

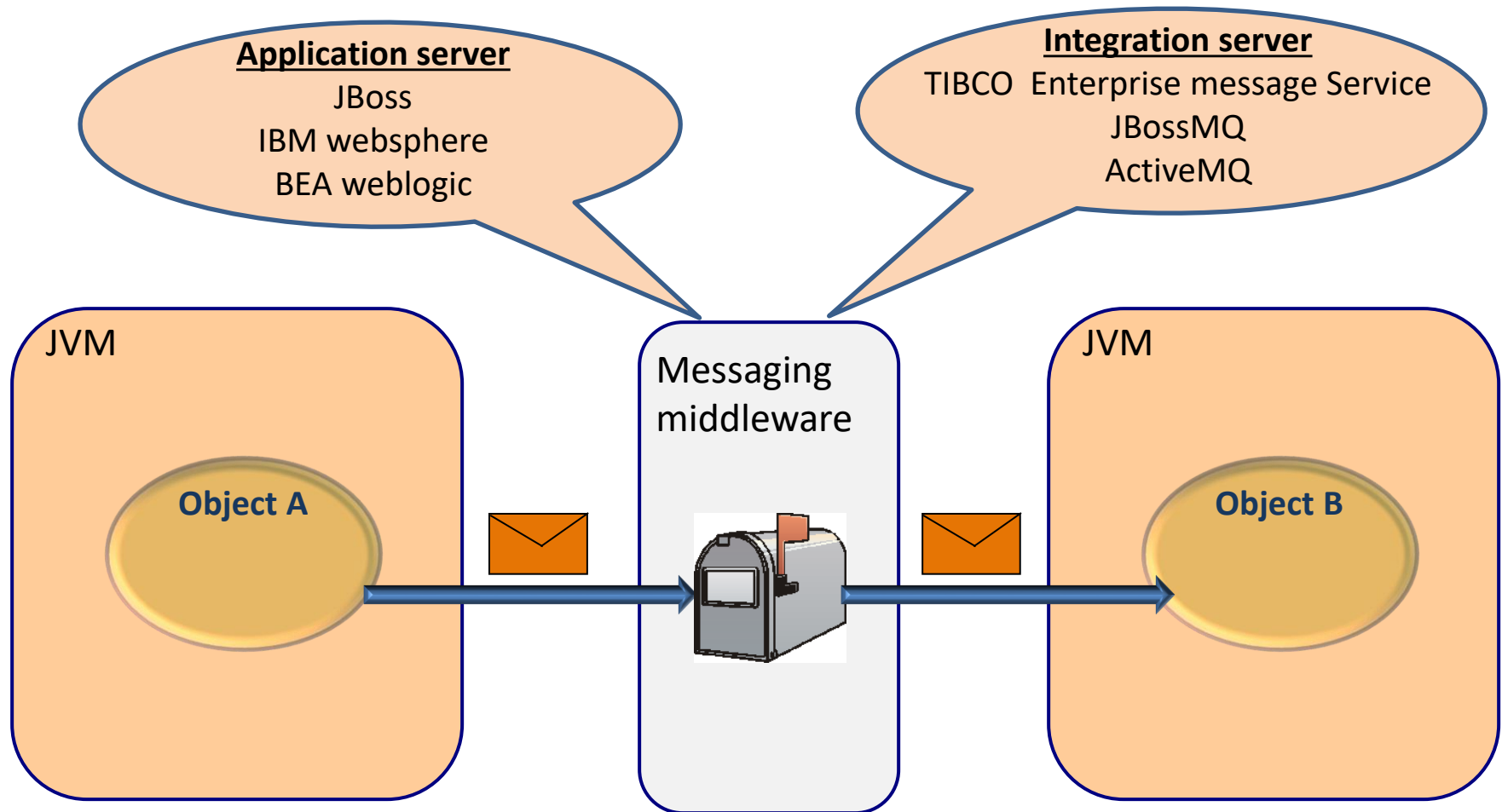
CS544

LESSON 11

MESSAGING

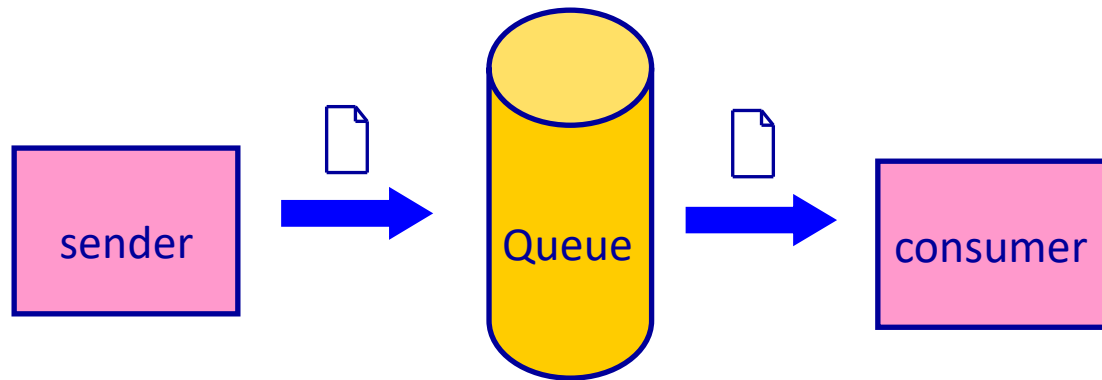
Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
November 28 Lesson 1 Introduction Spring framework Dependency injection	November 29 Lesson 2 Spring Boot AOP	November 30 Lesson 3 JDBC JPA	December 1 Lesson 4 JPA mapping 1	December 2 Lesson 5 JPA mapping 2	December 3 Lesson 6 JPA queries	December 4
December 5 Lesson 7 Transactions	December 6 Lesson 8 MongoDB	December 7 Midterm Review	December 8 Midterm exam	December 9 Lesson 9 REST webservices	December 10 Lesson 10 SOAP webservices	December 11
December 12 Lesson 11 Messaging	December 13 Lesson 12 Scheduling Events Configuration	December 14 Lesson 13 Monitoring	December 15 Lesson 14 Testing your application	December 16 Final review	December 17 Final exam	December 18
December 19 Project	December 20 Project	December 21 Project	December 22 Presentations			

Java Message Service (JMS)



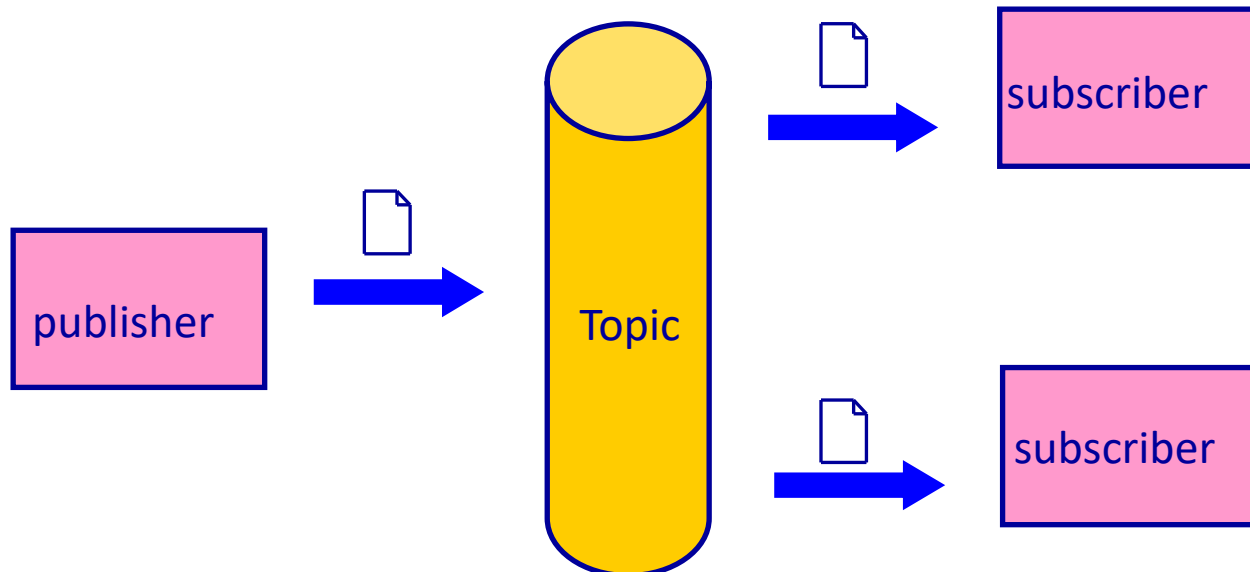
Point-To-Point (PTP)

- A dedicated consumer per Queue message



Publish-Subscribe (Pub-Sub)

- A message channel can have more than one '*consumer*'
 - Ideal for broadcasting



JMS sender

```
//Lookup a ConnectionFactory with JNDI
QueueConnectionFactory queueConnectionFactory = (QueueConnectionFactory)
jndiContext.lookup("MyJMS Connection Factory");
// Lookup a Destination with JNDI
Queue queue = (Queue) jndiContext.lookup("MyJMSQueue");
// Use the ConnectionFactory to create a Connection
QueueConnection queueConnection = queueConnectionFactory.createQueueConnection();
// Use the Connection to create a Session
QueueSession queueSession =
    queueConnection.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);
// Use the Session to create a MessageProducer for this queue
QueueSender queueSender = queueSession.createSender(queue);
// Use the Session to create a Message
TextMessage message = queueSession.createTextMessage();
message.setText("Hello World");
// Use the MessageProducer to send the Message
queueSender.send(message);
```

JMS receiver

```
//Lookup a ConnectionFactory with JNDI
QueueConnectionFactory queueConnectionFactory = (QueueConnectionFactory)
jndiContext.lookup("MyJMS Connection Factory");
// Lookup a Destination with JNDI
Queue queue = (Queue) jndiContext.lookup("MyJMSQueue");
// Use the ConnectionFactory to create a Connection
QueueConnection queueConnection = queueConnectionFactory.createQueueConnection();
// Use the Connection to create a Session
QueueSession queueSession =
    queueConnection.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);
// Use the Session to create a MessageReceiver for this queue
QueueReceiver queueReceiver = queueSession.createReceiver(queue);
//Start the connection such that messages get delivered
queueConnection.start();
//Receive the message
Message m = queueReceiver.receive(1);
TextMessage message = (TextMessage) m;
System.out.println("Receiving message: " +message.getText());
```

Spring ActiveMQ libraries

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-activemq</artifactId>  
</dependency>
```


Spring JMS sender

```
@SpringBootApplication
@PropertySource(value = "classpath:application.properties")
@EnableJms
public class SpringJmsSenderApplication {

    public static void main(String[] args) {
        AnnotationConfigApplicationContext context = new
            AnnotationConfigApplicationContext(SpringJmsSenderApplication.class);
        JmsTemplate jmsTemplate = context.getBean(JmsTemplate.class);

        System.out.println("Sending a JMS message.");
        jmsTemplate.convertAndSend("testQueue", "Hello world!");
    }
}
```

Name of the queue

application.properties

```
spring.activemq.broker-url=tcp://localhost:61616
spring.activemq.user=admin
spring.activemq.password=admin
```

Spring JMS receiver

```
@Component
public class MessageListener {

    @JmsListener(destination = "testQueue")
    public void receiveMessage(String msg) {
        System.out.println("Received :" + msg);
    }
}
```

Name of the queue

```
@SpringBootApplication
@PropertySource(value = "classpath:application.properties")
@EnableJms
public class SpringJmsReceiverApplication {

