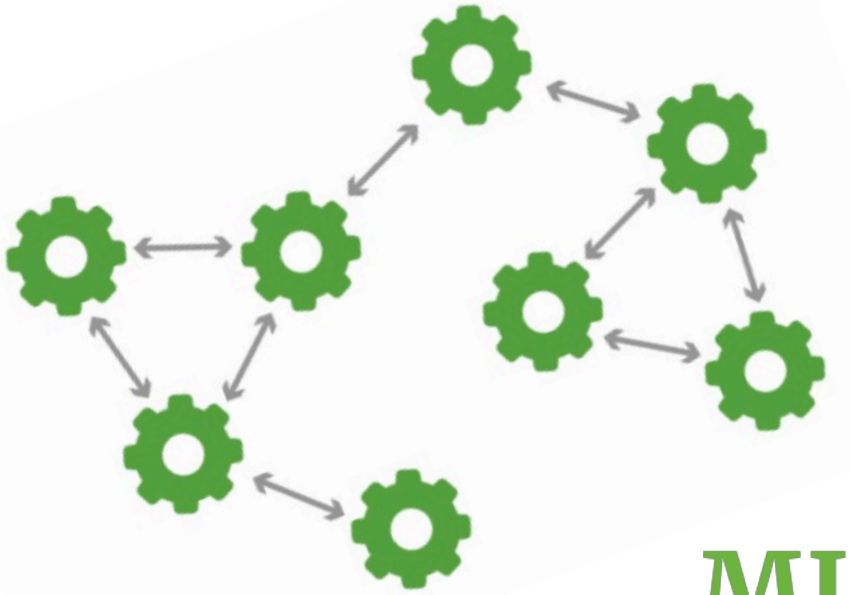
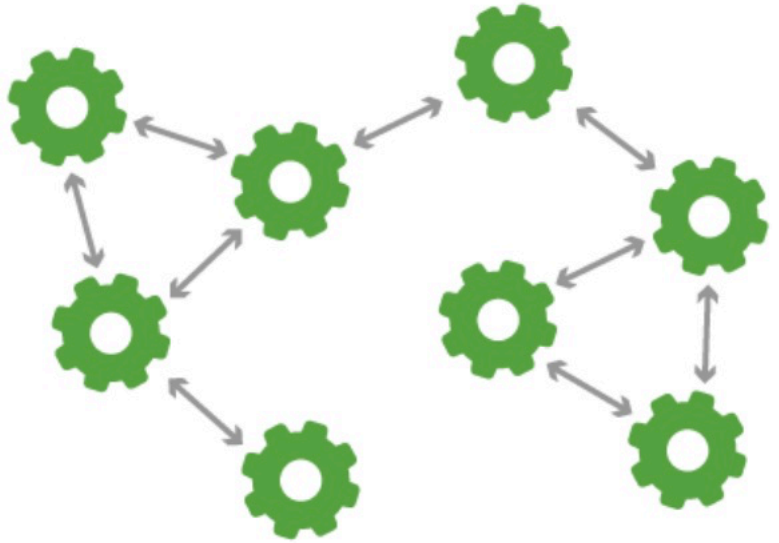


