SolarLab>_

Kafka
подробности использования

Алексей Андреев https://t.me/AlekseyAndreev1984

Содержание

SolarLab>_

Введение в Apache Kafka

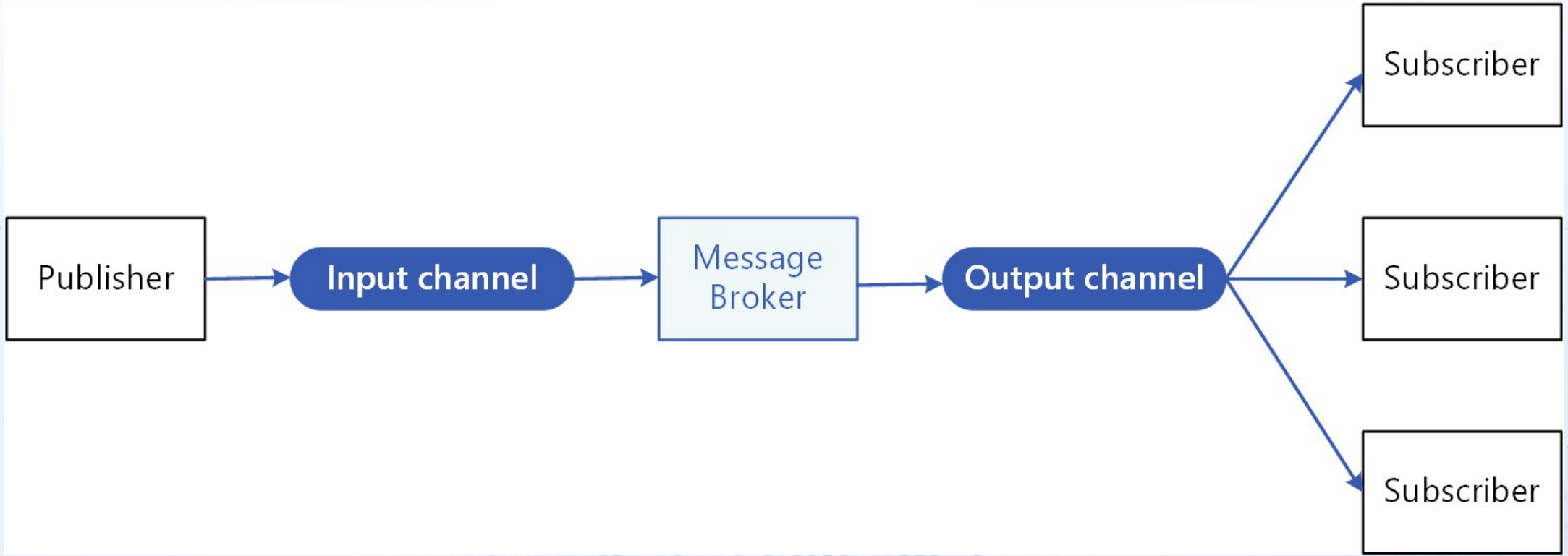
Введение в Apache Kafka

Apache Kafka - это распределенная платформа потоковой передачи событий с открытым исходным кодом, используемая тысячами компаний для высокопроизводительных конвейеров передачи данных, потоковой аналитики, интеграции данных и критически важных приложений

https://kafka.apache.org/

Введение в Apache Kafka

Pub-Sub с pool-механикой чтения



Введение в Apache Kafka

Pub-Sub с pool-механикой чтения

pull-модель — консьюмеры сами отправляют запрос раз в n секунд на сервер для получения новой порции сообщений. При таком подходе клиенты могут эффективно контролировать собственную нагрузку. Кроме того, pull-модель позволяет группировать сообщения в батчи, таким образом достигая лучшей пропускной способности. К минусам модели можно отнести потенциальную разбалансированность нагрузки между разными консьюмерами, а также более высокую задержку обработки данных. К плюсам – это увеличение пропускной способности

push-модель — сервер делает запрос к клиенту, посылая ему новую порцию данных. По такой модели, например, работает RabbitMQ. Она снижает задержку обработки сообщений и позволяет эффективно балансировать распределение сообщений по консьюмерам.

Введение в Apache Kafka

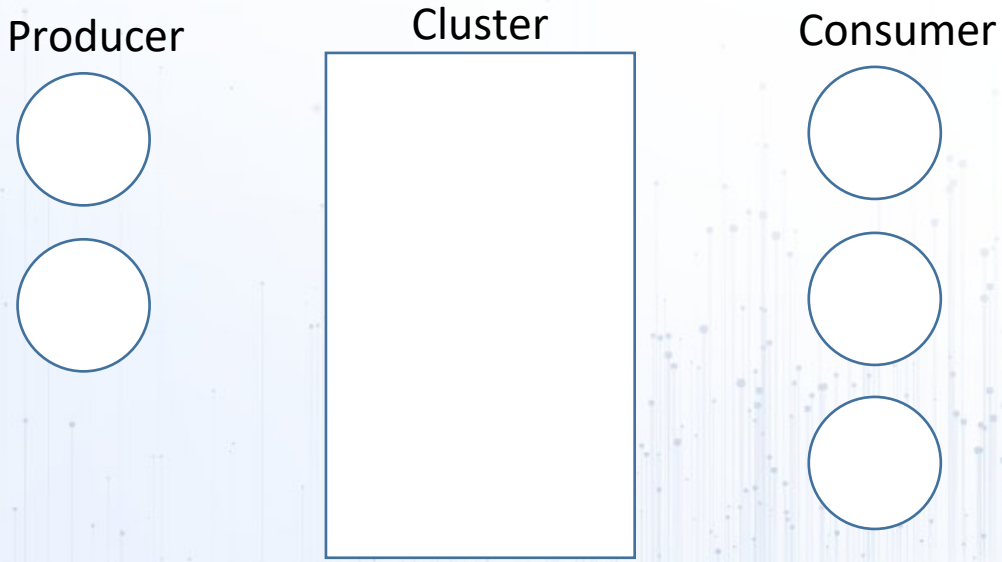
Пожалуй, фундаментальное отличие Kafka состоит в том, как сообщения хранятся на брокере и как потребляются консьюмерами.

Сообщения в Kafka **не удаляются** брокерами по мере их обработки консьюмерами — данные в Kafka могут храниться днями, неделями, годами.

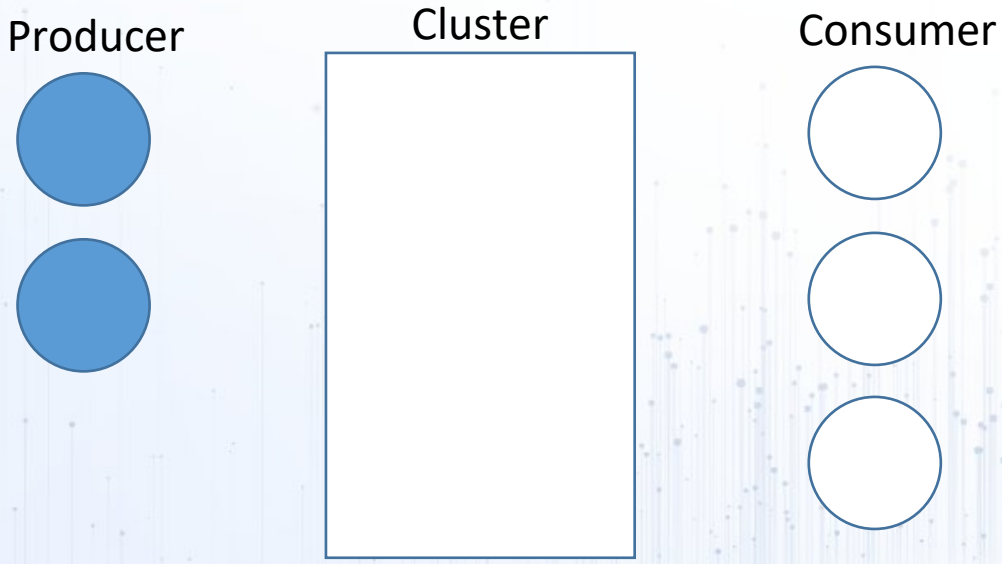
Благодаря этому одно и то же сообщение может быть обработано **сколько угодно раз** разными консьюмерами и в разных контекстах.

