

# Going with Data.

Christoph Strobl  
Pivotal Software Inc.  
[@stroblchristoph](#)

# JAVADAY<sub>2017</sub>

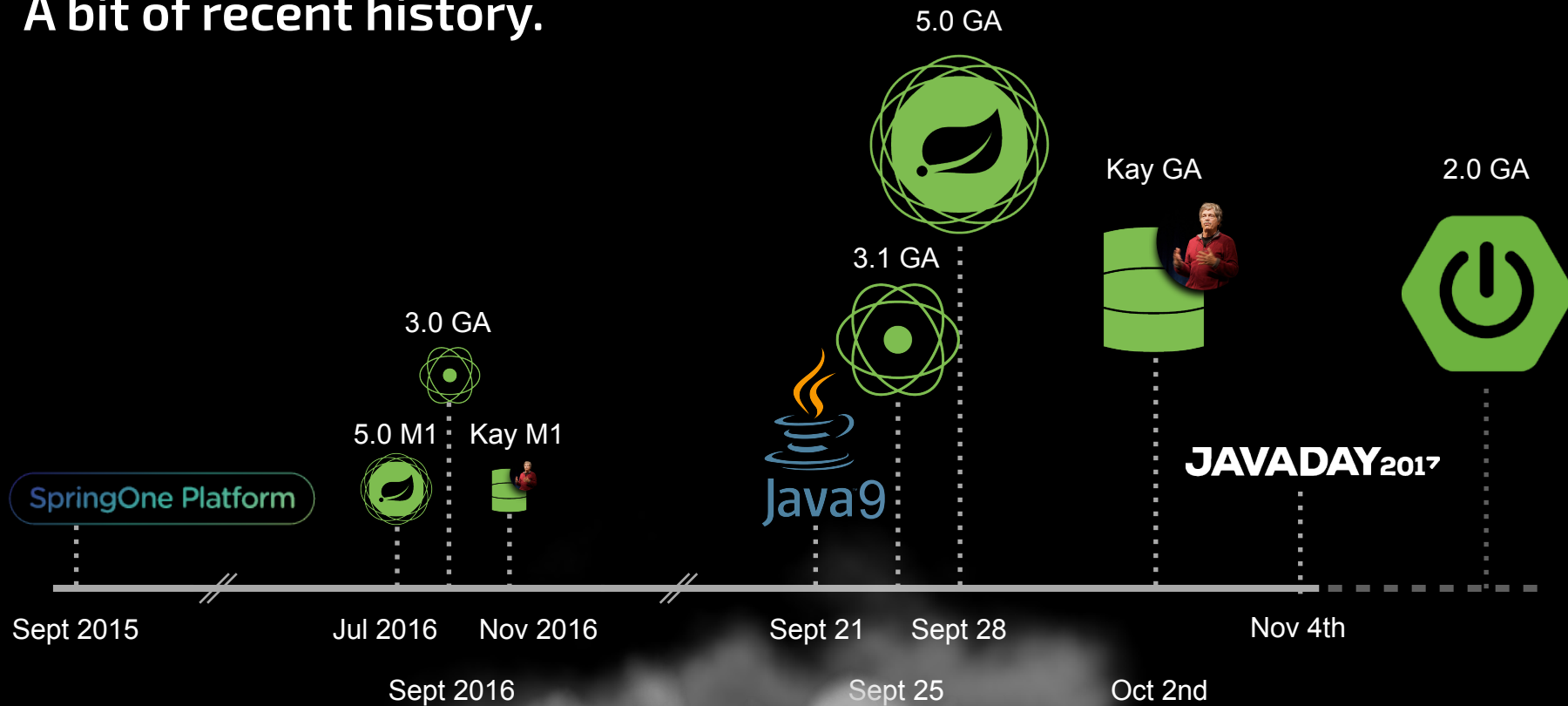
*In a nutshell, reactive programming is about  
non-blocking, event-driven applications  
that scale with a small number of threads  
with back pressure as a key ingredient  
that aims to ensure producers to not overwhelm consumers.*

*(Rossen Stoyanchev)*

# The next 60 Minutes

- A bit of recent history.
- Project Reactor / Spring Data Kay / Spring Framework 5.
- Some Code.

