



ВОЩИЛО ЮРИЙ

---

РАЗРАБОТКА КОРПОРАТИВНЫХ  
РЕШЕНИЙ С ИСПОЛЬЗОВАНИЕМ  
ТЕХНОЛОГИЙ JAVA

## 1. МЕТОДЫ ИНТЕРФЕЙСОВ ПО УМОЛЧАНИЮ

---

# МЕТОДЫ ИНТЕРФЕЙСОВ ПО УМОЛЧАНИЮ

Java 8 позволяет вам добавлять неабстрактные реализации методов в интерфейс, используя ключевое слово `default`. Эта фича также известна, как методы расширения:

```
package by.part6;

public class Example1 {

    interface Formula {
        double calculate(int a);

        default double sqrt(int a) {
            return Math.sqrt(a);
        }
    }

    public static void main(String[] args) {
        Formula formula = new Formula() {
            @Override
            public double calculate(int a) {
                return sqrt(a * 100);
            }
        };

        System.out.println(formula.calculate(100));      // 100.0
        System.out.println(formula.sqrt(16));           // 4.0
    }
}
```

## 2. ЛЯМБДА-ВЫРАЖЕНИЯ

---

# ЛЯМБДА-ВЫРАЖЕНИЯ

```
package by.part6;

import java.util.Arrays;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

public class Example2 {

    public static void main(String[] args) {
        //old style
        List<String> names = Arrays.asList("peter", "anna", "mike", "xenia");

        Collections.sort(names, new Comparator<String>() {
            @Override
            public int compare(String a, String b) {
                return b.compareTo(a);
            }
        });

        //java8 style 1
        Collections.sort(names, (String a, String b) -> {
            return b.compareTo(a);
        });

        //java8 style 2
        Collections.sort(names, (String a, String b) -> b.compareTo(a));

        //java8 style 3
        Collections.sort(names, (a, b) -> b.compareTo(a));
    }
}
```

### 3. ФУНКЦИОНАЛЬНЫЕ ИНТЕРФЕЙСЫ

---

## ФУНКЦИОНАЛЬНЫЕ ИНТЕРФЕЙСЫ

Как лямбда-выражения соответствуют системе типов языка Java? Каждой лямбде соответствует тип, представленный интерфейсом. Так называемый **функциональный интерфейс** должен содержать **ровно один абстрактный метод**. Каждое лямбда-выражение этого типа будет сопоставлено объявленному методу. Также, поскольку методы по умолчанию не являются абстрактными, вы можете добавлять в функциональный интерфейс сколько угодно таких методов.

Мы можем использовать какие угодно интерфейсы для лямбда-выражений, содержащие ровно один абстрактный метод. Для того, чтобы гарантировать, что ваш интерфейс отвечает этому требованию, используется аннотация `@FunctionalInterface`. Компилятор осведомлен об этой аннотации, и выдаст ошибку компиляции, если вы добавите второй абстрактный метод в функциональный интерфейс.

## 4. ФУНКЦИОНАЛЬНЫЕ ИНТЕРФЕЙСЫ

---

# ФУНКЦИОНАЛЬНЫЕ ИНТЕРФЕЙСЫ

```
package by.part6;

public class Example3 {

    @FunctionalInterface
    interface Converter<F, T> {
        T convert(F from);
    }

    public static void main(String[] args) {
        Converter<String, Integer> converter = (from) -> Integer.valueOf(from);
        Integer converted = converter.convert("123");
        System.out.println(converted); // 123
    }
}
```

## 5. ССЫЛКИ НА МЕТОДЫ И КОНСТРУКТОРЫ

---

# ССЫЛКИ НА МЕТОДЫ И КОНСТРУКТОРЫ

```
package by.part6;

public class Example4 {

    @FunctionalInterface
    interface Converter<F, T> {
        T convert(F from);
    }

    static class Something {
        String startsWith(String s) {
            return String.valueOf(s.charAt(0));
        }
    }

    static class Person {
        String firstName;
        String lastName;

        Person() {}

        Person(String firstName, String lastName) {
            this.firstName = firstName;
            this.lastName = lastName;
        }
    }

    interface PersonFactory<P extends Person> {
        P create(String firstName, String lastName);
    }

    public static void main(String[] args) {
        //Ссылки на методы интерфейса
        Converter<String, Integer> converter = Integer::valueOf;
        Integer converted = converter.convert("123");
        System.out.println(converted); // 123

        //Ссылки на методы класса
        Something something = new Something();
        Converter<String, String> converter2 =
        something::startsWith;
        String converted2 = converter2.convert("Java");
        System.out.println(converted2); // "J"

        //Ссылки на конструктор
        PersonFactory<Person> personFactory = Person::new;
        Person person = personFactory.create("Peter", "Parker");
        System.out.println(person.firstName + " " +
        person.lastName);
    }
}
```

## 6. ОБЛАСТИ ДЕЙСТВИЯ ЛЯМБД

---

# ОБЛАСТИ ДЕЙСТВИЯ ЛЯМБД

Доступ к переменным внешней области действия из лямбда-выражения очень схож к доступу из анонимных объектов. Вы можете ссылаться на переменные, объявленные как `final`, на экземплярные поля класса и статические переменные.

```
package by.part6;

import by.part6.Example4.Converter;

public class Example5 {

    public static void main(String[] args) {
        final int num = 1;
        Converter<Integer, String> stringConverter = (from) -> String.valueOf(from + num);

        stringConverter.convert(2);      // 3
    }
}
```

## 7. ВСТРОЕННЫЕ ФУНКЦИОНАЛЬНЫЕ ИНТЕРФЕЙСЫ

---

### ПРЕДИКАТЫ

Предикаты – это функции, принимающие один аргумент, и возвращающие значение типа boolean. Интерфейс содержит различные методы по умолчанию, позволяющие строить сложные условия (`and`, `or`, `negate`).

```
package by.part6;

import java.util.Objects;
import java.util.function.Predicate;

public class Example6 {

    public static void main(String[] args) {
        Predicate<String> predicate = (s) -> s.length() > 0;

        System.out.println(predicate.test("foo"));           // true
        System.out.println(predicate.negate().test("foo")); // false

        Predicate<Boolean> nonNull = Objects::nonNull;
        Predicate<Boolean>isNull = Objects::isNull;

        Predicate<String> isEmpty = String::isEmpty;
        Predicate<String> isNotEmpty = isEmpty.negate();

    }
}
```

## 7. ВСТРОЕННЫЕ ФУНКЦИОНАЛЬНЫЕ ИНТЕРФЕЙСЫ

---

### ФУНКЦИИ, ПОСТАВЩИКИ И ПОТРЕБИТЕЛИ

Функции принимают один аргумент и возвращают некоторый результат. Методы по умолчанию могут использоваться для построения цепочек вызовов (`compose`, `andThen`).

Поставщики (`suppliers`) предоставляют результат заданного типа. В отличии от функций, поставщики не принимают аргументов.

Потребители (`consumers`) представляют собой операции, которые производятся на одним входным аргументом.

Компараторы хорошо известны по предыдущим версиям Java. Java 8 добавляет в интерфейс различные методы по умолчанию.

## 7. ВСТРОЕННЫЕ ФУНКЦИОНАЛЬНЫЕ ИНТЕРФЕЙСЫ

---

# ФУНКЦИИ, ПОСТАВЩИКИ И ПОТРЕБИТЕЛИ

```
package by.part6;

import by.part6.Example4.Person;
import java.util.function.Consumer;
import java.util.function.Function;
import java.util.function.Supplier;

public class Example7 {

    public static void main(String[] args) {
        //Функции
        Function<String, Integer> toInteger = Integer::valueOf;
        Function<String, String> backToString = toInteger.andThen(String::valueOf);
        System.out.println(backToString.apply("123")); // "123"

        //Поставщики
        Supplier<Person> personSupplier = Person::new;
        Person person = personSupplier.get(); // new Person
        person.print();

        //Потребители
        Consumer<Person> greeter = (p) -> System.out.println("Hello, " + p.firstName);
        greeter.accept(new Person("Luke", "Skywalker"));
    }
}
```

## 8. ОПЦИОНАЛЬНЫЕ ЗНАЧЕНИЕ

---

# ОПЦИОНАЛЬНЫЕ ЗНАЧЕНИЕ

Опциональные значение – это по сути контейнер для значения, которое может быть равно null. Например, вам нужен метод, который возвращает какое-то значение, но иногда он должен возвращать пустое значение. Вместо того, чтобы возвращать null, в Java 8 вы можете вернуть опциональное значение.

```
package by.part6;

import java.util.Optional;

public class Example8 {

    public static void main(String[] args) {
        Optional<String> optional = Optional.of("bam");

        optional.isPresent();           // true
        optional.get();                // "bam"
        optional.orElse("fallback");   // "bam"

        optional.ifPresent(s -> System.out.println(s.charAt(0))); // "b"
    }
}
```

# ПОТОКИ

Тип `java.util.Stream` представляет собой последовательность элементов, над которой можно производить различные операции. Операции над потоками бывают или промежуточными (*intermediate*) или конечными (*terminal*).

Конечные операции возвращают результат определенного типа, а промежуточные операции возвращают тот же поток.

Таким образом вы можете строить цепочки из нескольких операций над одним и тем же потоком.

Поток создаются на основе источников, например типов, реализующих `java.util.Collection`, такие как списки или множества (ассоциативные массивы не поддерживаются). Операции над потоками могут выполняться как последовательно, так и параллельно.

# ПОТОКИ

```
package by.part6;

import java.util.Arrays;
import java.util.List;
import java.util.stream.Stream;

public class Example9 {

    public static void main(String[] args) {
        List<String> stringCollection = Arrays.asList(
            "ddd2",
            "aaa2",
            "bbb1",
            "aaa1",
            "bbb3",
            "ccc",
            "bbb2",
            "ddd1"
        );

        Stream<String> stream = stringCollection.stream();
        Stream<String> stream1 = Stream.of("ddd2",
            "aaa2",
            "bbb1",
            "aaa1",
            "bbb3",
            "ccc",
            "bbb2",
            "ddd1"
        );
    }
}
```

### FILTER

Операция Filter принимает предикат, который фильтрует все элементы потока. Эта операция является промежуточной, т.е. позволяет нам вызвать другую операцию (например, forEach) над результатом. ForEach принимает функцию, которая вызывается для каждого элемента в (уже отфильтрованном) потоке. ForEach является конечной операцией. Она не возвращает никакого значения, поэтому дальнейший вызов потоковых операций невозможен.

### SORTED

Операция `Sorted` является промежуточной операцией, которая возвращает отсортированное представление потока. Элементы сортируются в обычном порядке, если вы не предоставили свой компаратор.

Помните, что `sorted` создает всего лишь отсортированное представление и не влияет на порядок элементов в исходной коллекции. Порядок строк в `stringCollection` остается нетронутым:

### MAP

Промежуточная операция `map` преобразовывает каждый элемент в другой объект при помощи переданной функции. Вы можете использовать `map` для преобразования каждого объекта в объект другого типа. Тип результирующего потока зависит от типа функции, которую вы передаете при вызове `map`.

### МАТСН

Для проверки, удовлетворяет ли поток заданному предикату, используются различные операции сопоставления (match). Все операции сопоставления являются конечными и возвращают результат типа boolean.

### COUNT

Операция Count является конечной операцией и возвращает количество элементов в потоке. Типом возвращаемого значения является long.

## 10. АССОЦИАТИВНЫЕ МАССИВЫ

---

# АССОЦИАТИВНЫЕ МАССИВЫ

putIfAbsent позволяет нам не писать дополнительные проверки на null; forEach принимает потребителя, который производит операцию над каждым элементом массива.

```
package by.part6;

import java.util.HashMap;
import java.util.Map;

public class Example11 {

    public static void main(String[] args) {
        Map<Integer, String> map = new HashMap<>();

        for (int i = 0; i < 10; i++) {
            map.putIfAbsent(i, "val" + i);
        }

        map.forEach((id, val) -> System.out.println(val));
    }
}
```

## 10. АССОЦИАТИВНЫЕ МАССИВЫ

---

# АССОЦИАТИВНЫЕ МАССИВЫ

код показывает как использовать для вычислений код при помощи различных функций:

```
map.computeIfPresent(3, (num, val) -> val + num);
map.get(3);                      // val33

map.computeIfPresent(9, (num, val) -> null);
map.containsKey(9);               // false

map.computeIfAbsent(23, num -> "val" + num);
map.containsKey(23);              // true

map.computeIfAbsent(3, num -> "bam");
map.get(3);                      // val33
```

## 10. АССОЦИАТИВНЫЕ МАССИВЫ

---

# АССОЦИАТИВНЫЕ МАССИВЫ

как удалить объект по ключу, только если этот объект ассоциирован с ключом:

```
map.remove(3, "val3");
map.get(3);           // val3

map.remove(3, "val3");
map.get(3);           // null
System.out.println(map);
```

еще один полезный метод:

```
map.getOrDefault(42, "not found"); // not found
```

## 10. АССОЦИАТИВНЫЕ МАССИВЫ

---

# ОБЪЕДИНИТЬ ЗАПИСИ ДВУХ МАССИВОВ

```
map.merge(9, "val9", (value, newValue) -> value.concat(newValue));  
map.get(9); // val9
```

```
map.merge(9, "concat", (value, newValue) -> value.concat(newValue));  
map.get(9); // val9concat
```

## 10. АССОЦИАТИВНЫЕ МАССИВЫ

---

# СВОЙ COLLECTOR

```
public interface Collector<T, A, R> { ... }
```

- **T** - тип объектов, которые будут доступны для сбора,
- **A** - тип изменяемого объекта-аккумулятора,
- **R** - тип конечного результата.

```
Supplier<A> supplier();
BiConsumer<A, T> accumulator();
BinaryOperator<A> combiner();
Function<A, R> finisher();
Set<Characteristics> characteristics();
```

## 10. АССОЦИАТИВНЫЕ МАССИВЫ

---

# СВОЙ COLLECTOR

**Метод supplier ()** возвращает экземпляр *Supplier*, который генерирует пустой экземпляр аккумулятора

```
@Override  
public Supplier<BigDecimal[]> supplier() {  
    return () → new BigDecimal[]{BigDecimal.ZERO, BigDecimal.ZERO};  
}
```

**Аккумулятор ()** метод возвращает функцию, которая используется для добавления нового элемента в существующем *аккумуляторном* объект.

```
@Override  
public BiConsumer<BigDecimal[], BigDecimal> accumulator() {  
    return (acc, b) → {  
        acc[0] = acc[0].add(b);  
        acc[1] = acc[1].add(BigDecimal.ONE);  
    };  
}
```

**Метод combiner ()** возвращает функцию, которая используется для объединения двух аккумуляторов вместе:

```
@Override  
public BinaryOperator<BigDecimal[]> combiner() {  
    return (acc1, acc2) → new BigDecimal[]{  
        acc1[0].add(acc2[0]),  
        acc1[1].add(acc2[1])  
    };  
}
```

**Метод finisher ()** метод возвращает функцию, которая используется для преобразования аккумулятора до конечного типа результата

```
@Override  
public Function<BigDecimal[], BigDecimal> finisher() {  
    return acc → acc[1].intValue() > 0 ? acc[0].divide(acc[1], BigDecimal.ROUND_HALF_UP) : BigDecimal.ZERO;  
}
```

## 10. АССОЦИАТИВНЫЕ МАССИВЫ

---

# СВОЙ COLLECTOR

Метод `characteristics()` используется для предоставления Stream некоторой дополнительной информации, которая будет использоваться для внутренней оптимизации.

```
enum Characteristics {
    /**
     * Indicates that this collector is <em>concurrent</em>, meaning that
     * the result container can support the accumulator function being
     * called concurrently with the same result container from multiple
     * threads.
     *
     * <p>If a {@code CONCURRENT} collector is not also {@code UNORDERED},
     * then it should only be evaluated concurrently if applied to an
     * unordered data source.
     */
    CONCURRENT,

    /**
     * Indicates that the collection operation does not commit to preserving
     * the encounter order of input elements. (This might be true if the
     * result container has no intrinsic order, such as a {@link Set}.)
     */
    UNORDERED,

    /**
     * Indicates that the finisher function is the identity function and
     * can be elided. If set, it must be the case that an unchecked cast
     * from A to R will succeed.
     */
    IDENTITY_FINISH
}

@Override
public Set<Characteristics> characteristics() {
    return Collections.unmodifiableSet(EnumSet.of(Characteristics.UNORDERED, Characteristics.CONCURRENT));
}
```

## 10. ТЕСТИРОВАНИЕ

---

# JUNIT JUPITER

```
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <version>5.7.1</version>
    <scope>test</scope>
</dependency>
```

```
dependencies {
    testImplementation 'org.junit.jupiter:junit-jupiter-engine:5.7.1'
}

test {
    useJUnitPlatform()
}
```

# JUNIT JUPITER

- `@TestFactory` - обозначает метод, являющийся тестовой фабрикой для динамических тестов.
- `@DisplayName` - определяет настраиваемое отображаемое имя для тестового класса или тестового метода
- `@Nested` - обозначает, что аннотированный класс является вложенным нестатическим тестовым классом.
- `@Tag` - объявляет теги для фильтрации тестов
- `@ExtendWith` - используется для регистрации пользовательских расширений
- `@BeforeEach` - обозначает, что аннотированный метод будет выполняться перед каждым тестовым методом (ранее `@Before`)
- `@AfterEach` - обозначает, что аннотированный метод будет выполняться после каждого тестового метода (ранее `@After`)
- `@BeforeAll` - означает, что аннотированный метод будет выполнен перед всеми тестовыми методами в текущем классе (ранее `@BeforeClass`)
- `@AfterAll` - означает, что аннотированный метод будет выполнен после всех тестовых методов в текущем классе (ранее `@AfterClass`)
- `@Disable` - используется для отключения тестового класса или метода (ранее `@Ignore`)

## 10. ТЕСТИРОВАНИЕ

---

# JUNIT JUPITER

```
@BeforeAll
static void setup() {
    log.info("@BeforeAll - executes once before all test methods in this class");
}

@BeforeEach
void init() {
    log.info("@BeforeEach - executes before each test method in this class");
}

@AfterEach
void tearDown() {
    log.info("@AfterEach - executed after each test method.");
}

@AfterAll
static void done() {
    log.info("@AfterAll - executed after all test methods.");
}
```

## 10. ТЕСТИРОВАНИЕ

---

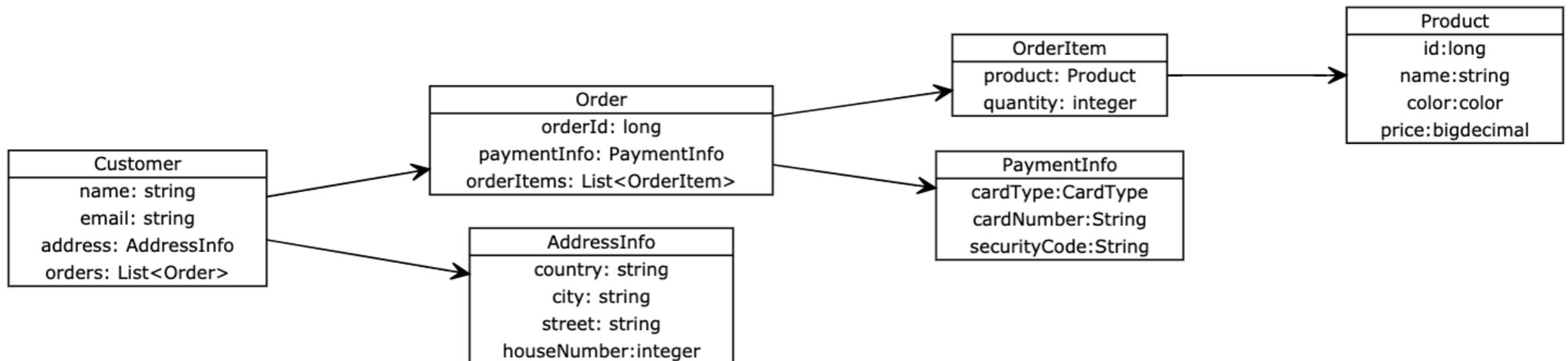
# JUNIT JUPITER

```
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotEquals;
import static org.junit.jupiter.api.Assertions.assertArrayEquals;
import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertTrue;
import static org.junit.jupiter.api.Assertions.assertNull;
import static org.junit.jupiter.api.Assertions.assertThrows;
```

```
@Test
void lambdaExpressions() {
    assertTrue(Stream.of(1, 2, 3)
        .mapToInt(i → i)
        .sum() > 5, () → "Sum should be greater than 5");
}
```

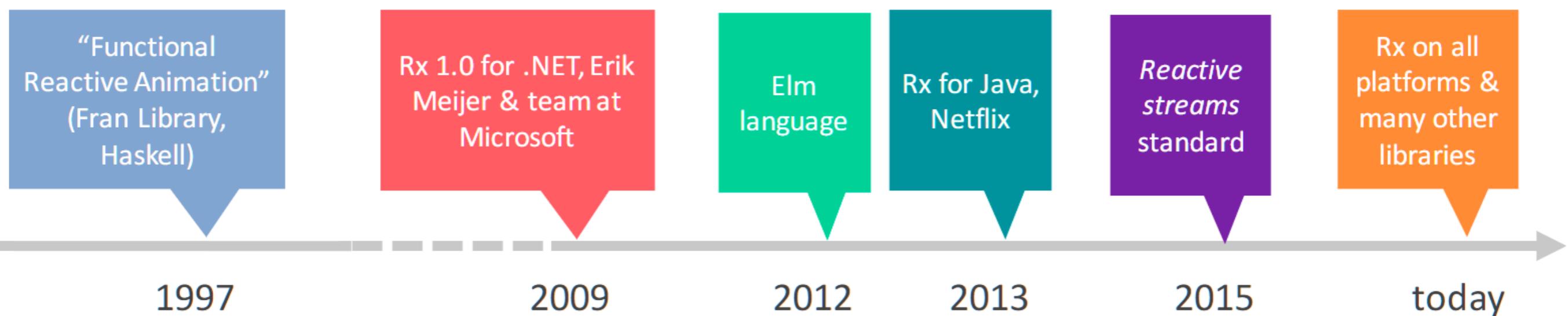
```
@Test
void groupAssertions() {
    int[] numbers = {0, 1, 2, 3, 4};
    assertAll("numbers",
        () → assertEquals(numbers[0], 1),
        () → assertEquals(numbers[3], 3),
        () → assertEquals(numbers[4], 1)
    );
}
```

