CS 7172
Parallel and Distributed Computation

Distributed Transaction

Kun Suo

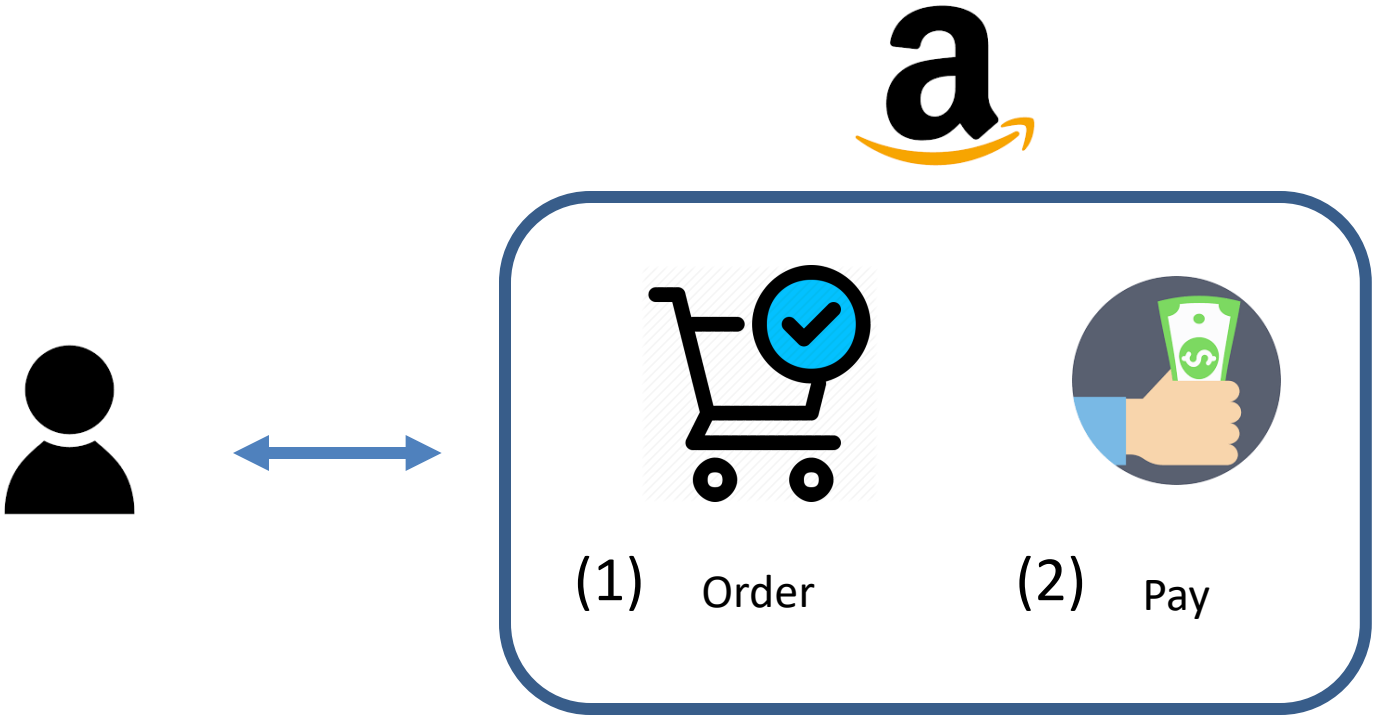
Computer Science, Kennesaw State University

https://kevinsuo.github.io/

Outline

- Computer networks, primarily from an application perspective
- Protocol layering
- Client-server architecture
- End-to-end principle
- TCP
- Socket programming

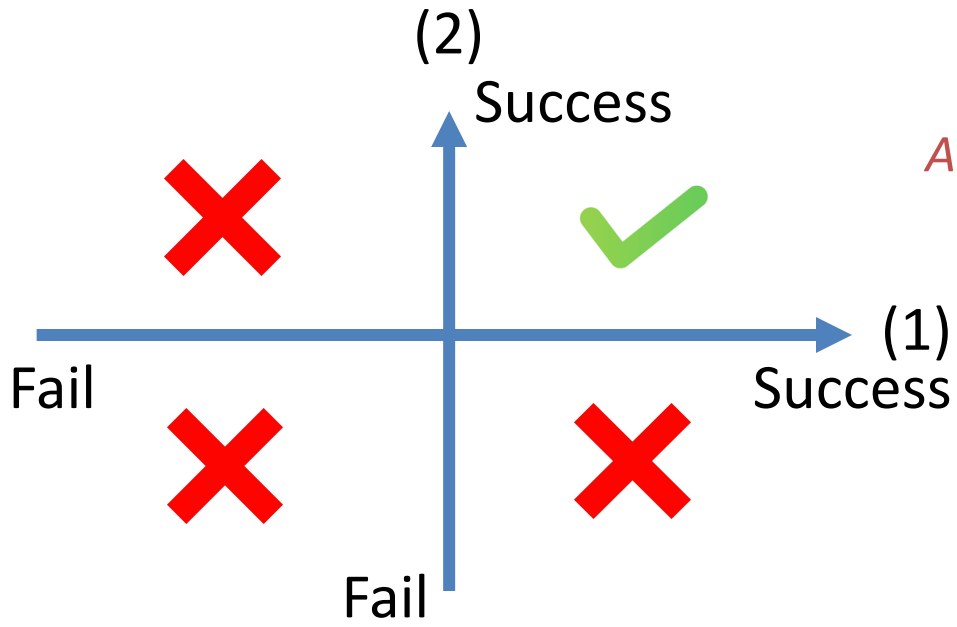
Start from an example



- For each order, the e-commerce platform generally has two core steps: first, the order business accepts the order placement operation, and second, the payment processes the online pay operation.

Start from an example

- These two businesses will run on different machines, or even machines in different regions. For the same order, the correctness of the transaction can be guaranteed if and only if the order operation and the payment operation are consistent.



All or nothing

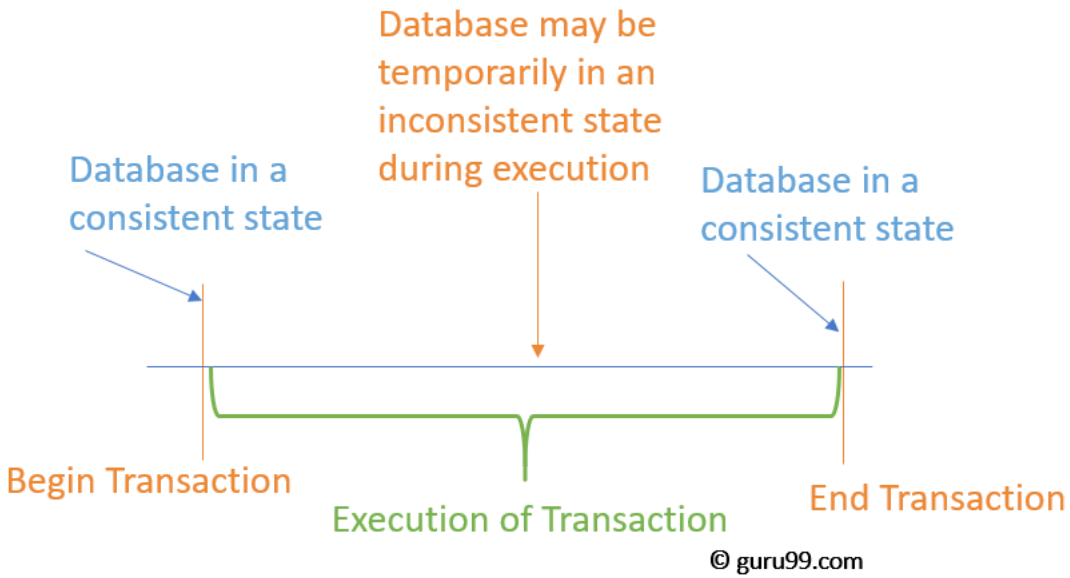
What is distributed transaction?