# A bit of recent history.



The background of the slide is black, featuring a complex pattern of thin, intersecting lines in various shades of green and orange. These lines create a sense of depth and movement, with some lines appearing to recede into the distance while others cross the foreground. The overall effect is reminiscent of a digital or architectural blueprint.

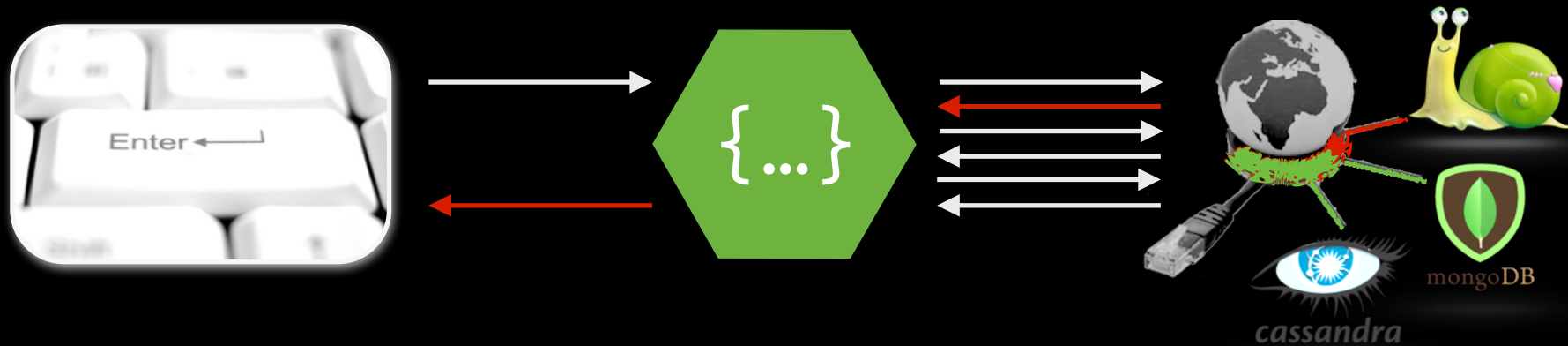
# Imperative Applications

```
@RestController
static class PersonController {

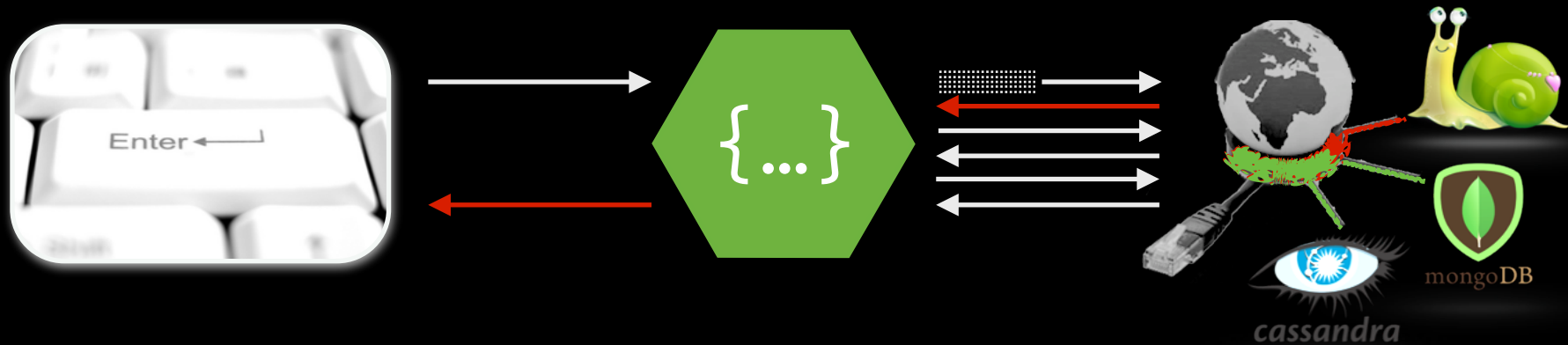
    PersonRepository repository

    @GetMapping("/")
    List<Person> listPersons(String name) {
        return repository.findAllByName(name);
    }
}
```

