Apache Kafka
От теории к практике

Григорий Кошелев
Ѣконтур

DotNEXT 2022

План

1. Введение в Apache Kafka
2. .NET Kafka Producer
3. .NET Kafka Consumer
4. Когда Apache Kafka 🔥
5. Выводы

Обо мне

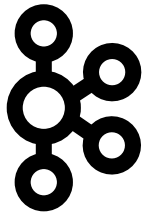
Кафковод — 4+ лет

— Когда всё пошло по Кафке
(JPoint 2019)

— Когда всё пошло по Кафке 2: разгоняем продьюсеров
(JPoint 2020)

— Как готовить Кафку, чтобы не пригорало
(DevOops 2019)

— Vostok Hercules: 3 года доставляем телеметрию — полёт нормальный
(JPoint 2022)



Кафка в Контуре

7 кластеров в проде

— Телеметрия

https://github.com/vostok/hercules

Кафка в Контуре

7 кластеров в проде

— Телеметрия
— **К**-архитектура

https://www.oreilly.com/radar/questioning-the-lambda-architecture/

Кафка в Контуре

7 кластеров в проде

— Телеметрия
— **К**-архитектура
— Шина данных

Кафка в Контуре

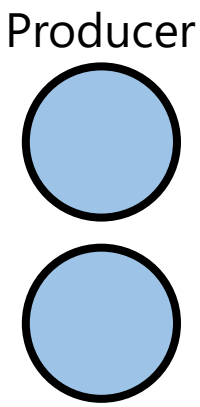
7 кластеров в проде

— Телеметрия
— **К**-архитектура
— Шина данных
— Интеграционная шина

Введение в Apache Kafka

Введение в Apache Kafka

Kafka Producer

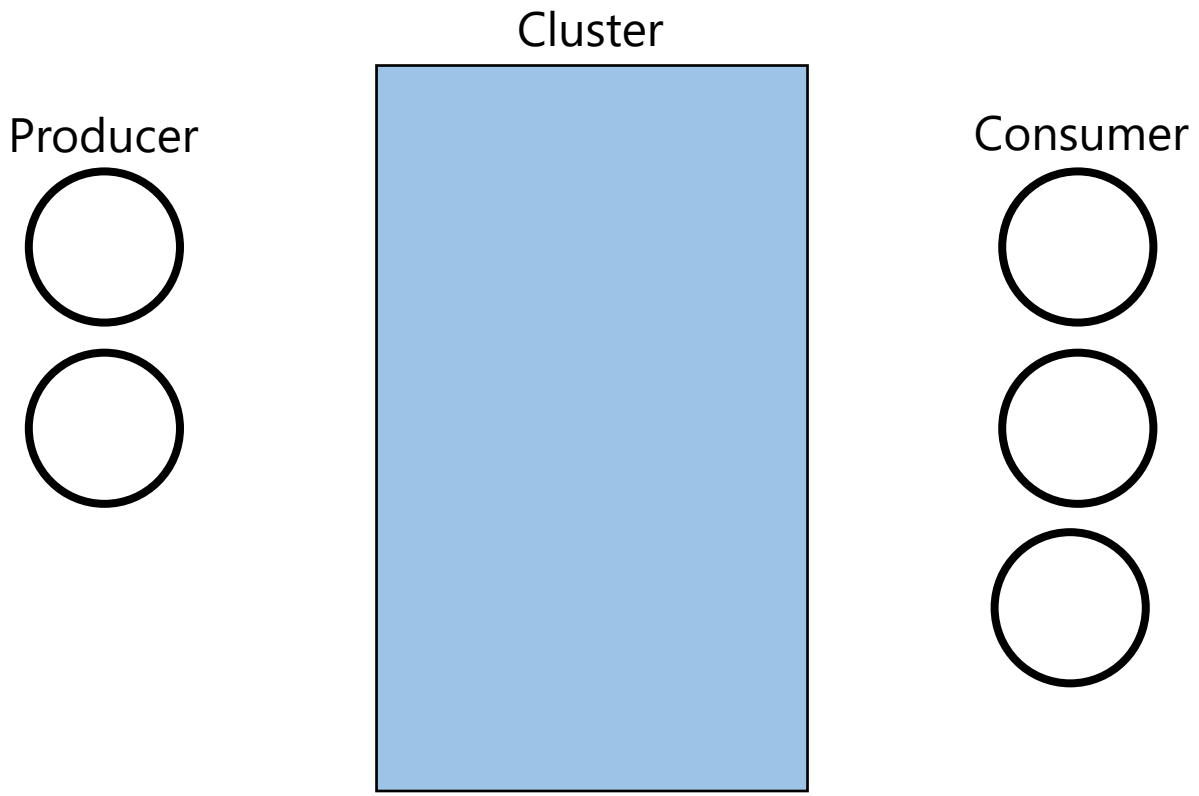


Введение в Apache Kafka

Kafka Consumer

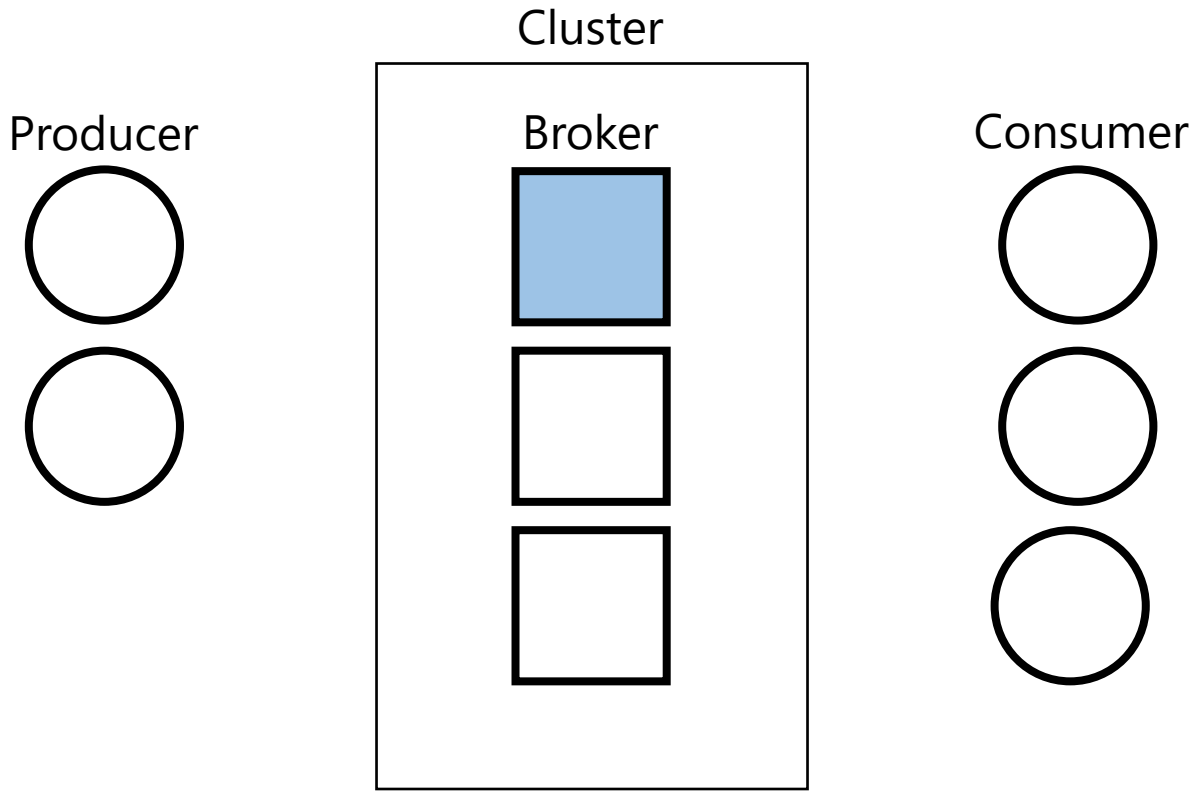
Введение в Apache Kafka

Kafka Cluster

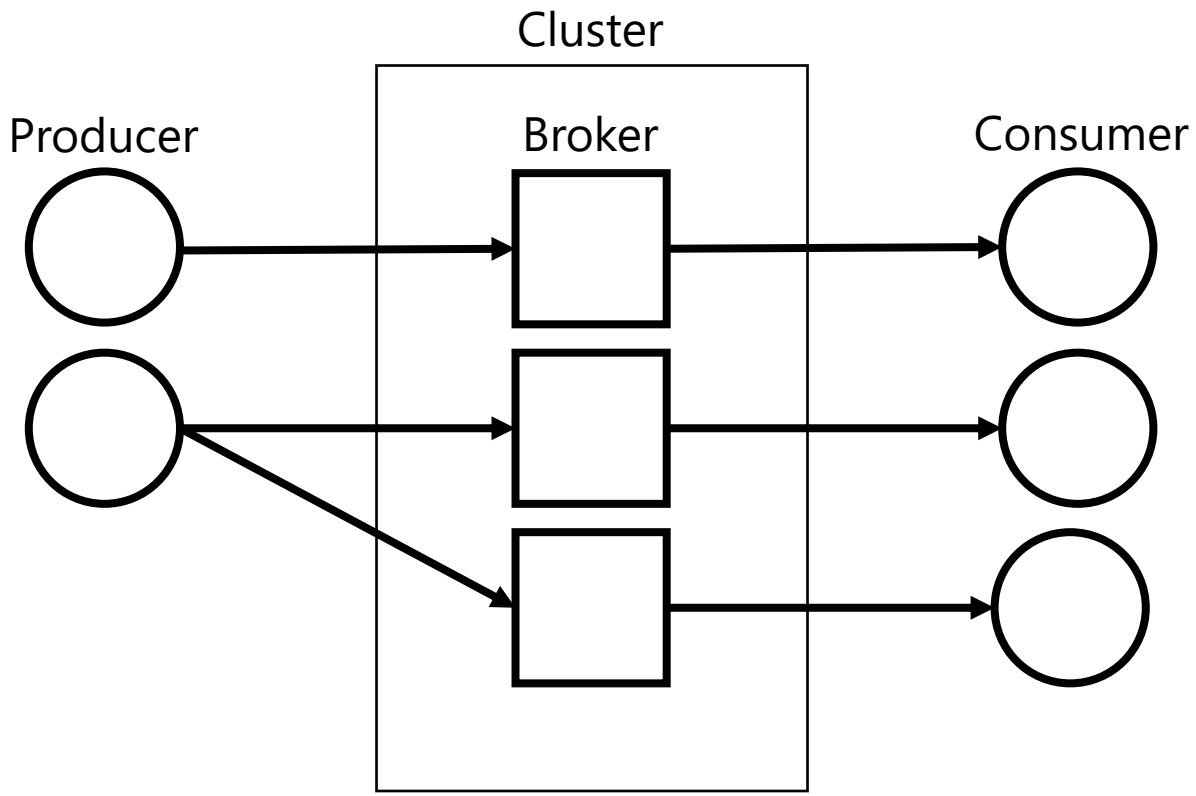


Введение в Apache Kafka

Kafka Broker

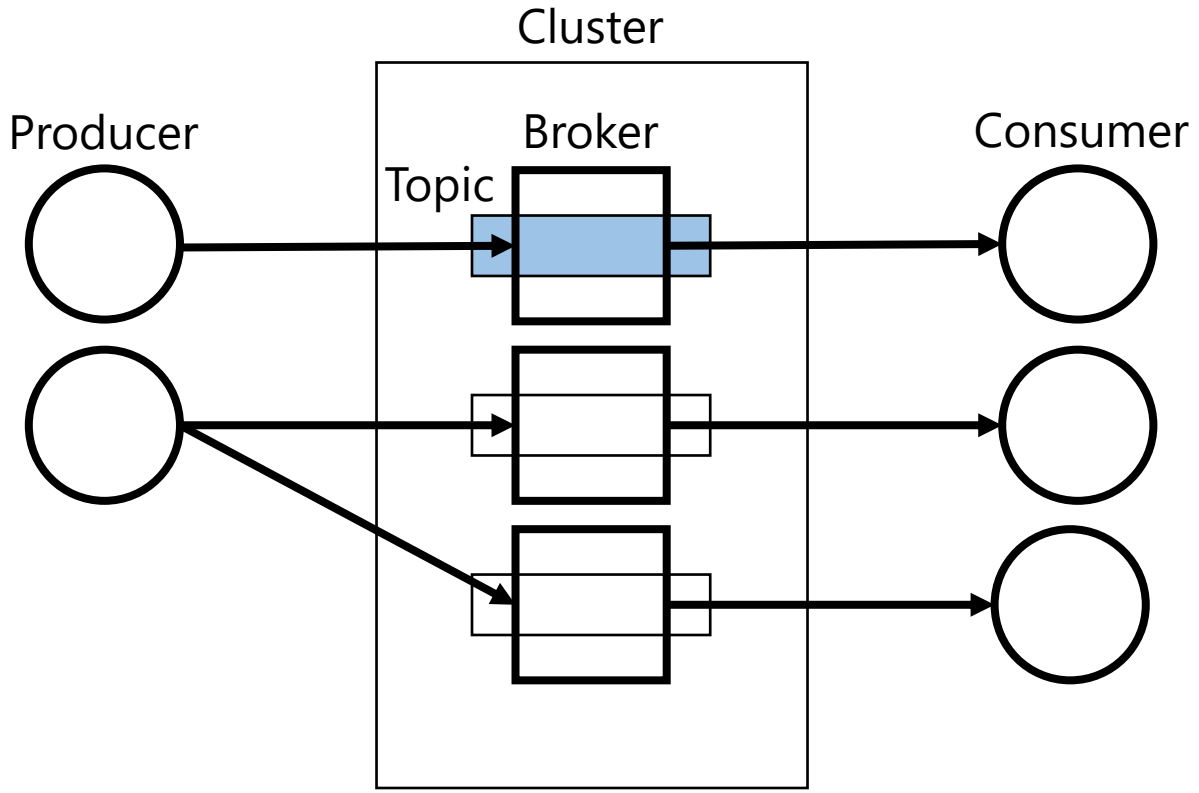


Введение в Apache Kafka



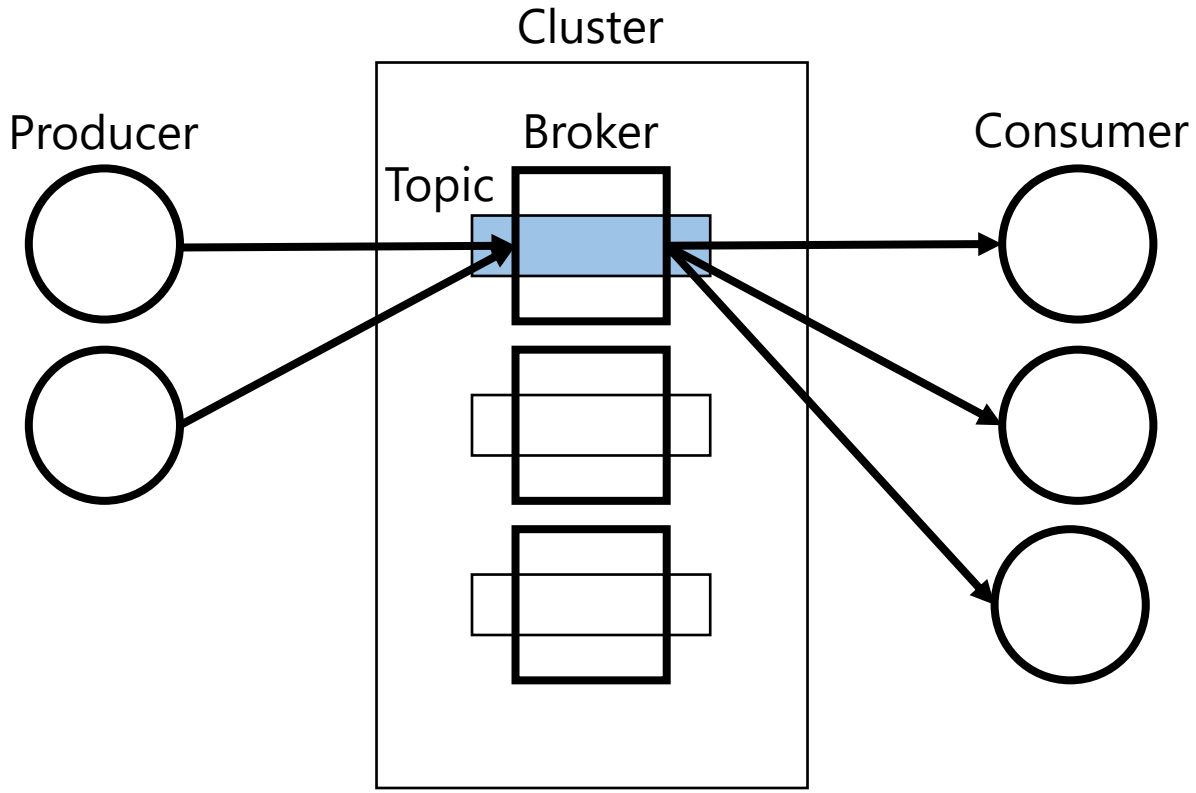
Введение в Apache Kafka

Kafka Topic



Введение в Apache Kafka

Pub-Sub с poll-механикой чтения

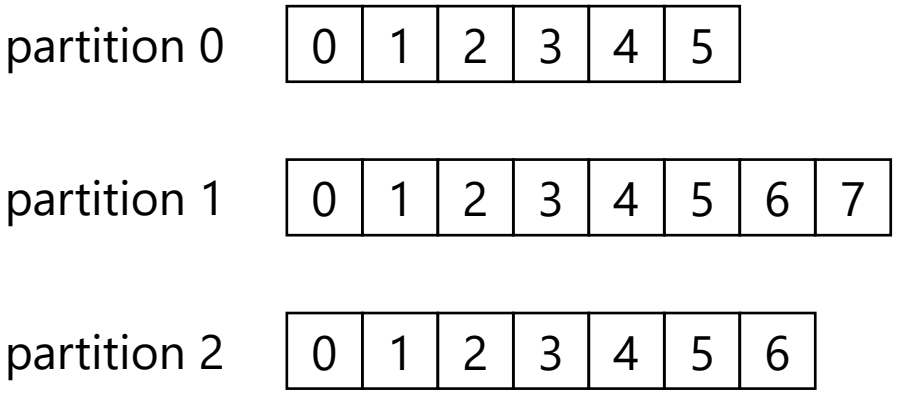


Архитектура Apache Kafka

— Topic
— Broker
— Producer
— Consumer

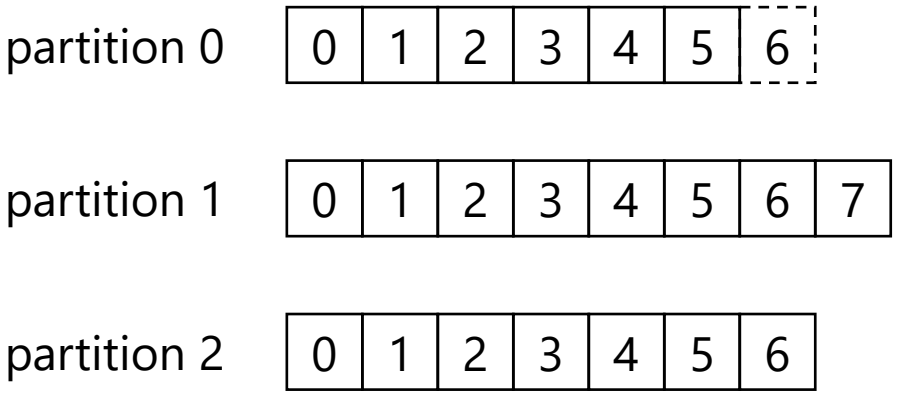
Архитектура Kafka Topic

topic = {partition}



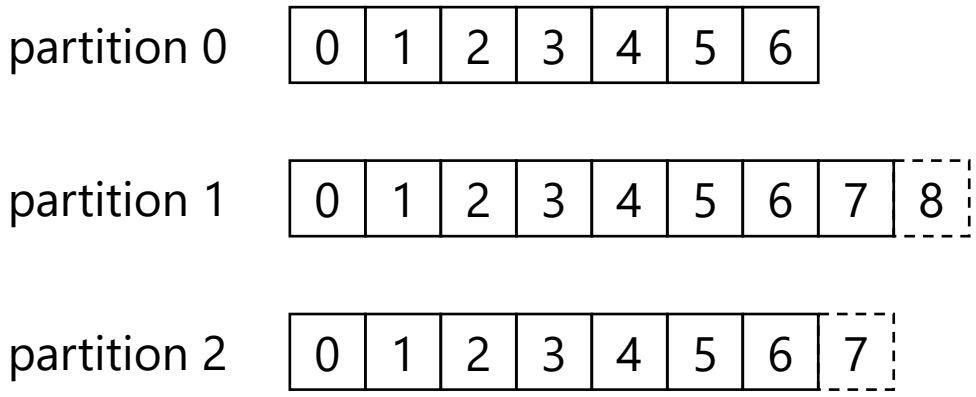
Архитектура Kafka Topic

topic = {partition}



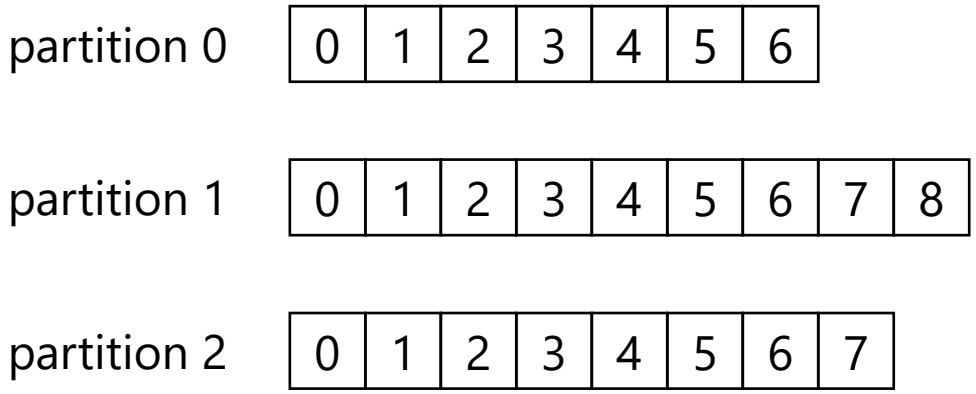
Архитектура Kafka Topic

topic = {partition}



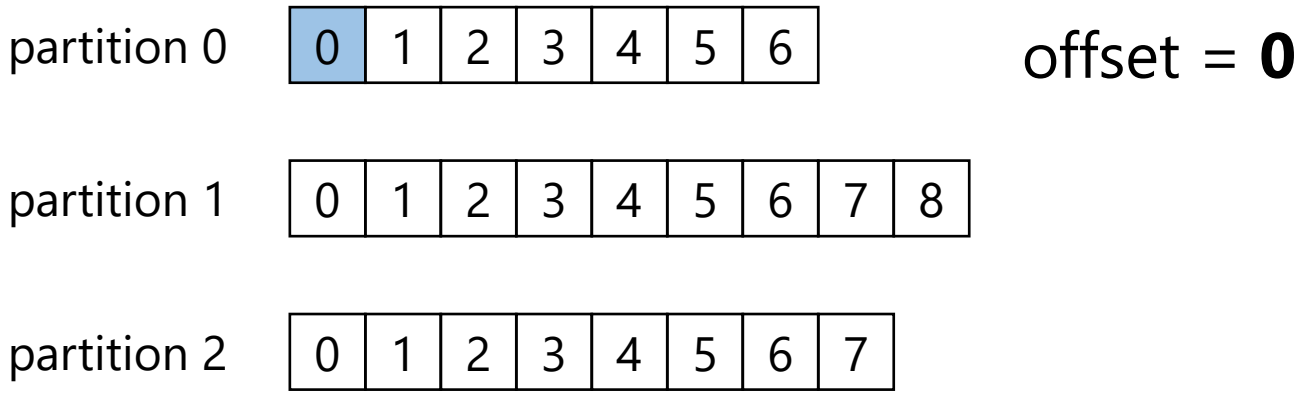
Архитектура Kafka Topic

topic = {partition}



Архитектура Kafka Topic

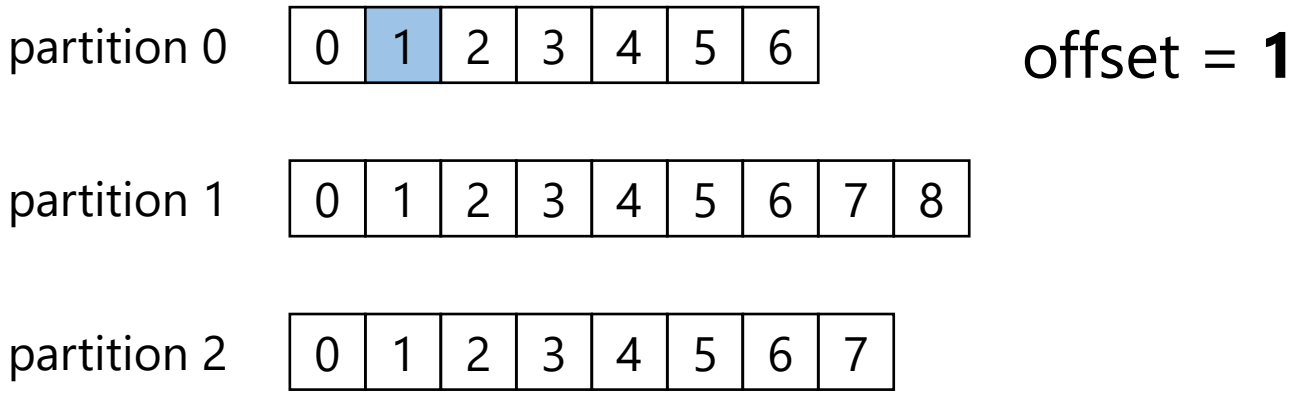
topic = {partition}



offset = **0**

Архитектура Kafka Topic

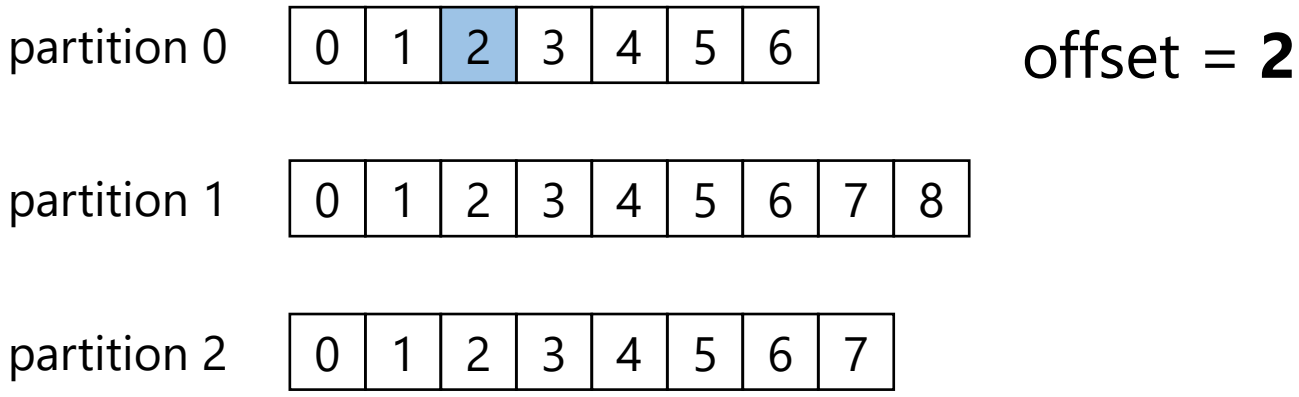
topic = {partition}



offset = **1**

Архитектура Kafka Topic

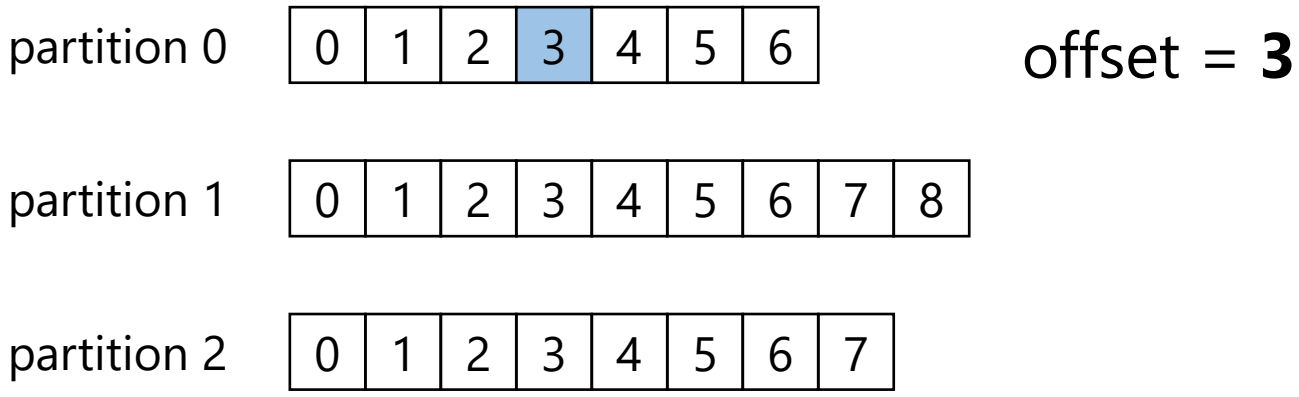
topic = {partition}



offset = **2**

Архитектура Kafka Topic

topic = {partition}



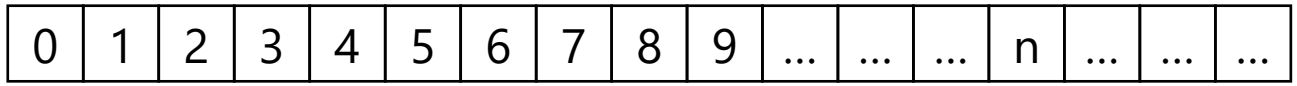
offset = **3**

Архитектура Kafka Topic

partition = {segment}

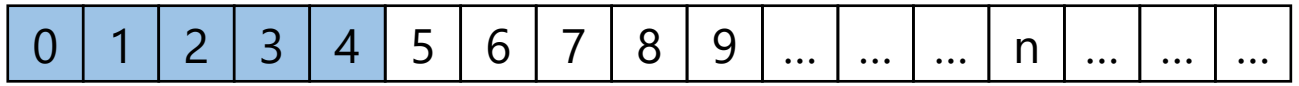
Архитектура Kafka Topic

partition = {segment}



Архитектура Kafka Topic

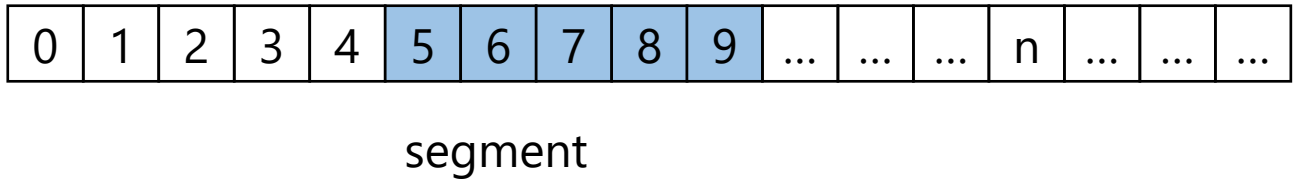
partition = {segment}



segment

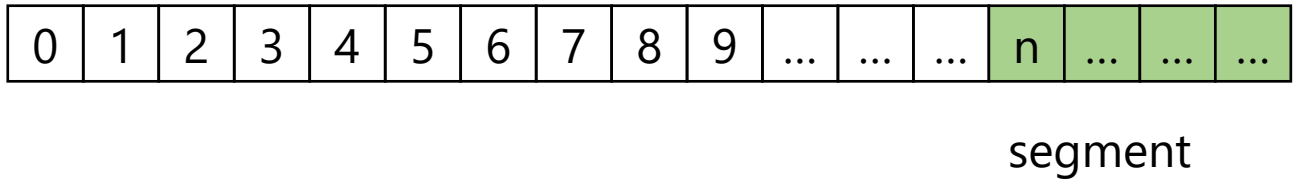
Архитектура Kafka Topic

partition = {segment}



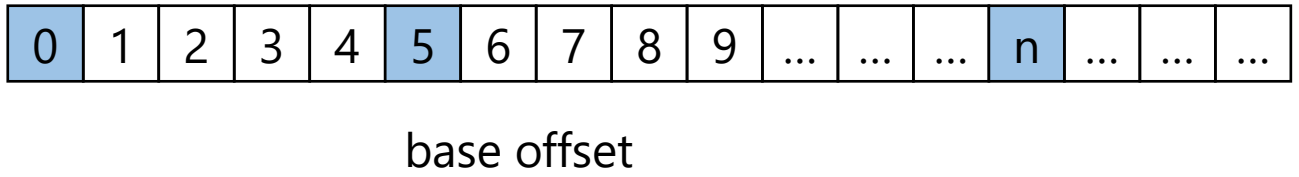
Архитектура Kafka Topic

partition = {segment}



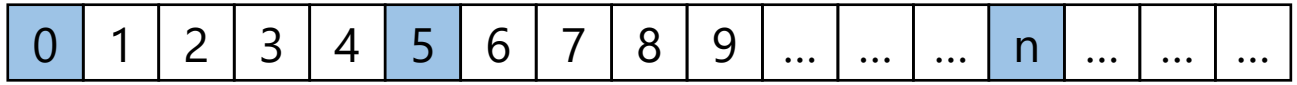
Архитектура Kafka Topic

partition = {segment}



Архитектура Kafka Topic

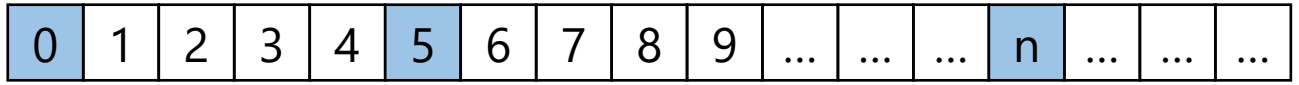
partition = {segment}



retention.bytes=-1
retention.ms=604800000

Архитектура Kafka Topic

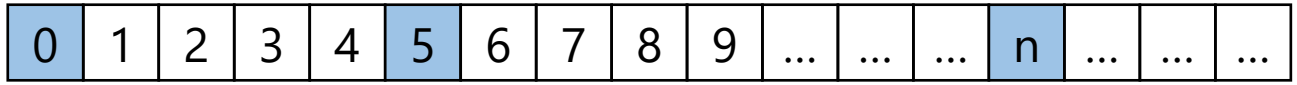
partition = {segment}



retention.bytes=-1
retention.ms=604800000

Архитектура Kafka Topic

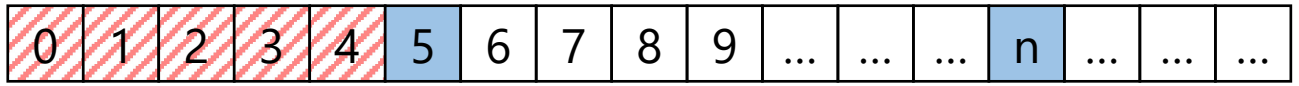
partition = {segment}



retention.bytes=-1
retention.ms=604800000

Архитектура Kafka Topic

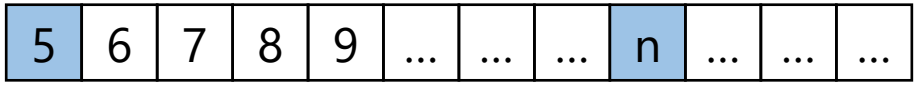
partition = {segment}



retention.bytes=-1
retention.ms=604800000

Архитектура Kafka Topic

partition = {segment}



retention.bytes=-1
retention.ms=604800000

Архитектура Kafka Topic

segment = (base_offset, data, index, timeindex)

```
00000000001234567890.log
00000000001234567890.index
00000000001234567890.timeindex
```

