

Part 1

Spring MVC 4.x

Spring 5 Web Reactive

Rossen Stoyanchev

@rstoya05

Spring MVC 4.3

Reactive programming for Java devs

Spring 5 Web Reactive

Shortcut Annotations

@RequestMapping

@GetMapping

@PostMapping

@PutMapping

@DeleteMapping

```
@RequestMapping(path="/foos/{id}", method=RequestMethod.GET)  
public Foo getFoo(@PathVariable Long id) {  
    return retrieveFoo(id);  
}
```

```
@GetMapping("/foos/{id}")
```

```
@RequestMapping(path="/foos/{id}", method=RequestMethod.GET)
```

```
public Foo getFoo(@PathVariable Long id) {
```

```
    return retrieveFoo(id);
```

```
}
```

```
@Controller
@RequestMapping("/foos")
public class FooController {

    @GetMapping("{id}")
    public Foo getFoo(@PathVariable Long id) {
        return retrieveFoo(id);
    }

    // ...

}
```

@RequestScope

@SessionScope

@ApplicationScope


```
@Component
@Scope(scopeName = "session")
public class Foo {

    // ...

}
```

```
@Component
@SessionScope
@Scope(scopeName = "session")
public class Foo {

    // ...

}
```

@RequestAttribute

@SessionAttribute

```
@PostMapping("/{id}")  
public void handle(@RequestAttribute Foo foo) {  
    // ...  
}
```

```
@PostMapping("/{id}")  
public void handle(@SessionAttribute Foo foo) {  
    // ...  
}
```

Not to be confused with:

@SessionAttribute 

```
@Controller
@SessionAttributes("foo")
public class FooController {

    @GetMapping
    public Foo getFoo(@PathVariable Long id) {
        return new Foo();
    }

    @PostMapping("/{id}")
    public void handle(@ModelAttribute Foo foo) {
        // ...
    }
}
```

@RestControllerAdvice


```
@ModelAttribute (binding=false)
```

Pre-load **Foo** in the model

```
@ModelAttribute  
public Foo addFoo(@PathVariable Long id) {  
    return retrieveFoo(id);  
}
```

Form object
with binding

```
@PostMapping("/{id}")  
public void update(@Valid FooForm form,  
    @ModelAttribute(binding=false) Foo foo) {  
  
    // ...  
}
```

Foo object without binding

HTTP OPTIONS, HEAD

automated handling

```
curl -v -X OPTIONS http://localhost:8080/foos/123
```

```
< HTTP/1.1 200 OK  
< Server: Apache-Coyote/1.1  
< Allow: GET,HEAD  
< Content-Length: 0
```

```
@GetMapping("/foos/{id}")  
public Foo getFoo(@PathVariable Long id) {  
    return retrieveFoo(id);  
}
```



```
curl --head http://localhost:8080/foos/123
```

```
HTTP/1.1 200 OK
```

```
Server: Apache-Coyote/1.1
```

```
Content-Type: application/json;charset=UTF-8
```

```
Content-Length: 155
```

```
@GetMapping("/foos/{id}")
```

```
public Foo getFoo(@PathVariable Long id) {
```

```
    return retrieveFoo(id);
```

```
}
```

ForwardedHeaderFilter

Complements existing support

for “X-Forwarded-*” in

[Mvc | Servlet] UriComponentsBuilder

RestTemplate

default URI variable values


```
Map<String, String> defaultVars = new HashMap<>(2);  
defaultVars.put("host", "api.example.com");  
defaultVars.put("port", "443");
```

```
RestTemplate template = new RestTemplate();  
template.setDefaultUriVariables(defaultVars);
```

```
Map<String, Object> vars = new HashMap<>(1);  
vars.put("id", 123L);
```

```
String url = "https://{host}:{port}/v42/customers/{id}";  
template.getForObject(url, String.class, vars);
```

Client Mock REST Tests

expected count + order of requests

```
RestTemplate restTemplate = new RestTemplate();
```

```
MockRestServiceServer server = MockRestServiceServer  
    .bindTo(restTemplate).ignoreExpectOrder().build();
```

```
server.expect(times(2), requestTo("/foo")).andRespond(withSuccess());  
server.expect(times(3), requestTo("/bar")).andRespond(withSuccess());
```

```
// ...
```