# Total Control

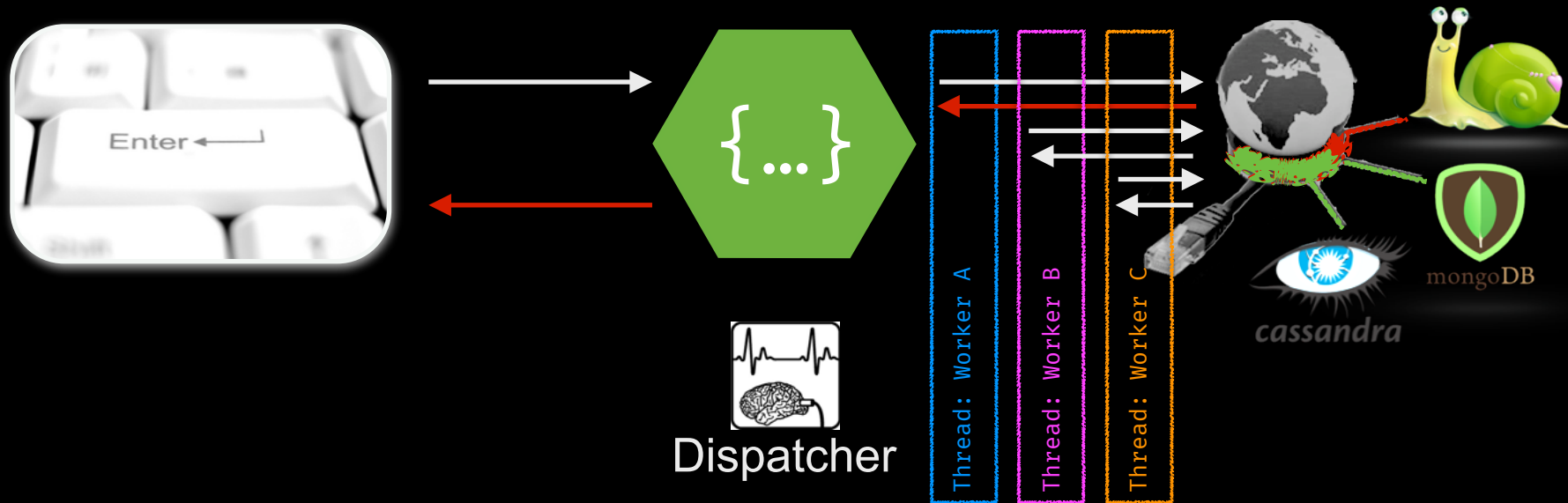


# Batching

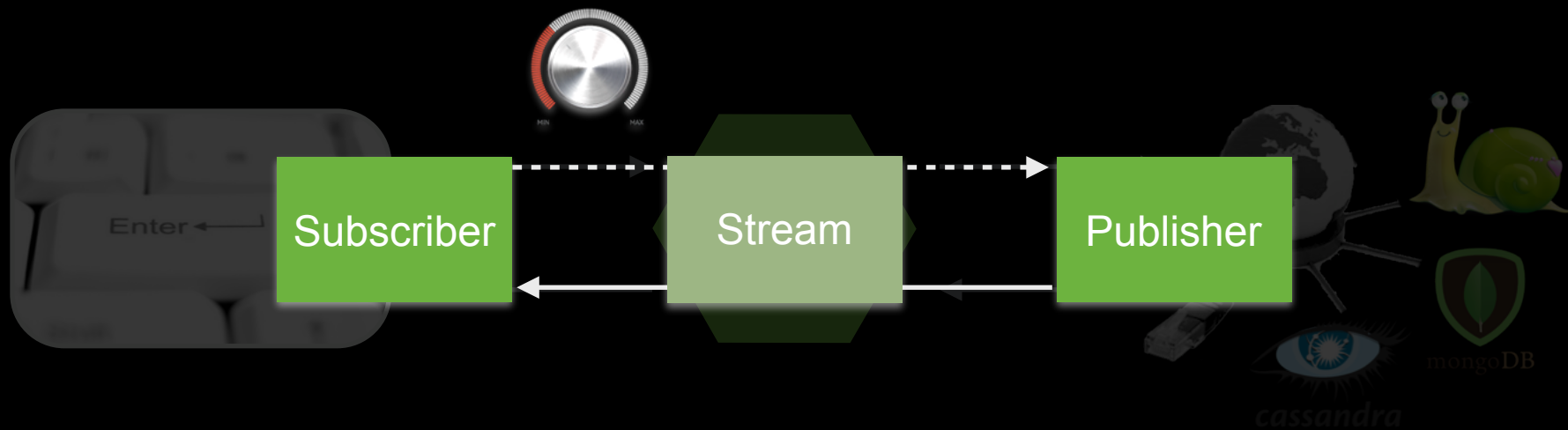




# Async



# Reactive



# Remember

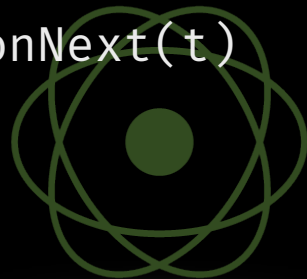
`Iterator.next()`  
`Future.get()`



vs



`Subscriber.onNext(t)`



# Reactor 3.1



# Reactive Streams

`Publisher<T>`

Provider of a potentially unbounded number of sequenced elements.

`Subscriber<T>`

`Subscription`

One-to-one lifecycle of a Subscriber subscribing to a Publisher.

`Processor<T, R>`

A processing stage - actually both a Subscriber and a Publisher.



`Flux<T> (0..n)`



`Mono<T> (0..1)`

`Flux<Person> findAllByLastname(String lastname)`

`Mono<Person> findByEmail(String email)`

**#jürgenized**

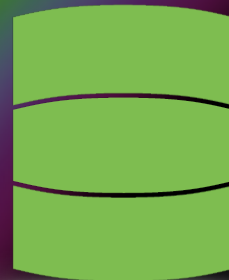
# Reactive Applications with Spring




Project Reactor  
3.1



Spring Framework  
5



Spring Data  
Kay



Apply familiar concepts  
to a new way of expressing functionality.  
With minimal fluff and surprise!