# JUNIT JUPITER



CREATED WITH YUML

# РЕАКТИВНОЕ ПРОГРАММИРОВАНИЕ

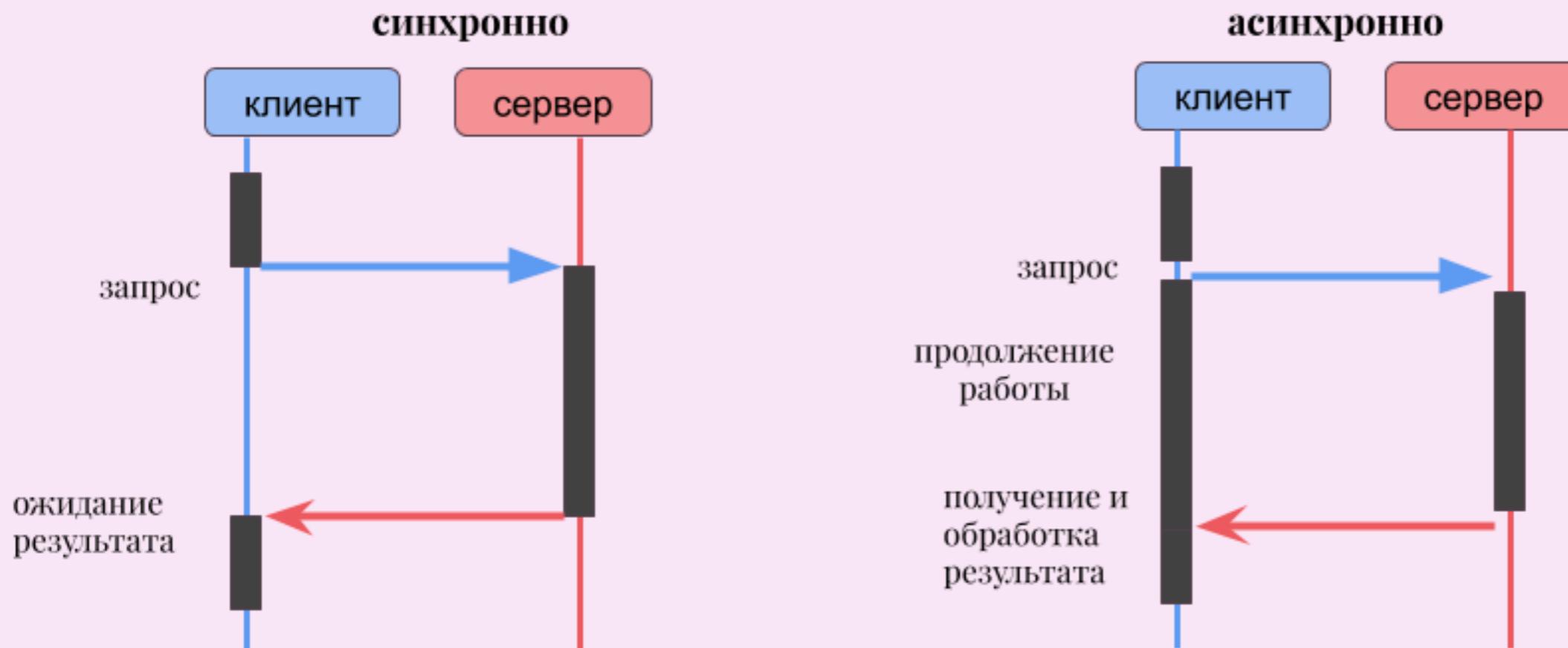


# РЕАКТИВНОЕ ПРОГРАММИРОВАНИЕ

РЕАКТИВНОЕ ПРОГРАММИРОВАНИЕ —  
ЭТО ПАРАДИГМА АСИНХРОННОГО  
ПРОГРАММИРОВАНИЯ, СВЯЗАННАЯ С  
ПОТОКАМИ ДАННЫХ И  
РАСПРОСТРАНЕНИЕМ ИЗМЕНЕНИЙ

## 10. РЕАКТИВНОЕ ПРОГРАММИРОВАНИЕ

### Асинхронность



# РЕАКТИВНОЕ ПРОГРАММИРОВАНИЕ

```
<dependency>
  <groupId>io.projectreactor</groupId>
  <artifactId>reactor-core</artifactId>
  <version>3.4.5</version>
</dependency>
```

```
implementation 'io.projectreactor:reactor-core:3.4.5'
```

# РЕАКТИВНОЕ ПРОГРАММИРОВАНИЕ

Reactor – это полностью неблокирующая основа реактивного программирования для JVM с эффективным управлением требованиями (в форме управления «противодавлением»).

напрямую интегрируется с функциональными API-интерфейсами Java 8, в частности, `CompletableFuture`, `Stream` и `Duration`

предлагает составные API-интерфейсы асинхронной последовательности – `Flux` (для элементов [N]) и `Mono` (для элементов [1]) – и широко реализует спецификацию `Reactive Streams`.

# РЕАКТИВНОЕ ПРОГРАММИРОВАНИЕ

Реактивные типы не предназначены для обработки запросов или данных быстрее.

Смысл заключается в их способности одновременно обслуживать больше запросов и более эффективно обрабатывать операции с задержкой, такие как запрос данных с удаленного сервера.

# РЕАКТИВНОЕ ПРОГРАММИРОВАНИЕ

Они позволяют обеспечить лучшее качество обслуживания и предсказуемое планирование пропускной способности, изначально имея дело со временем и задержкой, не затрачивая больше ресурсов.

В отличие от традиционной обработки, которая блокирует текущий поток во время ожидания результата, Reactive API запрашивает только тот объем данных, который он способен обработать и предоставляет новые возможности, поскольку он имеет дело с потоком данных, а не с отдельными элементами (объектами).

# РЕАКТИВНОЕ ПРОГРАММИРОВАНИЕ

Reactor – это реализация Reactive Streams, которая дополнительно расширяет базовый контракт Reactive Streams Publisher с типами API-интерфейсов, которые можно комбинировать с Flux и Mono.

Reactive Streams (Реактивные Потоки) состоят из 4-х простых Java – интерфейсов (Publisher, Subscriber, Subscription и Processor).

# РЕАКТИВНОЕ ПРОГРАММИРОВАНИЕ

```
public static interface Publisher<T> {  
    public void subscribe(Subscriber<? super T> subscriber);  
}  
public static interface Subscriber<T> {  
    public void onSubscribe(Subscription subscription);  
    public void onNext(T item);  
    public void onError(Throwable throwable);  
    public void onComplete();  
}  
  
public static interface Subscription {  
    public void request(long n);  
    public void cancel();  
}  
  
public static interface Processor<T,R> extends Subscriber<T>, Publisher<R> {  
}
```

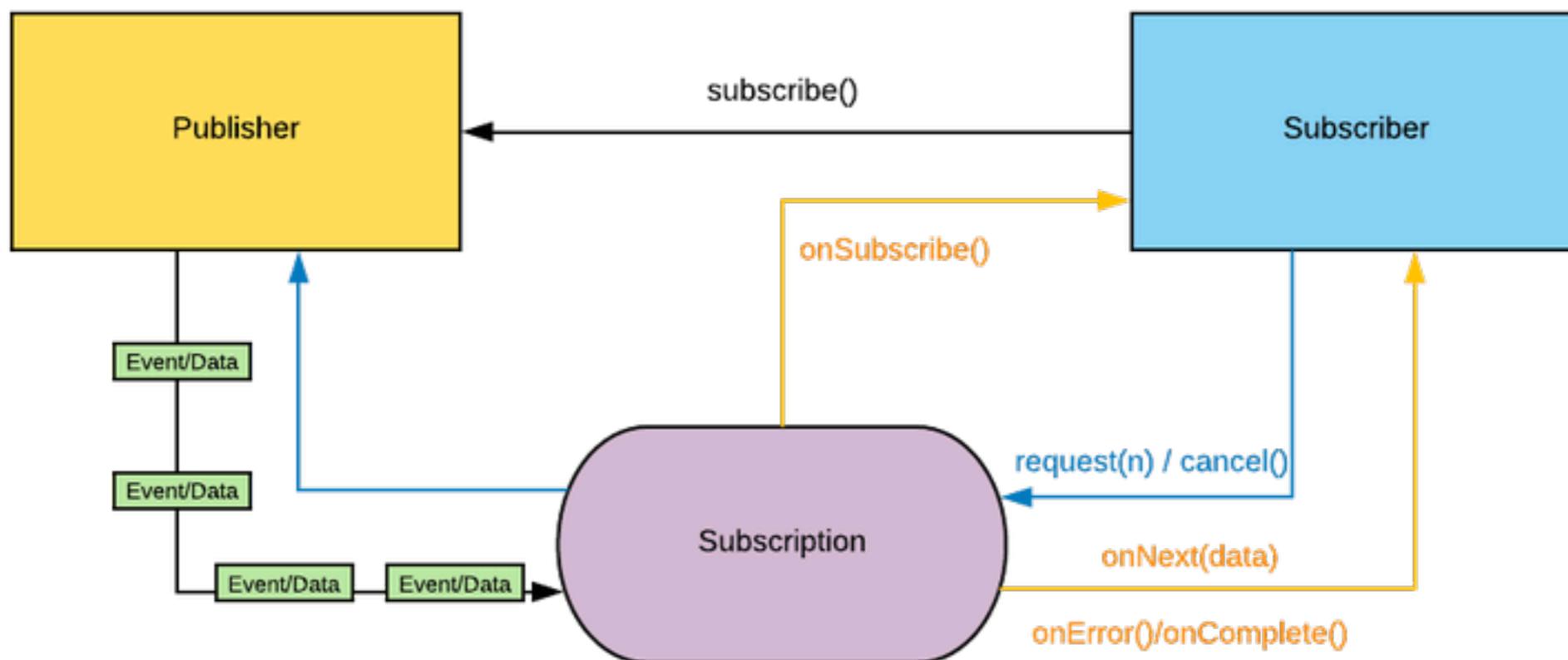
# РЕАКТИВНОЕ ПРОГРАММИРОВАНИЕ

ASYNC – асинхронность

NIO – “неблокируемость” ввода/вывода

RESPECT BACKPRESSURE – умение обрабатывать ситуации, когда данные появляются быстрее, чем потребляются

# РЕАКТИВНОЕ ПРОГРАММИРОВАНИЕ



### PUBLISHER

**Publisher** выдаёт какие-то данные (материалы). Данные идут по цепочке из операторов (конвейерной ленте), обрабатываются, в конце получается готовый продукт, который передаётся в нужный Consumer/Subscriber и употребляется уже там.

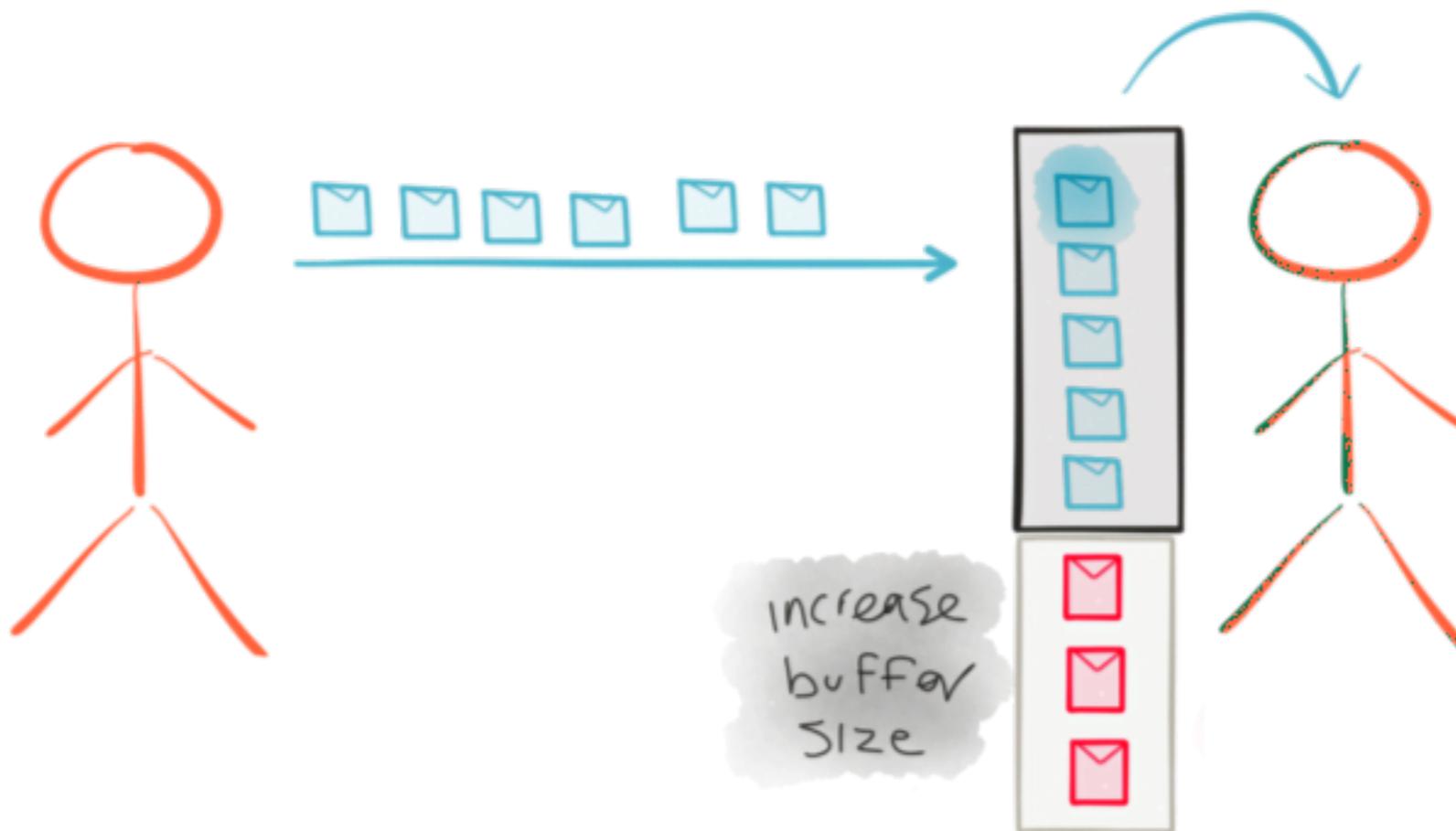
# BACKPRESSURE

**Backpressure** — это механизм, который позволяет получателю спрашивать, сколько данных он хочет получить.

Т.е. получатель начинает получать данные только тогда, когда он готов их обработать.