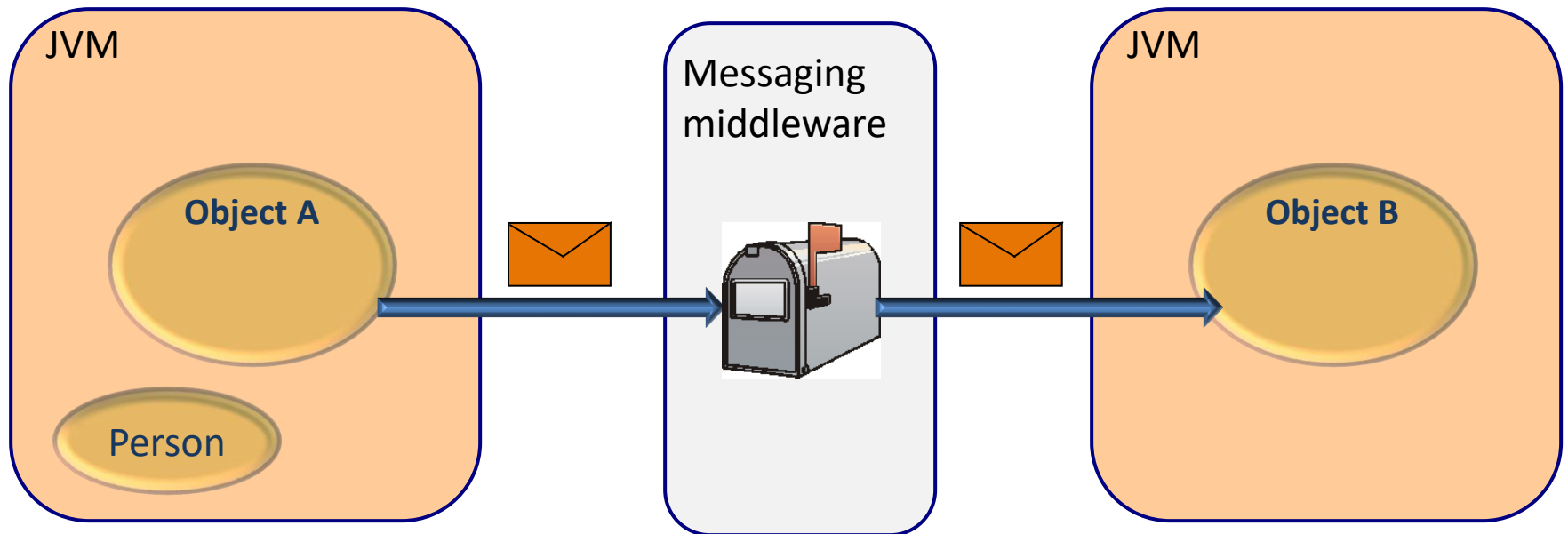
    public static void main(String[] args) {
        AnnotationConfigApplicationContext context = new
            AnnotationConfigApplicationContext(SpringJmsReceiverApplication.class);
    }
}
```

application.properties

```
spring.activemq.broker-url=tcp://localhost:61616
spring.activemq.user=admin
spring.activemq.password=admin
```

Sending an object

```
public class Person {  
    private String firstName;  
    private String lastName;  
    ...  
}
```



Sending an object

@SpringBootApplication

@EnableJms

public class SpringJmsPersonSenderApplication implements CommandLineRunner {

@Autowired

JmsTemplate jmsTemplate;

public static void main(String[] args) {

SpringApplication.run(SpringJmsPersonSenderApplication.class, args);

}

@Override

public void run(String... args) throws Exception {

Person person = new Person("Frank", "Brown");

//convert person to JSON string

ObjectMapper objectMapper = new ObjectMapper();

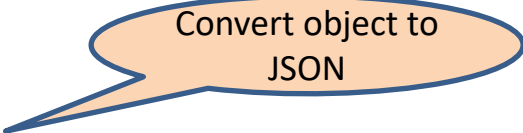
String personAsString = objectMapper.writeValueAsString(person);

System.out.println("Sending a JMS message:" + personAsString);

jmsTemplate.convertAndSend("testQueue", personAsString);

}

}



Convert object to
JSON

Receiving an object

@SpringBootApplication

@EnableJms

