Reactive for the Impatient
(Java Edition)

IBM Developer

Mary Grygleski
Java Developer Advocate
@mgrygles

A Gentle Intro to Reactive Programming and Reactive Systems with a survey of

4 popular Reactive Java tools and libraries:

RxJava
Spring Reactor
Akka
Eclipse Vert.x

Why Reactive?

IBM **Developer**

Evolving changes/demands in the Computing Ecosystem

- Hardware level
- * Virtualization and cloud strategies
- Software System Level
- Software Application Level
- The impatient human beings!

What is Reactive?

IBM Developer

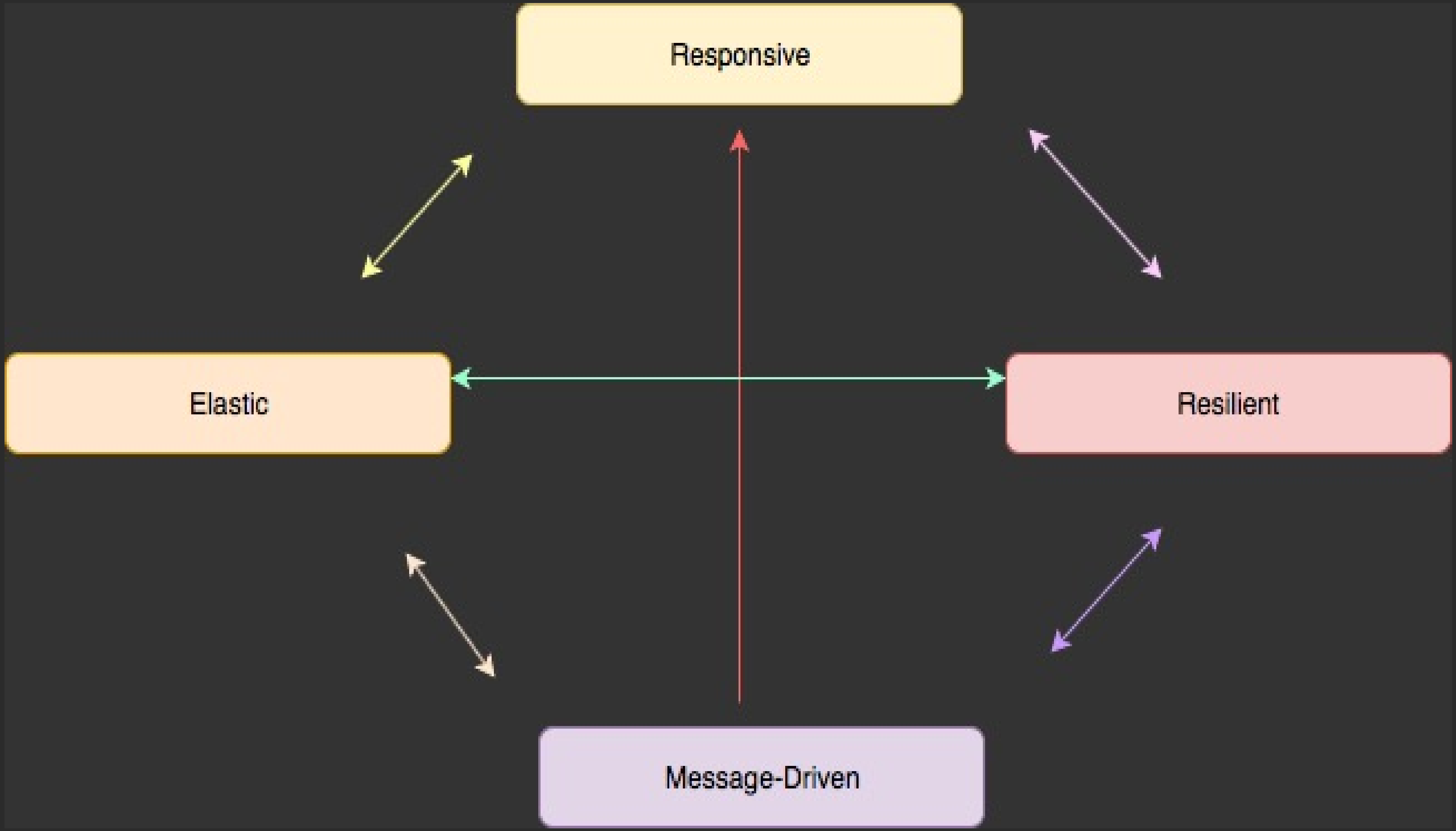
Reactive Manifesto

https://www.reactivemanifesto.org

Version 2.0 (September 2014)