Архитектура Kafka Topic

segment = (**base_offset**, data, index, timeindex)

000000000**1234567890**.log
000000000**1234567890**.index
000000000**1234567890**.timeindex

Архитектура Kafka Topic

segment = (base_offset, **data**, index, timeindex)

00000000001234567890.log
00000000001234567890.index
00000000001234567890.timeindex

Архитектура Kafka Topic

segment = (base_offset, **data**, index, timeindex)

00000000001234567890.log
00000000001234567890.index
00000000001234567890.timeindex

log

Архитектура Kafka Topic

segment = (base_offset, data, **index**, timeindex)

```
00000000001234567890.log
00000000001234567890.index
00000000001234567890.timeindex
```

log

Index record = (relative offset, position)

Архитектура Kafka Topic

segment = (base_offset, data, **index**, timeindex)

```
00000000001234567890.log
00000000001234567890.index
00000000001234567890.timeindex
```

log

offset = 1234567890 relative offset = **0**

size = 100 position = **0**

Index record = (relative offset, position)

Архитектура Kafka Topic

segment = (base_offset, data, **index**, timeindex)

```
00000000001234567890.log
00000000001234567890.index
00000000001234567890.timeindex
```

log

offset = 1234567891 relative offset = **1**

size = 100 position = **100**

Index record = (relative offset, position)

Архитектура Kafka Topic

segment = (base_offset, data, **index**, timeindex)

```
00000000001234567890.log
00000000001234567890.index
00000000001234567890.timeindex
```

log

offset = 1234567892 relative offset = **2**

size = 50 position = **200**

Index record = (relative offset, position)

Архитектура Kafka Topic

segment = (base_offset, data, **index**, timeindex)

```
00000000001234567890.log
00000000001234567890.index
00000000001234567890.timeindex
```

log

offset = 1234567893 relative offset = **3**

size = 150 position = **250**

Index record = (relative offset, position)

Архитектура Kafka Topic

segment = (base_offset, data, index, **timeindex**)

```
00000000001234567890.log
00000000001234567890.index
00000000001234567890.timeindex
```

Архитектура Kafka Topic

Выводы

— Топик разбит на партиции

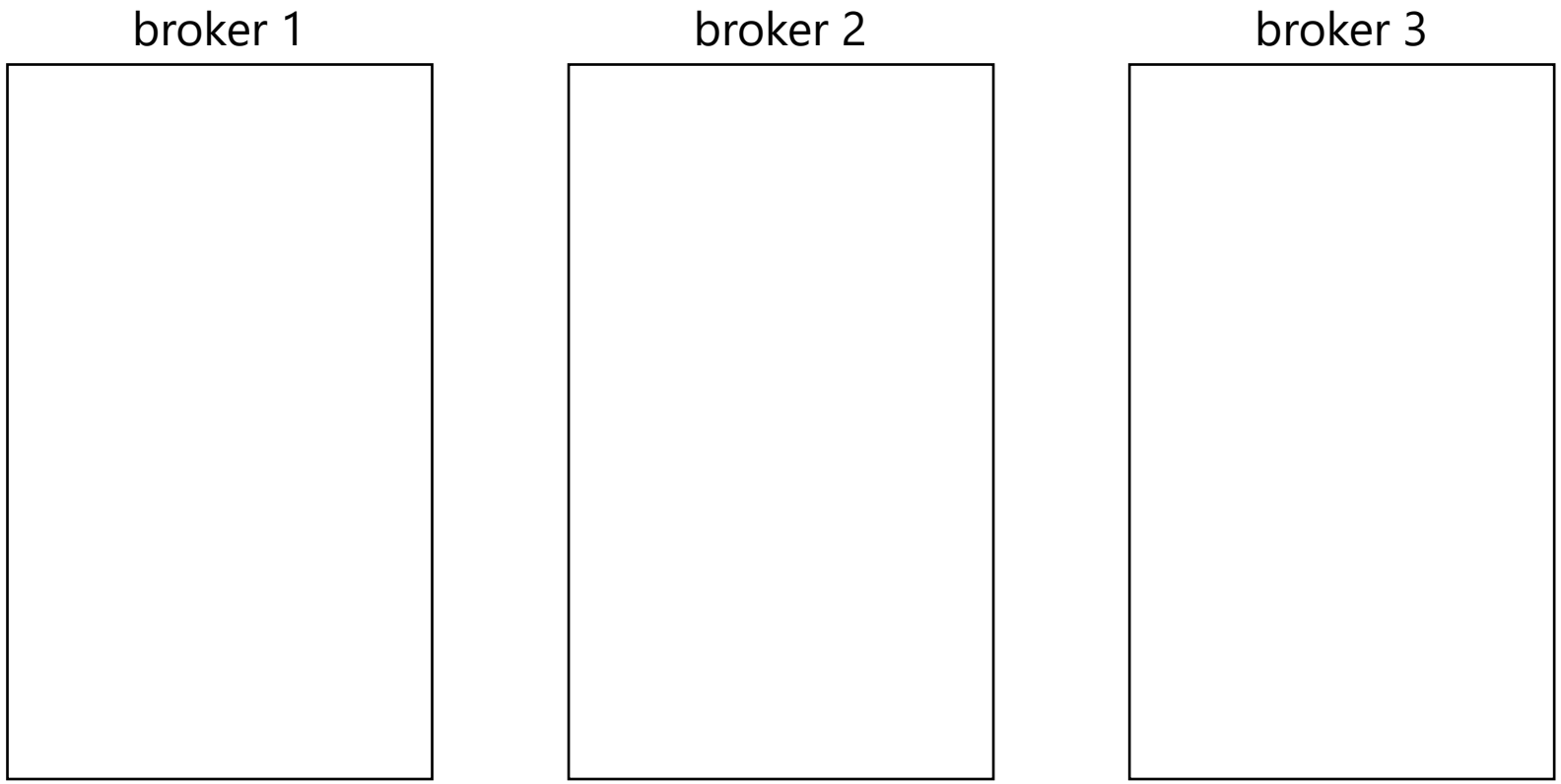
— Партиции хранятся на диске

— Данные удаляются либо по времени, либо по размеру

— Сообщение можно быстро найти по его Offset

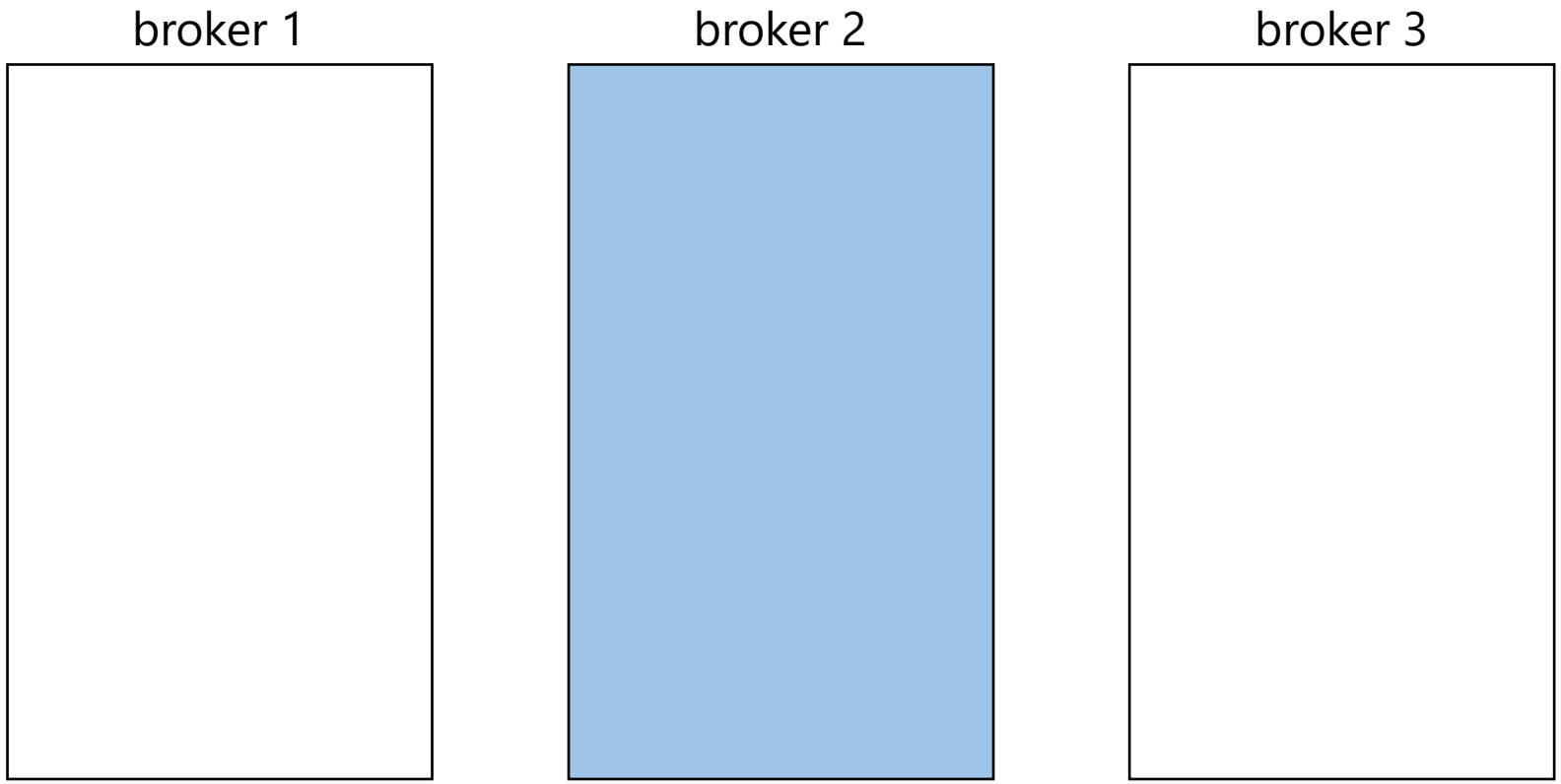
Архитектура Kafka Broker

cluster = {broker}



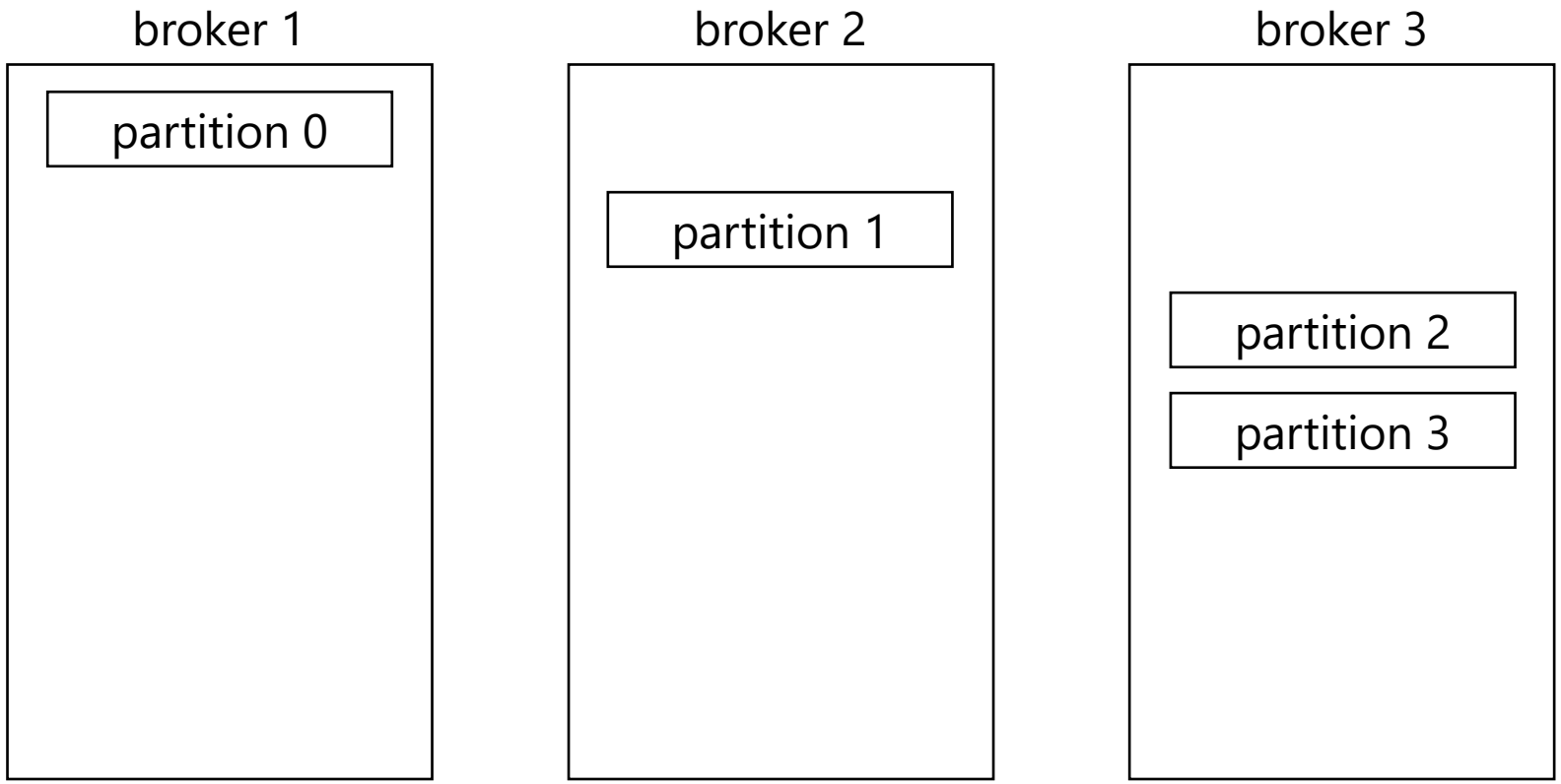
Архитектура Kafka Broker

Controller – координирует работу кластера



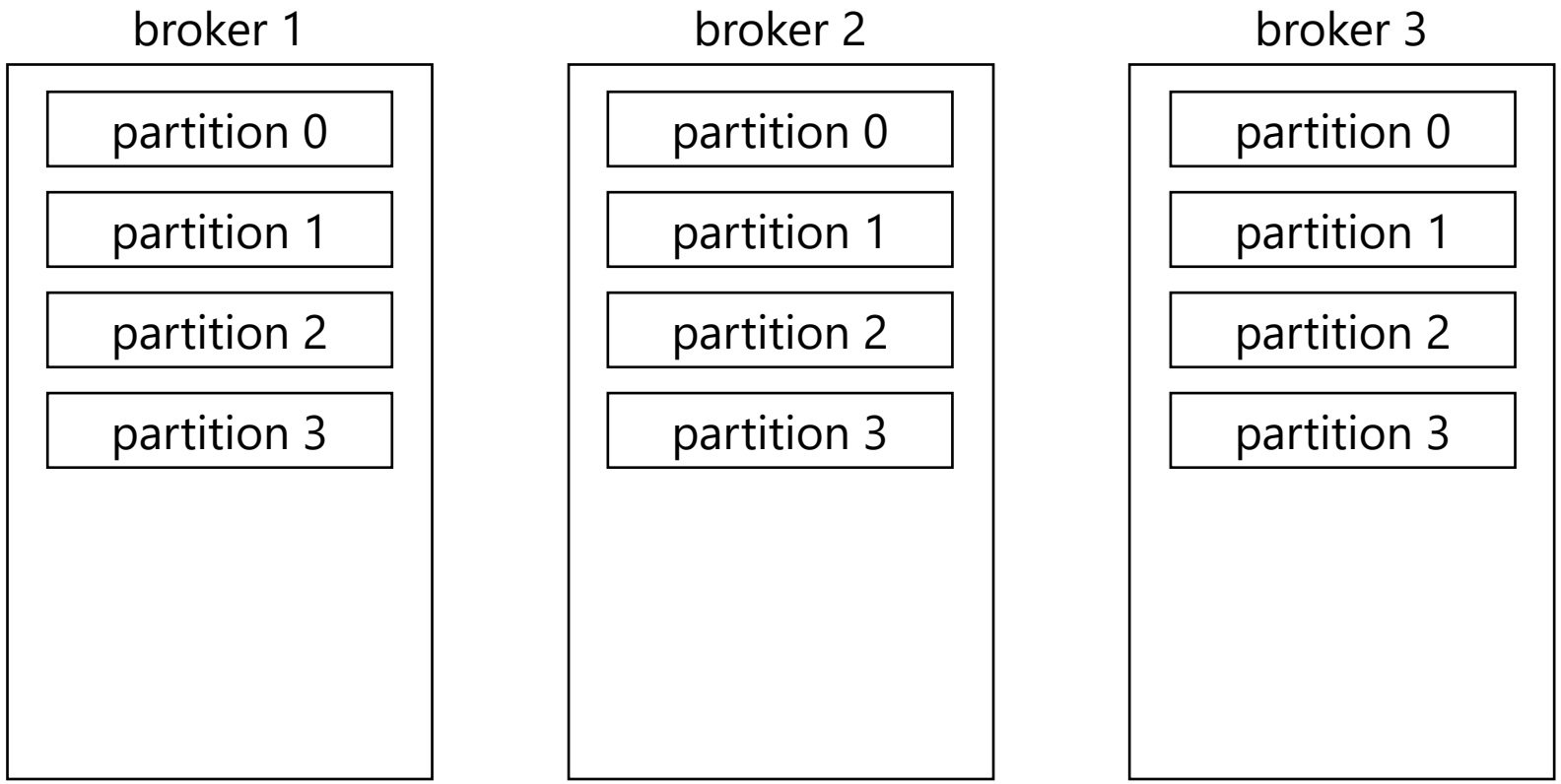
Архитектура Kafka Broker

topic = {partition}



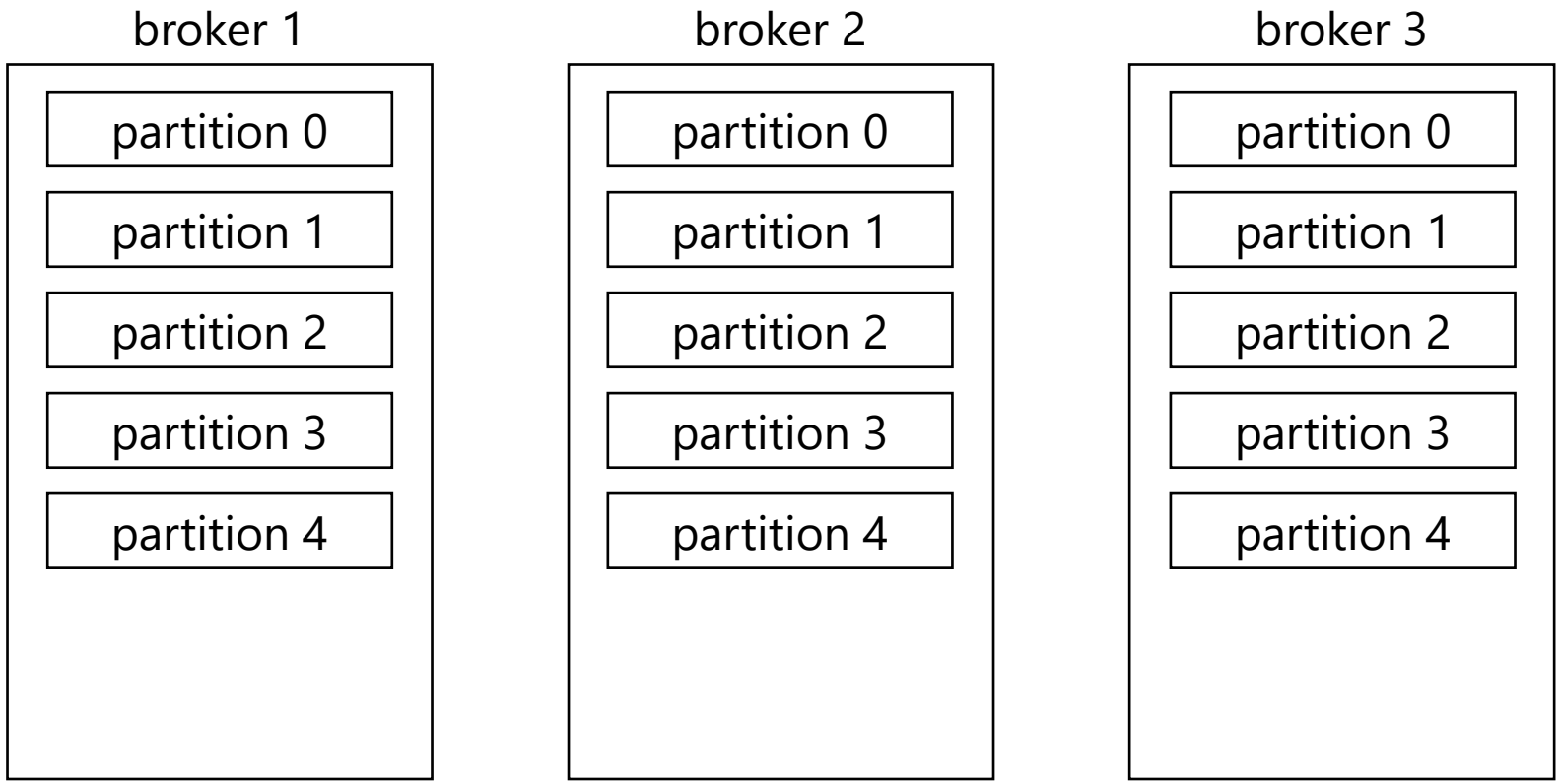
Архитектура Kafka Broker

replication factor = 3



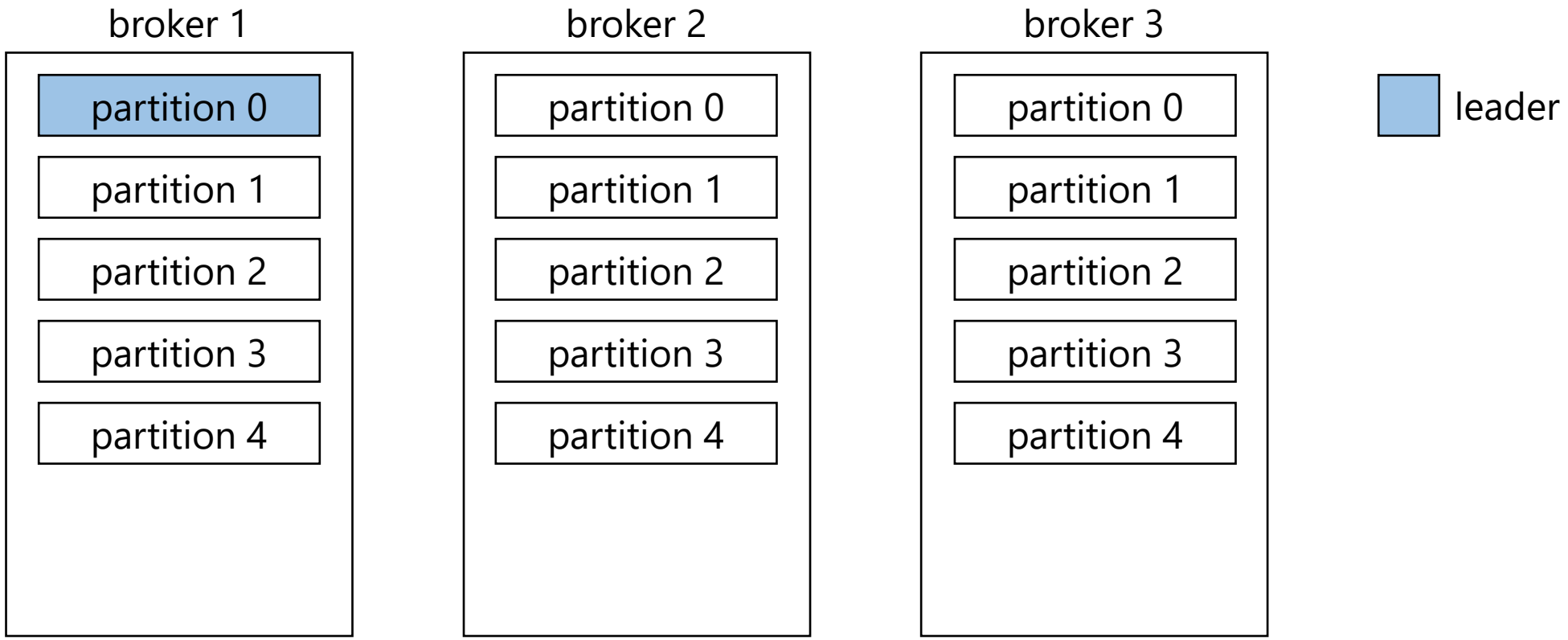
Архитектура Kafka Broker

Добавление partition



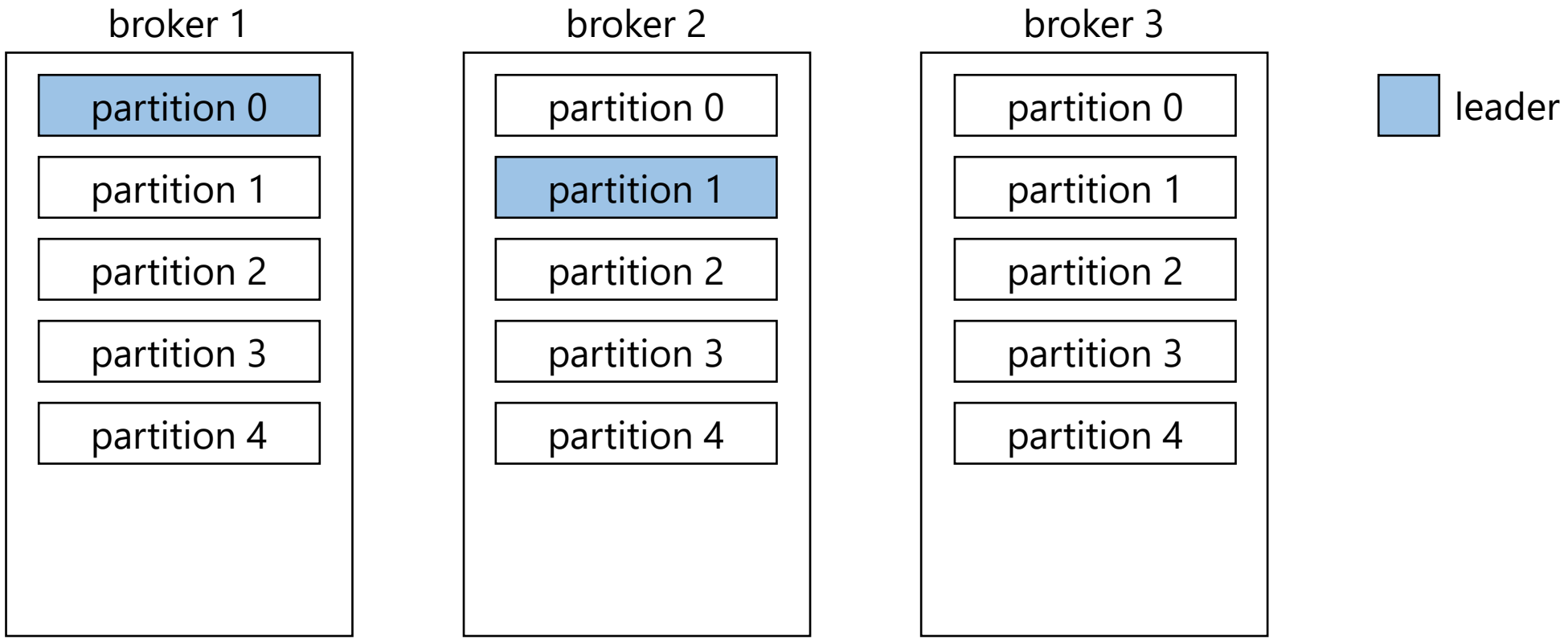
Архитектура Kafka Broker

broker 1 – leader для partition 0.



