Events Driven Architecture

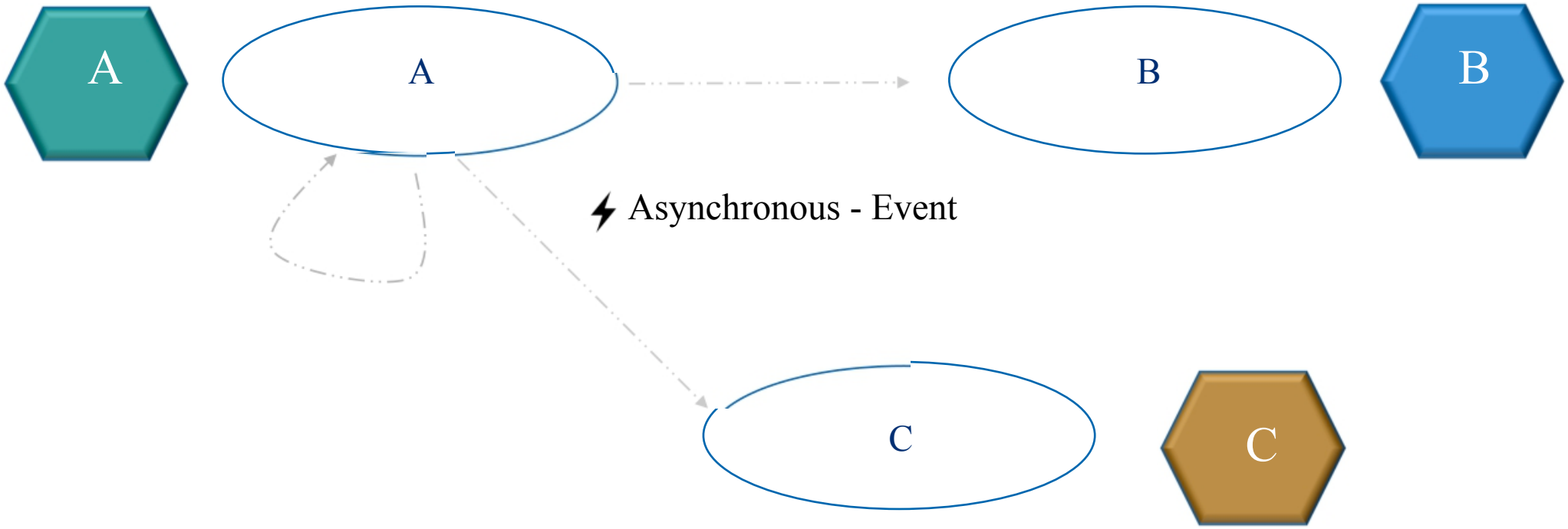
Bounded Contexts

Events Driven Architecture

Microservice Communication

Events Driven Architecture

Events



EDA & Microservices

EDA is an architecture paradigm promoting the production, detection, consumption of, and reaction to events

Microservices are producers and consumers of events

Event Driven Architecture

Messaging & Microservices

Events are Asynchronous and require use of messaging technology for realization

Section objective

1. Microservices Communication patterns
2. Event Driven Architecture
3. Domain & Integration Events
4. AMQP Concepts
5. Case Study : ACME Model Events (UML, JAVA, RabbitMQ)

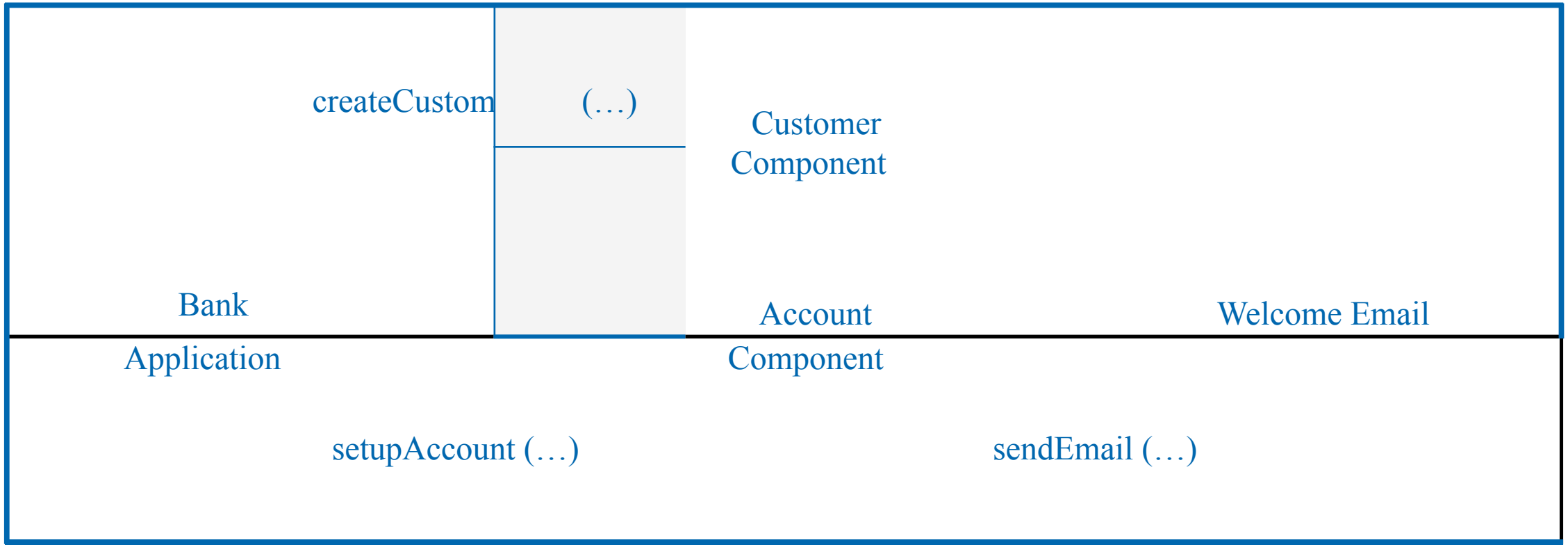
Communication Patterns

Fundamentals

1. Synchronous versus Asynchronous
2. Single Receiver versus Multiple Receivers
3. Technology examples

Monolithic : Object communication

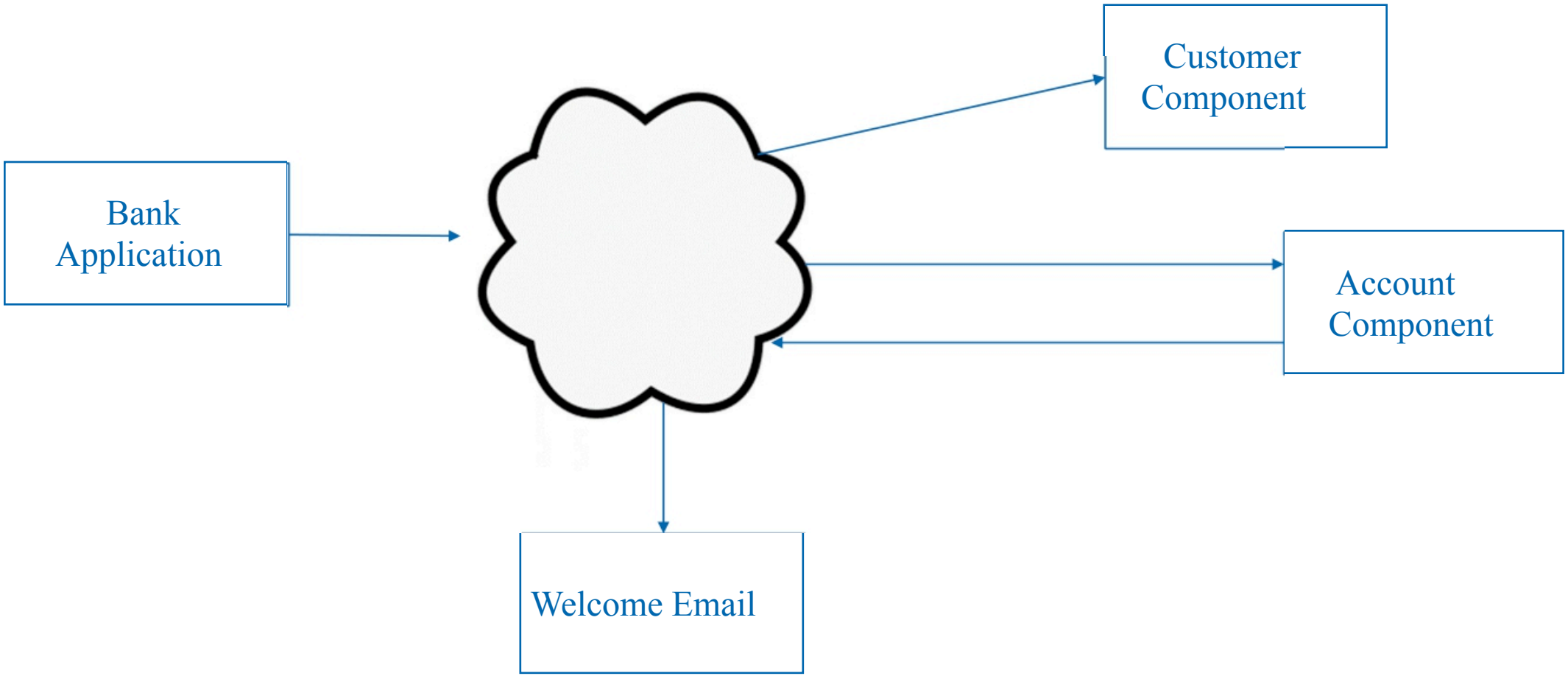
An object calls methods on other objects



Objects share the memory space i.e., same process

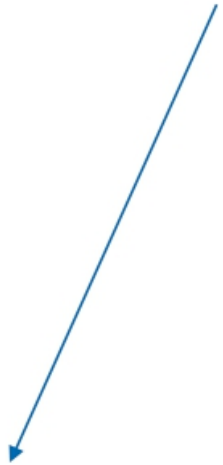
Distributed Applications

Objects communicate using a network protocol



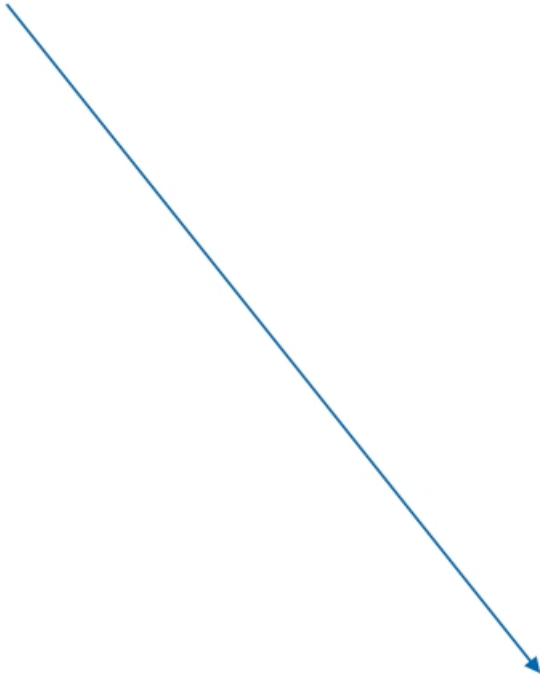
Network Protocols

May be Synchronous or Asynchronous



Caller waits for the response (stays blocked)

- HTTP
- Proprietary RPC

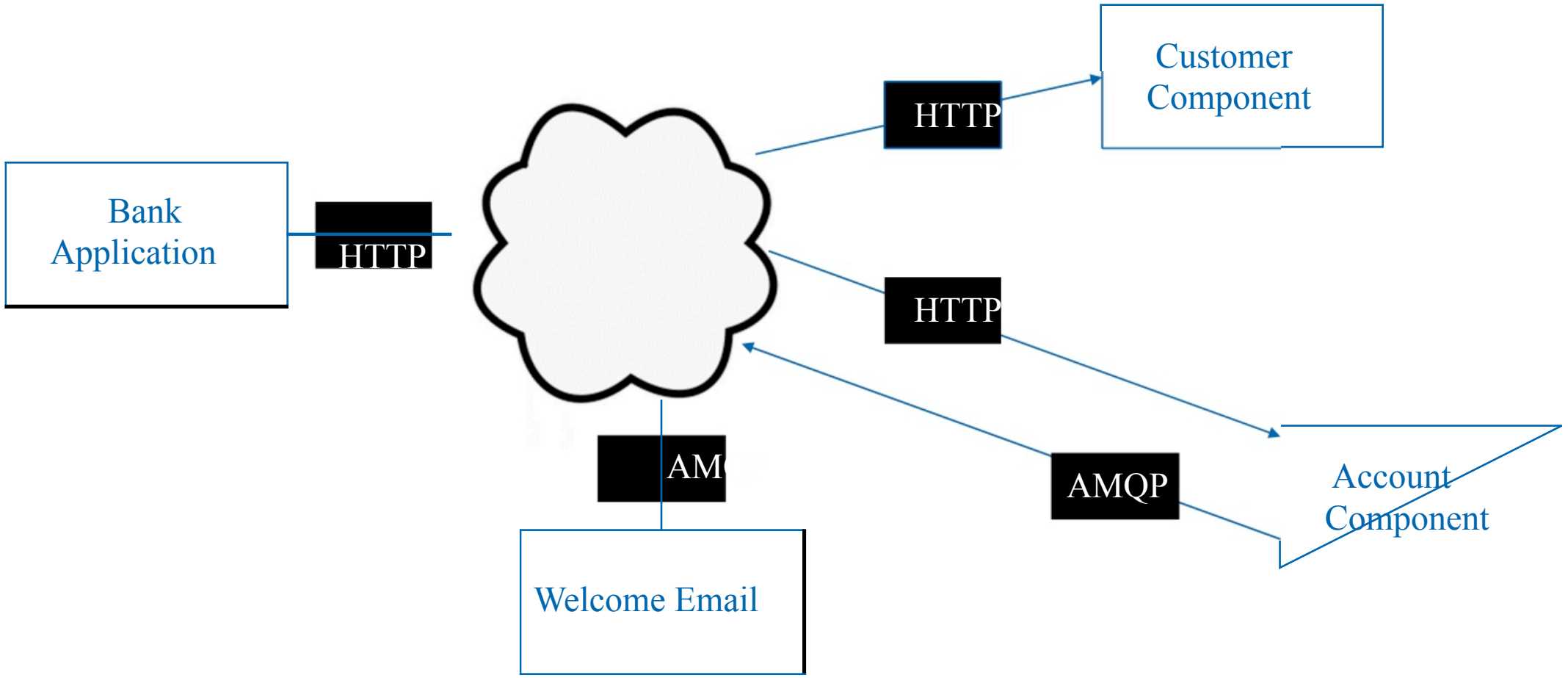


Caller does not wait for response

- Messaging / AMQP

Distributed Applications

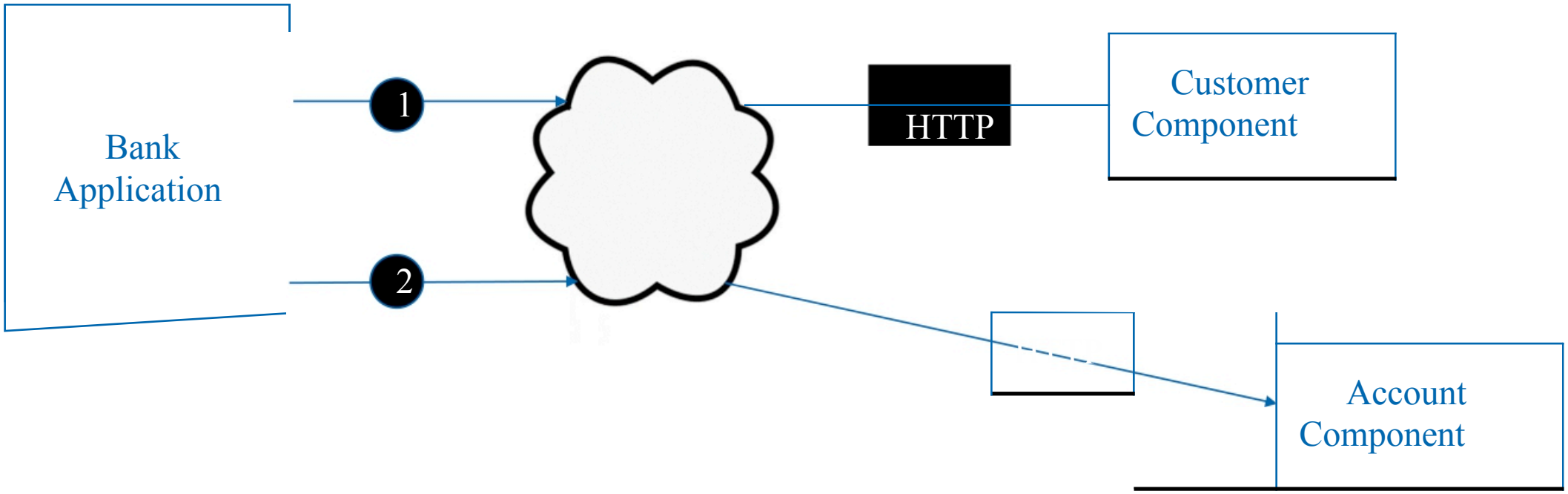
Use of Synch / Asynch is not mutually exclusive



One- to - One Communication (a.k.a. Single Receiver)

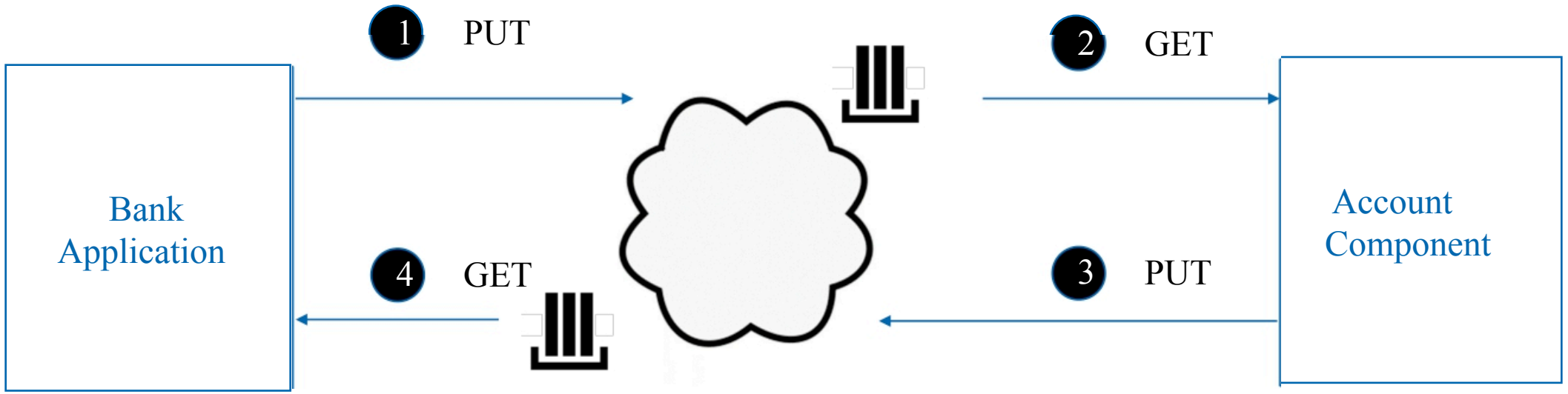
Communication is between two components

- Common to use HTTP for such communication



Single Receiver with Messaging (a.k.a. point to point)

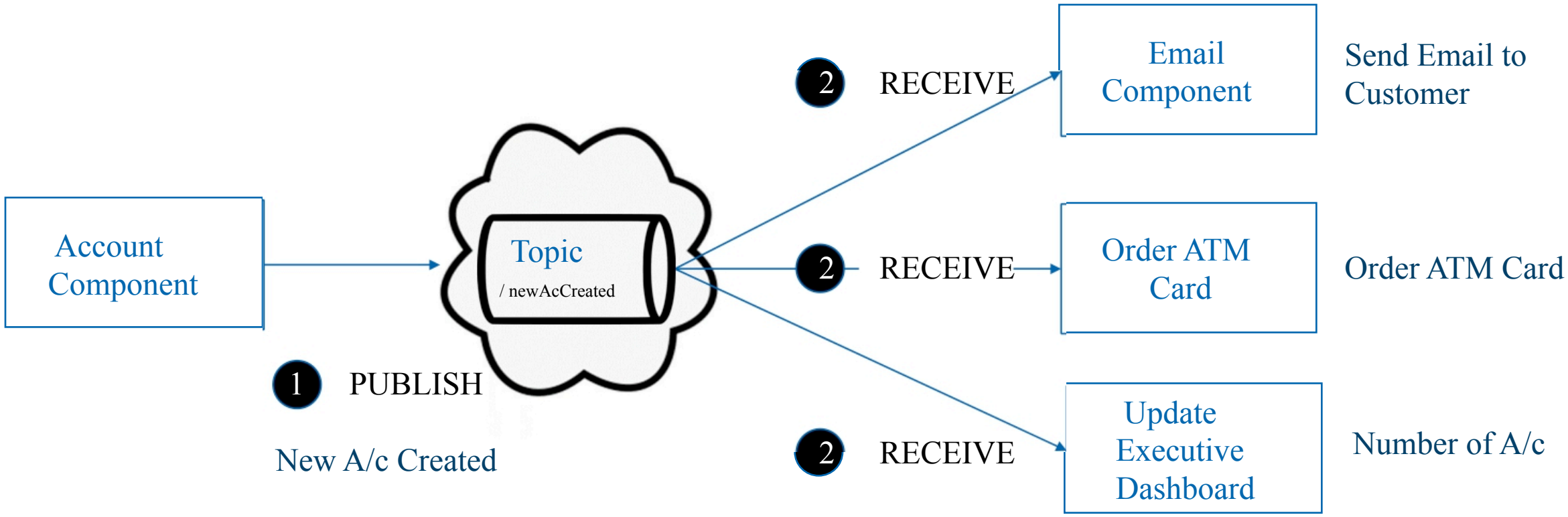
May be built with messaging , using request/response queues



One- to - Many (a.k.a. Multiple Receivers)

Messages are of interest to multiple components

- Pub- Sub messaging pattern



Technology Examples

Synchronous

Asynchronous

Non AMQP compliant products are used as well

Quick Review

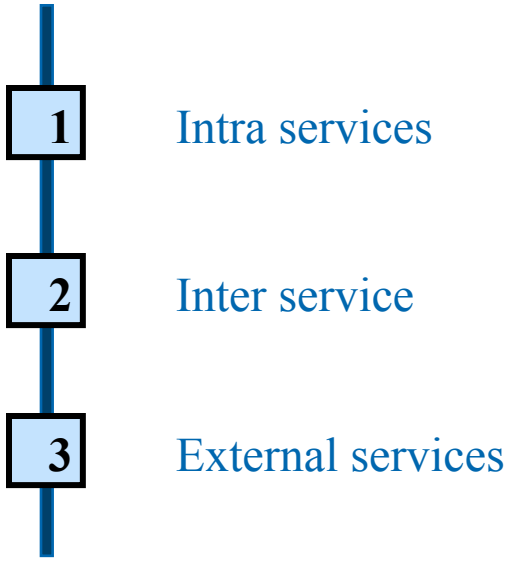
Communication patterns

- Synchronous — REST/HTTP
- Asynchronous — Messaging

Pub- Sub for multiple Receivers

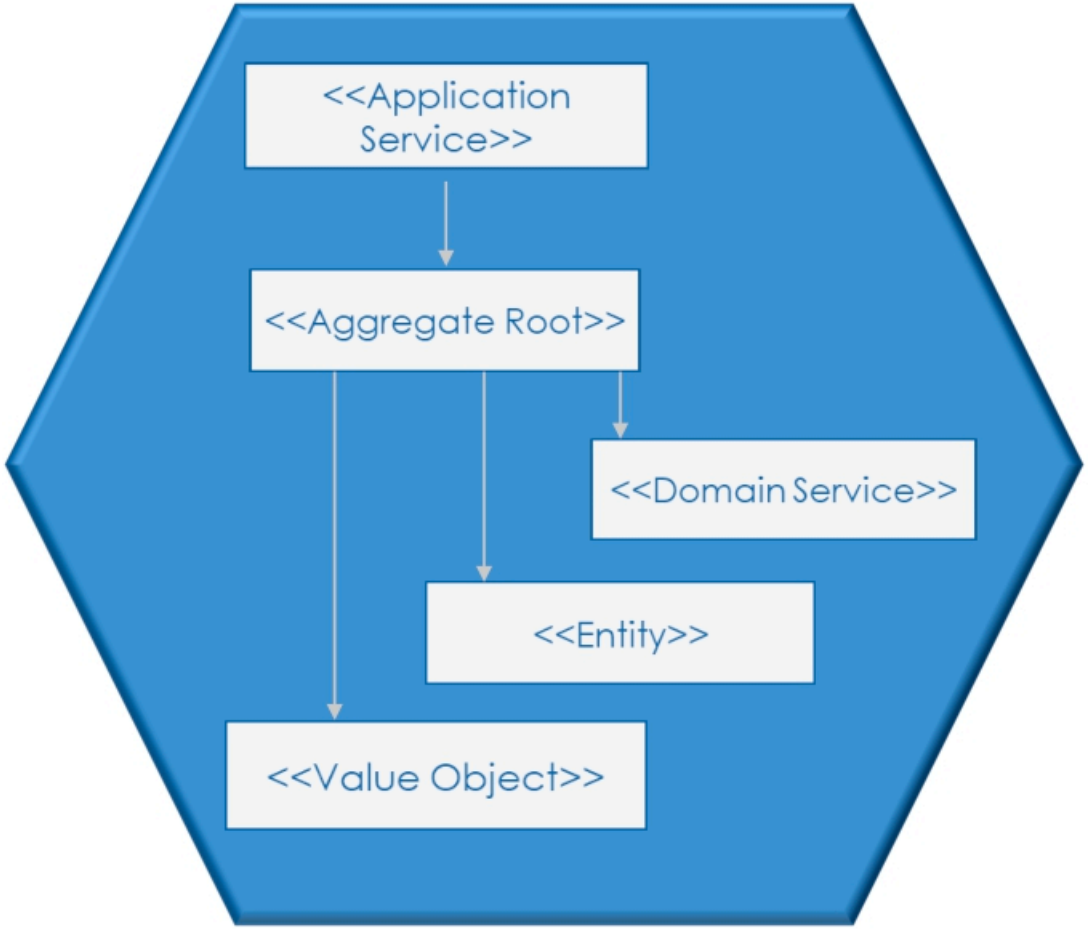
Microservices Interactions

How Microservices use the foundational communication patterns



Intra Service

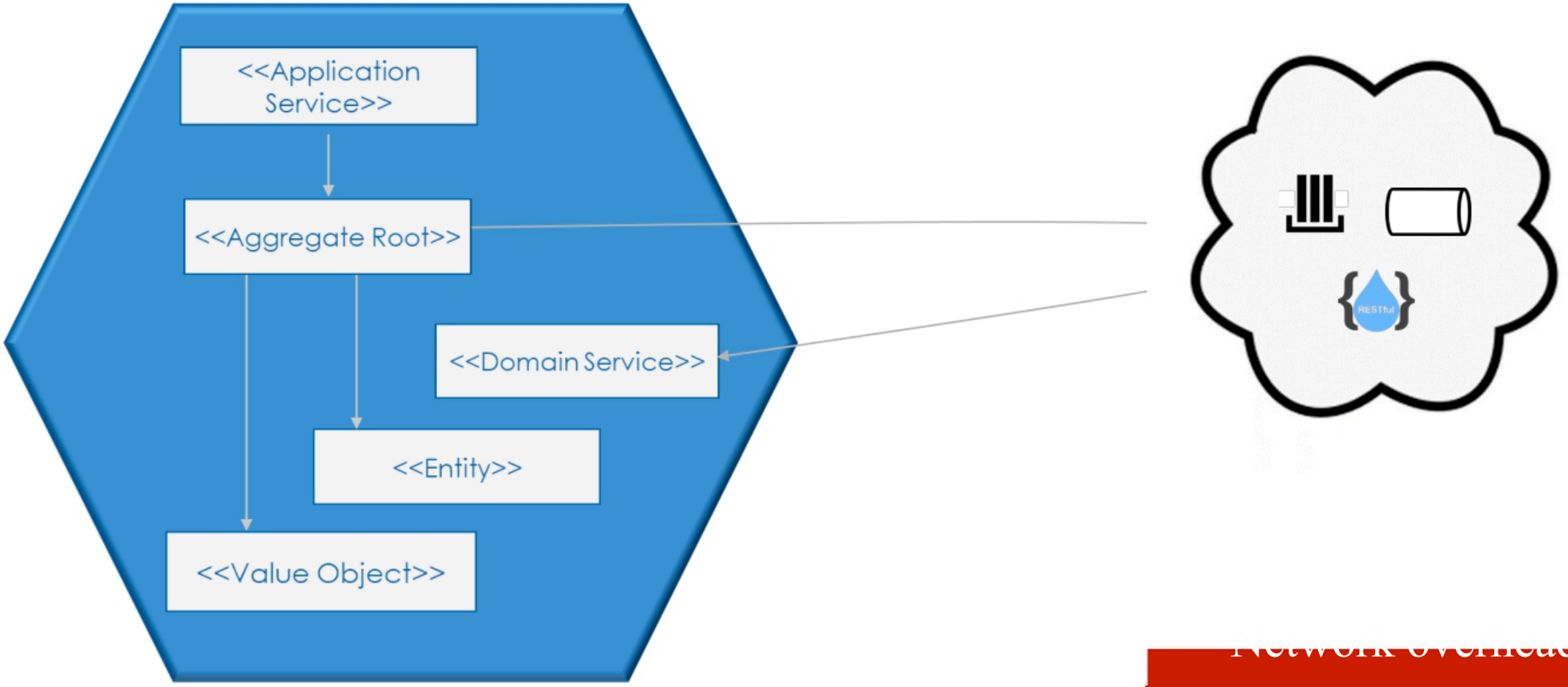
Function calls - within the Bounded Context



