Kafka

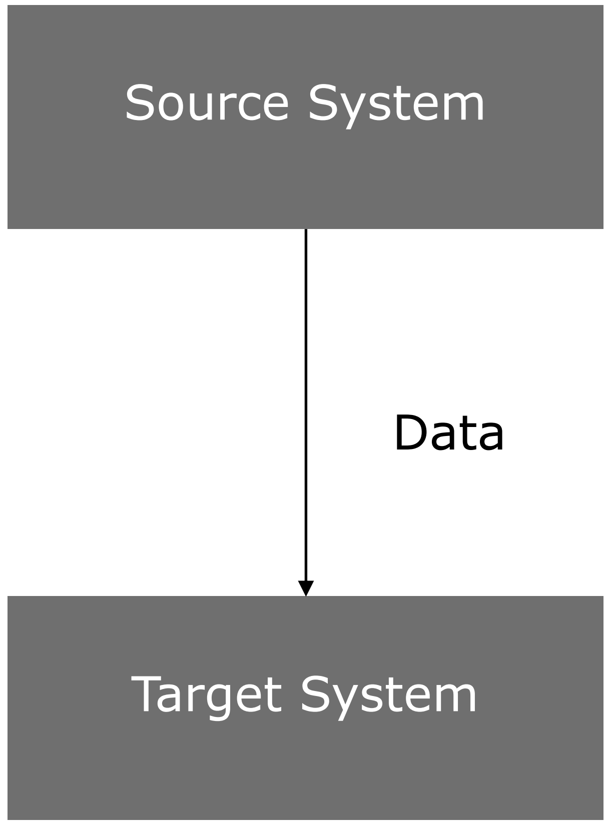
Apache Kafka

Document version 1.0

Messaging and Why do we need it?

Let’s try to understand…

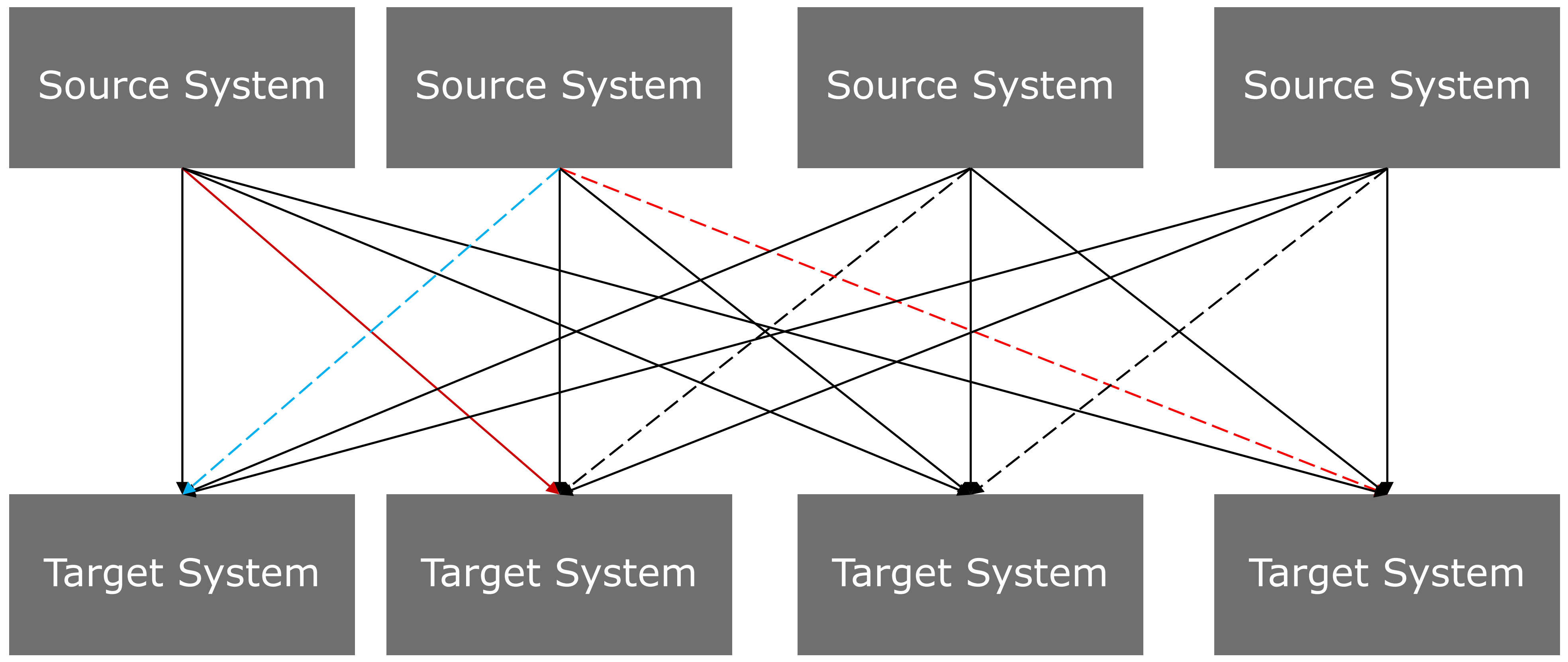
Data Exchange between two applications



Very Simple ☺

But, If we need to make this communication among many systems, what happens

Data Exchange between many applications



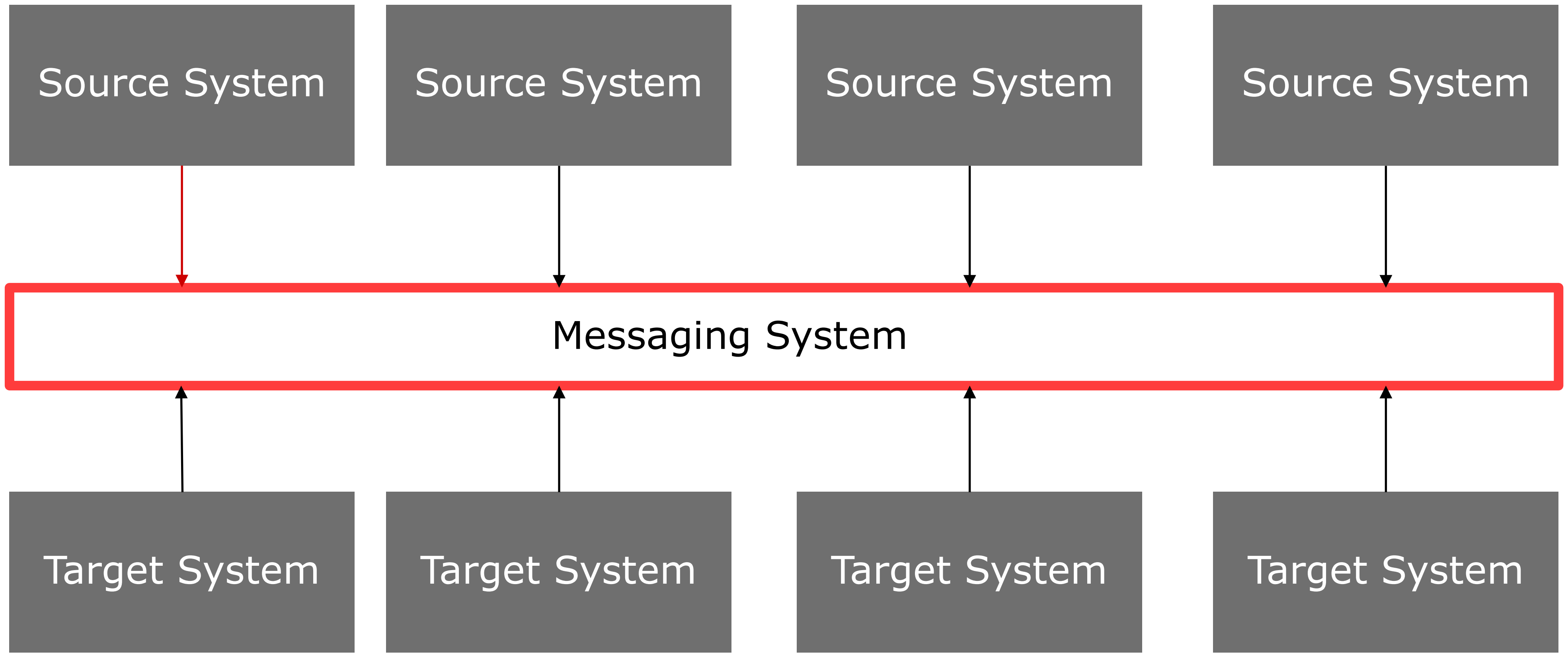
4 source Systems and 4 Target Systems => 16 integrations (!!!)   Not Good

