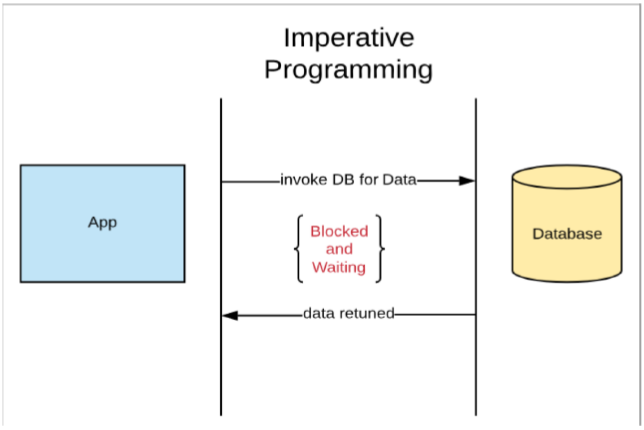
Imperative Programming:

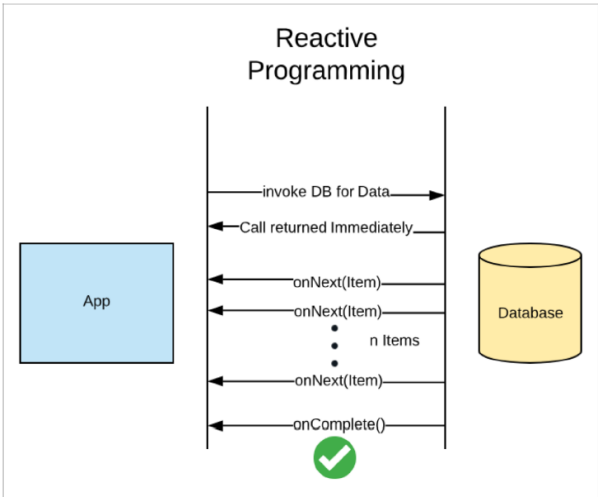
```
List<Item> items = itemRepository.getAllItems();
```



- Synchronous and blocking communication Model.

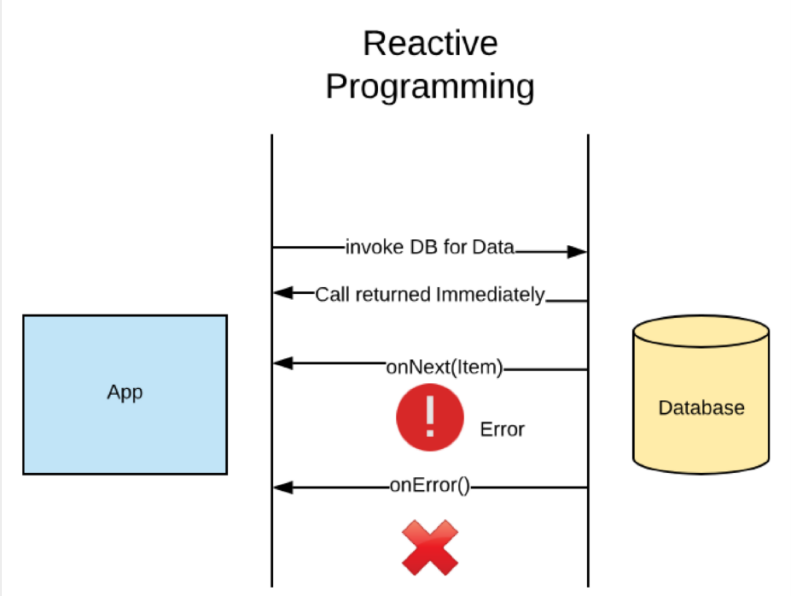
Data flow as an Event Driven stream

```
List<Item> items = itemRepository.getAllItems();
```



