Understanding

The Principles of Messaging Systems

— Nandakumar Purohit

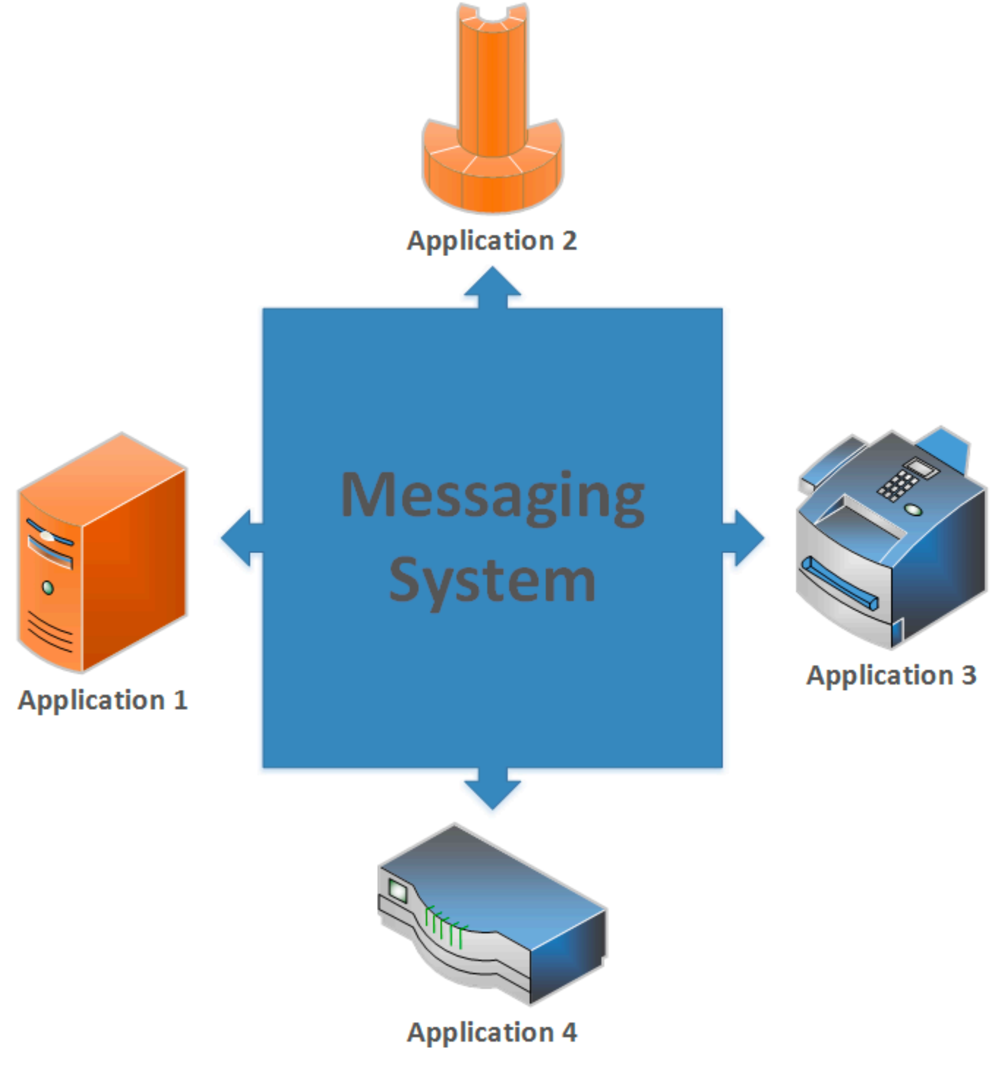
The Principles of Messaging Systems

What will you learn?

- Understanding Messaging Systems
- Peeking into P2P Messaging System
- Quick glimpse on Pub-Sub Messaging System

The Principles of Messaging Systems

Understanding Messaging Systems



"A messaging system acts as an integration component between multiple applications. Such an integration invokes different application behaviors based on application information exchanges."

