Event Driven Microservices with Spring Cloud Stream

Toshiaki Maki (@making)

2016-12-03 JJUG CCC 2016 Fall

#jjug_ccc #ccc_ab3

http://bit.ly/making_ccc_a3 (source code)

Pivotal

Who am I ?

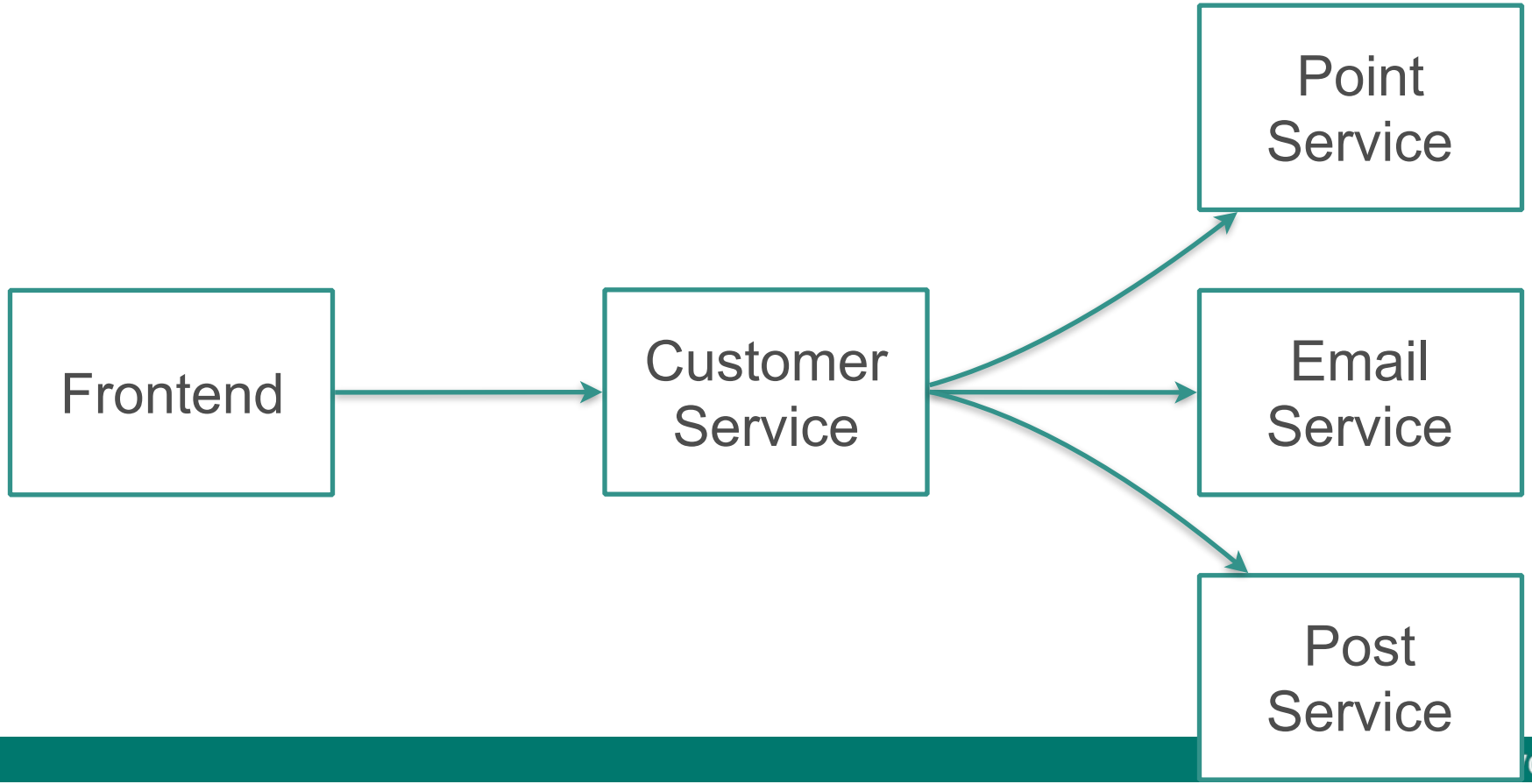
- Toshiaki Maki (@making) http://blog.ik.am
- Sr. Solutions Architect @Pivotal
- Spring Framework enthusiast

bit.ly/hajiboot2

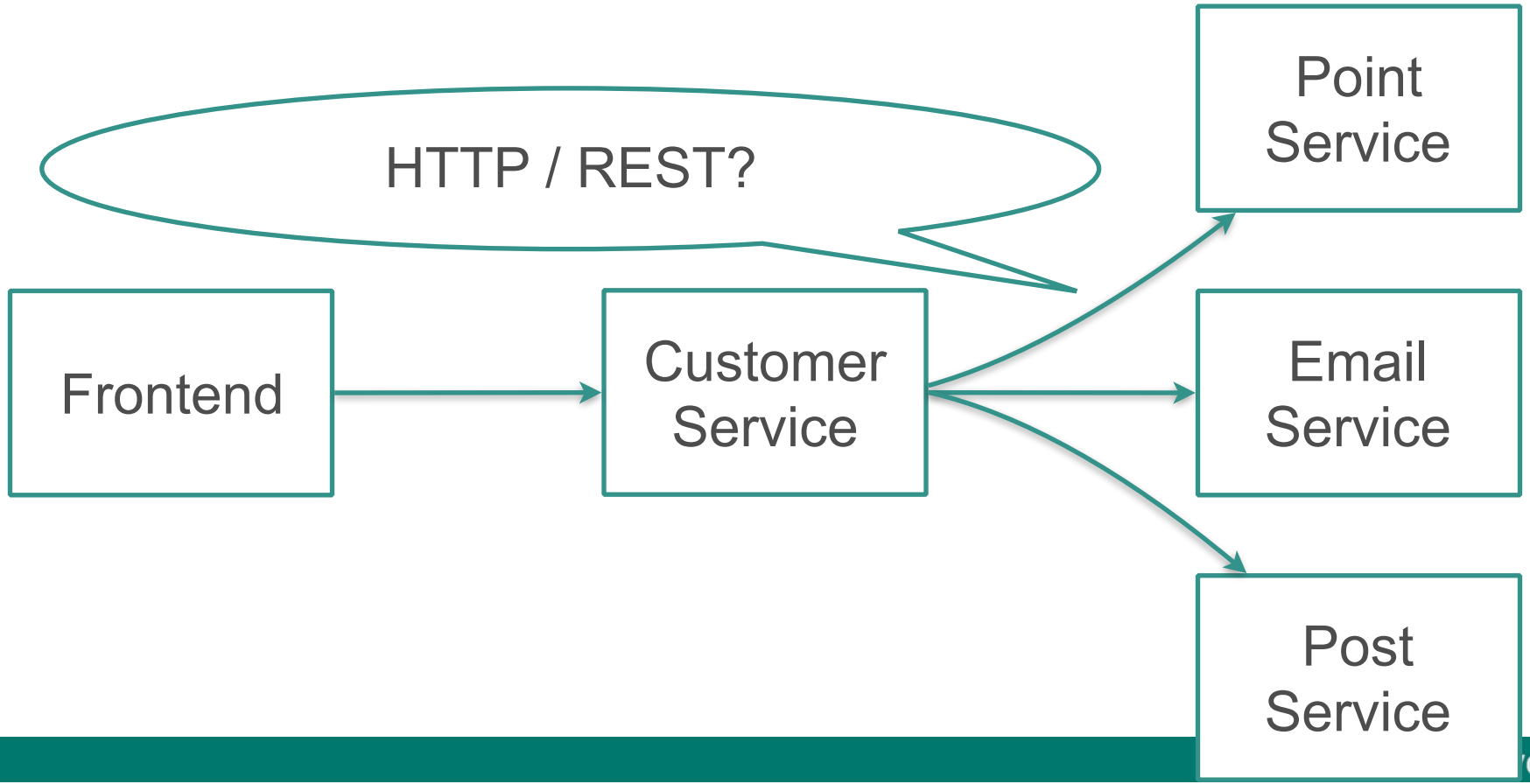
Contents

- Spring Cloud Stream (25min)
- Advanced Topic (20min)
- Spring Cloud Data Flow (2min)
- Deploy Stream Apps to Cloud Foundry (3min)

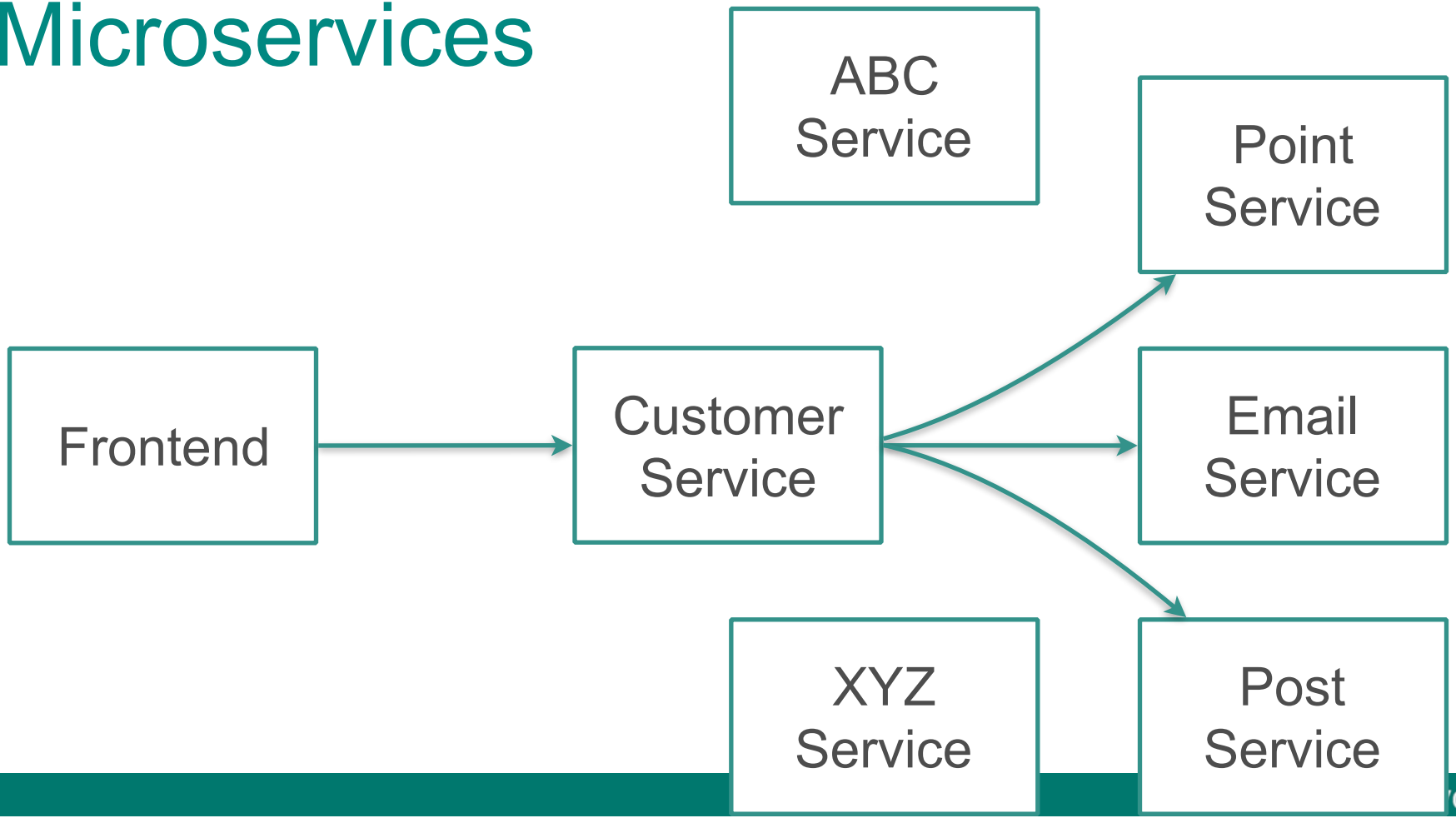
Microservices



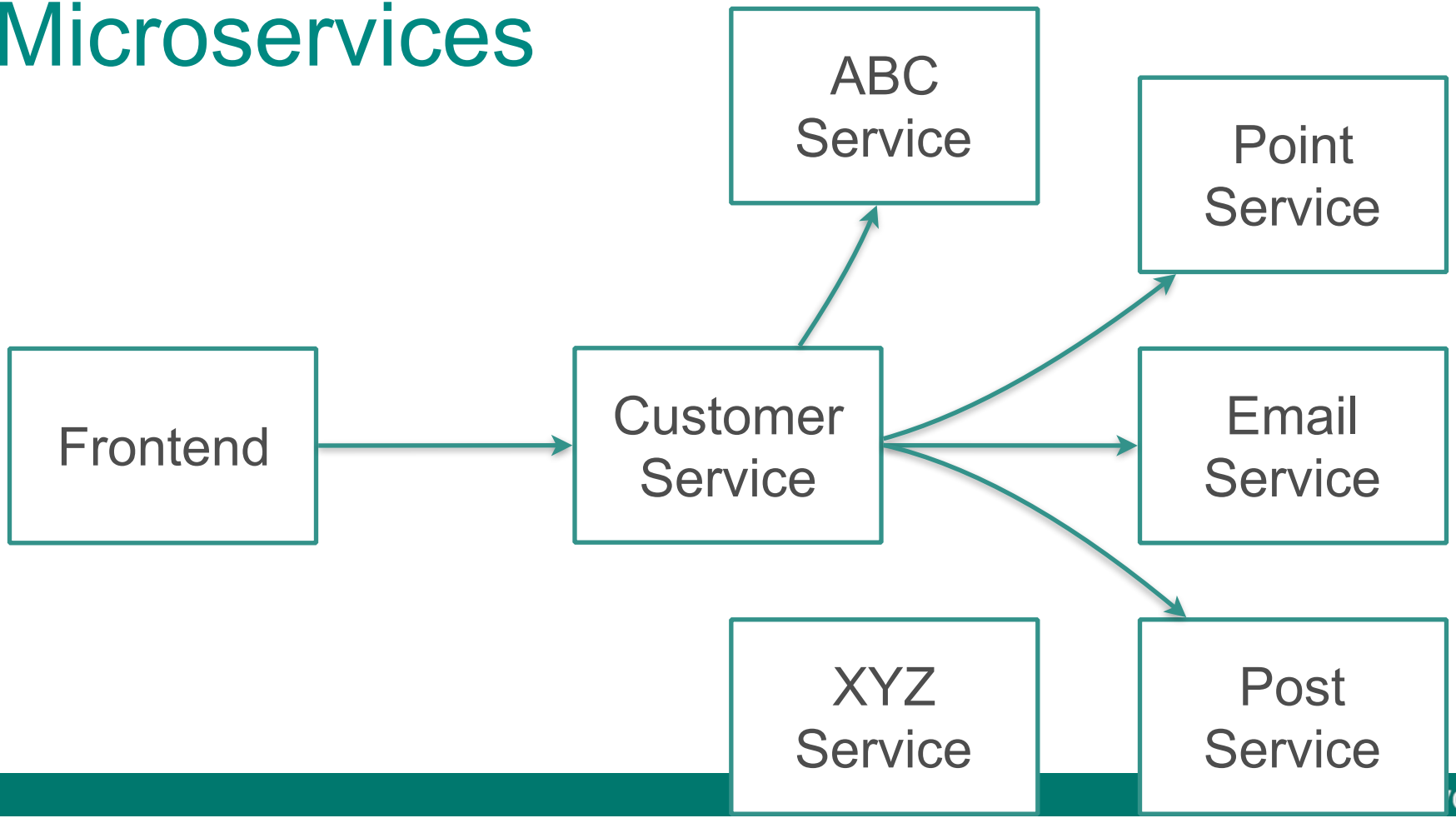
Microservices



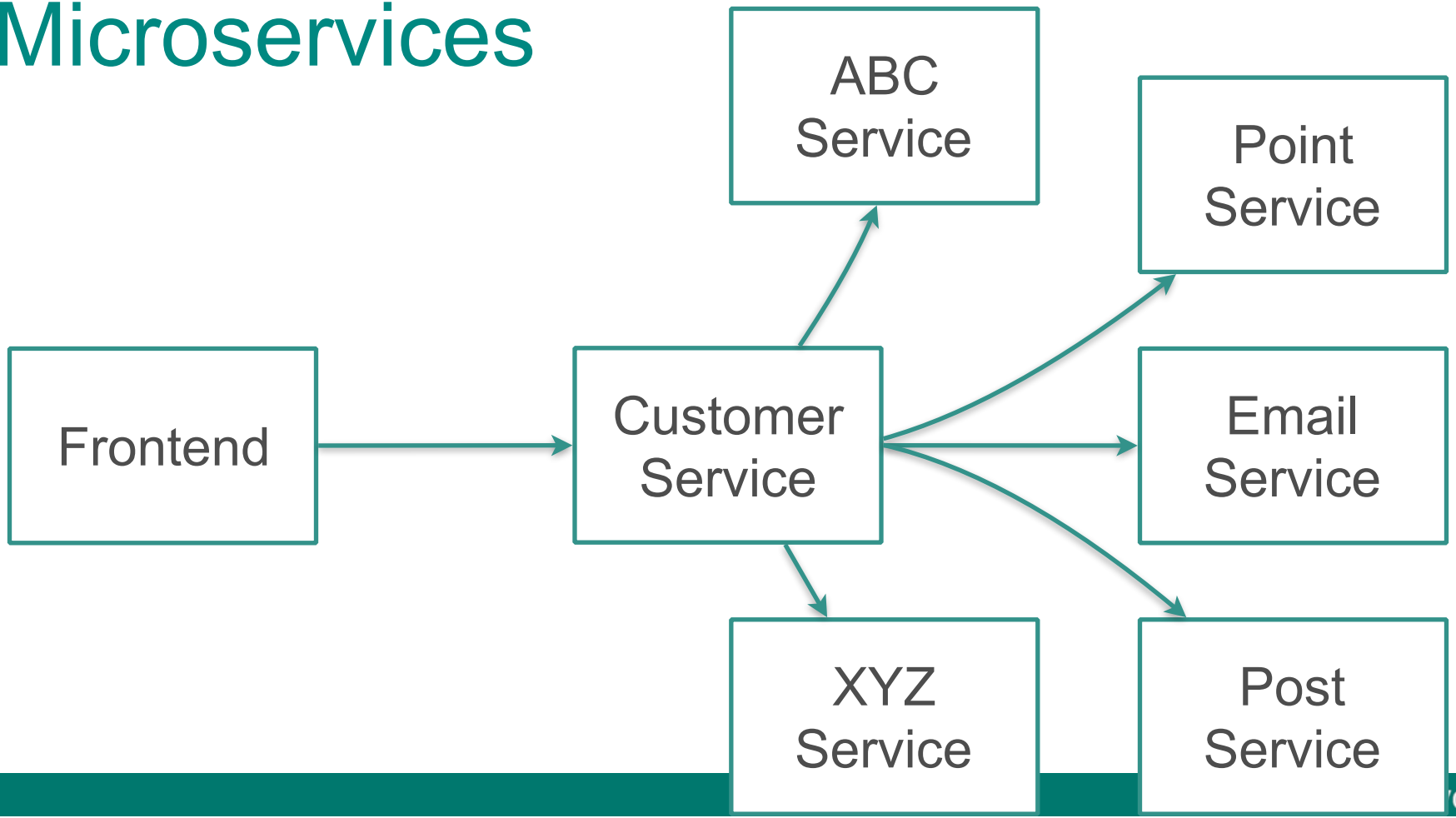
Microservices



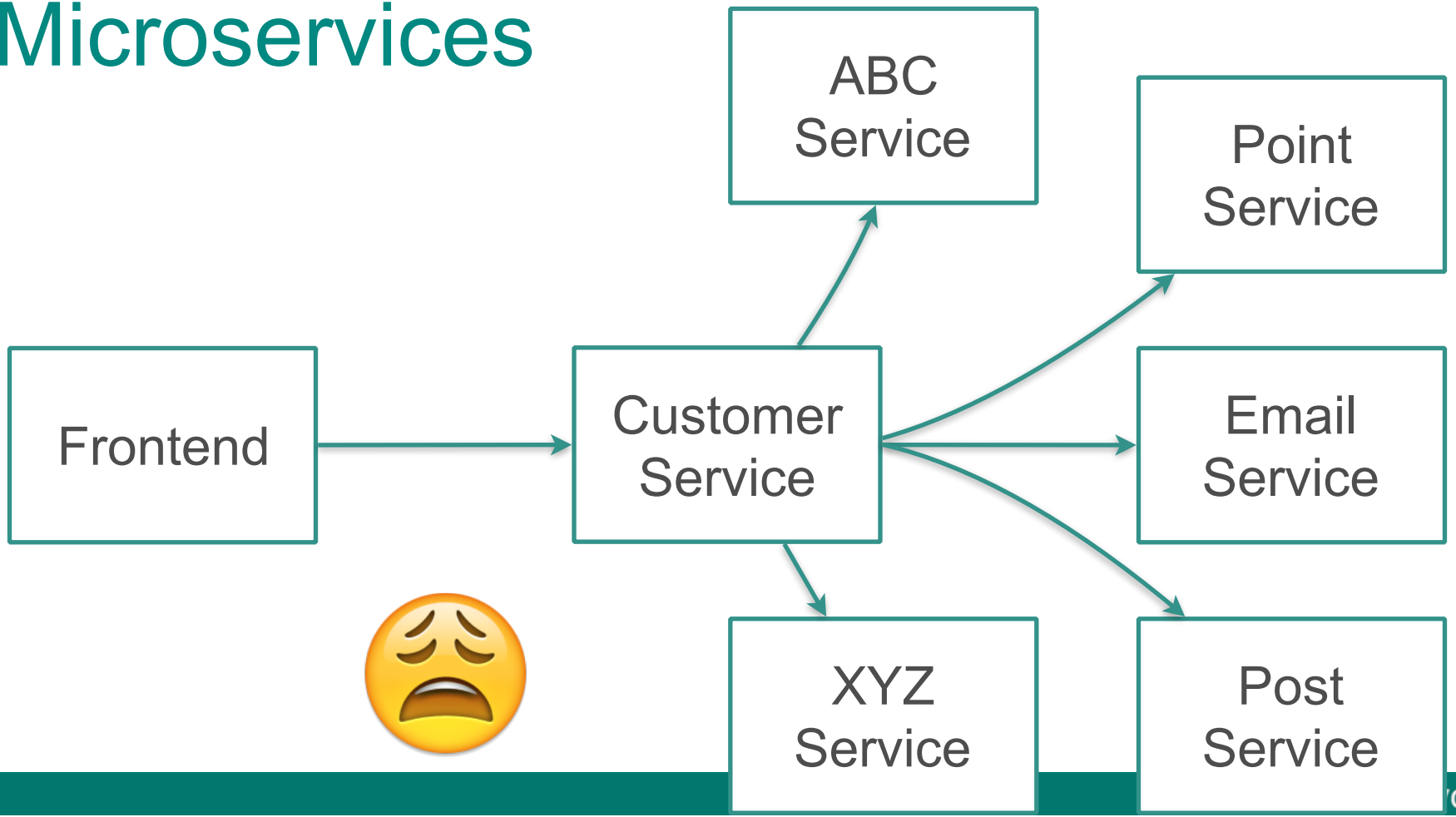
Microservices



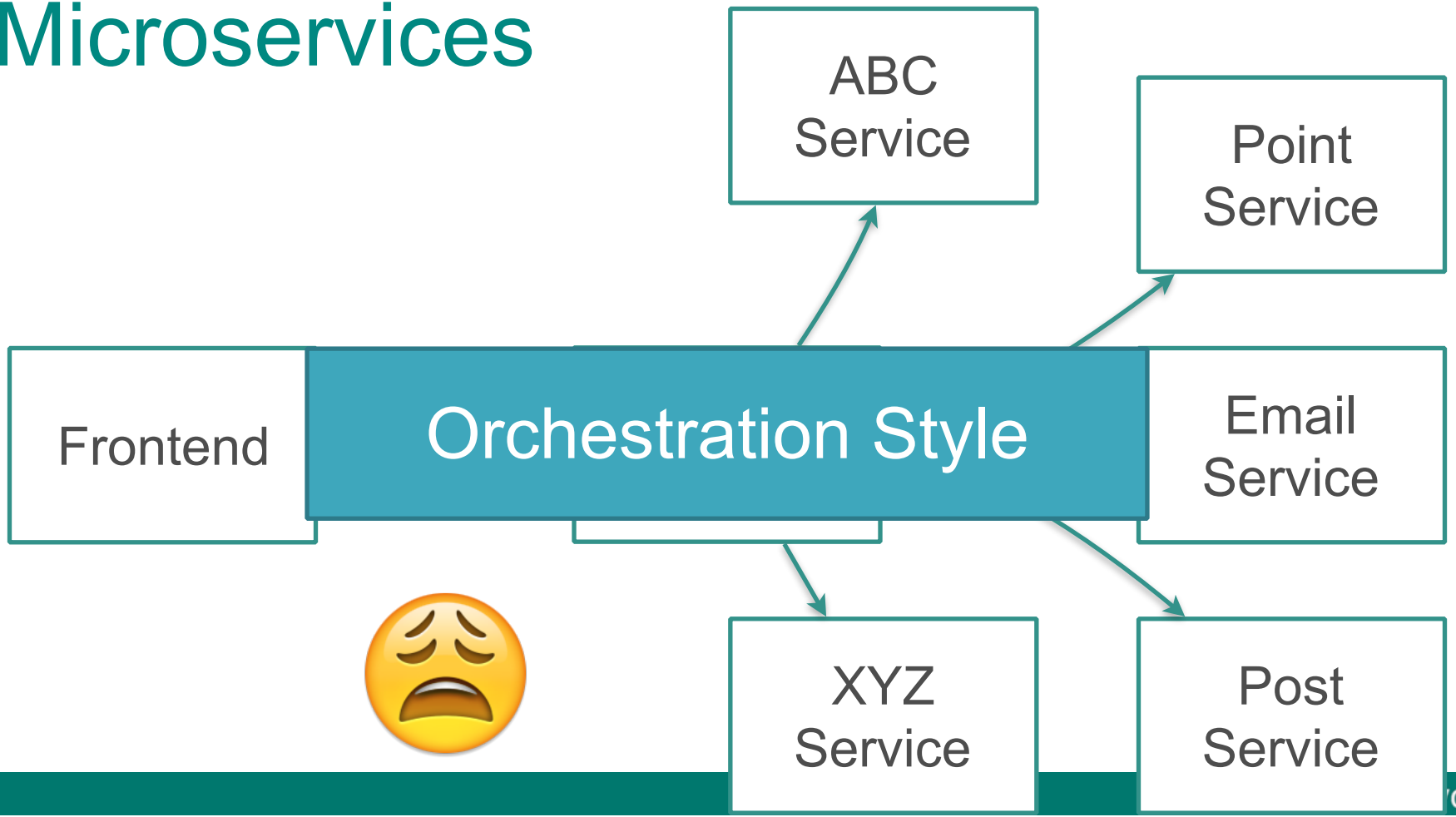
Microservices



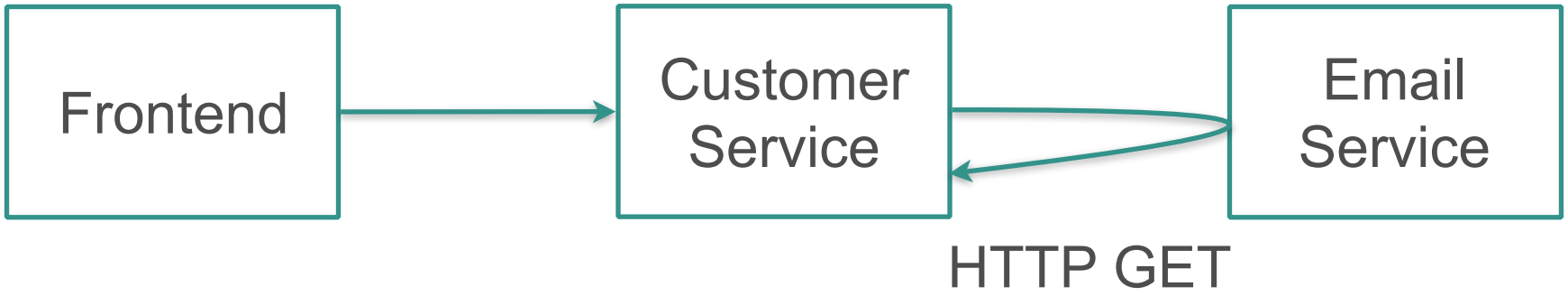
Microservices



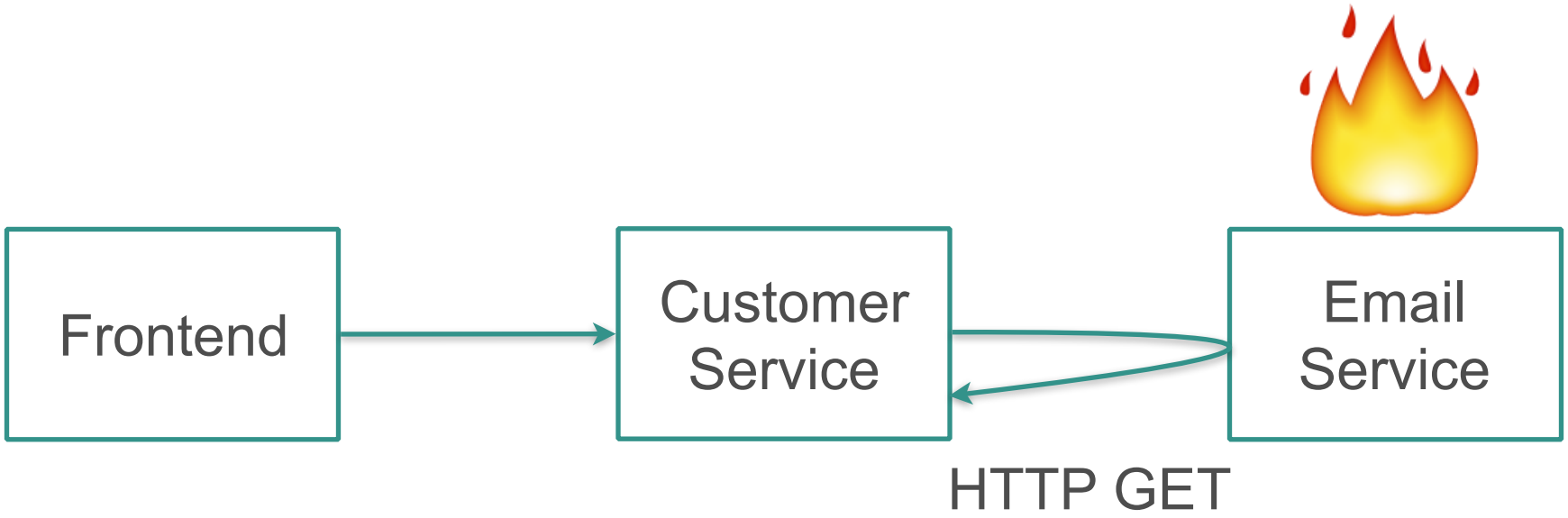
Microservices



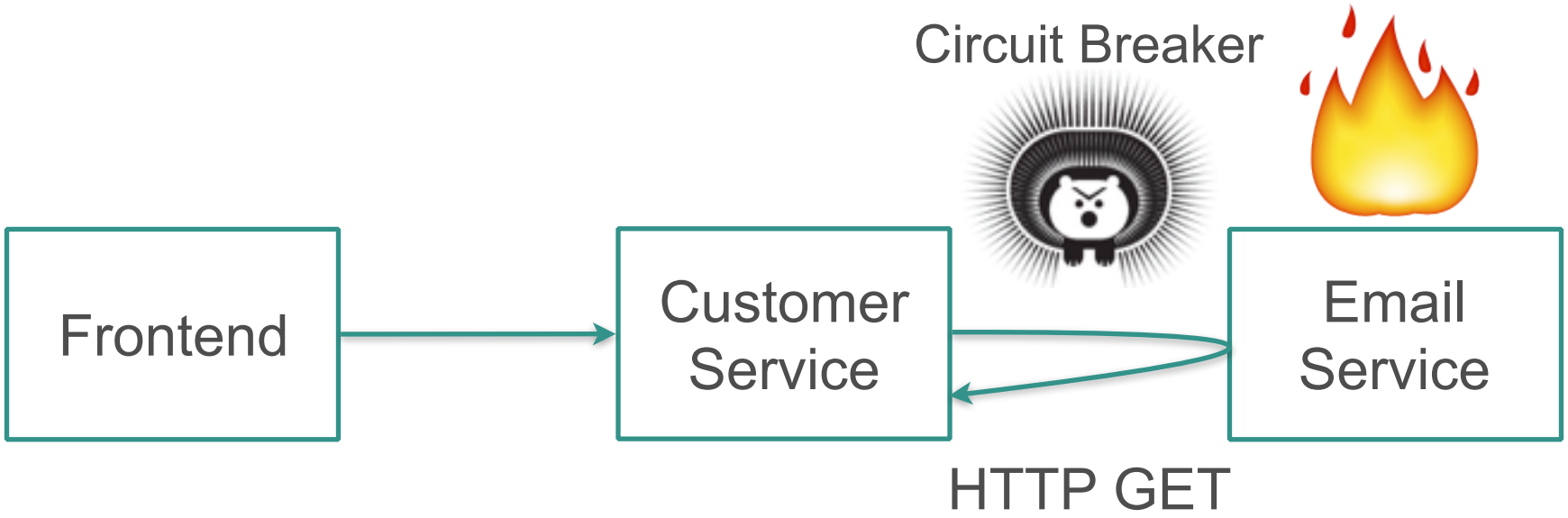
Read Failure



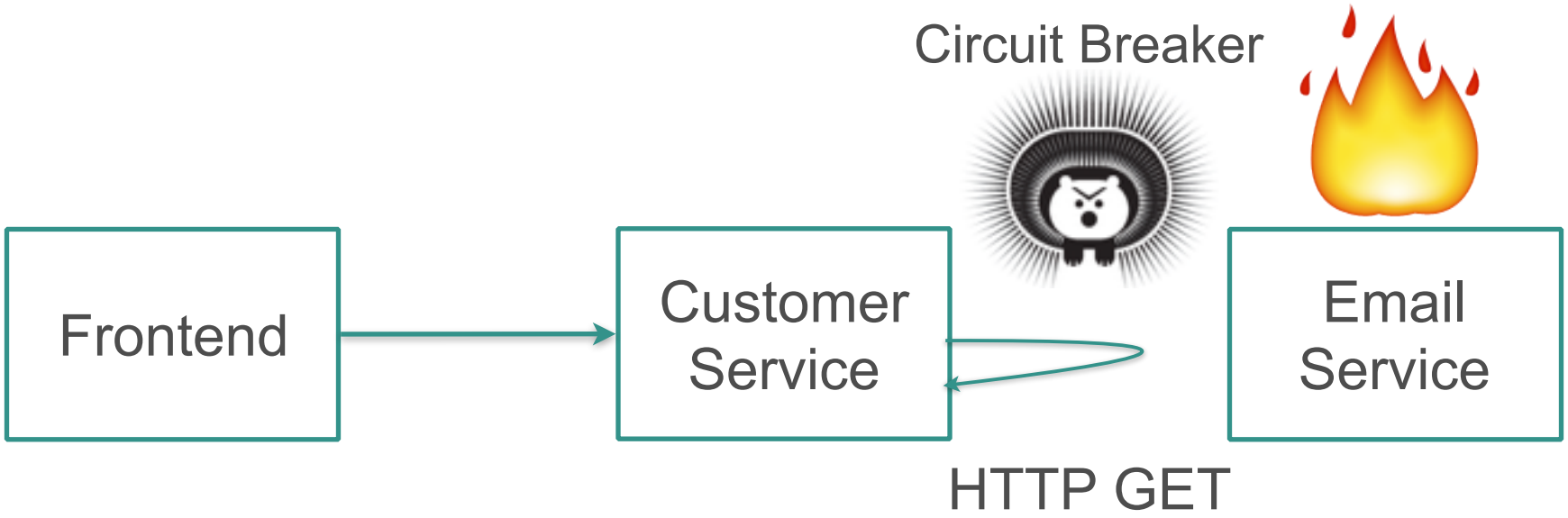
Read Failure



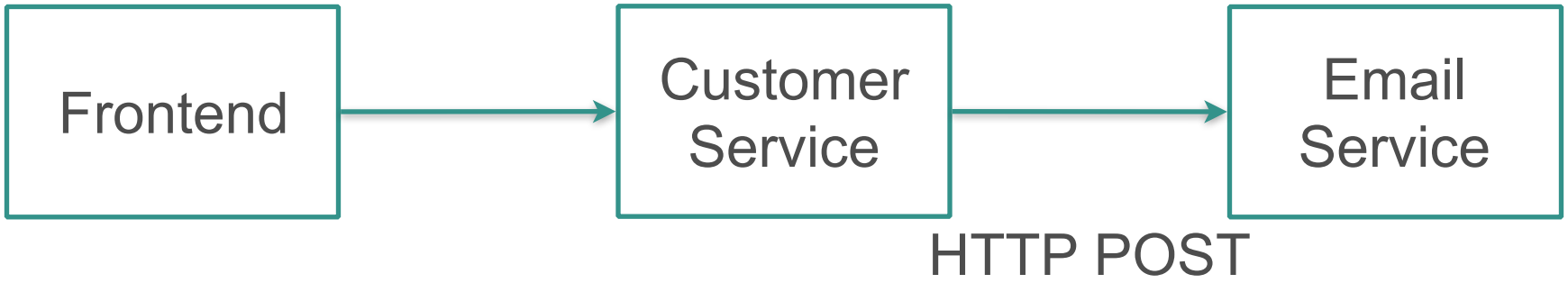
Read Failure



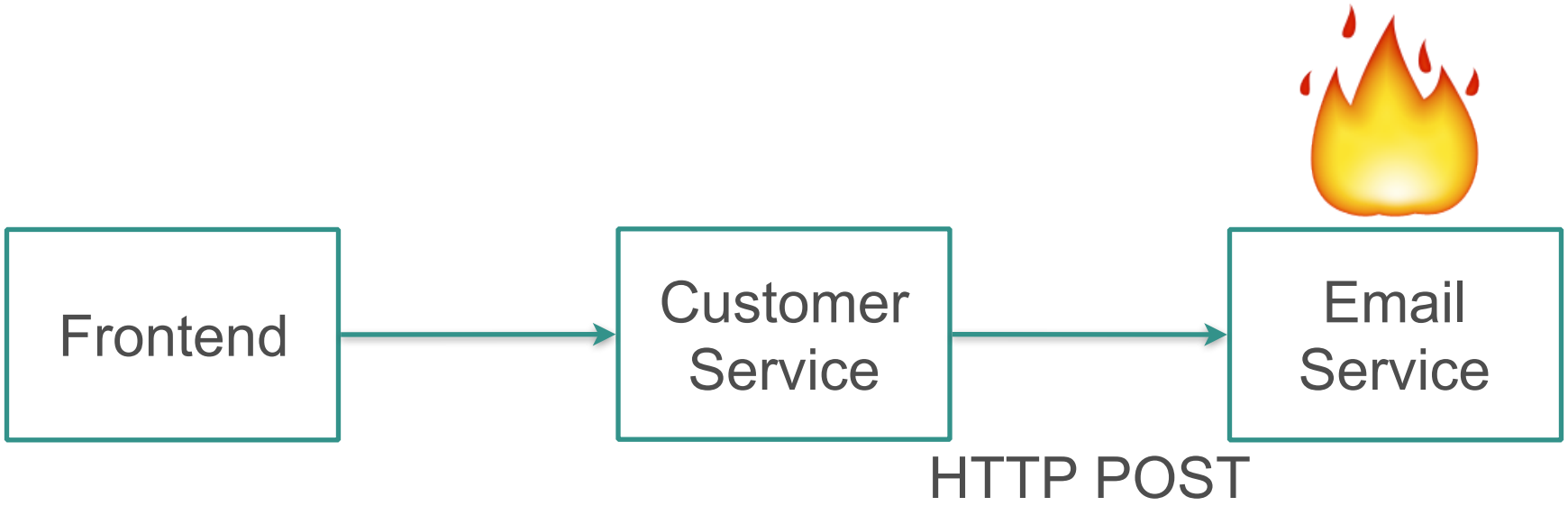
Read Failure



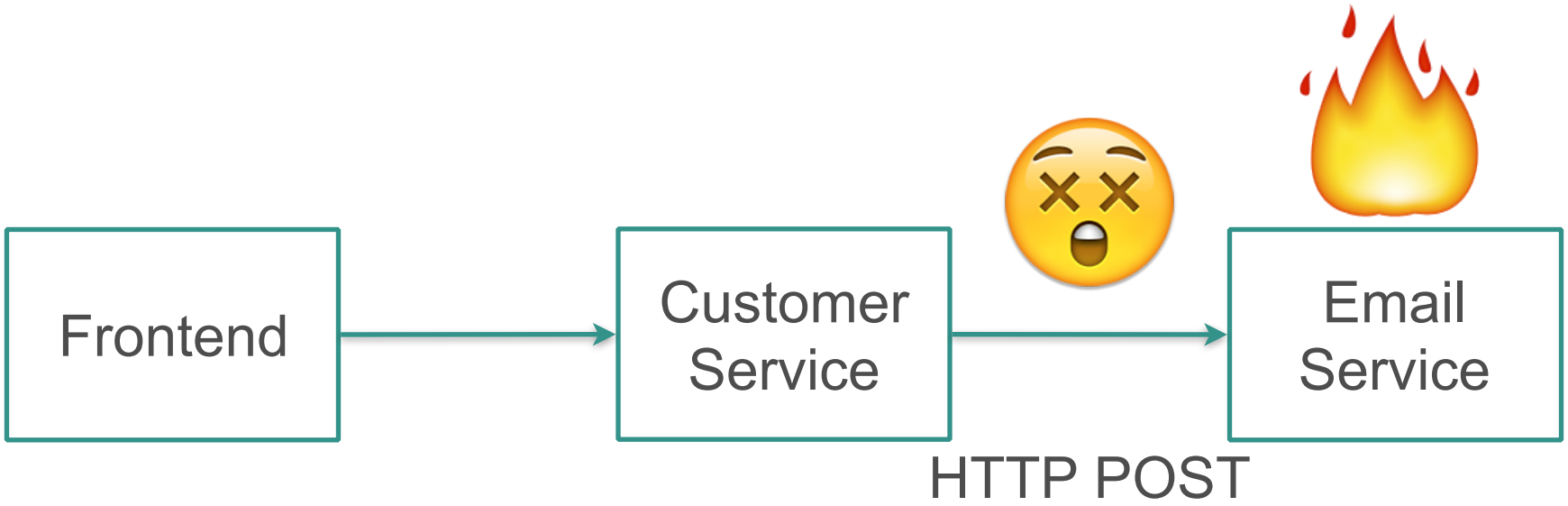
Write Failure



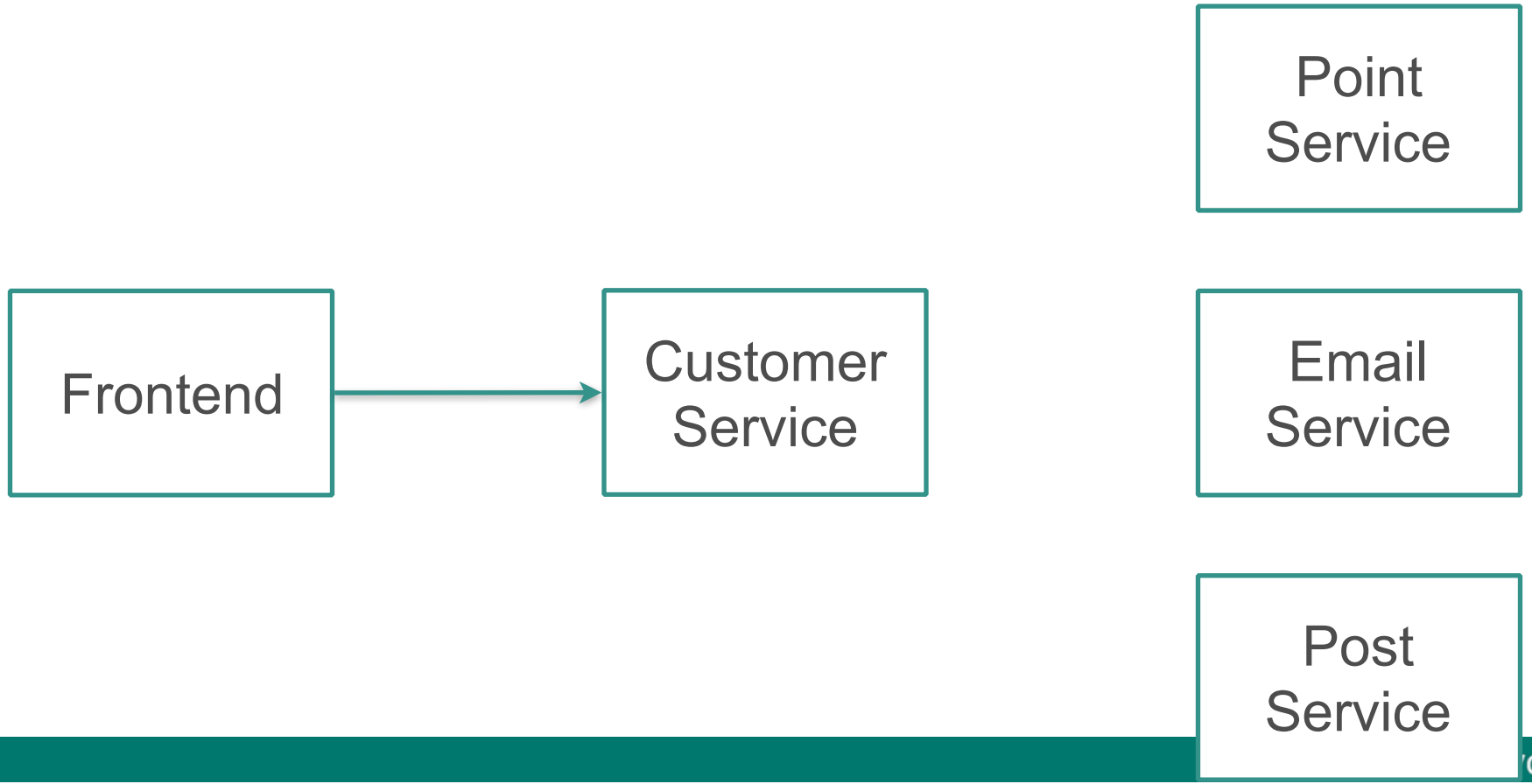
Write Failure



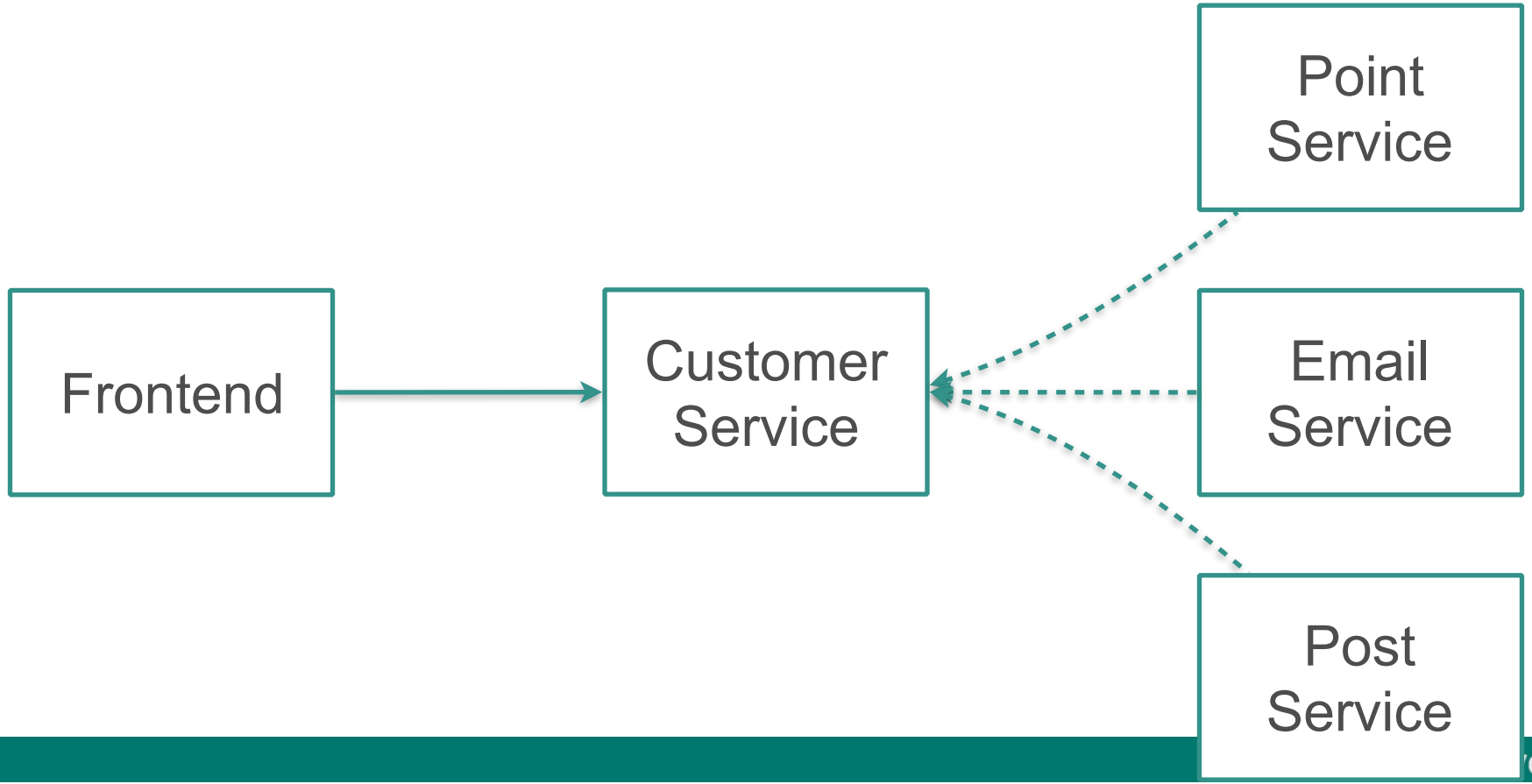
Write Failure



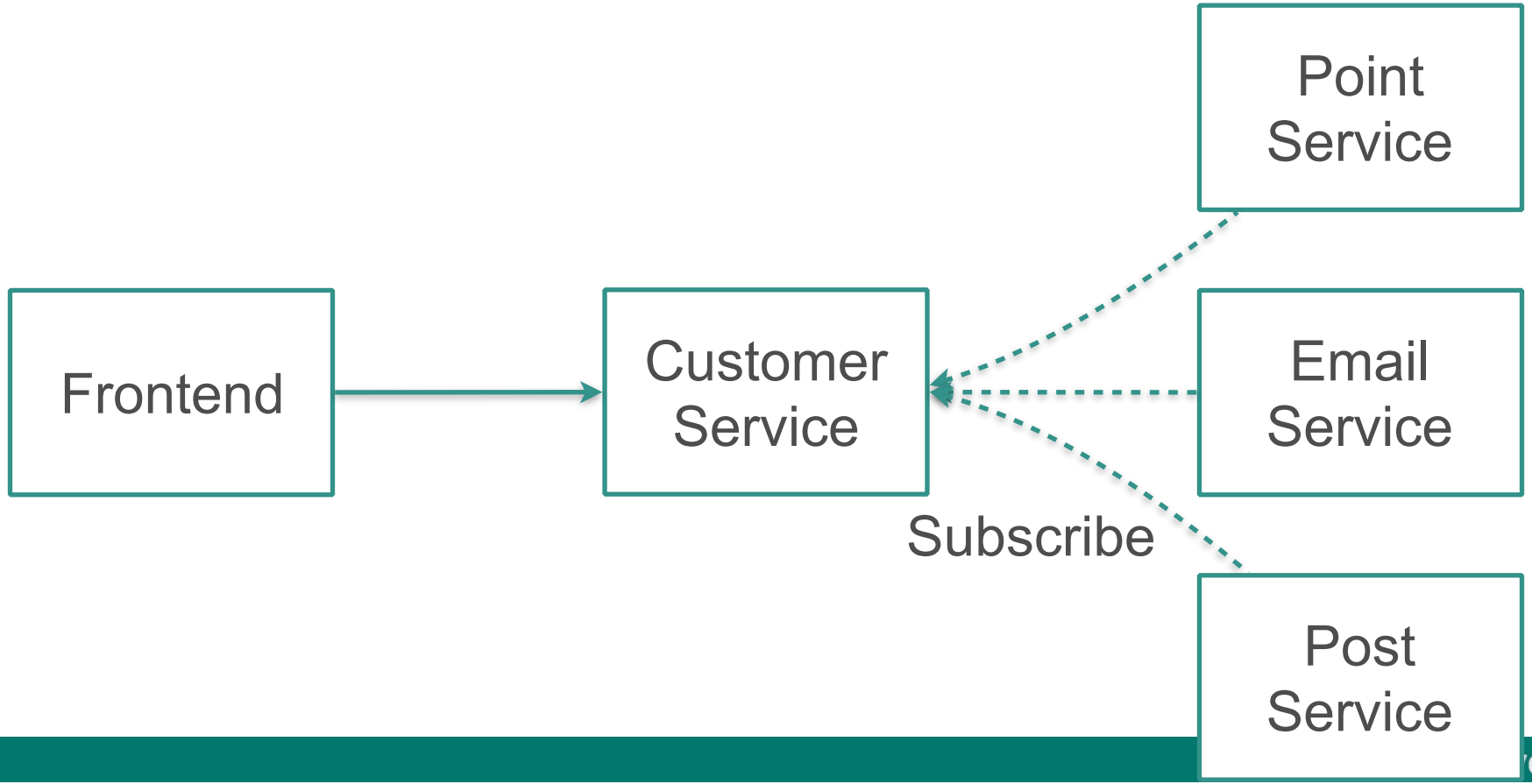
Event Driven MicroServices



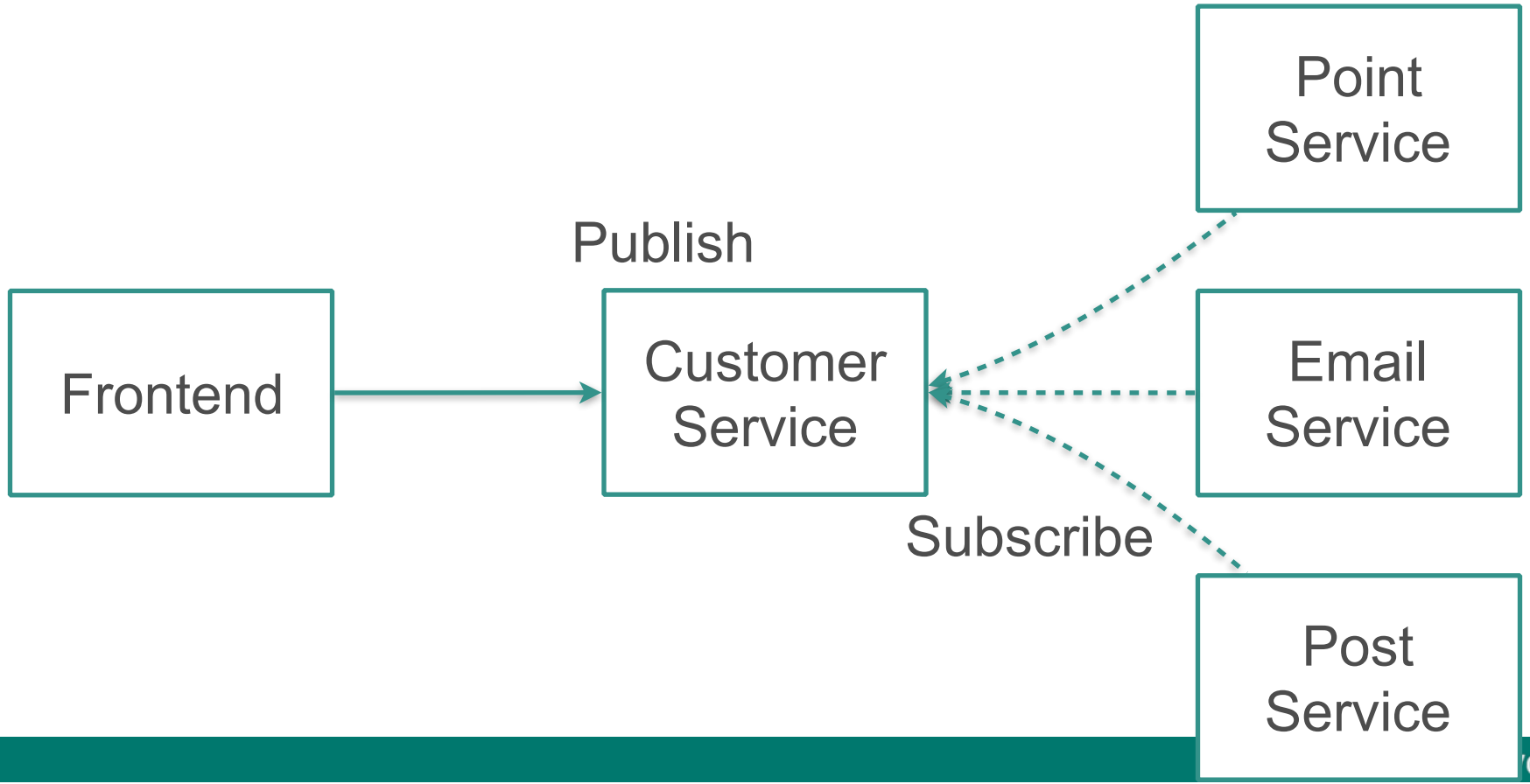
Event Driven MicroServices



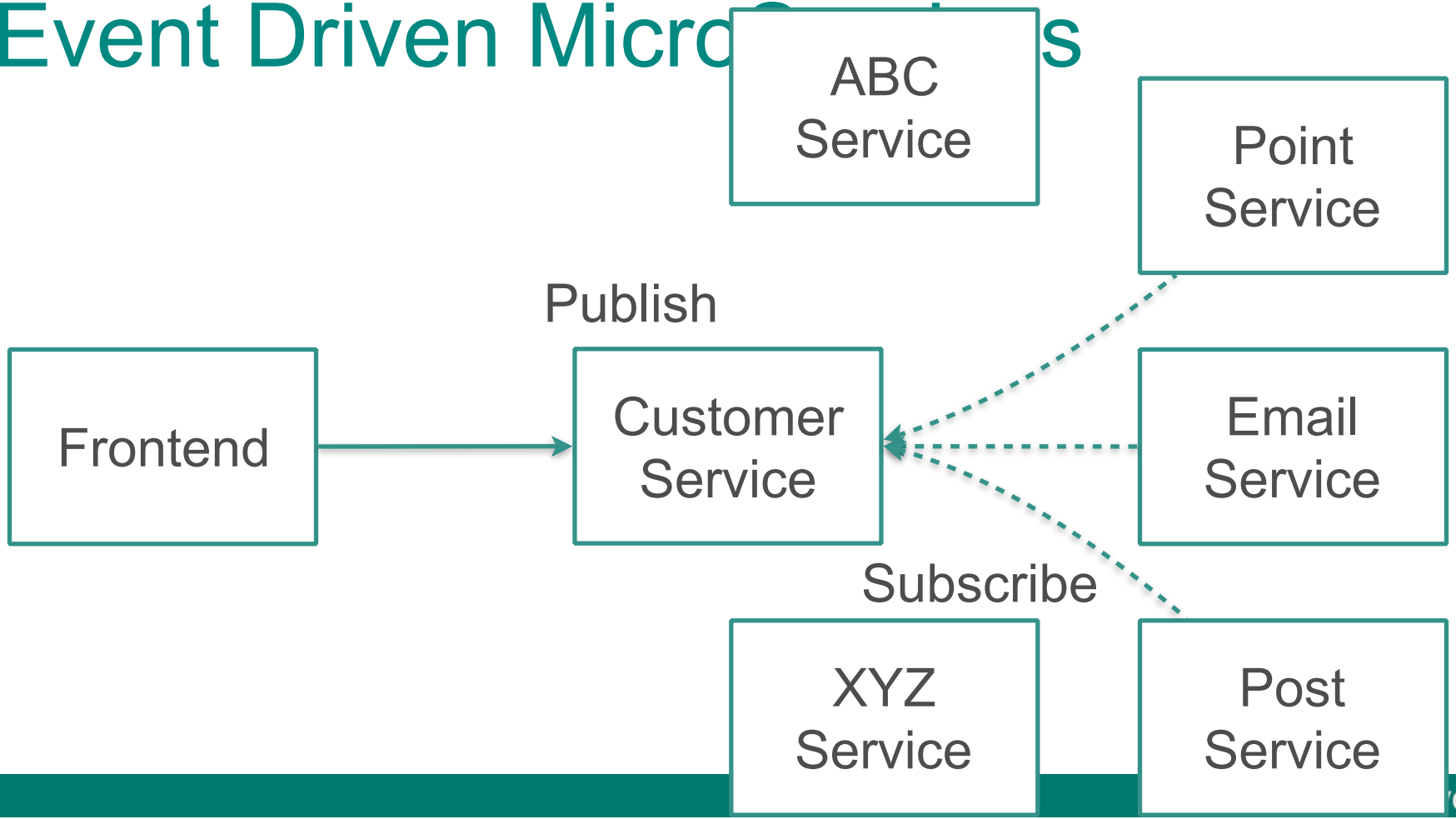
Event Driven MicroServices



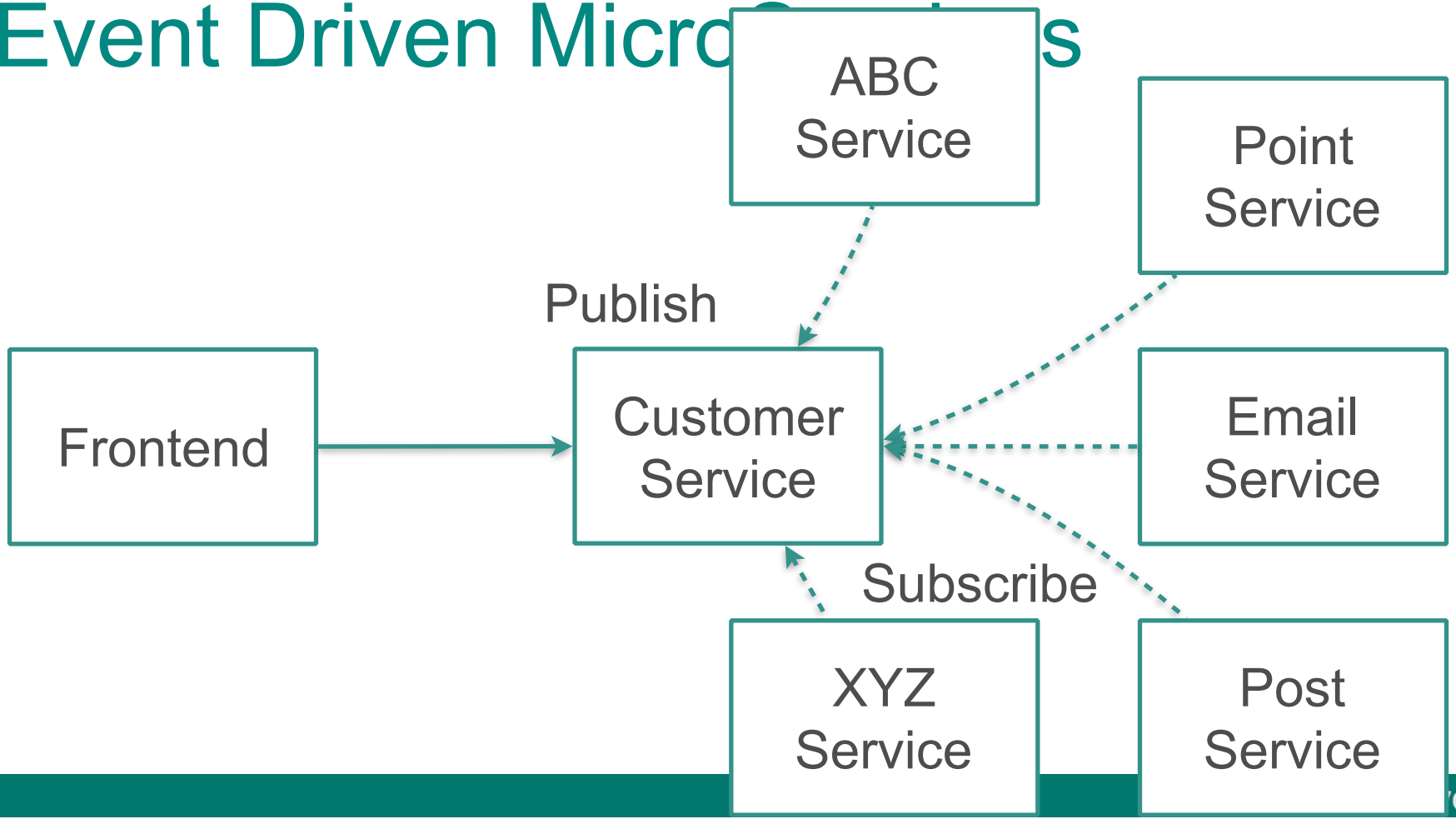
Event Driven MicroServices



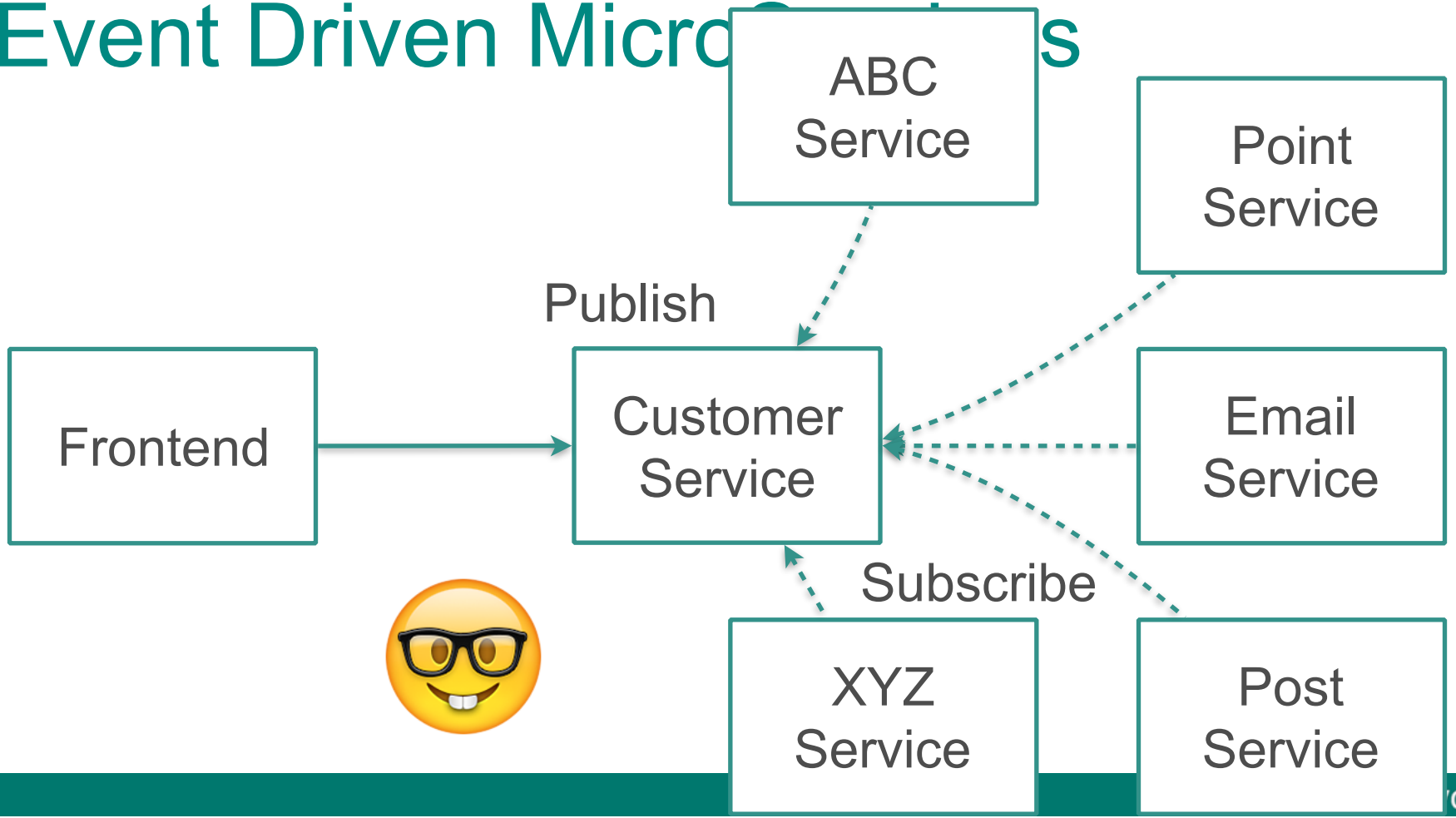
Event Driven Micro[illegible]s



Event Driven Micro Services



Event Driven Micro Services



Choreography Style

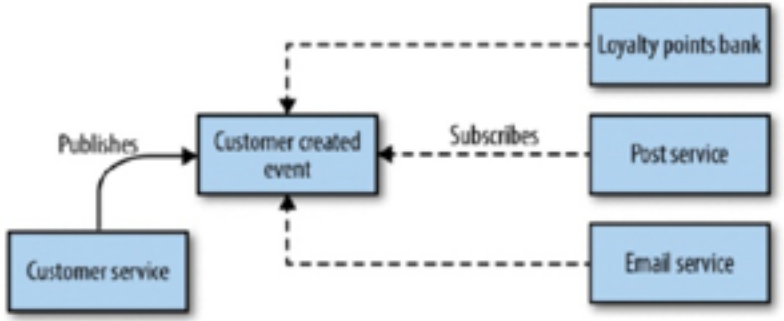


Figure 4-4. Handling customer creation via choreography

ans additional work is needed to ensure that you can monitor and track th
ngs have happened. For example, would you know if the loyalty points ha
for some reason didn't set up the correct account? One approach I like fo
s is to build a monitoring system that explicitly matches the view of the bu
in Figure 4-2, but then tracks what each of the services does as independe
letting you see odd exceptions mapped onto the more explicit process flo
rt we saw earlier isn't the driving force, but just one lens through which w
the system is behaving.

