Cas d'utilisation avec Spring Boot

#1 Intégration de Vavr

Support de Spring Web MVC (JSON)

- Via le module vavr-jackson :

```
<dependency>
    <groupId>io.vavr</groupId>
    <artifactId>vavr-jackson</artifactId>
    <version>0.9.0</version>
</dependency>
```

- Et la configuration du parseur :

```
@Bean
public ObjectMapper jacksonBuilder() {
    ObjectMapper mapper = new ObjectMapper();
    return mapper.registerModule(new VavrModule());
}
```

- La conversion en JSON fonctionne :

```
import io.vavr.collection.List;

@RestController
@RequestMapping("/v1/persons")
public class PersonV1Controller {

    @GetMapping
    public List<Person> findAll() {

        return List.of(new Person());
    }
```

Support de Spring Data

```
import io.vavr.collection.List;


public interface PersonRepository extends Repository<Person, Long> {
    List<Person> findAll();
}
```

Reactive ?

Modèle de programmation basé sur la **réaction au changement**.

Ingrédients

- Asynchronisme
- Opérations non-bloquantes
- Gestion de la pression arrière

Reactive Streams

- http://www.reactive-streams.org/
- Initiative pour fournir un standard pour la JVM
- Intégré nativement dans le JDK depuis Java 9, accessible via une dépendance externe pour les versions antérieures.

Interfaces

Publisher

```
public interface Publisher<T> {
    public void subscribe(Subscriber<? super T> s);
}
```

Subscriber

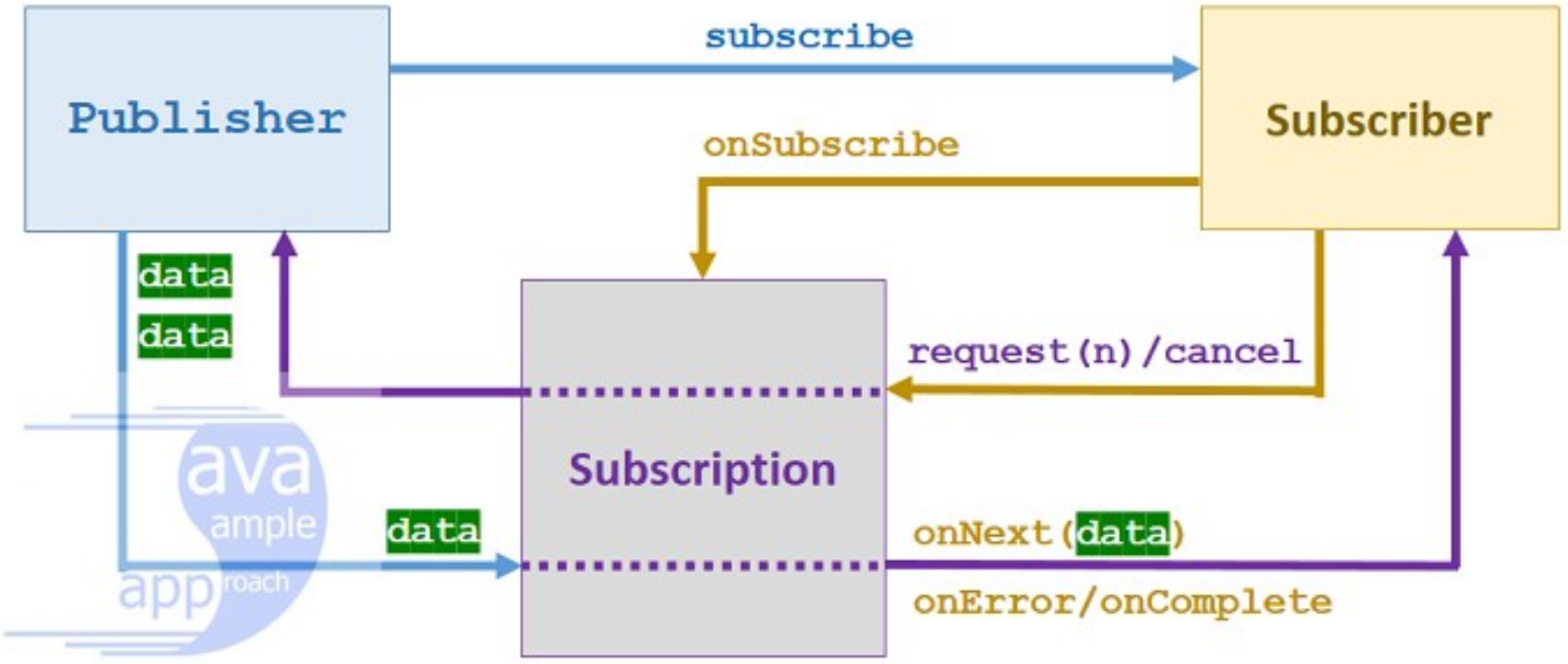
```
public interface Subscriber<T> {
    public void onSubscribe(Subscription s);
    public void onNext(T t);
    public void onError(Throwable t);
    public void onComplete();
}
```

