Spring Reactive

Practical Efficiency

Overview

- Introductions
  - Instructor - Geoff Matrangola - geoff@matrangola.com @triglm
  - VILT: Lori deRoin (Develop Intelligence) & Sofiya Momchilova (VMWare)
  - Company - DevelopIntelligence http://www.developintelligence.com/ (show 2 slides)
  - Students - Names, Current projects, Class Expectations
- Logistics
  - Start, end, break times
  - Online
- Class Flow
  - Slides
  - Demo
  - Lab

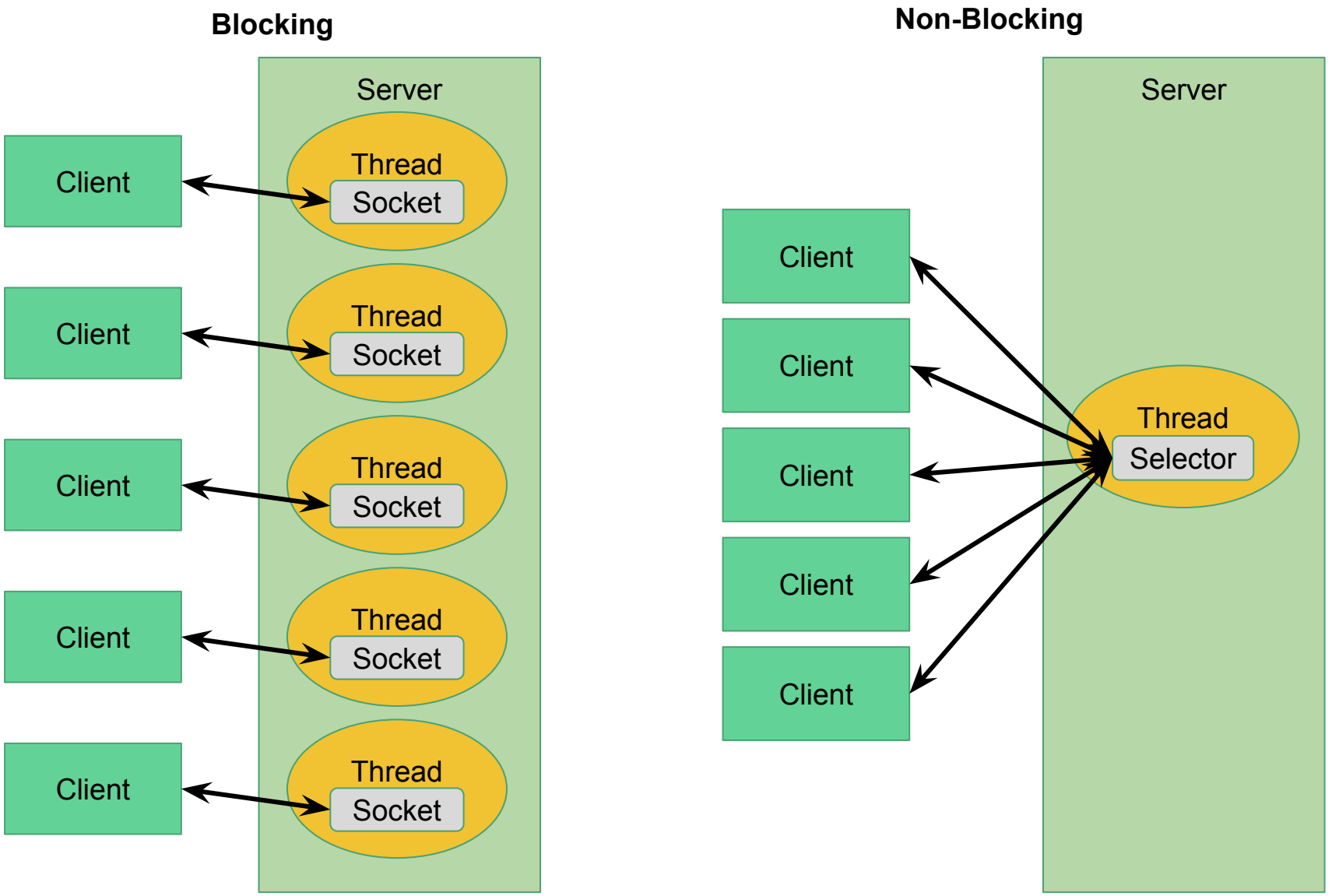
What We Will Cover

1. Introduction and Overview
2. Non-Blocking http servers
3. Reactive vs Traditional
4. Spring Support for Reactive
    a. Configuration
    b. Mono
    c. Flux
5. Reactive with NOSQL datastore (Cassandra)
6. Reactive WIth Microservices
    a. Cooperative non-blocking flow
    b. Reactive WebClient
7. Reactive web service with Server Side Events

