управление данными в микросервисах на C#

Kafka и .NET

kafka

основные требования

- Отделение издателей и потребителей по средствам реактивной модели взаимодействия
- Персистентное хранение событий
- Упорядоченность событий
- Возможность работы с большими потоками данных
- Возможность горизонтального масштабирования

устройство kafka

устройство kafka

действующие лица

- издатели (producer) – создают публикуют данные в потоках событий
- потребители (consumer) – обрабатывают данные из потоков событий
- топики (topic) – именованный поток событий
- сообщение (message) – событие из топика

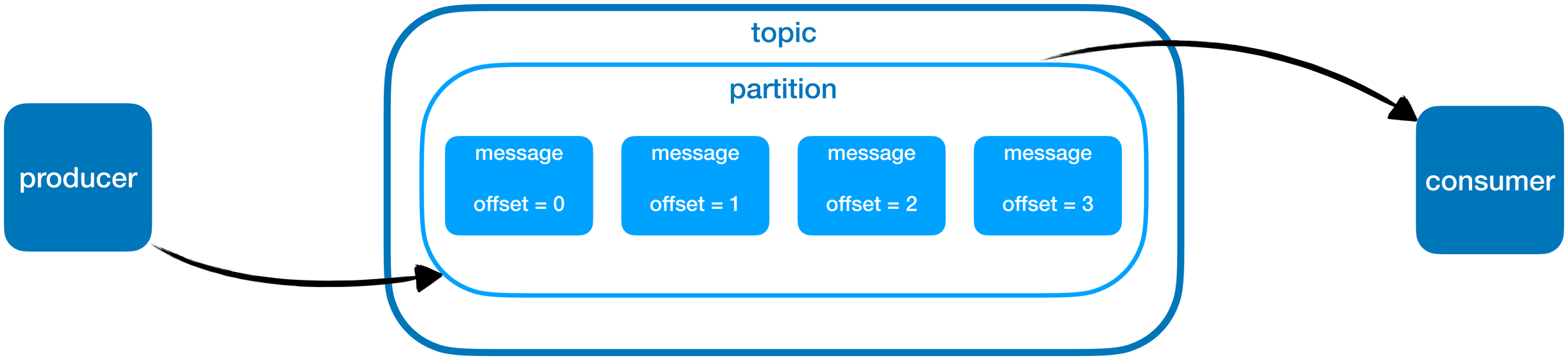
устройство kafka

сообщений

- ключ
- значение
- временная отметка
- хедеры

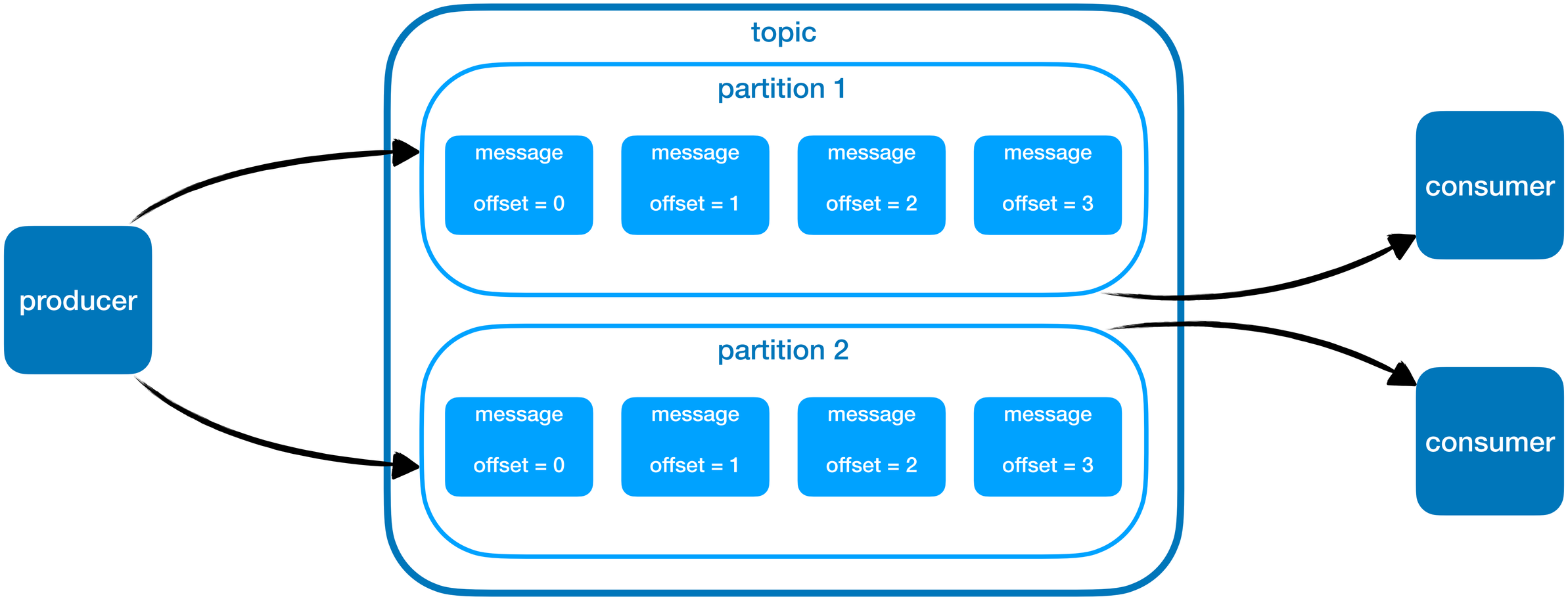
устройство kafka

партиции



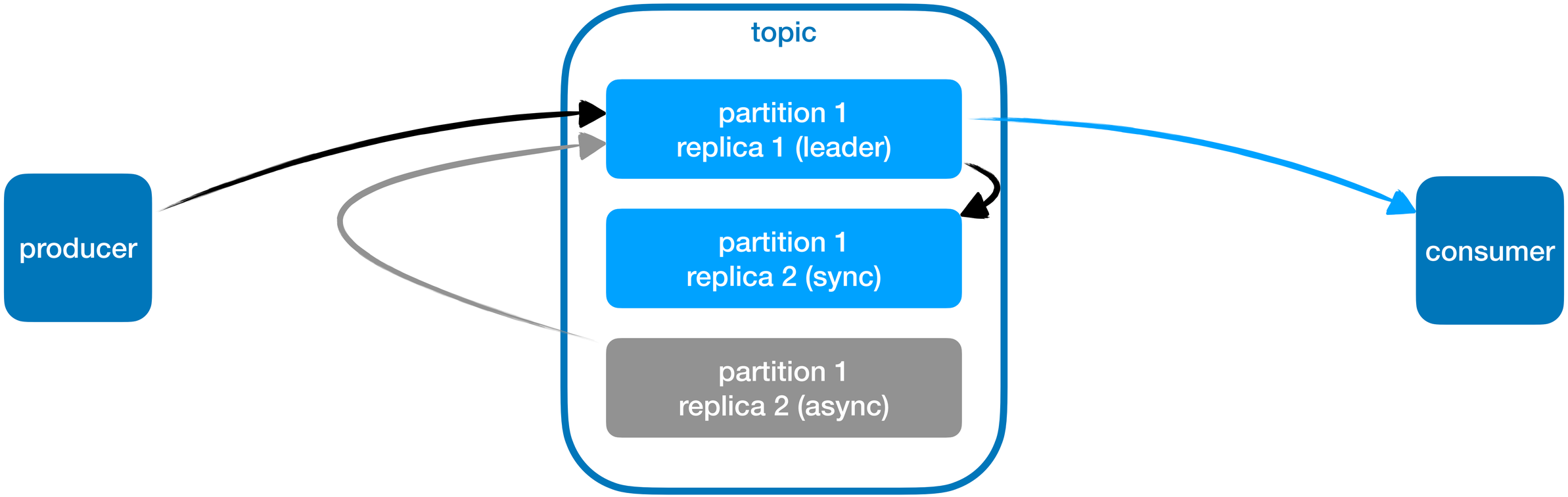
устройство kafka

партиции



устройство kafka

репликация



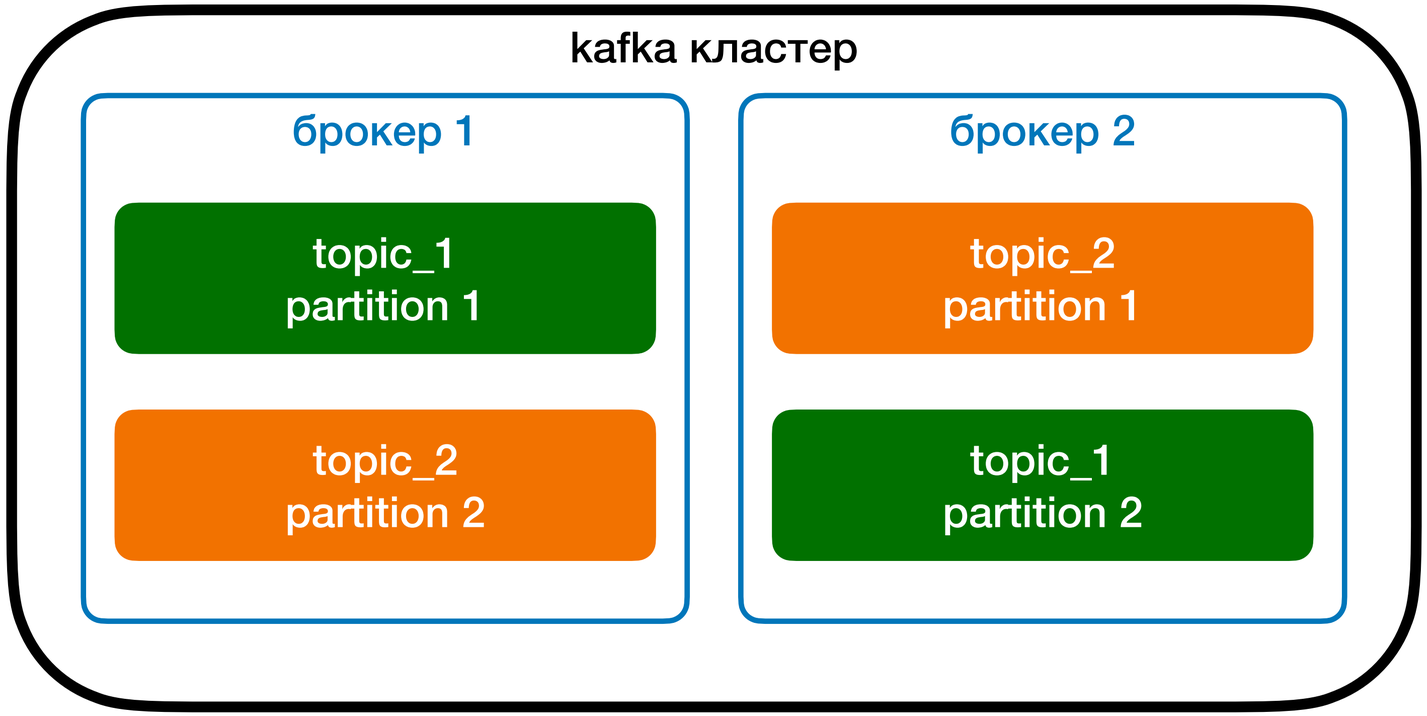
устройство kafka

брокеры

- отдельные узлы в kafka кластере
- хранят в себе данные партиций
- отвечают за запись и чтение данных с партиций
- отвечают за некоторые аспекты управления
- Zookeeper выбирает брокер-контроллер
- брокер-контроллер выбирает лидер реплики для партиций

