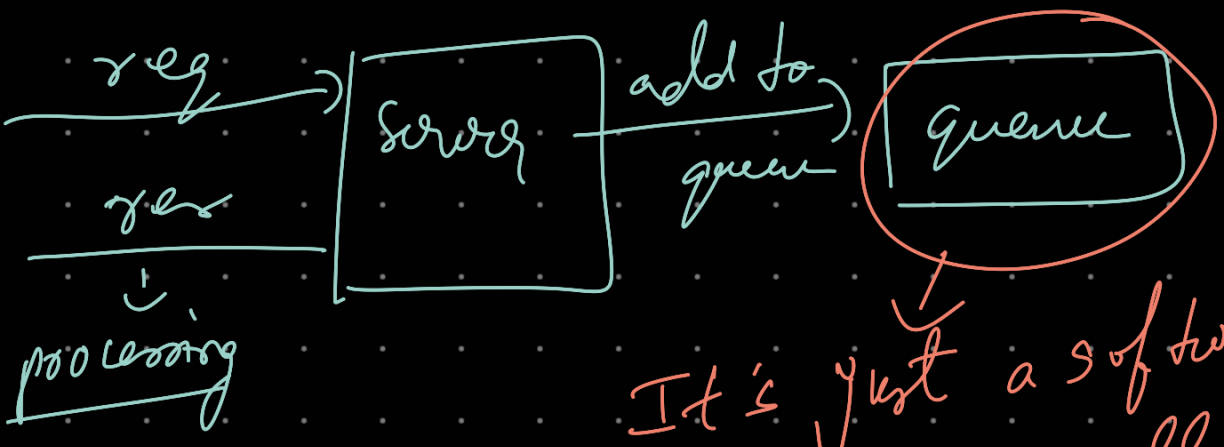
[Message Queues]

* 

* On client side, no response is recieved, can also lead to timeout

* 

↳ Better UX

It's just a software running on generally another Machine

(Eg-) Kafka, RabbitMQ etc.

Will this be useful for Microservices?

Lets talk about monolith

* Once user reacts to some post

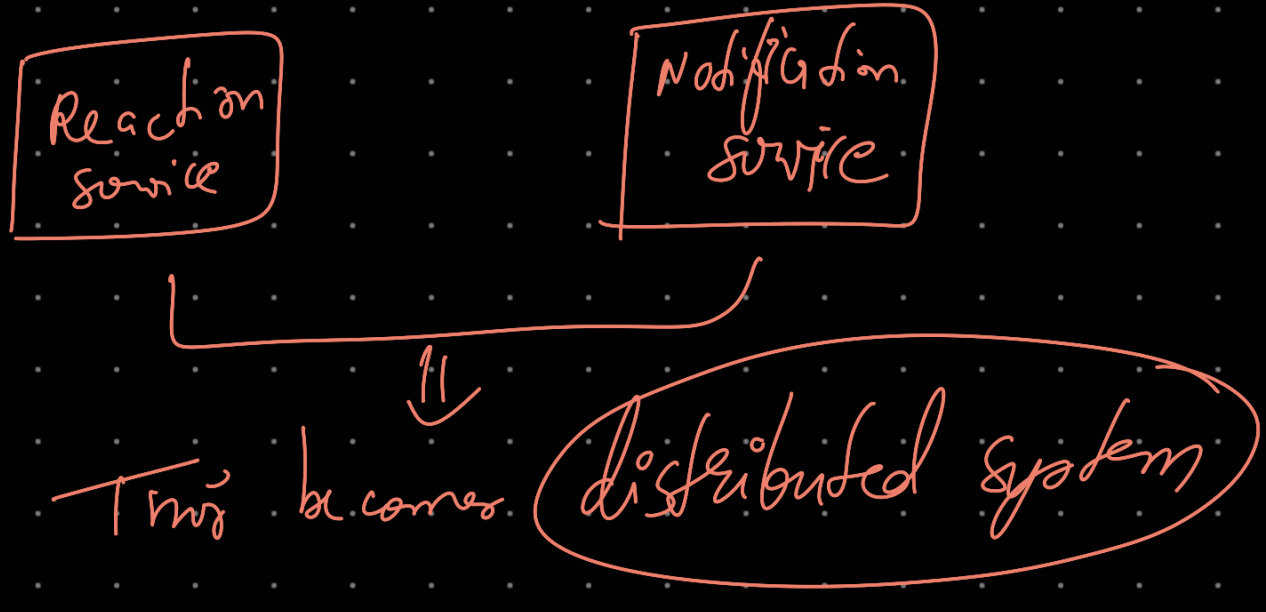
↓

Send notification to owner of post

* In monlith, this would be just function call

||

No lets, say two services



How do two service interact

[Synchronous]

