

# **Reactive Streams & Spring WebFlux**

Букин Д.Ю., 4ИВТ(2)

---

# Содержание

- О реактивном программировании
  - Reactive Streams API
  - Project Reactor
  - Spring WebFlux
  - DI и IoC
  - *<код>*
  - Корутины Kotlin
  - *<код>*
-

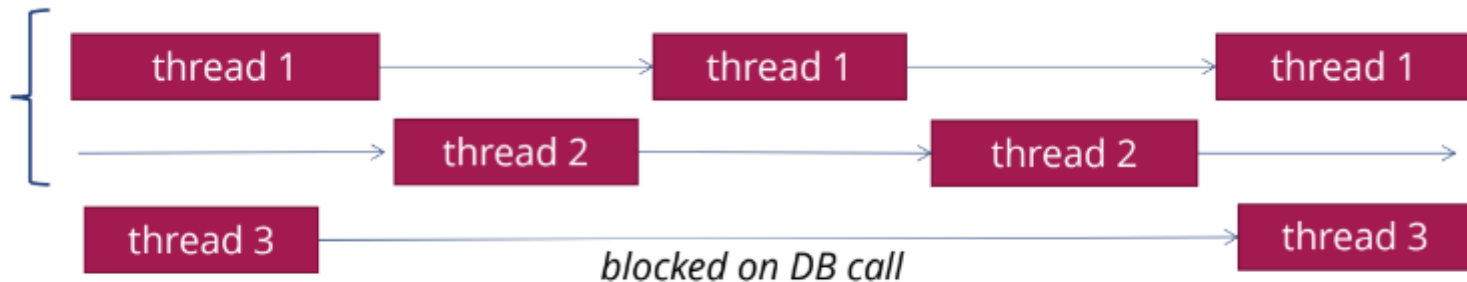


# Реактивное программирование

Что это?