Введение в Apache Kafka

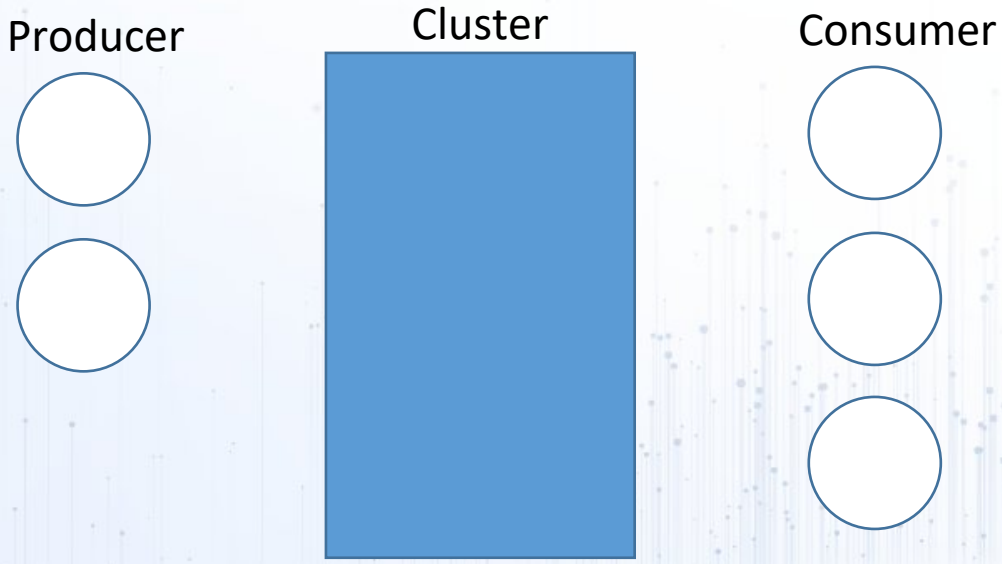
Общий вид Cluster, Producer, Consumer



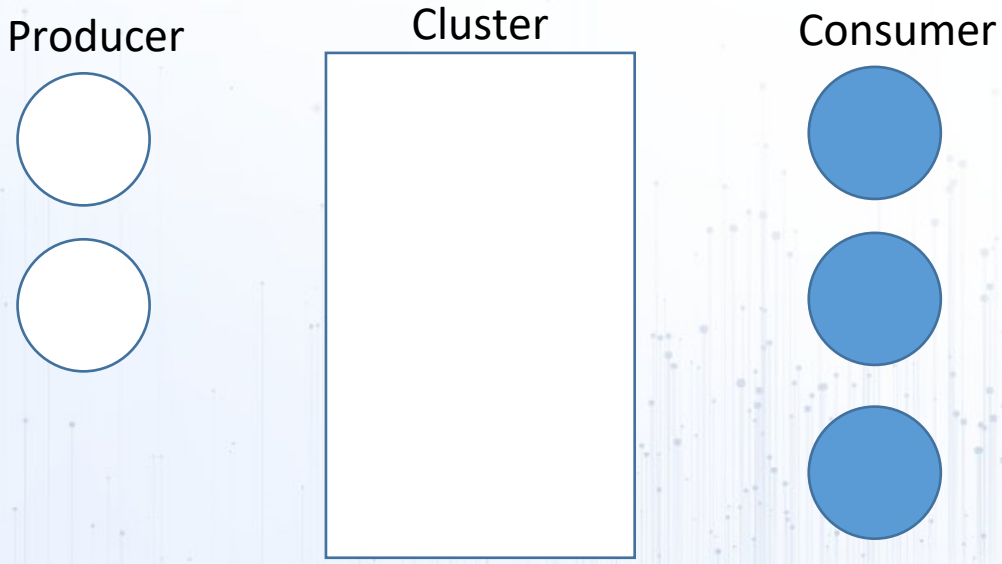
Введение в Apache Kafka



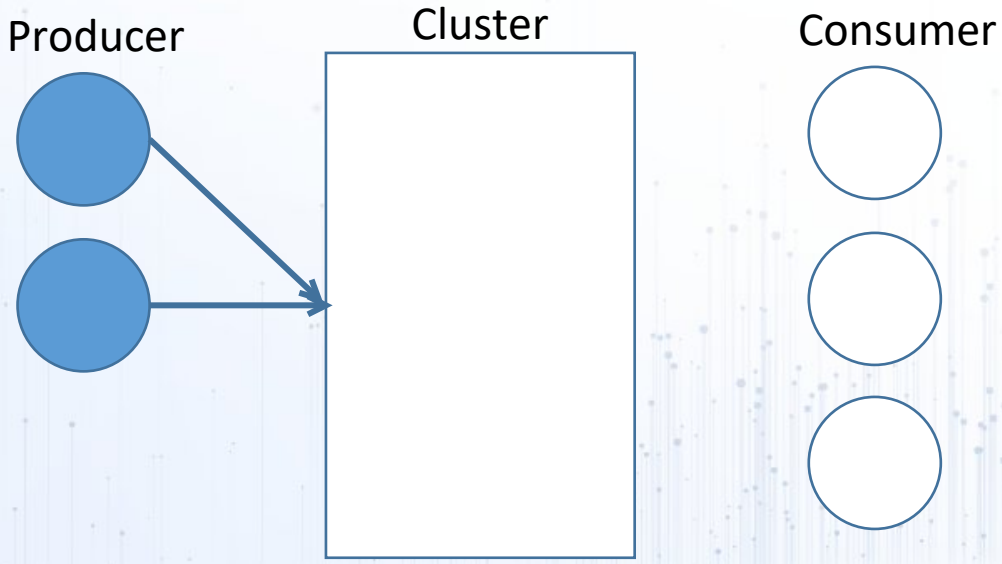
Введение в Apache Kafka



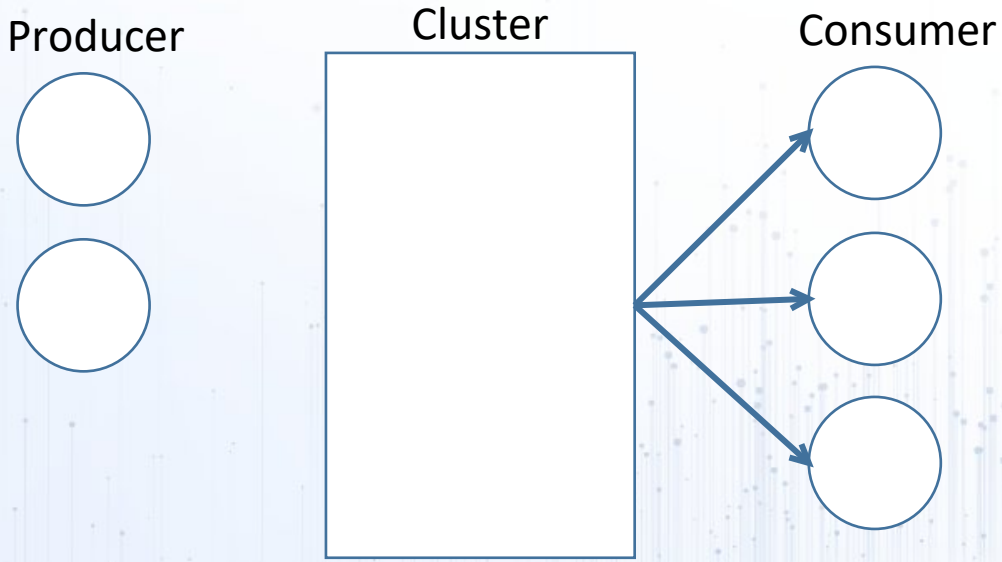
Введение в Apache Kafka



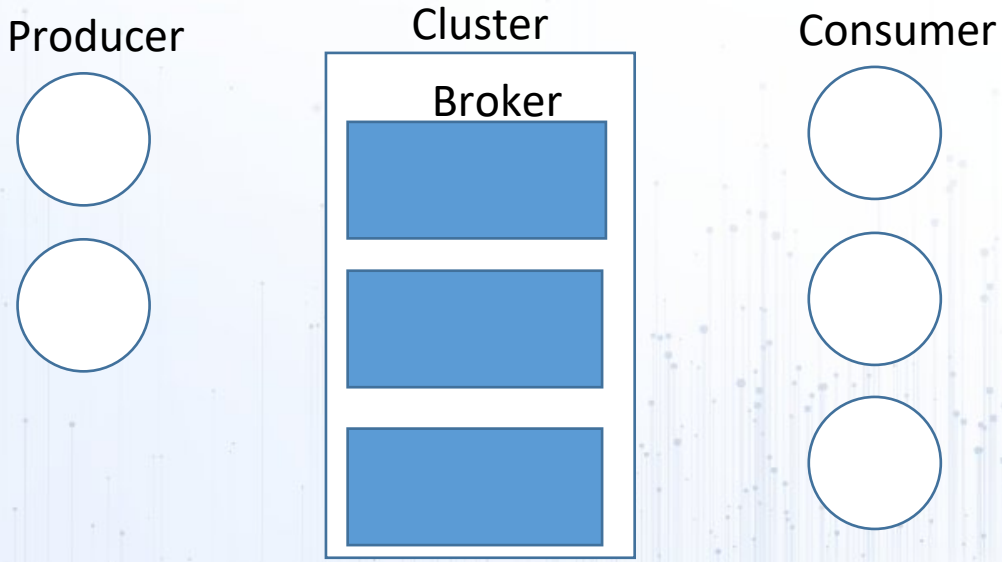
Введение в Apache Kafka



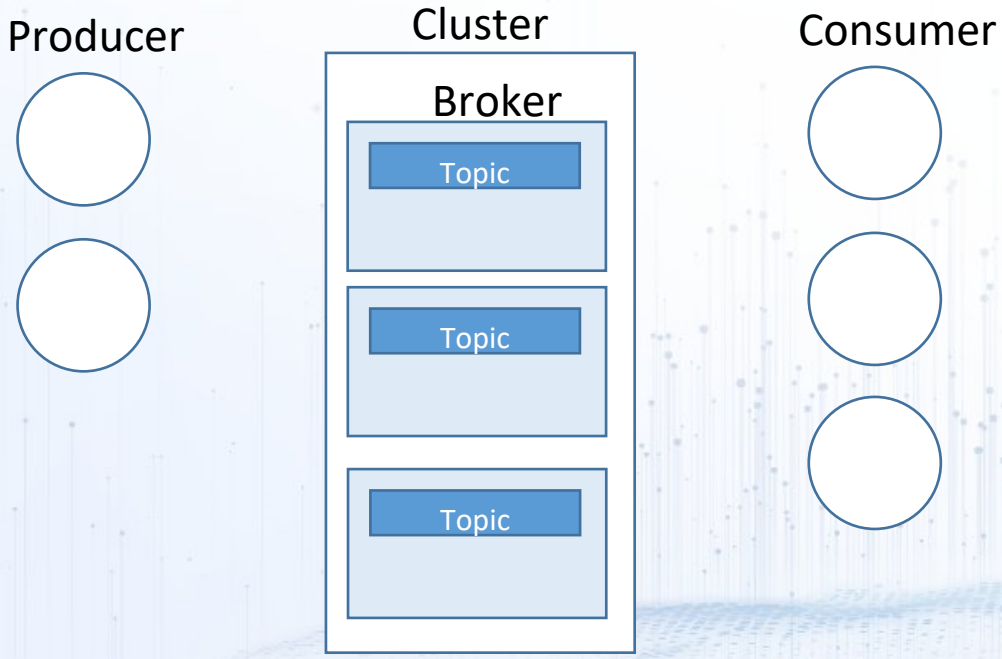
Введение в Apache Kafka



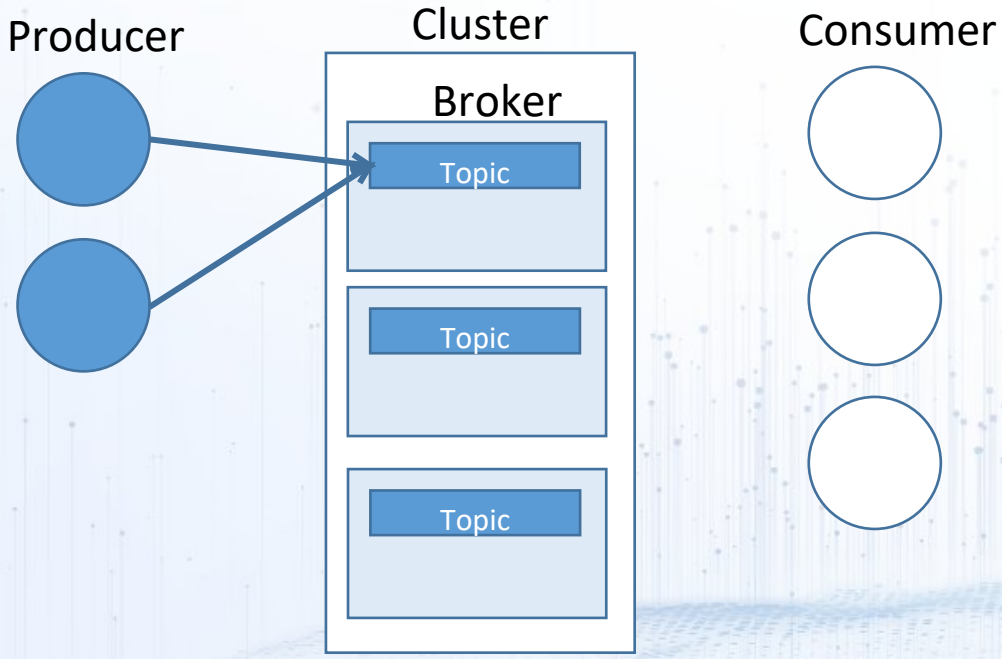
Введение в Apache Kafka



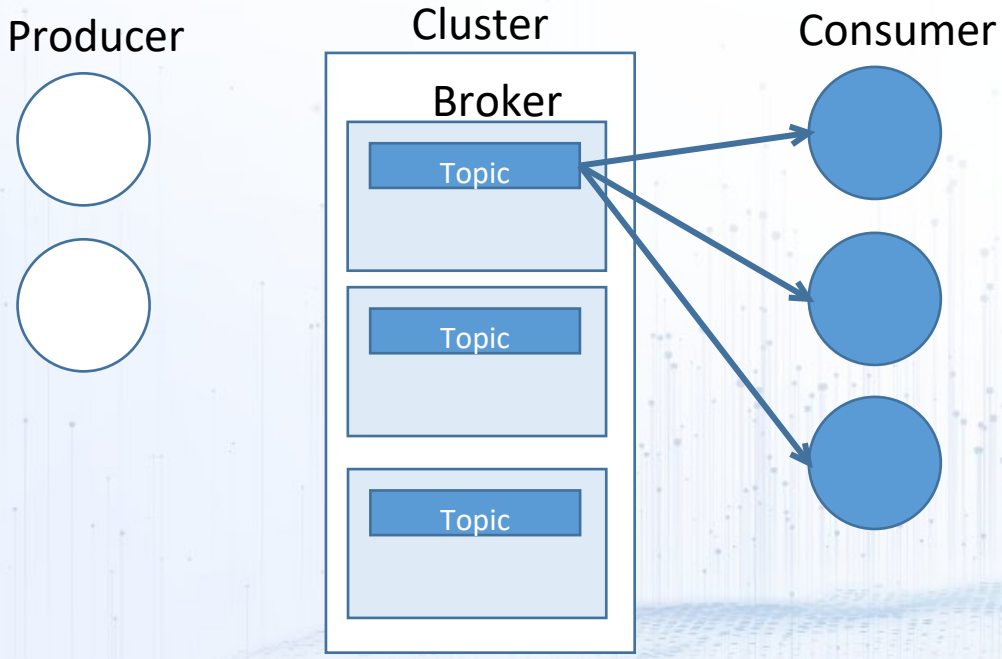
Введение в Apache Kafka



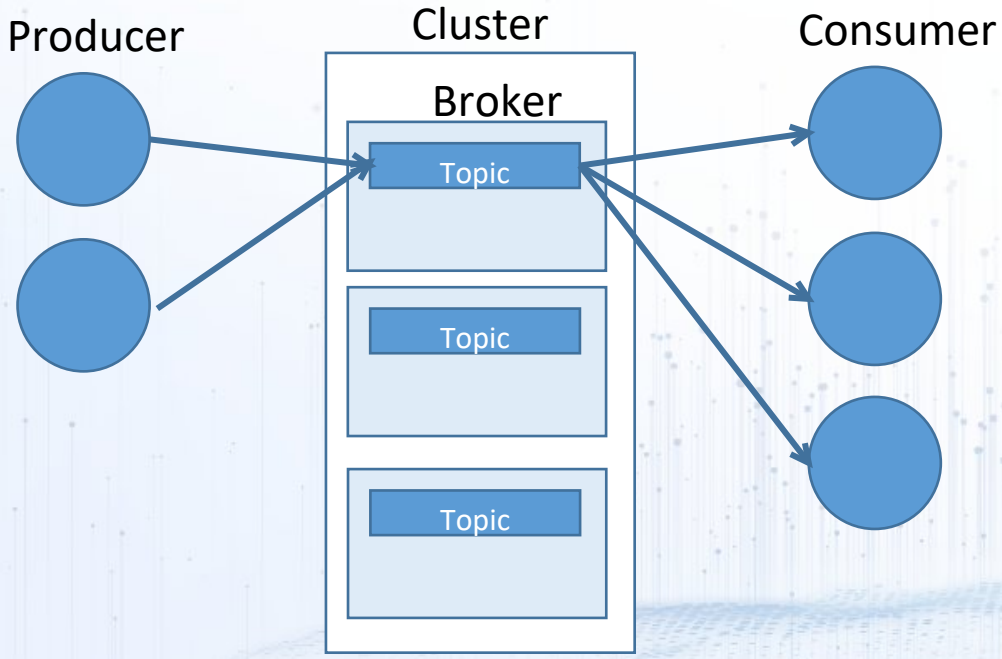
Введение в Apache Kafka



Введение в Apache Kafka



Введение в Apache Kafka



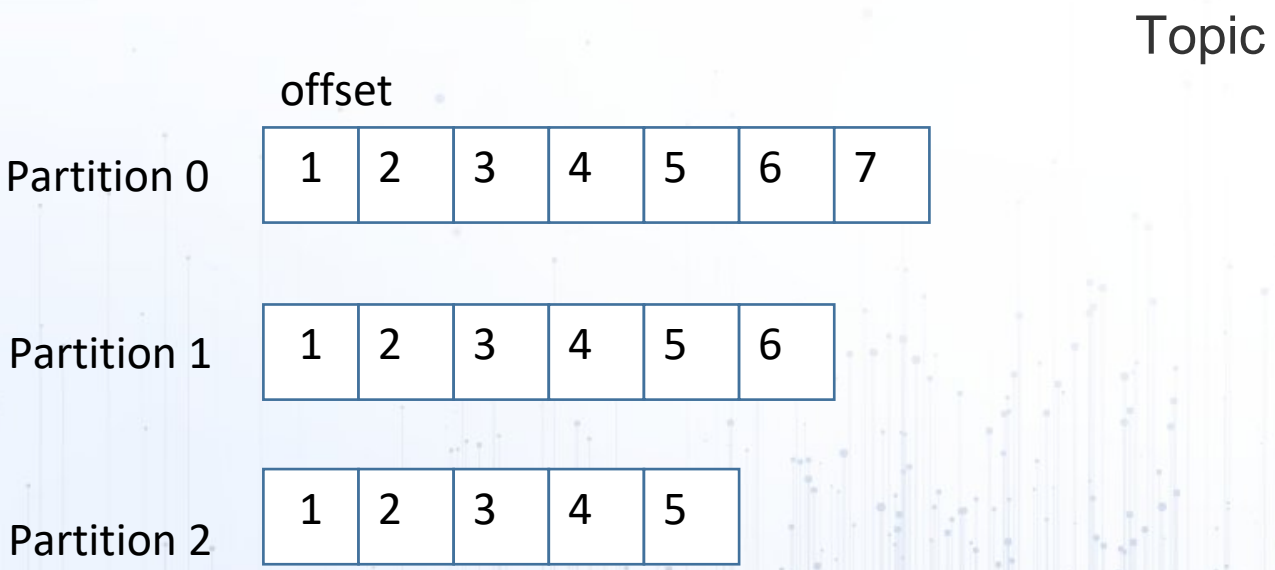
Введение в Apache Kafka

- Topic
- Broker
- Producer
- Consumer

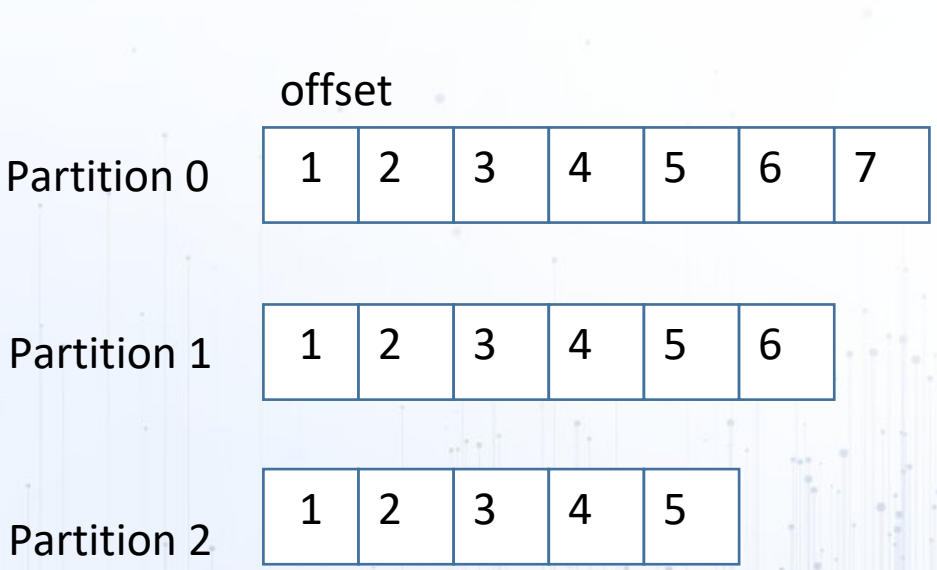
SolarLab>_

Архитектура Apache Kafka

Архитектура Apache Kafka

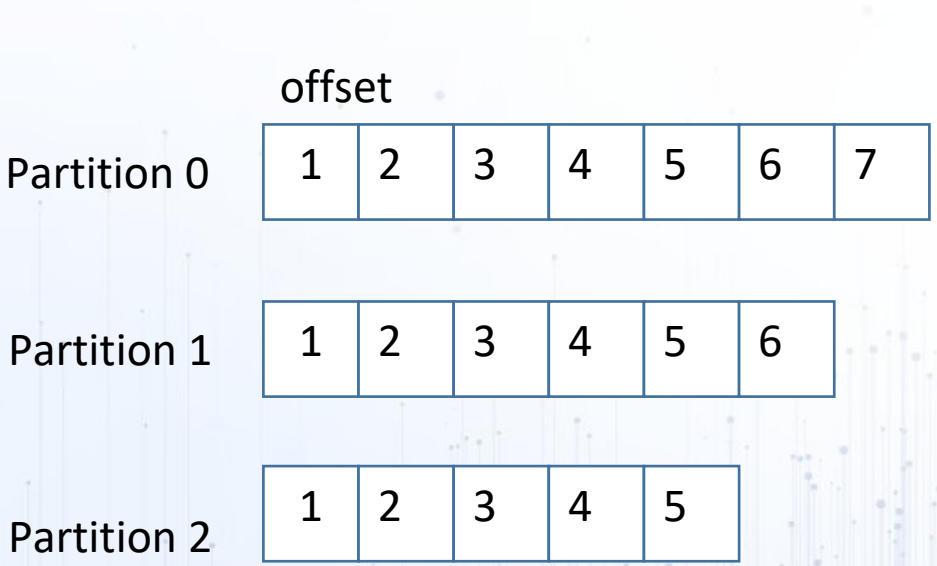


Архитектура Apache Kafka



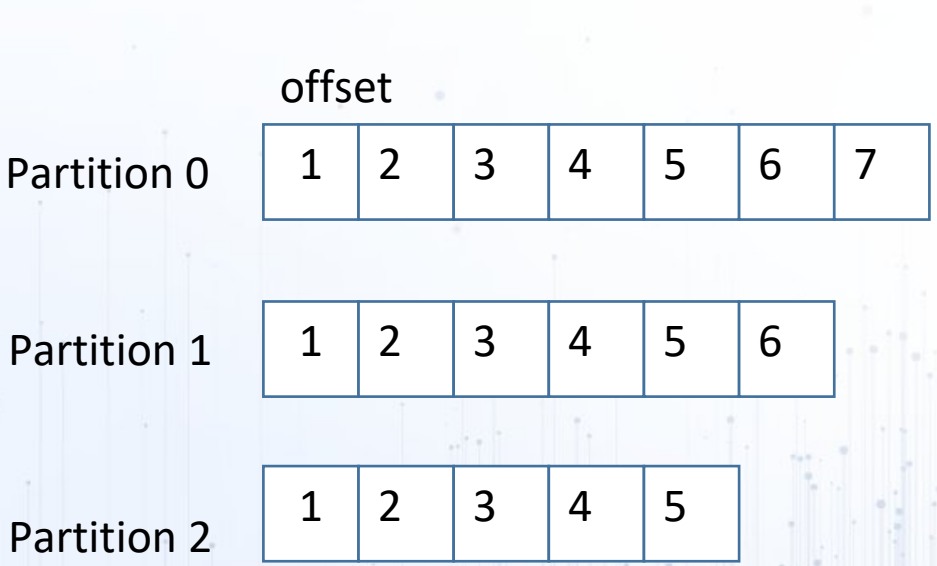
Партиция хранится на диске

Архитектура Apache Kafka



Писать и читать можно в конкретную партицию для топика

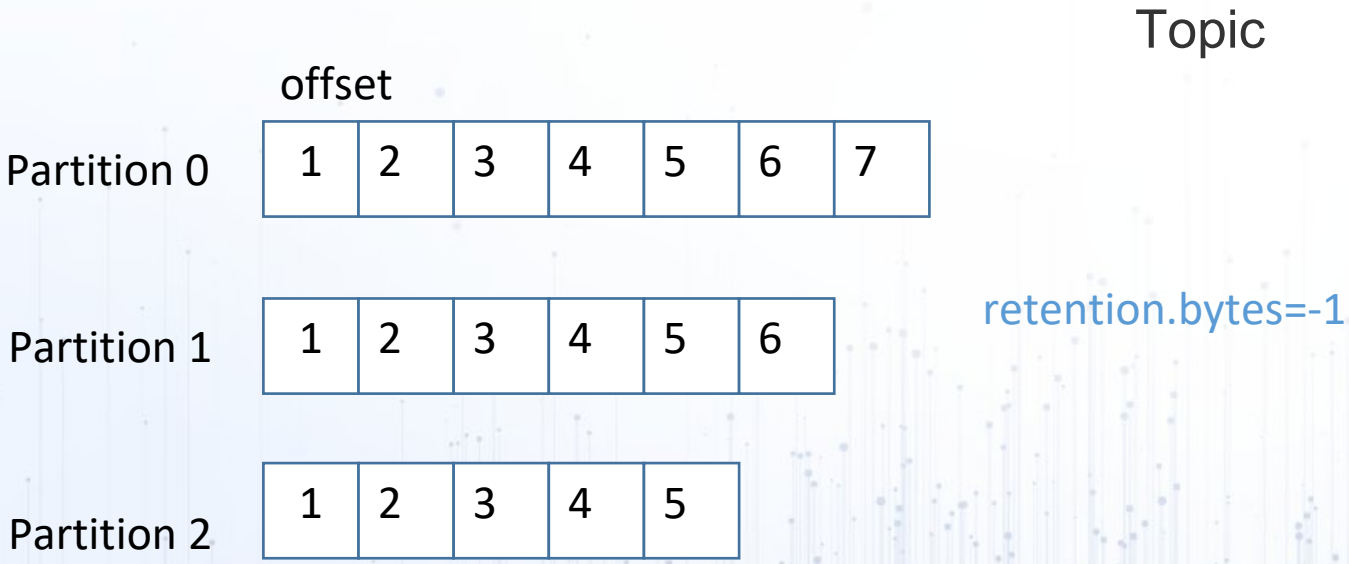
Архитектура Apache Kafka



Или писать в топик, а кафка сама распределит в какую партицию ей поместить данные

