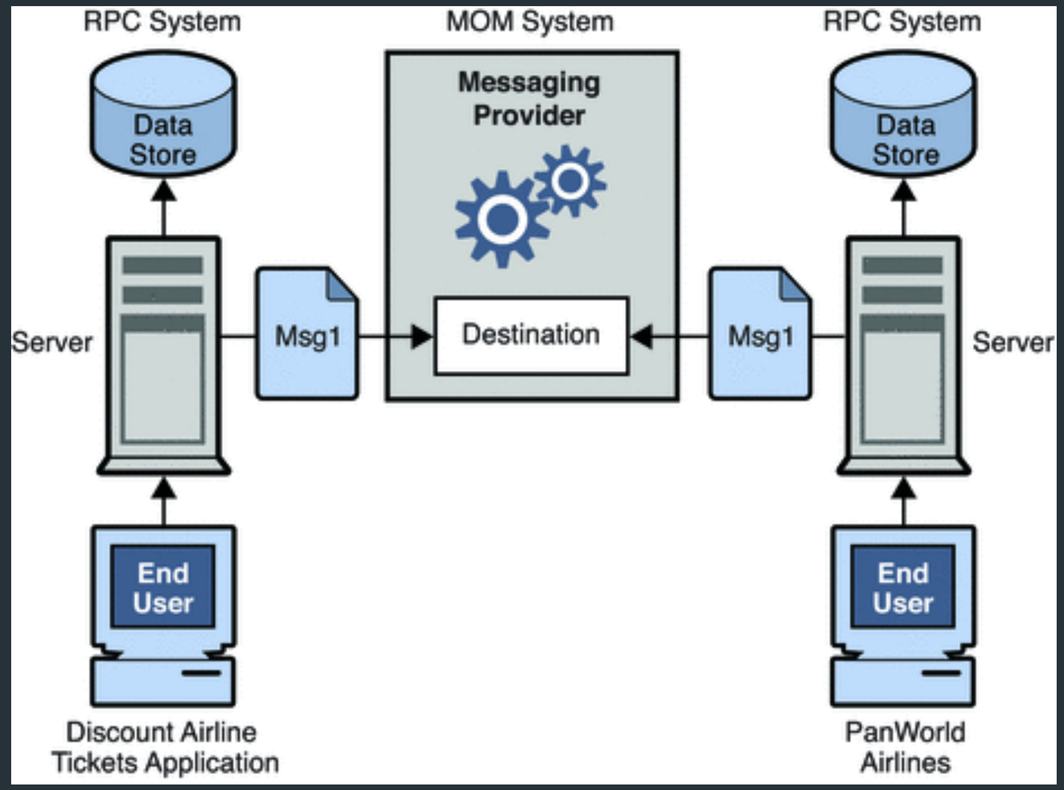
Lecture #8
MOM - Message Oriented Middleware

WSMT2023

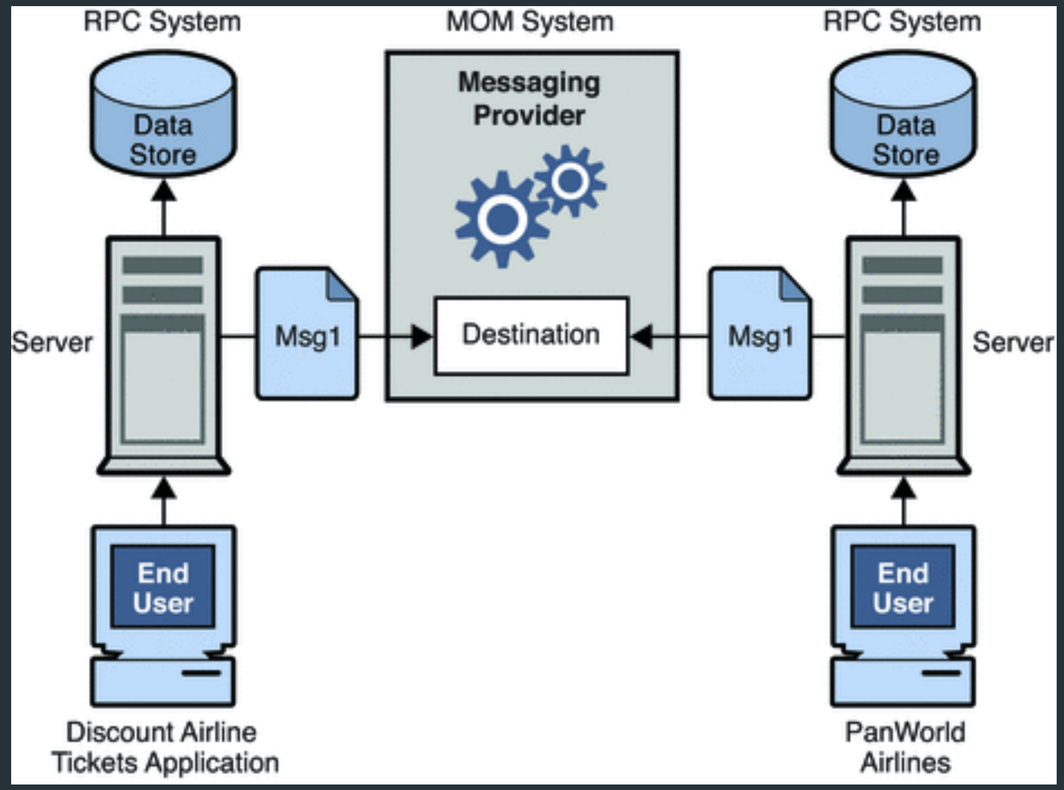
Message Oriented Middleware

- Message-oriented middleware (MOM) is a software platform that facilitates the exchange of messages between applications.
- MOM provides a number of benefits, including:
  - Loose coupling.
  - Scalability.
  - Reliability.



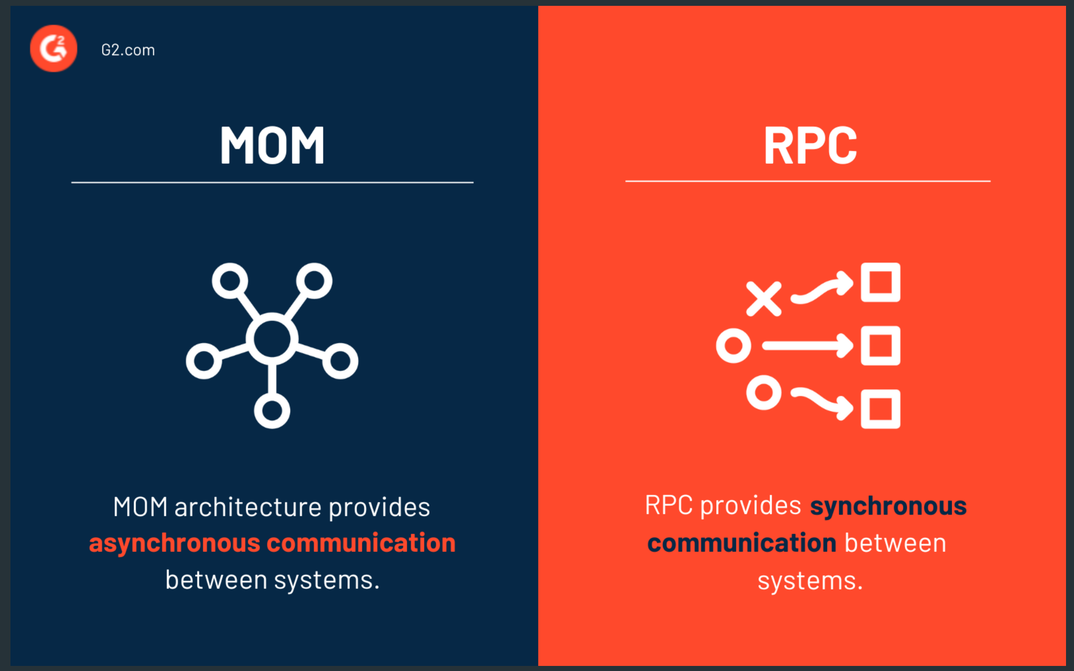
Message Oriented Middleware

- Message-oriented middleware (MOM) is a software platform that facilitates the exchange of messages between applications.
- MOM provides a number of benefits, including:
  - Loose coupling.
  - Scalability.
  - Reliability.



Fundamental Concepts

- A message is a unit of data that is exchanged between applications.
- A message broker is a software component that routes messages between applications.
- A queue is a temporary storage location for messages.
- A topic is a logical grouping of messages.



Message

- Messages are a unit of data that can be exchanged between applications.
- Messages can contain any type of data, such as text, binary data, or XML.

```
// Create a message.
Message message = new Message();

// Set the message content.
message.setContent("This is a message.");

// Send the message to the MOM.
mom.send(message);
```

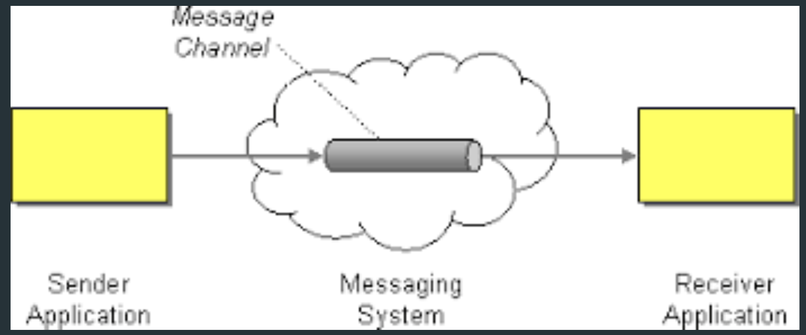
Broker

- Message broker is a software component that routes messages between applications.
- Message brokers are used to decouple applications from each other, making them more loosely coupled and easier to maintain.
- Message brokers can also provide a number of other features, such as:
  - Message transformation.
  - Message logging.
  - Message security.
  - Message monitoring.

```
// Create a message listener.
MessageListener listener = new MessageListener() {

    @Override
    public void onMessage(Message message) {
        // Do something with the message.
    }
};

// Register the message listener with the message broker.
messageBroker.registerMessageListener(listener);
```

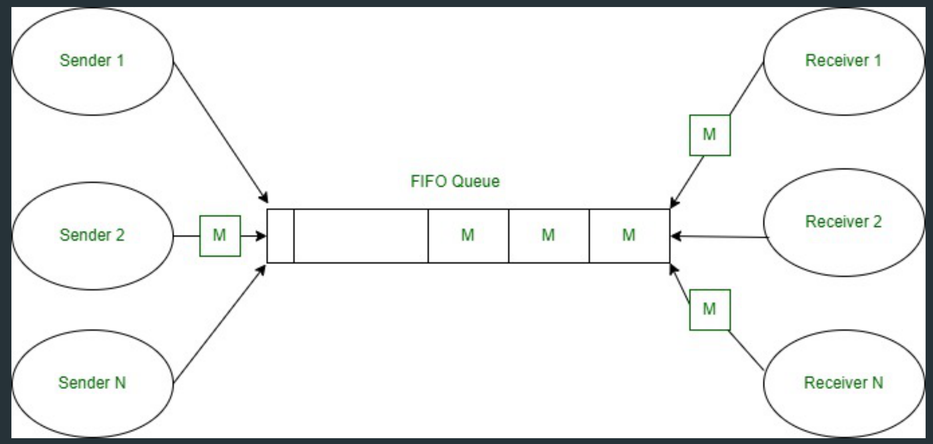
Channels

- A channel is a logical connection between two applications.
- Channels can be used to send and receive messages.
- Channels can be either point-to-point or publish-subscribe.