- Transaction:
  - A series of operations with boundary
  - With clear start and end marks
  - Either being completely executed or failed → all or nothing

- Distributed transaction:
  - Transactions running in a distributed system are composed of multiple local transactions
  - The transaction processing operations may come from different machines or systems

ACID of Distributed Transaction

- Atomicity:
  - only two final state of the transaction, all executed successfully, or all not executed. Any part of distributed transaction fails, the entire operation fails and has to be re-executed.

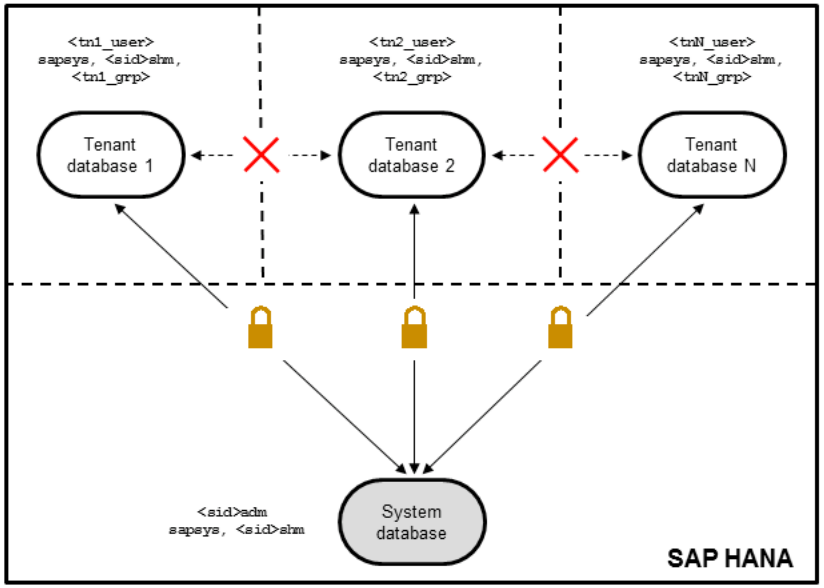


ACID of Distributed Transaction

- Consistency:
  - Before and after transaction operations, data integrity remains consistent or meets integrity constraints.
    - Example: A = 800 and B = 600. When A sends 200 to B, two operations are involved: (1) minus 200 from A; (2) add 200 onto B. After the operations, A = 600 and B = 800. Any other results do not meet integrity constraints.

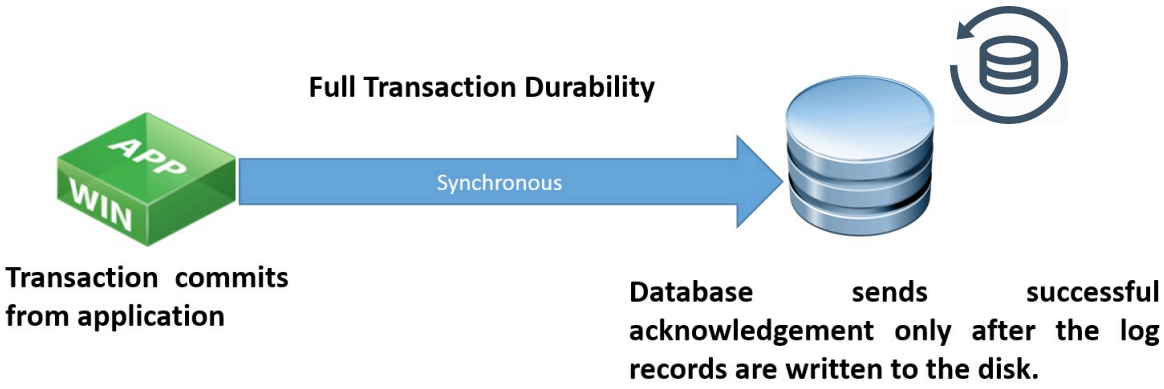
ACID of Distributed Transaction

- Isolation:
  - When multiple transactions are executed concurrently, these transactions will not interfere with each other. The operations and data used in one transaction are isolated from other concurrent transactions.



ACID of Distributed Transaction

- Durability:
  - When a transaction is completed, the updates it makes to the database are permanently saved. Even if a system crashes, as long as the database can be accessed again, it must be restored to the state when the transaction was completed.

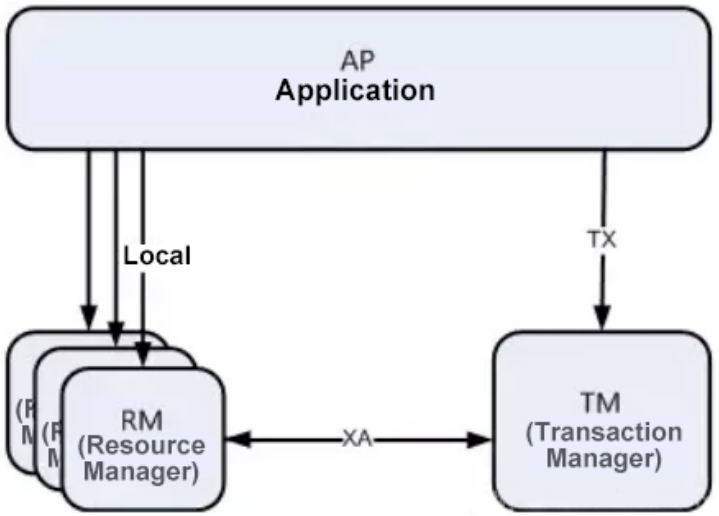


How to achieve distributed transaction?

- There are three basic methods for implementing distributed transactions:
  - a two-phase commit based on the XA protocol;
  - a three-phase commit protocol;
  - a message-based consistency.

1. Two-phase commit based on the XA protocol

- XA is a distributed transaction protocol that specifies the transaction manager and resource manager interfaces
  - **Transaction manager** works as the coordinator, which is responsible for the submission and rollback of local resources;
  - **Resource manager** is a participant in distributed transactions, usually implemented by databases, such as Oracle, DB2

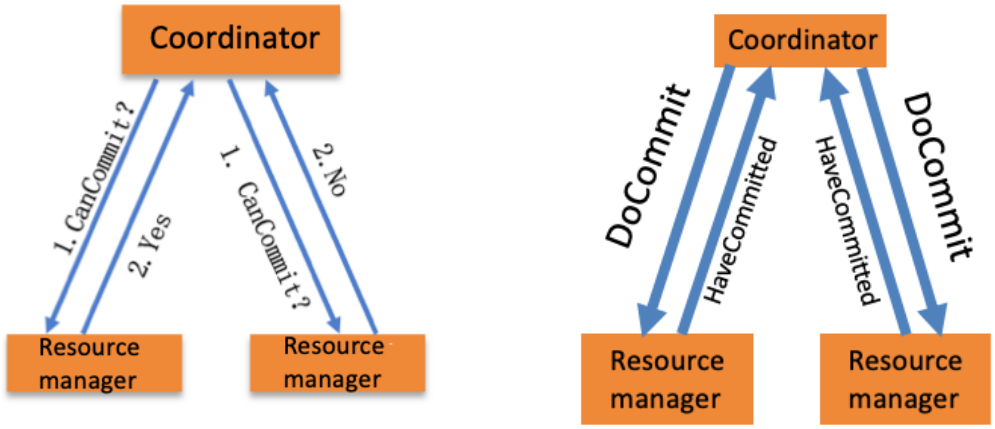


1. Two-phase commit based on the XA protocol

- The two-phase commit protocol (2PC):
  - used to ensure ***data consistency*** when transactions are committed in a distributed system

