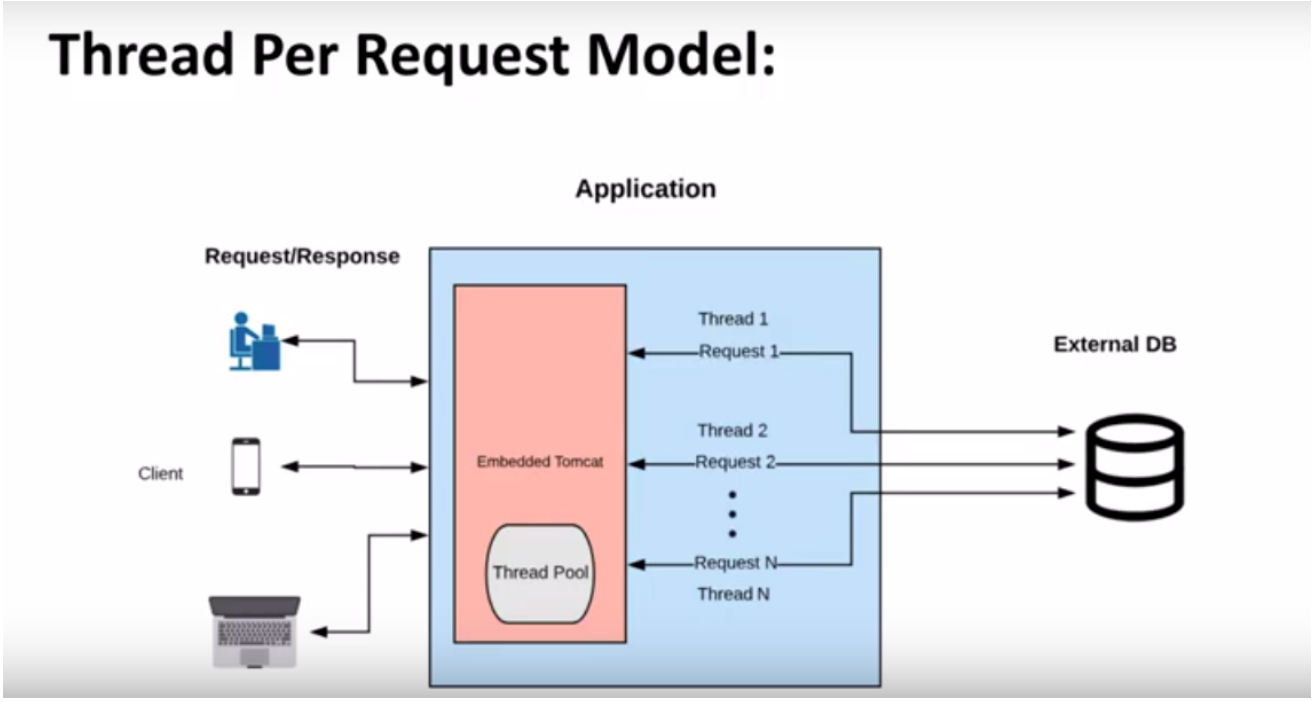
Reactive Programming

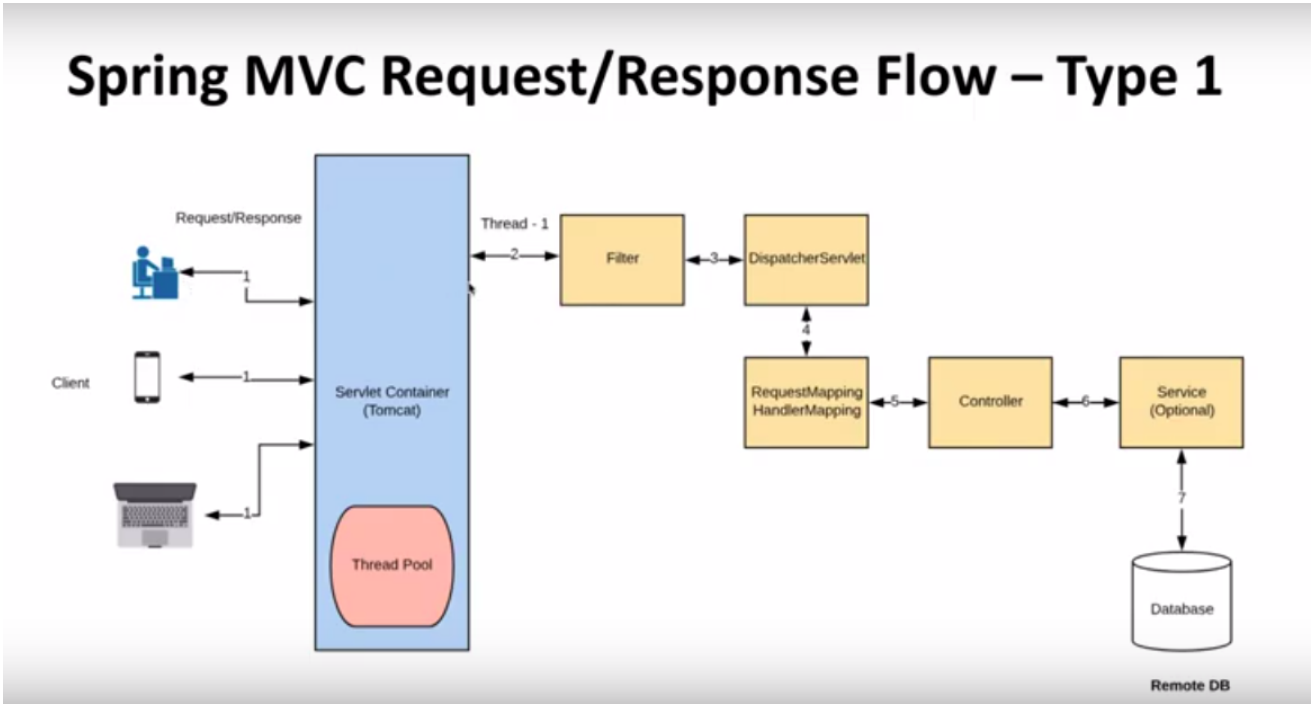
Reactive programming is an approach to writing software that embraces asynchronous I/O. Asynchronous I/O inverts the normal design of I/O processing: the clients are notified of new