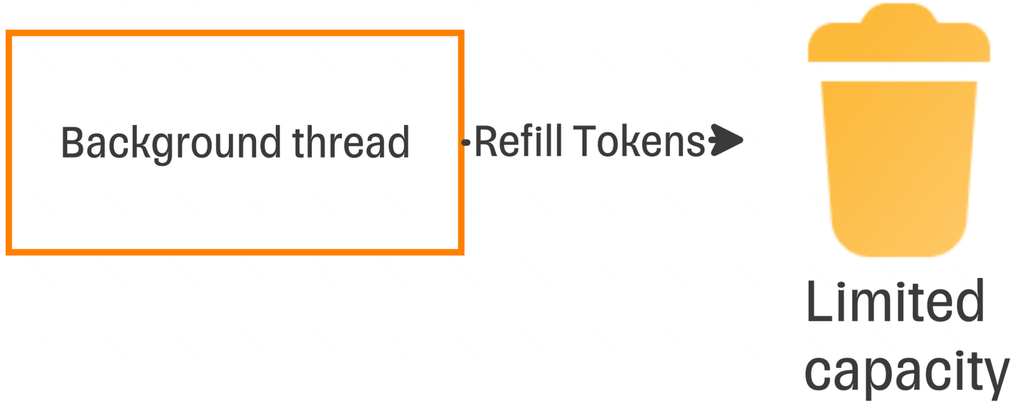
Rate-limiting



Bucket4j

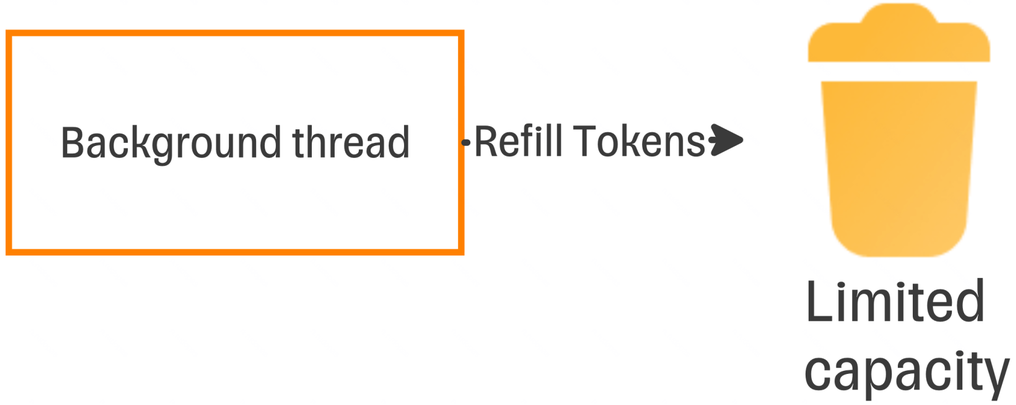

Background thread
Refill Tokens
Limited capacity
Take Tokens
Workload

Делаем корзинку



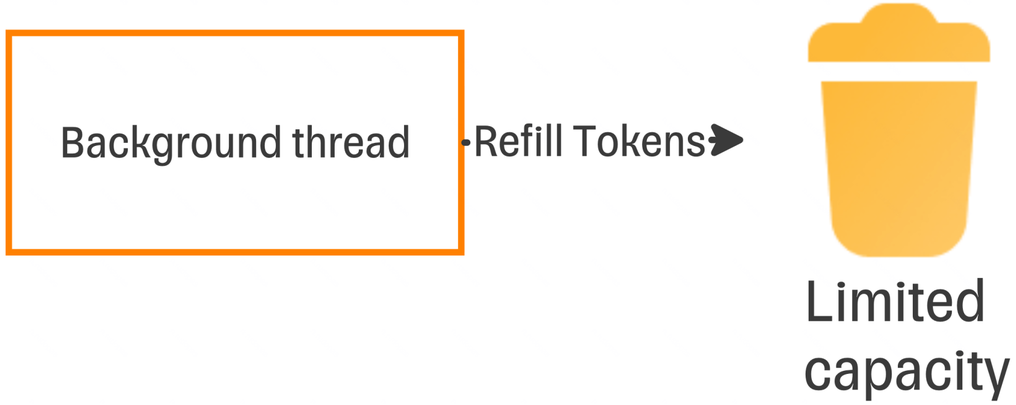

```
22  fun getBucket(
23      maxCapacity: Long,
24      initialCapacity: Long = 0,
25      refillTokens: Long,
26      refillPeriod: Duration,
27  ): Bucket =
28      Bucket.builder().addLimit { limit ->
29          limit
30              // макимальный размер корзины
31              .capacity(maxCapacity)
32              // стратегия и параметры пополнения
33              // (как, на сколько, как часто)
34              .refillGreedy(refillTokens, refillPeriod)
35              // начальный размер корзины
36              .initialTokens(initialCapacity)
37      }.build()
38
39  fun getBucket(maxPollRecords: Long): Bucket =
40      getBucket(
41          maxCapacity = 2 * maxPollRecords,
42          refillTokens = 20,
43          refillPeriod = Duration.ofSeconds( seconds: 1)
44      )
```

Делаем корзинку



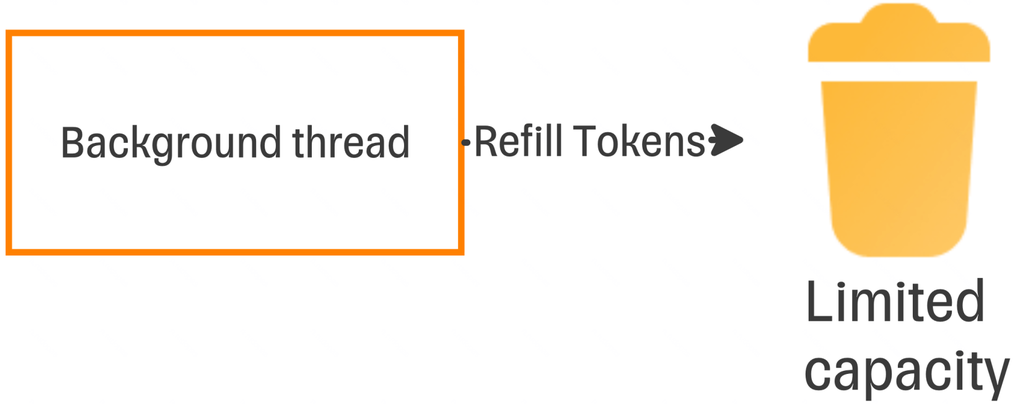

```
22  fun getBucket(
23      maxCapacity: Long,
24      initialCapacity: Long = 0,
25      refillTokens: Long,
26      refillPeriod: Duration,
27  ): Bucket =
28      Bucket.builder().addLimit { limit ->
29          limit
30              // макимальный размер корзины
31              .capacity(maxCapacity)
32              // стратегия и параметры пополнения
33              // (как, на сколько, как часто)
34              .refillGreedy(refillTokens, refillPeriod)
35              // начальный размер корзины
36              .initialTokens(initialCapacity)
37      }.build()
38
39  fun getBucket(maxPollRecords: Long): Bucket =
40      getBucket(
41          maxCapacity = 2 * maxPollRecords,
42          refillTokens = 20,
43          refillPeriod = Duration.ofSeconds( seconds: 1)
44      )
```

Делаем корзинку



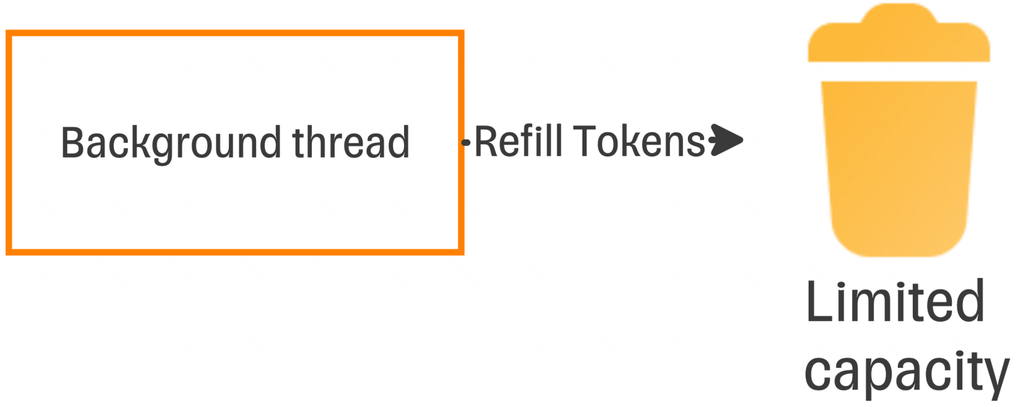

```kotlin
fun getBucket(
    maxCapacity: Long,
    initialCapacity: Long = 0,
    refillTokens: Long,
    refillPeriod: Duration,
): Bucket =
    Bucket.builder().addLimit { limit ->
        limit
            // макимальный размер корзины
            .capacity(maxCapacity)
            // стратегия и параметры пополнения
            // (как, на сколько, как часто)
            .refillGreedy(refillTokens, refillPeriod)
            // начальный размер корзины
            .initialTokens(initialCapacity)
    }.build()

fun getBucket(maxPollRecords: Long): Bucket =
    getBucket(
        maxCapacity = 2 * maxPollRecords,
        refillTokens = 20,
        refillPeriod = Duration.ofSeconds( seconds: 1)
    )
```

Делаем корзинку




```
22  fun getBucket(
23      maxCapacity: Long,
24      initialCapacity: Long = 0,
25      refillTokens: Long,
26      refillPeriod: Duration,
27  ): Bucket =
28      Bucket.builder().addLimit { limit ->
29          limit
30              // макимальный размер корзины
31              .capacity(maxCapacity)
32              // стратегия и параметры пополнения
33              // (как, на сколько, как часто)
34              .refillGreedy(refillTokens, refillPeriod)
35              // начальный размер корзины
36              .initialTokens(initialCapacity)
37      }.build()
38
39  fun getBucket(maxPollRecords: Long): Bucket =
40      getBucket(
41          maxCapacity = 2 * maxPollRecords,
42          refillTokens = 20,
43          refillPeriod = Duration.ofSeconds( seconds: 1)
44      )
```

Делаем корзинку




```
22  fun getBucket(
23      maxCapacity: Long,
24      initialCapacity: Long = 0,
25      refillTokens: Long,
26      refillPeriod: Duration,
27  ): Bucket =
28      Bucket.builder().addLimit { limit ->
29          limit
30              // макимальный размер корзины
31              .capacity(maxCapacity)
32              // стратегия и параметры пополнения
33              // (как, на сколько, как часто)
34              .refillGreedy(refillTokens, refillPeriod)
35              // начальный размер корзины
36              .initialTokens(initialCapacity)
37      }.build()
38
39  fun getBucket(maxPollRecords: Long): Bucket =
40      getBucket(
41          maxCapacity = 2 * maxPollRecords,
42          refillTokens = 20,
43          refillPeriod = Duration.ofSeconds( seconds: 1)
44      )
```

Реализуем rate-limiting

```kotlin
fun pollWithLimit(
    bucket: Bucket,
    consumer: KafkaConsumer<String, Request>,
    pollTimeout: Duration,
    maxPollRecords: Long
): ConsumerRecords<String, Request> {
    // Ждем свободных токенов на максимальный батч
    bucket.asBlocking().consume(maxPollRecords)
    // Полим батч
    val records = consumer.poll(pollTimeout)
    // Возвращаем неиспользованые токены обратно в бакет
    val diff = maxPollRecords - records.count()
    if (diff > 0) {
        bucket.addTokens(diff)
    }
    return records
}
```

→

```kotlin
) {
    val bucket = getBucket(maxPollRecords)
    consumer.subscribe(listOf(requestTopic))
    while (true) {
        val requests = pollWithLimit(
            bucket,
            consumer,
            pollTimeout,
            maxPollRecords
        )
        val responses = processRequests(
            requests,
            batchDispatcher
        )
        produceResponses(
            responses,
            responseTopic,
            producer
        )
        commitOffsets(consumer, requests)
        logMetrics(requests)
    }
}
```

Реализуем rate-limiting

```
46  fun pollWithLimit(
47      bucket: Bucket,
48      consumer: KafkaConsumer<String, Request>,
49      pollTimeout: Duration,
50      maxPollRecords: Long
51  ): ConsumerRecords<String, Request> {
52      // Ждем свободных токенов на максимальный батч
53      bucket.asBlocking().consume(maxPollRecords)
54      // Полим батч
55      val records = consumer.poll(pollTimeout)
56      // Возвращаем неиспользованые токены обратно в бакет
57      val diff = maxPollRecords - records.count()
58      if (diff > 0) {
59          bucket.addTokens(diff)
60      }
61      return records
62  }
```

→

```
116 ) {
117     val bucket = getBucket(maxPollRecords)
118     consumer.subscribe(listOf(requestTopic))
119     while (true) {
120         val requests = pollWithLimit(
121             bucket,
122             consumer,
123             pollTimeout,
124             maxPollRecords
125         )
126         val responses = processRequests(
127             requests,
128             batchDispatcher
129         )
130         produceResponses(
131             responses,
132             responseTopic,
133             producer
134         )
135         commitOffsets(consumer, requests)
136         logMetrics(requests)
137     }
138 }
```

Реализуем rate-limiting

```
46  fun pollWithLimit(
47      bucket: Bucket,
48      consumer: KafkaConsumer<String, Request>,
49      pollTimeout: Duration,
50      maxPollRecords: Long
51  ): ConsumerRecords<String, Request> {
52      // Ждем свободных токенов на максимальный батч
53      bucket.asBlocking().consume(maxPollRecords)
54      // Полим батч
55      val records = consumer.poll(pollTimeout)
56      // Возвращаем неиспользованые токены обратно в бакет
57      val diff = maxPollRecords - records.count()
58      if (diff > 0) {
59          bucket.addTokens(diff)
60      }
61      return records
62  }
```

→

```
116 ) {
117     val bucket = getBucket(maxPollRecords)
118     consumer.subscribe(listOf(requestTopic))
119     while (true) {
120         val requests = pollWithLimit(
121             bucket,
122             consumer,
123             pollTimeout,
124             maxPollRecords
125         )
126         val responses = processRequests(
127             requests,
128             batchDispatcher
129         )
130         produceResponses(
131             responses,
132             responseTopic,
133             producer
134         )
135         commitOffsets(consumer, requests)
136         logMetrics(requests)
137     }
138 }
```

Реализуем rate-limiting

```kotlin
fun pollWithLimit(
    bucket: Bucket,
    consumer: KafkaConsumer<String, Request>,
    pollTimeout: Duration,
    maxPollRecords: Long
): ConsumerRecords<String, Request> {
    // Ждем свободных токенов на максимальный батч
    bucket.asBlocking().consume(maxPollRecords)
    // Полим батч
    val records = consumer.poll(pollTimeout)
    // Возвращаем неиспользованые токены обратно в бакет
    val diff = maxPollRecords - records.count()
    if (diff > 0) {
        bucket.addTokens(diff)
    }
    return records
}
```

→

```kotlin
) {
    val bucket = getBucket(maxPollRecords)
    consumer.subscribe(listOf(requestTopic))
    while (true) {
        val requests = pollWithLimit(
            bucket,
            consumer,
            pollTimeout,
            maxPollRecords
        )
        val responses = processRequests(
            requests,
            batchDispatcher
        )
        produceResponses(
            responses,
            responseTopic,
            producer
        )
        commitOffsets(consumer, requests)
        logMetrics(requests)
    }
}
```

Собираем и деплоим

Вроде работает

А что от нас хотели-то

Хотелки

Ответ на каждый запрос

Никогда и ни при каких обстоятельствах не терять сообщения

Хотелки

Ответ на каждый запрос

Никогда и ни при каких обстоятельствах не терять сообщения

Сохранять порядок

Стараться не менять порядок запросов и ответов

Хотелки

Ответ на каждый запрос

Никогда и ни при каких обстоятельствах не терять сообщения

Сохранять порядок

Стараться не менять порядок запросов и ответов

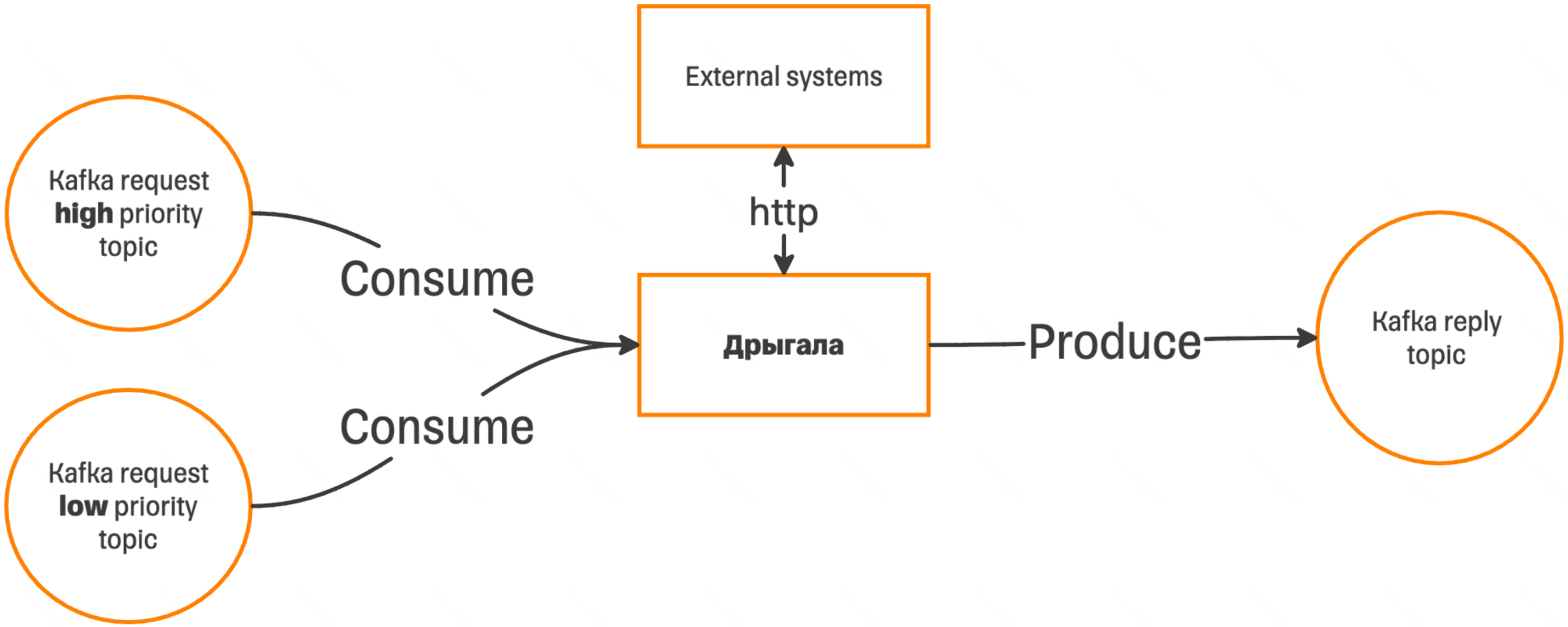
Не дублировать ответы

По возможности избегать дублирования ответа по одному запросу

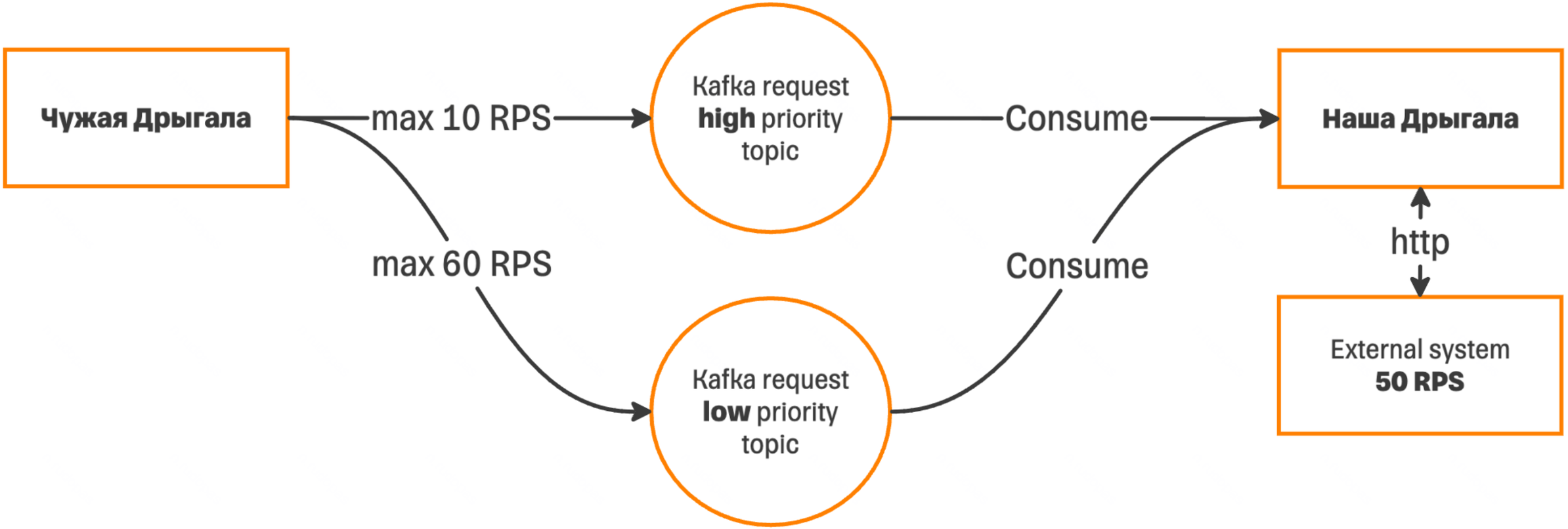
Нарекаем выполненным и расходимся?

Второй поток хочу!

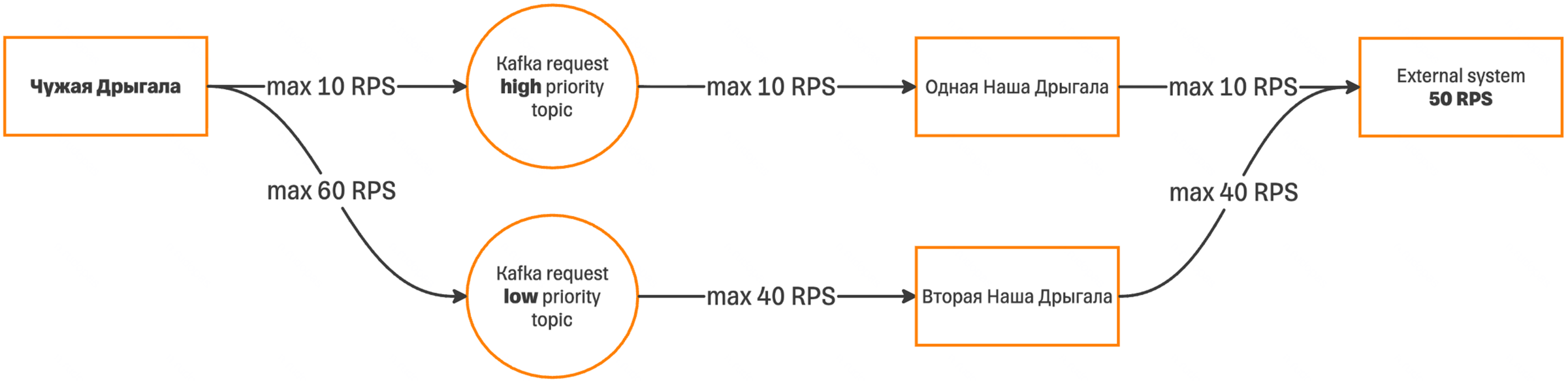
У вас будет два топика на вход, говорили они



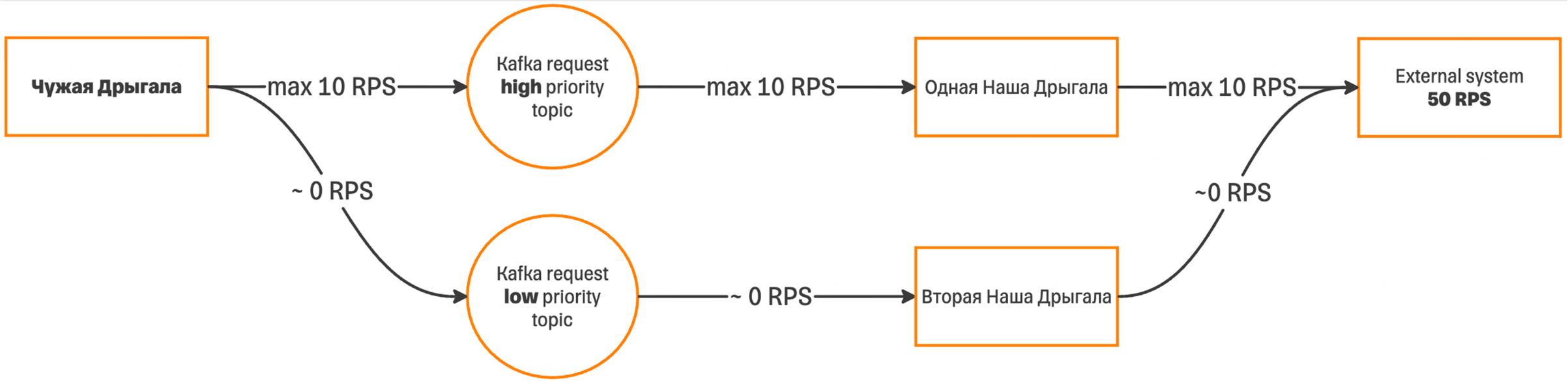
Можно, а зачем



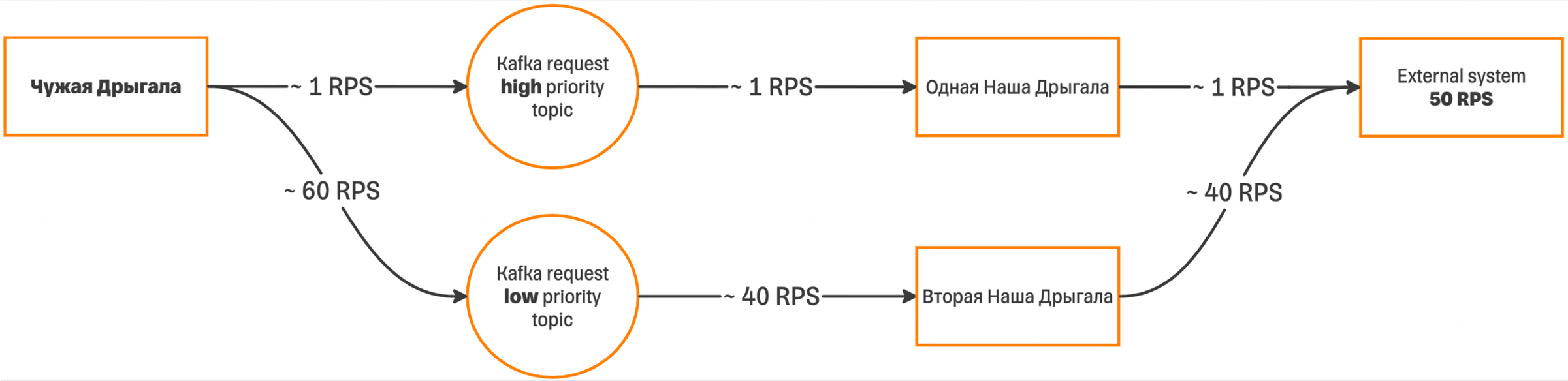
Давайте в две дрыгалы



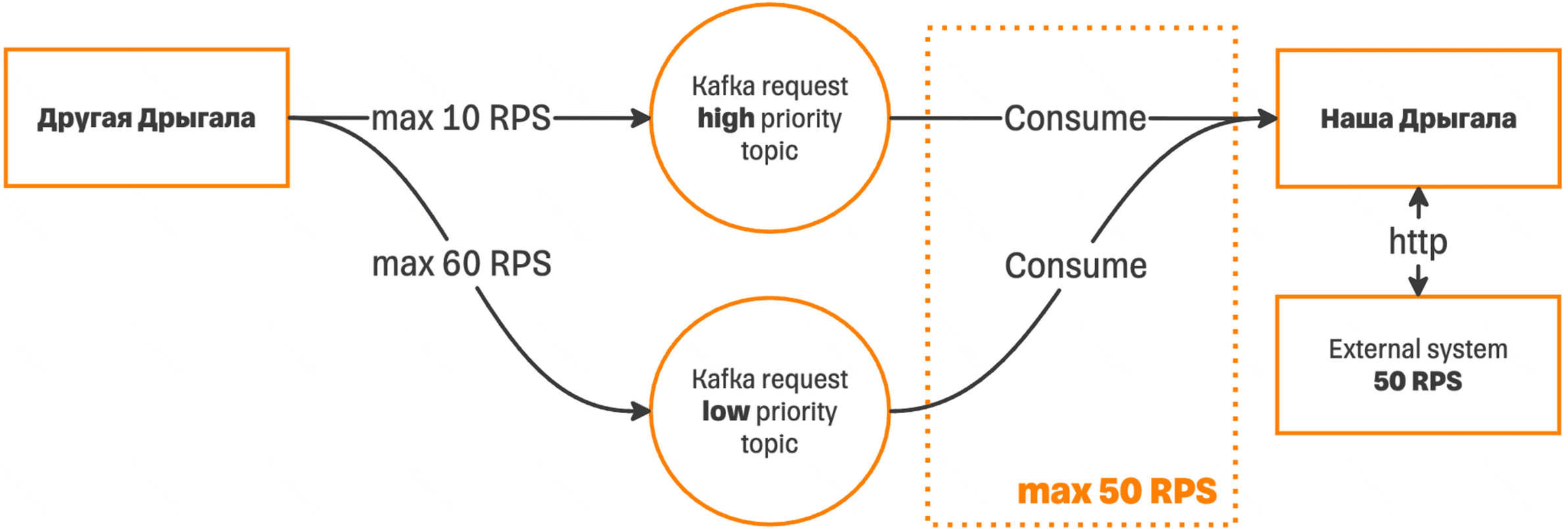
Предполагаемая нагрузка днем



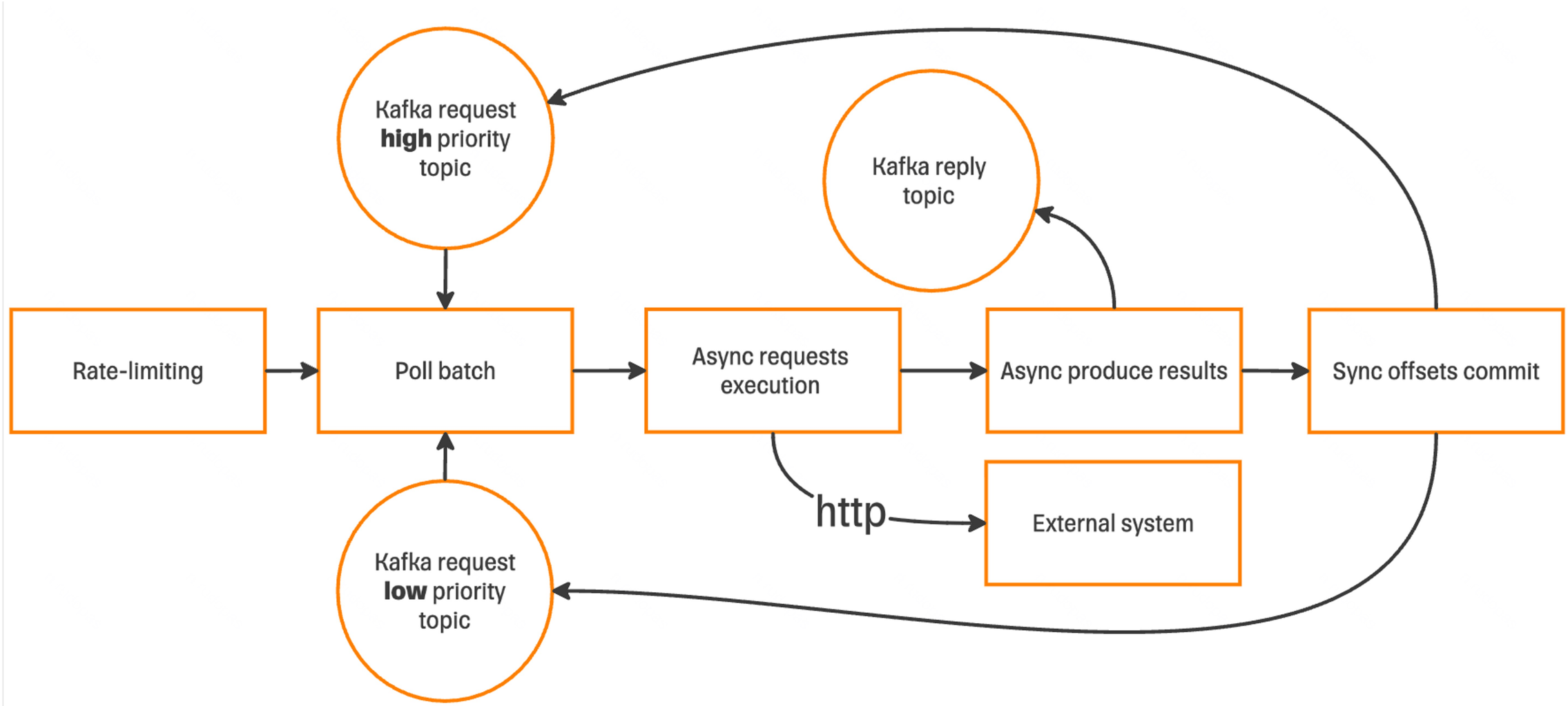
Предполагаемая нагрузка ночью



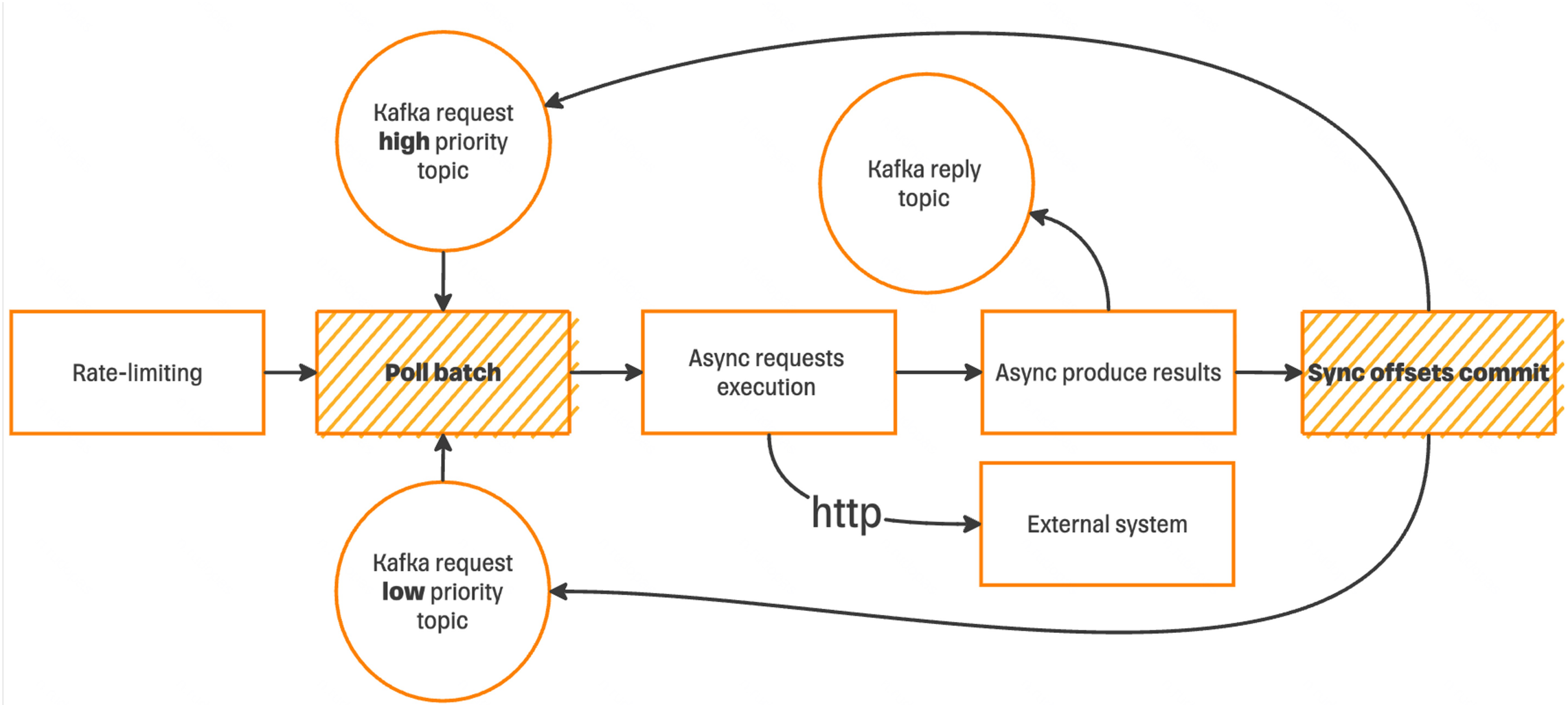
Надо утилизировать канал



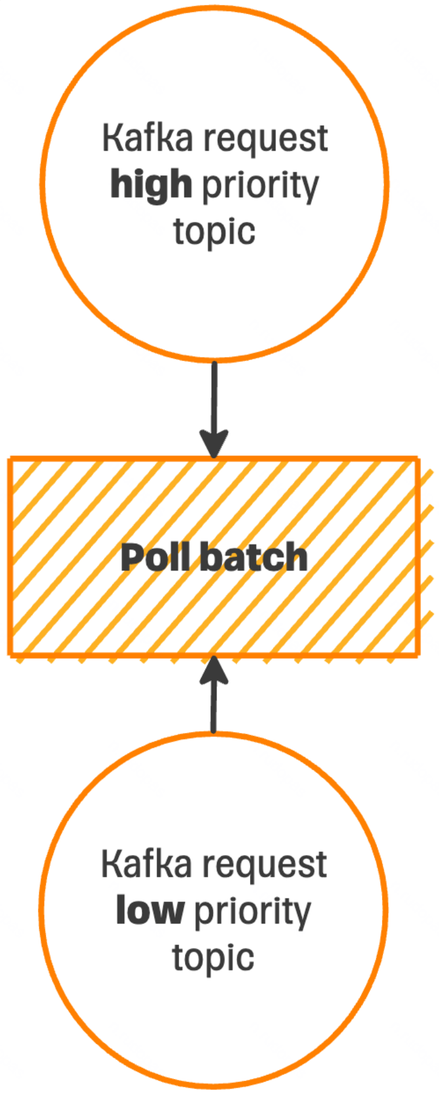
Очень понятная схема



А теперь

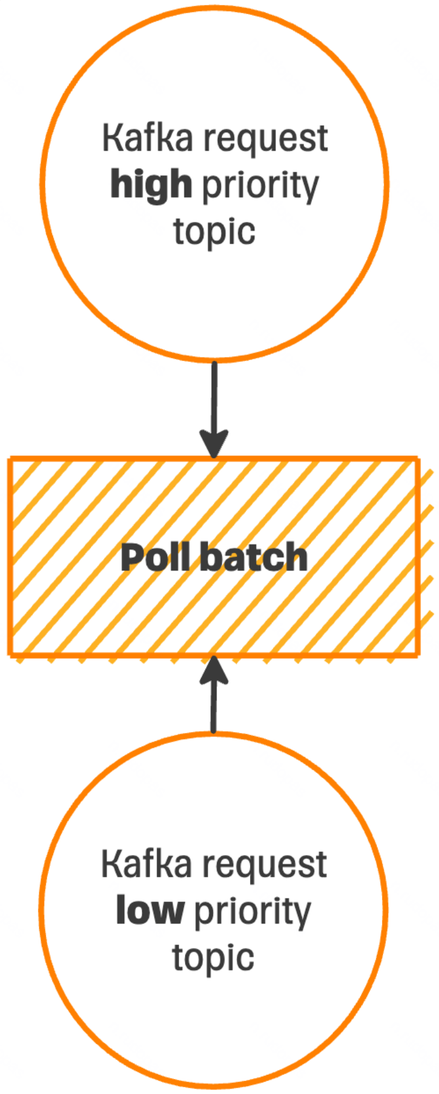

Kafka request
high priority
topic
Kafka reply
topic
Rate-limiting
Poll batch
Async requests
execution
Async produce results
Sync offsets commit
http
External system
Kafka request
low priority
topic

Полим дважды



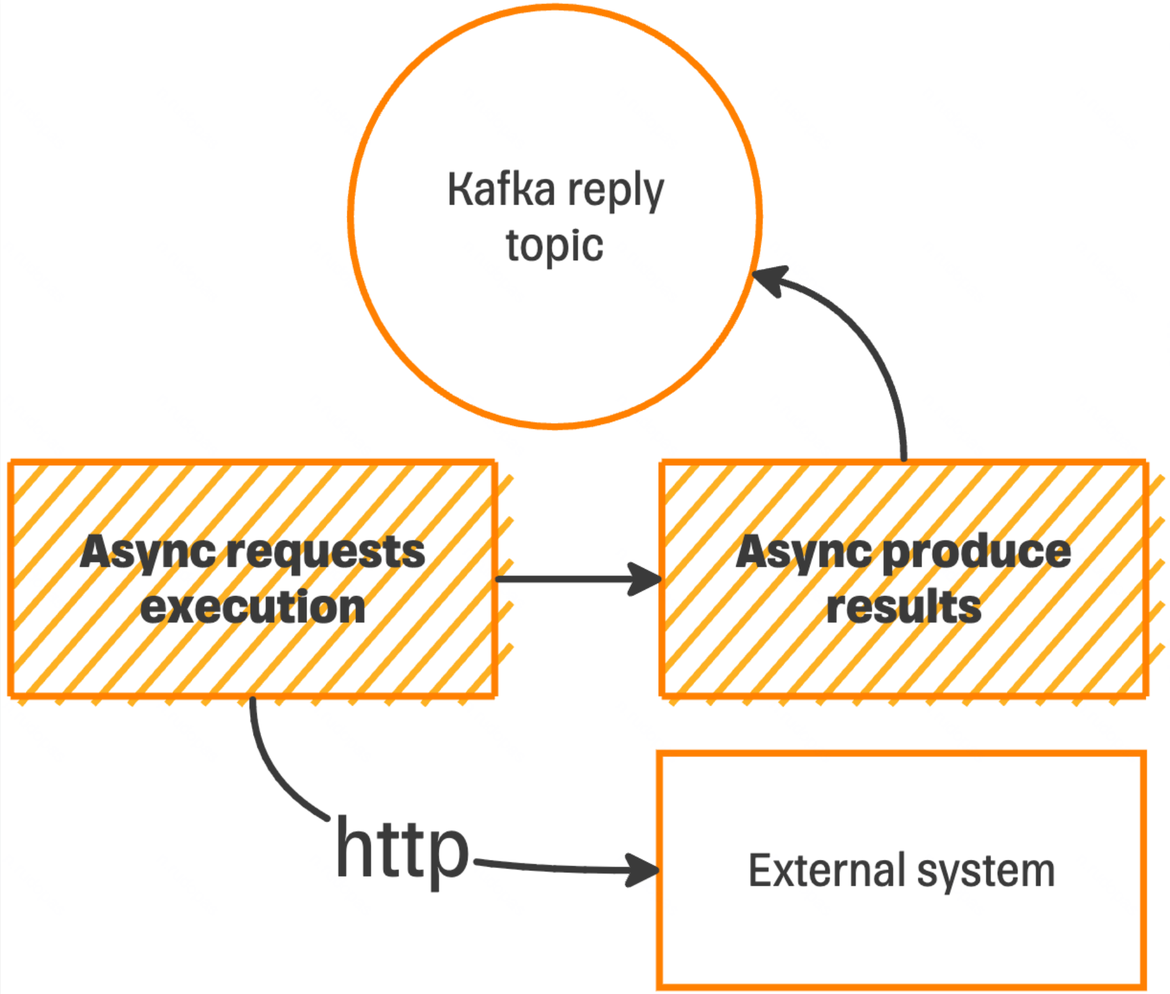

```
119     val bucket = getBucket(maxPollRecords)
120     highPriorityConsumer.subscribe(listOf(highPriorityRequestTopic))
121     lowPriorityConsumer.subscribe(listOf(lowPriorityRequestTopic))
122     while (true) {
123         val highPriorityRequests = pollWithLimit(
124             bucket,
125             highPriorityConsumer,
126             pollTimeout,
127             maxPollRecords
128         )
129         val lowPriorityRequests = pollWithLimit(
130             bucket,
131             lowPriorityConsumer,
132             pollTimeout,
133             maxPollRecords
134         )
```

