Overview of RSocket Interaction Models

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

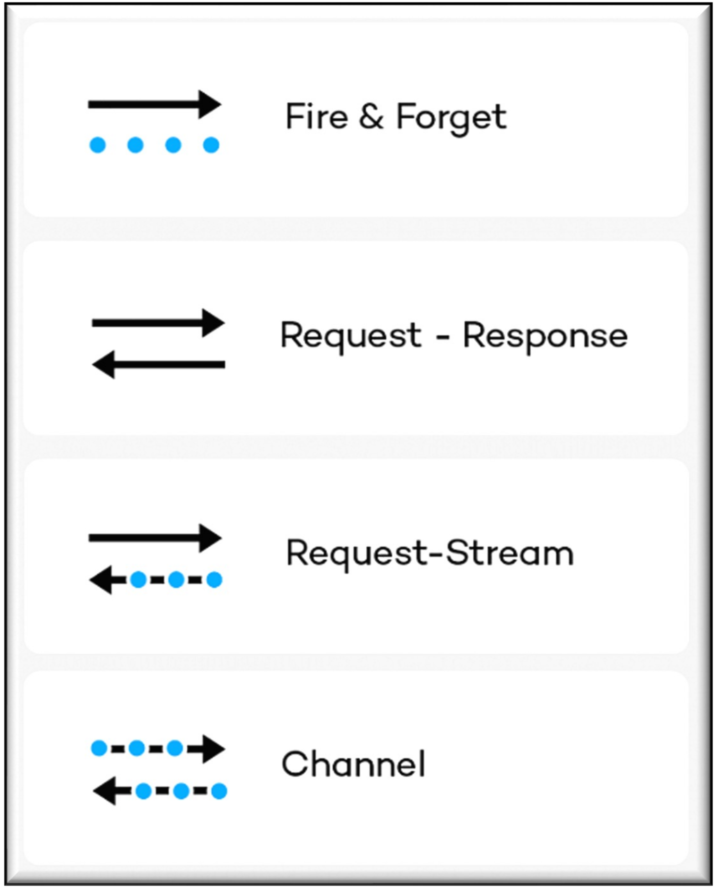
Professor of Computer Science

Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

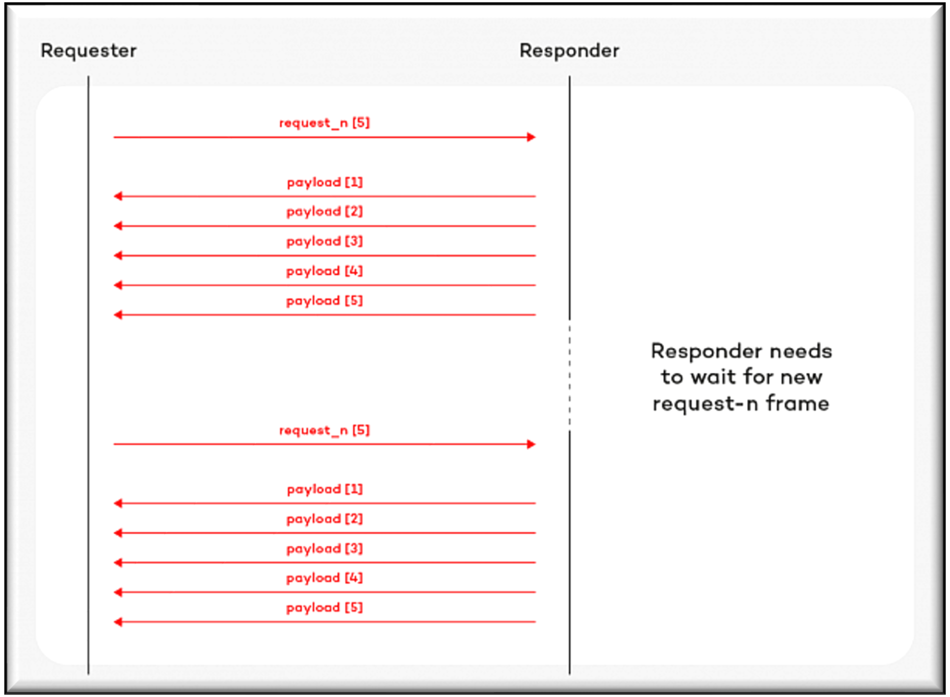
Learning Objectives in this Part of the Lesson