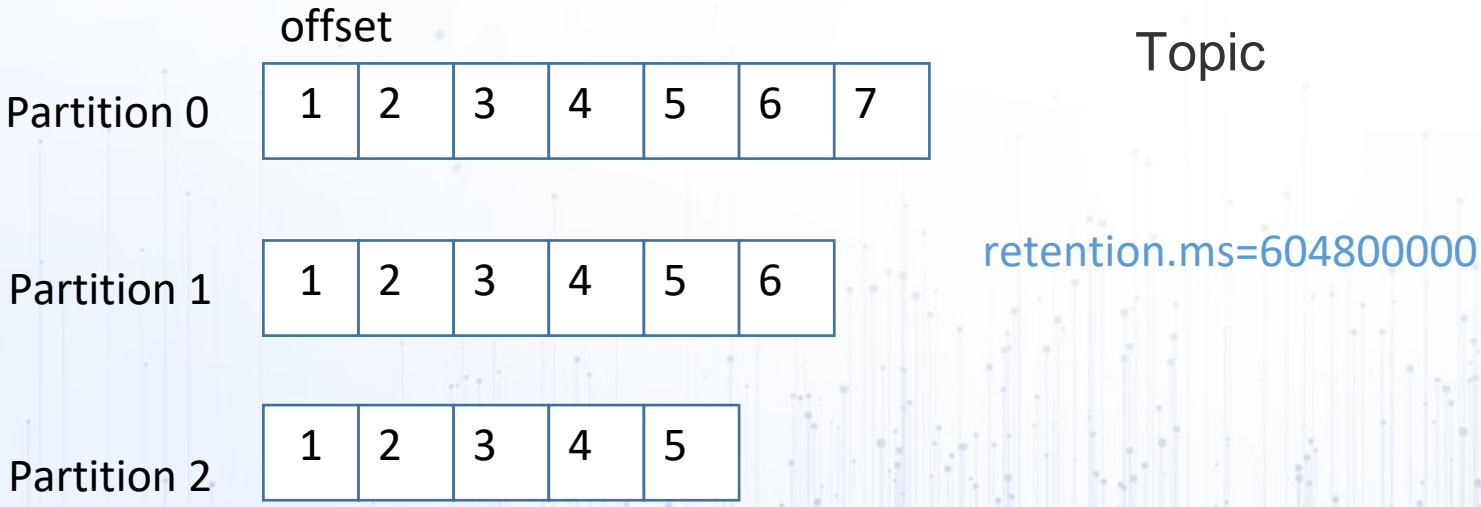
Архитектура Apache Kafka

Время хранения сегмента в партиции в байтах(если превышает, то удаляем)



Архитектура Apache Kafka

Время хранения сегмента в партиции в миллисекундах(если превышает, то удаляем)



Topic

retention.ms=604800000

Архитектура Apache Kafka

Для каждого кластера у нас есть множество брокеров

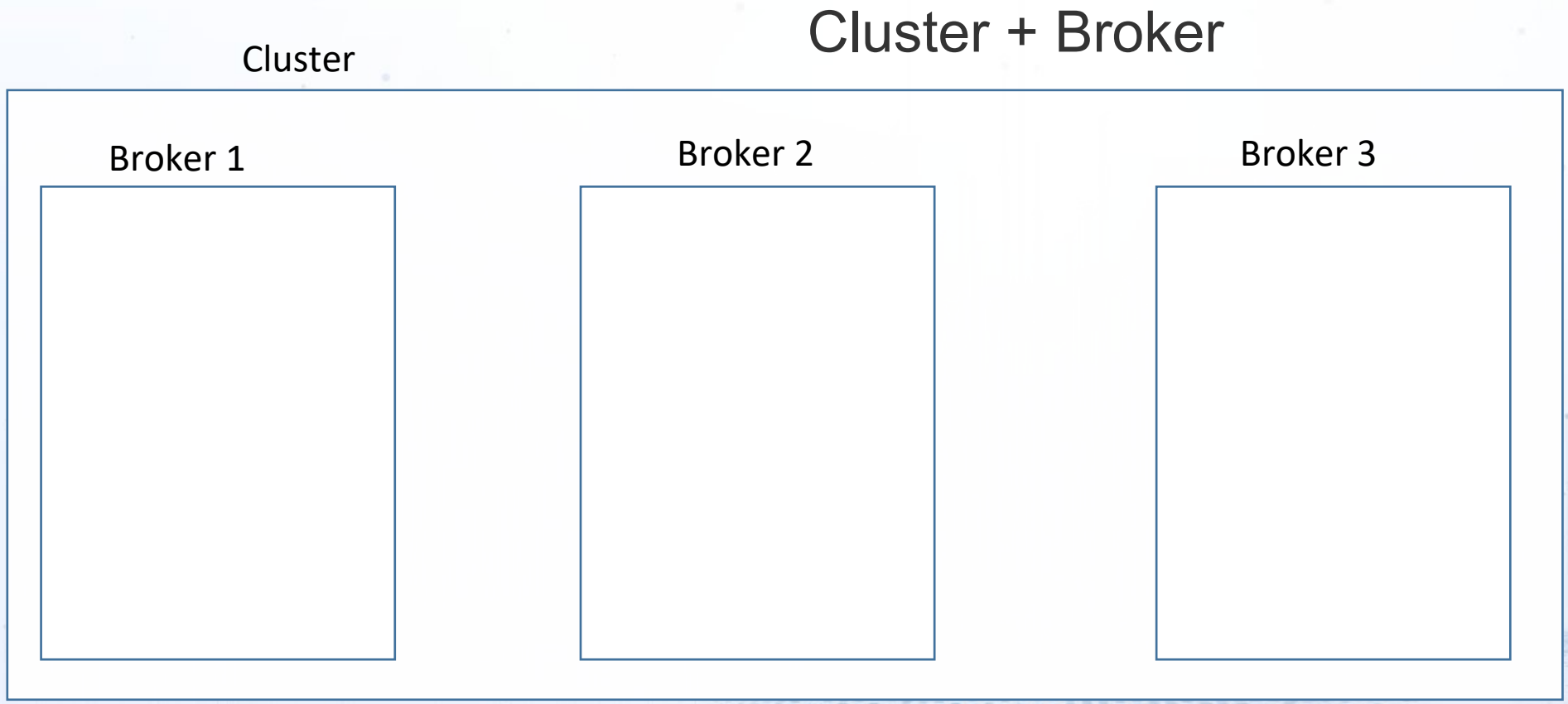
Для каждого брокера у нас есть множество топиков

Для каждого топика у нас есть множество партиций

Для каждой партиции сообщение хранится в конкретной оффсете

Таким образом конкретное сообщение определяется связкой Топик, Партиция, Оффсет

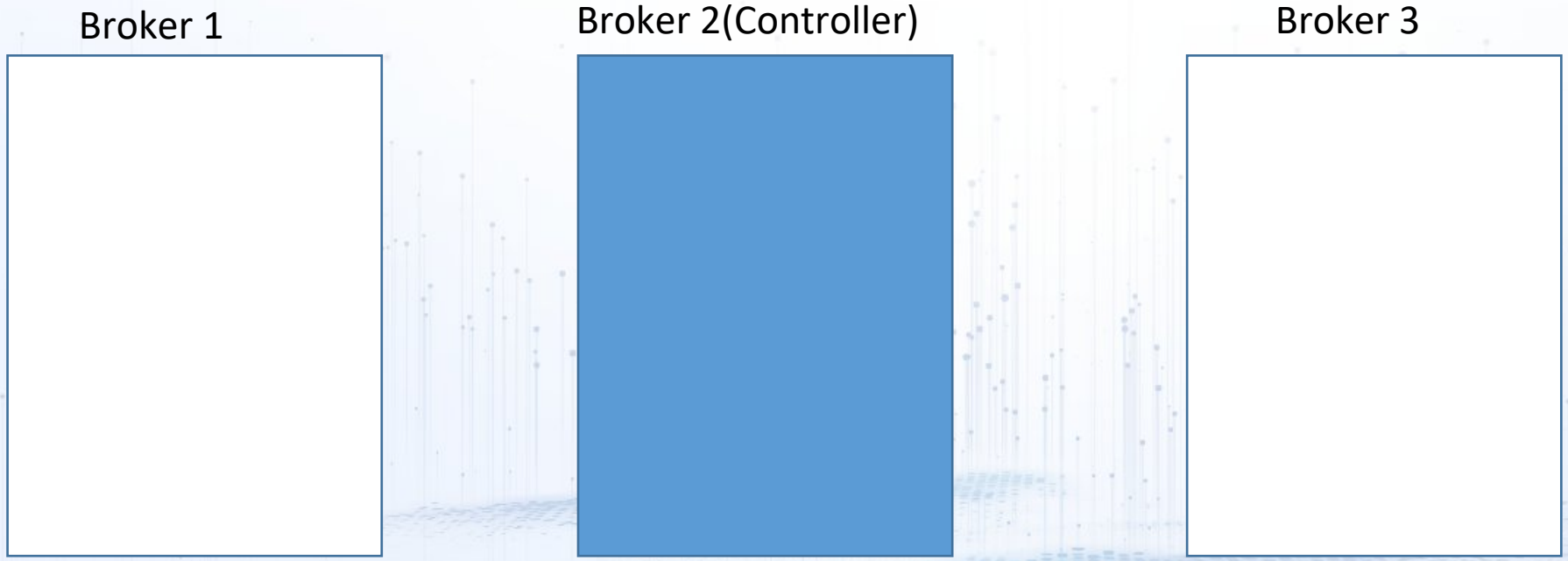
Архитектура Apache Kafka



Архитектура Apache Kafka

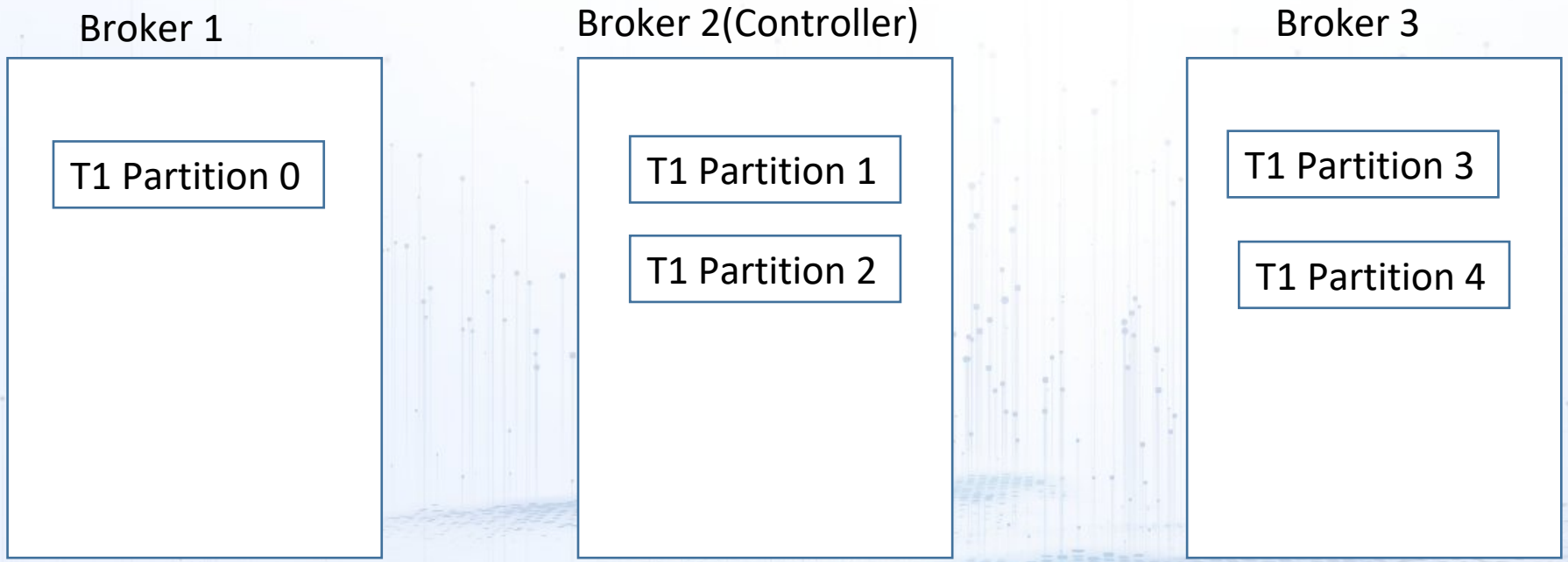
Broker

Контроллер – координирует работу брокеров. Отвечает за работу кластера. Кто лидер и прочее



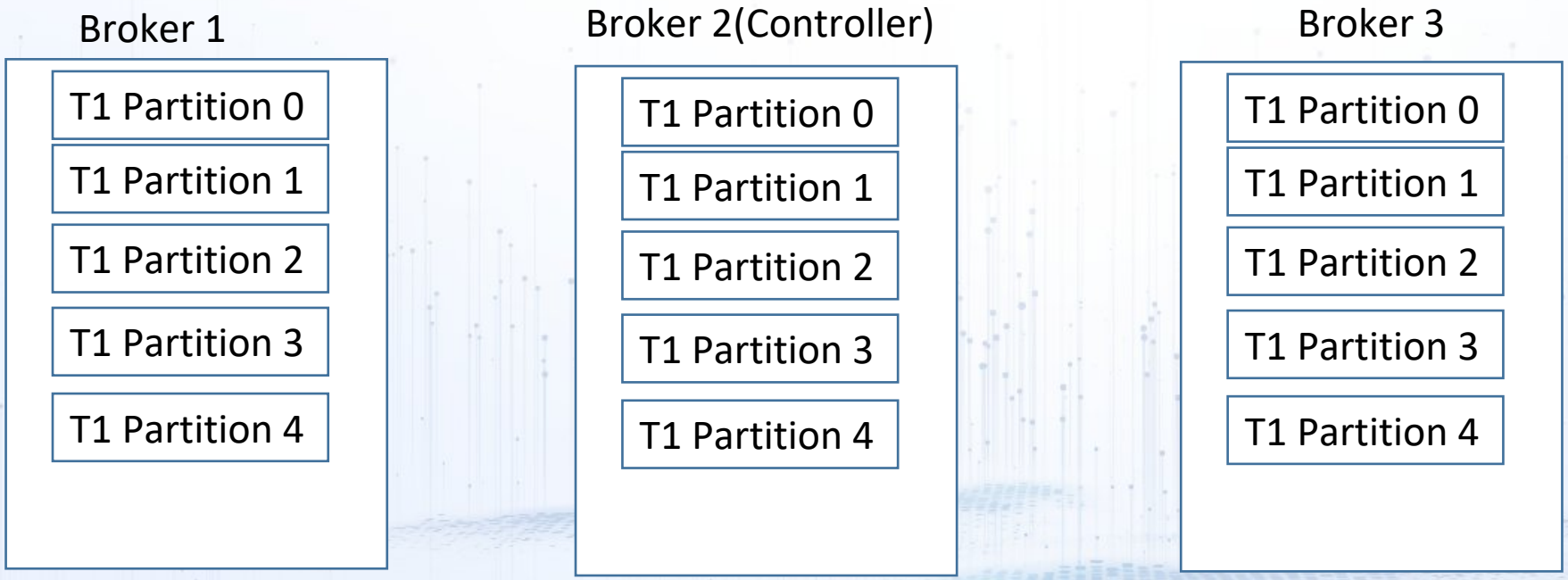
Архитектура Apache Kafka

Предположим топик имеет 5 партиций



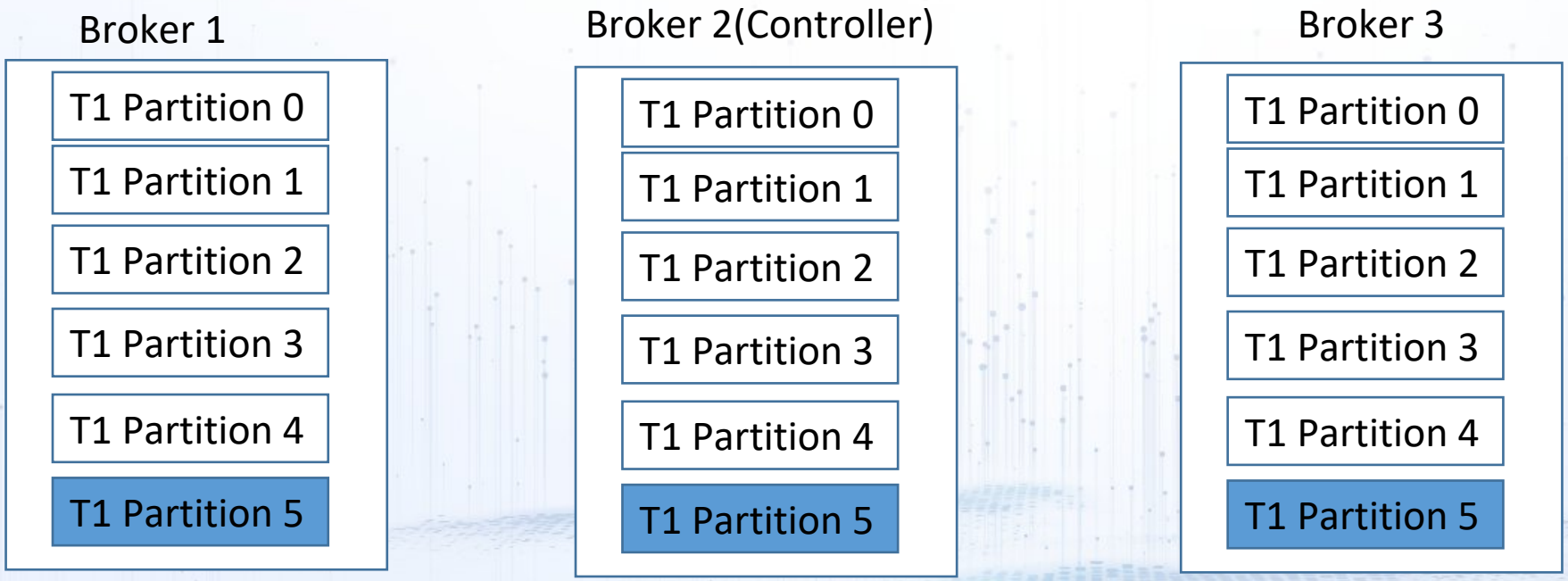
Архитектура Apache Kafka

Replication factor = 3(настройка топика)