```
public class SpringJmsReceiverApplication {  
    public static void main(String[] args) {  
        SpringApplication.run(SpringJmsReceiverApplication.class, args);  
    }  
}
```

@Component

public class PersonMessageListener {

@JmsListener(destination = "testQueue")

public void receiveMessage(**final** String personAsString) {

ObjectMapper objectMapper = **new** ObjectMapper();

try {

Person person = objectMapper.readValue(personAsString, Person.class);

System.out.println("JMS receiver received message:" + person.getFirstName()+" "+person.getLastName());

} **catch** (IOException e) {

System.out.println("JMS receiver: Cannot convert : " + personAsString+" to a Person object");

}

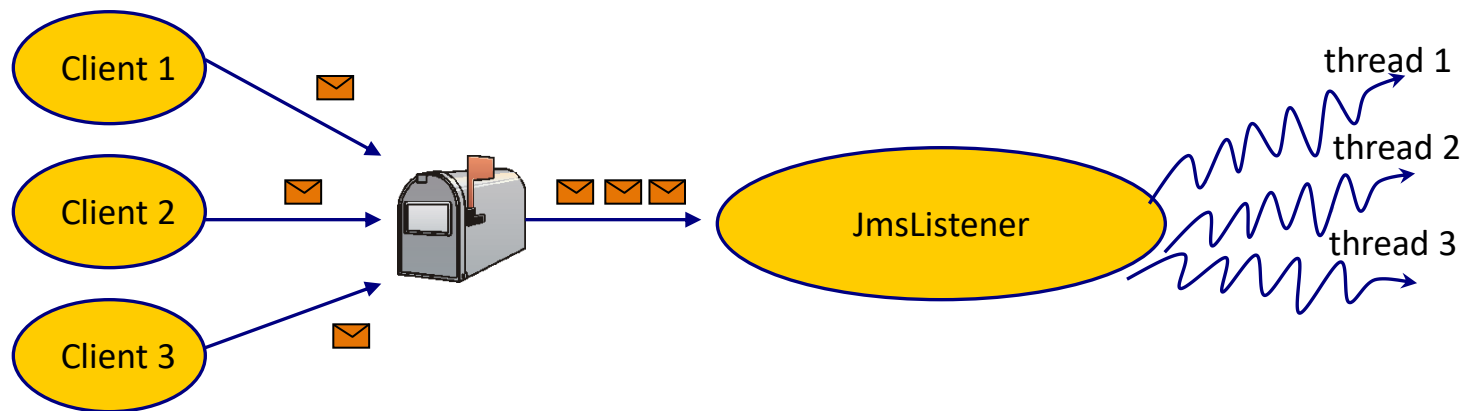
}

}

Convert JSON String to
object

JMS and concurrency