Disadvantages of Direct Integration

- The Integration System has to handle a large number of connections (LOAD)

- Each Integration between Systems comes with difficulties
  - Data Format -> How the data is exchanged(text, XML, JSON, Binary etc)
  - Protocol-> Which protocol is used in the communication (http,jdbc,ftp,tcp)
  - Schema Handling-> how the data is shaped and organised.

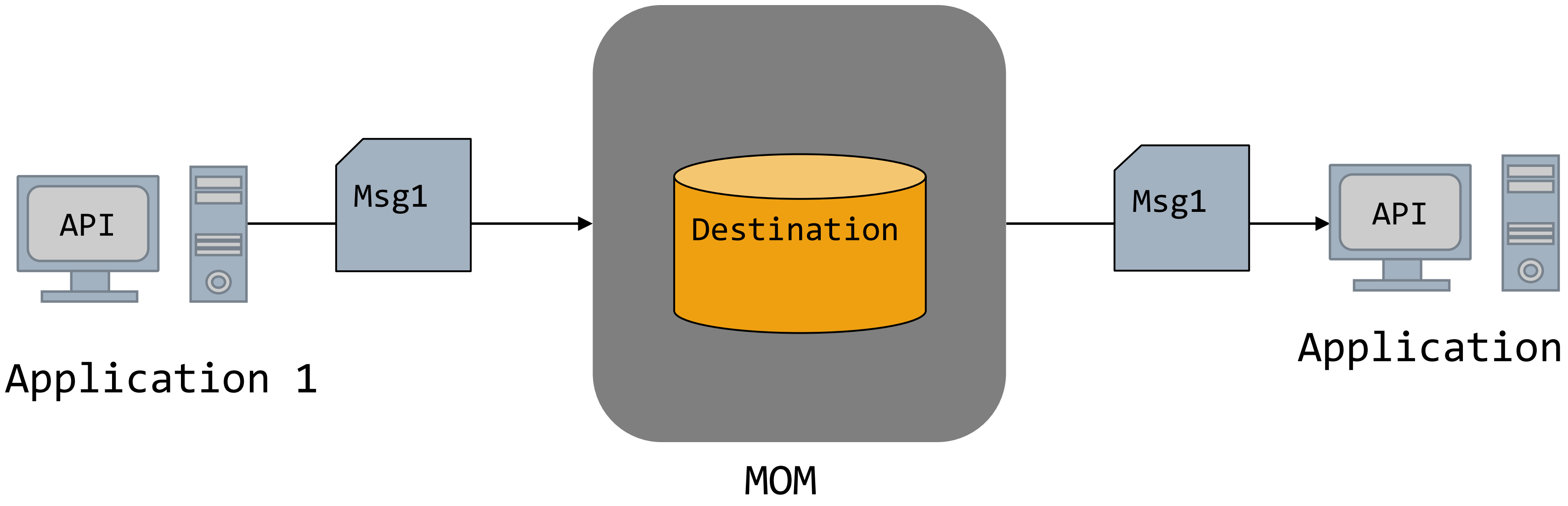
Data Exchange with messaging System



So, what is this messaging system?

Message Oriented Middleware

“Message Oriented Middleware is a concept that involves the passing of data between applications using a communication channel that carries self-contained units of information (messages).”

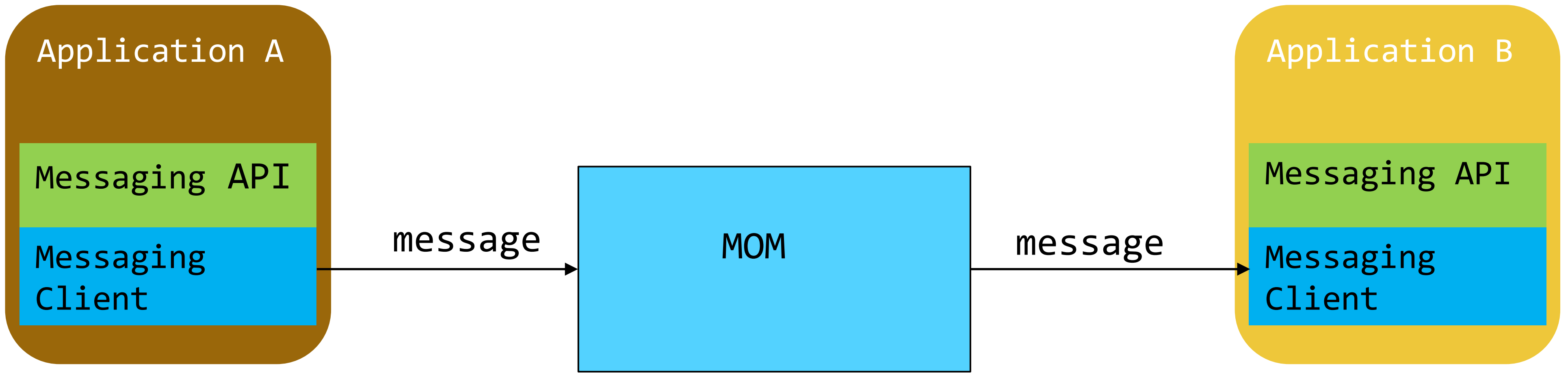


Communication in Message Oriented Middleware

- In a MOM-based communication environment, messages are usually sent and received asynchronously.
- Using message-based communications, applications are abstractly decoupled
  - senders and receivers are never aware of each other.
  - they send and receive messages to and from the messaging system
  - It is the responsibility of the messaging system (MOM) to get the messages to their intended destinations.

