Secret of Eventual Consistency in Apache RocketMQ, with No Budget

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Alibaba's products remains at the sole discretion of Alibaba.

Du Heng

Contributor of RocketMQ

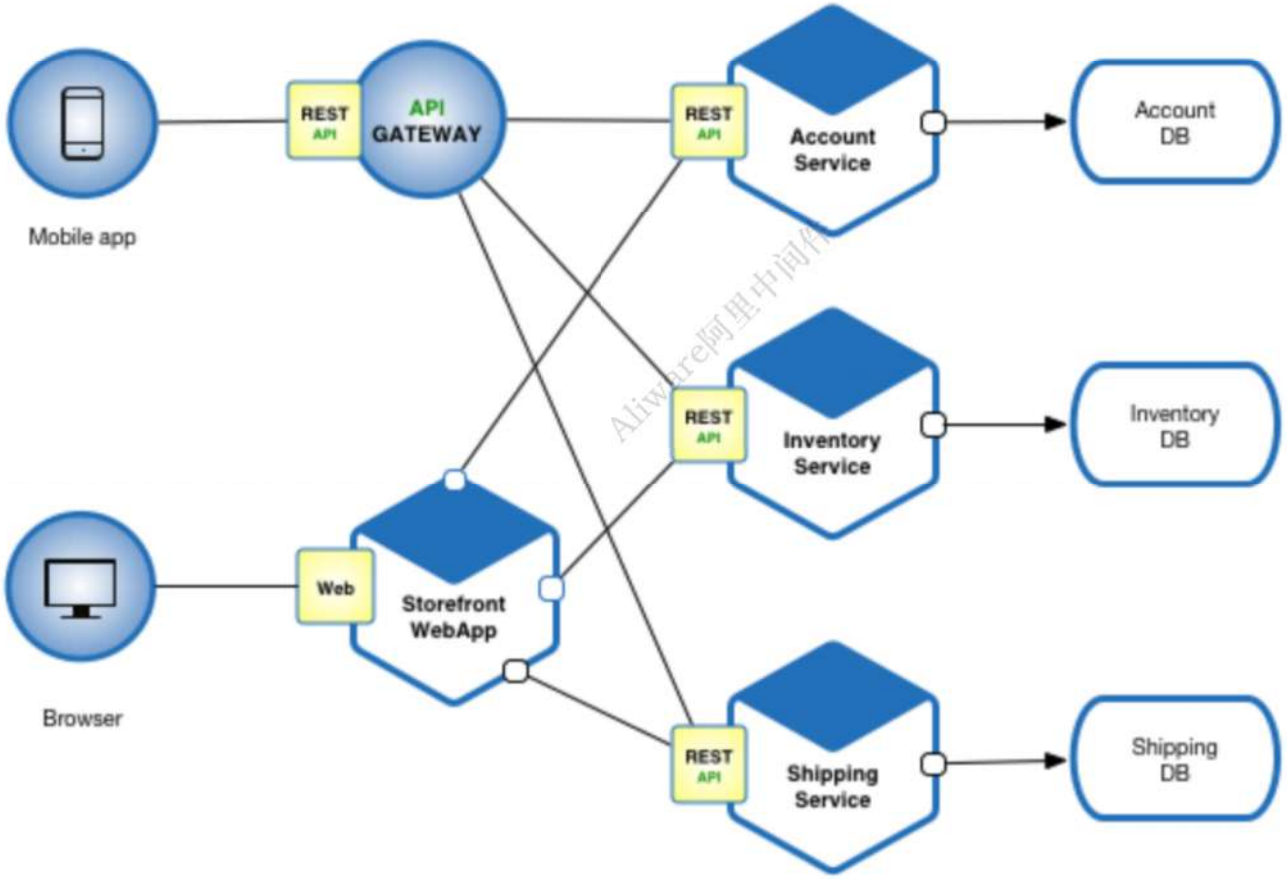
Interested in distributed system, such as MQ, Microservice, and has rich experience in performance tuning.

Outline

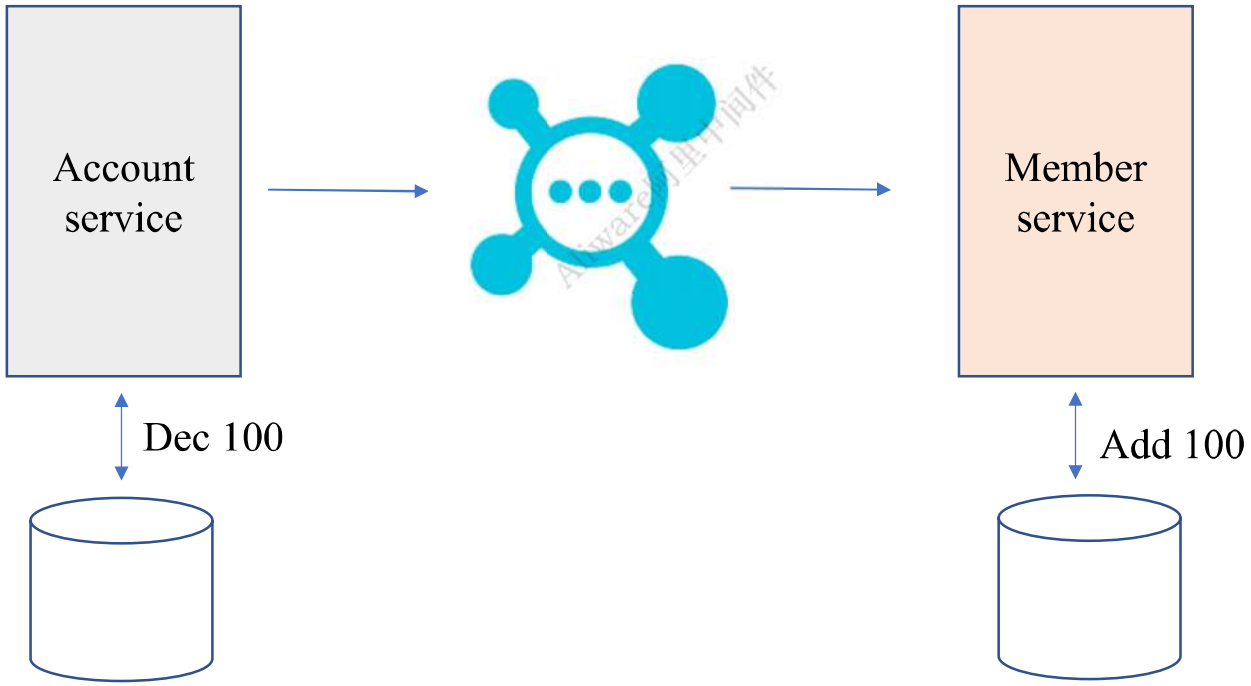
1. Challenge
2. Overview
3. Architecture
4. Best Practice
5. Future

