CS 7172
Parallel and Distributed Computation

Publish and Subscribe

Kun Suo

Computer Science, Kennesaw State University

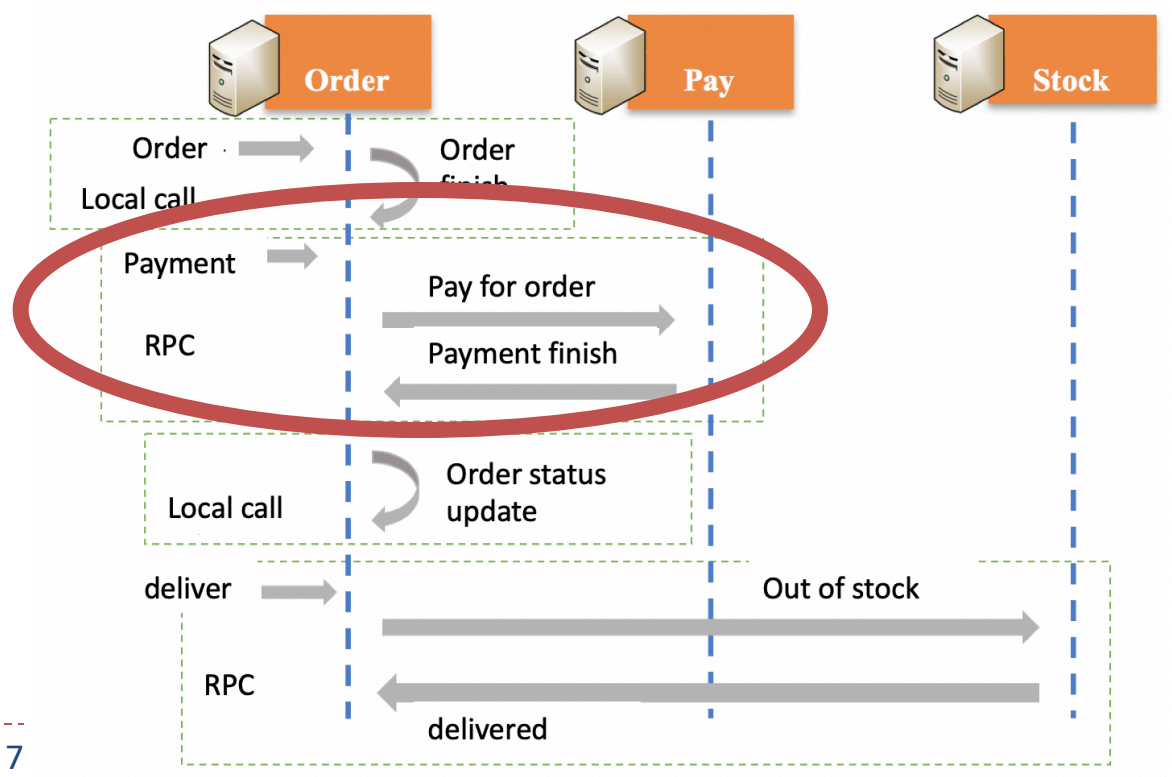
https://kevinsuo.github.io/

Outline

- RPC problems and what is pub/sub model
- P2P model vs. pub/sub model
- An example of pub/sub model: Kafka
- Design issues in pub/sub model
- Observer Model vs. Publish/Subscribe Model

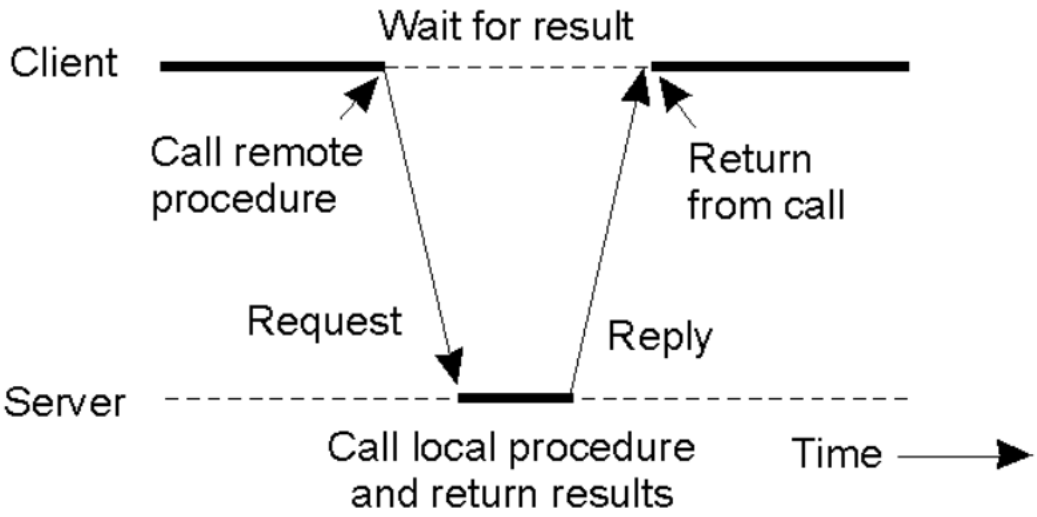
RPC and its Problems

- The caller uses the parameter passing method to call a function on the local machine to execute the function on the remote machine and receives the return results



RPC and its Problems

- Usually, RPCs are blocking
  - Thus, also useful for synchronization

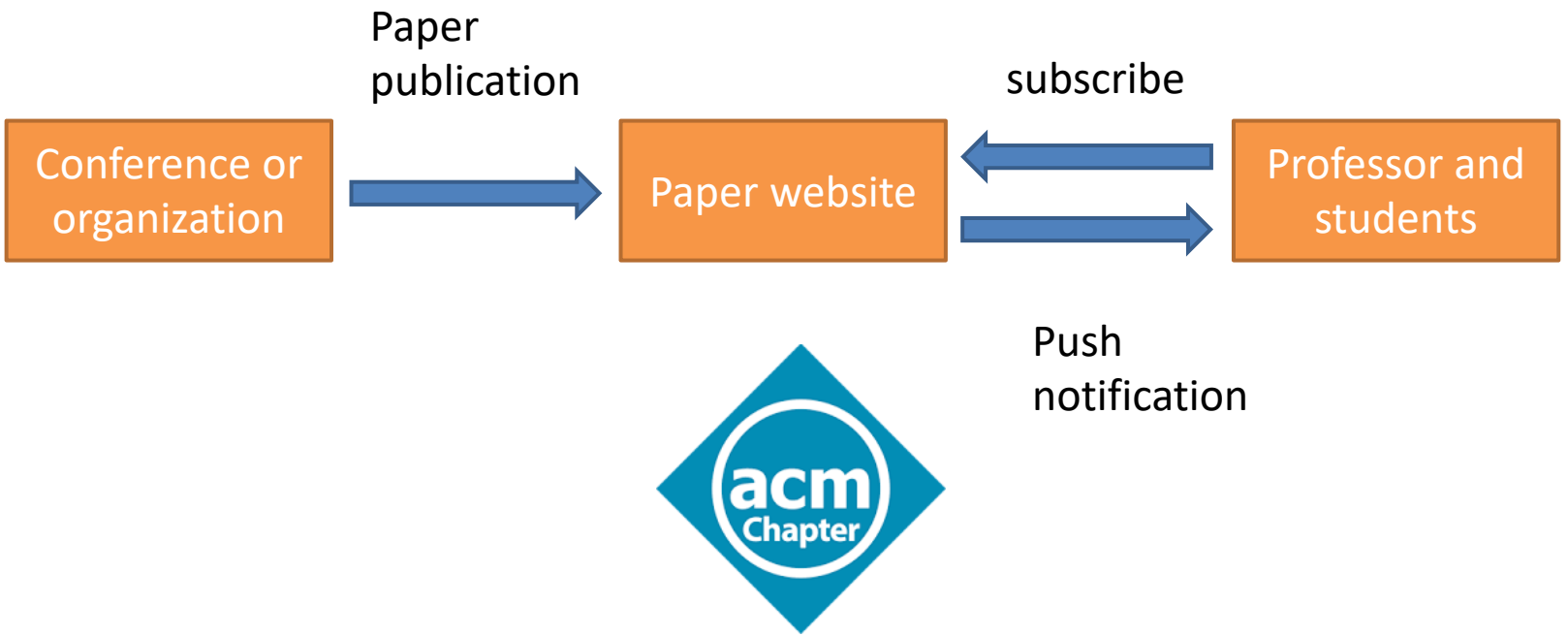


RPC and its Problems

- Most of RPC works in blocking call, which is only efficient for small number of processes
- It requires to maintain the function APIs and <function, ID> tables. When one function changes, all processes communicate with it will be influenced
- The synchronization in RPC has significant overhead as the scale of distributed system grows and system becomes more complex

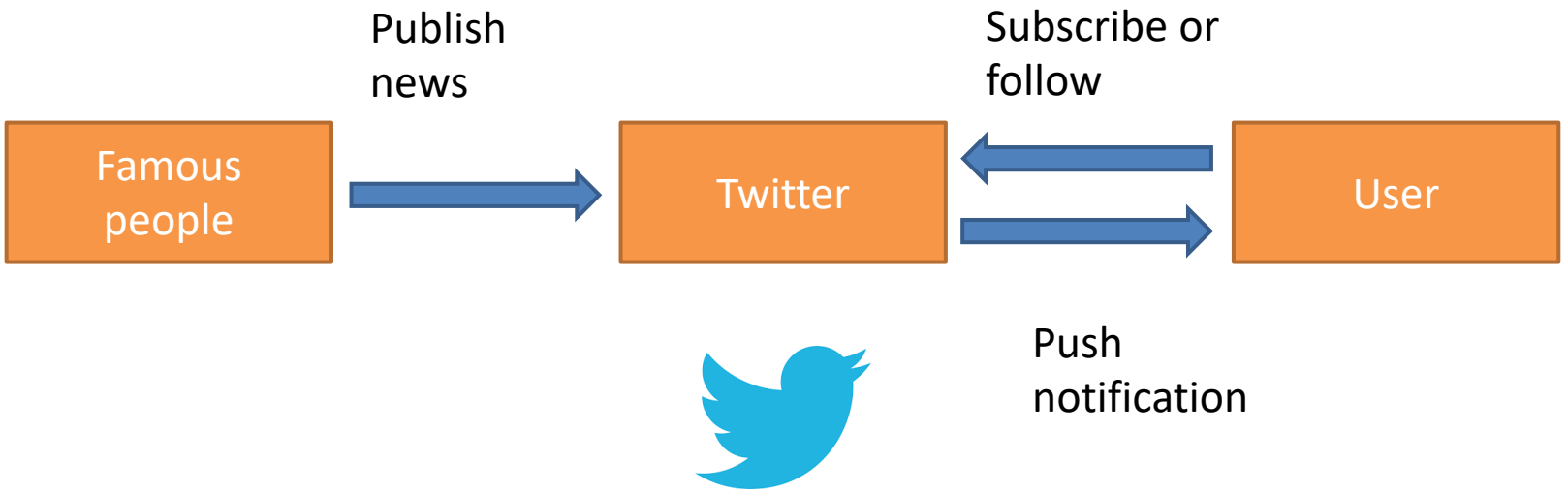
What is publish & subscription?