- Understand the RSocket framework
- Recognize the RSocket interaction models



Learning Objectives in this Part of the Lesson

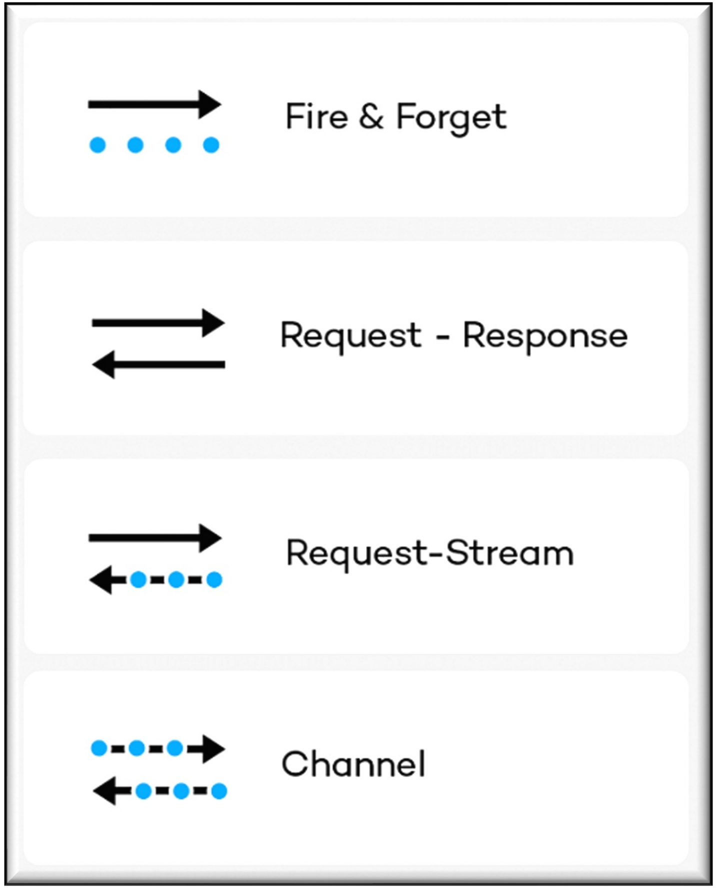
- Understand the RSocket framework
- Recognize the RSocket interaction models
  - As well as backpressure support



Overview of RSocket Interaction Modes

Overview of RSocket Interaction Models

- RSocket provides four interaction models



Fire & Forget

Request - Response

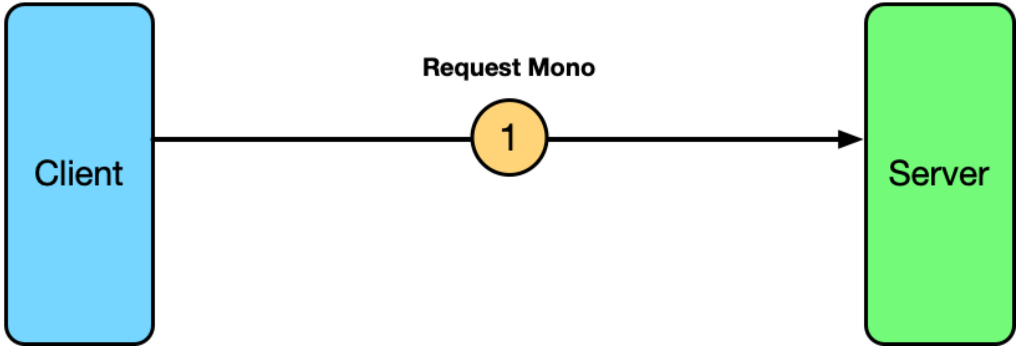
Request-Stream

Channel

See projectreactor.io

Overview of RSocket Interaction Models

- RSocket provides four interaction models
  - **Fire-and-Forget**
    - Each one-way message receives no response from the server



See spring.io/blog/2020/03/16/getting-started-with-rsocket-spring-boot-fire-and-forget

Overview of RSocket Interaction Models

- RSocket provides four interaction models
  - **Fire-and-Forget**
    - Each one-way message receives no response from the server
    - This optimization is useful when a response is not needed