Полим дважды



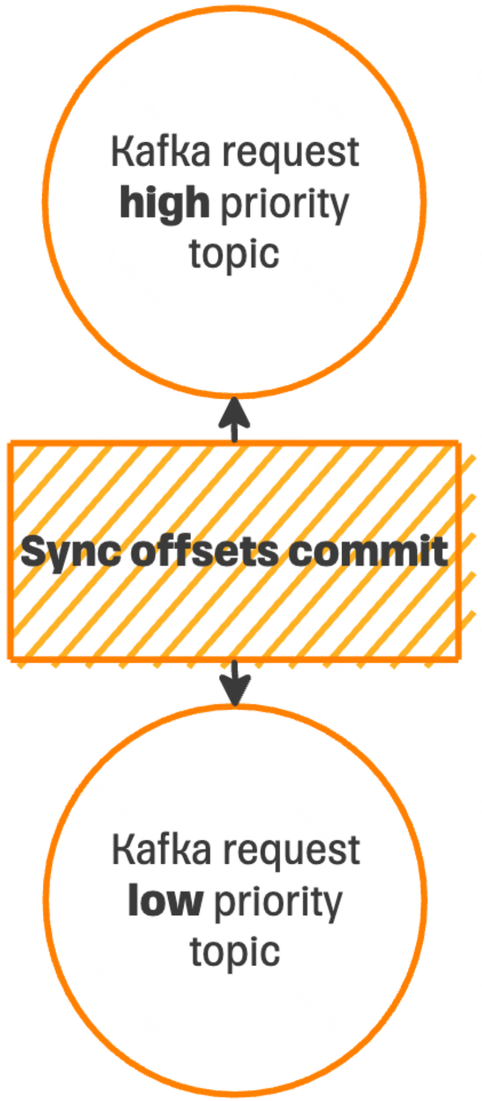

```
119     val bucket = getBucket(maxPollRecords)
120     highPriorityConsumer.subscribe(listOf(highPriorityRequestTopic))
121     lowPriorityConsumer.subscribe(listOf(lowPriorityRequestTopic))
122     while (true) {
123         val highPriorityRequests = pollWithLimit(
124             bucket,
125             highPriorityConsumer,
126             pollTimeout,
127             maxPollRecords
128         )
129         val lowPriorityRequests = pollWithLimit(
130             bucket,
131             lowPriorityConsumer,
132             pollTimeout,
133             maxPollRecords
134         )
```

Обрабатываем и продьюсим единожды



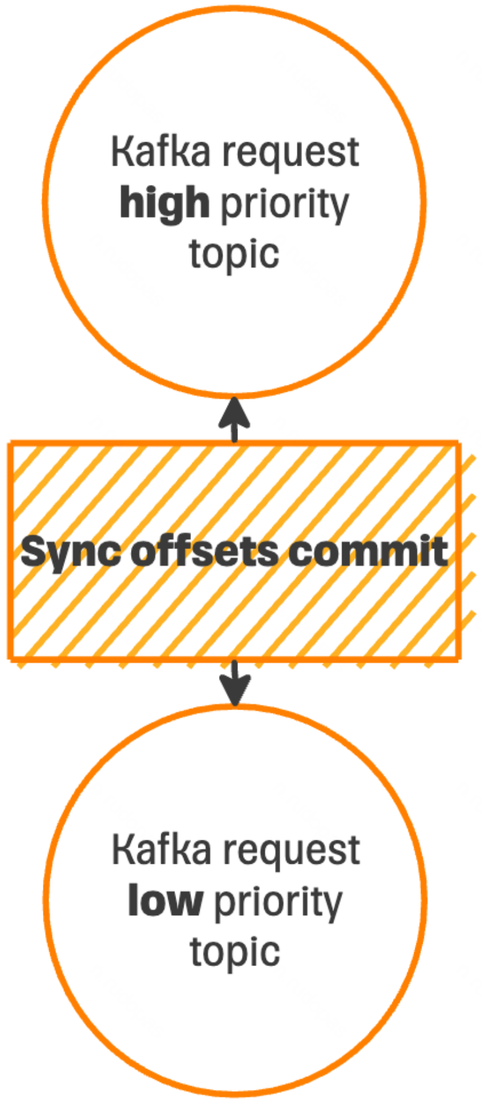

```
136     val responses = processRequests(
137         Iterables.concat( ...inputs:
138             highPriorityRequests +
139                 lowPriorityRequests
140         ),
141         batchDispatcher
142     )
143
144     produceResponses(
145         responses,
146         responseTopic,
147         producer
148     )
```

Сдвигаем оффсеты на дважды




```
136            val responses = processRequests(
137                Iterables.concat( ...inputs:
138                    highPriorityRequests +
139                        lowPriorityRequests
140                ),
141                batchDispatcher
142            )
143
144            produceResponses(
145                responses,
146                responseTopic,
147                producer
148            )
149
150            commitOffsets(highPriorityConsumer, highPriorityRequests)
151            logMetrics(highPriorityRequests)
152
153            commitOffsets(lowPriorityConsumer, lowPriorityRequests)
154            logMetrics(lowPriorityRequests)
155        }
156    }
```

Сдвигаем оффсеты на дважды




```
136            val responses = processRequests(
137                Iterables.concat( ...inputs:
138                    highPriorityRequests +
139                        lowPriorityRequests
140                ),
141                batchDispatcher
142            )
143
144            produceResponses(
145                responses,
146                responseTopic,
147                producer
148            )
149
150            commitOffsets(highPriorityConsumer, highPriorityRequests)
151            logMetrics(highPriorityRequests)
152
153            commitOffsets(lowPriorityConsumer, lowPriorityRequests)
154            logMetrics(lowPriorityRequests)
155        }
156    }
```

Все вместе

Как мы хороши

```
    val bucket = getBucket(maxPollRecords)
    highPriorityConsumer.subscribe(listOf(highPriorityRequestTopic))
    lowPriorityConsumer.subscribe(listOf(lowPriorityRequestTopic))
    while (true) {
        val highPriorityRequests = pollWithLimit(
            bucket,
            highPriorityConsumer,
            pollTimeout,
            maxPollRecords
        )
        val lowPriorityRequests = pollWithLimit(
            bucket,
            lowPriorityConsumer,
            pollTimeout,
            maxPollRecords
        )

        val responses = processRequests(
            Iterables.concat( ...inputs: highPriorityRequests + lowPriorityRequests),
            batchDispatcher
        )

        produceResponses(
            responses,
            responseTopic,
            producer
        )

        commitOffsets(highPriorityConsumer, highPriorityRequests)
        logMetrics(highPriorityRequests)

        commitOffsets(lowPriorityConsumer, lowPriorityRequests)
        logMetrics(lowPriorityRequests)
    }
}
```

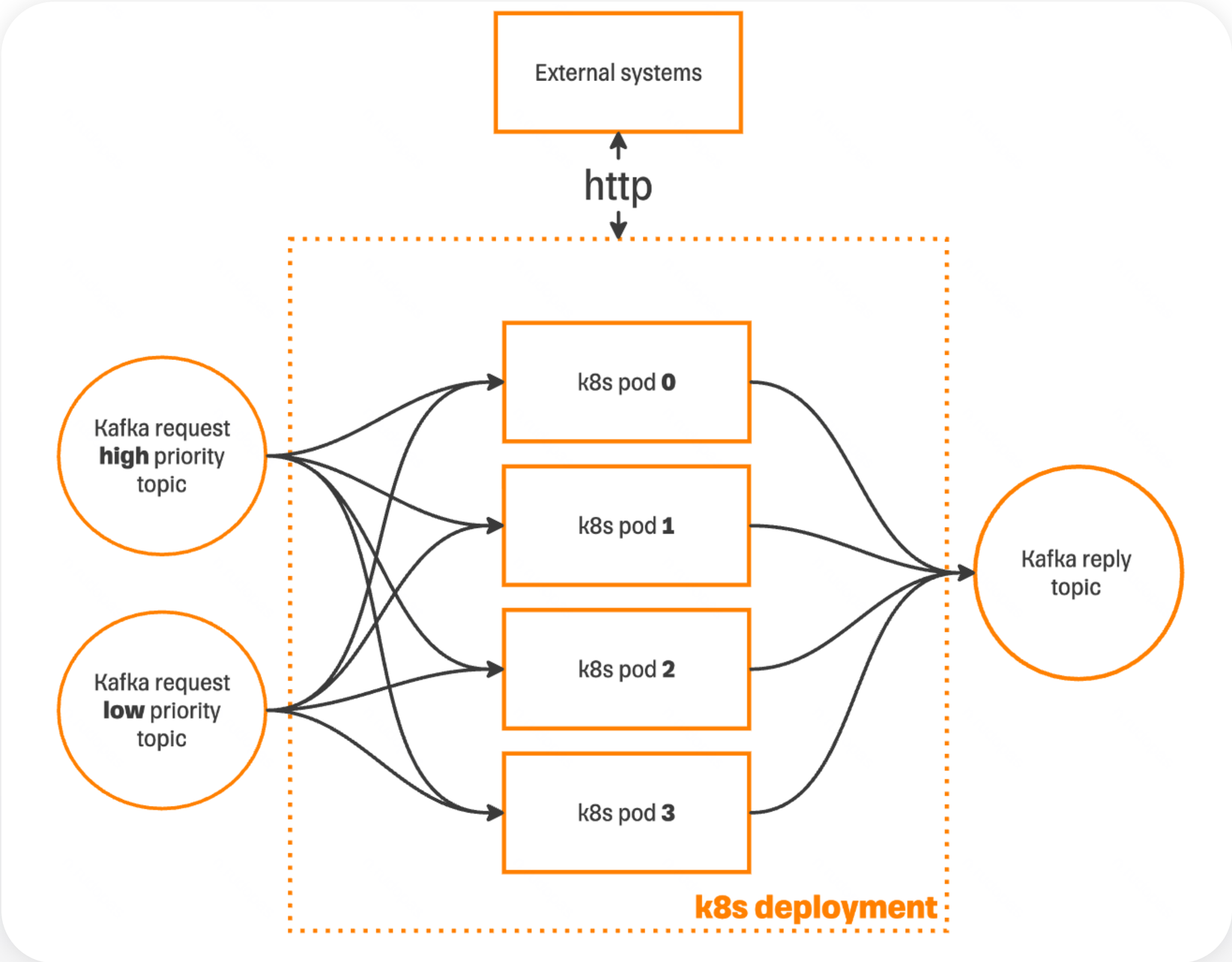
Собираем и деплоим

Все хорошо

Или нет?

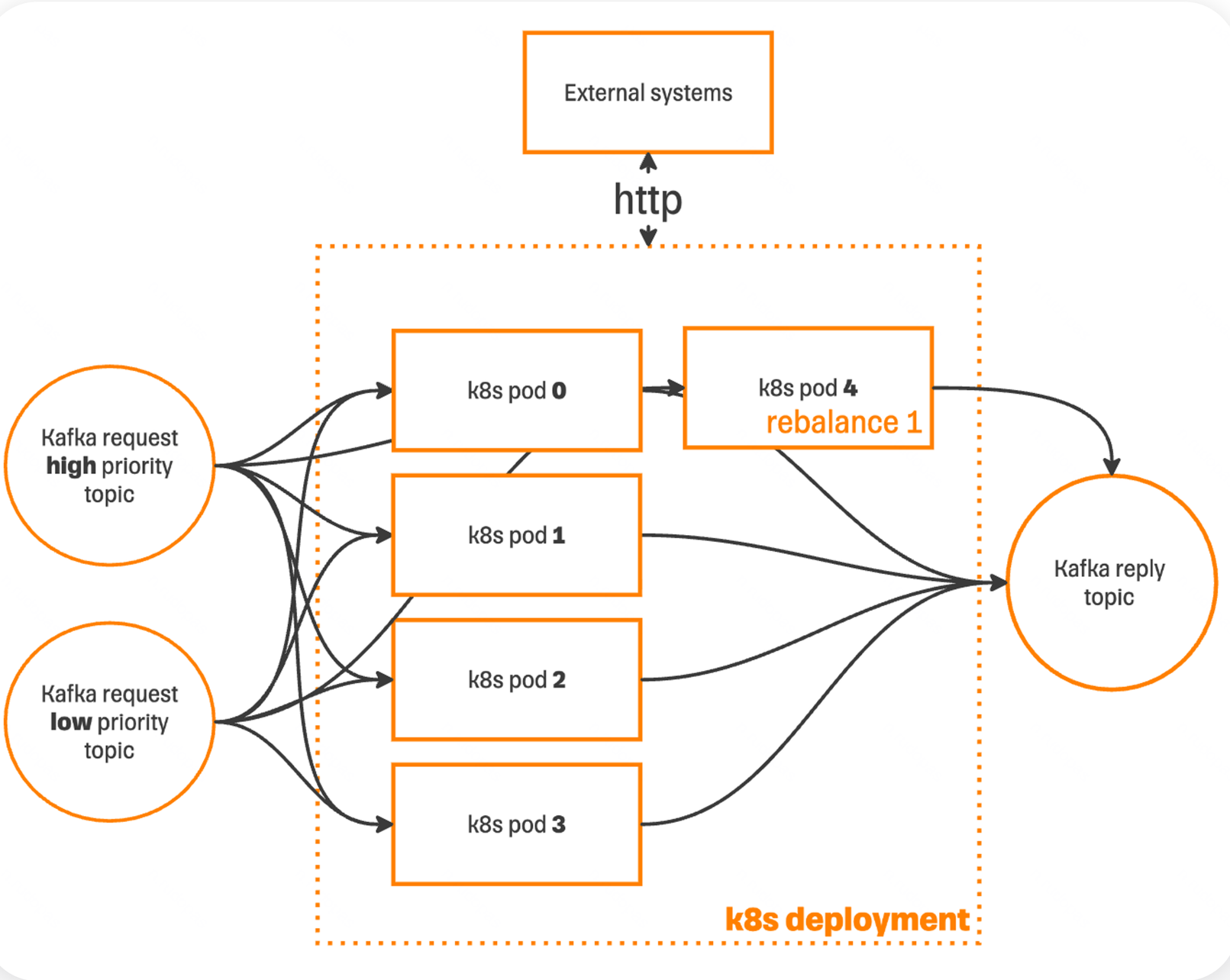
Красивое

- На pod по партиции
- Latency external system avg 300ms
- Consistancy lag avg 500ms
- Consumer group lag <10



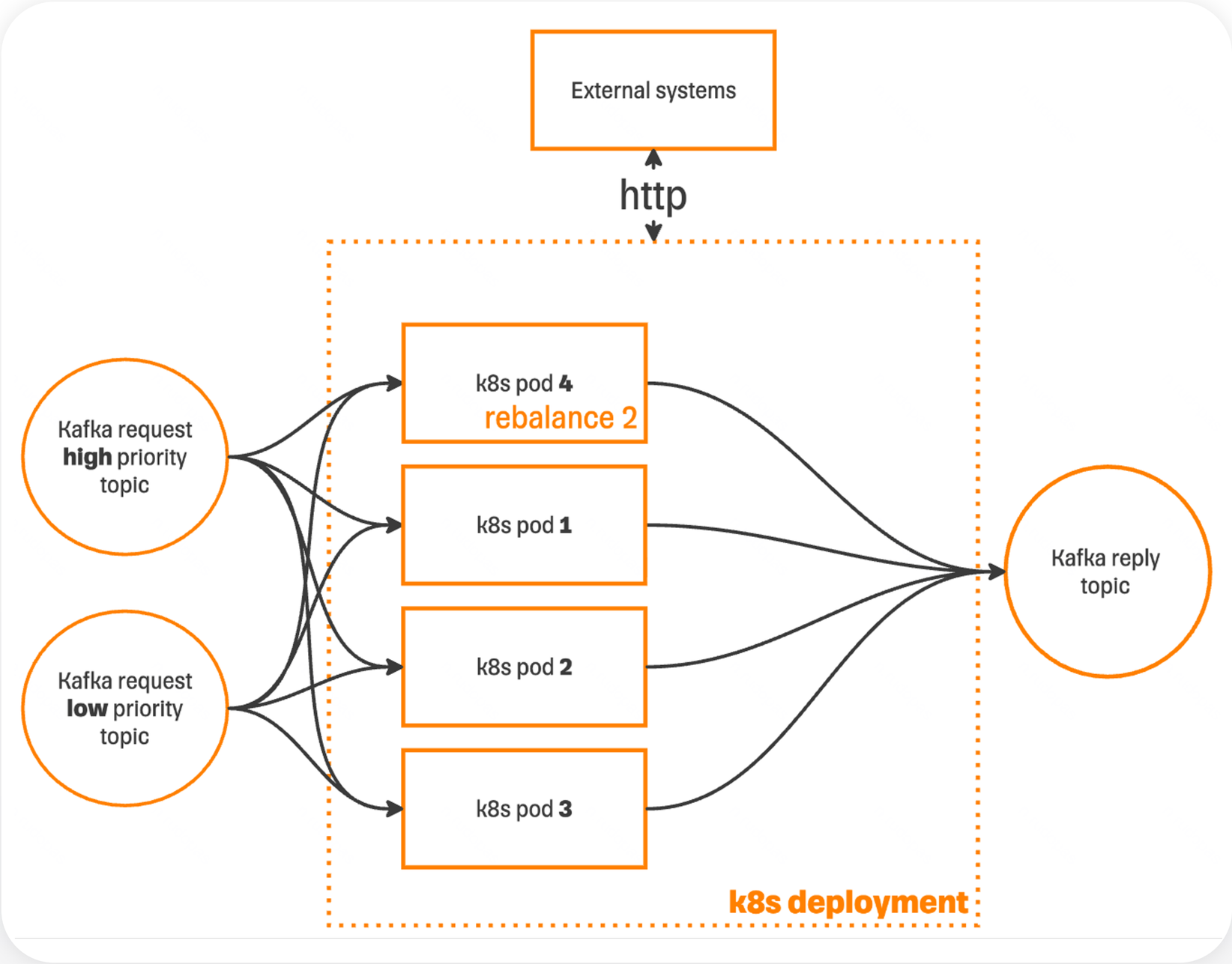
Релизим
Rolling update

+ новый под – ребалансировка



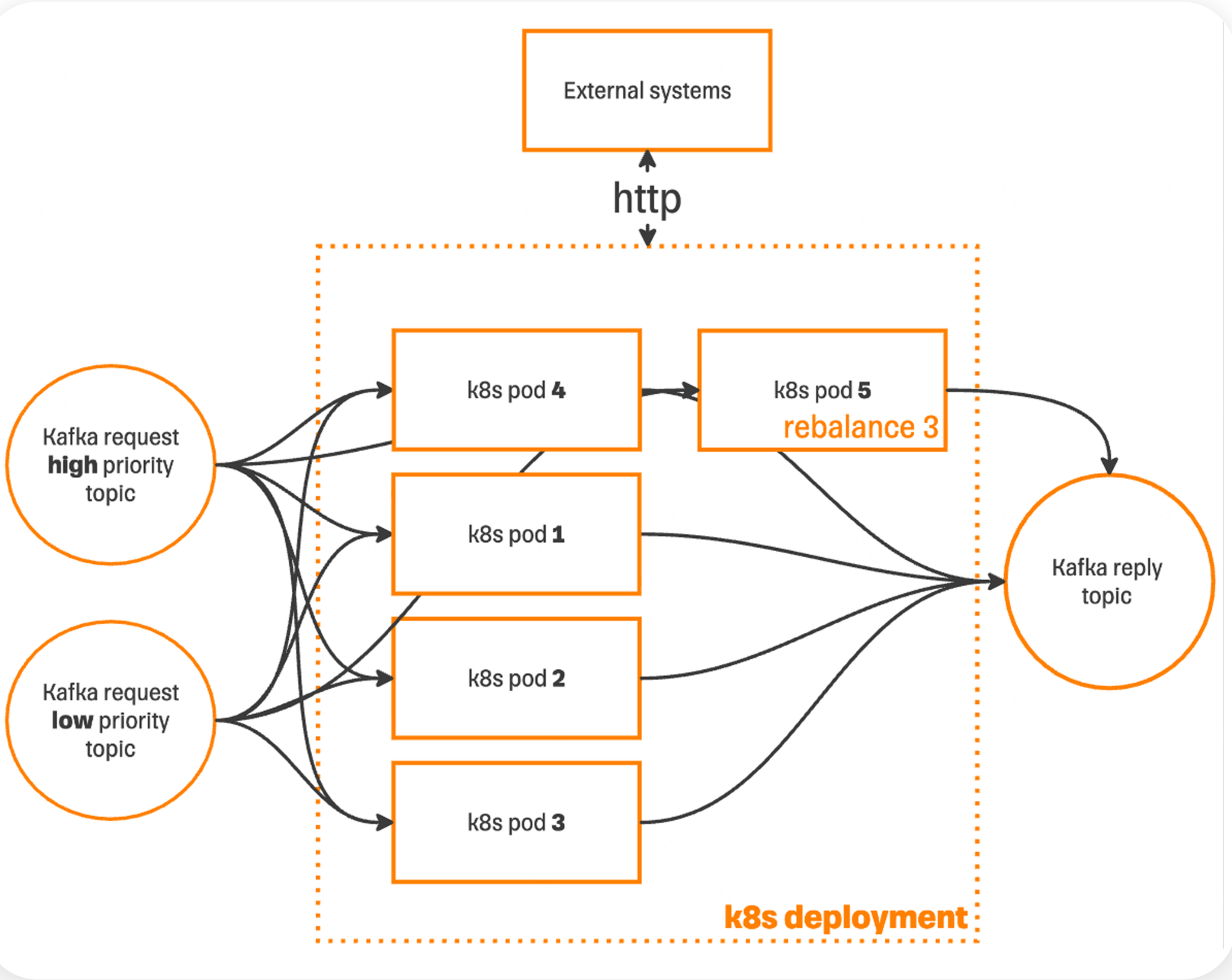
Релизим
Rolling update

- + новый под – ребалансировка
- - старый – ребалансировка



Релизим
Rolling update

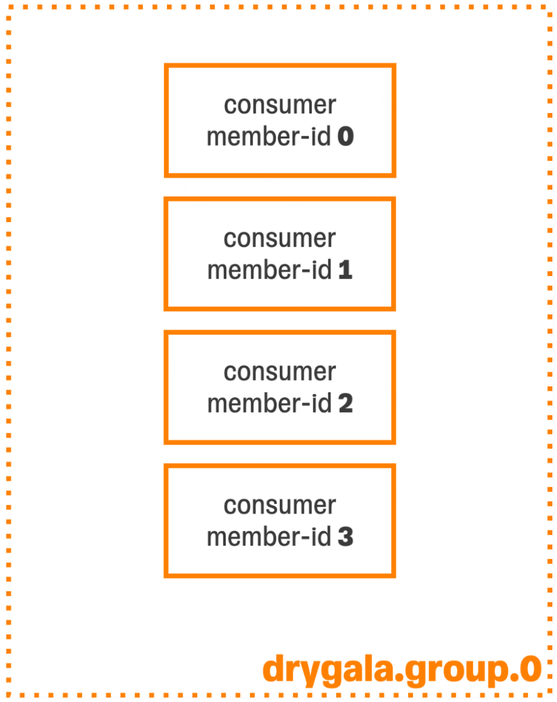
- + новый под – ребалансировка
- - старый – ребалансировка
- etc.



Читаем доки

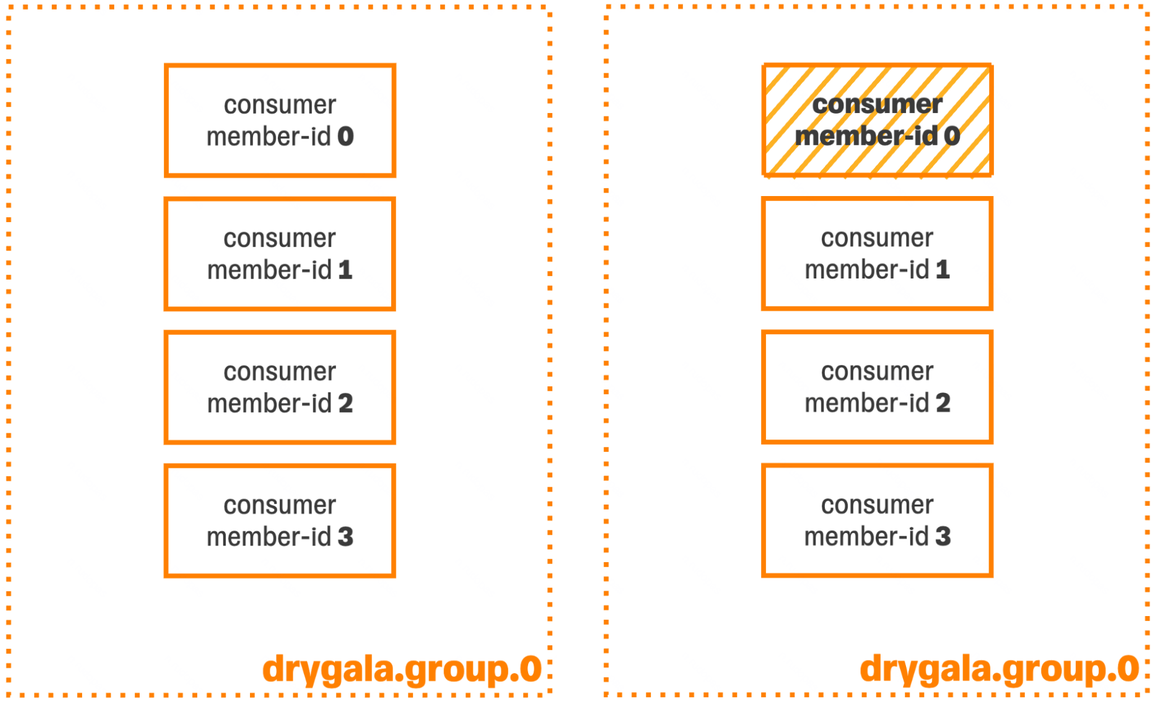
Consumer configuration

```
val consumer = KafkaConsumer<String, Request>(
    mapOf(
        "bootstrap.servers" to "localhost:9093",
        "group.id" to "drygala.group.0"
    )
)
```



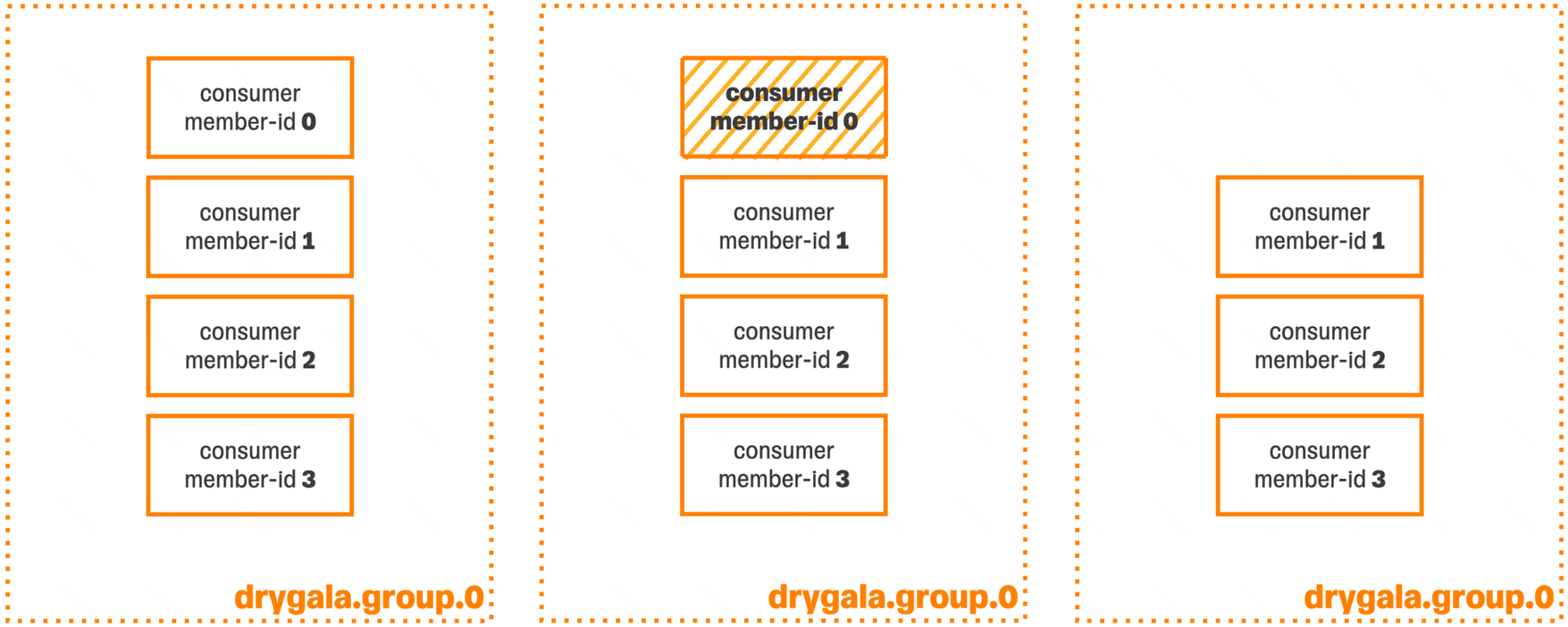
Consumer configuration

```
val consumer = KafkaConsumer<String, Request>(
    mapOf(
        "bootstrap.servers" to "localhost:9093",
        "group.id" to "drygala.group.0"
    )
)
```



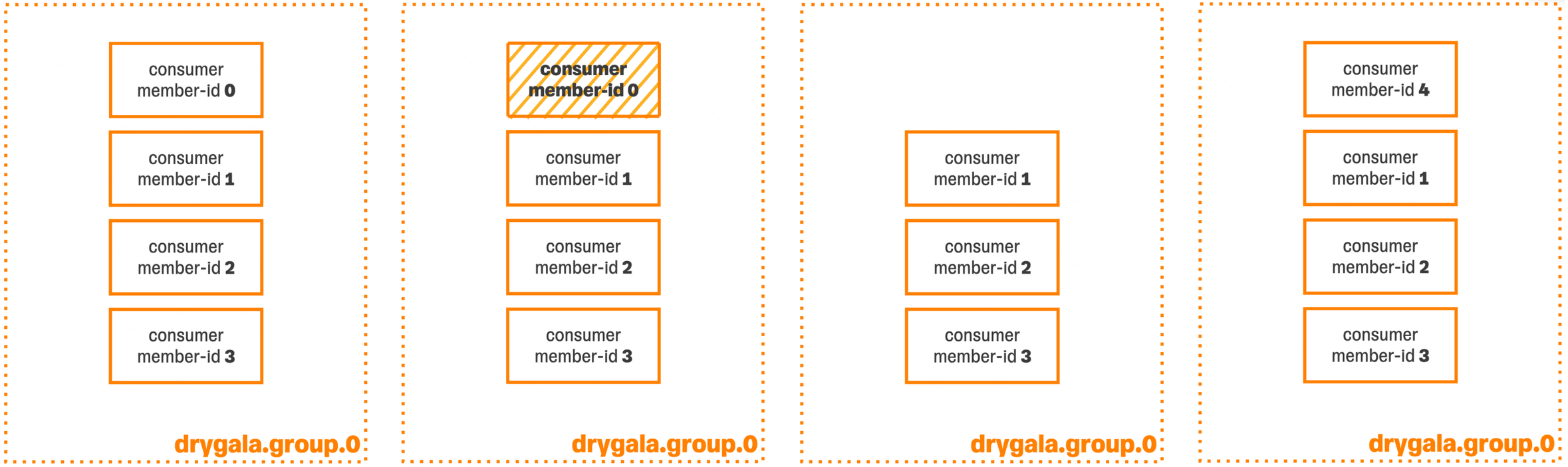
Consumer configuration

```
val consumer = KafkaConsumer<String, Request>(
    mapOf(
        "bootstrap.servers" to "localhost:9093",
        "group.id" to "drygala.group.0"
    )
)
```



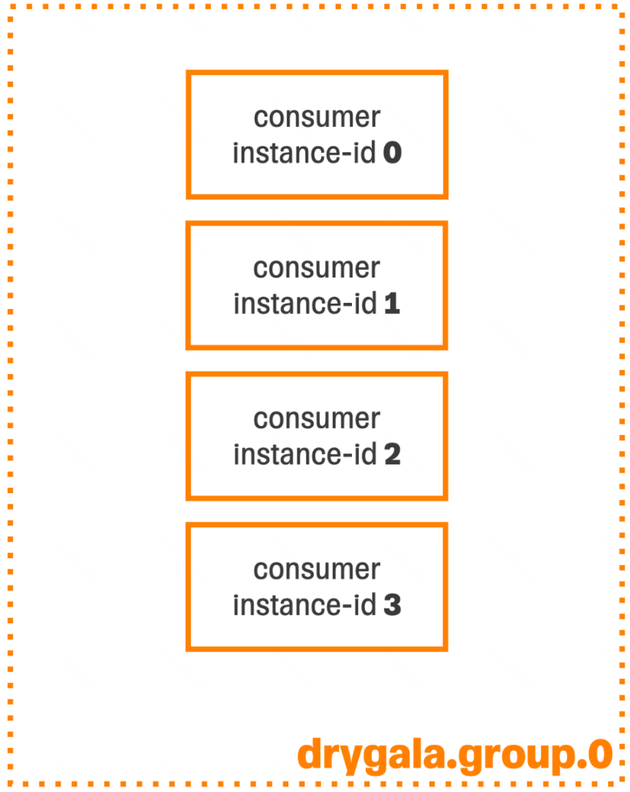
Consumer configuration

```
val consumer = KafkaConsumer<String, Request>(
    mapOf(
        "bootstrap.servers" to "localhost:9093",
        "group.id" to "drygala.group.0"
    )
)
```



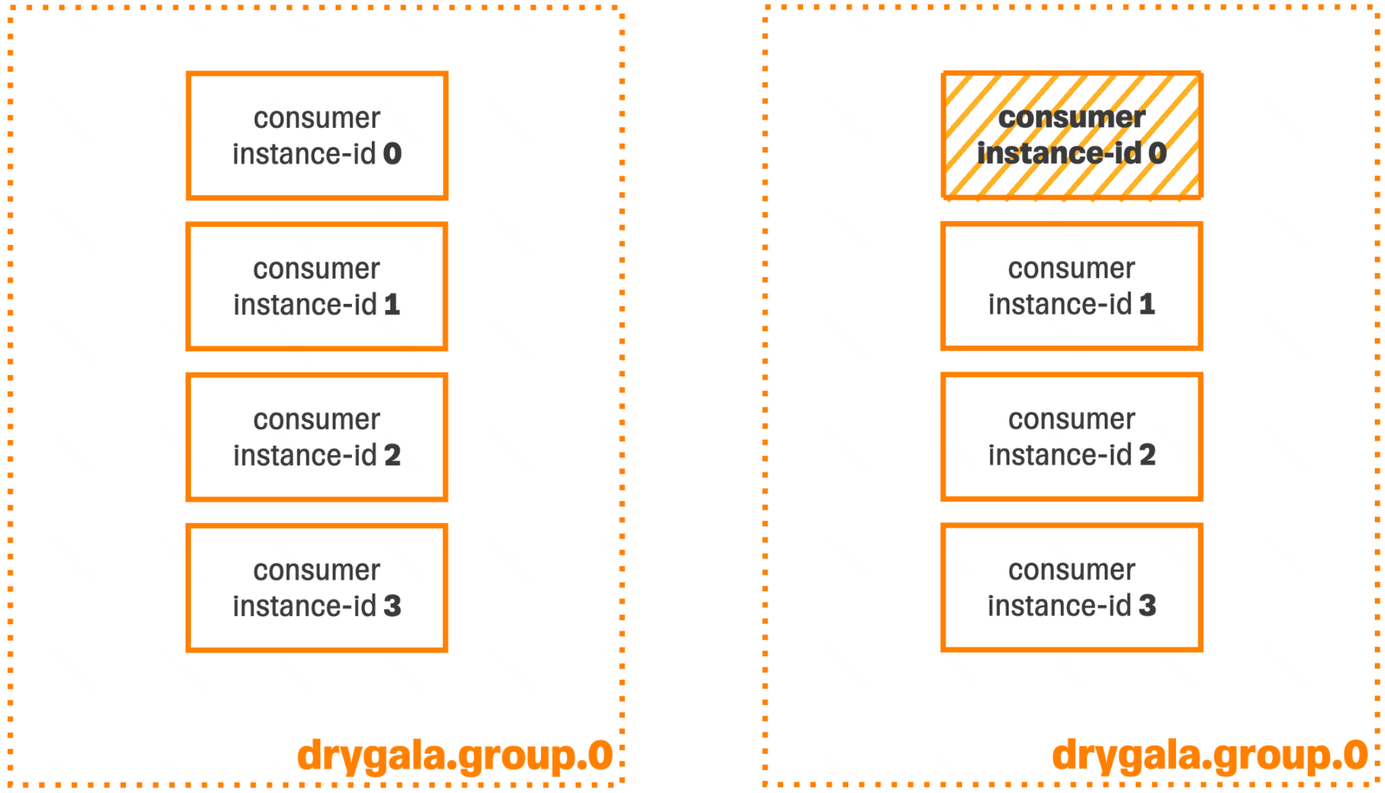
Consumer group static membership

```
val consumer = KafkaConsumer<String, Request>(
    mapOf(
        "bootstrap.servers" to "localhost:9093",
        "group.id" to "drygala.group.0",
        "group.instance.id" to "drygala.instance.0",
        "session.timeout.ms" to "60000",
        "partition.assignment.strategy" to RangeAssignor::class.java.name
    )
)
```



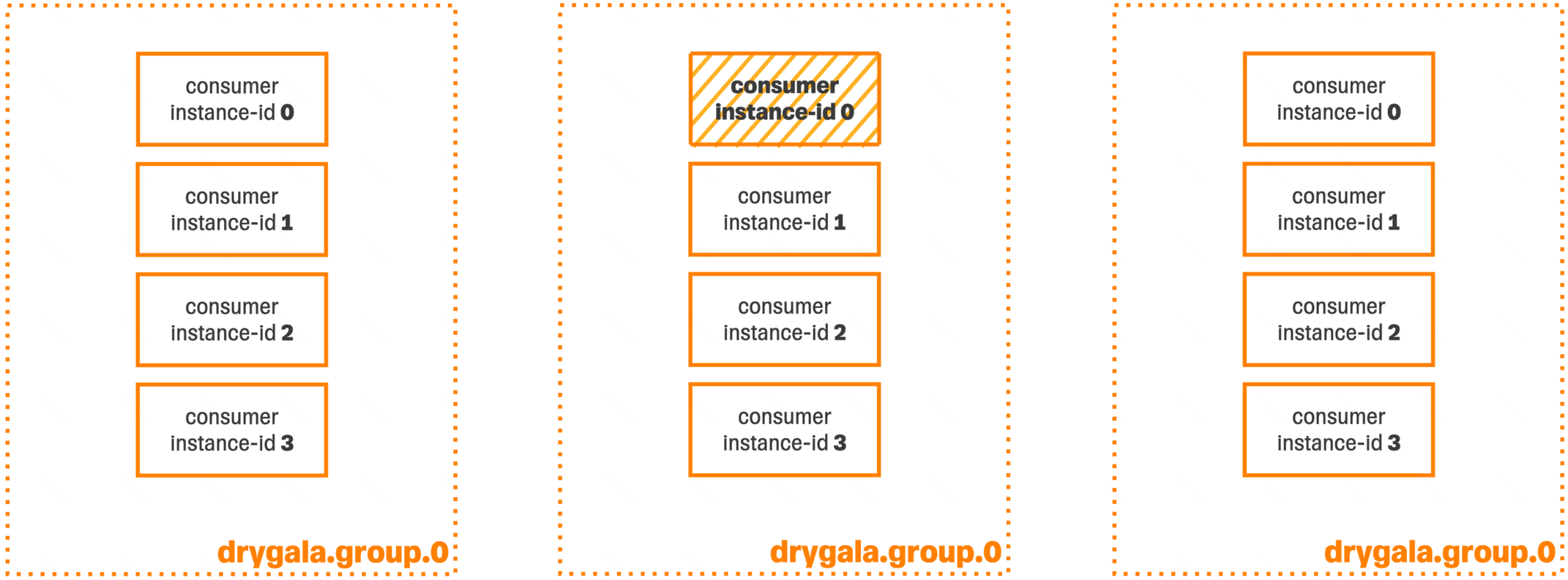
Consumer group static membership

```
val consumer = KafkaConsumer<String, Request>(
    mapOf(
        "bootstrap.servers" to "localhost:9093",
        "group.id" to "drygala.group.0",
        "group.instance.id" to "drygala.instance.0",
        "session.timeout.ms" to "60000",
        "partition.assignment.strategy" to RangeAssignor::class.java.name
    )
)
```



Consumer group static membership

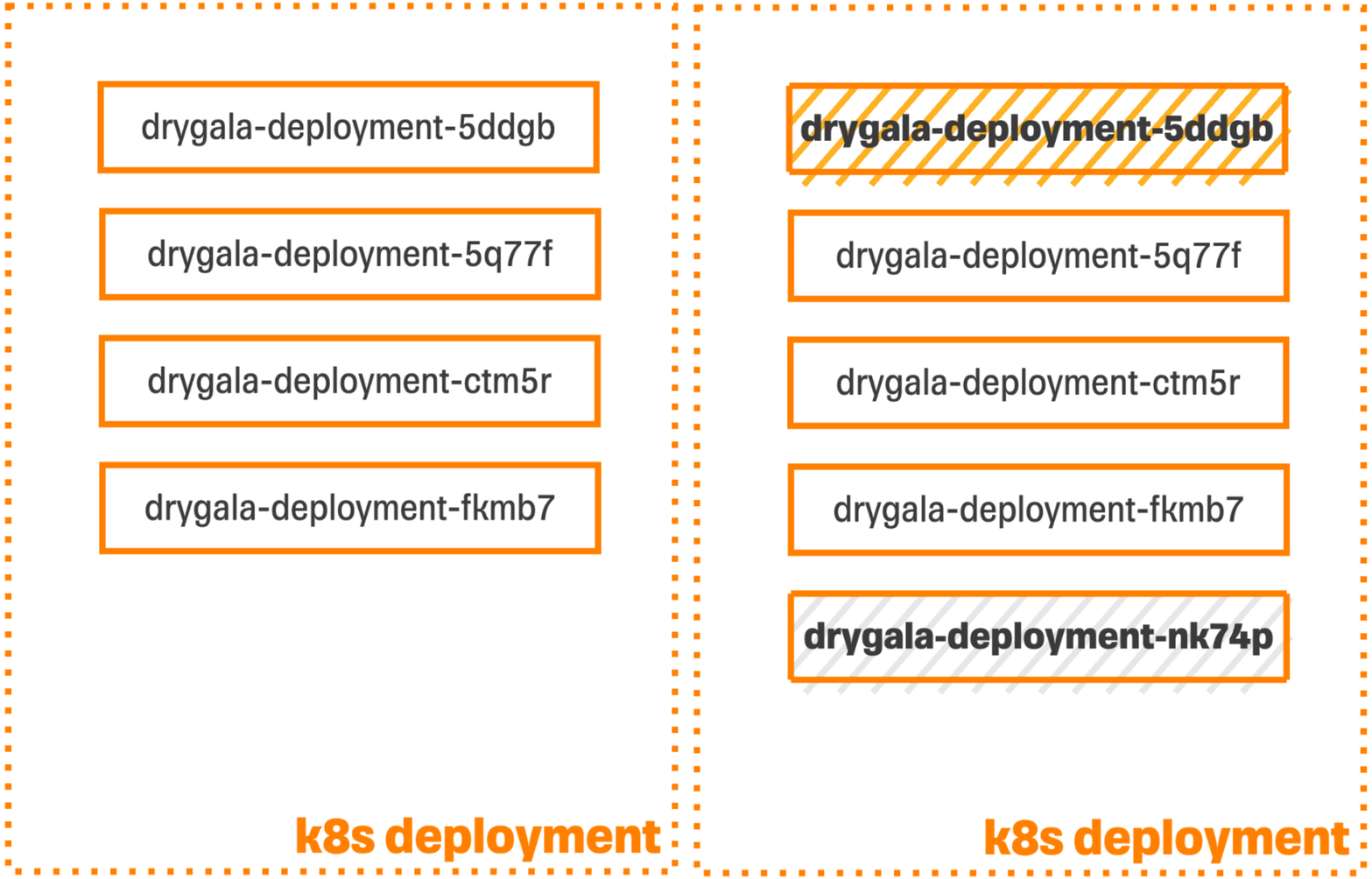
```
val consumer = KafkaConsumer<String, Request>(
    mapOf(
        "bootstrap.servers" to "localhost:9093",
        "group.id" to "drygala.group.0",
        "group.instance.id" to "drygala.instance.0",
        "session.timeout.ms" to "60000",
        "partition.assignment.strategy" to RangeAssignor::class.java.name
    )
)
```