- Every JmsListener method executes in its own thread



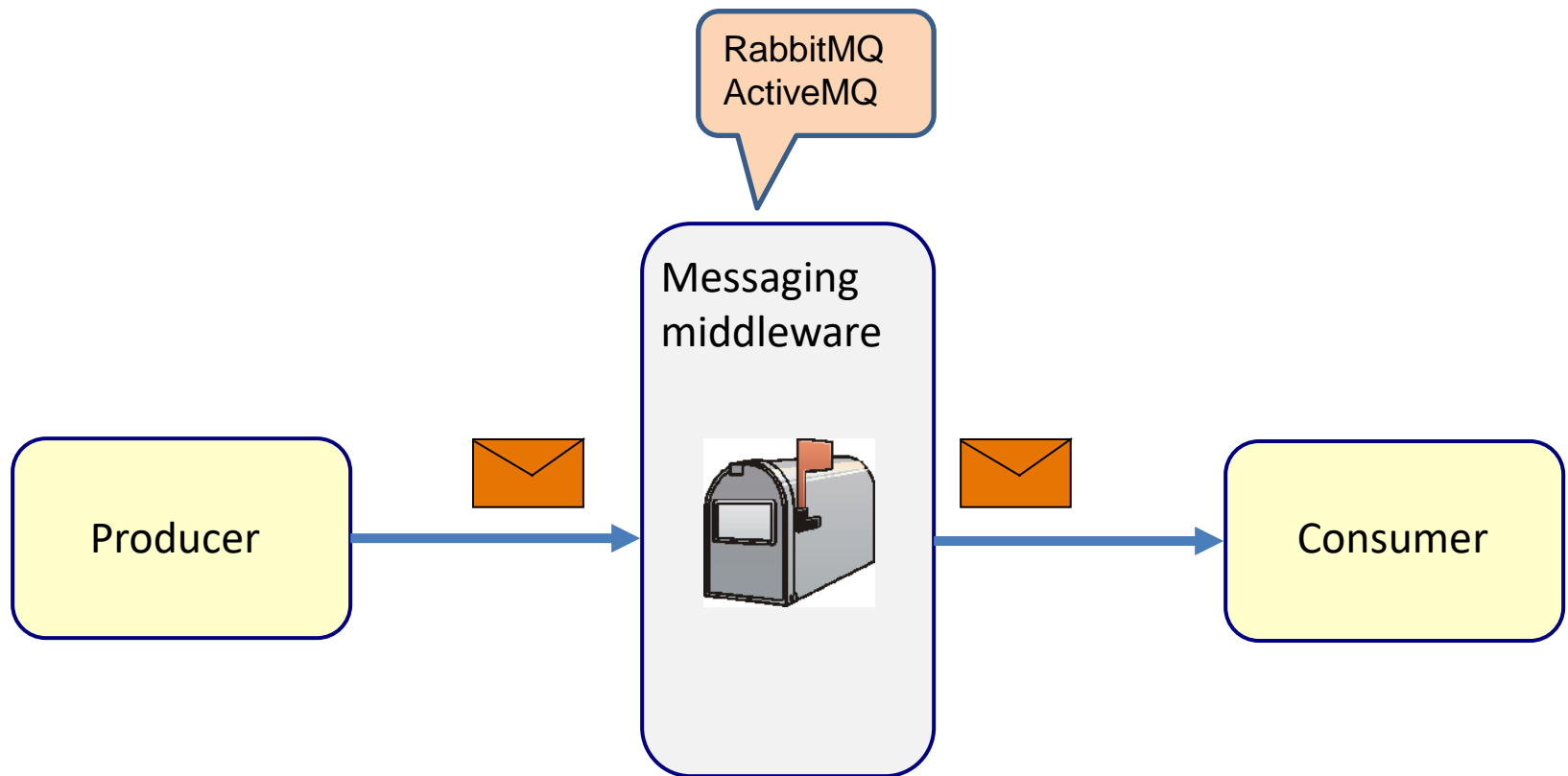
Main point

- Messaging gives loose coupling between the sender and the receiver.

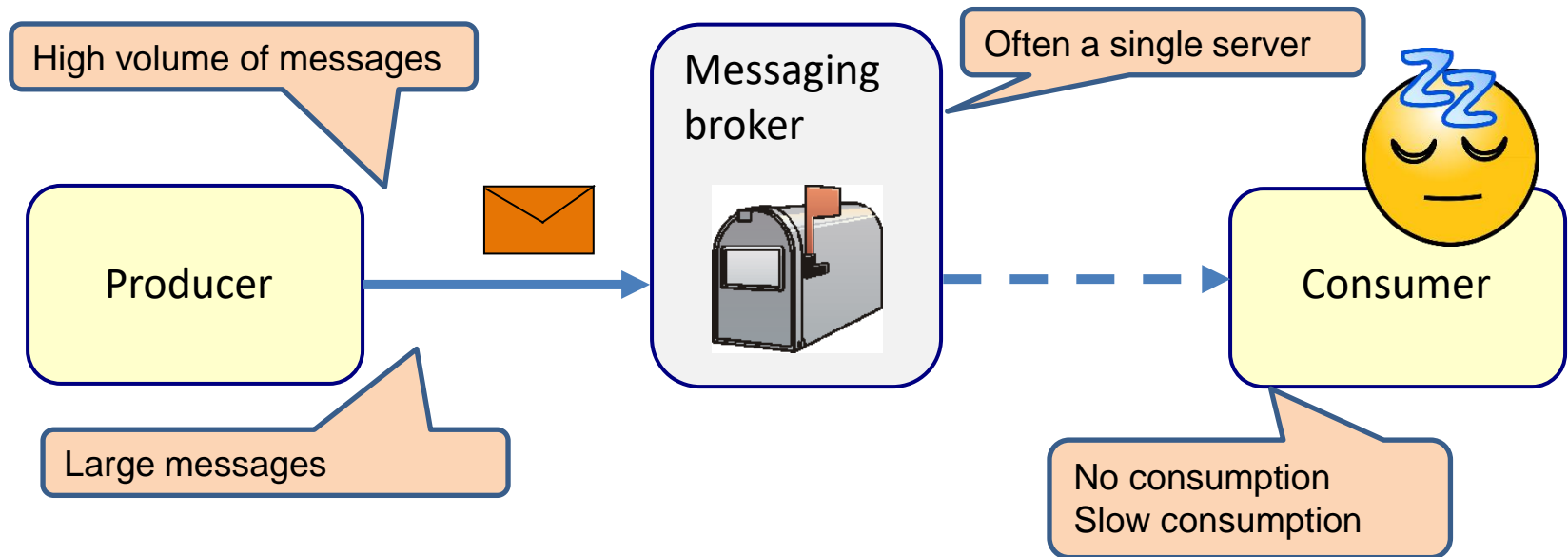
Science of Consciousness: The whole relative creation is an expression of the same one unified field.

KAFKA

Traditional Messaging Systems

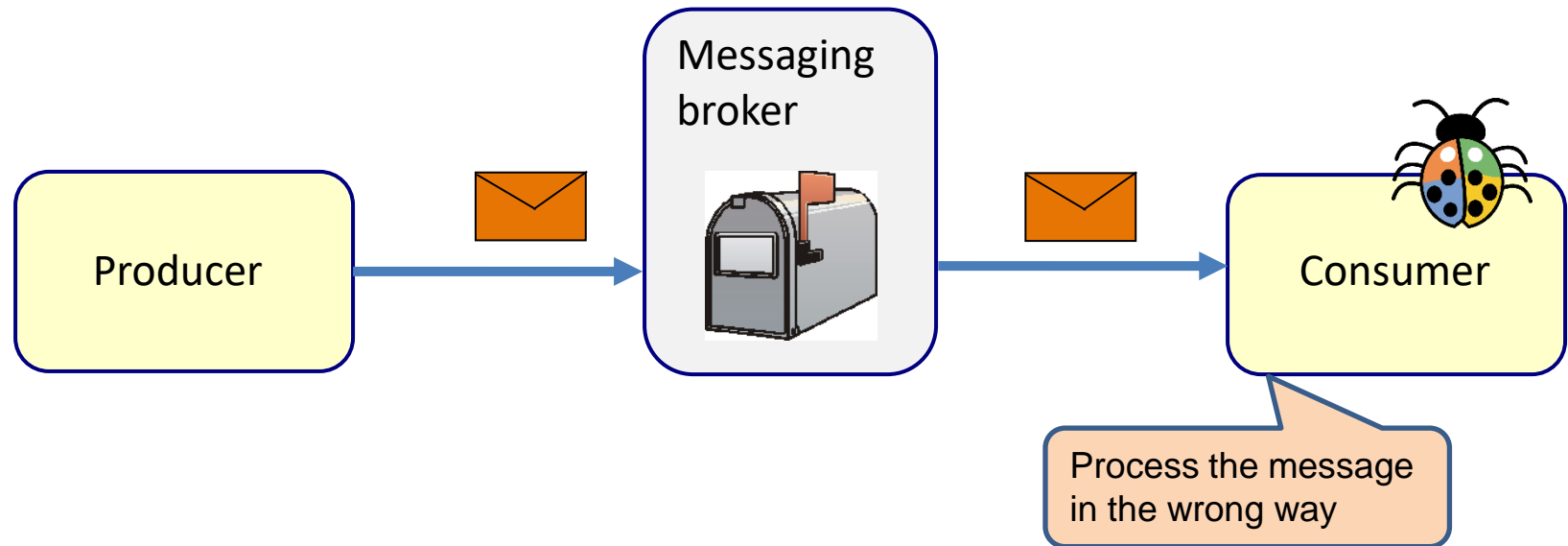


Problems with traditional messaging middleware



- If the consumer is temporally not available (or very slow) the message middleware has to store the messages
 - This restricts the volume of messages and the size of the messages
 - Eventually the message broker will fail

Problems with traditional messaging middleware



- If the consumer has a bug, and handles the messages incorrectly, then the messages are gone.
