CS 7172
Parallel and Distributed Computation

Message Queue

Kun Suo

Computer Science, Kennesaw State University

https://kevinsuo.github.io/

Outline

- Message queue in OS
- What is message queue in distributed system
- Example of message queue: RocketMQ
- Pub/Sub vs. Message queue

Revisit Message Queue in the OS

- A linked list of messages, stored in the kernel and identified by the message queue identifier

- Message Queuing overcomes the disadvantages of
  - less information (signal),
  - only passing unformatted byte streams (pipe), and limited buffer size

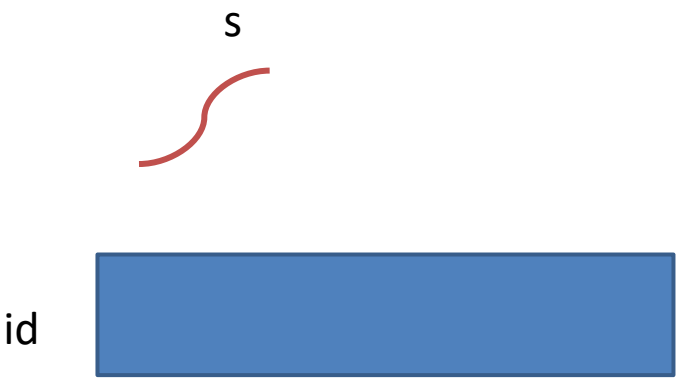
Message queue example

https://github.com/kevinsuo/CS3502/blob/master/msg_queue_sender.c

https://github.com/kevinsuo/CS3502/blob/master/msg_queue_receiver.c

Sender

```c
int main()
{
        int running = 1;
        struct msg_st data;
        char buffer[BUFSIZ];
        int msgid = -1;
        int len;

        /* create a message queue */
        msgid = msgget((key_t)1234, 0666 | IPC_CREAT);
        if(msgid == -1)
        {
                fprintf(stderr, "msgget failed with error: %d\n", errno);
                exit(EXIT_FAILURE);
        }

        while(running)
        {

                printf("Send data : ");
                fgets(buffer, BUFSIZ, stdin);
                data.msg_type = 5;
                strcpy(data.text, buffer);
                len = strlen(data.text);

                /* Send message to msg queue */
                if(msgsnd(msgid, (void*)&data, len-1, 0) == -1)
                {
                        fprintf(stderr, "msgsnd failed\n");
                        exit(EXIT_FAILURE);
                }

                if(strncmp(buffer, "end", 3) == 0)
                        running = 0;
                usleep(100000);
        }
        exit(EXIT_SUCCESS);
}
```



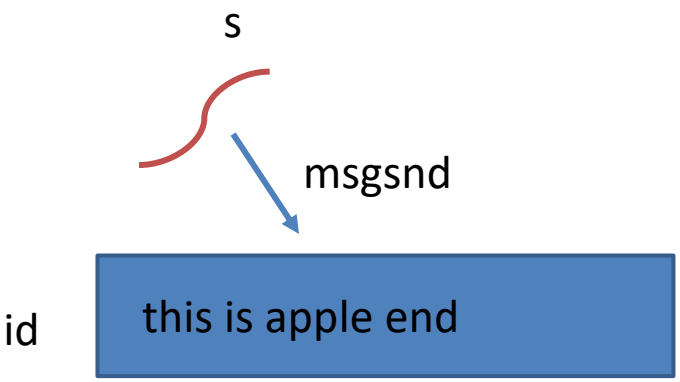
Message queue example

https://github.com/kevinsuo/CS3502/blob/master/msg_queue_sender.c

https://github.com/kevinsuo/CS3502/blob/master/msg_queue_receiver.c

Sender

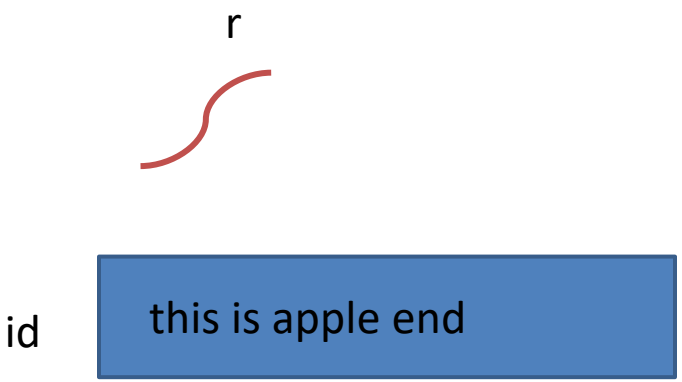
```c
int main()
{
        int running = 1;
        struct msg_st data;
        char buffer[BUFSIZ];
        int msgid = -1;
        int len;

        /* create a message queue */
        msgid = msgget((key_t)1234, 0666 | IPC_CREAT);
        if(msgid == -1)
        {
                fprintf(stderr, "msgget failed with error: %d\n", errno);
                exit(EXIT_FAILURE);
        }

        while(running)
        {

                printf("Send data : ");
                fgets(buffer, BUFSIZ, stdin);
                data.msg_type = 5;
                strcpy(data.text, buffer);
                len = strlen(data.text);

                /* Send message to msg queue */
                if(msgsnd(msgid, (void*)&data, len-1, 0) == -1)
                {
                        fprintf(stderr, "msgsnd failed\n");
                        exit(EXIT_FAILURE);
                }

                if(strncmp(buffer, "end", 3) == 0)
                        running = 0;
                usleep(100000);
        }
        exit(EXIT_SUCCESS);
}
```



https://github.com/kevinsuo/CS3502/blob/master/msg_queue_sender.c

https://github.com/kevinsuo/CS3502/blob/master/msg_queue_receiver.c

Message queue example



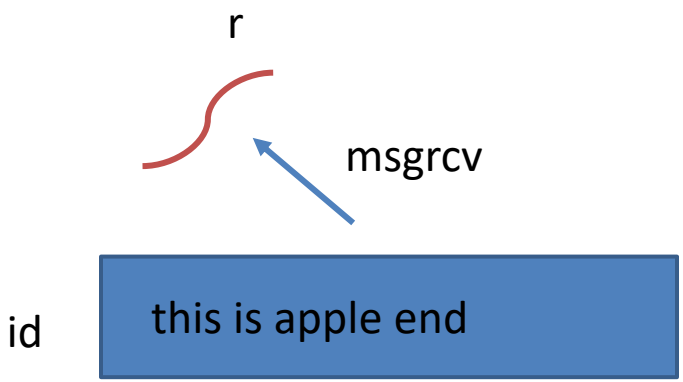
Receiver

```c
int main()
{
        int running = 1;
        int msgid = -1;
        int len = 0;
        long int msgtype = 5;
        struct msg_st data;

        /* create a message queue */
        msgid = msgget((key_t)1234, 0666 | IPC_CREAT);
        if(-1 == msgid )
        {
                fprintf(stderr, "msgget failed with error: %d\n", errno);
                exit(EXIT_FAILURE);
        }

        while(running)
        {
                memset(&data.text, 0, 128);
                /* receive message from msg queue */
                len = msgrcv(msgid, (void*)&data, 128, msgtype, 0);
                if(-1 == len)
                {
                        fprintf(stderr, "msgrcv failed with errno: %d\n", errno);
                        exit(EXIT_FAILURE);
                }
                printf("Receive: %s\n",data.text);

                if(0 == strncmp(data.text, "end", 3))
                        running = 0;
        }

        /* remove message queue */
        if(-1 == msgctl(msgid, IPC_RMID, 0))
        {
                fprintf(stderr, "msgctl(IPC_RMID) failed\n");
                exit(EXIT_FAILURE);
        }
        exit(EXIT_SUCCESS);
}
```

https://github.com/kevinsuo/CS3502/blob/master/msg_queue_sender.c

https://github.com/kevinsuo/CS3502/blob/master/msg_queue_receiver.c

Message queue example



Receiver

```c
int main()
{
        int running = 1;
        int msgid = -1;
        int len = 0;
        long int msgtype = 5;
        struct msg_st data;

        /* create a message queue */
        msgid = msgget((key_t)1234, 0666 | IPC_CREAT);
        if(-1 == msgid )
        {
                fprintf(stderr, "msgget failed with error: %d\n", errno);
                exit(EXIT_FAILURE);
        }

        while(running)
        {
                memset(&data.text, 0, 128);
                /* receive message from msg queue */
                len = msgrcv(msgid, (void*)&data, 128, msgtype, 0);
                if(-1 == len)
                {
                        fprintf(stderr, "msgrcv failed with errno: %d\n", errno);
                        exit(EXIT_FAILURE);
                }
                printf("Receive: %s\n",data.text);

                if(0 == strncmp(data.text, "end", 3))
                        running = 0;
        }

        /* remove message queue */
        if(-1 == msgctl(msgid, IPC_RMID, 0))
        {
                fprintf(stderr, "msgctl(IPC_RMID) failed\n");
                exit(EXIT_FAILURE);
        }
        exit(EXIT_SUCCESS);
}
```