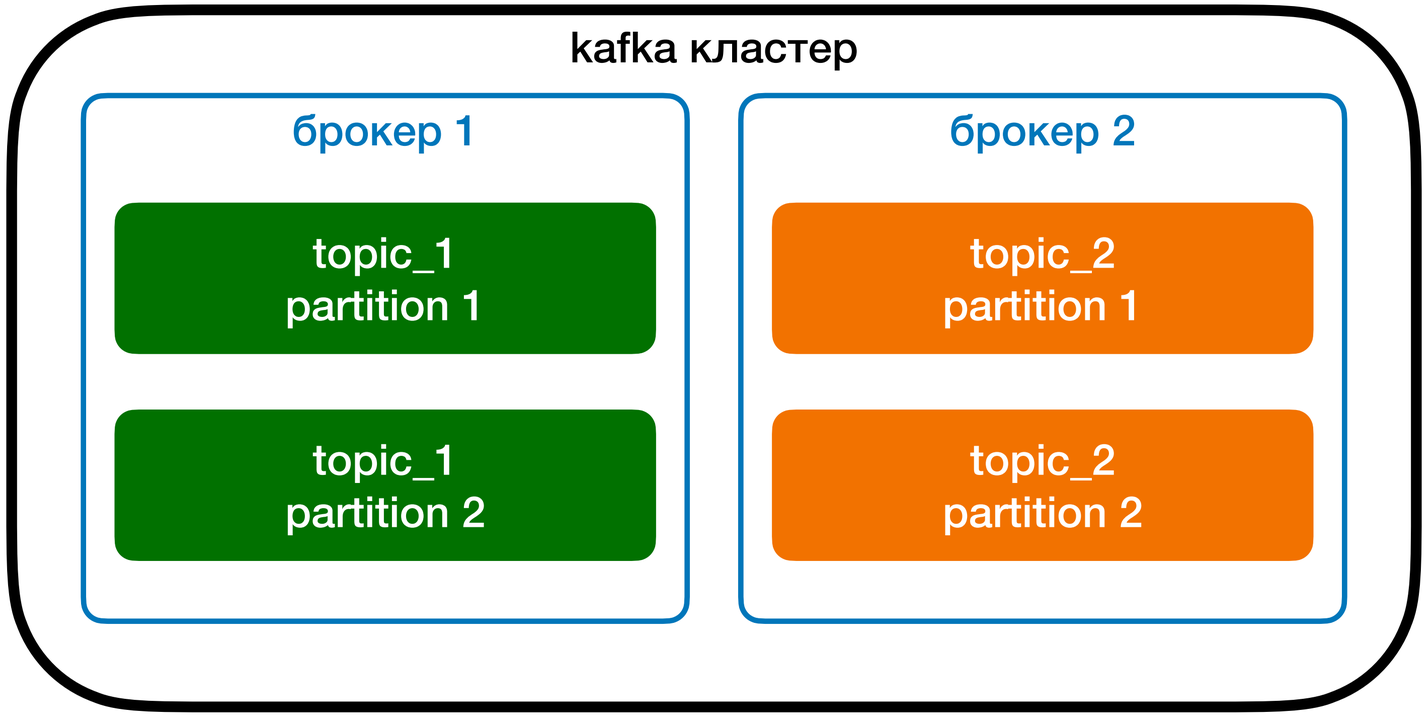
устройство kafka

брокеры



устройство kafka

брокеры



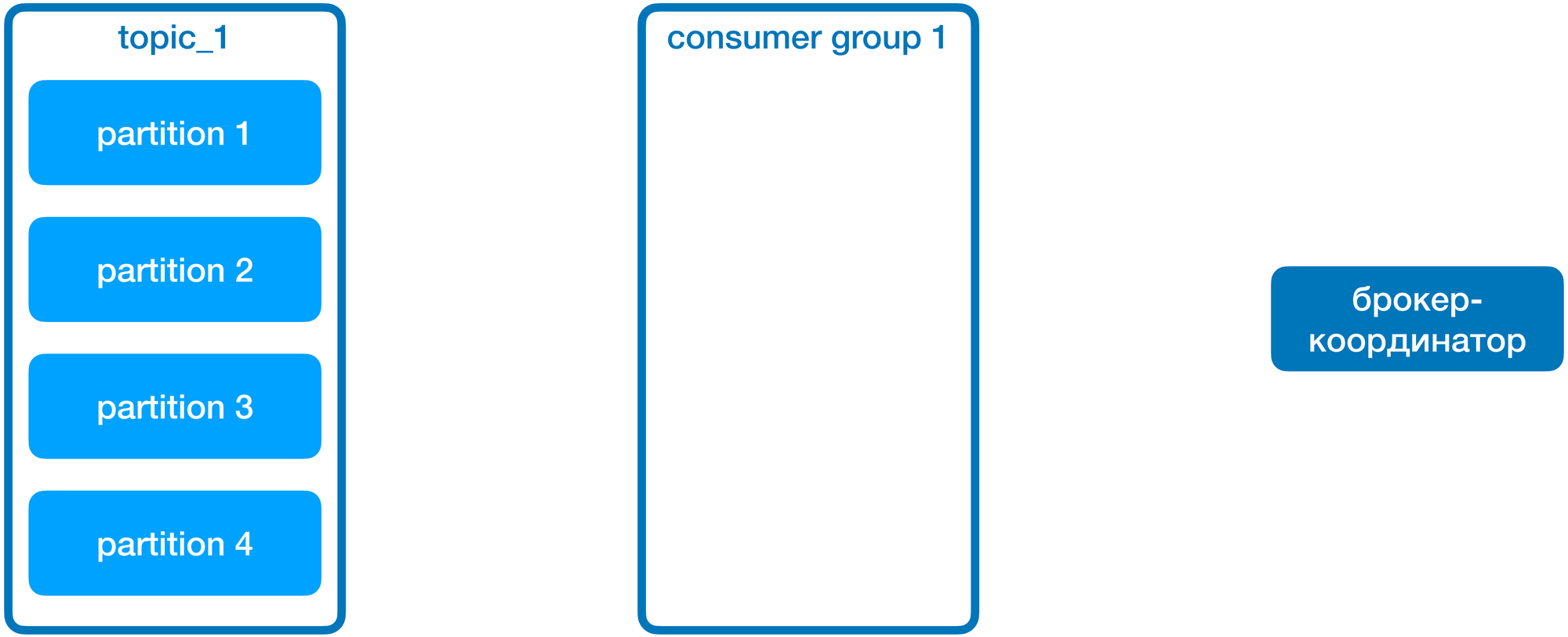
устройство kafka

consumer groups

- позволяют объединить несколько экземпляров сервисов в “одного потребителя”
- позволяют избежать повторной обработки сообщений

