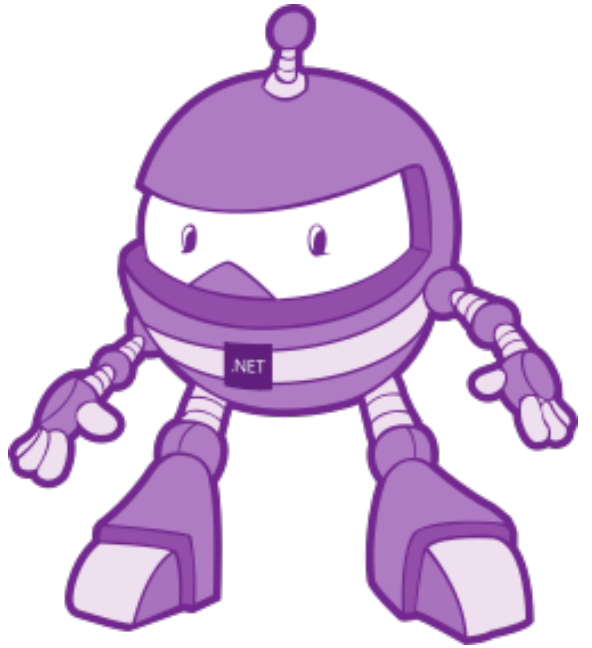
Apache Kafka и реактивные микросервисы на .NET Core

Денис Иванов
denis@ivanovdenis.ru
@denisivanov

Обо мне

2GIS

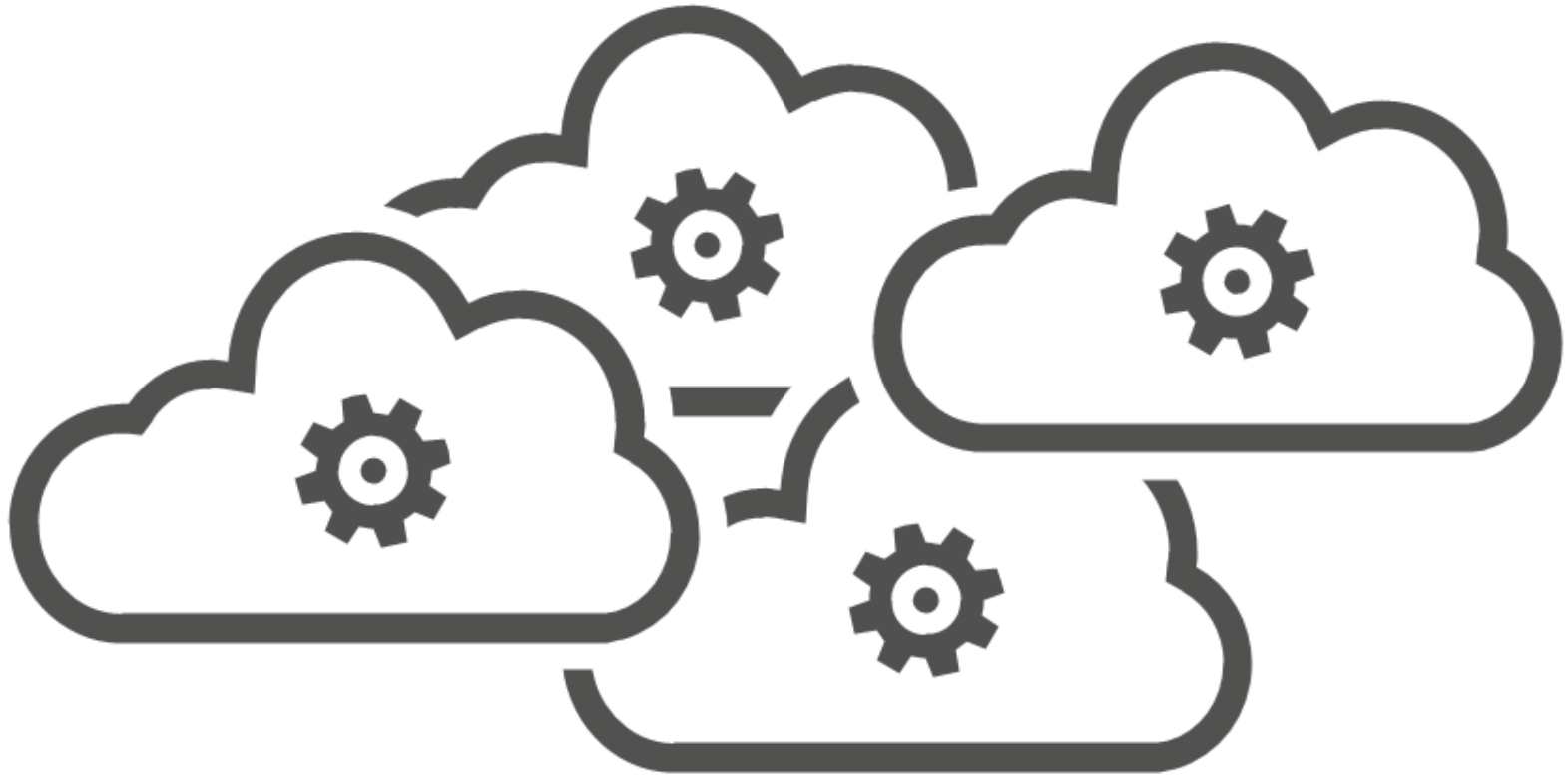
MVP Microsoft® Most Valuable Professional



Код и презентация

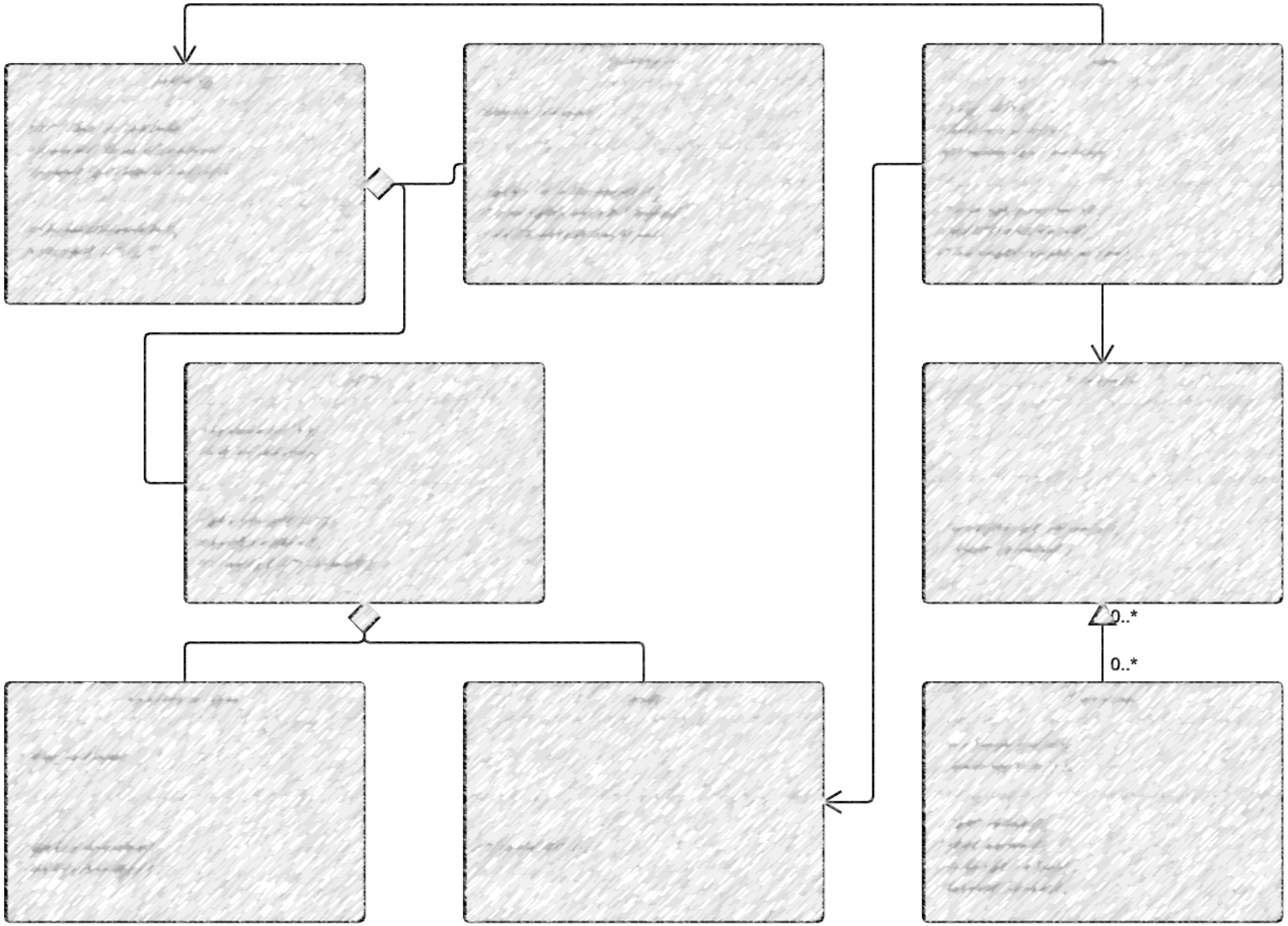
https://github.com/denisivan0v/dotnext-moscow-2017

План

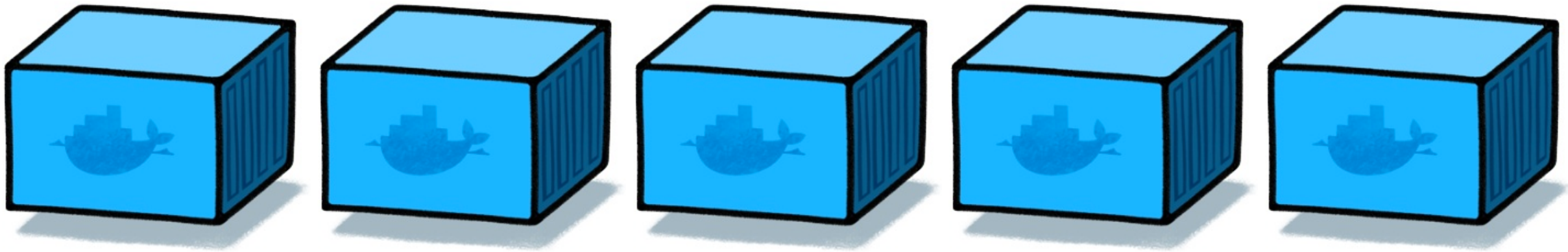


Microservices

Терминология



Терминология



kubernetes

Откуда берутся микросервисы?

Откуда берутся микросервисы?

Advertising Management System

в 2GIS

Откуда берутся микросервисы?

Frontend App

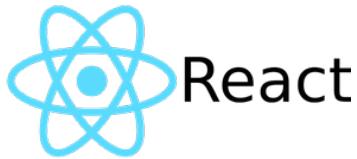
High-level API

Storage API

S3 Storage

Откуда берутся микросервисы?

Frontend App



High-level API

Storage API

S3 Storage

Challenges

- Синхронное/асинхронное взаимодействие

Challenges

- Синхронное/асинхронное взаимодействие
- Гарантированная доставка данных с минимальными задержками

Challenges

- Синхронное/асинхронное взаимодействие
- Гарантированная доставка данных с минимальными задержками
- Распределенные изменения и согласованность

Challenges

- Синхронное/асинхронное взаимодействие
- Гарантированная доставка данных с минимальными задержками
- Распределенные изменения и согласованность

Распределенные изменения

- Блокирующие
- Неблокирующие (отложенные)

