

# Реактивные микросервисы с Apache Kafka



Денис Иванов

[denis@ivanovdenis.ru](mailto:denis@ivanovdenis.ru)

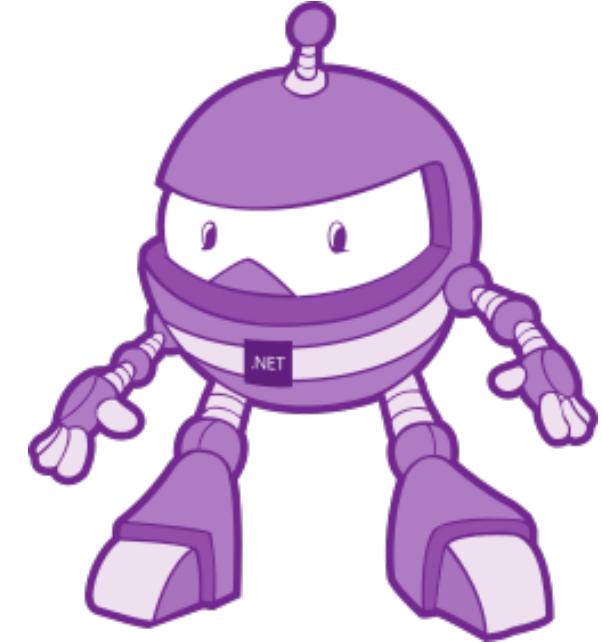
@denisivanov



**HighLoad<sup>++</sup>**

Профессиональная конференция  
разработчиков высоконагруженных  
систем

# Обо мне

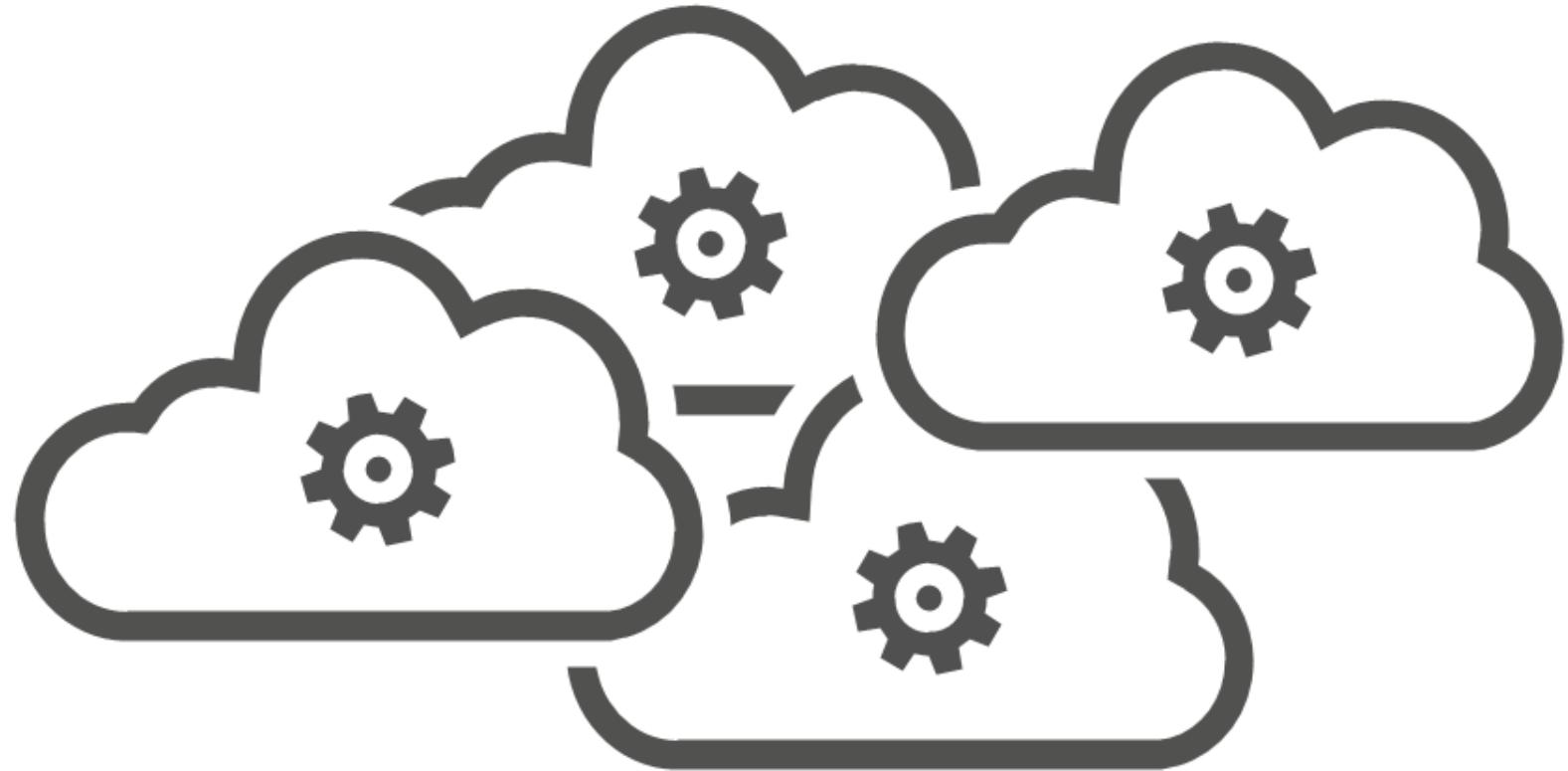


# Код и презентация

<https://github.com/denisivanov/highload-2017>

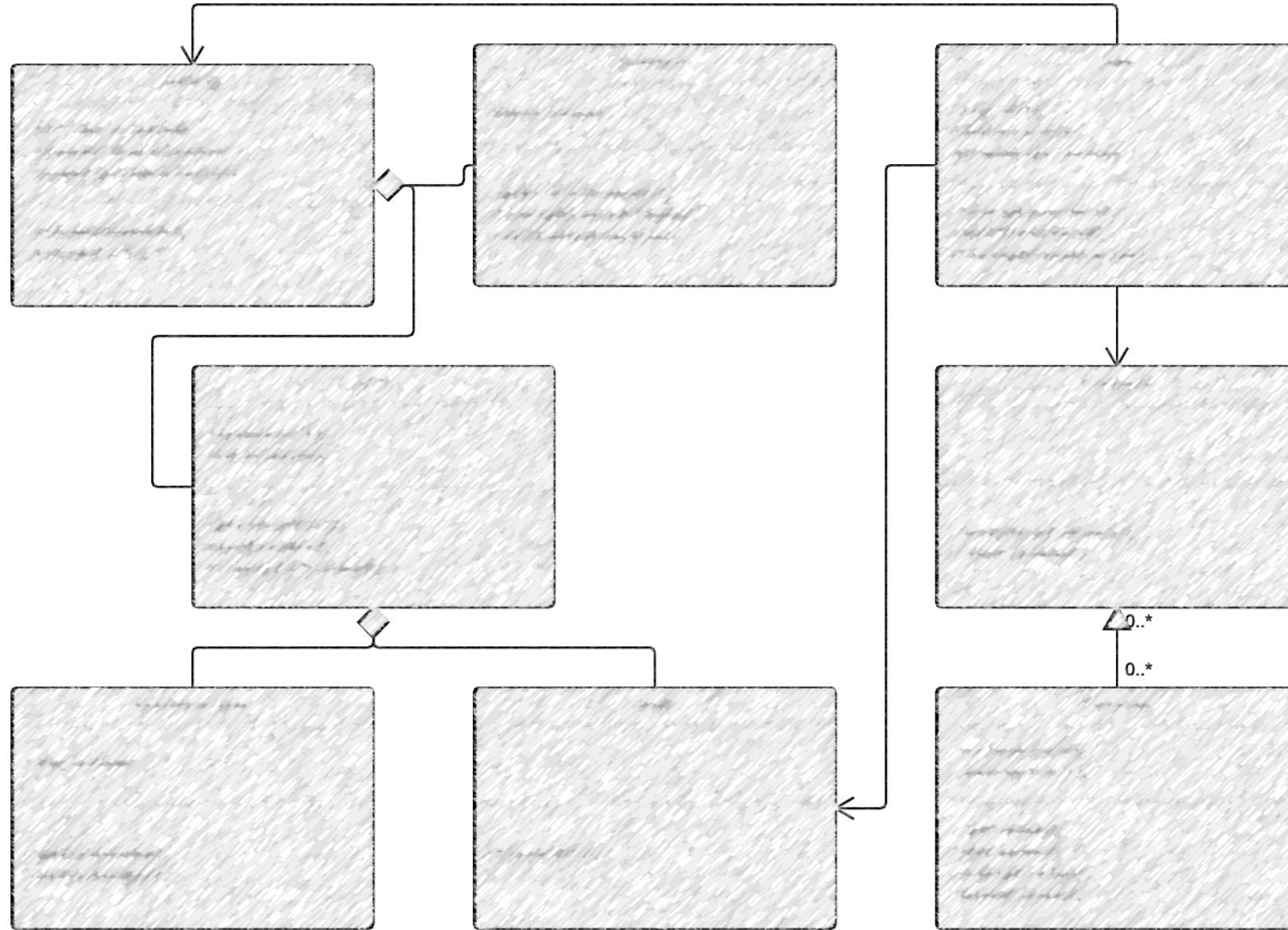
# План



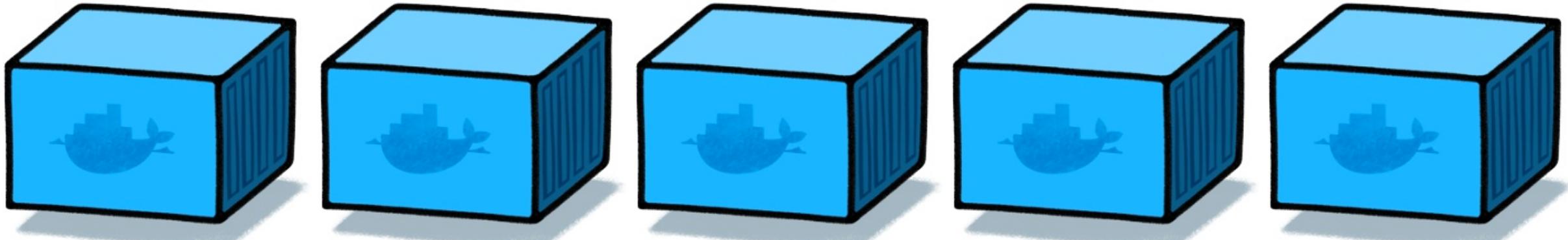


# Microservices

# Терминология



# Терминология



**kubernetes**

# Откуда берутся микросервисы?

# Откуда берутся микросервисы?

## Advertising Management System



# Откуда берутся микросервисы?

Frontend App

High-level API

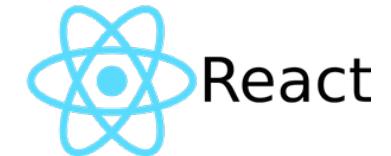
Storage API

S3 Storage



# Откуда берутся микросервисы?

Frontend App



High-level API



Storage API



S3 Storage



# Challenges

- Синхронное/асинхронное взаимодействие компонентов

# Challenges

- Синхронное/асинхронное взаимодействие компонентов
- Гарантиированная доставка данных с минимальными задержками

# Challenges

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

# Challenges

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

# Распределенные изменения

# Распределенные изменения

- Блокирующие
- Неблокирующие (отложенные)