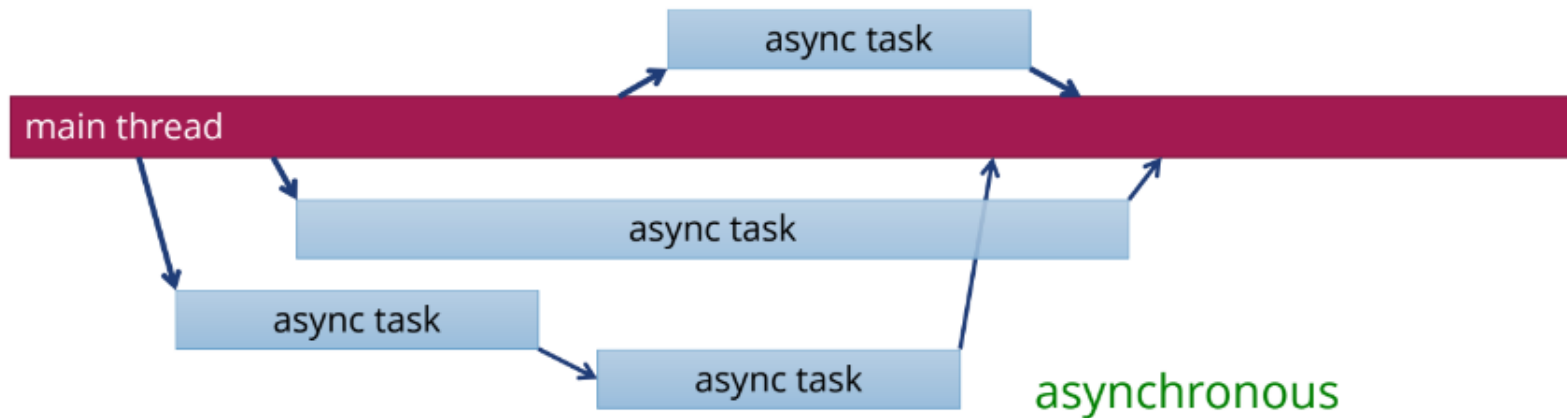
---

thread 1&2  
use same  
**shared  
resource**

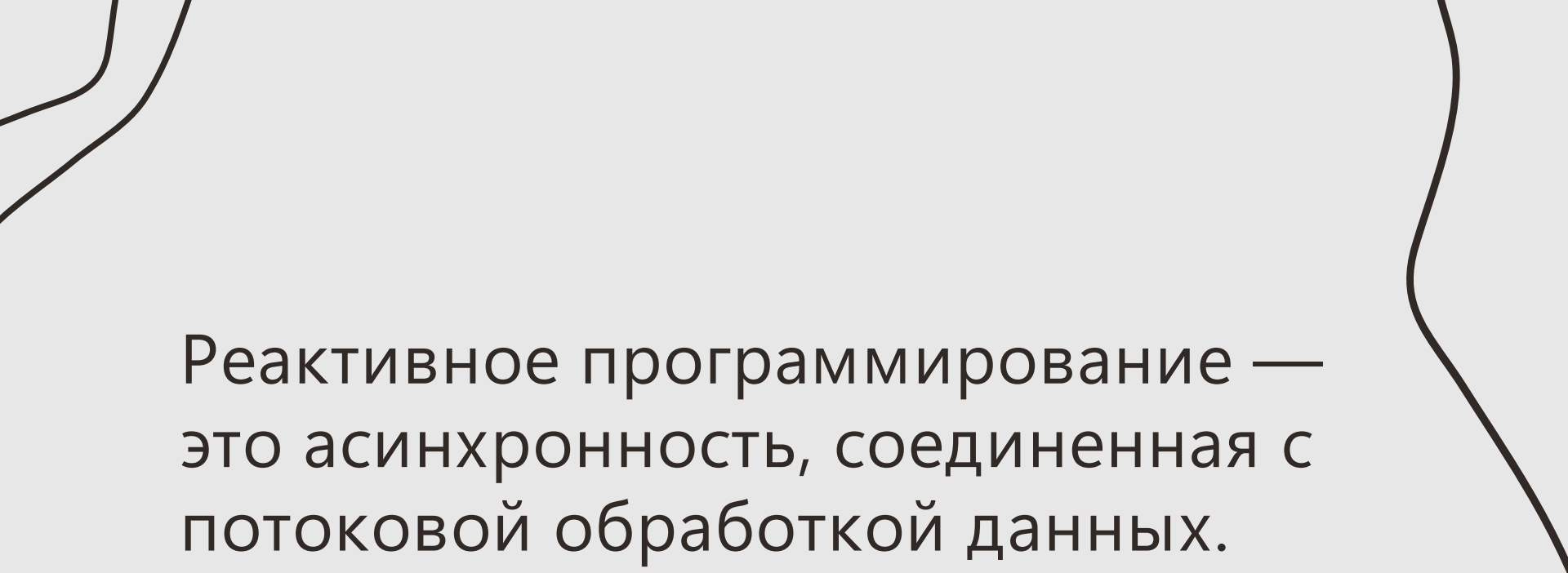


parallel

Многопоточное  
программирование – сложно,  
мучительно, все устали



