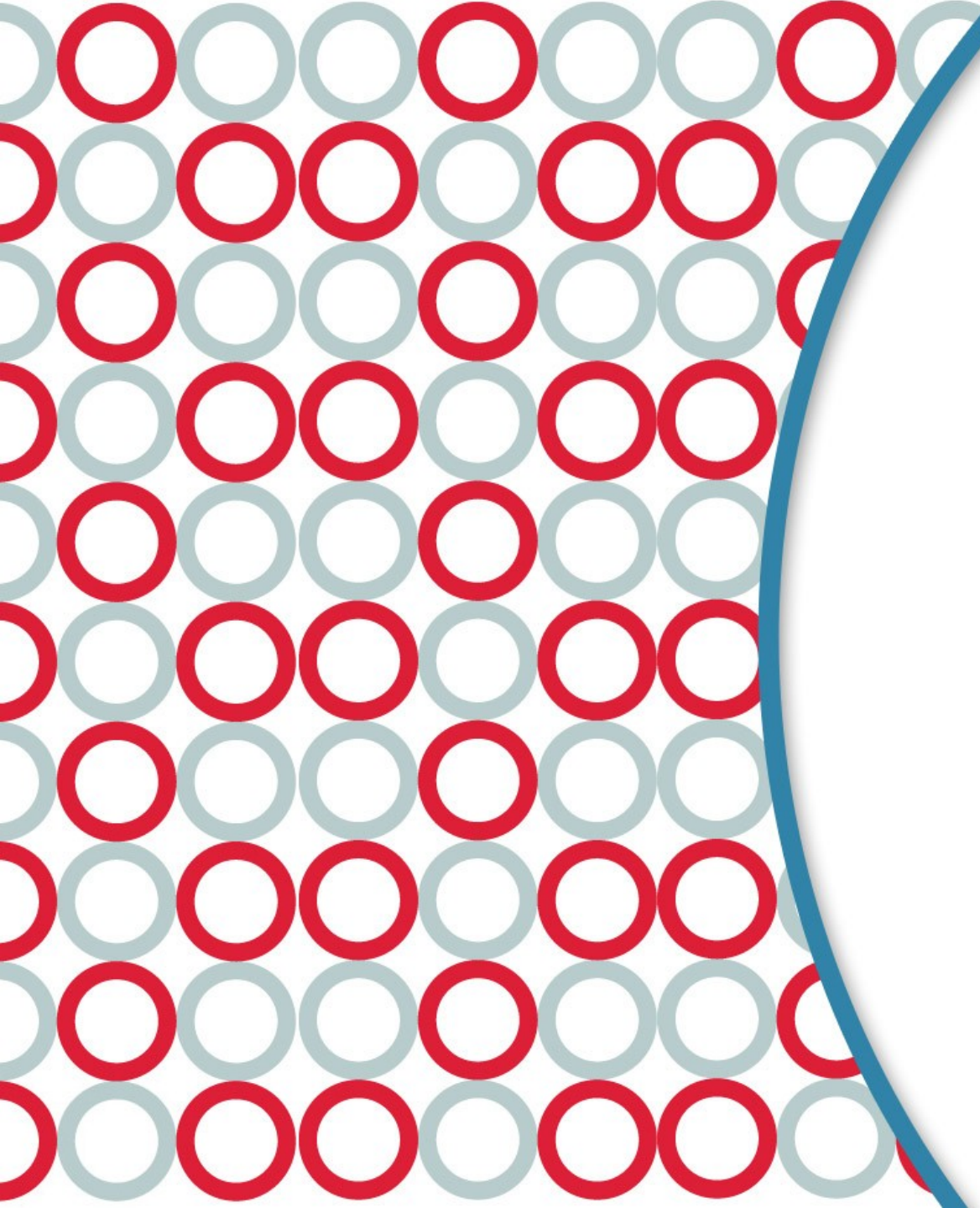


Oct 9, 2020
IPT – Intellectual Products & Technologies

Reactive Microservices with Spring 5 WebFlux

Trayan Iliev
tiliev@iproduct.org
http://iproduct.org

Reactive Microservices with Spring 5 WebFlux

- Introduction to FRP, Reactive Streams spec
- Project Reactor
- REST services with Spring 5: WebFlux
- Router, handler and filter functions
- Reactive repositories and reactive database access with Spring Data. Building
- End-to-end non-blocking reactive SOA with Netty
- Reactive WebClients and integration testing
- Realtime event streaming to JS clients using SSE

Reactive Microservices with Spring 5 WebFlux

- Introduction to FRP, Reactive Streams spec
- Project Reactor
- REST services with Spring 5: WebFlux
- Router, handler and filter functions
- Reactive repositories and reactive database access with Spring Data. Building
- End-to-end non-blocking reactive SOA with Netty
- Reactive WebClients and integration testing
- Realtime event streaming to JS clients using SSE

Best Explained in Code

About me

Trayan Iliev

– CEO of IPT – Intellectual Products & Technologies

– Oracle® certified programmer 15+ Y

– end-to-end reactive fullstack apps with Java, ES6/7, TypeScript, Angular, React and Vue.js

– 12+ years IT trainer

– Voxxed Days, jPrime, jProfessionals, BGOUG, BGJUG, DEV.BG speaker

– Organizer RoboLearn hackathons and IoT enthusiast (http://robolearn.org)

Since 2003: IT Education Evolved

- Spring, Java SE/Web/EE 7/8/9, JSF
- Reactive IoT with Reactor / RxJava / RxJS
- Node.js + Express + React + Redux + GraphQL
- Angular + TypeScript + Redux (ngrx)
- SOA & REST HATEOAS
- DDD & Reactive Microservices

Where to Find the Demo Code?

Spring 5 WebFlux demos available @ GitHub:

https://github.com/iproduct/spring-5-webflux

Data / Event / Message Streams

"Conceptually, a stream is a (potentially never-ending) flow of data records, and a transformation is an operation that takes one or more streams as input, and produces one or more output streams as a result."

Apache Flink: Dataflow Programming Model

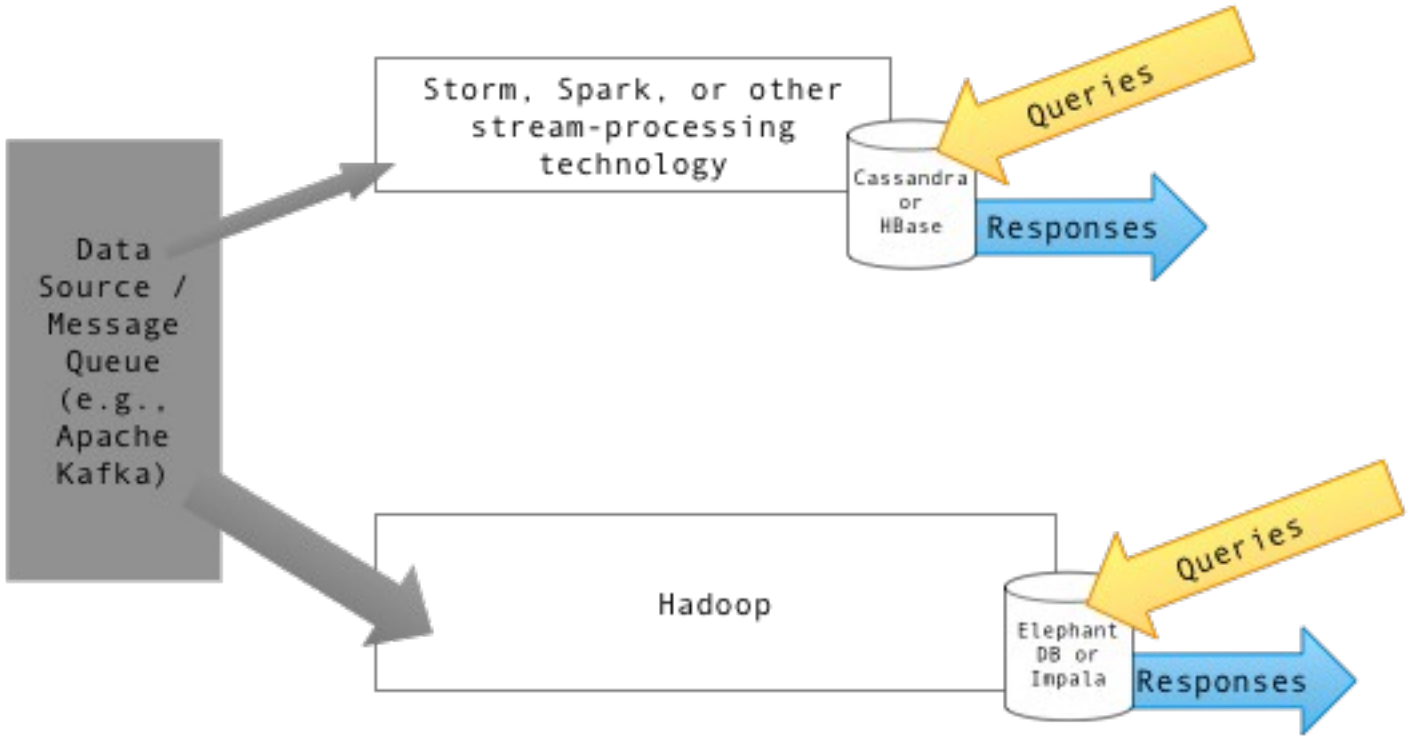
Data Stream Programming

The idea of abstracting logic from execution is hardly new -- it was the dream of SOA. And the recent emergence of microservices and containers shows that the dream still lives on.

