Apache Kafka
для Python-разработчиков

Григорий Кошелев

Контур

PiterPy 2024

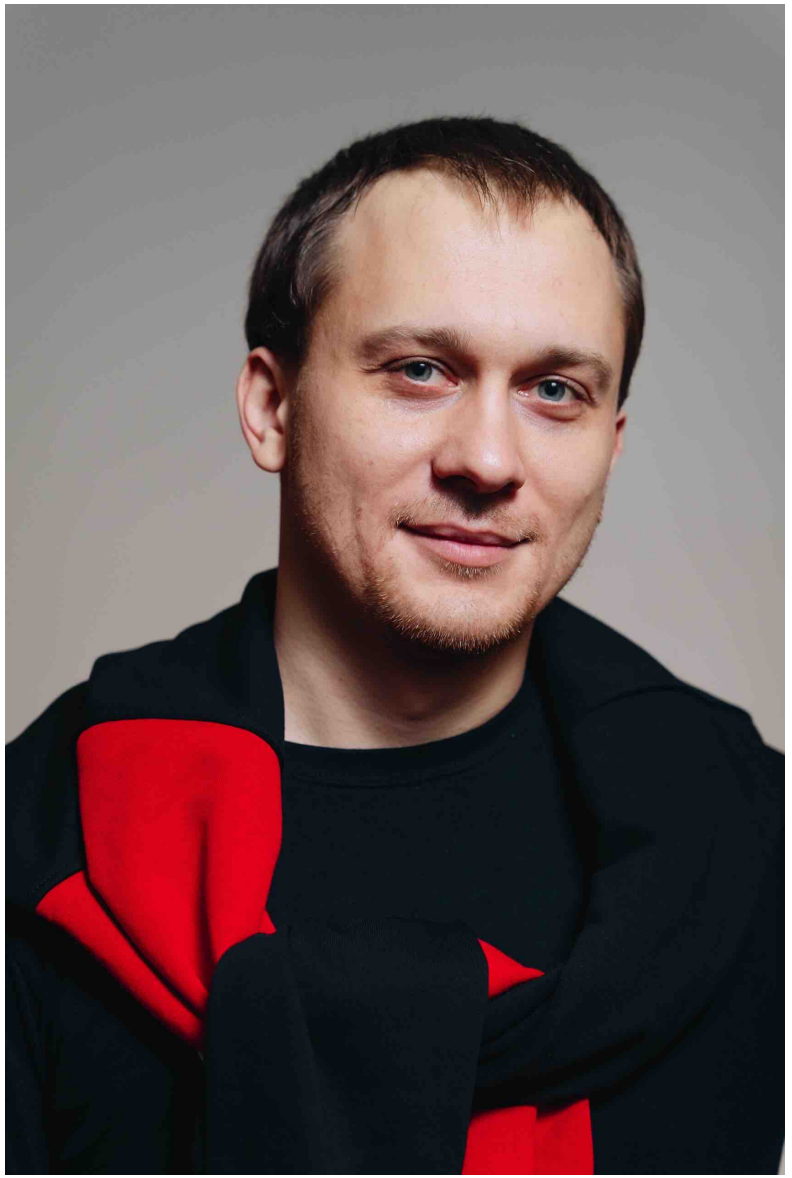
Григорий Кошелев

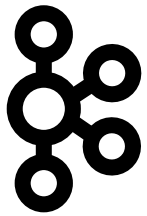
— 10+ лет опыта создания Messaging и Stream Processing систем

— 5+ лет опыта работы с Apache Kafka

— 10+ докладов и выступлений про Apache Kafka

— Автор учебного курса по Кафке





Григорий Кошелев

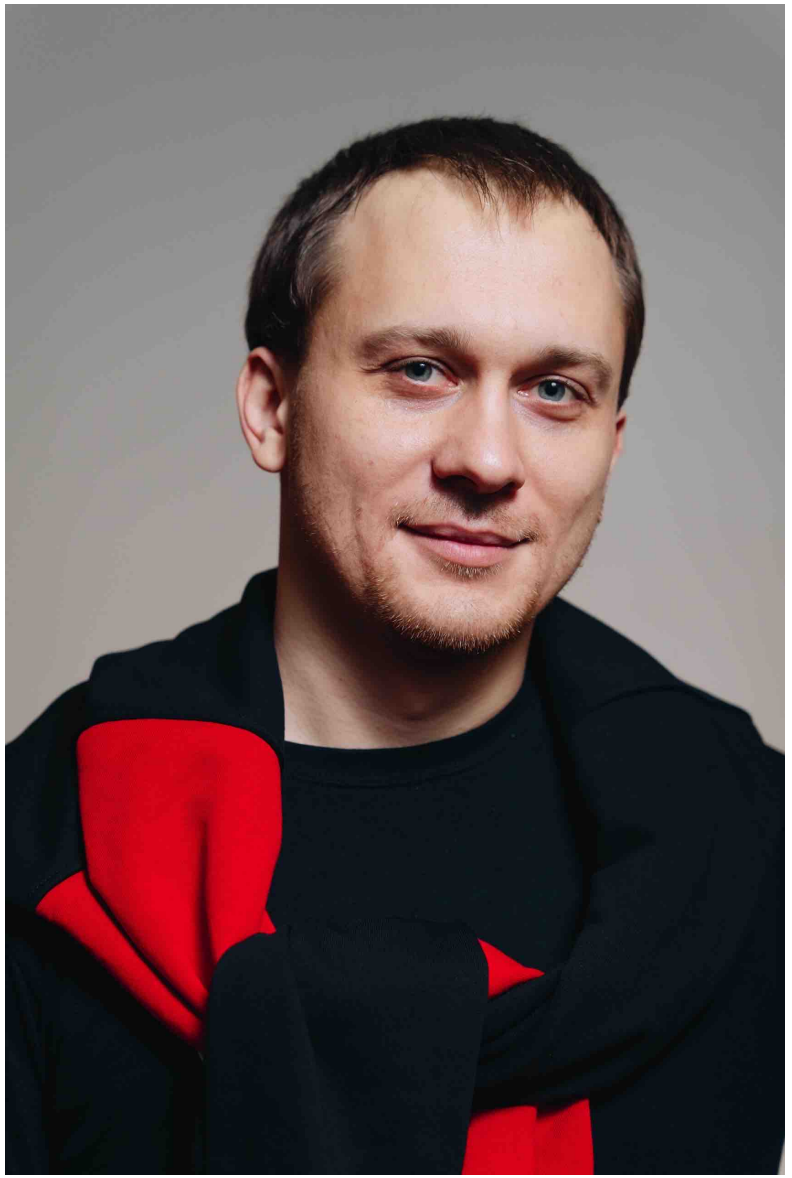
— 10+ лет опыта создания Messaging и Stream Processing систем

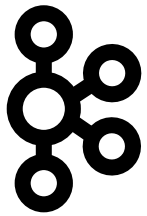
— 5+ лет опыта работы с Apache Kafka

— 10+ докладов и выступлений про Apache Kafka

— Автор учебного курса по Кафке

Основной стек Java/JVM





Григорий Кошелев

— 10+ лет опыта создания Messaging и Stream Processing систем

— 5+ лет опыта работы с Apache Kafka

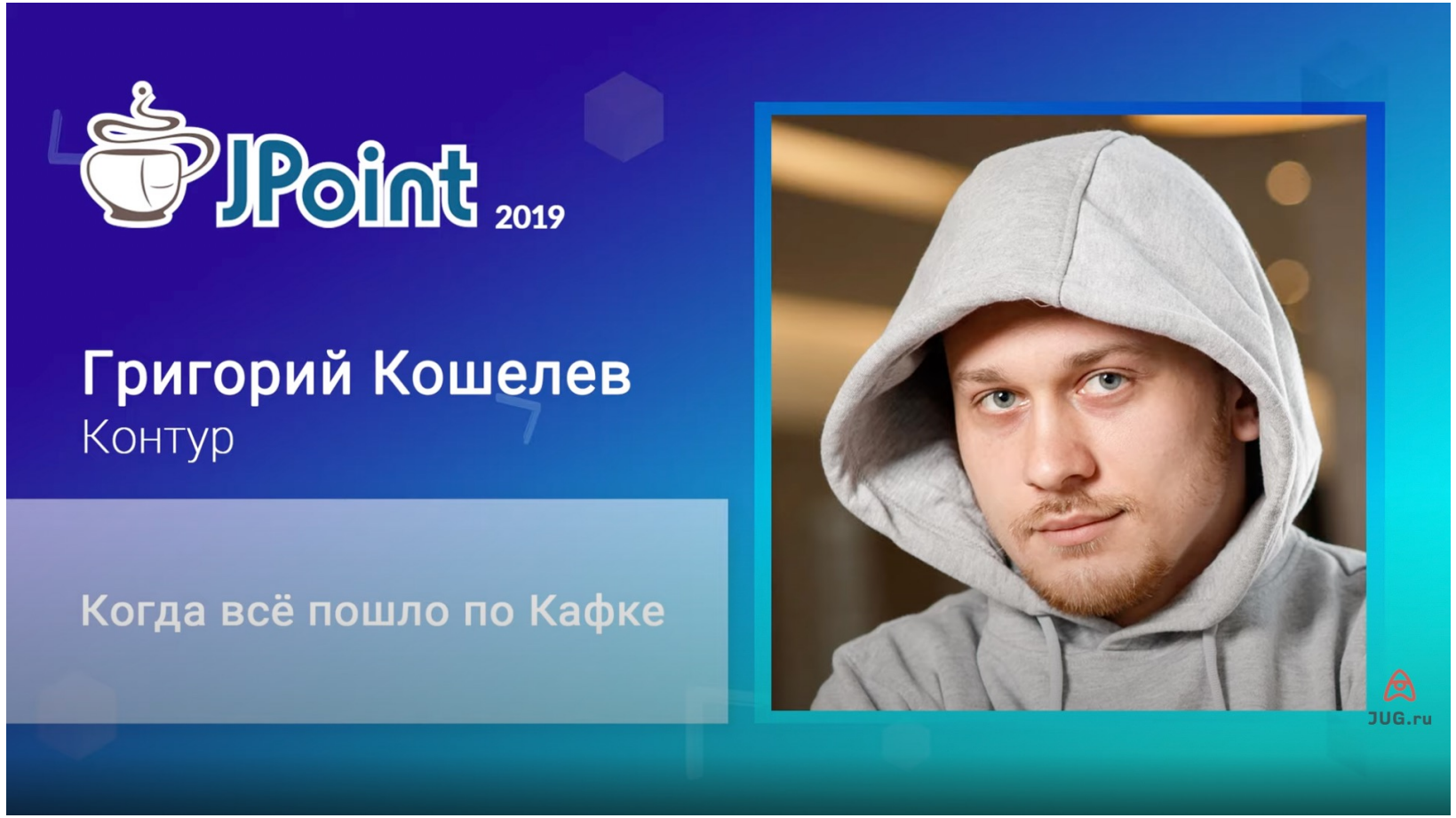
— 10+ докладов и выступлений про Apache Kafka

— Автор учебного курса по Кафке

Основной стек Java/JVM

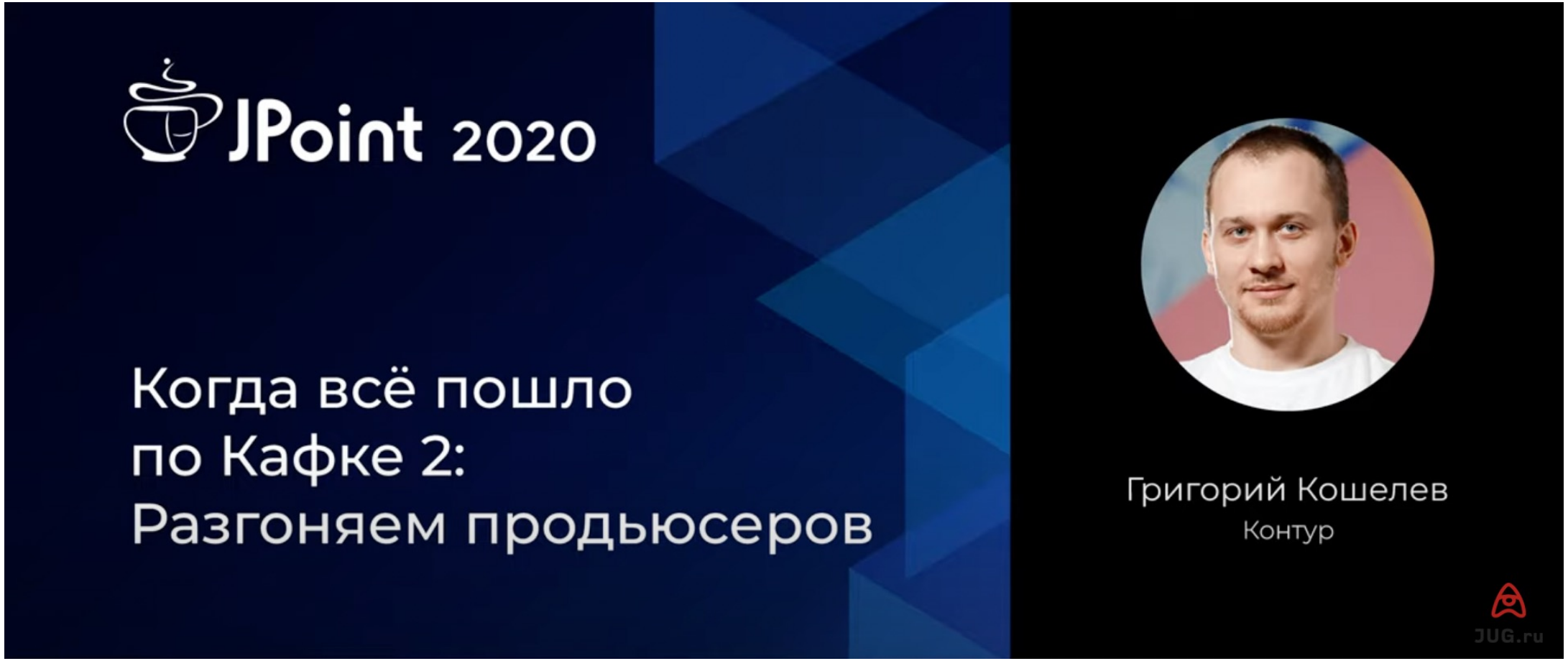
https://t.me/devopsina/5087

Трилогия «Когда всё пошло по Кафке»



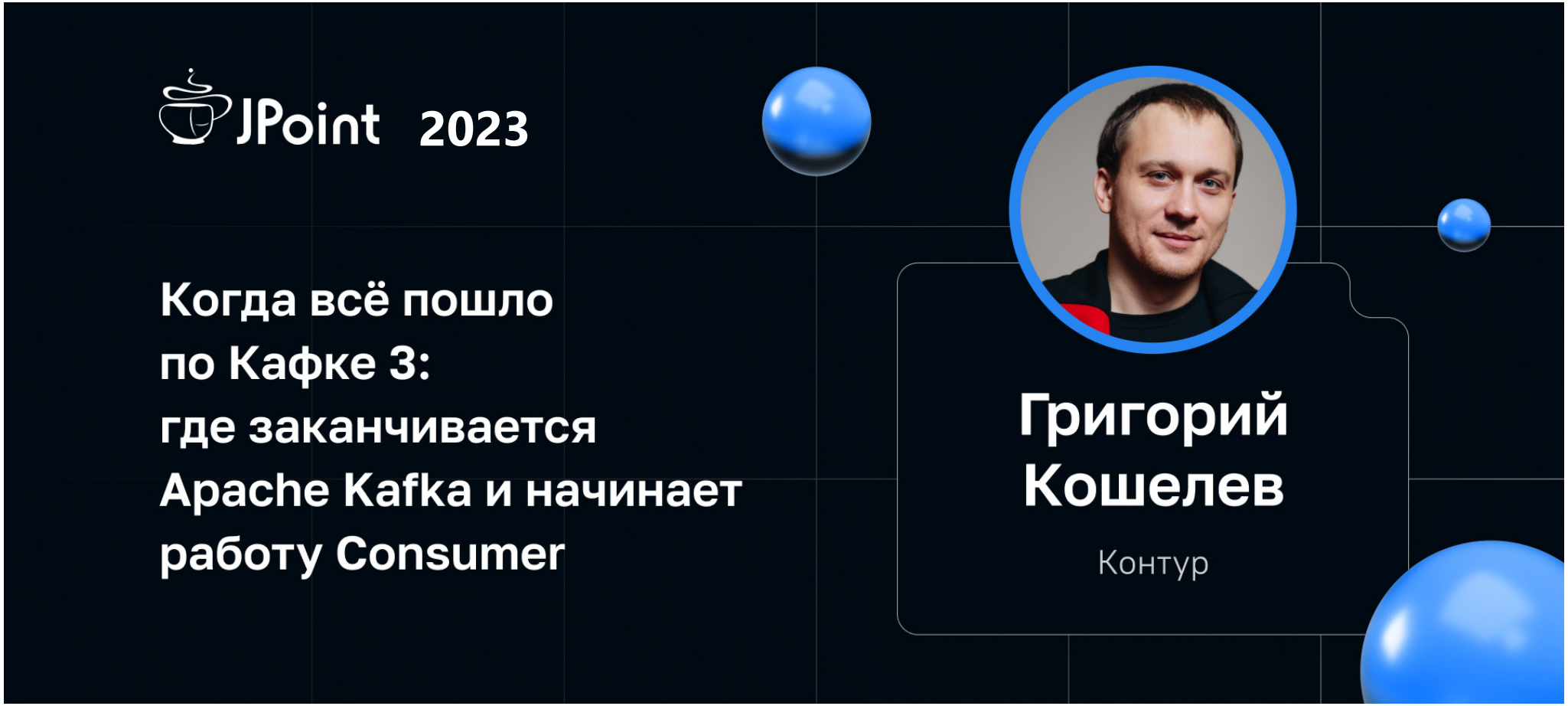
https://www.youtube.com/watch?v=A_yUaPARv8U

Трилогия «Когда всё пошло по Кафке»



https://www.youtube.com/watch?v=zMLfxztAVlo

Трилогия «Когда всё пошло по Кафке»

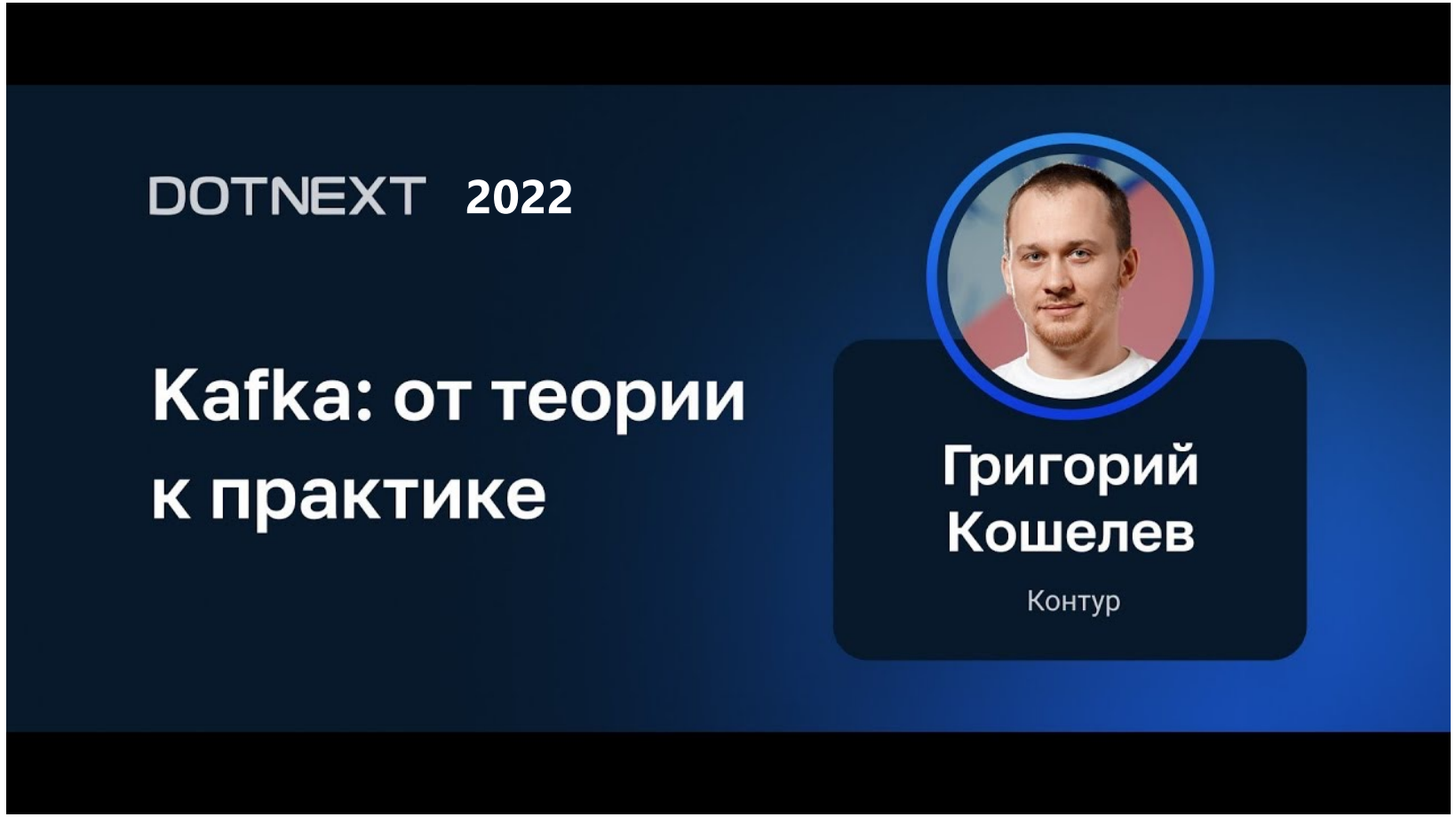


https://www.youtube.com/watch?v=NXU_F_7STSM

Как готовить Кафку, чтобы не пригорало

https://www.youtube.com/watch?v=M3HTM81P-Sg

Apache Kafka для .NET-разработчиков



https://www.youtube.com/watch?v=ghKnX5fuW5s

Про что доклад

1. Введение в Apache Kafka
2. Зоопарк Python-клиентов
3. Kafka Producer
4. Kafka Consumer
5. Примеры сценариев использования
6. Подводные камни и ограничения

Кафка в Контуре

Около 20 кластеров Apache Kafka

Кафка в Контуре

Около 20 кластеров Apache Kafka

По типу окружения

Кафка в Контуре

Около 20 кластеров Apache Kafka

По типу окружения

— Тестовое

Кафка в Контуре

Около 20 кластеров Apache Kafka

По типу окружения

— Тестовое
    * функ. тестирование
    * нагрузочное тестирование

Кафка в Контуре

Около 20 кластеров Apache Kafka

По типу окружения

— Тестовое
    * функ. тестирование
    * нагрузочное тестирование
— Стейджинг

Кафка в Контуре

Около 20 кластеров Apache Kafka

По типу окружения

— Тестовое
    * функ. тестирование
    * нагрузочное тестирование
— Стейджинг
— Продакшен

Кафка в Контуре

Около 20 кластеров Apache Kafka

По типу окружения

— Тестовое
    * функ. тестирование
    * нагрузочное тестирование
— Стейджинг
— Продакшен

По назначению

Кафка в Контуре

Около 20 кластеров Apache Kafka

По типу окружения

— Тестовое
    * функ. тестирование
    * нагрузочное тестирование
— Стейджинг
— Продакшен

По назначению

— Телеметрия

Кафка в Контуре

Около 20 кластеров Apache Kafka

По типу окружения

— Тестовое
    * функ. тестирование
    * нагрузочное тестирование
— Стейджинг
— Продакшен

По назначению

— Телеметрия
— Интеграционная шина

Кафка в Контуре

Около 20 кластеров Apache Kafka

По типу окружения

— Тестовое
    * функ. тестирование
    * нагрузочное тестирование
— Стейджинг
— Продакшен

По назначению

— Телеметрия
— Интеграционная шина
— Шина данных

Кафка в Контуре

Около 20 кластеров Apache Kafka

По типу окружения

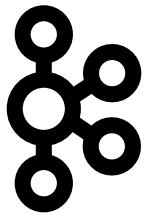
— Тестовое
    * функ. тестирование
    * нагрузочное тестирование
— Стейджинг
— Продакшен

По назначению

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

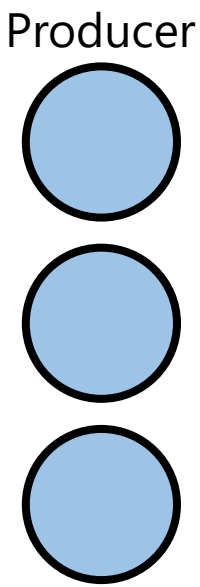
https://www.oreilly.com/radar/questioning-the-lambda-architecture/