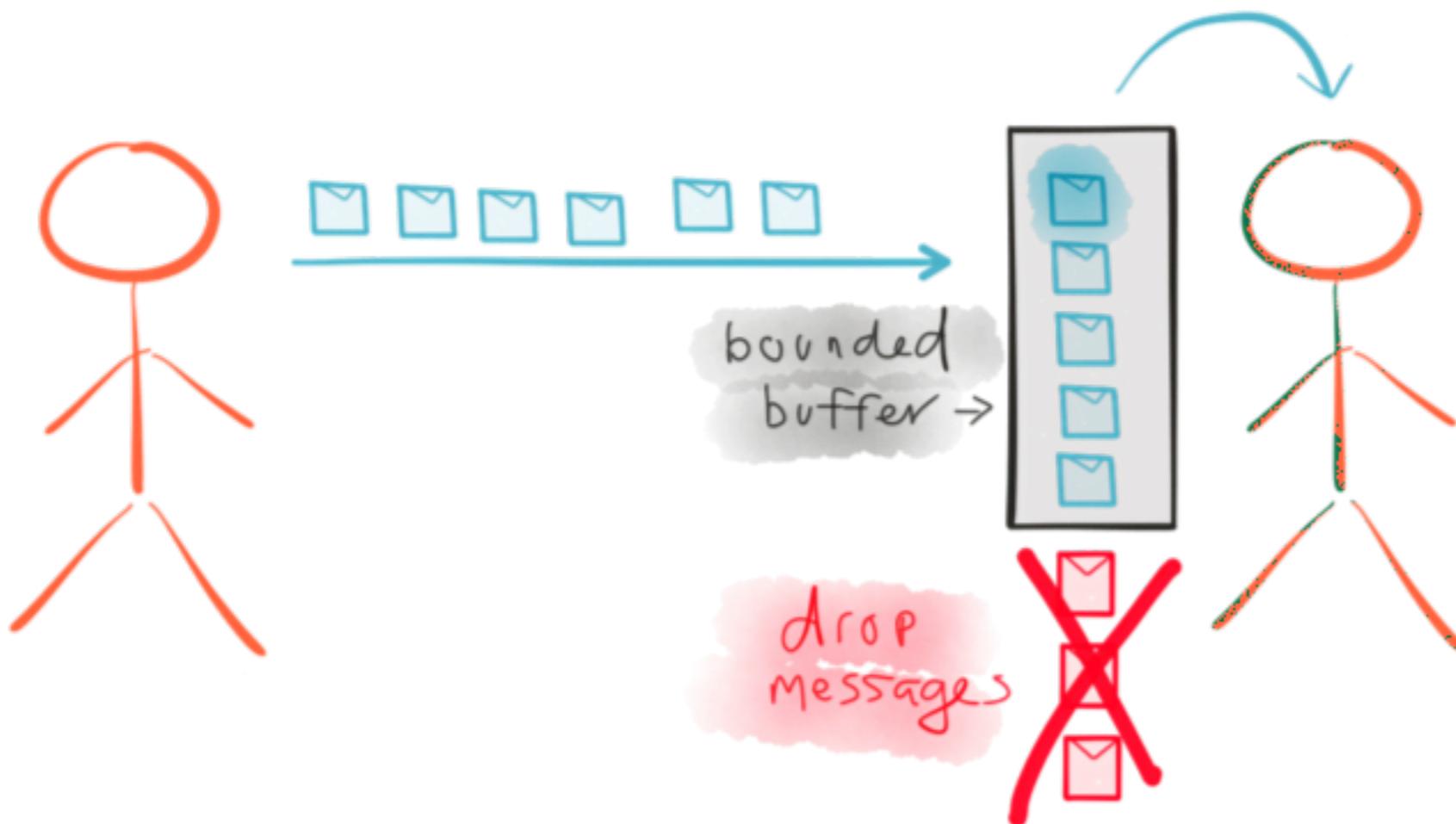
## 10. РЕАКТИВНОЕ ПРОГРАММИРОВАНИЕ

---



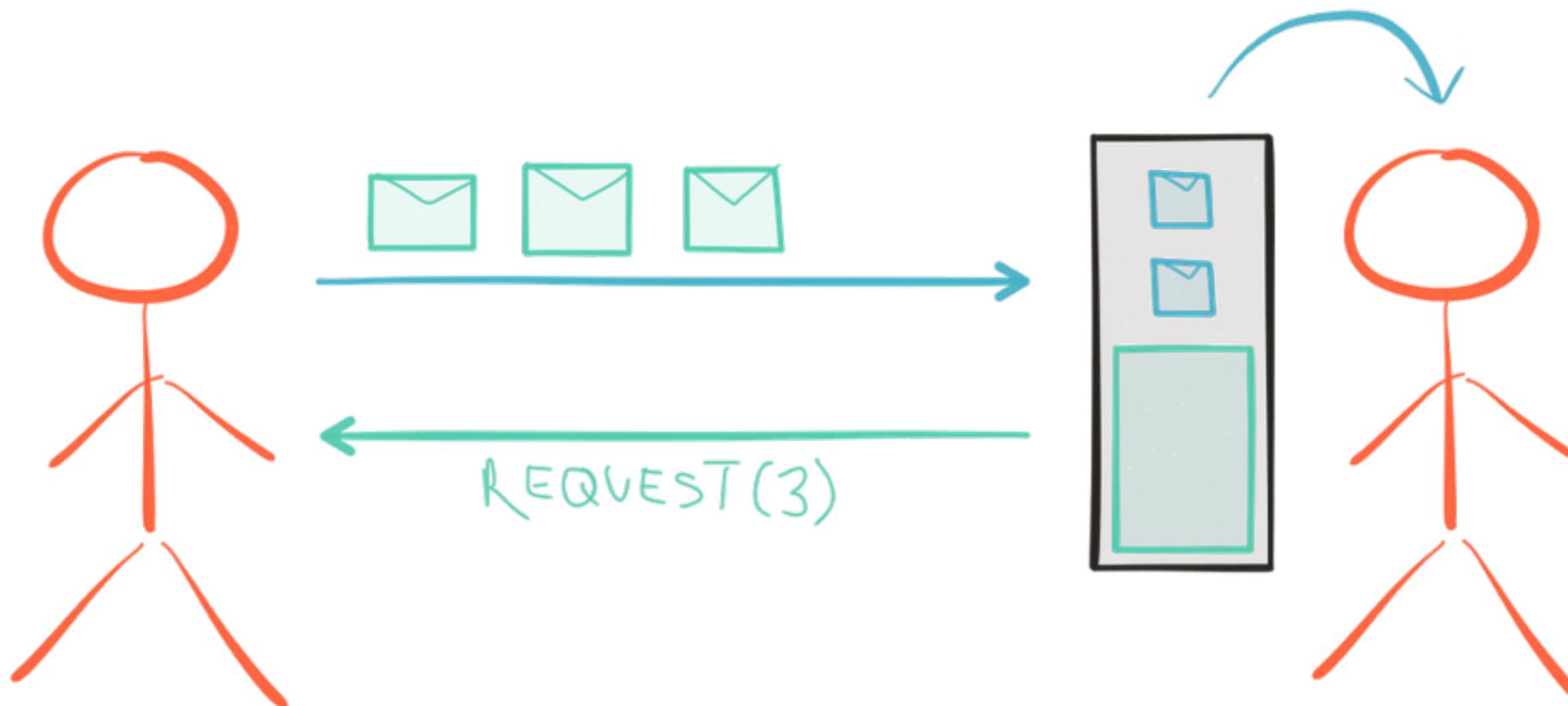
## 10. РЕАКТИВНОЕ ПРОГРАММИРОВАНИЕ

---



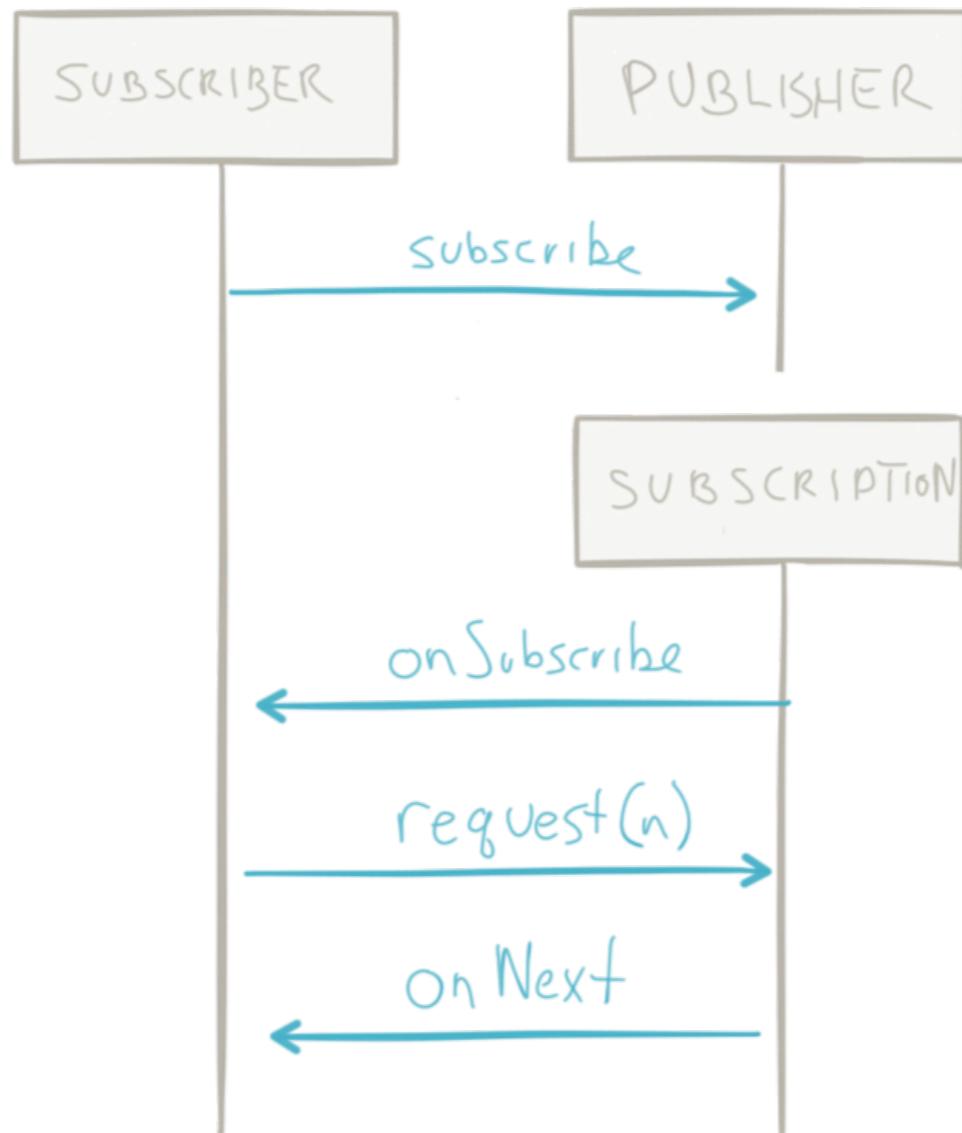
## 10. РЕАКТИВНОЕ ПРОГРАММИРОВАНИЕ

---



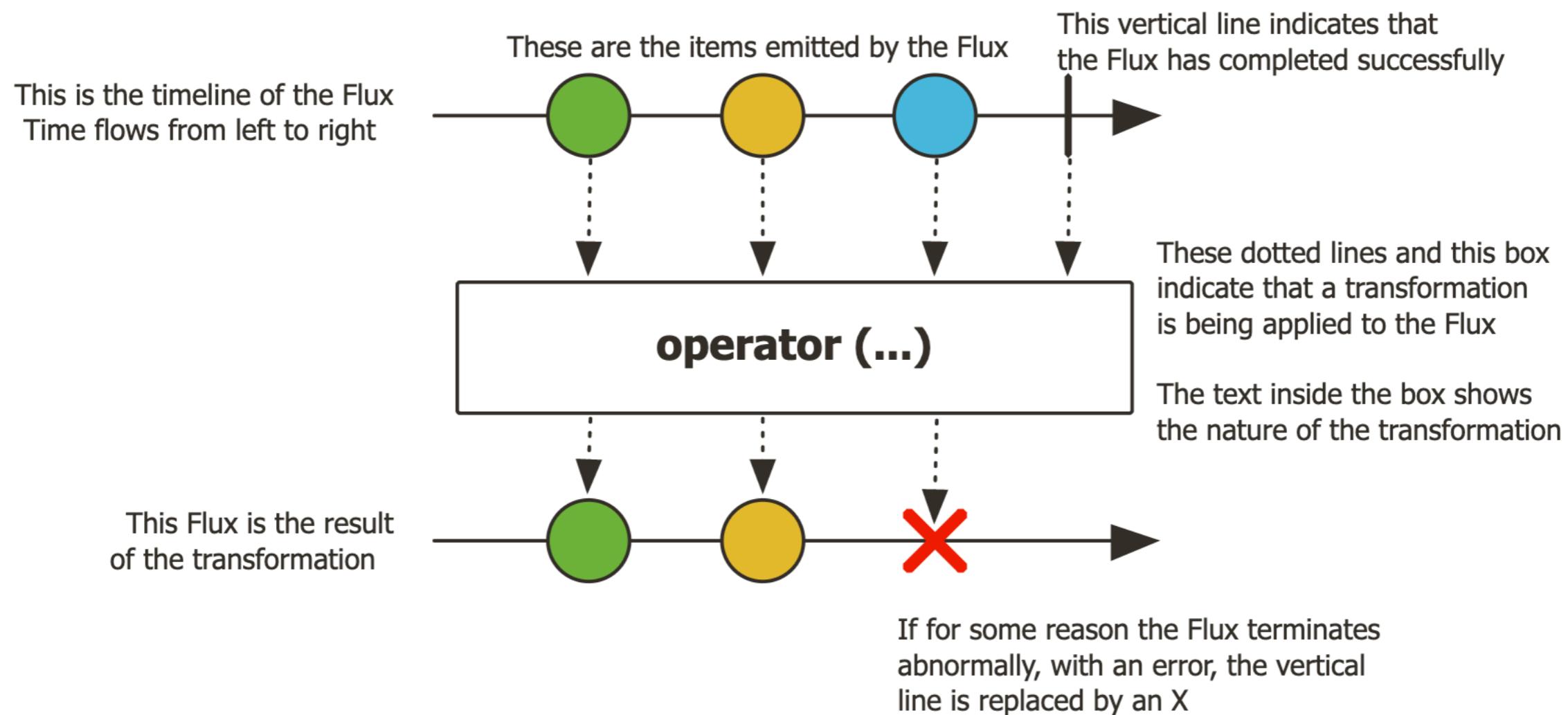
## 10. РЕАКТИВНОЕ ПРОГРАММИРОВАНИЕ

---



## 10. РЕАКТИВНОЕ ПРОГРАММИРОВАНИЕ

---



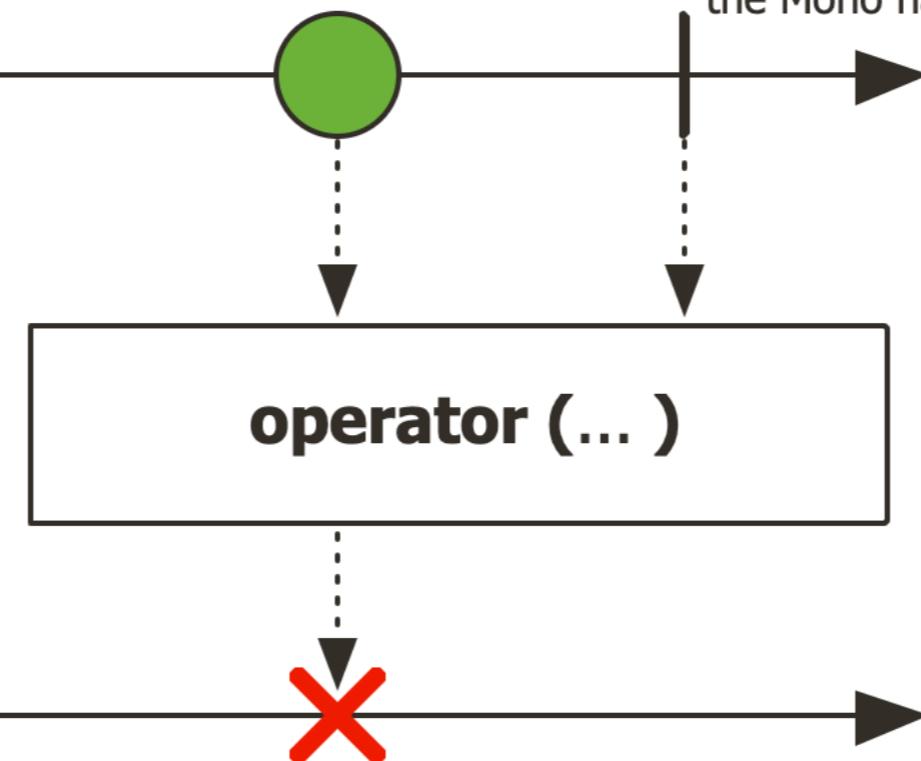
## 10. РЕАКТИВНОЕ ПРОГРАММИРОВАНИЕ

---

This is the timeline of the Mono  
Time flows from left to right

This is the optional item emitted by the Mono

This vertical line indicates that the Mono has completed successfully

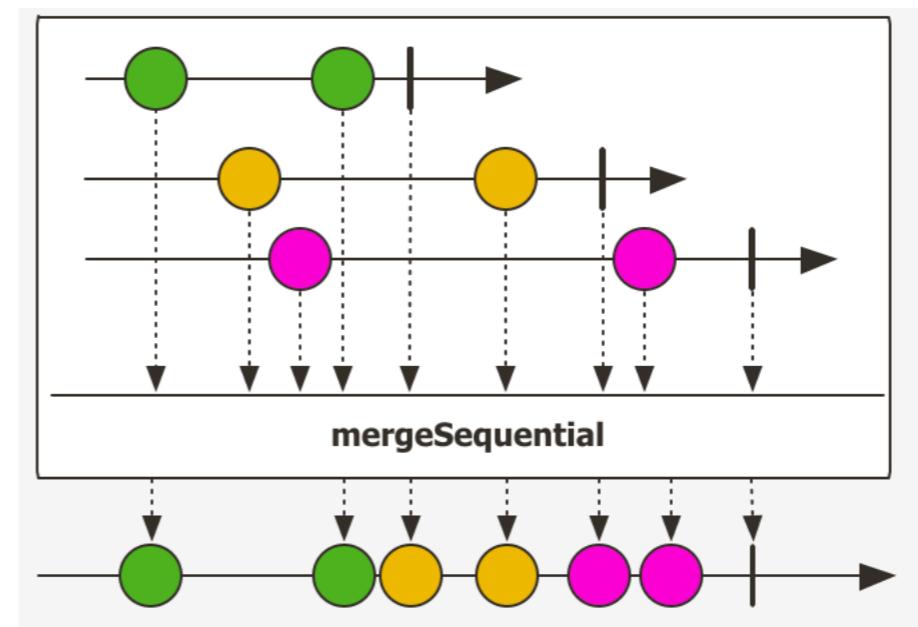
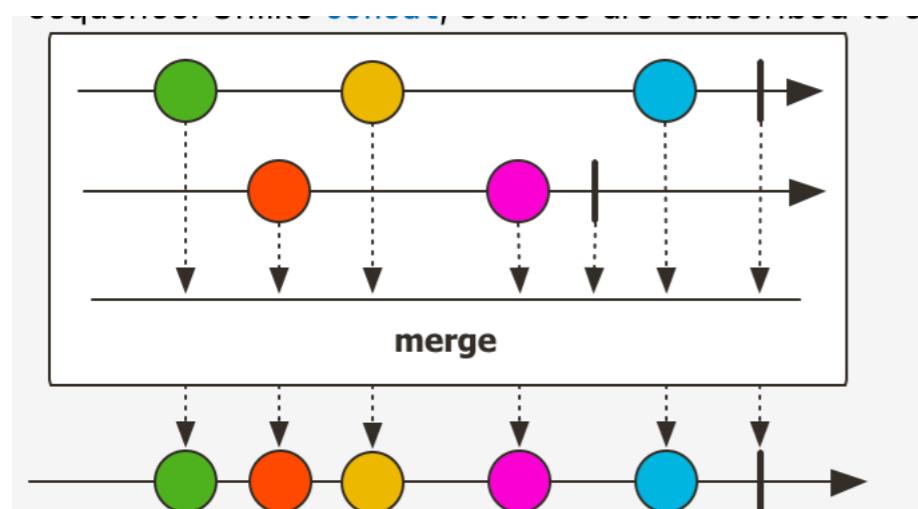
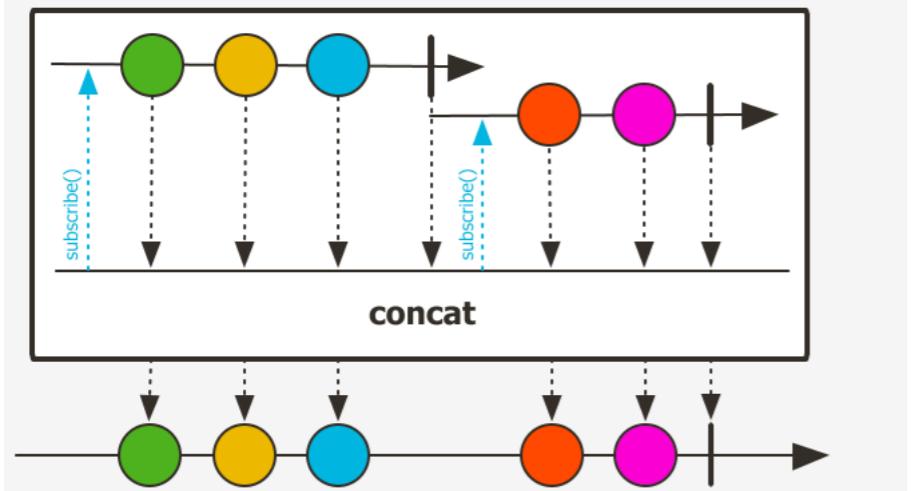


This Mono is the result of the transformation

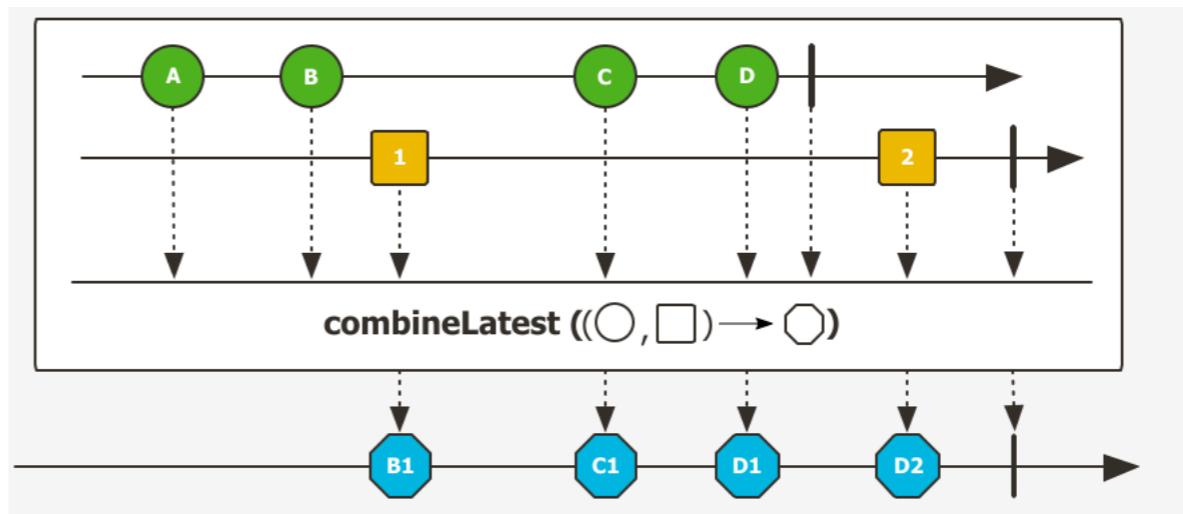
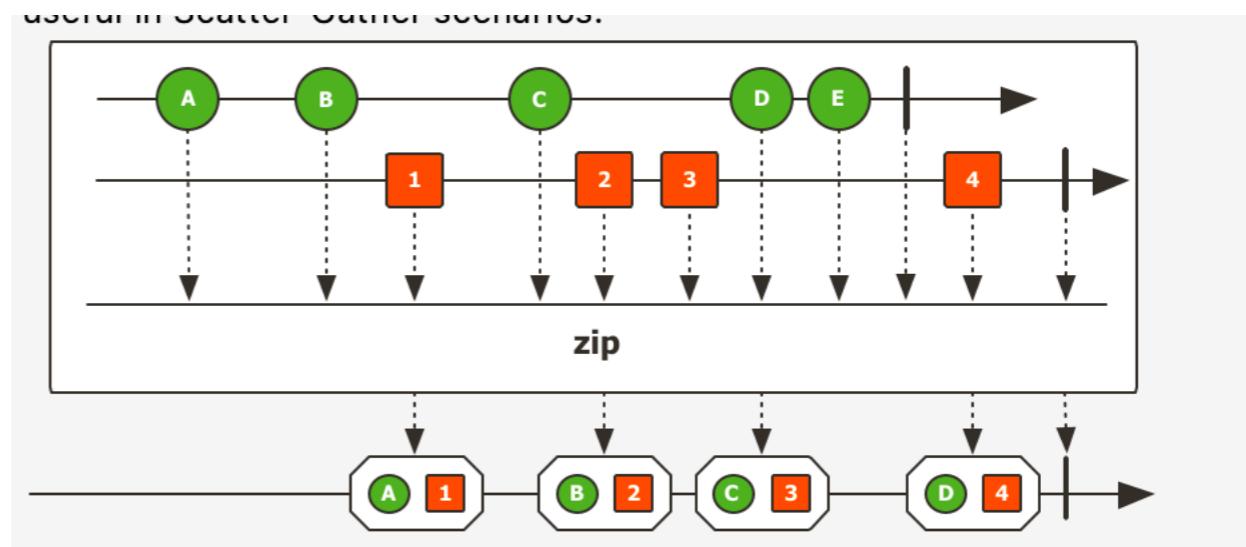
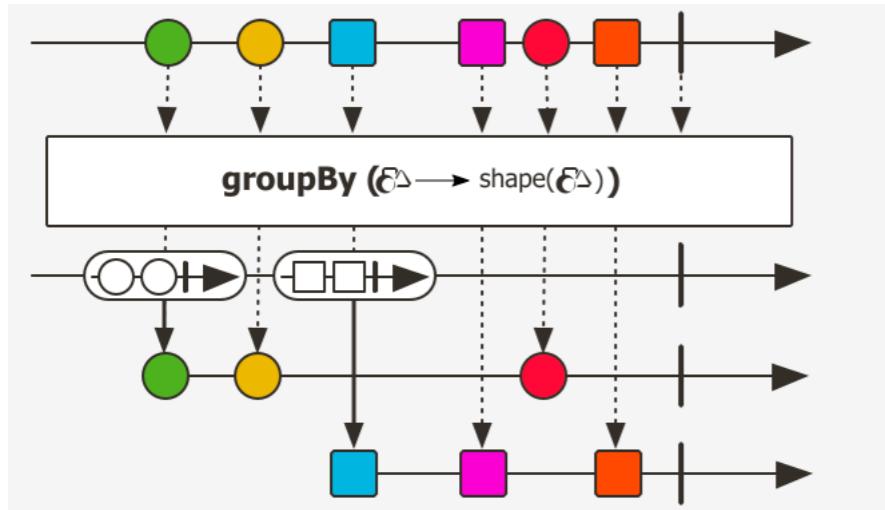
If for some reason the Mono terminates abnormally, with an error, the vertical line is replaced by an X

## 10. РЕАКТИВНОЕ ПРОГРАММИРОВАНИЕ

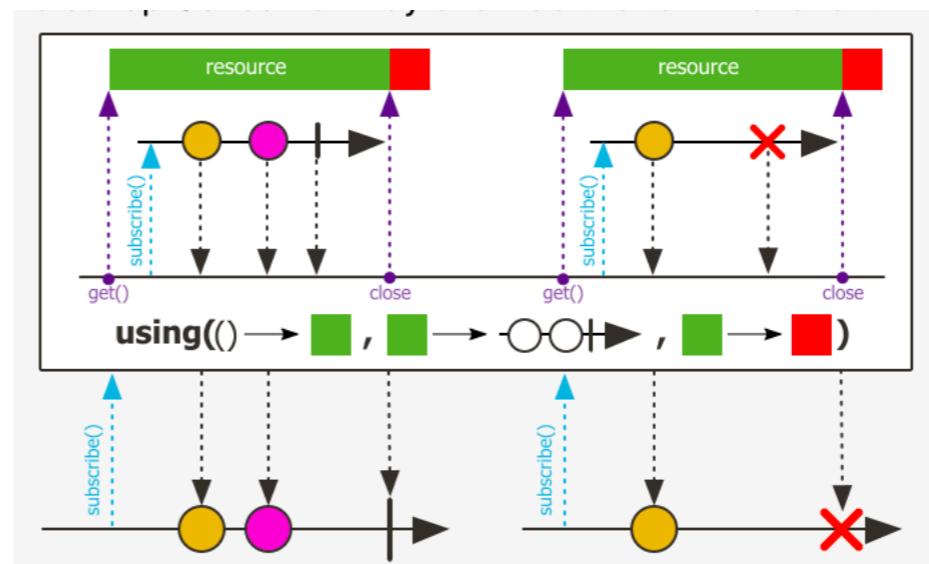
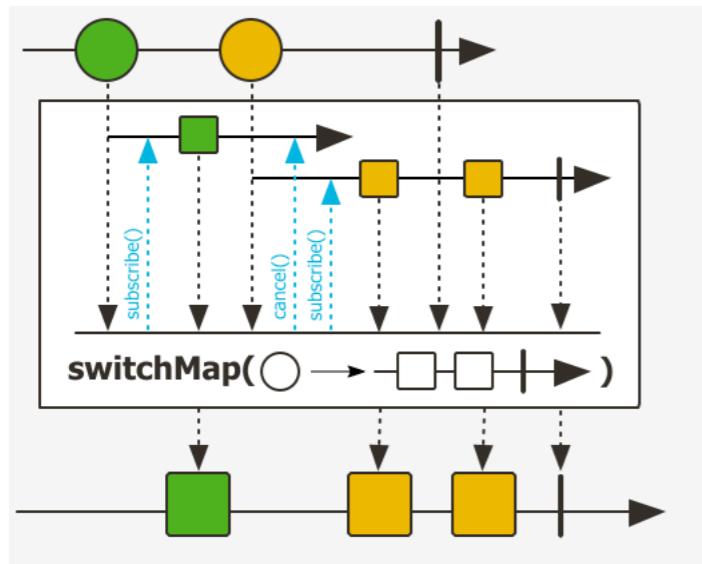
interrupts the sequence immediately and is forwarded