1. Challenge

Challenge

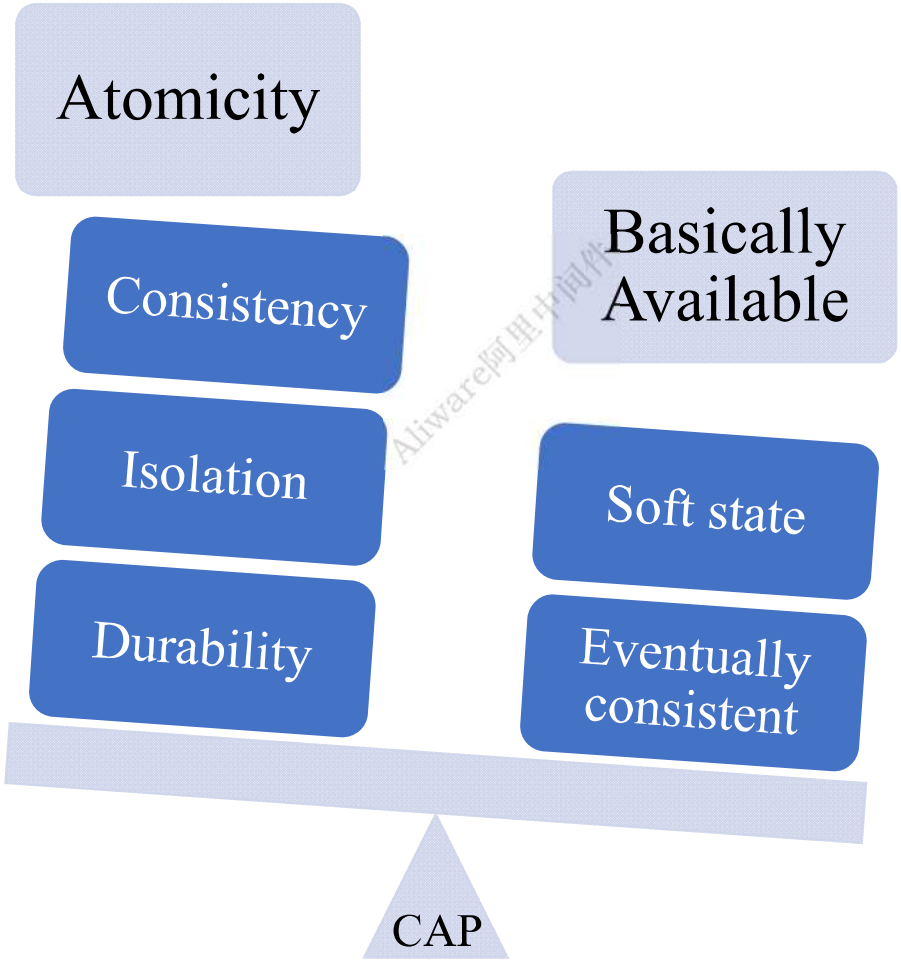

Mobile app
REST API
API GATEWAY
REST API
Account Service
Account DB
REST API
Inventory Service
Inventory DB
Browser
Web
Storefront WebApp
REST API
Shipping Service
Shipping DB