al, I have found that systems that tend more toward the choreographed ap
e loosely coupled, and are more flexible and amenable to change. You do

https://www.thoughtworks.com/insights/blog/scaling-microservices-event-stream

Spring Cloud Stream

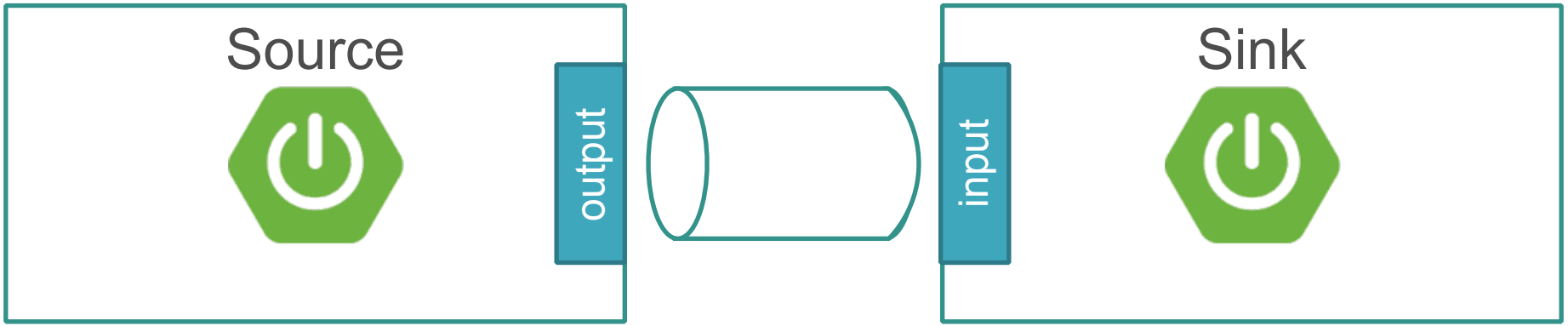
Pivotal

Spring Cloud Stream

- Event-driven microservice framework
- Built on battle-tested components (**Spring Boot** / **Spring Integration**)
- Opinionated primitives for streaming applications
  - Persistent Pub/Sub
  - Consumer Groups
  - Partitioning Support
- Pluggable messaging middleware bindings

```
source | processor | sink
```

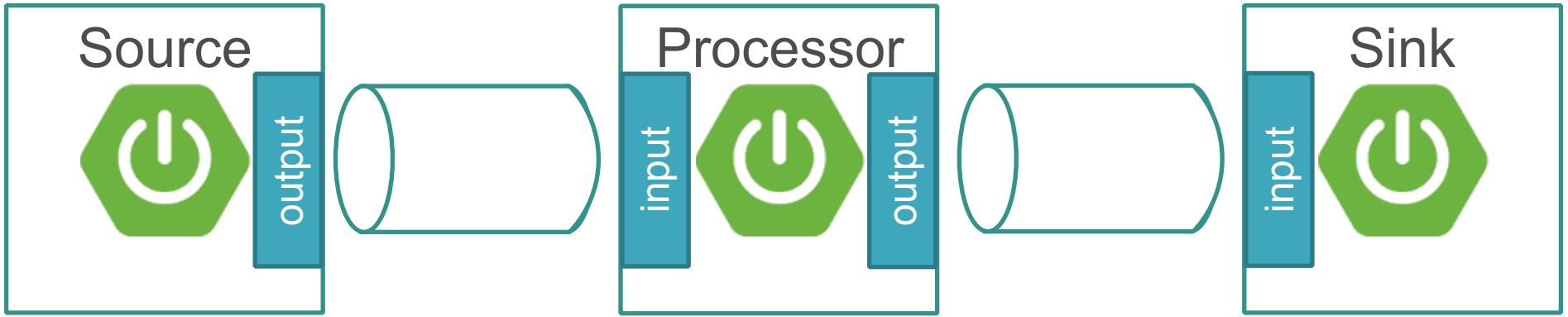
Source | Sink



RabbitMQ

APACHE kafka
A distributed streaming platform

Source | Processor | Sink



RabbitMQ

APACHE kafka
A distributed streaming platform

RabbitMQ

APACHE kafka
A distributed streaming platform

Spring Cloud Stream Applications

Source

Twitter Stream

```
java -jar twittersource.jar --server.port=8080
        --consumerKey=XYZ --consumerSecret=ABC
        --spring.cloud.stream.bindings.
          output.destination=ingest
```

Sink

Cassandra

```
java -jar cassandrasink.jar --server.port=8081
        --spring.cassandra.keyspace=tweet
        --spring.cloud.stream.bindings.
          input.destination=ingest
```

Spring Cl

Source


```
java -jar twittersource.jar --server.port=8080
        --consumerKey=XYZ --consumerSecret=ABC
        --spring.cloud.stream.bindings.
          output.destination=ingest
```

Sink


```
java -jar cassandrasink.jar --server.port=8081
        --spring.cassandra.keyspace=tweet
        --spring.cloud.stream.bindings.
          input.destination=ingest
```

Message Binders

- `@EnableBinding`
- Binder Implementations
  - Production-Ready
    - Rabbit MQ
    - Apache Kafka
  - Experimental
    - JMS
    - Google PubSub

Programming Model (Sink)

```
@SpringBootApplication
@EnableBinding(Sink.class)
public class DemoSinkApp {
  @StreamListener(Sink.INPUT)
  void receive(Message<String> message) {
    System.out.println("Received " + message.getPayload());
  }
  public static void main(String[] args) {
    SpringApplication.run(DemoSinkApp.class, args);
  }
}
```

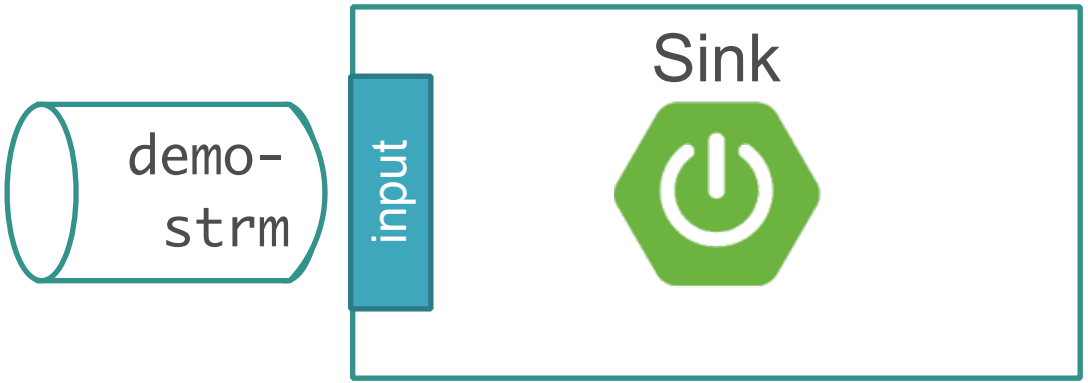
Programming Mo

```
@SpringBootApplication
@EnableBinding(Sink.class)
public class DemoSinkApp {
  @StreamListener(Sink.INPUT)
  void receive(Message<String> message) {
    System.out.println("Received " + message.getPayload());
  }
  public static void main(String[] args) {
    SpringApplication.run(DemoSinkApp.class, args);
  }
}
```

```
public interface Sink {
  String INPUT = "input";
  @Input(Sink.INPUT)
  SubscribableChannel input();
}
```

Properties (Sink)

```
spring.cloud.stream.bindings.input.destination=demo-strm
```



Programming Model (Source)

```
@SpringBootApplication @RestController
@EnableBinding(Source.class)
public class DemoSourceApp {
  @Autowired @Output(Source.OUTPUT)
  MessageChannel output;
  @GetMapping void send(@RequestParam String text) {
    output.send(MessageBuilder.withPayload(text).build());
  }
  public static void main(String[] args) {
    SpringApplication.run(DemoSourceApp.class, args);
  }
}
```

Programming Mode

```
@SpringBootApplication @RestCon
@EnableBinding(Source.class)
public class DemoSourceApp {
  @Autowired @Output(Source.OUTPUT)
  MessageChannel output;
  @GetMapping void send(@RequestParam String text) {
    output.send(MessageBuilder.withPayload(text).build());
  }
  public static void main(String[] args) {
    SpringApplication.run(DemoSourceApp.class, args);
  }
}
```

```
public interface Source {
  String OUTPUT = "output";
  @Output(Source.OUTPUT)
  MessageChannel output();
}
```

Programming Model (Source)

```
@SpringBootApplication @RestController
@EnableBinding(Source.class)
public class DemoSourceApp {
  @Autowired Source source;
  @GetMapping void send(@RequestParam String text) {
    source.output()
      .send(MessageBuilder.withPayload(text).build());
  }
  public static void main(String[] args) {
    SpringApplication.run(DemoSourceApp.class, args);
  }
}
```

Properties (Source)

```
spring.cloud.stream.bindings.output.destination=demo-strm
```

Properties (Source)

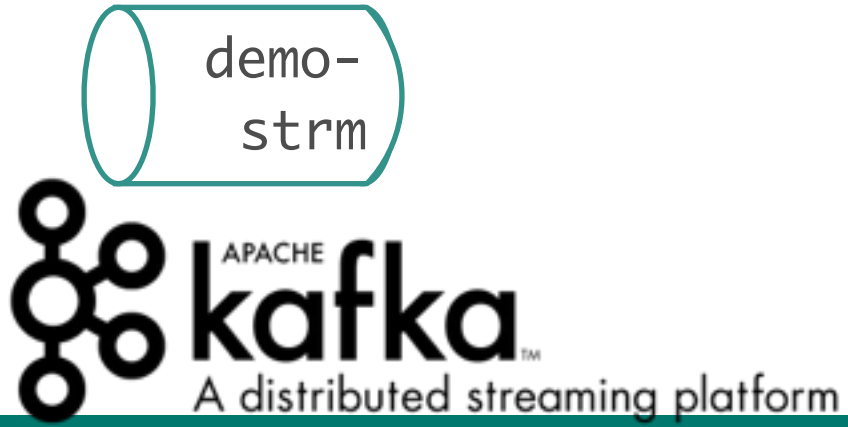
```
spring.cloud.stream.bindings.output.destination=demo-strm
spring.cloud.stream.bindings.output.contentType=applicati
on/json
```

Binder (RabbitMQ)

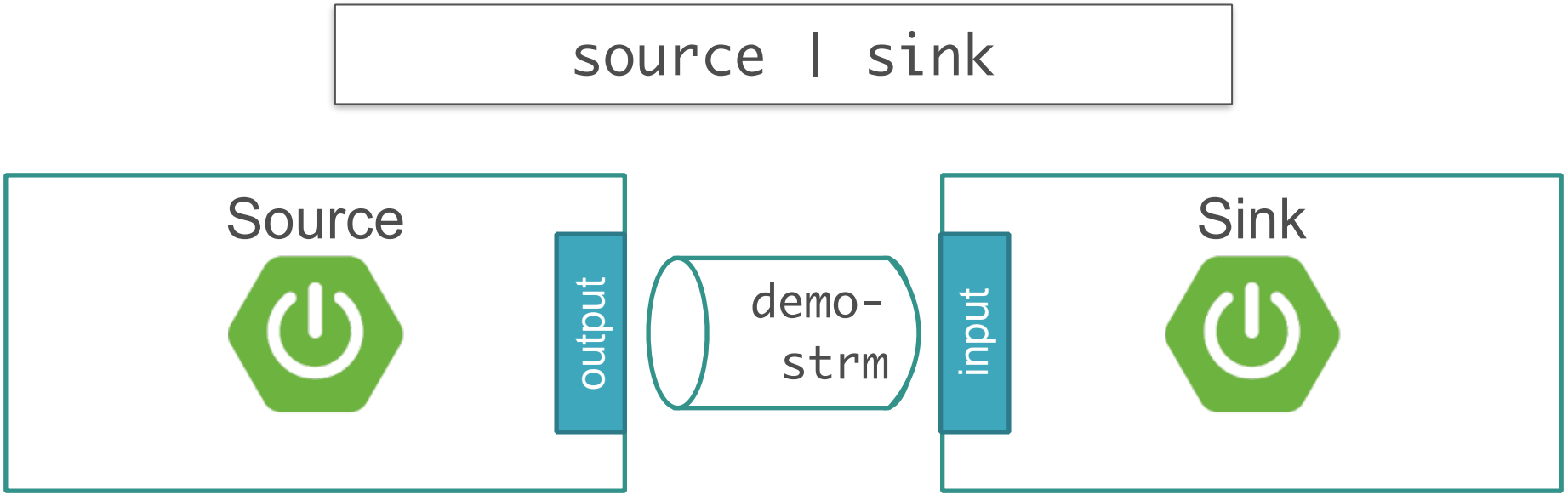
```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-stream-rabbit</artifactId>
</dependency>
```

Binder (Apache Kafka)

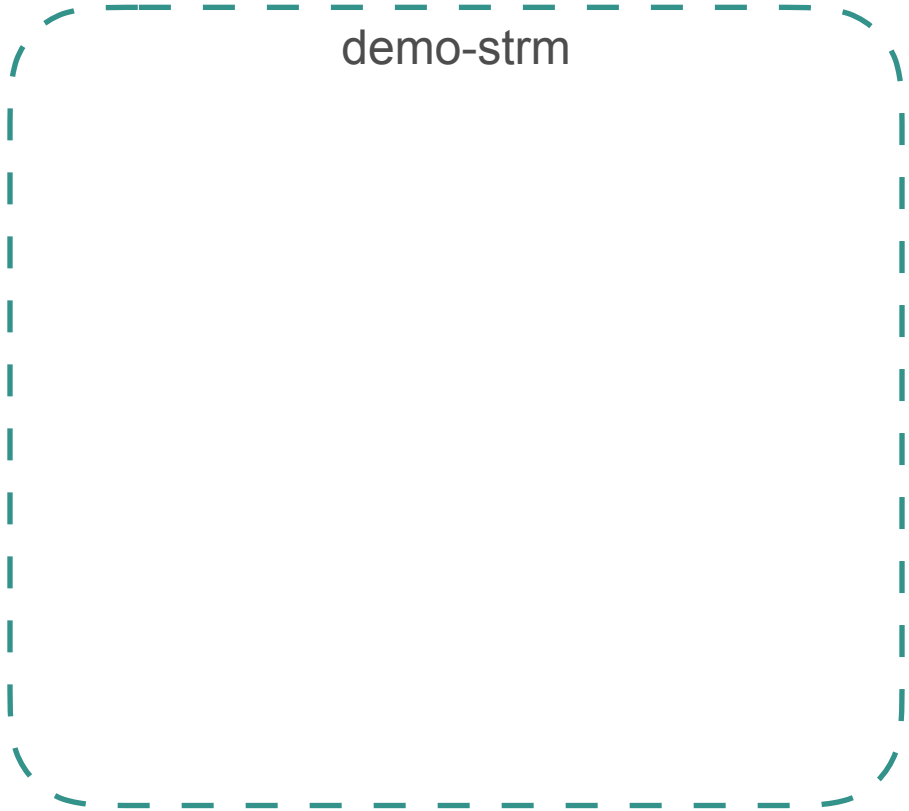
```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-stream-kafka</artifactId>
</dependency>
```



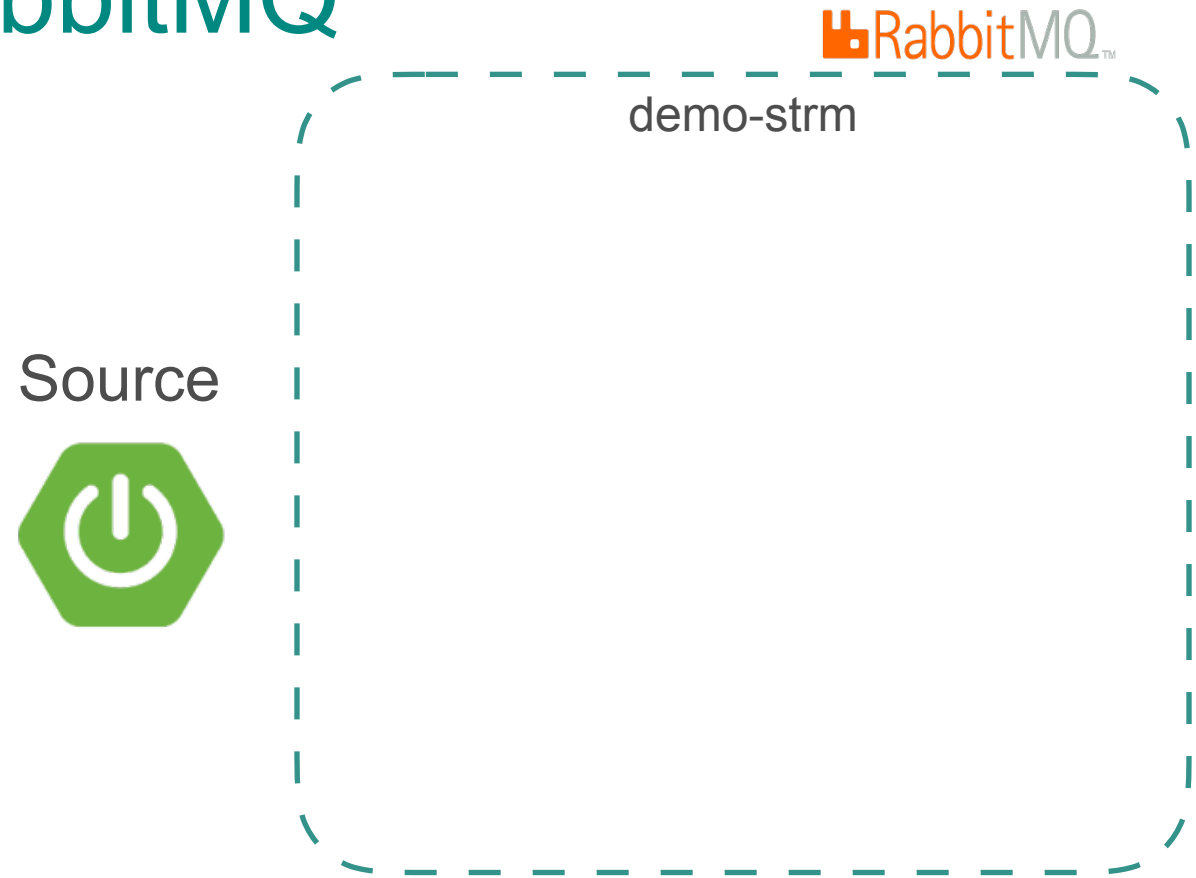
Pipeline



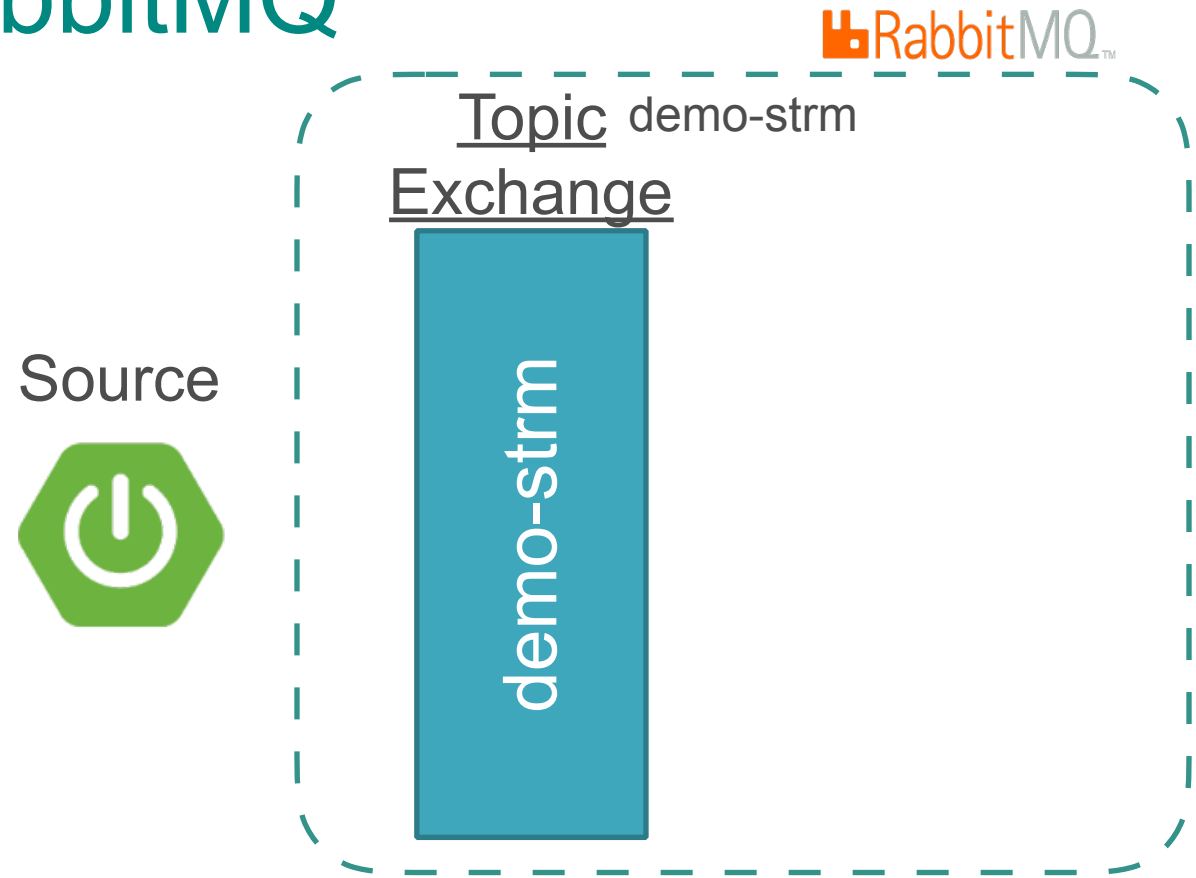
RabbitMQ



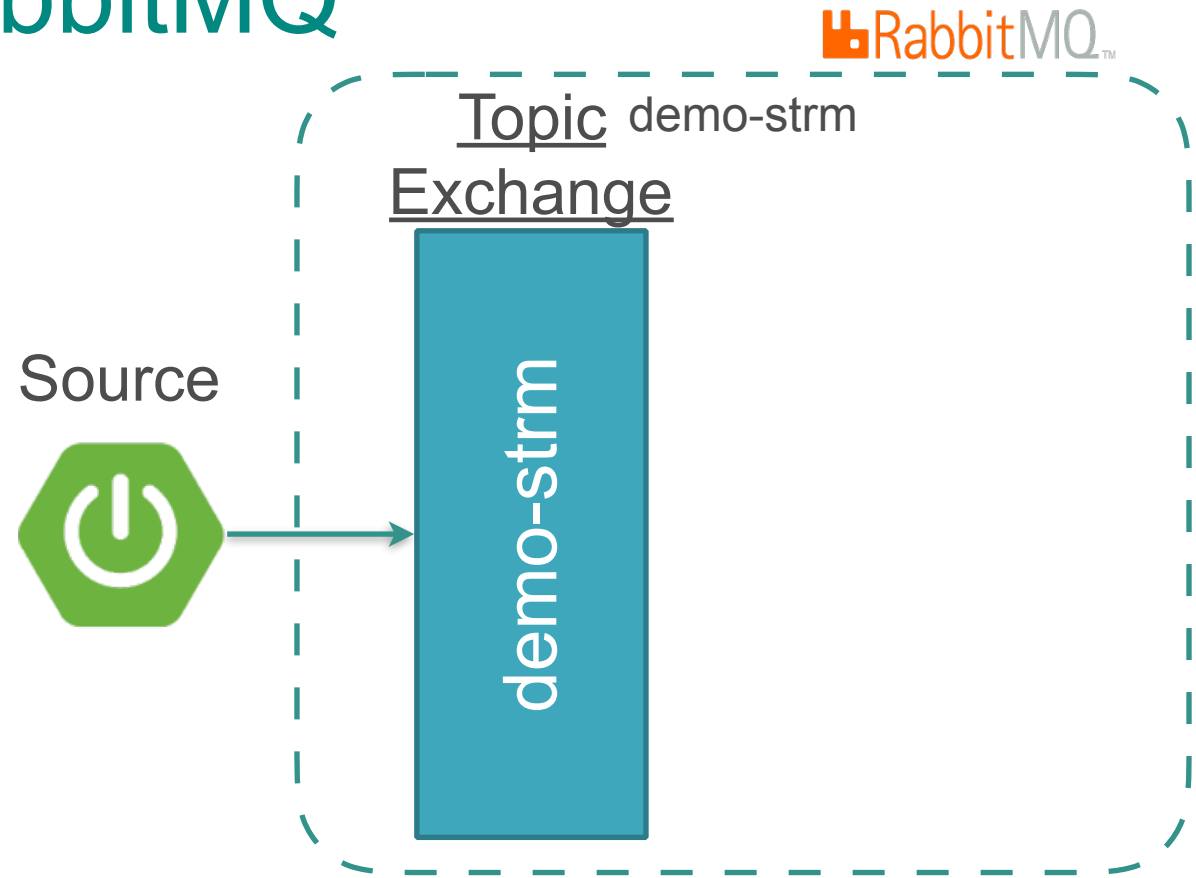
RabbitMQ



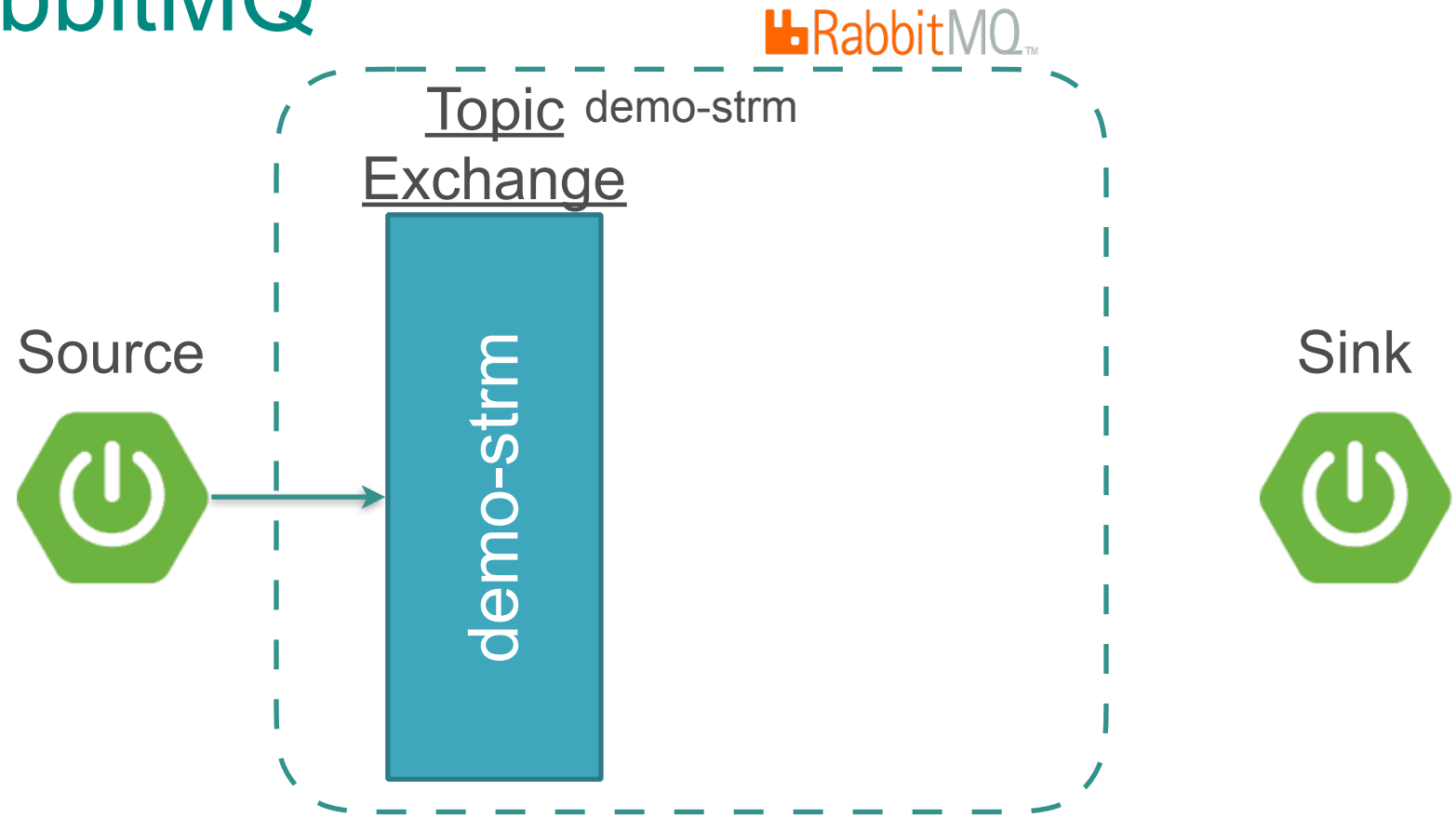
RabbitMQ



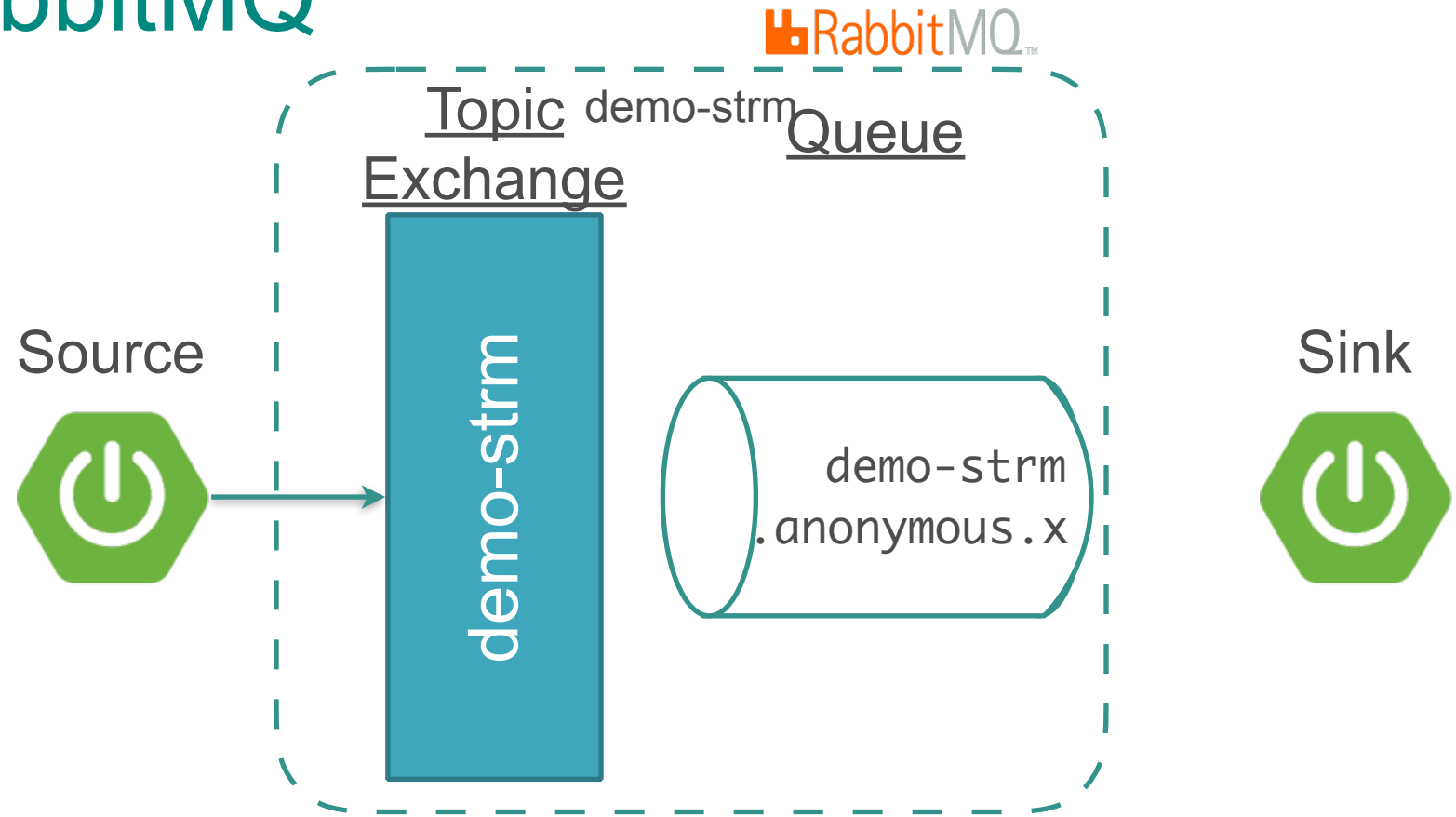
RabbitMQ



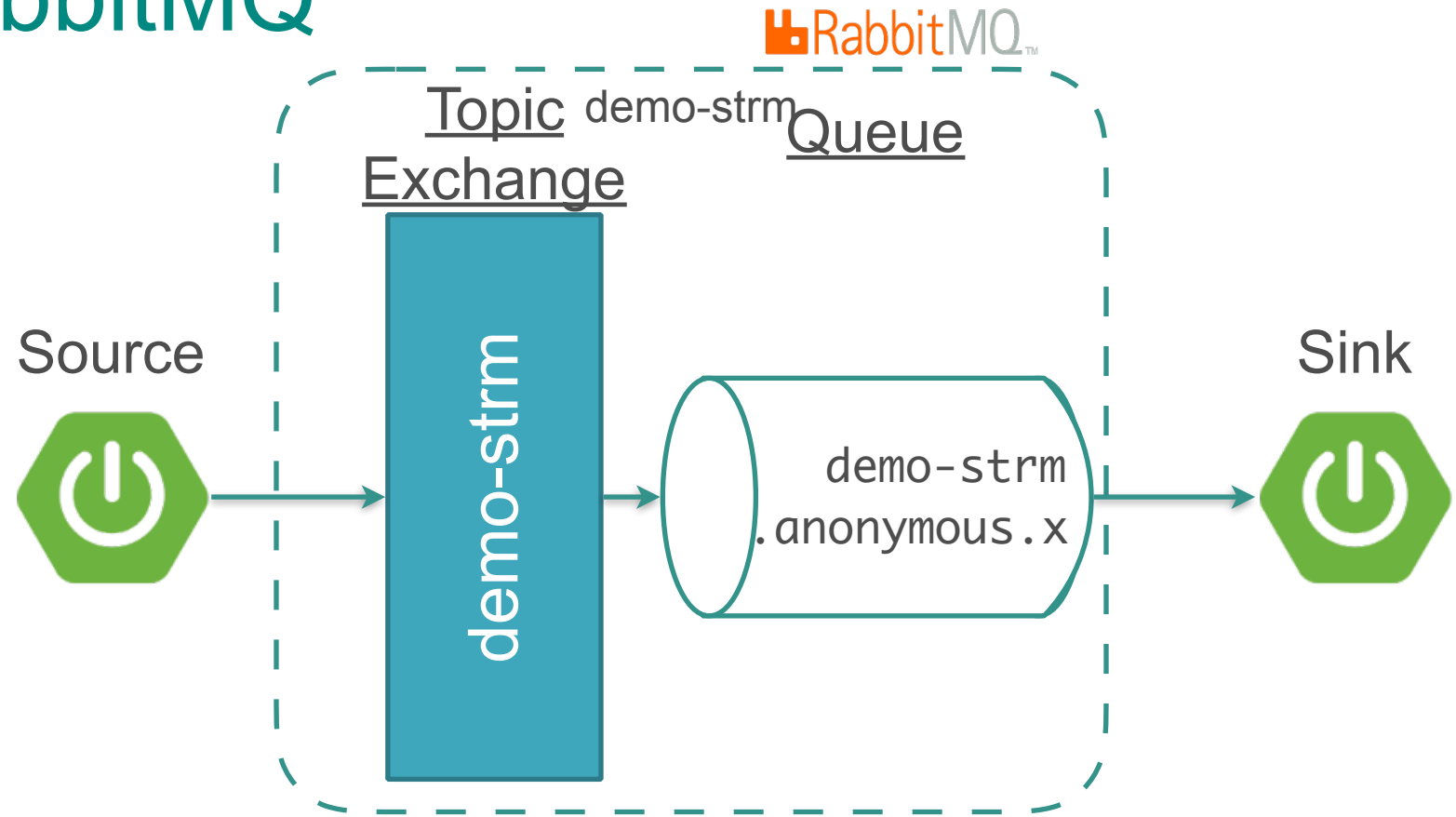
RabbitMQ



RabbitMQ



RabbitMQ



Programming Model (Processor)

```
@SpringBootApplication
@EnableBinding(Processor.class)
public class DemoProcessorApp {
  @StreamListener(Processor.INPUT)
  @SendTo(Processor.OUTPUT)
  void receive(String text) {
    return "[[" + text + "]]";
  }
  public static void main(String[] args) {
    SpringApplication.run(DemoProcessorApp.class, args);
  }
}
```

Properties (Processor)

Source

```
spring.cloud.stream.bindings.output.destination=my-source
```

Processor

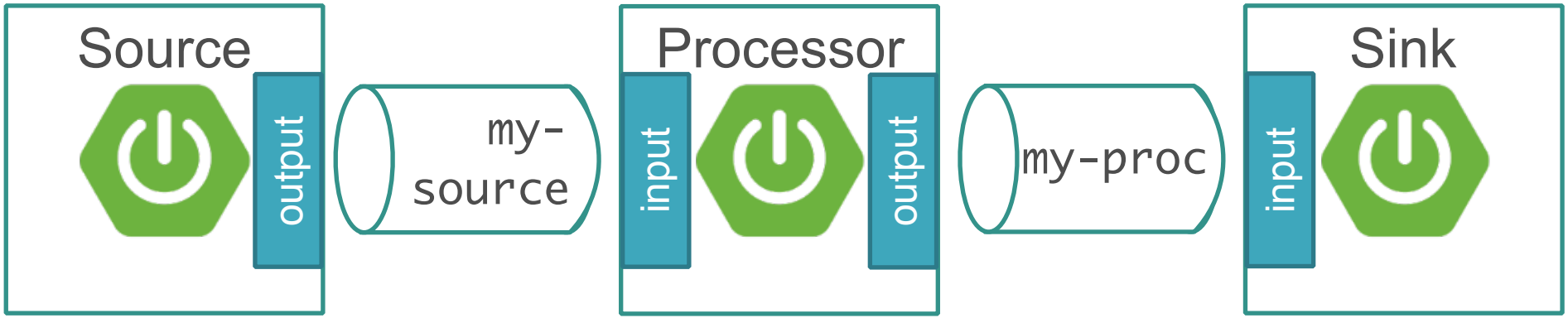
```
spring.cloud.stream.bindings.input.destination=my-source
spring.cloud.stream.bindings.output.destination=my-proc
```

Sink

```
spring.cloud.stream.bindings.input.destination=my-proc
```

Pipeline

```
source | processor | sink
```



Reactive API Support by Reactor

```
@SpringBootApplication
@EnableBinding(Processor.class)
public class DemoProcessorRxApp {
  @StreamListener @Output(Processor.OUTPUT)
  public Flux<String> receive(@Input(Processor.INPUT)
                              Flux<String> stream) {
    return stream.map(text -> "[[" + text + "]]");
  }
  public static void main(String[] args) {
    SpringApplication.run(DemoProcessorRxApp.class, args);
  }
}
```

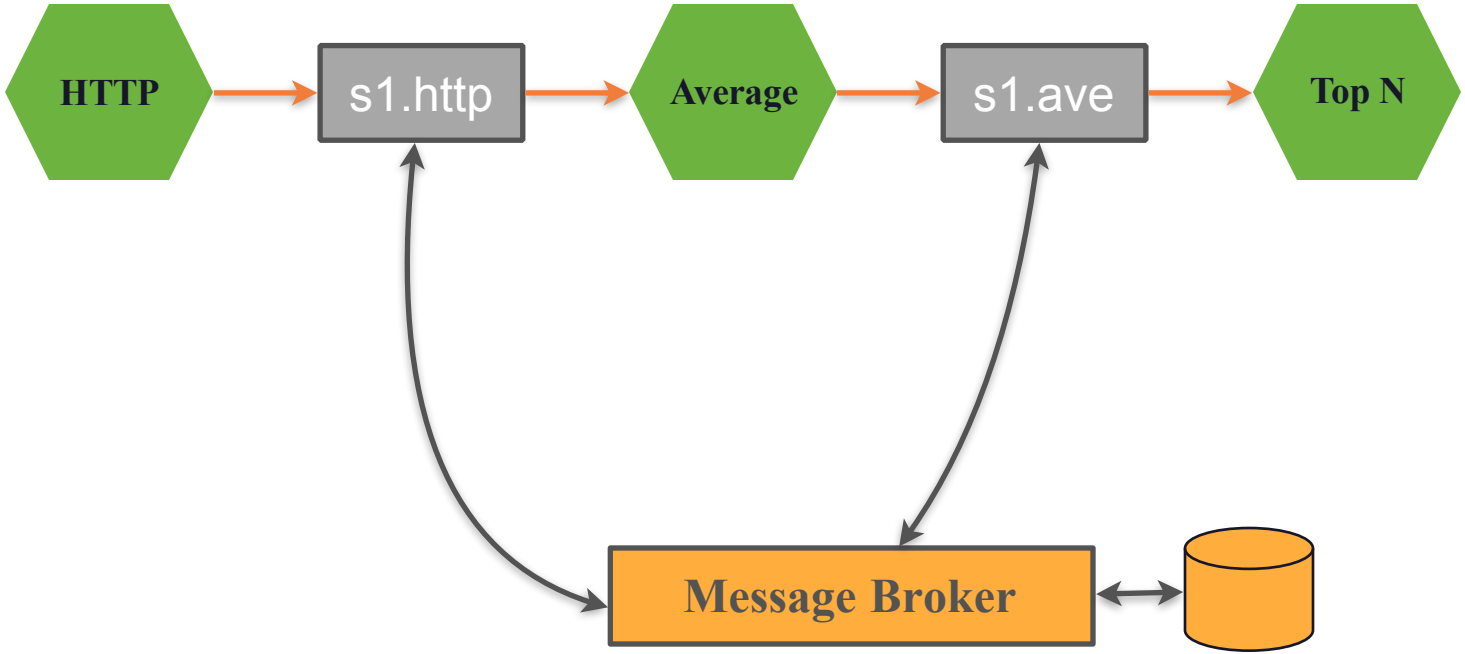
Reactive API Support by Reactor

```
@StreamListener @Output(Processor.OUTPUT)
public Flux<AverageData> receive(@Input(Processor.INPUT)
                                 Flux<SensorData> stream) {
  return stream.window(Duration.ofSecond(20),
                       Duration.ofSecond(10))
         .flatMap(win -> win.groupBy(sensor -> sensor.id))
         .flatMap(group -> calcAverage(group));
}
```