Message queue example

https://github.com/kevinsuo/CS3502/blob/master/msg_queue_sender.c

https://github.com/kevinsuo/CS3502/blob/master/msg_queue_receiver.c

id

this is apple end

this is apple end

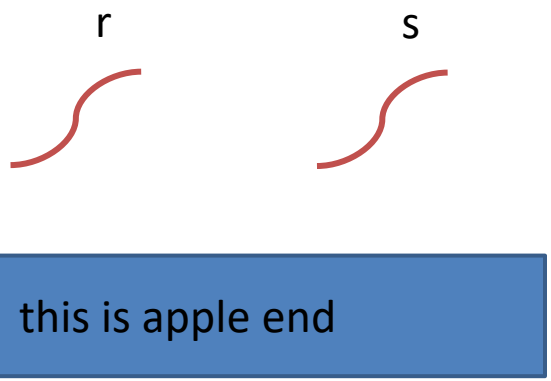
Receiver

```c
int main()
{
        int running = 1;
        int msgid = -1;
        int len = 0;
        long int msgtype = 5;
        struct msg_st data;

        /* create a message queue */
        msgid = msgget((key_t)1234, 0666 | IPC_CREAT);
        if(-1 == msgid )
        {
                fprintf(stderr, "msgget failed with error: %d\n", errno);
                exit(EXIT_FAILURE);
        }

        while(running)
        {
                memset(&data.text, 0, 128);
                /* receive message from msg queue */
                len = msgrcv(msgid, (void*)&data, 128, msgtype, 0);
                if(-1 == len)
                {
                        fprintf(stderr, "msgrcv failed with errno: %d\n", errno);
                        exit(EXIT_FAILURE);
                }
                printf("Receive: %s\n",data.text);

                if(0 == strncmp(data.text, "end", 3))
                        running = 0;
        }

        /* remove message queue */
        if(-1 == msgctl(msgid, IPC_RMID, 0))
        {
                fprintf(stderr, "msgctl(IPC_RMID) failed\n");
                exit(EXIT_FAILURE);
        }
        exit(EXIT_SUCCESS);
}
```

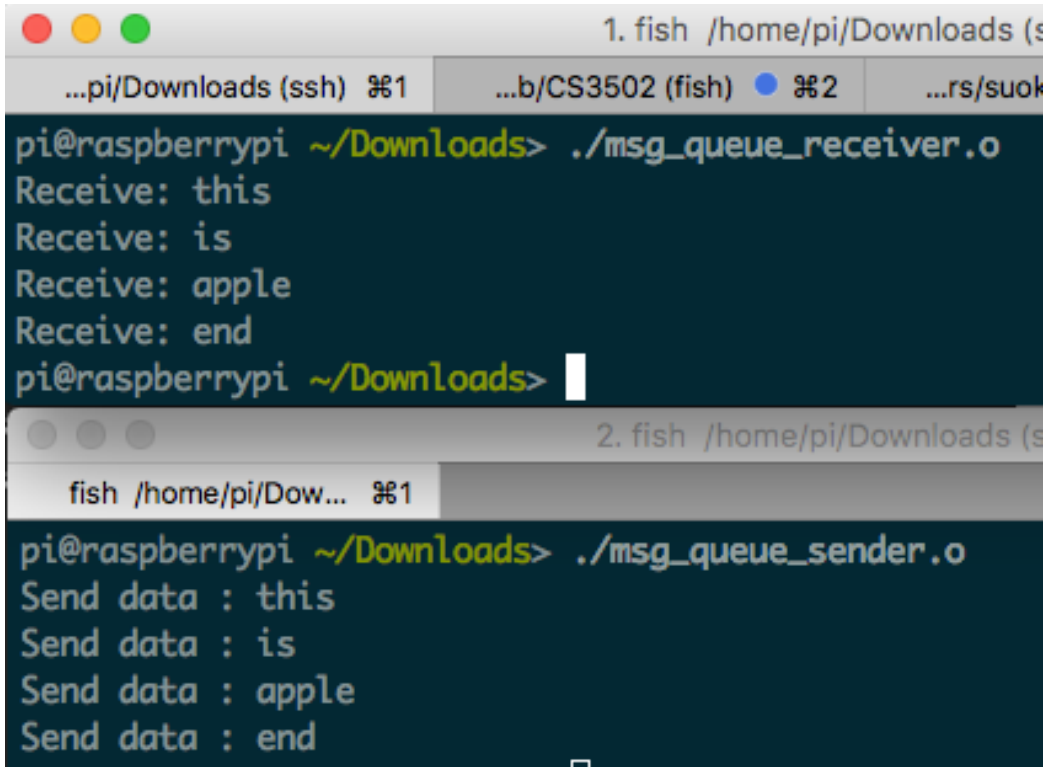
Message queue example

https://github.com/kevinsuo/CS3502/blob/master/msg_queue_sender.c

https://github.com/kevinsuo/CS3502/blob/master/msg_queue_receiver.c

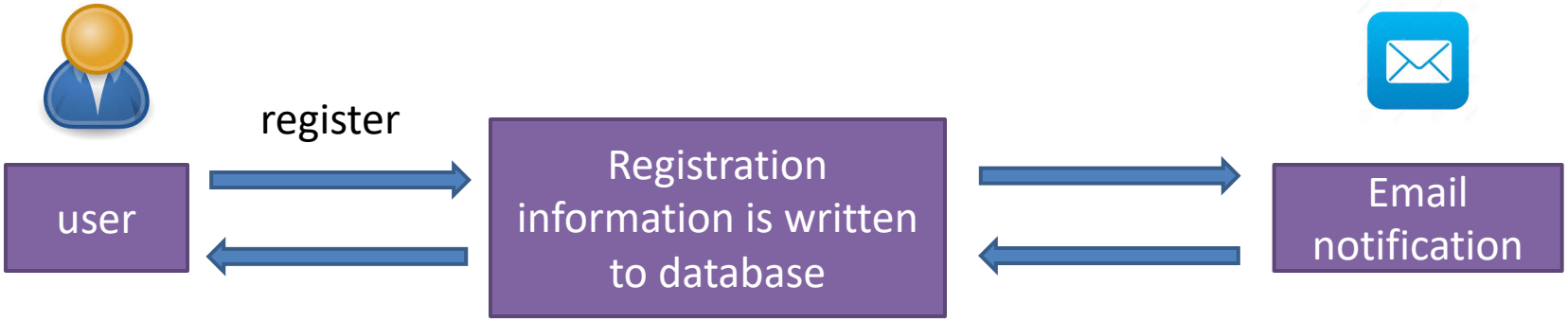


this is apple end



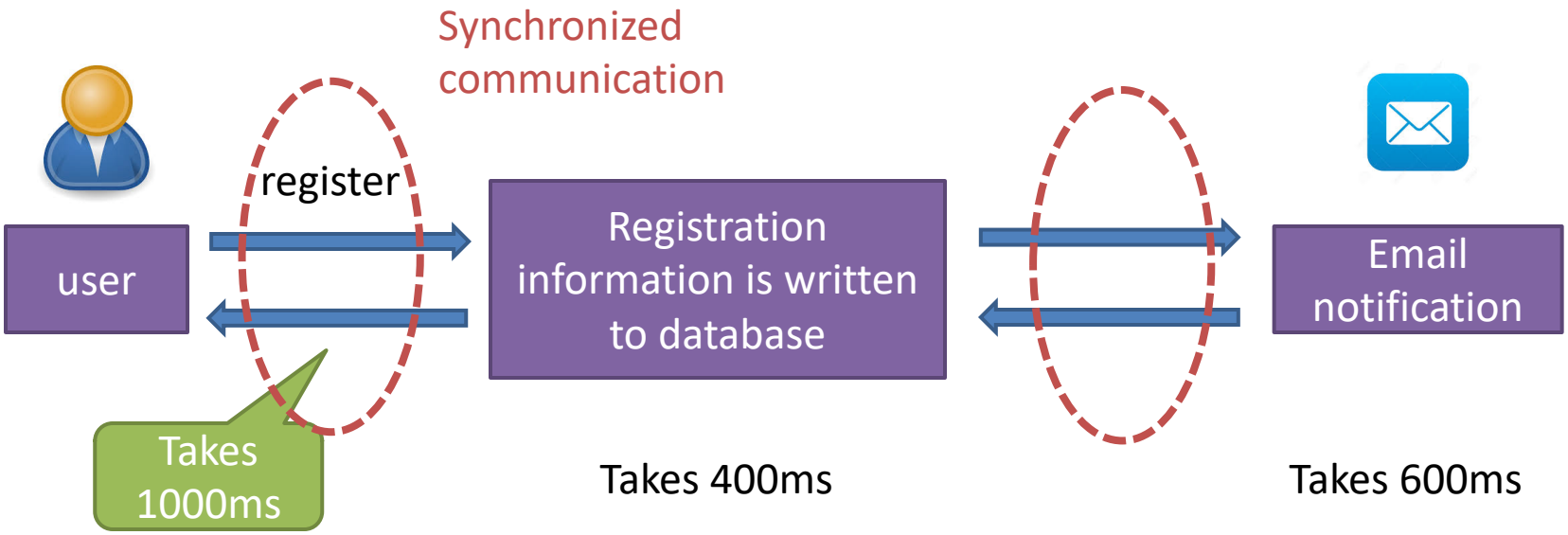
What is Message Queue in Distributed System?

