Spring Integration: Advanced Message Handling Using Routing and Transformations

SPLITTERS AND AGGREGATORS

Steven Haines
PRINCIPAL SOFTWARE ARCHITECT

@geekcap www.geekcap.com

Overview

Overview of Splitters and Aggregators and the Scatter-Gather Design Pattern

Splitters

Aggregators

Advanced Concepts

Message Splitter

A messaging component that partitions a message into several parts and sends the resulting messages to be processed independently

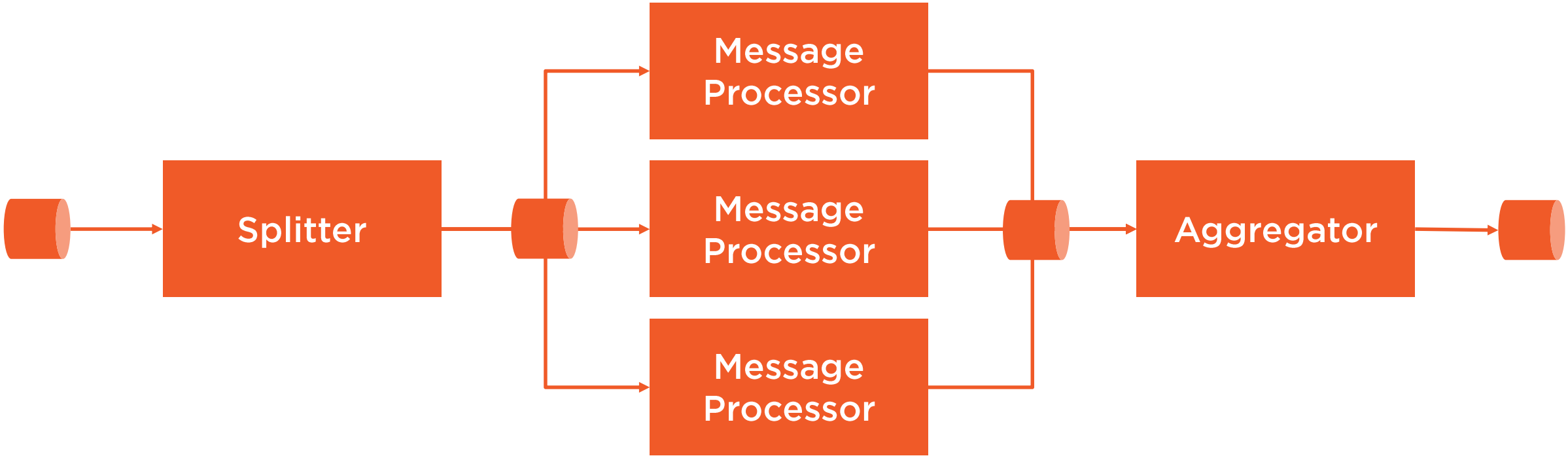
Message Aggregator

A messaging component that receives multiple messages and combines them into a single message

Scatter-Gather Design Pattern

The Scatter-Gather broadcasts a message to multiple recipients and re-aggregates the responses back into a single message