Введение в Apache Kafka



Введение в Apache Kafka

Kafka Producer

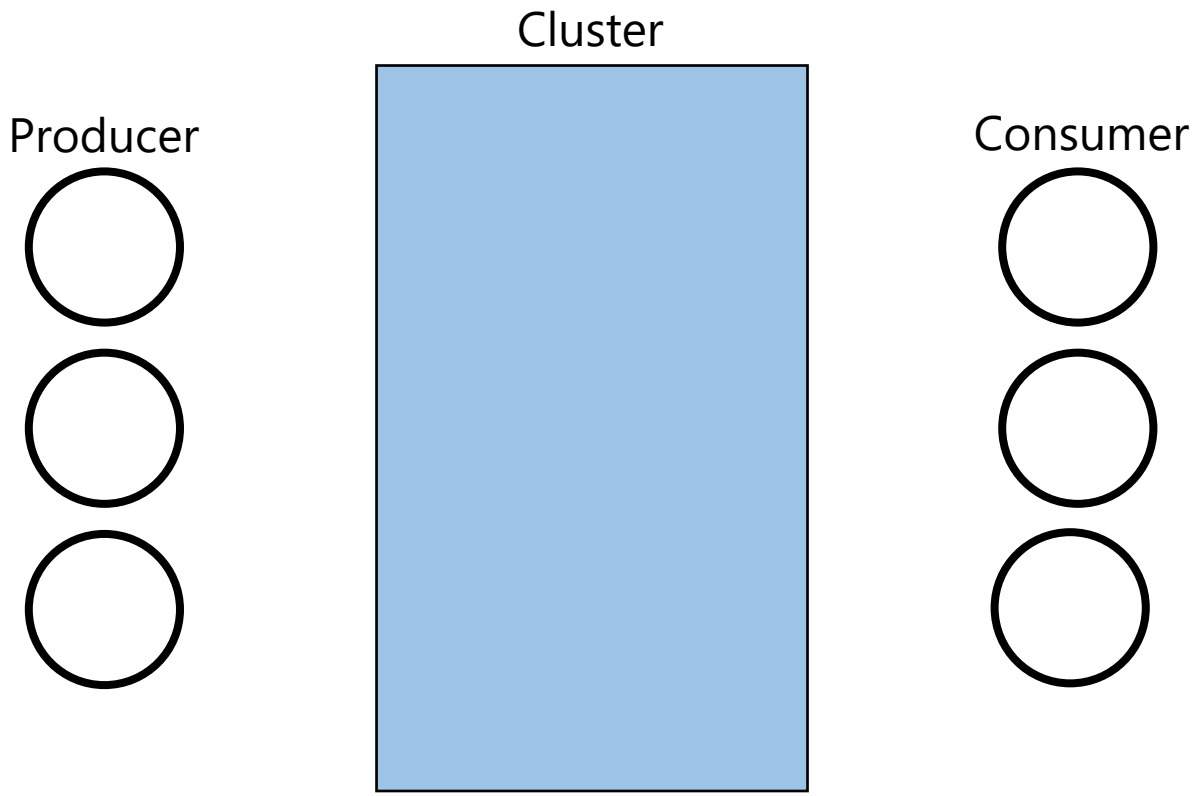


Введение в Apache Kafka

Kafka Consumer

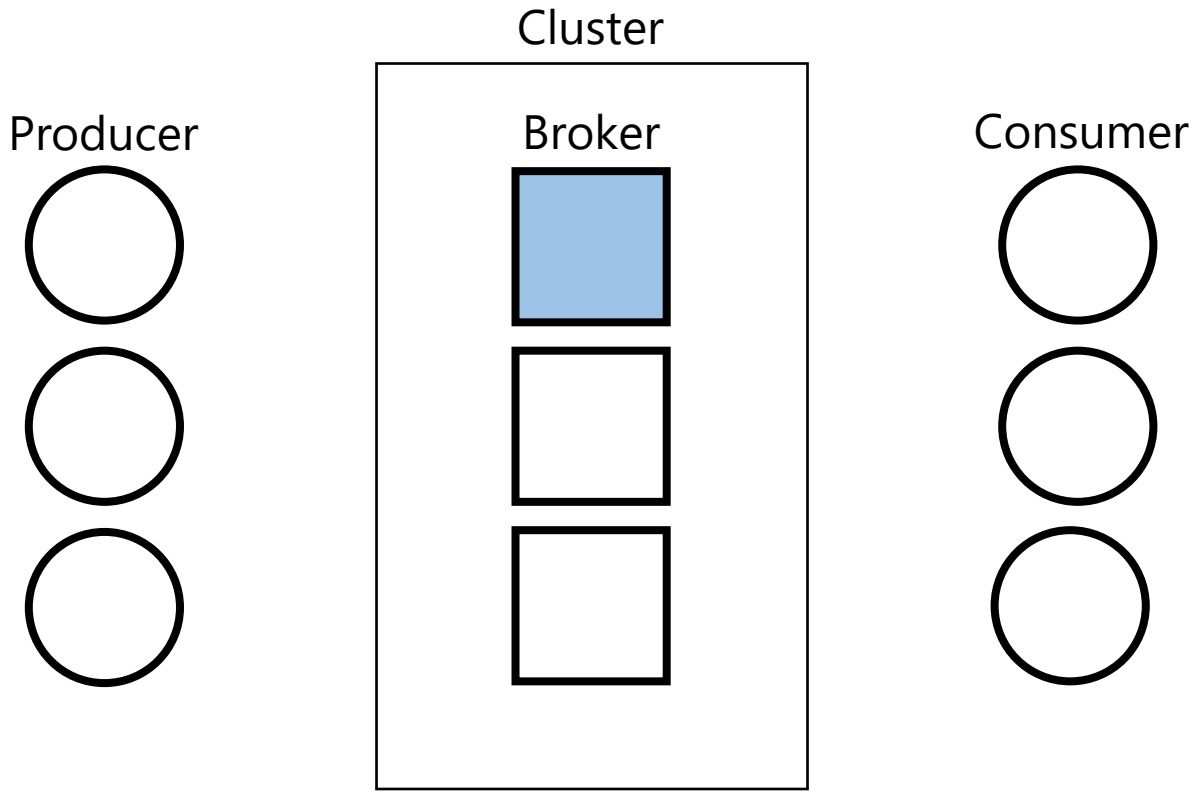
Введение в Apache Kafka

Kafka Cluster

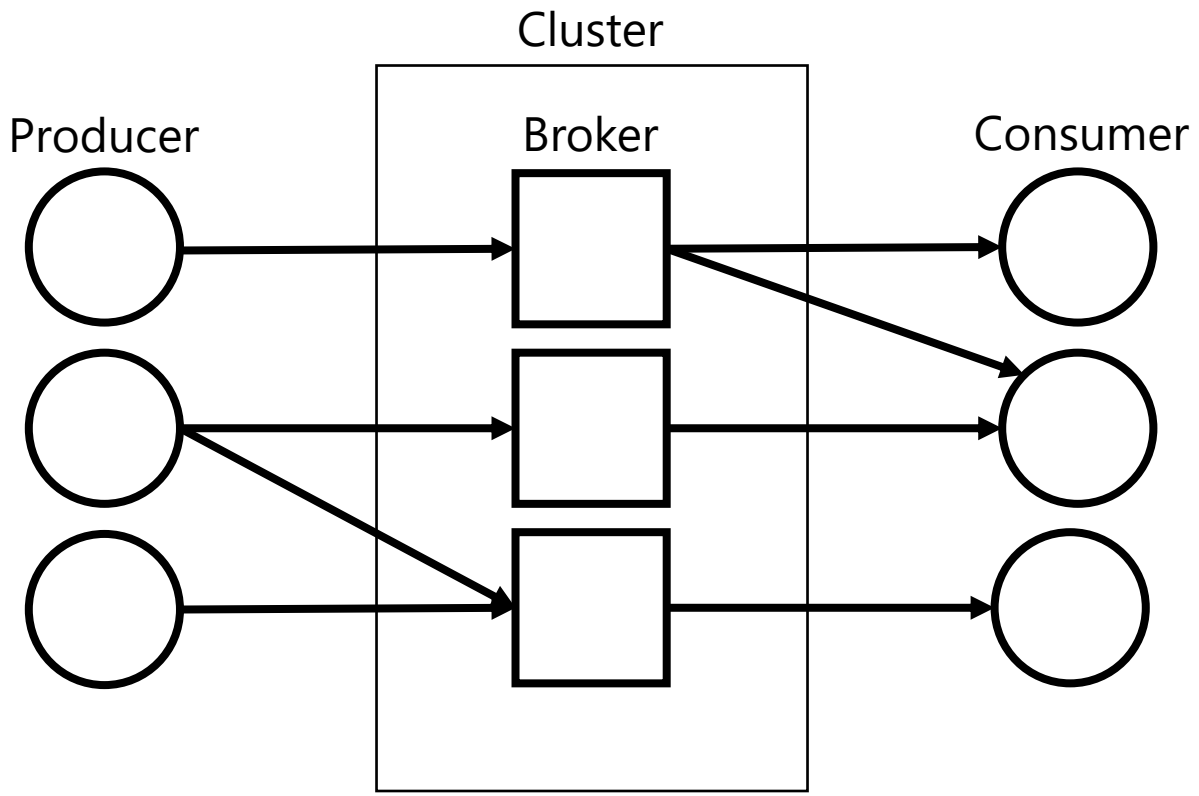


Введение в Apache Kafka

Kafka Broker

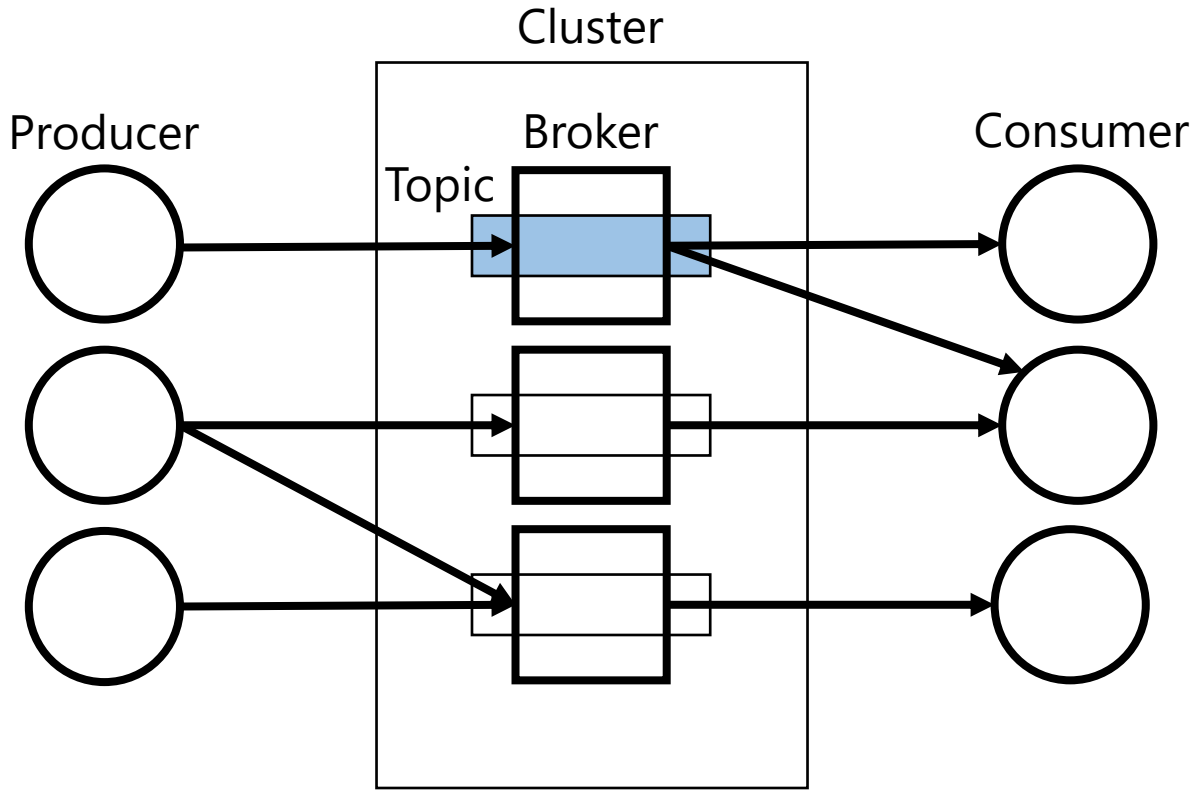


Введение в Apache Kafka

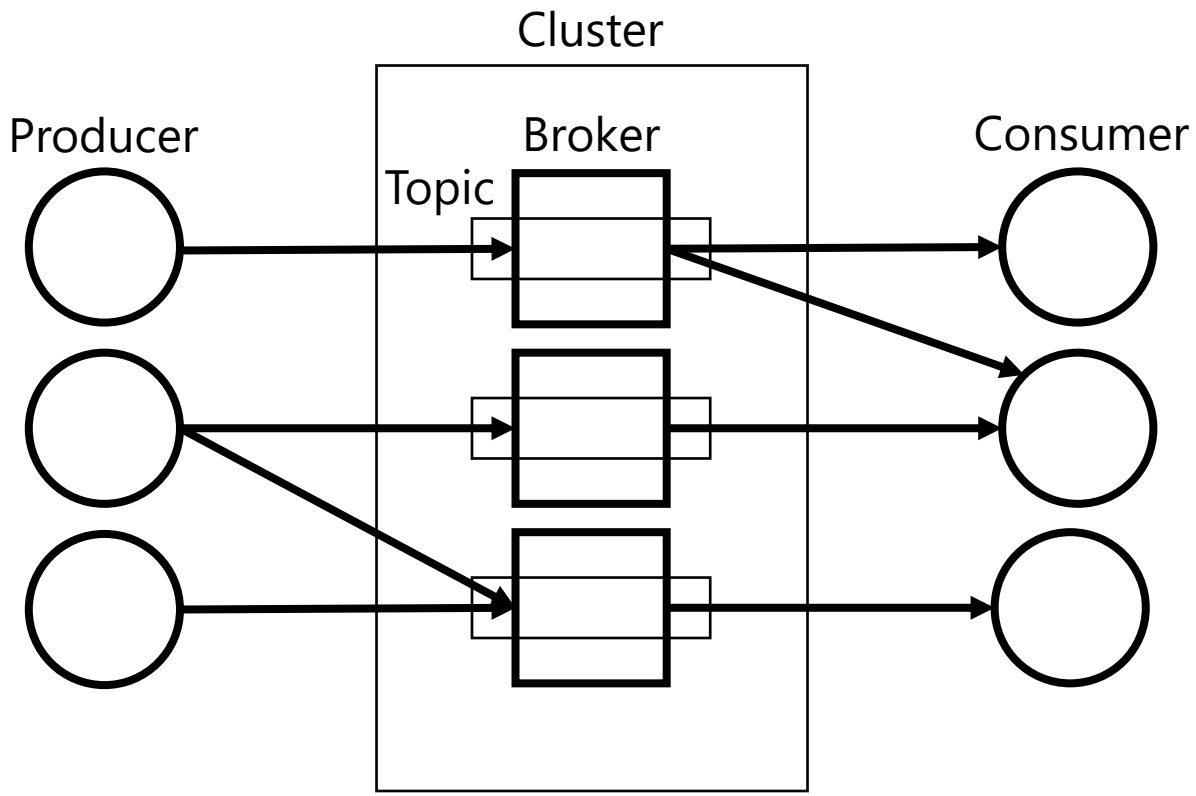


Введение в Apache Kafka

Kafka Topic

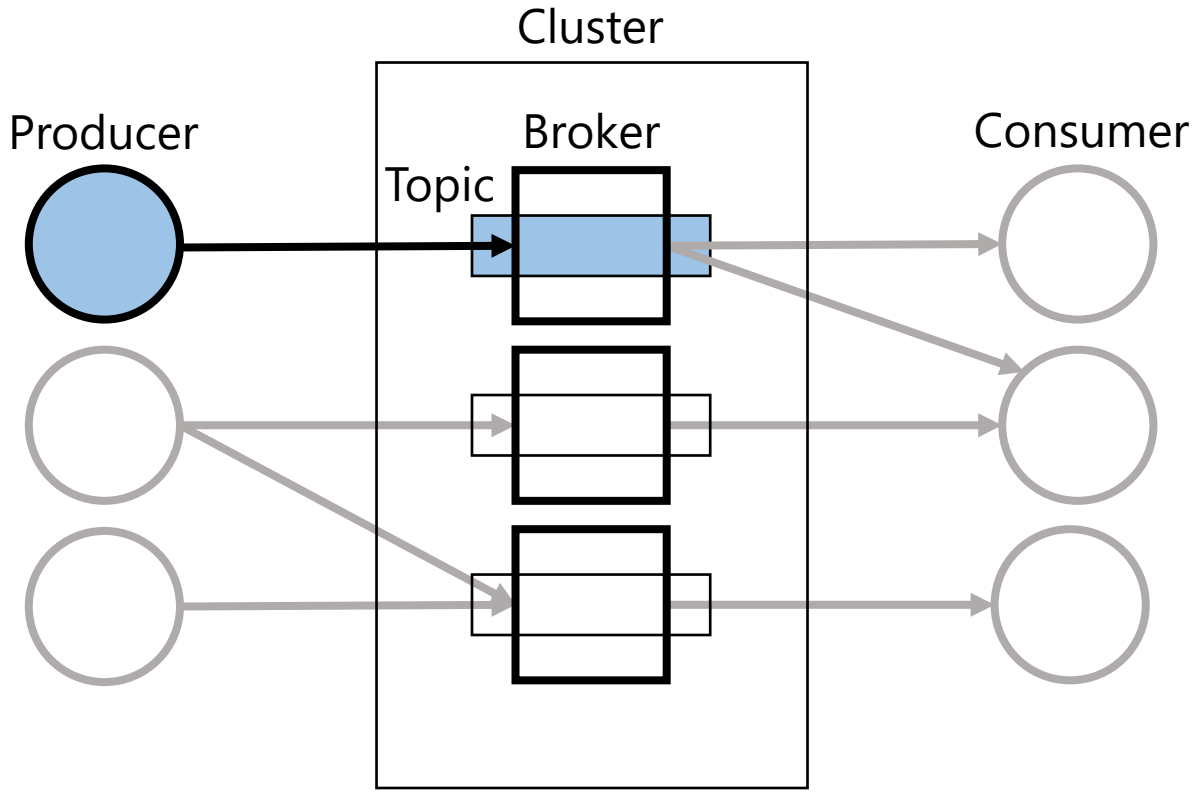


Введение в Apache Kafka



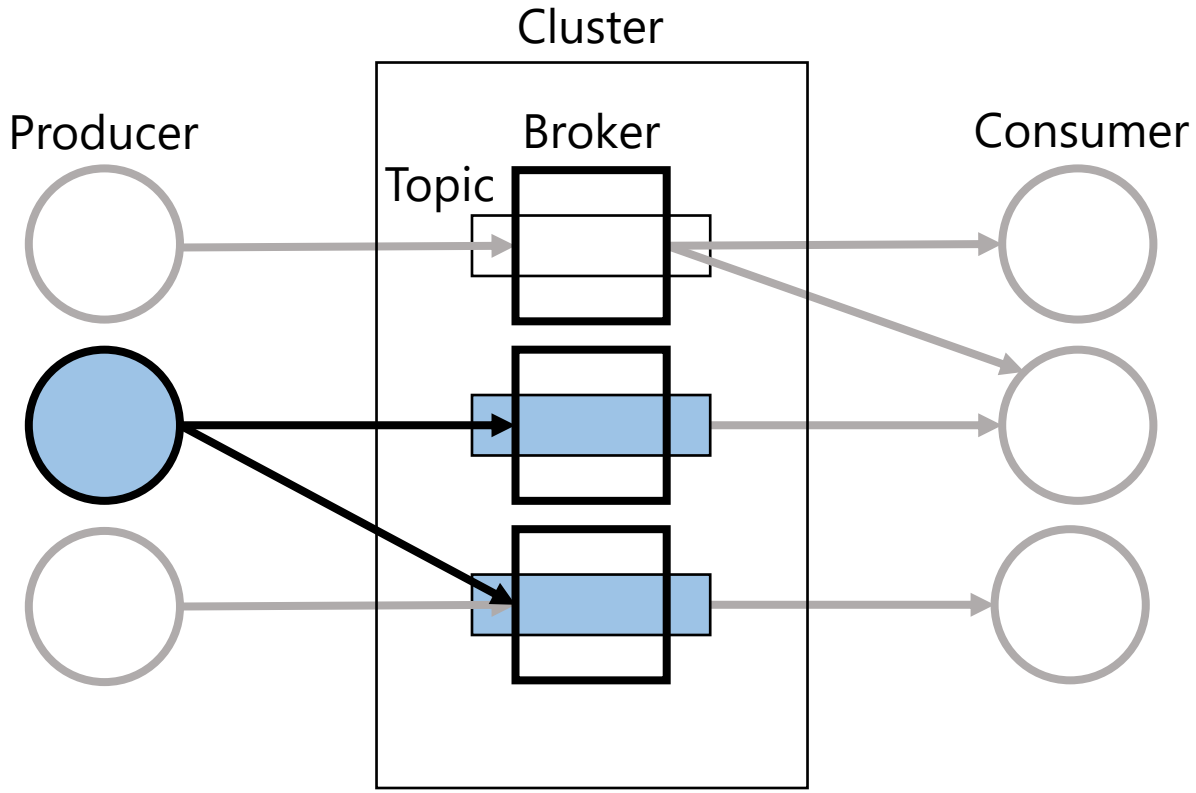
Введение в Apache Kafka

Отправка 1-to-1



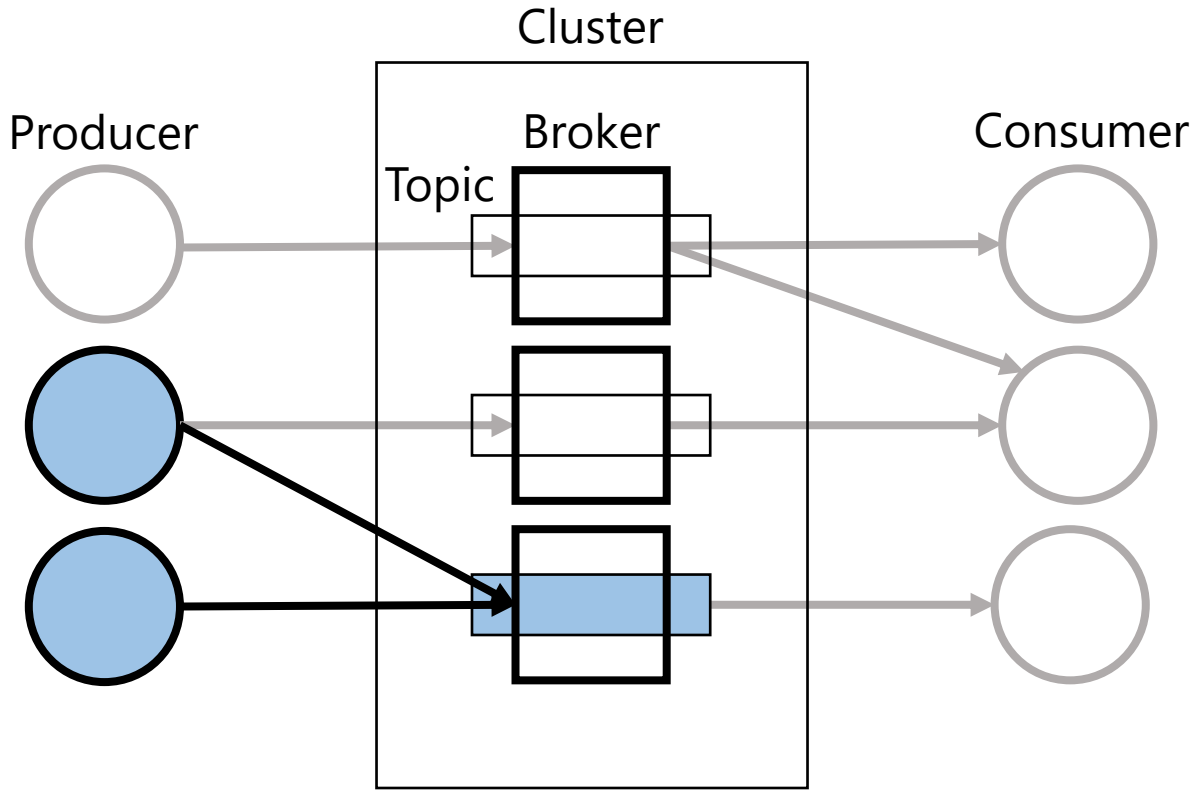
Введение в Apache Kafka

Отправка 1-to-N

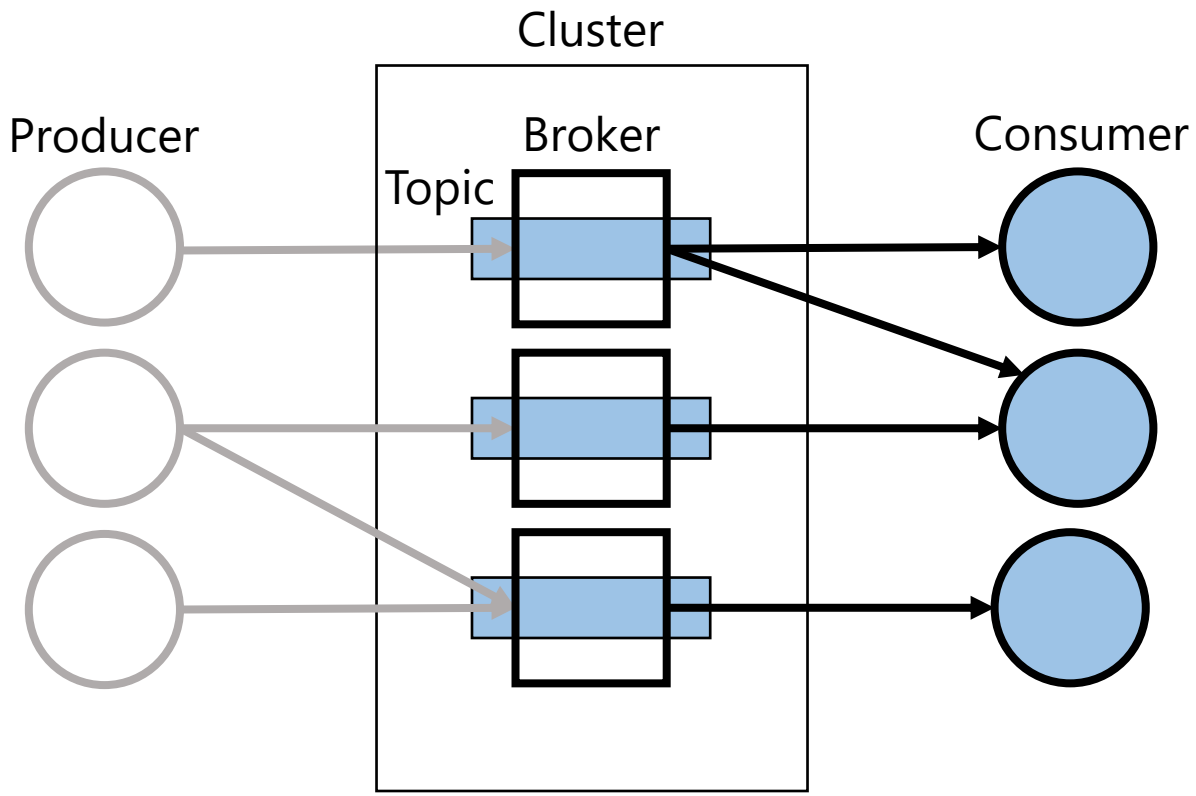


Введение в Apache Kafka

Отправка N-to-1

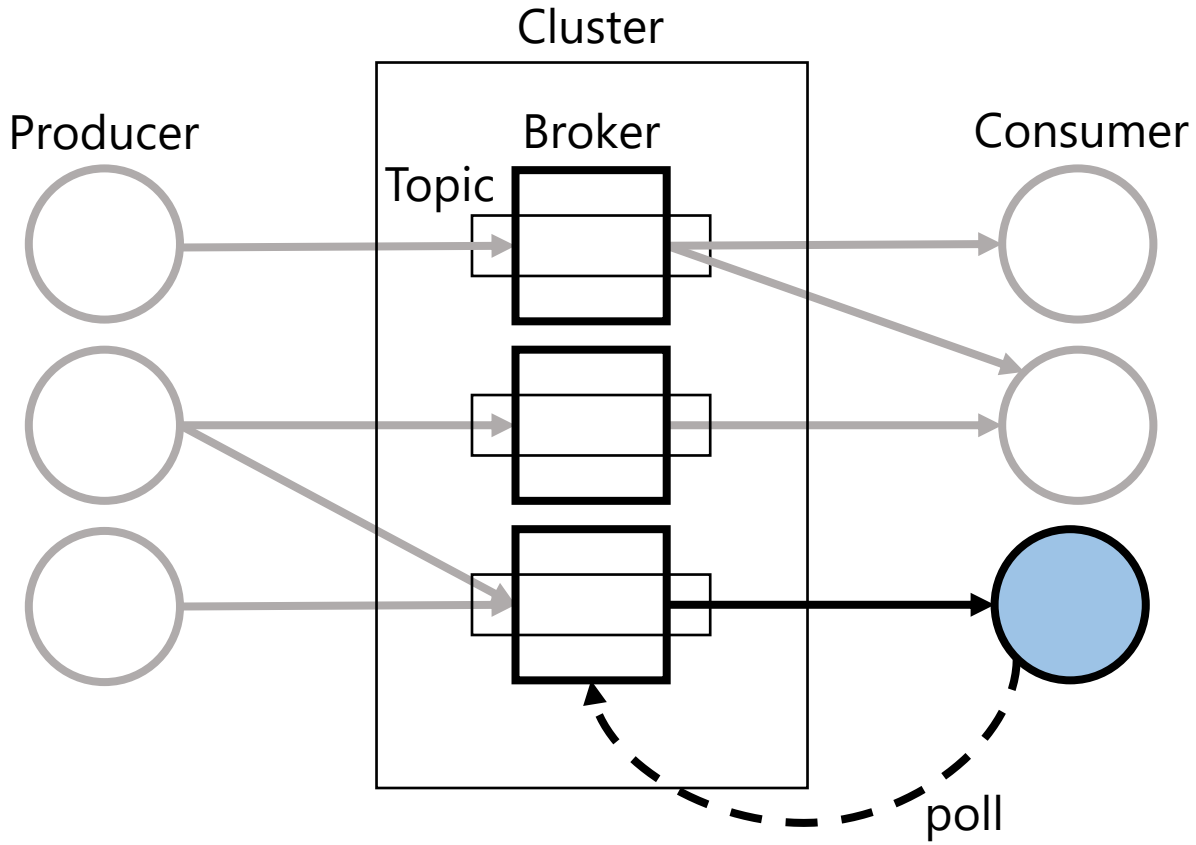


Введение в Apache Kafka



Введение в Apache Kafka

Poll-механика чтения

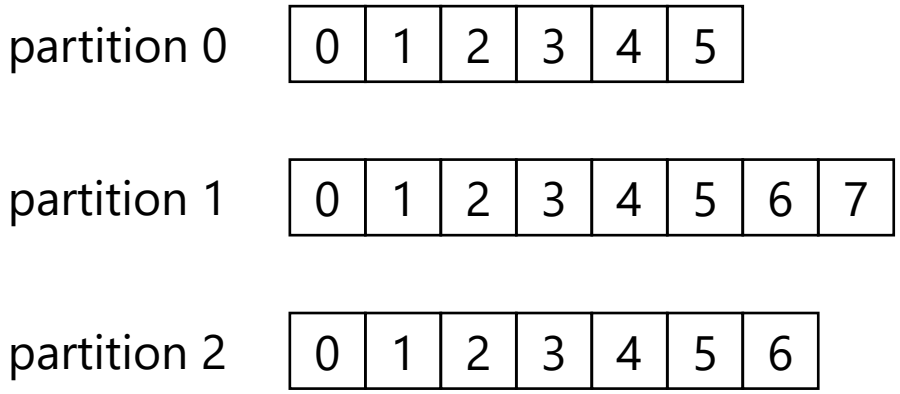


Архитектура Apache Kafka

— Topic
— Broker
— Producer
— Consumer

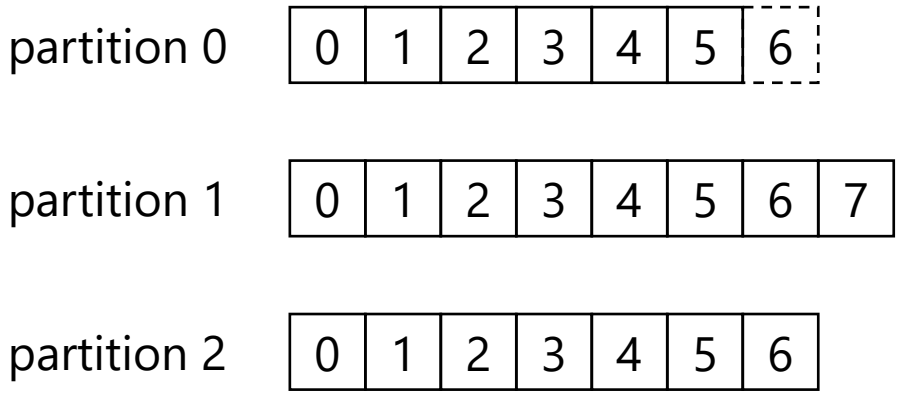
Архитектура Kafka Topic

topic = {partition}



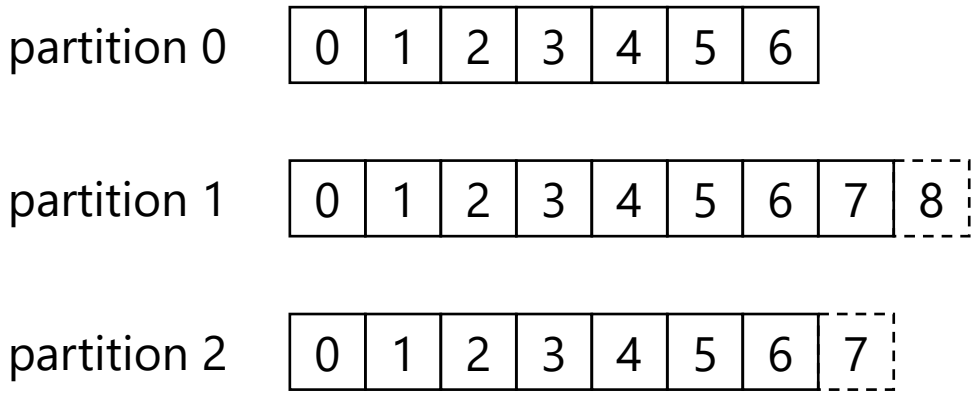
Архитектура Kafka Topic

topic = {partition}



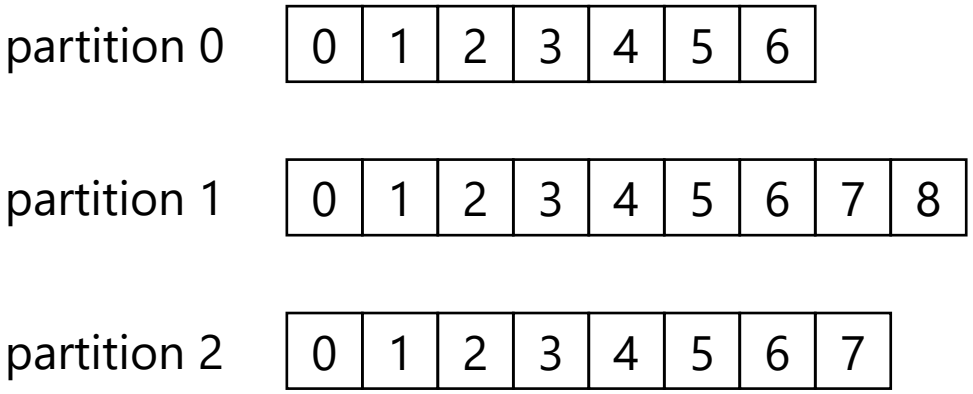
Архитектура Kafka Topic

topic = {partition}



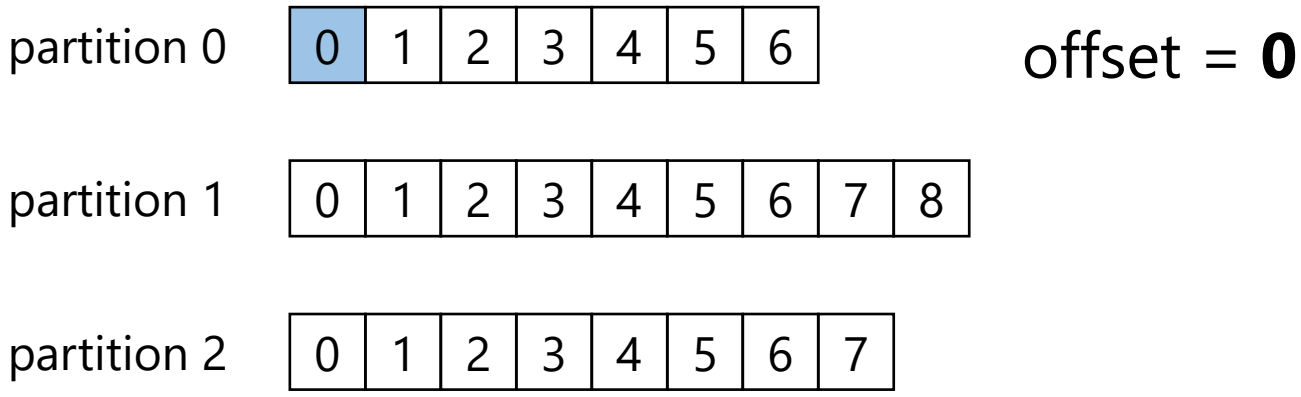
Архитектура Kafka Topic

topic = {partition}



Архитектура Kafka Topic

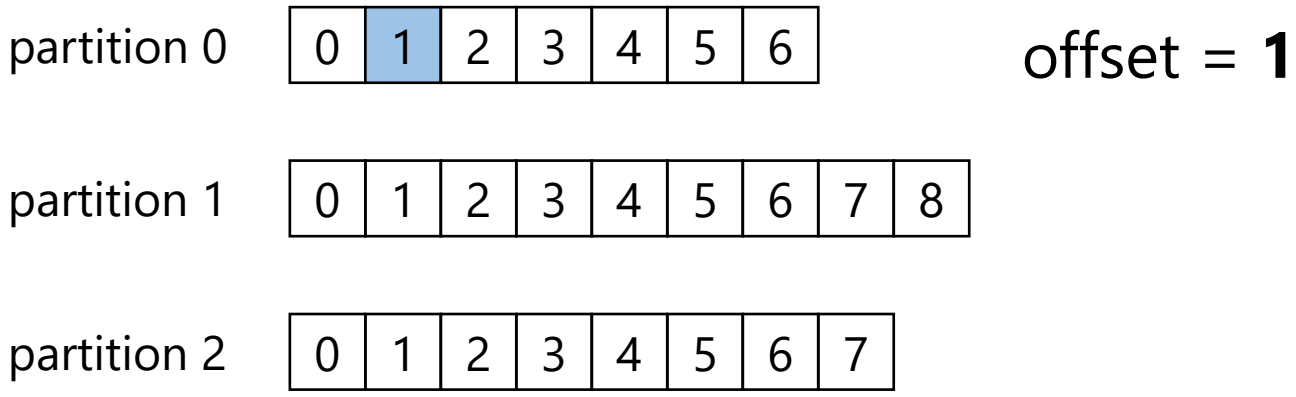
topic = {partition}



offset = **0**

Архитектура Kafka Topic

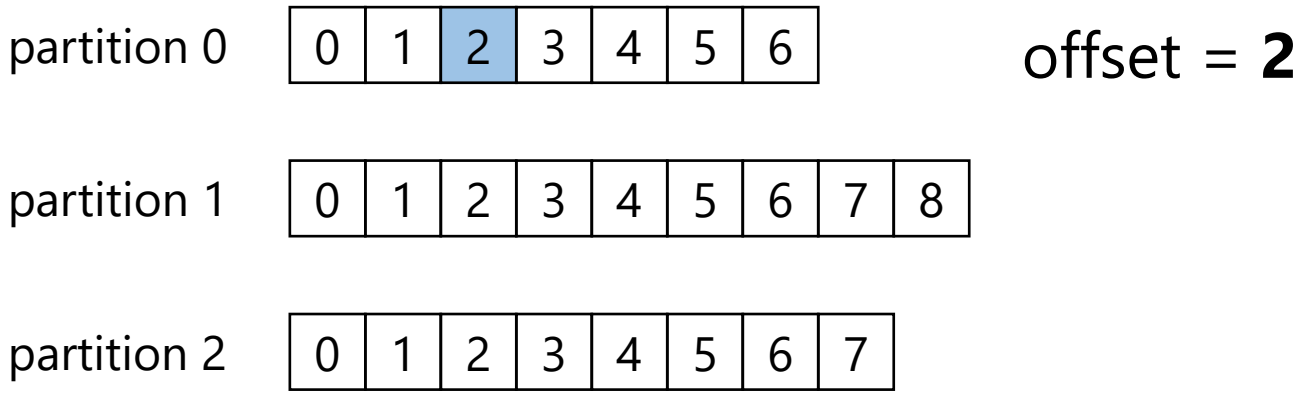
topic = {partition}



offset = **1**

Архитектура Kafka Topic

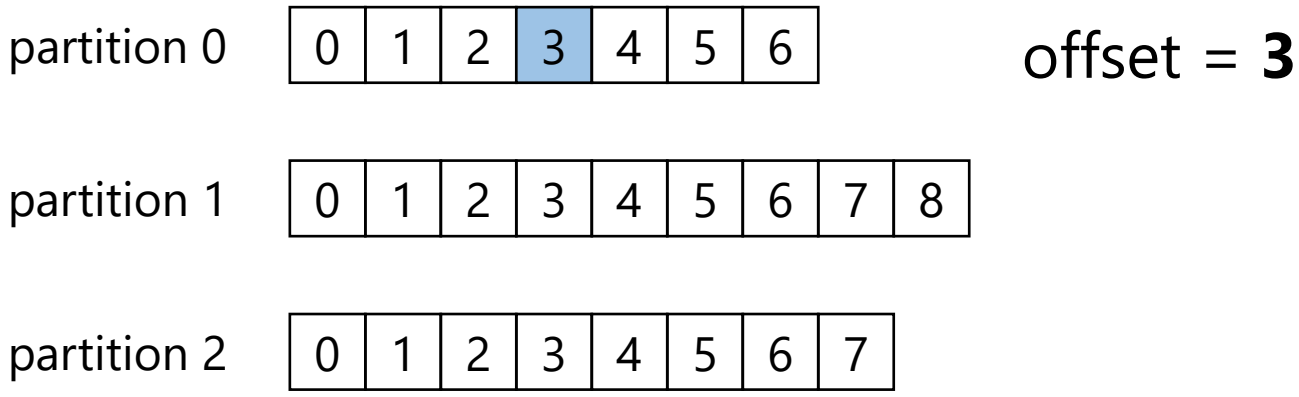
topic = {partition}



offset = **2**

Архитектура Kafka Topic

topic = {partition}



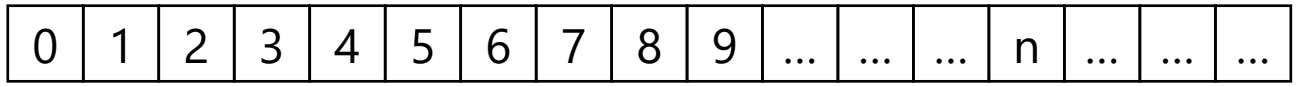
offset = **3**

Архитектура Kafka Topic

partition = {segment}

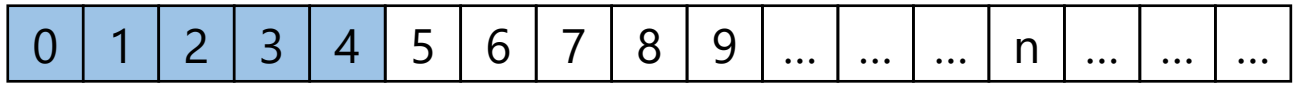
Архитектура Kafka Topic

partition = {segment}



Архитектура Kafka Topic

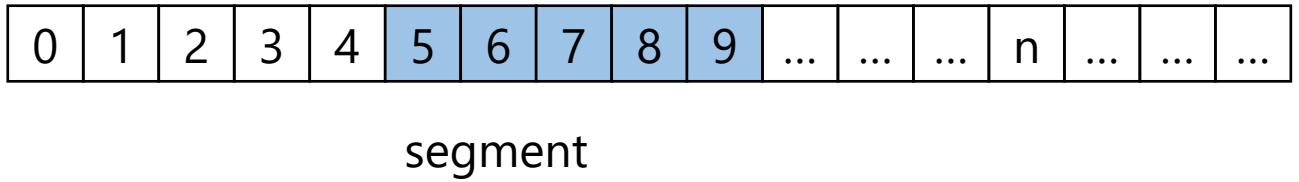
partition = {segment}



segment

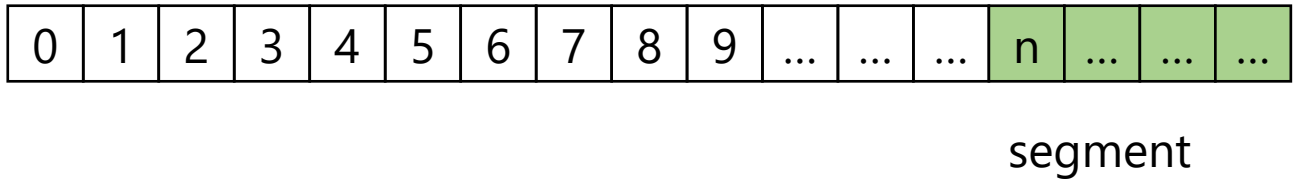
Архитектура Kafka Topic

partition = {segment}



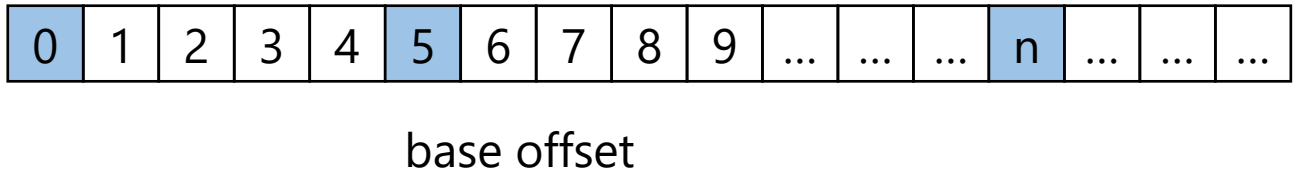
Архитектура Kafka Topic

partition = {segment}



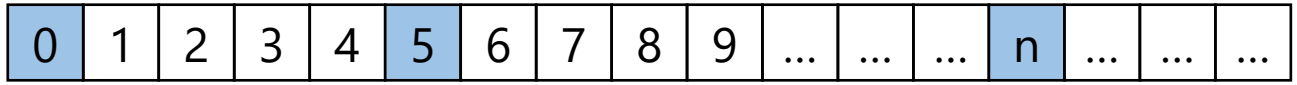
Архитектура Kafka Topic

partition = {segment}



Архитектура Kafka Topic

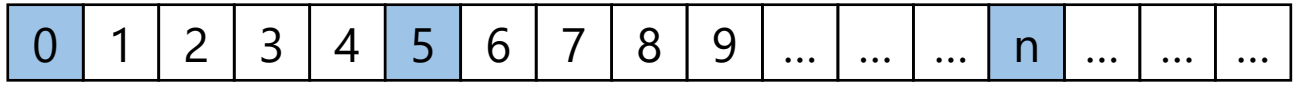
partition = {segment}




```
retention.bytes=-1
retention.ms=604800000
```

Архитектура Kafka Topic

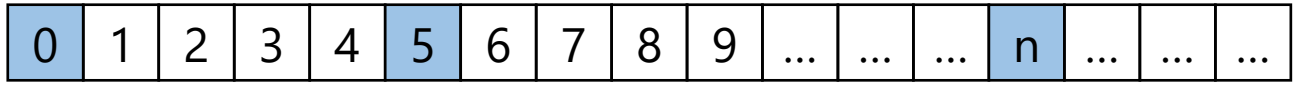
partition = {segment}




```
retention.bytes=-1
retention.ms=604800000
```

Архитектура Kafka Topic

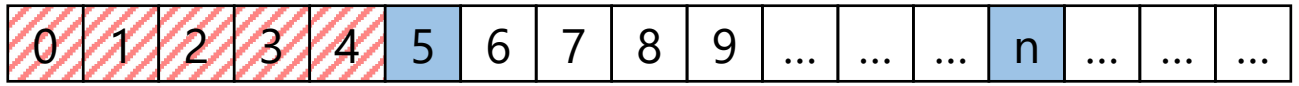
partition = {segment}




```
retention.bytes=-1
retention.ms=604800000
```

Архитектура Kafka Topic

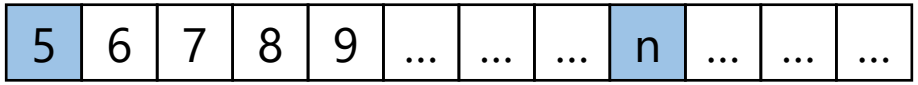
partition = {segment}




```
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

Index record = (relative offset, position)

offset = 1234567892 relative offset = **2**

size = 50 position = **200**

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
```