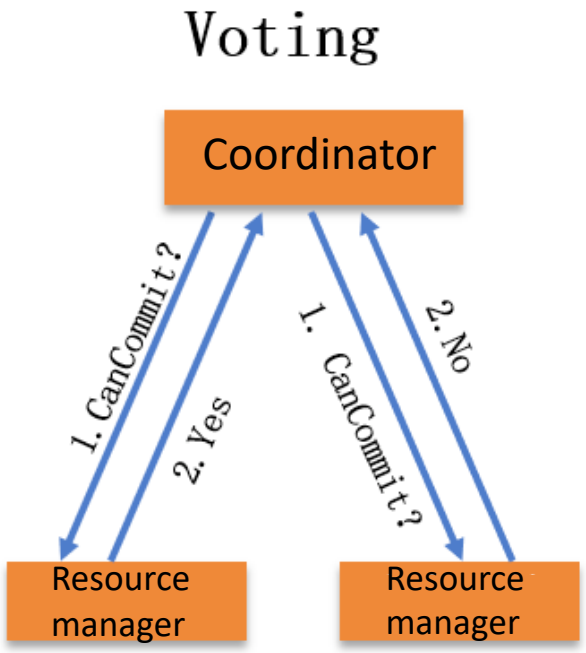
Phase 1: Voting

Phase 2: Commit



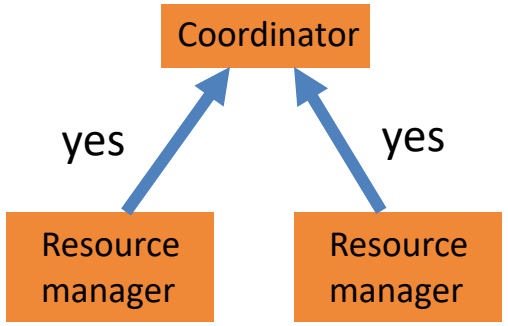
1. Two-phase commit based on the XA protocol

- Phase 1: voting
  1. Coordinator sends CanCommit requests to resource manager and waits for reply
  2. Resource manager receives requests and do the transactions.
     - Reply Yes, means approving the operation;
     - Reply No, means stop the operating.

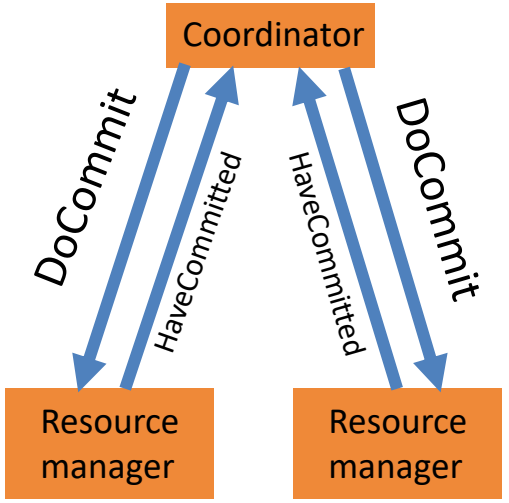


1. Two-phase commit based on the XA protocol

- Phase 2: commit
  1. If all replies are yes, Coordinator sends “DoCommit” operation

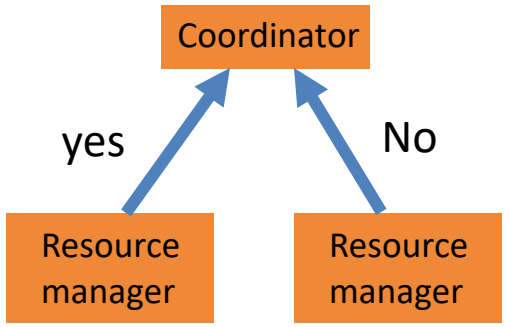


  2. The resource manager will do the rest of operations, release all resources and return “HaveCommitted” message

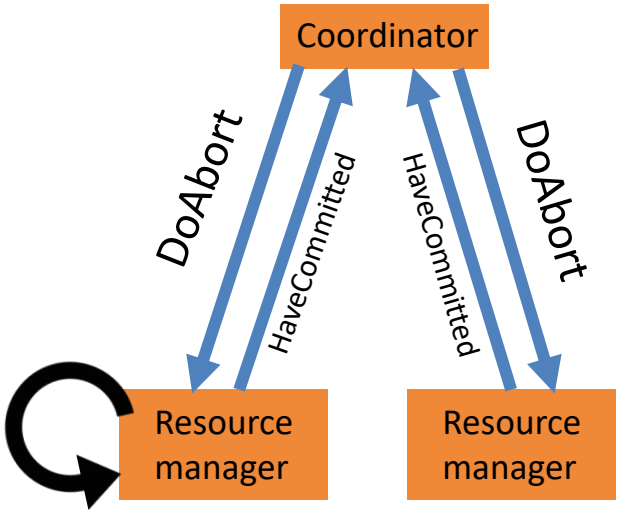


1. Two-phase commit based on the XA protocol

- Phase 2: commit
  1. If one of replies is no, Coordinator sends “DoAbort” operation

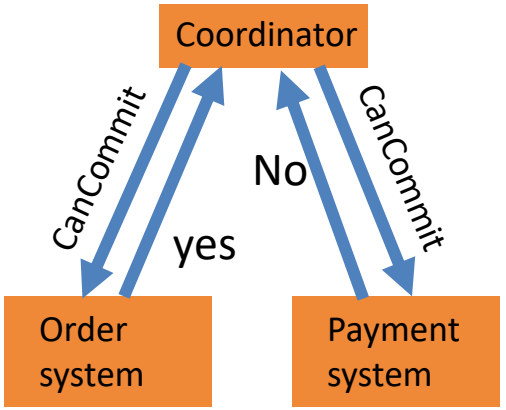


  2. The resource manager which sends “yes” before will rollback based on logs to the end of previous transaction, then sends “HaveCommitted” message



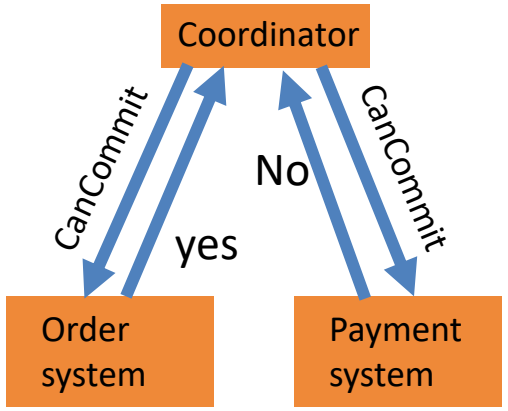
Example: user A buys 100 T-shirts on Amazon, involving order operation and payment

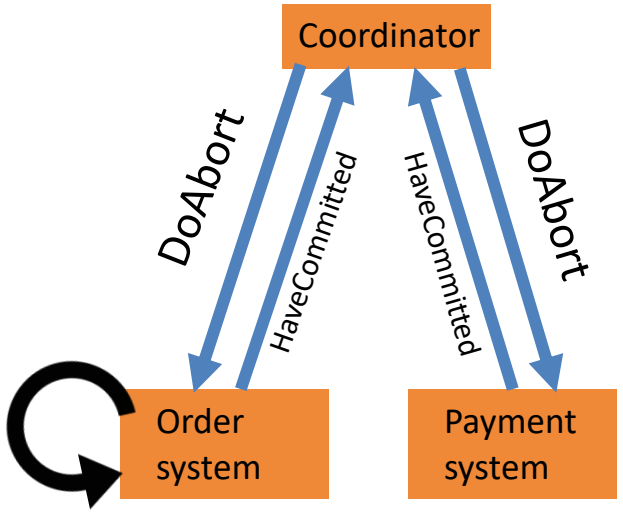
- Phase 1: voting
  1. Coordinator sends CanCommit requests to order and payment system and waits for reply
  2. Order system reply yes, lock the database, and do operations on database
  3. Payment system reply no due to the too high payment traffic



Example: user A buys 100 T-shirts on Amazon, involving order operation and payment