Распределенные изменения

High-level API

Scala

Storage API

.NET Core

S3 Storage

ceph

Распределенные изменения

High-level API

Scala

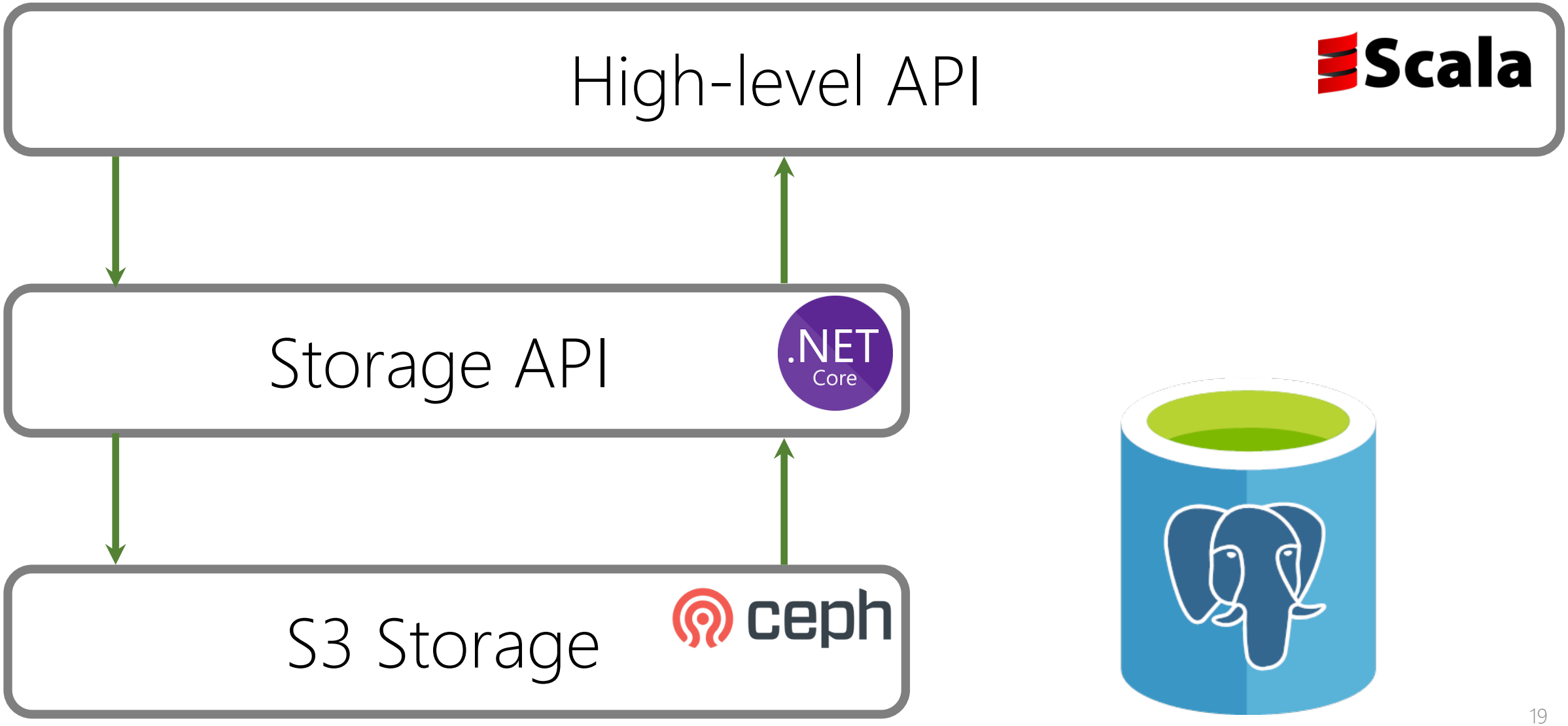
Storage API

.NET Core

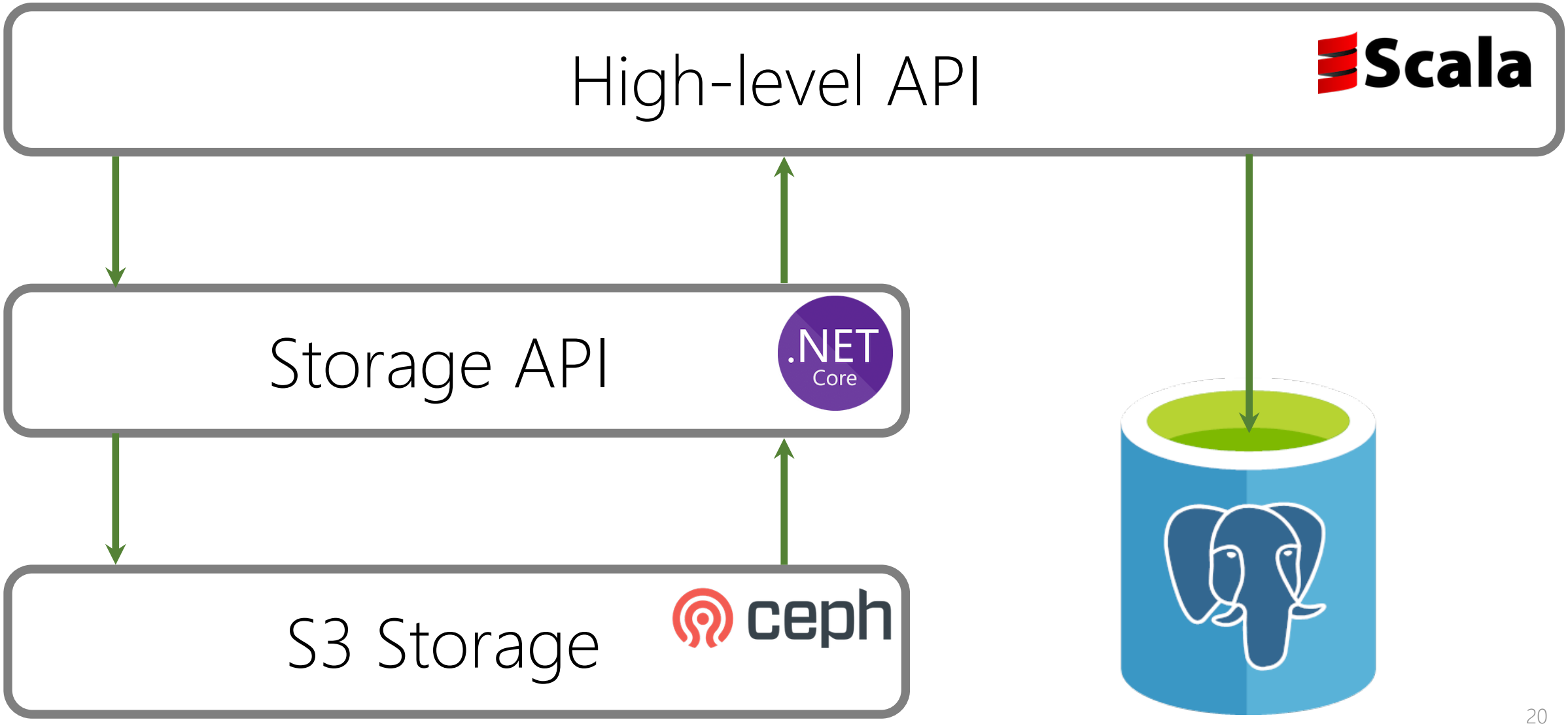
S3 Storage

ceph

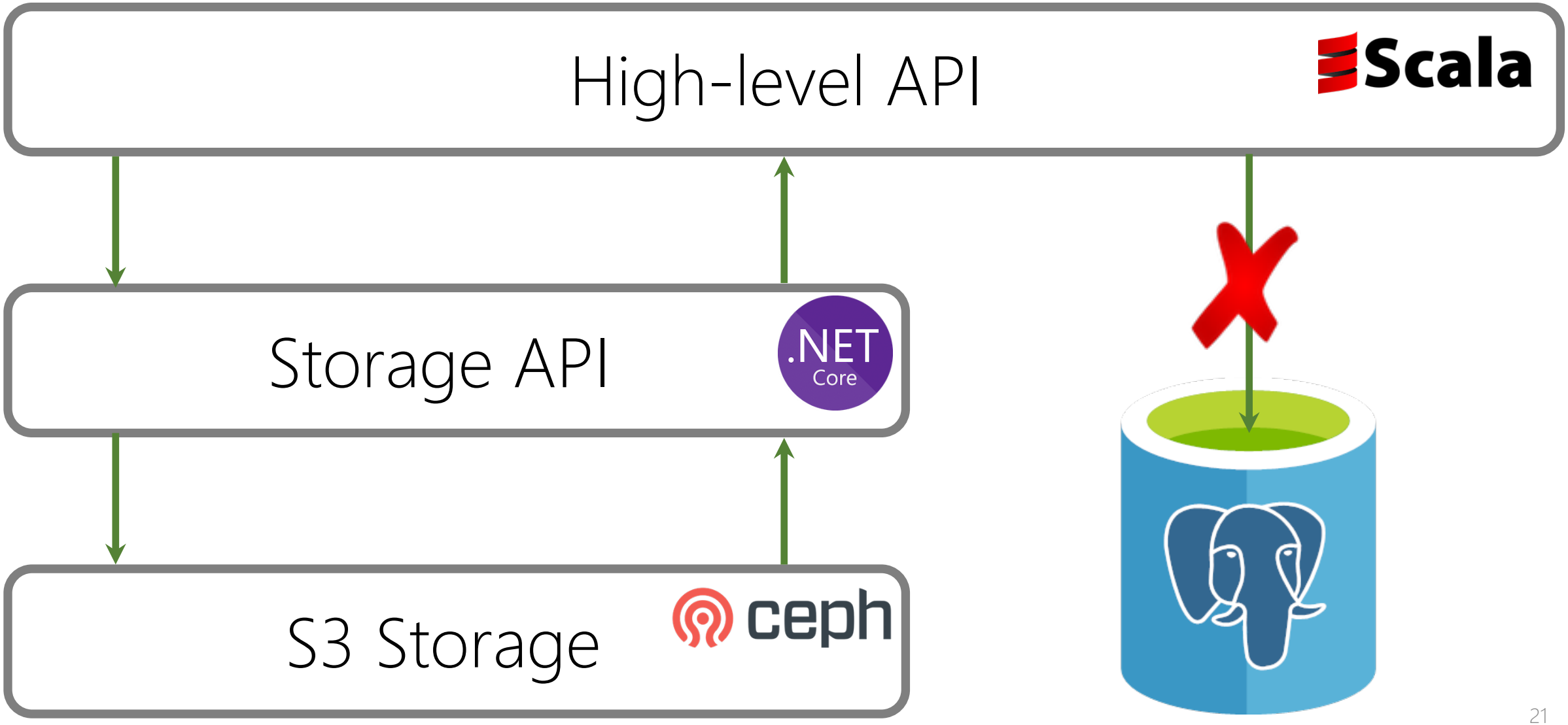
Распределенные изменения



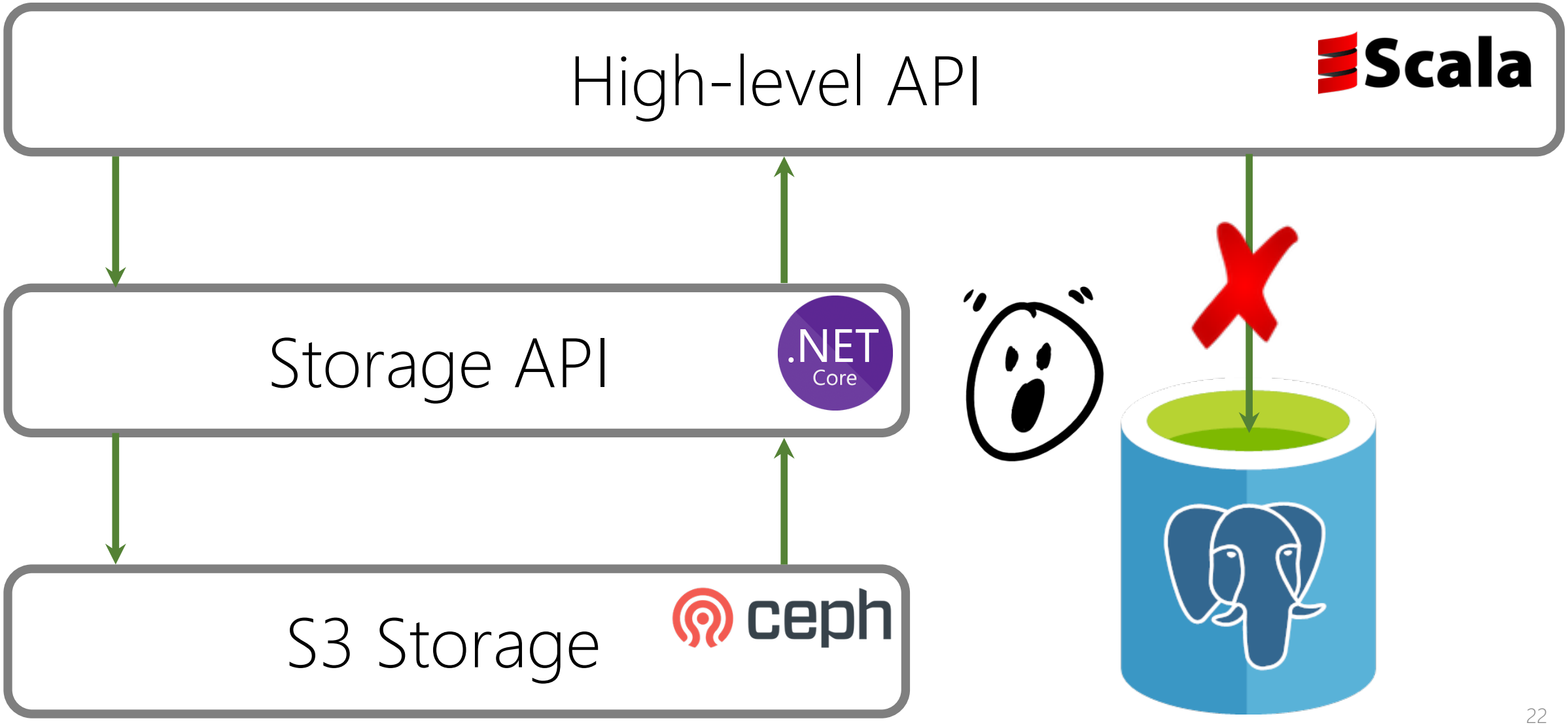
Распределенные изменения



Распределенные изменения

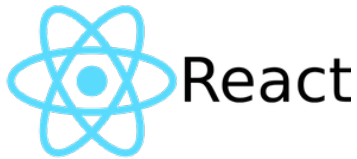


Распределенные изменения



Фоновые процессы

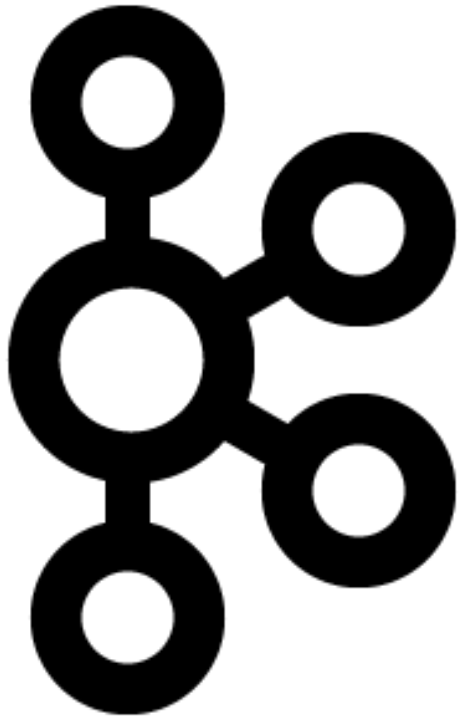
Frontend App



REST API

Storage

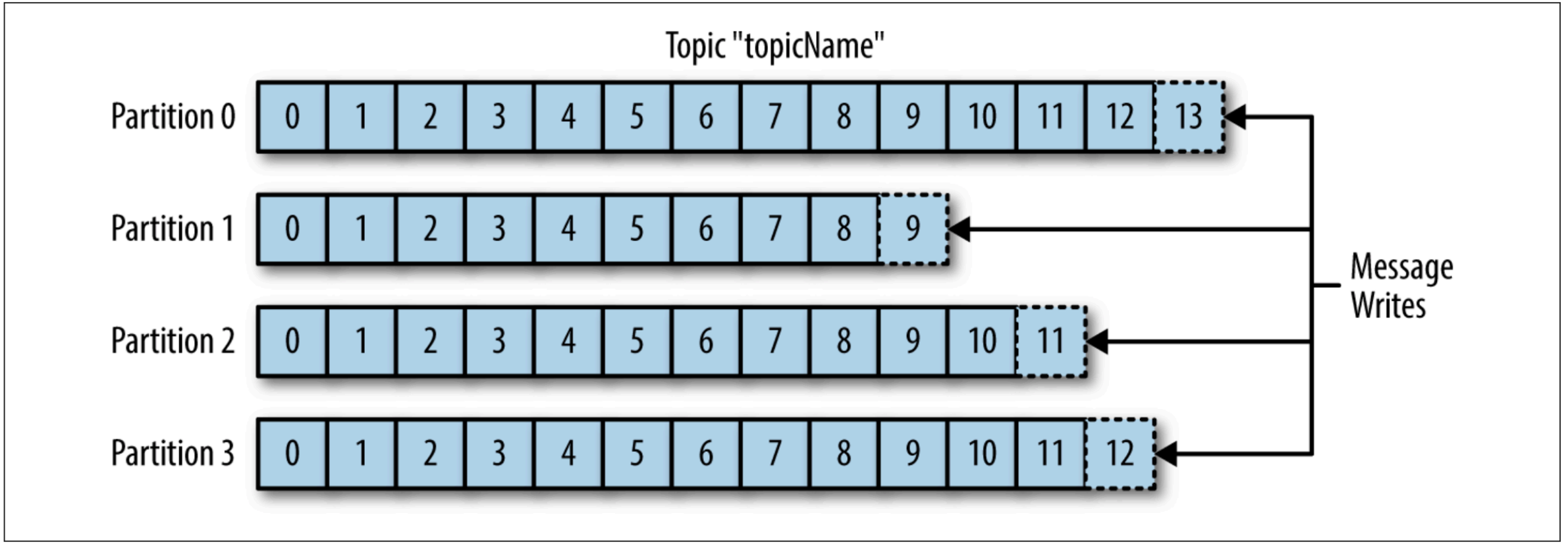
Background workers



APACHE kafka™

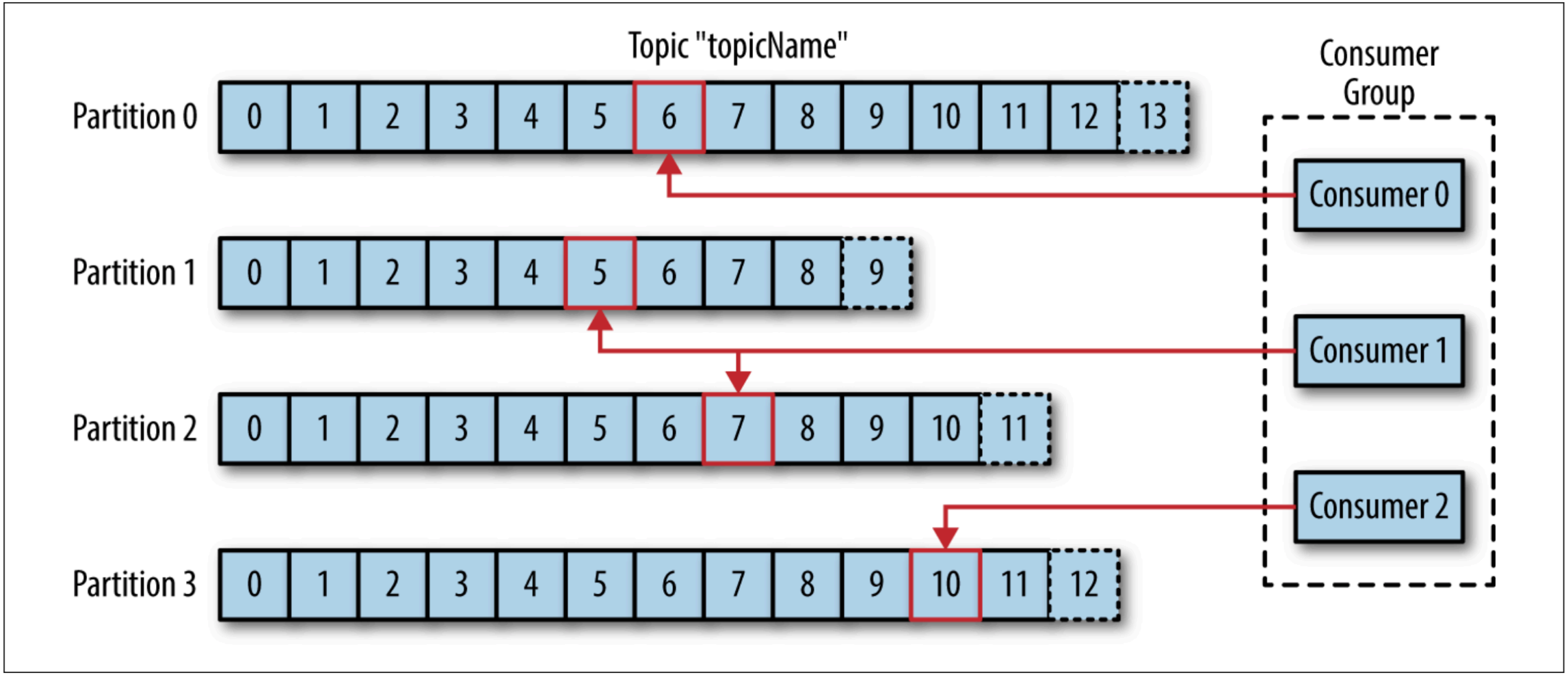
A distributed streaming platform

Topics and partitions



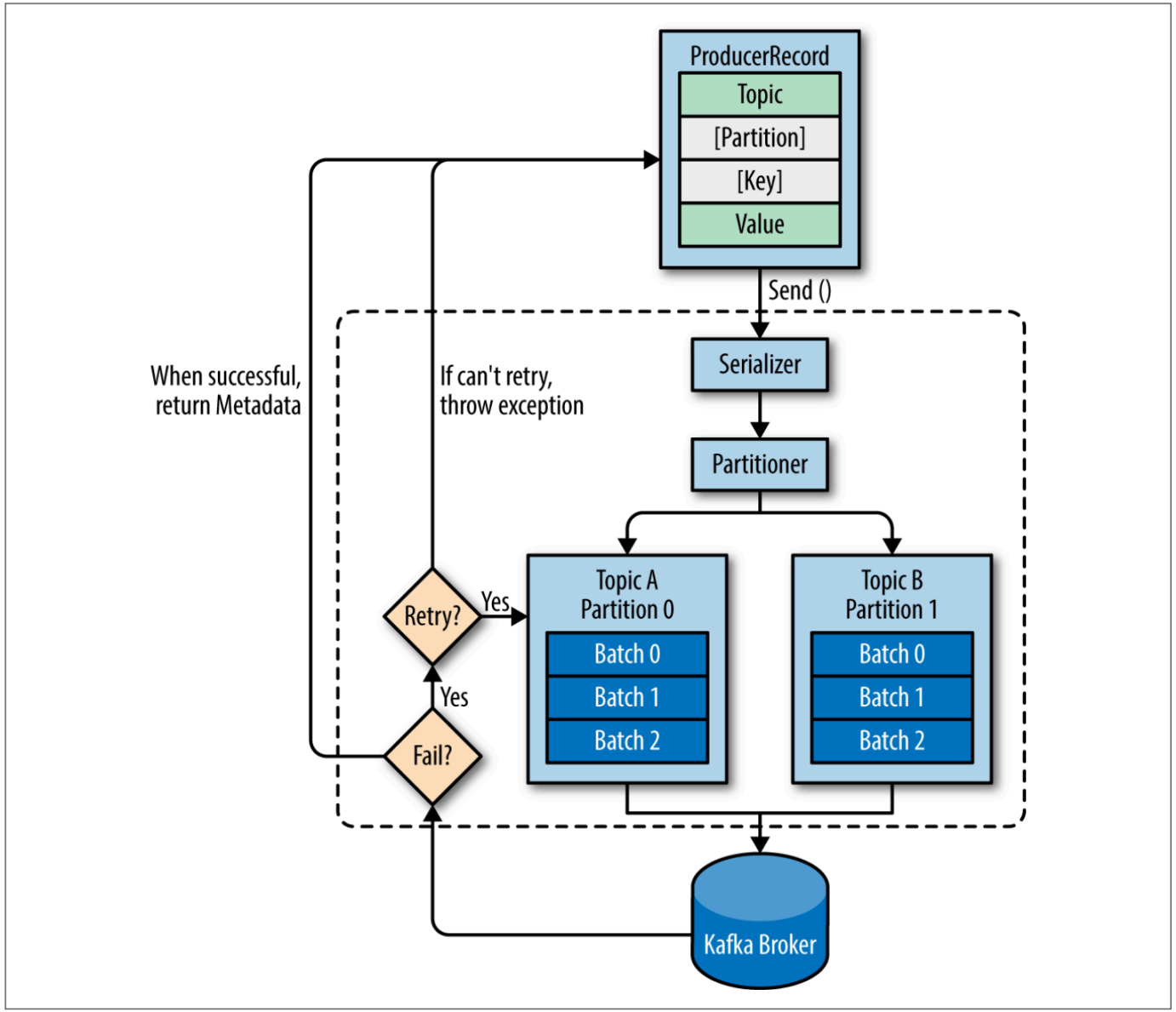
Kafka: The Definitive Guide by Neha Narkhede, Gwen Shapira, and Todd Palino (O'Reilly).