С версии 0.10.0.0+

Архитектура Kafka Topic

Выводы

— Топик разбит на партиции

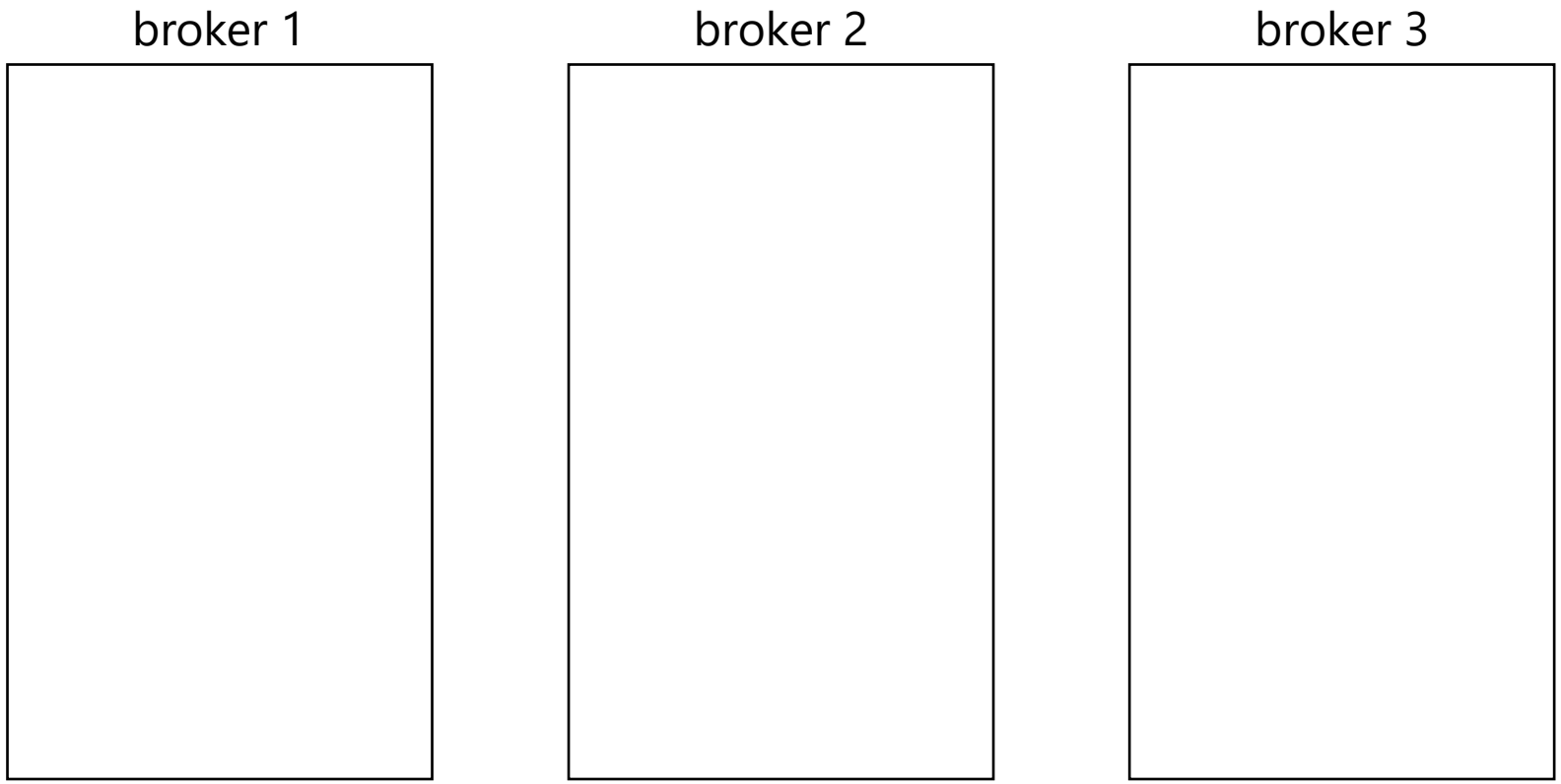
— Партиции хранятся на диске

— Данные удаляются либо по времени, либо по размеру

— Сообщение можно быстро найти по его Offset

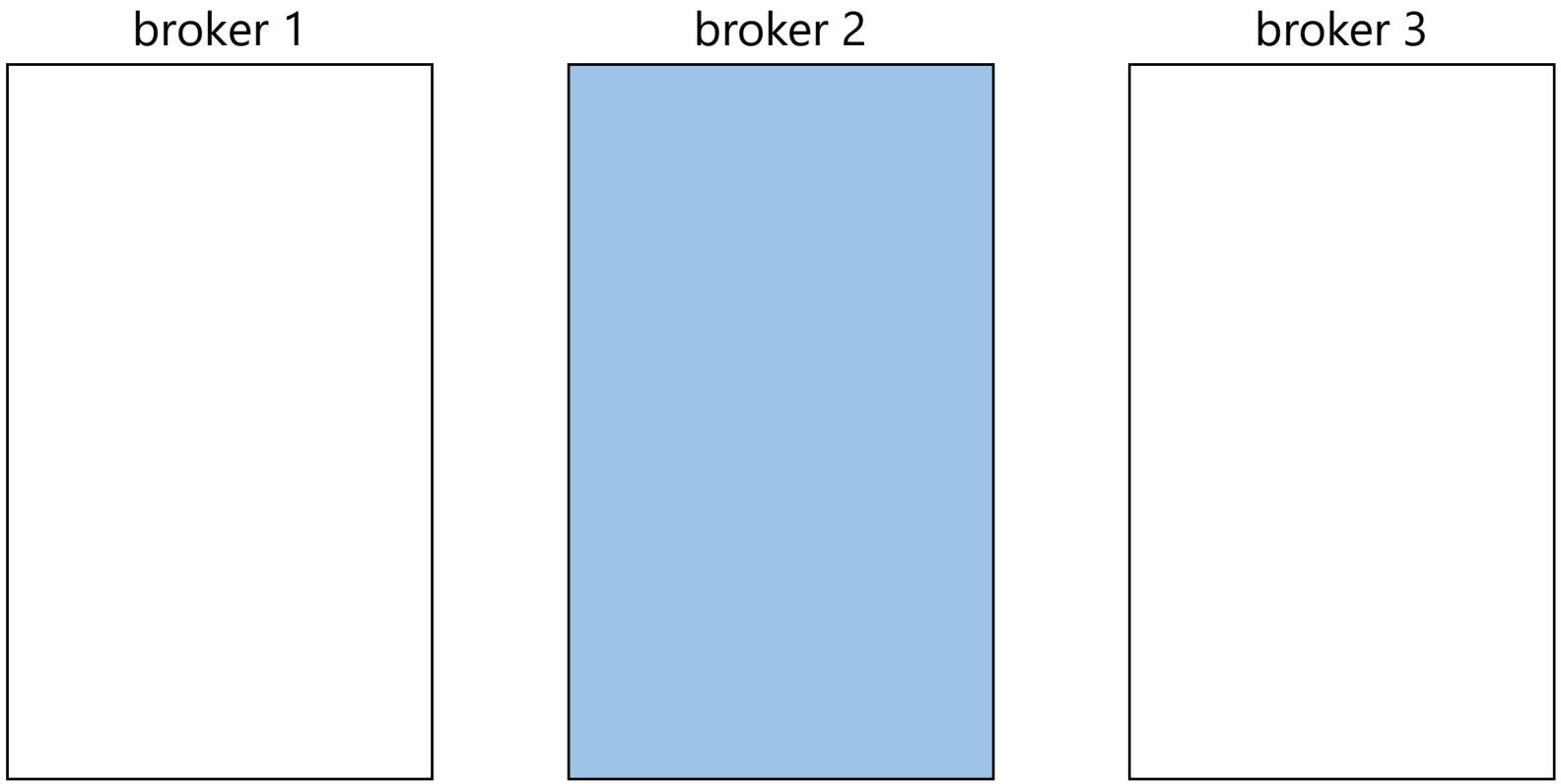
Архитектура Kafka Broker

cluster = {broker}



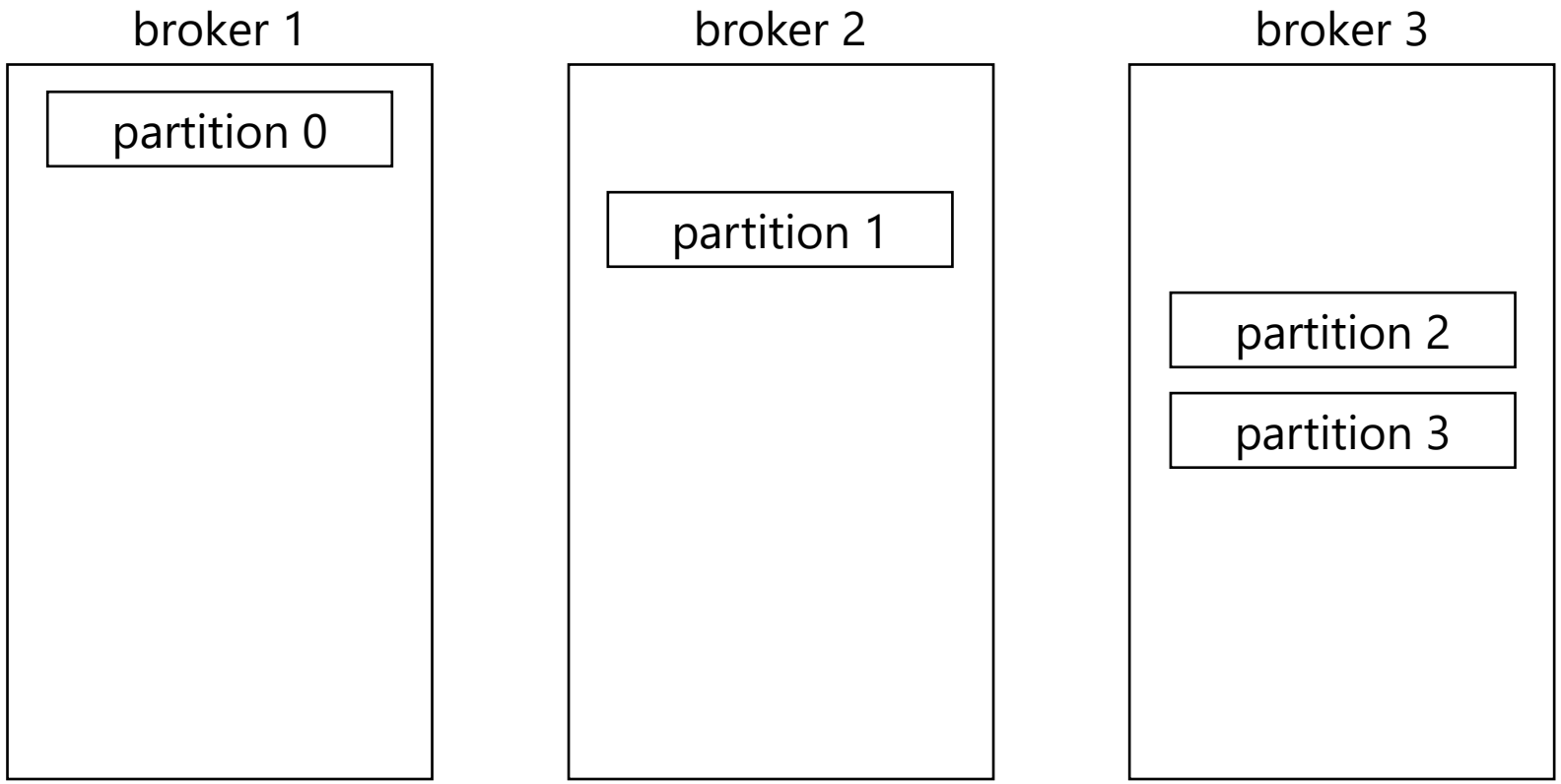
Архитектура Kafka Broker

Controller – координирует работу кластера



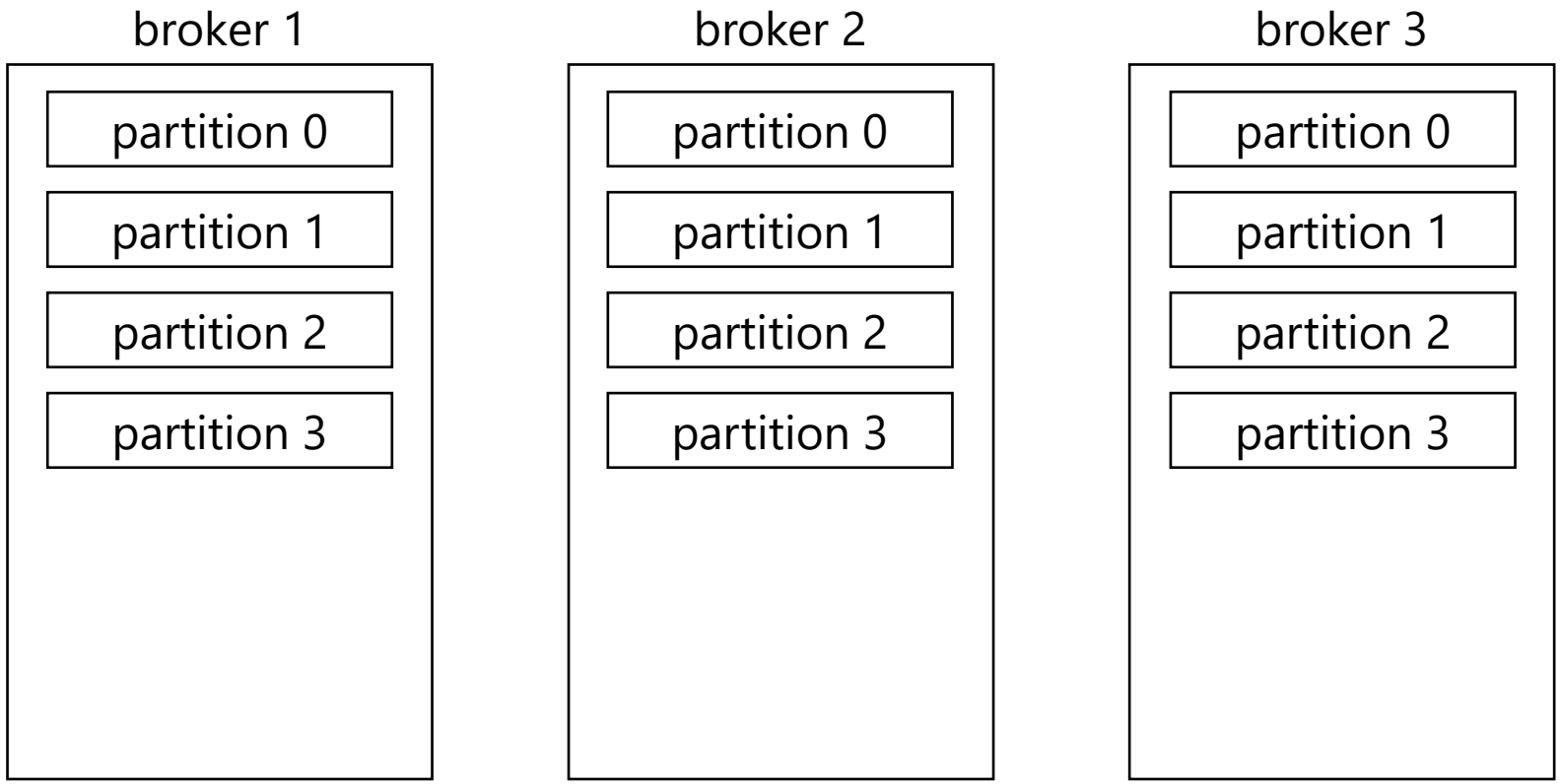
Архитектура Kafka Broker

topic = {partition}



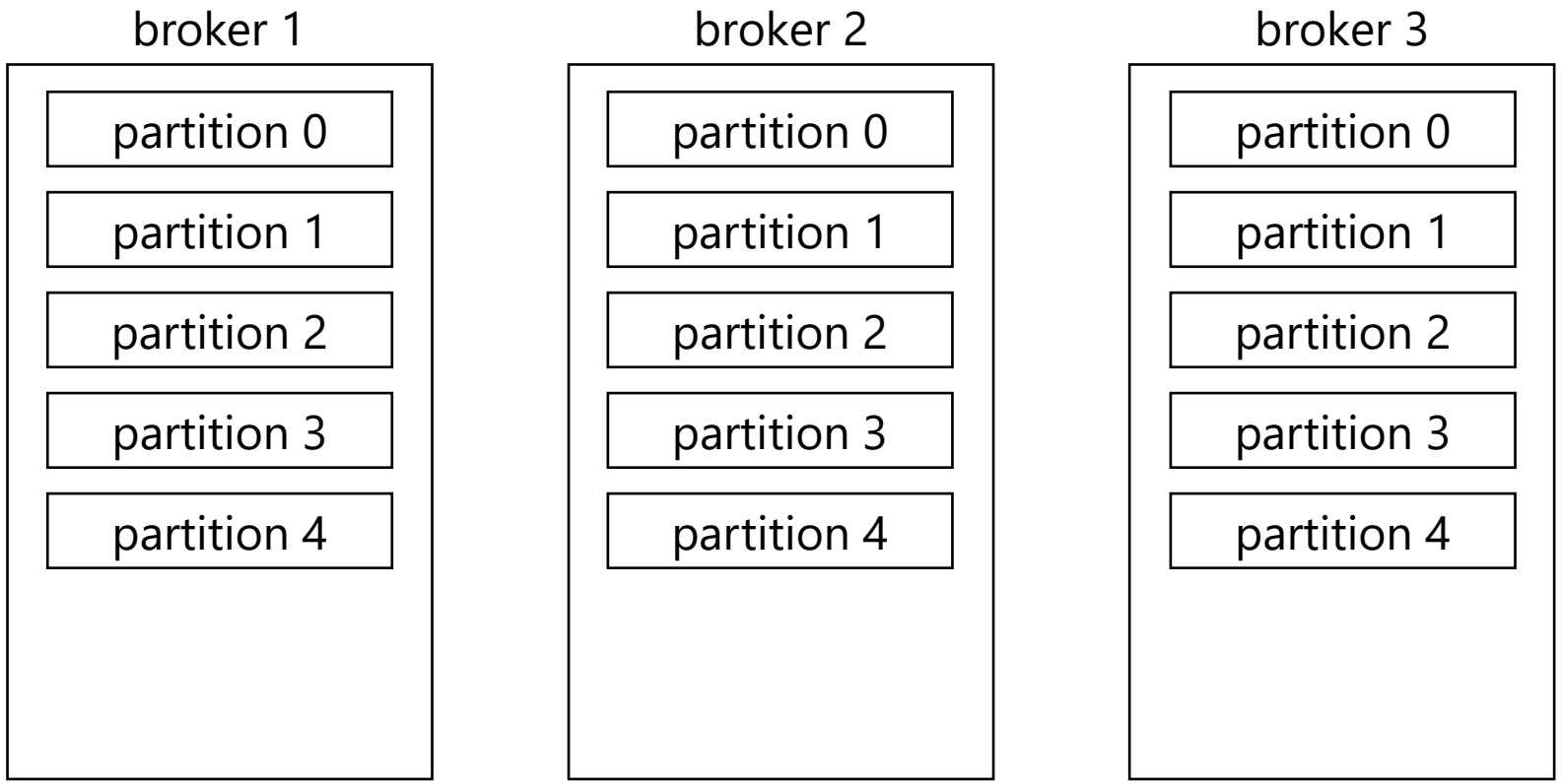
Архитектура Kafka Broker

replication factor = 3



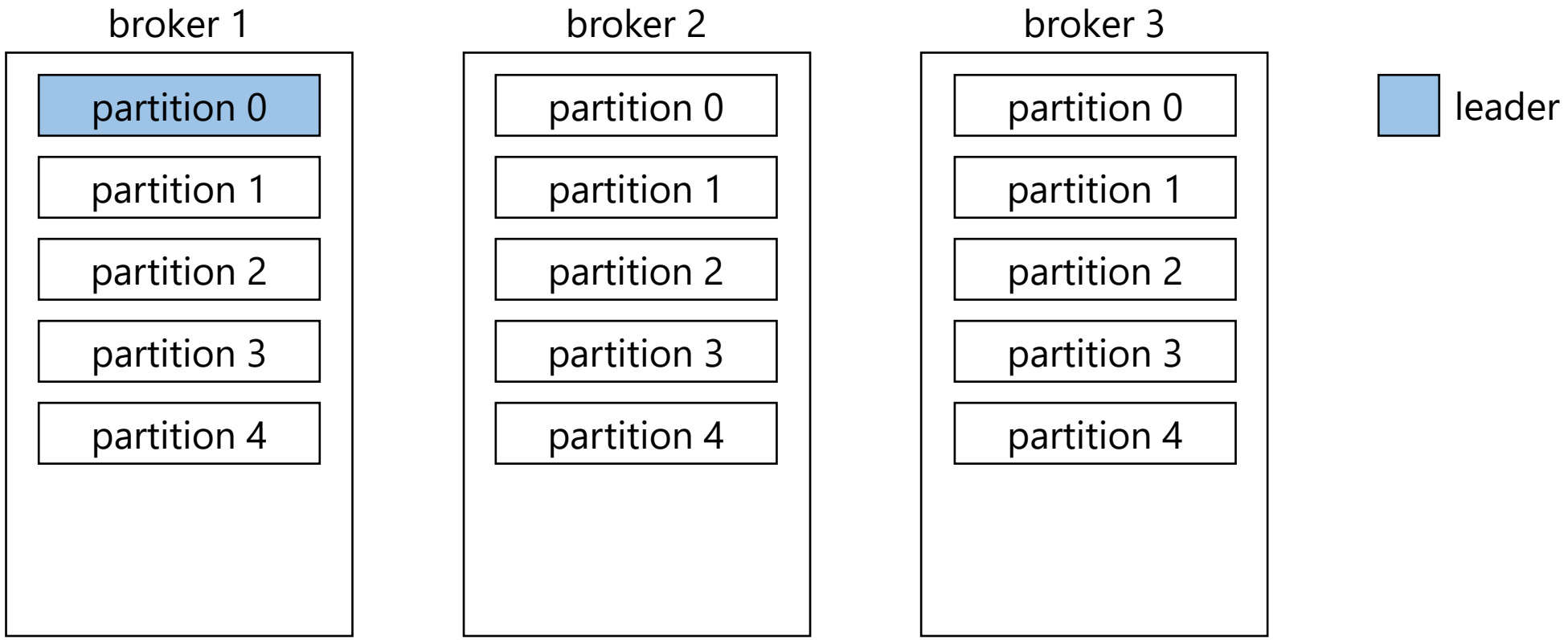
Архитектура Kafka Broker

Добавление partition



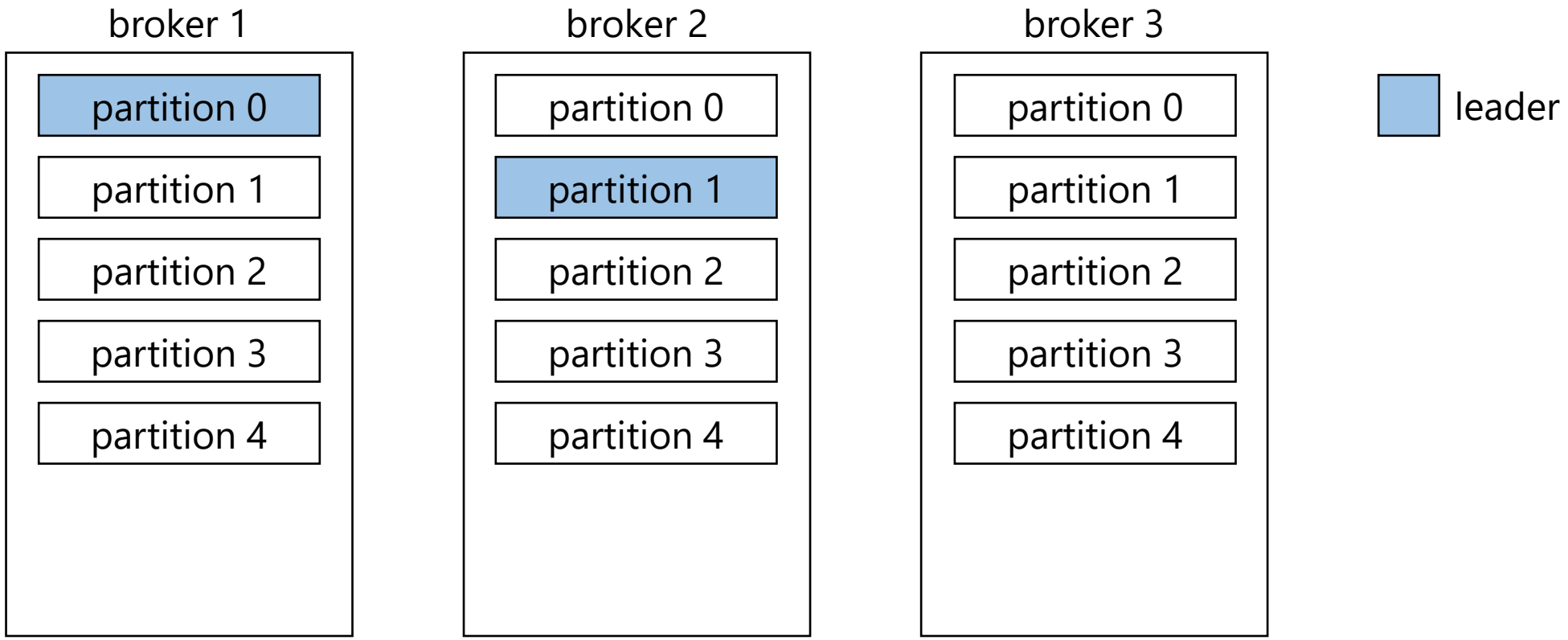
Архитектура Kafka Broker

broker 1 – leader для partition 0.