Non-Blocking vs Blocking HTTP Servers

- Blocking
  - One thread for each connection
  - Additional OS and memory overhead per connection
  - Careful thead management and capacity planning required
- Non-Blocking
  - Multiple connections per thread
  - Server requires less memory and OS overhead
  - Non-Blocking Server has a more complex implementation
  - Using Java NIO package Selector, Channels, and Buffer classes
  - Most efficient if it is reactive/non-blocking all the way through the stack

HTTP Connections



Spring Reactive

Publisher<T> - sends messages to the Subscriber

Subscriber<T> - Receives Messages from the Publisher

Mono<T> - Publisher for a single object *0..1*

Flux<T> - Publisher for a list of objects *0..n*

ReactiveCrudRepository<T, ID> - Repository that accepts and returns Mono<T> and Flux<T>

produces = **MediaType.TEXT_EVENT_STREAM_VALUE** - Turns Flux into a Server Sent Event message for Servlet 3

Demo 1 Web on Reactive

- Create a Web Service that will publish global economic indicators
- Open Project in IDE that was created with
  - http://start.spring.io : Create a Web Project Using Gradle, Spring Boot, Reactive Web, Reactive Cassandra, Lombok and DevTools.
  - Configured IDE to Use Lombok
- Create a data model for the Indicator
  - Use Lombok Annotations to reduce boilerplate Java Code
- Create a RequestMapping that returns a hardcoded indicator
  - Show as standard Rest Indicator
  - Convert to return **Mono**<**Indicator**>
- Use Browser or http client test

