Microservice Architecture

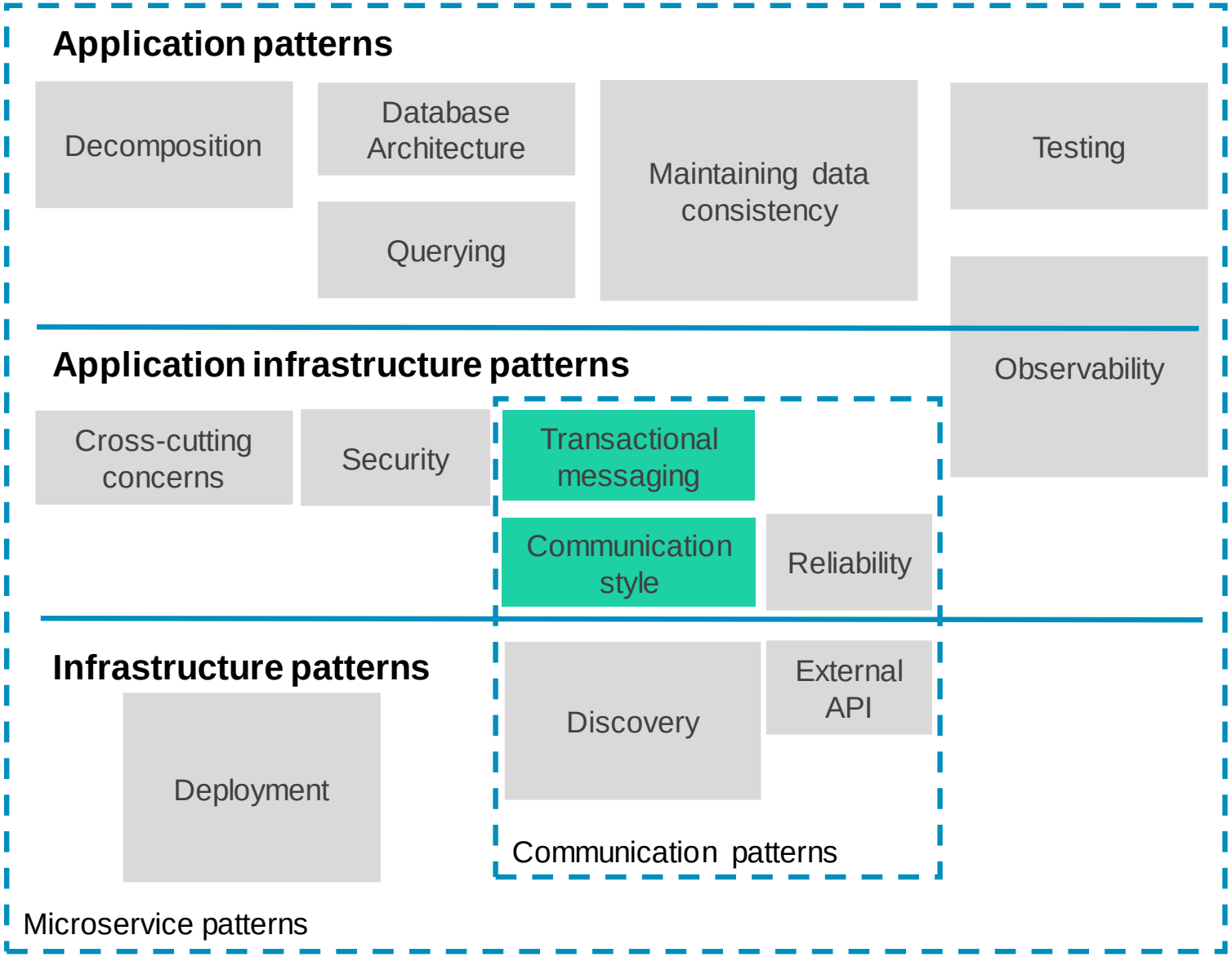
IPC - Asynchronous communication

Asynchronous Messaging

By the end of this course, you will be able to

1. **Implement** different **communication styles** using the **Asynchronous Messaging**.
2. **Create** a **specification for APIs** relying on asynchronous messaging.
3. **Choose** and **Use** a **message broker** for your asynchronous communication.
4. **Deal with** some famous **design issues** when going asynchronous.

Problem areas to solve



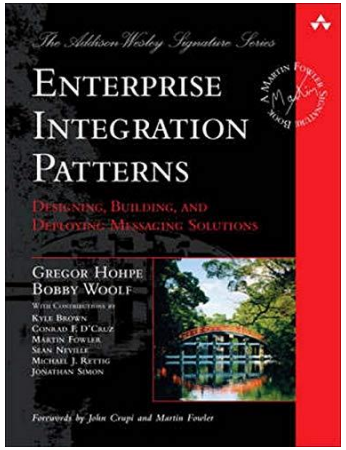
Agenda

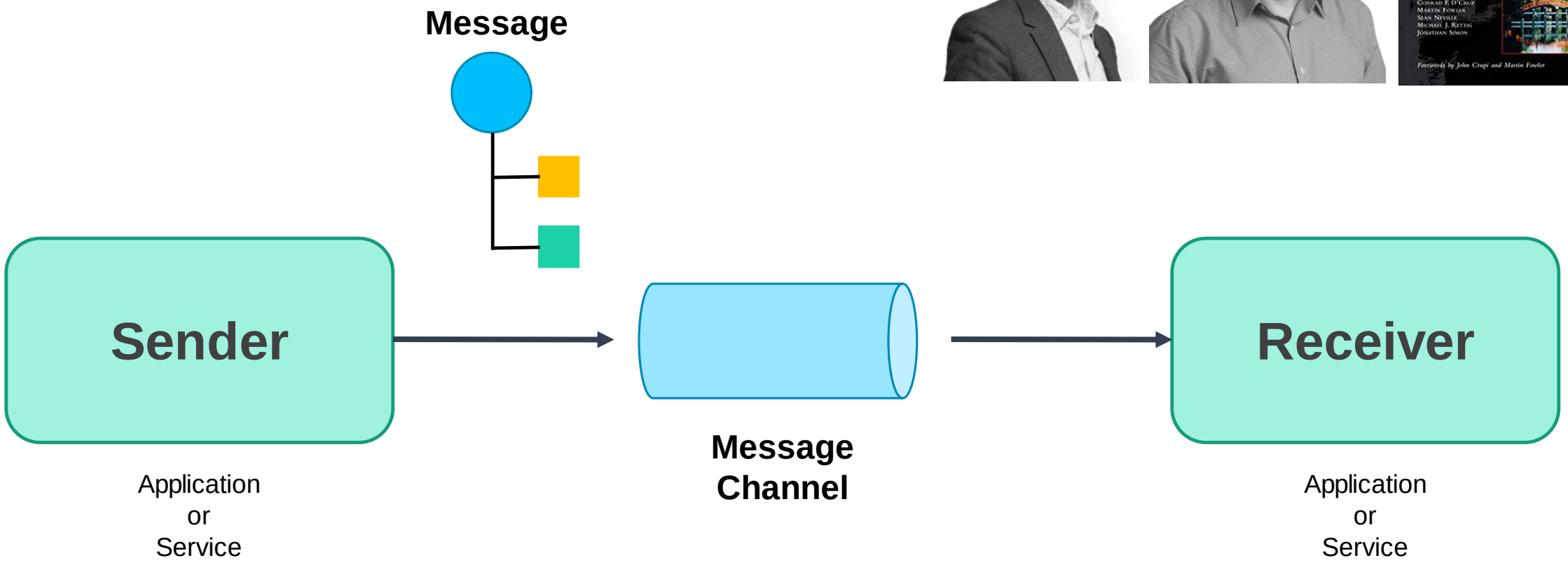
Communicating using the Asynchronous Messaging pattern

1. Overview of Messaging
2. Implementing the interaction styles
3. Creating an API specification
4. Using a Message Broker
5. Design issues
    a. Message ordering
    b. Duplicate messages
    c. Transactional Messaging

Asynchronous Messaging

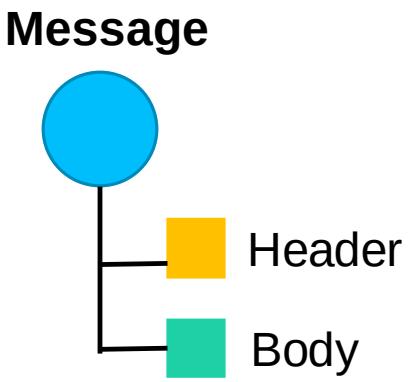
Overview of messaging





Asynchronous Messaging

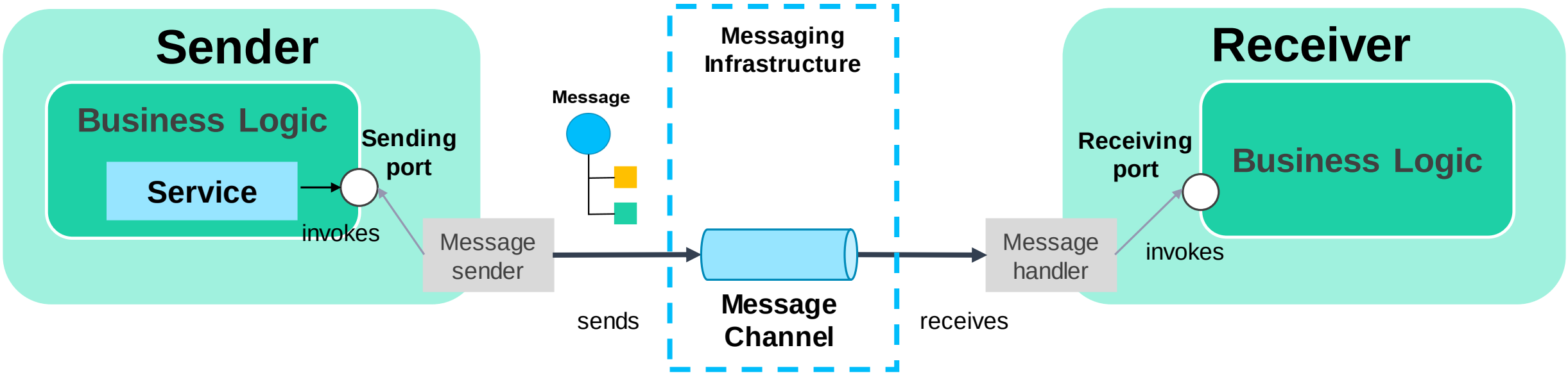
Overview of messaging: **Messages**



- Header
  - **metadata:** data description
  - **message-id** *(unique)*
  - **return address** *(optional)*
- Body
  - **Document** D
  - **Command** C
  - **Event** E