Producers and consumers



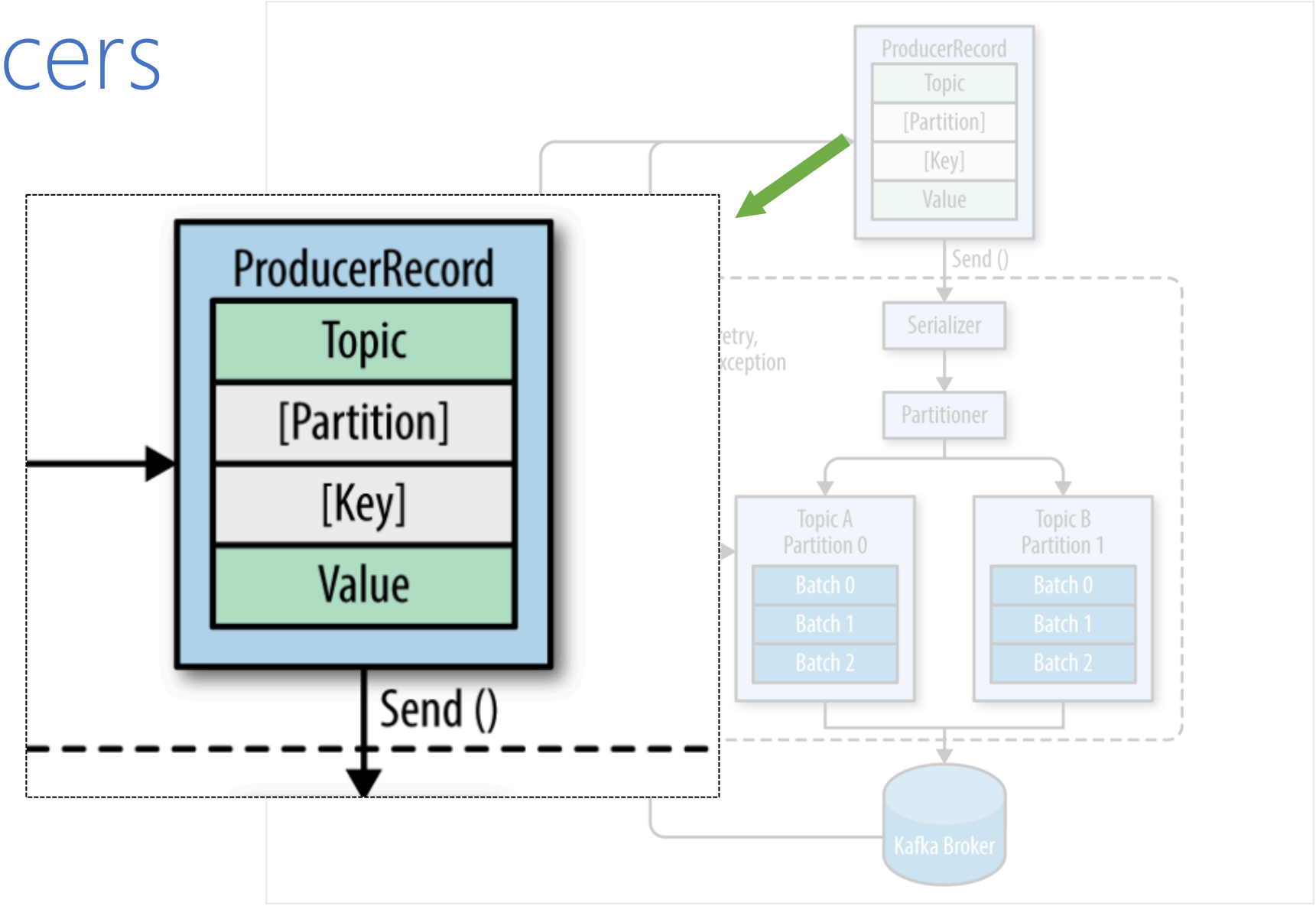
Kafka: The Definitive Guide by Neha Narkhede, Gwen Shapira, and Todd Palino (O'Reilly).

Producers



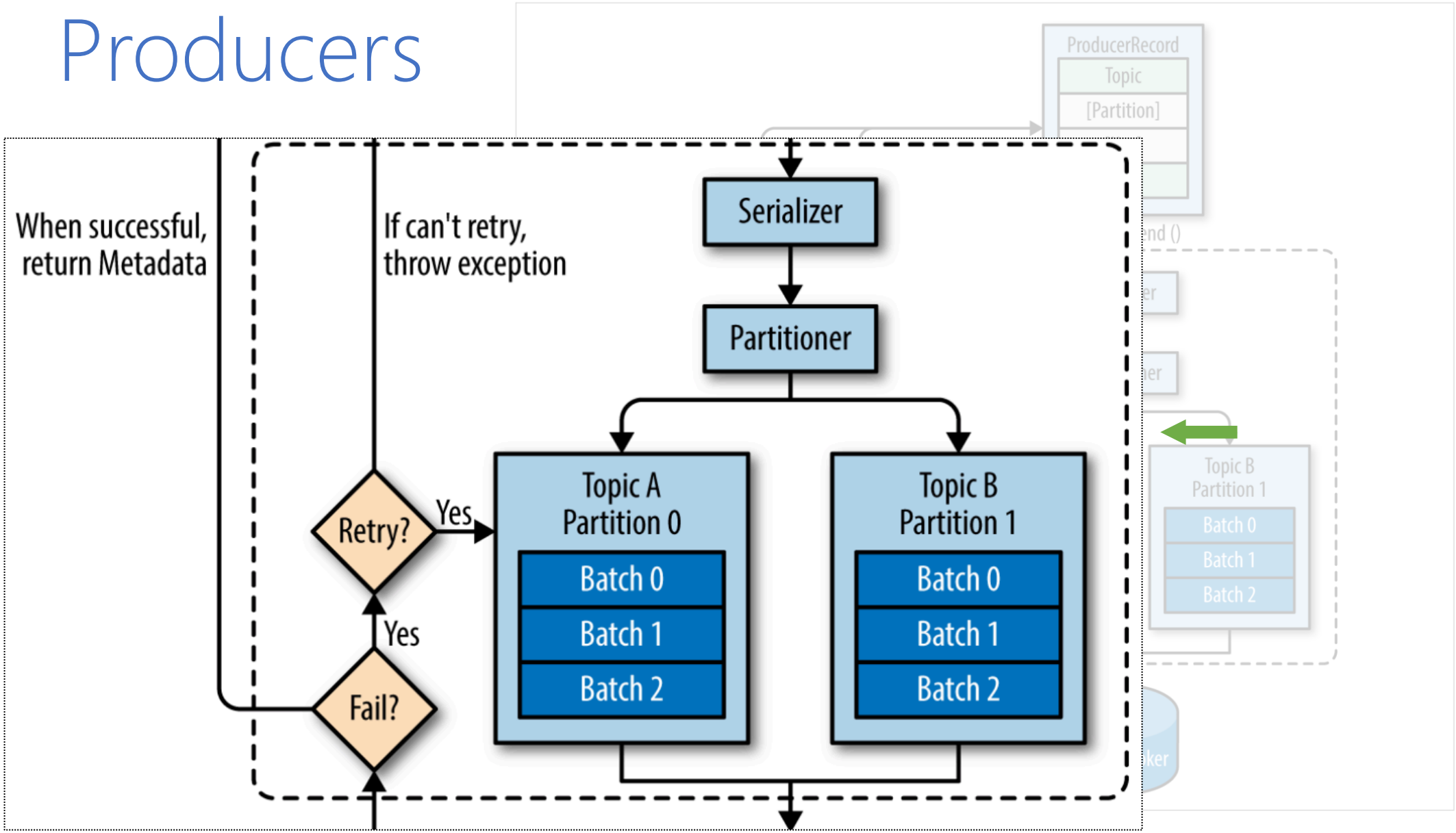
Kafka: The Definitive Guide by Neha Narkhede, Gwen Shapira, and Todd Palino (O'Reilly).

Producers



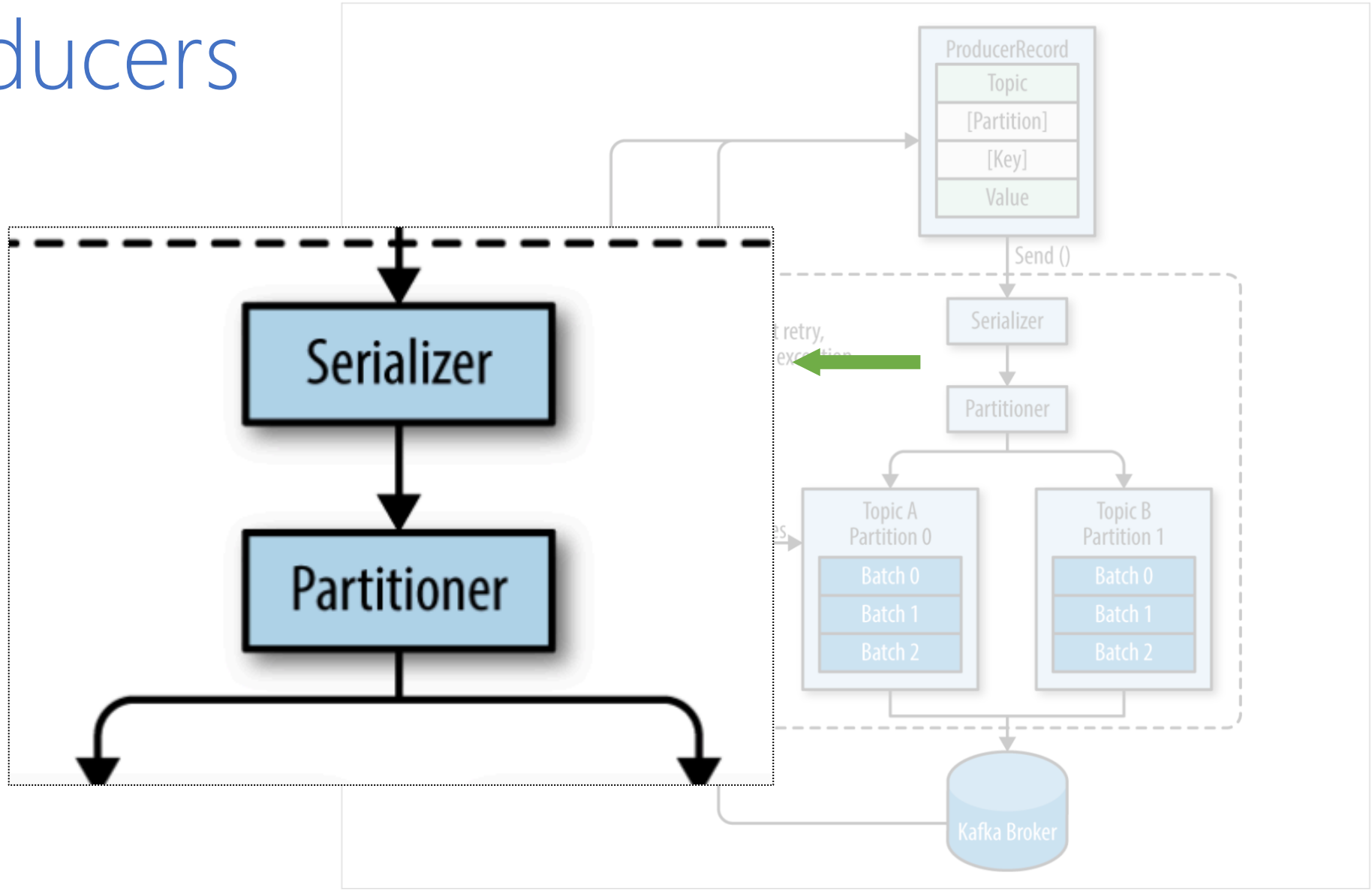
Kafka: The Definitive Guide by Neha Narkhede, Gwen Shapira, and Todd Palino (O'Reilly).

Producers



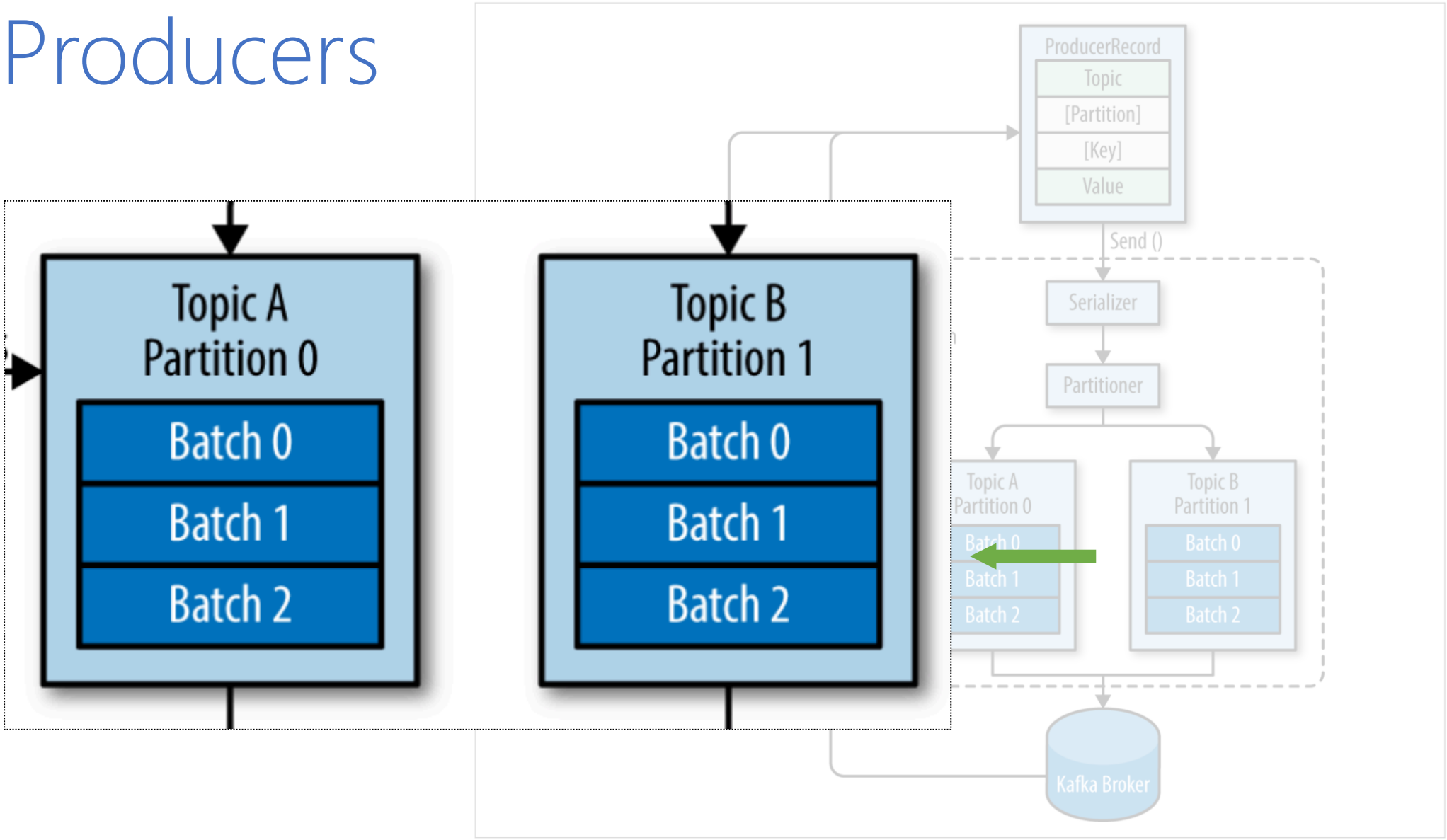
Kafka: The Definitive Guide by Neha Narkhede, Gwen Shapira, and Todd Palino (O'Reilly).