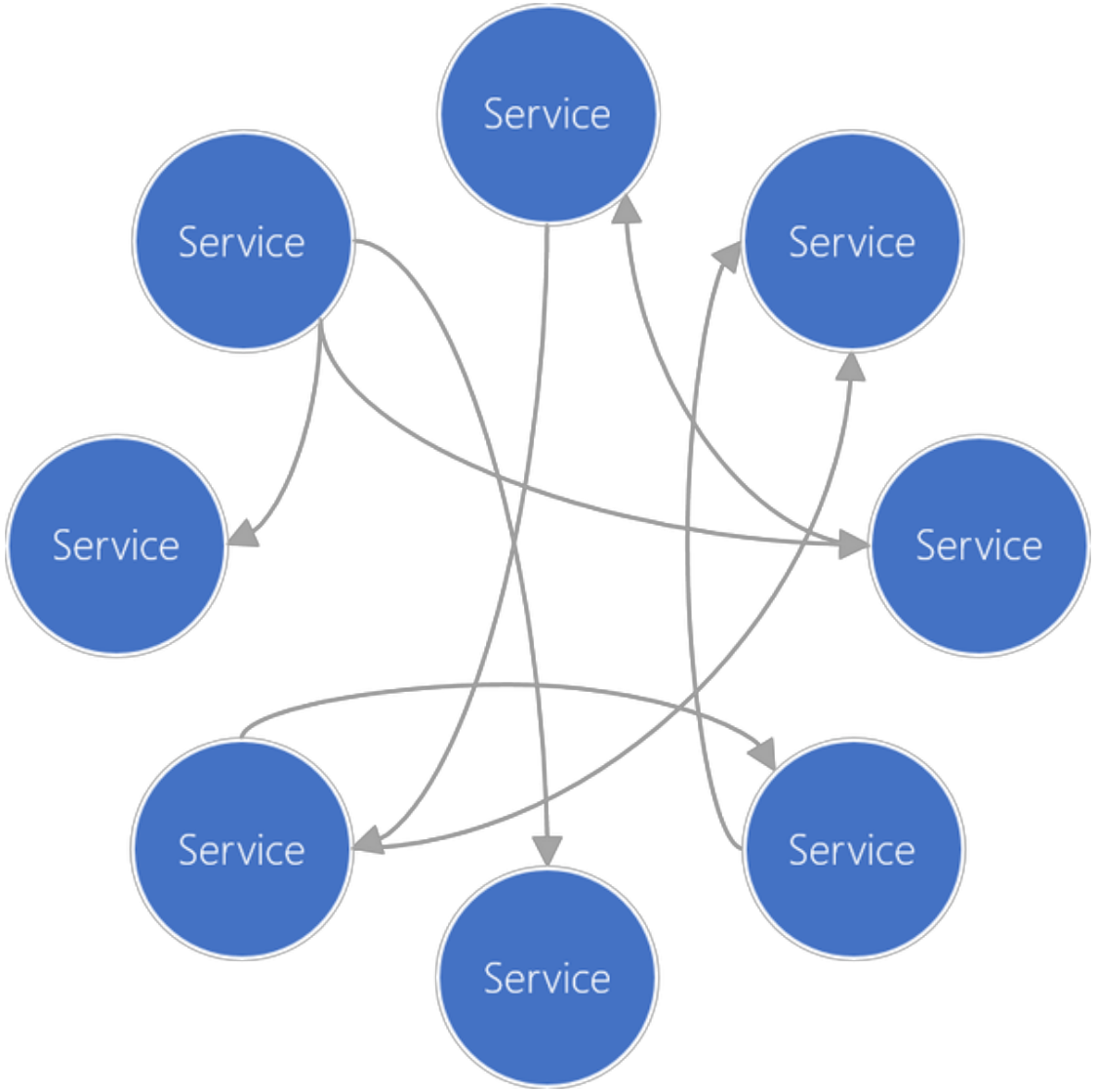


MICROSERVICES

비동기 트랜잭션

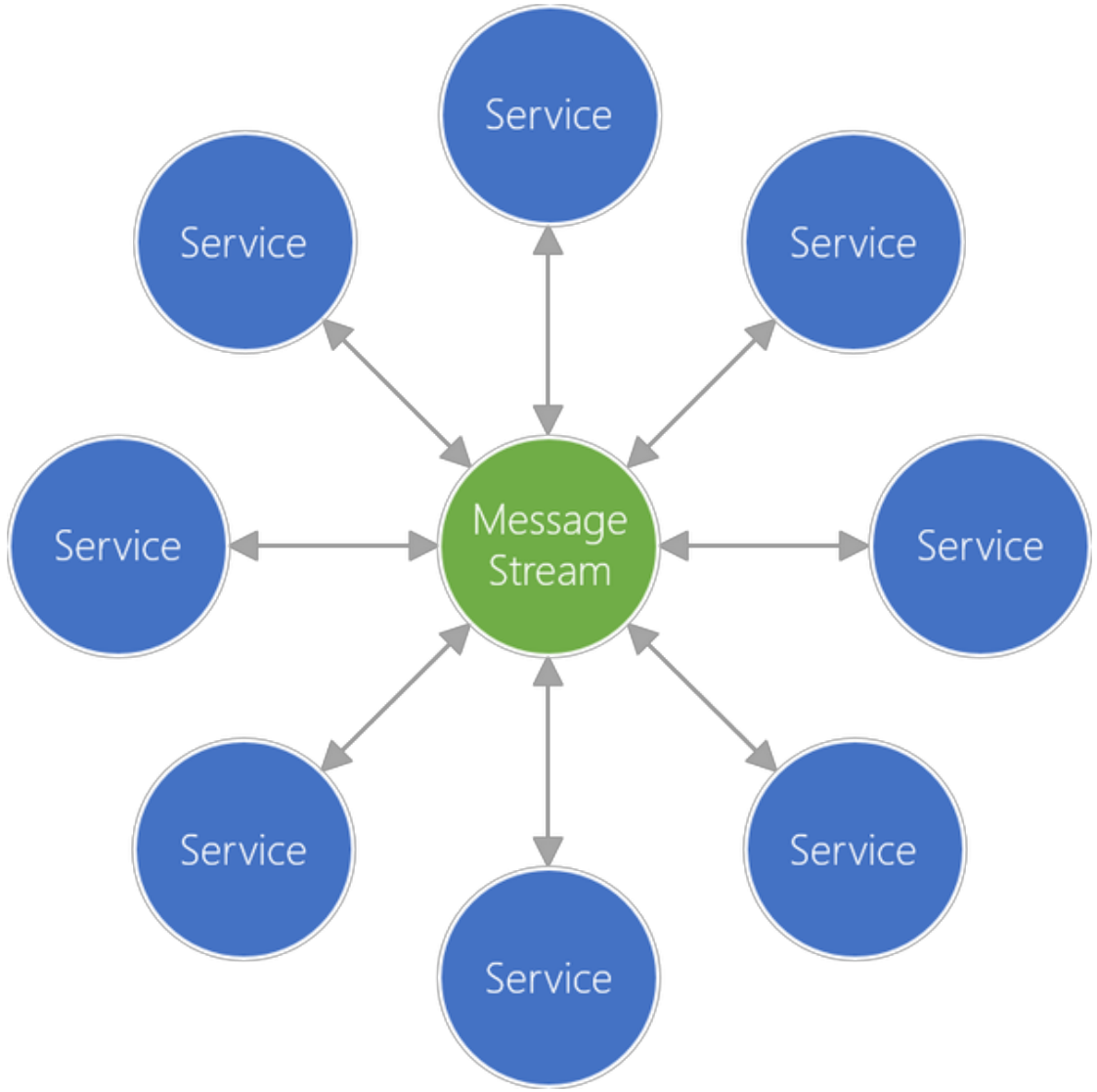
1. Web API 기반 메시징

- Request, Response
- 서비스 장애로 프로세스 실패 가능성 존재
- Retry policy 필요
- 서비스 간의 관계 복잡
- 결합도가 생길 수 있음



