Києво-Могилянська академія

РОЗРОБКА РОЗПОДІЛЕНОЇ СИСТЕМИ З АСИНХРОННОЮ КОМУНІКАЦІЄЮ З ВИКОРИСТАННЯМ SPRING WEBFLUX

ВИКОНАВ

Буковський С. В.

НАУКОВИЙ КЕРІВНИК

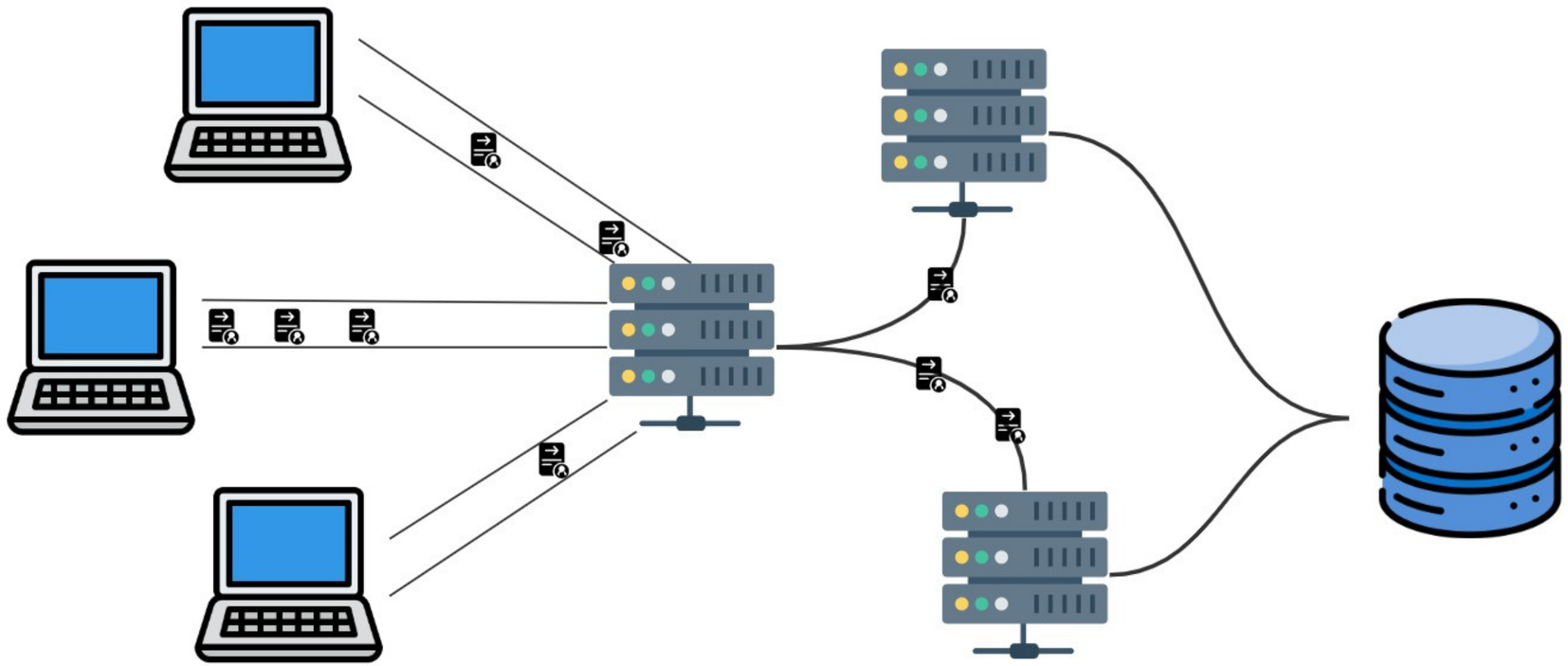
Андрощук М. В.