Communication in Message Oriented Middleware

- In a messaging system, an application uses an API to communicate through a messaging client that is provided by the MOM vendor.
- The messaging client sends and receives messages through a messaging system



Messaging System

- **The messaging system is the core of a Message Oriented Middleware**
- It is responsible for managing the connection points between multiple messaging clients, and for managing multiple channels of communication between the connection points.
- Usually implemented as a software process, which is commonly known as a message server or a message broker.
- Message servers are usually capable of being grouped together to form clusters
  - Clusters provide advanced capabilities such as load balancing, fault tolerance, and sophisticated routing using managed security domains.

Some Popular Messaging Systems

- ☐ TIBCO Enterprise Message Service
- ☐ IBM Websphere MQ
- ☐ Sonic MQ
- ☐ Apache Active MQ
- ☐ Java Message Service (JMS) brokers

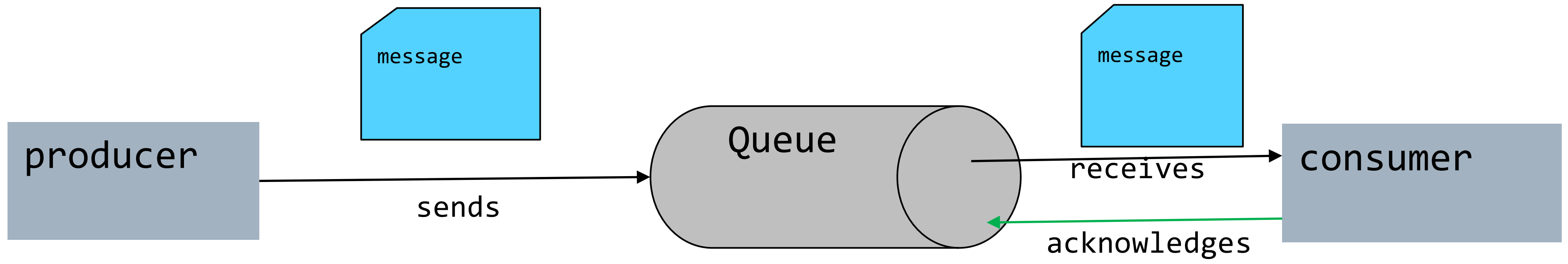
….. and many more

Messaging Models

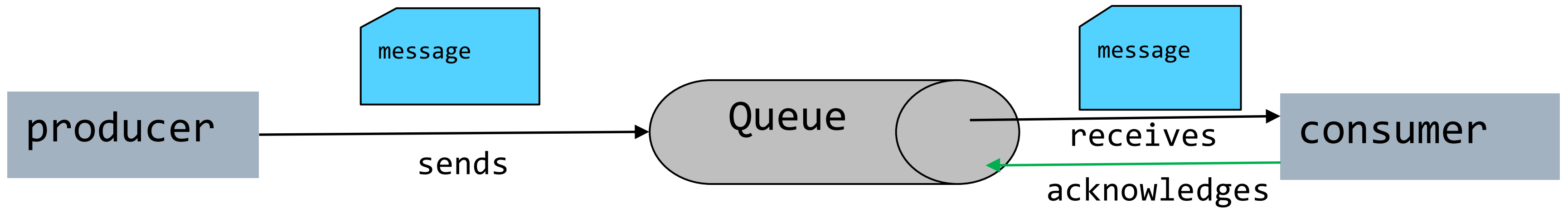
- Two Messaging Models
  - P2P (Point-to-Point)
  - Publish-Subscribe (PubSub messaging)

Point to Point Messaging

- The **point-to-point (P2P) model** is intended for a **one-to-one communication** (Unicast) between two specific applications.
- Based on Message Queue



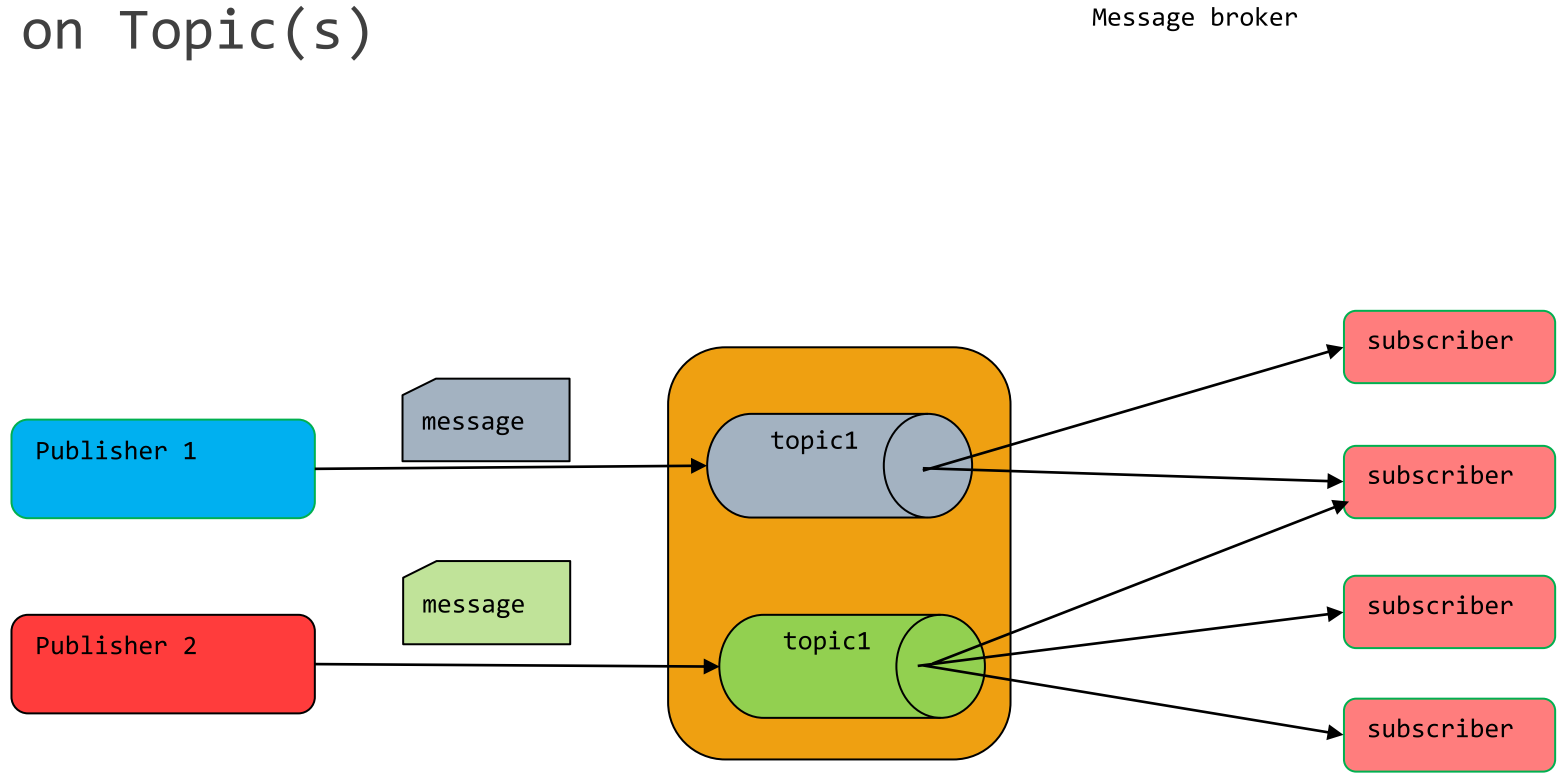
Point to Point Messaging



- In the point-to-point model, **only one consumer may receive a message that is sent to a queue.**
- A point-to-point queue may have multiple consumers listening for the purposes of load-balancing or "hot backup"; however, only one receiver may consume each individual message.
- There may also be no receivers listening, in which case the message stays in the queue until a receiver attaches itself to the queue to retrieve messages.