- Example: user use their email to register on a website
  1. Check user information is valid or not. If yes, write into database; if no, return.
  2. After written to database, system sends email notifications to end users



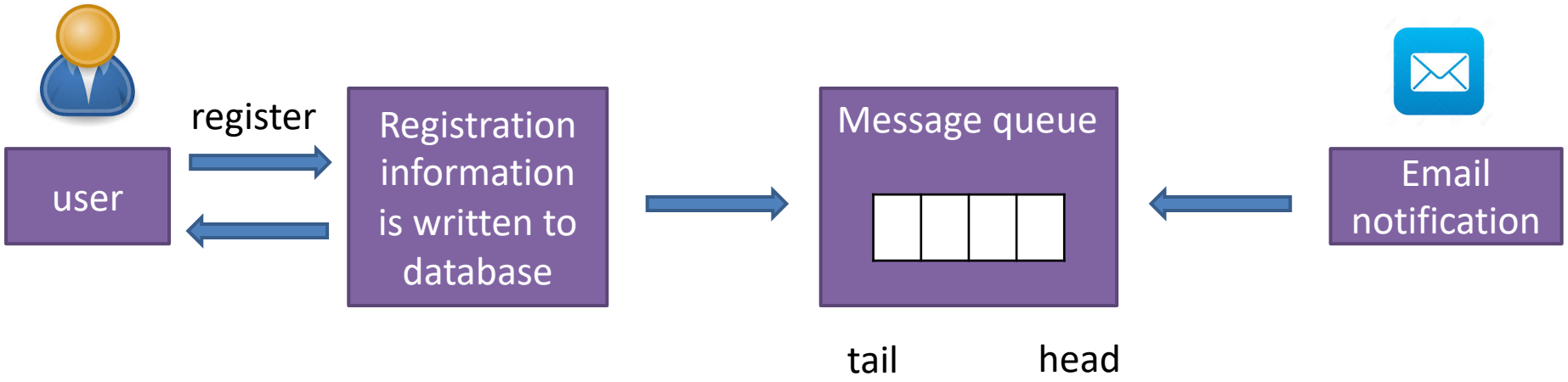
What is Message Queue?

- Example: user use their email to register on a website
  1. Check user information is valid or not. If yes, write into database; if no, return.
  2. After written to database, system sends email notifications to end users



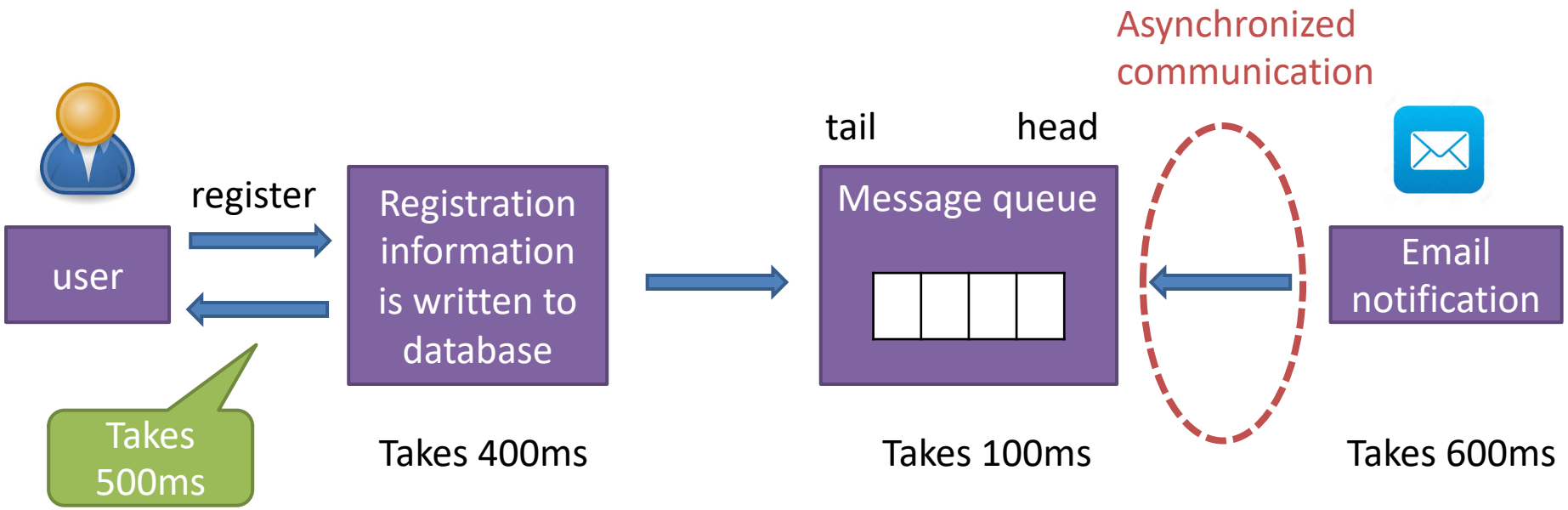
What is Message Queue?

- When using message queue:
  1. Check user information is valid or not. If yes, write into database; if no, return.
  2. After written to database, write message to the end of message queue
  3. System gets message from queue, then sends email notifications to end users

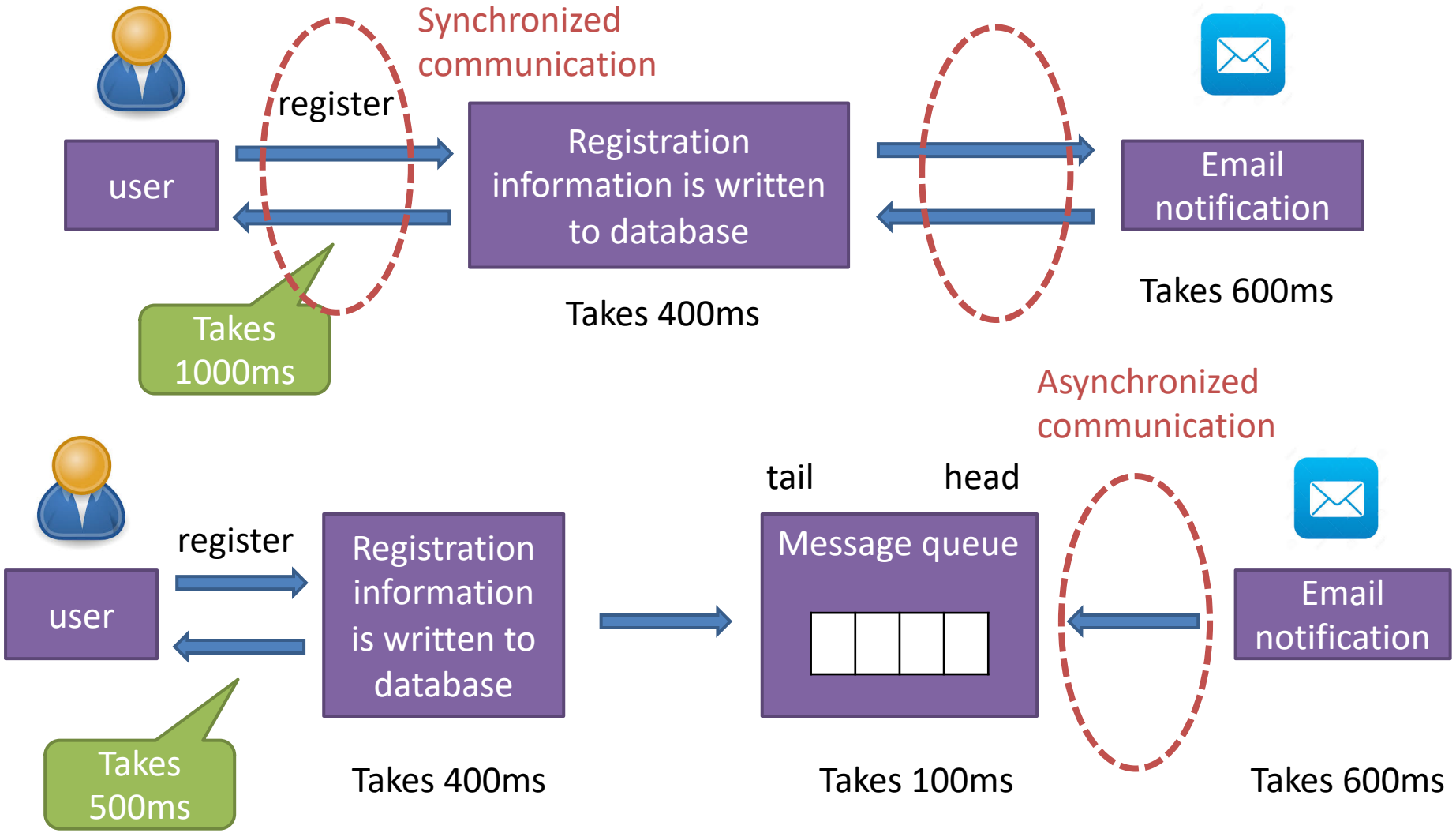


What is Message Queue?

- When using message queue:
  1. Check user information is valid or not. If yes, write into database; if no, return.
  2. After written to database, write message to the end of message queue
  3. System gets message from queue, then sends email notifications to end users



Message Queue can significantly reduce the request latency

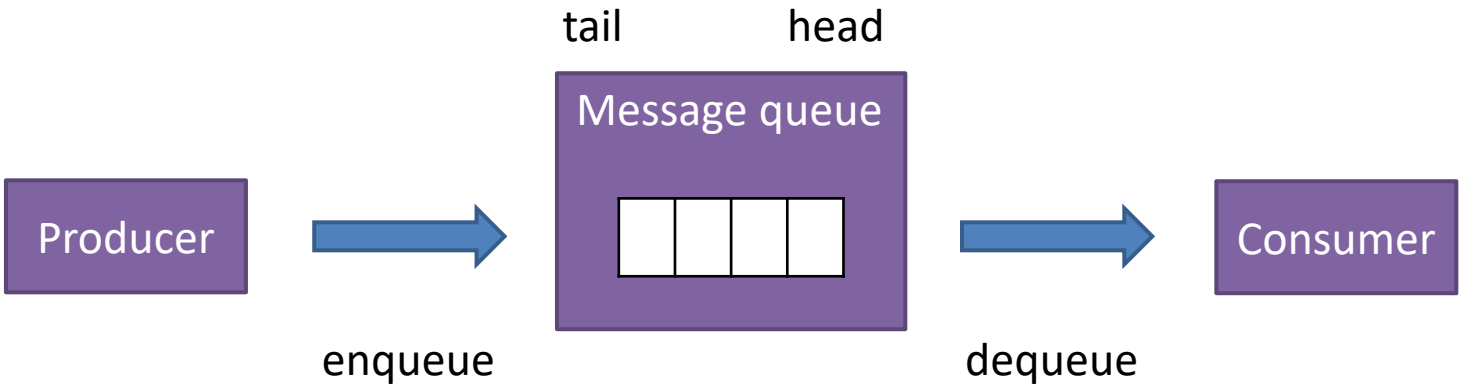


What is Message Queue?

