



## **RSocket or how to communicate efficiently with your microservices**

<https://github.com/volkaert/rsocket-demo-2020>

A series of thin, overlapping wavy lines in shades of orange, yellow, and teal flow horizontally across the middle of the slide, creating a sense of motion and energy.

**Fabrice VOLKAERT**  
**@volkaert**

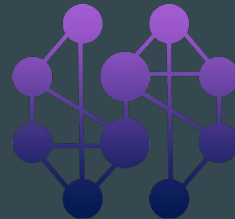
# # whoami

# ~~root~~

 @volkaert



Java



# 1974: TCP/IP



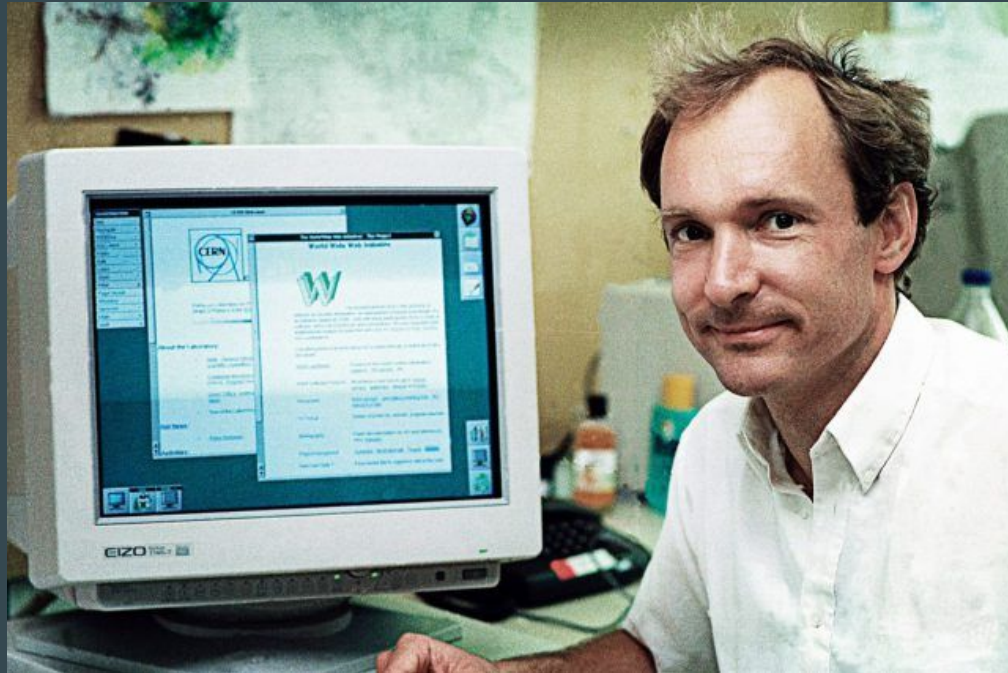
**Vint Cerf**



**Bob Kahn**

TCP/IP is **bi-directional** and has **flow control**

# 1989: HTTP



Tim Berners Lee

HTTP has been designed to **exchange documents**

# HTTP suffers from limitations



Single request connections (HTTP 1.0) are inefficient, **pipelining** (HTTP 1.1) has blocking issues

In order to be reliable, **patterns and external systems** must be added, each with their own reliability concerns

