Overview of RSocket

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

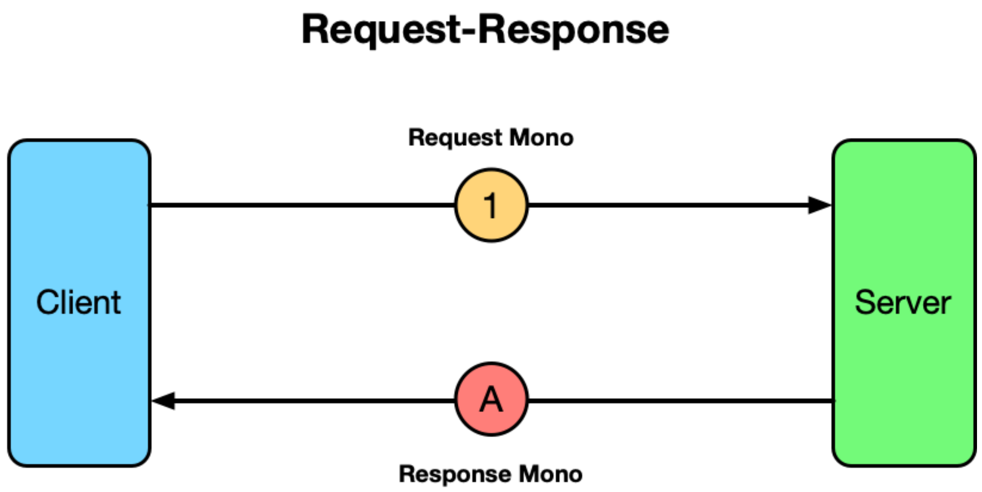
Learning Objectives in this Part of the Lesson

- Understand the motivation for & features of the RSocket framework

See rsocket.io

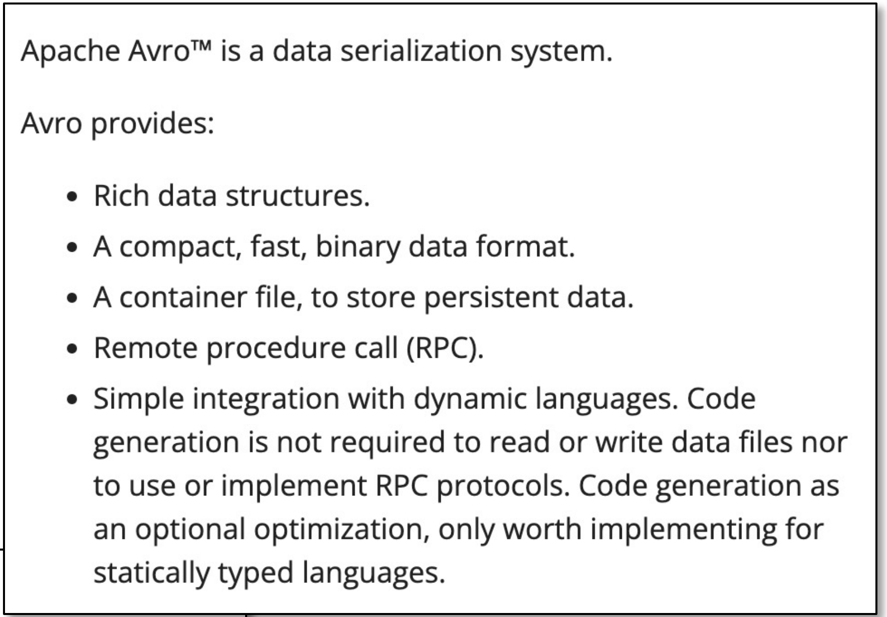
Learning Objectives in this Part of the Lesson

- Understand the motivation for & features of the RSocket framework
  - It provides reactive streams semantics to pass messages across host/process boundaries



Learning Objectives in this Part of the Lesson

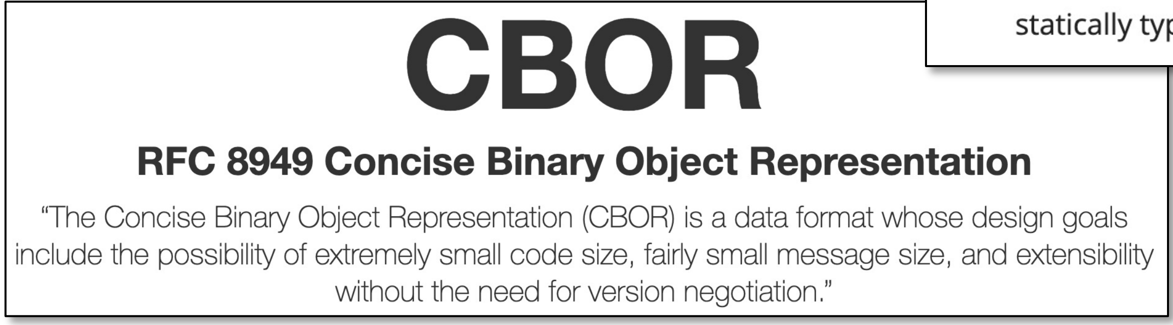
- Understand the motivation for & features of the RSocket framework
  - It provides reactive streams semantics to pass messages across host/process boundaries
  - It also supports application-level binary protocols



Apache Avro™ is a data serialization system.