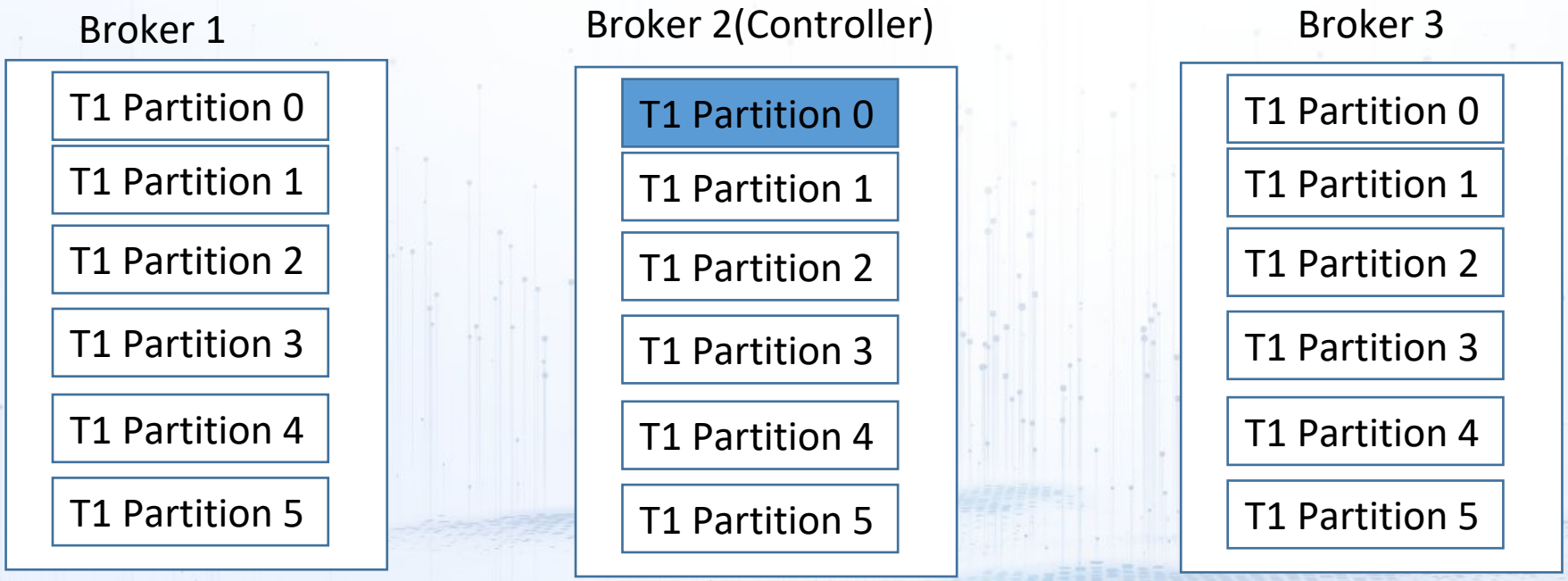
Архитектура Apache Kafka

Replication factor = 3, Добавили одну партицию



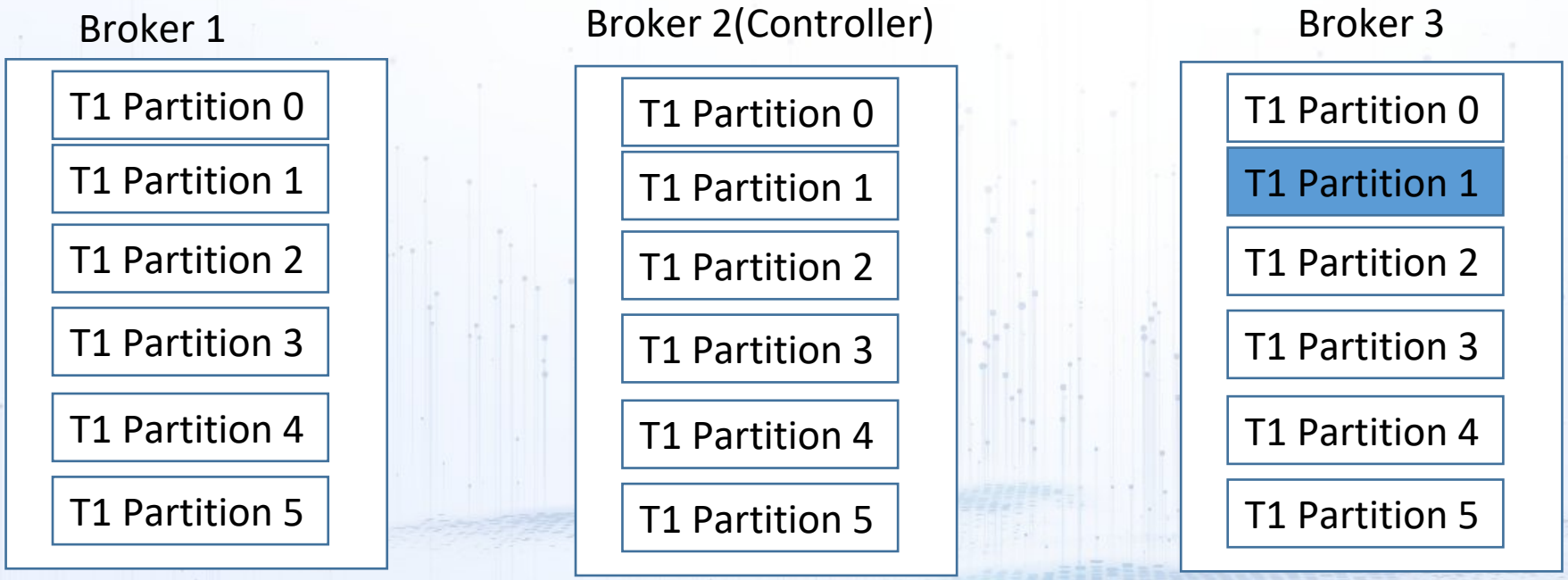
Архитектура Apache Kafka

Чтобы корректно писать, надо выбирать брокер, кто будет лидером для каждой партиции



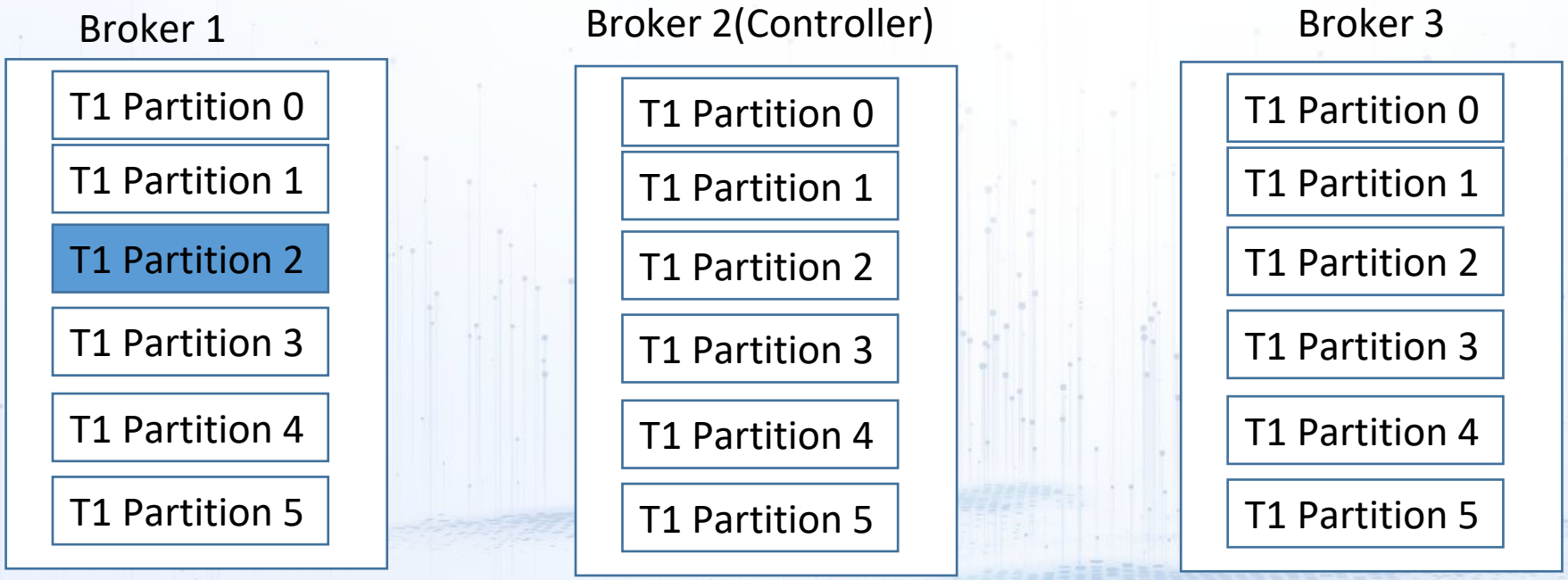
Архитектура Apache Kafka

Чтобы корректно писать, надо выбирать брокер, кто будет лидером для каждой партиции



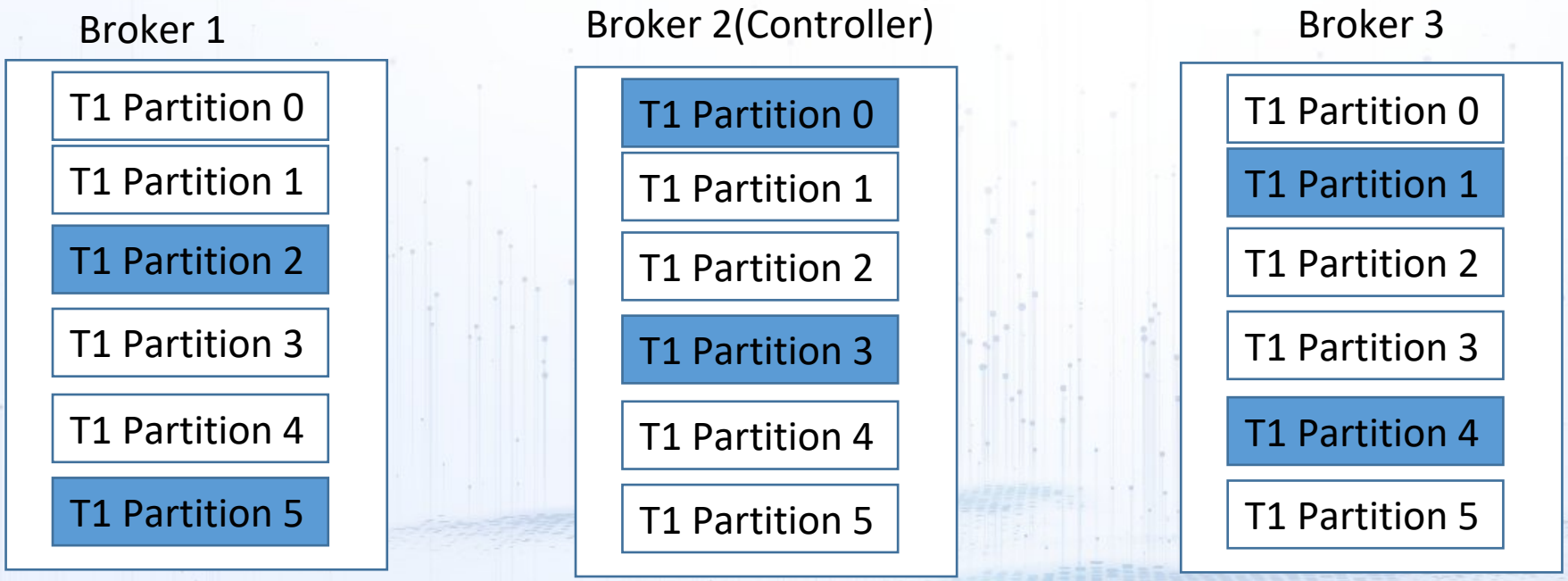
Архитектура Apache Kafka

Чтобы корректно писать, надо выбирать брокер, кто будет лидером для каждой партиции



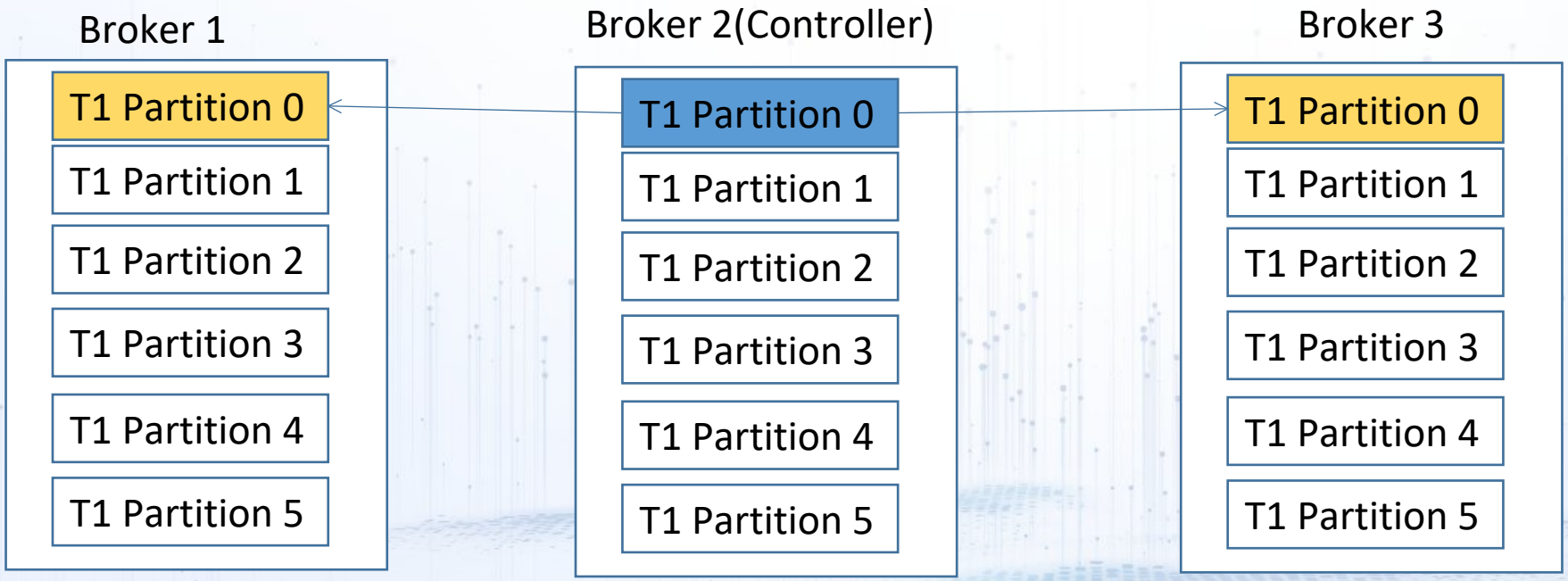
Архитектура Apache Kafka

Нагрузку желательно распределять равномерно по брокерам(следить, чтобы в одном брокере не было очень много лидеров для партиций)