data instead of asking for it; this frees the client to do other things while waiting for new notifications.

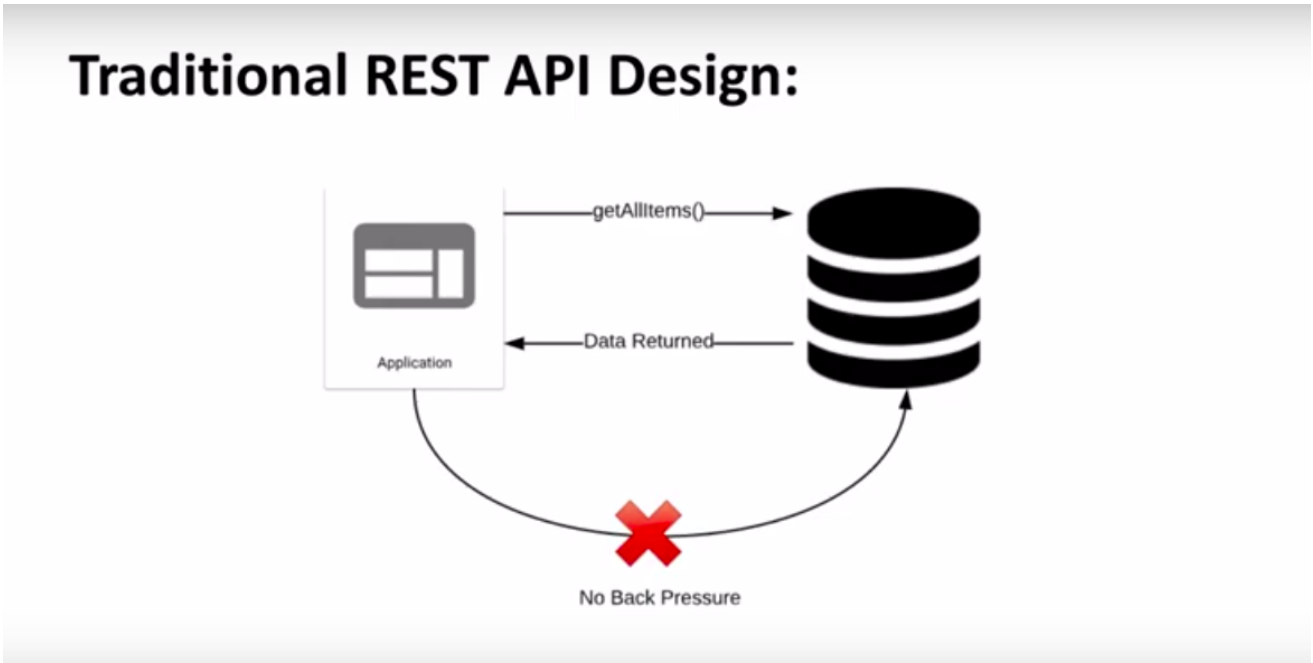
Reactive programming is a programming paradigm that promotes an asynchronous, non-blocking, event-driven approach to data processing.