Limited to **Request-Response** interaction model

# The Reactive Manifesto

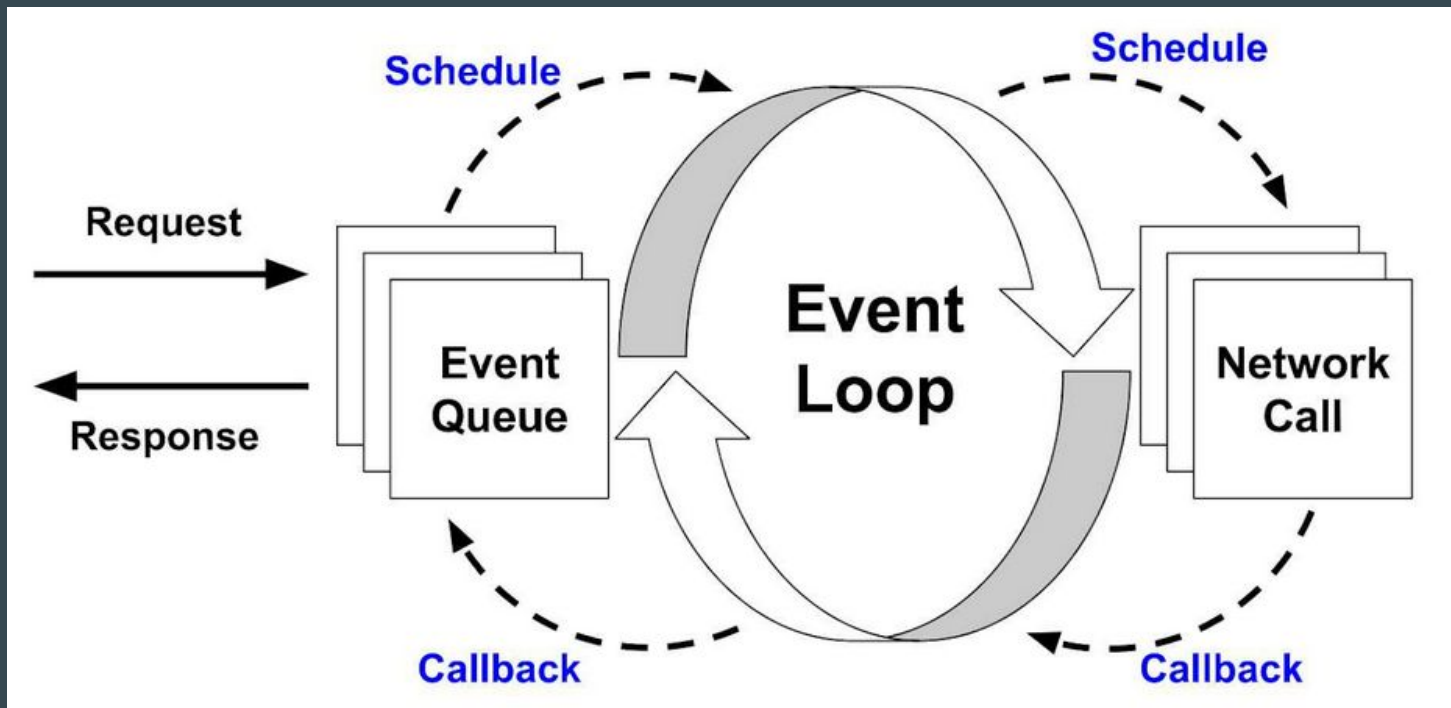


Reactive systems are:

Responsive  
Resilient  
Elastic  
Message Driven

# Reactive Architecture

Highly Efficient and Fundamentally Non Blocking



# Reactive Java Building Blocks

## Reactive Streams

- **Standard** for async stream processing with non-blocking back-pressure
- **Publisher/Subscriber/Subscription/Processor**

## Project Reactor

- **Implementation** of the Reactive Streams specification for the JVM
- Adds **Flux** and **Mono** operators