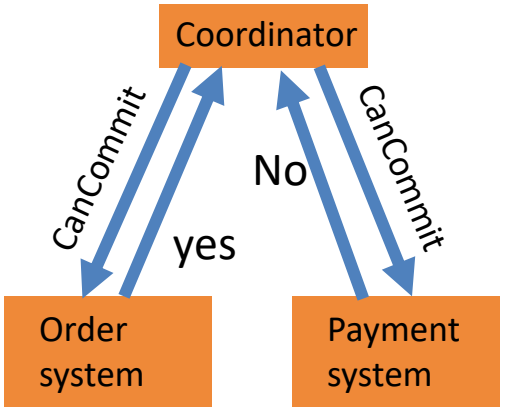
- Phase 2: commit
  1. Because one of replies is no, Coordinator sends “DoAbort” operation
  2. The order system rollbacks based on logs to the end of previous transaction, then sends “HaveCommitted” message
  3. The payment system just sends “HaveCommitted” message when receiving “DoAbort”

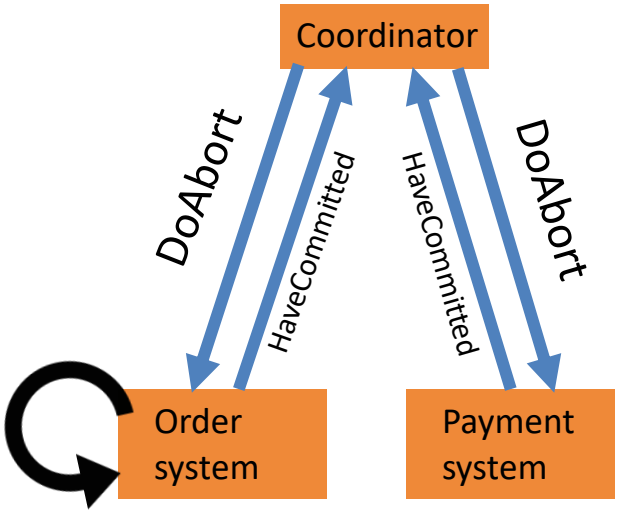




Example: user A buys 100 T-shirts on Amazon, involving order operation and payment

- The system return “failure” message on the user interface





1. Two-phase commit based on the XA protocol

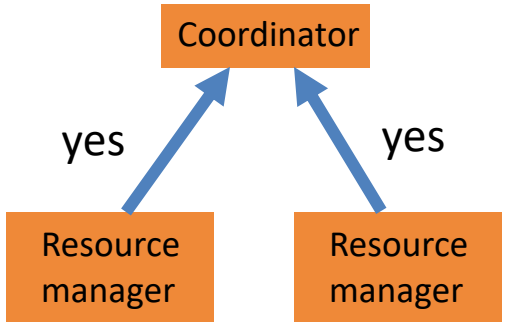
- Advantage:
  - Simple
  - Easy to implement
  - Can basically satisfy the ACID requirement

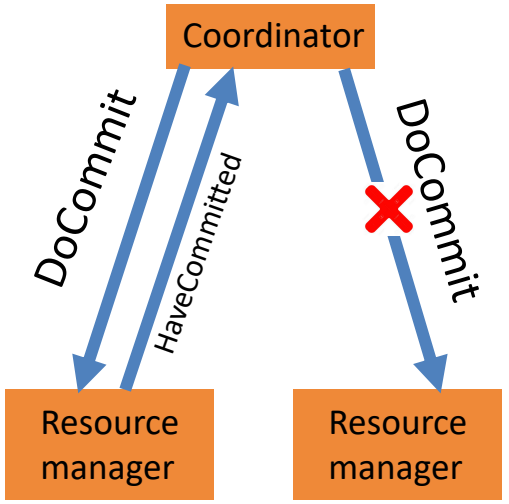
1. Two-phase commit based on the XA protocol

- Disadvantage:
  - Synchronous blocking: During the execution of the two-phase commit algorithm, all participating nodes are transaction blocking, which will block others to access the resource
  - Single point of failure: The 2PC is similar as the centralized algorithm. Once the coordinator fails, the entire system cannot work.

1. Two-phase commit based on the XA protocol

- Disadvantage:
  - Data inconsistency: After the coordinator sends a DoCommit request to the participants during the submission phase, if a local network exception occurs or the coordinator fails during the submission of the submission request, only a part of the participants will receive the submission Request and execute the commit operation, but the other participants who have not received the commit request cannot perform the transaction commit.





2. Three-phase commit protocol

- The three-phase commit protocol (3PC) is an improvement over the two-phase commit (2PC). In order to solve the problem of synchronous blocking and data inconsistency, the 3PC introduced a ***timeout mechanism*** and a ***preparation phase***.

  - Timeout mechanism: If the coordinator or participant does not receive a response from other nodes within the specified time, they will choose to commit or terminate the entire transaction according to the current status.

  - A preparation phase was introduced between the first and second phases. Eliminate some inconsistencies during the pre-commit stage.

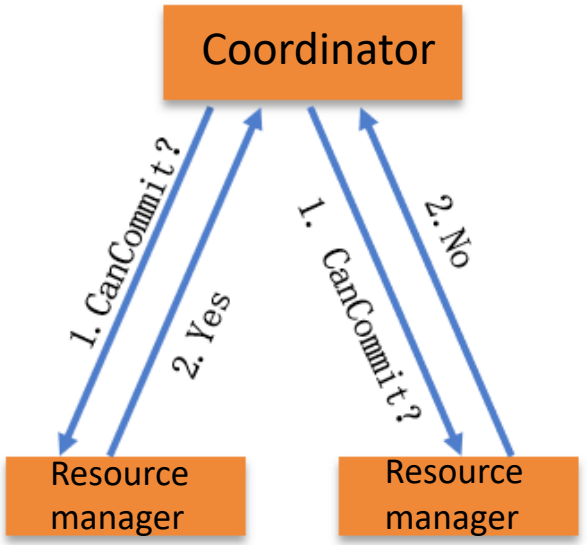
2. Three-phase commit protocol

- The three-phase commit protocol (3PC) contains three phases:

1. CanCommit phase
2. Precommit phase
3. DoCommit phase

2. Three-phase commit protocol

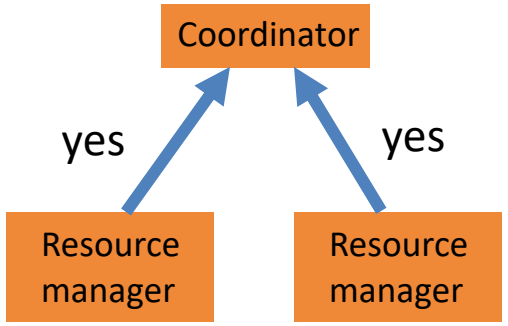
- Phase 1: CanCommit
  1. Coordinator sends CanCommit requests to resource manager and waits for reply
  2. Resource manager receives requests and do the transactions.
     - Reply Yes, means approving the operation;
     - Reply No, means stop the operating.