Scatter-Gather Design Pattern



Message Splitters

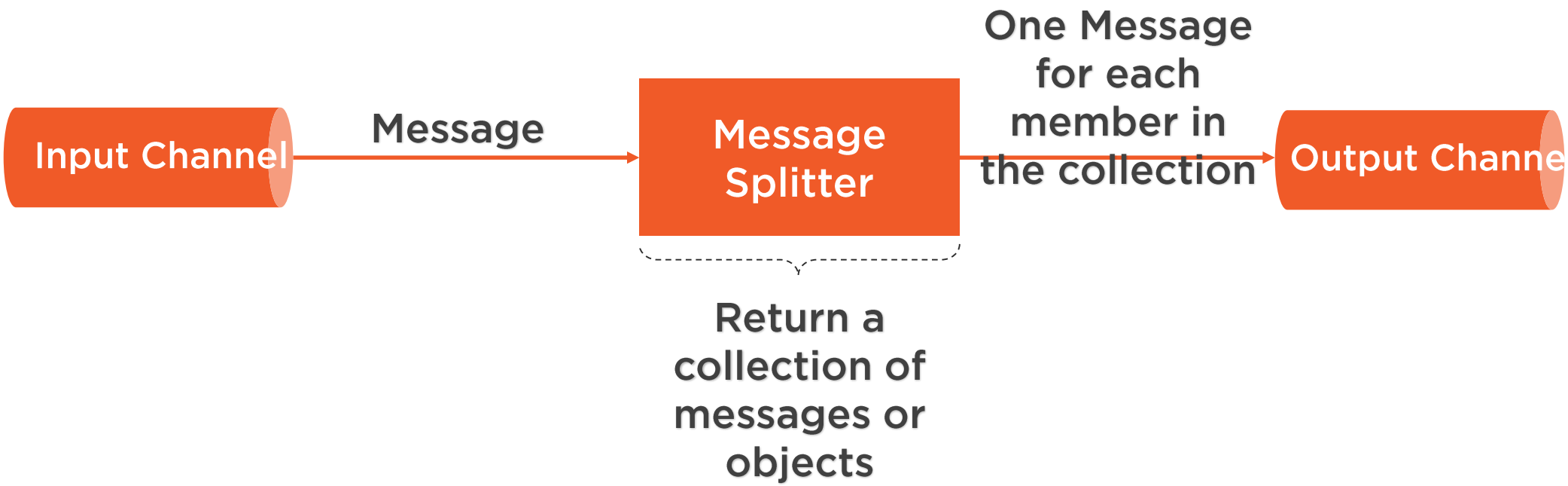
Message Splitter

A messaging component that partitions a message into several parts and sends the resulting messages to be processed independently