*** More flexible systems**
***Highly responsive**
***More tolerant of failures**
***Handling of failures**

Reactive Principles



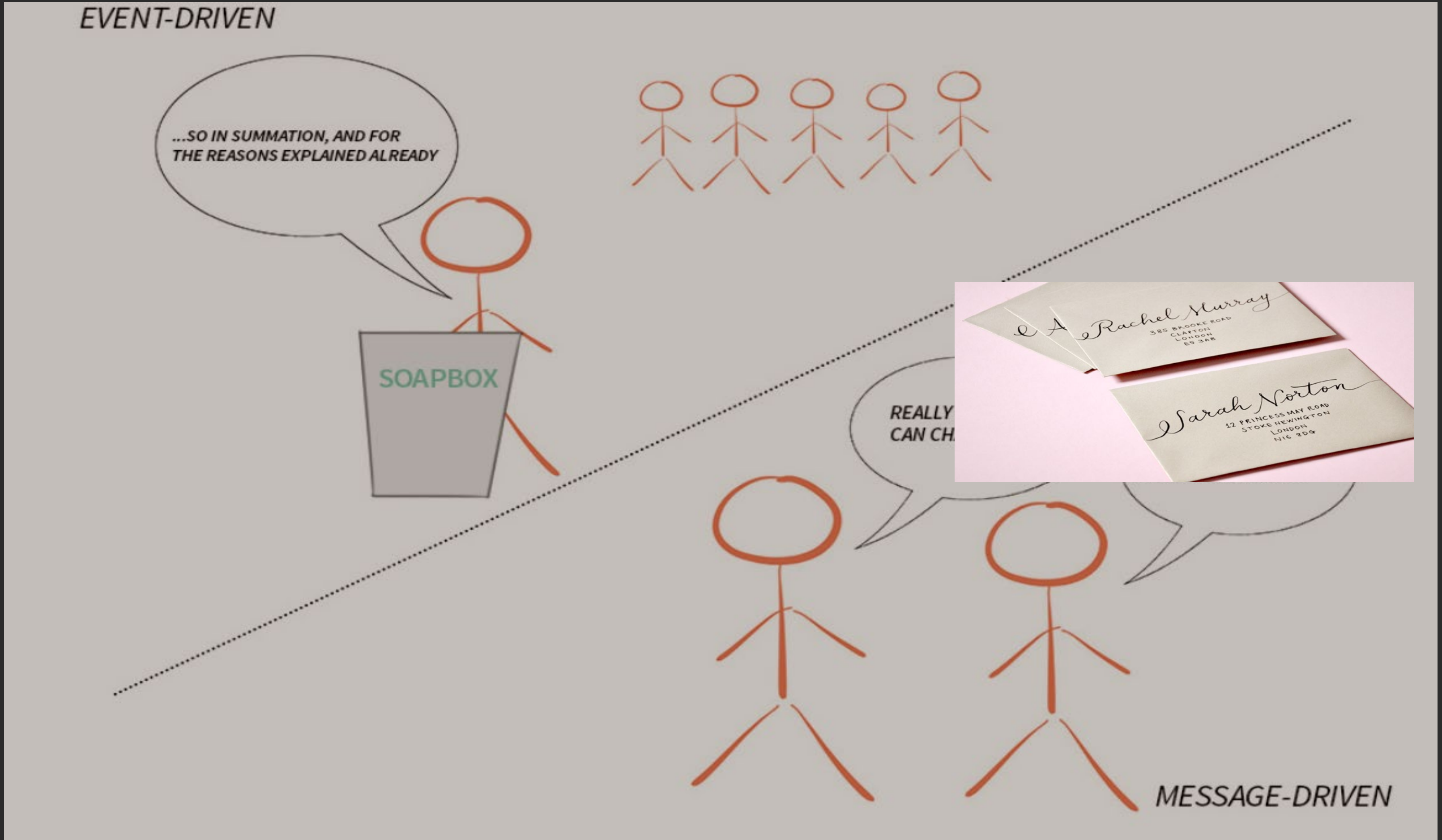
Important distinctions…

Reactive Programming

Functional Reactive Programming

Reactive Systems and Architecture

Event-Driven vs Message-Driven



While event-driven communication takes a "soapbox" approach, broadcasting facts (events) for others to observe (if they are listening), message-driven communication has an addressable recipient and a single purpose.

An Interesting "Reactive" Use Case:

