Spring Boot Messaging RabbitMQ and Kafka

Rajeev Gupta

rgupta.mtech@gmail.com

https://www.linkedin.com/in/rajeevguptajavatrainer

rgupta.mtech@gmail.com

Messaging with RabitMQ

rgupta.mtech@gmail.com

Agenda

- What is Messaging Queue, How it works and its uses
- What is Rabbit MQ
- Different types of Exchanges in Rabbit, MQ
- What is Messaging Queue, How it works and its uses
- Different types of Exchanges in Rabbit MQ .
- Direct Exchanges
- Fanout Exchanges
- Topic Exchanges
- Header Exchanges
- Retry and Error Handling Example

What is RabbitMQ ?

What is RabbitMQ ?

- **RabbitMQ is a message broker that originally implements the Advance Message Queuing Protocol (AMQP)**

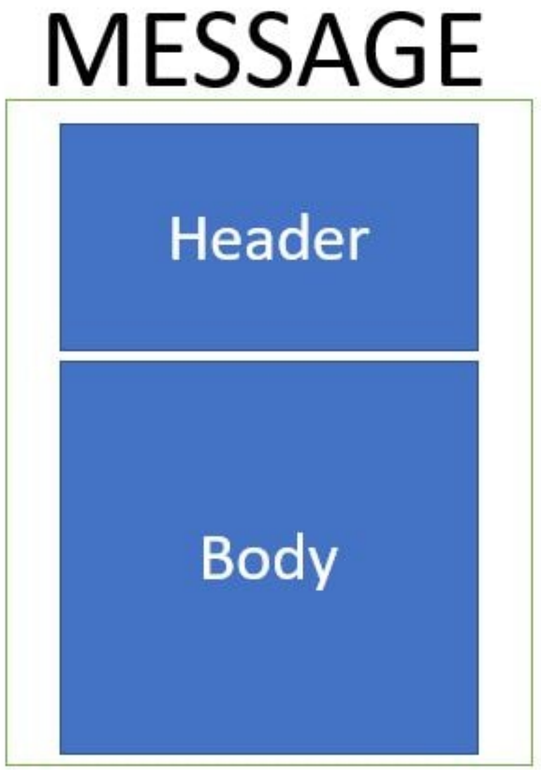
AMQP

- **AMQP standardizes messaging using Producers, Broker and Consumers.**
- **AMQP standards was designed with the following main characteristics Security, Reliability, Interoperability**
- **Reliability confirms the message was successfully delivered to the message broker and confirms that the message was successfully processed by the consumer**

What is Messaging?

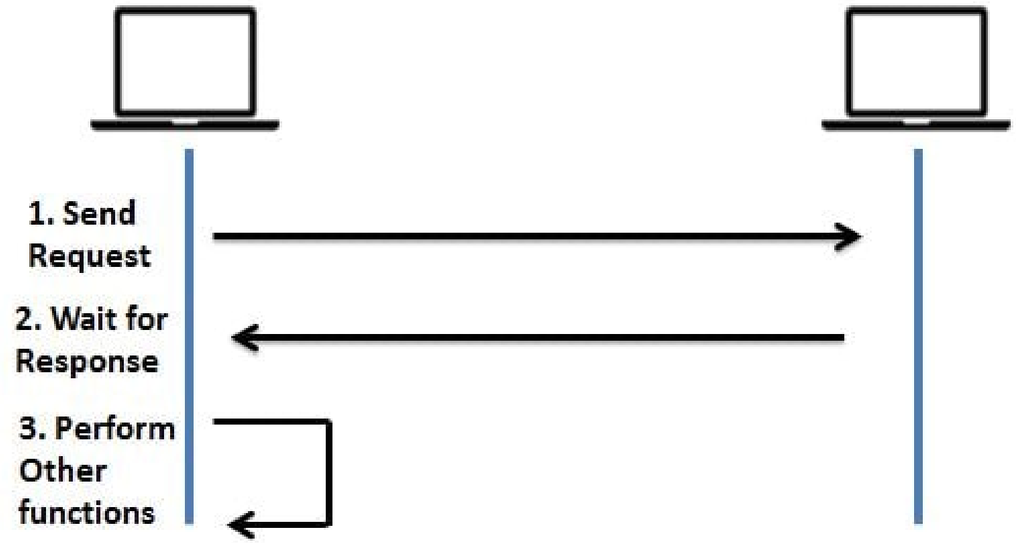
Messaging is a communication mechanism used for system interactions. In software development messaging enables distributed communication that is loosely coupled.

A messaging client can send messages to, and receive messages from, any other client. The structure of message can be defined as follows



Synchronous Messaging

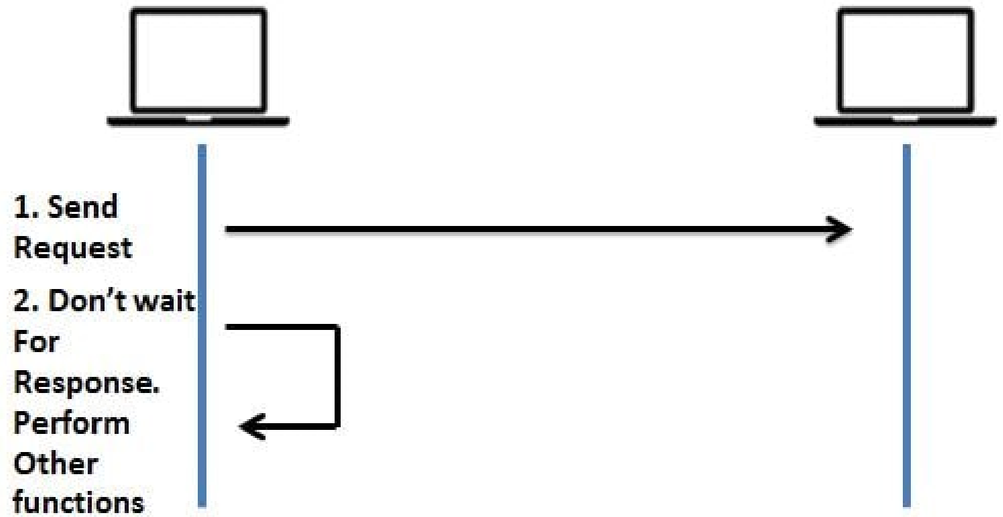
- **Synchronous Messaging is implemented when such that the messaging client sends a message and expects the response immediately. So the sender client waits for the response before he can execute the next task. So until and unless the message is recieved the sender is blocked.**



Synchronous Communication

Asynchronous Messaging

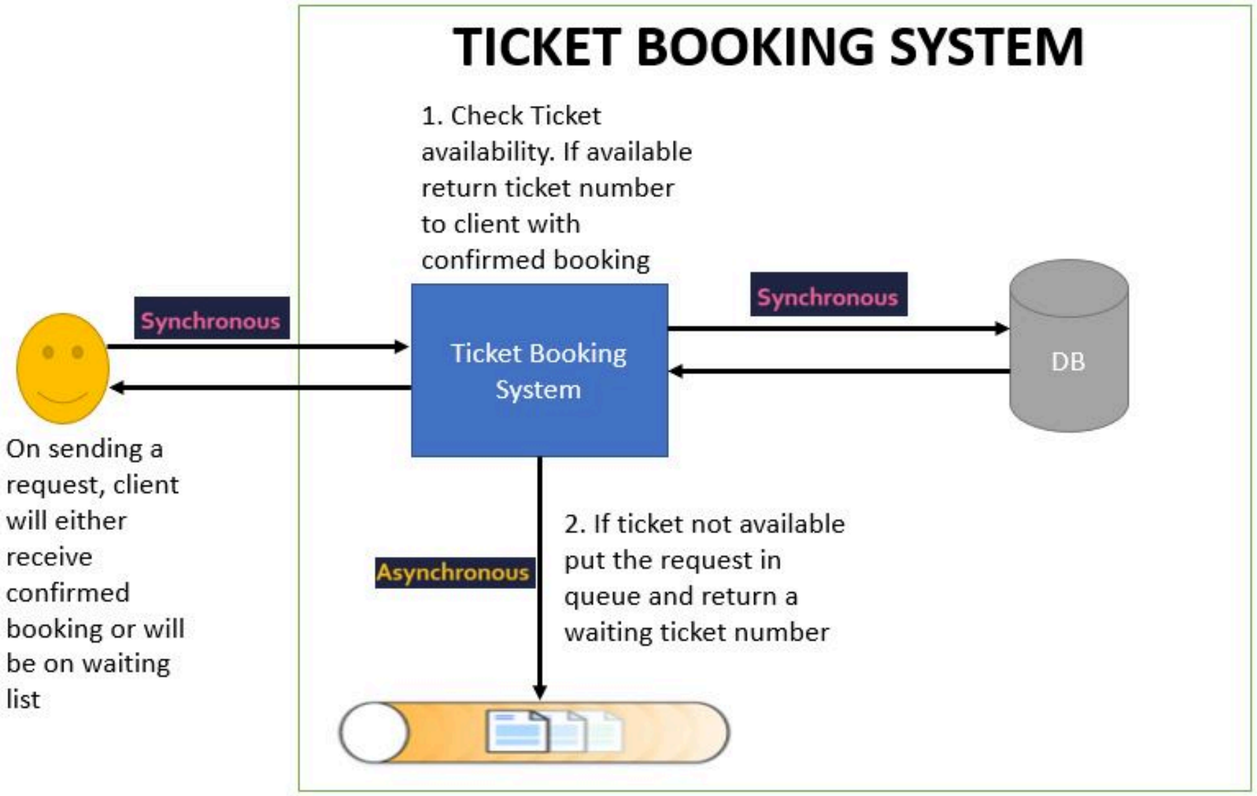
Asynchronous Messaging is implemented such that the messaging client sends a message and does not expect the response immediately. So the sender client does not for the response before he can execute the next task. So the sender is not blocked



Asynchronous Communication

Real time systems

- **Real time systems usually have a combination of syschronous and ascynchronous communication**

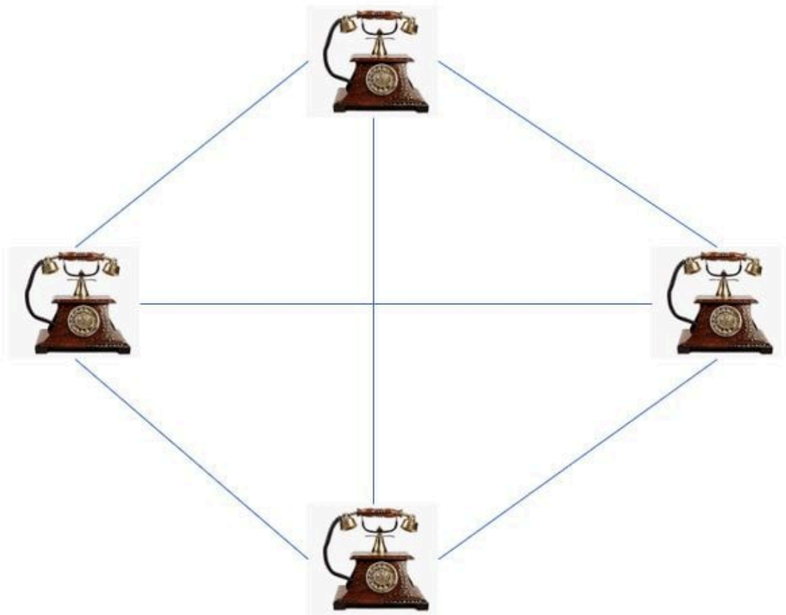


Busy Coder Academy

Message Broker

Message Broker - are responsible for establishing connections with various client systems.

