Spring Integration: Advanced Message Handling Using Routing and Transformations

MESSAGE TRANSFORMATION

Steven Haines
PRINCIPAL SOFTWARE ARCHITECT

@geekcap www.geekcap.com

Overview

Message Transformer

Header Enrichers and Filters

Content Enrichers

Message Transformers

A messaging component that converts a message from one type to another. It is used to help loosely couple message producers and message consumers.

Header Enrichers and Filters

Messaging components that augment or remove message headers

Content Enrichers

A messaging component that adds information to a message

Message Transformers

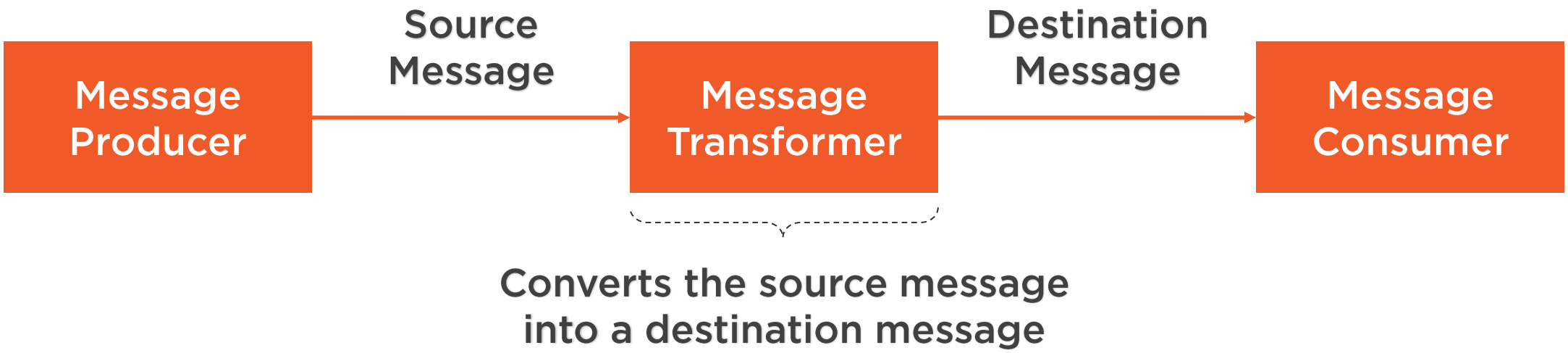
Message Transformers

A messaging component that converts a message from one type to another

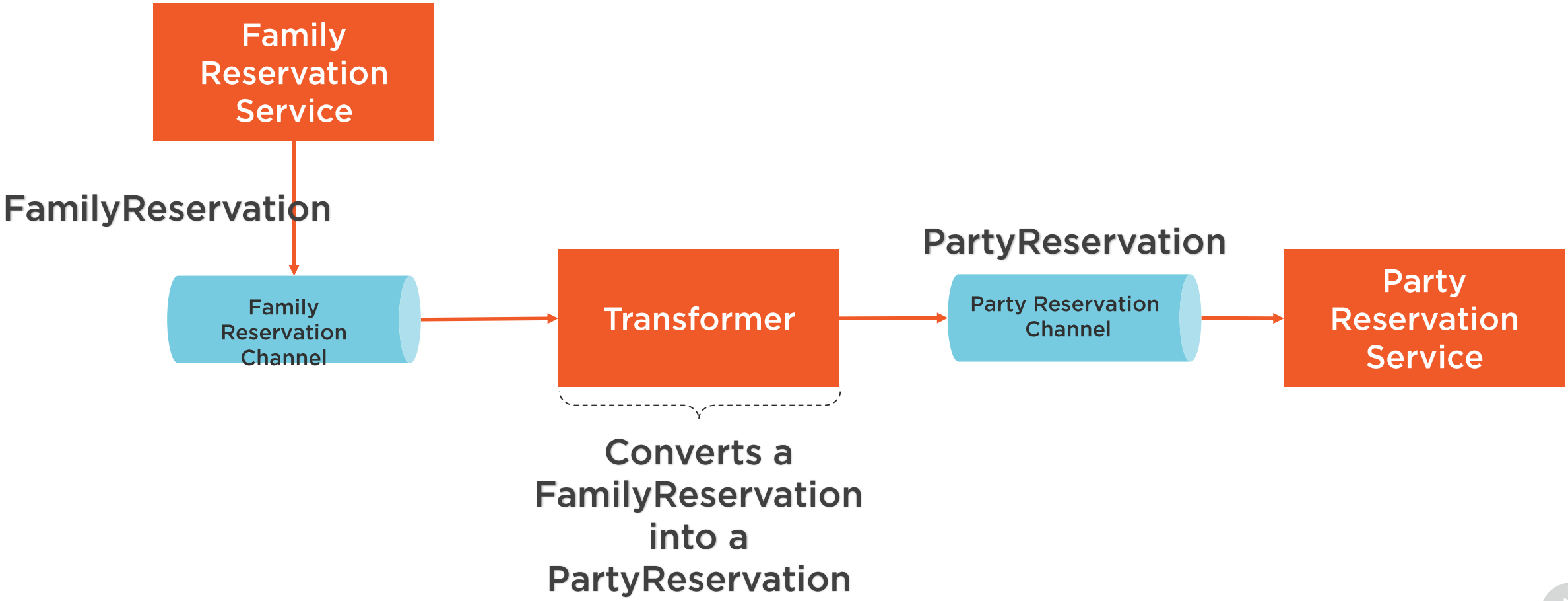
Use Case

You want to handle a message from a message producer with a message consumer that does not support the same message type

How Message Transformers Work



Example: Family Reservations



```
@Configuration
@EnableIntegration