- Single deployment unit
- Components within same process

Intra Service

May leverage network protocol for Decoupling | Extensibility

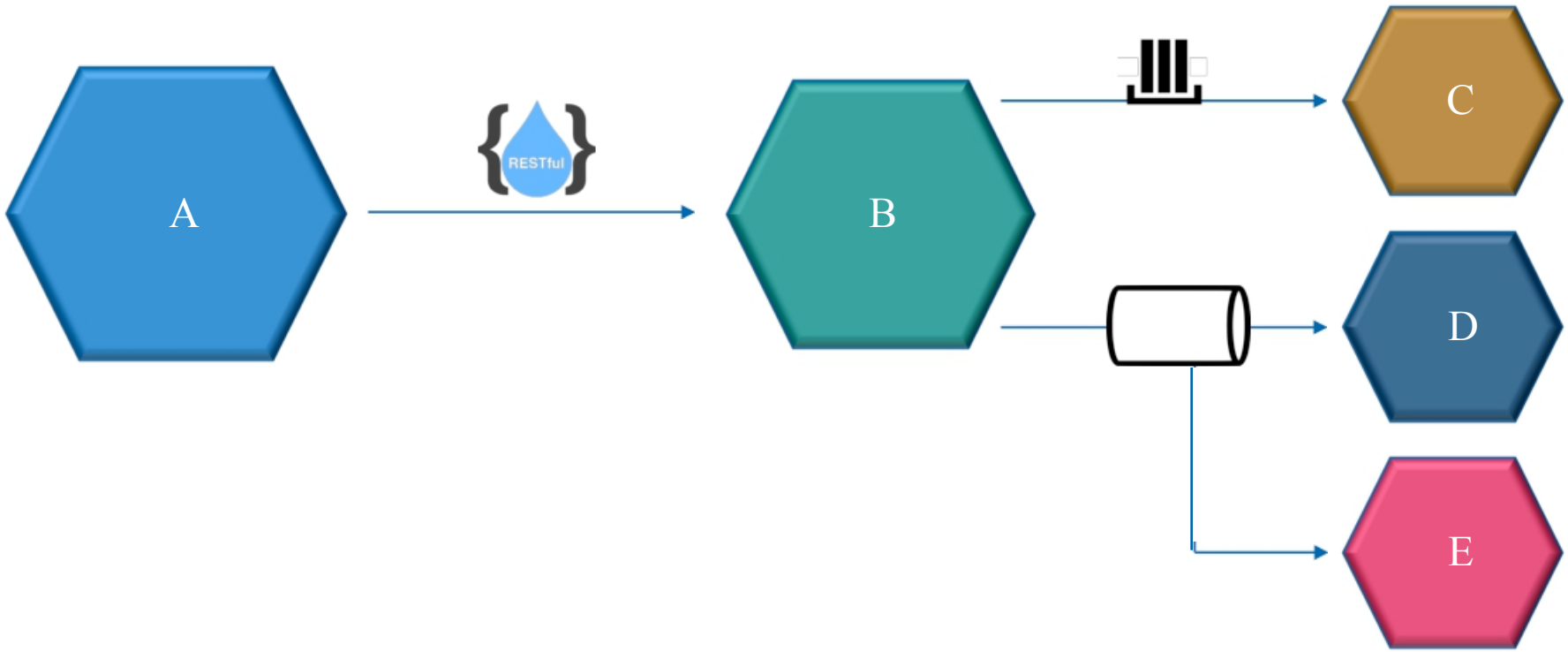


Network overhead

Inter Service

Between different BC - Network protocol **ALWAYS**

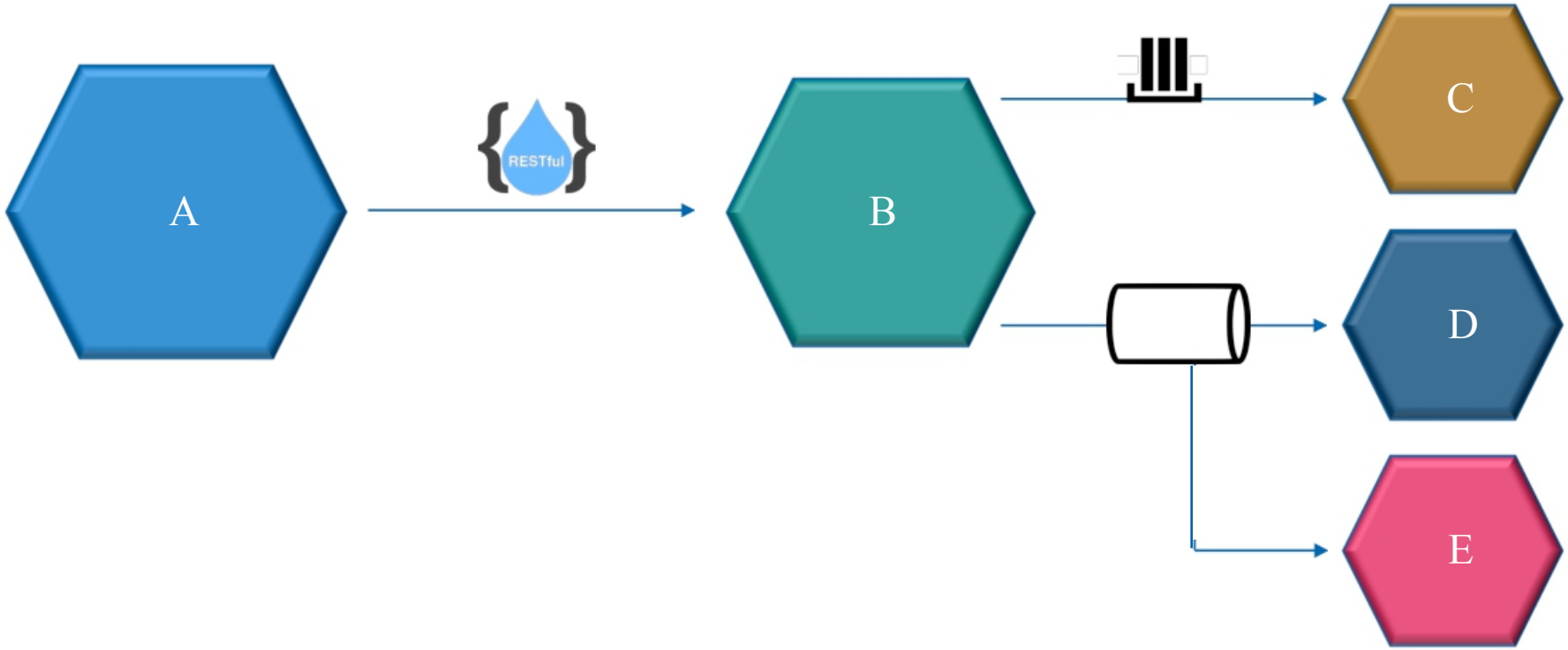
- Messaging or HTTP - depends on the use case



Inter Service

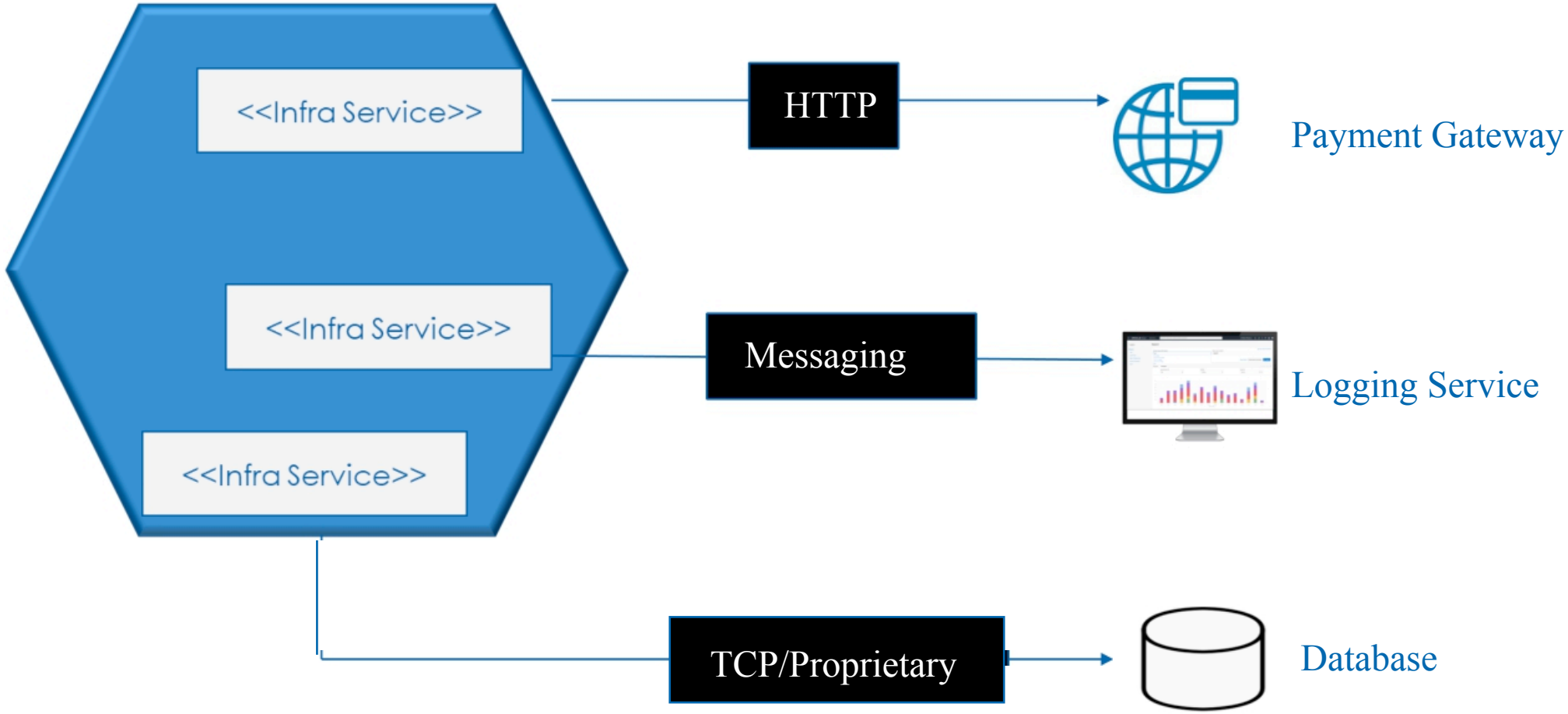
Synchronous | Asynchronous

Single Receiver | Multiple Receivers



External Services

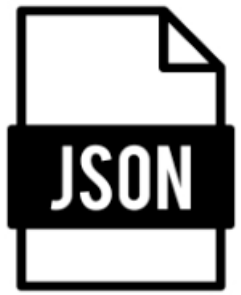
Depends on the external service interface



Message Format & Structure

Structure defined by the model - entities, value objects ...

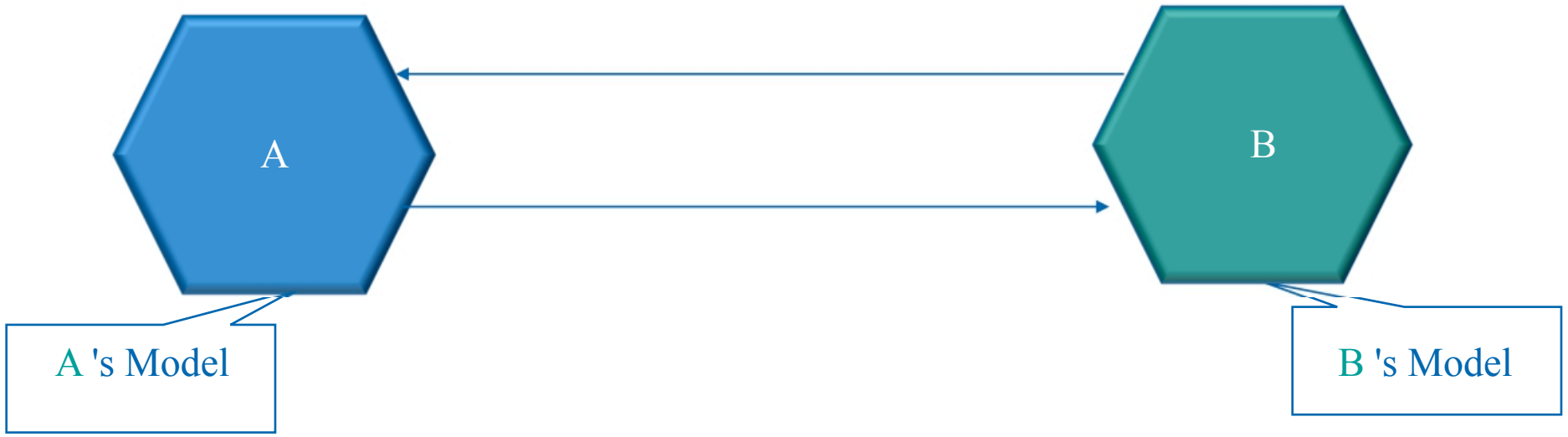
JSON is a popular format



. . . .

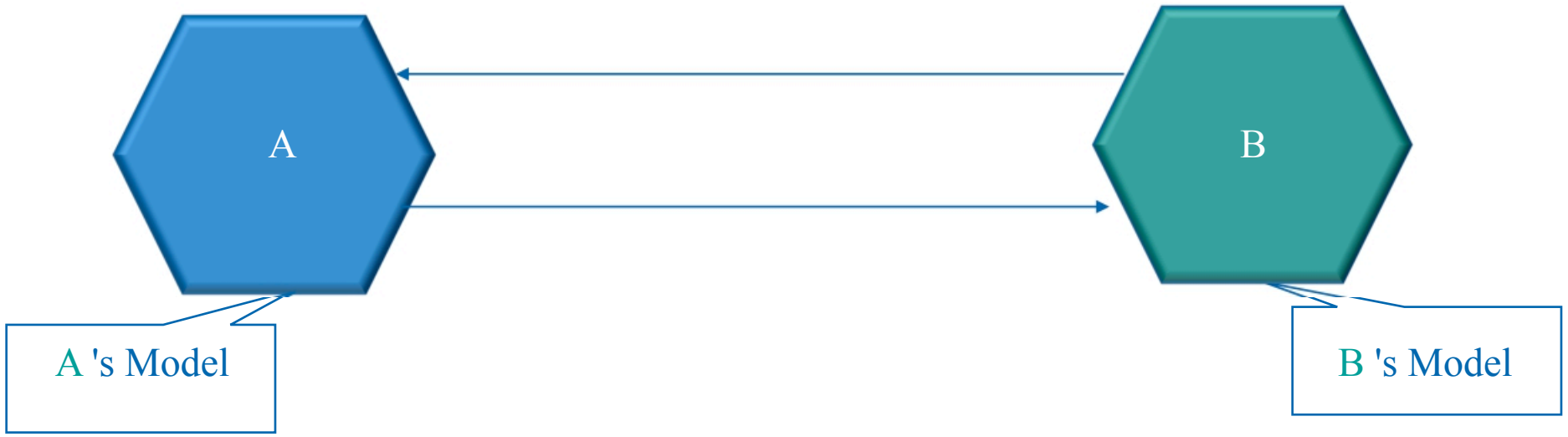
https://developers.google.com/protocol - buffers

Message Format - Inter Service communication



Which model should be used A or B ?

Message Format - Inter Service communication



Depends on how you would like manage dependencies !!!

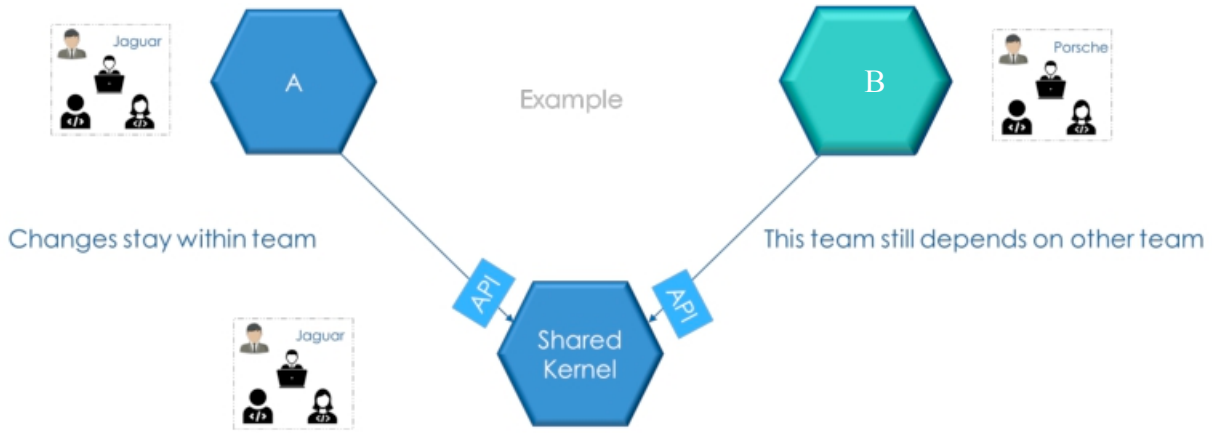
Message Format - Inter Service communication

Refer : Bounded Context Integration Patterns

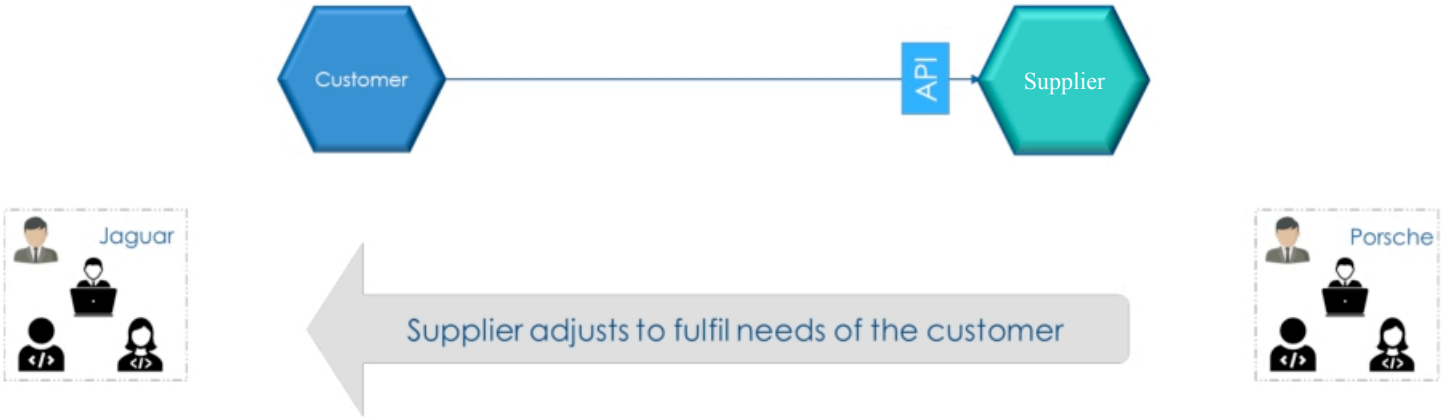
Depends on how you would like manage dependencies !!!

Dependency Management - BC Integration Patterns

Shared kernel pattern

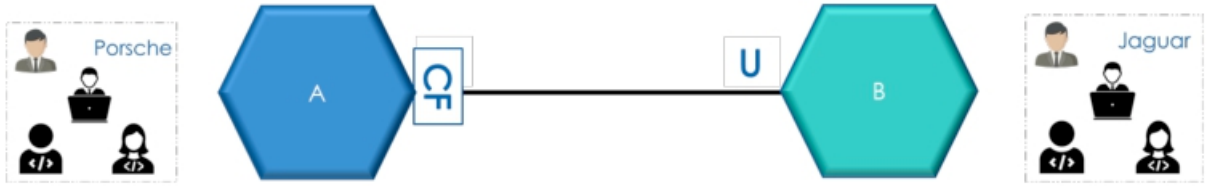


Customer Supplier

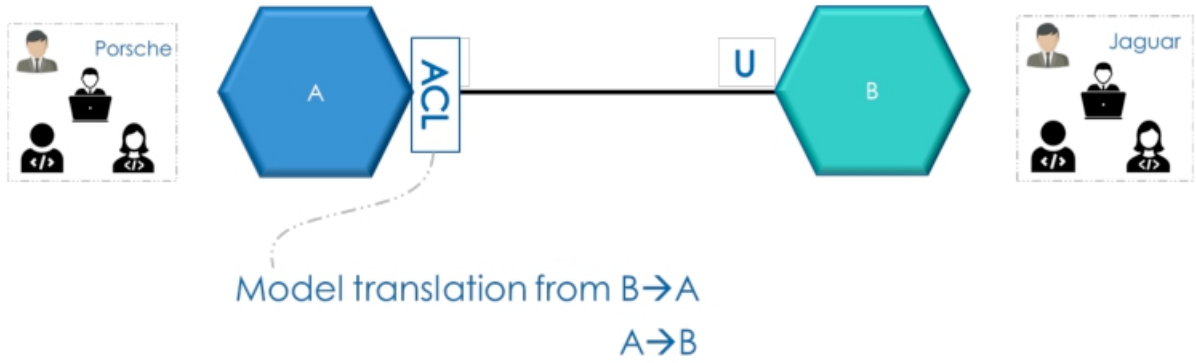


Dependency Management - BC Integration Patterns

Conformist pattern



Anti Corruption Layer (ACL)



Quick Review

Intra Service - Function calls, API, Messaging

External Service - Microservice depends on external Services

Inter Service - ALWAYS use API, Messaging

Manage dependencies using BC Integration patterns

Event Driven Architecture

Foundation for designing microservices interactions

1. Introduction to Events
2. Event Driven Architecture (EDA)
3. API versus EDA

Online Shopping

Event

Customer Checks outout in an online store

Email system sends an email to customer

Consumer

Inventory system updates

Consumer

Shipping department ships out the order

Consumer

Events

Events are an indication of something of significance that has happened at a point in time i.e., past

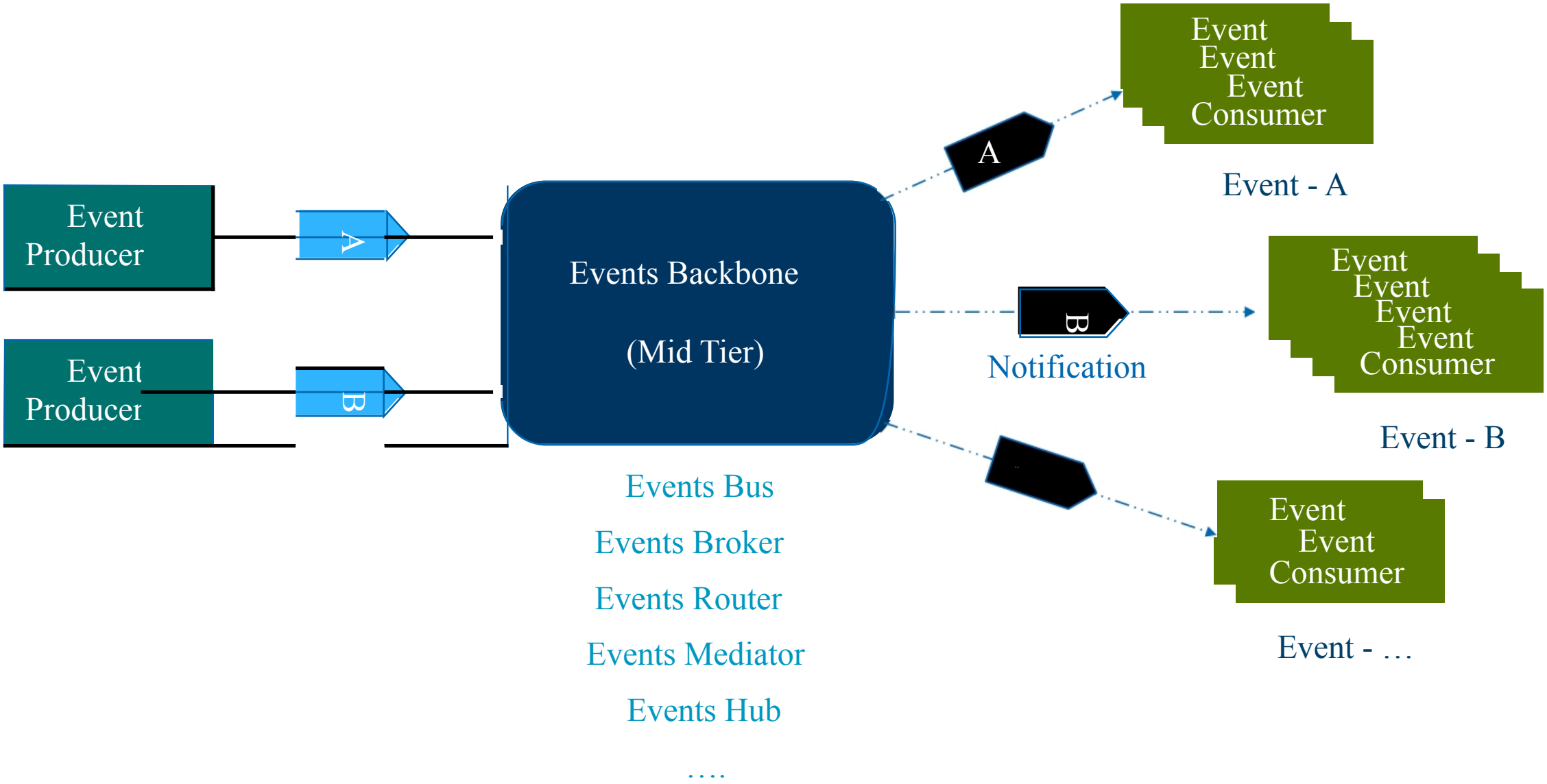
When an event occurs one or more Consumers are notified.

Consumers react to the event independent of the producer and other consumers.

Event Driven Architecture (EDA)

Software architecture paradigm that promotes designing of systems as a collection of loosely coupled components that act as event producers and consumers.