For developers, the question is whether they want to learn yet one more layer of abstraction to their coding. On one hand, there's the elusive promise of a common API to streaming engines that in theory should let you mix and match, or swap in and swap out.

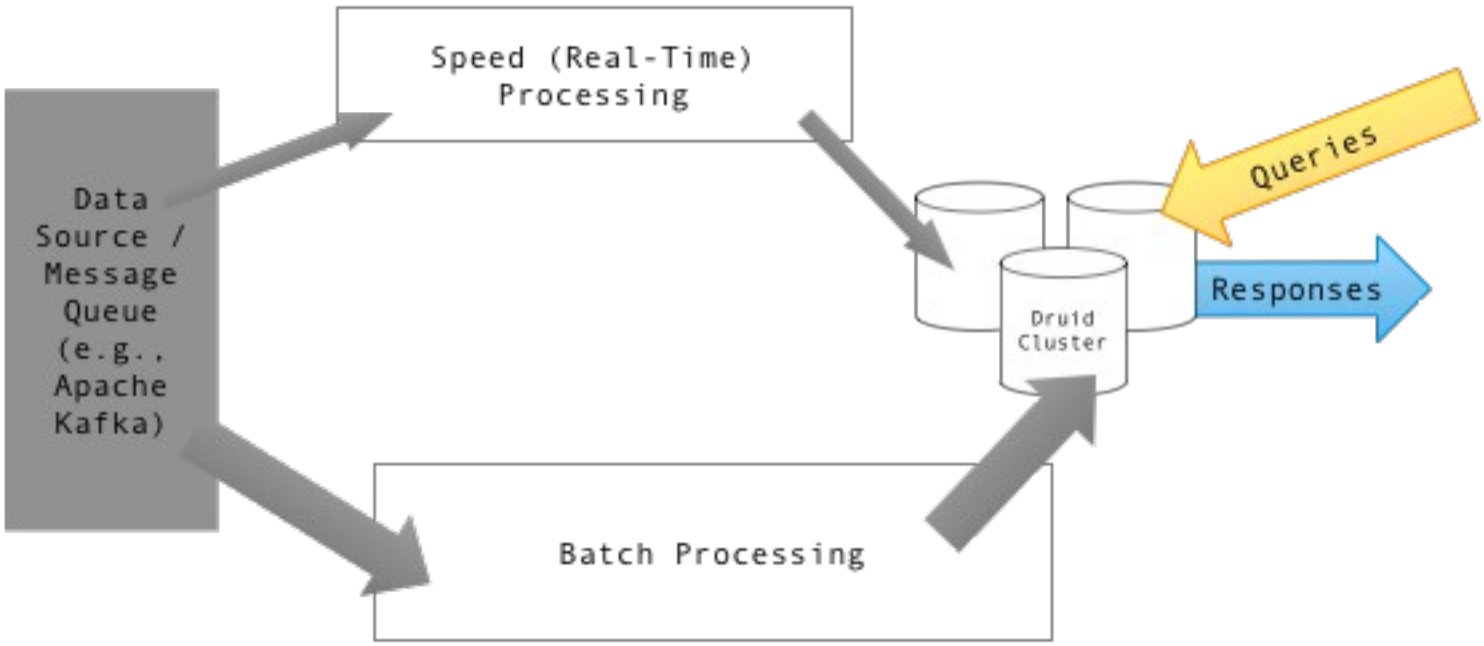
Tony Baer (Ovum) @ ZDNet - Apache Beam and Spark: New coopetition for squashing the Lambda Architecture?

Lambda Architecture - I



https://commons.wikimedia.org/w/index.php?curid=34963986, By Textractor - Own work, CC BY-SA 4

Lambda Architecture - II

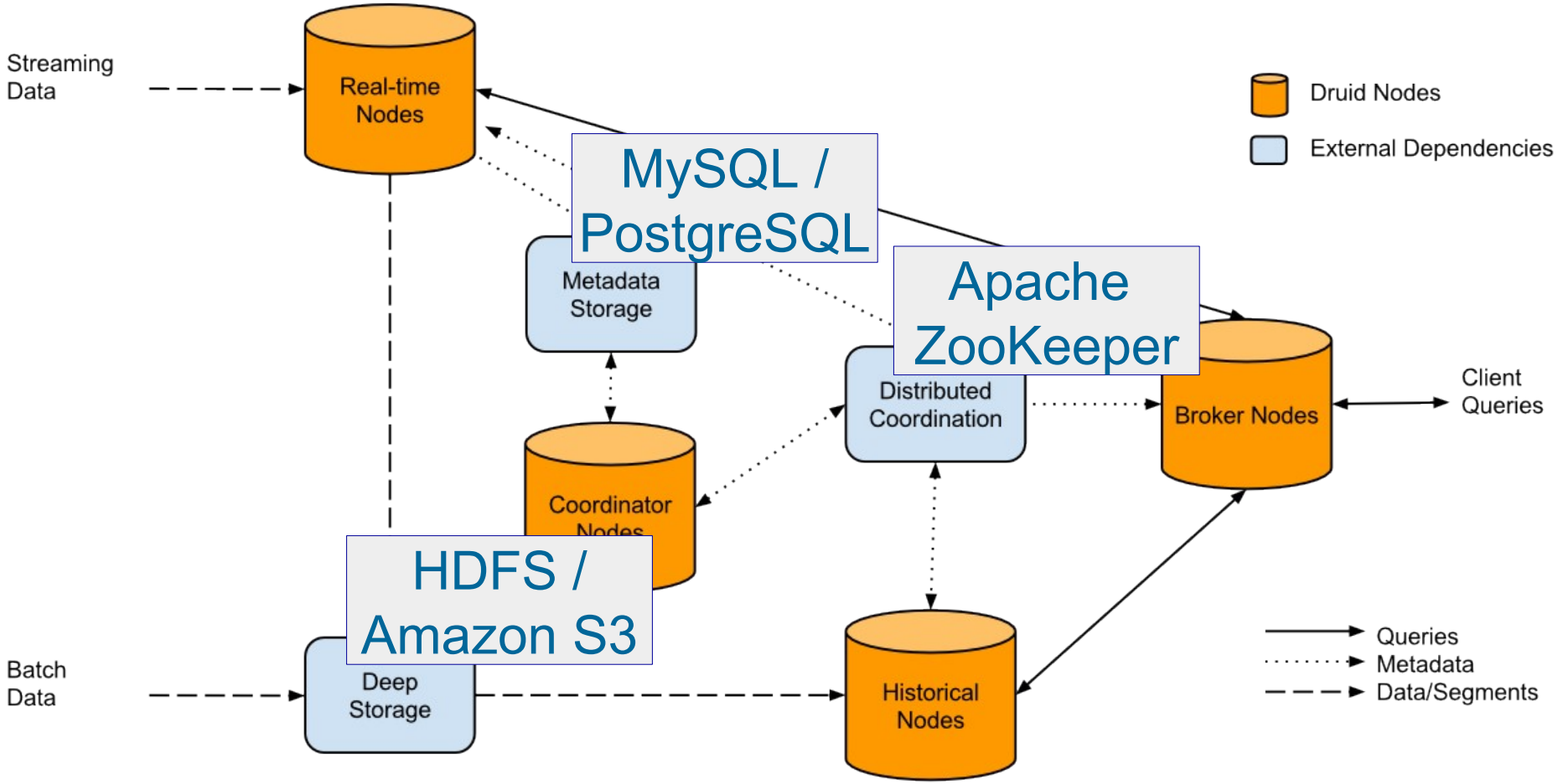


https://commons.wikimedia.org/w/index.php?curid=34963987, By Textractor - Own work, CC BY-SA 4

Lambda Architecture - III

- Data-processing architecture designed to handle massive quantities of data by using both *batch*- and *stream-processing* methods
- Balances *latency, throughput, fault-tolerance, big data, real-time analytics*, mitigates the latencies of map-reduce
- Data model with an append-only, *immutable data* source that serves as a system of record
- Ingesting and processing *timestamped events* that are appended to existing events. State is determined from the *natural time-based ordering* of the data.