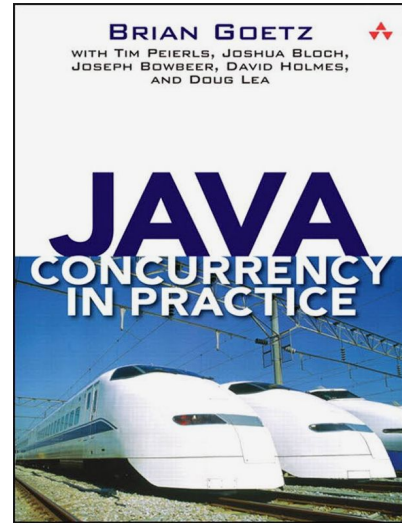
```
server.verify();
```

~~Spring MVC 4.3~~

Reactive programming for Java devs

Spring 5 Web Reactive

Long Running Shift To Concurrency

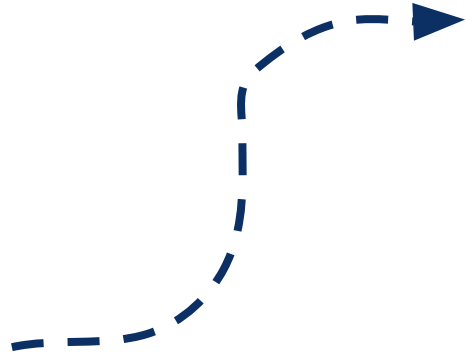


10 years ago

Self-sufficient apps

App server

Keep it simple, don't distribute



Today

Independent services

Cloud environment

Distributed apps

Impact on Programming Model

Imperative logic is not a simple path any more

Forced to deal with asynchronicity

Limits of scale

Fundamentally Asynchronous

Async and non-blocking by design

Small number of threads

Efficient scale

Design `async` API with Java Futures



Return a Value

```
public interface UserRepository {  
    User findById(String id) throws IOException;  
    ...  
    ...  
}
```

Usage

```
try {  
    User user = userRepository.findById(id);  
    // ...  
}  
catch (IOException e) {  
    // ...  
}
```

Return Value with Future

```
public interface UserRepository {  
    Future<User> findById(String id) throws IOException;  
    ...  
}
```

May be thrown in
different thread



Usage

```
try {  
    Future<User> future = userRepository.findById(id);  
    User user = future.get(); // block  
}  
catch (InterruptedException e) {  
    // ...  
}  
catch (ExecutionException e) {  
    // ...  
}
```

} Ugh

Return Value with CompletableFuture

```
public interface UserRepository {  
    CompletableFuture<User> findById(String id);  
    ...  
    ...  
}
```



Usage

```
CompletableFuture<User> future = repository.findById(id);  
future.whenComplete((user, throwable) -> {  
    // ...  
});
```

Async callback



Return Collection

```
public interface UserRepository {  
    ...  
    CompletableFuture<List<User>> findAll();  
    ...  
}
```


Return Void

```
public interface UserRepository {  
    ...  
    ...  
    CompletableFuture<Void> save(User user);  
}
```

Return Void

```
public interface UserRepository {  
    ...  
    ...  
    CompletableFuture<Void> save(User user);  
}
```

Success or Failure
callback

CompletableFuture

- ✓ Fundamentally the right idea for use in async APIs
- ✓ Allows declarative composition of async logic
- ✓ Lambdas keep it readable

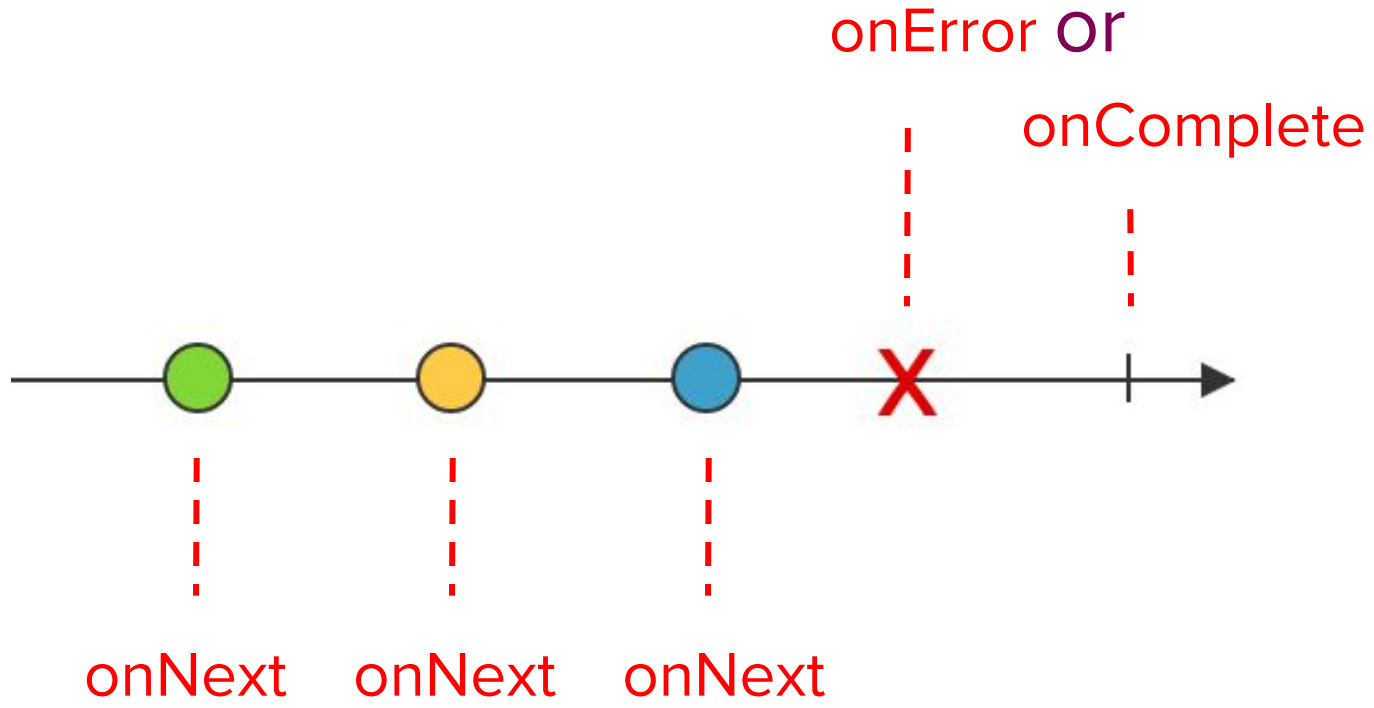
CompletableFuture

- ✗ Not ideal for Collection return values
- ✗ Nor for latency sensitive data sets
- ✗ Nor large or infinite data sets



Async return values
as a “stream” ?