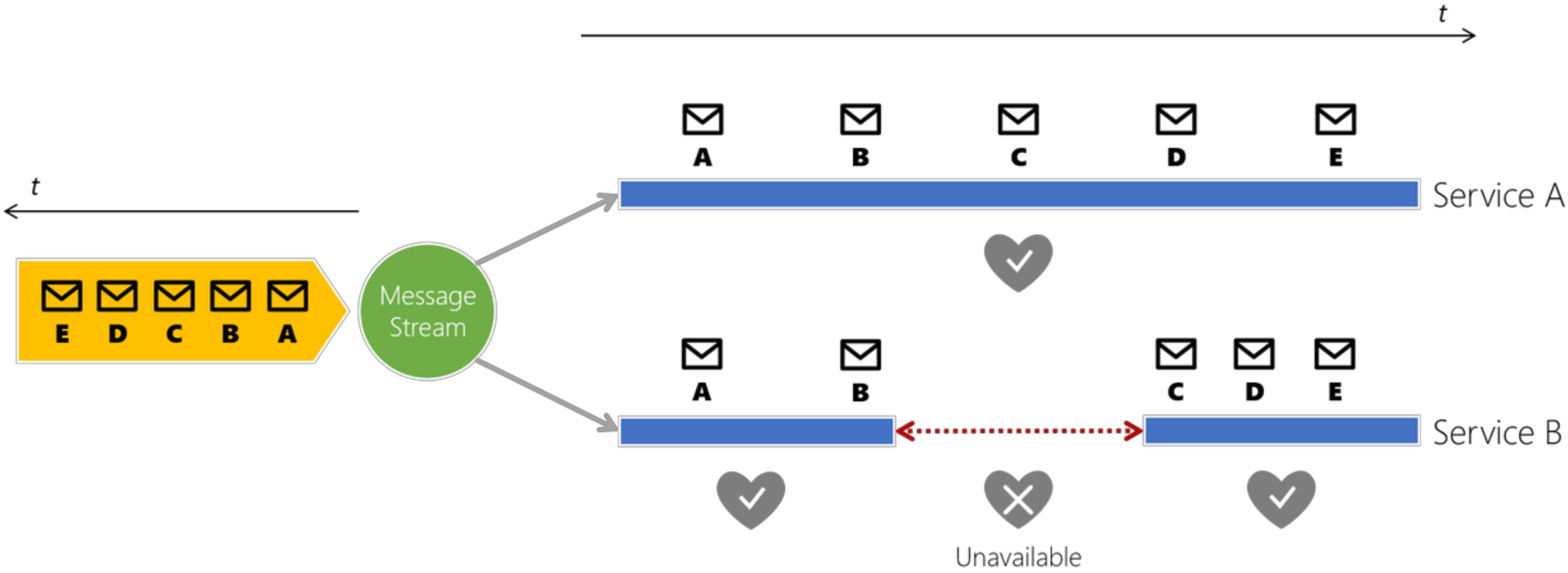
2. 메시지 스트림을 사용한 서비스 통합

- 중앙화 된 메시지 스트림 기반
- Publish/Subscribe
- 메시지 흐름 단순화
- 비동기 서비스이기 때문에 응답 지연 감소



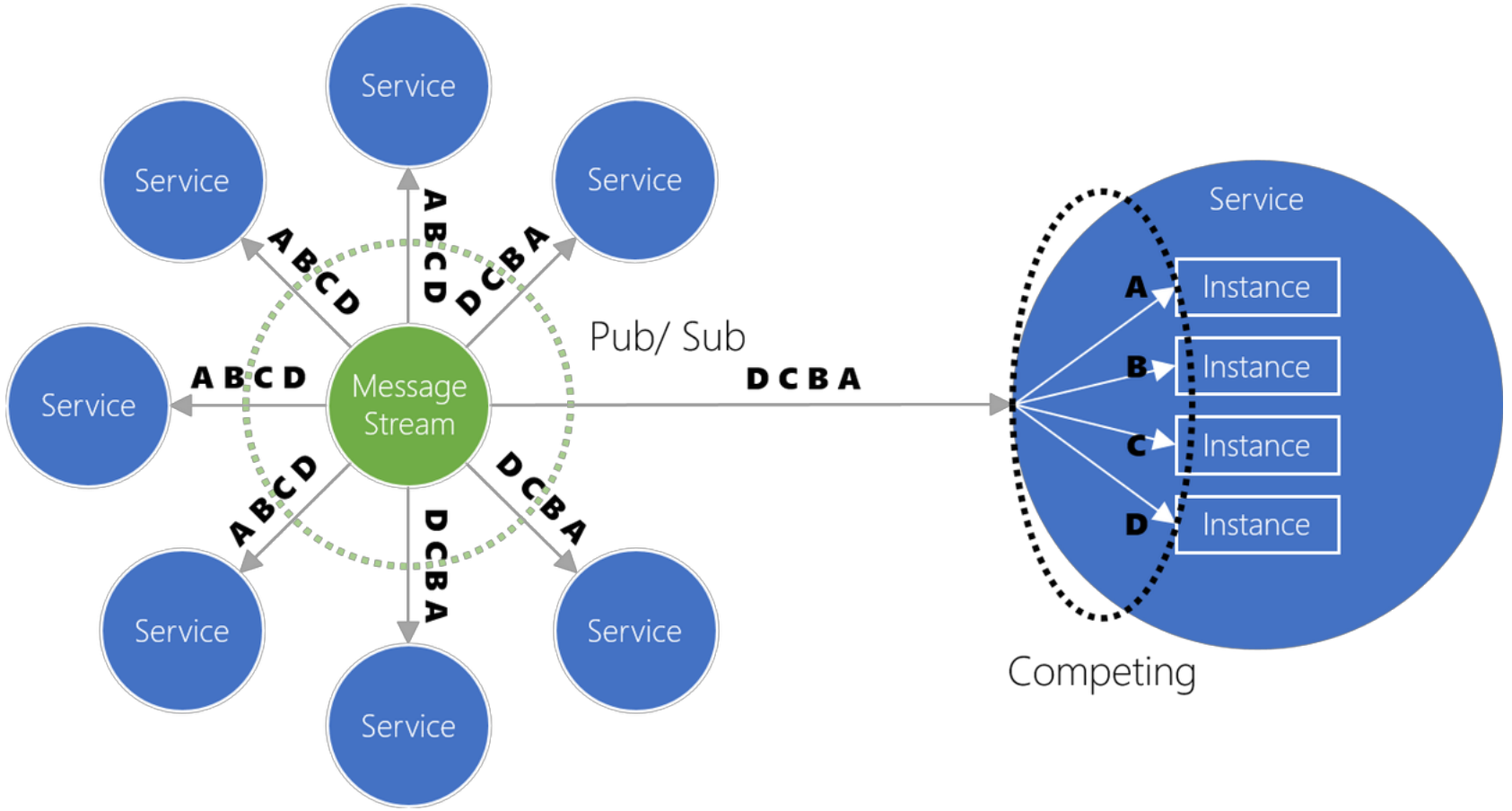
2. 메시지 스트림을 사용한 서비스 통합

- 일시적인 서비스 불능에 대해 메시지 전달 보장해야 함



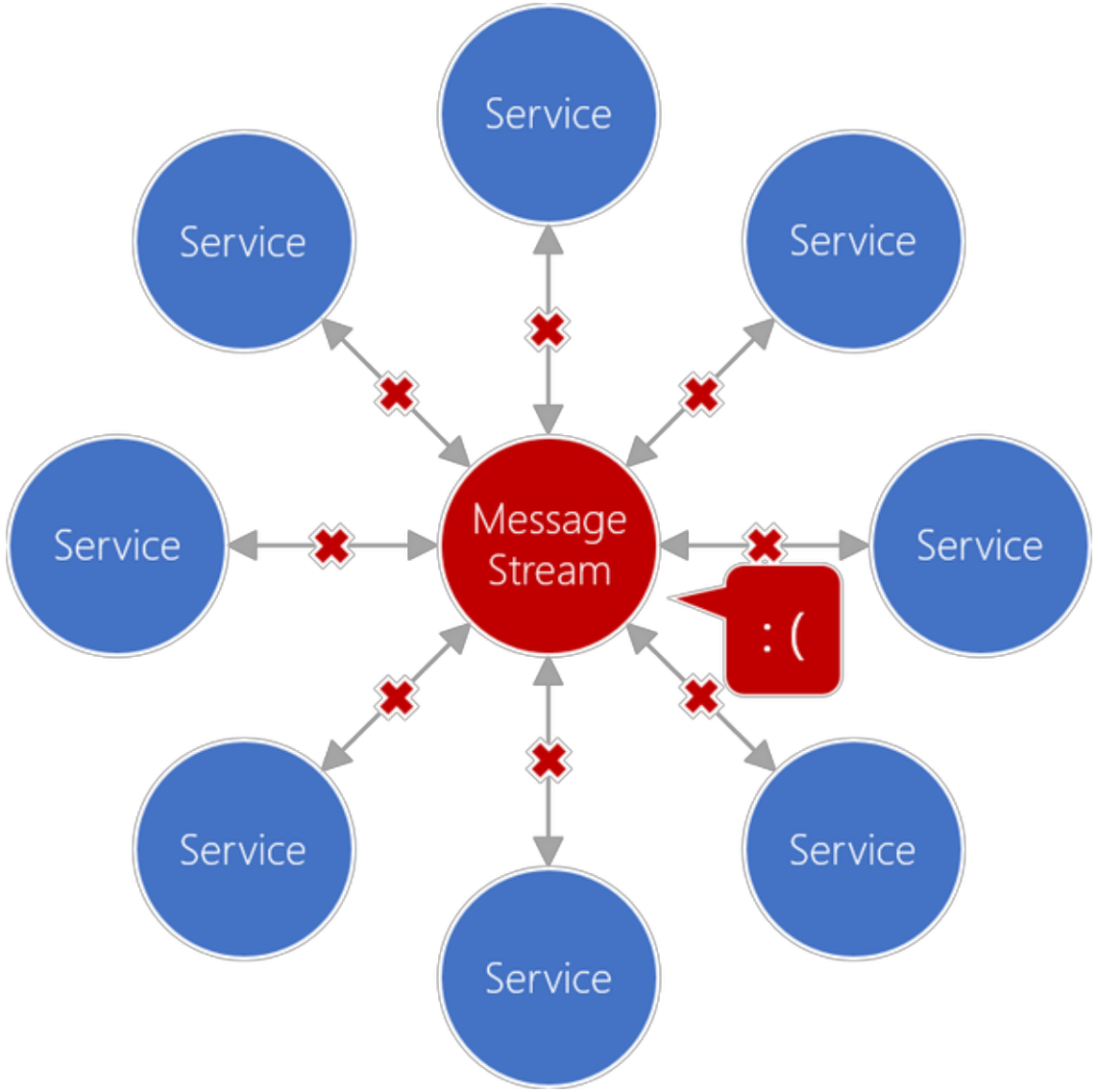
3. 서비스 수평 확장성

- 성능 요구사항을 만족하기 위한 방법
- 메시지 분할
- Idempotency
  - *f(x) = f(f(x))*

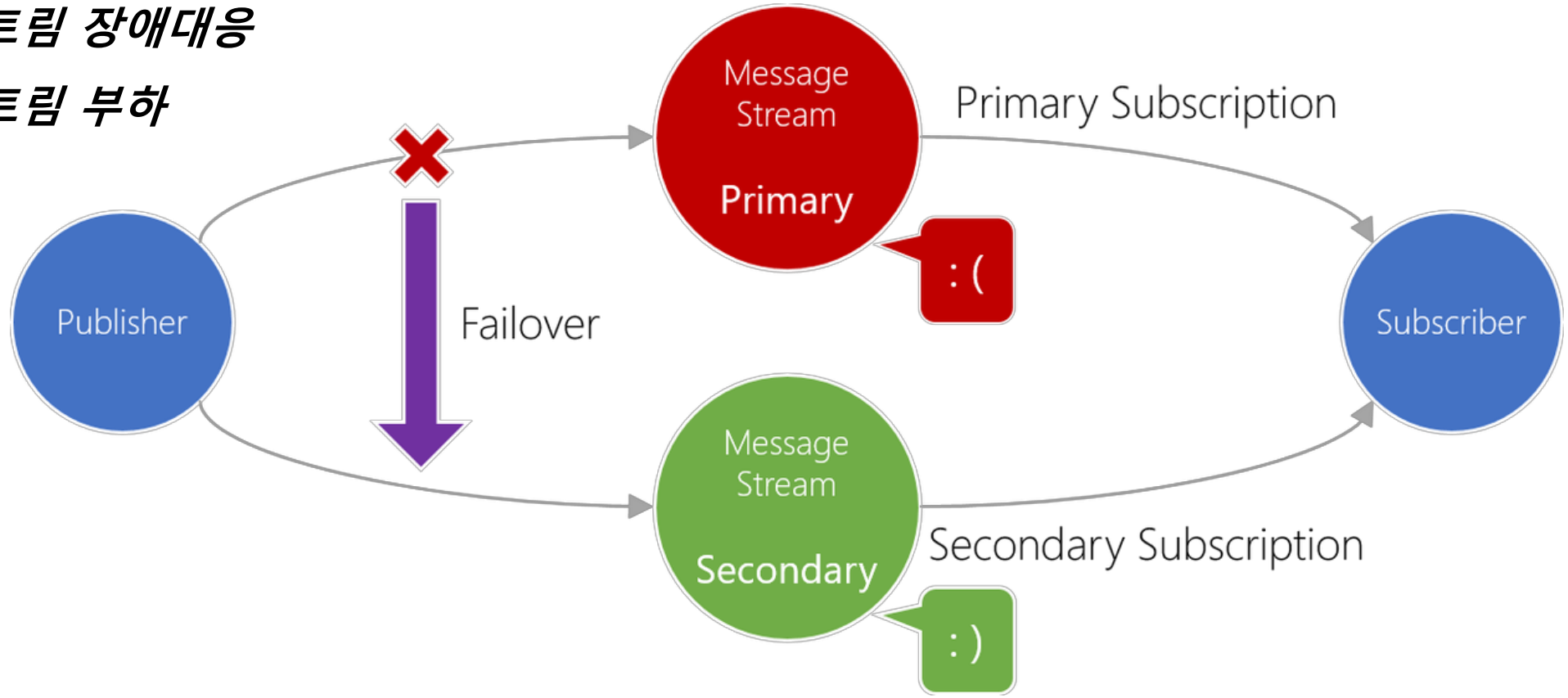


비동기 분산 아키텍처

- 중앙화 된 메시지 스트림의 단점
  - 높은 간접수준
  - 동기적 호출 및 조회
  - 메시지 스트림 장애대응
  - 메시지 스트림 부하



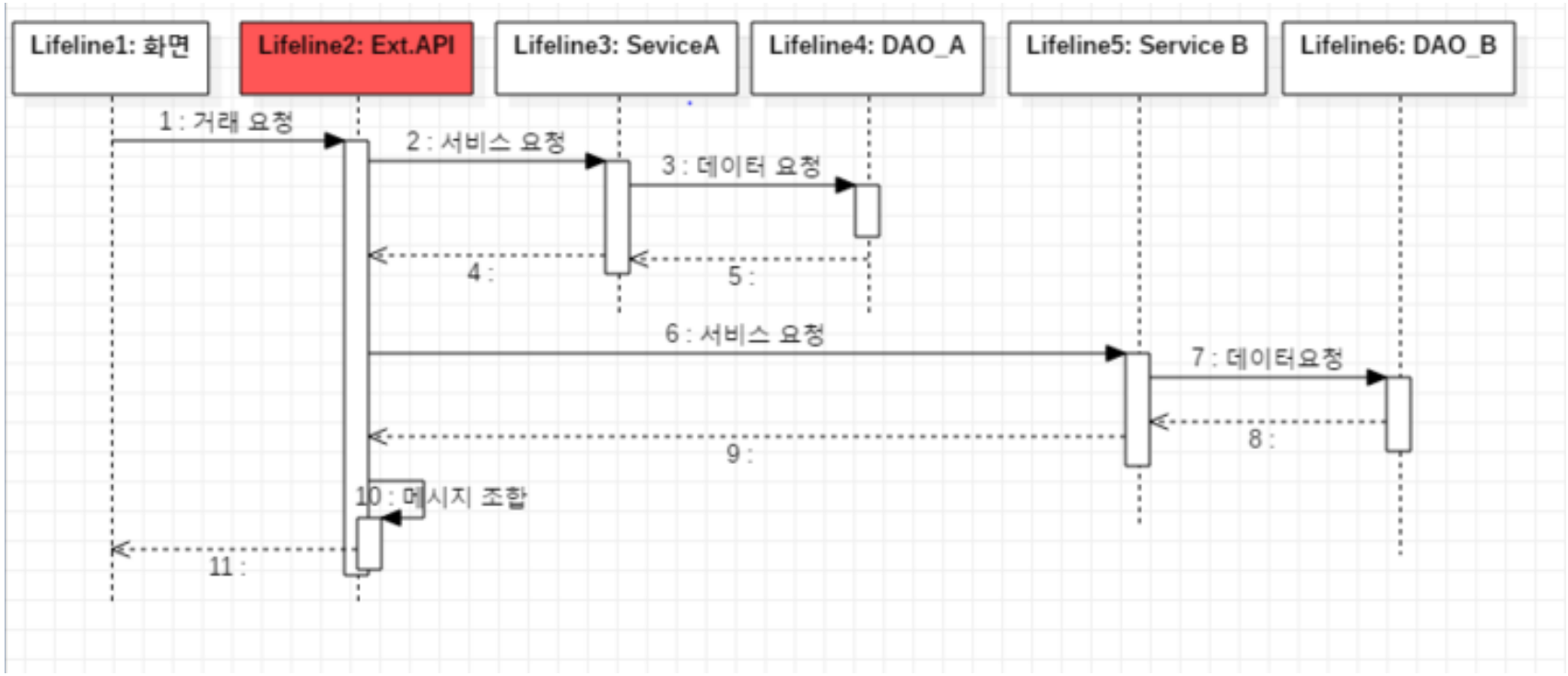
- 중앙화 된 메시지 스트림의 단점
  - 높은 간접수준
  - 동기적 호출 및 조회