# Reactive Programming - Example



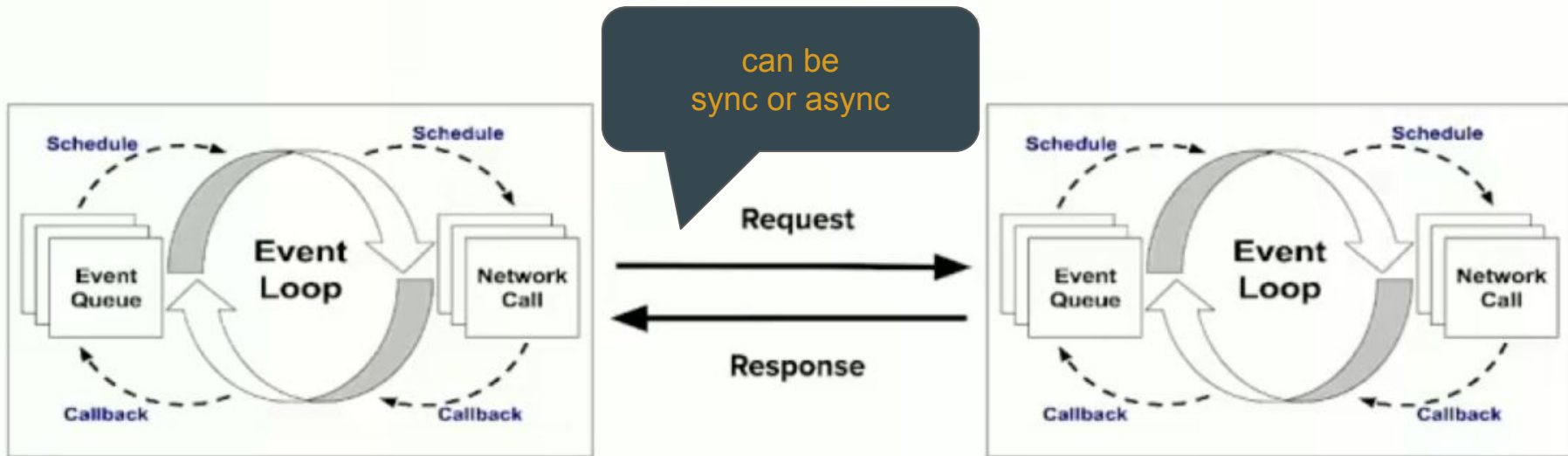
```
@GetMapping("/health")
Mono<Health> compositeHealth() {
    return Mono.zip(
        webClient.get().uri("https://serviceA/health")
            .retrieve().bodyToMono(Health.class),
        webClient.get().uri("https://serviceB/health")
            .retrieve().bodyToMono(Health.class))
        .map(t -> composite(t.getT1(), t.getT2()));
}
```

# Reactive support stops at the application boundary



# Reactive Inter-process Communication

Reactive has no opinion on synchronous vs asynchronous  
Key differentiator is **back-pressure** (Reactive pull/push)



# Back-pressure

(Flow-control)



Without



With



# Definition



RSocket is a **bi-directional, multiplexed, message-based, binary protocol** based on **Reactive Streams** backpressure

RSocket is **transport agnostic**  
(TCP, WebSocket, UDP, HTTP2...)

# 4 interaction models

Request-Response (1 to 1)



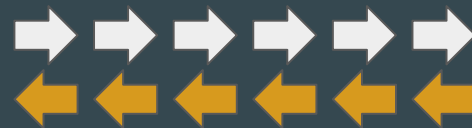
Fire-and-Forget (1 to 0)



Request-Stream (1 to many)



Channel (many to many)



# Promoters and Users

Facebook is migrating its Thrift and GraphQL clients to use RSocket (up to 10x faster than HTTP, using 90% less resources)