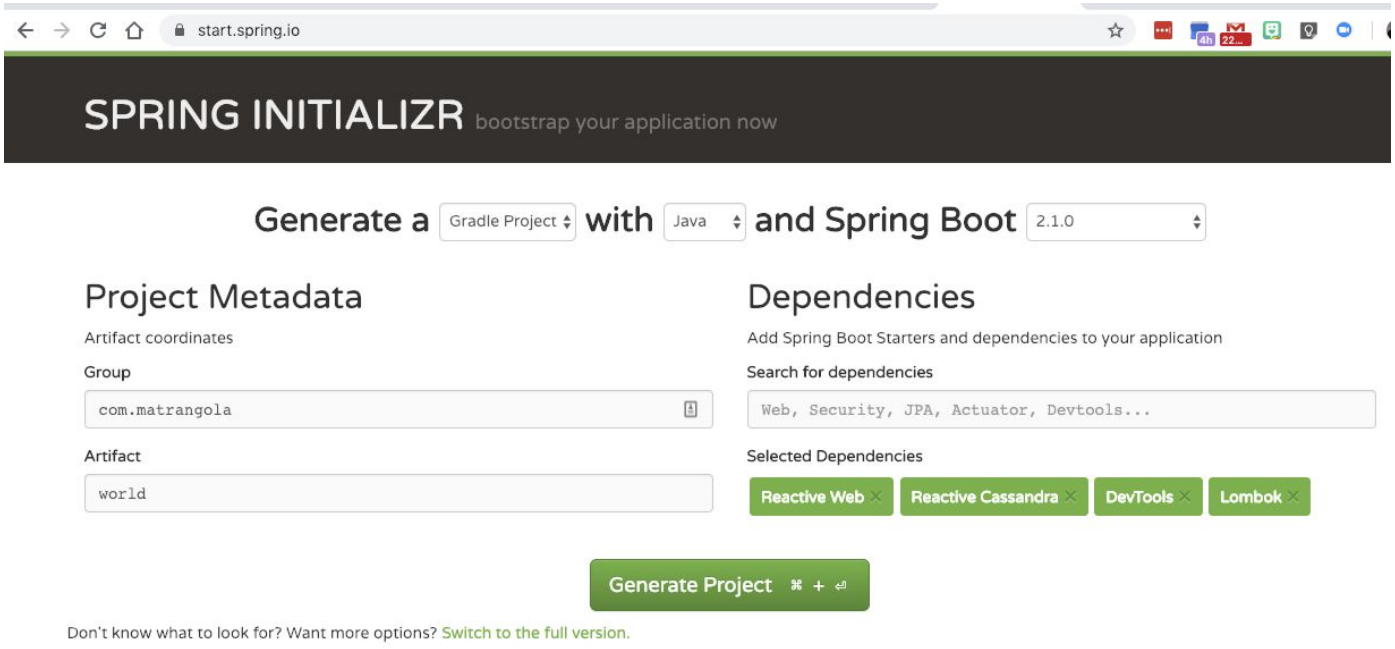
Lab 1 Web on Reactive

- Setup
  - Use Lab Starter on github <TBD>
  - **OR**
  - http://start.spring.io : Create a Web Project Using Gradle, Spring Boot, Reactive Web, Reactive Cassandra, Lombok and DevTools.
  - Configure IDE to Use Lombok
- Create a data model for the Indicator
  - Use Lombok Annotations to reduce boilerplate Java Code
- Create 2 **RequestMapping**s
  - Use **Mono**<**T**> Return the Patent Application Indicator for BGR, IP.PAT.RESD
  - Use **Flux**<**T**> Return the Patent Application Indicator and High Tech exports indicator (IP.PAT.RESD and TX.VAL.TECH.CD)
- Use Browser or http client test