  - 메시지 스트림 장애대응
  - 메시지 스트림 부하



1. 동기 거래 패턴

- 다른 서비스를 API Gateway or Service mesh or REST API를 통해 호출
- 분산 트랜잭션 고려

- API Gateway Orchestration

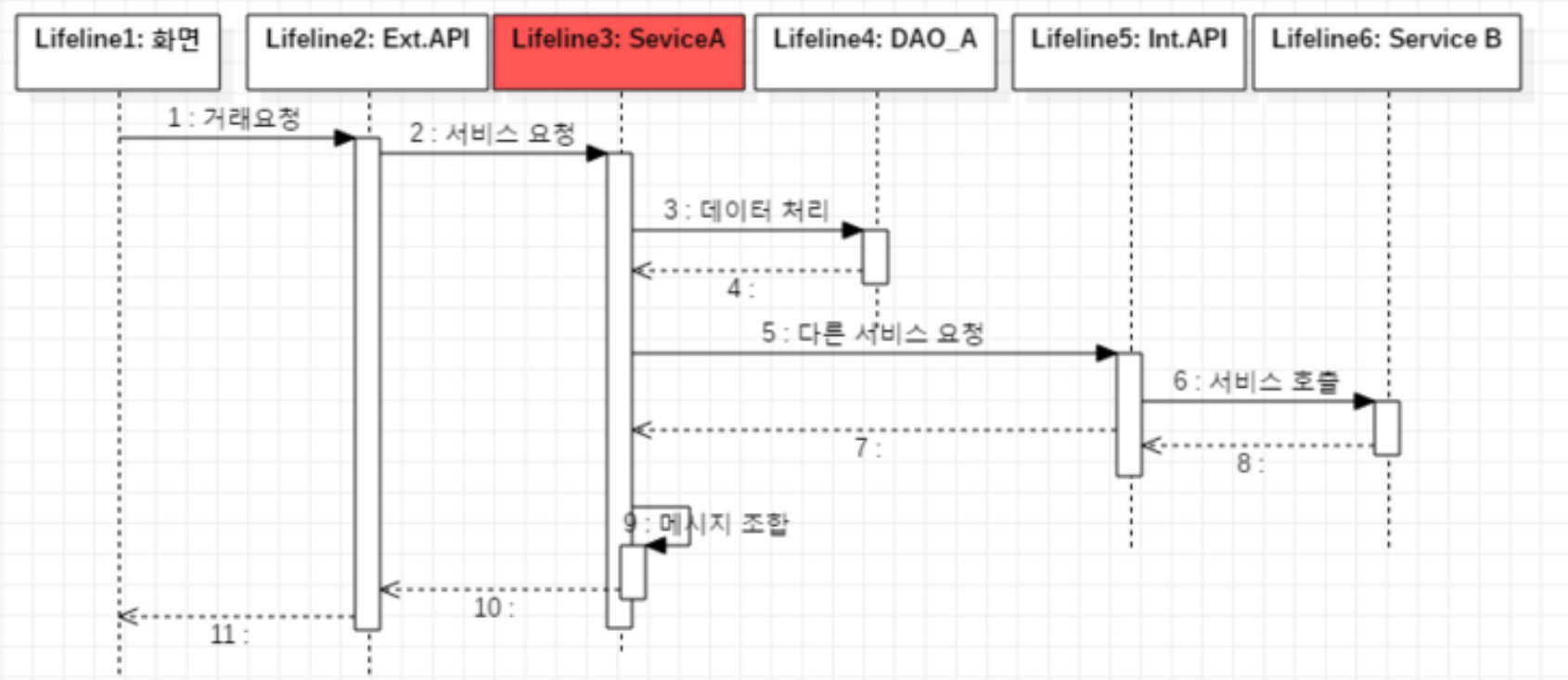


https://blog.naver.com/PostView.nhn?blogId=stmshra&logNo=221490750735&redirect=Dlog&widgetTypeCall=true&directAccess=false

1. 동기 거래 패턴

- 다른 서비스를 API Gateway or Service mesh or REST API를 통해 호출
- 분산 트랜잭션 고려

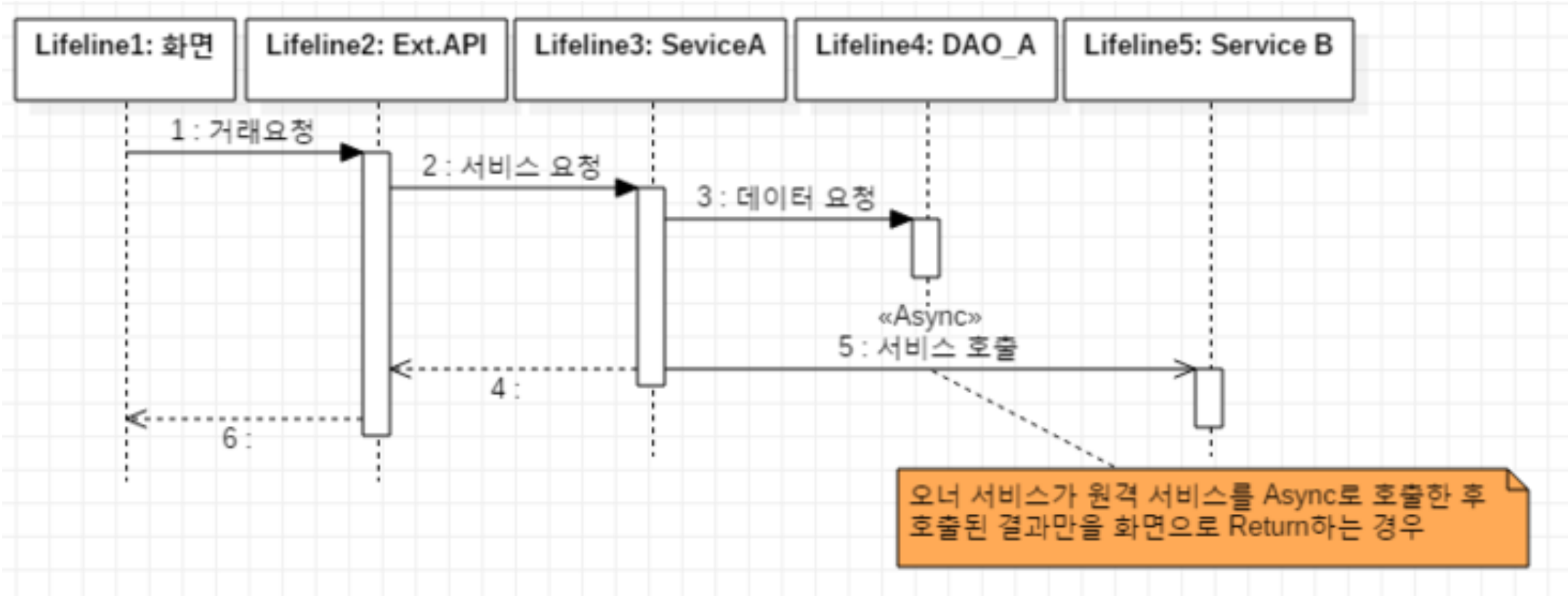
- 내부 서비스 Orchestration



https://blog.naver.com/PostView.nhn?blogId=stmshra&logNo=221490750735&redirect=Dlog&widgetTypeCall=true&directAccess=false