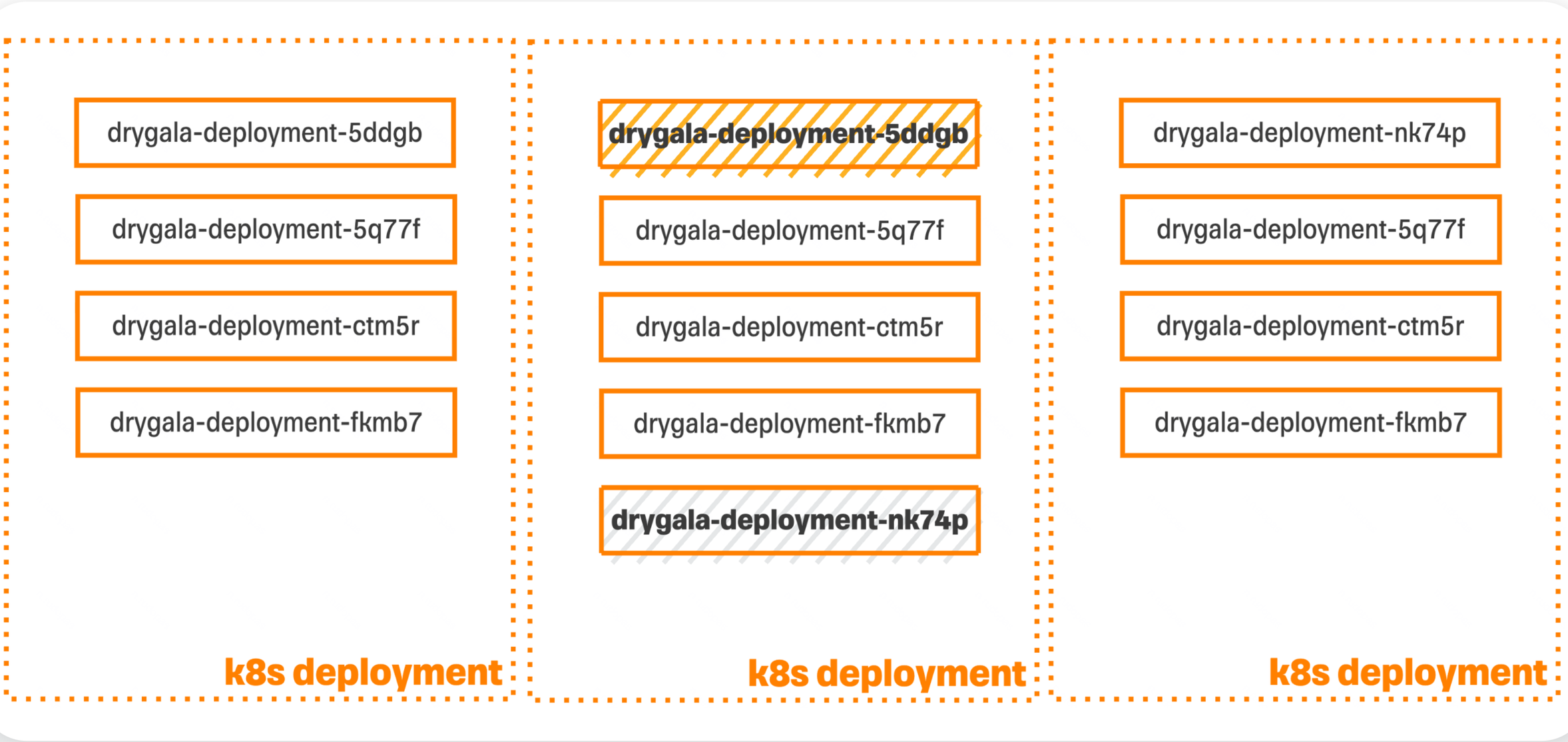
Где взять group.instance.id на практике

k8s Deployment

k8s Deployment

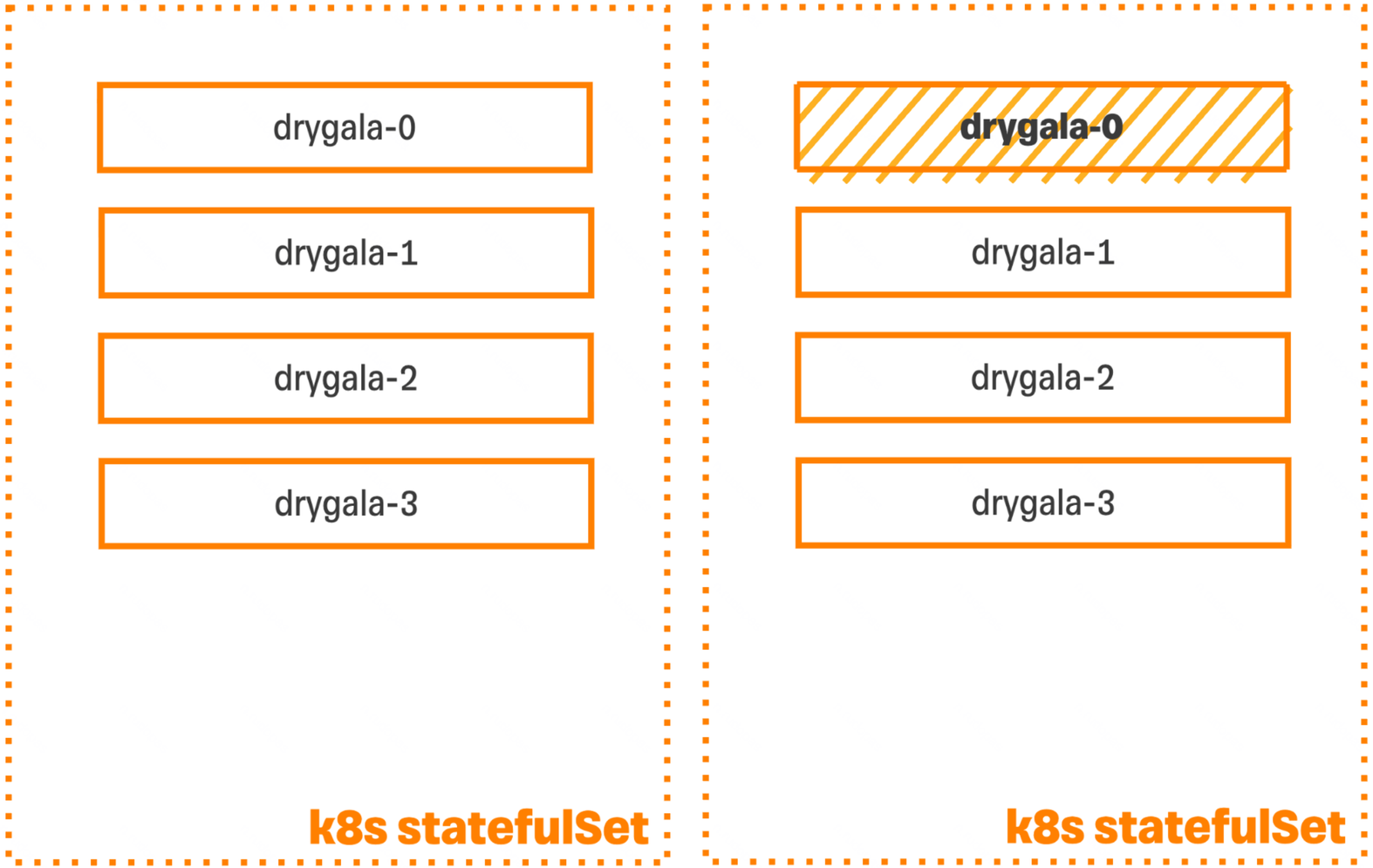


k8s Deployment

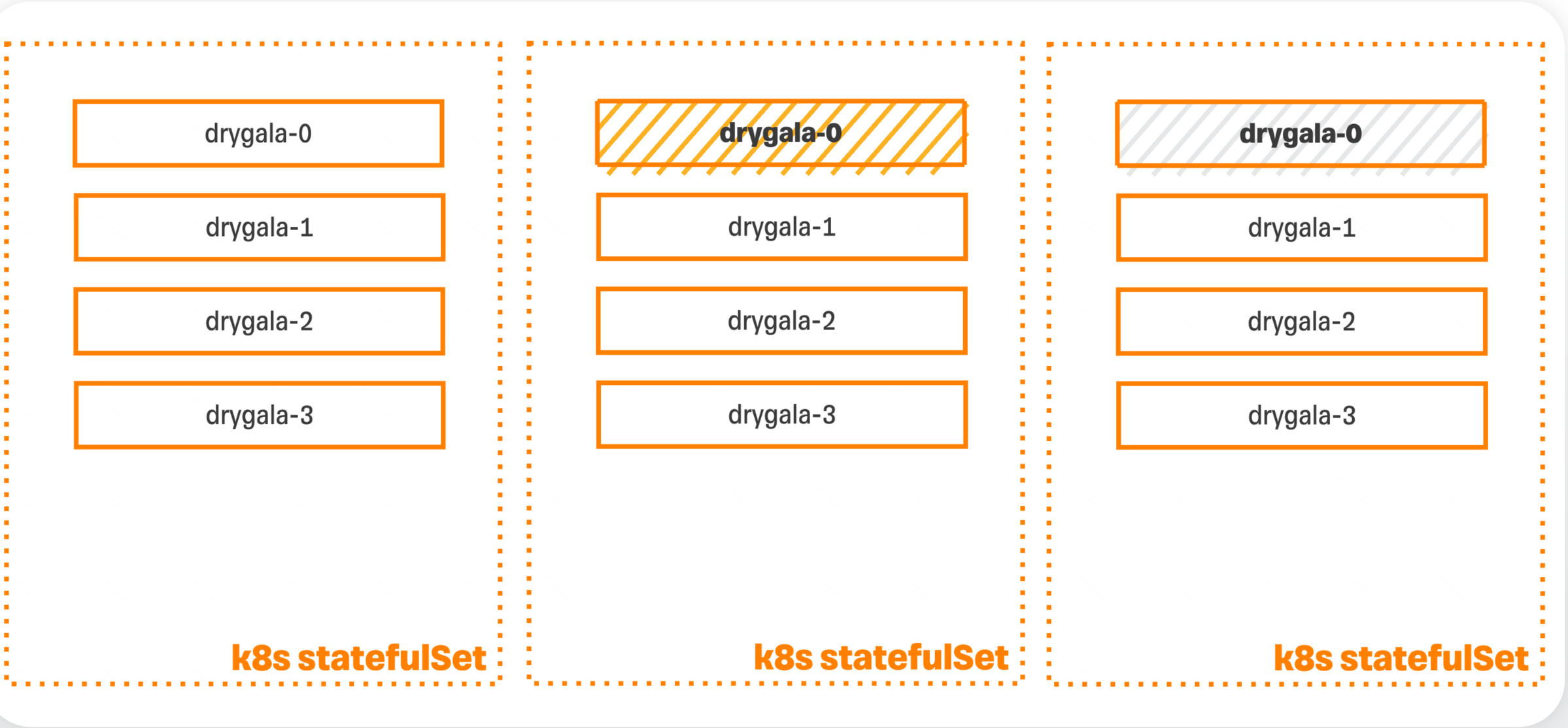


k8s StatefulSets

k8s StatefulSets



k8s StatefulSets



Собираем consumer

```
val consumer = KafkaConsumer<String, Request>(
    mapOf(
        "bootstrap.servers" to "localhost:9093",
        "group.id" to "drygala.group.0"
    )
)
```



```
val consumer = KafkaConsumer<String, Request>(
    mapOf(
        "bootstrap.servers" to "localhost:9093",
        "group.id" to "drygala.group.0",
        "group.instance.id" to hostname,
        "session.timeout.ms" to "60000",
        "partition.assignment.strategy" to RangeAssignor::class.java.name
    )
)
```

Вроде работает

Важное про static membership

01

session.timeout.ms на продолжение обработки без ребалансировки

Другие партиции не ждут

02

session.timeout.ms простоя совсем без ребалансировки при полном отказе

Затронутые партиции ждут

03

session.timeout.ms нужно подбирать под каждое конкретное окружение

Не всегда тривиально

Важное про static membership

01

session.timeout.ms на продолжение обработки без ребалансировки

Другие партиции не ждут

02

session.timeout.ms простоя совсем без ребалансировки при полном отказе

Затронутые партиции ждут

03

session.timeout.ms нужно подбирать под каждое конкретное окружение

Не всегда тривиально

Важное про static membership

01

session.timeout.ms на продолжение обработки без ребалансировки

Другие партиции не ждут

02

session.timeout.ms простоя совсем без ребалансировки при полном отказе

Затронутые партиции ждут

03

session.timeout.ms нужно подбирать под каждое конкретное окружение

Не всегда тривиально

А что от нас хотели-то

Хотелки

Ответ на каждый запрос

Никогда и ни при каких обстоятельствах не терять сообщения

Хотелки

Ответ на каждый запрос

Никогда и ни при каких обстоятельствах не терять сообщения

Сохранять порядок

Стараться не менять порядок запросов и ответов

Хотелки

Ответ на каждый запрос

Никогда и ни при каких обстоятельствах не терять сообщения

Сохранять порядок

Стараться не менять порядок запросов и ответов

Не дублировать ответы

По возможности избегать дублирования ответа по одному запросу

Мы уже читали доки

Kafka транзакции

Звучит хайпово

В это мы еще не вляпались

Kafka транзакции

Звучит хайпово

В это мы еще не вляпались

Помогает сохранять порядок

Вместе с идемпотентным produce

Kafka транзакции

Звучит хайпово

В это мы еще не вляпались

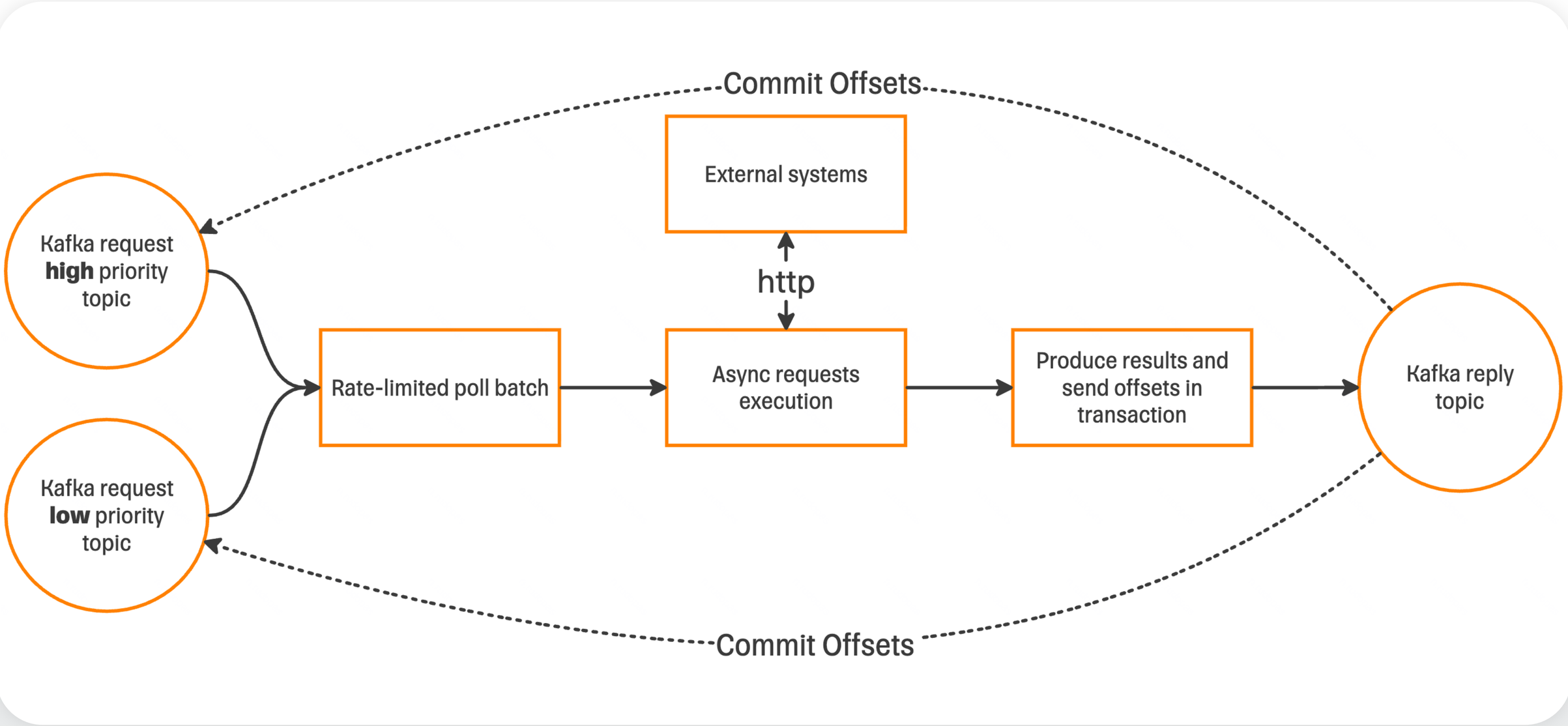
Помогает сохранять порядок

Вместе с идемпотентным produce

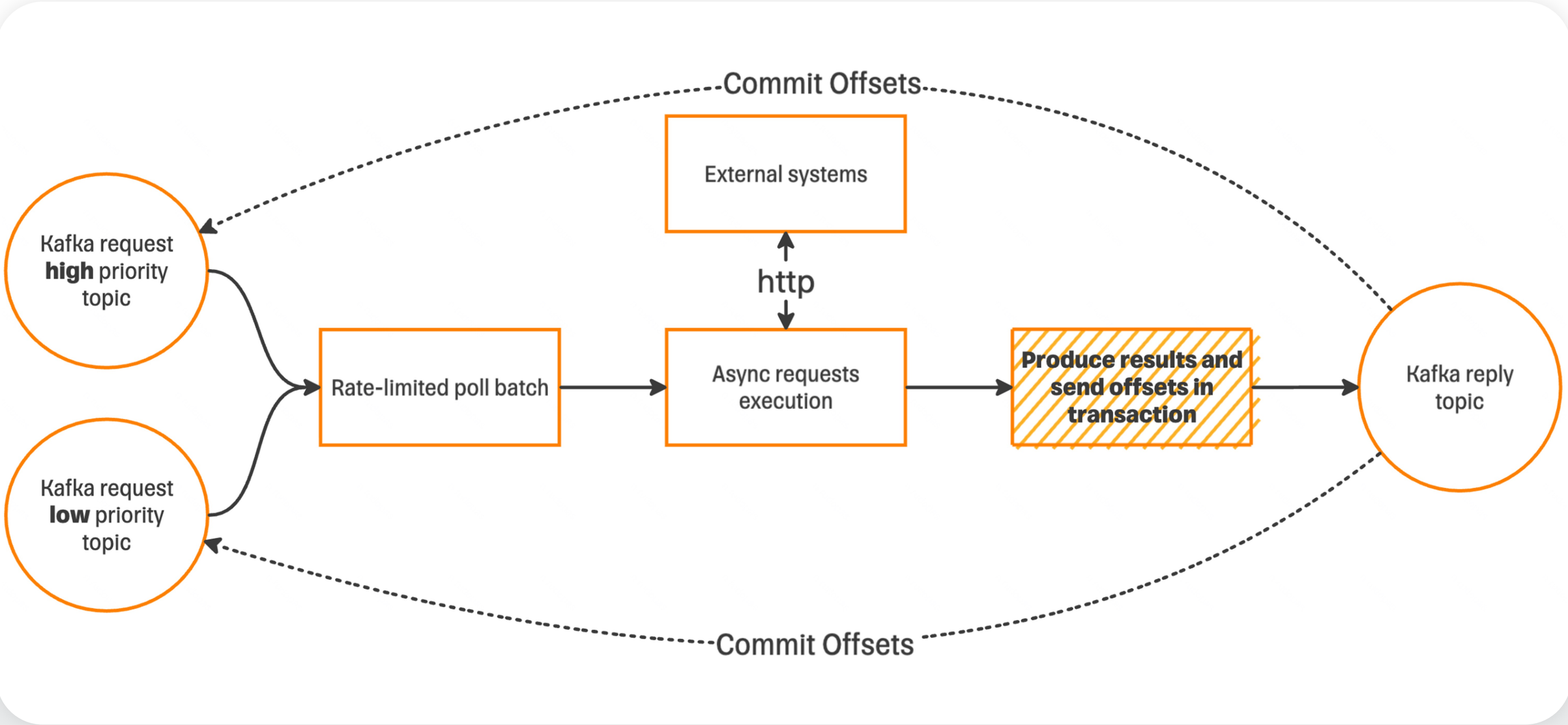
Можно не дублировать ответы

Можно двигать оффсеты внутри транзакции

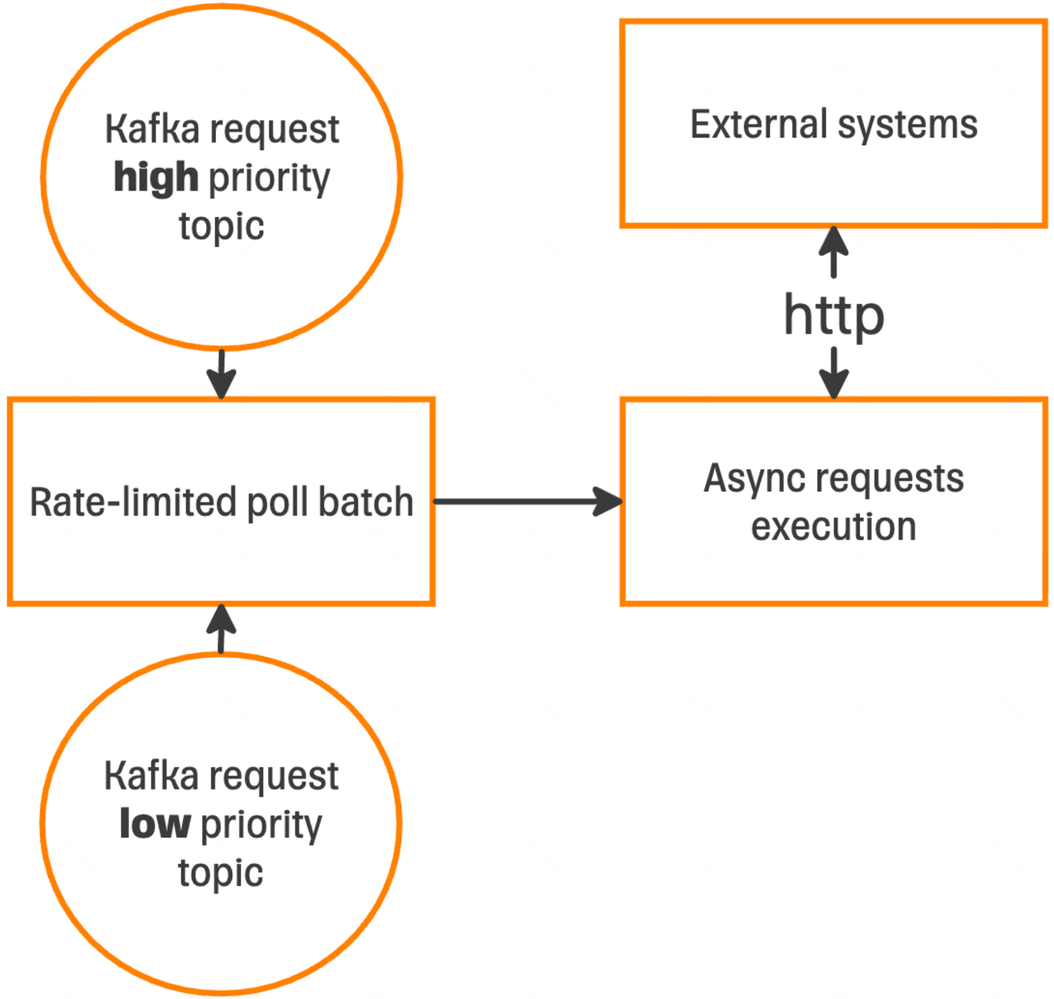
Звучит как план



Звучит как план

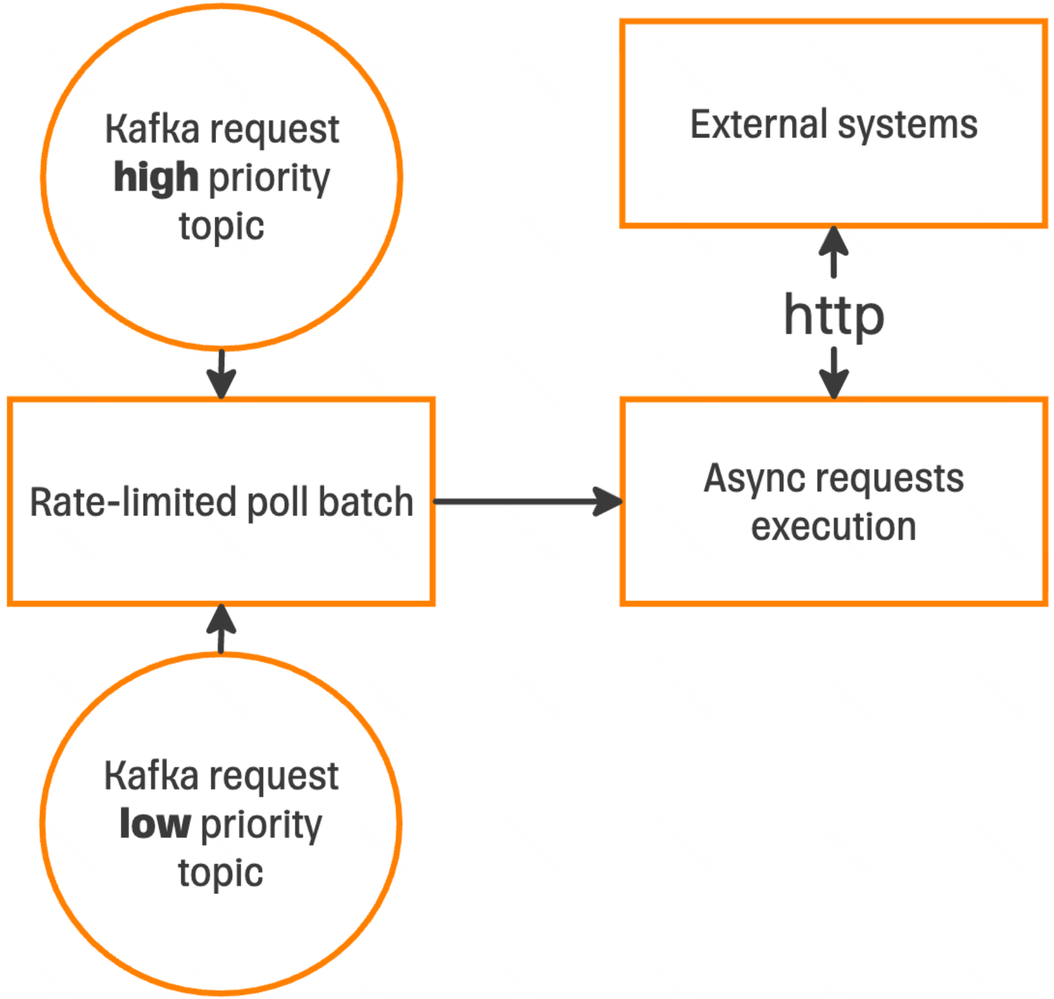


Начало не меняется



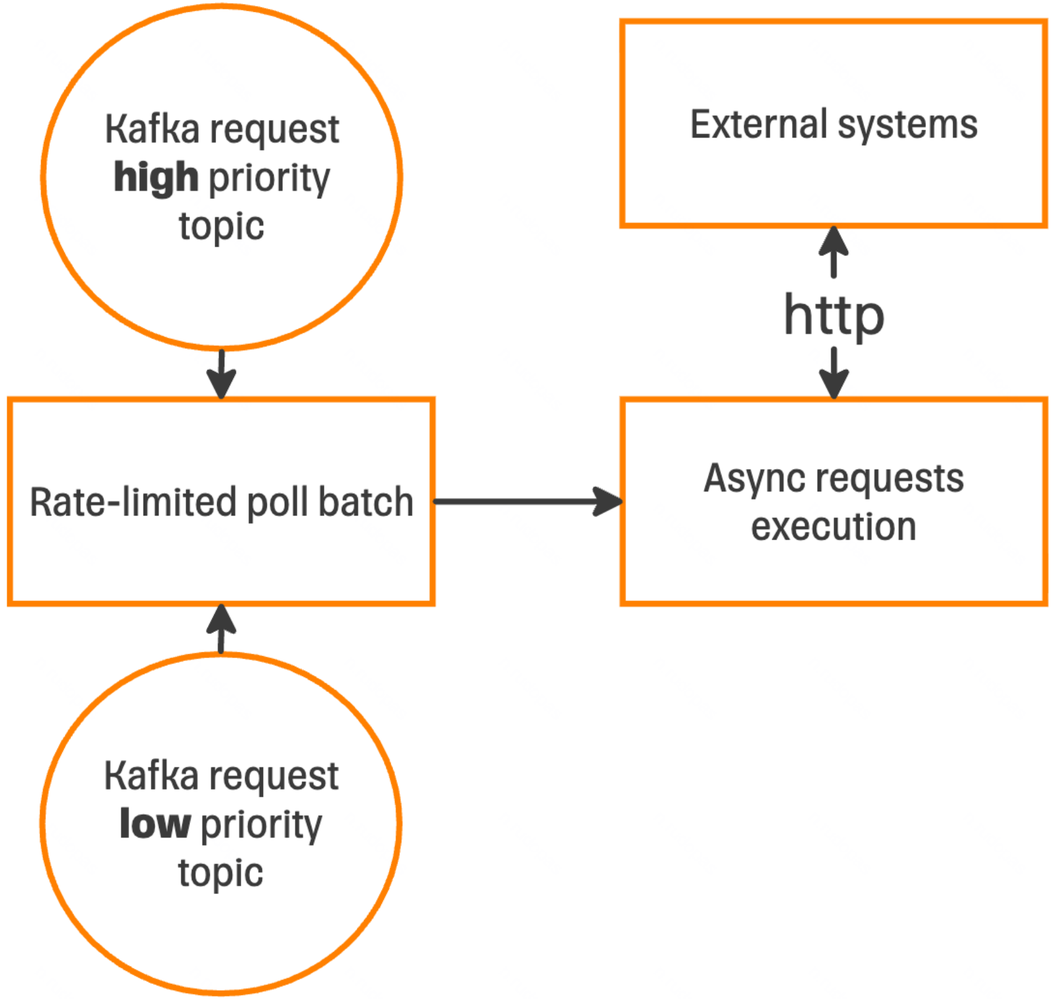

```
147     ) {
148         val bucket = getBucket(maxPollRecords)
149         highPriorityConsumer.subscribe(listOf(highPriorityRequestTopic))
150         lowPriorityConsumer.subscribe(listOf(lowPriorityRequestTopic))
151         while (true) {
152             val highPriorityRequests = pollWithLimit(
153                 bucket,
154                 highPriorityConsumer,
155                 pollTimeout,
156                 maxPollRecords
157             )
158             val lowPriorityRequests = pollWithLimit(
159                 bucket,
160                 lowPriorityConsumer,
161                 pollTimeout,
162                 maxPollRecords
163             )
164
165             val responses = processRequests(
166                 Iterables.concat( ...inputs: highPriorityRequests + lowPriorityRequests),
167                 batchDispatcher
168             )
```

Начало не меняется




```
147     ) {
148         val bucket = getBucket(maxPollRecords)
149         highPriorityConsumer.subscribe(listOf(highPriorityRequestTopic))
150         lowPriorityConsumer.subscribe(listOf(lowPriorityRequestTopic))
151         while (true) {
152             val highPriorityRequests = pollWithLimit(
153                 bucket,
154                 highPriorityConsumer,
155                 pollTimeout,
156                 maxPollRecords
157             )
158             val lowPriorityRequests = pollWithLimit(
159                 bucket,
160                 lowPriorityConsumer,
161                 pollTimeout,
162                 maxPollRecords
163             )
164
165             val responses = processRequests(
166                 Iterables.concat( ...inputs: highPriorityRequests + lowPriorityRequests),
167                 batchDispatcher
168             )
```

Начало не меняется




```
147     ) {
148         val bucket = getBucket(maxPollRecords)
149         highPriorityConsumer.subscribe(listOf(highPriorityRequestTopic))
150         lowPriorityConsumer.subscribe(listOf(lowPriorityRequestTopic))
151         while (true) {
152             val highPriorityRequests = pollWithLimit(
153                 bucket,
154                 highPriorityConsumer,
155                 pollTimeout,
156                 maxPollRecords
157             )
158             val lowPriorityRequests = pollWithLimit(
159                 bucket,
160                 lowPriorityConsumer,
161                 pollTimeout,
162                 maxPollRecords
163             )
164
165             val responses = processRequests(
166                 Iterables.concat( ...inputs: highPriorityRequests + lowPriorityRequests),
167                 batchDispatcher
168             )
```

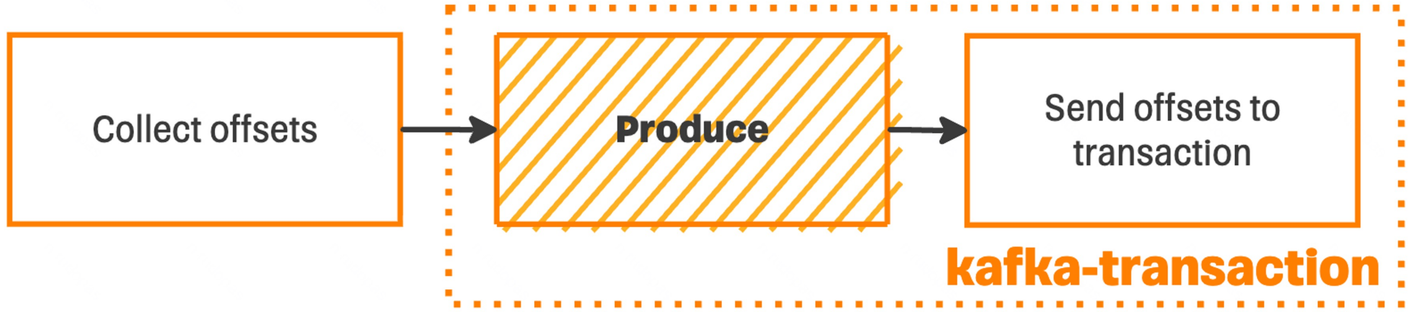
Собираем оффсеты

```kotlin
fun getOffsets(
    consumerRecords: Iterable<ConsumerRecord<String, Request>>
): Map<TopicPartition, OffsetAndMetadata> =
    consumerRecords
        .groupingBy { TopicPartition(it.topic(), it.partition()) }
        .fold(
            initialValueSelector = { _, it ->
                OffsetAndMetadata( offset: it.offset() + 1)
            },
            operation = { _, _, it ->
                OffsetAndMetadata( offset: it.offset() + 1)
            }
        )
```

```kotlin
fun startProcessing(
) {
    val bucket = getBucket(maxPollRecords)
    highPriorityConsumer.subscribe(listOf(highPriorityRequestTopic))
    lowPriorityConsumer.subscribe(listOf(lowPriorityRequestTopic))
    while (true) {
        val highPriorityRequests = pollWithLimit(
            bucket,
            highPriorityConsumer,
            pollTimeout,
            maxPollRecords
        )
        val lowPriorityRequests = pollWithLimit(
            bucket,
            lowPriorityConsumer,
            pollTimeout,
            maxPollRecords
        )

        val responses = processRequests(
            Iterables.concat( ...inputs: highPriorityRequests + lowPriorityRequests),
            batchDispatcher
        )

        val offsets = listOf(
            highPriorityConsumer.groupMetadata() to getOffsets(highPriorityRequests),
            lowPriorityConsumer.groupMetadata() to getOffsets(lowPriorityRequests)
        )

        produce(responses, responseTopic, producer, offsets)
```

Продьюсим

```
79  fun produce(
80      responses: List<Pair<String, Response>>,
81      responseTopic: String,
82      producer: KafkaProducer<String, Response>,
83      offsets: Collection<
84          Pair<
85              ConsumerGroupMetadata,
86              Map<TopicPartition, OffsetAndMetadata>
87              >
88          >
89  ) {
90      try {
91          producer.beginTransaction()
92          responses.forEach { response ->
93              producer.send(
94                  ProducerRecord(
95                      responseTopic,
96                      response.first,
97                      response.second
98                  )
99              )
100         }
101         offsets.forEach { (consumerGroupMetadata, offset) ->
102             producer.sendOffsetsToTransaction(
103                 offset,
104                 consumerGroupMetadata
105             )
106         }
107         producer.commitTransaction()
108     } catch (e: Exception) {
109         producer.abortTransaction()
110         throw e
111     }
112 }
```




```
165         val responses = processRequests(
166             Iterables.concat( ...inputs: highPriorityRequests + lowPriorityRequests),
167             batchDispatcher
168         )
169
170         val offsets = listOf(
171             highPriorityConsumer.groupMetadata() to getOffsets(highPriorityRequests),
172             lowPriorityConsumer.groupMetadata() to getOffsets(lowPriorityRequests)
173         )
174
175         produce(responses, responseTopic, producer, offsets)
176
177         logMetrics(highPriorityRequests)
178         logMetrics(lowPriorityRequests)
179     }
```

Продьюсим

```
79  fun produce(
80      responses: List<Pair<String, Response>>,
81      responseTopic: String,
82      producer: KafkaProducer<String, Response>,
83      offsets: Collection<
84          Pair<
85              ConsumerGroupMetadata,
86              Map<TopicPartition, OffsetAndMetadata>
87              >
88          >
89  ) {
90      try {
91          producer.beginTransaction()
92          responses.forEach { response ->
93              producer.send(
94                  ProducerRecord(
95                      responseTopic,
96                      response.first,
97                      response.second
98                  )
99              )
100         }
101         offsets.forEach { (consumerGroupMetadata, offset) ->
102             producer.sendOffsetsToTransaction(
103                 offset,
104                 consumerGroupMetadata
105             )
106         }
107         producer.commitTransaction()
108     } catch (e: Exception) {
109         producer.abortTransaction()
110         throw e
111     }
112 }
```

```
165         val responses = processRequests(
166             Iterables.concat( ...inputs: highPriorityRequests + lowPriorityRequests),
167             batchDispatcher
168         )
169
170         val offsets = listOf(
171             highPriorityConsumer.groupMetadata() to getOffsets(highPriorityRequests),
172             lowPriorityConsumer.groupMetadata() to getOffsets(lowPriorityRequests)
173         )
174
175         produce(responses, responseTopic, producer, offsets)
176
177         logMetrics(highPriorityRequests)
178         logMetrics(lowPriorityRequests)
179     }
```

Создаем producer

```
186     val env = mapOf(
187         "drygala-0" to "drygala-trn-id-0",
188         "drygala-1" to "drygala-trn-id-1",
189         "drygala-2" to "drygala-trn-id-2",
190         "drygala-3" to "drygala-trn-id-3"
191     )
192
193     val producer = KafkaProducer<String, Response>(
194         mapOf(
195             ProducerConfig.ENABLE_IDEMPOTENCE_CONFIG to true,
196             ProducerConfig.TRANSACTIONAL_ID_CONFIG to env[hostname]
197         )
198     )
199     producer.initTransactions()
200
201 }
```

Создаем producer

```
186     val env = mapOf(
187         "drygala-0" to "drygala-trn-id-0",
188         "drygala-1" to "drygala-trn-id-1",
189         "drygala-2" to "drygala-trn-id-2",
190         "drygala-3" to "drygala-trn-id-3"
191     )
192
193     val producer = KafkaProducer<String, Response>(
194         mapOf(
195             ProducerConfig.ENABLE_IDEMPOTENCE_CONFIG to true,
196             ProducerConfig.TRANSACTIONAL_ID_CONFIG to env[hostname]
197         )
198     )
199     producer.initTransactions()
200
201 }
```

Создаем producer

На наших Kafka transactional-id ограничены по формату и их заданное количество

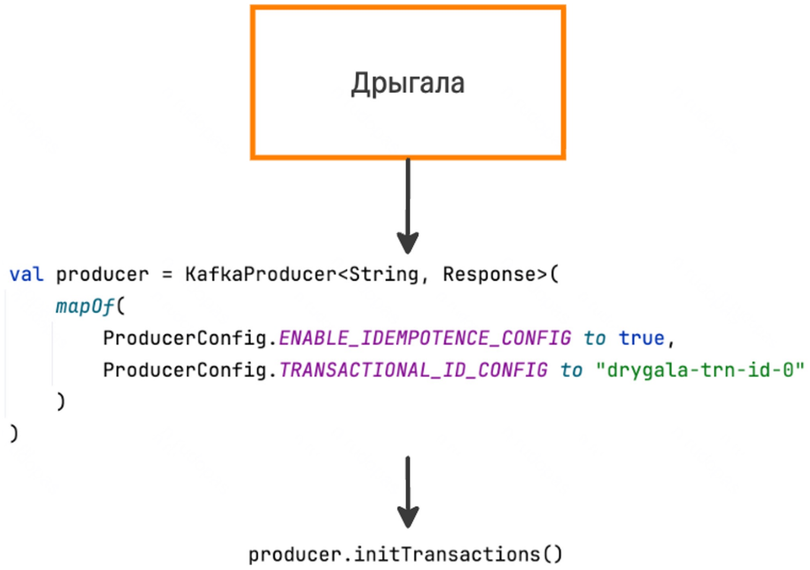
```
186     val env = mapOf(
187         "drygala-0" to "drygala-trn-id-0",
188         "drygala-1" to "drygala-trn-id-1",
189         "drygala-2" to "drygala-trn-id-2",
190         "drygala-3" to "drygala-trn-id-3"
191     )
192
193     val producer = KafkaProducer<String, Response>(
194         mapOf(
195             ProducerConfig.ENABLE_IDEMPOTENCE_CONFIG to true,
196             ProducerConfig.TRANSACTIONAL_ID_CONFIG to env[hostname]
197         )
198     )
199     producer.initTransactions()
200
201 }
```

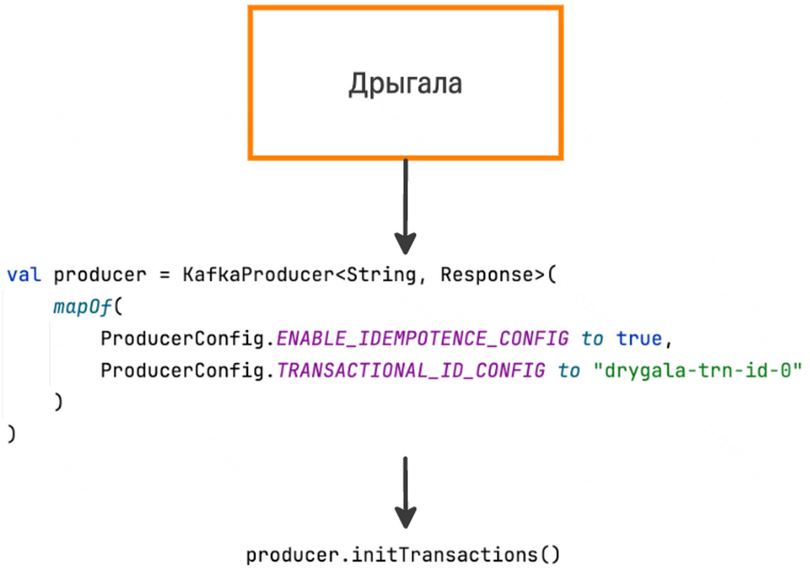
Создаем producer

- На наших Kafka transactional-id ограничены по формату и их заданное количество
- Он должен быть уникальным в рамках брокера и это тяжело контролировать