- Example: paper publication



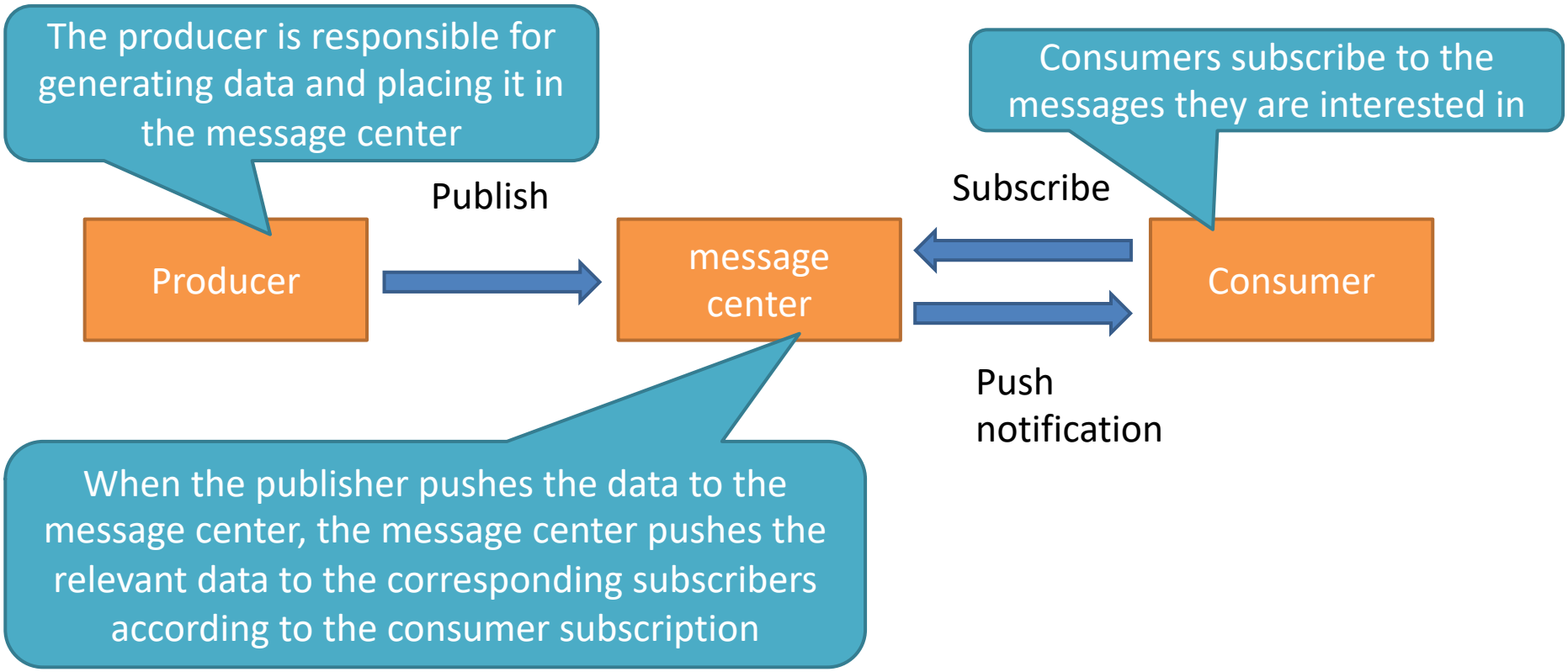
What is Publish/Subscribe Messaging Model?

- Example: twitter news feed



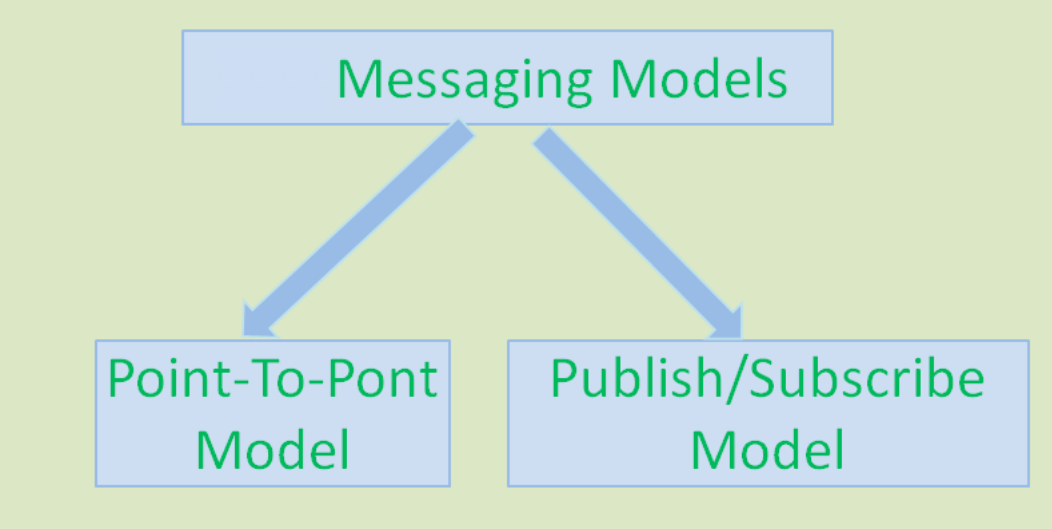
What is Publish/Subscribe Messaging Model?

- A general case

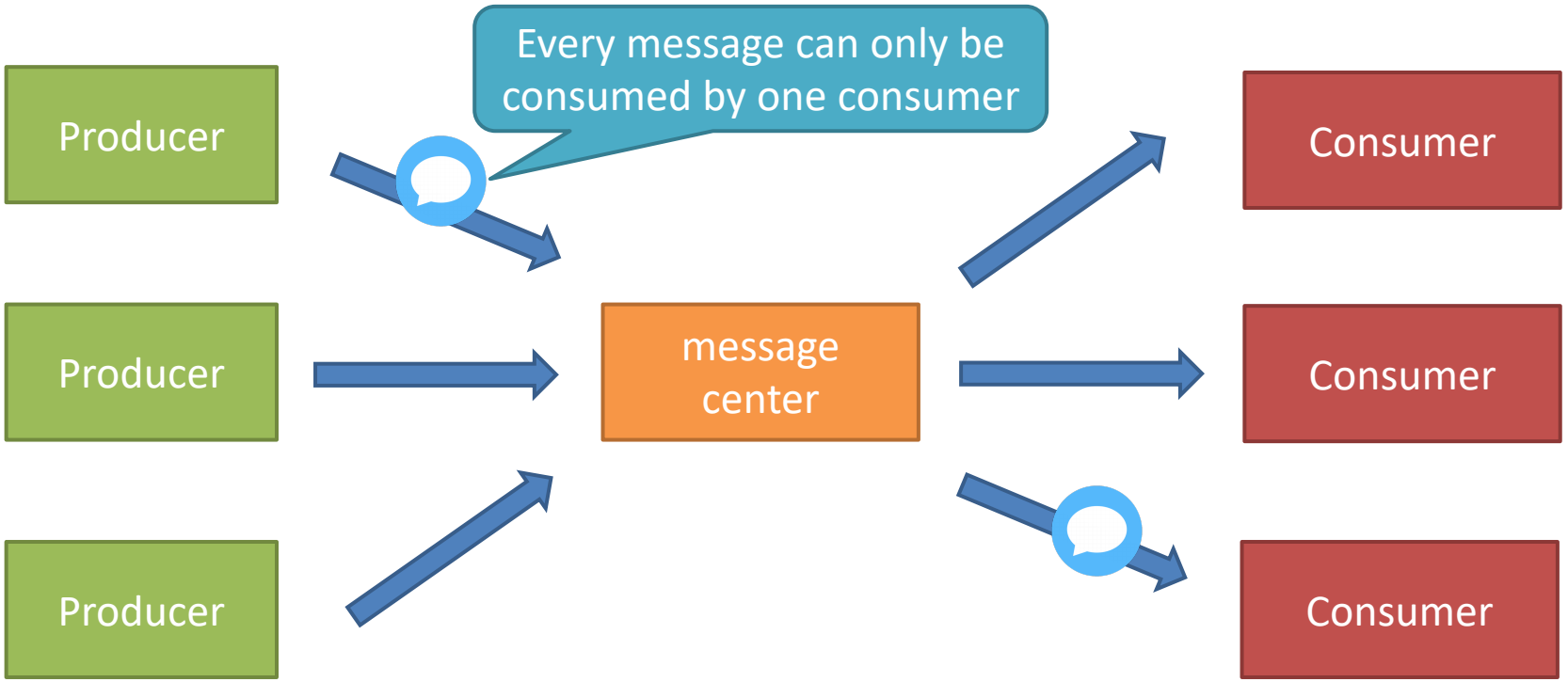


How publish & Subscribe works?