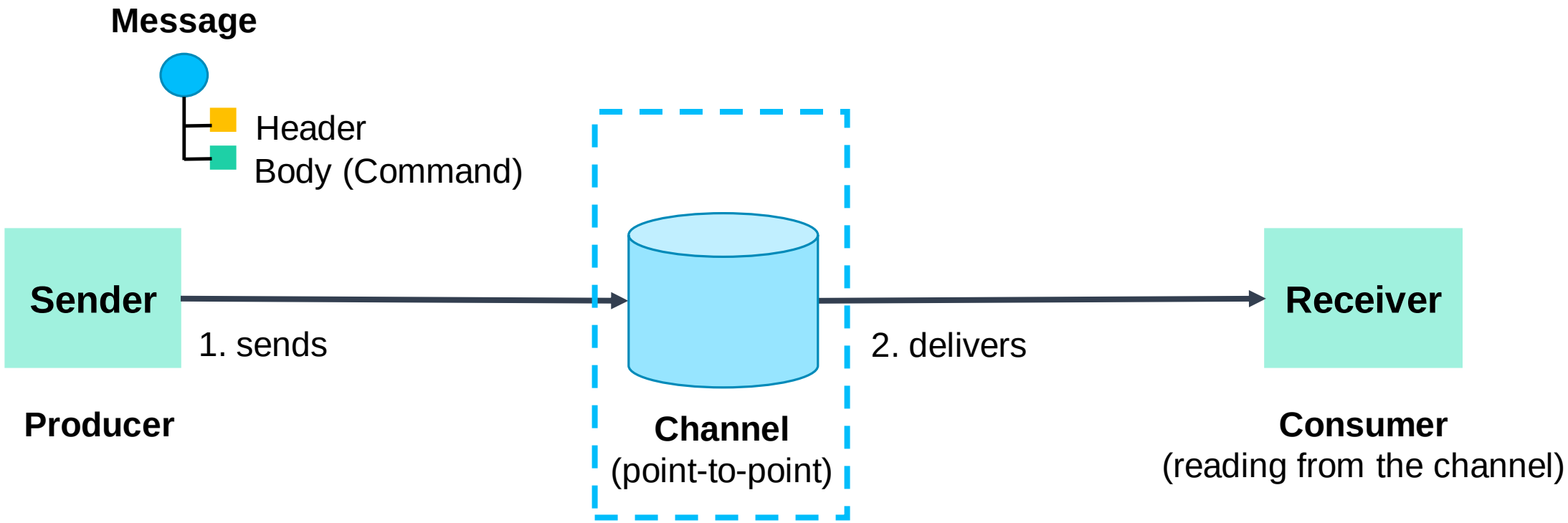
Asynchronous Messaging

Overview of messaging: **Channels**



Asynchronous Messaging

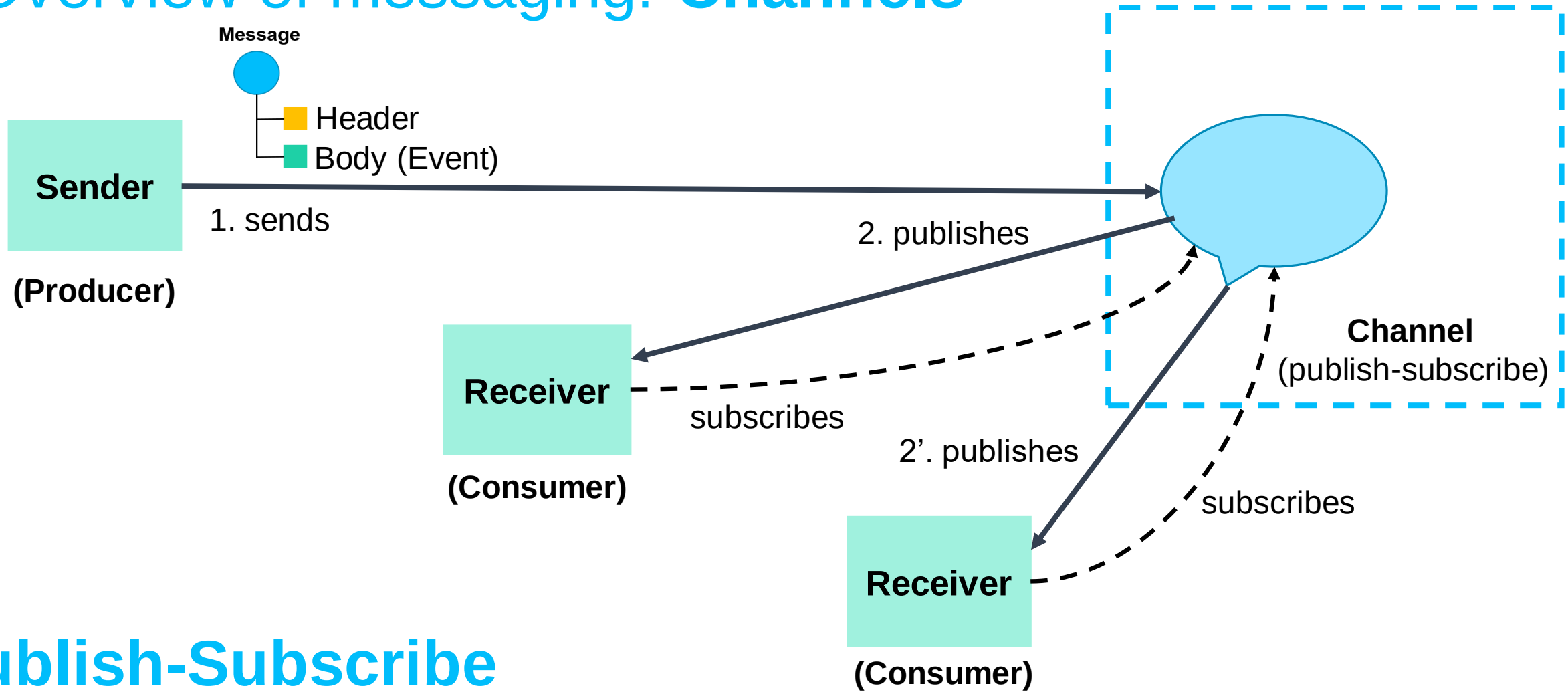
Overview of messaging: **Channels**



Point-to-Point

Asynchronous Messaging

Overview of messaging: **Channels**



Publish-Subscribe

Agenda

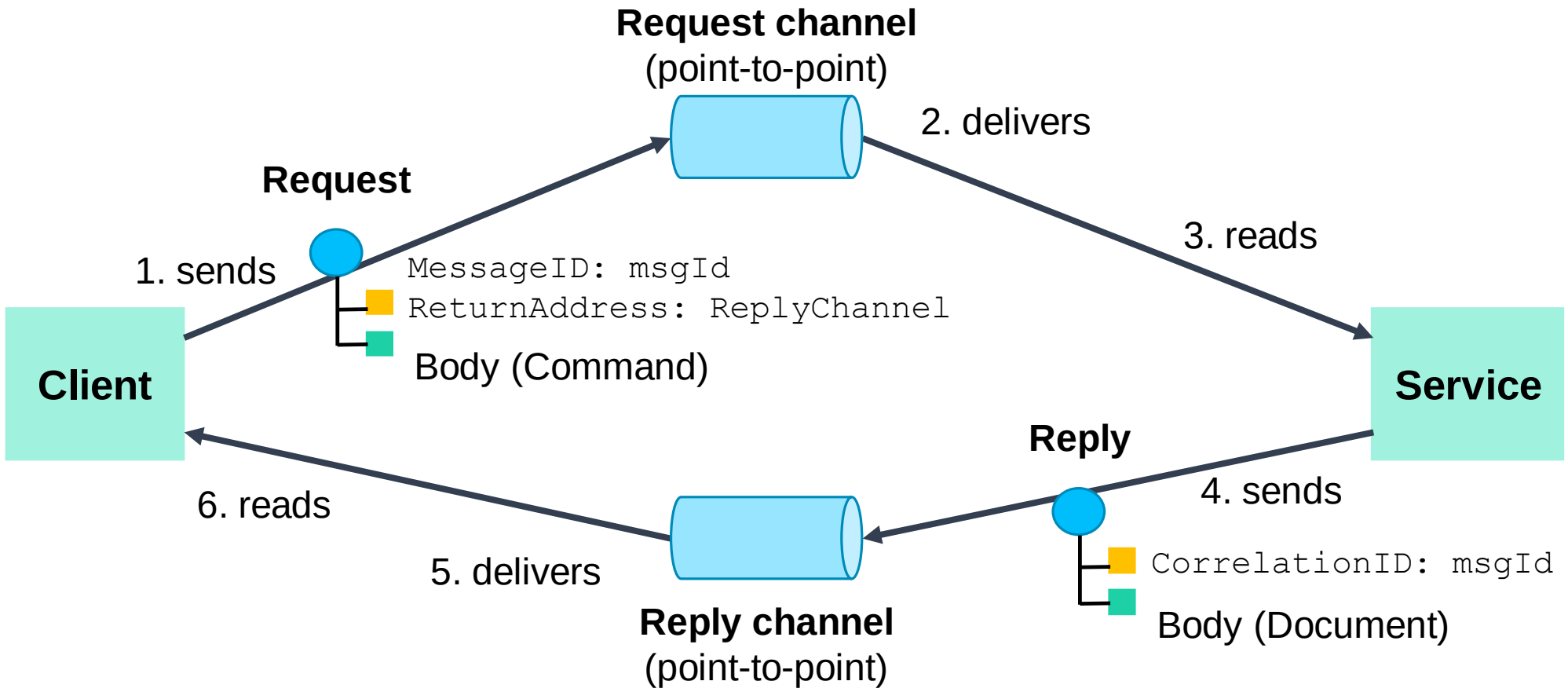
Communicating using the Asynchronous Messaging pattern