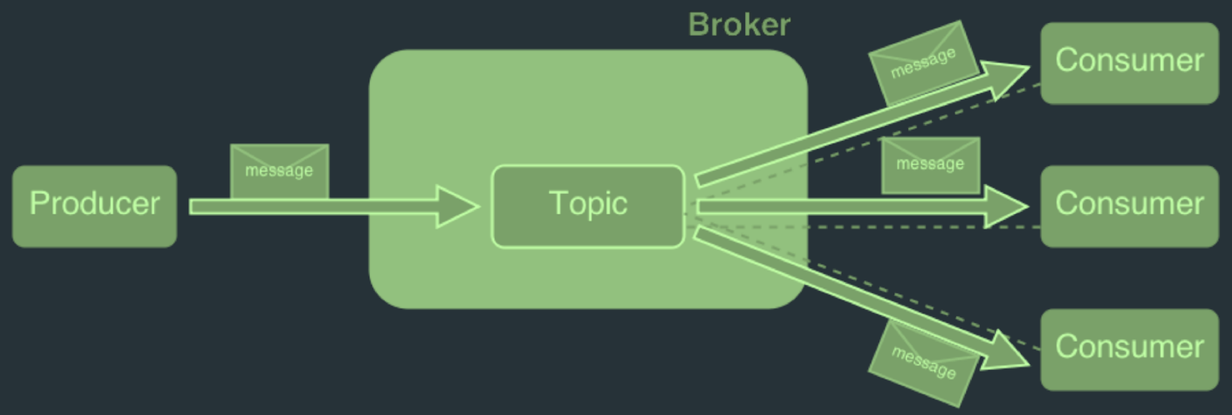
Queue

- A queue is a temporary storage location for messages.
- Messages are stored in the queue in the order in which they are received.
- Messages are processed in the order in which they are stored in the queue.

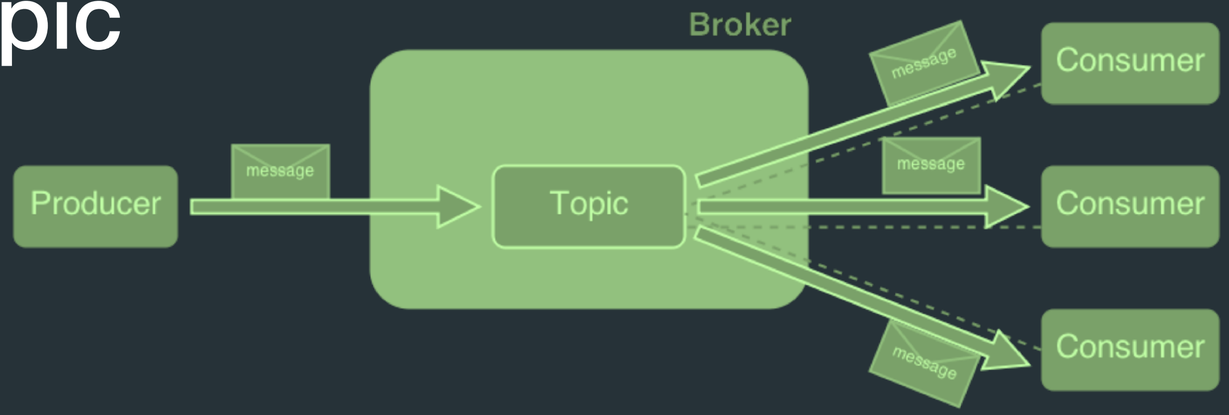


Topic

- A topic is a logical grouping of messages.
- Messages are published to a topic.
- Consumers subscribe to a topic to receive messages.



Topic

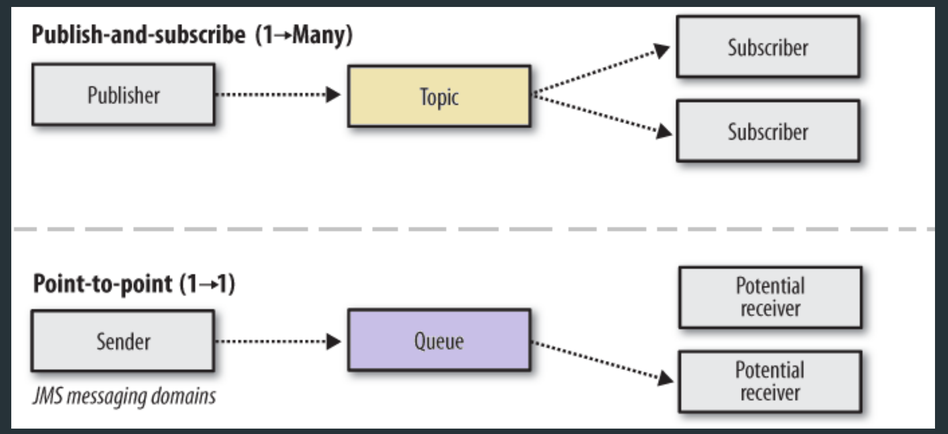


- A topic is a logical grouping of messages.
- Messages are published to a topic.
- Consumers subscribe to a topic to receive messages.

```
// Create a consumer.
Consumer consumer = new Consumer();

// Subscribe the consumer to the topic.
consumer.subscribe(topic);

// Receive messages from the topic.
while (true) {
  // Get the next message from the topic.
  Message message = consumer.receive();

  // Do something with the message.
  // ...
}
```

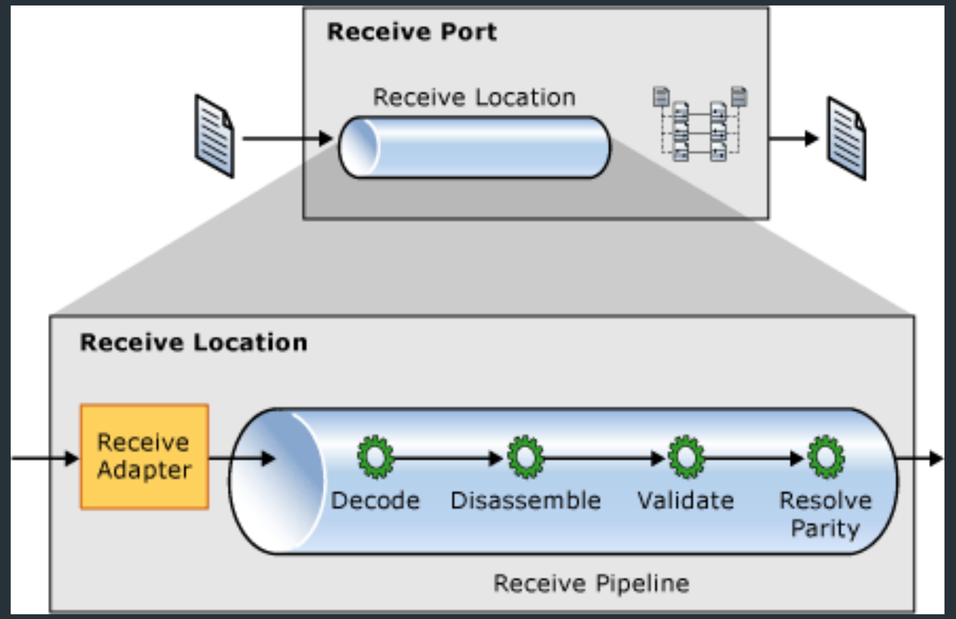
Types of MOM

- There are two main types of MOM:
- Point-to-point MOM: This type of MOM is designed for applications that need to communicate directly with each other.
- Publish-subscribe MOM: This type of MOM is designed for applications that need to publish messages to a central location and have other applications subscribe to those messages.



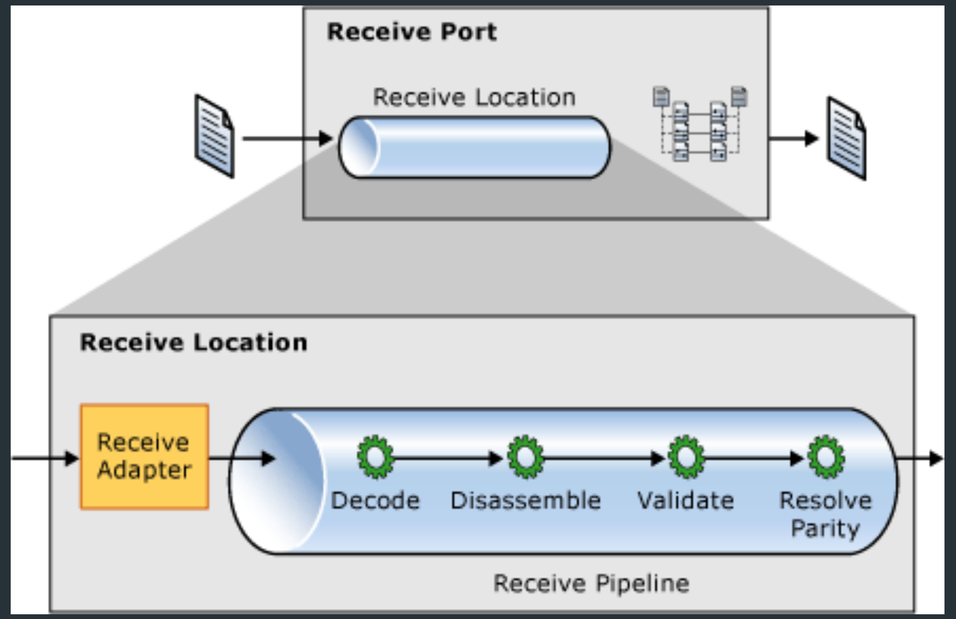
Pipelines

- A pipeline is a sequence of processing steps that are applied to a message.
- Pipelines can be used to transform, filter, and route messages.
- Pipelines can be used to implement complex business logic.



