Reaktív programozás szerver oldalon

Tóth Márton

Supercharge

Agenda

- Reactive streams
- Spring WebFlux
  - Reactive Spring Data
- Benchmarks
- Konklúzió

Reactive streams

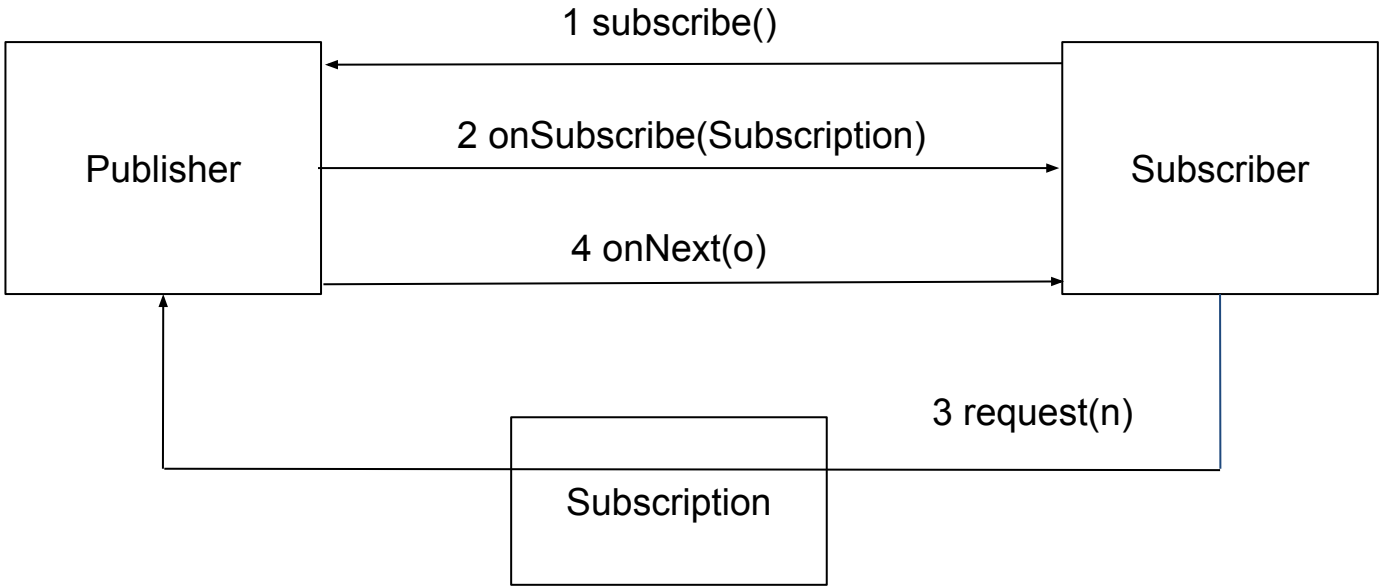
Reactive streams

- **Reaktív** programozási modell
- **Aszinkron** adatfeldolgozó folyamok **non-blocking backpressure**-rel
  - **Aszinkron:** egy szálon több feldolgozás
  - **Non-blocking:** A feldolgozó szálon nem végzünk blokkoló műveletet
  - **Backpressure**: A fogyasztó lassabb mint a termelő
- **Specifikáció**, több implementációval: http://reactive-streams.org
  - RxJava, ReactiveX, Java 9 Flow...