1. Overview of Messaging
2. Implementing the interaction styles
3. Creating an API specification
4. Using a Message Broker
5. Design issues
    a. Message ordering
    b. Duplicate messages
    c. Transactional Messaging

Asynchronous Messaging

Implementing the interaction styles using messaging

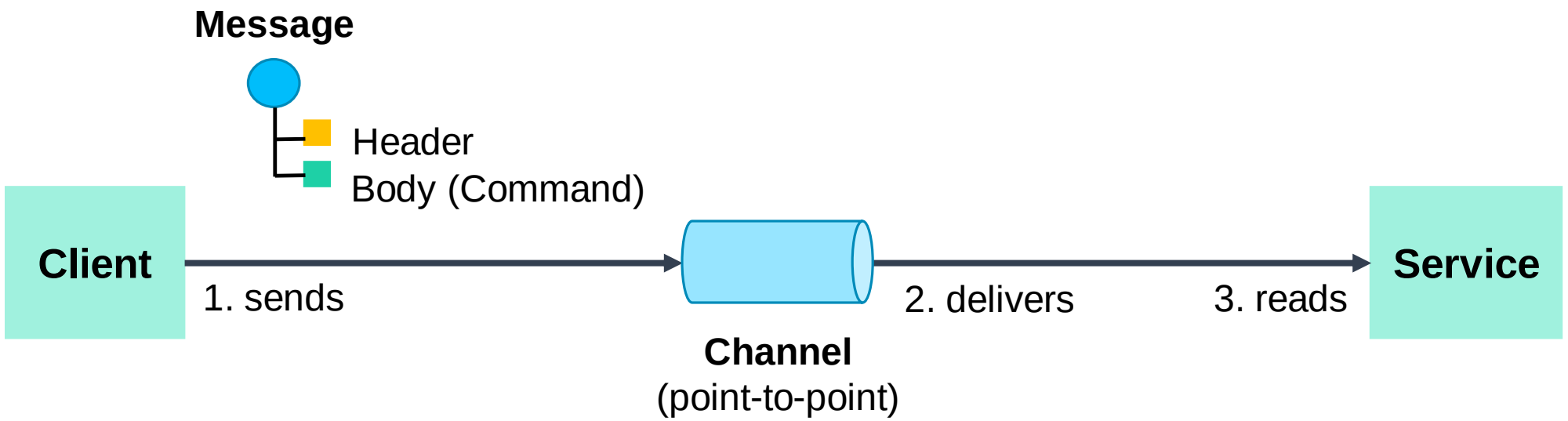
Request/Response and **Asynchronous Request/Response**