Avro provides:

- Rich data structures.
- A compact, fast, binary data format.
- A container file, to store persistent data.
- Remote procedure call (RPC).
- Simple integration with dynamic languages. Code generation is not required to read or write data files nor to use or implement RPC protocols. Code generation as an optional optimization, only worth implementing for statically typed languages.



CBOR

RFC 8949 Concise Binary Object Representation

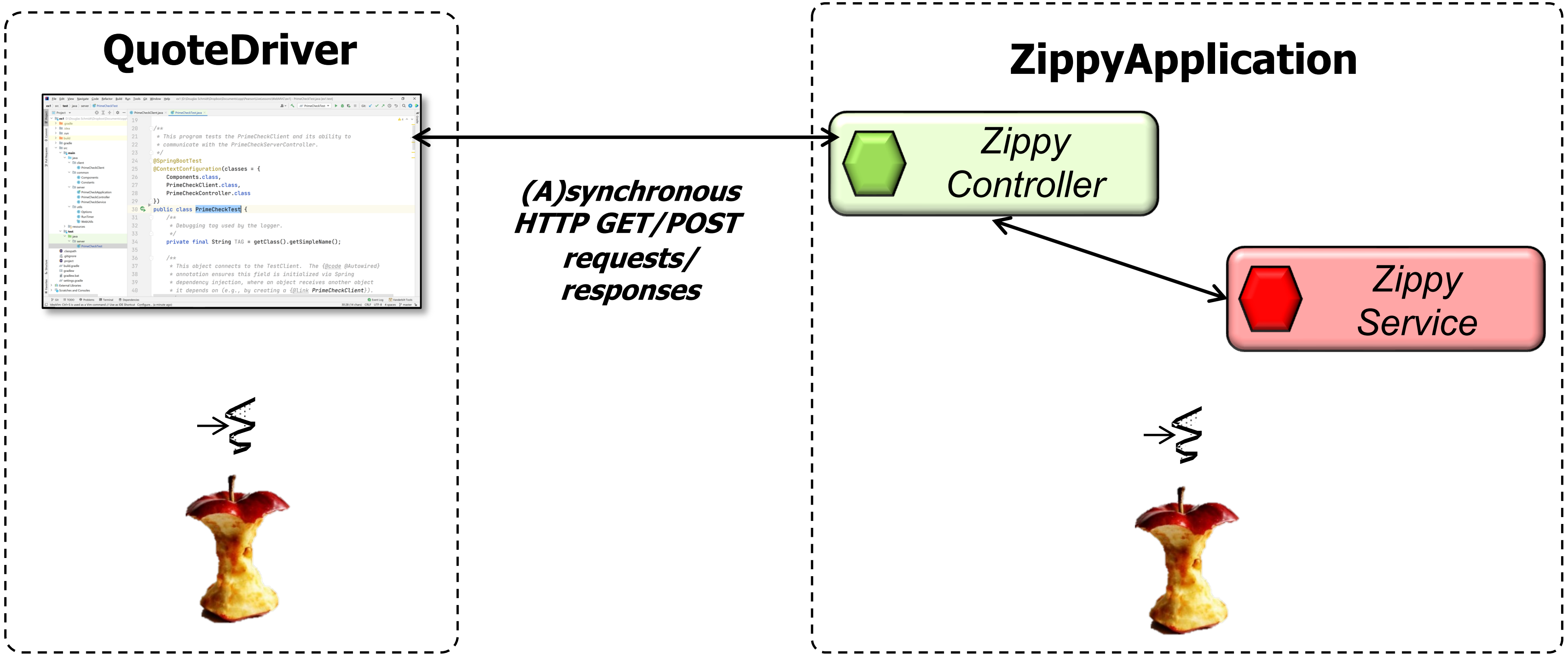
"The Concise Binary Object Representation (CBOR) is a data format whose design goals include the possibility of extremely small code size, fairly small message size, and extensibility without the need for version negotiation."

See en.wikipedia.org/wiki/Communication_protocol#Binary

Motivation for RSocket

Motivation for RSocket

- Thus far our focus has been on using Spring endpoint handler methods to send/receive synchronous & asynchronous requests/responses via HTTP