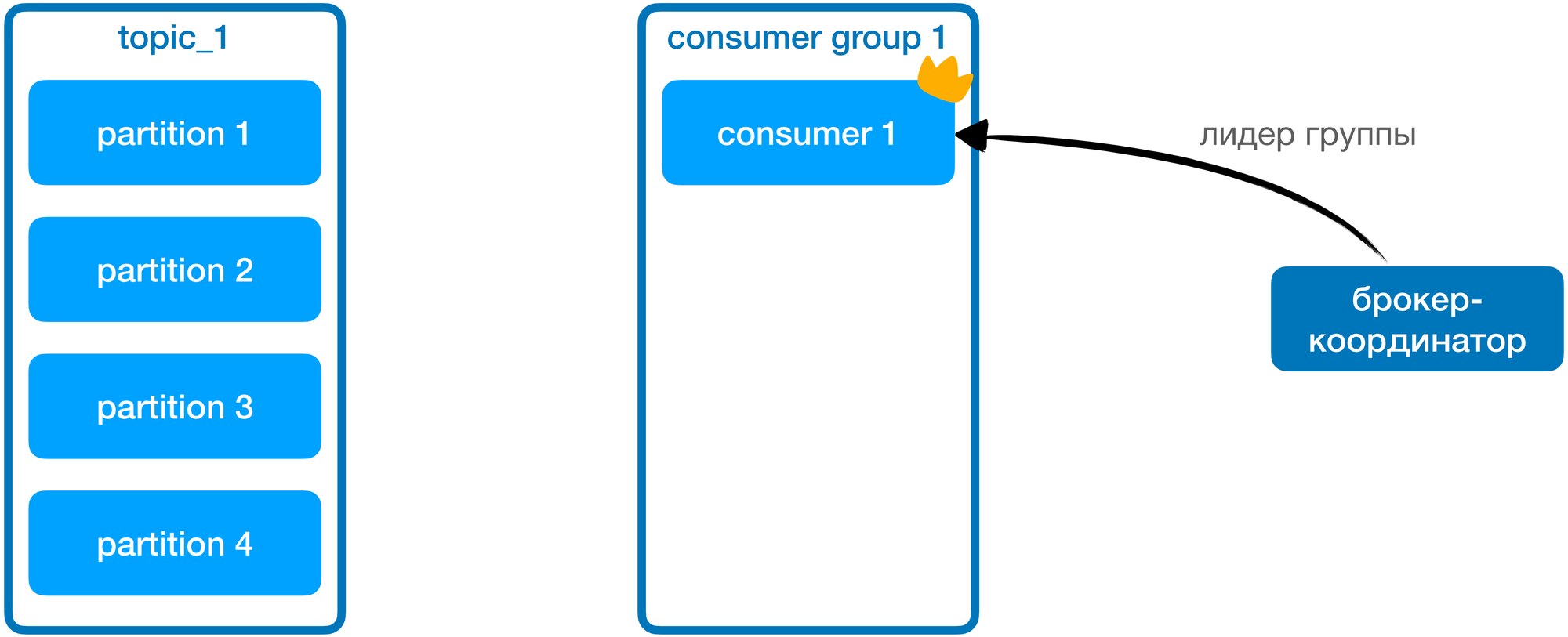
устройство kafka

consumer groups



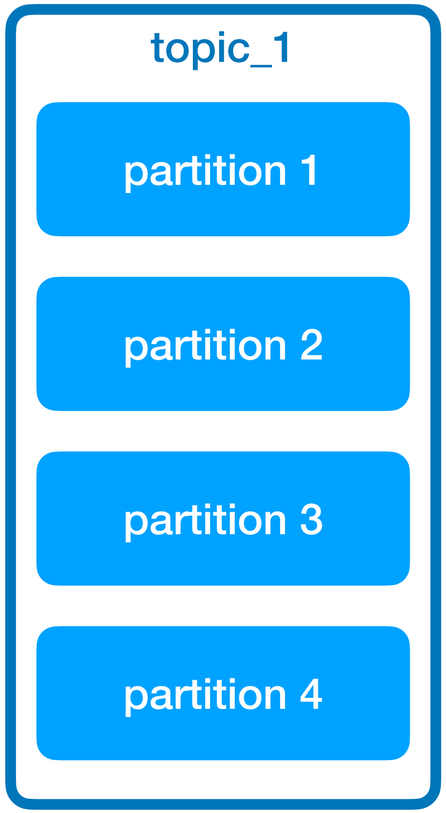
устройство kafka

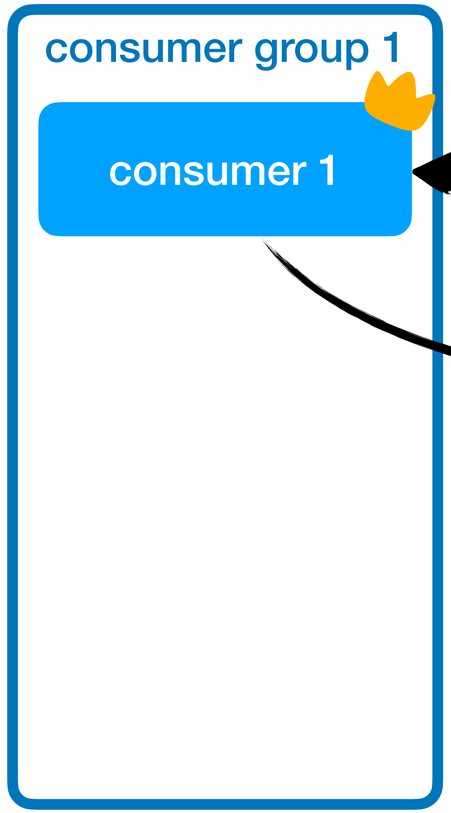
consumer groups