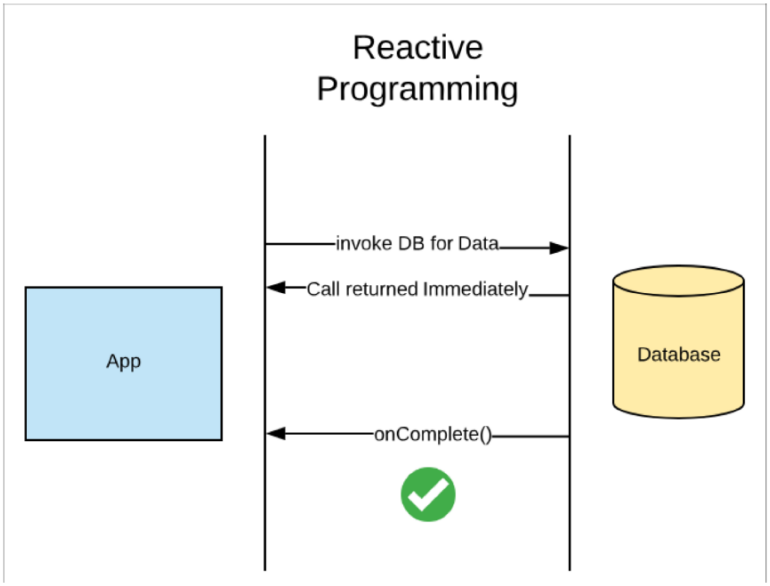
Data flow as an Event Driven stream

- Error Flow

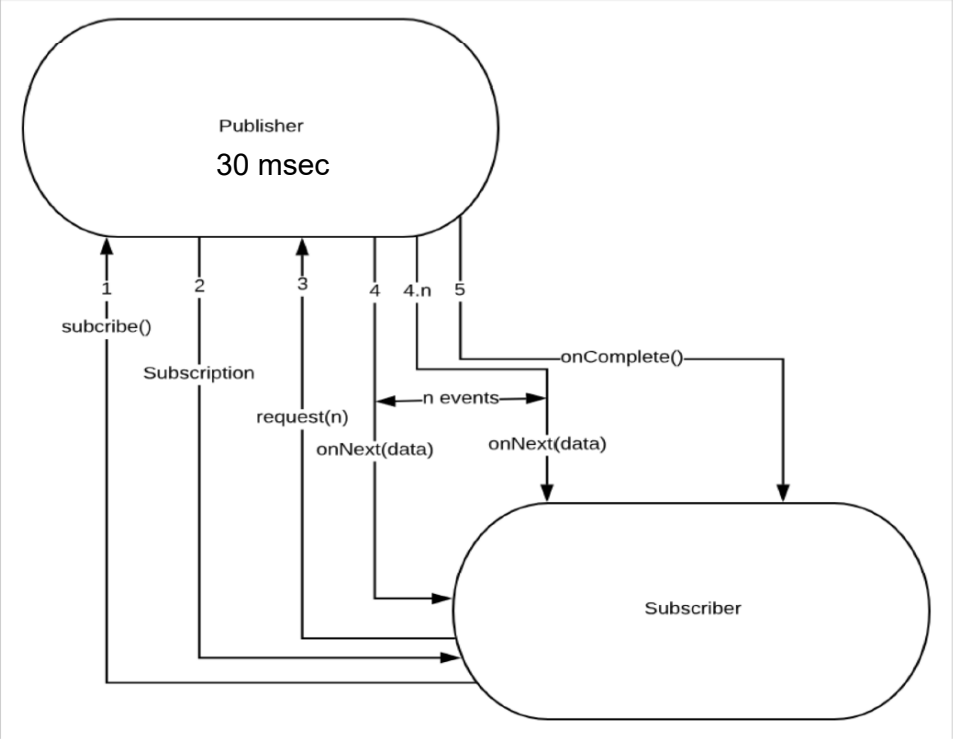


Data flow as an Event Driven stream

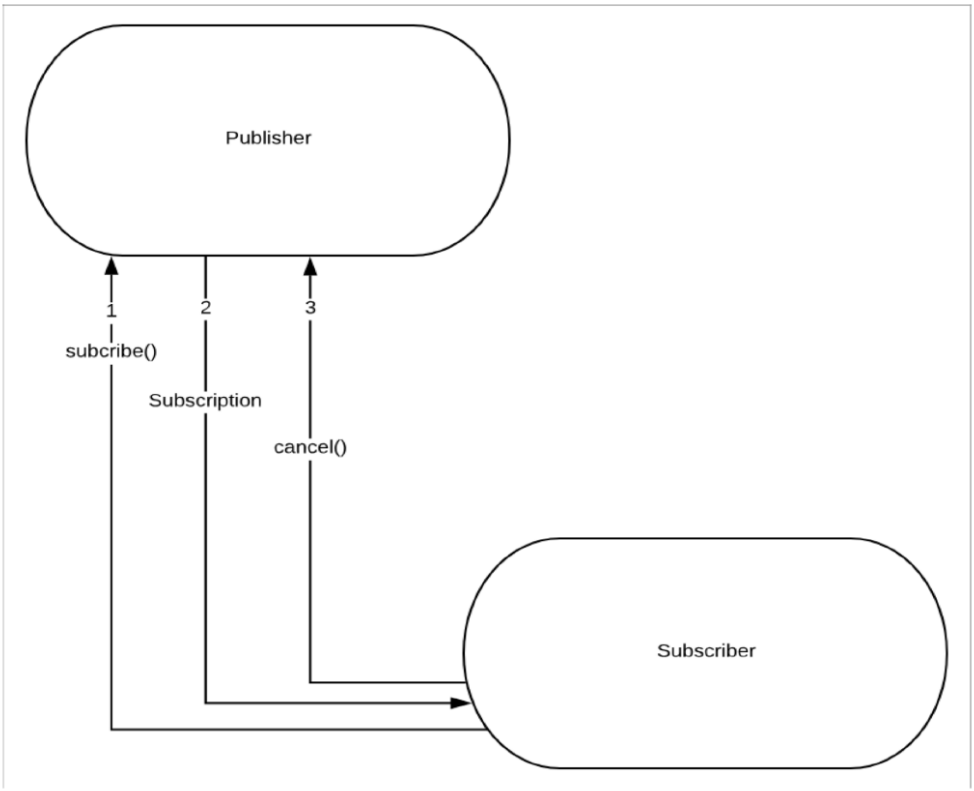
- No Data



Publisher/Subscriber Event Flow



Publisher/Subscriber Event Flow



Project Reactor : REcommended Library for Spring Boot

Modules :

reactor-core : contain implementations of Reactive Stream Specifications (interface)

reactor-test : api to create unit tests for REactive Streams

reactor-netty : non-blocking http server

Reactor-core provides implementation of interface in form of :