Druid Distributed Data Store (Java)



https://commons.wikimedia.org/w/index.php?curid=33899448 By Fangjin Yang - sent to me personally, GFDL

Lambda Architecture: Projects - I

- Apache Spark is an open-source cluster-computing framework. Spark Streaming, Spark Mllib

- Apache Storm is a distributed stream processing – streams DAG

- Apache Apex™ unified stream and batch processing engine.

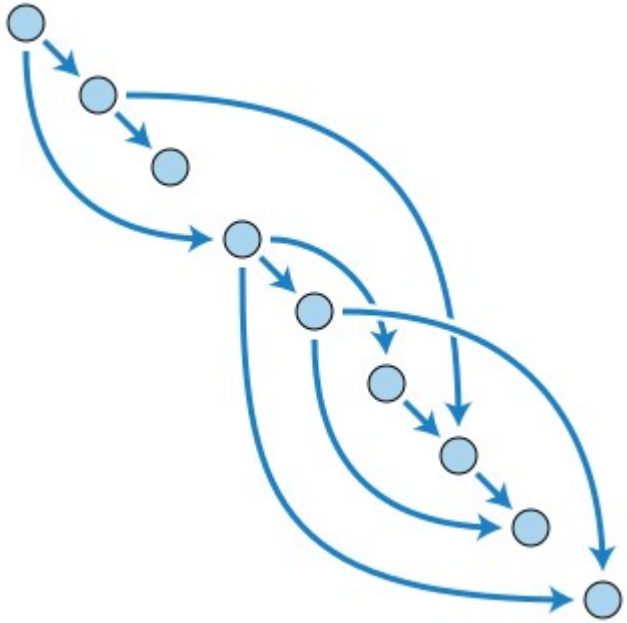
Lambda Architecture: Projects - II

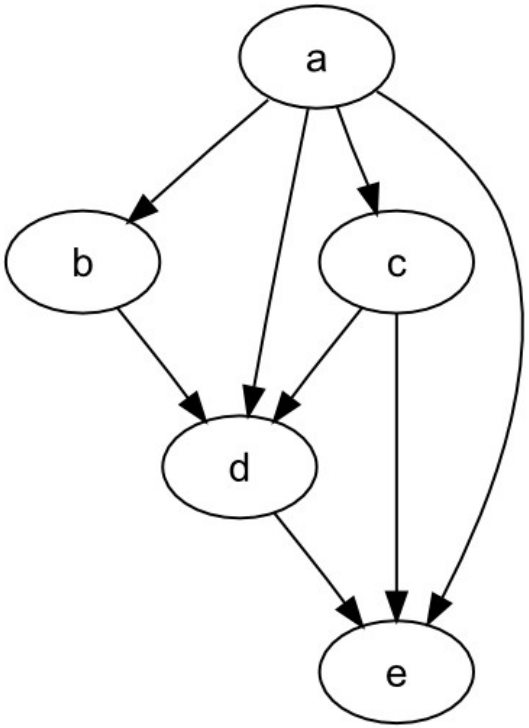
- Apache Flink - open source stream processing framework – Java, Scala

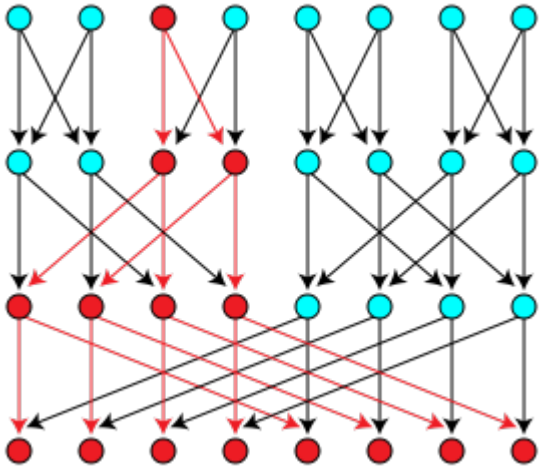
- Apache Kafka - open-source stream processing (Kafka Streams), real-time, low-latency, high-throughput, massively scalable pub/sub