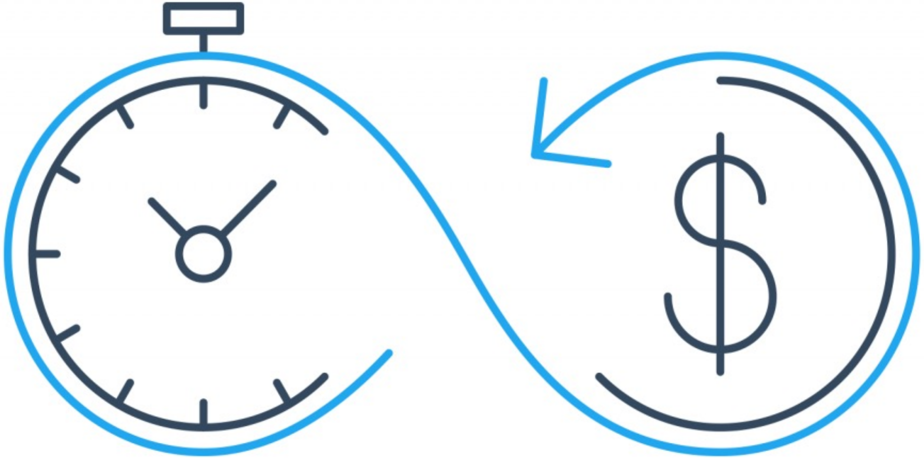
```
Mono<Void> completionSignal =
  rsocketClientProxy
    .fireAndForget(message);
```

Spring WebFlux (& WebMVC) don’t really support this use case

Overview of RSocket Interaction Models

- RSocket provides four interaction models
  - **Fire-and-Forget**
    - Each one-way message receives no response from the server
    - This optimization is useful when a response is not needed
      - Saves network & computer processing time

```
Mono<Void> completionSignal =
  rsocketClientProxy
     .fireAndForget(message);
```



Overview of RSocket Interaction Models

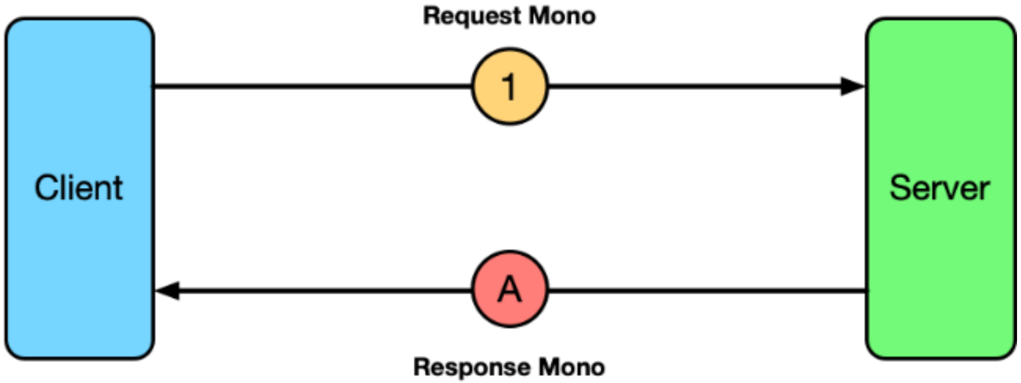
- RSocket provides four interaction models
  - **Fire-and-Forget**
    - Each one-way message receives no response from the server
    - This optimization is useful when a response is not needed
    - Primarily intended for use cases that support lossiness
      - e.g., non-critical event logging

```
Mono<Void> completionSignal =
  rsocketClientProxy
    .fireAndForget(message);
```

See medium.com/mandiri-engineering/fire-and-forget-e59b745c9f97

Overview of RSocket Interaction Models

- RSocket provides four interaction models
  - **Request-Response**
    - Each two-way async request receives a single async response from the server



See spring.io/blog/2020/03/02/getting-started-with-rsocket-spring-boot-server

Overview of RSocket Interaction Models

- RSocket provides four interaction models
  - **Request-Response**
    - Each two-way async request receives a single async response from the server
    - A very common async use case

```
Mono<Response> response =
  rsocketClientProxy
    .requestResponse
      (monoRequest);
```

COMMON

Spring WebFlux also supports this async two-way use case for HTTP requests/responses