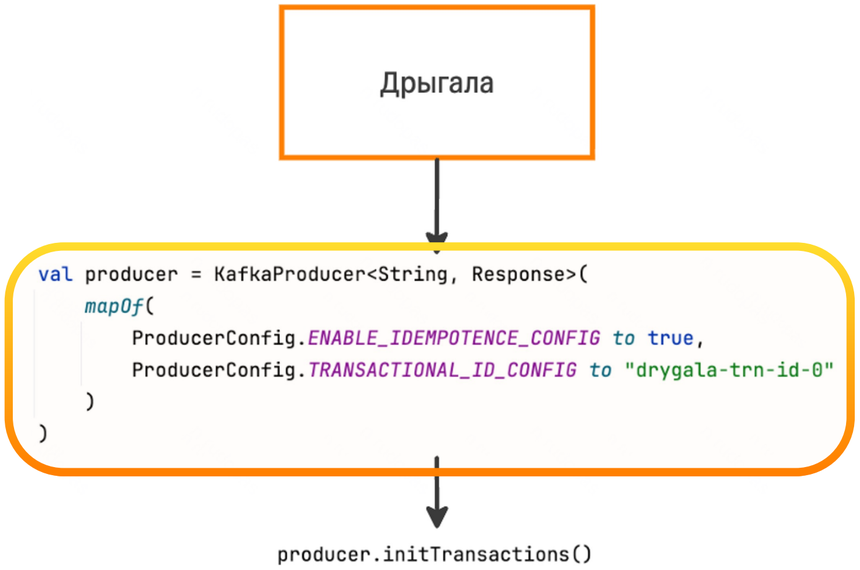
```
186     val env = mapOf(
187         "drygala-0" to "drygala-trn-id-0",
188         "drygala-1" to "drygala-trn-id-1",
189         "drygala-2" to "drygala-trn-id-2",
190         "drygala-3" to "drygala-trn-id-3"
191     )
192
193     val producer = KafkaProducer<String, Response>(
194         mapOf(
195             ProducerConfig.ENABLE_IDEMPOTENCE_CONFIG to true,
196             ProducerConfig.TRANSACTIONAL_ID_CONFIG to env[hostname]
197         )
198     )
199     producer.initTransactions()
200
201 }
```

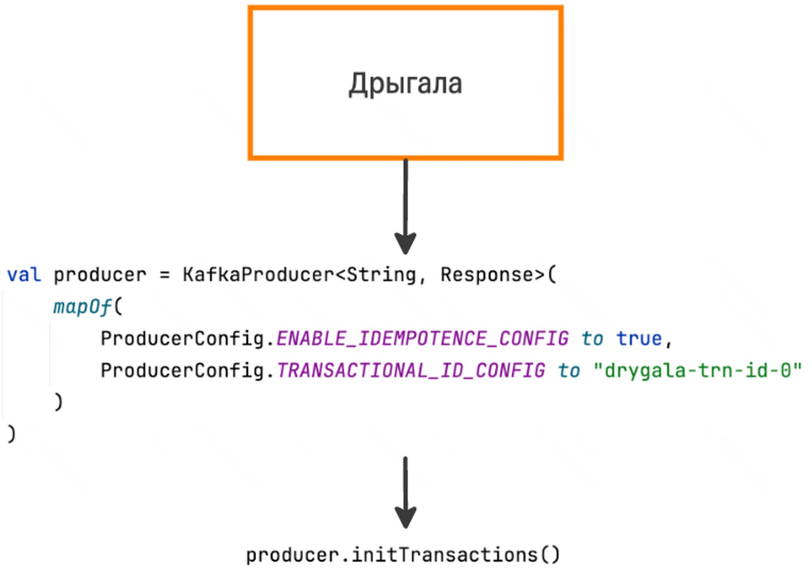
ProducerFencedException



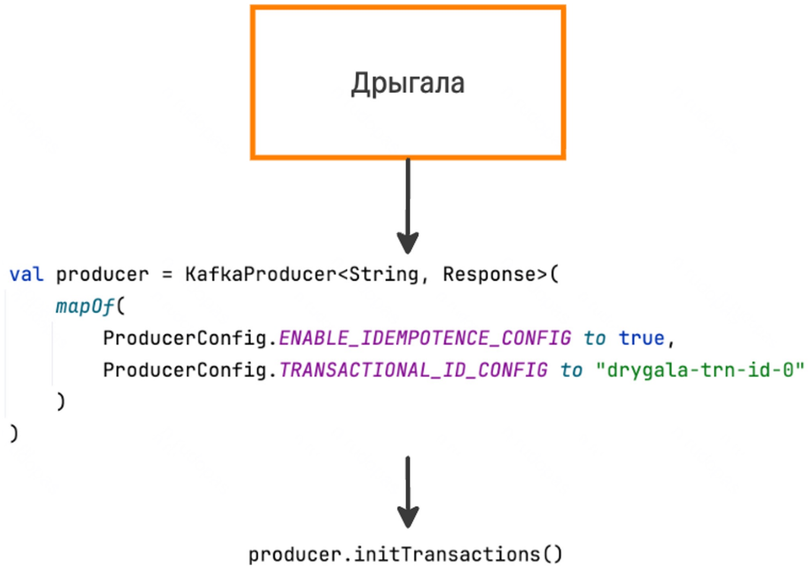


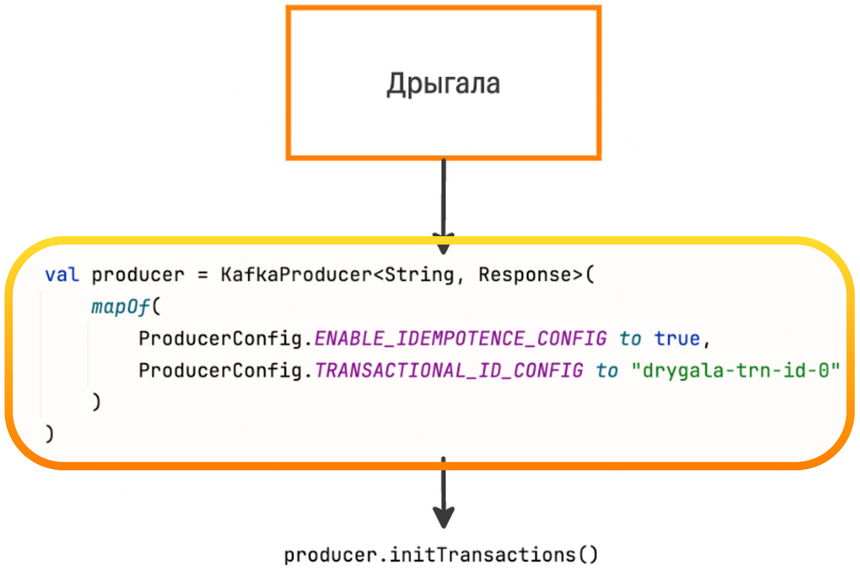
ProducerFencedException



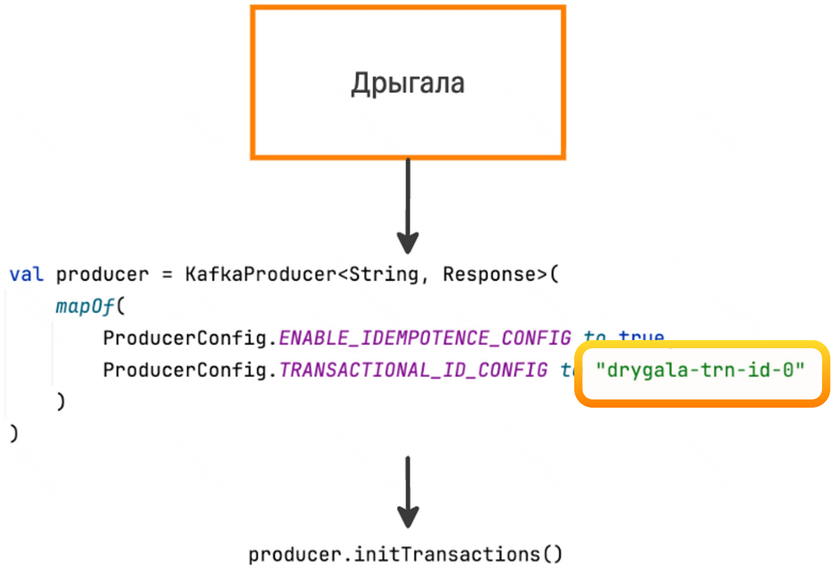


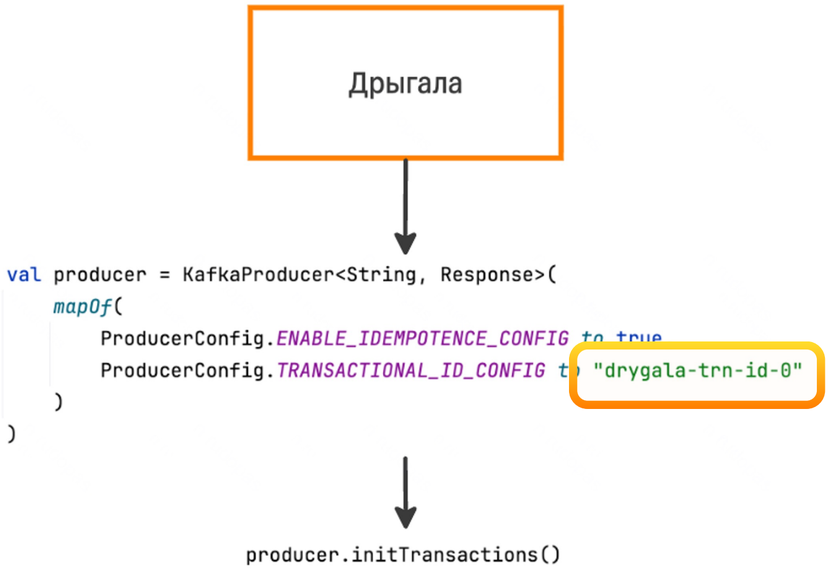
ProducerFencedException



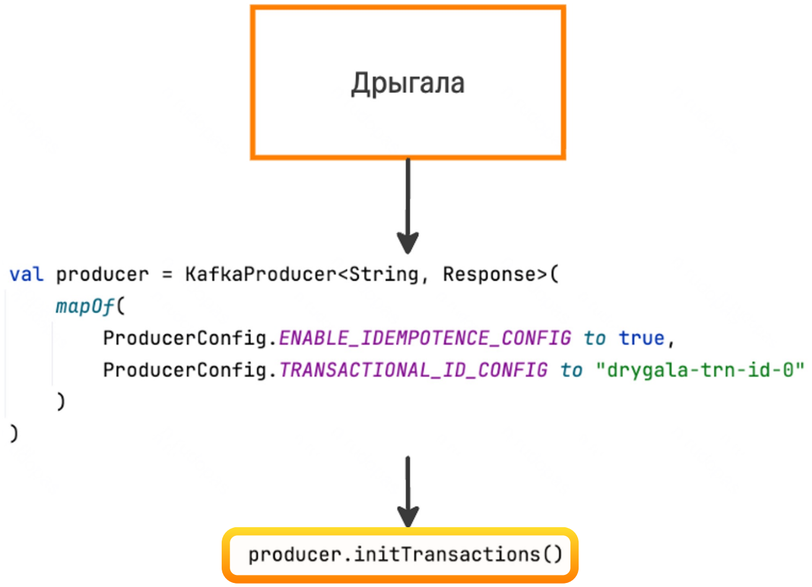


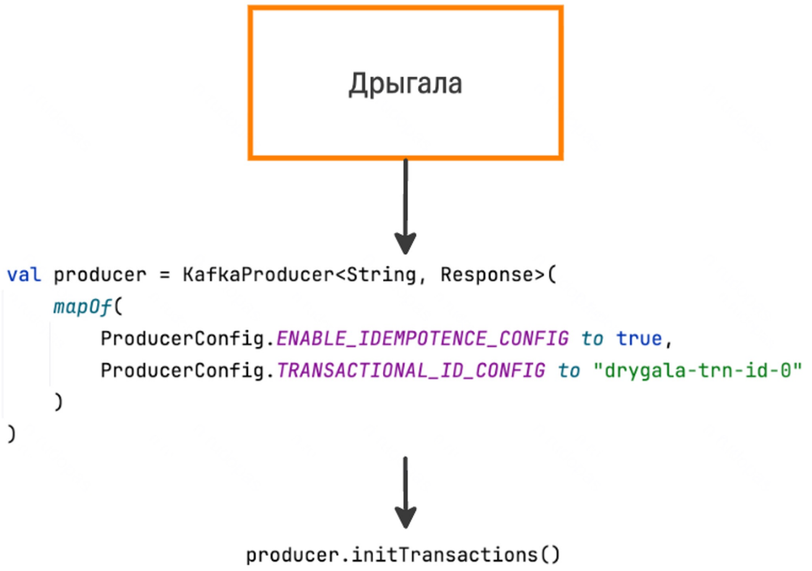
ProducerFencedException



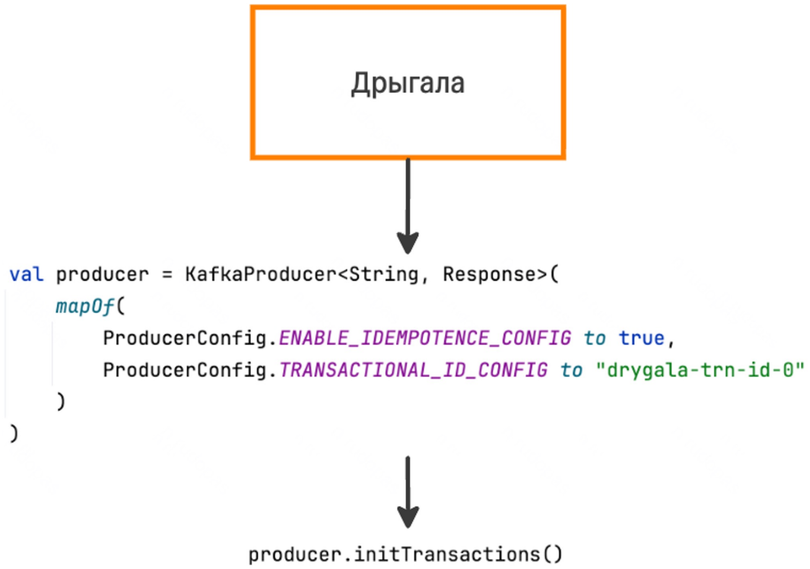


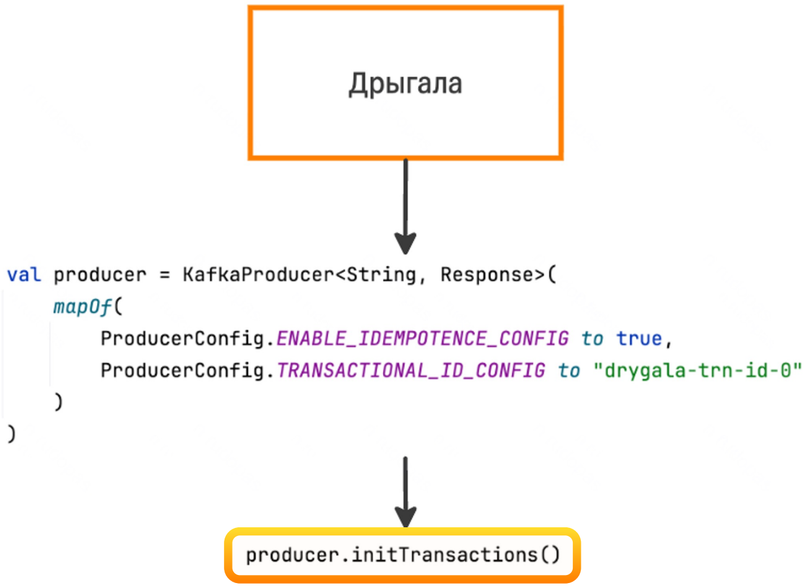
ProducerFencedException



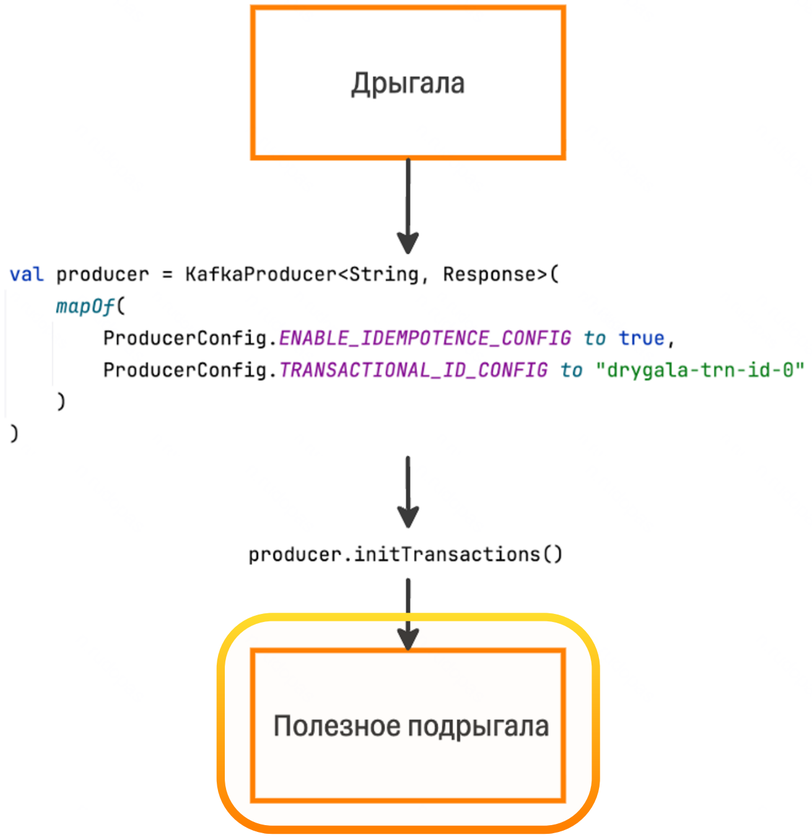


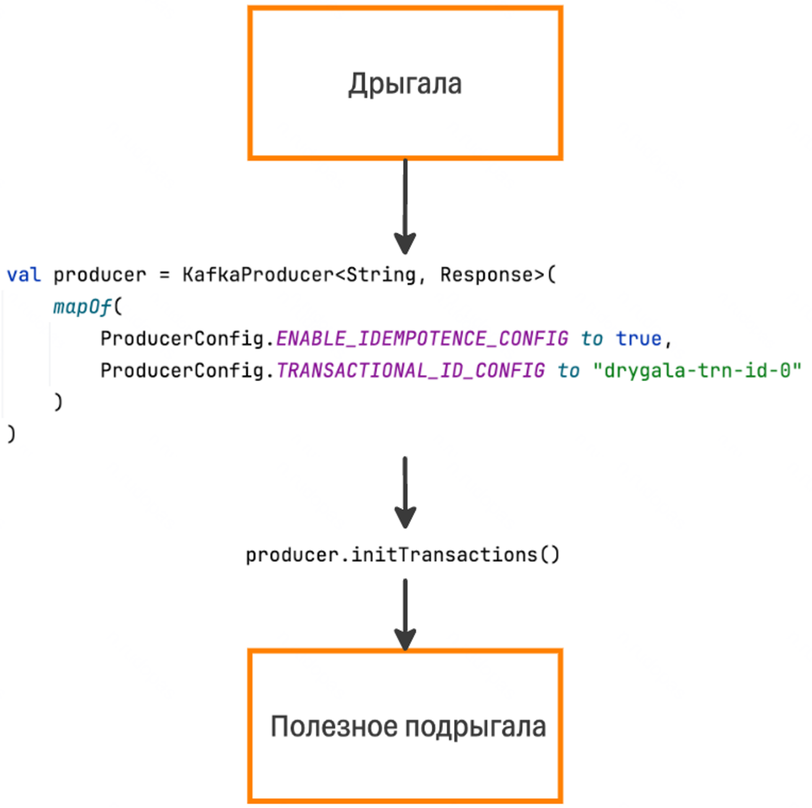
ProducerFencedException



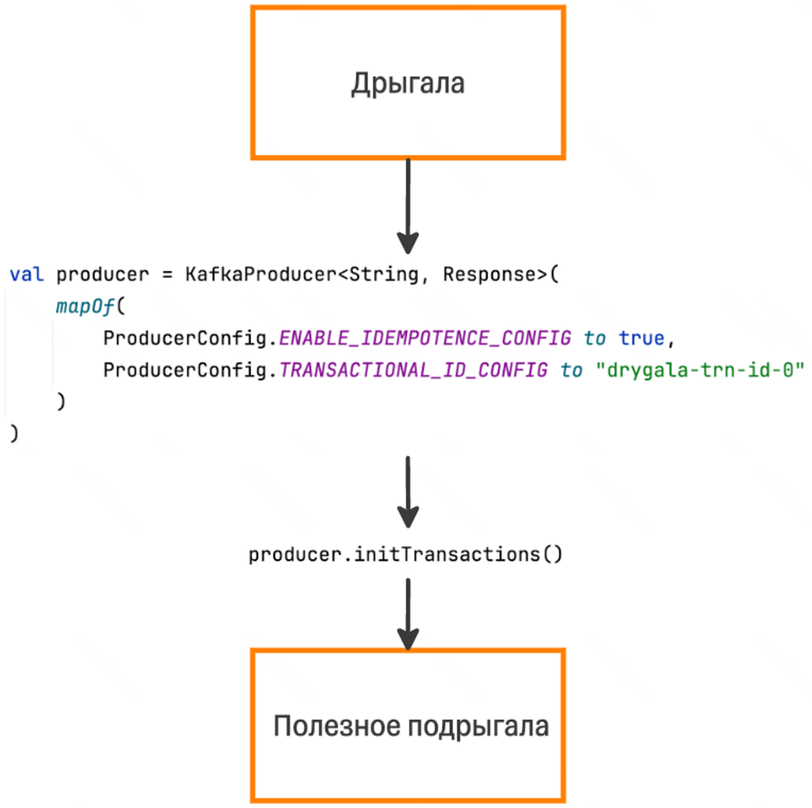


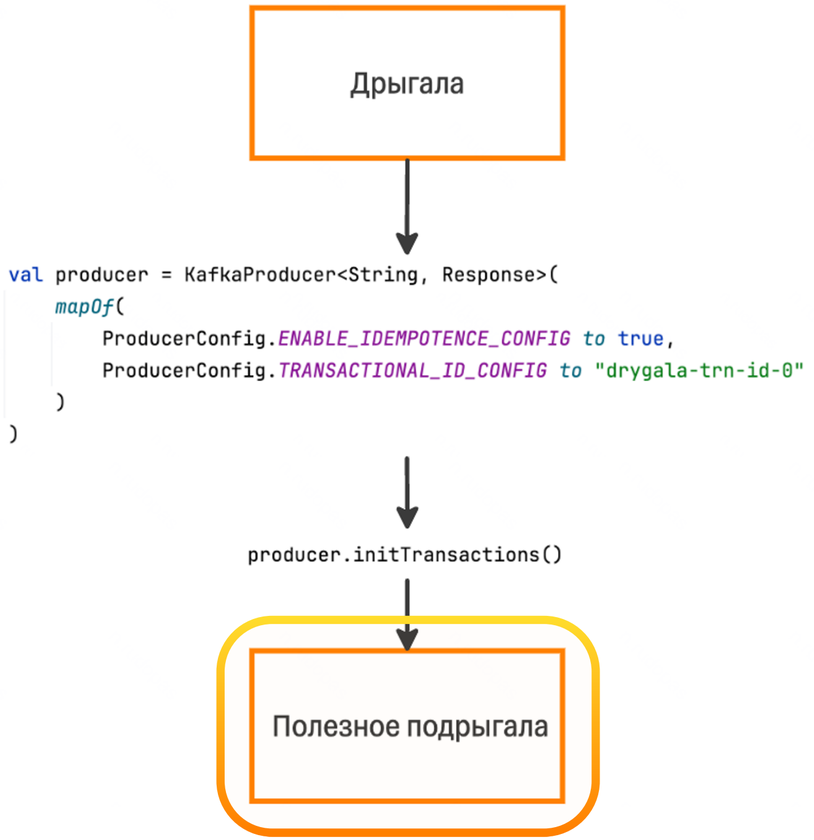
ProducerFencedException



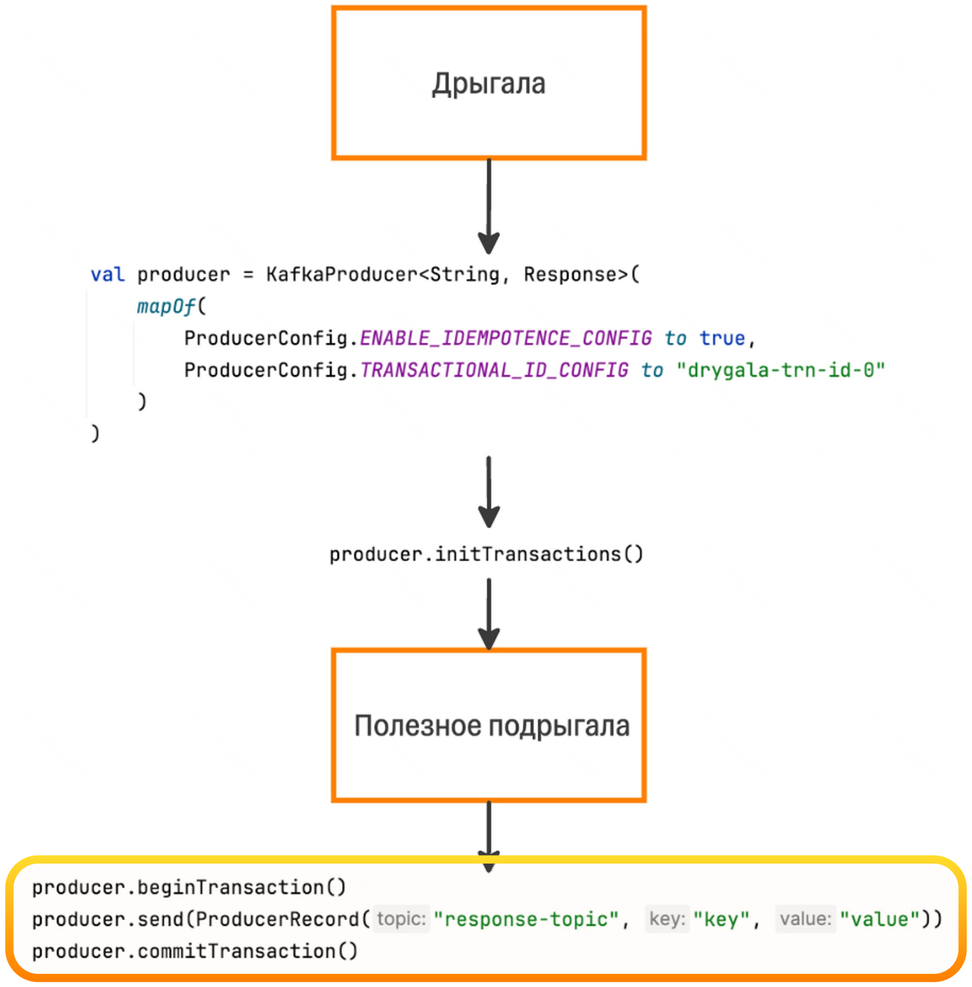


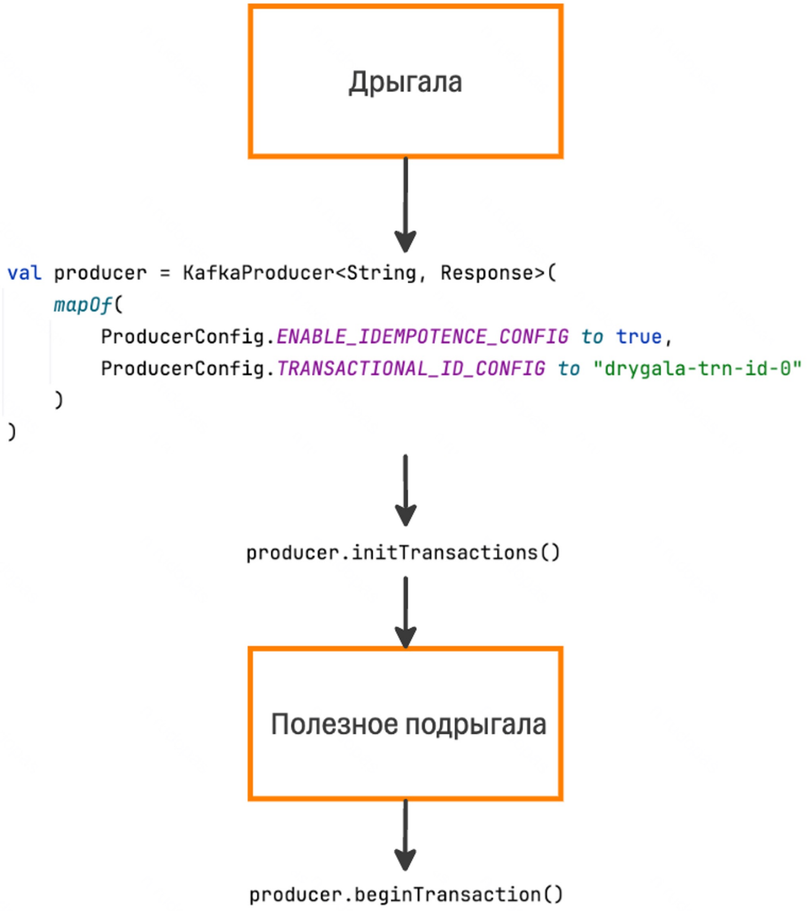
ProducerFencedException



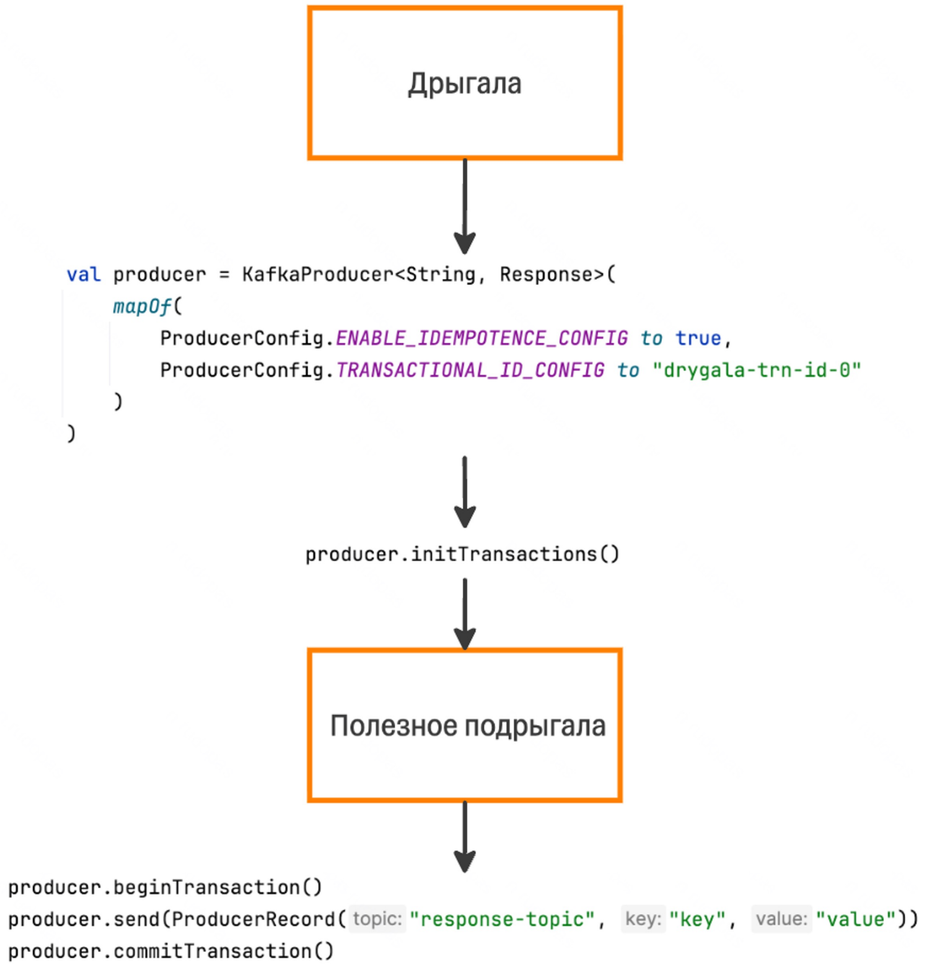


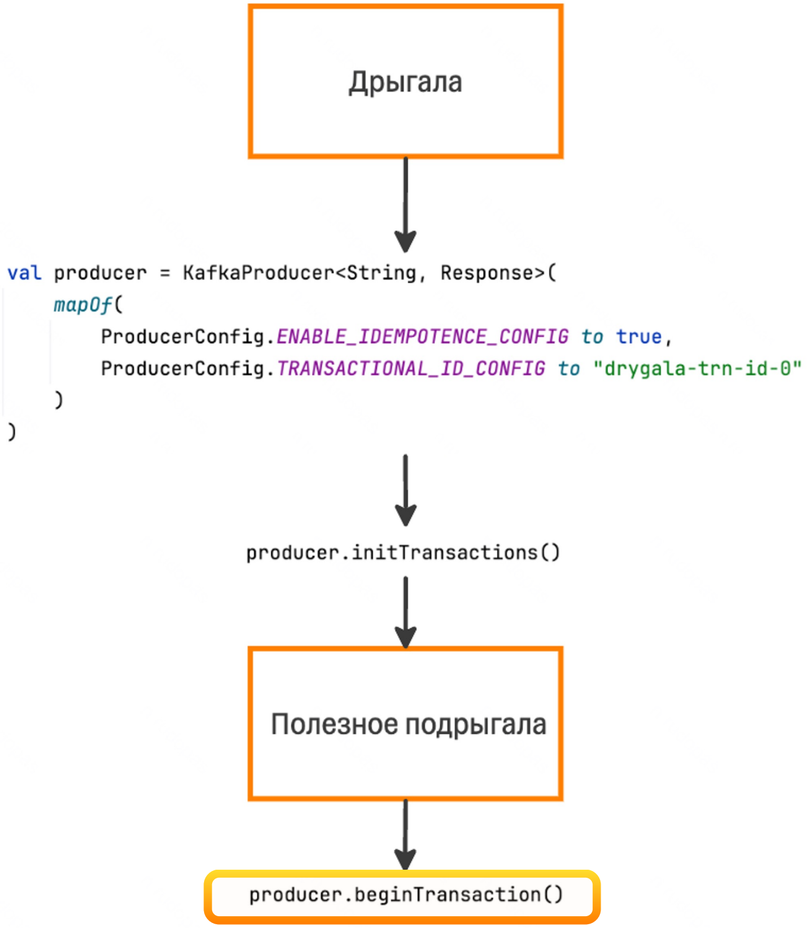
ProducerFencedException



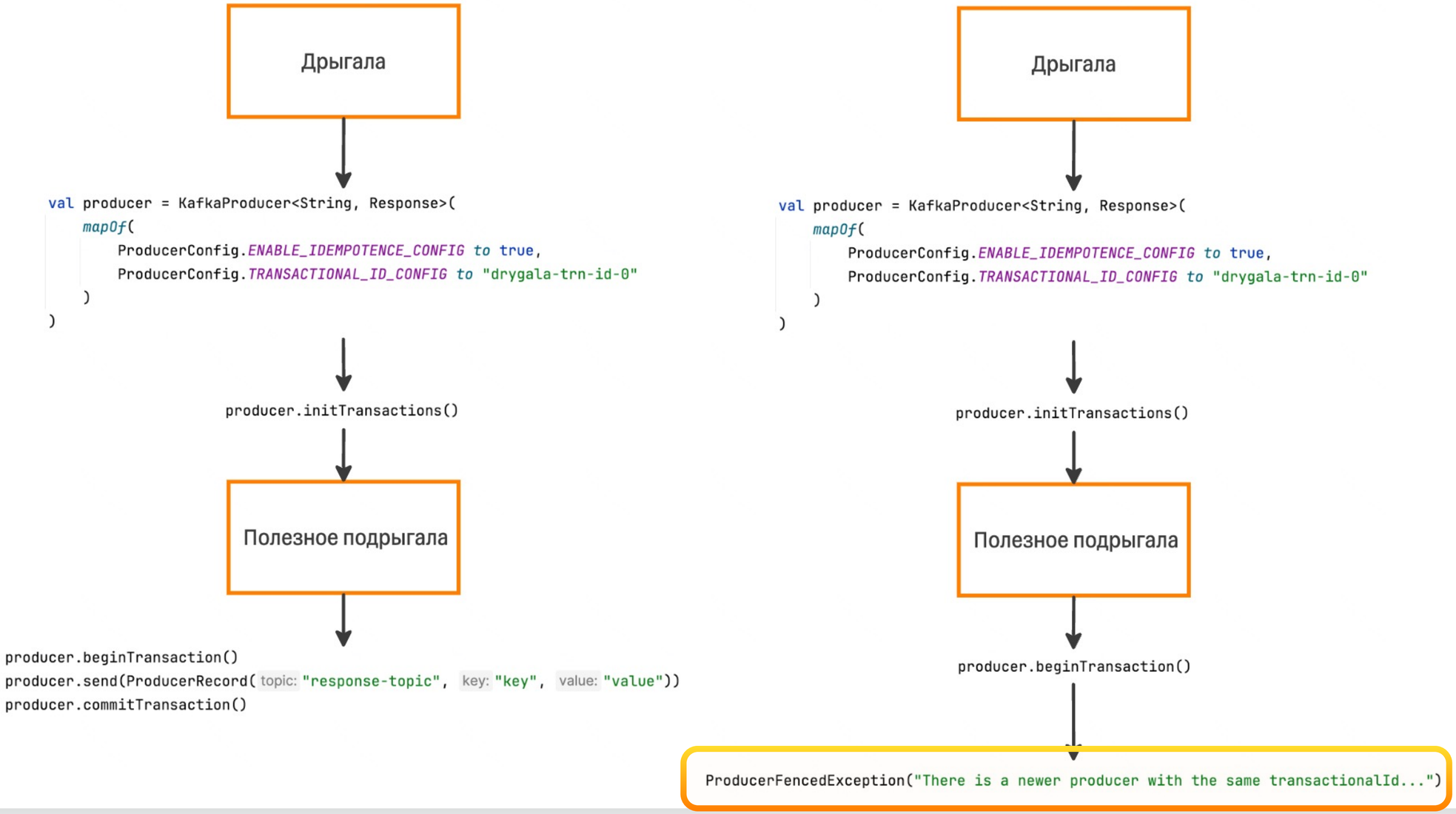


ProducerFencedException





ProducerFencedException



Красота?