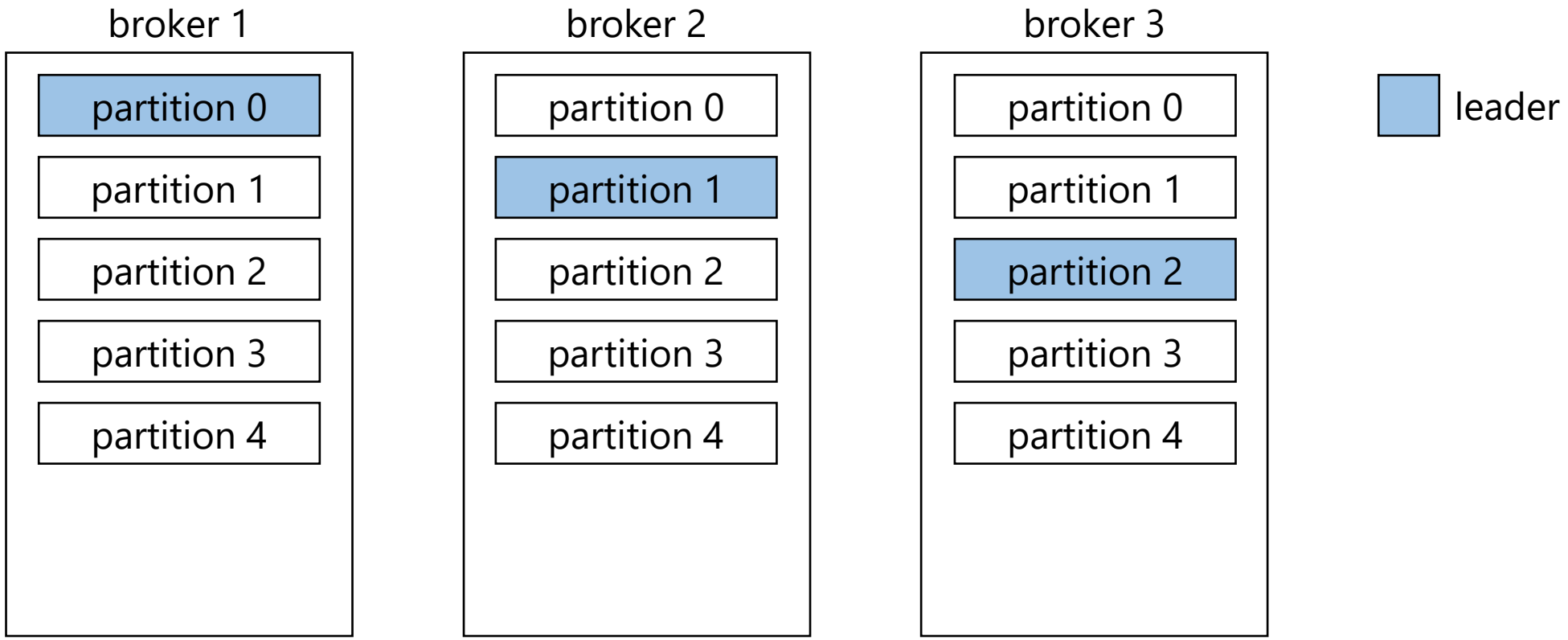
Архитектура Kafka Broker

broker 2 – leader для partition 1



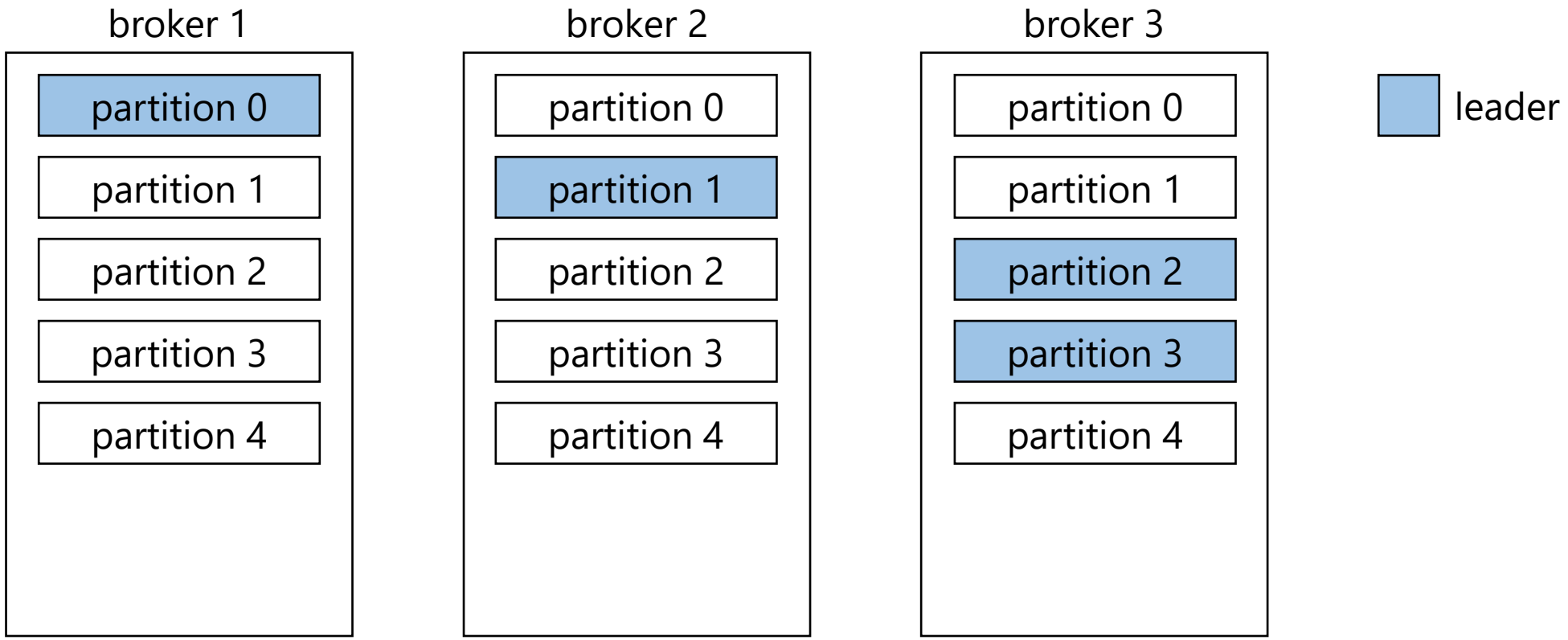
Архитектура Kafka Broker

broker 3 – leader для partition 2



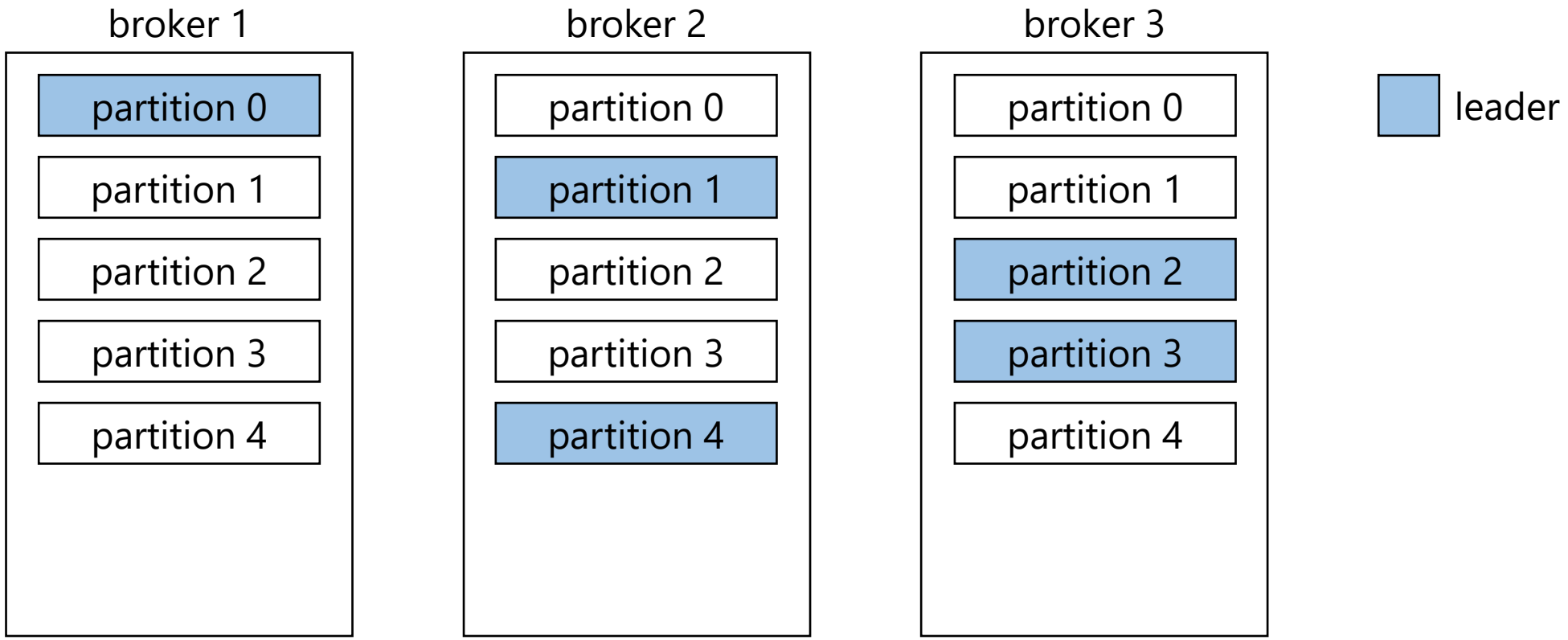
Архитектура Kafka Broker

broker 3 – leader для partition 3



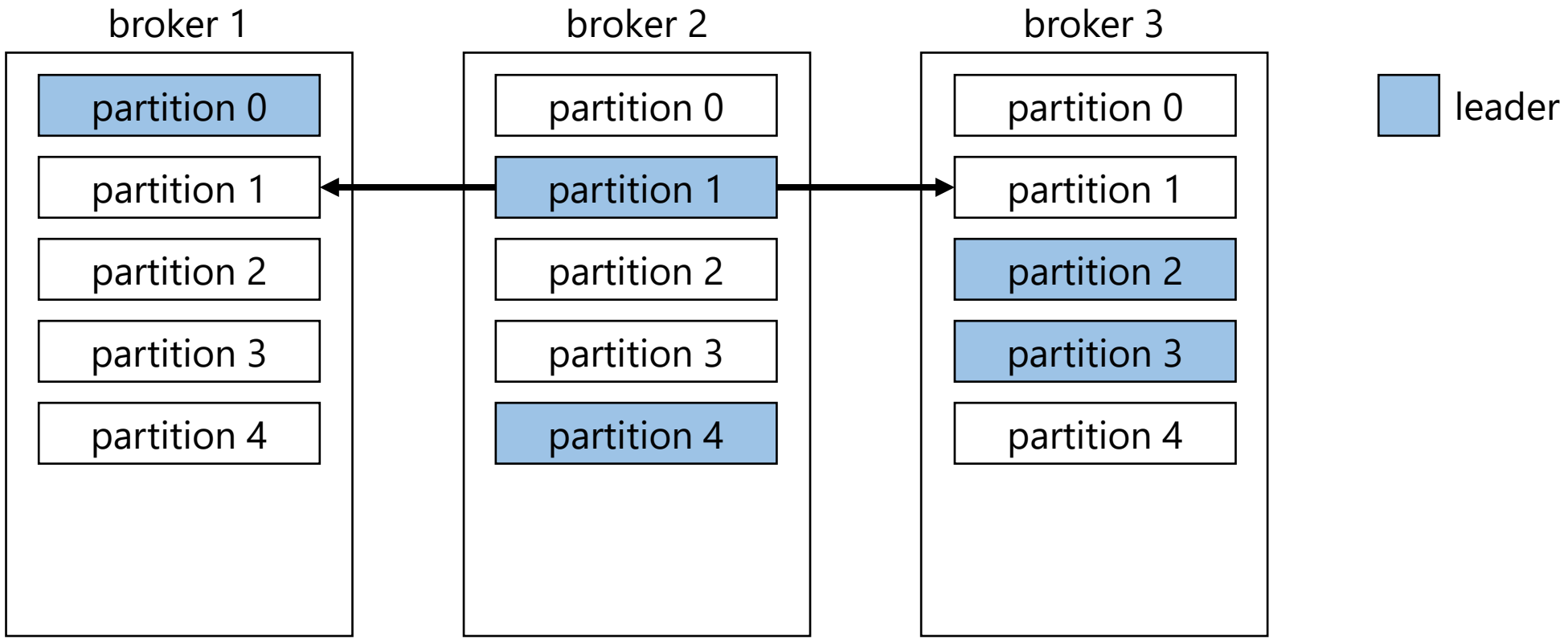
Архитектура Kafka Broker

broker 2 – leader для partition 4



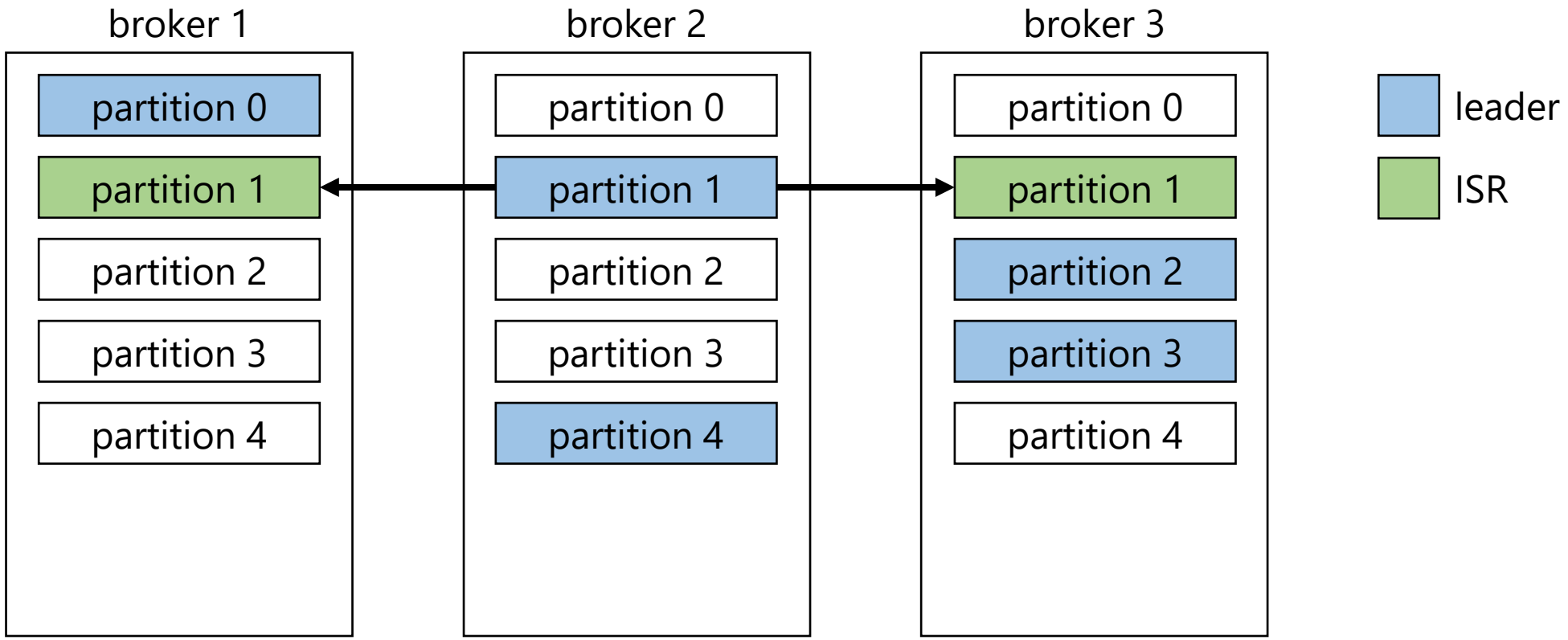
Архитектура Kafka Broker

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



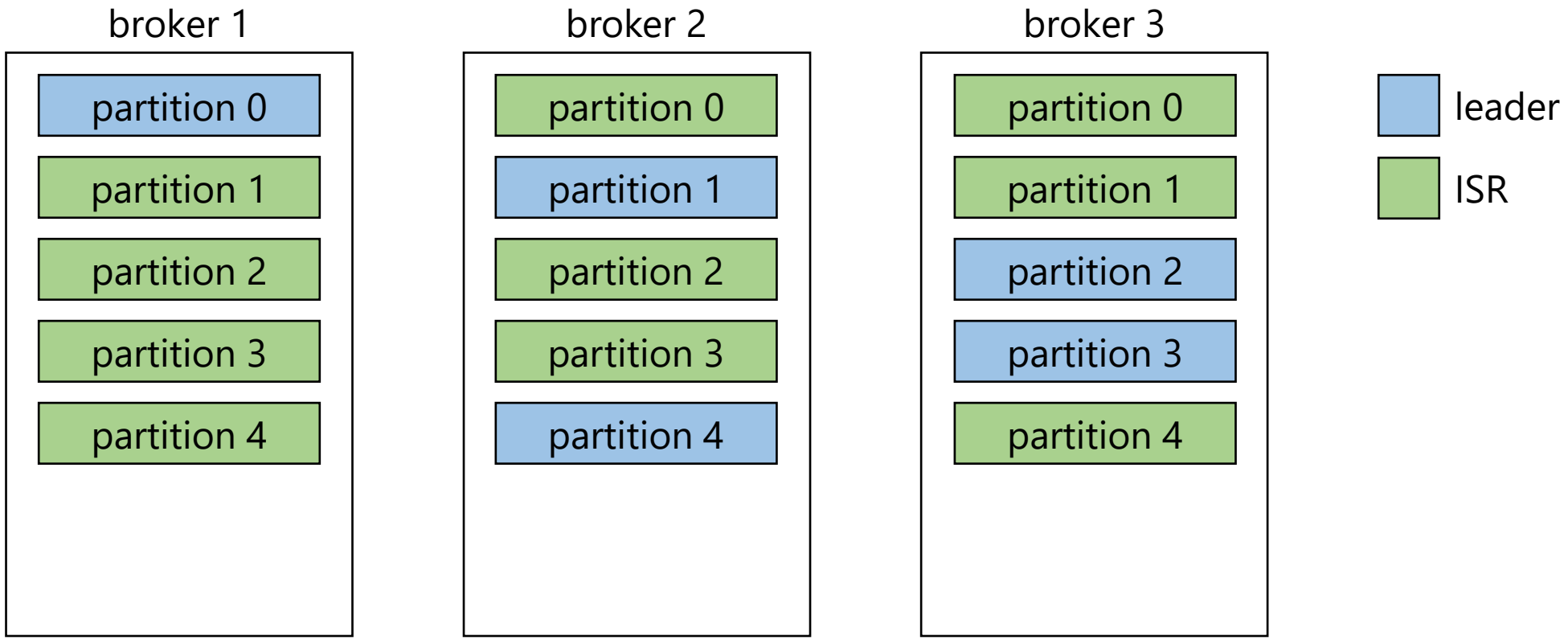
Архитектура Kafka Broker

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



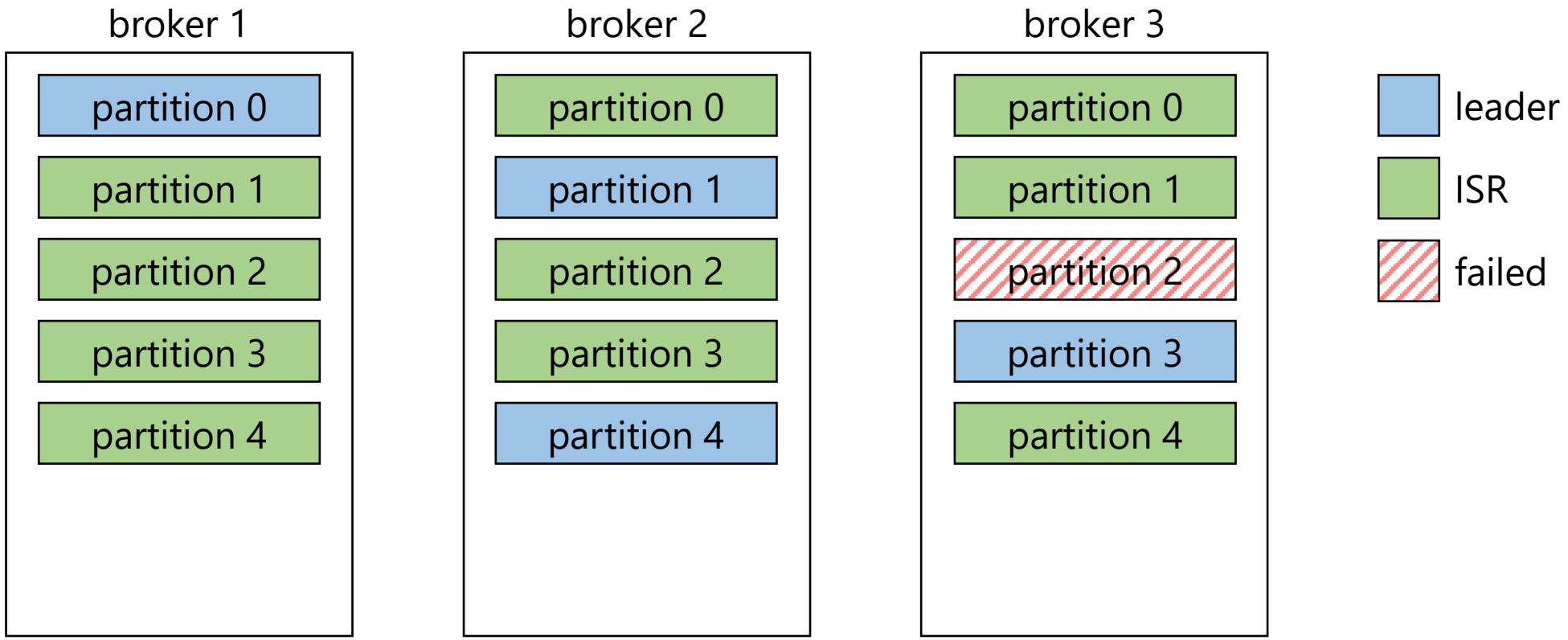
Архитектура Kafka Broker

Все реплики синхронизированы



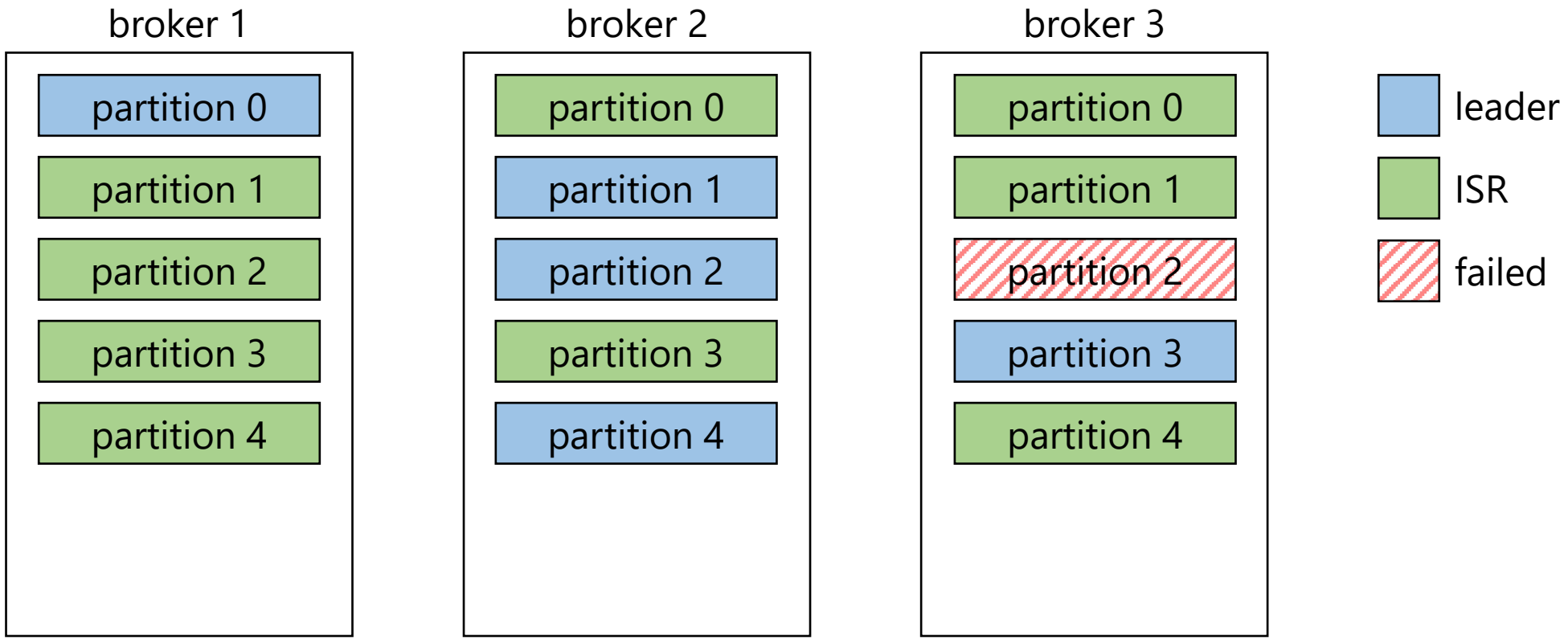
Архитектура Kafka Broker

Недоступность *лидера* у partition 2



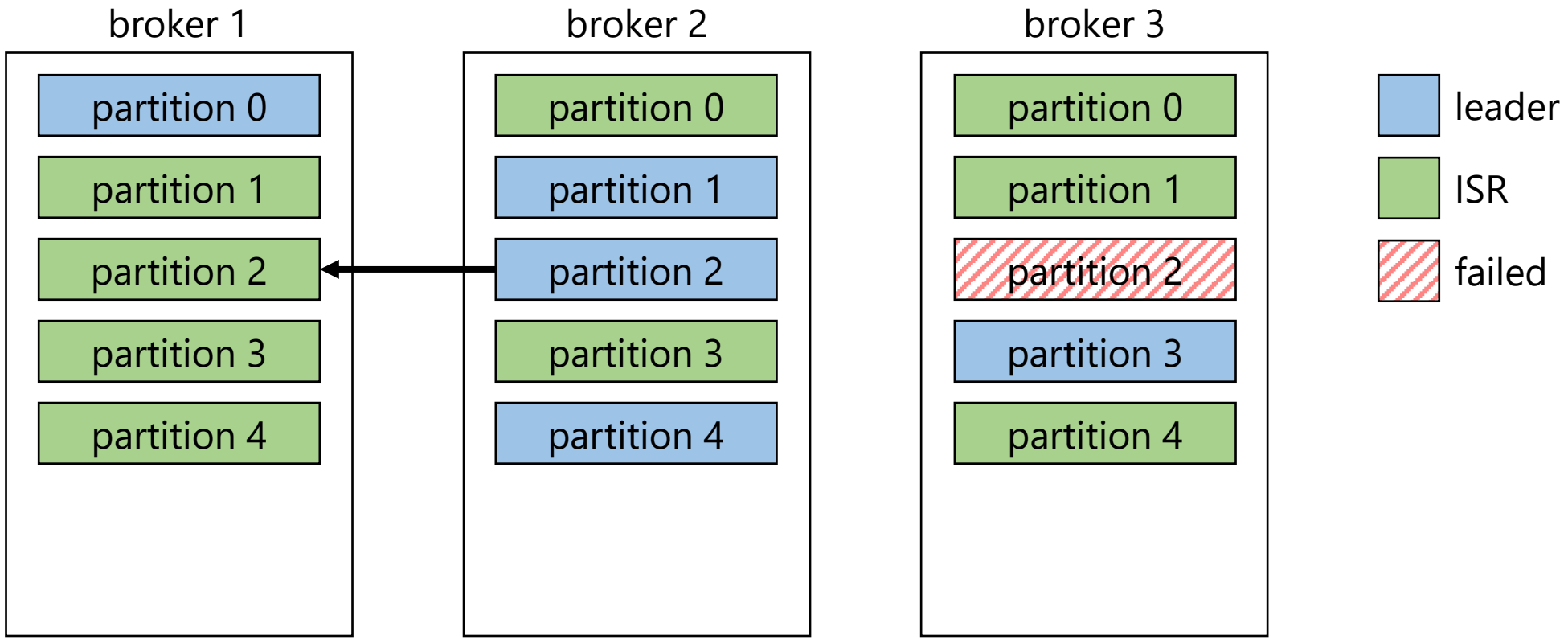
Архитектура Kafka Broker

Выбор нового *лидера* в случае недоступности

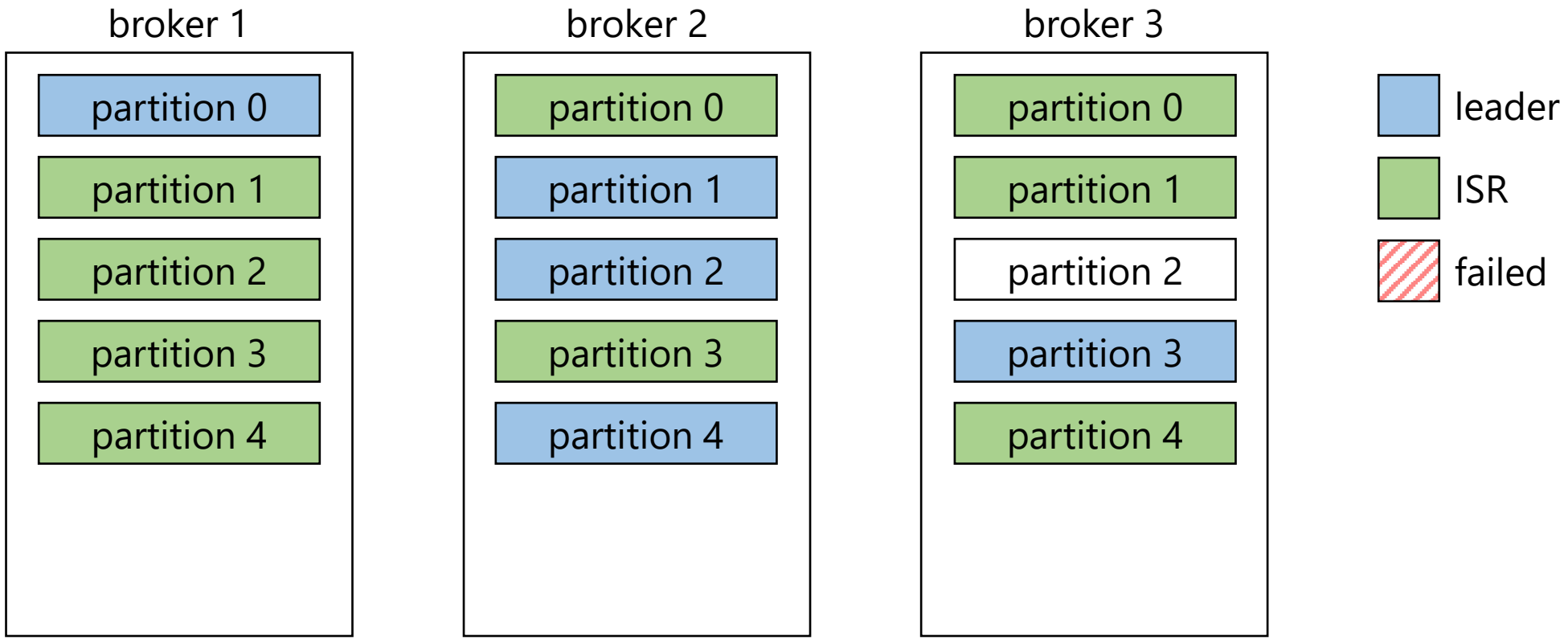


Архитектура Kafka Broker

Репликация с нового *лидера*

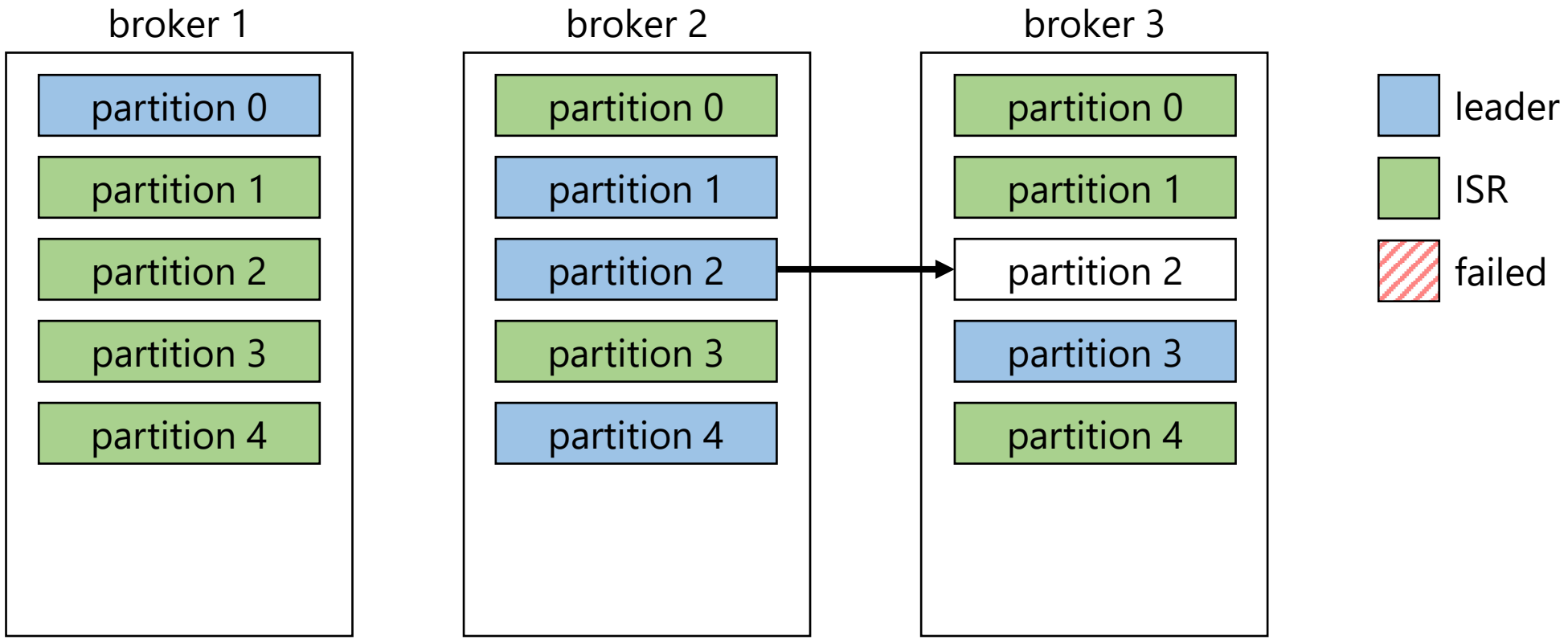


Архитектура Kafka Broker

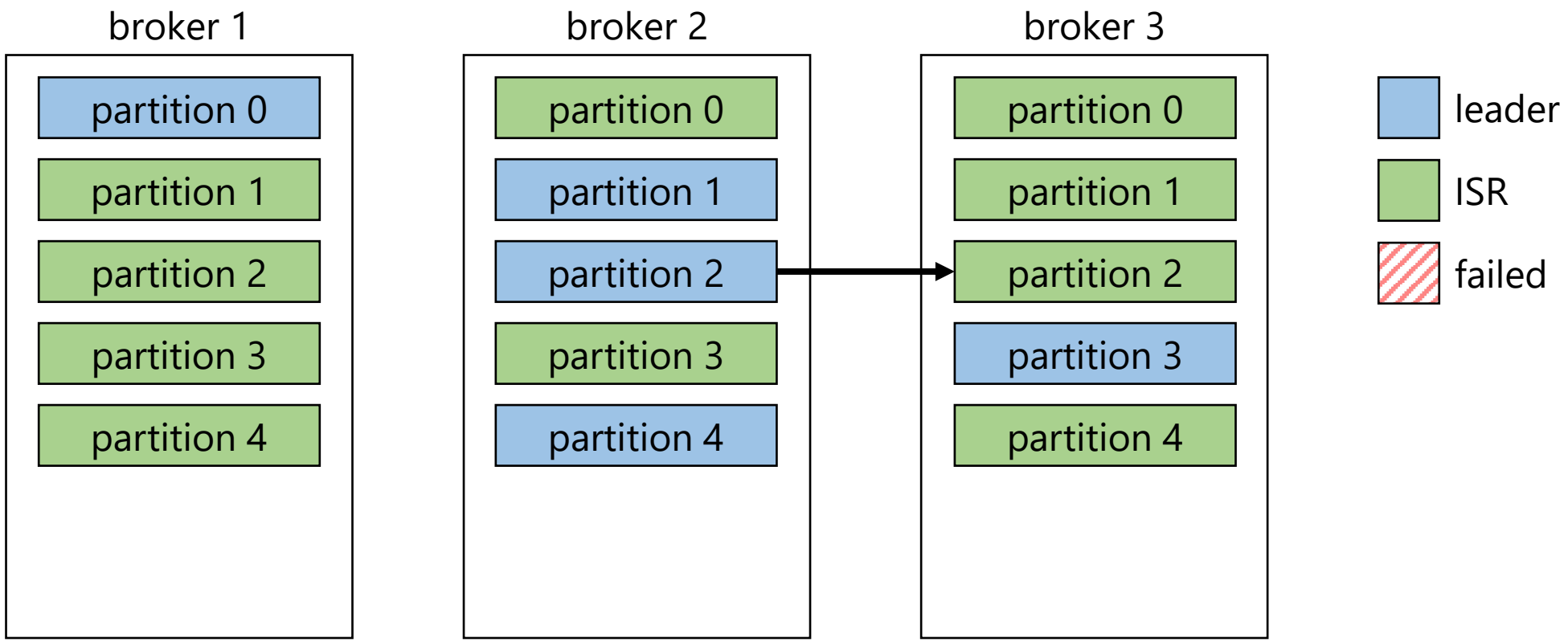


Архитектура Kafka Broker

Синхронизация реплики с *лидером* после восстановления

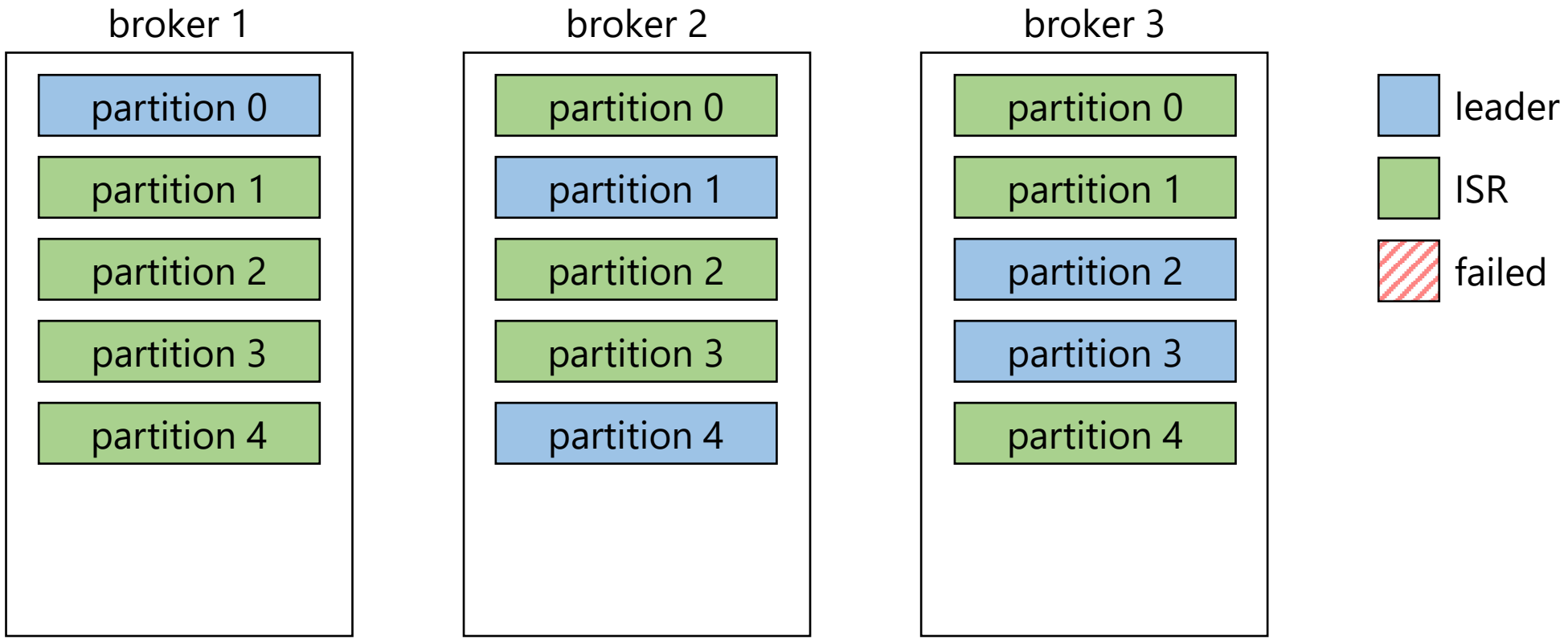


Архитектура Kafka Broker



Архитектура Kafka Broker

Перебалансировка *лидеров*



Архитектура Kafka Broker

Выводы

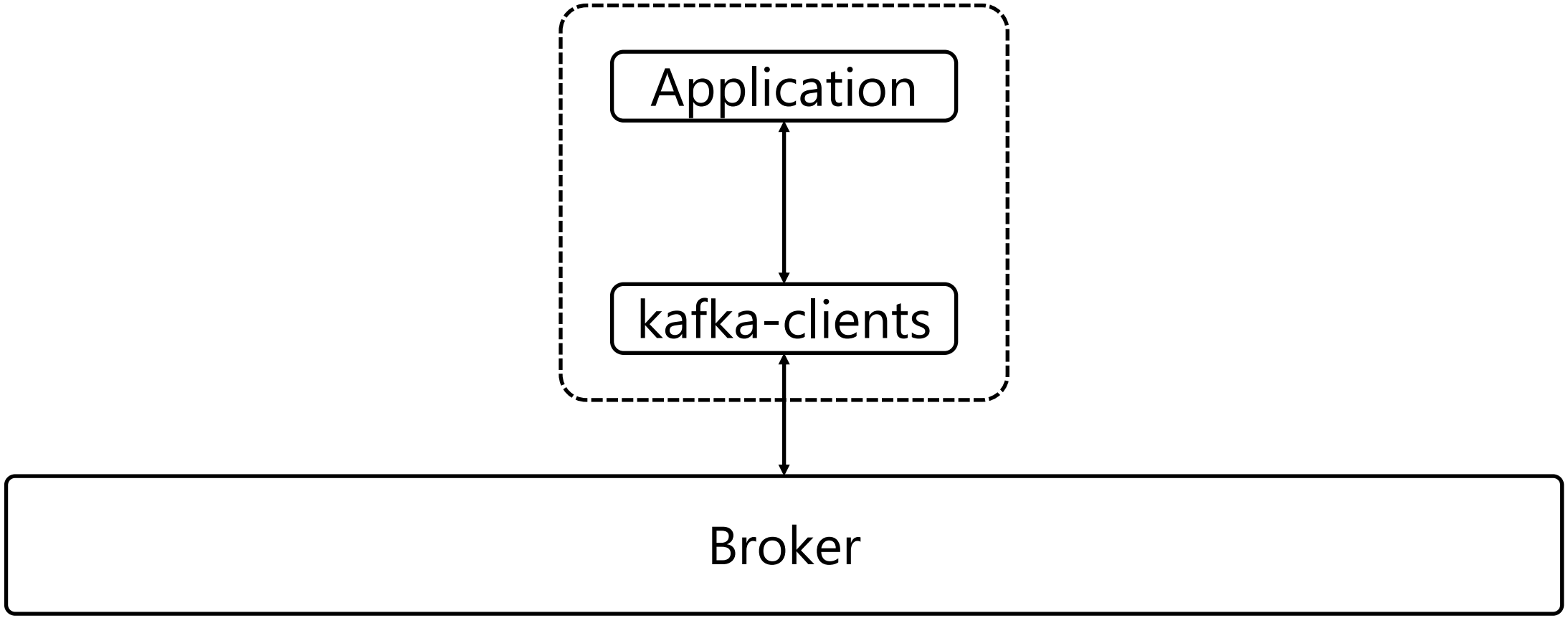
— У каждой партиции свой Лидер

— Сообщения пишутся в Лидера

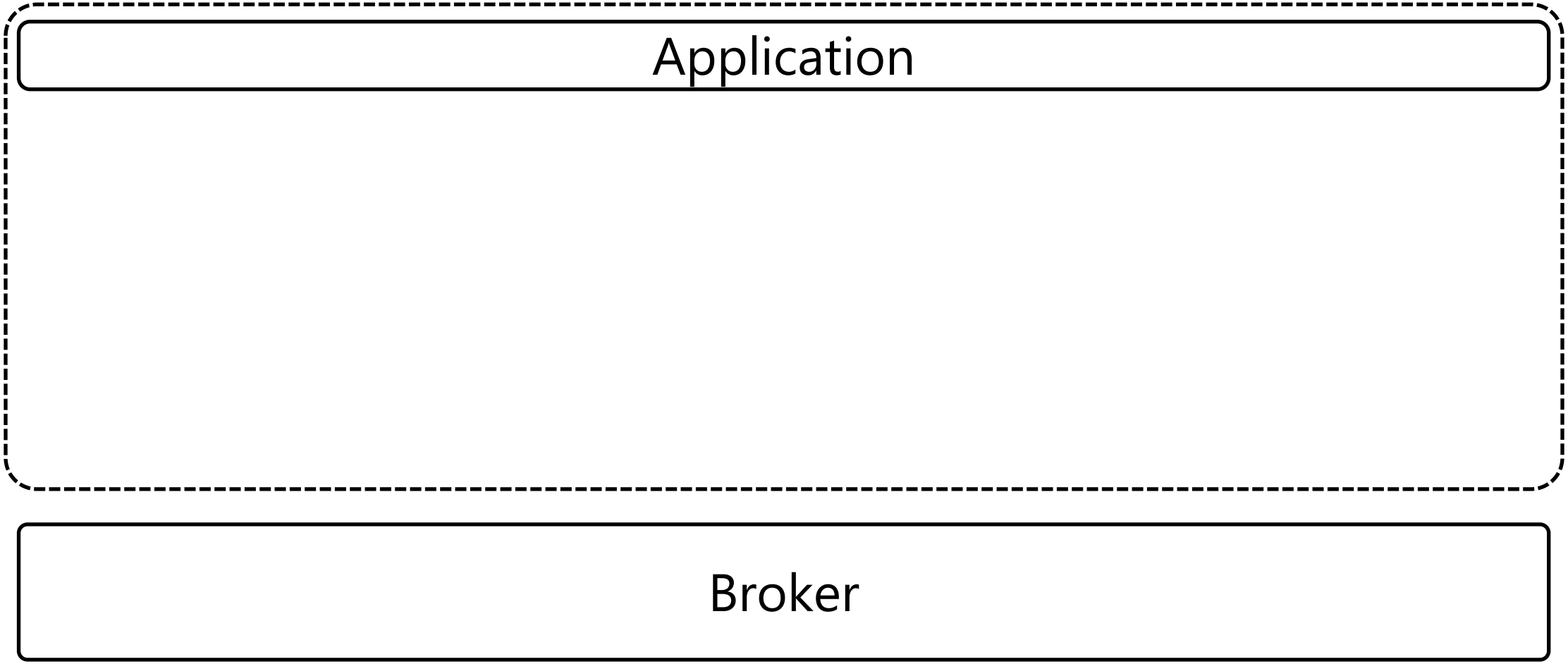
— Данные реплицируются между брокерами

— Автоматический фейловер лидерства

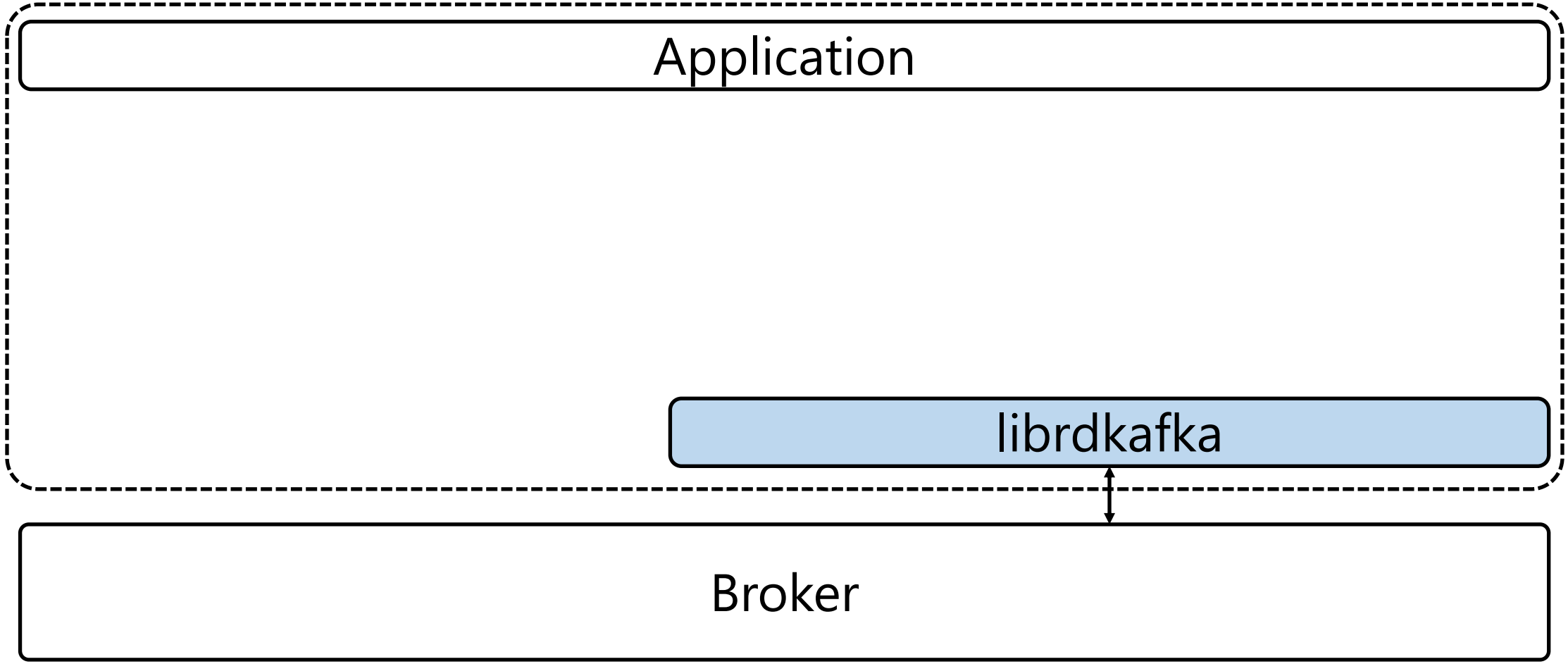
Kafka-клиент, когда используется Java/JVM