Reactive types/Streams

Flux : 0-N	Publisher Stream (implementation of publisher)
Mono : 0-1	

Traditional (Imperative)

DAO implementation

```
List<User> users = userDao.getUsers();
List<String> names = new ArrayList<String>();
for(int i = 0; i<user.size(); i++)
    names.add(users.get(i).getName());
```

Functional (Java Stream: declarative)

```
List<String> names = userDao.getUsers().stream()
                        .map(user -> user.getName())
                        .collect(Collectors.toList());
```

if db is busy (lot of data)
take some time:
our thread will be blocked

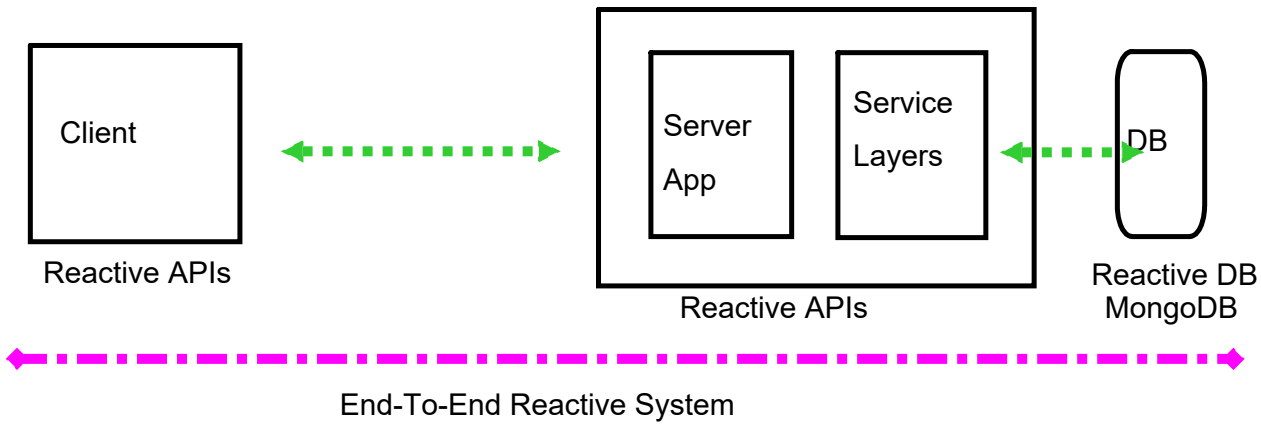
Reactive Stream

Reactive DAO implementation

```
Flux<String> names = reactiveUserDao.getUsers()
        .map(user->user.getName());
```

Non-blocking approach

To take complete advantage of Reactive Programming, a end-to-end Reactive system must be setup



Webflux web framework for Spring Boot (uses the project reactor)

reacter-core api : unit test cases

Creating a Flux Stream/ Reactive Stream/ Reactive Publisher

Flux.just(<data>) Mono.just(<single data>)

java 8 Stream are not - reusable
Reactive Streams (Flux) is reusable

By default when we subscribe to Publisher (Flux/Mono),
other activities :
returning subscription, sending request(unbounded) : behind the scene
publisher will start streaming data using onNext(data) event, for each data (auto)
==> Directly access the data as a stream

Factory Methods : for creating Flux and Mono

Just like Java 8 Stream a series of activities can be associated with Reactive Stream

Filter

Map/flatMap

Combine Reactive Streams...

Restart/Retry generating reactive stream after error : multiple retry...

BackOff Retry if still OnError() is propagated : IllegalStateException

BackPressure

TO have an absolute control over backpressure, we need to provide an implementation of abstract class: BaseSubscriber

Traditional Client-server : Push based model (Server push data to client)

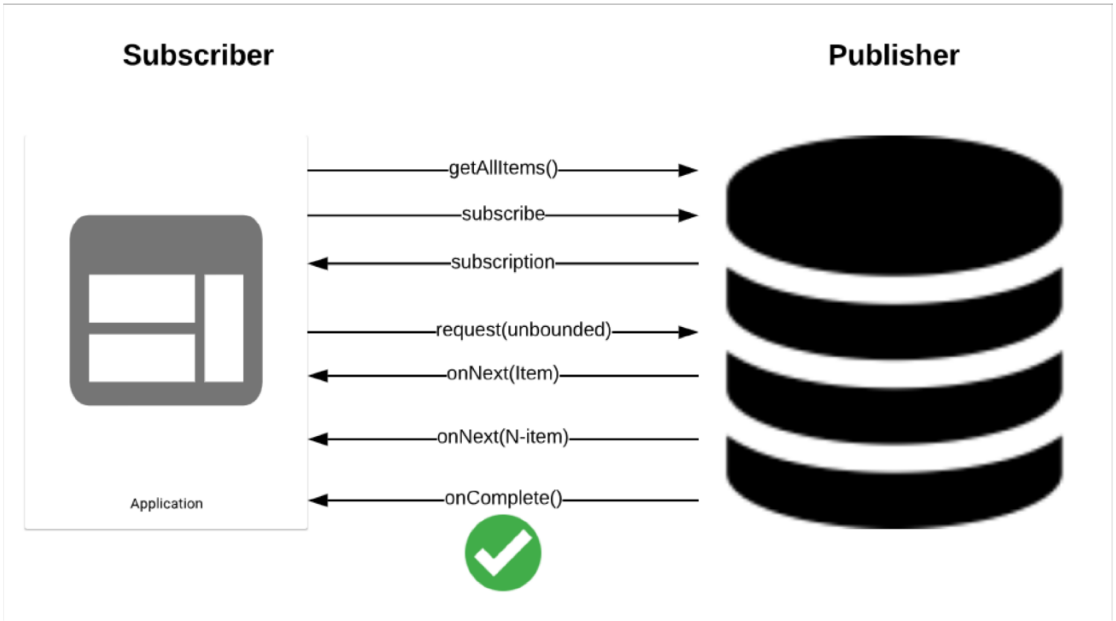
Push/Pull model : Client-Server both are having equal control over data flow