- Two models in distributed asynchronous message system:
  - P2P: Point to Point
  - Publish/Subscribe



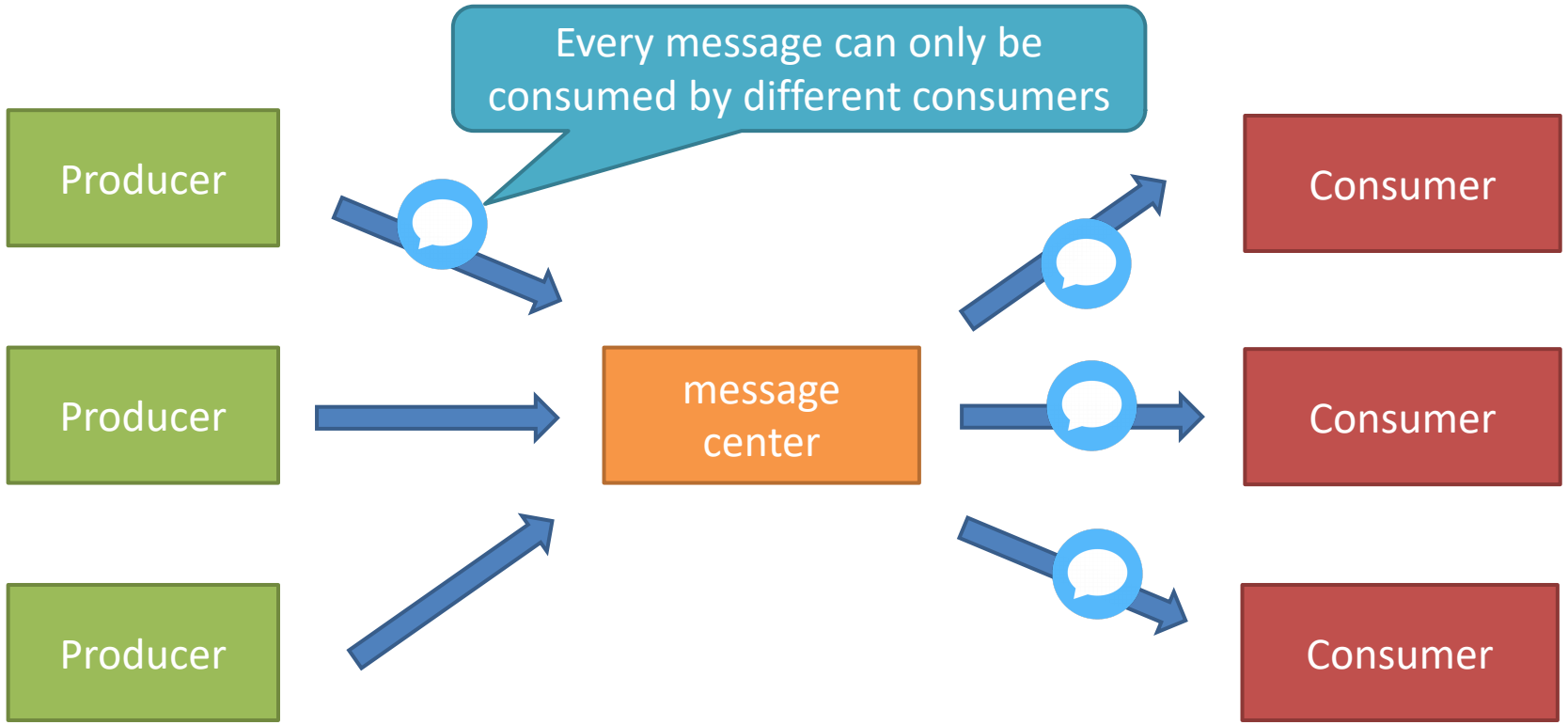
What is Point to Point model?



Point to Point model

- Point-To-Point model follows these concepts:
  - P2P Model uses “Queue” as Destination
  - In P2P Model, a Sender or Producer creates and sends messages to a Queue.
  - In P2P Model, a Receiver or Consumer receives and reads messages from a Queue.
  - In P2P Model, a Message is delivered to one and only one Consumer.
  - Any message should be delivered to one and only one Receiver.
  - There is no timing dependency between Sender and Receiver. That means the Receiver can consume the messages whether it is alive or not when the Sender sent that message.
  - In this model, Destination stores messages till its consumed by Receiver.
  - Receiver sends acknowledgements to Sender once it receives messages.

Publish/Subscribe Messaging Model



Publish/Subscribe Messaging Model

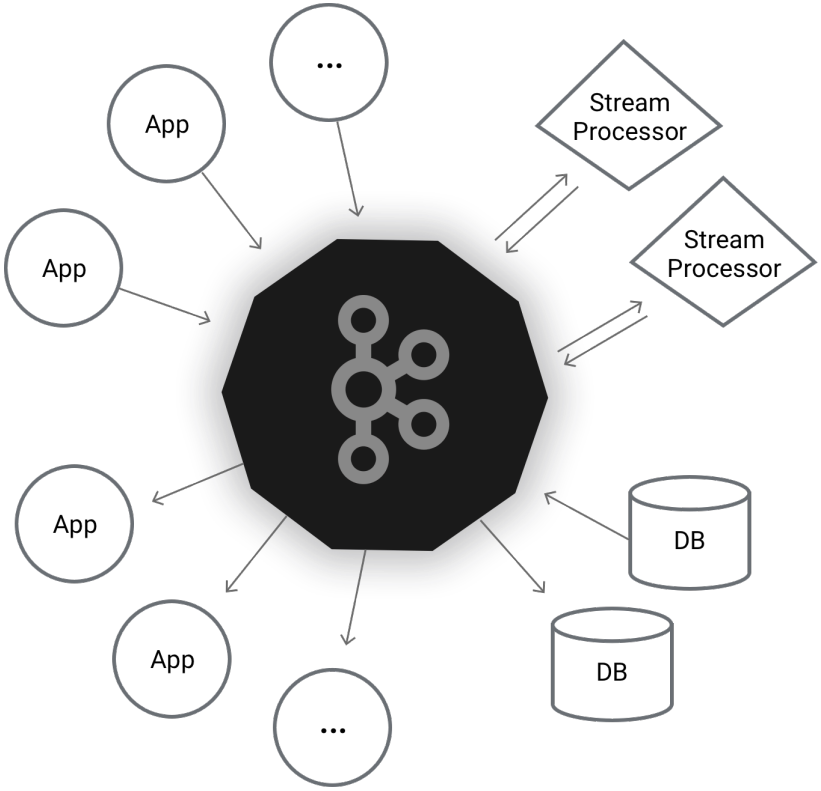
- Publish/Subscribe model follows these concepts:
  - Pub/Sub model uses Topic as Destination.
  - Publisher creates and publishes messages to Topics.
  - Subscriber subscribes to interested Topics and consumes all messages.
  - Pub/Sub Messaging model has timing dependency. That means Subscriber can consume messages which are published to the Topic only after it subscribes to that Topic. Any messages posted before its subscription or any messages posted when it is inactive, cannot be delivered to that Consumer.
  - Unlike P2P Model, in this model Destination does not store messages.

Differences between P2P and Pub/Sub Messaging Model

Point-To-Point Messaging Model	Publish/Subscribe Messaging Model
Each message is delivered to one and only one Receiver	Each message is delivered to multiple Consumers.
P2P Model has no timing dependency.	Pub/Sub model has some timing dependency.
Receiver sends acknowledgements to Sender once it receives messages.	Acknowledgement is not required.
Destination stores messages till its consumed by Receiver	Destination does not store messages

Example: Kafka

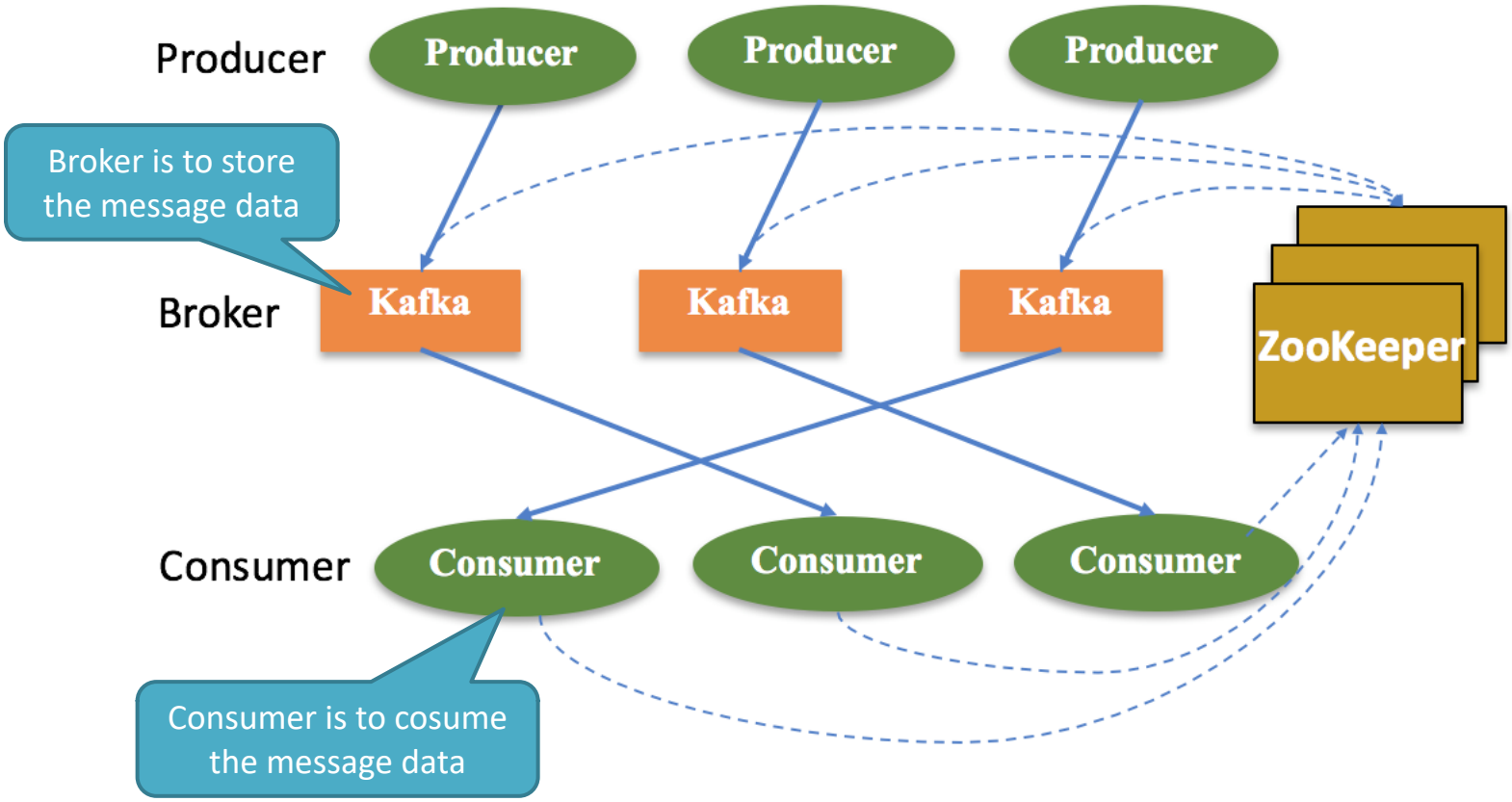
- Apache Kafka is an open-source stream-processing software platform developed by LinkedIn
- The project aims to provide a unified, high-throughput, low-latency platform for handling real-time data feeds
- https://kafka.apache.org/