15.05.2024

Agenda

1. Проблематика
2. Означення реактивного програмування
3. Java-реалізації
4. Spring WebFlux
5. Event Sourcing та CQRS

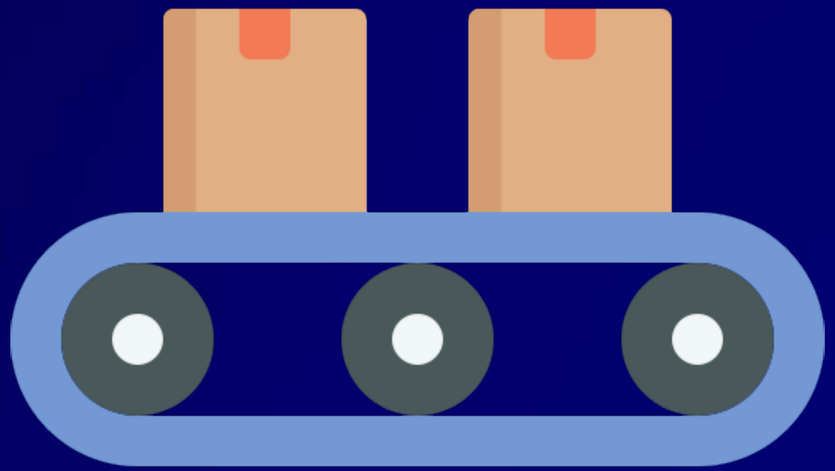
Проблематика



Реактивне програмування - ефективне рішення

Реактивне програмування - це парадигма програмування, яка фокусується на обробці потоків даних та подій.