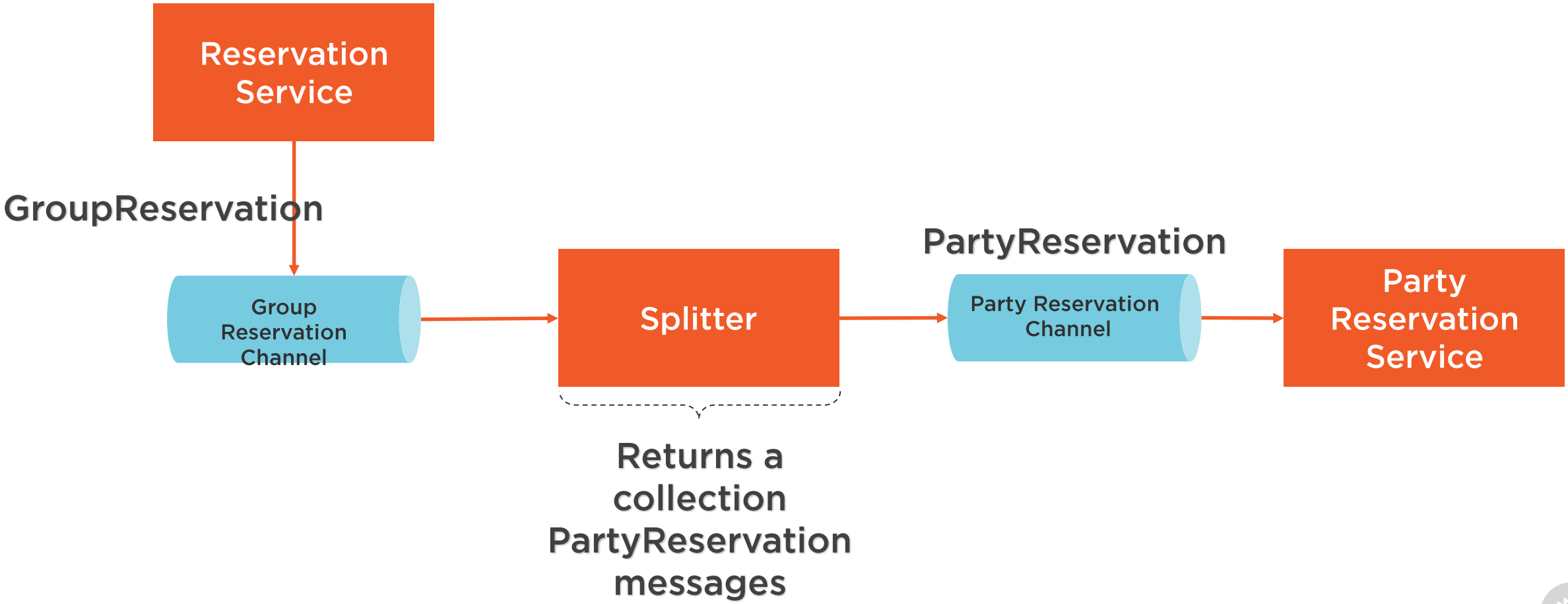
Use Case

You want to process the individual components of a message independently

How Message Splitters Work



Example: Group Reservations



```
@Configuration
@EnableIntegration
public class SplitterConfig {
    @Bean
    public MessageChannel
groupReservationChannel() {…}
    @Bean
    public MessageChannel
partyReservationChannel() {…}

   @Splitter(inputChannel =
"groupReservationChannel",
                  outputChannel =
"partyReservationChannel")
    public List<Message<PartyReservation>>
         splitter(Message<GroupReservation>
message) {

        GroupReservation gr =
message.getPayload();

        return gr.getParties().stream()
                .map(pr ->
MessageBuilder.withPayload(pr).build())
                .collect(Collectors.toList());
    }
```

- Setup a configuration class and enable Spring Integration
- Create channels
- Define a Splitter
- Return a list of messages with a PartyReservation payload

```
@Configuration
@EnableIntegration
public class SplitterConfig {
    @Bean
    public MessageChannel
groupReservationChannel() {…}
    @Bean
    public MessageChannel
partyReservationChannel() {…}

    @Splitter(inputChannel =
"groupReservationChannel",
                  outputChannel =
"partyReservationChannel")
    public List<PartyReservation> splitter(

GroupReservation groupReservation) {
        return groupReservation.getParties();
    }
}
```

- Setup a configuration class and enable Spring Integration
- Create channels
- Define a Splitter
- Return a list of PartyReservations

Demo

Define our components

- Two channels
- Splitter
- Reservation Service
- Party Reservation Service

Invoke the Reservation Service to publish group reservation message

Split the group reservation message into party reservation messages

Validate that the party reservation service receives each of our messages

Summary

A splitter is a messaging component that partitions a message into several parts and sends the resulting messages to be processed independently

Splitters are best used when you want to process the individual components of a message independently

Splitters and aggregators can work together

Next up: Aggregators

Aggregators

Message Aggregator

A messaging component that receives multiple messages and combines them into a single message