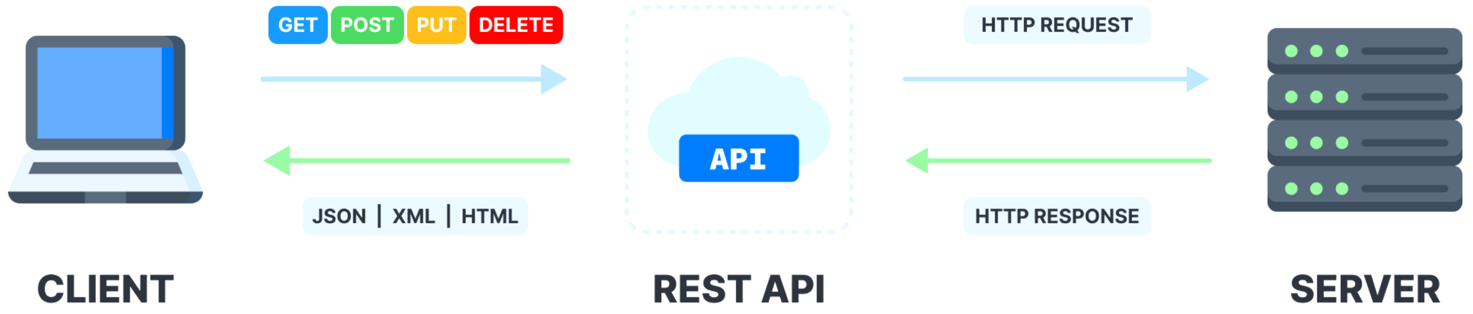
```
/**
 * This program tests the PrimeCheckClient and its ability to
 * communicate with the PrimeCheckServerController.
 */
@SpringBootTest
@ContextConfiguration(classes = {
    Components.class,
    PrimeCheckClient.class,
    PrimeCheckController.class
})
public class PrimeCheckTest {
    /**
     * Debugging tag used by the logger.
     */
    private final String TAG = getClass().getSimpleName();

    /**
     * This object connects to the TestClient.  The {@code @Autowired}
     * annotation ensures this field is initialized via Spring
     * dependency injection, where an object receives another object
     * it depends on (e.g., by creating a {@link PrimeCheckClient}).
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/WebFlux/ex3

Motivation for RSocket

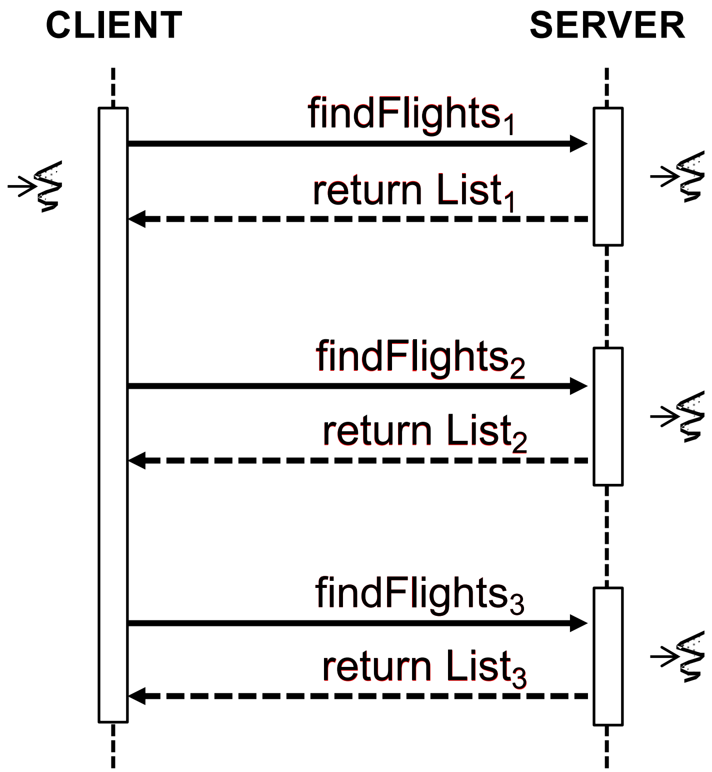
- Although two-way invocations to/from RESTful APIs is a popular approach, there are several limitations

RESTful API Model



Motivation for RSocket

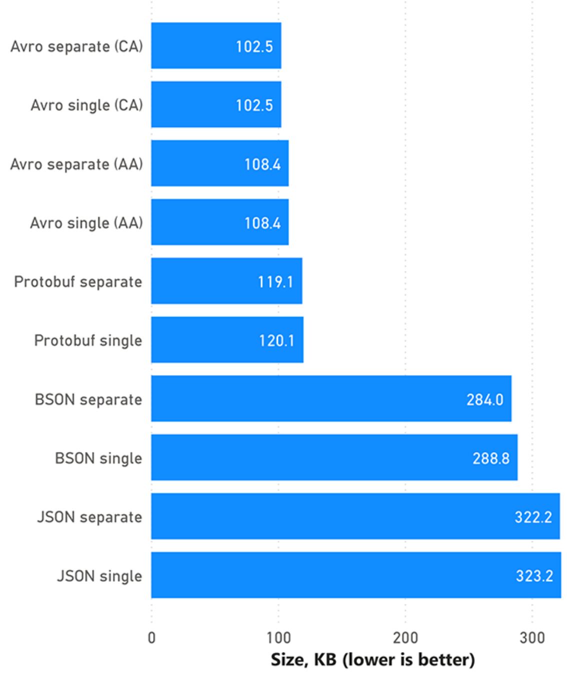
- Although two-way invocations to/from RESTful APIs is a popular approach, there are several limitations
  - Request-response only
    - Does not support bidirectional communication or other interaction models

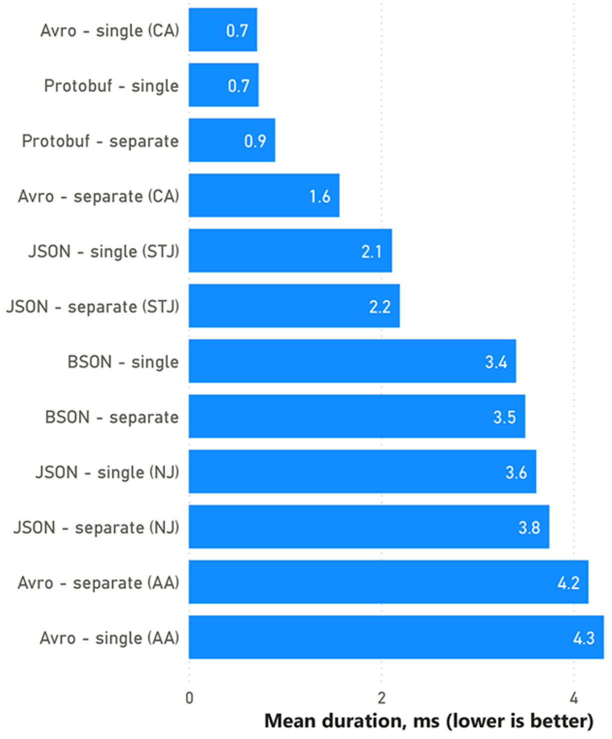


Motivation for RSocket

- Although two-way invocations to/from RESTful APIs is a popular approach, there are several limitations
  - Request-response only
  - Higher overhead & less efficient resource usage

- AA — Apache.Avro
- CA — Chr.Avro
- NJ — Newtonsoft.Json
- STJ — System.Text.Json