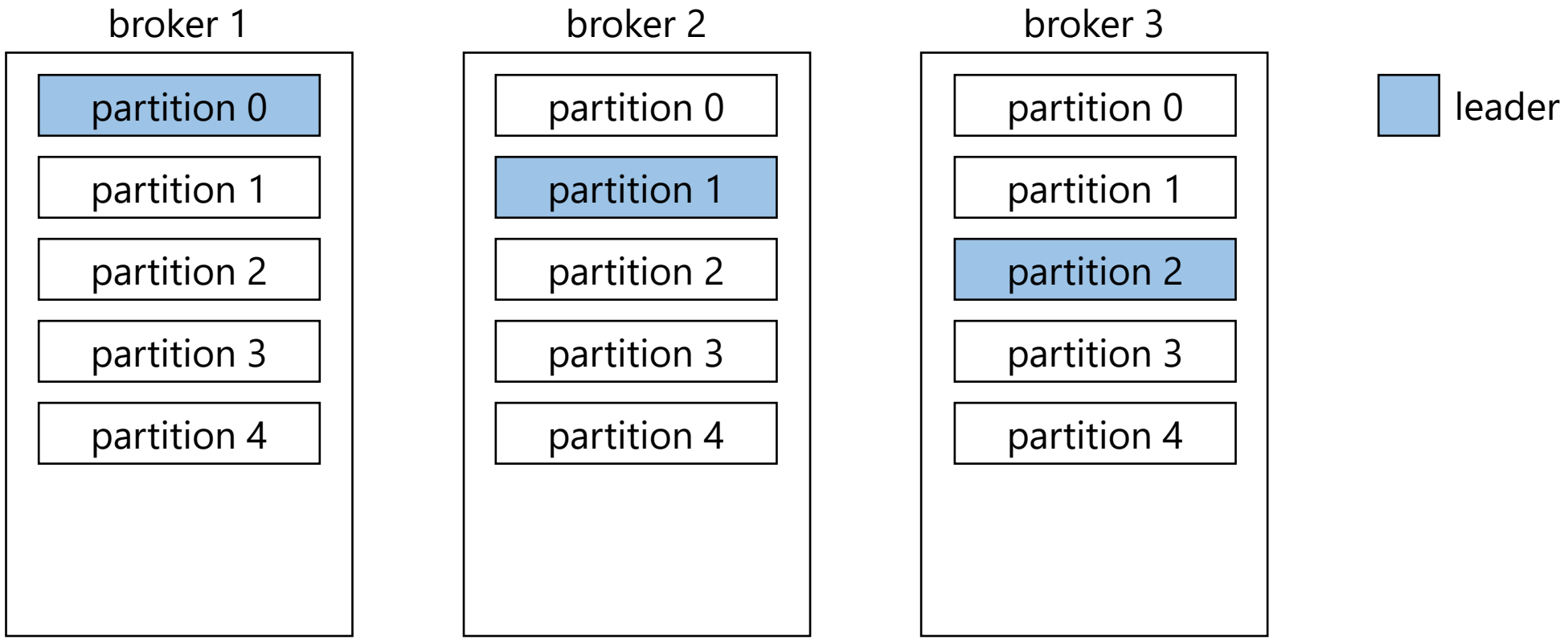
Архитектура Kafka Broker

broker 2 – leader для partition 1



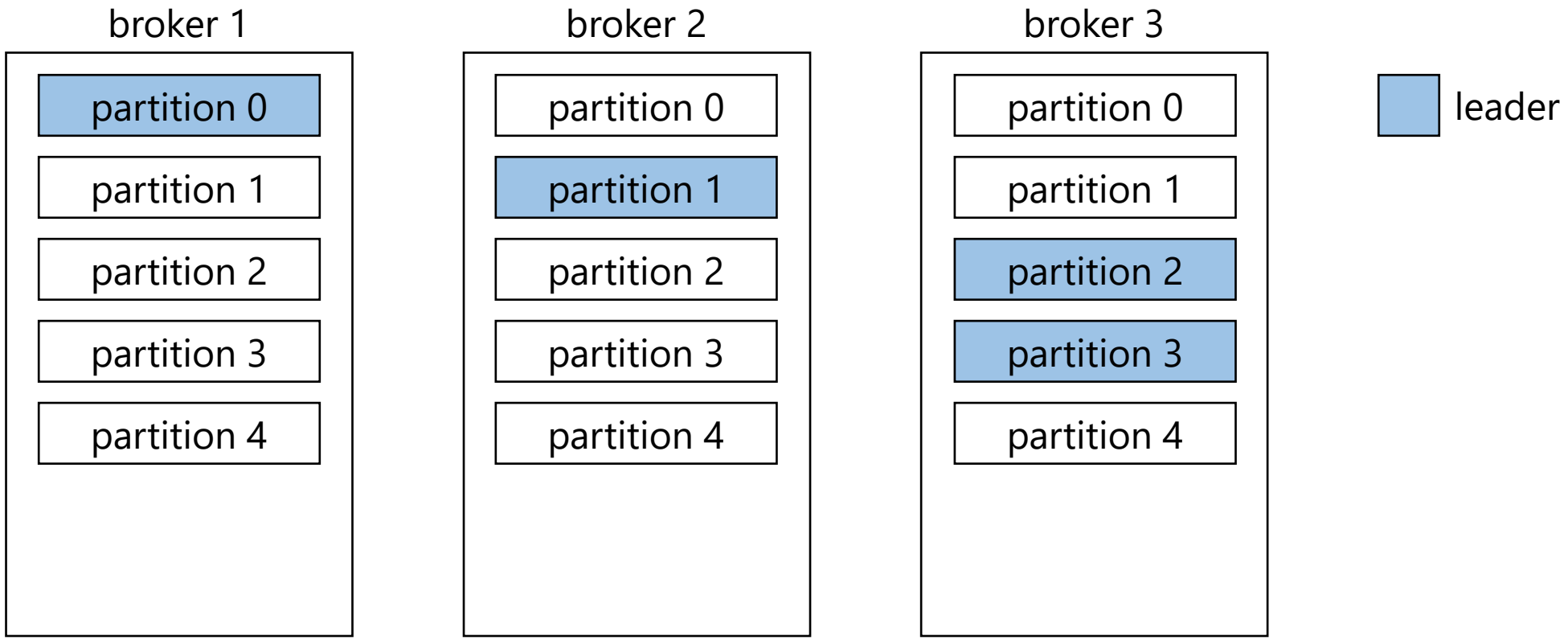
Архитектура Kafka Broker

broker 3 – leader для partition 2



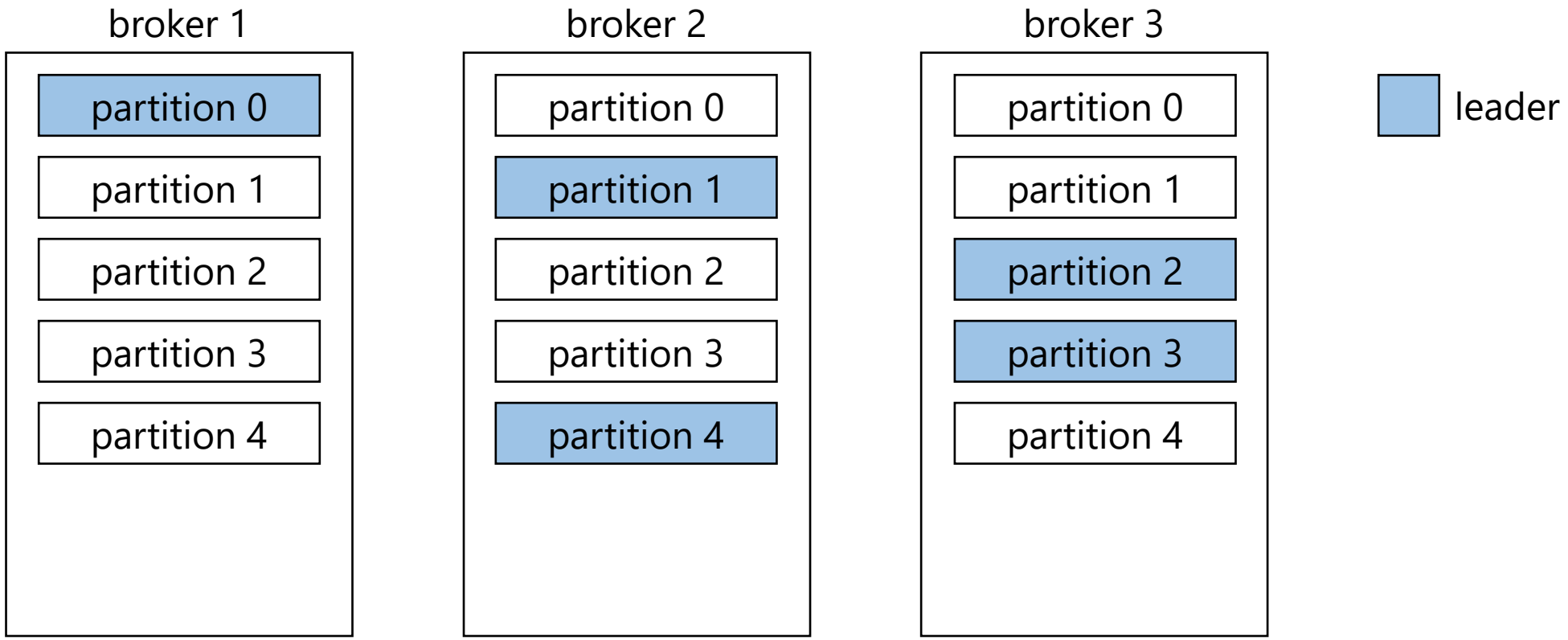
Архитектура Kafka Broker

broker 3 – leader для partition 3



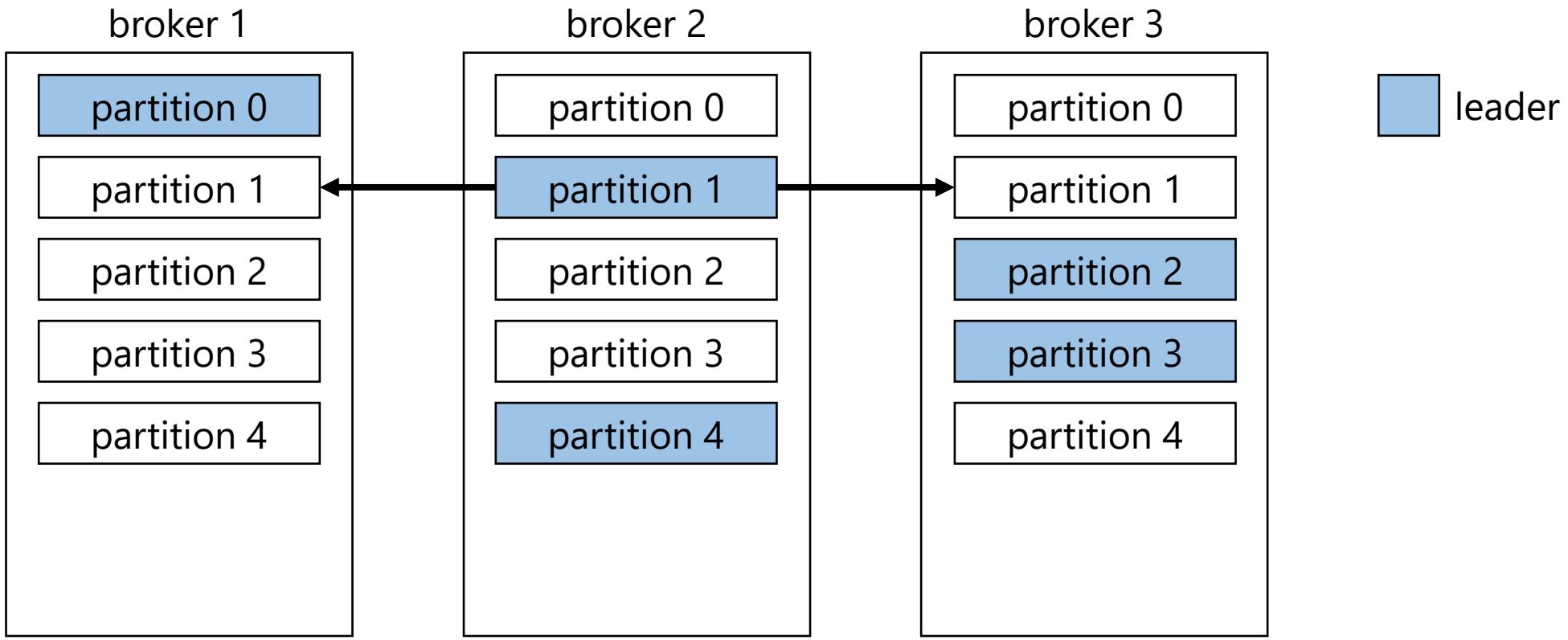
Архитектура Kafka Broker

broker 2 – leader для partition 4



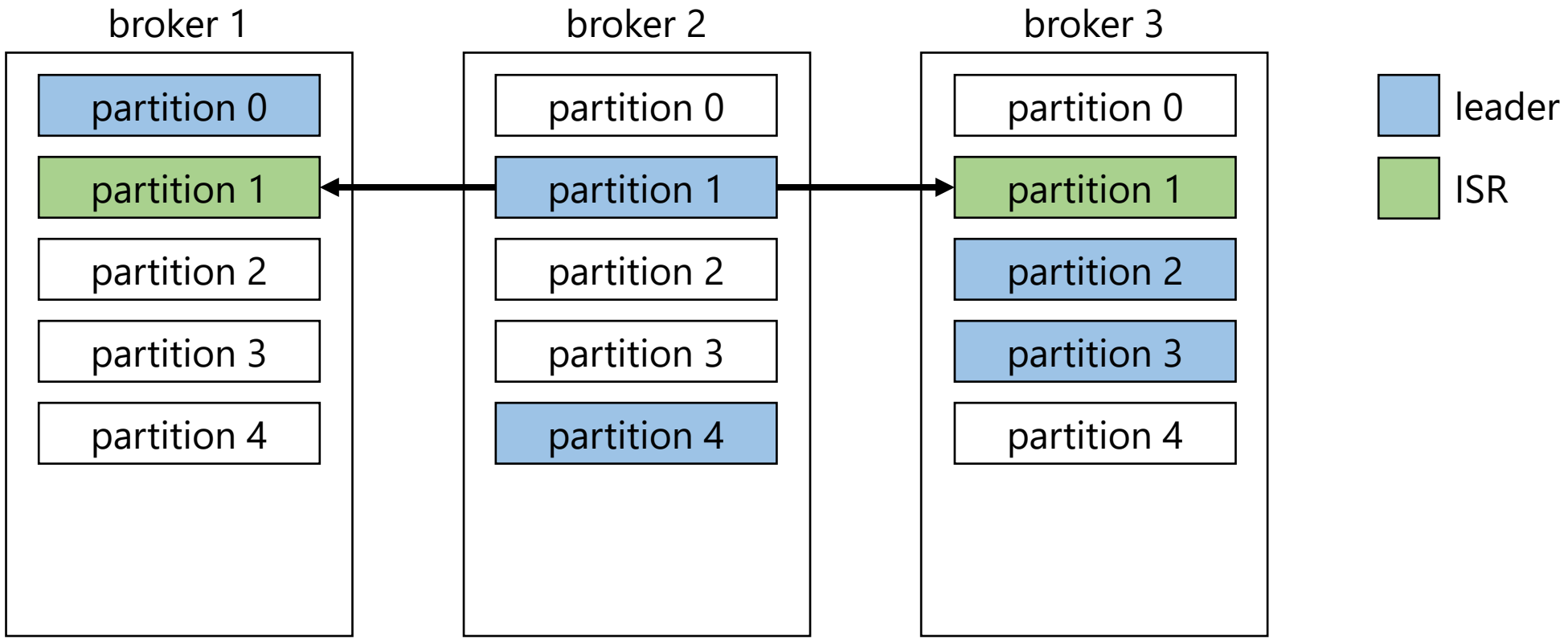
Архитектура Kafka Broker

Репликация с *лидера* на другие *брокеры* (*фолловеры*)