# Распределенные изменения

High-level API



Storage API



S3 Storage



# Распределенные изменения

High-level API



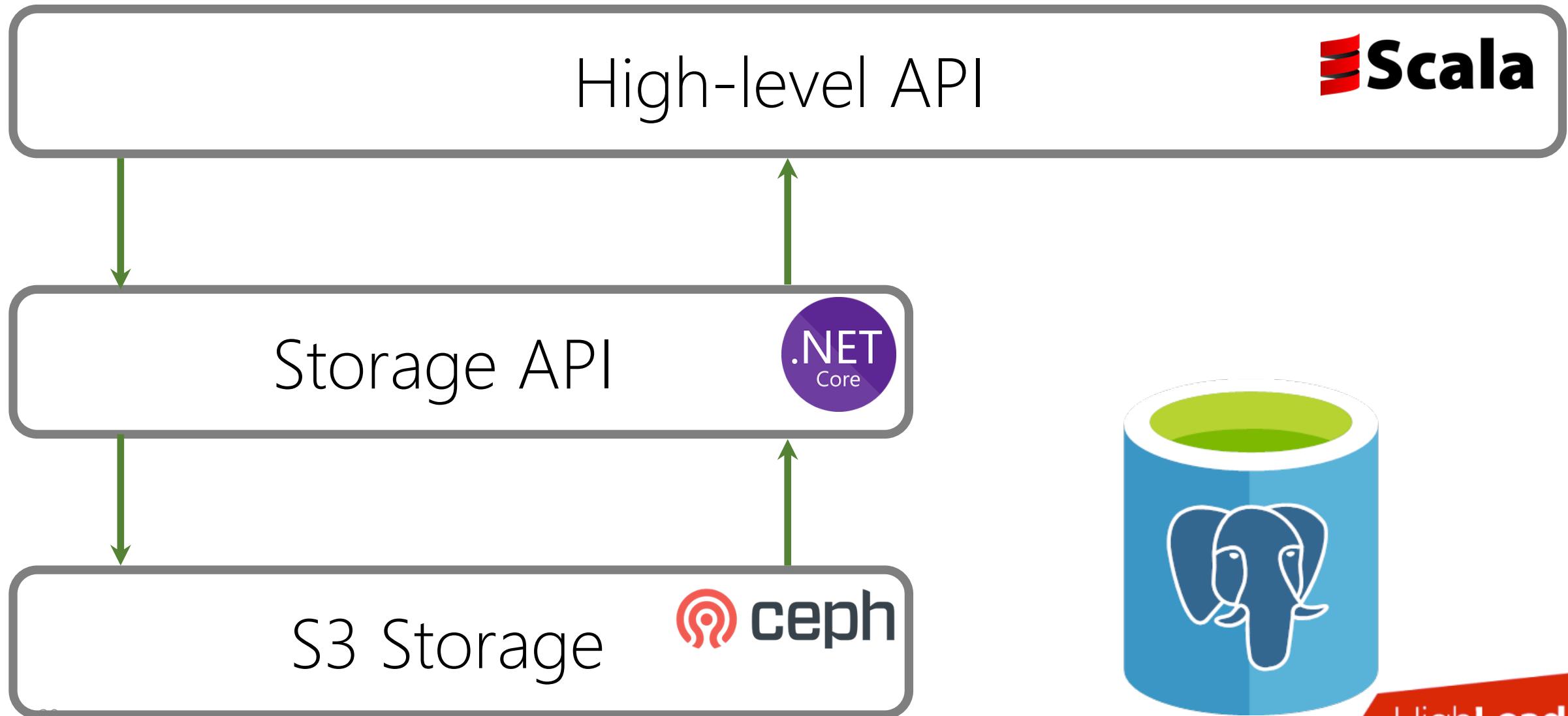
Storage API



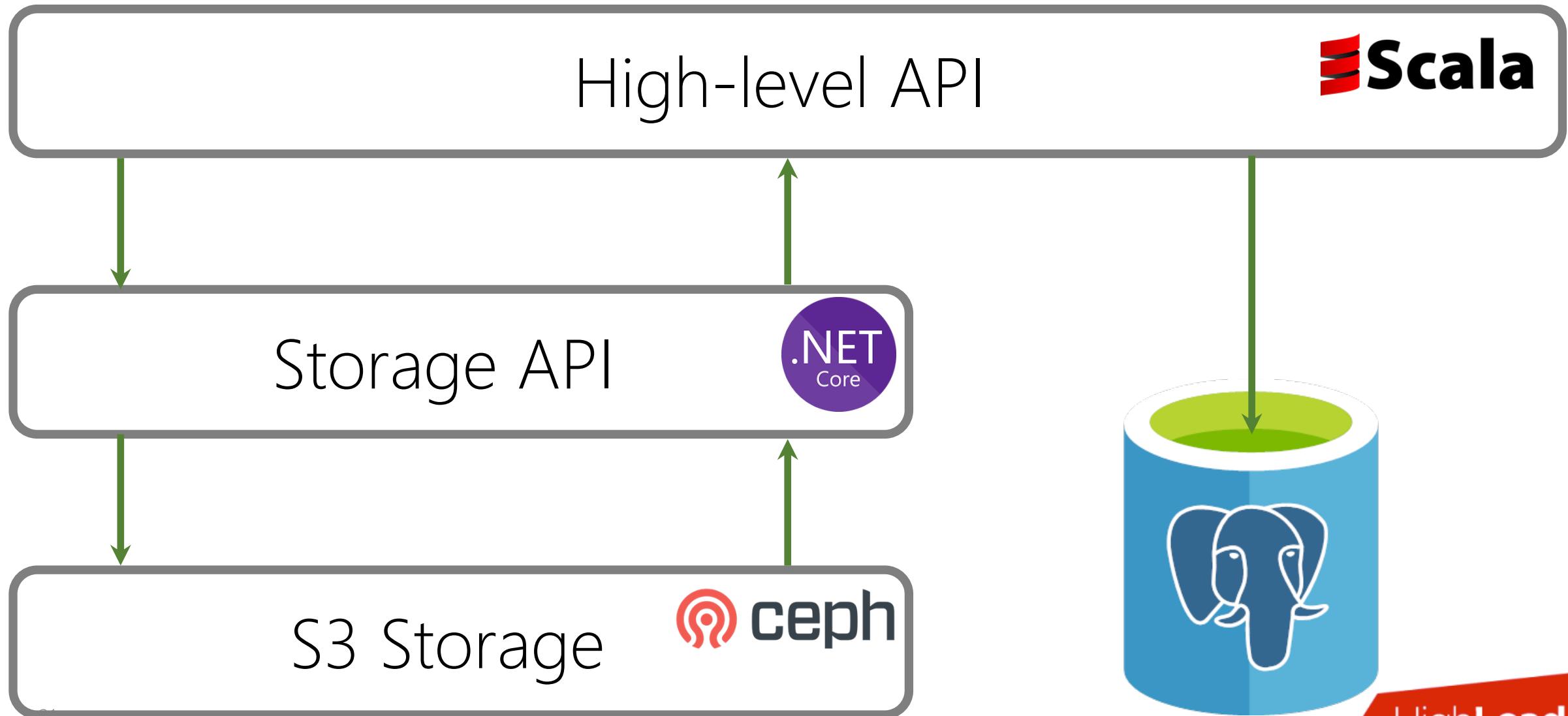
S3 Storage



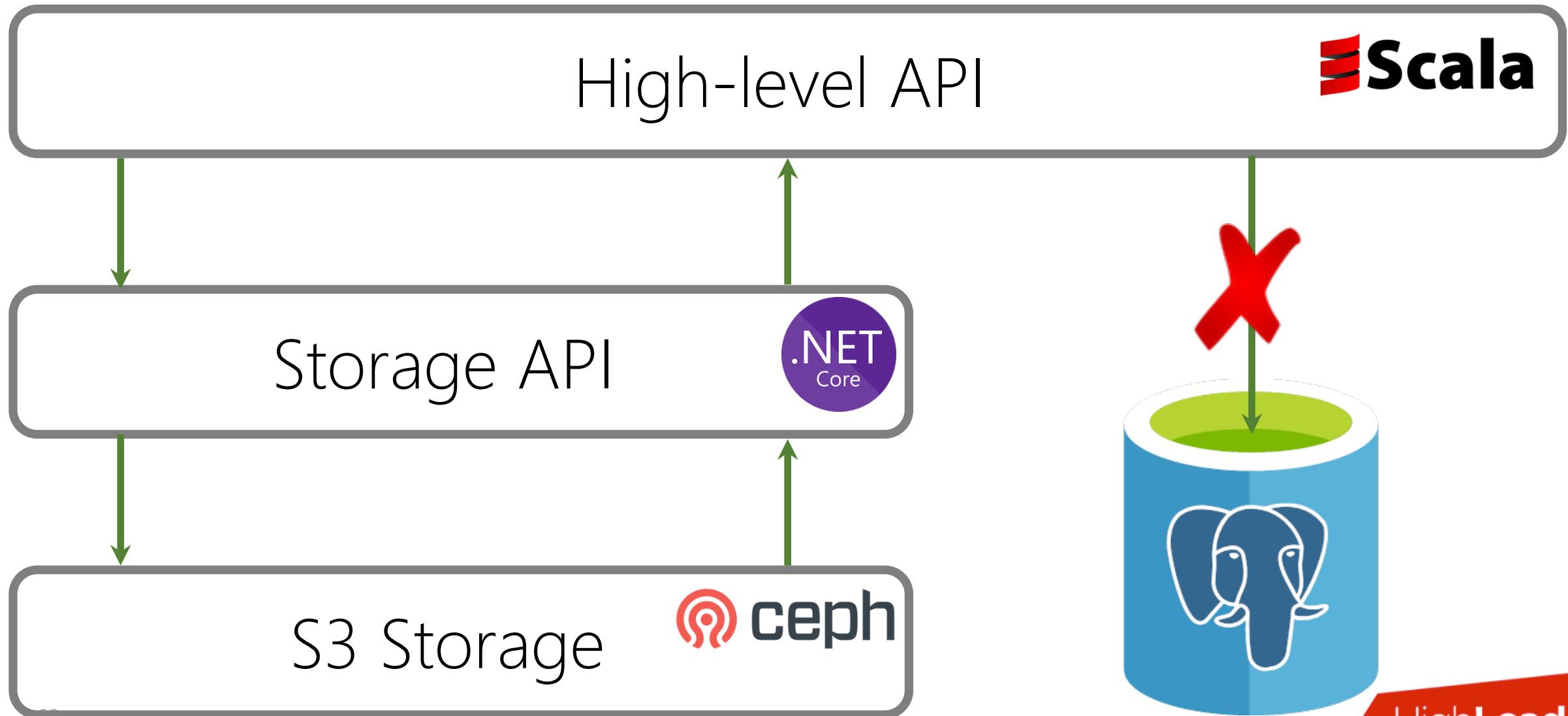
# Распределенные изменения



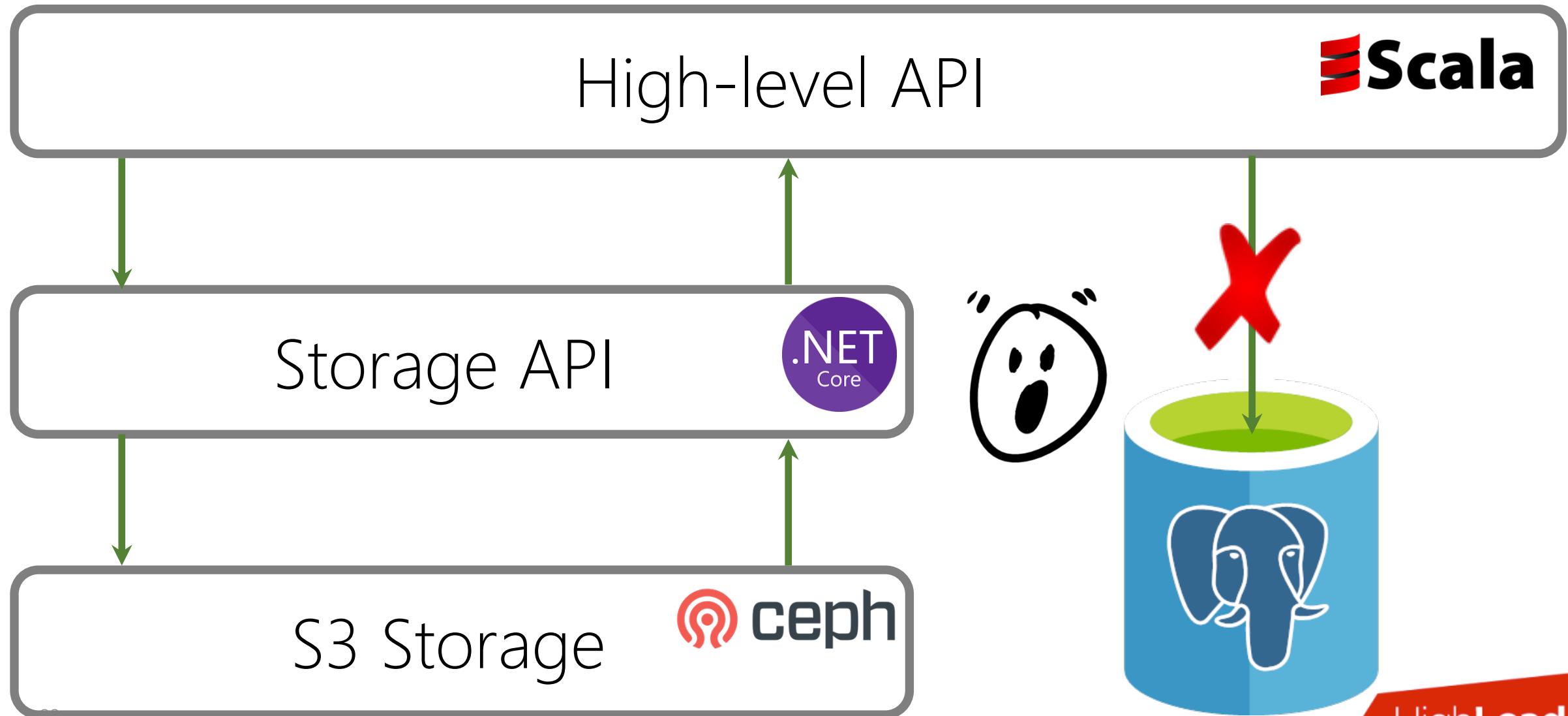
# Распределенные изменения



# Распределенные изменения

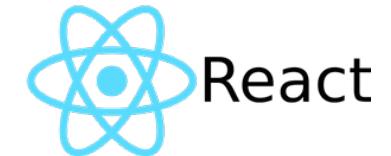


# Распределенные изменения



# Фоновые процессы

Frontend App



REST API



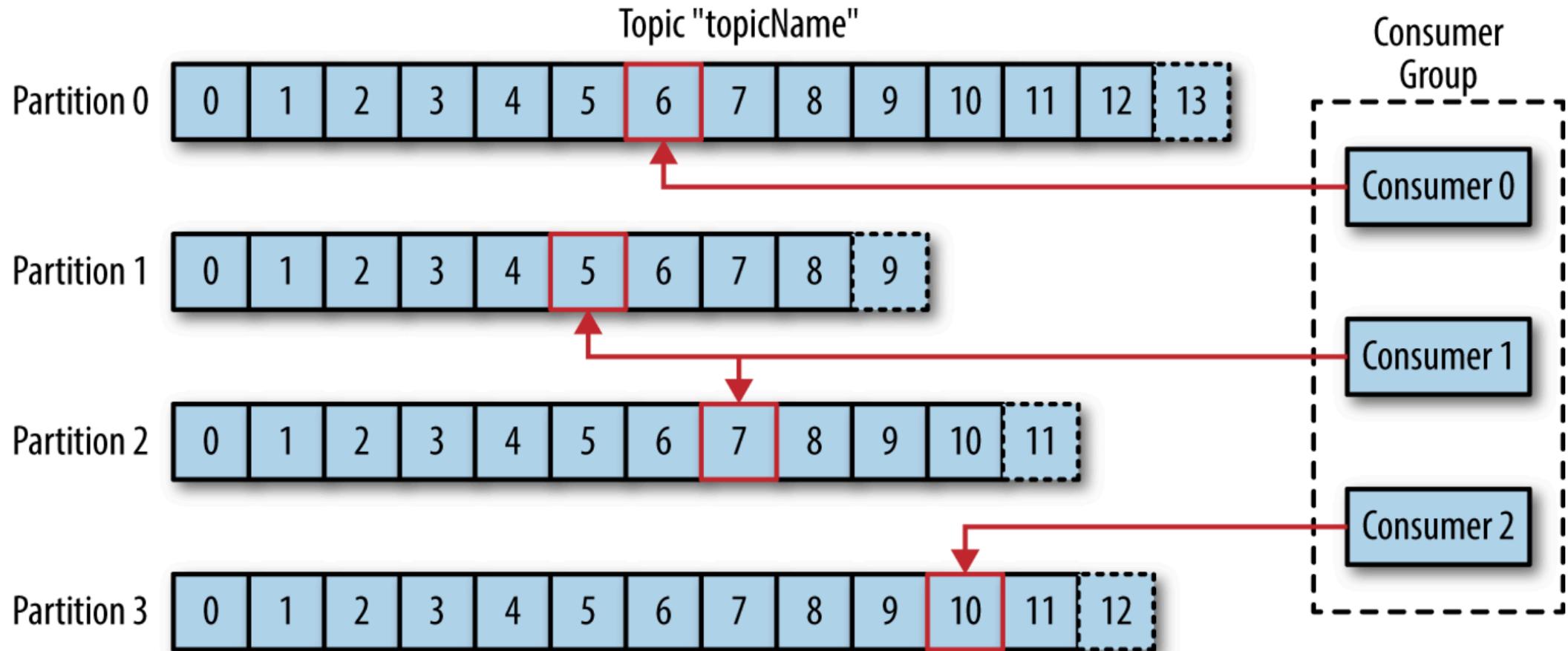
Storage

Background workers





# Распределенный лог



# Kafka .NET Client

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

# 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);
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);
    }
}
```

# DEMO: Kafka + Docker + .NET Core

# More challenges

- Легко написать неправильно

# More challenges

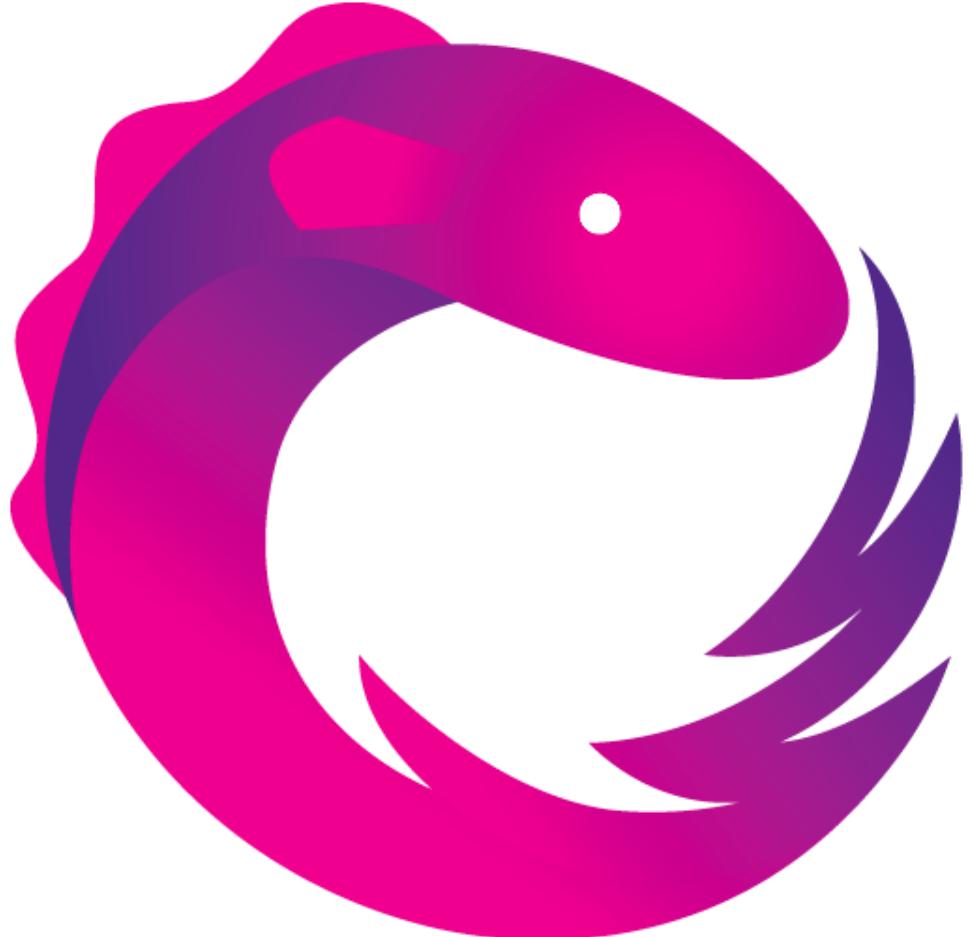
- Легко написать неправильно
- Частые перебалансировки consumer group

# More challenges

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

# More challenges

- Легко написать неправильно
- Частые перебалансировки consumer group
- Управление分散ными consumerами
- Задержки, синхронизация, утилизация ресурсов



# Reactive Extensions **(Rx)**

# Реактивные приложения

<https://www.reactivemanifesto.org/>  
<http://reactivex.io/>

# Реактивные приложения

- Событийно-ориентированные

<https://www.reactivemanifesto.org/>

<http://reactivex.io/>

# Реактивные приложения

- Событийно-ориентированные
- Масштабируемые

<https://www.reactivemanifesto.org/>

<http://reactivex.io/>

# Реактивные приложения

- Событийно-ориентированные
- Масштабируемые
- Отказоустойчивые

<https://www.reactivemanifesto.org/>

<http://reactivex.io/>

# Реактивные приложения

- Событийно-ориентированные
- Масштабируемые
- Отказоустойчивые
- Отзывчивые

<https://www.reactivemanifesto.org/>

<http://reactivex.io/>

# 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);
}
```