- Apache Beam – unified batch and streaming, portable, extensible

Direct Acyclic Graphs - DAG

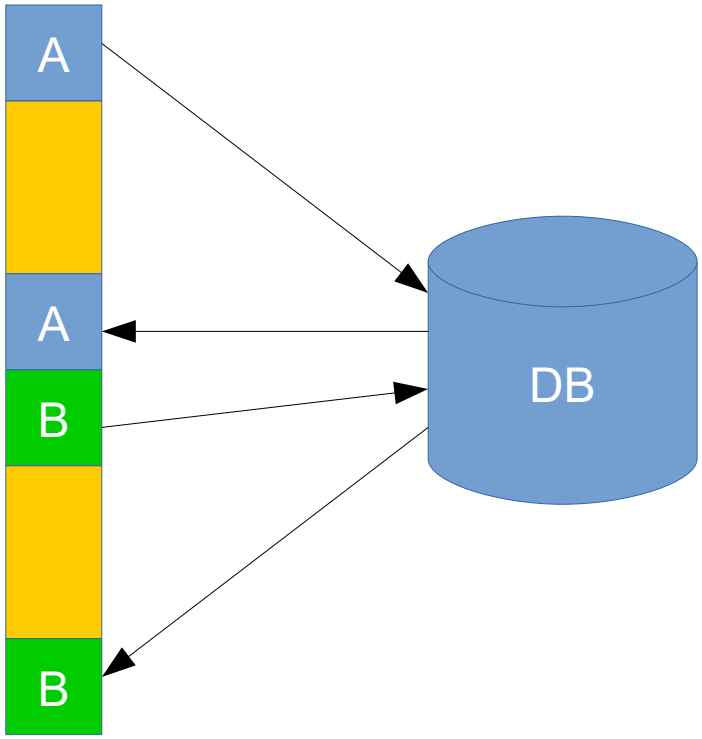




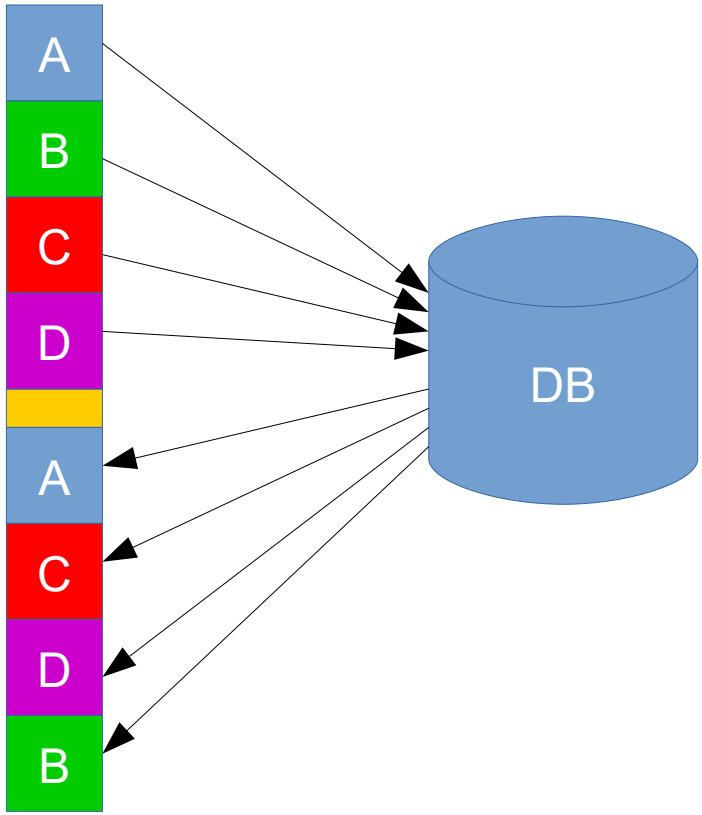


Synchronous vs. Asynchronous IO

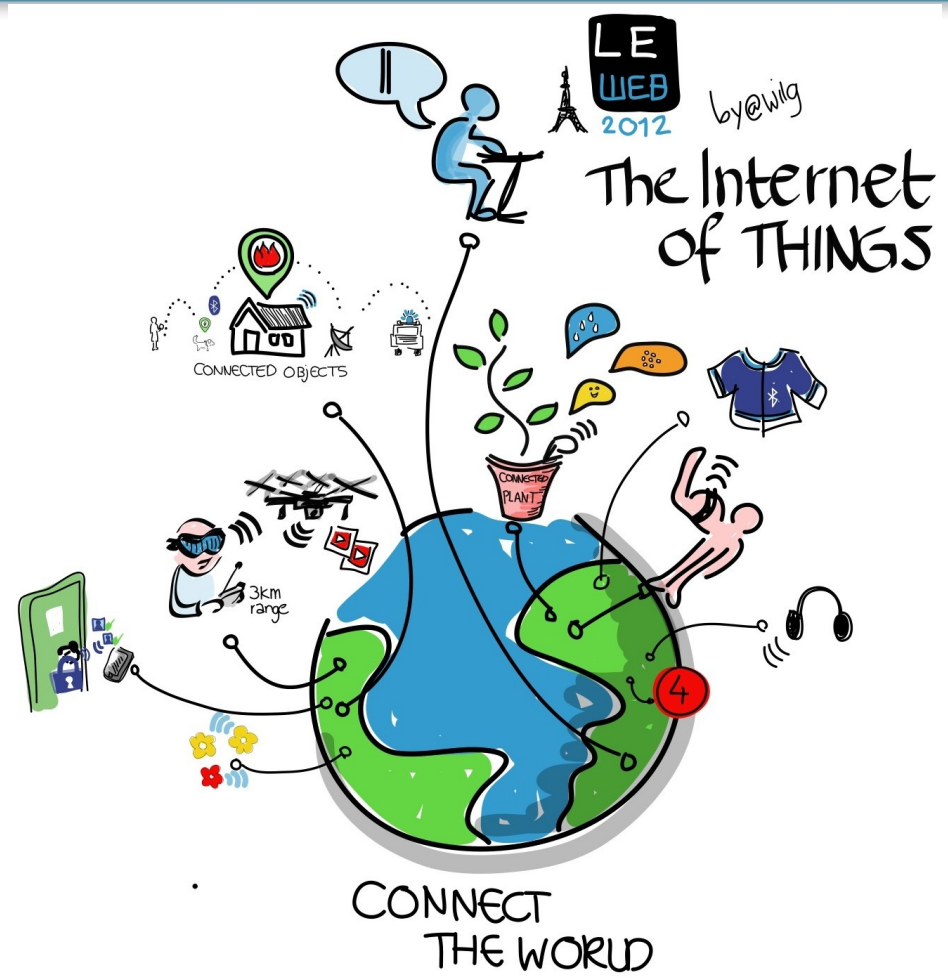
Synchronous



Asynchronous



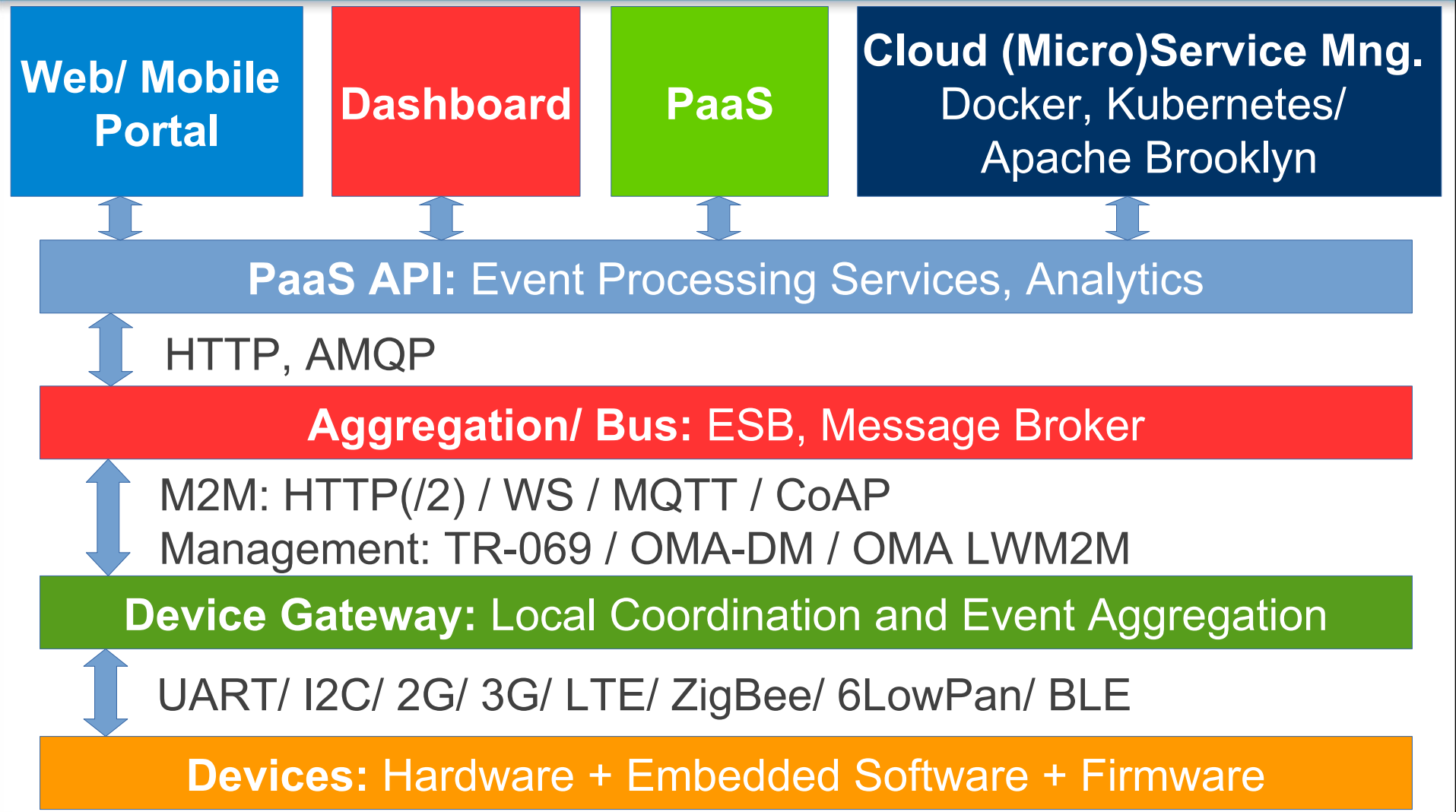
Example: Internet of Things (IoT)



CC BY 2.0, Source: https://www.flickr.com/photos/wilgengebroed/8249565455/

Radar, GPS, lidar for navigation and obstacle avoidance (2007 DARPA Urban Challenge)

IoT Services Architecture



What's High Performance?

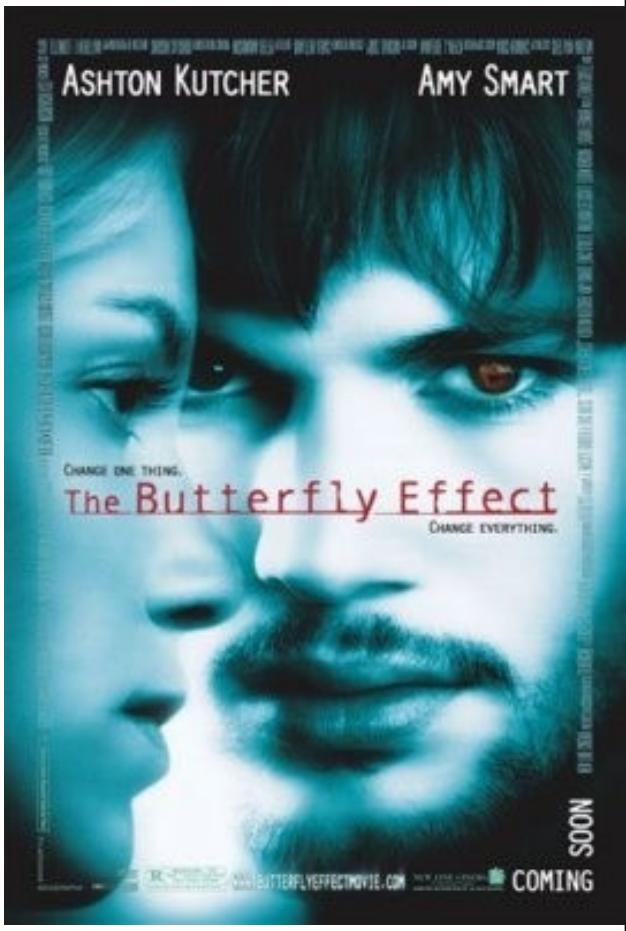
- **Performance** is about 2 things (Martin Thompson – http://www.infoq.com/articles/low-latency-vp):
  - **Throughput** – units per second, and
  - **Latency** – response time
- **Real-time** – time constraint from input to response regardless of system load.
- **Hard real-time system** if this constraint is not honored then a total system failure can occur.
- **Soft real-time system** – low latency response with little deviation in response time
- **100 nano-seconds** to **100 milli-seconds**. [Peter Lawrey]

Imperative and Reactive

We live in a Connected Universe

... there is hypothesis that all the things in the Universe are intimately connected, and you can not change a bit without changing all.

Action – Reaction principle is the essence of how Universe behaves.