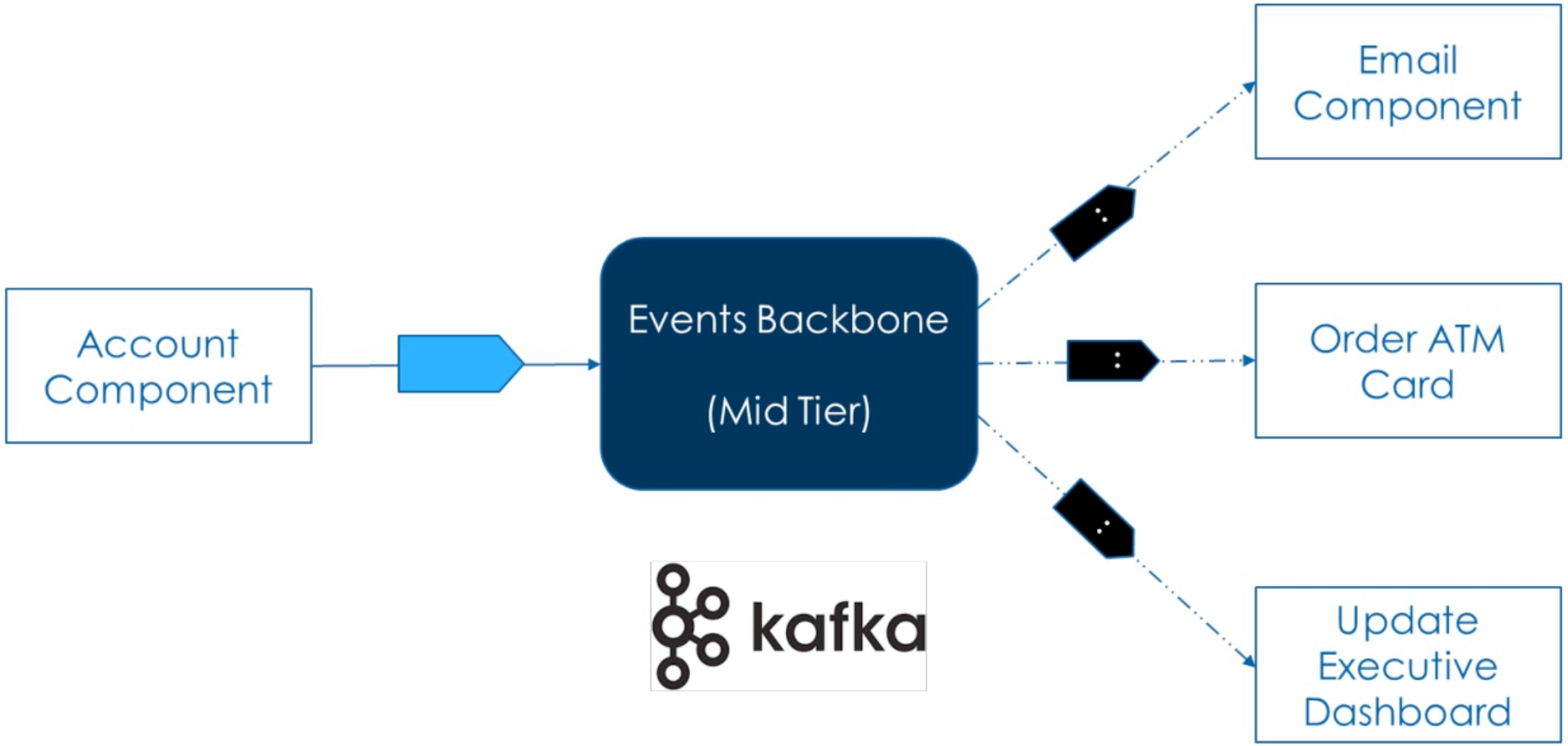
Event Driven Architecture (EDA)



Events Backbone - Technology examples

AWS
Event Bridge

Example : EDA



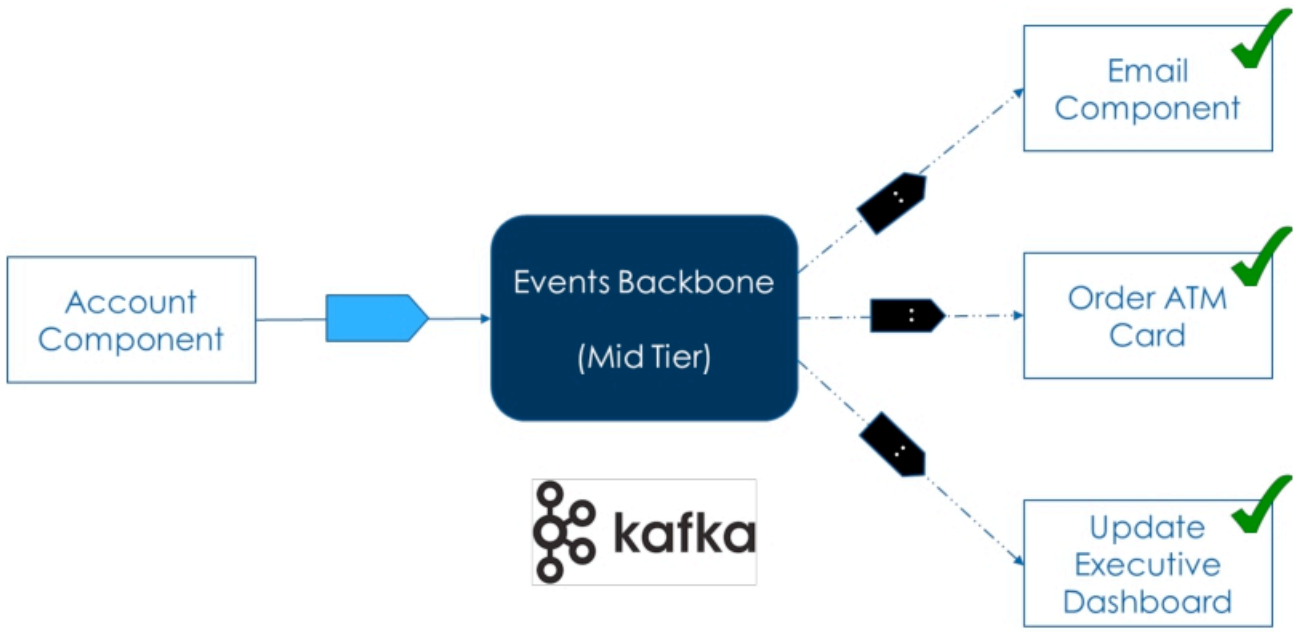
Pub -Sub messaging pattern is used for the realization of EDA !!!

Event Message Content

May contain the State data *OR* the Meta data

Account# 12345
Lname Doe
Fname John
Address 456 Main Street
… …

Consumers gets ALL of the relevant data



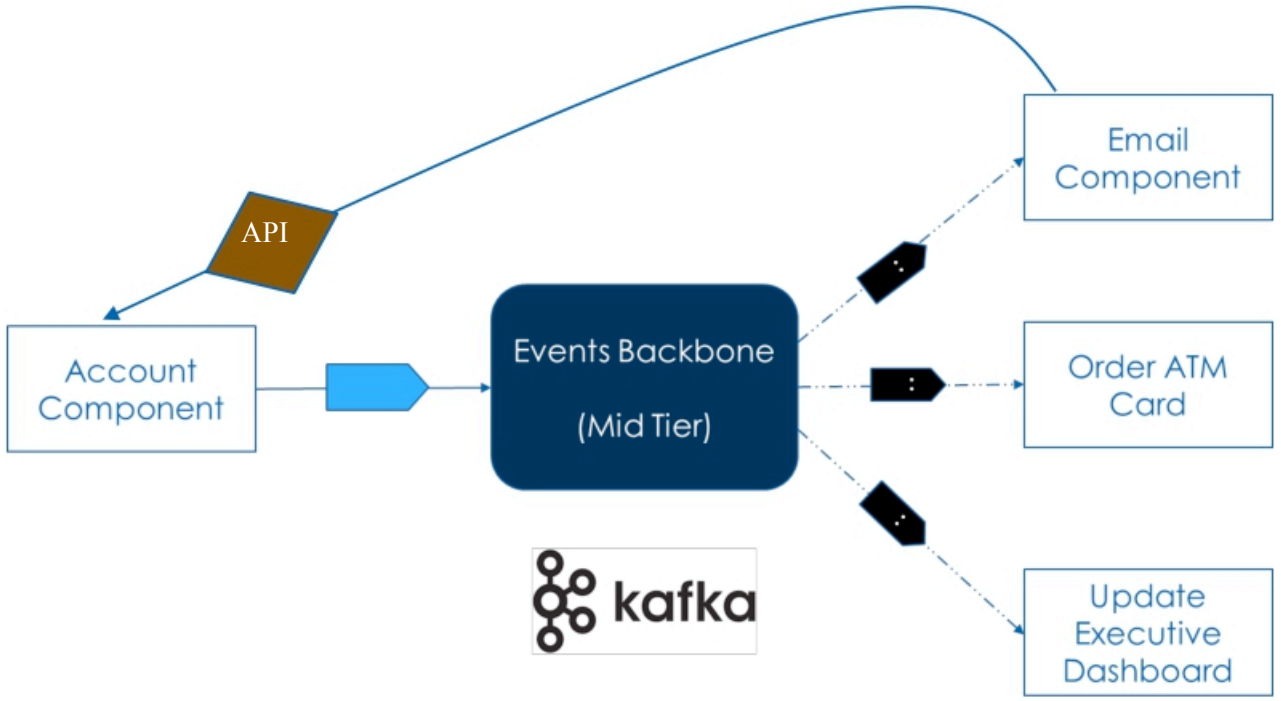
Event Message Content

May contain the State data or the Meta data

Account Created

Account# 12345

Consumers get the state data from the Source



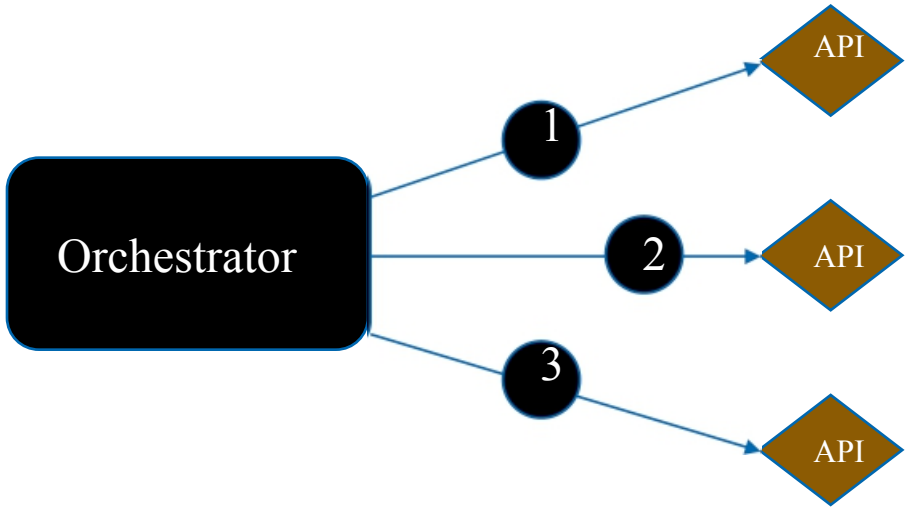
State Data versus Meta Data

No hard & fast rules - consider:

- Large size of the message — Meta Data
- Too many call backs to producer — State Data

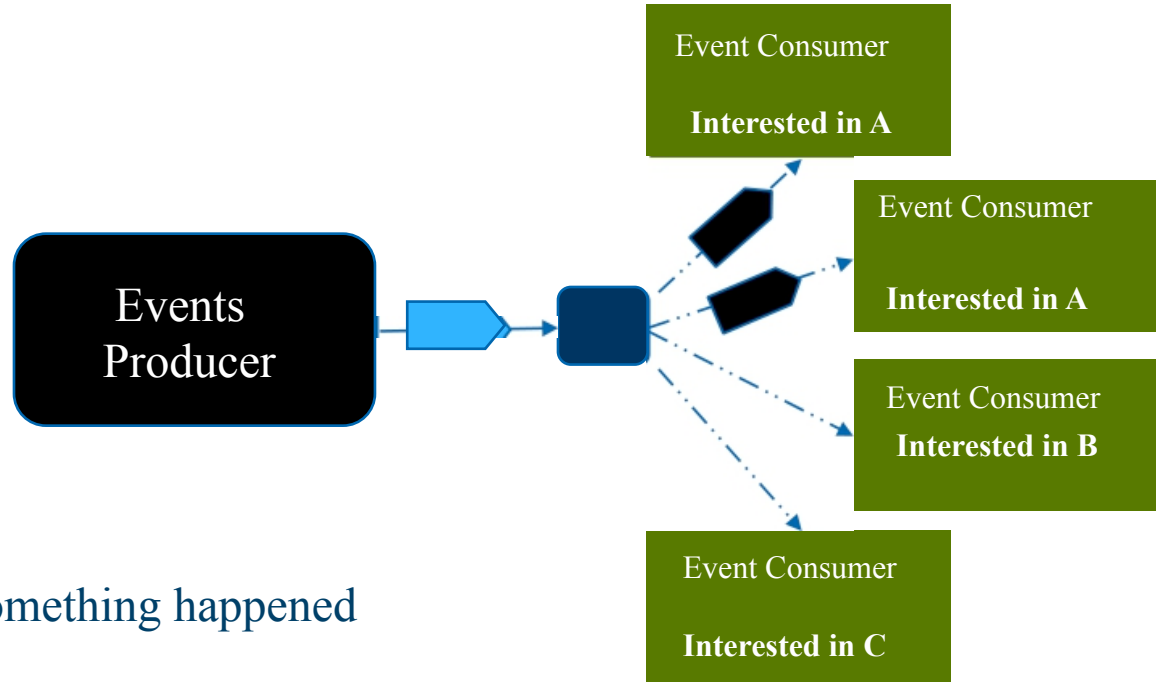
API versus EDA

API are directed commands whereas events are observable



Do something (waits for response)

Business logic & flow deisions



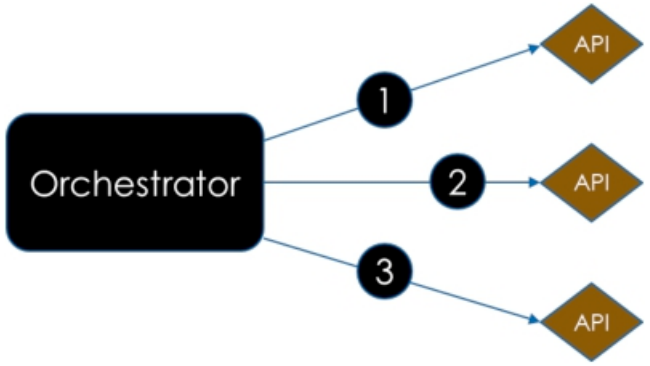
Something happened

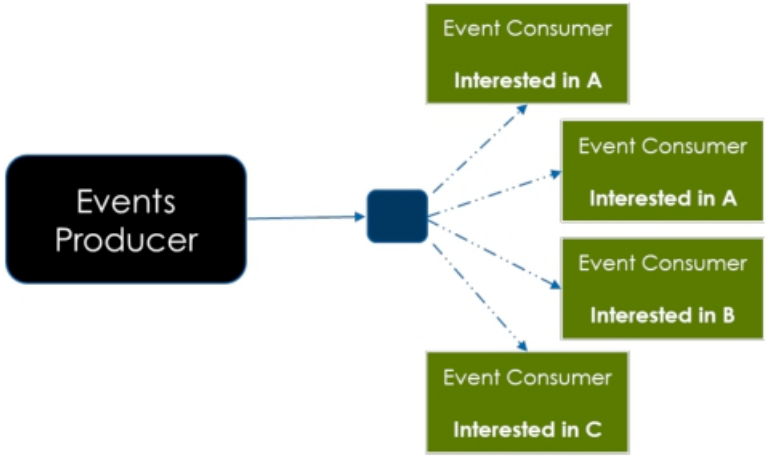
Event Observers

Autonomy in decisions

API vs EDA

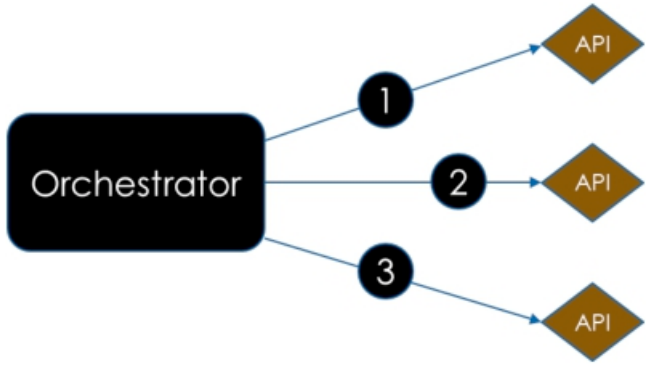
API	Events
Caller has knowledge of API	Producer doesn't know Consumers
Synchronous	Asynchronous | Higher availability
Message = Request / Response	Message = Event data
Relatively high coupling	Highly Decoupled & Extensible
Centralized business logic	Business logic across components
Easy to understand the flow	Relatively difficult to follow

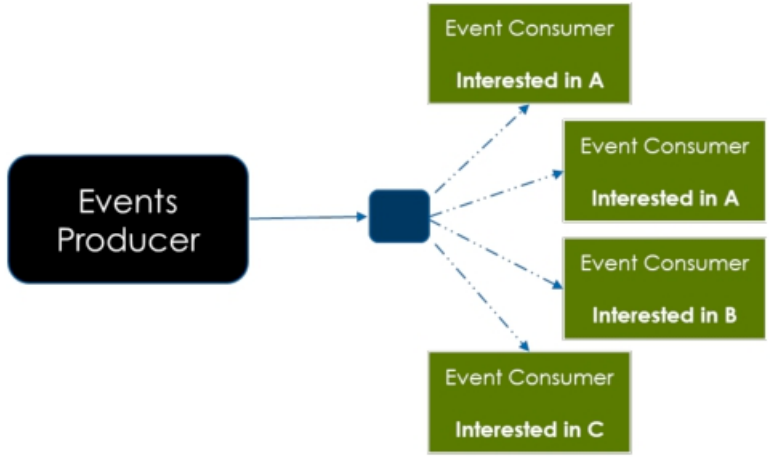




What does Microservices use?

Both - depending on the use case & requirements





What does Microservices use?

Quick Review

Event - Indicates that something happened

EDA - Architecture paradigm based on events

- Asynchronous
- Highly De- coupled | Extensible
- De - Centralized business logic/process

Pub- Sub in action

Demonstration of the Pub- Sub using RabbitMQ

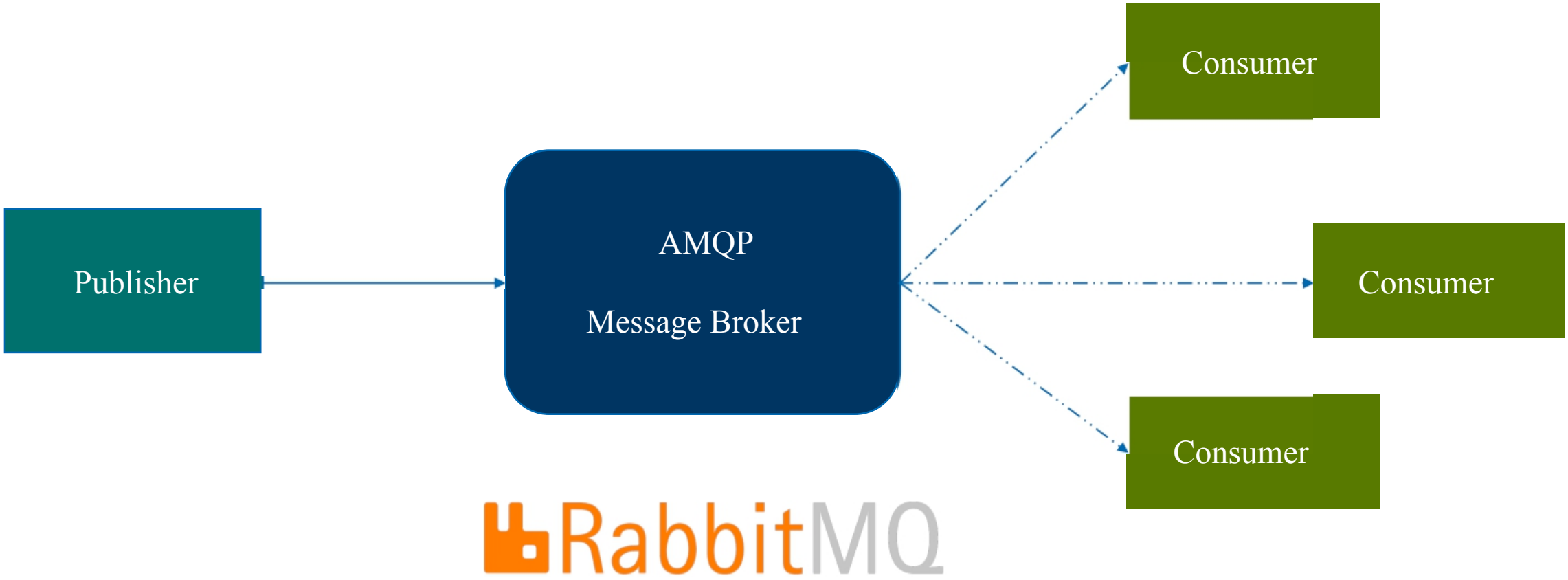
1. Basic AMQP Concepts
2. Setup a broker instance on Cloud AMQP (free)
3. Demonstrate - working of an Exchange (Pub- Sub)

Intent is to provide a high- level overview of essential concepts !!!

https://www.rabbitmq.com/tutorials/amqp -concepts.html

Intent is to provide a high- level overview of essential concepts !!!

AMQP Publisher | Broker | Consumer



https://www.rabbitmq.com/tutorials/amqp - concepts.html

AMQP protocol is Programmable

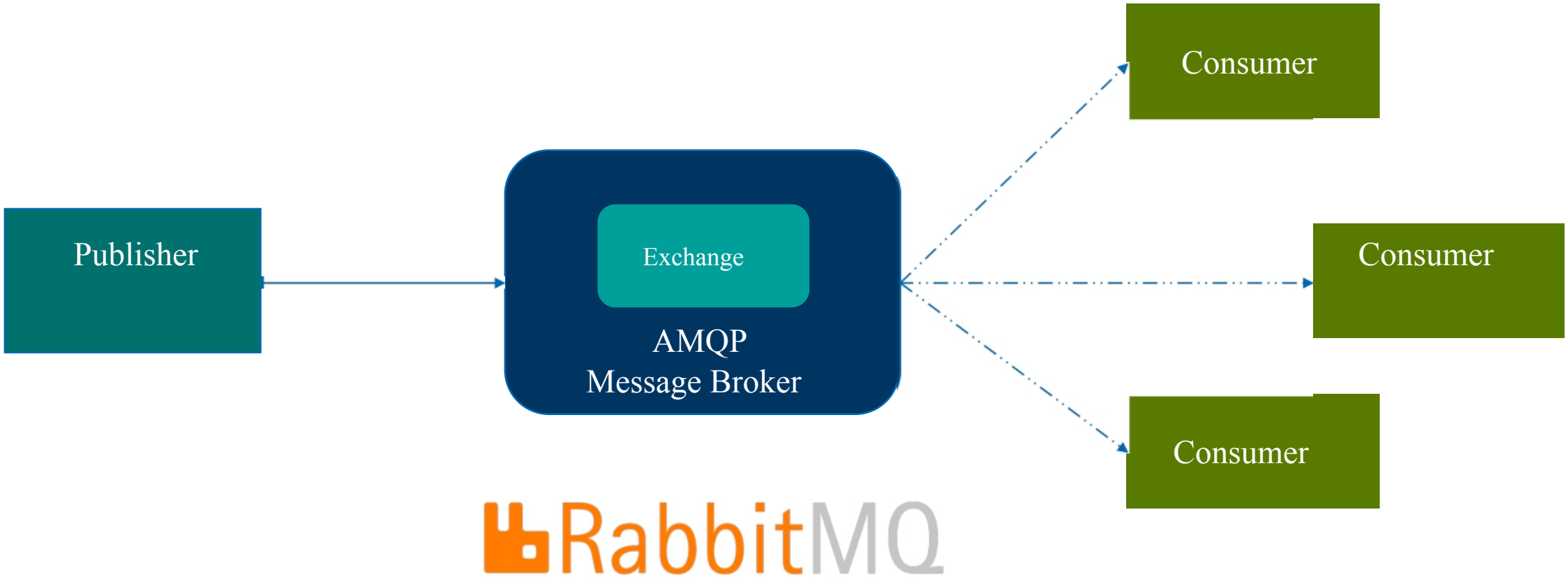
Developers make the routing decision

- Older Messaging systems required Administrators to setup routing !!!

  E.g., IBM MQ

Gives Developers lot more freedom; but raises potential for conflicts !!!

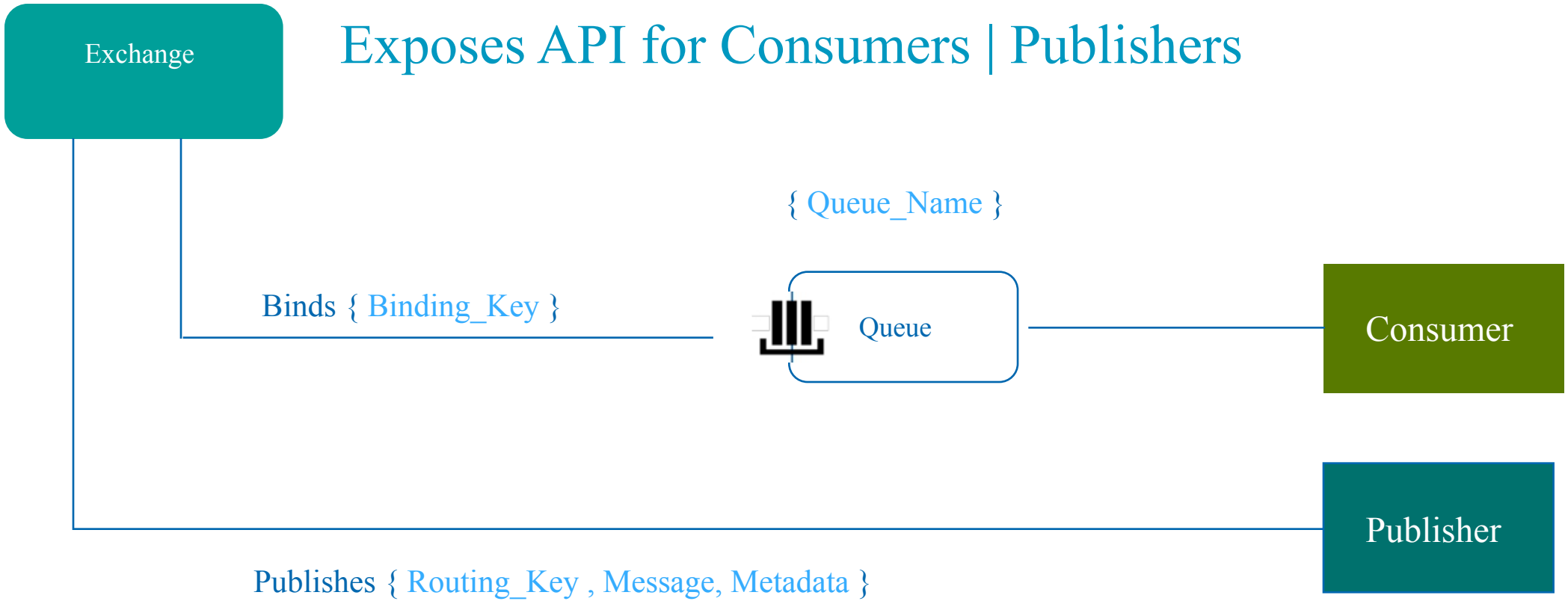
AMQP Publisher | Broker | Consumer



https://www.rabbitmq.com/tutorials/amqp - concepts.html

Message Routing

RabbitMQ



Exchange uses {Binding_Key , Routing_Key , Metadata} for routing decisions

Exchange types

RabbitMQ

Exchange

Different routing logic

Direct Exchange { Routing_Key } = { Binding_Key }

- ONLY Once processing
- Multiple workers/consumers
  - Round Robin

Exchange types

Different routing logic

Fanout Exchange

Keys are IGNORED

- Broadcast to ALL Queues bound to the exchange

Exchange types

RabbitMQ

Exchange

Different routing logic

Headers Exchange

Routing on Header data

- Binding based on matching rules for data in header

Exchange types

RabbitMQ

Exchange

Different routing logic

Topic Exchange

Pattern matching for Routing Key

- Consumer specifies the routing *pattern in the binding key*
- Example { Routing_Key } = { Binding_Key }