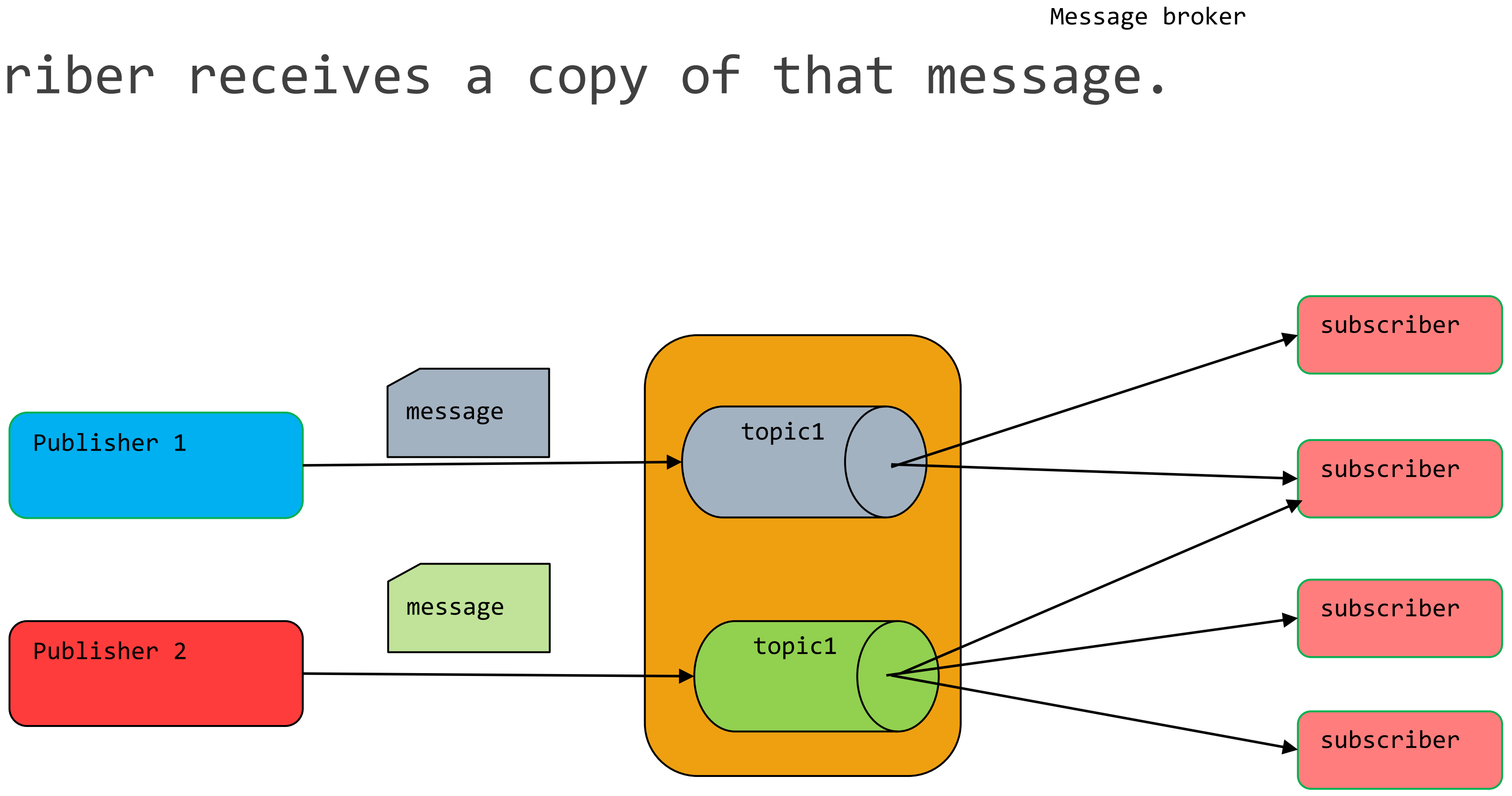
Publish Subscribe Messaging

- The **publish-and-subscribe (pub/sub) model** is intended for a **one-to-many broadcast** (Multicast) of information
- Based on Topic(s)

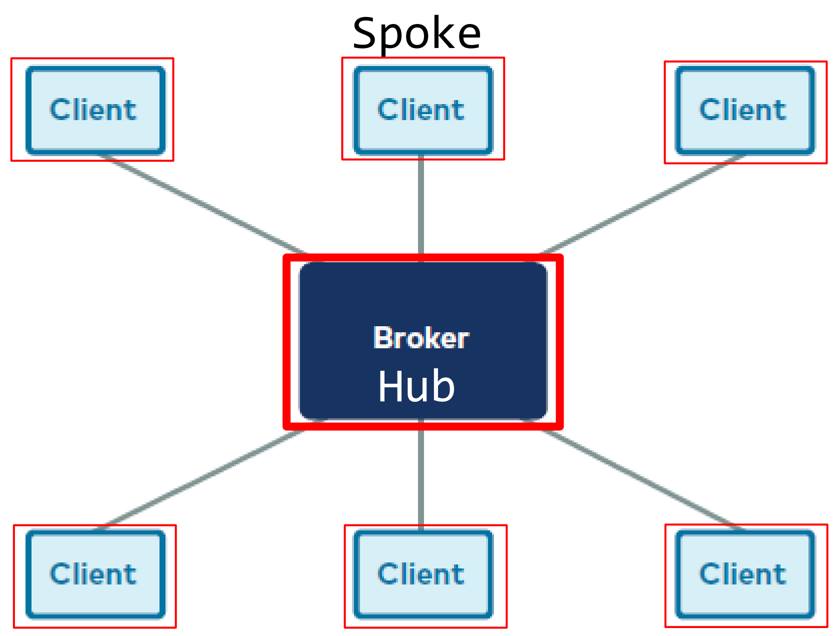


Publish Subscribe Messaging

- In the pub/sub model, **multiple consumers may subscribe to, a topic.**
- A producer sends a message on that channel by publishing on that **topic**.
- Each subscriber receives a copy of that message.