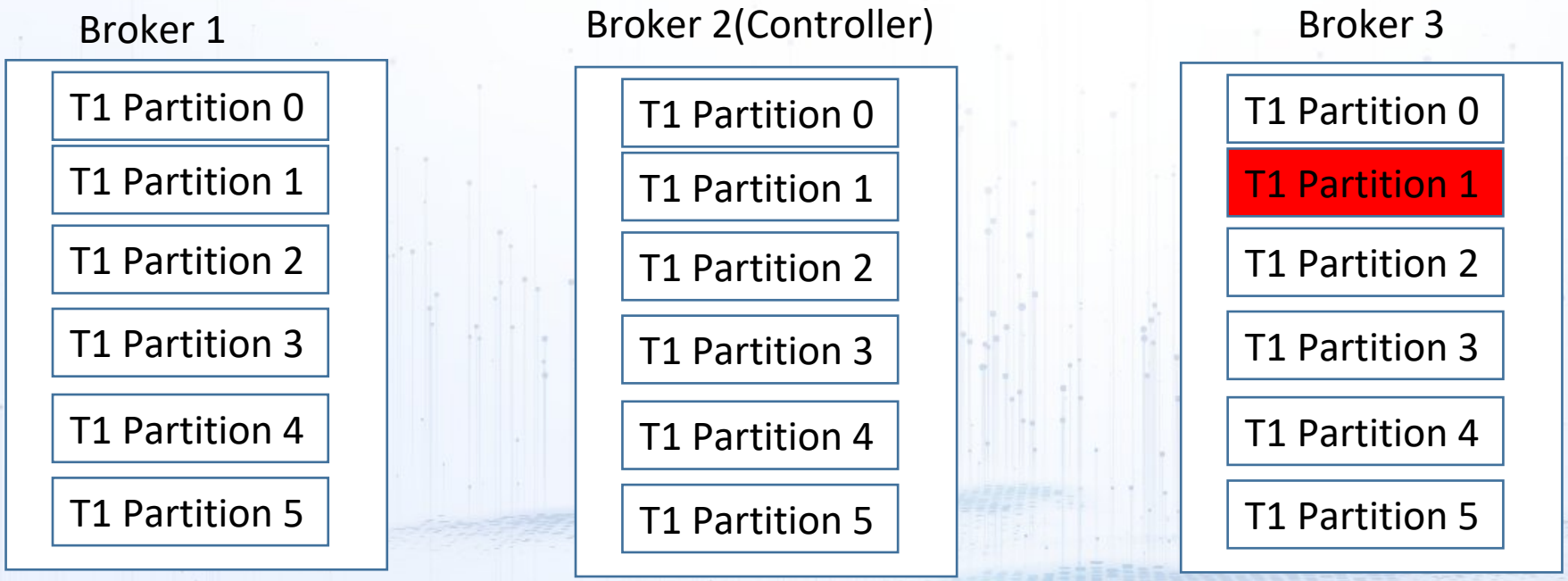
Архитектура Apache Kafka

ISR – In-Sync Replica. ISR могут участвовать в перевыборе лидера



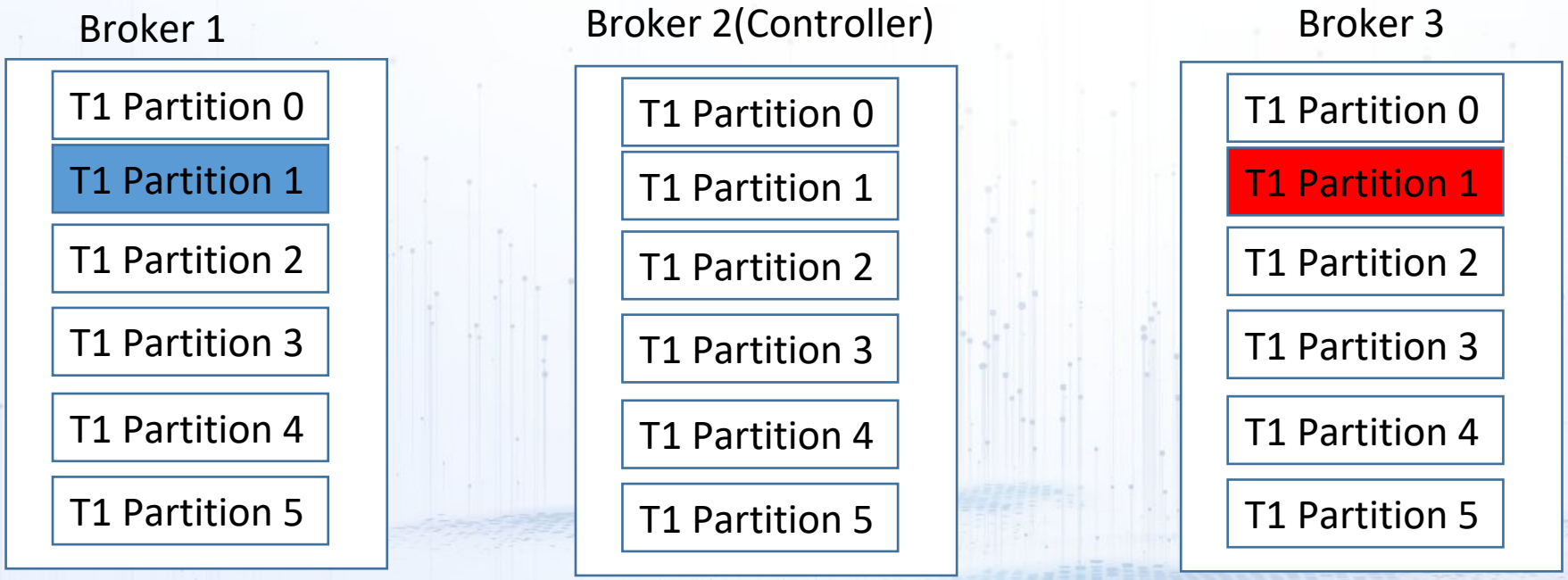
Архитектура Apache Kafka

Если погибает лидер



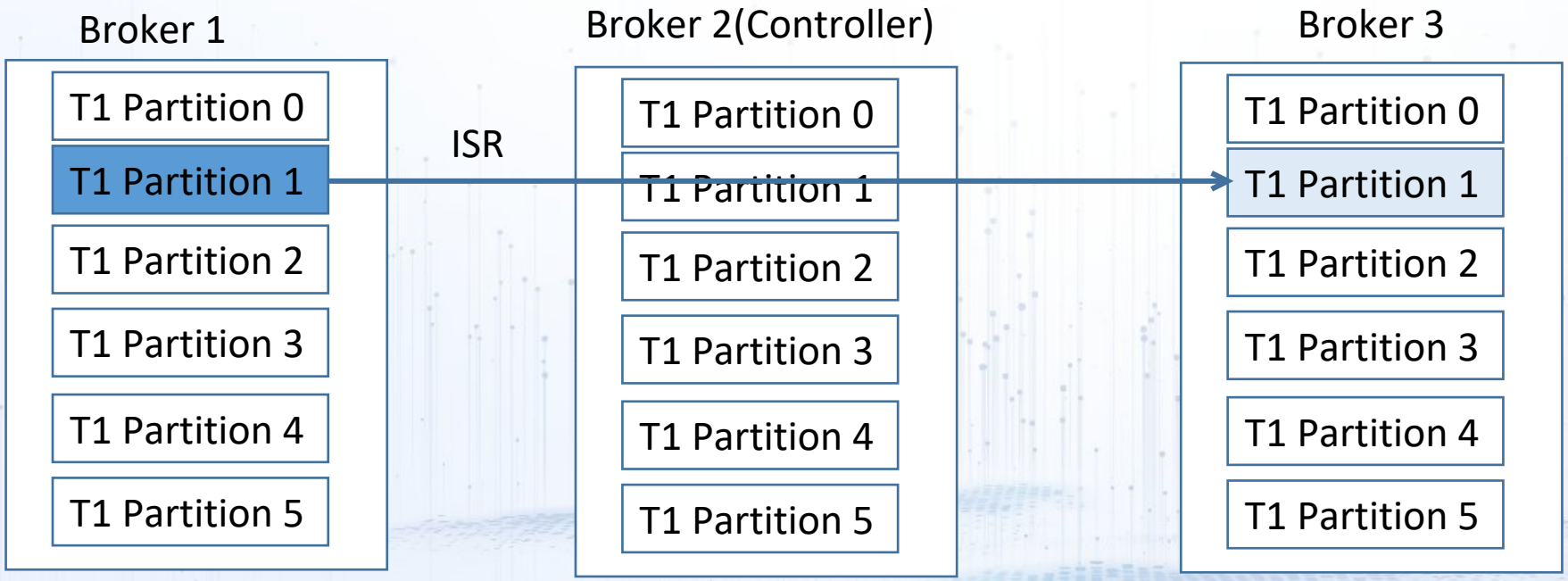
Архитектура Apache Kafka

То мы можем выбрать лидера на другом брокере(за это отвечает контроллер)



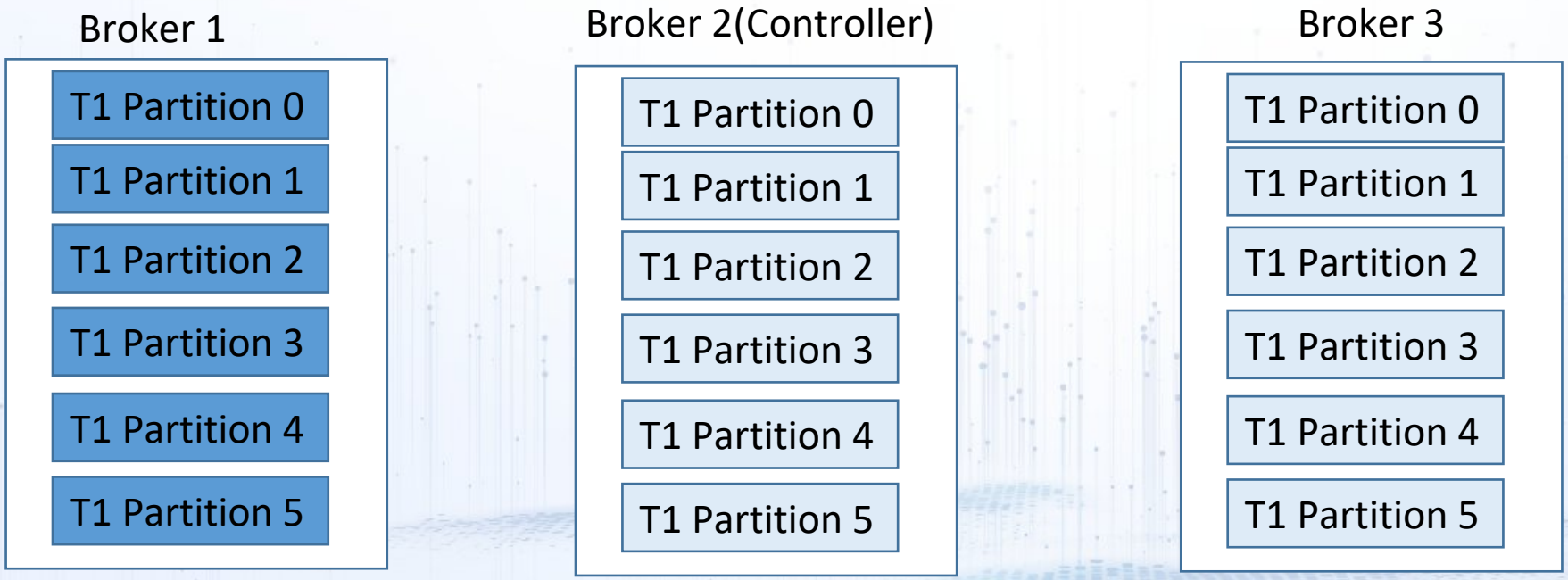
Архитектура Apache Kafka

Когда брокер 3 восстановится, то и партиция на нём восстановится как ISR



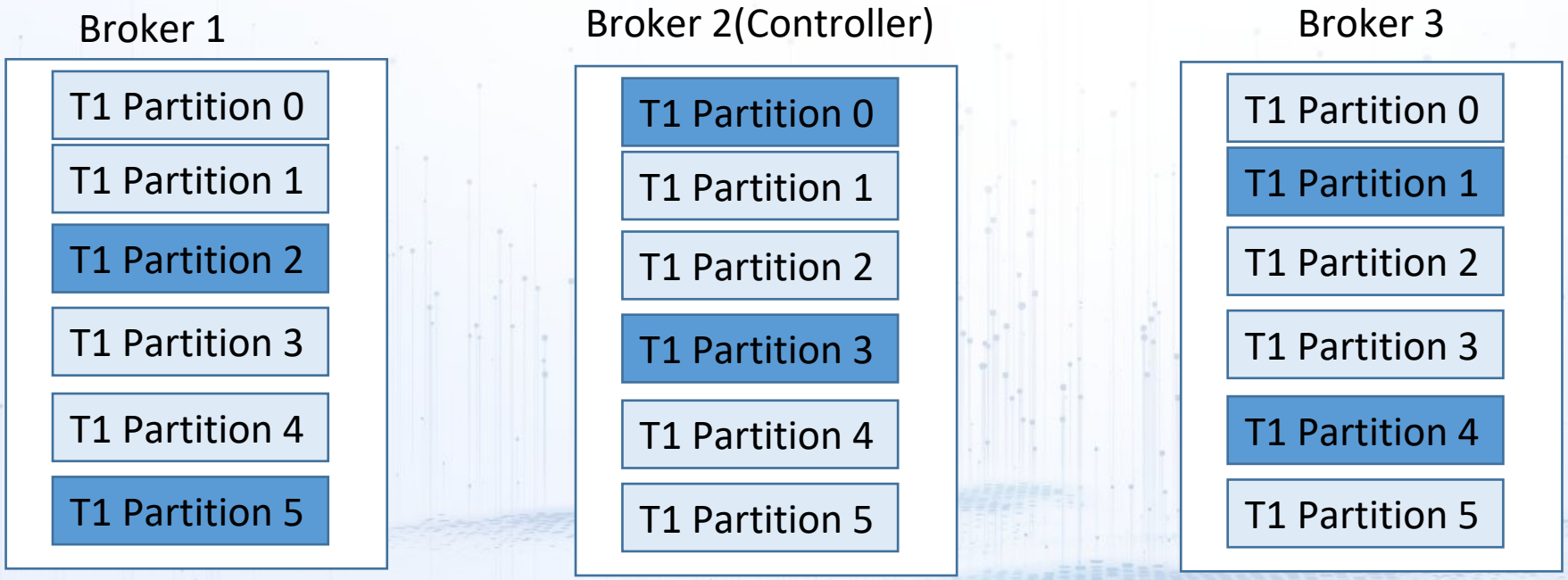
Архитектура Apache Kafka

Если на одном из брокером будет много лидеров, то будет перевыбор лидеров



Архитектура Apache Kafka

Нагрузку желательно распределять равномерно по брокерам(следить, чтобы в одном брокере не было очень много лидеров для партиций)



SolarLab>_

Вопросы?

SolarLab>_

Kafka Producer

Kafka Producer Message

```
namespace Confluent.Kafka
{
    /// <summary>
    ///     Represents a (deserialized) Kafka message.
    /// </summary>
    21 references
    public class Message<TKey, TValue> : MessageMetadata
    {
        /// <summary>
        ///     Gets the message key value (possibly null).
        /// </summary>
        14 references
        public TKey Key { get; set; }

        /// <summary>
        ///     Gets the message value (possibly null).
        /// </summary>
        14 references
        public TValue Value { get; set; }
    }
}
```

Kafka Producer Message

```
/// <summary>
///     All components of <see cref="Message{TKey,TValue}" /> except Key and Value.
/// </summary>
1 reference
public class MessageMetadata
{
    /// <summary>
    ///     The message timestamp. The timestamp type must be set to CreateTime.
    ///     Specify Timestamp.Default to set the message timestamp to the time
    ///     of this function call.
    /// </summary>
    15 references
    public Timestamp Timestamp { get; set; }

    /// <summary>
    ///     The collection of message headers (or null). Specifying null or an
    ///      empty list are equivalent. The order of headers is maintained, and
    ///     duplicate header keys are allowed.
    /// </summary>
    14 references
    public Headers Headers { get; set; }
}
```

Kafka Producer .net client Message

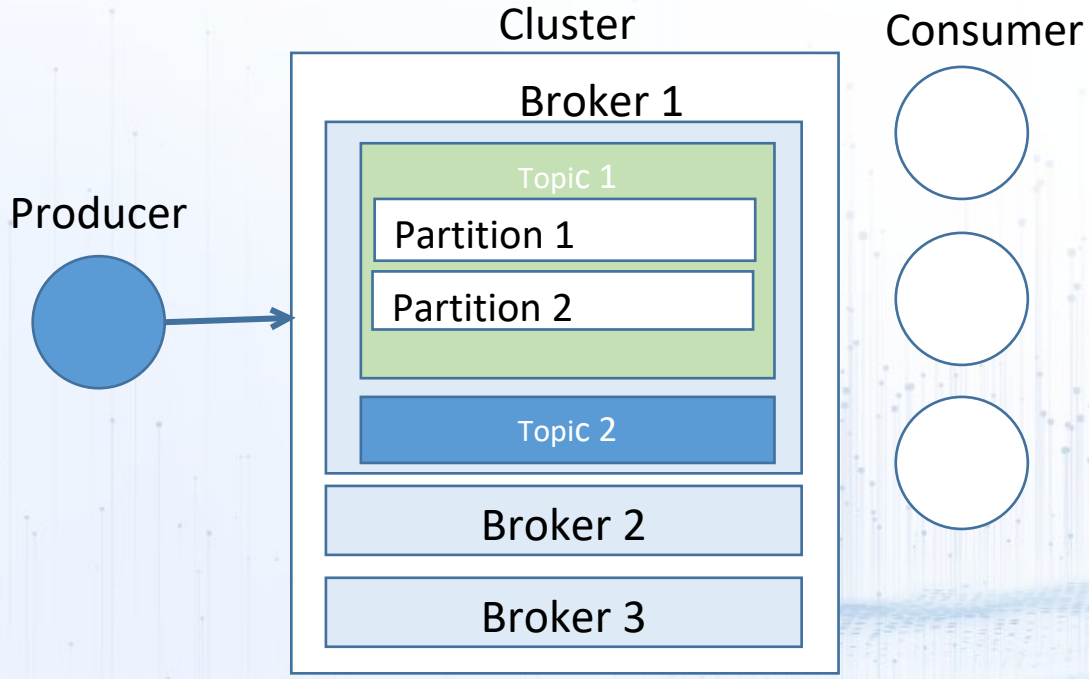
Key = любое значение, и даже Confluent.Kafka.Null

Value = так же любое значение. Очень часто string

С точки зрения кафки и ключ и значение – это массив байт.

Kafka Producer

На Broker 1 Topic 1 является лидером в кластере. В какую партицию для топика попадут сообщения?



Kafka Producer

Существует Kafka Default Partitioner, который направляет сообщения в разные партиции. Он следует ряду правил следующим образом:

- Если producer явно указывает номер партиции в сообщении, то используется предоставленный номер.
- Если producer не указывает номер партиции, но предоставляет ключ, выбирается раздел на основе хэш-значения ключа.
- Если producer не отправляет ни номер раздела, ни ключ, назначается случайный раздел

Kafka Producer Partitioner

ConsistentRandom – Хэш CRC2(Message.Key)(по умолчанию)

Random – случайная партиция, в не зависимости от ключа

Consistent - Хэш CRC2(Message.Key), если key = null, то все события попадут в одну партицию

MurMur2 – алгоритм MurMur2 для получения хэш значений для совместимости с Java, если key = null, то все события попадут в одну партицию

MurMur2Random - алгоритм MurMur2 для получения хэш значений для совместимости с Java

```
/// <summary>
///     Partitioner enum values
/// </summary>
4 references
public enum Partitioner
{
    Random,

    Consistent,

    ConsistentRandom,

    Murmur2,

    Murmur2Random
}
```

Kafka Producer – смотрим код в студии

```
var deliveryResult =
    await producer.ProduceAsync(
        "TestDescribe",
        new Message<Null, string>
        {
            Value = DateTime.Now.ToString("o")
        },
        cancellationToken);
```

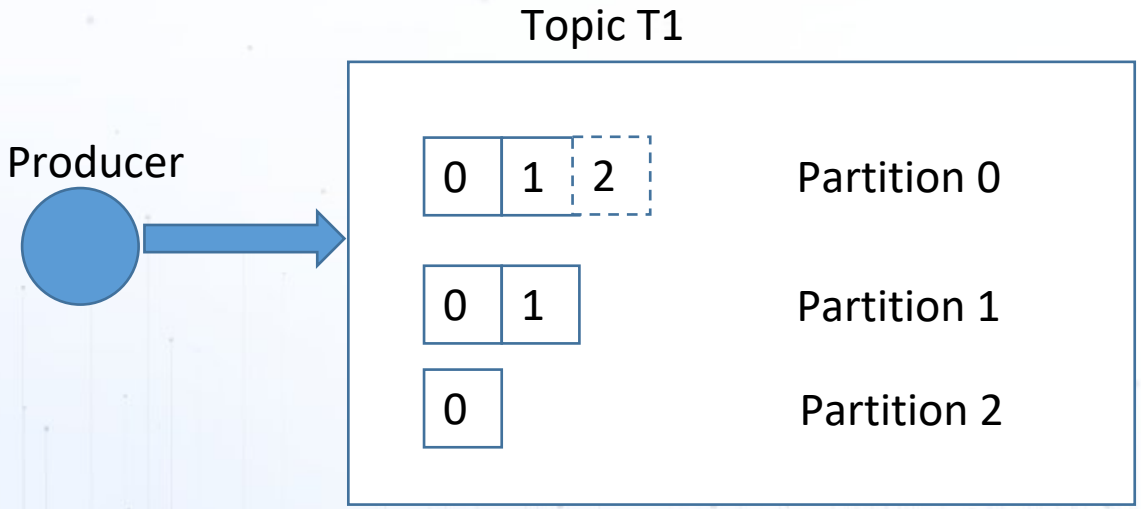
Kafka Producer – гарантии доставки

None – Брокер прочитал сообщение, и говорит что всё ок,
Не важно сохранил на диск или нет. Сделал replica factor или нет
Leader – Сообщение сохранилось на диске для брокера лидера,
Реплика фактор не учитывается. Если есть реплики, то они
попадут на брокеры позже
All – Сообщение сохранилось на лидера и на все брокеры согласно
replica factor и значения min.insync.replica(2 = leader + 1 ISR). (**По умолчанию)**

min.insync.replica – настройка топика

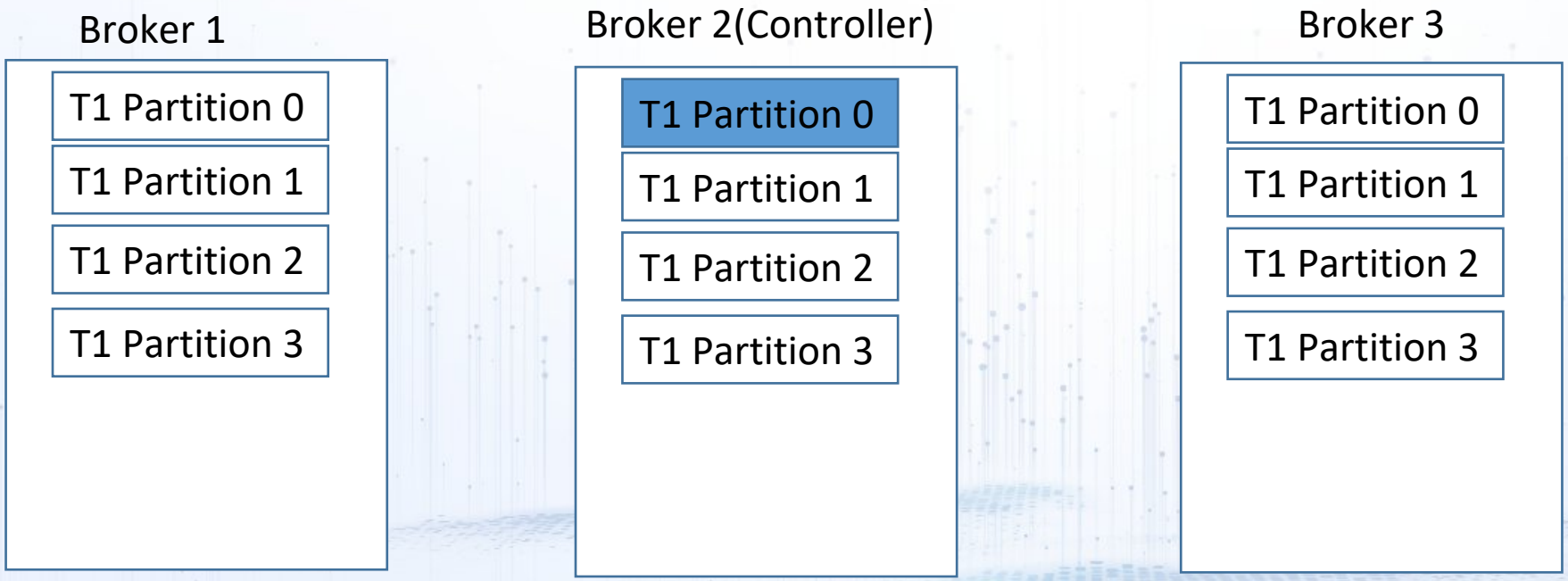
```
public enum Acks : int
{
    /// <summary>
    ///     None
    /// </summary>
    None = 0,

    /// <summary>
    ///     Leader
    /// </summary>
    Leader = 1,

    /// <summary>
    ///     All
    /// </summary>
    All = -1
}
```