 - Not fault-tolerant

Apache Kafka



- Created by Linked In



- Characteristics

- High throughput

- Distributed

- Unlimited scalable

- Fault-tolerant

- Reliable and durable

- Loosely coupled Producers and Consumers

- Flexible publish-subscribe semantics

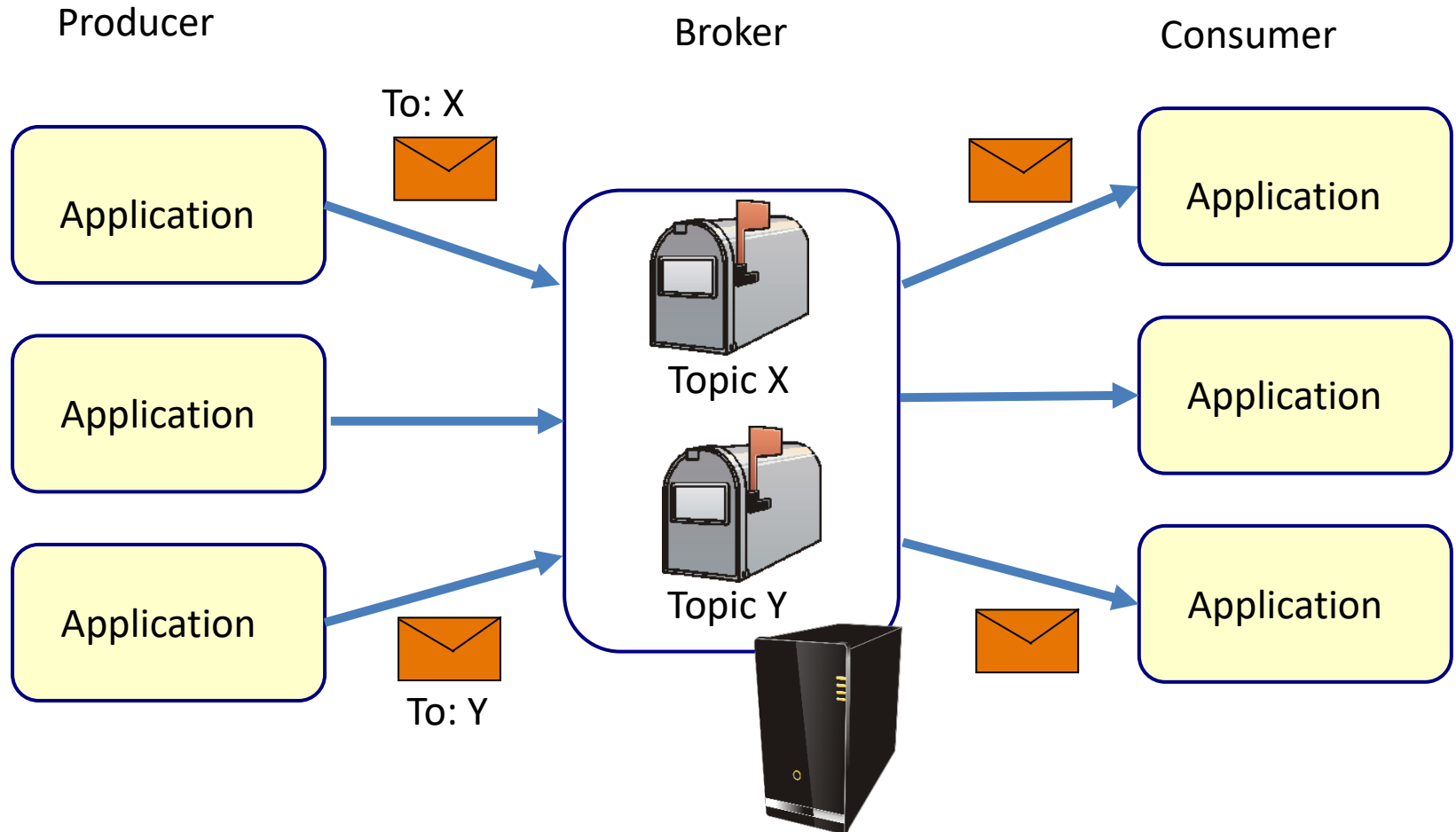
High Volume:

- Over 1.4 trillion messages per day
- 175 terabytes per day

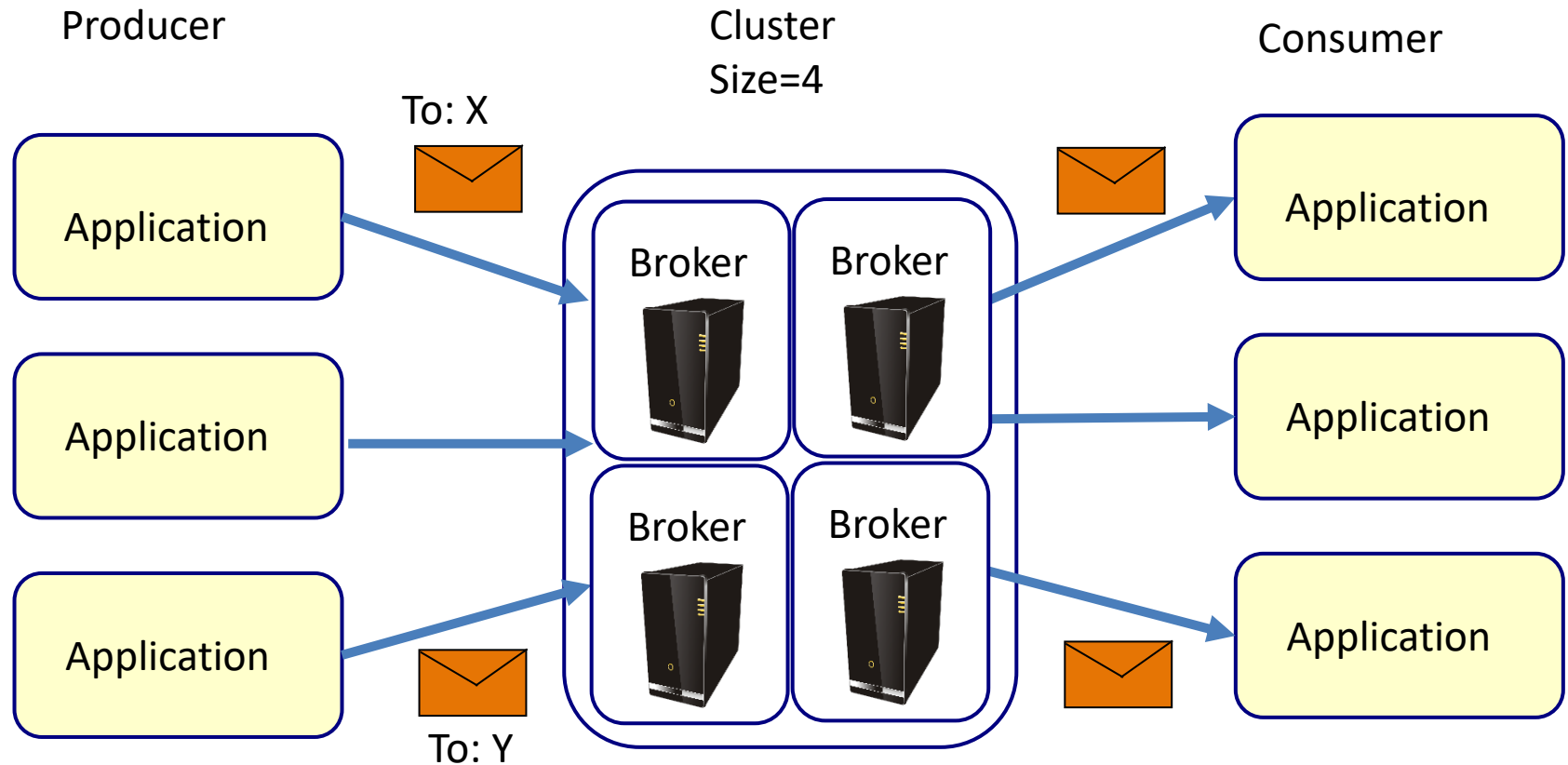
High Velocity:

- Peak 13 million messages per second
- 2.75 gigabytes per second

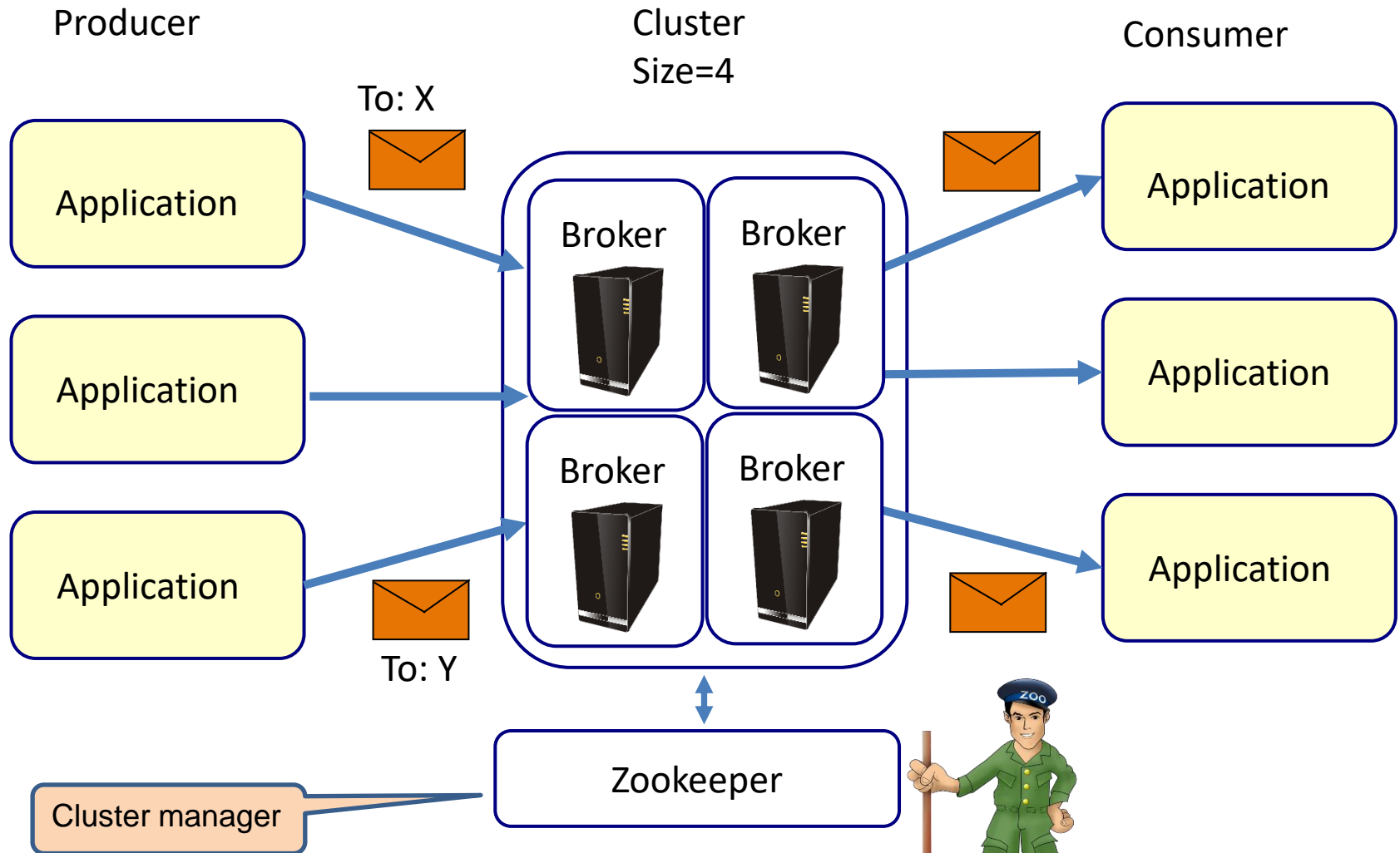
Kafka



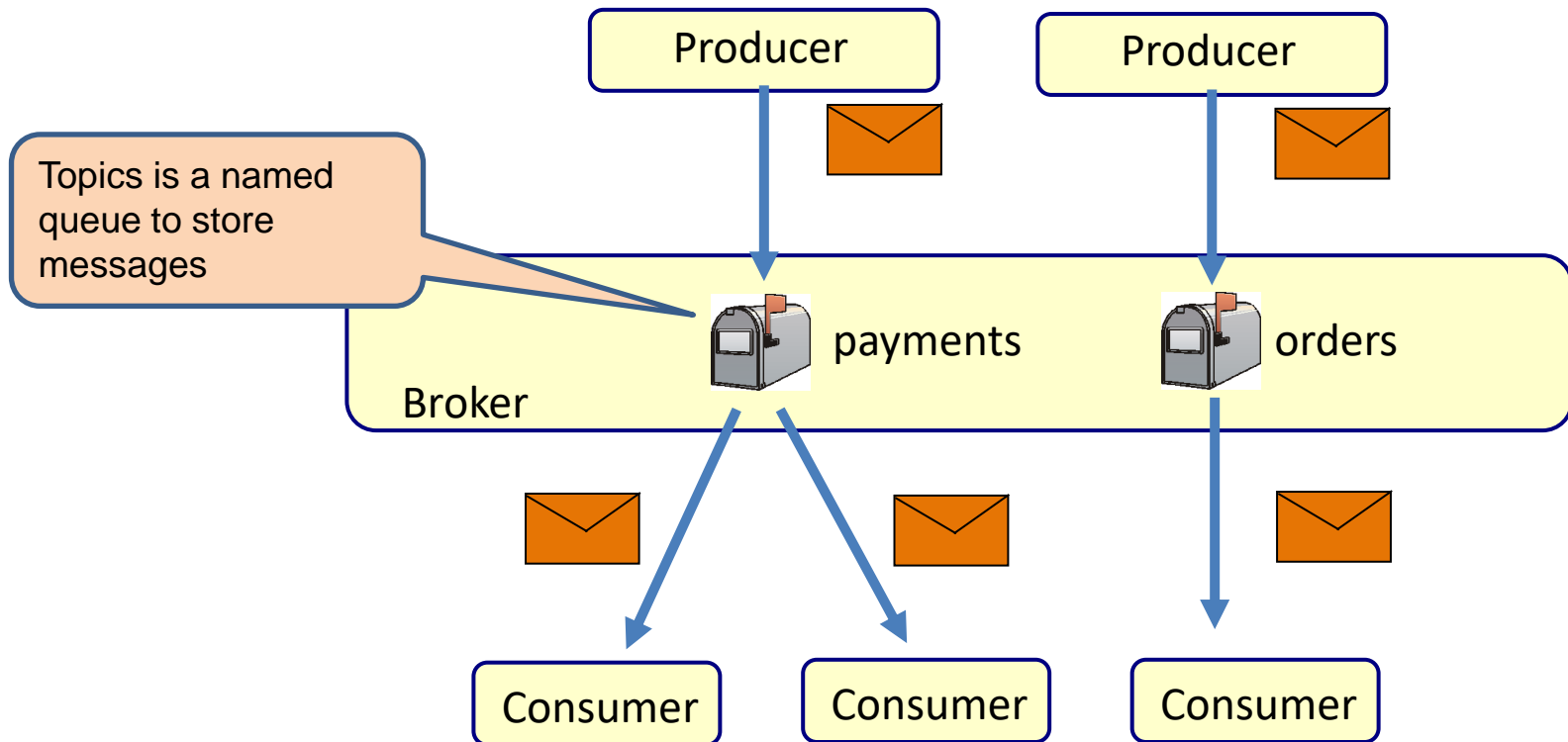
Cluster of Brokers



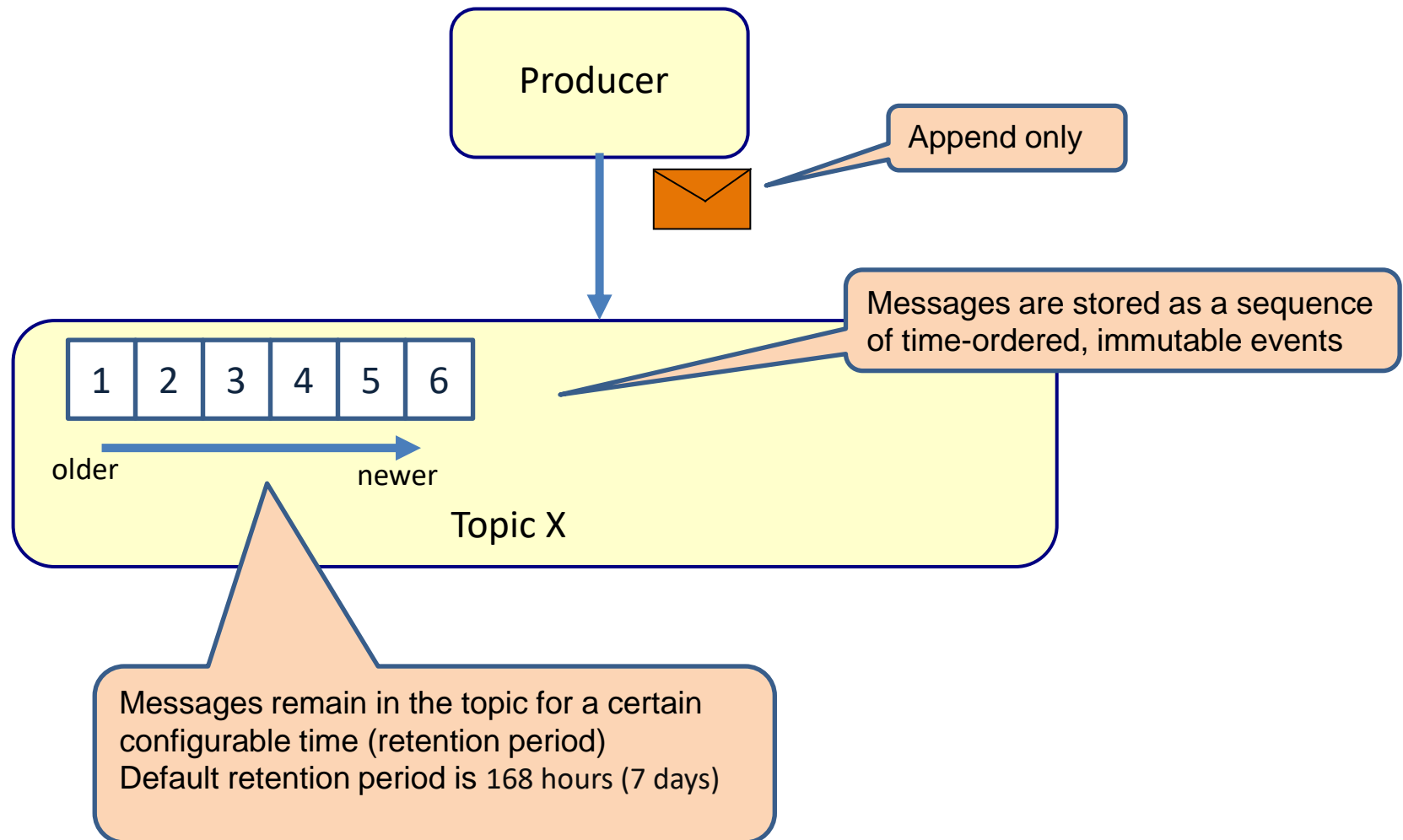
Apache Zookeeper



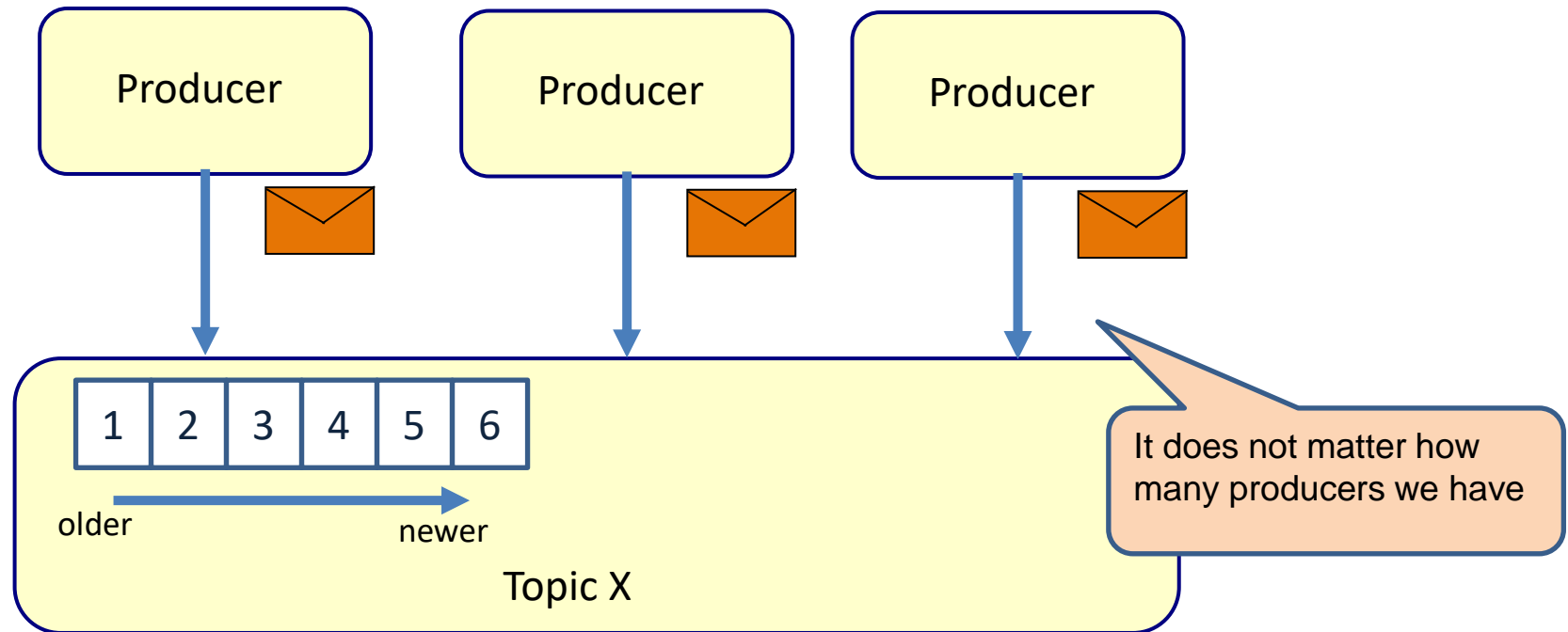
Topics



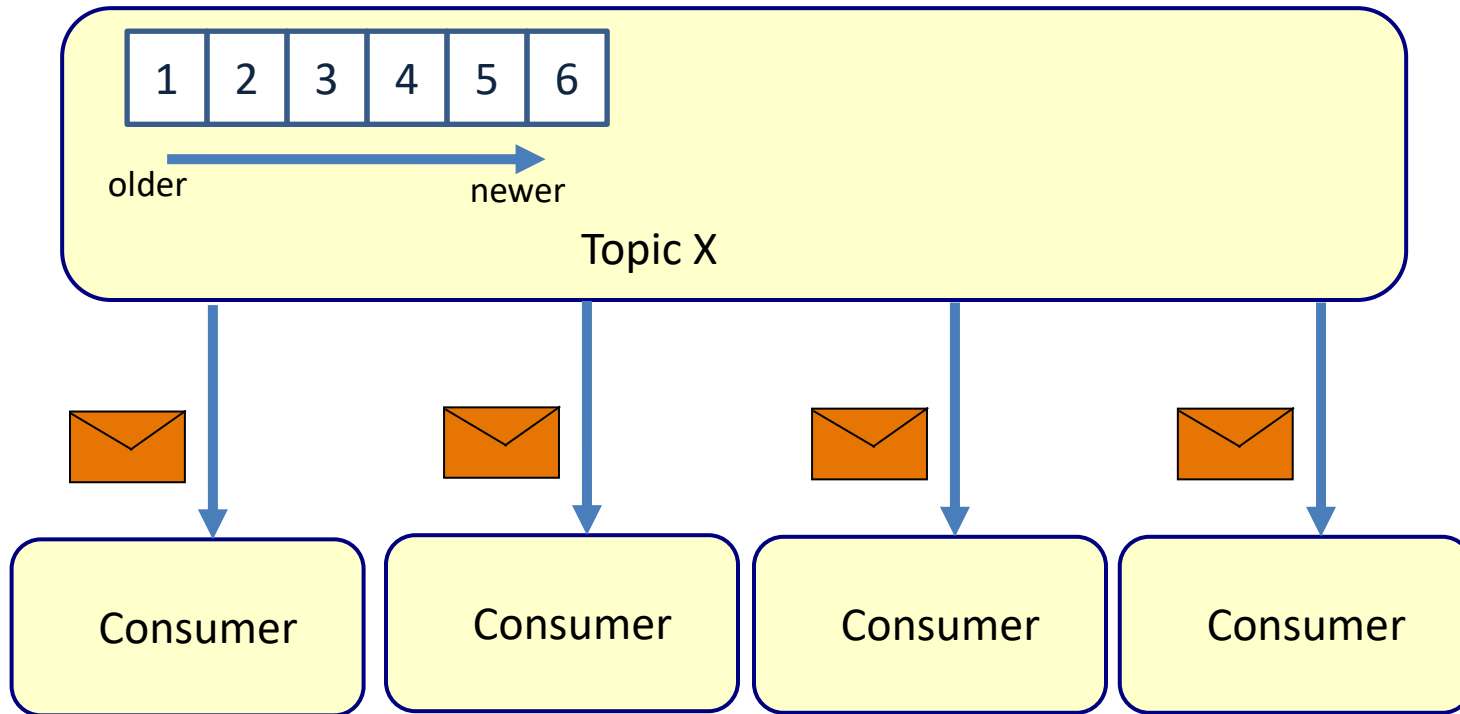
Event sourcing



Why event sourcing?