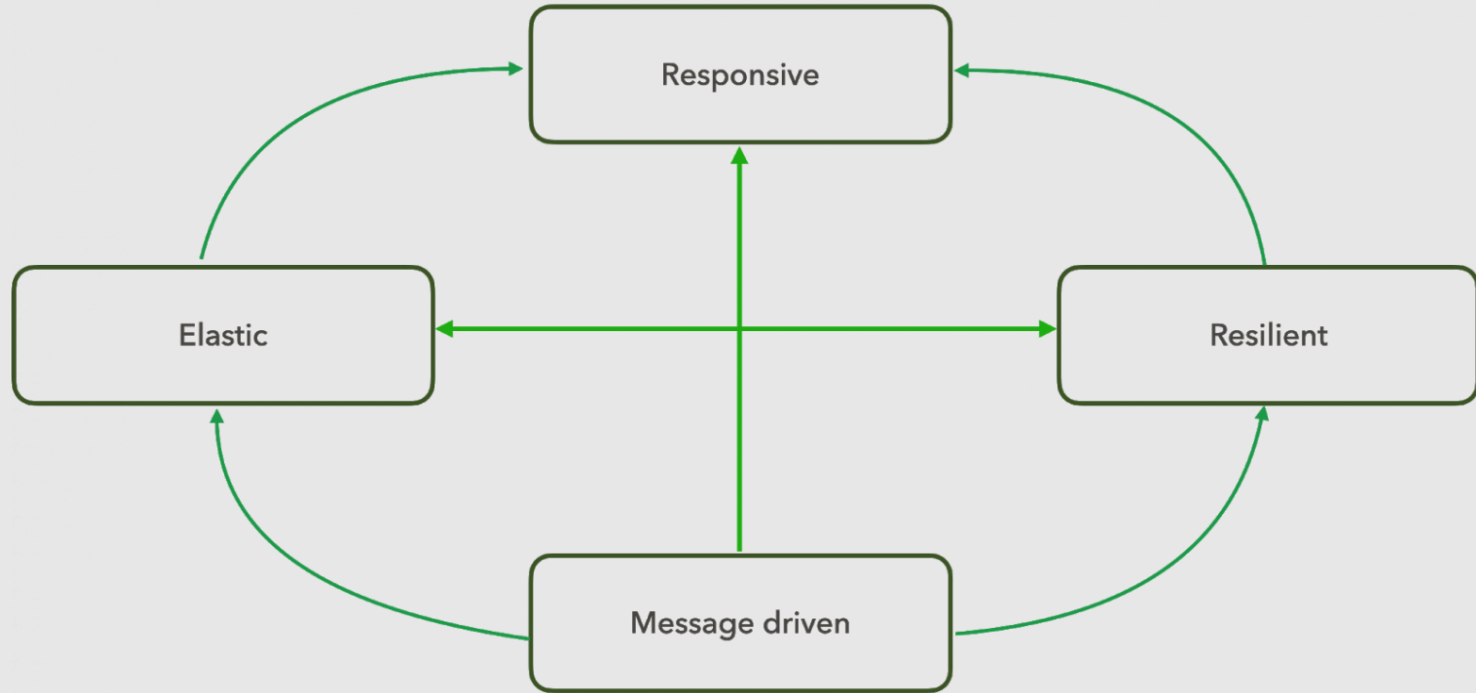
Асинхронное программирование –  
легче, но проще без всего этого...



Реактивное программирование —  
это асинхронность, соединенная с  
поточковой обработкой данных.