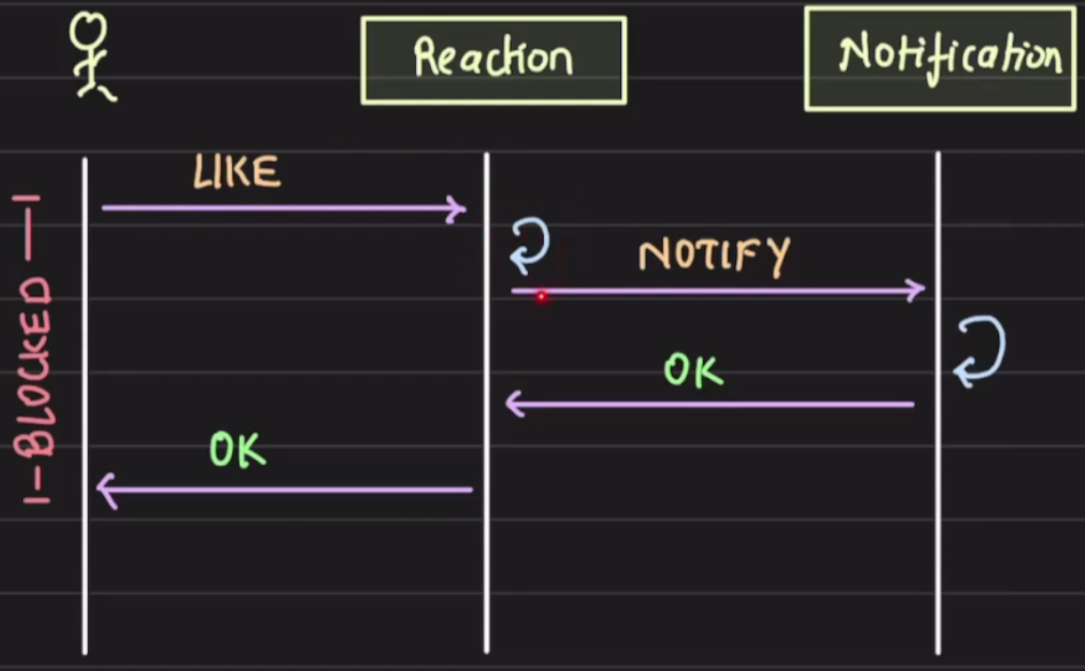
Synchronous Communication

User

a reliable connection connecting the two services

One service sends the request, and waits for the response from the other service

Request made from one service to another is BLOCKING



How to implement Synchronous?

1. REST API
2. GraphQL
3. gRPC

* Reaction service should know endpoint of Notification service.

* This becomes more complex when we have lots services in our system

Advantages of Synchronous Communication

↳ Communication happens in realtime

↳ Super simple, intuitive

Disadvantages of Synchronous Communication

↳ Caller is blocked until the response is received (ms, sec, minutes)
  ↳ would be an issue if it takes too much time
  (Timeouts)

↳ servers need to be pro-actively provisioned for peaks.

* And leads to Cascading failures.

When should you use synchronous communication ?

↳ when you cannot move on → You need result before you move forward

eg: Database Queries , API responses

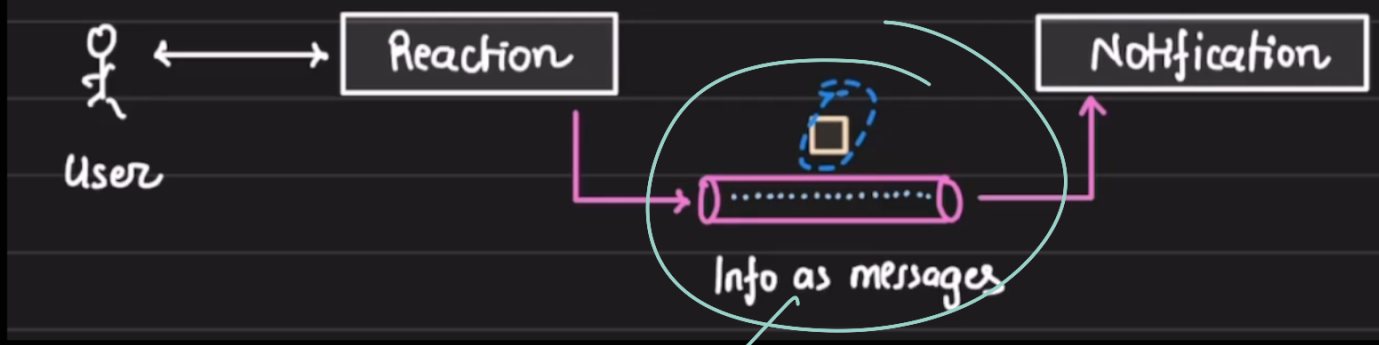
↳ when you want realtime response

eg: chat, checkout

↳ when it will take relatively less time to compute and

[Asynchronous]