...

```
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;
```

...

```
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);
```

# Rx Kafka Consumer Wrapper

```
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);

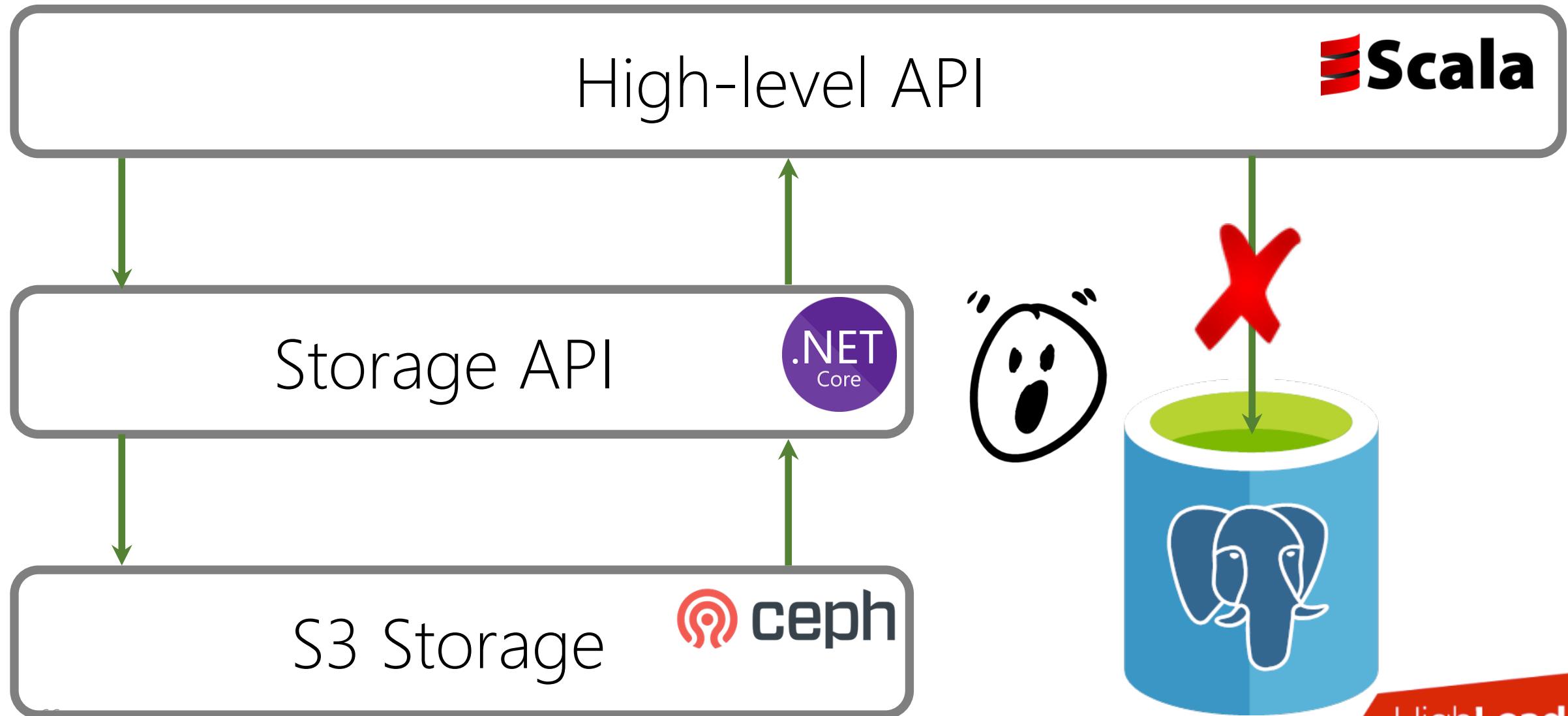
var subscription = observable
    .Buffer(batchSize)
    .Subscribe(
        messages =>
    {
        foreach (var message in messages)