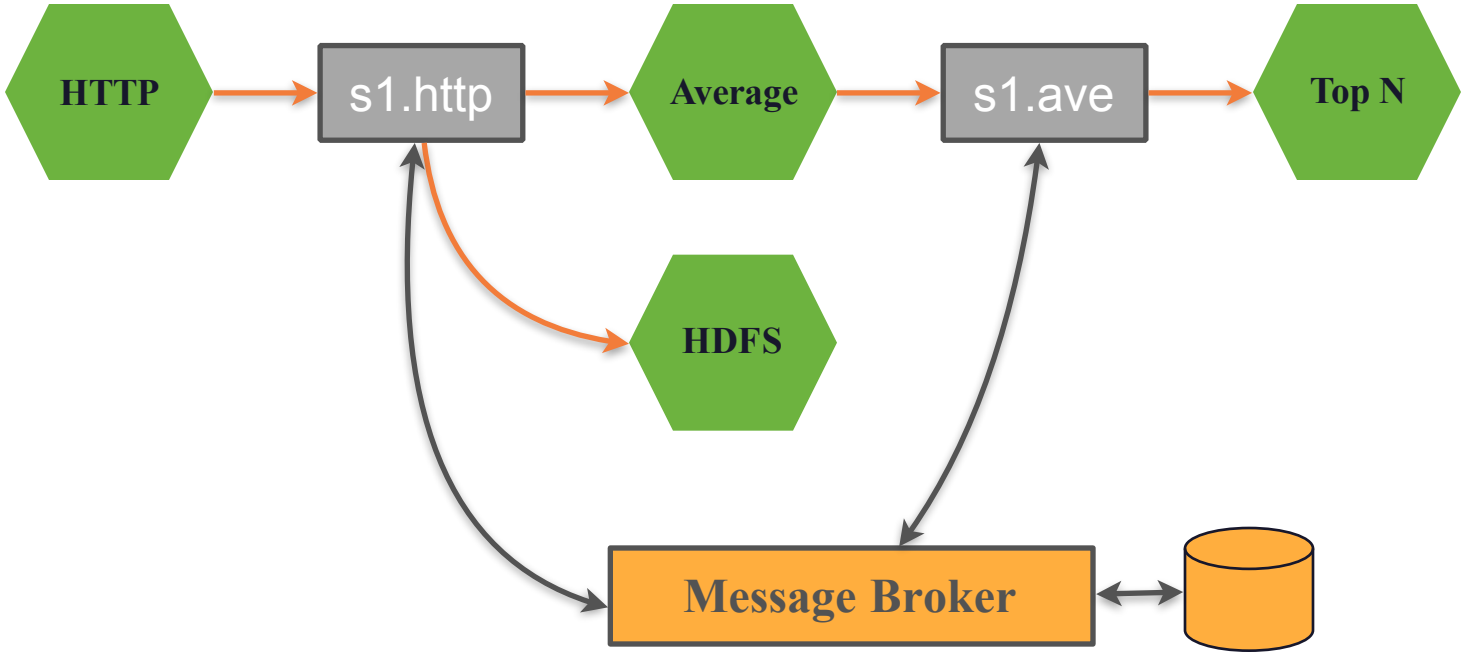
Stream Core Features

- Persistent Pub-Sub
- Consumer Group
- Partitioning Support

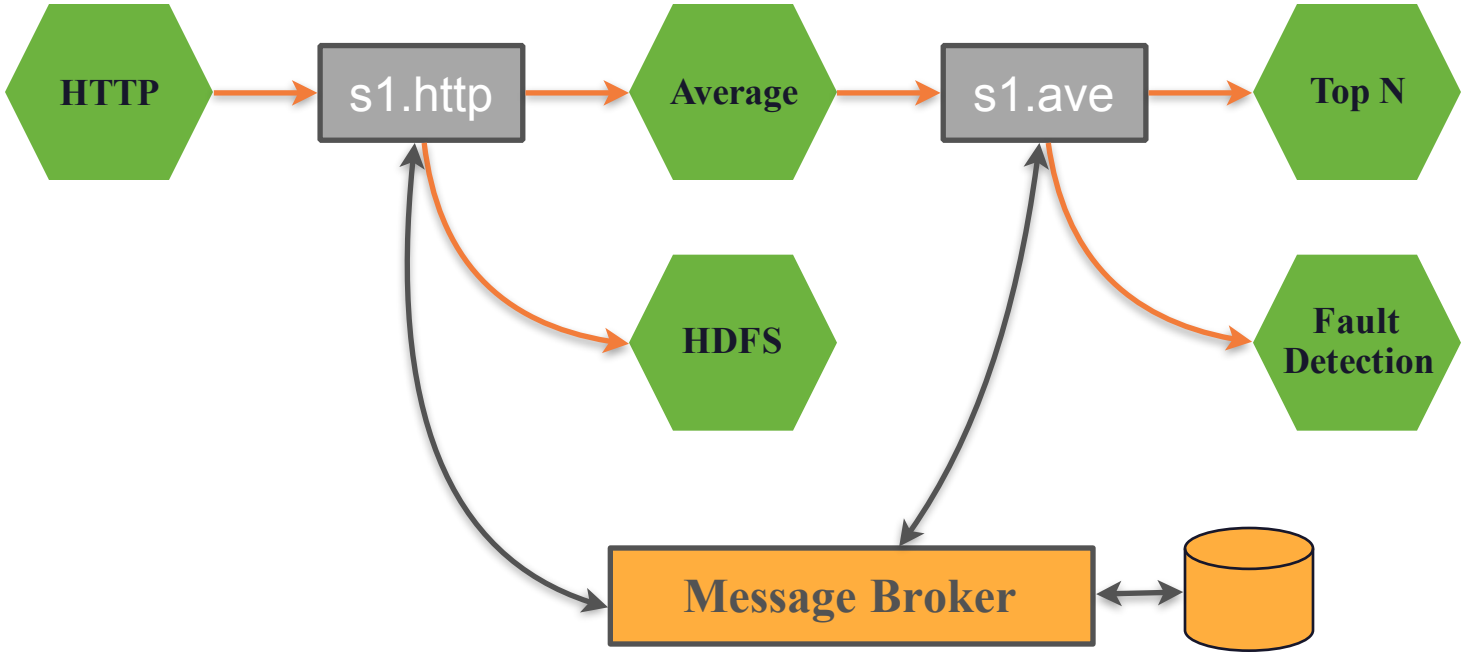
Persistent Pub-Sub



Persistent Pub-Sub

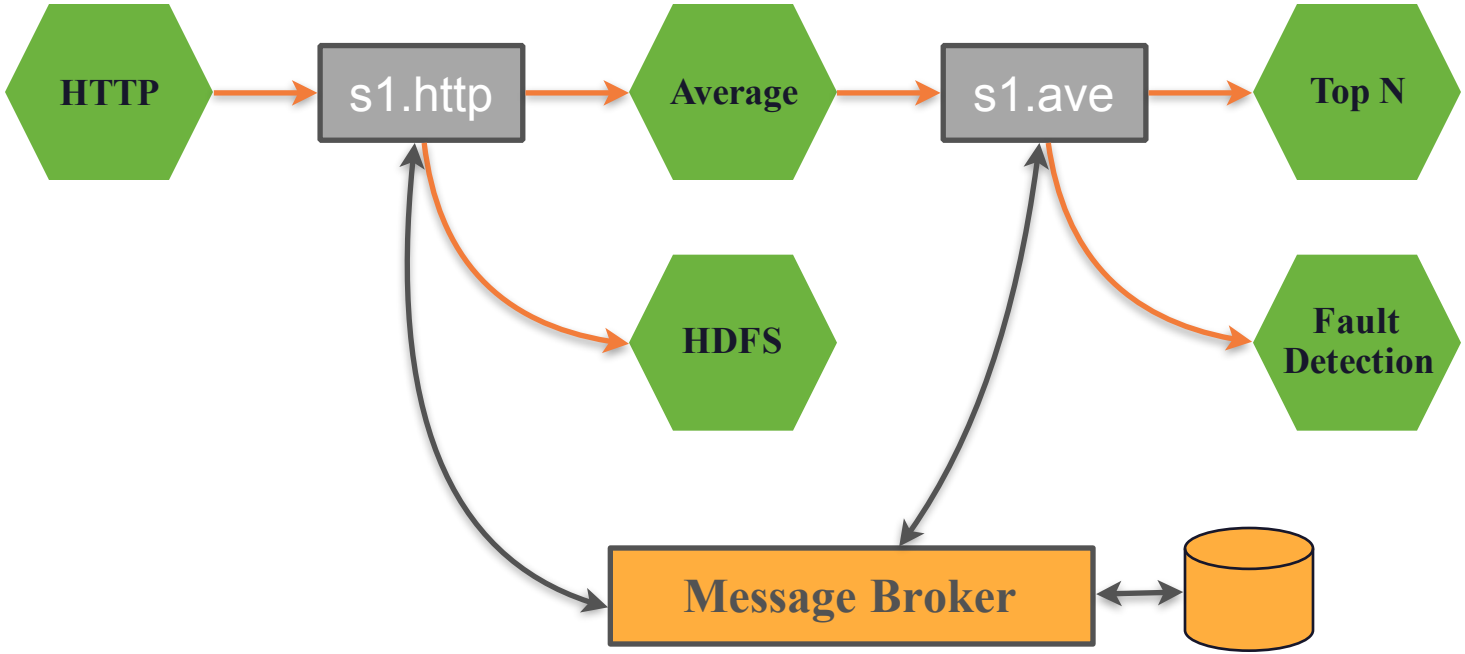


Persistent Pub-Sub

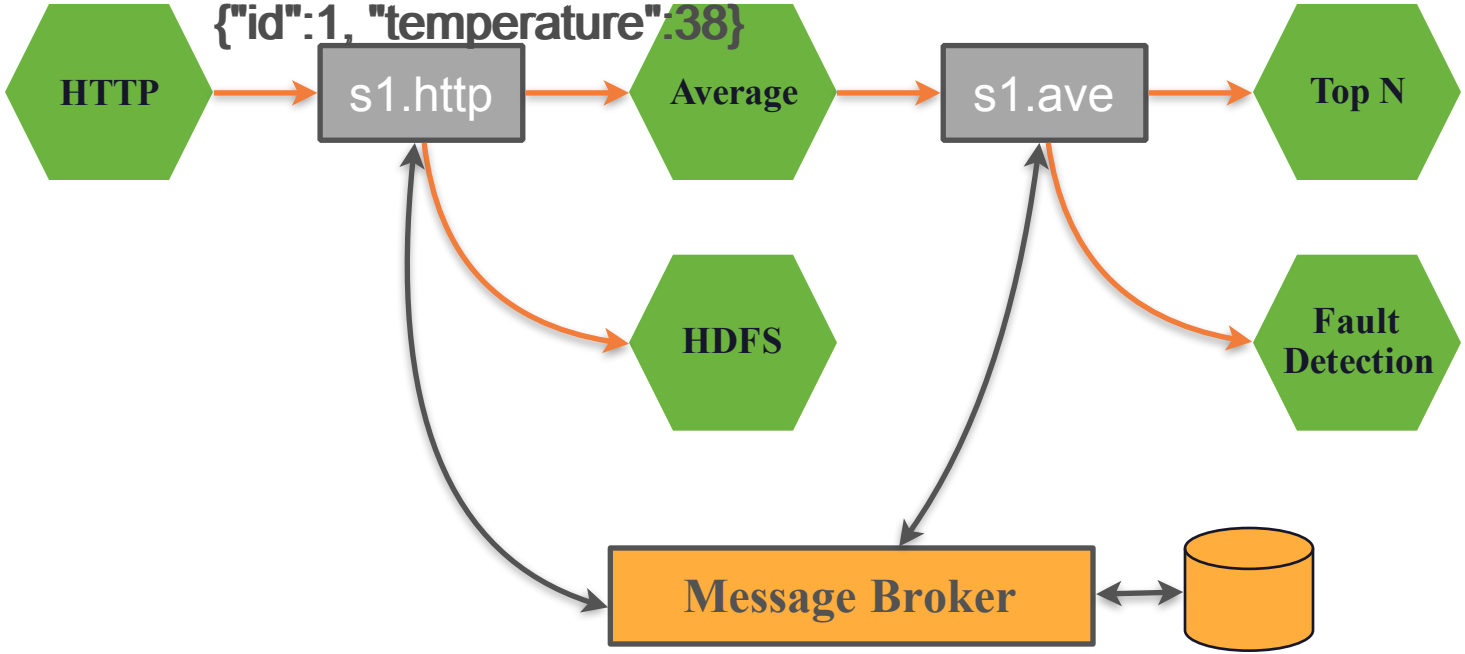


Persistent Pub-Sub

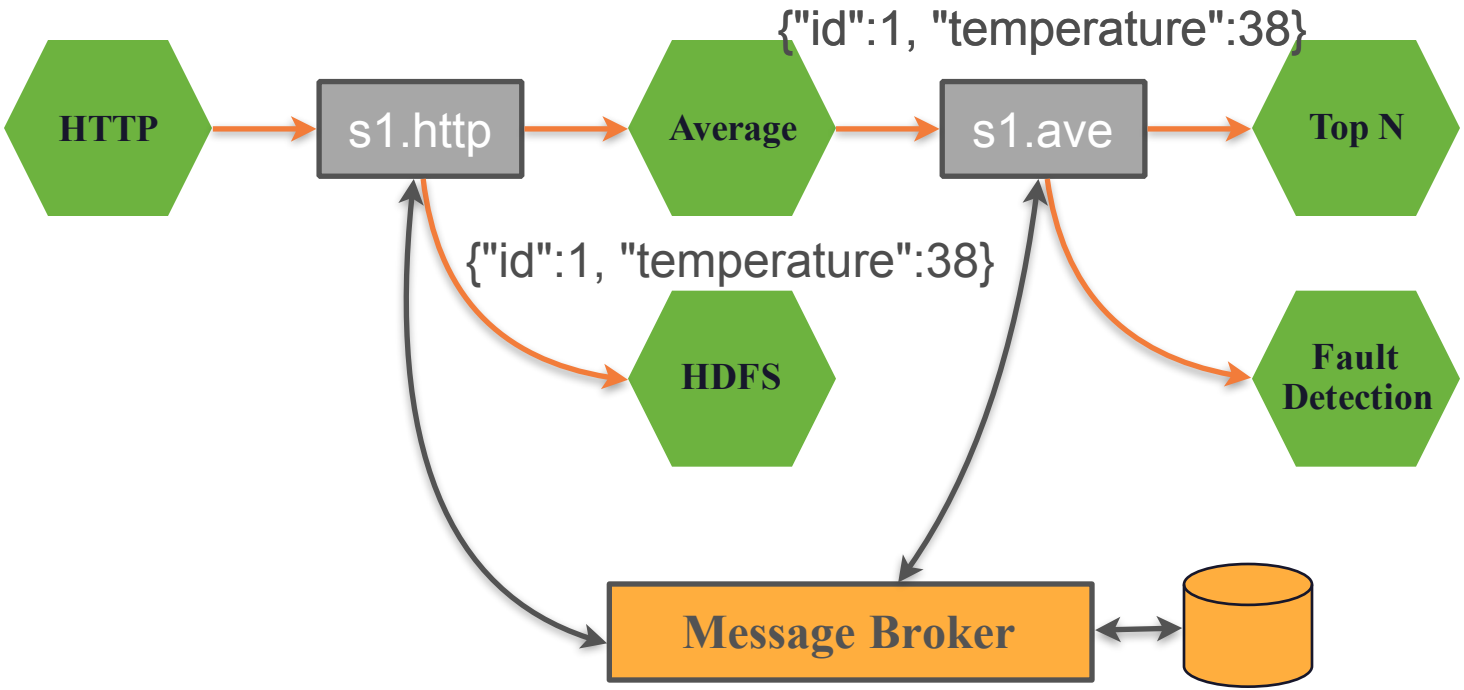
{"id":1, "temperature":38}



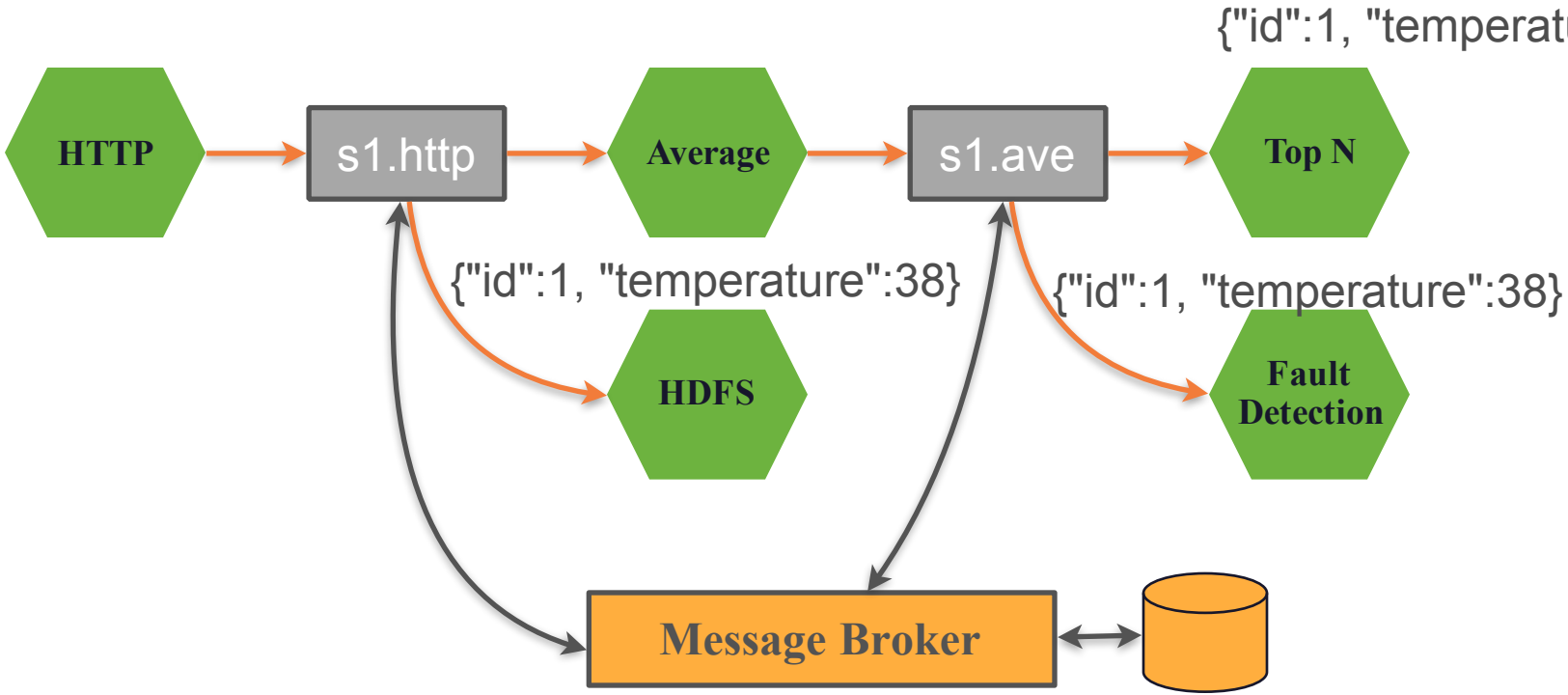
Persistent Pub-Sub

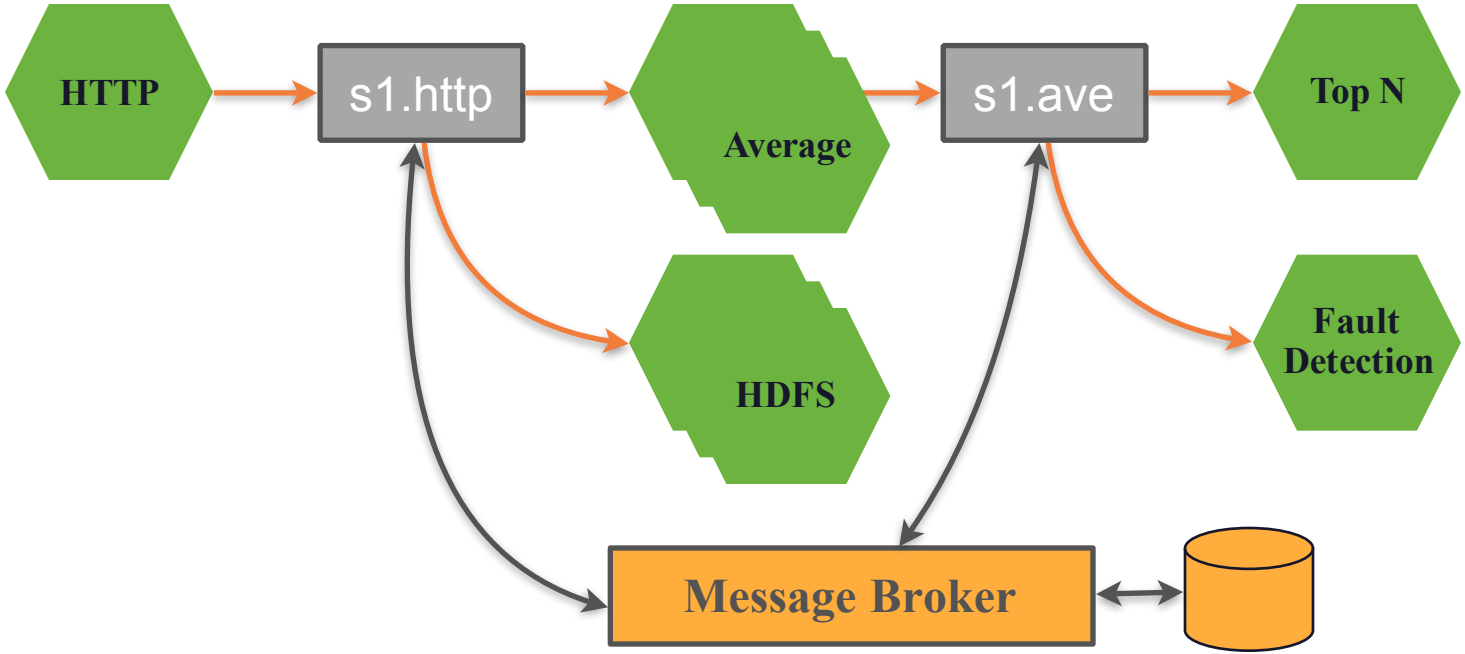


Persistent Pub-Sub

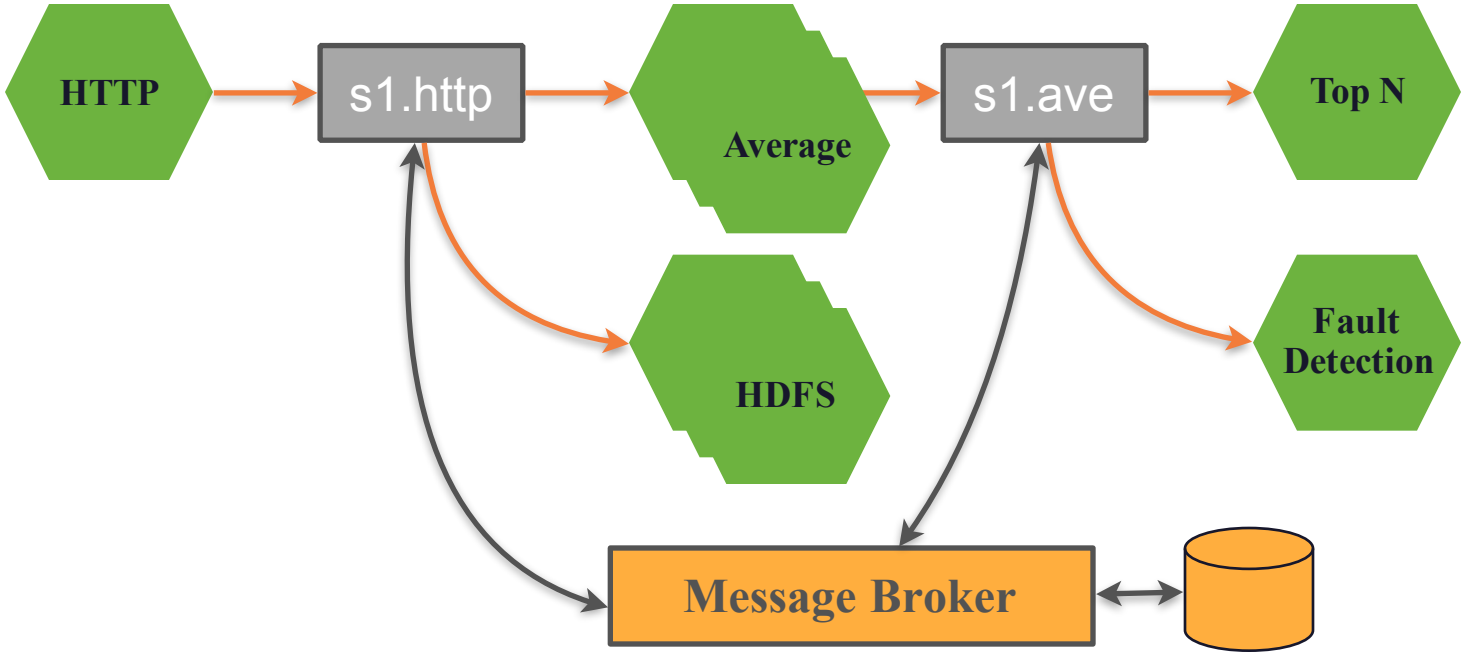


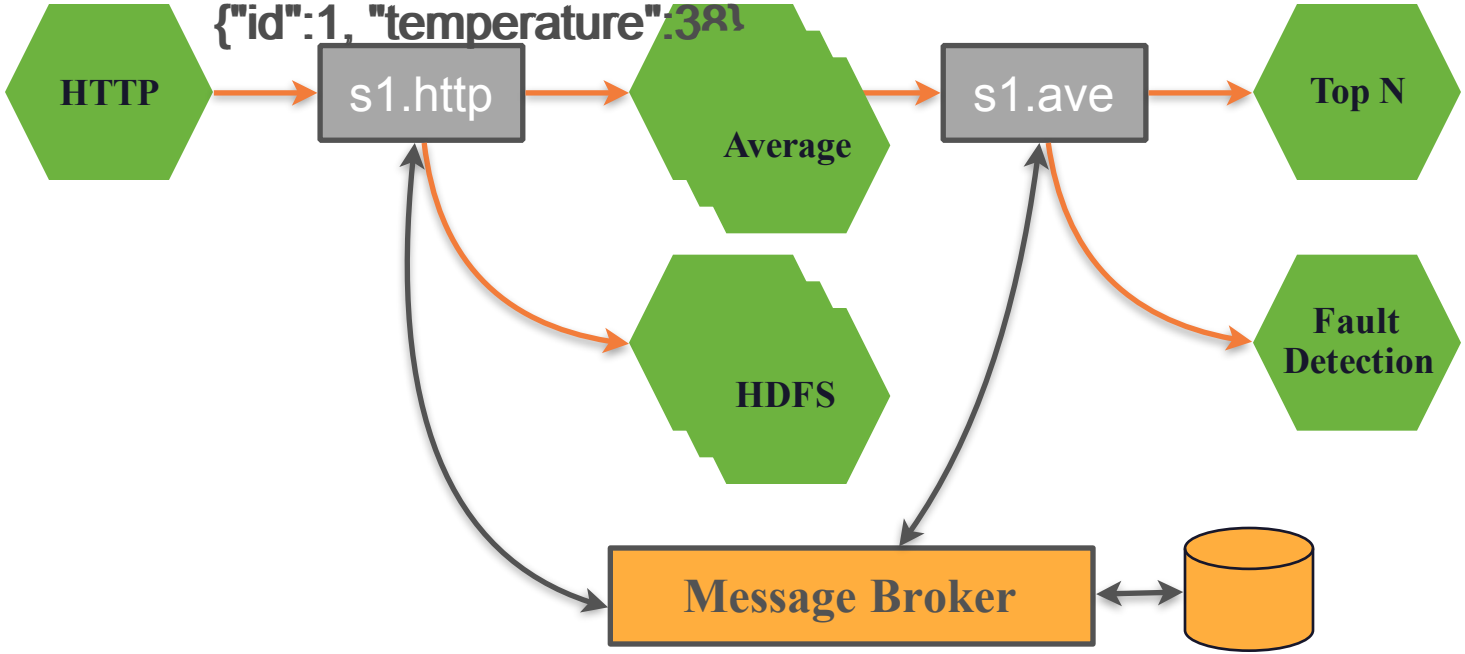
Persistent Pub-Sub

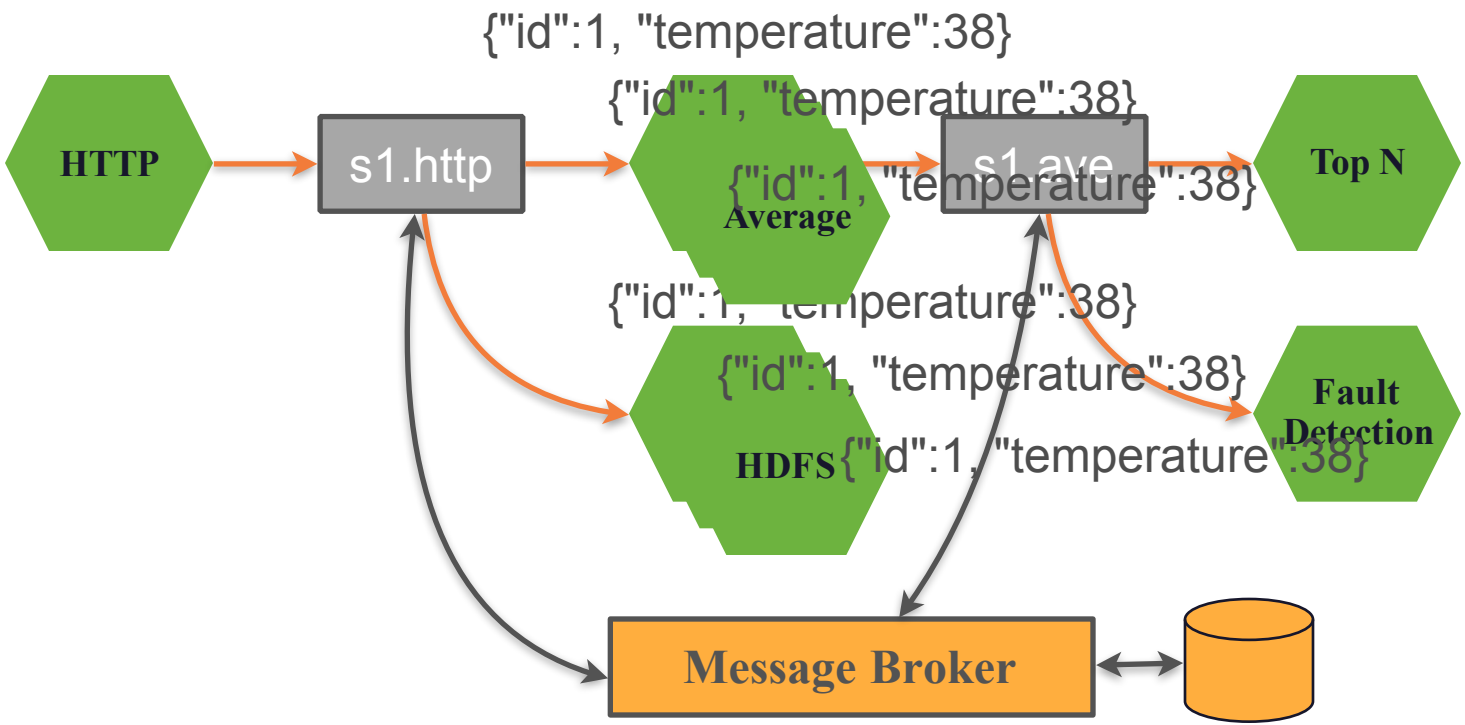


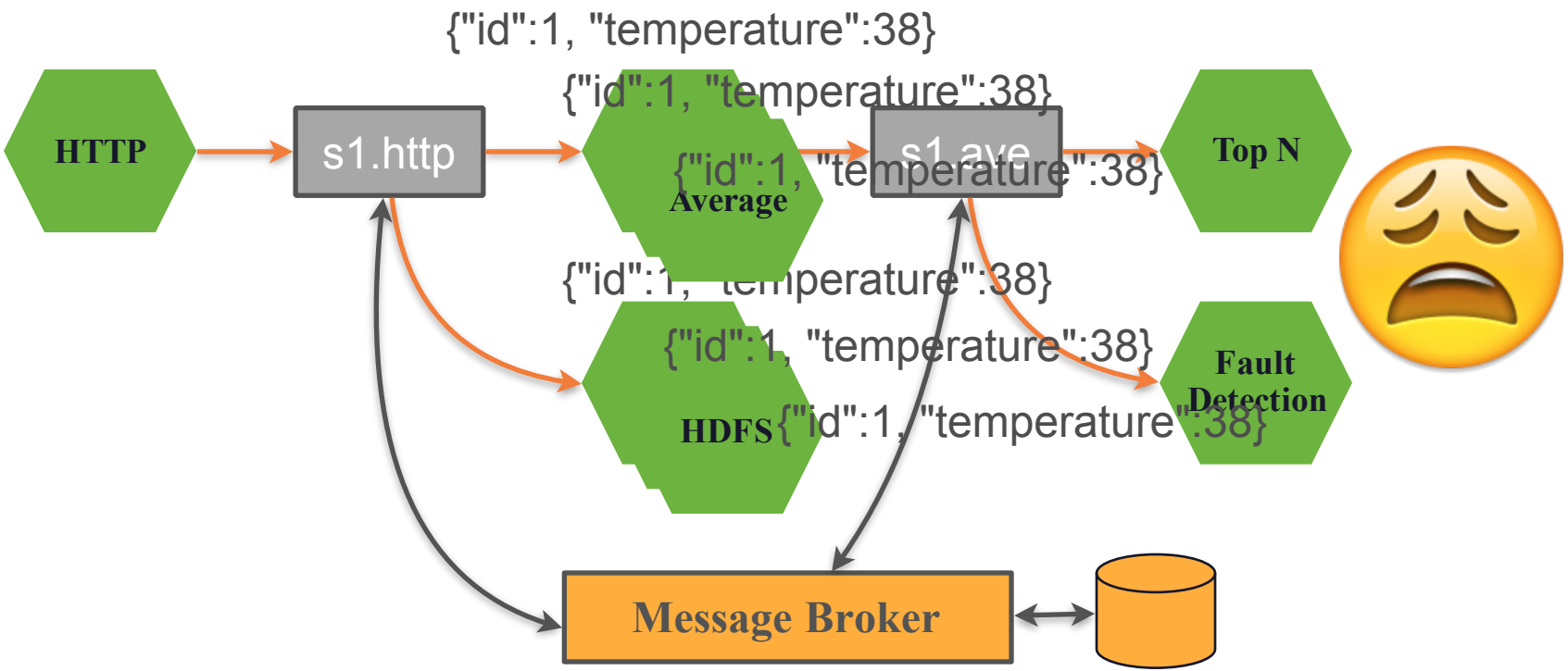


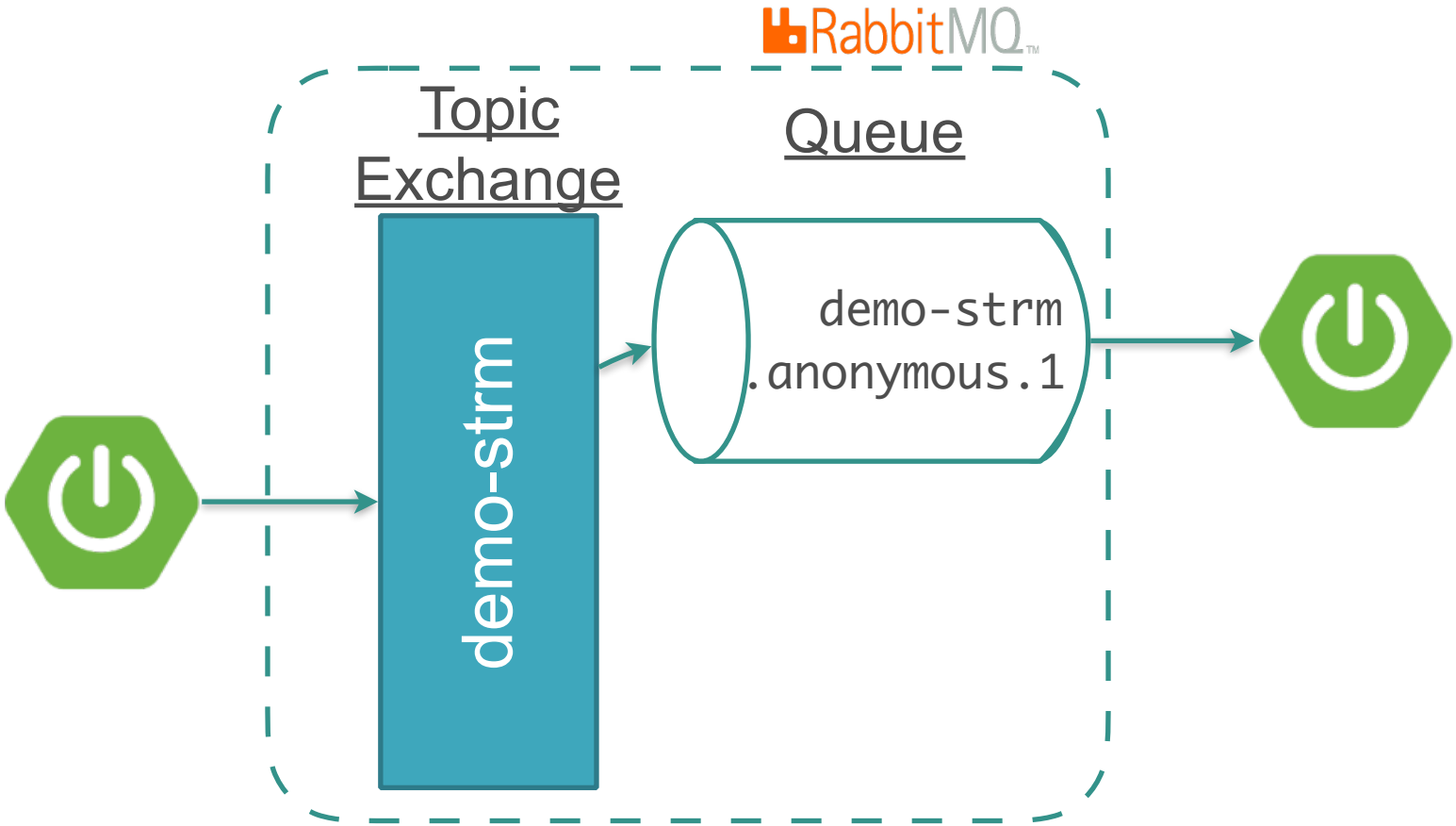
{"id":1, "temperature":38}

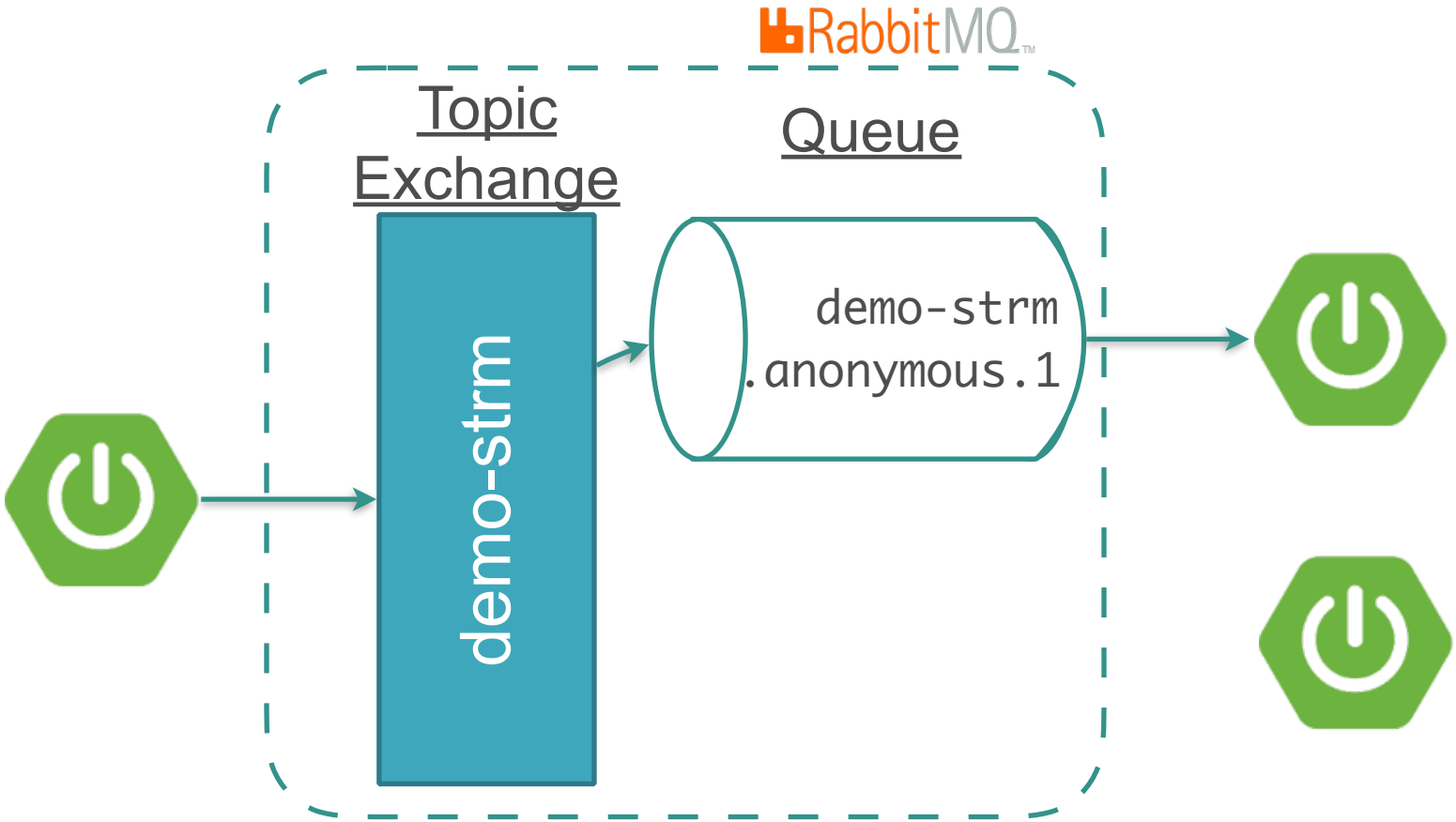


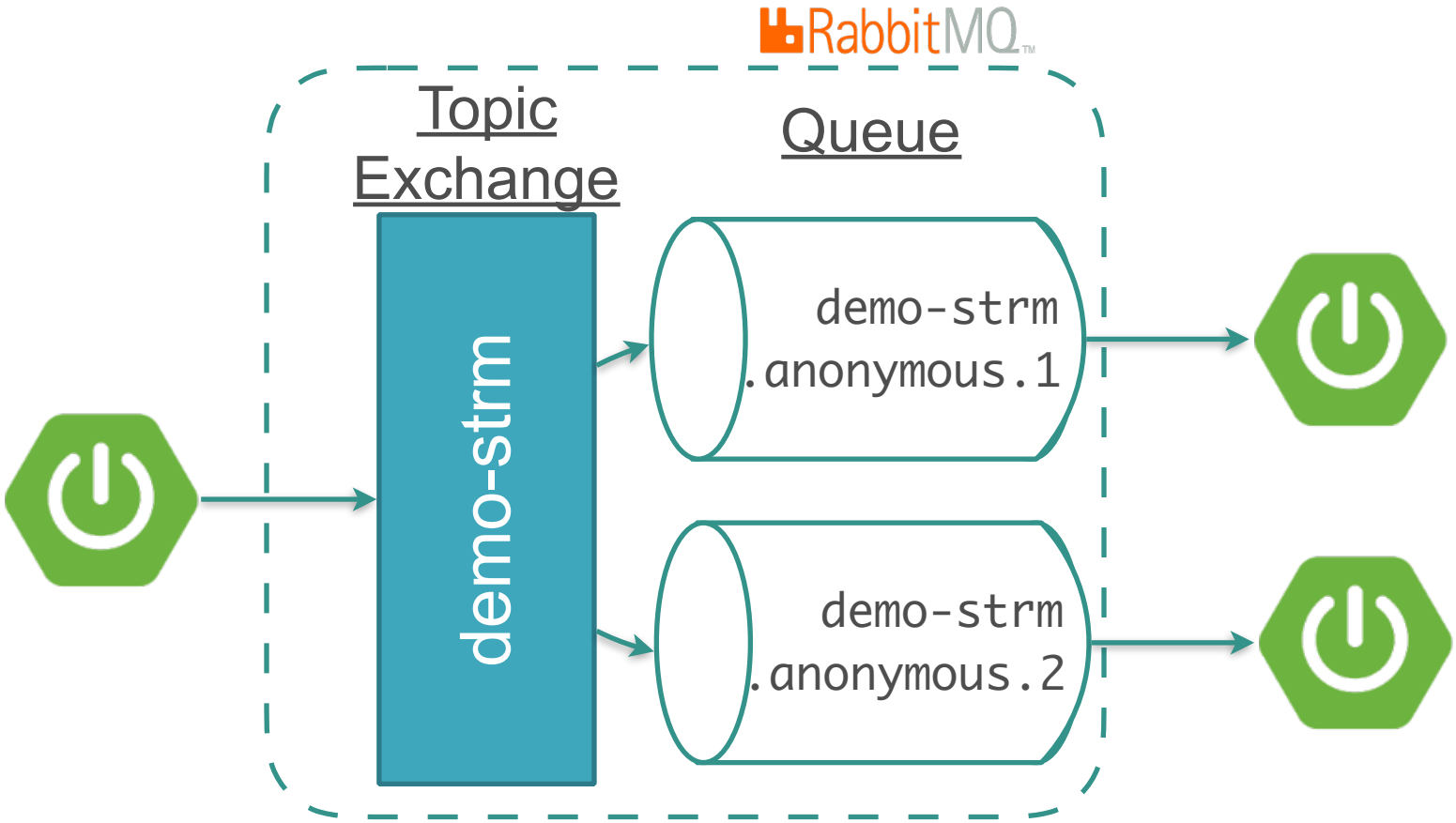




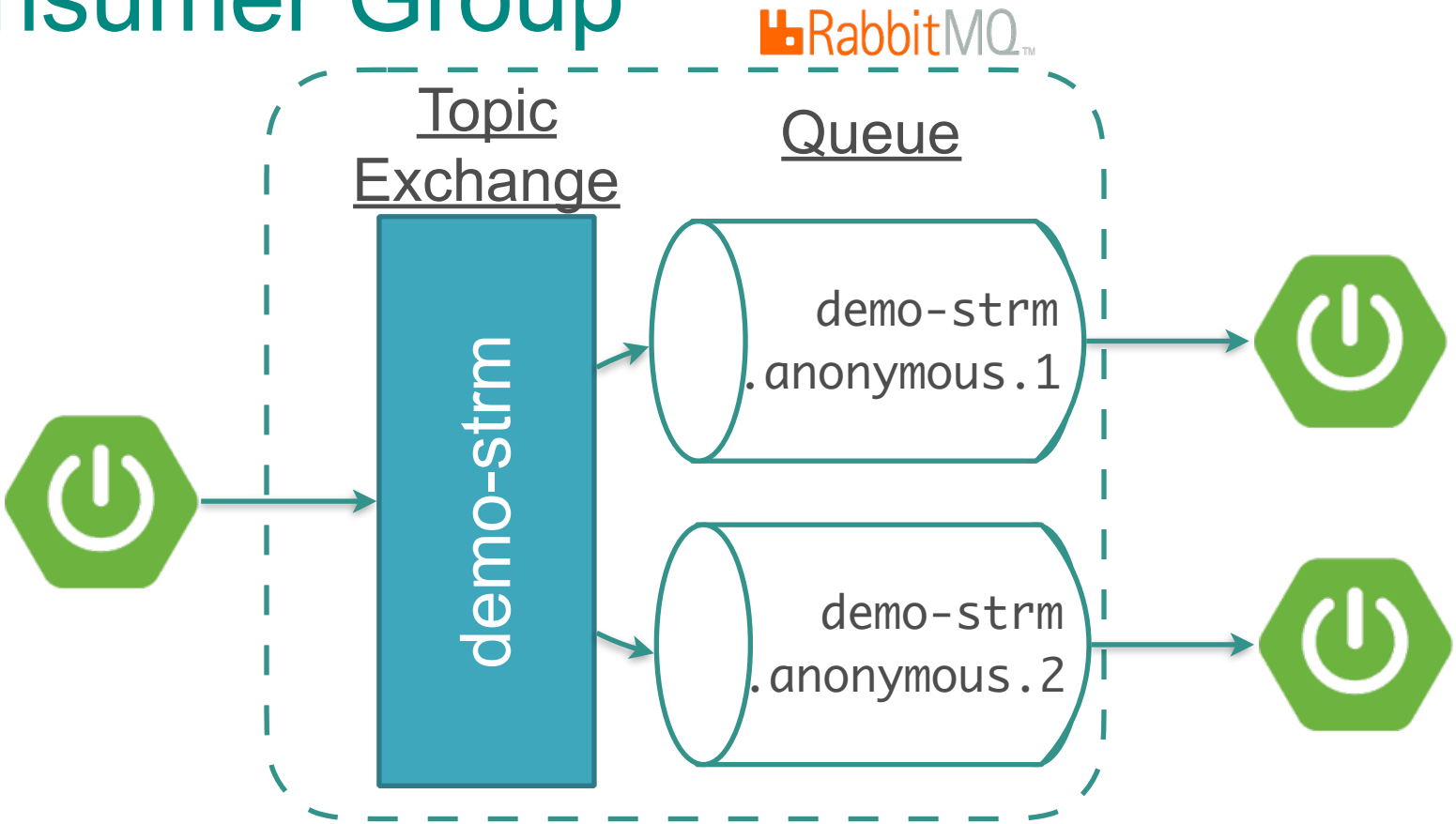




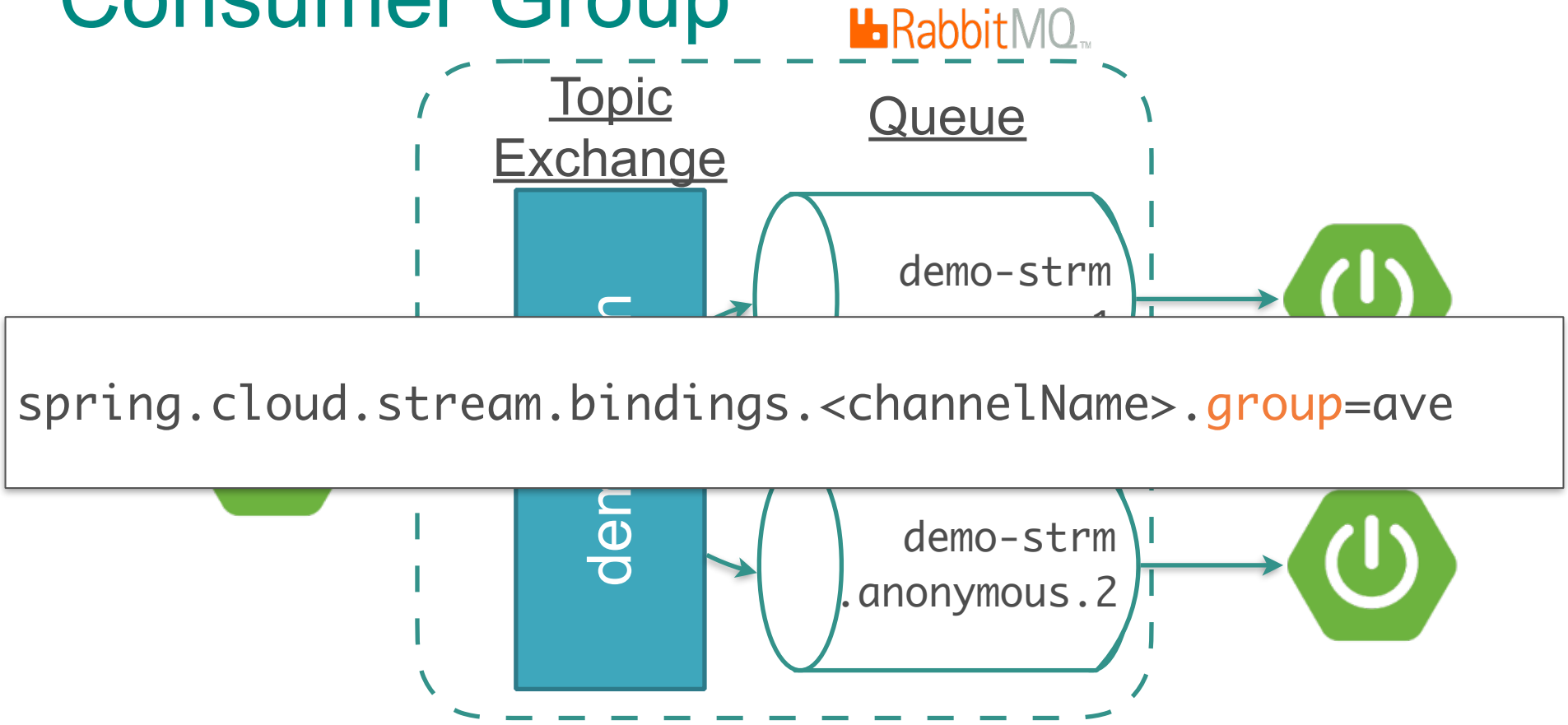




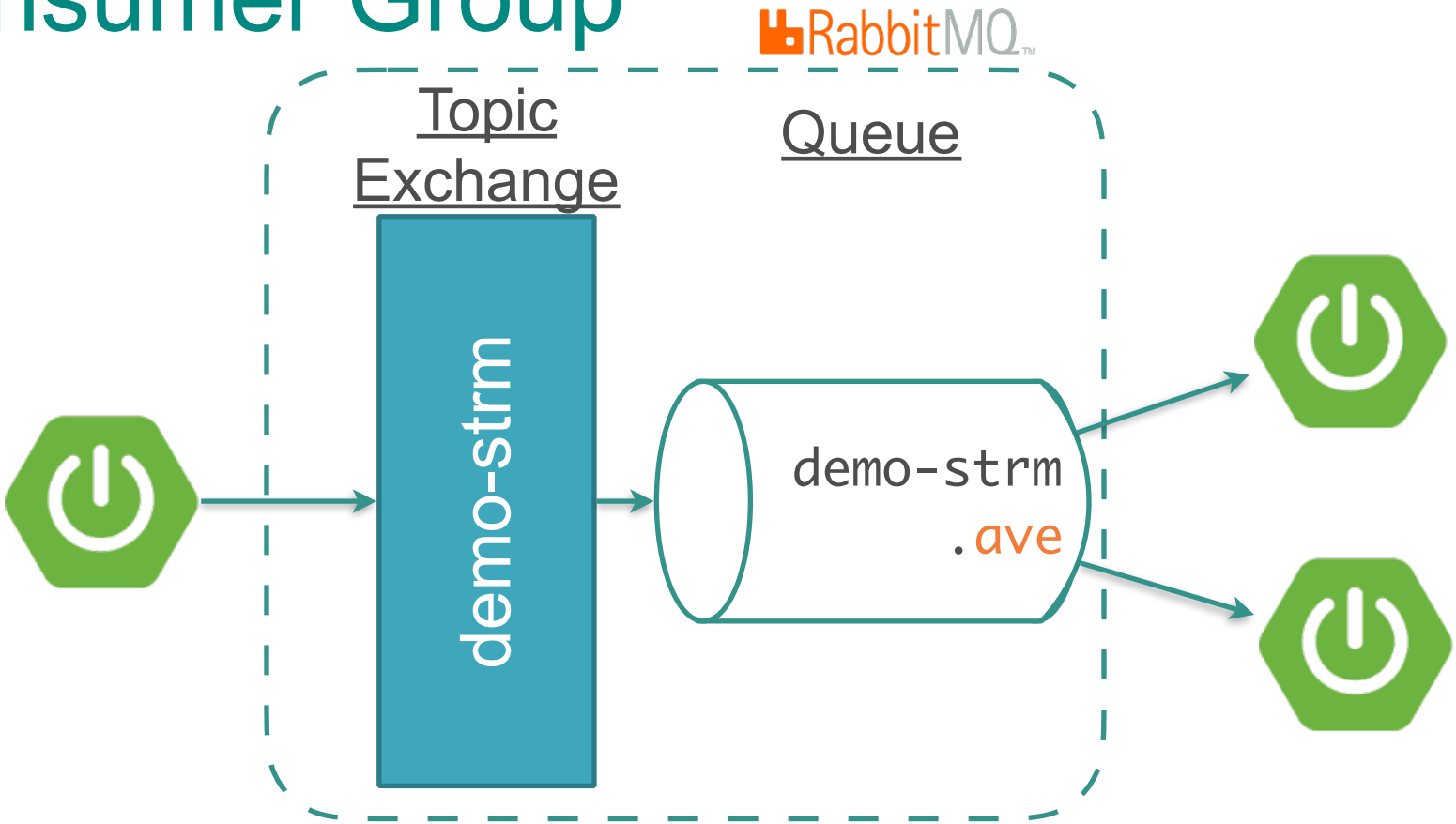
Consumer Group



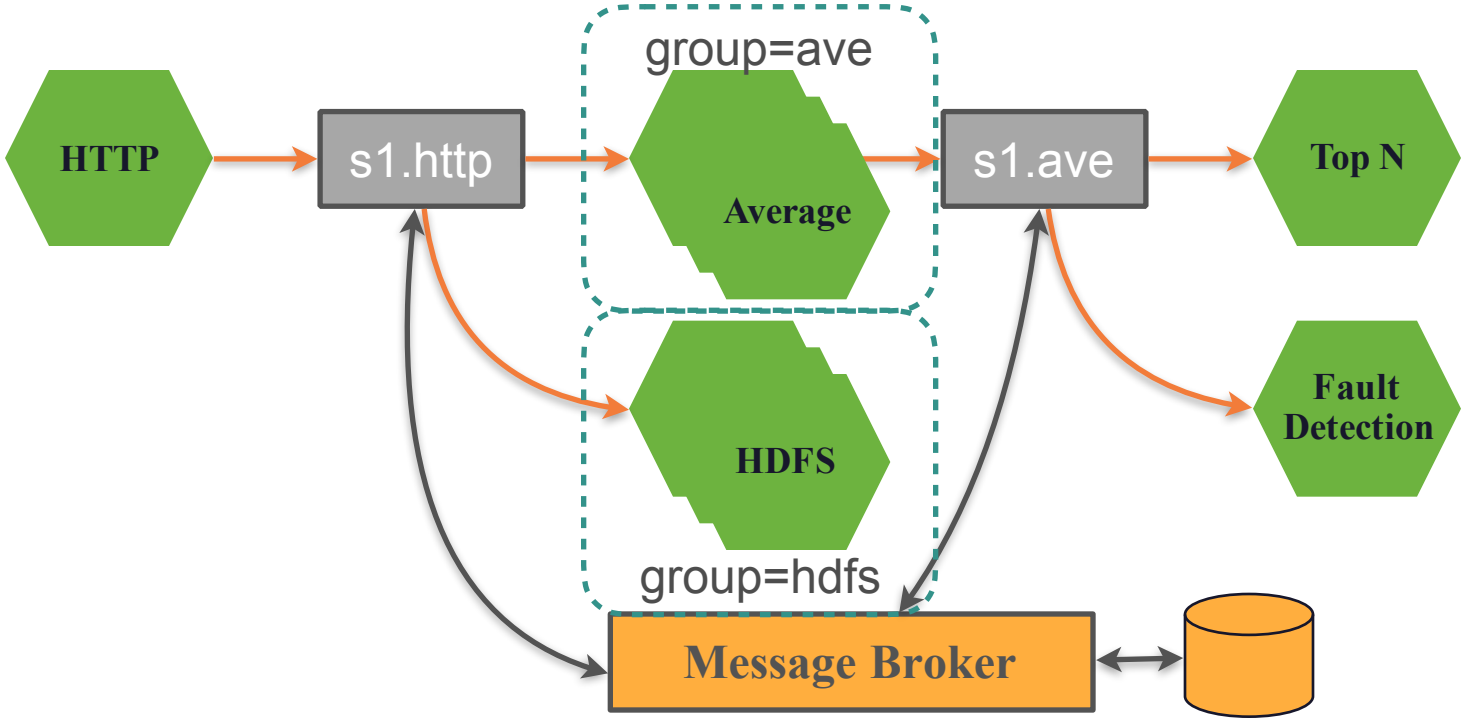
Consumer Group



Consumer Group

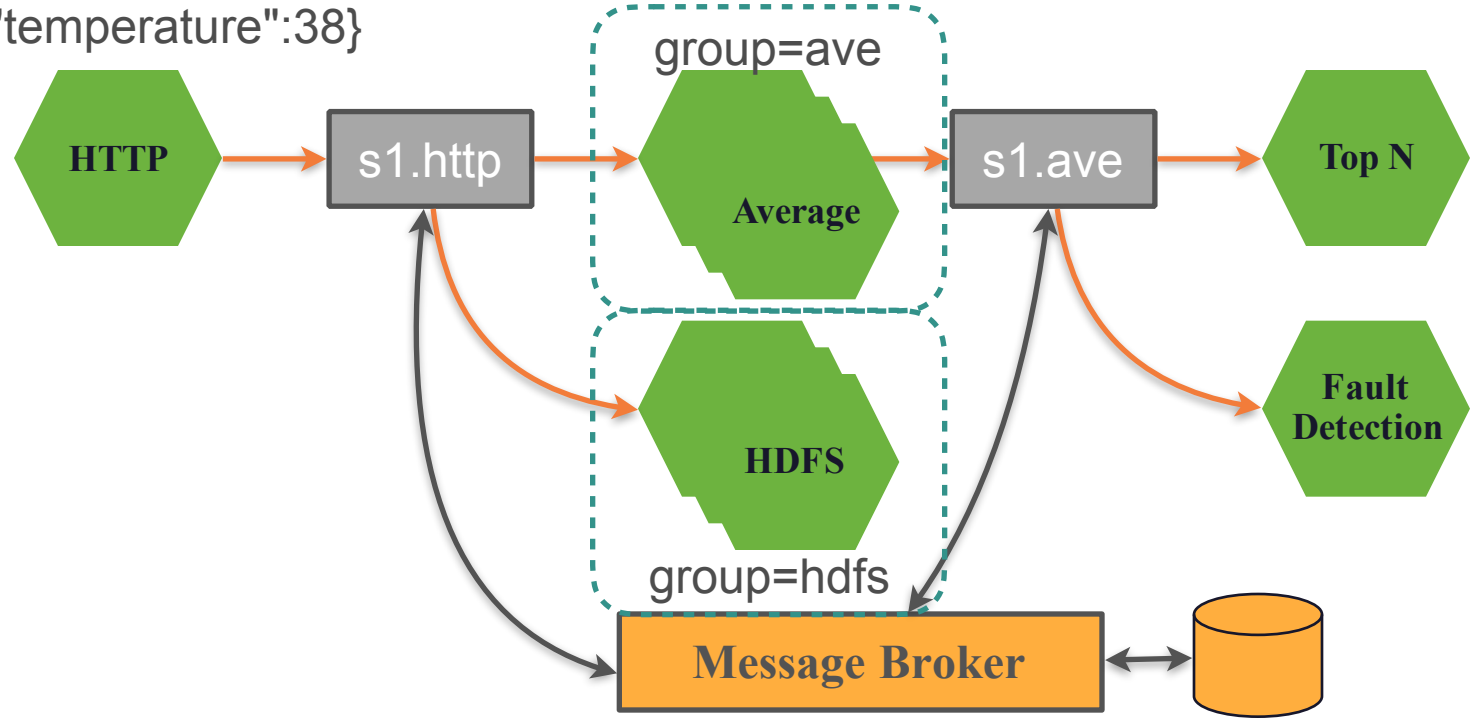


Consumer Group

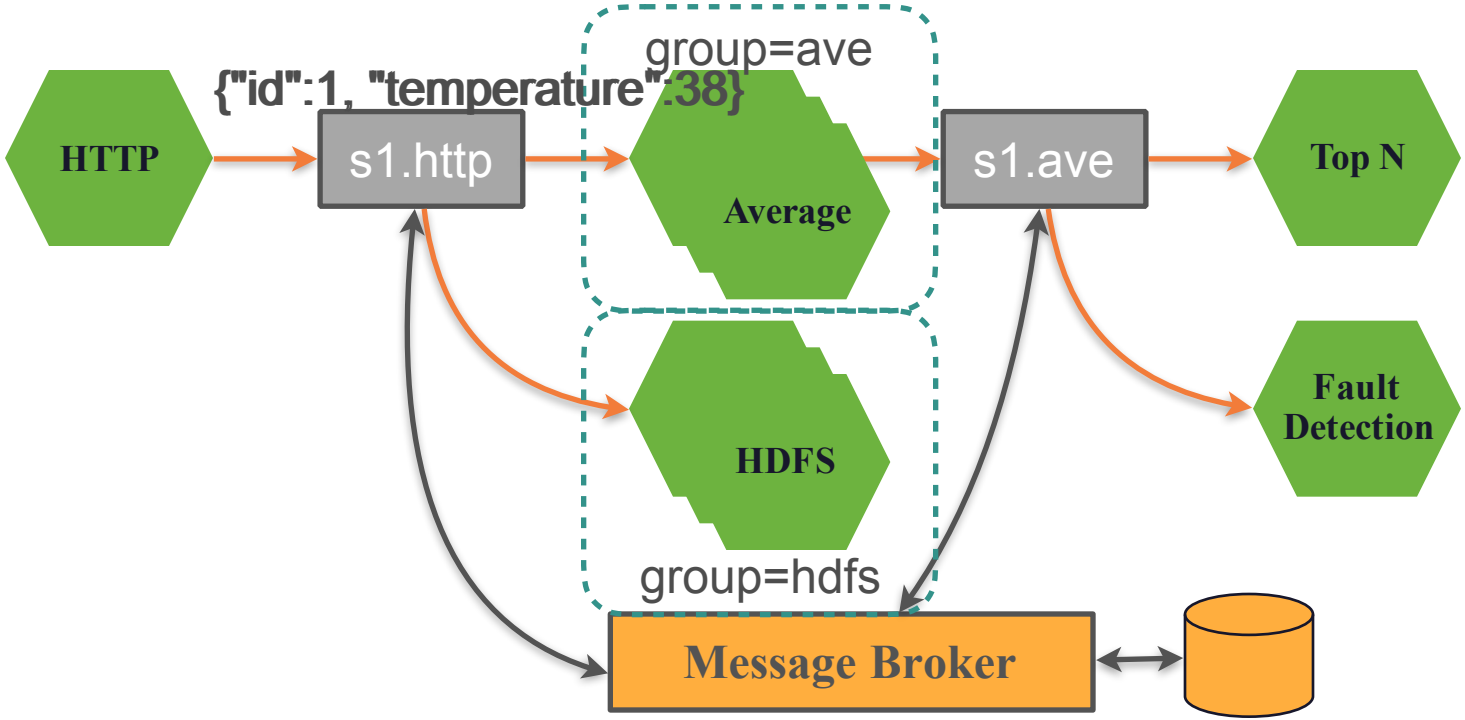


Consumer Group

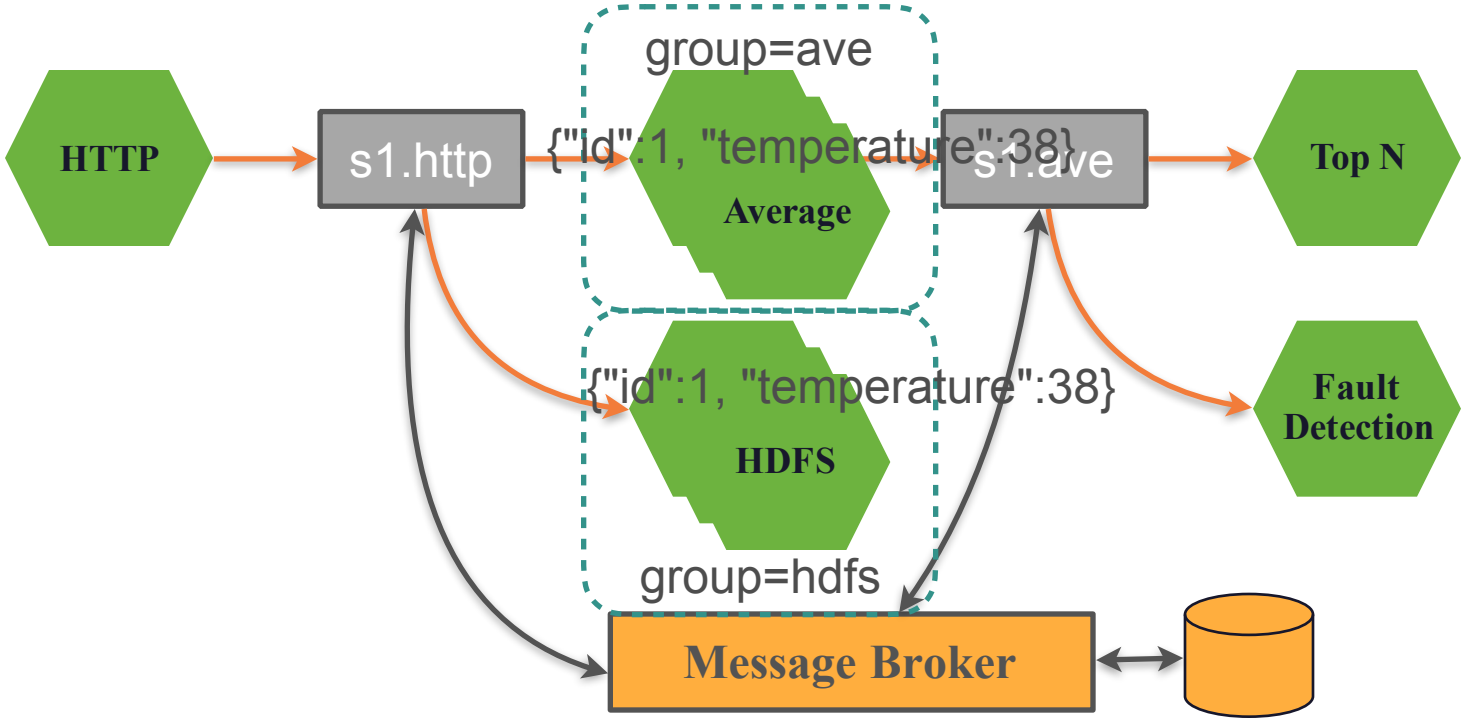
{"id":1, "temperature":38}



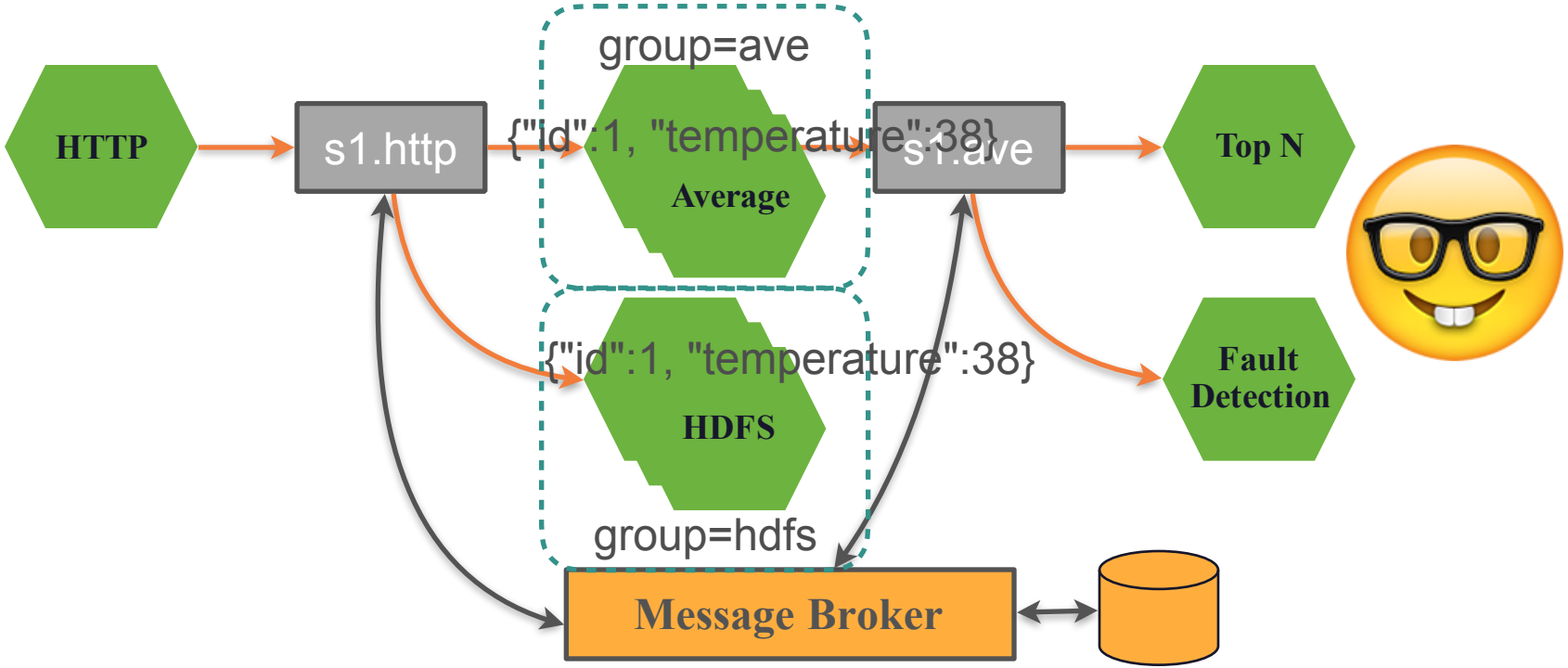
Consumer Group



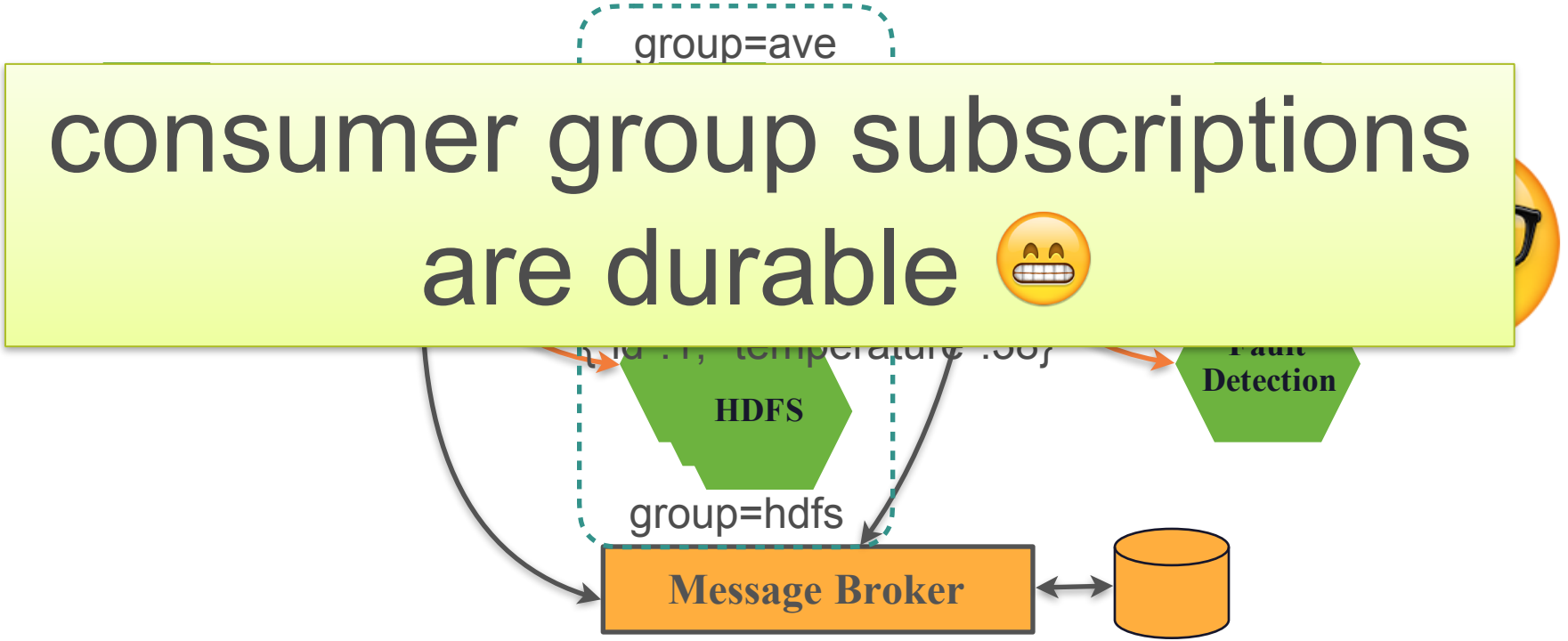
Consumer Group



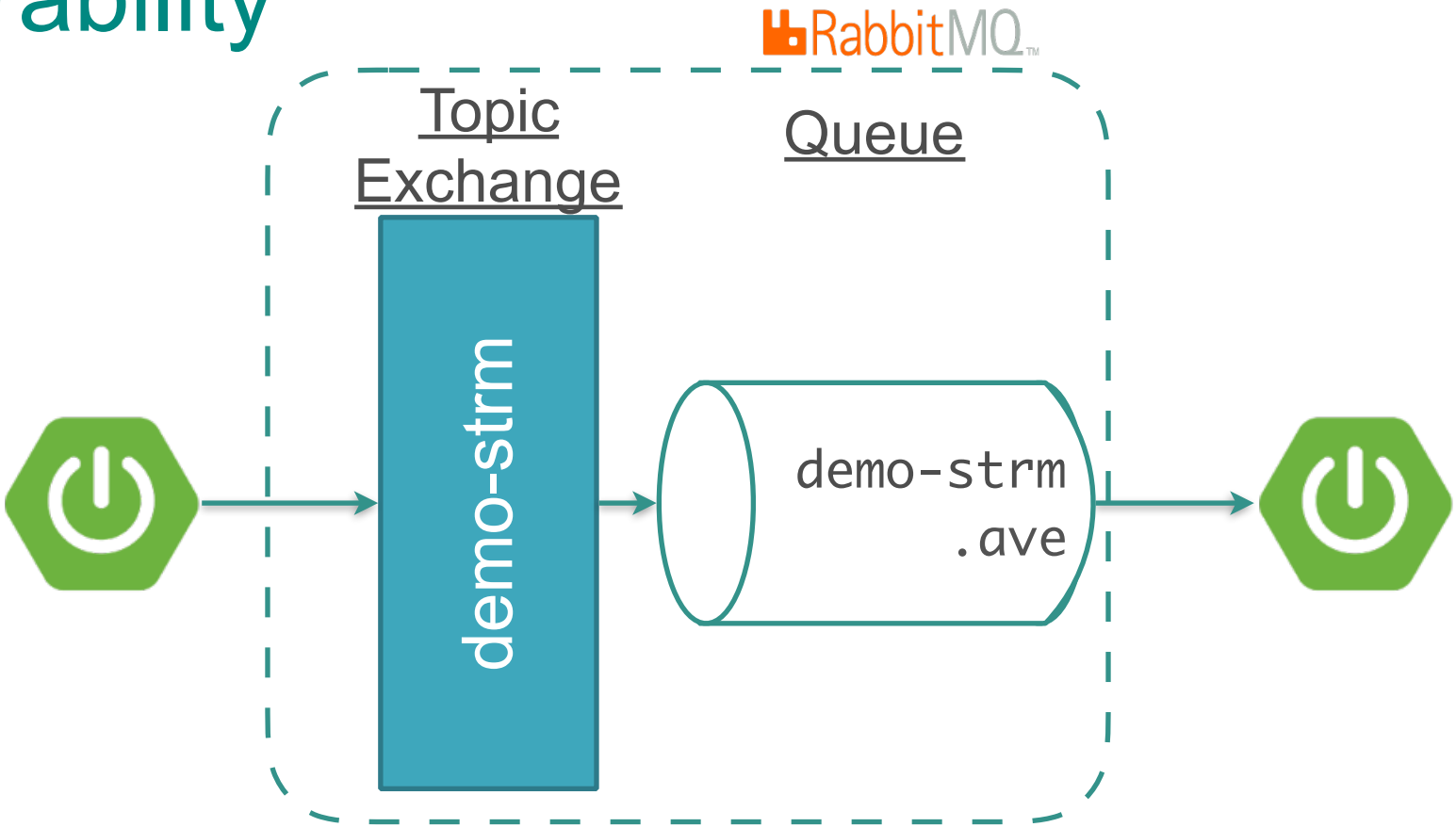
Consumer Group



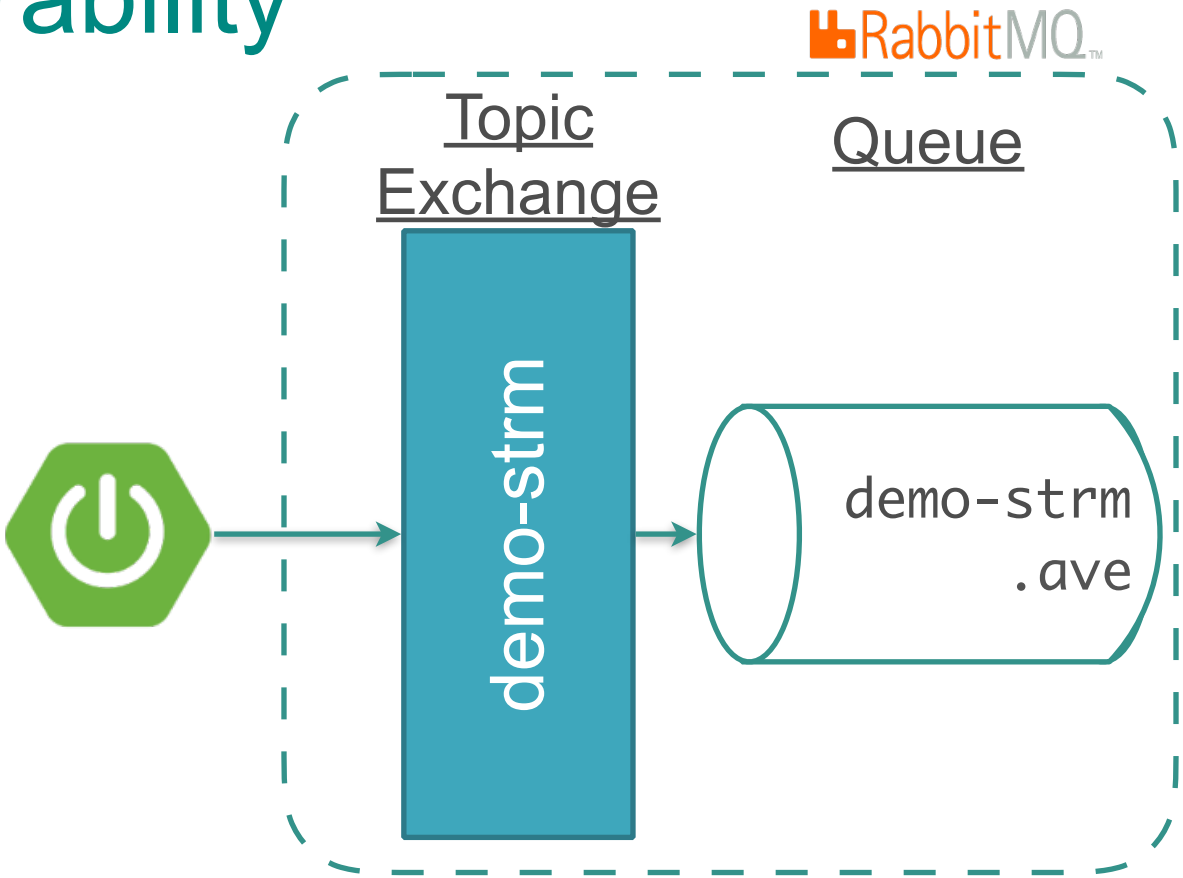
Consumer Group



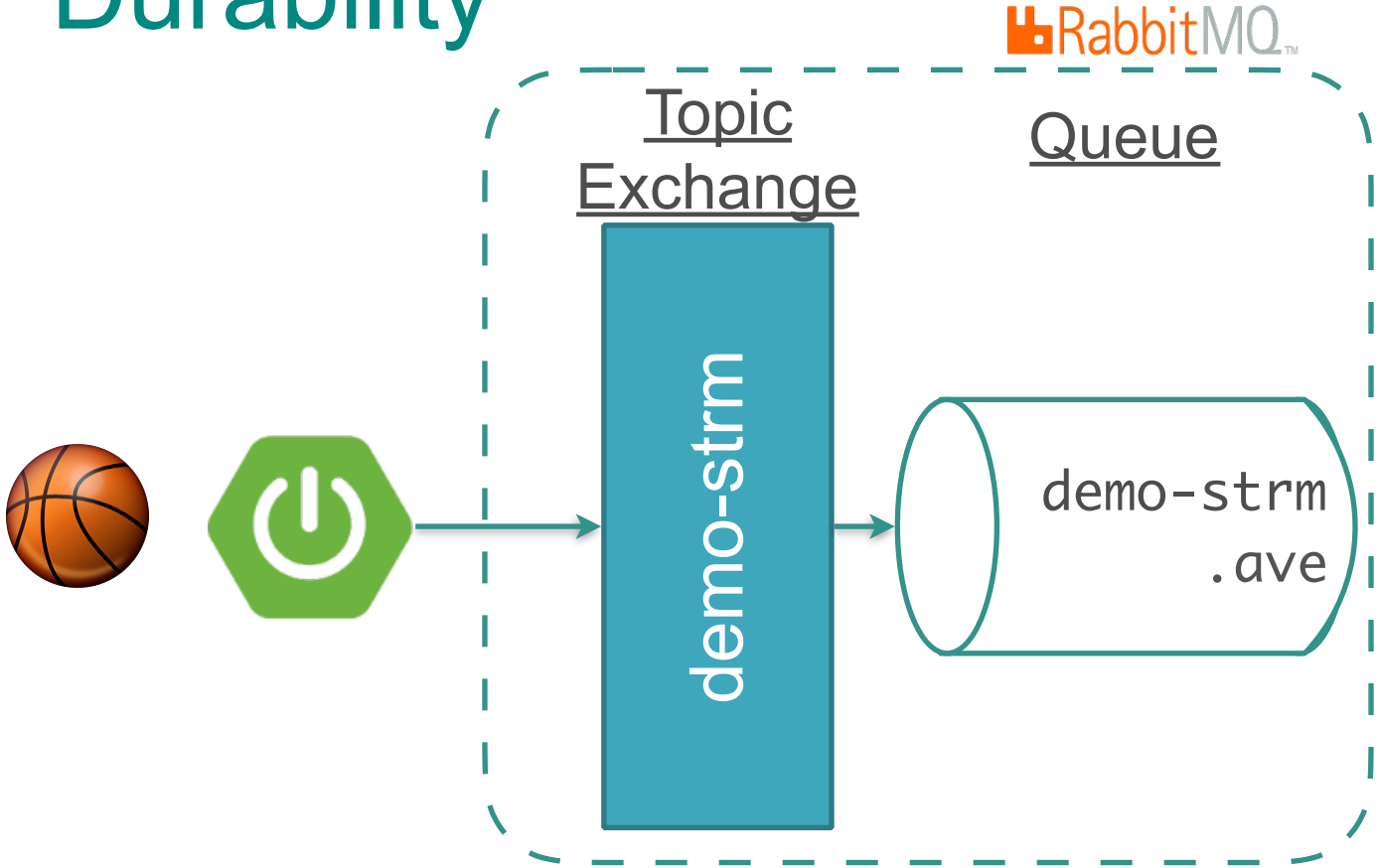
Durability



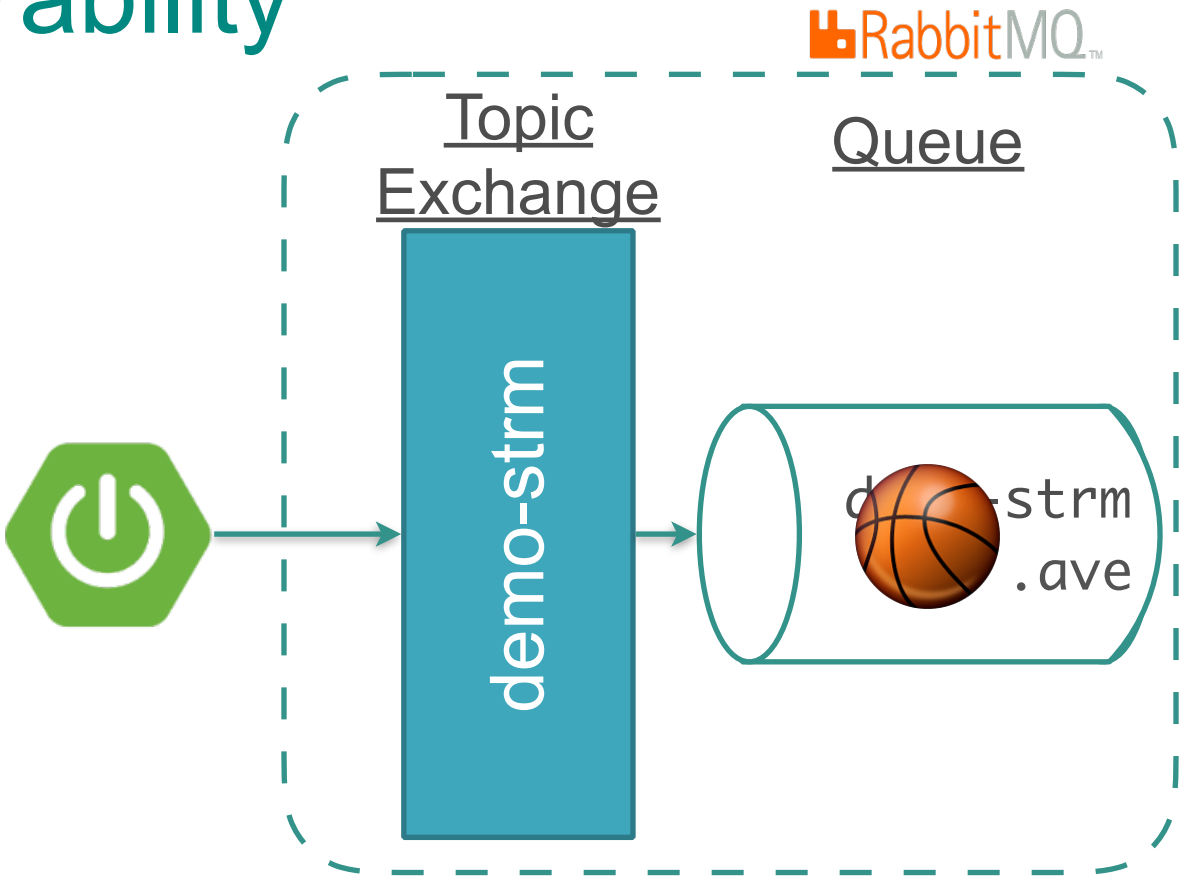
Durability



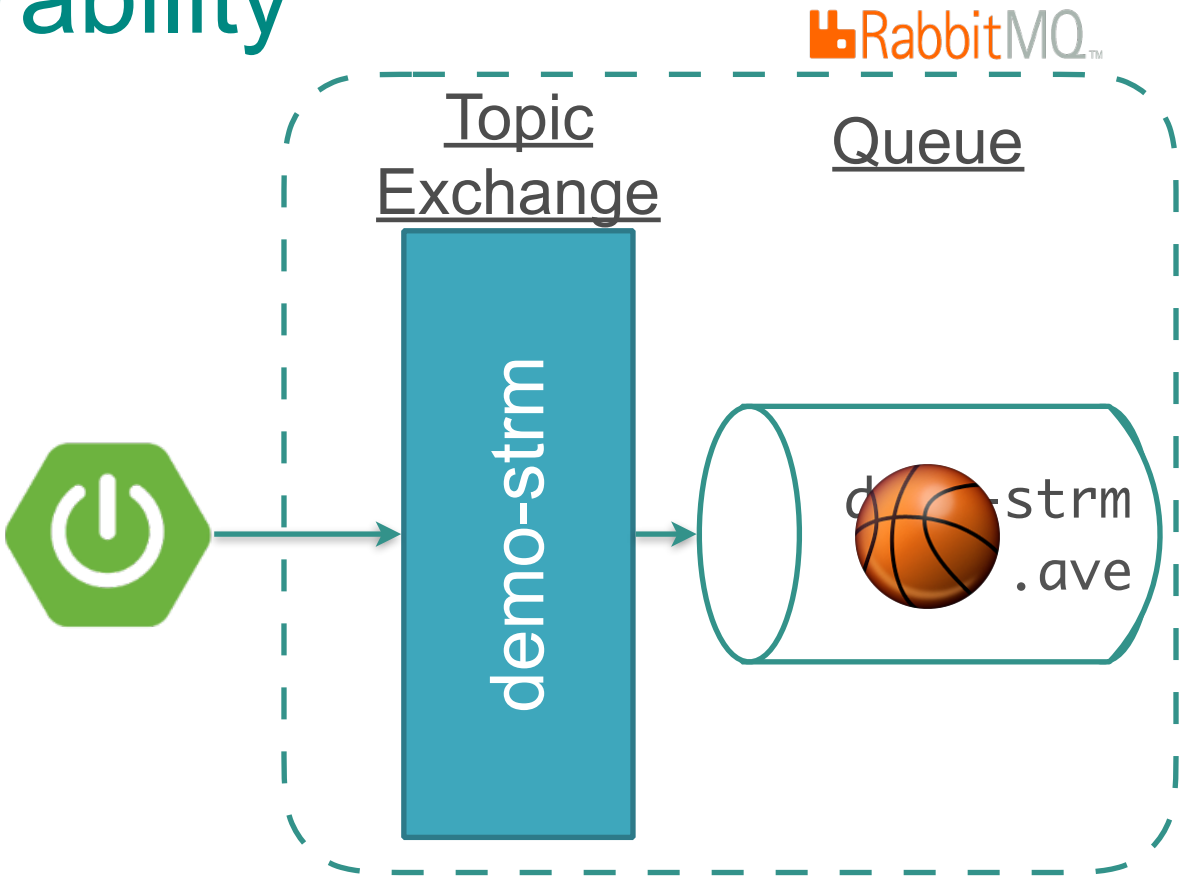
Durability



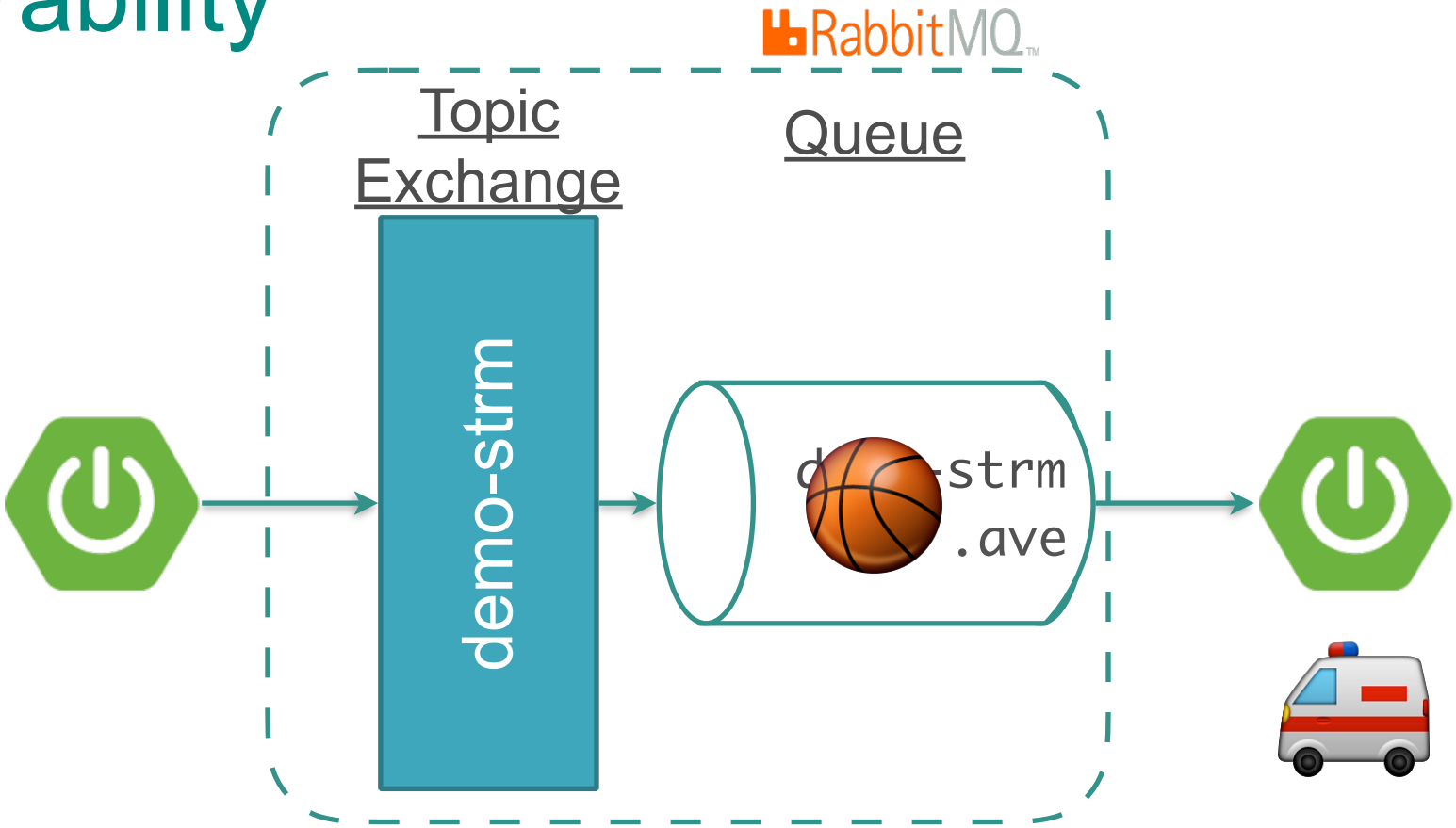