## 10. РЕАКТИВНОЕ ПРОГРАММИРОВАНИЕ

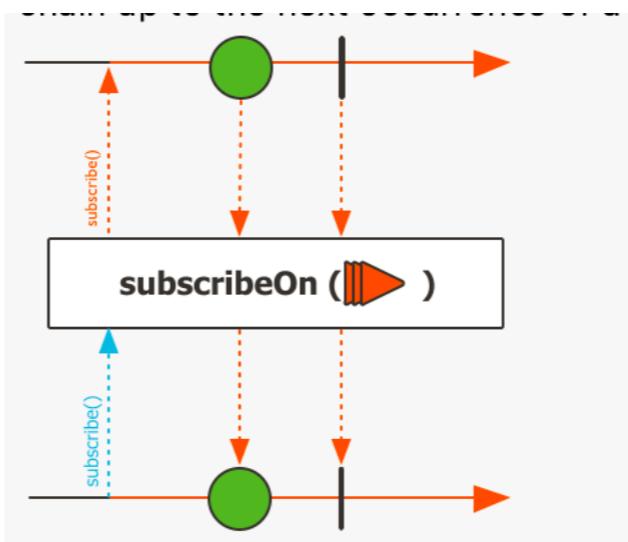
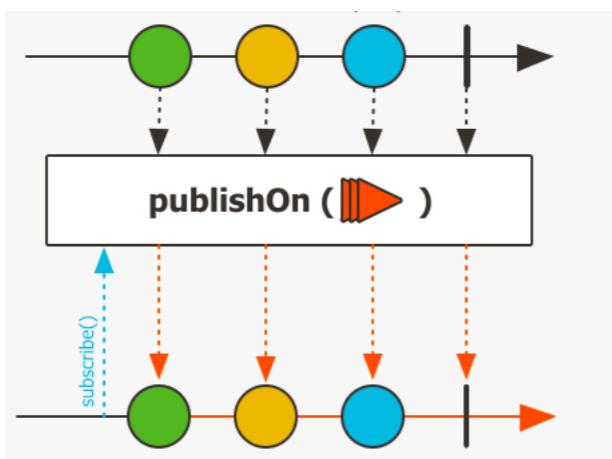


## 10. РЕАКТИВНОЕ ПРОГРАММИРОВАНИЕ

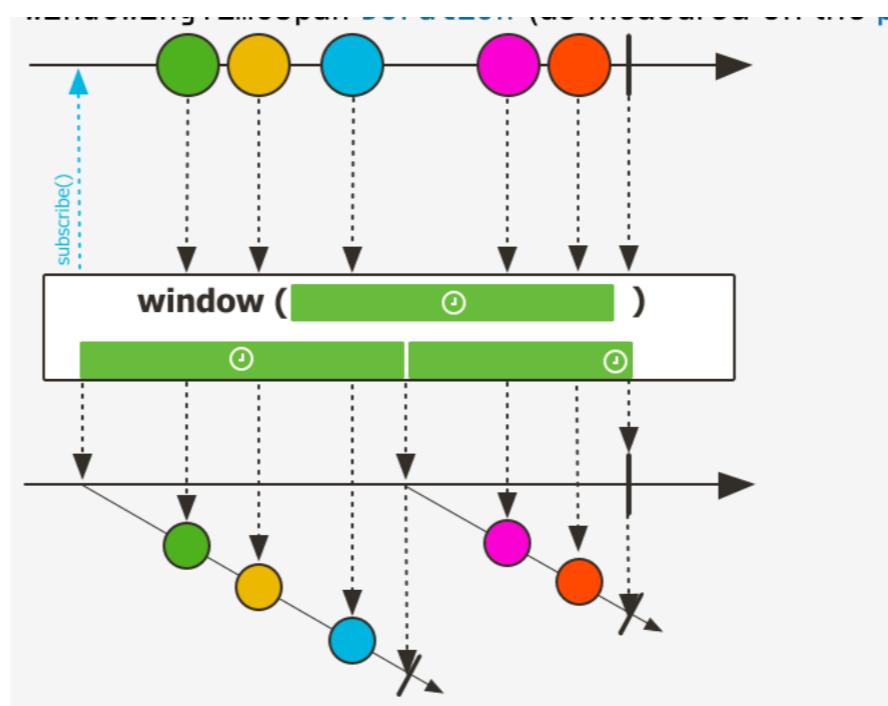
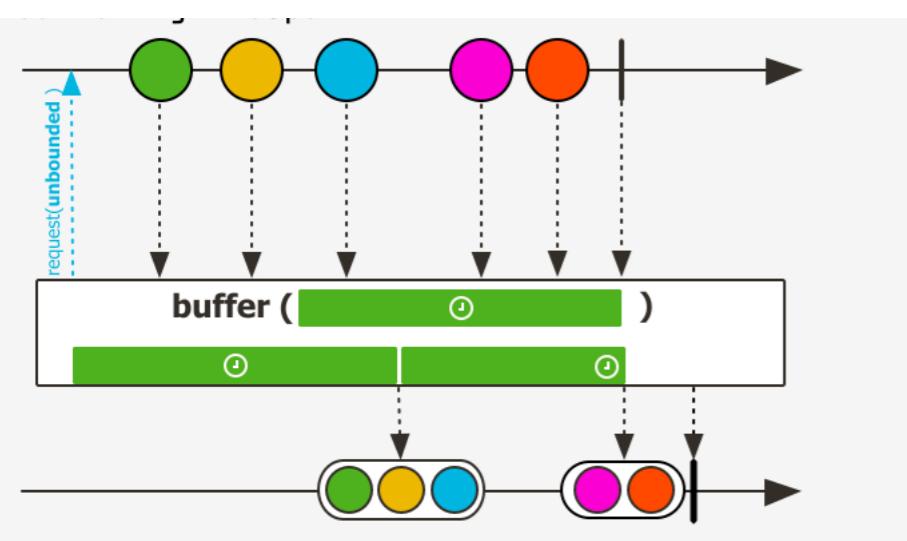
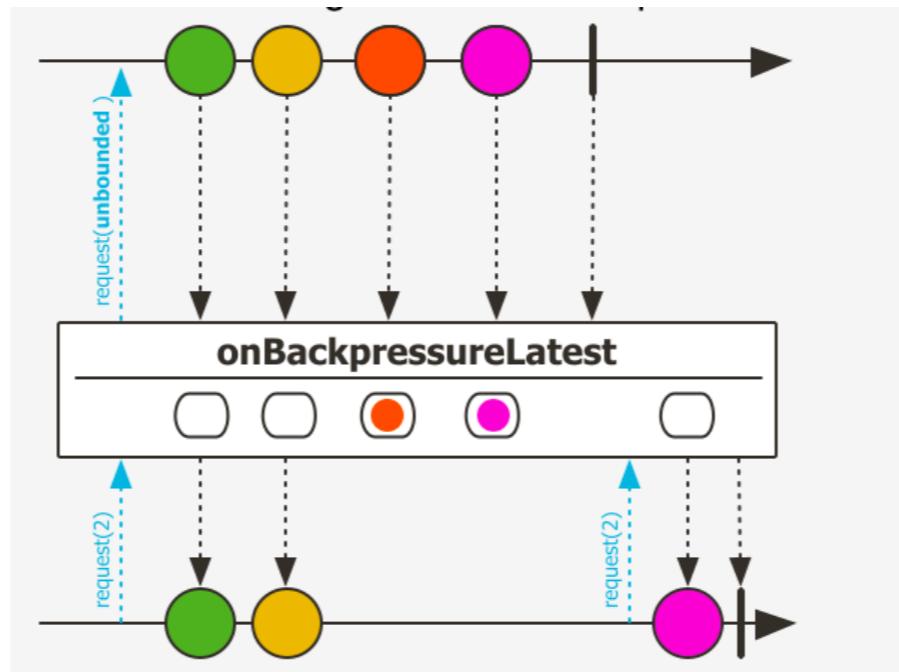
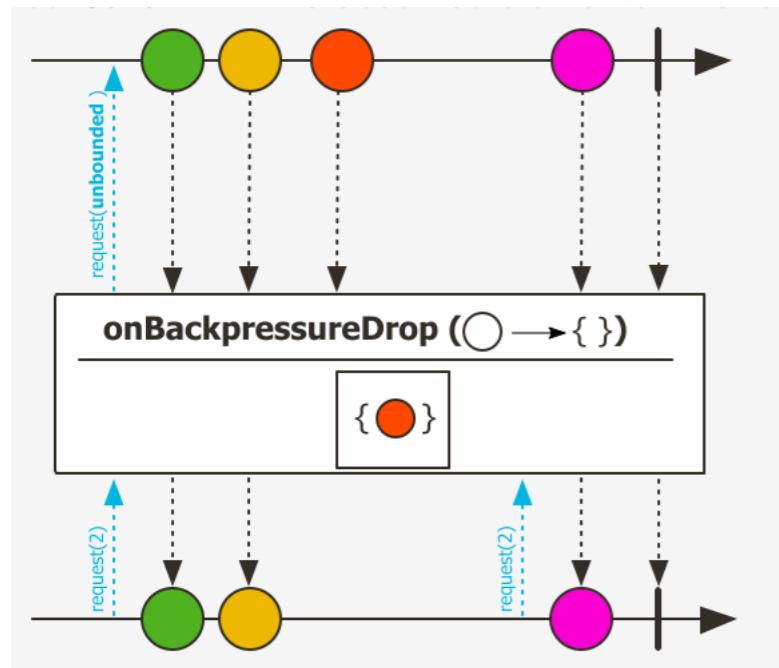


## 10. РЕАКТИВНОЕ ПРОГРАММИРОВАНИЕ

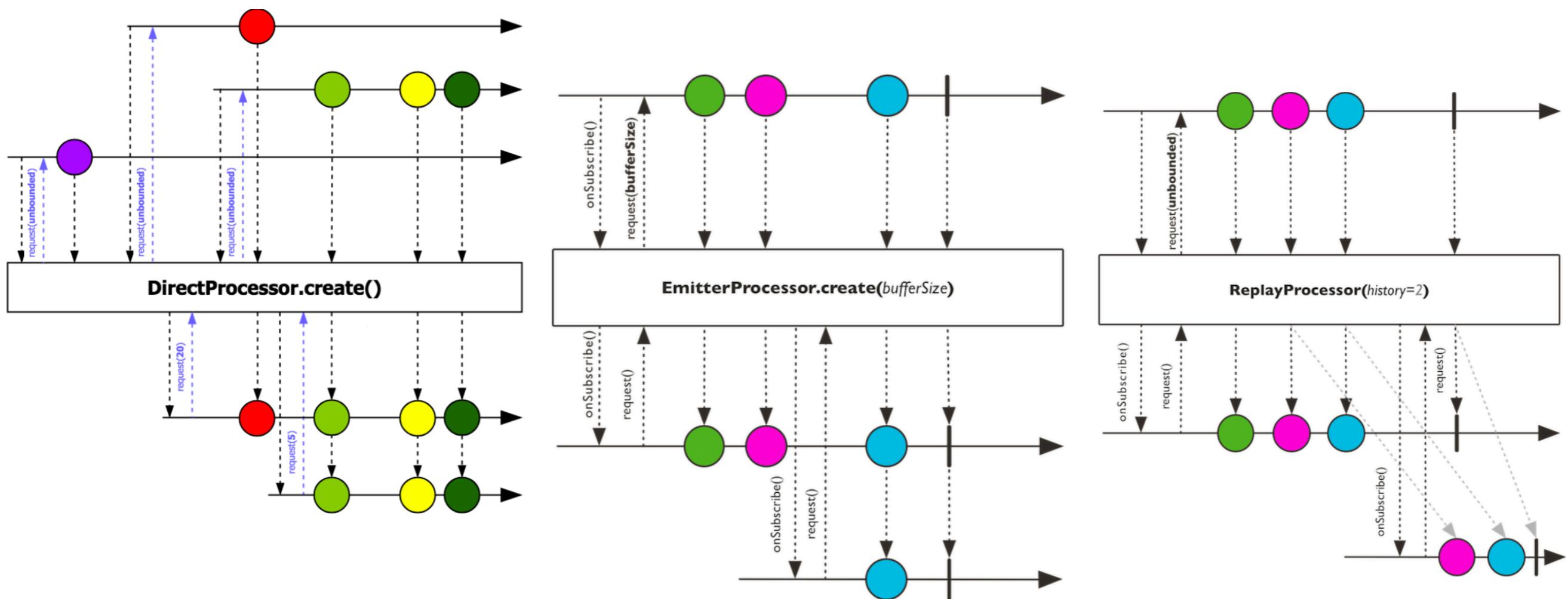
---



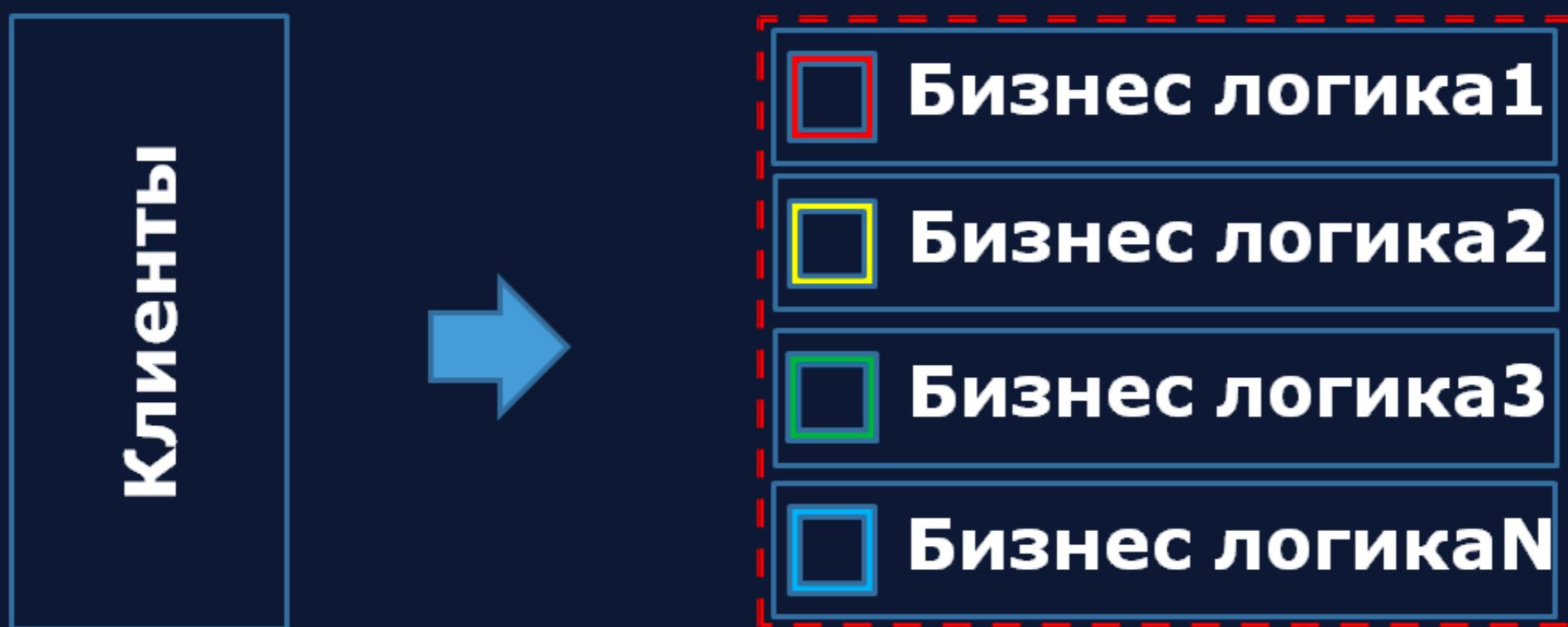
## 10. РЕАКТИВНОЕ ПРОГРАММИРОВАНИЕ



## 10. РЕАКТИВНОЕ ПРОГРАММИРОВАНИЕ



# Монолитная архитектура



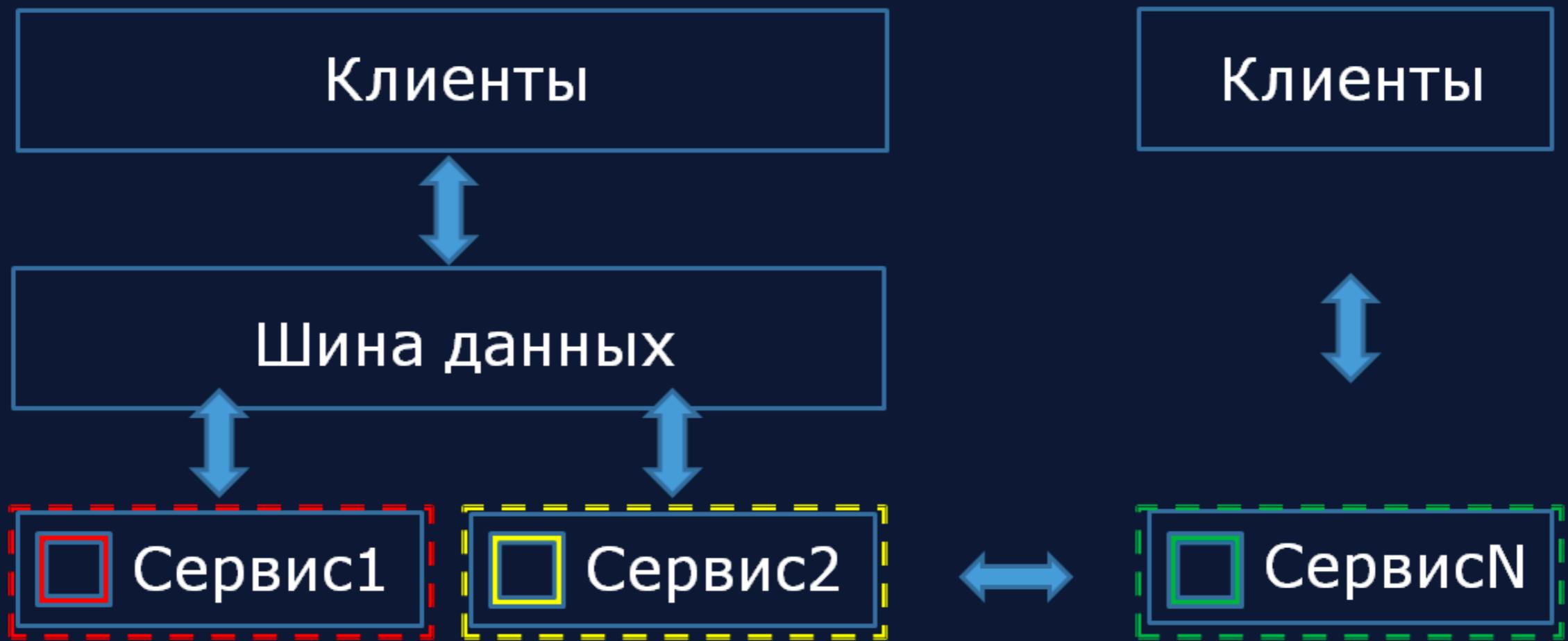
# Проблемы монолитной архитектуры

- » Разобраться в ней трудно
  - » Поддерживать её всё сложнее и сложнее
  - » Дорого вносить изменения
  - » Плохая отказоустойчивость
  - » Нет возможности переиспользовать
  - » Плохая масштабируемость
  - » Застрение на технологии
- 
- Производительность**

# МИКРОСЕРВИСНАЯ АРХИТЕКТУРА

Микросервисная архитектура — вариант сервис-ориентированной архитектуры программного обеспечения, направленный на взаимодействие насколько это возможно небольших, слабо связанных и легко изменяемых модулей — микросервисов, получивший распространение в середине 2010-х годов в связи с развитием практик гибкой разработки и DevOps.