User Case

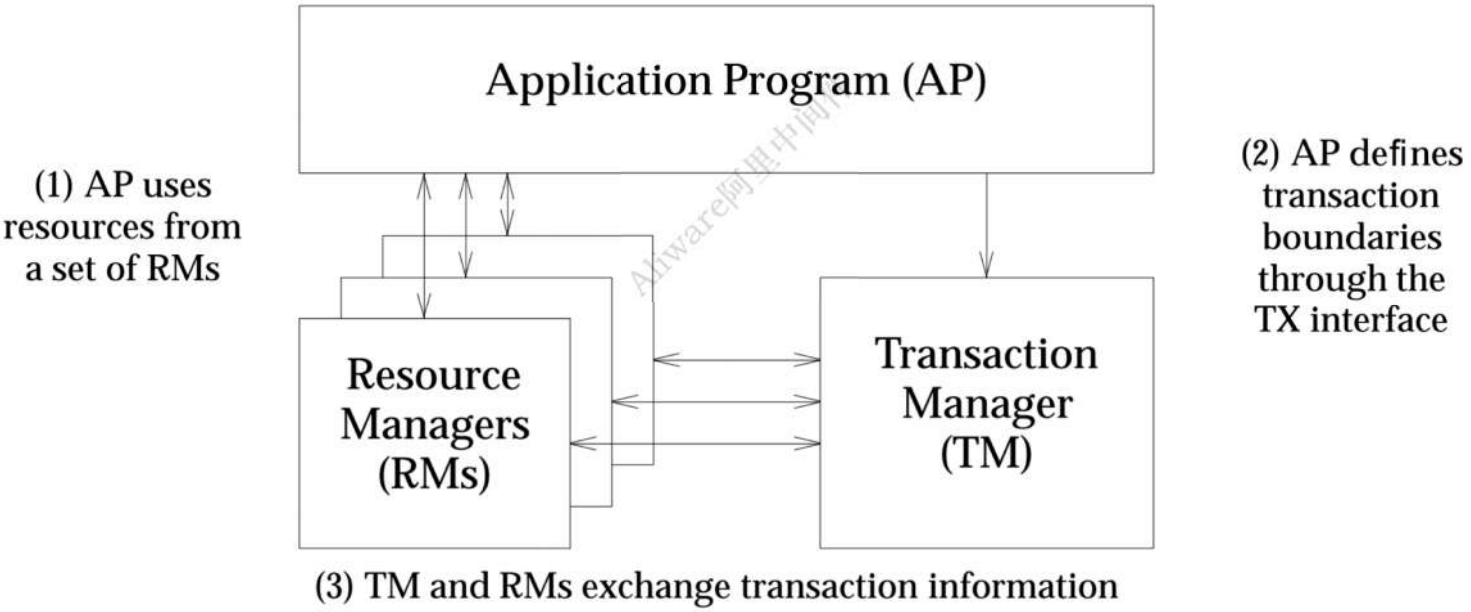


2. Overview

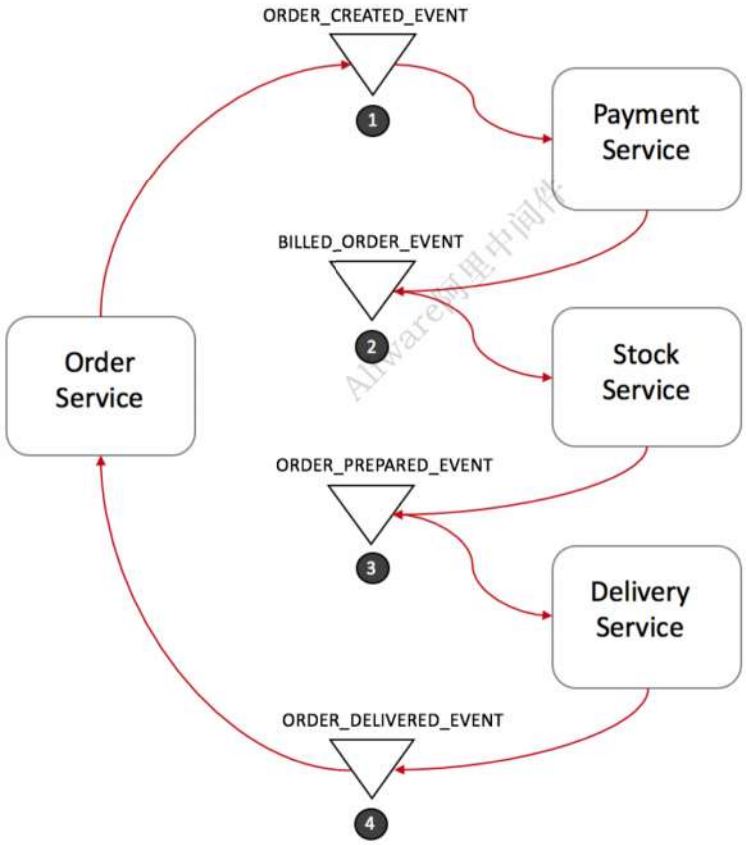
From ACID to BASE



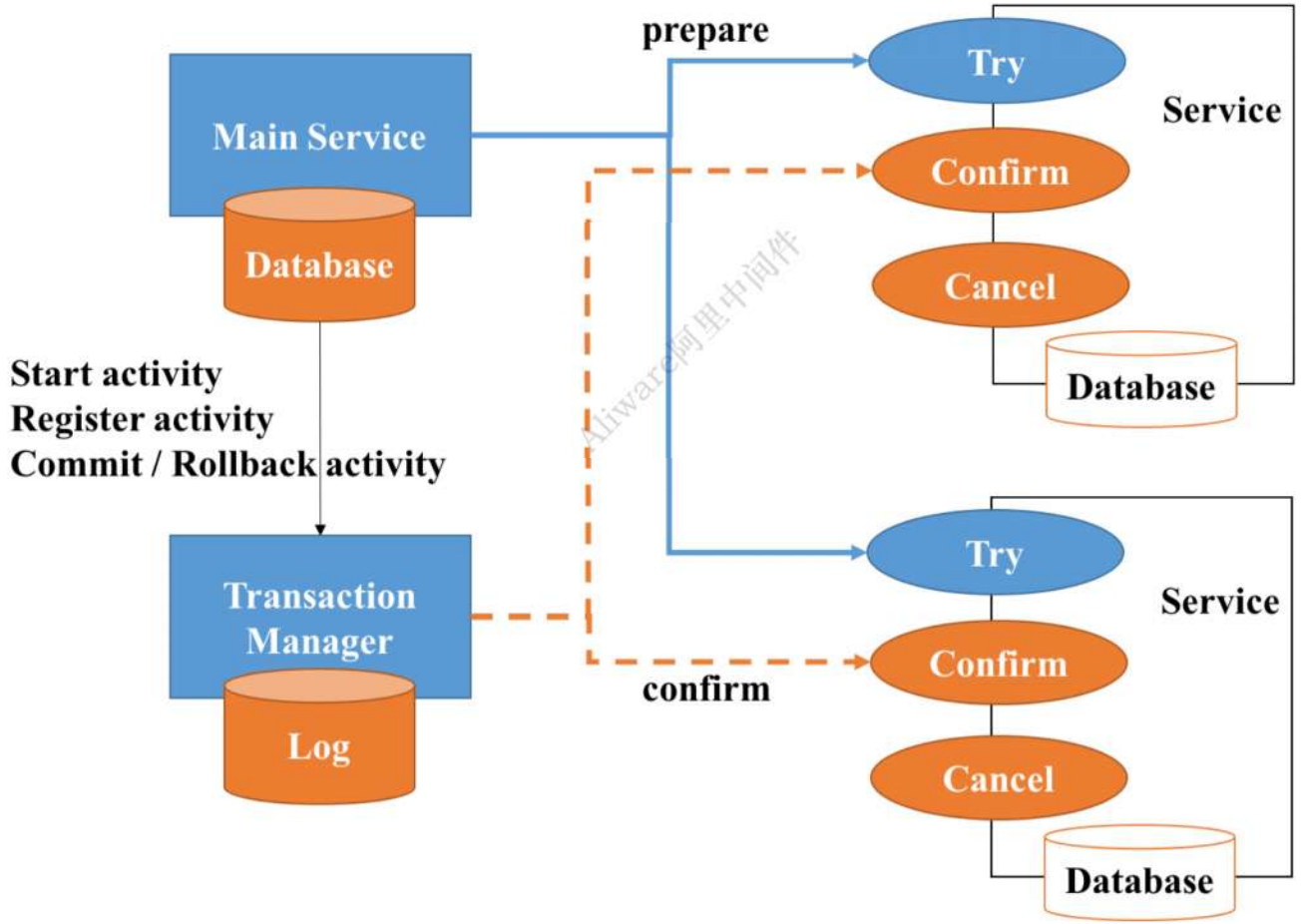
XA


Application Program (AP)
(1) AP uses resources from a set of RMs
(2) AP defines transaction boundaries through the TX interface
Resource Managers (RMs)
Transaction Manager (TM)
(3) TM and RMs exchange transaction information