Producers



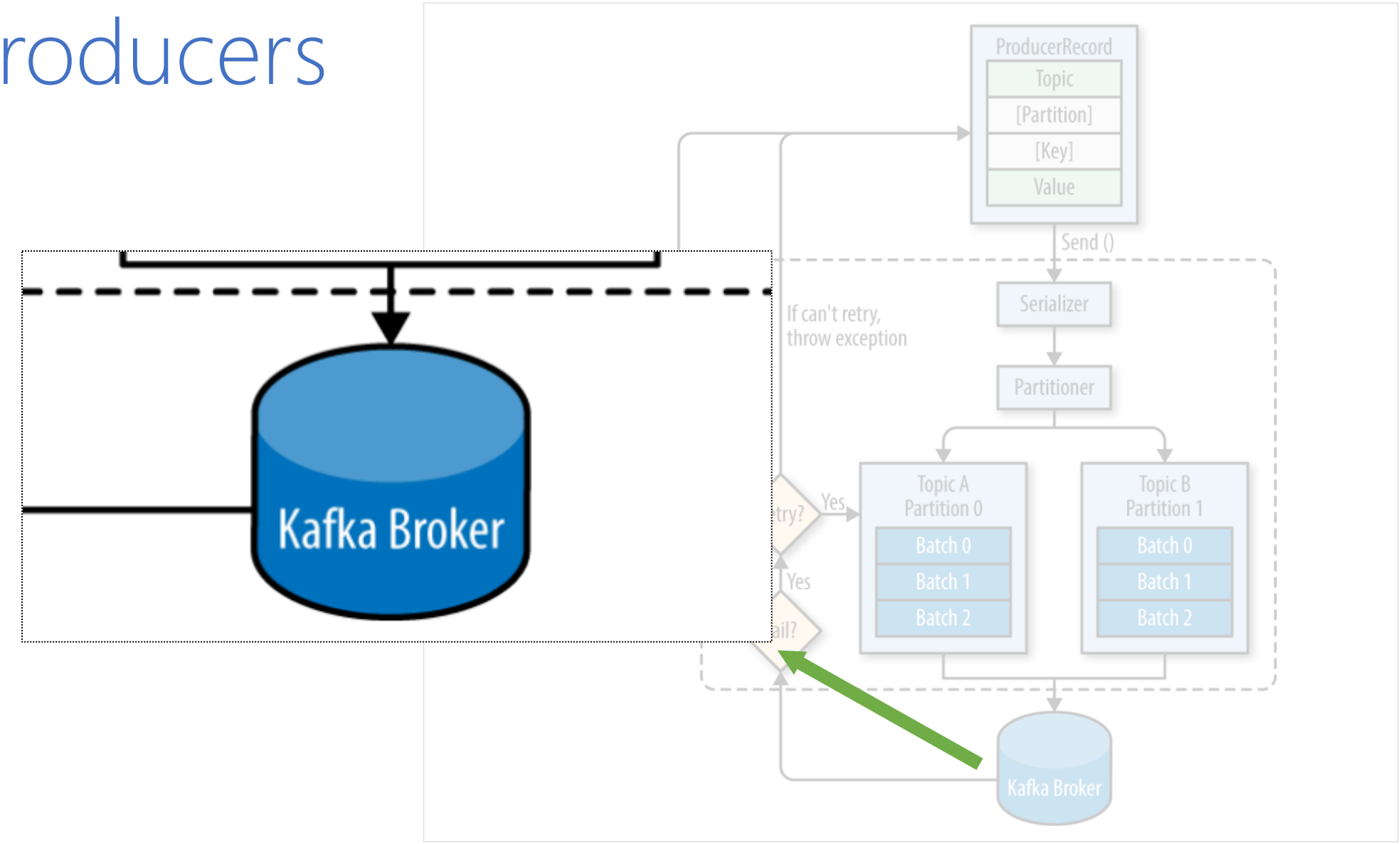
Kafka: The Definitive Guide by Neha Narkhede, Gwen Shapira, and Todd Palino (O'Reilly).

Producers



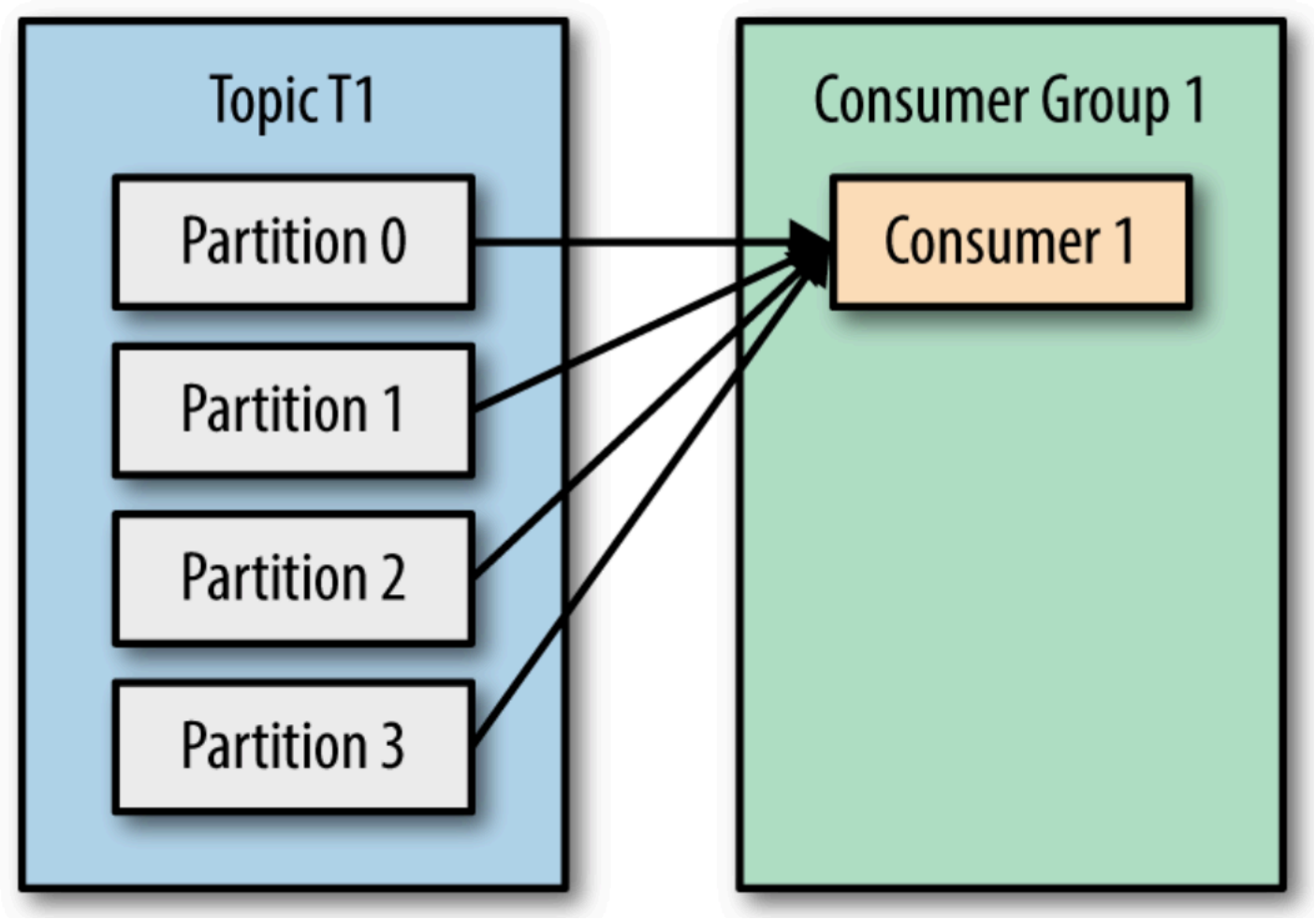
Kafka: The Definitive Guide by Neha Narkhede, Gwen Shapira, and Todd Palino (O'Reilly).

Producers



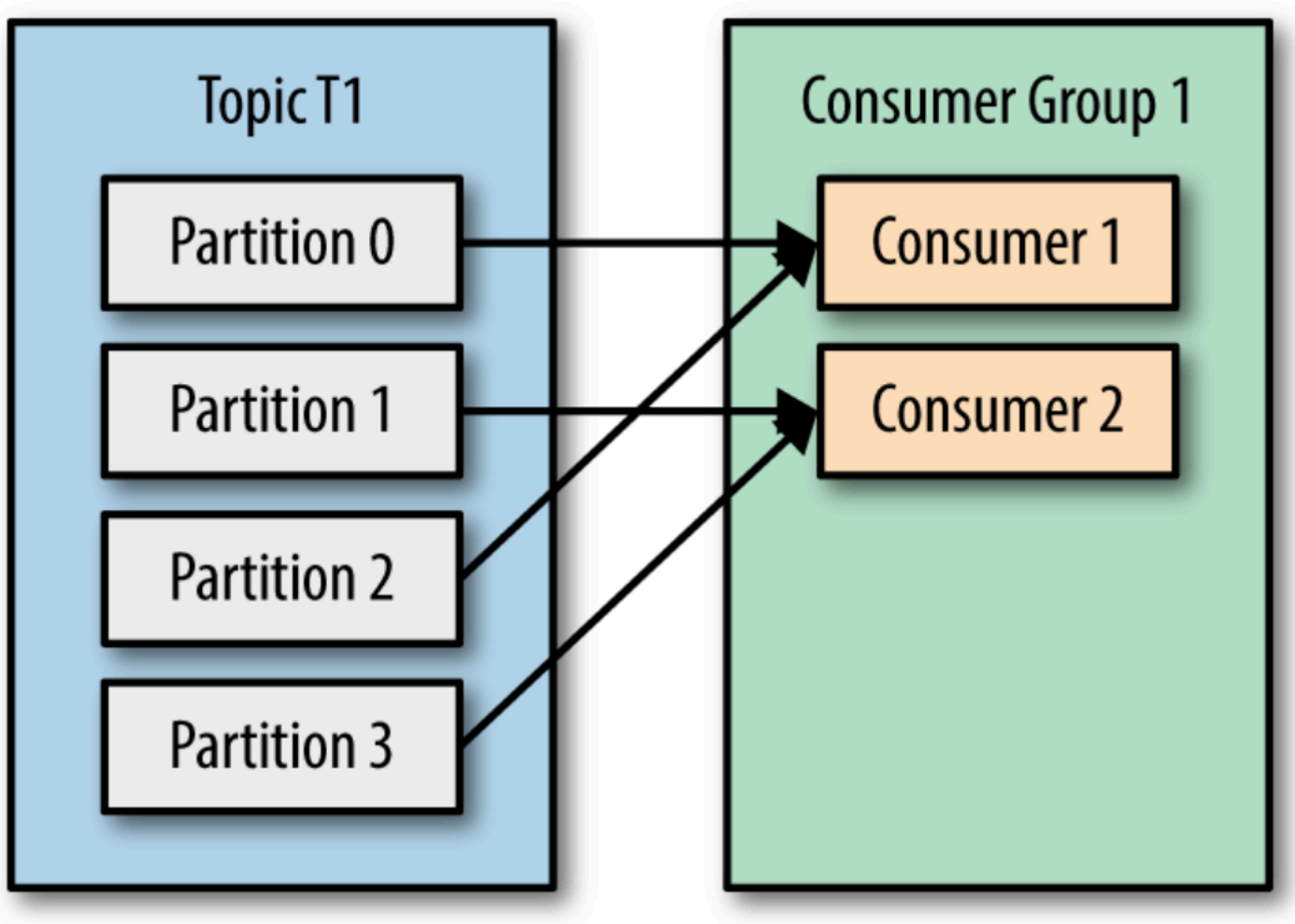
Kafka: The Definitive Guide by Neha Narkhede, Gwen Shapira, and Todd Palino (O'Reilly).

Consumers and consumer groups



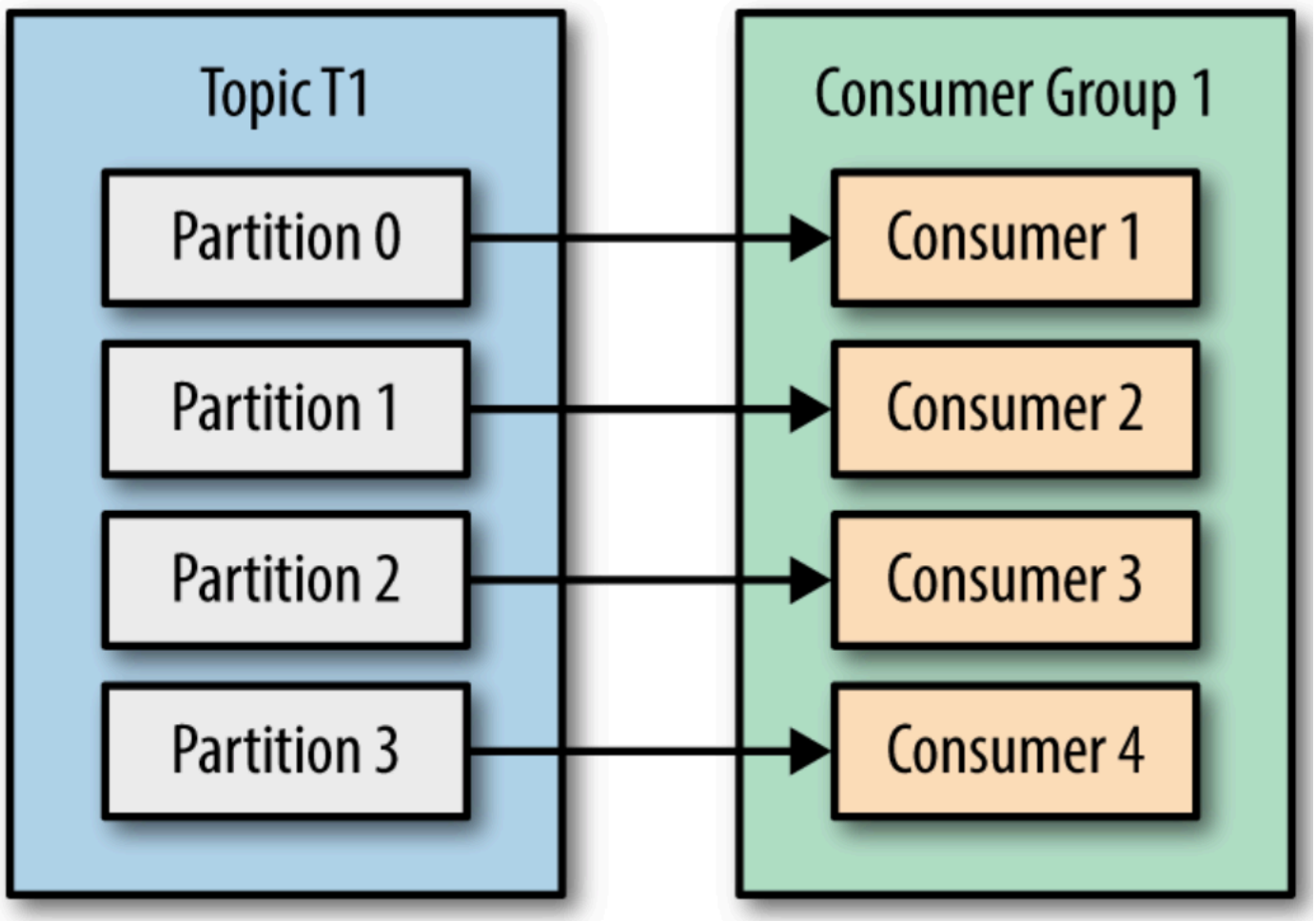
Kafka: The Definitive Guide by Neha Narkhede, Gwen Shapira, and Todd Palino (O'Reilly).

Consumers and consumer groups



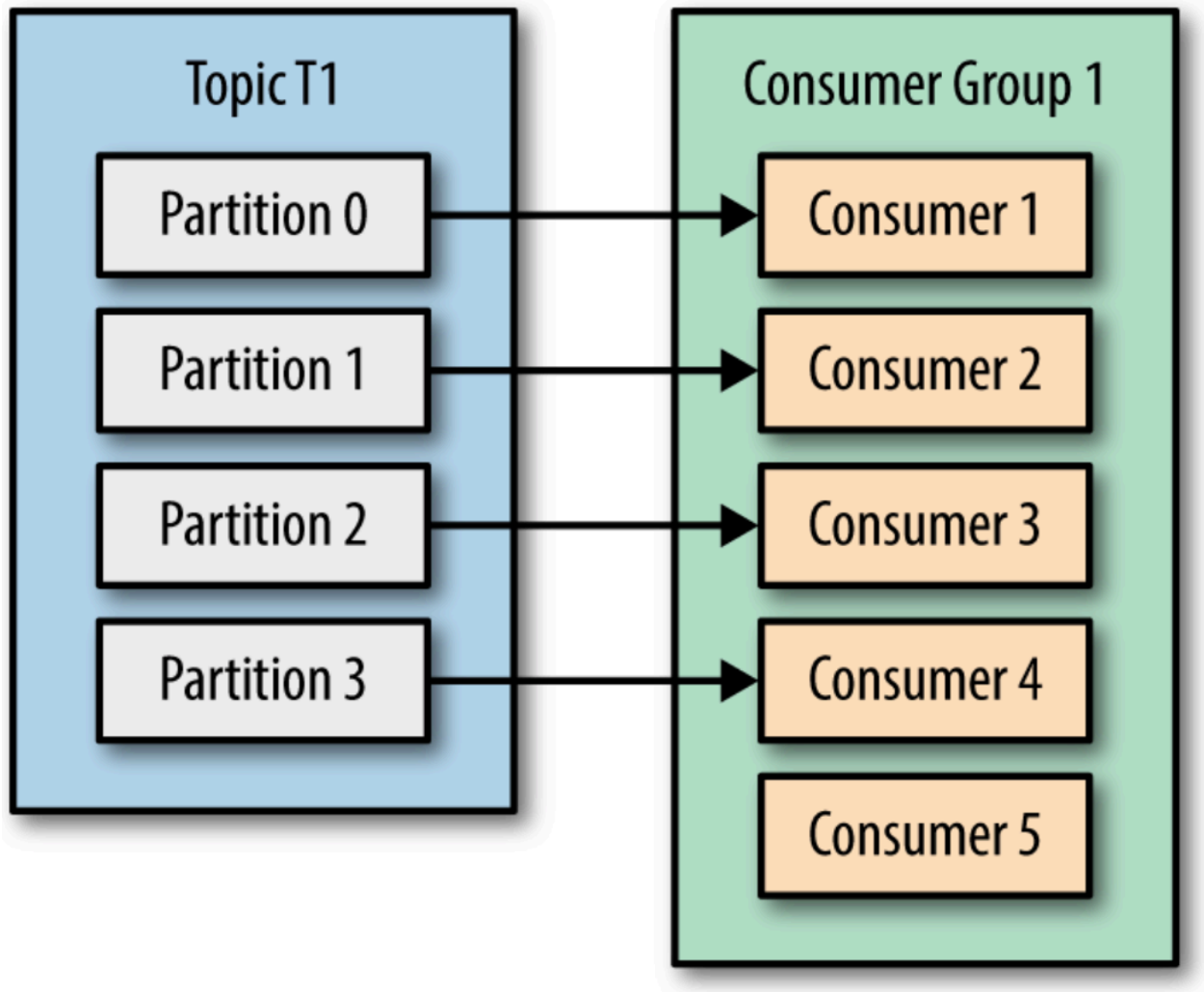
Kafka: The Definitive Guide by Neha Narkhede, Gwen Shapira, and Todd Palino (O'Reilly).