The Principles of Messaging Systems

Understanding Messaging Systems

Messaging Concepts	
Message Queues	• Also referred as **channels**. • Their core function is to **receive message packets** from the source application and send it to the receiver application in a timely and reliable manner.
Messages (Data Packets)	• **A message is an atomic data packet** that gets transmitted over a network to a message queue. • The sender application breaks data into **smaller data packets** and wraps it as a message with protocol and header information. It then sends it to the message queue. • Receiver receives it and extracts data.
Sender (Producer)	• They establish **connections** to message queue endpoints and send data in smaller message packets **adhering to common interface standards**. • Depending on the **type of messaging system in use**, sender applications can decide to send data one by one or in a batch.

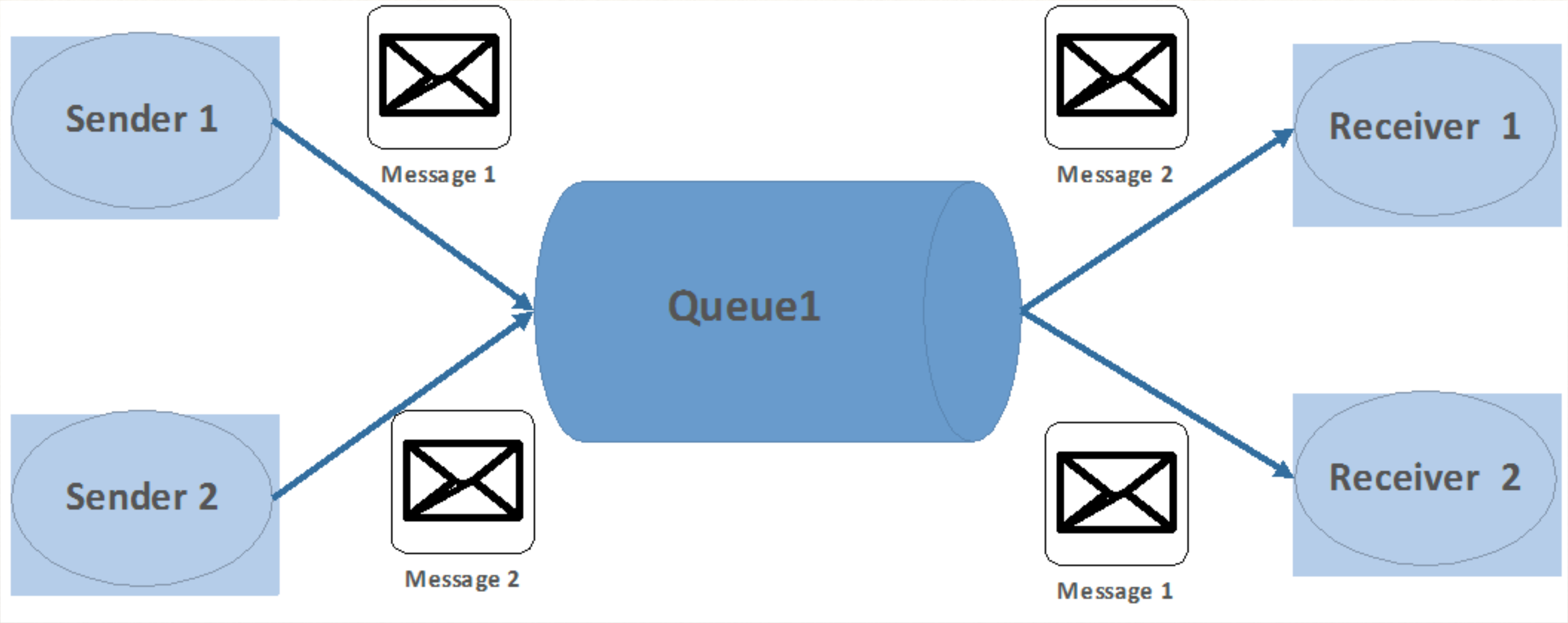
The Principles of Messaging Systems

Understanding Messaging Systems

Messaging Concepts	
Receiver (Consumer)	• Receivers either pull data from message queues or they receive data from messages queues through a **persistent connection**. • On receiving messages, they extract data
Data Transmission Protocols	• Data transmission protocols **determine rules to govern message exchanges** between applications. • **Different queuing** systems use **different** data transmission **protocols**. • It depends on the technical implementation of the messaging endpoints. • **Kafka** uses **binary protocols** over TCP. • The client initiates a **socket connection with Kafka** queues and then writes messages along with reading back the acknowledgment message. • Some examples of such **data transmission protocols are:** • **AMQP** (Advance Message Queuing Protocol), • **STOMP** (Streaming Text Oriented Message Protocol), • **MQTT** (Message Queue Telemetry Protocol) • HTTP (Hypertext Transfer Protocol)