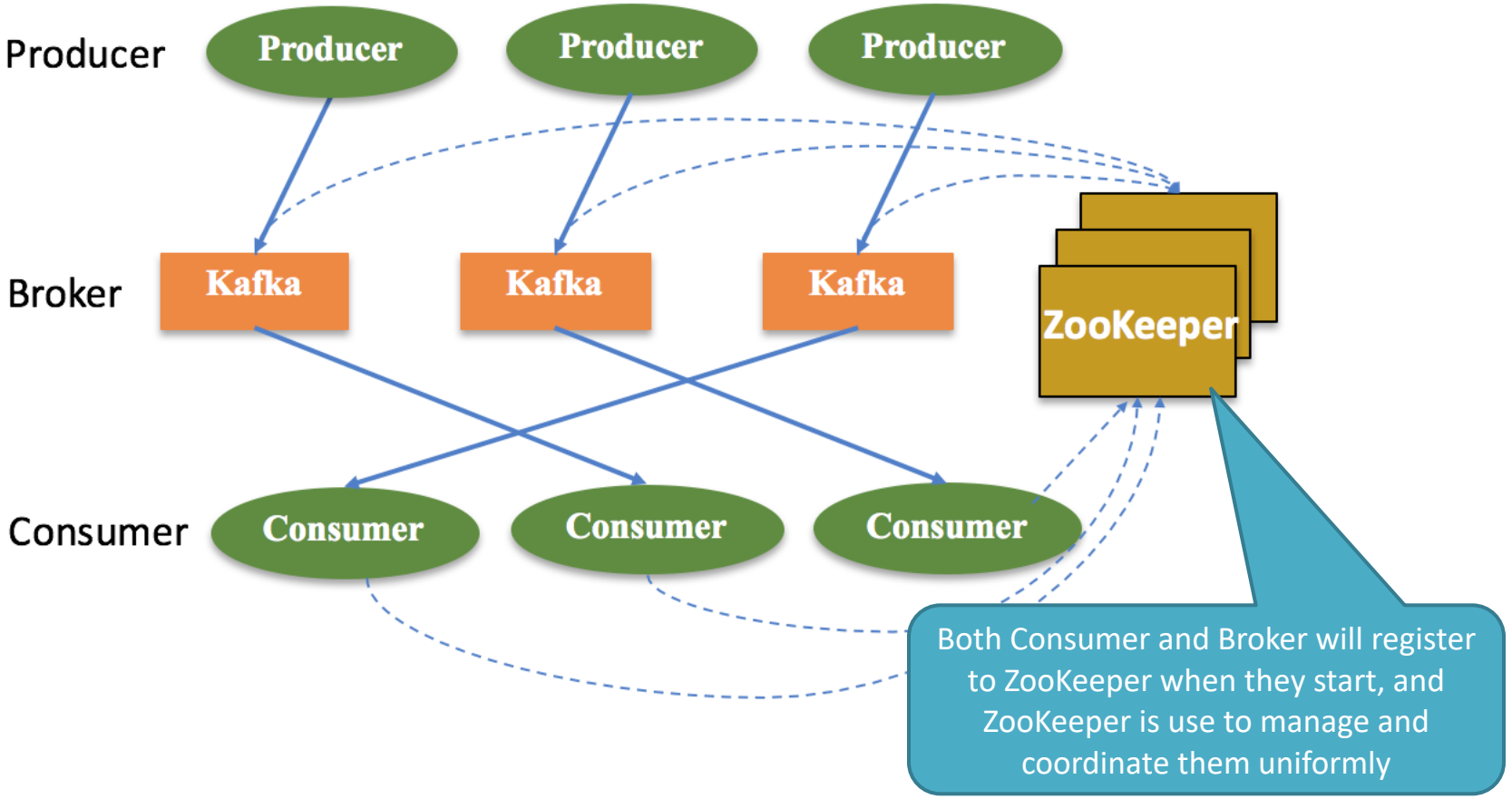
Kafka architecture

- The architecture of Kafka includes 3 parts:
  - Producers are responsible for publishing messages to the message center;
  - Consumers subscribe to messages of interest to the message center, and then perform data processing after obtaining data;
  - Message center (Broker) is responsible for storing messages published by producers and managing consumer subscription information and pushing messages to consumers based on consumer subscription information.

Kafka architecture

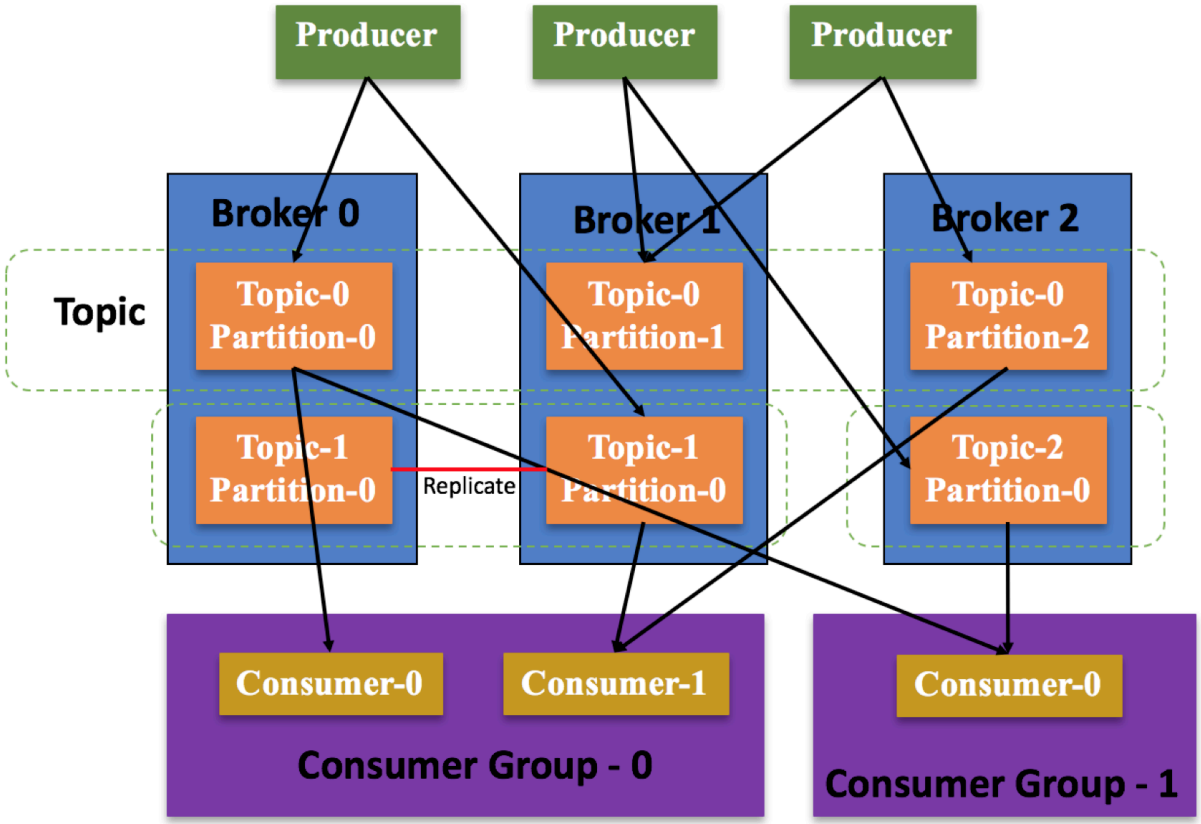


Kafka architecture



Topics in Kafka

- Topic refers to message types or data types
- Each producer can publish different topics of messages
- Each consumer can subscribe different topics of messages



Topics Example

Home

Search Twitter

Salesforce Developers @SalesforceDevs · 11m
If you don't have a ticket to #TDXIndia19 yet, here are our top seven reasons to attend. 💡 sforce.co/2QTnJnC

2 5

honeycomb @honeycombio
Happy engineers, happy customers, and a stronger business overall - learn how to develop a culture of observability. info.honeycomb.io/culture-of-obs...

#WarmUpWithJameson
Whiskey Season is Here. Join Us.
Promoted by Jameson U.S.

Trending in United States
#WhatLiberalsCallThanksgiving

Hobbies and interests · Trending
Toyota Camry
4,978 Tweets

Politics · Trending
Pete Buttigieg Called Me
4,716 Tweets

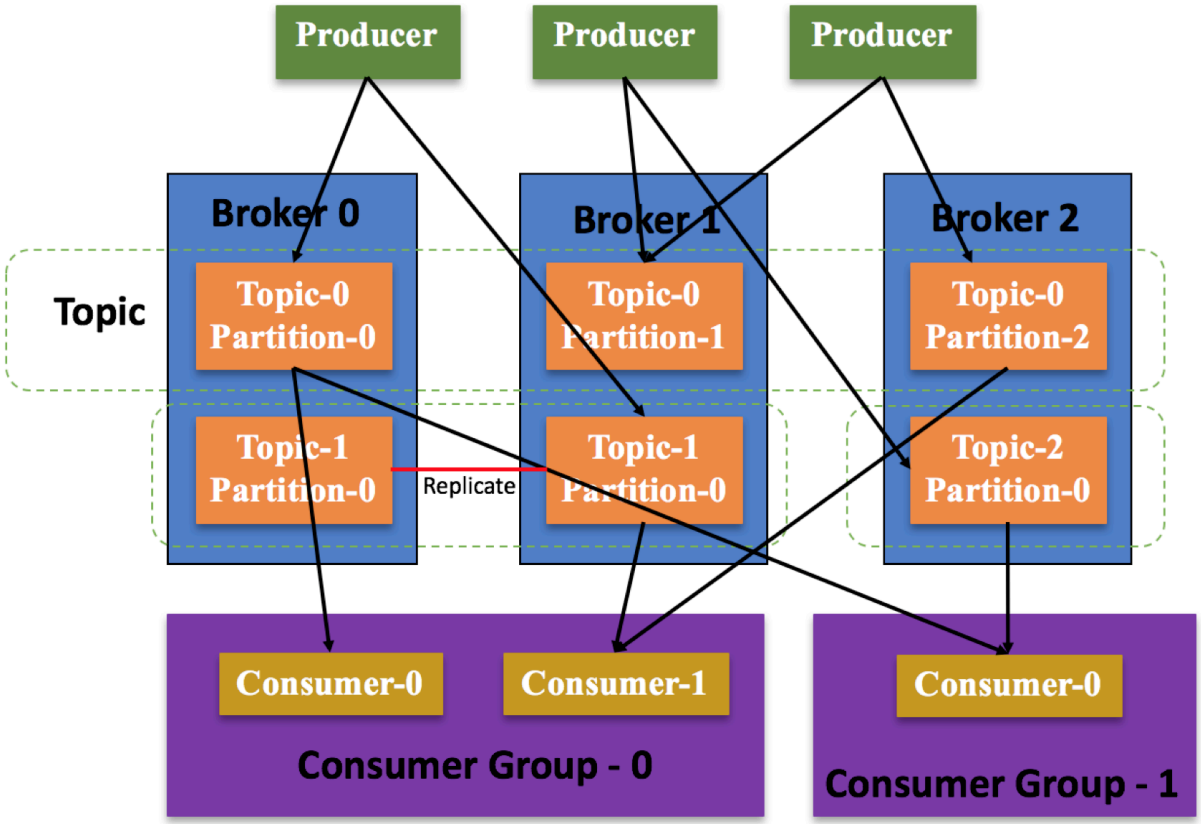
Politics · Trending
#Investigatellhan
20.4K Tweets