устройство kafka

consumer groups





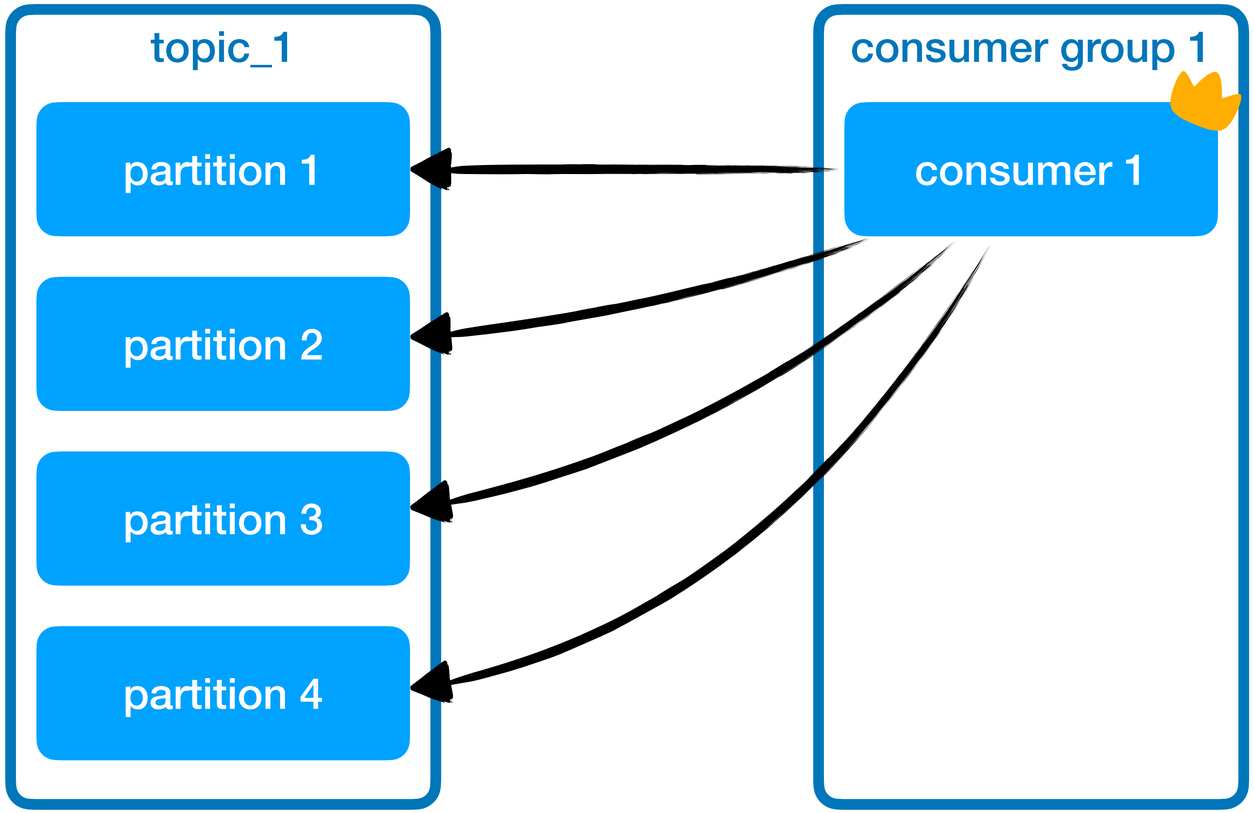

```
consumers = [
  ‘consumer 1’
]
```

брокер-координатор

```
consumer_partitions = [
  ‘consumer 1’: [1, 2, 3, 4]
]
```