2. Three-phase commit protocol

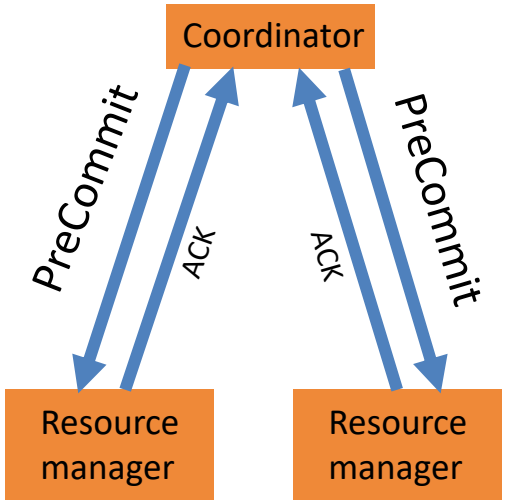
- Phase 2: PreCommit

1. If all replies are yes, Coordinator sends "PreCommit" operation



2. The resource manager will do the transaction operations after receiving "PreCommit", and save "Undo" & "Redo" information into logs

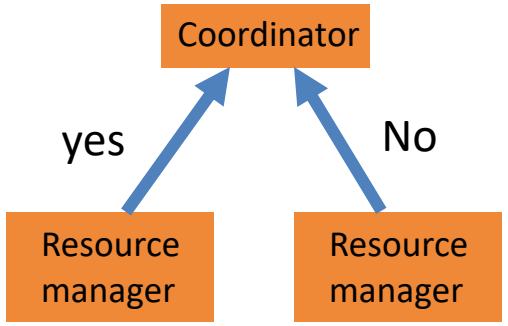
3. If the resource manager finishes transactions, returns "ACK" message and waits for final command



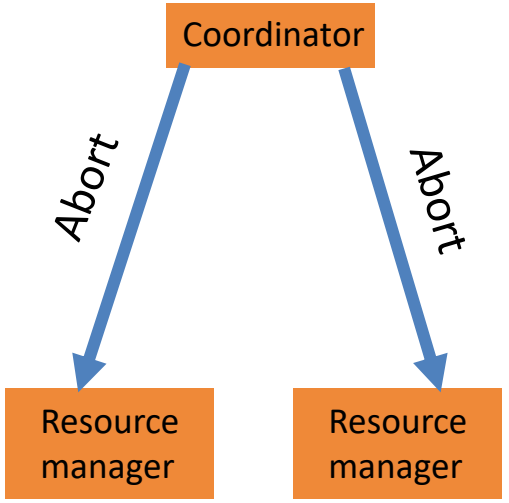
2. Three-phase commit protocol

- Phase 2: PreCommit

1. If one of replies is no, or waiting is timeout, Coordinator sends "Abort" operation to terminate transaction

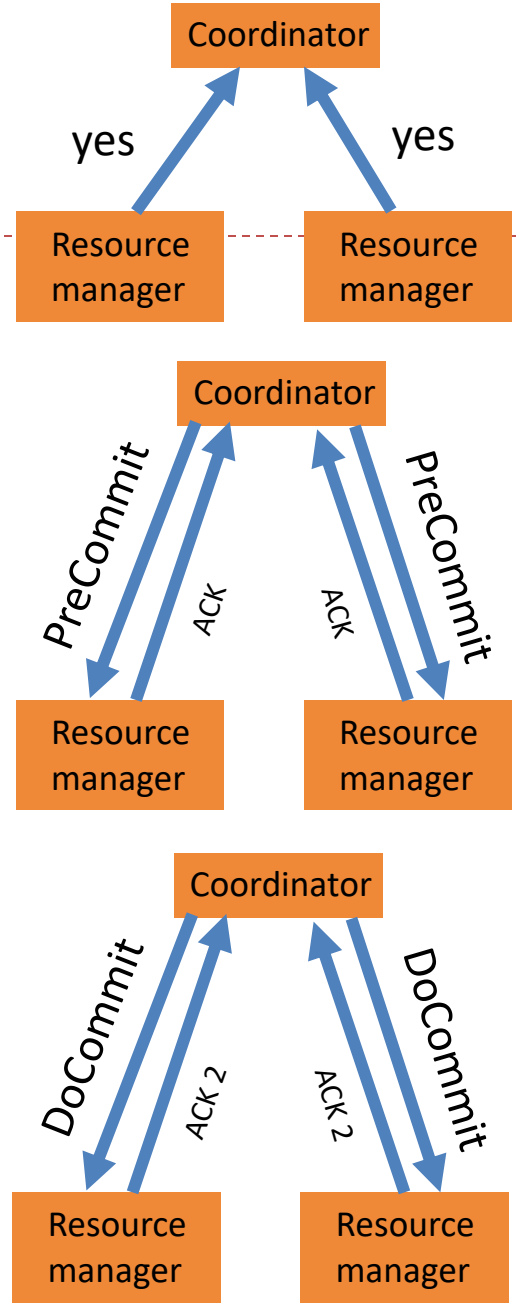


2. The resource manager performs the operations to terminate transaction when receiving "Abort" message



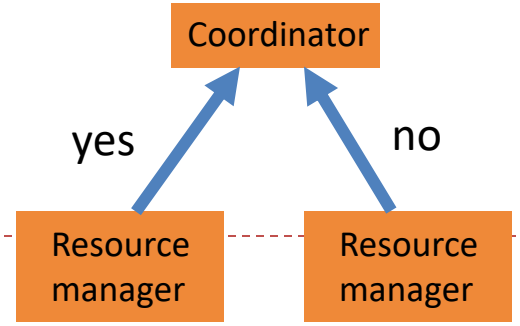
2. Three-phase commit protocol

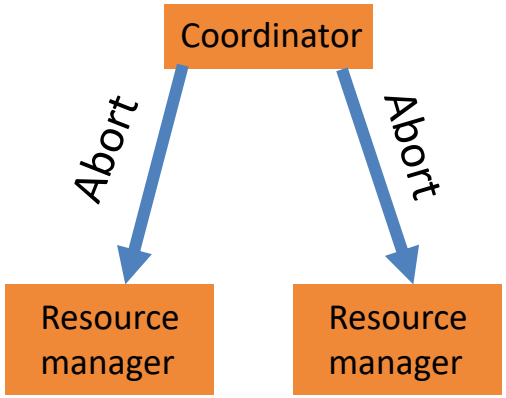
- Phase 3: DoCommit (for commit)
  1. The coordinator sends “DoCommit” message to all nodes when receiving ACK response.
  2. After the participants receive the “DoCommit” message, they formally commit the transaction and release all locked resources.
  3. After the participant submits the transaction, a new ACK response is sent to the coordinator.
  4. The coordinator completes the transaction after receiving the ACK responses from all participants.

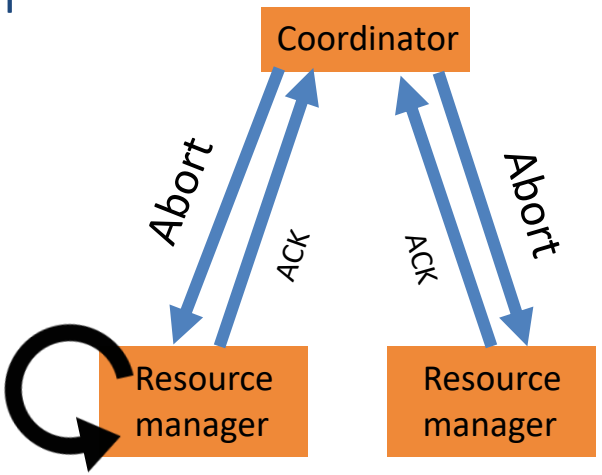


2. Three-phase commit protocol

- Phase 3: DoCommit (for termination)
  1. The coordinator sends “Abort” message to all participants.
  2. After the participants receive the “Abort”, they undo all transactions in Phase 2 and release all locked resources.
  3. After the participant undo the transaction, an ACK response is sent to the coordinator.
  4. The coordinator completes the transaction after receiving the ACK responses from all participants.