Зоопарк Python-клиентов

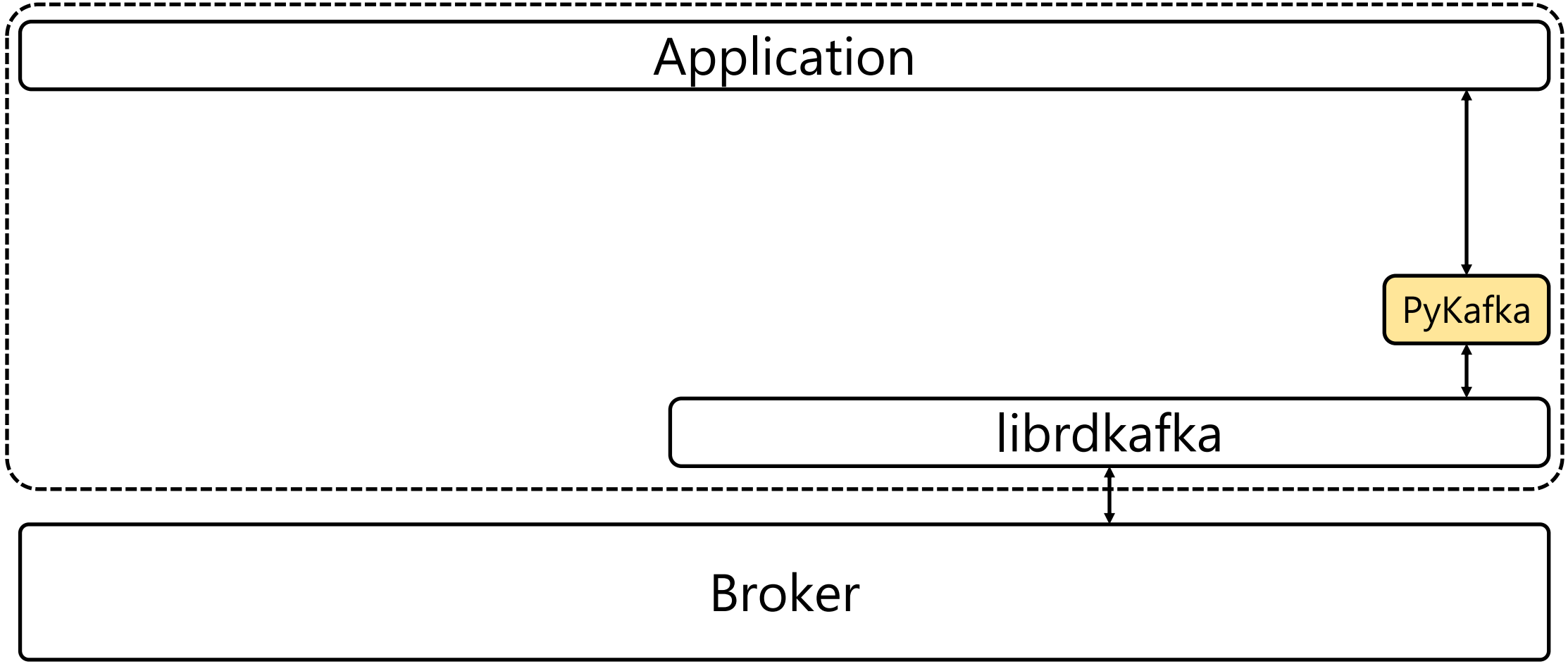


Зоопарк Python-клиентов



https://github.com/confluentinc/librdkafka

Зоопарк Python-клиентов

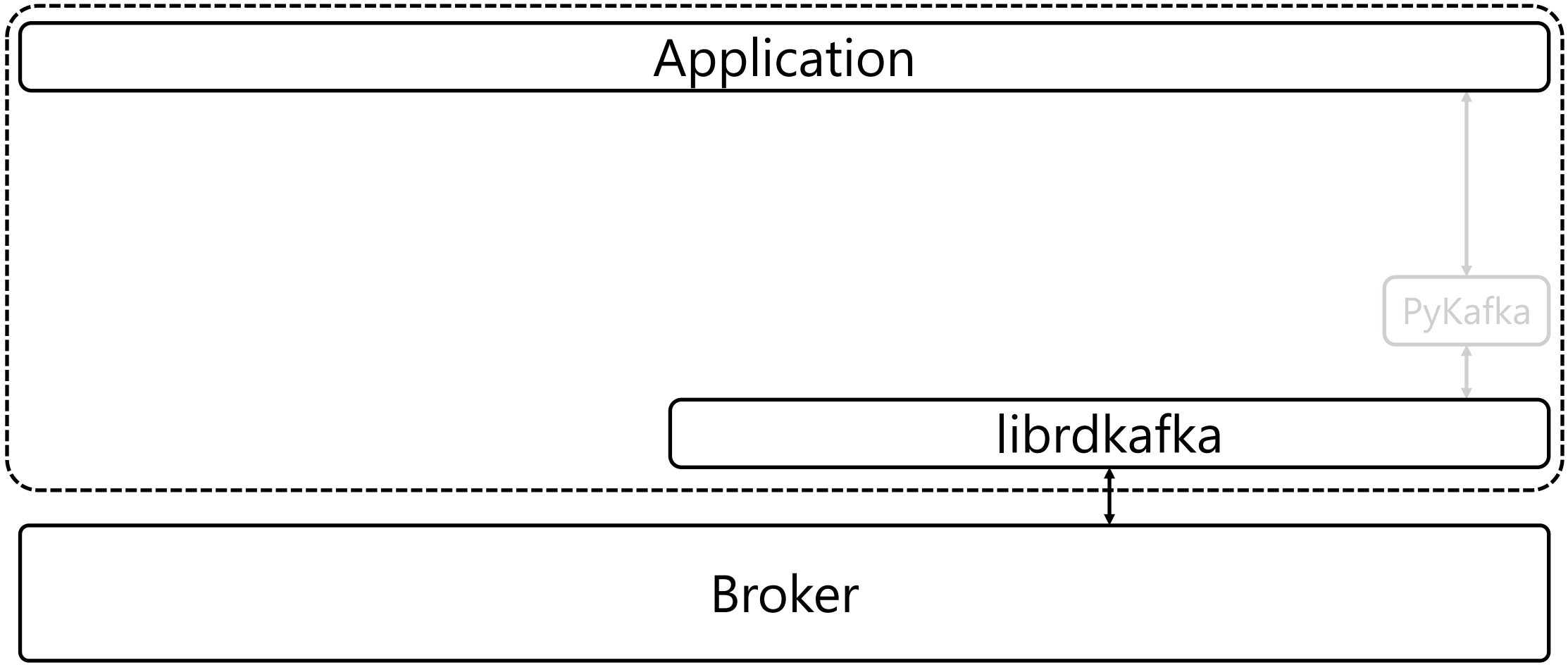


https://github.com/Parsely/pykafka

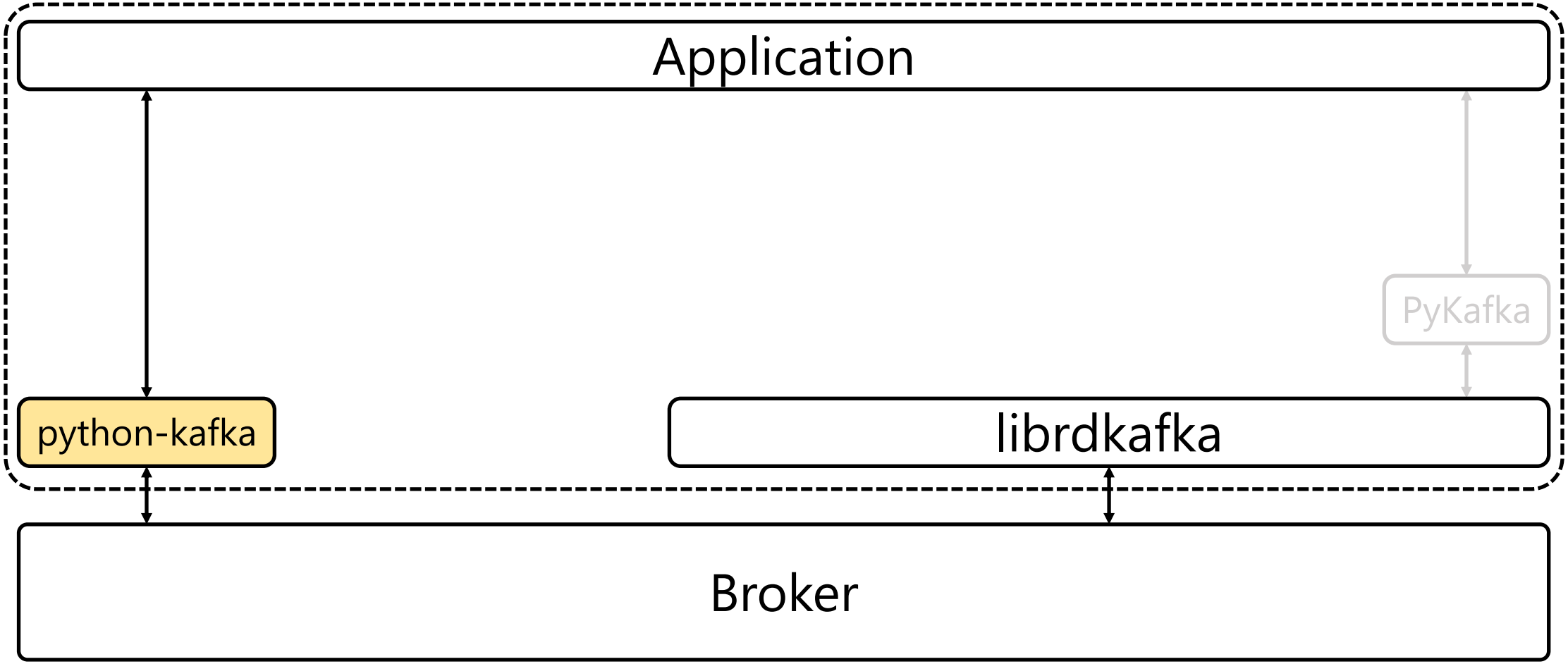
Зоопарк Python-клиентов

https://github.com/Parsely/pykafka

Зоопарк Python-клиентов

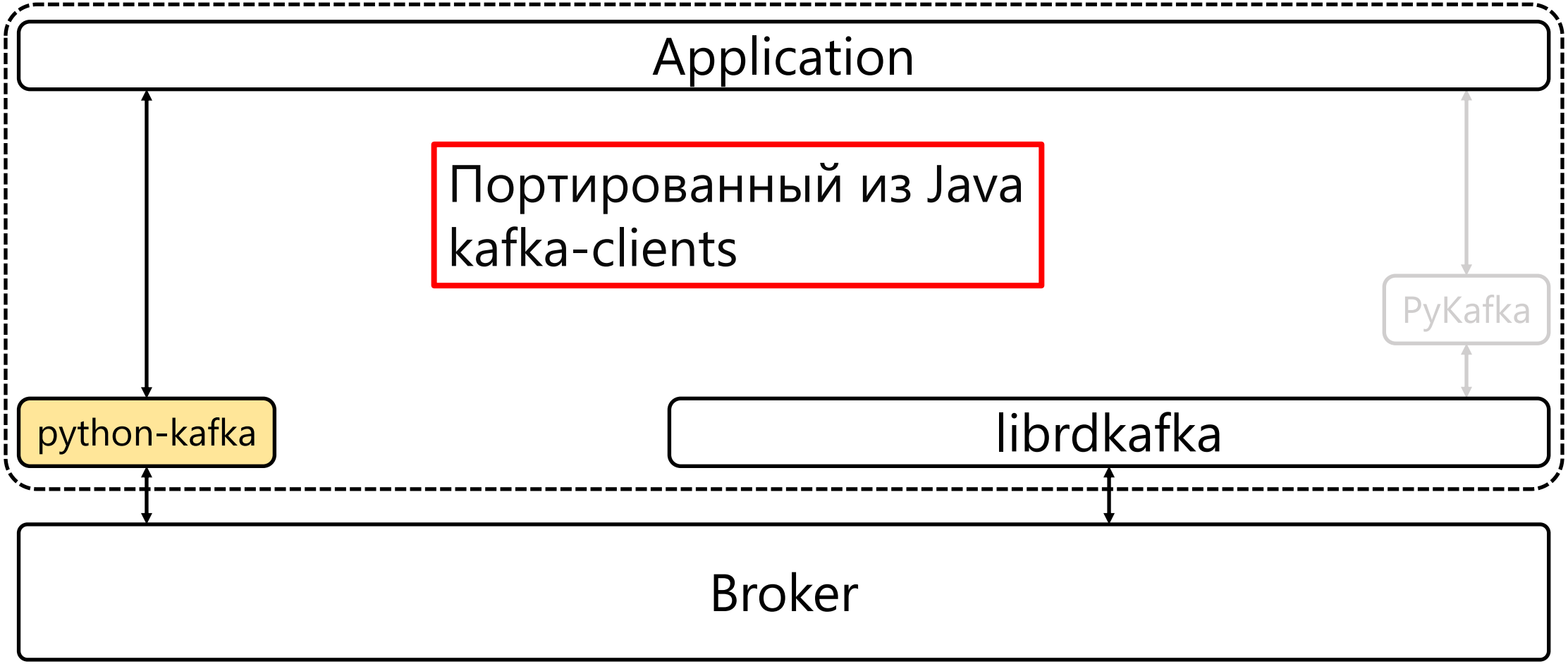


Зоопарк Python-клиентов



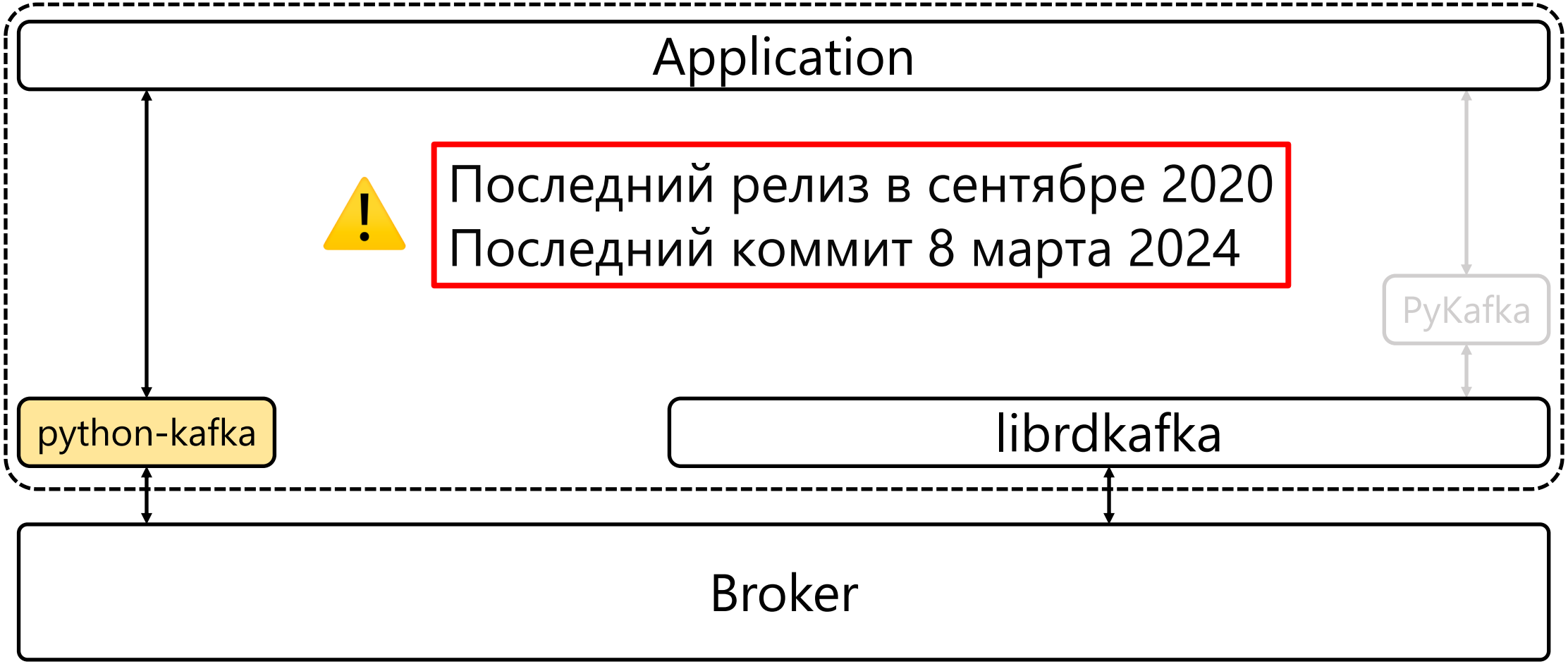
https://github.com/dpkp/kafka-python

Зоопарк Python-клиентов



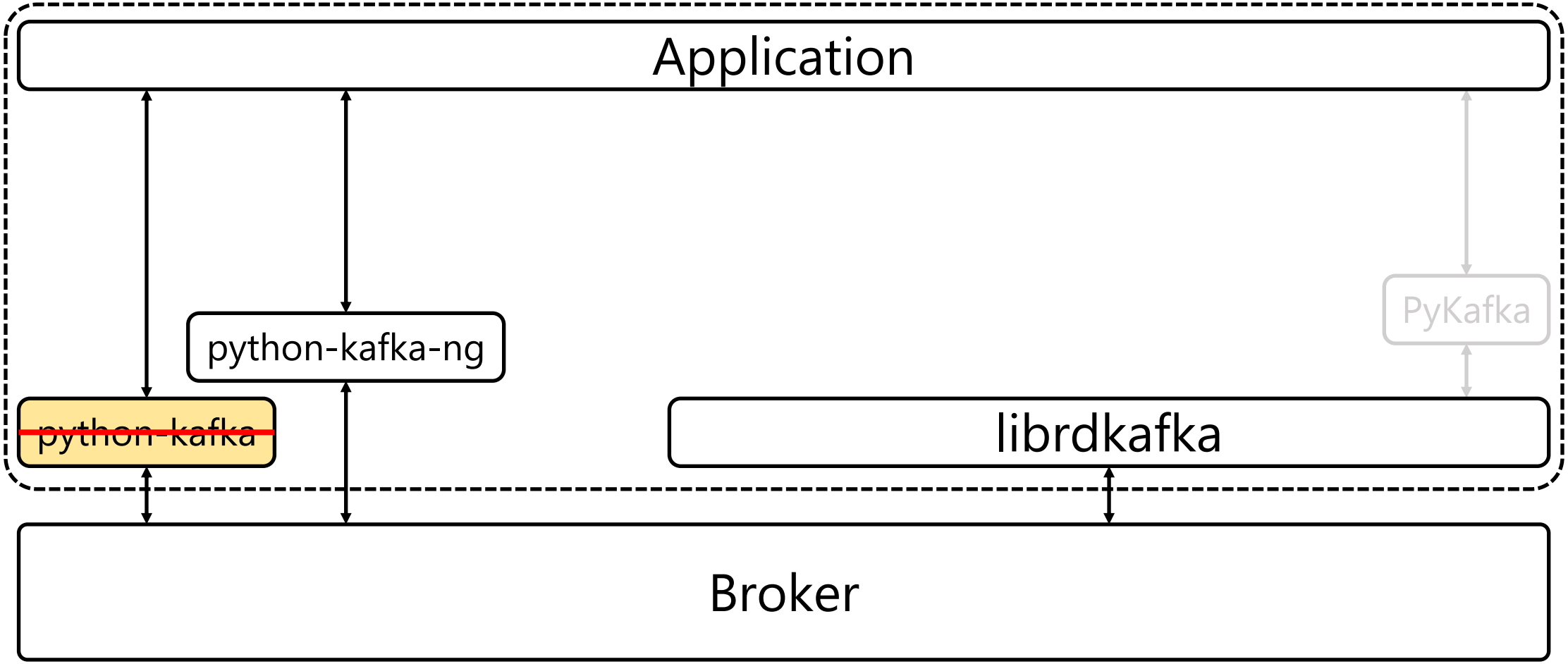
https://github.com/dpkp/kafka-python

Зоопарк Python-клиентов



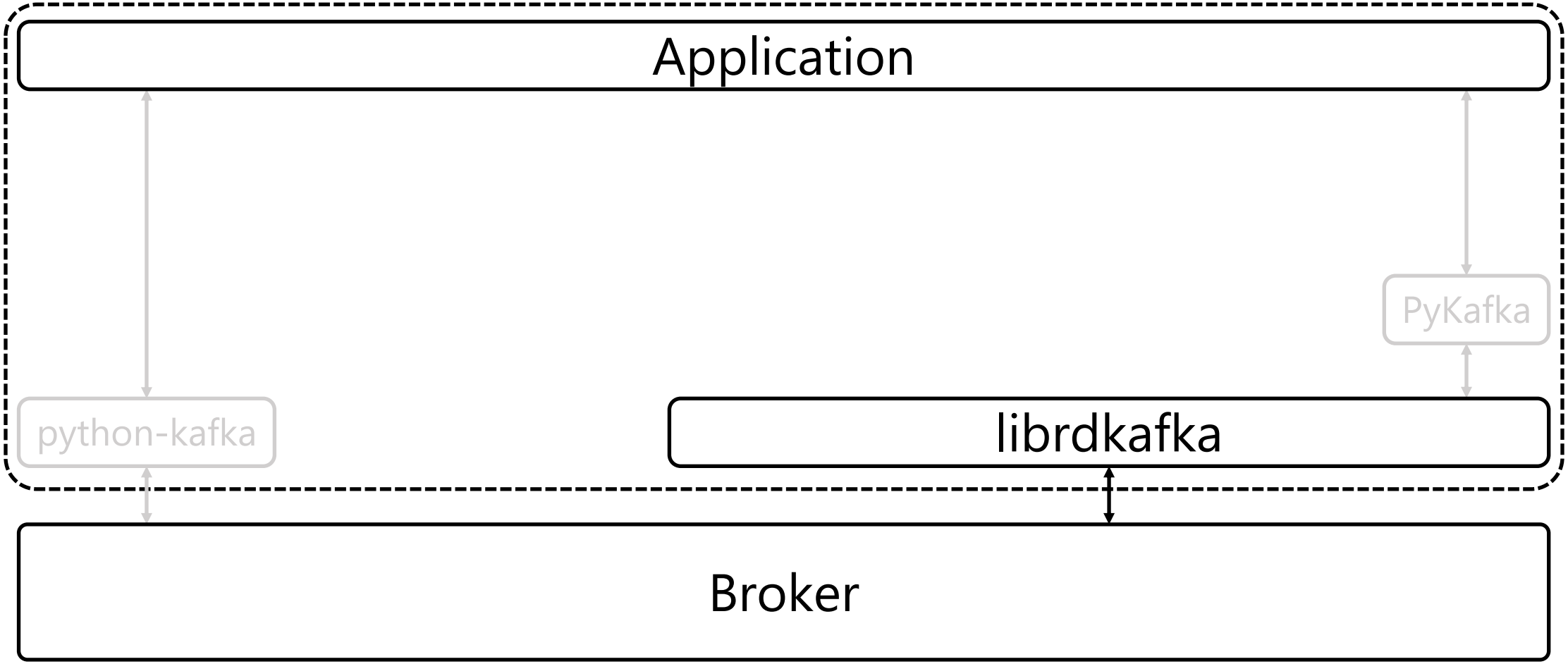
https://github.com/dpkp/kafka-python

Зоопарк Python-клиентов

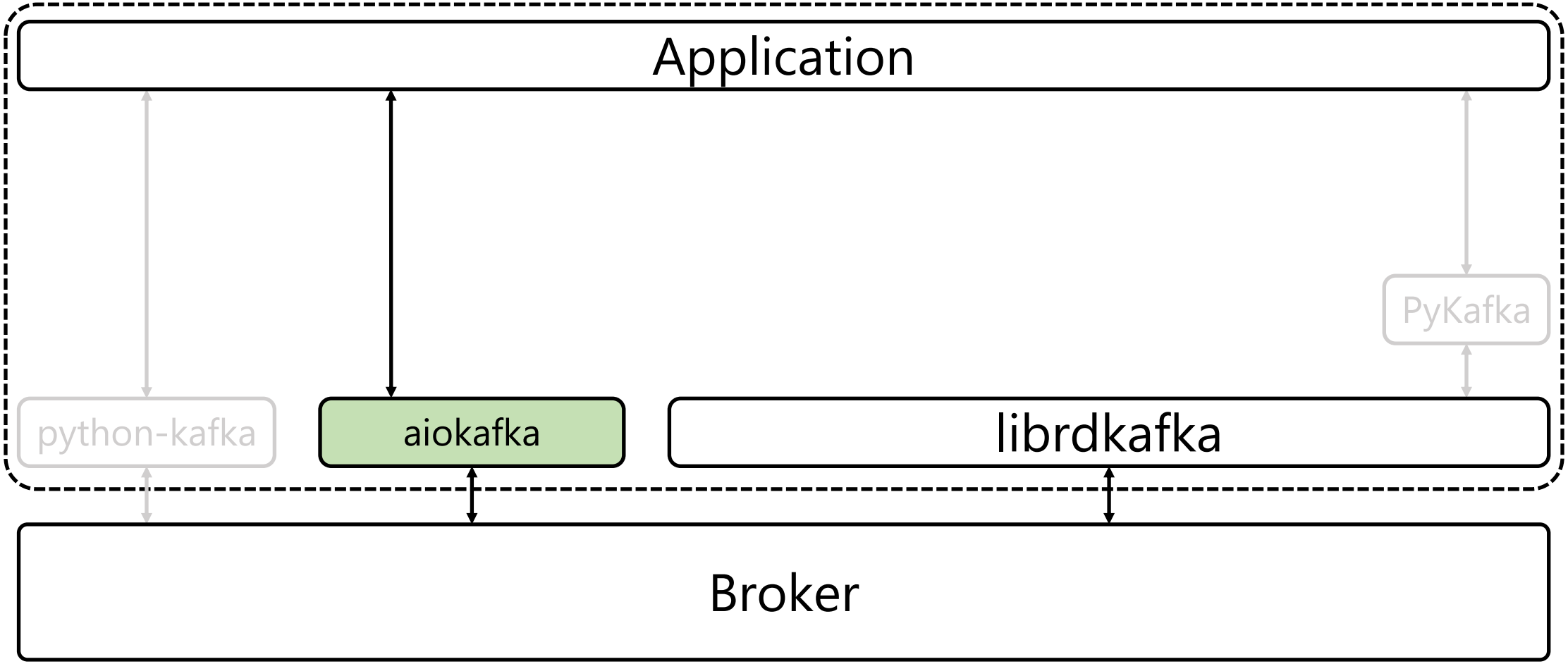


https://github.com/wbarnha/kafka-python-ng

Зоопарк Python-клиентов

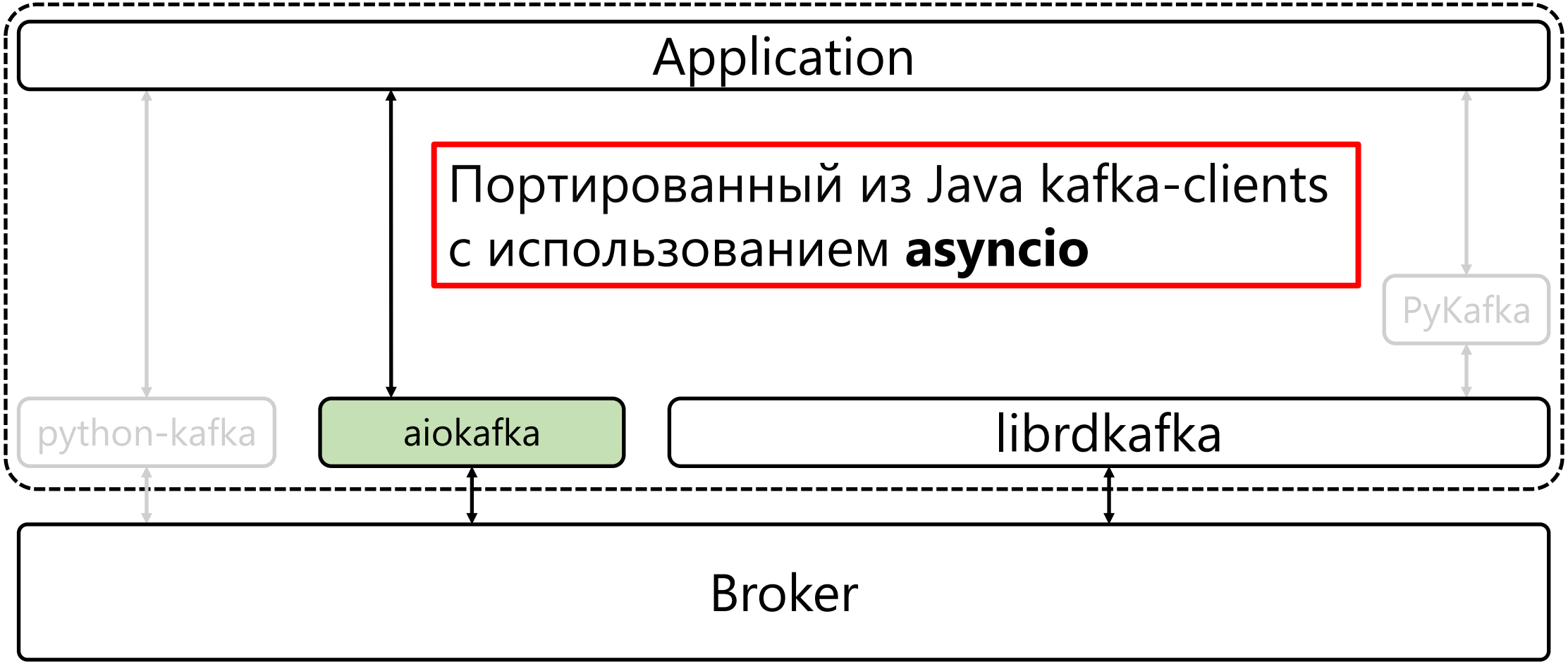


Зоопарк Python-клиентов



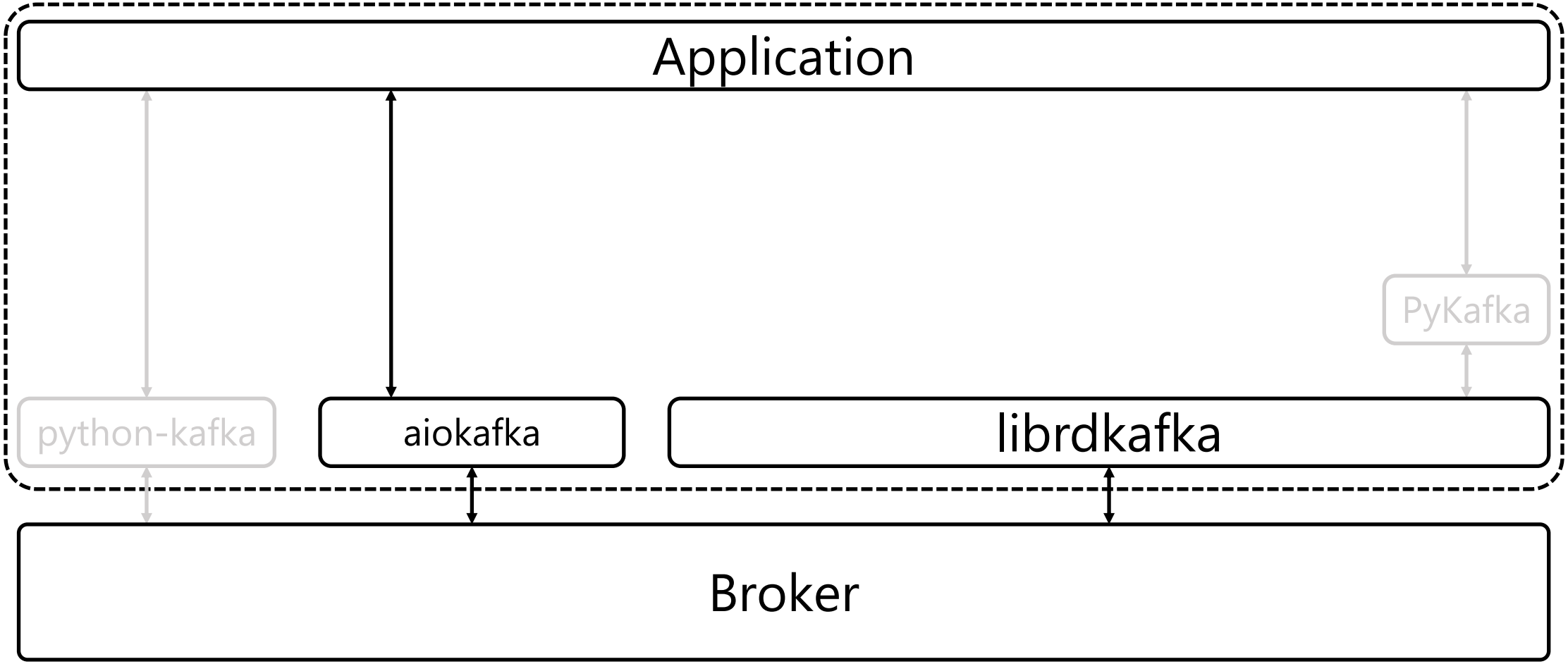
https://github.com/aio-libs/aiokafka

Зоопарк Python-клиентов

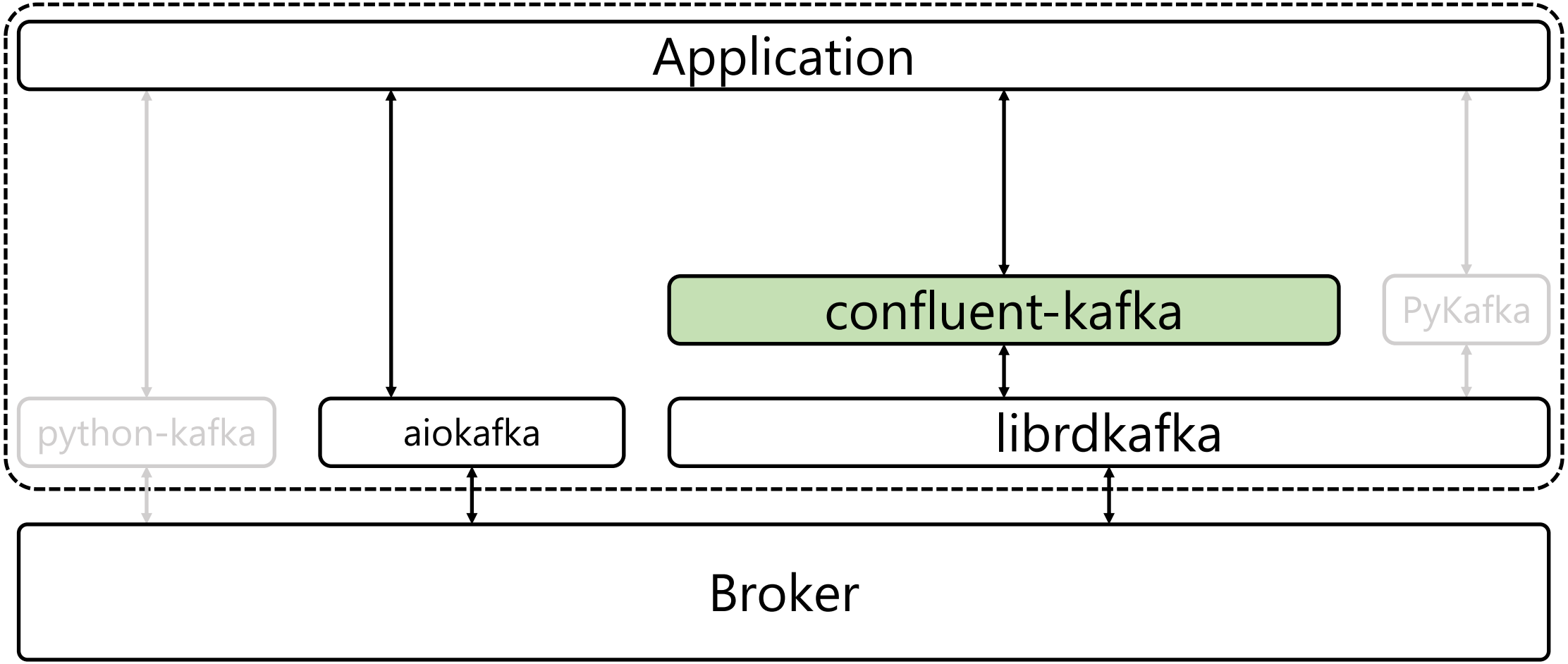


https://github.com/aio-libs/aiokafka

Зоопарк Python-клиентов

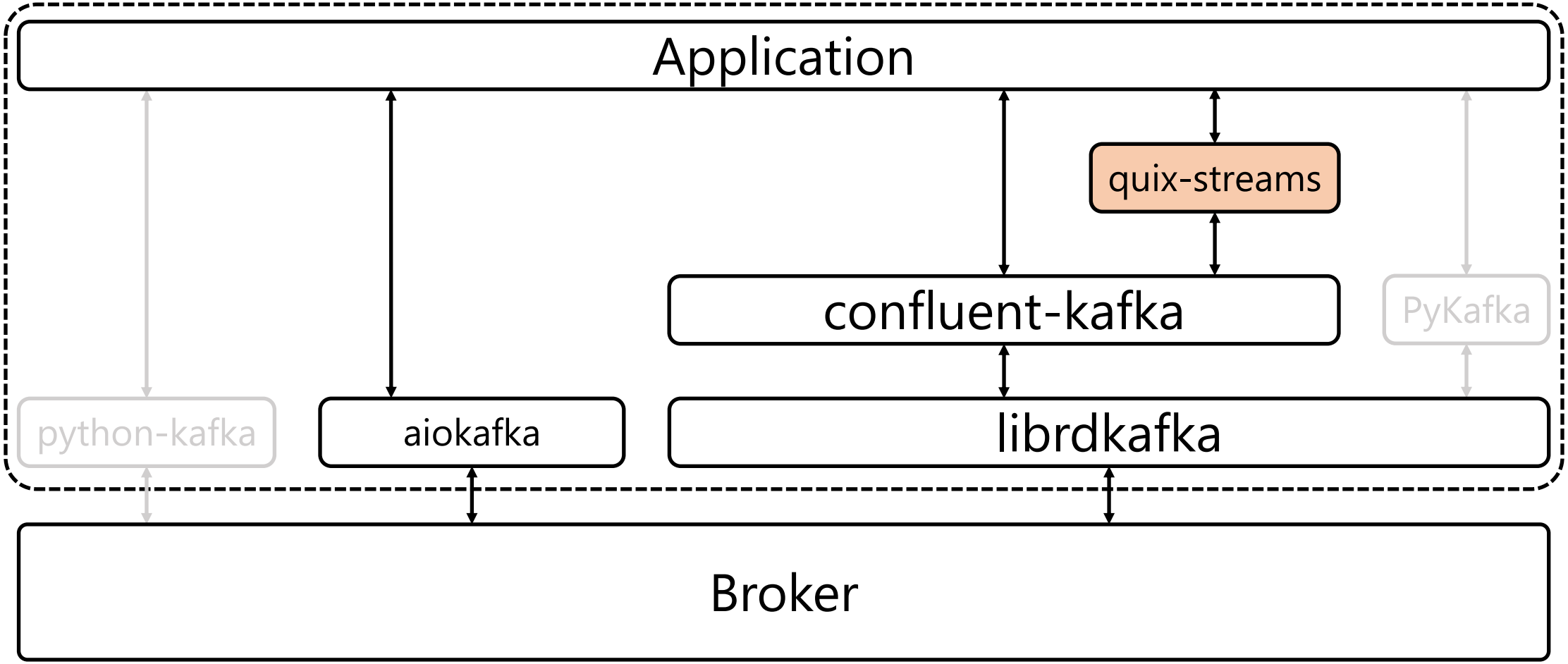


Зоопарк Python-клиентов



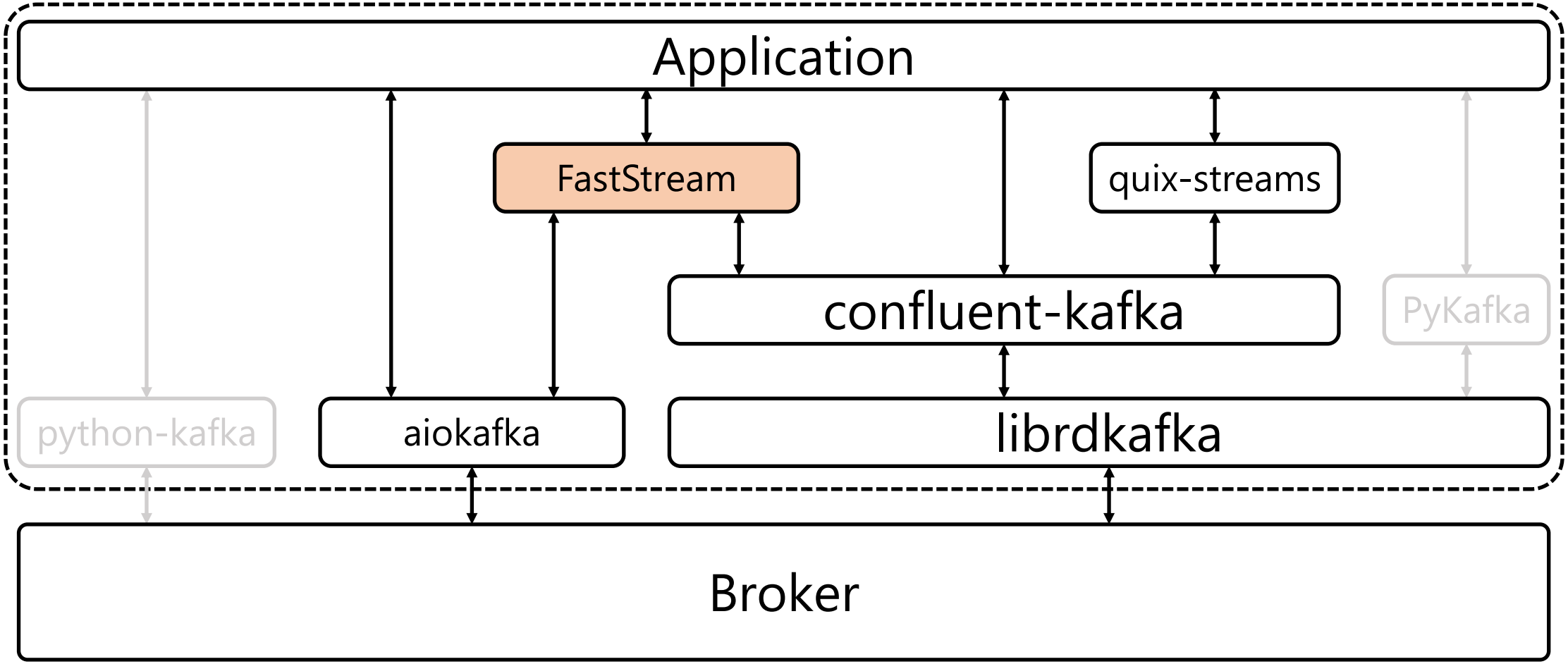
https://github.com/confluentinc/confluent-kafka-python

Зоопарк Python-клиентов



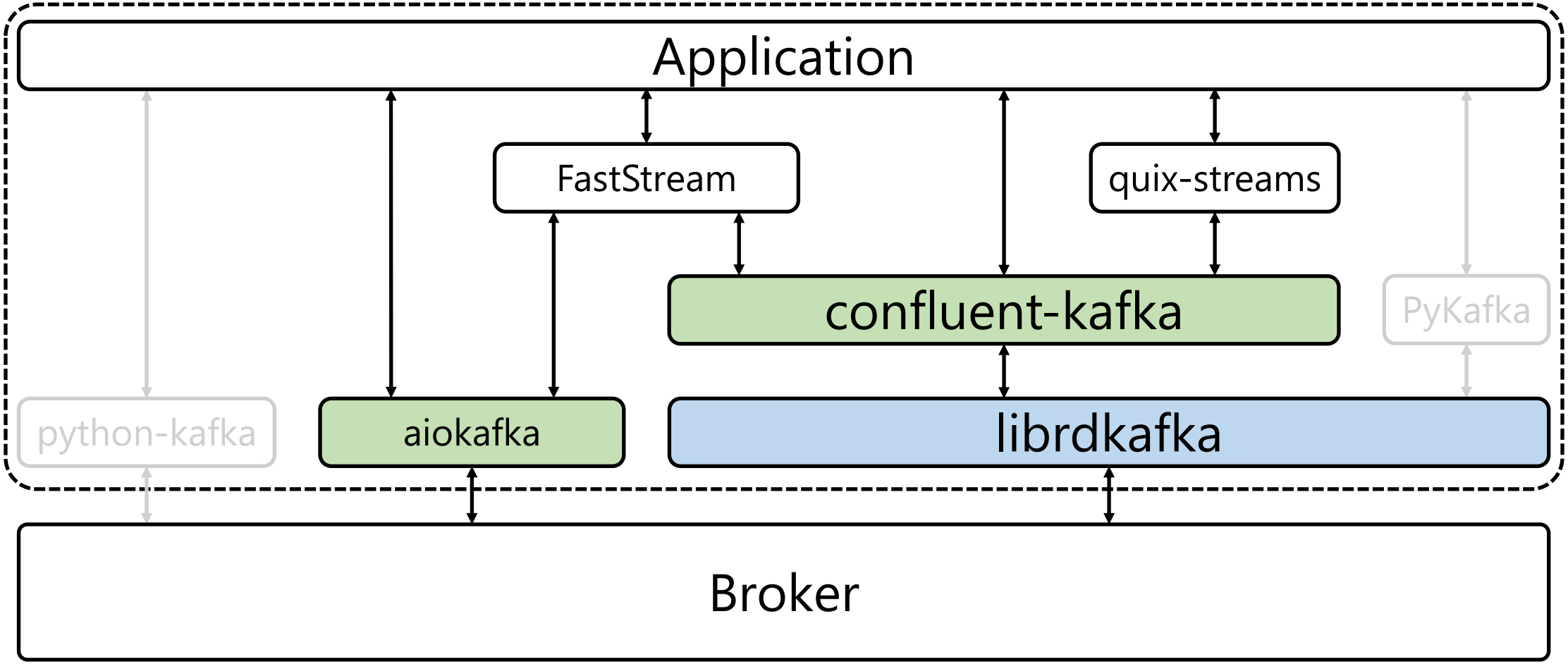
https://github.com/quixio/quix-streams

Зоопарк Python-клиентов

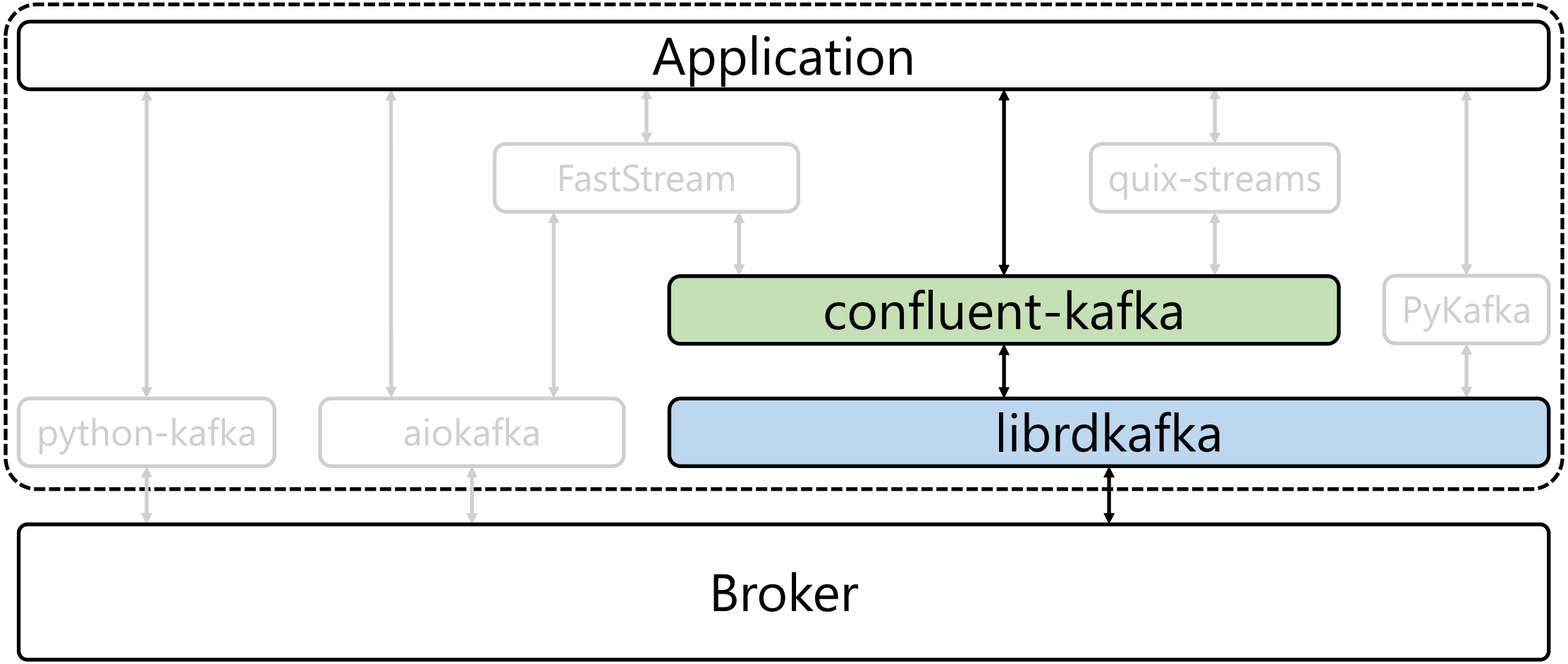


https://github.com/airtai/faststream

Зоопарк Python-клиентов



Зоопарк Python-клиентов



Kafka Producer

Kafka Producer

```
from confluent_kafka import Producer

producer = Producer(conf)
```

Kafka Producer

```
from confluent_kafka import Producer

producer = Producer(conf)
```

Kafka Producer

```
conf = {
    'bootstrap.servers': "kafka1:9092,"
                         "kafka2:9092,"
                         "...",
    # ...
}
```

Kafka Producer

```
def produce(self,
            topic, partition=-1,
            key=None, value=None,
            on_delivery=None,
            timestamp=0,
            headers=None)
```

Kafka Producer

```
def produce(self,
            topic, partition=-1,
            key=None, value=None,
            on_delivery=None,
            timestamp=0,
            headers=None)
```

Kafka Producer

```
def produce(self,
            topic, partition=-1,
            key=None, value=None,
            on_delivery=None,
            timestamp=0,
            headers=None)
```

Kafka Producer

```
def produce(self,
            topic, partition=-1,
            key=None, value=None,
            on_delivery=None,
            timestamp=0,
            headers=None)
```

Kafka Producer

```
def produce(self,
            topic, partition=-1,
            key=None, value=None,
            on_delivery=None,
            timestamp=0,
            headers=None)
```

str или bytes

Kafka Producer

```
def produce(self,
            topic, partition=-1,
            key=None, value=None,
            on_delivery=None,
            timestamp=0,
            headers=None)
```

Kafka Producer

```
def produce(self,
            topic, partition=-1,
            key=None, value=None,
            on_delivery=None,
            timestamp=0,
            headers=None)
```

Callback вызывается как side effect в методах poll и flush

Kafka Producer

```
def produce(self,
            topic, partition=-1,
            key=None, value=None,
            on_delivery=None,
            timestamp=0,
            headers=None)
```

Callback вызывается как side effect в методах poll и flush

```
BufferError: Local: Queue full
```

Kafka Producer

```
def produce(self,
            topic, partition=-1,
            key=None, value=None,
            on_delivery=None,
            timestamp=0,
            headers=None)
```

Kafka Producer

```
def produce(self,
            topic, partition=-1,
            key=None, value=None,
            on_delivery=None,
            timestamp=0,
            headers=None)
```

Поддерживается
с версии 0.10.0.0+

Kafka Producer

```
def produce(self,
            topic, partition=-1,
            key=None, value=None,
            on_delivery=None,
            timestamp=0,
            headers=None)
```

Поддерживается
с версии 0.10.0.0+

Unix-time в миллисекундах *(по умолчанию)*

Kafka Producer

```
def produce(self,
            topic, partition=-1,
            key=None, value=None,
            on_delivery=None,
            timestamp=0,
            headers=None)
```

Kafka Producer

```
def produce(self,
            topic, partition=-1,
            key=None, value=None,
            on_delivery=None,
            timestamp=0,
            headers=None)
```

Поддерживается
с версии 0.11.0.0+

Kafka Producer

Kafka Producer

```
conf = {
    # ...
    'partitioner': "random" |
                   "consistent" |
                   "consistent_random" |
                   "murmur2" |
                   "murmur2_random" |
                   "fnv1a" |
                   "fnv1a_random"
}
```

https://github.com/confluentinc/librdkafka/blob/master/CONFIGURATION.md

Kafka Producer

```
conf = {
    # ...
    'partitioner': "random" |
                   "consistent" |
                   "consistent_random" |
                   "murmur2" |
                   "murmur2_random" |
                   "fnv1a" |
                   "fnv1a_random"
}
```

CRC32

https://github.com/confluentinc/librdkafka/blob/master/CONFIGURATION.md

Kafka Producer

```
conf = {
    # ...
    'partitioner': "random" |
                   "consistent" |
                   "consistent_random" |
                   "murmur2" |
                   "murmur2_random" |
                   "fnv1a" |
                   "fnv1a_random"
}
```

MurMur2

https://github.com/confluentinc/librdkafka/blob/master/CONFIGURATION.md

Kafka Producer

```
conf = {
    # ...
    'partitioner': "random" |
                   "consistent" |
                   "consistent_random" |
                   "murmur2" |
                   "murmur2_random" |
                   "fnv1a" |
                   "fnv1a_random"
}
```

FNV-1a

https://github.com/confluentinc/librdkafka/blob/master/CONFIGURATION.md

Kafka Producer

```
conf = {
    # ...
    'partitioner': "random" |
                   "consistent" |
                   "consistent_random" |
                   "murmur2" |
                   "murmur2_random" |
                   "fnv1a" |
                   "fnv1a_random"
}
```

https://github.com/confluentinc/librdkafka/blob/master/CONFIGURATION.md

Kafka Producer

```
conf = {
    # ...
    'partitioner': "random" |
                   "consistent" |
                   "consistent_random" |
                   "murmur2" |
                   "murmur2_random" |
                   "fnv1a" |
                   "fnv1a_random"
}
```

Выбирается случайная партиция для сообщений с key=None

https://github.com/confluentinc/librdkafka/blob/master/CONFIGURATION.md

Kafka Producer

```
conf = {
    # ...
    'partitioner': "random" |
                   "consistent" |
                   "consistent_random" |
                   "murmur2" |
                   "murmur2_random" |
                   "fnv1a" |
                   "fnv1a_random"
}
```

https://github.com/confluentinc/librdkafka/blob/master/CONFIGURATION.md

Kafka Producer

```
conf = {
    # ...
    'partitioner': "random" |
                   "consistent" |
                   "consistent_random" |
                   "murmur2" |
                   "murmur2_random" |
                   "fnv1a" |
                   "fnv1a_random"
}
```

https://github.com/confluentinc/librdkafka/blob/master/CONFIGURATION.md

Kafka Producer

```
conf = {
    # ...
    'partitioner': "random" |
                   "consistent" |
                   "consistent_random" |
                   "murmur2" |
                   "murmur2_random" |
                   "fnv1a" |
                   "fnv1a_random"
}
```

Все сообщения с key=None попадут в одну партицию!

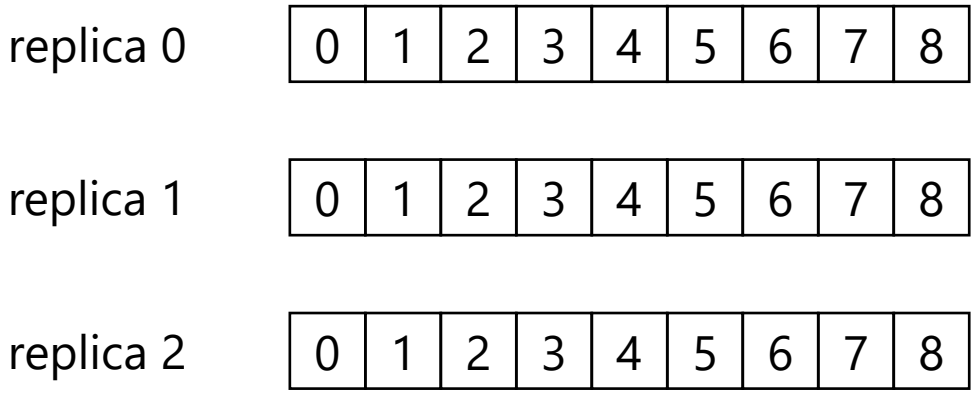
https://github.com/confluentinc/librdkafka/blob/master/CONFIGURATION.md

Kafka Producer

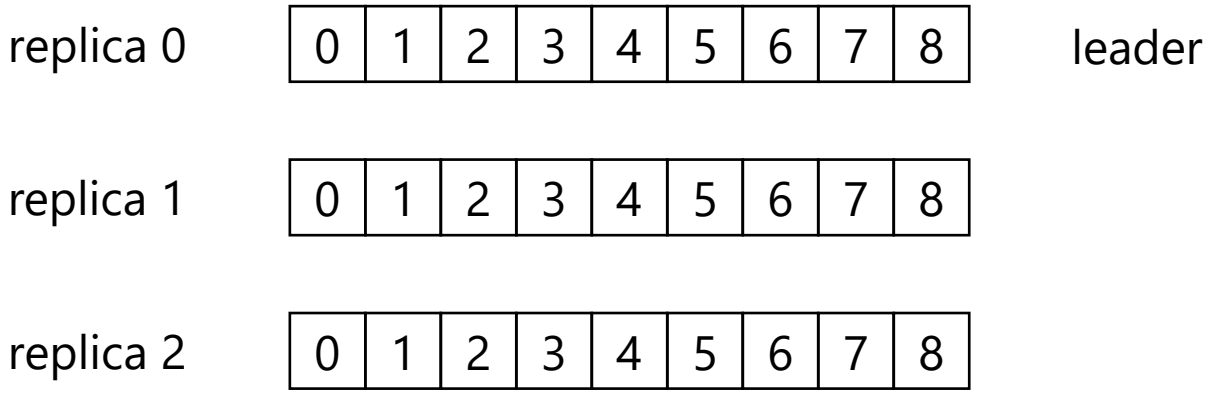
Kafka Producer

```
conf = {
    # ...
    'acks': "0" | "1" | "all"
}
```

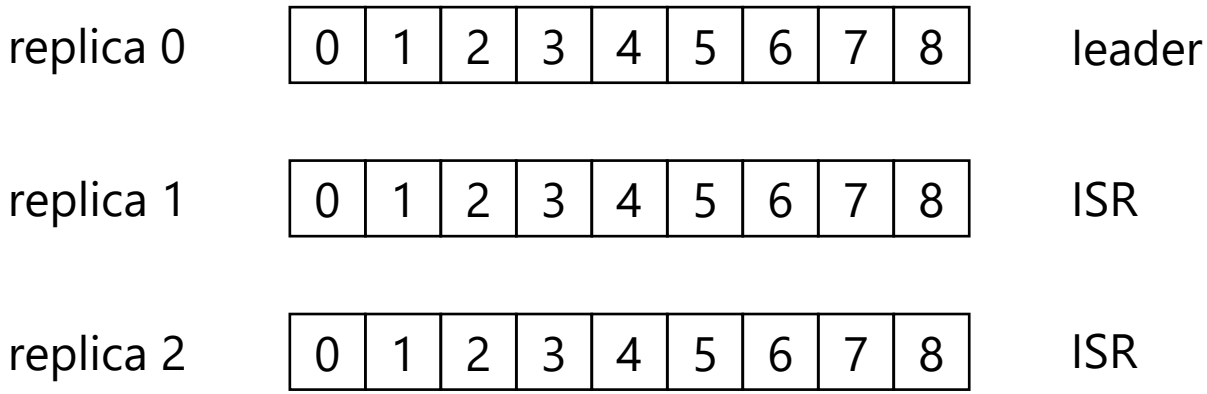
Kafka Producer



Kafka Producer

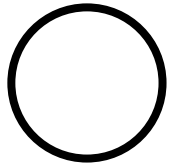


Kafka Producer

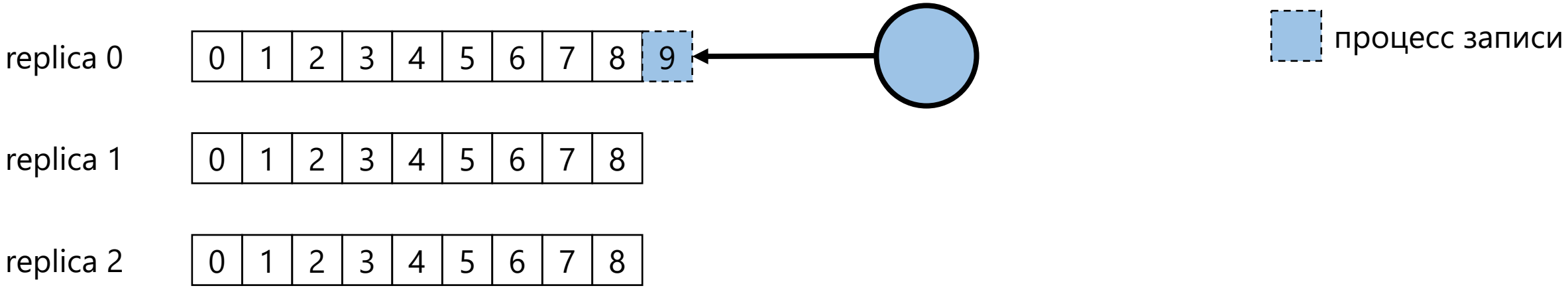


Kafka Producer

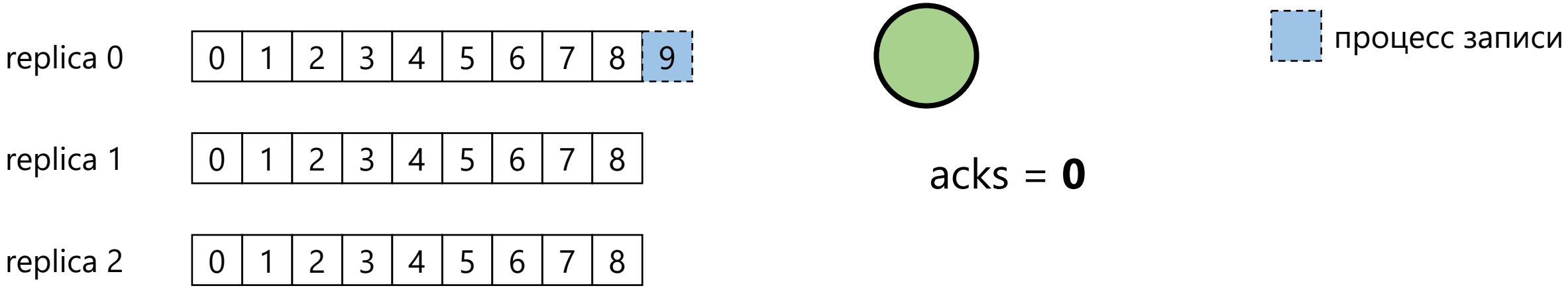
replica 0	0	1	2	3	4	5	6	7	8
replica 1	0	1	2	3	4	5	6	7	8
replica 2	0	1	2	3	4	5	6	7	8