SAGA

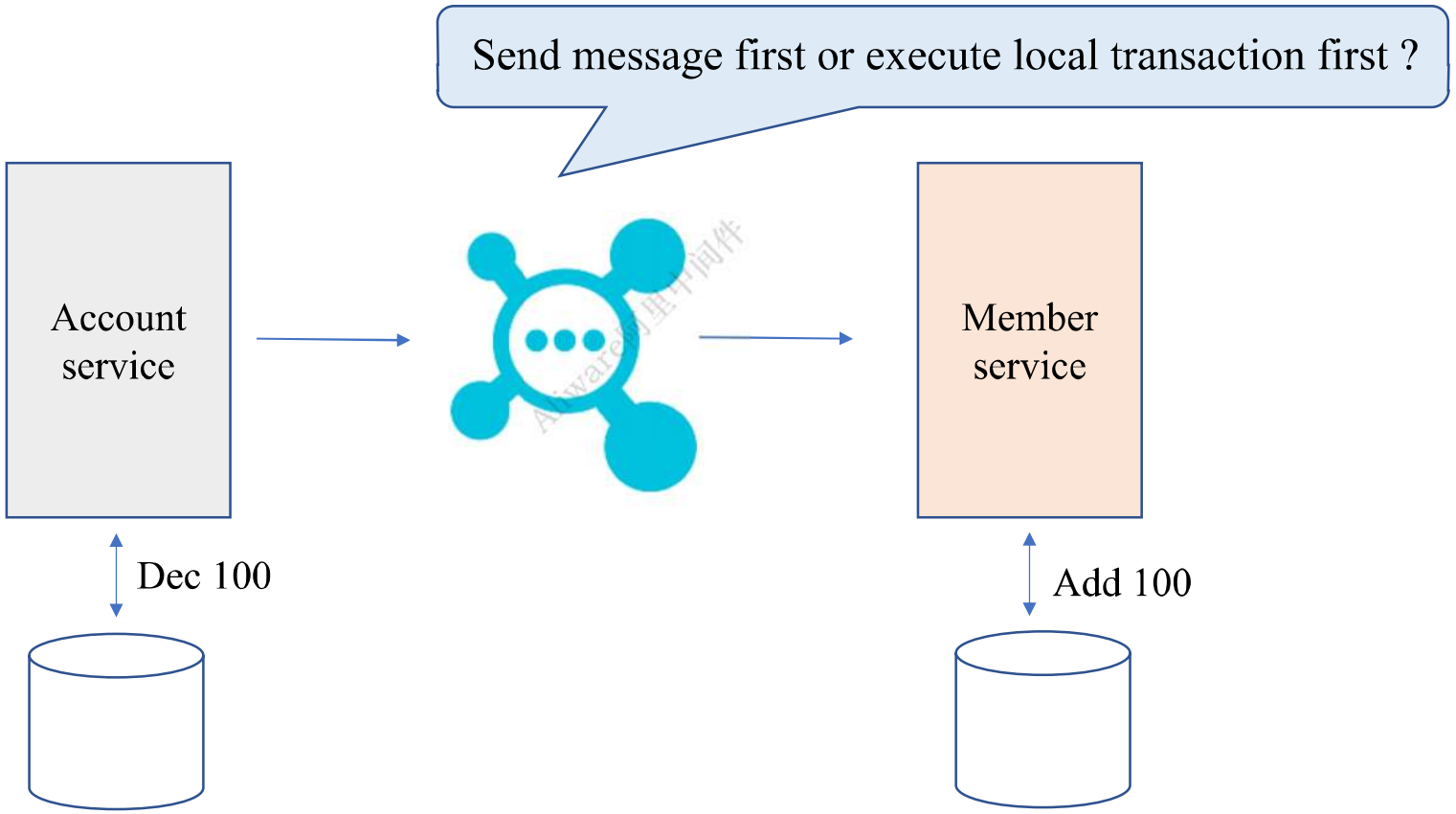

ORDER_CREATED_EVENT
1
Payment Service
BILLED_ORDER_EVENT
2
Order Service
Stock Service
ORDER_PREPARED_EVENT
3
Delivery Service
ORDER_DELIVERED_EVENT
4

TCC


prepare
Main Service
Database
Start activity
Register activity
Commit / Rollback activity
Transaction Manager
Log
confirm
Try
Confirm
Cancel
Database
Service
Try
Confirm
Cancel
Database
Service