RSocket is built into Spring Framework 5.2 and Spring Boot 2.2

RSocket is built into Alibaba's Dubbo 3.0



**REACTIVE**  
FOUNDATION



Pivotal

netifi

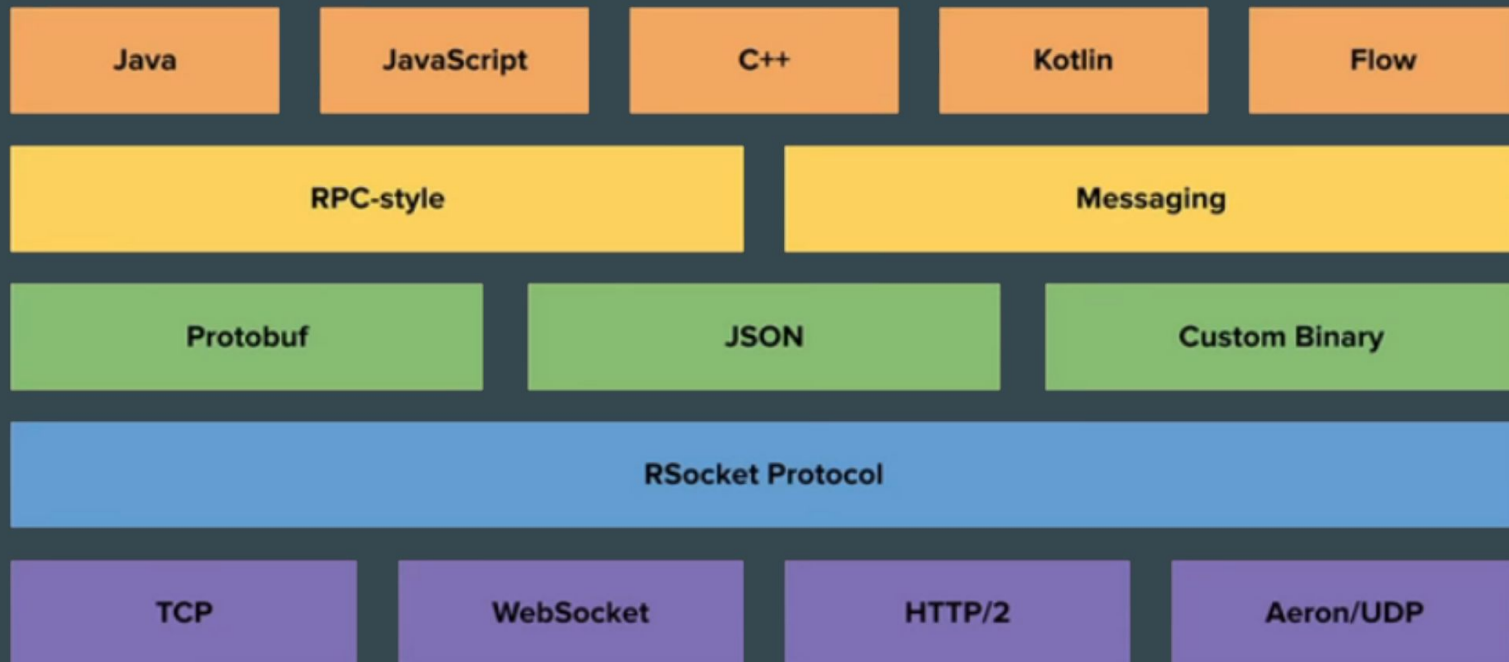
Alibaba Cloud

Lightbend

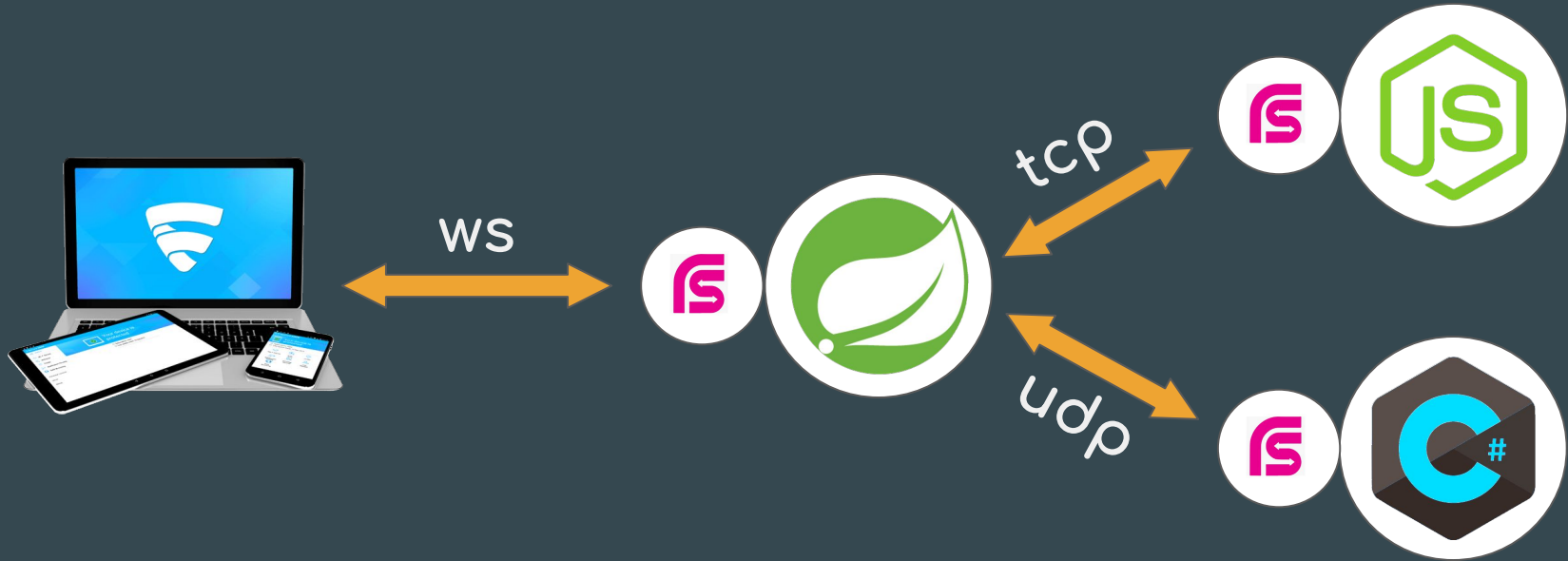
vlingo  
Fluent. Reactive. Delivered.