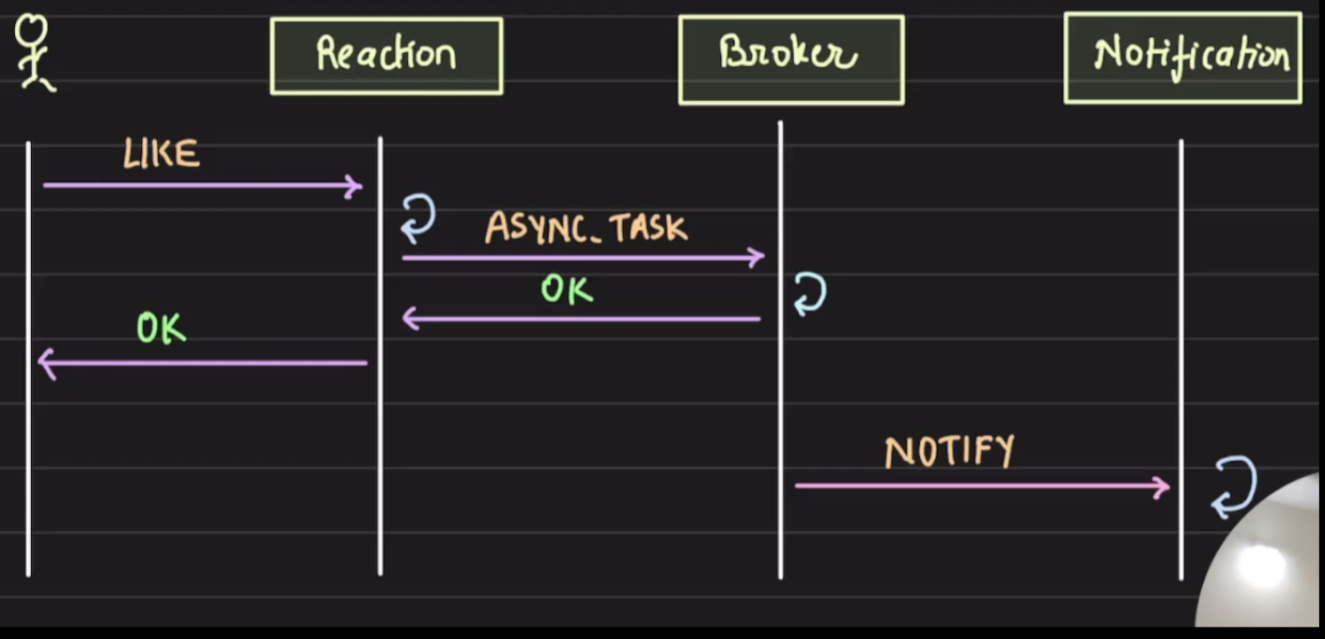
Asynchronous Communication



Message Queue

The messages are buffered in the broker and the consuming services will consume them when it can.
If the consuming service is down, the messages will be consumed when the service comes back up again. So, no cascading failure

Request made from one service to another is NONBLOCKING



⇓

Some specialised software helps in Async Communication and manages Message Queues

⇓

Kafka, RabbitMQ etc

↳ Message Brokers

Disadvantages of Asynchronous Communication

↳ Eventual Consistency

You cannot have a strongly consistent system with brokers and hence you have to be okay for the messages to be eventually consumed.

But with brokers our system does scale better

↳ Broker is a SPoF

The message broker is the backbone of the system, hence we need to be super cautious about it.

The broker we use should be horizontally scalable

↳ Harder to track the flow of communication

↳ Tracing

When should we use Asynchronous communication?

↳ when delay in processing is okay

eg: notification, analytics, reporting

↳ when the job at hand is long-running

eg: provisioning a server, order tracking, DB backups

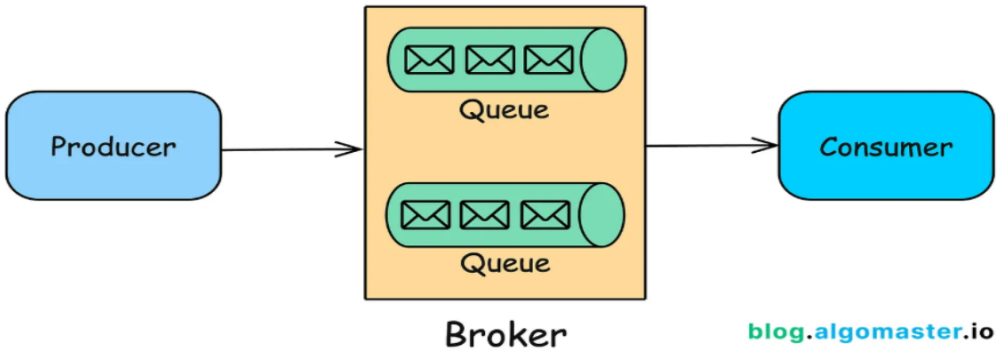
↳ when multiple services need to 'react' to same event

eg: blog published → index in search
→ notify the followers
→ update user analytics

↳ when it is okay for you to allow failures and retries

eg: send notification. if failed retry

Core Components of a Message Queue



1. Producer/Publisher

The entity that sends messages to the queue. Producers push messages into the queue without worrying about the consumer's state.

2. Consumer/Subscriber

The entity that reads messages from the queue. Consumers pull messages from the queue and process them.

3. Queue

The data structure that stores messages until they are consumed.

4. Broker/Queue Manager

The software or service that manages the message queue, handles the delivery of messages, and ensures that messages are routed correctly between producers and consumers.

5. Message

The unit of data sent through the queue. A message typically contains the **payload** (the actual data being sent) and **metadata** (such as headers, timestamps, and priority).