public class TransformerConfig {
    @Bean
    public MessageChannel
familyReservationChannel() {…}
    @Bean
    public MessageChannel
partyReservationChannel() {…}

   @Transformer(inputChannel =
"familyReservationChannel",
                         outputChannel =
"partyReservationChannel")
    public PartyReservation transform(

FamilyReservation familyReservation) {
      return new PartyReservation(

familyReservation.getFamilyId(),

familyReservation.getRoomType(),

familyReservation.getName());
    }
}
```

- Setup a configuration class and enable Spring Integration
- Create channels
- Define a Transformer
- Return a new PartyReservation from the FamilyReservation

Demo

Define our components

- Two channels
- Transformer
- Family Reservation Service
- Party Reservation Service

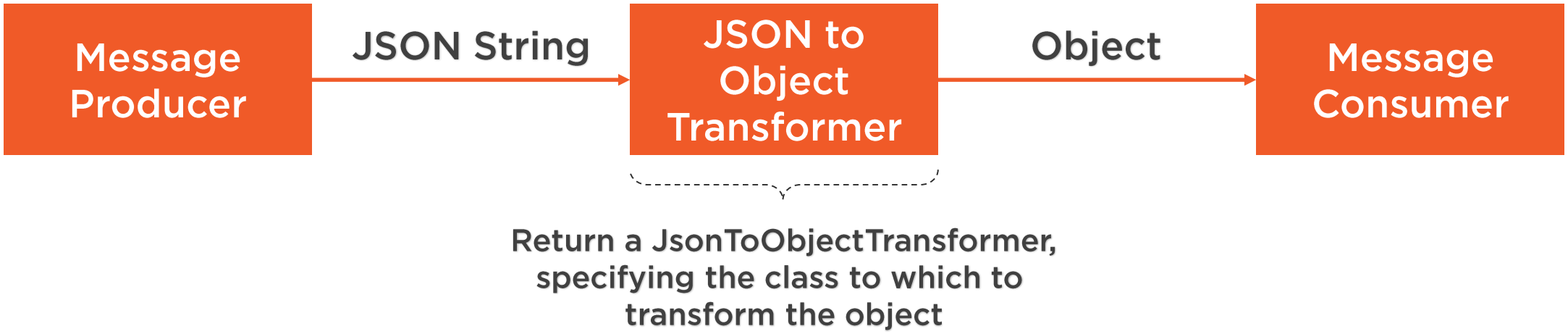
Invoke the Family Reservation Service to publish family reservation message

Transform the family reservation into a party reservation and publish it to the party reservation service