2. 비동기 거래 패턴

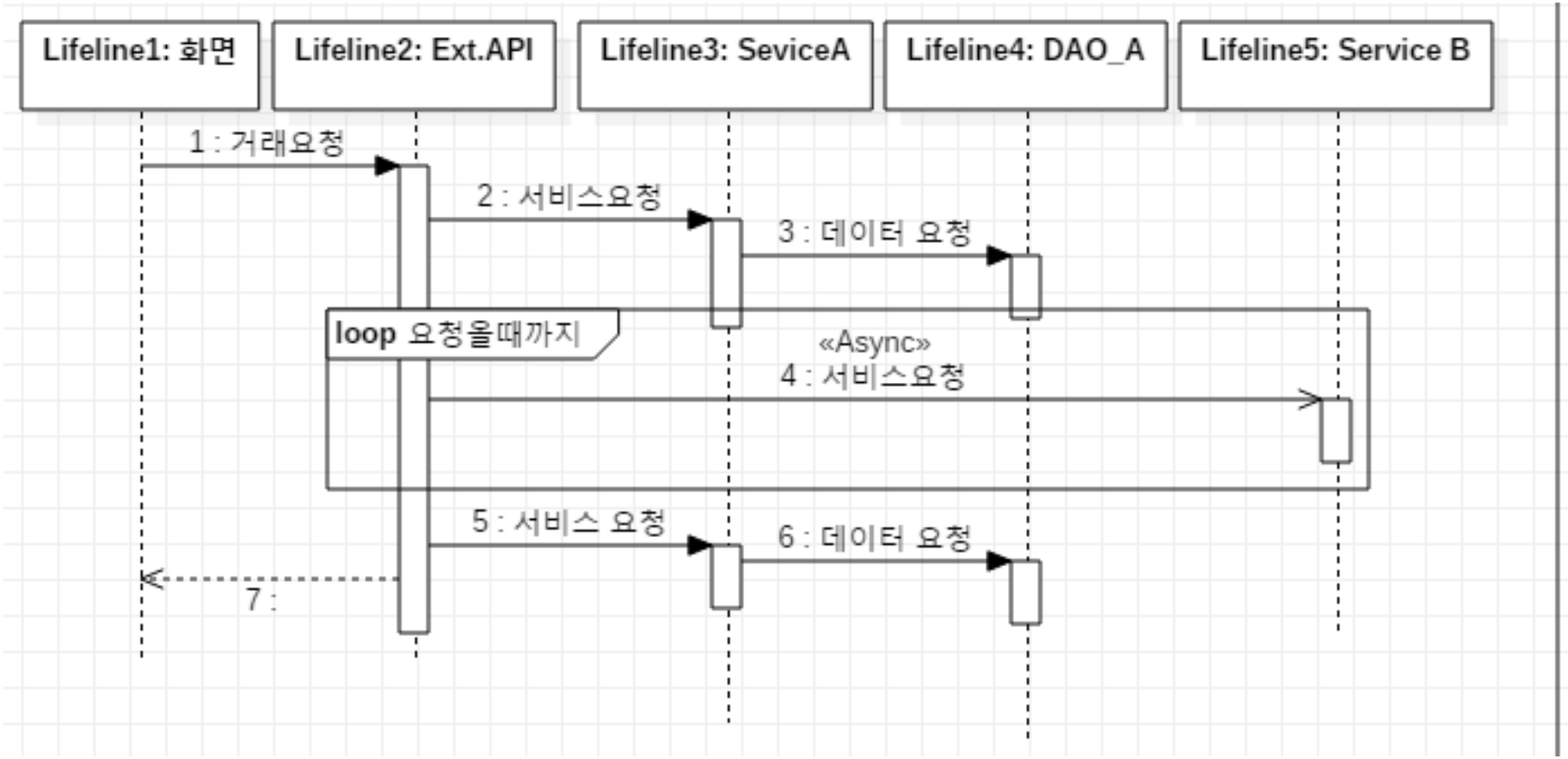
- 비동기 호출 성공 여부 판단 후 결과 Return



https://blog.naver.com/PostView.nhn?blogId=stmshra&logNo=221490750735&redirect=Dlog&widgetTypeCall=true&directAccess=false

2. 비동기 거래 패턴

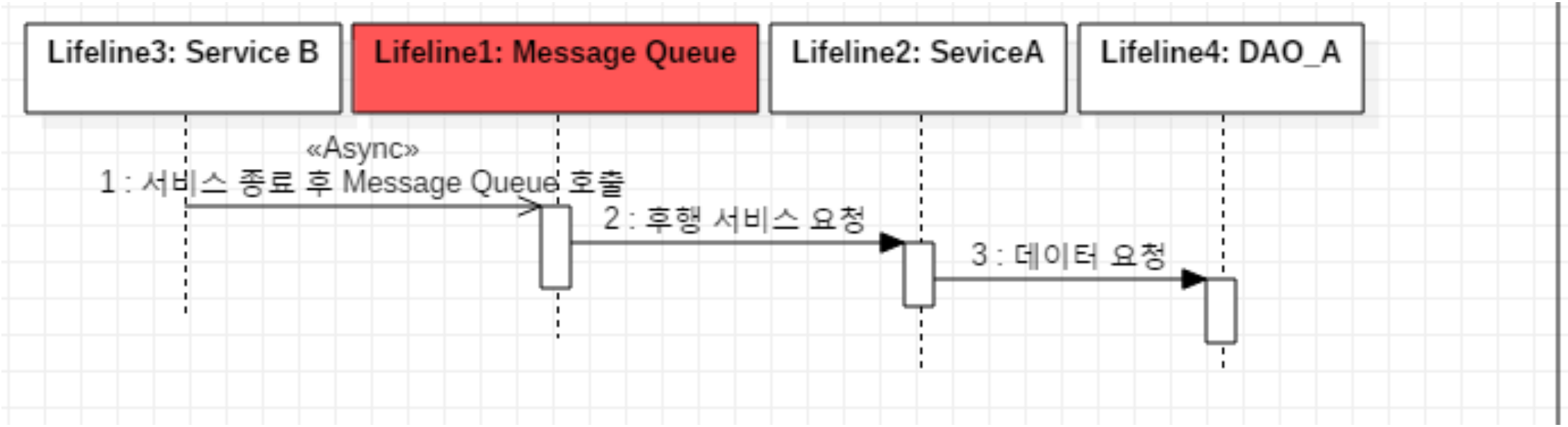
- 비동기로 호출 된 서비스가 완료될 때까지 Wait



https://blog.naver.com/PostView.nhn?blogId=stmshra&logNo=221490750735&redirect=Dlog&widgetTypeCall=true&directAccess=false