        {
            Console.WriteLine(message.Value);
            consumerWrapper.CommitAsync(message)
                .GetAwaiter().GetResult();
        }
    });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

# DEMO: Kafka + Rx

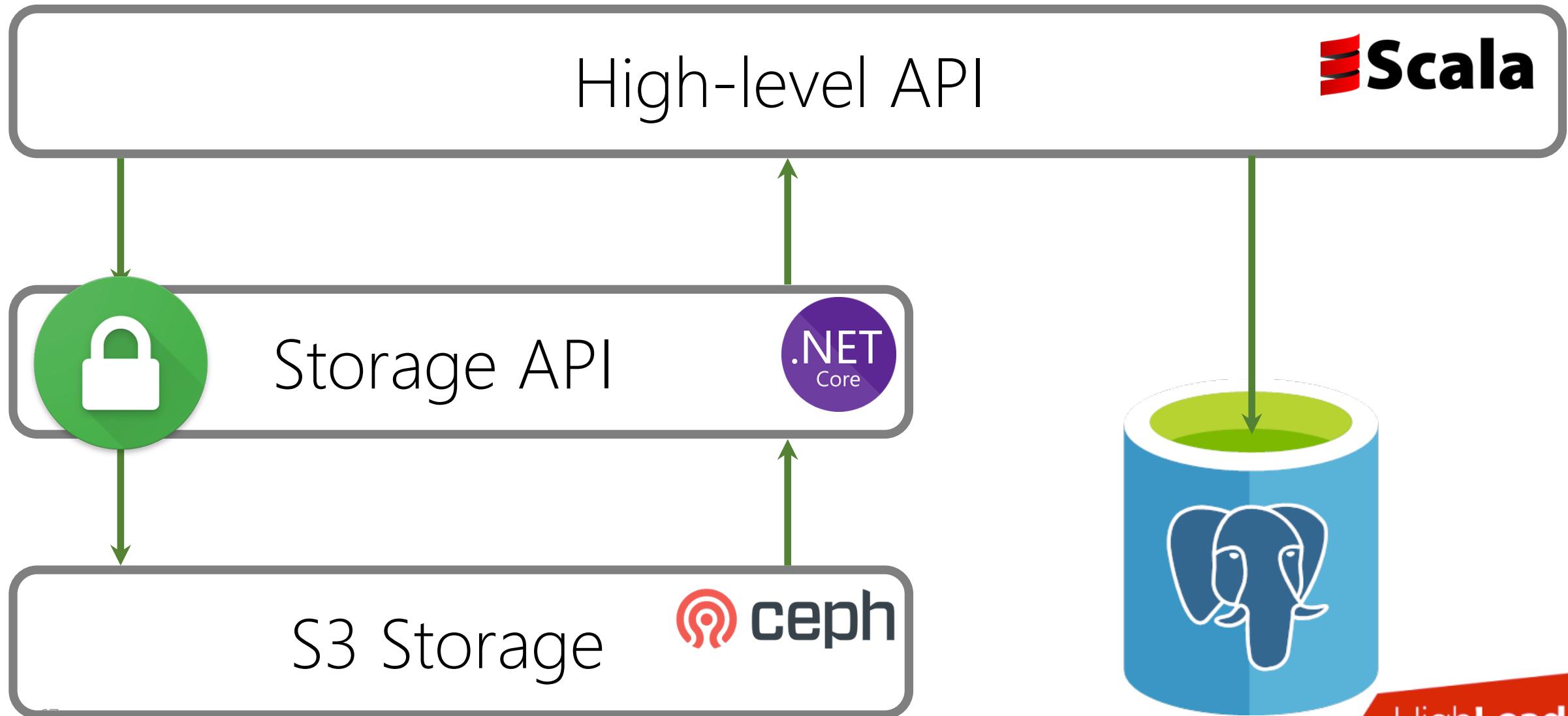


# Распределенные изменения

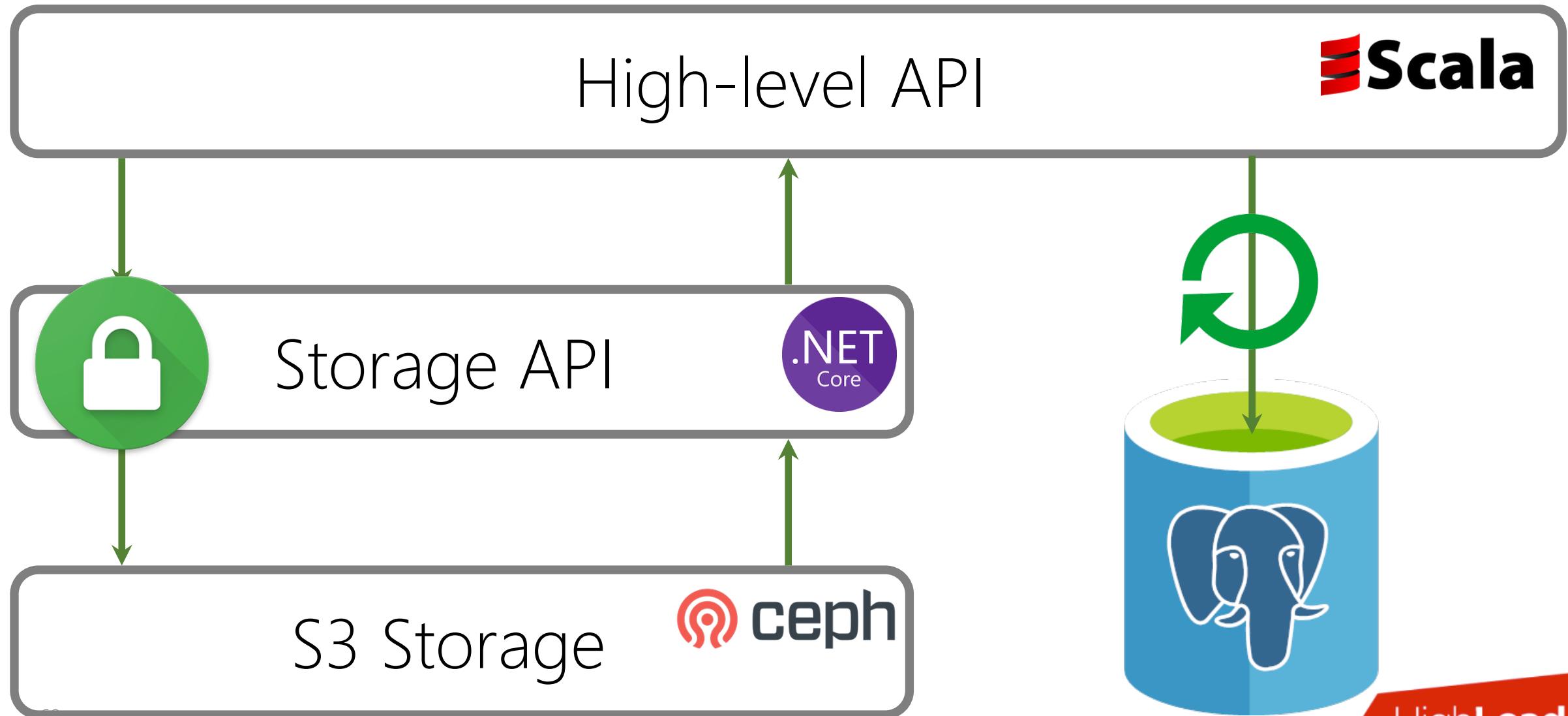