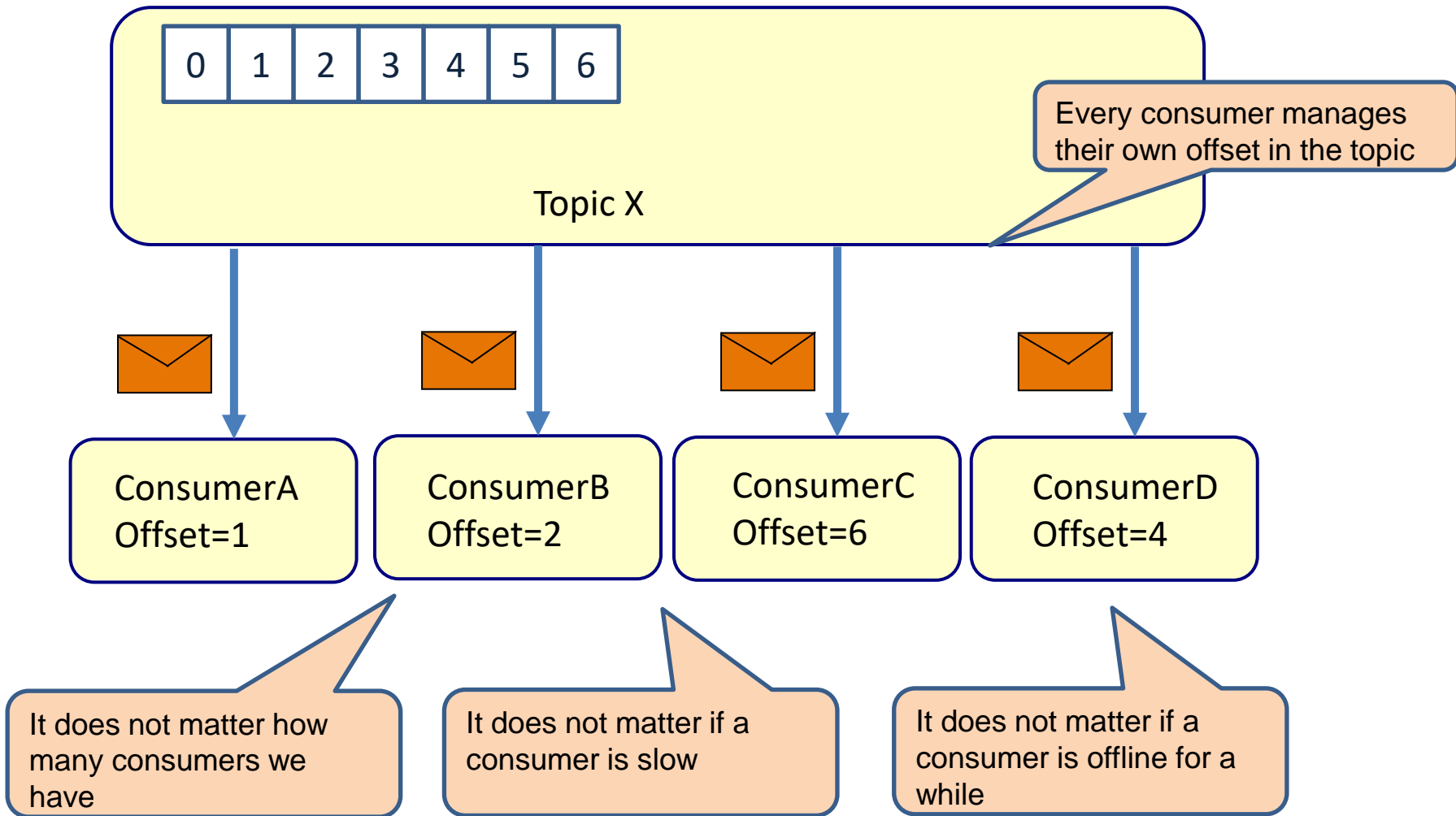


Why event sourcing?



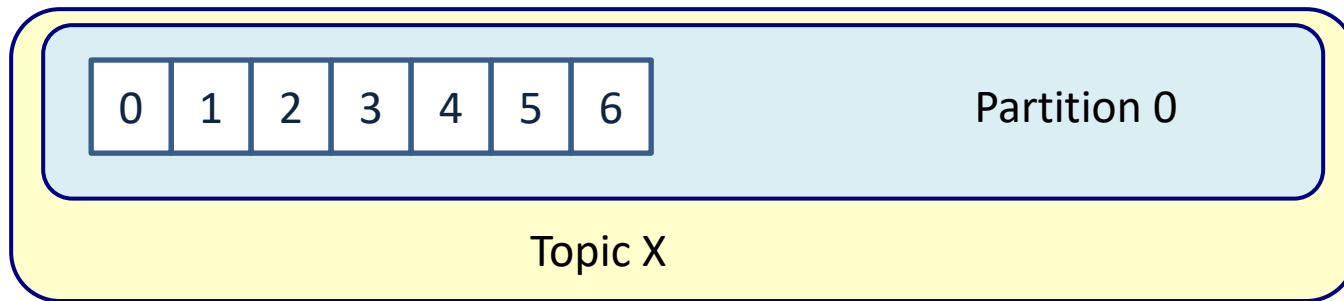
It does not matter how many consumers we have

Offset

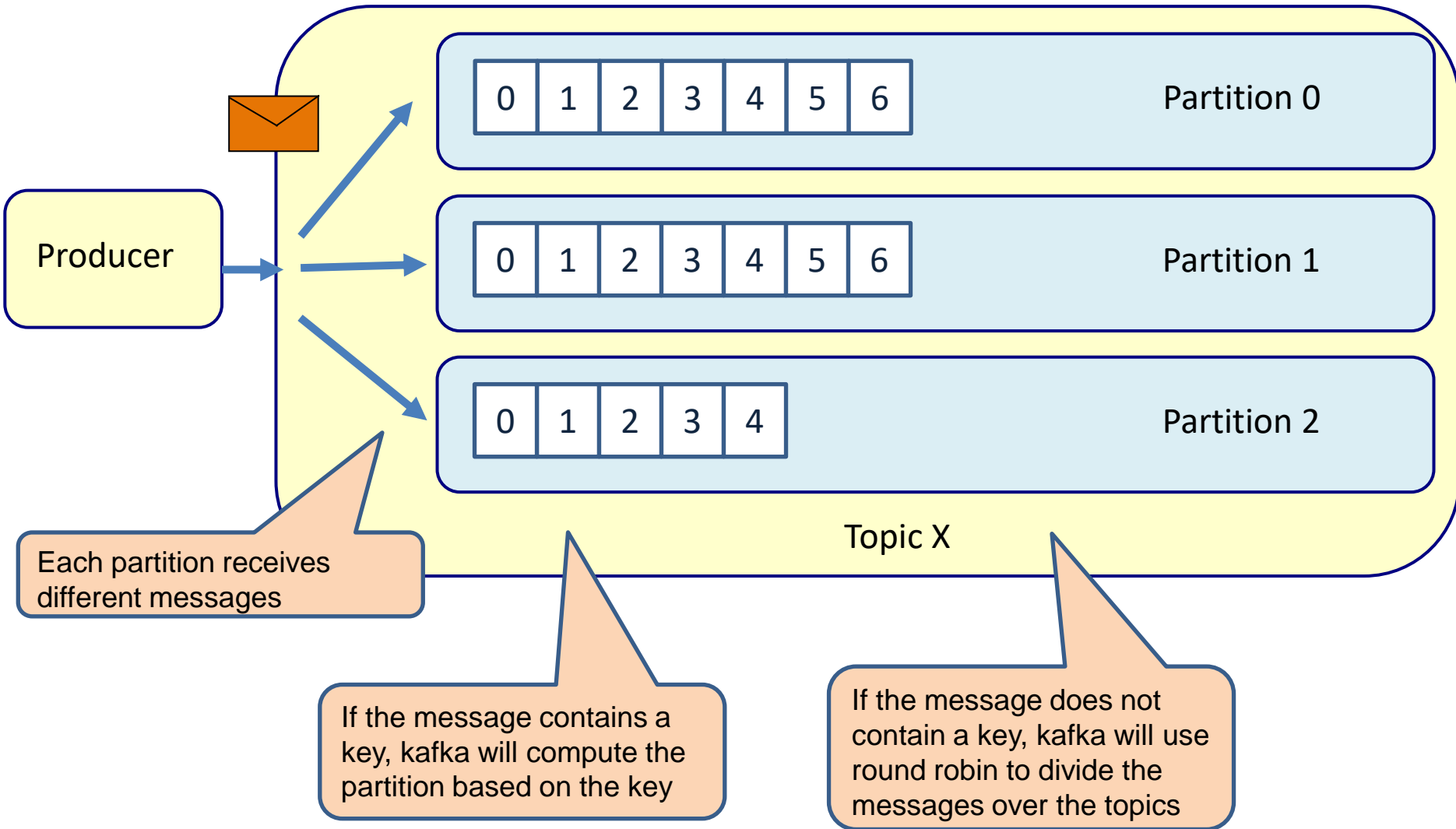


Partition

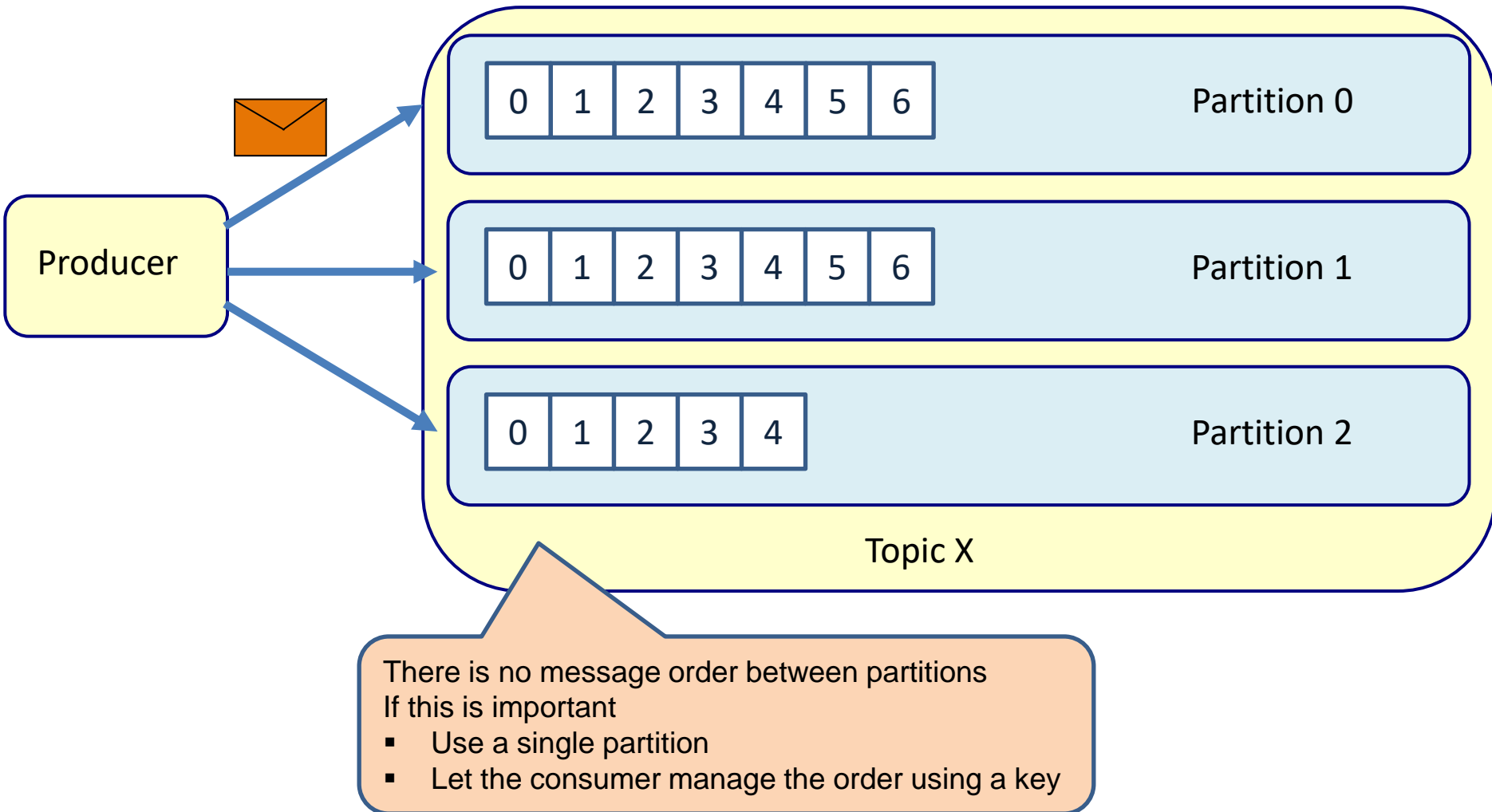
- Each topic has one or more partitions
 - This is configurable
- Each partition must fit on 1 broker



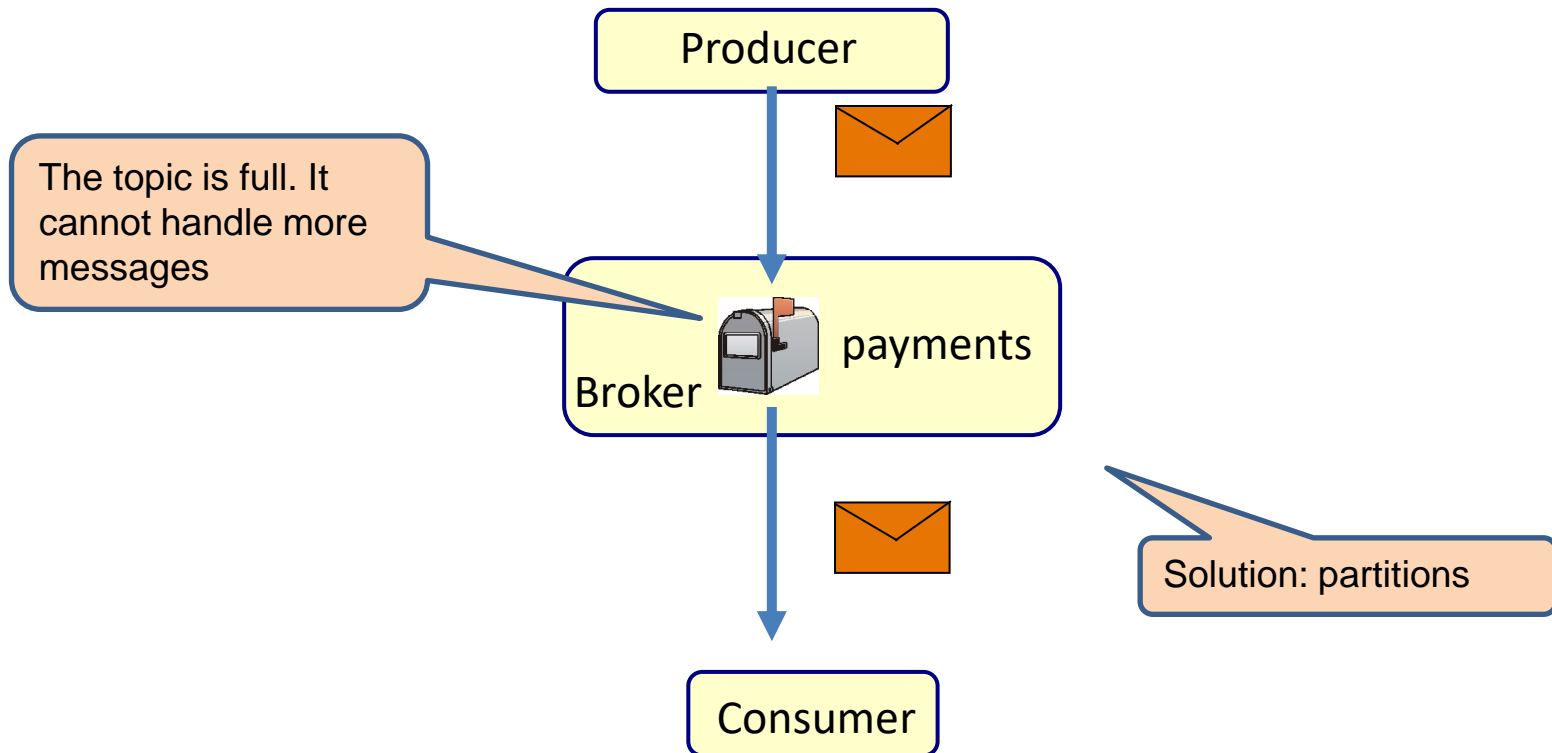
3 partitions



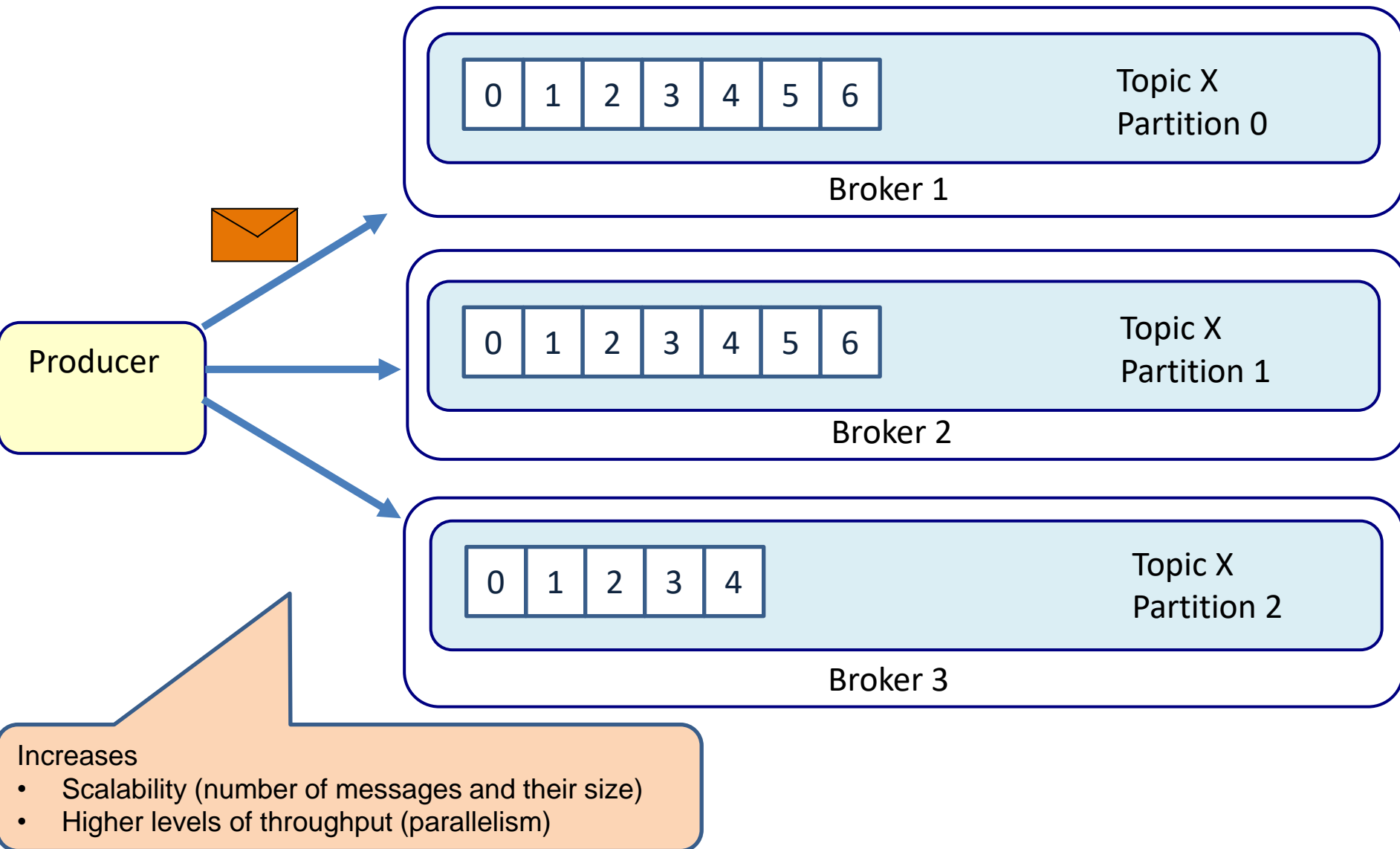
3 partitions



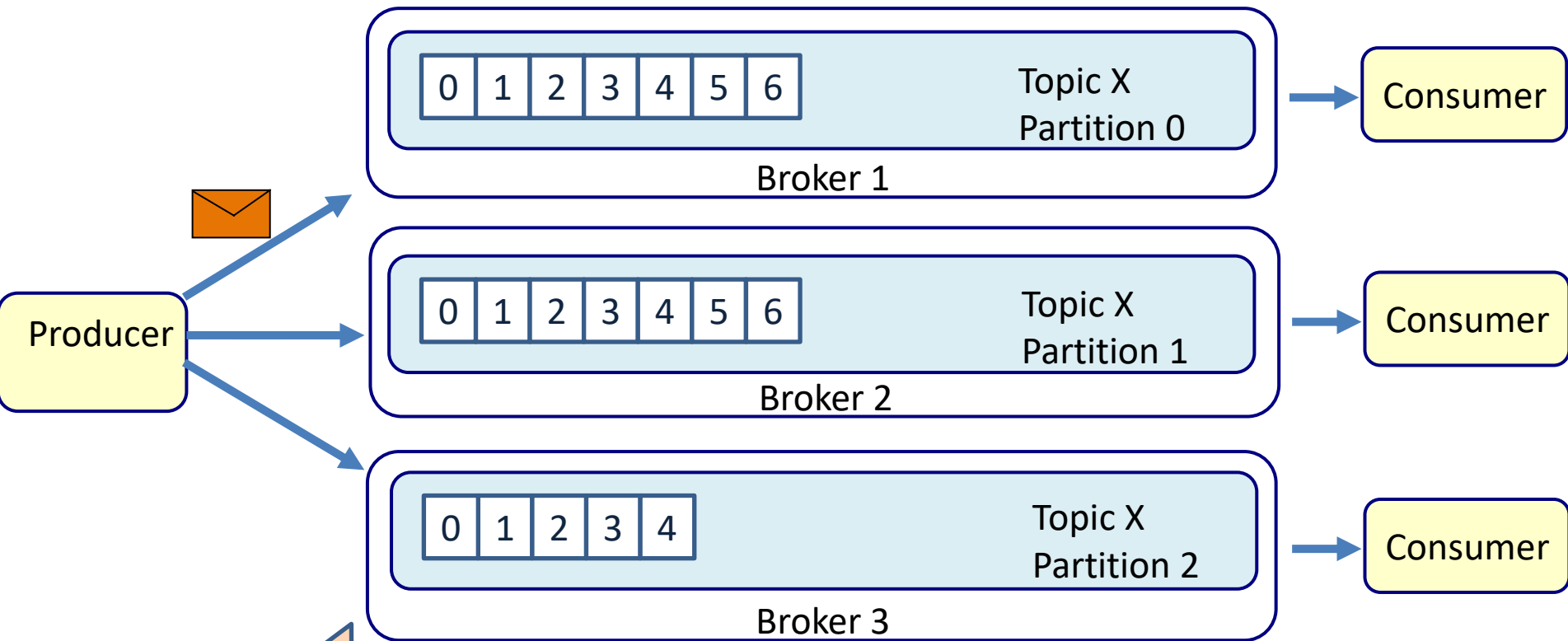
What if the topic gets too full?



Scale out partitions



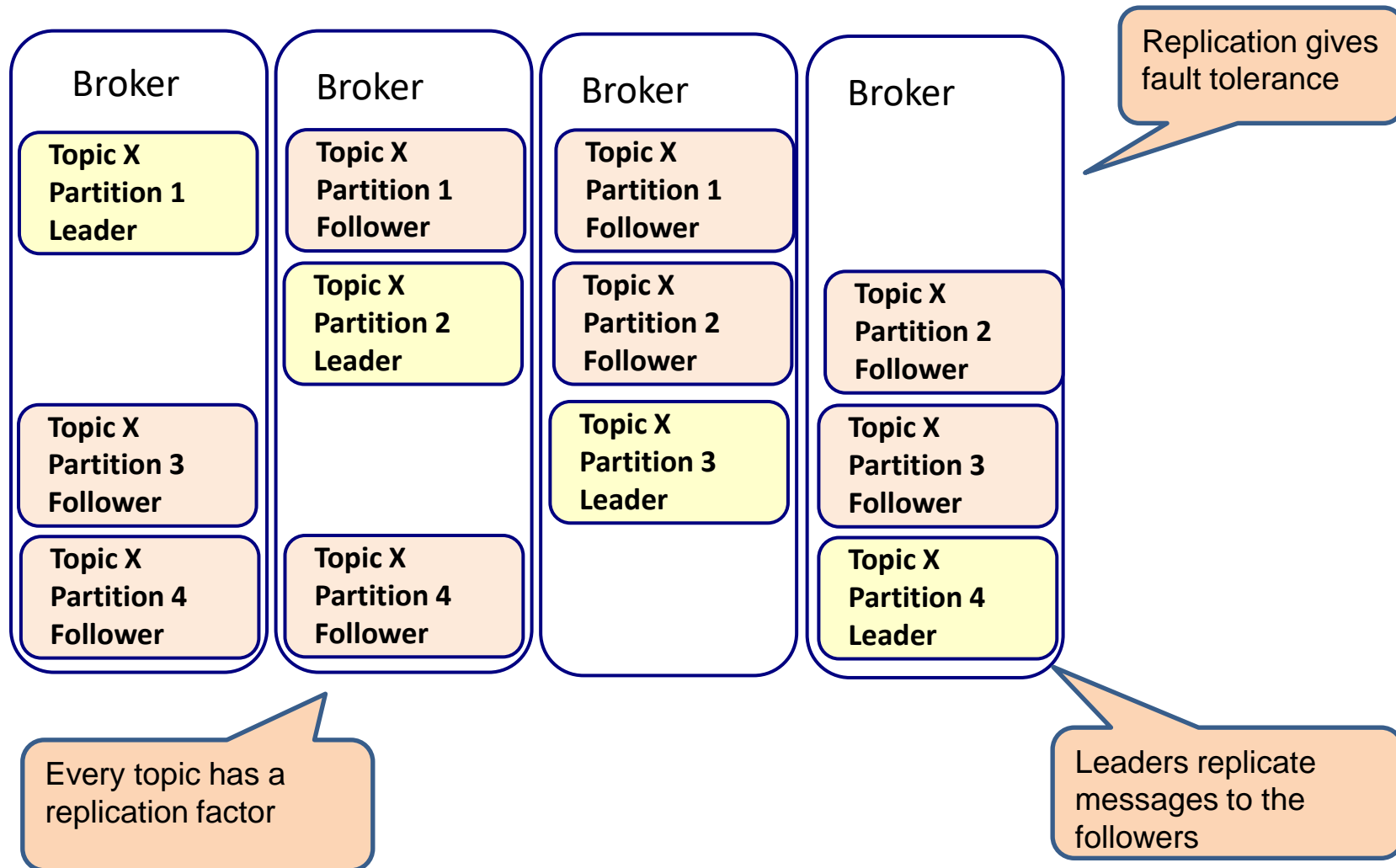
Scale out partitions



Increases

- Scalability (number of messages and their size)
- Higher levels of throughput (parallelism)

Replication



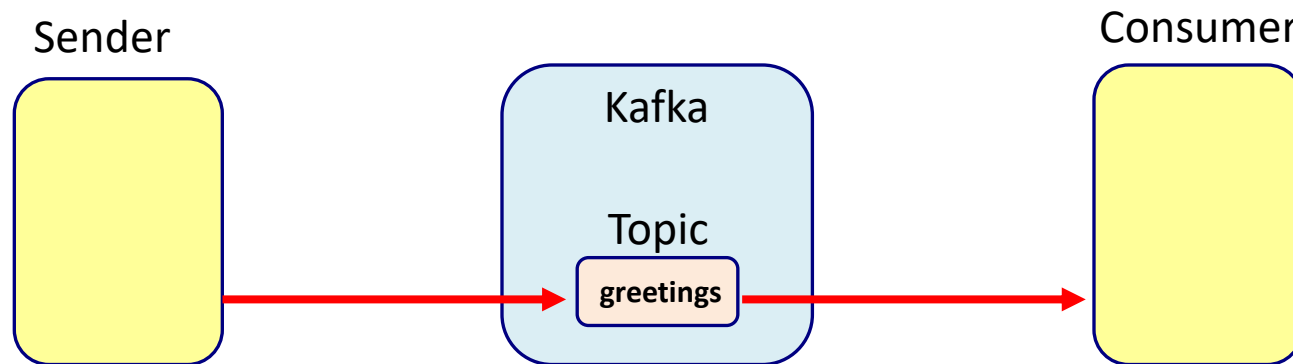
Creating a topic