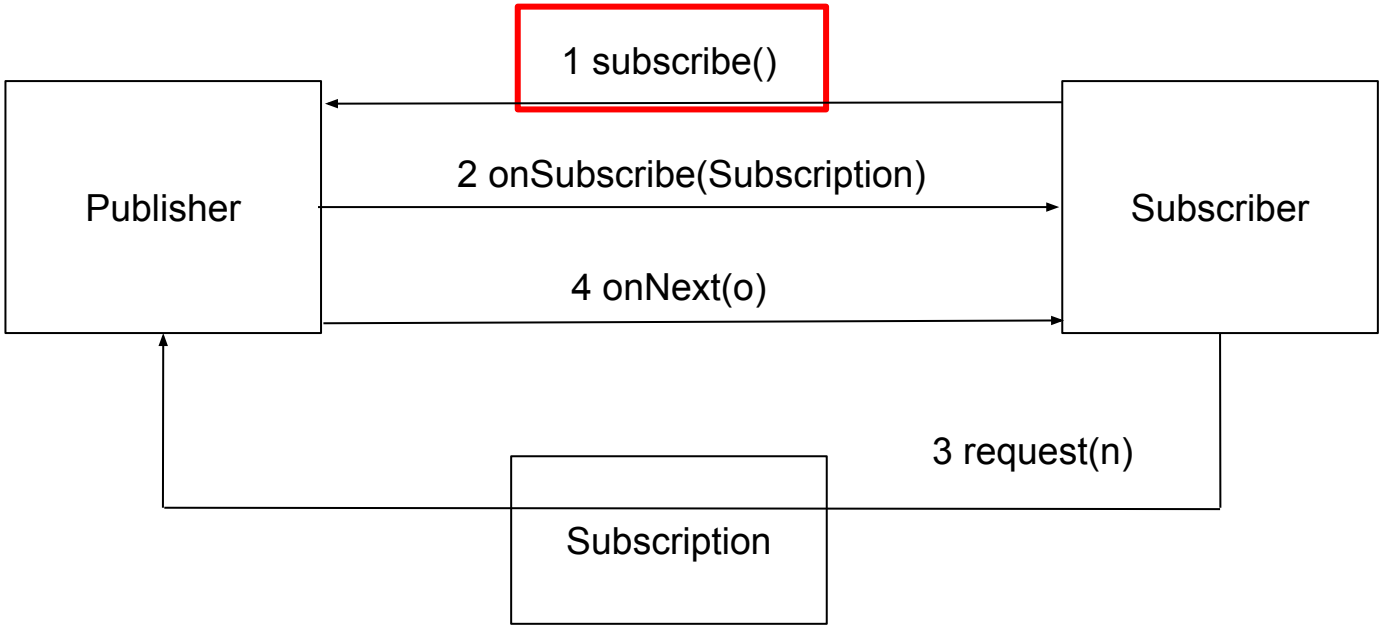
Mire jó ez az egész?

- Szálak: **overhead**
  - Szál indítás, szálak közti váltás, stack memória...
- **Aszinkronitás**: kevés szállal elvégezhető **ugyanaz a munka**
- Jobb áteresztőképesség, kiszámíthatóbb viselkedés terhelés alatt