# Распределенные изменения



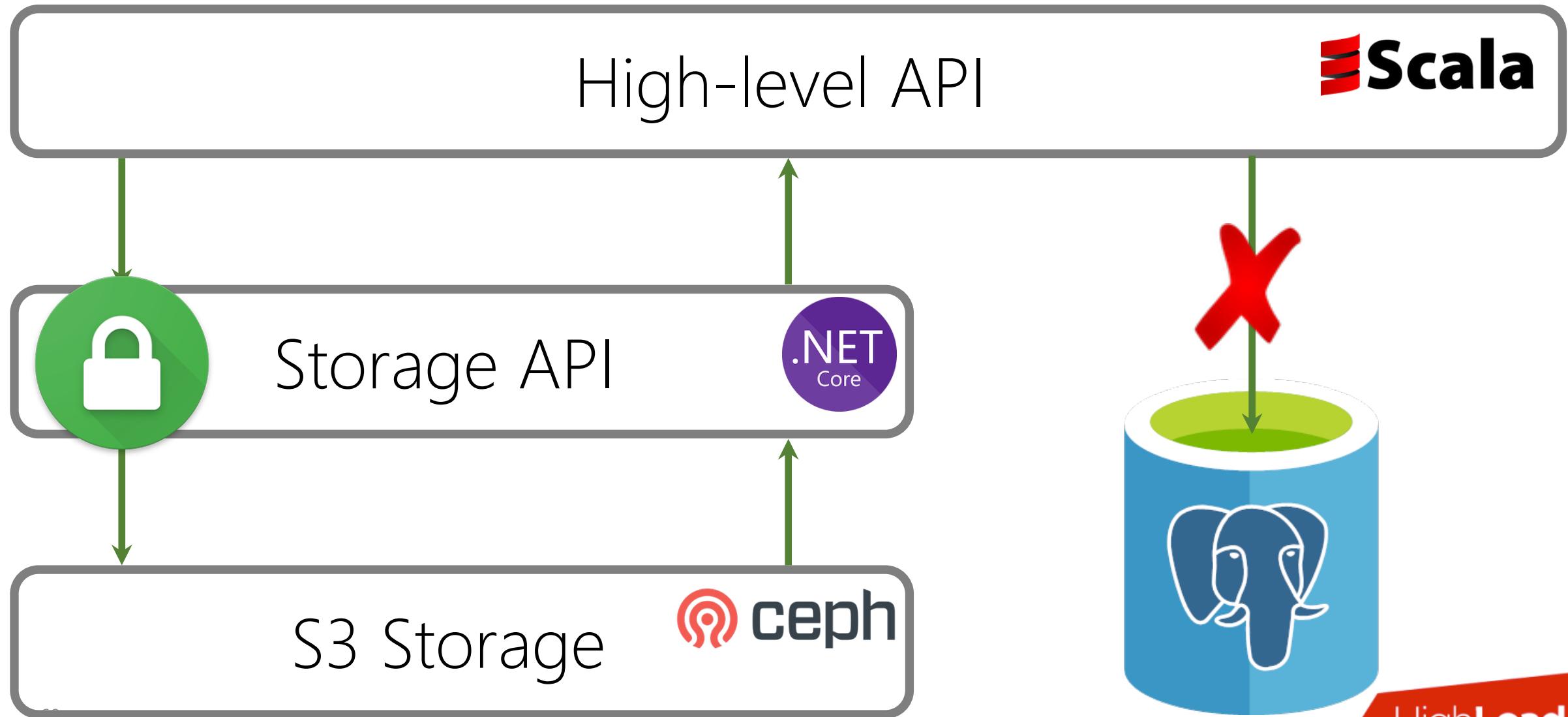
# Распределенные изменения



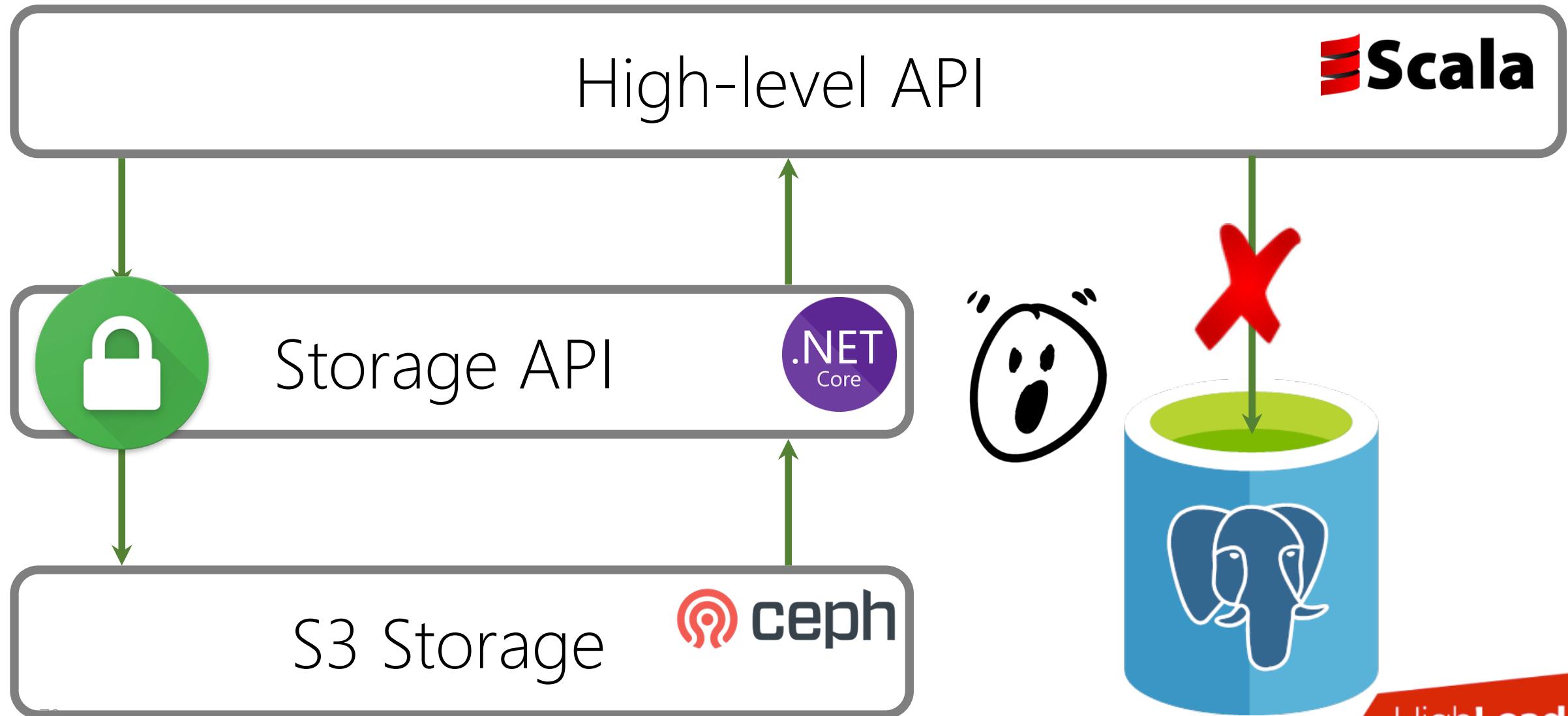
# Распределенные изменения



# 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

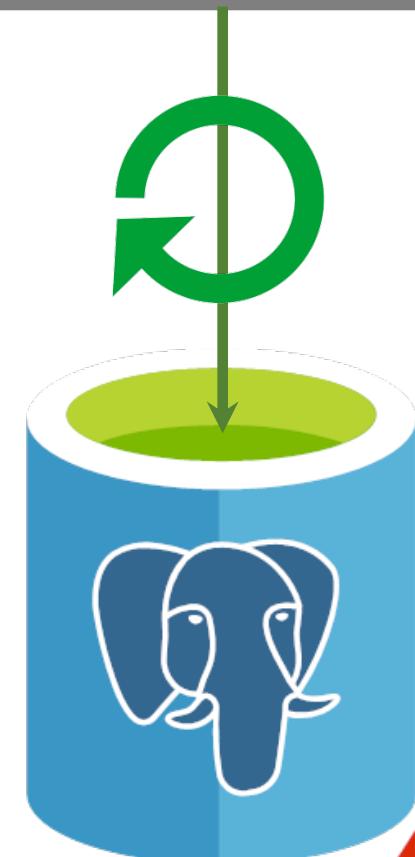


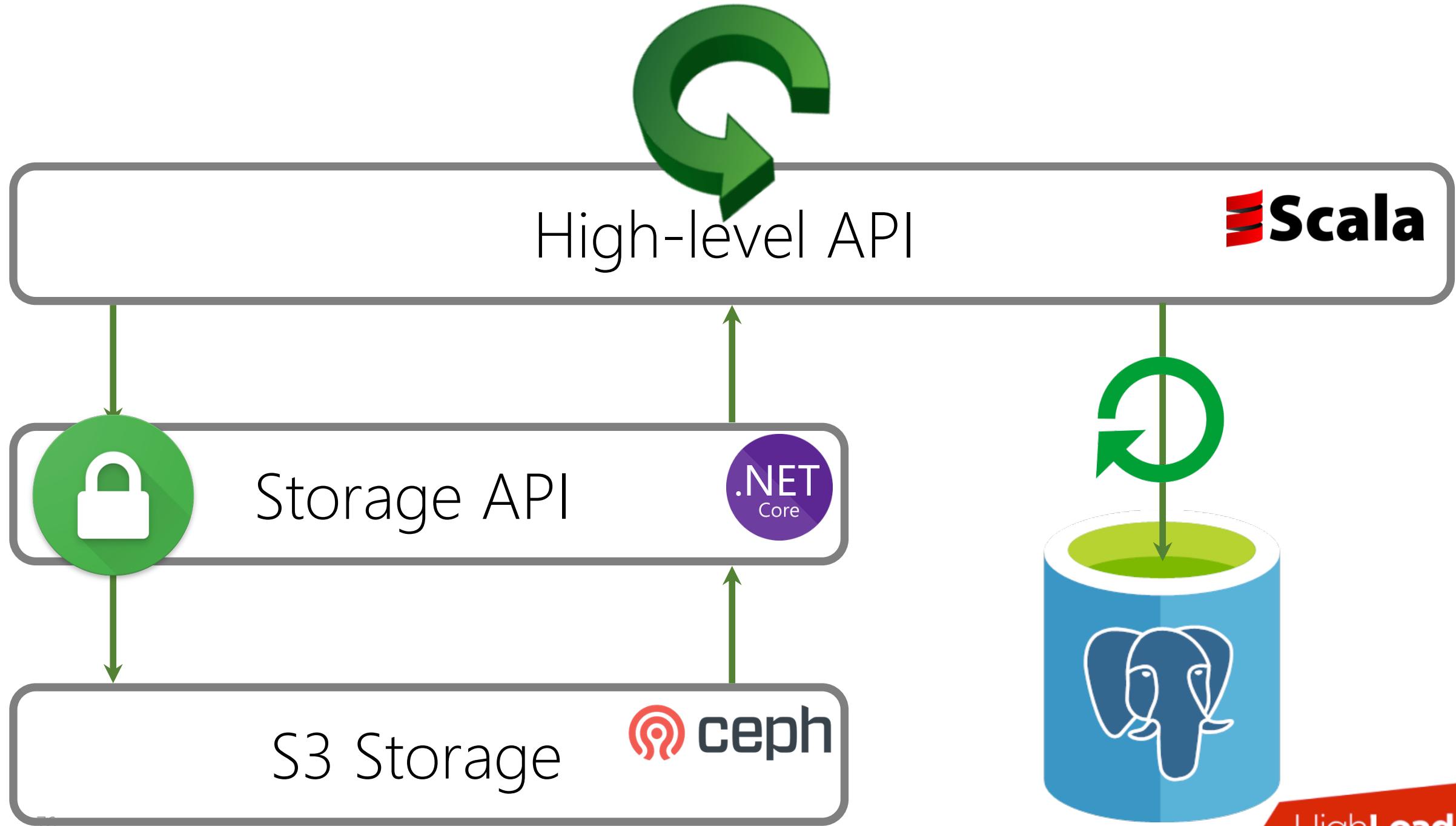
Storage API

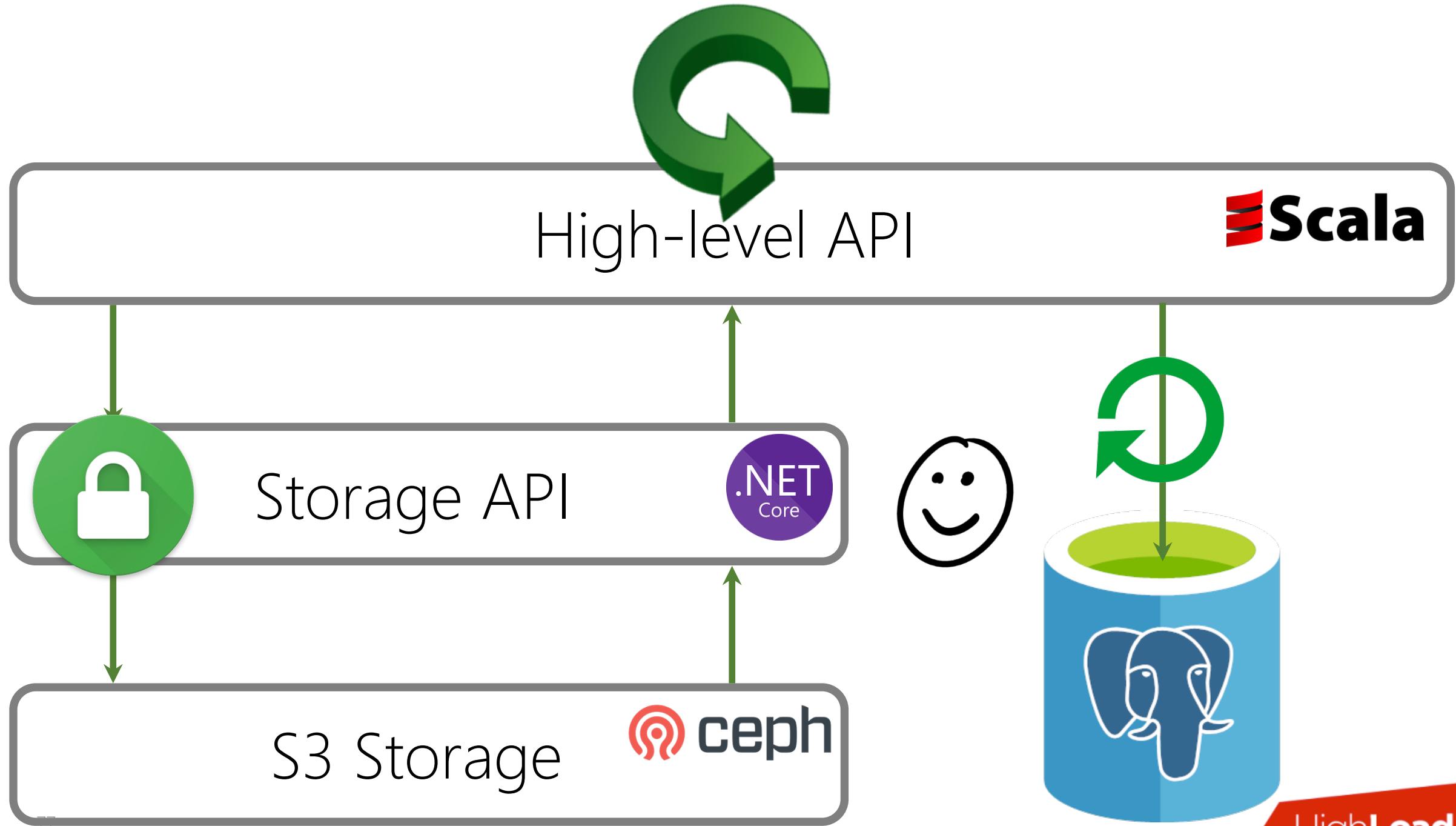


.NET  
Core

S3 Storage