- **Let us consider role of Message Broker in a telecom system. Suppose initially there is no message broker. Then each telephone connection will have a direct line with all other telephone connections.**



Rabbit mq installation ubuntu

```
echo "deb http://www.rabbitmq.com/debian/ testing main" | sudo tee -a
/etc/apt/sources.list
echo "deb http://packages.erlang-solutions.com/ubuntu wheezy contrib" | sudo tee -a
/etc/apt/sources.list
wget http://packages.erlang-solutions.com/ubuntu/erlang_solutions.asc
sudo apt-key add erlang_solutions.asc
sudo apt-get update
sudo apt-get -y install erlang erlang-nox
sudo apt-get -y --force-yes install rabbitmq-server
```

Rabbit mq installation ubuntu

 # Enable the web interface
sudo rabbitmq-plugins enable rabbitmq_management
sudo service rabbitmq-server restart
ss -tunelp | grep 1567

sudo service rabbitmq-server start
sudo service rabbitmq-server stop
http://localhost:15672

https://stackoverflow.com/questions/8808909/simple-way-to-install-rabbitmq-in-ubuntu

Busy Coder Academy

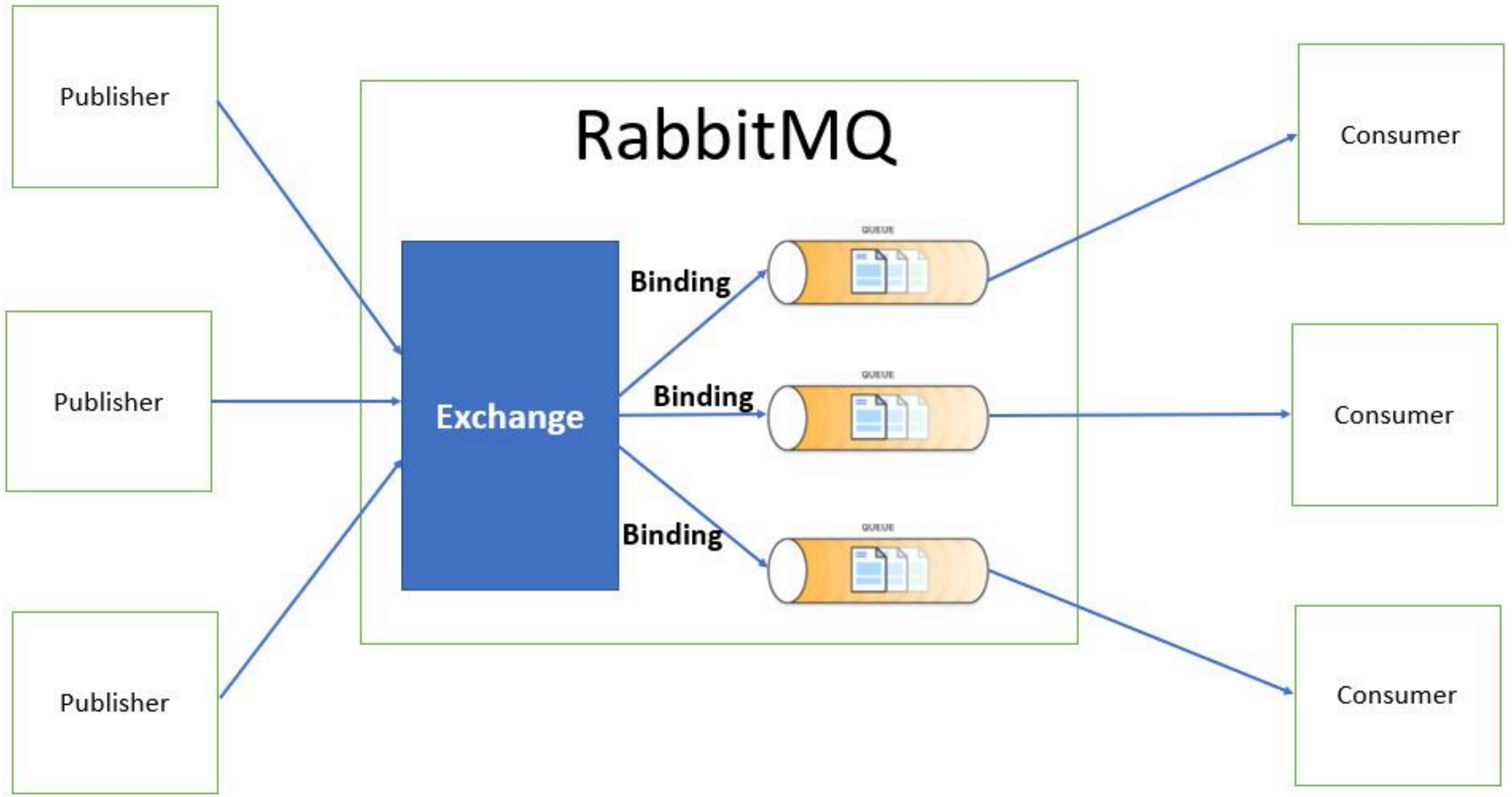
RabbitMQ Messaging Flow

When using RabbitMQ the publisher never directly sends a message to a queue. Instead, the publisher sends messages to an exchange.

Exchange is responsible for sending the message to an appropriate queue based on routing keys, bindings and header attributes.

Exchanges are message routing agents which we can define and bindings are what connects the exchanges to queues.

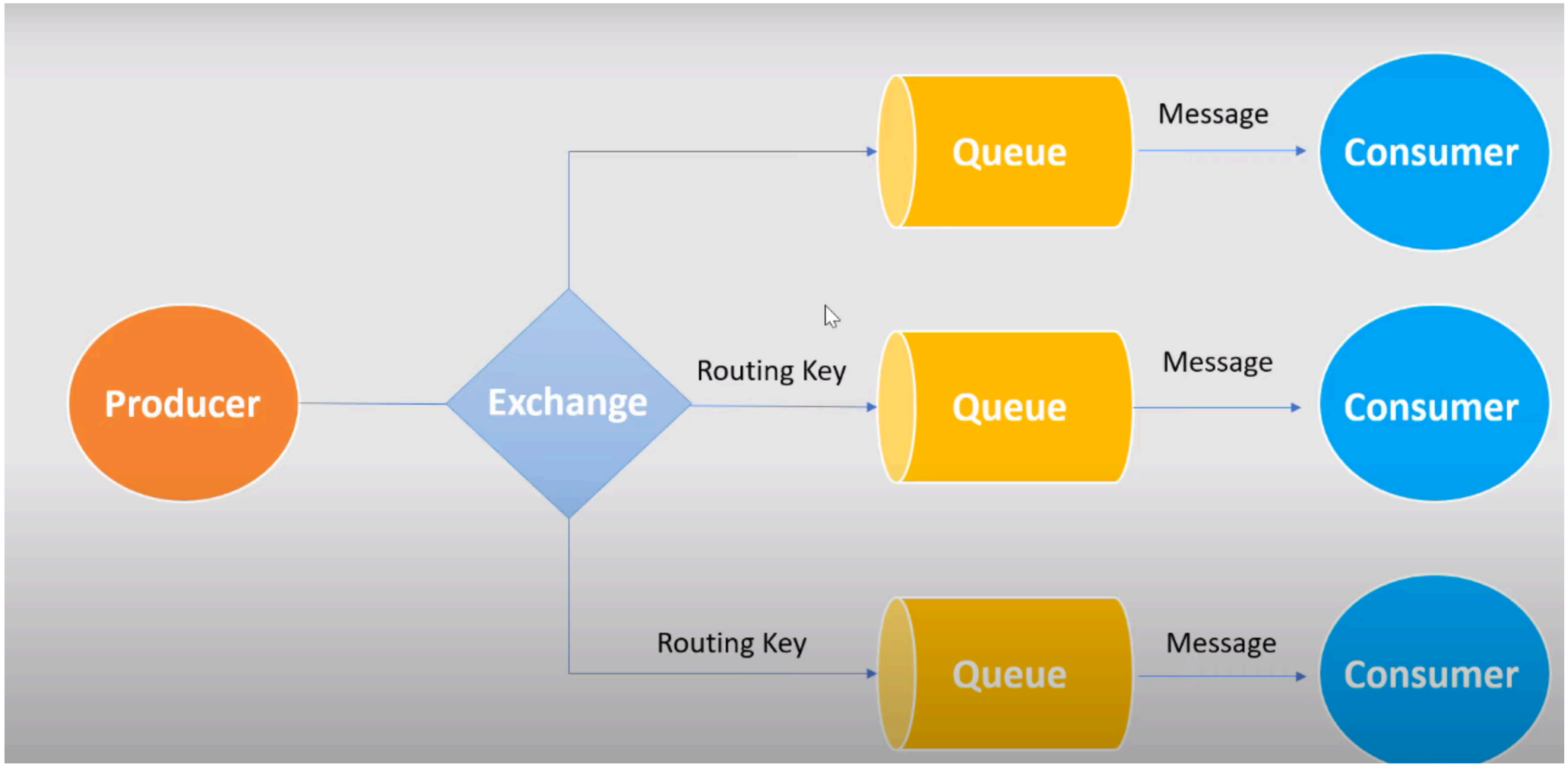
So in all our examples we will be creating first a Queue and Exchange, then bind them together.



Busy Coder Academy

Spring boot rabbitMQ hello world

Create spring boot project with web, ampq dependencies



Busy Coder Academy

Configuration rabbitmq

```java
@Configuration
public class MessagingConfig {
    public static final String QUEUE = "javademo_queue";
    public static final String EXCHANGE = "javademo_exchange";
    public static final String ROUTING_KEY = "javademo_routingKey";

    @Bean
    public Queue queue() {
        return new Queue(QUEUE);
    }
    @Bean
    public TopicExchange exchange() {
        return new TopicExchange(EXCHANGE);
    }

    @Bean
    public Binding binding(Queue queue, TopicExchange exchange) {
        return BindingBuilder.bind(queue).to(exchange).with(ROUTING_KEY);
    }

    @Bean
    public MessageConverter converter() {
        return new Jackson2JsonMessageConverter();
    }
    @Bean
    public AmqpTemplate template(ConnectionFactory connectionFactory) {
        final RabbitTemplate rabbitTemplate = new RabbitTemplate(connectionFactory);
        rabbitTemplate.setMessageConverter(converter());
        return rabbitTemplate;
    }
```

Request and response objects

```java
@Data
@AllArgsConstructor
@NoArgsConstructor
@ToString
public class Order {

    private String orderId;
    private String name;
    private int qty;
    private double price;
}
```

```java
@Data
@AllArgsConstructor
@NoArgsConstructor
@ToString
public class OrderStatus {

    private Order order;
    private String status;//progress,completed
    private String message;
}
```

Order Producer

```java
@RestController
@RequestMapping("/order")
public class OrderPublisher {

    @Autowired
    private RabbitTemplate template;

    @PostMapping("/{restaurantName}")
    public String bookOrder(@RequestBody Order order, @PathVariable String restaurantName) {
        order.setOrderId(UUID.randomUUID().toString());
        //restaurantservice
        //payment service
        OrderStatus orderStatus = new OrderStatus(order, "PROCESS", "order placed succesfully in " + restaurantName);
        template.convertAndSend(MessagingConfig.EXCHANGE, MessagingConfig.ROUTING_KEY, orderStatus);
        return "Success !!";
    }
}
```

Order Consumer

```
@Component
public class OrderConsumer {

    @RabbitListener(queues = MessagingConfig.QUEUE)
    public void consumeMessageFromQueue(OrderStatus orderStatus) {
        System.out.println("Message recieved from queue : " + orderStatus);
    }
}
```