Pipeline Components

- A pipeline consists of a sequence of pipeline components.
- A pipeline component is a software component that performs a specific processing step.
- Common pipeline components include:
  - Filters
  - Aggregators
  - Routers
  - Transformers



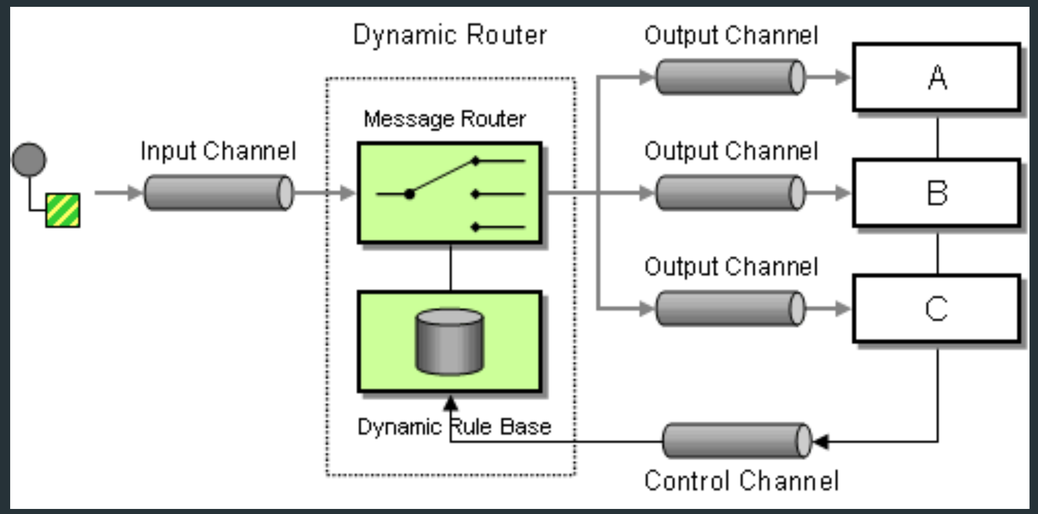
Example

```
// Create a pipeline
Pipeline pipeline = new Pipeline();

// Add a filter to the pipeline
Filter filter = new Filter();
filter.setCondition("message.body == 'Hello, World!'");
pipeline.add(filter);

// Add an aggregator to the pipeline
Aggregator aggregator = new Aggregator();
aggregator.setOperation("sum");
pipeline.add(aggregator);

// Add a router to the pipeline
Router router = new Router();
router.addRoute("message.destination == 'queue1'");
router.addRoute("message.destination == 'queue2'");
pipeline.add(router);

// Add a transformer to the pipeline
Transformer transformer = new Transformer();
transformer.setOperation("uppercase");
pipeline.add(transformer);
```

```
Pipeline pipeline = new Pipeline();

// Add a filter to the pipeline
Filter filter = new Filter();
filter.setCondition("message.body == 'Hello, World!'");
pipeline.add(filter);

// Add an aggregator to the pipeline
Aggregator aggregator = new Aggregator();
aggregator.setOperation("sum");
pipeline.add(aggregator);

// Add a router to the pipeline
Router router = new Router();
router.addRoute("message.destination == 'queue1'");
router.addRoute("message.destination == 'queue2'");
pipeline.add(router);

// Add a transformer to the pipeline
Transformer transformer = new Transformer();
transformer.setOperation("uppercase");
pipeline.add(transformer);

// Send a message through the pipeline
Message message = new Message("Hello, World!");
pipeline.send(message);
```

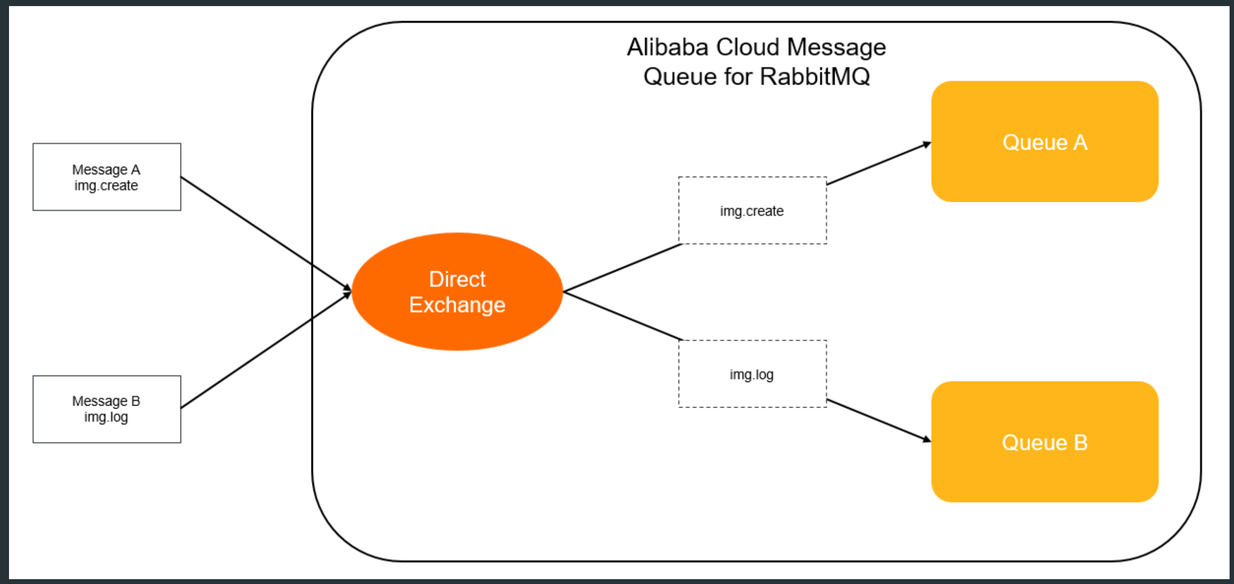
Routers

- A router is a software component that routes messages to different destinations.
- Routers can be used to implement load balancing, failover, and other routing strategies.
- Routers can be either static or dynamic.



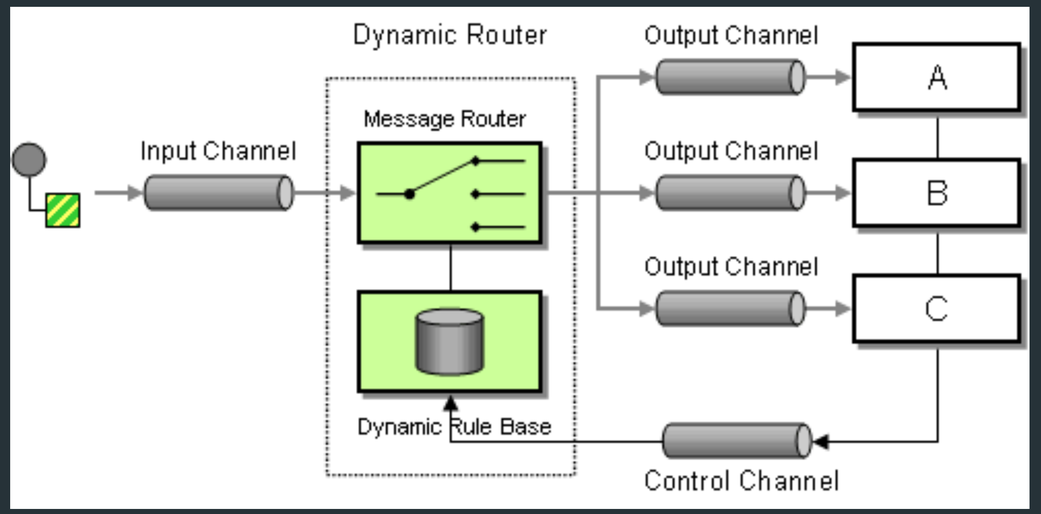
Static Routers

- A static router is a router that routes messages to a fixed set of destinations.
- Static routers are easy to configure and manage.
- Static routers are not as flexible as dynamic routers.



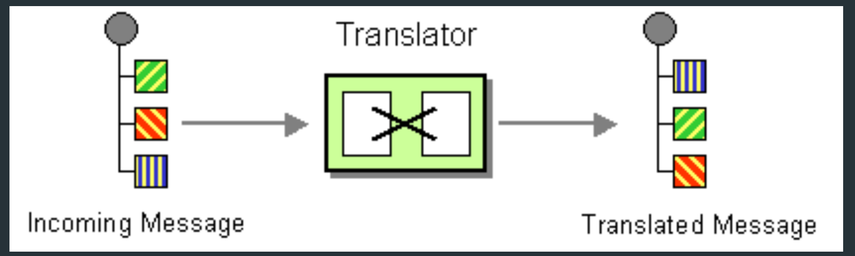
Dynamic Routers

- A dynamic router is a router that routes messages to different destinations based on their properties.
- Dynamic routers are more flexible than static routers.
- Dynamic routers can be more complex to configure and manage.



Translators

- A translator is a software component that converts messages from one format to another.
- Translators can be used to integrate applications that use different messaging formats.
- Translators can be either static or dynamic.