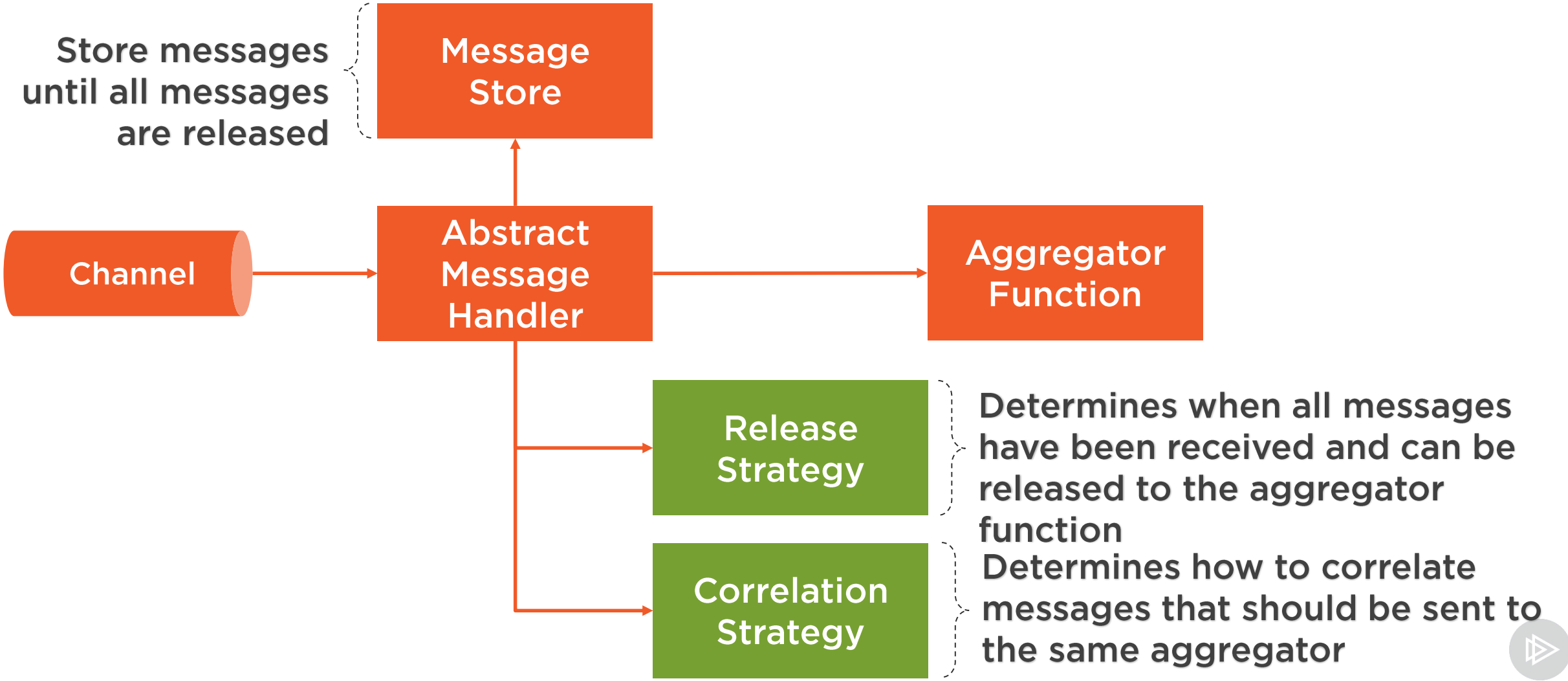
Use Case

You want to combine multiple messages into a single message

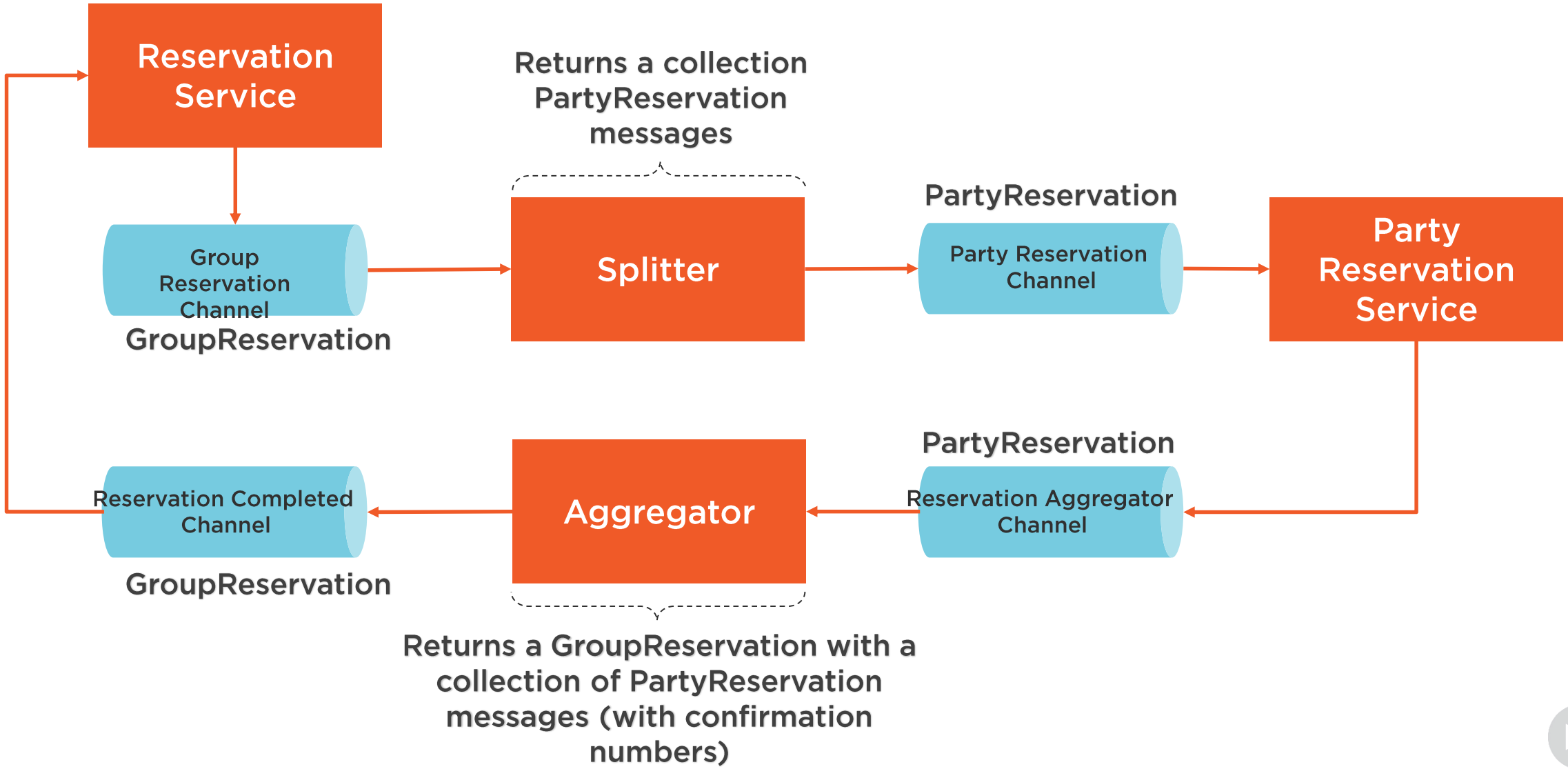
Scatter-Gather Design Pattern

The Scatter-Gather broadcasts a message to multiple recipients and re-aggregates the responses back into a single message

Aggregator Internals



Example: Group Reservations



```
@Configuration
@EnableIntegration
public class AggregatorConfig {
    @Bean
    public MessageChannel
groupReservationChannel() {…}
    @Bean
    public MessageChannel
partyReservationChannel() {…}
    @Bean
    public MessageChannel
reservationAggregatorChannel() {…}
    @Bean
    public MessageChannel
reservationCompletedChannel() {…}

   @Aggregator(inputChannel =
"reservationAggregatorChannel",
                  outputChannel =
"reservationCompletedChannel")
   public GroupReservation aggregator(
         List<Message<PartyReservation>>
partyReservations) {
            GroupReservation groupReservation =

new GroupReservation(…);
```

- Setup a configuration class and enable Spring Integration
- Create channels
- Define an Aggregator
- Receive a list of PartyReservations
- Create a GroupReservation
- Add all PartyReservations to the group reservation

Demo

Define our components

- Four channels
- Aggregator Function
- Reservation Aggregator Gateway
- Updated Services

Invoke the reservation service with a group reservation

See the message split and aggregated

Validate that the correct services received the correct messages

Summary

An aggregator is a messaging component that receives multiple messages and combines them into a single message

It allows several different messages to be combined into one and processed as one holistic unit

Next up: Advanced Aggregation

Aggregators: Advanced Features

Correlation Strategy

A Correlation Strategy determines how messages are grouped for aggregation

```
public interface CorrelationStrategy {

  Object getCorrelationKey(Message<?>
message);

}
```

Correlation Strategy

The Correlation Strategy determines a correlation key for the specified message

Release Strategy

A release strategy determines when a group of messages is complete and a sequence of individual messages can be released to an aggregator

```
public interface ReleaseStrategy {
  boolean canRelease(MessageGroup group);
}
```