Source: https://en.wikipedia.org/wiki/The_Butterfly_Effect#/media/File:Butterflyeffect_poster.jpg
Fair use, File:Butterflyeffect poster.jpg, Uploaded by Yaminator, Uploaded: 8 July 2008

Imperative and Reactive

- Reactive Programming: using static or dynamic data flows and propagation of change

Example: a := b + c

- Functional Programming: evaluation of mathematical functions,
  - Avoids changing-state and mutable data, declarative programming
  - Side effects free => much easier to understand and predict the program behavior.

Example: books.stream().filter(book -> book.getYear() > 2010)
.forEach(System.out::println)

Functional Reactive (FRP)

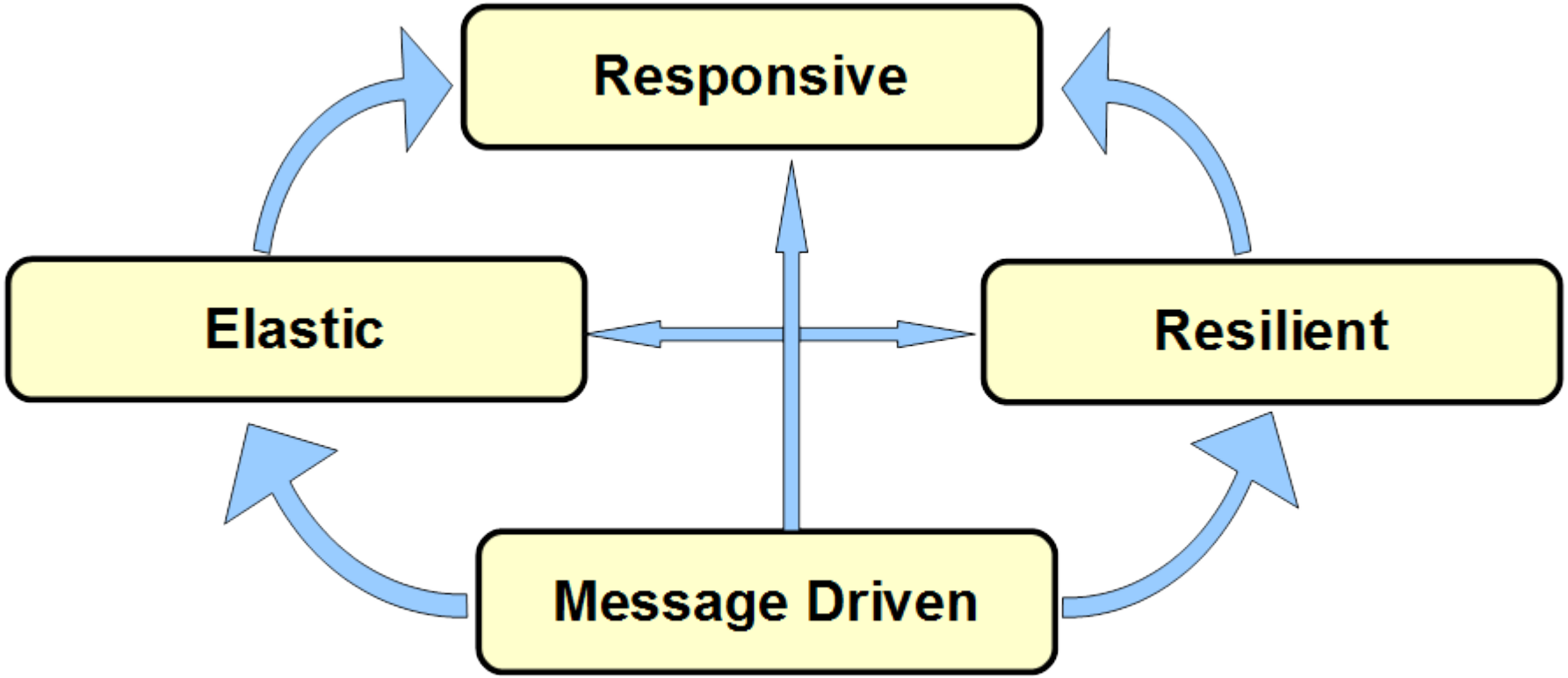
According to **Connal Elliot's** (ground-breaking paper @ Conference on Functional Programming, 1997), **FRP** is:

(a) Denotative

(b) Temporally continuous

Reactive Manifesto

[http://www.reactivemanifesto.org]



Scalable, Massively Concurrent

- **Message Driven** – asynchronous message-passing allows to establish a boundary between components that ensures loose coupling, isolation, location transparency, and provides the means to delegate errors as messages [Reactive Manifesto].
- The main idea is to separate concurrent producer and consumer workers by using **message queues.**
- **Message queues** can be **unbounded** or **bounded** (limited max number of messages)
- **Unbounded** message queues can present memory allocation problem in case the producers outrun the consumers for a long period → **OutOfMemoryError**

Reactive Programming

- Microsoft® opens source polyglot project ReactiveX (Reactive Extensions) [http://reactivex.io]:

Rx = Observables + LINQ + Schedulers :)

Java: RxJava, JavaScript: RxJS, C#: Rx.NET, Scala: RxScala, Clojure: RxClojure, C++: RxCpp, Ruby: Rx.rb, Python: RxPY, Groovy: RxGroovy, JRuby: RxJRuby, Kotlin: RxKotlin ...

- Reactive Streams Specification [http://www.reactive-streams.org/] used by:
- (Spring) Project Reactor [http://projectreactor.io/]
- Actor Model – Akka (Java, Scala) [http://akka.io/]

Reactive Streams Spec.

- Reactive Streams – provides standard for asynchronous stream processing with non-blocking back pressure.
- Minimal set of interfaces, methods and protocols for asynchronous data streams
- April 30, 2015: has been released version 1.0.0 of Reactive Streams for the JVM (Java API, Specification, TCK and implementation examples)
- Java 9: java.util.concurrent.Flow

Reactive Streams Spec.

- **Publisher** – provider of potentially unbounded number of sequenced elements, according to Subscriber(s) demand.

Publisher.subscribe(Subscriber) => **onSubscribe onNext* (onError | onComplete)?**

- **Subscriber** – calls **Subscription.request(long)** to receive notifications
- **Subscription** – one-to-one **Subscriber** ↔ **Publisher**, request data and cancel demand (allow cleanup).
- **Processor** = **Subscriber** + **Publisher**

FRP = Async Data Streams

- **FRP** is asynchronous data-flow programming using the building blocks of functional programming (e.g. map, reduce, filter) and explicitly modeling time
- Used for GUIs, robotics, and music. Example (RxJava):

```
Observable.from(
     new String[]{"Reactive", "Extensions", "Java"})
    .take(2).map(s -> s + " : on " + new Date())
    .subscribe(s -> System.out.println(s));
```

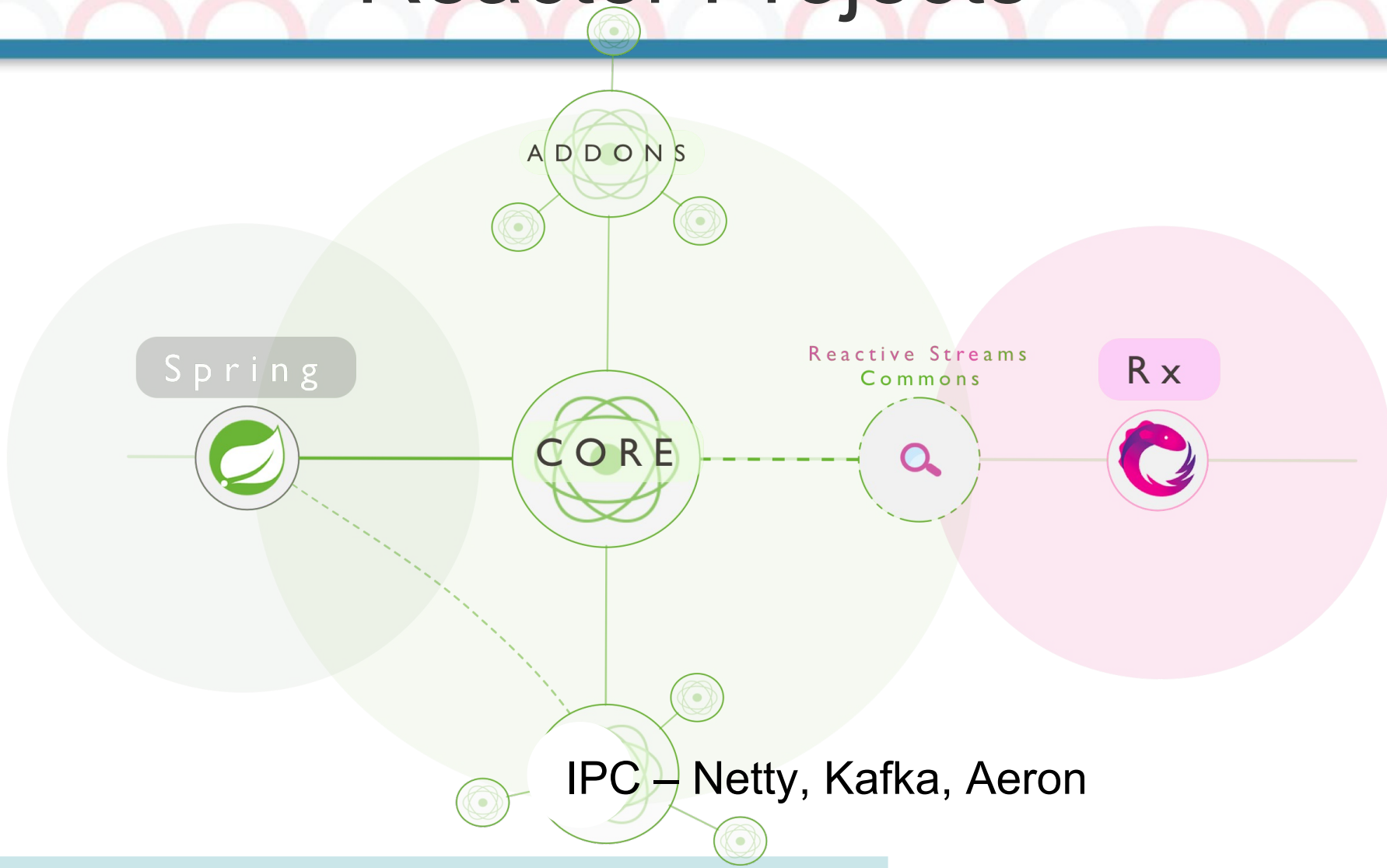
Result:

```
Reactive : on Wed Jun 17 21:54:02 GMT+02:00 2015
Extensions : on Wed Jun 17 21:54:02 GMT+02:00 2015
```