# Микросервисная архитектура



### СВОЙСТВА

модули можно легко заменить в любое время: акцент на простоту, независимость развертывания и обновления каждого из микросервисов;

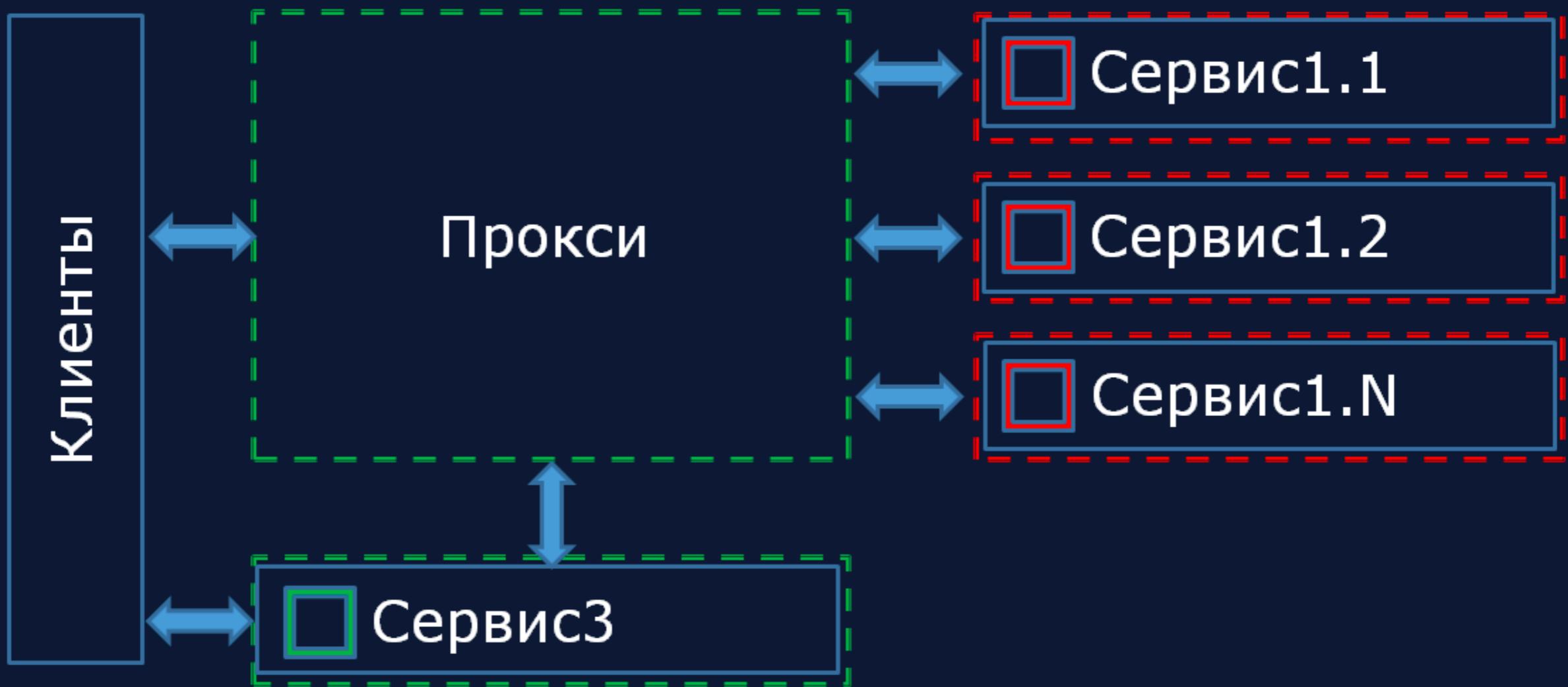
модули организованы вокруг функций: микросервис по возможности выполняет только одну достаточно элементарную функцию;

модули могут быть реализованы с использованием различных языков программирования, фреймворков, связующего программного обеспечения, выполняться в различных средах контейнеризации, виртуализации, под управлением различных операционных систем на различных аппаратных платформах: приоритет отдается в пользу наибольшей эффективности для каждой конкретной функции, нежели стандартизации средств разработки и исполнения;

архитектура симметричная, а не иерархическая: зависимости между микросервисами одноранговые.

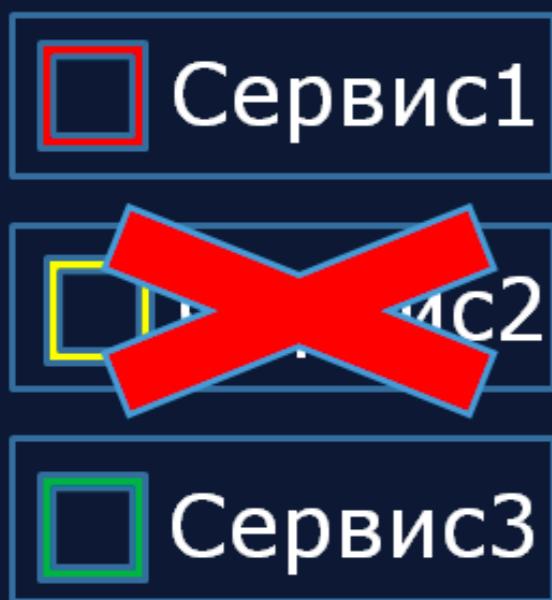
# Плюсы микросервисной архитектуры

## » Масштабирование

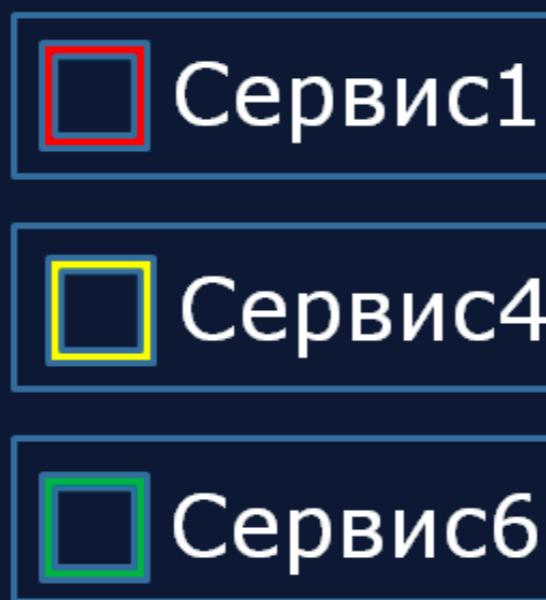


# Плюсы микросервисной архитектуры

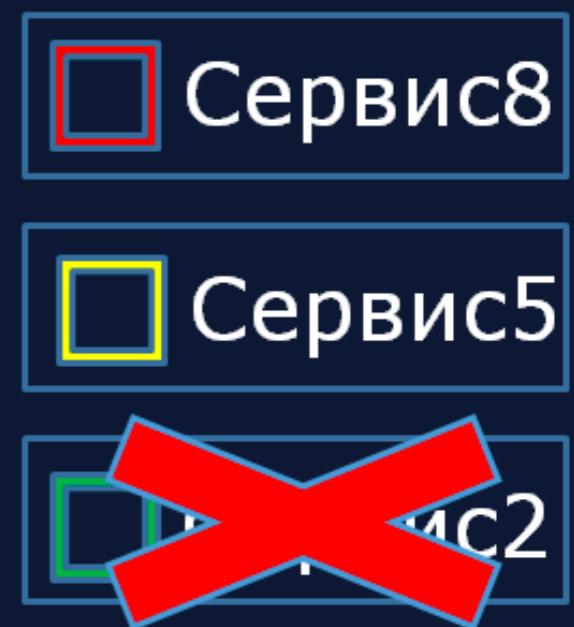
## » Отказоустойчивость



Проект 1



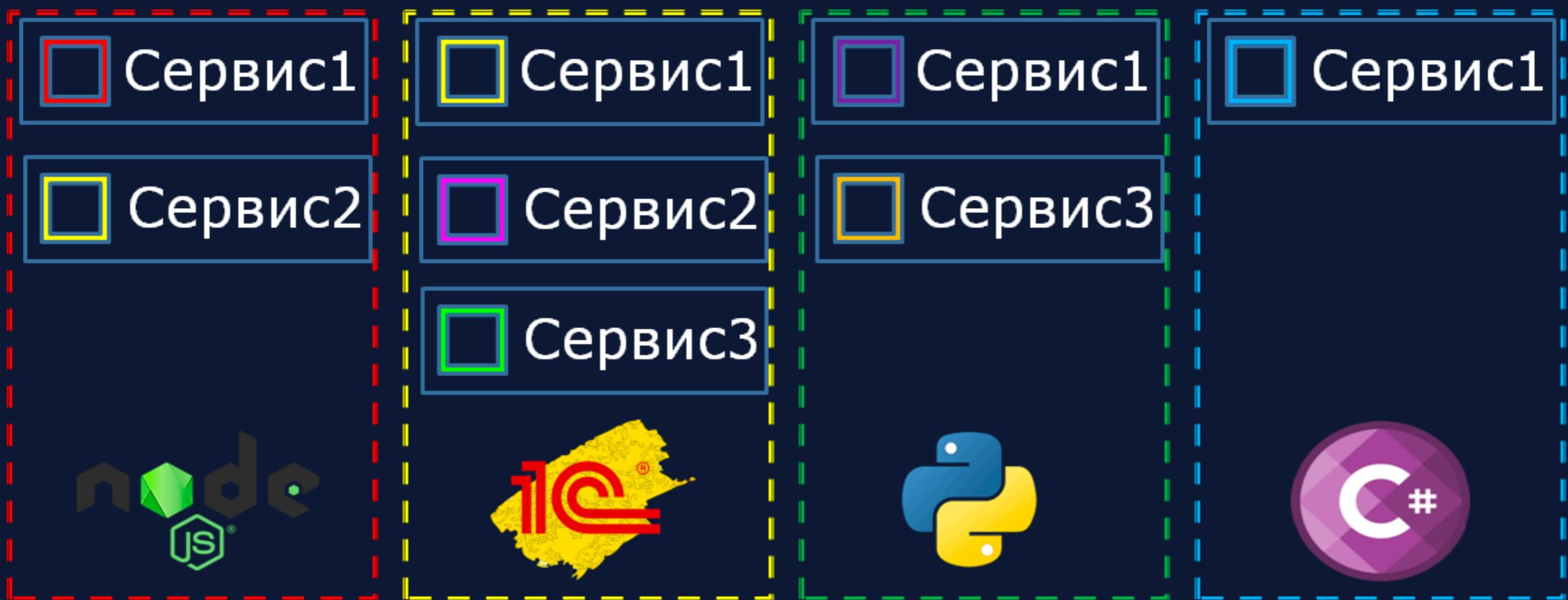
Проект 2



Проект 3

# Плюсы микросервисной архитектуры

- » Без привязки к стеку



# ПРОБЛЕМЫ

**сетевые задержки:** если в модулях, выполняющих несколько функций, взаимодействие локально, то микросервисная архитектура накладывает требование атомизации модулей и взаимодействия их по сети;

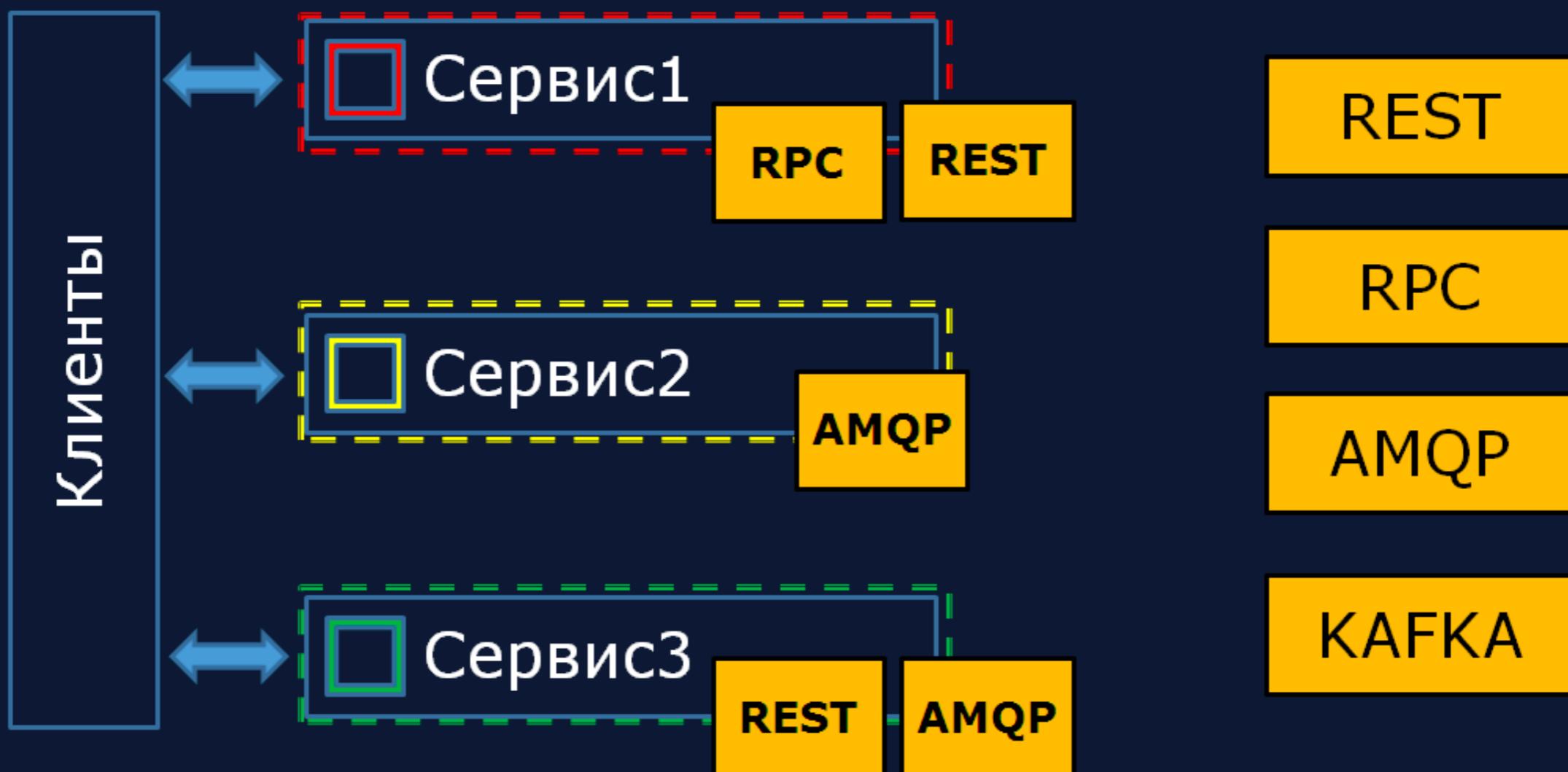
**форматы сообщений:** отсутствие стандартизации и необходимость согласования форматов обмена фактически для каждой пары взаимодействующих микросервисов приводит как к потенциальным ошибкам, так и сложностям отладки.