Exchange types

RabbitMQ

Exchange

Different routing logic

- Direct Exchange { Routing_Key } = { Binding_Key }
- Fanout
- Topic
- Header

Demo of Topic Exchange for Publish- Subscribe

Step -1 Setup the Test account Cloudamqp.com

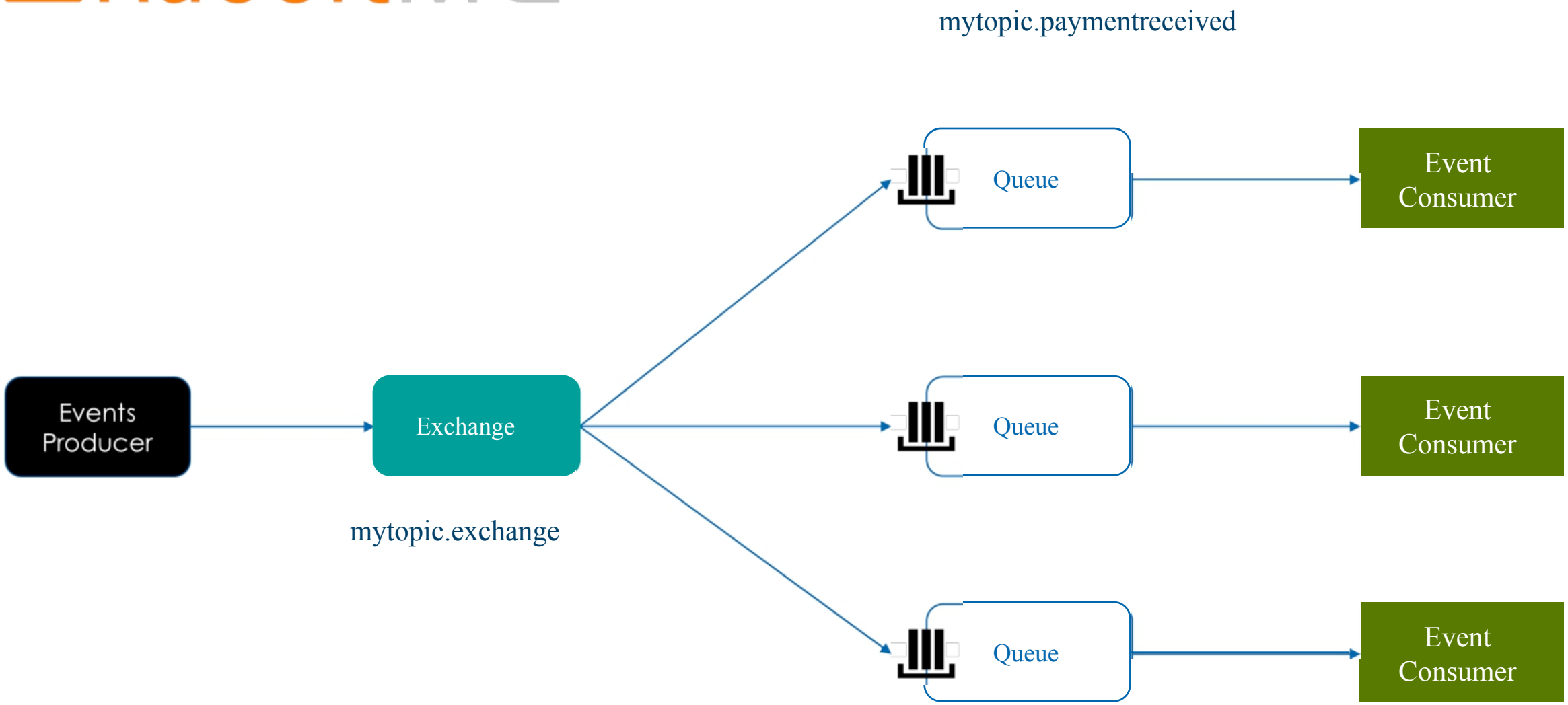
https://www.cloudamqp.com/

Step -2 Create a Free instance of the RabbitMQ Broker

Step -3 Setup a Topic Exchange (acme.test)

Step -4 Bind Queue(s) to Topic Exchange & Test

RabbitMQ


mytopic.paymentreceived
Events Producer
Exchange
mytopic.exchange
Queue
Queue
Queue
Event Consumer
Event Consumer
Event Consumer

Domain Events

Modeling the Domain Events

1. Types of Events in DDD
2. Domain Events & Handlers
3. Transactional nature of events

DDD & Events

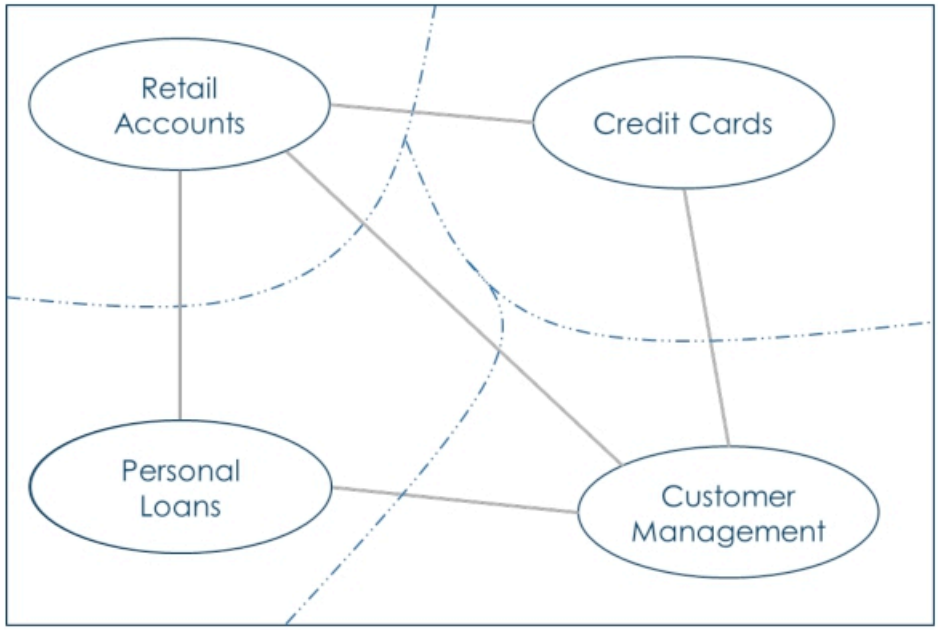
Events are an integral part of the model | Bounded Context

DepositMade

Withdrawal

LoanRejected

PaymentMade



PaymentMade

MerchantTransaction

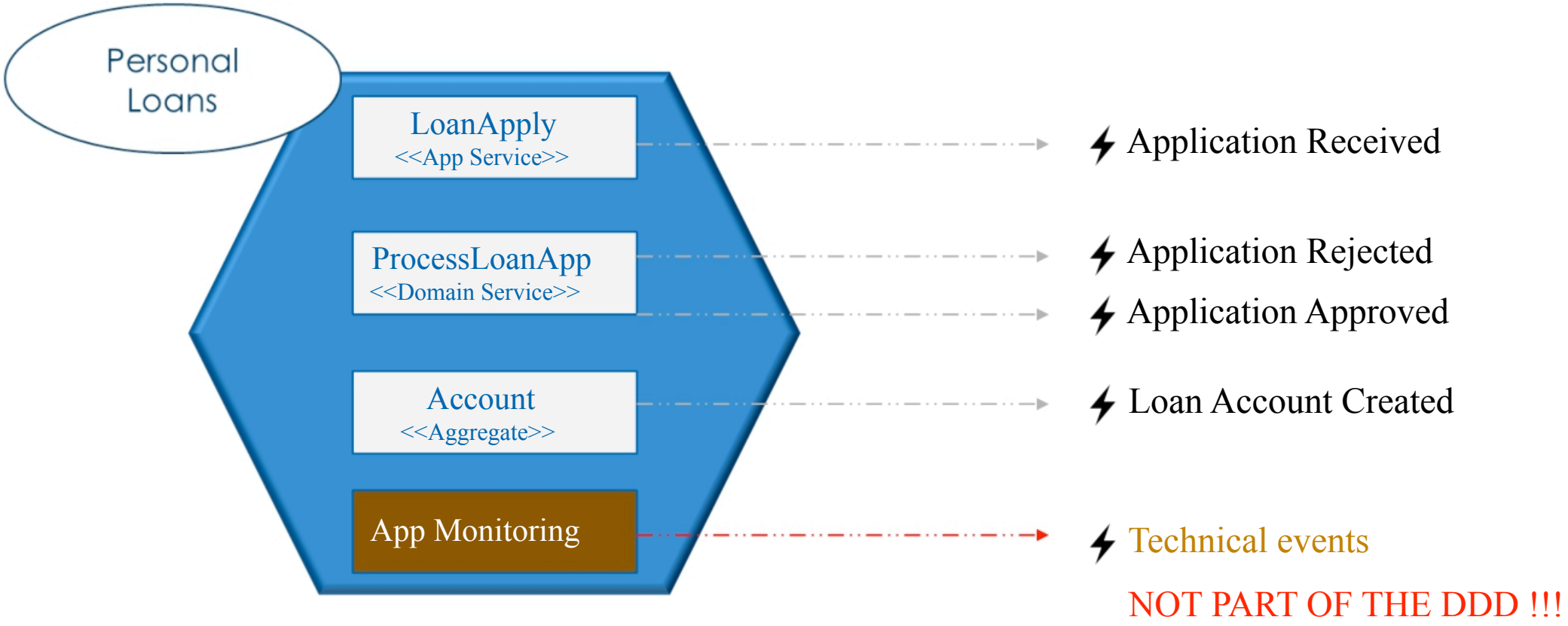
CustomerSetup

RatingUpdated

Events are part of the Ubiquitous Language

Event Examples : Loan Application

Components within a Microservice raise events



Why use events in DDD?

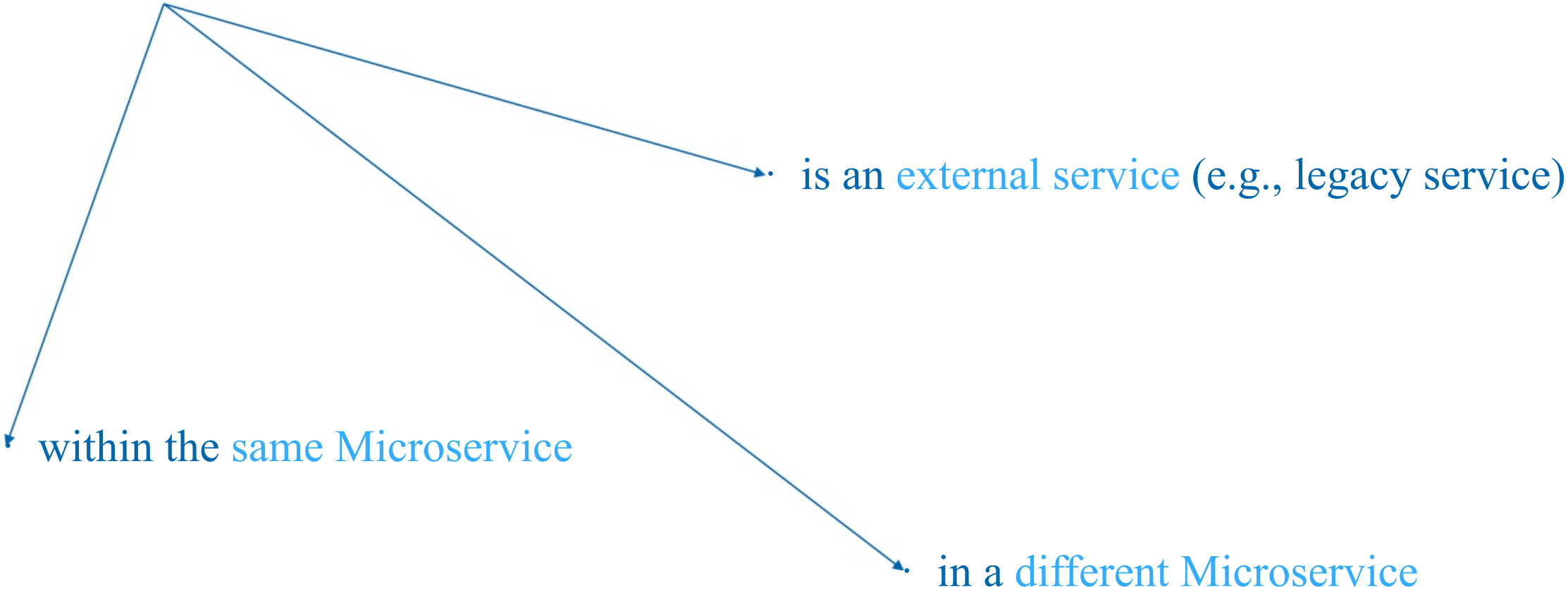
Events occur naturally in domains

Makes it easier to understand and manage the domain logic

Leads to higher levels decoupling

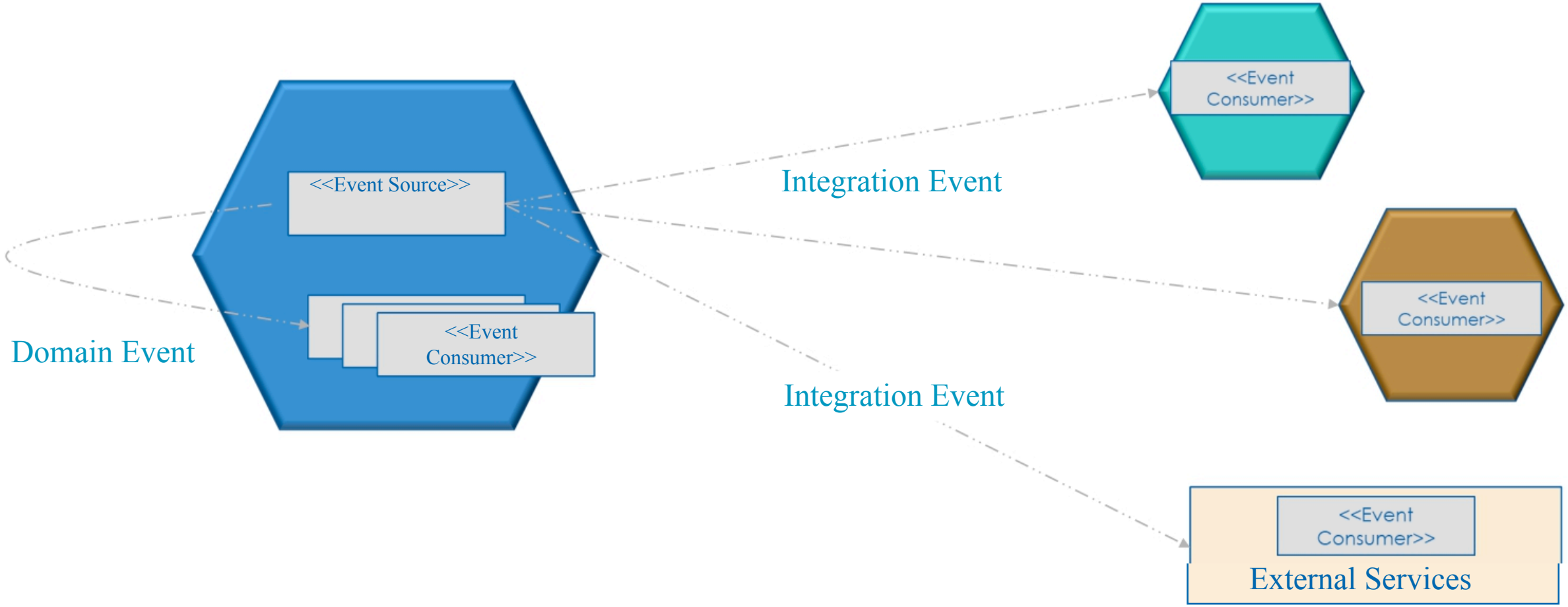
Event Consumers

Consumers subscribe to the events of interest & react to it



DDD Types of Events

Based on the Bounded Context of Source & Consumer



Domain Event

A Domain Event is a message that informs other parts within the same Bounded Context that something of significance has happened.

Domain Event indicates a state change in a BC; consumers respond by executing some business logic within the same BC.

Domain Events are natural

All domains have the concept of events

Customer checks out

Customer Checked Out

Start Order Processing

Order is Shipped

Order Shipped

Notify Customer - Order on the way

Identifying Events in a Domain

Look for statements like

"When this has happened then do ……. "

This part represent the Event

This part represents the Reaction to that event

a.k.a. Side Effect

Event Naming

Always use past tense as *event* has already happened

- **MUST** use Ubiquitous Language
  - ⚡ CustomerCheckedOut
  - ⚡ OrderShipped
- DO NOT add *Event* as a suffix ☺
  - ⚡ ~~CustomerCheckedOutEvent~~
- DO NOT add *Operation* as suffix
  - ⚡ User Registered
  - ⚡ ~~User Created~~

Domain Event Handler = Consumer

Refers to the implementation of event consumer logic

- Are part of the Microservice code base
- Subscribes to the events of interest
- Zero or More Handler(s) per event

Event Handler Naming

Use the event name for naming the handler function

- **MUST** use Ubiquitous Language
  - CustomerCheckedOut
  - OrderShipped
- DO NOT add *Handler* or *Receiver* as a suffix ☺
  - ~~OrderShippedHandler~~

Domain Events are natural

All domains have the concept of events

Customer Deposited a Check

Check Deposited

Start Check clearing process

Customer Check Cleared

Check Cleared

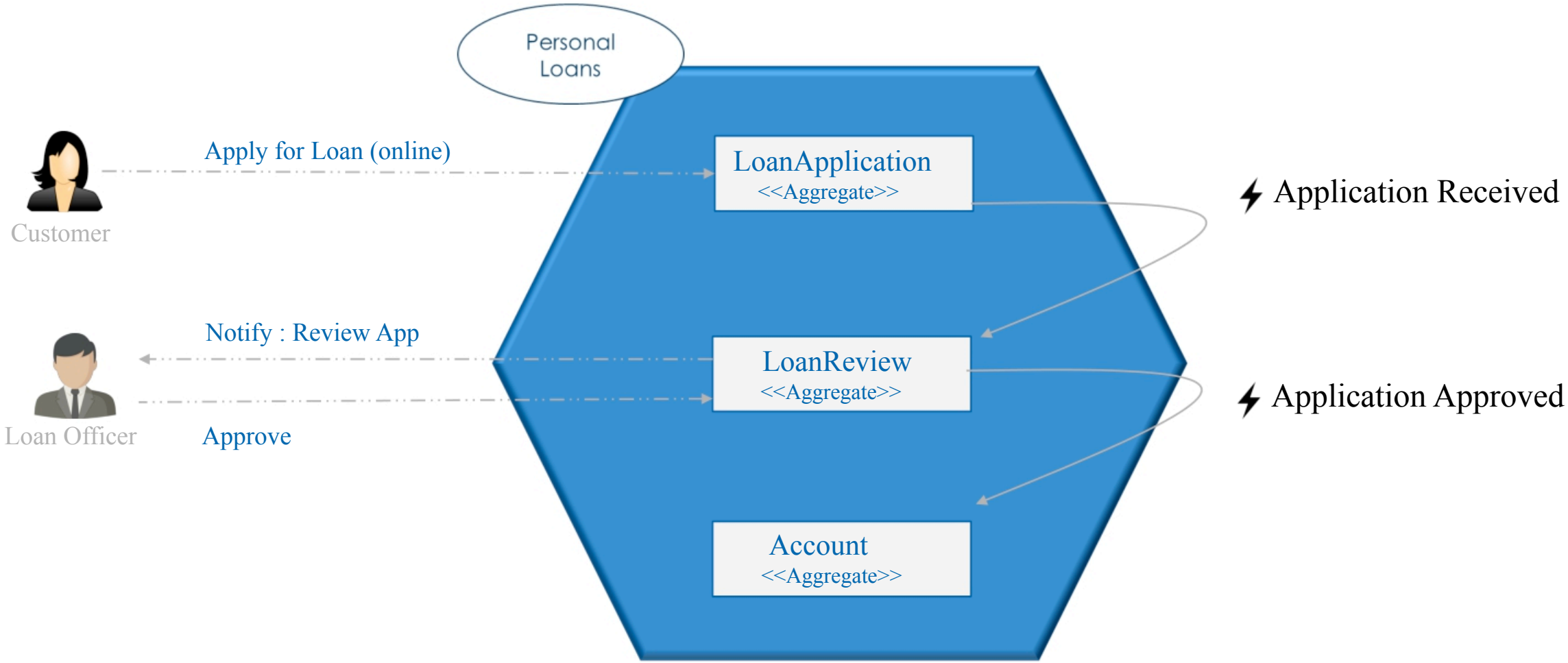
Update Account Balance

Customer A/c balance updated

Check Posted to Account

Example : Personal Loan Microservice

Domain Event Source & Handler are in the same process



Domain Event Realization

Does NOT have to be messaging !!!

- Synchronous — E.g., direct function calls
- Asynchronous — E.g., in memory messaging

  E.g., external message broker

Domain Events & Transactions

MUST be processed atomically

- Synchronous — Easier to manage with function calls
- Asynchronous — Need to use appropriate patterns

Domain Events & Transactions

Aggregate operations & events must be raised atomically

Customer

Apply

LoanApplication
<<Aggregate>>

1. Save to database

2. FAILED to Raise

Application Received

LoanReview
<<Aggregate>>

Will not start as event NOT received

1. FAILED to Save to the database

2. BUT Raised the

Application Received

LoanReview
<<Aggregate>>

Will start but will NOT find the loan application !!!

Domain Events & Transactions

Multiple patterns for addressing transactional behavior

Customer

Apply

LoanApplication
<<Aggregate>>

1. Save all the event to storage

Application Received

2. Save the application to database

On Error: Do not raise the event(s)

On Success: Raise the event (s)

LoanReview
<<Aggregate>>

Will receive event & process

Domain Events & Transactions

Multiple patterns for addressing transactional behavior

Customer

Apply

LoanApplication
<<Aggregate>>

1. Save to database

2. Raise the Application Received

On Error: Save event in storage

Retry after X seconds

On Success: Mark event raised

LoanReview
<<Aggregate>>

Will receive event & process

Quick Review

Two types of Events - Domain & Integration events

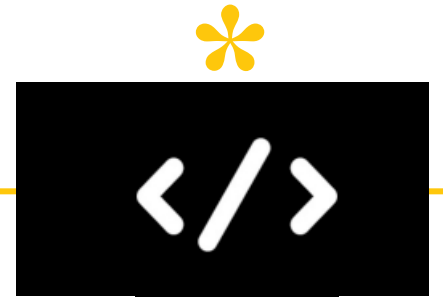
Domain events may be handled Synchronously | Asynchronously

Follow best practices for naming the events & handlers

State updates & raising of events must be done in a Transaction

Static Class Broker Pattern

Static class for raising & handling domain events



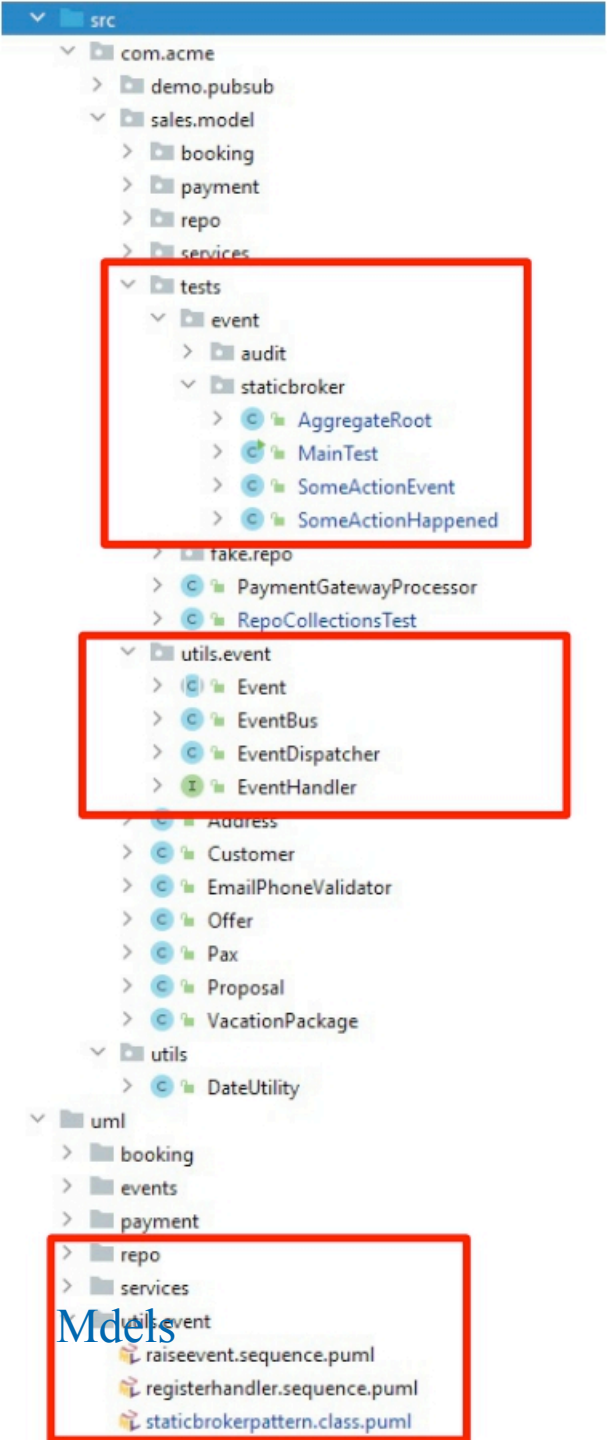
1. Framework Classes
2. Registering | Raising events
3. Code walkthrough & testing

Pattern: Static class events broker

Static class exposes functions for events management

- Raising the events
- Registering Handler
- De -Registering Handler
- All calls are Synchronous