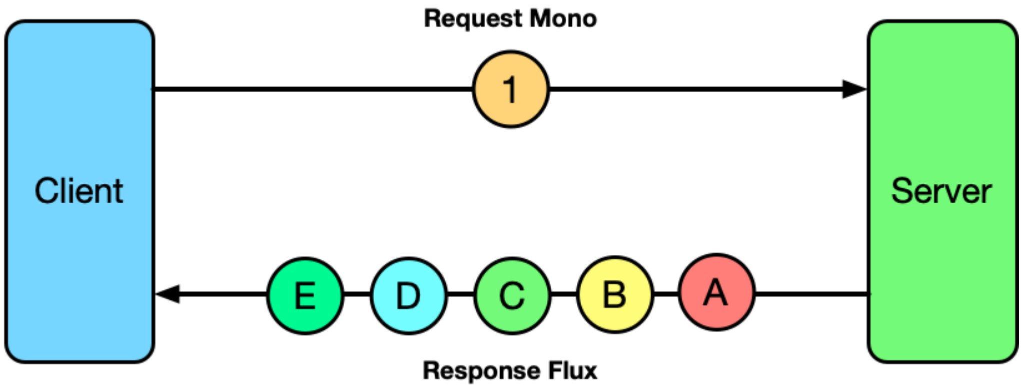
Overview of RSocket Interaction Models

- RSocket provides four interaction models
  - **Request-Response**
    - Each two-way async request receives a single async response from the server
    - A very common async use case
    - Although it looks like a typical request/ response, underneath it never blocks synchronously

```
Mono<Response> response =
  rsocketClientProxy
    .requestResponse
      (monoRequest);
```

Overview of RSocket Interaction Models

- RSocket provides four interaction models
  - **Request-Stream**
    - Each async request receives a stream of responses from the server



See spring.io/blog/2020/03/23/getting-started-with-rsocket-spring-boot-request-stream

Overview of RSocket Interaction Models

- RSocket provides four interaction models
  - **Request-Stream**
    - Each async request receives a stream of responses from the server
    - Allows streaming of multiple response messages

```
Flux<Response> response =
  rsocketClientProxy
    .requestStream
       (monoRequest);
```

Overview of RSocket Interaction Models

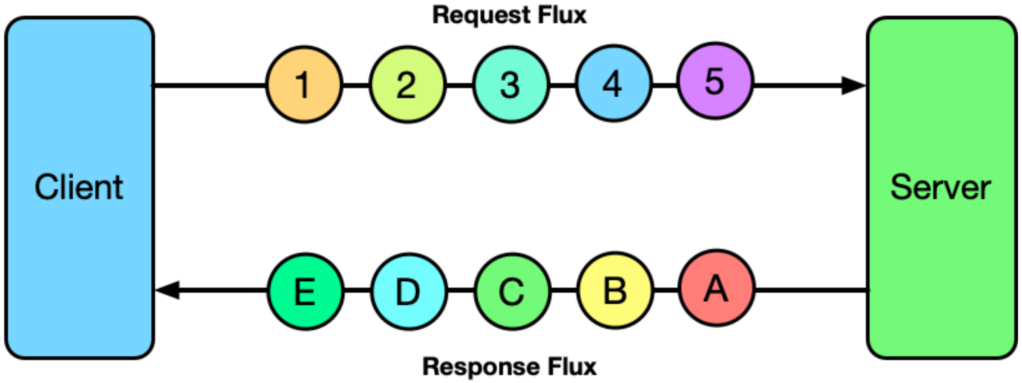
- RSocket provides four interaction models
  - **Request-Stream**
    - Each async request receives a stream of responses from the server
    - Allows streaming of multiple response messages
    - Instead of getting back all data as a single response, each element is streamed back in order

```
Flux<Response> response =
  rsocketClientProxy
    .requestStream
       (monoRequest);
```

Spring WebFlux also supports this async use case for HTTP requests/responses

Overview of RSocket Interaction Models

- RSocket provides four interaction models
  - **Channel**
    - A stream of async messages can be sent bi-directionally between client & server



See spring.io/blog/2020/04/06/getting-started-with-rsocket-spring-boot-channels

Overview of RSocket Interaction Models

- RSocket provides four interaction models
  - **Channel**
    - A stream of async messages can be sent bi-directionally between client & server
    - A data stream from client-to-server coexists alongside a data stream from server-to-client