See blog.devgenius.io/serialization-performance-in-net-json-bson-protobuf-avro-a25e8207d9de

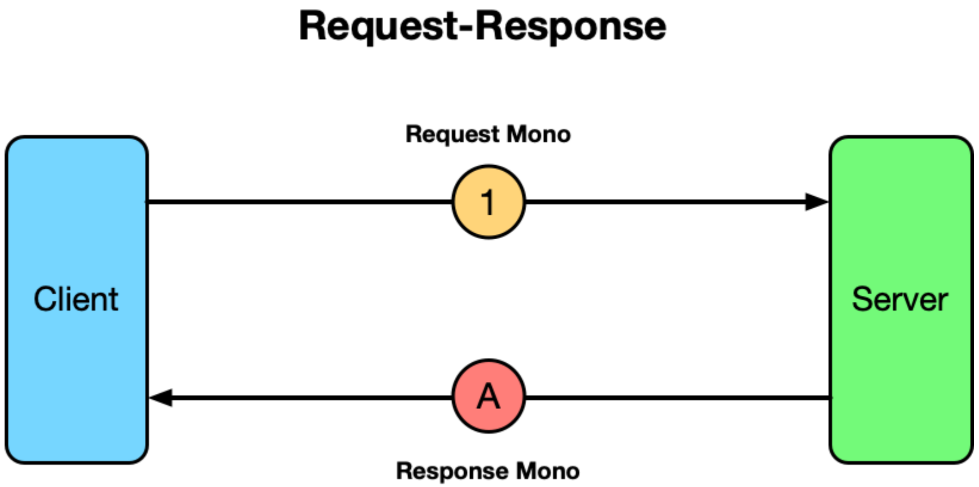
Motivation for RSocket

- Although two-way invocations to/from RESTful APIs is a popular approach, there are several limitations
  - Request-response only
  - Higher overhead & less efficient resource usage
  - No built-in support for resumable streams
    - If a connection is lost, the client can resume the stream from the point where it was interrupted, without losing data

Overview of RSocket

Overview of RSocket

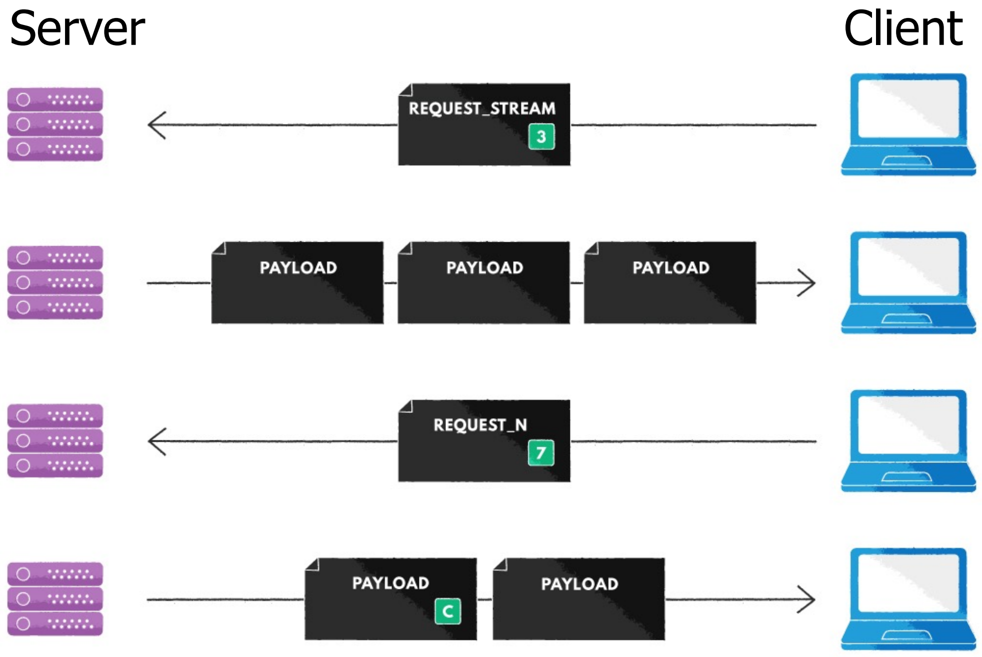
- RSocket is reactive point-to-point messaging framework designed to overcome RESTful limitations



See rsocket.io/about/motivations

Overview of RSocket

- RSocket is reactive point-to-point messaging framework designed to overcome RESTful limitations
  - It supports client/server programs that interact via Flux & Mono reactive types in Project Reactor

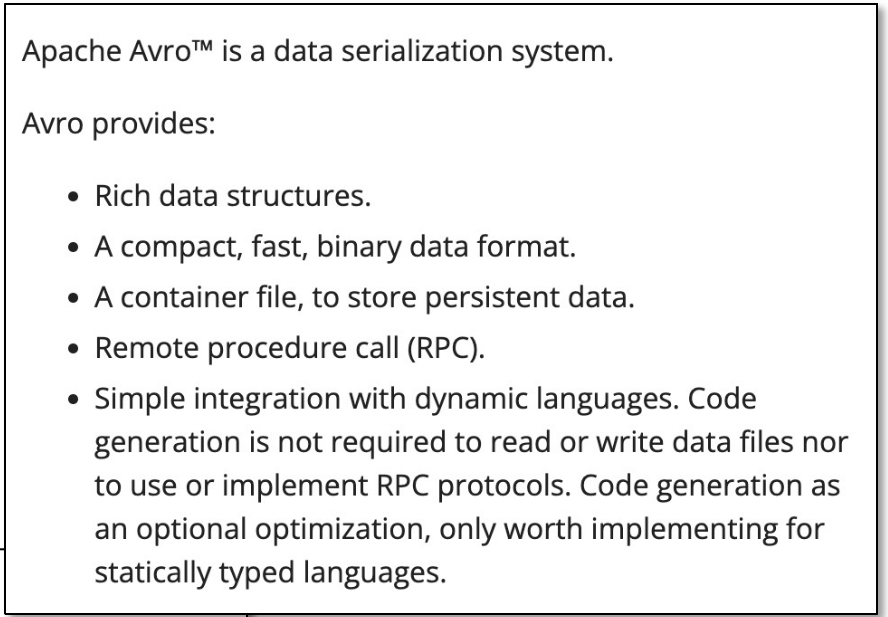


These interactions occur asynchronously

See projectreactor.io

Overview of RSocket

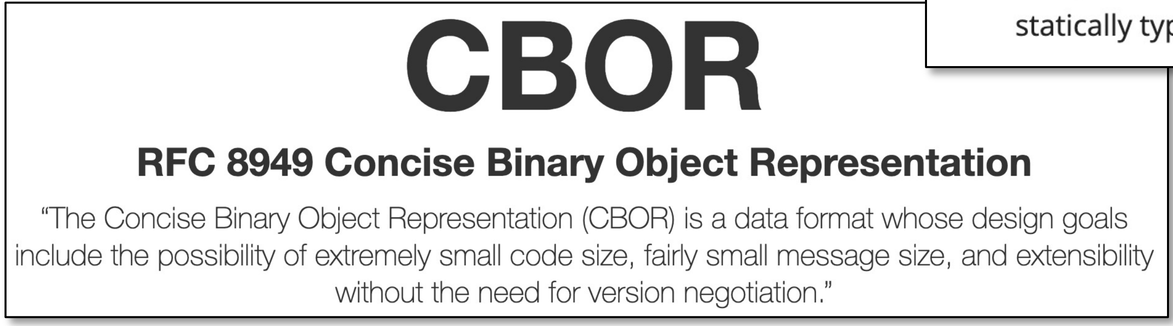
- RSocket is reactive point-to-point messaging framework designed to overcome RESTful limitations
  - It supports client/server programs that interact via Flux & Mono reactive types in Project Reactor
  - It can be configured to use various application-level binary protocols



Apache Avro™ is a data serialization system.

Avro provides:

- Rich data structures.
- A compact, fast, binary data format.
- A container file, to store persistent data.
- Remote procedure call (RPC).