Applies ONLY to Domain Events



JAVA Classes - Testing

com.acme.sales.model.tests.event.staticbroker

JAVA Classes

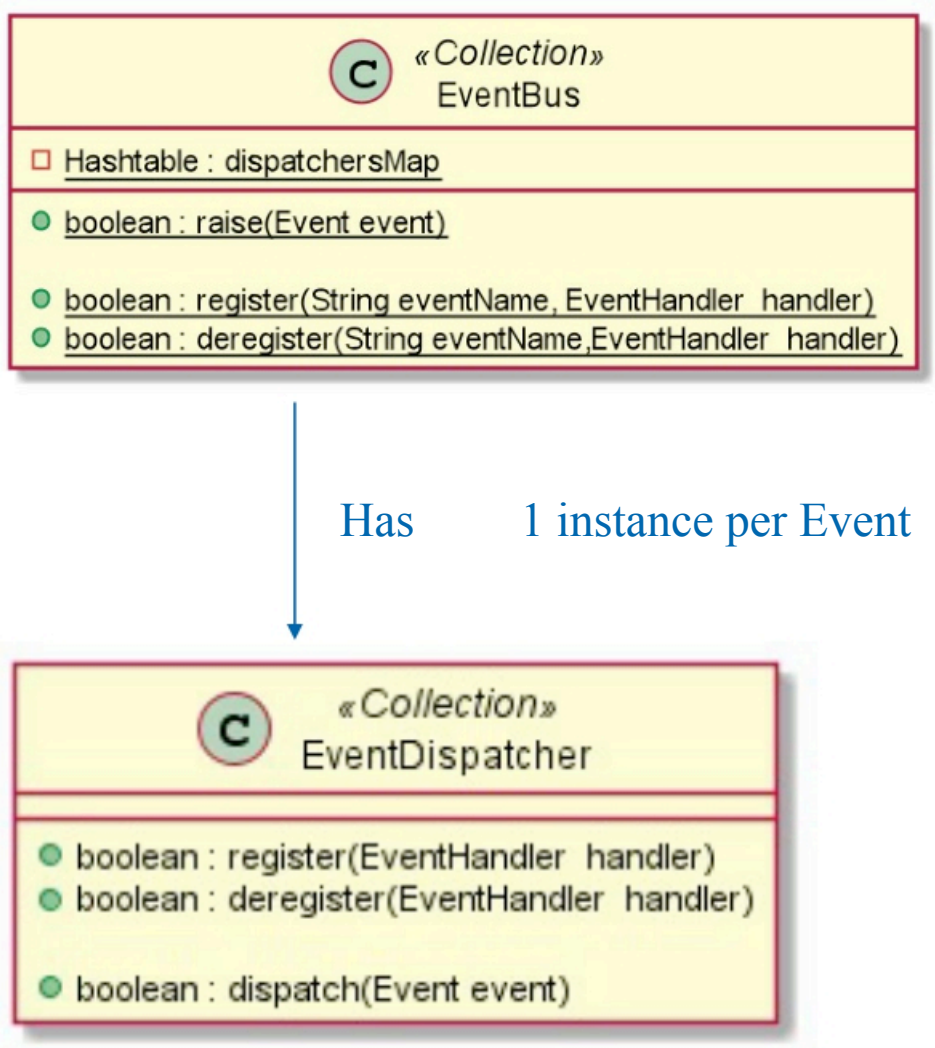
com.acme.sales.model.utils.event

Mdels

uml / utils.events

Git Branch : events

Event Framework classes

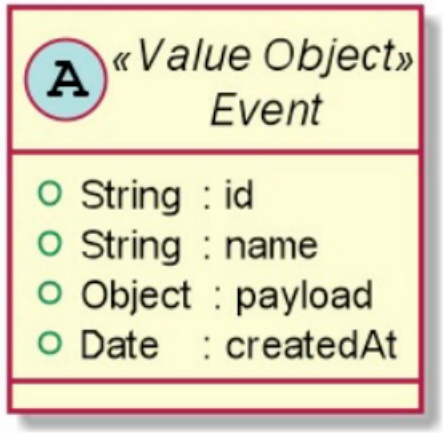


Static class that exposes functions for event management

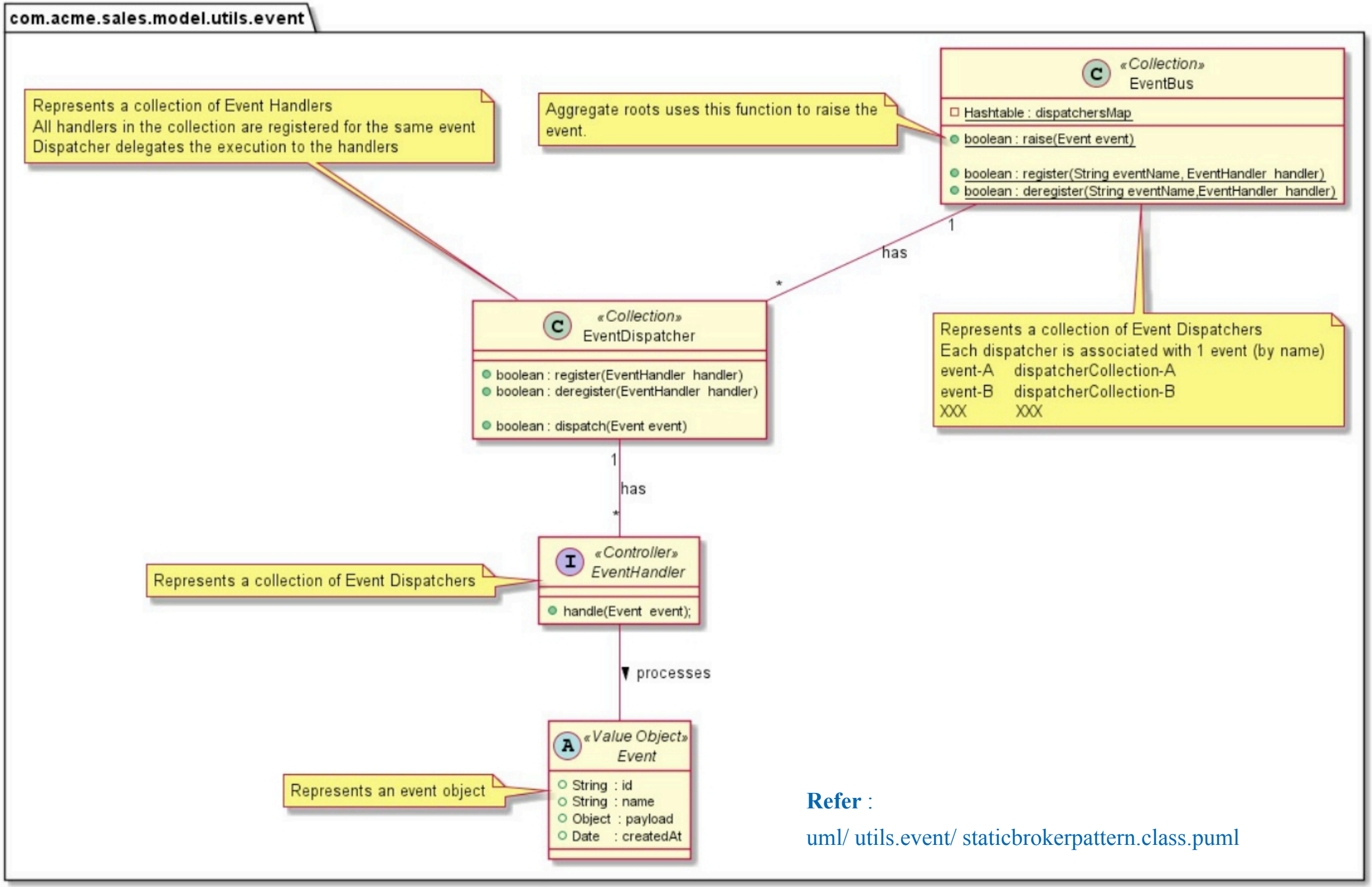
Collection of Handlers for an event

Framework classes

All Handlers implement this interface



All events extend this class



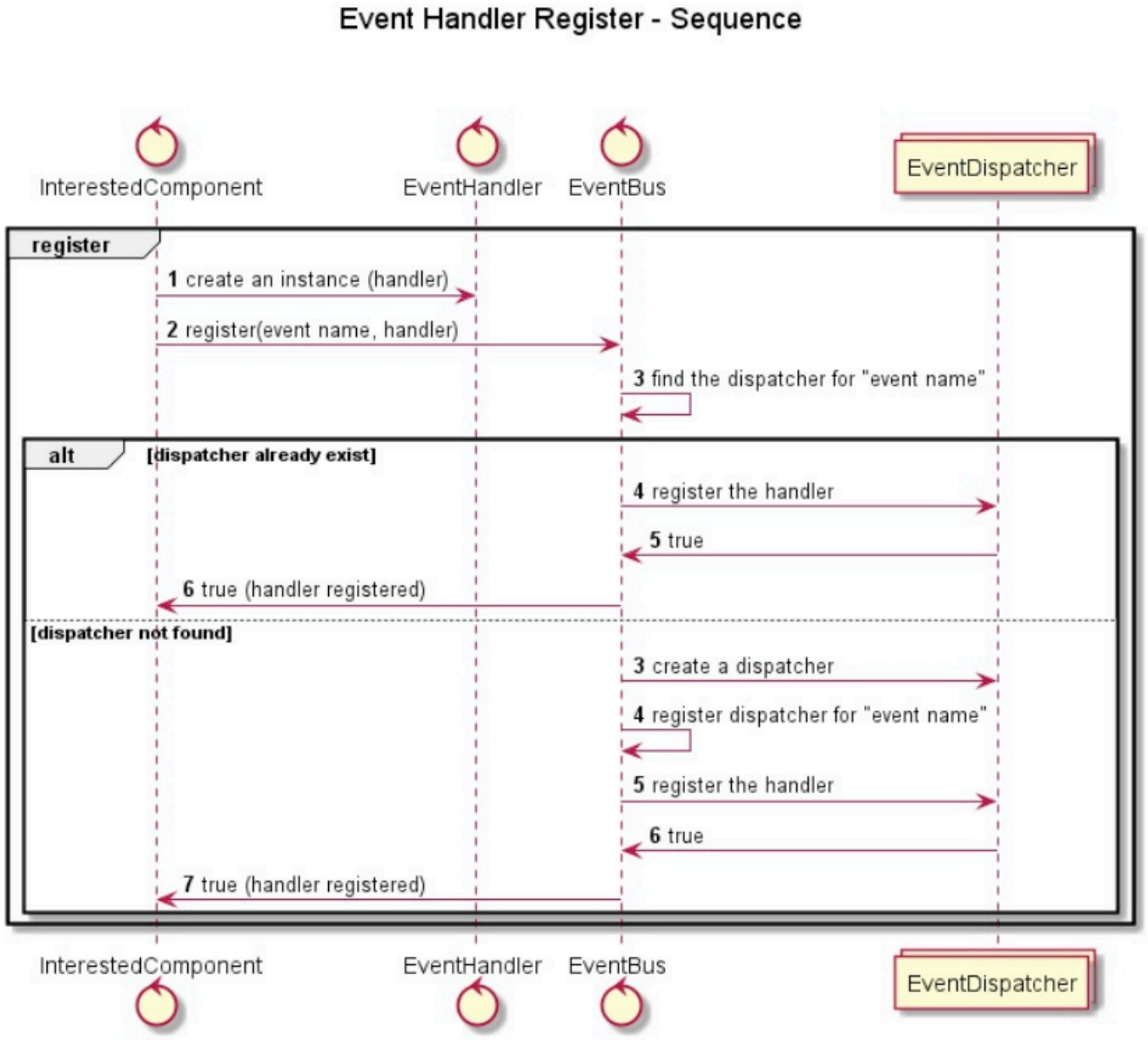
Refer :

uml/ utils.event/ staticbrokerpattern.class.puml

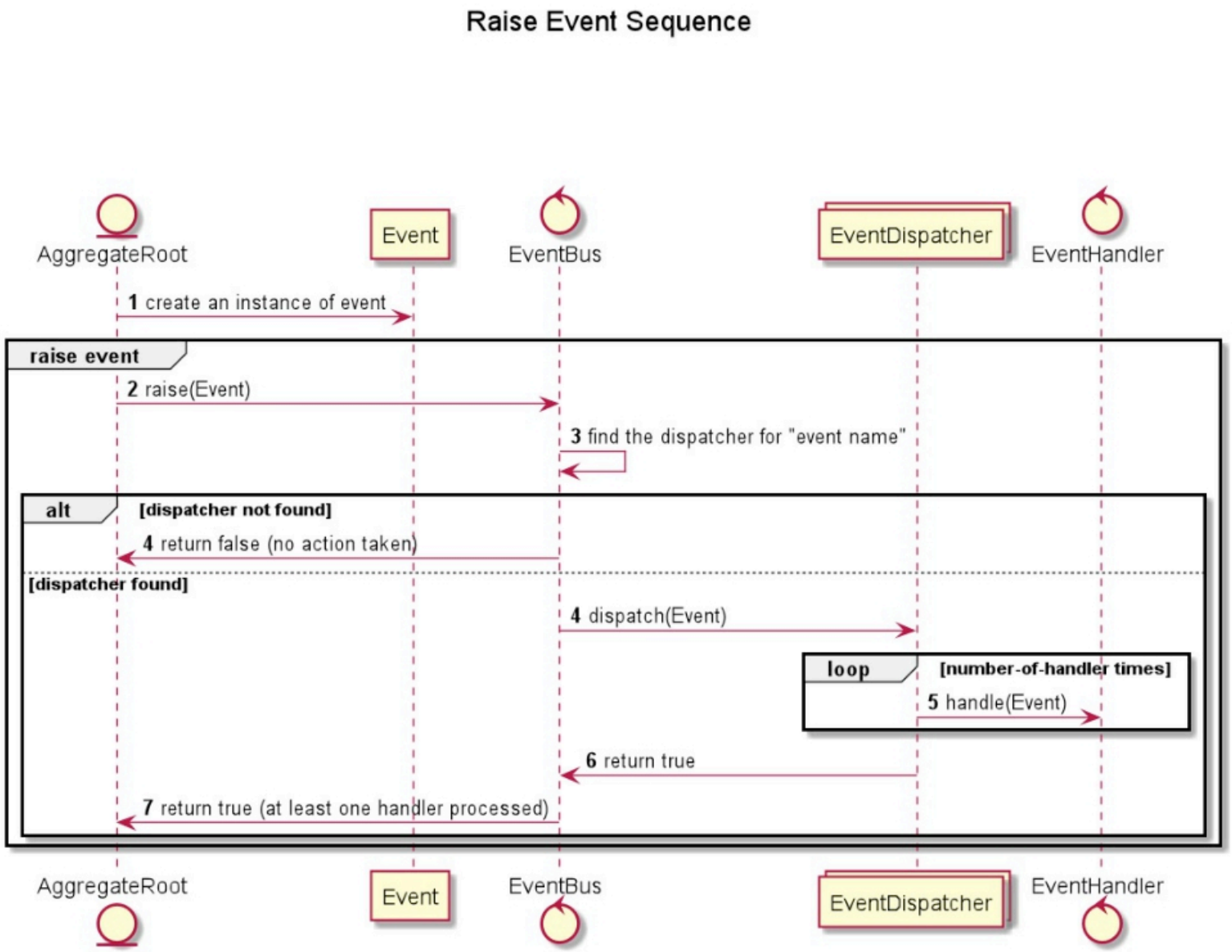
Register Sequence

Refer:

uml/utils.event/registerhandler.sequence.puml



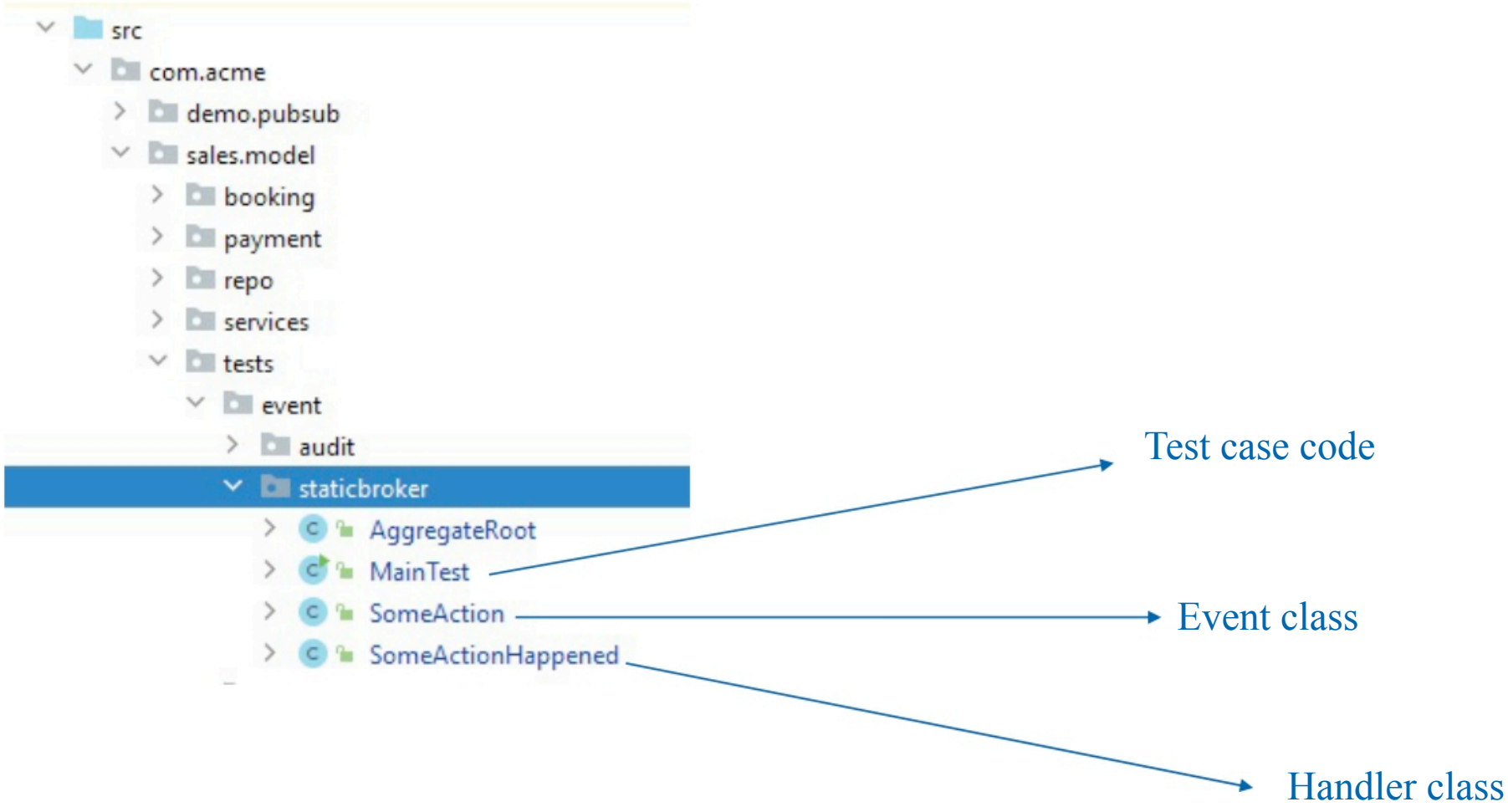
Raise event



Refer:

uml/utils.event/registerhandler.sequence.puml

Testing : Static Class Broker pattern



ACME Sales Events

Domain Events

1. Identify the Domain Event
2. Model the Event & Handler

John, Travel Advisor

I know there are multiple regulatory and compliance requirements but since I am not an expert, lets engage with Owen from legal team.

Payment Audit

Owen, Legal Expert

We are in a regulated industry. We need to ensure that we are in compliance with all regulations. Although there are multiple regulations but specifically related to the payments, it is a MUST that when a customer payment is received, it needs to maintained in payment audits for a minimum of 7 years.

Owen, Legal Expert

We are in a regulated industry. We need to ensure that we are in compliance with all regulations. Although there are multiple regulations but specifically related to the payments, it is a MUST that when a customer payment is received, it needs to maintained in payment audits for a minimum of 7 years.

Payment Audit

John, Travel Advisor

And after we receive the payment received message, the reservations are started. After all the reservations have been made successfully we mark the booking as confirmed.

Owen, Legal Expert

We are in a regulated industry. We need to ensure that we are in compliance with all regulations. Although there are multiple regulations but specifically related to the payments, it is a MUST that when a customer payment is received, it needs to maintained in payment audits for a minimum of 7 years.

John, Travel Advisor

When we receive the payment received message, the reservations are started. After all the reservations have been made successfully we mark the booking as confirmed.

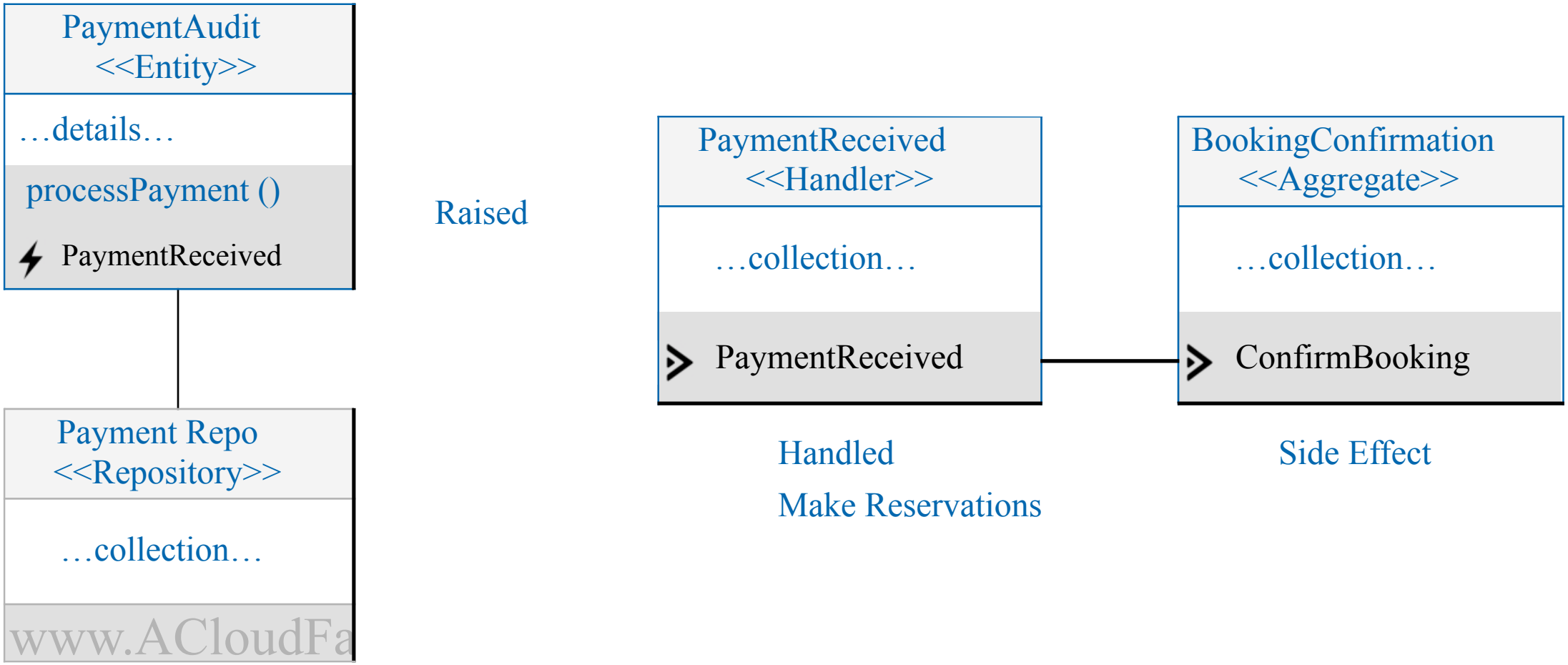
Identity an event and a side effect?

We are in a regulated industry. We need to ensure that we are in compliance with all regulations. Although there are multiple regulations but specifically related to the payments, it is a MUST that when a customer payment is received, it needs to maintained in payment audits for a minimum of 7 years.

When we receive the payment received message, the reservations are started. After all the reservations have been made successfully we mark the booking as confirmed.

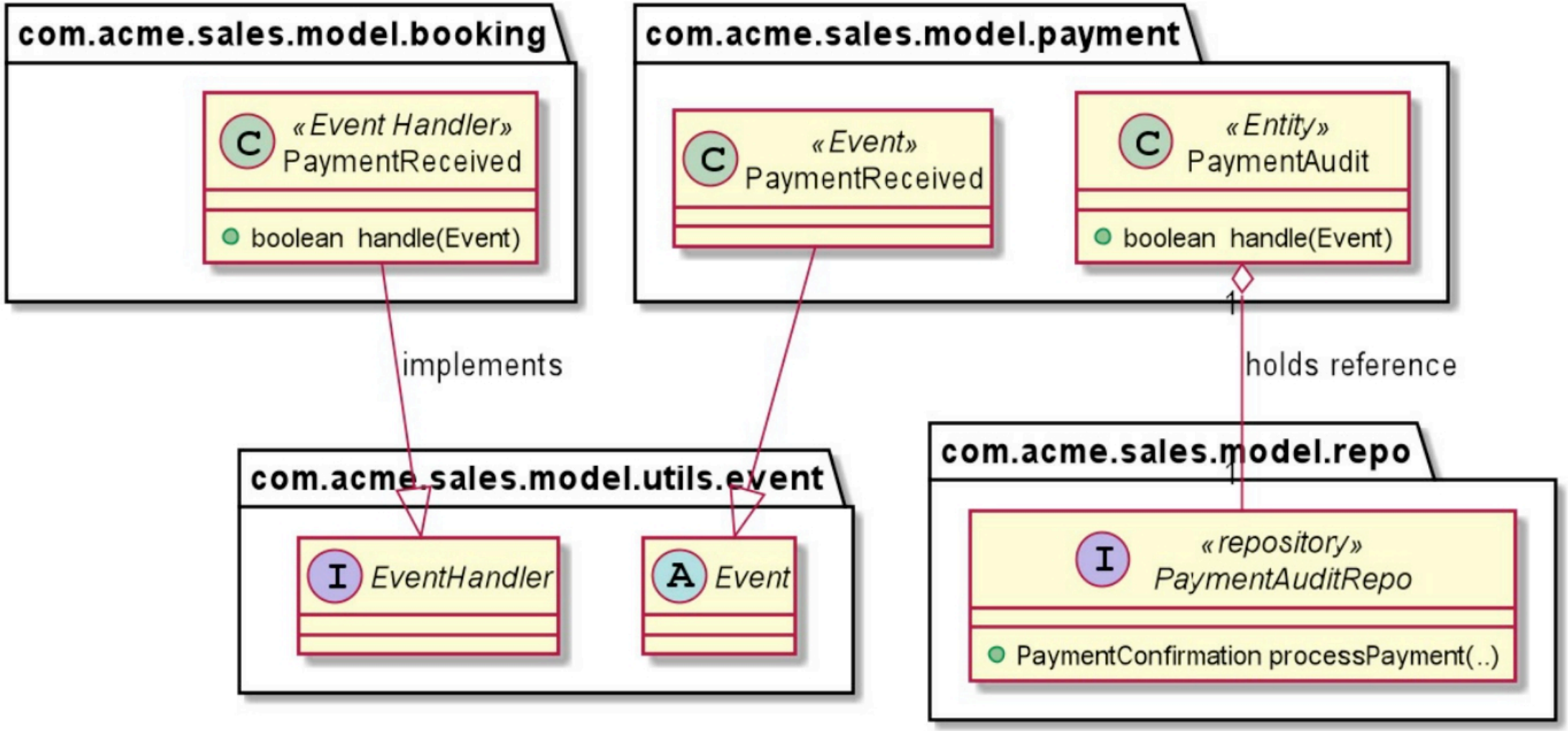
Payments Compliance Requirements

All Payment information MUST be retained for 7 years



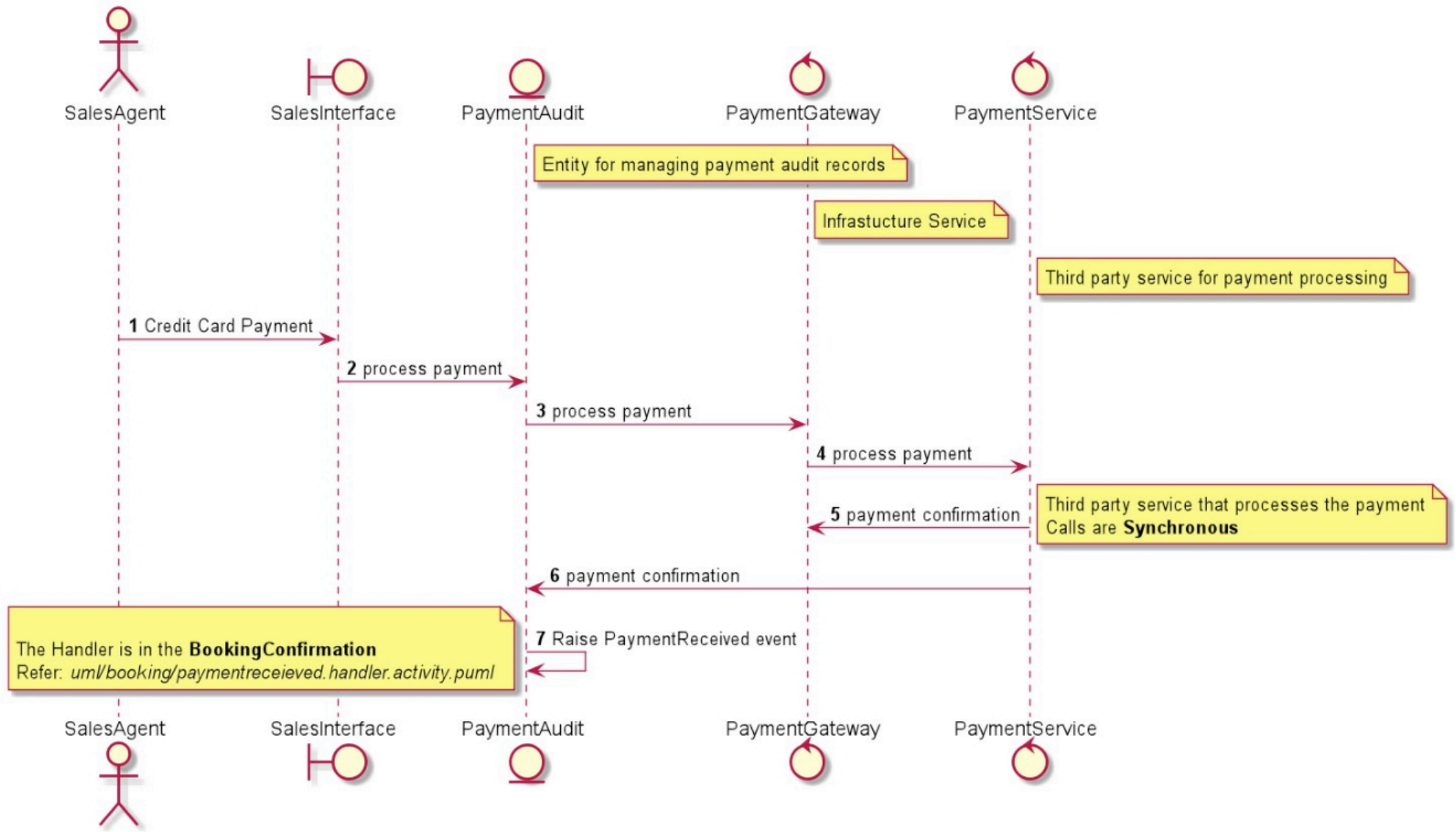
Classes

uml/events/receivedpayment.class.puml



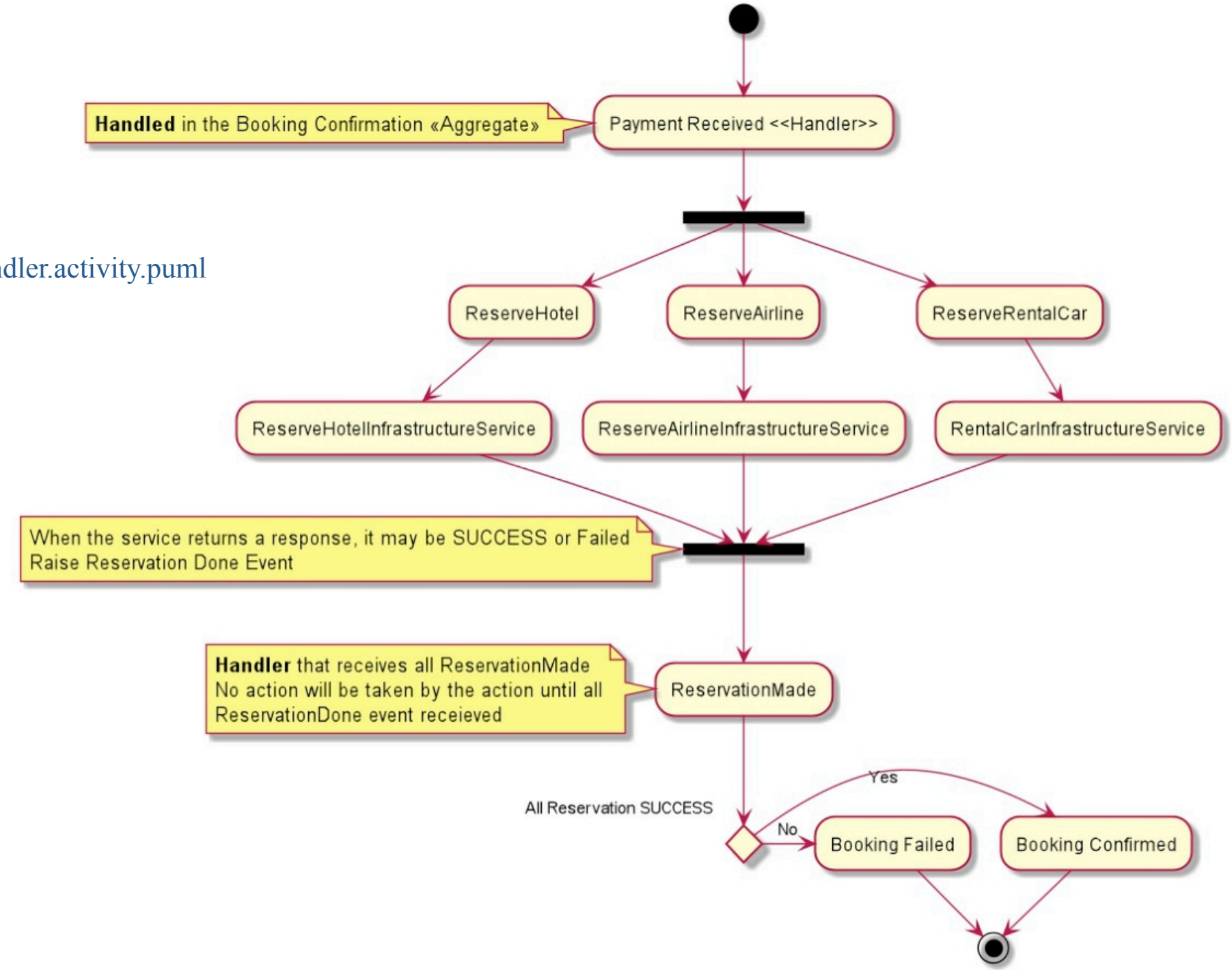
Payment Received - Domain Event

uml/payment/receivedpayment.class.puml



Handler (overview)