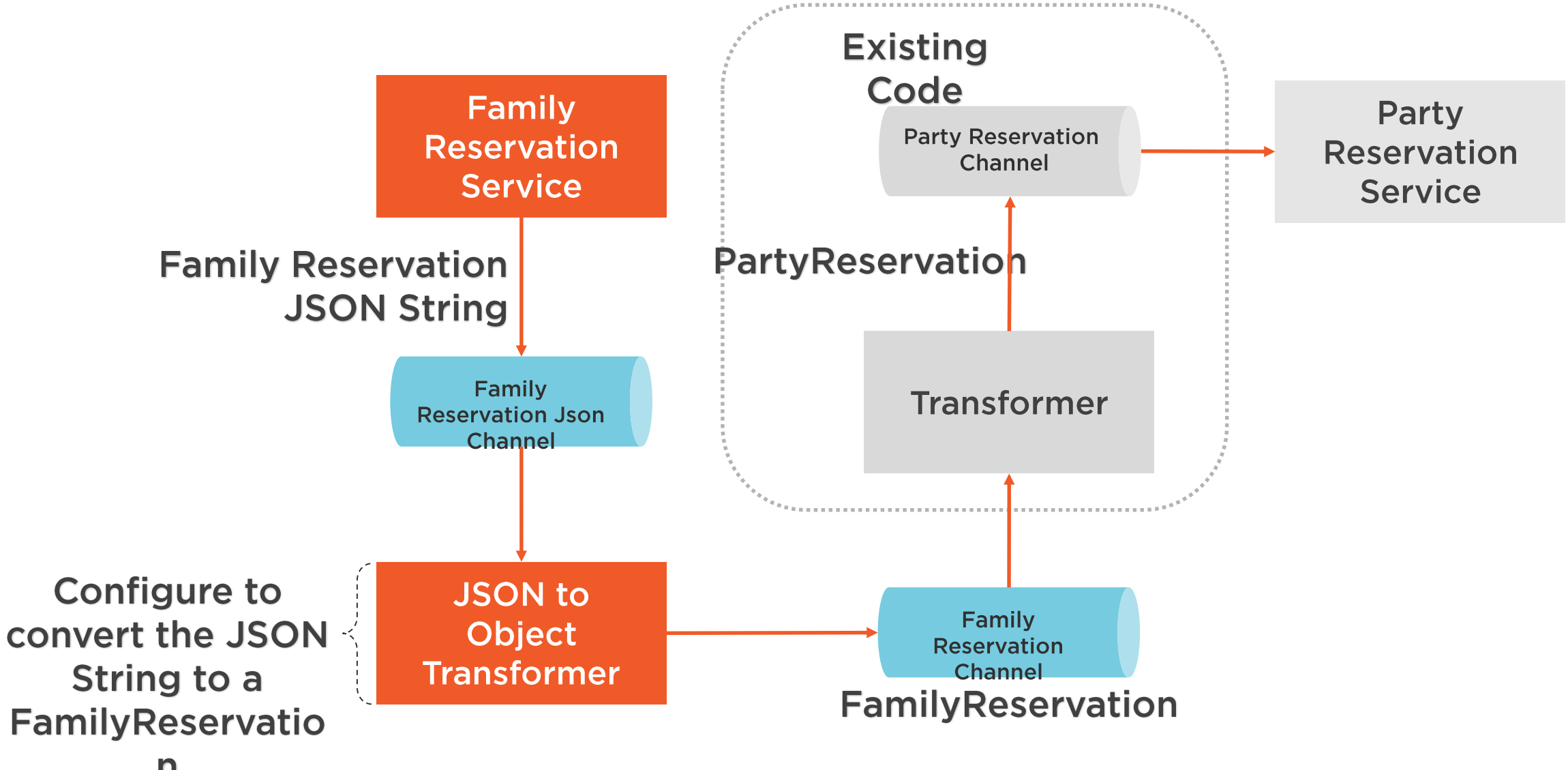
Validate that the party reservation service receives the message

JSON to Object Transformer

How JSON to Object Transformers Work



Example: Family Reservations (as JSON)