Consumers and consumer groups



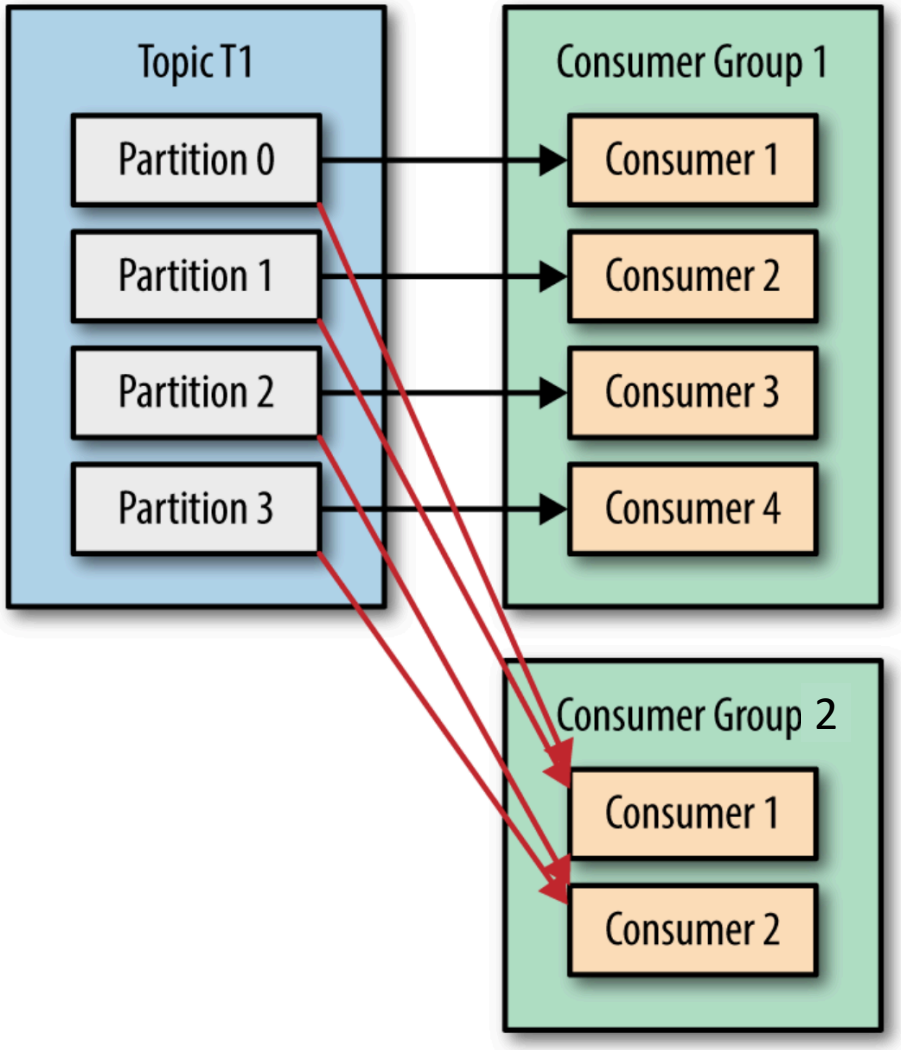
Kafka: The Definitive Guide by Neha Narkhede, Gwen Shapira, and Todd Palino (O'Reilly).

Consumers and consumer groups



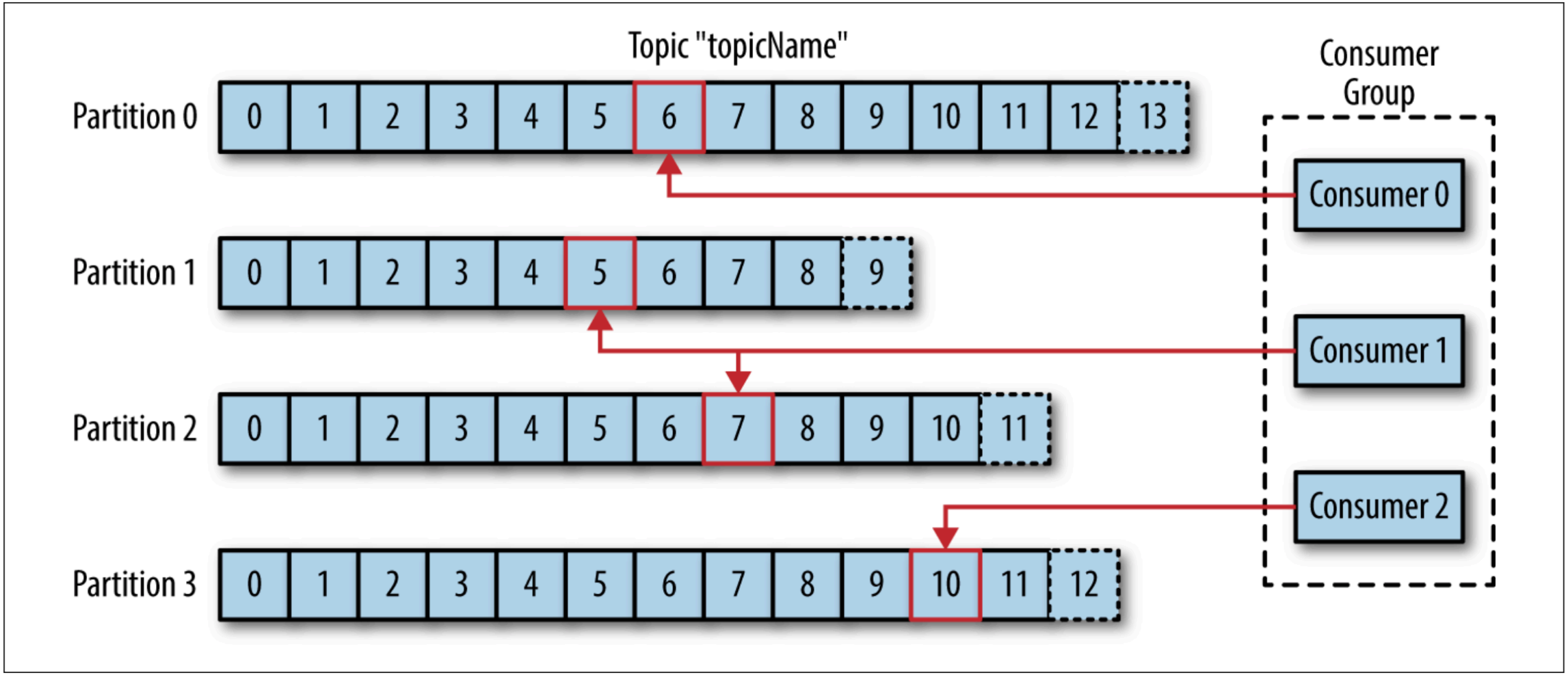
Kafka: The Definitive Guide by Neha Narkhede, Gwen Shapira, and Todd Palino (O'Reilly).

Consumers and consumer groups



Kafka: The Definitive Guide by Neha Narkhede, Gwen Shapira, and Todd Palino (O'Reilly).

Offsets and commits



Kafka: The Definitive Guide by Neha Narkhede, Gwen Shapira, and Todd Palino (O'Reilly).

Kafka .NET Client

- Обертка над C-библиотекой **`librdkafka`**
- Реализовано подмножество API (open source)
- Использует managed и unmanaged память и потоки
- xplat, .NET Core

Kafka .NET Producer

Kafka .NET Producer

```
var config = new Dictionary<string, object>
{
    { "bootstrap.servers", brokerEndpoints },
    { "api.version.request", true },
    {
      "default.topic.config",
      new Dictionary<string, object>
      {
          { "message.timeout.ms", 5000 },
          { "request.required.acks", -1 }
      }
    }
};
```

Kafka .NET Producer

```
var config = new Dictionary<string, object>
{
    { "bootstrap.servers", brokerEndpoints },
    { "api.version.request", true },
    {
      "default.topic.config",
      new Dictionary<string, object>
      {
          { "message.timeout.ms", 5000 },
          { "request.required.acks", -1 }
      }
    }
};

var producer = new Producer<Null, string>(
        config, new NullSerializer(), new StringSerializer(Encoding.UTF8));
```

Kafka .NET Producer

```
var config = new Dictionary<string, object>
{
    { "bootstrap.servers", brokerEndpoints },
    { "api.version.request", true },
    {
      "default.topic.config",
      new Dictionary<string, object>
      {
          { "message.timeout.ms", 5000 },
          { "request.required.acks", -1 }
      }
    }
};

var producer = new Producer<Null, string>(
        config, new NullSerializer(), new StringSerializer(Encoding.UTF8));
```

Kafka .NET Producer

```
var config = new Dictionary<string, object>
{
    { "bootstrap.servers", brokerEndpoints },
    { "api.version.request", true },
    {
      "default.topic.config",
      new Dictionary<string, object>
      {
          { "message.timeout.ms", 5000 },
          { "request.required.acks", -1 }
      }
    }
};

var producer = new Producer<Null, string>(
    config, new NullSerializer(), new StringSerializer(Encoding.UTF8));
```

Kafka .NET Producer

```
var config = new Dictionary<string, object>
{
    { "bootstrap.servers", brokerEndpoints },
    { "api.version.request", true },
    {
      "default.topic.config",
      new Dictionary<string, object>
      {
          { "message.timeout.ms", 5000 },
          { "request.required.acks", -1 }
      }
    }
};

var producer = new Producer<Null, string>(
        config, new NullSerializer(), new StringSerializer(Encoding.UTF8));

var message = await producer.ProduceAsync(topic, null, value, partition);
```

Kafka .NET Producer

```
var config = new Dictionary<string, object>
{
    { "bootstrap.servers", brokerEndpoints },
    { "api.version.request", true },
    {
      "default.topic.config",
      new Dictionary<string, object>
      {
          { "message.timeout.ms", 5000 },
          { "request.required.acks", -1 }
      }
    }
};

var producer = new Producer<Null, string>(
        config, new NullSerializer(), new StringSerializer(Encoding.UTF8));

var message = await producer.ProduceAsync(topic, null, value, partition);
```

Kafka .NET Producer

```
var config = new Dictionary<string, object>
{
    { "bootstrap.servers", brokerEndpoints },
    { "api.version.request", true },
    {
      "default.topic.config",
      new Dictionary<string, object>
      {
          { "message.timeout.ms", 5000 },
          { "request.required.acks", -1 }
      }
    }
};

var producer = new Producer<Null, string>(
        config, new NullSerializer(), new StringSerializer(Encoding.UTF8));

var message = await producer.ProduceAsync(topic, null, value, partition);
```

Kafka .NET Consumer

Kafka .NET Consumer

```
var config = new Dictionary<string, object>
{
    { "bootstrap.servers", brokerEndpoints },
    { "api.version.request", true },
    { "group.id", !string.IsNullOrEmpty(groupId)
                    ? groupId
                    : Guid.NewGuid().ToString() },
    { "enable.auto.commit", false },
    {
      "default.topic.config",
      new Dictionary<string, object>
      {
          { "auto.offset.reset", "beginning" }
      }
    }
};
```

Kafka .NET Consumer

```
var config = new Dictionary<string, object>
{
    { "bootstrap.servers", brokerEndpoints },
    { "api.version.request", true },
    { "group.id", !string.IsNullOrEmpty(groupId)
                    ? groupId
                    : Guid.NewGuid().ToString() },
    { "enable.auto.commit", false },
    {
        "default.topic.config",
        new Dictionary<string, object>
        {
            { "auto.offset.reset", "beginning" }
        }
    }
};
```

Kafka .NET Consumer

```
var config = new Dictionary<string, object>
{
    { "bootstrap.servers", brokerEndpoints },
    { "api.version.request", true },
    { "group.id", !string.IsNullOrEmpty(groupId)
                      ? groupId
                      : Guid.NewGuid().ToString() },
    { "enable.auto.commit", false },
    {
        "default.topic.config",
        new Dictionary<string, object>
        {
            { "auto.offset.reset", "beginning" }
        }
    }
};
```

Kafka .NET Consumer

```
var config = new Dictionary<string, object>
{
    { "bootstrap.servers", brokerEndpoints },
    { "api.version.request", true },
    { "group.id", !string.IsNullOrEmpty(groupId)
                    ? groupId
                    : Guid.NewGuid().ToString() },
    { "enable.auto.commit", false },
    {
        "default.topic.config",
        new Dictionary<string, object>
        {
            { "auto.offset.reset", "beginning" }
        }
    }
};
```

Kafka .NET Consumer

```
var config = new Dictionary<string, object>
{
    { "bootstrap.servers", brokerEndpoints },
    { "api.version.request", true },
    { "group.id", !string.IsNullOrEmpty(groupId)
                    ? groupId
                    : Guid.NewGuid().ToString() },
    { "enable.auto.commit", false },
    {
        "default.topic.config",
        new Dictionary<string, object>
        {
            { "auto.offset.reset", "beginning" }
        }
    }
};
var consumer = new Consumer<Null, string>(
    config, new NullDeserializer(), new StringDeserializer(Encoding.UTF8));
```

Batched Kafka Consumer Wrapper

```
var consumer = new Consumer<Null, string>(
    config, new NullDeserializer(), new StringDeserializer(Encoding.UTF8));
```

```
var consumer = new Consumer<Null, string>(
    config, new NullDeserializer(), new StringDeserializer(Encoding.UTF8));

var messages = new List<Message<Null, string>>();
var isEof = false;
```

```
var consumer = new Consumer<Null, string>(
   config, new NullDeserializer(), new StringDeserializer(Encoding.UTF8));

var messages = new List<Message<Null, string>>();
var isEof = false;

void OnMessage(object sender, Message<Null, string> message) =>
   messages.Add(message);

void OnEof(object sender, TopicPartitionOffset offset) => isEof = true;
```

```
var consumer = new Consumer<Null, string>(
   config, new NullDeserializer(), new StringDeserializer(Encoding.UTF8));

var messages = new List<Message<Null, string>>();
var isEof = false;

void OnMessage(object sender, Message<Null, string> message) =>
   messages.Add(message);

void OnEof(object sender, TopicPartitionOffset offset) => isEof = true;

consumer.OnMessage += OnMessage;
consumer.OnPartitionEOF += OnEof;
consumer.Subscribe(topics);
```

```
var consumer = new Consumer<Null, string>(
   config, new NullDeserializer(), new StringDeserializer(Encoding.UTF8));

var messages = new List<Message<Null, string>>();
var isEof = false;

void OnMessage(object sender, Message<Null, string> message) =>
   messages.Add(message);

void OnEof(object sender, TopicPartitionOffset offset) => isEof = true;

consumer.OnMessage += OnMessage;
consumer.OnPartitionEOF += OnEof;
consumer.Subscribe(topics);

while (messages.Count < batchSize && !isEof &&
       !cancellationToken.IsCancellationRequested)
{
    consumer.Poll(TimeSpan.FromMilliseconds(100));
}
```

```
var consumer = new Consumer<Null, string>(
   config, new NullDeserializer(), new StringDeserializer(Encoding.UTF8));

var messages = new List<Message<Null, string>>();
var isEof = false;

void OnMessage(object sender, Message<Null, string> message) =>
   messages.Add(message);

void OnEof(object sender, TopicPartitionOffset offset) => isEof = true;

consumer.OnMessage += OnMessage;
consumer.OnPartitionEOF += OnEof;
consumer.Subscribe(topics);

while (messages.Count < batchSize && !isEof &&
      !cancellationToken.IsCancellationRequested)
{
    consumer.Poll(TimeSpan.FromMilliseconds(100));
}

consumer.Unsubscribe(topics);
```

```
var consumerWrapper = new ConsumerWrapper(brokerEndpoints, groupId);
```

```
var consumerWrapper = new ConsumerWrapper(brokerEndpoints, groupId);
```

```
var consumerWrapper = new ConsumerWrapper(brokerEndpoints, groupId);
while (!cancellationToken.IsCancellationRequested)
{
    var messages = consumerWrapper.Consume(
                        topics, batchSize, cancellationToken);

}
```

```
var consumerWrapper = new ConsumerWrapper(brokerEndpoints, groupId);
while (!cancellationToken.IsCancellationRequested)
{
    var messages = consumerWrapper.Consume(
                    topics, batchSize, cancellationToken);

}
```

```
var consumerWrapper = new ConsumerWrapper(brokerEndpoints, groupId);
while (!cancellationToken.IsCancellationRequested)
{
    var messages = consumerWrapper.Consume(
                        topics, batchSize, cancellationToken);
    foreach (var message in messages)
    {
        Console.WriteLine(
           $"{message.Topic}/{message.Partition} @" +
           $"{message.Offset}: '{message.Value}'");
    }
}
```

```
var consumerWrapper = new ConsumerWrapper(brokerEndpoints, groupId);
while (!cancellationToken.IsCancellationRequested)
{
    var messages = consumerWrapper.Consume(
                        topics, batchSize, cancellationToken);
    foreach (var message in messages)
    {
        Console.WriteLine(
           $"{message.Topic}/{message.Partition} @" +
           $"{message.Offset}: '{message.Value}'");

        await consumerWrapper.CommitAsync(message);
    }
}
```