Blocking vs non-blocking (async) request processing

Blocking way-



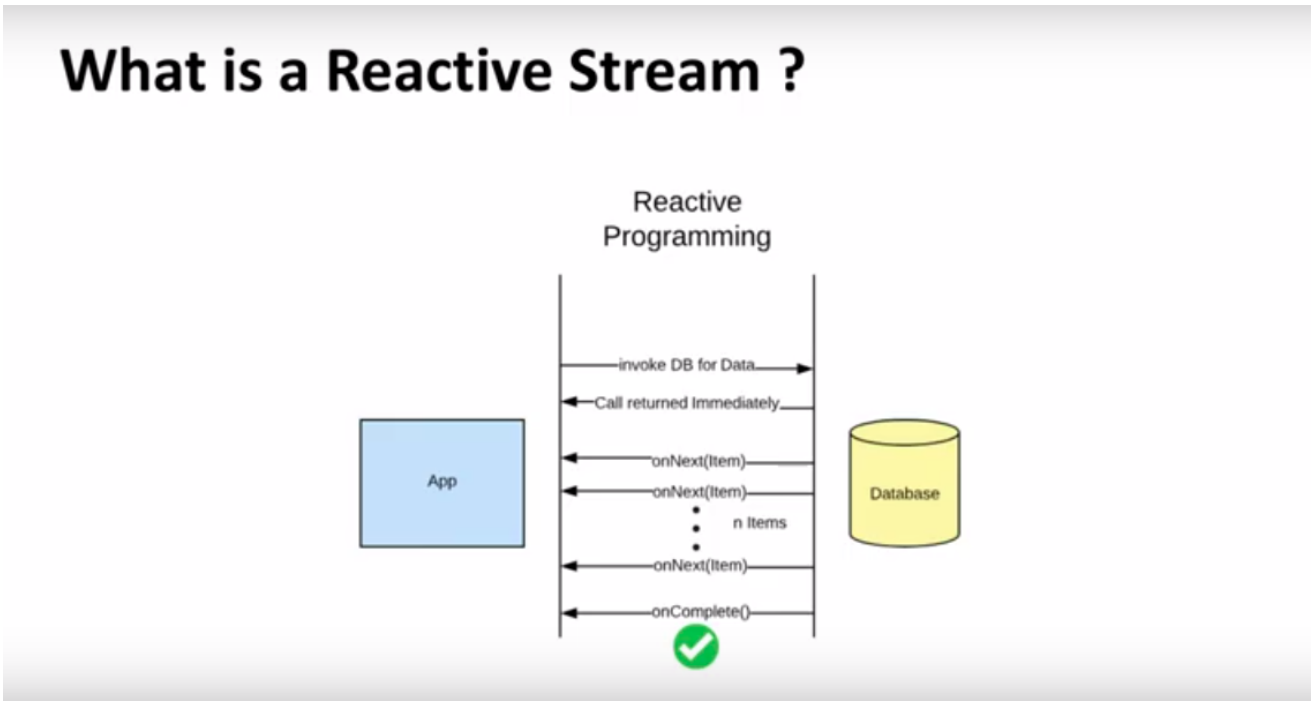




Better Api Design -

- Aynchronous and non blocking
- Move away from thread per request model
- Use fewer threads
- Back Pressure compatible

What is Reactive Stream:- Data flowing as an event driven stream



Reactive Stream Specification:-
https://github.com/reactive-streams/reactive-streams-jvm

- **Publisher** :- The Publisher<T> is a producer of values that may eventually arrive. A Publisher<T> produces values of type T to a Subscriber<T>.