Transfer Mode	• Synchronous • Asynchronous • Batch modes.

The Principles of Messaging Systems

Peeking into P2P System



- Messages are **sorted** in the order in which they were **received** and as they are consumed, they are **removed** from the head of the queue.
- **Queues** such as **Kafka** maintain **message offsets**.
- Instead of deleting the messages, they **increment the offsets** for the receiver.
- Offset-based models provide **better support** for **replaying messages**.

The Principles of Messaging Systems

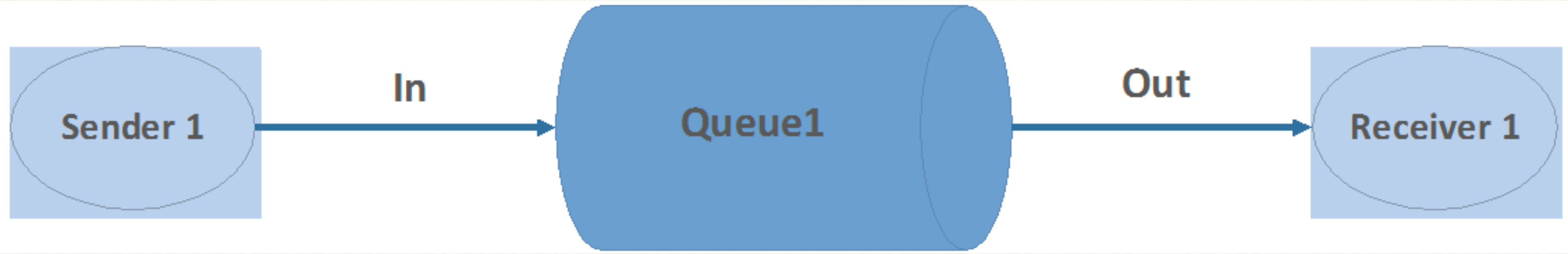
Peeking into P2P System

Messaging Concepts	
Many Senders to One Queue	• **More than one sender** can produce and send messages **to a queue**. • Senders can **share a connection** or use **different connections**, but they can all access the **same queue**.
Many Receivers to One Queue	• **More than one receiver** can consume messages **from a queue**, but **each message** can be consumed by **only one receiver**. • Thus, **Message 1**, **Message 2**, and **Message 3** are consumed by **different receivers**.
Message Queue Extension	• Receivers can **share a connection** or use **different connections**, but they can all access the **same queue**.
No Timing Dependency	• Senders and receivers have no timing dependencies • The receiver can consume a message whether or not it was running when the sender produced and sent the message.
Order of Message Consuming	• Messages are placed in a queue in the order they are produced • The order in which they are consumed depends on factors such as: • **Message expiration date** • **Message priority** • **Message Selectors** • The relative message **processing rate** of the consumers.