uml/payment/ paymentreceieved.handler.activity.puml



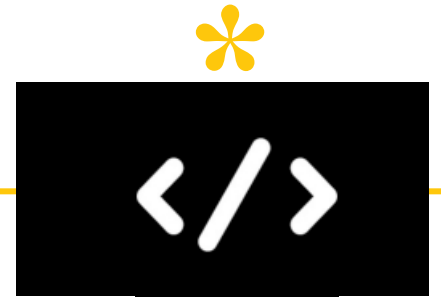
Quick Review

Static class Broker Pattern

Event processing may be Asynchronous

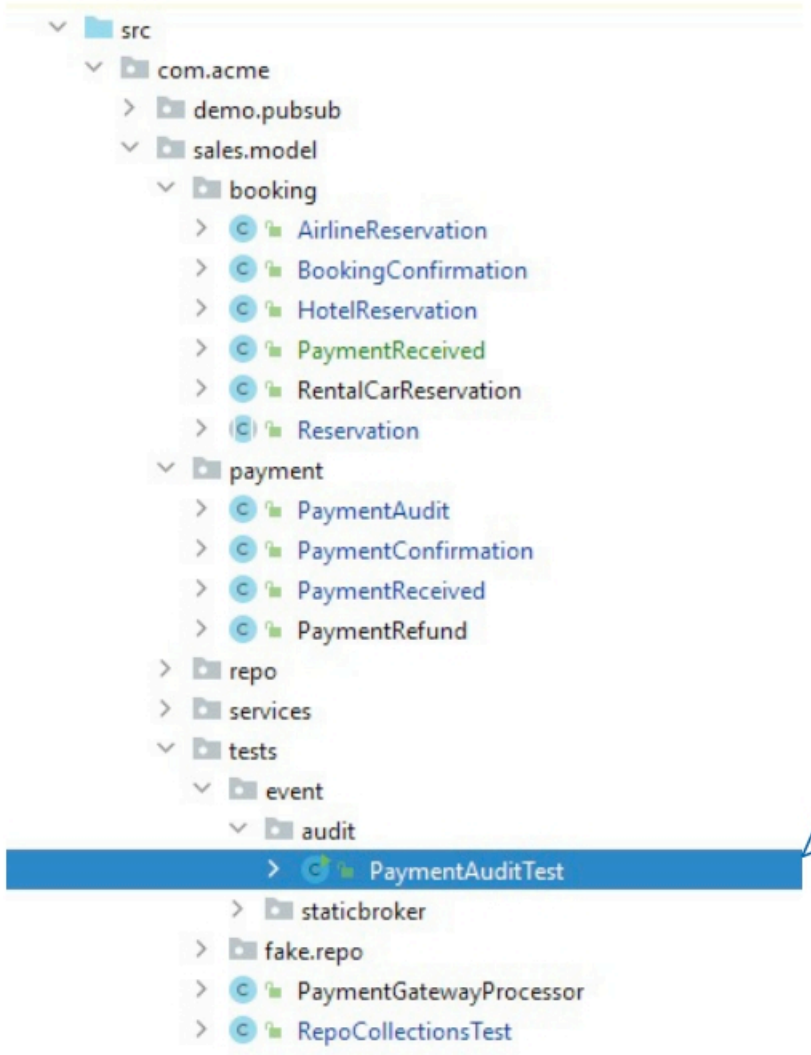
PaymentReceieved processing

Demonstrates the processing of the event

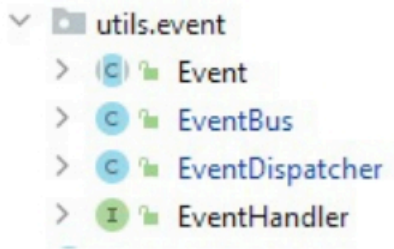


1. Test Code Walkthrough
2. Demonstrate the handler in action

Testing : PaymentRecieved → BookingConfirmation (state change)



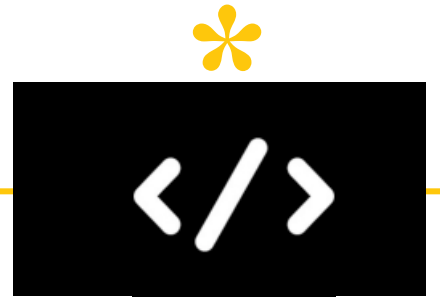
Uses the Static class pattern

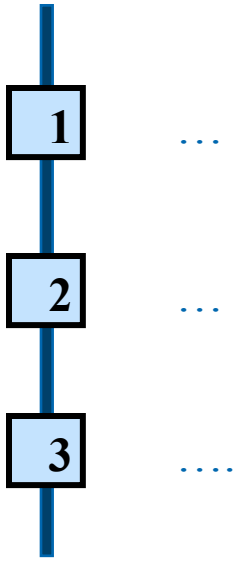


Demonstrates the triggering of events & state change in BookingConfirmation

Domain Event Implementation

Patterns for event publishing





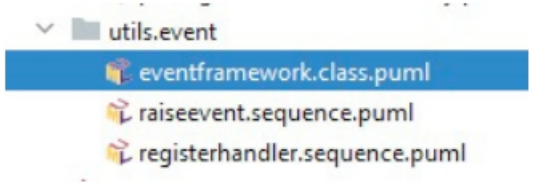
Publishing the Domain Events

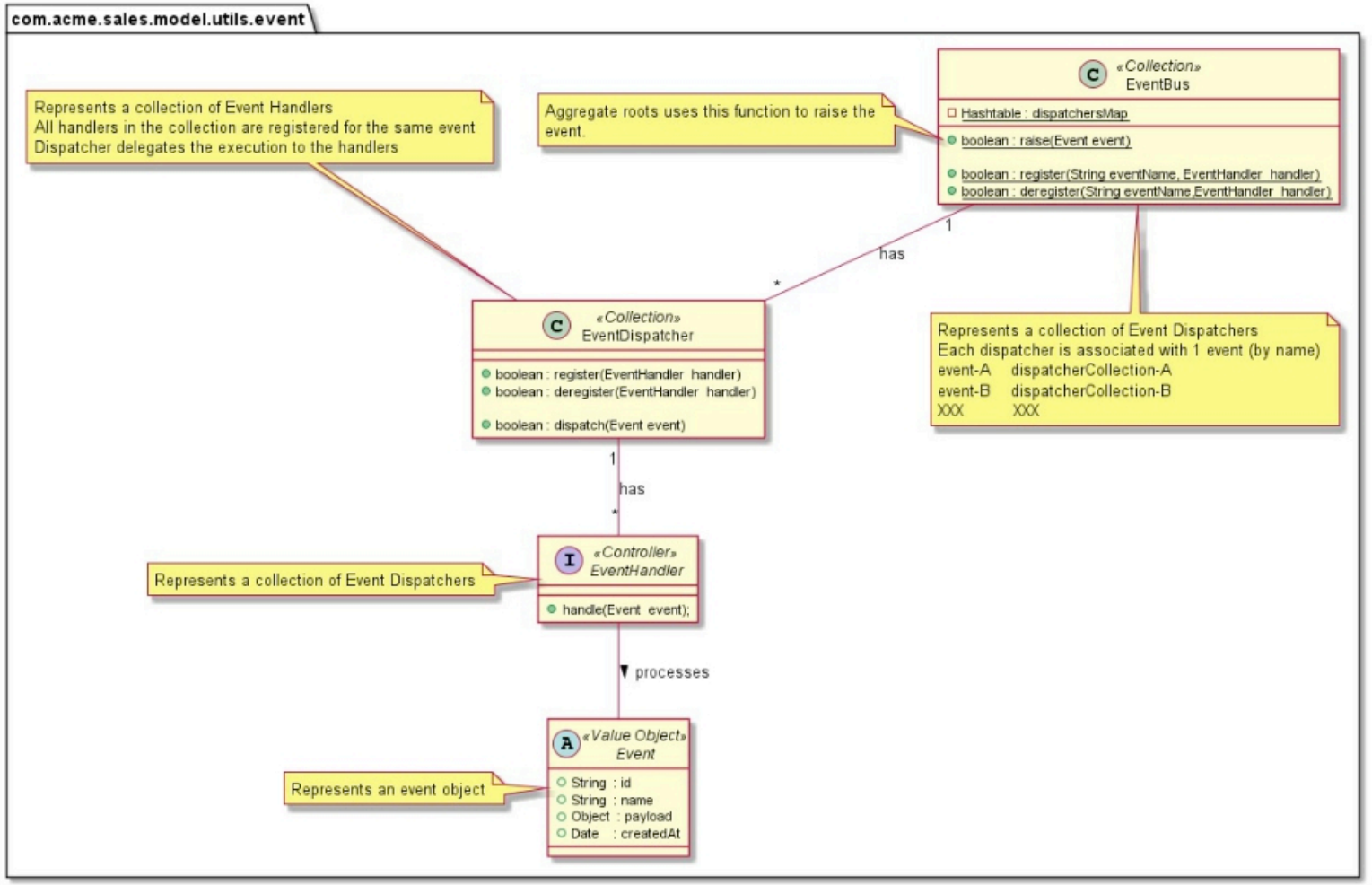
1. Using a static class

2. Return Domain Event from Aggregate

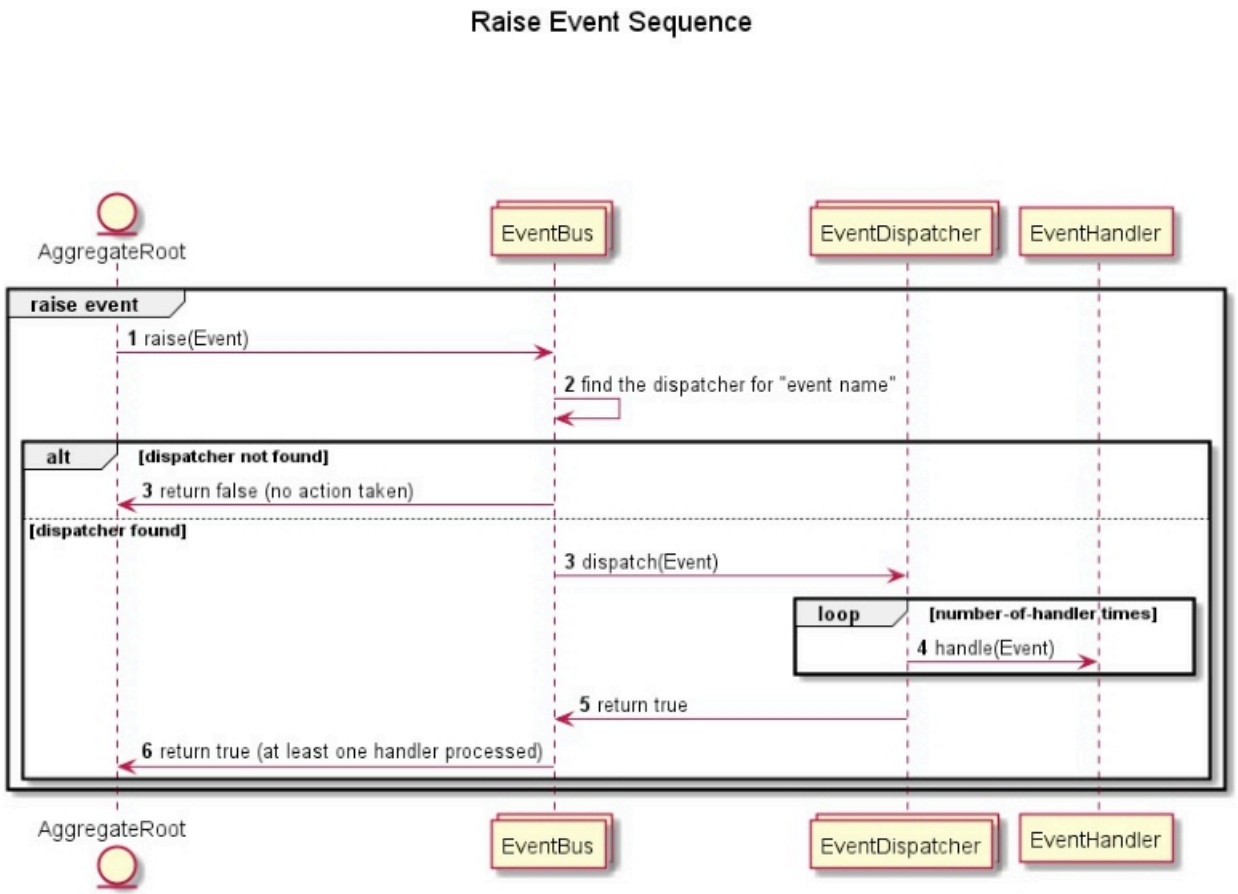
3. Add events to a collection and then publish

Static Class

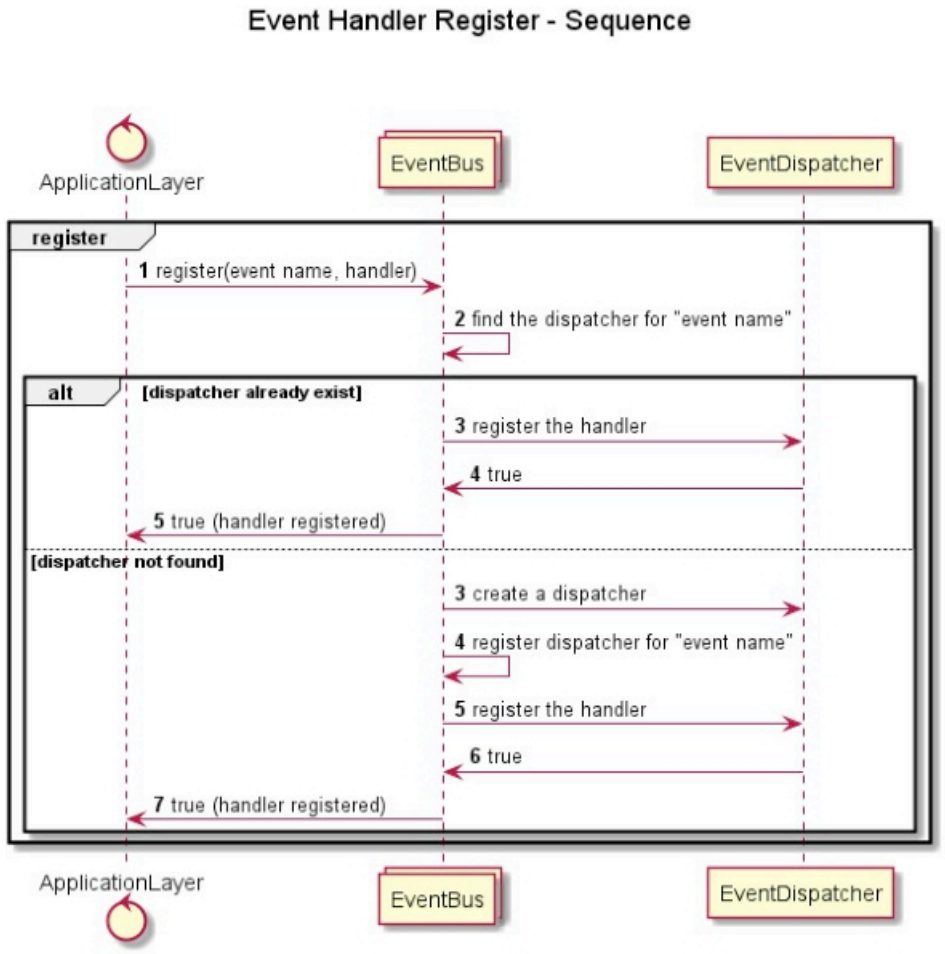




Raise event Sequence



Register Handler Sequence



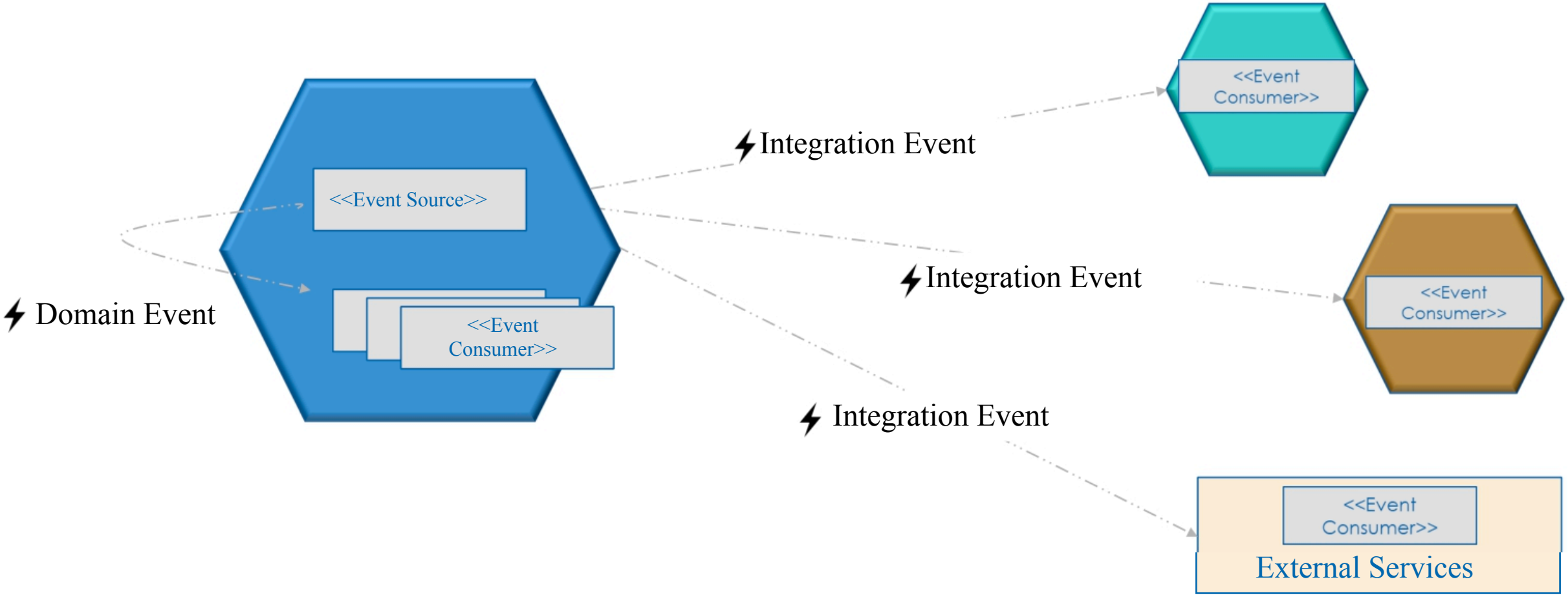
Integration Events

Raising and Handling Integration events

1. Integration versus Domain Events
2. Consumer BC perspective
3. Enhance "Loan Management BC Model"

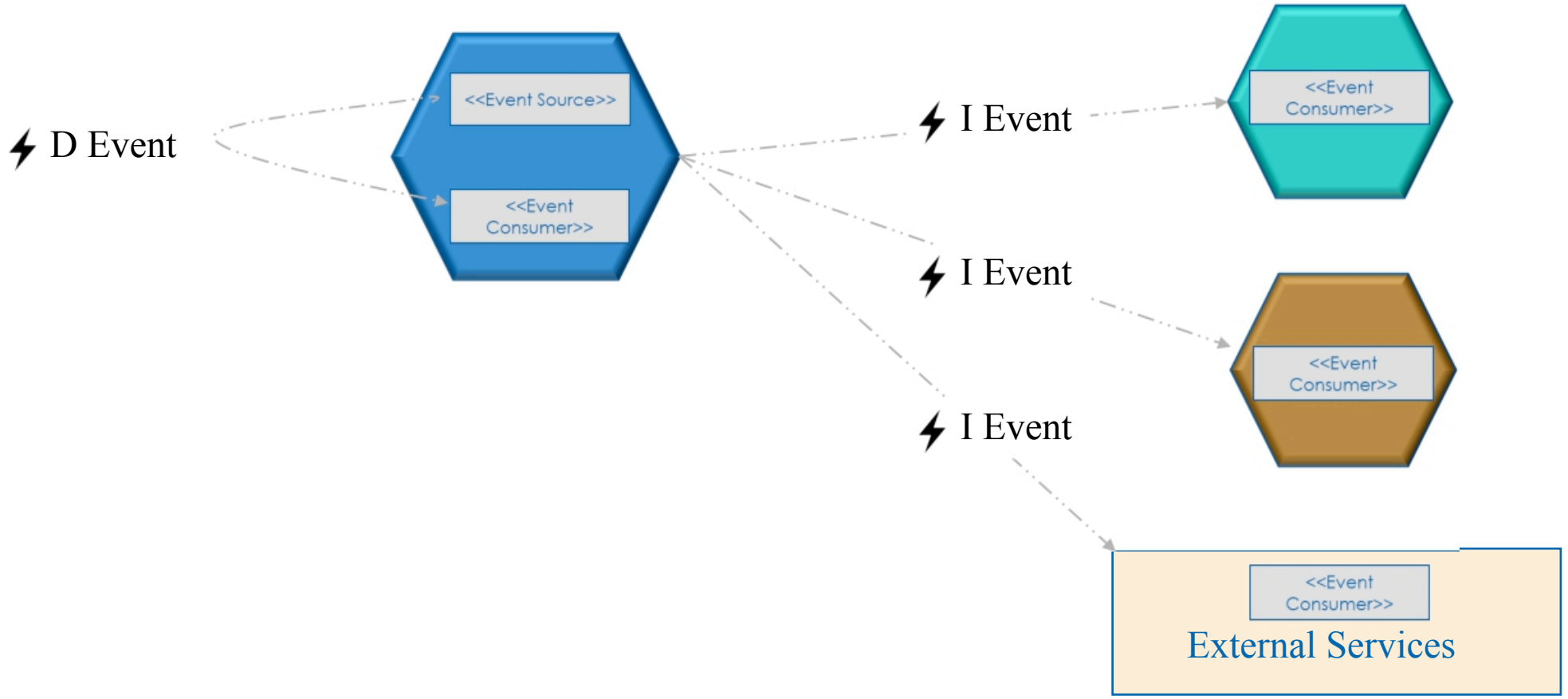
Integration Events

Event Consumers are outside the source Bounded Context



Integration Events

Event Consumers are outside the source Bounded Context



Integration Event

An Integration Event is a message that informs components outside of the source Bounded Context that something of significance has happened

An Integration Event does NOT lead to any state changes in the source Bounded Context.