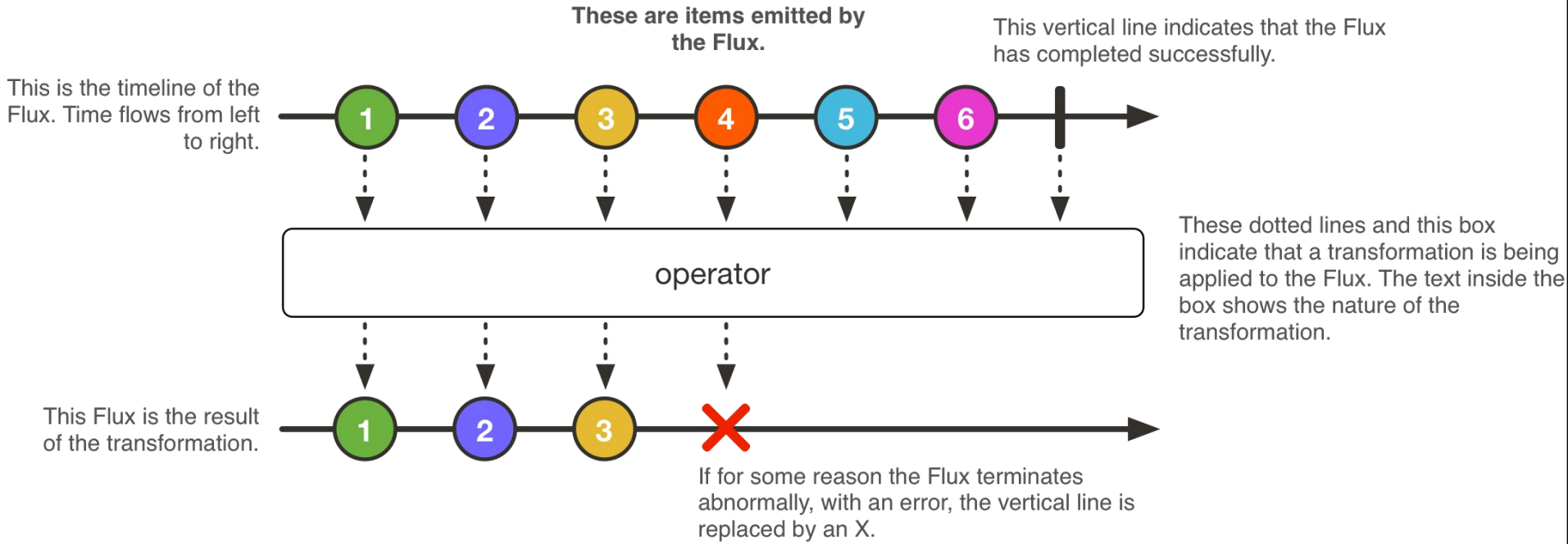
Project Reactor

- Reactor project allows building high-performance (low latency high throughput) non-blocking asynchronous applications on JVM.
- Reactor is designed to be extraordinarily fast and can sustain throughput rates on order of 10's of millions of operations per second.
- Reactor has powerful API for declaring data transformations and functional composition.
- Makes use of the concept of Mechanical Sympathy built on top of Disruptor / RingBuffer.