Flux (publisher)variant :

cold (default)

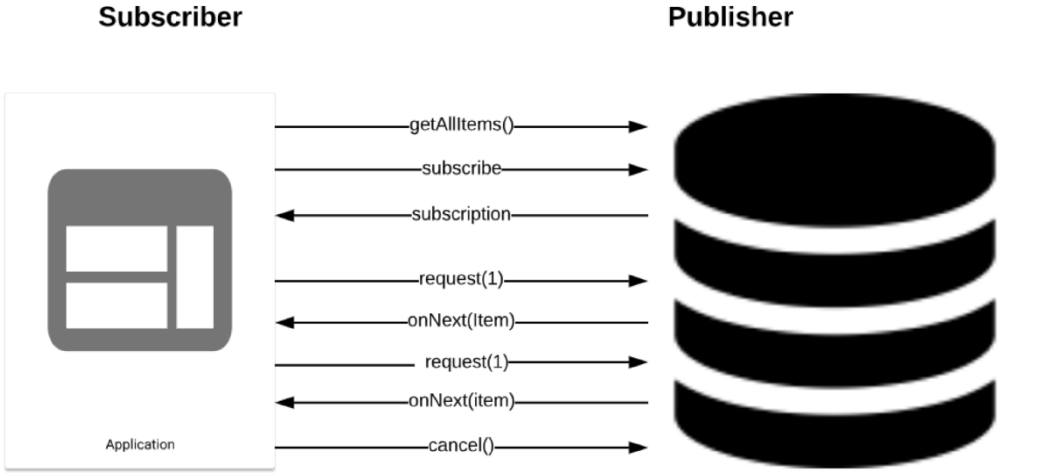
hot

Default Data Flow – Project Reactor



What is Backpressure?

- Subscriber controls the data flow from the Publisher.



Spring Boot

Develop Reactive REST APIs

Traditional Spring MVC does not have reactive support

Webflux : Web MVC Reactive framework

Webflux : Two approaches of reactive development

Similar approch of traditional MVC style (Annotation based controller)