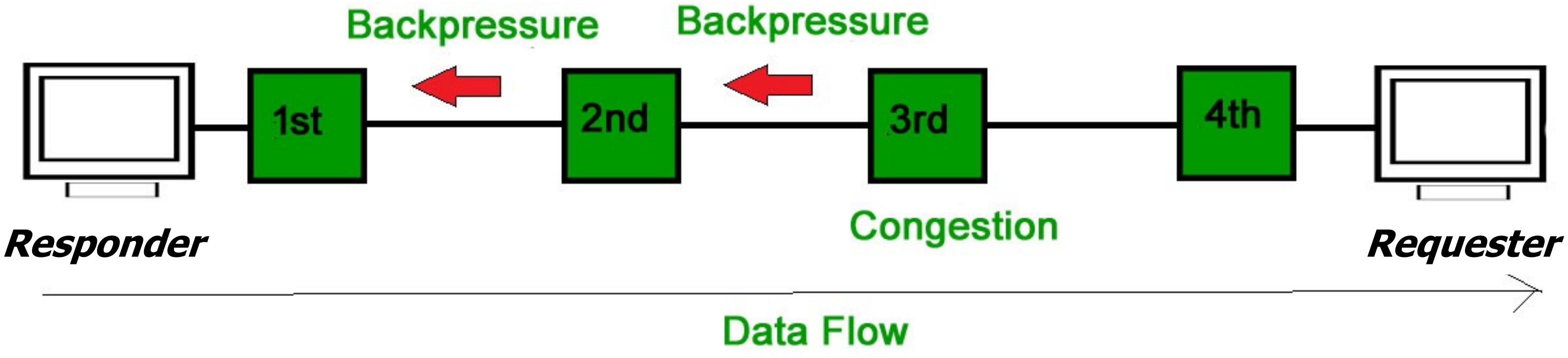
```
Flux<Response> output =
  rsocketClientProxy
    .requestChannel
      (fluxRequest);
```

Spring WebFlux also supports this async use case for HTTP requests/responses

Overview of RSocket Backpressure Support

Overview of RSocket Backpressure Support

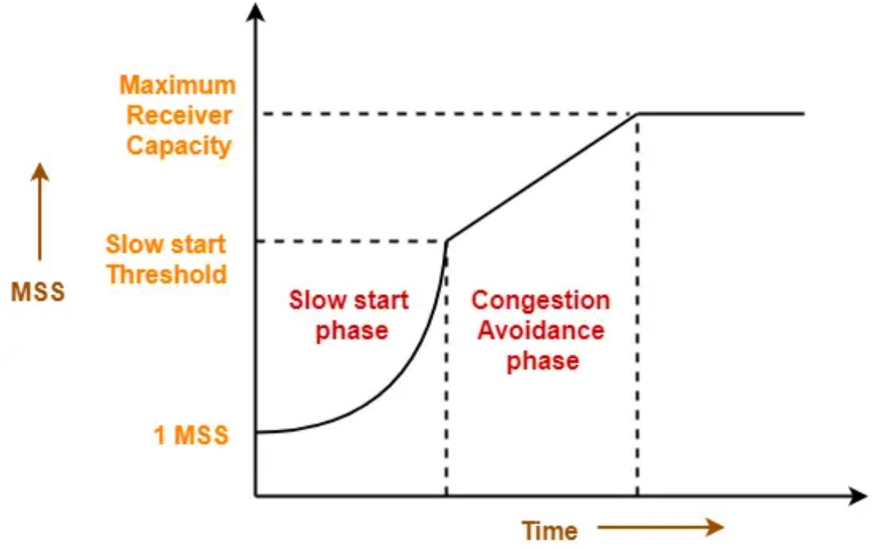
- For Request-Stream & Channel models backpressure signals travel between requester & responder, allowing a requester to slow down a responder at the source



See grapeup.com/blog/reactive-service-to-service-communication-with-rsocket-introduction

Overview of RSocket Backpressure Support

- For Request-Stream & Channel models backpressure signals travel between requester & responder, allowing a requester to slow down a responder at the source
  - Backpressure reduces reliance on transport layer congestion control