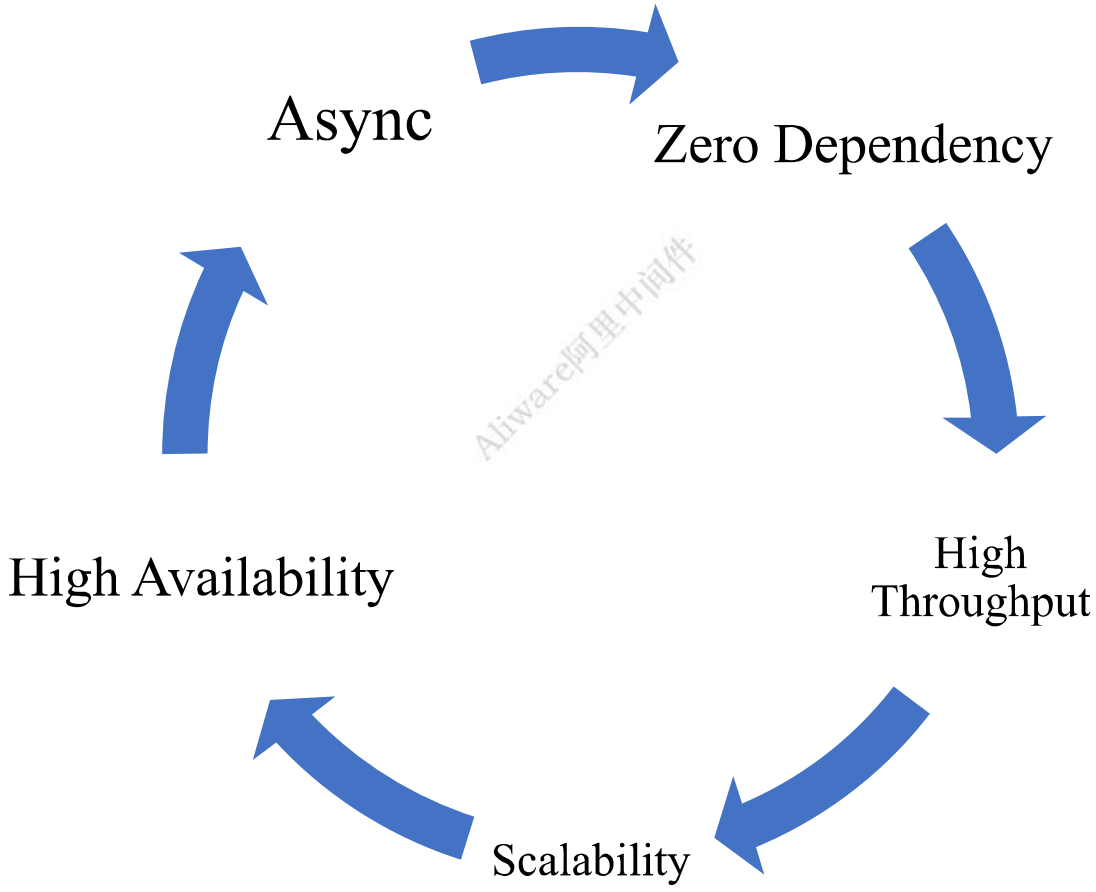
3. Architecture

Transactional Message

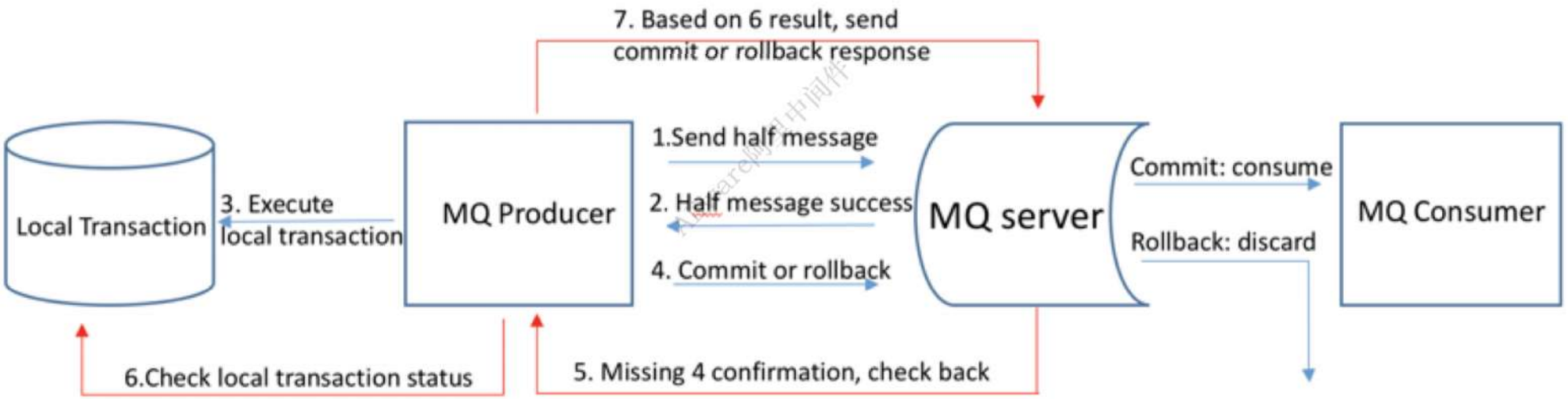


Counterpart Implementation

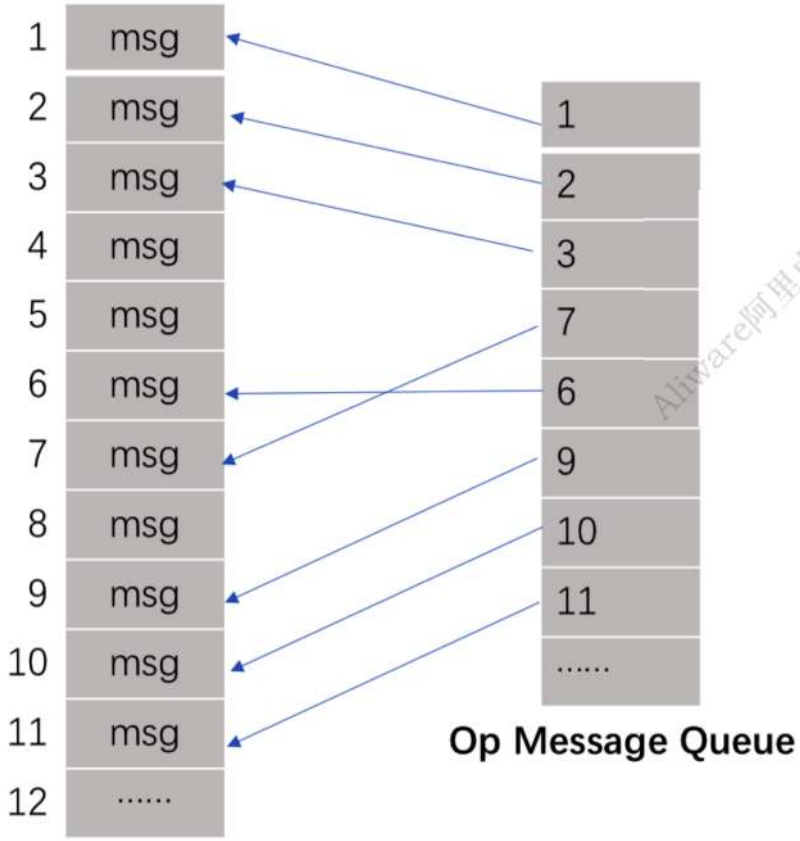
Goal



Procedure


7. Based on 6 result, send
commit or rollback response
1.Send half message
2. Half message success
4. Commit or rollback
3. Execute
local transaction
Local Transaction
MQ Producer
MQ server
Commit: consume
Rollback: discard
MQ Consumer
6.Check local transaction status
5. Missing 4 confirmation, check back