Functional Web

Traditional style built on top of new tech

Traditional request is converted into reactive request by netty server

Unit Testing RestAPI

Traditional MVC : Test RestTemplate

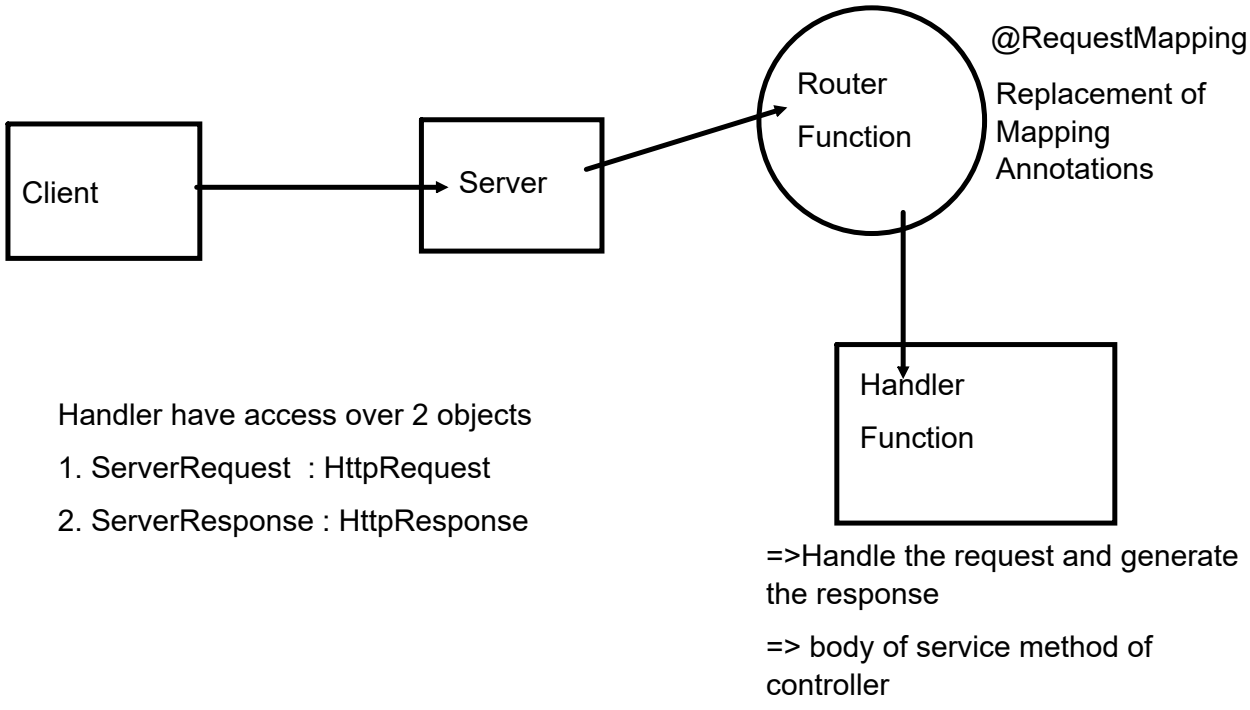
WebFlux : WebClient

1. WebClient : Reactive Client
2. WebTestClient : Unit Test for Reactive Rest Endpoints

WebFlux - Functional Web

Using Functions to route request and response

1. Router Function
2. Handler Functions



Handler have access over 2 objects

1. ServerRequest : HttpRequest
2. ServerResponse : HttpResponse

=>Handle the request and generate the response

=> body of service method of controller

2 classes

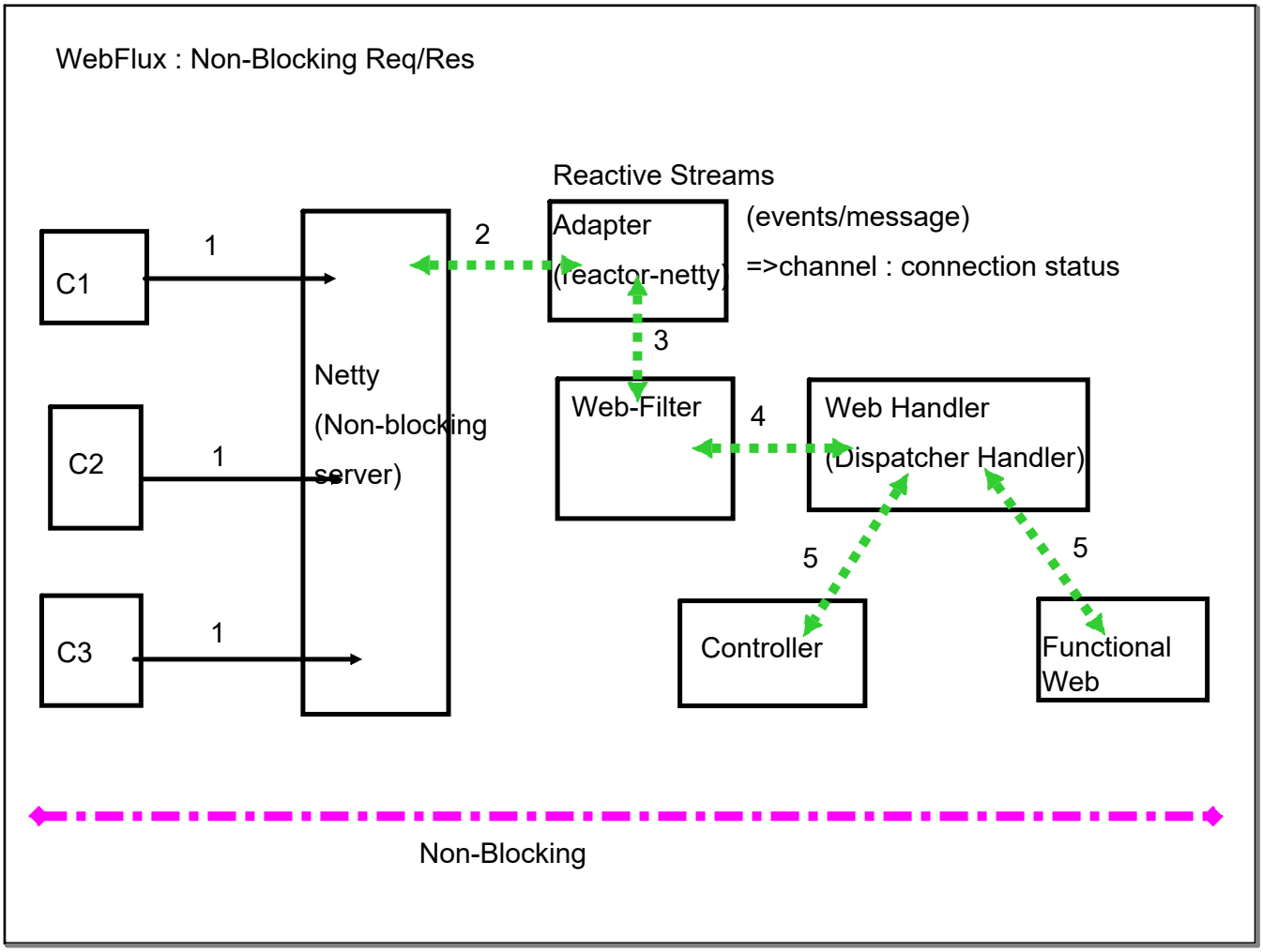
1. Container for handler function
2. Container for router function

ServerResponse provides factory method/status (BodyBuilders : used to create body of response)

