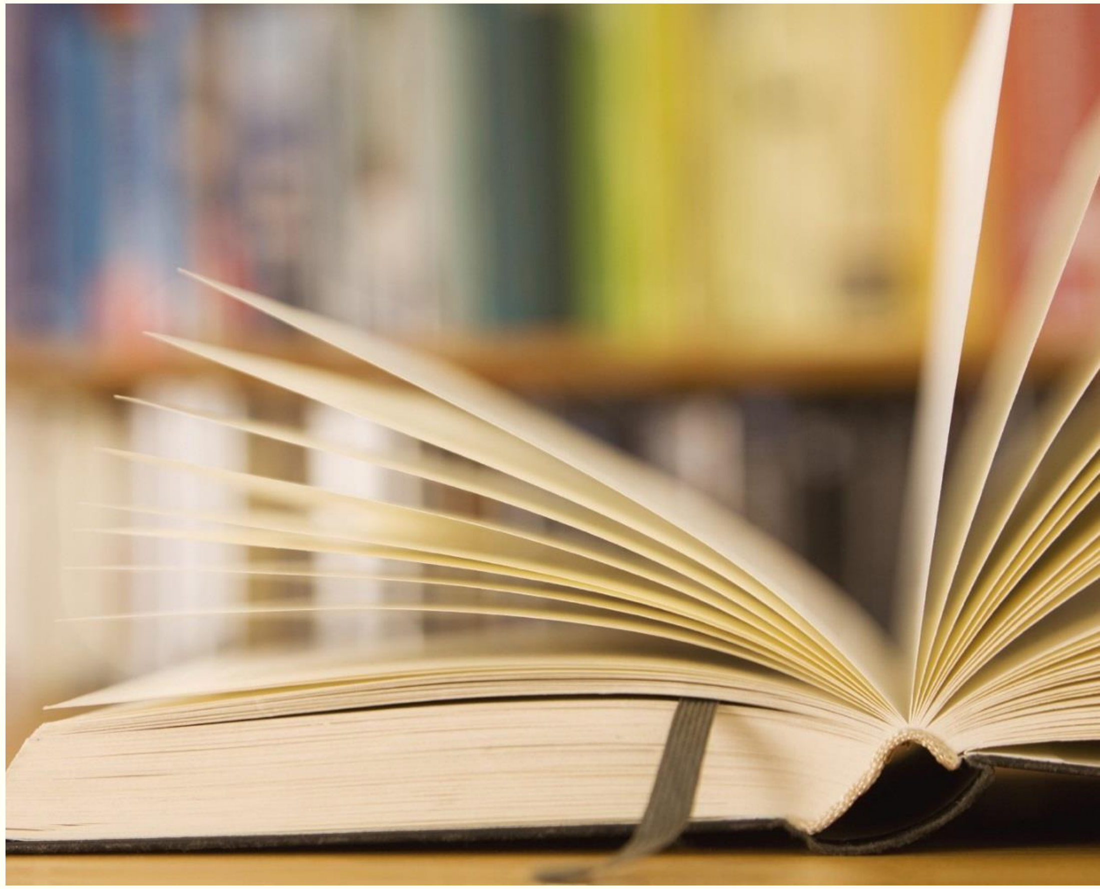
ENTERPRISE ARCHITECTURE

Najeeb Najeeb, PhD

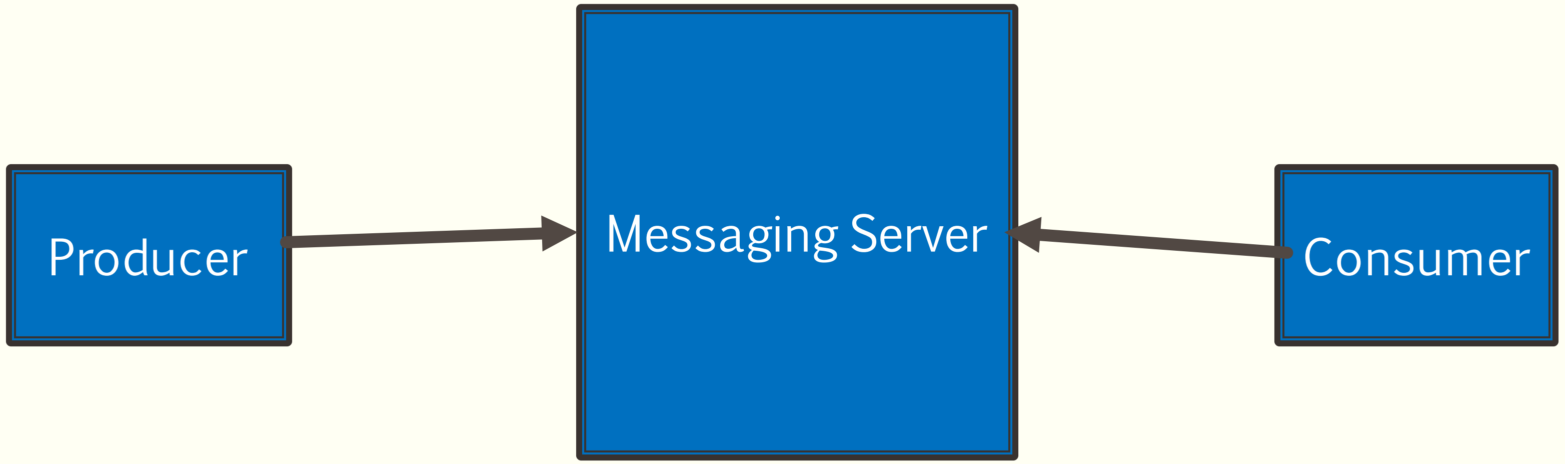
Version 2.2 ©2022



LESSON 12 SPRING JMS

Messaging

- MOM : Message Oriented Middelware
- MQ: ActiveMQ, SonicMQ, WebsphereMQ, TIBCO MQ