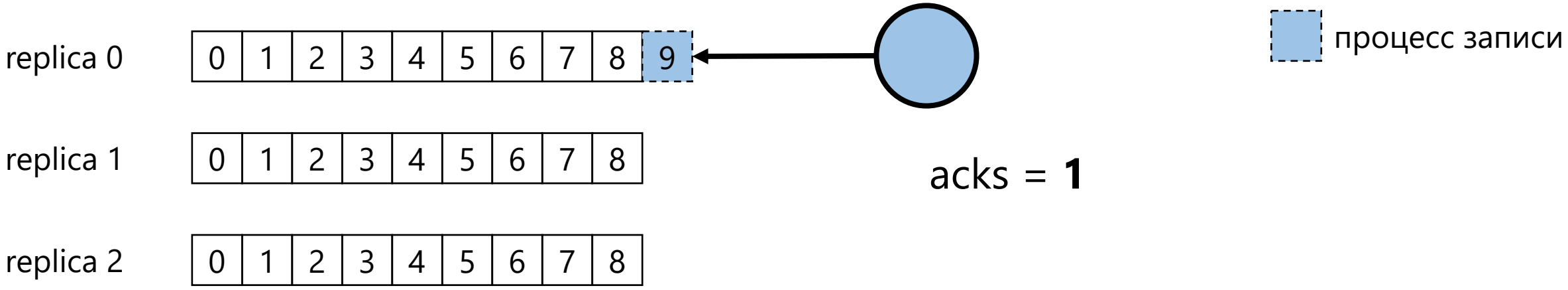
Kafka Producer



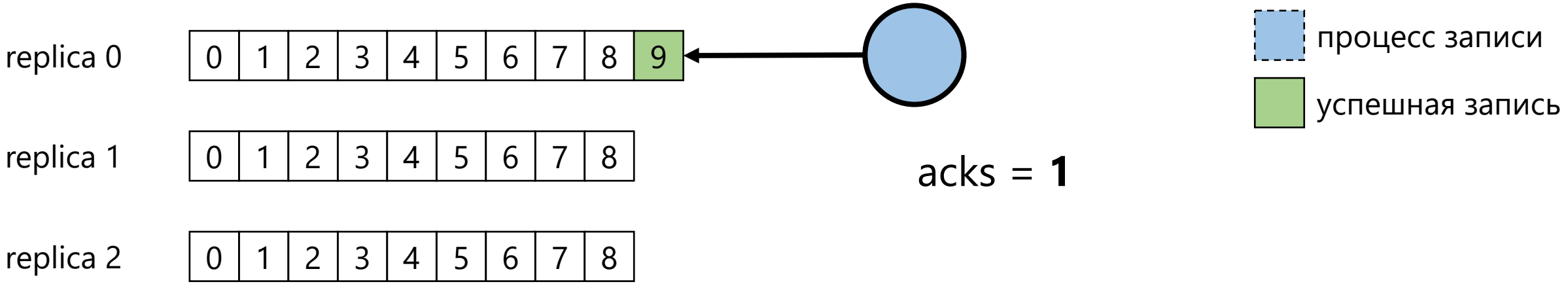
Kafka Producer



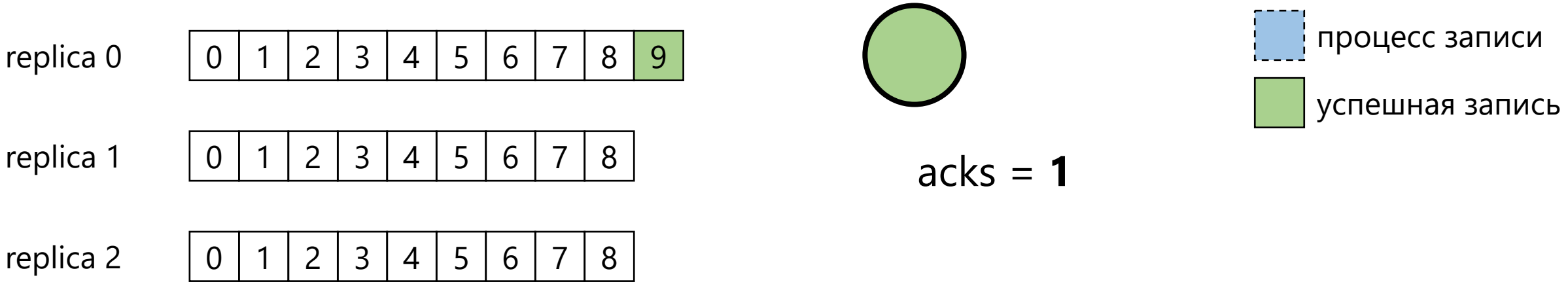
Kafka Producer



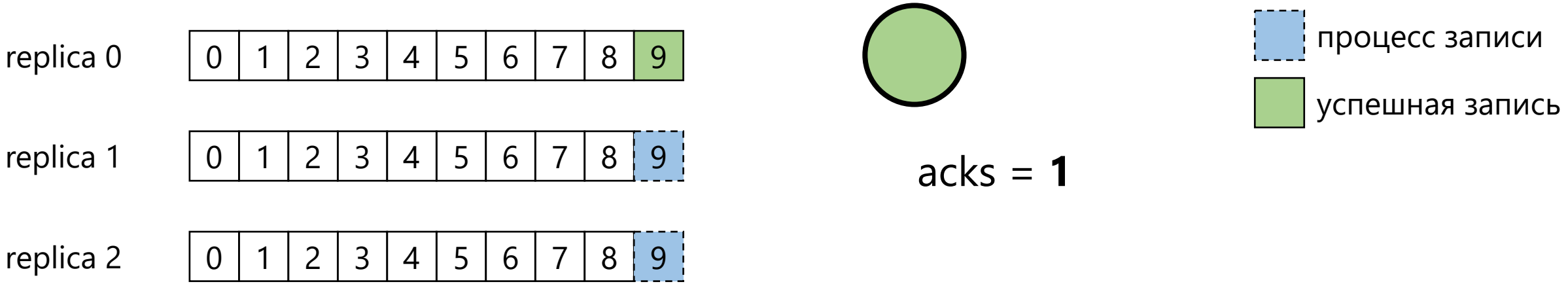
Kafka Producer



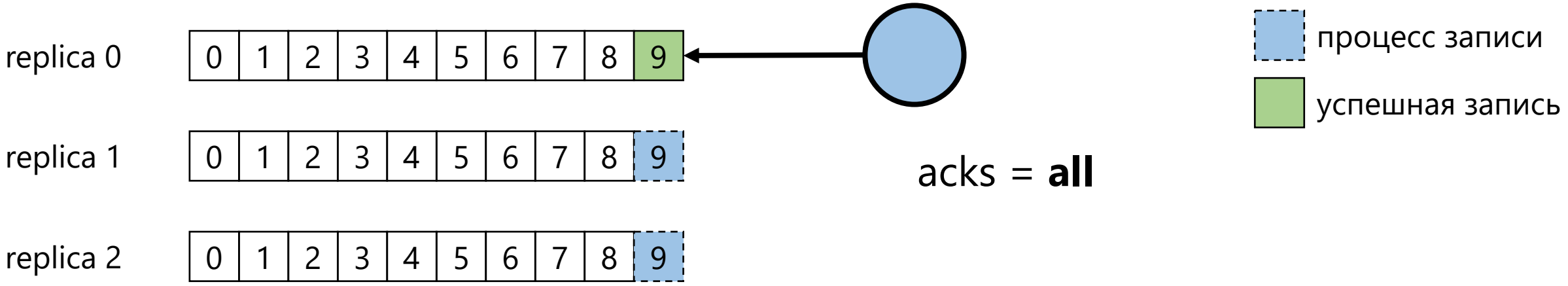
Kafka Producer



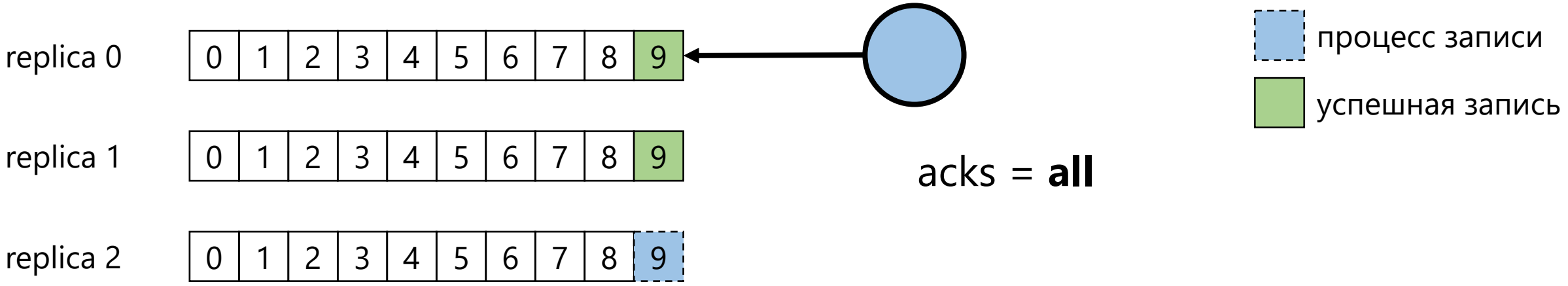
Kafka Producer



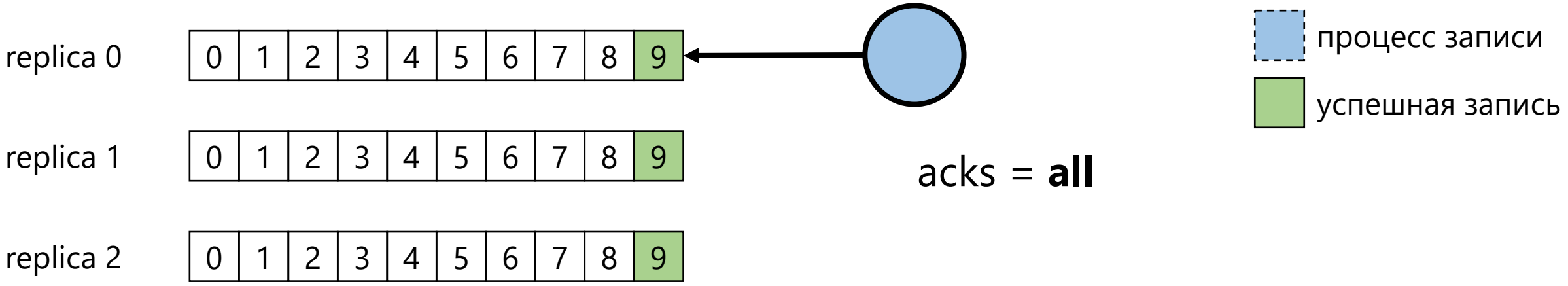
Kafka Producer



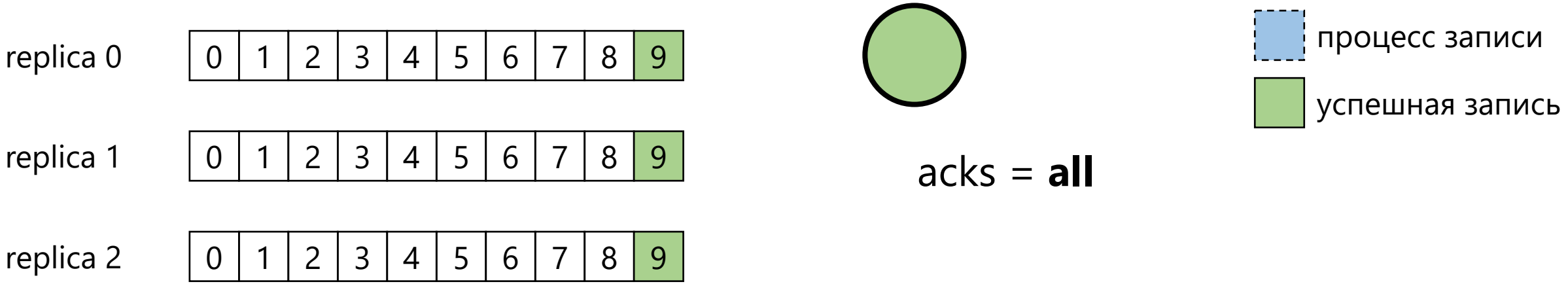
Kafka Producer



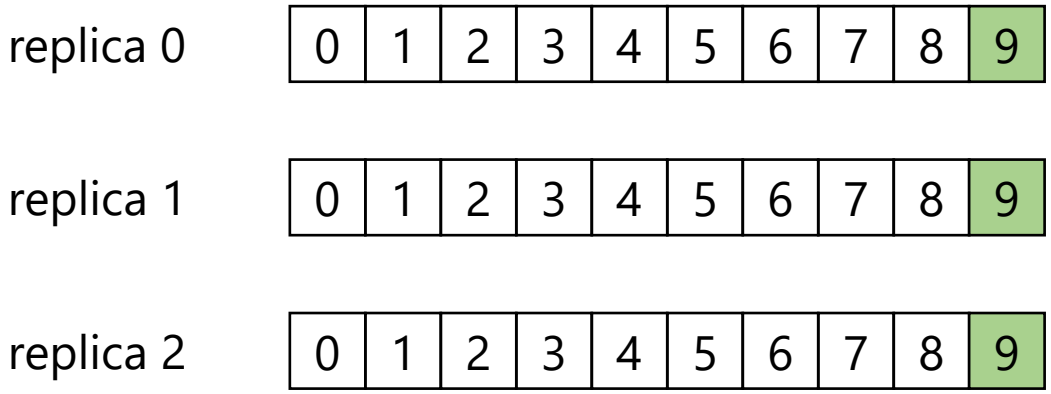
Kafka Producer

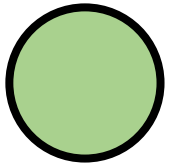


Kafka Producer

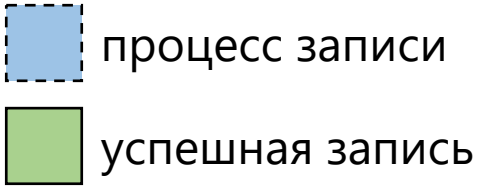


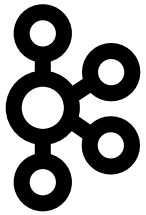
Kafka Producer





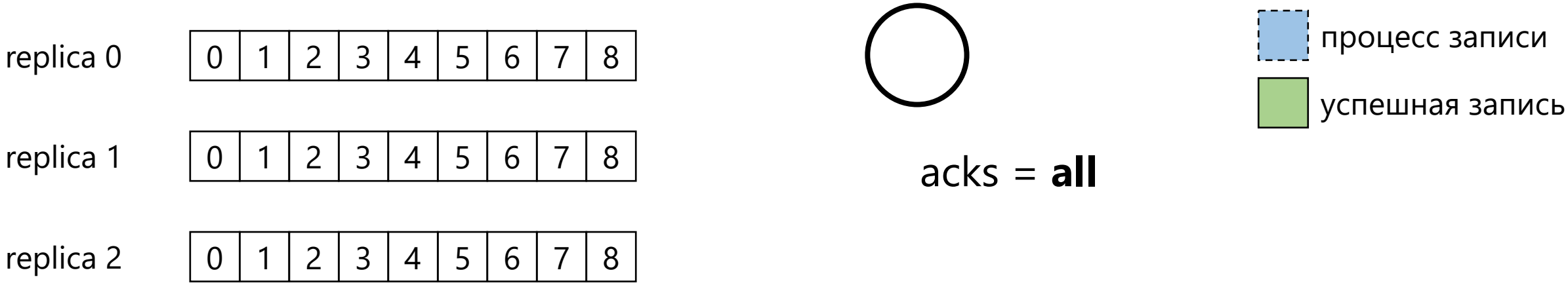
acks = **all**





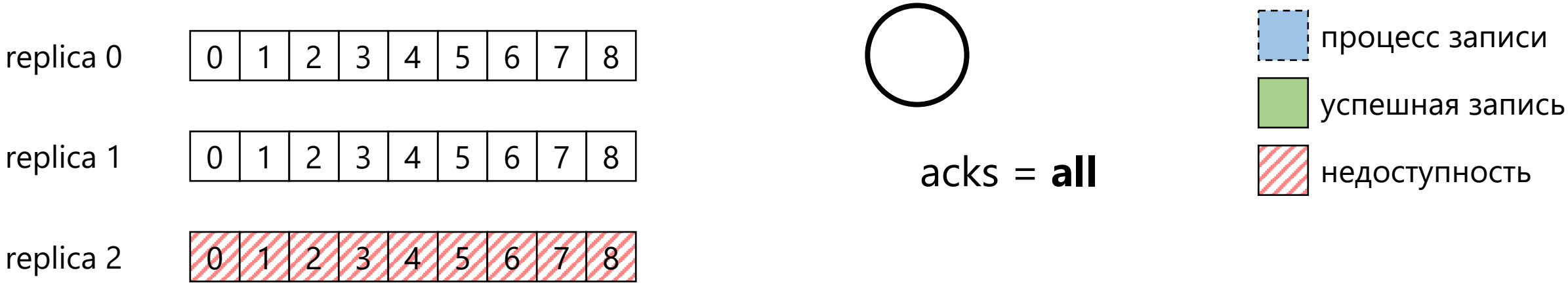
Kafka Producer

Topic config: min.insync.replicas = 2



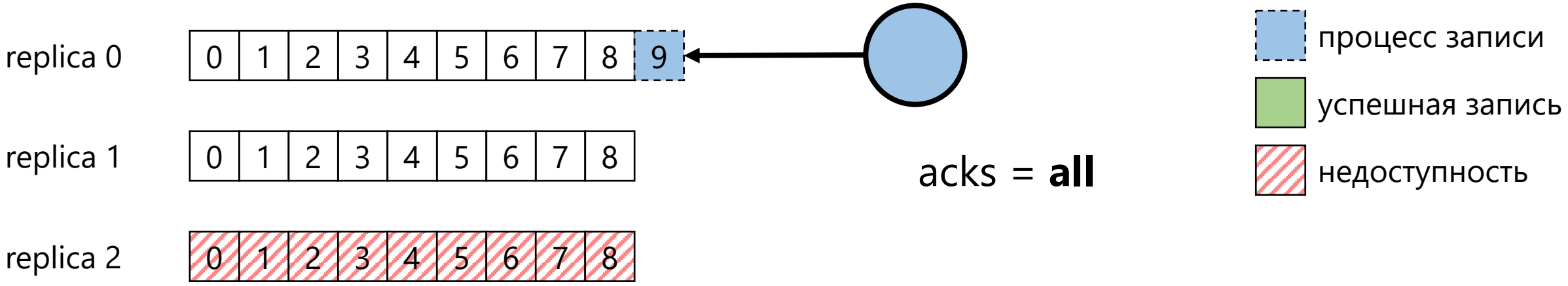
Kafka Producer

Topic config: min.insync.replicas = 2



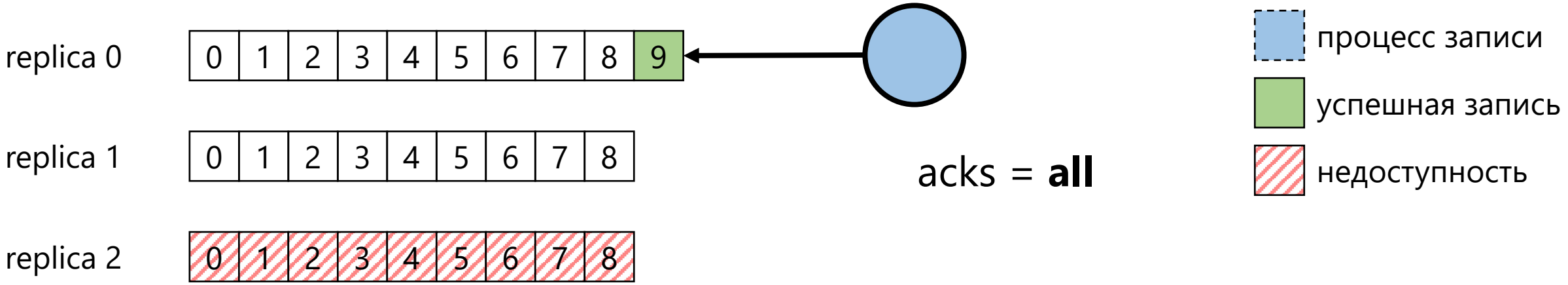
Kafka Producer

Topic config: min.insync.replicas = 2



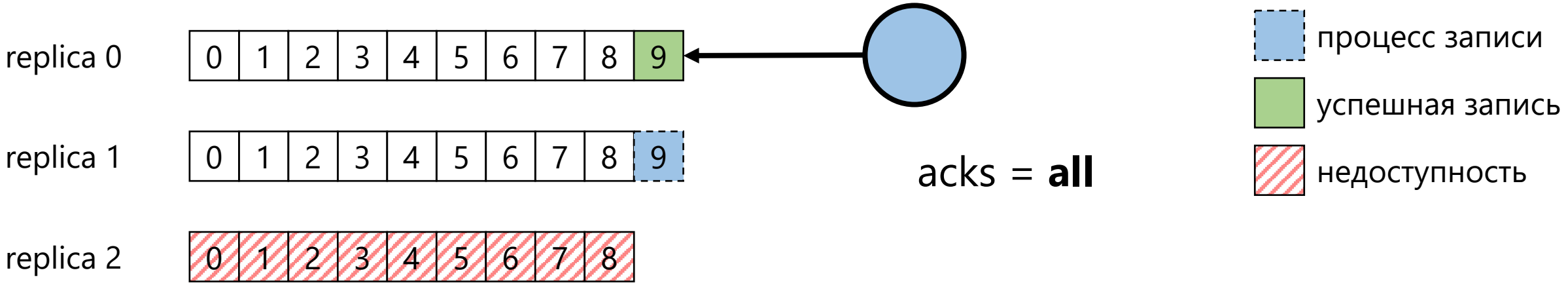
Kafka Producer

Topic config: min.insync.replicas = 2



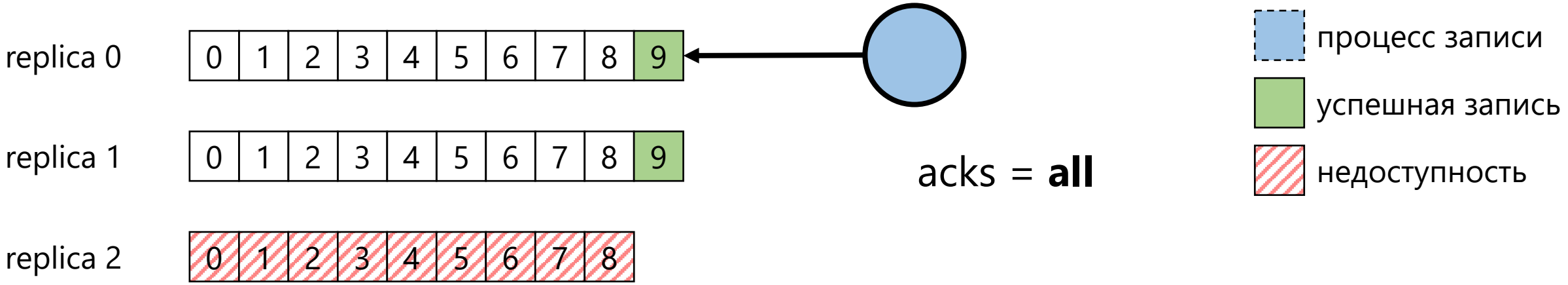
Kafka Producer

Topic config: min.insync.replicas = 2



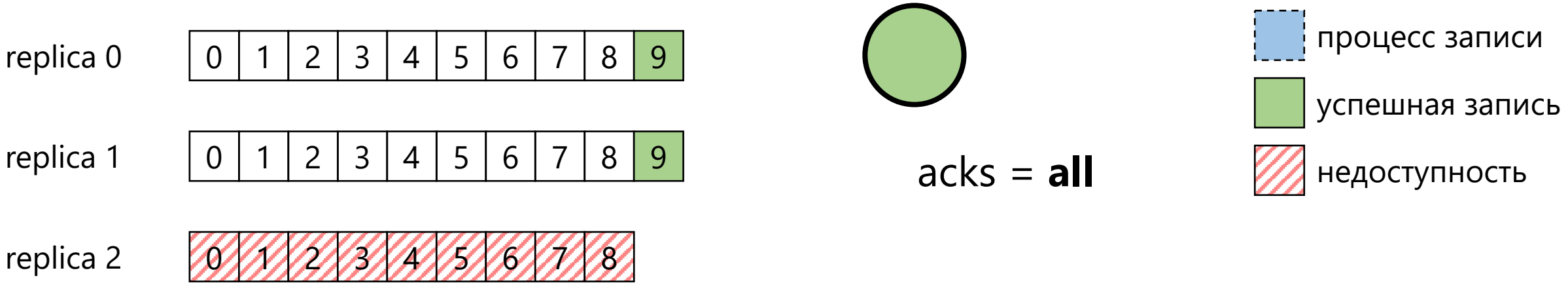
Kafka Producer

Topic config: min.insync.replicas = 2



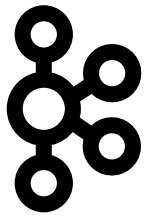
Kafka Producer

Topic config: min.insync.replicas = 2



Kafka Producer

Производительность



Kafka Producer

Производительность

— Низкая задержка

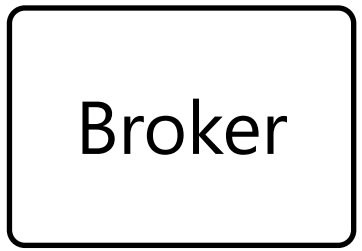
— Высокая пропускная способность

Kafka Producer

Производительность

— Низкая задержка

— Высокая пропускная способность



Kafka Producer

Производительность

— Низкая задержка

— Высокая пропускная способность

Kafka Producer

Производительность

— Низкая задержка

— Высокая пропускная способность

Kafka Producer

Производительность

— Низкая задержка

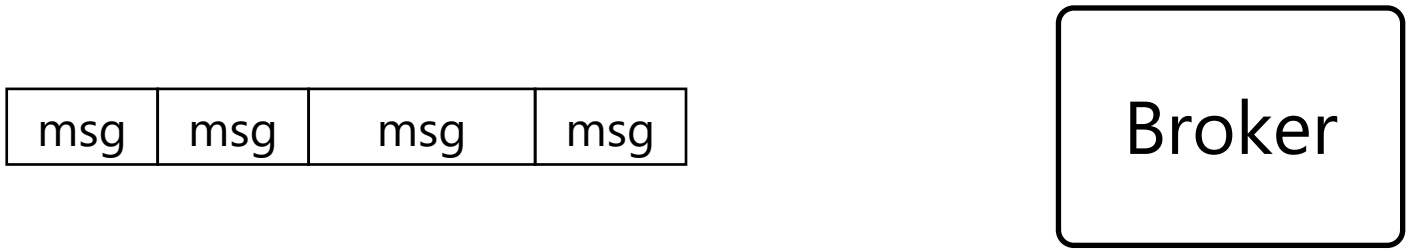
— Высокая пропускная способность

Kafka Producer

Производительность

— Низкая задержка

— Высокая пропускная способность

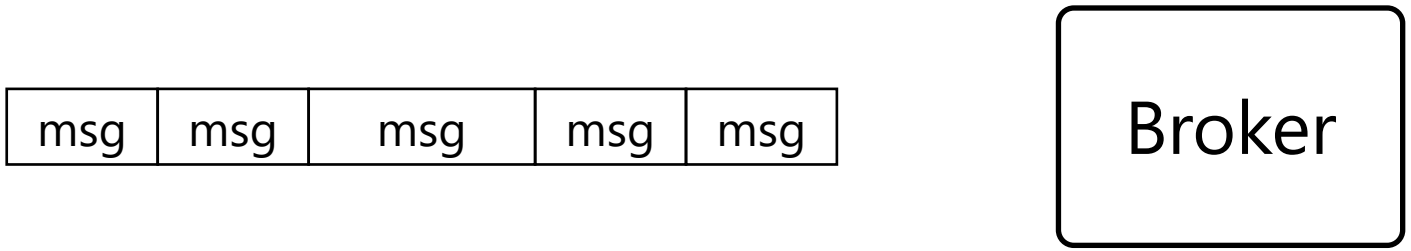


Kafka Producer

Производительность

— Низкая задержка

— Высокая пропускная способность

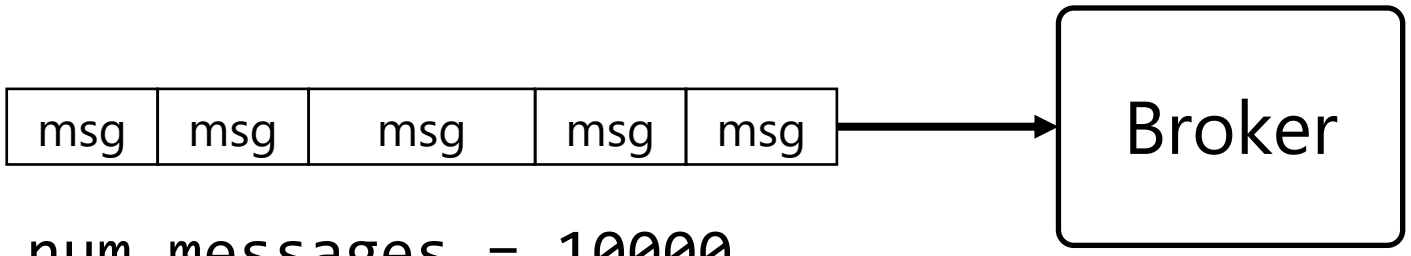


Kafka Producer

Производительность

— Низкая задержка

— Высокая пропускная способность



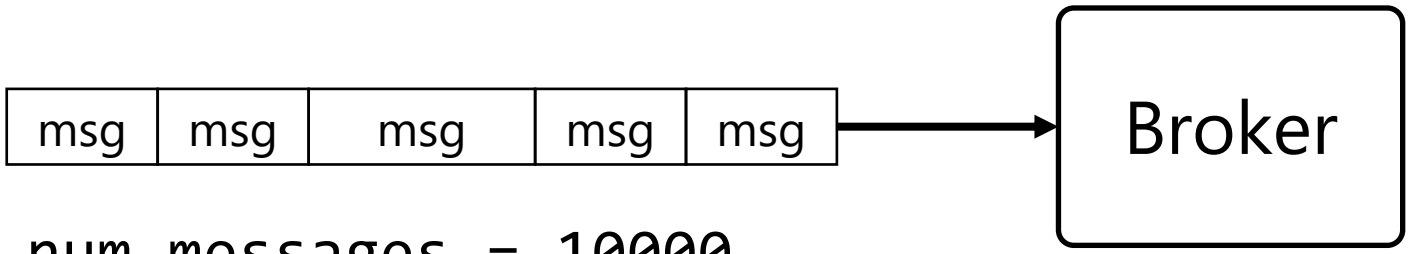

```
batch.num.messages = 10000
```

Kafka Producer

Производительность

— Низкая задержка

— Высокая пропускная способность




```
batch.num.messages = 10000
batch.size = 100000
```

Kafka Producer

Производительность

— Низкая задержка

— Высокая пропускная способность

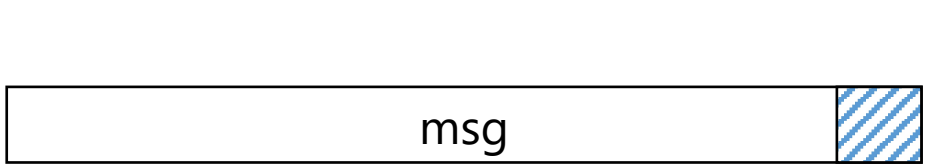

```
batch.num.messages = 10000
batch.size = 100000
```

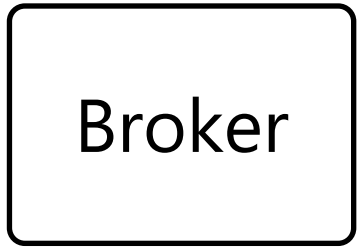
Kafka Producer

Производительность

— Низкая задержка

— Высокая пропускная способность






```
batch.num.messages = 10000
batch.size = 100000
```

Kafka Producer

Производительность

— Низкая задержка

— Высокая пропускная способность


```
batch.num.messages = 10000
batch.size = 100000
message.max.bytes = 1000000
```

Kafka Producer

Производительность

— Низкая задержка

— Высокая пропускная способность

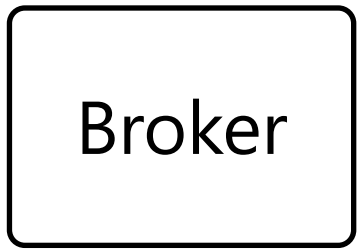

```
batch.num.messages = 10000
batch.size = 100000
message.max.bytes = 1000000
```

Kafka Producer

Производительность

— Низкая задержка

— Высокая пропускная способность



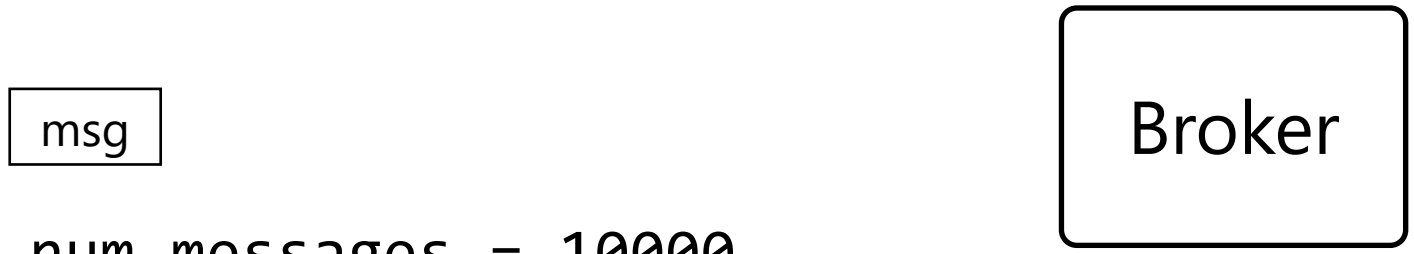

```
batch.num.messages = 10000
batch.size = 100000
message.max.bytes = 1000000
linger.ms = 5
```

Kafka Producer

Производительность

— Низкая задержка

— Высокая пропускная способность




```
batch.num.messages = 10000
batch.size = 100000
message.max.bytes = 1000000
linger.ms = 5
```

Kafka Producer

Производительность

— Низкая задержка

— Высокая пропускная способность

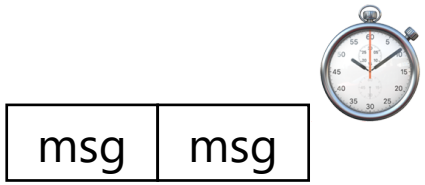

```
batch.num.messages = 10000
batch.size = 100000
message.max.bytes = 1000000
linger.ms = 5
```

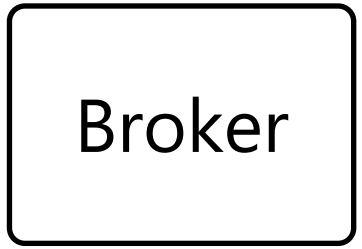
Kafka Producer

Производительность

— Низкая задержка

— Высокая пропускная способность






```
batch.num.messages = 10000
batch.size = 100000
message.max.bytes = 1000000
linger.ms = 5
```

Kafka Producer

Производительность

— Низкая задержка

— Высокая пропускная способность

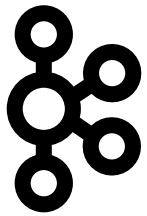

```
batch.num.messages = 10000
batch.size = 100000
message.max.bytes = 1000000
linger.ms = 5
```

Kafka Producer

Производительность

— Низкая задержка

— Высокая пропускная способность



Kafka Producer

```
conf = {
    # ...
    'compression.type': "none" |
                        "gzip" |
                        "snappy" |
                        "lz4" |
                        "zstd"
}
```

https://github.com/confluentinc/librdkafka/blob/master/CONFIGURATION.md

Kafka Producer

```
conf = {
    # ...
    'compression.type': "none" |
                        "gzip" |
                        "snappy" |
                        "lz4" |
                        "zstd"
}
```

https://github.com/confluentinc/librdkafka/blob/master/CONFIGURATION.md

Kafka Producer

Выводы

— Producer выбирает партицию для сообщения

— Producer определяет уровень гарантии доставки

— В Producer можно тюнить производительность

Kafka Consumer

Kafka Consumer

```
from confluent_kafka import Consumer

consumer = Consumer(conf)
```

Kafka Consumer

```
from confluent_kafka import Consumer

consumer = Consumer(conf)
```

Kafka Consumer

```
conf = {
    'bootstrap.servers': "kafka1:9092,"
                         "kafka2:9092,"
                         "...",
    # ...
}
```

Kafka Consumer

```
# Подписаться на список топиков
consumer.subscribe(["topic_1", "topic_2"])
```

```
# Подписаться на топики по шаблону имени
consumer.subscribe(["^topic_[0-9]+"])
```

Kafka Consumer

```
# Подписаться на список топиков
consumer.subscribe(["topic_1", "topic_2"])
```

```
# Подписаться на топики по шаблону имени
consumer.subscribe(["^topic_[0-9]+"])
```

Kafka Consumer

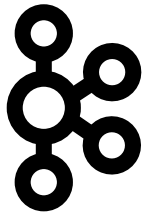
```
# Подписаться на список топиков
consumer.subscribe(["topic_1", "topic_2"])
```

```
# Подписаться на топики по шаблону имени
consumer.subscribe(["^topic_[0-9]+"])
```

Kafka Consumer

```
# Подписаться на список топиков
consumer.subscribe(["topic_1", "topic_2"])
```

```
# Подписаться на топики по шаблону имени
consumer.subscribe(["^topic_[0-9]+"])
```



Kafka Consumer

```
conf = {
    # ...
    'topic.metadata.refresh.interval.ms': 300000
}
```

Kafka Consumer

```
conf = {
    # ...
    'topic.metadata.refresh.interval.ms': 300000
}
```

Kafka Consumer

```
while True:
    msg = consumer.poll(5.0)
    if msg is None:
        continue

    obj = deserialize(msg.value())
    if obj is not None:
        process(obj)
```

Kafka Consumer

```
while True:
    msg = consumer.poll(5.0)
    if msg is None:
        continue

    obj = deserialize(msg.value())
    if obj is not None:
        process(obj)
```

Kafka Consumer

```
while True:
    msg = consumer.poll(5.0)
    if msg is None:
        continue

    obj = deserialize(msg.value())
    if obj is not None:
        process(obj)
```

Kafka Consumer

```
while True:
    msg = consumer.poll(5.0)
    if msg is None:
        continue

    obj = deserialize(msg.value())
    if obj is not None:
        process(obj)
```

Kafka Consumer

```
while True:
    msg = consumer.poll(5.0)
    if msg is None:
        continue

    obj = deserialize(msg.value())
    if obj is not None:
        process(obj)
```

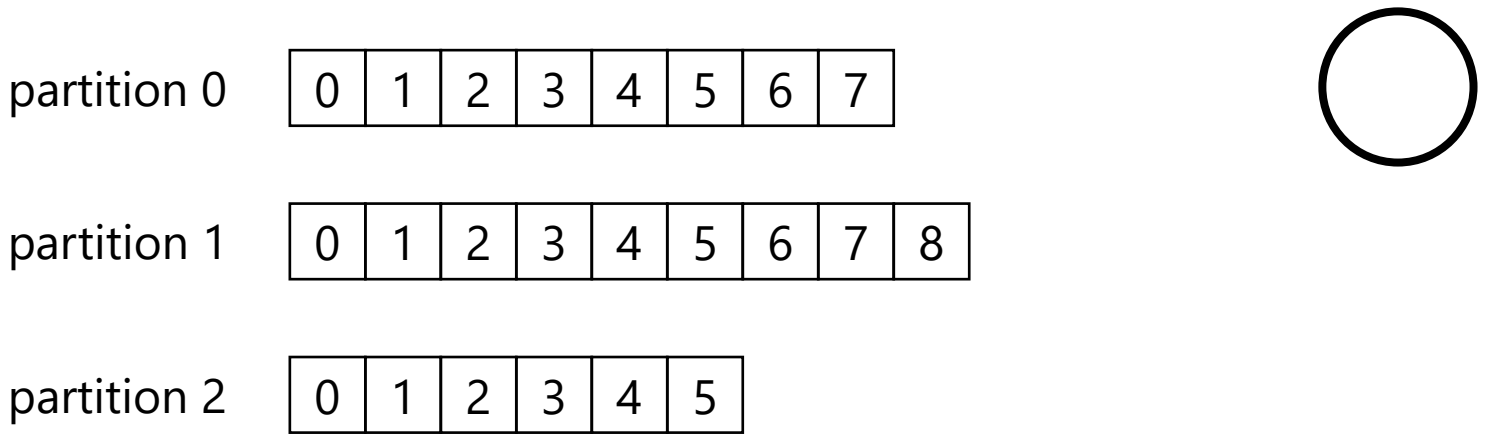
Kafka Consumer

```
while True:
    msgs = consumer.consume(
        num_messages=500,
        timeout=5.0)

    # ...
```

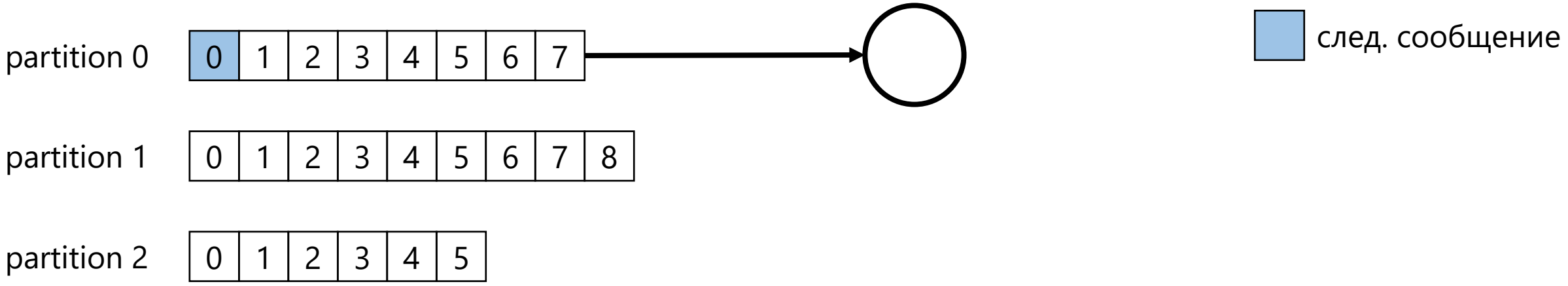
Kafka Consumer

```
while True:
    msgs = consumer.consume(
        num_messages=500,
        timeout=5.0)

    # ...
```

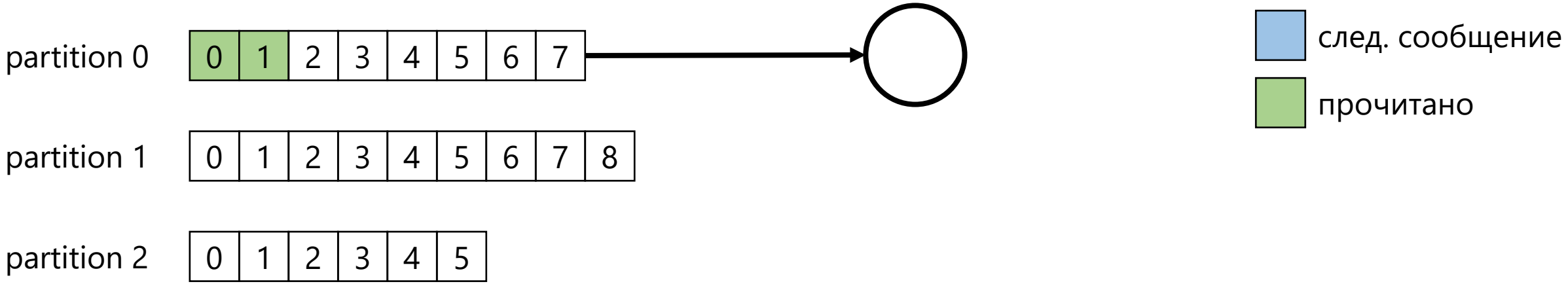
Kafka Consumer



Kafka Consumer

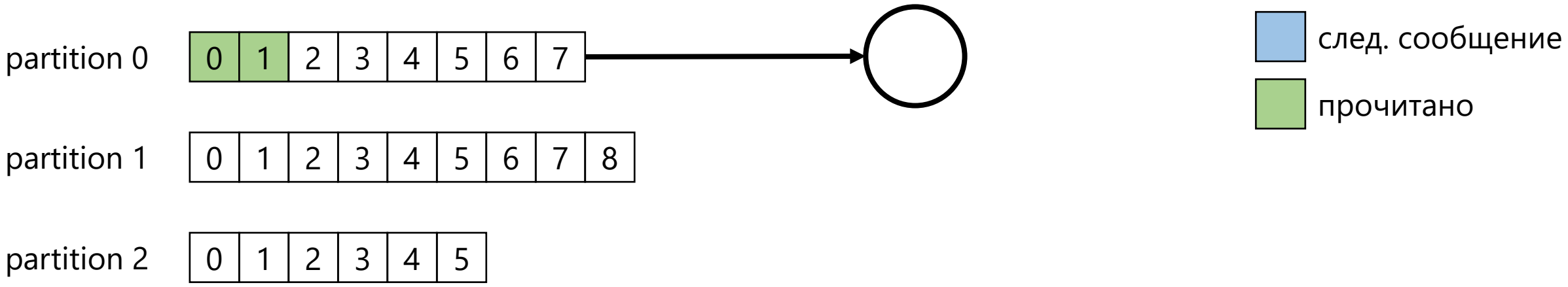


Kafka Consumer

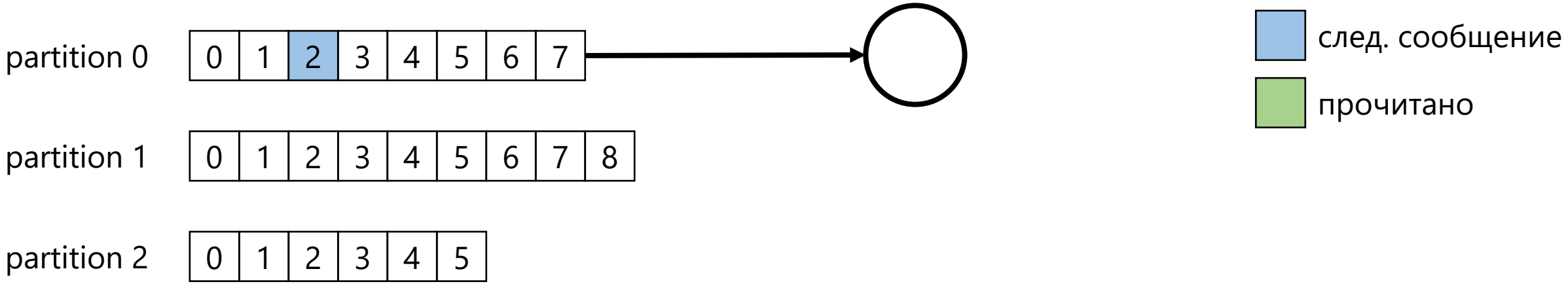


Kafka Consumer

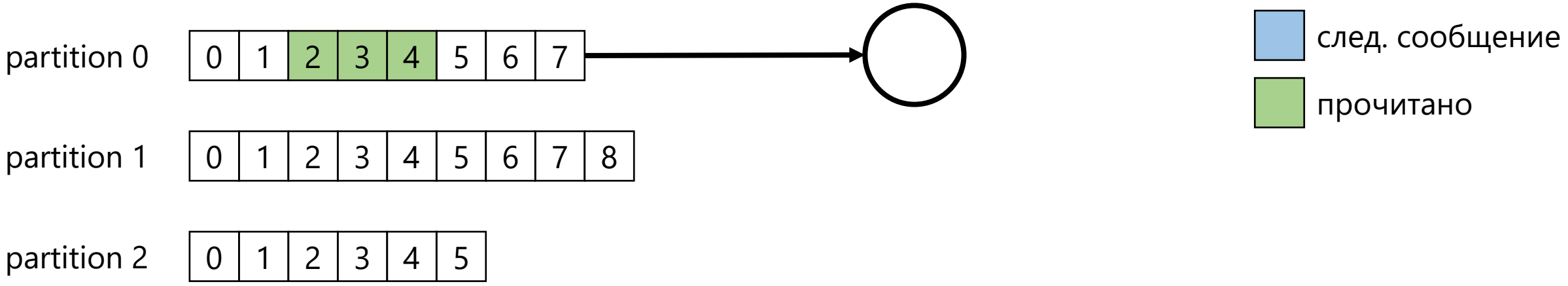
```
fetch.message.max.bytes = 1048576
```