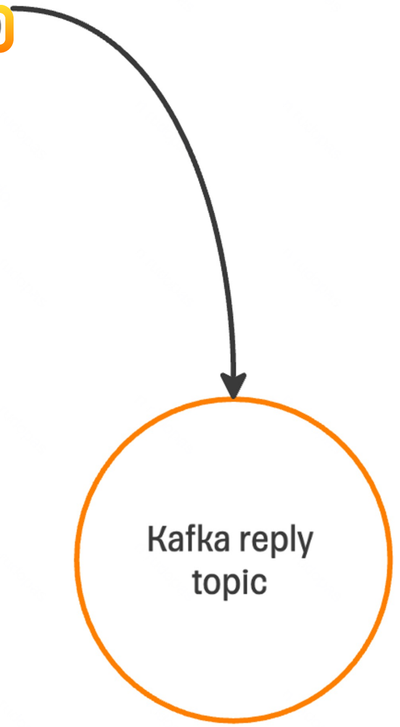
О чем мы сразу прочитали

Магия вне Хогвартса

```
9   fun produce(
10      responseTopic: String,
11      producer: KafkaProducer<String, String>,
12      consumerMetadata: ConsumerGroupMetadata,
13      offsets: Map<TopicPartition, OffsetAndMetadata>
14  ) {
15      try {
16          producer.beginTransaction()
17          producer.send(ProducerRecord(responseTopic, key: "messageKey", value: "messageValue"))
18          producer.sendOffsetsToTransaction(offsets, consumerMetadata)
19          producer.commitTransaction()
20      } catch (e: Exception) {
21          producer.abortTransaction()
22          throw e
23      }
24  }
```

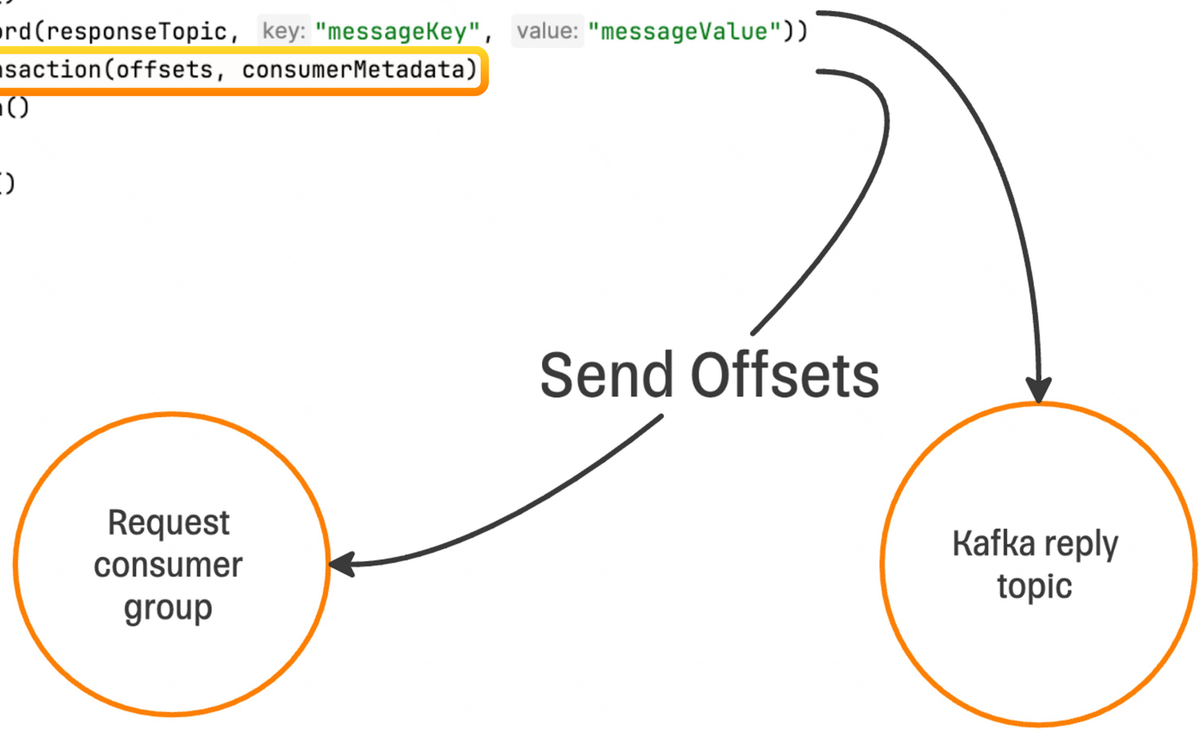
Магия вне Хогвартса

```
9   fun produce(
10      responseTopic: String,
11      producer: KafkaProducer<String, String>,
12      consumerMetadata: ConsumerGroupMetadata,
13      offsets: Map<TopicPartition, OffsetAndMetadata>
14  ) {
15      try {
16          producer.beginTransaction()
17          producer.send(ProducerRecord(responseTopic, key: "messageKey", value: "messageValue"))
18          producer.sendOffsetsToTransaction(offsets, consumerMetadata)
19          producer.commitTransaction()
20      } catch (e: Exception) {
21          producer.abortTransaction()
22          throw e
23      }
24  }
```



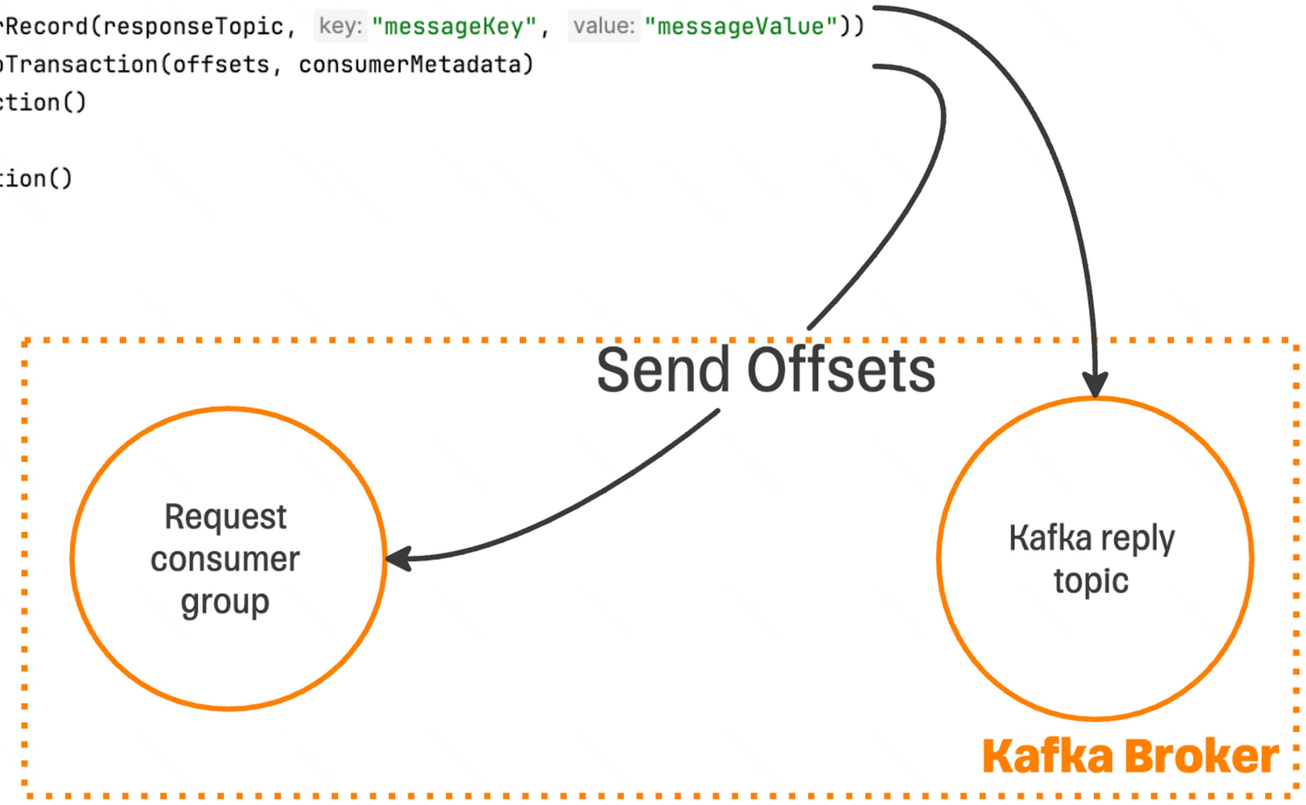
Магия вне Хогвартса

```kotlin
fun produce(
    responseTopic: String,
    producer: KafkaProducer<String, String>,
    consumerMetadata: ConsumerGroupMetadata,
    offsets: Map<TopicPartition, OffsetAndMetadata>
) {
    try {
        producer.beginTransaction()
        producer.send(ProducerRecord(responseTopic, key: "messageKey", value: "messageValue"))
        producer.sendOffsetsToTransaction(offsets, consumerMetadata)
        producer.commitTransaction()
    } catch (e: Exception) {
        producer.abortTransaction()
        throw e
    }
}
```



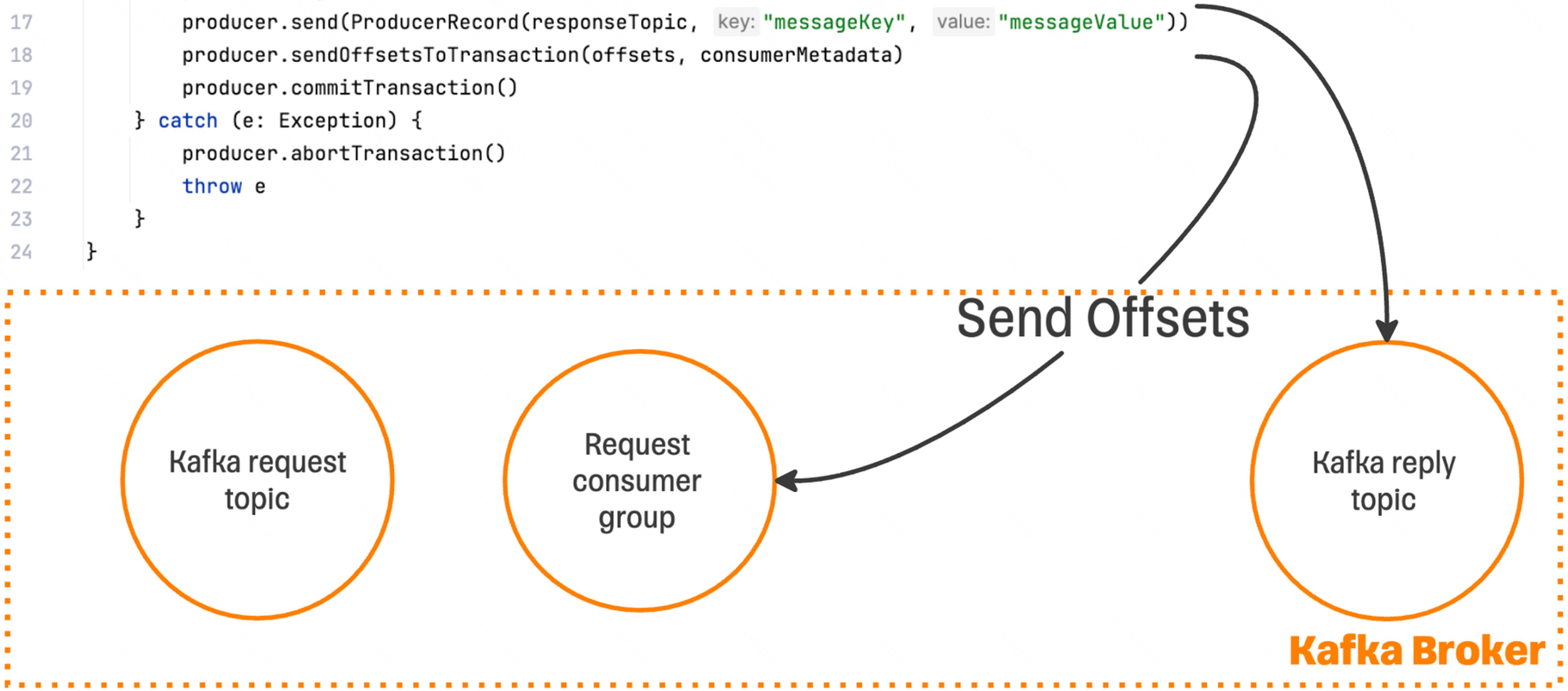
Оффсеты должны быть у брокера

```kotlin
fun produce(
    responseTopic: String,
    producer: KafkaProducer<String, String>,
    consumerMetadata: ConsumerGroupMetadata,
    offsets: Map<TopicPartition, OffsetAndMetadata>
) {
    try {
        producer.beginTransaction()
        producer.send(ProducerRecord(responseTopic, key: "messageKey", value: "messageValue"))
        producer.sendOffsetsToTransaction(offsets, consumerMetadata)
        producer.commitTransaction()
    } catch (e: Exception) {
        producer.abortTransaction()
        throw e
    }
}
```



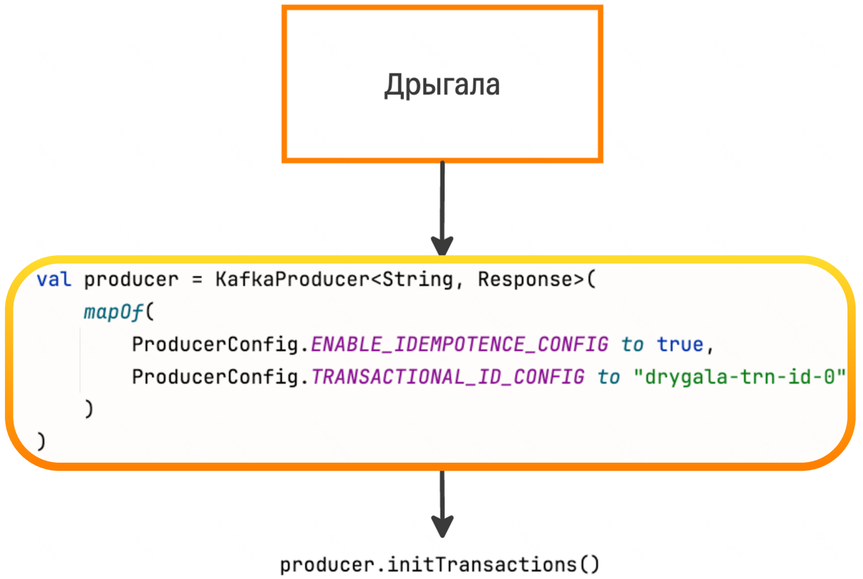
Все топики должны быть на одном брокере

```kotlin
fun produce(
    responseTopic: String,
    producer: KafkaProducer<String, String>,
    consumerMetadata: ConsumerGroupMetadata,
    offsets: Map<TopicPartition, OffsetAndMetadata>
) {
    try {
        producer.beginTransaction()
        producer.send(ProducerRecord(responseTopic, key: "messageKey", value: "messageValue"))
        producer.sendOffsetsToTransaction(offsets, consumerMetadata)
        producer.commitTransaction()
    } catch (e: Exception) {
        producer.abortTransaction()
        throw e
    }
}
```

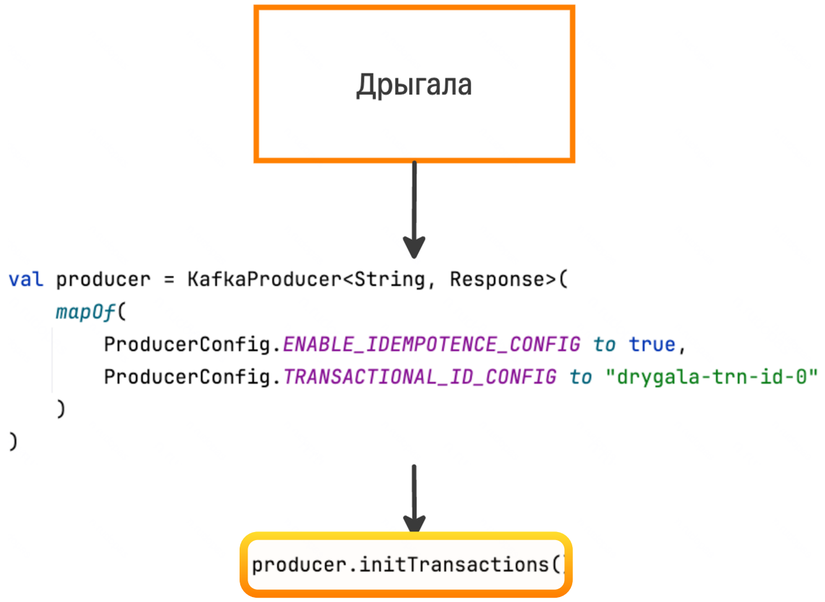


О чем мы не сразу прочитали

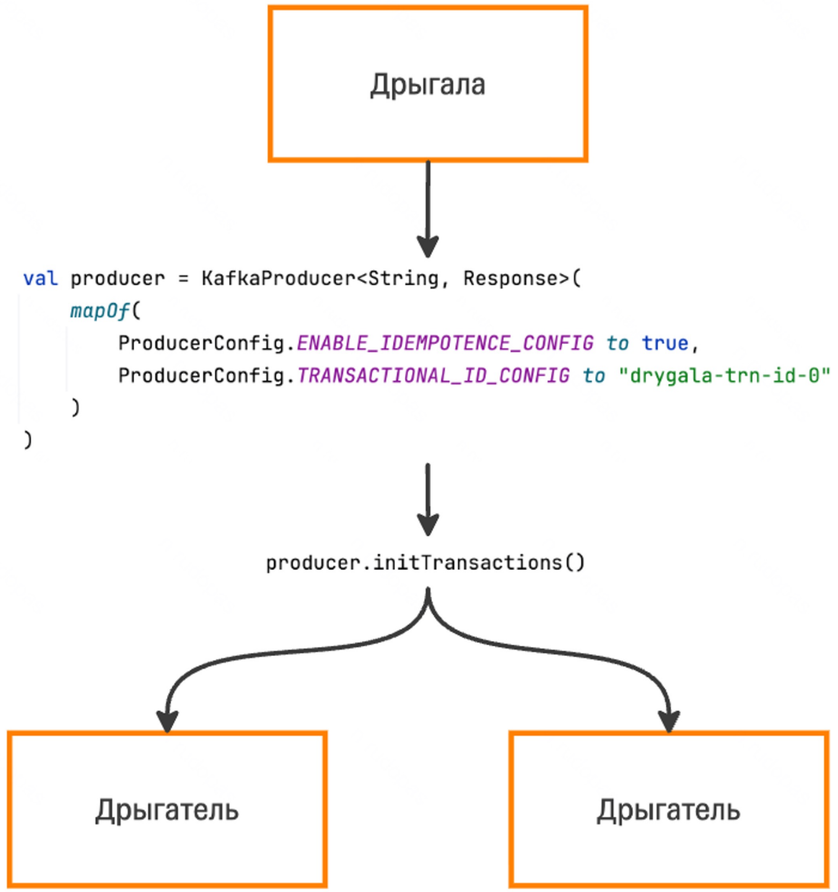
Transactional Producer



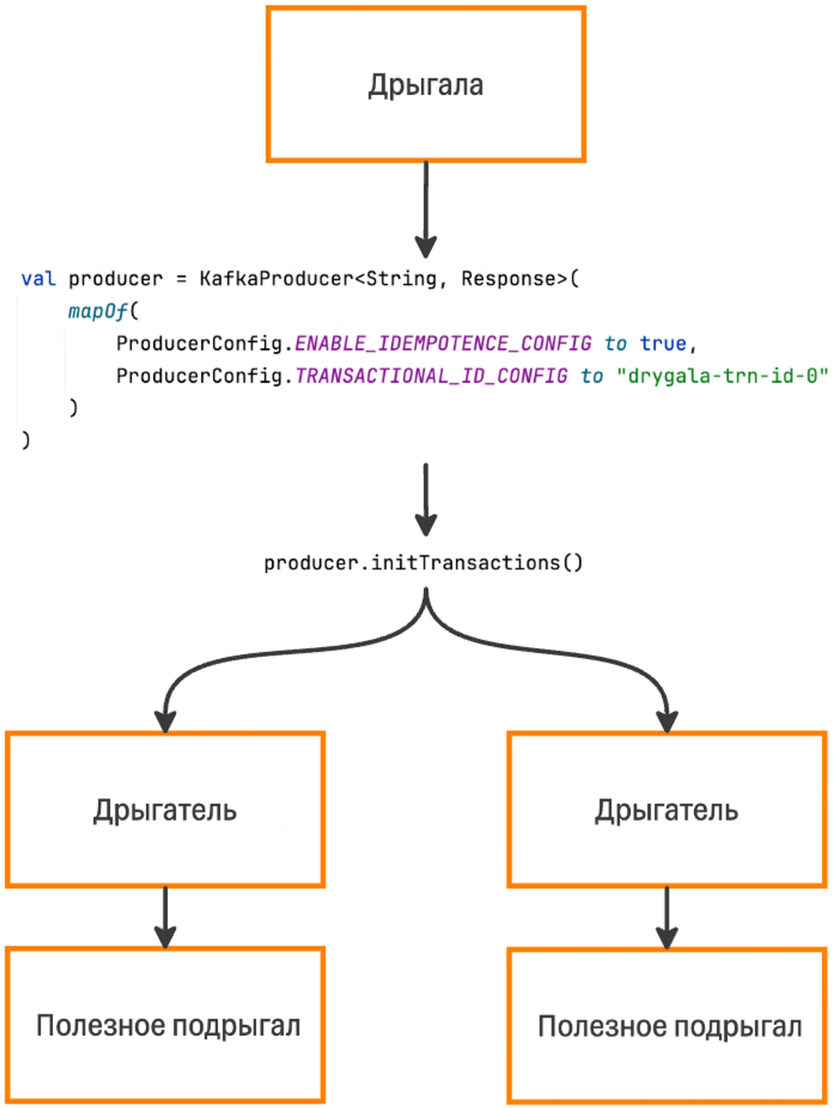
Transactional Producer



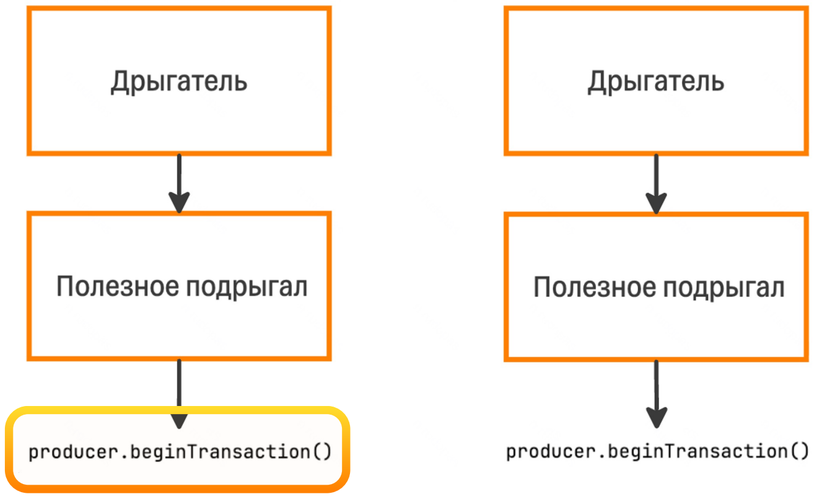
Transactional Producer



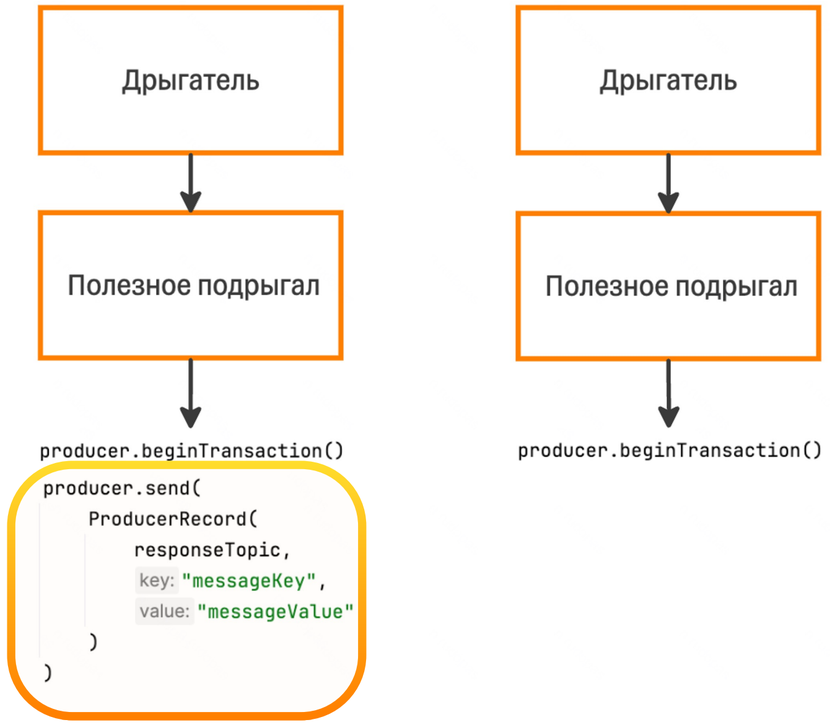
Transactional Producer



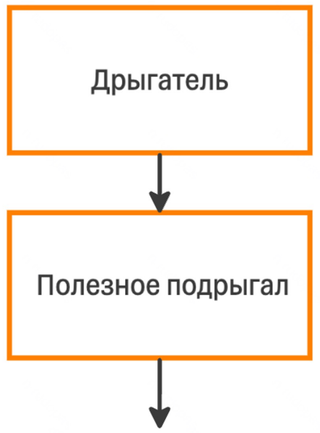
Transactional Producer



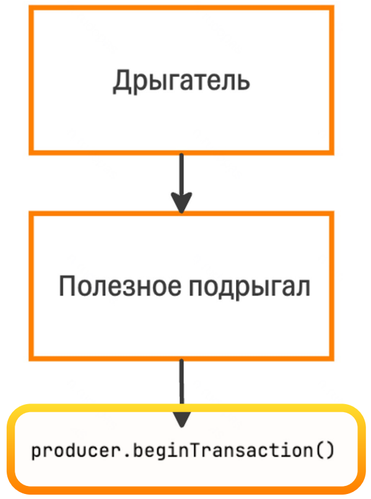
Transactional Producer



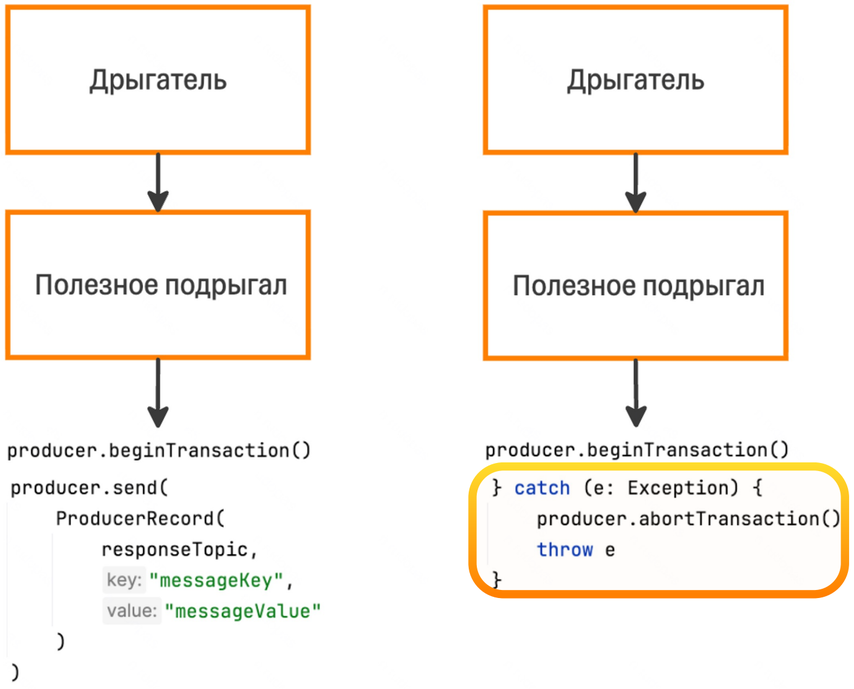
Transactional Producer



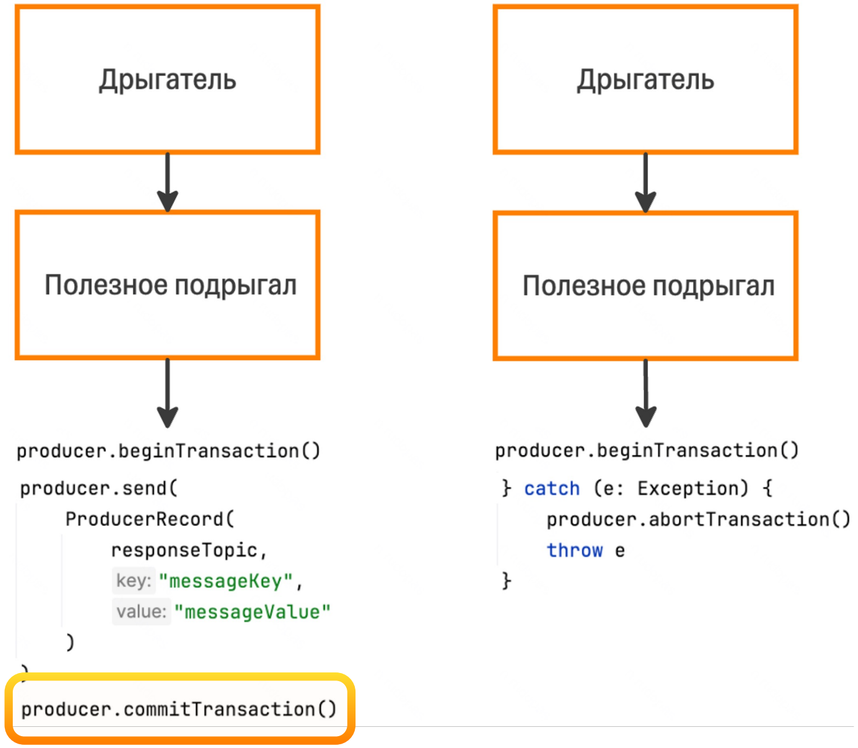

```
producer.beginTransaction()
producer.send(
    ProducerRecord(
        responseTopic,
        key: "messageKey",
        value: "messageValue"
    )
)
```



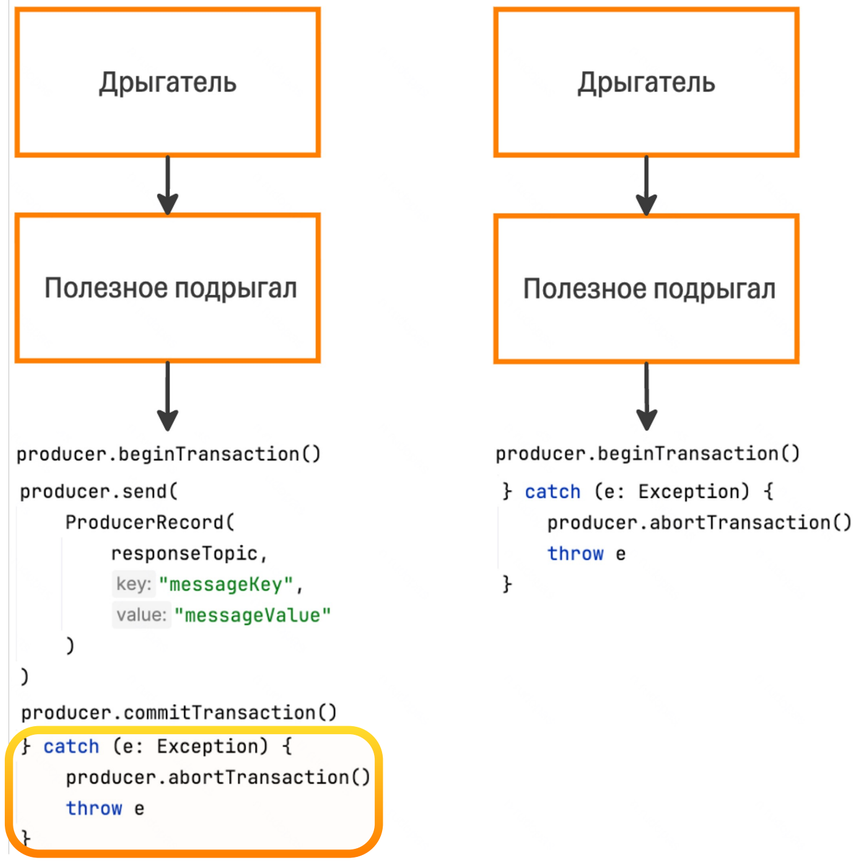
Transactional Producer очень



Transactional Producer очень-очень



Transactional Producer очень-очень потоконебезопасный



Transactional Producer очень-очень потоконебезопасный

```
producer.beginTransaction()
producer.send(
    ProducerRecord(
        responseTopic,
        key: "messageKey",
        value: "messageValue"
    )
)
producer.commitTransaction()
} catch (e: Exception) {
    producer.abortTransaction()
    throw e
}
```

```
producer.beginTransaction()
} catch (e: Exception) {
    producer.abortTransaction()
    throw e
}
```

Две поломанные транзакции по цене одного продьюсера

Ничего мы не забыли

Ответ на каждый запрос

Никогда и ни при каких обстоятельствах не теряем сообщения

Ничего мы не забыли

Ответ на каждый запрос

Никогда и ни при каких обстоятельствах не теряем сообщения

Сохранять порядок

Не меняем порядок запросов и ответов

Ничего мы не забыли

Ответ на каждый запрос

Никогда и ни при каких обстоятельствах не теряем сообщения

Сохранять порядок

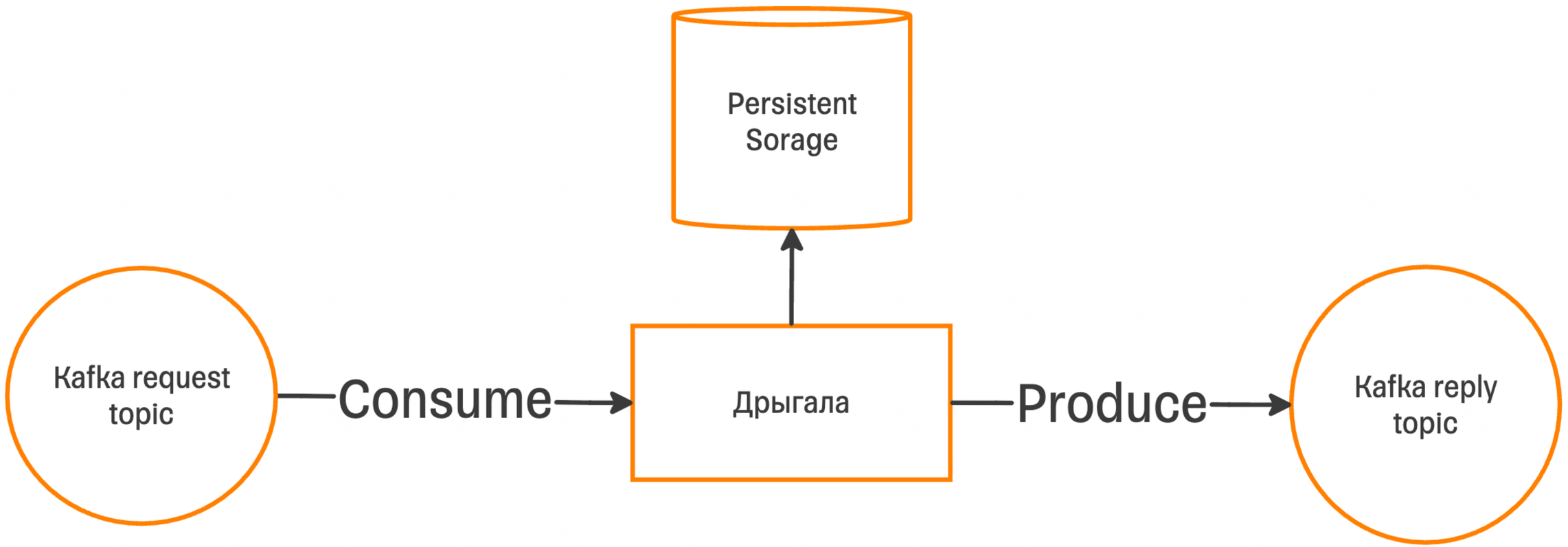
Не меняем порядок запросов и ответов

Не дублировать ответы

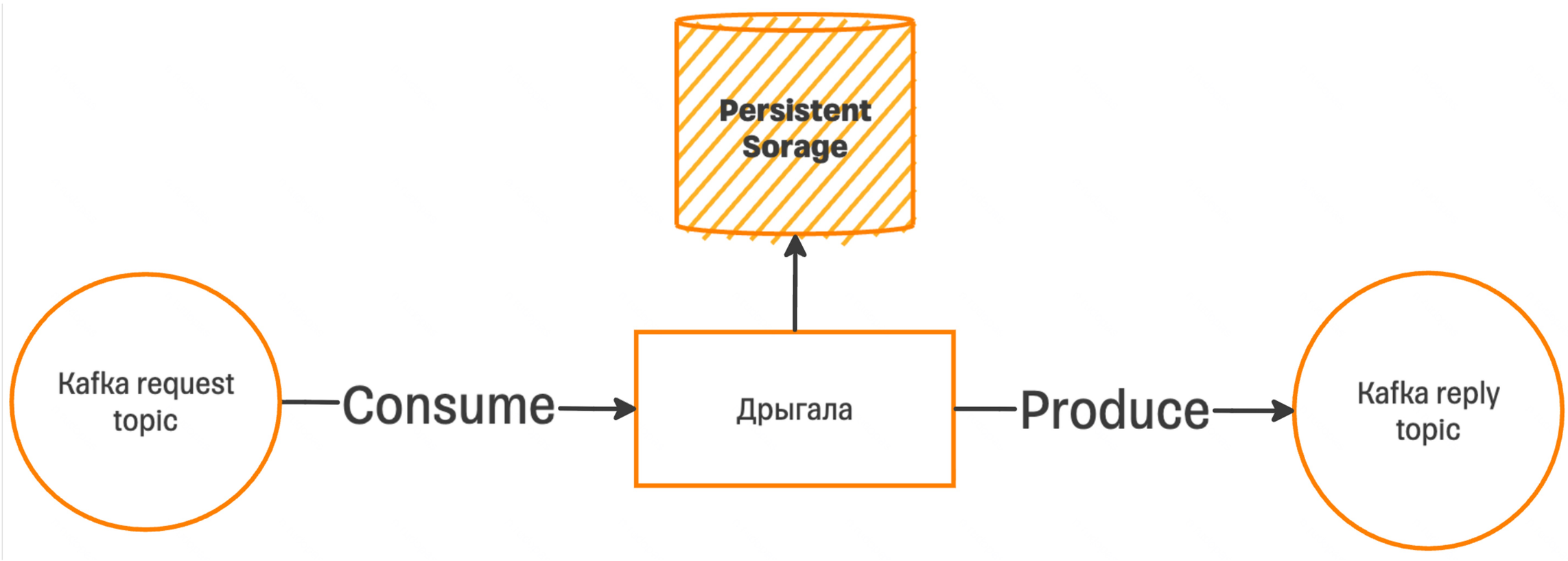
isolation.level: read_committed у consumer наших ответов и никогда

В БД хочу!