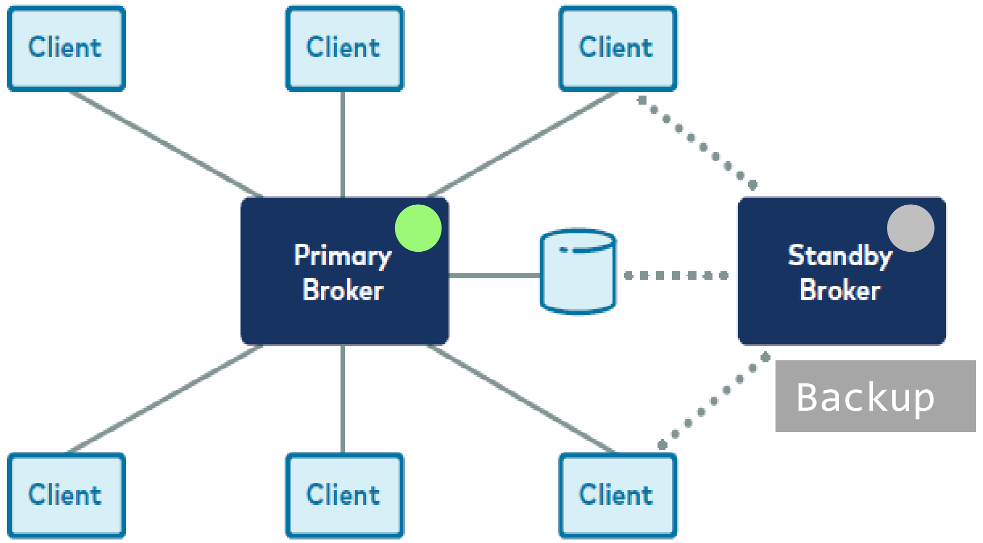
Problems With Traditional Messaging Systems



Hub & Spoke Message Broker

- Traditional Messaging Systems are not designed to be horizontally scalable
- Most traditional messaging started with a hub and spoke design where each client application connects to only one server/broker at a time.
- All the topics and queues reside on this central hub.

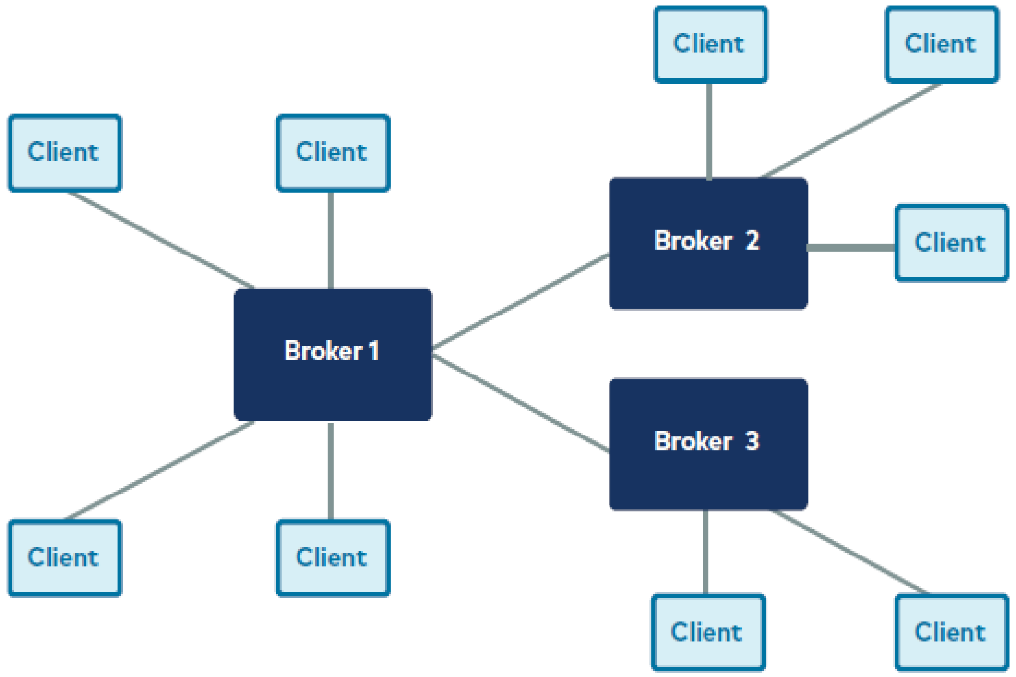
Problems With Traditional Messaging Systems



High Availability Pair of Message Brokers

- Hub and Spoke design has a single point of failure
- Hence, brokers are deployed in High Availability (HA) pairs such
- that the primary (active) broker has a hot backup (standby) which can take over if the primary should fail.
- Even in this architecture, clients only connect to the one active broker where all active topics and queues reside.

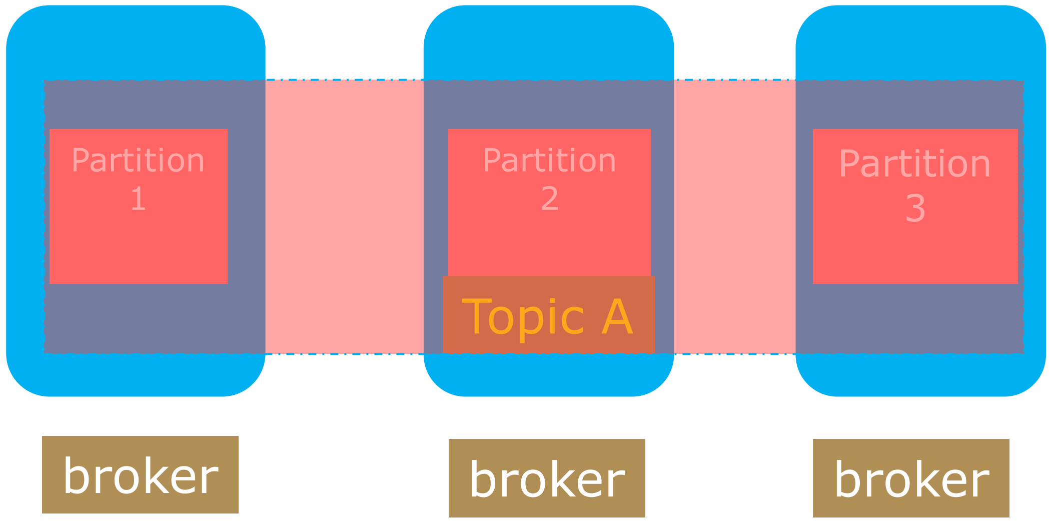
Problems With Traditional Messaging Systems



Multi-Node Network/Cluster of Message Brokers

- Hub/spoke messaging architectures in a more complicated multi-node networks, a single client application still only connects to one broker node at a time, even if the brokers themselves are bridged into a larger multi-node network.
- It does not scale horizontally because as you increase the number of nodes, you also increase the amount of inter-node traffic between the nodes that are processing writes (i.e. handling publishers) versus the nodes that are processing reads (i.e. handling subscribers).

Then What Type of Messaging System do We need?



Horizontal Scalability of Topic A

- Messaging systems where topics and queues do not reside as a monolithic entity on a single node.
- Instead, topics are broken into smaller pieces(partitions), and these topic partitions are spread across multiple nodes in the cluster.
- A client can connect to any single node of the cluster, it should be able to get the metadata it needs to understand how to connect to any and all of the other nodes in the cluster that contain topic partitions of interest for either read or write.