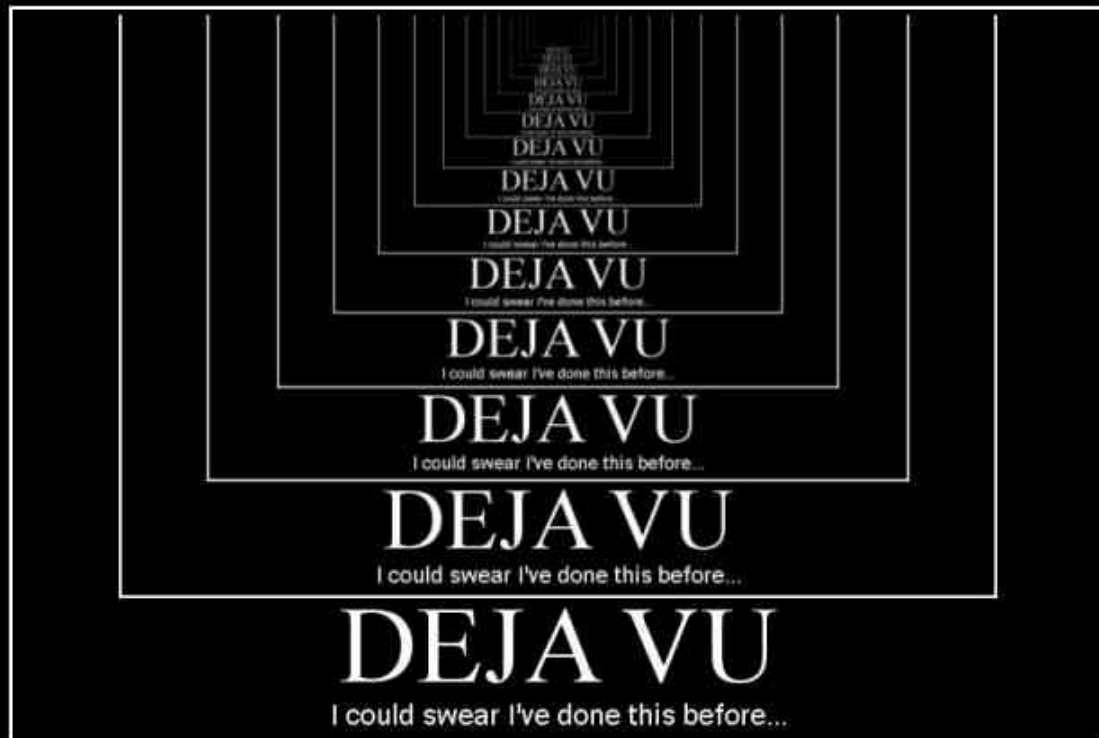


# Spring WebFlux

```
@RestController
static class PersonController {

    PersonRepository repository

    @GetMapping("/")
    Flux<Person> fluxPersons(String name) {
        return repository.findAllByName(name);
    }
}
```



DEJA VU  
I could swear I've done this before...

# Spring WebFlux

```
@RestController
static class PersonController {

    PersonRepository repository

    @GetMapping("/")
    Flux<Person> fluxPersons(String name) {
        return repository.findAllByName(name);
    }
}
```

# Spring Data Reactive

```
interface Repo extends ReactiveCrudRepository<Person, String> {  
    Flux<Person> findAllByLastname(String lastname);  
    Flux<Person> findAllByLastname(Mono<String> lastname);  
    @Query("{lastname : ?0}");  
    Flux<Person> customQuery(String lastname);  
    Flux<Person> findAllByLastname(String ln, Pageable page);  
}
```

# Reactive Repository

```
<S extends T> Mono<S> save(S entity);  
<S extends T> Mono<S> save(Mono<S> entity);  