2. 비동기 거래 패턴

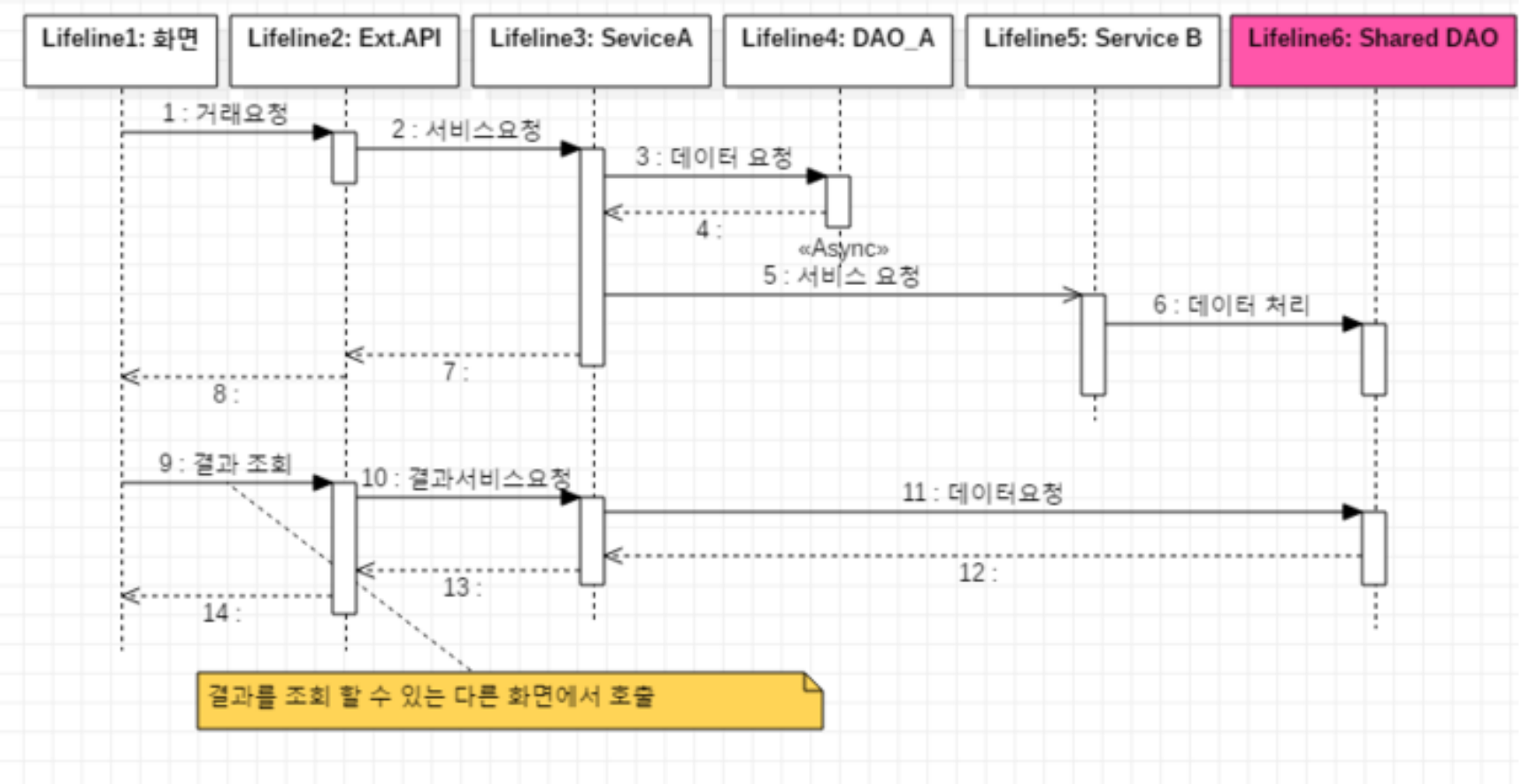
- 비동기로 호출 된 서비스 완료 후 다시 비동기 본 서비스에 사후 처리 요청



https://blog.naver.com/PostView.nhn?blogId=stmshra&logNo=221490750735&redirect=Dlog&widgetTypeCall=true&directAccess=false

2. 비동기 거래 패턴

- SharedDB로 결과를 Update한 후 해당 결과를 기반으로 데이터 처리



https://blog.naver.com/PostView.nhn?blogId=stmshra&logNo=221490750735&redirect=Dlog&widgetTypeCall=true&directAccess=false

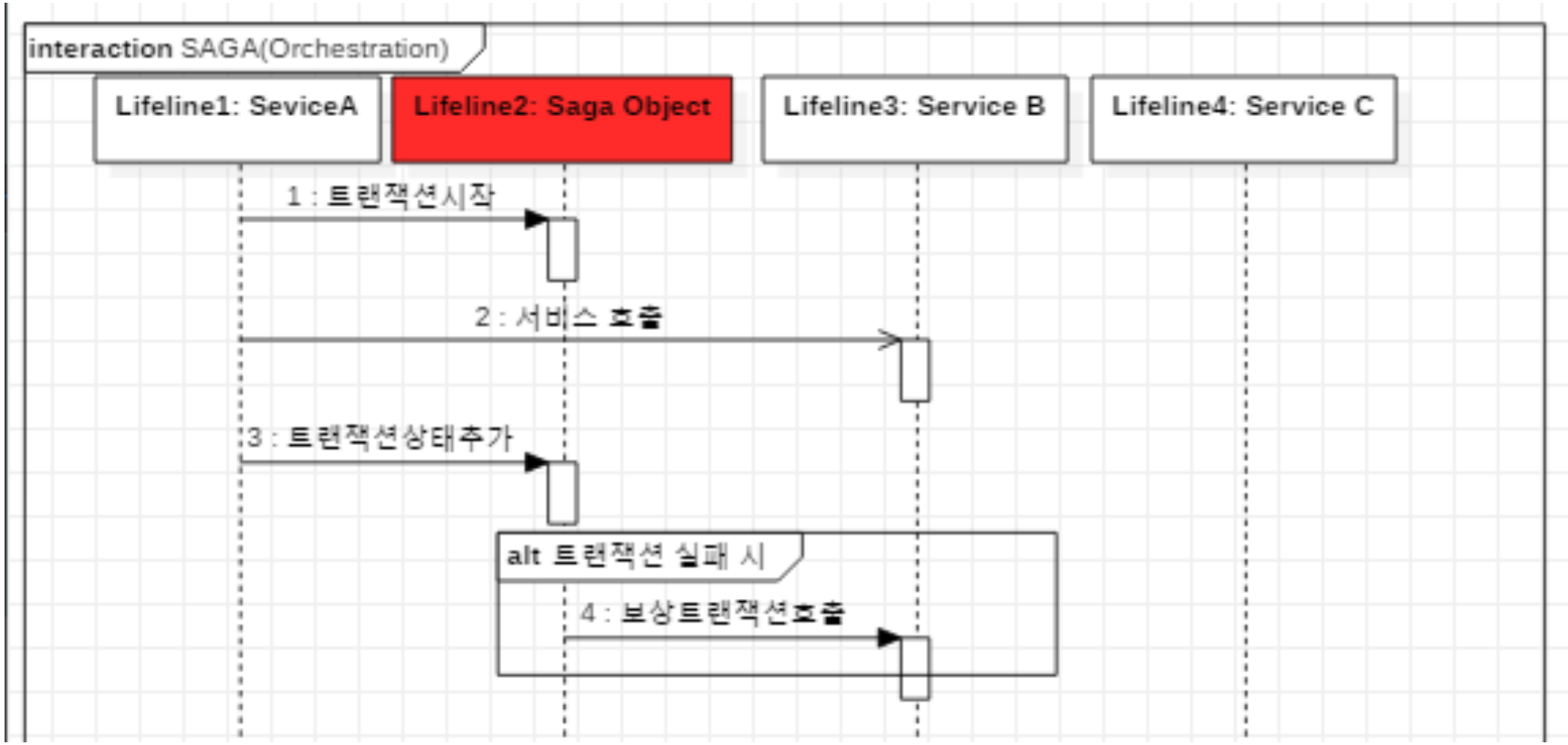
3. 트랜잭션 처리

- *2 Phase Commit*
  - 장애에 취약, 비추천
- *취소 프로세스에 의한 처리*
  - Error 발생 후 취소 요청
- *Retry*
  - Retry 회수 지정
- *보상 트랜잭션*
  - *MSA 시스템에 적용*

3. 트랜잭션 처리

- 보상 트랜잭션

  - Saga Pattern(Orchestration 동기/비동기)