Kafka Producer – гарантии доставки All



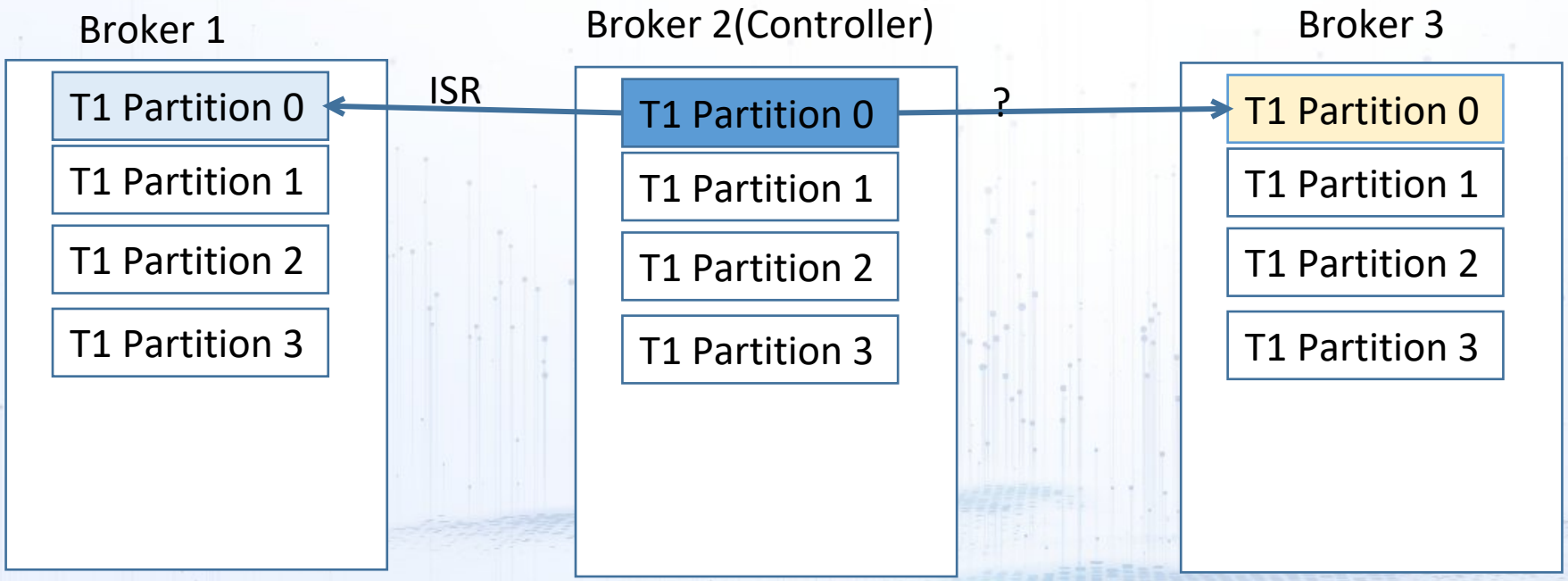
Kafka Producer – гарантии доставки All

Когда Broker 2 для топика T1 Partition 0 является лидером



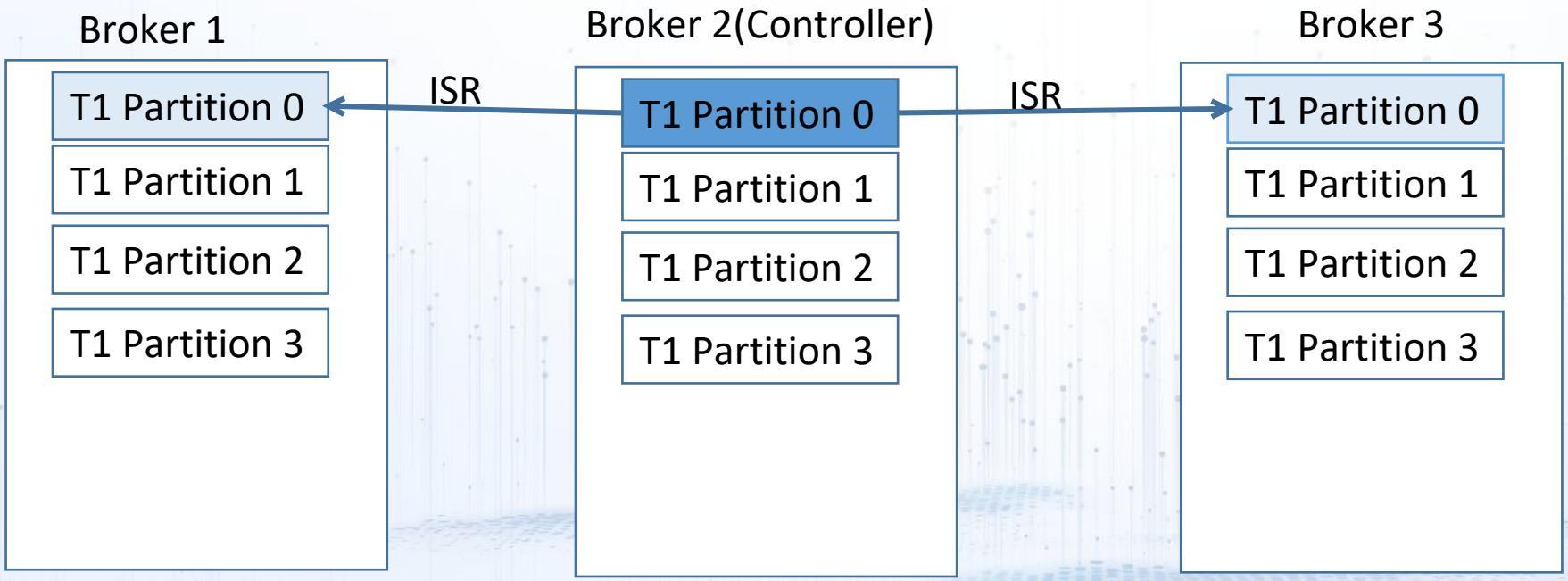
Kafka Producer – гарантии доставки All

min.insync.replica = 2. То уже считаем гарантии доставки выполненными



— Kafka Producer – гарантии доставки All

Для третьего брокера партиция 0 так же будет синхронизированна



Kafka Producer – отправка пачками

Настройки Producer

- **BatchSize** - Maximum size (in bytes) of all messages batched in one MessageSet, including protocol framing overhead. This limit is applied after the first message has been added to the batch, regardless of the first message's size, this is to ensure that messages that exceed batch.size are produced. The total MessageSet size is also limited by batch.num.messages and message.max.bytes.

default: 1 000 000

- **BatchNumMessages** - Maximum number of messages batched in one MessageSet. The total MessageSet size is also limited by batch.size and message.max.bytes.

default: 10 000

- **LingerMs** - Delay in milliseconds to wait for messages in the producer queue to accumulate before constructing message batches (MessageSets) to transmit to brokers. A higher value allows larger and more effective (less overhead, improved compression) batches of messages to accumulate at the expense of increased message delivery latency.

default: 5

Kafka Producer – сжатие

- По умолчанию без сжатия None
- Можно выставлять сжатие, которое считаете необходимым

Сжатие расходует процессорное время, но уменьшает объём сообщения

```
public enum CompressionType
{
    /// <summary>
    ///     None
    /// </summary>
    None,

    /// <summary>
    ///     Gzip
    /// </summary>
    Gzip,

    /// <summary>
    ///     Snappy
    /// </summary>
    Snappy,

    /// <summary>
    ///     Lz4
    /// </summary>
    Lz4,

    /// <summary>
    ///     Zstd
    /// </summary>
    Zstd
}
```

SolarLab>_

Вопросы?

SolarLab>_

Kafka Consumer

Kafka Consumer – смотрим код в студии

```
var consumerConfig = new ConsumerConfig
{
    GroupId = "test-consumer-group",
    BootstrapServers = "127.0.0.1:9092",
    AutoOffsetReset = AutoOffsetReset.Latest
};
```

```
using var kafkaConsumer = _consumerBuilder
    .Build();

kafkaConsumer.Subscribe(topic);
```

Kafka Consumer

Если подключение было первый раз, и для нашей консьюмер группы нет записей в кафке по сдвигу offset, то используется правило auto.offset.reset

Latest – читаем только новые пришедшие сообщения

(Это значение по умолчанию)

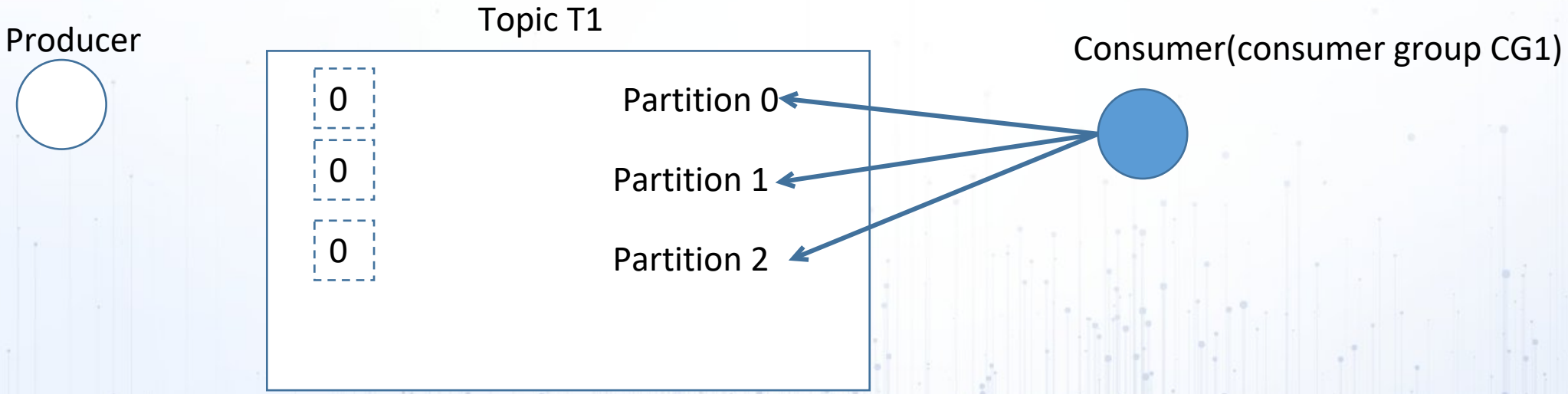
Earliest – читаем все сообщения с самого начала, которые есть в топике во всех партиция, что мы подключились

Error(none) - если нет более раннего смещения, для потребителя выдается исключение, сообщающее потребителю, что такого смещения нет во всей группе потребителей.

```
/// <summary>
///     AutoOffsetReset enum values
/// </summary>
4 references
public enum AutoOffsetReset
{
    /// <summary>
    ///     Latest
    /// </summary>
    Latest,

    /// <summary>
    ///     Earliest
    /// </summary>
    Earliest,

    /// <summary>
    ///     Error
    /// </summary>
    Error
}
```

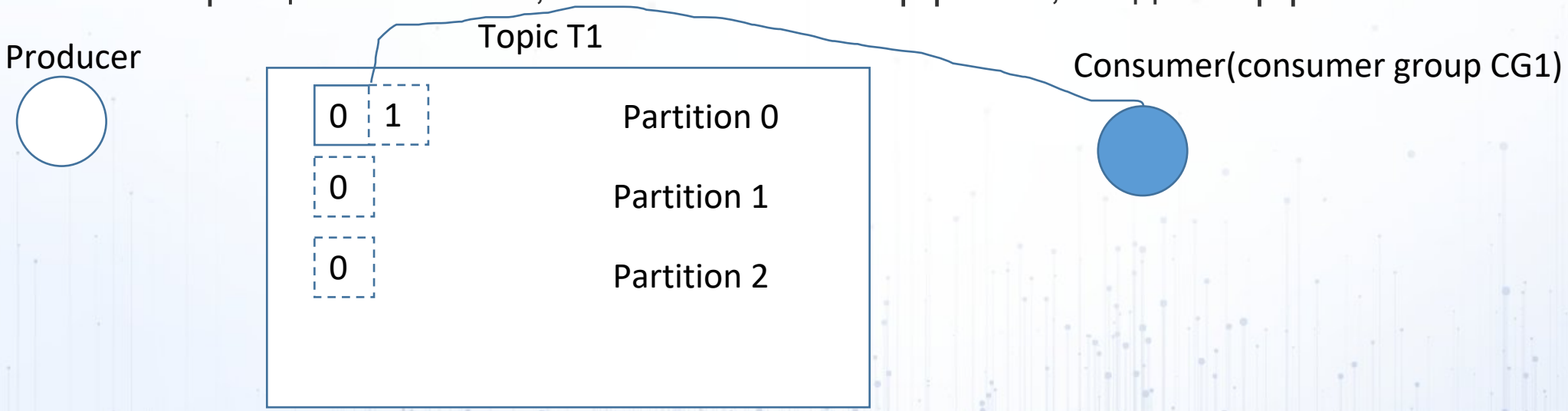
Kafka Consumer - auto.offset.reset =Latest

Первое подключение к топику с помощью .Subscribe. Если одни экземпляр, то мы подключаемся ко всем партициям. Если auto.offset.reset стоит в Latest. Если данных в топике нет



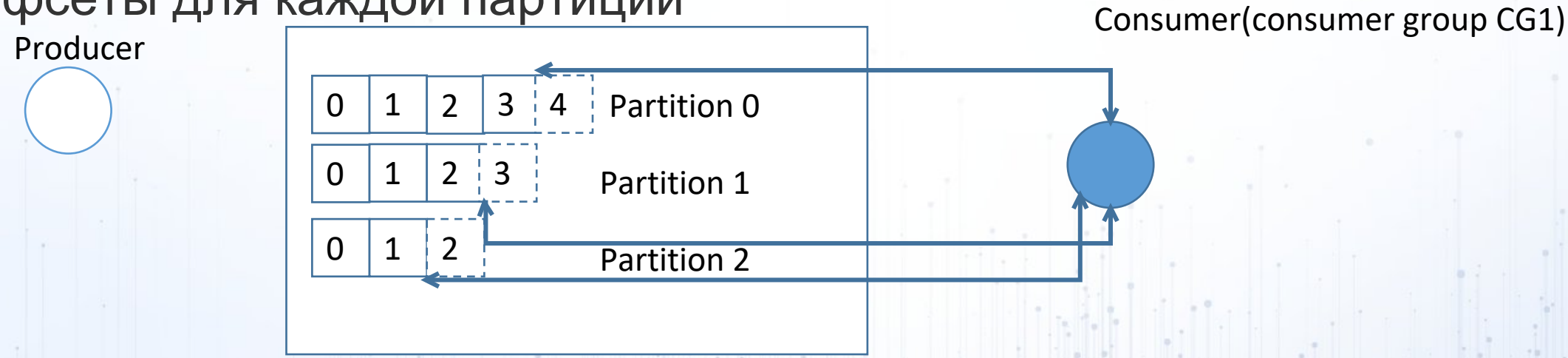
Kafka Consumer - auto.offset.reset =Latest

Первое сообщение, которое придёт, мы его и считаем. И закоммитаем оффсет, для нашего топика T1 партиции Partition 0, что мы считали оффсет 0, и ждём оффсета 1.



Kafka Consumer - auto.offset.reset =Latest

Когда придут данные в другие партиции, мы их так же считаем и сохраним офсеты для каждой партиции

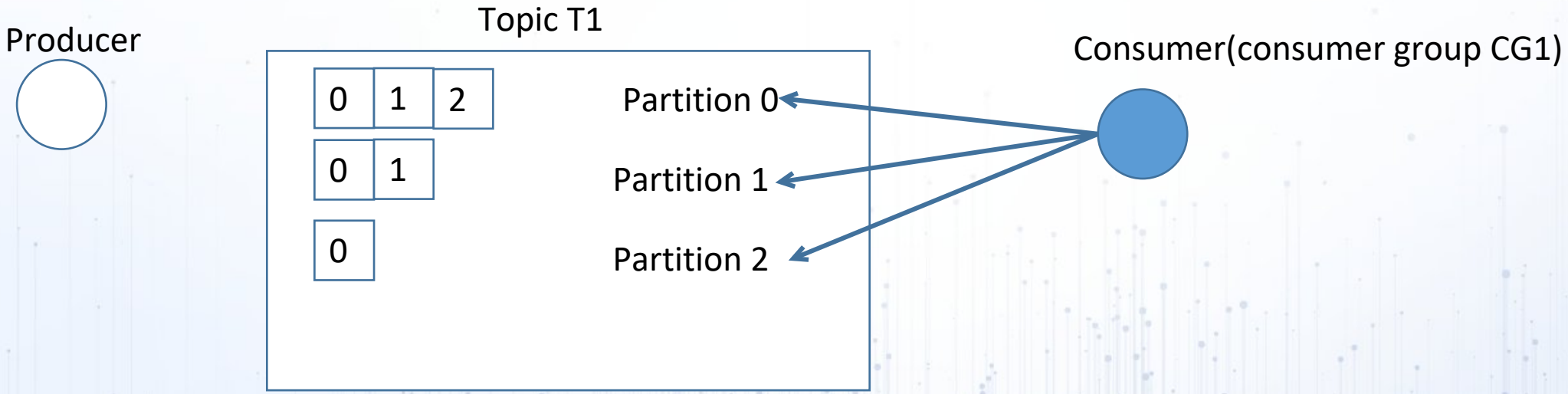


Kafka Consumer

Но что делать, если на момент подключения у нас auto.offset.reset=Latest, а данные в топике уже есть?