- Асинхронний і неблокуючий код.
- В основі реактивного програмування лежить ідея реагування на події, а не блокування та очікування на них.
- Потоки даних і подій замість об'єктів і методів.



Базова абстракції

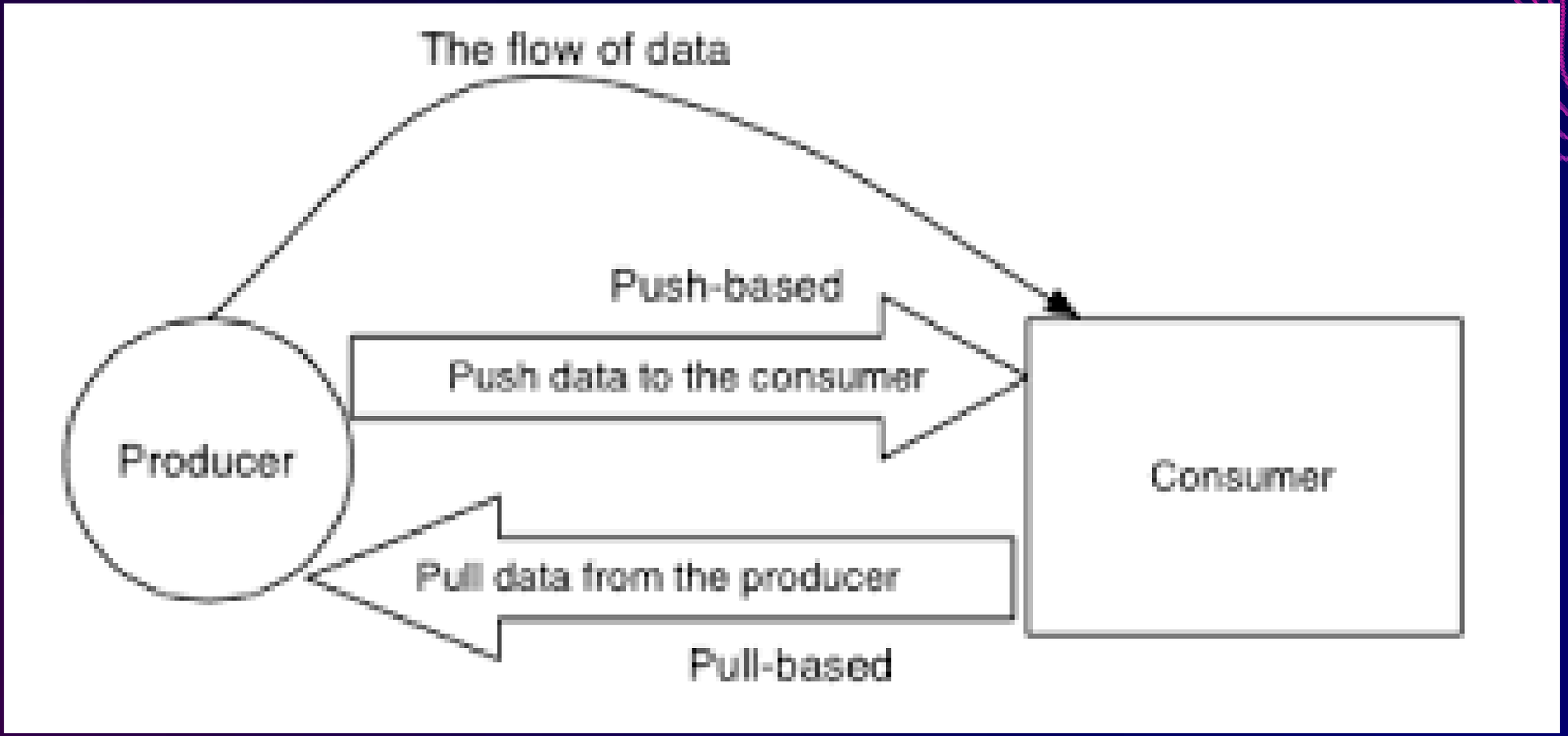
Observables представляє потік даних у часі, який може набувати різних значень.