```
~$ bin/kafka-topics.sh --create --topic my_topic \  
> --zookeeper localhost:2181 \  
> --partitions 3 \  
> --replication-factor 3
```

SPRING BOOT AND KAFKA

Example



SenderApplication

@SpringBootApplication

@EnableKafka

public class SenderApplication implements CommandLineRunner {

@Autowired

Sender sender;

public static void main(String[] args) {

SpringApplication.run(SenderApplication.class, args);

}

@Override

public void run(String... args) throws Exception {

sender.send("topicA", "Hello World");

System.out.println("Message has been sent");

}

}

Sender

```
@Service
public class Sender {
    @Autowired
    private KafkaTemplate<String, String> kafkaTemplate;

    public void send(String topic, String message){
        kafkaTemplate.send(topic, message);
    }
}
```

application.properties

```
spring.kafka.bootstrap-servers=localhost:9092
spring.kafka.consumer.group-id= gid
spring.kafka.consumer.auto-offset-reset= earliest
spring.kafka.consumer.key-deserializer= org.apache.kafka.common.serialization.StringDeserializer
spring.kafka.consumer.value-deserializer= org.springframework.kafka.support.serializer.JsonDeserializer
spring.kafka.producer.key-serializer= org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.value-serializer= org.springframework.kafka.support.serializer.JsonSerializer
spring.kafka.consumer.properties.spring.json.trusted.packages=kafka
```