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

- **Subscriber**:- The Subscriber subscribes to a Publisher<T>, receiving notifications on any new values of type T through its onNext(T) method. If there are any errors, its onError(Throwable) method is called. When processing has completed normally, the subscriber's onComplete method is called.

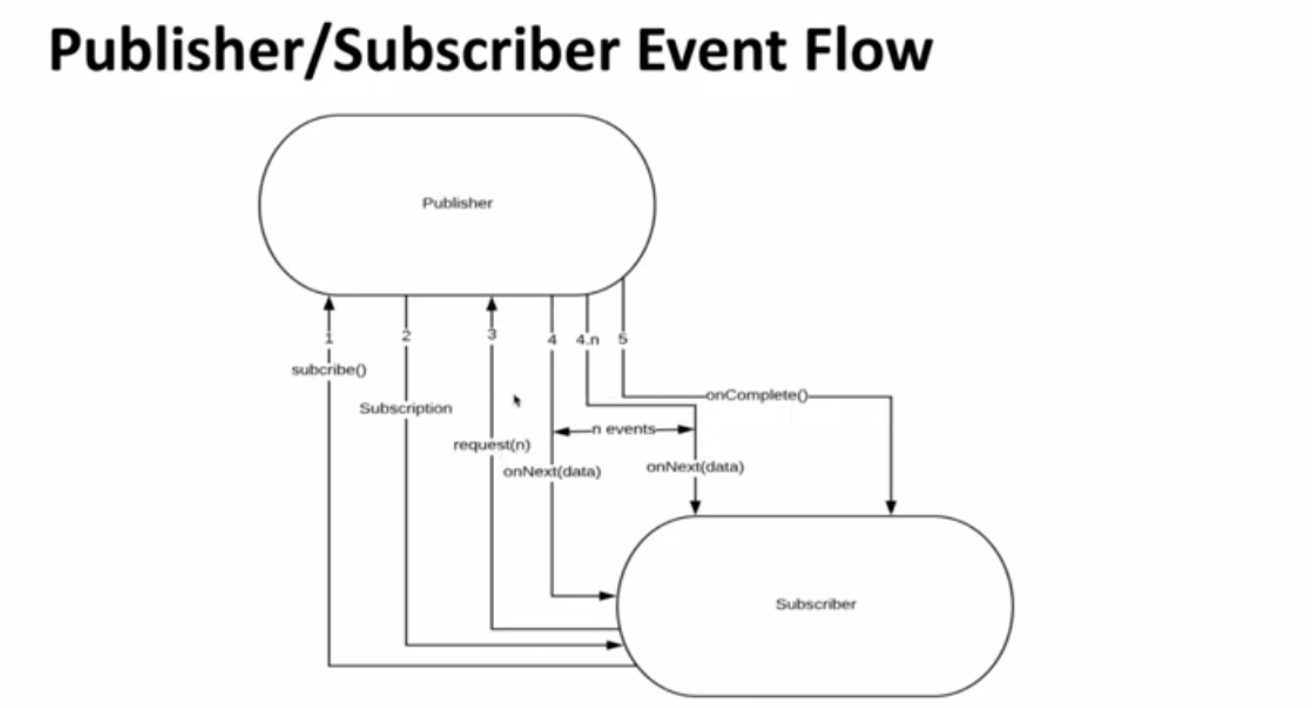
```
public interface Subscriber<T> {
    public void onSubscribe(Subscription s);
    public void onNext(T t);
    public void onError(Throwable t);
    public void onComplete();
}
```

- **Subscription:-** It enables backpressure. The Subscriber uses the Subscription#request method to request more data or the Subscription#cancel method to halt processing.