Asynchronous Messaging

Implementing the interaction styles using messaging

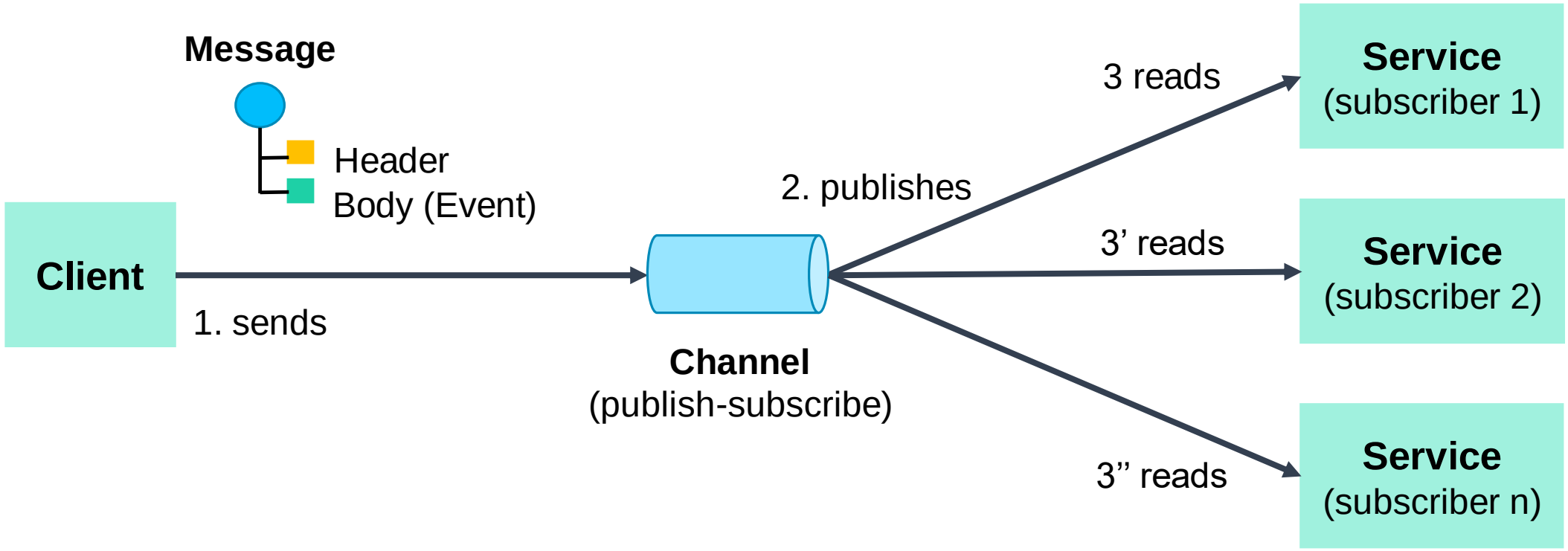
One-way Notification



Asynchronous Messaging

Implementing the interaction styles using messaging

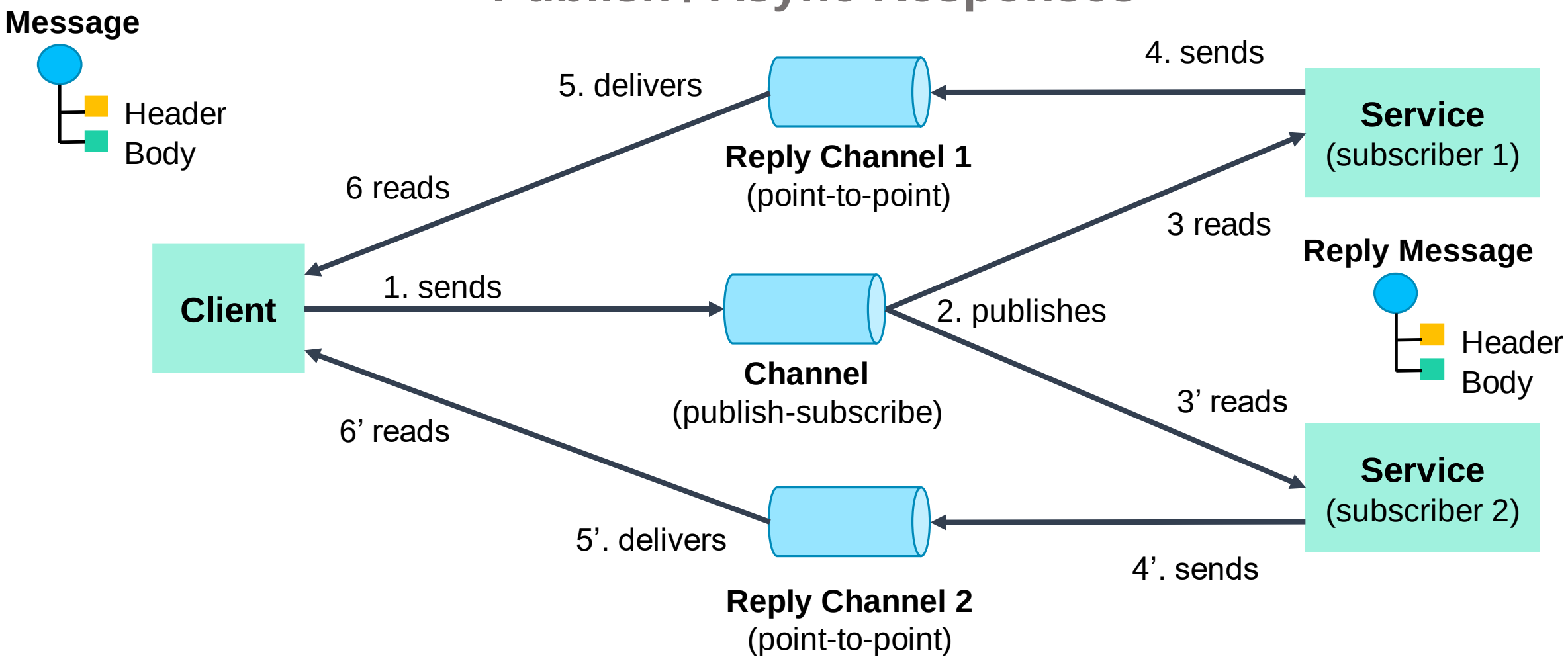
Publish / Subscribe



Asynchronous Messaging

Implementing the interaction styles using messaging

Publish / Async Responses



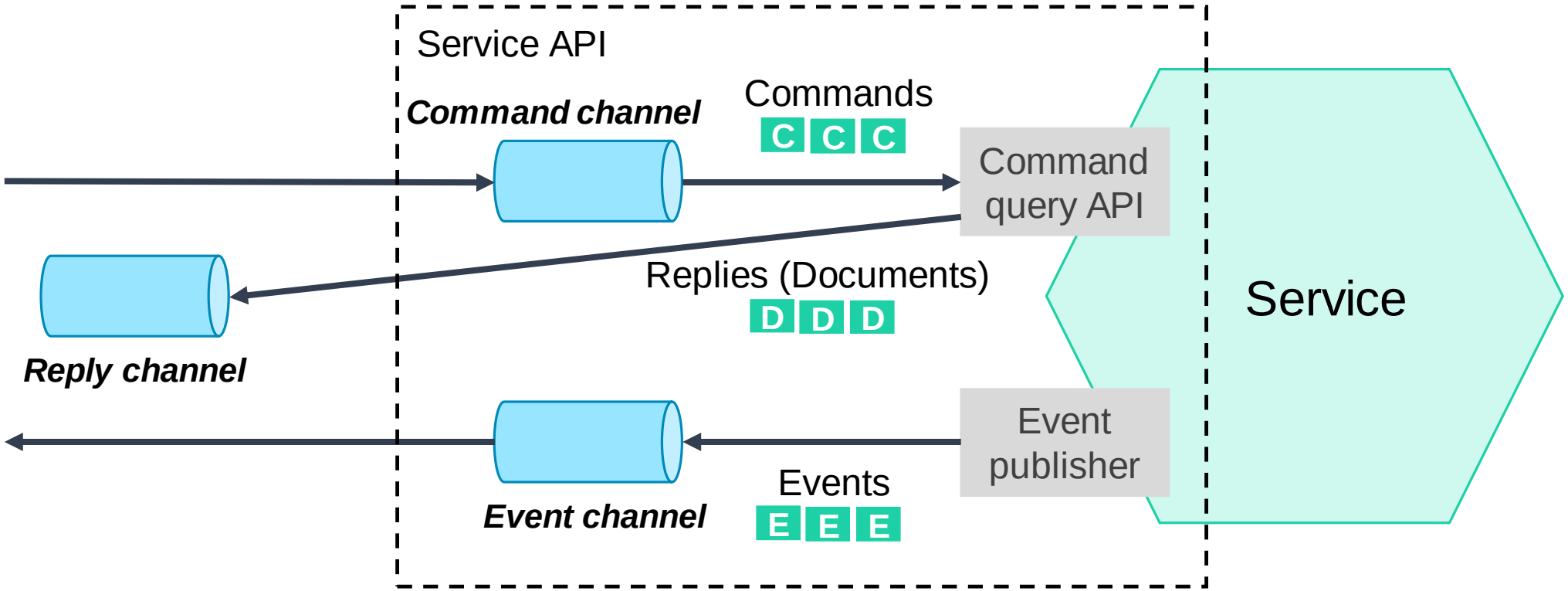
Agenda

Communicating using the Asynchronous Messaging pattern

1. Overview of Messaging
2. Implementing the interaction styles
3. Creating an API specification
4. Using a Message Broker
5. Design issues
    a. Message ordering
    b. Duplicate messages
    c. Transactional Messaging

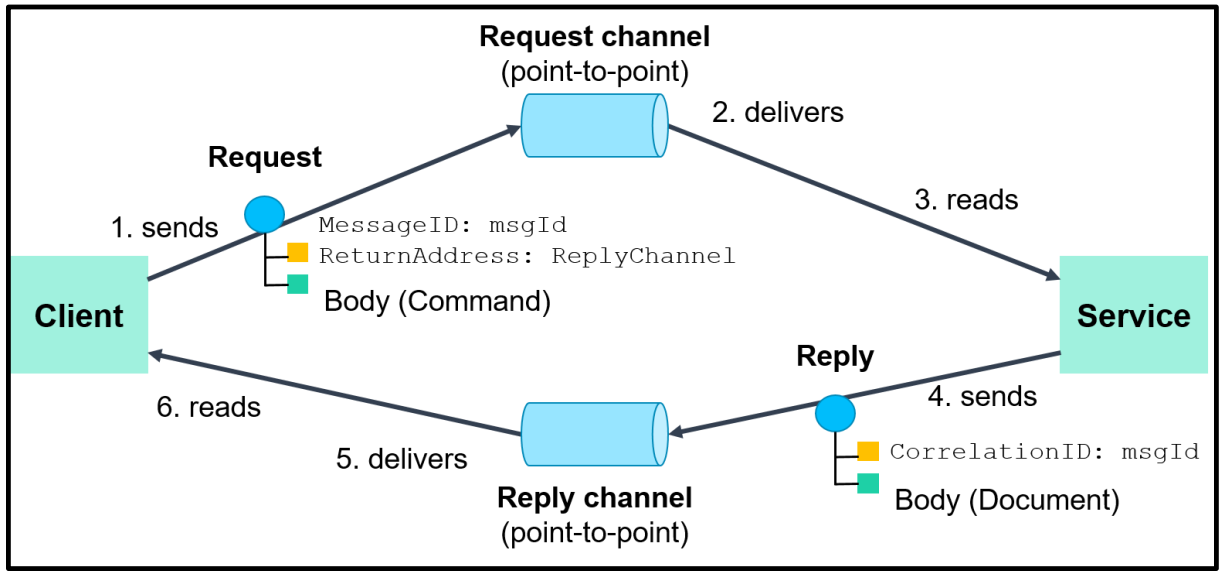
Asynchronous Messaging

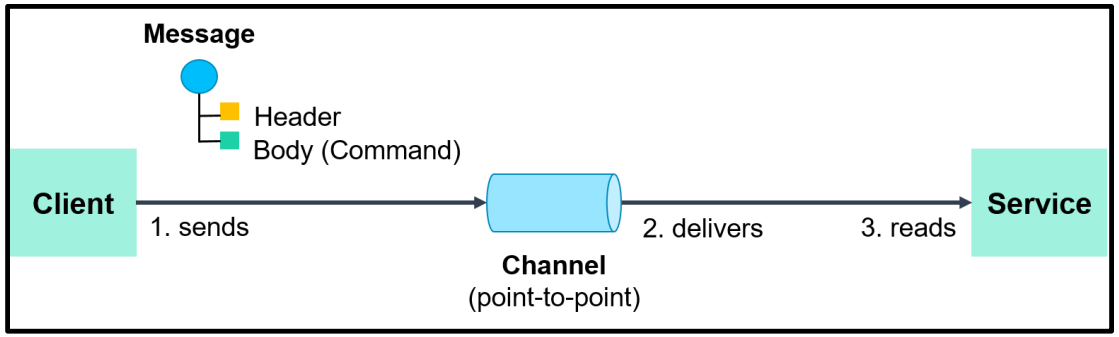
Creating an API specification for a messaging-based service API



Asynchronous Messaging

Documenting **asynchronous operations**





Request / Async Response

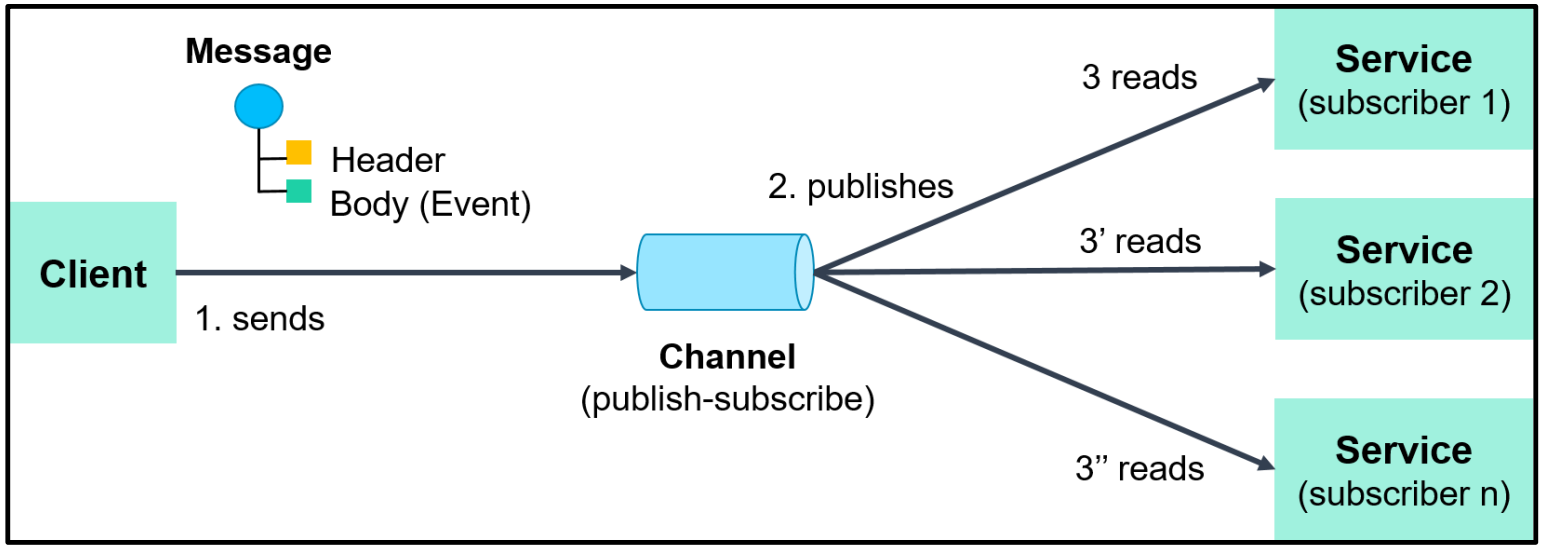
- The service's **command message channel**
- The **types** and **formats** of the **command message types**
- The **types** and **formats** of the **reply messages**