---

## Reactive Manifesto



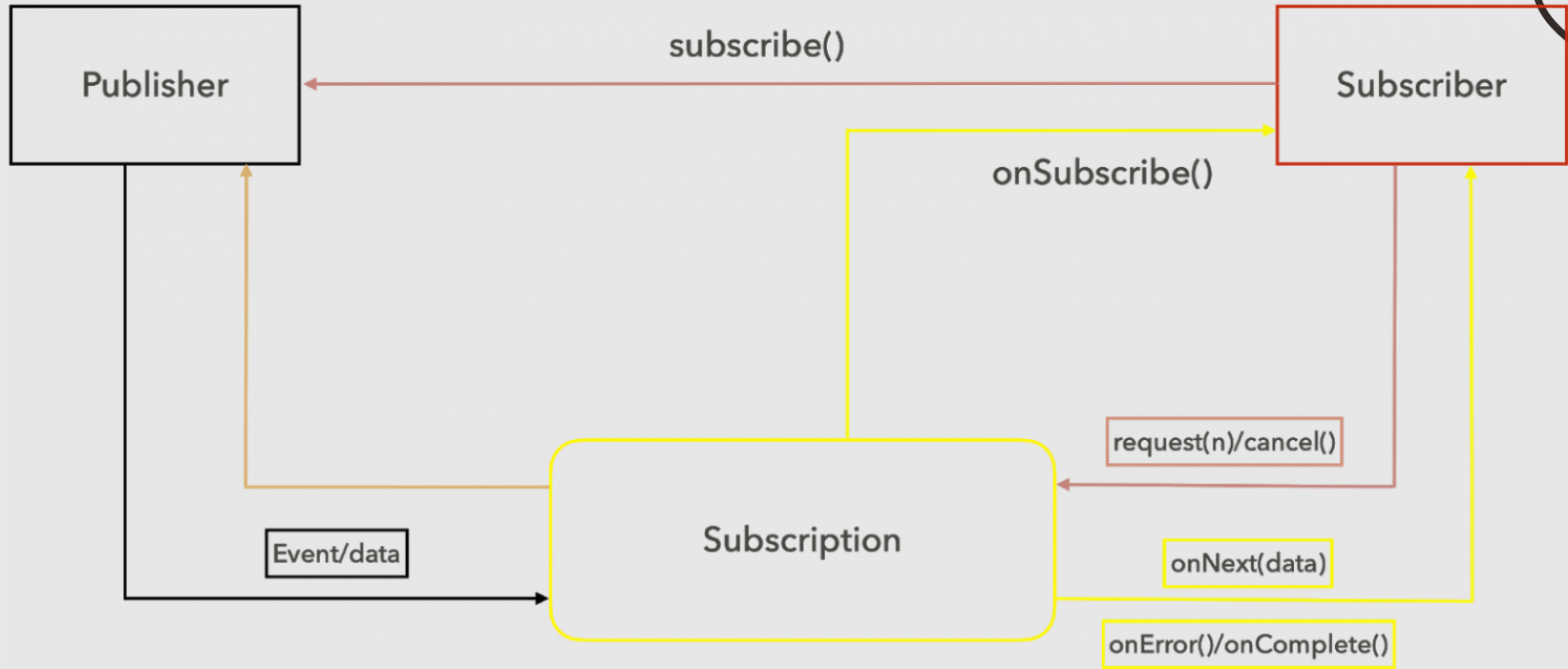
# Reactive Streams API

Интерфейсы:

- subscriber
  - publisher
  - subscription
  - processor
-



## Reactive Streams



# Известные библиотеки

- RxJava (фреймворк от ReactiveX, используется в Android)
  - Akka Streams (форк акторной модели языка Scala)
  - Project Reactor (наш случай)
-

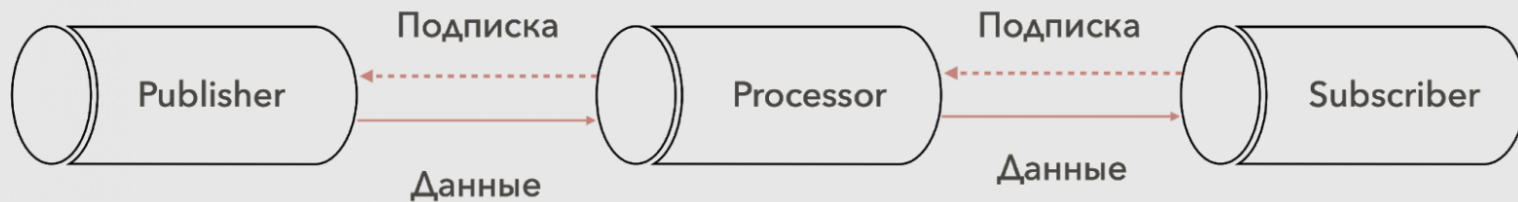


# Project Reactor

Что нового?

---

## Модель работы



## Базовые типы Publisher

### Виды publisher

Hot Publisher

Cold Publisher

От 0 до N	Flux<T>
От 0 до 1	Mono<T>
Всегда 0	Mono<Void>



# Spring **Boot** 2



## Reactor

Optional Dependency

### Reactive Stack

Spring WebFlux is a non-blocking web framework built from the ground up to take advantage of multi-core, next-generation processors and handle massive numbers of concurrent connections.