RabbitMQ types of Exchanges

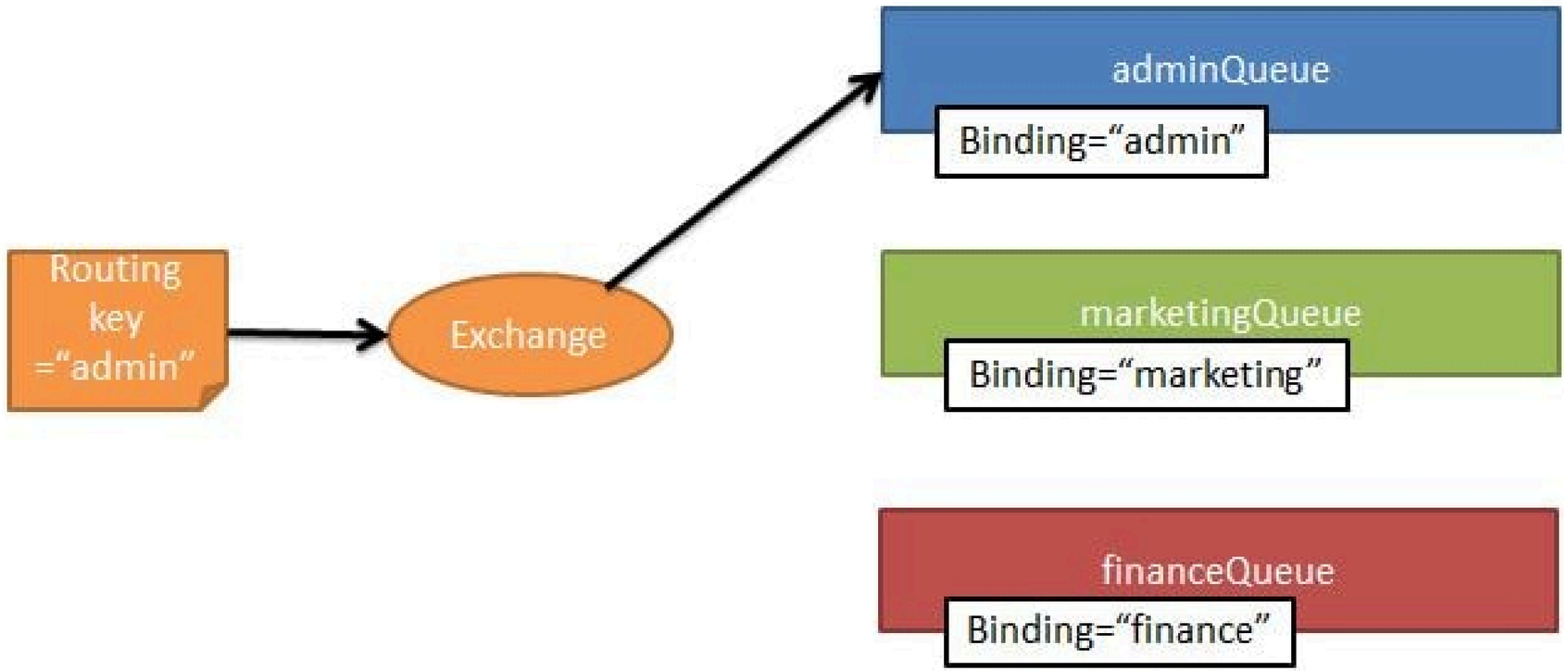
With RabbitMQ we have the following types of Exchanges-

Direct Exchange
Fanout Exchange
Topic Exchange
Header Exchange

Direct Exchange

Based on the routing key a message is sent to the queue having the same routing key specified in the binding rule.

The routing key of exchange and the binding queue have to be an exact match. A message is sent to exactly one queue.



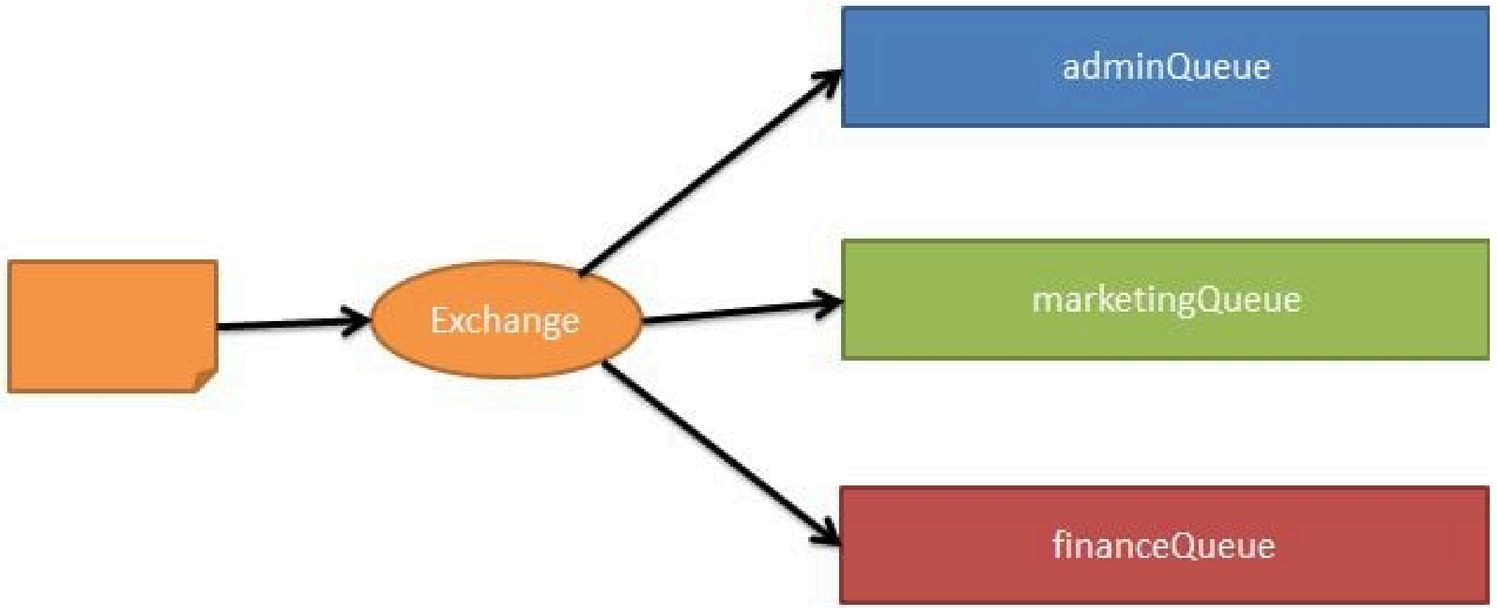
Busy Coder Academy

Fanout Exchange

The message is routed to all the available bounded queues.

The routing key if provided is completely ignored.

So this is a kind of publish-subscribe design pattern.

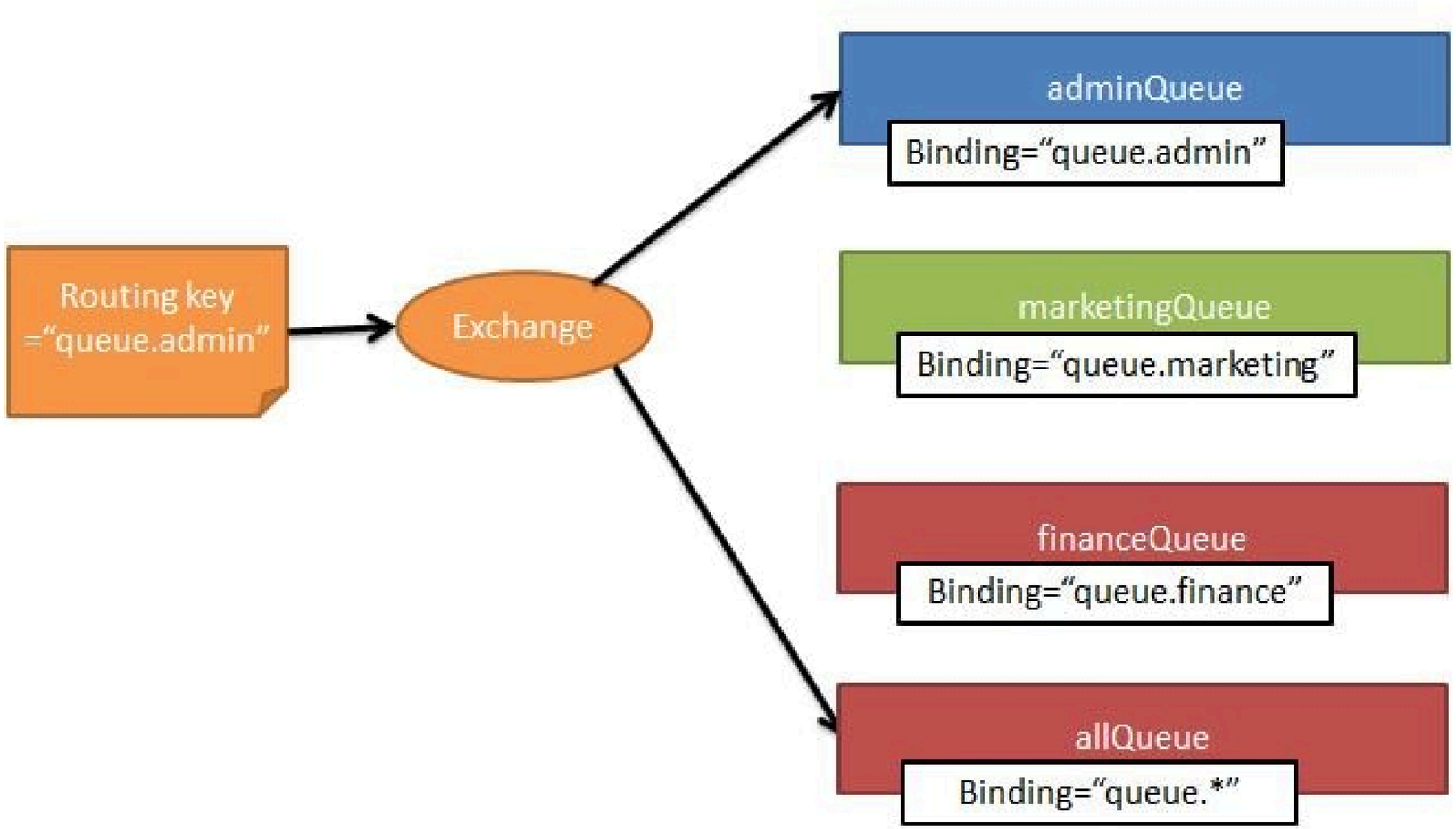


Topic Exchange

Here again the routing key is made use of.

But unlike in direct exchange type, here the routing key of the exchange and the bound queues should not necessarily be an exact match.

Using regular expressions like wildcard we can send the exchange to multiple bound queues.