Routers

Approach 1 : Individual method for each route

Approach 2 : Single method for all route


WebFlux : Non-Blocking Req/Res
Reactive Streams
Adapter
(reactor-netty)
(events/message)
=>channel : connection status
C1
C2
C3
1
1
1
2
3
4
5
5
Netty
(Non-blocking
server)
Web-Filter
Web Handler
(Dispatcher Handler)
Controller
Functional
Web
Non-Blocking

Netty :

An asynchronous event-driven network application framework used for development of high performance servers

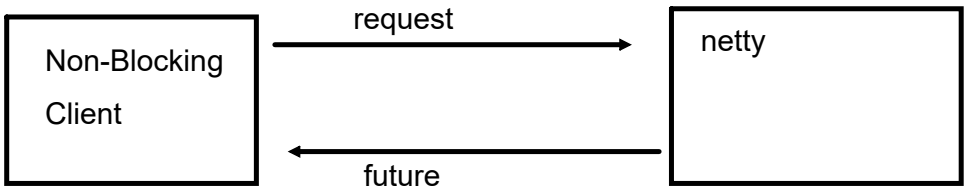
Netty is build on top of JAVA

Major clients uses netty

Protocols

FTP/HTTP/SMTP/WebSocket

Asynchronous



Events (Netty)