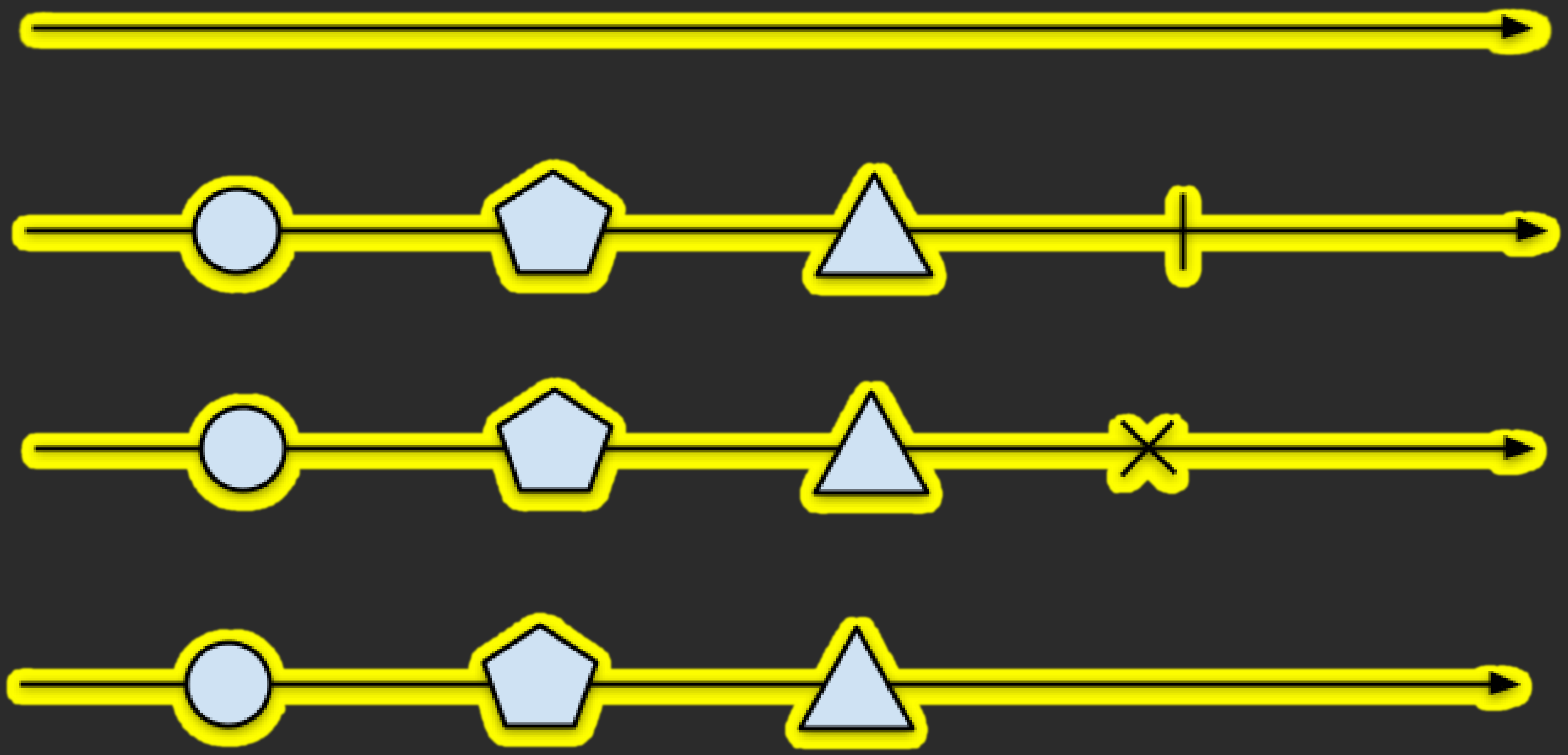
Menya Musashi Ramen Shop in Tokyo

Reactive Programming: Patterns, Terminologies

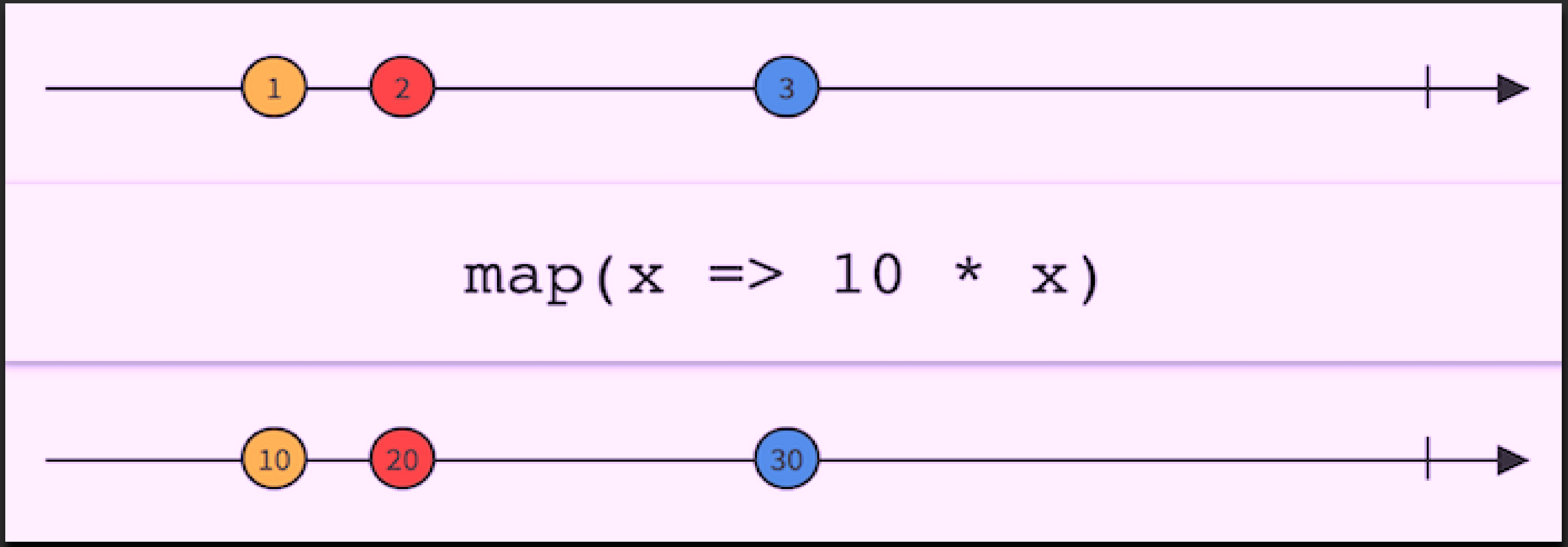
- Reactivity
- Events
- Streams
- Observables