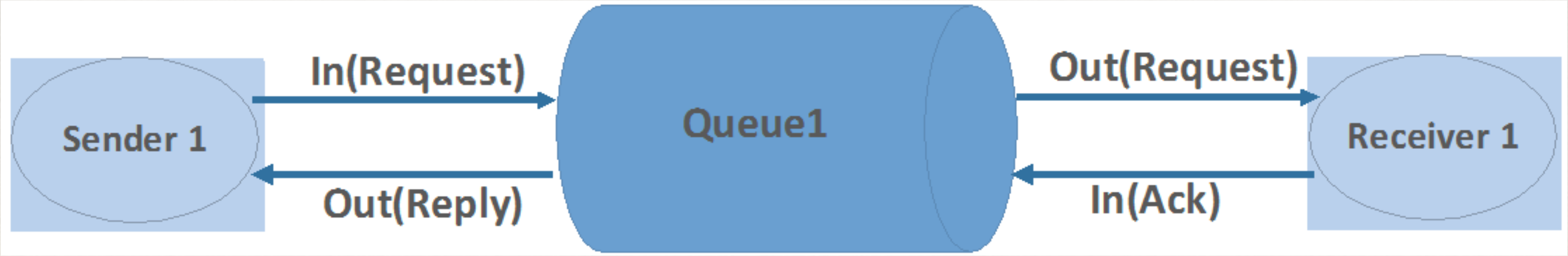
The Principles of Messaging Systems

P2P Models

Fire-and-Forget Model

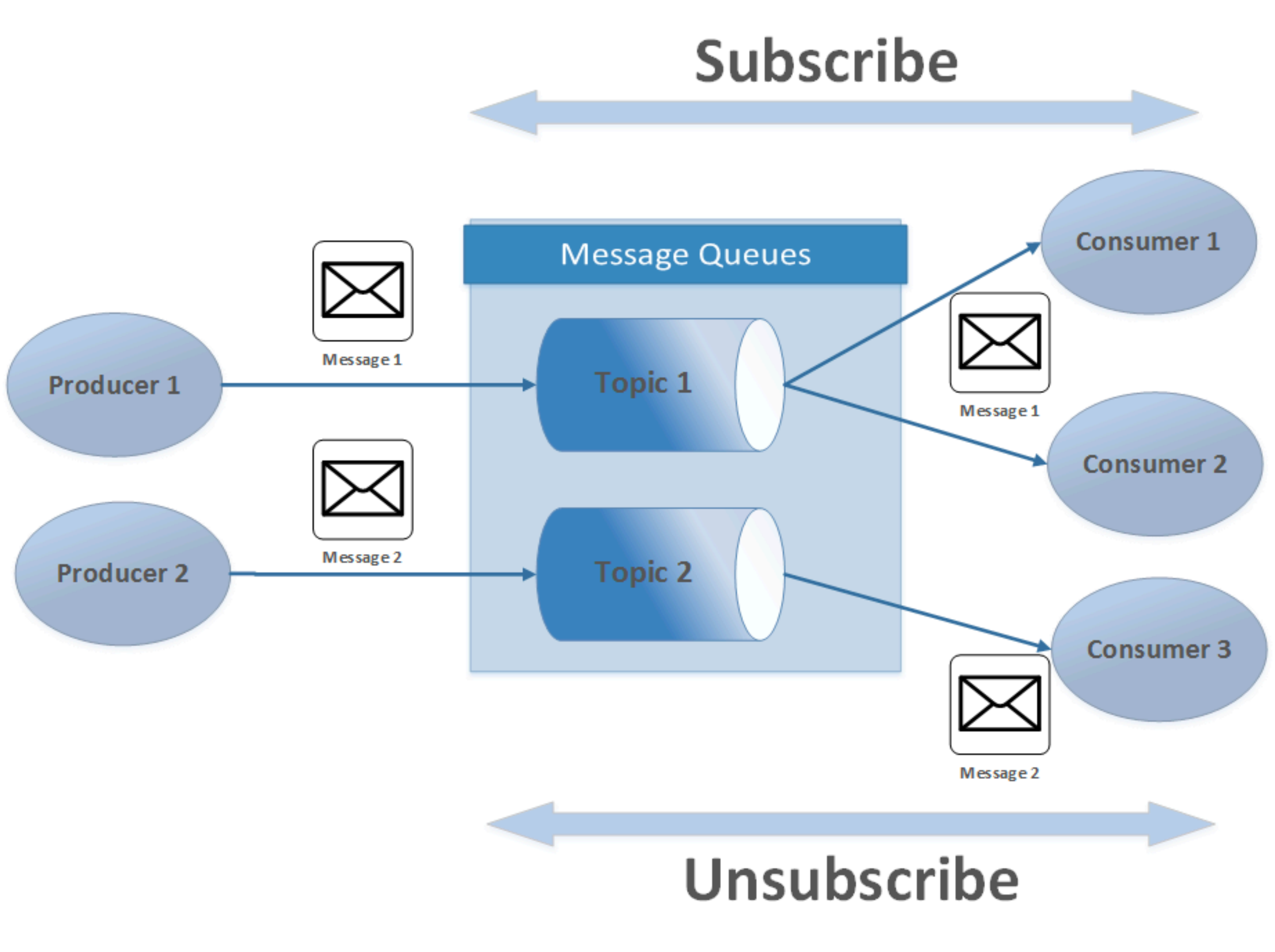


Request/Reply Model



The Principles of Messaging Systems

Publish-Subscribe Messaging System



The Principles of Messaging Systems

Peeking into P2P System

Messaging Concepts	
Topic	• Messages are shared through a channel called atopic. • A topic is a centralized place where producers can publish, and subscribers can consume, messages.
Subscribers / Consumers	Each message is delivered to one or more message consumers, called subscribers.
Publisher	The publisher generally does not know and is not aware of which subscribers are receiving the topic messages.
Asynchronous	Messages are pushed to consumers, which means that messages are delivered to consumers without having to request them.
No coupling	• There is no coupling of the producers to the consumers. • Subscribers and publishers can be added dynamically at runtime, which allows the system to grow or shrink in complexity over time.