```
logging.level.root= ERROR
org.springframework= ERROR
```


ReceiverApplication



@SpringBootApplication

@EnableKafka

public class ReceiverApplication **implements** CommandLineRunner {

public static void **main**(String[] args) {

SpringApplication.run(ReceiverApplication.class, args);

}

@Override

public void run(String... args) **throws** Exception {

System.out.println("Receiver is running and waiting for messages");

}

}

Receiver

```
@Service
public class Receiver {

    @KafkaListener(topics = {"topicA"})
    public void receive(@Payload String message) {
        System.out.println("Receiver received message= " + message);
    }
}
```

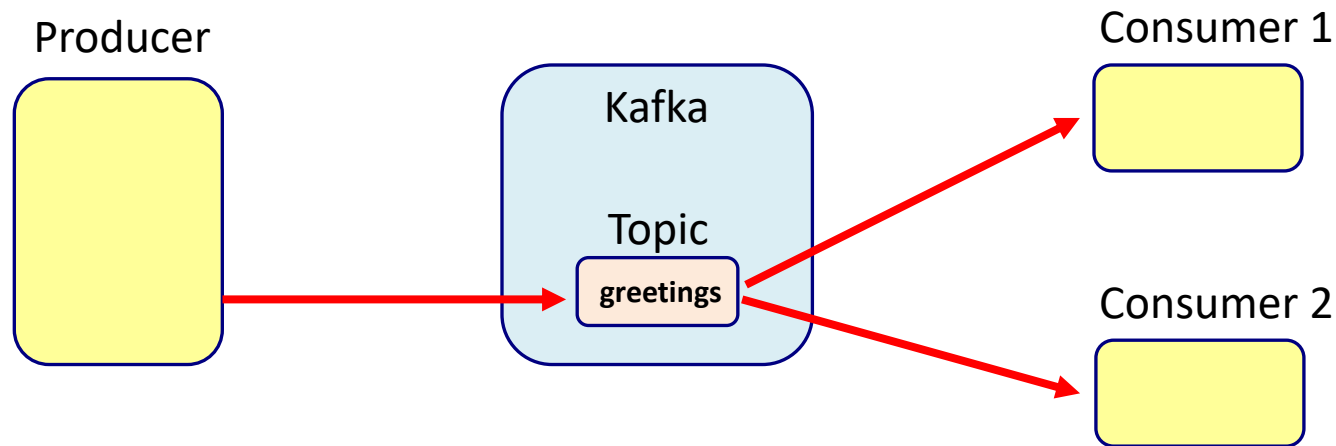
application.properties

```
spring.kafka.bootstrap-servers=localhost:9092
spring.kafka.consumer.group-id= gid
spring.kafka.consumer.auto-offset-reset= earliest
spring.kafka.consumer.key-deserializer= org.apache.kafka.common.serialization.StringDeserializer
spring.kafka.consumer.value-deserializer= org.springframework.kafka.support.serializer.JsonDeserializer
spring.kafka.producer.key-serializer= org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.value-serializer= org.springframework.kafka.support.serializer.JsonSerializer
spring.kafka.consumer.properties.spring.json.trusted.packages=kafka
```

```
logging.level.root= ERROR
org.springframework= ERROR
```

What if we have 2 consumers

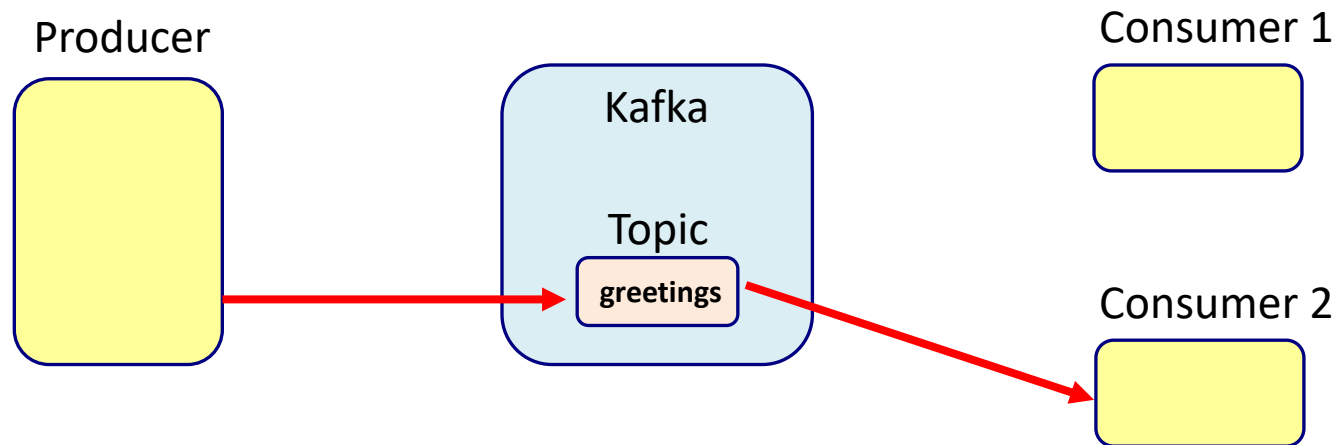
- The default behavior is pub/sub
 - Instead of point to point



- Both consumers receive the message

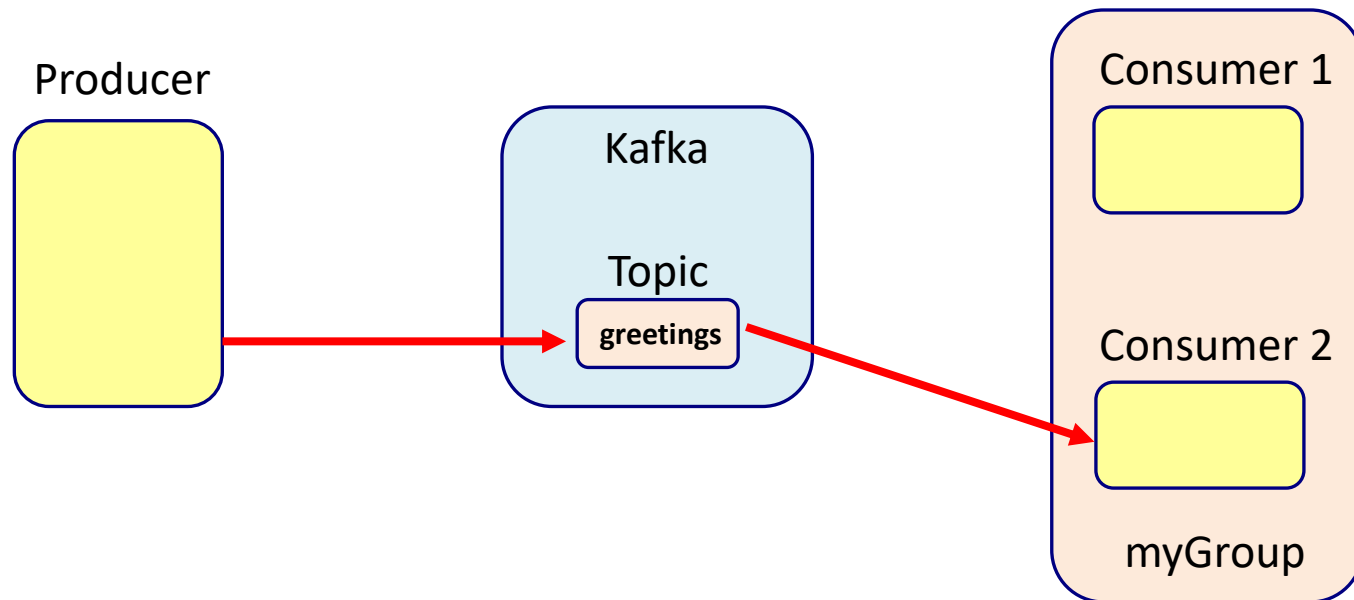
What if we want point to point

- Competing consumers



- Only one consumers receives the message

Consumer groups



application.properties

```
spring.kafka.bootstrap-servers=localhost:9092  
spring.kafka.consumer.group-id= gid  
...
```

Give both consumers
the same group-id

Send an object: Sender

@SpringBootApplication

@EnableKafka

public class OrderApplication implements CommandLineRunner {

@Autowired

Sender sender;

public static void main(String[] args) {

SpringApplication.run(OrderApplication.class, args);

}

@Override

public void run(String... args) throws Exception {

sender.send("ordertopic", new Order("A1276", LocalDate.now()+"" , 1200.0));

System.out.println("Order has been sent");

}

}

public class Order {

private String orderNumber;

private String date;

private double amount;

Sender

```
@Service
public class Sender {
    @Autowired
    private KafkaTemplate<String, Order> kafkaTemplate;

    public void send(String topic, Order order){
        kafkaTemplate.send(topic, order);
    }
}
```

application.properties

```
spring.kafka.bootstrap-servers=localhost:9092
spring.kafka.consumer.group-id= gid
spring.kafka.consumer.auto-offset-reset= earliest
spring.kafka.consumer.key-deserializer= org.apache.kafka.common.serialization.StringDeserializer
spring.kafka.consumer.value-deserializer= org.springframework.kafka.support.serializer.JsonDeserializer
spring.kafka.producer.key-serializer= org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.value-serializer= org.springframework.kafka.support.serializer.JsonSerializer
spring.kafka.consumer.properties.spring.json.trusted.packages=kafka
```

```
logging.level.root= ERROR
org.springframework= ERROR
```

Receiver Application

```
@SpringBootApplication
```

```
@EnableKafka
```

```
public class OrderApplication {
```

```
    public static void main(String[] args) {  
        SpringApplication.run(OrderApplication.class, args);  
    }
```

```
}
```

```
public class Order {  
    private String orderNumber;  
    private String date;  
    private double amount;
```


Receiver

```
@Service
public class Receiver {

    @KafkaListener(topics = {"ordertopic"})
    public void receive(@Payload Order order) {
        System.out.println("OrderReceiver 1 received order="+ order);
    }
}
```

```
spring.kafka.bootstrap-servers=localhost:9092
spring.kafka.consumer.group-id= gid
spring.kafka.consumer.auto-offset-reset= earliest
spring.kafka.consumer.key-deserializer= org.apache.kafka.common.serialization.StringDeserializer
spring.kafka.consumer.value-deserializer= org.springframework.kafka.support.serializer.JsonDeserializer
spring.kafka.producer.key-serializer= org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.value-serializer= org.springframework.kafka.support.serializer.JsonSerializer
spring.kafka.consumer.properties.spring.json.trusted.packages=kafka
```

application.properties

```
logging.level.root= ERROR
org.springframework= ERROR
```

Main point

- Kafka is a distributed message broker that is fast, reliable and can handle large amounts of messages.

Science of Consciousness: Pure consciousness is the field of all possibilities. At this level there are no limitations.