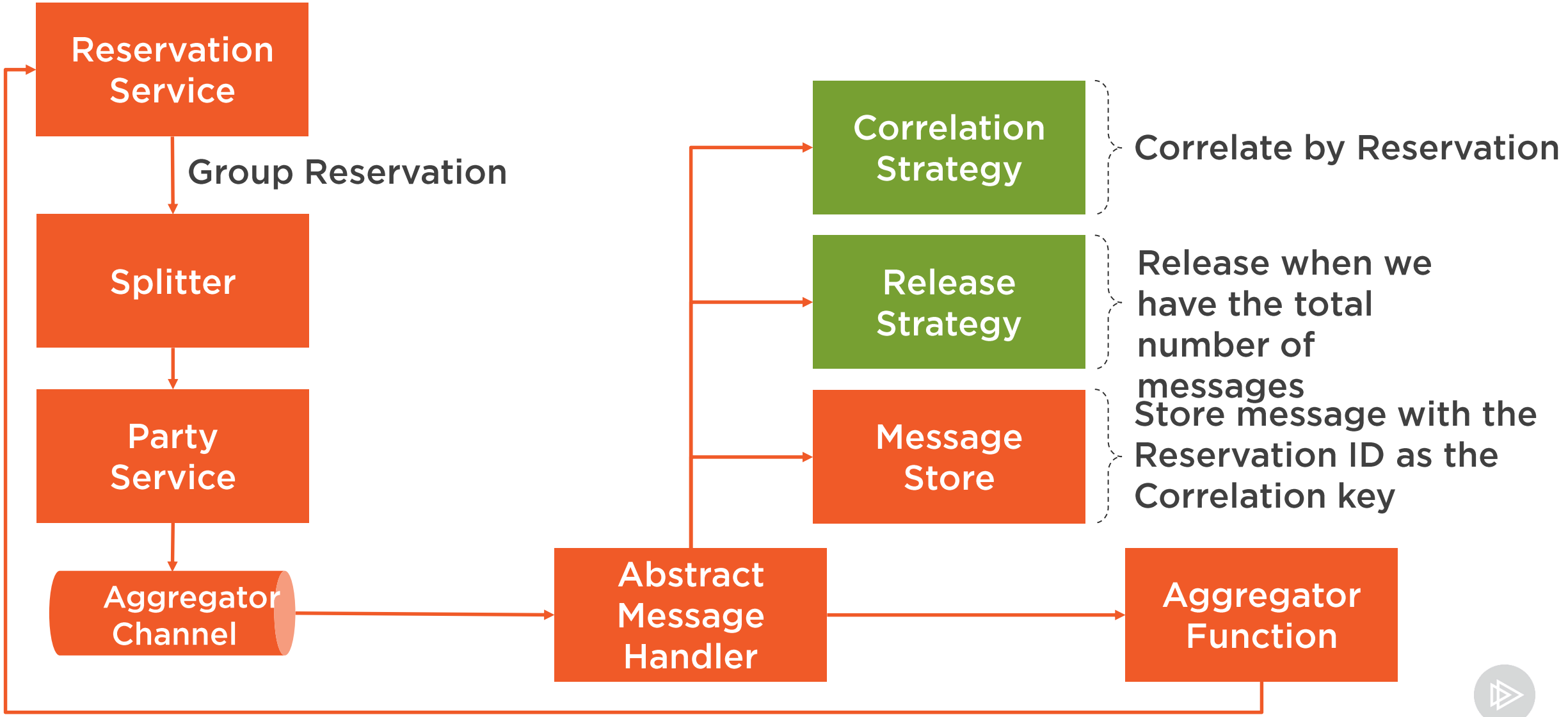
Release Strategy

The Correlation Strategy determines a correlation key for the specified message

Use Case

Handle multiple group reservations for the same reservation ID

Example: Multiple Group Reservations



```
@Component
public class PartyReservationAggregator {
   @CorrelationStrategy
    public Object correlateBy(
                    Message<PartyReservation>
partyReservation) {

        return (String)partyReservation

.getHeaders().get("RESERVATION_ID");
    }

    @ReleaseStrategy
    public boolean releaseChecker(

List<Message<PartyReservation>> messages) {

        int totalMessages =
(Integer)(messages.get(0)

.getHeaders().get("TOTAL_MESSAGES"));

        if (messages.size() >= totalMessages)
{
           return true;
        }
```

- Correlation Strategy
- Return the RESERVATION_ID from the message header
- Release Strategy
- Release the messages only if we have received the total number of messages from the TOTAL_MESSAGES header

```
@Component
public class PartyReservationAggregator {
 @Aggregator(
        inputChannel =
"reservationAggregatorChannel",
        outputChannel =
"reservationCompletedChannel")
    public GroupReservation aggregator(
         List<Message<PartyReservation>>
partyReservations) {

        GroupReservation groupReservation =
         new GroupReservation((String)
partyReservations.get(0)

.getHeaders().get("RESERVATION_ID"));

        groupReservation.getParties()

.addAll(partyReservations.stream()

.map(Message::getPayload)

.collect(Collectors.toList()));

        return groupReservation;
```

- Aggregator
- Receives a list of PartyReservation messages
- Create a new group reservation with the reservation ID from the PartyReservation message header
- Add all PartyReservations to the aggregated message
- Return the aggregated GroupReservation

Demo

Define our components

- Four channels
- Splitter
- Aggregator Component
- Payment and Reservation Services