One-Way Notification

- The service's **command message channel**
- The **types** and **formats** of the **command message types**

Asynchronous Messaging

Documenting **published events**



Publish / Subscribe

- The **event channel**
- The **types** and **formats** of the **event messages**

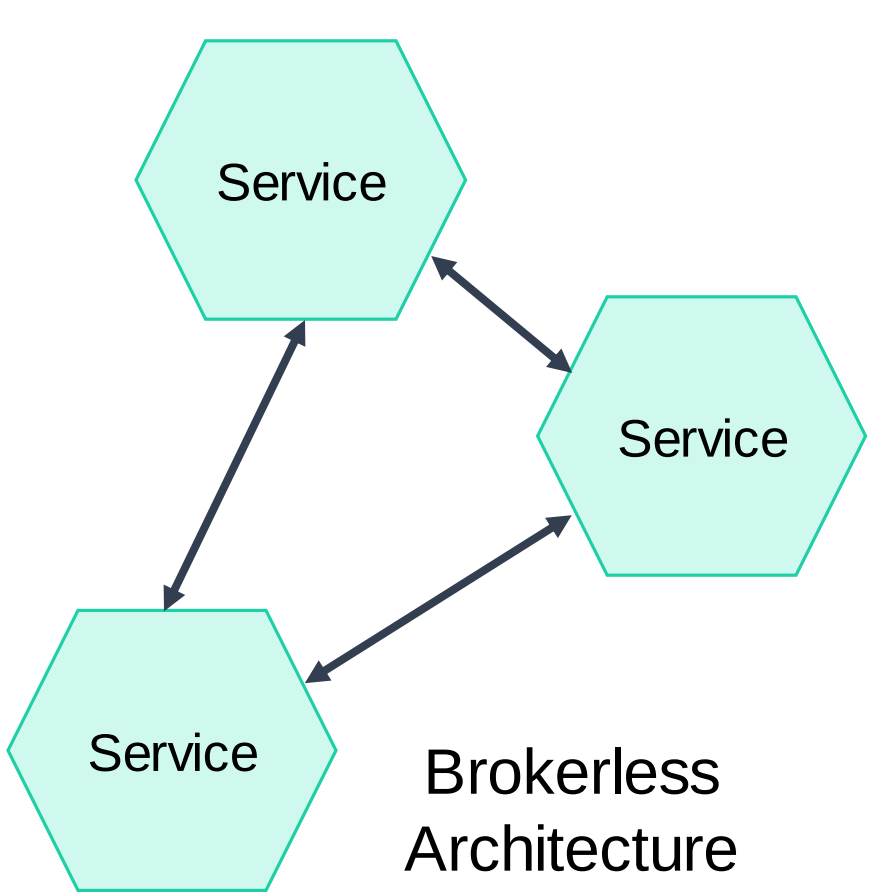
Agenda

Communicating using the Asynchronous Messaging pattern

1. Overview of Messaging
2. Implementing the interaction styles
3. Creating an API specification
4. Using a Message Broker
5. Design issues
    a. Message ordering
    b. Duplicate messages
    c. Transactional Messaging

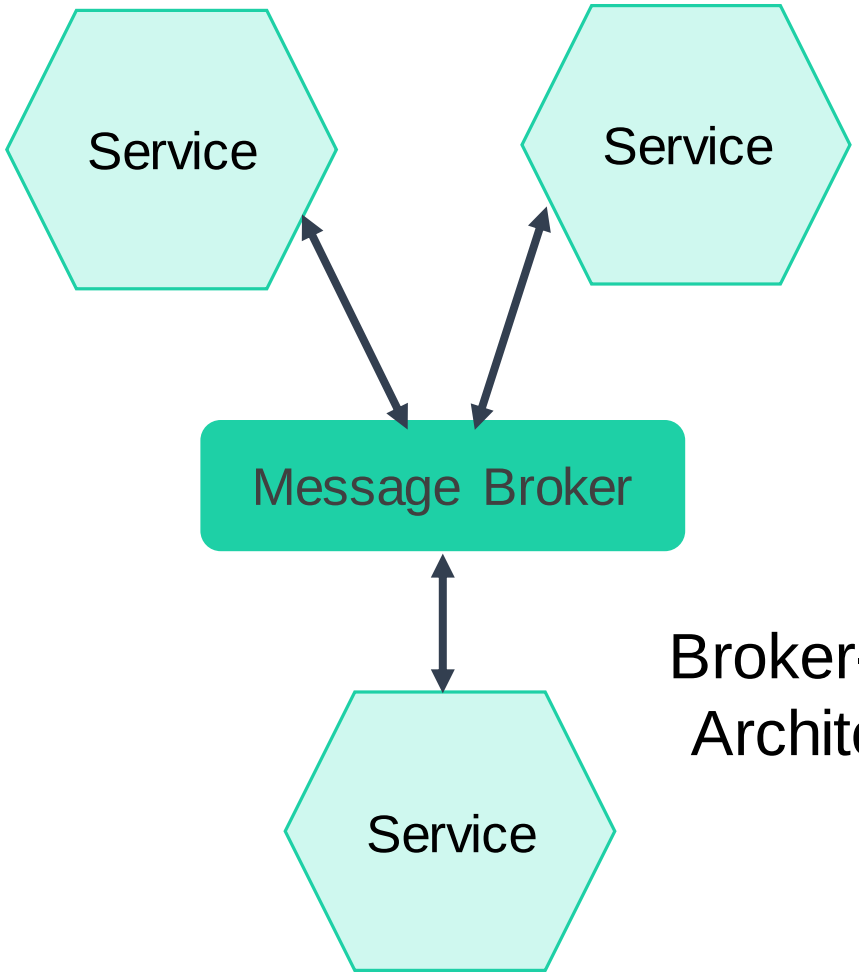
Asynchronous Messaging

Using a Message Broker? Or not?



Brokerless Architecture

vs.