Design Patterns:
Observer, Composite, Iterator

Rx Marble Diagram



Rx Marble Diagram: Map()



```
Observable.just(1, 2, 3)
        .map(x -> 10 * x)
        .subscribe(x -> Timber.d("item: " + x)));
```

Output:

```
item: 10
item: 20
item: 30
```

Source: ***https://rxmarbles.com***

Reactive Systems Design: Patterns

- State Management and Persistence Patterns
- Flow Control Patterns
- Message Flow Patterns
- Fault Tolerance and Recovery Patterns
- Replication Patterns
- Resource Management Patterns

Reactive Systems Design: Terminologies

- Reactive Microservices vs Monoliths
- Isolation of State, Space, Time, Failure
- Circuit Breakers
- Back –Pressure
- High Availability
- Eventual Consistency

- *CAP Theorem

Reactive Streams

- Specifications 1.0
- Working groups started in 2013: Netflix, Pivotal, LightBend – later joined by Oracle, Twitter, Red Hat, spray.io
- Standard for asynchronous stream processing with non-blocking back pressure
- Initial release: May 2015
- Latest release: August 2017

What about Microservices?

How are Microservices related to Reactive Programming, and also Reactive Systems?

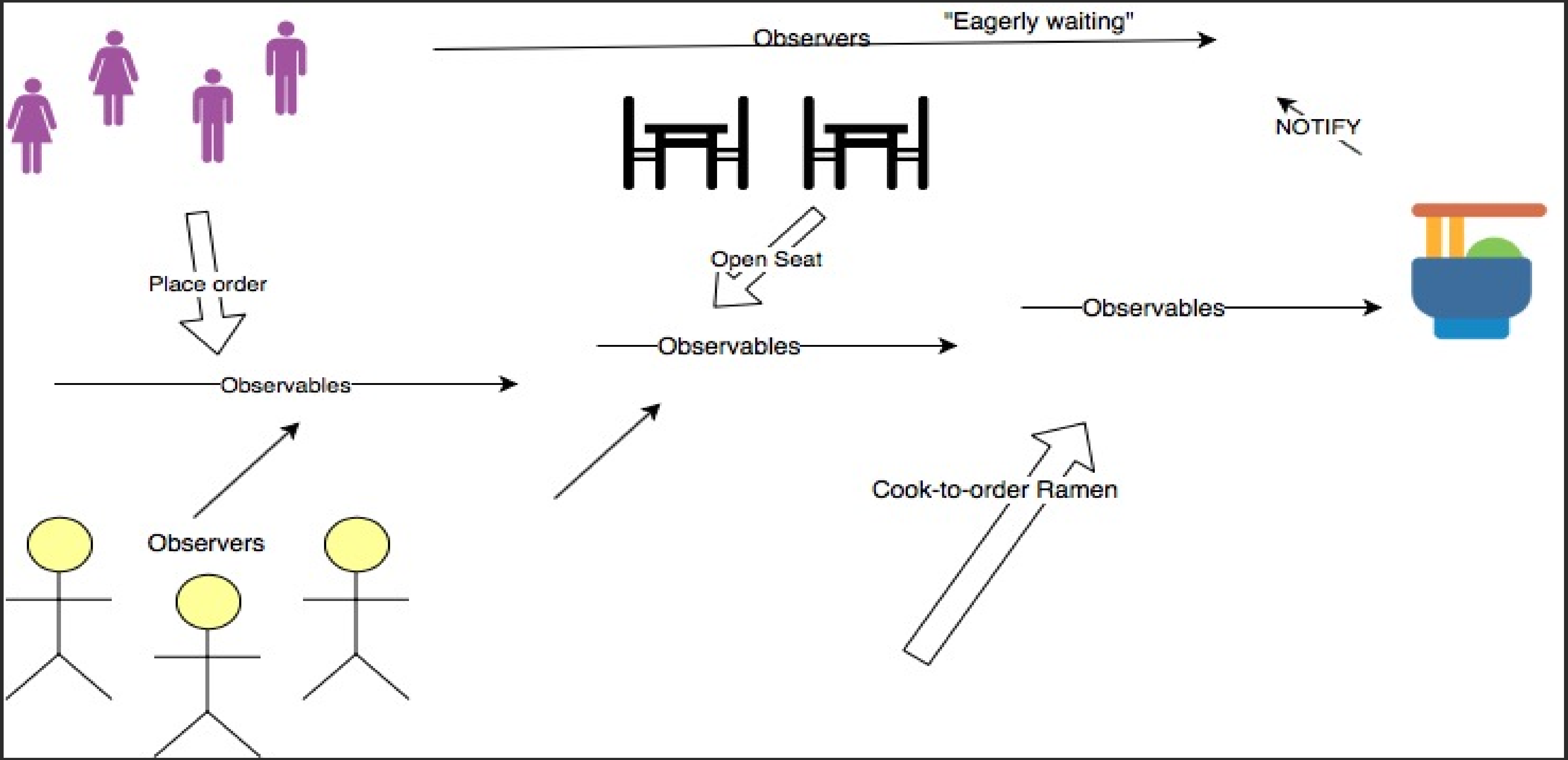
Lagom – A Reactive Microservices Framework