**Сложность мониторинга**

**Не согласованность данных**

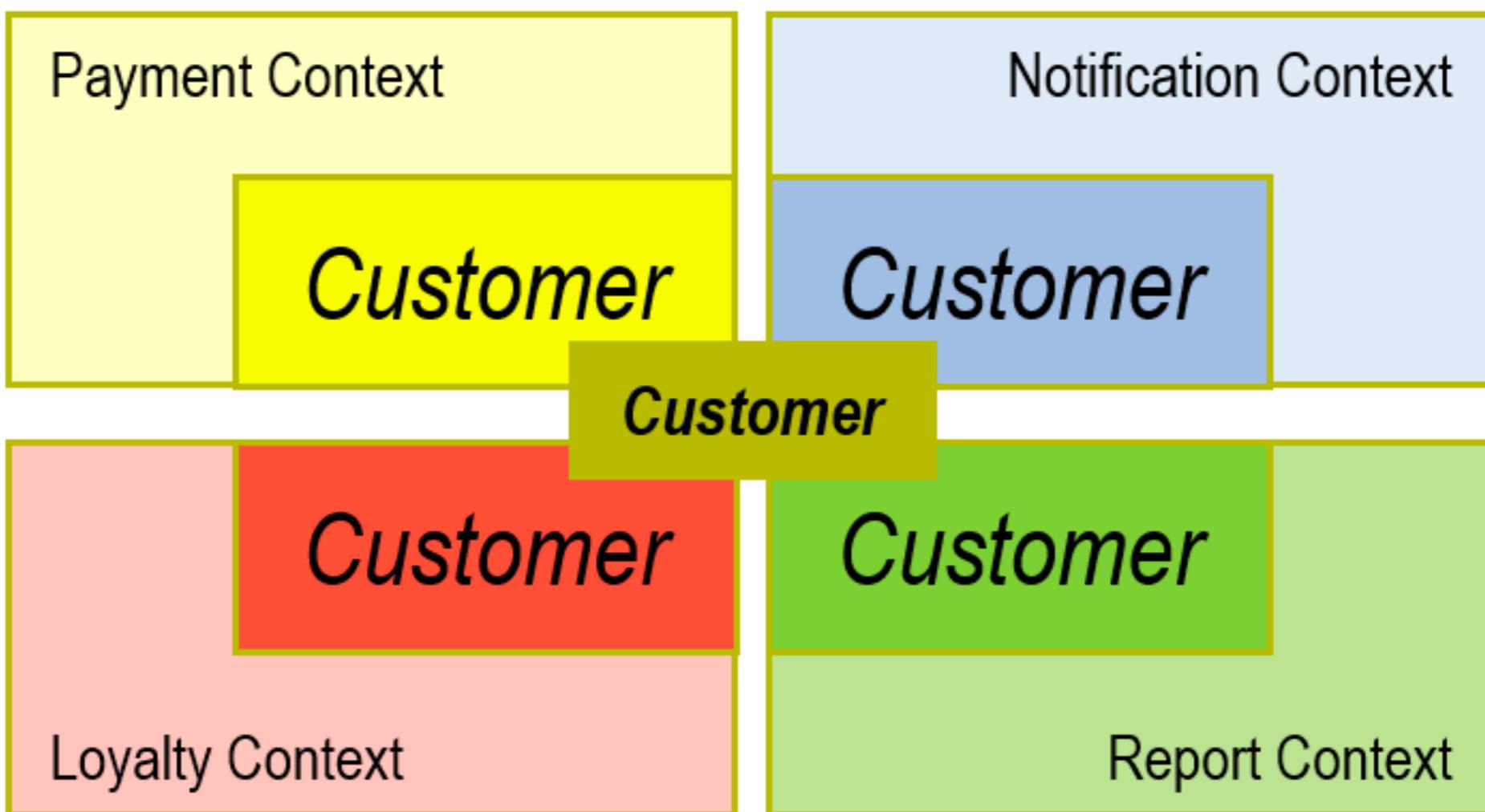
**Усложнение инфраструктуры**

# Коммуникация



## 10. МИКРОСЕРВИСНАЯ АРХИТЕКТУРА

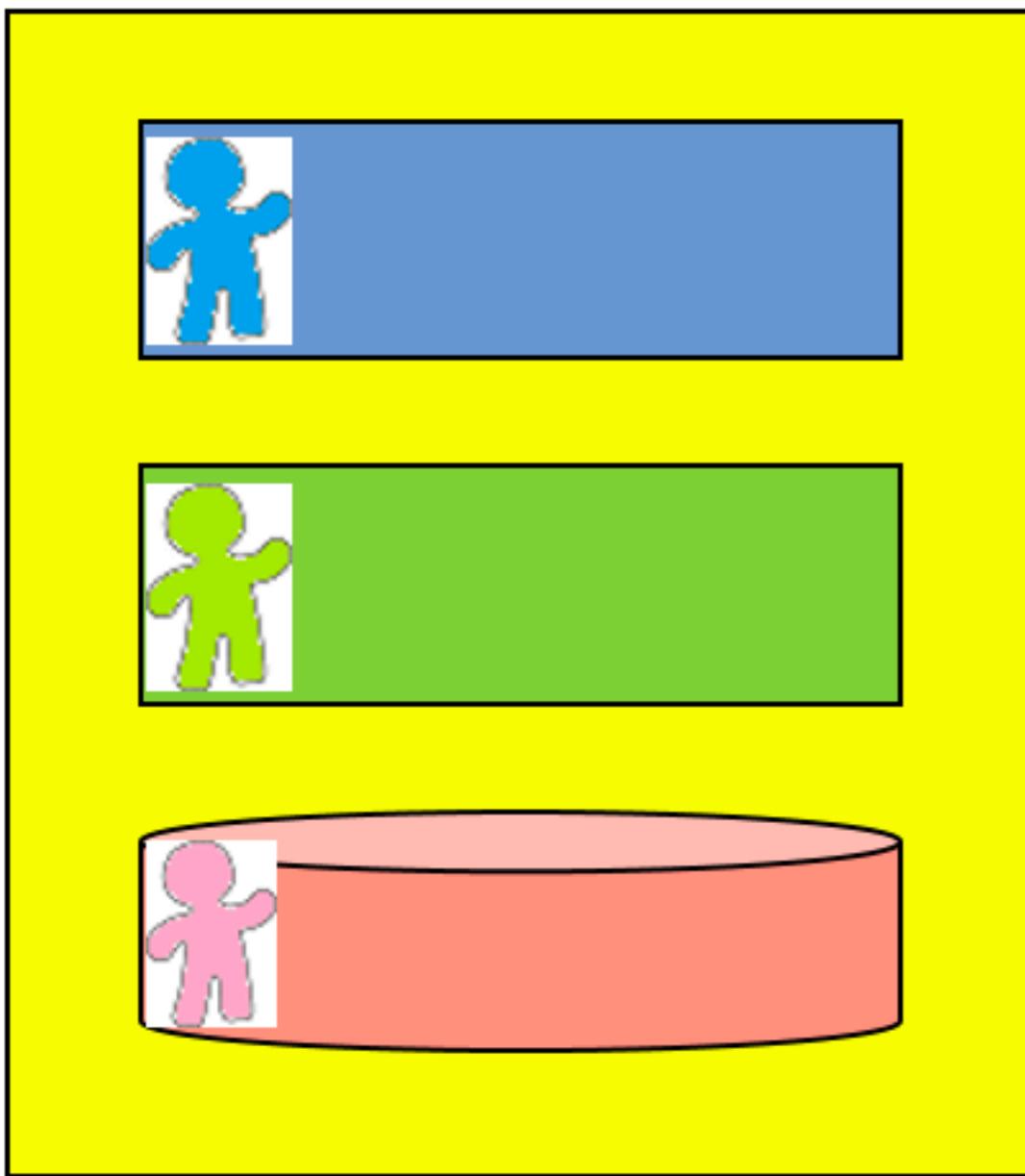
---



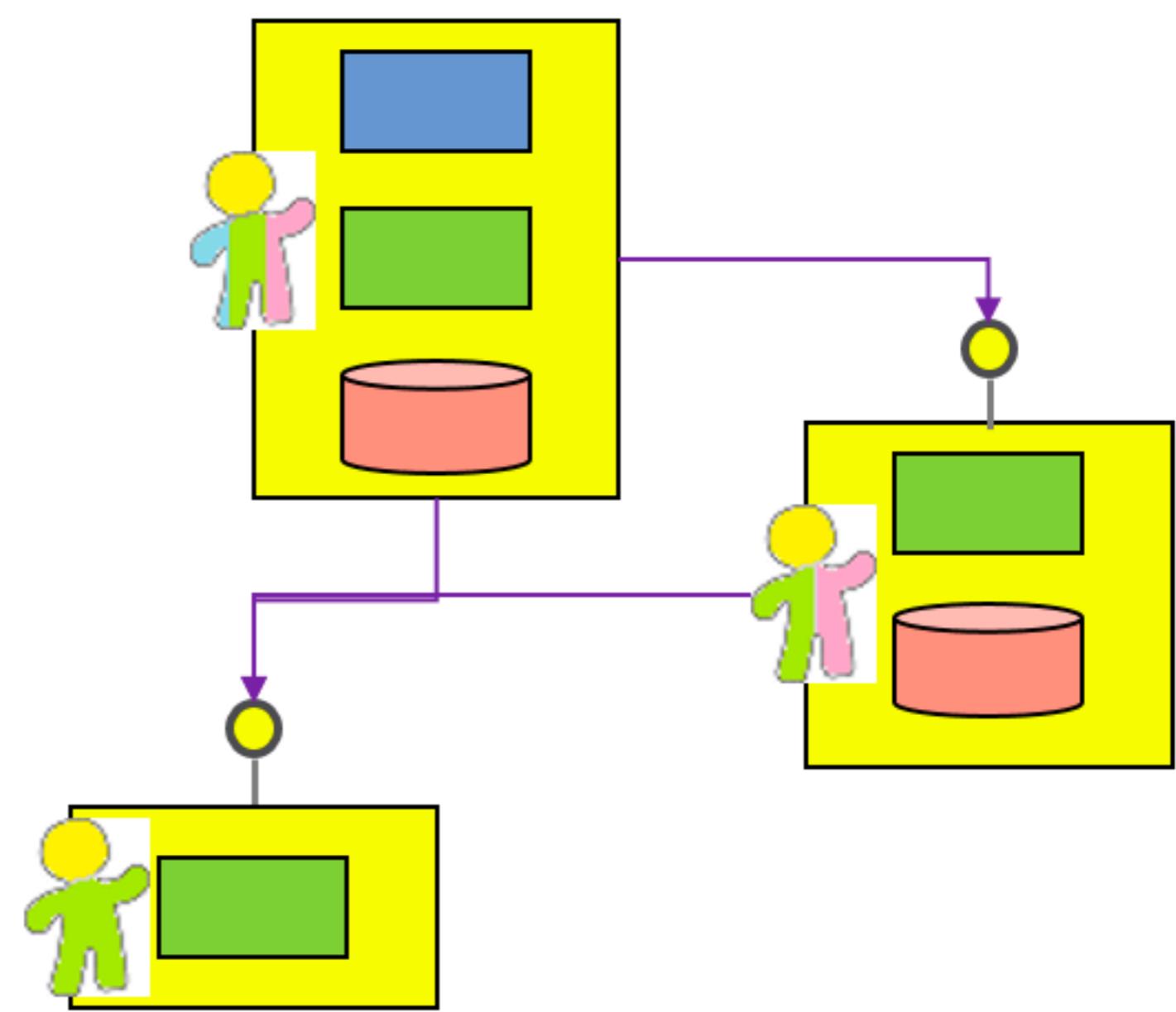
## 10. МИКРОСЕРВИСНАЯ АРХИТЕКТУРА

---

Монолит

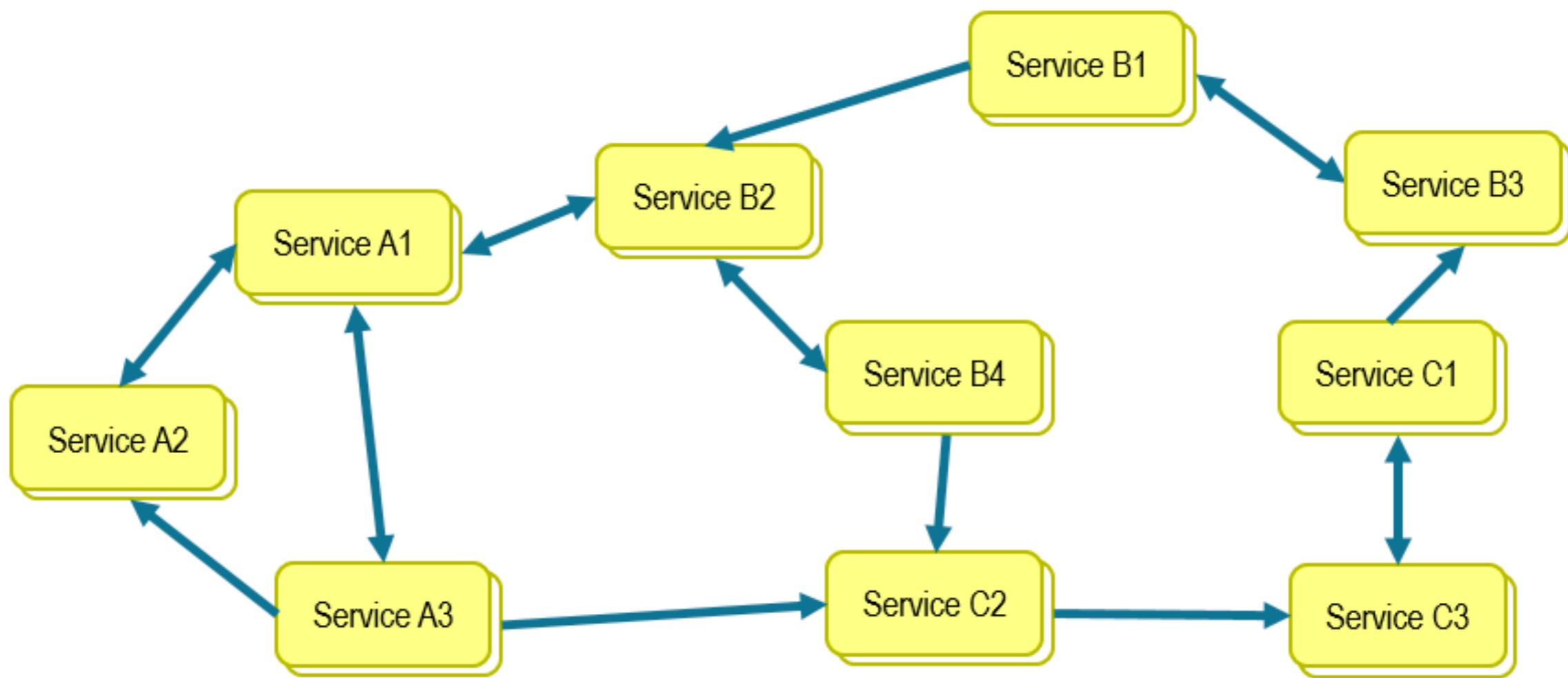


Микросервисы



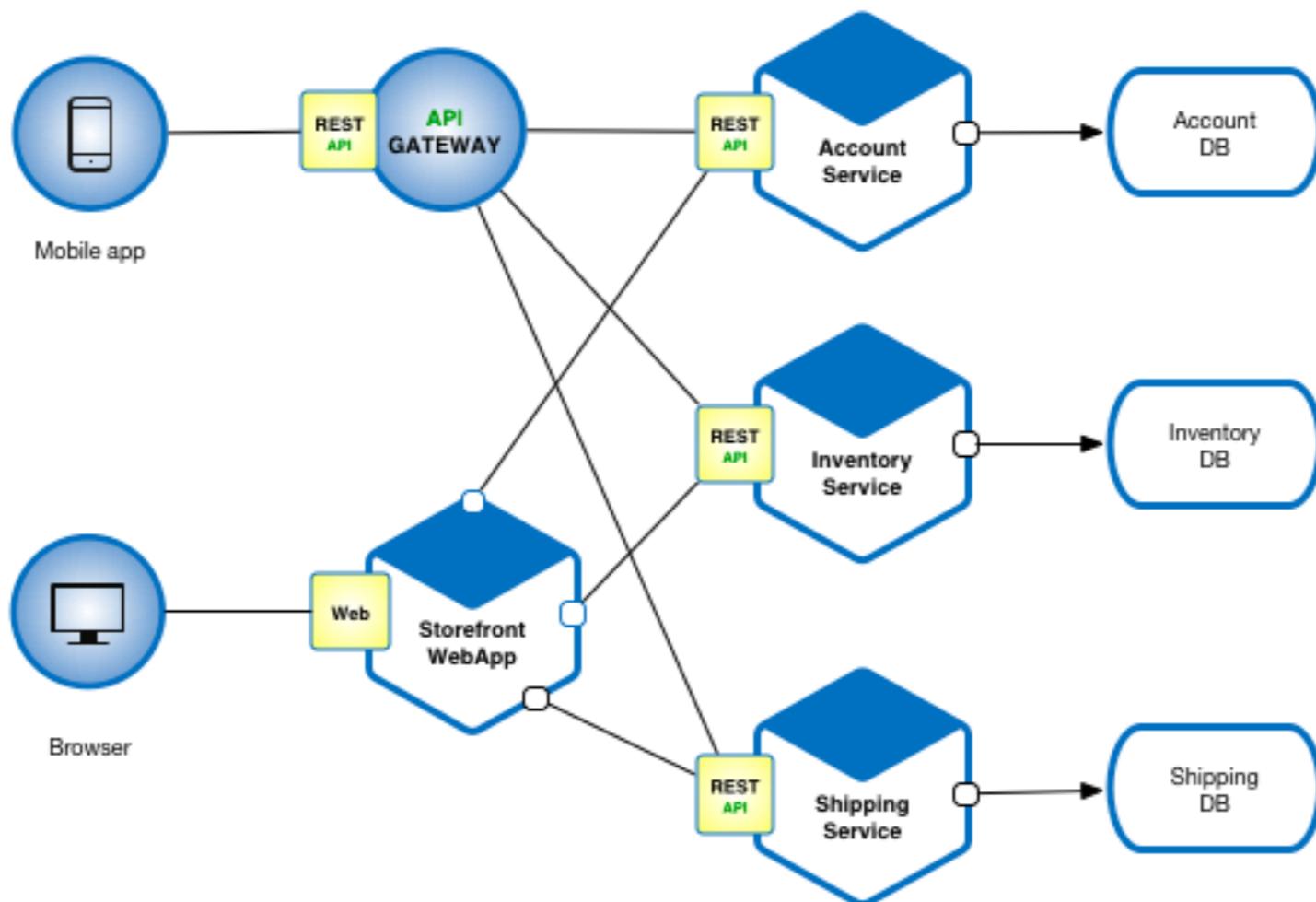
## 10. МИКРОСЕРВИСНАЯ АРХИТЕКТУРА

---



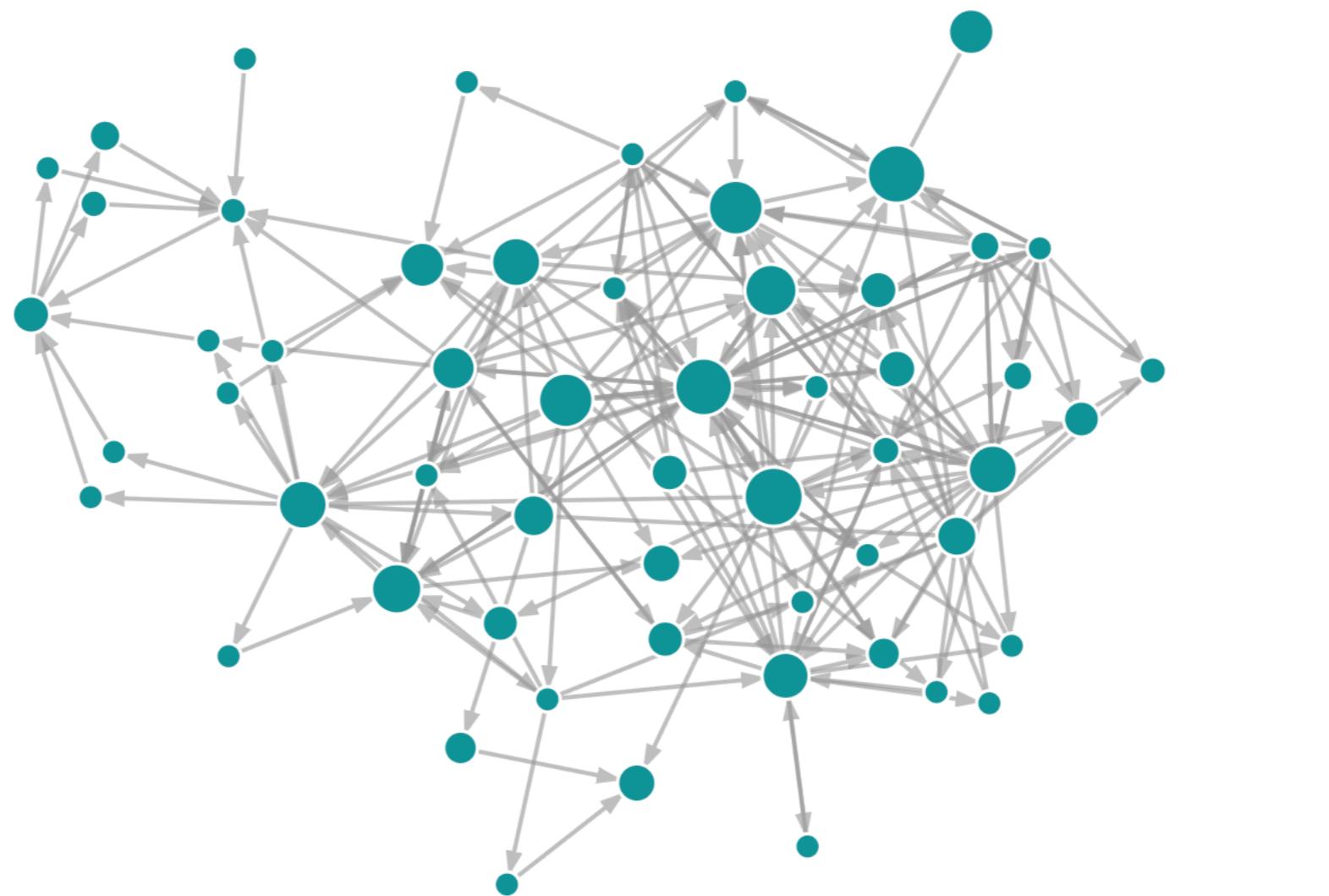
## 10. МИКРОСЕРВИСНАЯ АРХИТЕКТУРА

---

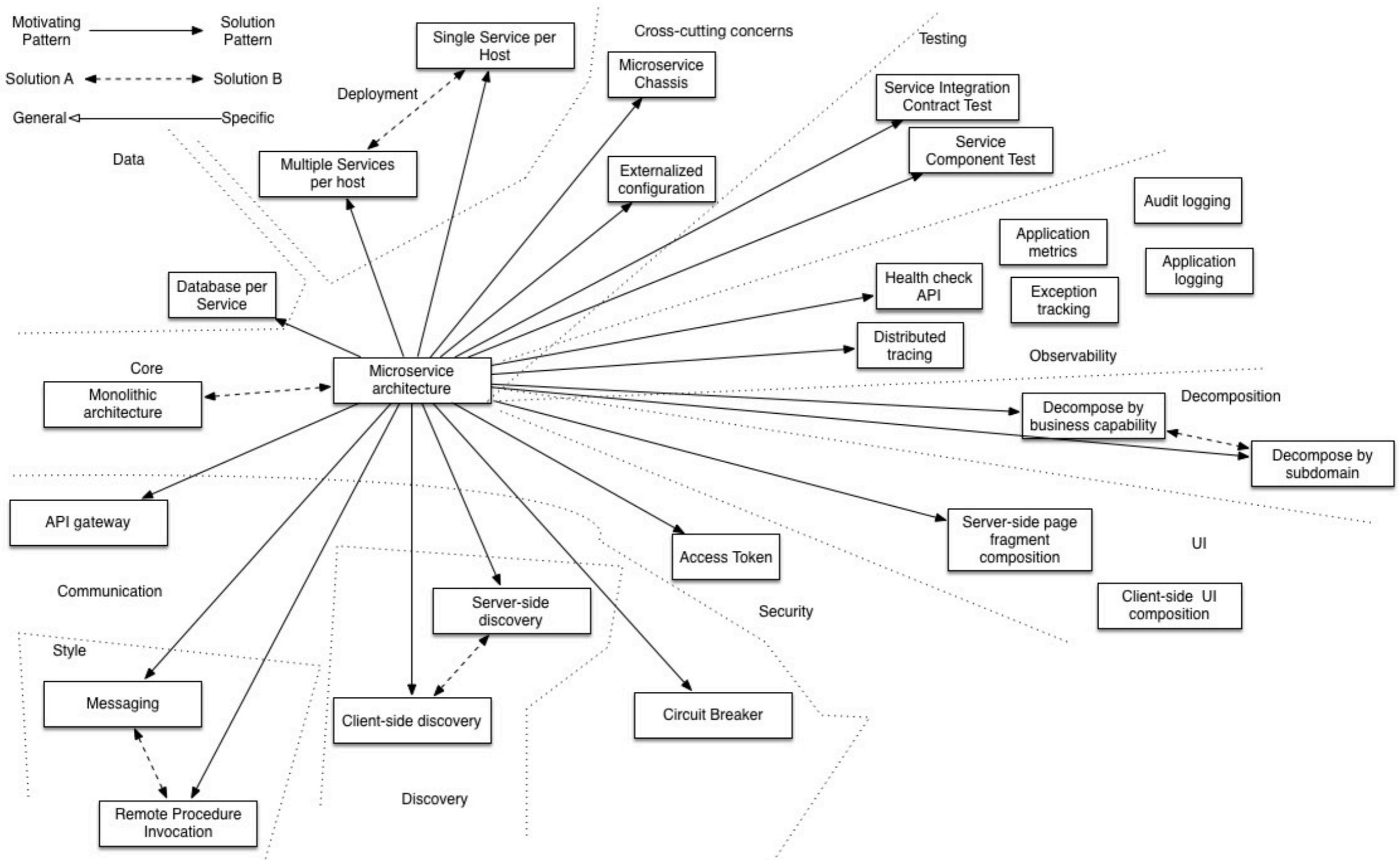


## 10. МИКРОСЕРВИСНАЯ АРХИТЕКТУРА

---



# 10. МИКРОСЕРВИСНАЯ АРХИТЕКТУРА



## 10. МИКРОСЕРВИСНАЯ АРХИТЕКТУРА

---

# ВЫБОР

Сервер (Netty)

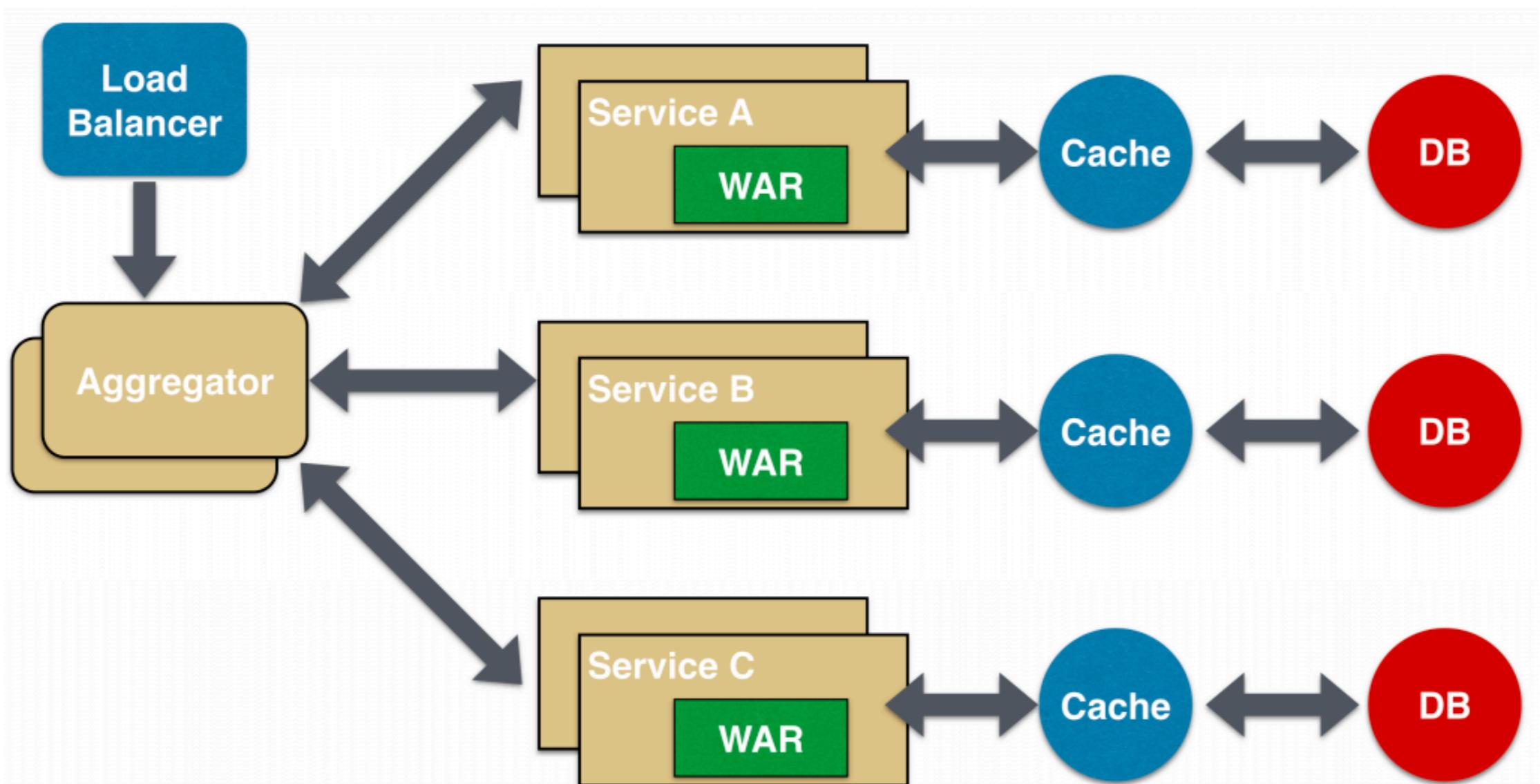
Инфраструктура (Docker <https://docker.com/>, consul, nomad, packer, terraform)