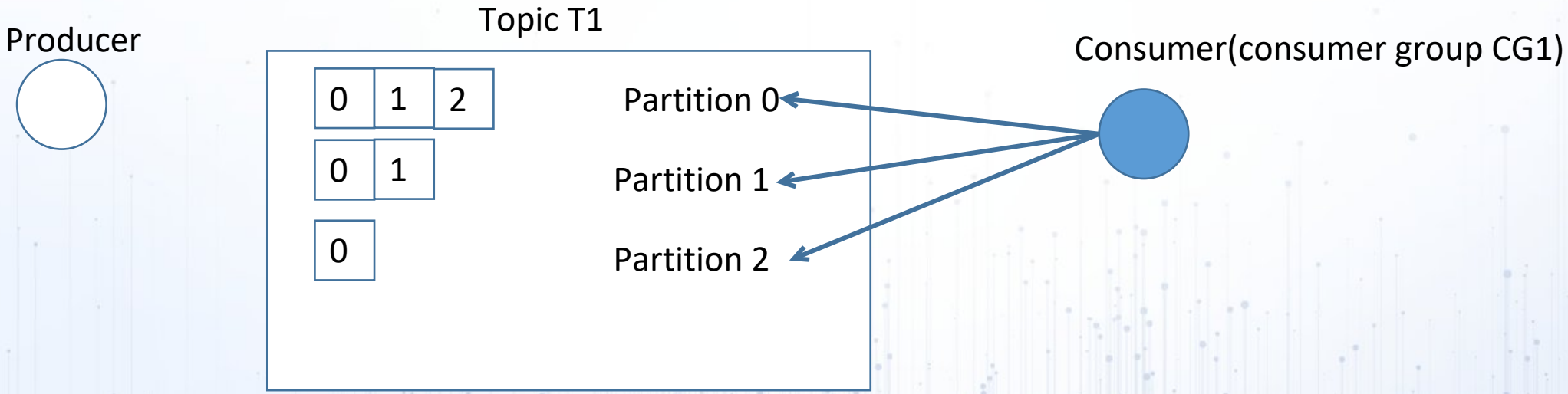
Kafka Consumer - auto.offset.reset =Latest

Первое подключение к топику с помощью .Subscribe. Если одни экземпляр, то мы подключаемся ко всем партициям. Если auto.offset.reset стоит в Latest



Kafka Consumer - auto.offset.reset =Latest

На начальном этапе мы не закоммитали ни одного офсета для нашей консьюмер группы, и при последующем подключении опять будет использоваться правило auto.offset.reset

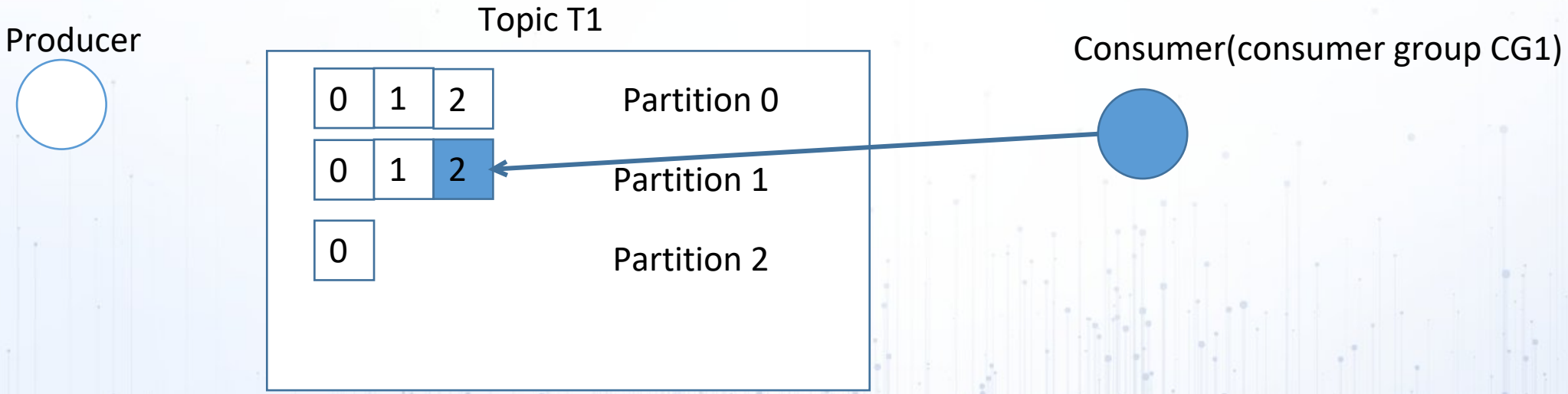


Kafka Consumer - auto.offset.reset =Latest

Как только в топик придёт новое сообщение в любую партицию для нашего топика

Kafka Consumer - auto.offset.reset =Latest

Консьюмер группы сообщение считает и приложение закоммитает offset для нашего топика, конкретной партиции, конкретной консьюмер группы



Kafka Consumer

При этом для других партиций не будет закоммиченного офсета

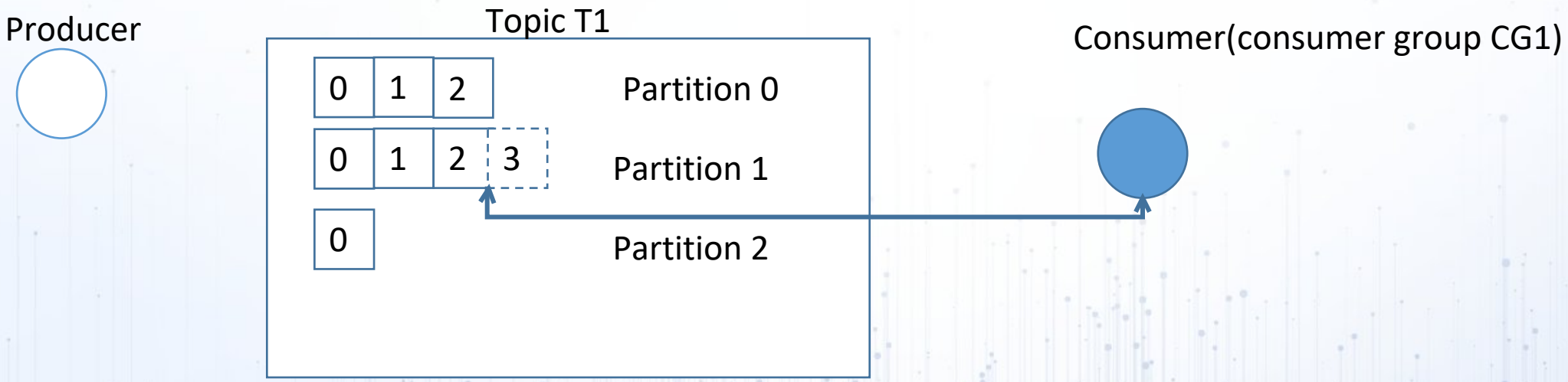
Consumer group CG1 имеет для Partition 0 offset(-1001 = Unset) – технический статус, означает, что нет закоммиченного офсета

Consumer group CG1 имеет для Partition 1 offset(2) – означает, что мы прочитали офсет 2, и ждём для этой партиции следующего сообщения в offset 3

Consumer group CG1 имеет для Partition 2 offset(-1001 = Unset) – технический статус, означает, что нет закоммиченного офсета

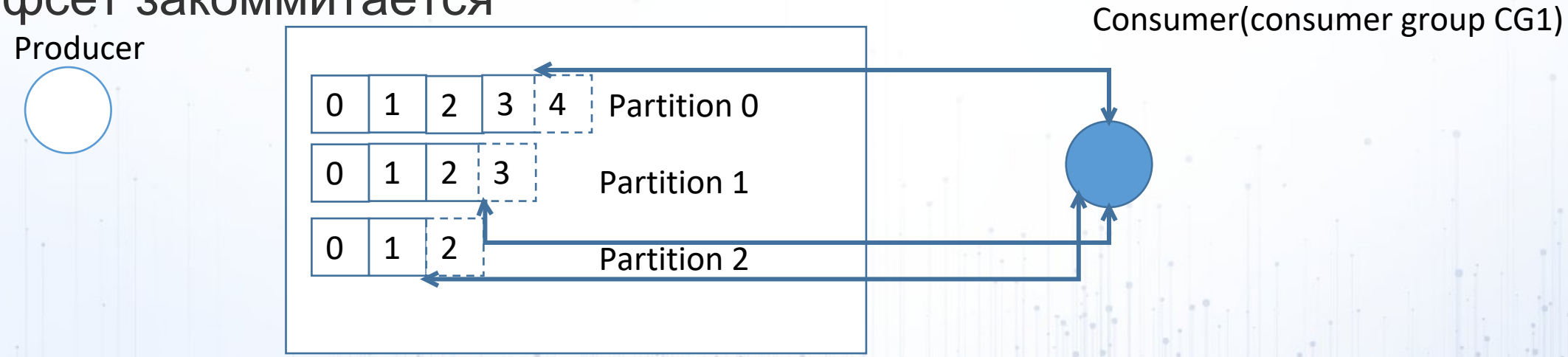
Когда данные придут в Partition 0 или в Partition 2 – тогда для них офсет закоммитается

Kafka Consumer - auto.offset.reset =Latest



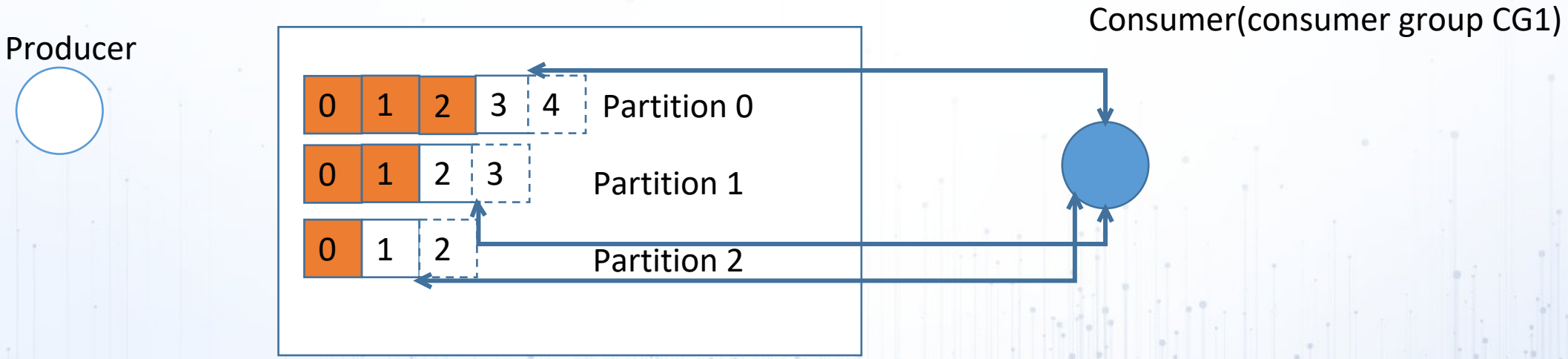
Kafka Consumer - auto.offset.reset =Latest

После того, как данные попали в Partition 0 или в Partition 2 – тогда для них офсет закоммитается



Kafka Consumer - auto.offset.reset =Latest

Новые сообщения считались успешно, а старые нет

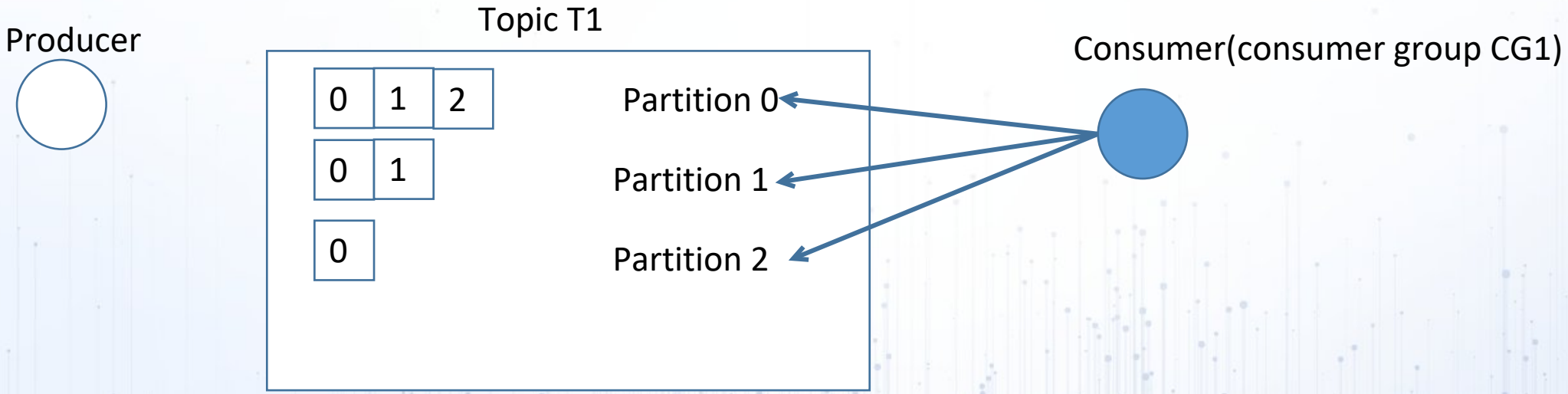


Kafka Consumer - auto.offset.reset =Earliest

Казалось бы – давайте используем **Earliest**
и наша проблема уходит

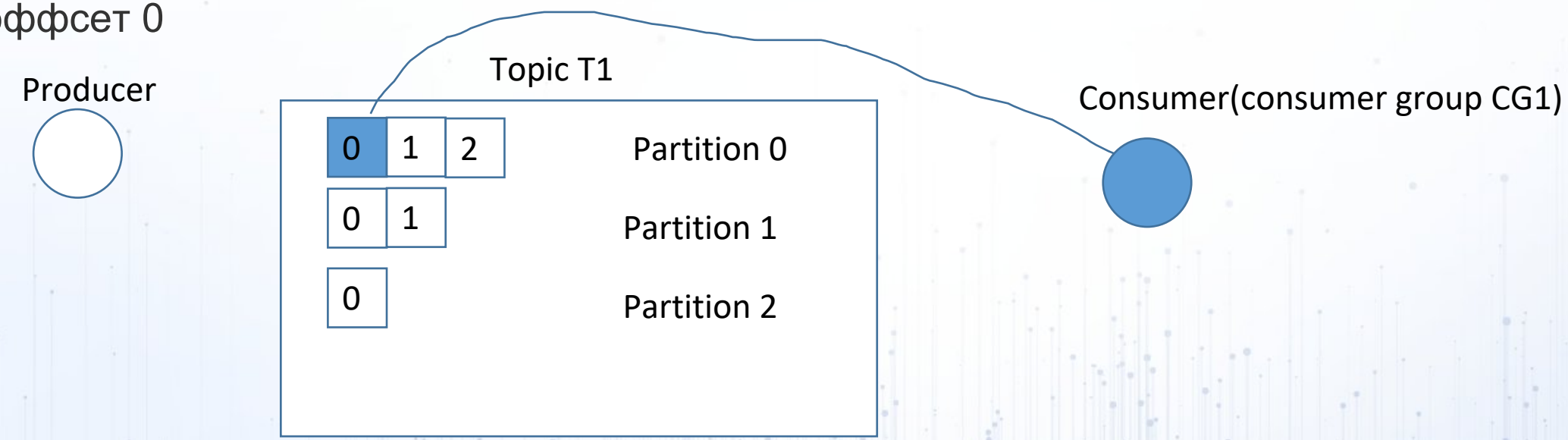
Kafka Consumer - auto.offset.reset =Earliest

Первое подключение к топику с помощью .Subscribe. Если одни экземпляр, то мы подключаемся ко всем партициям. Если auto.offset.reset стоит в Earliest



Kafka Consumer - auto.offset.reset =Earliest

Мы считаем все сообщения из топика T1 Partition 0. В начале оффсет 0 и сделаем коммит на оффсет 0



Kafka Consumer - auto.offset.reset =Earliest

И потом все оффсеты оставшиеся

Kafka Consumer - auto.offset.reset =Earliest

И потом все оффсеты оставшиеся

Kafka Consumer - auto.offset.reset =Earliest

И потом все оффсеты оставшиеся

Kafka Consumer - auto.offset.reset =Earliest

И потом все оффсеты оставшиеся

Kafka Consumer

И тут уже есть проблема – двойная(потенциально множественная) обработка сообщений. А возможно и сообщение обработается первый раз. При Earliest мы это не контролируем и не гарантируем.

Kafka Consumer - auto.offset.reset =Earliest

Как только в топик придёт новое сообщение в любую партицию для нашего топика

Kafka Consumer - auto.offset.reset =Earliest

У нас уже есть закоммиченый оффсет для Partition 1. И мы считаем оффсет 2 для Partition 1

Kafka Consumer

Выбирать какой способ лучше Latest или Earliest – надо в зависимости от данных, и подхода к обработке одного сообщения много раз.

Если новый сервис, и он должен перечитать все сообщения, что есть в партициях для топика – тут однозначно Earliest

Либо можно настроить подключение к конкретному топику, конкретной партиции, читать с нужного нам офсета

Kafka Consumer

Порядок чтения данных не гарантирован. Сообщения могут сохранится в Partition 0, потом Partition 2, потом Partition 1. А при подключении мы их считаем Partition 0, Partition 1, Partition 2.

SolarLab>_

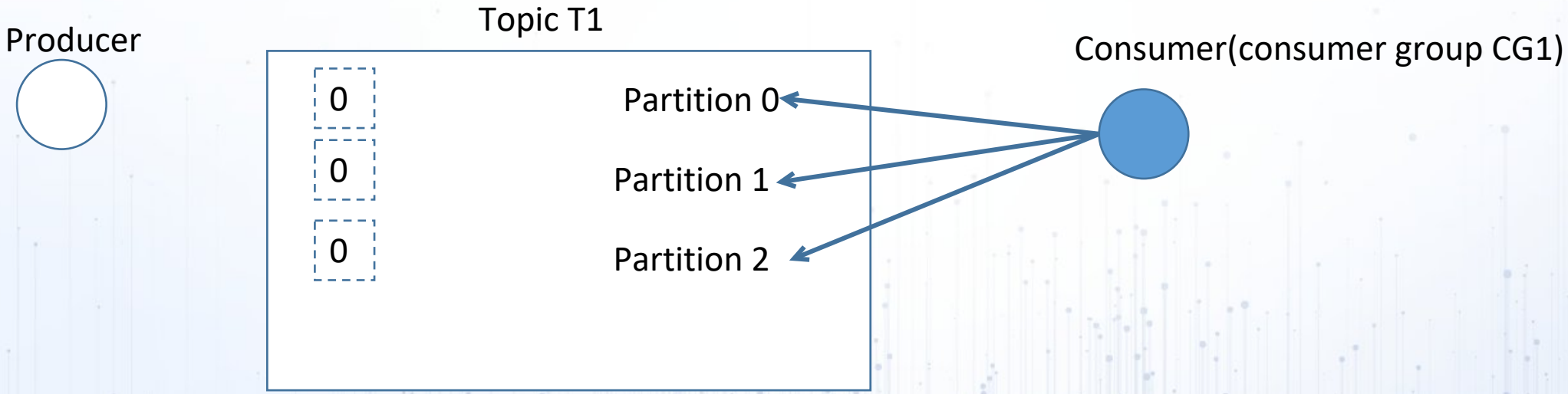
Вопросы?