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

- **Processor:-** A Processor<A,B> is a simple interface that extends both Subscriber<A> and a Publisher<B>

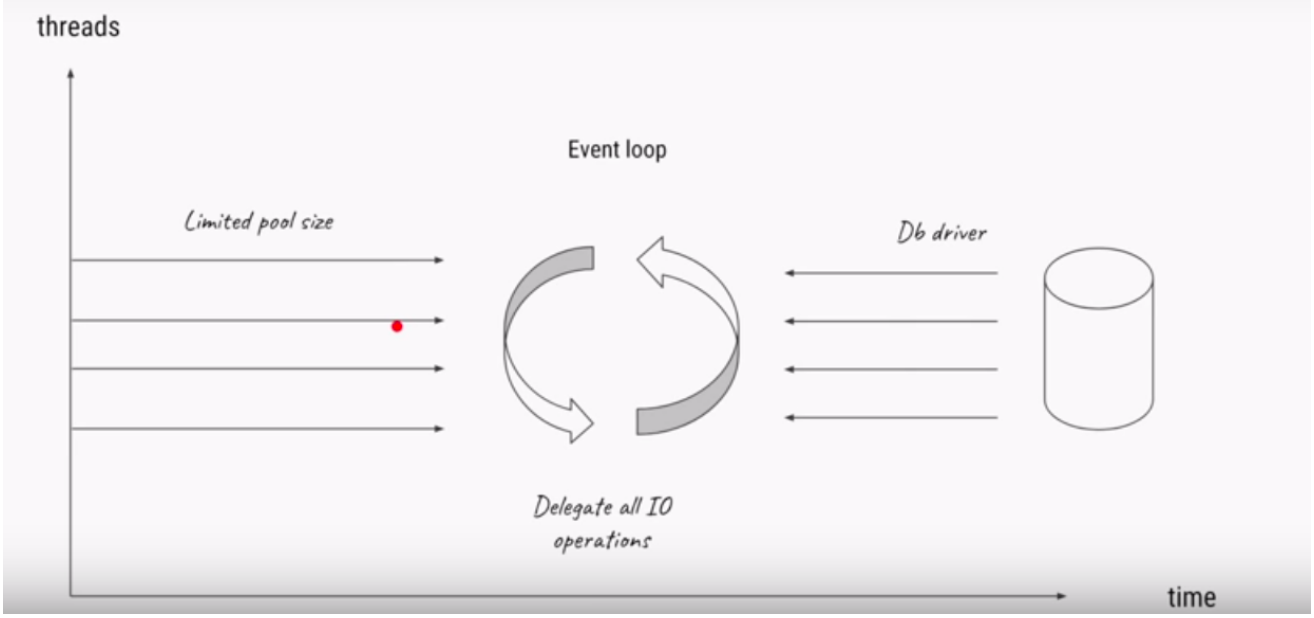
```
public interface Processor<T, R> extends Subscriber<T>, Publisher<R> {

}
```



Why was Spring WebFlux created ?

Need for a non-blocking web stack to handle concurrency with a small number of threads and scale with fewer hardware resources. Servlet 3.1 did provide an API for non-blocking I/O.