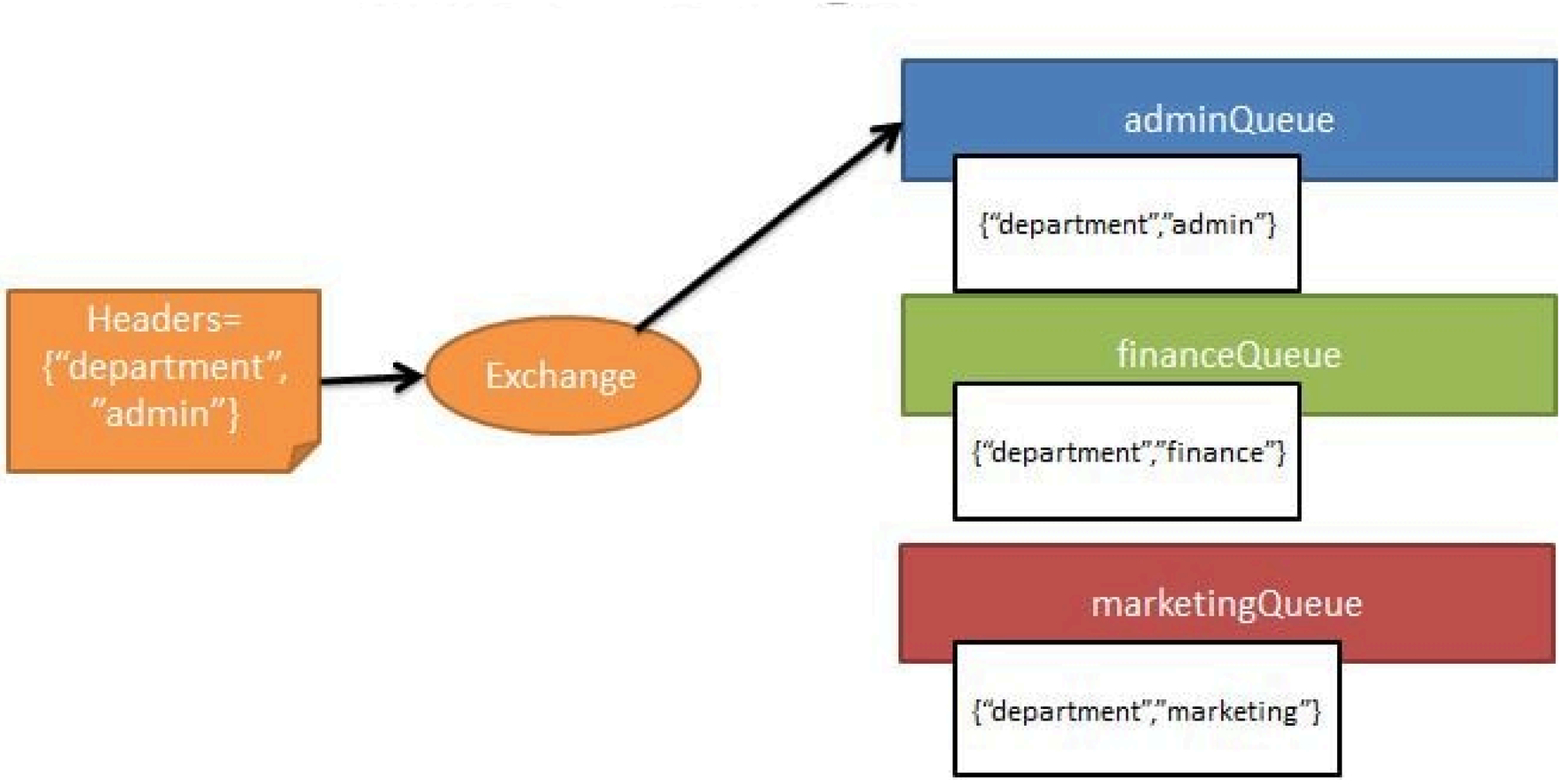


Busy Coder Academy

Header Exchange

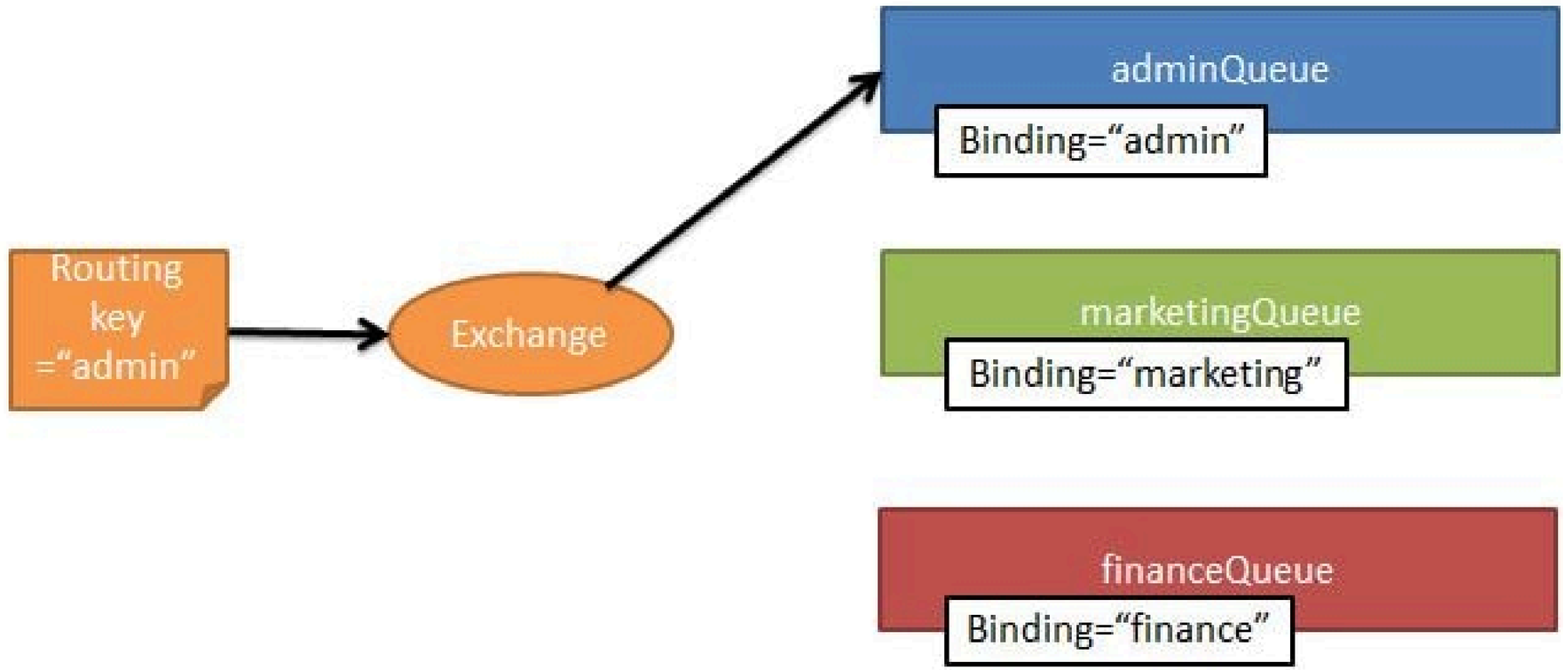
In this type of exchange the routing queue is selected based on the criteria specified in the headers instead of the routing key.

This is similar to topic exchange type, but here we can specify complex criteria for selecting routing queues.



Busy Coder Academy

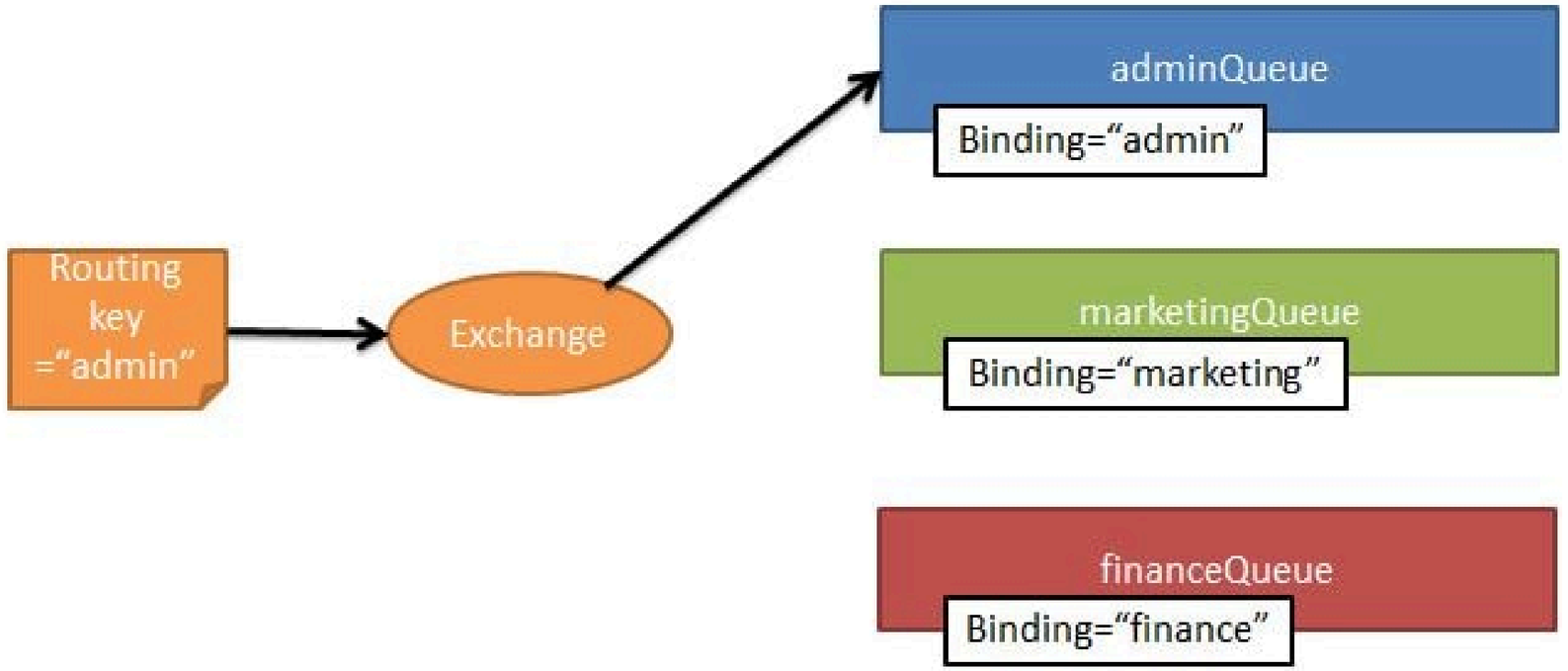
Direct exchange in details



Direct Exchange

Based on the routing key a message is sent to the queue having the same routing key specified in the binding rule.

The routing key of exchange and the binding queue have to be an exact match. A message is sent to exactly one queue.



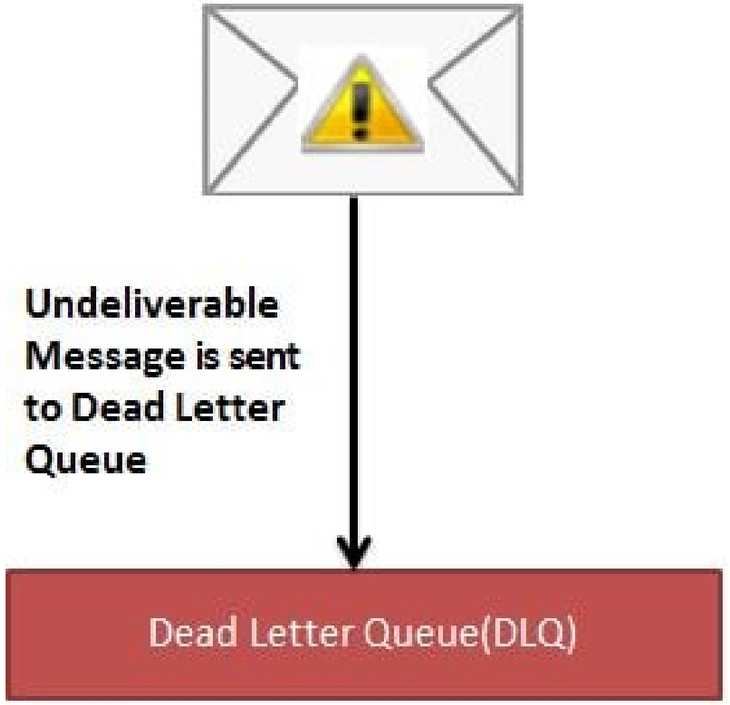
Retry and Error Handling

If exception exists after maximum retries then put message in a dead letter queue where it can be analyzed and corrected later.

What is a Dead Letter Queue?

In English vocabulary Dead letter mail is an undeliverable mail that cannot be delivered to the addressee.

A dead-letter queue (DLQ), sometimes which is also known as an undelivered-message queue, is a holding queue for messages that cannot be delivered to their destinations due to some reason or other.



