**Message Queues**

**Synchronous Communication**When client sends request to server and expects server to respond in fraction of millisecond, without any wait/delay.

**Asynchronous Communication**

When server is not expected to respond immediately, on client’s request.

Suppose, there are lakhs of customers who are requesting there past few months’ invoices to an online store’s server. The server cannot handle all requests all at once.  
All these requests are first lined up in a queue and handled one at a time.

Hence, this is an async communication.

**Message Queue**

A queue can be a process which has data structure in memory to store messages.

It could be a process running on the same machine as the components, which interact with queue.

It can be a process on different machine and components have to interact with that machine.

Ex: SQS, Kafka, Rabbit MQ

These are dedicated queue architectures built for handling highly scalable and complex systems.

Queue can store messages. Messages are short sized data which just tell what has to be done.

Message Queue

Producers Consumers

Components which send messages to queue are Producers. They tell what has to be done.

Consumers are components which carry out the operation.

Once the consumer processes the message, it sends an acknowledgement to queue that the message has been processed. Then, the queue pops out that message.  
If there is failure in processing the message, it is not popped out from queue.

**Advantage:**

We can handle a lot of requests with Queue.

Producer can keep pushing messages in queue. Consumer can keep processing.

If queue is not there, server will not be able handle load. **Queues decouple the load**.

If number of consumers are more, more requests are served per unit time.

Suppose, the **consumers go down**. The requests stay intact in queue. They are **not lost**.

**Ordered and Unordered Queue**

Suppose you build a chat application, where the order in which messages are delivered are important. Then in that case, ordered queue is used.

Here, each message is served one at a time in order, suppose a consumer fails to serve the one of the requests/ message. Then next message in queue is not allowed to be consumed or proceed. As the order is important.

The failed message is retried multiple times, and all messages in queues are stopped for some time till the failed message is passed.

In unordered queue, if a message fails, other messages in queue are put on hold till the failed message passes, they are taken up without waiting and the failed message is either handled separately by different consumer or put back in queue at end.