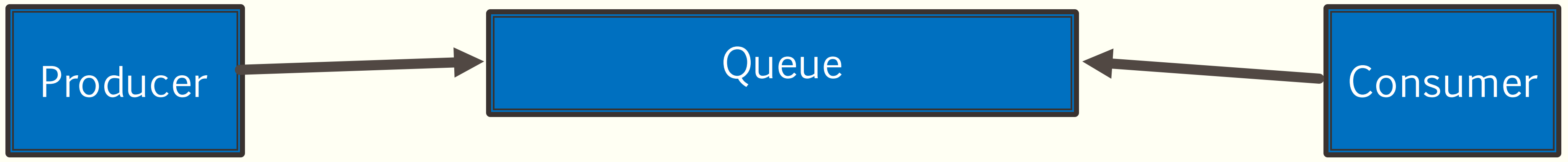


Why Messaging

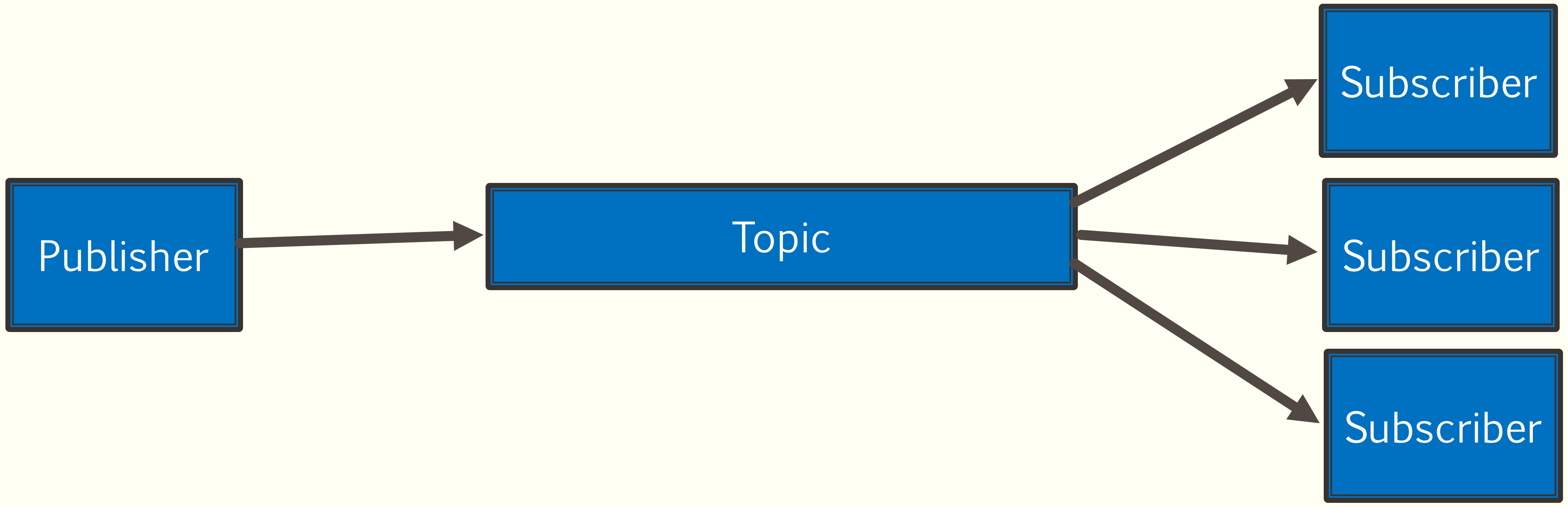
- Heterogenous Integration
  - Different application to communicate through a Queue
  - Application can be using different technologies
- MOM exposes an API for each application to work with
- Very helpful for microservice and SOA
- Loosely Coupled
  - Without Messaging communicating application need not know of each other
- More reliable
  - The Producer and Consumer do not have to be available at the same time
  - Less probability of messages to be lost (REST is over tcp/ip)
- Reduces System Bottlenecks
- Asynchronous communication
- Scalability (by increasing consumers)
- Flexibility & Agility

Model

- P2P



- Pub-Sub



How to Work with MQ

- Prior to JMS you need to know the API of the MQ API
  - ActiveMQ
  - WebsphereMQ
  - …
- JMS: Java Messaging Service
- Every MQ that passes JMS TCK is a JMS implementation and can be used :)
- JMS ==> Messaging Queues
- JDBC ==> Databases

ACTIVEMQ

Install

- https://activemq.apache.org/
- Download and unzip
- cd bin
  - artemis create cs544broker
    - Username: ea
    - Password: cs544
- .\cs544broker\bin\artemis run
  - HTTP Server started at http://localhost:8161
- localhost:8161
  - Management Console
- artemis stop

SPRING JMS

Spring JMS

- Starter
  - Web
  - Messaging > Artemis
- @SpringBootApplication
- @EnableJms
- public class SpringJmsApplication {

- Configure Spring
- springjms.cs544Queue=cs544Queue

JMS Code!

Producer

```
@Component
public class Sender {
    @Autowired
    private JmsTemplate jmsTemplate;