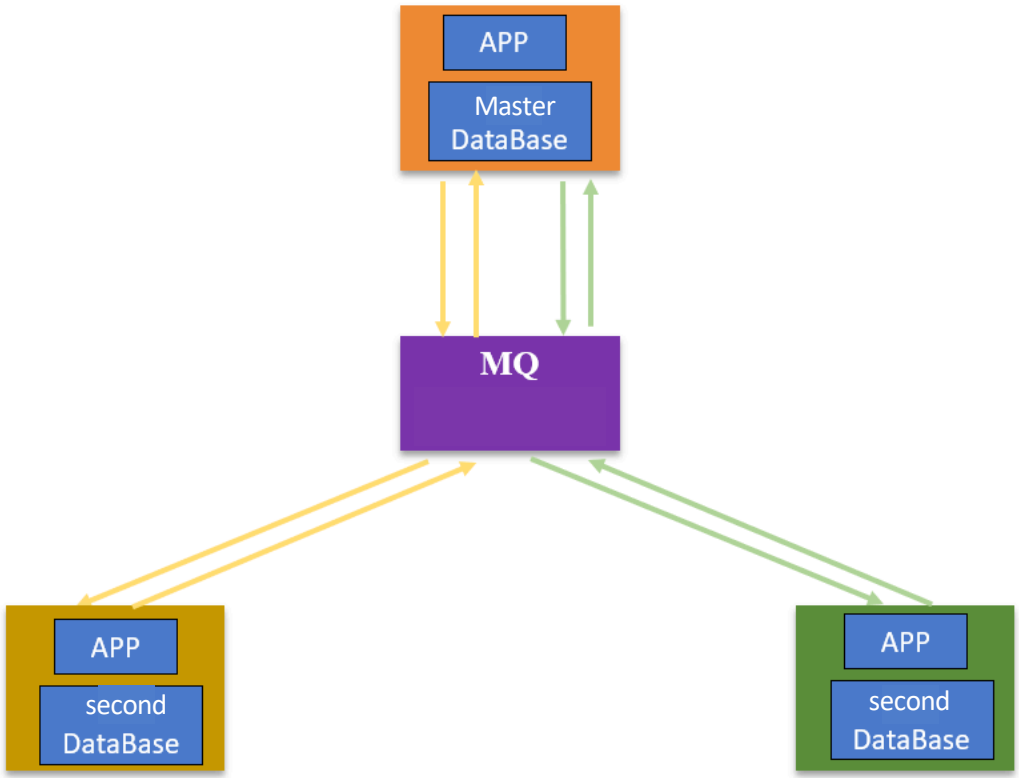




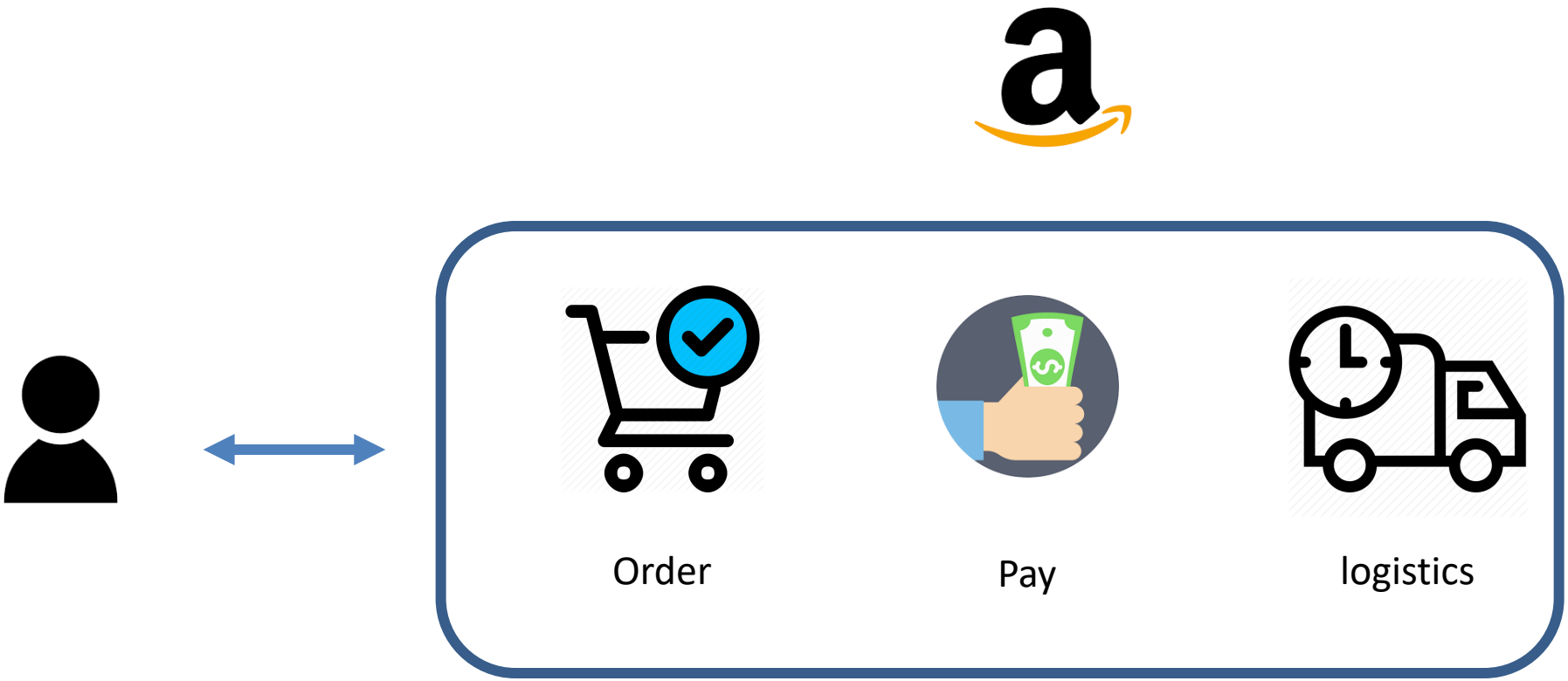


3. A distributed message-based consistency

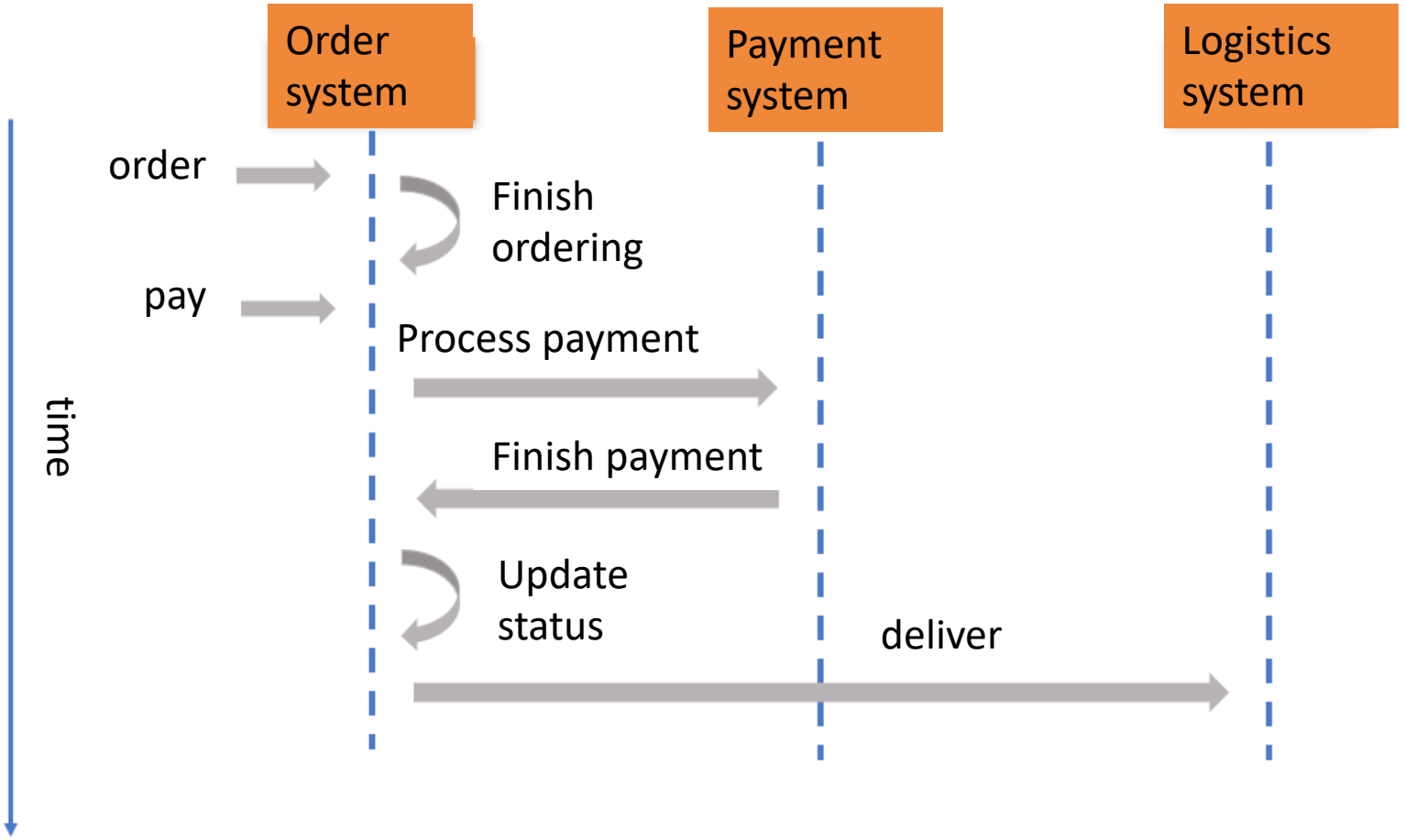
- This design introduces a message middleware (Message Queue, MQ) for message passing between multiple applications