Netty, Servlet 3.1+ Containers

Reactive Streams Adapters

Spring Security Reactive

Spring WebFlux

Spring Data Reactive Repositories

Mongo, Cassandra, Redis, Couchbase, R2DBC

### Servlet Stack

Spring MVC is built on the Servlet API and uses a synchronous blocking I/O architecture with a one-request-per-thread model.

Servlet Containers

Servlet API

Spring Security

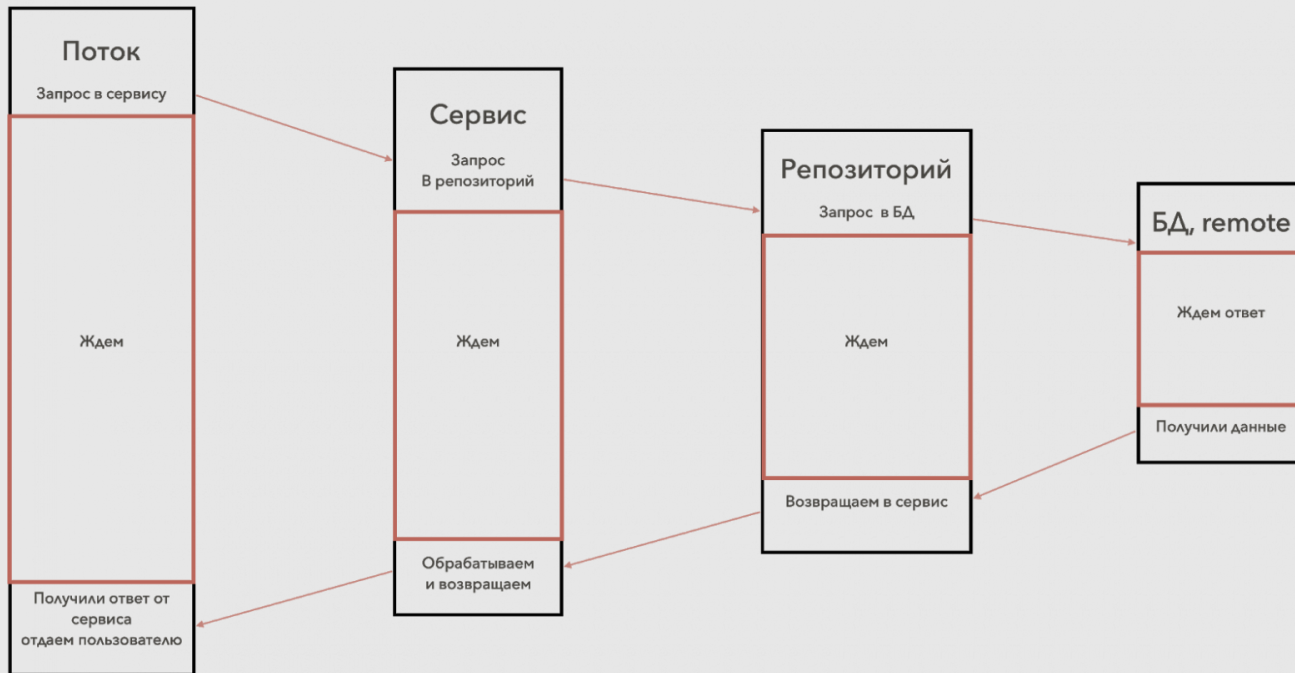
Spring MVC

Spring Data Repositories

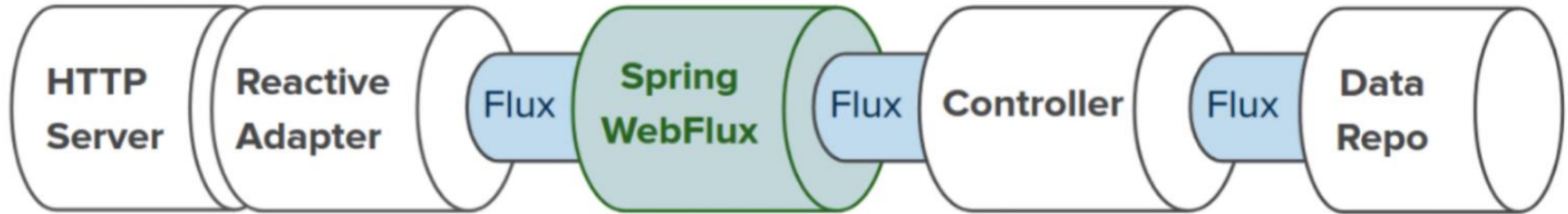
JDBC, JPA, NoSQL

# Классическое приложение

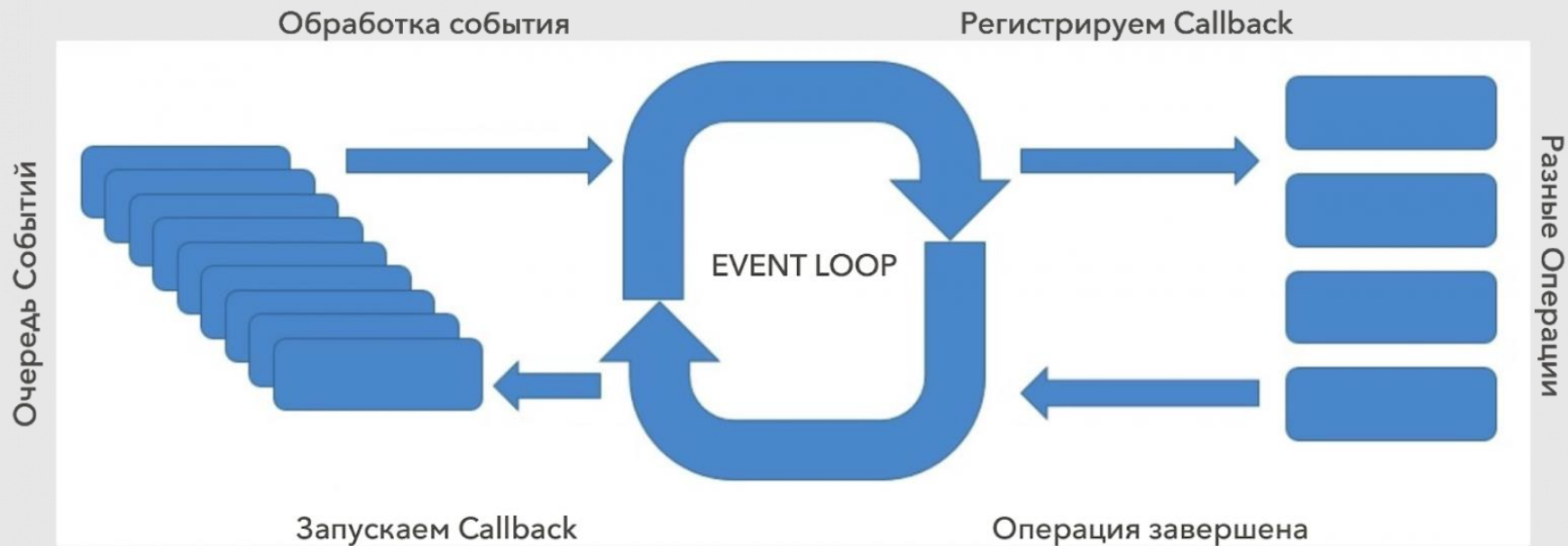
Classic



# Архитектура приложения



## Более Подробно о Netty

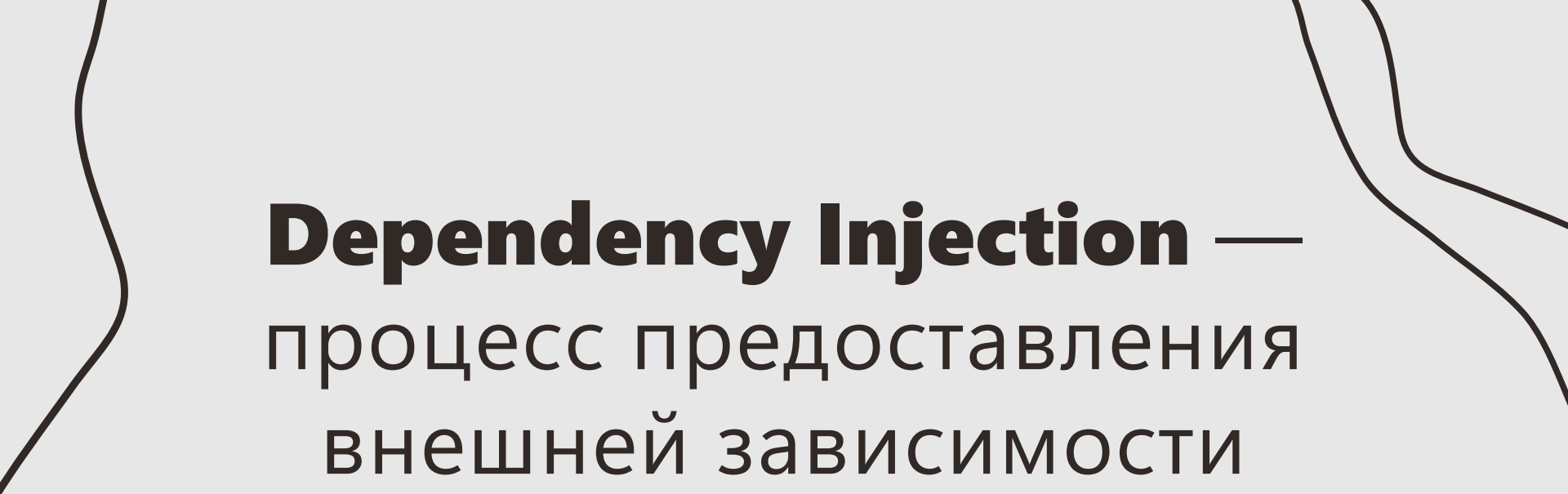