Retry and Error Handling

In message queueing the dead letter queue is a service implementation to store messages that meet

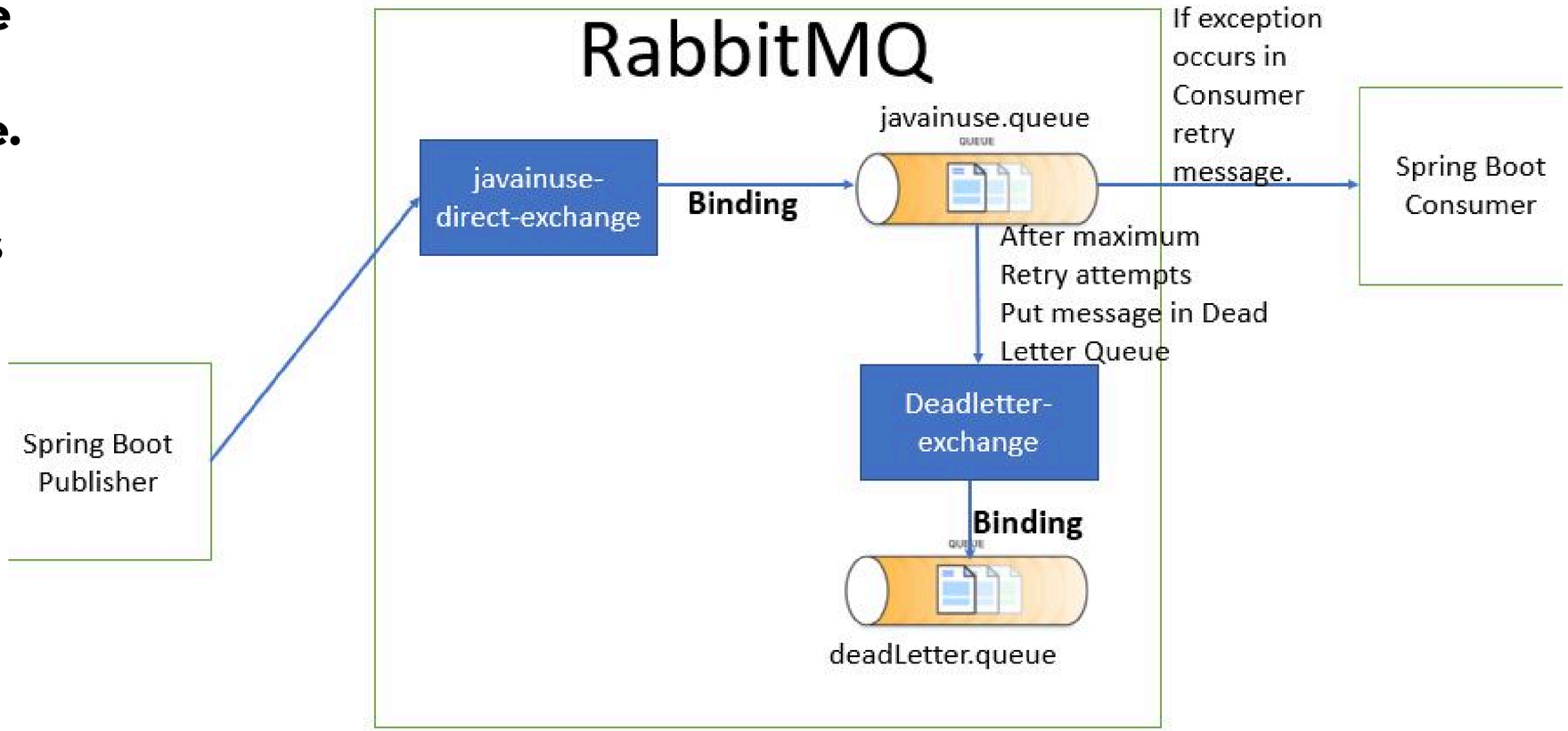
one or more of the following failure criteria:

- **Message that is sent to a queue that does not exist.**
- **Queue length limit exceeded.**
- **Message length limit exceeded.**
- **Message is rejected by another queue exchange.**
- **Message reaches a threshold read counter number, because it is not consumed. Sometimes this is called a "back out queue".**

Retry and Error Handling

Spring Boot Producer Module - It will produce a message and put it in RabbitMQ queue. It will also be responsible for creating the required queues including the dead letter queue.

Spring Boot Consumer Module - It will consume a message from RabbitMQ queue. We will be throwing an exception and then retrying the message. After maximum retries it will then be put in dead letter queue.



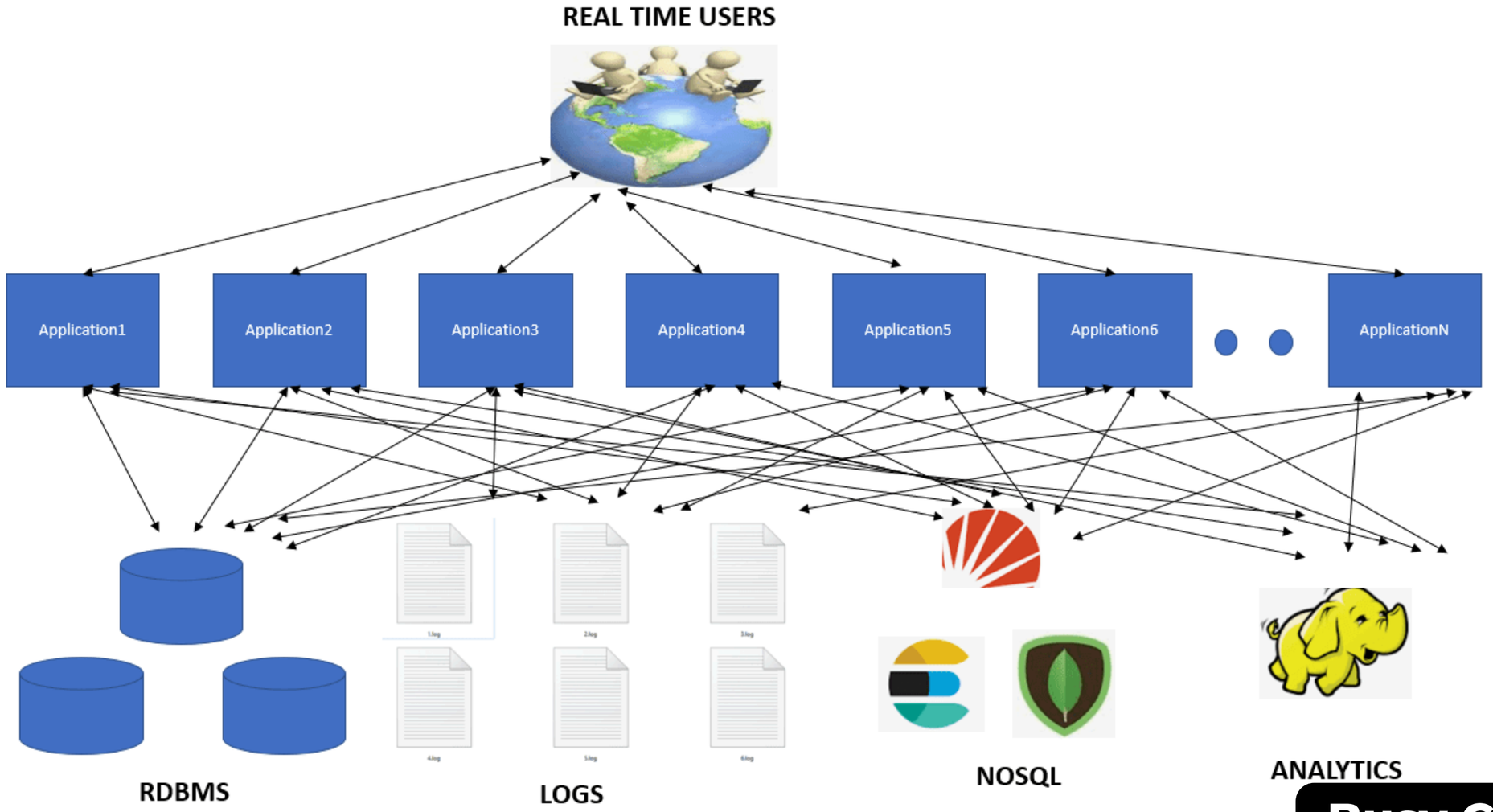
Busy Coder Academy

Messaging with Kafka

rgupta.mtech@gmail.com

What is the Need of Apache Kafka?

Complex web of applications involving point to point data movement. This involves moving large amount of data from one point to another.

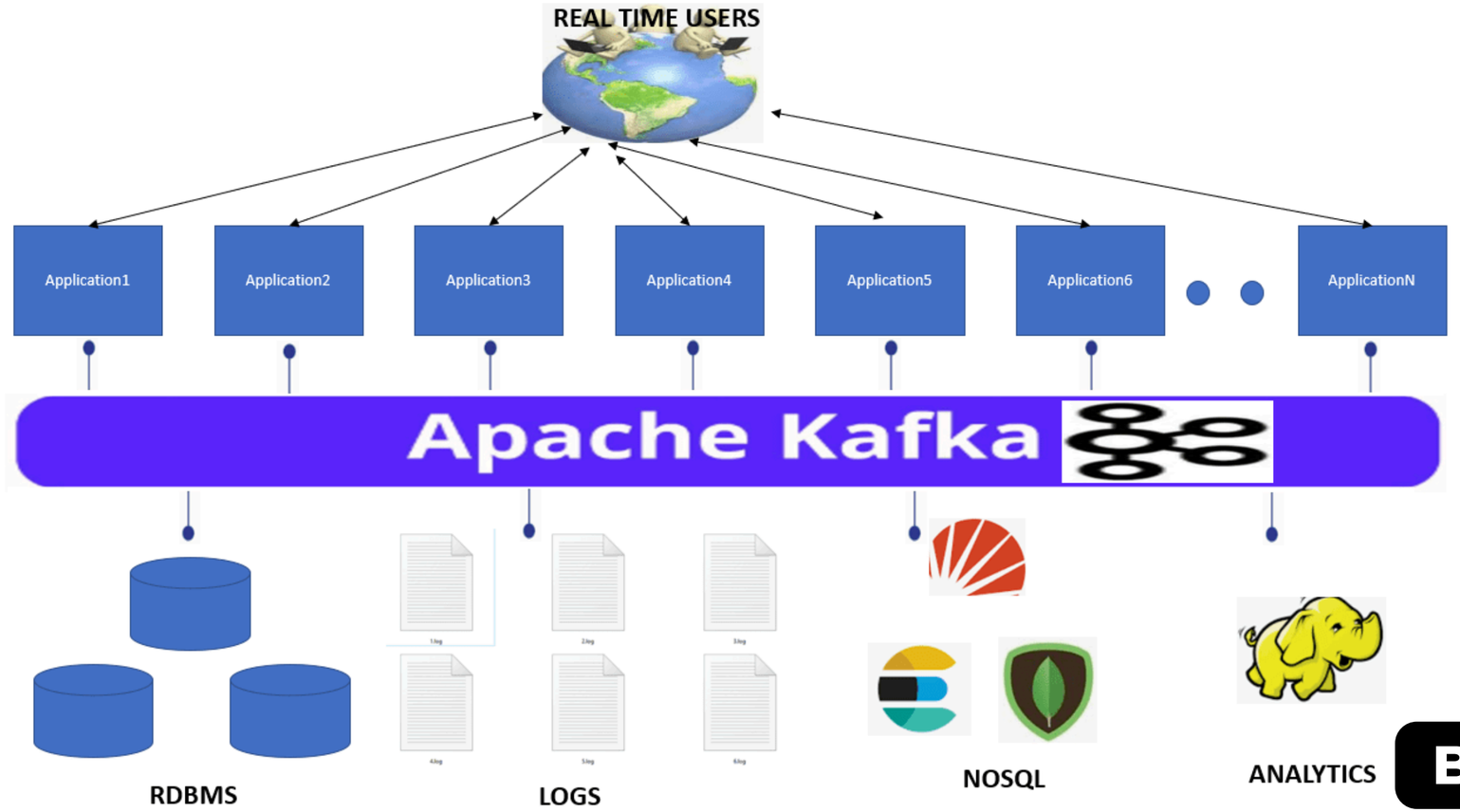


What is Apache Kafka?