Kafka + Docker + .NET Core

More challenges

- Легко написать неправильно

More challenges

- Легко написать неправильно
- Частые перебалансировки consumer group

More challenges

- Легко написать неправильно
- Частые перебалансировки consumer group
- Управление распределенными consumer-ами

More challenges

- Легко написать неправильно
- Частые перебалансировки consumer group
- Управление распределенными consumer-ами
- Задержки, синхронизация, утилизация ресурсов

Reactive Extensions (Rx)

Pull-модель

```
public interface IEnumerable
{
    IEnumerator GetEnumerator();
}


public interface IEnumerator
{
    bool MoveNext();

    object Current { get; }

    void Reset();
}
```

Push-модель

```
public interface IObservable<out T>
{
    IDisposable Subscribe(IObserver<T> observer);
}


public interface IObserver<in T>
{
    void OnCompleted();

    void OnError(Exception error);

    void OnNext(T value);
}
```

Rx Kafka Consumer Wrapper

```
…

void OnMessage(object sender, Message<Null, string> message) =>
    messages.Add(message);

void OnEof(object sender, TopicPartitionOffset offset) => isEof = true;


consumer.OnMessage += OnMessage;
consumer.OnPartitionEOF += OnEof;
consumer.Subscribe(topics);


…
```

```
IEnumerable<string> topics;
var observable =
   Observable.FromEventPattern<Message<Null, string>>(
      x =>
         {
            consumer.OnMessage += x;
            consumer.Subscribe(topics);
         },
      x =>
         {
            consumer.Unsubscribe();
            consumer.OnMessage -= x;
         })
   .Select(x => x.EventArgs);
```

```
IEnumerable<string> topics;
var observable =
   Observable.FromEventPattern<Message<Null, string>>(
      x =>
         {
            consumer.OnMessage += x;
            consumer.Subscribe(topics);
         },
      x =>
         {
            consumer.Unsubscribe();
            consumer.OnMessage -= x;
         })
   .Select(x => x.EventArgs);
```

```
IEnumerable<string> topics;
var observable =
   Observable.FromEventPattern<Message<Null, string>>(
      x =>
         {
            consumer.OnMessage += x;
            consumer.Subscribe(topics);
         },
      x =>
         {
            consumer.Unsubscribe();
            consumer.OnMessage -= x;
         })
   .Select(x => x.EventArgs);
```

```
IEnumerable<string> topics;
var observable =
    Observable.FromEventPatter<Message<Null, string>>
        x =>
            {
                consumer.OnMessage += x;
                consumer.Subscribe(topics);
            },
        x =>
            {
                consumer.Unsubscribe();
                consumer.OnMessage -= x;
            })
    .Select(x => x.EventArgs);
```

```
IEnumerable<string> topics;
var observable =
   Observable.FromEventPattern<Message<Null, string>>(…);
Task.Factory.StartNew(
   () =>
      {
         while (!cancellationToken.IsCancellationRequested)
         {
            consumer.Poll(TimeSpan.FromMilliseconds(100));
         }
      },
   cancellationToken,
   TaskCreationOptions.LongRunning,
   TaskScheduler.Default);
```

```
var observable = consumerWrapper.Consume(token);
```

```
var observable = consumerWrapper.Consume(token);
```

```
var observable = consumerWrapper.Consume(token);

var subscription = observable
    .Buffer(batchSize)
    .Subscribe(
       messages =>
       {
          foreach (var message in messages)
          {
             Console.WriteLine(message.Value);
             consumerWrapper.CommitAsync(message)
                .GetAwaiter().GetResult();
          }
       });
```

```
var observable = consumerWrapper.Consume(token);

var subscription = observable
    .Buffer(batchSize)
    .Subscribe(
        messages =>
        {
            foreach (var message in messages)
            {
                Console.WriteLine(message.Value);
                consumerWrapper.CommitAsync(message)
                    .GetAwaiter().GetResult();
            }
        });
```

```
var observable = consumerWrapper.Consume(token);

var subscription = observable
    .Buffer(batchSize)
    .Subscribe(
        messages =>
        {
            foreach (var message in messages)
            {
                Console.WriteLine(message.Value);
                consumerWrapper.CommitAsync(message)
                    .GetAwaiter().GetResult();
            }
        });
```

```
var observable = consumerWrapper.Consume(token);

var subscription = observable
    .Buffer(batchSize)
    .Subscribe(…);

var taskCompletionSource = new TaskCompletionSource<object>();
cancellationToken.Register(
   () =>
      {
         subscription.Dispose();
         taskCompletionSource.SetResult(null);
      });
```

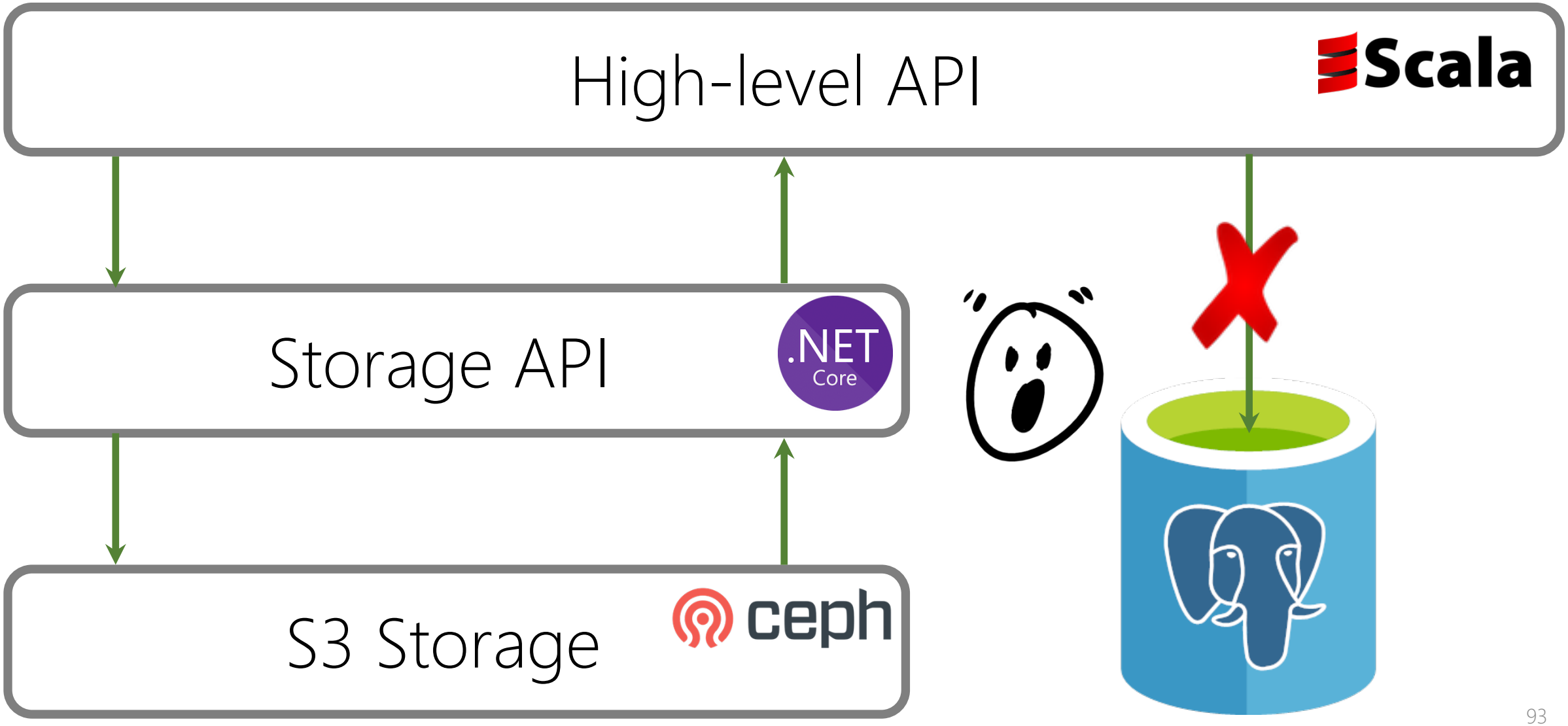
```
var observable = consumerWrapper.Consume(token);

var subscription = observable
    .Buffer(batchSize)
    .Subscribe(…);

var taskCompletionSource = new TaskCompletionSource<object>();
cancellationToken.Register(
   () =>
      {
         subscription.Dispose();
         taskCompletionSource.SetResult(null);
      });
```

```
var observable = consumerWrapper.Consume(token);

var subscription = observable
    .Buffer(batchSize)
    .Subscribe(…);

var taskCompletionSource = new TaskCompletionSource<object>();
cancellationToken.Register(
   () =>
      {
         subscription.Dispose();
         taskCompletionSource.SetResult(null);
      });
```

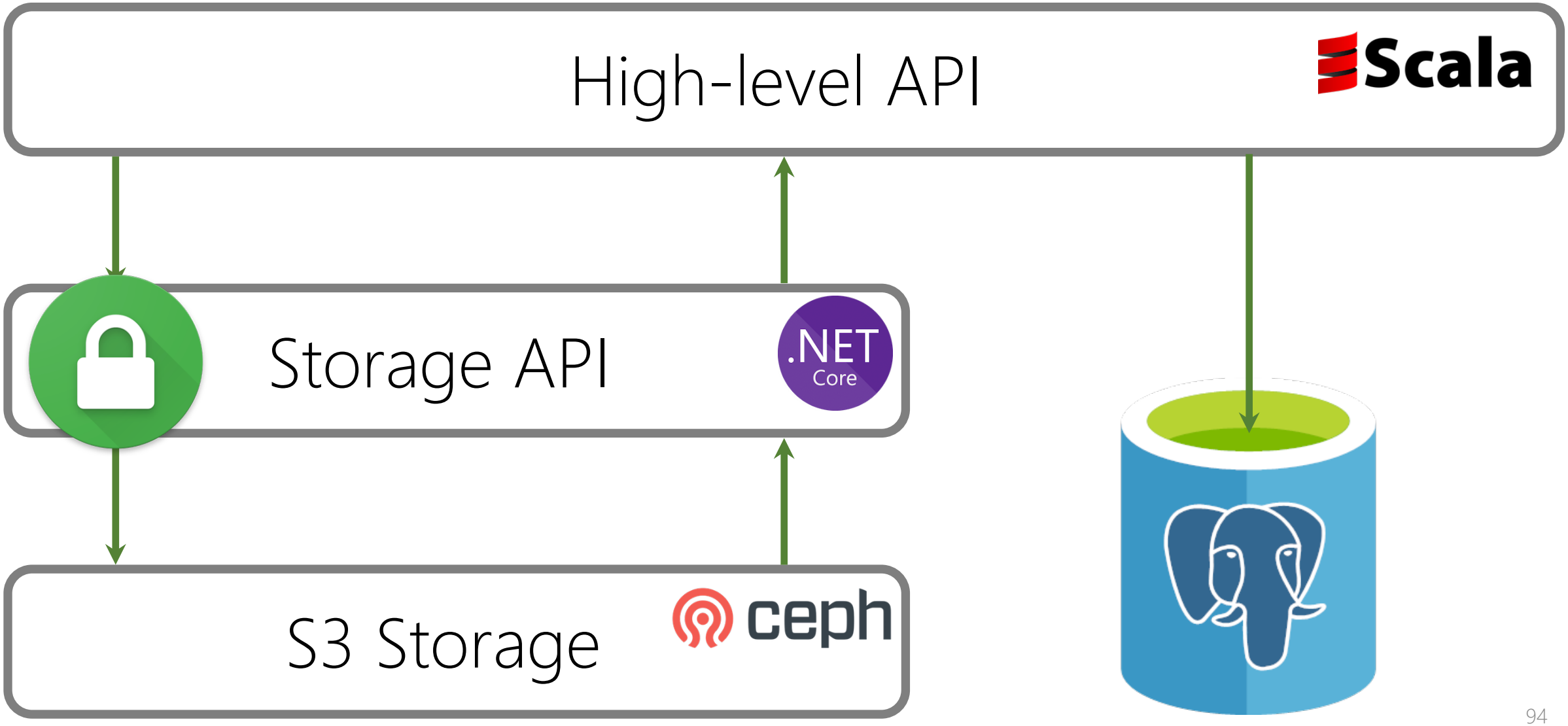
```
var observable = consumerWrapper.Consume(token);

var subscription = observable
    .Buffer(batchSize)
    .Subscribe(…);

var taskCompletionSource = new TaskCompletionSource<object>();
cancellationToken.Register(
   () =>
      {
         subscription.Dispose();
         taskCompletionSource.SetResult(null);
      });

await taskCompletionSource.Task;
```