устройство kafka

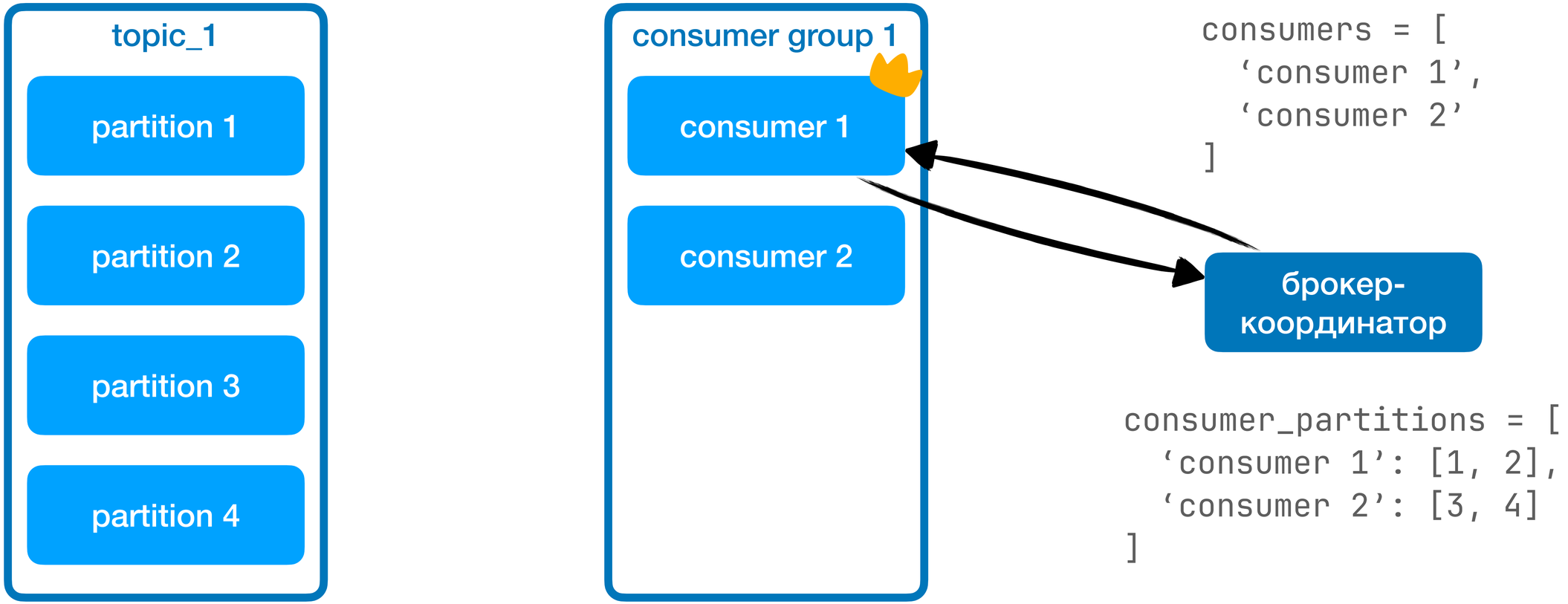
consumer groups



брокер-
координатор

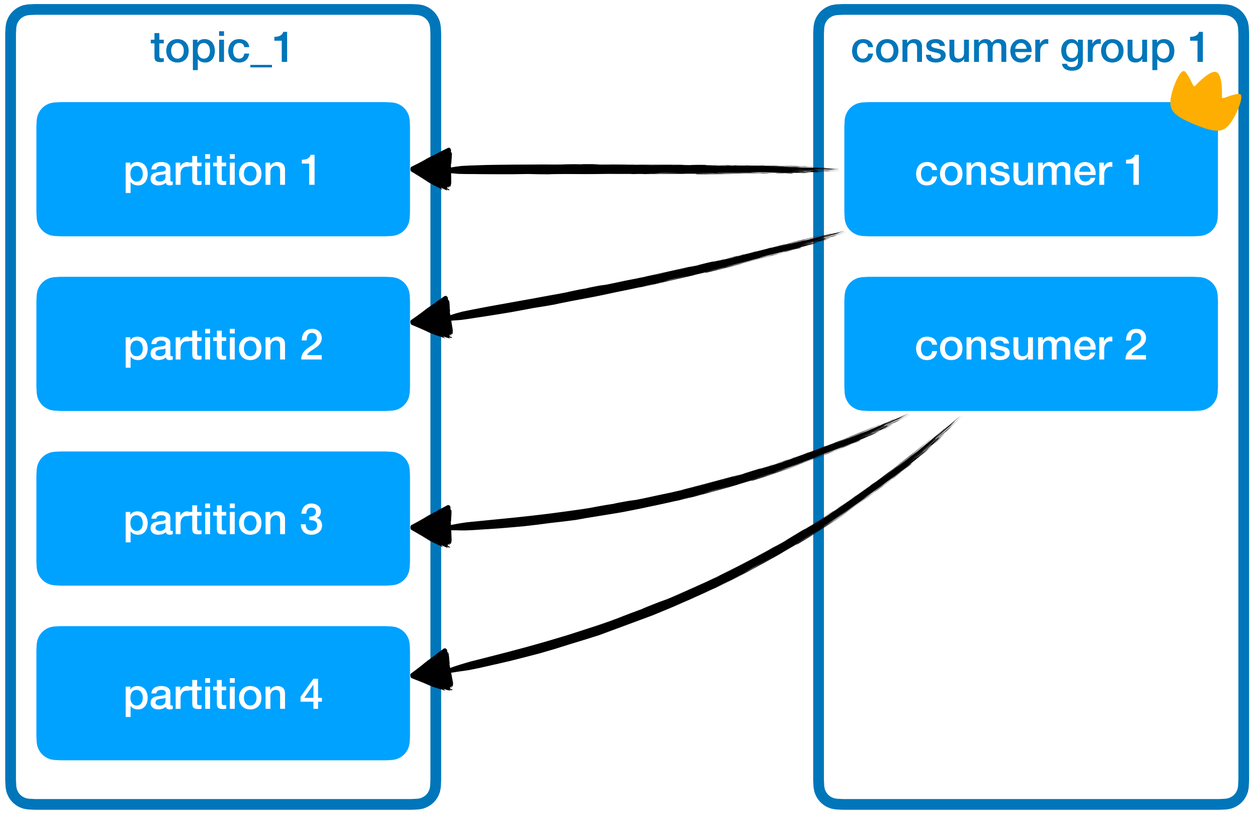
устройство kafka

consumer groups



устройство kafka

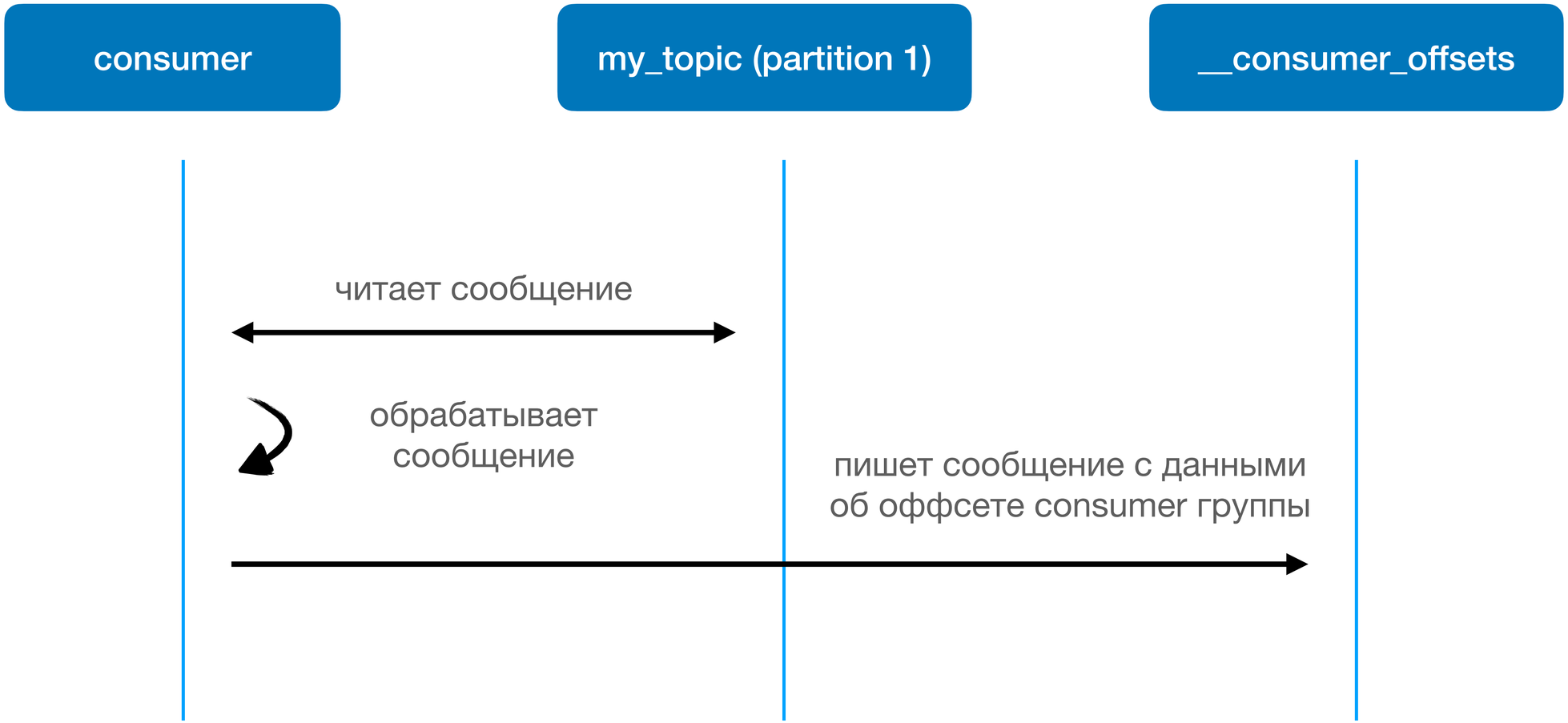
consumer groups



брокер-
координатор

устройство kafka

чтение сообщения



acknowledgement

acknowledgement

producer

- ack = 0 – подтверждение от брокера не ожидается
- ack = 1 – ожидается подтверждение записи в лидер реплику
- ack = all – ожидается подтверждение записи во все sync реплики

acknowledgement

consumer

- auto commit – как только сообщение считано – считается обработанным (at most once)
- manual commit – сообщение считается обработанным когда потребитель это явно укажет, обычно после обработки сообщения (at least once)
- custom offset manager – inbox+outbox (exactly once)

kafka и .NET

kafka и .NET

Confluent.Kafka

- Confluent.Kafka – SDK для работы с Kafka в .NET
- представляет собой обёртку над librdkafka
- предоставляет абстракции для конфигурации и использования consumer и producer

confluent kafka