- A message queue is implemented based on a queue. It stores message data in a specific format, such as a structure containing a message type, an ID that uniquely identifies the message, and the content of the message.
- Messages can be returned directly after being placed in the tail of this queue.
- The system does not need to process the message immediately. After that, other processes will read the messages from the head of the queue and process them one by one in the order in which the messages were placed.
- Message queue can improve response speed and decouple between components inside the system.

How Message Queue Works?

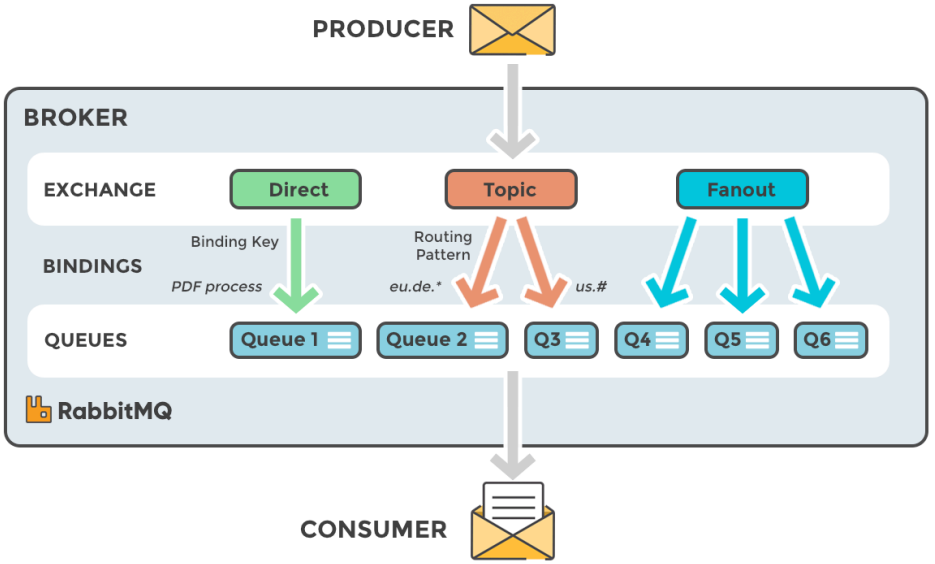
- The message queue model includes three core parts:
  1. Producer: produce messages or data and insert them into the message queue.
  2. Message queue: a first-in-first-out data structure for storing messages.
  3. Consumer: get messages or data from the message queue for further processing.



Example: RabbitMQ

- RabbitMQ is lightweight and easy to deploy on premises and in the cloud. It supports multiple messaging protocols. RabbitMQ can be deployed in distributed and federated configurations to meet high-scale, high-availability requirements.

- https://www.rabbitmq.com/



Example: RocketMQ

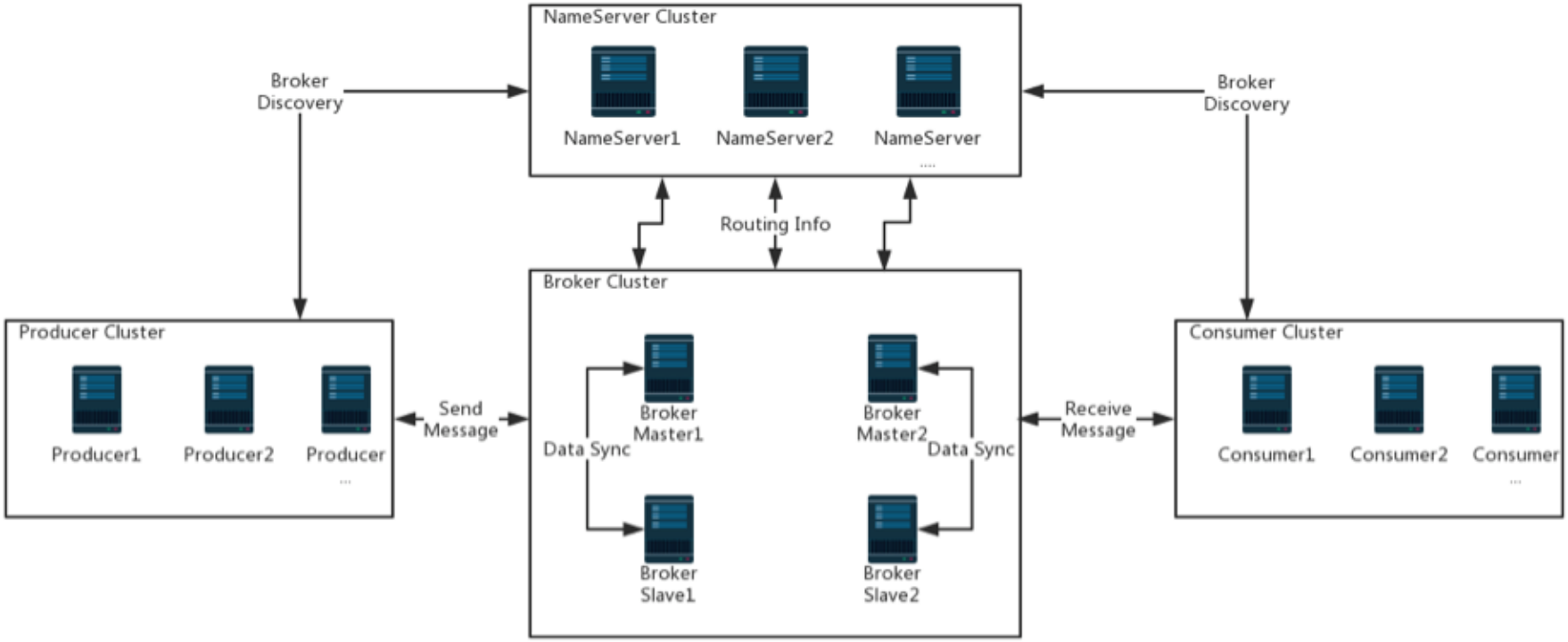
- Apache RocketMQ is a unified messaging engine, lightweight data processing platform.

- A distributed messaging and streaming platform with low latency, high performance and reliability, trillion-level capacity and flexible scalability

- https://rocketmq.apache.org/

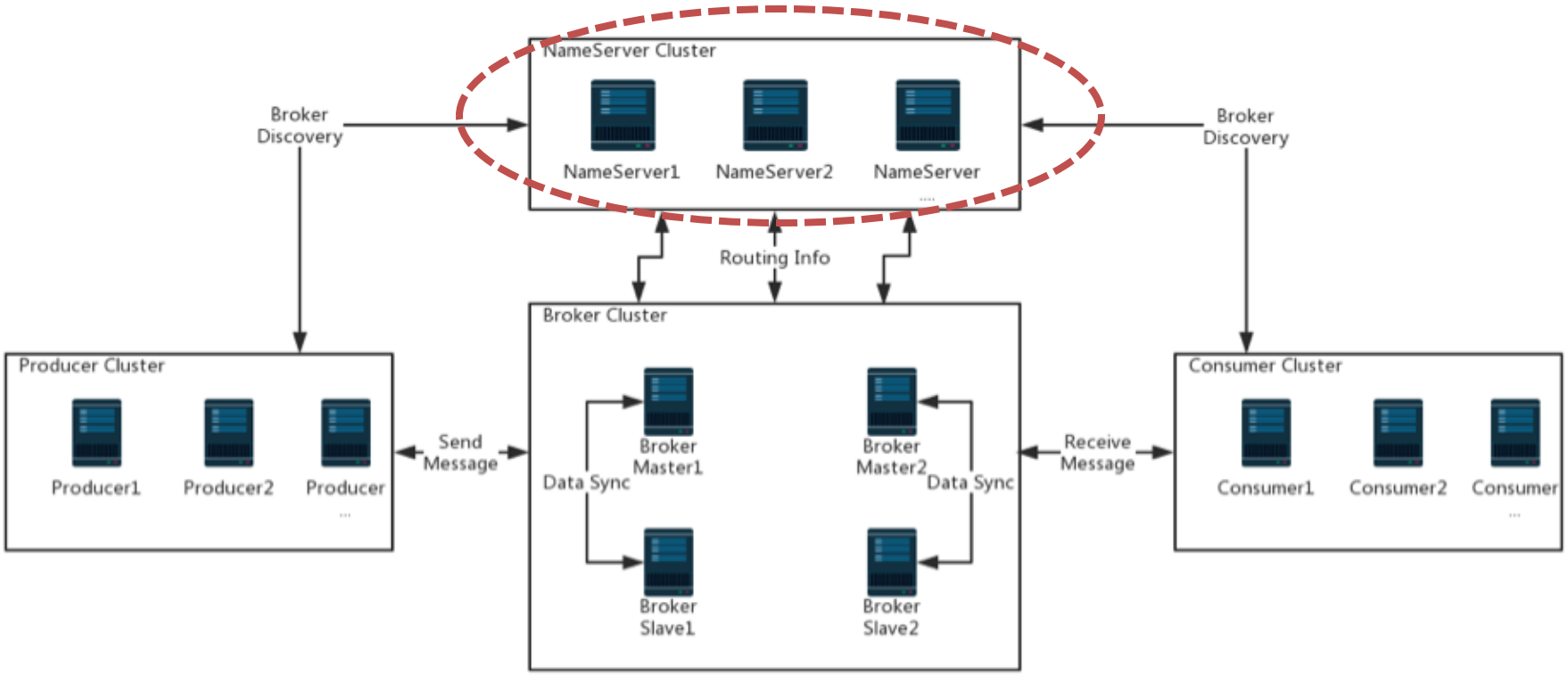
RocketMQ Architecture

- Apache RocketMQ includes 4 parts:
  - NameServer Cluster, Producer Cluster, Broker Cluster and Consumer Cluster