# A polyglot stack



# Compose with no semantic loss





.HU

# Spring Boot

## Application - Server Setup

pom.xml



```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-rsocket</artifactId>
</dependency>
```

application.properties



spring.rsocket.server.address:	Network address to which the server should bind.
spring.rsocket.server.mapping-path:	Path under which RSocket handles requests (only works with websocket transport).
spring.rsocket.server.port:	Server port.
spring.rsocket.server.transport:	RSocket transport protocol.

# Spring Boot

## Application - Client Setup



```
@Configuration
public class ClientConfiguration {

    @Bean
    public RSocket rSocket() {
        return RSocketFactory
            .connect()
            .mimeType(MimeTypeUtils.APPLICATION_JSON_VALUE, MimeTypeUtils.APPLICATION_JSON_VALUE)
            .frameDecoder(PayloadDecoder.ZERO_COPY)
            .transport(TcpClientTransport.create(7000))
            .start()
            .block();
    }

    @Bean
    RSocketRequester rSocketRequester(RSocketStrategies rSocketStrategies) {
        return RSocketRequester.wrap(rSocket(), MimeTypeUtils.APPLICATION_JSON, rSocketStrategies);
    }
}
```

# Spring Boot

## Request / Response - Server



```
@Controller
public class MarketDataRSocketController {

    private final MarketDataRepository marketDataRepository;

    public MarketDataRSocketController(MarketDataRepository marketDataRepository) {
        this.marketDataRepository = marketDataRepository;
    }

    @RequestMapping("currentMarketData")
    public Mono<MarketData> currentMarketData(MarketDataRequest marketDataRequest) {
        return marketDataRepository.getOne(marketDataRequest.getStock( ));
    }
}
```

# Spring Boot

## Request / Response - Client

```

@RestController
public class MarketDataRestController {

    private final RSocketRequester rSocketRequester;

    public MarketDataRestController(RSocketRequester rSocketRequester) {
        this.rSocketRequester = rSocketRequester;
    }

    @GetMapping(value = "/current/{stock}")
    public Publisher<MarketData> current(@PathVariable("stock") String stock) {
        return rSocketRequester
            .route(currentMarketData")
            .data(new MarketDataRequest(stock))
            .retrieveMono(MarketData.class);
    }
}

```

# Spring Boot

## Fire & Forget - Server



```
@Messaging("collectMarketData")
public Mono<Void> collectMarketData(MarketData marketData) {
    marketDataRepository.add(marketData);
    return Mono.empty();
}
```



# Spring Boot

## Fire & Forget - Client



```
@GetMapping(value = "/collect")
public Publisher<Void> collect() {
    return rSocketRequester
        .route("collectMarketData")
        .data(getMarketData())
        .send();
}
```

# Spring Boot


## Request / Stream - Server



```
@Messaging("feedMarketData")
public Flux<MarketData> feedMarketData(MarketDataRequest marketDataRequest) {
    return marketDataRepository.getAll(marketDataRequest.getStock());
}
```