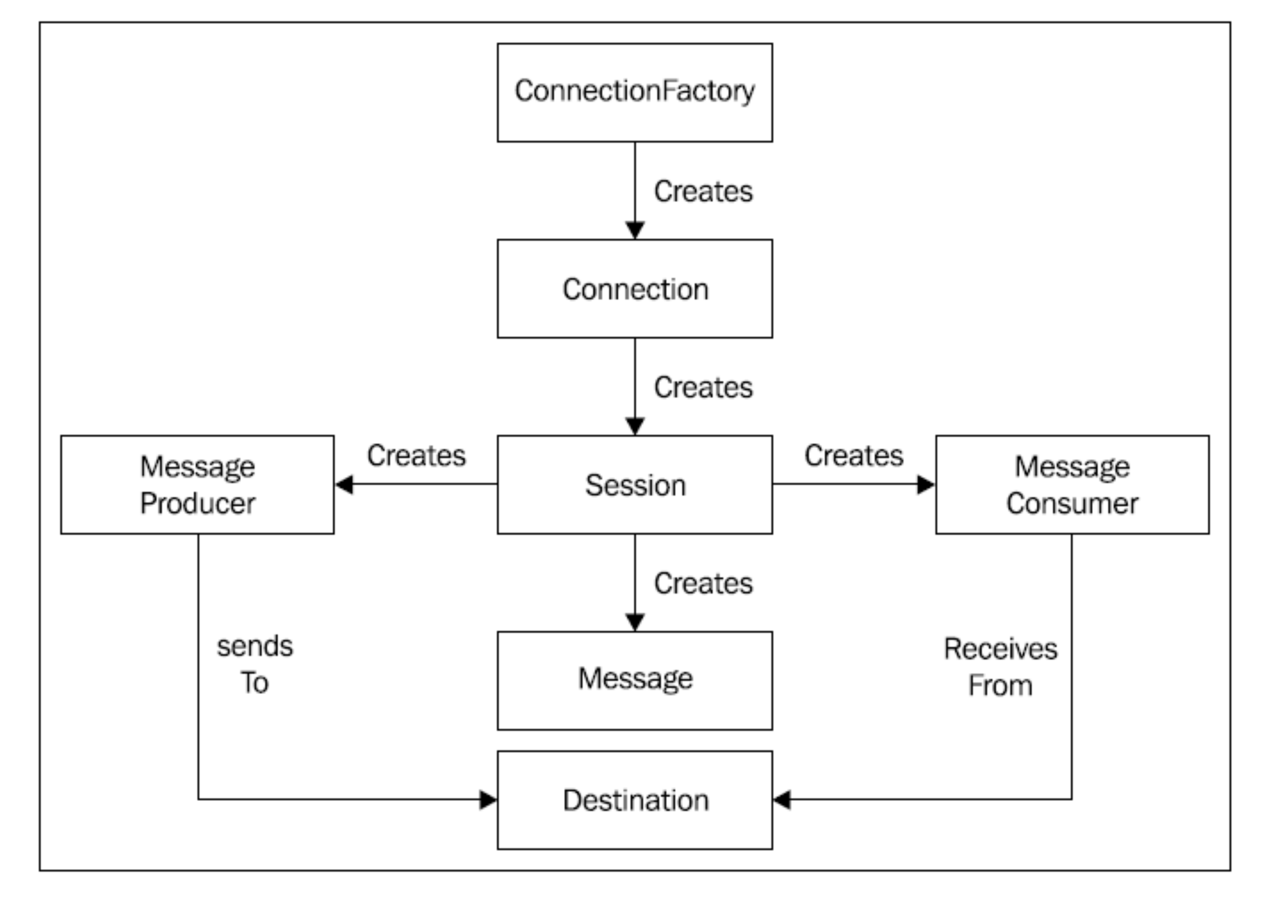
Java Messaging Service

What is JMS API?

- JMS API is a Java API which contains a **common set of interfaces**
- JMS API is used to **create, send, receive and read messages or exchange messages**
- Portable to any JMS Provider.

Java Messaging Service

JMS Architecture



Java Messaging Service

JMS Providers

1	Apache Kafka	Apache / LinkedIn (originally)
2	WebSphere MQ	IBM
3	Weblogic Messaging	Oracle Corporation
4	Active MQ	Apache Foundation
5	Rabbit MQ	Rabbit Technologies(acquired by Spring Source)
6	HornetQ	JBoss
7	Sonic MQ	Progress Software
8	TIBCO EMS	TIBCO
9	Open MQ	Oracle Corporation
10	SonicMQ	Aurea Software