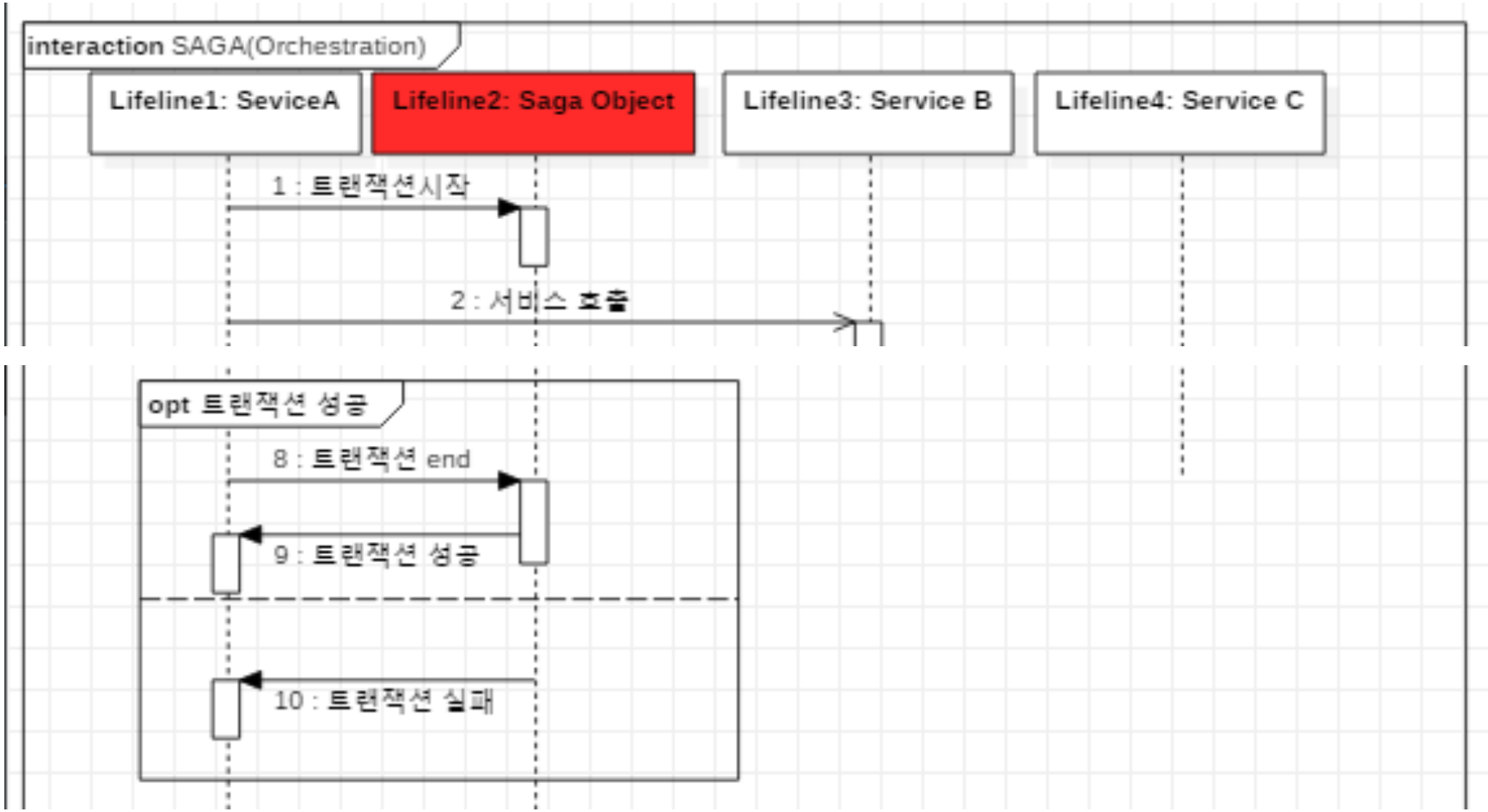


https://blog.naver.com/PostView.nhn?blogId=stmshra&logNo=221490750735&redirect=Dlog&widgetTypeCall=true&directAccess=false

3. 트랜잭션 처리

- 보상 트랜잭션

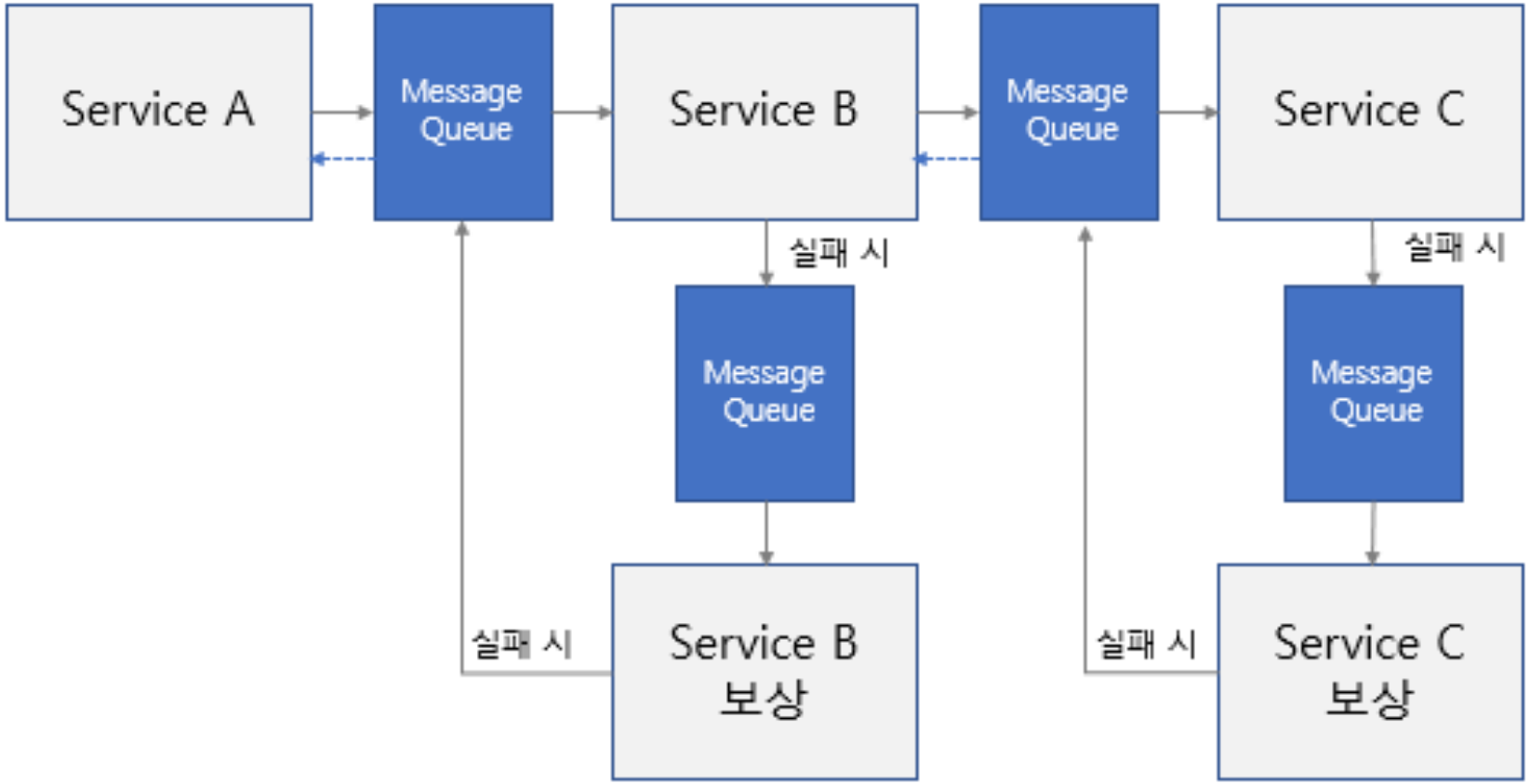
- Saga Pattern(Orchestration 동기/비동기)



https://blog.naver.com/PostView.nhn?blogId=stmshra&logNo=221490750735&redirect=Dlog&widgetTypeCall=true&directAccess=false

3. 트랜잭션 처리

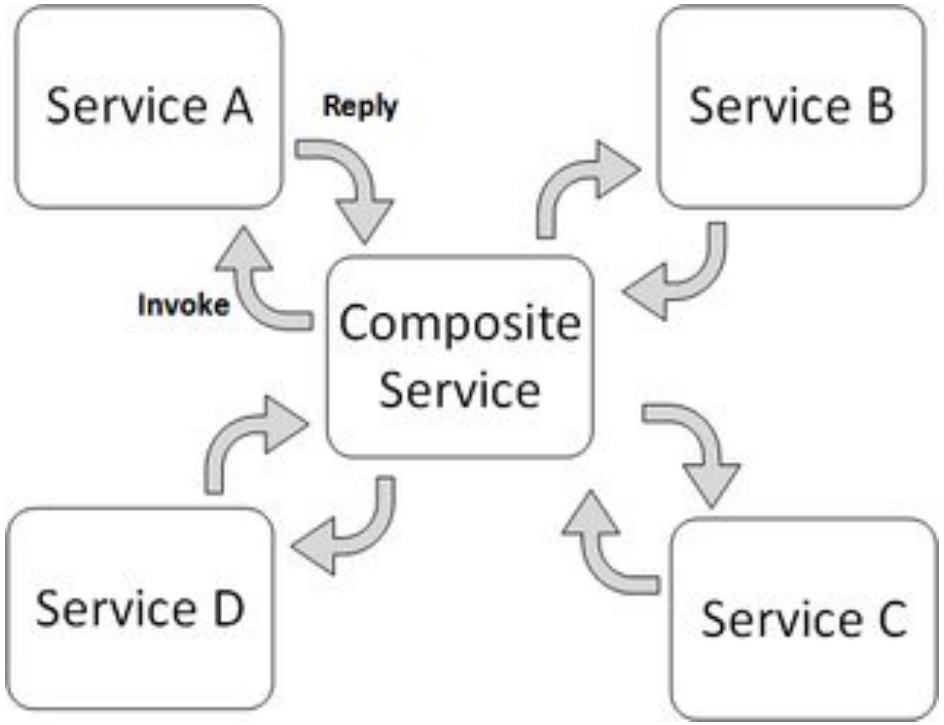
- 보상 트랜잭션
  - Saga Pattern(Choreography)