- From LightBend
- Built on the Play Framework and Akka Cluster
- RPC-style programming style
- Message Broker API implemented in Akka Streams and Kafka
- Persistence based on the concept of Entities – Domain-Driven Design
  - Event-sourcing
  - CQRS

1000円
1000円
850円
武蔵ら～麺
1000円
1200円
1300円

55872
12.-7.23
武蔵つけ麺
¥1000-
武蔵ら～麺
□ あっさり
□ こってり
並
大
¥1000-
20:37

"Reactive" Design Thinking



RxJava

* Java implementation of ReactiveX (reactivex.io)
* Very popular, especially on Android.
* Also available in Javascript, .NET, Scala, Clojure, Swift (and more!) with equivalent api.
* v1 (pre reactive streams specification) 2014
* v2 2016 (Flowable, with backpressure)

Simple example: Hello World

```
public static void hello(String... args) {
        Flowable.fromArray(args).subscribe(s -> System.out.println("Hello " + s + "!"));
}
```

```java
package com.ibm.reactive.samples;

import io.reactivex.Observable;
import io.reactivex.functions.Consumer;

public class RxJava2Example
{
    public static void main(String[] args)
    {
        //producer
        Observable<String> observable = Observable.just("how", "to", "do", "in", "java");

        //consumer
        Consumer<? super String> consumer = System.out::println;

        //Attaching producer to consumer
        observable.subscribe(consumer);
    }
}
```

Spring Reactor

* Based on Project Reactor from Pivotal
* Very similar api to RxJava2
* Built for Java 8+ allowing for cleaner interface.
* David Karnok (project lead of RxJava, engine contributor to Reactor)

David Karnok@akarnokd
Use Reactor 3 if you are allowed to use Java 8+, use RxJava 2 if you are stuck on Java 6+ or need your functions to throw checked exceptions
(Twitter: Sep 10th, 2016)

* Reactor Core
* Reactor Test
* Reactor Extra
* Reactor Netty
* Reactor Adapter
* Reactor Kafka
* Reactor RabbitMQ
* Incubating reactive streams foundation for .Net, Javascript

Spring 5: Spring Web Reactive (non-blocking web stack)

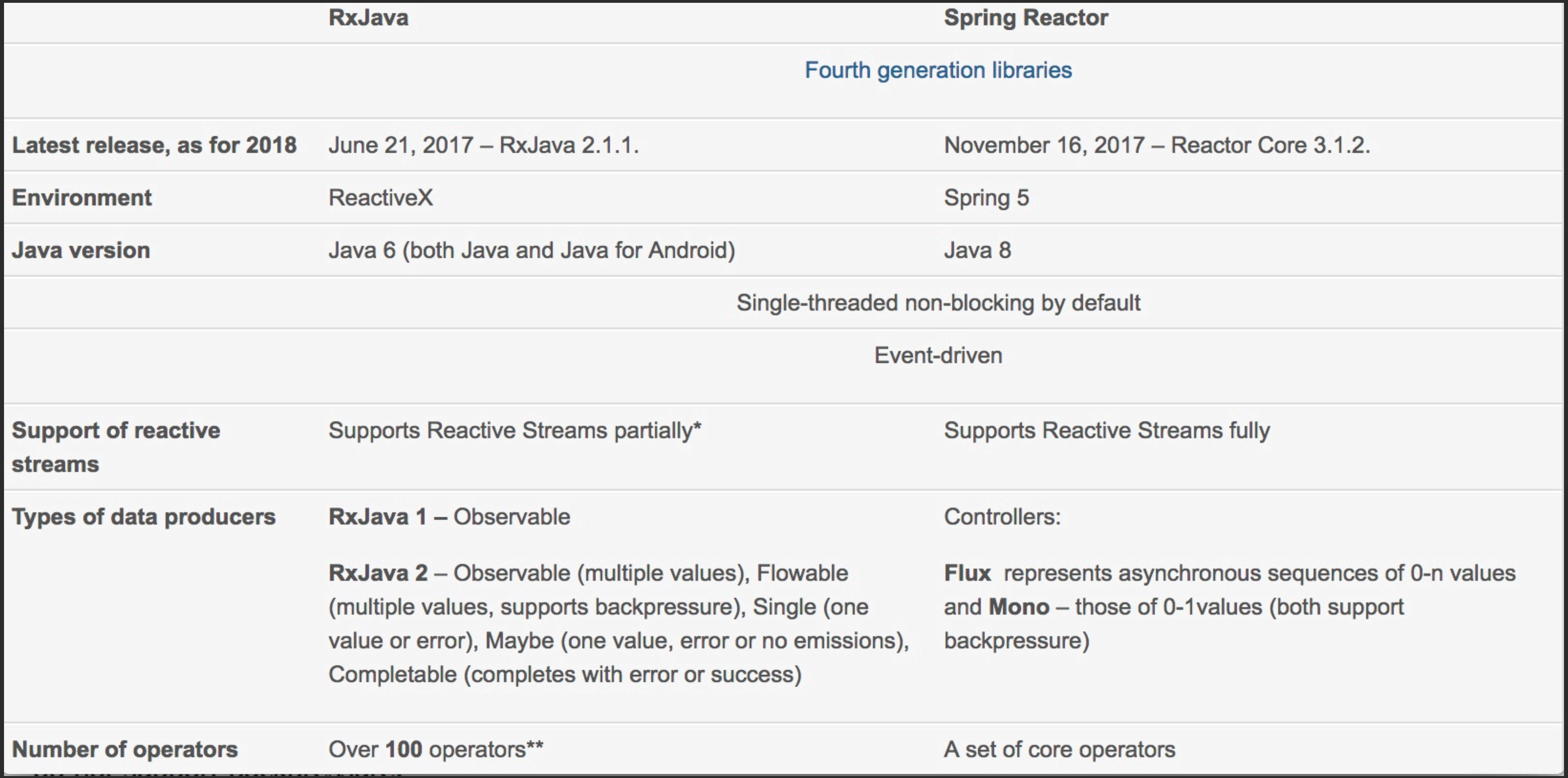
Traditional Approach (Spring MVC):

```
@GetMapping("/traditional")
public List < Product > getAllProducts() {
        System.out.println("Traditional way started");
        List < Product > products = prodService.getProducts("traditional");
        System.out.println("Traditional way completed");
        return products;
}
```