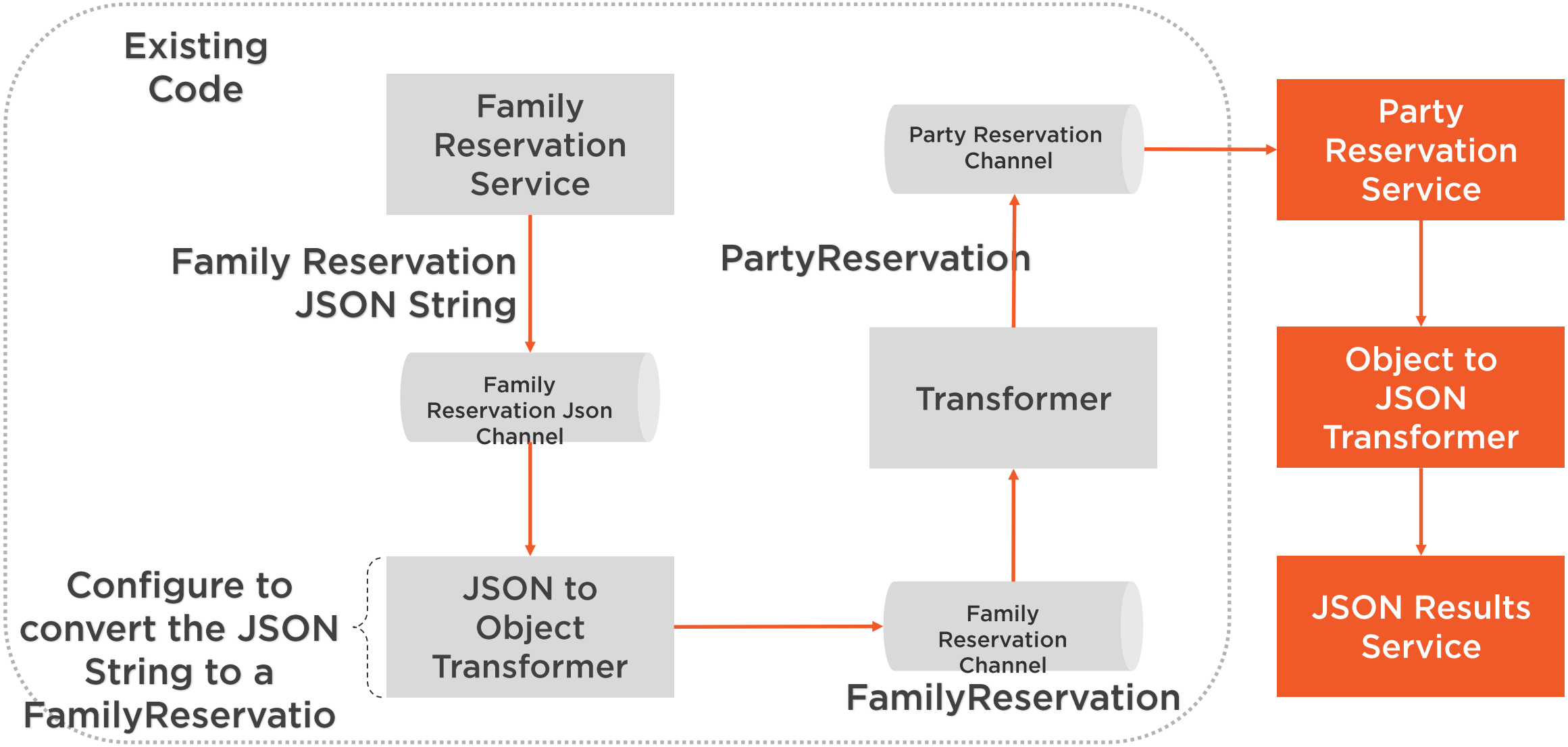
```
@Configuration
@EnableIntegration
public class TransformerConfig {
    @Bean
    public MessageChannel
familyReservationChannel() {}
    @Bean
    public MessageChannel
familyReservationJsonChannel() {}
    @Bean
    public MessageChannel
partyReservationChannel() {}

    @Bean
    @Transformer(
                    inputChannel =
"familyReservationJsonChannel",
                    outputChannel =
"familyReservationChannel")
    public JsonToObjectTransformer
transformer() {
        return new

JsonToObjectTransformer(FamilyReservation.clas
s);
    }
```

- Setup a configuration class and enable Spring Integration
- Create channels
- Define a Transformer
- Return a new JsonToObjectTransformer instance, specifying that it should convert the JSON input into a FamilyReservation instance

Object to JSON Transformer

How Object to JSON Transformers Work

Example: JSON Results



```
@Configuration
@EnableIntegration
public class TransformerConfig {
    @Bean
    public MessageChannel toJsonChannel() {}
    @Bean
    public MessageChannel jsonResultsChannel()
{}

    @Bean
    @Transformer(
                  inputChannel =
"toJsonChannel",
                  outputChannel =
"jsonResultsChannel")
    public ObjectToJsonTransformer
transformer() {
        return new ObjectToJsonTransformer();
    }
}
```

- Setup a configuration class and enable Spring Integration
- Create channels
- Define a Transformer
- Return a new ObjectToJsonTransformer instance

Summary

A transformer is a messaging component that converts a message from one type to another