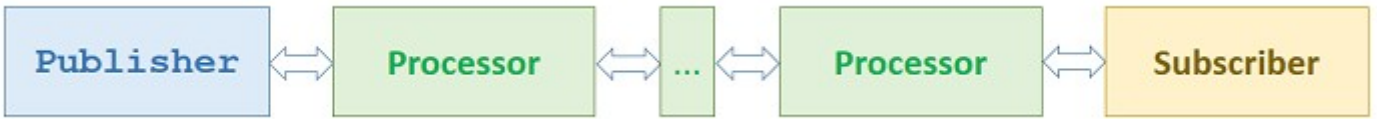
Subscription

```
public interface Subscription {
    public void request(long n);
    public void cancel();
}
```

Processor

```
public interface Processor<T, R> extends
            Subscriber<T>, Publisher<R> {
}
```





Quelques implémentations

- Reactor (Spring)
- RxJava 2 (Netflix)
- Akka Streams (TypeSafe)

RxJava 2

- Observable (0 -N)
- Flowable (0 - N, implements Publisher)
- Single (0 - 1)
- Completable (séquence vide)

Akka Streams

- Source
- Sink
- Flow

Reactor

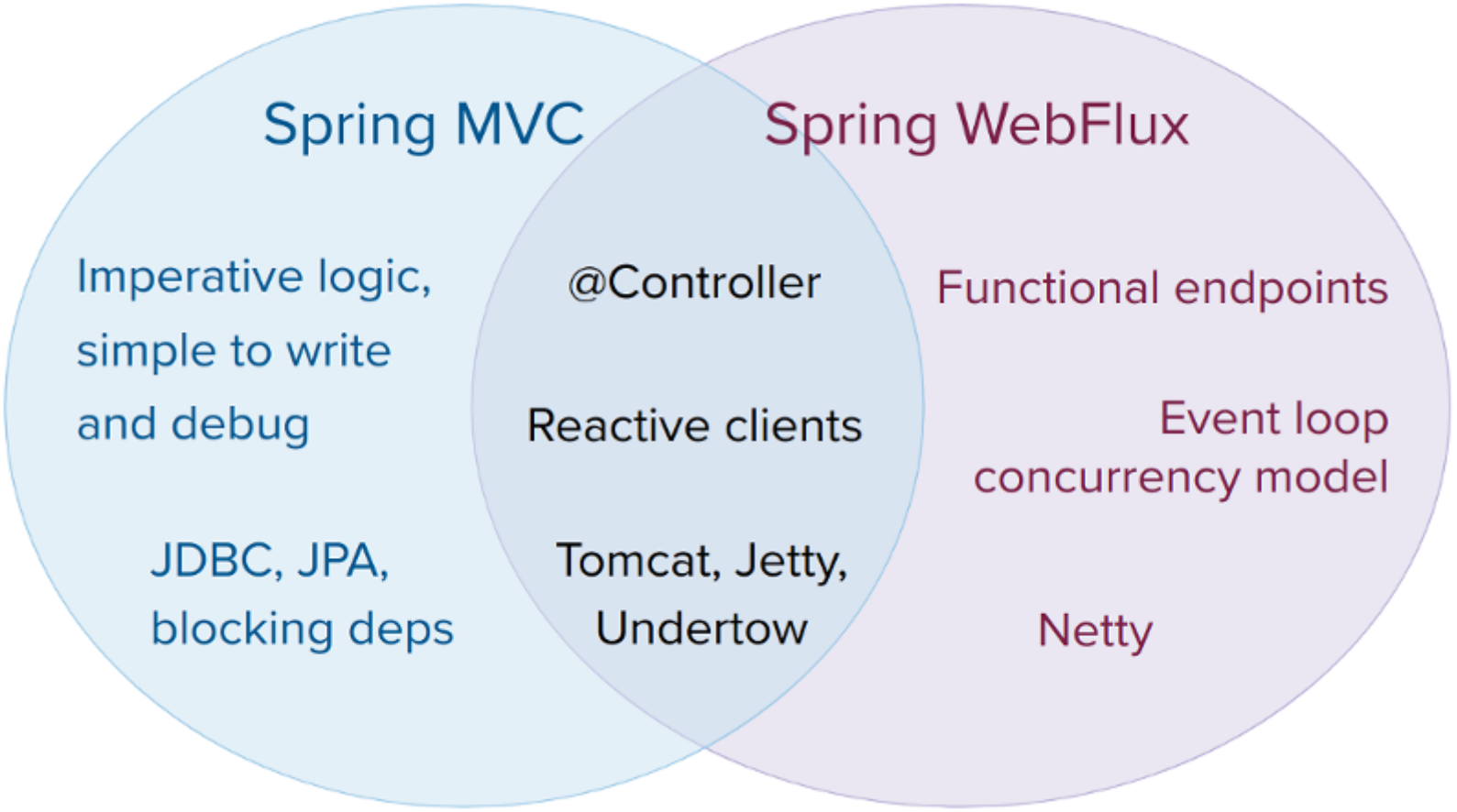
- Flow (0 - N)
- Mono (0 - 1)
- Mono<Void> (séquence vide)