Politics · Trending
#PeteButtigiegIsALyingMF
9,050 Tweets

Politics
Op-ed in The Root accuses Pete ...

Partitions in Kafka

- Each topics can be divided into different partitions
- Different partitions are existed at different brokers
- One partition of one topic is located at one broker (e.g., Topic-0, partition-0)

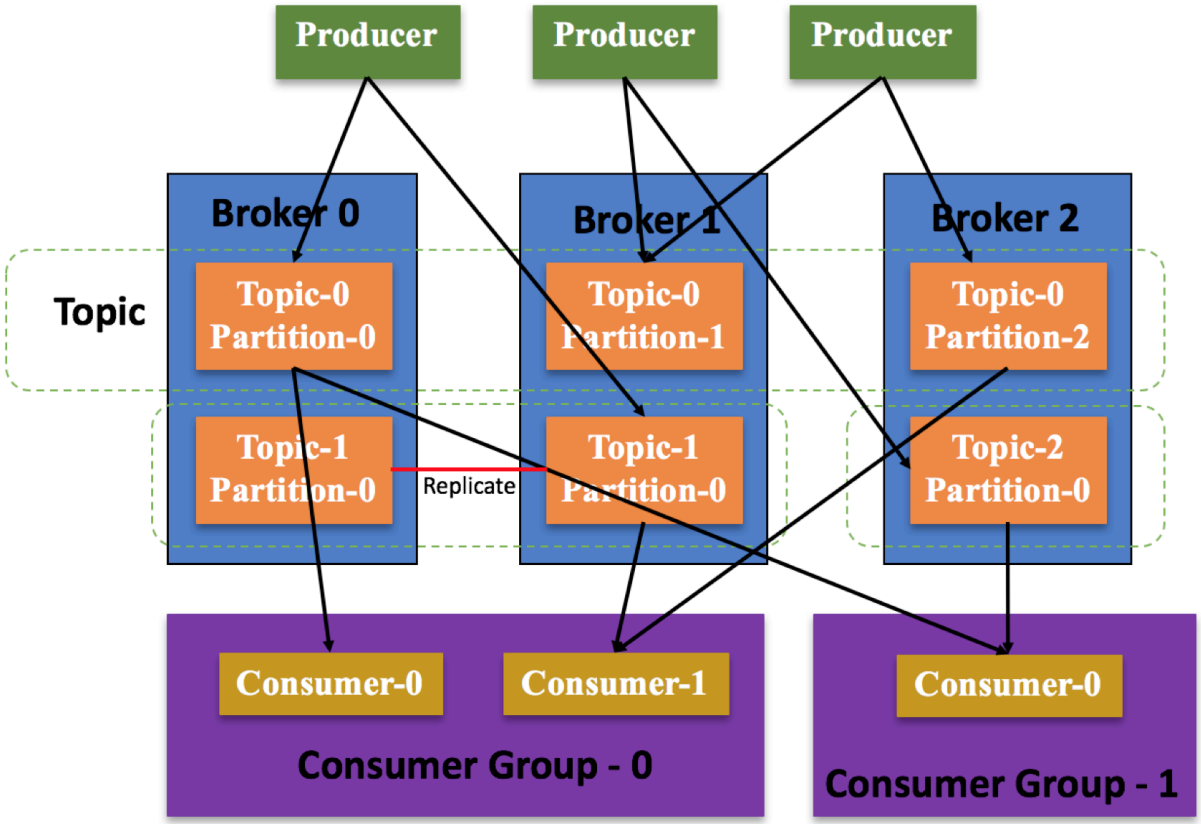


Partitions in Kafka

- The benefits of partitions:
  - Load balance: avoid too high load on a single broker.
    - For instance, topic 0 is divided into partition-0, partition-1, partition-2, and is located at broker-0, broker-1, broker-2. Thus, topic 0 is at three machines.
  - Message backup: achieve high reliability.
    - For instance, topic 1 contains two partition-0s, which are stored at broker 0 and broker 1.

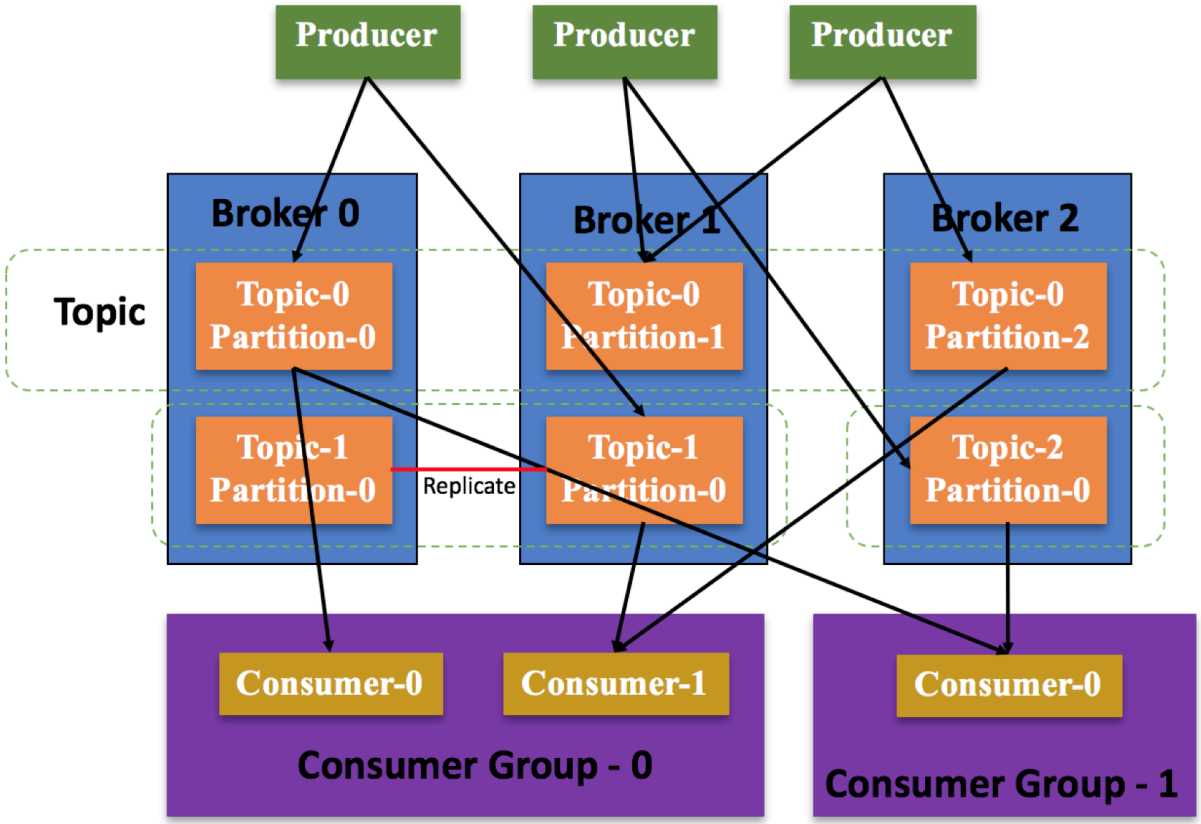
Consumer group in Kafka

- Consumer group is a set of consumers
- Every message of one topic can only be consumed by a single consumer with one consumer group



Consumer group in Kafka

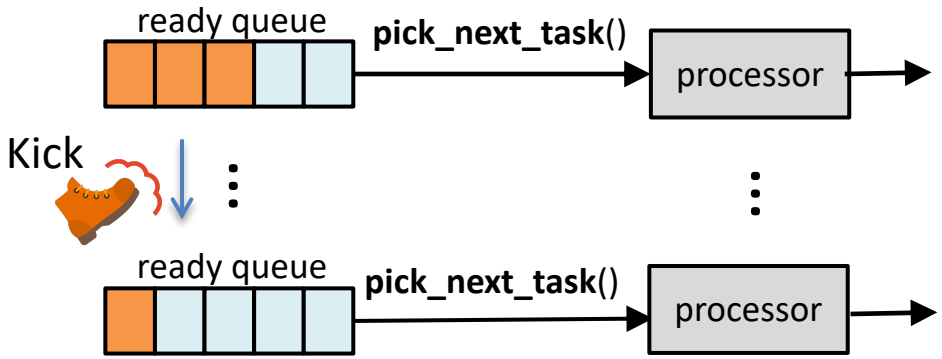
- The benefits of consumer group:
  - Because one consumer is limited to consume messages, consumer group can avoid storage overflow of broker when there are too many messages