- Simple integration with dynamic languages. Code generation is not required to read or write data files nor to use or implement RPC protocols. Code generation as an optional optimization, only worth implementing for statically typed languages.



CBOR

RFC 8949 Concise Binary Object Representation

"The Concise Binary Object Representation (CBOR) is a data format whose design goals include the possibility of extremely small code size, fairly small message size, and extensibility without the need for version negotiation."

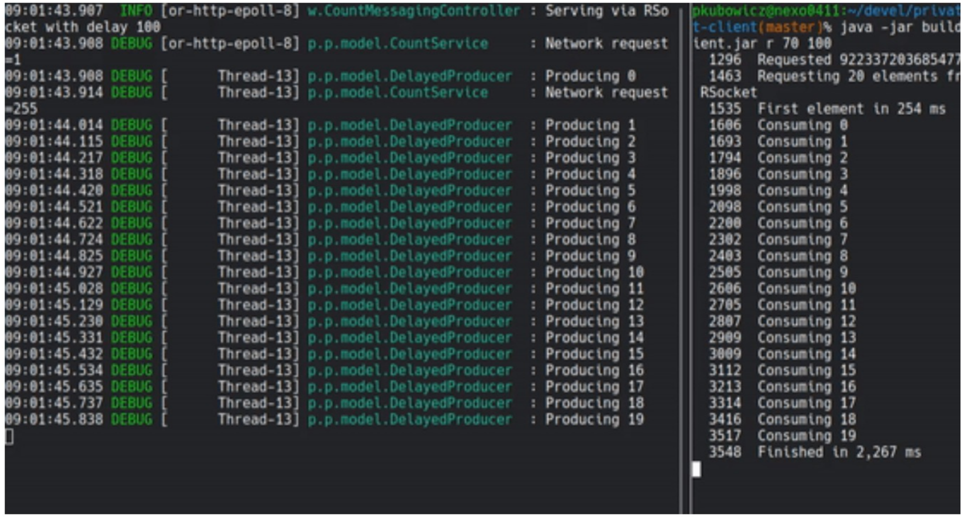
See en.wikipedia.org/wiki/CBOR & avro.apache.org/docs

Overview of RSocket

- RSocket is reactive point-to-point messaging framework designed to overcome RESTful limitations
  - It supports client/server programs that interact via Flux & Mono reactive types in Project Reactor
  - It can be configured to use various application-level binary protocols
    - These *may* be more efficient than other popular application-level protocols

Overview of RSocket

- RSocket is reactive point-to-point messaging framework designed to overcome RESTful limitations
  - It supports client/server programs that interact via Flux & Mono reactive types in Project Reactor
  - It can be configured to use various application-level binary protocols
    - These *may* be more efficient than other popular application-level protocols
      - e.g., HTTP using non-binary encodings like XML & JSon



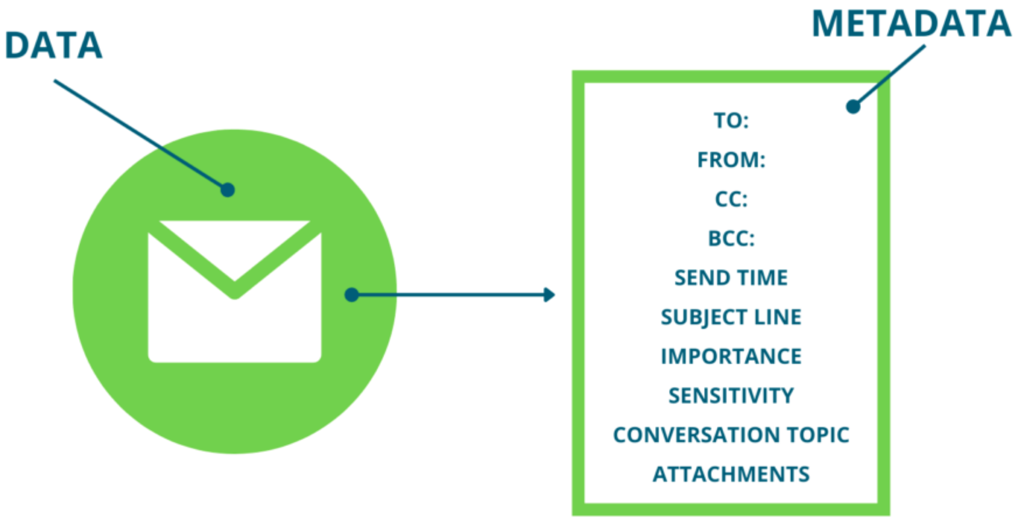
```
09:01:43.907  INFO [or-http-epoll-8] w.CountMessagingController : Serving via RSo
cket with delay 100
09:01:43.908 DEBUG [or-http-epoll-8] p.p.model.CountService     : Network request
=1