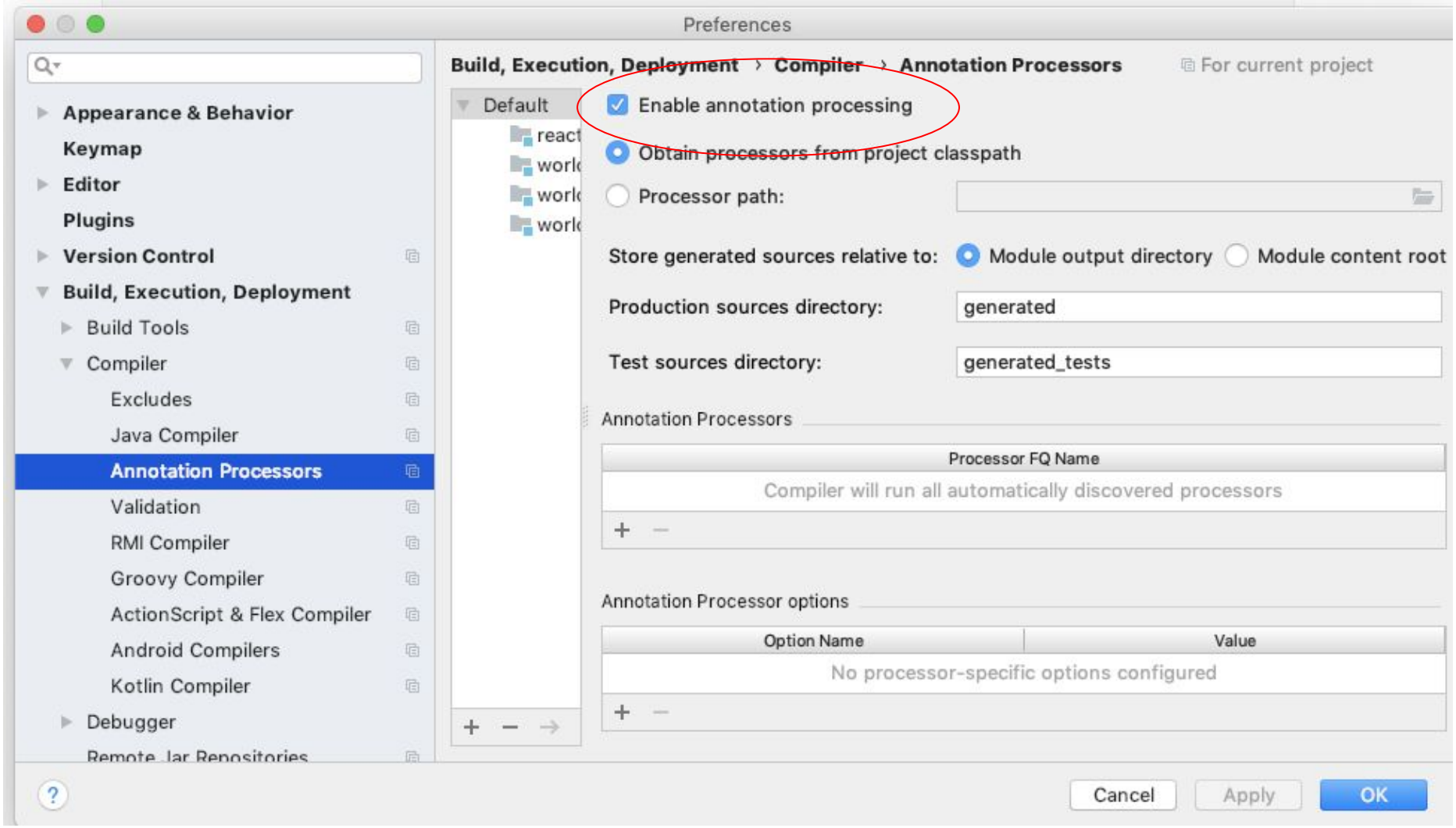
Helper: WebFlux Documentation

- https://docs.spring.io/spring/docs/current/spring-framework-reference/web-reactive.html#webflux-controller
- https://grokonez.com/reactive-programming/reactor/reactor-create-flux-and-mono-simple-ways-to-create-publishers-reactive-programming

Lab 1: Helper start.spring.io

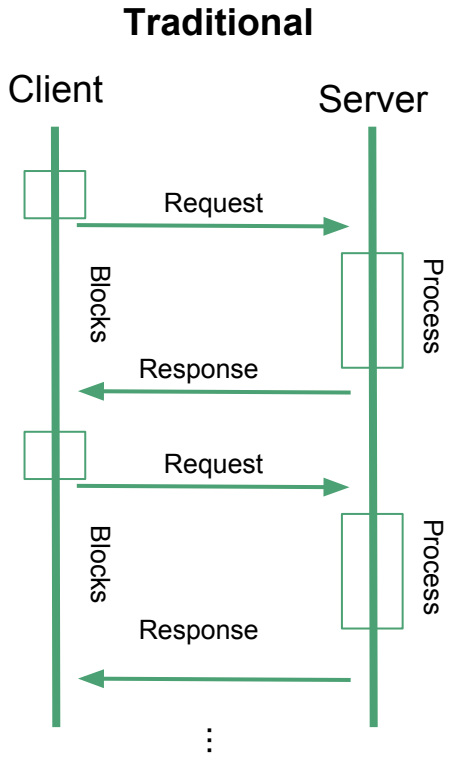


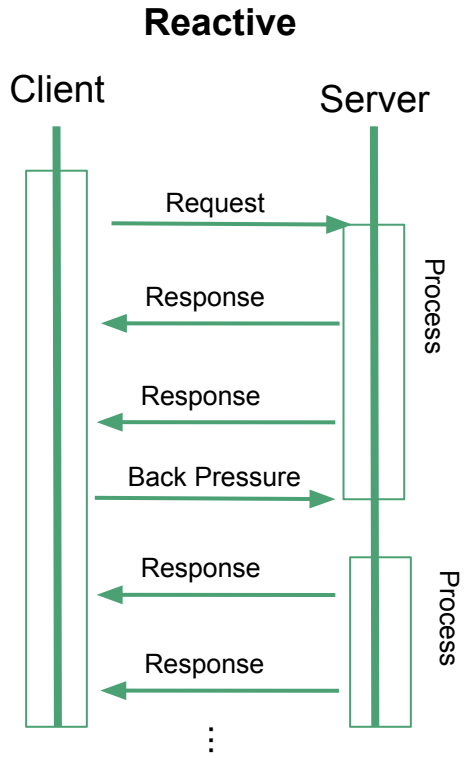
Lab 1 Helper: Idea IntelliJ Annotation Lombok