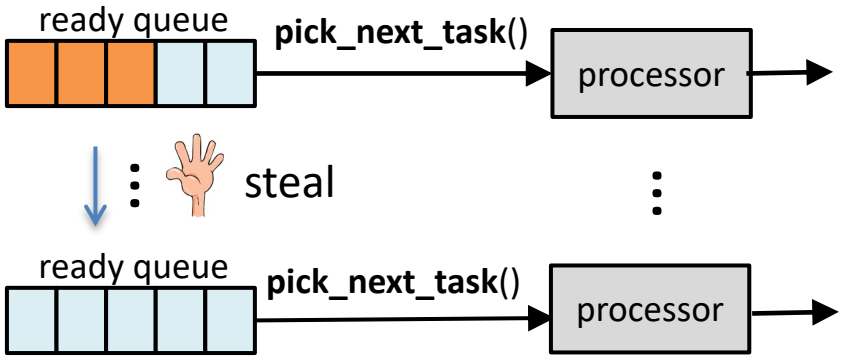
Revisit Load Imbalance in OS

Every a while, a kernel thread checks load imbalance and move threads

Made by OS



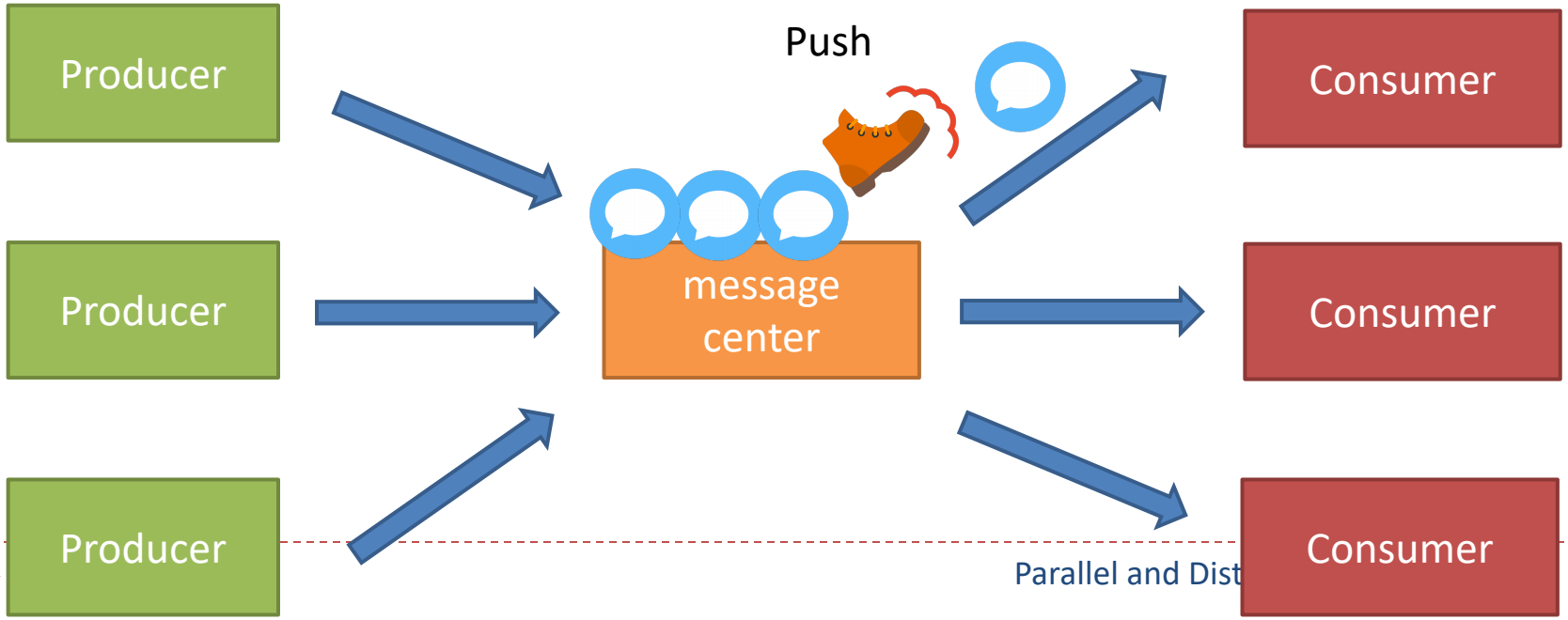
Whenever a queue becomes empty, steal a thread from non-empty queues



Made by local queue. Both are widely used

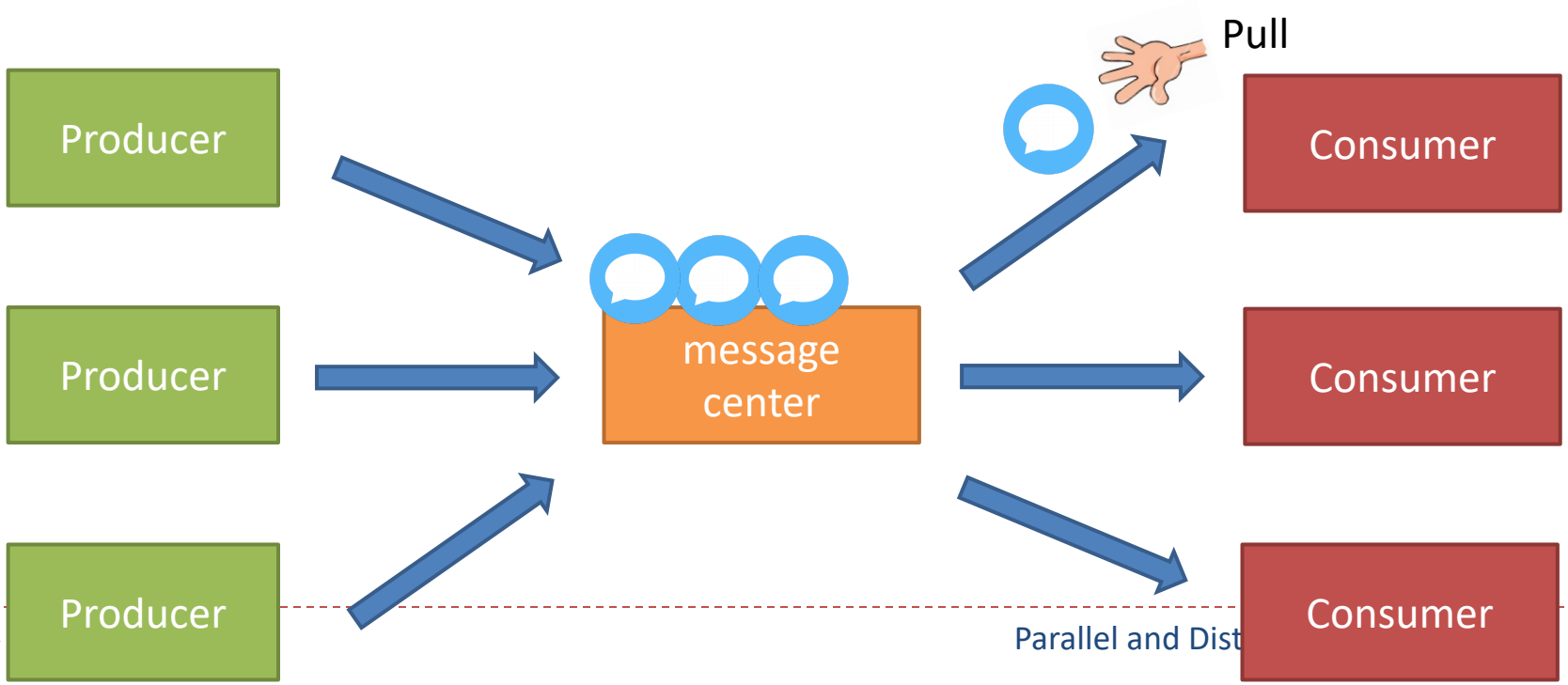
Load Imbalance in Publish/Subscribe Models

- Message center pushes message to consumers:
  - Controlled by the message center
  - Risky for the consumer side because too many messages could overwhelm the consumer side