Новая салфетка



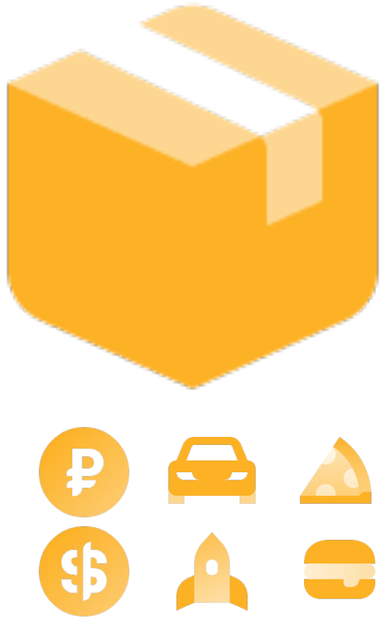
Что мы еще не видели



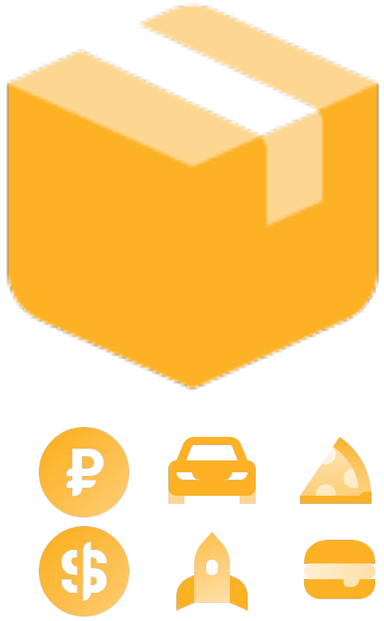
Мы не просто так дрыгалы делаем

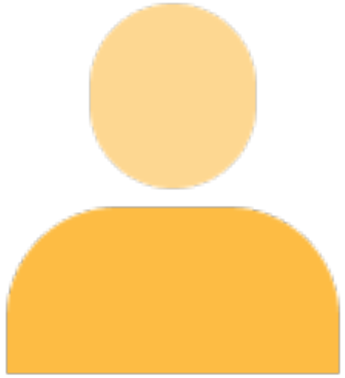
Коробка

Коробка с ништяками

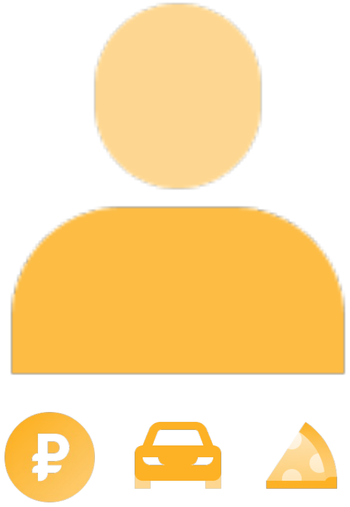


Фёдор





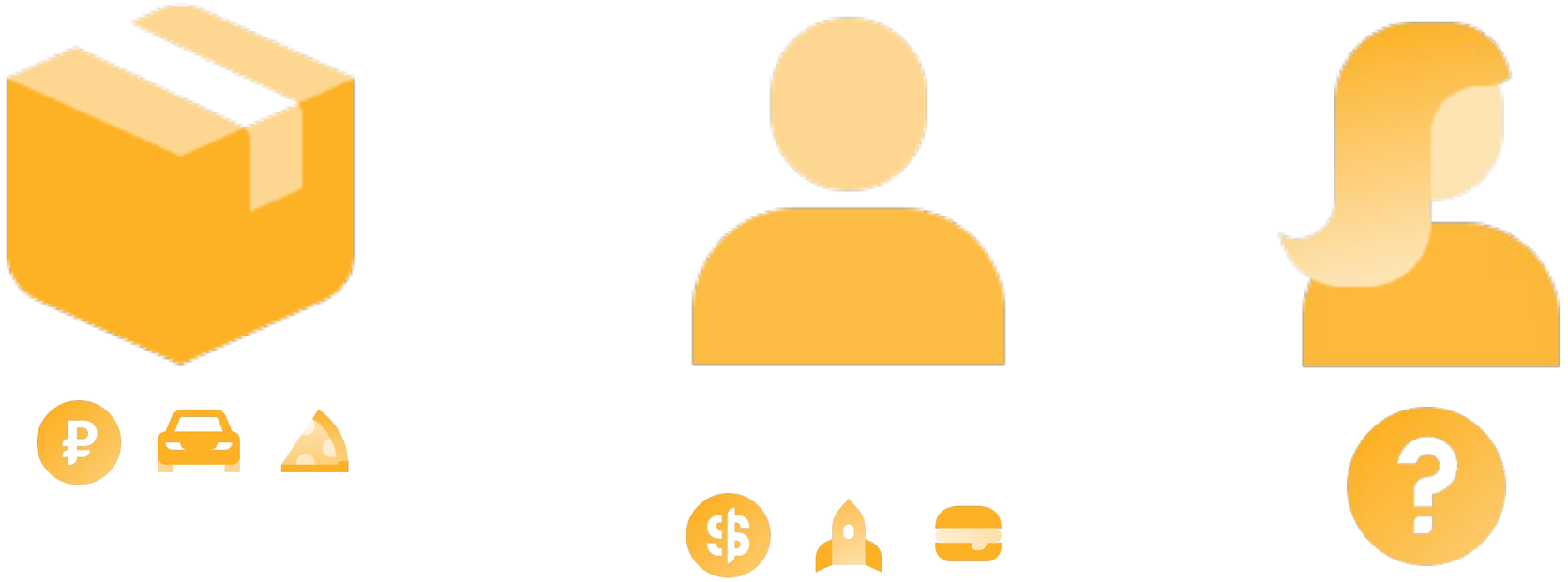
Федор с ништяками



Майор Евгения бдит

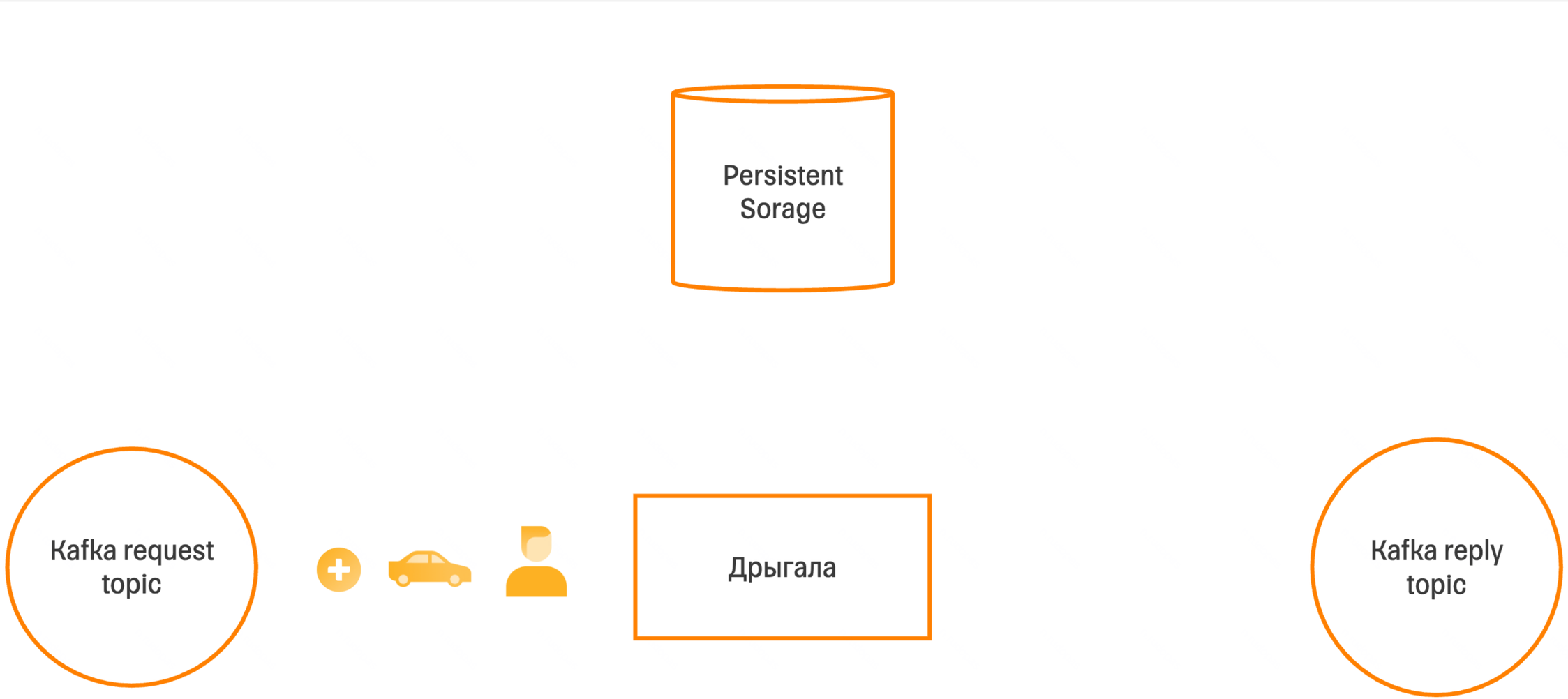
Федор избавляется от ништяков

Майор Евгения имеет вопросики

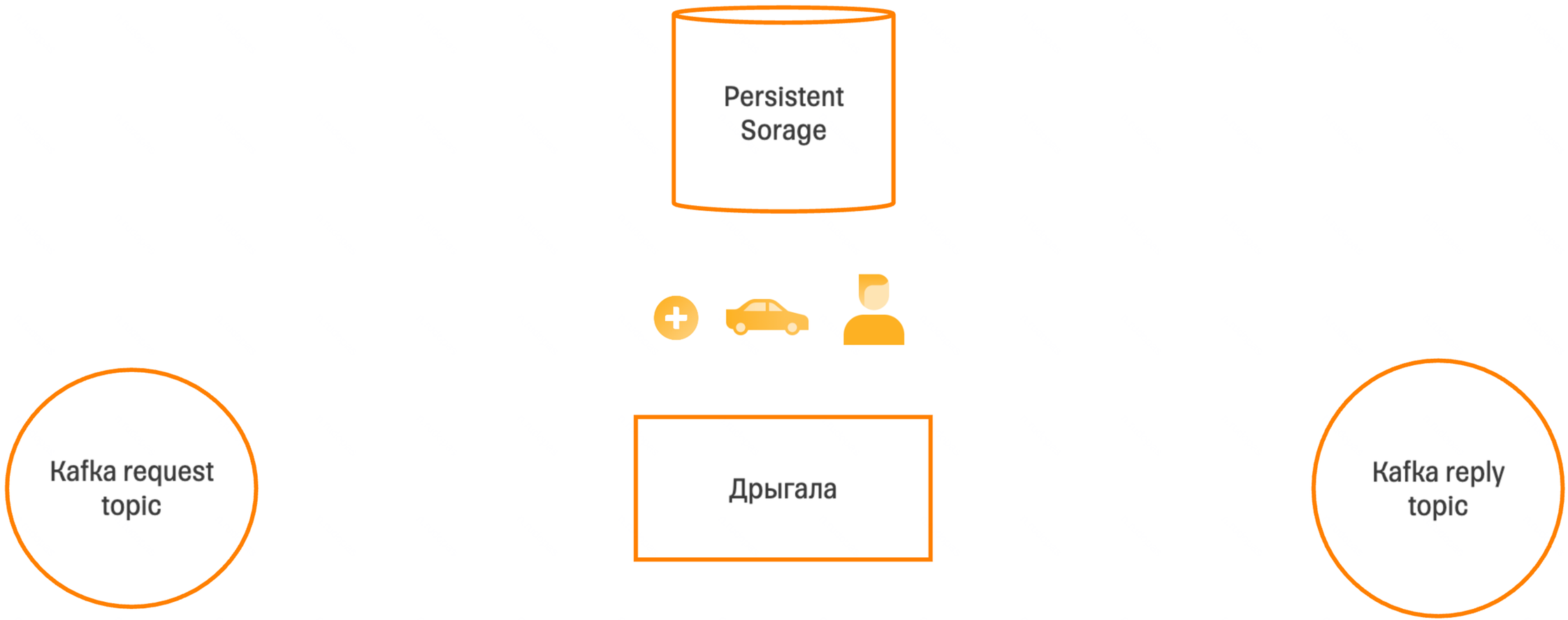


Функционал

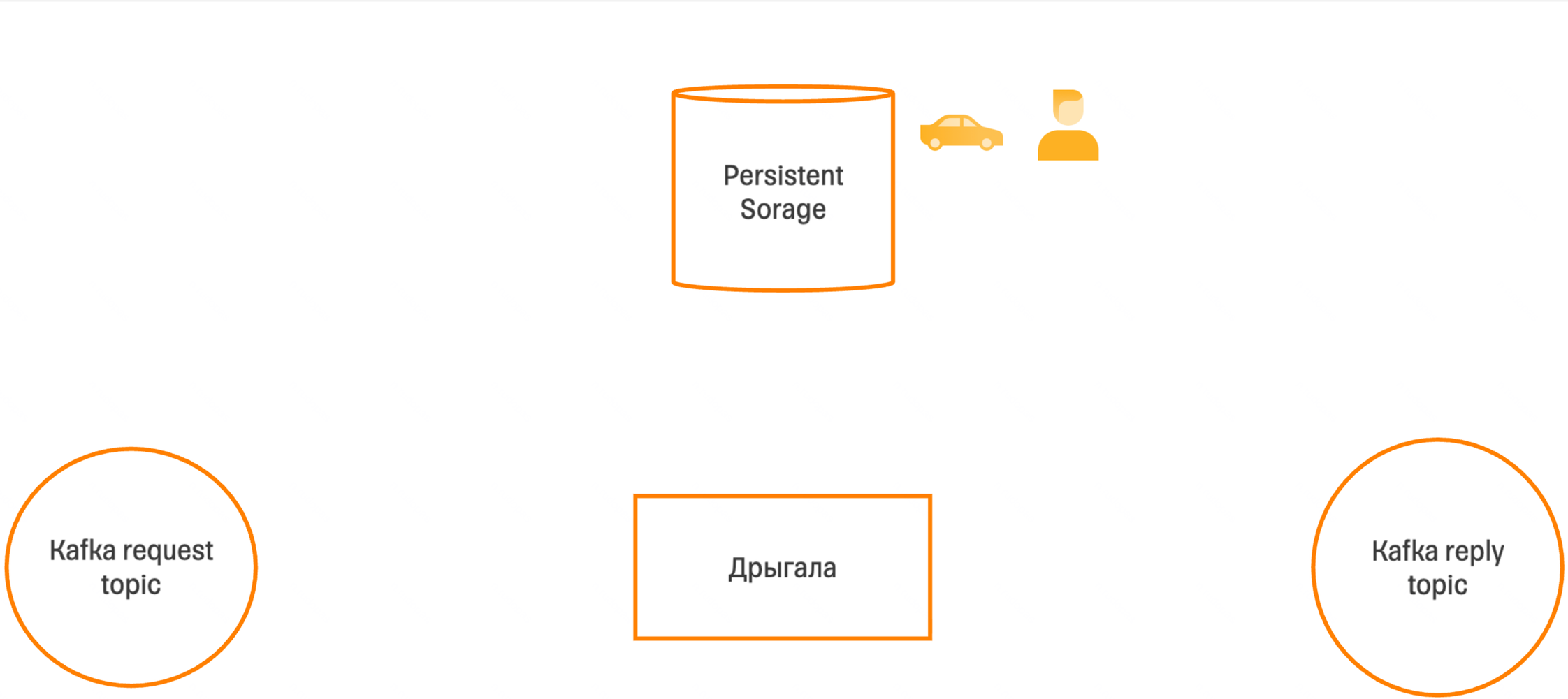
Даёшь машинку Фёдору



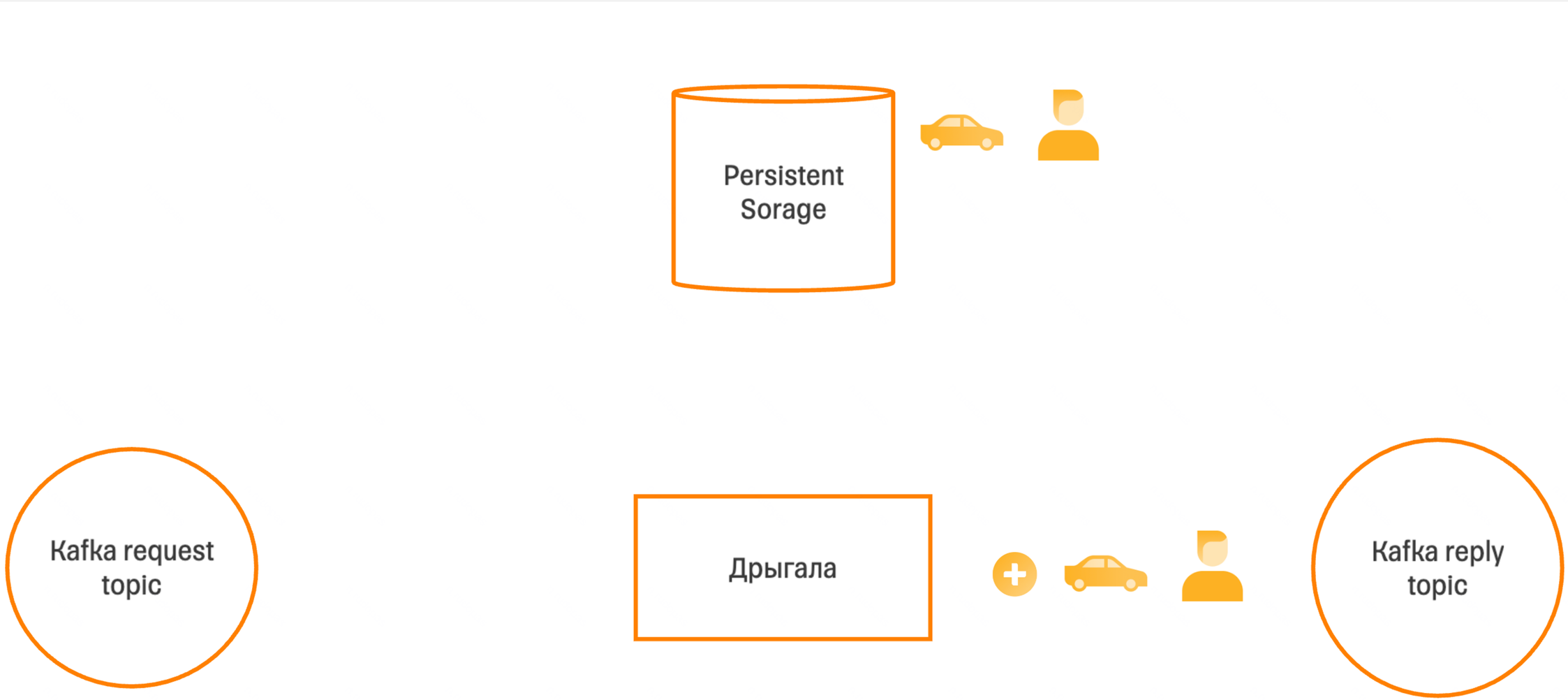
Даёшь машинку Фёдору



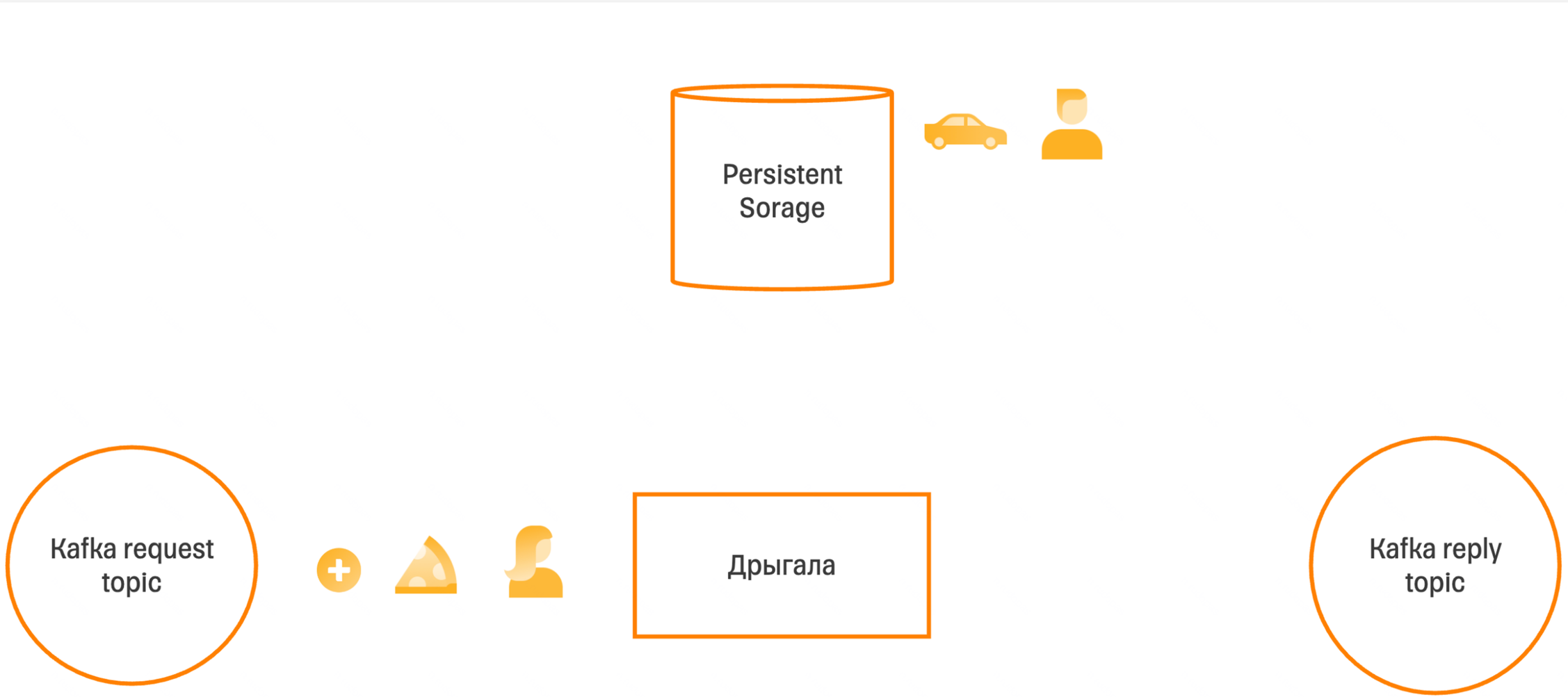
Даёшь машинку Фёдору



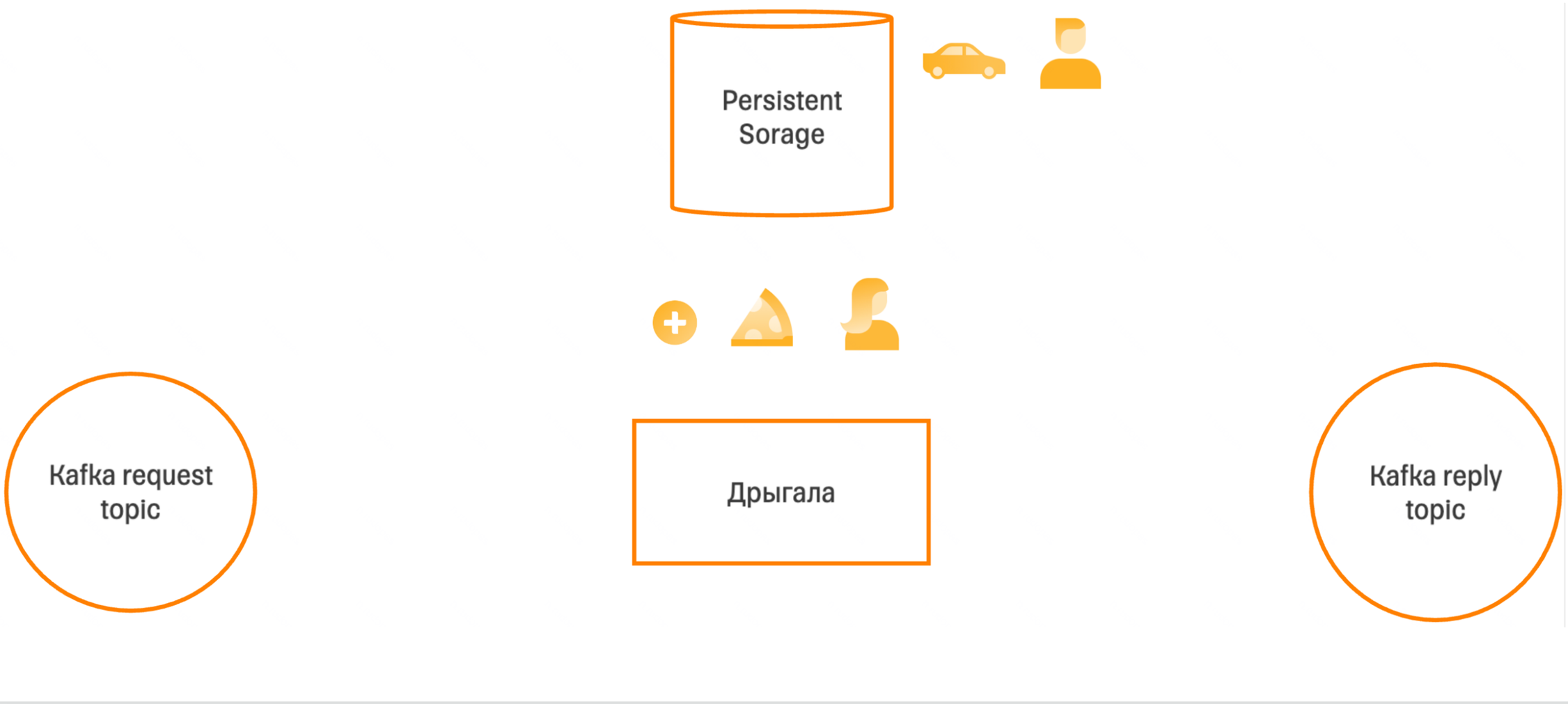
Даёшь машинку Фёдору



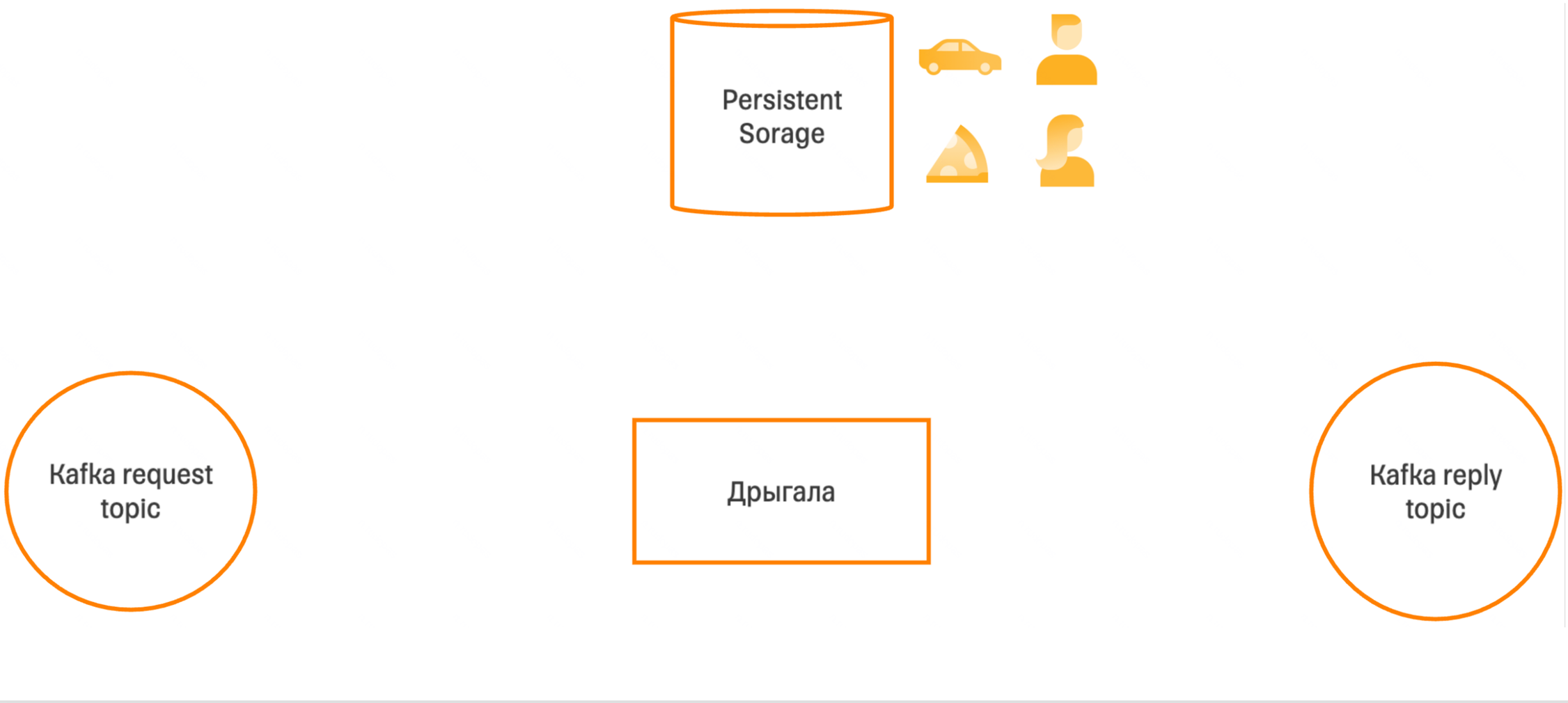
Пиццу Маргарите



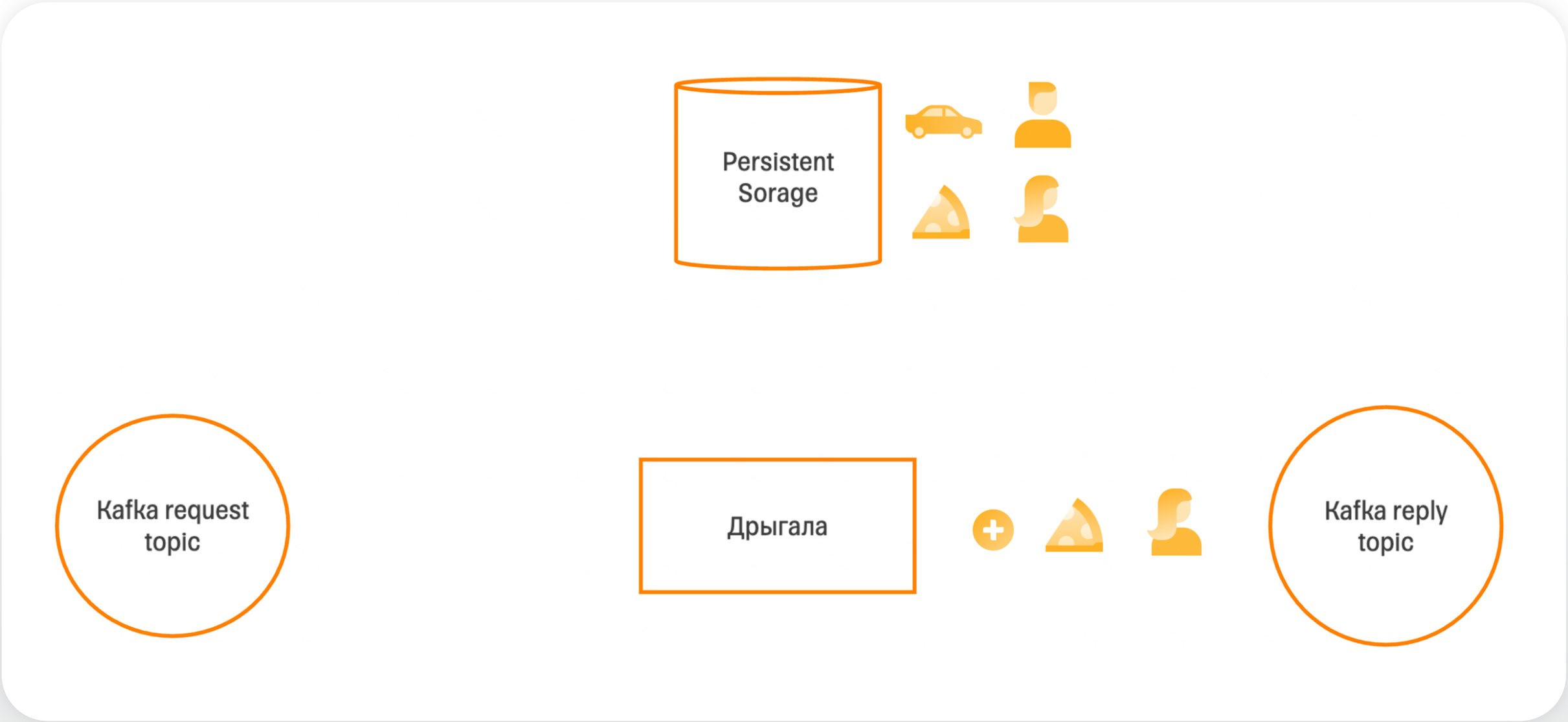
Пиццу Маргарите



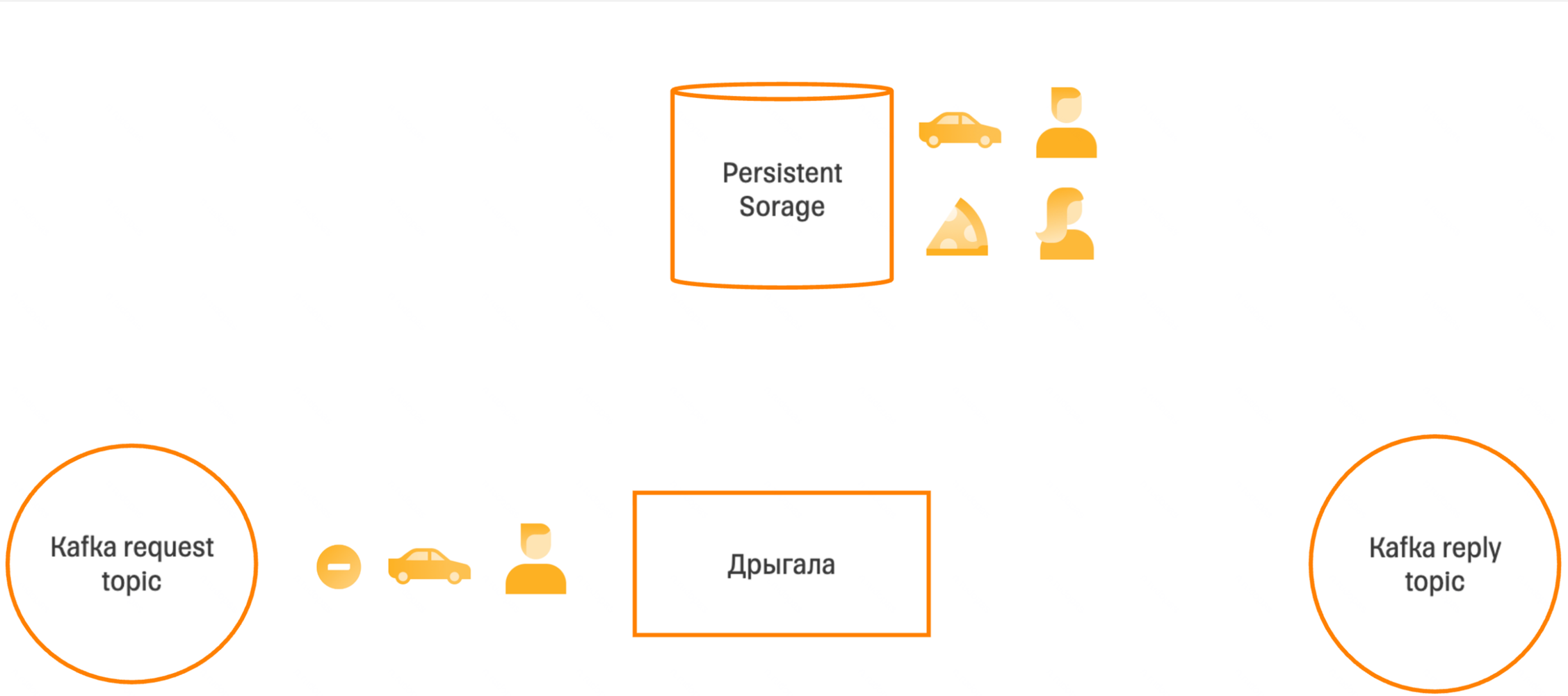
Пиццу Маргарите



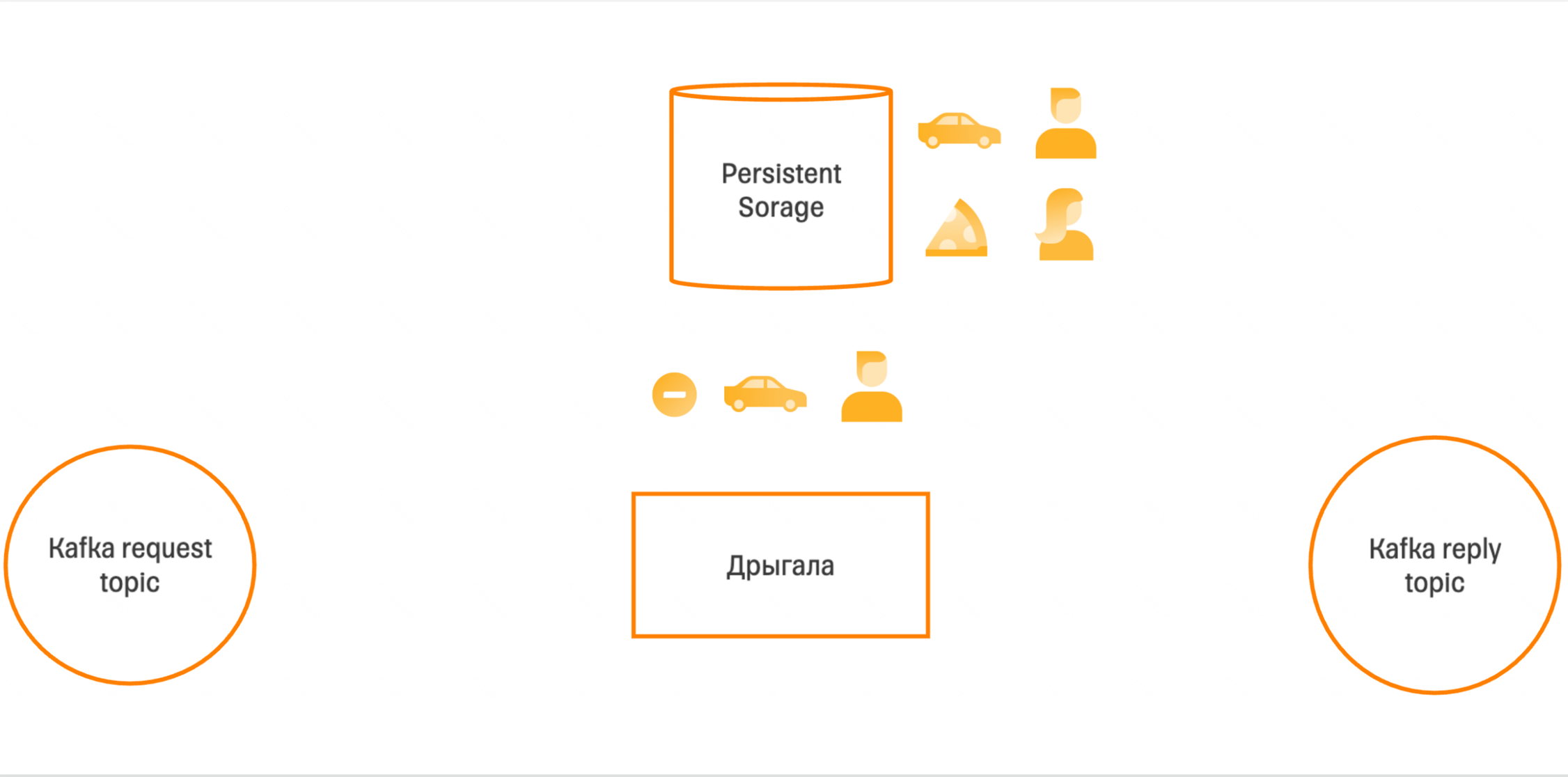
Пиццу Маргарите



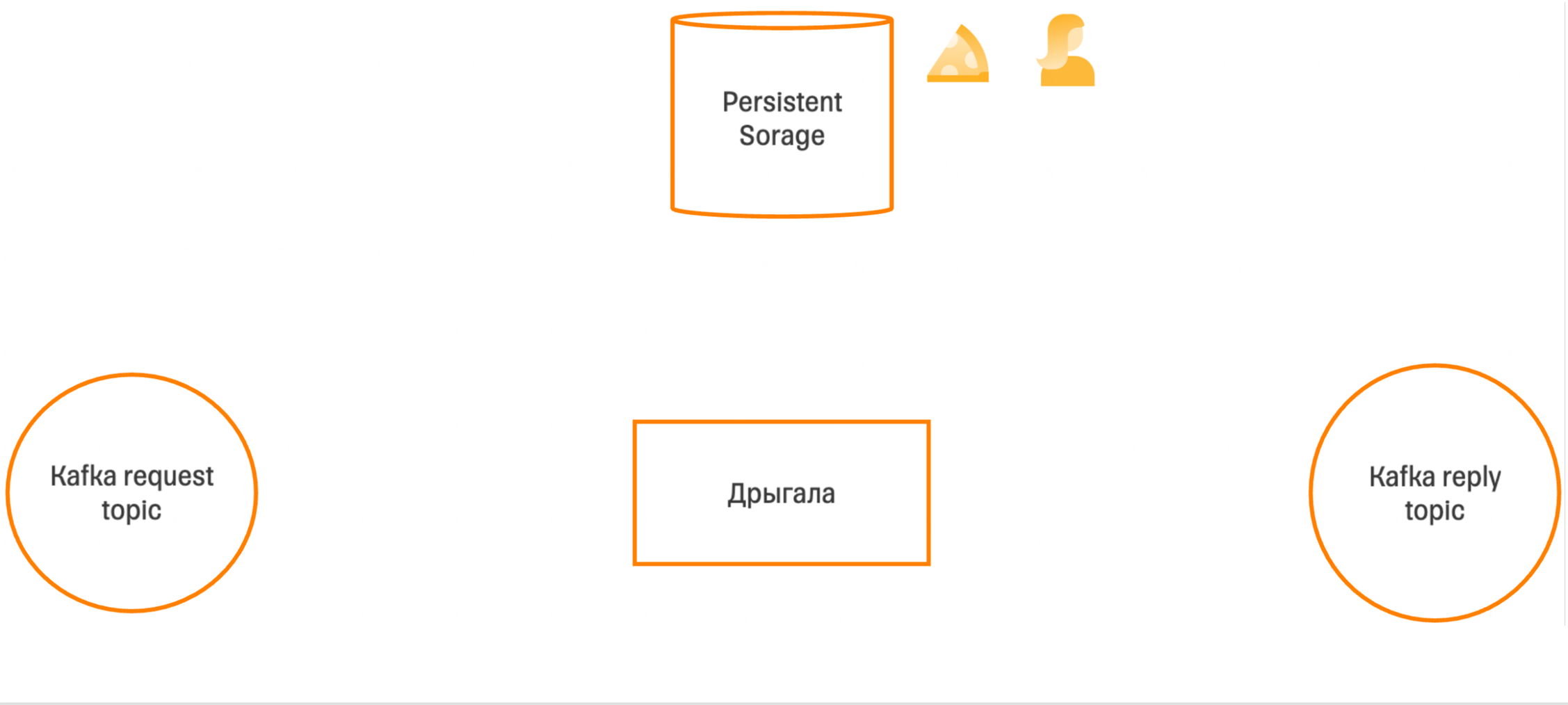
Забираем машинку у Фёдора



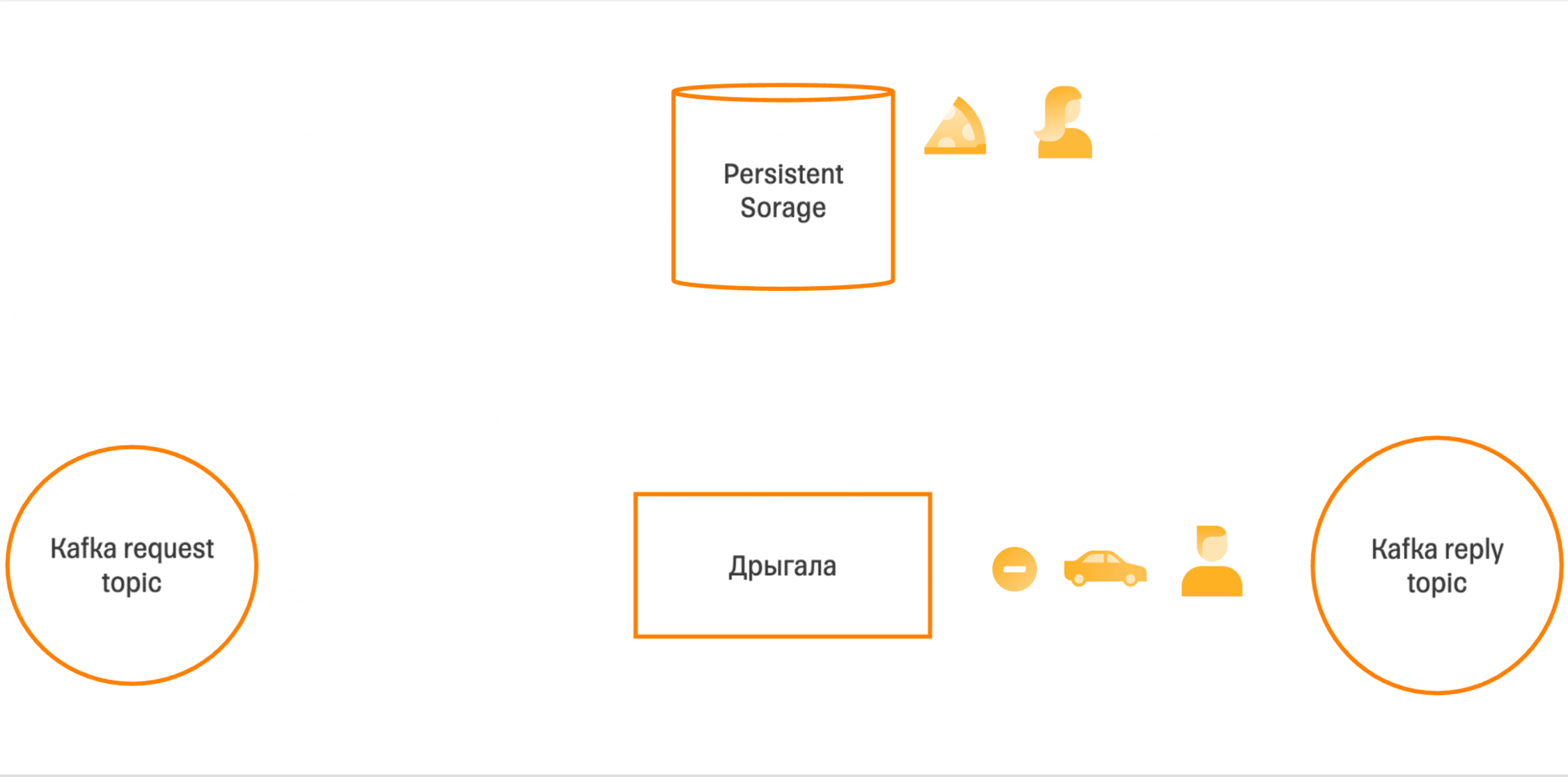
Забираем машинку у Фёдора



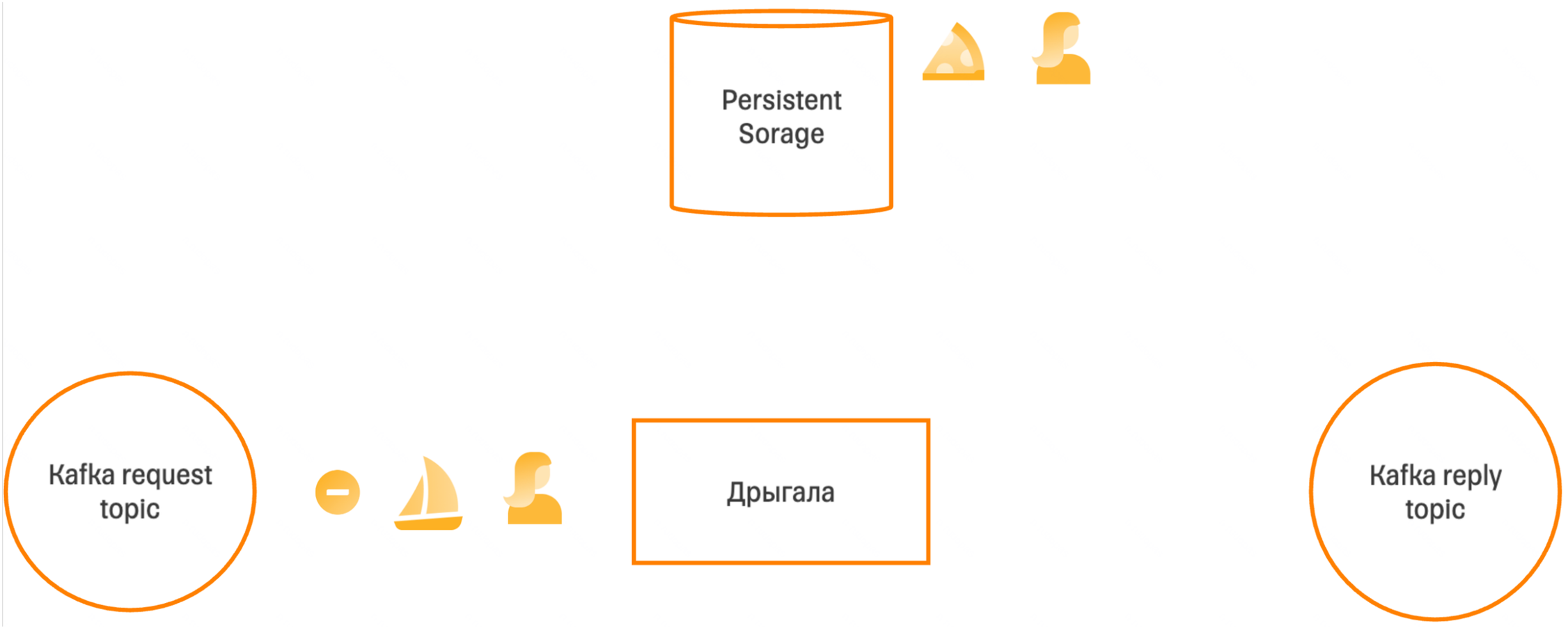
Забираем машинку у Фёдора



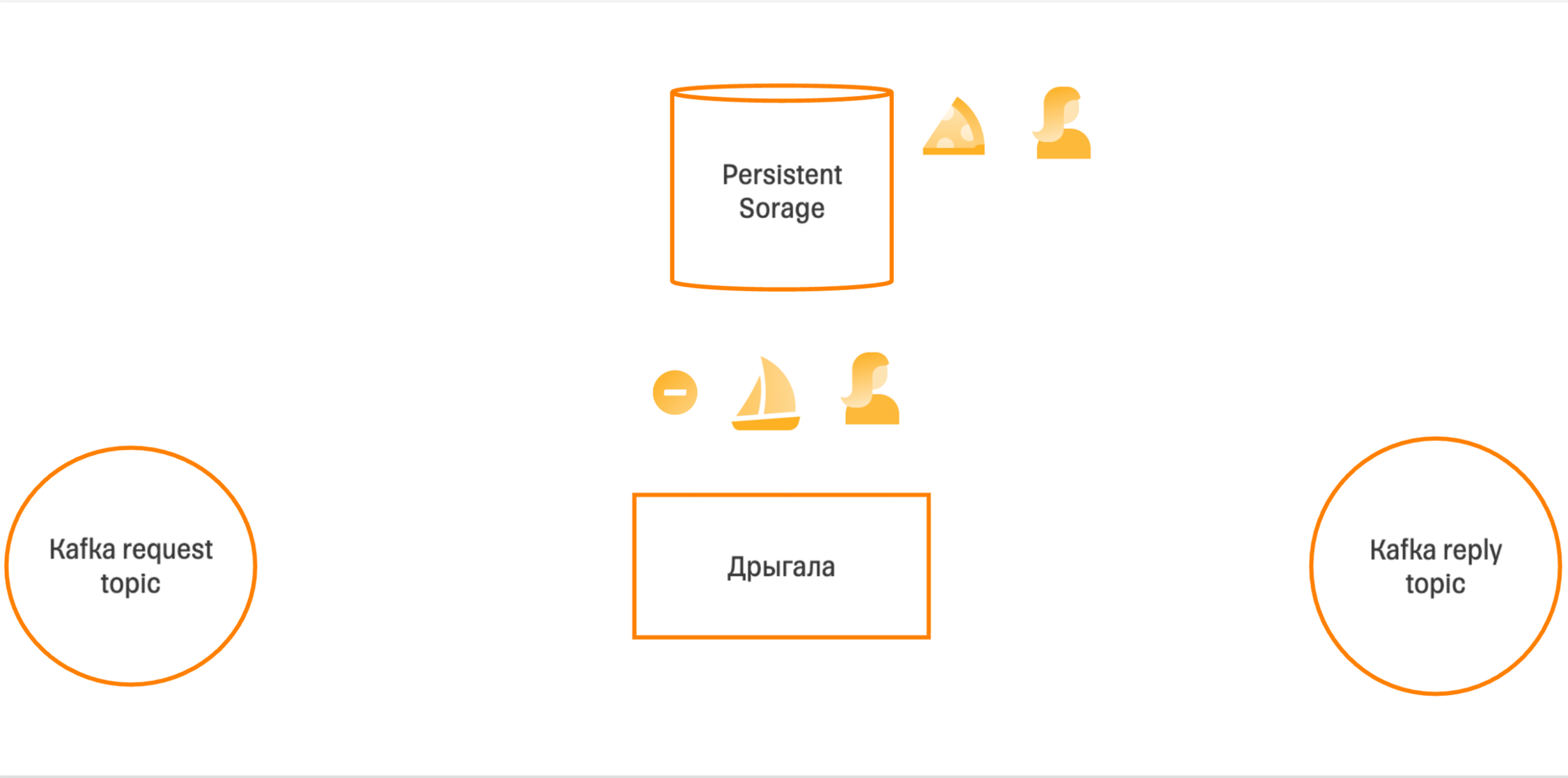
Забираем машинку у Фёдора



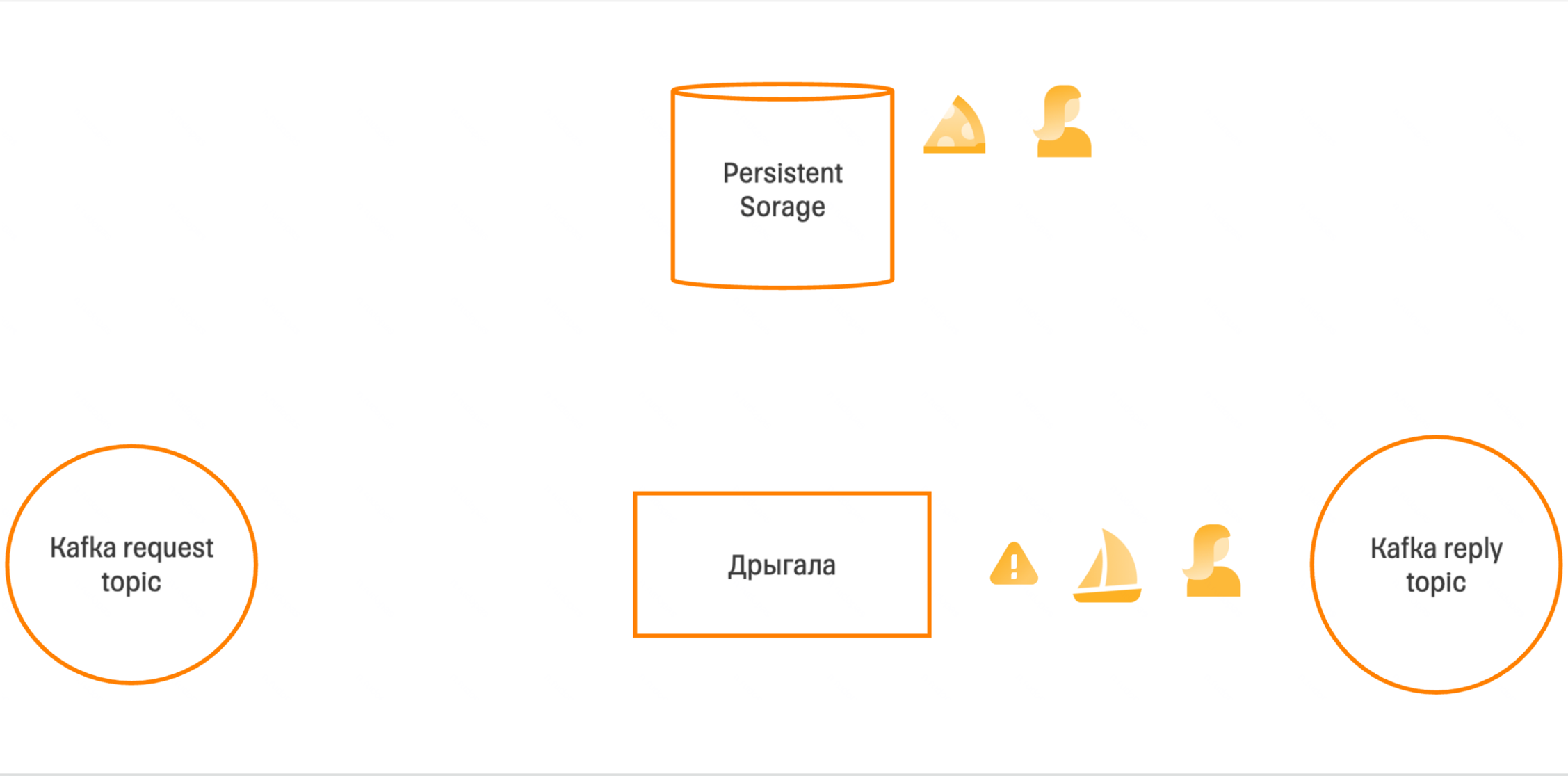
Кораблик Маргариты



Кораблик Маргариты



Кораблик Маргариты



Хотелки

Ответ на каждый запрос

Никогда и ни при каких обстоятельствах не терять сообщения

Хотелки

Ответ на каждый запрос

Никогда и ни при каких обстоятельствах не терять сообщения

Сохранять порядок

Обязательно не менять порядок запросов и ответов

Хотелки

Ответ на каждый запрос

Никогда и ни при каких обстоятельствах не терять сообщения

Сохранять порядок

Обязательно не менять порядок запросов и ответов

Не терять изменения

Должна быть возможность воссоздать актуальное состояние по нашим ответам

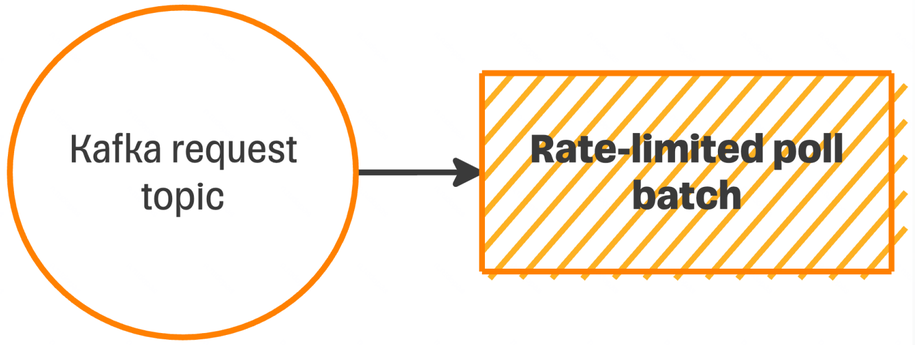
Пилим

Запрос/Ответ

```kotlin
interface NishtyakRequest {
    val humanId: String
    val requestType: String
}

data class CreateNishtyakRequest(
    override val humanId: String,
    override val requestType: String,
    val payload: String
) : NishtyakRequest

data class DeleteNishtyakRequest(
    override val humanId: String,
    override val requestType: String,
    val nishtyakType: String
) : NishtyakRequest
```

```kotlin
interface NishtyakResponse {
    val humanId: String
    val responseType: String
}

data class CreateNishtyakResponse(
    override val humanId: String,
    val payload: String
) : NishtyakResponse {
    override val responseType: String = SUCCESS
}

data class DeleteNishtyakResponse(
    override val humanId: String,
    val payload: String
) : NishtyakResponse {
    override val responseType: String = SUCCESS
}

data class ErrorNishtyakResponse(
    override val humanId: String,
    val errorMessage: String
) : NishtyakResponse {
    override val responseType: String = ERROR
}
```