3. A distributed message-based consistency

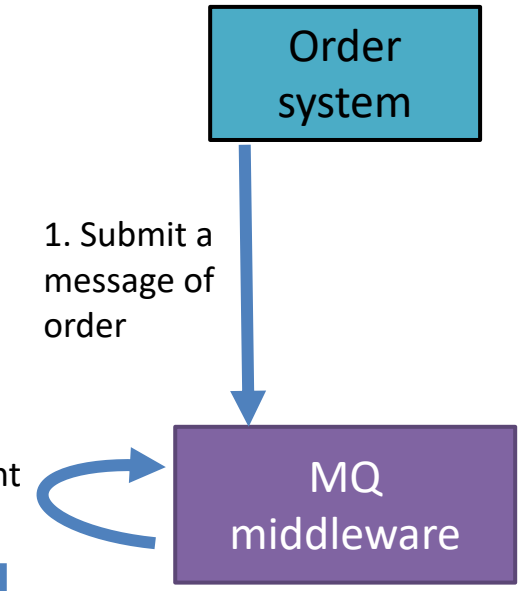


3. A distributed message-based consistency



3. A distributed message-based consistency

If the order is successful



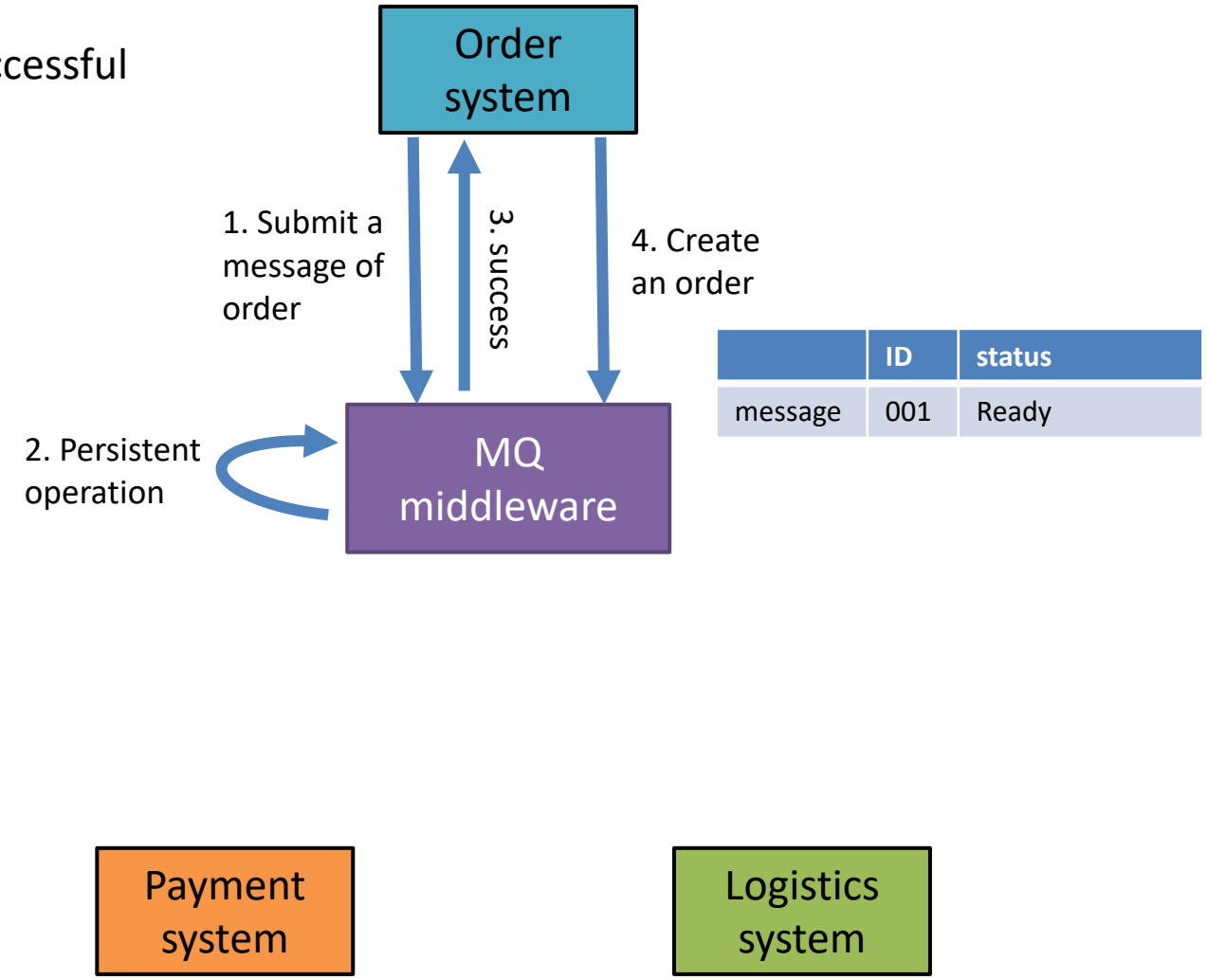
	ID	status
message	001	Wait to send

Payment system

Logistics system

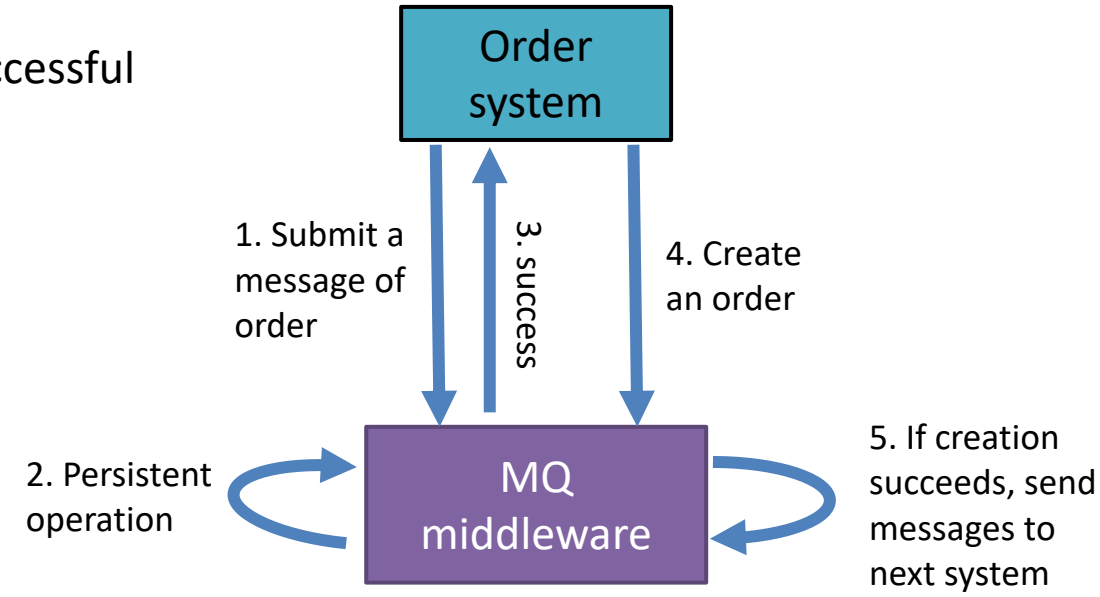
3. A distributed message-based consistency