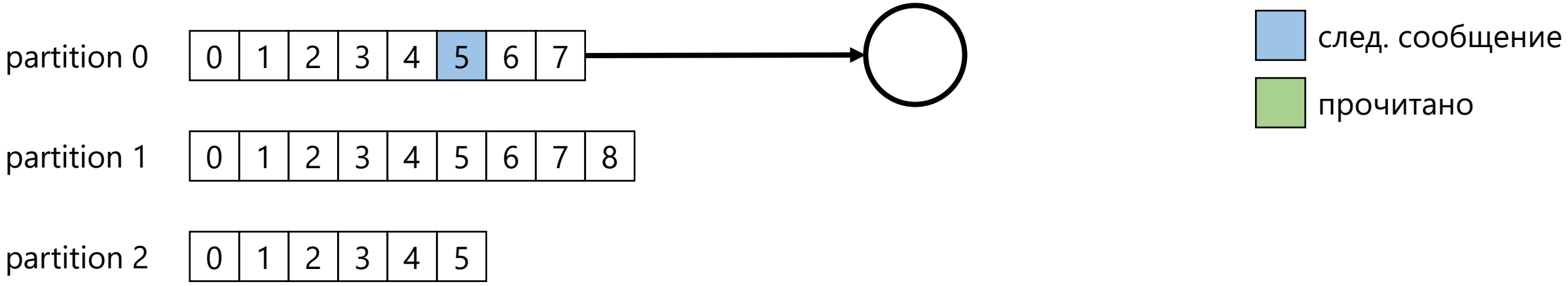
Kafka Consumer



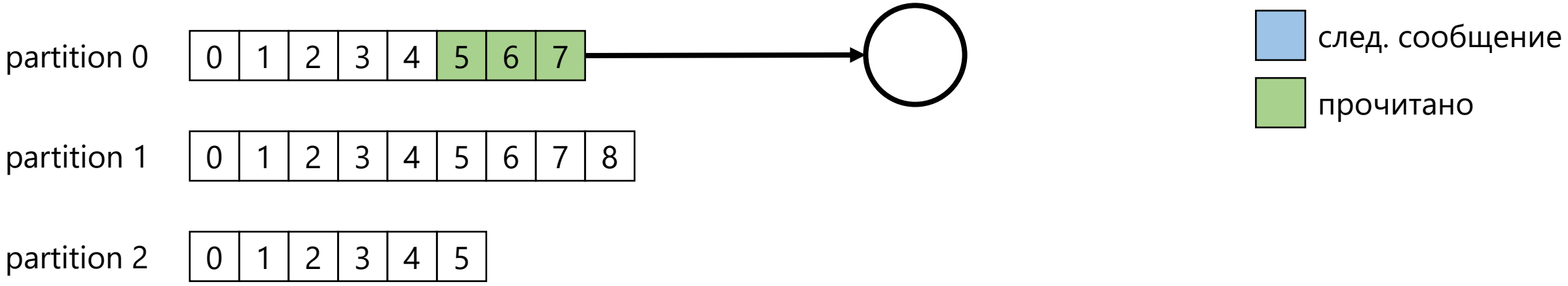
Kafka Consumer



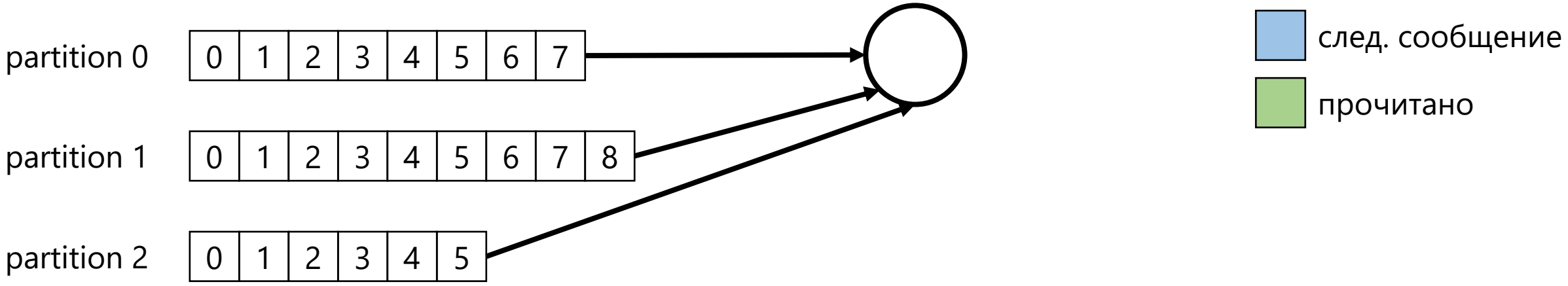
Kafka Consumer



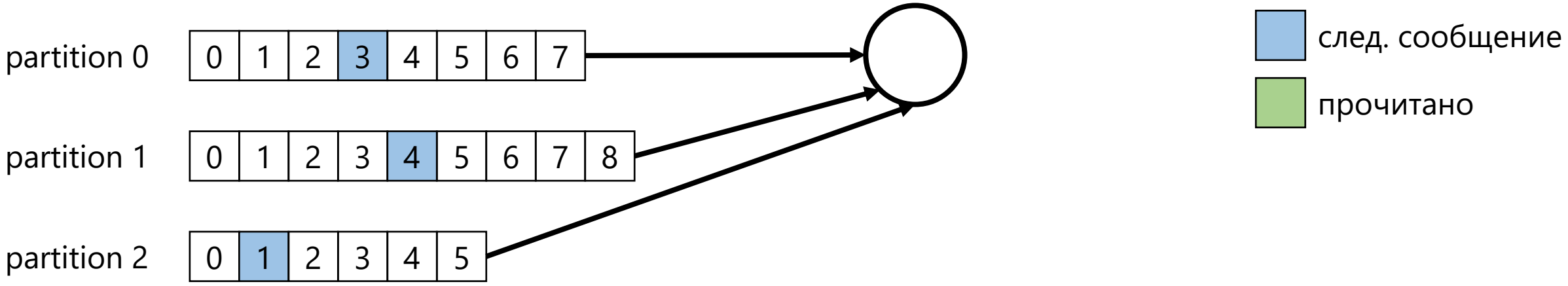
Kafka Consumer



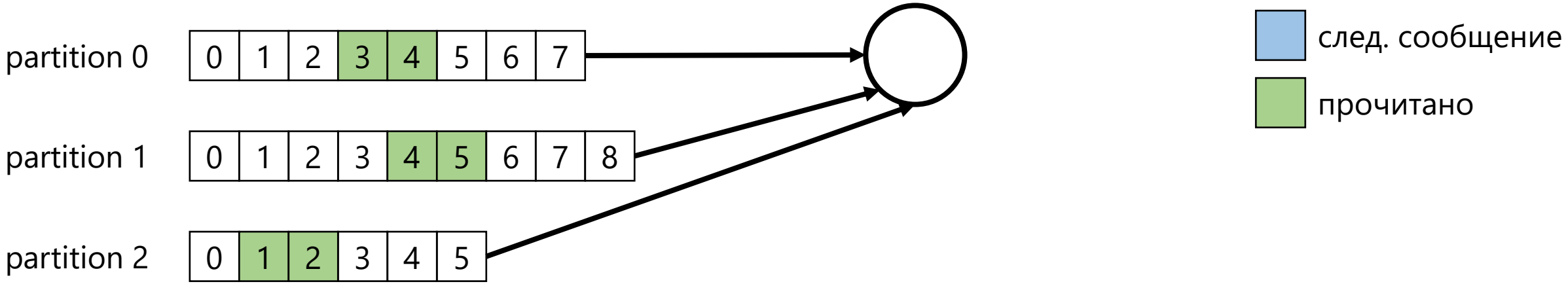
Kafka Consumer



Kafka Consumer

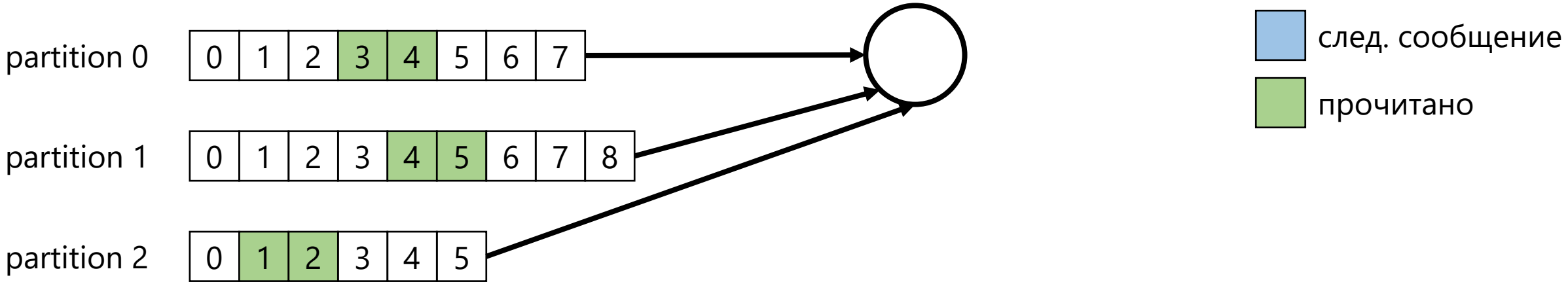


Kafka Consumer



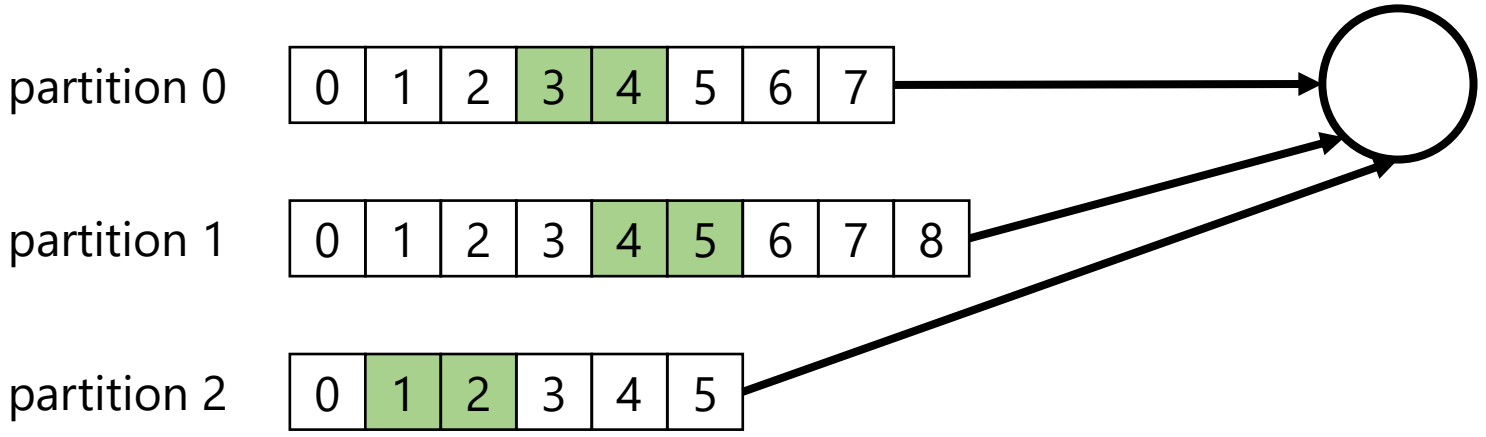
Kafka Consumer

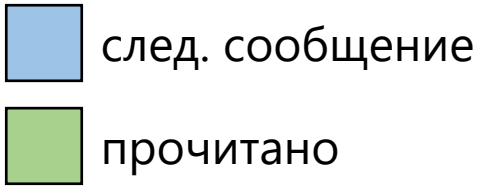
```
fetch.message.max.bytes = 1048576
fetch.max.bytes = 52428800
```



Kafka Consumer

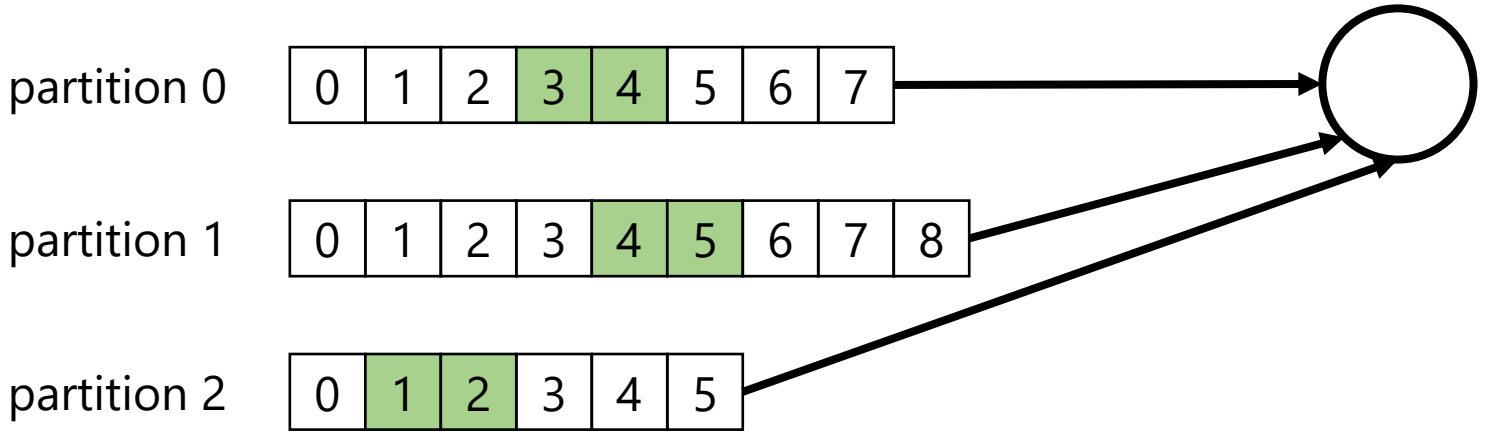
```
fetch.message.max.bytes = 1048576
fetch.max.bytes = 52428800
```

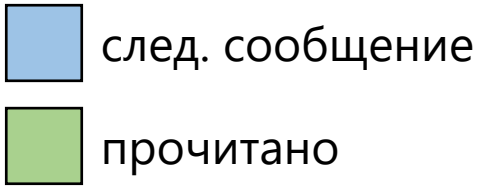




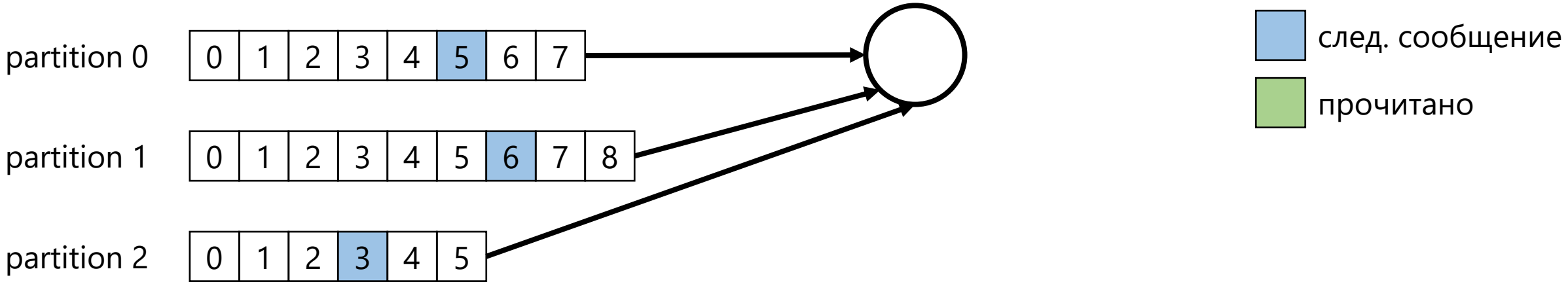
Kafka Consumer

```
fetch.message.max.bytes = 1048576
fetch.max.bytes = 52428800
```

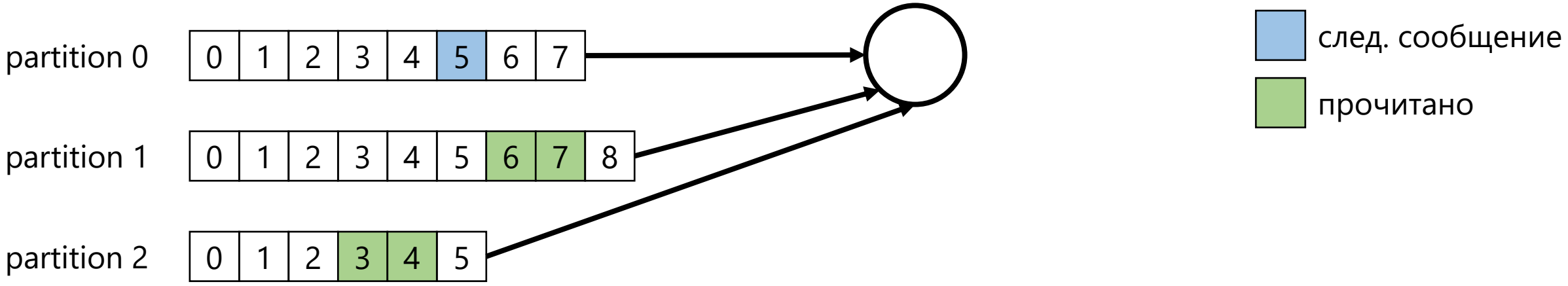




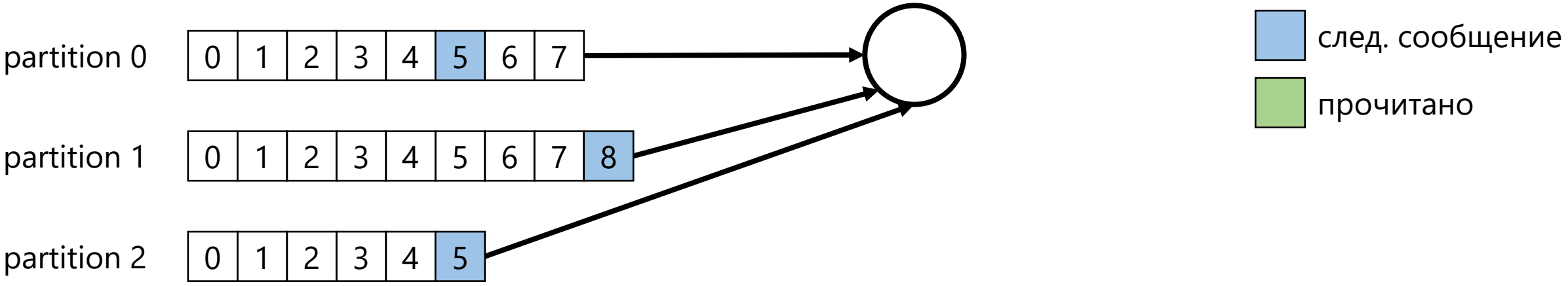
Kafka Consumer



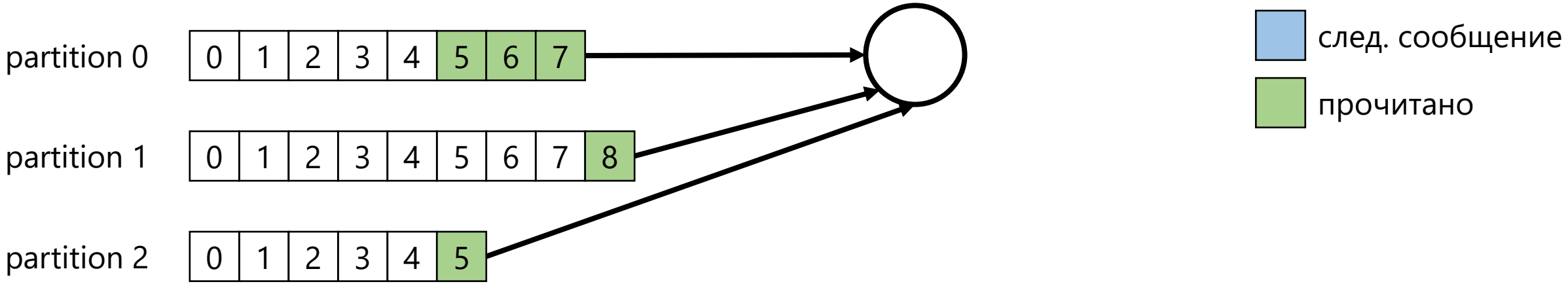
Kafka Consumer



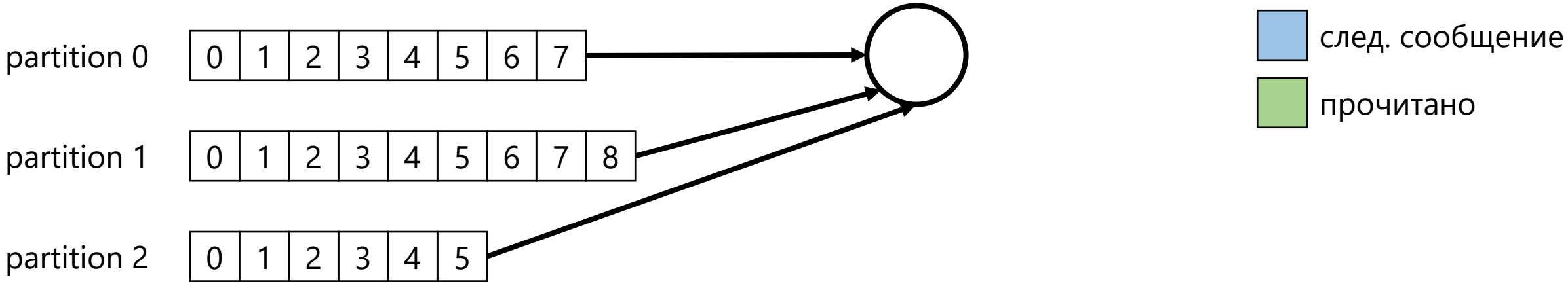
Kafka Consumer



Kafka Consumer



Kafka Consumer



Kafka Consumer

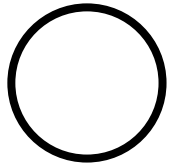
Commit offset

Kafka Consumer

Commit offset

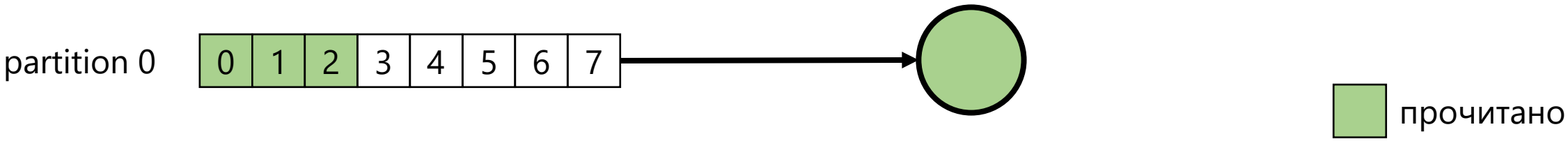
partition 0

0	1	2	3	4	5	6	7



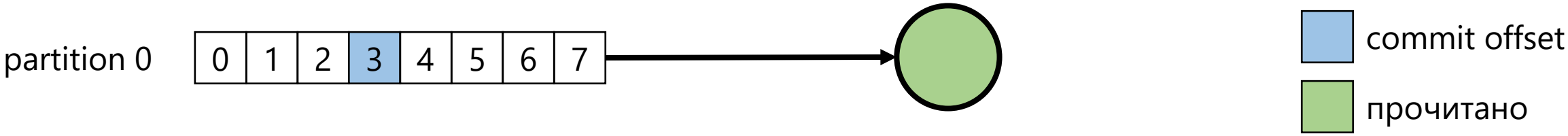
Kafka Consumer

Commit offset



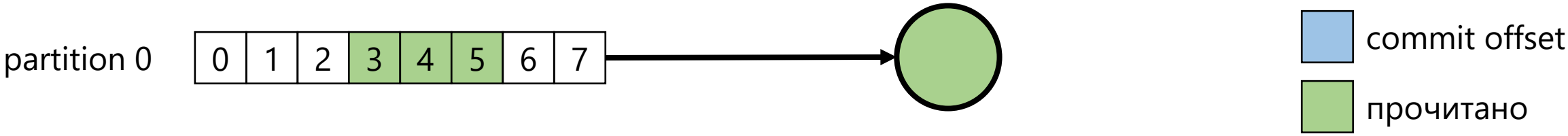
Kafka Consumer

Commit offset



Kafka Consumer

Commit offset



Kafka Consumer

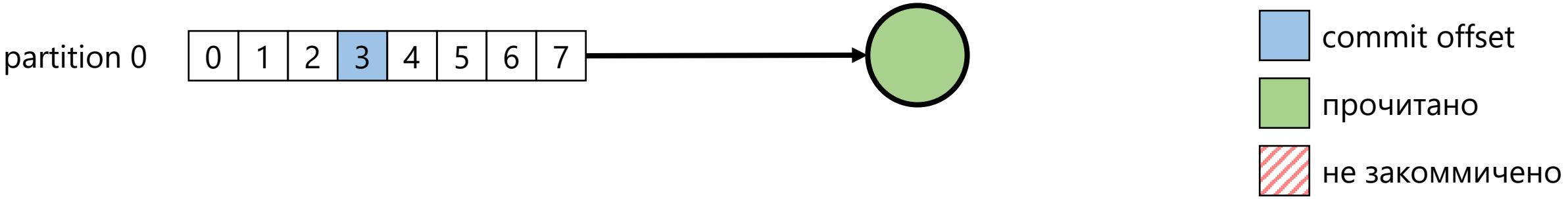
Commit offset

Kafka Consumer

Commit offset

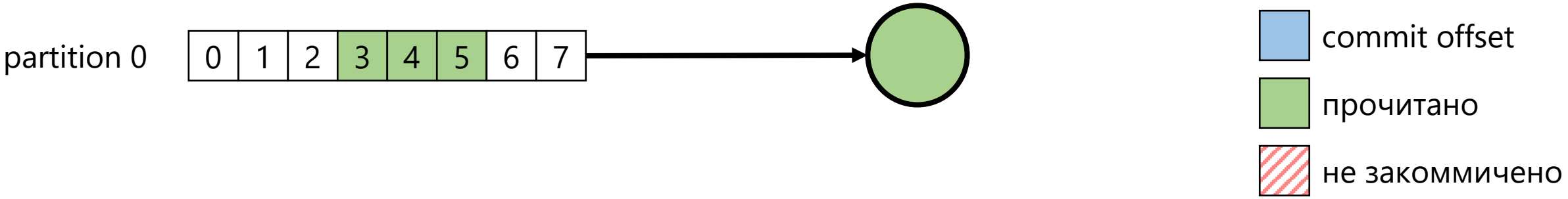
Kafka Consumer

Commit offset



Kafka Consumer

Commit offset



Kafka Consumer

Гарантии обработки

— at least once

— mostly once

Kafka Consumer

```
while True:
    msg = consumer.poll(5.0)
    if msg is None:
        continue

    process(msg)
    consumer.commit()
```

Kafka Consumer

```
while True:
    msg = consumer.poll(5.0)
    if msg is None:
        continue

    process(msg)
    consumer.commit()
```

Kafka Consumer

```
conf = {
    # ...
    'enable.auto.offset.store': True | False
}
```

Kafka Consumer

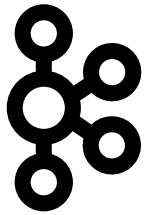
```
conf = {
    # ...
    'enable.auto.offset.store': True | False
}
```

Kafka Consumer

```
while True:
    msg = consumer.poll(5.0)
    if msg is None:
        continue

    process(msg)
    consumer.commit(asynchronous=True)
```

Kafka Consumer

```
while True:
    msg = consumer.poll(5.0)
    if msg is None:
        continue

    process(msg)
    consumer.commit(asynchronous=True)
```



Kafka Consumer

```
while True:
    msg = consumer.poll(5.0)
    if msg is None:
        continue

    process(msg)
    consumer.commit(asynchronous=True)
```

at least once
Commit после обработки

Kafka Consumer

```
while True:
    msg = consumer.poll(5.0)
    if msg is None:
        continue

    consumer.commit(asynchronous=True)
    process(msg)
```

Kafka Consumer

```
while True:
    msg = consumer.poll(5.0)
    if msg is None:
        continue

    consumer.commit(asynchronous=False)
    process(msg)
```

Kafka Consumer

```
while True:
    msg = consumer.poll(5.0)
    if msg is None:
        continue

    consumer.commit(asynchronous=False)
    process(msg)
```

mostly once
Commit до обработки

Kafka Consumer

— 1 коммит на 1 сообщение

— 1 коммит на N сообщений

Kafka Consumer

— 1 коммит на 1 сообщение

— 1 коммит на N сообщений ✓

⇧ производительность

Kafka Consumer

```
while True:
    msg = consumer.poll(5.0)
    if msg is None:
        continue

    process(msg)
    consumer.commit(
        message=None,
        offsets=None,
        asynchronous=True)
```

Kafka Consumer

```
while True:
    msg = consumer.poll(5.0)
    if msg is None:
        continue

    process(msg)
    consumer.commit(
        message=None,
        offsets=None,
        asynchronous=True)
```

Kafka Consumer

```
conf = {
    # ...
    'enable.auto.commit': True | False,
    'auto.commit.interval.ms': 5000
}
```

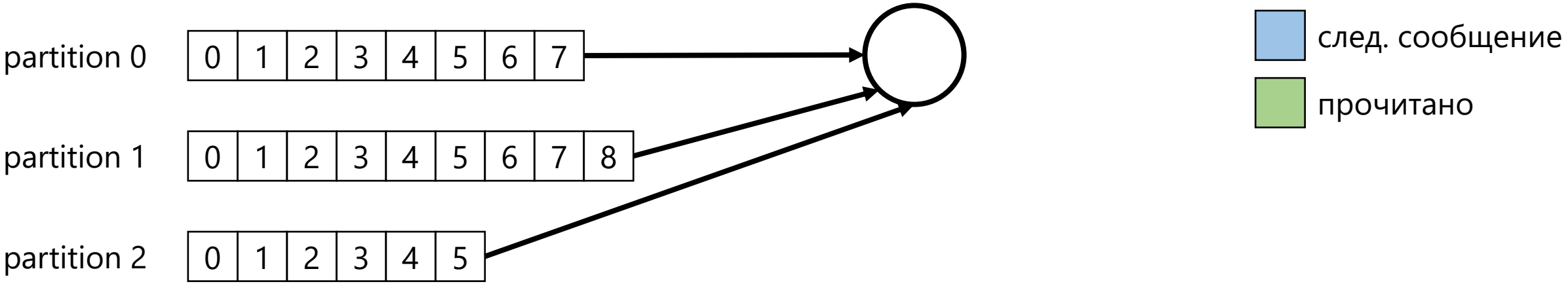
Kafka Consumer

```
conf = {
    # ...
    'enable.auto.commit': True | False,
    'auto.commit.interval.ms': 5000
}
```

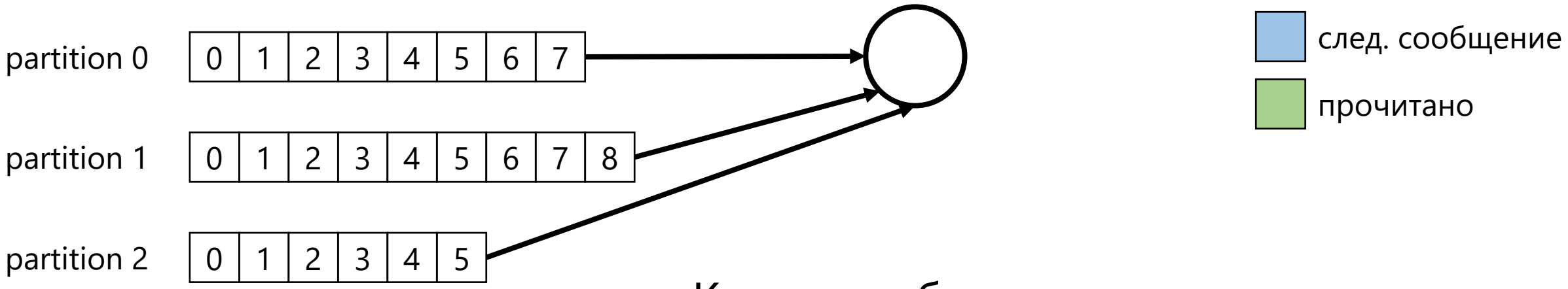
Kafka Consumer

```
while True:
    msg = consumer.poll(5.0)
    if msg is None:
        continue

    process(msg)

    # ...
```

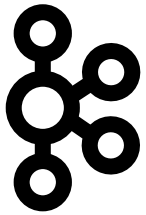
Kafka Consumer



Kafka Consumer



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



Kafka Consumer

```
conf = {
    # ...
    'auto.offset.reset':
        "earliest" |
        "latest" |
        "none"
}
```

Kafka Consumer

```
conf = {
    # ...
    'auto.offset.reset':
        "earliest" |
        "latest" |
        "none"
}
```

Kafka Consumer

```
conf = {
    # ...
    'auto.offset.reset':
        "earliest" |
        "latest" |
        "none"
}
```

Kafka Consumer

```
conf = {
    # ...
    'auto.offset.reset':
        "earliest" |
        "latest" |
        "none"
}
```

⚠️ Значение по умолчанию

Kafka Consumer

```
conf = {
    # ...
    'auto.offset.reset':
        "earliest" |
        "latest" |
        "none"
}
```

Kafka Consumer

```
conf = {
    # ...
    'auto.offset.reset':
        "earliest" |
        "latest" |
        "none"
}
```

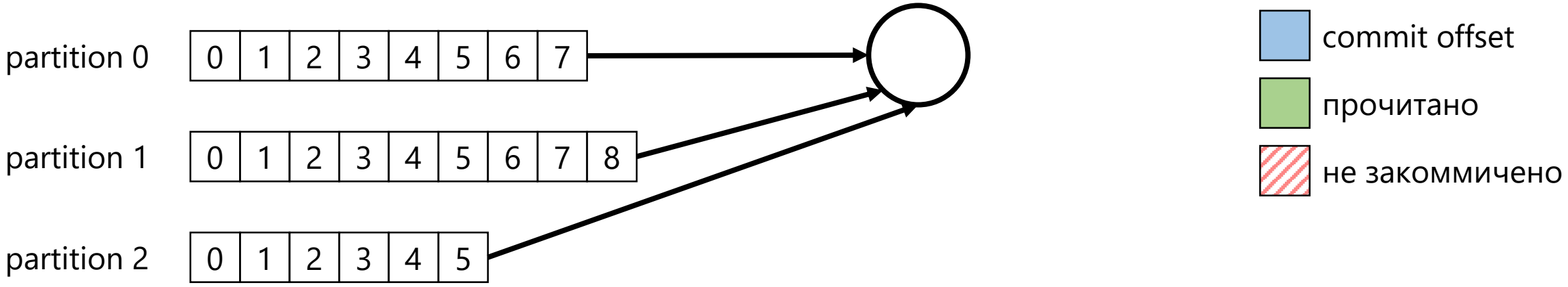
⚠️ Если нет сохранённого Commit Offset — будет exception