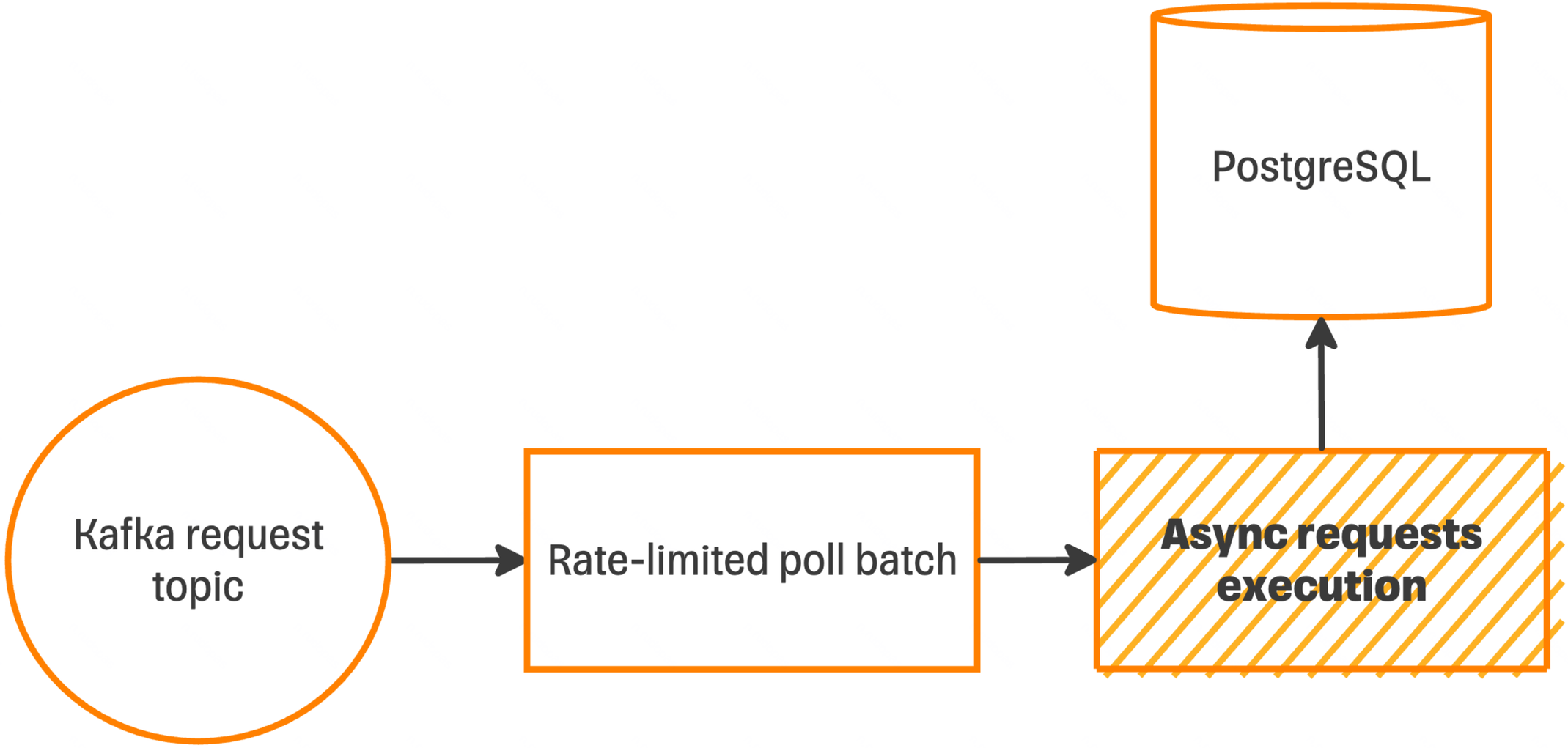
Полим с рейт-лимитом




```kotlin
fun <Key, Value> pollWithLimit(
    bucket: Bucket,
    consumer: KafkaConsumer<Key, Value>,
    pollTimeout: Duration,
    maxPollRecords: Long
): ConsumerRecords<Key, Value> {
    // Ждем свободных токенов на максимальный батч
    bucket.asBlocking().consume(maxPollRecords)
    // Полим батч
    val records = consumer.poll(pollTimeout)
    // Возвращаем неиспользованые токены обратно в бакет
    val diff = maxPollRecords - records.count()
    if (diff > 0) {
        bucket.addTokens(diff)
    }
    return records
}
```

```kotlin
fun startProcessing(
    consumer: KafkaConsumer<String, NishtyakRequest>,
    requestTopic: String,
    pollTimeout: Duration,
    maxPollRecords: Long, //max.poll.records
    producer: KafkaProducer<String, NishtyakResponse>,
    responseTopic: String,
    nishtyakService: NishtyakRequestService
) {
    val bucket = getBucket(maxPollRecords)
    consumer.subscribe(listOf(requestTopic))
    while (true) {
        val requests = pollWithLimit(
            bucket,
            consumer,
            pollTimeout,
            maxPollRecords
        )
```

Обрабатываем запросы

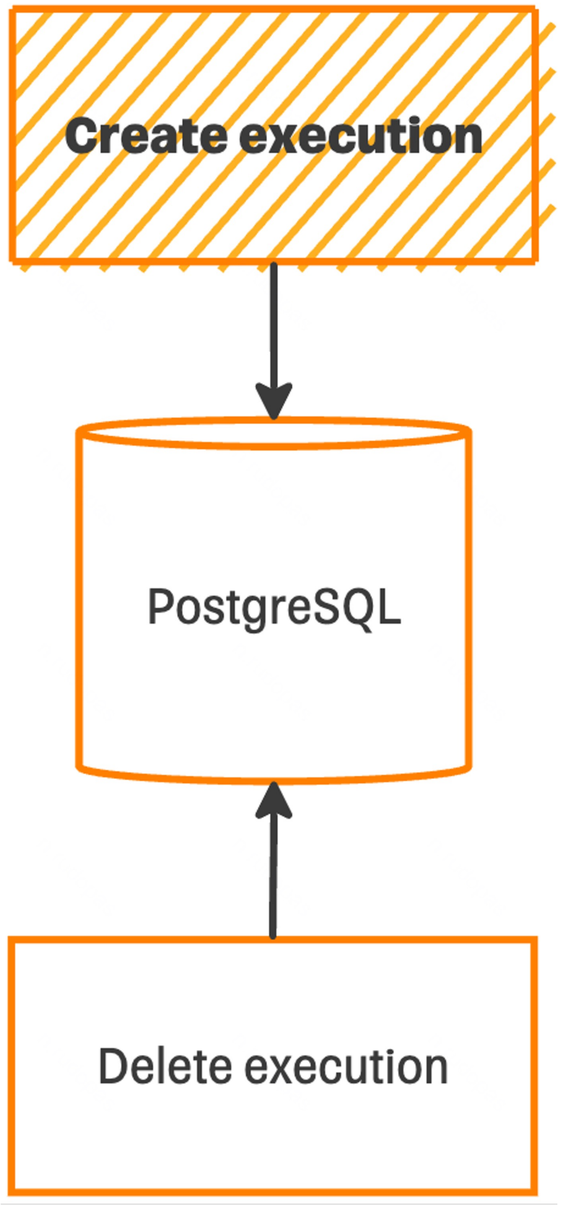


Начнем снизу

Nishtyak		
PK	human_id	TEXT
PK	nishtyak_type	TEXT
	nishtyak_payload	JSONB
	created_at	TIMESTAMP

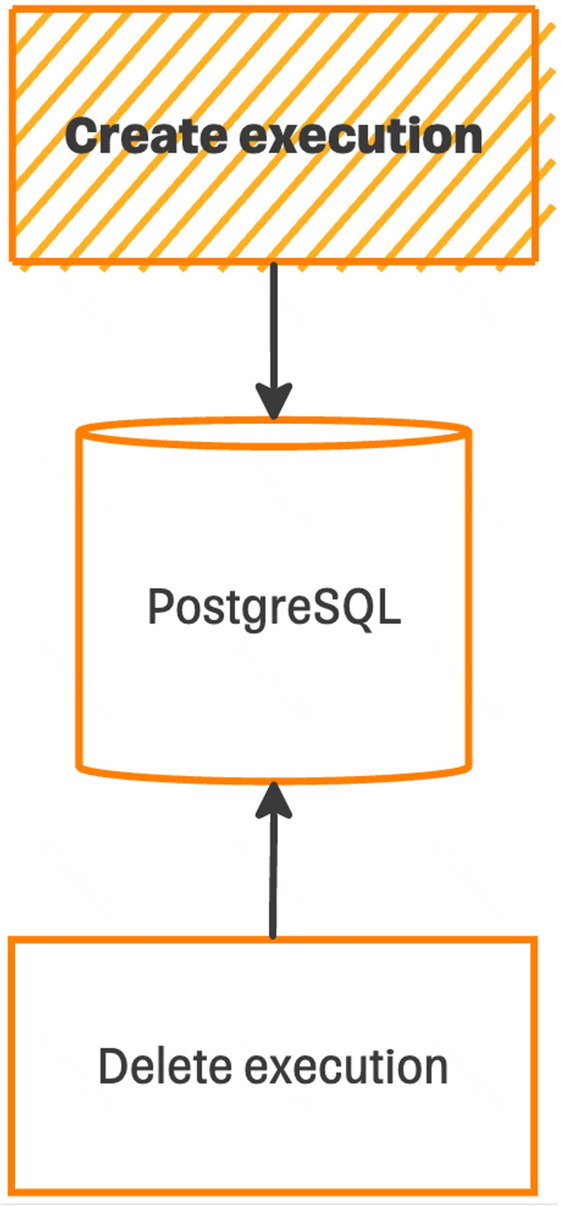
```
24  data class NishtyakEntity(
25      val humanId: String,
26      val type: String,
27      val payload: String,
28      val createdAt: Instant
29  )
30
31  @Repository
32  interface NishtyakRepository {
33      fun create(nishtyak: NishtyakEntity)
34
35      fun delete(
36          humanId: String,
37          type: String
38      ): NishtyakEntity?
39  }
```

Обработка запроса на создание ништяка



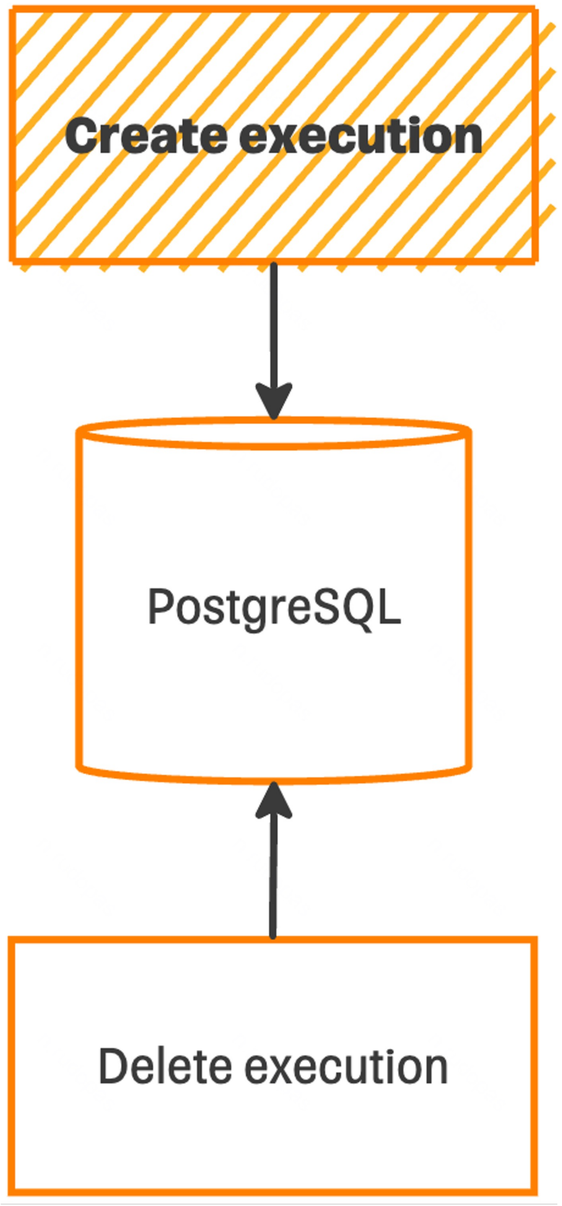

```kotlin
interface NishtyakRequestProcessor {
    val requestType: String
    fun process(request: NishtyakRequest): NishtyakResponse
}

@Service
class CreateNishtyakRequestProcessor(
    private val nishtyakRepository: NishtyakRepository
) : NishtyakRequestProcessor {
    override val requestType: String = "create"

    override fun process(
        request: NishtyakRequest
    ): NishtyakResponse {
        val createRequest = request
            as? CreateNishtyakRequest
            ?: return getInvalidRequestResponse(request.humanId)
        nishtyakRepository.create(
            NishtyakEntity(
                humanId = createRequest.humanId,
                type = createRequest.requestType,
                payload = createRequest.payload,
                createdAt = Instant.now()
            )
        )
        return CreateNishtyakResponse(
            humanId = createRequest.humanId,
            payload = createRequest.payload
        )
    }
```

Обработка запроса на создание ништяка



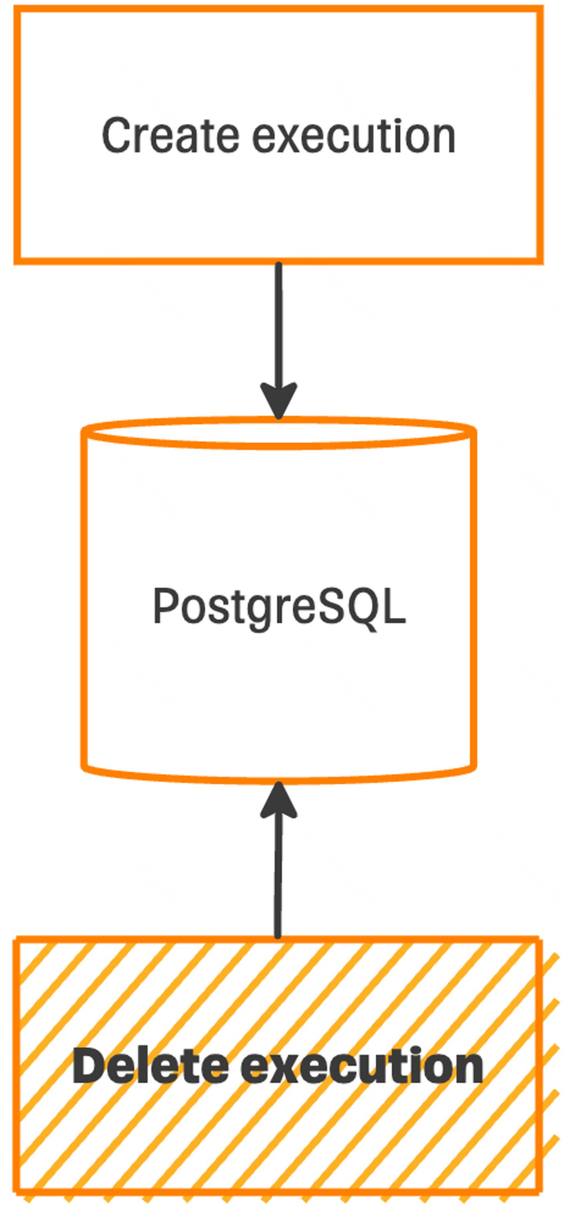

```kotlin
interface NishtyakRequestProcessor {
    val requestType: String
    fun process(request: NishtyakRequest): NishtyakResponse
}

@Service
class CreateNishtyakRequestProcessor(
    private val nishtyakRepository: NishtyakRepository
) : NishtyakRequestProcessor {
    override val requestType: String = "create"

    override fun process(
        request: NishtyakRequest
    ): NishtyakResponse {
        val createRequest = request
            as? CreateNishtyakRequest
            ?: return getInvalidRequestResponse(request.humanId)
        nishtyakRepository.create(
            NishtyakEntity(
                humanId = createRequest.humanId,
                type = createRequest.requestType,
                payload = createRequest.payload,
                createdAt = Instant.now()
            )
        )
        return CreateNishtyakResponse(
            humanId = createRequest.humanId,
            payload = createRequest.payload
        )
    }
```

Обработка запроса на создание ништяка



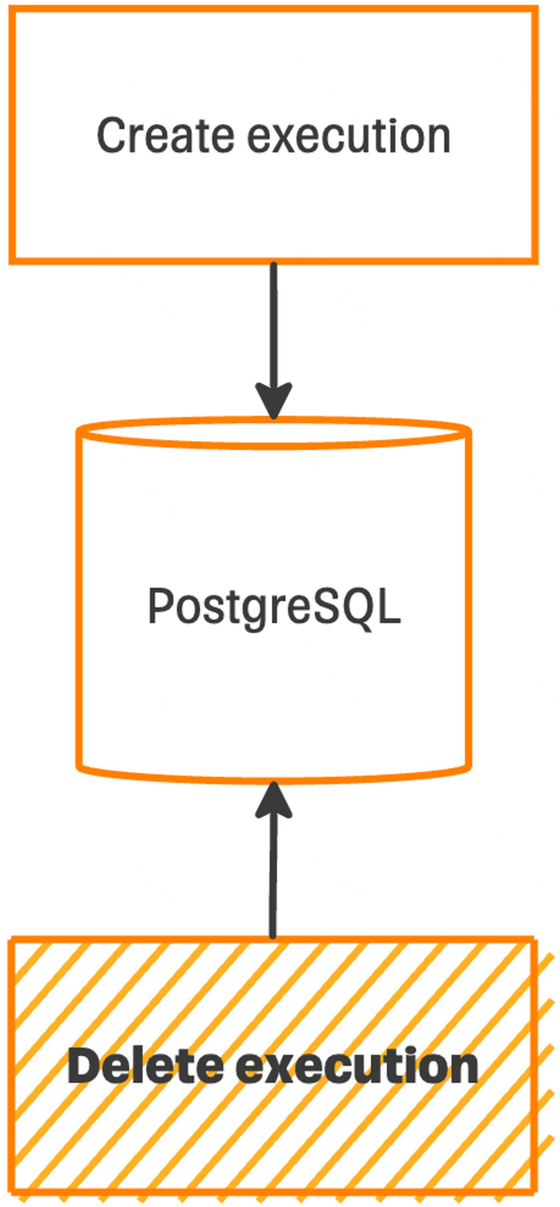

```
84  interface NishtyakRequestProcessor {
85      val requestType: String
86      fun process(request: NishtyakRequest): NishtyakResponse
87  }
88
89  @Service
90  class CreateNishtyakRequestProcessor(
91      private val nishtyakRepository: NishtyakRepository
92  ) : NishtyakRequestProcessor {
93      override val requestType: String = "create"
94
95      override fun process(
96          request: NishtyakRequest
97      ): NishtyakResponse {
98          val createRequest = request
99              as? CreateNishtyakRequest
100             ?: return getInvalidRequestResponse(request.humanId)
101         nishtyakRepository.create(
102             NishtyakEntity(
103                 humanId = createRequest.humanId,
104                 type = createRequest.requestType,
105                 payload = createRequest.payload,
106                 createdAt = Instant.now()
107             )
108         )
109         return CreateNishtyakResponse(
110             humanId = createRequest.humanId,
111             payload = createRequest.payload
112         )
113     }
```

Обработка конкретных запросов



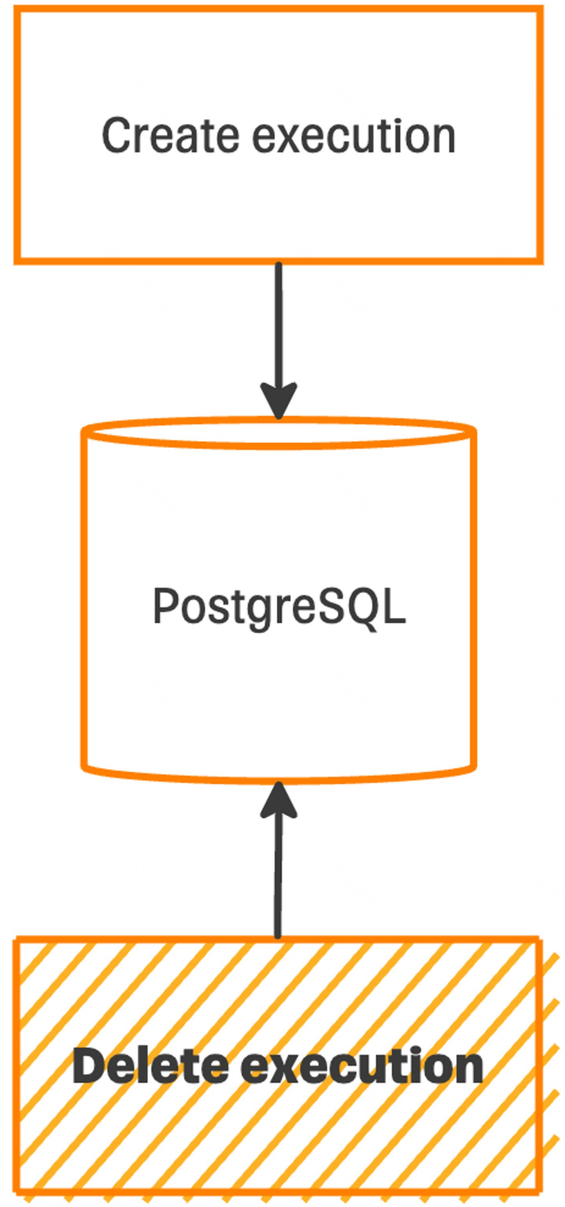

```kotlin
@Service
class DeleteNishtyakRequestProcessor(
    private val nishtyakRepository: NishtyakRepository
) : NishtyakRequestProcessor {
    override val requestType: String = "delete"

    override fun process(
        request: NishtyakRequest
    ): NishtyakResponse {
        val deleteRequest: DeleteNishtyakRequest = request
            as? DeleteNishtyakRequest
            ?: return getInvalidRequestResponse(request.humanId)

        val deleted = nishtyakRepository.delete(
            humanId = deleteRequest.humanId,
            type = deleteRequest.requestType
        )
        return if (deleted == null) {
            getNotFoundResponse(
                humanId = deleteRequest.humanId,
                nishtyakType = deleteRequest.nishtyakType
            )
        } else {
            DeleteNishtyakResponse(
                humanId = deleteRequest.humanId,
                payload = deleted.payload
            )
        }
    }
```

Обработка конкретных запросов



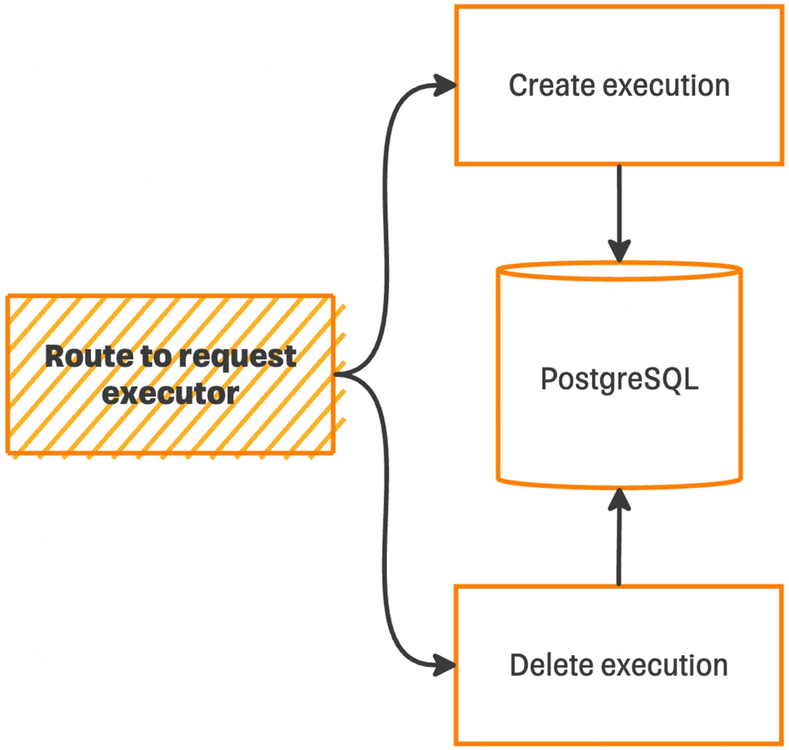

```
122  @Service
123  class DeleteNishtyakRequestProcessor(
124      private val nishtyakRepository: NishtyakRepository
125  ) : NishtyakRequestProcessor {
126      override val requestType: String = "delete"
127
128      override fun process(
129          request: NishtyakRequest
130      ): NishtyakResponse {
131          val deleteRequest: DeleteNishtyakRequest = request
132              as? DeleteNishtyakRequest
133              ?: return getInvalidRequestResponse(request.humanId)
134
135          val deleted = nishtyakRepository.delete(
136              humanId = deleteRequest.humanId,
137              type = deleteRequest.requestType
138          )
139          return if (deleted == null) {
140              getNotFoundResponse(
141                  humanId = deleteRequest.humanId,
142                  nishtyakType = deleteRequest.nishtyakType
143              )
144          } else {
145              DeleteNishtyakResponse(
146                  humanId = deleteRequest.humanId,
147                  payload = deleted.payload
148              )
149          }
150      }
```

Обработка конкретных запросов



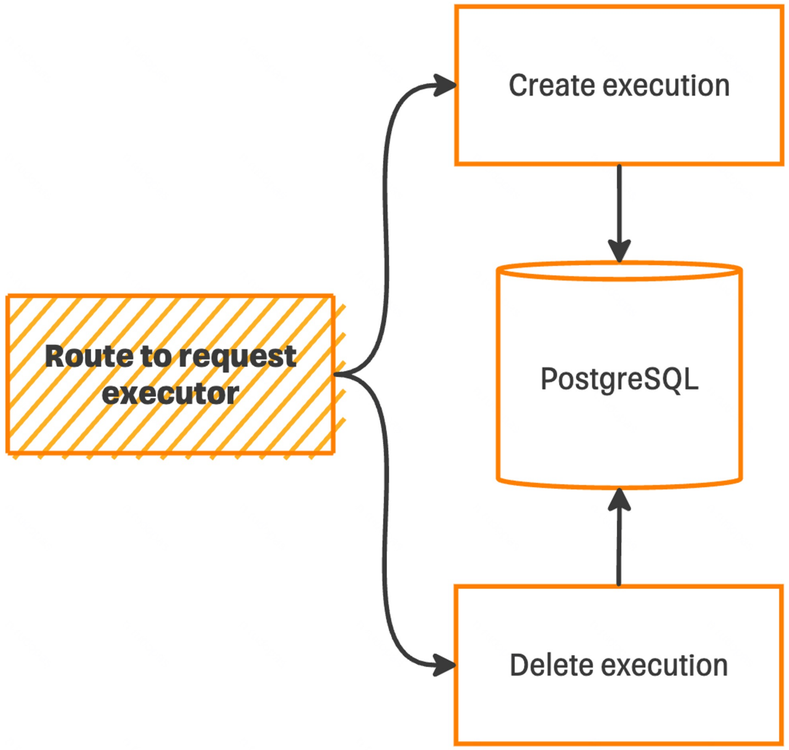

```
122  @Service
123  class DeleteNishtyakRequestProcessor(
124      private val nishtyakRepository: NishtyakRepository
125  ) : NishtyakRequestProcessor {
126      override val requestType: String = "delete"
127
128      override fun process(
129          request: NishtyakRequest
130      ): NishtyakResponse {
131          val deleteRequest: DeleteNishtyakRequest = request
132              as? DeleteNishtyakRequest
133              ?: return getInvalidRequestResponse(request.humanId)
134
135          val deleted = nishtyakRepository.delete(
136              humanId = deleteRequest.humanId,
137              type = deleteRequest.requestType
138          )
139          return if (deleted == null) {
140              getNotFoundResponse(
141                  humanId = deleteRequest.humanId,
142                  nishtyakType = deleteRequest.nishtyakType
143              )
144          } else {
145              DeleteNishtyakResponse(
146                  humanId = deleteRequest.humanId,
147                  payload = deleted.payload
148              )
149          }
150      }
```

Выбираем как обрабатывать запрос



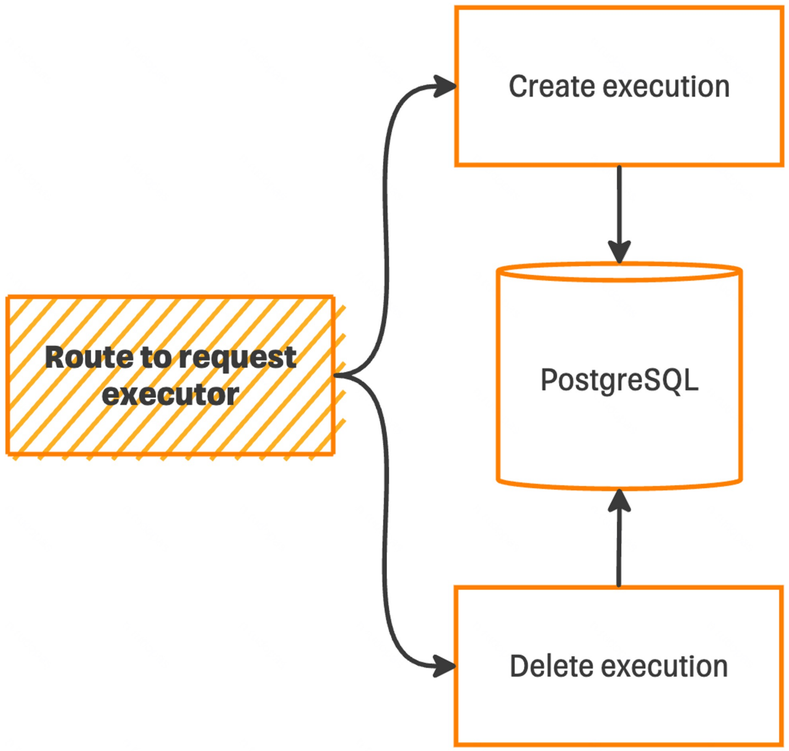

```
167  @Service
168  class NishtyakRequestRouter(
169      processors: List<NishtyakRequestProcessor>
170  ) {
171      private val processors = processors.associateBy { it.requestType }
172
173      fun route(requests: Iterable<NishtyakRequest>): List<NishtyakResponse> =
174          requests.map(this::route)
175
176      private fun route(request: NishtyakRequest): NishtyakResponse {
177          val processor = processors[request.requestType]
178              ?: return ErrorNishtyakResponse(
179                  humanId = request.humanId,
180                  errorMessage = "Unprocessable request type ${request.requestType}"
181              )
182
183          return try {
184              processor.process(request)
185          } catch (e: Exception) {
186              ErrorNishtyakResponse(
187                  humanId = request.humanId,
188                  errorMessage = "Something went wrong: ${e.message}"
189              )
190          }
191      }
192  }
```

Выбираем как обрабатывать запрос



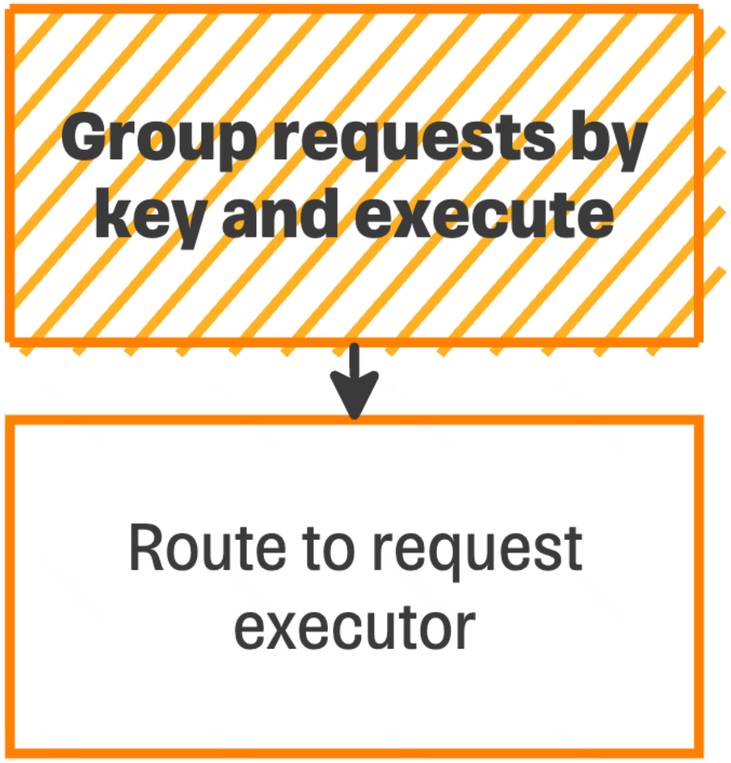

```kotlin
@Service
class NishtyakRequestRouter(
    processors: List<NishtyakRequestProcessor>
) {
    private val processors = processors.associateBy { it.requestType }

    fun route(requests: Iterable<NishtyakRequest>): List<NishtyakResponse> =
        requests.map(this::route)

    private fun route(request: NishtyakRequest): NishtyakResponse {
        val processor = processors[request.requestType]
            ?: return ErrorNishtyakResponse(
                humanId = request.humanId,
                errorMessage = "Unprocessable request type ${request.requestType}"
            )

        return try {
            processor.process(request)
        } catch (e: Exception) {
            ErrorNishtyakResponse(
                humanId = request.humanId,
                errorMessage = "Something went wrong: ${e.message}"
            )
        }
    }
}
```

Выбираем как обрабатывать запрос



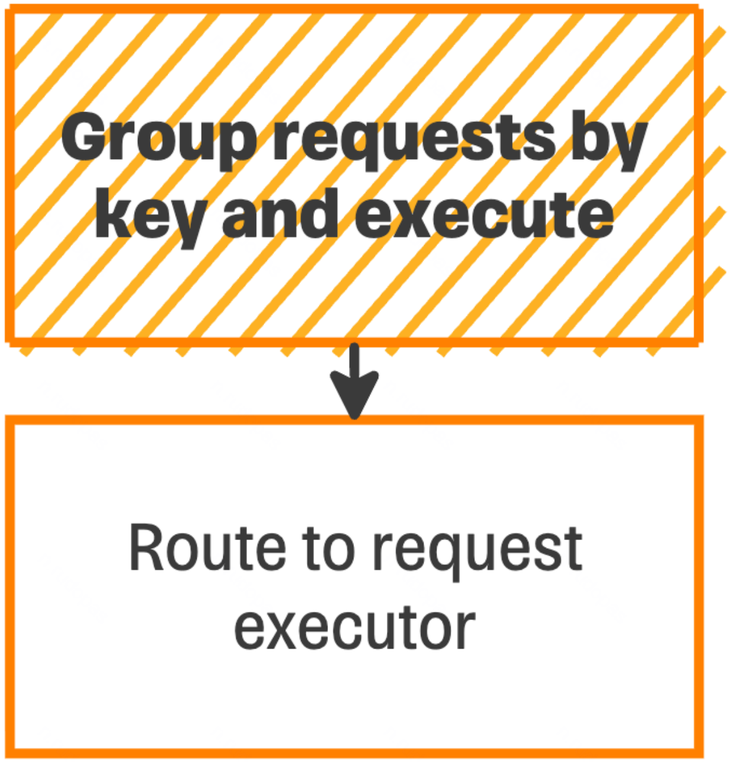

```kotlin
@Service
class NishtyakRequestRouter(
    processors: List<NishtyakRequestProcessor>
) {
    private val processors = processors.associateBy { it.requestType }

    fun route(requests: Iterable<NishtyakRequest>): List<NishtyakResponse> =
        requests.map(this::route)

    private fun route(request: NishtyakRequest): NishtyakResponse {
        val processor = processors[request.requestType]
            ?: return ErrorNishtyakResponse(
                humanId = request.humanId,
                errorMessage = "Unprocessable request type ${request.requestType}"
            )

        return try {
            processor.process(request)
        } catch (e: Exception) {
            ErrorNishtyakResponse(
                humanId = request.humanId,
                errorMessage = "Something went wrong: ${e.message}"
            )
        }
    }
}
```

Параллелим обработку



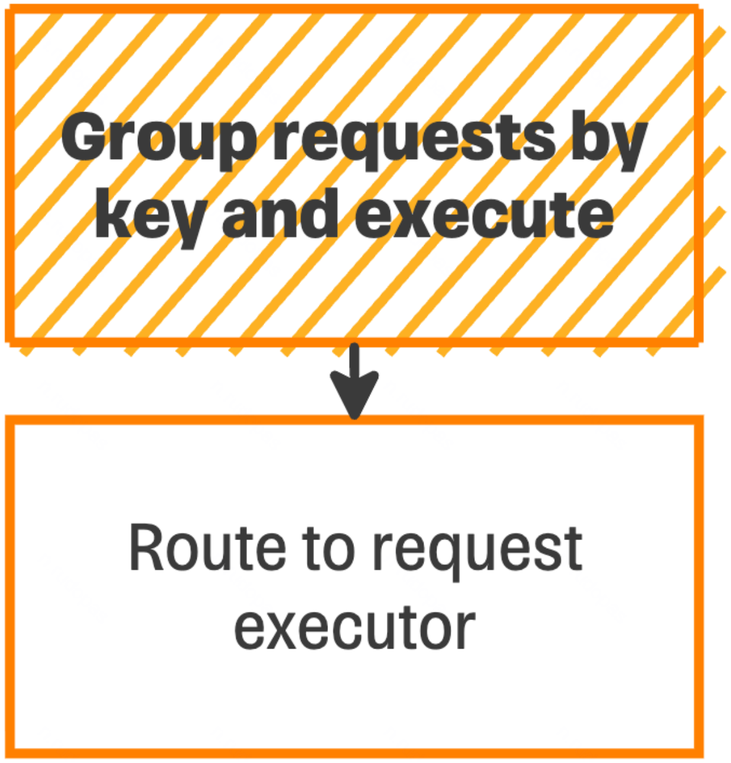

```
192  @Service
193  class NishtyakRequestService(
194      private val nishtyakRequestRouter: NishtyakRequestRouter,
195      private val batchDispatcher: CoroutineDispatcher,
196  ) {
197      fun process(
198          requests: Iterable<Pair<String, NishtyakRequest>>
199      ): List<Pair<String, NishtyakResponse>> =
200          runBlocking(context = batchDispatcher) {
201              requests.groupBy({ it.first }, { it.second })
202                  .map { (key, requests) ->
203                      async {
204                          nishtyakRequestRouter.route(requests)
205                              .map { key to it }
206                      }
207                  }
208                  .awaitAll()
209                  .flatten()
210          }
211  }
```