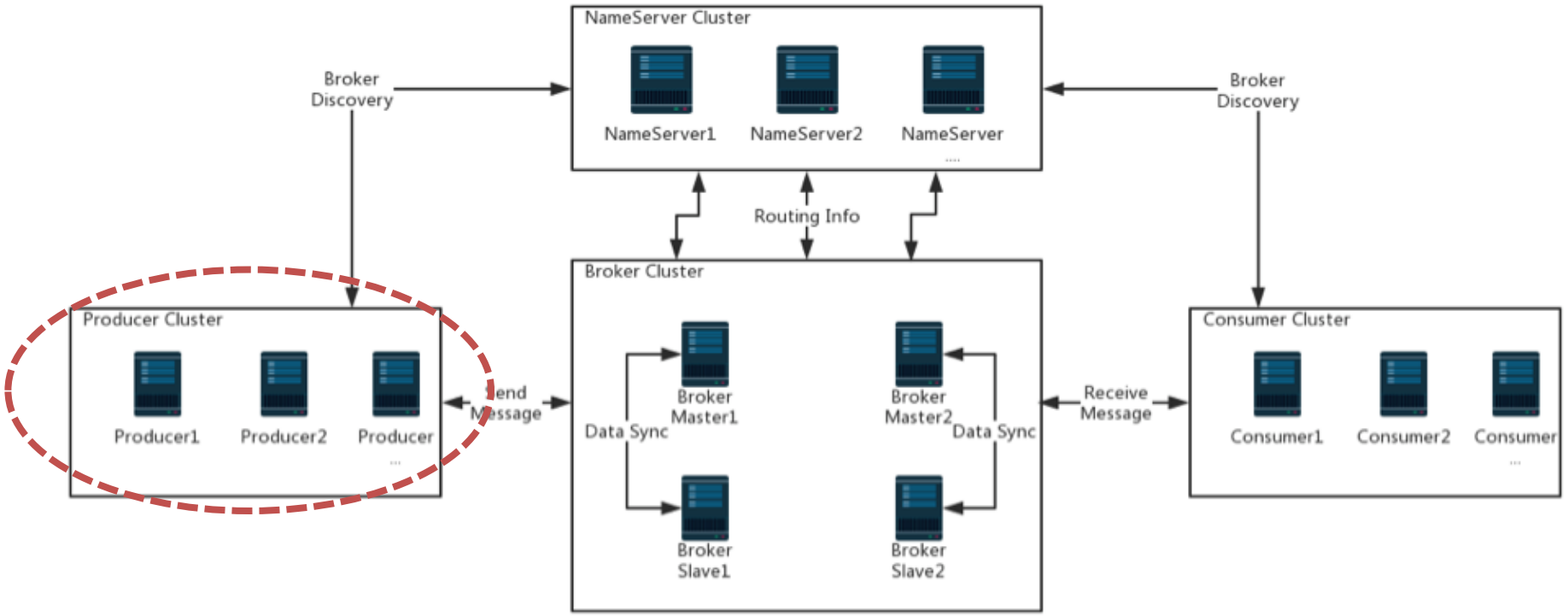
RocketMQ Architecture

- NameServer Cluster:
  - Like ZooKeeper in Kafka, provide lightweight service discovery, routing and management
  - Is responsible for manage information for Broker, such as ID, address, etc.



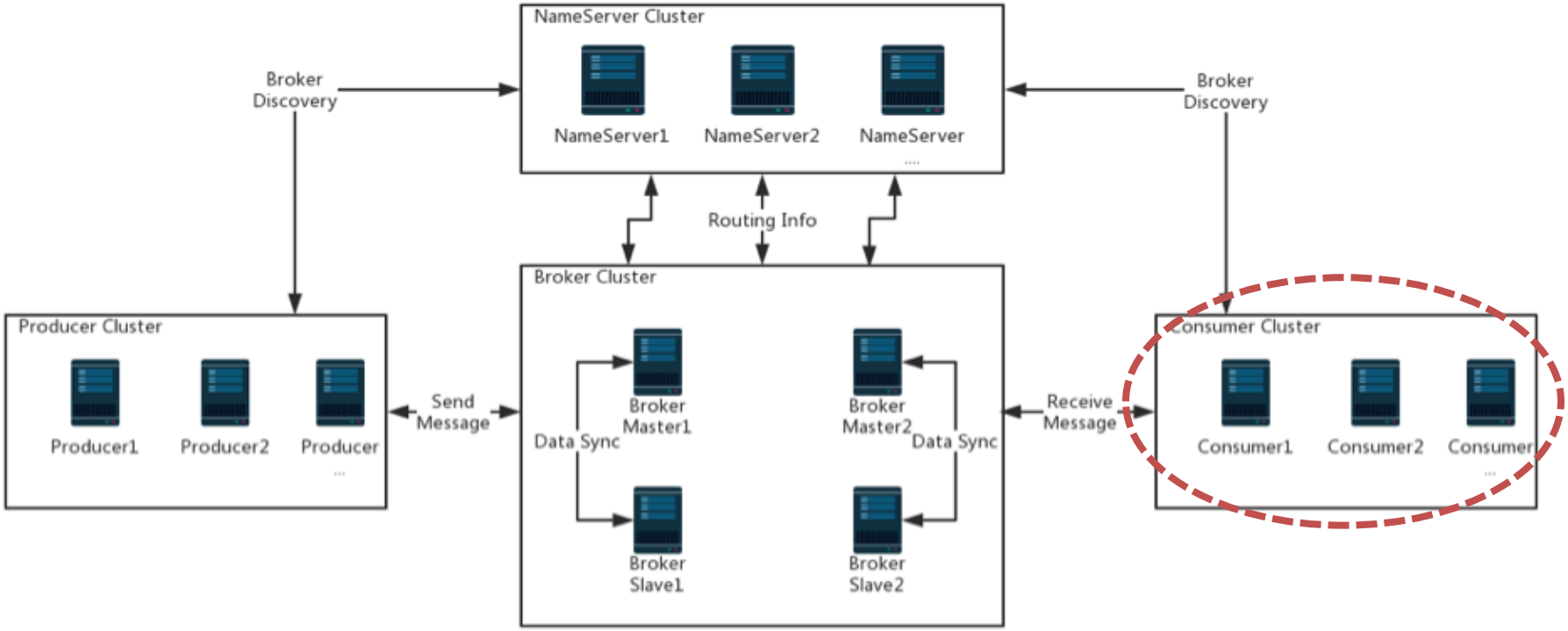
RocketMQ Architecture

- Producer Cluster:
  - Interact with NameServer Cluster for consumer information
  - Is responsible for sending messages to message queue(e.g., broker cluster)



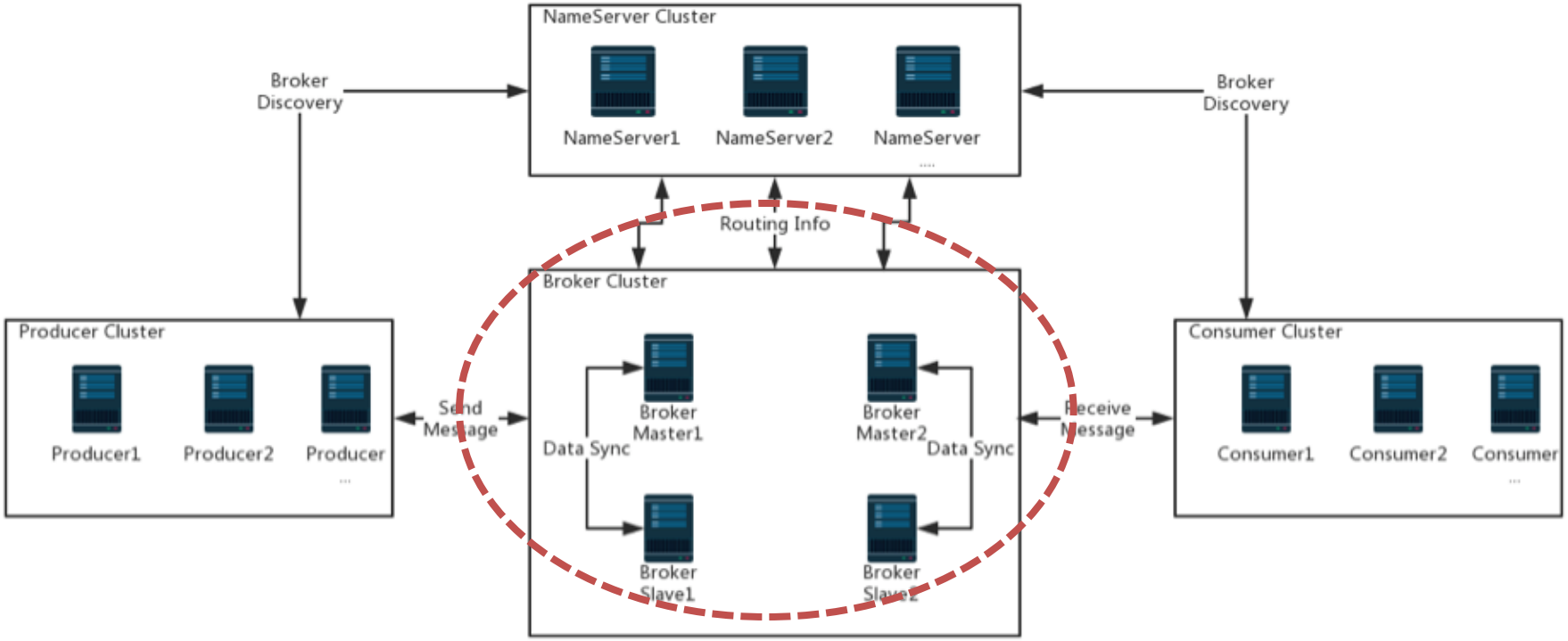
RocketMQ Architecture

- Consumer Cluster:
  - Interact with NameServer Cluster for producer information
  - Is responsible for receiving messages from message queue(e.g., broker cluster)