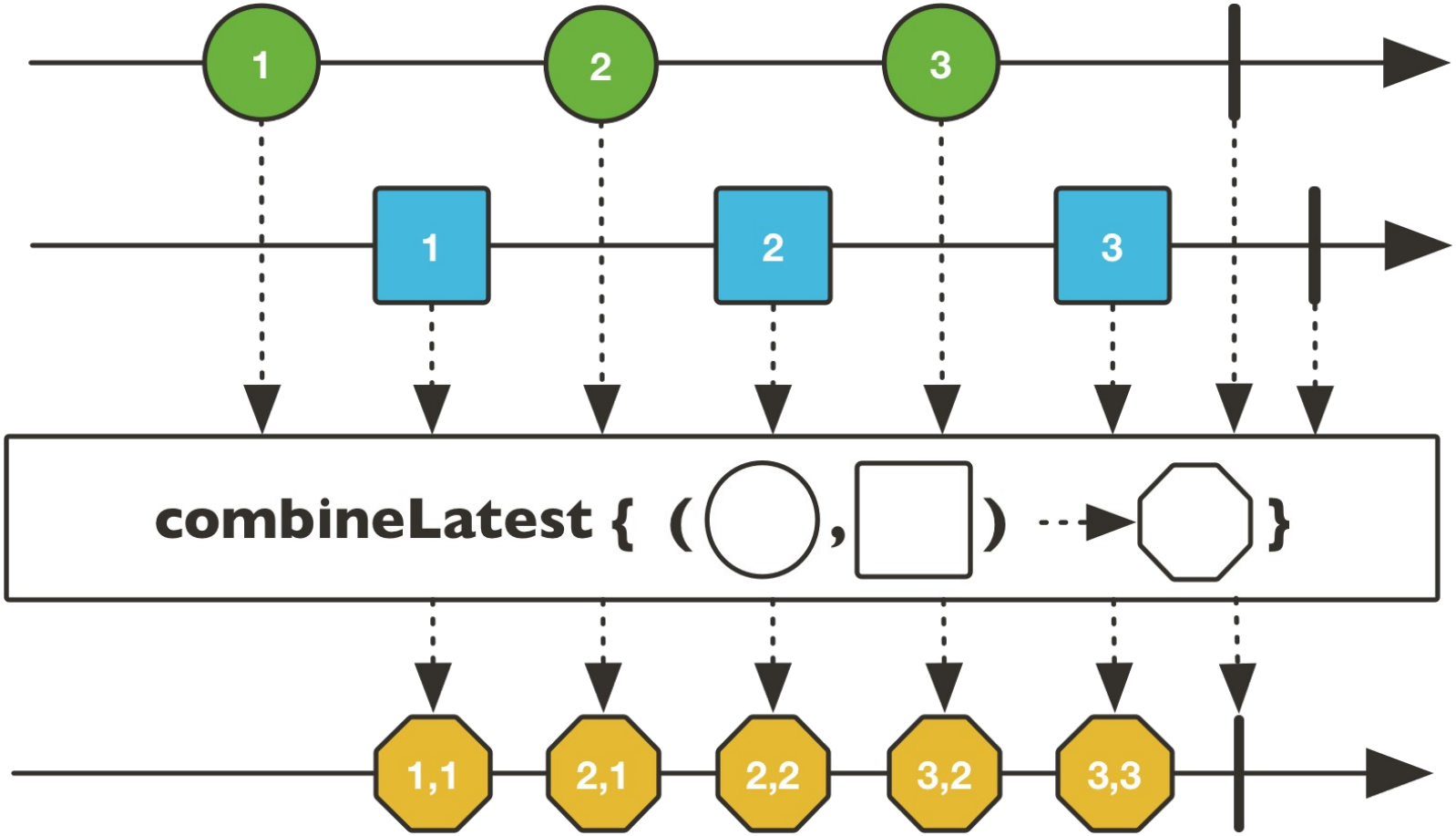
Reactor Projects



Reactor Flux

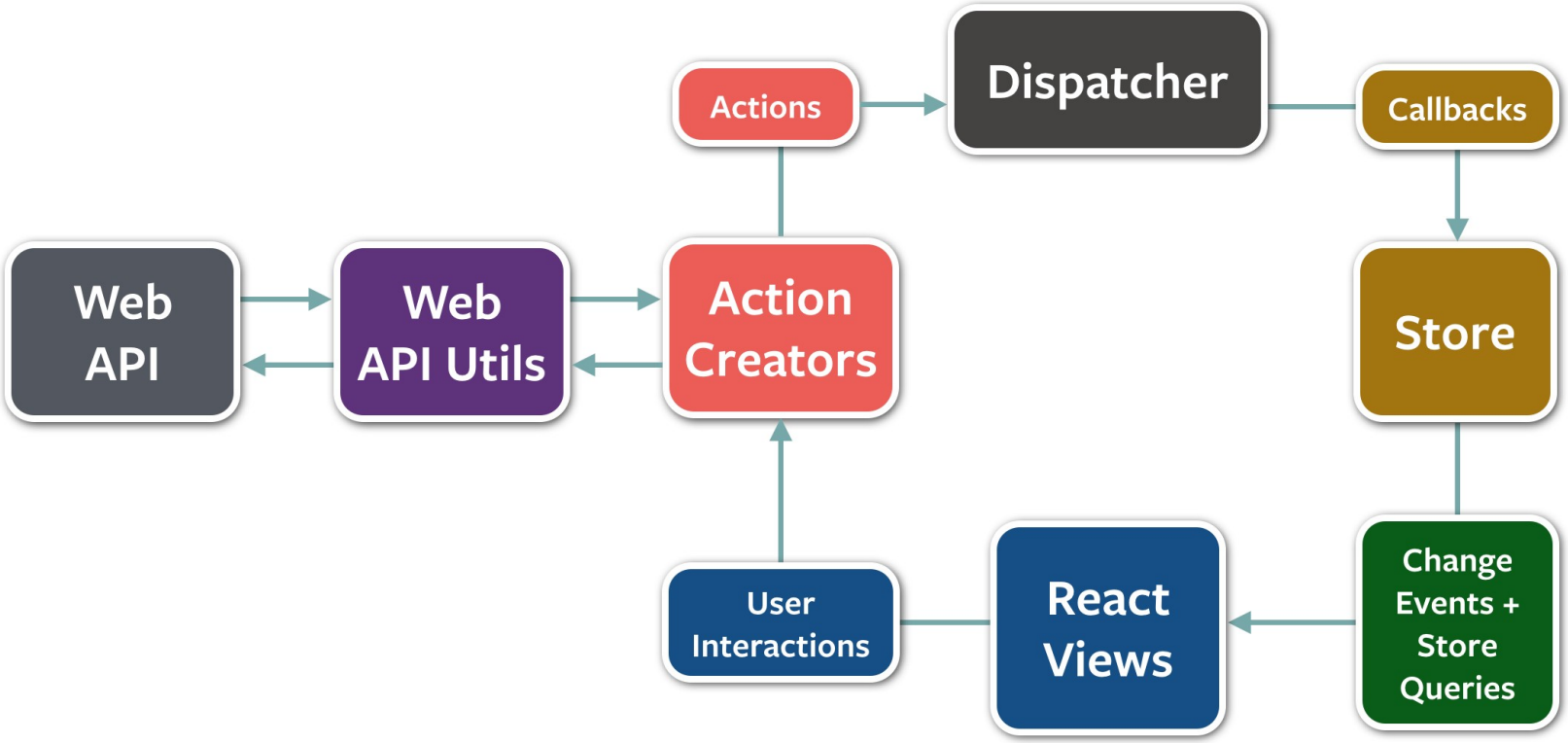


Example: Flux.combineLatest()



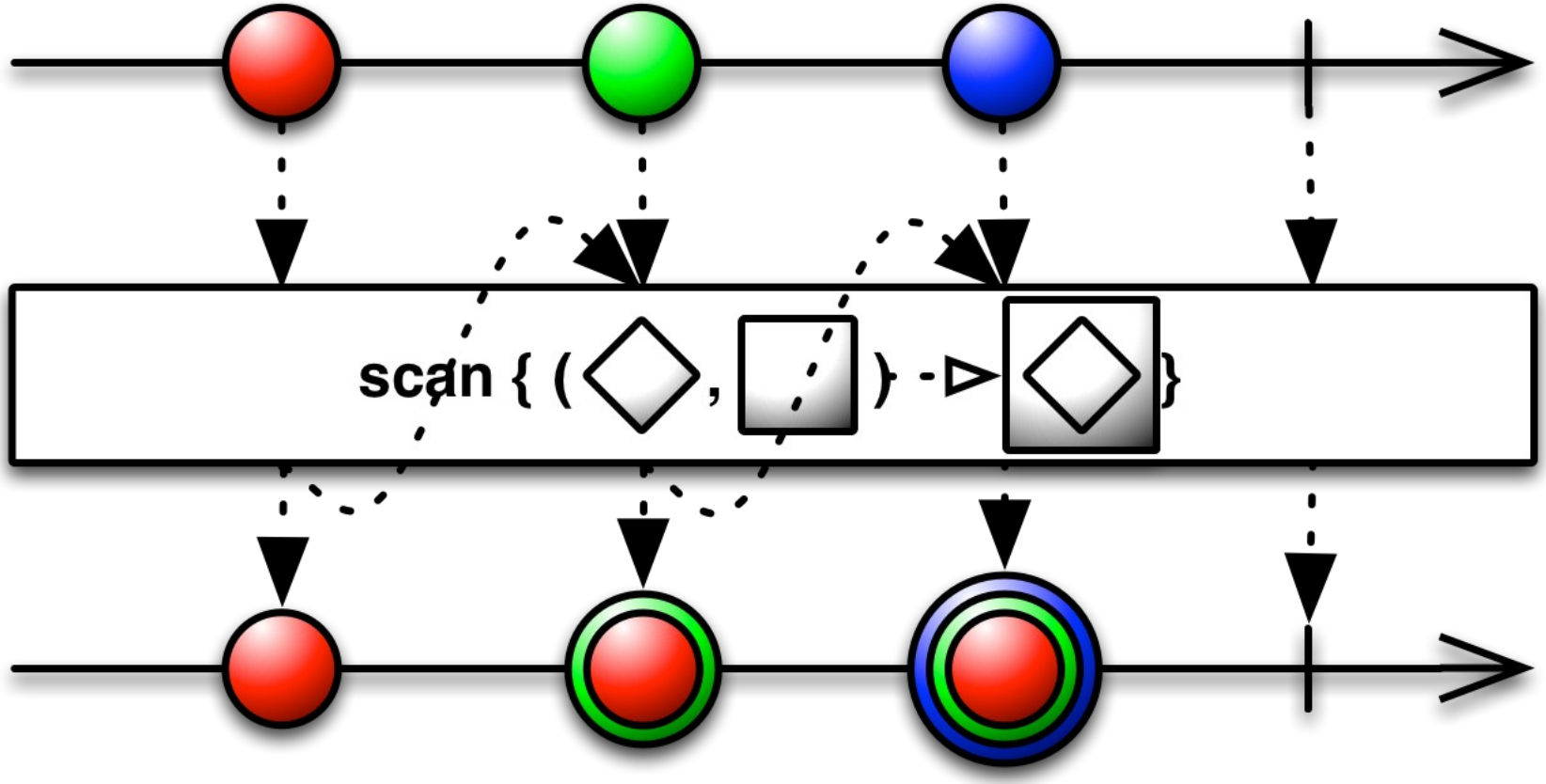
https://projectreactor.io/core/docs/api/, Apache Software License 2.0

Flux & Redux Design Patterns



Source: Flux in GitHub, https://github.com/facebook/flux, License: BSD 3-clause "New" License