Мониторинг (Grafana <https://grafana.com/grafana/>)

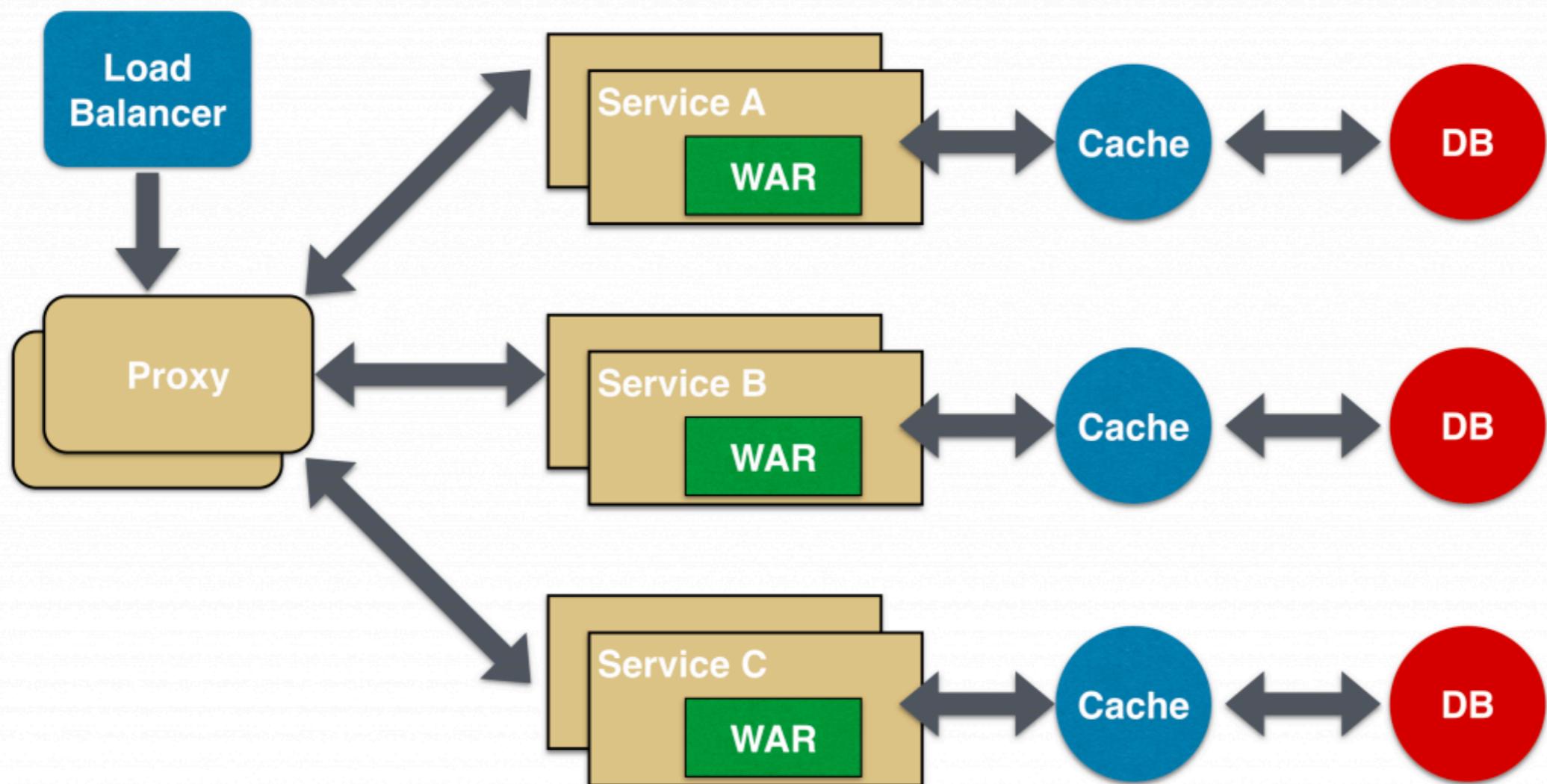
Логи (Loki <https://grafana.com/oss/loki/>)

Трейсинг (Tempo <https://grafana.com/oss/tempo/>)

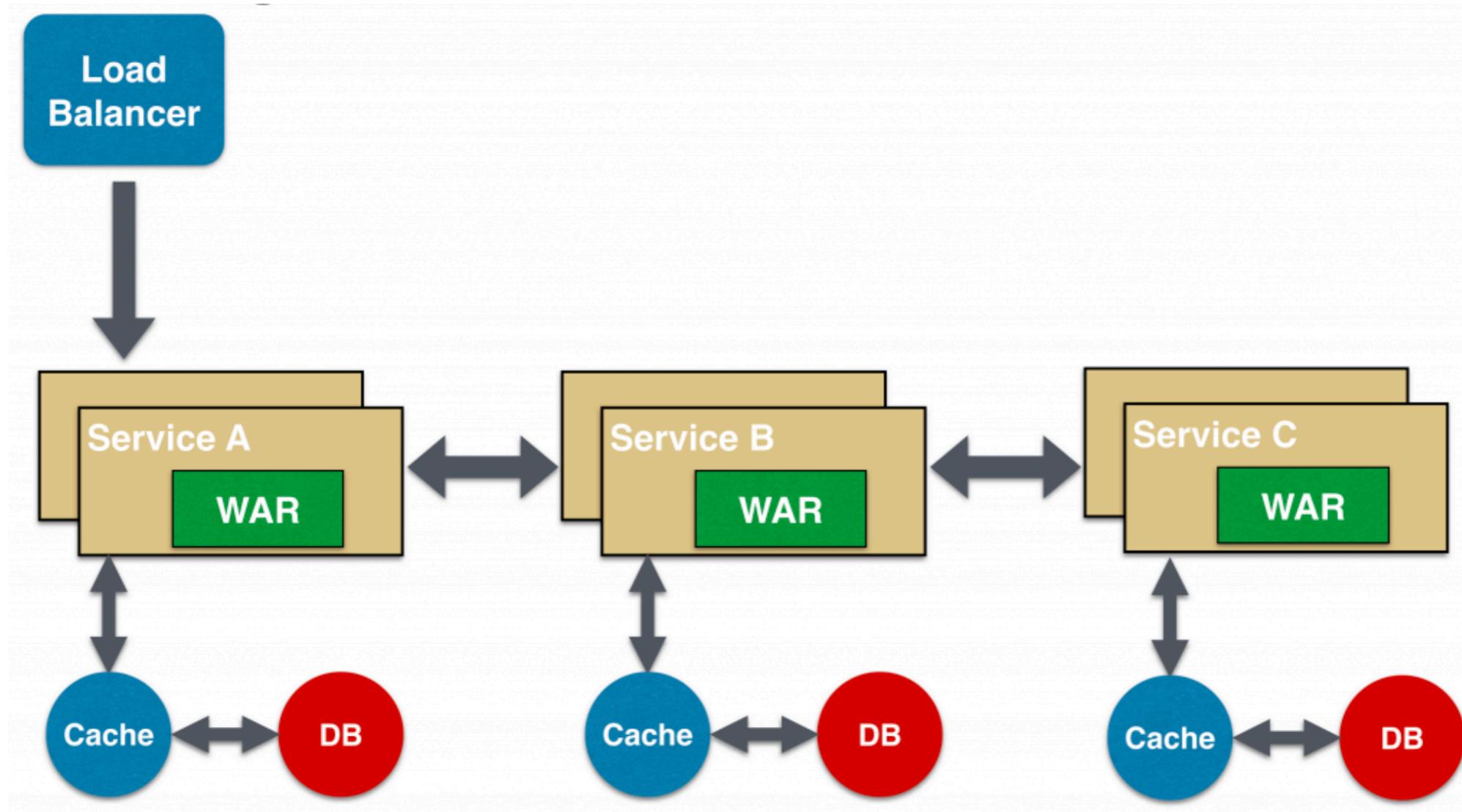
# ПАТТЕРН АГРЕГАТОР (AGGREGATOR)



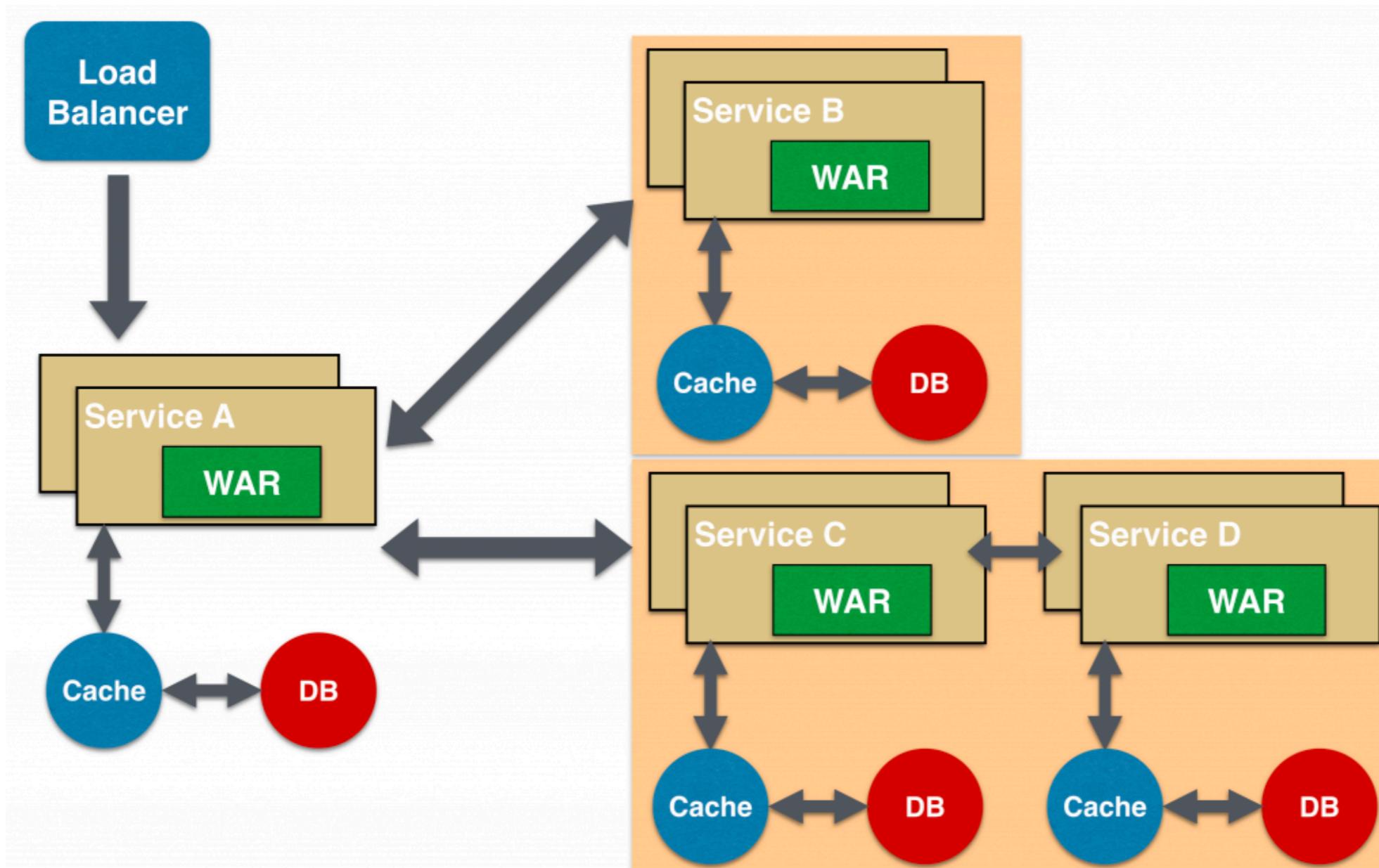
# ПАТТЕРН ПОСРЕДНИК (PROXY)



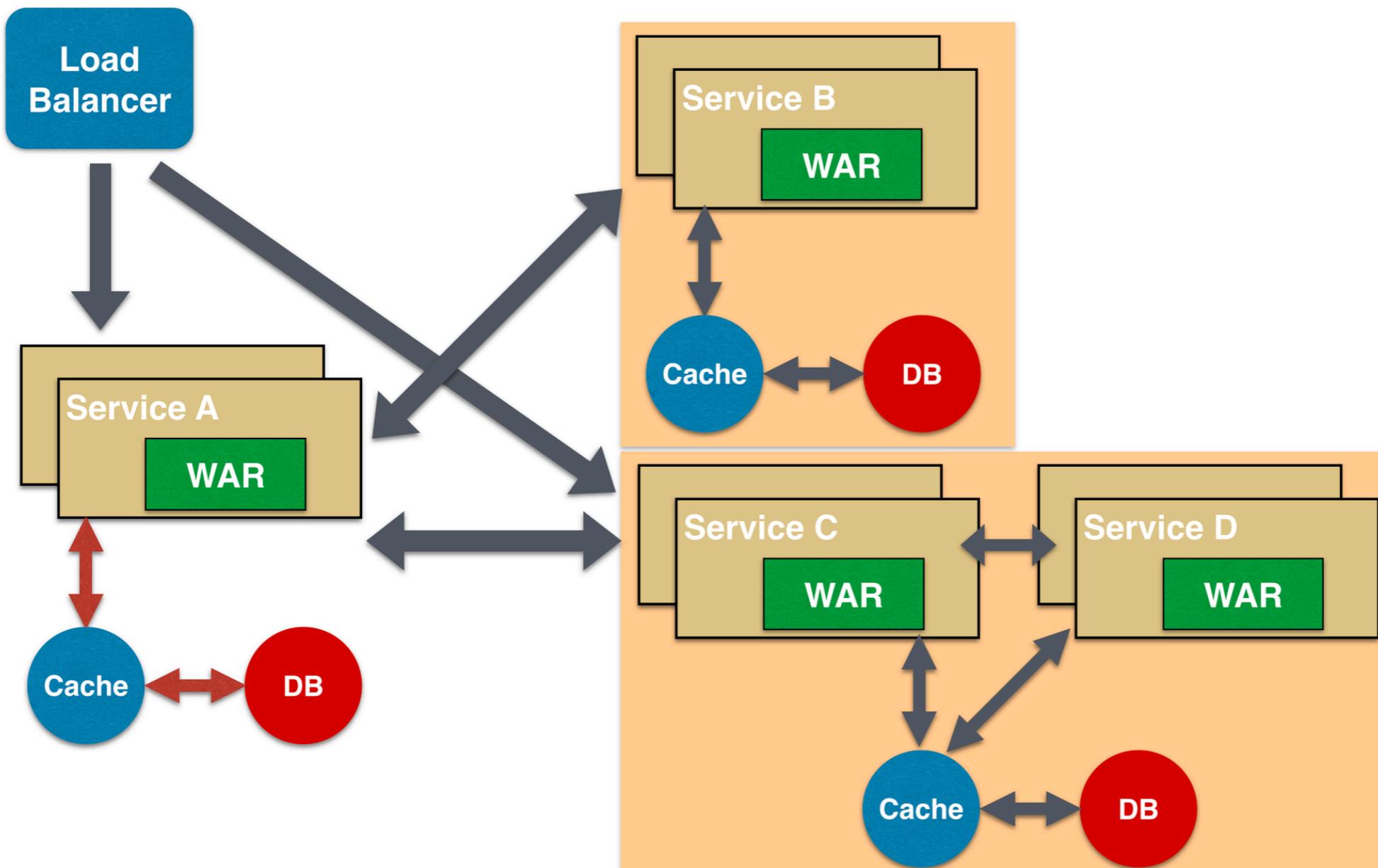
### ПАТТЕРН «ЦЕПОЧКА» (CHAINED)



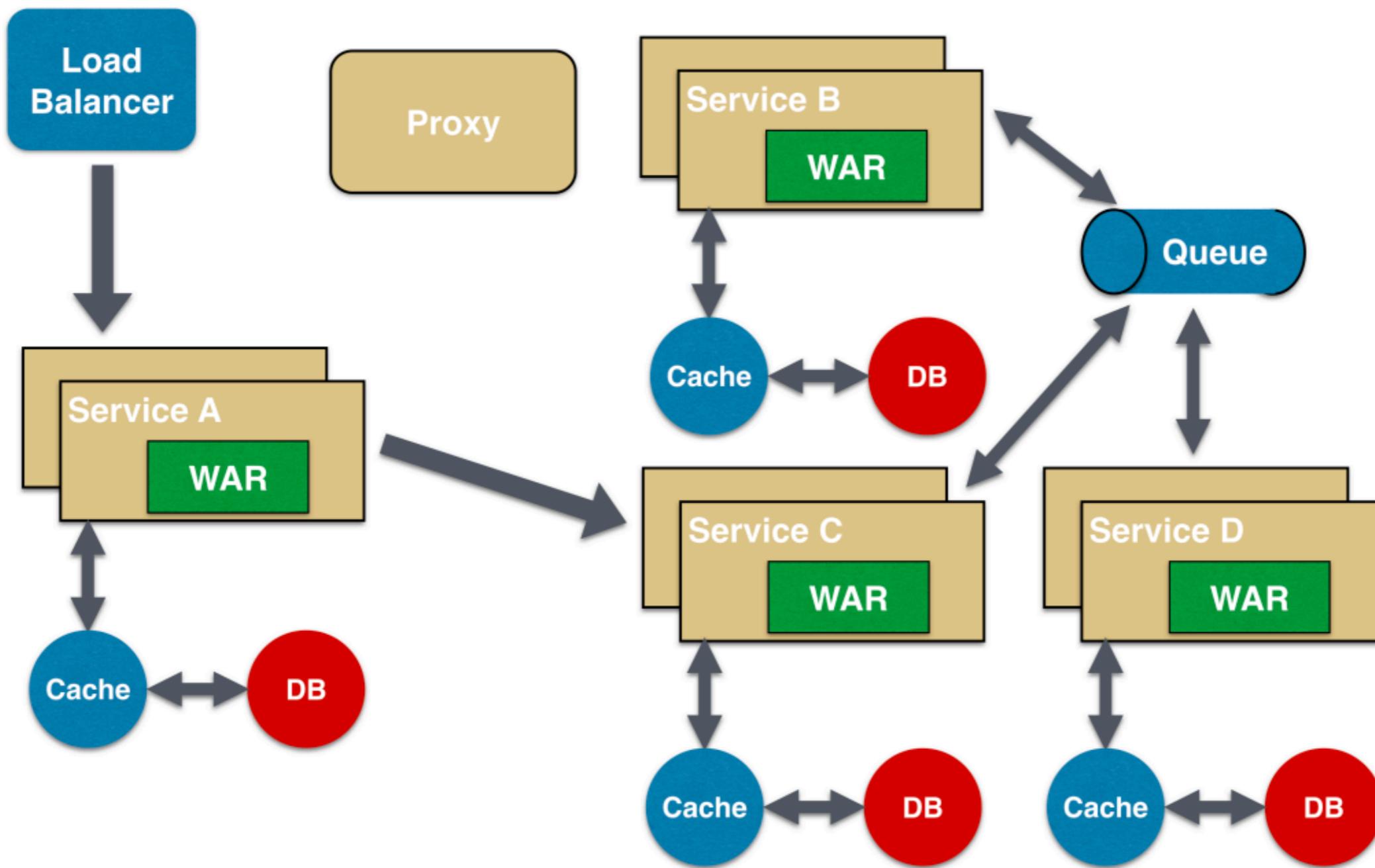
### ПАТТЕРН «ВЕТКА» (BRANCH)

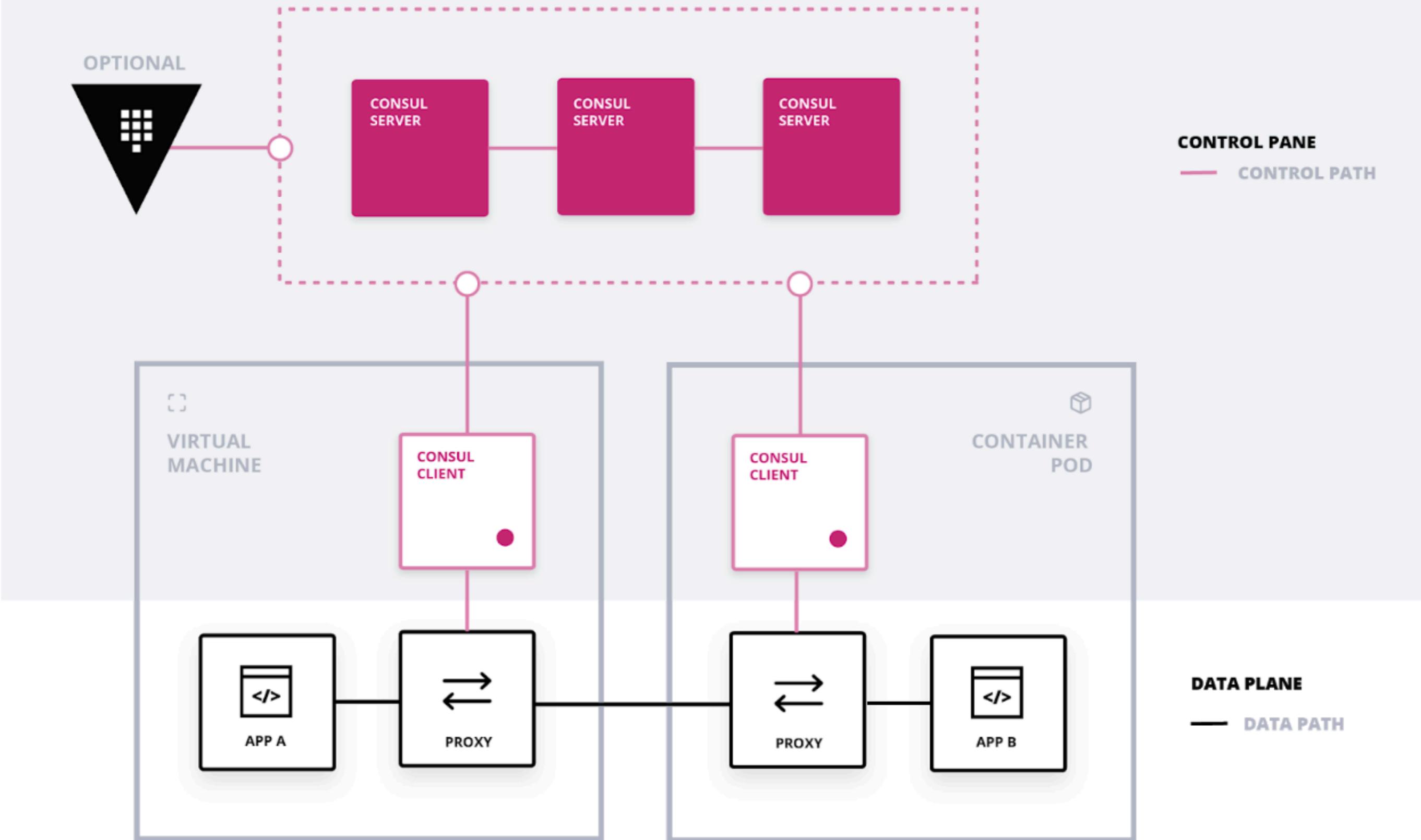


# ПАТТЕРН «РАЗДЕЛЯЕМЫЕ ДАННЫЕ» (SHARED DATA)



# ПАТТЕРН «АСИНХРОННЫЕ СООБЩЕНИЯ» (ASYNCHRONOUS MESSAGING)





# ВЫБОР

```
<dependency>
  <groupId>io.projectreactor.netty</groupId>
  <artifactId>reactor-netty-core</artifactId>
  <version>1.0.7</version>
</dependency>
```

```
<dependency>
  <groupId>io.projectreactor.netty</groupId>
  <artifactId>reactor-netty-http</artifactId>
  <version>1.0.7</version>
</dependency>
```