09:01:43.908 DEBUG [      Thread-13] p.p.model.DelayedProducer  : Producing 0
09:01:43.914 DEBUG [      Thread-13] p.p.model.CountService     : Network request
=255
09:01:44.014 DEBUG [      Thread-13] p.p.model.DelayedProducer  : Producing 1
09:01:44.115 DEBUG [      Thread-13] p.p.model.DelayedProducer  : Producing 2
09:01:44.217 DEBUG [      Thread-13] p.p.model.DelayedProducer  : Producing 3
09:01:44.318 DEBUG [      Thread-13] p.p.model.DelayedProducer  : Producing 4
09:01:44.420 DEBUG [      Thread-13] p.p.model.DelayedProducer  : Producing 5
09:01:44.521 DEBUG [      Thread-13] p.p.model.DelayedProducer  : Producing 6
09:01:44.622 DEBUG [      Thread-13] p.p.model.DelayedProducer  : Producing 7
09:01:44.724 DEBUG [      Thread-13] p.p.model.DelayedProducer  : Producing 8
09:01:44.825 DEBUG [      Thread-13] p.p.model.DelayedProducer  : Producing 9
09:01:44.927 DEBUG [      Thread-13] p.p.model.DelayedProducer  : Producing 10
09:01:45.028 DEBUG [      Thread-13] p.p.model.DelayedProducer  : Producing 11
09:01:45.129 DEBUG [      Thread-13] p.p.model.DelayedProducer  : Producing 12
09:01:45.230 DEBUG [      Thread-13] p.p.model.DelayedProducer  : Producing 13
09:01:45.331 DEBUG [      Thread-13] p.p.model.DelayedProducer  : Producing 14
09:01:45.432 DEBUG [      Thread-13] p.p.model.DelayedProducer  : Producing 15
09:01:45.534 DEBUG [      Thread-13] p.p.model.DelayedProducer  : Producing 16
09:01:45.635 DEBUG [      Thread-13] p.p.model.DelayedProducer  : Producing 17
09:01:45.737 DEBUG [      Thread-13] p.p.model.DelayedProducer  : Producing 18
09:01:45.838 DEBUG [      Thread-13] p.p.model.DelayedProducer  : Producing 19
```