Durability



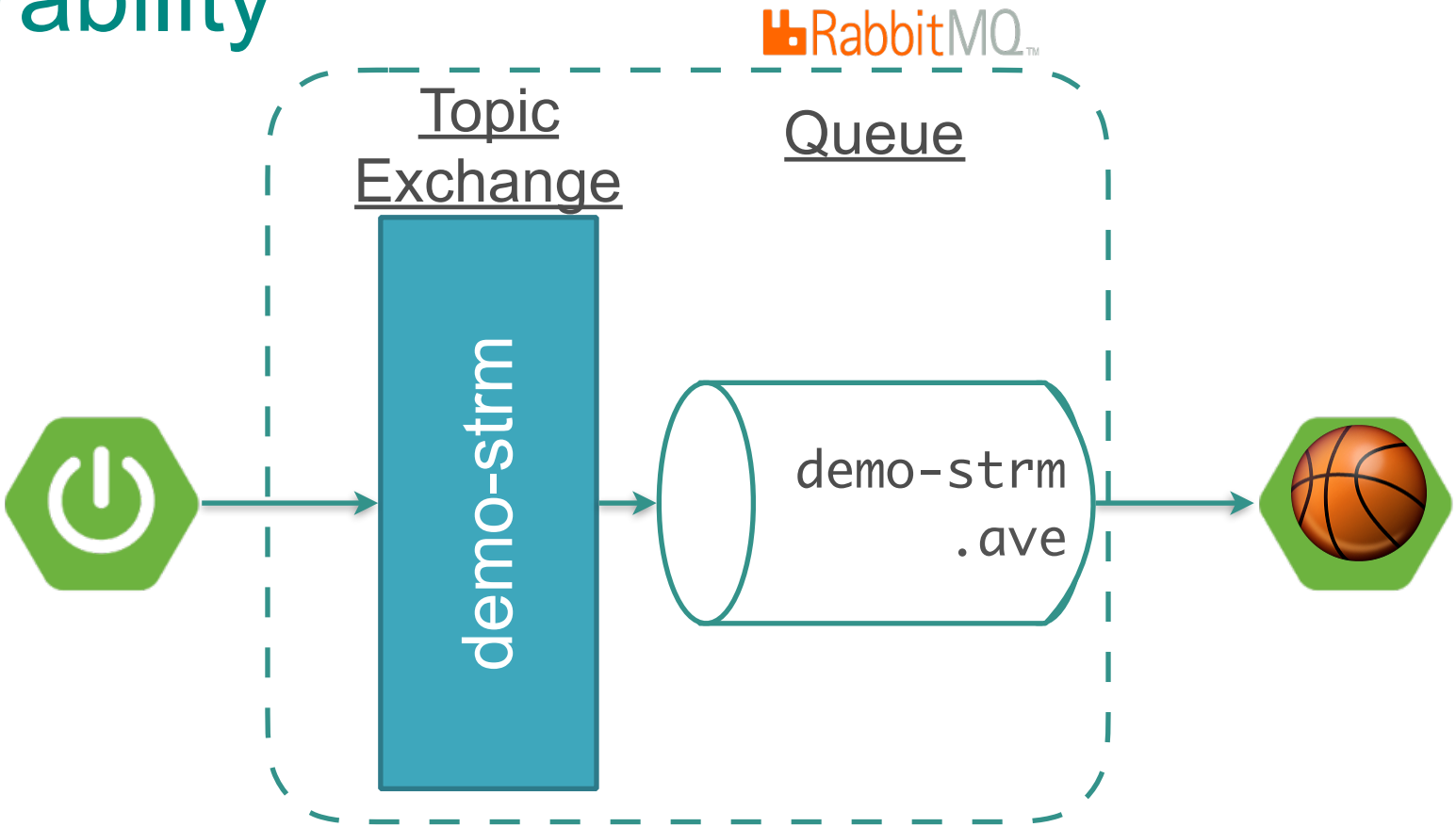
Durability



Durability



Durability



Sink

Point Service

Source

Customer Service

Email Service

Post Service

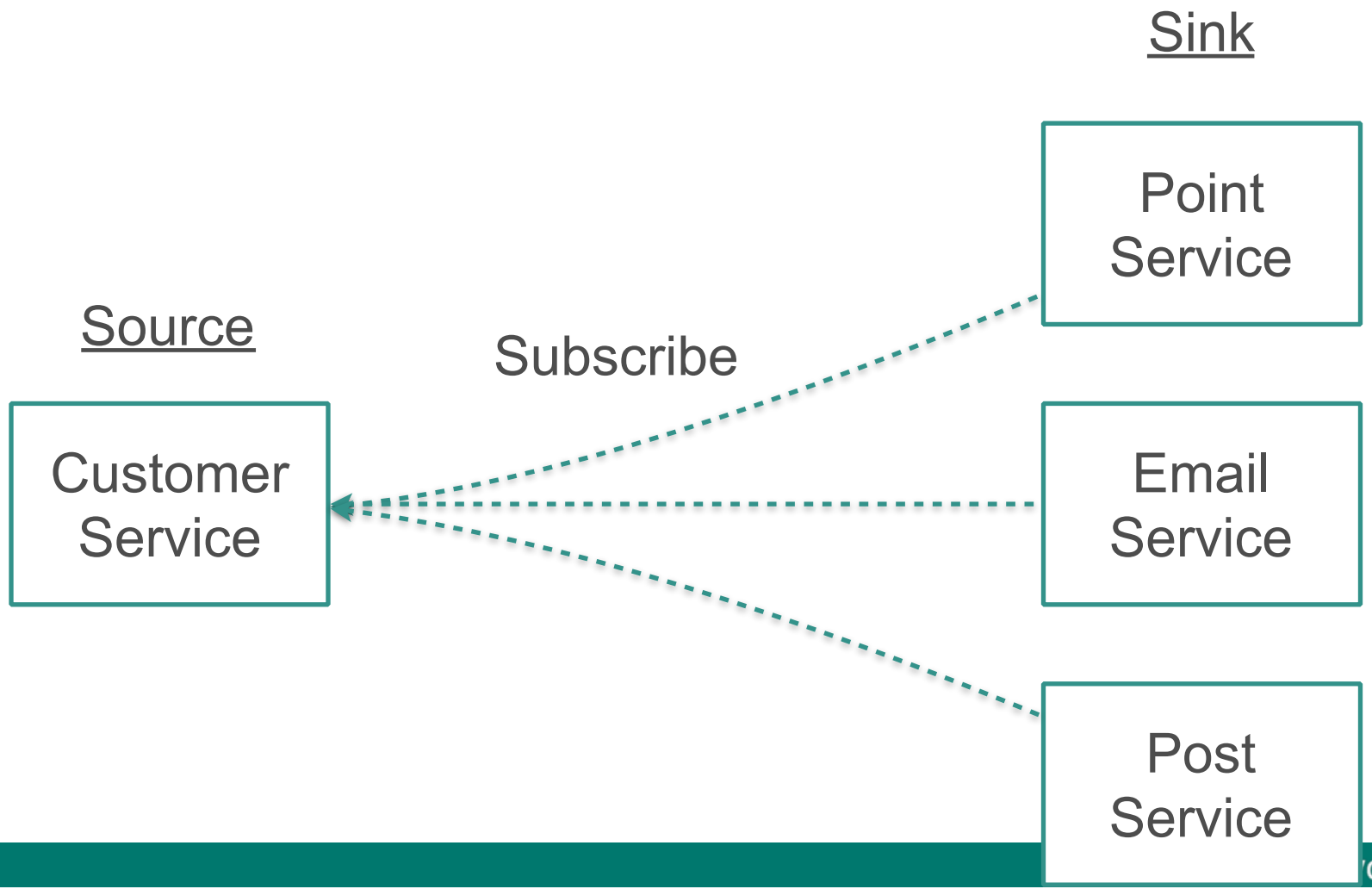
Sink

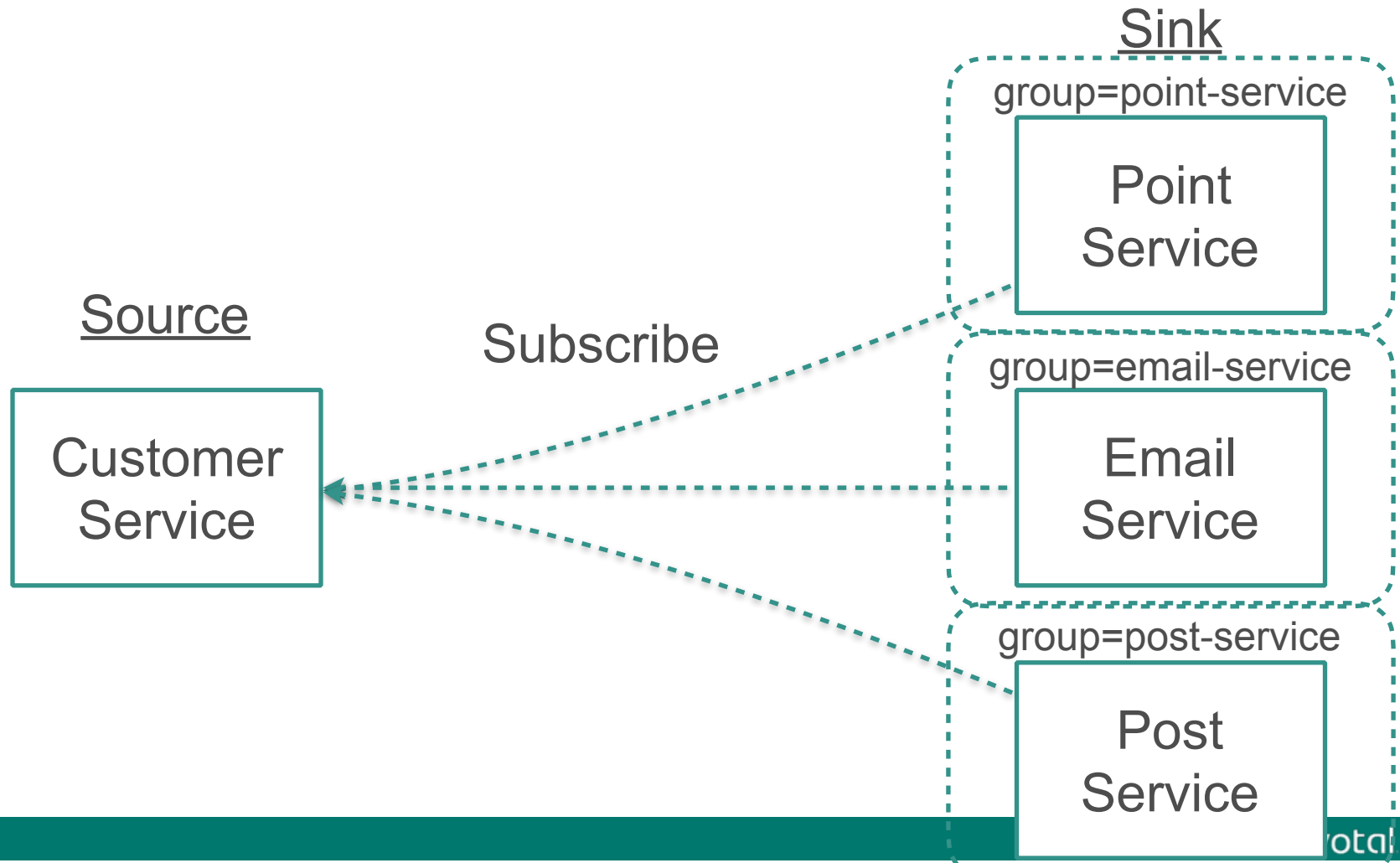
Point Service

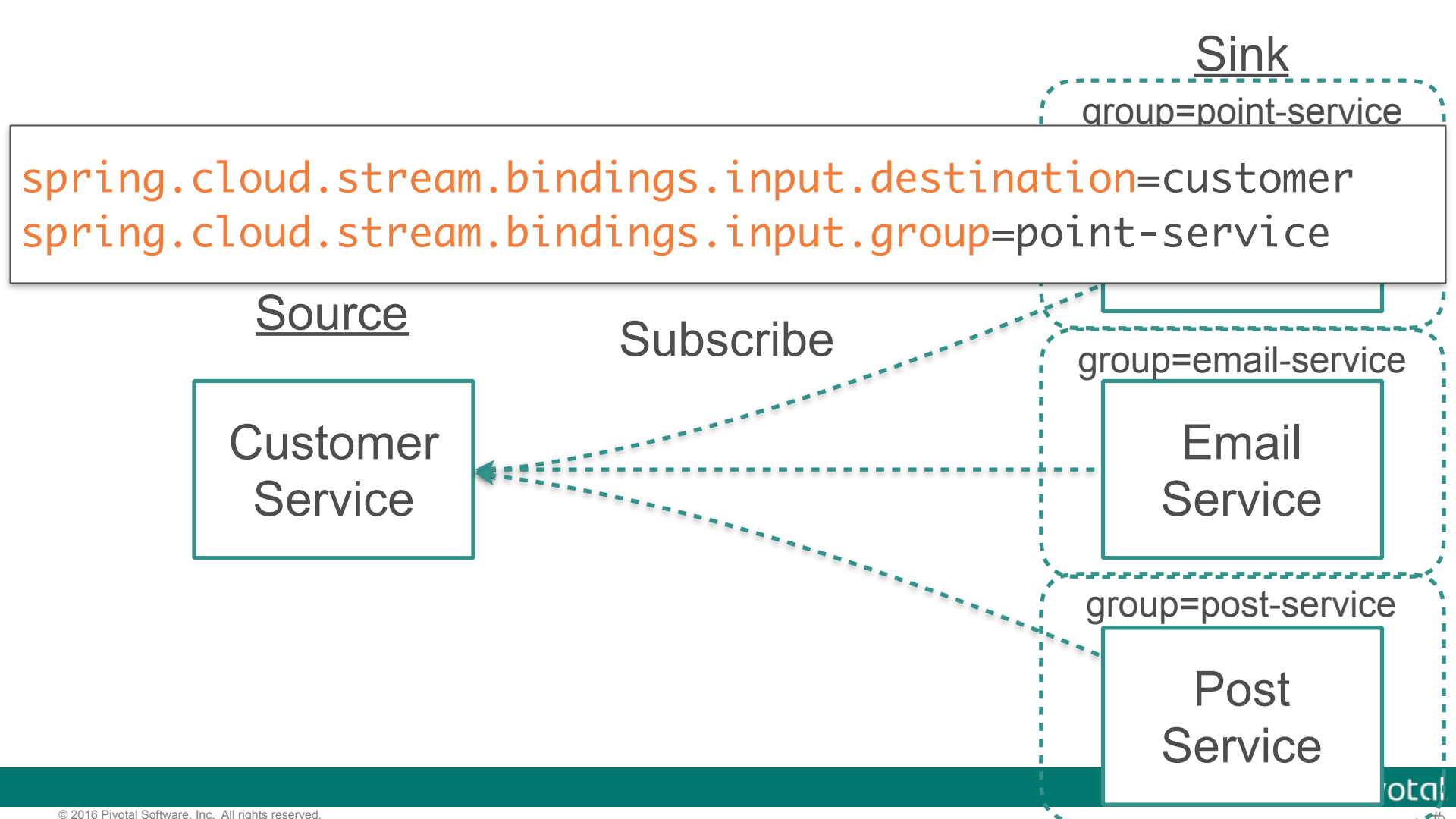
Source

```
spring.cloud.stream.bindings.output.destination=customer
spring.cloud.stream.bindings.output.contentType=application/json
```

Post Service







Sink

group=point-service

```
spring.cloud.stream.bindings.input.destination=customer
spring.cloud.stream.bindings.input.group=point-service
```

Source

Subscribe

group=email-service

```
spring.cloud.stream.bindings.input.destination=customer
spring.cloud.stream.bindings.input.group=email-service
```

group=post-service

Post Service

Sink

group=point-service

```
spring.cloud.stream.bindings.input.destination=customer
spring.cloud.stream.bindings.input.group=point-service
```

Source

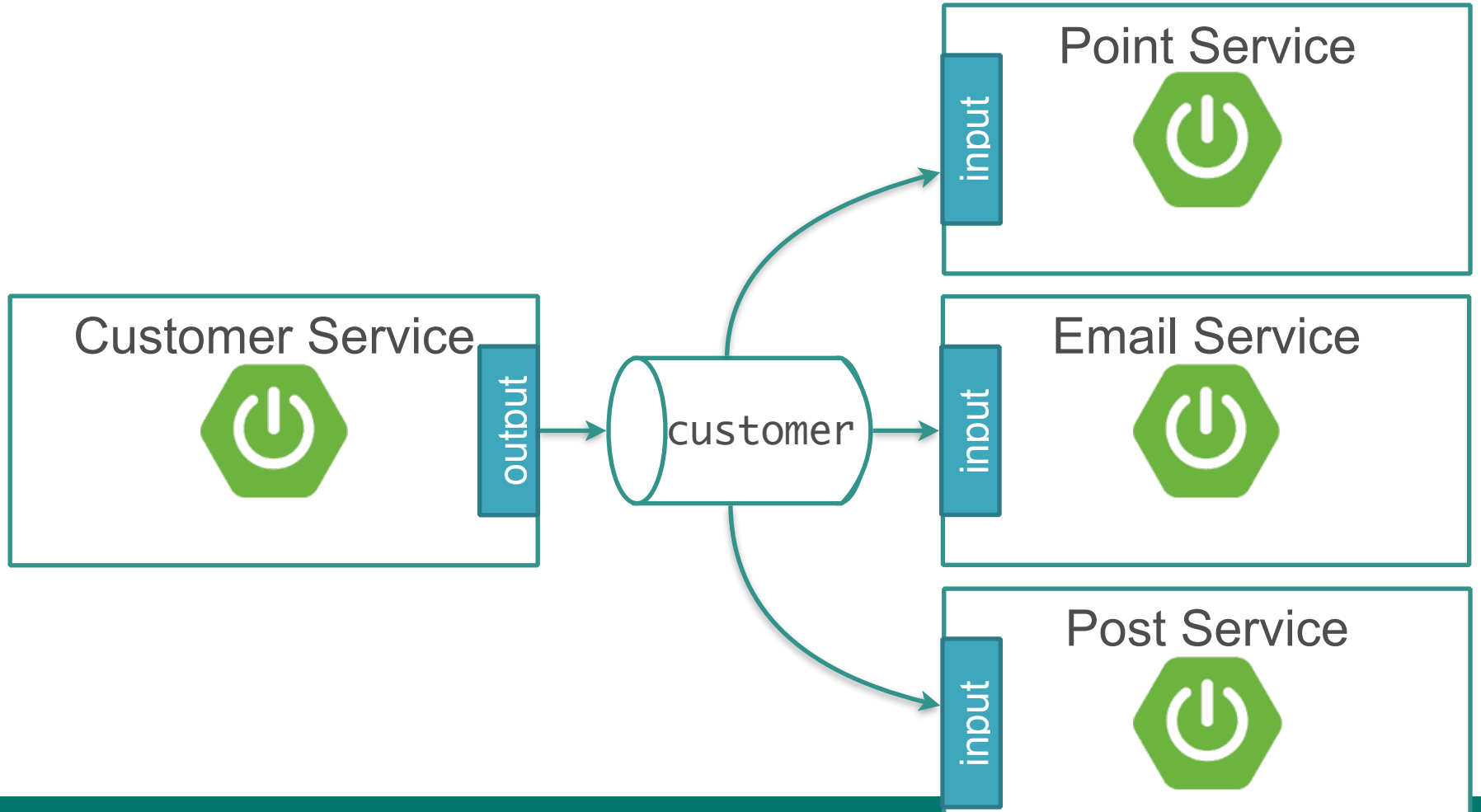
Subscribe

group=email-service

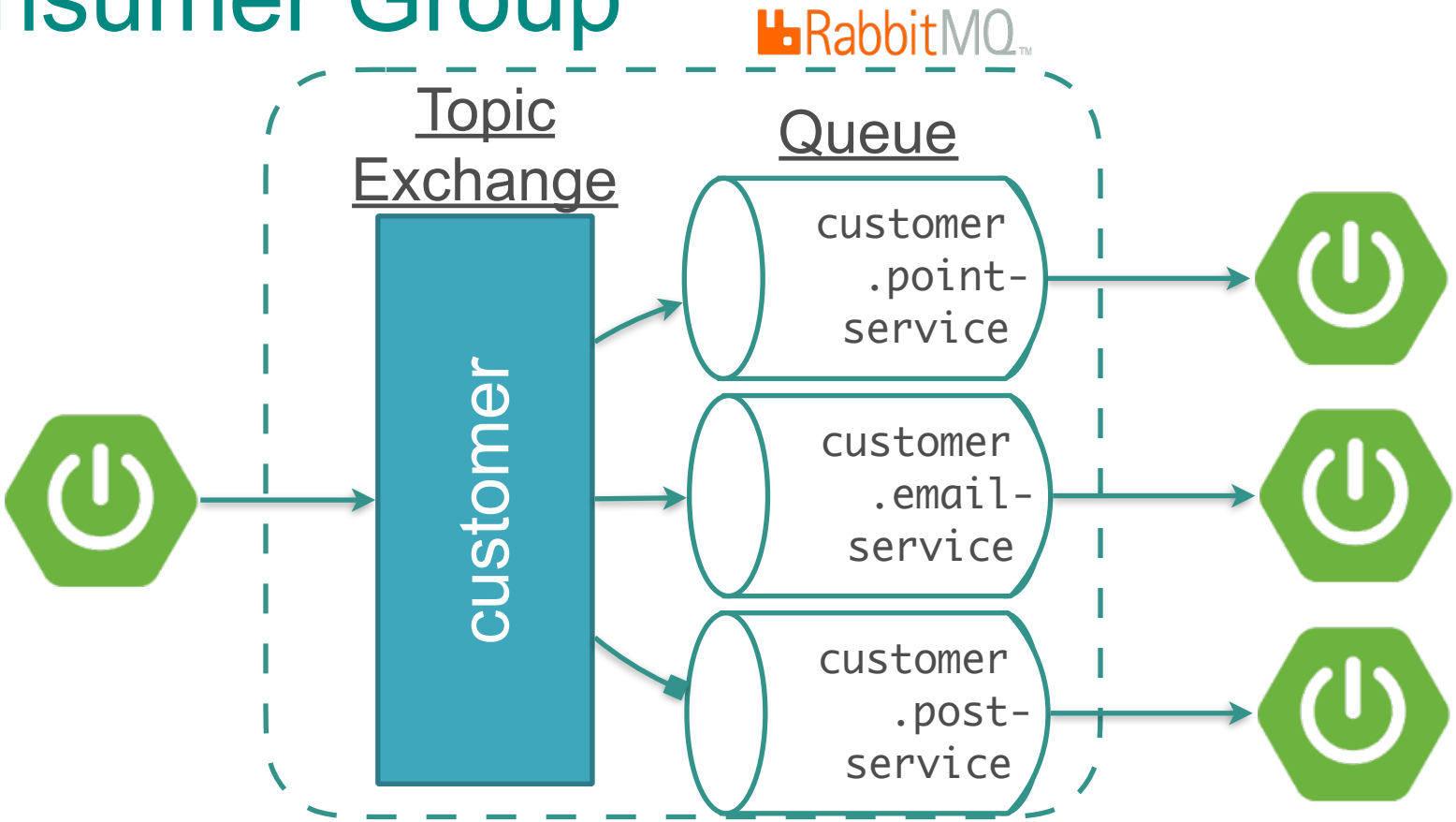
```
spring.cloud.stream.bindings.input.destination=customer
spring.cloud.stream.bindings.input.group=email-service
```

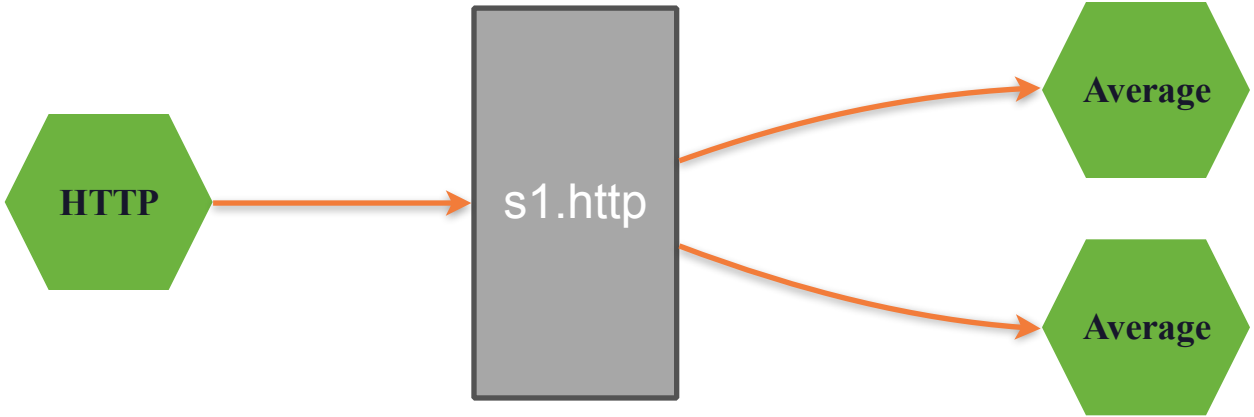
group=post-service

```
spring.cloud.stream.bindings.input.destination=customer
spring.cloud.stream.bindings.input.group=post-service
```

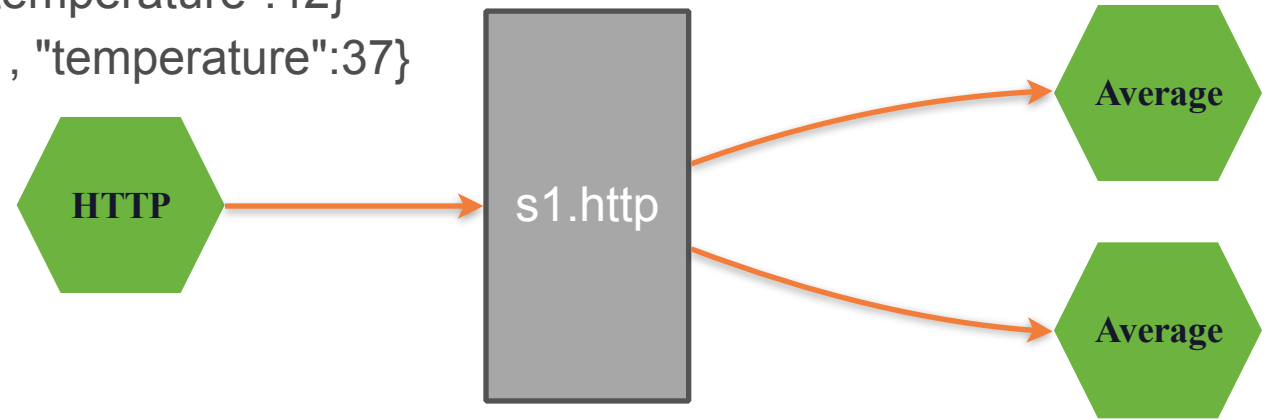


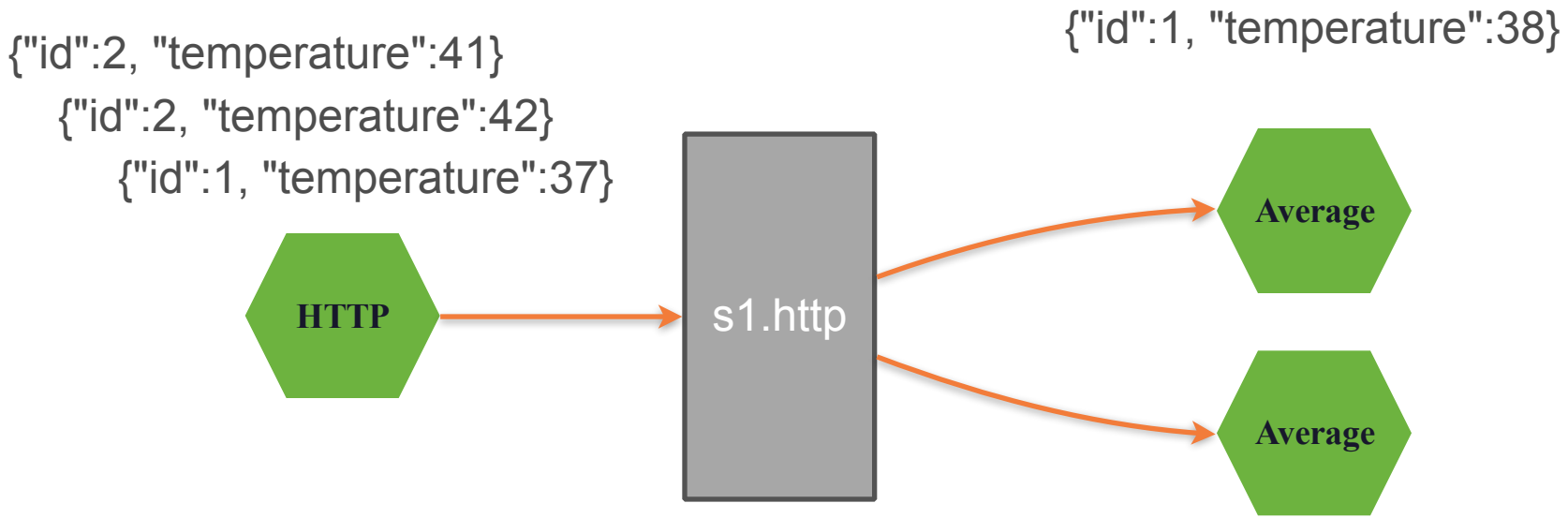
Consumer Group

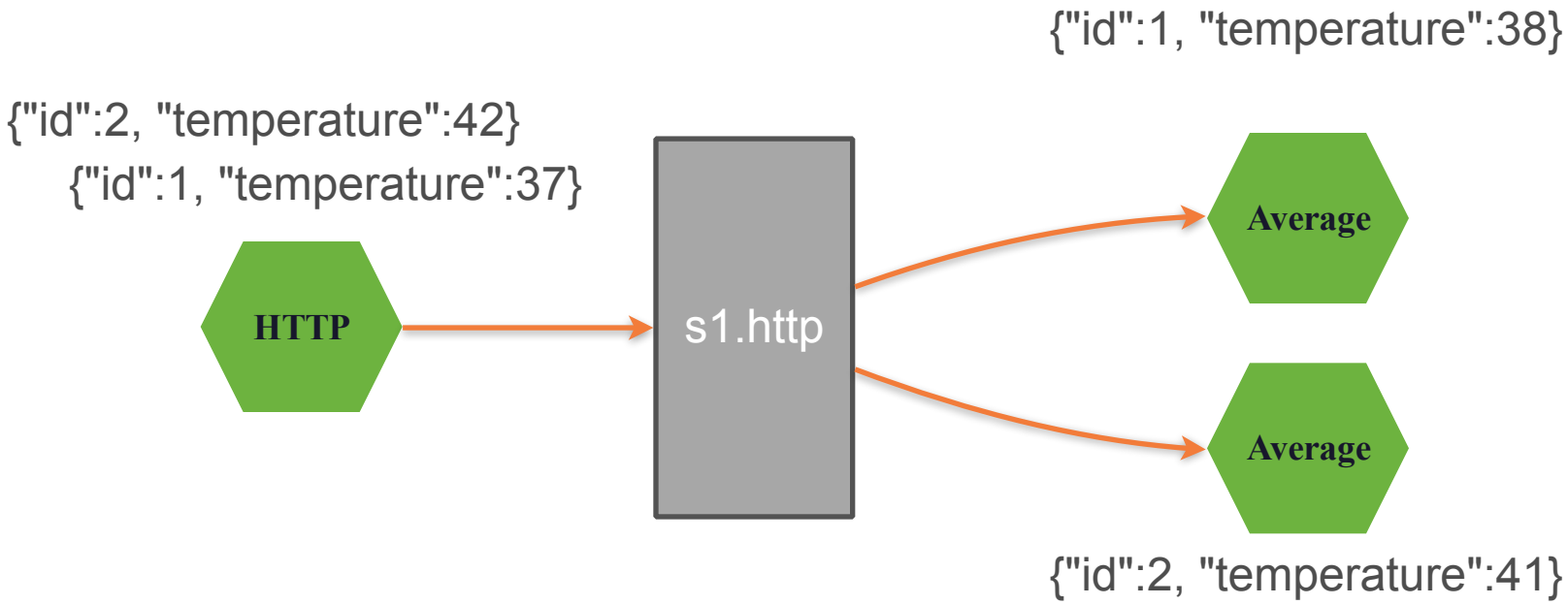


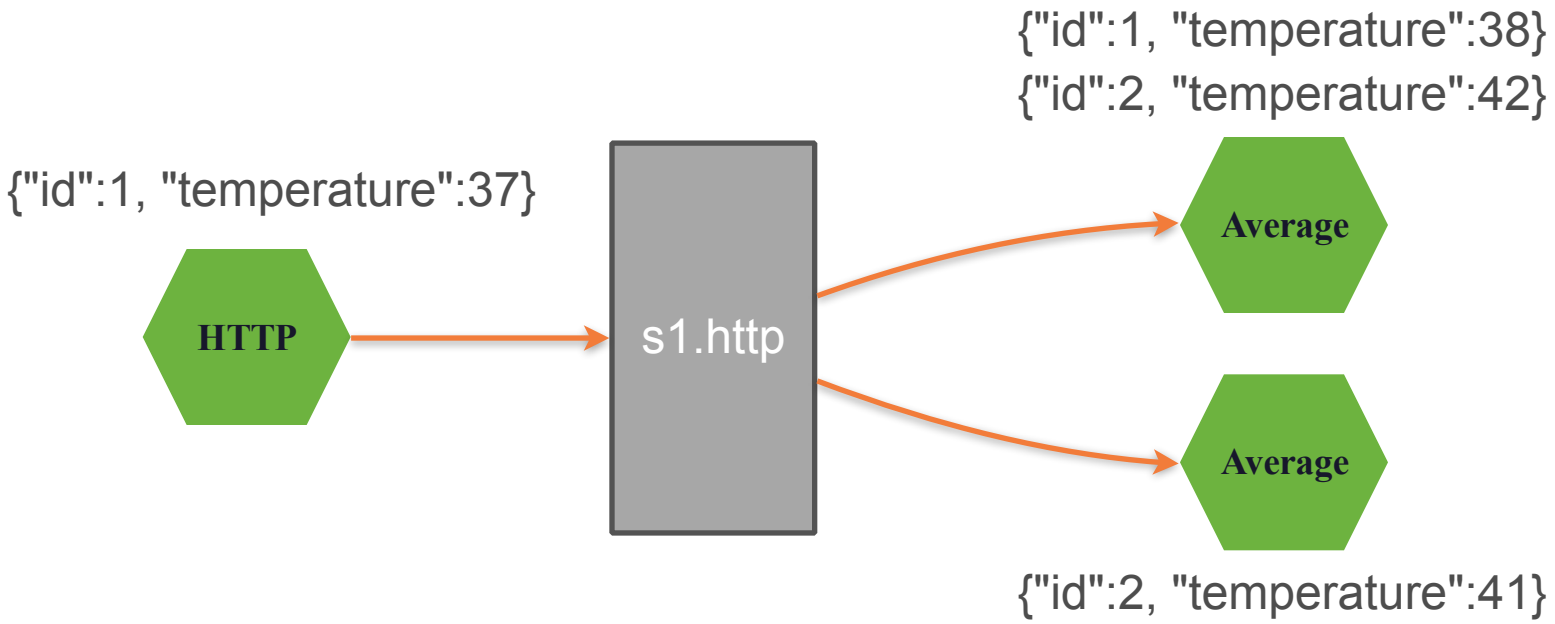


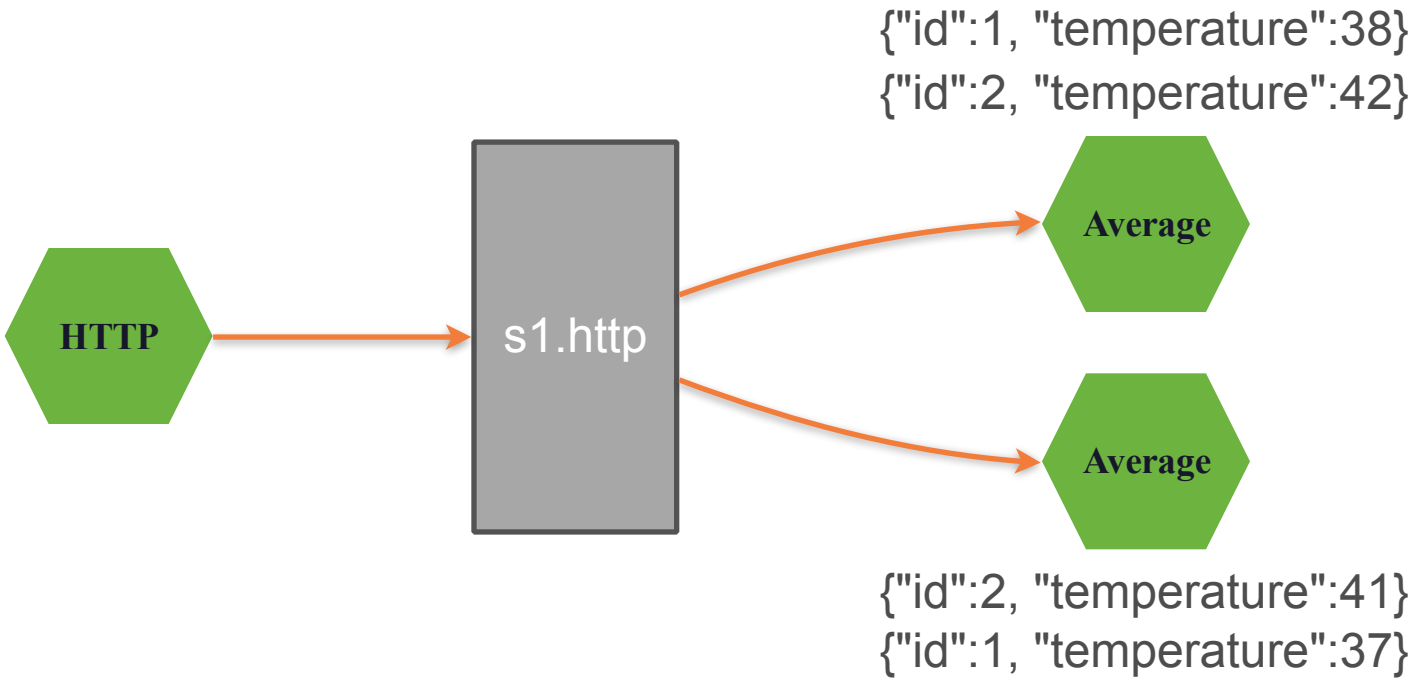
{"id":1, "temperature":38}
{"id":2, "temperature":41}
{"id":2, "temperature":42}
{"id":1, "temperature":37}