Reactive Approach (Spring Web Reactive – Web Flux):

```
@GetMapping(value = "/reactive", .TEXT_EVENT_STREAM_VALUE)
public Flux < Product > getAll() {
        System.out.println("Reactive way using Flux started");
        Flux < Product > fluxProducts = prodService.getProductsStream("Flux");
        System.out.println("Reactive way using Flux completed");
        return fluxProducts;
}
```

Quick comparison: RxJava vs Spring Reactor



	RxJava	Spring Reactor
	Fourth generation libraries	
Latest release, as for 2018	June 21, 2017 – RxJava 2.1.1.	November 16, 2017 – Reactor Core 3.1.2.
Environment	ReactiveX	Spring 5
Java version	Java 6 (both Java and Java for Android)	Java 8
	Single-threaded non-blocking by default	
	Event-driven	
Support of reactive streams	Supports Reactive Streams partially*	Supports Reactive Streams fully
Types of data producers	**RxJava 1 –** Observable **RxJava 2** – Observable (multiple values), Flowable (multiple values, supports backpressure), Single (one value or error), Maybe (one value, error or no emissions), Completable (completes with error or success)	Controllers: **Flux** represents asynchronous sequences of 0-n values and **Mono** – those of 0-1values (both support backpressure)
Number of operators	Over **100** operators**	A set of core operators

Akka

* From LightBend. - IBM Partnership (June 2017)
    https://www.lightbend.com/ibm-alliance
* Actor Model (from Erlang – Actor Programming Model in the 1980’s)
* Event-Driven
* Location Transparency
* Lightweight

* Resiliency/Recoverability – Supervisor capability

```
package sample.hello;

public class Main {

  public static void main(String[] args) {
    akka.Main.main(new String[] { HelloWorld.class.getName() });
  }
}
```

```
package sample.hello;

import akka.actor.AbstractActor;
import akka.actor.ActorRef;
import akka.actor.Props;

import static sample.hello.Greeter.Msg;

public class HelloWorld extends AbstractActor {

  @Override
  public Receive createReceive() {
    return receiveBuilder()
      .matchEquals(Msg.DONE, m -> {
        // when the greeter is done, stop this actor and with it the application
        getContext().stop(self());
      })
      .build();
  }

  @Override
  public void preStart() {
    // create the greeter actor
    final ActorRef greeter = getContext().actorOf(Props.create(Greeter.class), "greeter");
    // tell it to perform the greeting
    greeter.tell(Msg.GREET, self());
  }
}
```

```java
package sample.hello;

import akka.actor.AbstractActor;

public class Greeter extends AbstractActor {

  public static enum Msg {
    GREET, DONE;
  }

  @Override
  public Receive createReceive() {
    return receiveBuilder()
      .matchEquals(Msg.GREET, m -> {
        System.out.println("Hello World!");
        sender().tell(Msg.DONE, self());
      })
      .build();
  }
}
```

```
import akka.actor.{Actor, ActorSystem, Props}

        //greet message
        case class Greet(name: String)
        //greeter Actor

         class Greeter extends Actor {
             def receive = {
                 case Greet(name) => println(s"Hello $name")
             }

}

object HelloAkka extends App {
        val system=ActorSystem("Intro-Akka")
        val greeter=system.actorOf(Props[Greeter],"greeter")
        greeter ! Greet("Akka")
}
```