Broker-based Architecture

Asynchronous Messaging

Brokerless messaging

ØMQ

TCP

UNIX domain socket

Multicast

+ Lighter network

+ Better latency

+ Less complexity

+ No Bottleneck

+ No SPOF

- Need for service discovery

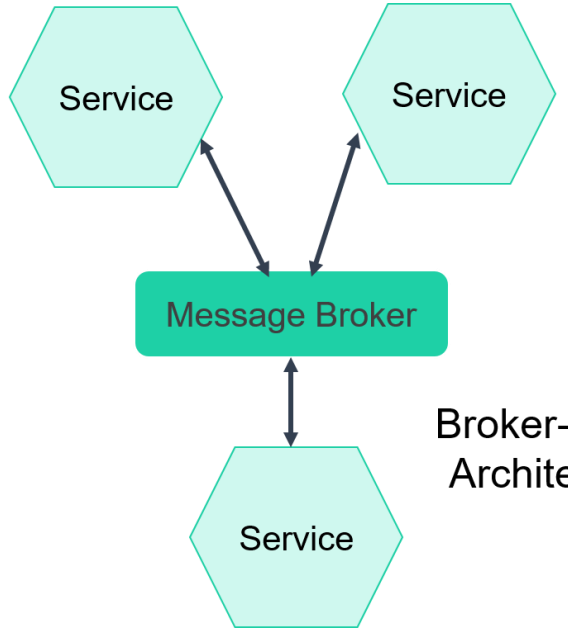
- Reduced availability

Asynchronous Messaging

Broker-based messaging

The sender doesn't need to know the network location of the consumer

A message broker buffers messages until the consumer is able to process them



Broker-based Architecture

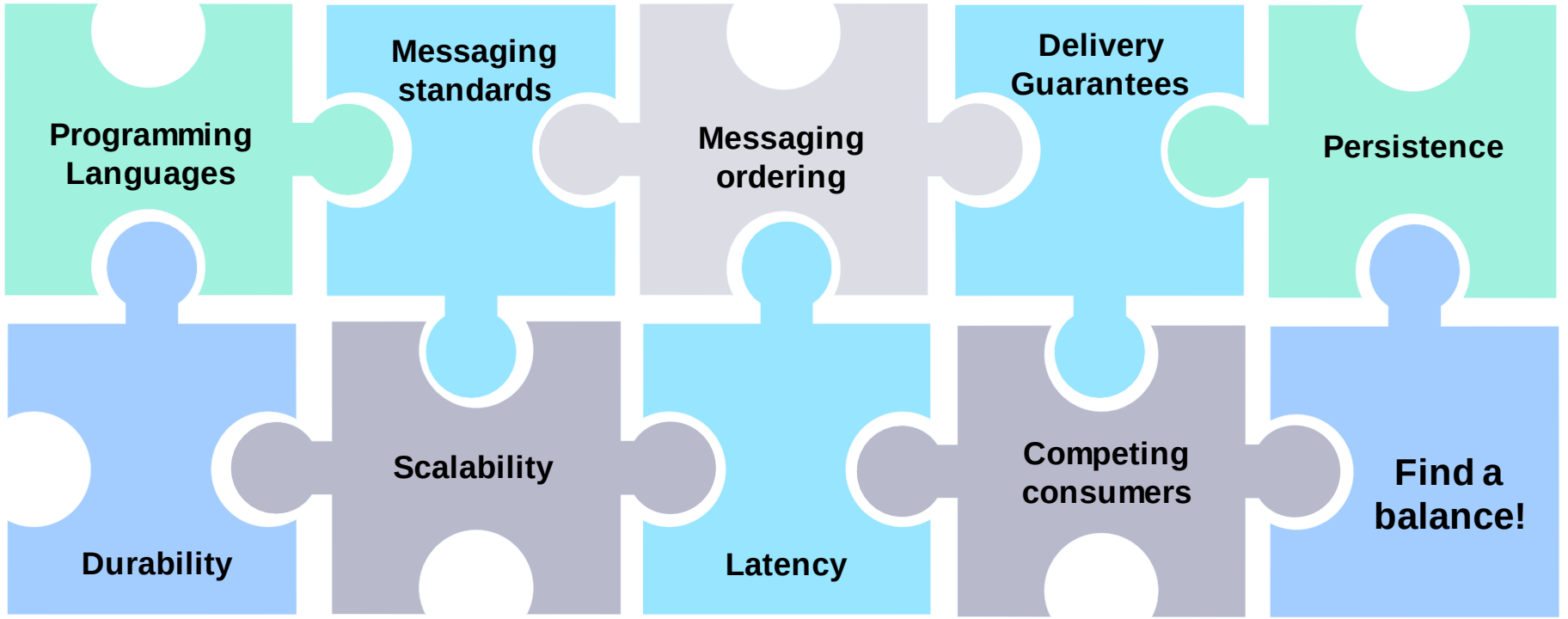
RabbitMQ™

APACHE kafka™

Apache ACTIVEMQ

Asynchronous Messaging

Broker-based messaging: selecting a message broker



Asynchronous Messaging

Implementing message channels using a message broker

Message Broker	Point-to-Point Channel	Publish-Subscribe Channel
JMS message brokers **(ActiveMQ)**	Queue	Topic
AMQP message brokers **(RabbitMQ)**	Exchange + Queue	Fanout exchange and a queue per consumer
Apache **Kafka**	Topic	Topic
AWS **Kinesis**	Stream	Stream
AWS **SQS**	Queue	---
AWS **SNS**	---	Topic

Asynchronous Messaging

Benefits of broker-based messaging

+ Loose coupling

+ Message buffering

+ Flexible communication

+ Explicit IPC

Asynchronous Messaging

Downsides of broker-based messaging

- Potential performance bottleneck
- Potential single point of failure (SPOF)
- Additional operational complexity

Agenda

Communicating using the Asynchronous Messaging pattern

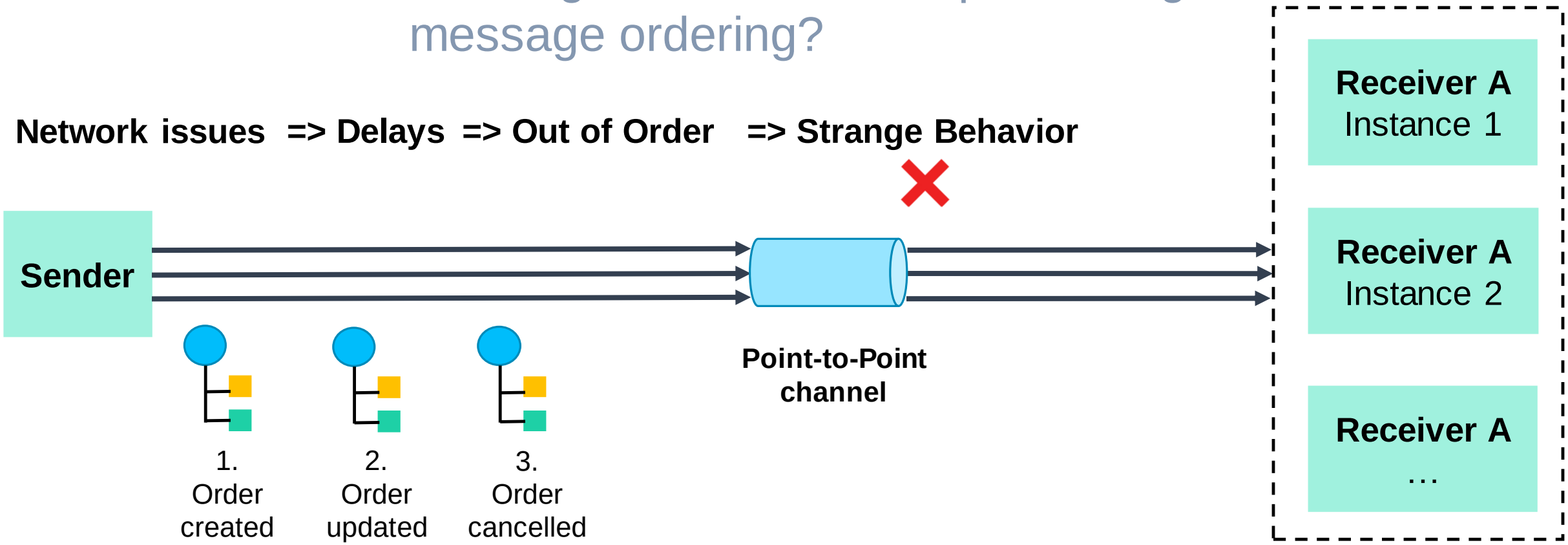
1. Overview of Messaging
2. Implementing the interaction styles
3. Creating an API specification
4. Using a Message Broker
5. Design issues
    a. Message ordering
    b. Duplicate messages
    c. Transactional Messaging

Asynchronous Messaging

A. Competing receivers and message ordering

Problem Statement

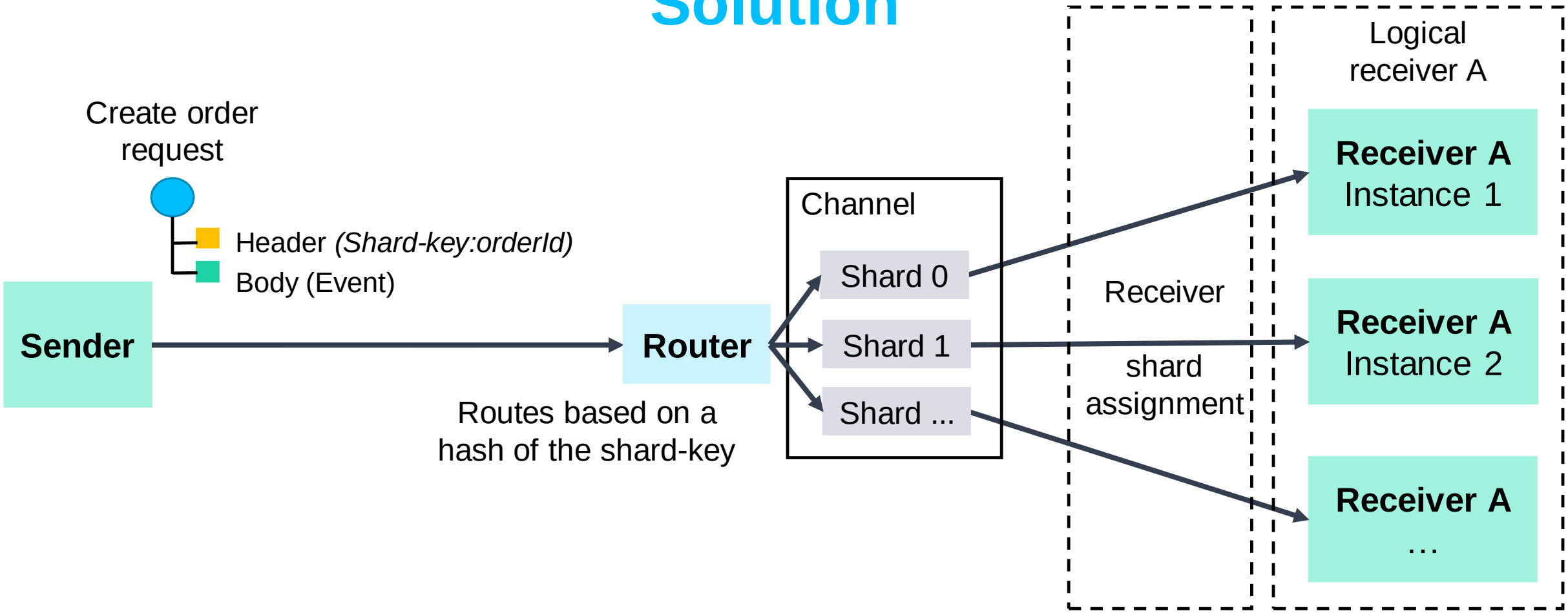
How to scale out message receivers while preserving message ordering?