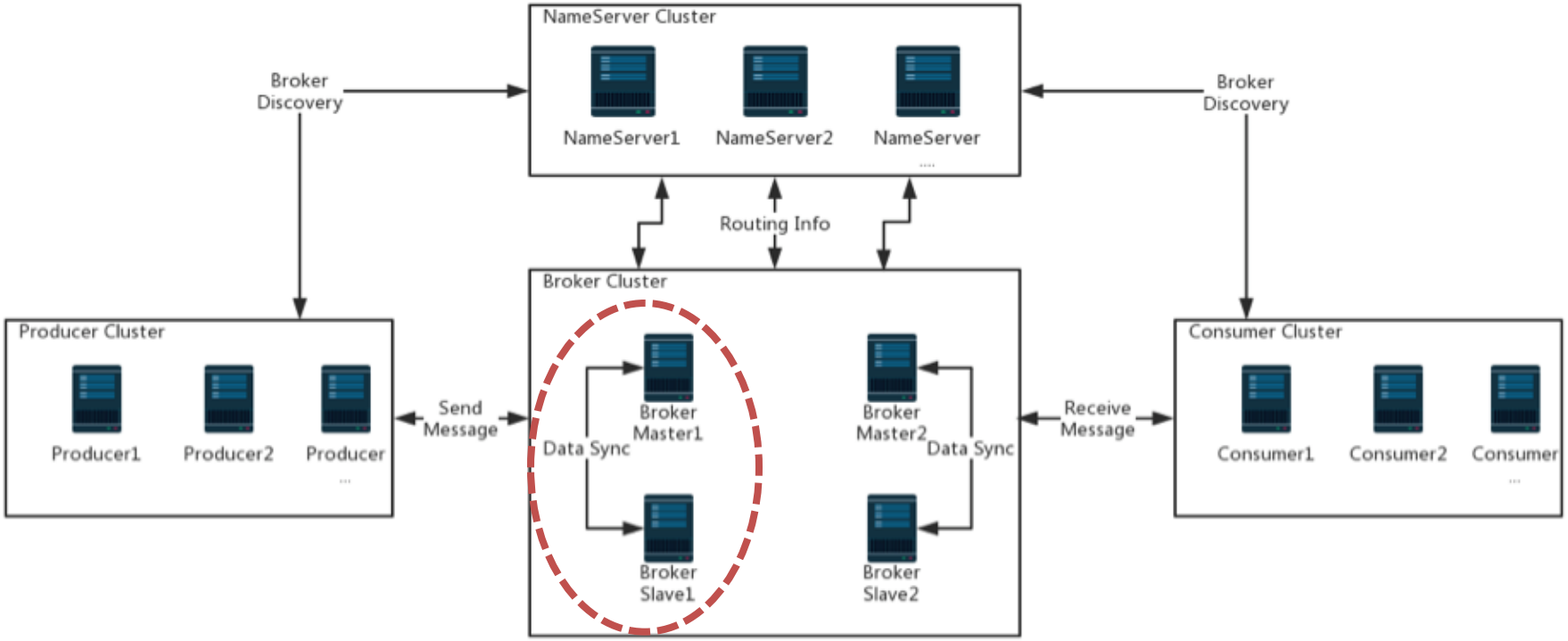
RocketMQ Architecture

- Broker Cluster:
  - Store the message data which is sent from producers and received from consumers

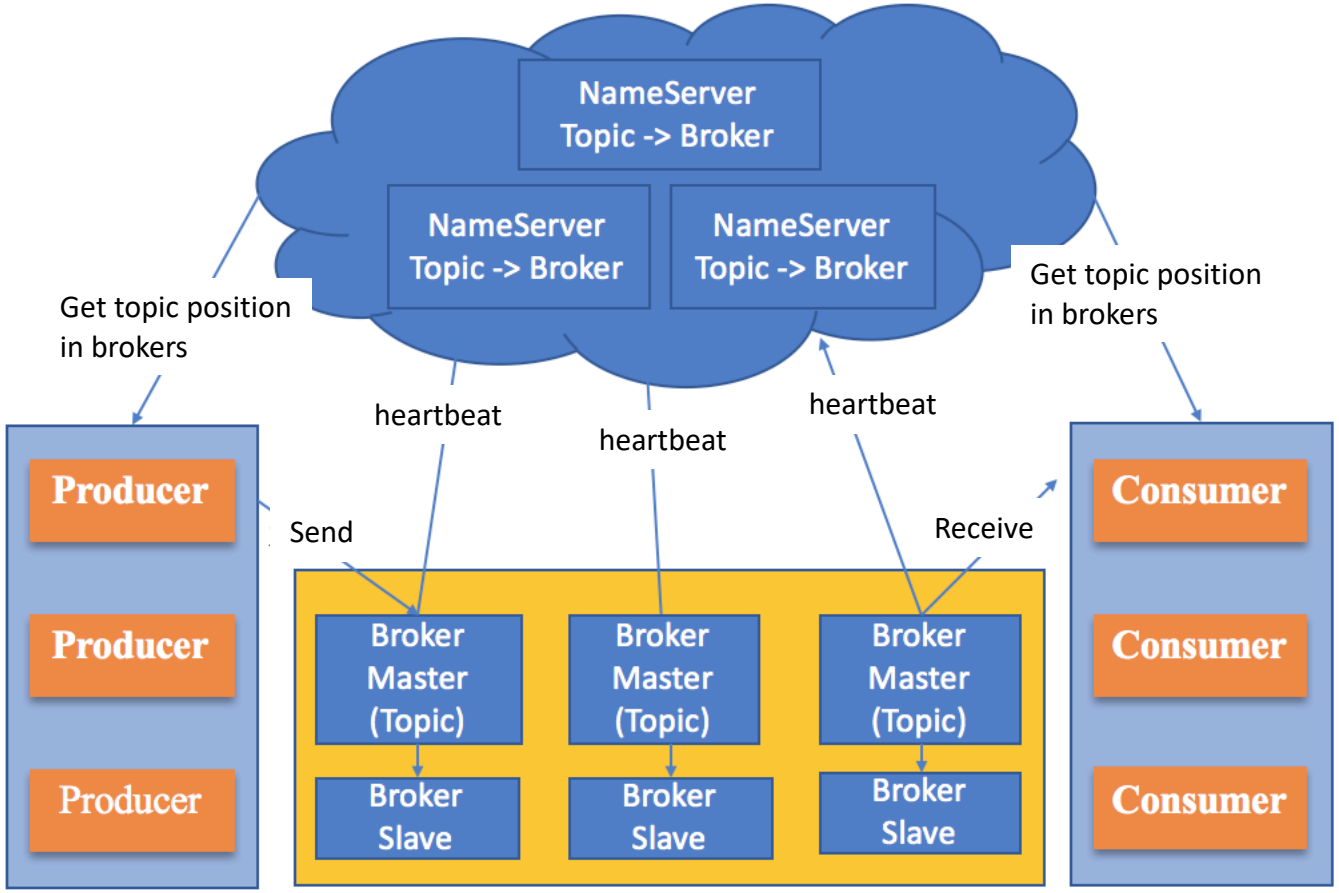


RocketMQ Architecture

- Broker Cluster:
  - Every broker is implemented as master-slave mode for high reliability.
  - Master can read and write while the slaves can only read.
  - Master receives messages from producers and syncs all data with slaves.

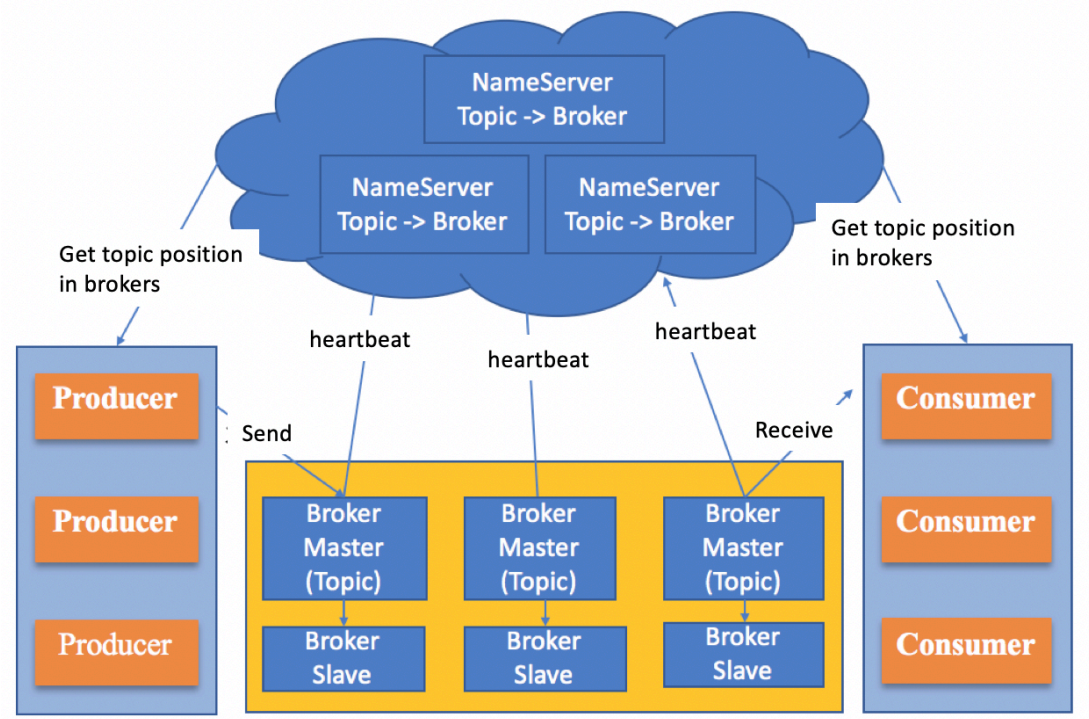


How Broker Cluster works as Message Queue?