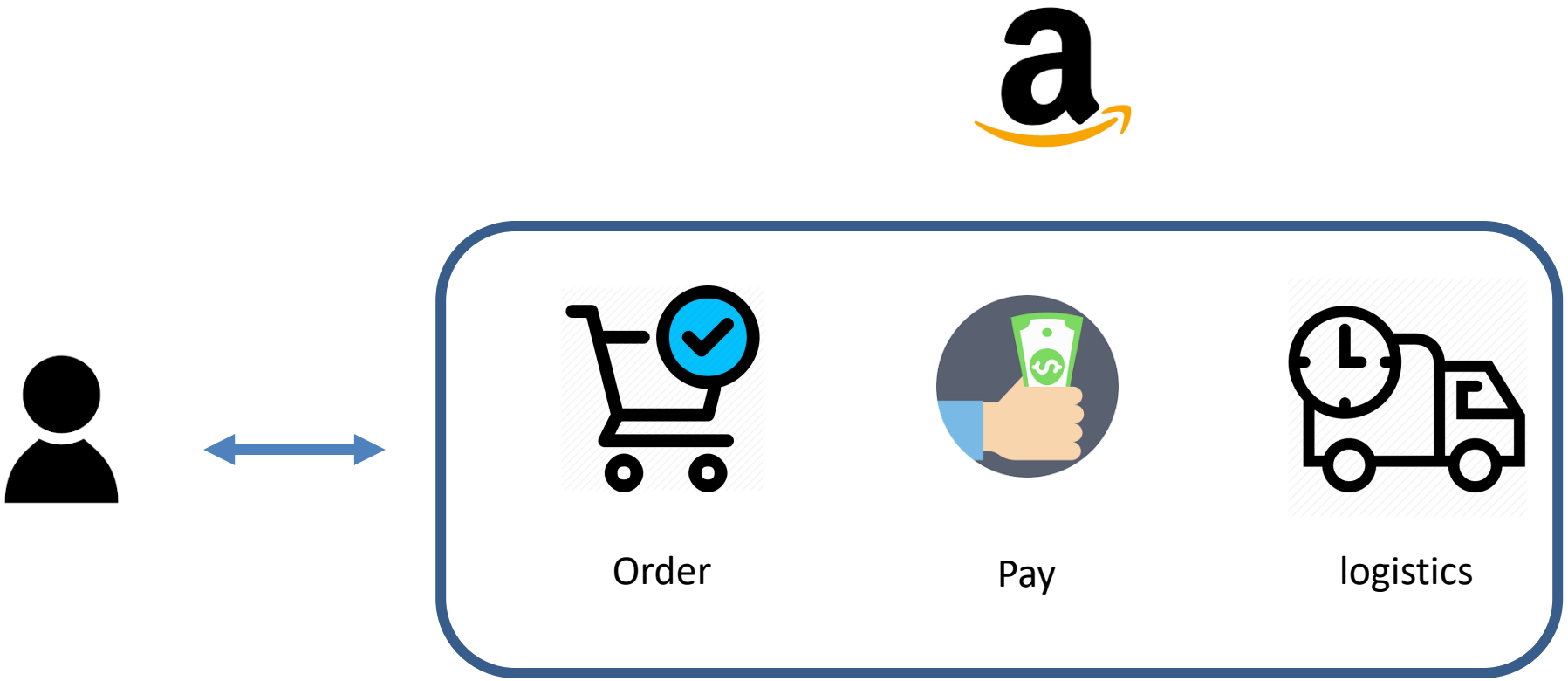


Load Imbalance in Publish/Subscribe Models

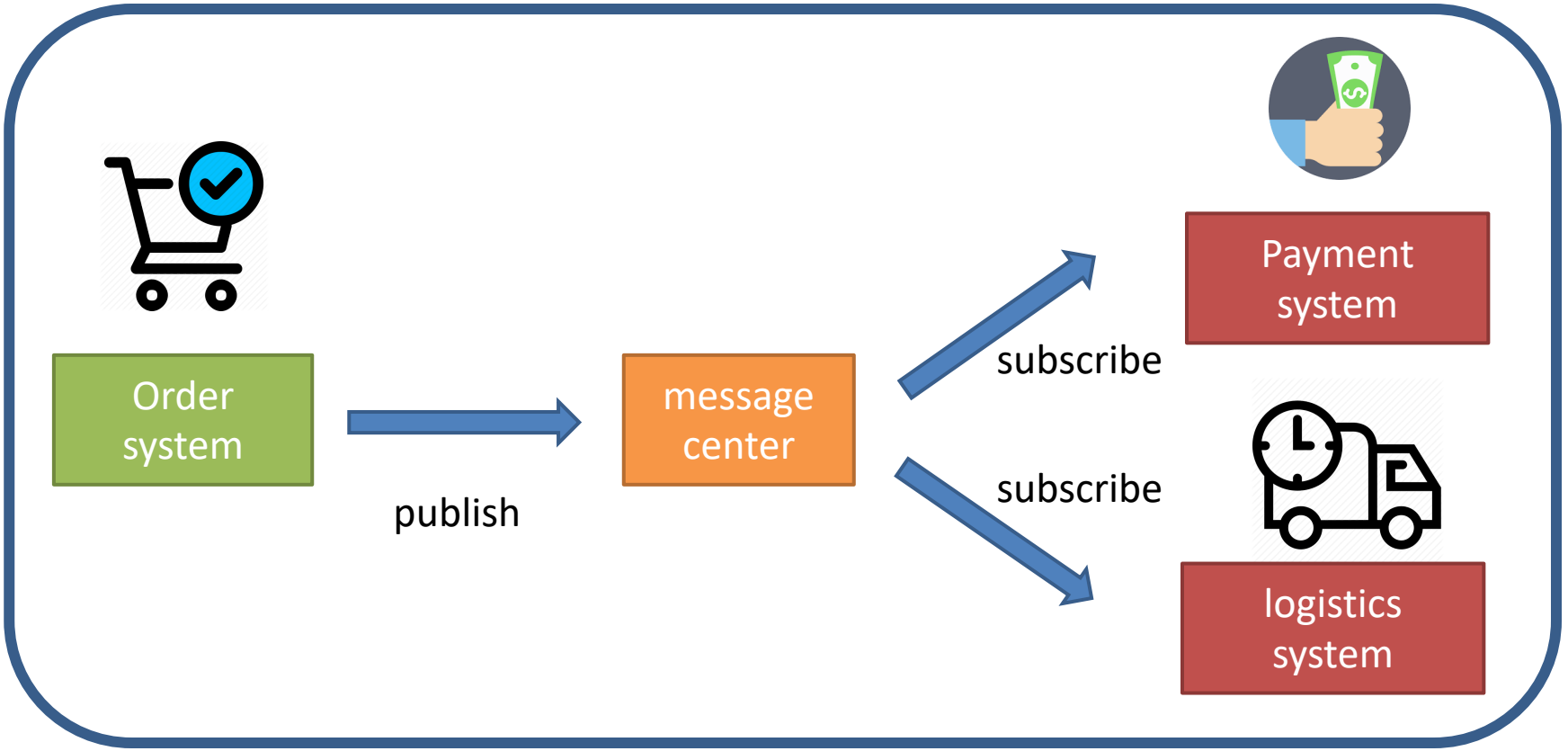
- The message is pulled by the consumer actively
  - Controlled by the consumer
  - Risky for the message center side because too many messages could use up the storage resource of message center and make some messages lost



Publish/Subscribe Models in Amazon



Publish/Subscribe Models in Amazon



Publish-subscribe in ZeroMQ

Server

```
1 import zmq, time
2
3 context = zmq.Context()
4 s  = context.socket(zmq.PUB)          # create a publisher socket
5  p = "tcp://"+ HOST +":"+ PORT       # how and where to communicate
6  s.bind(p)                           # bind socket to the address
7 while True:
8    time.sleep(5)                     # wait every 5 seconds
9    s.send("TIME " + time.asctime()) # publish the current time
```

Client

```
1 importzmq
2
3 context = zmq.Context()
4 s  = context.socket(zmq.SUB)          #  create a  subscriber socket
5  p = "tcp://"+ HOST +":"+ PORT       # how and where to communicate
6  s.connect(p)                        # connect to the server
7 s.setsockopt(zmq.SUBSCRIBE, "TIME")  # subscribe to TIME messages
8
9  for i in range(5):  # Five iterations
10   time = s.recv()   # receive a message
11   print time
```

Advantages of Publish/Subscribe Models

1. The system is decoupled and easy to maintain.
   - Producers/publishers are only responsible for the release of messages
   - Producers/publishers do not need to know the number of subscribers/consumers
   - Producers/publishers do not need to know what subscribers/consumers use messages for
   - Subscribers/consumers do not need to know when Producers/publishers post messages.

Decoupled in Different Aspects

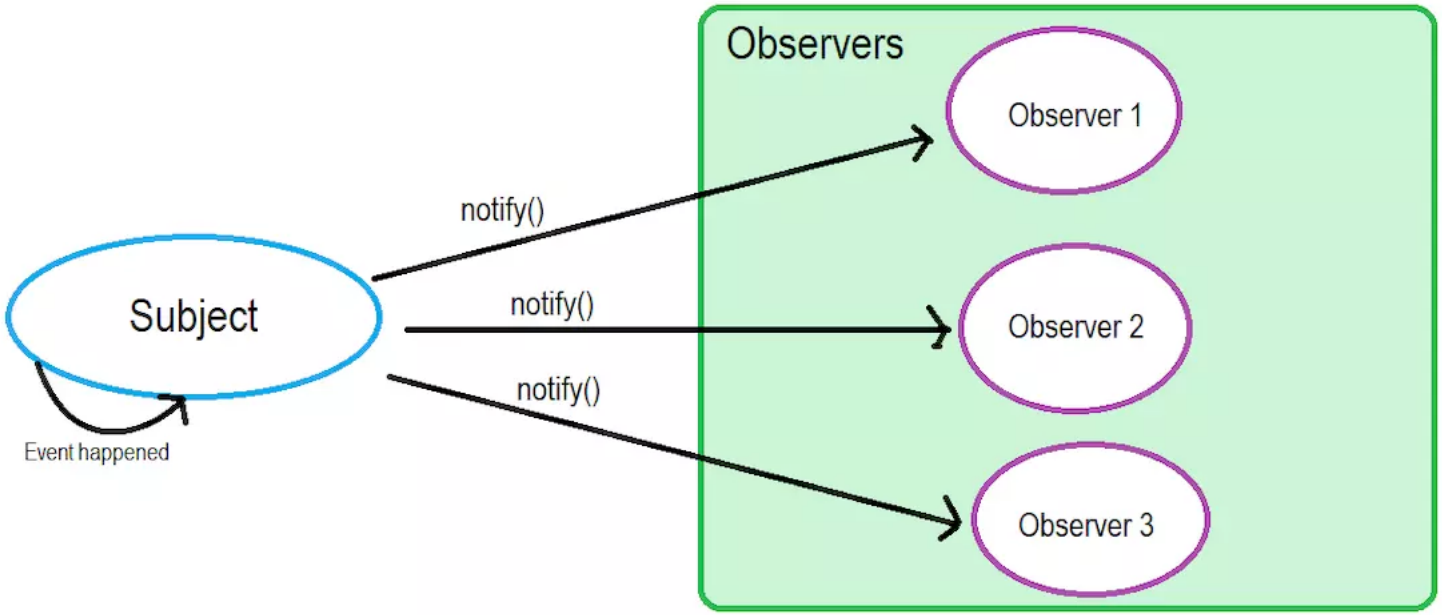
- Space decoupling. Interacting parties don't need to know about each other
  - Publishers/subscribers don't need to hold references to each other
- Time decoupling. Need not be actively participating/running at the same time.
  - Subscribers can get disconnected and be notified later
- Synchronization decoupling. Sender, receiver should not block
  - Publishers not blocked while producing events
  - Subscribers can get async notified via callbacks
  - Production/consumption do not happen in main flow of control
- Increases scaling, since there is no explicit dependency