Asynchronous Messaging

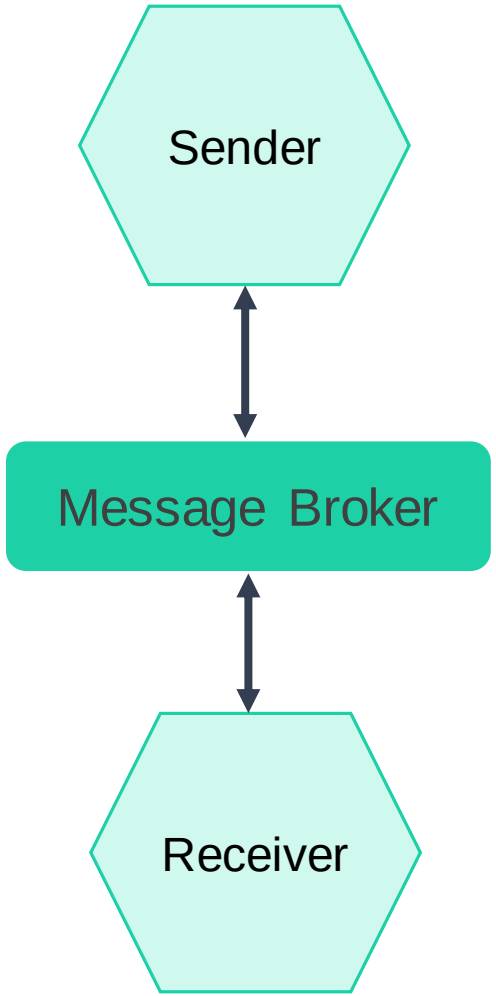
A. Competing receivers and message ordering

Solution



Asynchronous Messaging

B. Handling duplicate messages



Problem Statement

Once-messaging

Costly

At least once

Client failure

Network failure

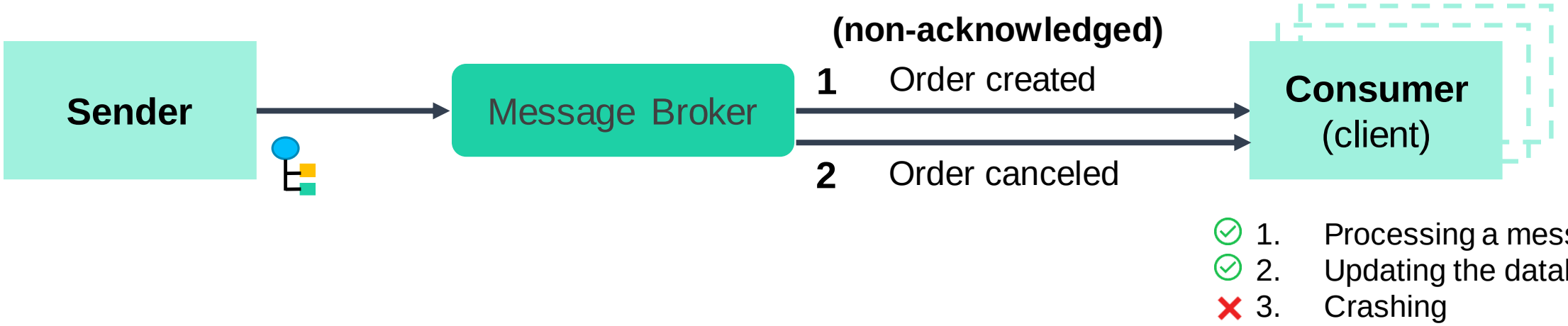
Message Broker failure

Message delivered multiple times

Asynchronous Messaging

B. Handling duplicate messages

Problem Statement



The Message Broker should redeliver both ***Order created*** and ***Order cancelled***
To avoid a non-desired processing *(undo the cancelling of the Order)*

Message not acknowledged

Asynchronous Messaging

B. Handling duplicate messages

Solution

Solution 1:

Write idempotent message handlers

Application logic is ***idempotent*** if calling it multiple times with the same input values has no additional effect.

Examples:

- Cancelling an already-cancelled order
- Creating an order with a client-supplied ID
- etc.

Result: Idempotent message handler can be safely executed multiple times.
(Assuming that the message broker preserves ordering)

❌ **Application logic** is often **not idempotent**

❌ **Message Broker doesn't preserve ordering**

Out-of-order messages

Bugs !

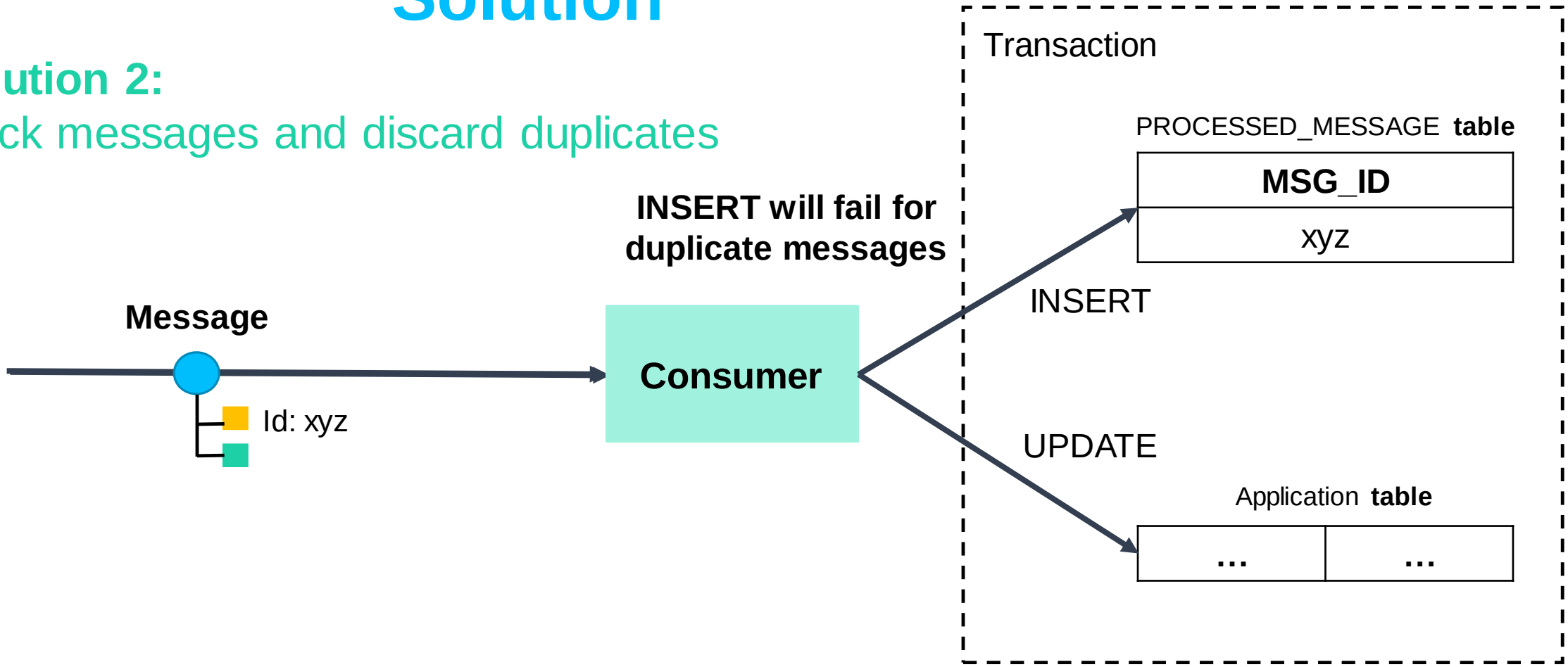
Asynchronous Messaging

B. Handling duplicate messages

Solution

Solution 2:

Track messages and discard duplicates

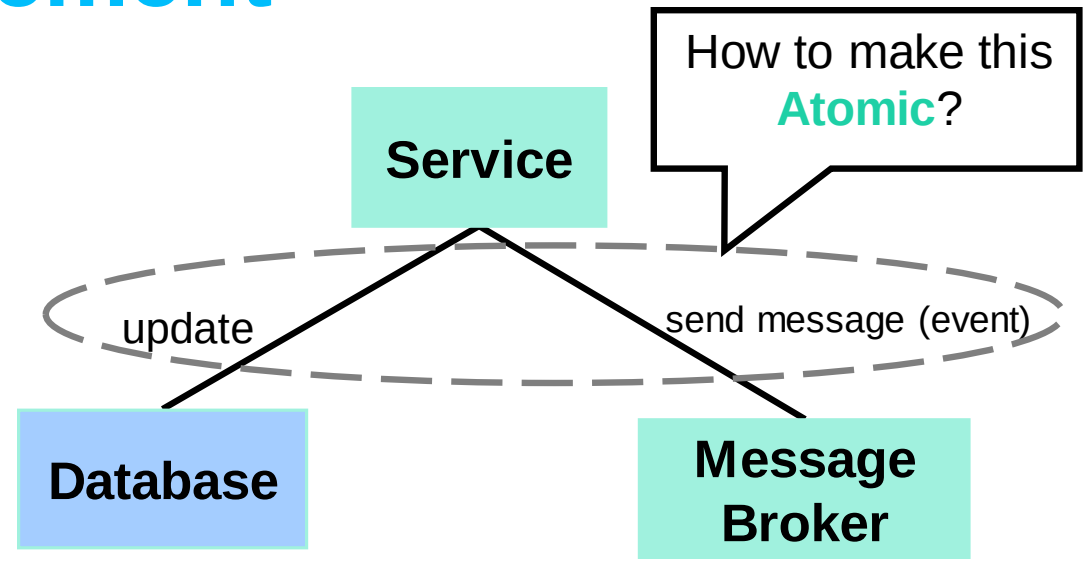


Asynchronous Messaging

C. Transactional messaging

Problem Statement

The **problem of atomically** **updating the database** and **publishing an event**.



- It is essential that these **two operations** are done **atomically**.
- If the service crashes **after updating** the database but **before publishing** the event,
  - ⊖ The system becomes **inconsistent**.