Domain versus Integration Events

Domain

Within a BC | Microservice

State changes in BC

Direct function calls

Synchronous calls

Modeled as part of BC model

Integration

Between BC | BC and External services

NO state change in source BC

Must be a Network protocol

Asynchronous preferred

Consumer decides

Integration Events Communication

Asynchronous is preferred

- To achieve HIGHER levels of decoupling
- Future extensibility i.e., add new consumers
- Enables one -to -many

Technology examples

Messaging technology is commonly used

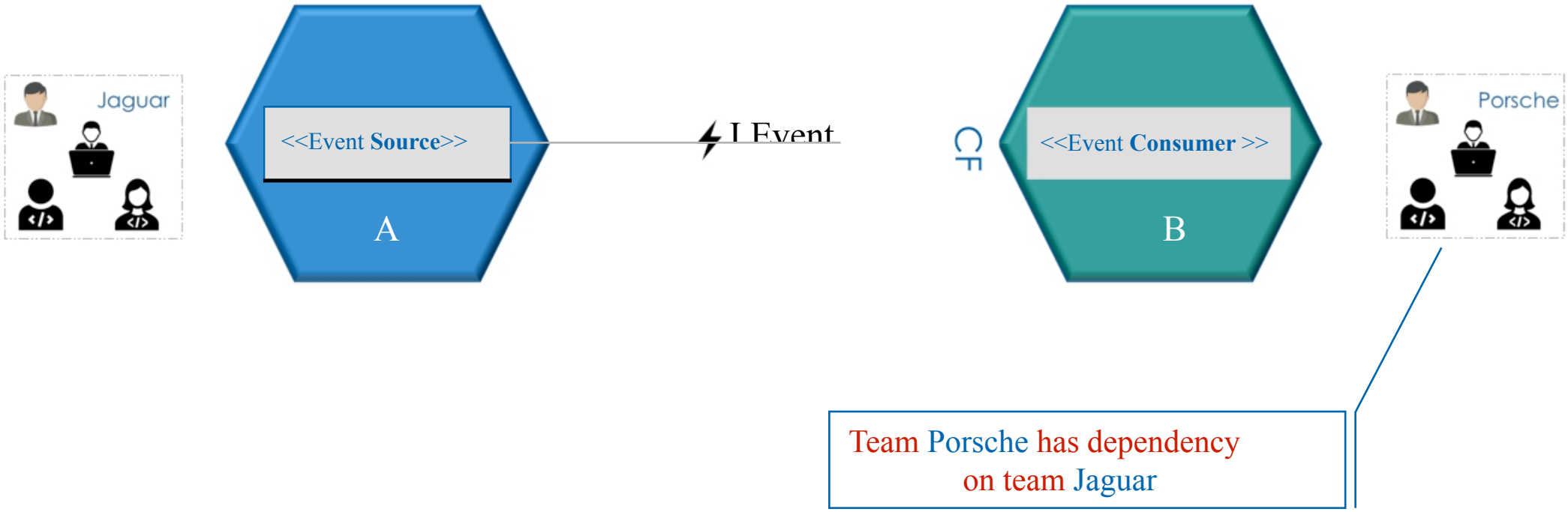
Relationship - Domain & Integration Events

Domain event may be published as Integration event

- Both events defined as part of the model for the BC
- Semantically same
- Publishing mechanism is different
  - *OR same if external messaging used for both event types*

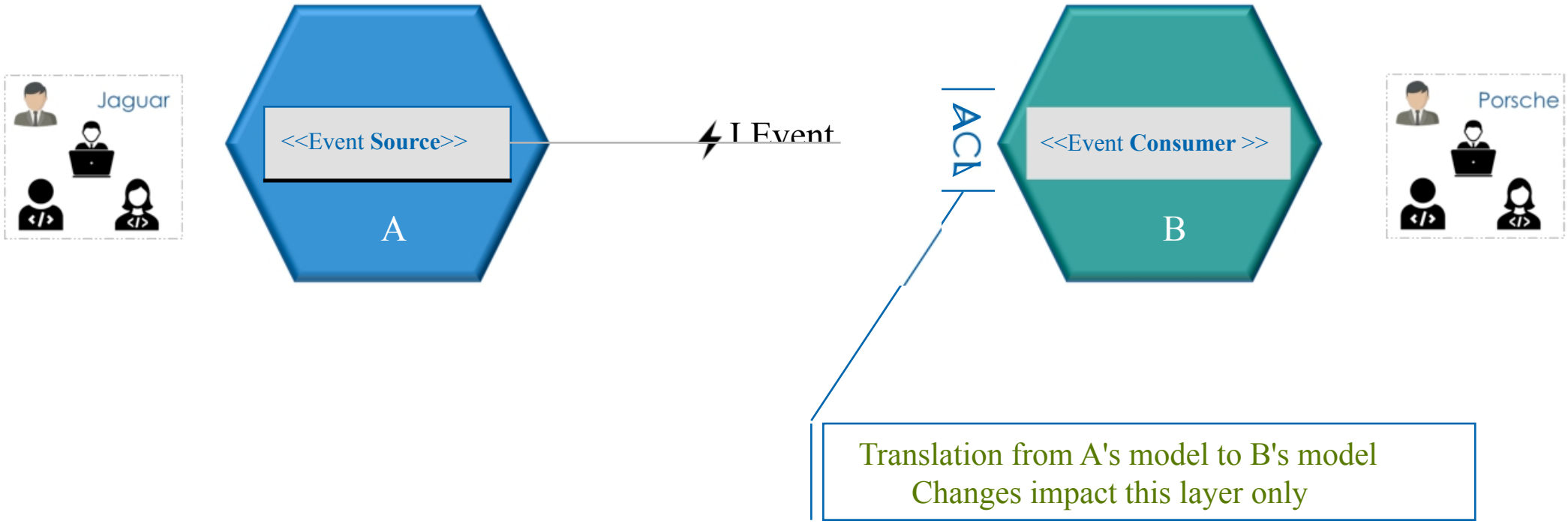
Integration event consumer

Consumer of Integration event may be a Conformist



Integration event consumer

Consumer may leverage Anti Corruption Layer

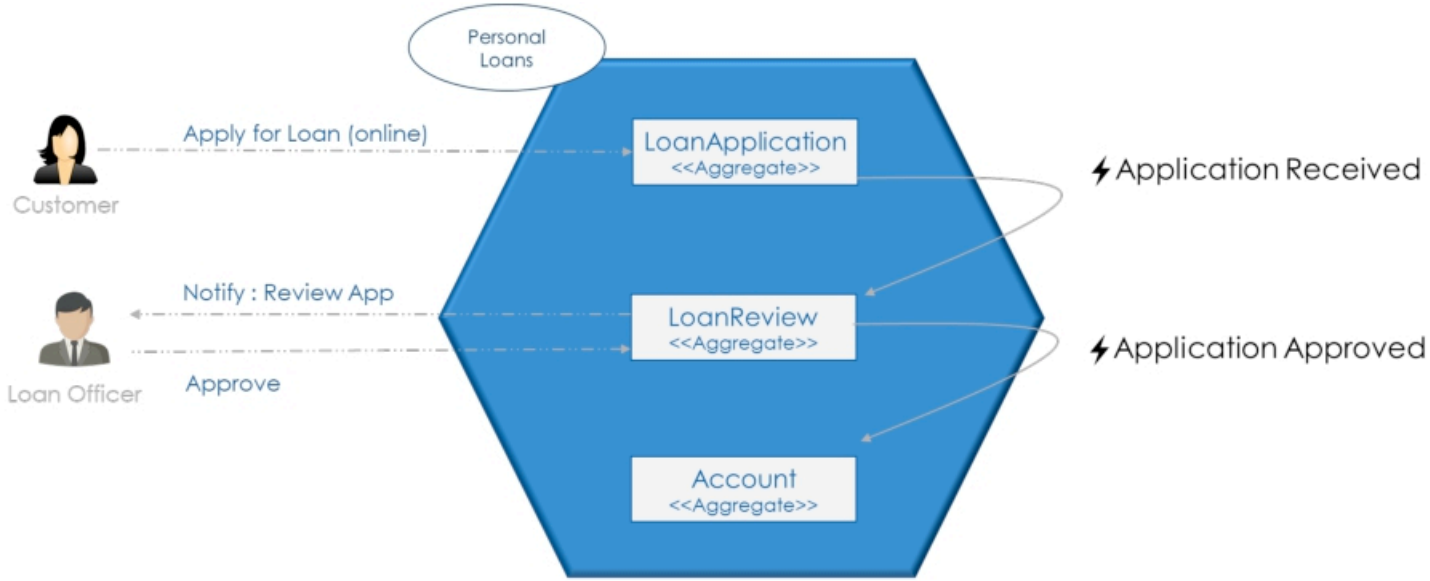


Example : Personal Loan Microservice

New requirements

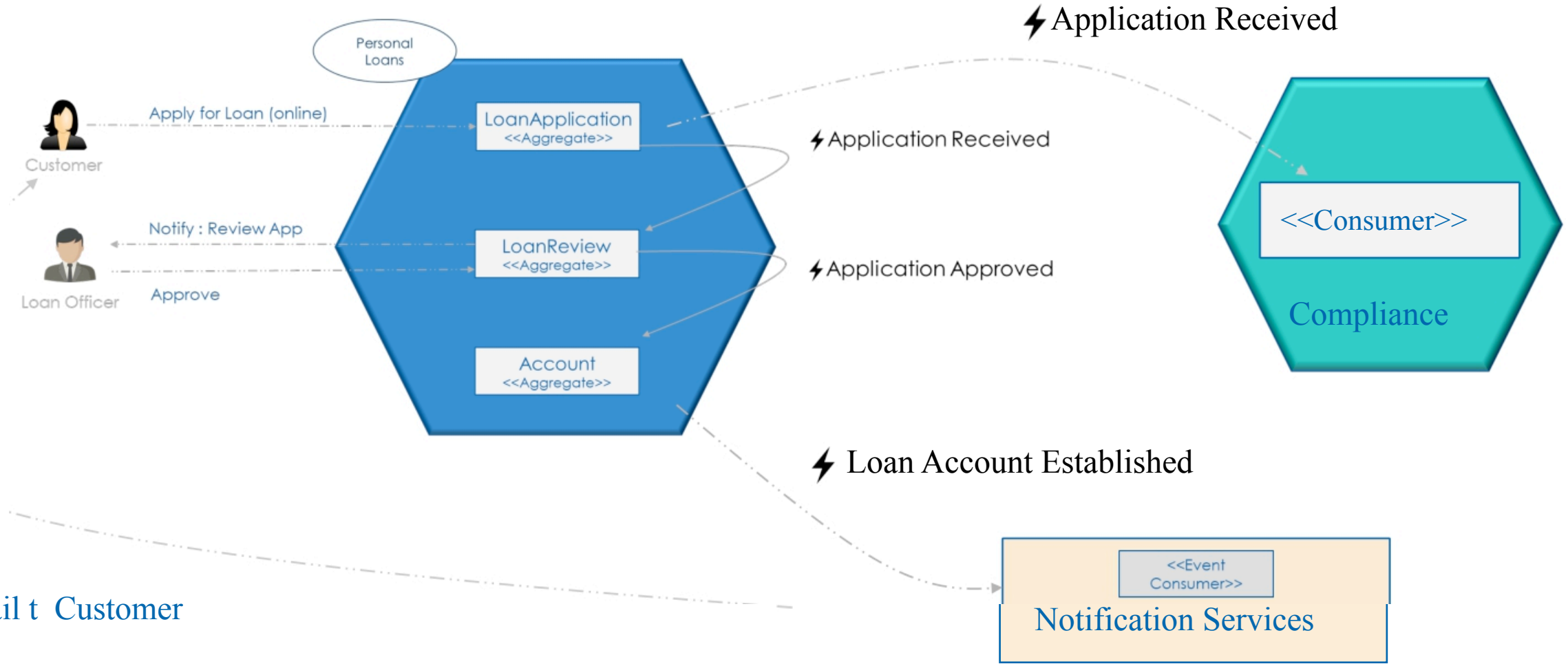
1. Inform Compliance BC/MS when a new Application is Received

2. Notify customer by Email when the Loan account is setup



Example : Personal Loan Microservice

External consumers of Integration events



Integration Events & Transactions

MUST ensure that event is raised ONLY after a successful commit

Quick Review

Domain events may be published as Integration events

Integration events published Asynchronously

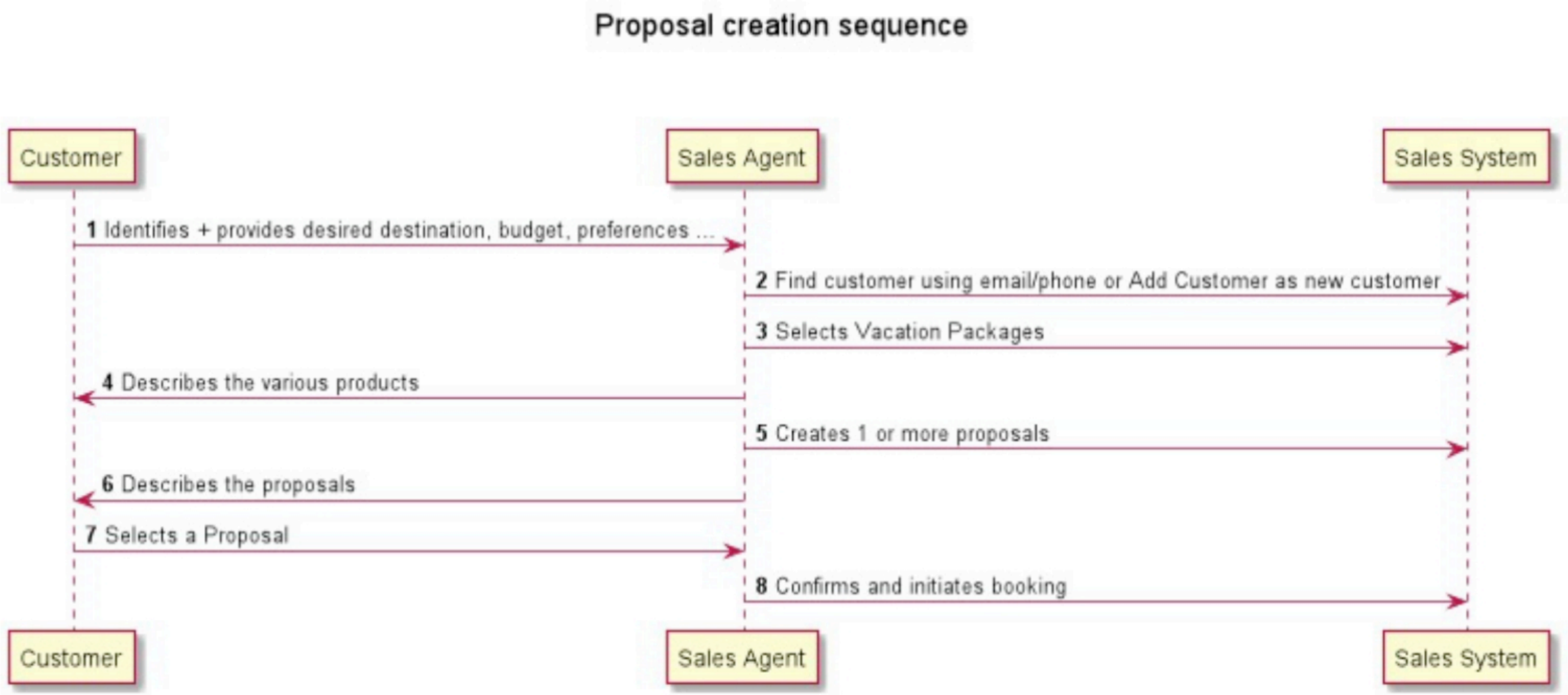
Consumer Bounded Context may use

Exercise : ACME Sales Events

Processing of Booking Confirmation

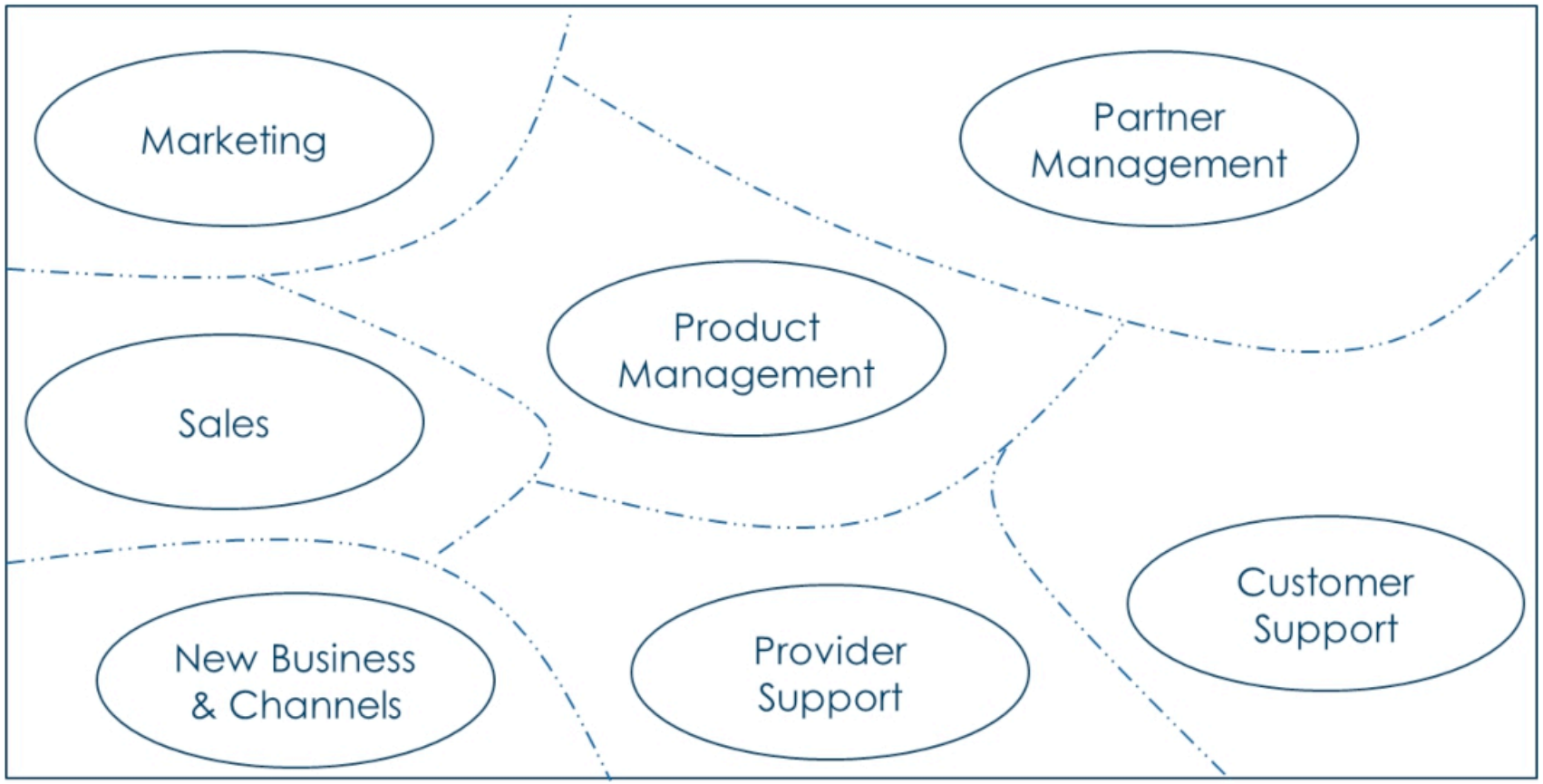
1. Extend the model
2. Introduce an Integration Event
3. Message Infrastructure Service

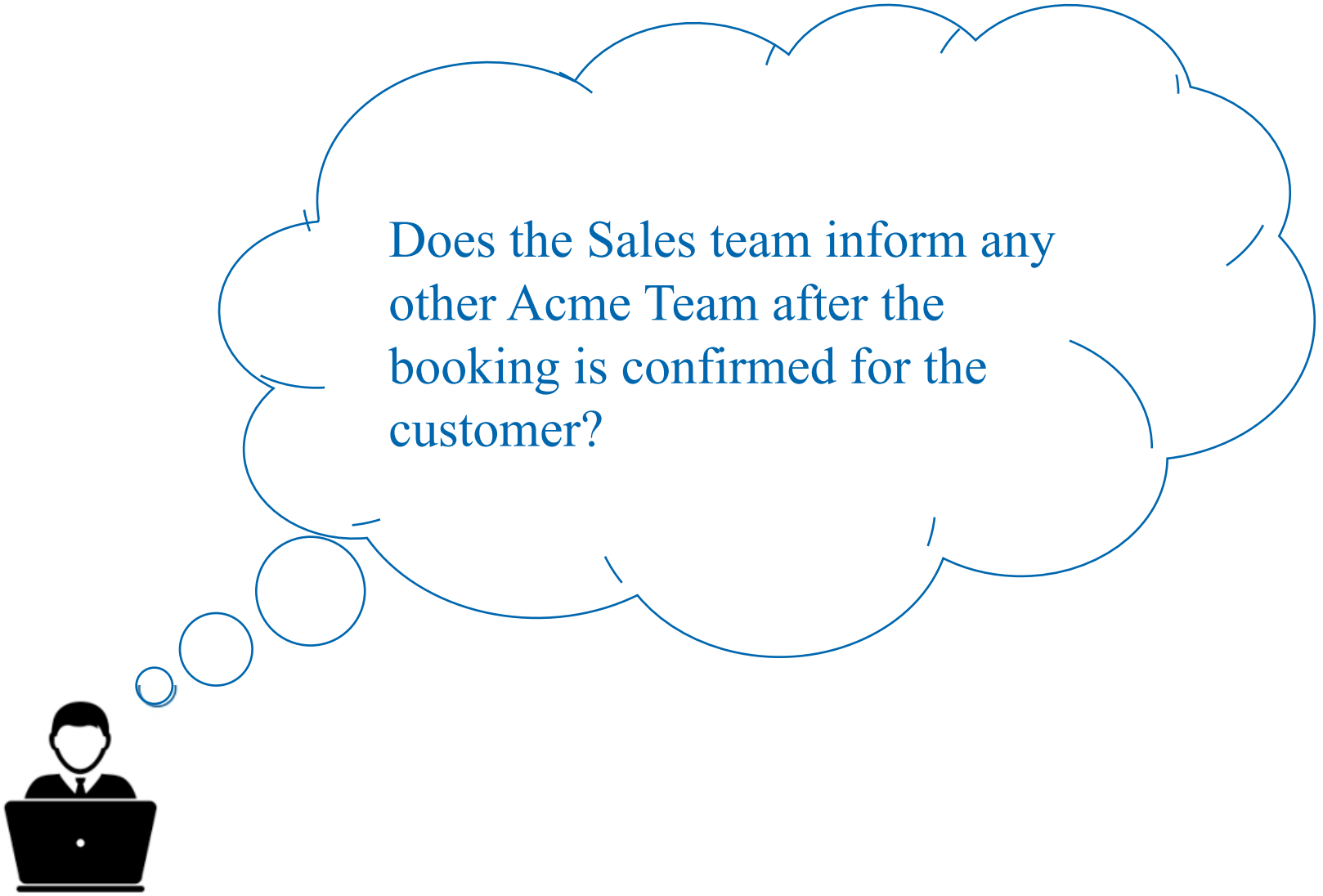
Sequence Testing using FAKE repo implementation



Business Capabilities | Organization structure

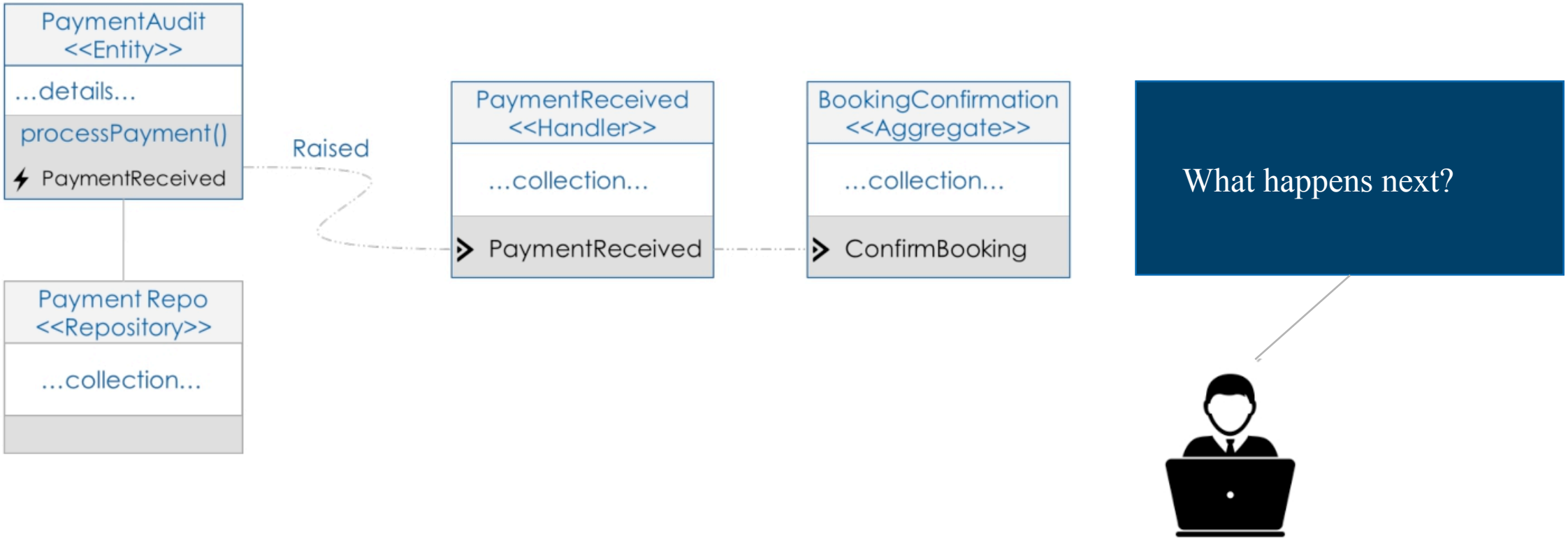
Support context split into 2 independent Bounded Contexts




Does the Sales team inform any other Acme Team after the booking is confirmed for the customer?

ACME Travel

Objective: Understand actions taken on Booking Confirmation



John, Travel Advisor

When the Booking is confirmed we do 2 things:

BookingConfirmed

1. Send an Email to customer with confirmation details

2. Update our daily & weekly reports for different ACME departments

John, Travel Advisor

Creating reports is a tiring work as every department wants these reports to be in a certain format for same data. We are always trying to catch up; sometimes we get blamed by other departments for things that go wrong in those departments - mainly for either not sending the reports on time or missing/bad data in the reports.

Interestingly these departments are just interested in the data and not the reports.

Let me give you some examples.

Booking Report

Daily

Revenue Report

Daily

Forecasting

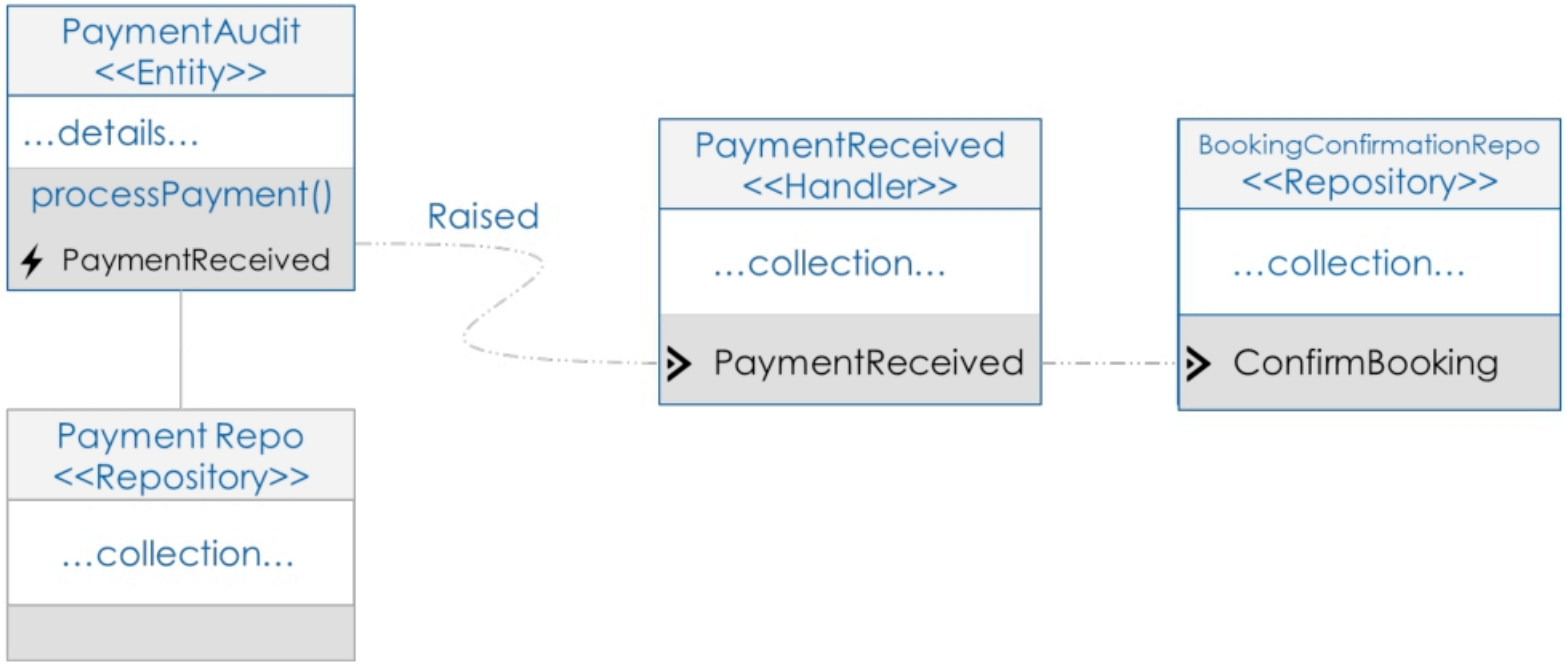
Weekly

And then there are ad - hoc data that departments ask for !!!

New Requirements

1. Send an Email to customer with confirmation details

2. Update our daily & weekly reports for different ACME departments

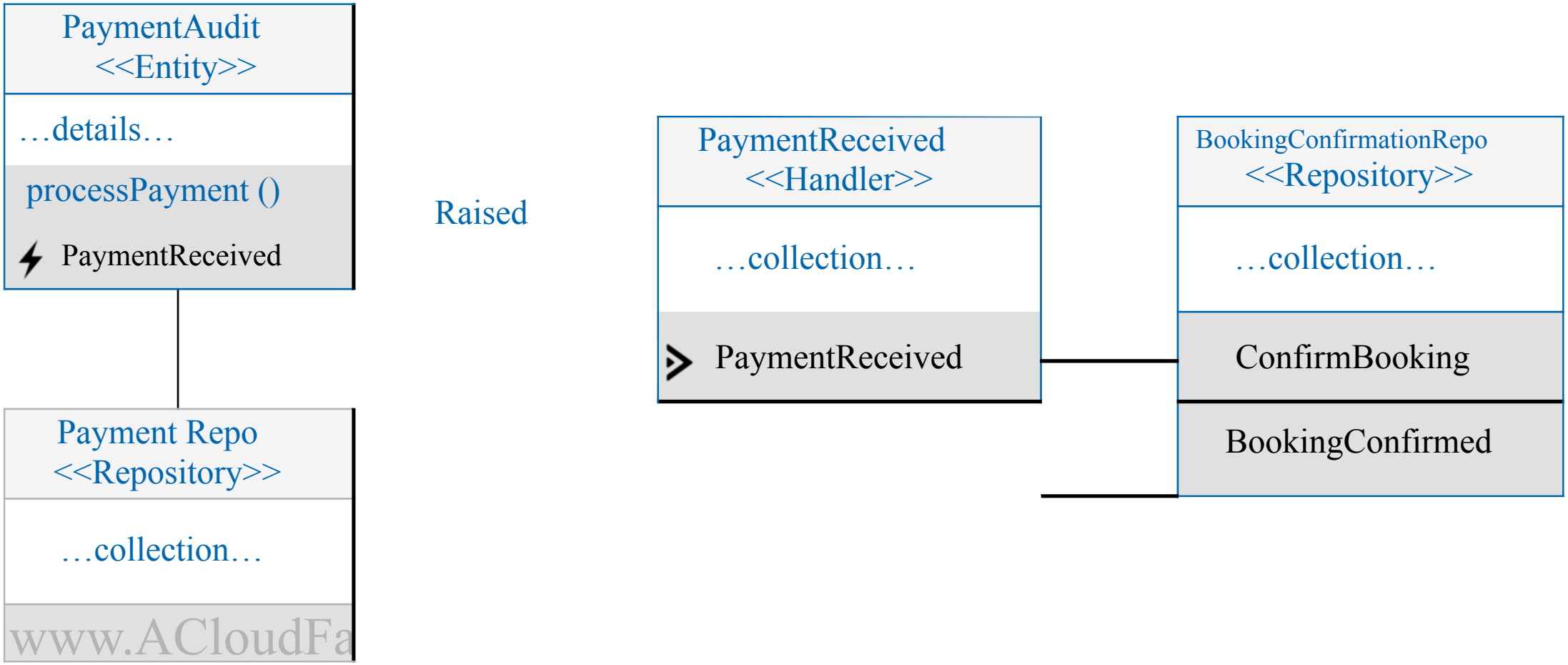


How do we address these requirements?