Redux == Rx Scan Opearator

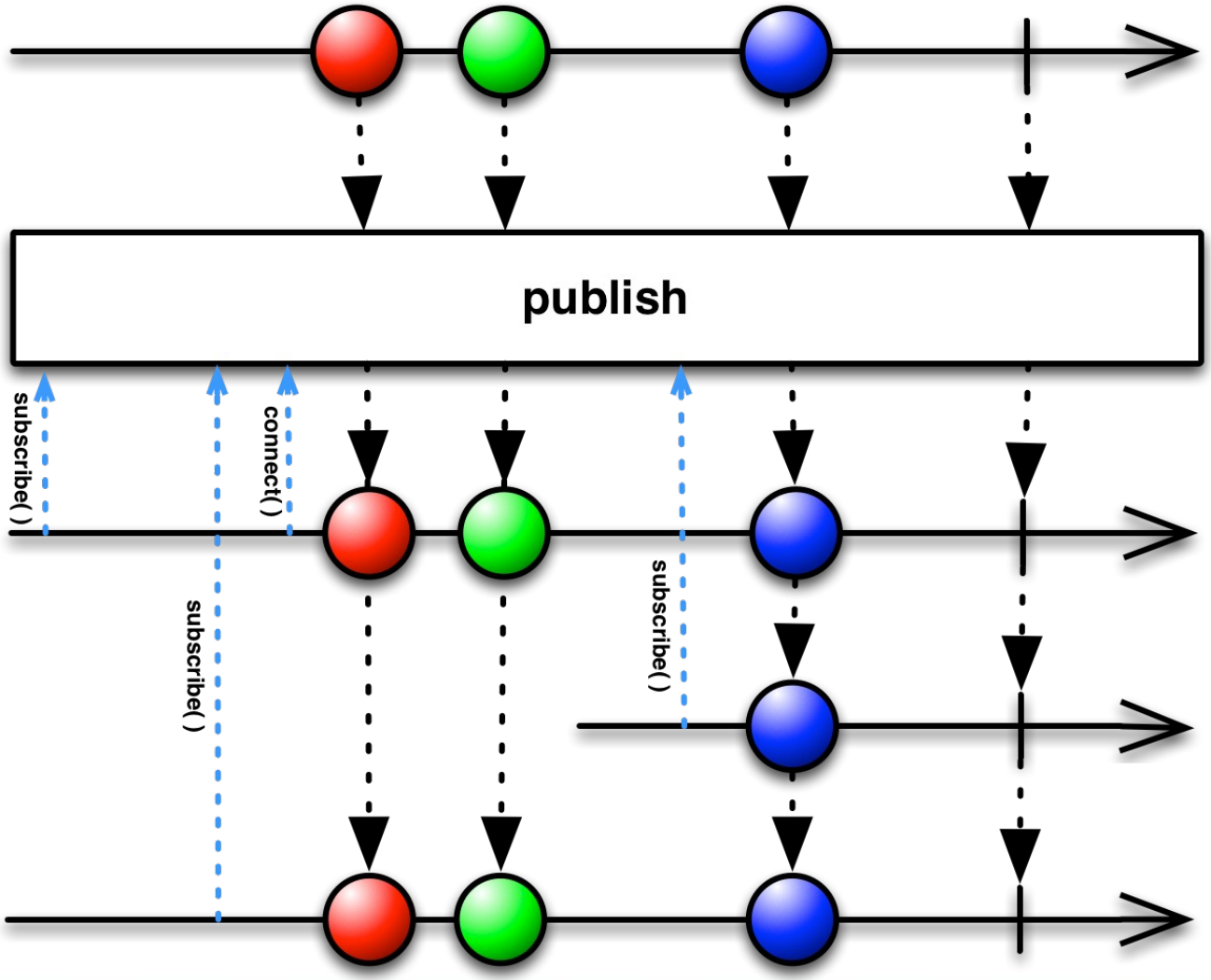


Source: RxJava 2 API documentation, http://reactivex.io/RxJava/2.x/javadoc/

Hot and Cold Event Streams

- PULL-based (Cold Event Streams) – Cold streams (e.g. RxJava Observable / Flowable or Reactor Flow / Mono) are streams that run their sequence when and if they are subscribed to. They present the sequence from the start to each subscriber.
- PUSH-based (Hot Event Streams) – emit values independent of individual subscriptions. They have their own timeline and events occur whether someone is listening or not. When subscription is made observer receives current events as they happen.
  Example: mouse events

Converting Cold to Hot Stream



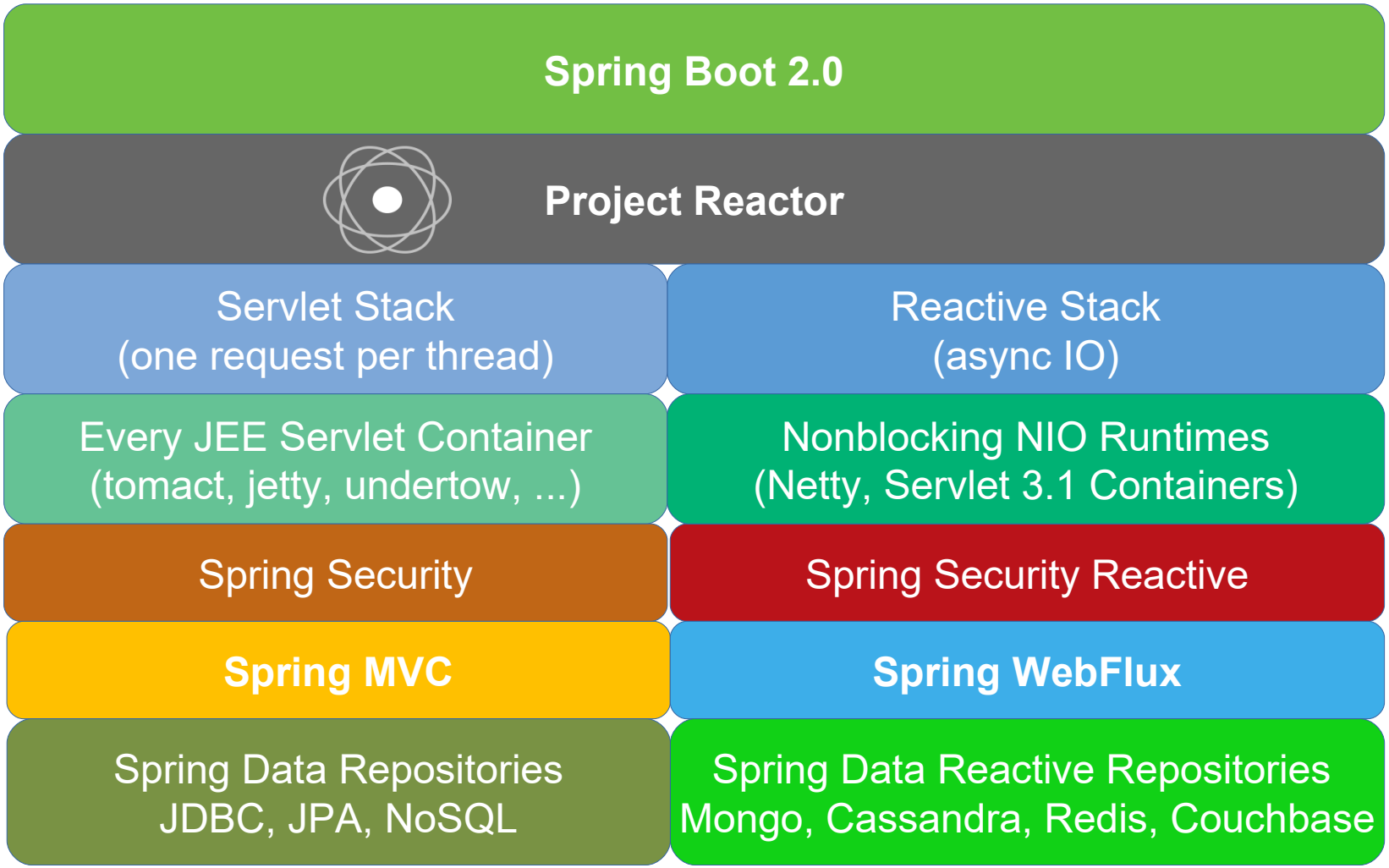
Hot Stream Example - Reactor

```
EmitterProcessor<String> emitter =
        EmitterProcessor.create();
FluxSink<String> sink = emitter.sink();
emitter.publishOn(Schedulers.single())
        .map(String::toUpperCase)
        .filter(s -> s.startsWith("HELLO"))
        .delayElements(Duration.of(1000, MILLIS))
        .subscribe(System.out::println);
sink.next("Hello World!"); // emit - non blocking
sink.next("Goodbye World!");
sink.next("Hello Trayan!");
Thread.sleep(3000);
```

Top New Features in Spring 5

- Reactive Programming Model
- Spring Web Flux
- Reactive DB repositories & integrations + hot event streaming: MongoDB, CouchDB, Redis, Cassandra, Kafka
- JDK 8+ and Java EE 7+ baseline
- Testing improvements – WebTestClient (based on reactive WebFlux WebClient)
- Kotlin functional DSL

Spring 5 Main Building Blocks



Source: *https://spring.io*

WebFlux: Best Expalined in Code

Lets see REST service Spring 5 WebFlux demos. Available @GitHub:

https://github.com/iproduct/spring-5-webflux

Thank's for Your Attention!

Trayan Iliev

CEO of IPT – Intellectual Products & Technologies

http://iproduct.org/

http://robolearn.org/

https://github.com/iproduct

https://twitter.com/trayaniliev

https://www.facebook.com/IPT.EACAD

https://plus.google.com/+IproductOrg