Enter

APACHE kafka®

What is Kafka?

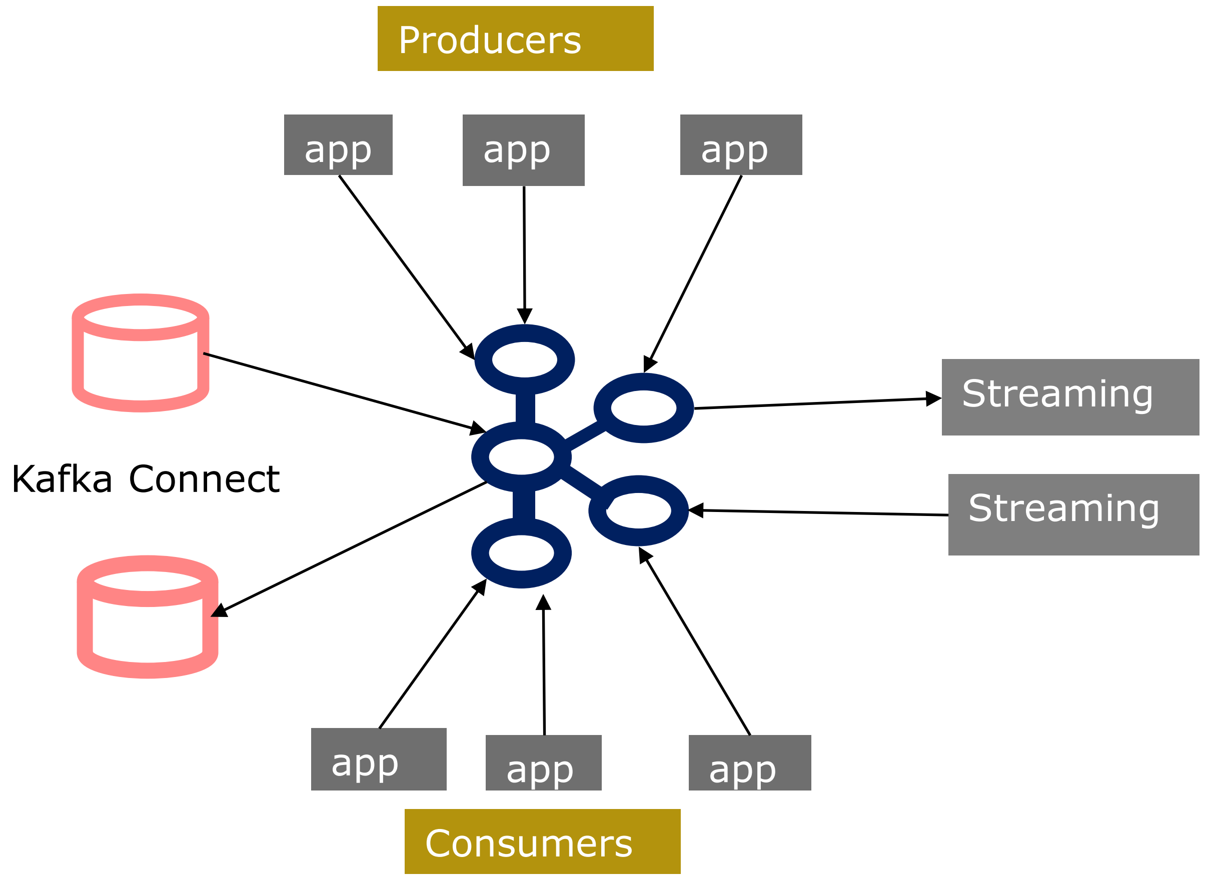
Apache Kafka is an open-source distributed event streaming platform for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications.

--as per https://kafka.apache.org/

What is Kafka?

- Kafka at its core is a message Broker

- It becomes an event Streaming platform in the presence of the supported APIs



Event Streaming Defined

- event streaming is the practice of capturing data in real-time from event sources like databases, sensors, mobile devices, cloud services, and software applications in the form of streams of events.
- storing these event streams durably for later retrieval.
- manipulating, processing, and reacting to the event streams in real-time as well as retrospectively.
- routing the event streams to different destination technologies as needed.
- Event streaming thus ensures a continuous flow and interpretation of data so that the right information is at the right place, at the right time.