Apache Kafka is an open-source stream-processing software platform developed by LinkedIn and donated to the Apache Software Foundation.

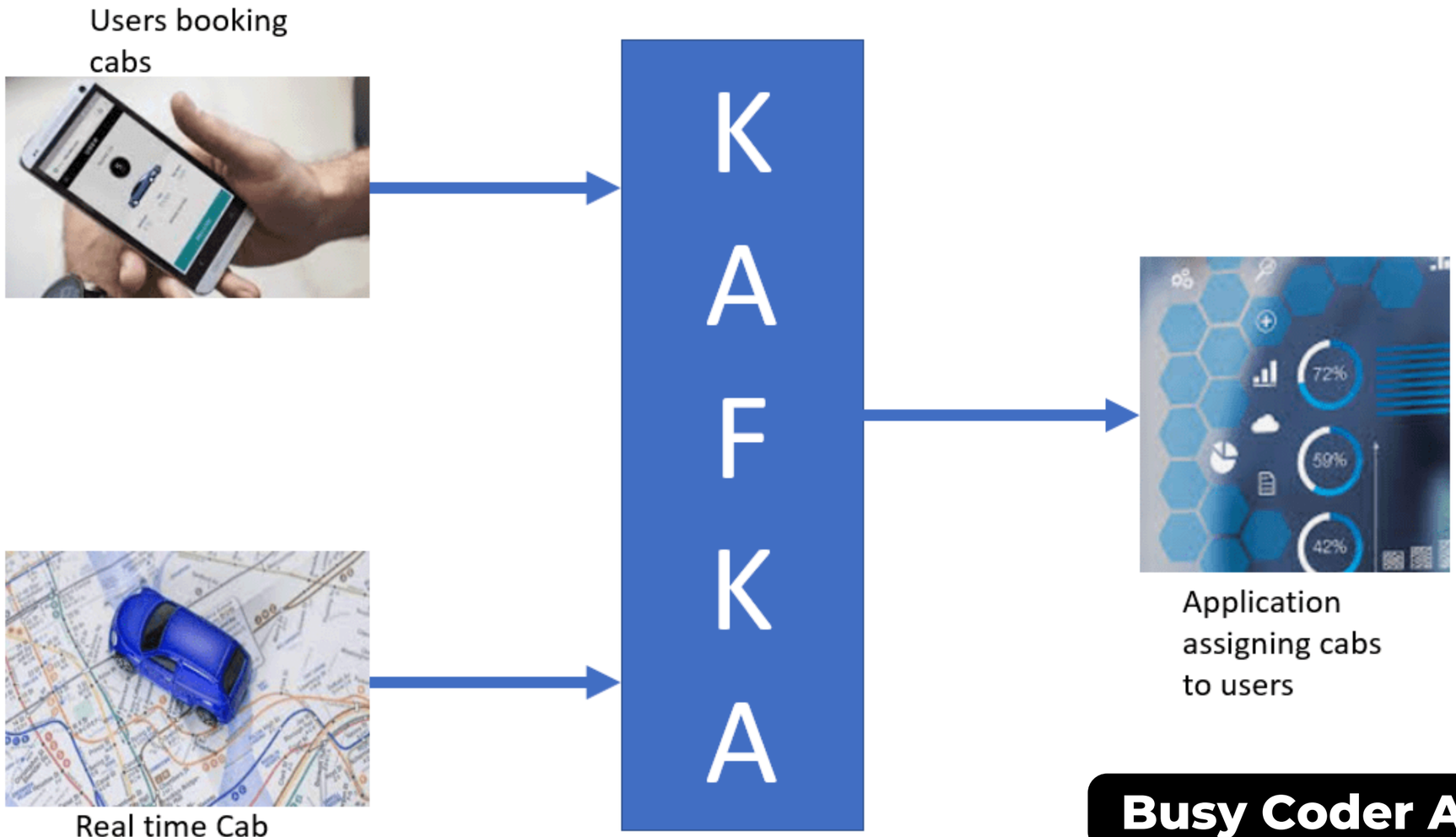
It has been developed using Java and Scala.

Apache Kafka is a high throughput distributed messaging system for handling real-time data feeds.



Apache Kafka Usages

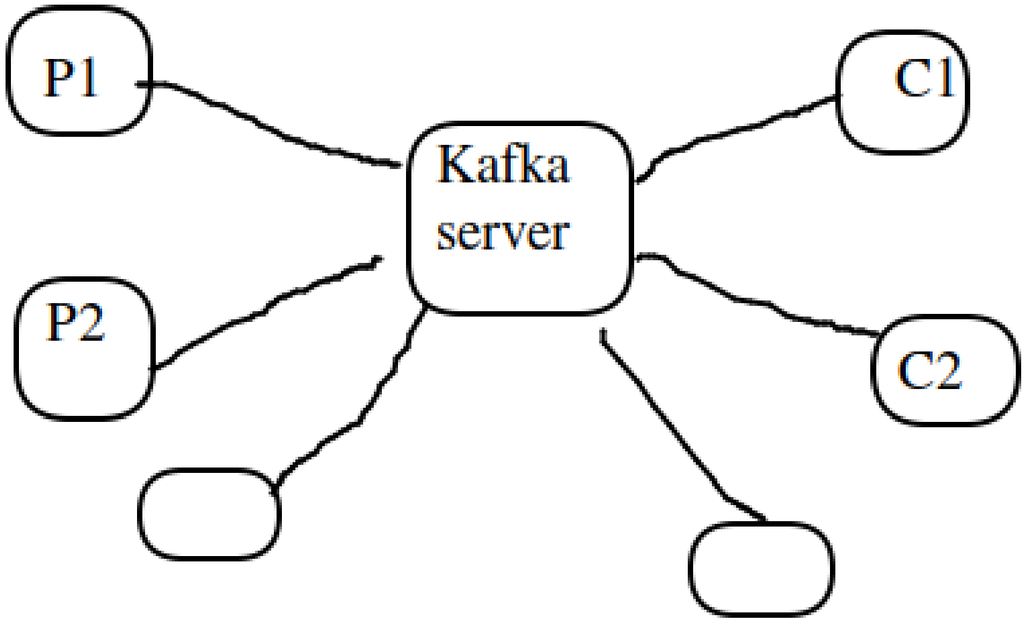
Real time example of Apache Kafka is Uber cab booking service. Uber makes use of Kafka to send User and Cab information to Uber Cab Booking System.



What is Apache Kafka?

Event driven arch, millions of messages can be process per second

multiple producer

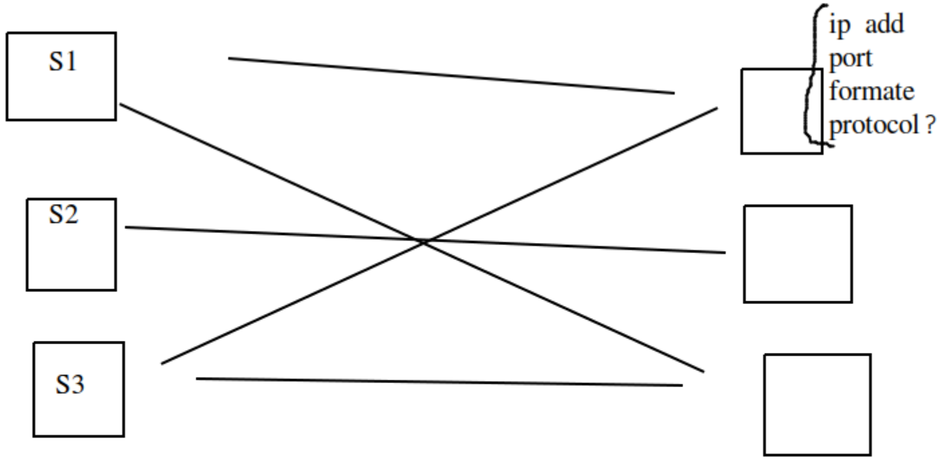


multiple broker and consumer

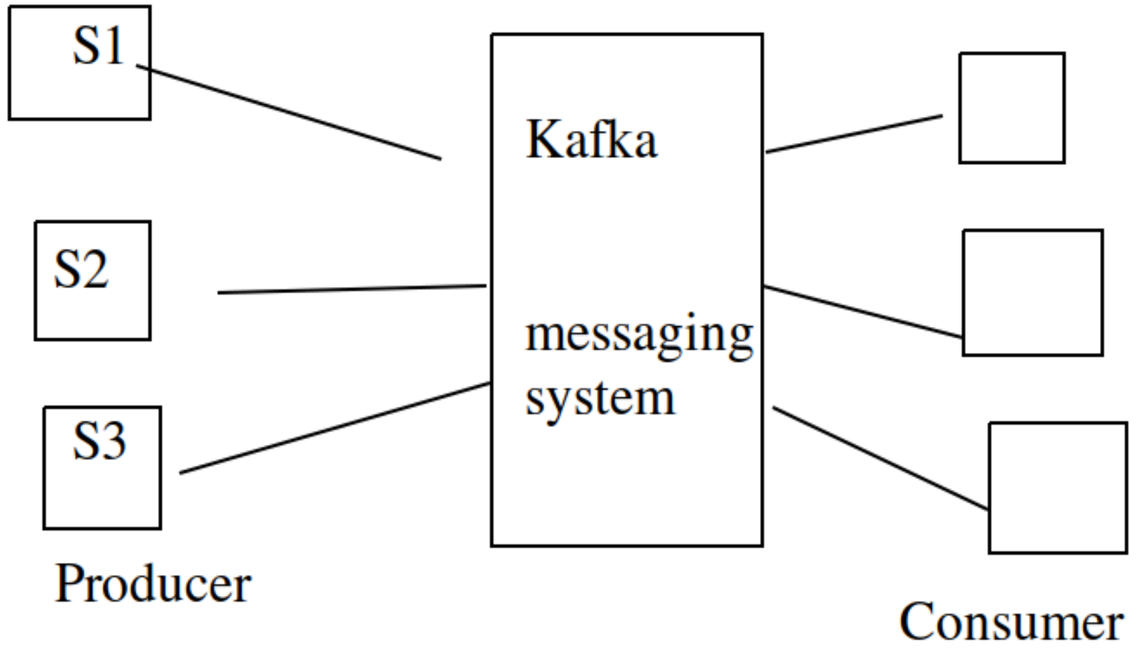
Apache Kafka is an open-source stream-processing software platform

In distributed env. kafka is reffered as kafka cluster made of more then one kafka server