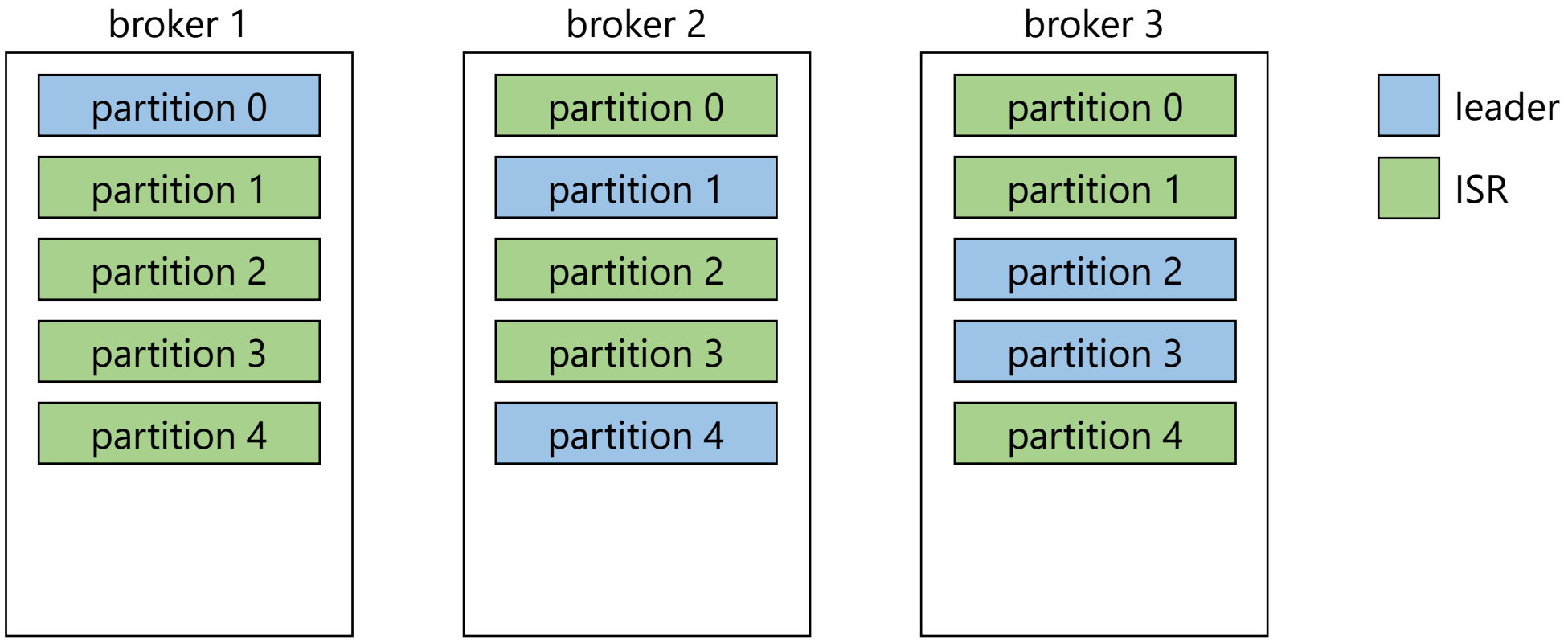
Архитектура Kafka Broker

ISR (in sync replica) – *реплика*, синхронизированная с *лидером*



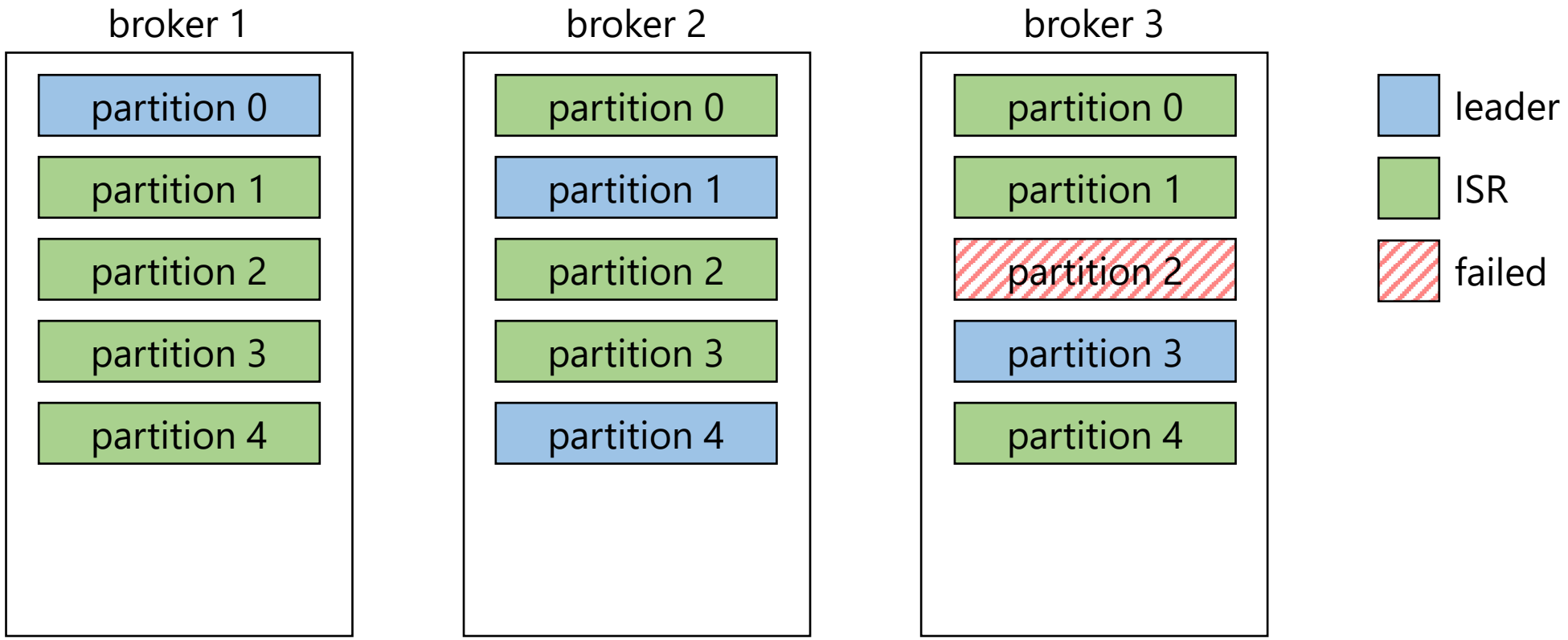
Архитектура Kafka Broker

Все реплики синхронизированы



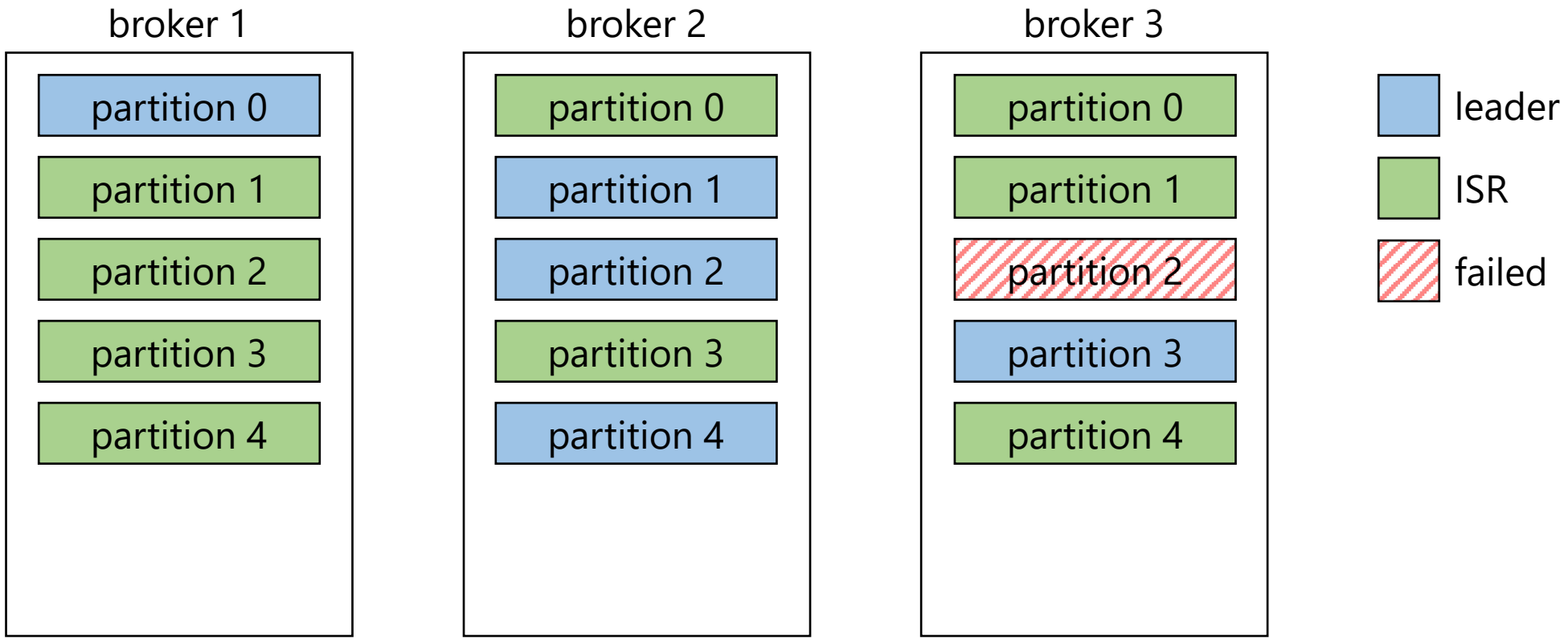
Архитектура Kafka Broker

Недоступность *лидера* у partition 2



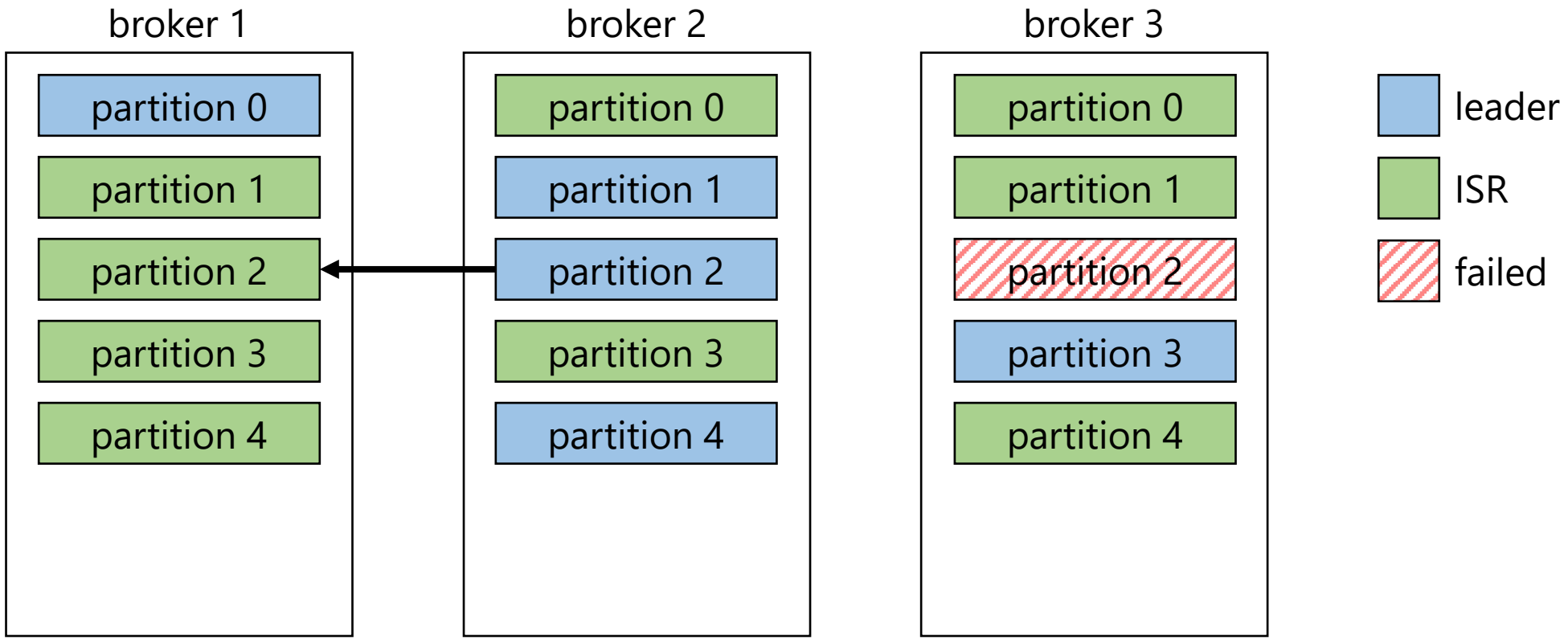
Архитектура Kafka Broker

Выбор нового *лидера* в случае недоступности

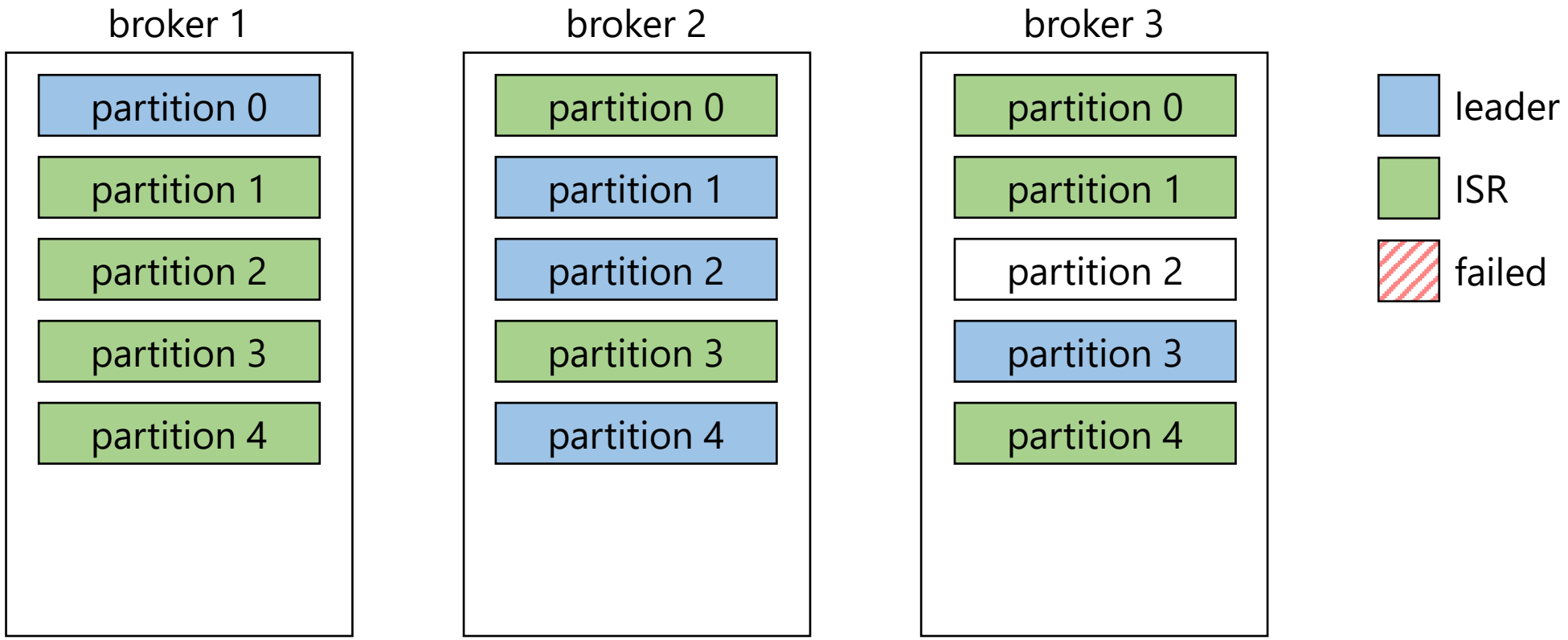


Архитектура Kafka Broker

Репликация с нового *лидера*

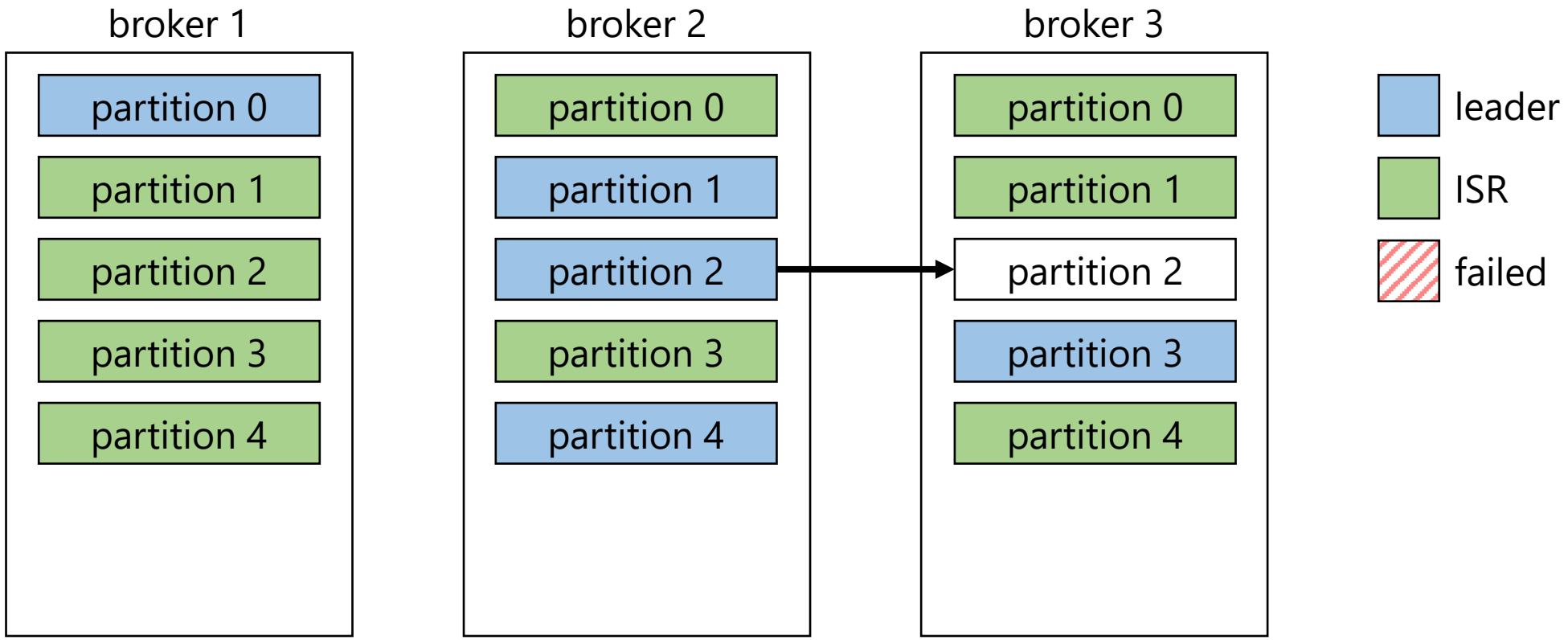


Архитектура Kafka Broker

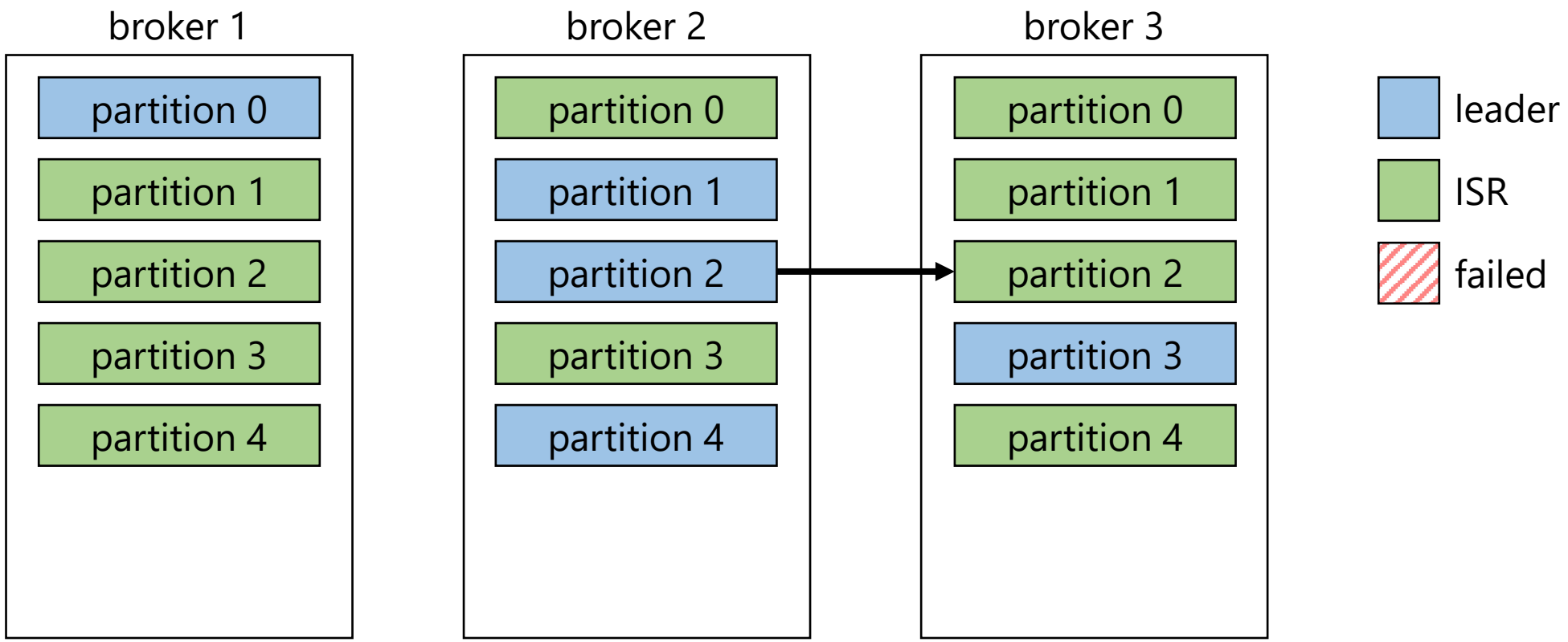


Архитектура Kafka Broker

Синхронизация реплики с *лидером* после восстановления

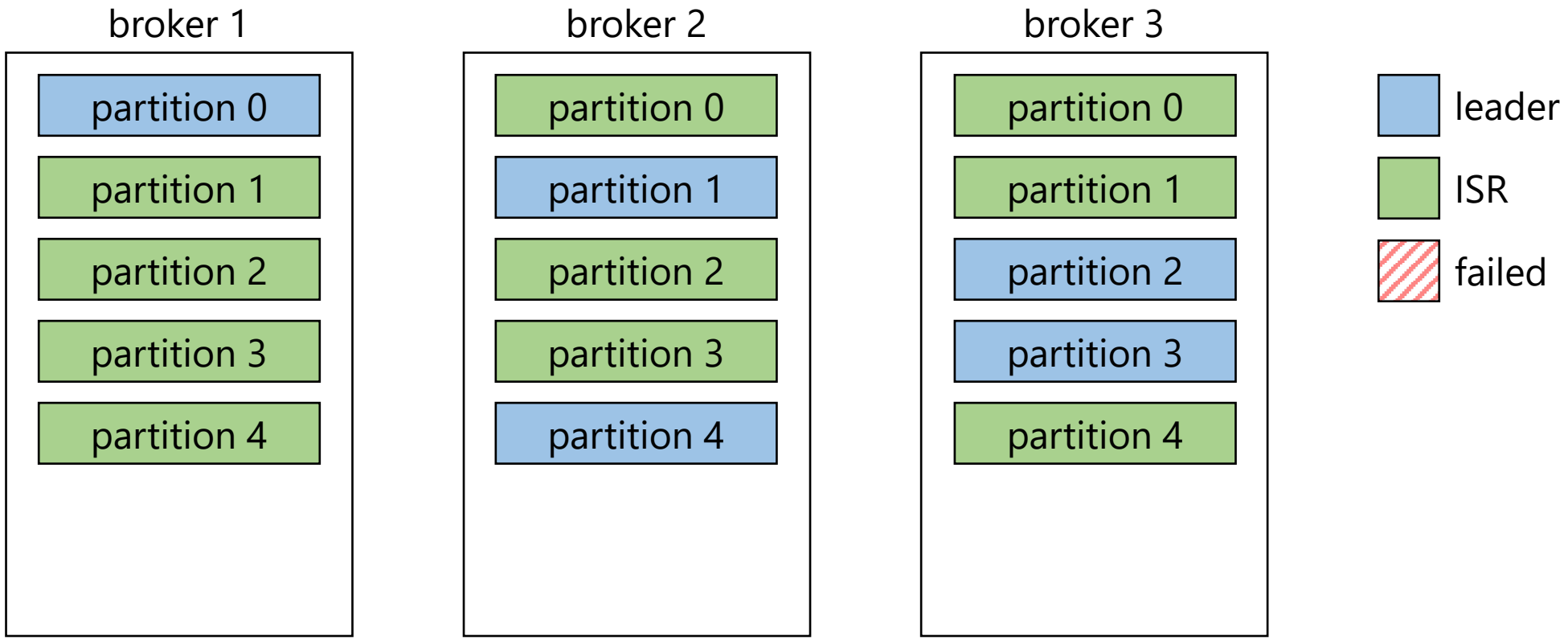


Архитектура Kafka Broker



Архитектура Kafka Broker

Перебалансировка *лидеров*



Архитектура Kafka Broker

Выводы

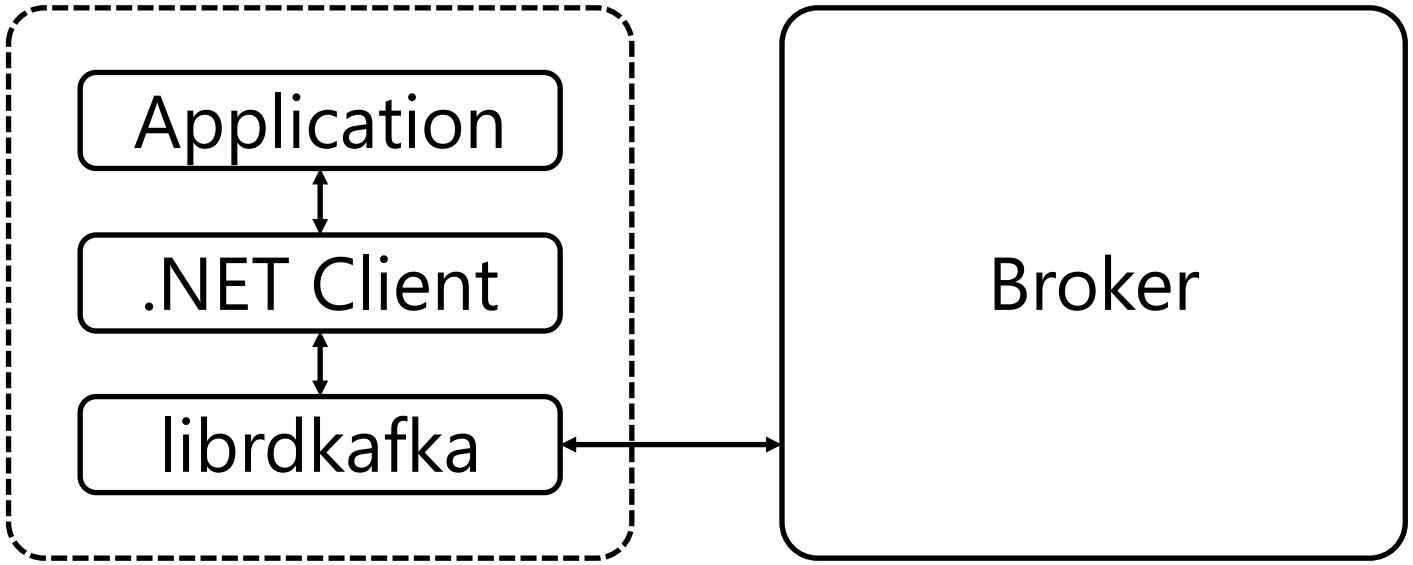
— У каждой партиции свой Лидер

— Сообщения пишутся в Лидера

— Данные реплицируются между брокерами

— Автоматический фейловер лидерства

.NET Kafka Client



.NET Kafka Client

Версия 1.8.2

— .NET Framework 4.5.1+

— .NET Core 1.0+

— .NET Standard 1.3+

.NET Kafka Client

Версия 1.8.2

— .NET Framework 4.5.1+

— .NET Core 1.0+

— .NET Standard 1.3+

Все примеры для доклада запускались на .NET 6.0 (Mac OS)

.NET Kafka Producer

.NET Kafka Producer

```
new Message<TKey, TValue> {
    Key = key,
    Value = value
}
```

.NET Kafka Producer

```
new Message<TKey, TValue> {
    Key = key,
    Value = value
}
```

.NET Kafka Producer

```
new Message<TKey, TValue> {
    Key = key,
    Value = value
}
```

.NET Kafka Producer

```
new Message<Guid, TValue> {
    Key = guid,
    Value = value
}
```

.NET Kafka Producer

```
new Message<Null, TValue> {
    Value = value
}
```

.NET Kafka Producer

```
new Message<Null, TValue> {
    Value = value
}
```

.NET Kafka Producer

```
public enum Partitioner
{
    Random,
    Consistent,
    ConsistentRandom,
    Murmur2,
    Murmur2Random
}
```

.NET Kafka Producer

```
public enum Partitioner
{
    Random,
    Consistent,
    ConsistentRandom,
    Murmur2,
    Murmur2Random
}
```

.NET Kafka Producer

```
public enum Partitioner
{
    Random,
    Consistent,
    ConsistentRandom,
    Murmur2,
    Murmur2Random
}
```

CRC32(Message.Key) → партиция

.NET Kafka Producer

```
public enum Partitioner
{
    Random,
    Consistent,
    ConsistentRandom,
    Murmur2,
    Murmur2Random
}
```

Все сообщения с Key == null распределяются по партициям равномерно

.NET Kafka Producer

```
public enum Partitioner
{
    Random,
    Consistent,
    ConsistentRandom,
    Murmur2,
    Murmur2Random
}
```

.NET Kafka Producer

```
public enum Partitioner
{
    Random,
    Consistent,
    ConsistentRandom,
    Murmur2,
    Murmur2Random
}
```

Все сообщения с Key == null попадут в одну партицию!

.NET Kafka Producer

```
public enum Partitioner
{
    Random,
    Consistent,
    ConsistentRandom,
    Murmur2,
    Murmur2Random
}
```

.NET Kafka Producer

```
public enum Partitioner
{
    Random,
    Consistent,
    ConsistentRandom,
    Murmur2,
    Murmur2Random
}
```

Алгоритм MurMur2
для совместимости с Java

.NET Kafka Producer

```
public enum Partitioner
{
    Random,
    Consistent,
    ConsistentRandom,
    Murmur2,
    Murmur2Random
}
```

.NET Kafka Producer

```
public enum Partitioner
{
    Random,
    Consistent,
    ConsistentRandom,
    Murmur2,
    Murmur2Random
}
```

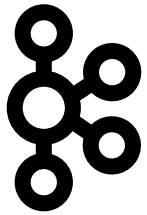
Все сообщения равномерно распределяются по партициям

.NET Kafka Producer

```
// ...
await producer.ProduceAsync(
        topic,
        new Message<Null, string> {
            Value = value
        });
// ...
```

.NET Kafka Producer

```
// ...
await producer.ProduceAsync(
        topic,
        new Message<Null, string> {
            Value = value
        });
// ...
```



.NET Kafka Producer

```
// ...
var task = producer.ProduceAsync(
        topic,
        new Message<Null, string> {
            Value = value
        });
```

.NET Kafka Producer

```
task.ContinueWith(task =>
{
    if (task.IsFaulted)
        Console.WriteLine("Failed");
    else
    {
        // ...
    }
});
```

.NET Kafka Producer

```
// ...
producer.Produce(
    topic,
    new Message<Null, string> {
        Value = value
    },
    callback);
```

.NET Kafka Producer

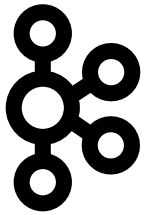
```
// ...
producer.Produce(
    topic,
    new Message<Null, string> {
        Value = value
    },
    callback);
```

.NET Kafka Producer

```
private static void callback(
                DeliveryReport<Null, string> result)
{
    if (result.Error.IsError)
        Console.WriteLine("Failed");
    else
    {
        // ...
    }
}
```

.NET Kafka Producer

ProduceAsync vs Produce



.NET Kafka Producer

.NET Kafka Producer

```
public enum Acks : int
{
    None = 0,
    Leader = 1,
    All = -1
}
```

.NET Kafka Producer

```
public enum Acks : int
{
    None = 0,
    Leader = 1,
    All = -1
}
```

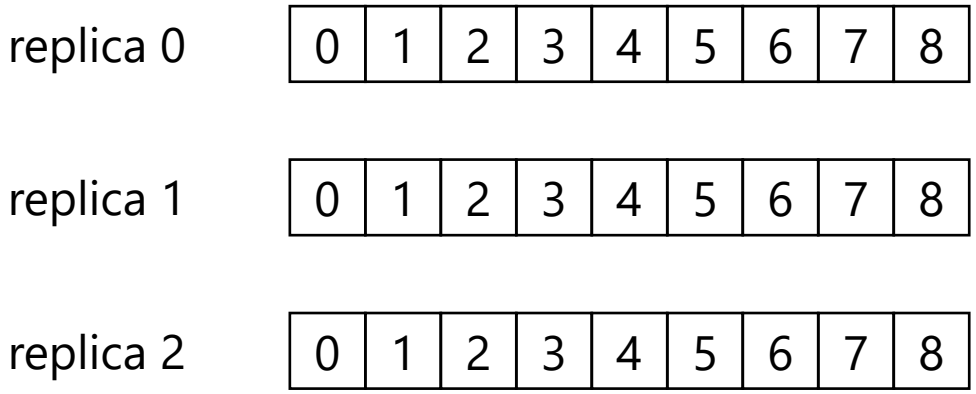
.NET Kafka Producer

```
public enum Acks : int
{
    None = 0,
    Leader = 1,
    All = -1
}
```

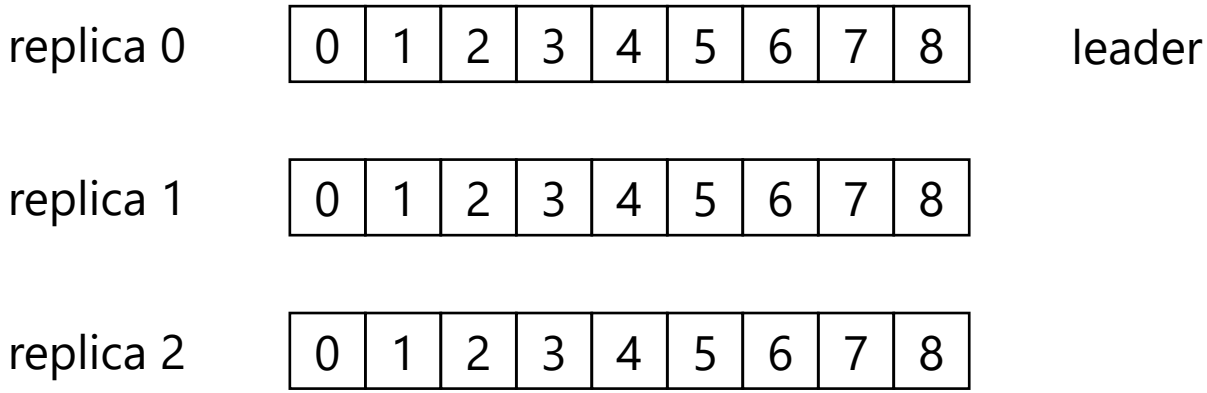
.NET Kafka Producer

```
public enum Acks : int
{
    None = 0,
    Leader = 1,
    All = -1
}
```