We reviewed how to create a transformer manually

We saw the built-in JsonToObjectTransformer and ObjectToJsonTransformer

Next up: Header Enrichers and Filters

Header Enrichers and Filters

Header Enricher

A messaging component that adds values to a message's header

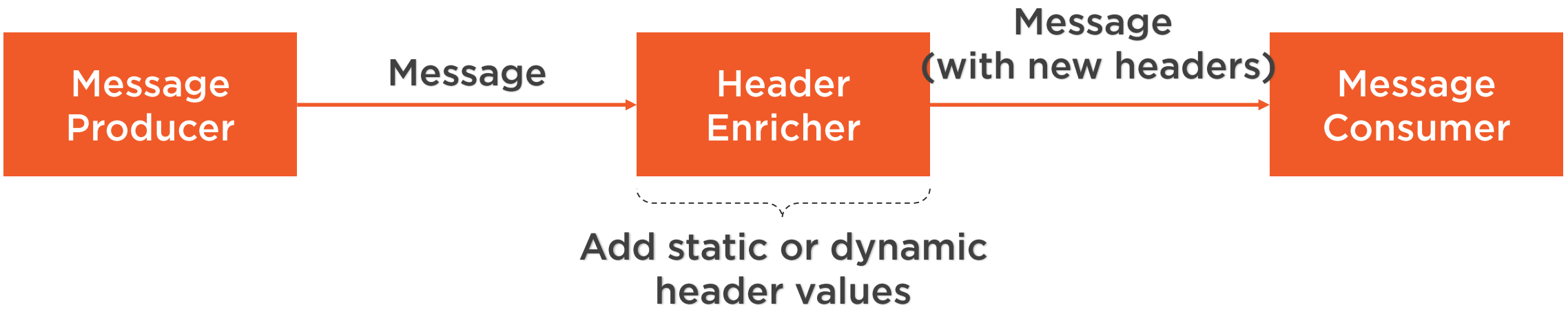
Header Filter

A messaging component that removes values from a message's header

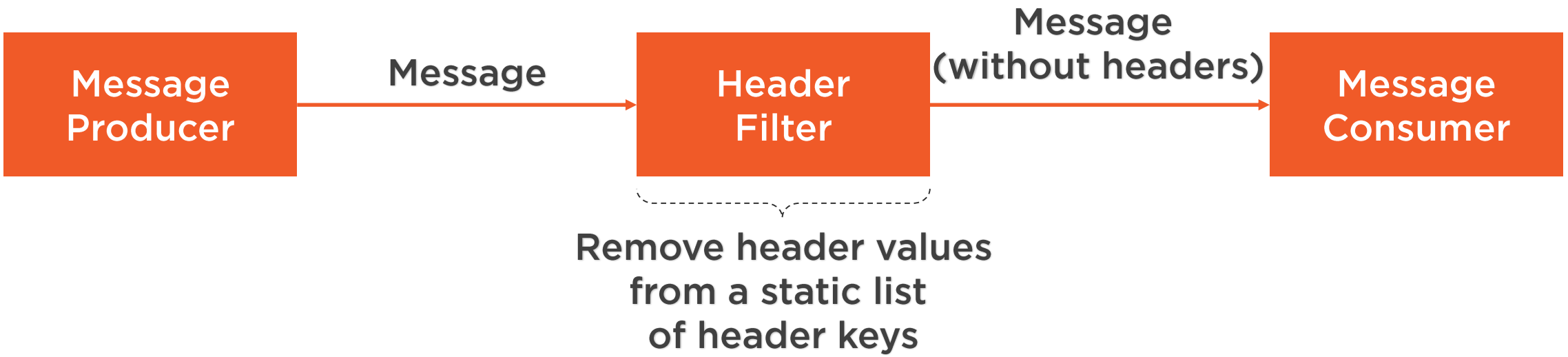
Use Case

You want to add or remove information from a message's header, such as adding an authorization token before making a request and removing it afterwards