IBM **Developer**

Vert.x

* From Eclipse
* A very flexible “polyglot” framework that interoperates with other frameworks and tools.
  - -> RxJava
  - -> Spring Reactor
  - -> Akka
  - -> not to mention the other non-Java frameworks and tools as well (JS, .Net…)
* Verticles – components that are being deployed and executed by Vert.x
  - -> event-driven
  - -> run only when they receive a message
* Vert.x event bus

* Not restrictively tied to any container – Vert.x libraries can be used with other libraries

```
package io.vertx.example;

import io.vertx.core.Vertx;


public class HelloWorldEmbedded {

  public static void main(String[] args) {

        // Create an HTTP server which simply returns "Hello World!" to each request.
        Vertx.vertx().createHttpServer().requestHandler(req -> req.response().end("Hello
        World!")).listen(8080);

  }
```

Recap & Takeaways

* Reactive is an overloaded word in today’s market
* Not for the ”faint of heart” but for the determined
* Reactive programming is not the same as Functional Reactive programming or Reactive systems
* Benefits of being “Reactive” on the programming level
* Reactive systems and architecture bring “reactivity” to another level
* Reactivity to this day has not been fully ready on the database level, despite some significant efforts on the database connectivity level

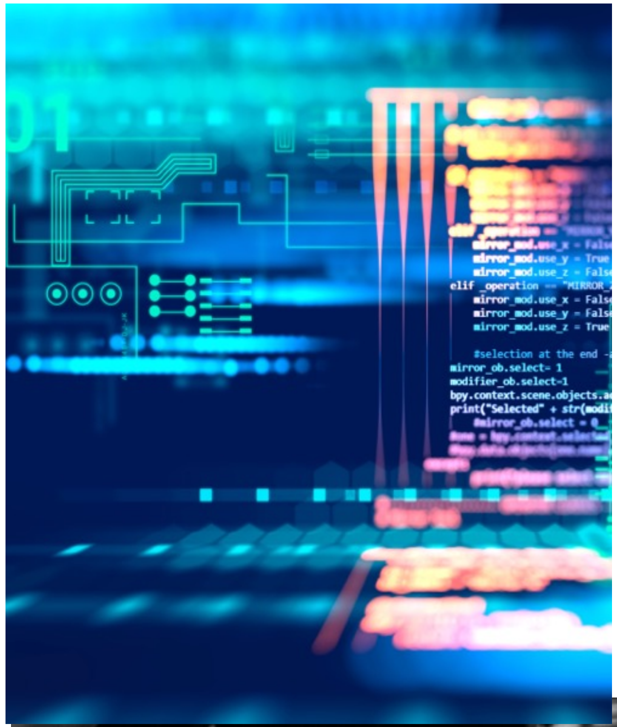
Thank you

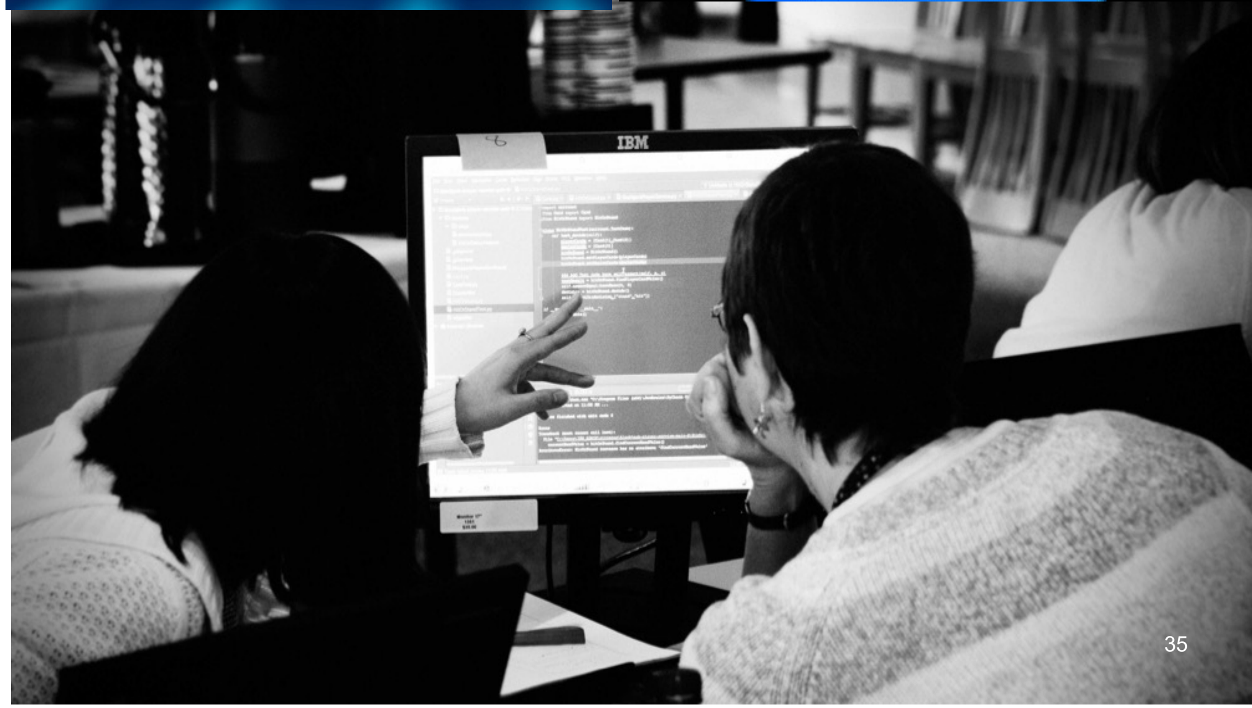
twitter.com/mgrygles

github.com/mgrygles

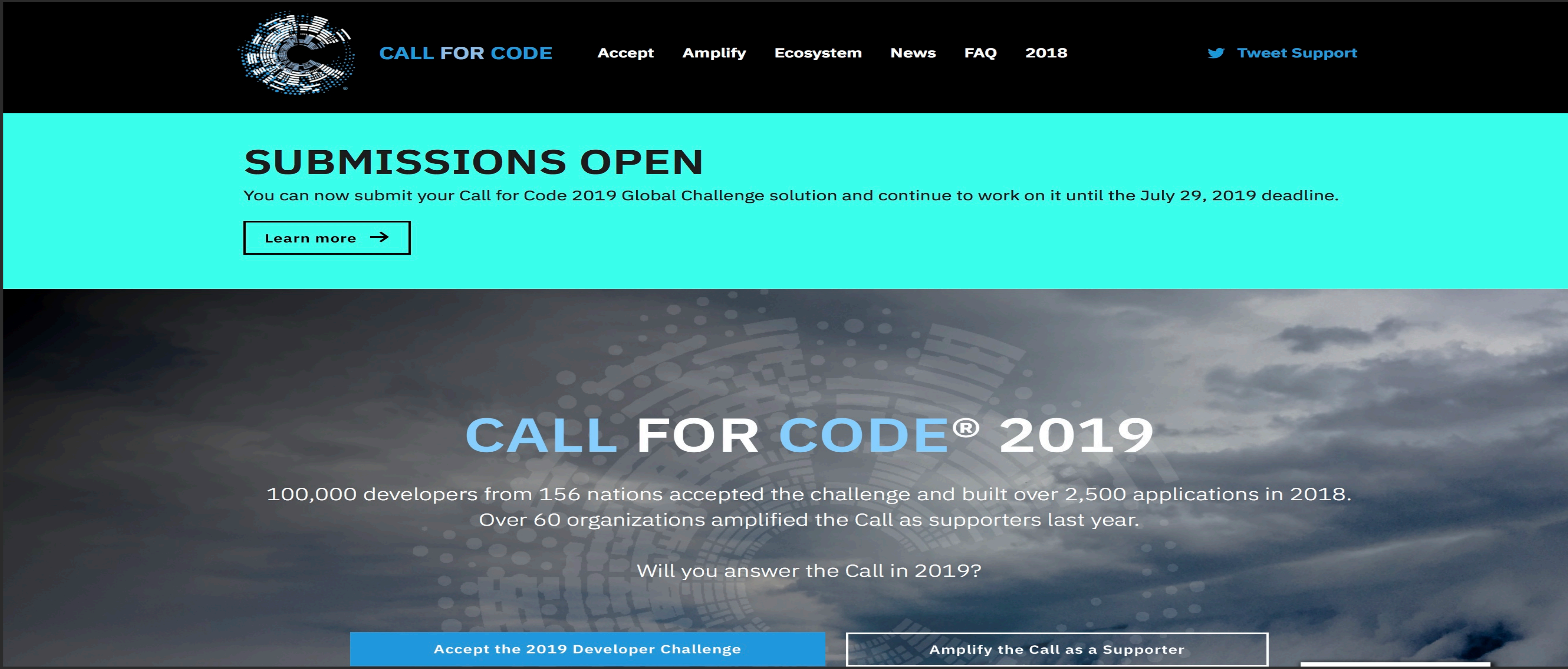
developer.ibm.com/profiles/mary.grygleski

IBM Cloud:
https://ibm.biz/BdzBiQ
https://developer.ibm.com/technologies/reactive-systems/
https://www.lightbend.com/ibm-alliance
https://www.reactivemanifesto.org
https://www.reactivedesignpatterns.com





IBM's Multi-Year Initiative for a Good Cause
http://callforcode.org



SUBMISSIONS OPEN

You can now submit your Call for Code 2019 Global Challenge solution and continue to work on it until the July 29, 2019 deadline.

CALL FOR CODE® 2019

100,000 developers from 156 nations accepted the challenge and built over 2,500 applications in 2018.
Over 60 organizations amplified the Call as supporters last year.

Will you answer the Call in 2019?

Accept the 2019 Developer Challenge
Amplify the Call as a Supporter

IBM®
M