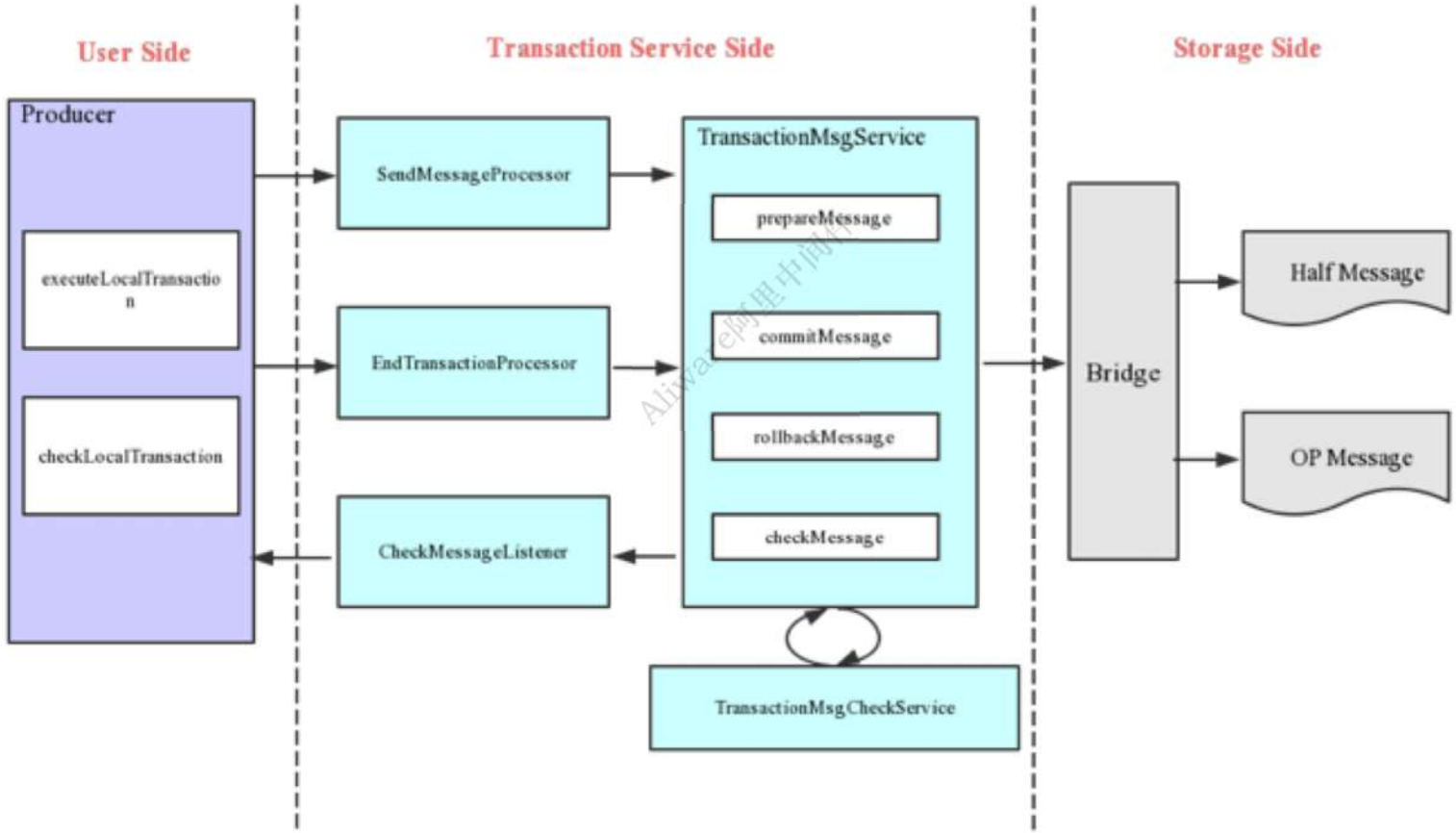
Design



Half Topic: prepare message

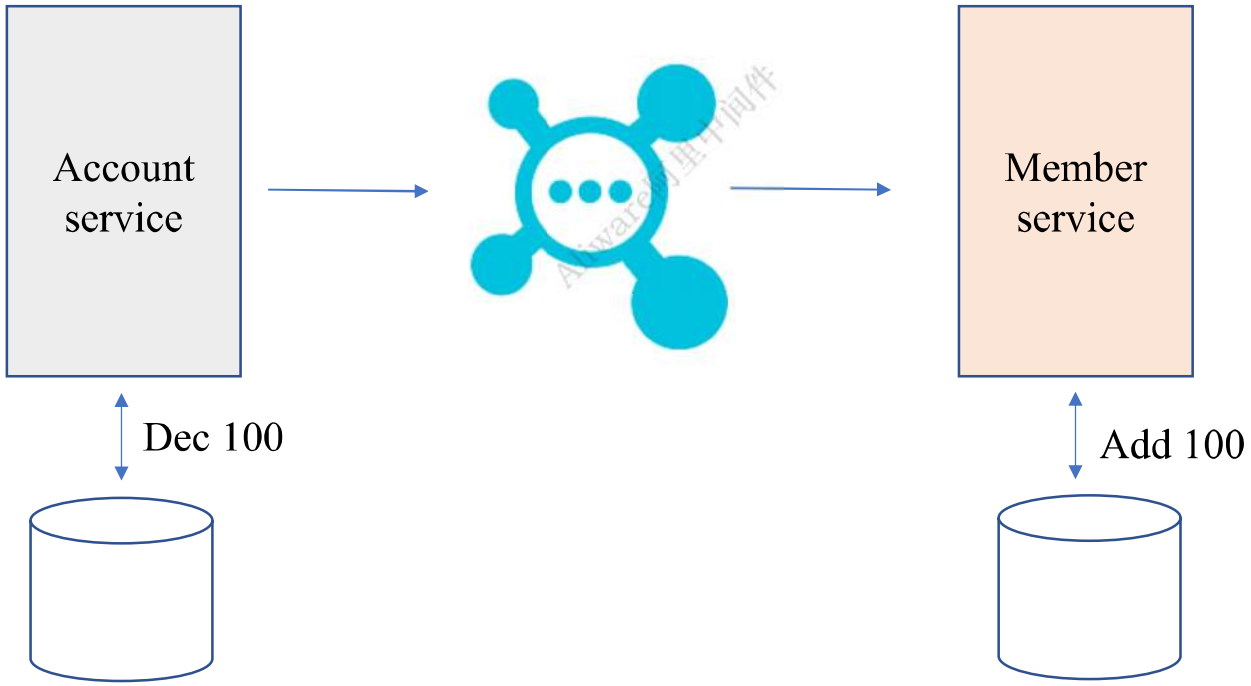
Operation Topic: commit/rollback message