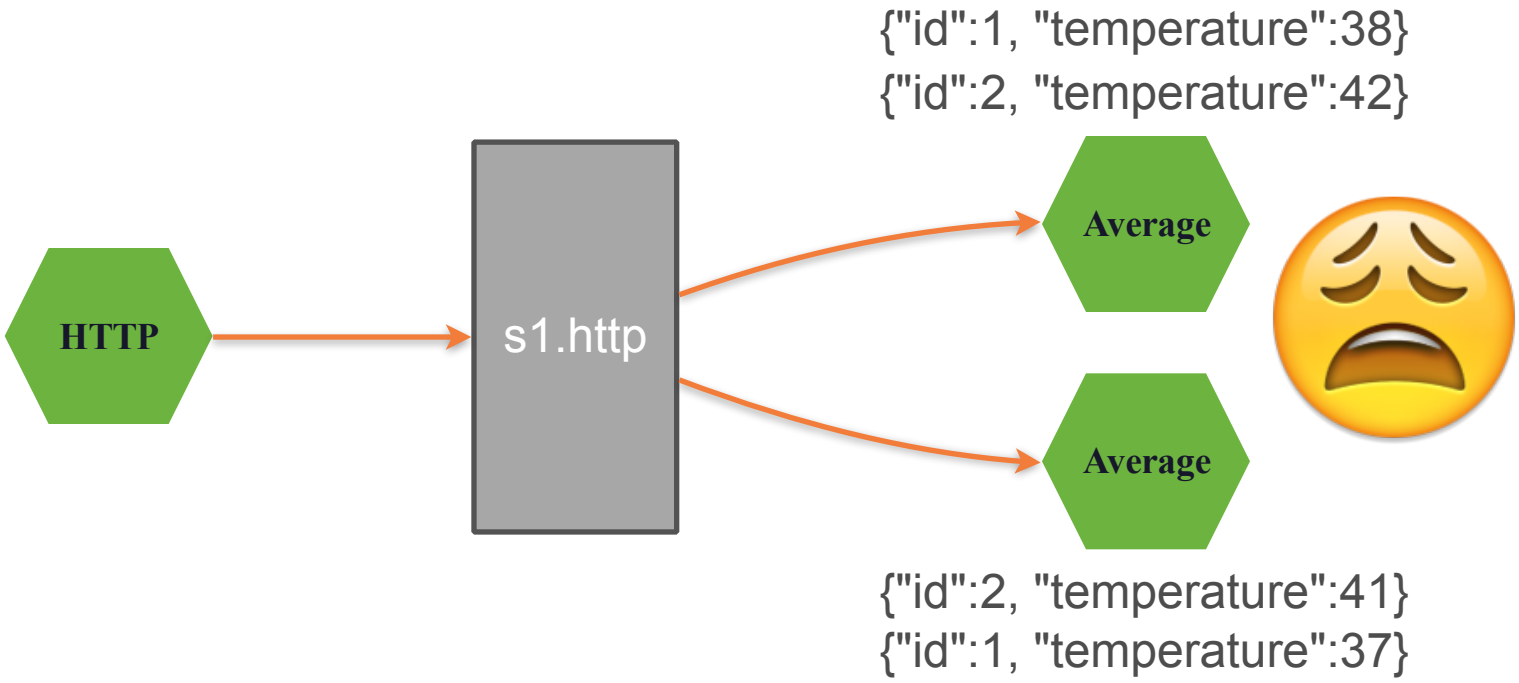




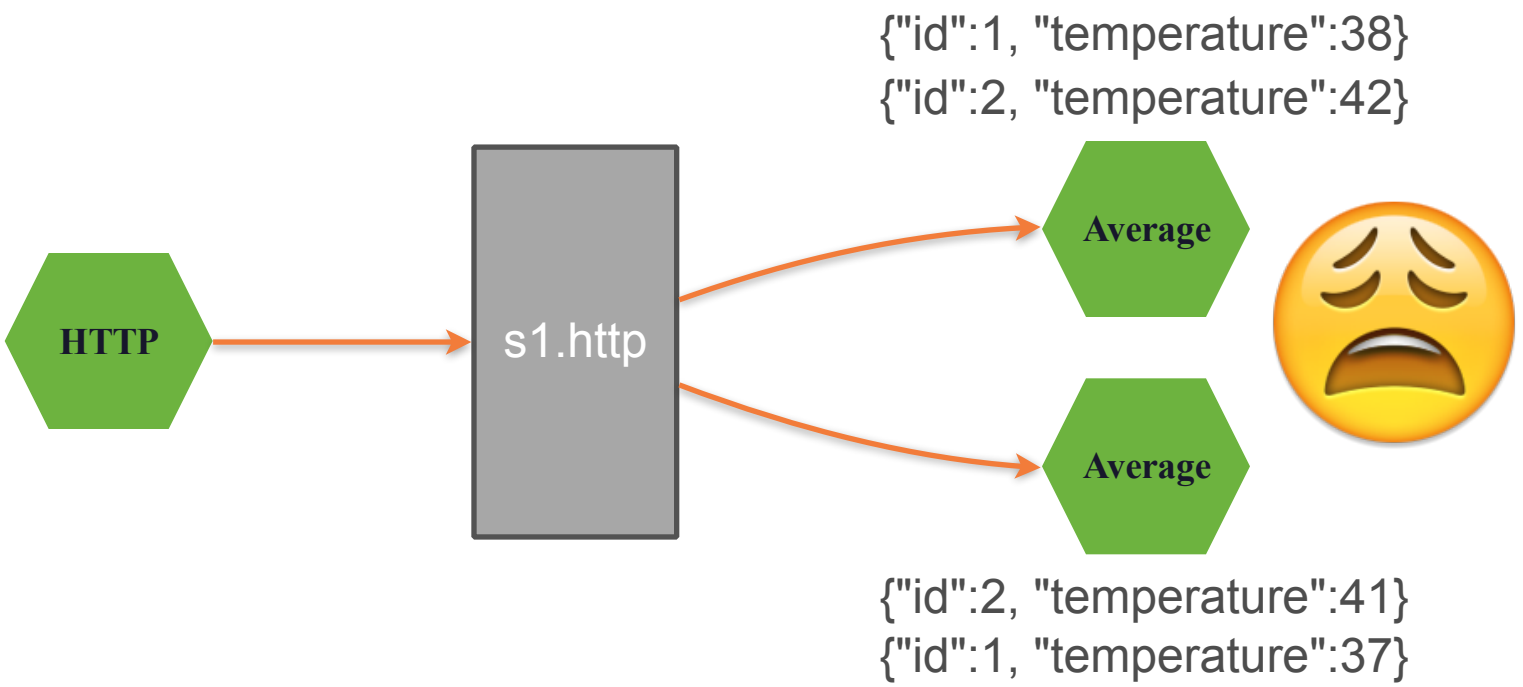






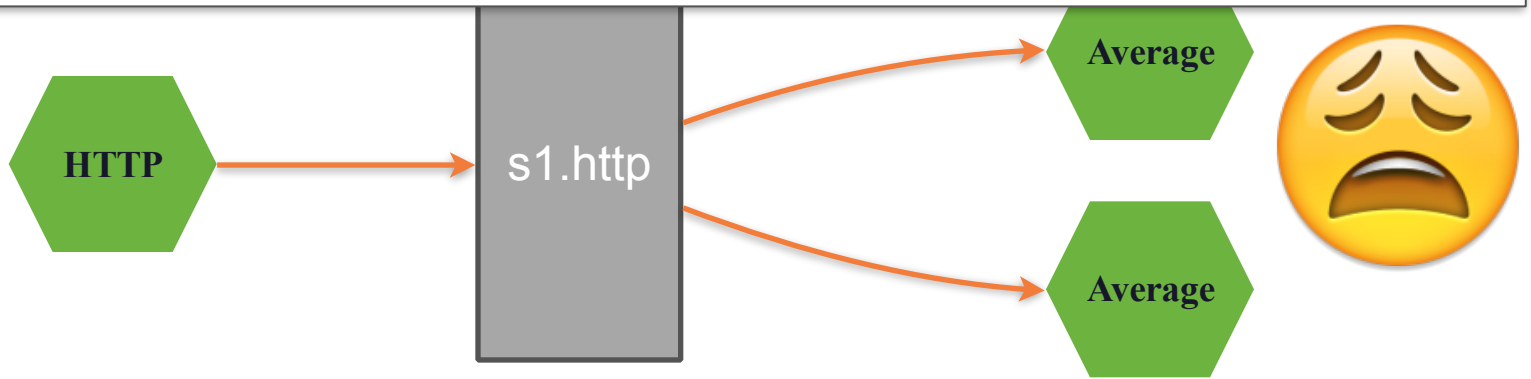


Partitioning Support(Stateful Stream)



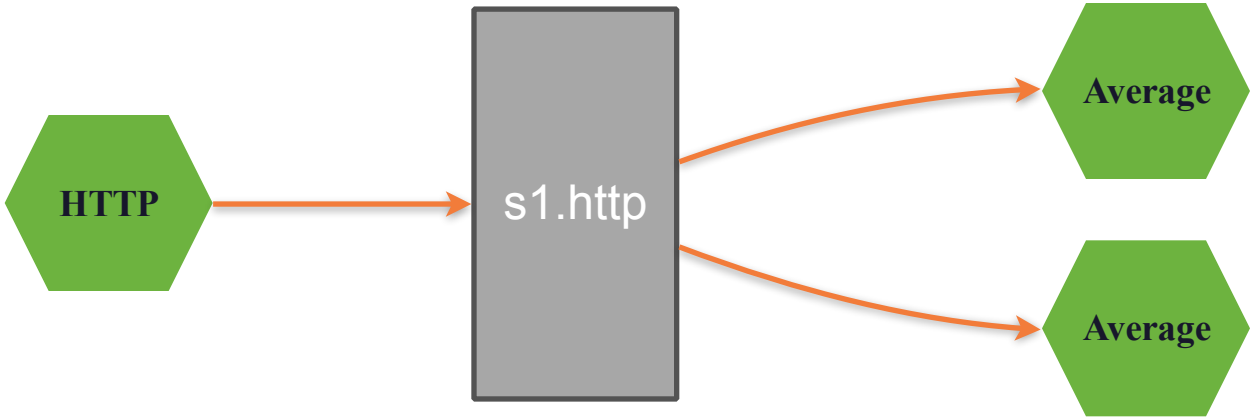
Partitioning Support(Stateful Stream)

```
spring.cloud.stream.bindings.<channelName>.producer.partitionKeyExpression=payload.id
```



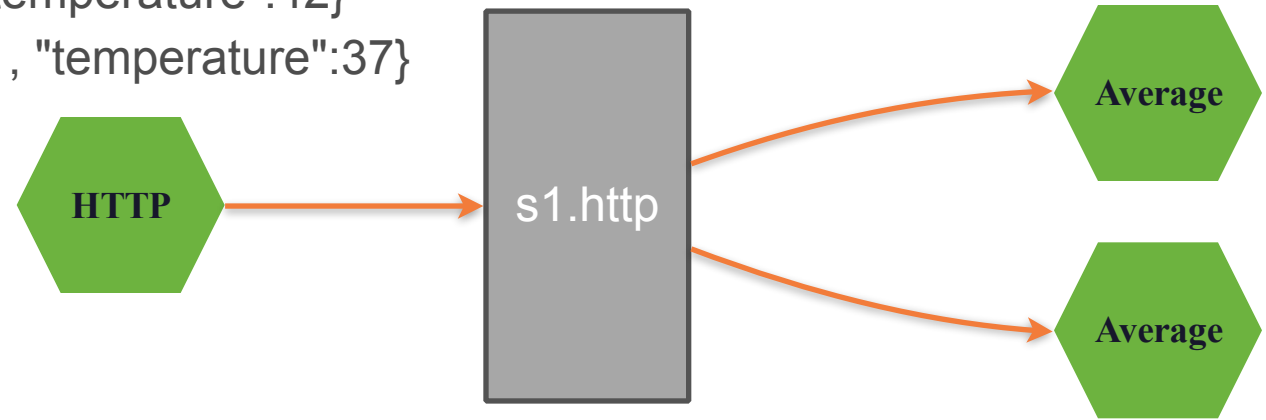
{"id":2, "temperature":41}
{"id":1, "temperature":37}

Partitioning Support(Stateful Stream)

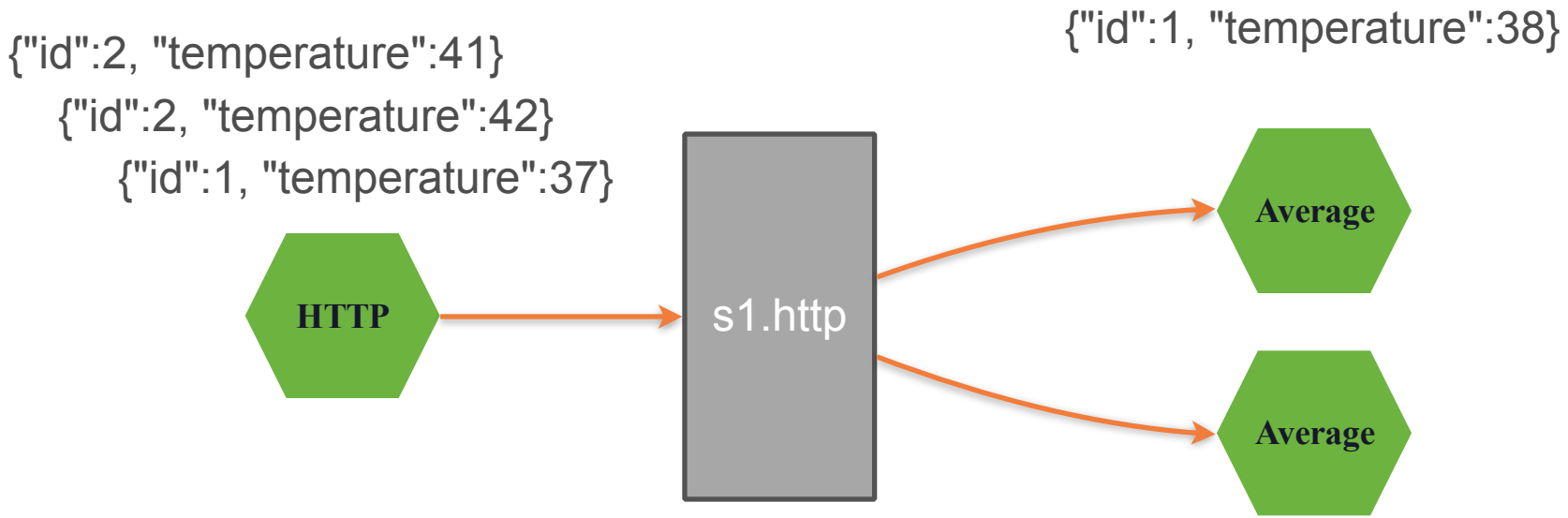


Partitioning Support(Stateful Stream)

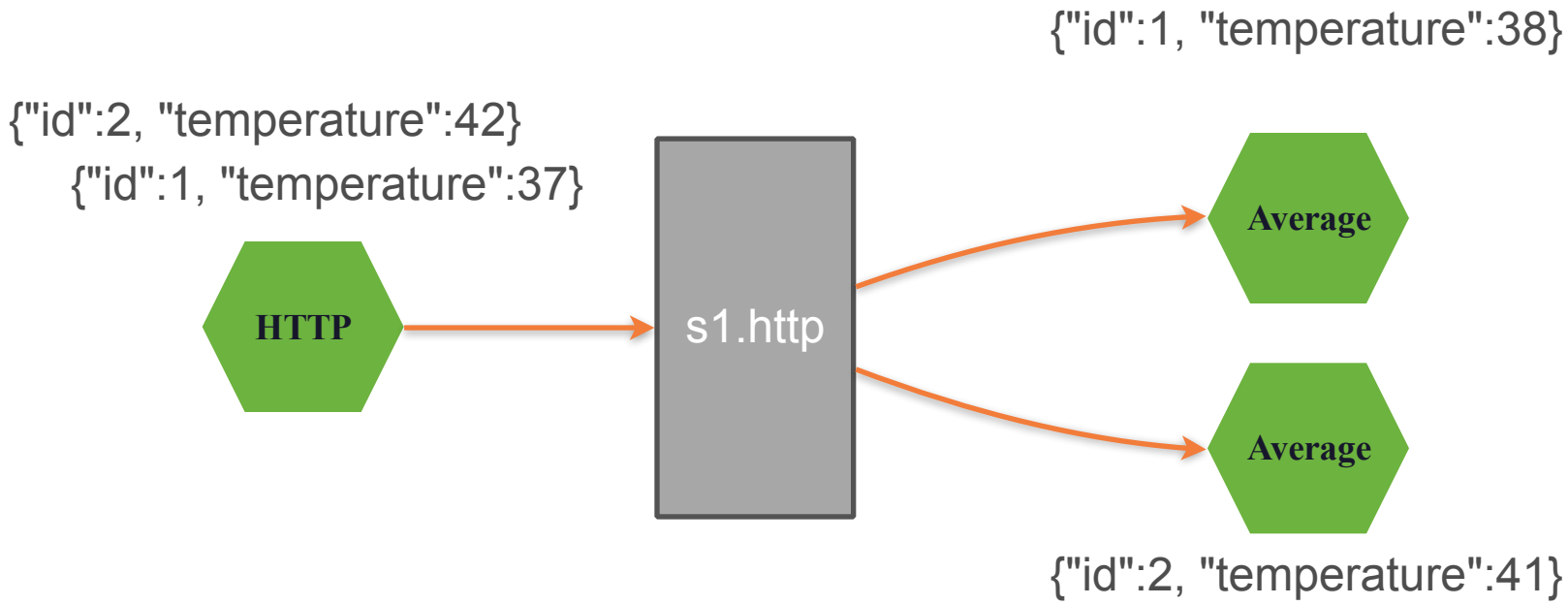
{"id":1, "temperature":38}
{"id":2, "temperature":41}
{"id":2, "temperature":42}
{"id":1, "temperature":37}



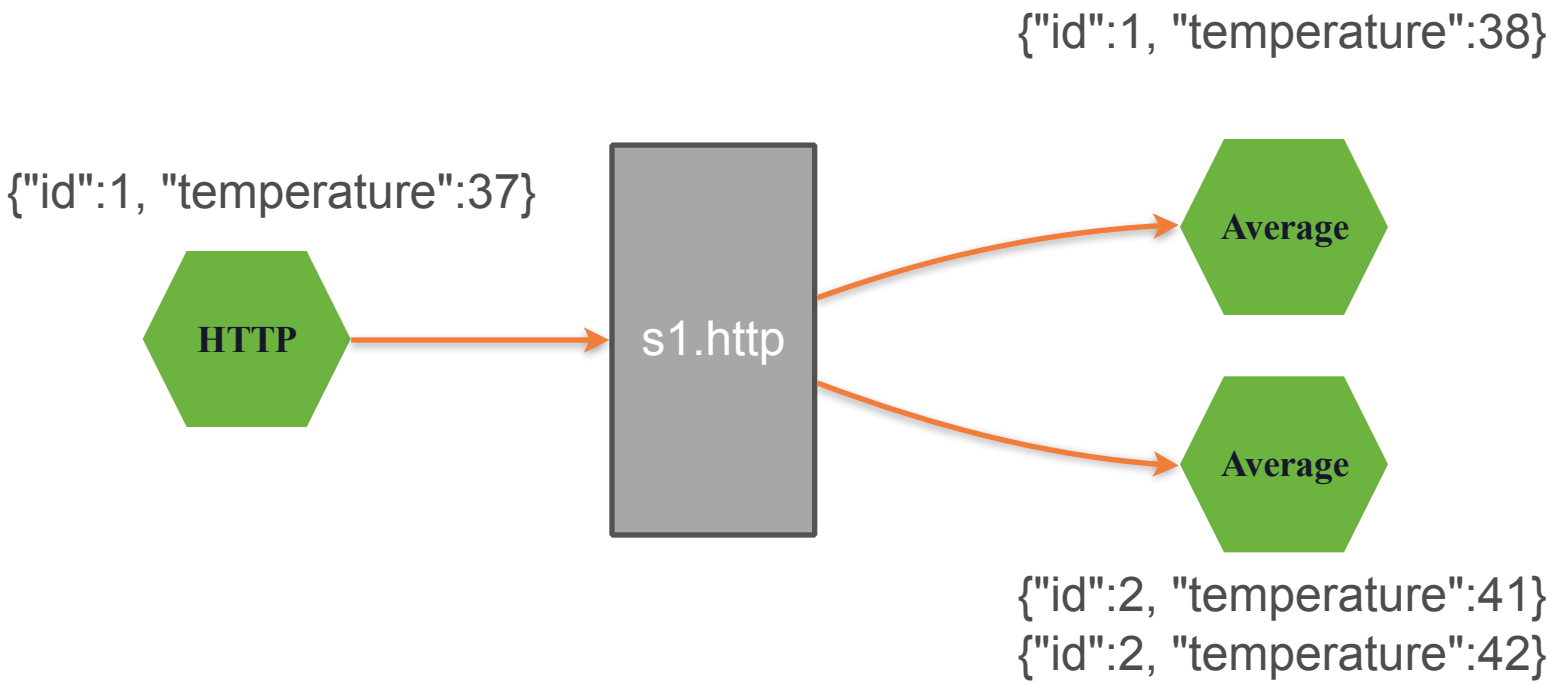
Partitioning Support(Stateful Stream)



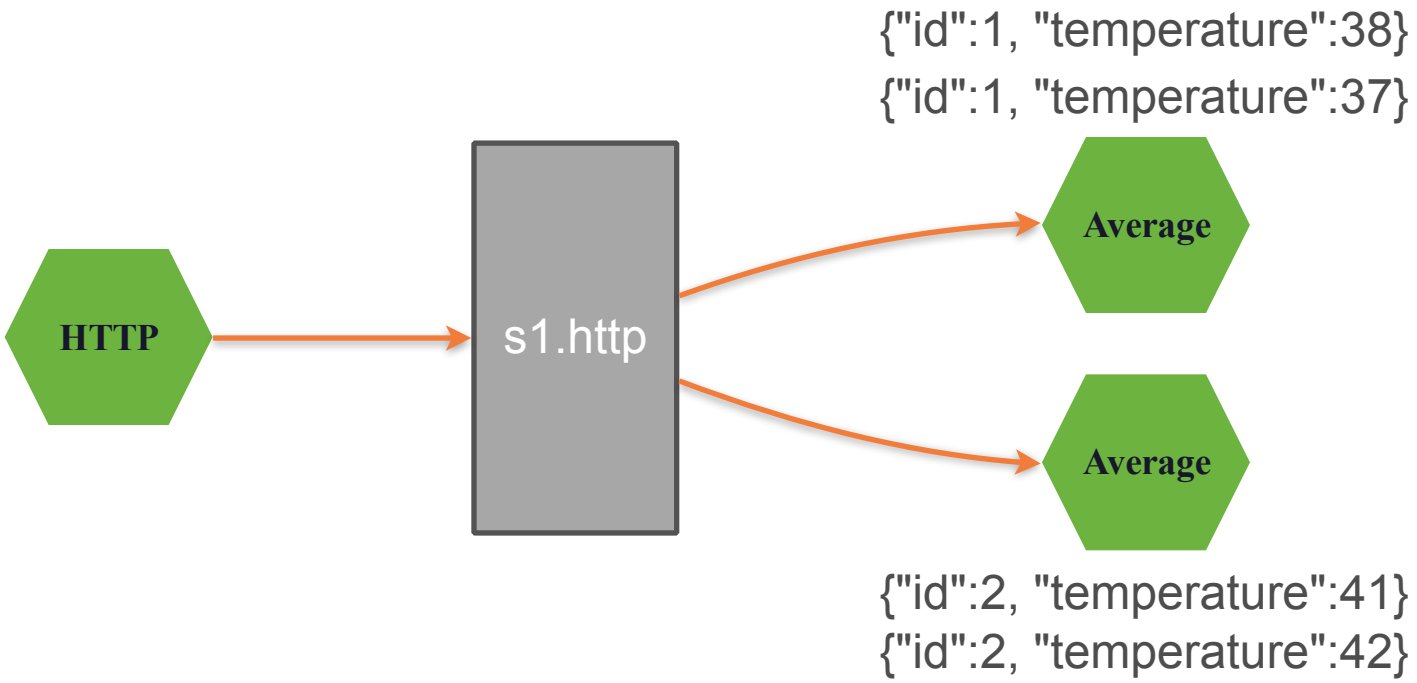
Partitioning Support(Stateful Stream)



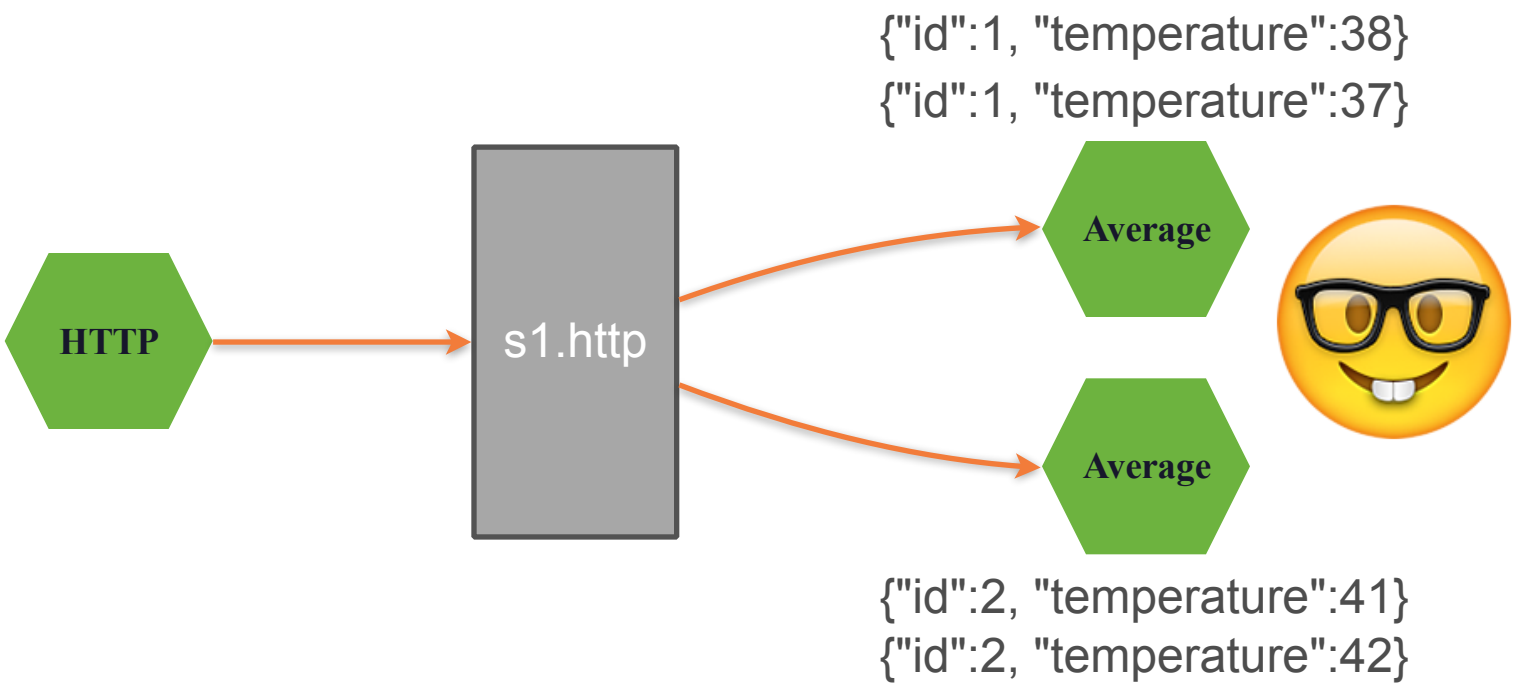
Partitioning Support(Stateful Stream)



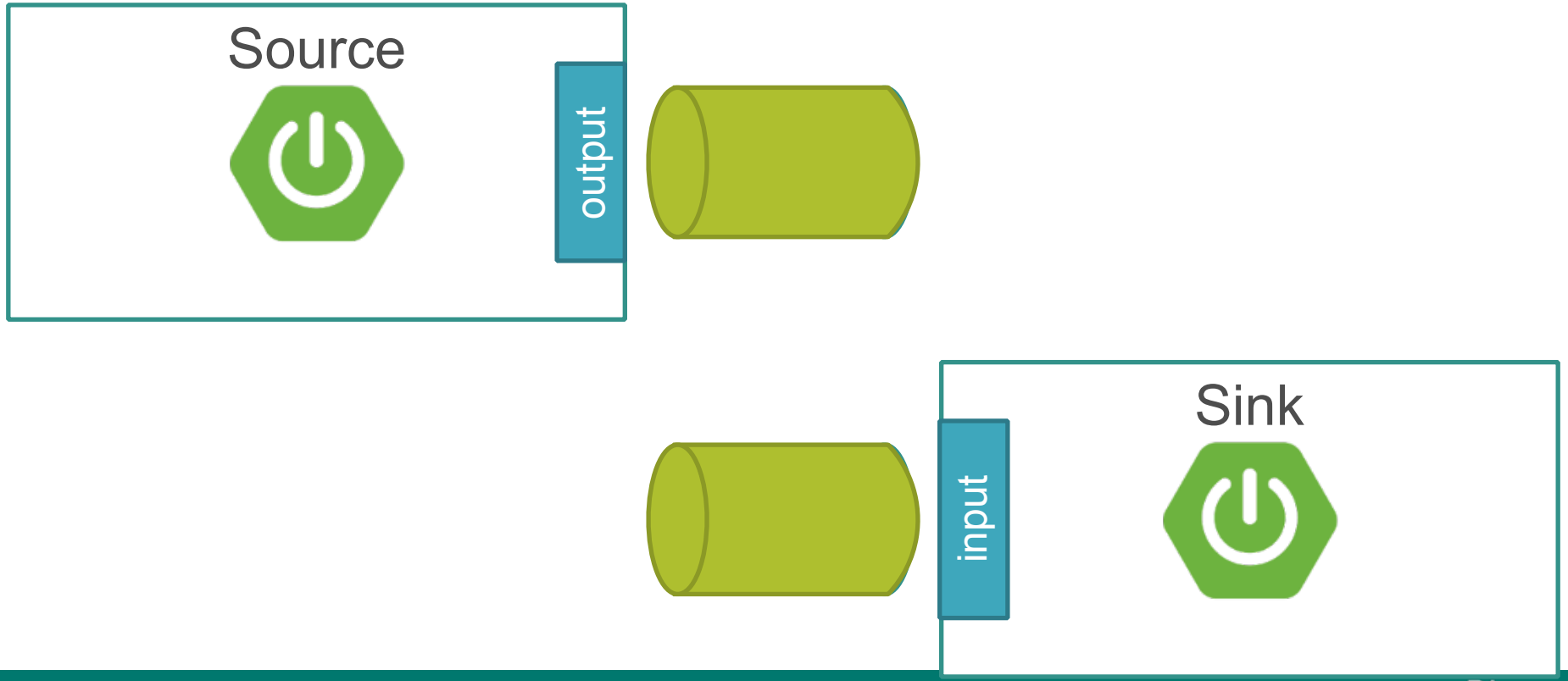
Partitioning Support(Stateful Stream)



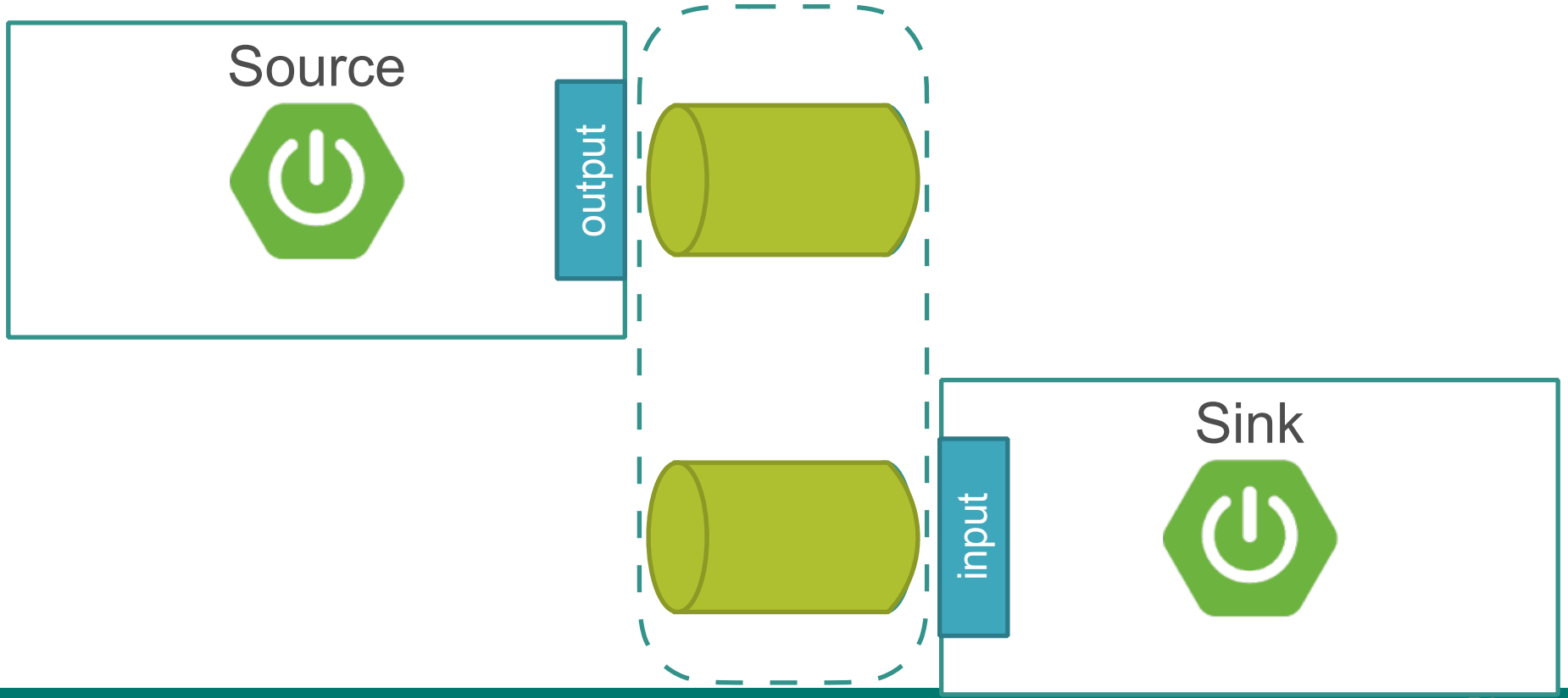
Partitioning Support(Stateful Stream)



Test Support



Test Support TestSupportBinder



Test Support

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-stream-test-support</artifactId>
  <scope>test</scope>
</dependency>
```

Unit Test (Sink)

```
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment = WebEnvironment.NONE)
public class DemoSinkAppTest {
  @Autowired Sink sink;
  @Rule public OutputCapture capture = new OutputCapture();
  @Test public void testReceive() {
    sink.input()
        .send(MessageBuilder.withPayload("foo").build());
    assertThat(capture.toString())
        .isEqualsTo("Received foo");
  }
}
```

Unit Test (Source)

```
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment = WebEnvironment.NONE)
public class DemoSourceAppTest {
  @Autowired DemoSourceApp app;
  @Autowired MessageCollector collector;
  @Autowired Source source;
  @Test public void testSend() {
    app.send("foo");
    Message<String> message = collector
                    .forChannel(source.output()).poll();
    assertThat(message.getPayload()).isEqualsTo("foo");
  }}
```

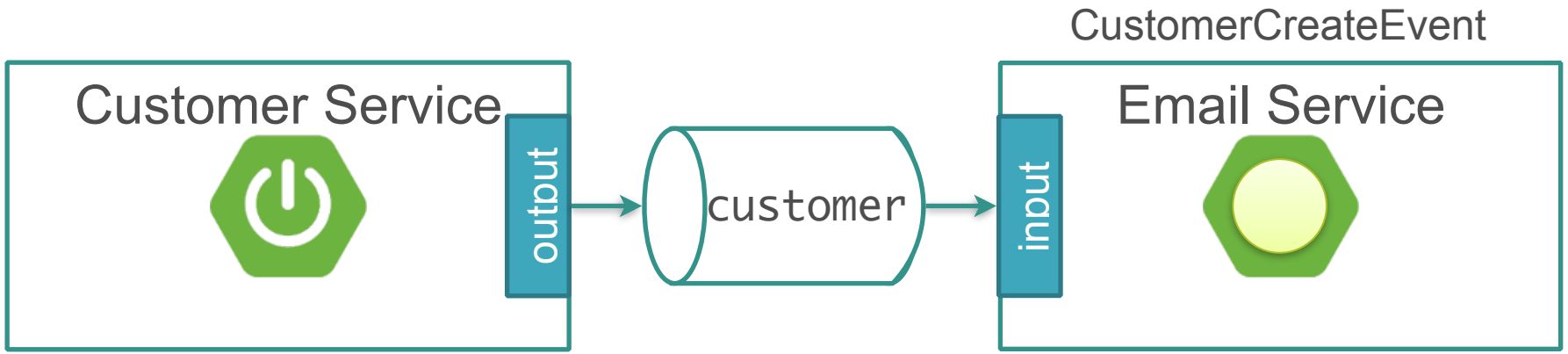
Advanced Topics

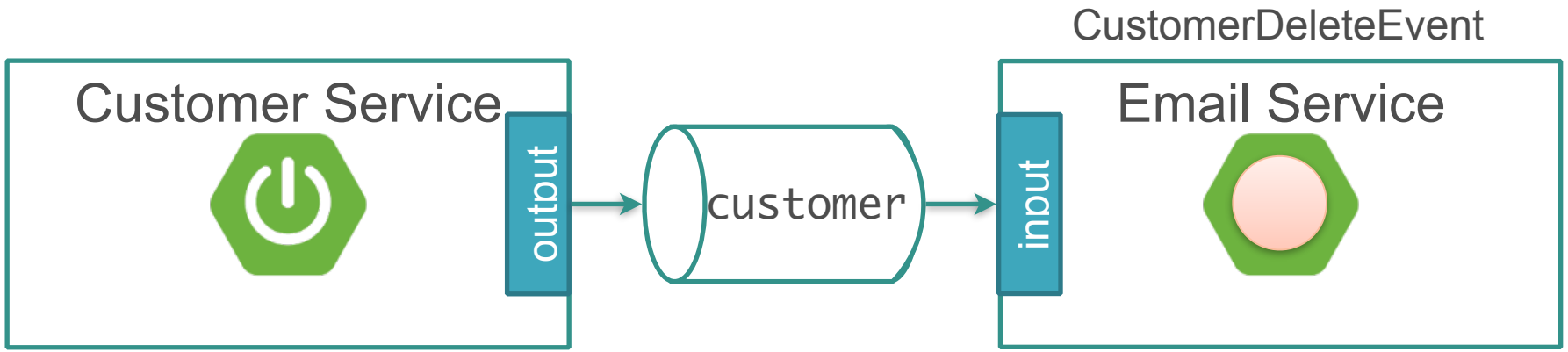
Pivotal

Advanced Topics

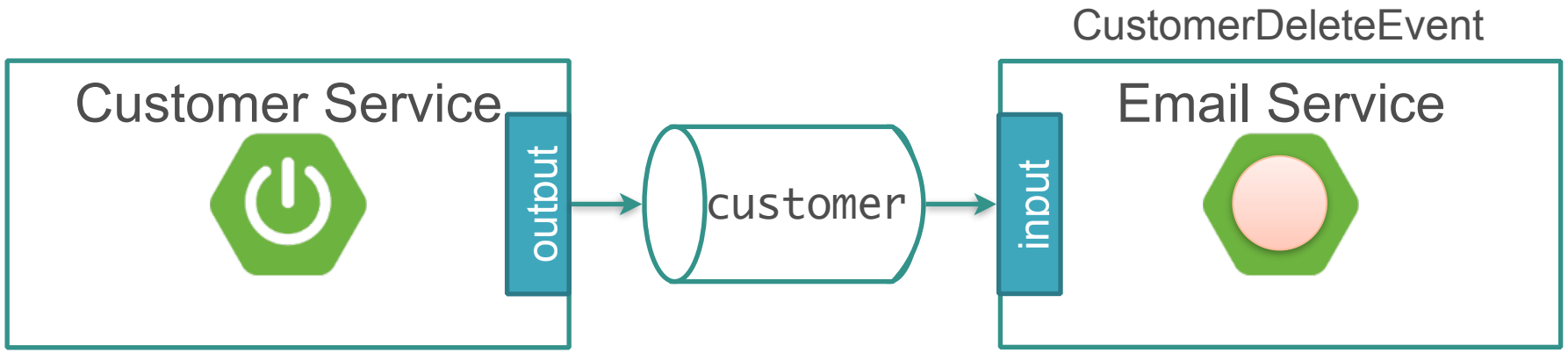
- Multi Binding
- Distributed Tracing
- Error Handling
- Consumer Driven Contract

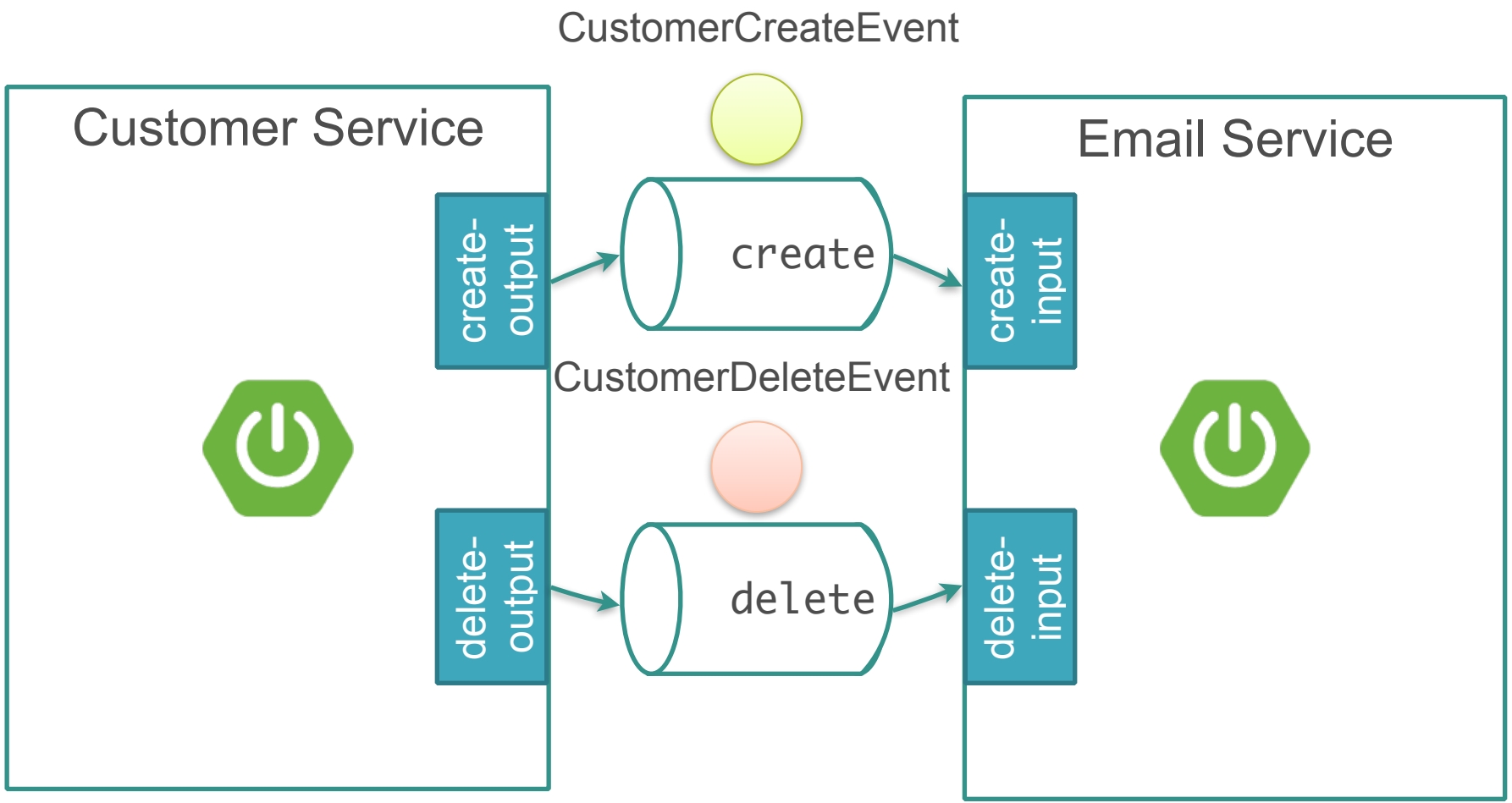
Multi Bindings





ClassCastException!!





Multi Bindings

```
public interface CustomerEventSource {
  String CREATE_OUTPUT = "create-output";
  String DELETE_OUTPUT = "delete-output";
  @Output(CustomerEventSource.CREATE_OUTPUT)
  MessageChannel createOutput();
  @Output(CustomerEventSource.DELETE_OUTPUT)
  MessageChannel deleteOutput();
}
```

```
@SpringBootApplication
@EnableBinding(CustomerEventSource.class)
public class CustomerServiceApplication { /* ... */ }
```

```
@Component
public class CustomerService {
  @Autowired CustomerEventSource source;
  public void create(...) {
    source.createOutput().send(...);
  }
  public void delete() {
    source.deleteOutput().send(...);
  }
}
```

```
@SpringBootApplication
@EnableBinding(CustomerEventSource.class)
public class CustomerServiceApplication { /* ... */ }
```

```
@Component
  }
  public void delete() {
    source.deleteOutput().send(...);
  }
}
```

```
spring.cloud.stream.bindings.create-output.destination
=create
spring.cloud.stream.bindings.delete-output.destination
=delete
```

Multi Bindings

```
public interface CustomerEventSink {
  String CREATE_INPUT = "create-input";
  String DELETE_INPUT = "delete-input";
  @Input(CustomerEventSink.CREATE_INPUT)
  SubscribableChannel createInput();
  @Input(CustomerEventSink.DELETE_INPUT)
  SubscribableChannel deleteInput();
}
```

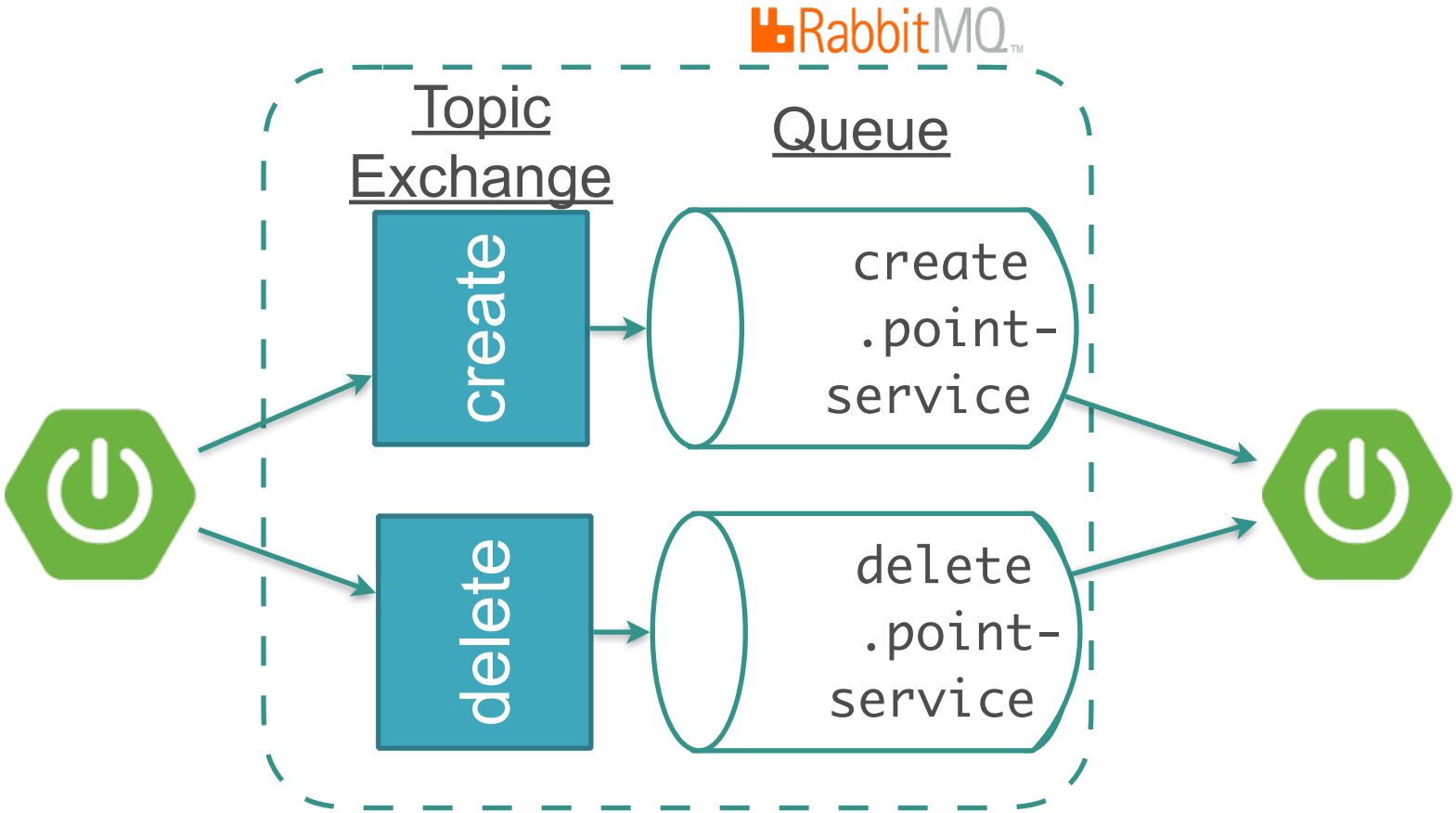
```
@SpringBootApplication
@EnableBinding(CustomerEventSink.class)
public class PointServiceApplication { /* ... */ }
```

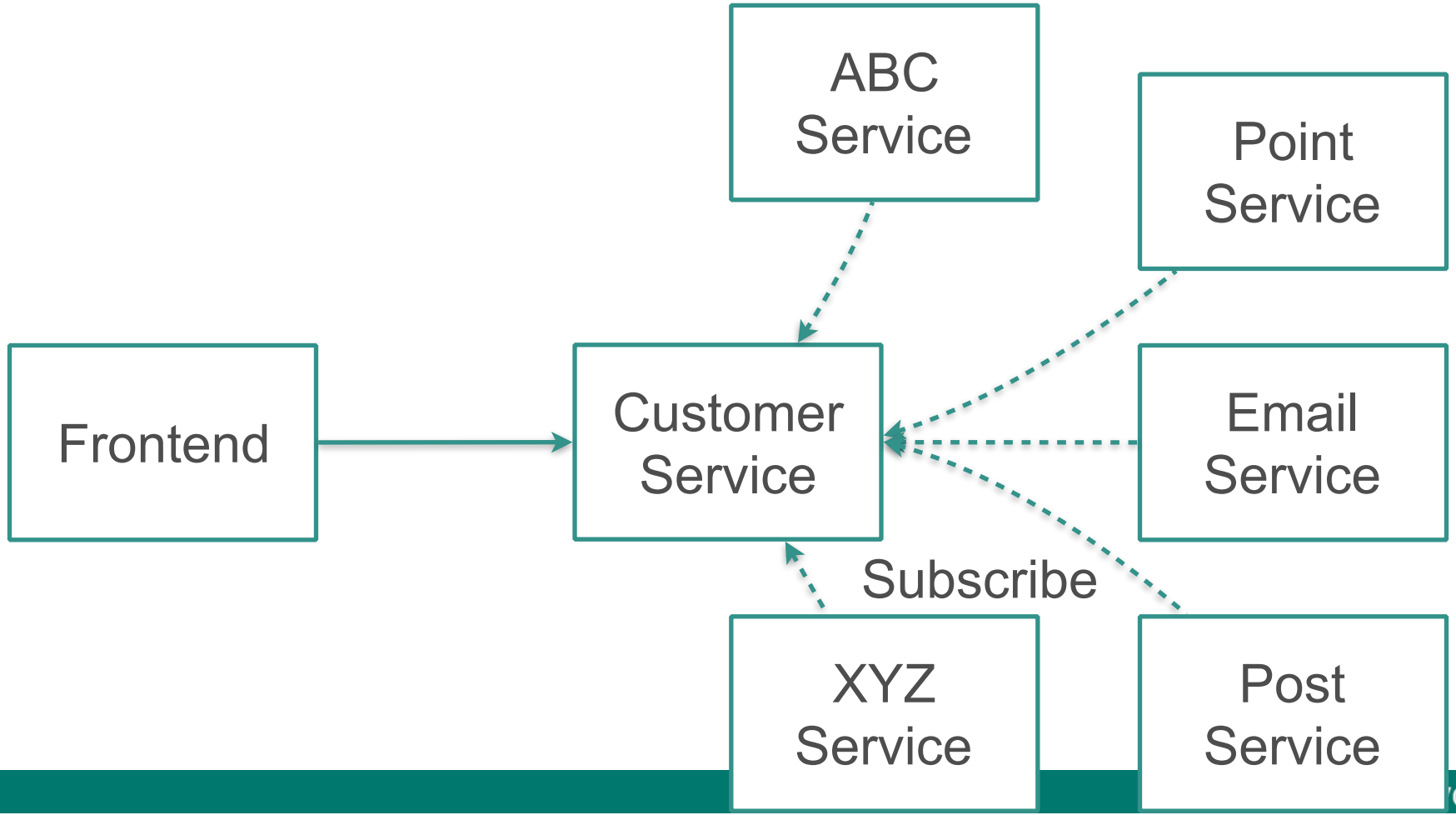
```
@Component
public class PointService {
  @StreamListener(CustomerEventSink.CREATE_INPUT)
  public void handleCreate(CustomerCreateEvent event) {
  }