Return type	Use case	Notifications
<code>void</code>	Success	<code>onComplete()</code>
<code>void</code>	Failure	<code>onError(Throwable)</code>
User	Match	<code>onNext(User)</code> , <code>onComplete()</code>
User	No match	<code>onComplete()</code>
User	Failure	<code>onError(Throwable)</code>
<code>List<User></code>	Two matches	<code>onNext(User)</code> , <code>onNext(User)</code> , <code>onComplete()</code>
<code>List<User></code>	No match	<code>onComplete()</code>
<code>List<User></code>	Failure	<code>onError(Throwable)</code>

Java 8 Stream

- ✓ Great example of the kind of API we'd like
- ✓ Declarative composition of async logic per item
- ✓ Lambdas keep it readable

Java 8 Stream

- ✗ Built for collections
- ✗ Not for active or “hot” source of data
- ✗ Latency-sensitive data sequences

No single best fluent async API [...]

Until now one **missing category** was "push" style operations on items as they become available from an active source.

Doug Lea

Reactive Streams & Java 9 [initial announcement](#)

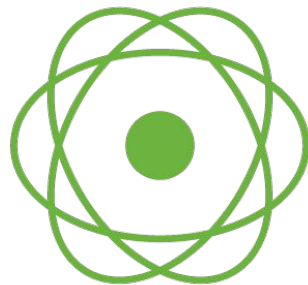
Reactive Streams

Small API, spec rules, and TCK

Publish-subscribe with **back pressure**

Interoperability across async components and libraries

Project Reactor



Reactive Streams library for the JVM

Declarative operations on items similar to Java 8 Stream

Flux and Mono reactive composable API types

Reactive Repository

```
public interface UserRepository {  
  
    Mono<User> findById(Long id);  
  
    Flux<User> findAll();  
  
    Mono<Void> save(User user);  
  
}
```

Reactive Repository In Use

```
repository.findAll()  
    .filter(user -> user.getName().matches("J.*"))  
    .map(user -> "User: " + user.getName())  
    .log()  
    .subscribe(user -> {});
```

Example output

```
onSubscribe
```

```
request (unbounded)
```

```
onNext (User: Jason)
```

```
onNext (User: Jay)
```

```
...
```

```
onComplete ()
```



By default consume without
back-pressure

Part 2

Spring MVC 4.x

Spring 5 Web Reactive

Rossen Stoyanchev

@rstoya05

~~Spring MVC 4.3~~

Reactive programming for Java devs

Spring 5 Web Reactive

Reactive Programming

A style of micro-architecture

Declarative, functional-style, composition of logic

The opposite of imperative

```
try {  
    User user = userRepository.findById(id);  
    // ...  
}  
catch (IOException e) {  
    // ...  
}
```

IMPERATIVE / BLOCKING

```
interface UserRepository {  
  
    User findById(Long id);  
  
    List<User> findAll();  
  
    void save(User user);  
  
}
```

```
repository.findAll()
    .filter(user -> user.getName().matches("J.*"))
    .map(user -> "User: " + user.getName())
    .log()
    .subscribe(user -> {});
```