Java Messaging Service

JMS Version History

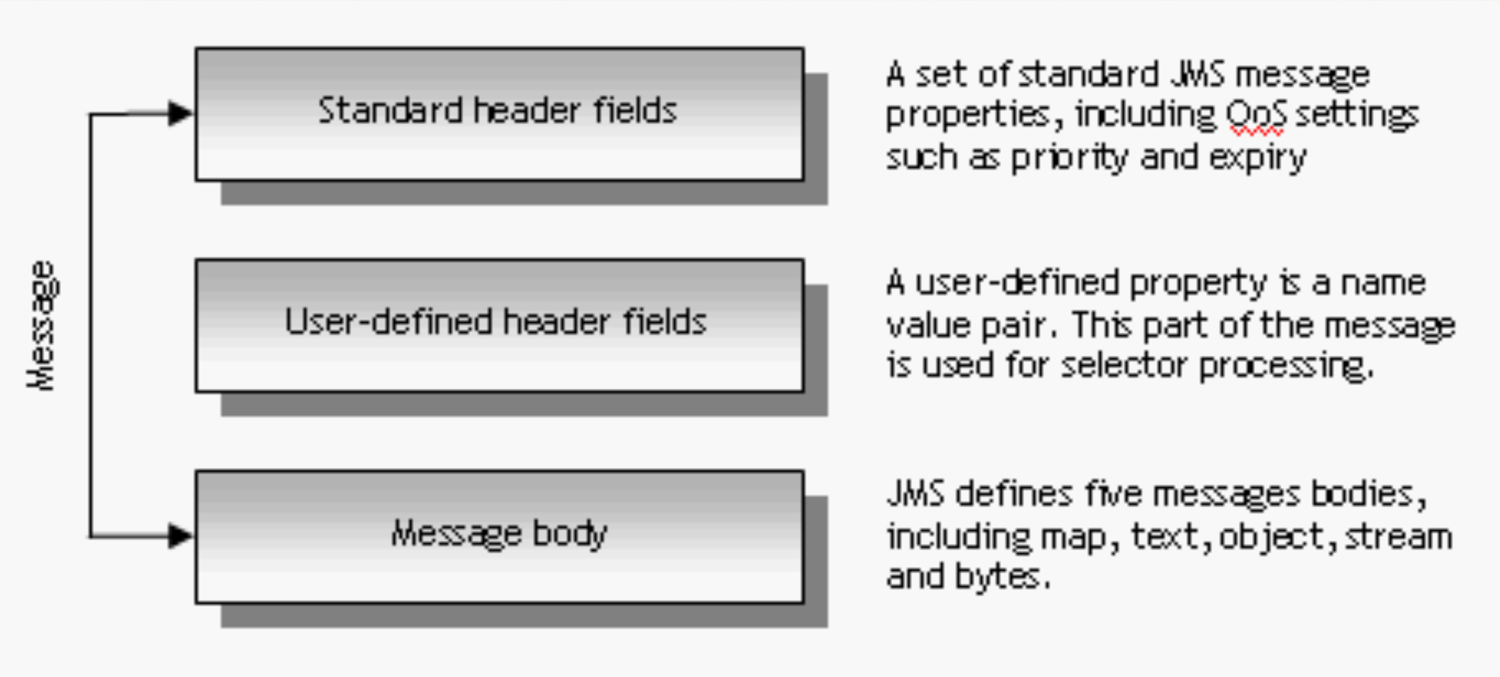
Version history

- JMS 1.0.1 (October 5, 1998)
- JMS 1.0.1a (October 30, 1998)
- JMS 1.0.2 (December 17, 1999)
- JMS 1.0.2a (December 23, 1999)
- JMS 1.0.2b (August 27, 2001)
- JMS 1.1 (April 12, 2002)
- JMS 2.0 (May 21, 2013)
- **JMS 2.0a (March 16, 2015)**

JMS 2.0 is currently maintained under the Java Community Process as **JSR 343.**

Java Messaging Service

Messaging Structure



A set of standard JMS message properties, including QoS settings such as priority and expiry

A user-defined property is a name value pair. This part of the message is used for selector processing.

JMS defines five messages bodies, including map, text, object, stream and bytes.

Java Messaging Service

Messaging Structure - Message Header Fields

Time to Expire	• The length of time a message can exist before it expires and is destroyed by the provider. • Expiration can be set to zero, which means the message will not expire.
Persistent Delivery (Guaranteed)	• Persistent (or guaranteed) delivery means that **delivery of a message is assured;** • It will persist until it is received by all subscribers who requested it. • The message is delivered **once and only once**.
Non-Persistent Delivery (Reliable)	• Messages are **delivered at most once**. • It requires lower overhead and is used in situations where guaranteed delivery is not required.
Priority	• However, this is **not guaranteed**. • There are 10 priority levels - from 0 the lowest to 9 the highest.
Redelivered Flag	• Used in case of **absence of acknowledgement of receipt.** • This flag is set by the **JMS provider application**, usually as the result of a **recovery operation.**
ReploTo	• Contains a Destination, provided by the client, indicating **where a reply to this message should be sent**. • When this field is filled, a response is generally expected.

Java Messaging Service

Messaging Structure - Message Properties

Properties are values that can add to the information contained in header fields, or convey vendor- or application-specific information.

JMS defines specific properties and reserves a block of names for them.

It also provides a **naming convention** for **provider-specific** properties.

Properties can be set when a **message is sent**, or by consumers **upon receiving the message.**

Along with header fields, properties can be **used by the application to filter and route** messages based on specified criteria.

Property values can be of the type **boolean, byte, short, int, long, float, double, and String.**

JMS provides a method to retrieve an **enumeration of all property names** and methods to retrieve the values of the named properties.

Java Messaging Service

Messaging Structure - Message Body

Message	A plain message with no body. This message consists only of the header and properties.
ByteMessage	A stream of uninterpreted bytes. This form can be used to encode information to match formats used by legacy messaging applications.
MapMessage	**Name: String** **Value: Java Primitive Type**
ObjectMessage	A single serializable **Java object or a collection of objects.**
StreamMessage	A stream of **Java primitive values** that are entered and read sequentially.
TextMessage	Text formatted as a java.lang.String.