# **Dependency Injection Inversion of Control**

Прежде чем перейти к коду

---



**Dependency Injection** —  
процесс предоставления  
внешней зависимости  
программному  
компоненту

---

```
interface ILog
    void print(mes) // Интерфейс

class OldLogger : ILog
    override void print(mes) => log(mes)

class NewLogger : ILog
    override void print(mes) => logPretty(mes) // Конкретные реализации

class Smth
    ILog logger

    Smth(ILog logger) => logger = logger

    void do() {
        mes = "Ok"
        logger.print(mes) // Класс, не зависящий от реализации
    }

void Main() {
    smth1 = new Smth(new OldLogger())
    smth2 = new Smth(new NewLogger()) // Внедрение зависимостей вручную
}
```

Псевдокод

# IoC

Набор рекомендаций для написания слабо связанного кода. DI - частный случай IoC

---

# Некоторые аннотации

@Bean – Для методов

@Component

@Service

@Repository

@Configuration

---

# О приложении

GET /api/anime/{id} – получение аниме по id и одного случайного фильма

GET /api/movie?count={n} – получение нескольких случайных фильмов

DB – MongoDB

Anime API - jikan

---



**Перейдём к коду!**

---



# Корутины в Kotlin

И это очень красиво

---



# О корутинах

- Корутина – блок кода, который может выполняться параллельно с остальным кодом
  - Корутина не привязана к конкретному потоку
  - Корутина может выполняться только в определенной области корутины
  - Для создания корутины нужен построитель корутин
-