Apache Kafka Use Cases

Messaging System

Website Activity Tracking

Gather Metrics from many locations

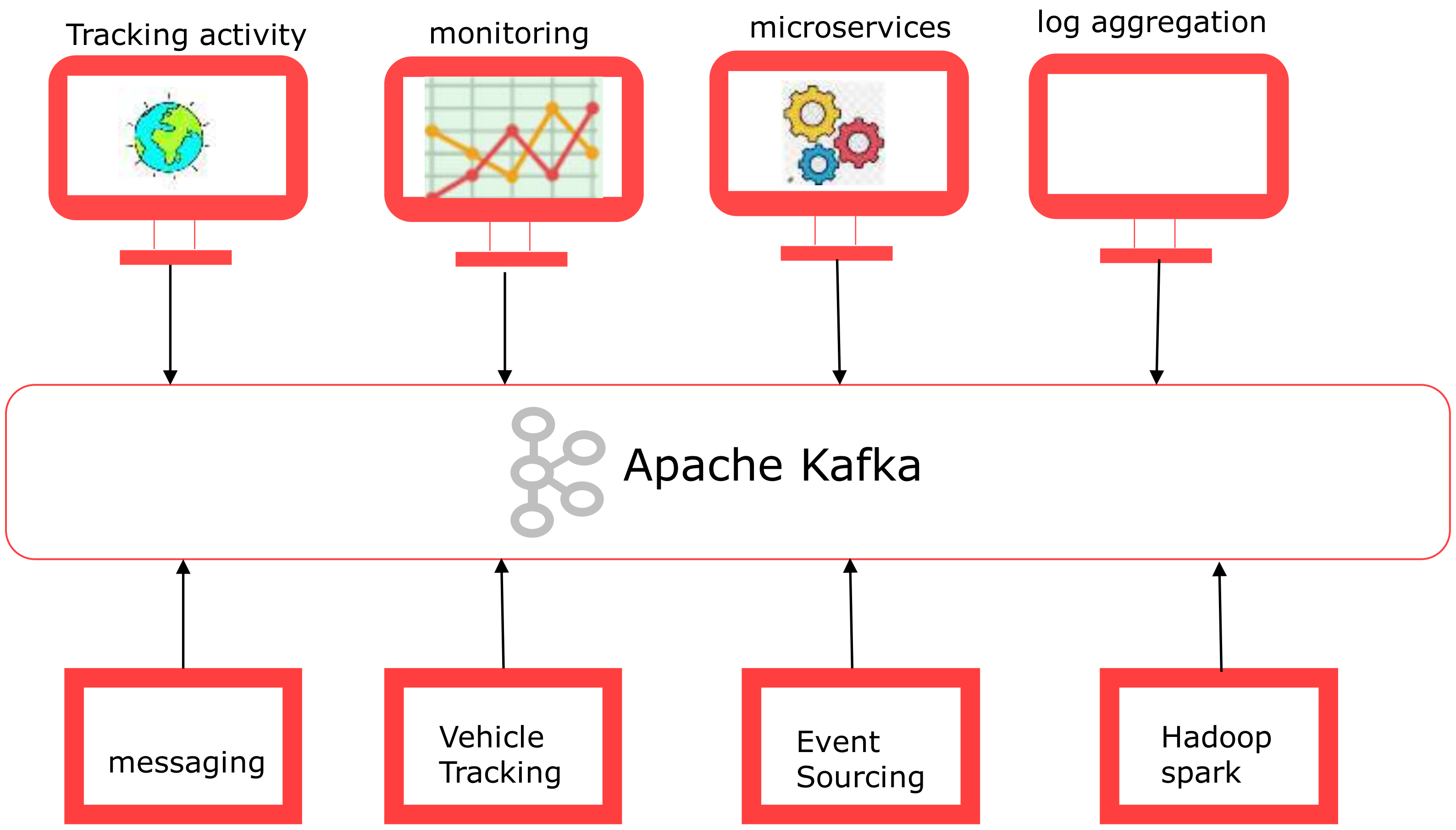
Log Aggregation

Event Processing (with Kafka APIs)

Event Sourcing

Commit Log

Apache Kafka Use Cases



… and many more …..

Kafka Concepts

Topic

Topic Partitions

Message Offset

Broker

Topic Replication

Producer

Consumer

Consumer Group

Zookeeper

Topic

A topic is a stream of data

In Kafka, events are organized and durably stored in **topics**.

A topic is similar to a **folder** in a filesystem, and the events are the files in that folder.

A topic can be thought of as a database table and events/messages can be compared to the rows of data.

A Topic is identified by a unique name

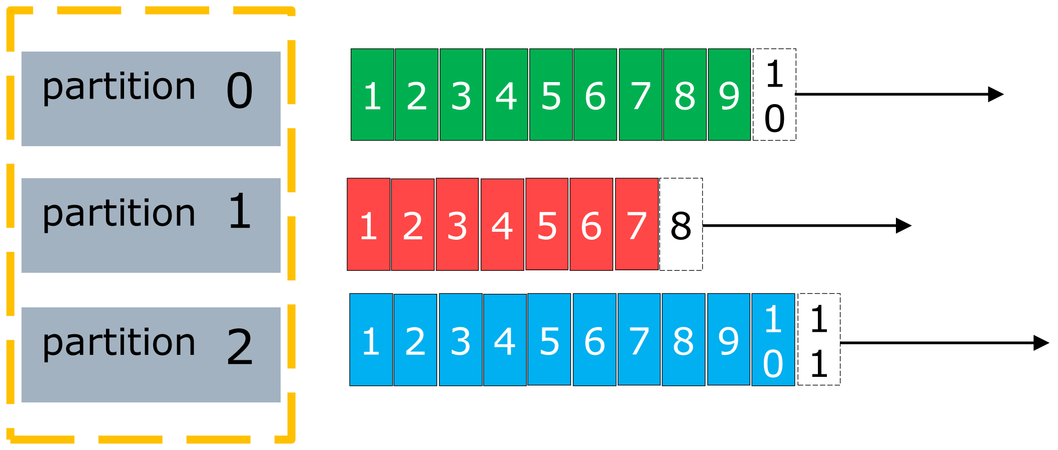
Topic Partitions

Kafka Topics are split into partition

Partition number starts from 0

Each partition is ordered

Each message within a partition gets an incremental id called “offset”



Topic 1

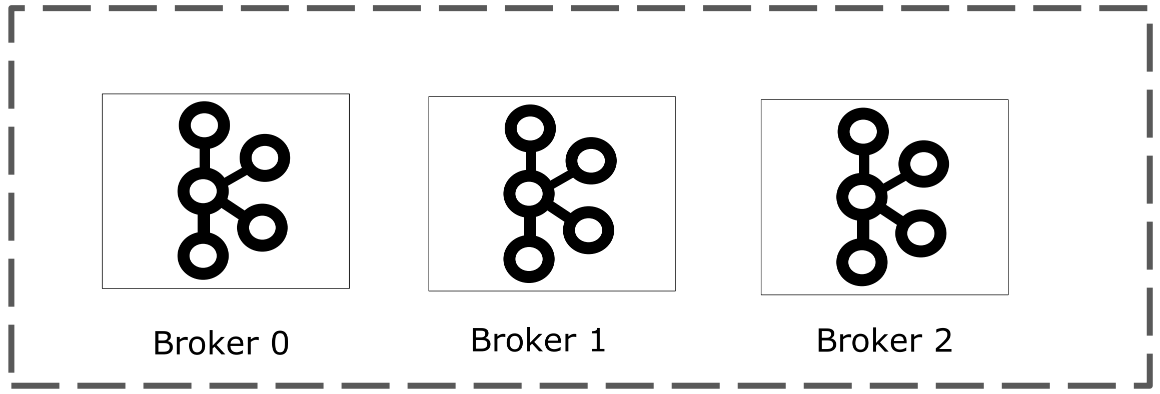
Brokers

Kafka is organized around clusters

A cluster is composed of multiple Brokers

Broker in Kafka is a **server** application which hosts topics

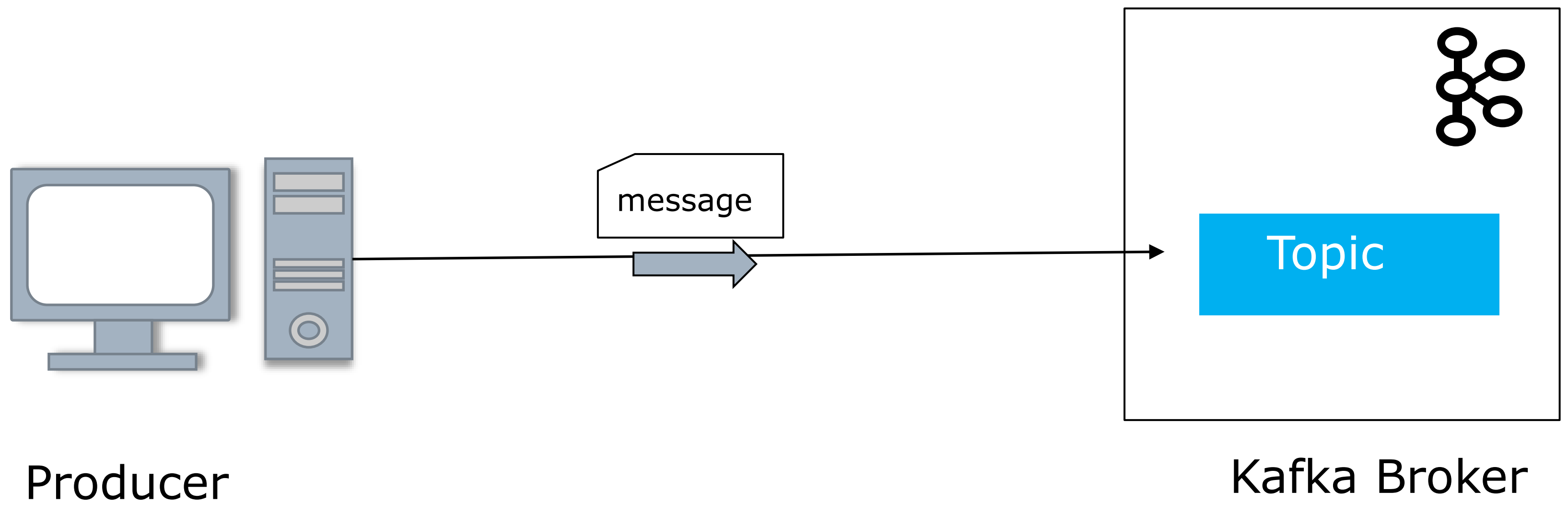
Each Broker is identified by a unique id