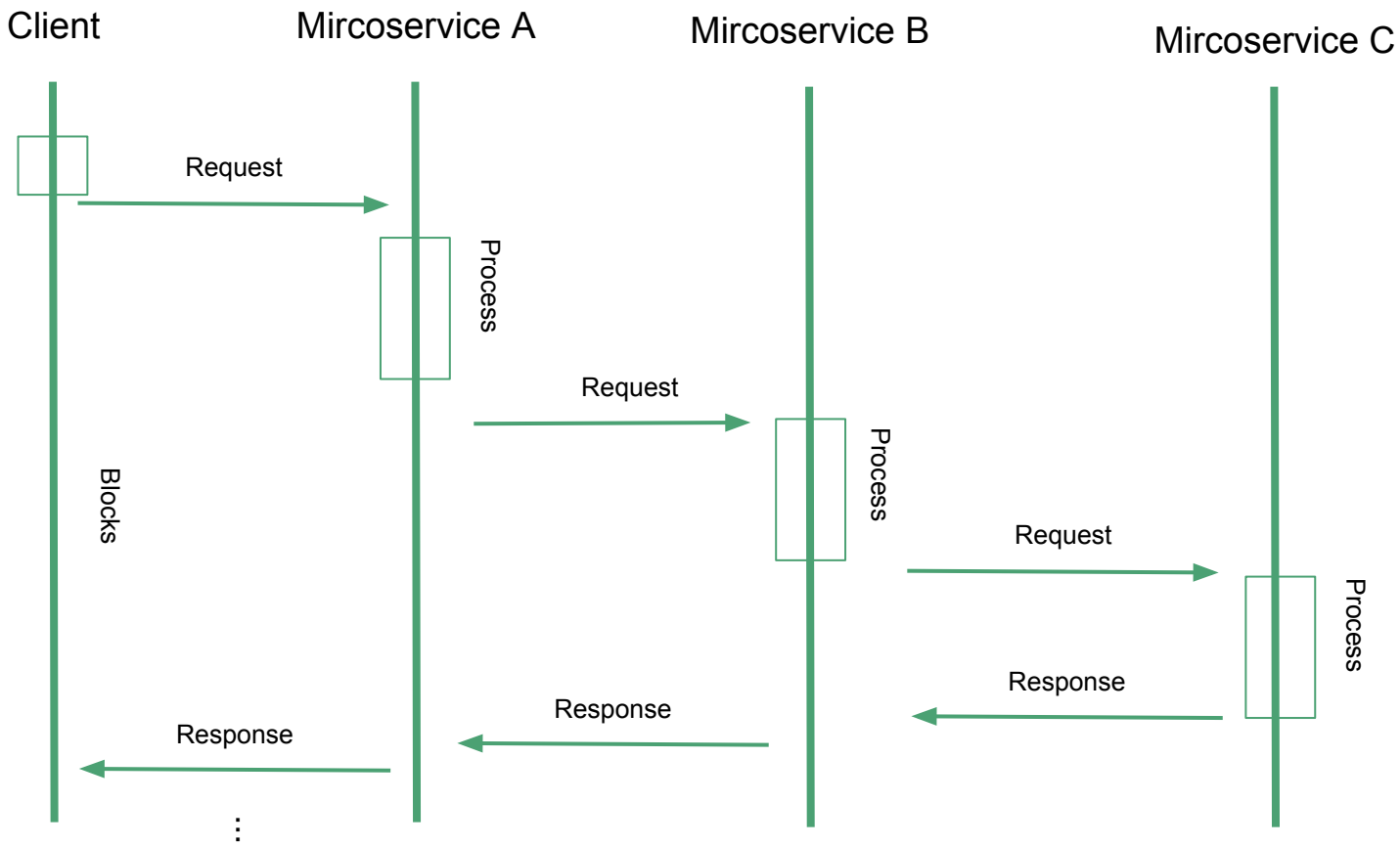
Reactive vs. Traditional OOP

- Non-Blocking
- Asynchronous and event-driven
- Scalable with a small number of threads