SolarLab>_

Масштабирование для Consumer

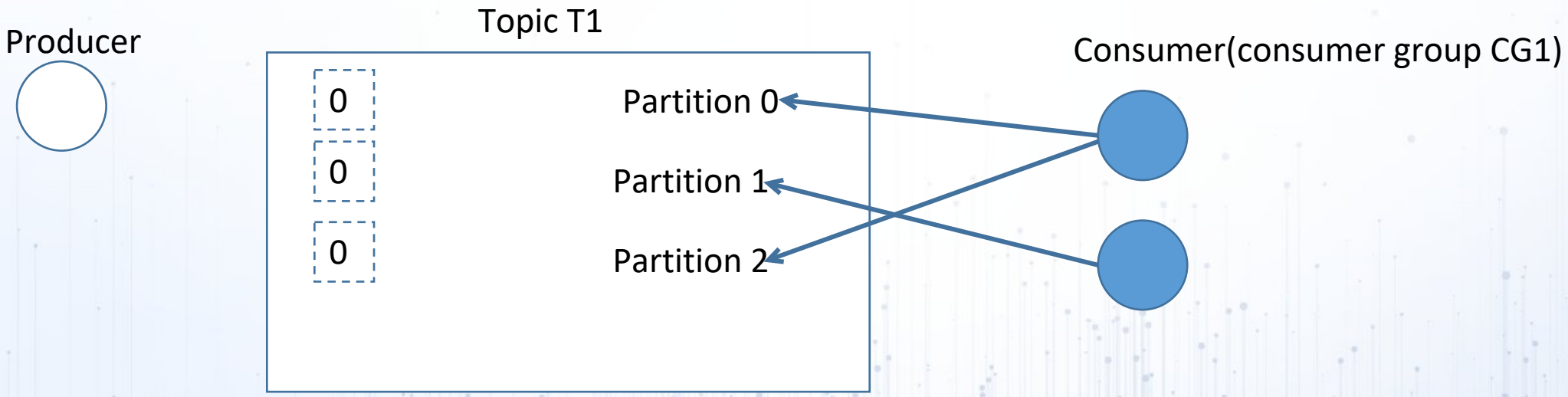
Kafka - масштабирование на чтение

Чем хороша кафка – возможностью быстрого масштабирования. При подключении одной consumer group – одного сервиса мы читаем все партиции топика.



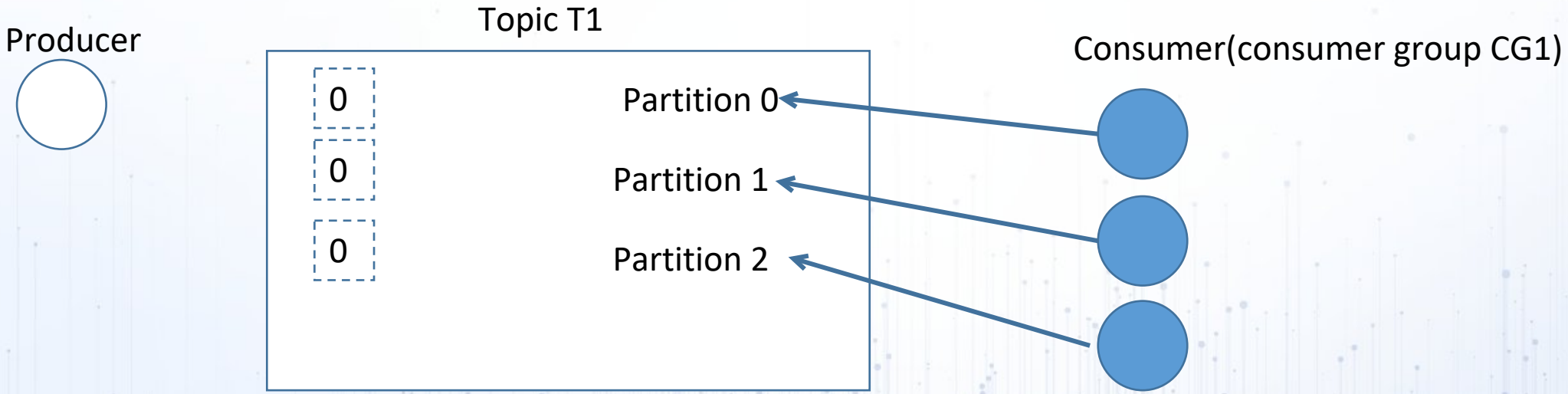
Kafka - масштабирование на чтение

Если мы добавил ещё один инстанс нашего сервиса – то будет перераспределение партиций



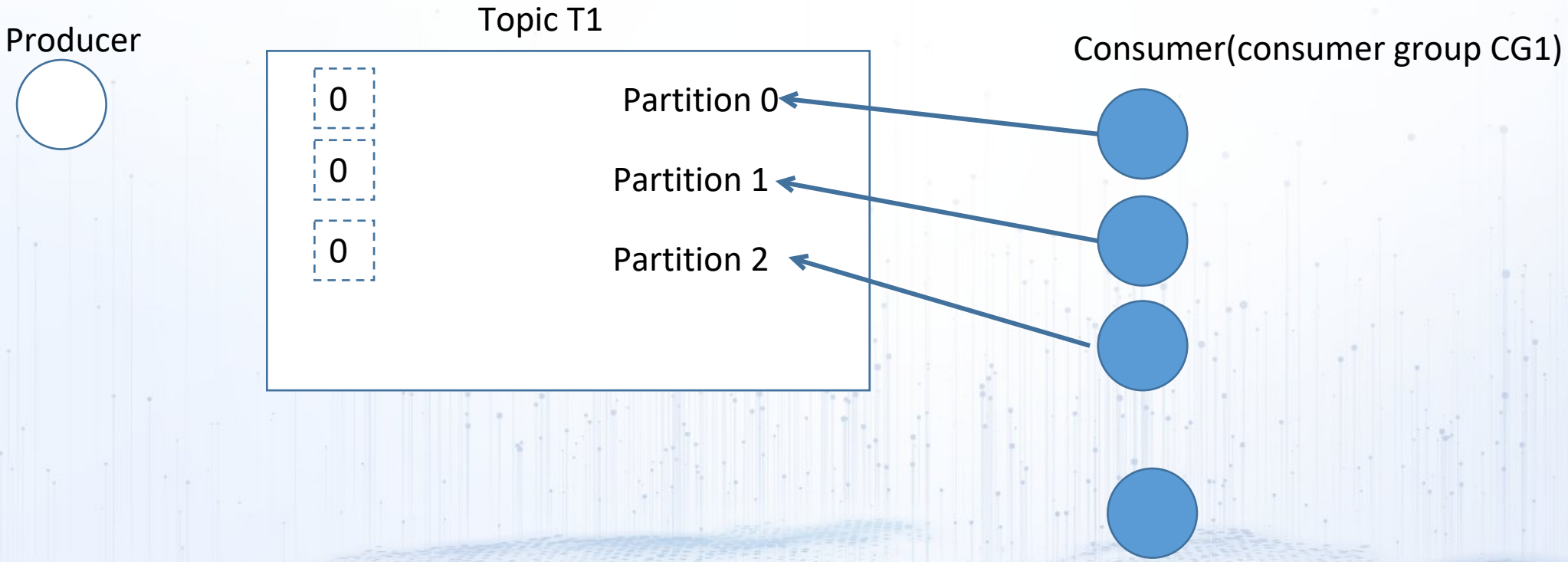
Kafka - масштабирование на чтение

Если будет три партиции и три сервиса с одинаковой consumer group – тогда мы будем обрабатывать каждый свою партицию



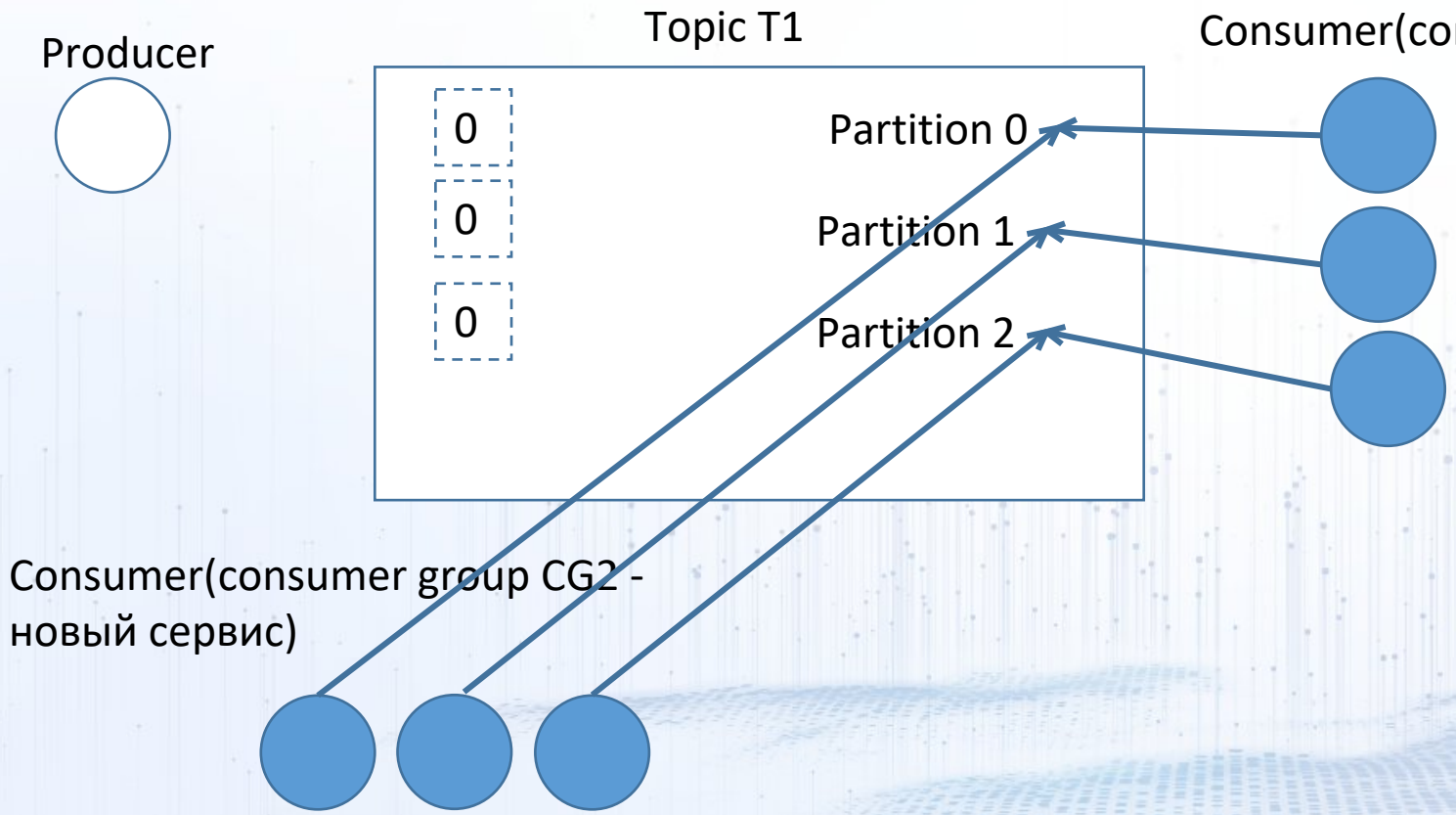
Kafka - масштабирование на чтение

Если сервисов с одинаковой consumer group будет больше чем партиций, то один из сервисов будет «простаивать», и как только сможет – он подключится



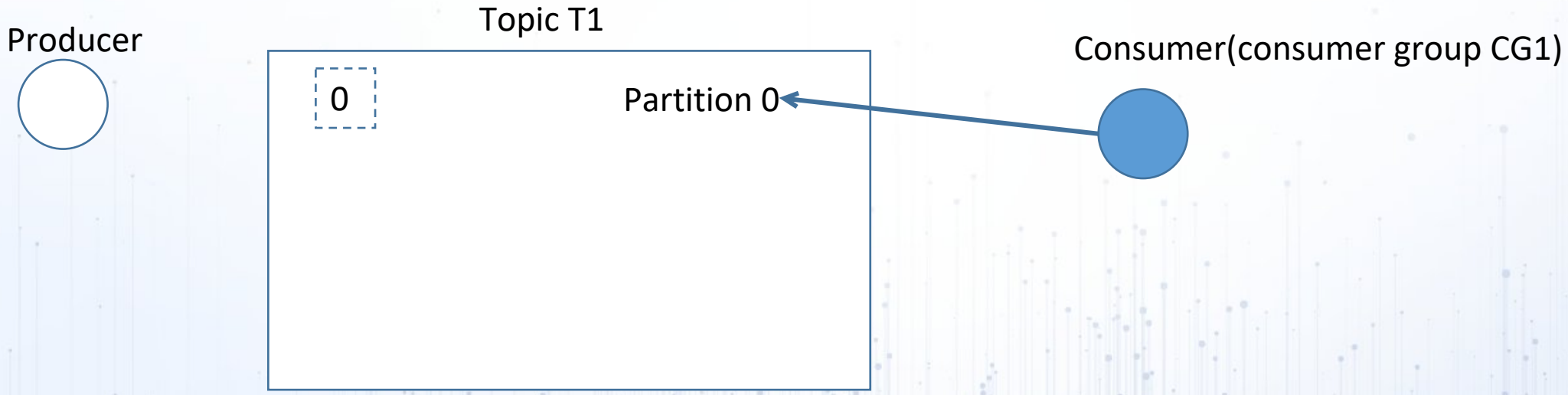
Kafka - масштабирование на чтение

При необходимости можем создать новую consumer group(например для нового сервиса). И этот сервис отдельно вычитает сообщения, и будет новые сообщения независимо обрабатывать



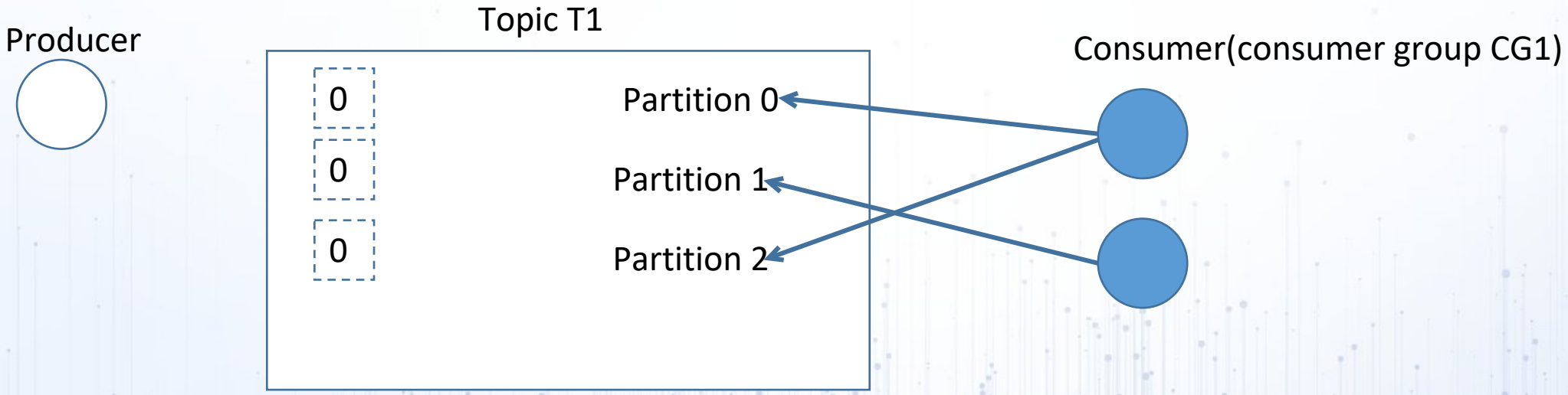
Kafka - масштабирование на чтение

Так же можно масштабировать, если у нас была всего одна партиция. И данные долго обрабатывались сервисом



Kafka - масштабирование на чтение

А мы добавили несколько новых партиций, а ещё один инстанс сервиса. Тогда данные будут обрабатываться быстрее



SolarLab>_

Что есть ещё?

Что ещё?

Kafka

RabbitMQ

Google Pub/Sub

Amazon SNS

Azure Service Bus

Что ещё?

Kafka
RabbitMQ
Google Pub/Sub
Amazon SNS
Azure Service Bus

Что же выбрать нам?

Что ещё?

Kafka
RabbitMQ
Google Pub/Sub
Amazon SNS
Azure Service Bus

КОНЕЧНО
КАФКА

Что ещё?

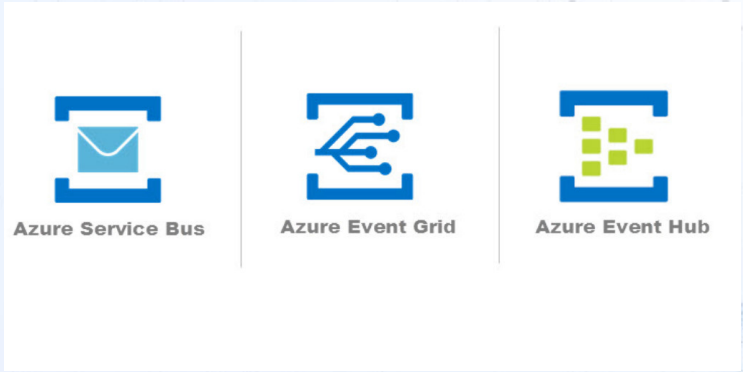
Kafka
RabbitMQ
Google Pub/Sub
Amazon SNS
Azure Service Bus

Надеюсь шутка вам понравилась

Что ещё?

Kafka
RabbitMQ
Google Pub/Sub
Amazon SNS
Azure Service Bus

Выбираем, то, что вам больше подходит



SolarLab>_

Выводы

Выводы

- Кафкой можно пользоваться
- Есть и другие продукты
- Кафка не является серебряной пулей и решением всех проблем, но знать её полезно
- Кафка умеет работать с огромными потоками данных

SolarLab>_

Ссылки

Ссылки

- https://github.com/confluentinc/confluent-kafka-dotnet
- https://kafka.apache.org/
- https://youtu.be/ghKnX5fuW5s?si=UWTF6ptQgirZn_6d
- https://www.youtube.com/watch?v=xrYefL1YlAo&list=PLmW6i9wxgkBzPZkYVQAniALZE0roLuERO&index=23&t=588s
- https://www.youtube.com/watch?v=-AZOi3kP9Js&list=PLmW6i9wxgkBzPZkYVQAniALZE0roLuERO&index=24&t=1374s
- Effective Kafka: A Hands-on Guide to Building Robust and Scalable Event-Driven Applications https://apachekafkabook.com/
- Apache Kafka. Потоковая обработка и анализ данных Бестселлеры O'Reilly https://www.piter.com/collection/all/product/apache-kafka-potokovaya-obrabotka-i-analiz-dannyh
- https://habr.com/ru/articles/270339/
- https://habr.com/ru/companies/slurm/articles/550934/
- **Битва брокеров сообщений**

  https://habr.com/ru/companies/yandex_praktikum/articles/700608/

Вопросы?

SolarLab>_

Спасибо за внимание