# Spring Boot

## Request / Stream - Client



```
@GetMapping(value = "/feed/{stock}", produces = MediaType.TEXT_EVENT_STREAM_VALUE,  
public Publisher<MarketData> feed(@PathVariable("stock") String stock) {  
    return rSocketRequester  
        .route("feedMarketData")  
        .data(new MarketDataRequest(stock))  
        .retrieveFlux(MarketData.class);  
}
```

# Other Features



Resumability

Cancellation

Metadata and Data in each frame

Leasing

Transparent fragmentation

# RSocket Low Level



```
public interface RSocket {  
  
    Mono<Payload> requestResponse(Payload payload);  
    Mono<Void> fireAndForget(Payload payload);  
    Flux<Payload> requestStream(Payload payload);  
    Flux<Payload> requestChannel(Publisher<Payload> payloads);  
    Mono<Void> metadataPush(Payload payload);  
  
}
```

# RSocket RPC

## Protobuf 3 IDL + Compiler Plugin



```
service SimpleService {  
  rpc RequestReply (SimpleRequest) returns (SimpleResponse) {}  
  rpc FireAndForget (SimpleRequest) returns (google.protobuf.Empty) {}  
  rpc StreamingRequestSingleResponse (stream SimpleRequest) returns (SimpleResponse) {}  
  rpc StreamingRequestAndResponse (stream SimpleRequest) returns (stream SimpleResponse) {}  
}  
  
message SimpleRequest { string requestMessage = 1; }  
  
message SimpleResponse { string responseMessage = 1; }
```

# RSocket CLI

## Request / Response Interaction



```
$ rsocket-cli -i "I am a Server" --server --debug tcp://localhost:8765 # window 1  
$ rsocket-cli --request -i "I am a Client" --debug tcp://localhost:8765 # window 2
```

A request stream of dictionary words, with frames debugged:



```
$ rsocket-cli --debug -i=@/usr/share/dict/words --server tcp://localhost:8765 # window 1  
$ rsocket-cli --stream -i "Word Up" tcp://localhost:8765 # window 2
```



# RSocket vs other protocols





# RSocket vs HTTP

## RSocket

Efficient and Responsive

- Single, shared long-live connection
- Multiplexes messages
- Communicates backpressure
- Either party can initiate requests  
(flexible requester/responder roles)
- Supports cancelling/resuming streams

## HTTP

Slowly Improving

- New connection per request (HTTP 1.0)
- Pipelines messages (HTTP 1.1)
- Does not communicate backpressure
- Only client can initiate requests  
(fixed client/server roles)
- Does not support cancelling/resuming streams