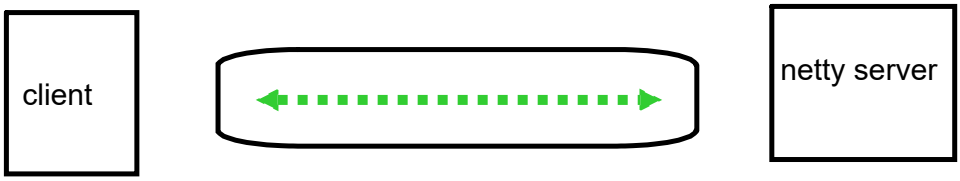
client request for new connection

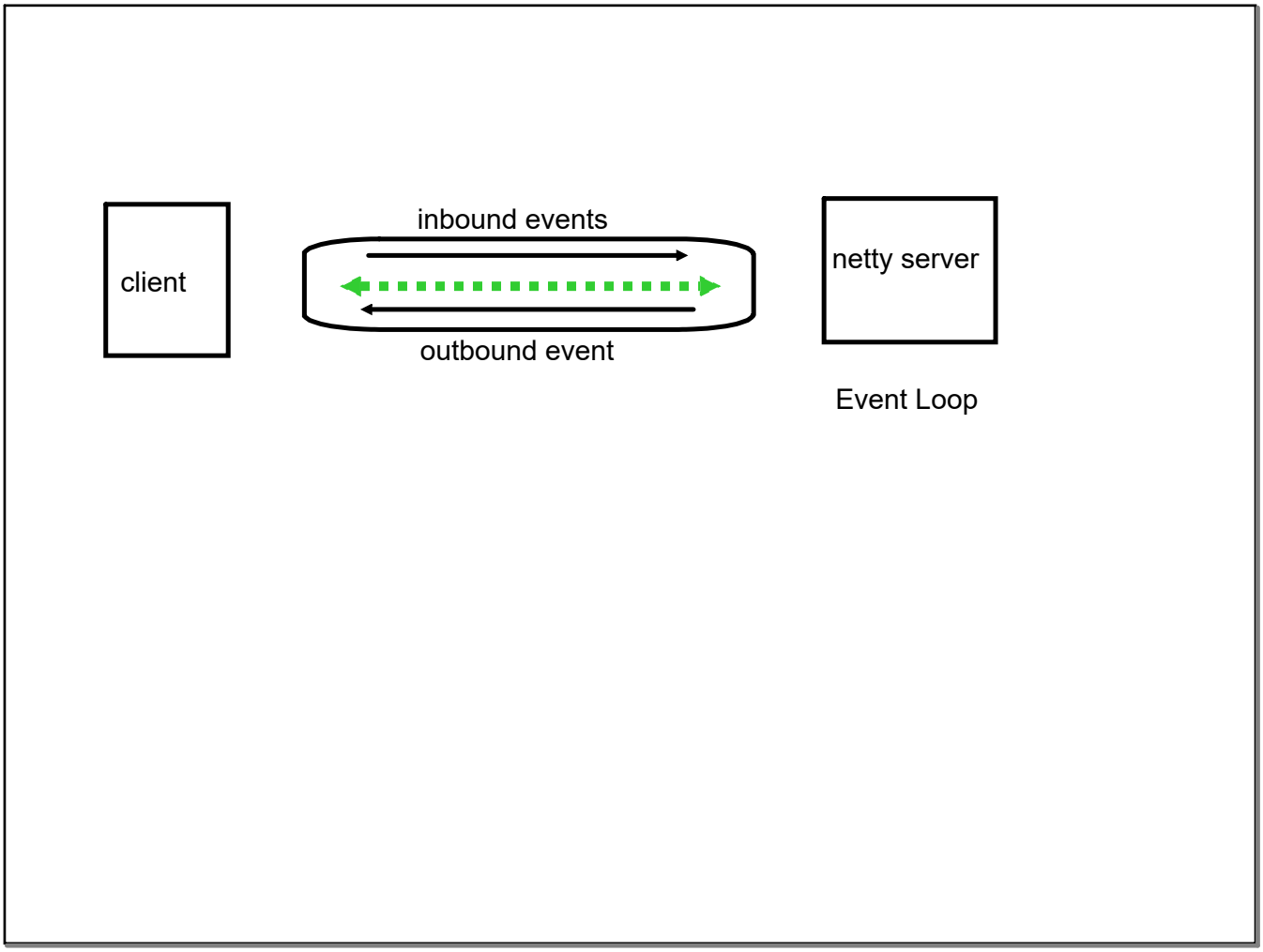
client request for data is event

client posting data is event

error are treated as event

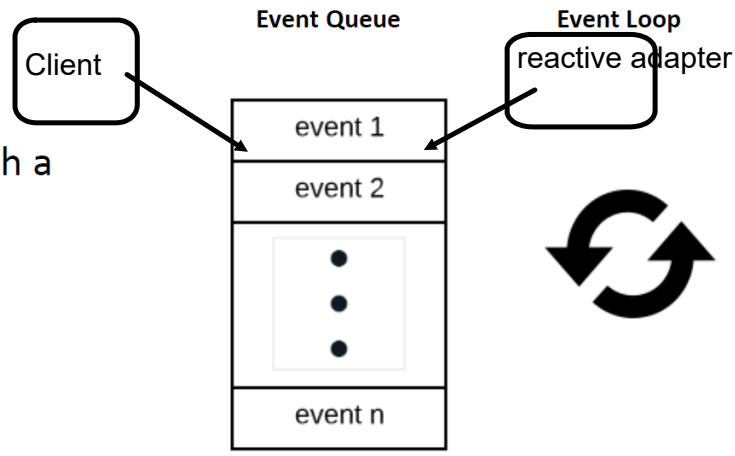
Netty -Channel : represents connection between client and server



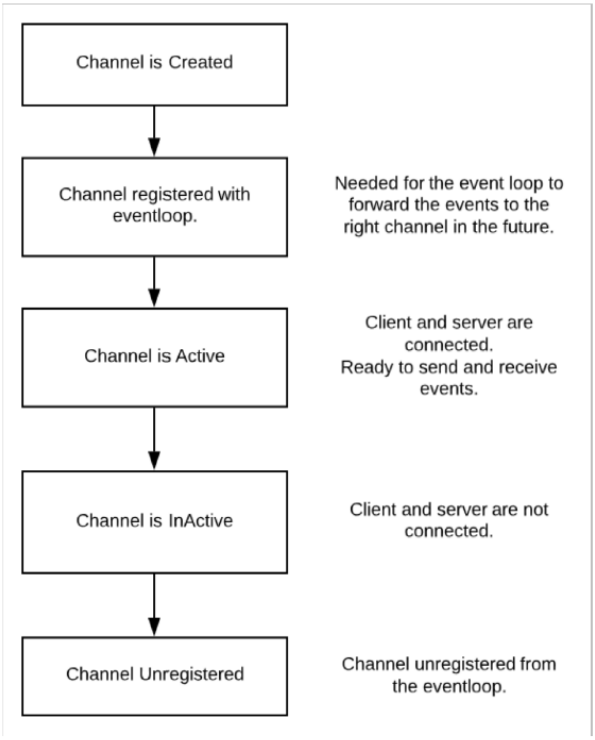

client
inbound events
outbound event
netty server
Event Loop

Netty - Event Loop

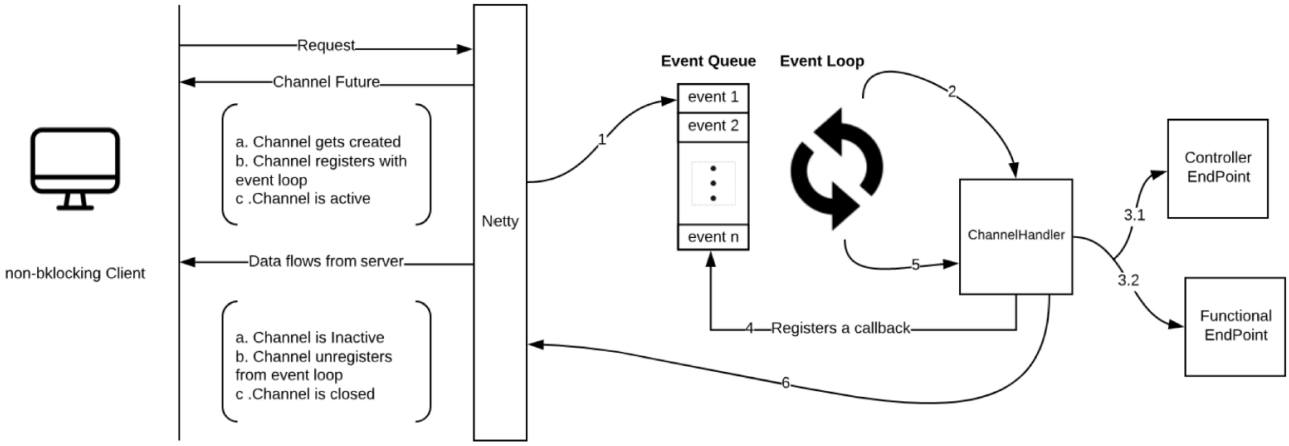
- Loop the looks for events.
- EventLoop is registered with a **single** dedicated thread.