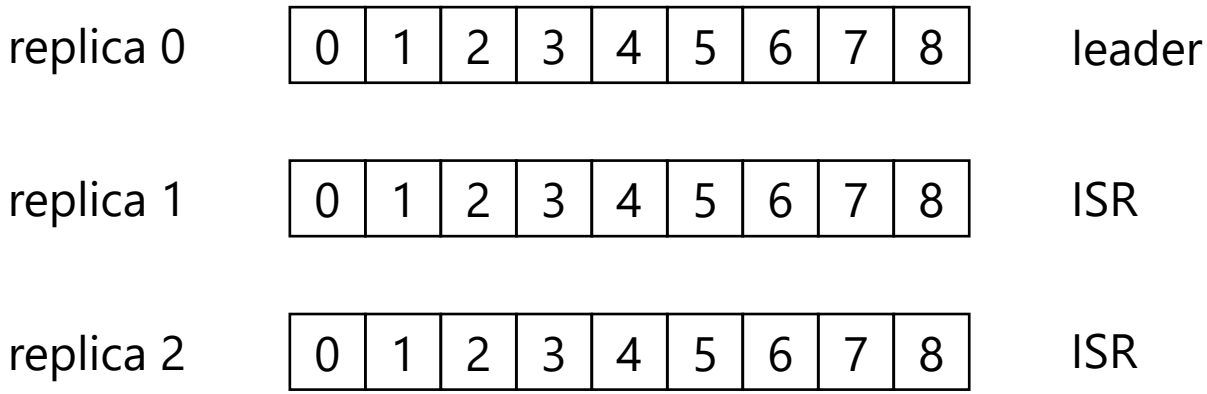
.NET Kafka Producer



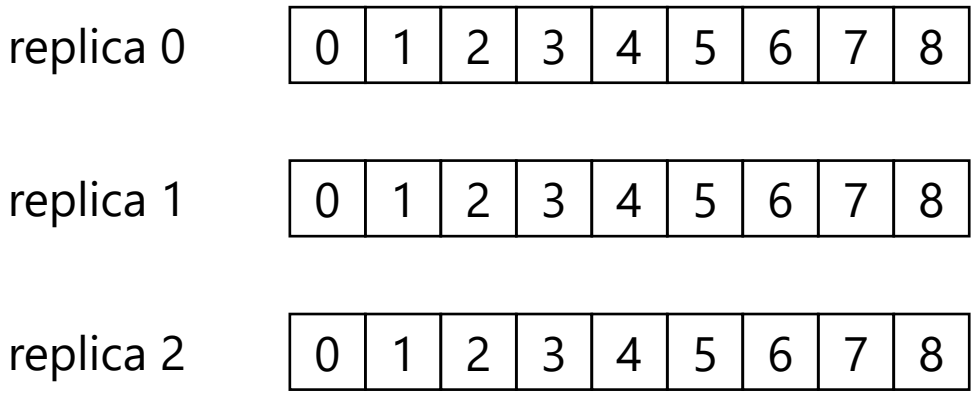
.NET Kafka Producer

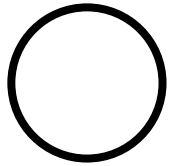


.NET Kafka Producer

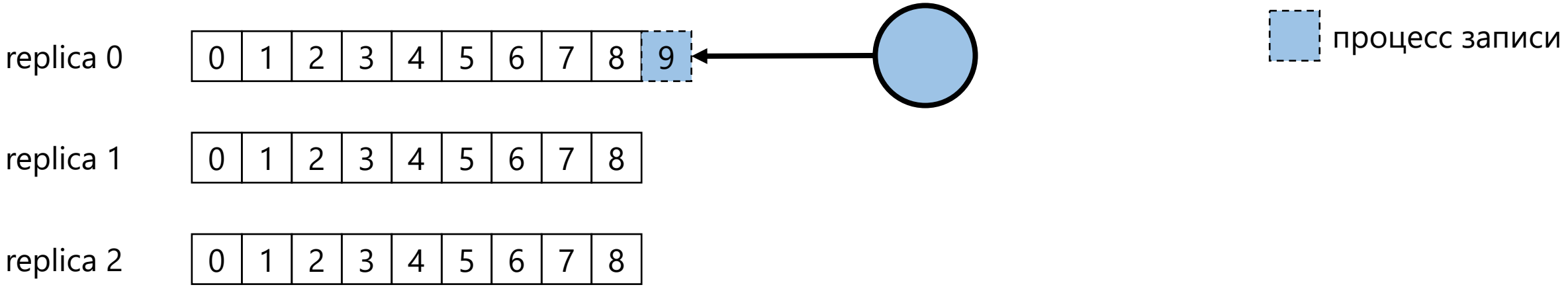


.NET Kafka Producer

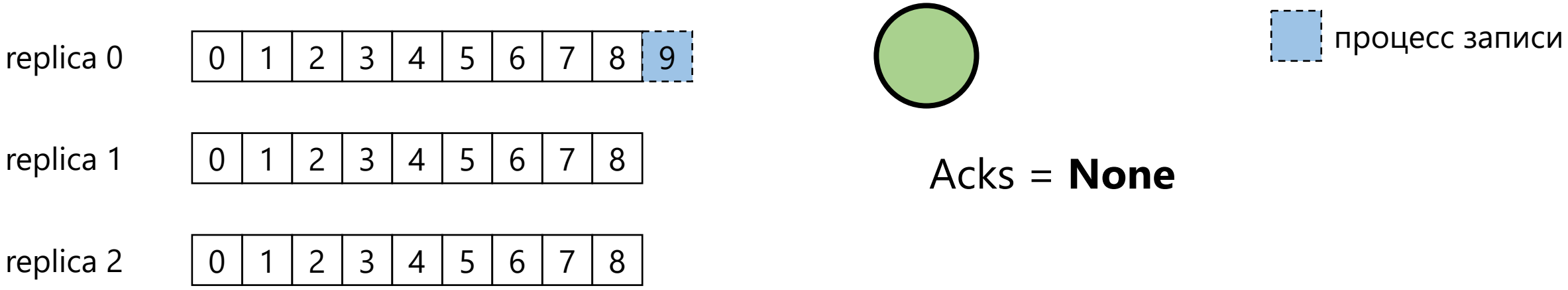




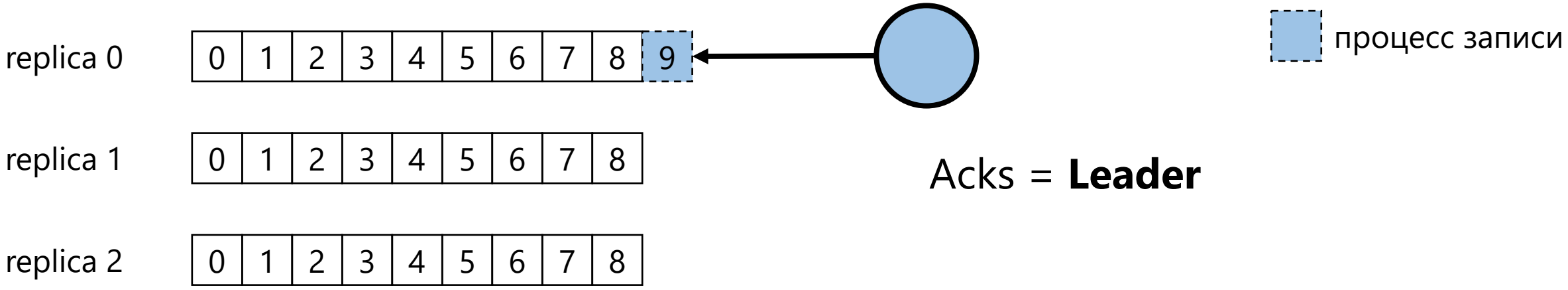
.NET Kafka Producer



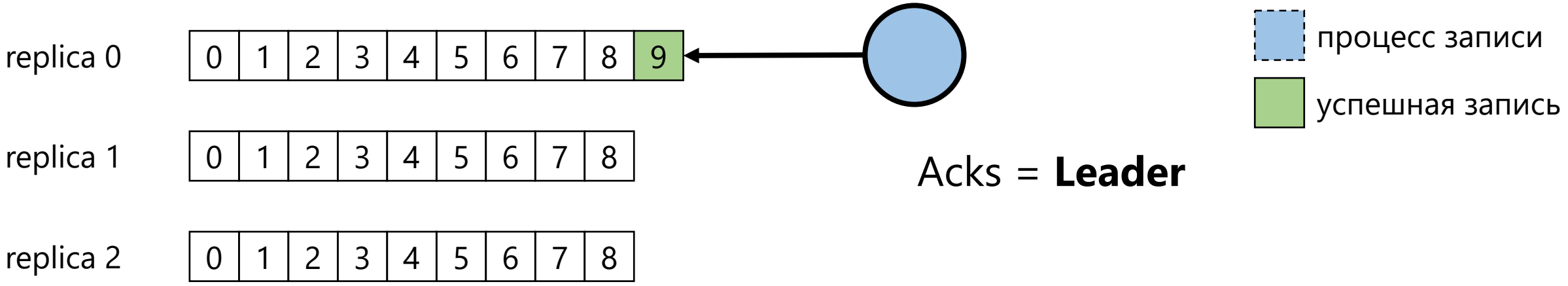
.NET Kafka Producer



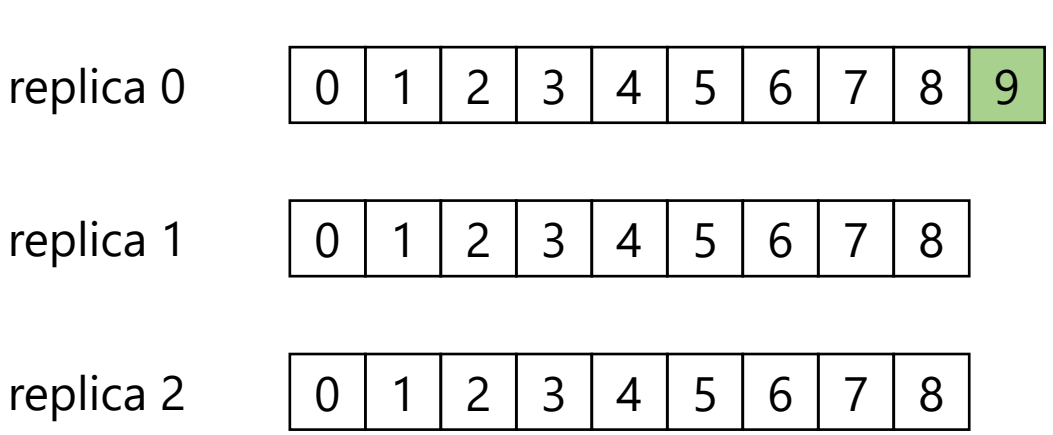
.NET Kafka Producer

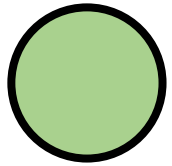


.NET Kafka Producer

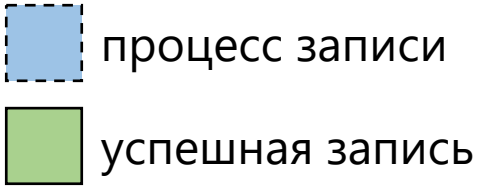


.NET Kafka Producer

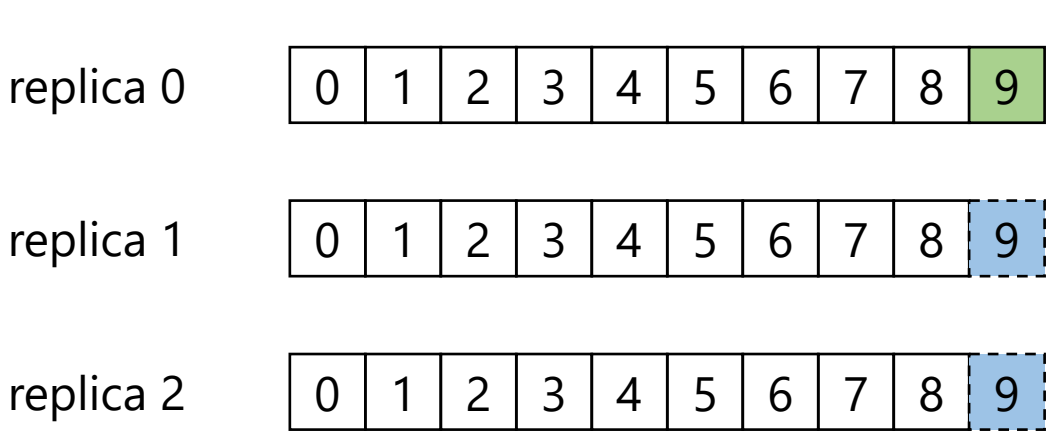


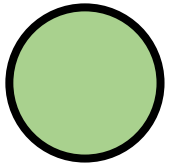


Acks = **Leader**

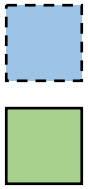


.NET Kafka Producer

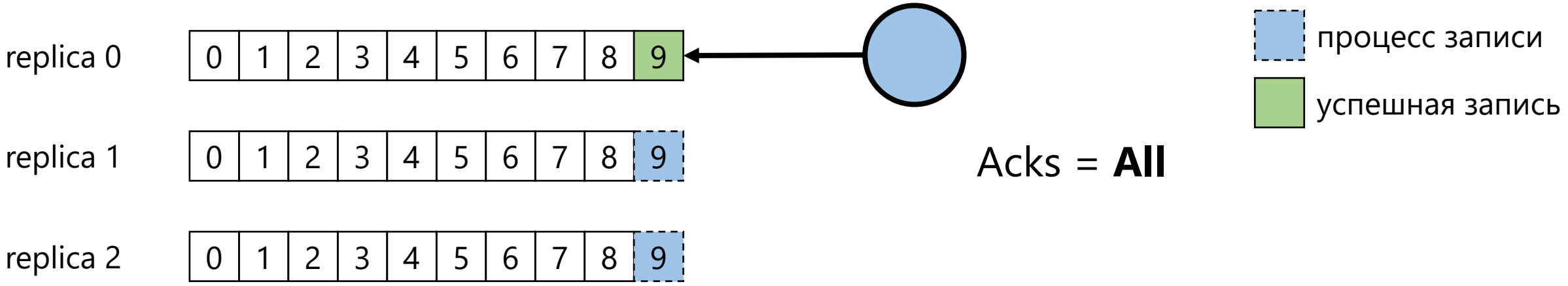




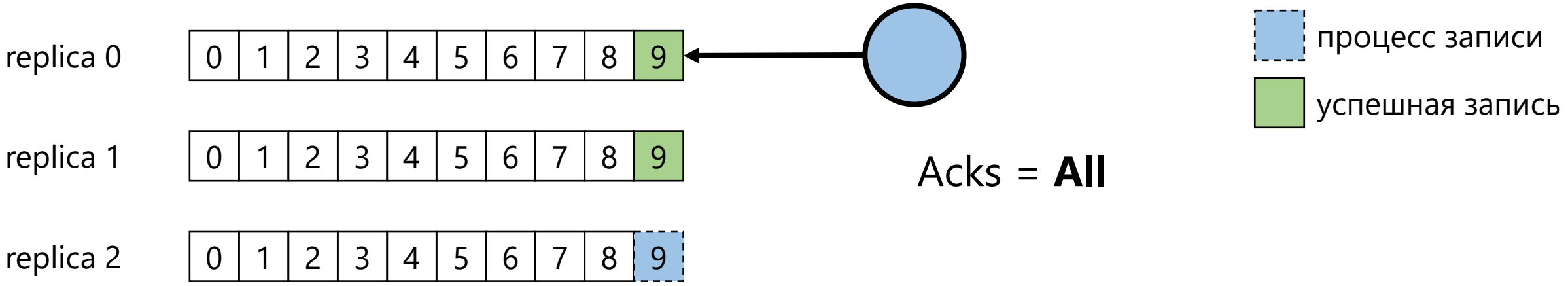
Acks = **Leader**



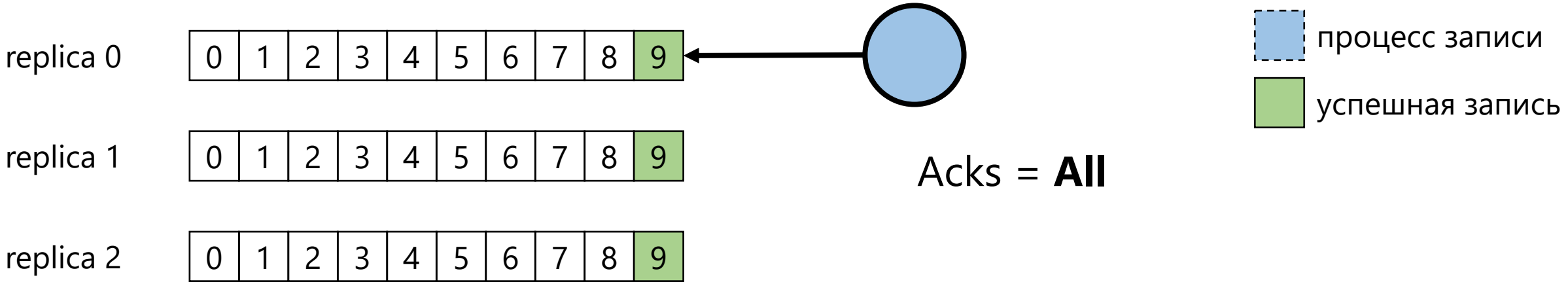
.NET Kafka Producer



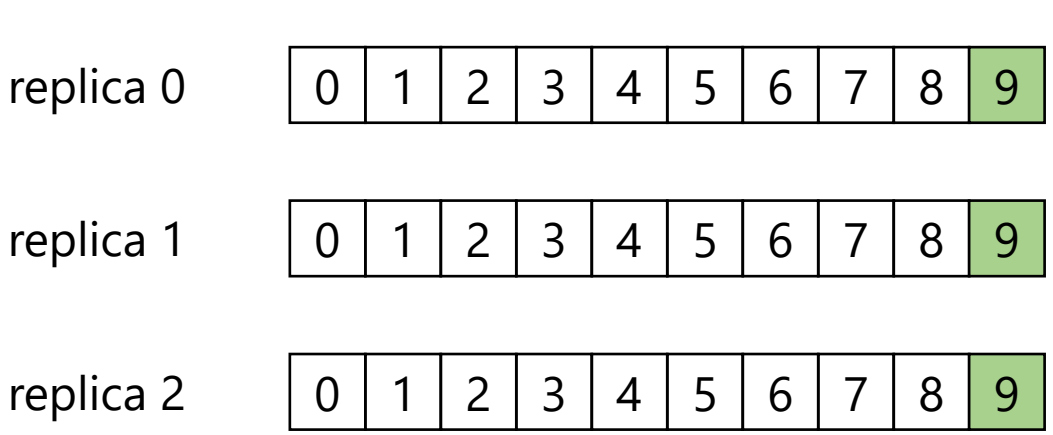
.NET Kafka Producer

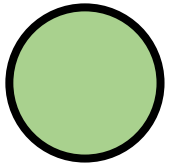


.NET Kafka Producer

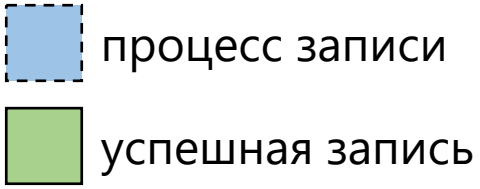


.NET Kafka Producer

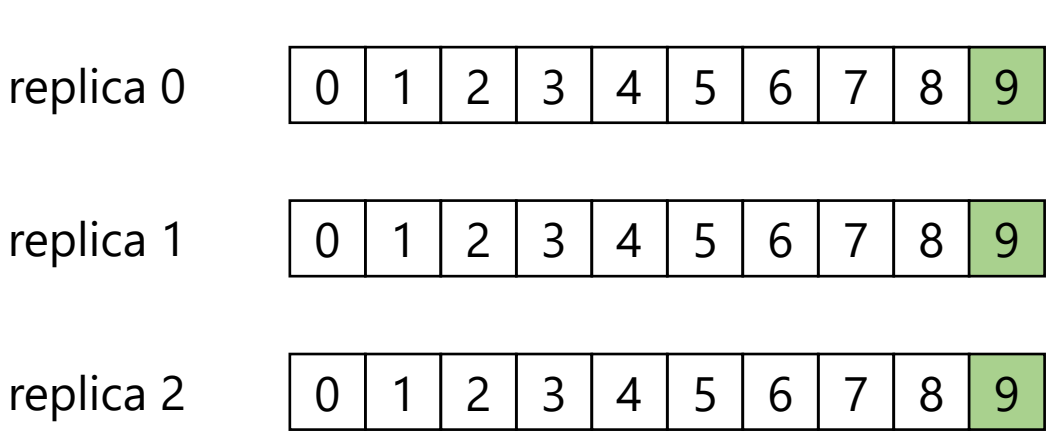


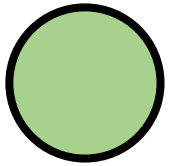


Acks = **All**

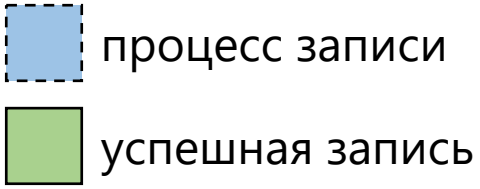


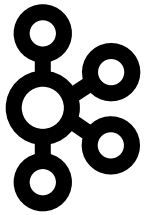
.NET Kafka Producer





Acks = **All**





.NET Kafka Producer

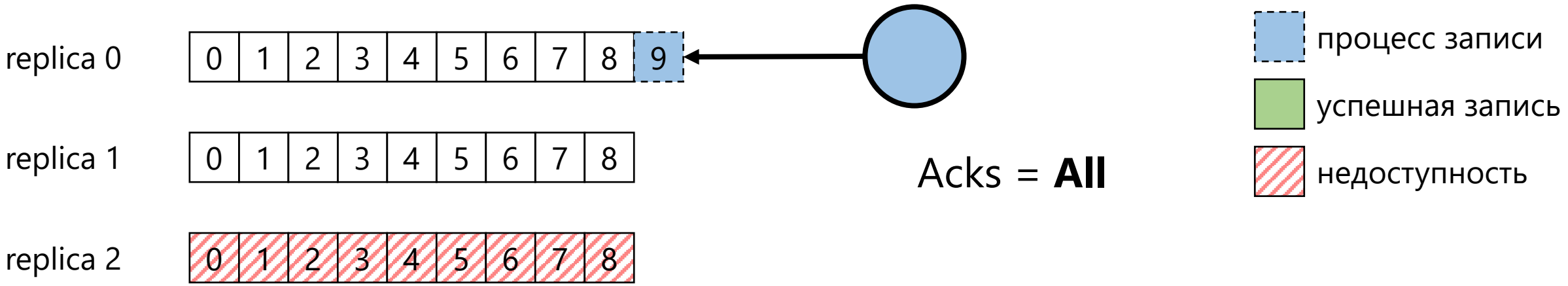
Topic config: min.insync.replicas = 2

.NET Kafka Producer

Topic config: min.insync.replicas = 2

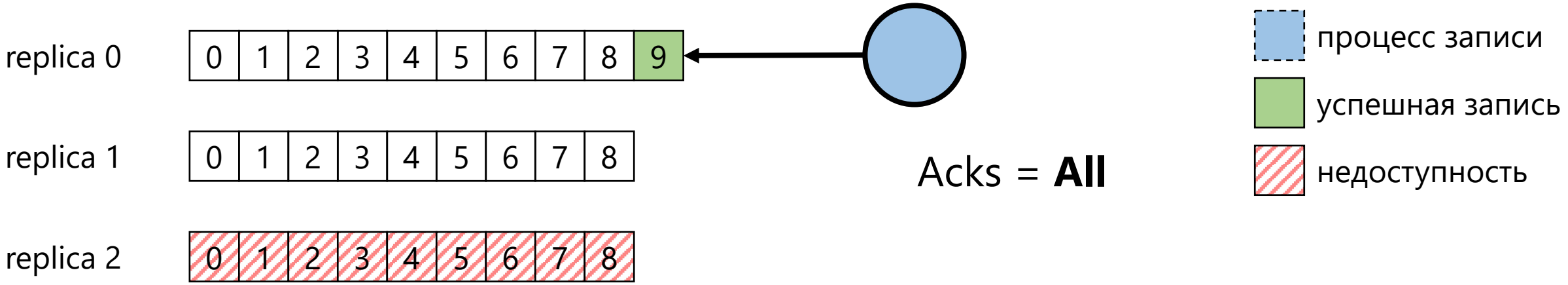
.NET Kafka Producer

Topic config: min.insync.replicas = 2



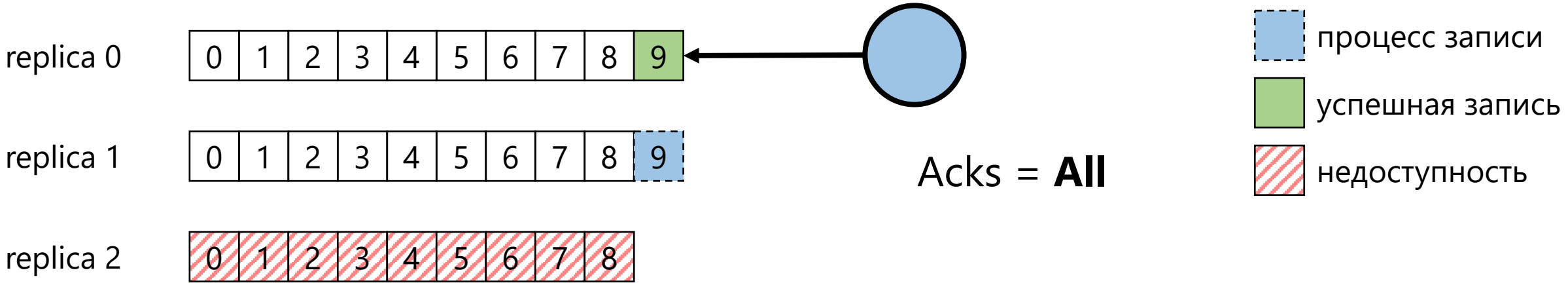
.NET Kafka Producer

Topic config: min.insync.replicas = 2



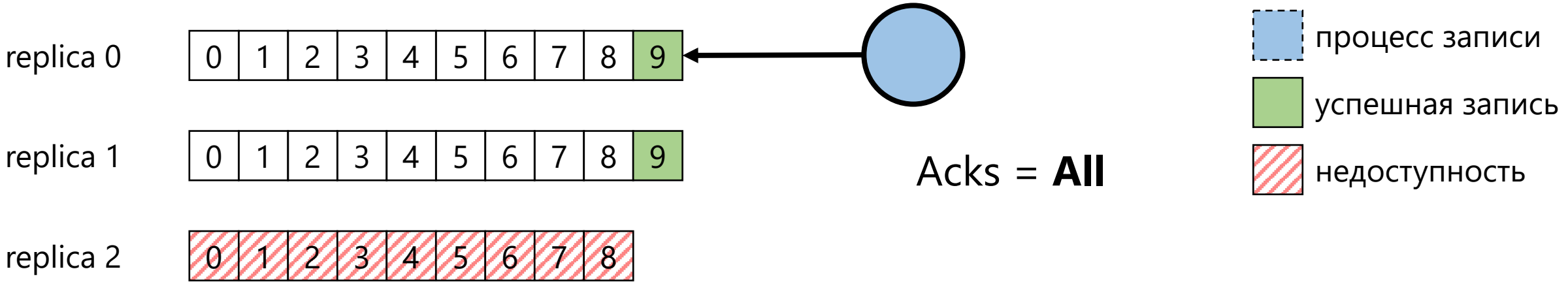
.NET Kafka Producer

Topic config: min.insync.replicas = 2



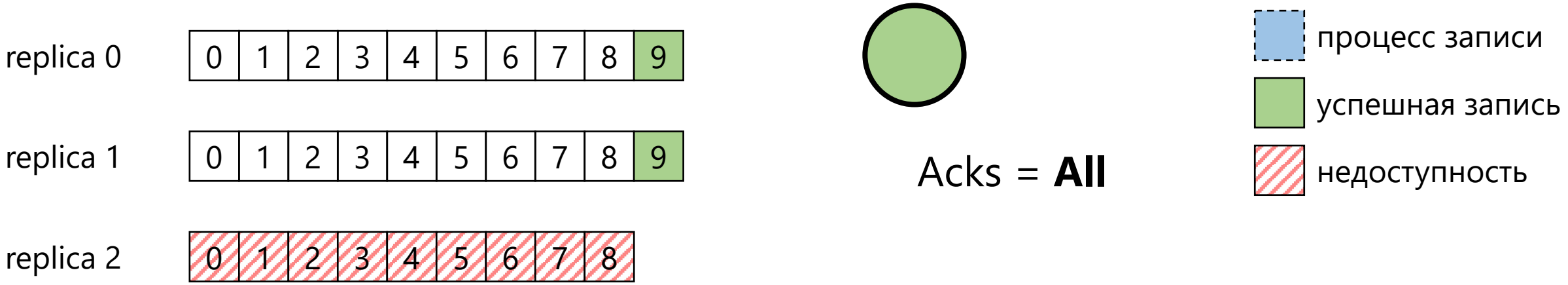
.NET Kafka Producer

Topic config: min.insync.replicas = 2



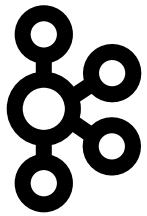
.NET Kafka Producer

Topic config: min.insync.replicas = 2



.NET Kafka Producer

Производительность



.NET Kafka Producer

Производительность

— Низкая задержка

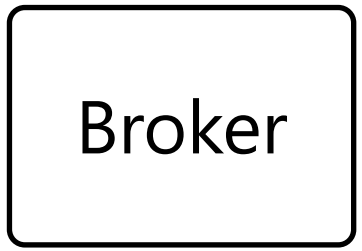
— Высокая пропускная способность

.NET Kafka Producer

Производительность

— Низкая задержка

— Высокая пропускная способность



.NET Kafka Producer

Производительность

— Низкая задержка

— Высокая пропускная способность

.NET Kafka Producer

Производительность

— Низкая задержка

— Высокая пропускная способность

.NET Kafka Producer

Производительность

— Низкая задержка

— Высокая пропускная способность

.NET Kafka Producer

Производительность

— Низкая задержка

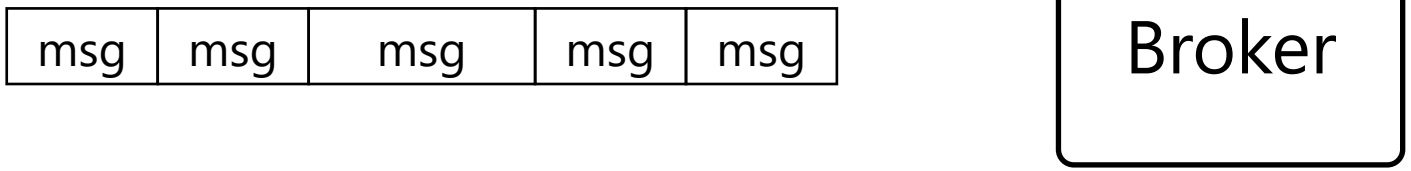
— Высокая пропускная способность

.NET Kafka Producer

Производительность

— Низкая задержка

— Высокая пропускная способность

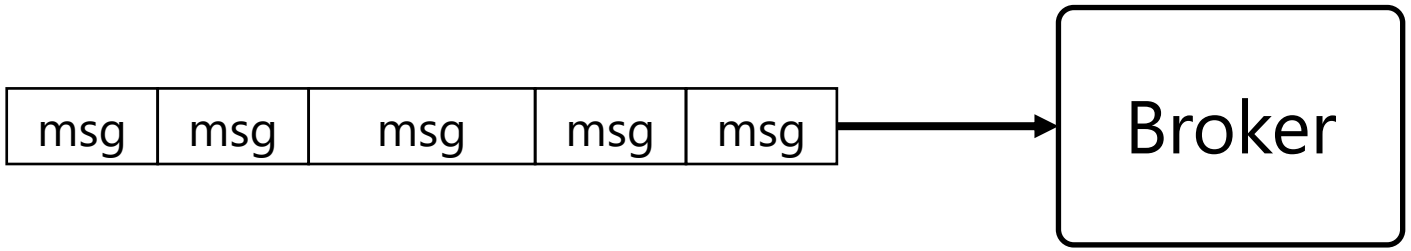


.NET Kafka Producer

Производительность

— Низкая задержка

— Высокая пропускная способность

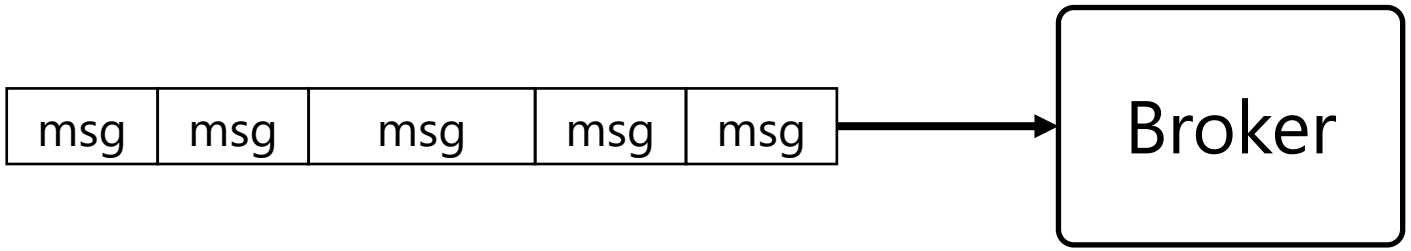


.NET Kafka Producer

Производительность

— Низкая задержка

— Высокая пропускная способность



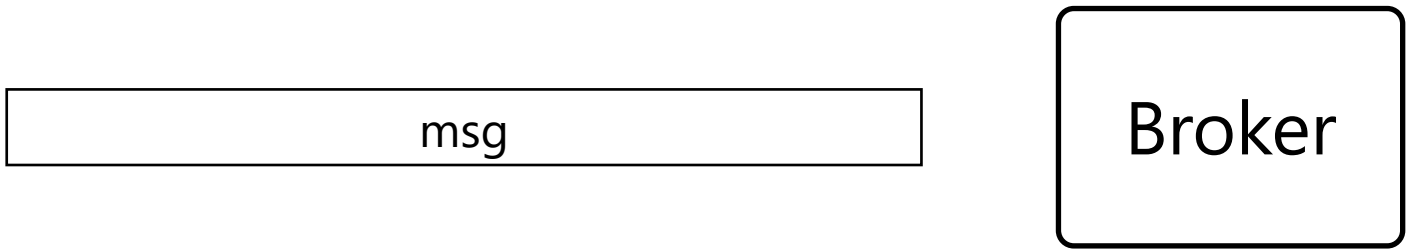
BatchSize = 1000000
BatchNumMessages = 10000

.NET Kafka Producer

Производительность

— Низкая задержка

— Высокая пропускная способность



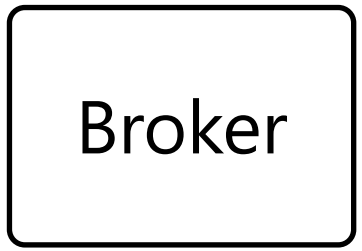
BatchSize = 1000000
BatchNumMessages = 10000

.NET Kafka Producer

Производительность

— Низкая задержка

— Высокая пропускная способность



BatchSize = 1000000

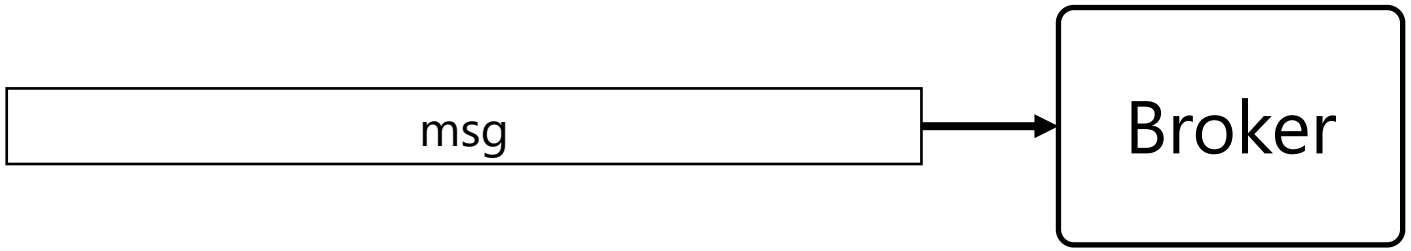
BatchNumMessages = 10000

.NET Kafka Producer

Производительность

— Низкая задержка

— Высокая пропускная способность



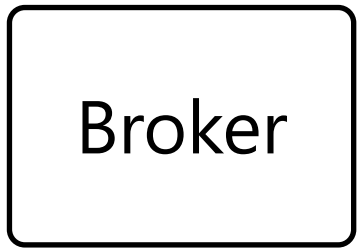
BatchSize = 1000000
BatchNumMessages = 10000

.NET Kafka Producer

Производительность

— Низкая задержка

— Высокая пропускная способность



BatchSize = 1000000
BatchNumMessages = 10000
LingerMs = 5

.NET Kafka Producer

Производительность

— Низкая задержка

— Высокая пропускная способность

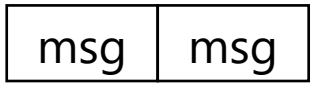
BatchSize = 1000000
BatchNumMessages = 10000
LingerMs = 5

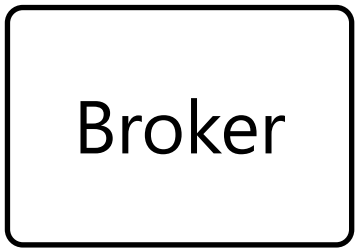
.NET Kafka Producer

Производительность

— Низкая задержка

— Высокая пропускная способность





BatchSize = 1000000
BatchNumMessages = 10000
LingerMs = 5

.NET Kafka Producer

Производительность

— Низкая задержка

— Высокая пропускная способность

Broker

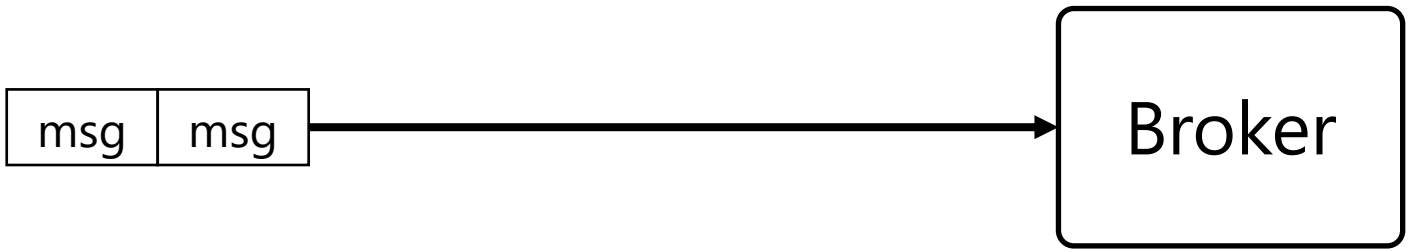
BatchSize = 1000000
BatchNumMessages = 10000
LingerMs = 5

.NET Kafka Producer

Производительность

— Низкая задержка

— Высокая пропускная способность



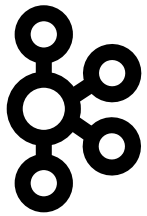
BatchSize = 1000000
BatchNumMessages = 10000
LingerMs = 5

.NET Kafka Producer

Производительность

— Низкая задержка

— Высокая пропускная способность



.NET Kafka Producer

```
public enum CompressionType
{
    None,
    Gzip,
    Snappy,
    Lz4,
    Zstd
}
```

.NET Kafka Producer

```
public enum CompressionType
{
    None,
    Gzip,
    Snappy,
    Lz4,
    Zstd
}
```

.NET Kafka Producer

Выводы

— Producer выбирает партицию для сообщения

— Producer определяет уровень гарантии доставки

— В Producer можно тюнить производительность

.NET Kafka Consumer

.NET Kafka Consumer

```
var consumer =
    new ConsumerBuilder<string, string>(config)
        .Build();

consumer.Subscribe("dotnext");

ConsumeResult<string, string> result =
    consumer.Consume(5.Seconds());
Console.WriteLine(result?.Message.Value);
```

.NET Kafka Consumer

```
var consumer =
    new ConsumerBuilder<string, string>(config)
        .Build();

consumer.Subscribe("dotnext");

ConsumeResult<string, string> result =
    consumer.Consume(5.Seconds());
Console.WriteLine(result?.Message.Value);
```

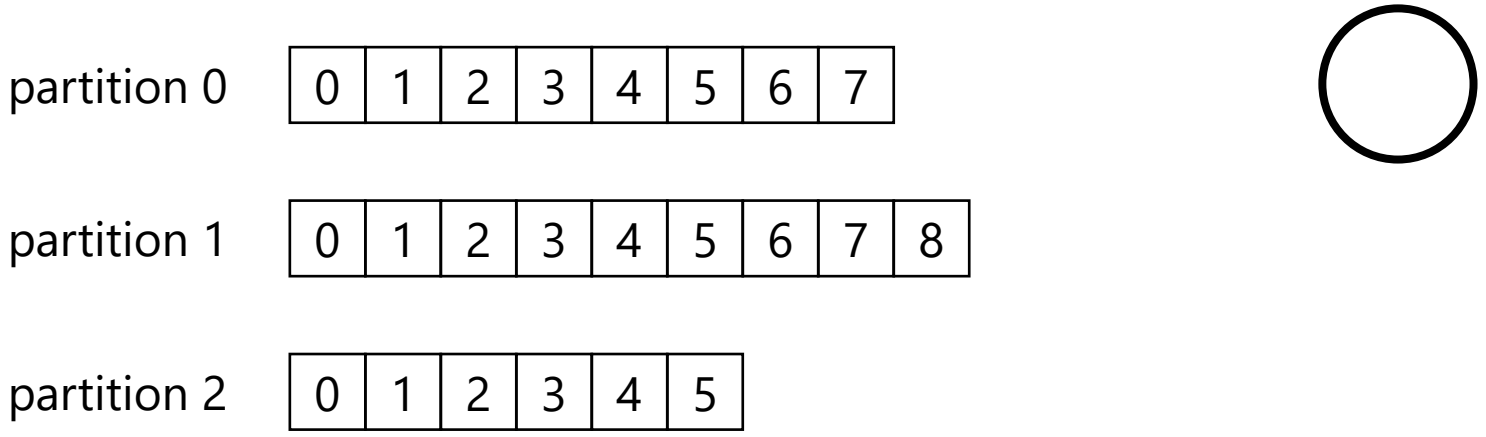
.NET Kafka Consumer

```
var consumer =
    new ConsumerBuilder<string, string>(config)
        .Build();

consumer.Subscribe("dotnext");

ConsumeResult<string, string> result =
    consumer.Consume(5.Seconds());
Console.WriteLine(result?.Message.Value);
```