  @StreamListener(CustomerEventSink.DELETE_INPUT)
  public void handleDelete(CustomerDeleteEvent event) {
  }
}
```

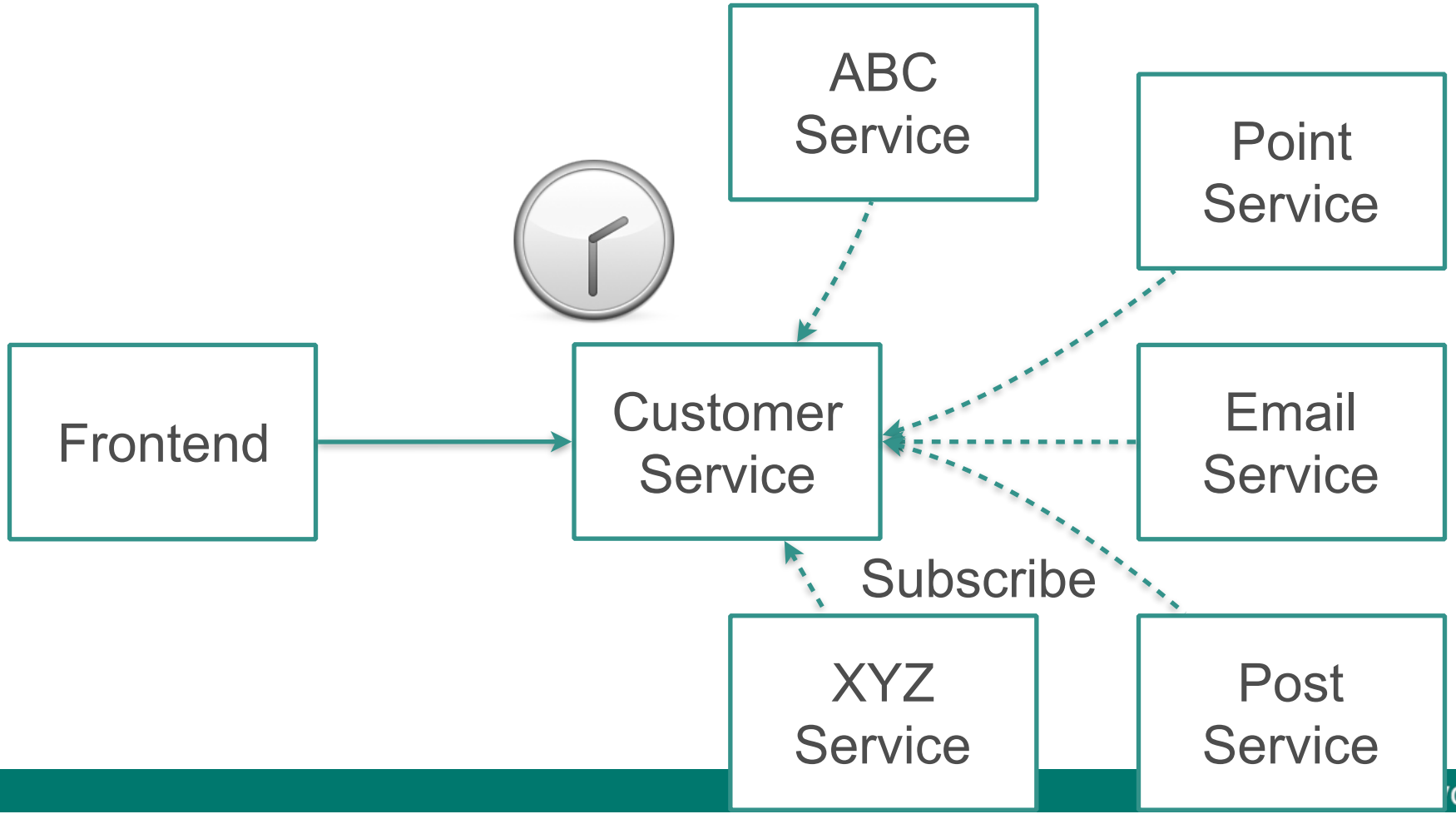
```
@SpringBootApplication
@EnableBinding(CustomerEventSink.class)
```

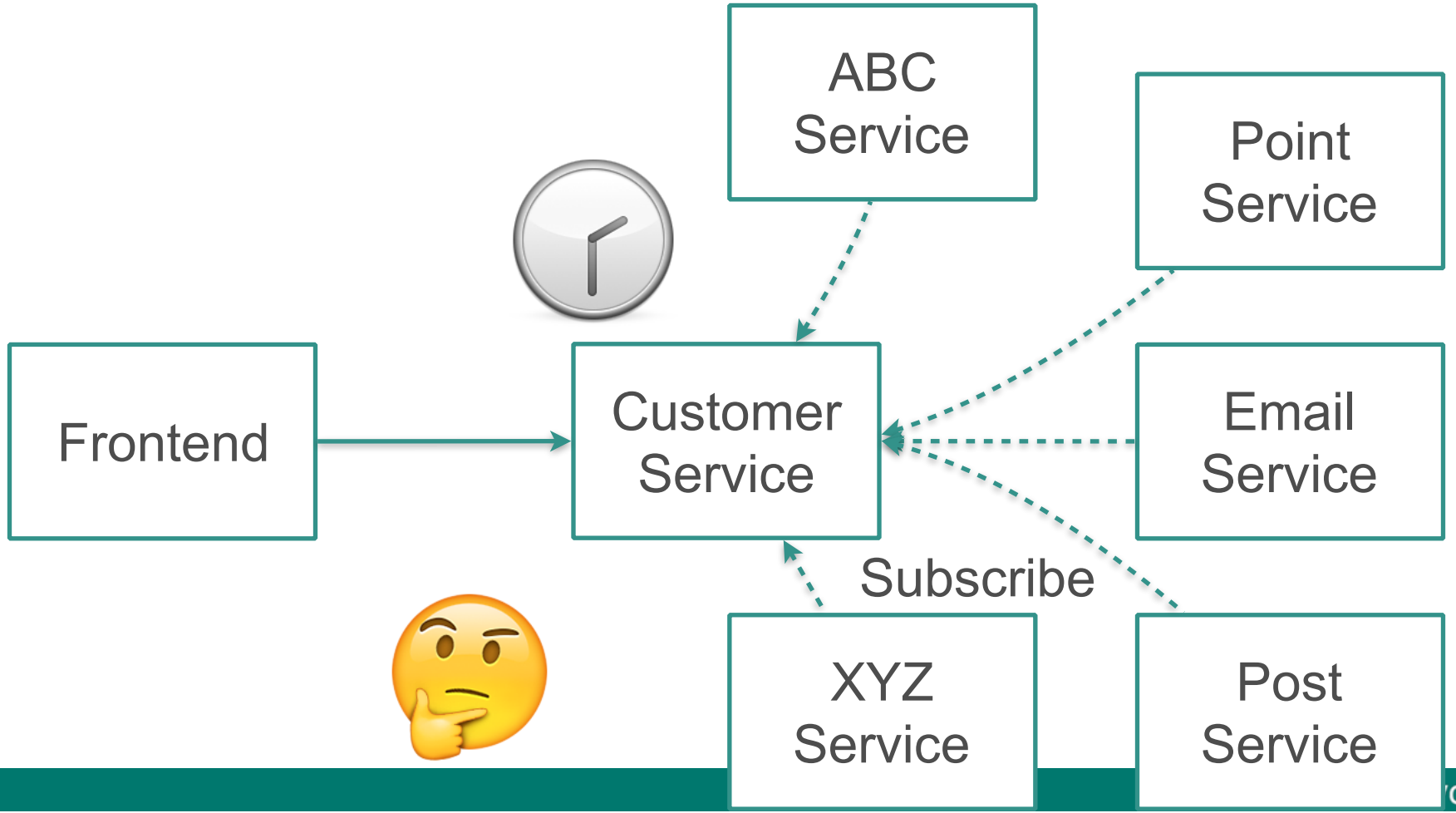
```
spring.cloud.stream.bindings.create-input.destination
=create
spring.cloud.stream.bindings.create-input.group
=point-service
spring.cloud.stream.bindings.delete-intput.destination
=delete
spring.cloud.stream.bindings.delete-input.group
=point-service
```

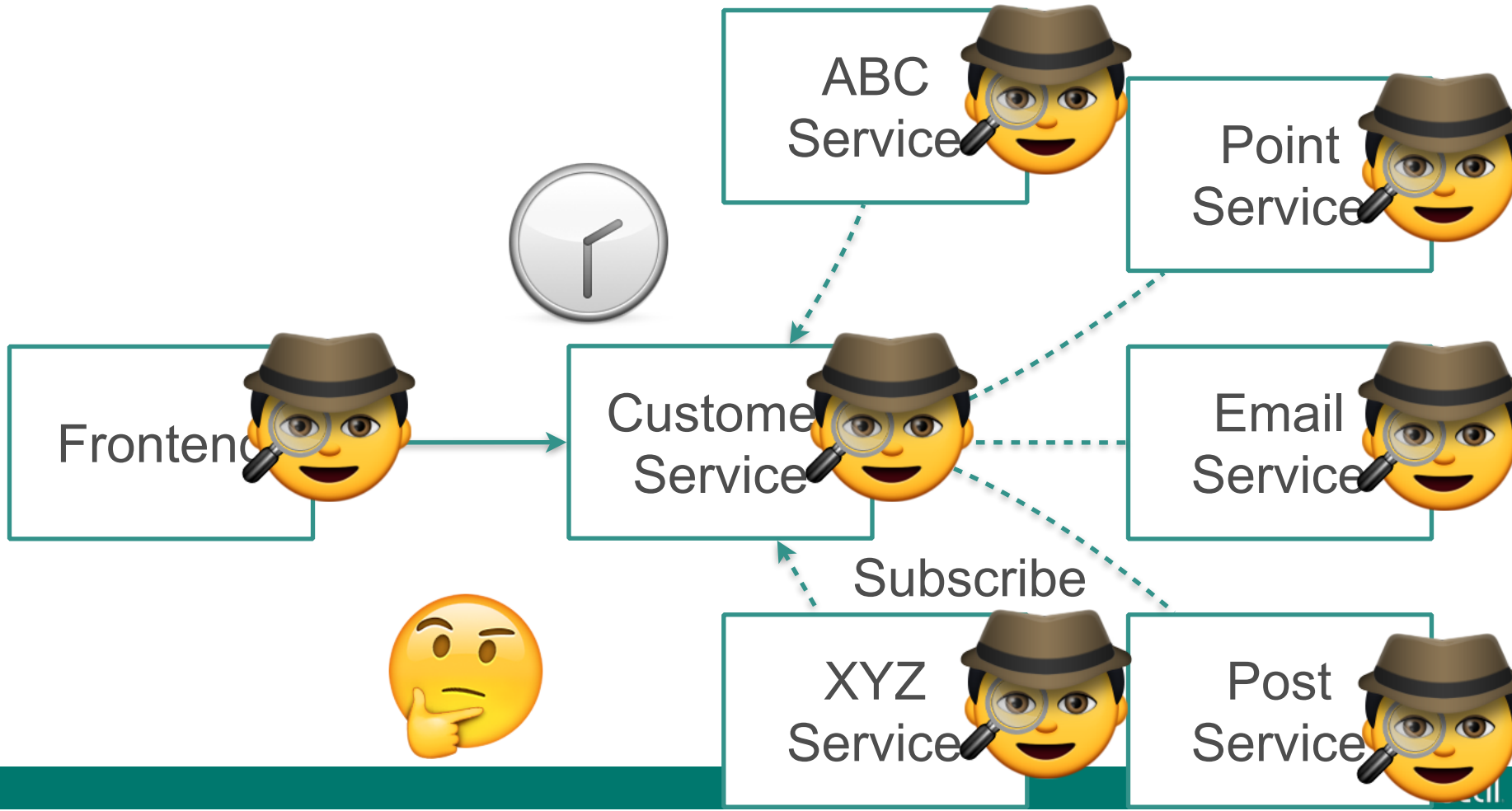
```
    }
}
```











ABC Service

Point Service

Frontend

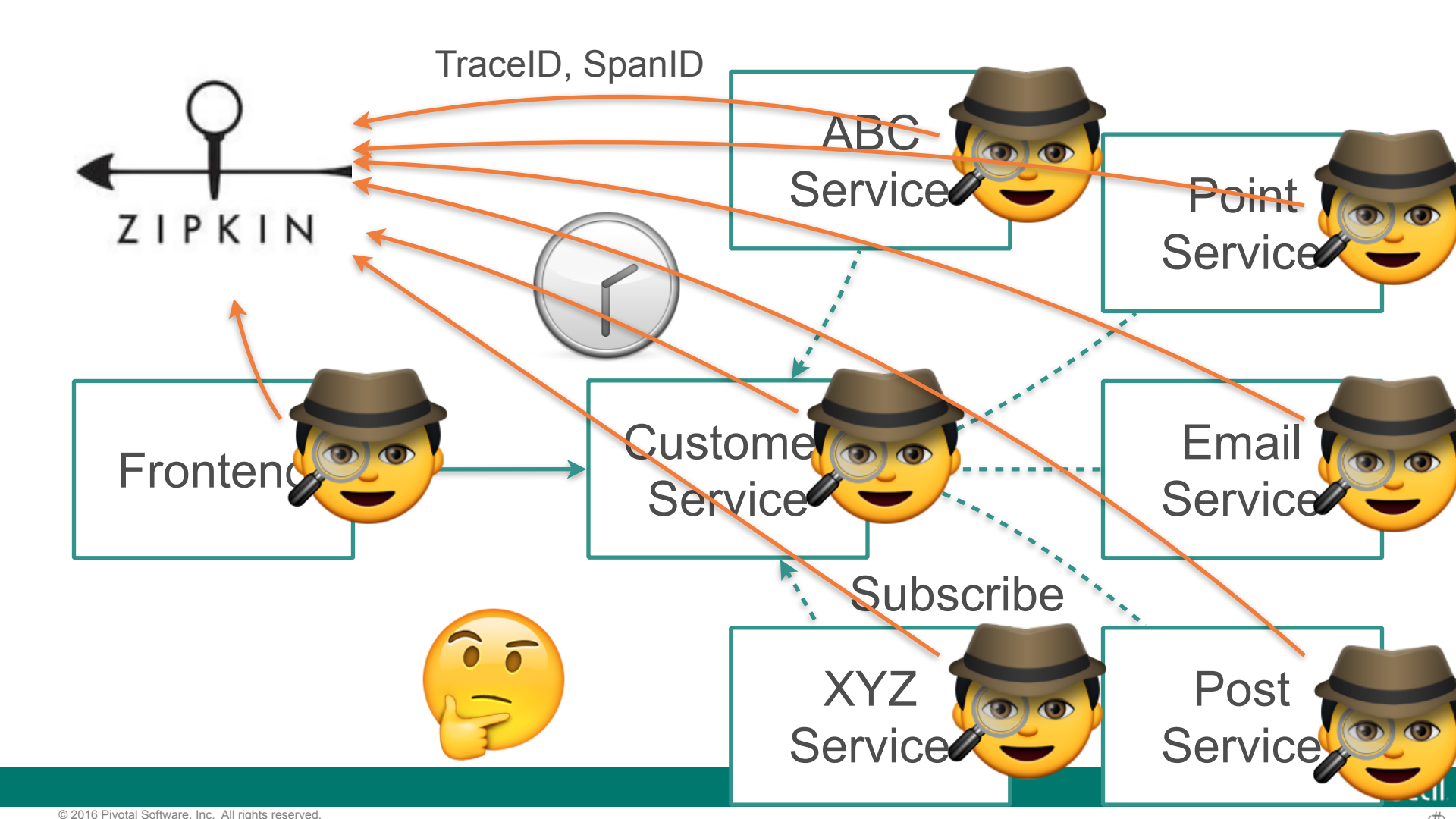
Custome Service

Email Service

Subscribe

XYZ Service

Post Service



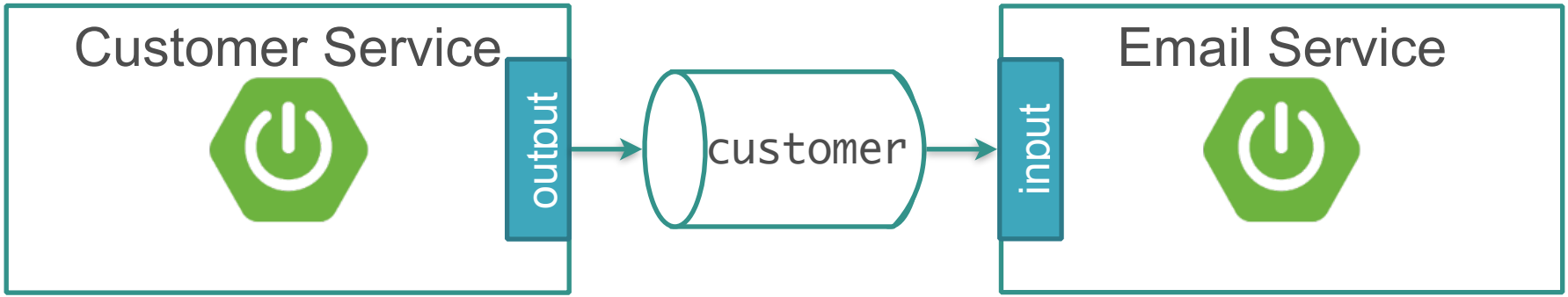
Spring Cloud Sleuth

- **Distributed tracing solution** for Spring Cloud
- Interactions with external systems should be instrumented **automatically**
- Capture data simply in logs, or by sending it to **Zipkin** via **RestTemplate / Spring Cloud Stream / ...**

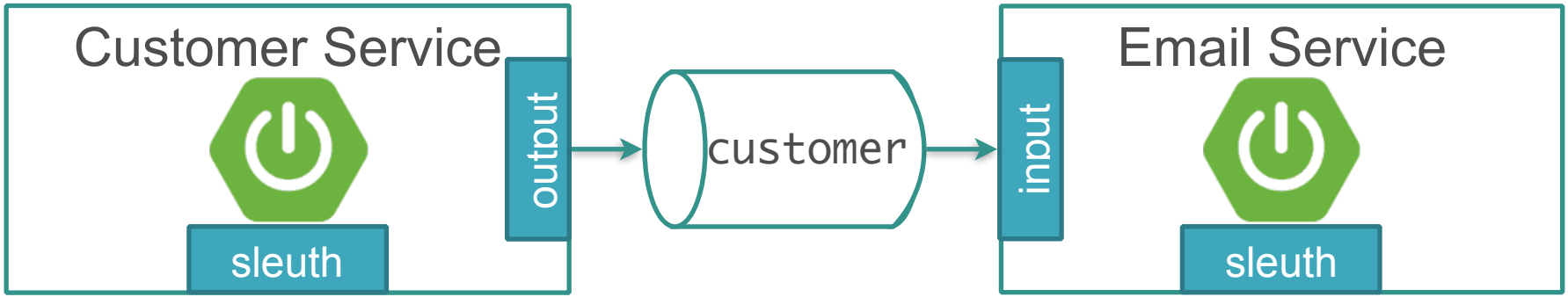
Spring Cloud Sleuth Stream

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-stream-slueth</artifactId>
</dependency>
```

Spring Cloud Sleuth Stream



Spring Cloud Sleuth Stream



Zipkin Stream Server

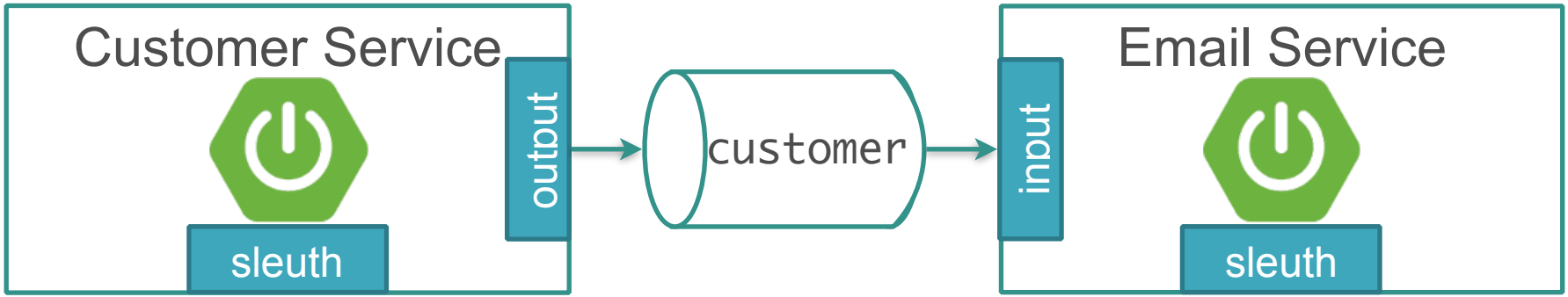
```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-sleuth-zipkin-stream</artifactId>
</dependency>
```

Zipkin Stream Server

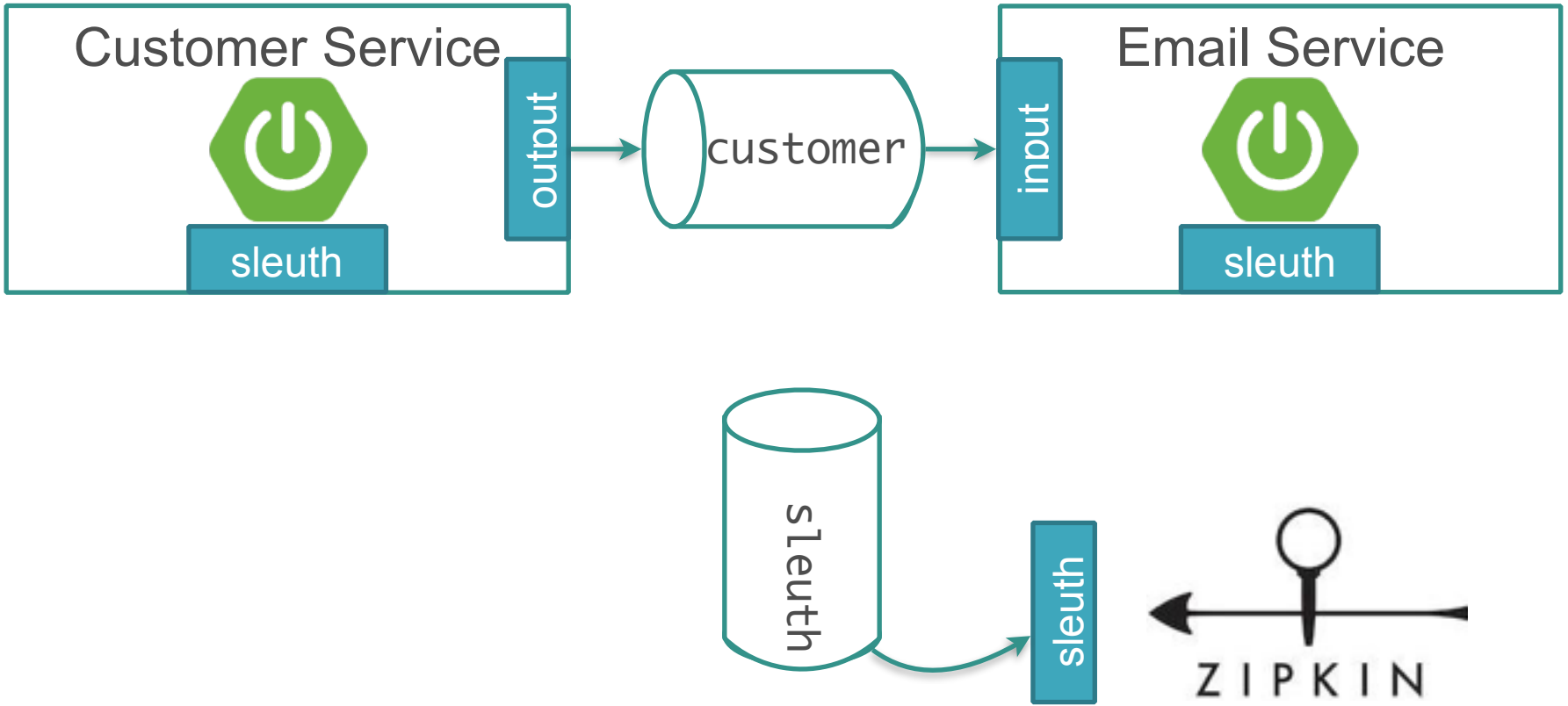
```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-sleuth-zipkin-stream</artifactId>
</dependency>
```

```
@SpringBootApplication
@EnableZipkinStreamServer
public class ZipkinStreamServer {
  public static void main(String[] args) {
    SpringApplication.run(DemoSinkApp.class, args);
  }
}
```

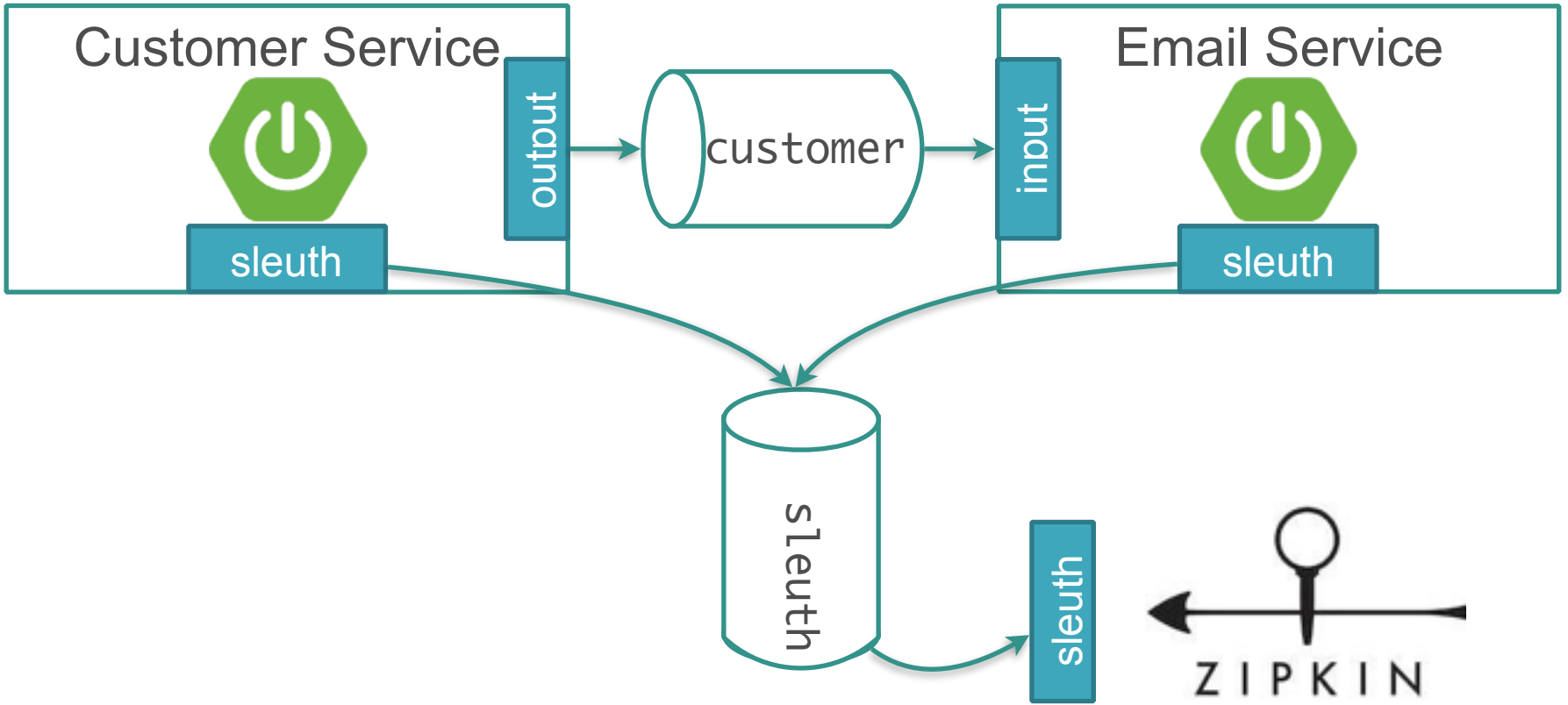
Zipkin Stream Server



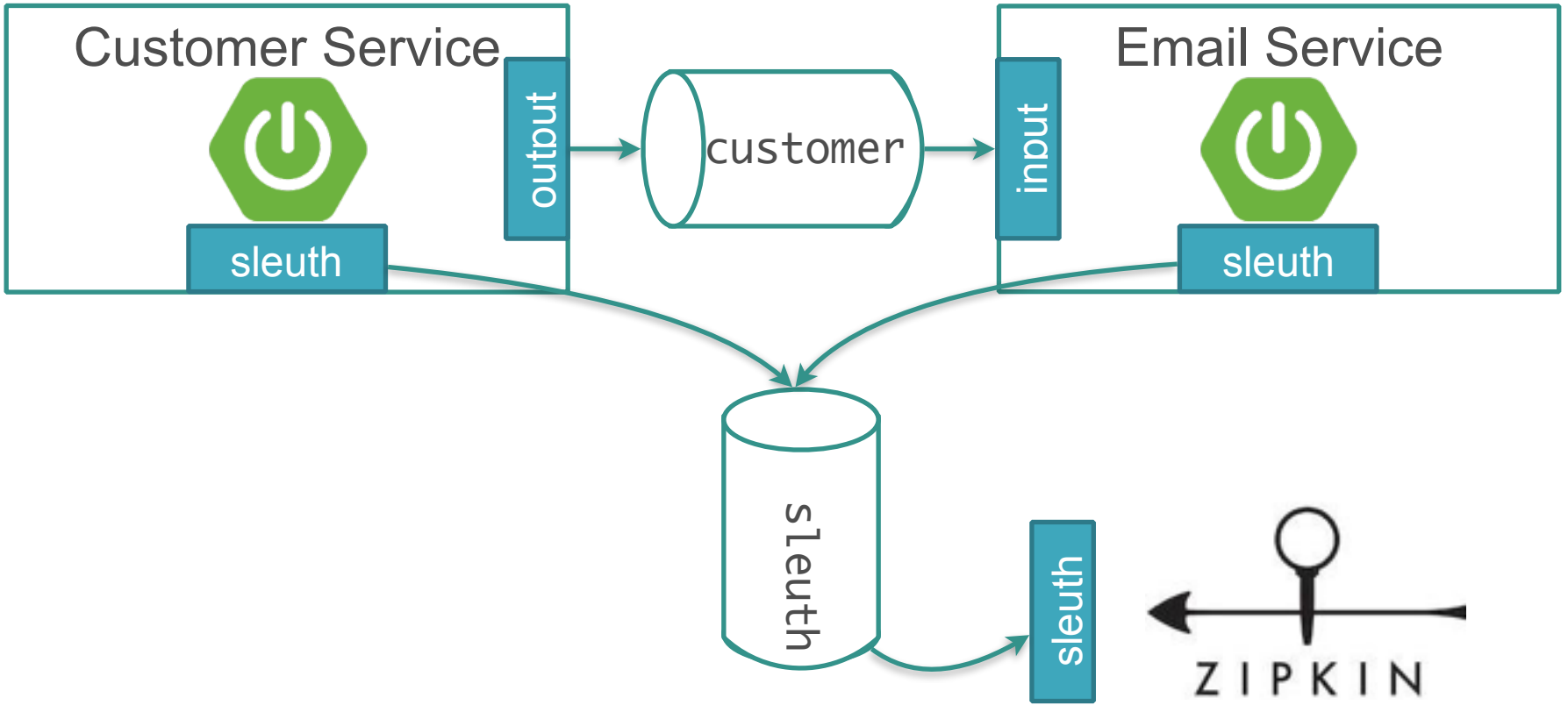
Zipkin Stream Server



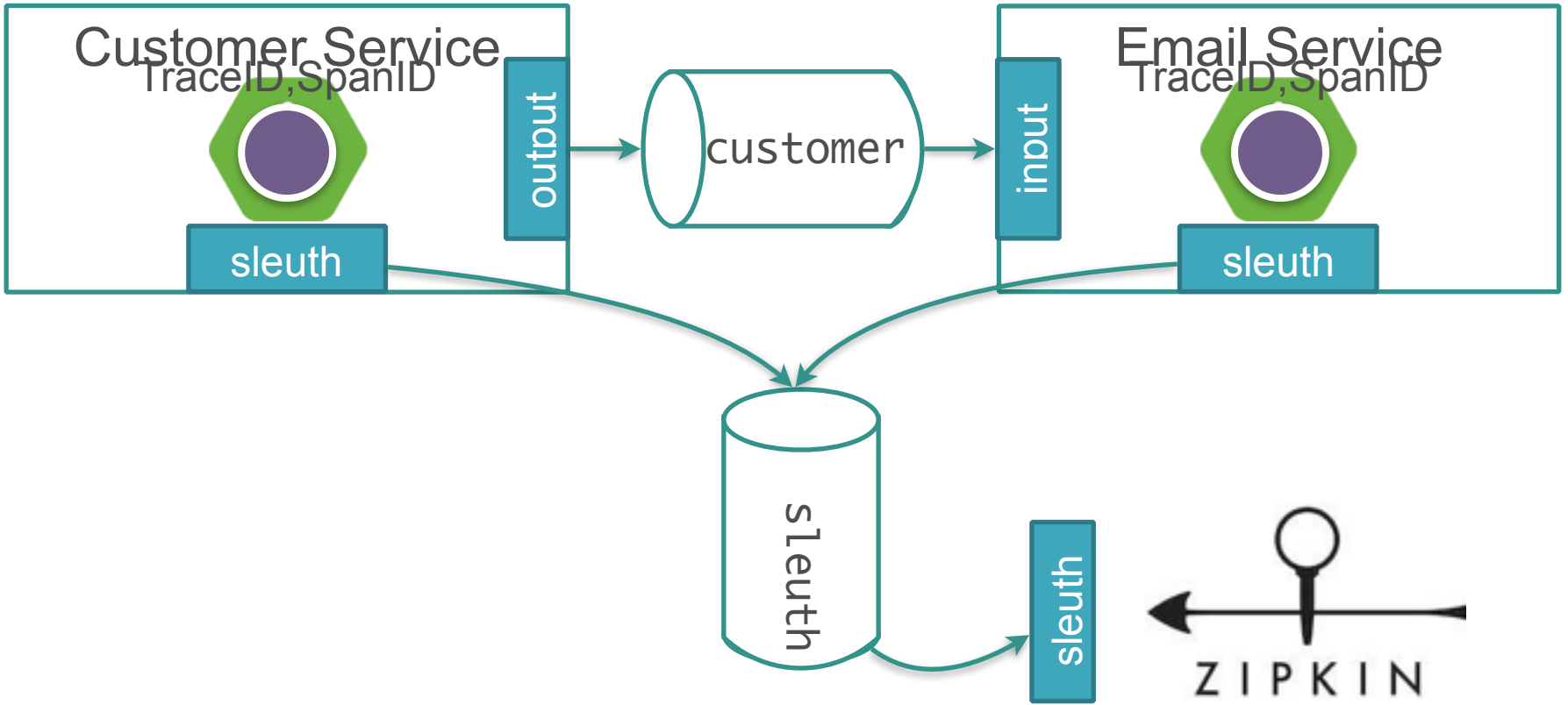
Zipkin Stream Server



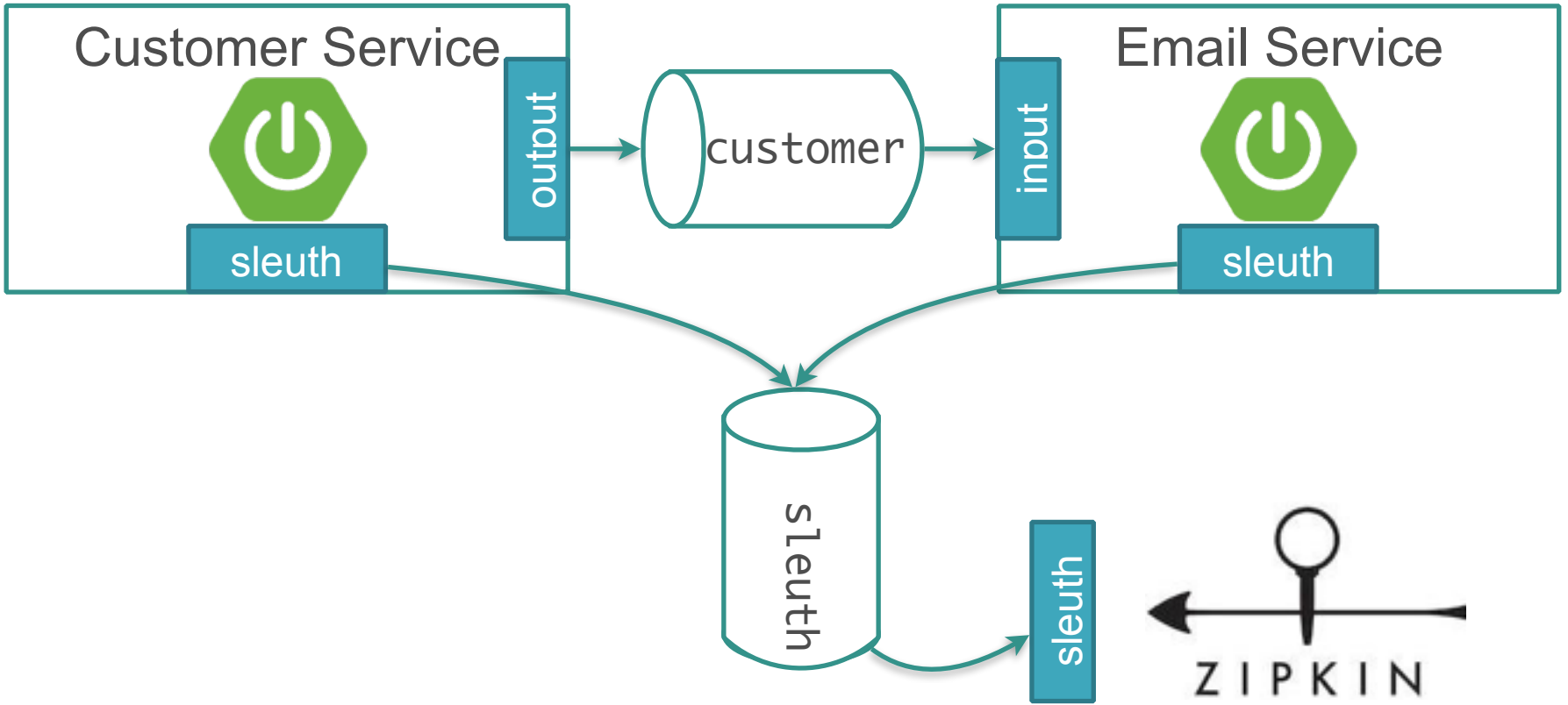
Zipkin Stream Server



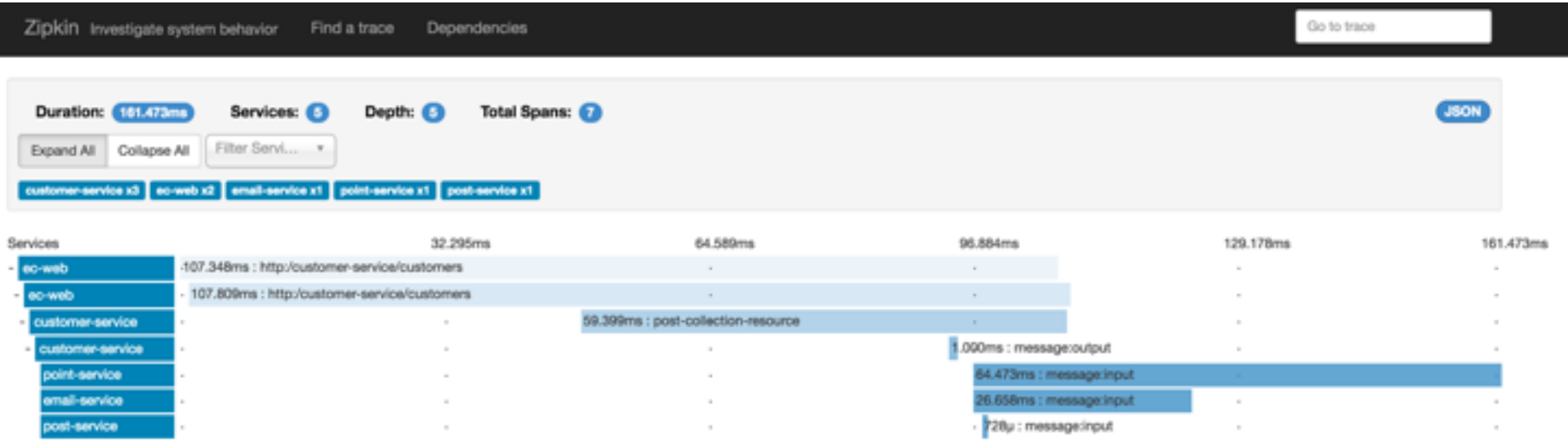
Zipkin Stream Server



Zipkin Stream Server

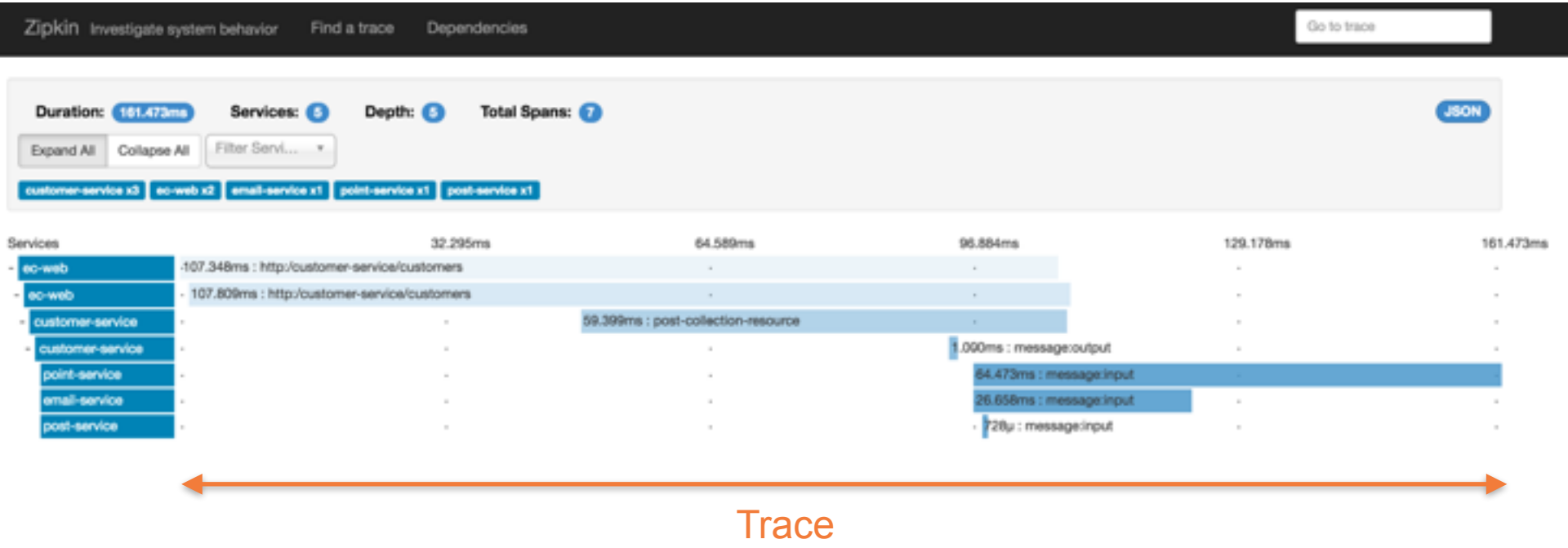


Zipkin UI



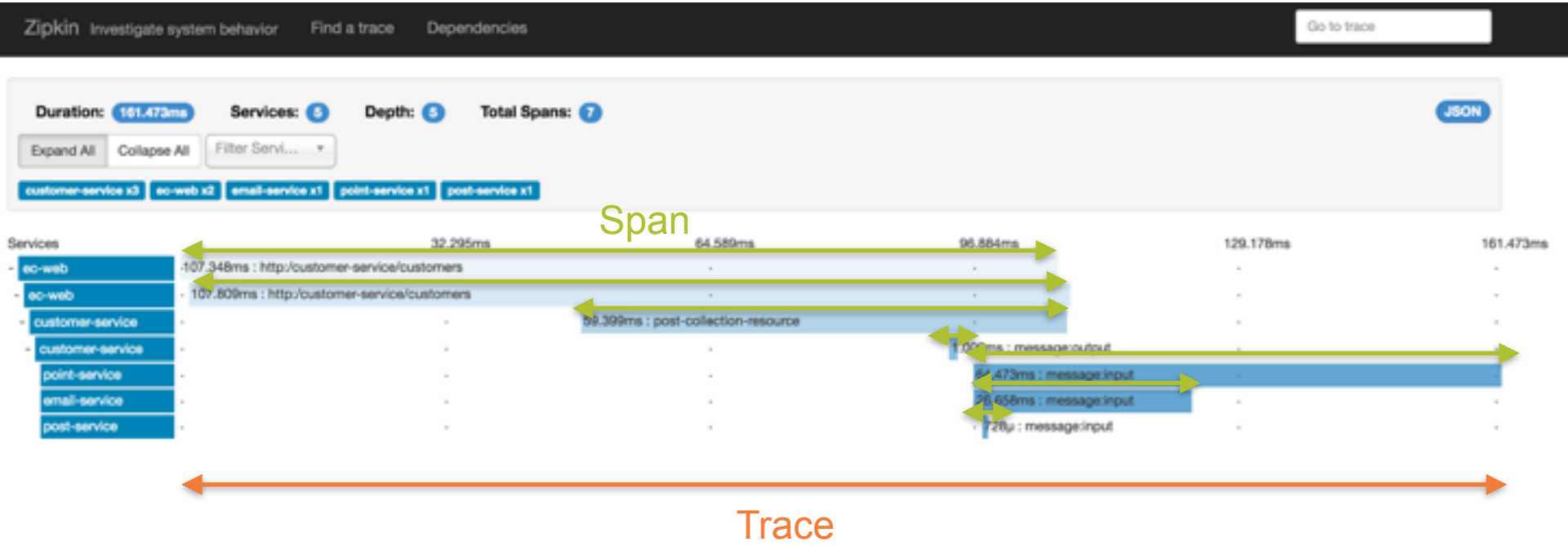
Pivotal

Zipkin UI

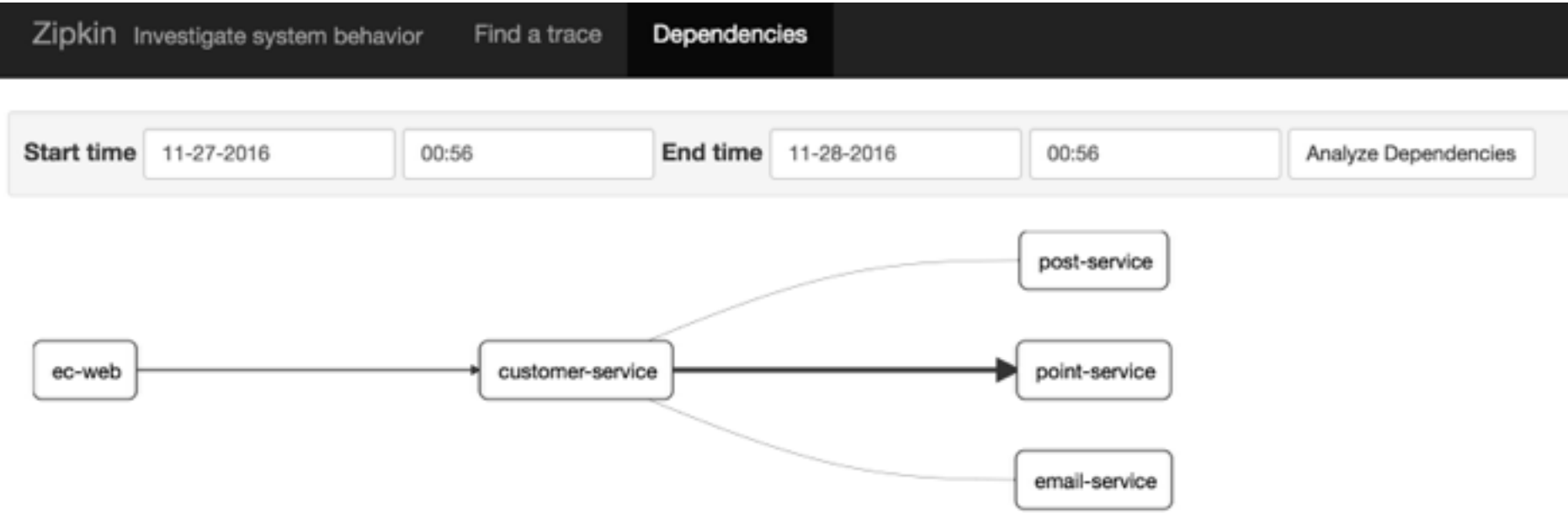


Pivotal

Zipkin UI



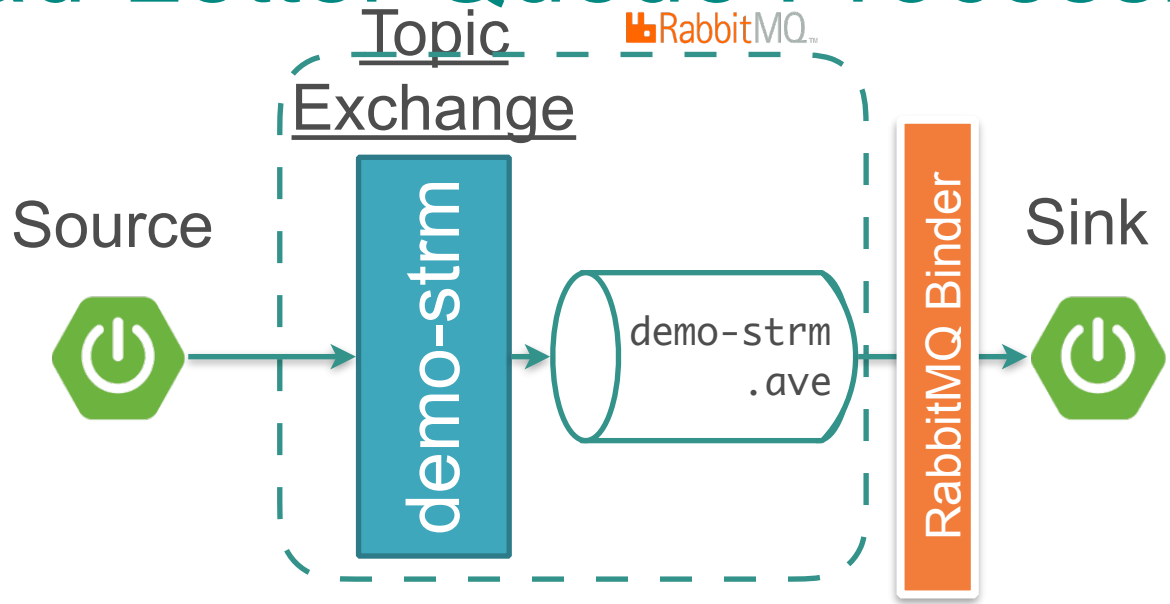
Pivotal



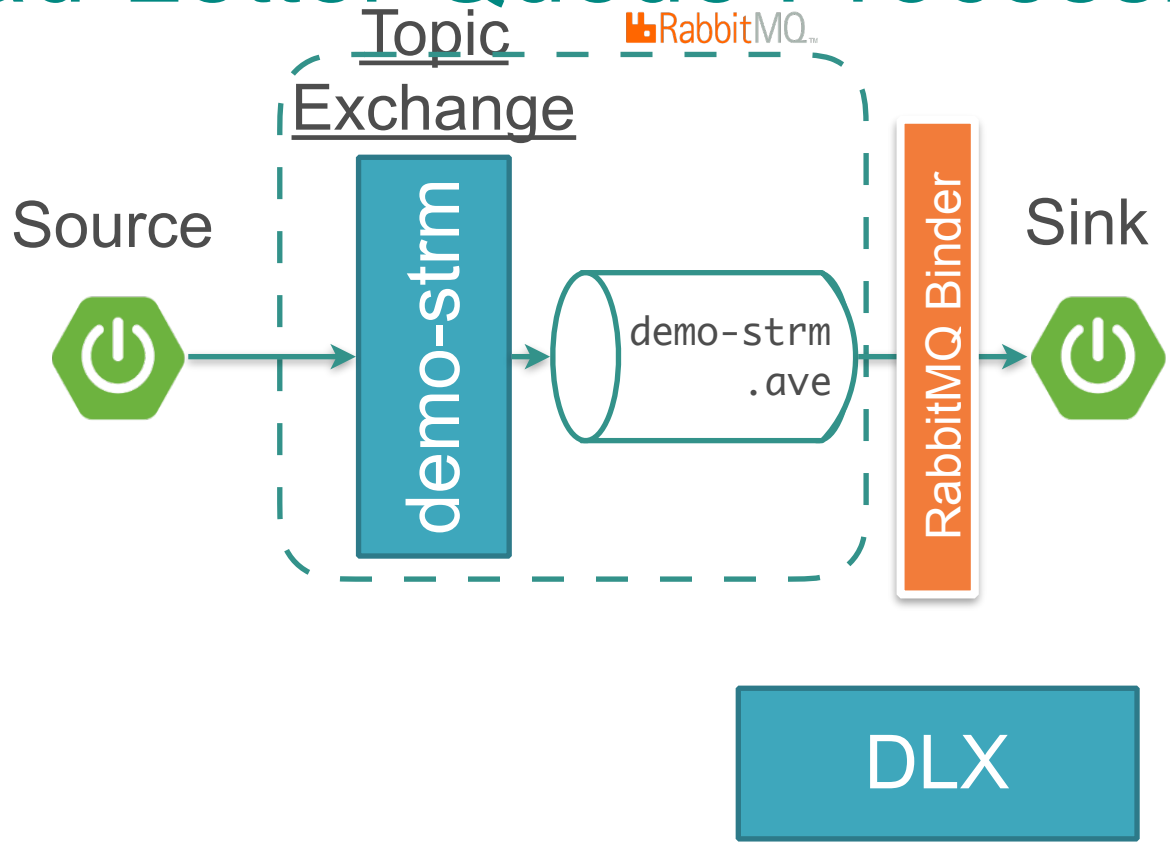
Error Handling

- Depends on the message binder implementation
- (Ex.) RabbitMQ binder routes **the failed message to the Dead-Letter Queue(DLQ)**. No mechanism to handle DLQs.