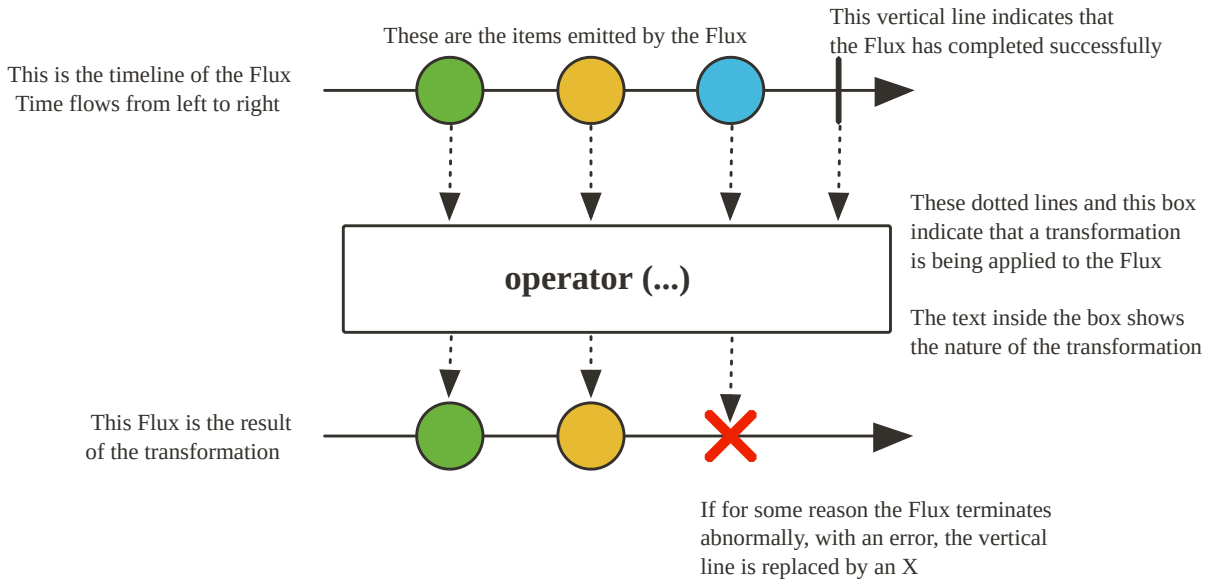
Event Loop:-



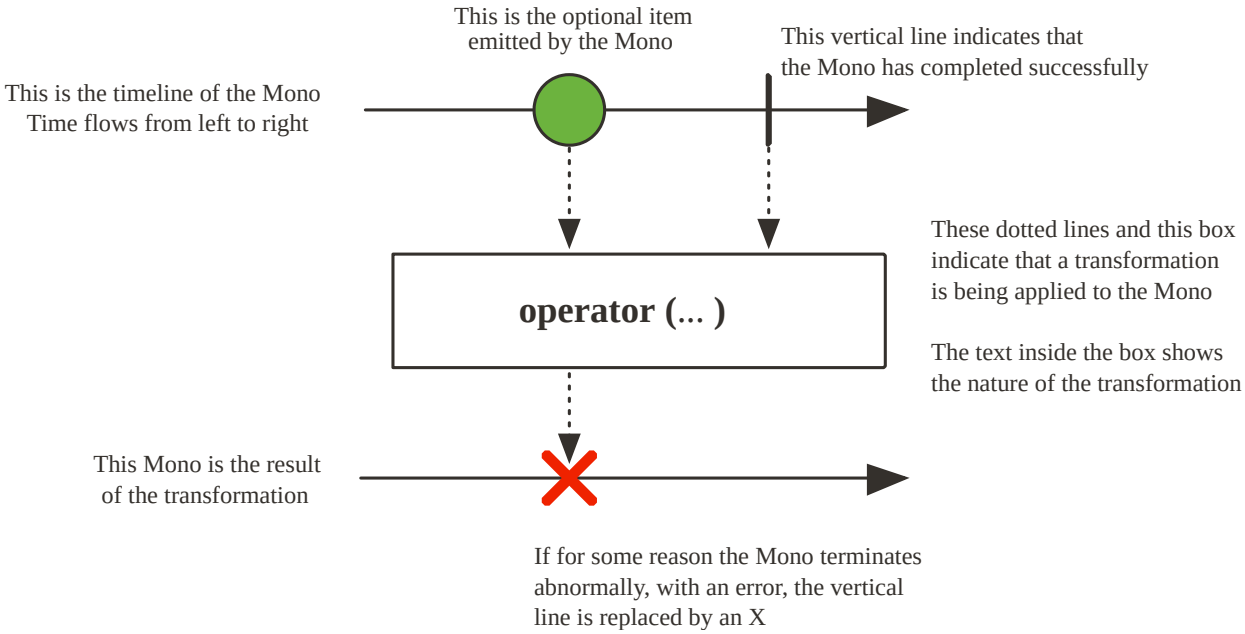
Spring boot uses project reactor which provides a lot of components like Reactor core, reactor test, reactor netty etc.

Reactor Core:- Spring-webflux depends on reactor-core and uses it internally to compose asynchronous logic and to provide Reactive Streams support. Generally, WebFlux APIs return Flux or Mono (since those are used internally) and leniently accept any Reactive Streams Publisher

Flux (0..N) & Mono (0..1)



Mono-



"Reactive and non-blocking generally do not make applications run faster."

WebClient:- *WebClient* is an interface representing the main entry point for performing web requests. Spring WebFlux includes a reactive, non-blocking WebClient for HTTP requests.
Internally WebClient delegates to an HTTP client library. By default, it uses Reactor Netty.