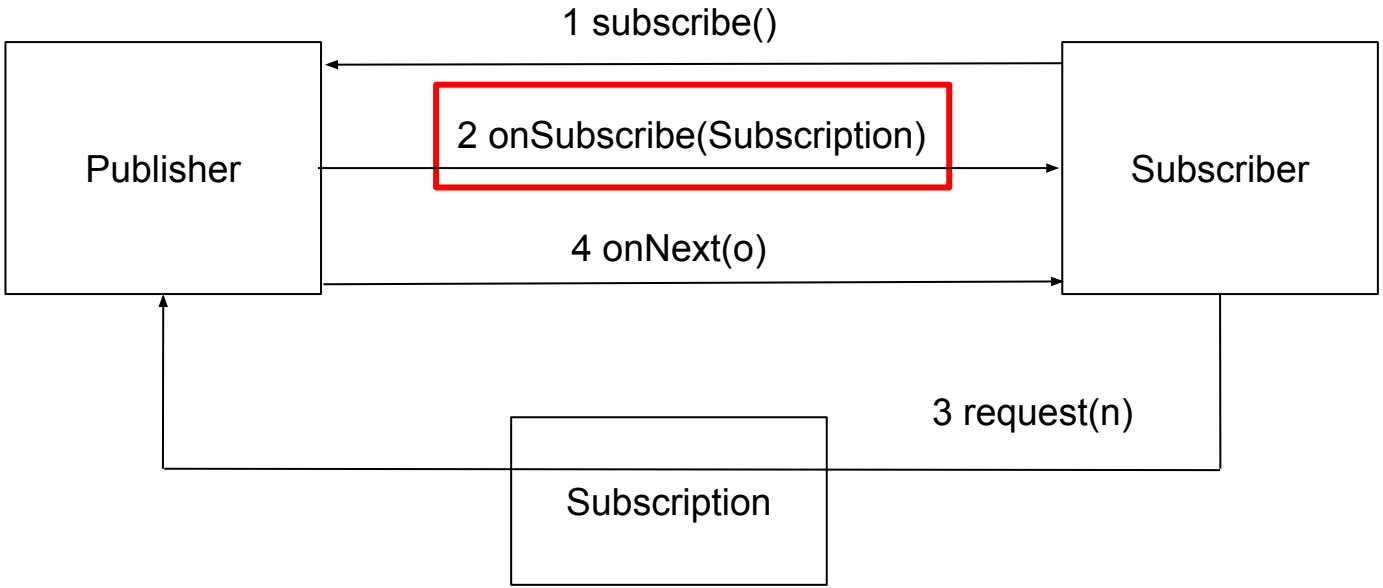
Reactive flow



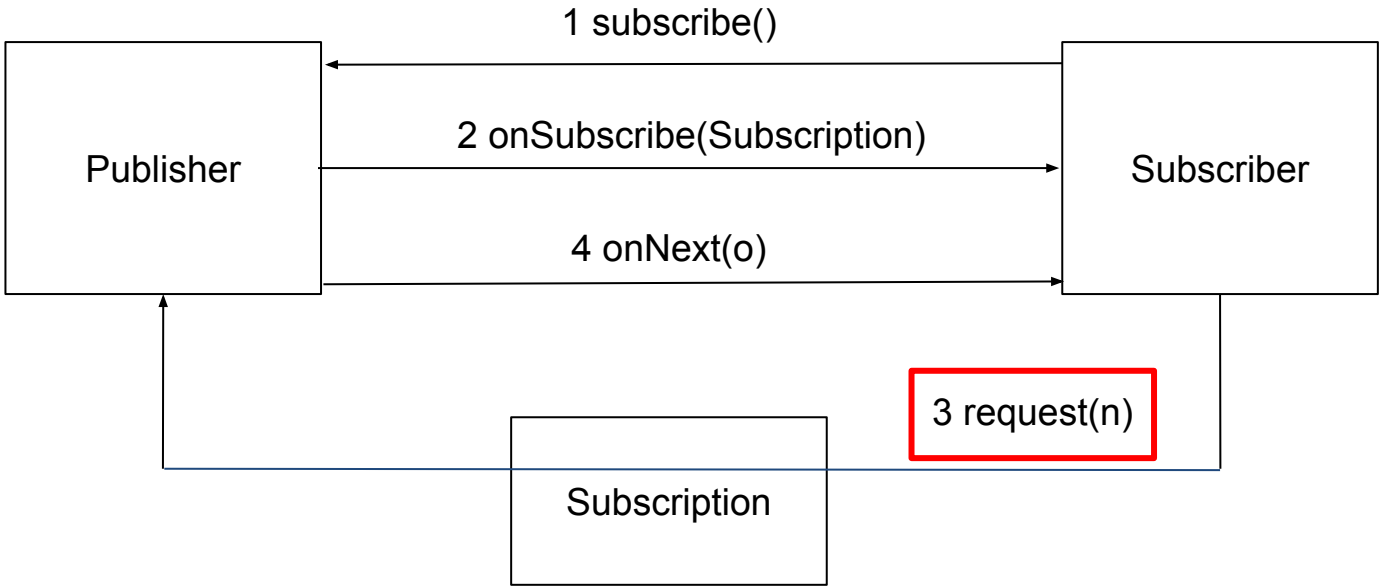
Reactive flow



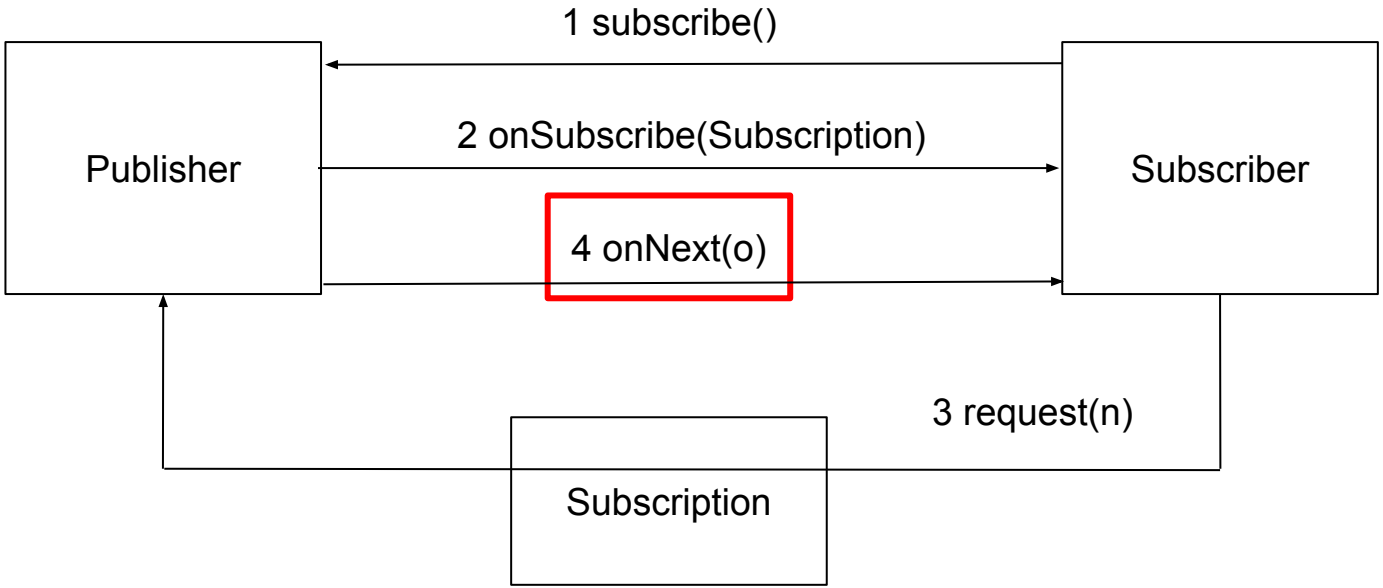
Reactive flow



Reactive flow



Reactive flow



Reactor project

- Reactive streams implementáció
- **API**-ja hasonlít a **Java Stream API**-hoz
- Források: **Flux**, **Mono**
- **Operátorok**: map, flatMap, zip, filter, cache, stb...

Példa

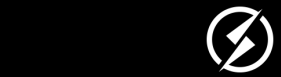
```
List<String> animals = ImmutableList.of("dog", "cat", "unicorn");
```

Stream API

```
animals.stream()
        .map(String::toUpperCase)
        .filter(a -> a.contains("O"))
        .forEach(System.out::println);
```

Reactor

```
Flux.fromIterable(animals)
        .map(String::toUpperCase)
        .filter(a -> a.contains("O"))
        .subscribe(System.out::println);
```



Spring + Reactive = WebFlux

Spring WebFlux

- Reaktív web keretrendszer
  - Kliens és szerver implementáció
- Reactor project
- Spring Web MVC alternatíva
  - **Mellett, nem helyett**

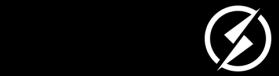
WebFlux példa

```
@RestController
public class Controller {

    @GetMapping
    public Mono<String> handle() {
        return Mono.just("unicorn")
                .flatMapMany(s ->
                     Flux.fromArray(s.split(""))
                )
                .filter(s -> !s.equals("o"))
                .collect(Collectors.joining());
    }
}
```

- Spring **Web MVC annotációk**
- Handler metódusok **visszatérési értéke**

  Flux vagy **Mono**
- A streamre a framework iratkozik fel
- A streamet **tilos** blokkolni



Mi blokkol jellemzően szerver oldalon?

- **Hálózat**
  - **HTTP**: WebClient
  - **Egyéb**: reactor-netty, salesforce/reactive-grpc, stb
- **Fájl műveletek**
  - **Java NIO**, Stream API
- **Adatbázisok**
  - **Spring Data**

Non-blocking Spring Data

- **Flux, Mono** egyenesen a **Repository-**kból
- **NoSQL** adatbázisok
  - MongoDB, Cassandra, Couchbase, Redis…
- **Relációs adatbázisok**
  - JDBC...