Invoke the Reservation Service to publish multiple Group Reservations

Split and Aggregate the messages

Validate that the Reservation Service receives the aggregated message

Summary

A Correlation Strategy determines how messages are grouped for aggregation

A Release Strategy determines when a group of messages is complete and a sequence of individual messages can be released to an aggregator

Next up: Module Wrap-up

Conclusion

Message Splitter

A messaging component that partitions a message into several parts and sends the resulting messages to be processed independently

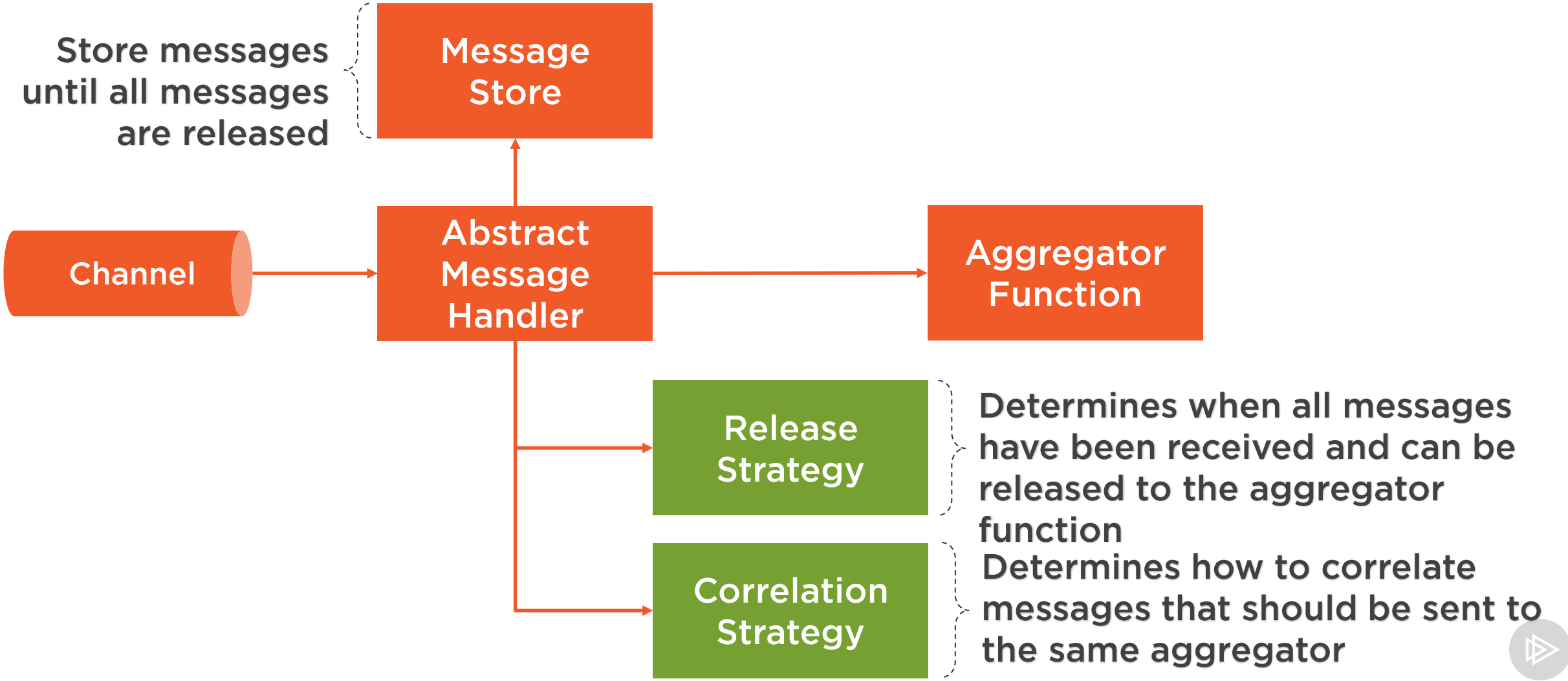
Message Aggregator

A messaging component that receives multiple messages and combines them into a single message

Scatter-Gather Design Pattern

The Scatter-Gather broadcasts a message to multiple recipients and re-aggregates the responses back into a single message

Aggregator Internals



Summary

You should understand splitters and aggregators

You should understand the scatter-gather design pattern

You should feel comfortable implementing them in your applications

Next Module: Message Transformation