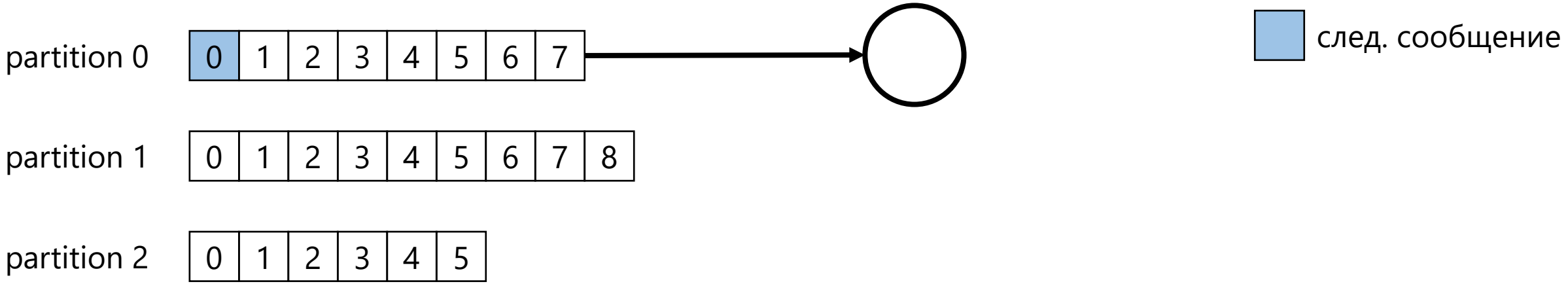
.NET Kafka Consumer

```
var consumer =
    new ConsumerBuilder<string, string>(config)
        .Build();

consumer.Subscribe("dotnext");

ConsumeResult<string, string> result =
    consumer.Consume(5.Seconds());
Console.WriteLine(result?.Message.Value);
```

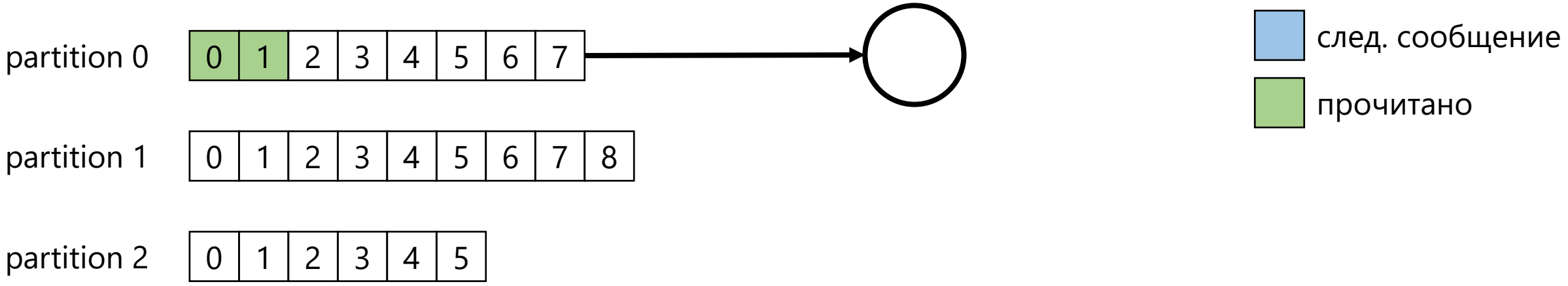
.NET Kafka Consumer



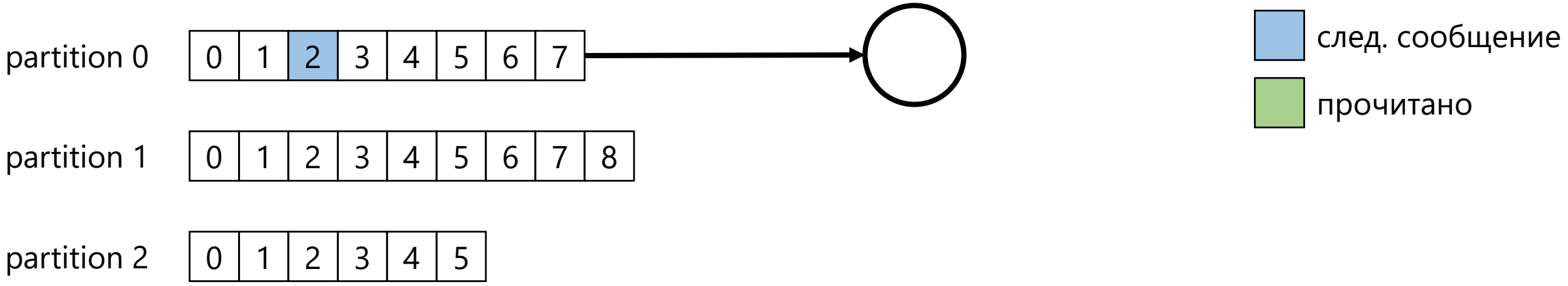
.NET Kafka Consumer



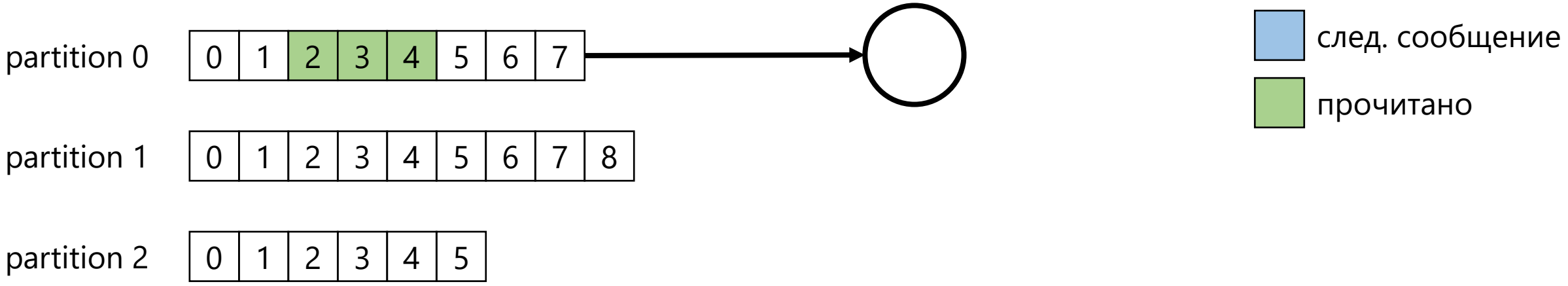
.NET Kafka Consumer



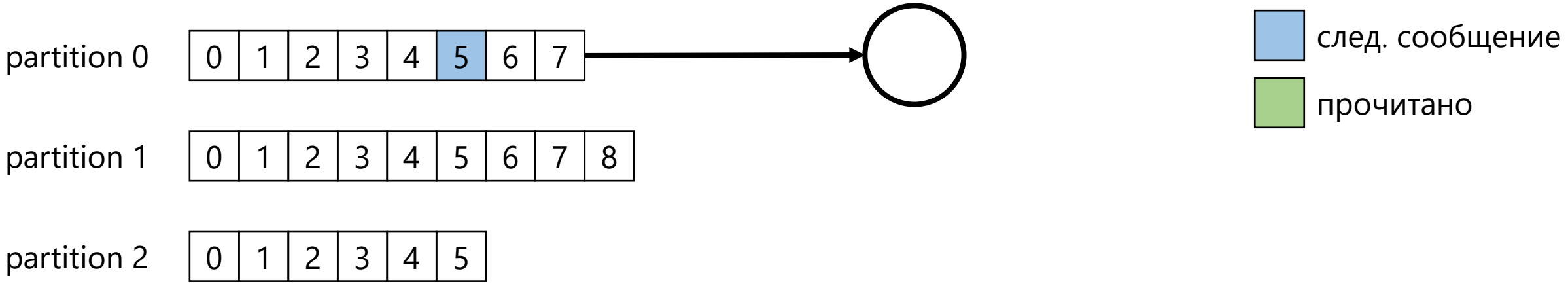
.NET Kafka Consumer



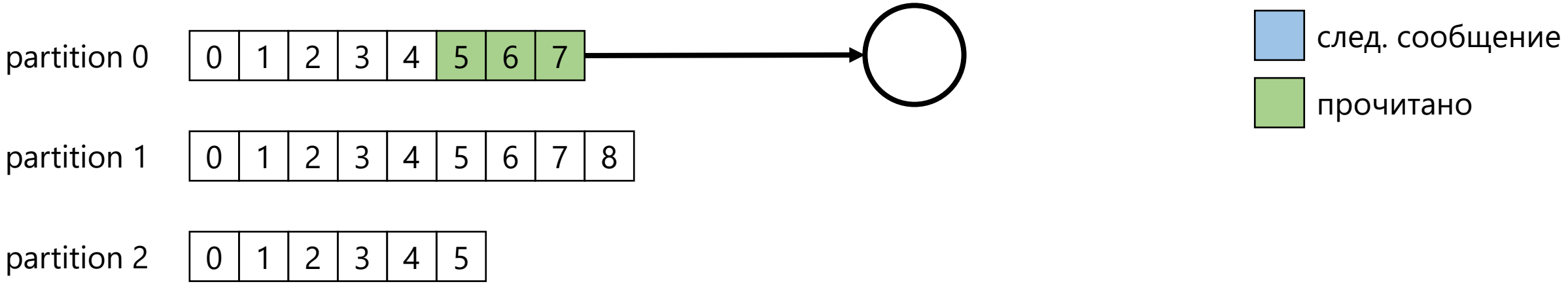
.NET Kafka Consumer



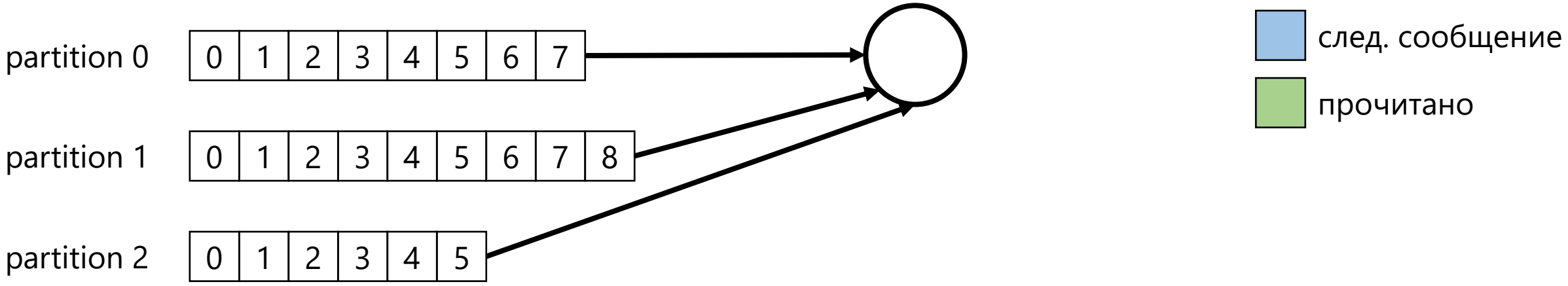
.NET Kafka Consumer



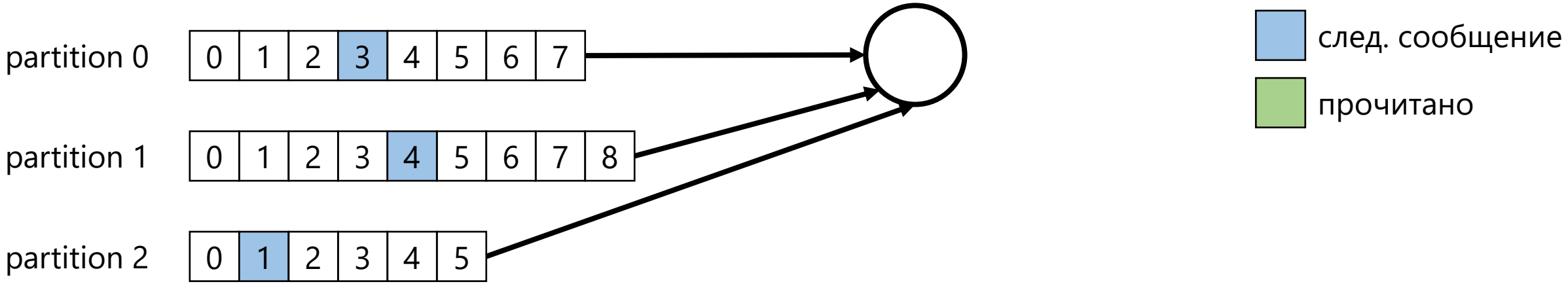
.NET Kafka Consumer



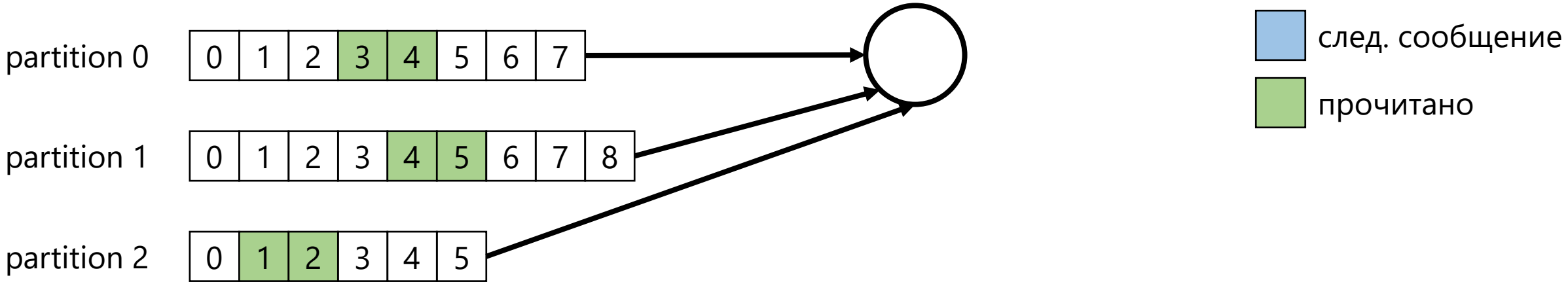
.NET Kafka Consumer



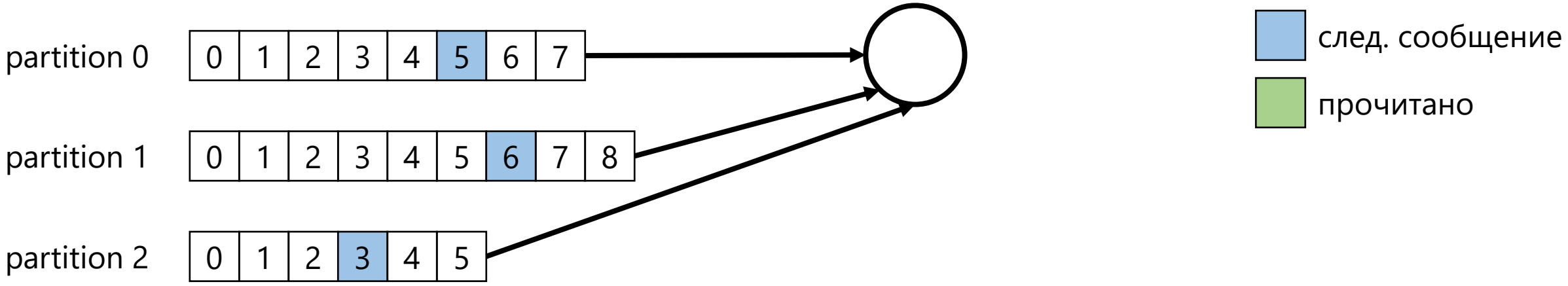
.NET Kafka Consumer



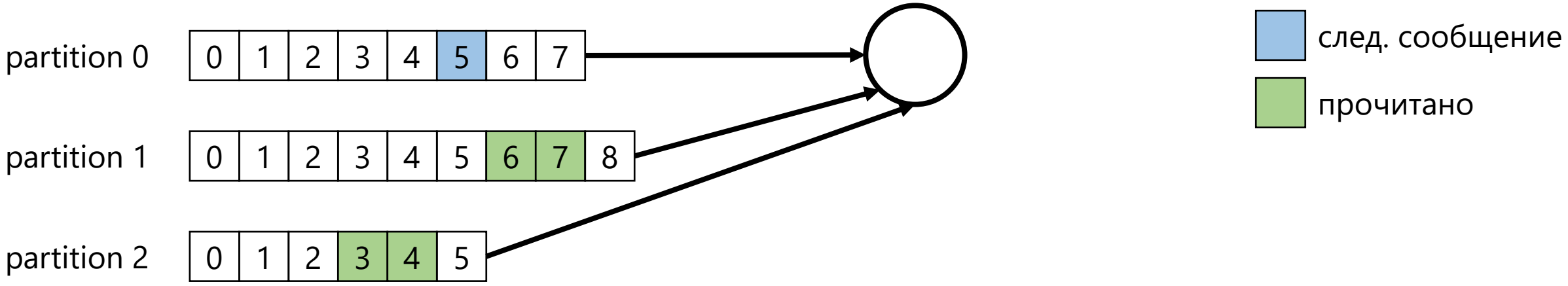
.NET Kafka Consumer



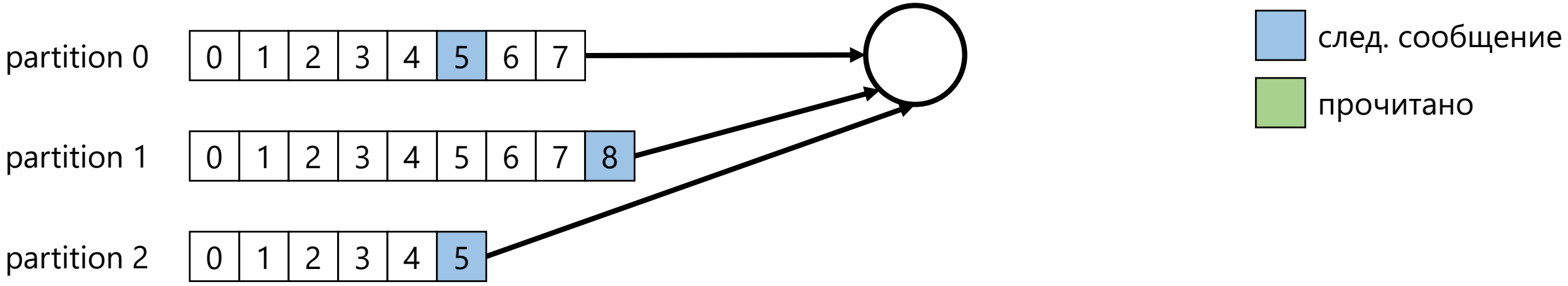
.NET Kafka Consumer



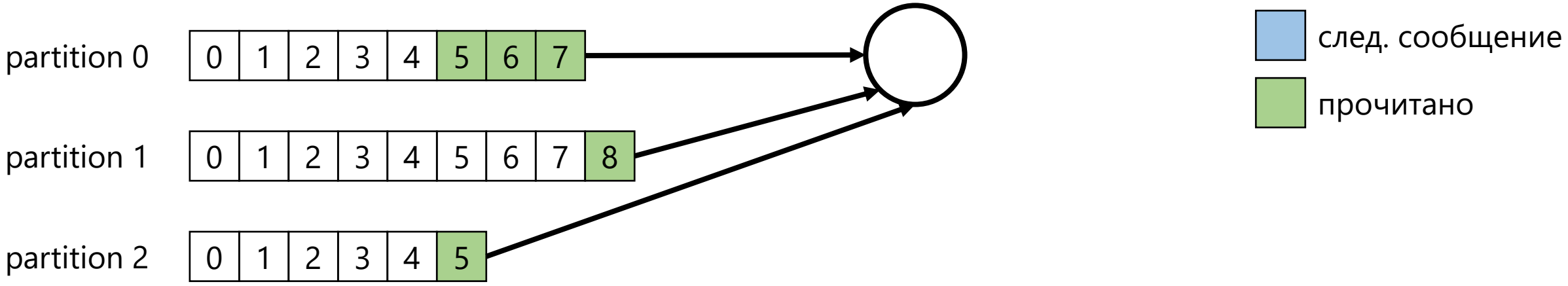
.NET Kafka Consumer



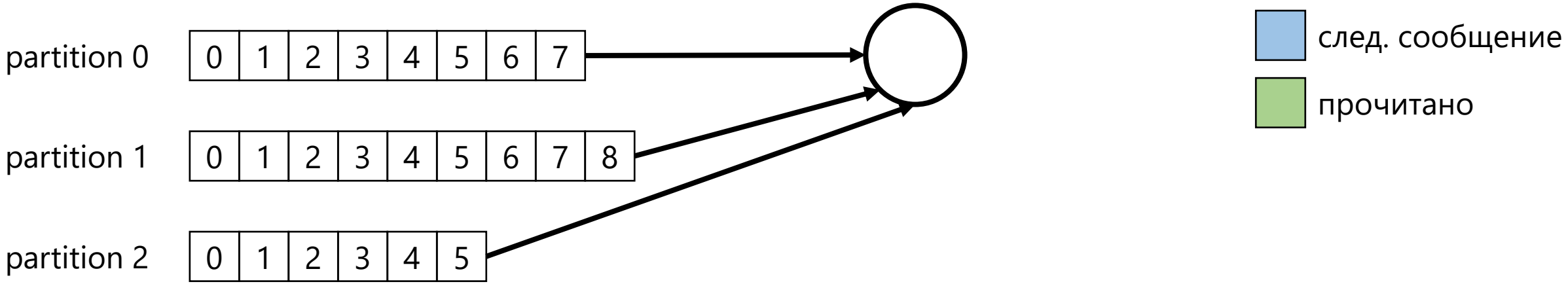
.NET Kafka Consumer



.NET Kafka Consumer



.NET Kafka Consumer

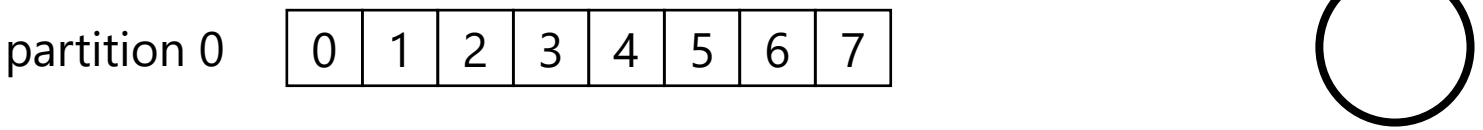


.NET Kafka Consumer

Commit offset

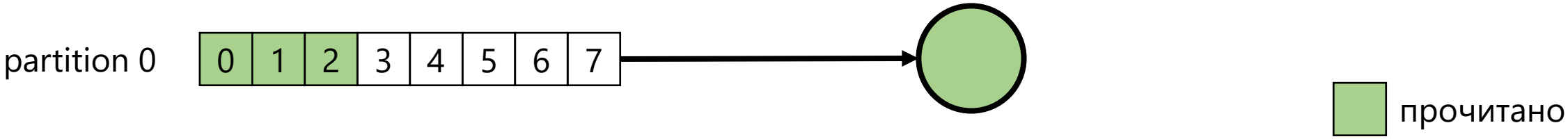
.NET Kafka Consumer

Commit offset



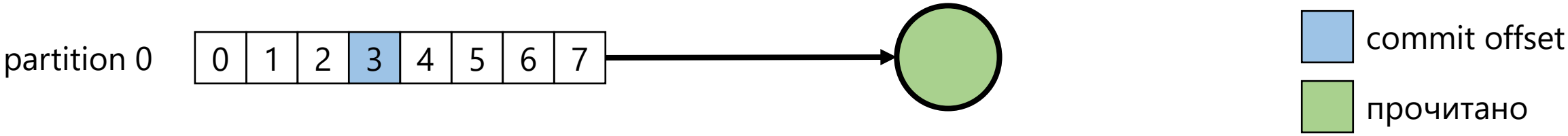
.NET Kafka Consumer

Commit offset



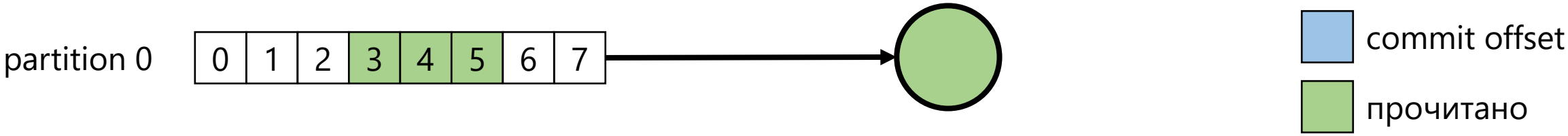
.NET Kafka Consumer

Commit offset



.NET Kafka Consumer

Commit offset



.NET Kafka Consumer

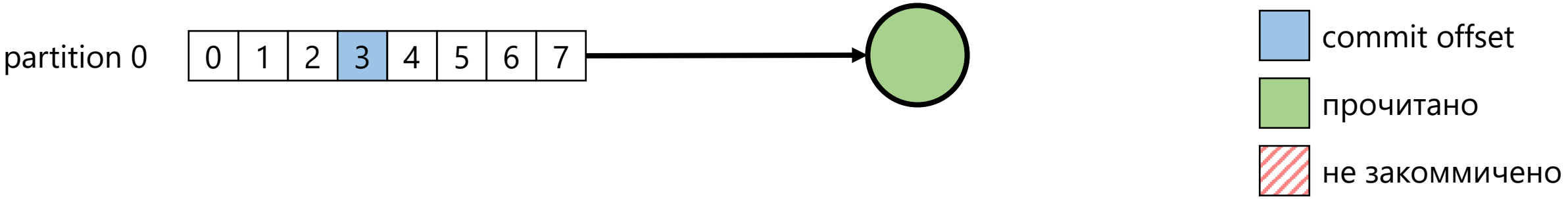
Commit offset

.NET Kafka Consumer

Commit offset

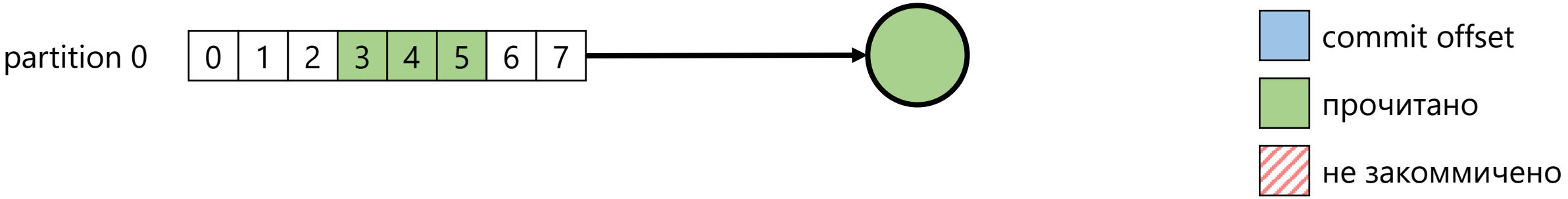
.NET Kafka Consumer

Commit offset



.NET Kafka Consumer

Commit offset



.NET Kafka Consumer

Гарантии обработки

— at least once

— mostly once

.NET Kafka Consumer

```
ConsumeResult<string, string> result =
    consumer.Consume(5.Seconds());

if (result != null)
{
    consumer.Commit(result);
    process(result);
}
```

.NET Kafka Consumer

```
ConsumeResult<string, string> result =
    consumer.Consume(5.Seconds());

if (result != null)
{
    consumer.Commit(result);
    process(result);
}
```

mostly once

Commit до обработки

.NET Kafka Consumer

```
ConsumeResult<string, string> result =
    consumer.Consume(5.Seconds());

if (result != null)
{
    process(result);
    consumer.Commit(result);
}
```

.NET Kafka Consumer

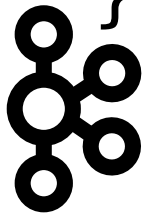
```
ConsumeResult<string, string> result =
    consumer.Consume(5.Seconds());

if (result != null)
{
    process(result);
    consumer.Commit(result);
}
```

at least once

Commit после обработки

.NET Kafka Consumer

```
ConsumeResult<string, string> result =
    consumer.Consume(5.Seconds());

if (result != null)
{
    process(result);
    consumer.Commit(result);
}
```



.NET Kafka Consumer

— 1 коммит на 1 сообщение

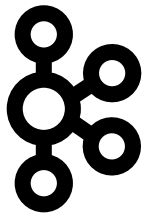
— 1 коммит на N сообщений

.NET Kafka Consumer

— 1 коммит на 1 сообщение

— 1 коммит на N сообщений ✓

⇧ производительность



.NET Kafka Consumer

```
ConsumeResult<string, string> result =
    consumer.Consume(5.Seconds());

if (result != null)
{
    process(result);
    consumer.StoreOffset(result);
}
```

.NET Kafka Consumer

```
ConsumeResult<string, string> result =
    consumer.Consume(5.Seconds());

if (result != null)
{
    process(result);
    consumer.StoreOffset(result);
}
```

.NET Kafka Consumer

```
ConsumeResult<string, string> result =
    consumer.Consume(5.Seconds());

if (result != null)
{
    process(result);
    consumer.StoreOffset(result);
}
```

EnableAutoCommit=true
AutoCommitIntervalMs=5000

.NET Kafka Consumer

```
ConsumeResult<string, string> result =
    consumer.Consume(5.Seconds());

if (result != null)
{
    process(result);
    consumer.StoreOffset(result);
}
```

EnableAutoCommit=true
AutoCommitIntervalMs=5000

.NET Kafka Consumer

```
ConsumeResult<string, string> result =
    consumer.Consume(5.Seconds());

if (result != null)
{
    process(result);
    consumer.StoreOffset(result);
}
```

EnableAutoCommit=true
AutoCommitIntervalMs=5000

.NET Kafka Consumer

```
ConsumeResult<string, string> result =
    consumer.Consume(5.Seconds());

if (result != null)
{
    process(result);
    consumer.StoreOffset(result);
}
```

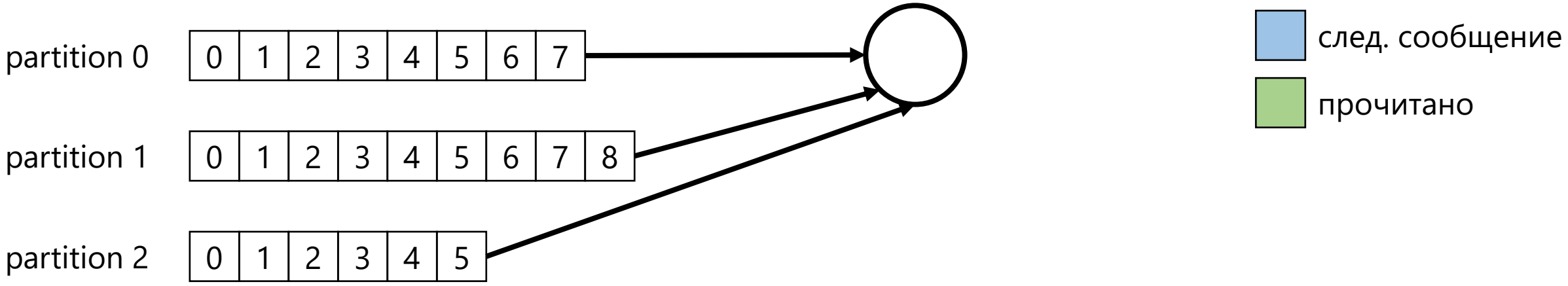
EnableAutoCommit=true
AutoCommitIntervalMs=5000

.NET Kafka Consumer

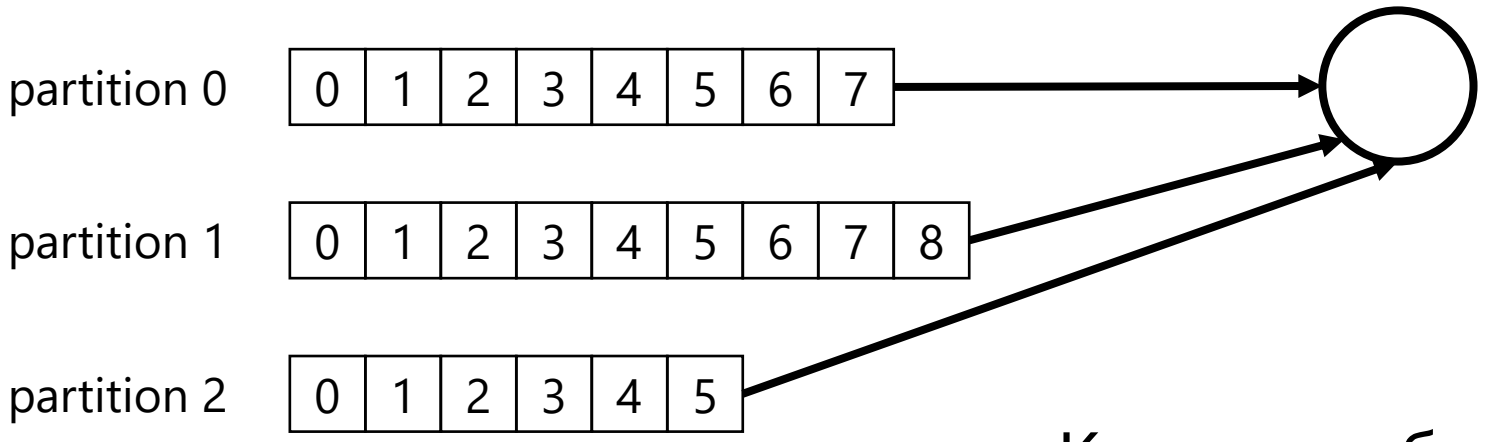
```
ConsumeResult<string, string> result =
    consumer.Consume(5.Seconds());

if (result != null)
{
    process(result);
    consumer.StoreOffset(result);
}
```

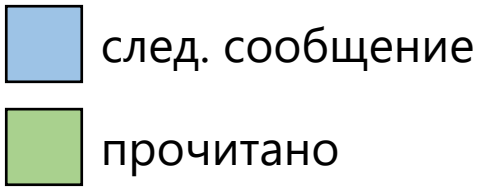
EnableAutoCommit=true
AutoCommitIntervalMs=5000
EnableAutoOffsetStore=true

.NET Kafka Consumer

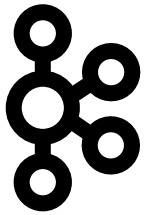


.NET Kafka Consumer





🤔 Какое сообщение читать следующим, если ещё ничего не коммитили?