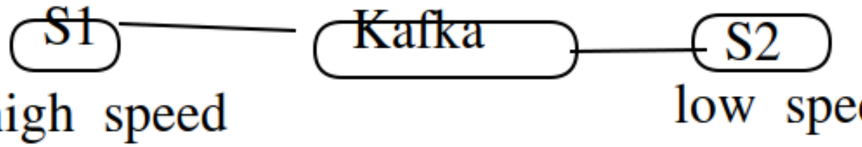
Without Kafka



Decoupling data processing pipeline

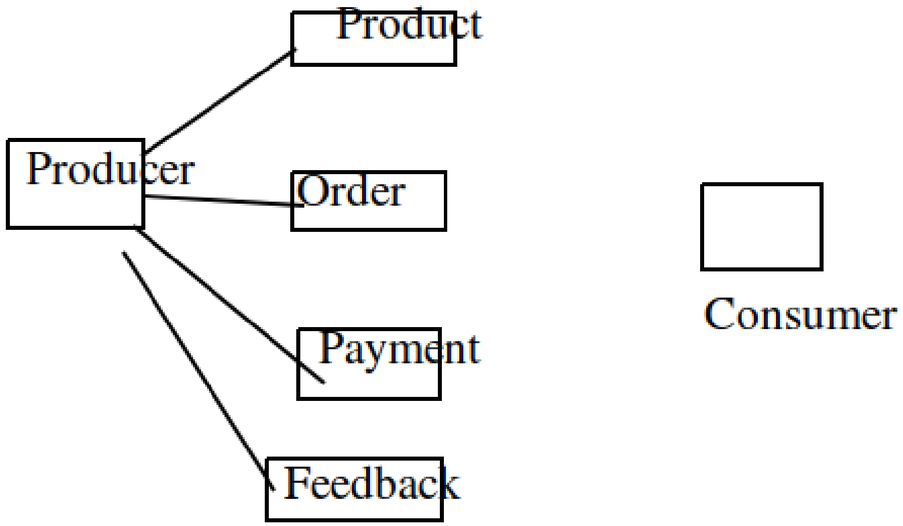


It solve Complex communication problem
it solve speed mismach problem

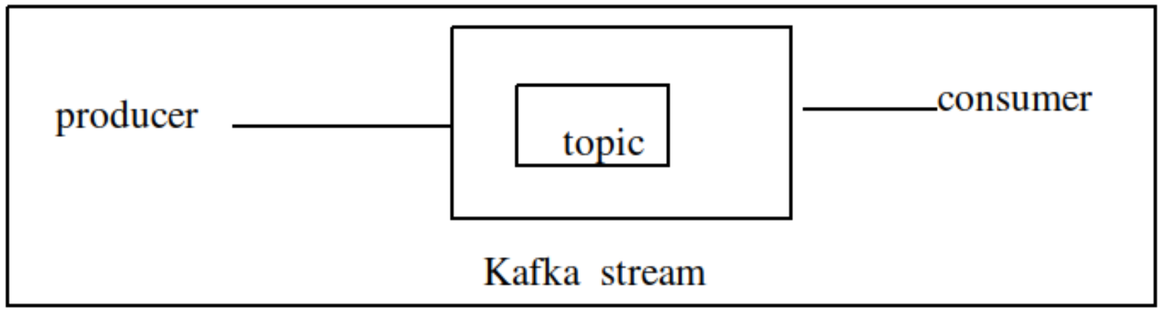


What is topic

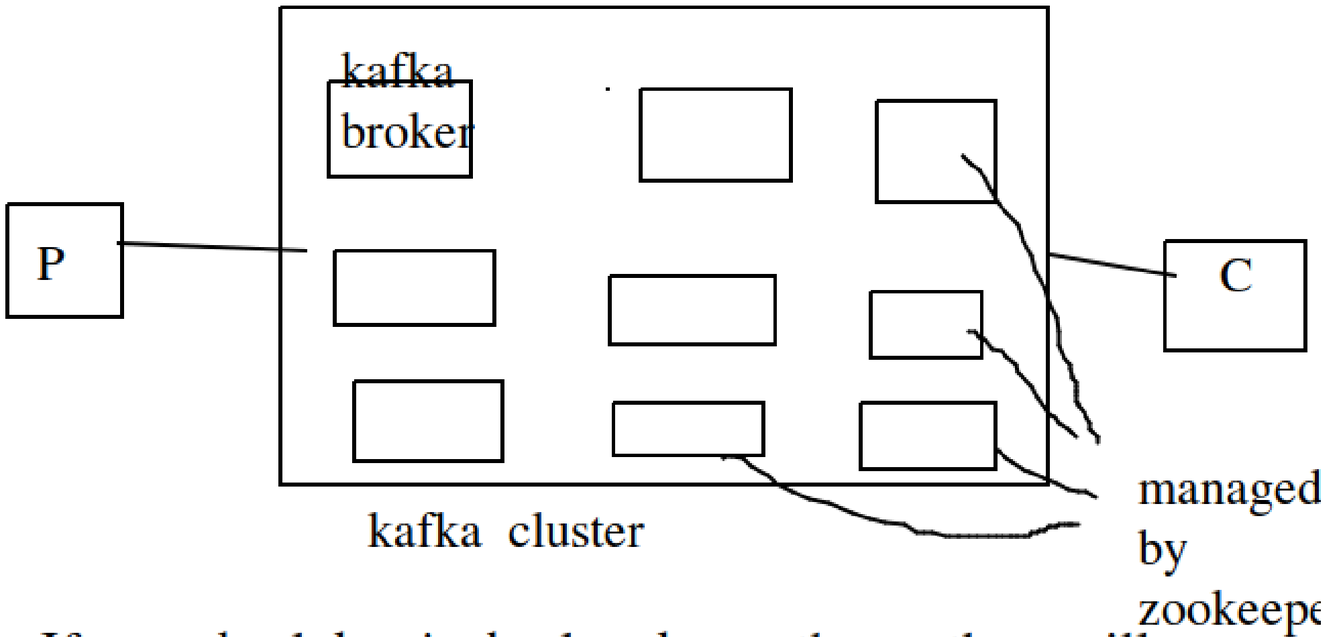
What if producer is sending 4 type of data?
Consumer get confused if he is only intrested on product data



Solution: segrate the data streams=> 4 topics for each category of data
similer to database table: related to one type of data



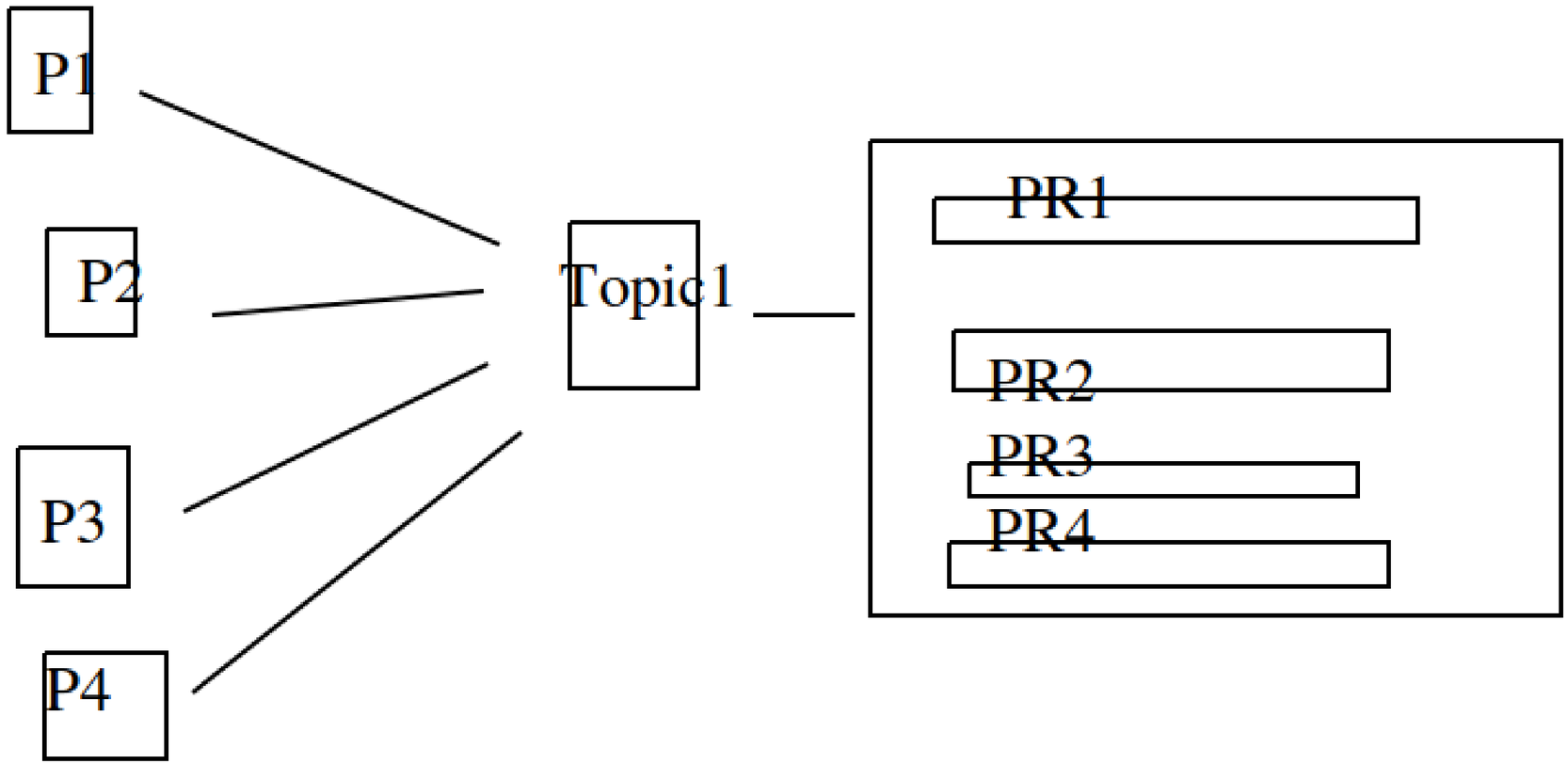
Scalability and fault tolerence(Zookeeper)



If any brokder is broke down then other will take care

How to manage it?
if one broker is broken down then who coordinate
Distributed service to manage lare amout of host
Focus on BL and not on distribution of logic
First we have to start the zookeeper and then kafka broker

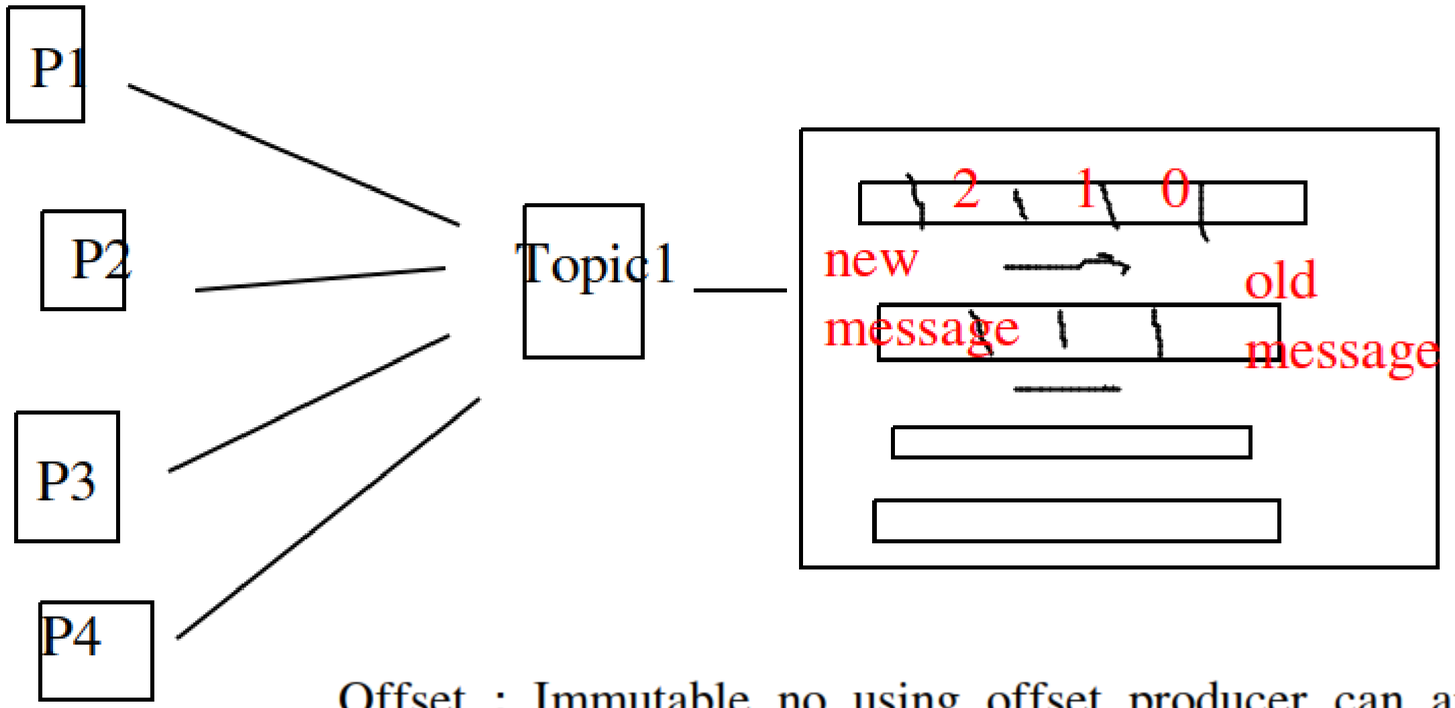
Problem of parallelism



We need to decide not of partitions while creating topics
we need to tell no of partitions