<S extends T> Flux<S> saveAll(Iterable<S> entities);  
<S extends T> Flux<S> saveAll(Publisher<S> entities);  
Mono<Boolean> existsById(Mono<ID> id);  
Flux<T> findAll();  
Mono<Long> count();  
Mono<Void> deleteById(ID id);
```

# Reactive Template (MongoDB)

```
/**
 * Map the results of an ad-hoc query on the collection for the entity class to a
 * single instance of an object of the
 * specified type.
 *
 * @param query the query class that specifies the criteria
 * @param entityClass the parametrized type of the returned {@link Mono}.
 * @return the converted object
 */
<T> Mono<T> findOne(Query query, Class<T> entityClass);

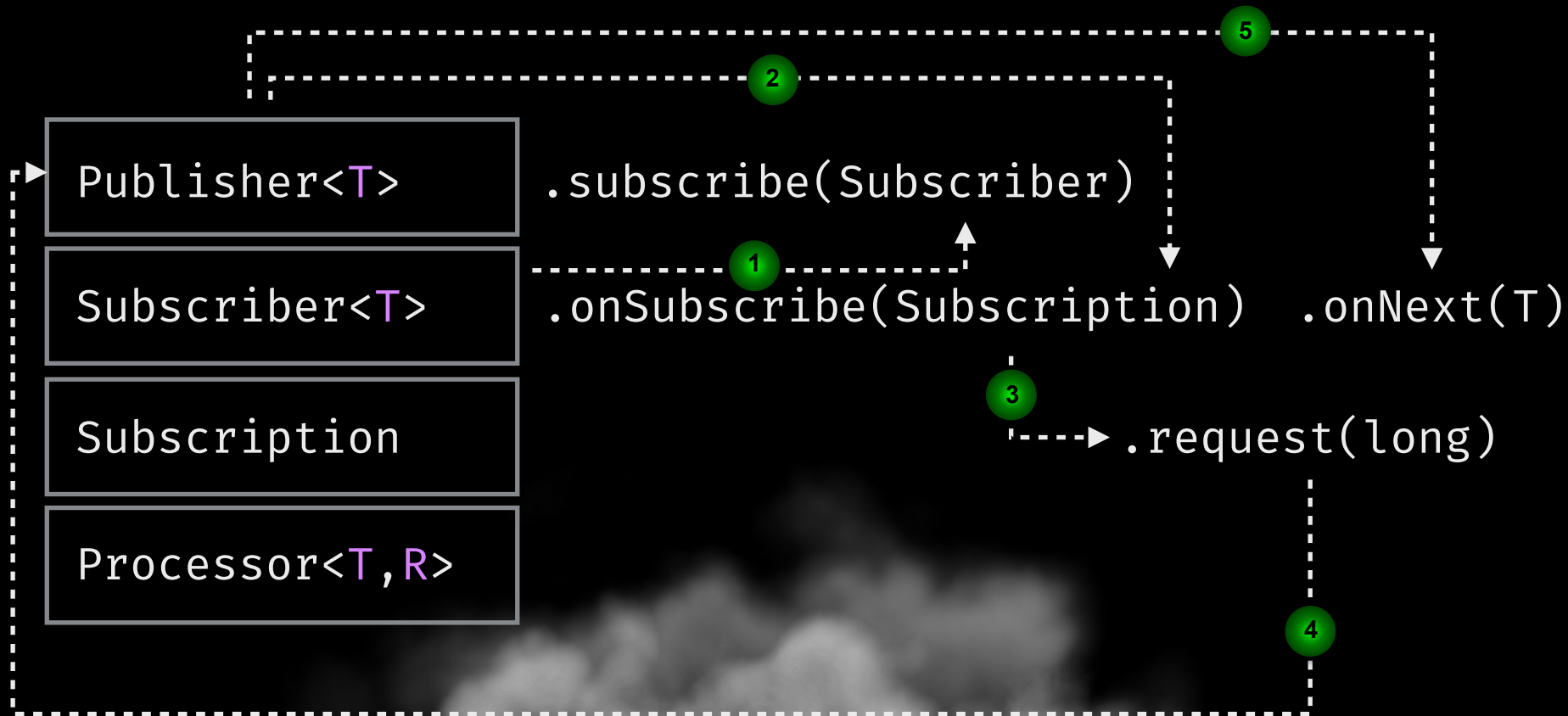
/**
 * Query for a {@link Flux} of objects of type T from the specified collection.
 *
 * @param entityClass the parametrized type of the returned {@link Flux}.
 * @param collectionName name of the collection to retrieve the objects from
 * @return the converted collection
 */
<T> Flux<T> findAll(Class<T> entityClass, String collectionName);
```