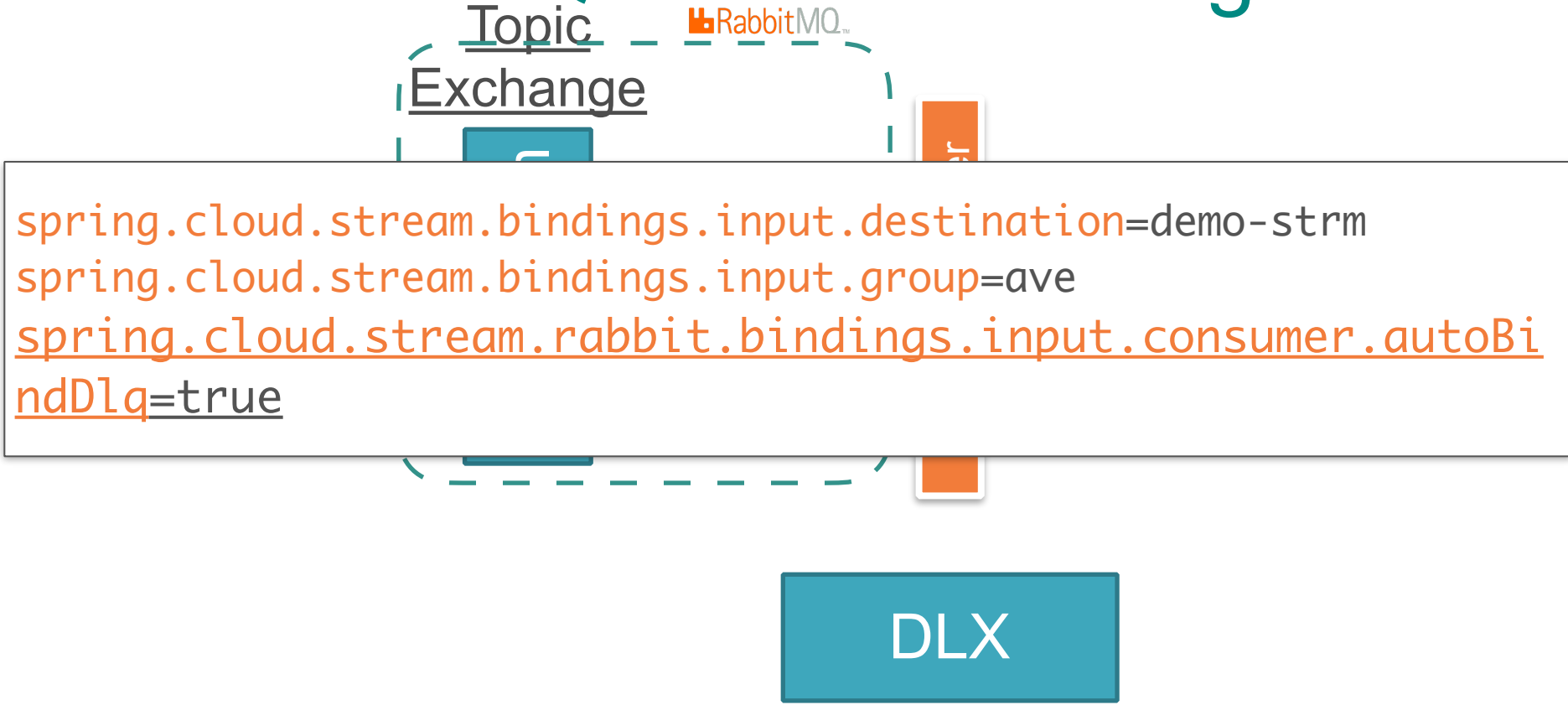
Dead-Letter Queue Processing



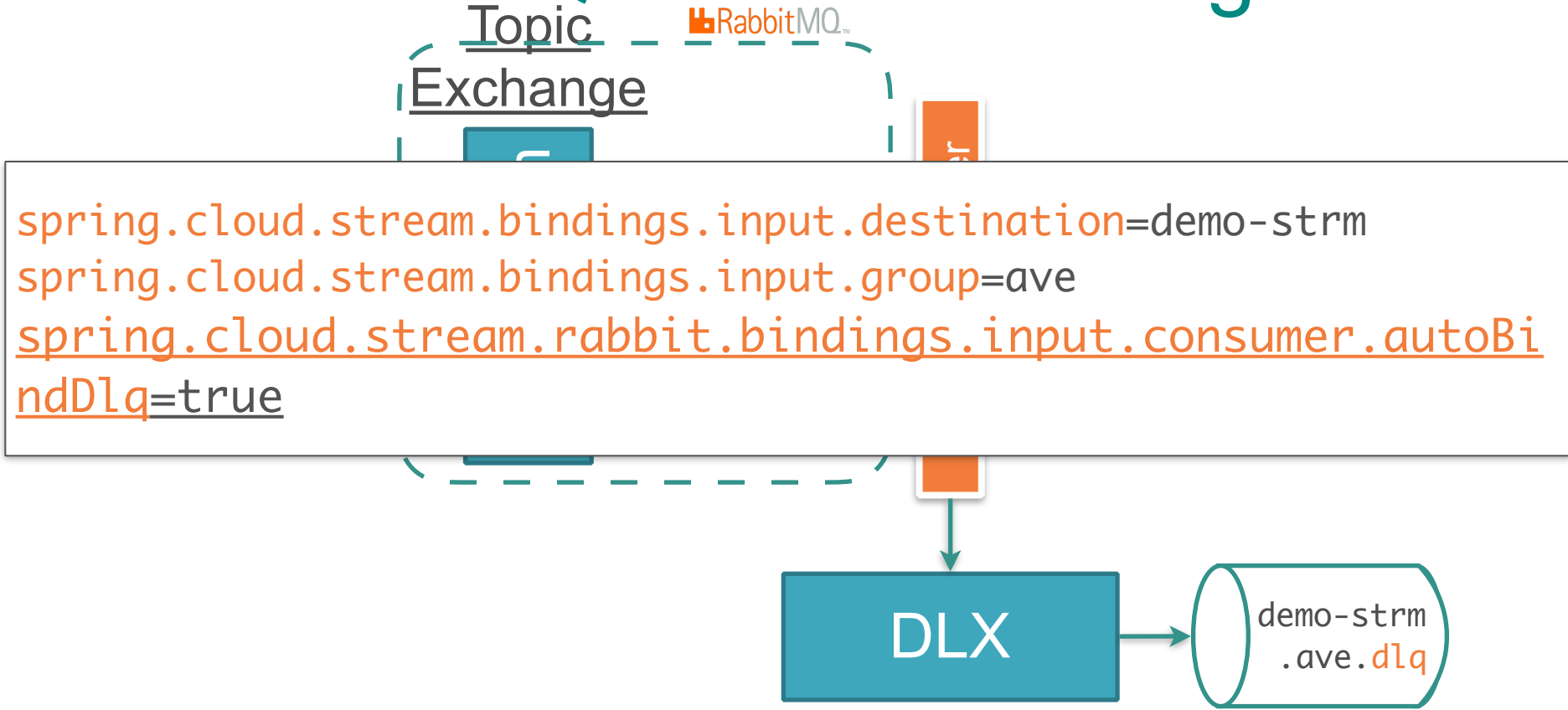
Dead-Letter Queue Processing



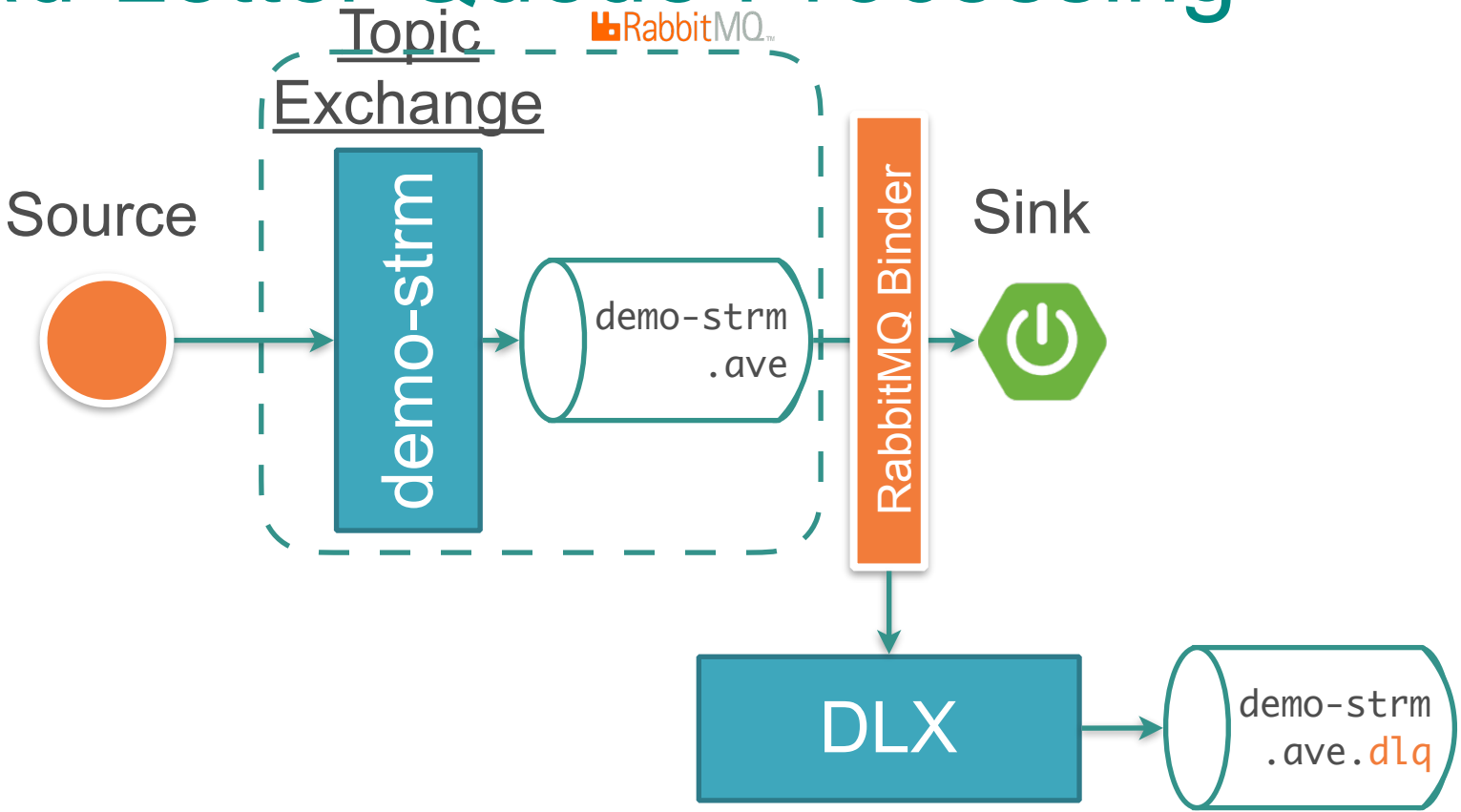
Dead-Letter Queue Processing



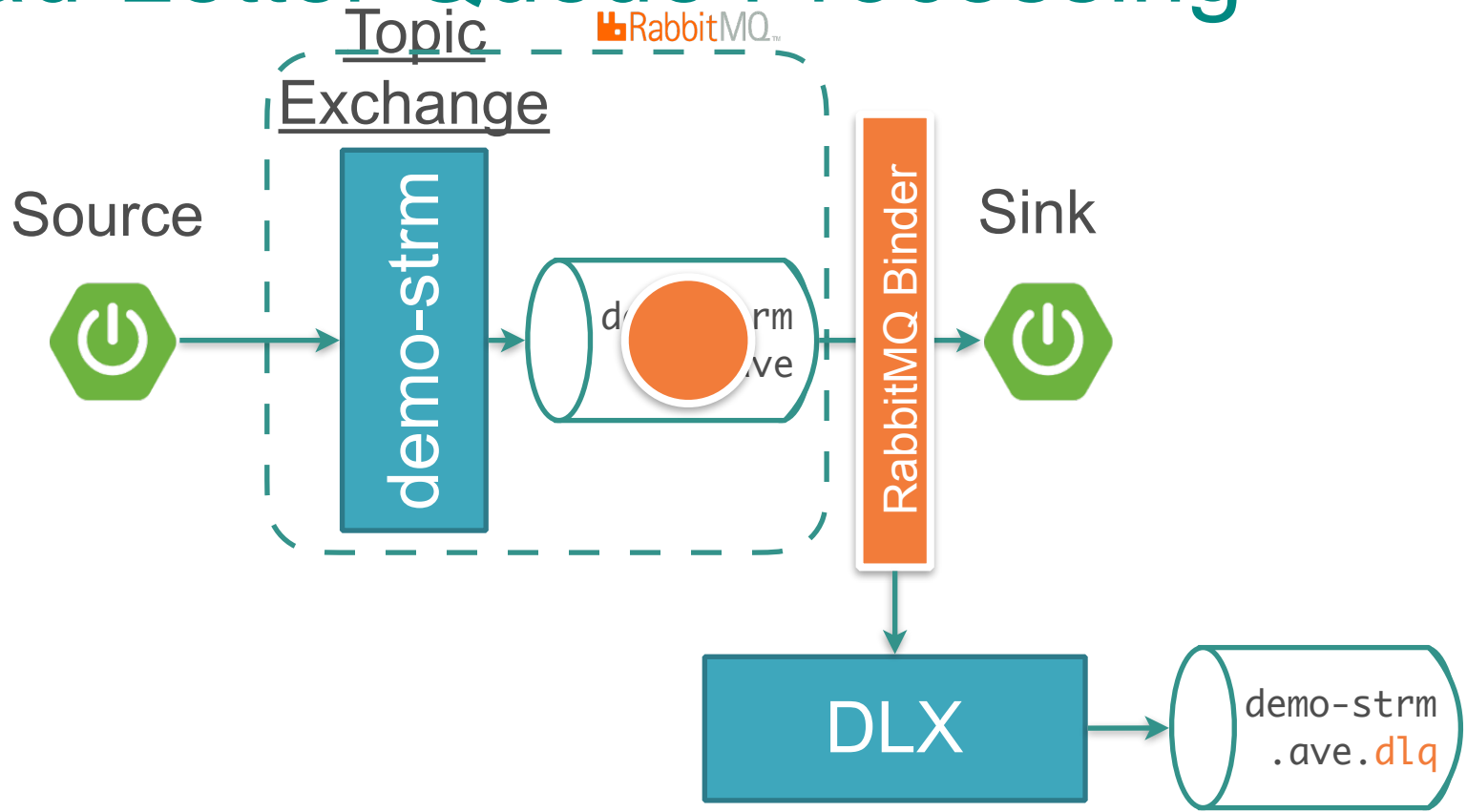
Dead-Letter Queue Processing



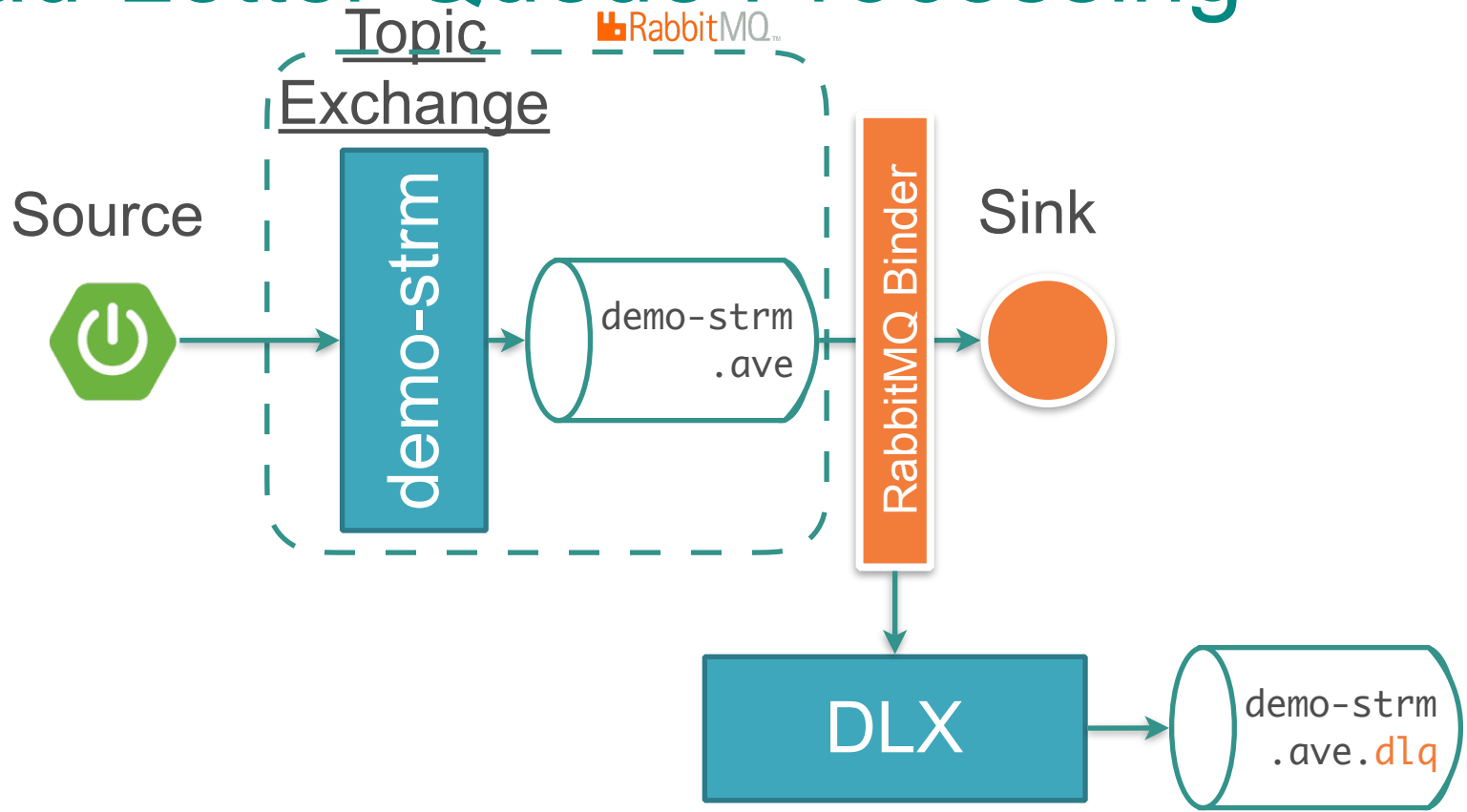
Dead-Letter Queue Processing



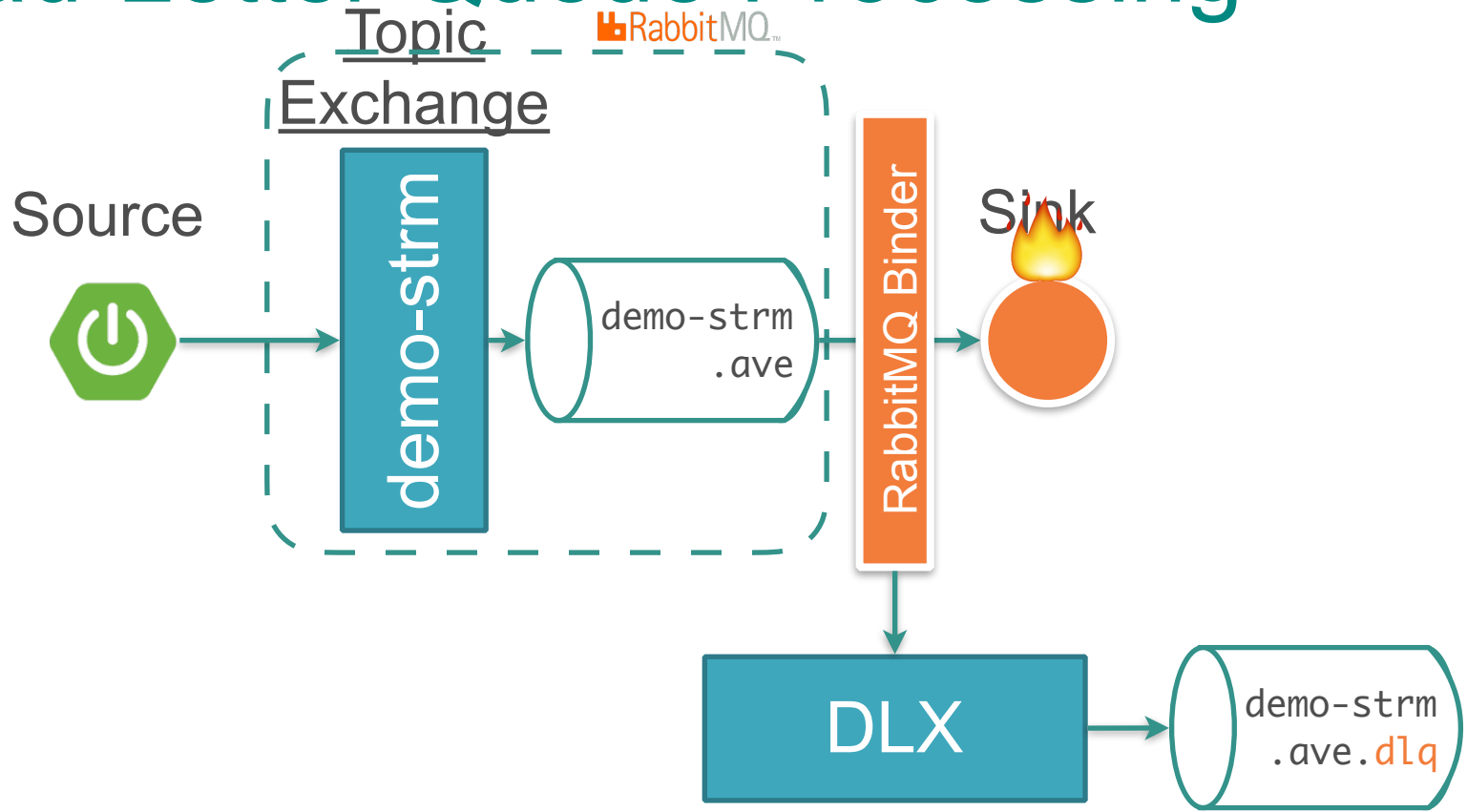
Dead-Letter Queue Processing



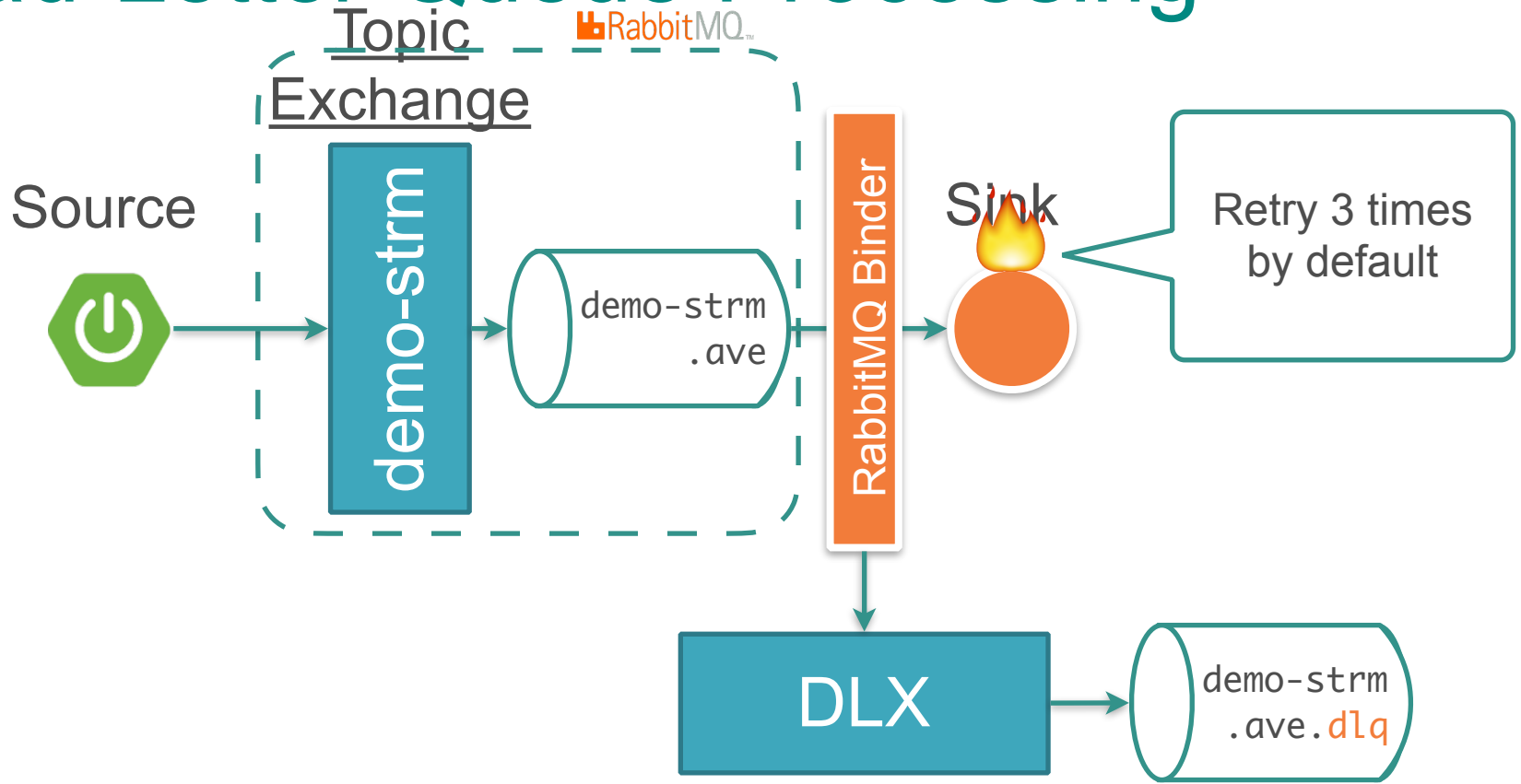
Dead-Letter Queue Processing



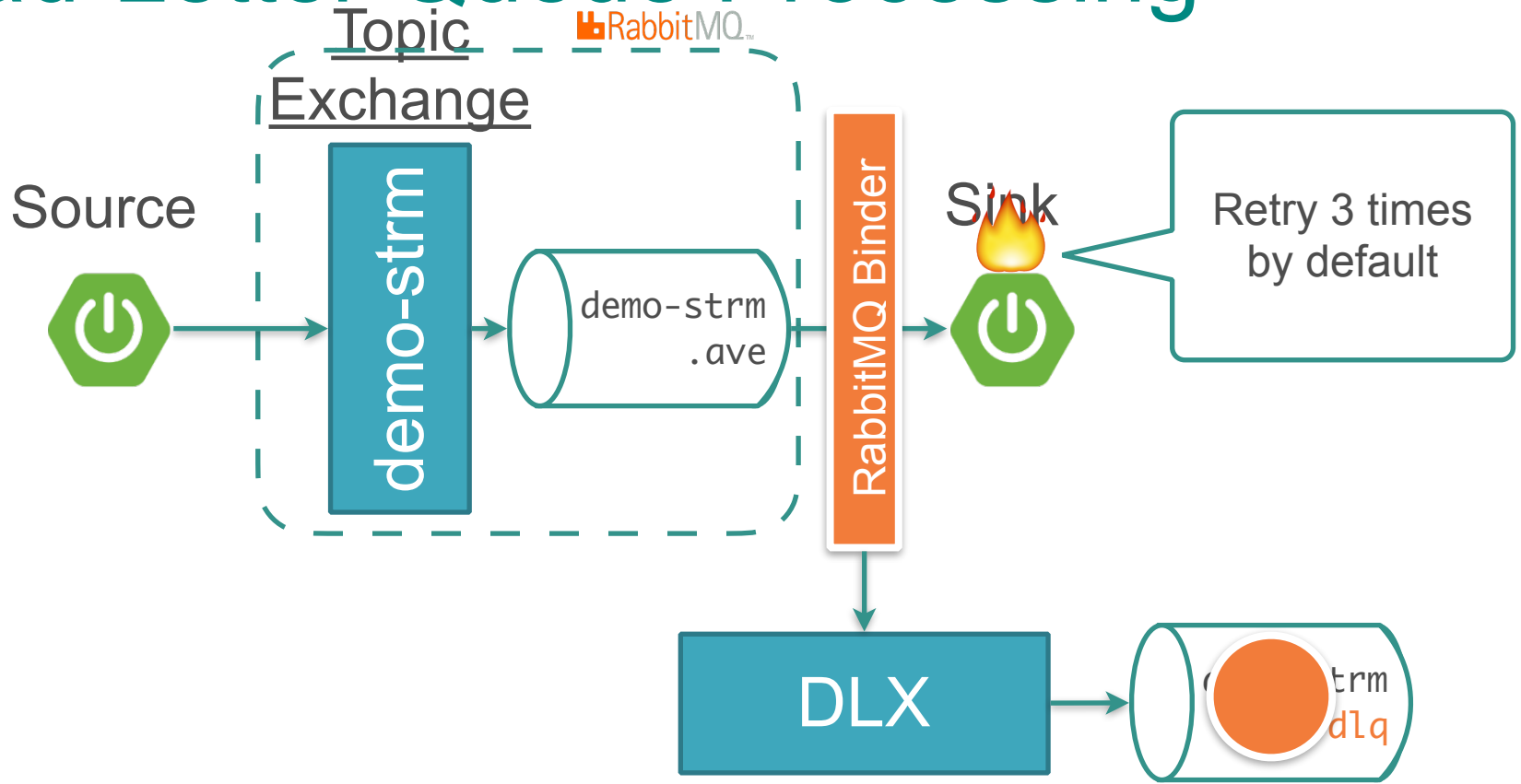
Dead-Letter Queue Processing



Dead-Letter Queue Processing



Dead-Letter Queue Processing



Overview | Connections | Channels | **Exchanges** | Queues | Admin

Exchanges

All exchanges (10)

Pagination

Page 1 of 1 - Filter: Regex (?)(?)

Name	Type	Features	Message rate in	Message rate out	+/-
(AMQP default)	direct	D HA	0.00/s	0.00/s	
DLX	direct	D HA			
amq.direct	direct	D			
amq.fanout	fanout	D			
amq.headers	headers	D			
amq.match	headers	D			
amq.rabbitmq.trace	topic	D I			
amq.topic	topic	D			
customer	topic	D HA	0.00/s	0.00/s	
sleuth	topic	D HA	0.00/s	0.00/s	

Overview | Connections | Channels | **Exchanges** | Queues | Admin

Exchanges

All exchanges (10)

Pagination

Page 1 of 1 - Filter: Regex (?)(?)

Name	Type	Features	Message rate in	Message rate out	+/-
(AMQP default)	direct	D HA	0.00/s	0.00/s	
DLX	direct	D HA			
amq.direct	direct	D			
amq.fanout	fanout	D			
amq.headers	headers	D			
amq.match	headers	D			
amq.rabbitmq.trace	topic	D I			
amq.topic	topic	D			
customer	topic	D HA	0.00/s	0.00/s	
sleuth	topic	D HA	0.00/s	0.00/s	

RabbitMQ

Overview | Connections | Channels | Exchanges | **Queues** | Admin

Queues

▸ **All queues (7)**

Overview			Messages			Message rates			+/-
Name	Features	**State**	**Ready**	**Unacked**	**Total**	**incoming**	**deliver / get**	**ack**	
customer.email-service	D DLX DLK HA	idle	0	0	0	0.00/s	0.00/s	0.00/s	
customer.email-service.dlq	D HA	idle	0	0	0	0.00/s	0.00/s	0.00/s	
customer.point-service	D DLX DLK HA	idle	0	0	0	0.00/s	0.00/s	0.00/s	
customer.point-service.dlq	D HA	idle	0	0	0	0.00/s	0.00/s	0.00/s	
customer.post-service	D DLX DLK HA	idle	0	0	0	0.00/s	0.00/s	0.00/s	
customer.post-service.dlq	D HA	idle	0	0	0				
sleuth.sleuth	D HA	idle	0	0	0	0.00/s	0.00/s	0.00/s	

Pivotal

RabbitMQ

Overview | Connections | Channels | Exchanges | **Queues** | Admin

Queues

▸ **All queues (7)**

Overview			Messages			Message rates			+/-
Name	Features	**State**	**Ready**	**Unacked**	**Total**	**incoming**	**deliver / get**	**ack**	
customer.email-service	D DLX DLK HA	idle	0	0	0	0.00/s	0.00/s	0.00/s	
customer.email-service.dlq	D HA	idle	0	0	0	0.00/s	0.00/s	0.00/s	
customer.point-service	D DLX DLK HA	idle	0	0	0	0.00/s	0.00/s	0.00/s	
customer.point-service.dlq	D HA	idle	0	0	0	0.00/s	0.00/s	0.00/s	
customer.post-service	D DLX DLK HA	idle	0	0	0	0.00/s	0.00/s	0.00/s	
customer.post-service.dlq	D HA	idle	0	0	0				
sleuth.sleuth	D HA	idle	0	0	0	0.00/s	0.00/s	0.00/s	

Pivotal

Exchange: DLX

Overview

Bindings

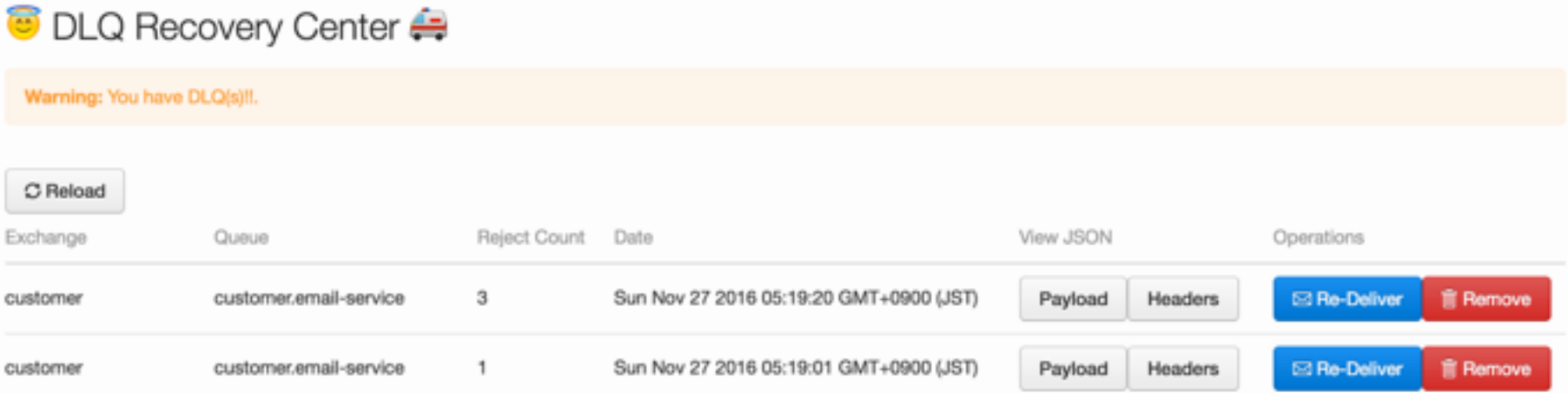
This exchange

To	Routing key	Arguments	
customer.email-service.dlq	customer.email-service		Unbind
customer.point-service.dlq	customer.point-service		Unbind
customer.post-service.dlq	customer.post-service		Unbind

Handling DLQ

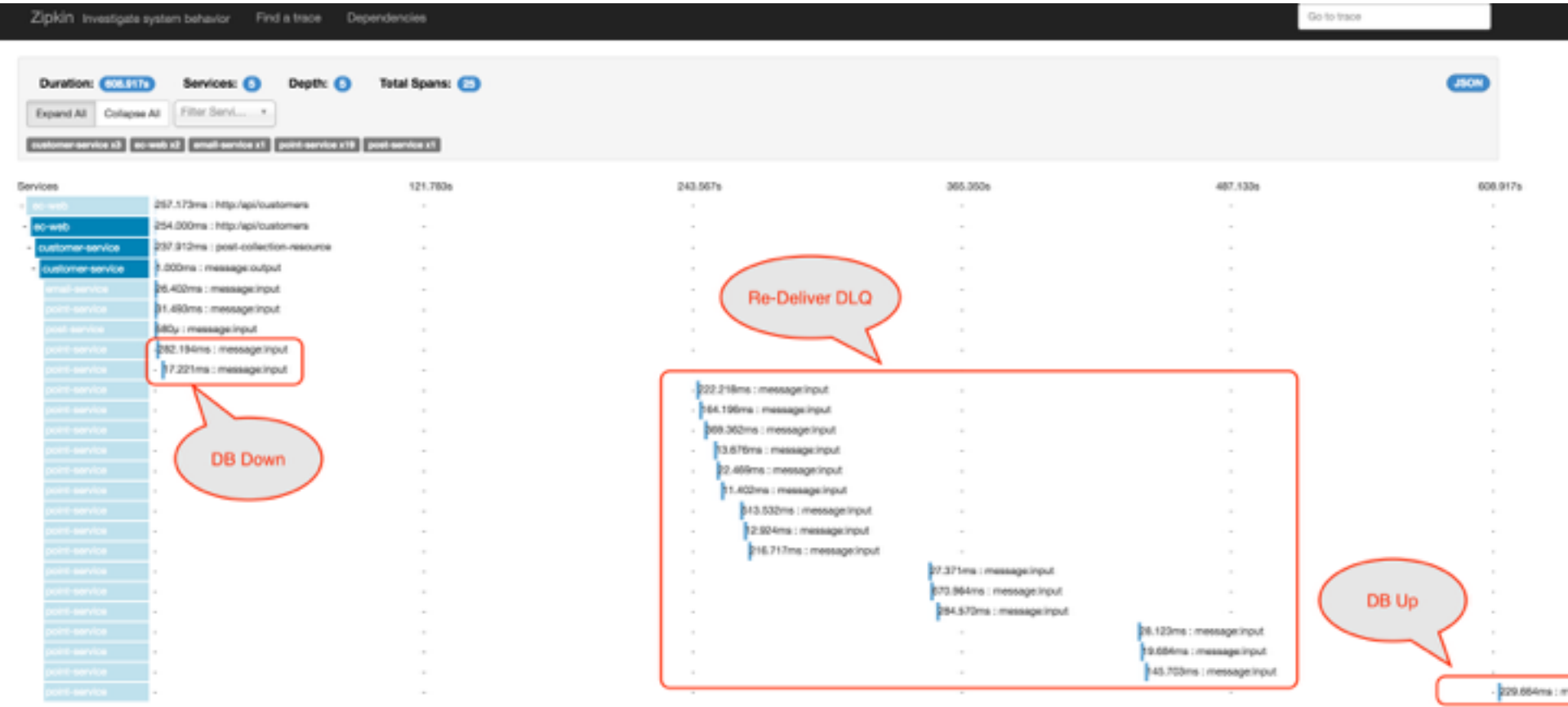
```
@Component
public class DlqHander {
  @RabbitListener(queues = "customer.email-service.dlq")
  public void handle(Message event) {
    // Re-deliver if you want
  }
}
```

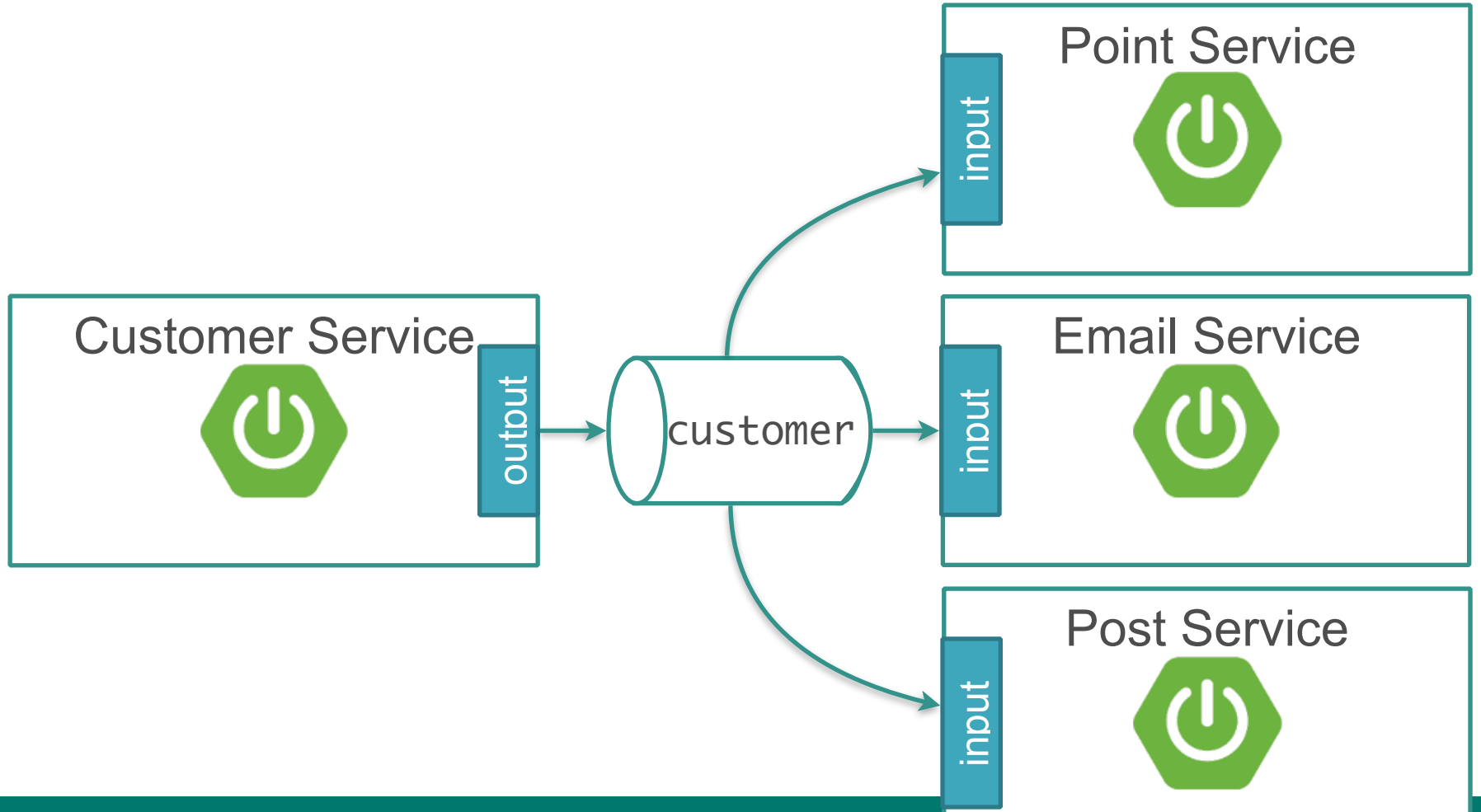
DLQ Recovery Center 😛

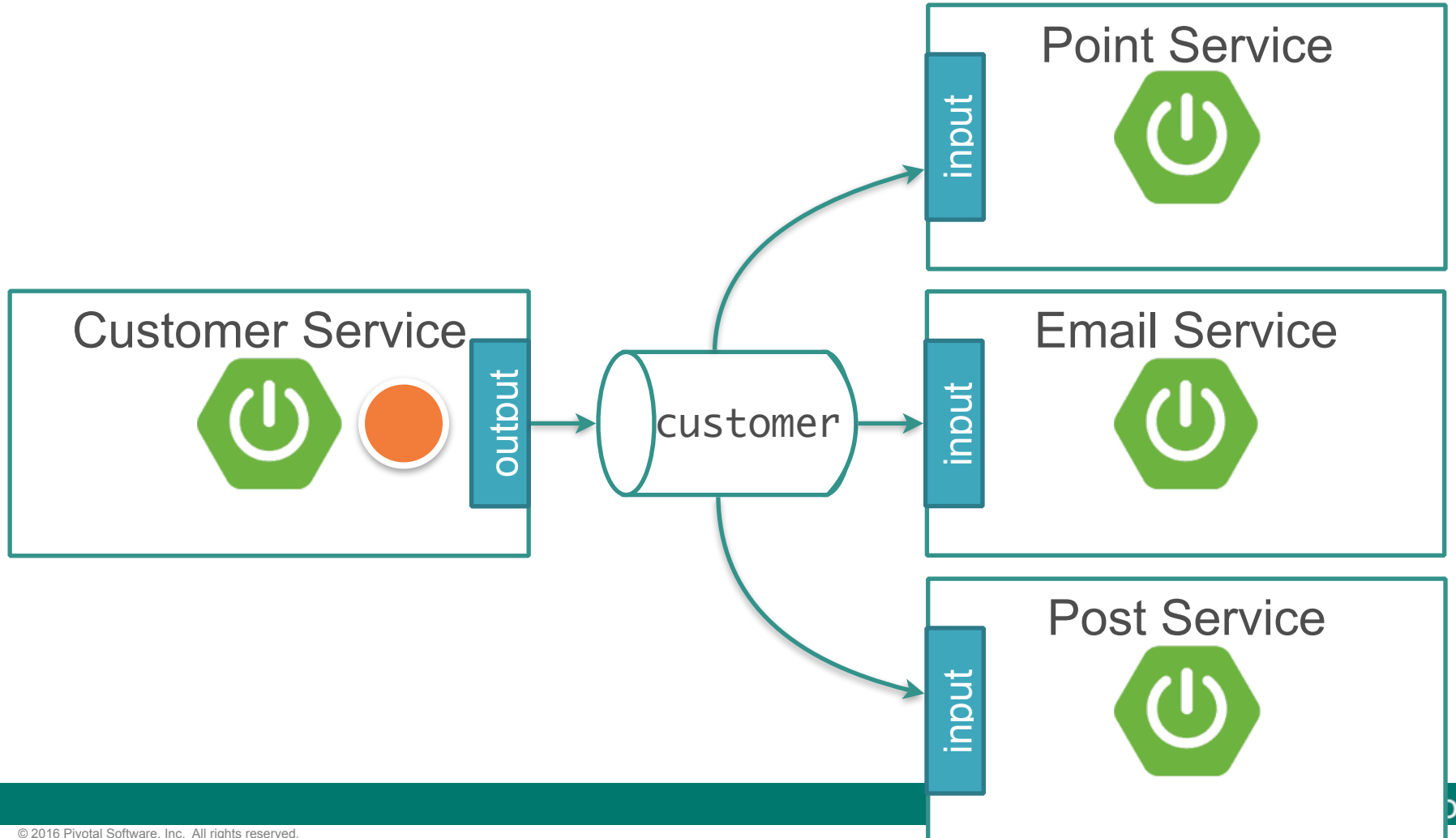


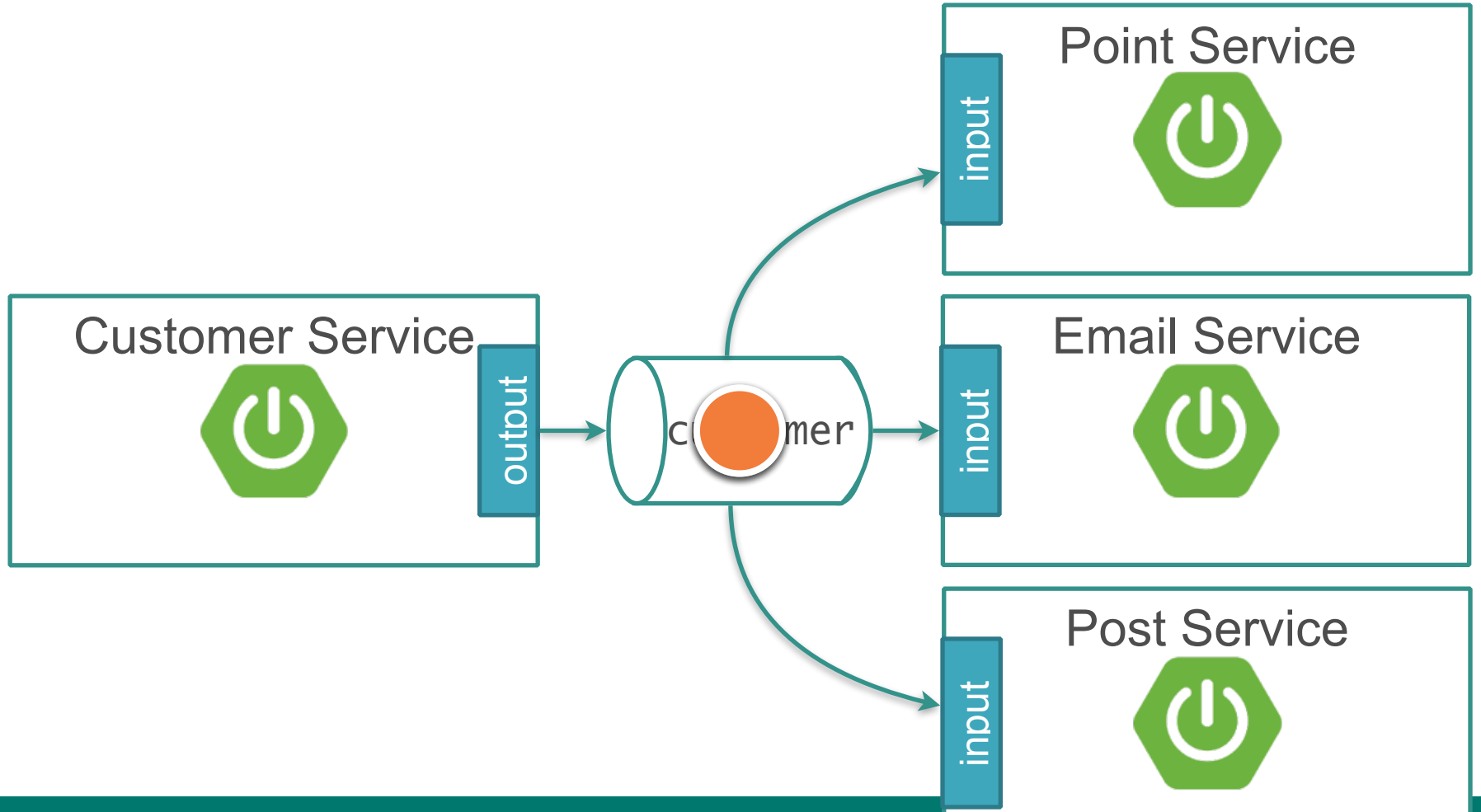
https://github.com/making-demo-scst/dlq-recover-service

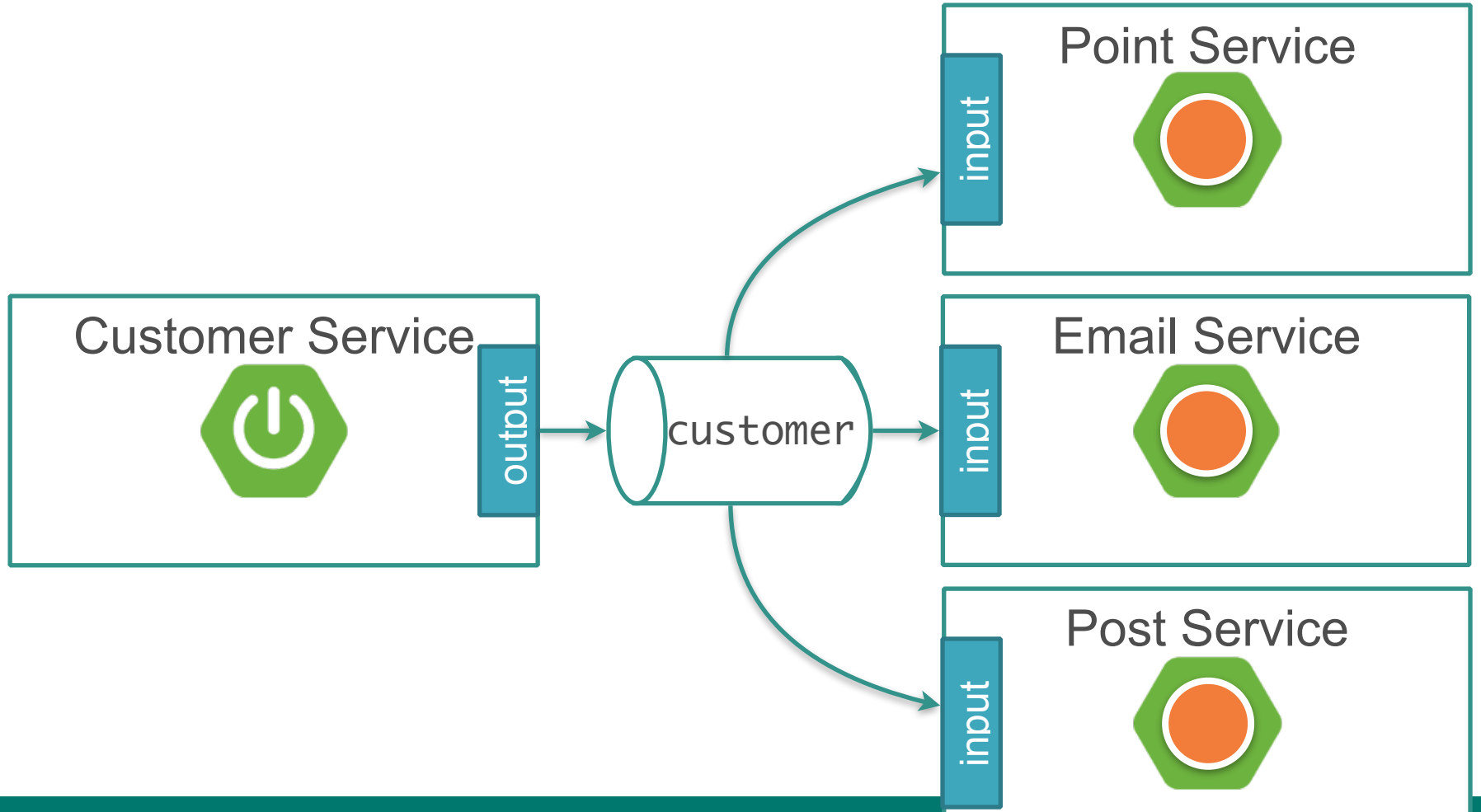
Trace everything

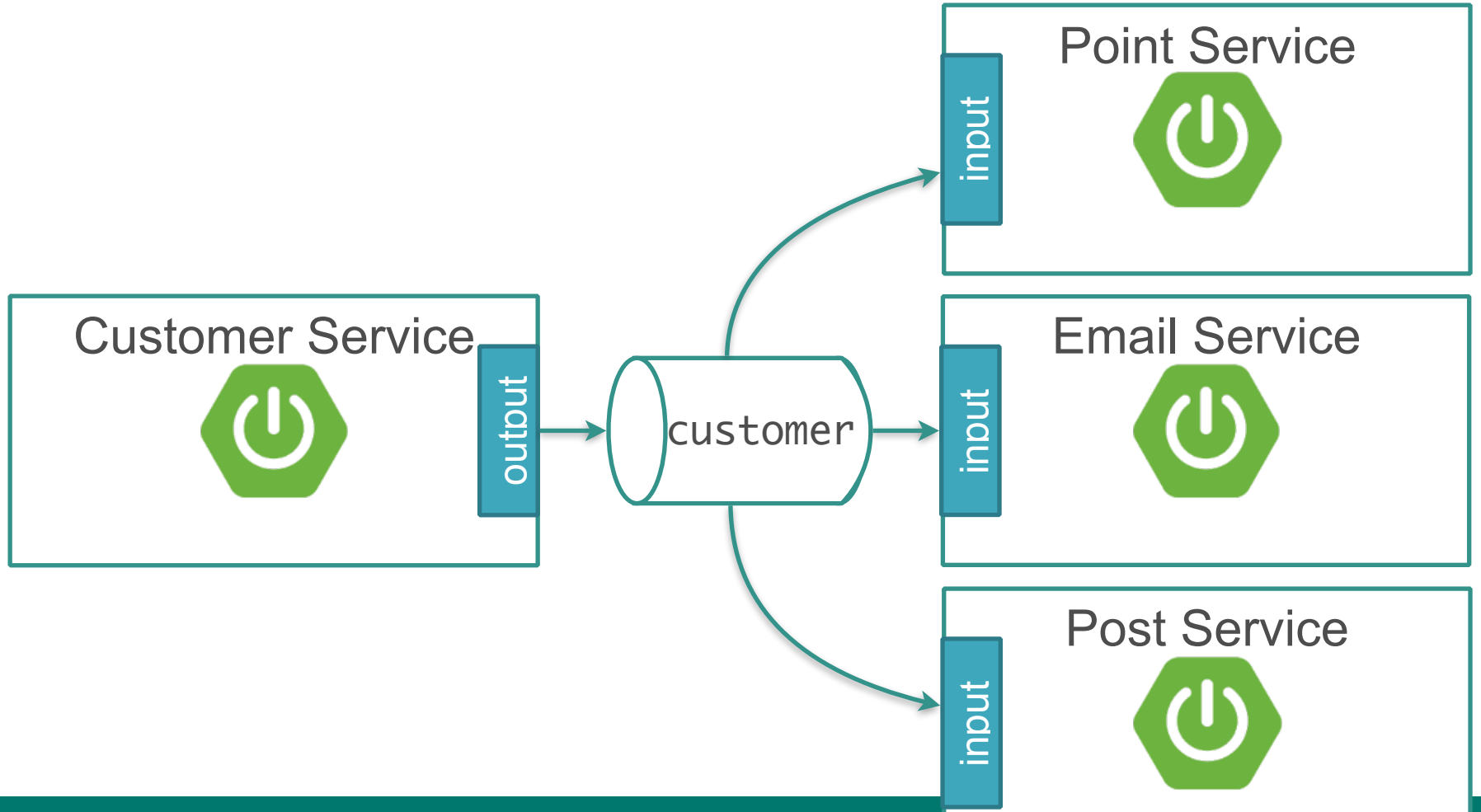


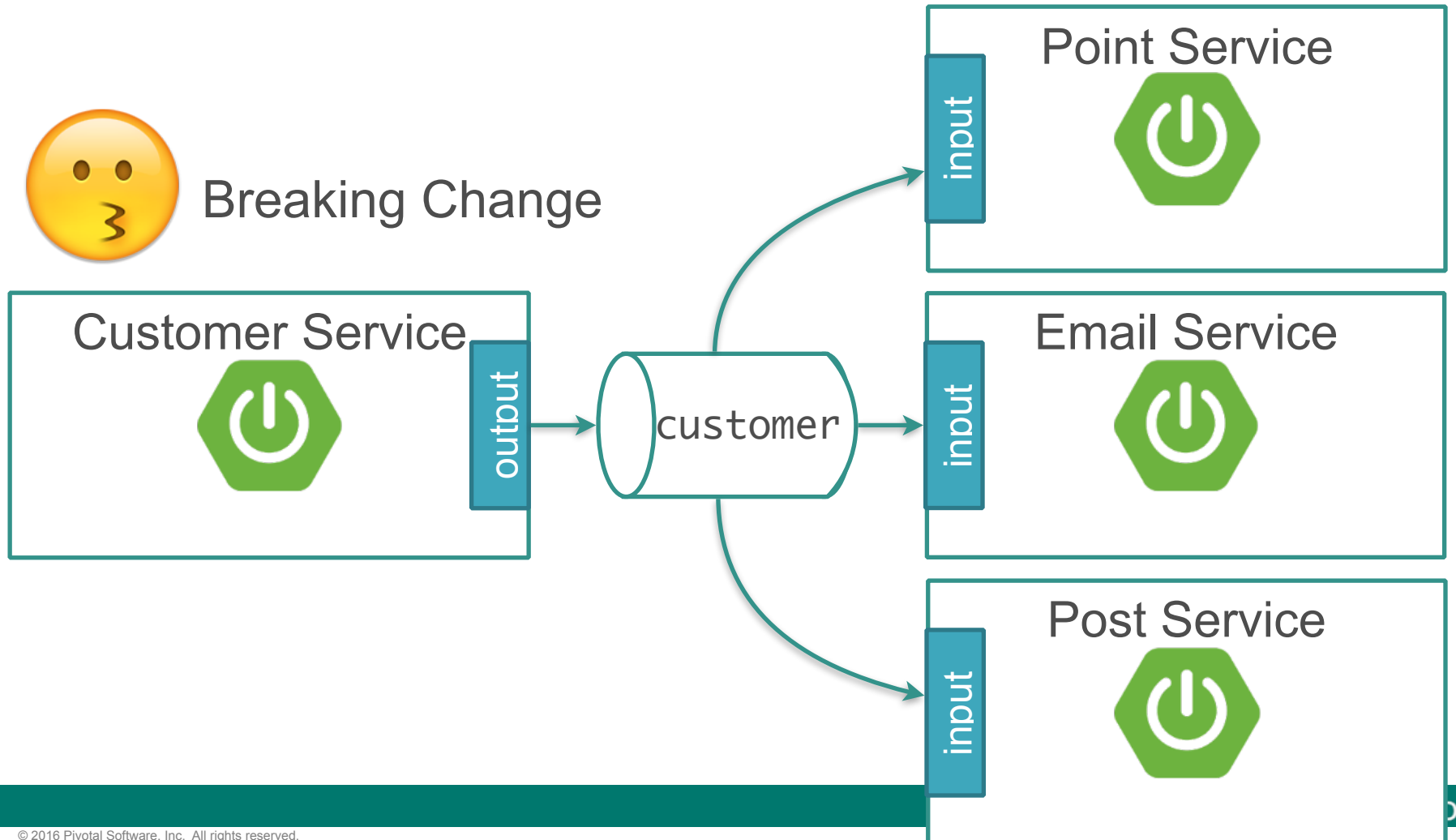


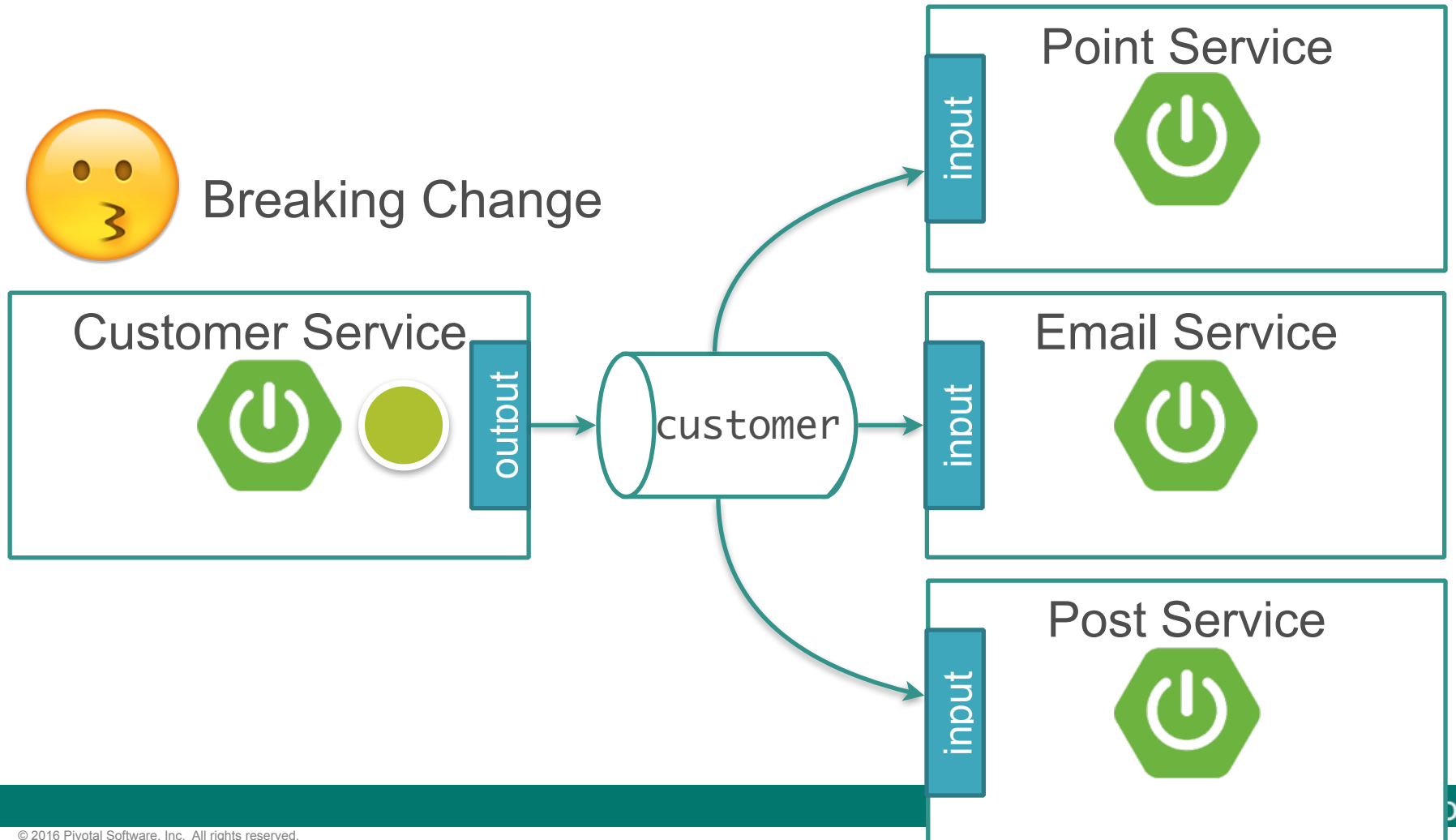


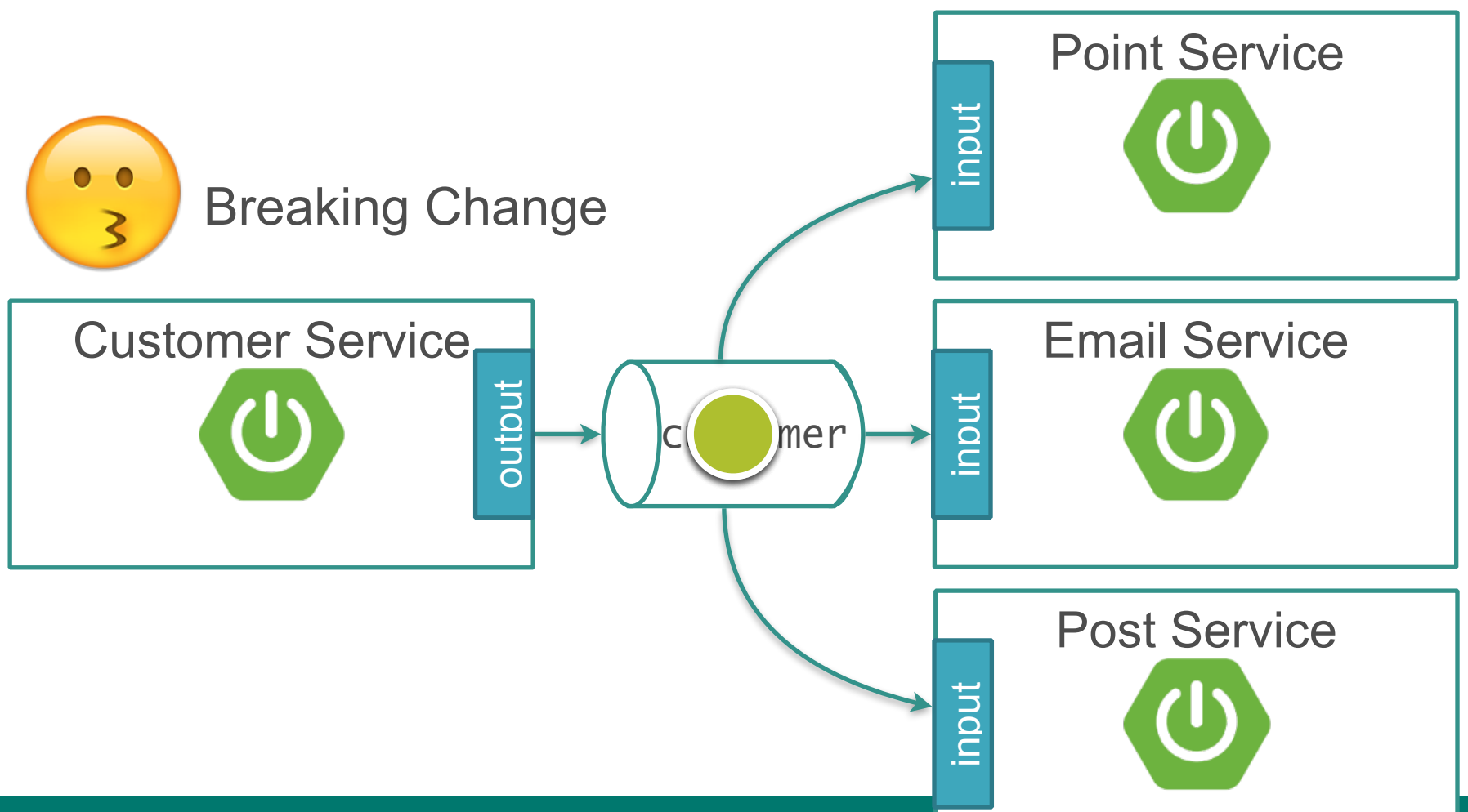


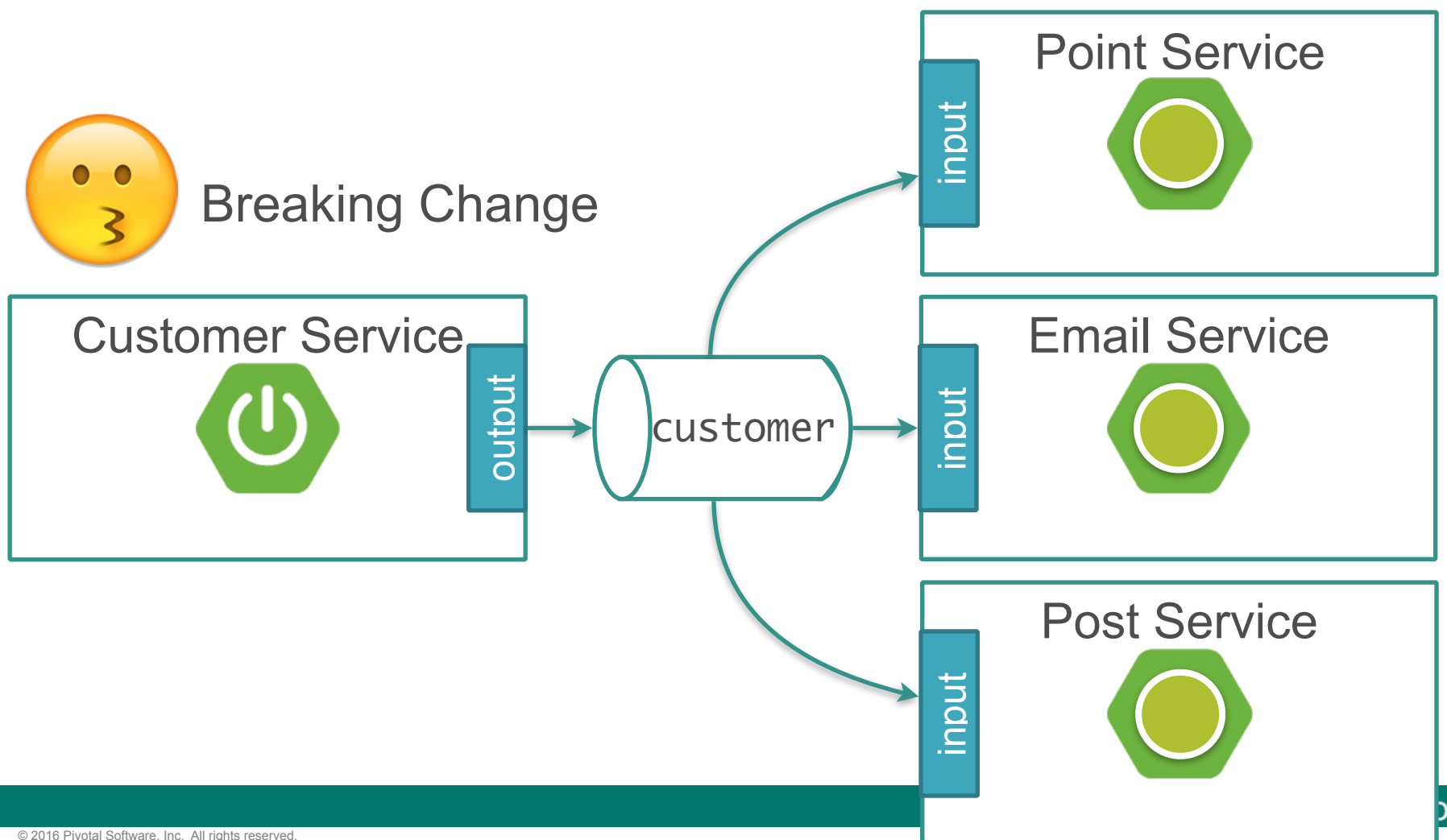


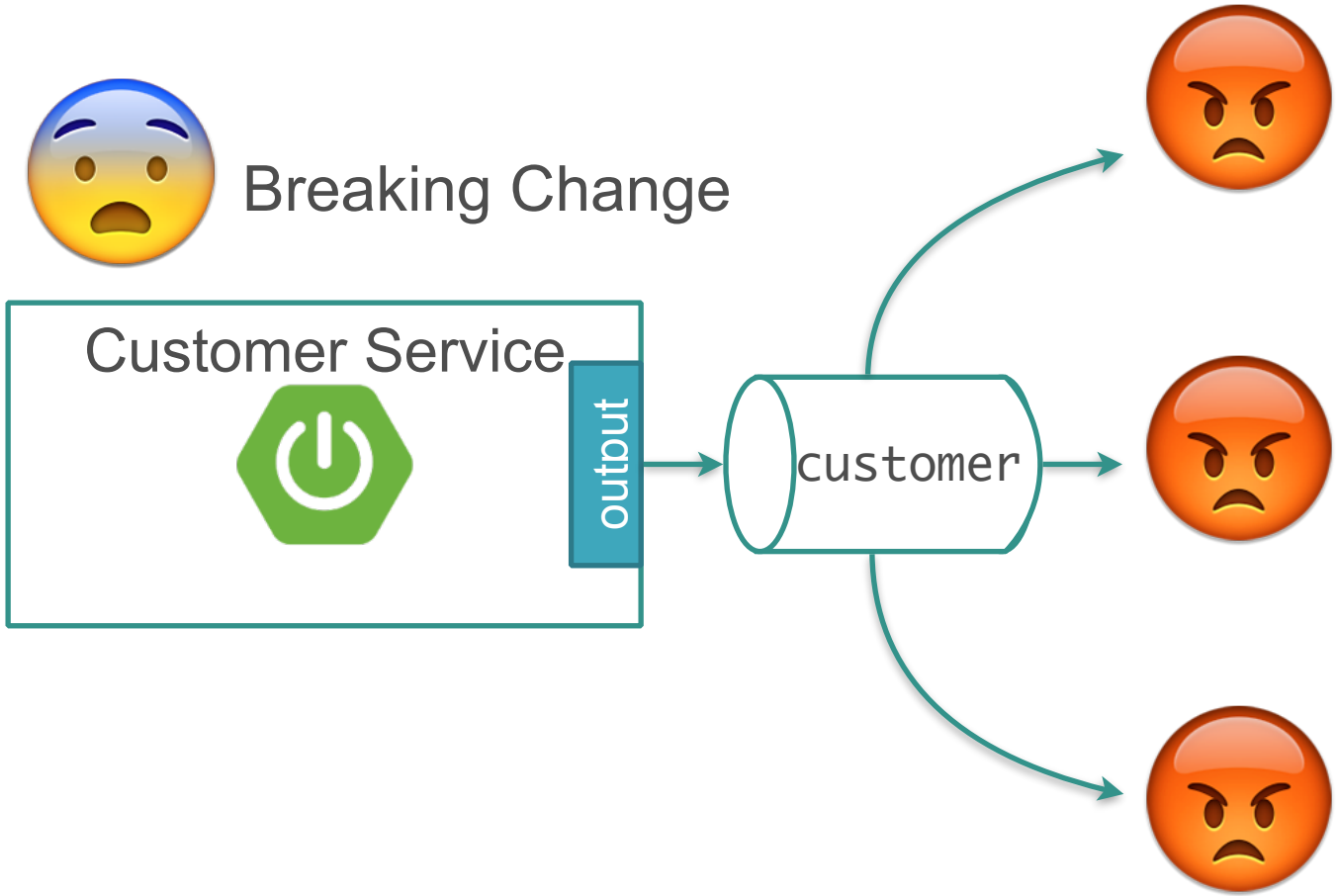












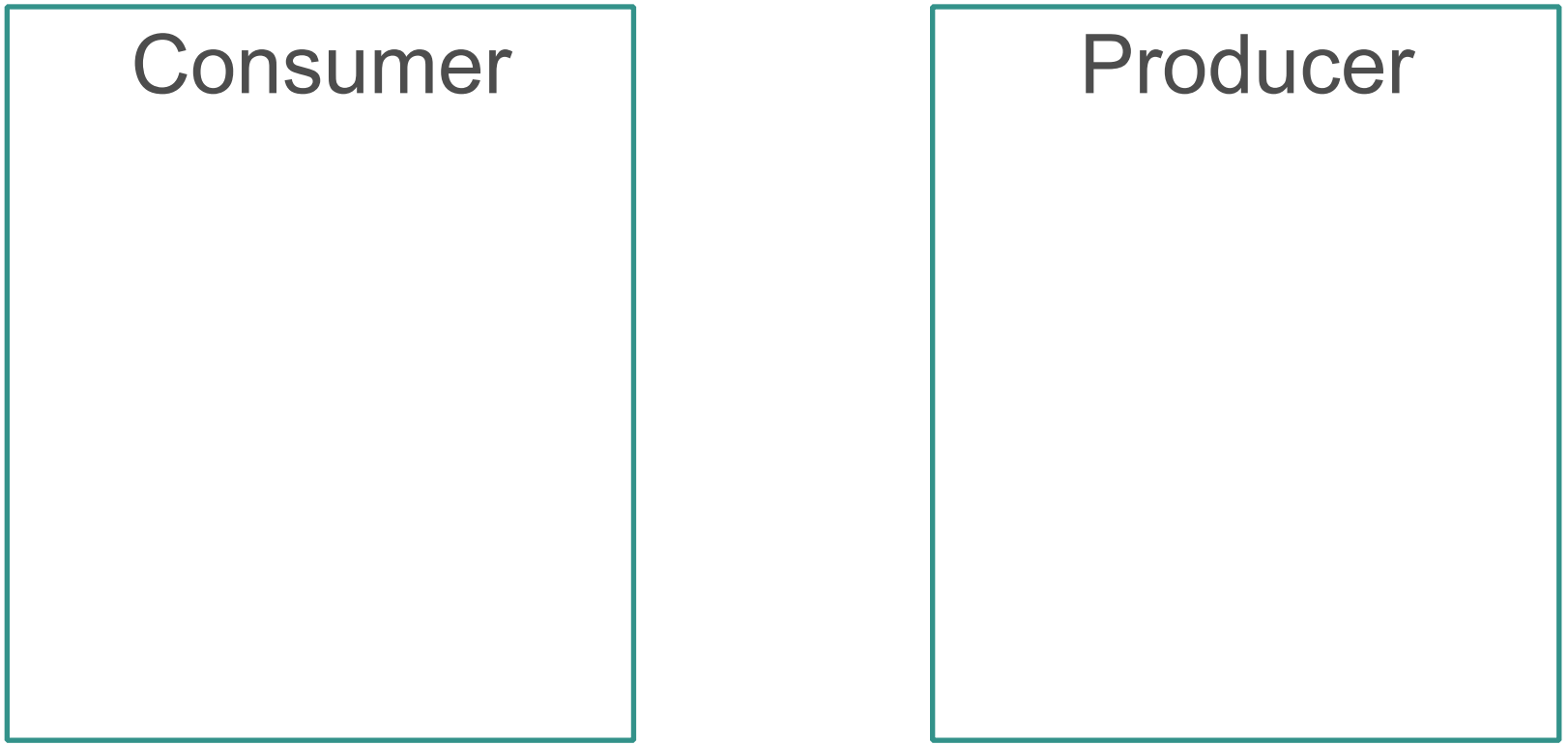
Consumer Driven Contracts

- **Consumer shares** "expectation" with Producer via **"Contract" (≈ DSL)**
- The contract violation should be detected by generated tests on the producer side.