.NET Kafka Consumer

```
public enum AutoOffsetReset
{
    Latest = 0,
    Earliest = 1,
    Error = 2
}
```

.NET Kafka Consumer

```
public enum AutoOffsetReset
{
    Latest = 0,
    Earliest = 1,
    Error = 2
}
```

.NET Kafka Consumer

```
public enum AutoOffsetReset
{
    Latest = 0,
    Earliest = 1,
    Error = 2
}
```

.NET Kafka Consumer

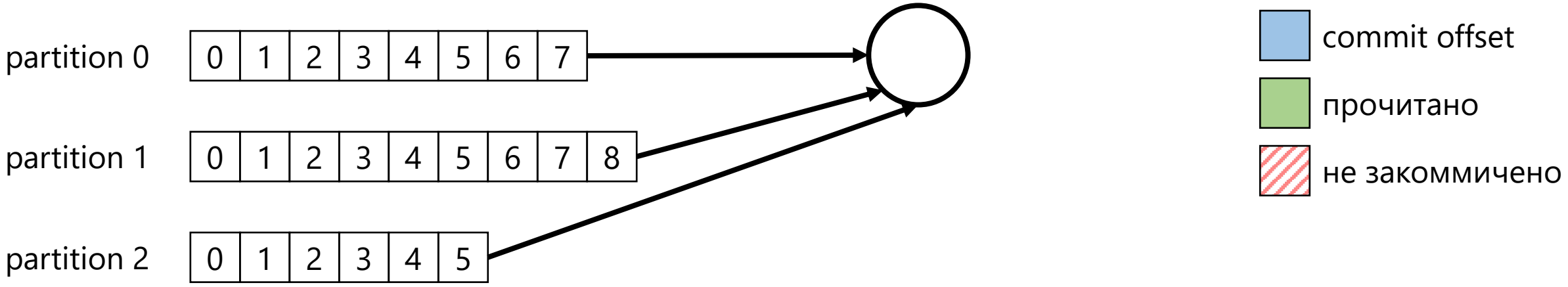
```
public enum AutoOffsetReset
{
    Latest = 0,
    Earliest = 1,
    Error = 2
}
```