    @Value(value = "${springjms.cs544Queue}")
    private String queueName;
    public void send(String message) {
        jmsTemplate.convertAndSend(queueName, message);
    }
}
```

Consumer

```
@Component
public class Receiver {
    @JmsListener(destination = "${springjms.cs544Queue}")
    public void receive(String message) {
        System.out.println("Received Message > "+message);
    }
}
```

REST Controller

```
▪ @RestController
▪ public class MessageController {
▪         @Autowired
▪         private Sender sender;
▪         @GetMapping("/message/{message}")
▪         public void sendMEssage(@PathVariable String message) {
▪                 sender.send(message);
▪         }
▪ }
```

Using send not ConvertAndSend

lambda

```
public void send(String message) {
    MessageCreator messageCreator= s -> {
        return s.createTextMessage(message);
    };
    jmsTemplate.send(queueName, messageCreator);
}
```

code

```
public void send(String message) {
    MessageCreator messageCreator= new
MessageCreator(){
        @Override
        public Message createMessage(Session session)
throws JMSException{
            return session.createTextMessage(message);
        }
    };
    jmsTemplate.send(queueName, messageCreator);
}
```

Receiver as a Different Application

- Take the Receiver code and put it in a Spring JMS application by itself

- Switch to Pub-Sub
- spring.jms.pub-sub-domain=true

Spring JMS

- JmsTemplate
- send
- receive
- ConvertAndSend
  - String to TextMessage
  - Object to ObjectMessage
- Listeners
- MDPs: Message Driver POJOs