Events - події які відбувається у потоці: next, complete, error.

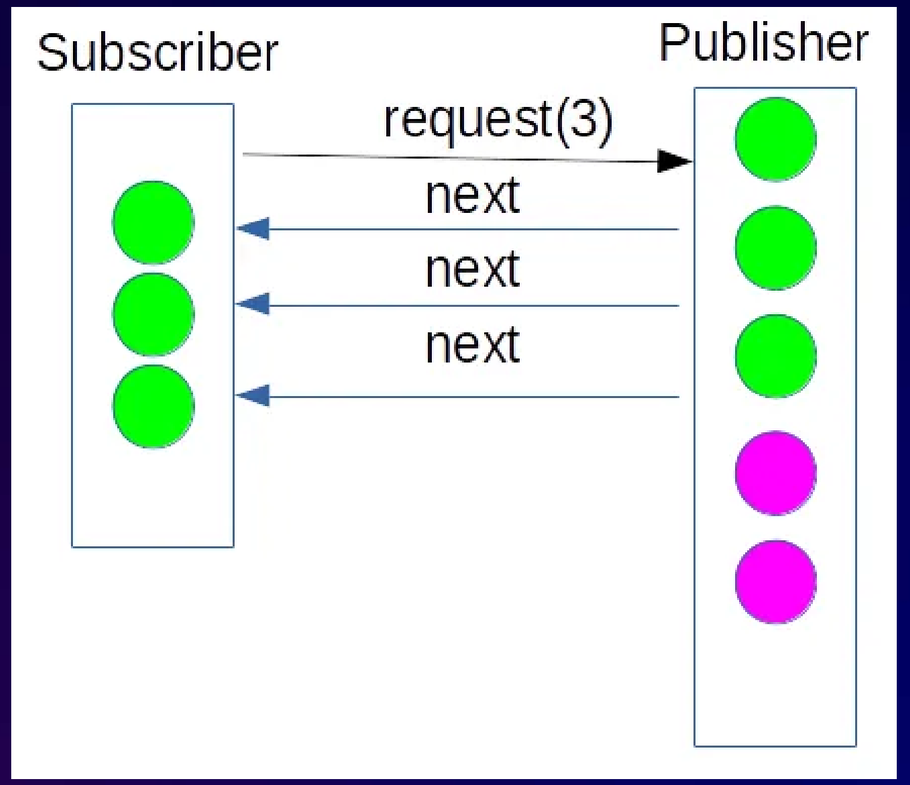
Observer - споживає події.