FUNCTIONAL /
NEUTRAL TO ASYNCHRONICITY

```
interface UserRepository {

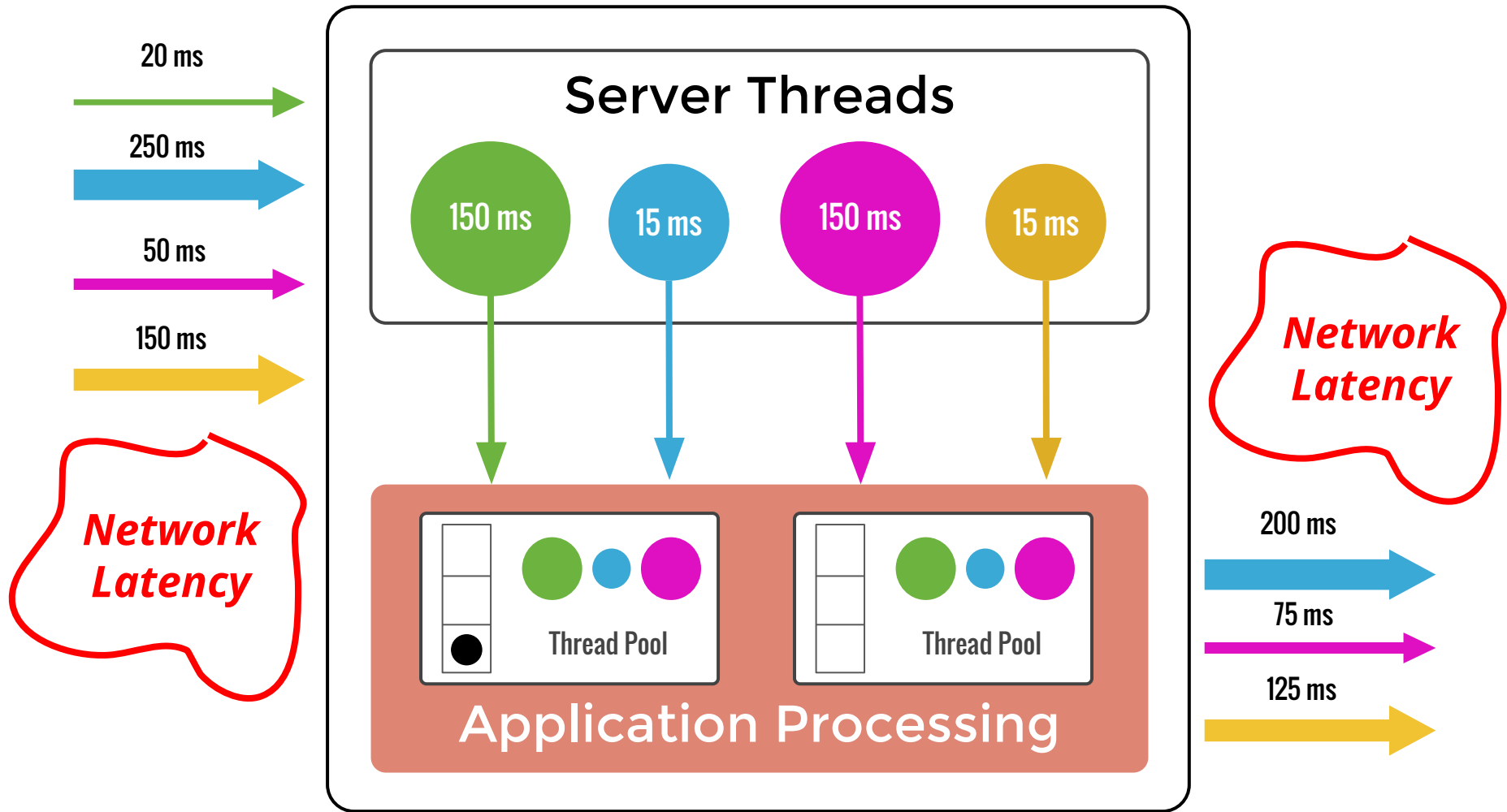
    Mono<User> findById(Long id);

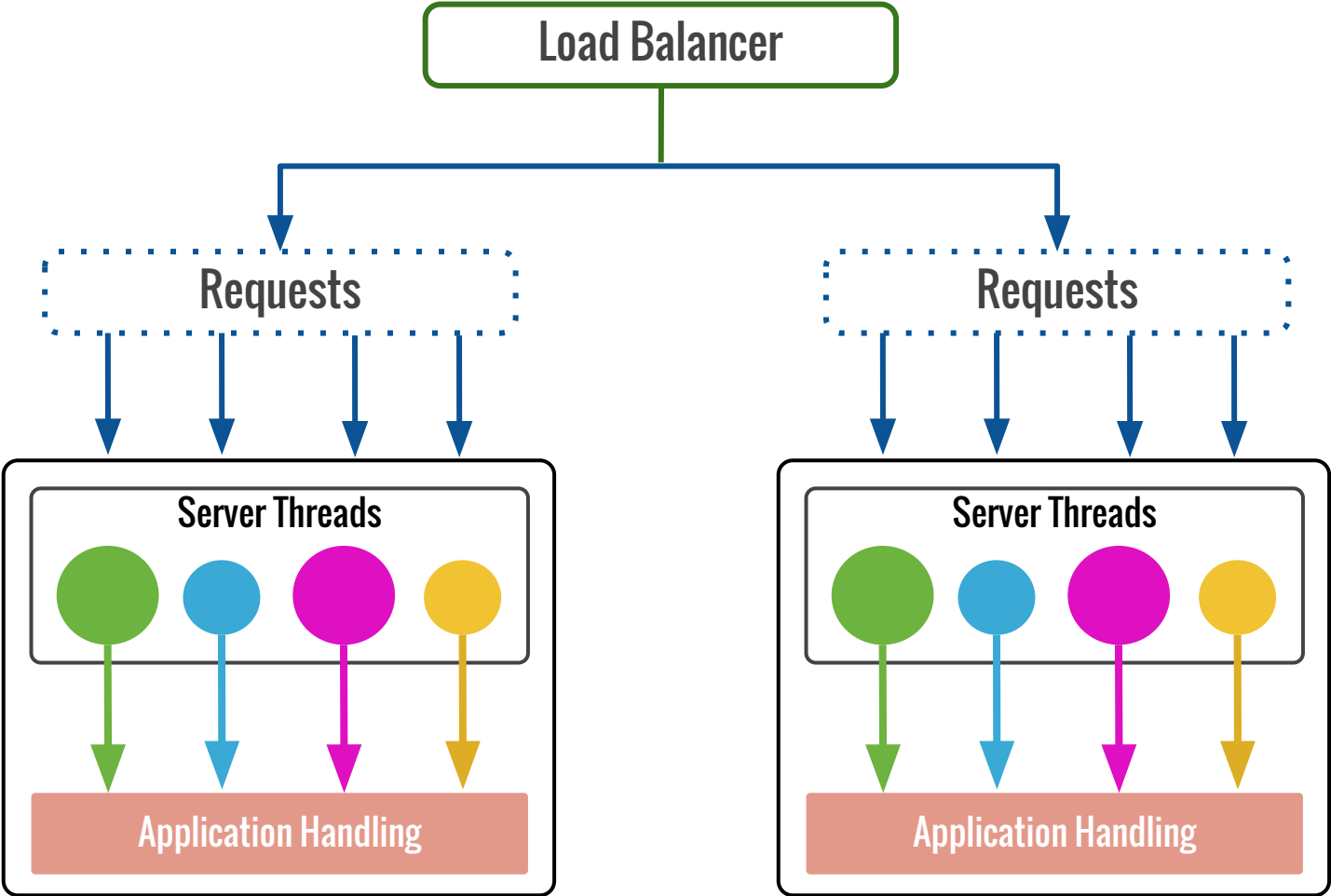