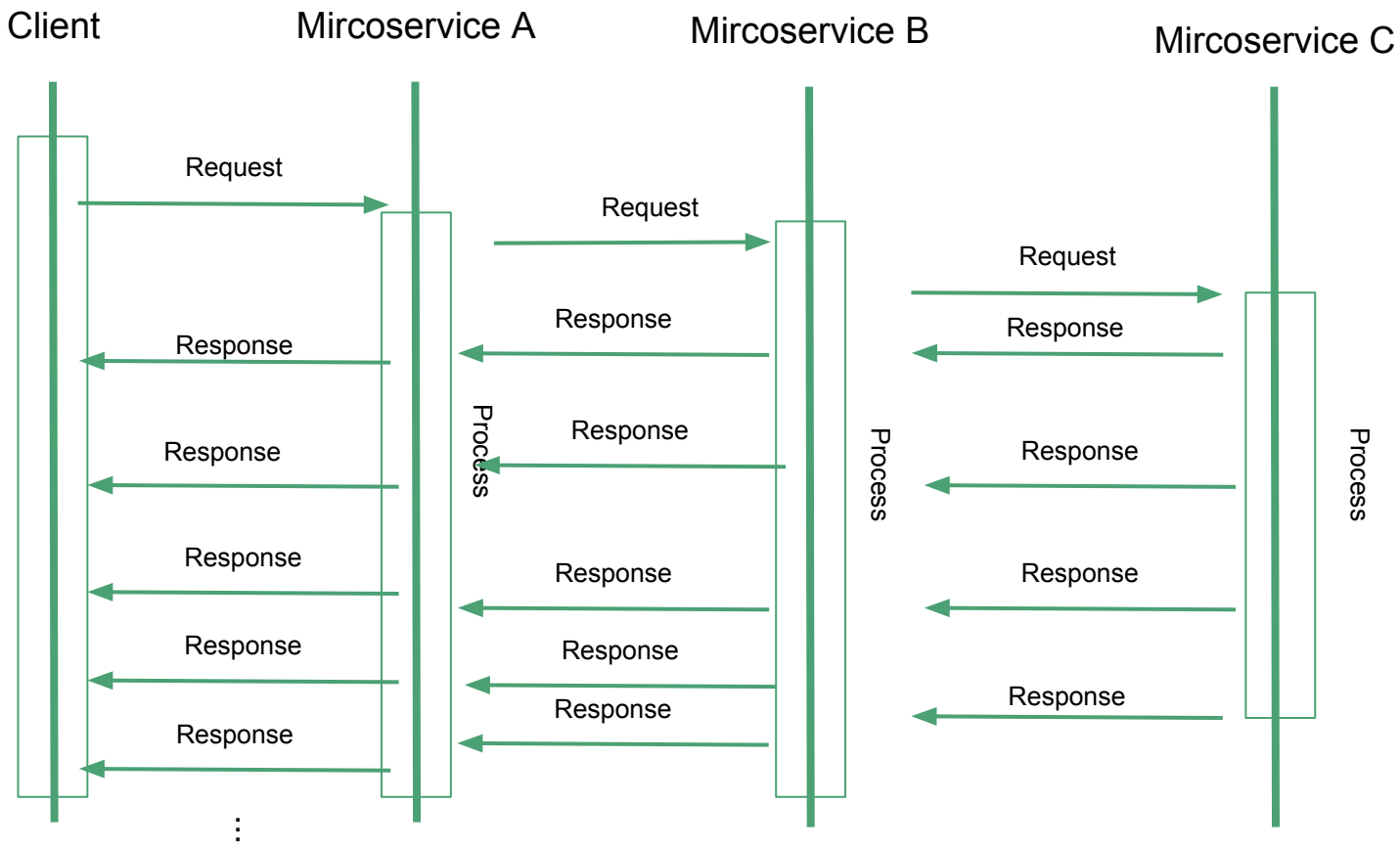




Microservice Traditional Messaging



Microservice Reactive



Reactive Cassandra NOSQL Server

- Non-Blocking for Reactive Web Efficiency
- A standard Cassandra Repository can be made to support Reactive by using as `ReactiveCassandraRepository` the base interface instead of `CassandraRepository`
- Methods can return **Mono**<T> and **Flux**<T> when returning results.
- SaveAll()/Insert() can accept a **Publisher**<T> so that values can be streamed into the database
- findAll() etc. will return a **Flux**<T>