Kafka + Rx

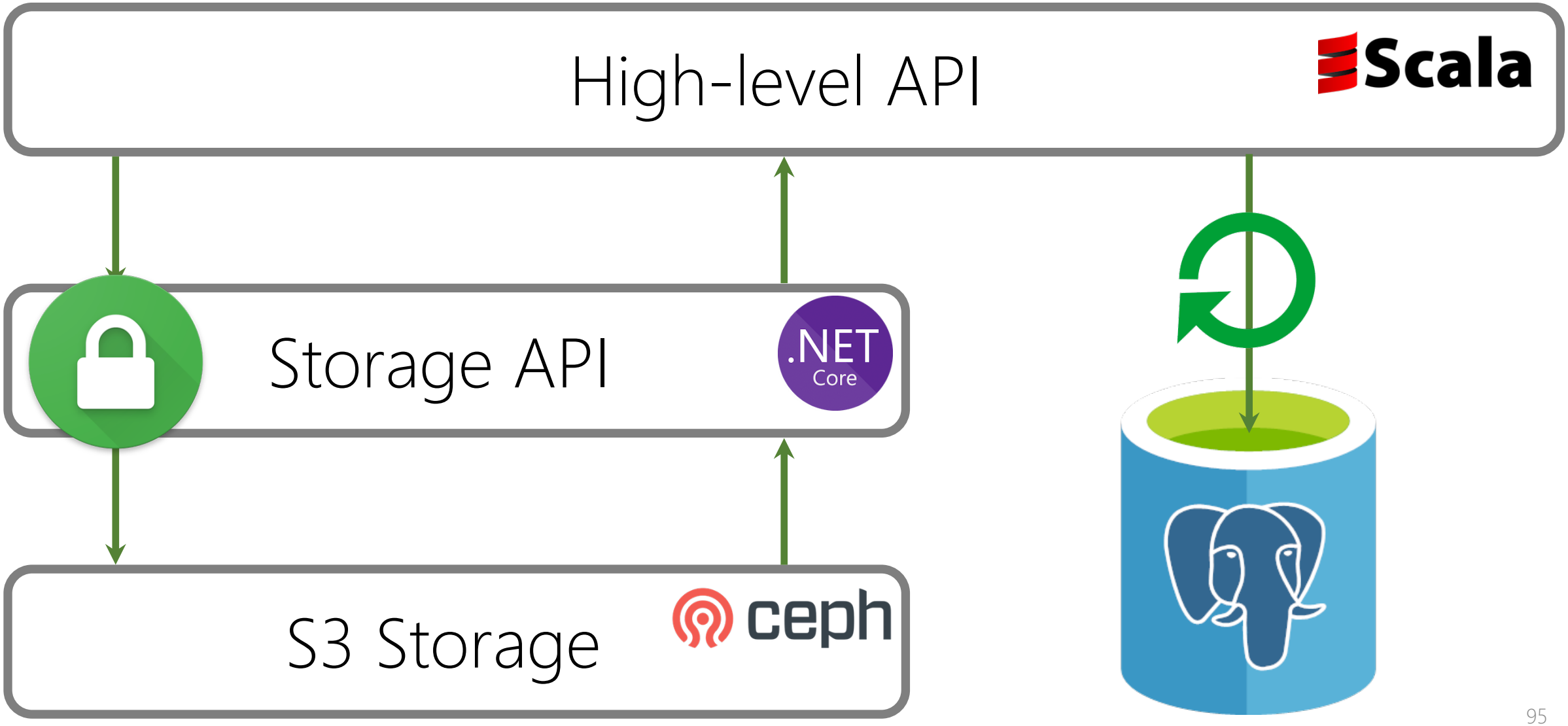
Распределенные изменения



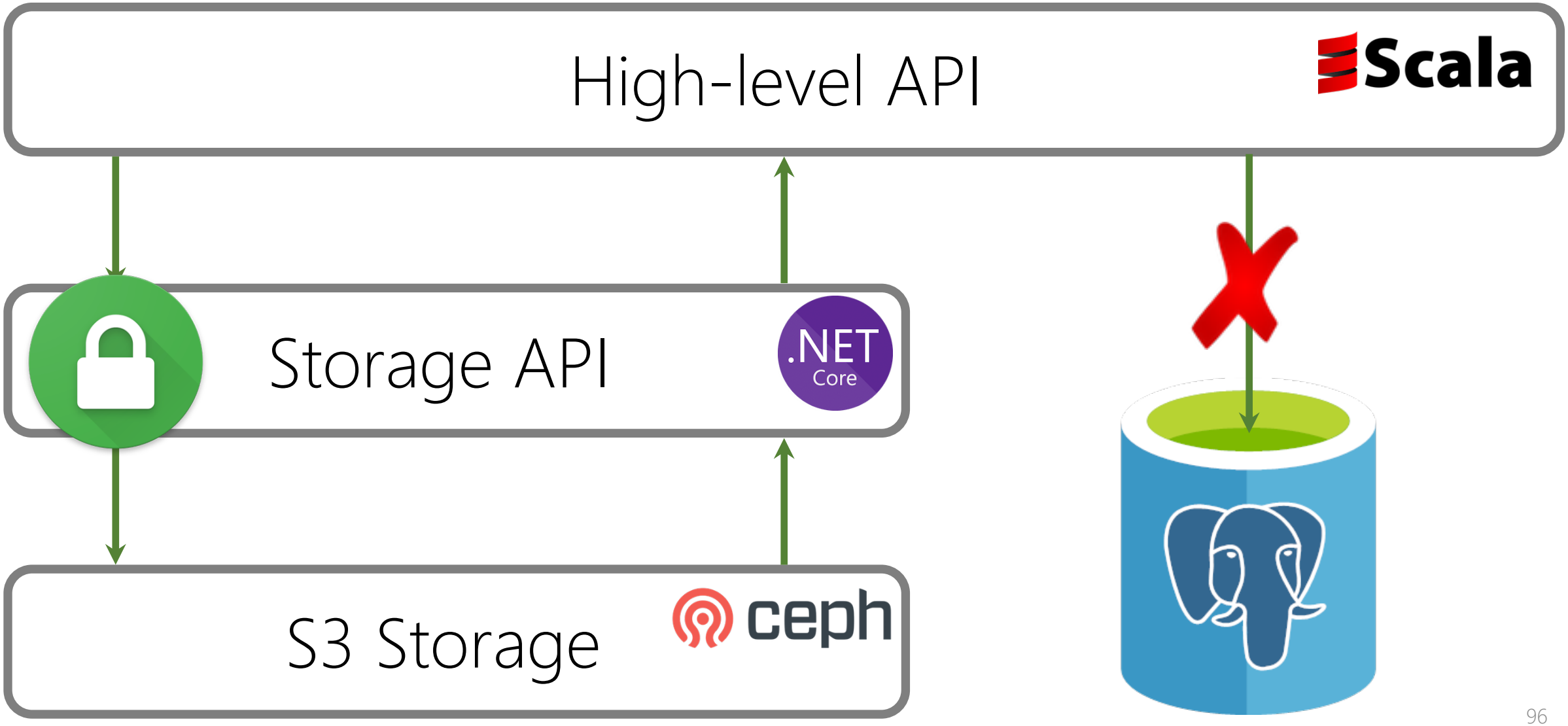
Распределенные изменения



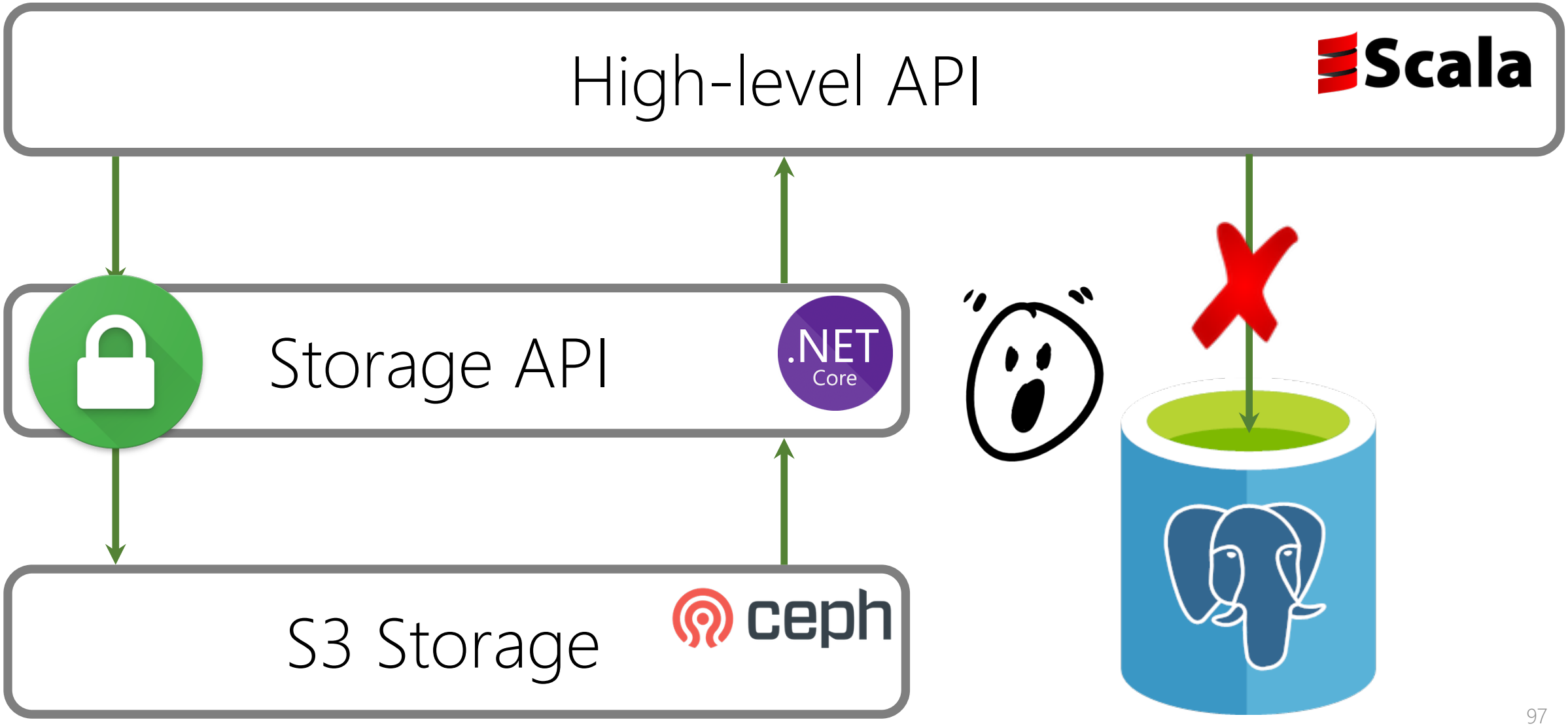
Распределенные изменения



Распределенные изменения



Распределенные изменения



Распределенные изменения

Неизменяемость (immutability)

Распределенные изменения

Неизменяемость (immutability)

- Версионирование данных

Распределенные изменения

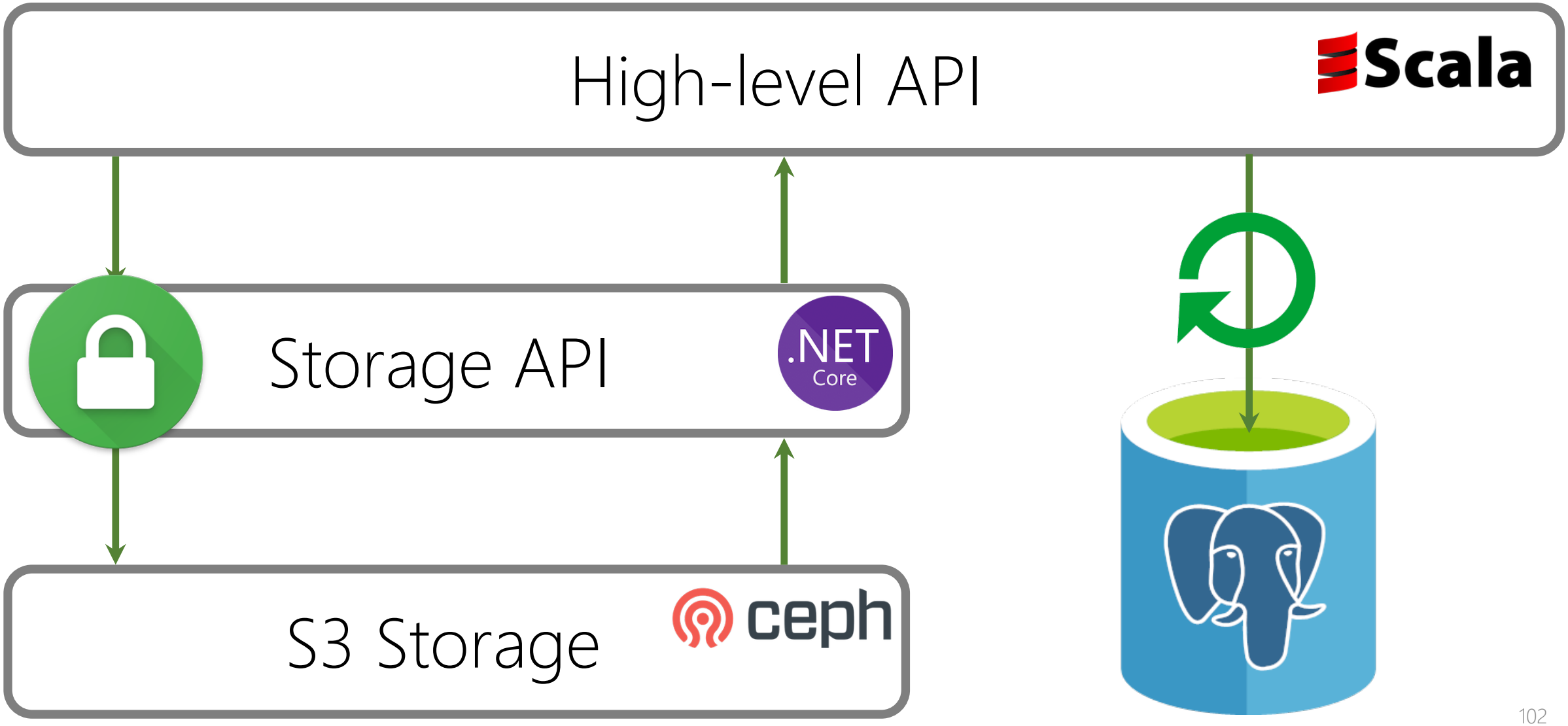
Неизменяемость (immutability)

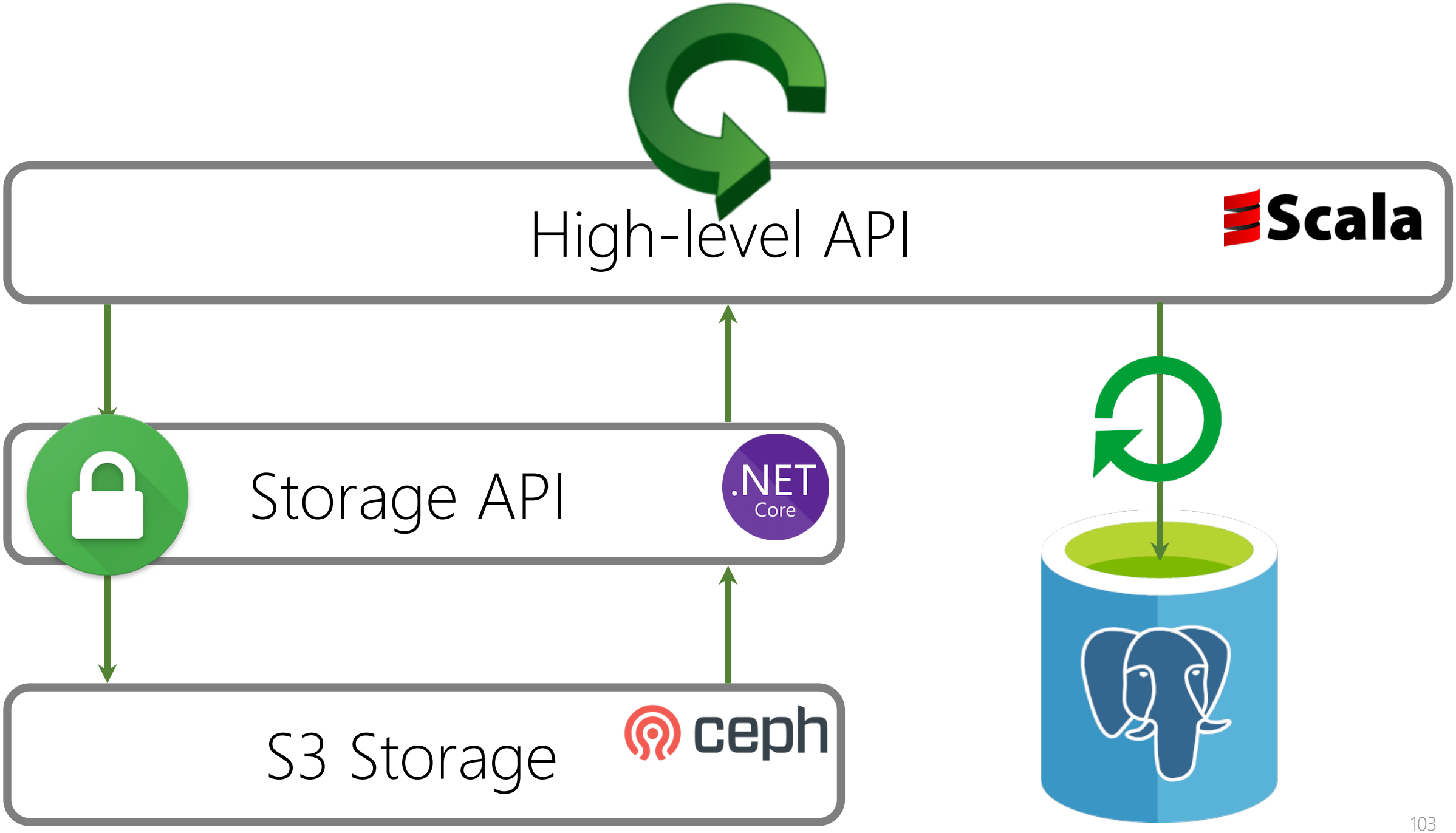
- Версионирование данных
- Данные + события

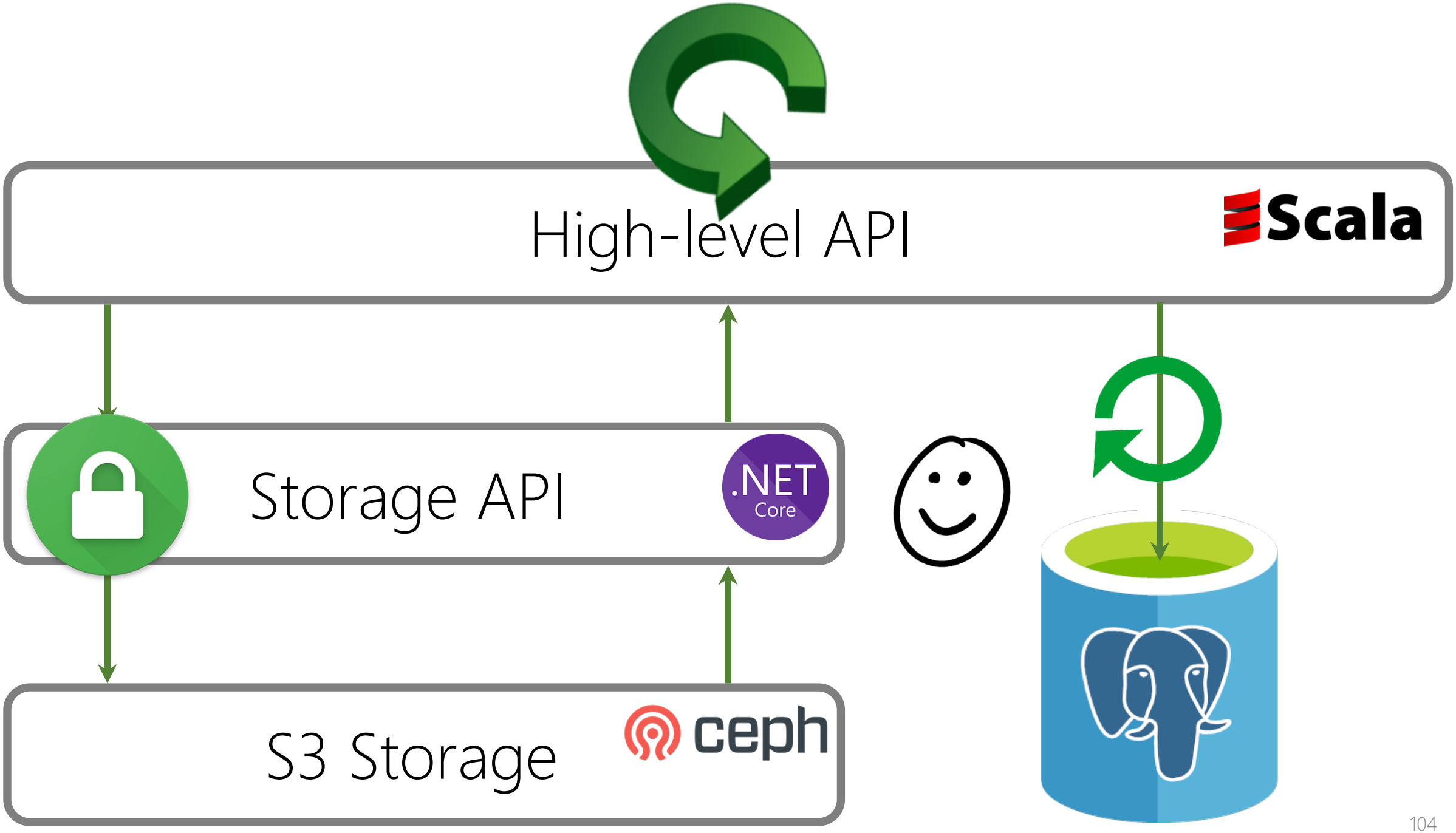
Распределенные изменения

Неизменяемость (immutability)

- Версионирование данных
- Данные + события
- Данные = события (event sourcing)