    Flux<User> findAll();

    Mono<Void> save(User user);

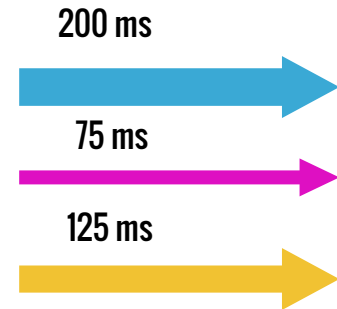
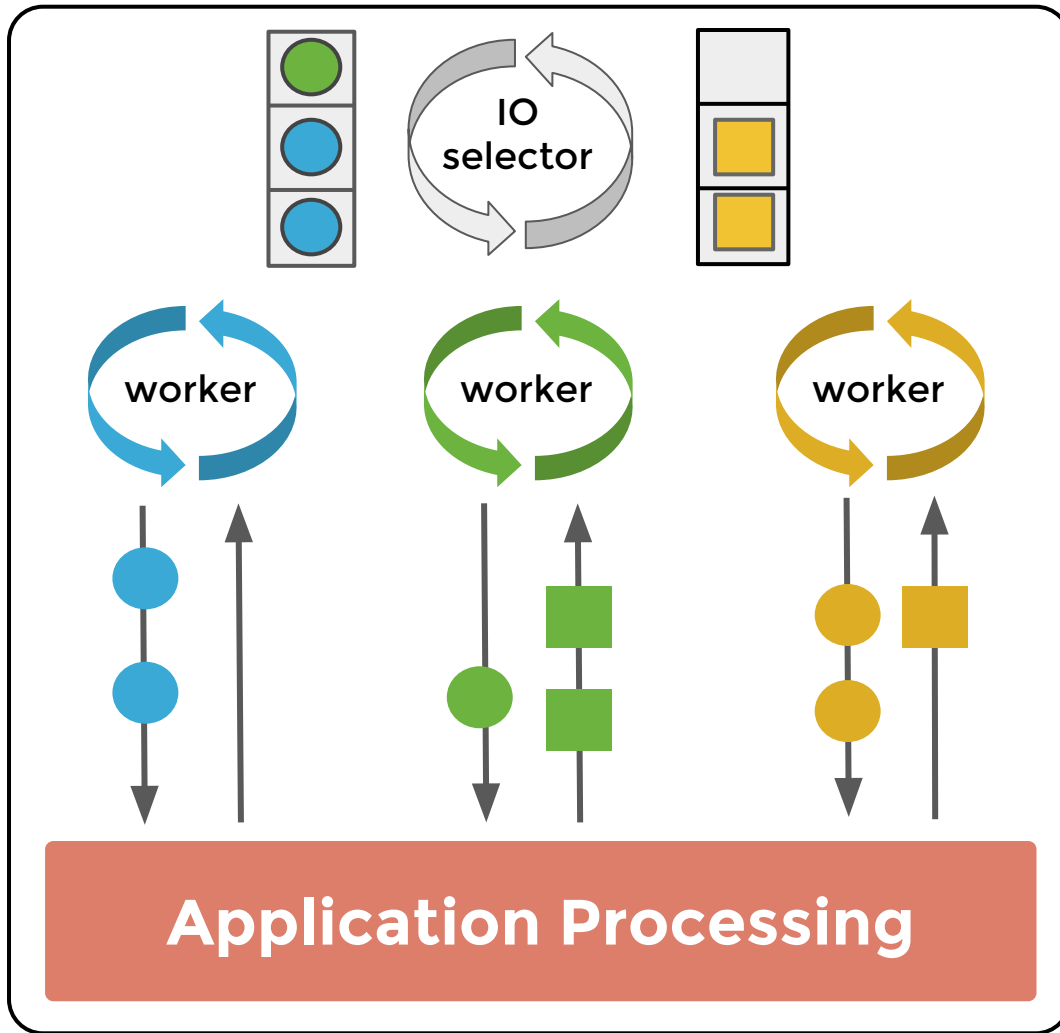
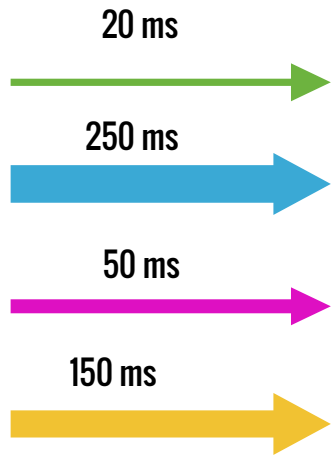
}
```