Advantages of Publish/Subscribe Models

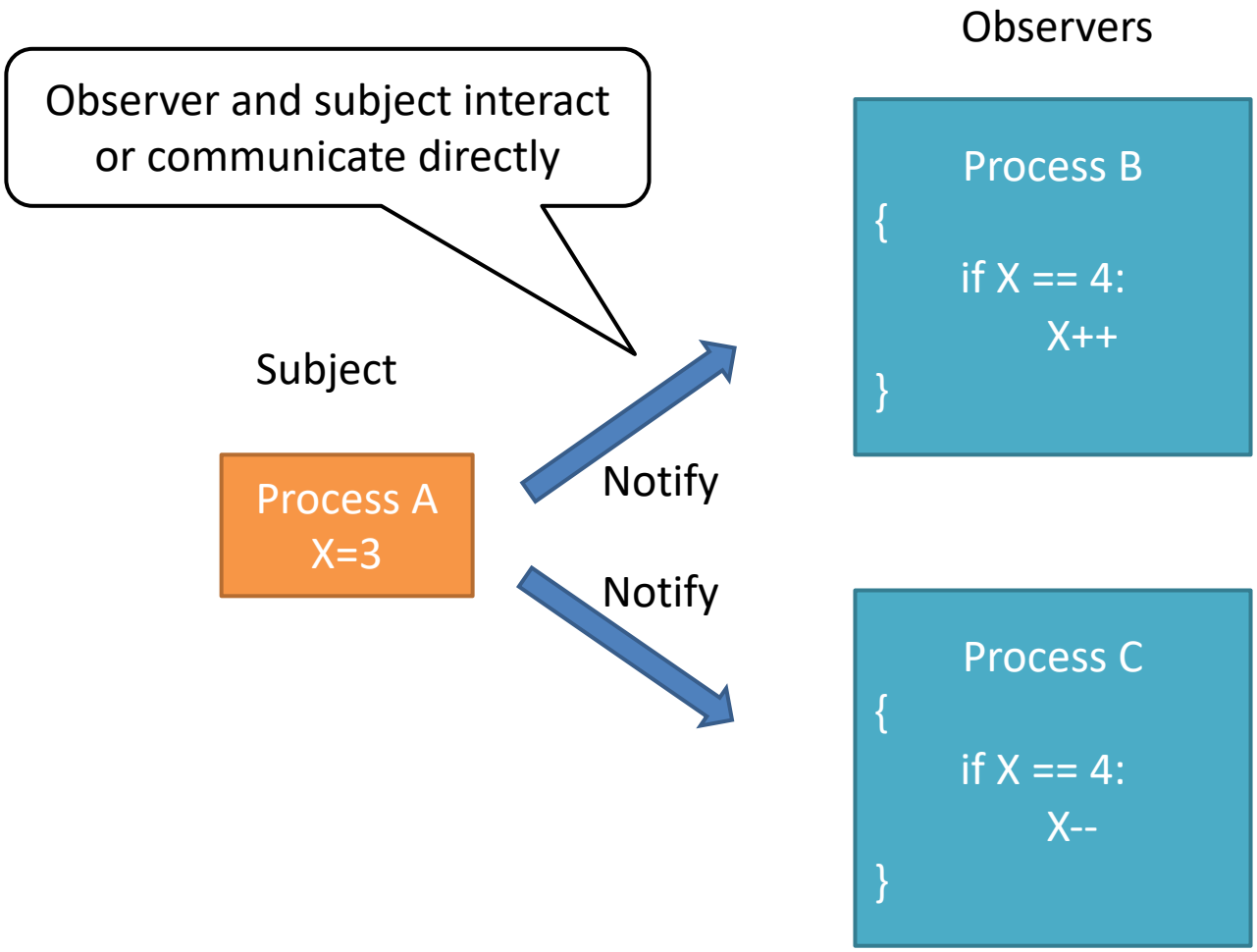
2. Implement asynchronous execution to avoid high load.
   - Producers/publishers publish messages to the message center. When the messages exceed the storage capacity of the message center, the message center will discard the excess messages, so that the system will not cause failure due to the large number of messages.

Observer Pattern/model

- The observer pattern/model is a software design pattern in which an object, called the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods.



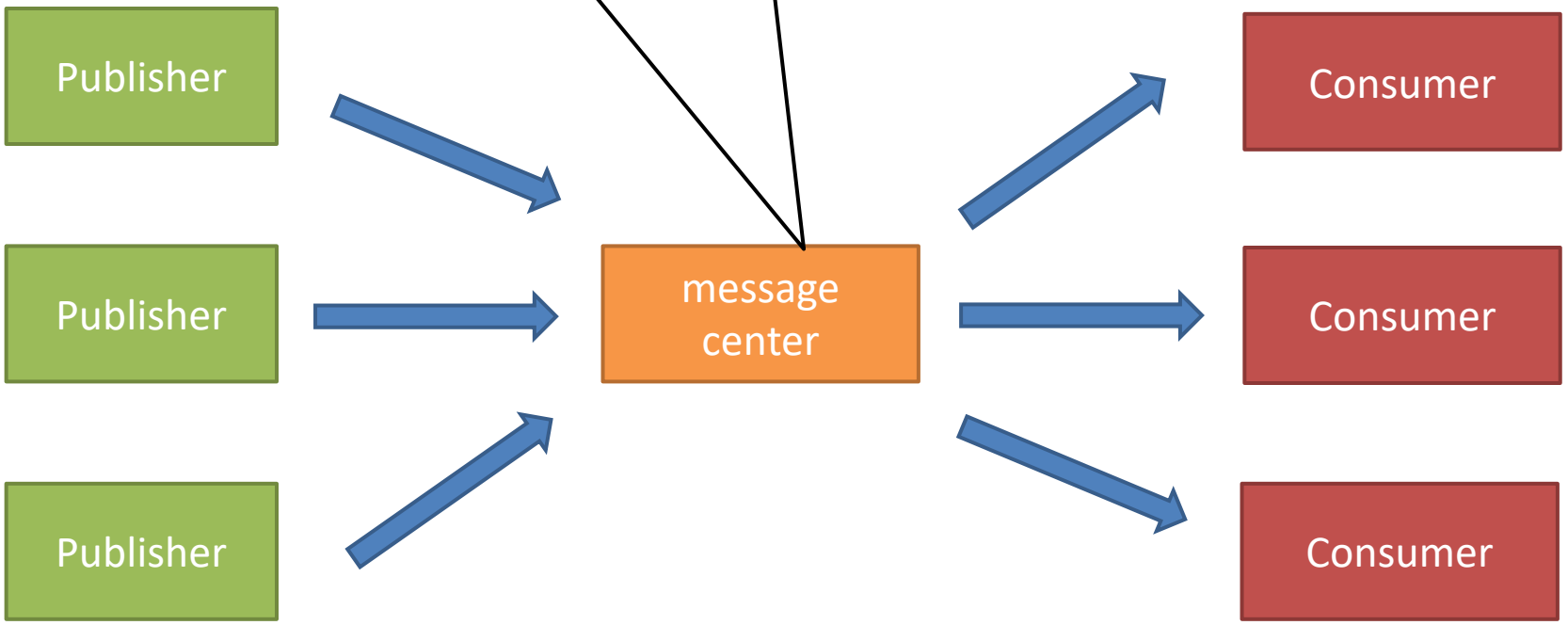
Example of Observer Pattern/model



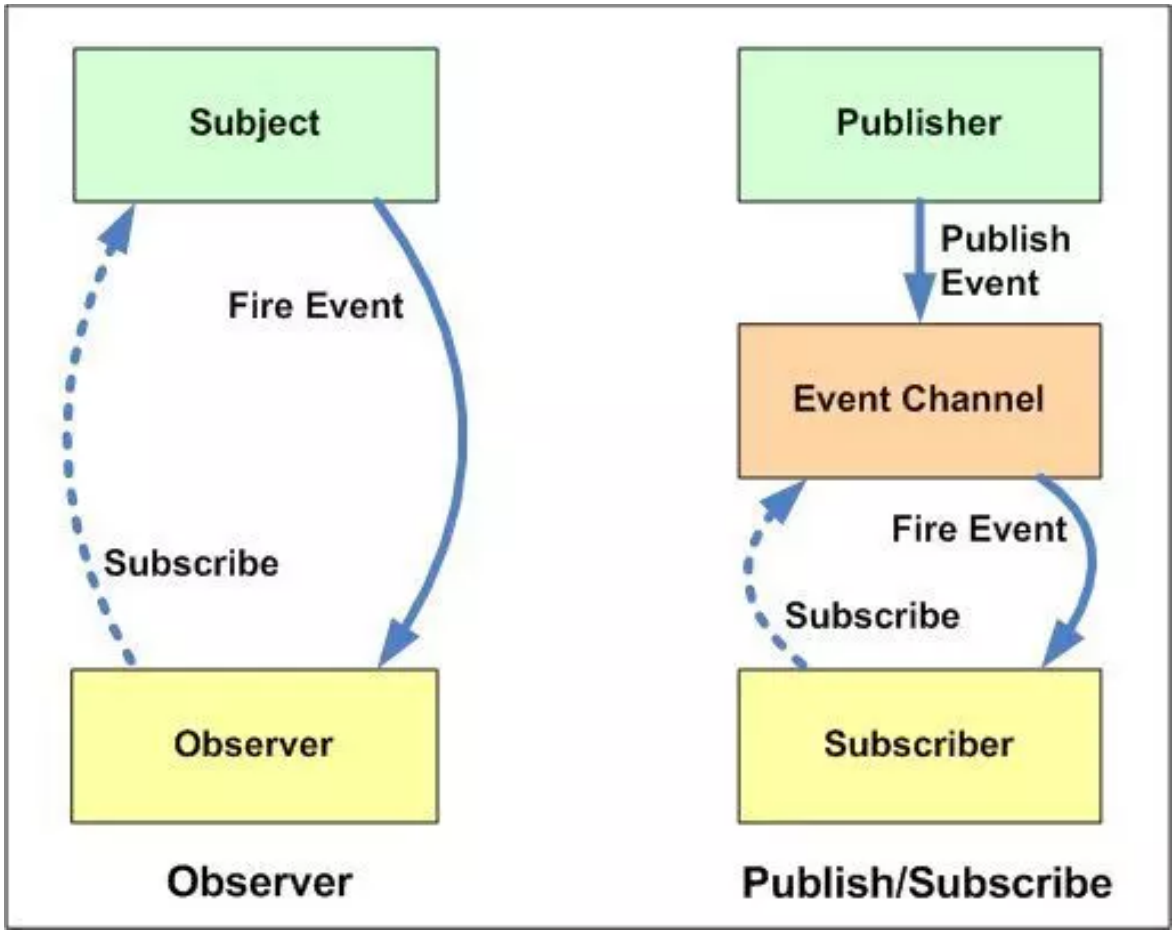
Publish/Subscribe Model

Publisher and consumer interact or communicate indirectly

Introduce more latency but decouple the system



Difference between Observer Model and Publish/Subscribe Model



Difference between Observer Model and Publish/Subscribe Model

Observer Model	Publish/Subscribe Model
The observer knows the subject, and the subject keeps a record of the observer.	Publishers and subscribers are unaware of each other's existence. They only communicate through the message broker.
Components are coupled	Components are decoupled
The observer mode is mostly synchronous. For example, when an event is triggered, the subject will call the observer's method	The publish-subscribe model is mostly asynchronous (using message queues)
The observer pattern is usually implemented in a single application	The publish-subscribe model is widely used in distributed systems

Conclusion

- RPC problems and what is pub/sub model
- P2P model vs. pub/sub model
- An example of pub/sub model: Kafka
- Design issues in pub/sub model
- Observer Model vs. Publish/Subscribe Model