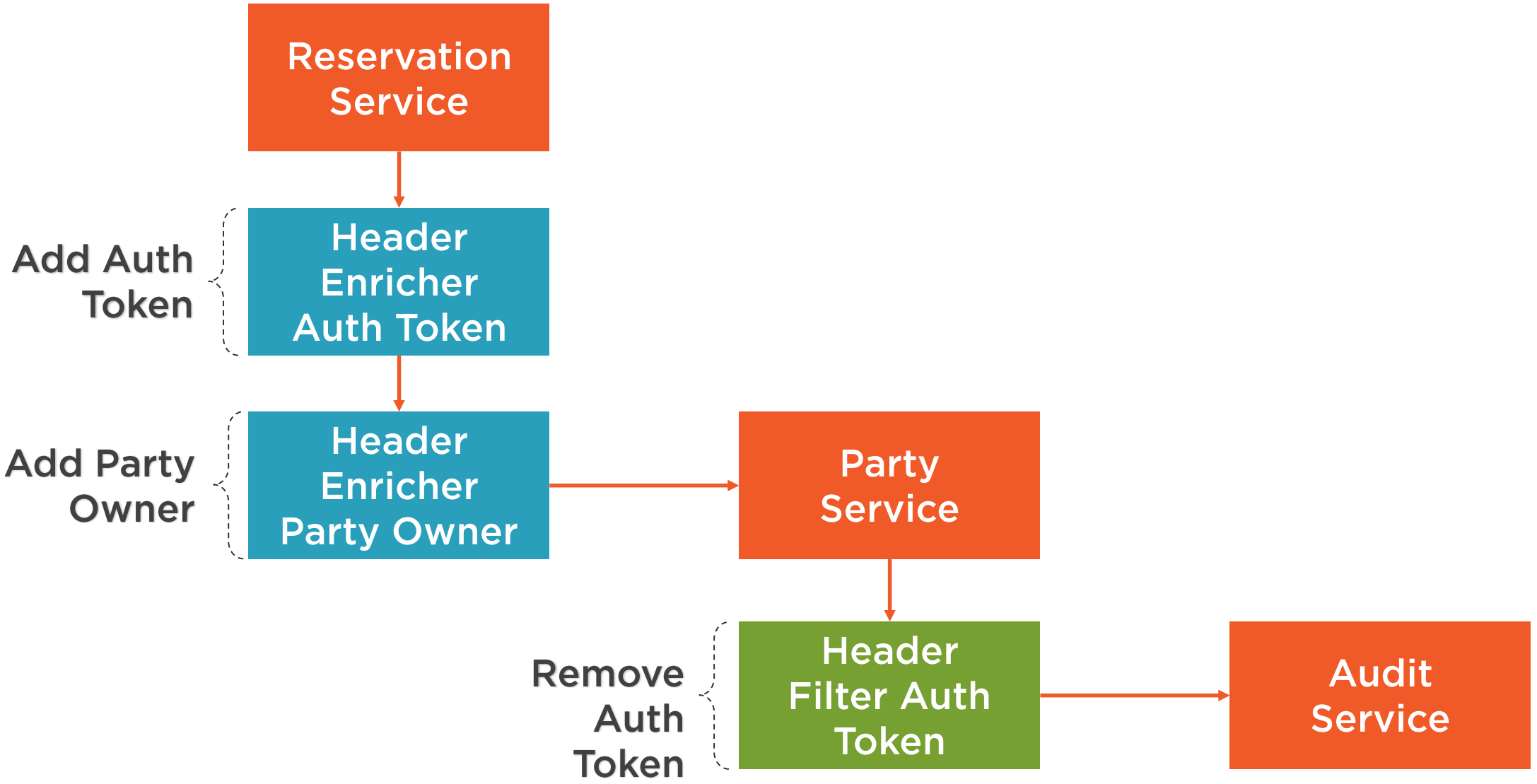
How Header Enrichers Work



How Header Filters Work



Example: Header Authorization



```
@Bean
@Transformer(inputChannel = "headerEnricherAddAuthChannel",
                          outputChannel = headerEnricherPartyOwnerChannel")
public HeaderEnricher headerEnricherAuthToken() {

        return new HeaderEnricher(Collections.singletonMap(
                          "AUTH_TOKEN", new
StaticHeaderValueMessageProcessor<>("12345")));
}
```

Static Header Enricher

Create a bean annotated with the @Transformer annotation

Return a new HeaderEnricher mapping the “AUTH_TOKEN” to the static value “12345”

```
 @Bean
 @Transformer(inputChannel = "headerEnricherPartyOwnerChannel",
                          outputChannel = "partyReservationChannel")
 public HeaderEnricher headerEnricherPartyOwner() {
        Expression expression = new
SpelExpressionParser().parseExpression("payload.partyId");

        return new HeaderEnricher(Collections.singletonMap(
                "PARTY_OWNER",
                new ExpressionEvaluatingHeaderValueMessageProcessor<>(expression,
String.class)));
}
```

Dynamically Derived Header Enricher

Create a bean annotated with the @Transformer annotation

Create a SpEL expression to extract the payload's partyId property

Return a new HeaderEnricher mapping the partyId SpEL expression to the PARTY_OWNER header

```
@Bean
@Transformer(inputChannel = "headerEnricherRemoveAuthChannel",
                          outputChannel = "auditChannel")
public HeaderFilter filterAuthToken() {
        return new HeaderFilter("AUTH_TOKEN");
}
```

Header Filter

Create a bean annotated with the @Transformer annotation

Return a new HeaderFilter with a list of header keys to remove from the message

Demo

Define our components

- Five channels
- Header Enrichers and Filter
- Gateways
- Services

Invoke the Reservation Service to publish a party reservation message

Manipulate headers and route messages

Validate the results

Summary

A Header Enricher is a messaging component that adds values to a message's header