# Reactive Streams MongoDB Java Driver

```
/**  
 * Finds all documents in the collection.  
 *  
 * @param filter the query filter  
 * @return the fluent find interface  
 */  
FindPublisher<TDocument> find(Bson filter);
```

```
/**  
 * Counts the number of documents in the collection according to the given options.  
 *  
 * @param filter the query filter  
 * @return a publisher with a single element indicating the number of documents  
 */  
Publisher<Long> count(Bson filter);
```

# Reactive Streams





# It's Demo Time

<https://github.com/christophstrobl/going-reactive-with-spring-data>



*Reactive is about efficient resource usage.  
Even if backed with familiar concepts and framework support,  
it is no free lunch.  
The price is complexity.*

# More Spring / Reactive at **JAVADAY**2017

15:00 Spring Boot 2.0 Web Applications (*Stéphane Nicoll*)

16:10 The API Gateway is dead! Long live the API Gateway! (*Spencer Gibb*)

17:30 Reactive Mythbusters With Reactor (*Simon Basle*)

18:40 Refactor to Reactive with Spring 5 and Project Reactor (*Oleh Dokuka*)

Today

12:30 Spring Cloud Stream - a new rick and Morty adventure (*Dieter Hubau*)

12:30 Advanced Client Load Balancing with Spring Cloud (*Aleksandr Tarasov*)

13:50 A Day in a Life of a Traveling Open Source Developer (*Spencer Gibb*)

14:30 Boot yourself, Spring is coming (*Evgeniy Borisov, Kirill Tolkachev*)

17:00 Data Microservices with Spring Cloud (*Orkhan Gasimov*)

Tomorrow

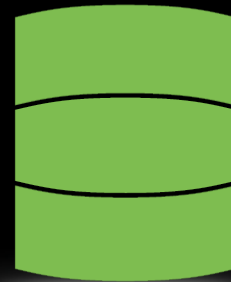
# Questions ?



Project Reactor  
3.1



Spring Framework  
5



Spring Data  
Kay