Partitions offset

Producer send the data into message offset system

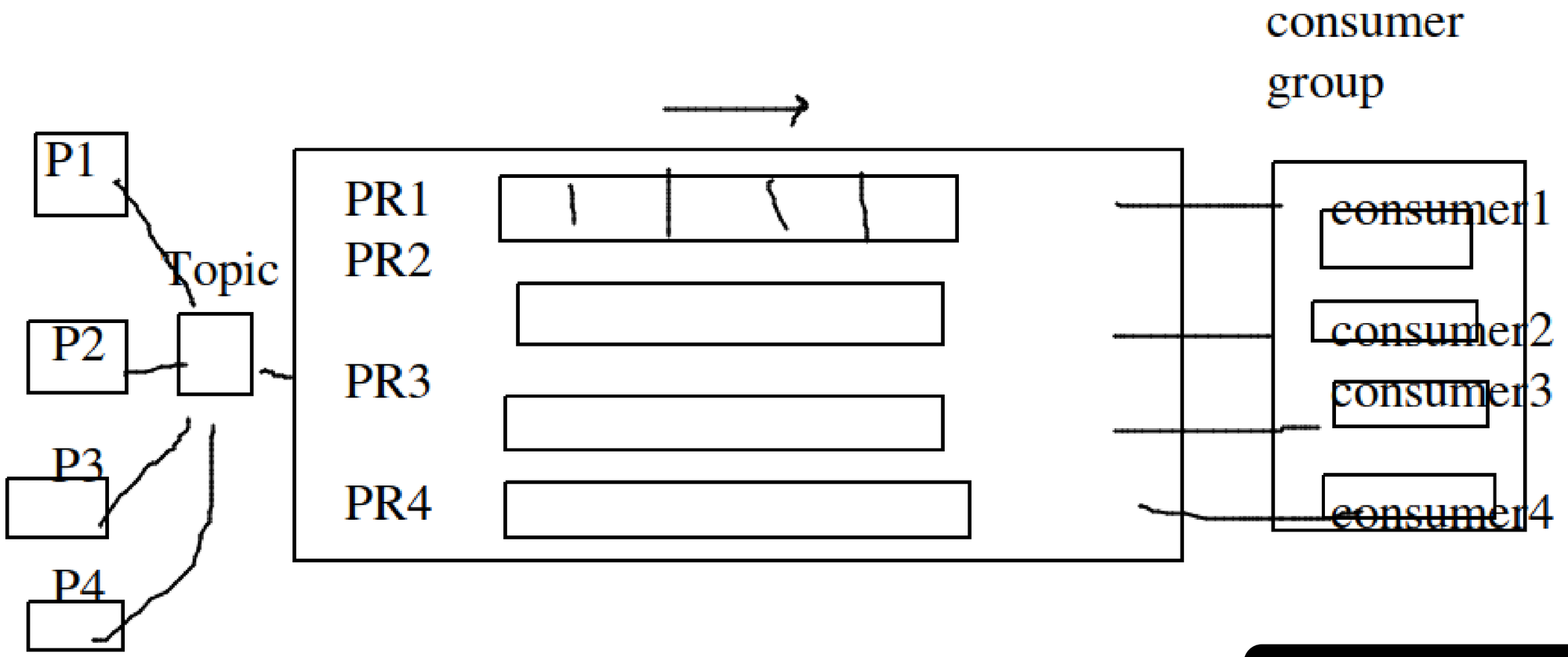


Offset : Immutable no using offset producer can arrange data into accending /decending order

Partition 1 data is not same as partion 2 data ⇒ To recognize message
which topic id?
which partition id?
which offset id?

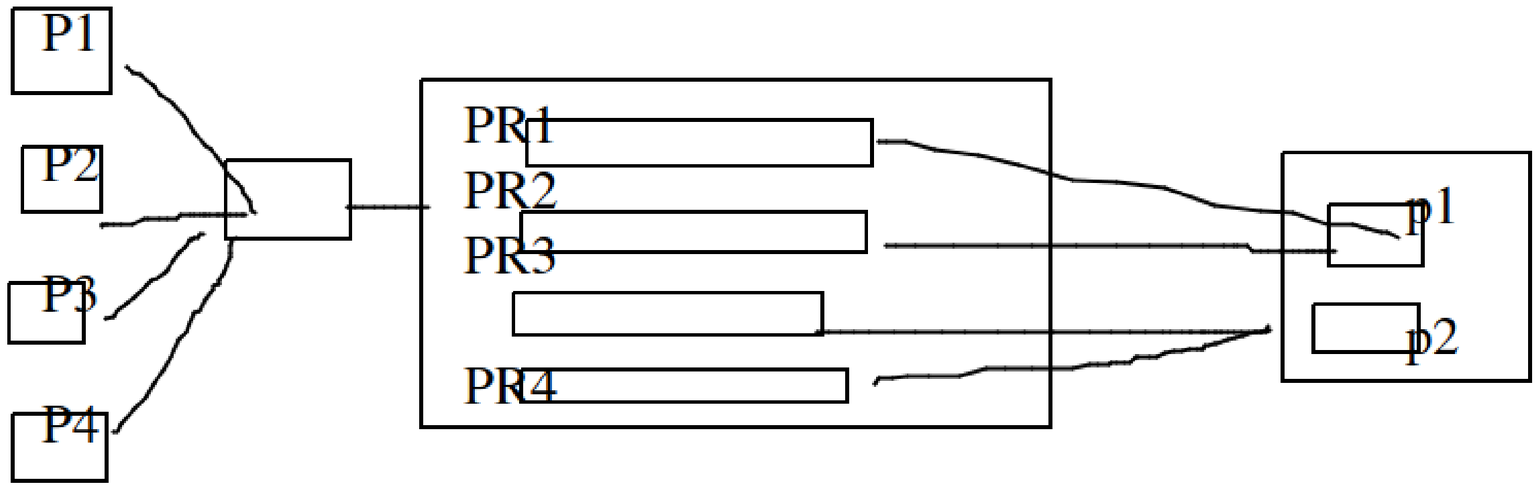
Consumer Group

- **We have only one consumer how data processing happens**
- **You create many consumer and connect one partion**
- **We can get all the data into one shot from 4 partitions, that is called consumer group**
- **Consumer group: single logical using they can share the work**



Consumer Group

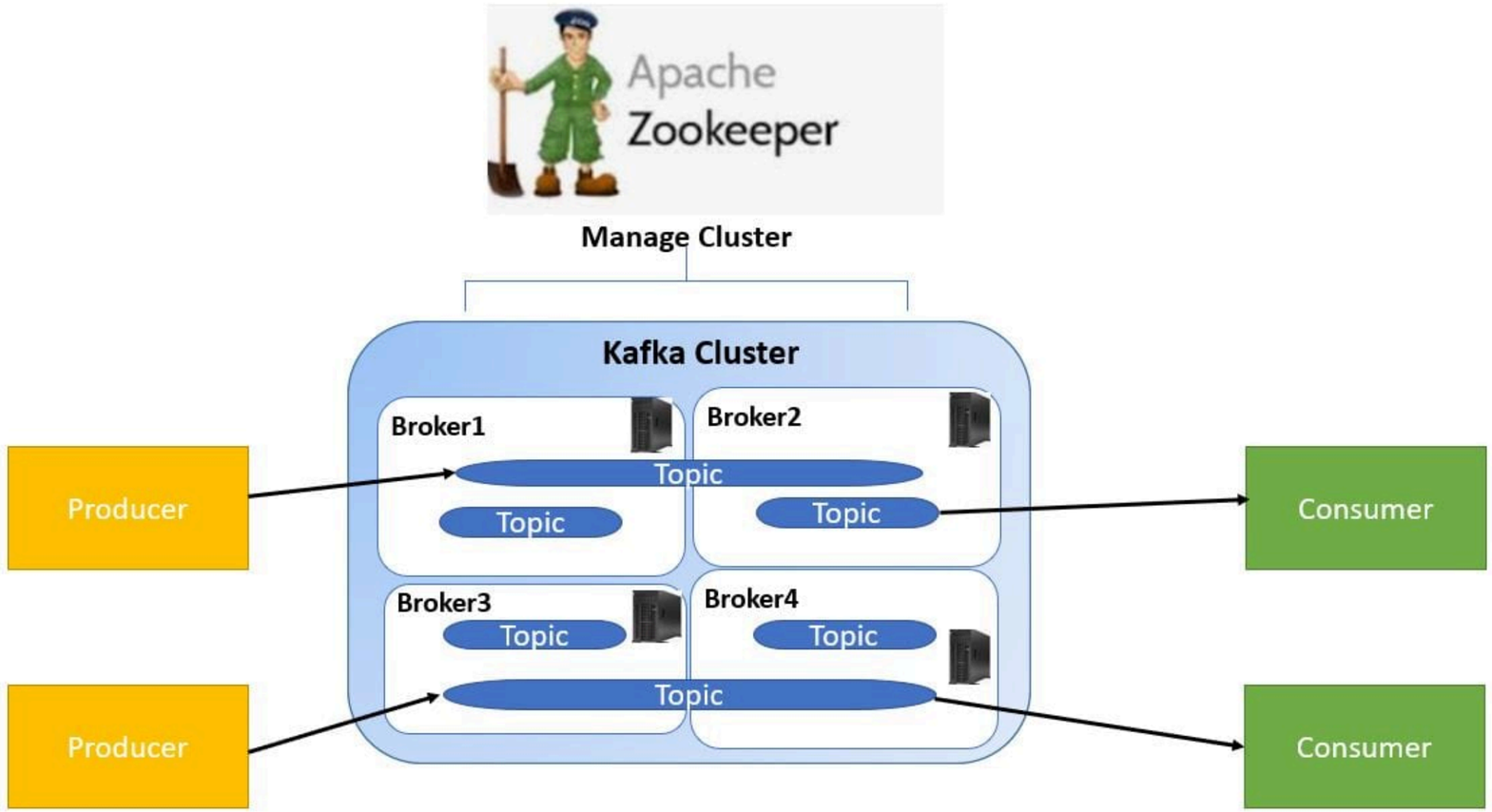
- **Case I: if only 2 consumer is there in consumer group**



- **Case 2: if 4 consumer is there , each one take data from each partition**
- **Case 3: if we have 5 consumer group, 1 will be idel**
- **Case 4: if one consumer then all data send to that**

Replication factor

- Consider topic1 with 4 partition, then not all partition go into the broker
- Partition will be distribtued into multiple broker
- What if broker 2 is gone?
- How to solve the issue => replication copy
- If RF =3 then T1P1 should be replicated to 3 places in different brokers
- Although T1P1 is at 3 places one of them is called leader
- Kakfa zooker send data to the leader and then leader distributed/ replicate to others



Busy Coder Academy

Zookeeper

- **To manage the cluster we make use of Apache Zookeeper. Apache Zookeeper is a coordination service for distributed application that enables synchronization across a cluster.**
- **Zookeeper can be viewed as centralized repository where distributed applications can put data and get data out of it.**
- **It is used to keep the distributed system functioning together as a single unit, using its synchronization, serialization and coordination goals, selecting leader node.**