Operators - функціональні перетворення, що здійснюються з елементами потоку.

Модель виконання



Реактивне Програмування = Pull + Push



Java - бібліотеки

RxJava

Великий вибір операторів.

Розширення та адаптери.

Стратегії Backpressure.

Багата екосистема.

Великий вибір Observables

Project Reactor

Тісна інтеграція з Spring.

Поширення контексту.

Гарячі та холодні потоки.

Сумісність з Reactive Streams

Mutiny

Зручний та зрозумілий API.

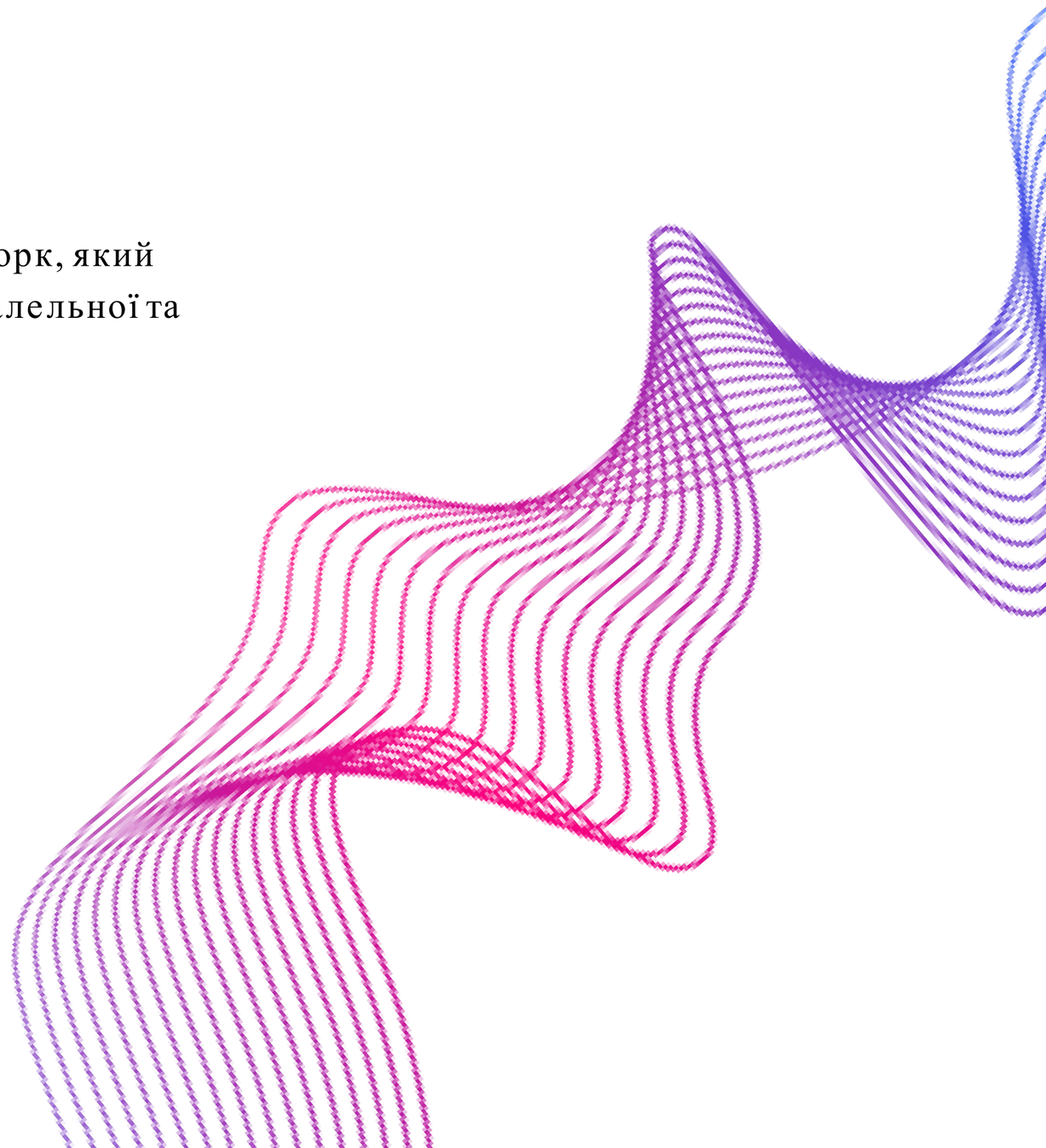
Оптимізовано для Quarkus.