A Header Filter is a messaging component that removes values from a message's header

Used when you want to add or remove information from a message's header

Next up: Content Enrichers

Content Enrichers

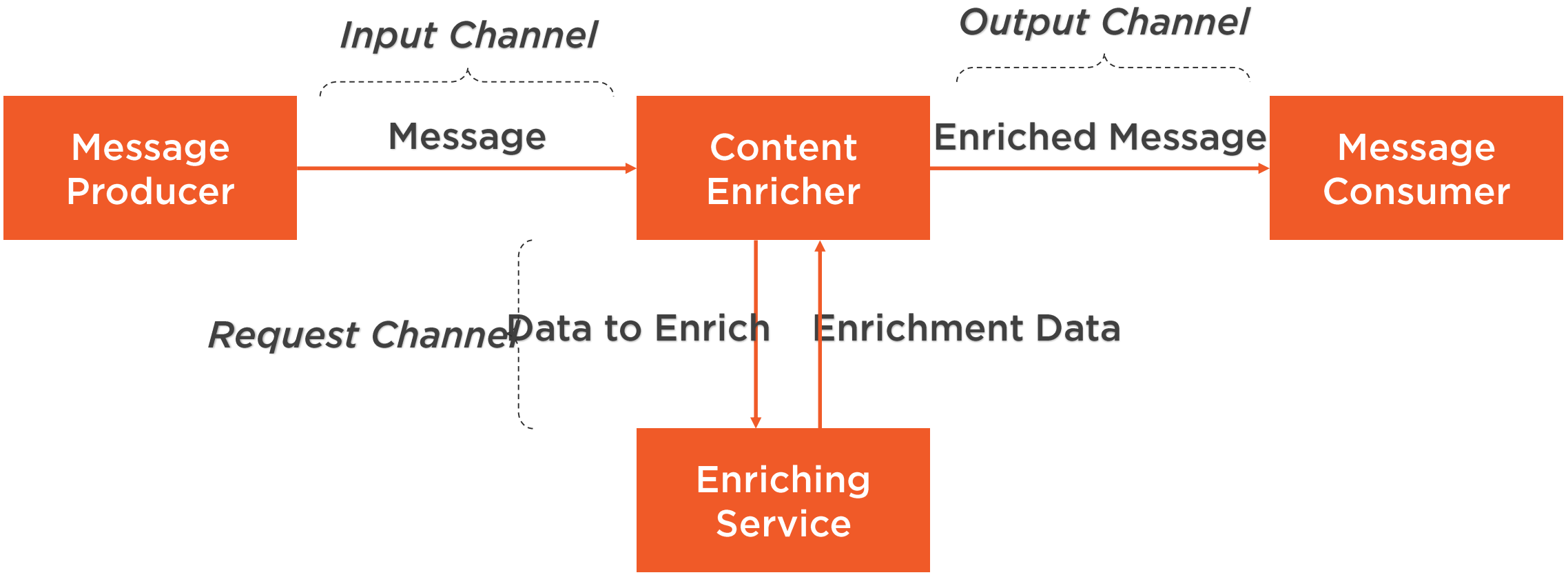
Content Enricher

A messaging component that adds additional information to a message payload

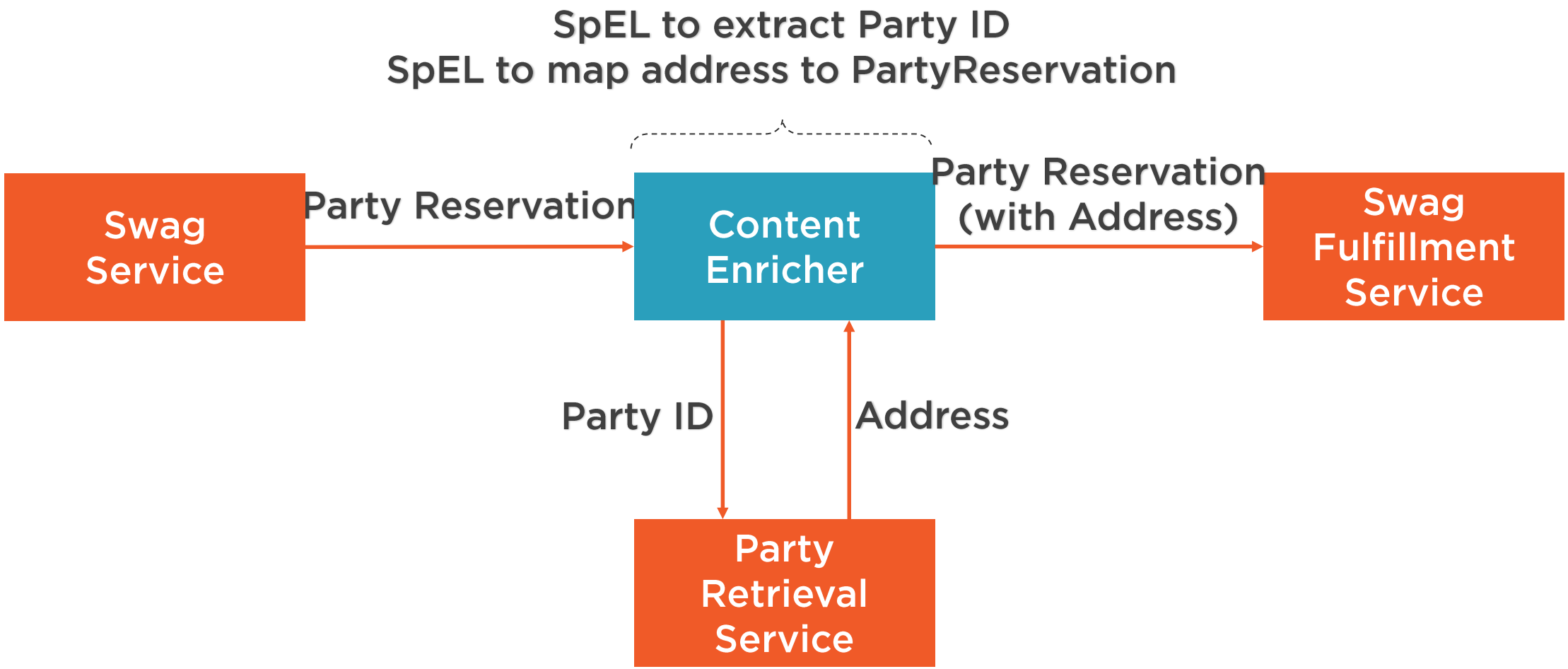
Use Case

A target system needs more information than a source system can provide

How Content Enrichers Work



Example: Sending a Swag Package



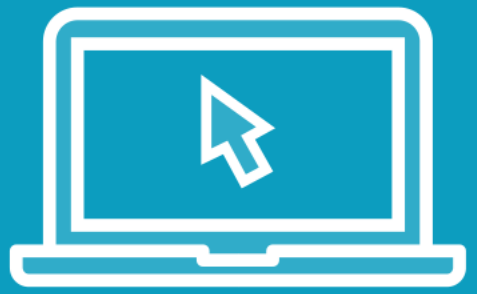
```
@ServiceActivator(inputChannel = "swagChannel")
public ContentEnricher contentEnricher() {
    ContentEnricher contentEnricher = new ContentEnricher();
    contentEnricher.setOutputChannelName("swagFulfillmentServiceChannel");
    contentEnricher.setRequestChannelName("partyReservationEnricherChannel");

contentEnricher.setRequestPayloadExpression(spelExpressionParser.parseExpression("payload.par
tyId"));
    contentEnricher.setPropertyExpressions(ImmutableMap.of(
                "address", spelExpressionParser.parseExpression("payload.address"),
                "city", spelExpressionParser.parseExpression("payload.city"),
                "state", spelExpressionParser.parseExpression("payload.state"),
                "zip", spelExpressionParser.parseExpression("payload.zip")));
    return contentEnricher;
}
```

Content Enricher

Create a bean annotated with the @ServiceActivator annotation

Return a new ContentEnricher with a specified output channel, request channel, and SpEL expressions that specify how to extract message fields

Demo



Define our components

- Three channels
- Content Enricher
- Gateway
- Services

Invoke the Swag Service with a party reservation

Enrich the payload

Validate that the Swag Fulfillment Service receives a party reservation with a populated address

Summary

A content enricher is a messaging component that adds additional information to a message payload

It's used when a target system needs more information than a source system can provide

Next up: Module Wrap-up

Conclusion

Message Transformers

A messaging component that converts a message from one type to another. It is used to help loosely couple message producers and message consumers.

Header Enrichers and Filters

Messaging components that augment or remove message headers

Content Enrichers

A messaging component that adds information to a message

Summary

You should understand message transformers

You should understand how to manage message headers using enrichers and filters

You should understand the role of content enrichers and how to use them

You should feel comfortable implementing them in your applications