producer

- конфигурация
  - BootstrapServers – адреса kafka брокеров
  - KeySerializer – сериализатор для преобразования объекта-ключа в байты
  - ValueSerializer – сериализатор для преобразования объекта-значения в байты
- создание сообщения и отправка его в конкретный топик

confluent kafka

producer

```
var config = new ProducerConfig
{
    BootstrapServers = kafkaOptions.Host,
};

_producer = new ProducerBuilder<TKey, TValue>(config)
    .SetKeySerializer(keySerializer)
    .SetValueSerializer(valueSerializer)
    .Build();
```

confluent kafka

producer

```
var message = new Message<TKey, TValue>
{
    Key = producerKafkaMessage.Key,
    Value = producerKafkaMessage.Value,
};

await _producer.ProduceAsync("my_topic", message, cancellationToken);
```

confluent kafka

consumer

- конфигурация
  - BootstrapServers – адреса kafka брокеров
  - GroupId – идентификатор consumer группы
  - GroupInstanceId – идентификатор конкретного консьюмера в рамках группы
  - AutoOffsetReset – стратегия выбора изначального офсета при чтении топика
  - KeyDeserializer – сериализатор для преобразования байтов в объект-ключ
  - ValueDeserializer – сериализатор для преобразования байтов в объект-значения
- подписка на конкретный топик
- чтение сообщения

confluent kafka

consumer

```
var config = new ConsumerConfig
{
    BootstrapServers = _kafkaOptions.Host,
    GroupId = _consumerOptions.Group,
    GroupInstanceId = _consumerOptions.InstanceId,
    AutoOffsetReset = AutoOffsetReset.Earliest,
    EnableAutoCommit = false,
};

using IConsumer<TKey, TValue> consumer = new ConsumerBuilder<TKey, TValue>(config)
    .SetKeyDeserializer(_keyDeserializer)
    .SetValueDeserializer(_valueDeserializer)
    .Build();
```

confluent kafka

consumer

```
consumer.Subscribe(_consumerOptions.Topic);

try
{
    while (cancellationToken.IsCancellationRequested is false)
    {
        ConsumeResult<TKey, TValue> result = consumer.Consume(cancellationToken);
        var message = new KafkaConsumerMessage<TKey, TValue>(consumer, result);
        await handler.HandleAsync(message, cancellationToken);
    }
}
finally
{
    consumer.Close();
}
```