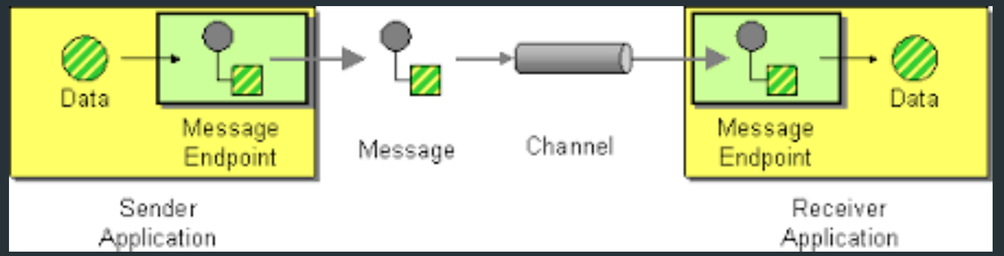


Translators

```
// Create a translator
Translator translator = new Translator();

// Add a translation
translator.addTranslation("xml", "json");

// Add another translation
translator.addTranslation("json", “text");

// Translate a message
Message message = new Message("Hello, World!");
Message translatedMessage = translator.translate(message);
```

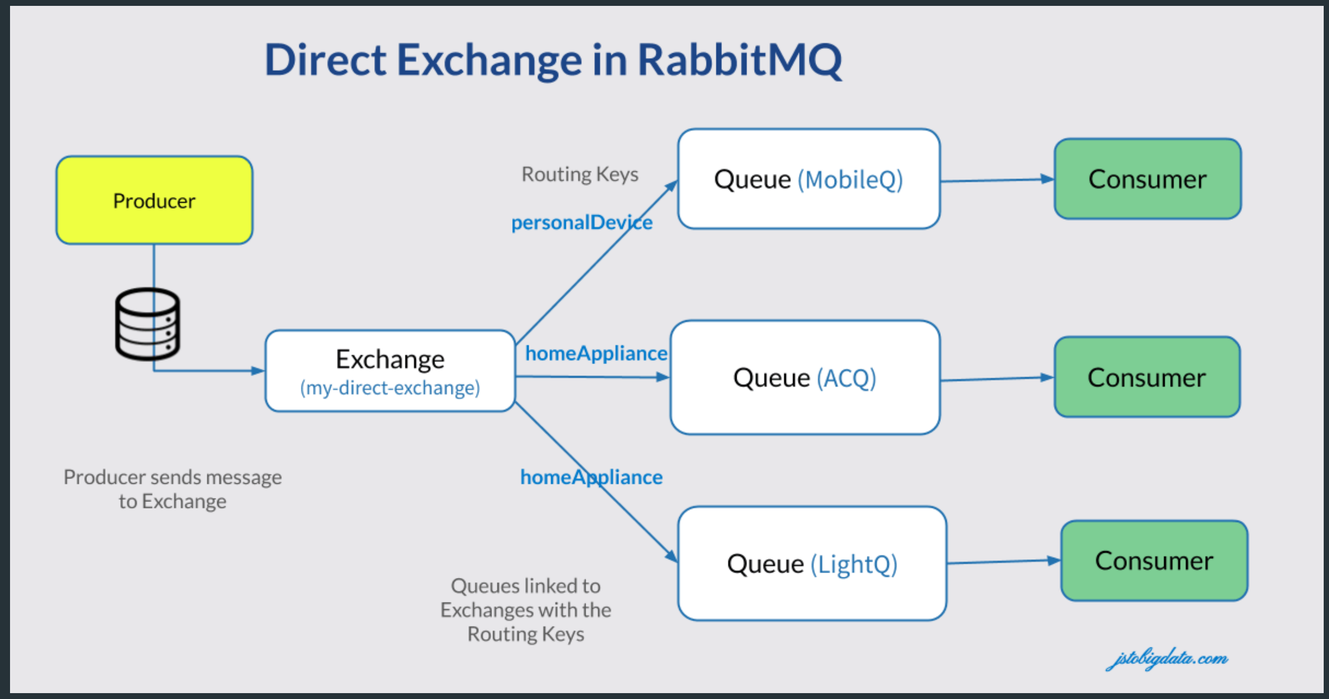
Endpoints

- An endpoint is a software component that sends or receives messages.
- Endpoints can be either producers or consumers.
- Endpoints can be either synchronous or asynchronous.



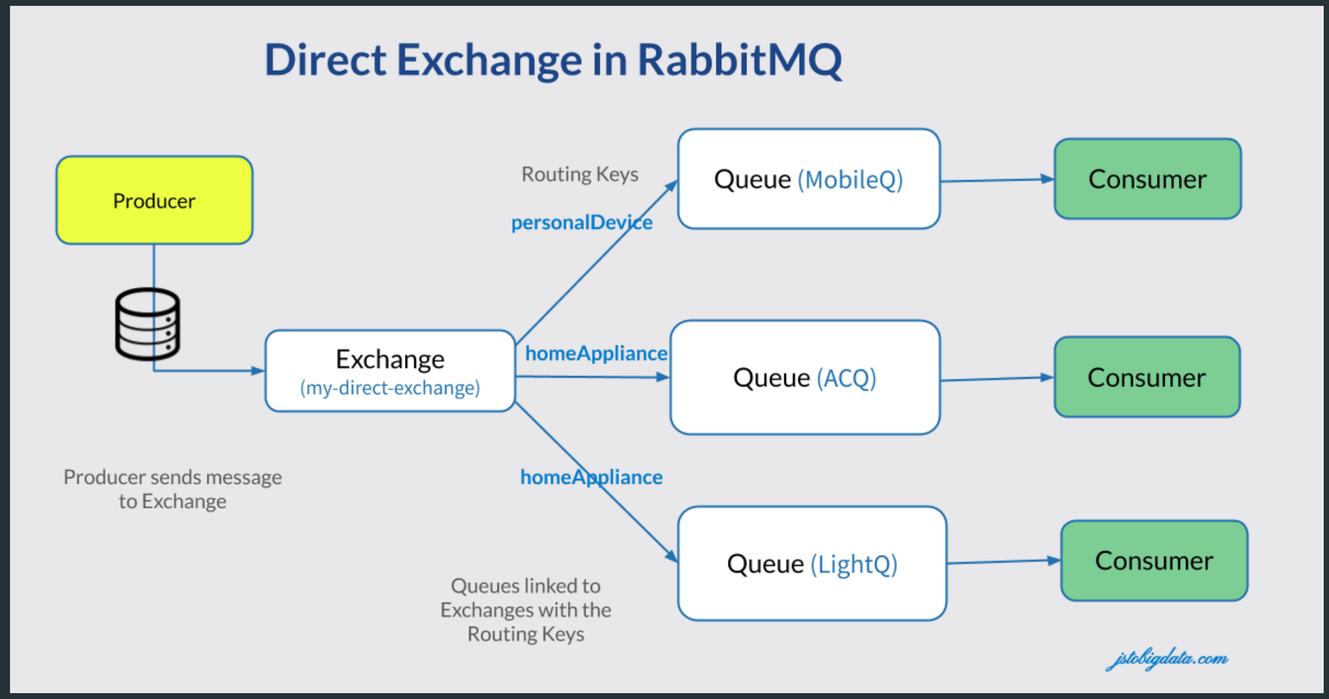
Producer Endpoints

- A producer endpoint is an endpoint that sends messages.
- Producer endpoints can be either point-to-point or publish-subscribe.
- Producer endpoints can be either synchronous or asynchronous.



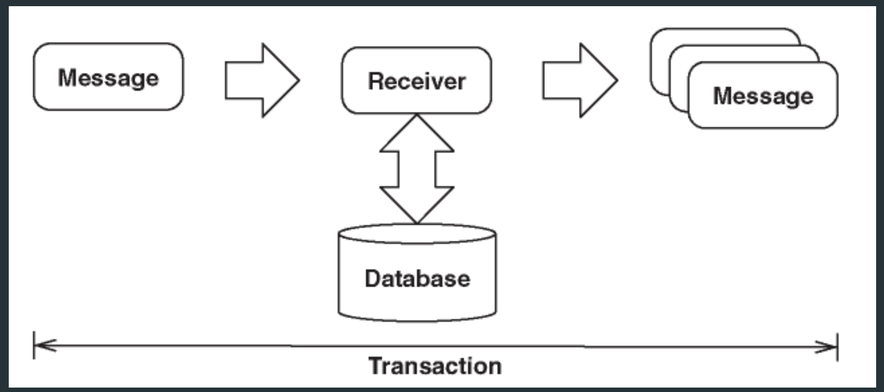
Consumer Endpoints

- A consumer endpoint is an endpoint that receives messages.
- Consumer endpoints can be either point-to-point or publish-subscribe.
- Consumer endpoints can be either synchronous or asynchronous.



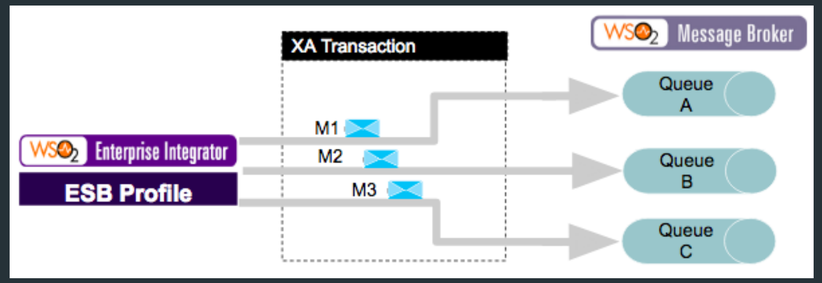
Message Transactions

- A message transaction is a unit of work that is performed on a message.
- Message transactions can be used to ensure that messages are processed correctly.
- Message transactions can be used to roll back changes if a message fails to process.



Types of Message Transactions

- There are two types of message transactions:
  - Local transactions: Local transactions are performed within a single message broker.
  - Distributed transactions: Distributed transactions are performed across multiple message brokers.