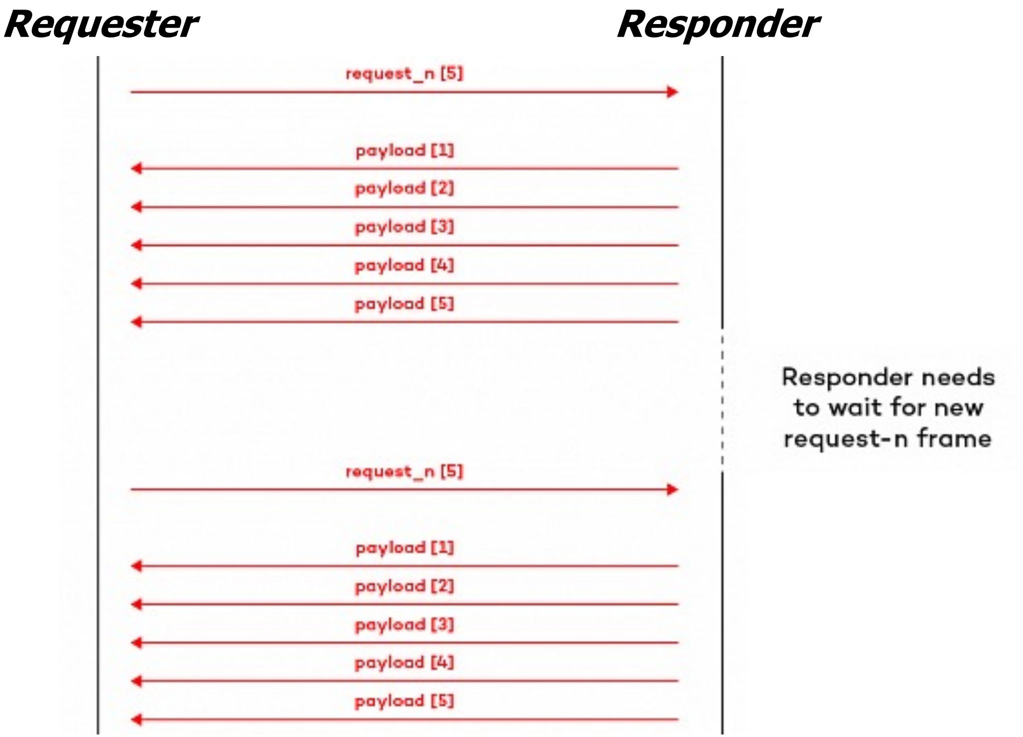
See en.wikipedia.org/wiki/TCP_congestion_control

Overview of RSocket Backpressure Support

- For Request-Stream & Channel models backpressure signals travel between requester & responder, allowing a requester to slow down a responder at the source
  - Backpressure reduces reliance on transport layer congestion control
    - It also minimizes the need for buffering at the network level
      - Or at any level…

Overview of RSocket Backpressure Support

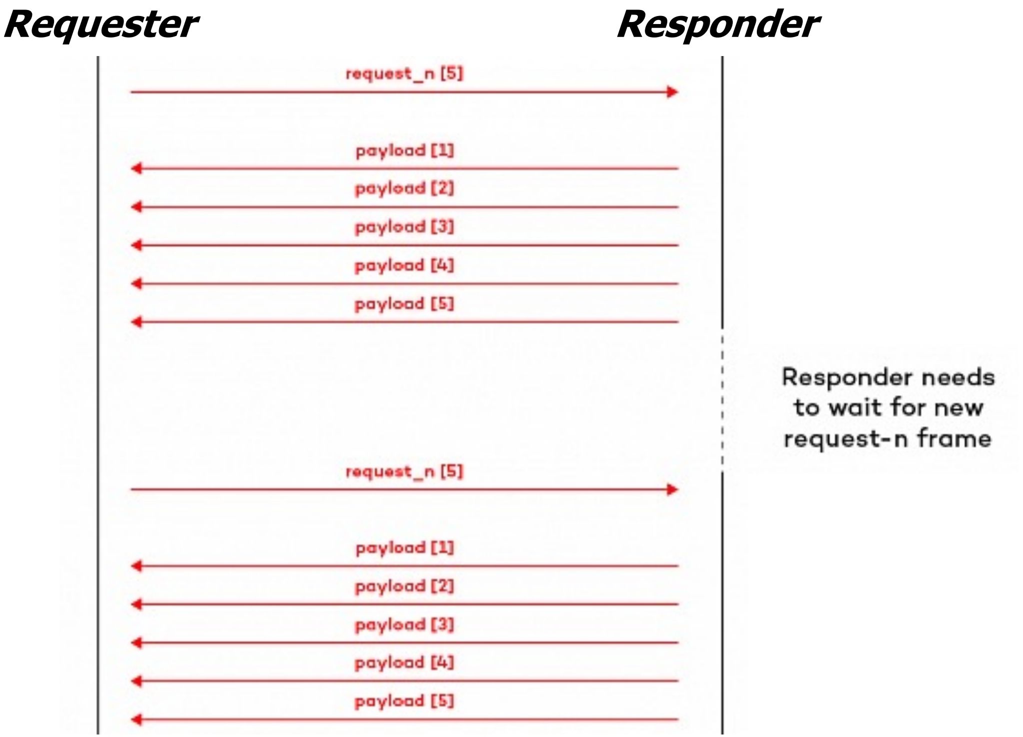
- For Request-Stream & Channel models backpressure signals travel between requester & responder, allowing a requester to slow down a responder at the source
  - Backpressure reduces reliance on transport layer congestion control
  - RSocket backpressure uses the Subscriber/Subscription model



