**Middleware**

Middleware is the software that assists an application to interact or communicate with other applications, networks, hardware, and/or operating systems.

**Message Oriented Middleware**

Message Oriented Middleware (or “MOM”) is one particular form of middleware, which is capable of facilitating the transportation of asynchronous messages from one component to another.

**MOM implementations:**

* JMS providers (eg. ActiveMQ ,OpenJMS)
* AMQP providers (eg. RabbitMQ,Apache Qpid)

Many more..

* MSMQ (Microsoft’s implementation of MOM)
* MQSeries(IBM’s implementation of MOM)

**Use case:**

<To Do>

**Advantages:**

* Supports Asynchronous messaging
* Supports message broadcast
* Decouples Sender and receiver
* Many more like support for transaction, priority based message processing etc.

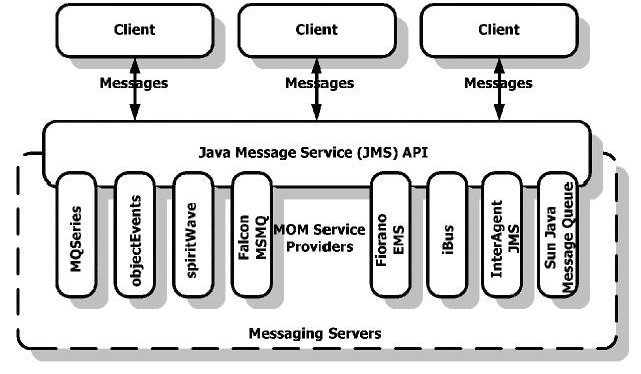
**Messaging prior to JMS:**

* There were multiple vendors proving messaging solutions.
* Each have their own api. Users code is tightly coupled to existing vendor’s api.
* So switching from one vendor to other was painful.
* In worst case user may have to start coding from scratch.

**JMS:**

* Java Message Service is a part of Sun's J2EE suite,
* which solves the previous problem by providing a standard set of APIs that developers can use to access the features of any underlying messaging system .
* It solves vendor lock in problem
* Switching from one JMS vendor to other is easy(only some configuration changes are required).

Most of the existing code remains intact.



**JMS Architecture:**

Messaging system has three entities:

1. Message Producer

-produces messages

2. Message Consumer

-consumes messages produced by Message Producer

3. Message Oriented middleware (broker)

-Enables Messages consumers consume messages produced by desired message producers.

-MOM can have multiple Destinations.

-Destinations are of two types

1. **Queue**-Used for point to point messaging
2. **Topic**-Used to send same message to multiple consumers(broadcast)

Message Producer sends message to a particular Destination(Topic/Queue),Which is finally consumed by one or more message consumers.

Message producer and consumer need not be aware of each other. The only need to know the Destination to talk to. This way MOM supports loose coupling.

**JMS Messaging Mode:**

JMS supports two types of messaging:

1. Point-to-Point Messaging:

-Asynchronous transfer of message from one producer to one consumer.

-Message Producer sends message to a **Queue.**

**-**Message consumers collects its message from Queue when available.

2.PubSub (One-to-Many) Messaging:

-Here all the Message consumers needs to subscribe Topic(Destination) for messages.

-When a new message arrives at Topic,it is broadcasted to all the subscribed Consumers.

**Problems with JMS:**

Since JMS is a java API, it can only be used from java application.

Advanced Message Queueing Protocol (AMQP):

* AMQP is a binary wire protocol which was designed for interoperability between different vendors.
* Information is organized into “frames”
* Each frame consists of header (type, channel id, payload size), payload, frame end packet.
* Frames can be protocol methods (commands),structured content (message headers), data.
* Since AMQP is just a protocol, the logic for producing, sending and receiving messages.So it can be written in any language ,we just need to create AMQP frames and sent them through channel.
* RabbitMQ is a MOM which has both server and client implementations of AMQP. Currently it has client api for following languages: java,python,javascript,ruby,php,.NET,pearl,C/C++,Erlang,Go etc.



**Message Routing:**

* AMQP handles message routing differently compared to JMS.
* The key components in the AMQP architecture are **exchanges**, **queues**, and **bindings**.
* You bind **queues** to **exchanges** based on **binding rules**.
* **Message Producers** send message to **exchanges**
* Based on the message **routing key** and the **exchange type**, the broker will decide to which **queue** it has to deliver the message.
* Then **Message Consumer** collects messages from **queue**.

**Exchanges:**

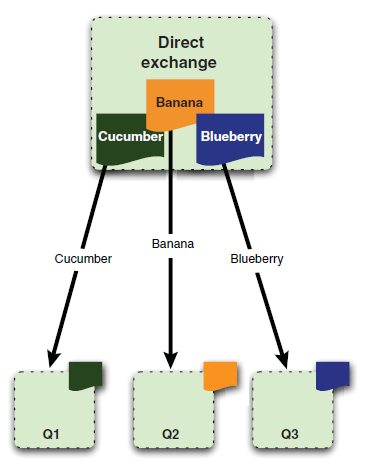
There are four:

**direct, fanout, topic, and headers**. Each implements a different routing algorithm.

**Direct**: if the routing key matches, then the message

is delivered to the corresponding queue. The broker must implement the direct exchange, including a default exchange with an empty string as its name. When a queue is declared, it’ll be automatically

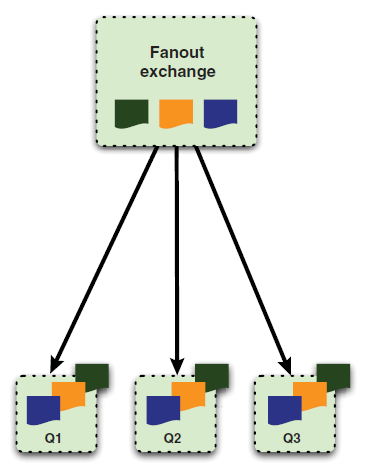
bound to that exchange using the queue name as routing key.



**Fanout**: this exchange will multicast the received message to the bound

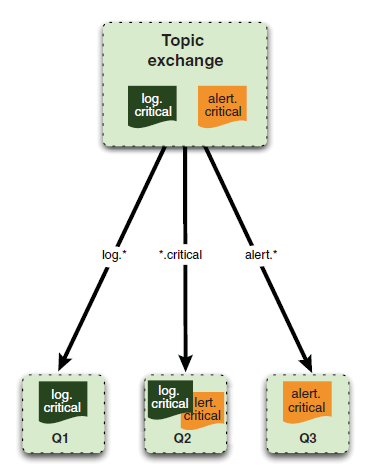
queues. when you send a message to a fanout exchange, it’ll be delivered to all the queues attached

to this exchange.



**Topic**: This exchange allows you to achieve interesting messaging scenarios, where messages can arrive to the same

queue coming from different sources



**Headers**.: allows you to match against a header in the AMQP message

instead of the routing key. Other than that, it operates identically to the direct

exchange but with much worse performance.( As a result, it doesn’t provide much

real-world benefit and is almost never used.)

Here are a couple of AMQP examples and links:

* It is used in one of the world’s largest biometric databases [India’s Aadhar project](http://blogs.vmware.com/vfabric/2012/07/spring-and-rabbitmq-behind-indias-12-billion-person-biometric-database-1.html)—home to 1.2 billion identities.
* It is used in the [Ocean Observatories Initiative](http://blogs.vmware.com/vfabric/2012/09/how-even-the-ocean-data-is-in-the-cloud.html" \o "Ocean Observatories Initiative" \t "_blank)—an architecture that collects 8 terabytes of data per day.