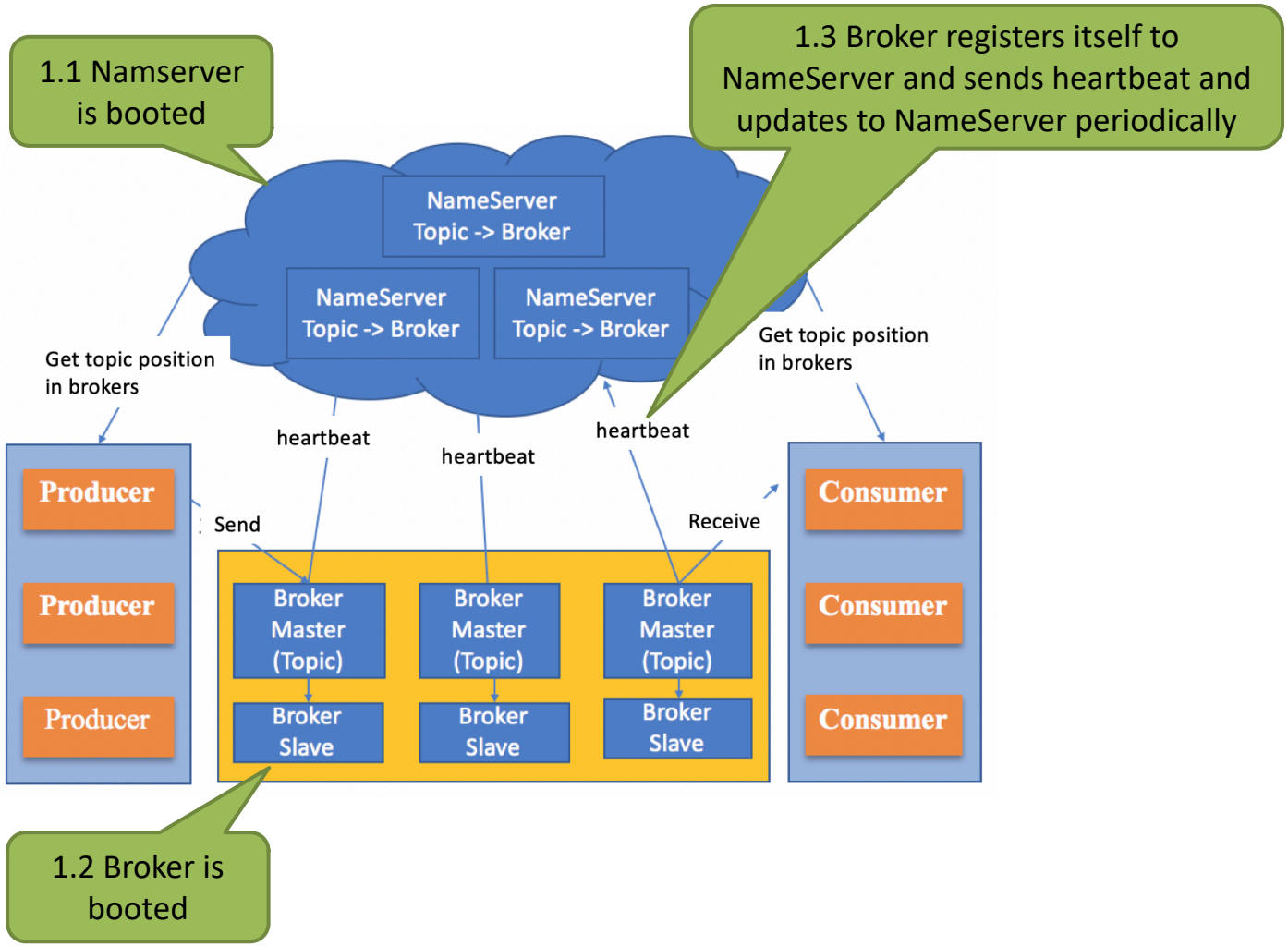


How Broker Cluster works as Message Queue?

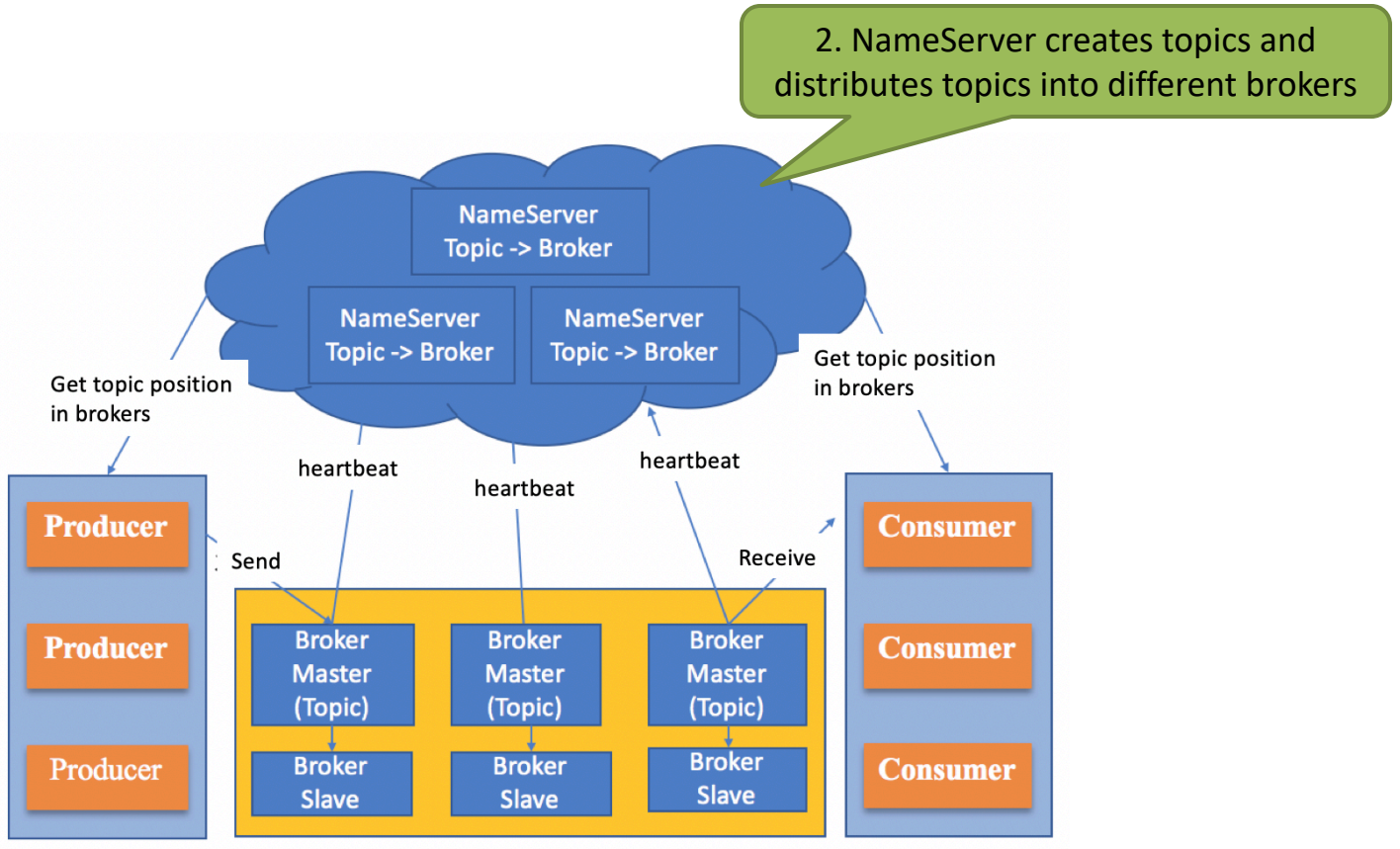
- Topic refers to message types or data types
- Each broker only stores part of one topic data to achieve high reliability