# RSocket vs gRPC

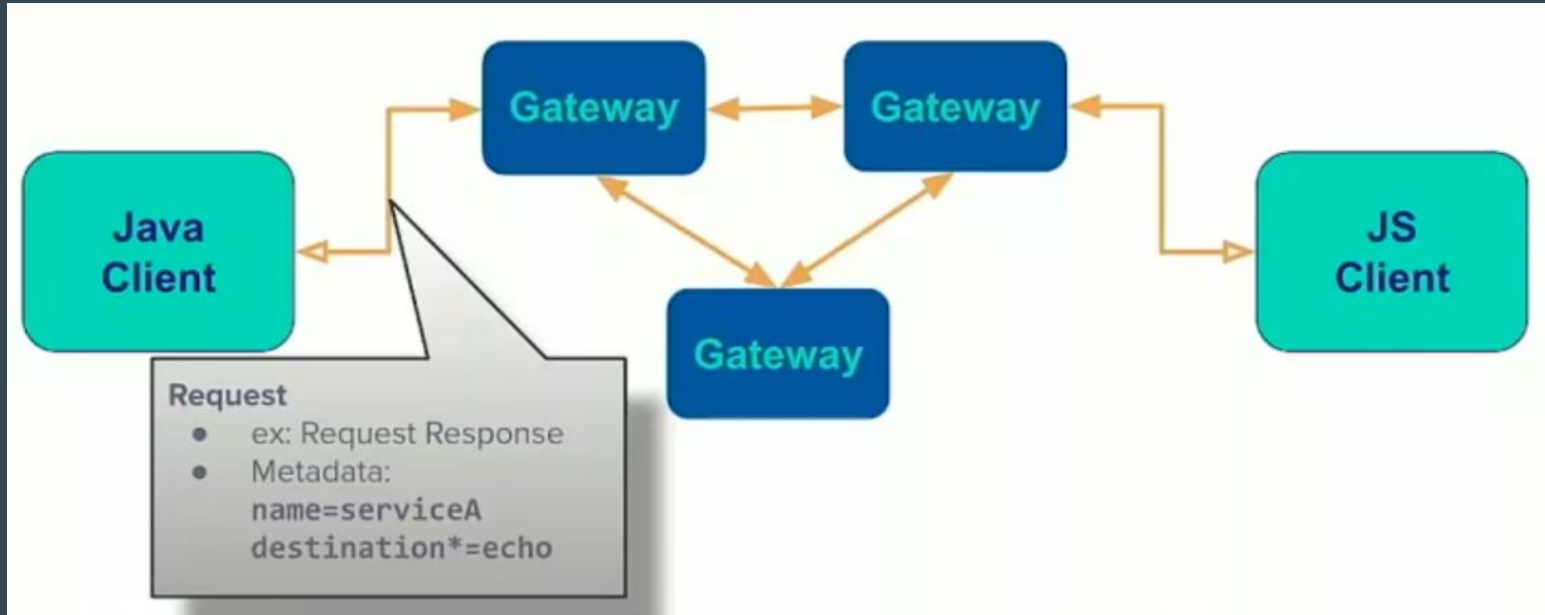
## RSocket

- Transport agnostic and opaque bytes payload
- Native support of browsers using WebSockets
- Requester/responder interaction (a server can make calls to a web browser once connected)
- Application level flow control
- Relaxed Spatial coupling (the recipient does not need to exist on the other side)
- RPC-style, Spring Messaging, low-level

## gRPC

- Limited to HTTP/2 and Protobuf
- gRPC not natively supported by browsers (gRPC-Web proxy required)
- Traditional client/server interaction
- Byte based flow control (HTTP/2)
- Strong Spatial coupling (there must be a recipient on the other end)
- RPC-style integration only

# Spring Cloud Gateway RSocket



# With a gateway, you won't need...



Separate Service Discovery

Message Broker

Circuit-Breaker

Client-side load-balancer

Sidecar

Startup ordering problems

(requests to non-existent services allowed)

Special cases for warmup

...

# Gateway: As-Is



**WE  
ARE  
HERE**

RSocket support is a **technical preview**  
in Hoxton release (Q3 2019)

RSocket **Routing and Forwarding  
extension specification** has not been  
finalized but will be soon  
(need correct metadata)

# Gateway: Roadmap



Documentation

Clustering enhancement

Tracing integration

Fault tolerance improvements  
(cross-regions clusters)

Shard routing

# Takeaway



Modern

Efficient

Polyglot stack

Various interaction models

Various programming levels

Spring integration

Spring Cloud Gateway

# Resources



<https://dzone.com/articles/reactive-service-to-service-communication-with-rso-1>

[https://www.youtube.com/watch?v=PfbycN\\_eqhg](https://www.youtube.com/watch?v=PfbycN_eqhg)

<https://www.youtube.com/watch?v=D2Z5d9dEBxQ>

<https://www.youtube.com/watch?v=Ylrxpbs6vcs&t=2794s>

<https://www.youtube.com/watch?v=awfwPWgPP3o>

<https://www.youtube.com/watch?v=iSSrZoGtoSE&feature=youtu.be>

<https://github.com/b3rnoulli/rsocket-examples>

<https://github.com/simonbasle-demos/rsocket-demo>

<https://www.baeldung.com/spring-boot-rsocket>

<https://medium.com/@domenicosibilio/rsocket-with-spring-boot-js-zero-to-hero-ef63128f973d>

<https://www.techtalks.lk/blog/2019/9/rsocket-authenticationauthorization-using-spring-security>