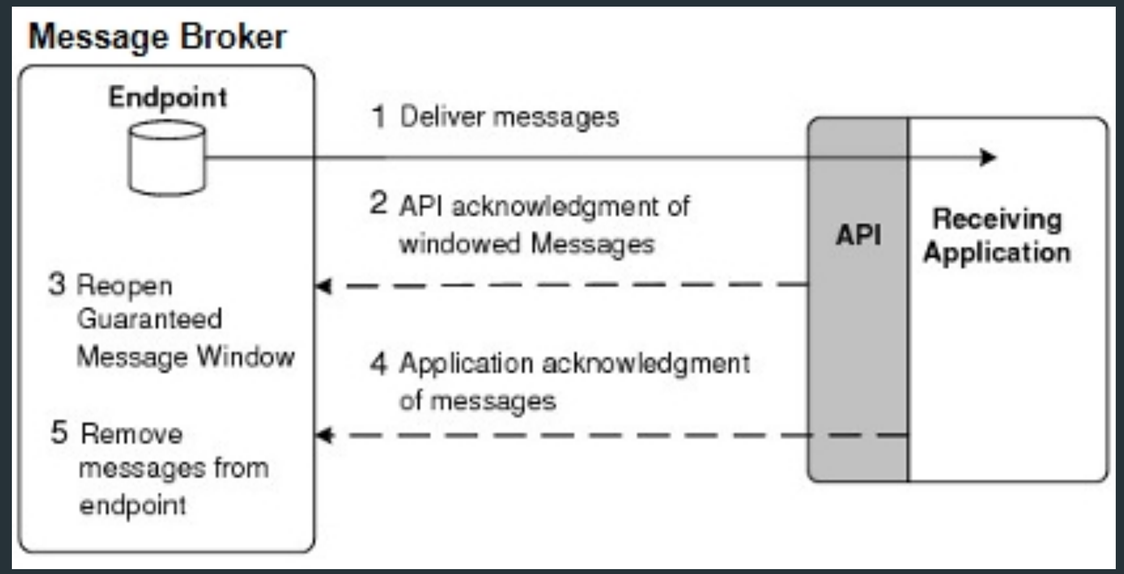


Transaction

```
// Start a transaction
Transaction transaction = messageBroker.startTransaction();

// Process the message
messageProcessor.processMessage(message);

// Commit the transaction
transaction.commit();
```

Message Acknowledgments

- A message acknowledgment is a notification to the sender that the message has been received and processed.
- Message acknowledgments can be used to improve the reliability of message delivery.
- Message acknowledgments can be used to track the status of message delivery.



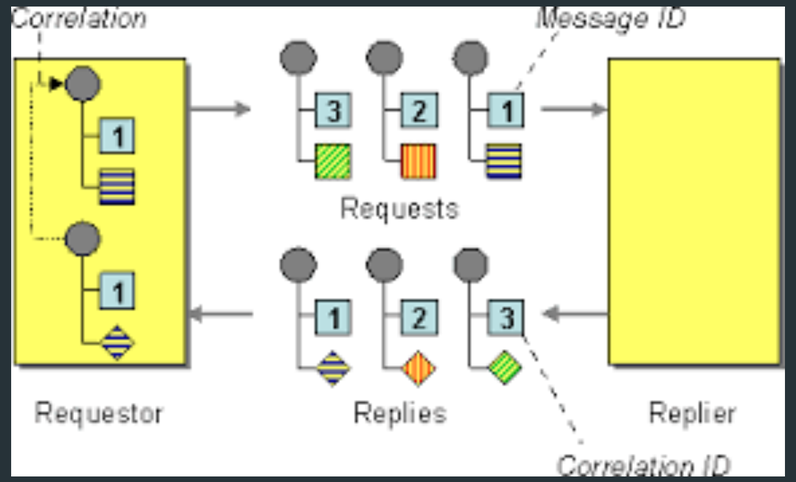
Types of Message Acknowledgments

- There are two types of message acknowledgments:
  - Auto-acknowledgements: Auto-acknowledgements are sent automatically when a message is received.
  - Manual acknowledgments: Manual acknowledgments must be sent manually by the recipient.

```
// Send a message
Message message = new Message("Hello, World!");
messageBroker.send(message);

// Receive a message
Message receivedMessage = messageBroker.receive();

// Acknowledge the message
messageBroker.acknowledge(receivedMessage);
```

Message Correlation

- Message correlation is the process of associating messages with each other.
- Message correlation is used to ensure that messages are processed in the correct order.
- Message correlation is used to group messages together for processing.



Types of Message Correlation

- There are two types of message correlation:
- Implicit correlation: Implicit correlation is the default type of message correlation. In implicit correlation, messages are correlated based on their properties. For example, messages can be correlated based on their destination, their sender, or their content.
- Explicit correlation: Explicit correlation is used when implicit correlation is not sufficient. In explicit correlation, messages are correlated using a correlation ID. The correlation ID is a unique identifier that is assigned to each message.

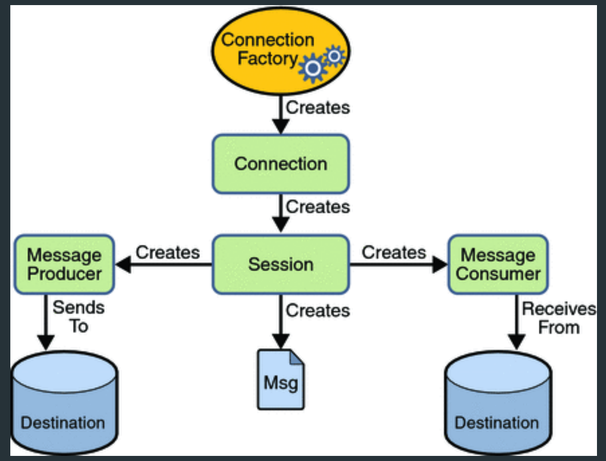
```
// Send a message
Message message = new Message("Hello, World!");
message.setCorrelationId("1234567890");
messageBroker.send(message);

// Receive a message
Message receivedMessage = messageBroker.receive();

// Check the correlation ID
if (receivedMessage.getCorrelationId().equals("1234567890")) {
    // Process the message
} else {
    // Do something else
}
```

The Java Message Service (JMS)

- A messaging standard that allows applications to send and receive messages.
- It provides a way for applications to communicate with each other without having to know about each other's existence.
- It is a loosely coupled messaging system.

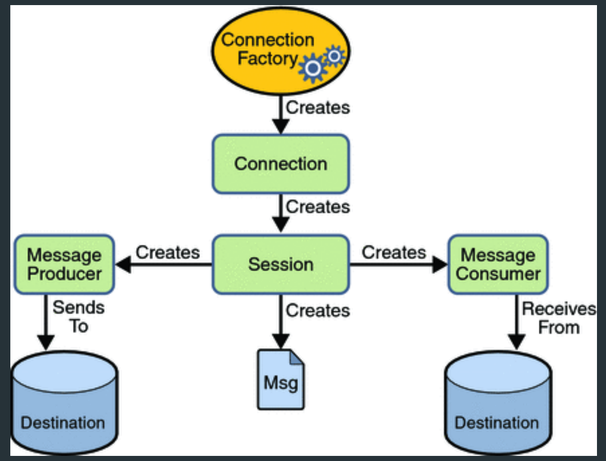
JMS Components

- JMS **Clients**: Applications that send and receive messages.
- JMS **Providers**: Implement the JMS API and provide the infrastructure for sending and receiving messages.
- JMS **Destinations**: Containers for messages.
- JMS **Messages**: Data that is sent and received by JMS clients.



JMS API

- The JMS API provides a set of interfaces and classes that applications use to send and receive messages.
- The JMS API is divided into two main parts:
  - The core API: Provides the basic functionality for sending and receiving messages.
  - The advanced API: Provides additional functionality for more complex messaging scenarios.



Sending Messages with JMS

- To send a message with JMS, you will need to create a JMS producer and specify the destination of the message.
- You can specify the destination of the message as a queue or topic.
- Once you have specified the destination, you can send the message.

```
// Create a JMS producer
JMSProducer producer = new JMSProducer(connectionFactory);

// Specify the destination of the message
Destination destination = new Queue("myQueue");

// Create a message
TextMessage message = new TextMessage();
message.setText("Hello, world!");

// Send the message
producer.send(message, destination);
```

Receiving Messages with JMS

- To receive a message with JMS, you will need to create a JMS consumer and specify the destination of the message.
- You can specify the destination of the message as a queue or topic.
- Once you have specified the destination, you can receive the message.

```
// Create a connection.
Connection connection = connectionFactory.createConnection();
// Create a session.
Session session = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);

// Create a consumer.
Queue destination = new Queue("myQueue");
MessageConsumer consumer = session.createConsumer(destination);

// Start receiving messages.
consumer.setMessageListener(new MyConsumer());

// Start the connection.
connection.start();
```

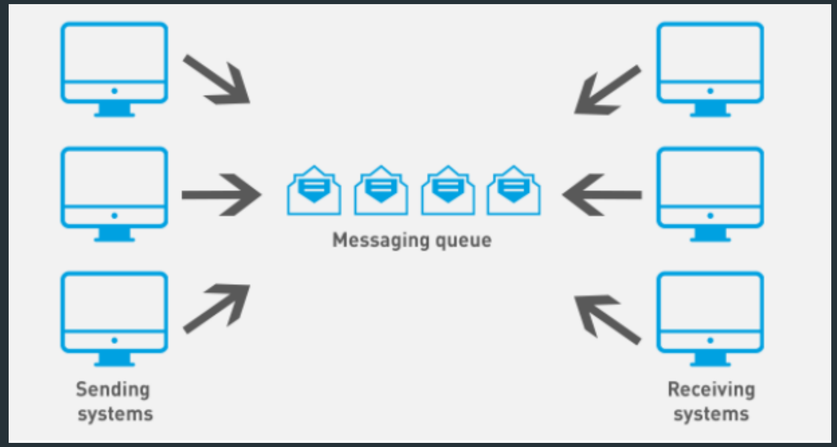
Microsoft Message Queuing (MSMQ)

- A messaging system that enables applications to send and receive messages.
- A scalable and reliable messaging platform that can be used to support a variety of applications.
- A great choice for applications that need to send and receive messages.

```
Payment myPayment;
myPayment.Payor = textBox1.Text;
myPayment.Payee = textBox2.Text;
myPayment.Amount = Convert.ToInt32(textBox3.Text);
myPayment.DueDate = textBox4.Text;

System.Messaging.Message msg = new
System.Messaging.Message();
msg.Body=myPayment;

MessageQueue msgQ =new MessageQueue(".\\Private$\\billpay");
msgQ.Send(msg);
```

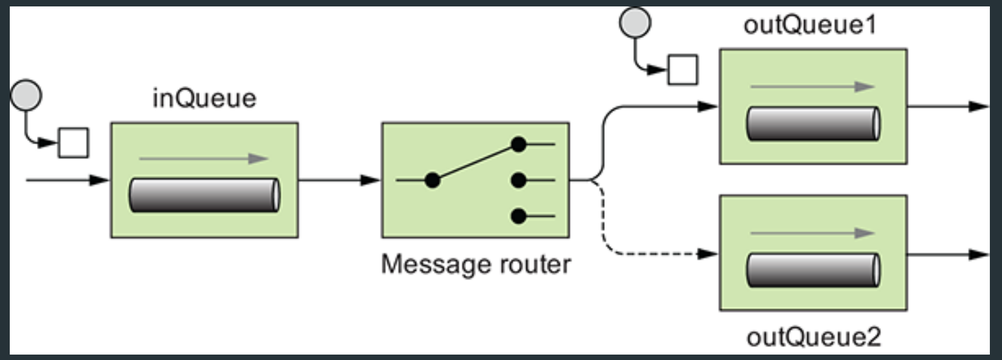
Features of MOM

- MOM platforms typically offer a number of features, including:
  - Message queuing.