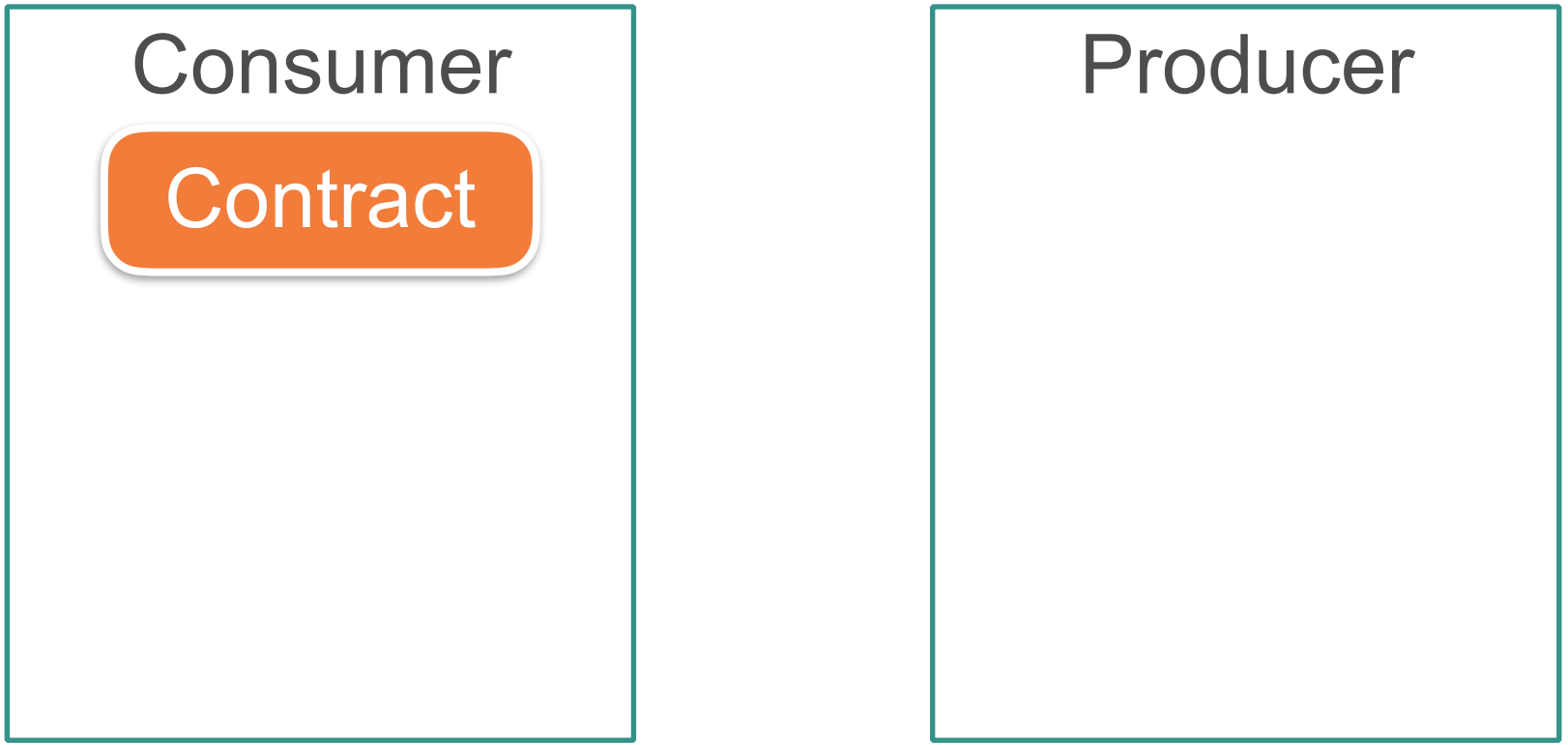
Consumer Driven Contracts

- **Consumer shares** "expectation" with Producer via **"Contract" (≈ DSL)**
- The contract violation should be detected by generated tests on the producer side.

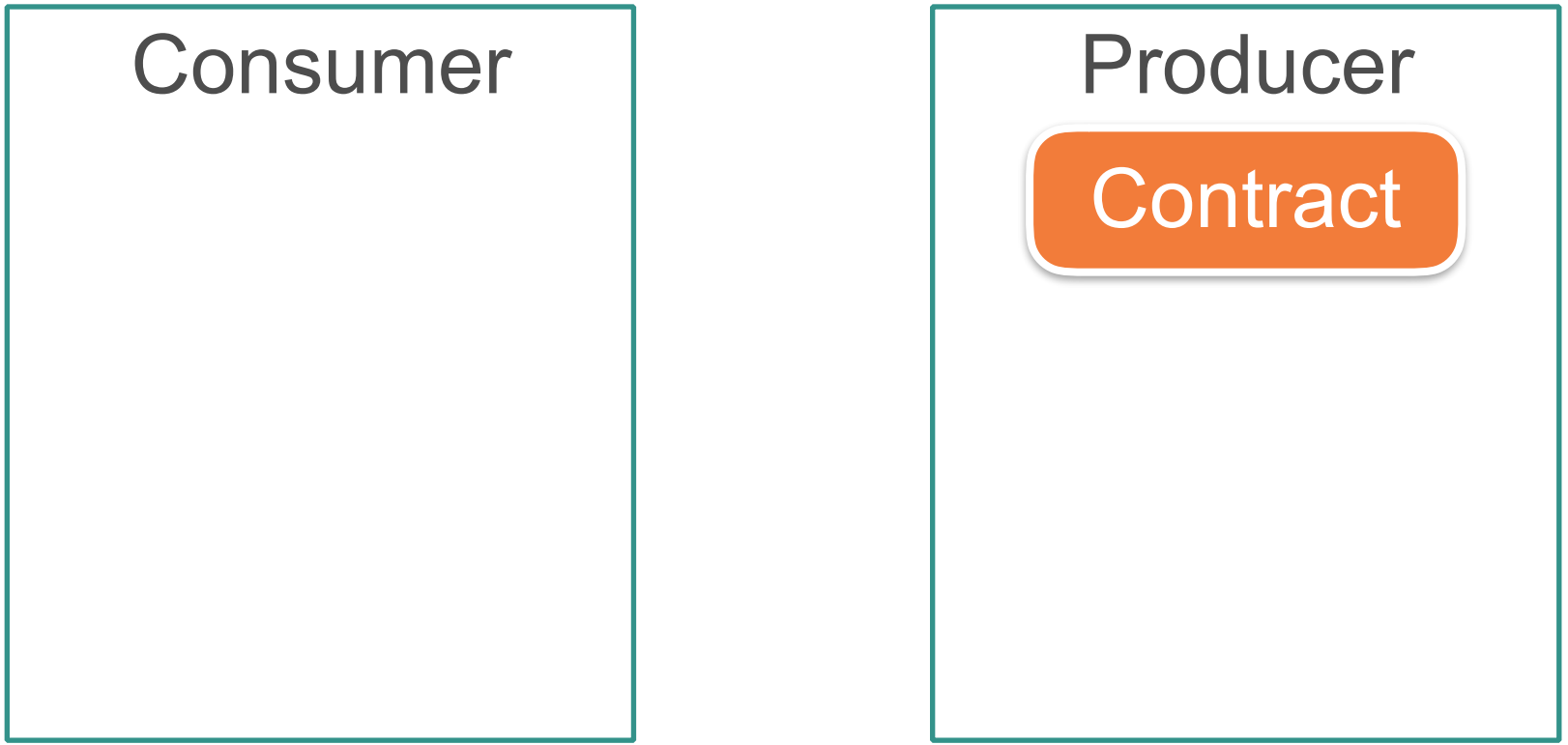
Flow of Consumer Driven Contracts



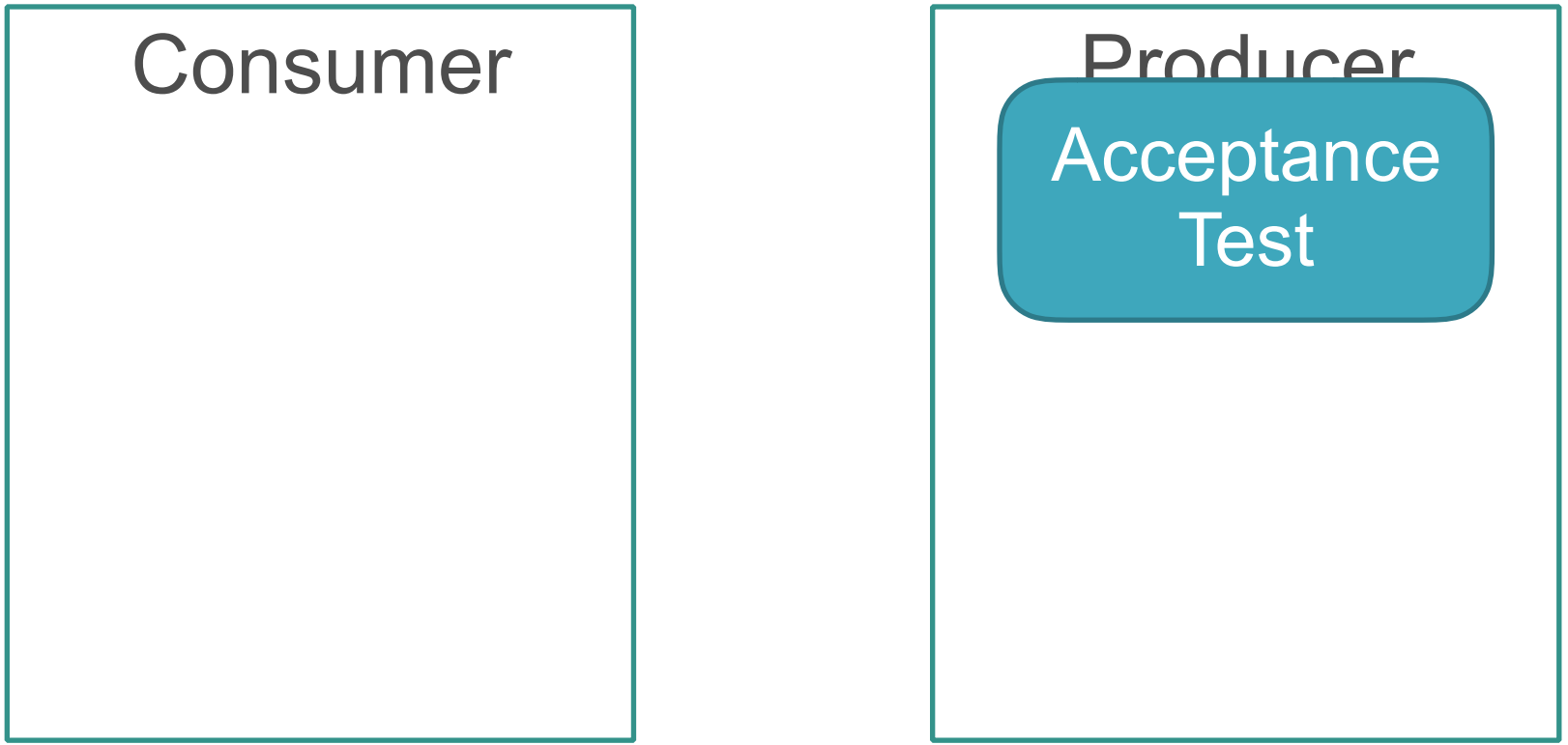
Flow of Consumer Driven Contracts



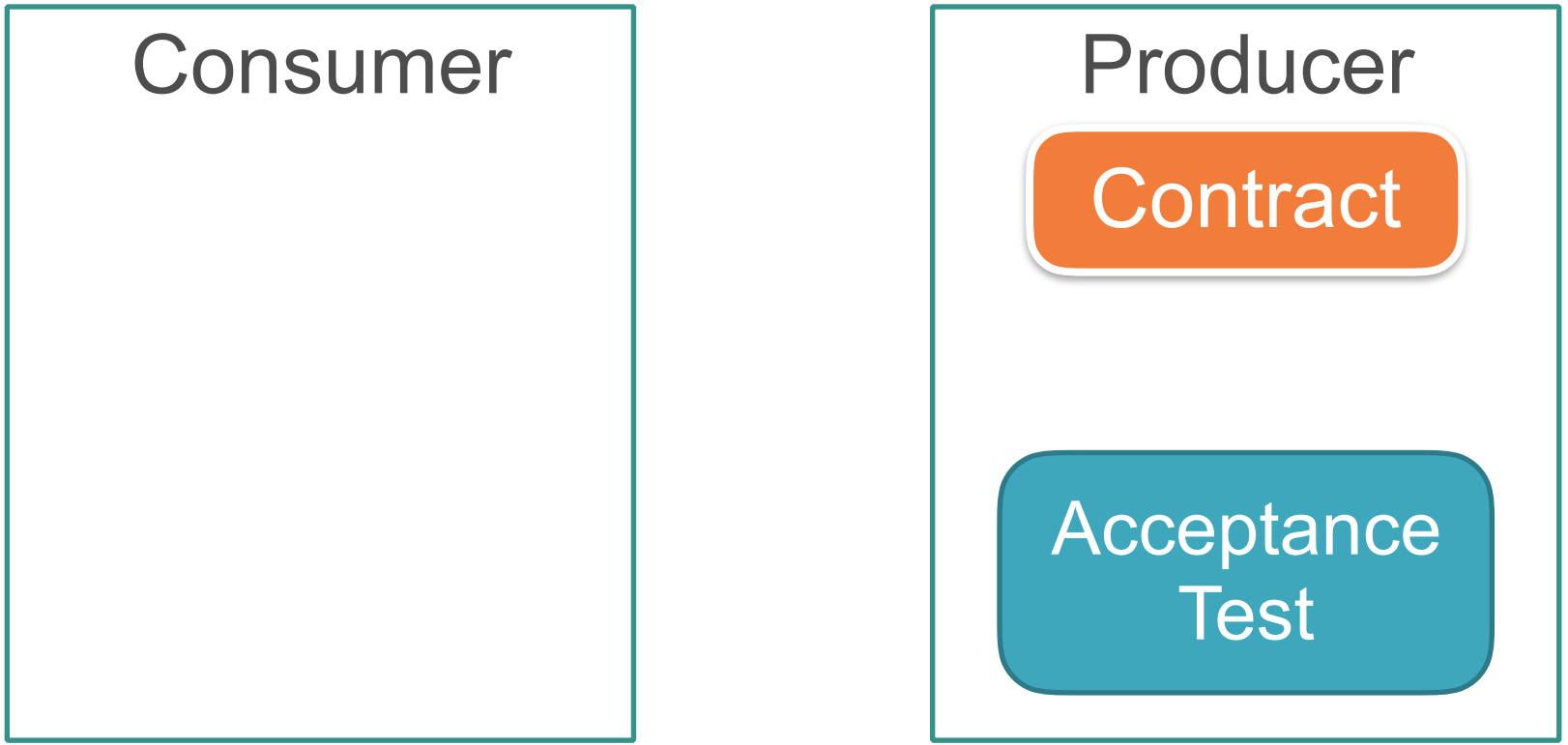
Flow of Consumer Driven Contracts



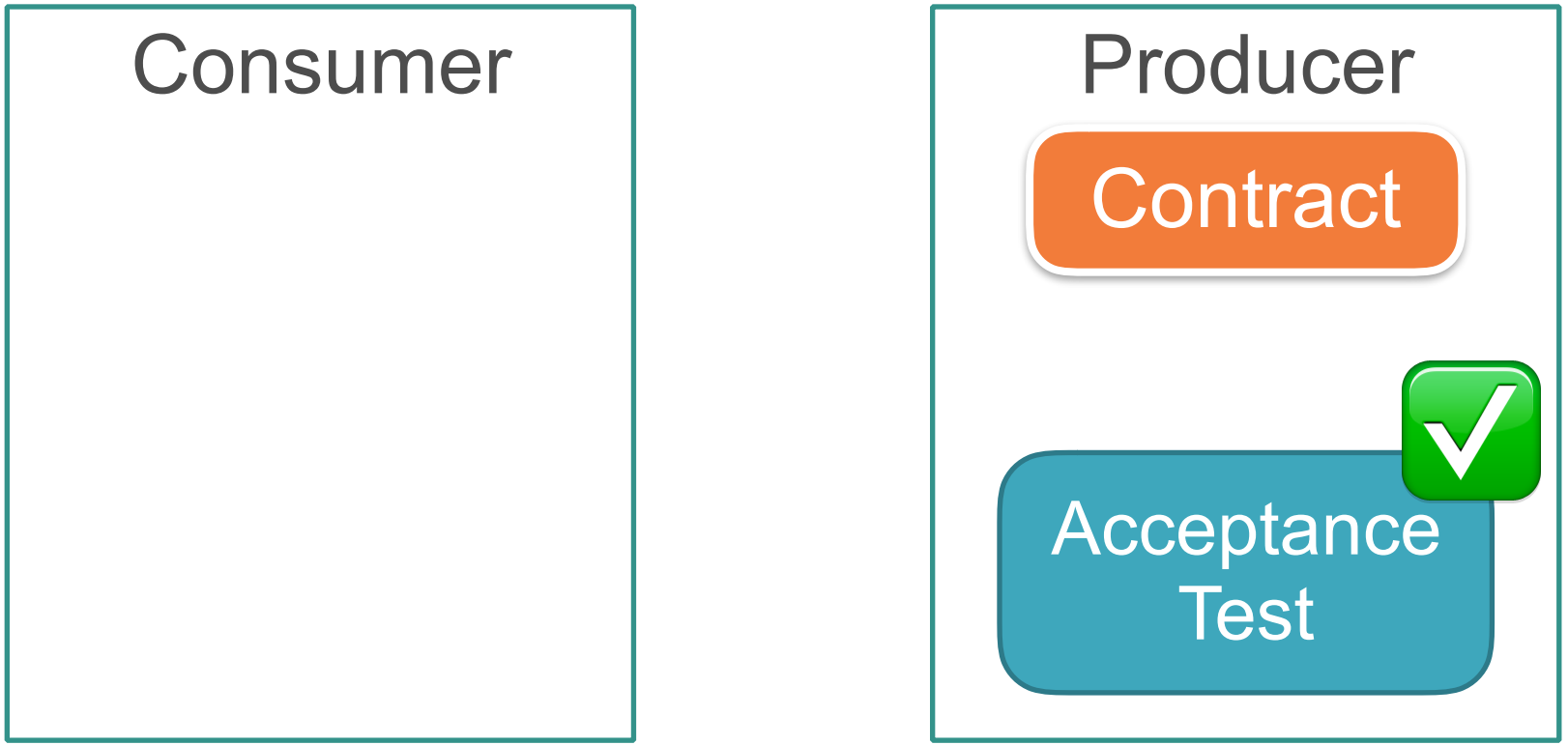
Flow of Consumer Driven Contracts



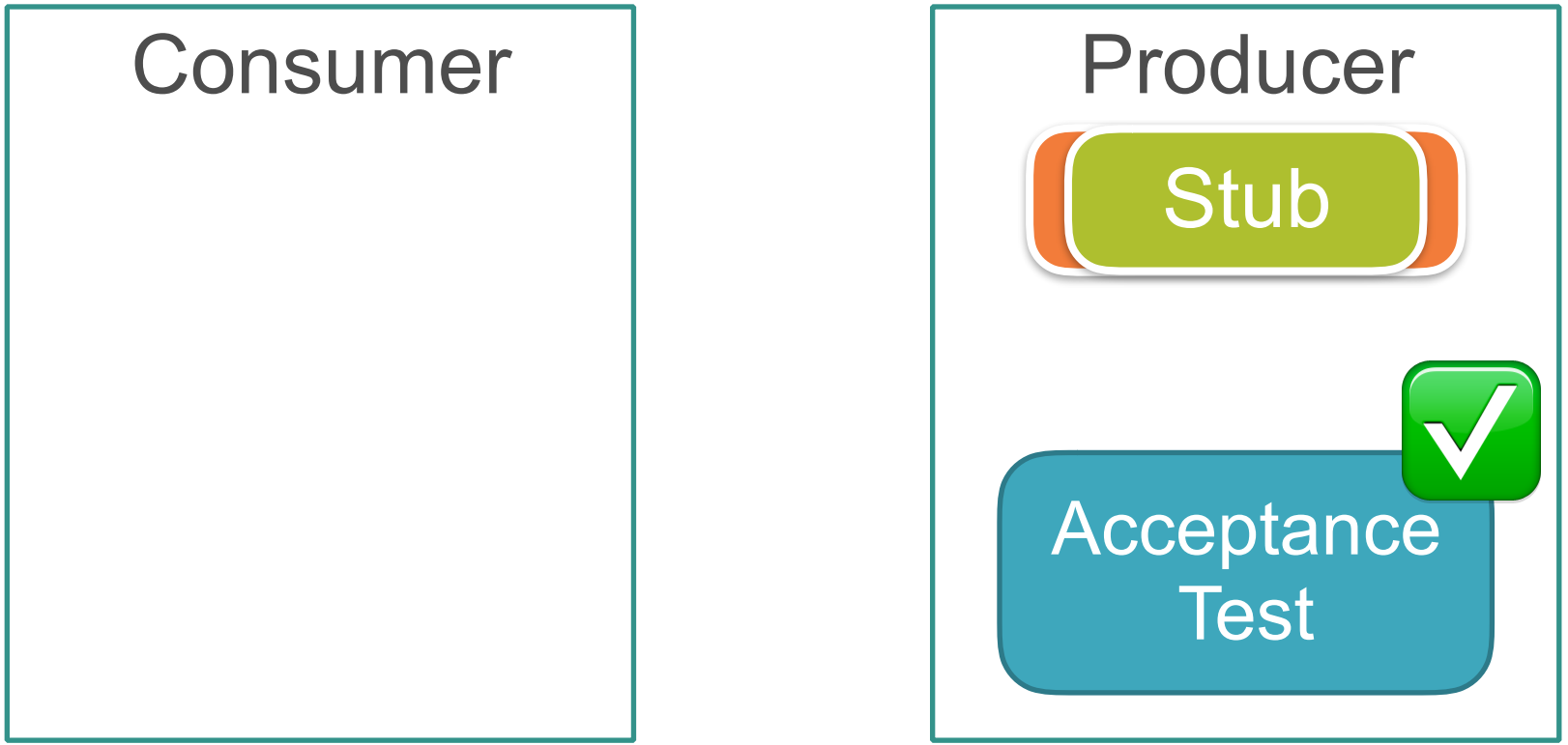
Flow of Consumer Driven Contracts



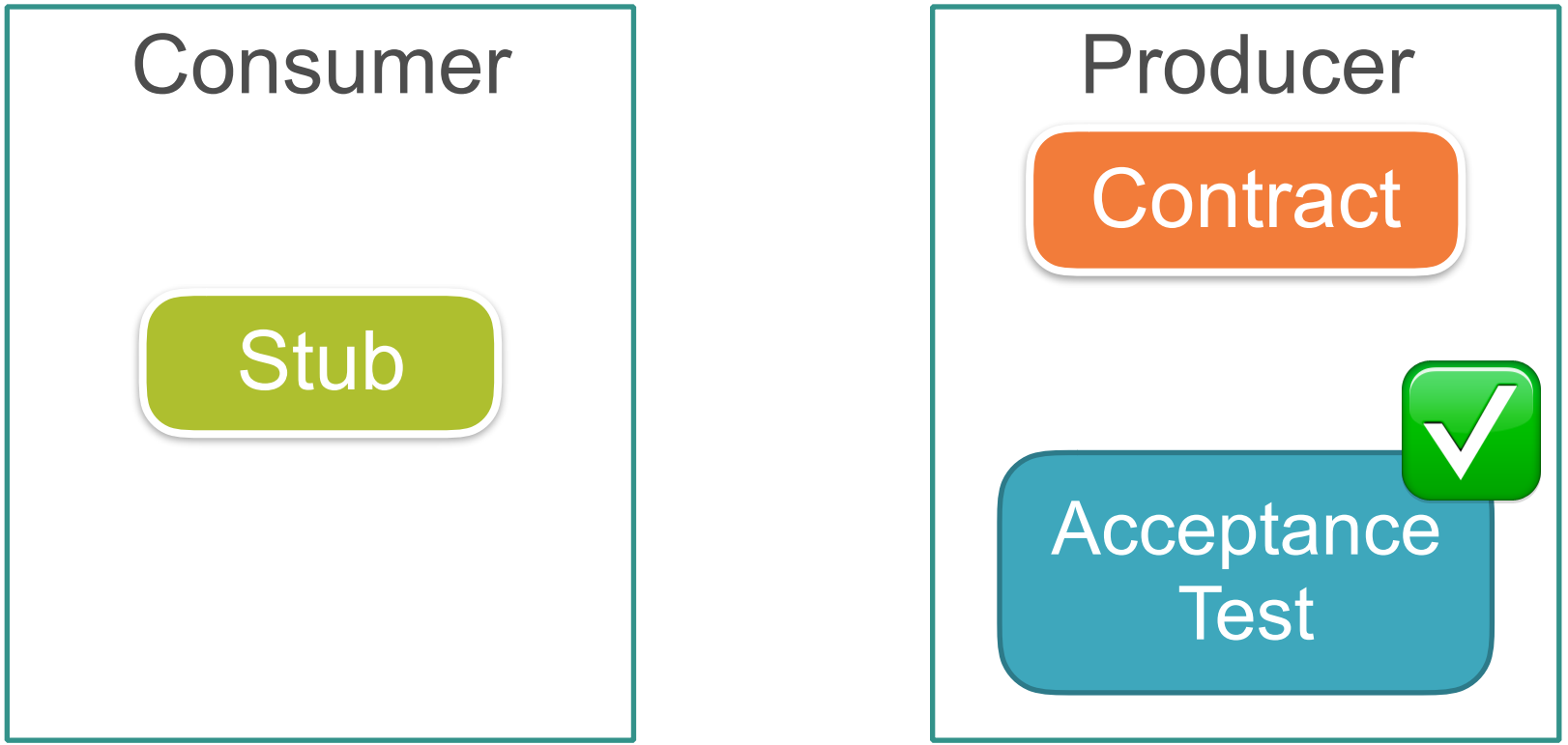
Flow of Consumer Driven Contracts



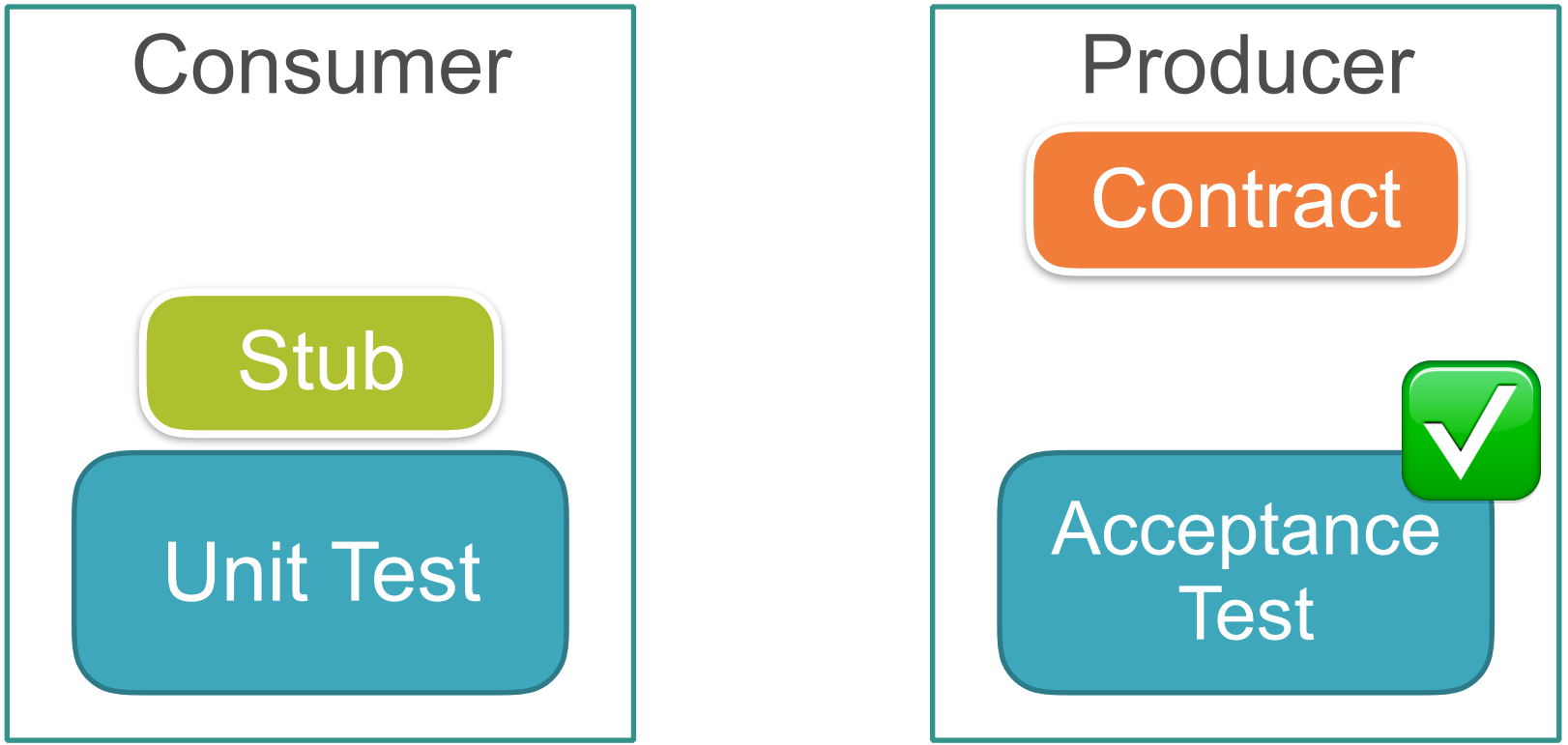
Flow of Consumer Driven Contracts



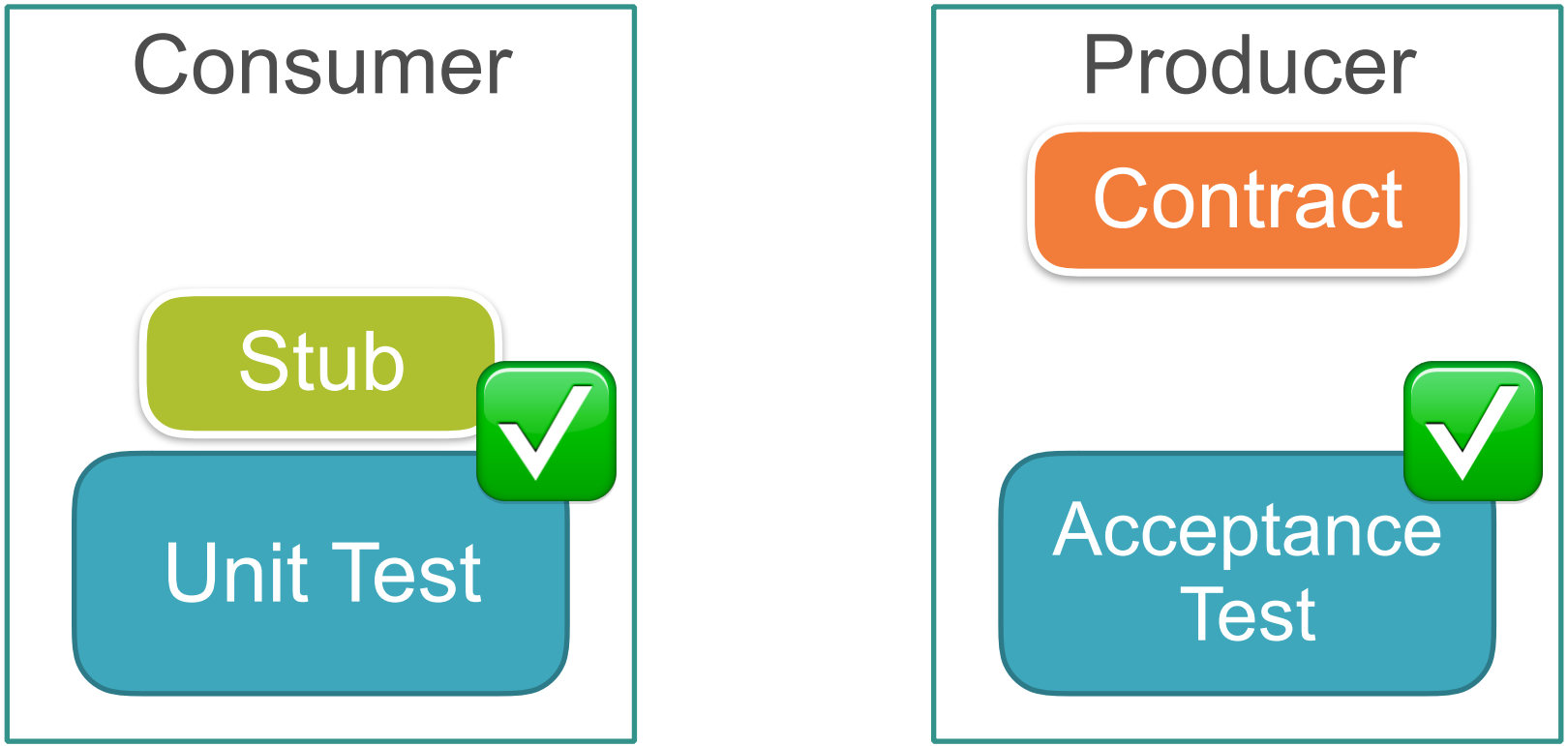
Flow of Consumer Driven Contracts



Flow of Consumer Driven Contracts



Flow of Consumer Driven Contracts



Spring Cloud Contract

- A CDC Solution for JVM apps (especially Spring)
- Contract **DSL using Groovy**
- Generates Acceptance test (JUnit or Spock) for producer
- Generates Stub for consumer
  - WireMock Support for REST Test
  - Messaging Support (Spring Integration, **Spring Cloud Stream** and Apache Camel)

Contract DSL

shouldCreateCustomer.groovy

```
Contract.make {
  label 'create-customer'
  input {

  }
  outputMessage {
    sentTo('demo-strm')
    headers({header('Content-Type':'...')})
    body('''{"name":"@making"}''')
  }}
```

Contract DSL

shouldCreateCustomer.groovy

```
Contract.make {
  label 'create-customer'
  input {

  }
  outputMessage {
    sentTo('demo-strm')
    headers({header('Content-Type':'...')})
    body('''{"name":"@making"}''')
  }}
```

Created by Consumer

Prepare Parent Test Class

```
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment = WebEnvironment.NONE)
@AutoConfigureMessageVerifier
public abstract class MsgTestBase {
  @Autowired CustomerService service;

  protected void create() {
    service.create("@making");
  }
}
```

Prepare Parent Test Class

```
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment = WebEnvironment.NONE)
@AutoConfigureMessageVerifier
public abstract class MsgTestBase {
  @Autowired CustomerService service;

  protected void create() {
    service.create("@making");
  }
}
```

Created by Producer

Contract DSL

shouldCreateCustomer.groovy

```
Contract.make {
  label 'create-customer'
  input {
    triggeredBy('create()')
  }
  outputMessage {
    sentTo('demo-strm')
    headers({header('Content-Type':'...')})
    body('''{"name":"@making"}''')
  }}
```

Contract DSL

shouldCreateCustomer.groovy

```
Contract.make {
  label 'create-customer'
  input {
    triggeredBy('create()')
  }
  outputMessage {
    sentTo('demo-strm')
    headers({header('Content-Type':'...')})
    body('''{"name":"@making"}''')
  }}
```

Updated by Producer

Generated Acceptance Test

```
public class ContractVerifierTest extends MsgTestBase {
  // ...
 @Test public void validate_shouldCreateCustomer() {
    create();
    ContractVerifierMessage res = verifierMessaging
                                        .receive("customer");
    assertThat(res).isNotNull();
    DocumentContext parsedJson = JsonPath.parse(
        objectMapper.writeValueAsString(res.getPayload()));
    assertThatJson(parsedJson).field("name")
        .isEqualTo("@making");
  }}
```

Spring Cloud Contract Maven Plugin

```
mvn spring-cloud-contract:generateTests
```

Acceptance Test

```
mvn spring-cloud-contract:convert
```

WireMock stub file (only for REST)

```
mvn spring-cloud-contract:generateStubs
```

Stub jar file

Consumer Side Test

```
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment = WebEnvironment.NONE)
@AutoConfigureStubRunner(ids = "com.example:customer-
service", workOffline = true)
public class PointServiceConsumerTest {
  @Autowired StubTrigger stubTrigger;
  // ...
  @Test public void testCreateCustomer() {
    stubTrigger.trigger("create-customer");
    // assert that the message is received ...
  }
}
```

Check sample code

- http://bit.ly/making_ccc_a3

(FYI) CQRS and Event Sourcing

- https://spring.io/blog/2016/11/08/cqrs-and-event-sourcing-with-jakub-pilimon
- https://github.com/pilloPl/event-source-cqrs-sample

Data Microservices with Spring Cloud Data Flow

Pivotal

Data Microservices

```
$ cat book.txt | tr ' ' '¥ ' | tr '[:upper:]' '[:lower:]' |
      tr -d '[:punct:]' |
      grep -v '[^a-z]`  |
      sort | uniq -c | sort -rn | head
```

Data Microservices

```
$ cat book.txt | tr ' ' '¥ ' | tr '[:upper:]' '[:lower:]' |
      tr -d '[:punct:]' |
      grep -v '[^a-z]\' |
      sort | uniq -c | sort -rn | head
```

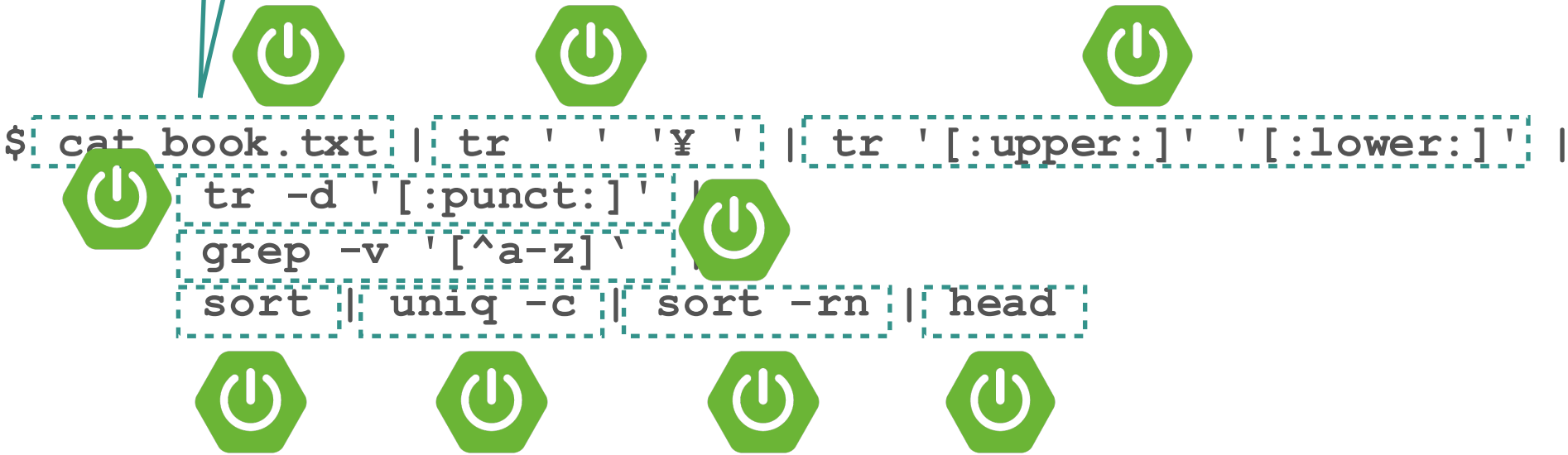
Data

Microservice for each data processing

```
$ cat book.txt | tr ' ' '¥ ' | tr '[:upper:]' '[:lower:]' |
      tr -d '[:punct:]' |
      grep -v '[^a-z]\' |
      sort | uniq -c | sort -rn | head
```

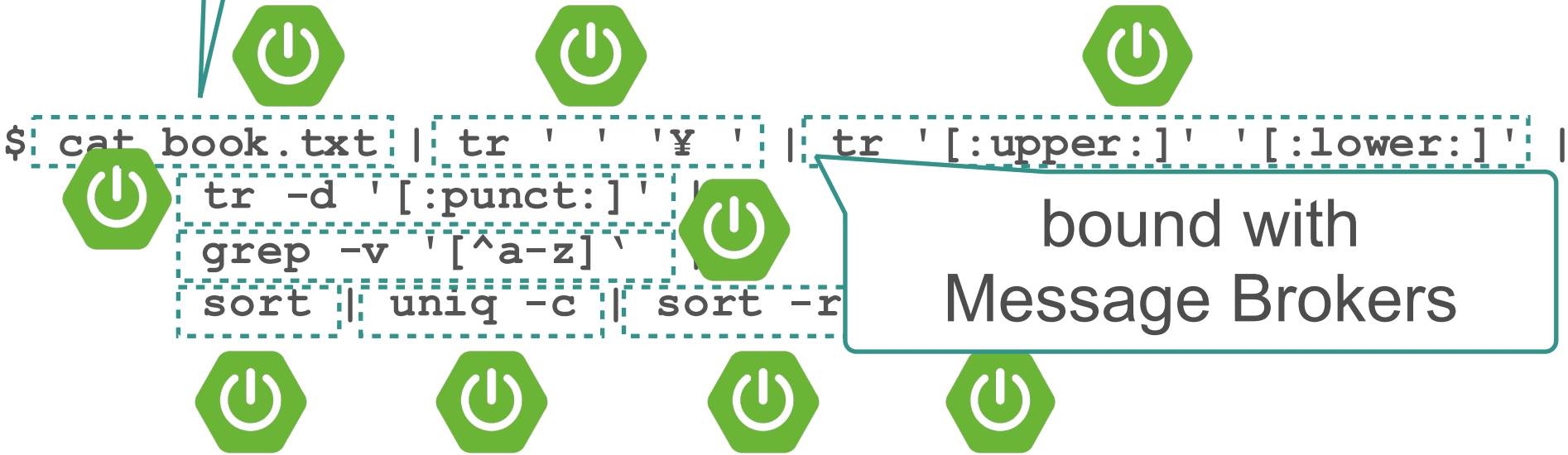
Data

Microservice for each data processing



Data

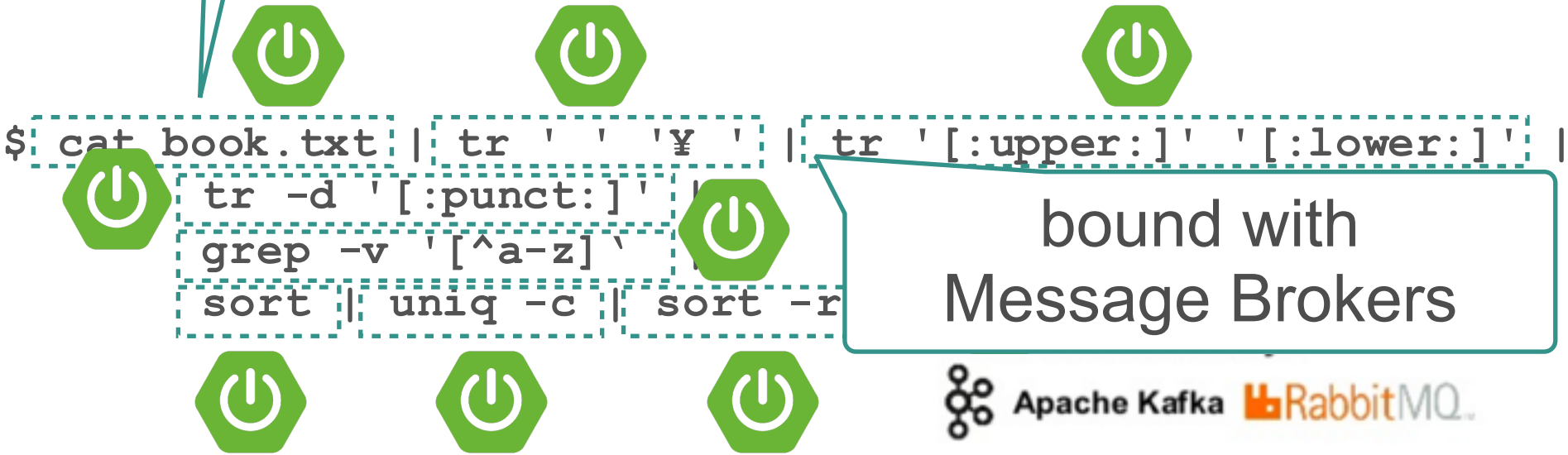
Microservice for each data processing



bound with
Message Brokers

Data

Microservice for each data processing

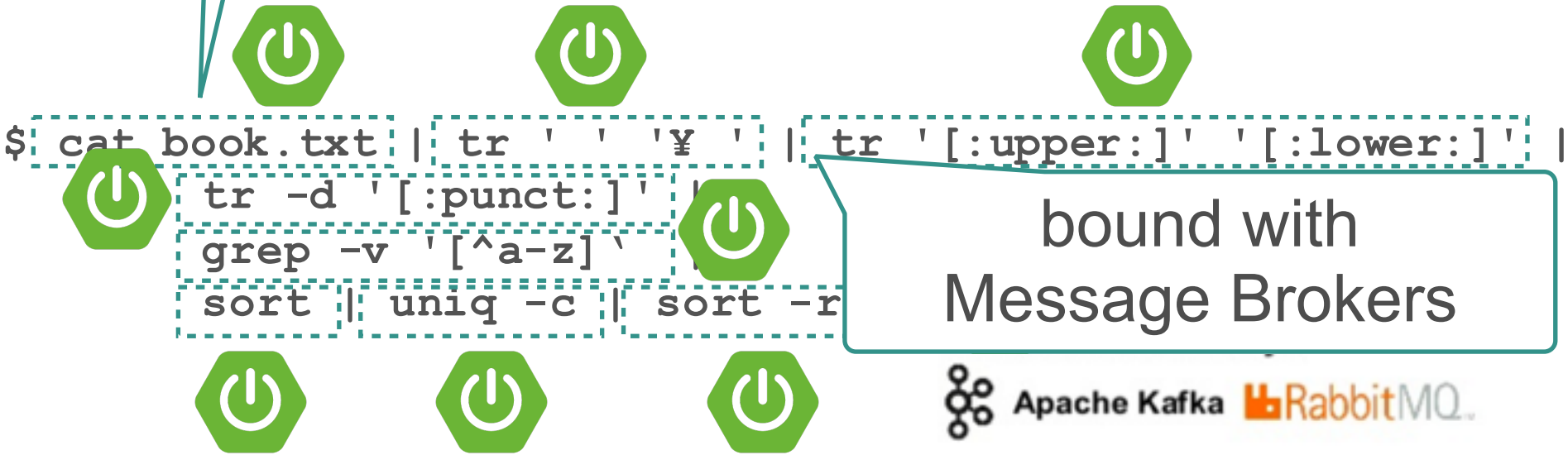


bound with
Message Brokers

Apache Kafka RabbitMQ

Data

Microservice for each data processing



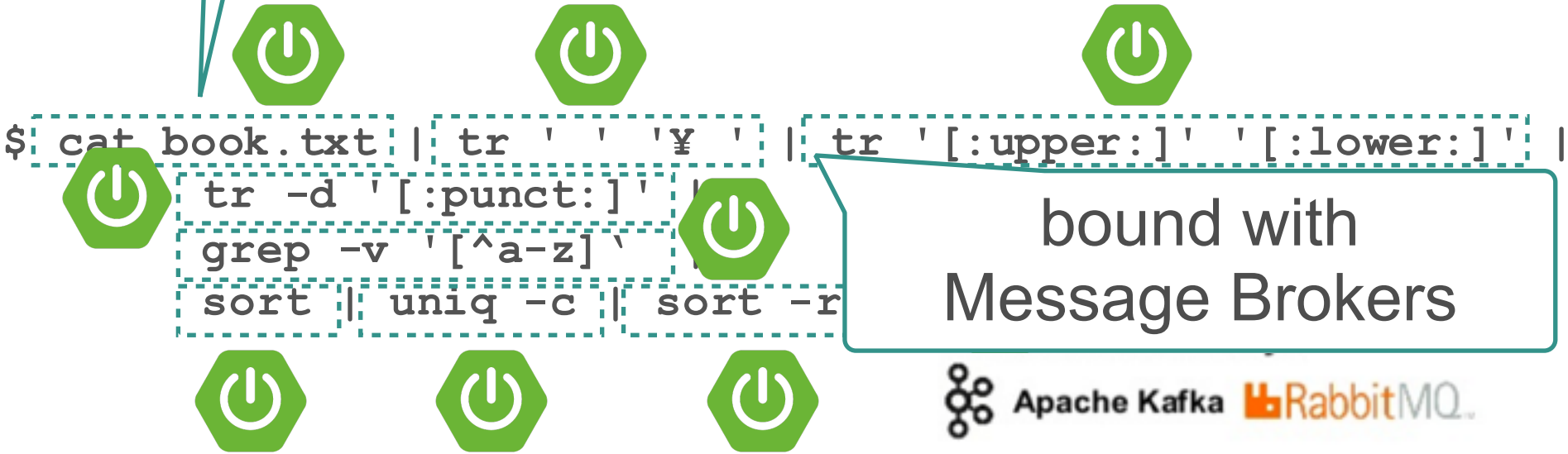
bound with
Message Brokers

Apache Kafka RabbitMQ

on the modern platform such as Cloud Foundry

Data

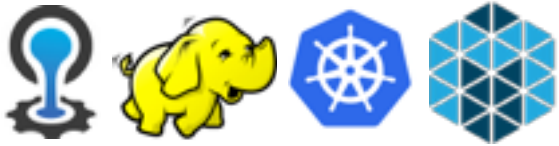
Microservice for each data processing



bound with
Message Brokers

Apache Kafka RabbitMQ

on the modern platform such as Cloud Foundry



Pivotal

Data

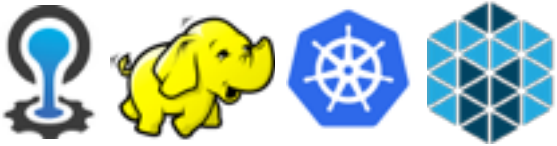
Microservice for each data processing

```
$ cat book.t                                    [:lower:]' |
  tr -
  grep
  sort
```

with
Brokers

RabbitMQ

on the modern platform such as Cloud Foundry



Pivotal

Spring Cloud Data Flow

- Microservices-based Distributed Data Pipelines
  - Long Lived Stream Applications
  - Short Lived Task Applications

Spring Cloud Data Flow

- Microservices-based Distributed Data Pipelines
  - Long Lived Stream Applications

Spring Cloud Stream

  - Short Lived Task Applications

Spring Cloud Data Flow

- Microservices-based Distributed Data Pipelines
  - Long Lived Stream Applications

Spring Cloud Stream

  - Short Lived Task Applications

Spring Cloud Task

Spring Cloud Data Flow

Orchestration Layer

- Microservices-based Distributed Data Pipelines
  - Long Lived Stream Applications

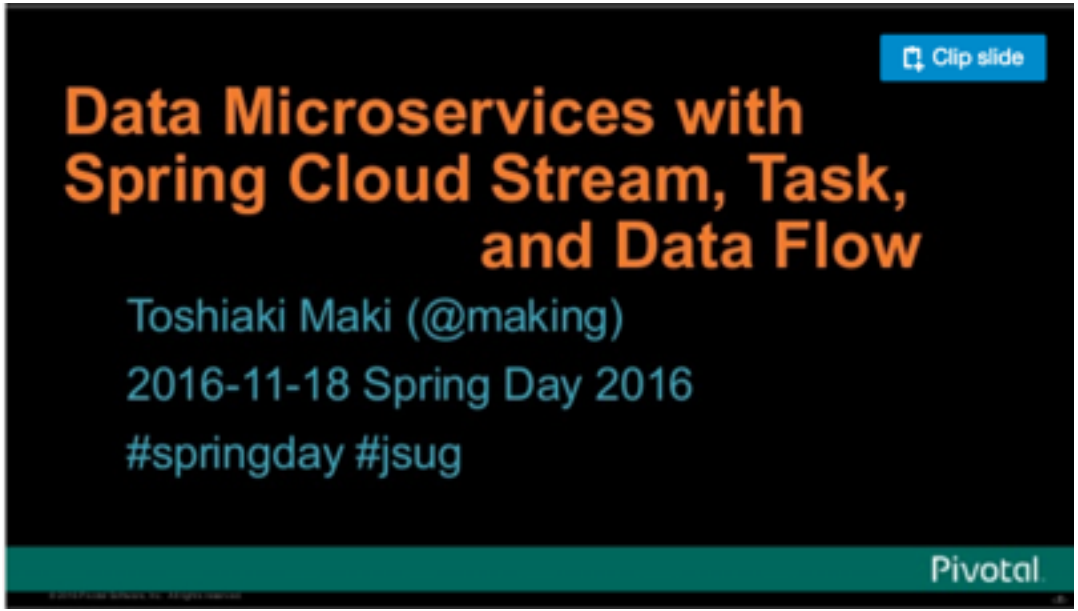
Spring Cloud Stream

  - Short Lived Task Applications

Spring Cloud Task

Check my slide

- http://www.slideshare.net/makingx/data-microservices-with-spring-cloud-stream-task-and-data-flow-jsug-springday



Deploy Stream Apps to Cloud Foundry

Pivotal

Cloud Foundry

- https://www.cloudfoundry.org/
- Cloud Native Platform
- OSS
- Spring ❤ Cloud Foundry
- Support Multi IaaS
  (AWS, Azure, GCP, vSphere, OpenStack)

Cloud Foundry everywhere

Development

Public Cloud Foundry

Private Cloud Foundry

OSS ver

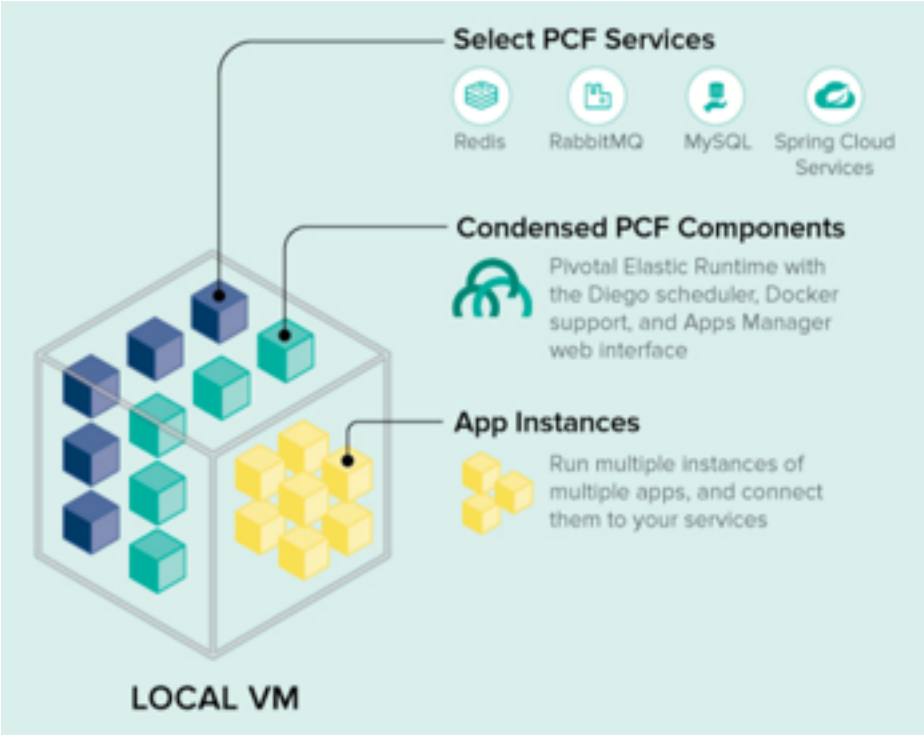
Pivotal
Cloud Foundry

on
Premise

on
Public Cloud

PCF Dev

- https://docs.pivotal.io/pcf-dev
- Cloud Foundry on your laptop
- Included
  - Redis / RabbitMQ / MySQL
  - Spring Cloud Services
- Install with `cf dev start`



Pivotal Web Services

Pivotal Web Services

- https://run.pivotal.io/
- Public Cloud Foundry managed by Pivotal
- $0.03/GB-hr (≈ ¥2200/GB-month)
- $87 of free trial credit.

Deploy Spring Cloud Stream Apps

```
# in case of PCF Dev
cf create-service p-rabbitmq standard my-binder
# in case of Pivotal Web Services
cf create-service cloudamqp lemur my-binder
```

```
cf push my-sink my-sink.jar --no-start
cf bind-service my-sink my-binder
cf start my-sink
```

```
cf push my-source my-source.jar --no-start
cf bind-service my-source my-binder
cf start my-source
```

Sample Application

http://bit.ly/making_ccc_a3

RabbitMQ

MySQL

Point Service

MySQL

Frontend

HTTP

Customer Service

Email Service

SendGrid

Post Service

MySQL

Zipkin Server

DLQ Recovery

Pivotal Web Services / PCF Dev

Tutorial

https://github.com/Pivotal-Japan/spring-cloud-stream-tutorial

Thanks!!

- https://cloud.spring.io/spring-cloud-stream/
- https://cloud.spring.io/spring-cloud-dataflow/
- https://cloud.spring.io/spring-cloud-sleuth/
- http://zipkin.io/
- https://cloud.spring.io/spring-cloud-contract/
- https://projects.spring.io/spring-amqp/
- https://run.pivotal.io/