.NET Kafka Consumer

Consumer Group

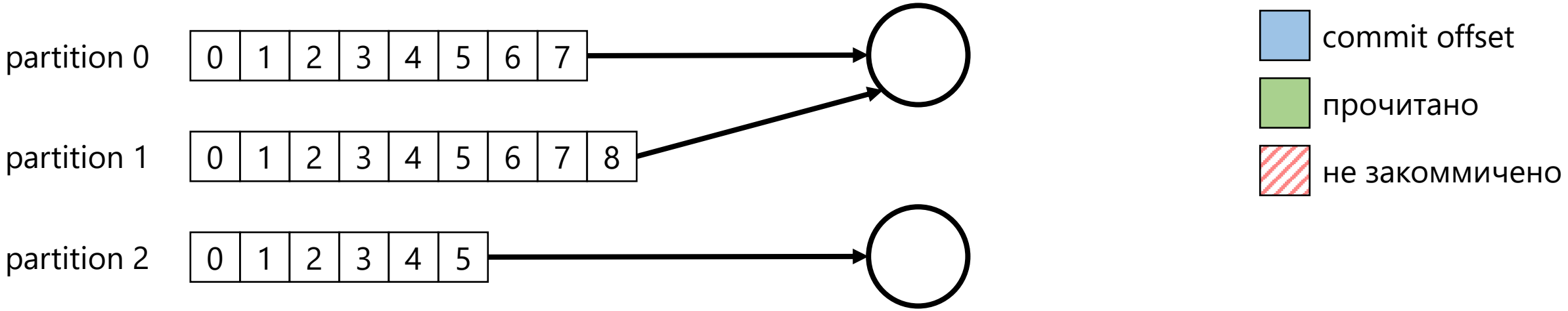
.NET Kafka Consumer

Consumer Group



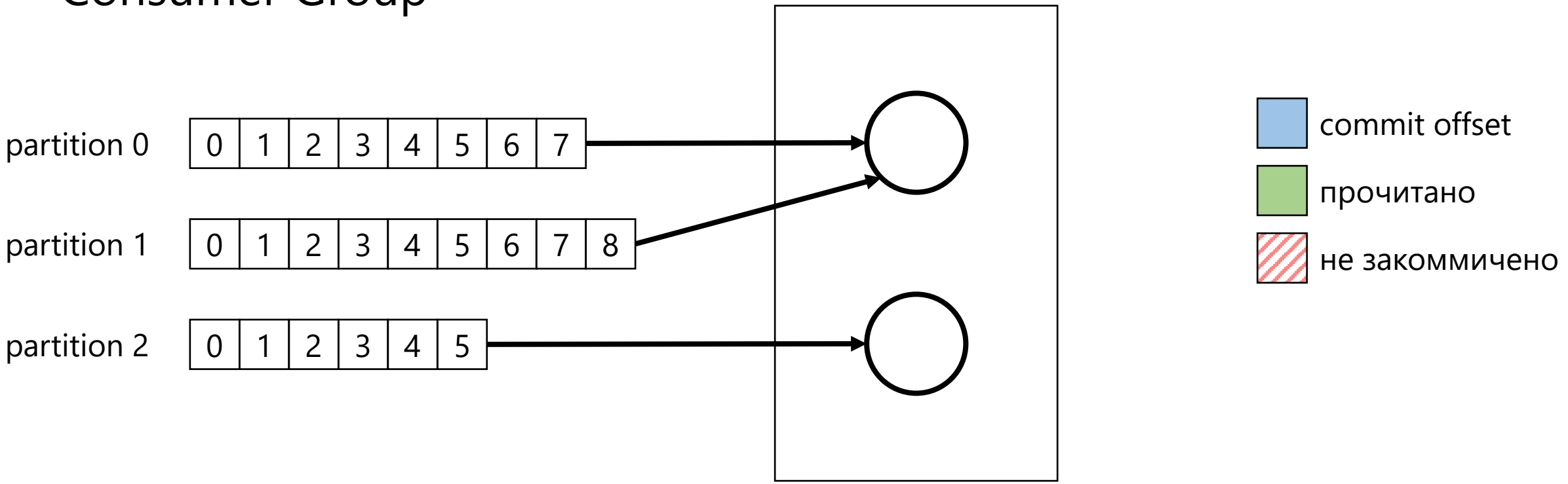
.NET Kafka Consumer

Consumer Group



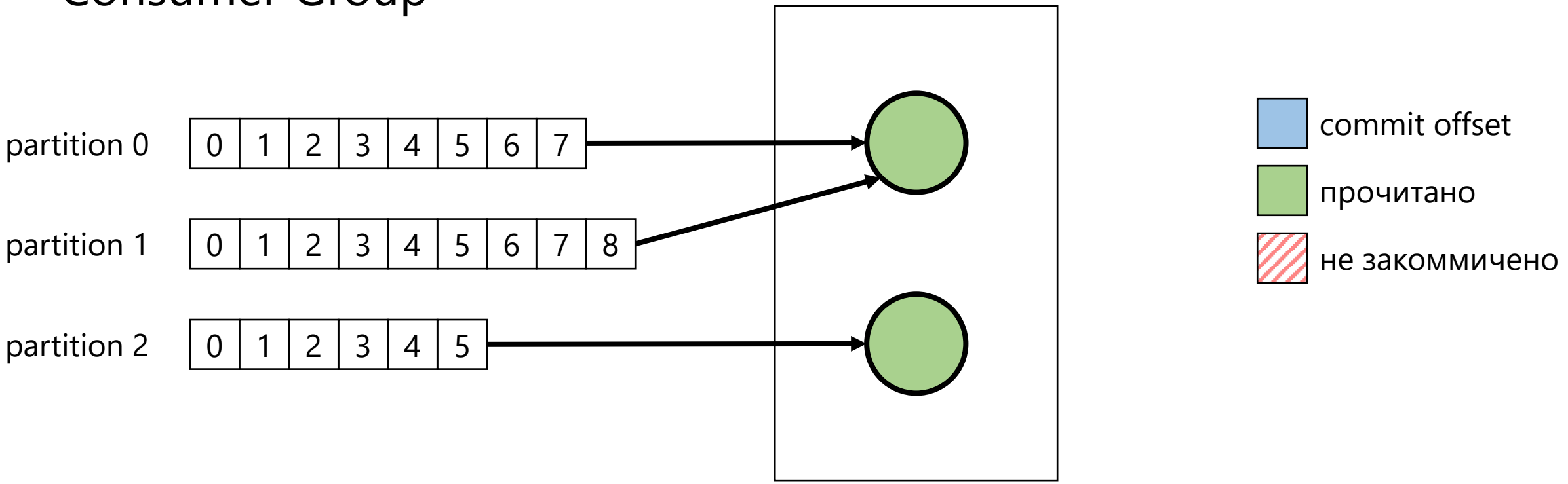
.NET Kafka Consumer

Consumer Group



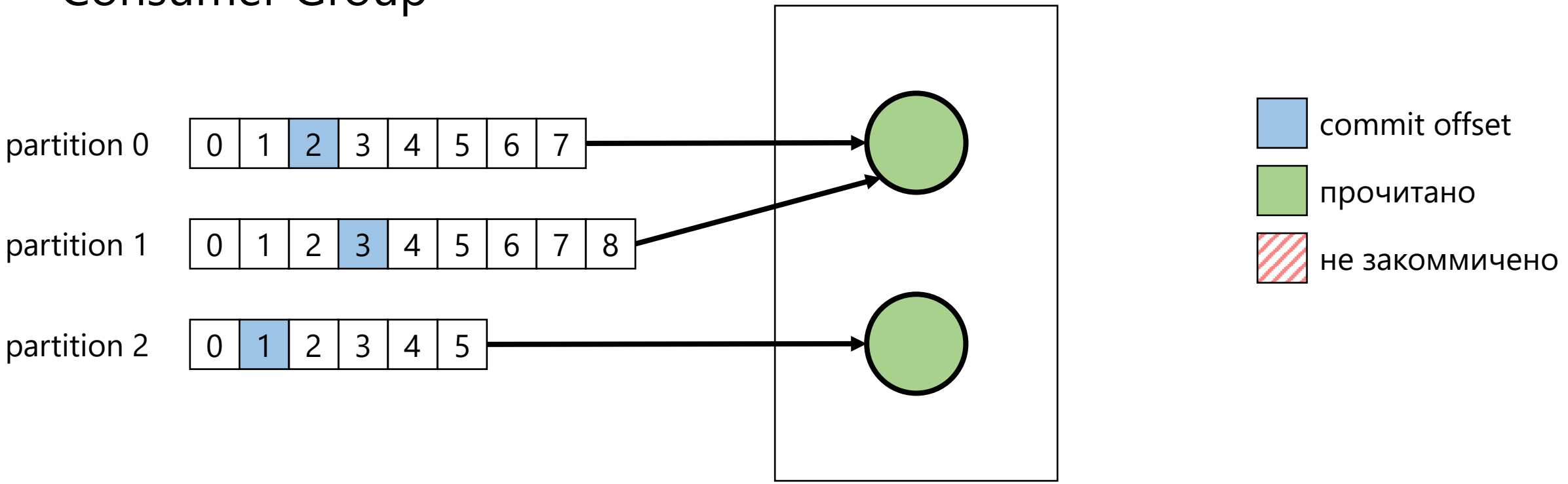
.NET Kafka Consumer

Consumer Group



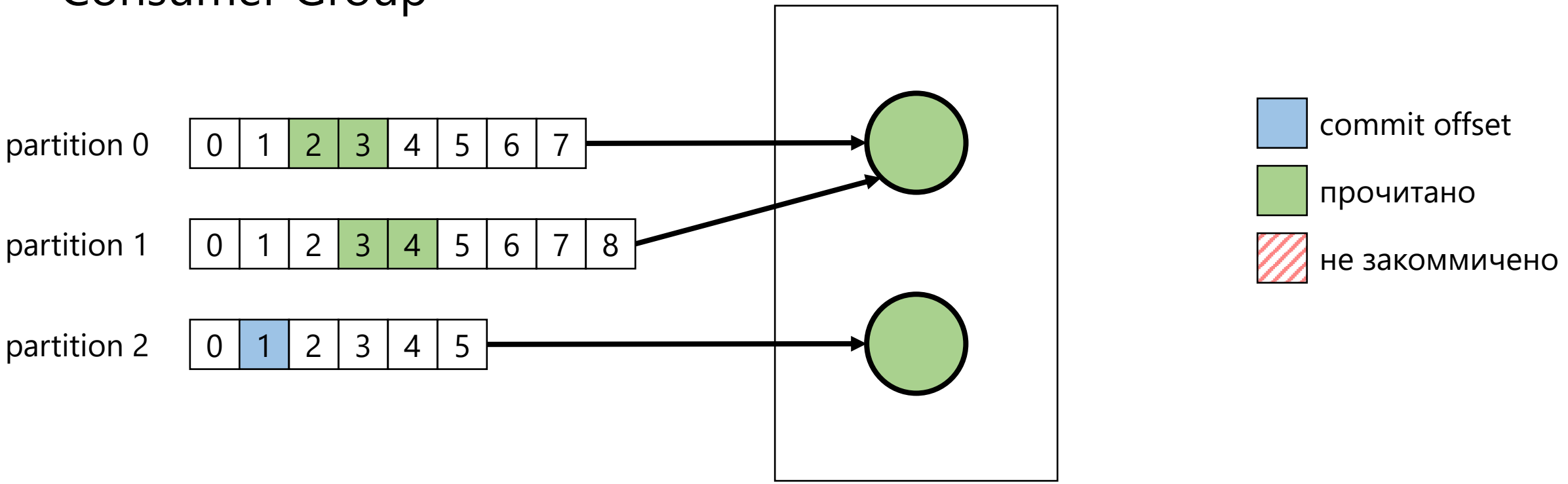
.NET Kafka Consumer

Consumer Group



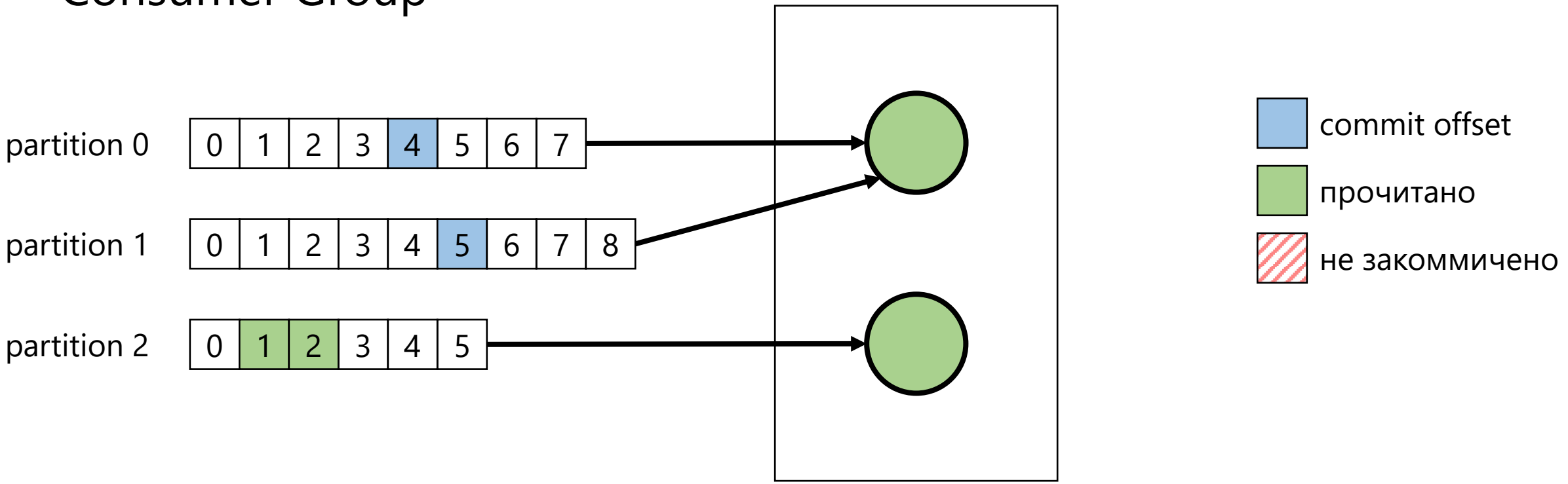
.NET Kafka Consumer

Consumer Group



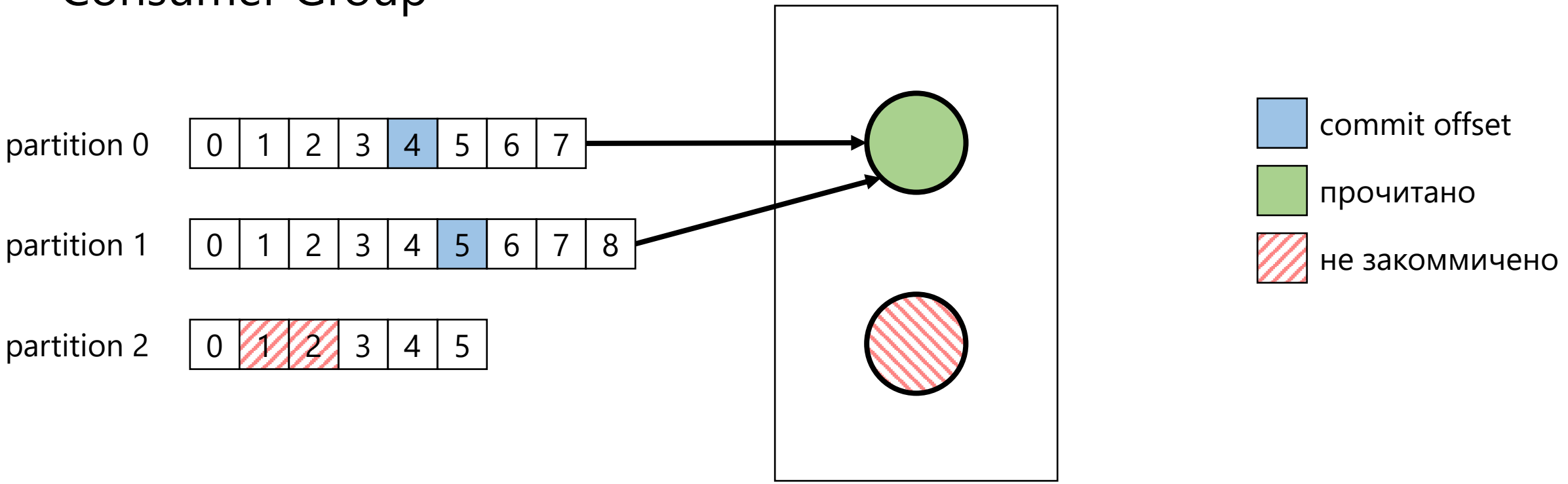
.NET Kafka Consumer

Consumer Group



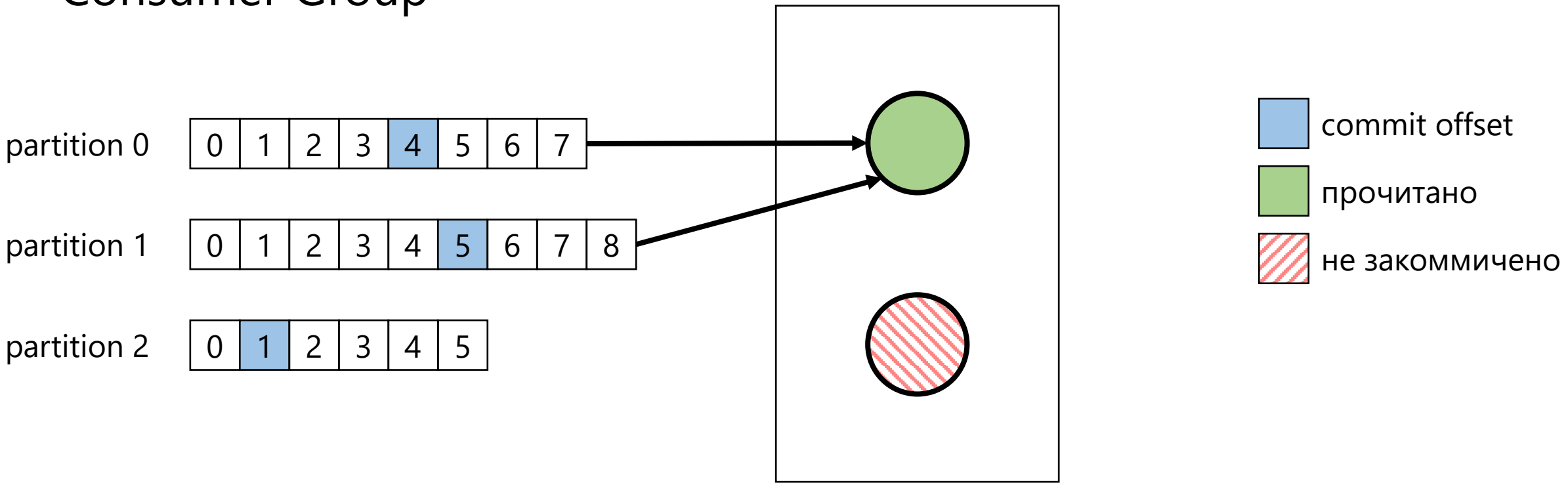
.NET Kafka Consumer

Consumer Group



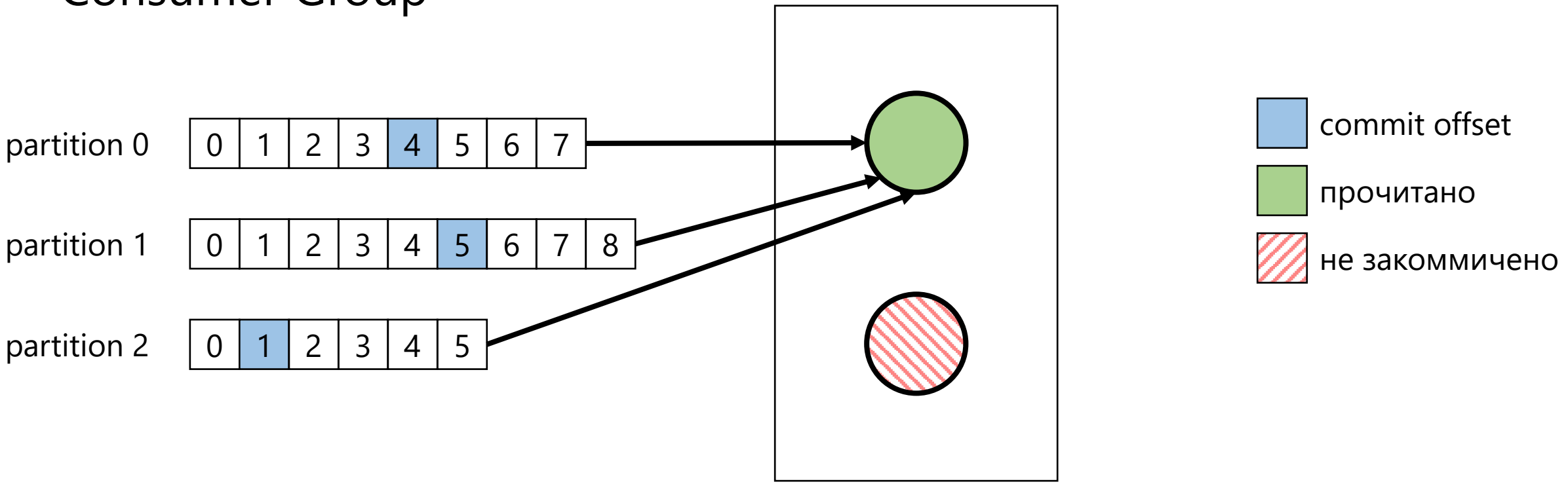
.NET Kafka Consumer

Consumer Group



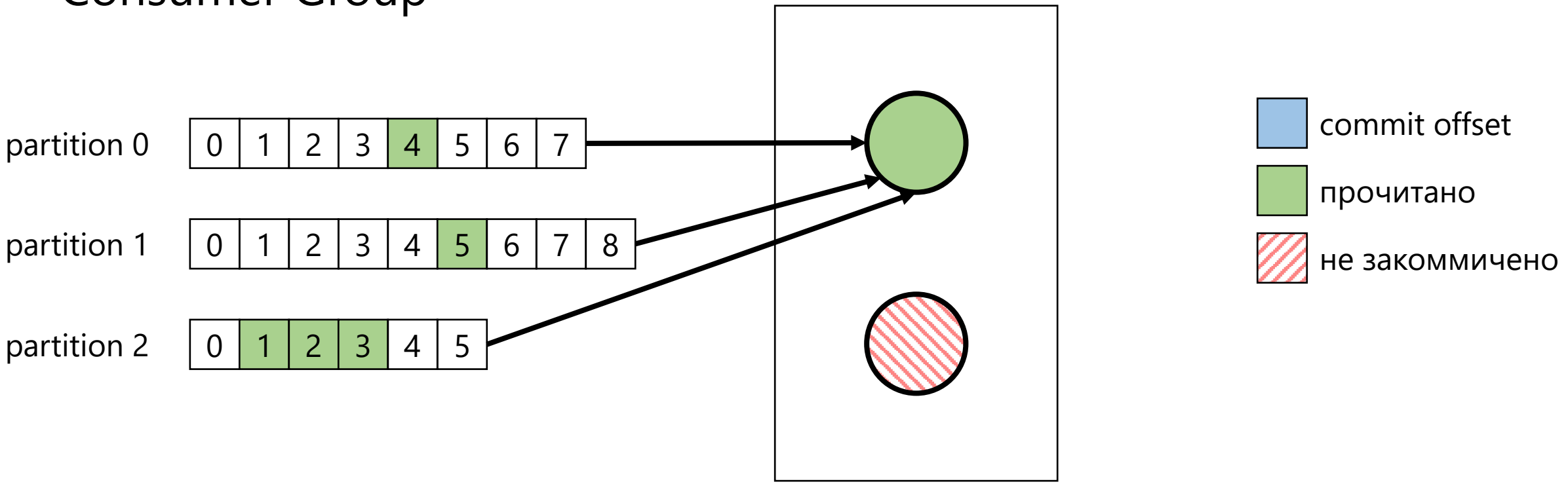
.NET Kafka Consumer

Consumer Group



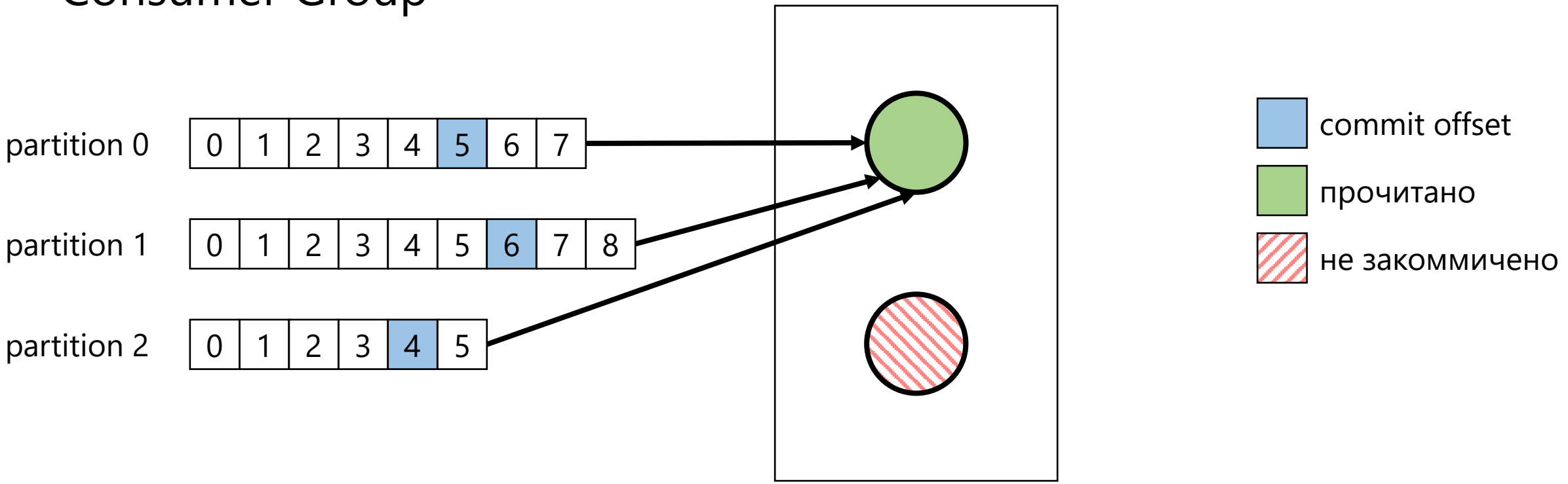
.NET Kafka Consumer

Consumer Group



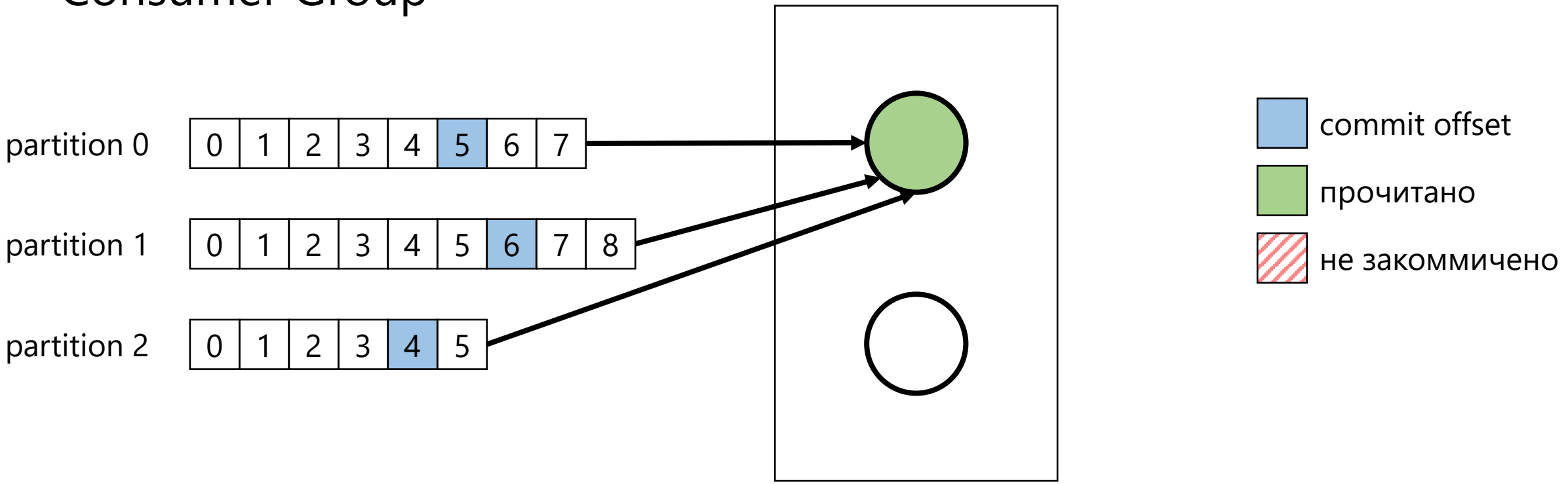
.NET Kafka Consumer

Consumer Group



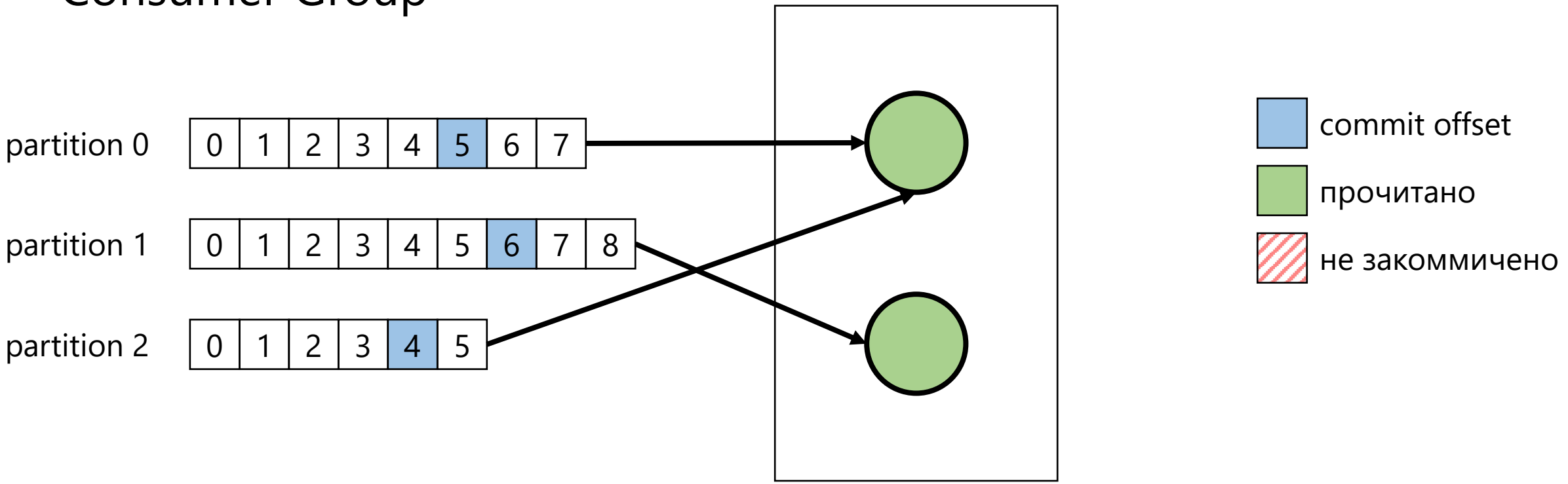
.NET Kafka Consumer

Consumer Group

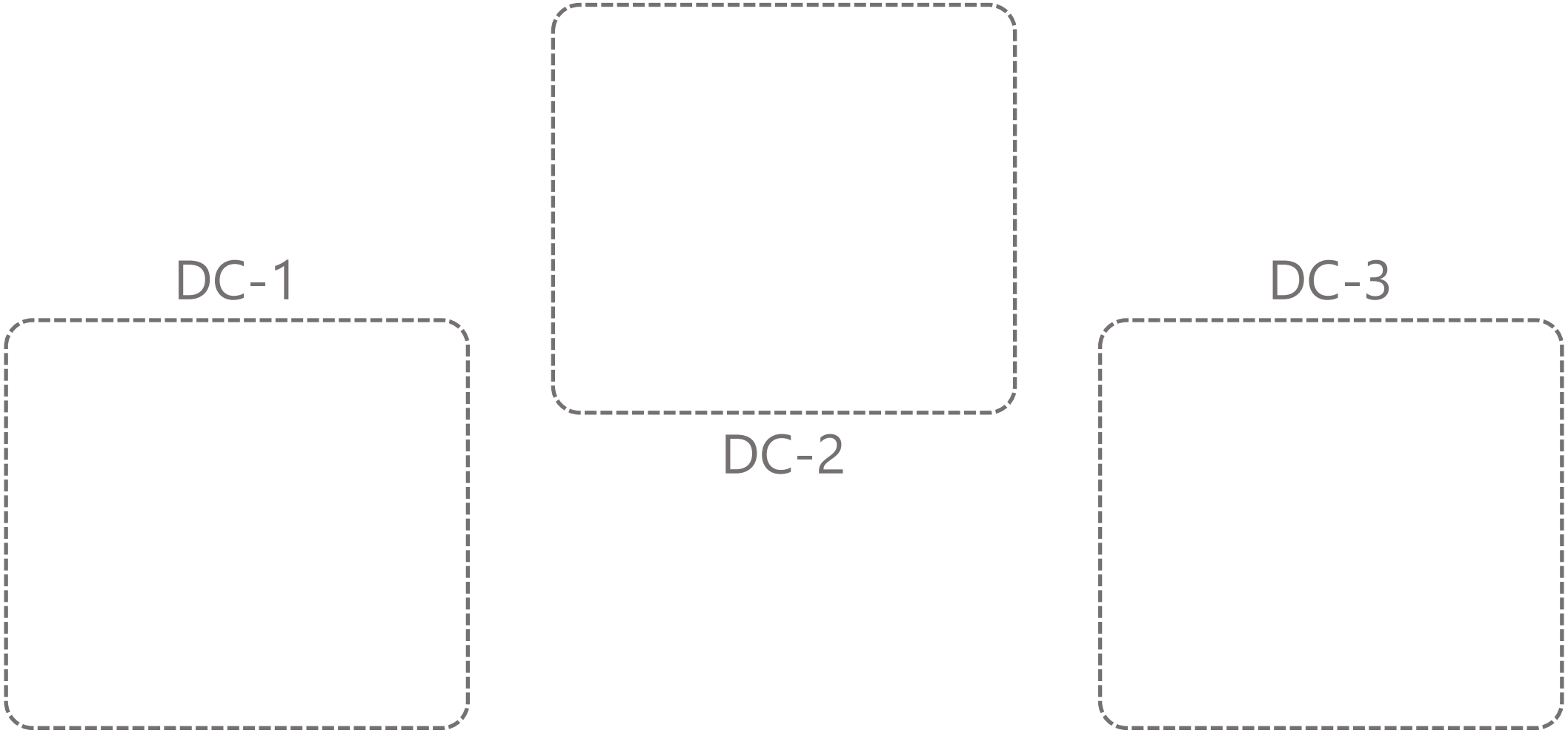


.NET Kafka Consumer

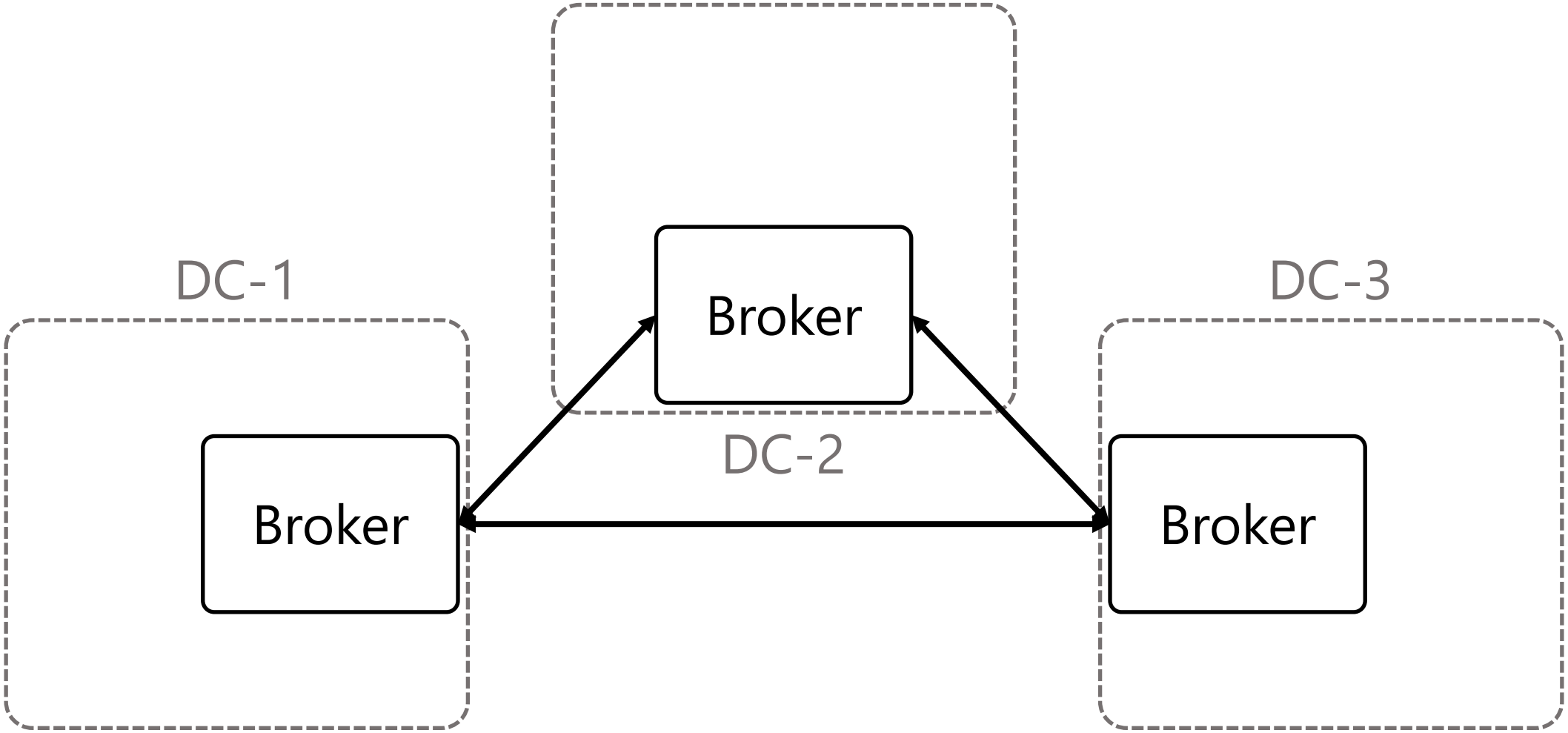
Consumer Group



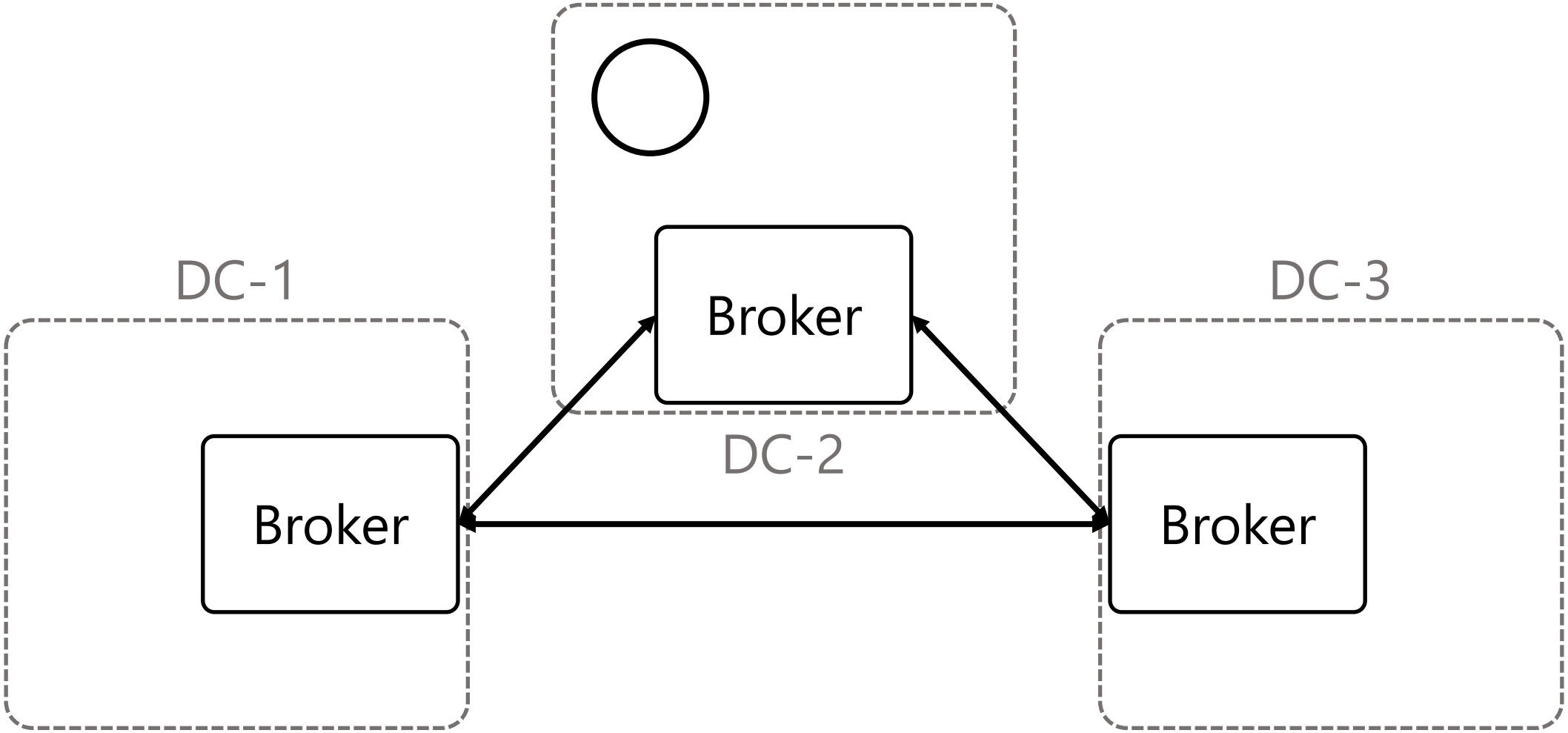
.NET Kafka Consumer



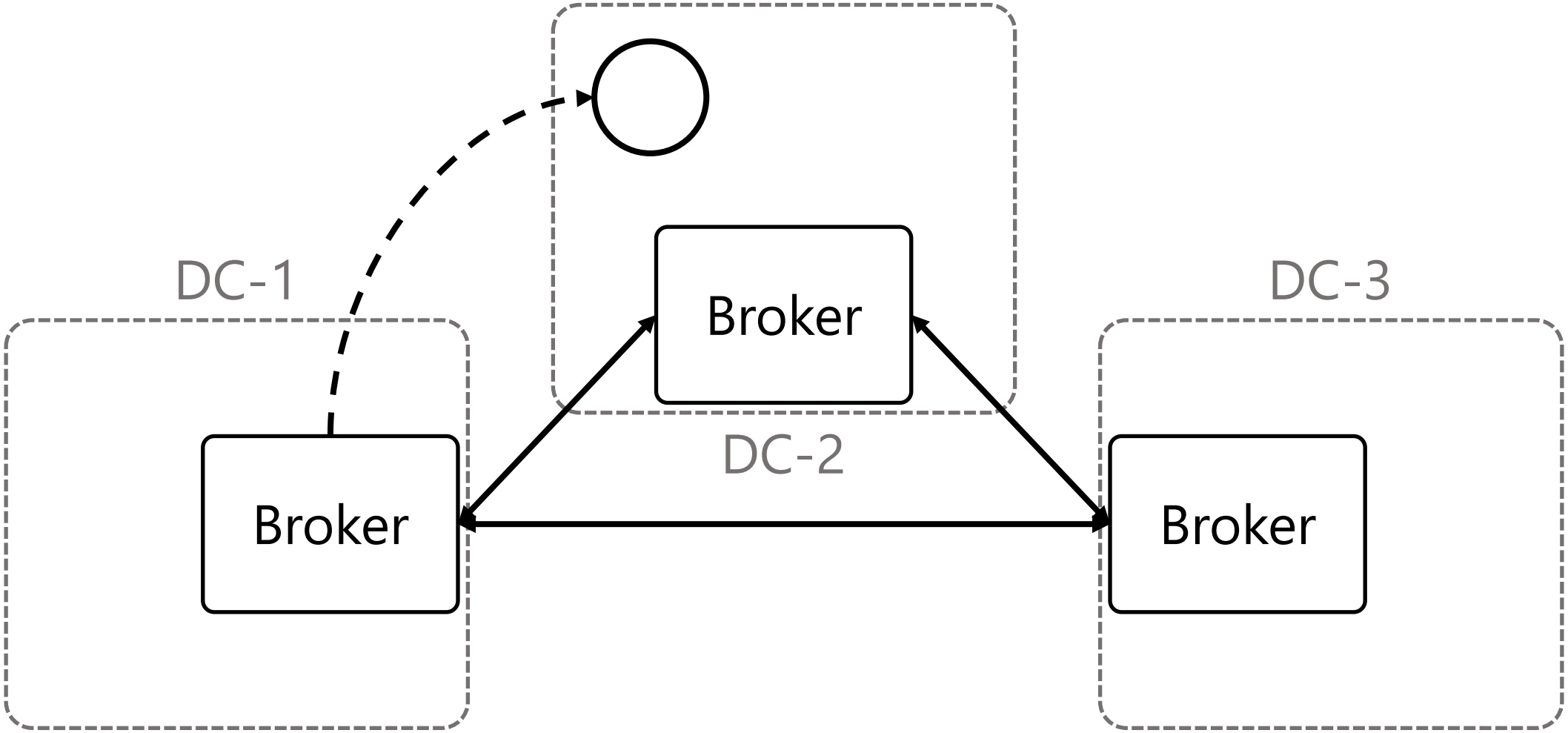
.NET Kafka Consumer



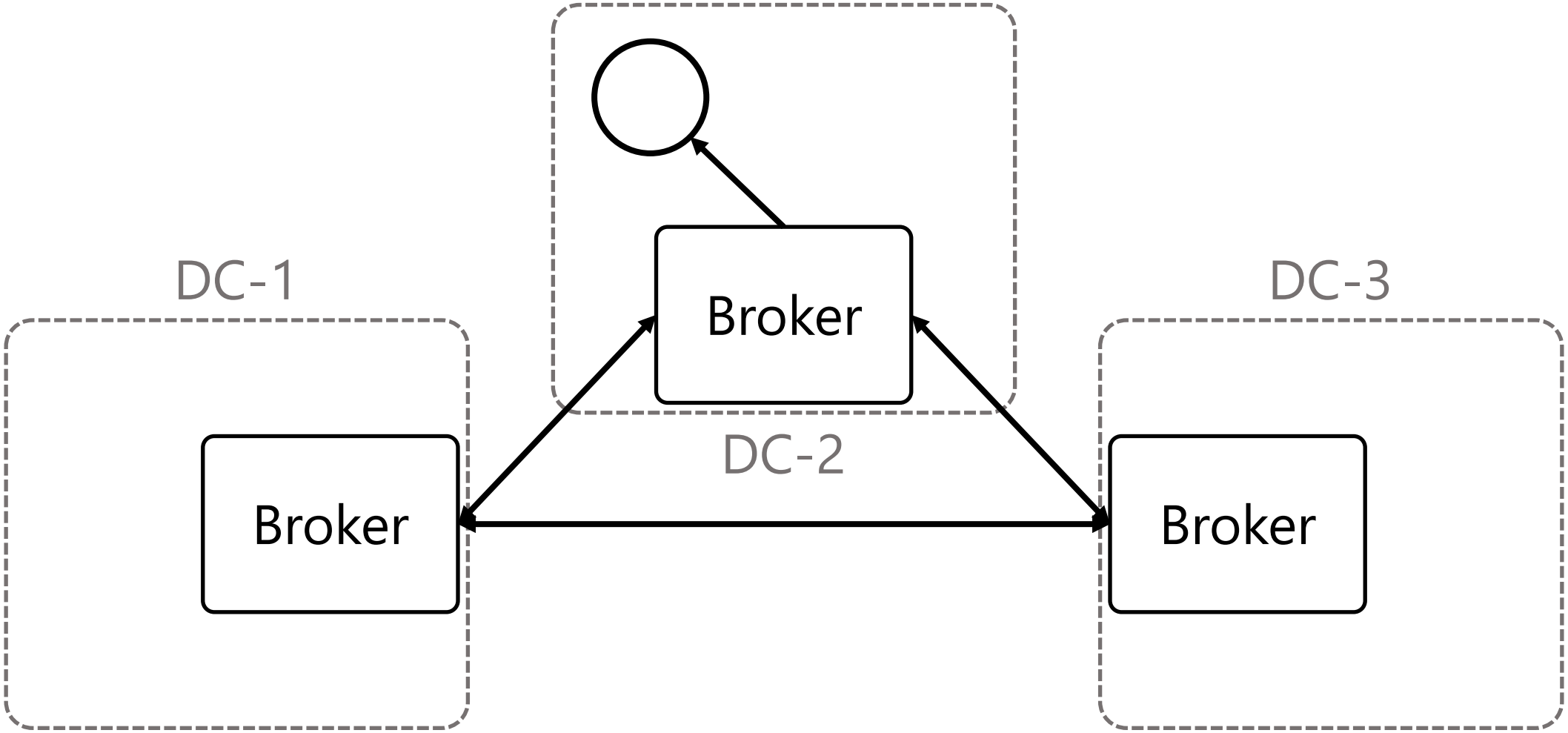
.NET Kafka Consumer



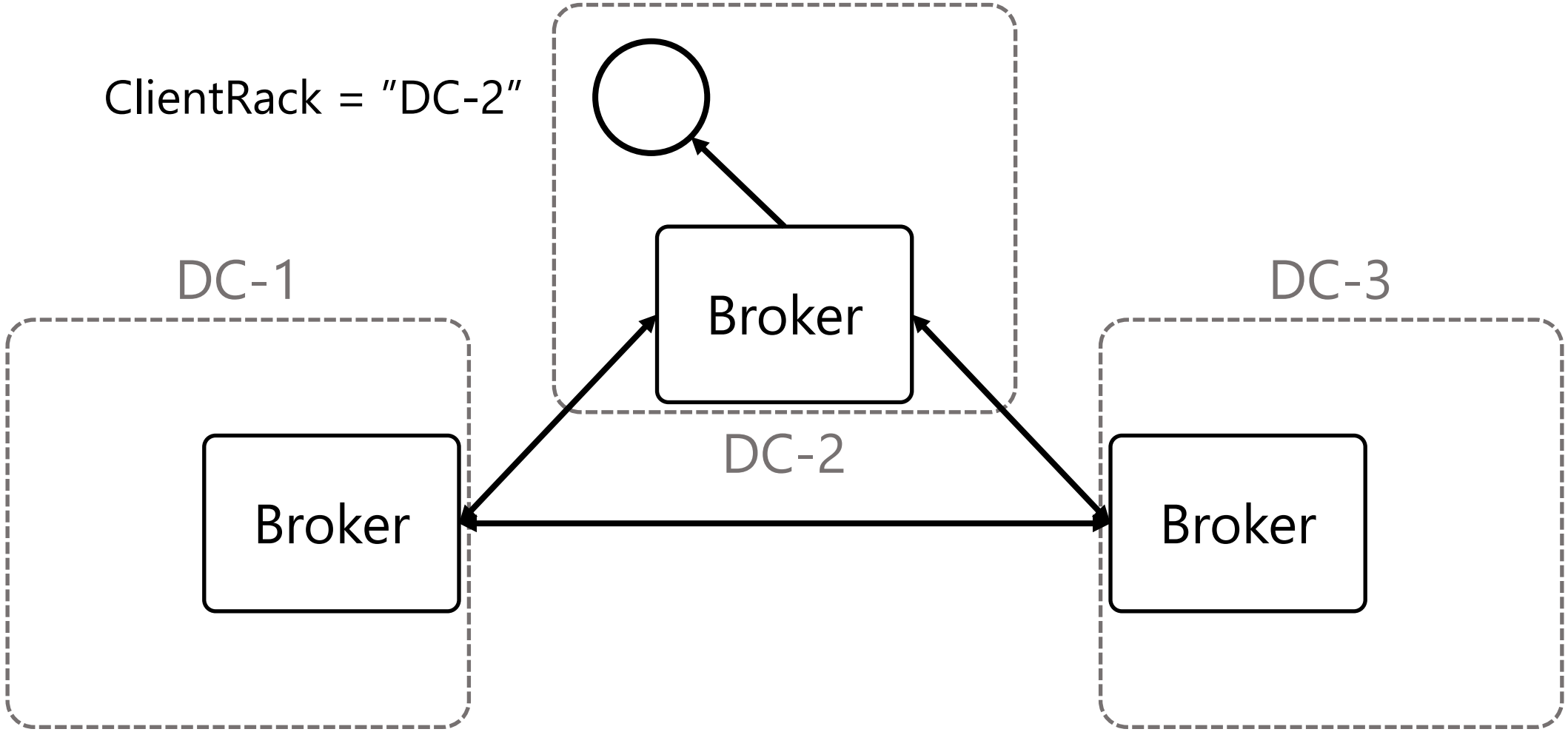
.NET Kafka Consumer



.NET Kafka Consumer



.NET Kafka Consumer



.NET Kafka Consumer

Выводы

— ”Smart” Consumer

— Consumer поллит Кафку

— Consumer отвечает за гарантию обработки

— Автоматический фейловер в Consumer-группе

— Независимая обработка разными Consumer-группами

Преимущества Apache Kafka

Преимущества Apache Kafka

— Персистентность данных

Преимущества Apache Kafka

— Персистентность данных
— Высокая производительность

Преимущества Apache Kafka

— Персистентность данных
— Высокая производительность
— Независимость пайплайнов обработки

Преимущества Apache Kafka

— Персистентность данных
— Высокая производительность
— Независимость пайплайнов обработки
— Возможность «проиграть» историю заново

Преимущества Apache Kafka

— Персистентность данных
— Высокая производительность
— Независимость пайплайнов обработки
— Возможность «проиграть» историю заново
— Гибкость в использовании (благодаря простоте)

Когда Apache Kafka 🔥

— Персистентность данных
— Высокая производительность
— Независимость пайплайнов обработки
— Возможность «проиграть» историю заново
— Гибкость в использовании (благодаря простоте)

Когда Apache Kafka 🔥

— **λ**-архитектура и **K**-архитектура

Когда Apache Kafka 🔥

— **λ**-архитектура и **K**-архитектура

— Стриминг данных

Когда Apache Kafka 🔥

— **λ**-архитектура и **K**-архитектура

— Стриминг БОЛЬШИХ данных

Когда Apache Kafka 🔥

— **λ**-архитектура и **K**-архитектура
— Стриминг БОЛЬШИХ данных
— Много клиентов (Producer и Consumer)

Когда Apache Kafka 🔥

— **λ**-архитектура и **K**-архитектура
— Стриминг БОЛЬШИХ данных
— Много клиентов (Producer и Consumer)
— Требуется кратное масштабирование

Когда Apache Kafka 🔥

— **λ**-архитектура и **K**-архитектура
— Стриминг БОЛЬШИХ данных
— Много клиентов (Producer и Consumer)
— Требуется кратное масштабирование
— Велосипедостроение

Чего нет в Kafka из коробки

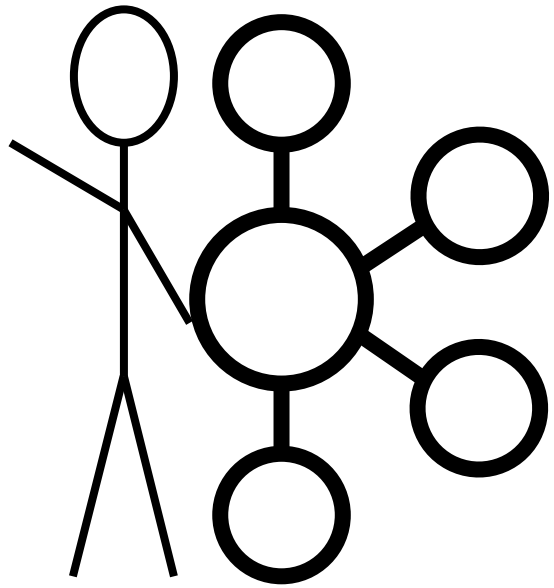
— Отложенные сообщения
— DLQ
— AMQP / MQTT
— TTL на сообщение
— Очереди с приоритетами

GregoryKoshelev

chat_GregoryKoshelev

gnkoshelev

tech.kontur.ru



А что есть ещё?

— RabbitMQ Streams https://www.rabbitmq.com/streams.html
— Apache Pulsar https://pulsar.apache.org
— Apache RocketMQ https://rocketmq.apache.org

Kafka Streams для .NET

Streamiz.Kafka.Net

— https://github.com/LGouellec/kafka-streams-dotnet

— https://lgouellec.github.io/kafka-streams-dotnet/