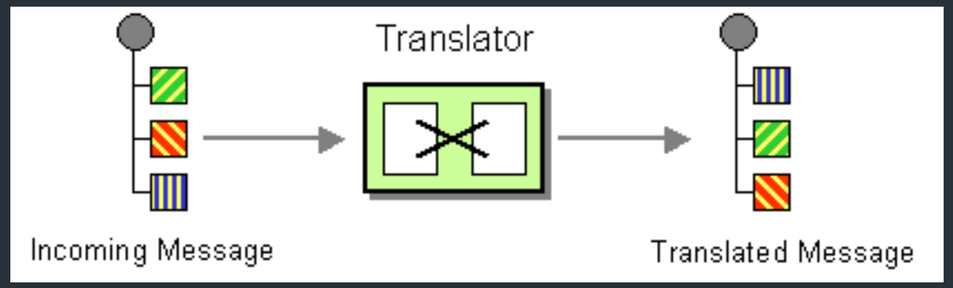
Features of MOM

- MOM platforms typically offer a number of features, including:
  - Message queuing.
  - Message routing.



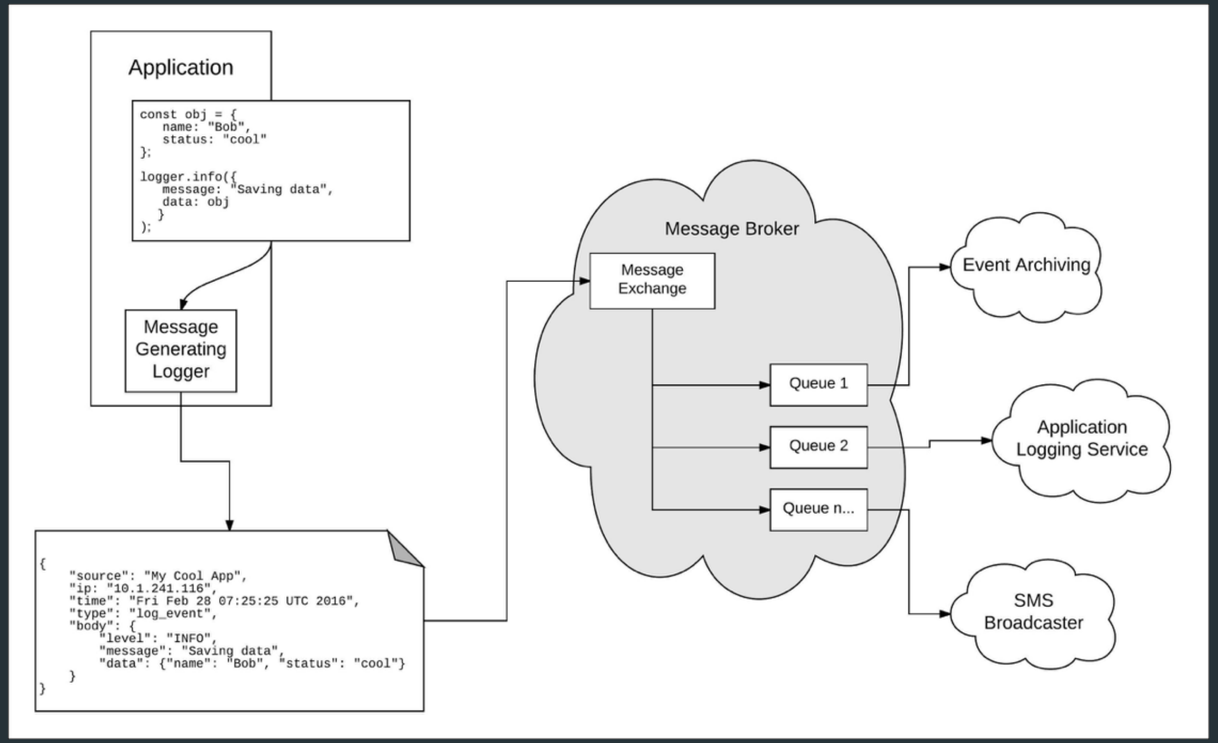
Features of MOM

- MOM platforms typically offer a number of features, including:
  - Message queuing.
  - Message routing.
  - Message transformation.



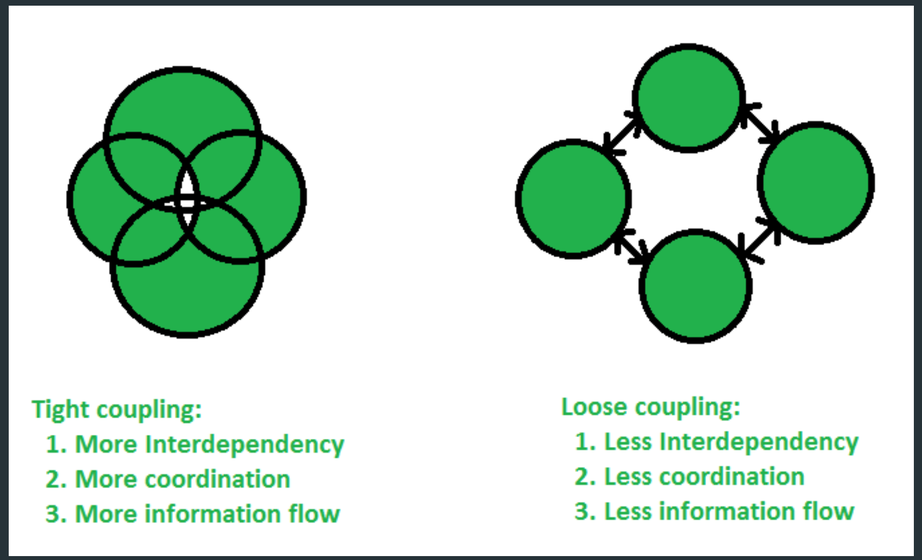
Features of MOM

- MOM platforms typically offer a number of features, including:
  - Message queuing.
  - Message routing.
  - Message transformation.
  - Message logging.



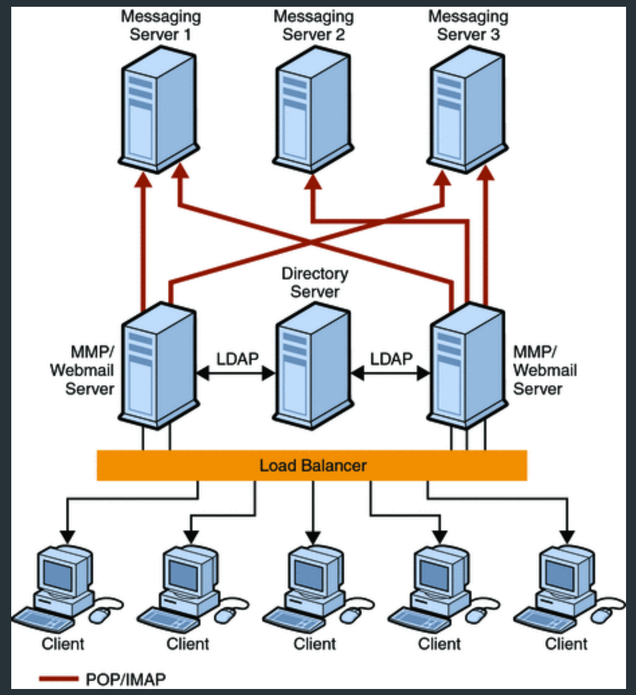
Benefits of MOM

- MOM provides a number of benefits, including:
  - Loose coupling.



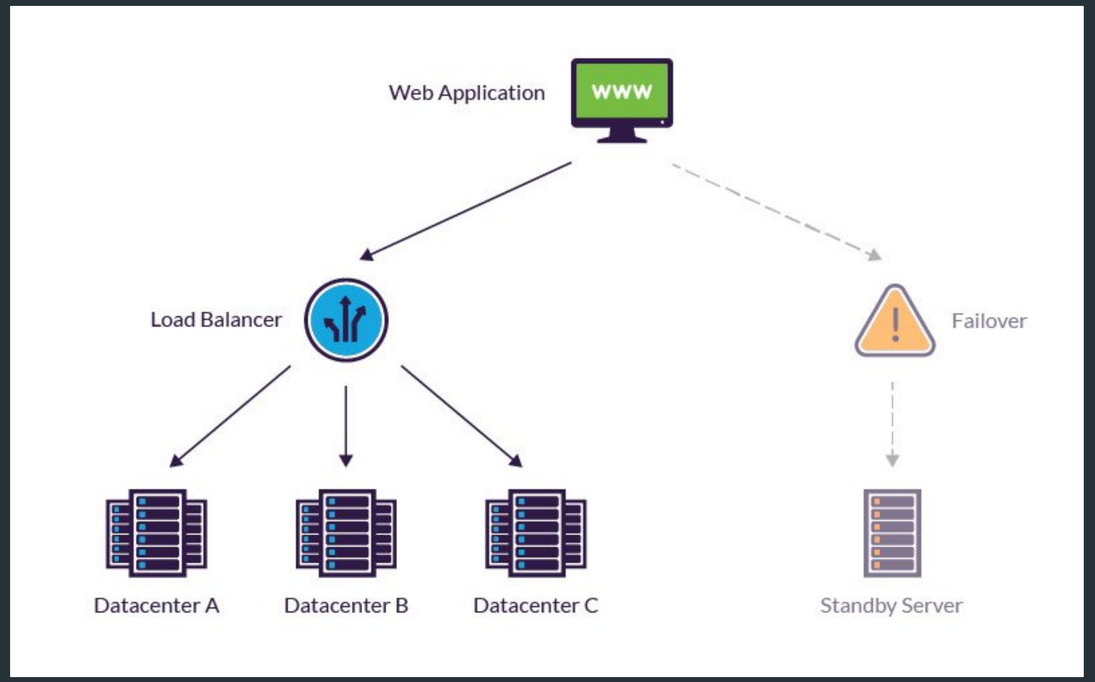
Benefits of MOM

- MOM provides a number of benefits, including:
  - Loose coupling.
  - Scalability.