Booking Confirmation Event

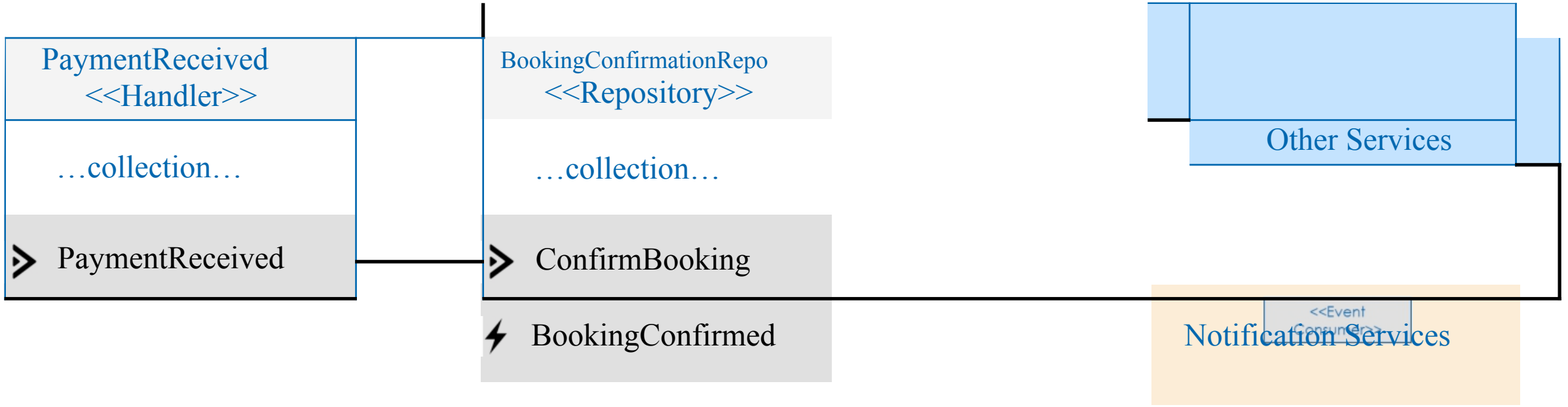
Introduce a new Integration Event

⚡ BookingConfirmed



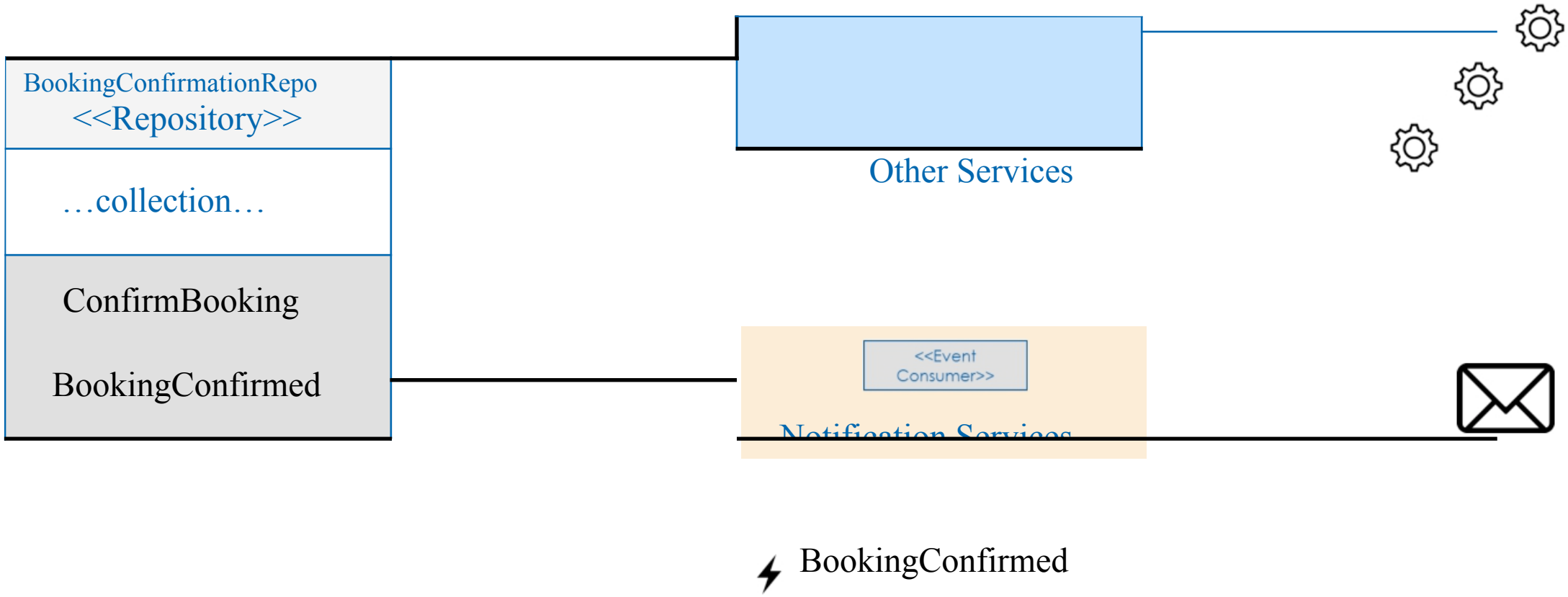
Booking Confirmation Event

Other Microservices & Notification service will subscribe to event



Booking Confirmation Event

All subscribers receive state updates in Realtime

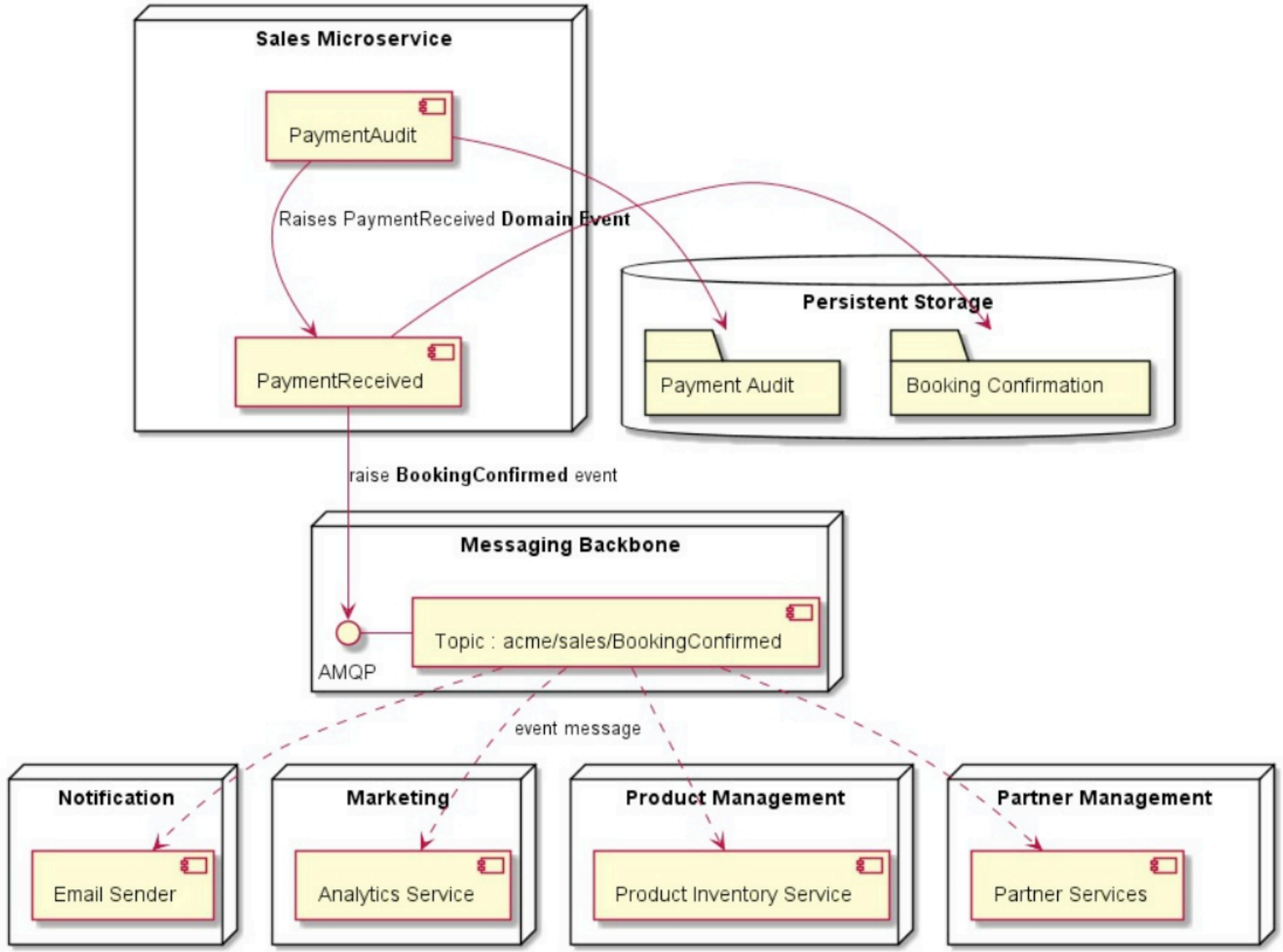


Component Diagram

Sales Microservice

RabbitMQ

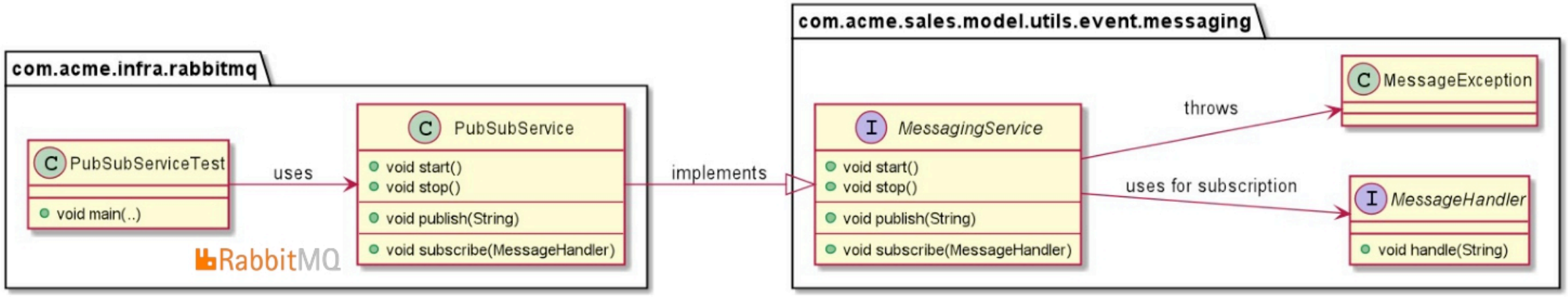
Subscribers



Messaging Infrastructure Service

Refer: uml /events/ messaging.infra.service.puml

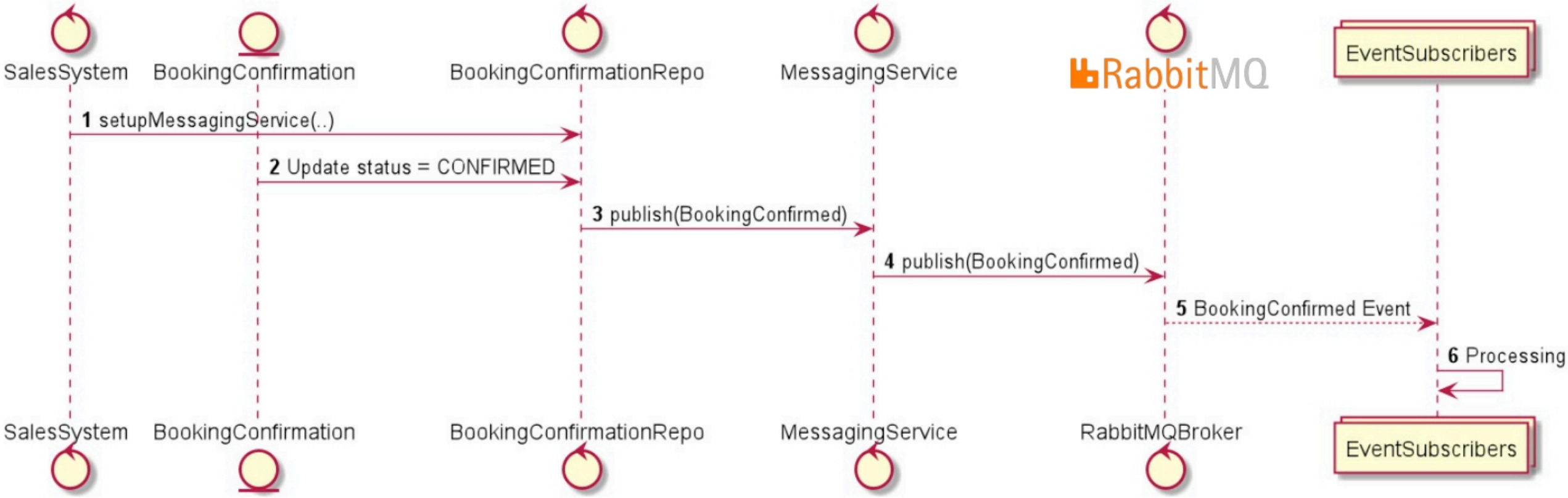
Keep the model independent of messaging technology



Concrete implementation for RabbitMQ

Infrastructure Service for messaging

Booking Confirmed Event - Sequence



Refer: uml /events/ bookingconfirmed.sequence.puml

Quick Review

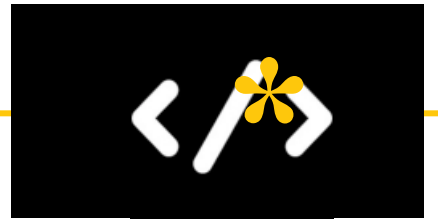
Repository is a good place to raise events

Event Raised ONLY if update to storage is successful

Message Infrastructure Service keeps the model tech agnostic

Hands On - ACME Integration Event (1 of 2)

Demonstrates implementation of BookingConfirmed Event



1. Setup exchange and queue on RabbitMQ
2. Test the Messaging Infrastructure Service

Objective

Part -1 Setup the Exchange & Topic on RabbitMQ Broker

Part -2 Test the Infrastructure Messaging Service

Part -3 Walkthrough & test implementation of BookingConfirmed event

Part - 1 Setup the Exchange & Topic on RabbitMQ Broker

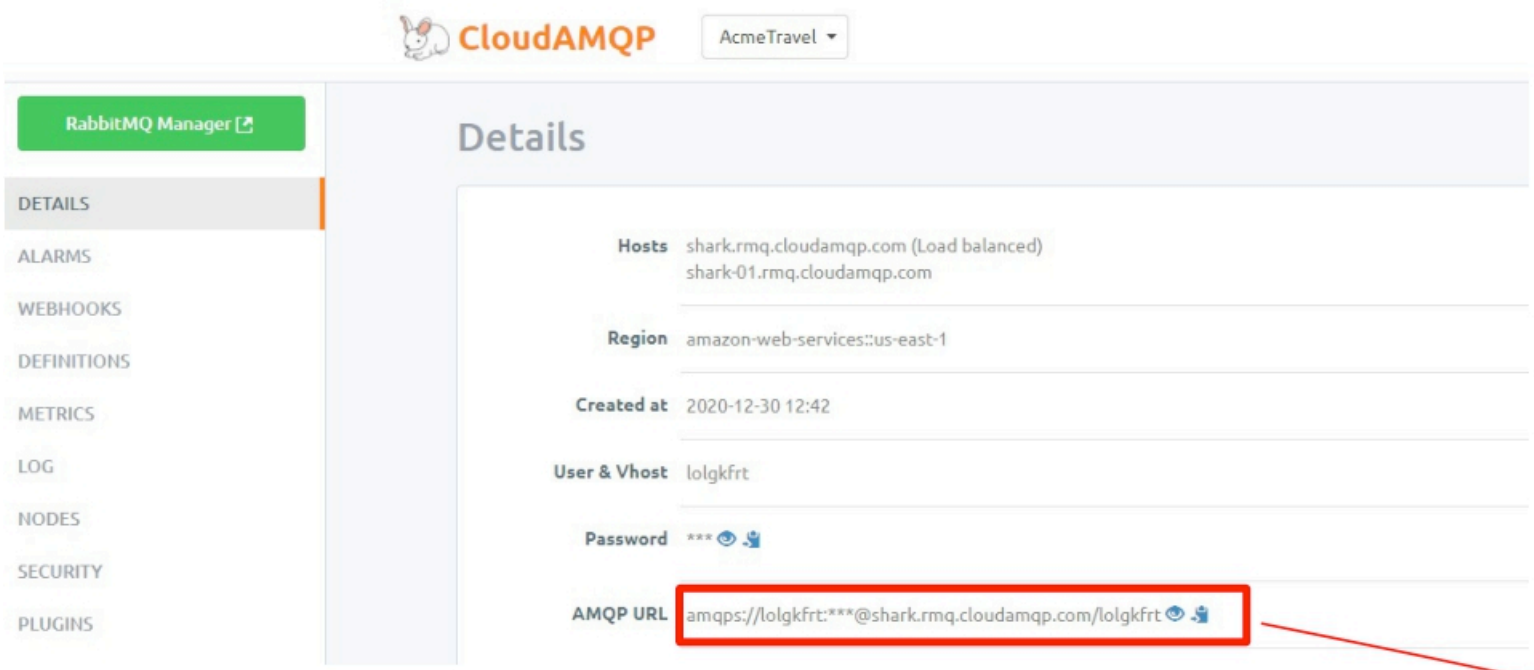
Step -1 Setup an Exchange acme.sales.topic of type Topic

Step -2 Setup a Queue with name = email.notifications

Step -3 Bind the Queue with routing key = acme.sales.bookingconfirmed

Step -4 Test by Publishing message to Exchange ; should end up Queue

Setup the AMQP_URL - PubSubServiceTest



Replace the AMQP_URL !!!

```
public class PubSubServiceTest {

    // MUST replace this URL to your instance on RabbitMQ
    private static final String AMQP_URL="amqps://lolgkfrt:JrxrfAVFIzKhztFY4NCLTtyRwvEIaFEX@shark.rmq.cloudamqp.com/lolgkfrt";

    private static final String AMQP_EXCHANGE = "acme.sales.topic";
    private static final String AMQP_TOPIC = "acme.sales.bookingconfirmed";


    public static void main(String[] args) throws  Exception{
```

Part - 2 Test the Messaging Infrastructure Service

Step -1 Purge the Queue email.notifications

Step -2 Setup a PubSubTest to publish a message; Run class

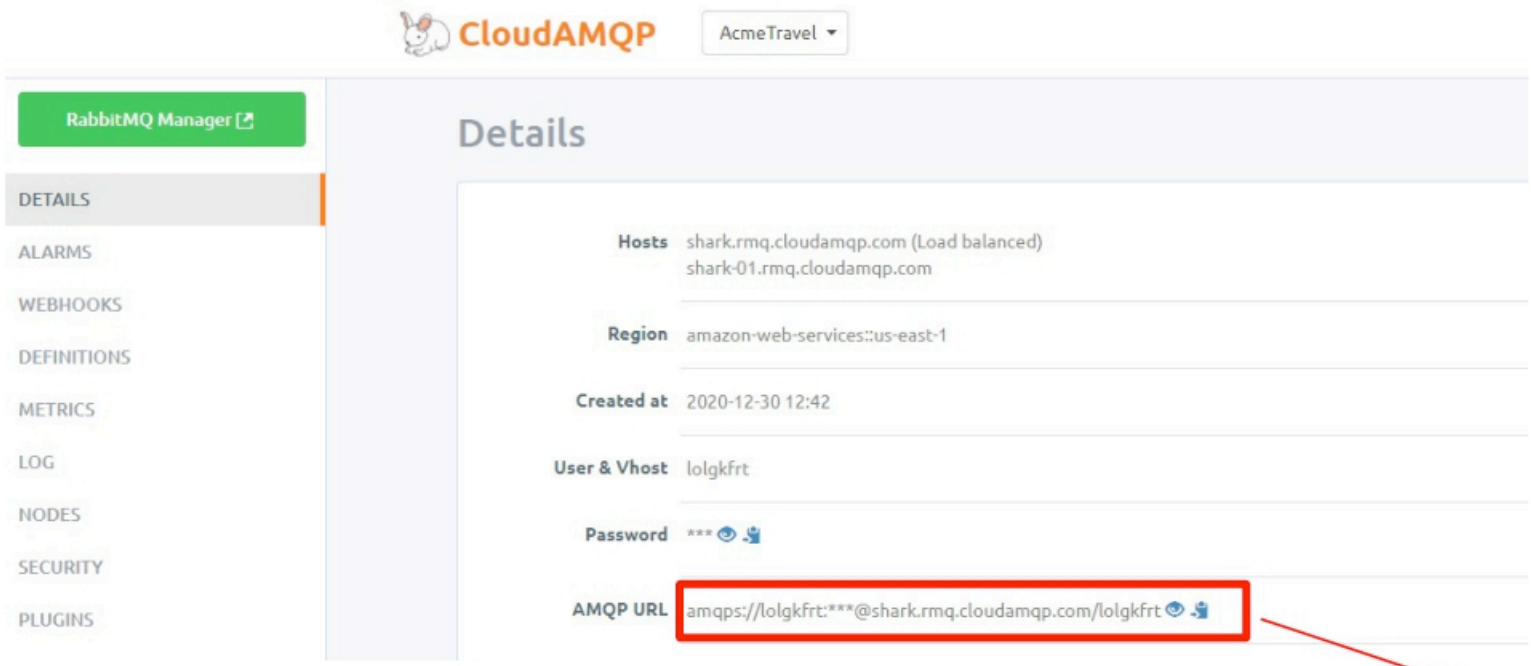
Message should show up in Queue email.notifications

Step -3 Setup a PubSubTest to subscribe for messages; Run class

Publish to Exchange/UI acme.sales.topic acme.sales.bookingconfirmed

Message should be printed on console

Setup the AMQP_URL - PaymentAuditTest class

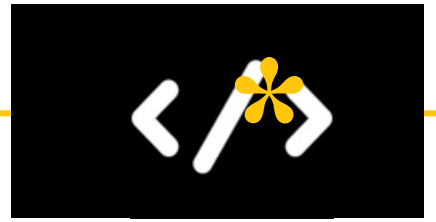


Replace the AMQP_URL !!!

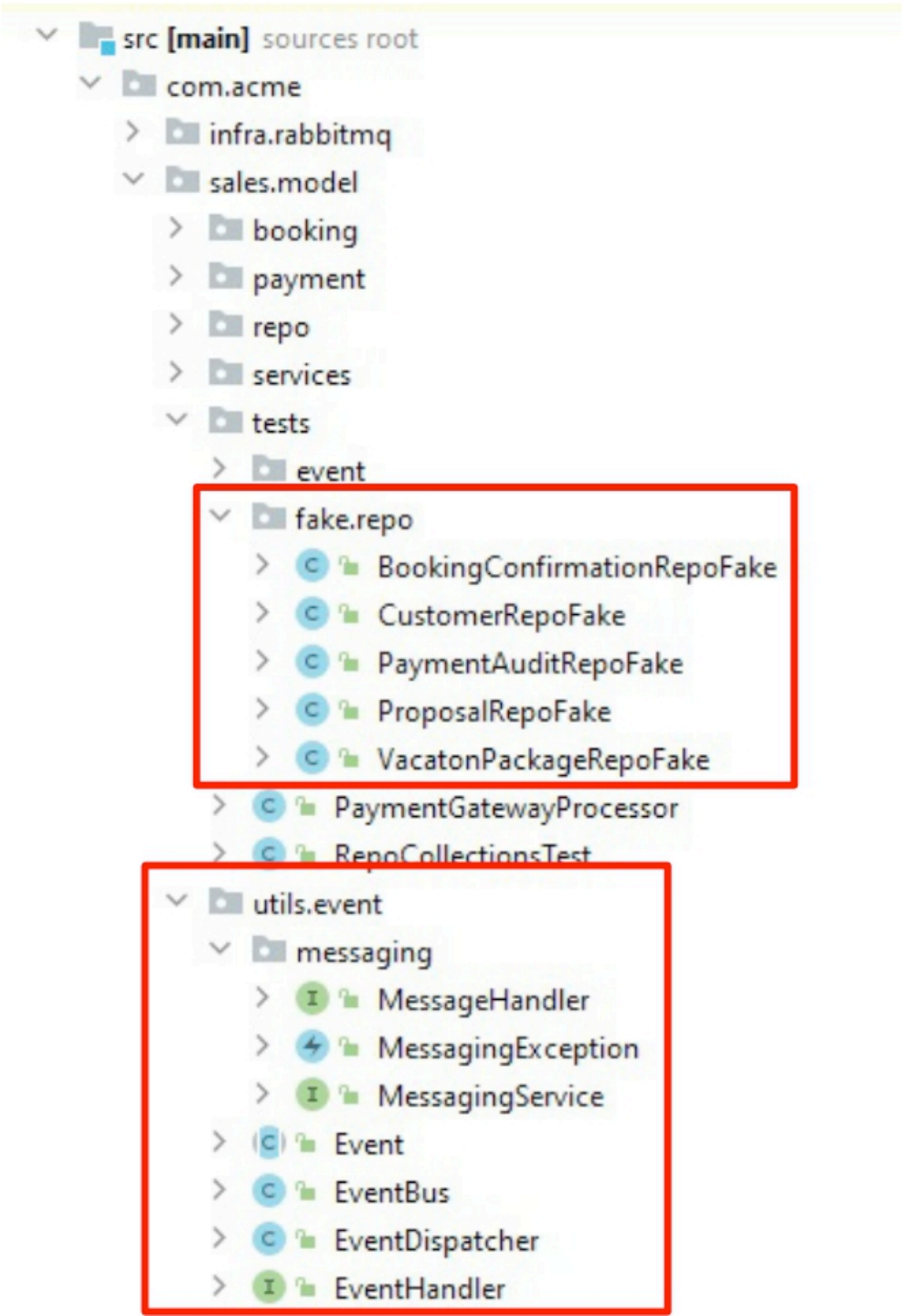
```
public class PaymentAuditTest {

    // MUST replace this URL to your instance on RabbitMQ
    final private static String AMQP_URL="amqps://lolgkfrt:JrxrfAVFIzKhztFY4NCLTtyRwvEIaFEX@shark.rmq.cloudamqp.com/lolgkfrt";

    private static MessagingService messagingService;
```

Hands On - ACME Integration Event (2 of 2)

Demonstrates implementation of BookingConfirmed Event



1. Walkthrough of Repository implementation
2. Walkthrough of test code
3. BookingConfirmed event in action



Git Branch : events

Part - 3

Step -1 Walkthrough of the test setup

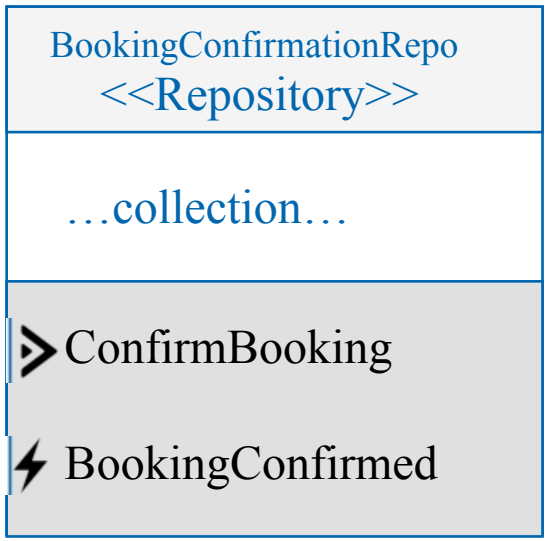
Step -2 Setup the PaymentAuditTest class to use Messaging

Step -3 Purge the Queue email.notifications

Step -4 Run the PaymentAuditTest class

Message should show up in Queue email.notifications

BookingConfirmationRepoFake raises the event



BookingConfirmationRepoFake

setupMessagingService (…)

updateState (…)

Booking Confirmed Event Test - Activity

Prerequisite:

MUST have the exchange setup on RabbitMQ

MUST have the correct AMPQ_URL in the PaymentAuditTest class

Refer: uml /tests/ bookingconfirmed.activity.puml