Спрощена обробка помилок.

Поширення контексту

Spring WebFlux

Spring WebFlux - це реактивний, неблокуючий вебфреймворк, який використовує Project Reactor для забезпечення високопаралельної та асинхронної обробки вебзапитів



- Неблокуючі I/O
- Підтримка реактивних потоків
- Функціональна модель програмування
- Підтримка Server-sent events і WebSockets
- Реактивний доступ до баз даних
- Підтримує кешування

```java
public interface TaskRepository extends ReactiveMongoRepository<TaskDocument, String> {
   //..
}
```

```java
public interface TaskRepository extends MongoRepository<TaskDocument, String> {
   //..
}
```

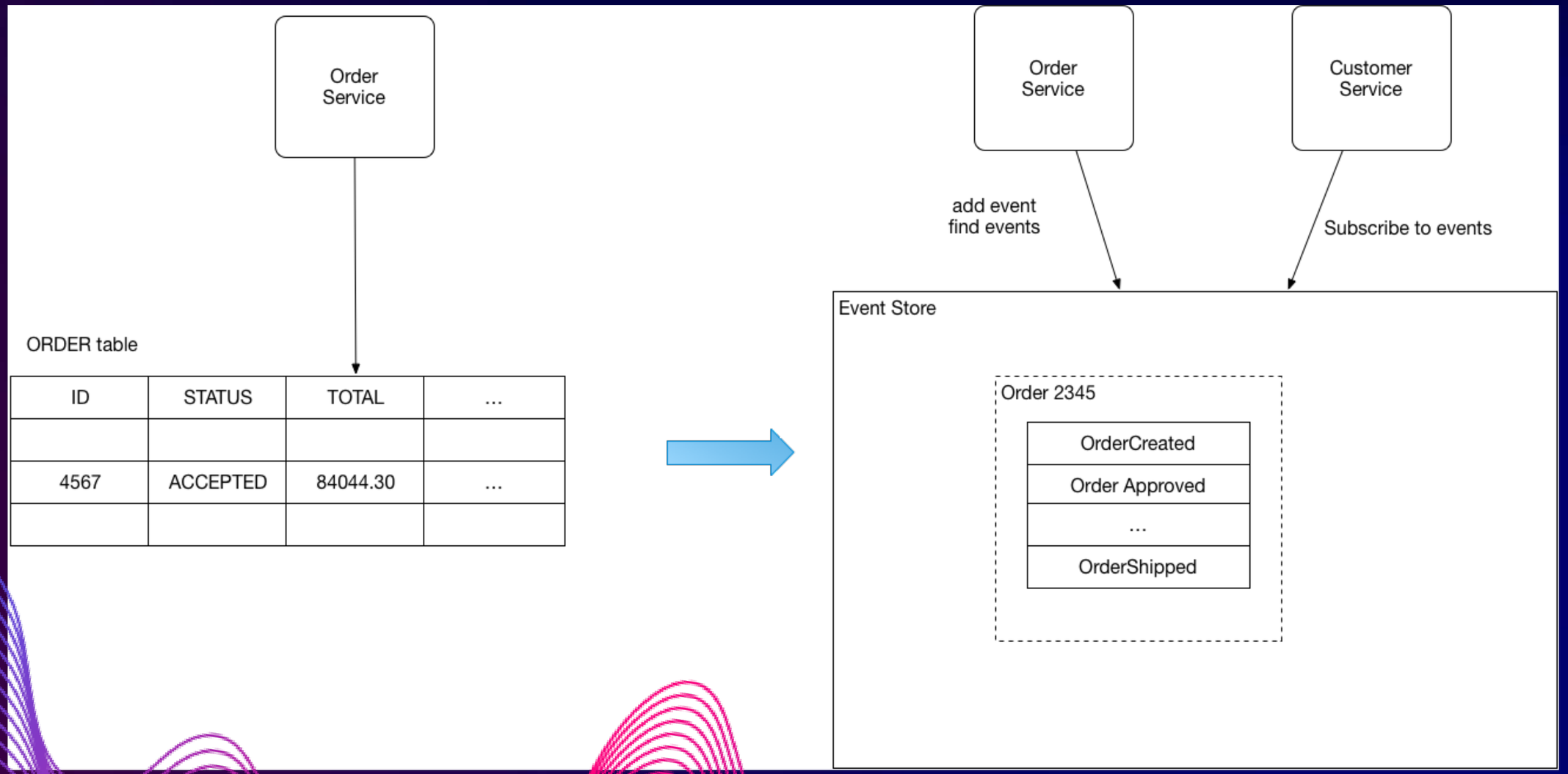
```java
@RestController
public class TaskController {
    //...

    @GetMapping("/tasks")
    public List<Task> getTasks(@RequestParam(required = false) String status,
                               @RequestParam(required = false) String assignee) {
        return taskService.getTasks(status, assignee);
    }
}
```

```java
@RestController
public class TaskController {
    //...

    @GetMapping("/tasks")
    public Flux<Task> getTasks(@RequestParam(required = false) String status,
                               @RequestParam(required = false) String assignee) {
        return taskService.getTasks(status, assignee);
    }
}
```