Kafka Consumer

Consumer Group

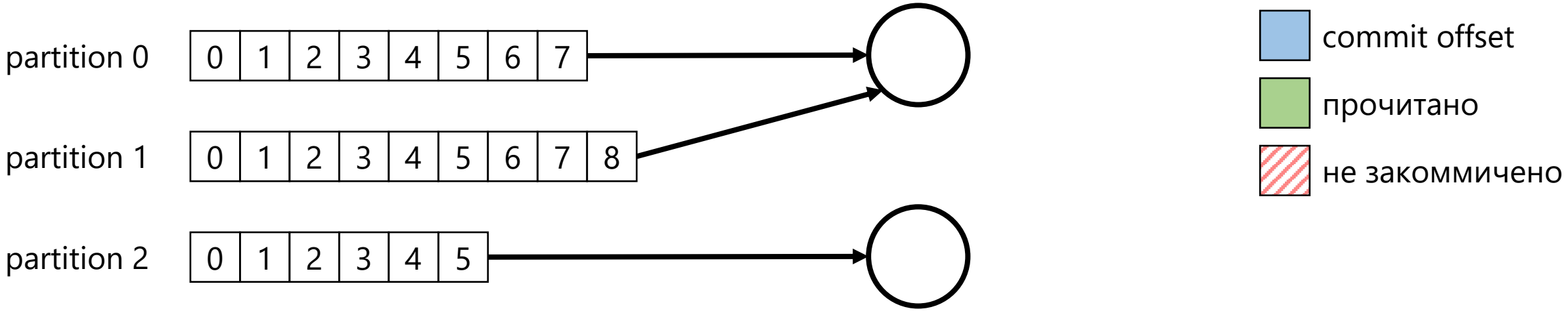
Kafka Consumer

Consumer Group



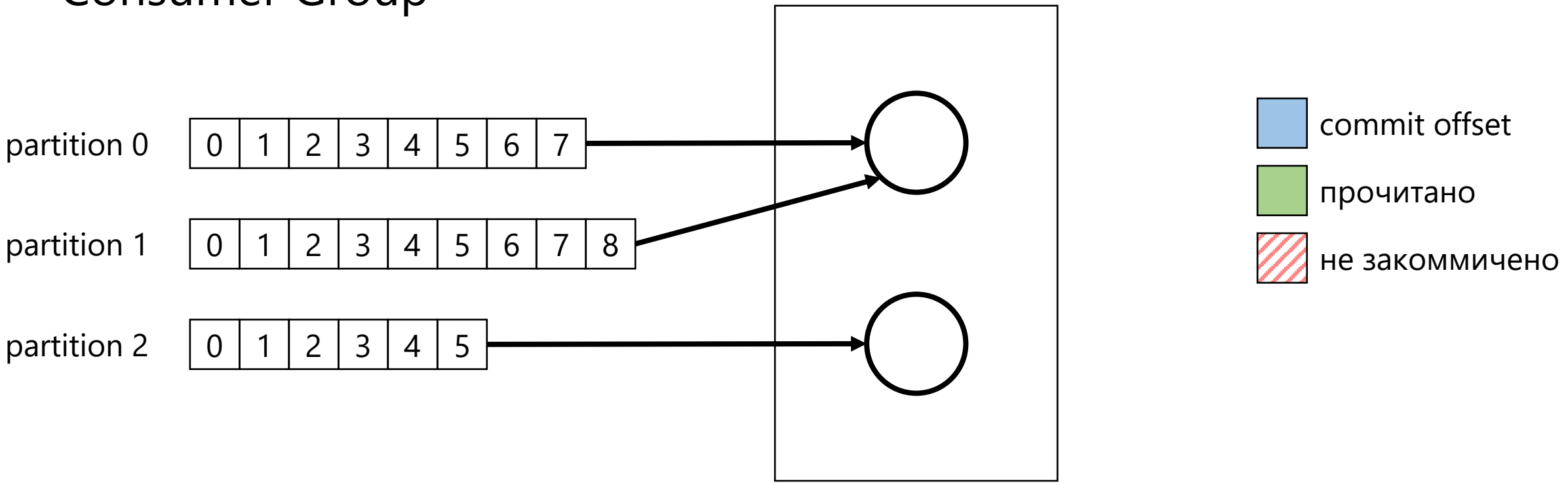
Kafka Consumer

Consumer Group



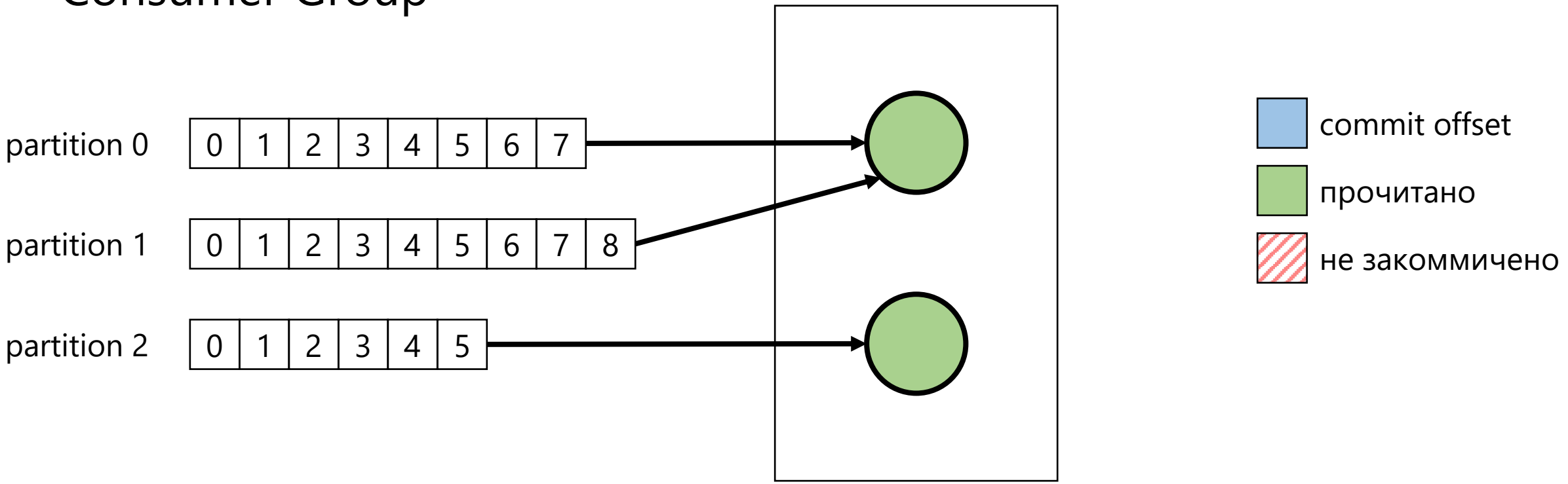
Kafka Consumer

Consumer Group



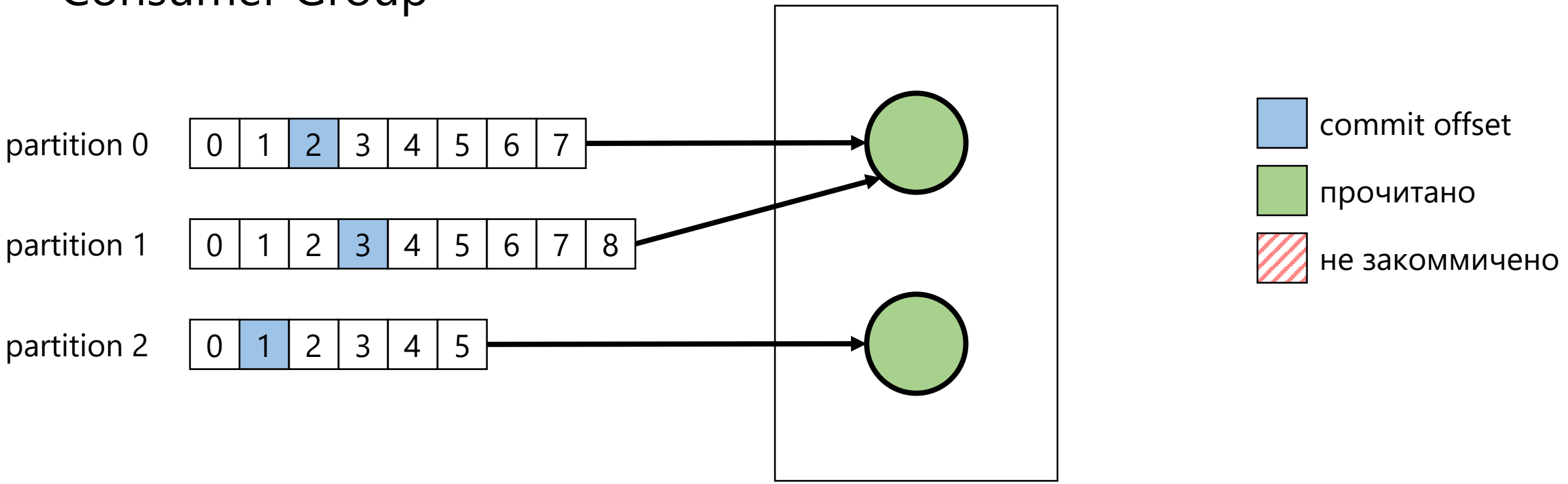
Kafka Consumer

Consumer Group



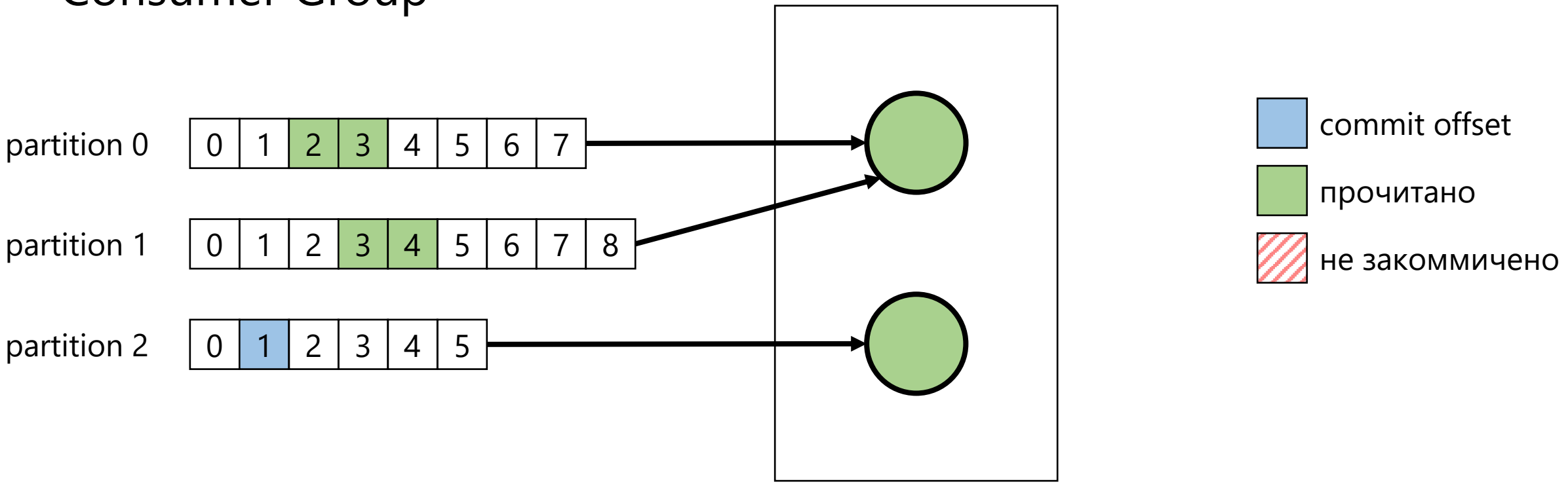
Kafka Consumer

Consumer Group



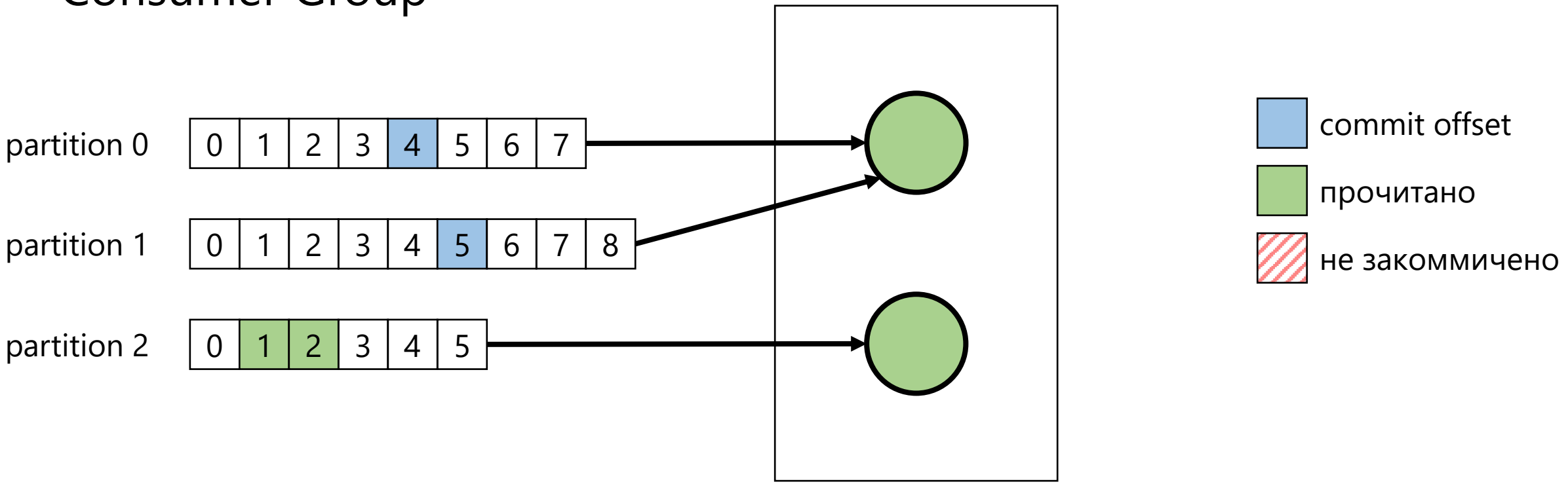
Kafka Consumer

Consumer Group



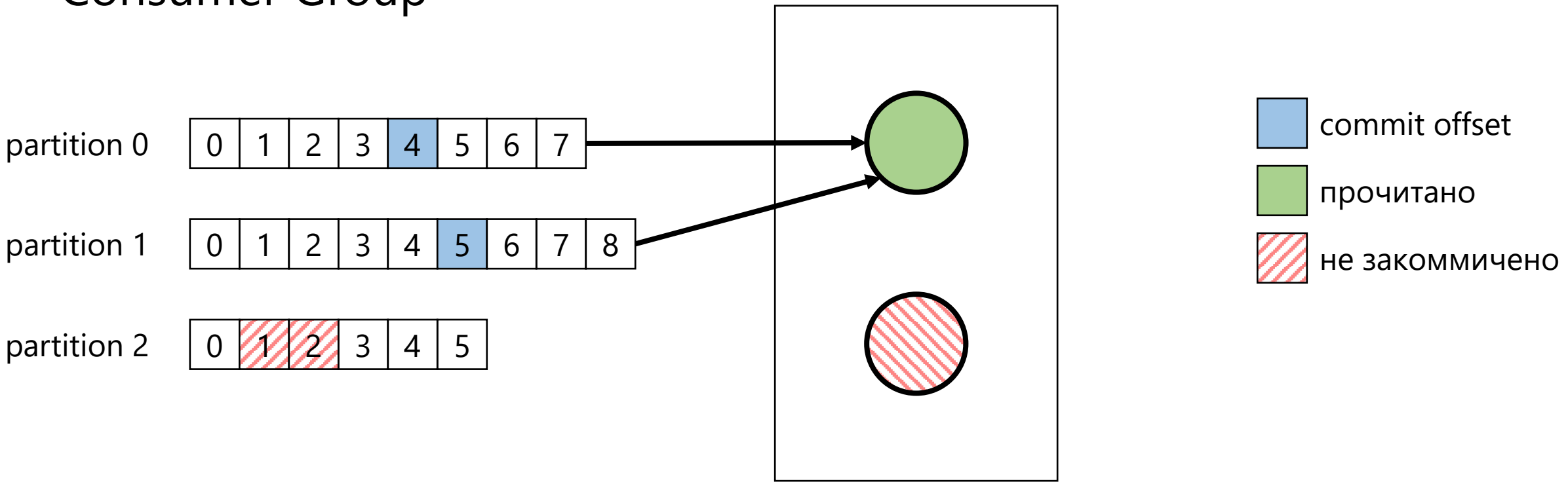
Kafka Consumer

Consumer Group



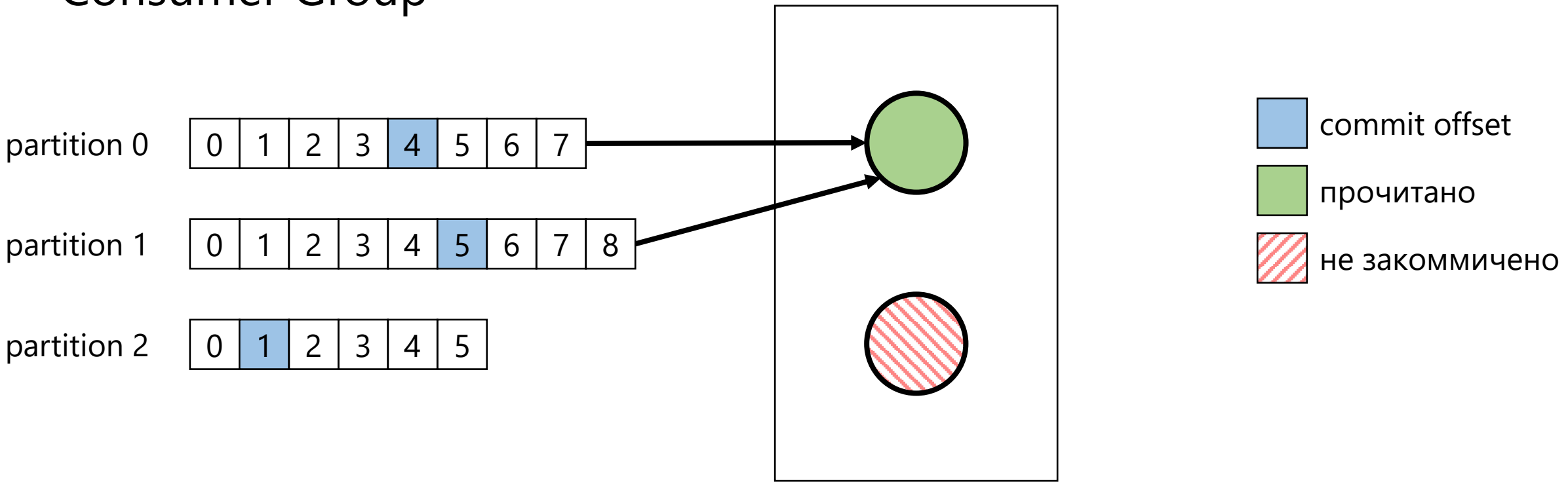
Kafka Consumer

Consumer Group



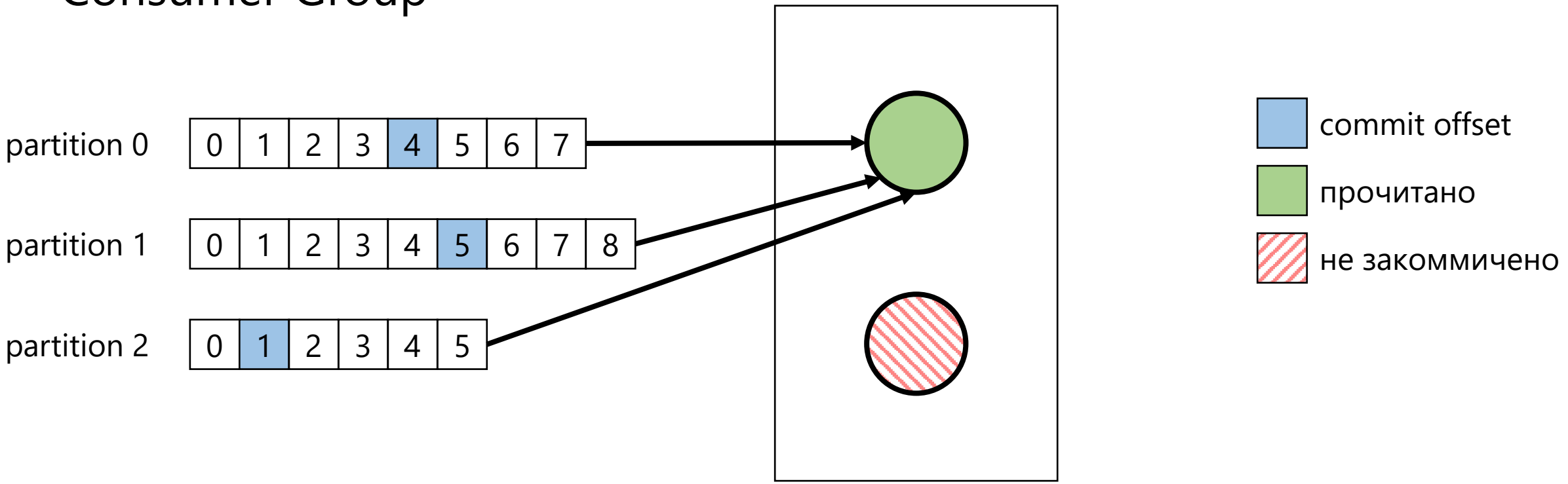
Kafka Consumer

Consumer Group



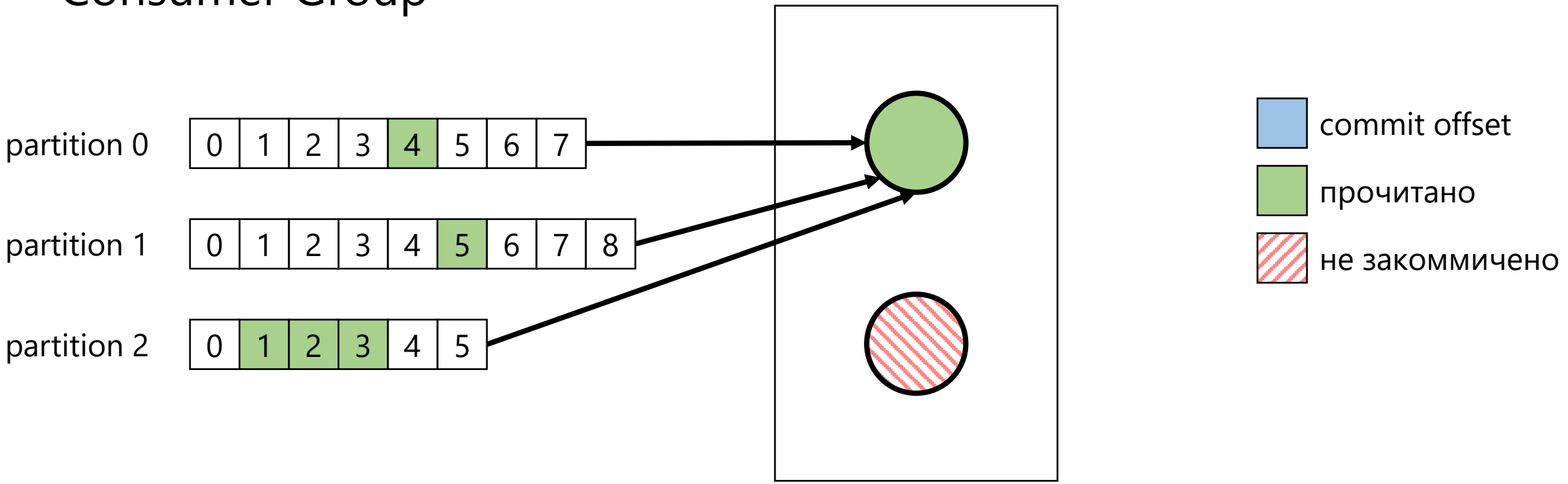
Kafka Consumer

Consumer Group



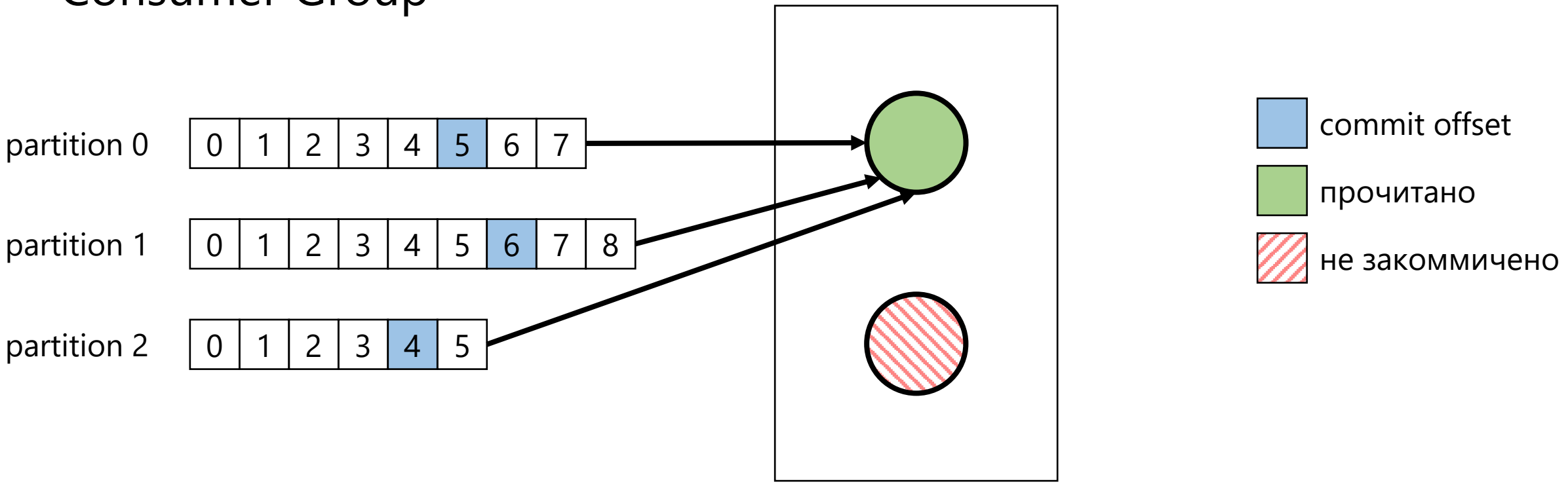
Kafka Consumer

Consumer Group



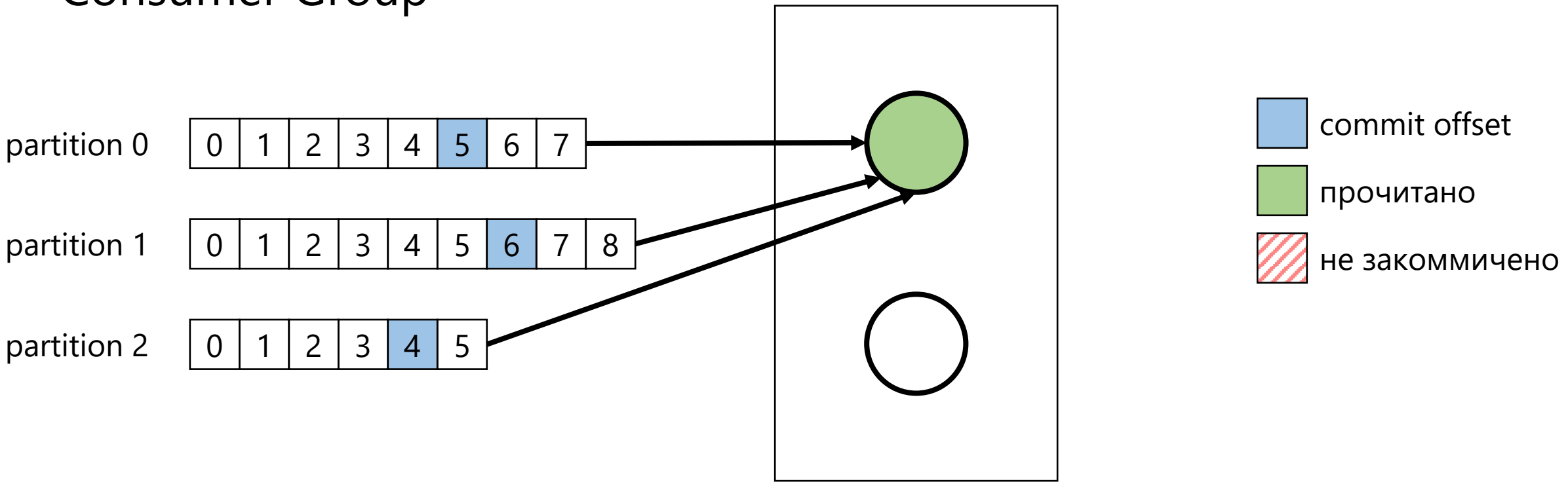
Kafka Consumer

Consumer Group



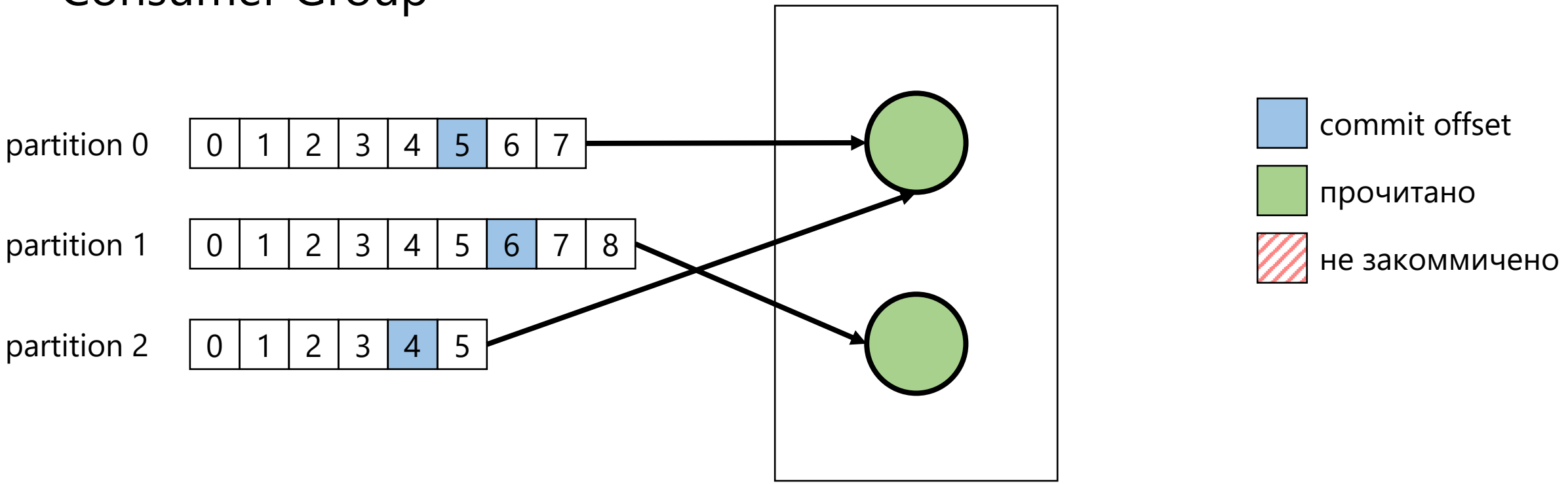
Kafka Consumer

Consumer Group



Kafka Consumer

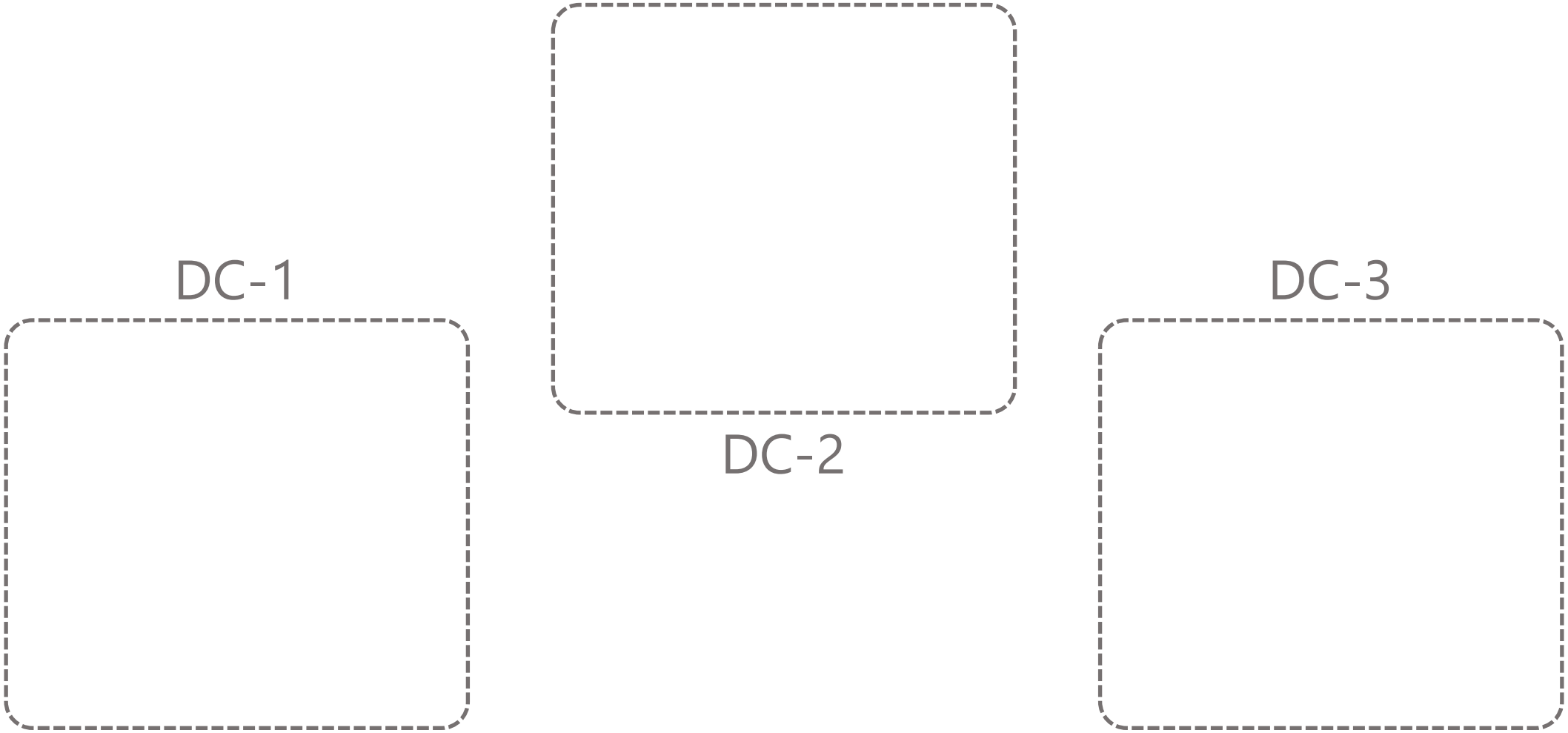
Consumer Group



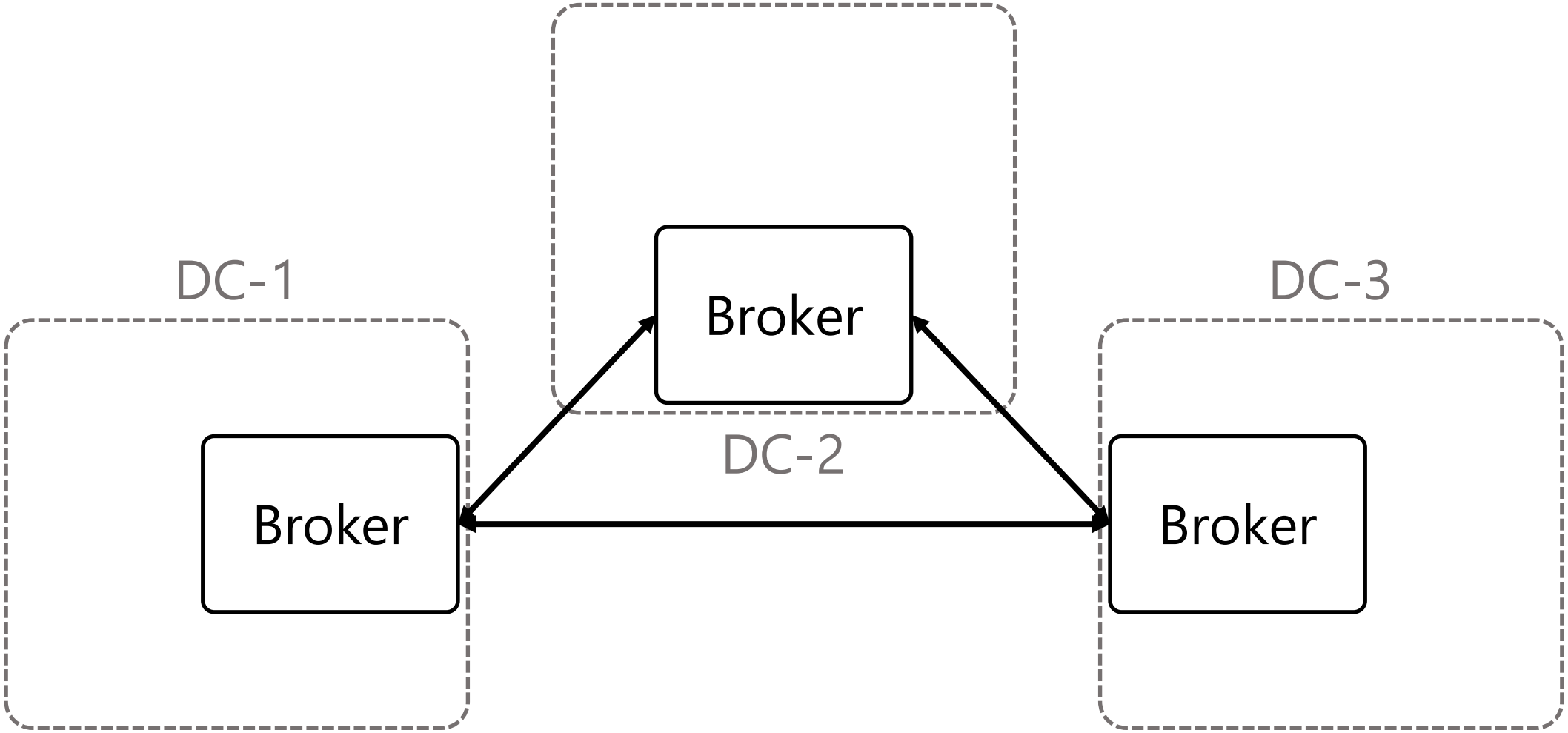
Kafka Consumer

```
conf = {
    # ...
    'group.id': groupId
}
```

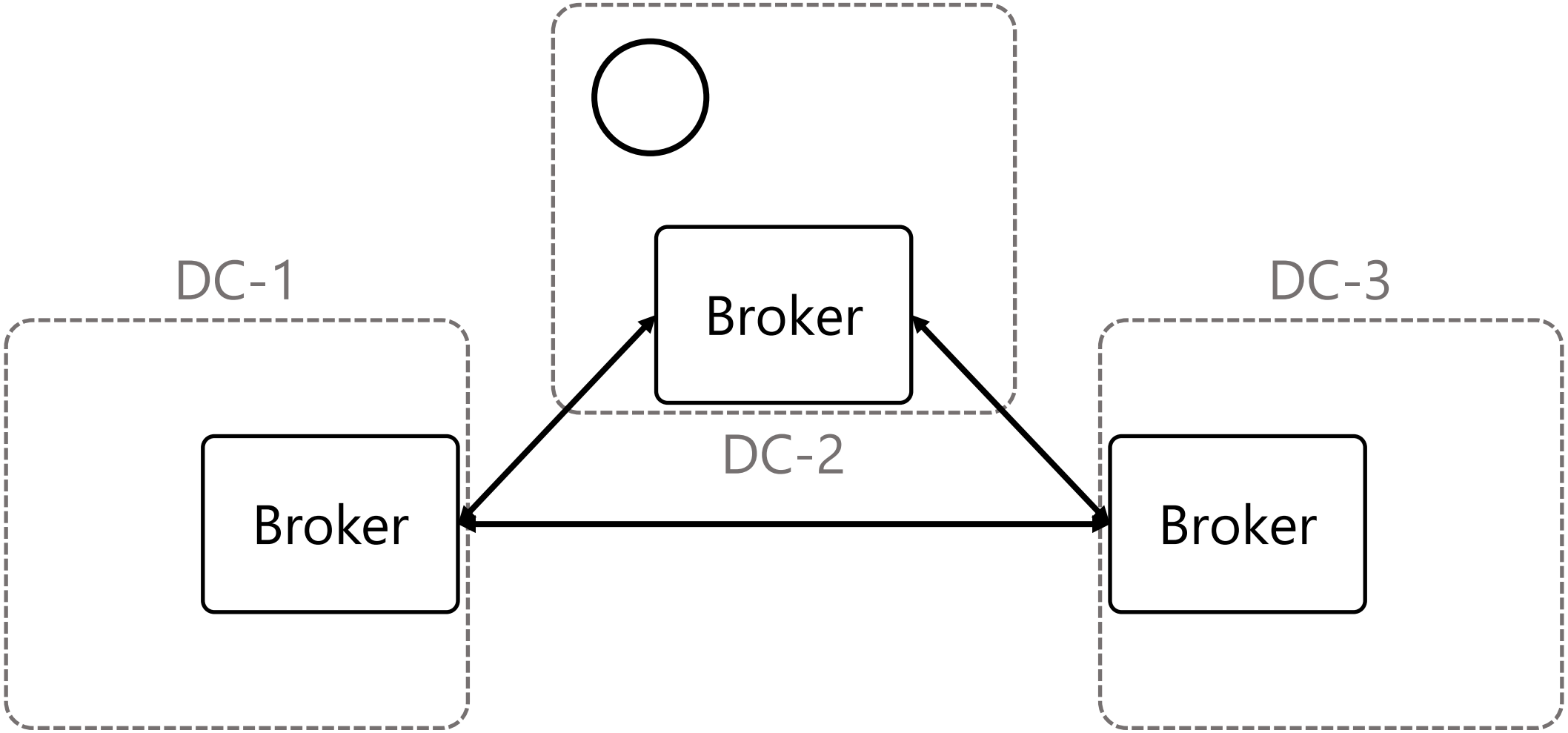
Kafka Consumer



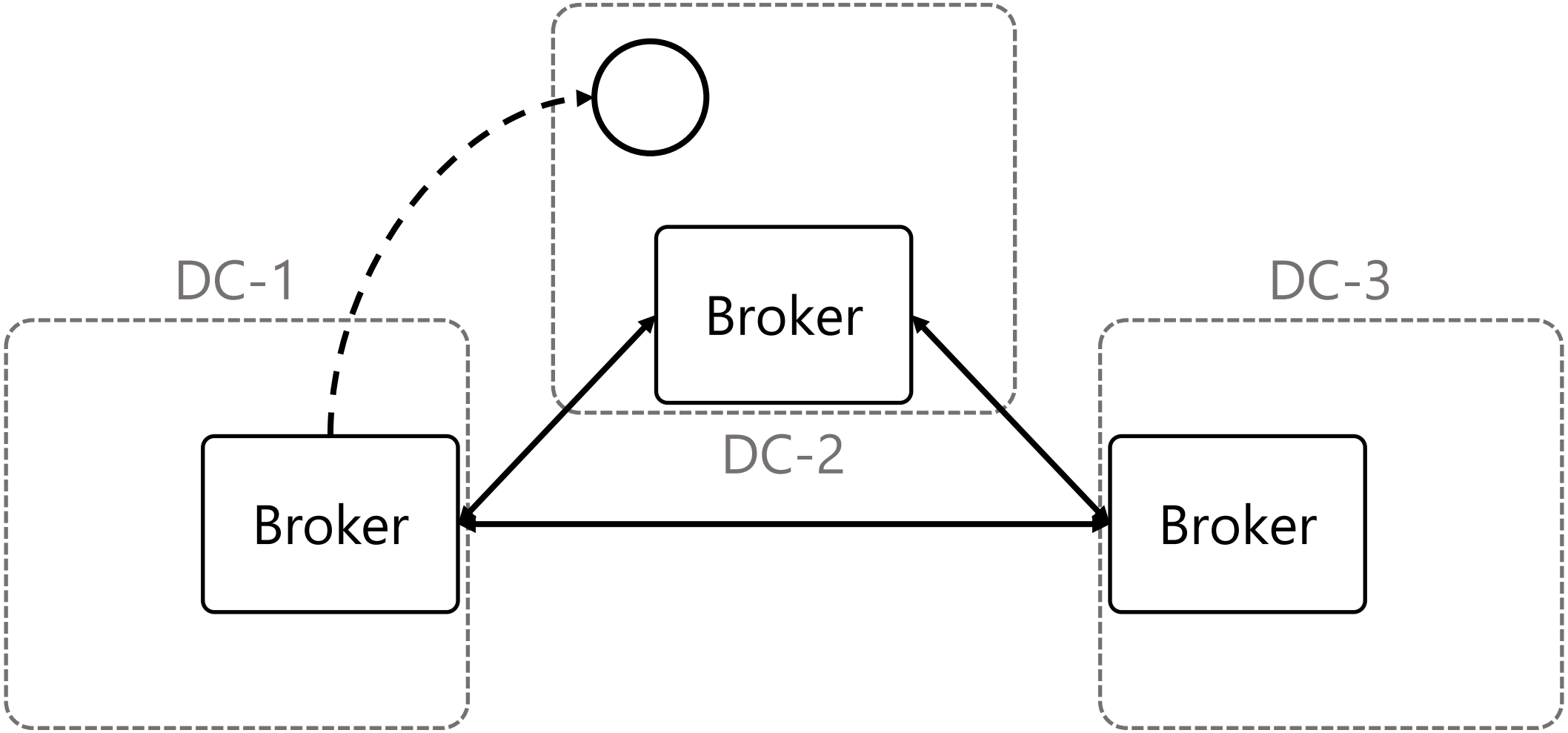
Kafka Consumer



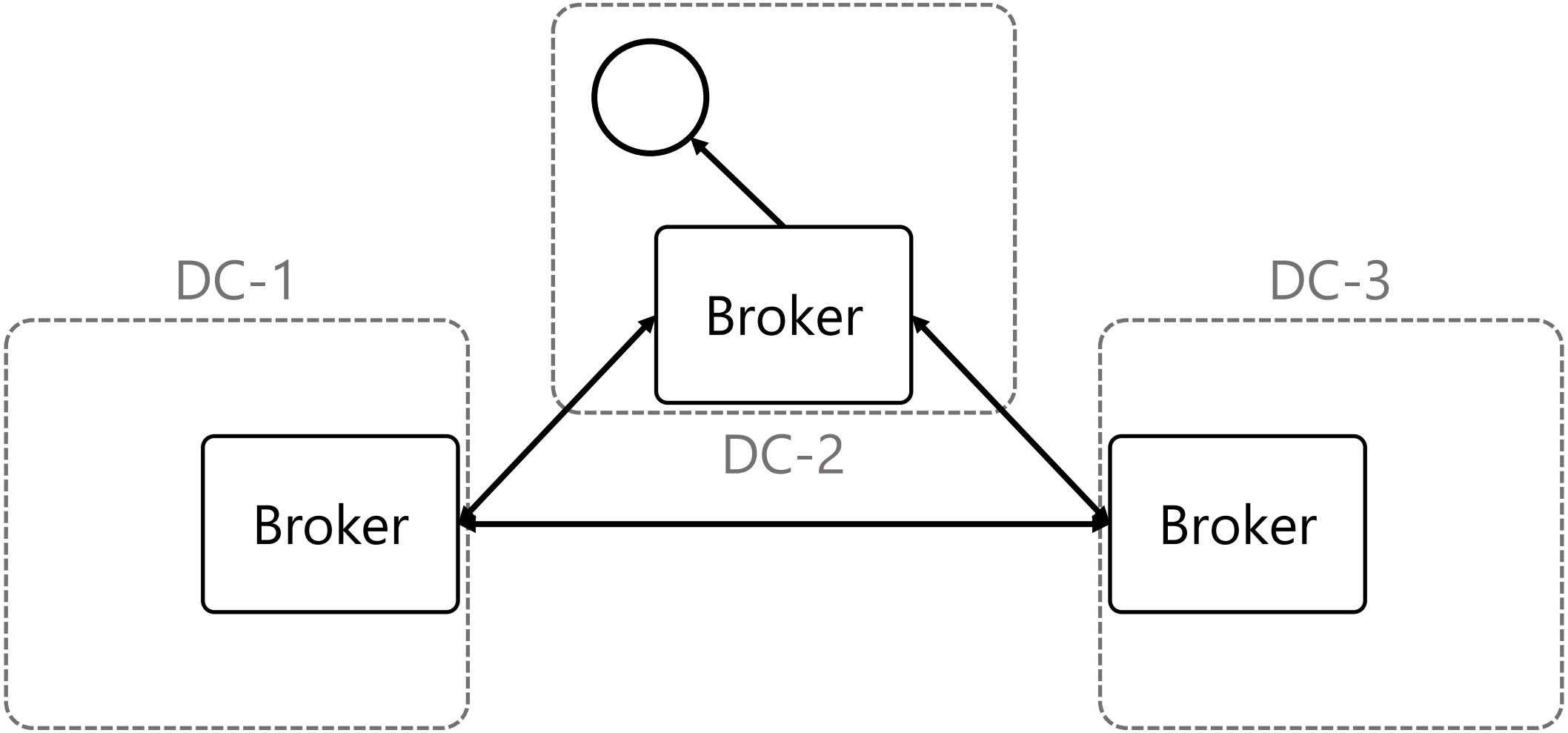
Kafka Consumer



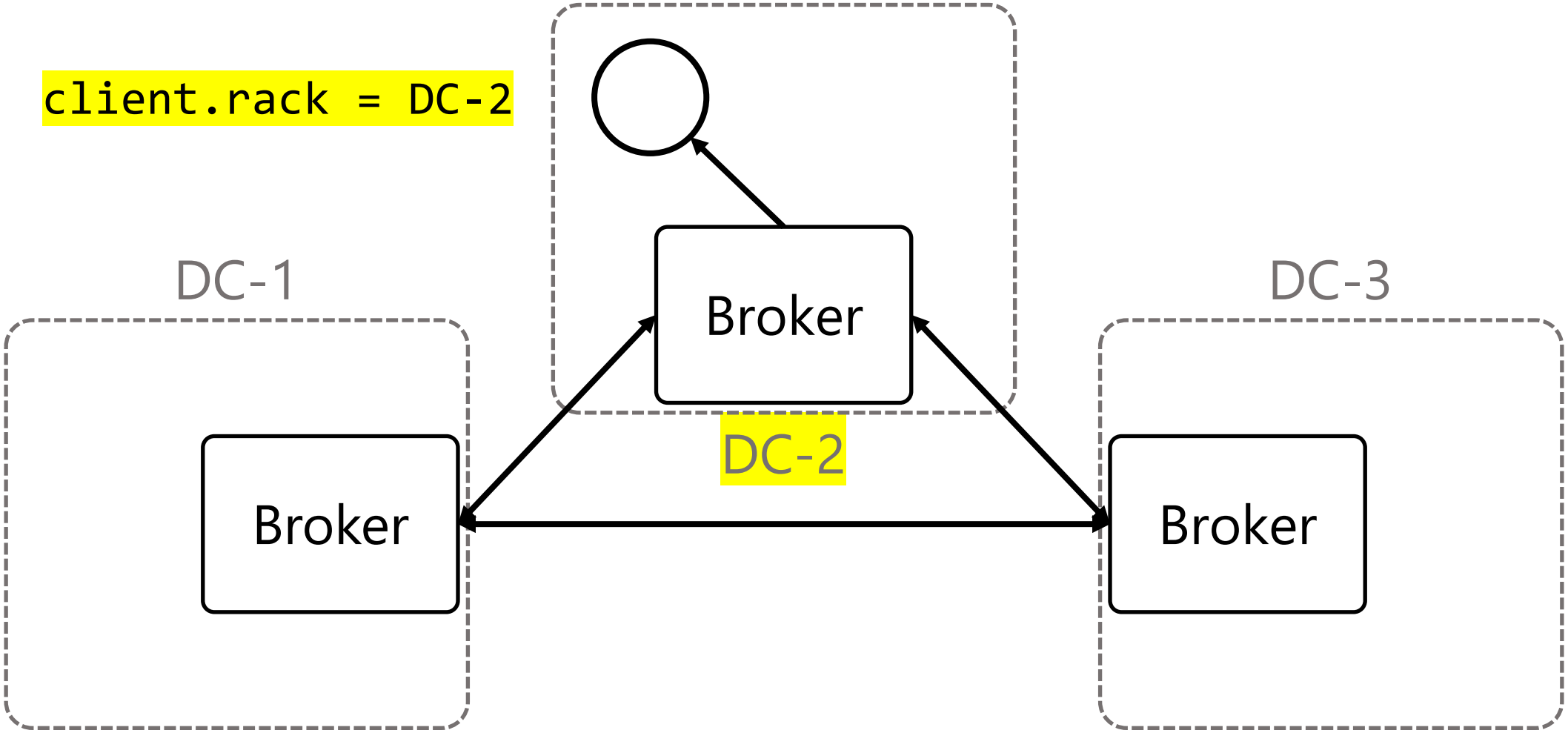
Kafka Consumer



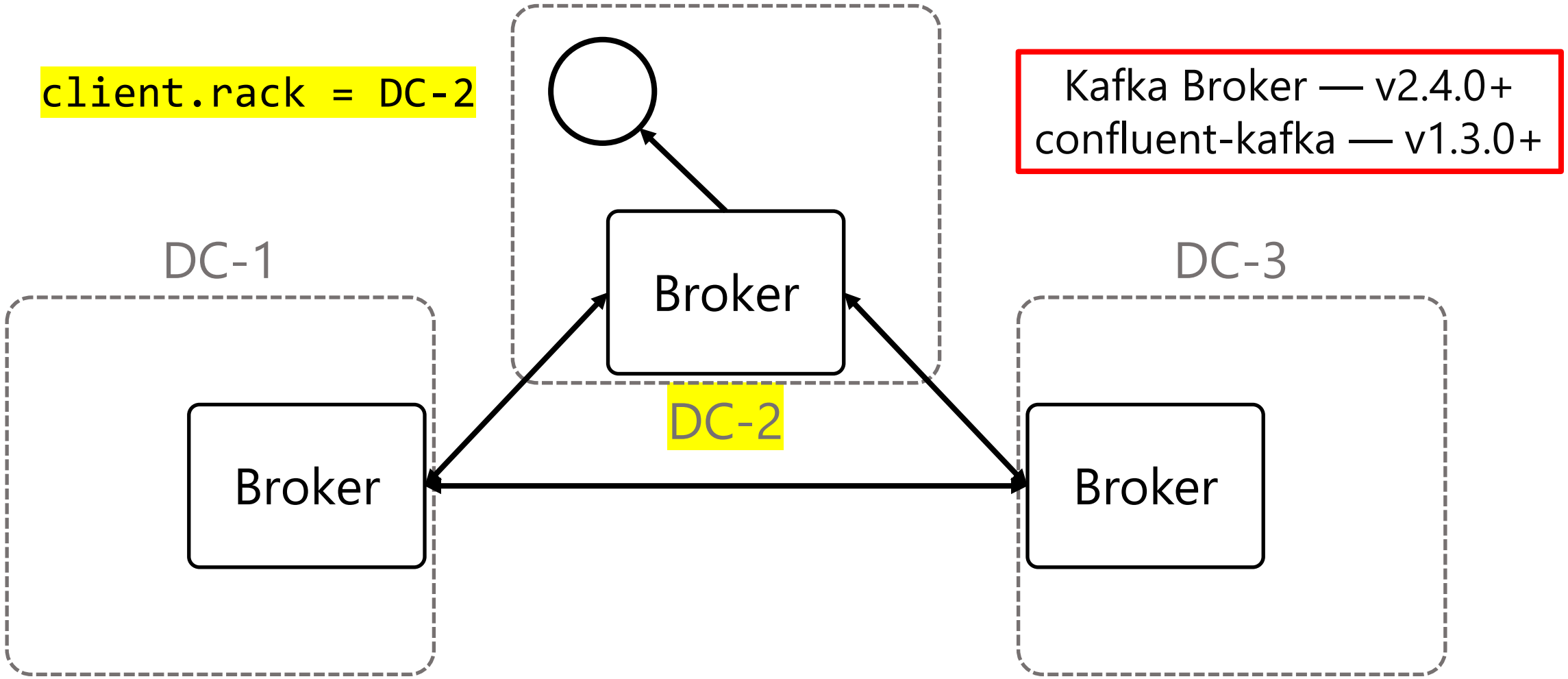
Kafka Consumer



Kafka Consumer



Kafka Consumer



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
— ML

Когда Apache Kafka 🔥

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

Чего нет в Kafka из коробки

Чего нет в Kafka из коробки

⚠️ Kafka — это не брокер сообщений!

Чего нет в Kafka из коробки

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

Q/A

Другие доклады и материалы:
`https://tg.me/chnl_GregoryKoshelev`

Какие есть альтернативы?

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