Thread Pool Style Processing





Event Loop Style Processing



Non-blocking Application

```
pan.addWater(() ->
  range.lowerHeat(() ->
    pan.addBrownRice(() ->
      pan.setTimeout(() -> {
        range.turnOff();
        // ready...
      }, Duration.ofMinutes(40)))));
```



Rise above the callbacks

ReactiveX

DeclarativeFunctional, programming on
Observable streams



Reactive Streams Specification

Async stream processing
with “reactive” back-pressure



Lightbend

Pivotal™



redhat.

KAazing >K®

Reactive Streams

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

```
public interface Subscriber<T> {  
    void onSubscribe(Subscription sub);  
    void onNext(T item);  
    void onError(Throwable ex);  
    void onComplete();  
}
```

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



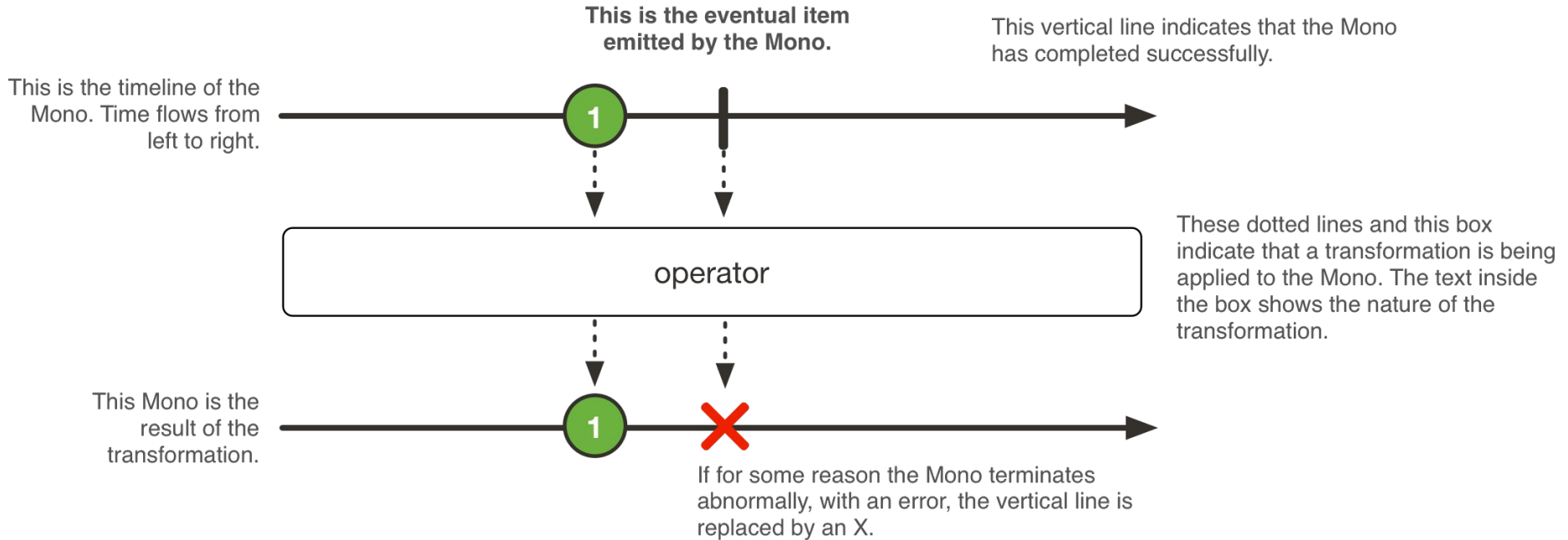
Backpressure

Project Reactor

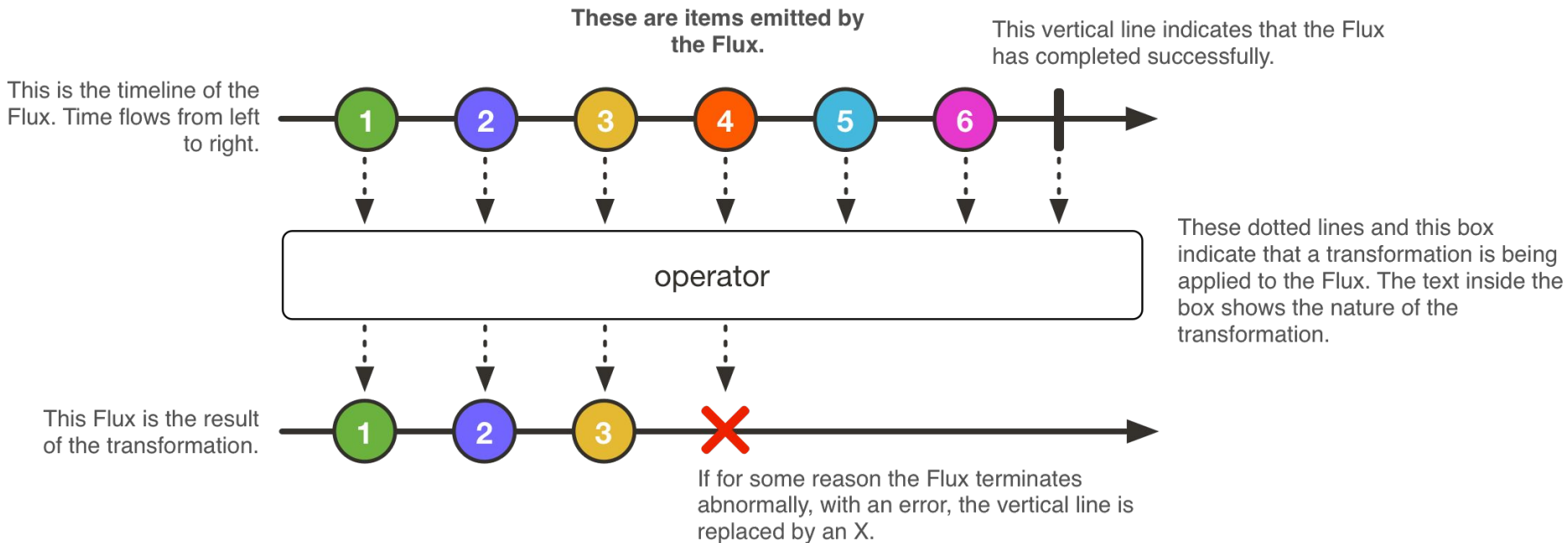
Reactive Streams + ReactiveX



Reactor Mono



Reactor Flux



Reactive Repository

```
public interface UserRepository {  
    Mono<User> findById(Long id);  
    Flux<User> findAll();  
    Mono<Void> save(User user);  
}
```

Reactive Repository in Use

```
repository.findAll()  
    .filter(user -> user.getName().matches("J.*"))  
    .map(user -> "User: " + user.getName())  
    .log()  
    .subscribe(user -> {});
```

 Subscriber triggers flow of data

Example output

```
onSubscribe
```

```
request (unbounded)
```

```
onNext (User: Jason)
```

```
onNext (User: Jay)
```

```
...
```

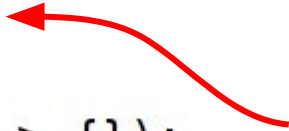
```
onComplete ()
```



By default consume without
back-pressure

Consume Two Items at a Time

```