Параллелим обработку



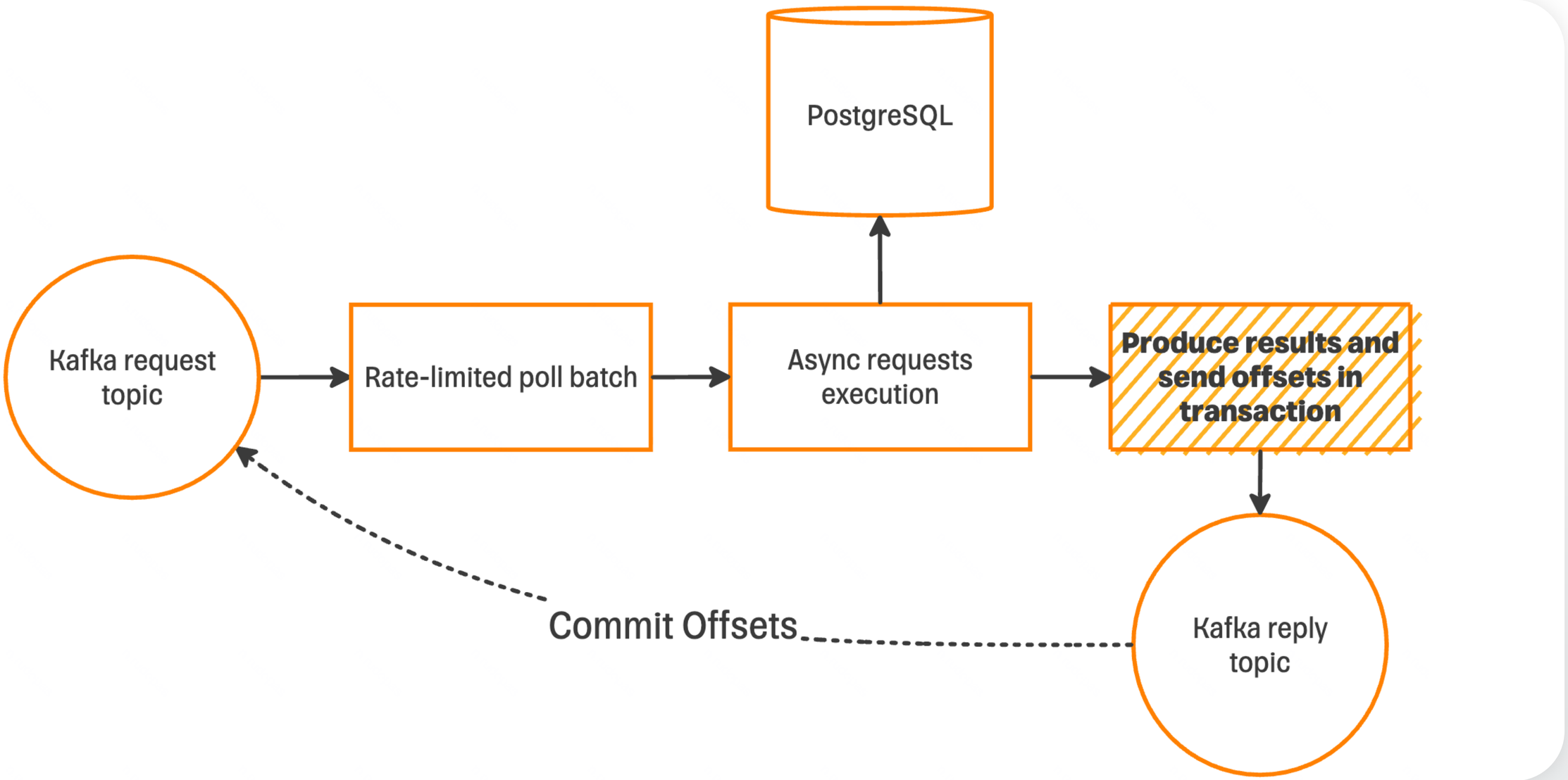

```
192  @Service
193  class NishtyakRequestService(
194      private val nishtyakRequestRouter: NishtyakRequestRouter,
195      private val batchDispatcher: CoroutineDispatcher,
196  ) {
197      fun process(
198          requests: Iterable<Pair<String, NishtyakRequest>>
199      ): List<Pair<String, NishtyakResponse>> =
200          runBlocking(context = batchDispatcher) {
201              requests.groupBy({ it.first }, { it.second })
202                  .map { (key, requests) ->
203                      async {
204                          nishtyakRequestRouter.route(requests)
205                              .map { key to it }
206                      }
207                  }
208                  .awaitAll()
209                  .flatten()
210          }
211  }
```

Параллелим обработку




```
192  @Service
193  class NishtyakRequestService(
194      private val nishtyakRequestRouter: NishtyakRequestRouter,
195      private val batchDispatcher: CoroutineDispatcher,
196  ) {
197      fun process(
198          requests: Iterable<Pair<String, NishtyakRequest>>
199      ): List<Pair<String, NishtyakResponse>> =
200          runBlocking(context = batchDispatcher) {
201              requests.groupBy({ it.first }, { it.second })
202                  .map { (key, requests) ->
203                      async {
204                          nishtyakRequestRouter.route(requests)
205                              .map { key to it }
206                      }
207                  }
208                  .awaitAll()
209                  .flatten()
210          }
211  }
```

Это мы уже проходили



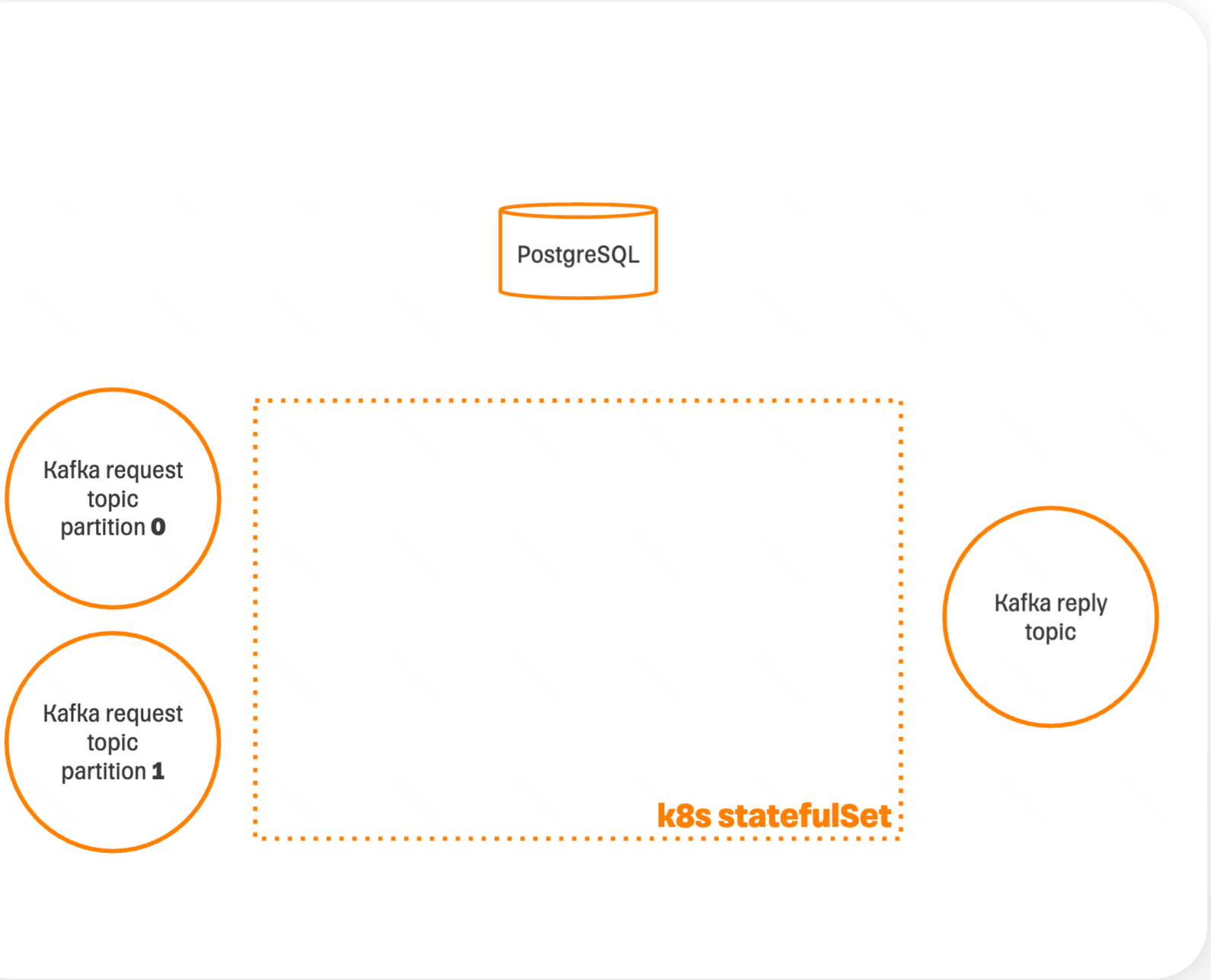
Продьюсим

```
fun <Key, Value> produce(
    responses: List<Pair<Key, Value>>,
    responseTopic: String,
    producer: KafkaProducer<Key, Value>,
    offsets: Collection<
        Pair<
            ConsumerGroupMetadata,
            Map<TopicPartition, OffsetAndMetadata>
            >
        >
) {
    try {
        producer.beginTransaction()
        responses.forEach { response ->
            producer.send(
                ProducerRecord(
                    responseTopic,
                    response.first,
                    response.second
                )
            )
        }
        offsets.forEach { (consumerGroupMetadata, offset) ->
            producer.sendOffsetsToTransaction(
                offset,
                consumerGroupMetadata
            )
        }
        producer.commitTransaction()
    } catch (e: Exception) {
        producer.abortTransaction()
        throw e
    }
}
```

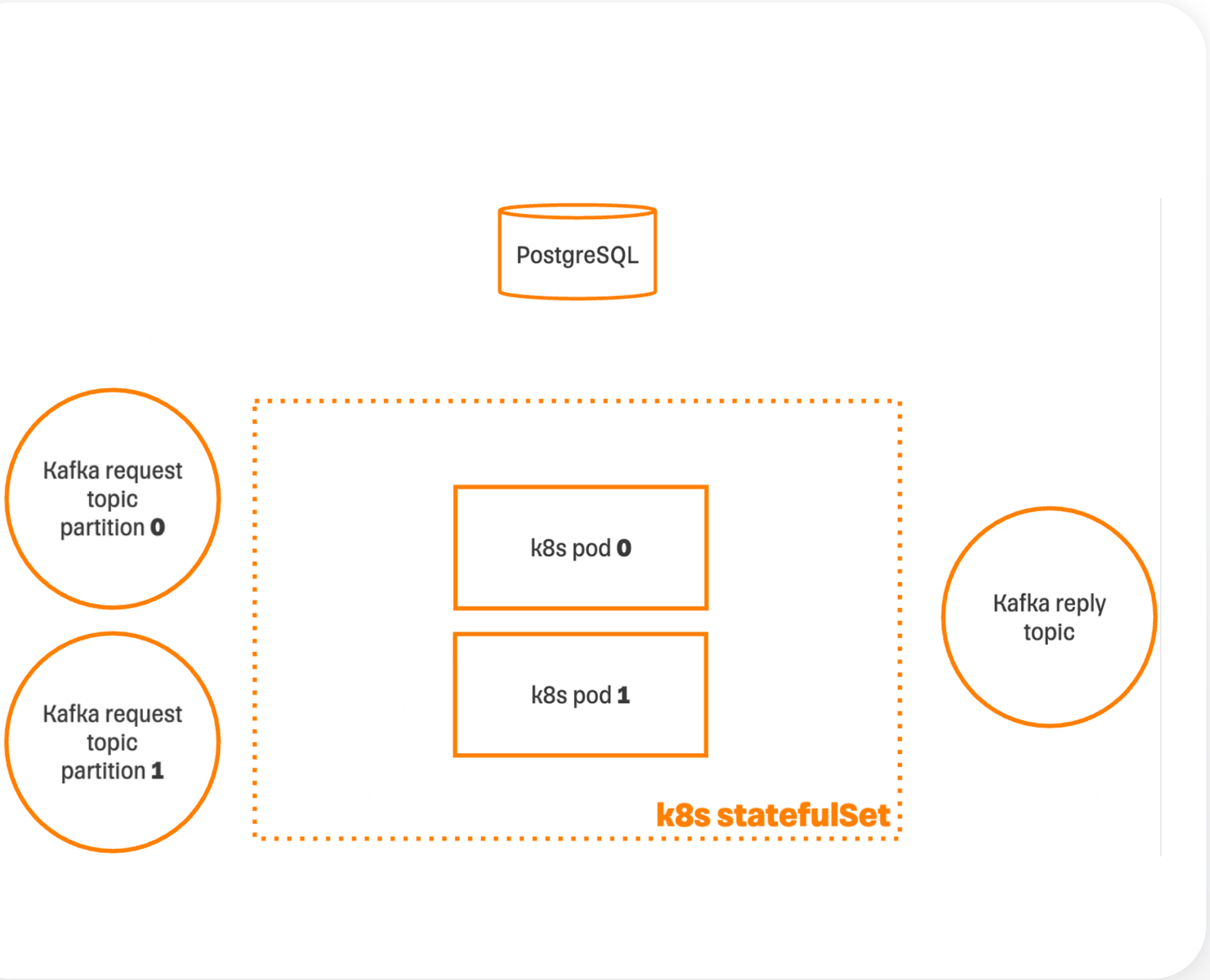
```
val bucket = getBucket(maxPollRecords)
consumer.subscribe(listOf(requestTopic))
while (true) {
    val requests = pollWithLimit(
        bucket,
        consumer,
        pollTimeout,
        maxPollRecords
    )

    val responses = nishtyakService.process(
        requests.map { it.key() to it.value() }
    )

    val offsets = listOf(
        consumer.groupMetadata() to getOffsets(requests)
    )

    produce(responses, responseTopic, producer, offsets)

    logMetrics(requests)
}
```

Собираем и деплоим

Деплоим на QA

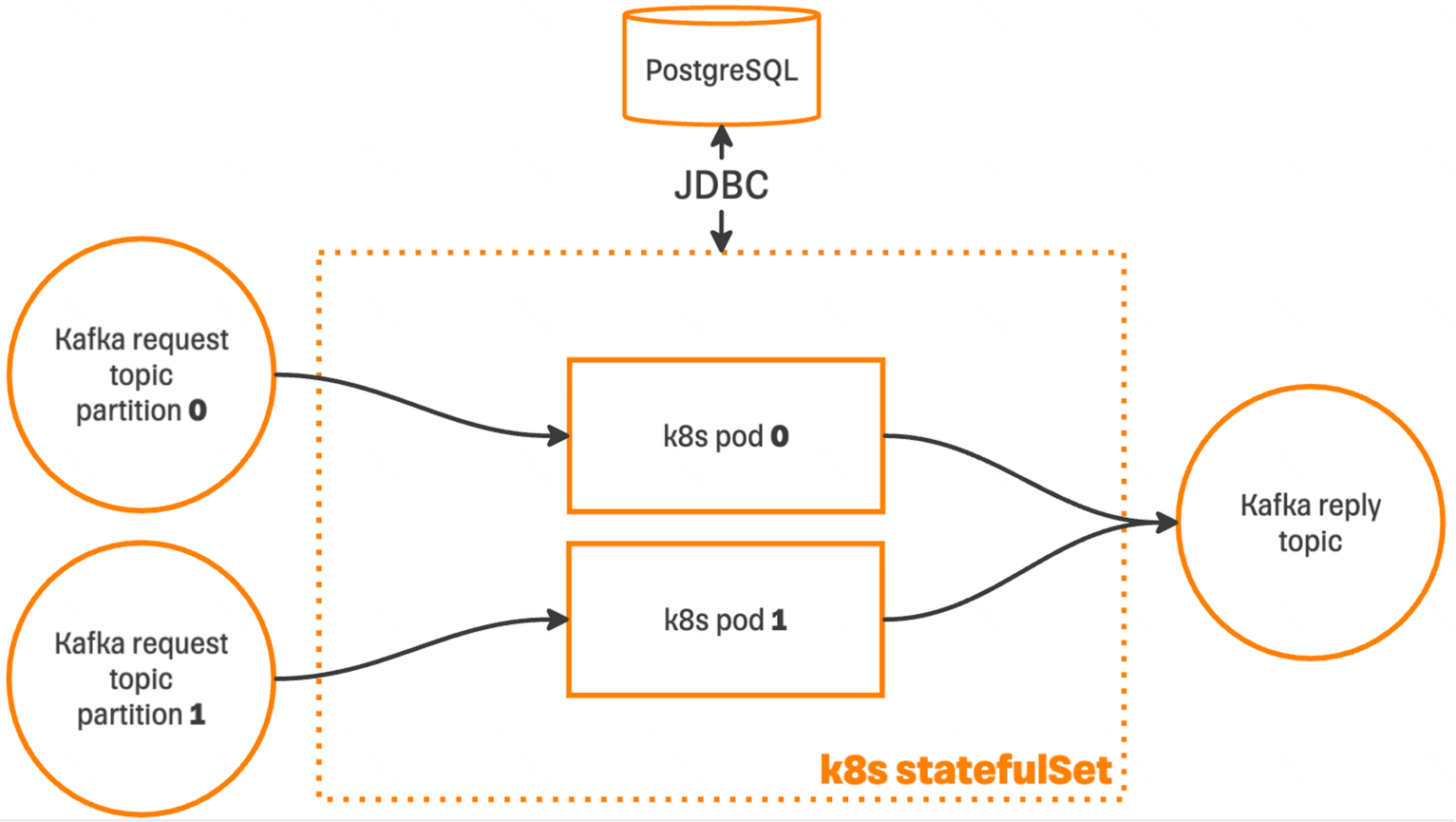


Деплоим на QA



Деплоим на QA

- На pod по партиции

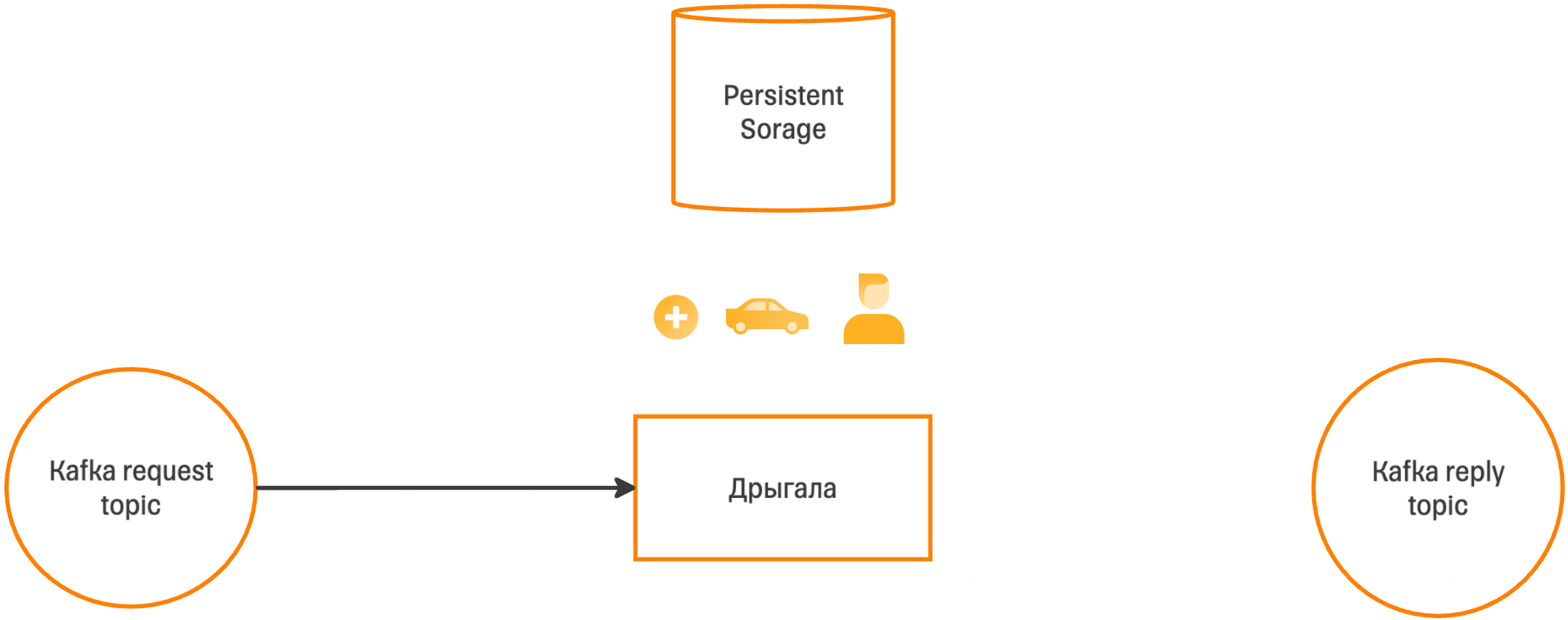


Через день нас находит QA

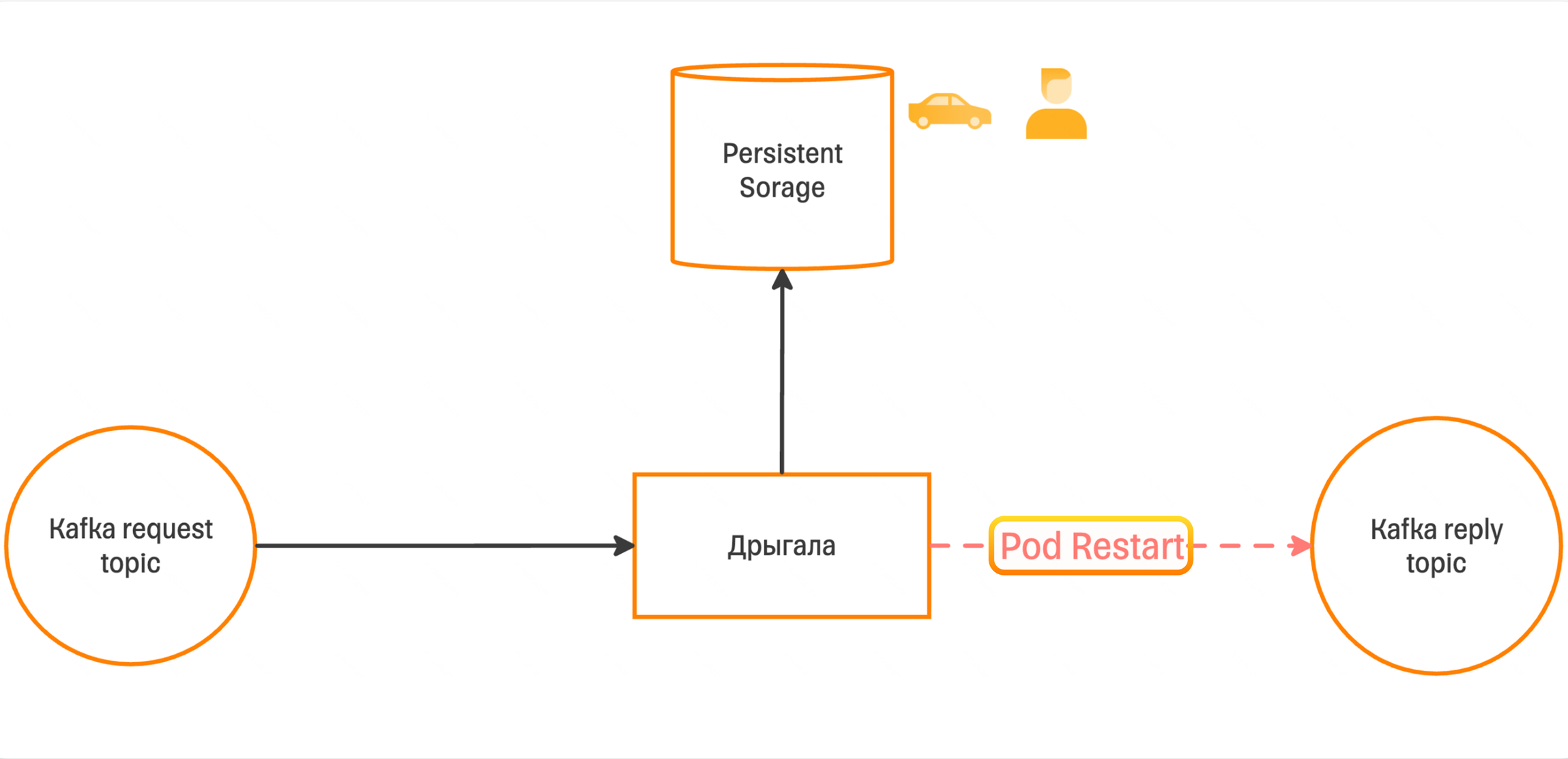
Кейс

Persistent
Sorage
Kafka request
topic
Дрыгала
Kafka reply
topic

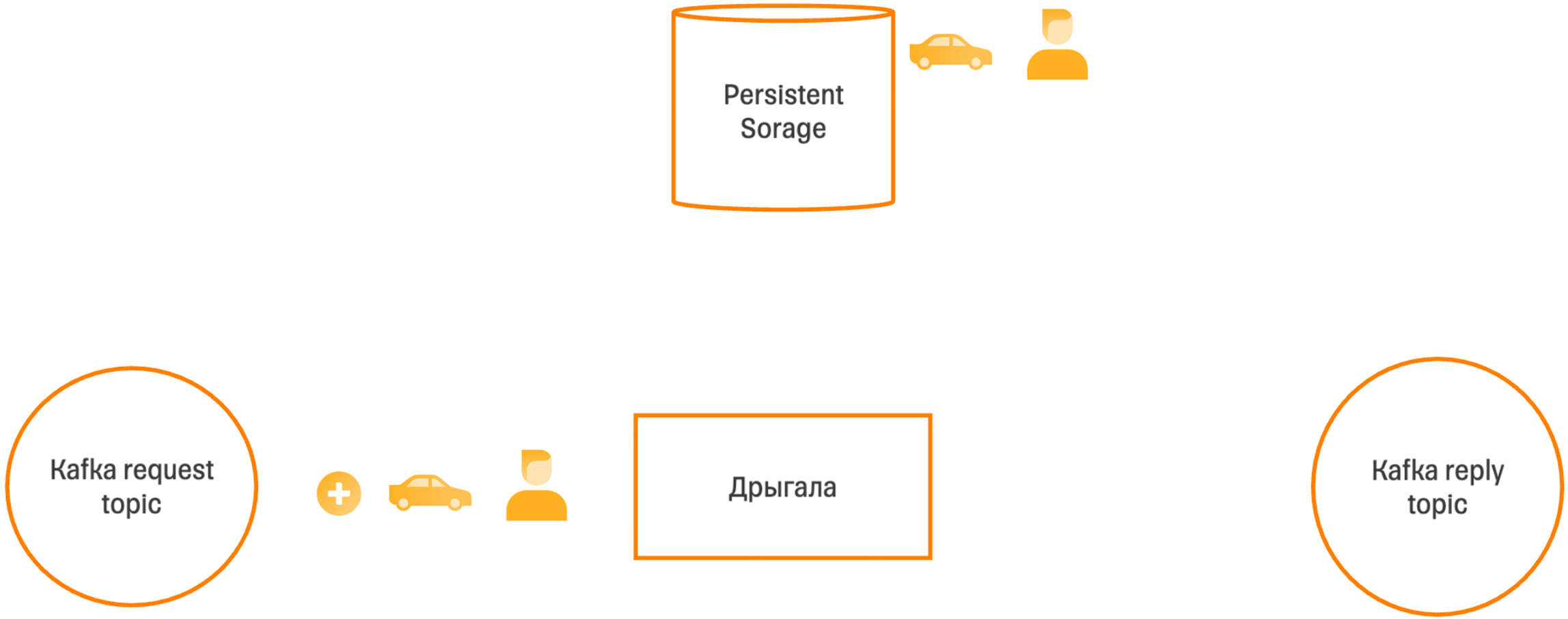
Кейс


Persistent
Sorage
Kafka request
topic
Дрыгала
Kafka reply
topic

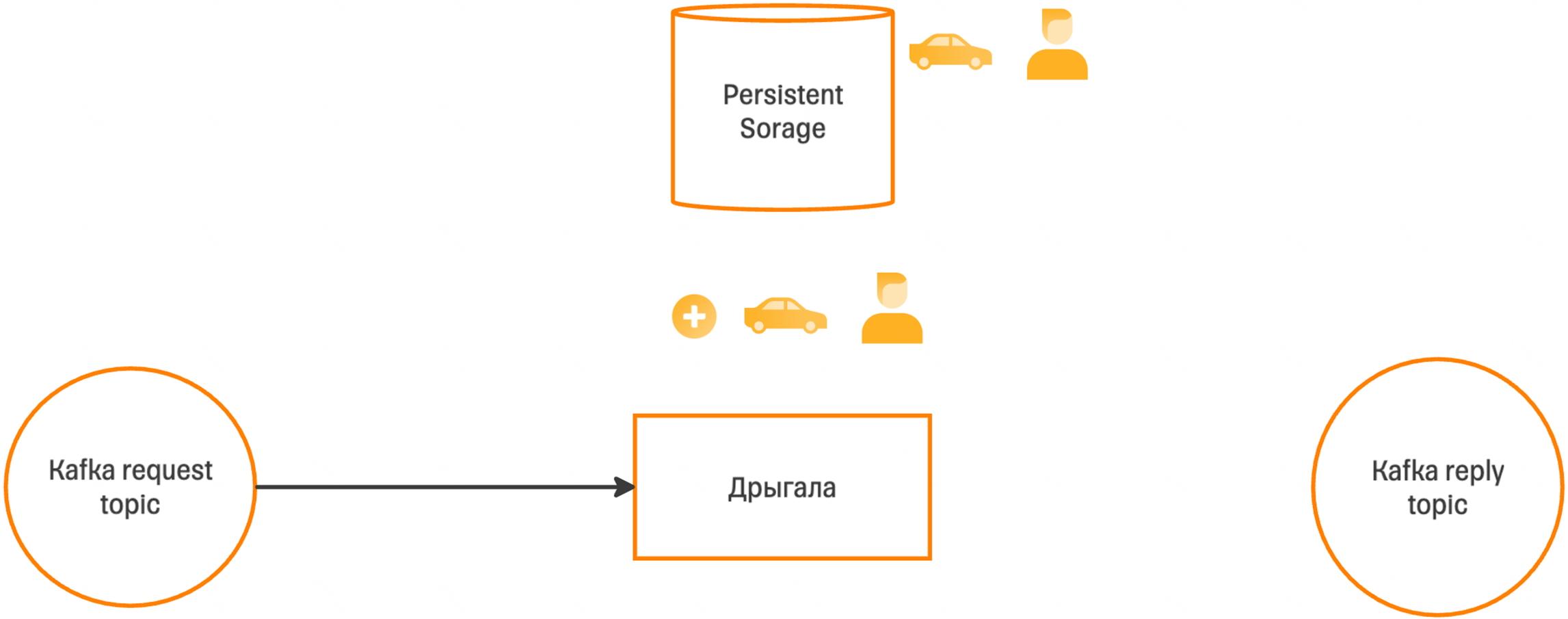
Кейс


Persistent
Sorage
Kafka request
topic
Дрыгала
Pod Restart
Kafka reply
topic

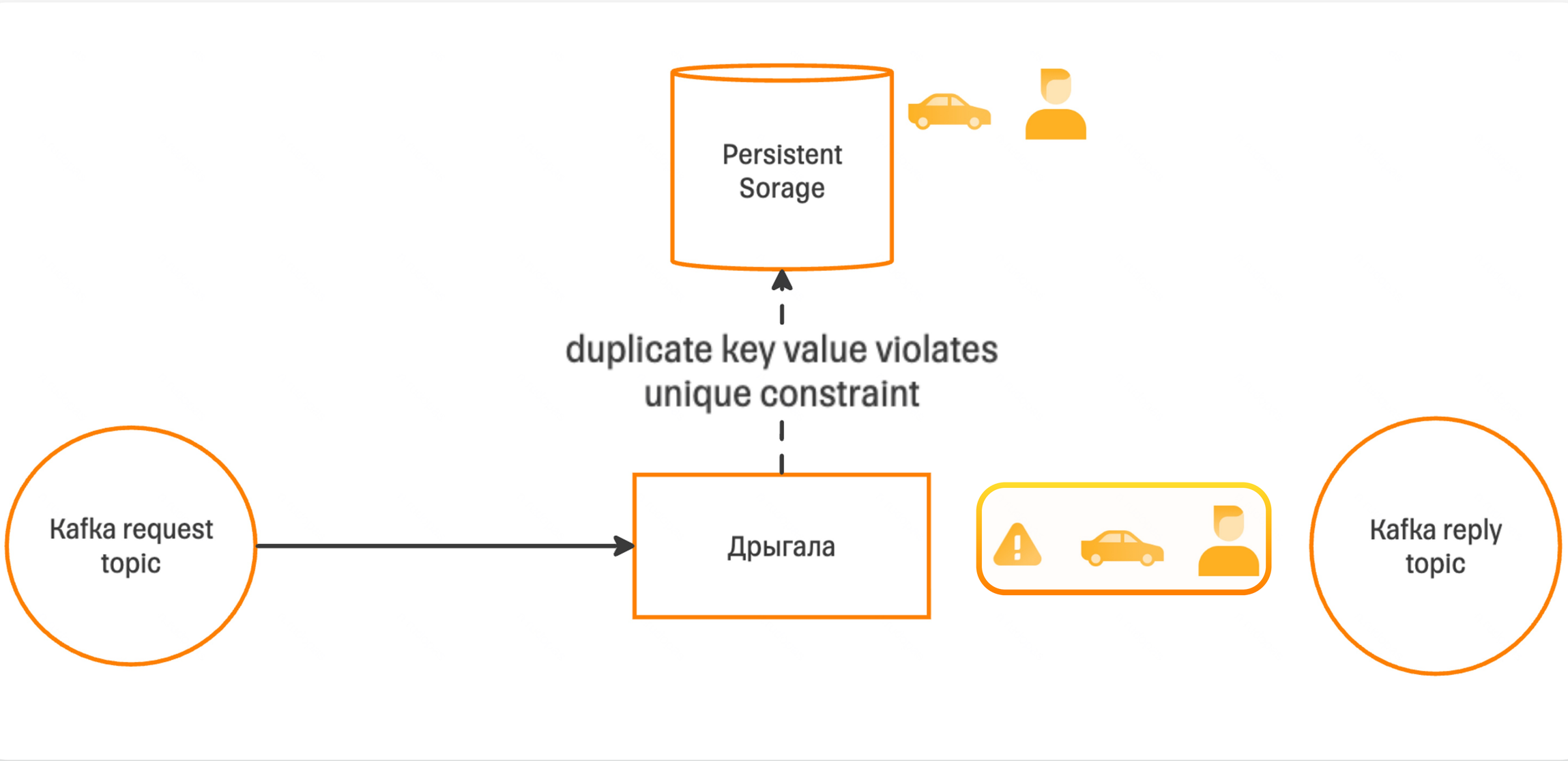
Кейс


Persistent
Sorage
Kafka request
topic
Дрыгала
Kafka reply
topic

Кейс


Persistent
Sorage
Kafka request
topic
Дрыгала
Kafka reply
topic

Кейс


Persistent
Sorage
duplicate key value violates
unique constraint
Kafka request
topic
Дрыгала
Kafka reply
topic

Transactional Otbox

Но для бедных

Outbox для бедных

Response		
PK	idempotency_key	TEXT
	response	JSONB

```
47    @Repository
48    interface ResponseRepository {
49        fun create(
50            idempotencyKey: String,
51            response: NishtyakResponse
52        )
53
54        fun get(
55            idempotencyKey: String
56        ): NishtyakResponse?
57    }
58
59    interface NishtyakRequest {
60        val idempotencyKey: String
61        val humanId: String
62        val requestType: String
63    }
```

Outbox для бедных

Response		
PK	idempotency_key	TEXT
	response	JSONB

```
47    @Repository
48    interface ResponseRepository {
49        fun create(
50            idempotencyKey: String,
51            response: NishtyakResponse
52        )
53
54        fun get(
55            idempotencyKey: String
56        ): NishtyakResponse?
57    }
58
59    interface NishtyakRequest {
60        val idempotencyKey: String
61        val humanId: String
62        val requestType: String
63    }
```

Outbox для бедных

Response		
PK	idempotency_key	TEXT
	response	JSONB

```
47  @Repository
48  interface ResponseRepository {
49      fun create(
50          idempotencyKey: String,
51          response: NishtyakResponse
52      )
53
54      fun get(
55          idempotencyKey: String
56      ): NishtyakResponse?
57  }
58
59  interface NishtyakRequest {
60      val idempotencyKey: String
61      val humanId: String
62      val requestType: String
63  }
```

Изменения в обработке

```
208    private fun doRoute(request: NishtyakRequest): NishtyakResponse {
209        val processor = processors[request.requestType]
210            ?: return ErrorNishtyakResponse(
211                humanId = request.humanId,
212                errorMessage =
213                "Unprocessable request type ${request.requestType}"
214            )
215
216        val response = try {
217            processor.process(request)
218        } catch (e: Exception) {
219            ErrorNishtyakResponse(
220                humanId = request.humanId,
221                errorMessage = "Something went wrong: ${e.message}"
222            )
223        }
224        responseRepository.create(
225            idempotencyKey = request.idempotencyKey,
226            response = response
227        )
228        return response
229    }
```

```
188    @Service
189    class NishtyakRequestRouter(
190        processors: List<NishtyakRequestProcessor>,
191        private val responseRepository: ResponseRepository,
192    ) {
193        private val processors = processors
194            .associateBy { it.requestType }
195
196        @Transactional
197        fun route(
198            requests: Iterable<NishtyakRequest>
199        ): List<NishtyakResponse> =
200            requests.map(this::route)
201
202        private fun route(
203            request: NishtyakRequest
204        ): NishtyakResponse =
205            responseRepository.get(request.idempotencyKey)
206                ?: doRoute(request)
```

Изменения в обработке

```
208  private fun doRoute(request: NishtyakRequest): NishtyakResponse {
209      val processor = processors[request.requestType]
210          ?: return ErrorNishtyakResponse(
211              humanId = request.humanId,
212              errorMessage =
213              "Unprocessable request type ${request.requestType}"
214          )
215
216      val response = try {
217          processor.process(request)
218      } catch (e: Exception) {
219          ErrorNishtyakResponse(
220              humanId = request.humanId,
221              errorMessage = "Something went wrong: ${e.message}"
222          )
223      }
224      responseRepository.create(
225          idempotencyKey = request.idempotencyKey,
226          response = response
227      )
228      return response
229  }
```

```
188  @Service
189  class NishtyakRequestRouter(
190      processors: List<NishtyakRequestProcessor>,
191      private val responseRepository: ResponseRepository,
192  ) {
193      private val processors = processors
194          .associateBy { it.requestType }
195
196      @Transactional
197      fun route(
198          requests: Iterable<NishtyakRequest>
199      ): List<NishtyakResponse> =
200          requests.map(this::route)
201
202      private fun route(
203          request: NishtyakRequest
204      ): NishtyakResponse =
205          responseRepository.get(request.idempotencyKey)
206              ?: doRoute(request)
```

Изменения в обработке

```
208  private fun doRoute(request: NishtyakRequest): NishtyakResponse {
209      val processor = processors[request.requestType]
210          ?: return ErrorNishtyakResponse(
211              humanId = request.humanId,
212              errorMessage =
213              "Unprocessable request type ${request.requestType}"
214          )
215
216      val response = try {
217          processor.process(request)
218      } catch (e: Exception) {
219          ErrorNishtyakResponse(
220              humanId = request.humanId,
221              errorMessage = "Something went wrong: ${e.message}"
222          )
223      }
224      responseRepository.create(
225          idempotencyKey = request.idempotencyKey,
226          response = response
227      )
228      return response
229  }
```

```
188  @Service
189  class NishtyakRequestRouter(
190      processors: List<NishtyakRequestProcessor>,
191      private val responseRepository: ResponseRepository,
192  ) {
193      private val processors = processors
194          .associateBy { it.requestType }
195
196      @Transactional
197      fun route(
198          requests: Iterable<NishtyakRequest>
199      ): List<NishtyakResponse> =
200          requests.map(this::route)
201
202      private fun route(
203          request: NishtyakRequest
204      ): NishtyakResponse =
205          responseRepository.get(request.idempotencyKey)
206              ?: doRoute(request)
```

Предположим, мы учли это сразу

```
215     fun startProcessing(
223     ) {
224         val bucket = getBucket(maxPollRecords)
225         consumer.subscribe(listOf(requestTopic))
226         while (true) {
227             val requests = pollWithLimit(
228                 bucket,
229                 consumer,
230                 pollTimeout,
231                 maxPollRecords
232             )
233
234             val offsets = listOf(
235                 consumer.groupMetadata() to getOffsets(requests)
236             )
237
238             val responses = nishtyakService.process(
239                 requests.map { it.key() to it.value() }
240             )
```

```
245     fun startProcessing(
253     ) {
254         val bucket = getBucket(maxPollRecords)
255         consumer.subscribe(listOf(requestTopic))
256         while (true) {
257             val requests = pollWithLimit(
258                 bucket,
259                 consumer,
260                 pollTimeout,
261                 maxPollRecords
262             )
263
264             val offsets = listOf(
265                 consumer.groupMetadata() to getOffsets(requests)
266             )
267
268             val responses = nishtyakService.process(
269                 requests.asSequence()
270                     .distinctBy { it.value().idempotencyKey }
271                     .map { it.key() to it.value() }
272                     .toList()
273             )
```

Ничего мы не забыли

Ответ на каждый запрос

Никогда и ни при каких обстоятельствах не терять сообщения

Ничего мы не забыли

Ответ на каждый запрос

Никогда и ни при каких обстоятельствах не терять сообщения

Сохранять порядок

Обязательно не менять порядок запросов и ответов

Ничего мы не забыли

Ответ на каждый запрос

Никогда и ни при каких обстоятельствах не терять сообщения

Сохранять порядок

Обязательно не менять порядок запросов и ответов

Не терять изменения

Должна быть возможность воссоздать актуальное состояние по нашим ответам

Подытожим

Когда колеса квадратные

- Не нужен request/reply по Kafka
- Вы гуру-вуду по бесплатному Transactional Inbox/Outbox
- Все операции идемпотентные и дедубликация бесплатная
- Обработка самого запроса по времени занимает часы

Подытожим

Когда колеса квадратные

- Не нужен request/reply по Kafka
- Вы гуру-вуду по бесплатному Transactional Inbox/Outbox
- Все операции идемпотентные и дедубликация бесплатная
- Обработка самого запроса по времени занимает часы

Подытожим

Когда колеса квадратные

- Не нужен request/reply по Kafka
- Вы гуру-вуду по бесплатному Transactional Inbox/Outbox
- Все операции идемпотентные и дедубликация бесплатная
- Обработка самого запроса по времени занимает часы

Подытожим

Когда колеса квадратные

- Не нужен request/reply по Kafka
- Вы гуру-вуду по бесплатному Transactional Inbox/Outbox
- Все операции идемпотентные и дедубликация бесплатная
- Обработка самого запроса по времени занимает часы

Подытожим

Когда велосипед красивый

- Почему-то нужен request/reply по Kafka
- Скорость обработки лучше не добивать об Transactional Inbox/Outbox
- Надежность обработки важнее скорости и накладных расходов
- Обработка самого запроса по времени занимает менее нескольких минут

Подытожим

Когда велосипед красивый

- Почему-то нужен request/reply по Kafka
- Скорость обработки лучше не добивать об Transactional Inbox/Outbox
- Надежность обработки важнее скорости и накладных расходов
- Обработка самого запроса по времени занимает менее нескольких минут

Подытожим

Когда велосипед красивый

- Почему-то нужен request/reply по Kafka
- Скорость обработки лучше не добивать об Transactional Inbox/Outbox
- Надежность обработки важнее скорости и накладных расходов
- Обработка самого запроса по времени занимает менее нескольких минут

Подытожим

Когда велосипед красивый

- Почему-то нужен request/reply по Kafka
- Скорость обработки лучше не добивать об Transactional Inbox/Outbox
- Надежность обработки важнее скорости и накладных расходов
- Обработка самого запроса по времени занимает менее нескольких минут