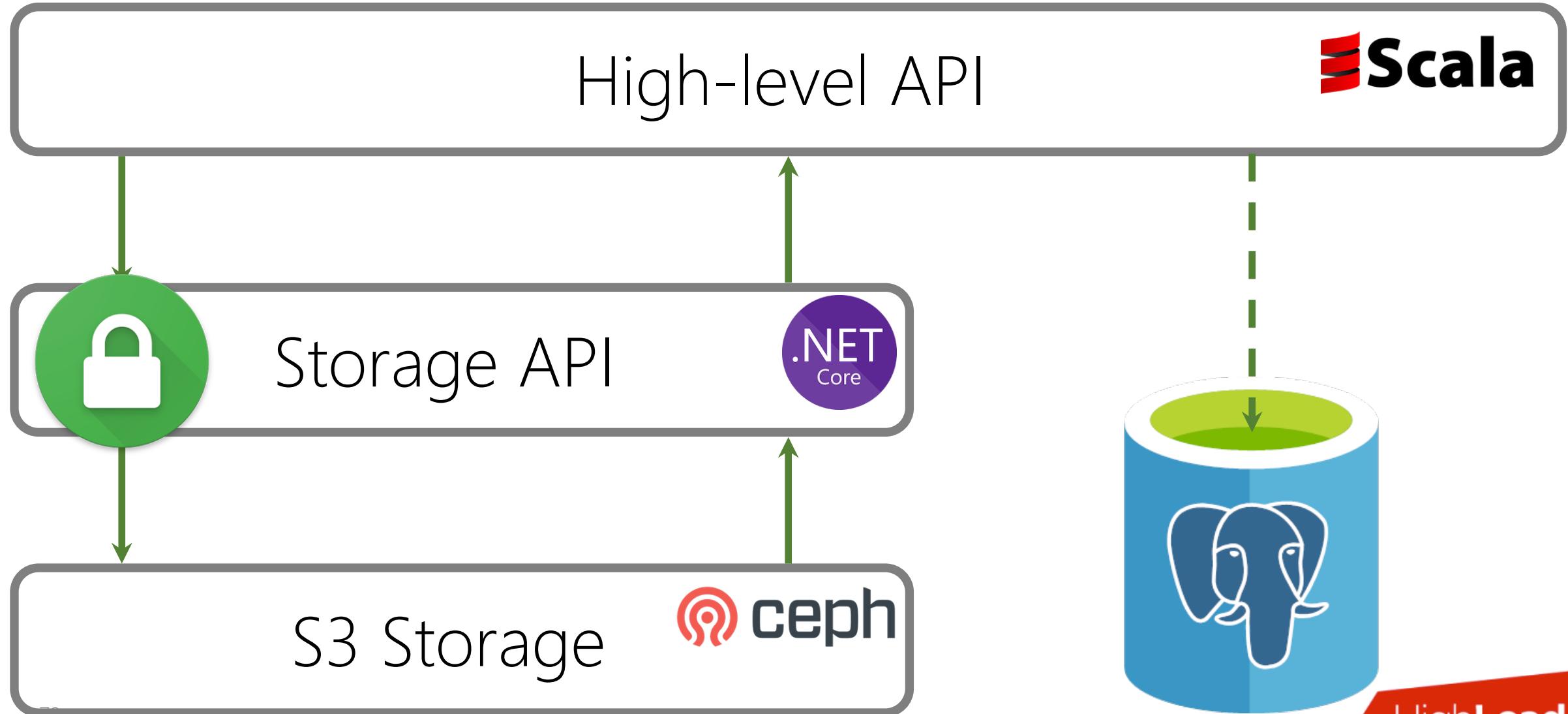






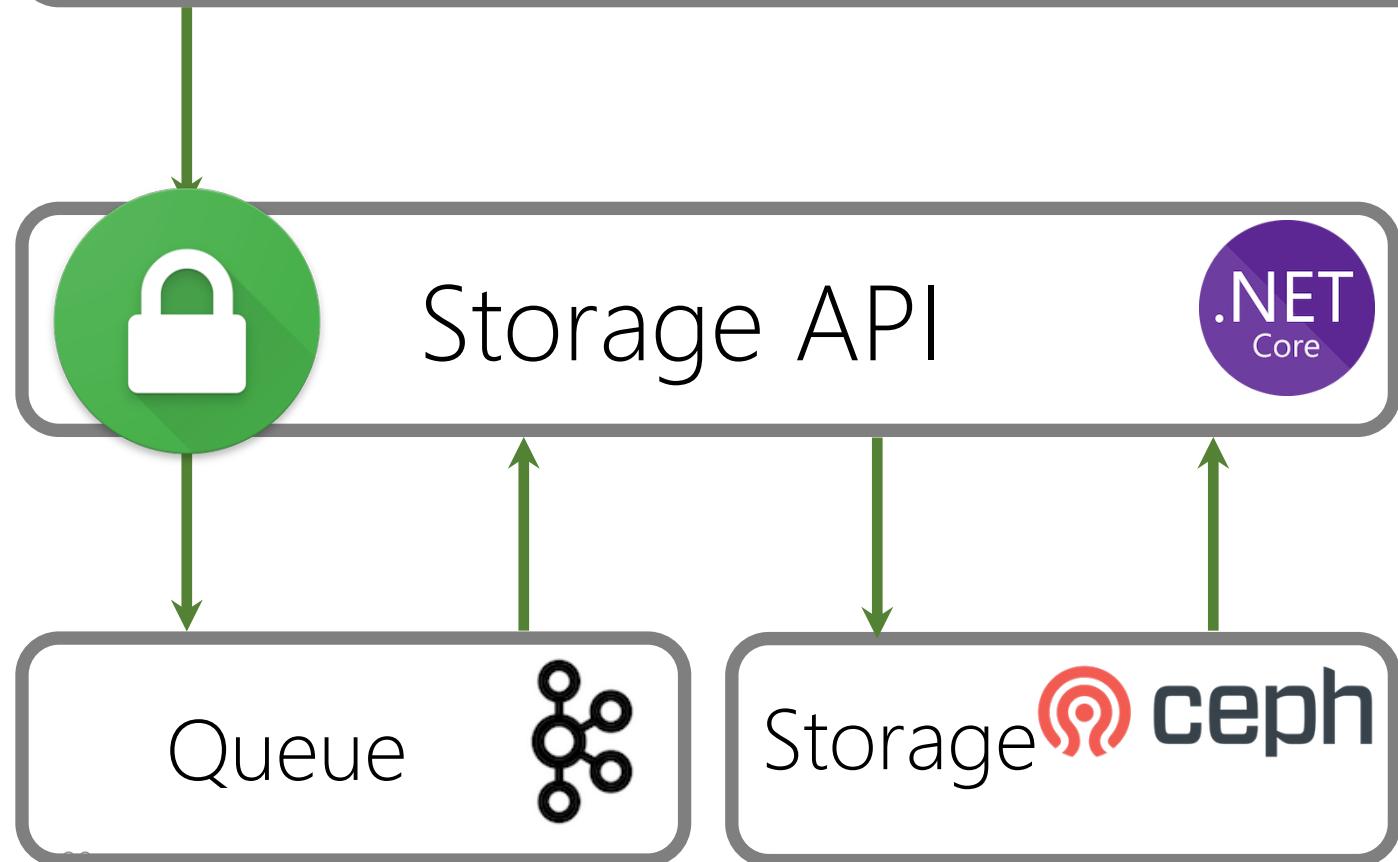
# Неблокирующие изменения

# Неблокирующие изменения

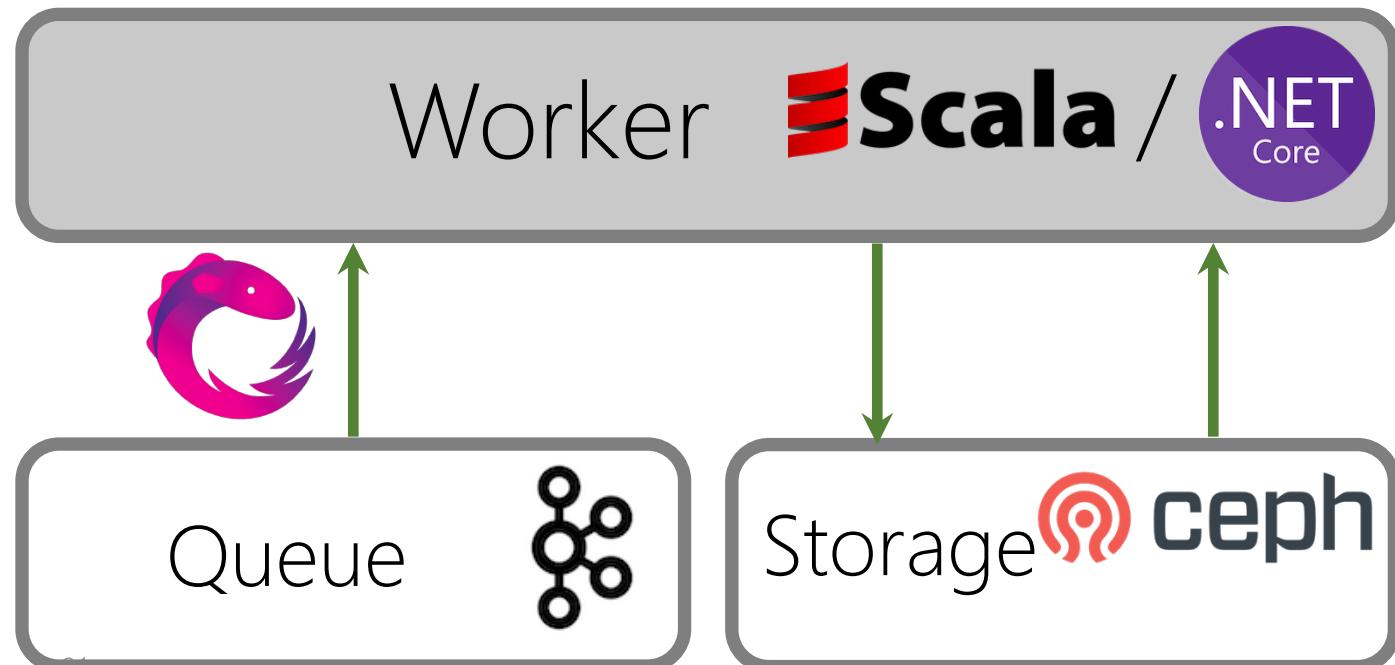


# Неблокирующие изменения

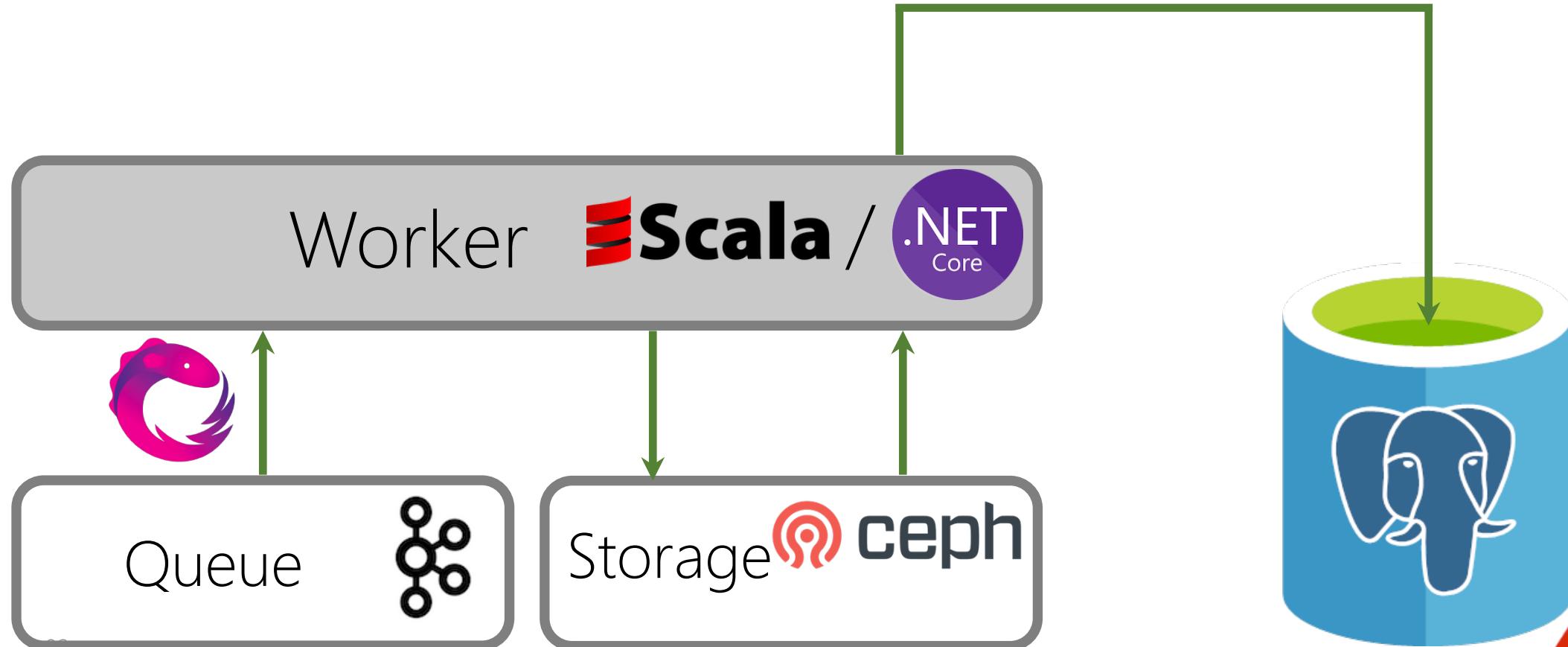
High-level API



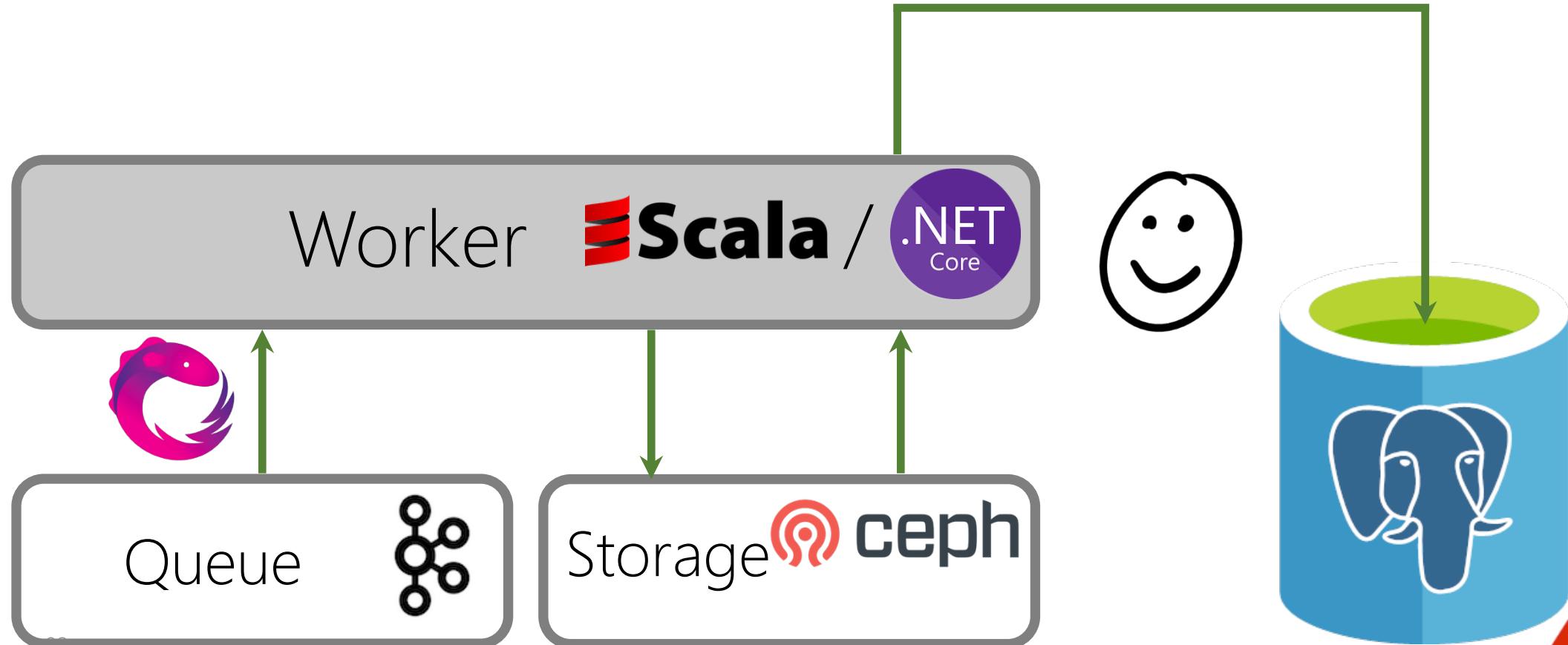
# Неблокирующие изменения



# 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 приложении значительно усложняется

# Краткие итоги

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

# Краткие итоги

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

# Полезные ссылки

<https://github.com/denisivanov/highload-2017>

<https://github.com/2gis/nuclear-vstore>



# Спасибо! Вопросы?

Денис Иванов

@denisivanov

denis@ivanovdenis.ru

<https://github.com/denisivan0v>