# SPRING BOOT

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
    <version>2.5.0</version>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webflux</artifactId>
    <version>2.5.0</version>
  </dependency>
</dependencies>
```

# SPRING BOOT

```
# application.yml
server:
  port: 8088
```

# SPRING BOOT ACTUATOR

health	Информация о работоспособности приложения
info	Информация о приложении
env	Свойства из окружающей среды
metrics	Различные метрики о приложении
mappings	@RequestMapping сопоставления контроллера
shutdown	Запускает завершение работы приложения
httptrace	Лог HTTP-запросов / ответов
loggers	Отображение и настройка информации логера
logfile	Содержимое файла логера
threaddump	Выполнить дамп потока
heapdump	Получить дамп кучи JVM
caches	Проверить доступные кеши
integrationgraph	График компонентов интеграции Spring

# SPRING BOOT ACTUATOR

```
# application.yml
management:
  endpoints:
    web.exposure.include: '*'
```

```
http://localhost:8088/actuator
```

# SPRING JPA

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
    <version>2.5.0</version>
</dependency>
```

```
# application.yml
spring:
    jpa.database-platform: org.hibernate.dialect.H2Dialect
    properties.hibernate.enable_lazy_load_no_trans: true
    datasource:
        url: jdbc:h2:mem:testdb
        driverClassName: org.h2.Driver
        username: sa
        password: password
```

## 10. МИКРОСЕРВИСНАЯ АРХИТЕКТУРА

---

### FLYWAY

```
<dependency>
    <groupId>org.flywaydb</groupId>
    <artifactId>flyway-core</artifactId>
    <version>7.9.1</version>
</dependency>
```

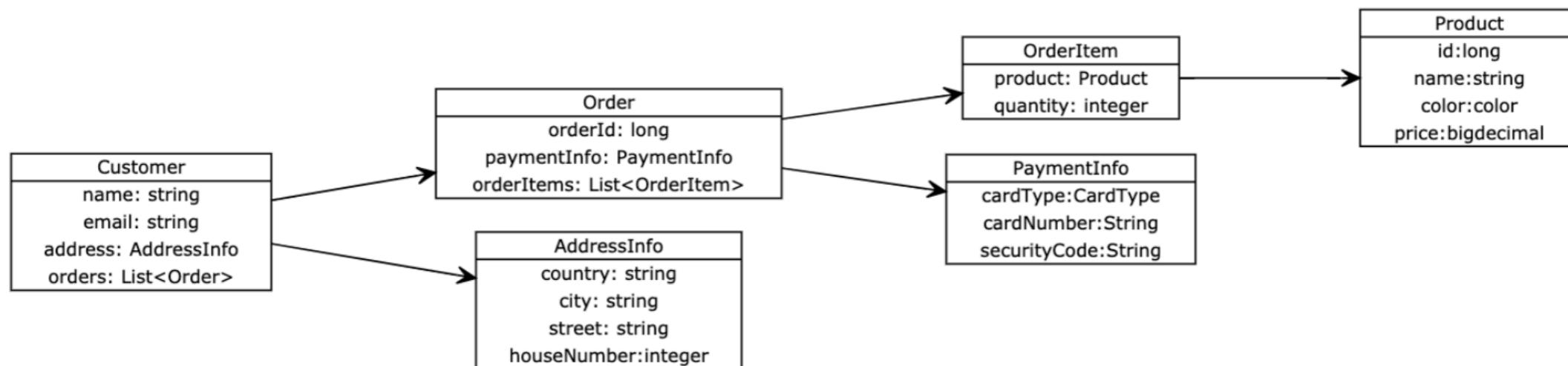
```
# application.yml
spring:
  flyway:
    locations: "classpath:migration"
```

```
V<VERSION>__<NAME>.sql
```

## 10. МИКРОСЕРВИСНАЯ АРХИТЕКТУРА

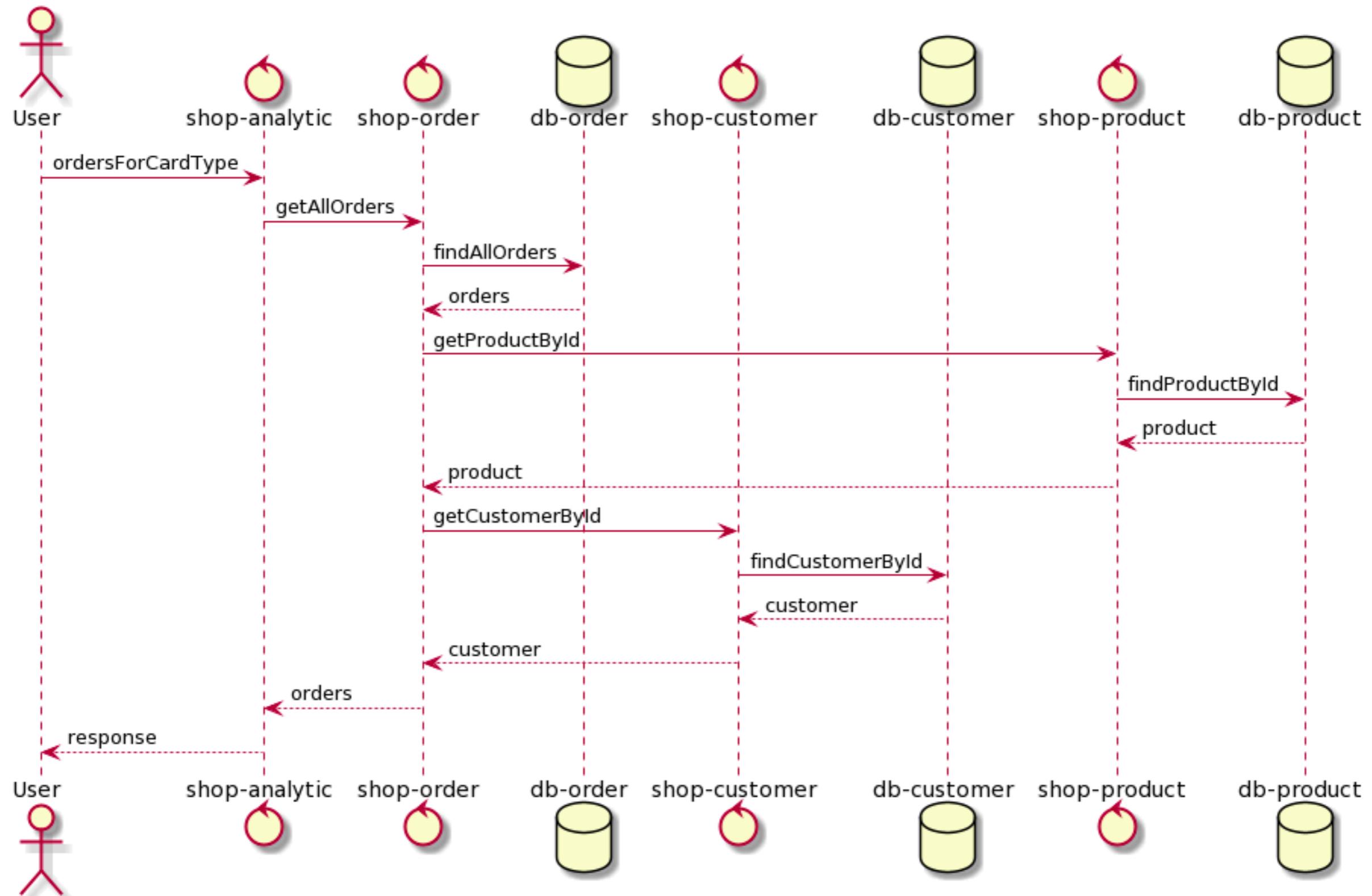
---

### EXAMPLE



## 10. МИКРОСЕРВИСНАЯ АРХИТЕКТУРА

### EXAMPLE



## 10. DOCKER

---

### DOCKER

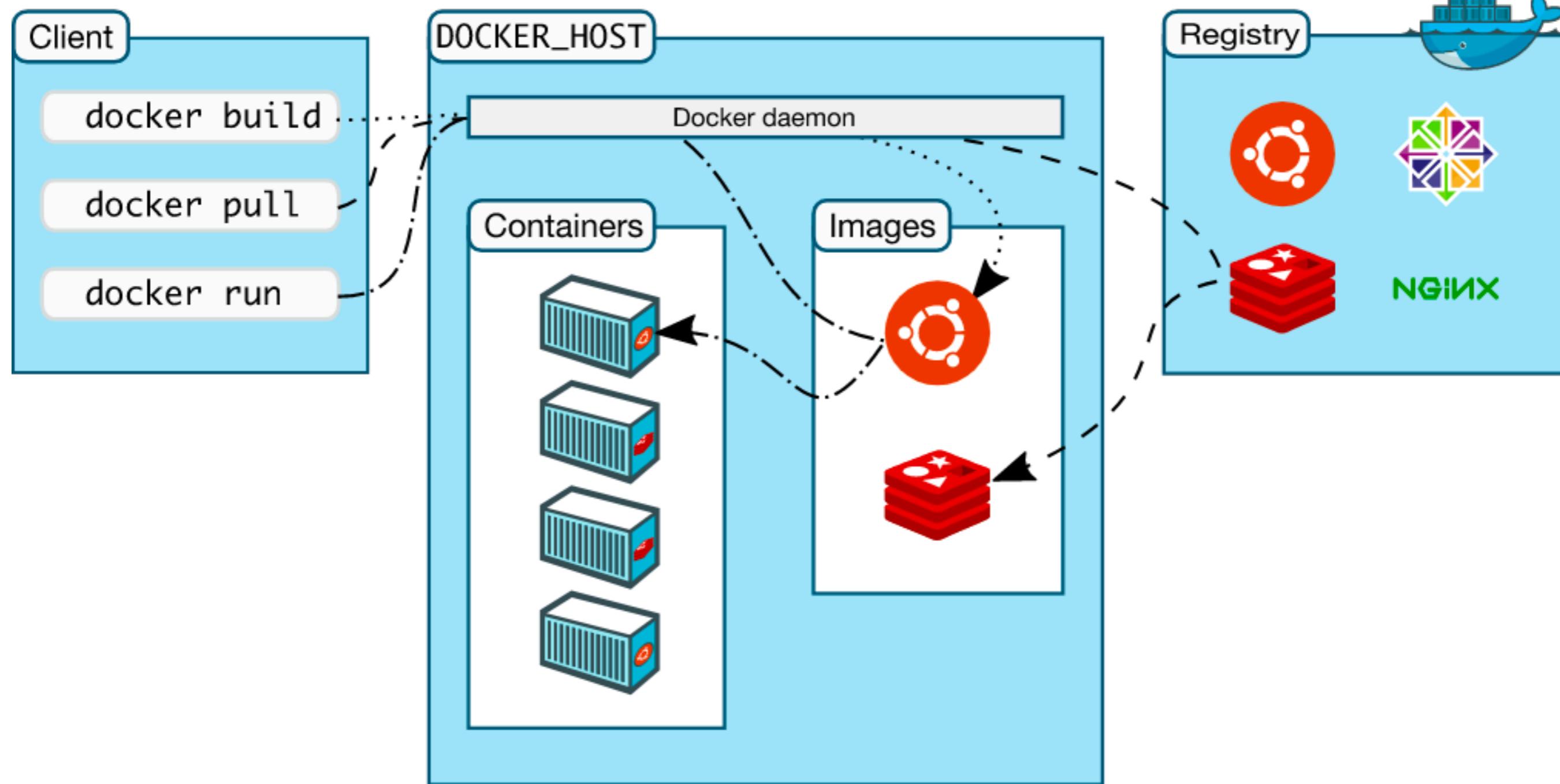
Docker — это программное обеспечение для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации.

Docker — это движок, который запускает виртуальную операционную систему

## 10. DOCKER

---

# DOCKER



# DOCKER ПЛЮСЫ

Абстрагирование приложение от хоста

Масштабирование

Управление версиями и зависимостями

Изолирование среды

Использование слоев

Компоновка

# DOCKER минусы

Тонкая настройка

Обратная совместимость

Производительность

Архитектура

Поддержка

## 10. DOCKER

---

# DOCKER COMPOSE

```
version: 'версия'
networks:
  сети
volumes:
  хранилища
services:
  контейнеры
  имя_контейнера:
    image: образ:версия
    ports:
      - внешний_порт:внутренний порт
    volumes:
      - /путь/к/внешней/папке:/путь/к/внутренней/папке
```

```
docker-compose up
```

```
docker-compose up -d
```

```
docker-compose stop
```

```
docker-compose down
```

## 10. DOCKER

---

# DOCKER НАСТРОЙКА

ssh root@85.159.214.182  
java-cources-2021

```
$ sudo apt-get update

$ sudo apt-get install \
  apt-transport-https \
  ca-certificates \
  curl \
  gnupg \
  lsb-release

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

$ echo \
"deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

## 10. DOCKER

---

### DOCKER COMPOSE

```
version: '3.5'  
services:  
  docker-nginx:  
    image: nginx
```

## 10. DOCKER

---

### DOCKER COMPOSE

```
version: '3.5'  
services:  
  docker-nginx:  
    image: nginx  
    ports:  
      - '8080:80'
```

# DOCKER COMPOSE

```
version: '3.5'
services:
  docker-nginx:
    image: nginx
    ports:
      - '8094:80'
  volumes:
    - /opt/data:/usr/share/nginx/html/
```

## 10. DOCKER

---

# DOCKER COMPOSE

```
version: '3.5'
volumes:
  mysql-data:

services:
  docker-mysql:
    image: mysql
    volumes:
      - mysql-data:/var/lib/mysql
    environment:
      - MYSQL_ROOT_PASSWORD=password
      - MYSQL_DATABASE=database
      - MYSQL_USER=user
      - MYSQL_PASSWORD=password
```

# DOCKER COMPOSE

```
version: '3.5'
volumes:
  mysql-data:
networks:
  courses-network:

services:

  docker-nginx:
    image: nginx
    ports:
      - '8094:80'
    volumes:
      - ./:/usr/share/nginx/html/
  networks:
    - courses-network

  docker-mysql:
    image: mysql
    volumes:
      - mysql-data:/var/lib/mysql
    environment:
      - MYSQL_ROOT_PASSWORD=password
      - MYSQL_DATABASE=database
      - MYSQL_USER=user
      - MYSQL_PASSWORD=password
    networks:
      - courses-network
```

# DOCKER МИНУСЫ

```
version: '3.5'  
volumes:  
  mysql-data:  
networks:  
  courses-network:  
services:  
  docker-nginx:  
    image: nginx  
    ports:  
      - '8094:80'  
    volumes:  
      - ./:/usr/share/nginx/html/  
networks:  
  - courses-network
```

```
docker-mysql:  
  image: mysql  
  volumes:  
    - mysql-data:/var/lib/mysql  
  environment:  
    - MYSQL_ROOT_PASSWORD=password  
    - MYSQL_DATABASE=database  
    - MYSQL_USER=user  
    - MYSQL_PASSWORD=password  
  networks:  
    - courses-network  
  
docker-phpmyadmin:  
  image: phpmyadmin/phpmyadmin:latest  
  ports:  
    - "8095:80"  
  environment:  
    - PMA_HOST=docker-mysql  
  networks:  
    - courses-network
```

# DOCKER ОБРАЗЫ

- **FROM** - образ, на основе которого будет создаваться наш образ;
- **RUN** - выполнить команду в окружении образа;
- **COPY** - скопировать файл в образ;
- **WORKDIR** - задать рабочую папку для образа;
- **ENV** - задать переменную окружения образа;
- **CMD** - задать основной процесс образа;

# DOCKER COMPOSE

```
version: '3.5'
volumes:
  mysql-data:
networks:
  courses-network:
services:
  docker-nginx:
    build:
      dockerfile: Dockerfile.nginx
      context: ../
    ports:
      - '8094:80'
    networks:
      - courses-network
  docker-mysql:
    image: mysql
    volumes:
      - mysql-data:/var/lib/mysql
    environment:
      - MYSQL_ROOT_PASSWORD=password
      - MYSQL_DATABASE=database
      - MYSQL_USER=user
      - MYSQL_PASSWORD=password
    networks:
      - courses-network
  docker-phpmyadmin:
    image: phpmyadmin/phpmyadmin:latest
    ports:
      - "8095:80"
    environment:
      - PMA_HOST=docker-mysql
    networks:
      - courses-network
```

# DOCKER ПОЛЕЗНЫЕ КОМАНДЫ

docker ps	список контейнеров
docker volume ls	список хранилищ
docker network ls	список сетей
docker image ls	список образов
docker logs -f <container>	логи контейнера
docker rm -f <container>	удаление контейнера
docker stop <container>	остановка контейнера
docker start <container>	старт контейнера

# DOCKER ОБРАЗЫ

```
FROM nginx
COPY ./index.html /usr/share/nginx/html/
```

## 10. DOCKER

---

# DOCKER REGISTRY

```
docker login docker.pkg.github.com/nestor-by/java-courses-2021
```

```
docker image build \  
-t http://docker.pkg.github.com/nestor-by/java-courses-2021/shop-order:0.0.1-SNAPSHOT .
```

```
docker image push \  
http://docker.pkg.github.com/nestor-by/java-courses-2021/shop-order:0.0.1-SNAPSHOT
```

# MAVEN DOCKER PLUGIN

```
<plugin>
  <groupId>com.spotify</groupId>
  <artifactId>dockerfile-maven-plugin</artifactId>
  <version>1.4.13</version>
  <executions>
    <execution>
      <id>default</id>
      <goals>
        <goal>build</goal>
      </goals>
    </execution>
  </executions>
  <configuration>
    <repository>docker.pkg.github.com/nestor-by/java-courses-2021/${project.name}</repository>
    <tag>${project.version}</tag>
    <buildArgs>
      <JAR_FILE>${project.build.finalName}.jar</JAR_FILE>
    </buildArgs>
  </configuration>
</plugin>
```

# MAVEN DEPENDENCY PLUGIN

```
<plugin>
  <artifactId>maven-dependency-plugin</artifactId>
  <executions>
    <execution>
      <id>docker</id>
      <goals>
        <goal>copy-dependencies</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

## 10. DOCKER

---

### DOCKER IMAGE

```
FROM openjdk:8-jre-alpine

ARG JAR_FILE

ADD target/dependency           /usr/share/service/lib
ADD target/${JAR_FILE}          /usr/share/service/service.jar

ENTRYPOINT [
    "/usr/bin/java",
    "-cp", "/usr/share/service/service.jar:/usr/share/service/lib/*",
    "by.part7.product.ProductApp"
]
```

## 10. DOCKER

---

### DOCKER IMAGE

```
FROM openjdk:8-jre-alpine

ARG JAR_FILE

ADD target/dependency           /usr/share/service/lib
ADD target/${JAR_FILE}          /usr/share/service/service.jar

ENTRYPOINT [
    "/usr/bin/java",
    "-cp", "/usr/share/service/service.jar:/usr/share/service/lib/*",
    "by.part7.product.ProductApp"
]
```

# PROJECT STRUCTURE

```
> shop-analytic
  <shop-customer>
    > src
    > target
      Dockerfile 3.06.21, 01:31, 345 B 9 minutes ago
      pom.xml 3.06.21, 01:35, 2.58 kB 2 minutes ago
  <shop-order>
    > src
    > target
      Dockerfile 3.06.21, 01:31, 339 B 11 minutes ago
      pom.xml 3.06.21, 01:35, 2.75 kB 7 minutes ago
```

# MAVEN REPOSITORY

```
mvn package  
mvn dockerfile:build  
mvn verify  
mvn dockerfile:push  
mvn deploy
```

# MAVEN REPOSITORY

#pom.xml

```
<distributionManagement>
  <repository>
    <id>java-courses-2021</id>
    <name>GitHub Packages</name>
    <url>https://maven.pkg.github.com/nestor-by/java-courses-2021</url>
  </repository>
</distributionManagement>
```

# MAVEN REPOSITORY

```
#~/.m2/settings.xml

<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 https://maven.apache.org/xsd/settings-1.0.0.xsd">
  <profiles>
    <profile>
      <id>nexus</id>
      <activation>
        <activeByDefault>true</activeByDefault>
      </activation>
      <repositories>
        <repository>
          <id>java-courses-2021</id>
          <url>https://maven.pkg.github.com/nestor-by/java-courses-2021</url>
        </repository>
        <repository>
          <id>central</id>
          <url>https://repo1.maven.org/maven2/</url>
        </repository>
      </repositories>
    </profile>
  </profiles>
  <servers>
    <server>
      <id>java-courses-2021</id>
      <username>nestor-by</username>
      <password>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</password>
    </server>
  </servers>
</settings>
```