```
pkubowicz@nexo0411:~/devel/privat
t-client(master)% java -jar build
ient.jar r 70 100
  1296  Requested 922337203685477
  1463  Requesting 20 elements fr
 RSocket
  1535  First element in 254 ms
  1606  Consuming 0
  1693  Consuming 1
  1794  Consuming 2
  1896  Consuming 3
  1998  Consuming 4
  2098  Consuming 5
  2200  Consuming 6
  2302  Consuming 7
  2403  Consuming 8
  2505  Consuming 9
  2606  Consuming 10
  2705  Consuming 11
  2807  Consuming 12
  2909  Consuming 13
  3009  Consuming 14
  3112  Consuming 15
  3213  Consuming 16
  3314  Consuming 17
  3416  Consuming 18
  3517  Consuming 19
  3548  Finished in 2,267 ms
```

See nexocode.com/blog/posts/rsocket-why

Overview of RSocket

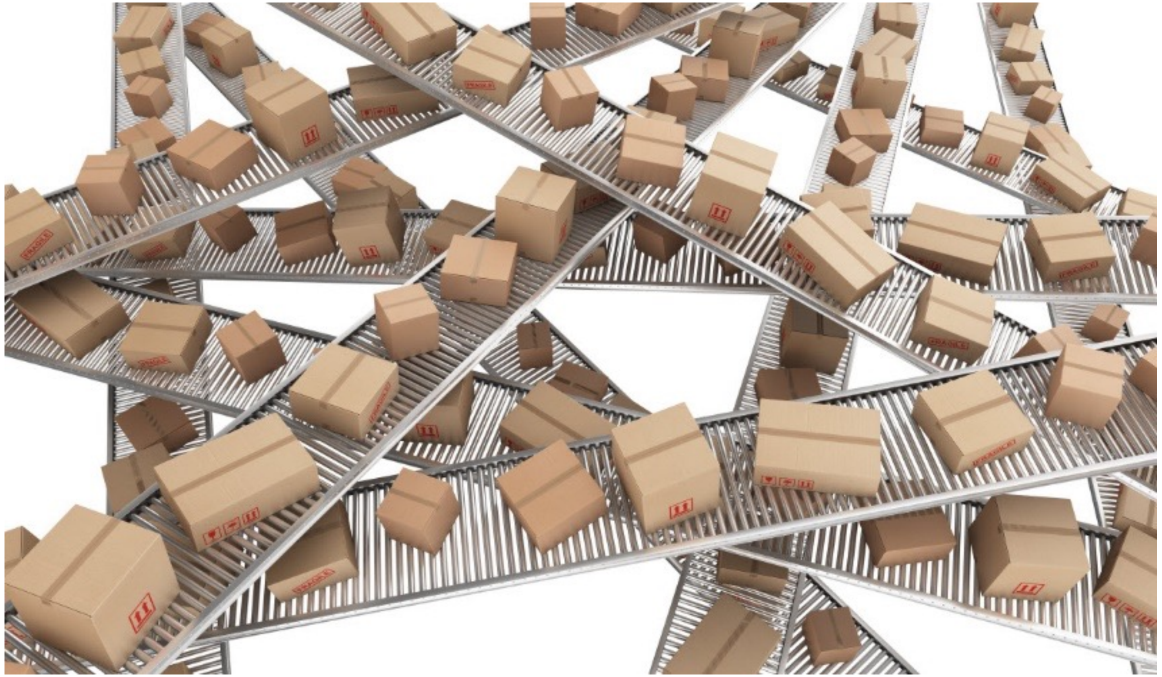
- RSocket messages contain metadata & data



See docs.spring.io/spring-framework/docs/5.3.5/reference/pdf/rsocket.pdf

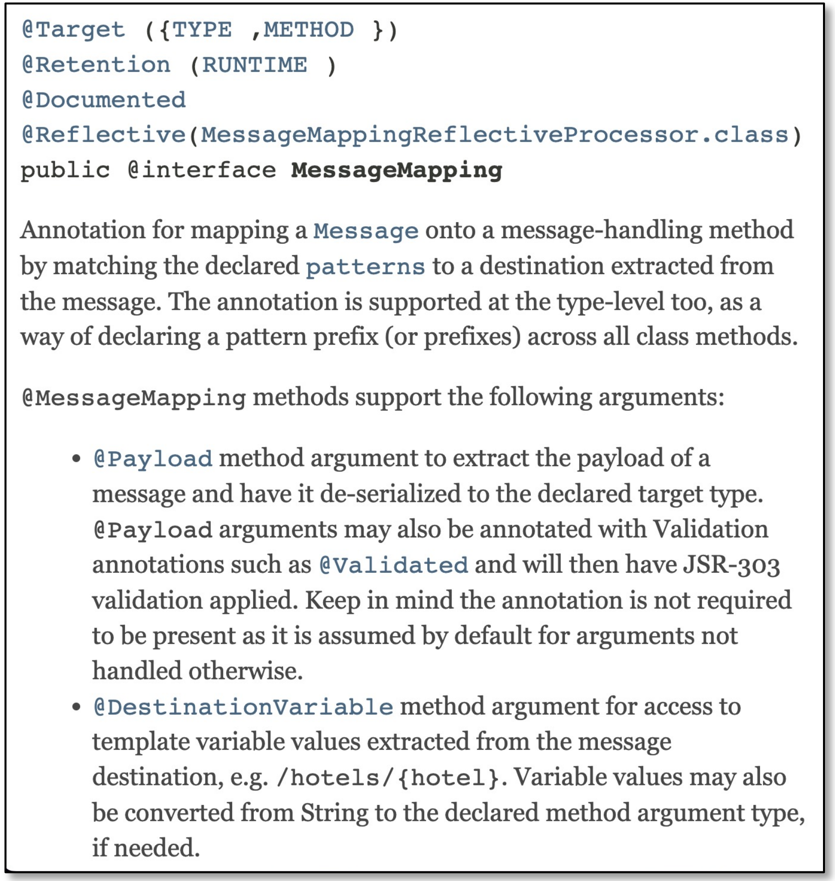
Overview of RSocket

- RSocket messages contain metadata & data
  - Metadata can select the route of a message



Overview of RSocket

- RSocket messages contain metadata & data
  - Metadata can select the route of a message
    - e.g., an endpoint specified via the @MessageMapping annotation in Spring