# Пример корутины

```
import kotlinx.coroutines.*

suspend fun main() {
    doWork()
}

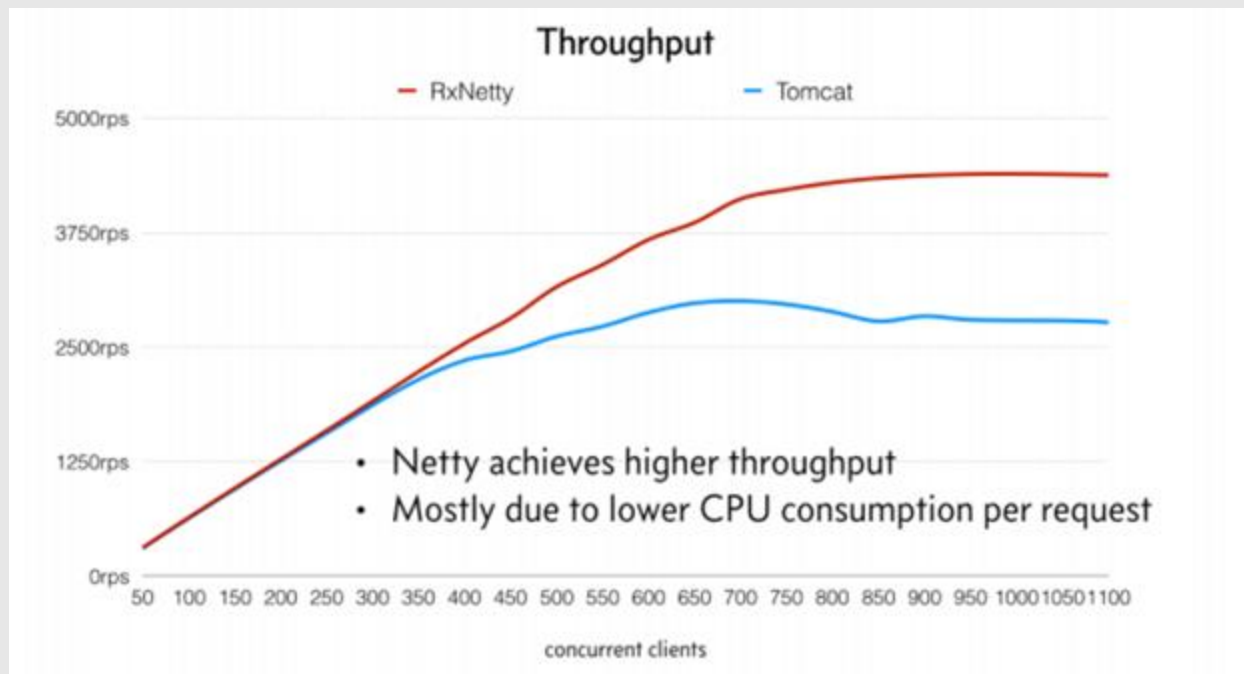
suspend fun doWork() = coroutineScope { // Область корутины
    launch { // Построитель корутины
        for (i in 0..5) {
            println(i)
            delay(400L)
        }
    }
}
```



**Перейдём к коду!**

---

# Выводы



Это не про скорость, это про количество  
одновременных запросов

Да, приложение может выдержать большее количество соединений, но надо ли использовать WebFlux, если у вас не highload?

(Вряд ли, т.к. overhead)

(Но ознакомиться можно)

---

# Источники и материалы

- Spring WebFlux: Реактивное программирование веб-сервисов
  - Spring MVC vs Spring WebFlux. Что лучше? Объясняем на пингвинах
  - Реактивное программирование со Spring (5 частей)
  - Реактивное программирование на Java: как, зачем и стоит ли? (часть 1, часть 2)
-

# Источники и материалы

- [Going Reactive with Spring, Coroutines and Kotlin Flow](#)
  - [Spring Boot with Kotlin Coroutines and Rsocket](#)
  - [Создание и тестирование неблокирующих веб-приложений с помощью Spring WebFlux, Kotlin и Coroutines](#)
  - <https://metanit.com/kotlin/tutorial/8.1.php>
  - <https://www.baeldung.com/>
-