See www.appsdeveloperblog.com/implementing-backpressure-in-project-reactor

Overview of RSocket Backpressure Support

- For Request-Stream & Channel models backpressure signals travel between requester & responder, allowing a requester to slow down a responder at the source
  - Backpressure reduces reliance on transport layer congestion control
  - RSocket backpressure uses the Subscriber/Subscription model
    - We covered this earlier in the context of Project Reactor



See earlier lesson on *"Overview of Backpressure Models in the Project Reactor Flux"*

Overview of RSocket Backpressure Support

- For Request-Stream & Channel models backpressure signals travel between requester & responder, allowing a requester to slow down a responder at the source
  - Backpressure reduces reliance on transport layer congestion control
  - RSocket backpressure uses the Subscriber/Subscription model
  - It also supports the concept of "request leases"
    - Inform the Requester that it may send Requests for a period of time & how many it may send during that duration

See jauntsdn.com/post/rsocket-lease-concurrency-limiting

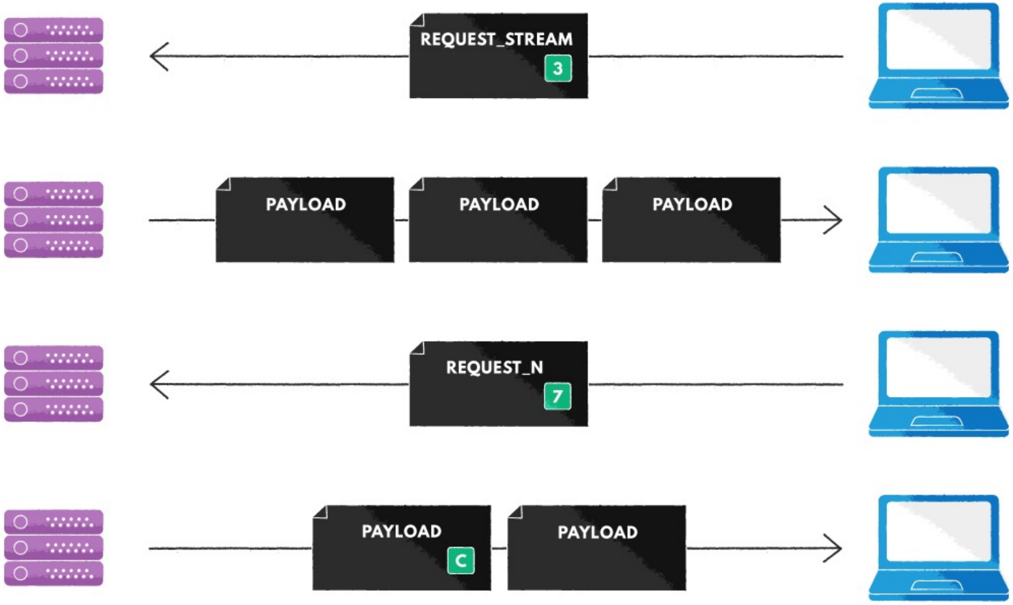
Overview of RSocket Backpressure Support

- The Java RSocket implementation is built upon Project Reactor & Reactor Netty for the transport

See projectreactor.io & www.baeldung.com/spring-boot-reactor-netty

Overview of RSocket Backpressure Support

- The Java RSocket implementation is built upon Project Reactor & Reactor Netty for the transport
  - Signals from reactive streams publishers therefore propagate transparently through RSocket across the network



See projectreactor.io & www.baeldung.com/spring-boot-reactor-netty

End of Overview of RSocket Interaction Models