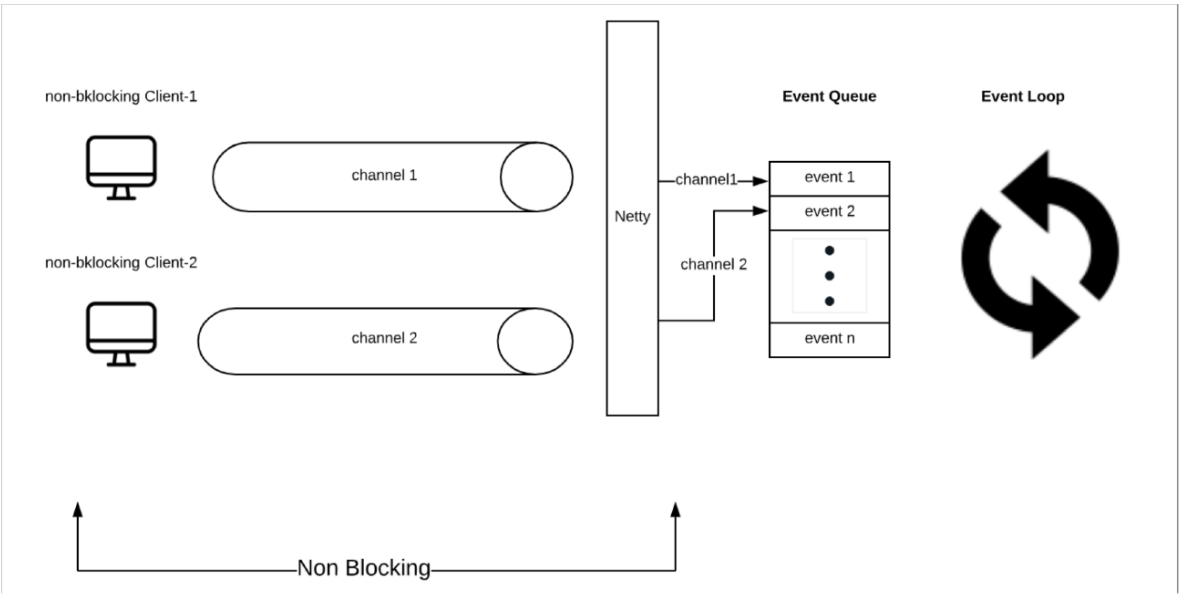
Channel LifeCycle



EventLoop + Channel



EventLoop + Multiple Channel



Netty might run multiple event loops

no of event loops : 2 x no of processor (4)

eg : 8

EventLoopGroups: multiple event loops can be clubbed

2 groups of 4 event loops:

1 : handle web request (4 event loops)

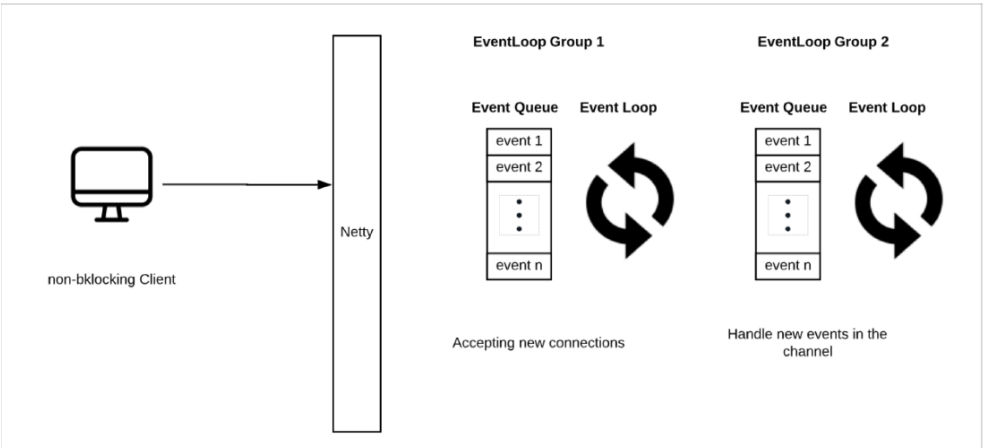
2 : IoT : (4 event loops)

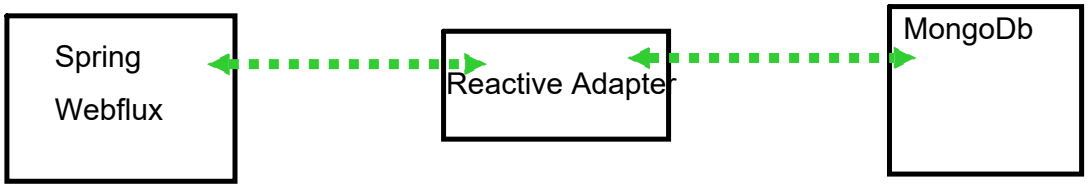
By default :

Provides 2 event-loop groups of 1 event-loop each

EventLoopGroup

- How many EventLoopGroup are there in Netty?
  - 2 EventLoop Groups.





Entity class :

Document class

Repository : JPA interface

mongodb-reactive-adapter reactive-repository interface (same as JPA)

unit test case for testing repository

For any reactive stream to work or activate :

1. we need to subscribe
2. we need to block // for testing