repository.findAll()  
    .filter(user -> user.getName().matches("J.*"))  
    .map(user -> "User: " + user.getName())  
    .useCapacity(2) ←  
    .log()  
    .subscribe(user -> {});
```



Example output

onSubscribe

request (2)

onNext (User: Jason)

onNext (User: Jay)

request (2)

onNext (User: Joe)

onNext (User: John)

...



Producer complies with
back-pressure

~~Spring MVC 4.3~~

~~Reactive programming for Java devs~~

Spring 5 Web Reactive

Spring Data Reactive



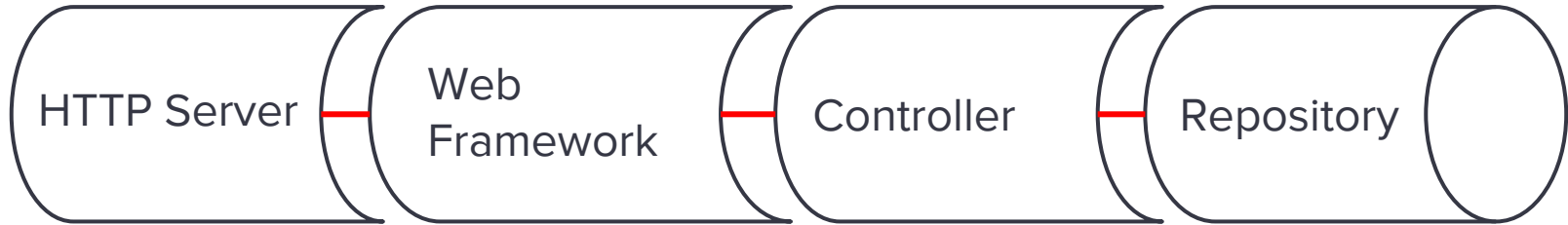
```
interface PersonRepository extends ReactiveCrudRepository<Person, ObjectId> {  
    Mono<Person> findByFirstnameAndLastname(String firstname, String lastname);  
    Flux<Person> findByLastname(String lastname);  
}
```


Spring Data Reactive



```
interface PersonRepository extends ReactiveCrudRepository<Person, ObjectId> {  
    Mono<Person> findByFirstnameAndLastname(String firstname, String lastname);  
    Flux<Person> findByLastname(String lastname);  
}
```

Reactive Processing Pipeline



End-to-end back-pressure

Reactive Web Controller

```
@Controller
public class UserController {

    private final UserRepository userRepository;

    @Autowired
    public UserController(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

}

}
```

Reactive Web Controller

Spring 5

```
@GetMapping("/users/{id}")
public Mono<User> getUser(@PathVariable Long id) {
    return this.userRepository.findById(id);
}
```

```
@GetMapping("/users")
public Flux<User> getUsers() {
    return this.userRepository.findAll();
}
```

```
@PostMapping("/users")
public Mono<Void> addUser(@RequestBody User user) {
    return this.userRepository.save(user);
}
```