#2 Spring WebFlux

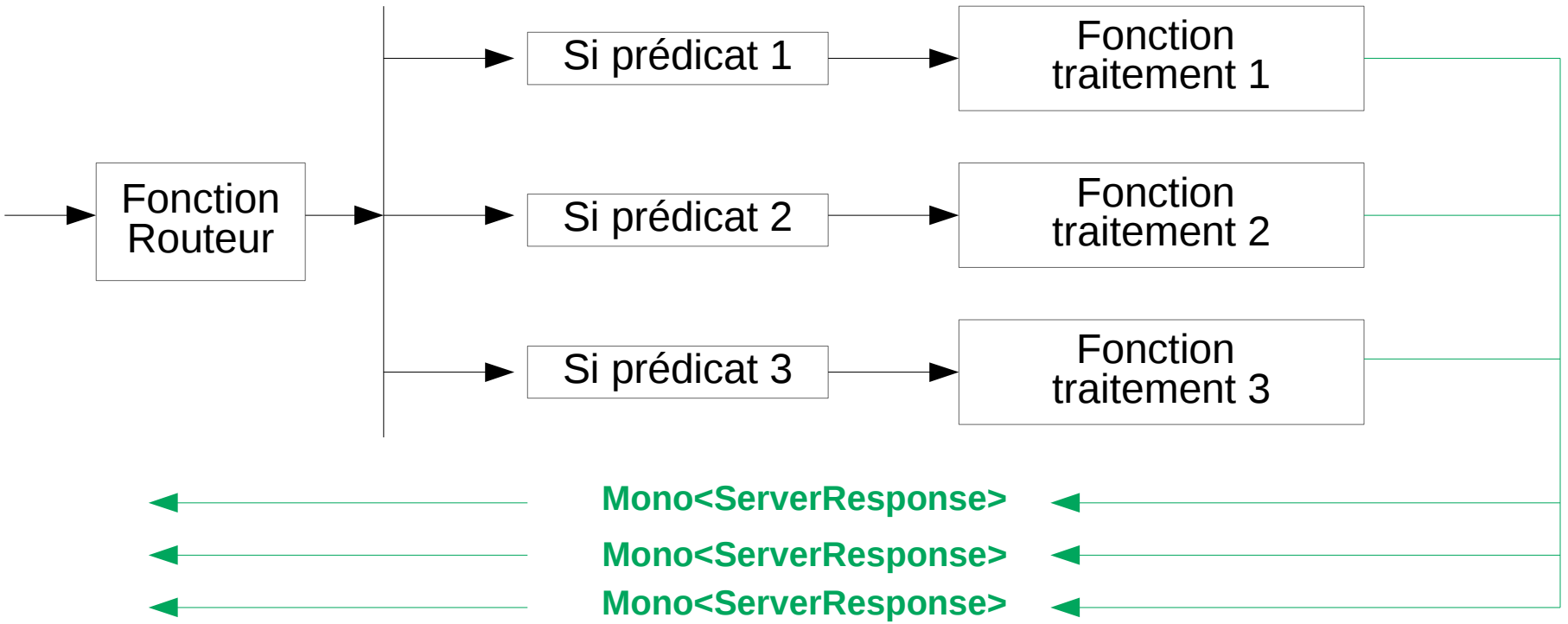
Pourquoi ?

Créer des applications fonctionnant avec **peu de Threads** Java et tenant la charge avec **peu de ressources**.

Intégrer pleinement une approche fonctionnelle.



Modèle de programmation fonctionnel



Exemple de Routeur

```
@Bean
public RouterFunction<ServerResponse> routeApp() {

    // la fonction route permet de définir les routes applicatives
    // Chaque route représente le mapping REQUETE <> Réponse
    return route (
            // HTTP GET /hello -> Réponse STATUT OK (code 200), le corps contient le texte "Bonjour".
            GET("/hello1"), request -> ServerResponse.ok().body(Mono.just("Bonjour 1"), String.class)
    )

    .andRoute (
            GET("/hello2"), request -> ServerResponse.ok().body(Mono.just("Bonjour 2"), String.class)
    )

    .andRoute (
            GET("/hello3"), request -> ServerResponse.ok().body(Mono.just("Bonjour 3"), String.class)
    )

}
```

Travaux Pratiques