Demo 2 - Cassandra Repository

Use World Bank Data for Country Economic Indicators in recent decades

- Use provided docker running Cassandra
- Load the reference data from csv into Cassandra
- Add Cassandra Annotations to the Indicator Class
  - Use Cassandra Annotations for @PrimaryKeyColumn and @Indexed
  - Add Index for country
  - Explain alternative ways to do primary keys (@PrimaryKey and @PrimaryKeyColumn)
- Copy and explain Create Cassandra Repository for the indicator
  - Add the findAllByCountryCode
- Create CassandraConfig.java and add fields to application.properties
- Add @Autowired reference to the Repository in the Controller
- Update WorldController
  - Add Mapping to find all Indicators and return a Flux<Indicator>
  - Add Mapping that gets by country code
- Create a http to get all and get by Country

Lab 2 - Cassandry Repository

- Use provided docker running Cassandra
- Load the reference data from csv into Cassandra
- Add Cassandra Annotations to the Indicator Class
  - Use Cassandra Annotations for @PrimaryKeyColumn and @Indexed
  - Add Index for country and indicatorCode
- Create Cassandra Repository for the indicator
  - Add the findAllByCountryCode
  - Add find for Index ID
- Copy CassandraConfig.java and add fields to application.properties
- Add @Autowired reference to the Repository in the Controller
- Update WorldController
  - Add Mapping to find all Indicators and return a **Flux**<**Indicator**>
  - Add Mapping that gets by country code
  - Add Mapping for Index ID
- Create a http or run in browser all new GetMappings

Server Sent Events

- Supported by JavaScript (in most browsers)
- Supported by Android SDK (but use a library like https://github.com/heremaps/oksse)
- Supported by iOS (user library like https://github.com/inaka/EventSource)

API	Chrome	IE	Firefox	Safari	Opera
SSE	6.0	Not supported	6.0	5.0	11.5

https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events/Using_server-sent_events

https://www.w3schools.com/html/html5_serversentevents.asp

Demo 3 - Server Sent Events

- Create a RequestMapping that returns simulated live update
- `@GetMapping(value = "/updates/{countryCode}", produces = MediaType.TEXT_EVENT_STREAM_VALUE)`
- Use Flux.interval

Lab 3 - SSE

- Create a RequestMapping that returns simulated live update
- `@GetMapping(value = "/updates/{countryCode}", produces = MediaType.TEXT_EVENT_STREAM_VALUE)`
- Use Flux.interval
- Create a RequestMapping that returns a Mono<Indicator> and simulates a long-running calculation by calling Sleep and finally published an SSE Event

WebClient

- Non-blocking interface using HTTP/1.1
- Uses fluent class factory API to define Base URL, Cookies, Headers, etc.
- Has .get() and post() methods
- bodyToMono and bodyToFlux are used to keep it Reactive

https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-webclient.html

https://www.baeldung.com/spring-5-webclient

Demo 4a - Consumer Microservice

- Open consumer project (global)
  - http://start.spring.io : Create a Web Project Using Gradle, Spring Boot, Reactive Web, Lombok and DevTools.
  - Configured IDE to Use Lombok
- Copy Indicator bean from other project
- Remove Cassandra stuff from Indicator
- Create GlobalController
  - Create Mapping that takes an indicator name on the PathVariable and returns a Mono<Double>
  - WebClient fluent API to build a connection to base URL **http://localhost:8080/world/**
  - Use fluent api to have the client get the index path, passing in the code on as a query param.
  - Take the resulting JSON body and convert it to an Indicator Flux
  - Use the filter, map and reduce methods to filter out nulls, map the year2017, and sum the result
- Put on port 8089 in application.properties
- Test with browser or request script

Demo 4b - MathFlux

- Refactor map() reduce() code to use MathFlux.sumDouble()

Lab 4 - Consumer Microservice

- http://start.spring.io : Create a Web Project Using Gradle, Spring Boot, Reactive Web, Lombok and DevTools.
- Open in IDE
- Add `implementation('io.projectreactor.addons:reactor-extra')` and `implementation('io.dropwizard.metrics:metrics-core:3.2.6')` to build.gradle
- Create Controller Class with @RestController and @RequestMapping
- Create method that is a @GetMapping with path variable code that returns **Mono<Double>**
- Use MathFlux to return the global sum of the year 2017 of the specified indicator
- Add a method that returns the global average for the specified Indicator for the year 2017