Benefits of MOM

- MOM provides a number of benefits, including:
  - Loose coupling.
  - Scalability.
  - Reliability.

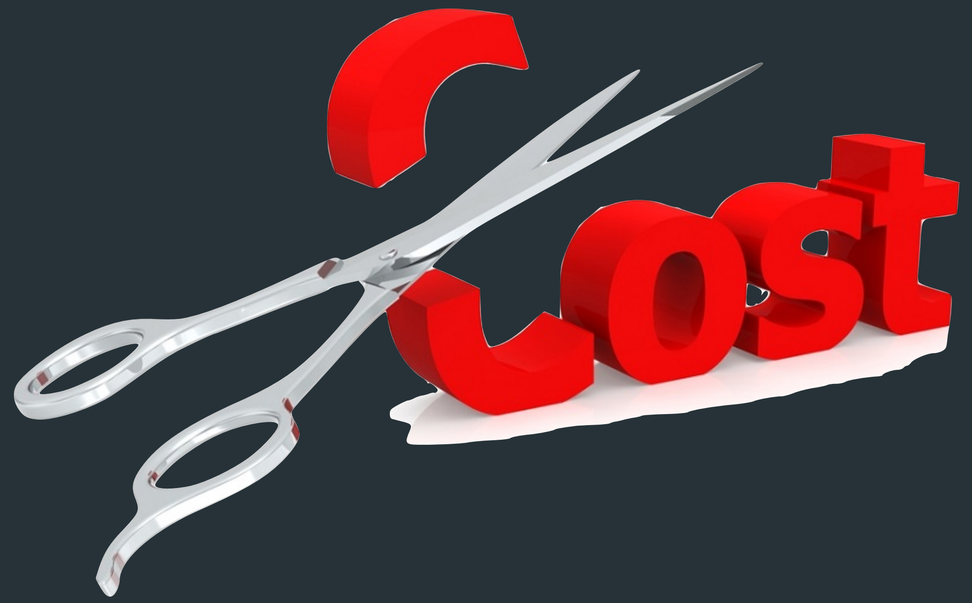


Benefits of MOM

- MOM provides a number of benefits, including:
  - Loose coupling.
  - Scalability.
  - Reliability.
  - Improved performance.

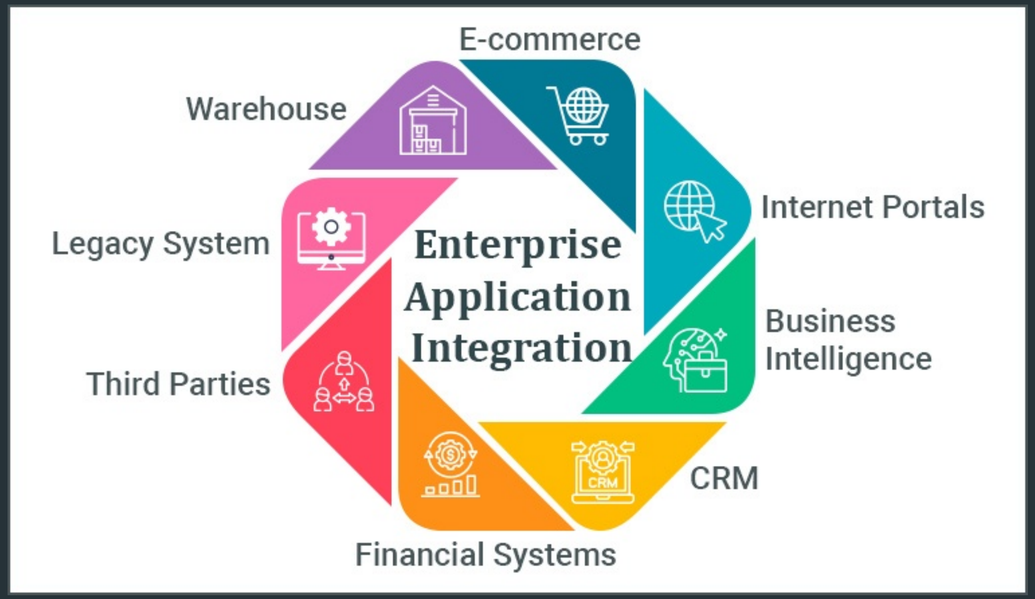
Benefits of MOM

- MOM provides a number of benefits, including:
  - Loose coupling.
  - Scalability.
  - Reliability.
  - Improved performance.
  - Reduced costs.



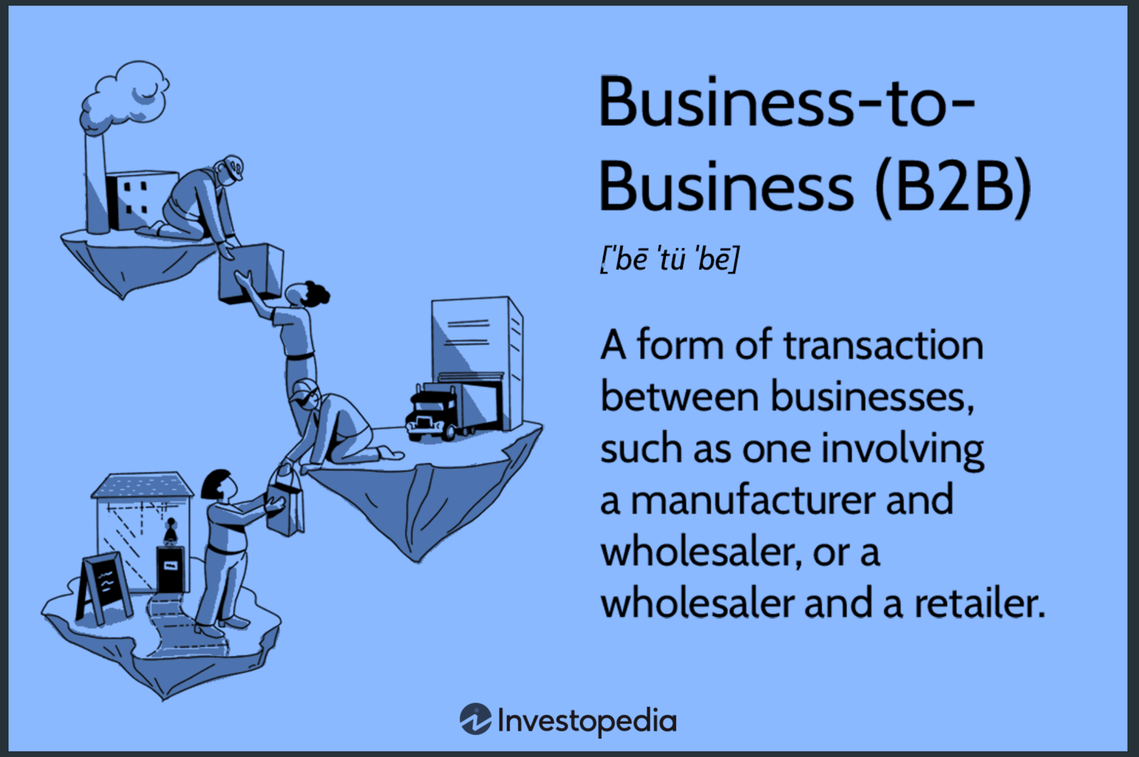
Uses of MOM

- MOM can be used in a variety of applications, including:
  - Enterprise application integration (EAI).



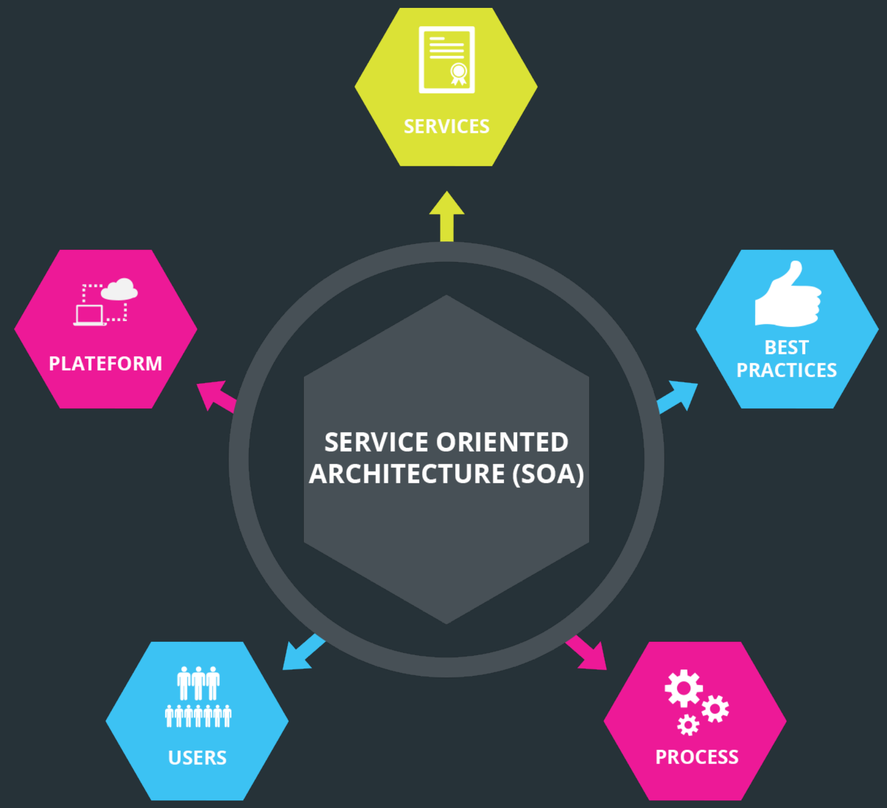
Uses of MOM

- MOM can be used in a variety of applications, including:
  - Enterprise application integration (EAI).
  - Business-to-business (B2B) integration.