Implementation



4. Best Practice

Scenario



Producer Practice

- Use TransactionalProducer class to create transactional producer
- Use a specific producerGroup name.

```
TransactionListener transactionListener = new TransactionListenerImpl();
TransactionMQProducer producer = new TransactionMQProducer("please_rename_unique_group_name");
ExecutorService executorService = new ThreadPoolExecutor(2, 5, 100, TimeUnit.SECONDS, new ArrayBlockingQueue<Runnable>(2000),
    new ThreadFactory() {
    @Override
    public Thread newThread(Runnable r) {
        Thread thread = new Thread(r);
        thread.setName("client-transaction-msg-check-thread");
        return thread;
    }
});

producer.setExecutorService(executorService);
producer.setTransactionListener(transactionListener);
producer.start();
```

Producer Practice

- Set the appropriate thread pool for your local transaction execution.
- transactionTimeOut vs CHECK_IMMUNITY_TIME_IN_SECONDS

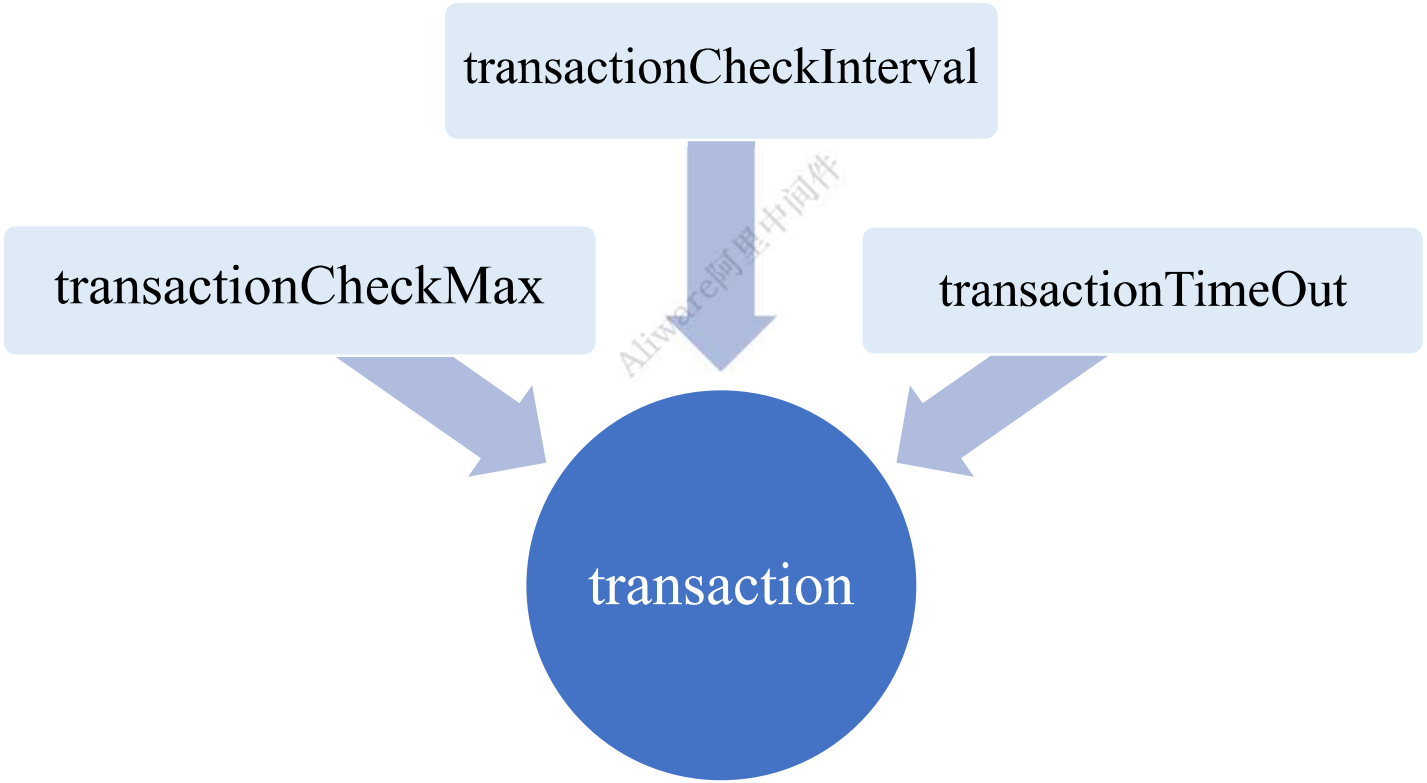
```
for (int i = 0; i < 10; i++) {
    try {
        Message msg =
            new Message("TopicTest1234", tags[i % tags.length], "KEY" + i,
                ("Hello RocketMQ " + i).getBytes(RemotingHelper.DEFAULT_CHARSET));
        msg.putUserProperty(MessageConst.PROPERTY_CHECK_IMMUNITY_TIME_IN_SECONDS,"5000");
        SendResult sendResult = producer.sendMessageInTransaction(msg, null);
        System.out.printf("%s%n", sendResult);

        Thread.sleep(10);
    } catch (MQClientException | UnsupportedEncodingException e) {
        e.printStackTrace();
    }
}
```