BLOCKING
LIGHTYEAR
BLOCKING EVERYWHERE
imgflip.com

R2DBC to the rescue

- Nem blokkoló **adatbázis connector**
  - Postgres, MSSQL, H2, MySQL
- **Spring Data** R2DBC
  - Standard Spring Data **Repository**-k
  - **Lightweight mapping**, nem ORM
  - **Nincs**: caching, lazy loading, write behind, egyéb feketemágia
- **Nagyon fiatal** projekt

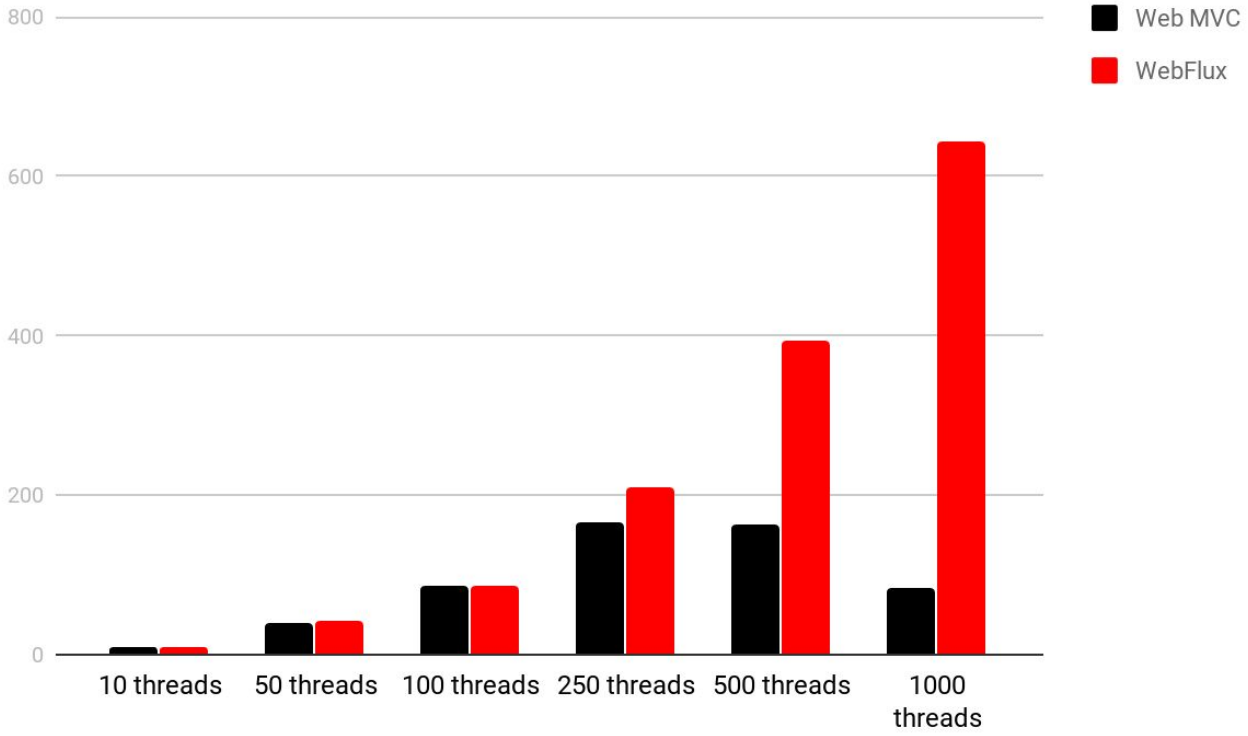
SUPER CHARGE

Benchmarks

Setup

- **2 Alkalmazás**
  - **Web MVC** vs **WebFlux**
  - **1 másodperces késleltetésű** szolgáltatás meghívása
  - **REST** interfész
- **AWS EC2 t3.micro**
  - 1 GiB RAM, 2 vCPU
- **JMeter load test**, külön EC2 instance-ről

Throughput (per second)




BRILLIANT!

Konklúzió

- Használd ha
  - Várhatóan nagy lesz a terhelés
  - Nem rettensz meg az érdekes framework bugoktól 
- Ne használd ha
  - Nem követelmény a brutális áteresztőképesség
  - Kevésbé tapasztalt a csapatod

SUPER CHARGE

We are hiring!

careers@supercharge.io


SUPER
CHARGE