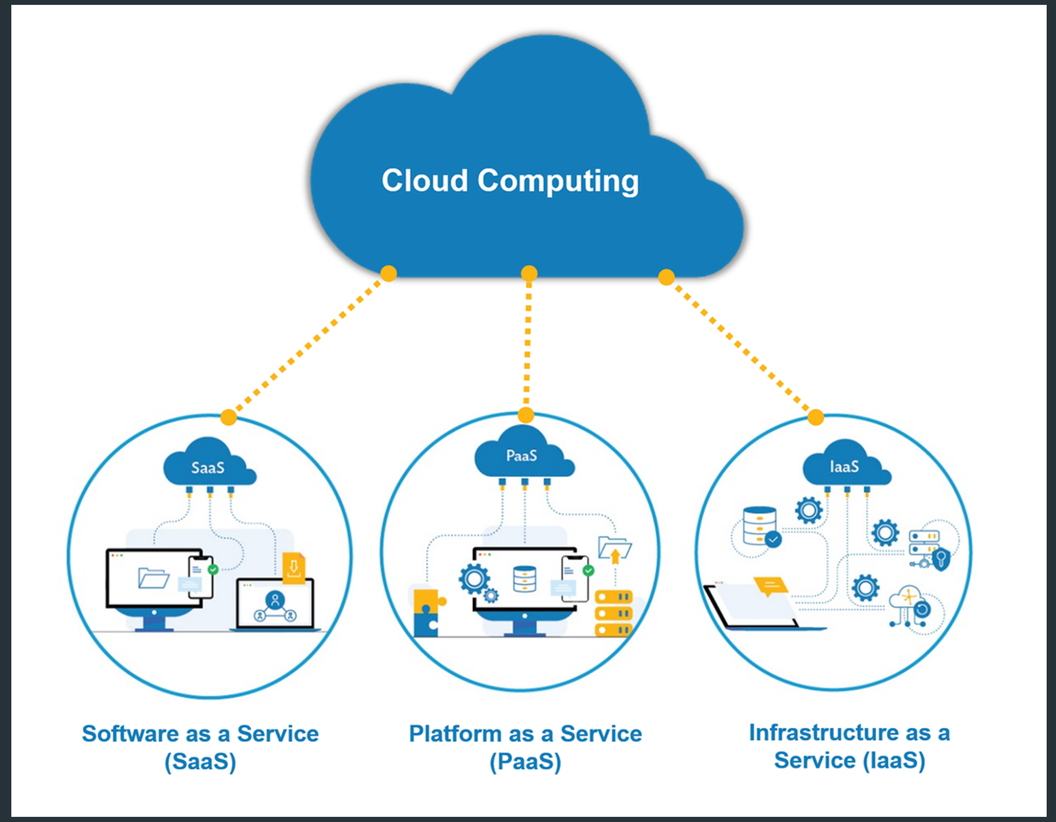
Uses of MOM

- MOM can be used in a variety of applications, including:
  - Enterprise application integration (EAI).
  - Business-to-business (B2B) integration.
  - Service-oriented architecture (SOA).



Uses of MOM

- MOM can be used in a variety of applications, including:
  - Enterprise application integration (EAI).
  - Business-to-business (B2B) integration.
  - Service-oriented architecture (SOA).
  - Cloud computing.



Popular MOM Platforms

- Some of the most popular MOM platforms include:
  - IBM WebSphere Message Broker
  - Microsoft BizTalk Server
  - Oracle Enterprise Service Bus
  - Tibco ActiveEnterprise
  - Red Hat JBoss Enterprise Service Bus

Choosing a MOM Platform

- When choosing a MOM platform, you should consider the following factors:
  - Your specific integration needs.
  - The size and complexity of your organization.
  - Your budget.
  - Your technical expertise.

IBM WebSphere Message Broker

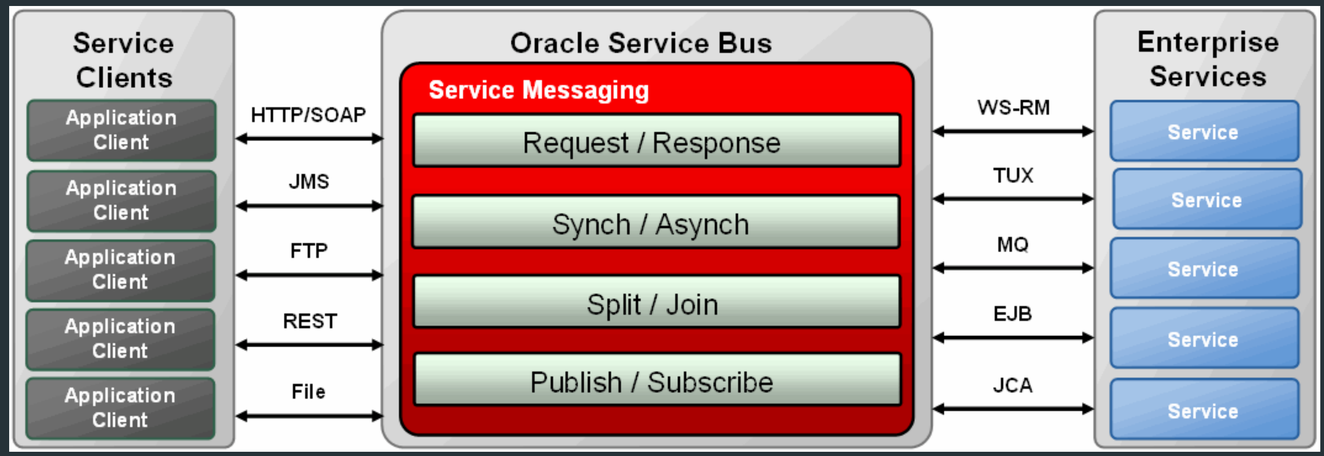
- IBM WebSphere Message Broker is a powerful MOM platform that can be used to integrate a wide variety of applications. It is a good choice for organizations that need a scalable and reliable platform for integration.
- Some of the key features of IBM WebSphere Message Broker include:
  - A powerful message broker engine.
  - A wide range of integration capabilities.
  - A scalable and reliable architecture.
  - A comprehensive set of security features.

Microsoft BizTalk Server

- Microsoft BizTalk Server is a MOM platform that is designed to integrate applications developed using Microsoft technologies. It is a good choice for organizations that are already using Microsoft technologies.
- Some of the key features of Microsoft BizTalk Server include:
  - A powerful message broker engine.
  - A wide range of integration capabilities.
  - A scalable and reliable architecture.
  - A comprehensive set of security features.
  - Integration with other Microsoft products.

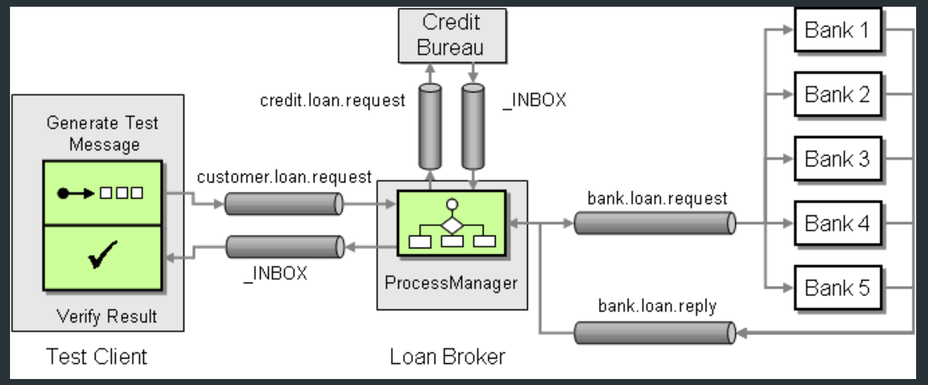
Oracle Enterprise Service Bus

- Oracle Enterprise Service Bus is a MOM platform that is designed to integrate applications developed using Oracle technologies. It is a good choice for organizations that are already using Oracle technologies.
- Some of the key features of Oracle Enterprise Service Bus include:
  - A powerful message broker engine.
  - A wide range of integration capabilities.
  - A scalable and reliable architecture.
  - A comprehensive set of security features.
  - Integration with other Oracle products.



Tibco ActiveEnterprise

- Tibco (The Information Bus Company) ActiveEnterprise is a MOM platform that is designed to integrate applications developed using a variety of technologies. It is a good choice for organizations that need a flexible and scalable platform for integration.
- Some of the key features of Tibco ActiveEnterprise include:
  - A powerful message broker engine.
  - A wide range of integration capabilities.
  - A scalable and reliable architecture.
  - A comprehensive set of security features.
  - Integration with a wide range of technologies.



Red Hat JBoss Enterprise Service Bus

- Red Hat JBoss Enterprise Service Bus is a MOM platform that is designed to integrate applications developed using open source technologies. It is a good choice for organizations that are looking for a cost-effective platform for integration.
- Some of the key features of Red Hat JBoss Enterprise Service Bus include:
  - A powerful message broker engine.
  - A wide range of integration capabilities.
  - A scalable and reliable architecture.
  - A comprehensive set of security features.
  - Integration with a wide range of open source technologies.

Comparison

Platform	Strengths	Weaknesses
IBM WebSphere Message Broker	Powerful message broker engine, wide range of integration capabilities, scalable and reliable architecture, comprehensive set of security features	Expensive, complex to use
Microsoft BizTalk Server	Powerful message broker engine, wide range of integration capabilities, scalable and reliable architecture, comprehensive set of security features, integration with other Microsoft products	Expensive, complex to use
Oracle Enterprise Service Bus	Powerful message broker engine, wide range of integration capabilities, scalable and reliable architecture, comprehensive set of security features, integration with other Oracle products	Expensive, complex to use
Tibco ActiveEnterprise	Flexible and scalable platform for integration, integration with a wide range of technologies	Expensive, complex to use
Red Hat JBoss Enterprise Service Bus	Cost-effective platform for integration, integration with a wide range of open source technologies	Not as powerful as some of the other platforms, not as scalable as some of the other platforms

Lecture outcomes

- MOM
  - Fundamentals
  - Options
- JMS
- MSMQ