Asynchronous Messaging

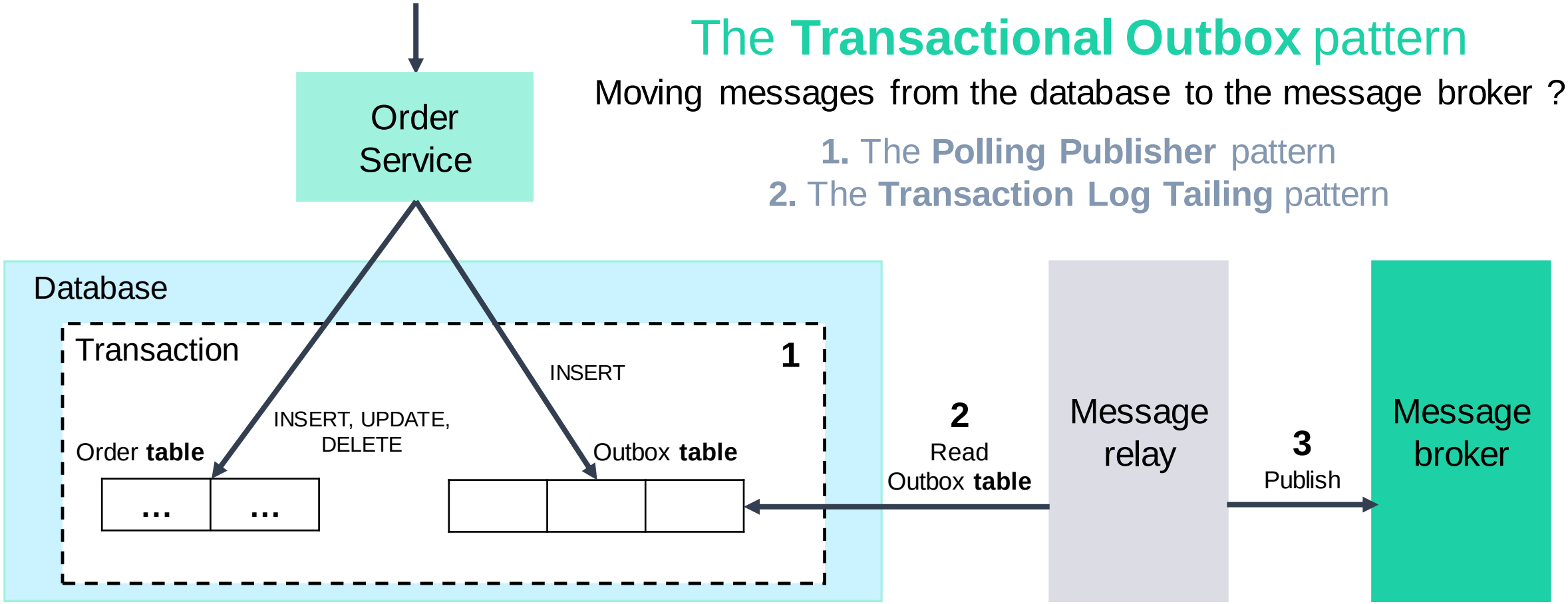
C. Transactional messaging

Solution: Using a database table as a message queue

The **Transactional Outbox** pattern

Moving messages from the database to the message broker ?

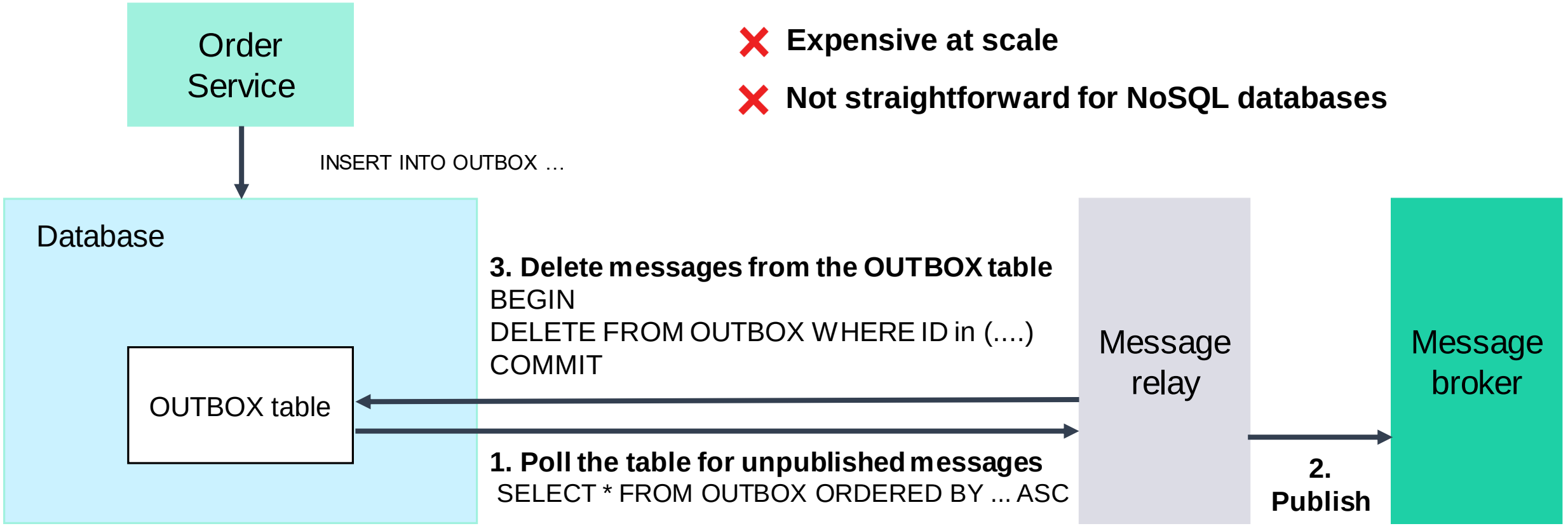
1. The **Polling Publisher** pattern
2. The **Transaction Log Tailing** pattern



Asynchronous Messaging

C. Transactional messaging

Publishing events by applying the **Polling Publisher pattern**

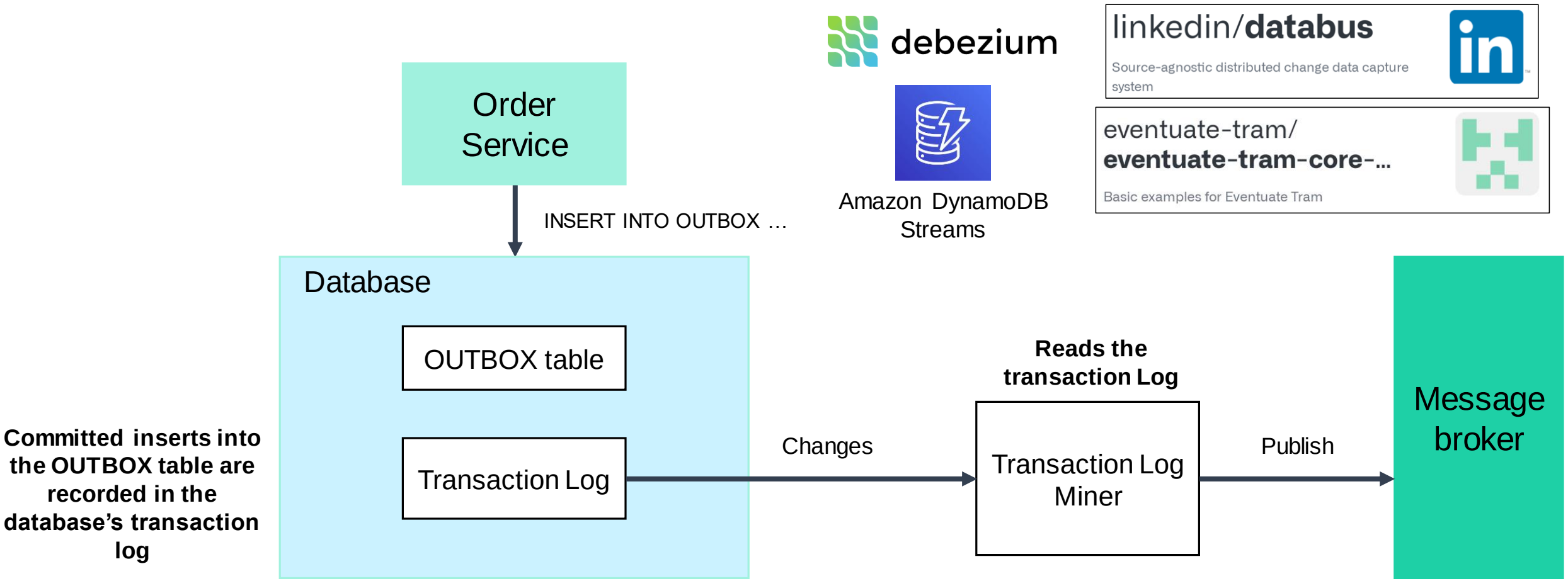


- ✗ **Expensive at scale**
- ✗ **Not straightforward for NoSQL databases**

Asynchronous Messaging

C. Transactional messaging

Publishing events by applying the **Transaction Log Tailing pattern**



Asynchronous Messaging

Key Takeaways

- **Asynchronous messaging** is an **IPC mechanism** enabling communication between a **sender** and a **receiver**. **Messages** and **channels** are its main two concepts.
- A **channel** can be a **Point-to-Point** (for one-to-one interaction styles) or a **Publish-Subscribe** (for one-to-many interaction styles).
- It is possible to **implement all the interaction styles** using asynchronous messaging.
- There is **no standard** for message-based **API specification**.
- To implement asynchronous messaging in practice, it is recommended to rely on a **message broker** (even if it is also possible to do it using a brokerless architecture).
- When going asynchronous, you have to deal with a couple of **design issues** including:
  - Preserving **message ordering** when scaling out message receivers using **sharded channels**.
  - Dealing with **duplicate messages** by writing **idempotent message handlers** or by **tracking messages and discarding duplicates**.
  - **Atomically** updating the database and publishing an event using the **Transactional Outbox** pattern. This later, relies on one of the two patterns to publish events:
    - The **Polling Publisher** pattern
    - The **Transaction Log Tailing** pattern

Questions are welcome