If the order is successful



3. A distributed message-based consistency

If the order is successful



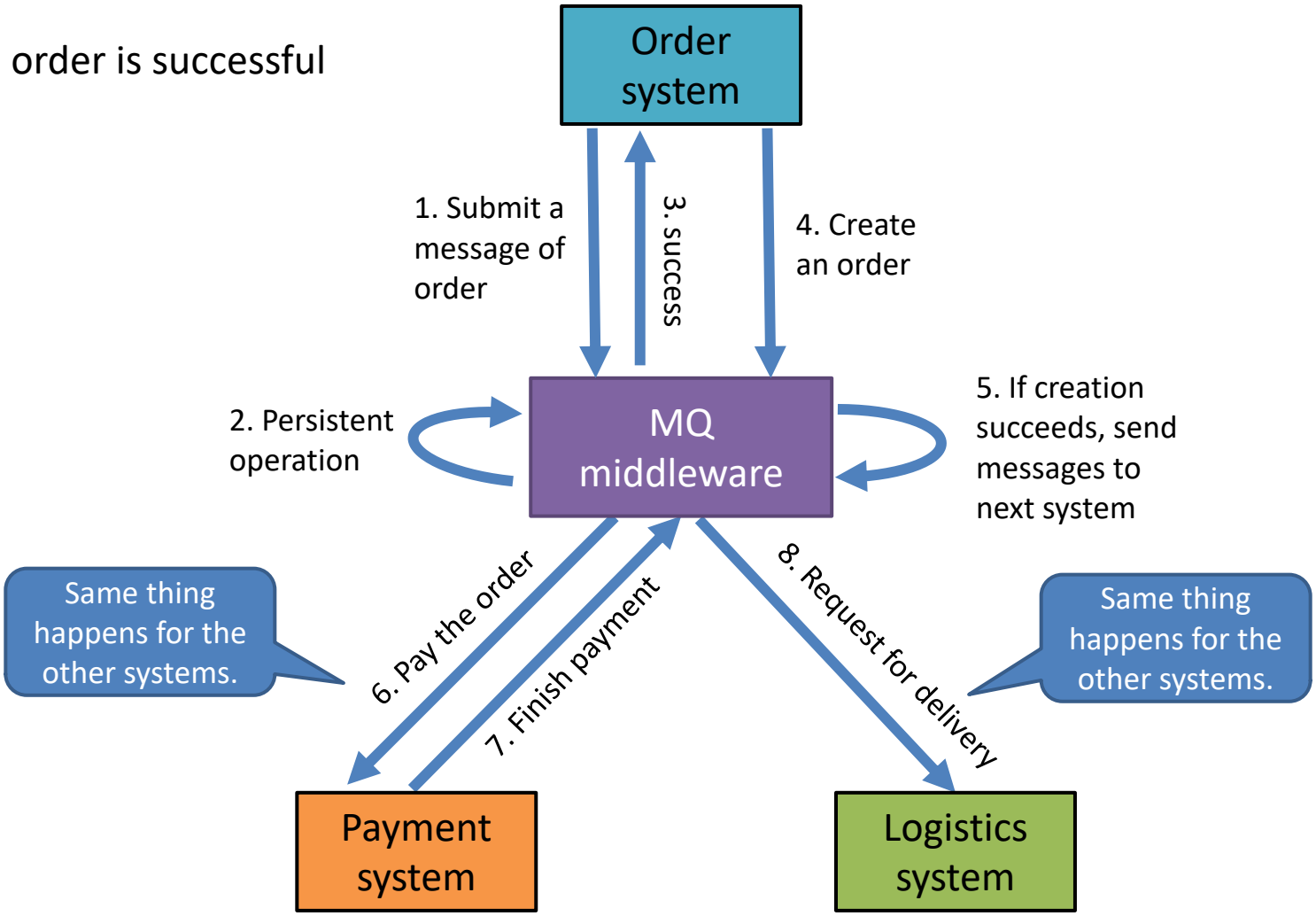
	ID	status
message	001	Sent

Payment system

Logistics system

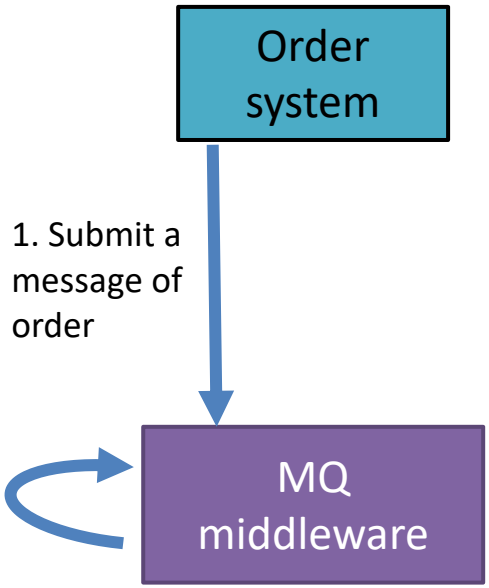
3. A distributed message-based consistency

If the order is successful



3. A distributed message-based consistency

If the order fails



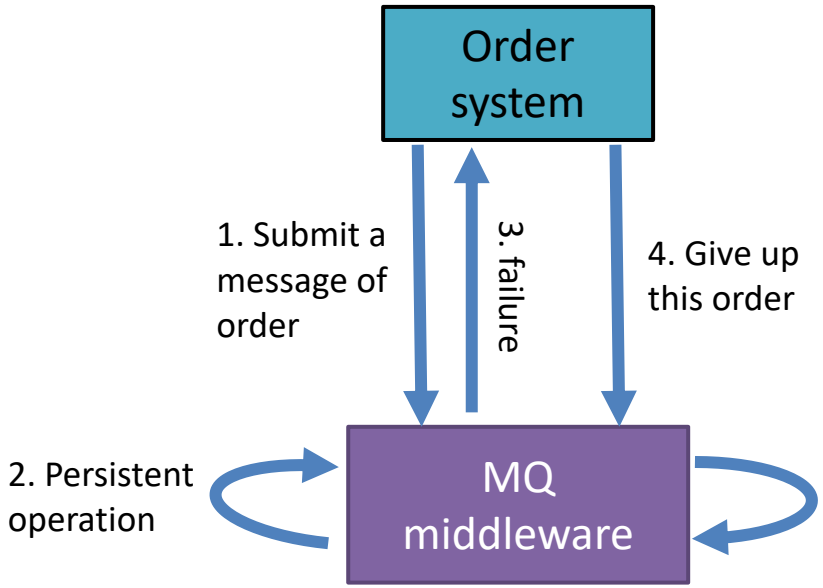
	ID	status
message	001	Wait to send

Payment system

Logistics system

3. A distributed message-based consistency

If the order fails



	ID	status
message	001	deleted

Payment system

Logistics system

Comparison

	Two-phase commit based on the XA protocol	Three-phase commit protocol	A distributed message-based consistency
Execution model	Synchronization	Synchronization	Asynchronization
Blocking or not	Yes	No	No
Single node failure risk?	Yes	No	No
Concurrency	2PC < 3PC < Message-based		
Performance (throughput)	2PC < 3PC < Message-based		