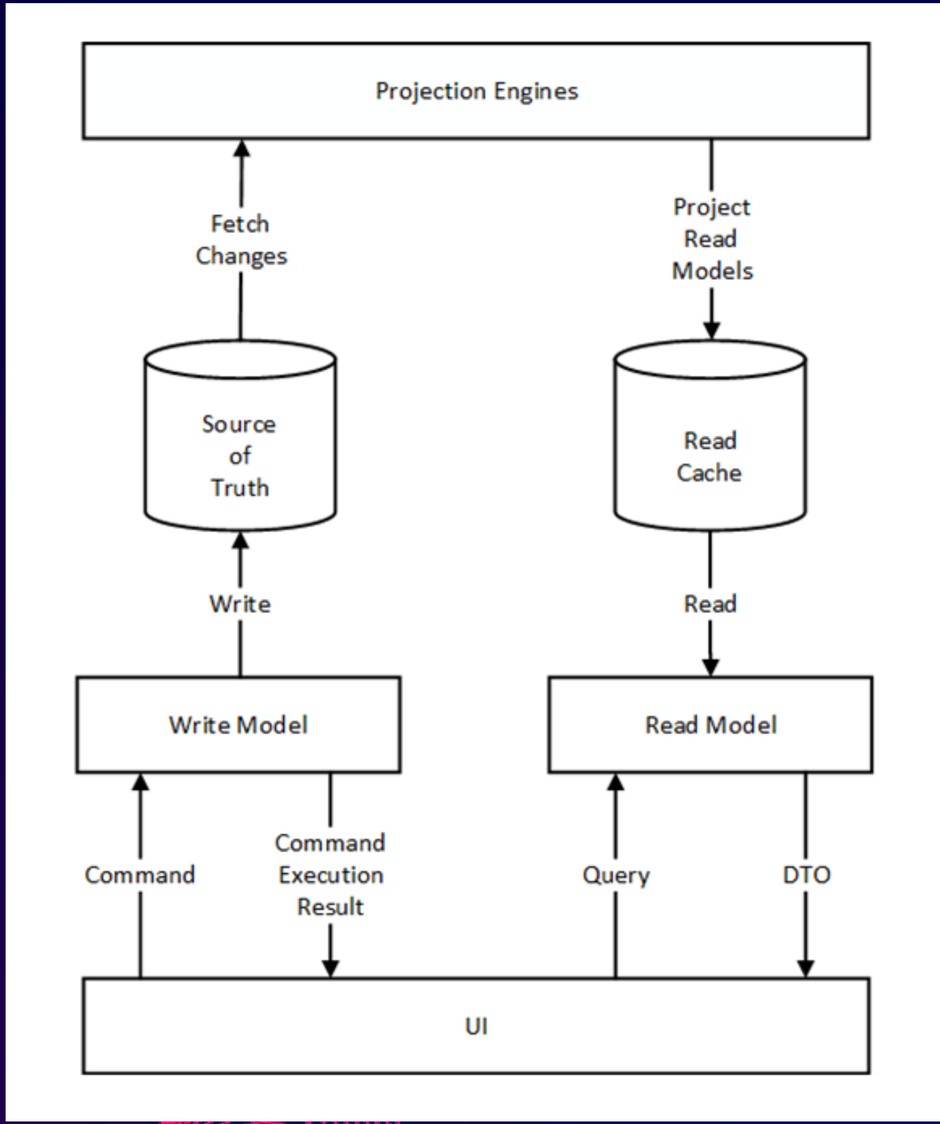
Найбільша відмінність - думати реактивно

```
public Mono<Void> inviteUsersToBoard(InviteUsersCommand command) {
        return boardService.getBoardAgg(command.boardId(), command.fromUserId())
                .zipWhen(board -> eventStore.exists(command.fromUserId(), UserAggregate.AGGREGATE_TYPE)
                        .handle((exists, sink) -> {
                            if (!exists) {
                                sink.error(new ItemNotFoundException("User not found for id: " + command.fromUserId()));
                            } else {
                                sink.next(true);
                            }
                        })
                        .flatMapMany(ignored2 -> Flux.fromIterable(command.toUserIds())
                                .filter(userId -> !board.getInvitedIds().contains(userId)
                                        && !board.getJoinedIds().contains(userId) && !board.getOwnerId().equals(userId))
                                .publishOn(Schedulers.boundedElastic())
                                .map(userId -> {
                                    var isExist = eventStore.exists(userId, UserAggregate.AGGREGATE_TYPE).block();
                                    return Tuples.of(userId, isExist == Boolean.TRUE);
                                })
                                .filter(Tuple2::getT2)
                                .map(Tuple2::getT1)
                        )
                        .collectList())
                .map(tuple -> {
                    final var board = tuple.getT1();
                    final var existedUserIds = tuple.getT2().stream()
                            .filter(userId -> !board.getInvitedIds().contains(userId));

                    existedUserIds.forEach(board::inviteUser);
                    return board;
                })
                .flatMap(eventStore::save)
                .then();
    }
```

Event Sourcing —історична модель мислення



CQRS або розділяй і пануй



Висновки

Основи реактивного програмування: Розуміння переходу від імперативного до декларативного програмування, робота з асинхронними потоками даних та управління backpressure.

Можливості Spring WebFlux: Використання неблокуючого вводу/виводу Spring WebFlux, підтримка реактивних потоків та інтеграція з ширшою екосистемою Spring.

Практичне застосування: Реалізація реактивної системи управління завданнями, яка підкреслює переваги оперативності, масштабованості та відмовостійкості в реальних додатках.

https://github.com/stas-bukovskiy/task-manager

Дякую за увагу!

Q&A