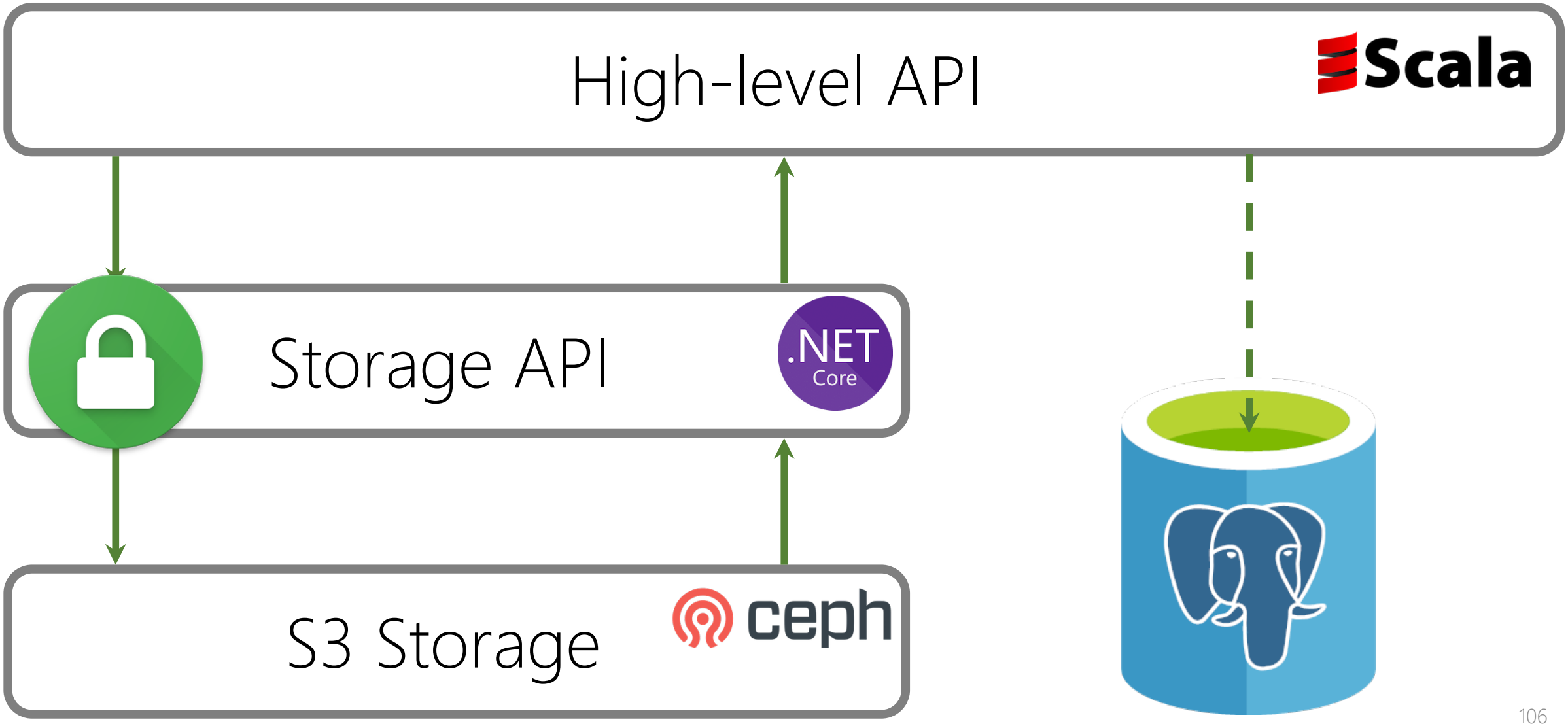

High-level API
Scala
Storage API
.NET
Core
S3 Storage
ceph


High-level API
Scala
Storage API
.NET
Core
S3 Storage
ceph

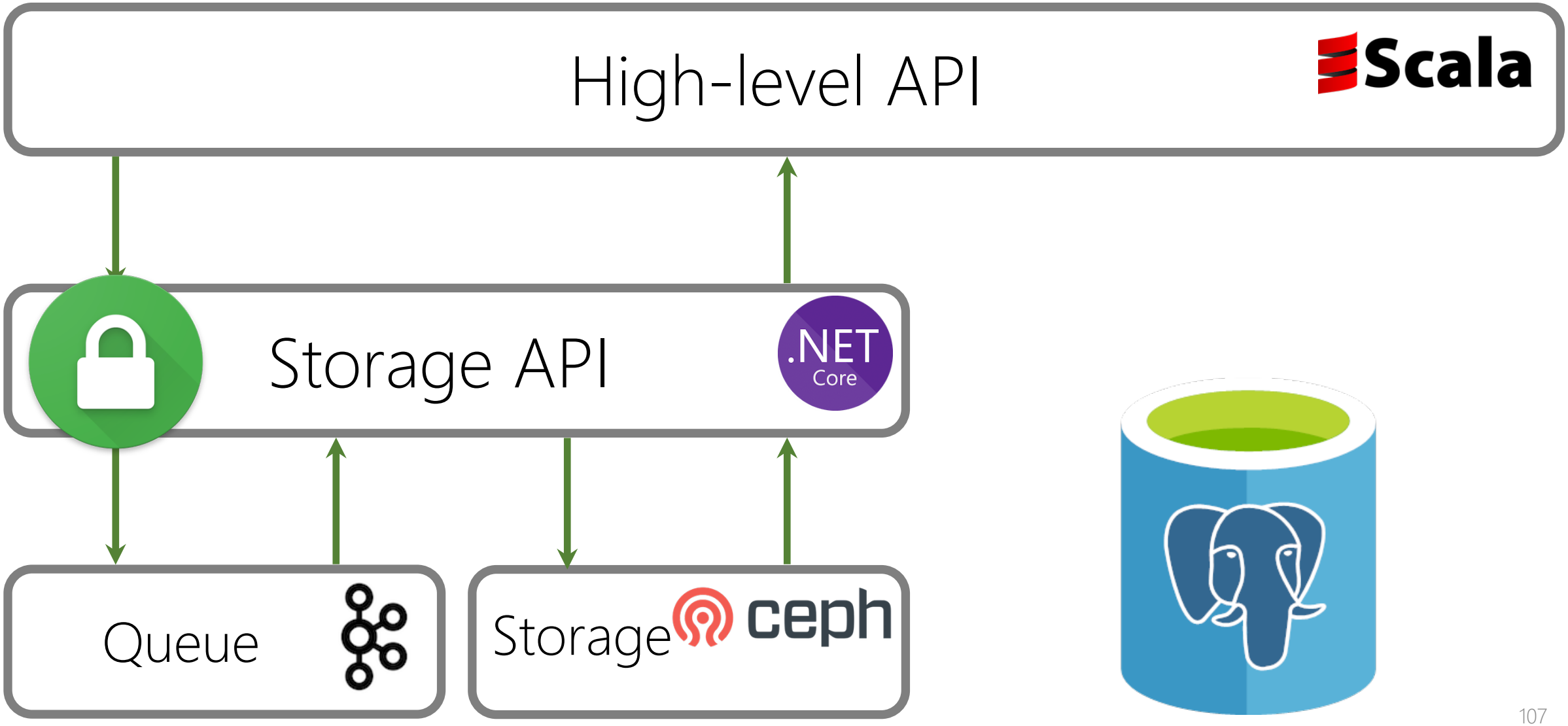

High-level API
Scala
Storage API
.NET
Core
S3 Storage
ceph

Неблокирующие изменения

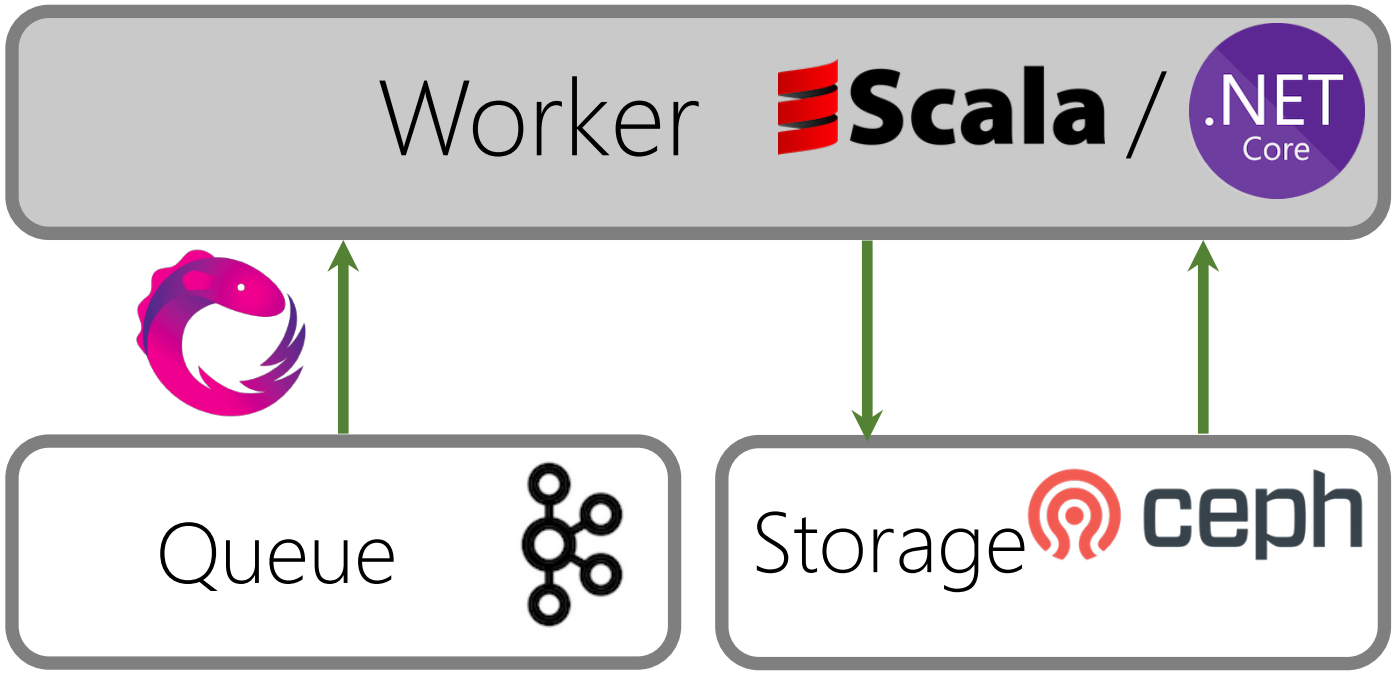
Неблокирующие изменения



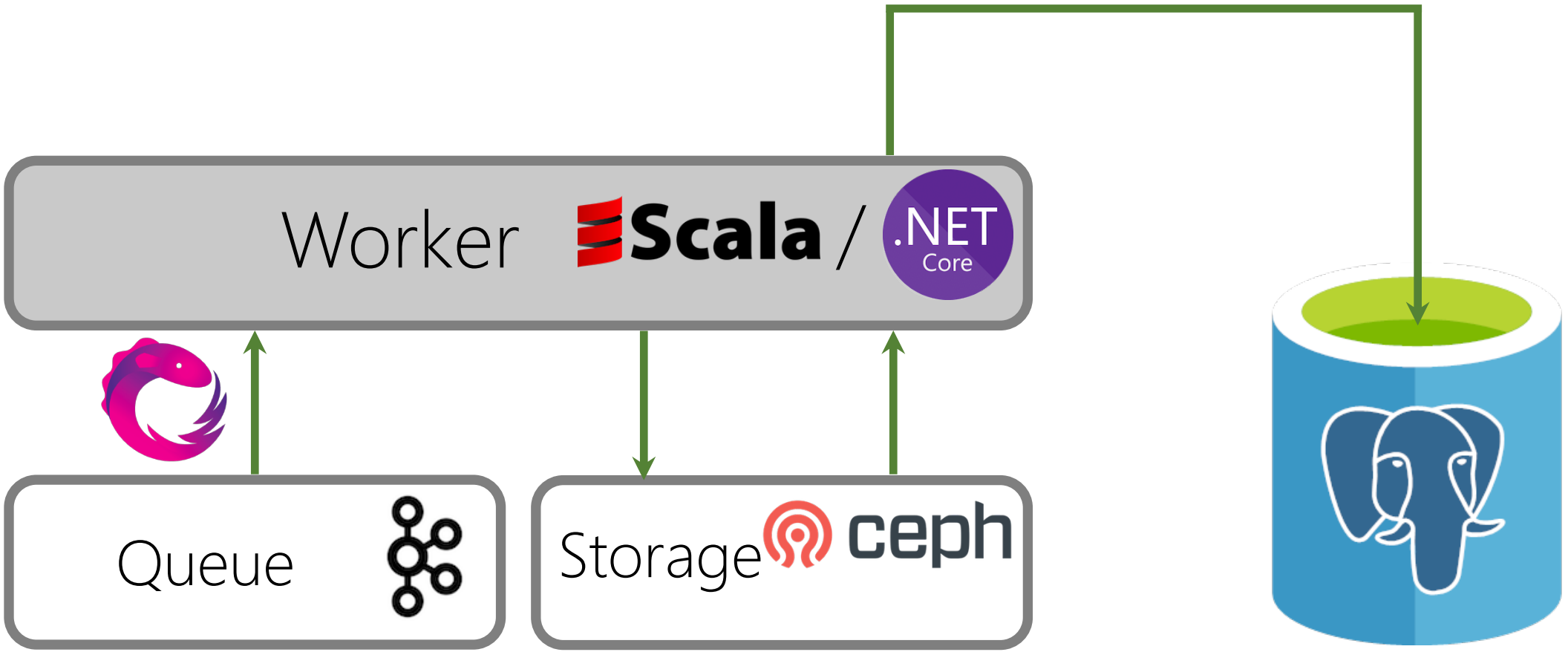
Неблокирующие изменения



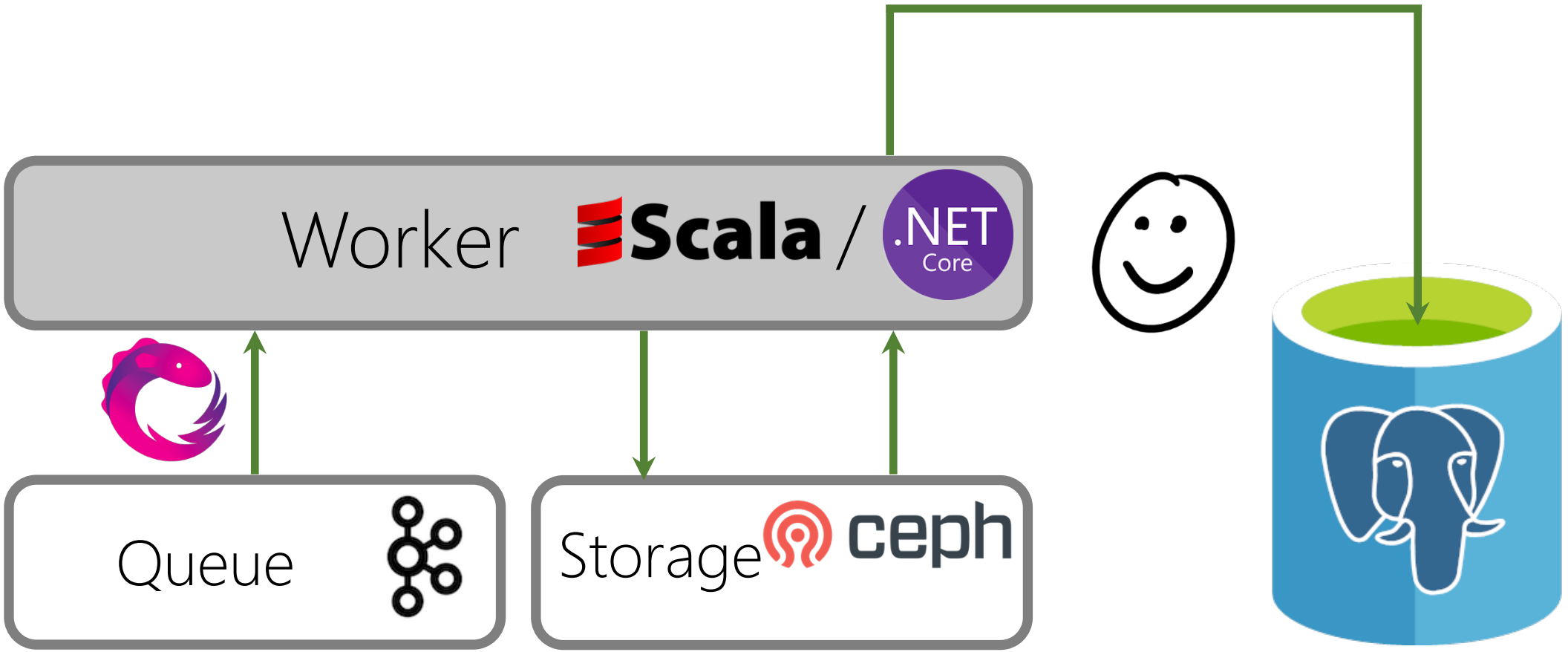
Неблокирующие изменения



Неблокирующие изменения



Неблокирующие изменения



Краткие итоги

Краткие итоги

- Микросервисы не появляются сами по себе

Краткие итоги

- Микросервисы не появляются сами по себе
- Управление изменениями в микросервисном приложении значительно усложняется

Краткие итоги

- Микросервисы не появляются сами по себе
- Управление изменениями в микросервисном приложении значительно усложняется
- Apache Kafka упрощает асинхронные взаимодействия

Краткие итоги

- Микросервисы не появляются сами по себе
- Управление изменениями в микросервисном приложении значительно усложняется
- Apache Kafka упрощает асинхронные взаимодействия
- Реактивный подход убирает границы между компонентами приложения

Полезные ссылки

- https://github.com/denisivan0v/dotnext-moscow-2017
- https://github.com/2gis/nuclear-vstore
- https://customers.microsoft.com/en-us/story/2gis-professional-services-asp-net-core

Спасибо! Вопросы?

Денис Иванов

@denisivanov

denis@ivanovdenis.ru
https://github.com/denisivan0v