Kafka Cluster

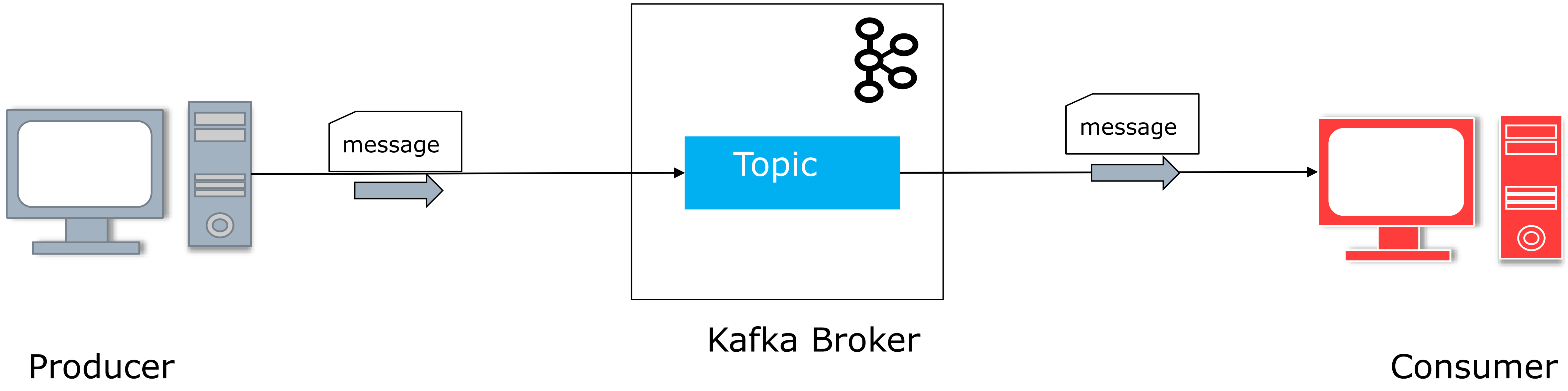
Producer

- A producer is an application that sends data to the broker
- Kafka stores messages a byte array
- Each message is a record
- The producer may send message in any format but it is stored as byte array



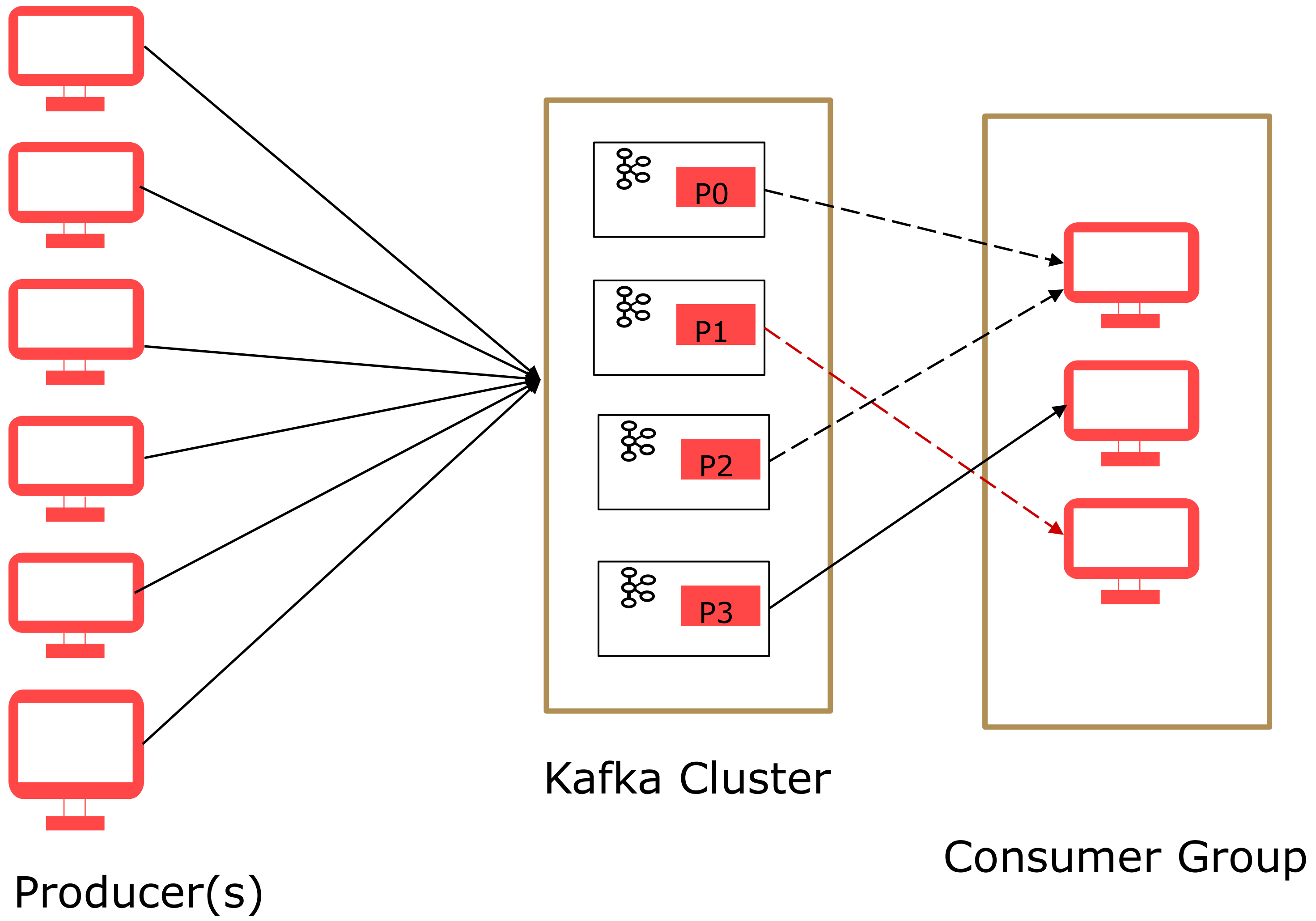
Consumer

- A consumer is an application that reads/receives data from the broker



Consumer Group

- When many producers send messages in volume to a topic, one consumer will not be able to process that data.
- More than one consumer is needed.
- Consumer group is a Group of consumers to share the work load


P0
P1
P2
P3
Kafka Cluster
Consumer Group
Producer(s)