Producer Practice

```java
public class TransactionListenerImpl implements TransactionListener {
    private AtomicInteger transactionIndex = new AtomicInteger(0);

    private ConcurrentHashMap<String, Integer> localTrans = new ConcurrentHashMap<>();

    @Override
    public LocalTransactionState executeLocalTransaction(Message msg, Object arg) {
        int value = transactionIndex.getAndIncrement();
        int status = value % 3;
        localTrans.put(msg.getTransactionId(), status);
        return LocalTransactionState.UNKNOW;
    }

    @Override
    public LocalTransactionState checkLocalTransaction(MessageExt msg) {
        Integer status = localTrans.get(msg.getTransactionId());
        if (null != status) {
            switch (status) {
                case 0:
                    return LocalTransactionState.UNKNOW;
                case 1:
                    return LocalTransactionState.COMMIT_MESSAGE;
                case 2:
                    return LocalTransactionState.ROLLBACK_MESSAGE;
                default:
                    return LocalTransactionState.UNKNOW;
            }
        }
        return LocalTransactionState.COMMIT_MESSAGE;
    }
}
```

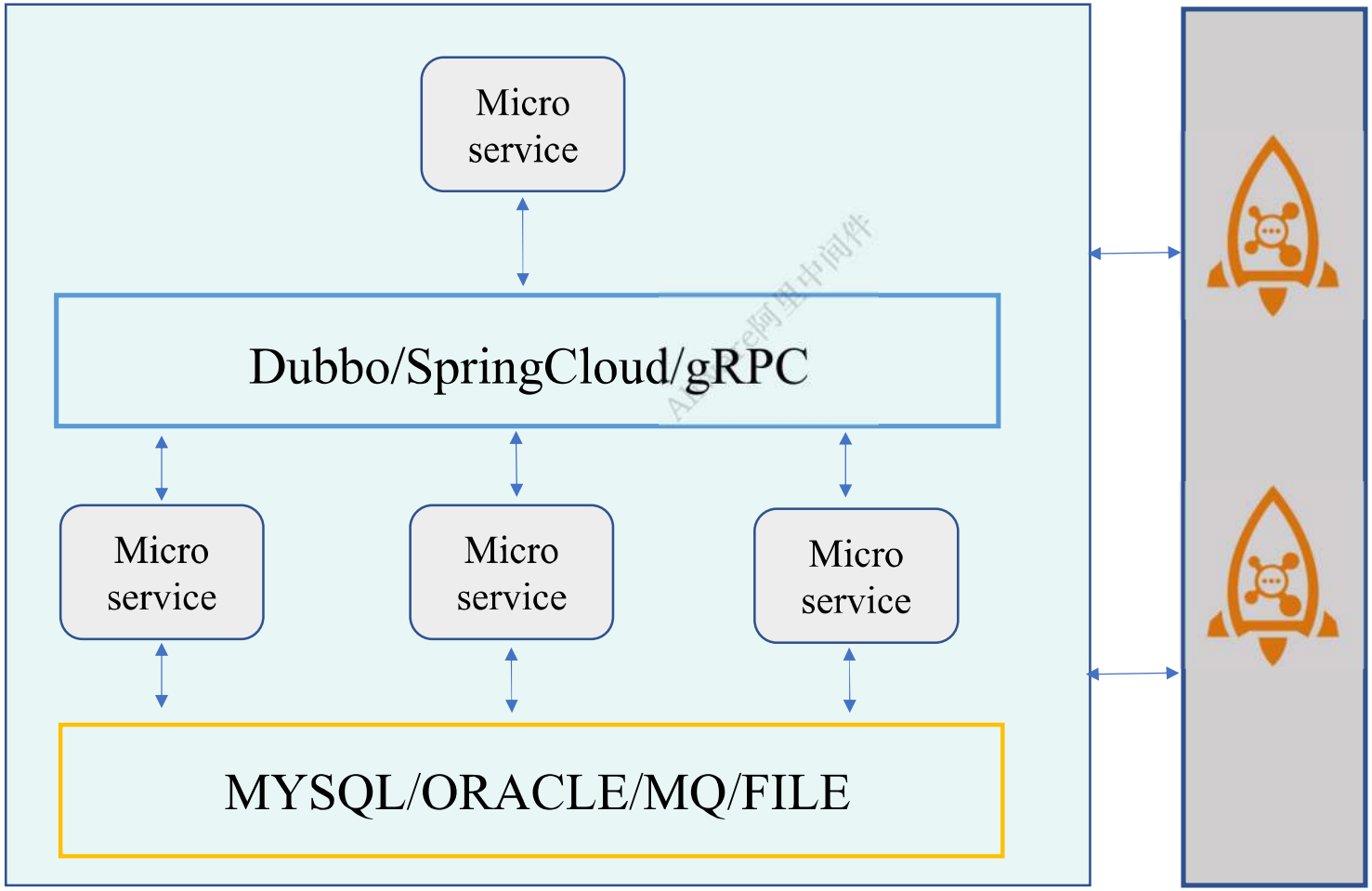
- Implement TransactionListener interface.
  - LocalTransactionState.UNKNOW
  - LocalTransactionState.COMMIT_MESSAGE
  - LocalTransactionState.ROLLBACK_MESSAGE
- Idempotency（transactionId，message body）

Broker Practice



5. Future

One-stop Distributed Transaction Solution



Thanks !

推荐阅读：

- From Alibaba to Apache: RocketMQ’s Past, Present, and Future
- Apache RocketMQ 顶级项目之路
- Apache RocketMQ 背后的设计思路与最佳实践.
- 专访RocketMQ联合创始人：项目思路、技术细节和未来规划
- 万亿级数据洪峰下的分布式消息引擎
- RocketMQ联合创始人：选择MQ时，要注意的有哪些