Reactive Web Controller / RxJava

Spring 5

```
@GetMapping("/users/{id}")
public Single<User> getUser(@PathVariable Long id) {
    return this.userRepository.findById(id);
}
```

```
@GetMapping("/users")
public Observable<User> getUsers() {
    return this.userRepository.findAll();
}
```

```
@PostMapping("/users")
public Completable addUser(@RequestBody User user) {
    return this.userRepository.save(user);
}
```

@Controller, @RequestMapping

Router Functions

Spring Web MVC

Spring Web Reactive

Servlet API

Reactive HTTP

Servlet Container

Tomcat, Jetty, Servlet 3.1+,
Netty, Undertow

Reactive HTTP Adaptation

```
public interface ReactiveHttpInputMessage extends HttpResponseMessage {  
    Flux<DataBuffer> getBody();  
}
```

```
public interface ReactiveHttpOutputMessage extends HttpResponseMessage {  
    Mono<Void> writeWith(Publisher<DataBuffer> body);  
}
```

Reactive HTTP Server Adaptation

Tomcat, Jetty, Servlet 3.1+ containers

Netty (Reactor Netty, RxNetty)

Undertow

Reactive Encoder and Decoder

Serialization to/from **Flux<T>** and **Flux<DataBuffer>**

JSON (Jackson) and XML (JAXB / Aalto)

Server-Sent Events

Resource with zero-copy file transfer

Spring Web Reactive

Along with Spring Web MVC

Operates on reactive HTTP request / response

Shared algorithms and mechanisms

Non-Blocking HTTP GET

```
@GetMapping("/users/{id}")  
public User getUser(@PathVariable Long id) {  
    return new User(id);  
}
```

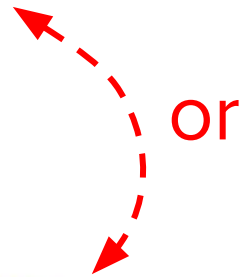
Synchronous, non-blocking method

```
@GetMapping("/users/{id}")  
public Mono<User> getUser(@PathVariable Long id) {  
    return this.userRepository.findById(id);  
}
```

Async, non-blocking method

Non-Blocking HTTP POST

```
@PostMapping("/users")  
public Mono<Void> addUser(@RequestBody User user) {  
    return this.userRepository.save(user);  
}
```



```
@PostMapping("/users")  
public Mono<Void> addUser(@RequestBody Mono<User> mono) {  
    return mono.then(this.userRepository::save);  
}
```

Server-Sent Events

```
@GetMapping("/users/events")
public Flux<SseEvent> getUserEvents() {

    return initUserEventStream()
        .map(user ->
            new SseEvent(user, APPLICATION_JSON));
}
```

Zero-Copy File Transfer

```
@GetMapping("/resource/{id}")  
public Resource handle(@PathVariable Long id) {  
    // ...  
    return getResource(id);  
}
```

WebClient

```
String url = "http://localhost:8080/accounts";  
Flux<User> body = getUsers();
```

```
Mono<ResponseEntity<Void>> response = webClient  
    .perform(post(url).body(body).contentType(APPLICATION_JSON))  
    .extract(response(Void.class));
```

WebClient / RxJava

```
String url = "http://localhost:8080/accounts";  
Observable<User> body = getUsers();  
  
Single<ResponseEntity<Void>> response = webClient  
    .perform(post(url).body(body).contentType(APPLICATION_JSON))  
    .extract(response(Void.class));
```


Non-blocking WebClient Scatter/Gather

```
@GetMapping("/accounts/{id}/alerts")
public Flux<Alert> getAccountAlerts(@PathVariable Long id) {

    return this.repository.getAccount(id)
        .flatMap(account ->

            this.webClient
                .perform(get("/alerts/{key}", account.getKey()))
                .extract(bodyStream(Alert.class)));

}
```

Functional Web Framework

No annotations, minimal, and transparent

Same reactive foundation

["Spring 5: Functional Web Framework"](#) blog post

Functional-style Web Routing

```
RouterFunction<?> route =  
    route(GET("/users/{id}"), request -> {  
        Mono<User> user = Mono.justOrEmpty(request.pathVariable("id"))  
            .map(Long::valueOf).then(userRepository::findById);  
        return Response.ok().body(fromPublisher(user, User.class));  
    })  
    .and(route(GET("/users"), request -> {  
        Flux<User> users = userRepository.findAll();  
        return Response.ok().body(fromPublisher(users, User.class));  
    }));
```

```
Handler handler = RouterFunctions.toHandler(route);  
HttpServer server = HttpServer.create("localhost", 8080);  
server.startAndAwait(new ReactorHandlerAdapter(handler));
```

<http://start.spring.io>

SPRING INITIALIZR bootstrap your application now

Generate a with Spring Boot

Project Metadata

Artifact coordinates

Group

com.example

Artifact

demo

Dependencies

Add Spring Boot Starters and dependencies to your application

Search for dependencies

reactive Web

Reactive Web

Reactive web development with Tomcat and Spring Reactive (experimental)

Actuator

Production ready features to help you monitor and manage your application

Generate Project

Don't know what to look for? Want more options? [Switch to the full version.](#)

Spring Boot “Web Reactive” Starter

Helpful instructions:

<https://github.com/bclozel/spring-boot-web-reactive>

Manual Bootstrap

Only a few lines of code:

[Spring Framework 5.0 snapshot reference](#)