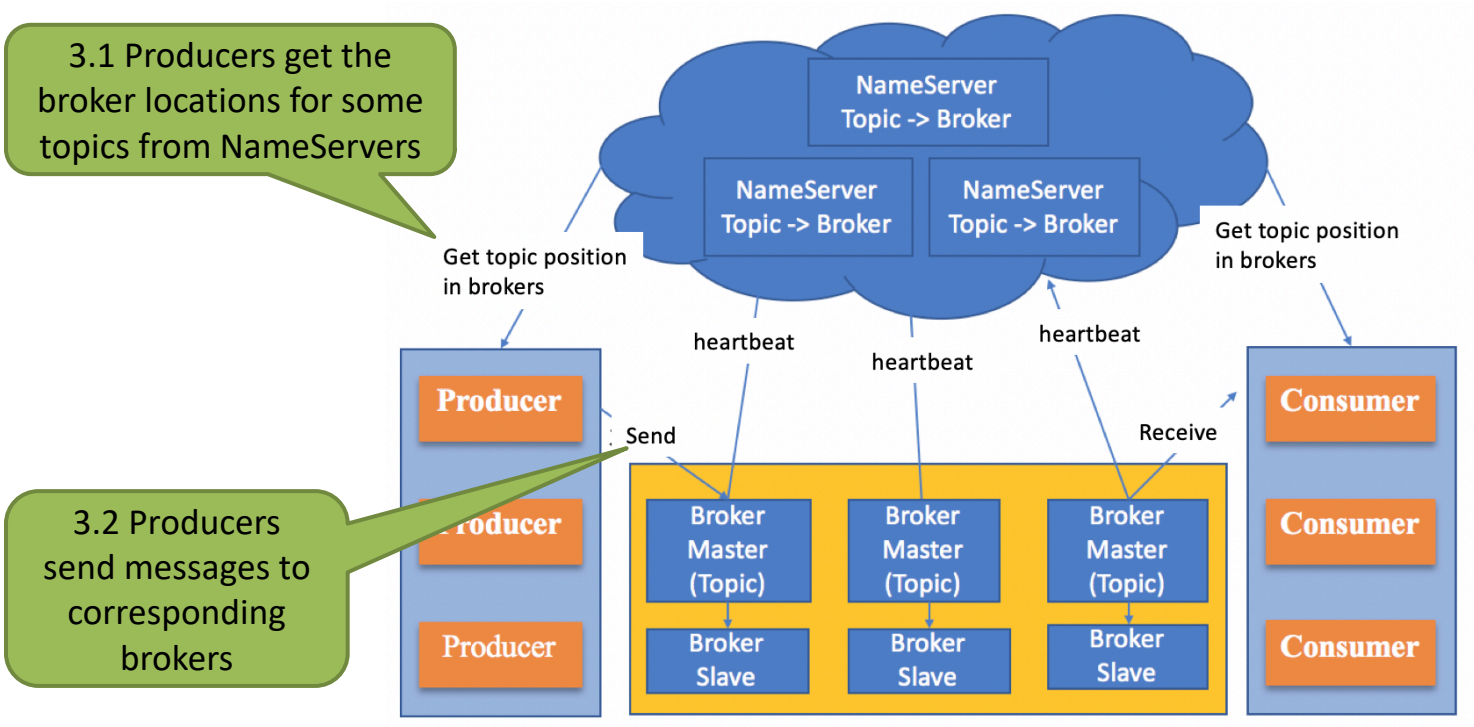
How Broker Cluster works as Message Queue?



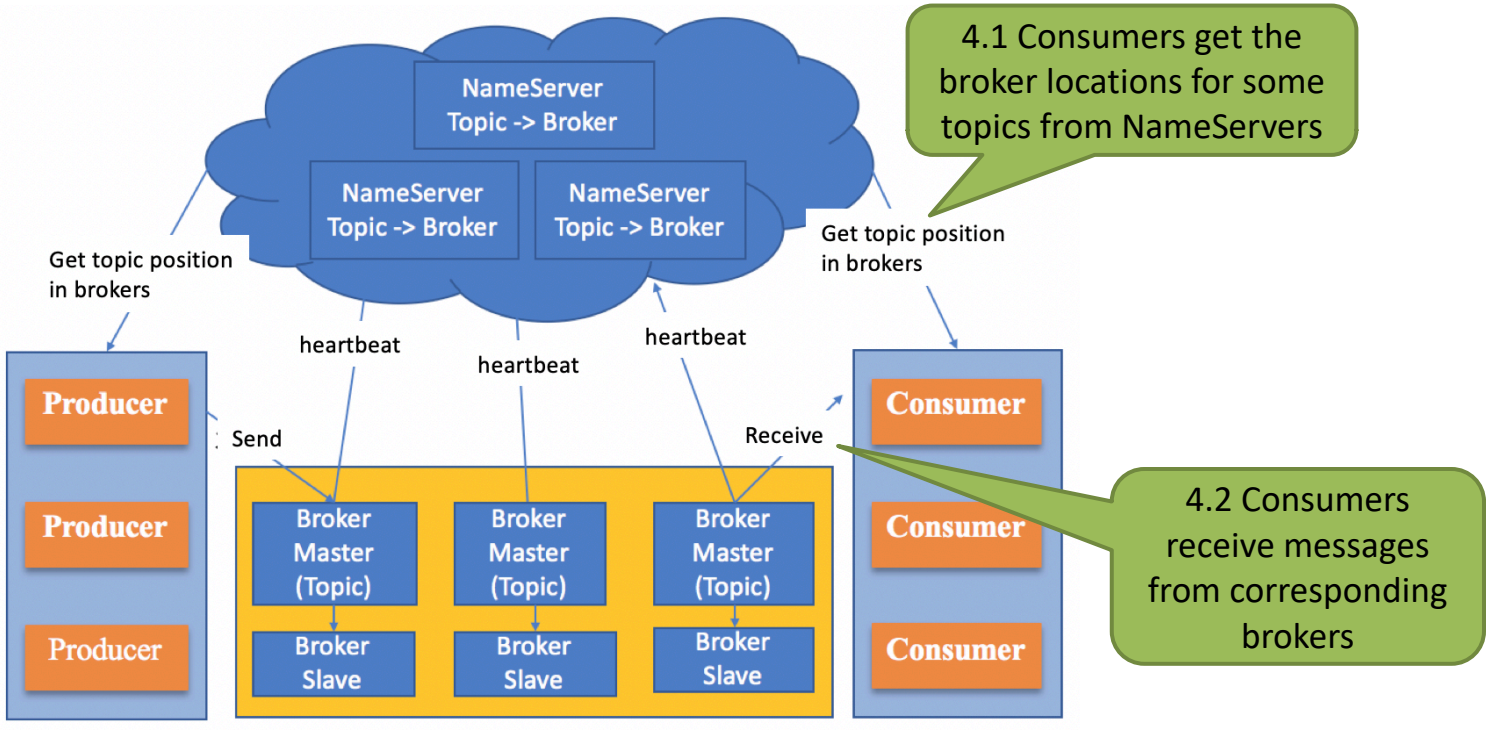
How Broker Cluster works as Message Queue?



How Broker Cluster works as Message Queue?



How Broker Cluster works as Message Queue?



Pub/Sub model vs. Message Queue model

- Data structure
  - Similarity:
    - The publish-subscribe model and message queue both use a message center, which stores data published by producers
    - They all have concepts such as topics and brokers
  - Difference:
    - The publish-subscribe model uses maps or arrays to implement the message center
    - Message queue adopts First-in, first-out queue structure

Pub/Sub model vs. Message Queue model

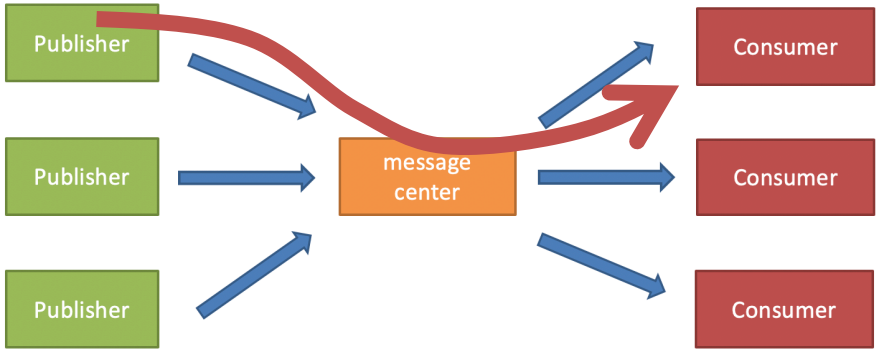
- Decoupling method
  - Pub/Sub model:
    - Consumers need to subscribe the topics **in advance**. After the producers publish the data to the message center, the message center pushes the data to consumers according to the subscriber's information.
  - Message Queue model:
    - Producers publish data to the message queue, and the message queue will store the data, waiting for consumers to obtain data on demand

Pub/Sub model vs. Message Queue model

- Scenario
  - Pub/Sub model:
    - Consumers are required to subscribe to messages from the message center in advance.
    - The message center needs to obtain consumer information in advance, which is more suitable for scenarios where consumers are **long lived processes or services**.
  - Message Queue model:
    - The message queue does not need to obtain consumer information in advance, so it is more flexible for consumers and suitable for scenarios where consumers are **temporary users or services**

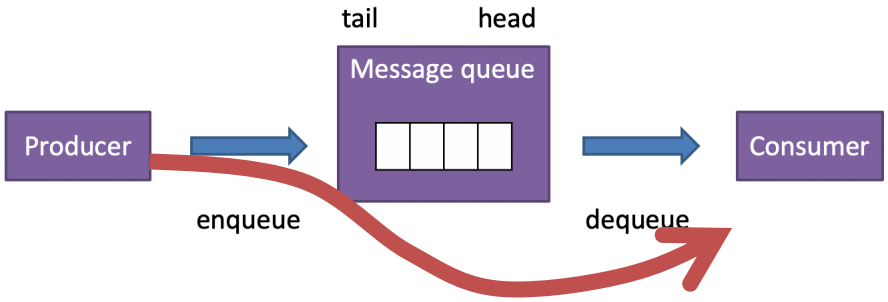
How to send message from one to another?

Publish/Subscribe Model



How to send message from one to another?

Message Queue model



Conclusion

- Message queue in OS
- What is message queue in distributed system
- Example of message queue: RocketMQ
- Pub/Sub vs. Message queue