```
@Target ({TYPE ,METHOD })
@Retention (RUNTIME )
@Documented
@Reflective(MessageMappingReflectiveProcessor.class)
public @interface MessageMapping
```

Annotation for mapping a `Message` onto a message-handling method by matching the declared `patterns` to a destination extracted from the message. The annotation is supported at the type-level too, as a way of declaring a pattern prefix (or prefixes) across all class methods.

`@MessageMapping` methods support the following arguments:

- `@Payload` method argument to extract the payload of a message and have it de-serialized to the declared target type. `@Payload` arguments may also be annotated with Validation annotations such as `@Validated` and will then have JSR-303 validation applied. Keep in mind the annotation is not required to be present as it is assumed by default for arguments not handled otherwise.
- `@DestinationVariable` method argument for access to template variable values extracted from the message destination, e.g. `/hotels/{hotel}`. Variable values may also be converted from String to the declared method argument type, if needed.

See springframework/messaging/handler/annotation/MessageMapping.html

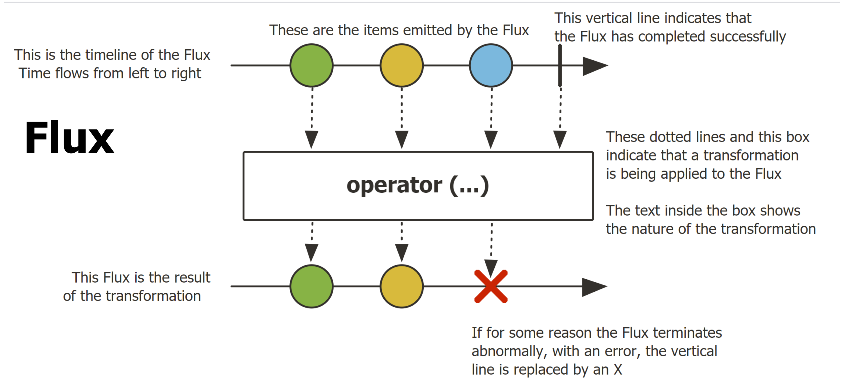
Overview of RSocket

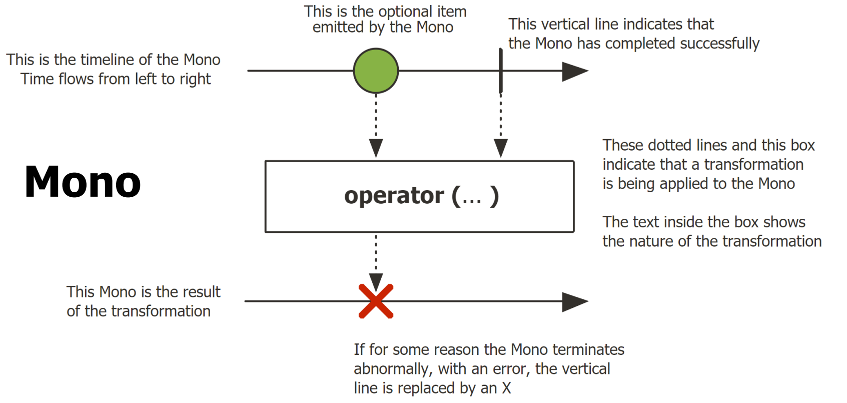
- RSocket messages contain metadata & data
  - Metadata can select the route of a message
  - Data contains the message payload

Overview of RSocket

- RSocket messages contain metadata & data
  - Metadata can select the route of a message
  - Data contains the message payload
    - e.g., specified via Mono or Flux reactive types

Project Reactor


These are the items emitted by the Flux
This vertical line indicates that the Flux has completed successfully
This is the timeline of the Flux Time flows from left to right
Flux
These dotted lines and this box indicate that a transformation is being applied to the Flux
operator (...)
The text inside the box shows the nature of the transformation
This Flux is the result of the transformation
If for some reason the Flux terminates abnormally, with an error, the vertical line is replaced by an X


This is the optional item emitted by the Mono
This vertical line indicates that the Mono has completed successfully
This is the timeline of the Mono Time flows from left to right
Mono
These dotted lines and this box indicate that a transformation is being applied to the Mono
operator (...)
The text inside the box shows the nature of the transformation
This Mono is the result of the transformation
If for some reason the Mono terminates abnormally, with an error, the vertical line is replaced by an X

See spring.io/blog/2016/04/19/understanding-reactive-types

End of Overview of RSocket