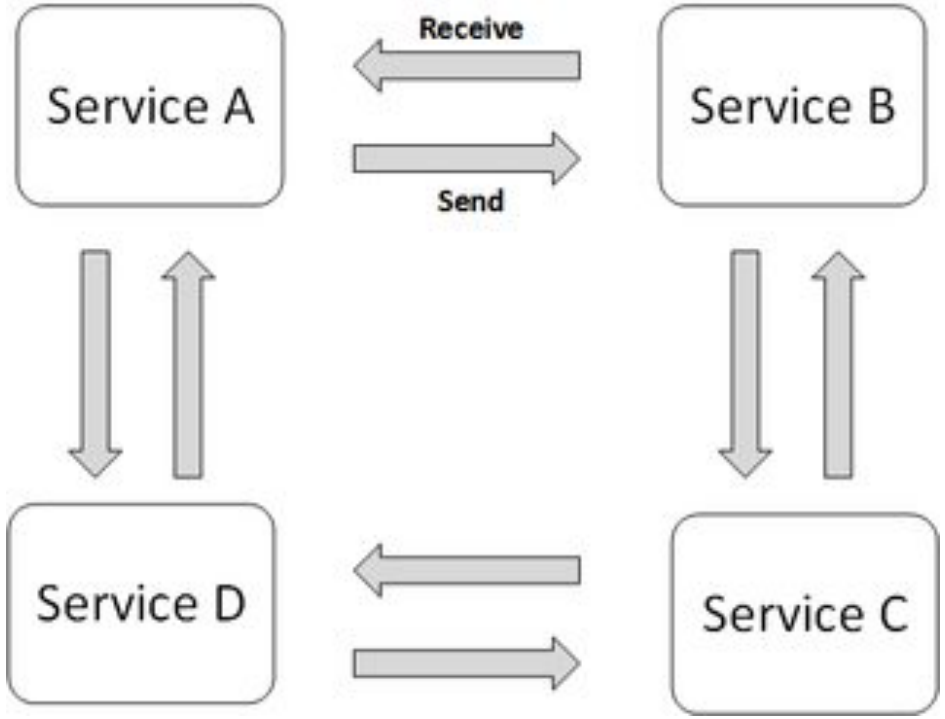


https://blog.naver.com/PostView.nhn?blogId=stmshra&logNo=221490750735&redirect=Dlog&widgetTypeCall=true&directAccess=false

3. 트랜잭션 처리

- Saga Pattern(Choreography) vs Saga Pattern(Orchestration)

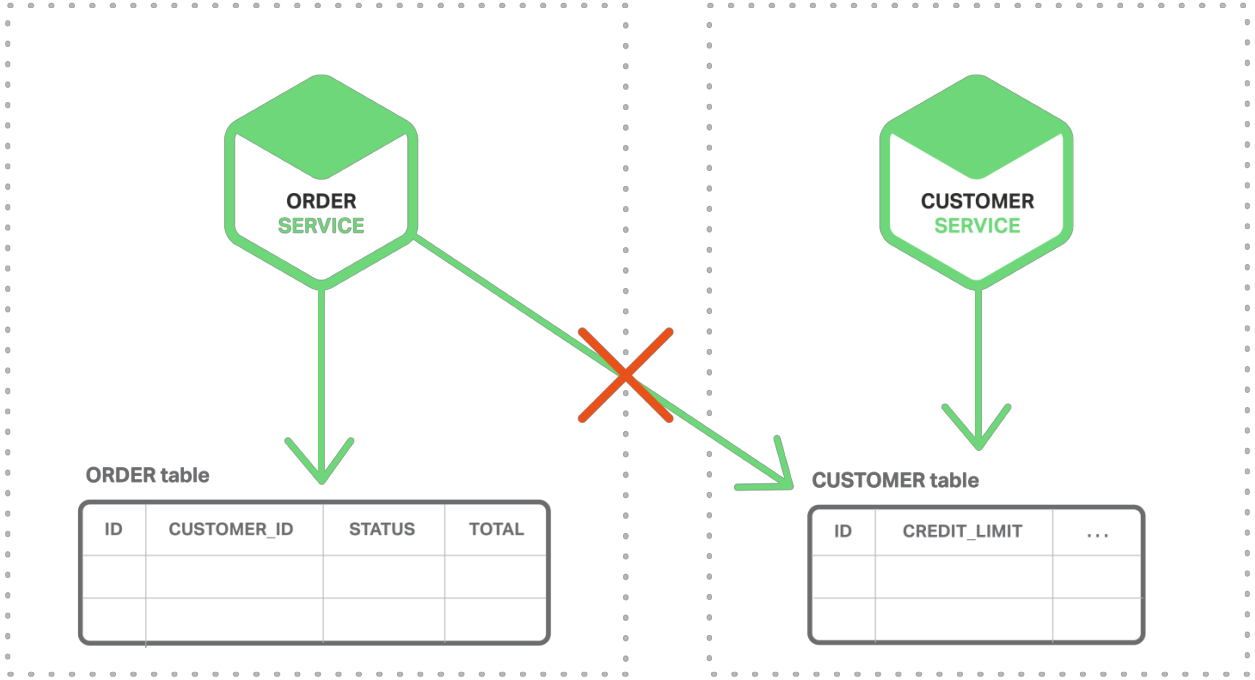




Event Driven Architecture

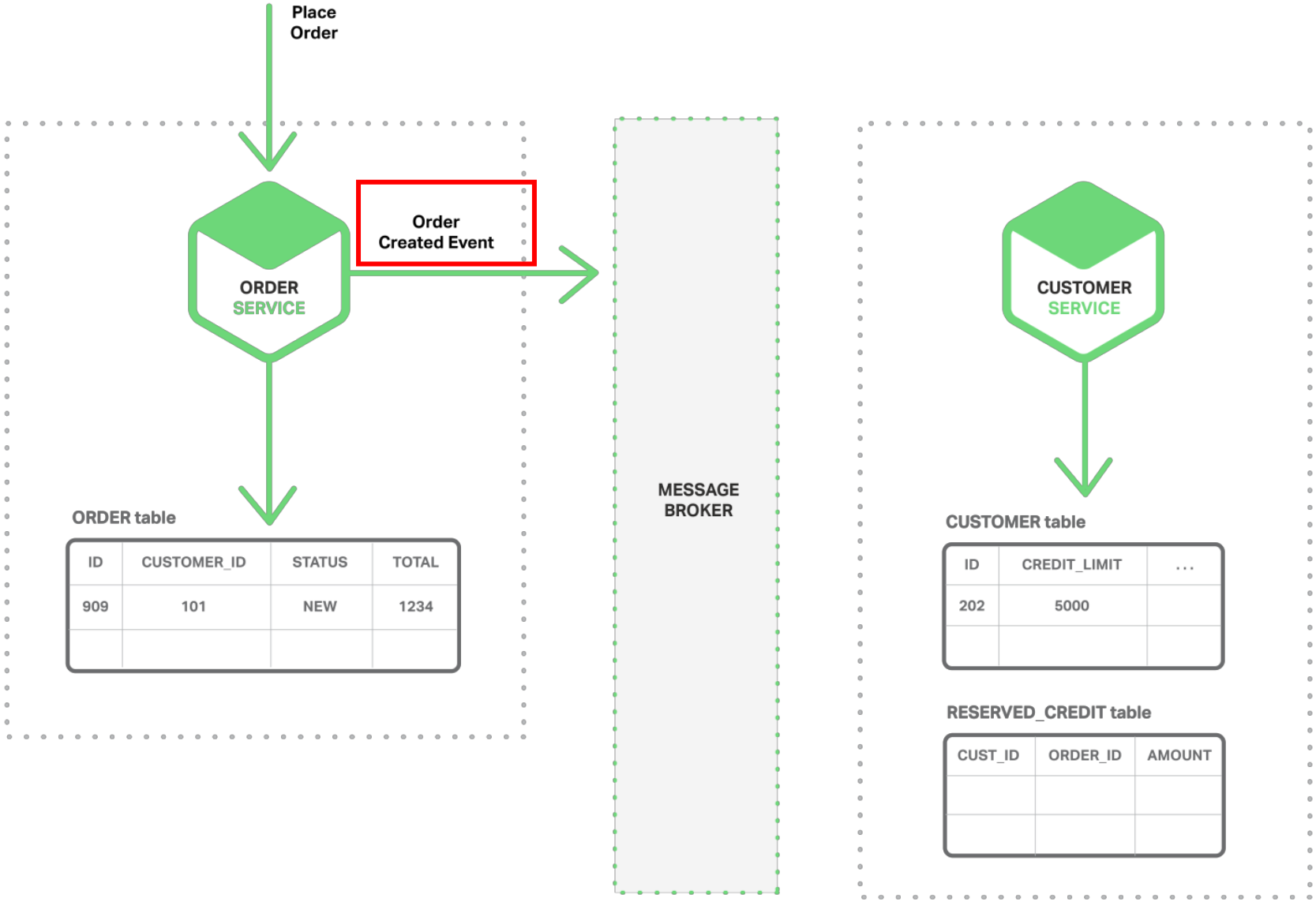
- Monolithic 애플리케이션
  - 하나의 RDB
  - Transaction ← ACID
    - Atomicity
    - Consistency
    - Isolation
    - Durable

- Microservice 애플리케이션
  - 각 서비스마다 독립적인 DB
  - 다른 서비스의 DB에 접근할 수 없음
  - 오로지 API를 통해서만 접근

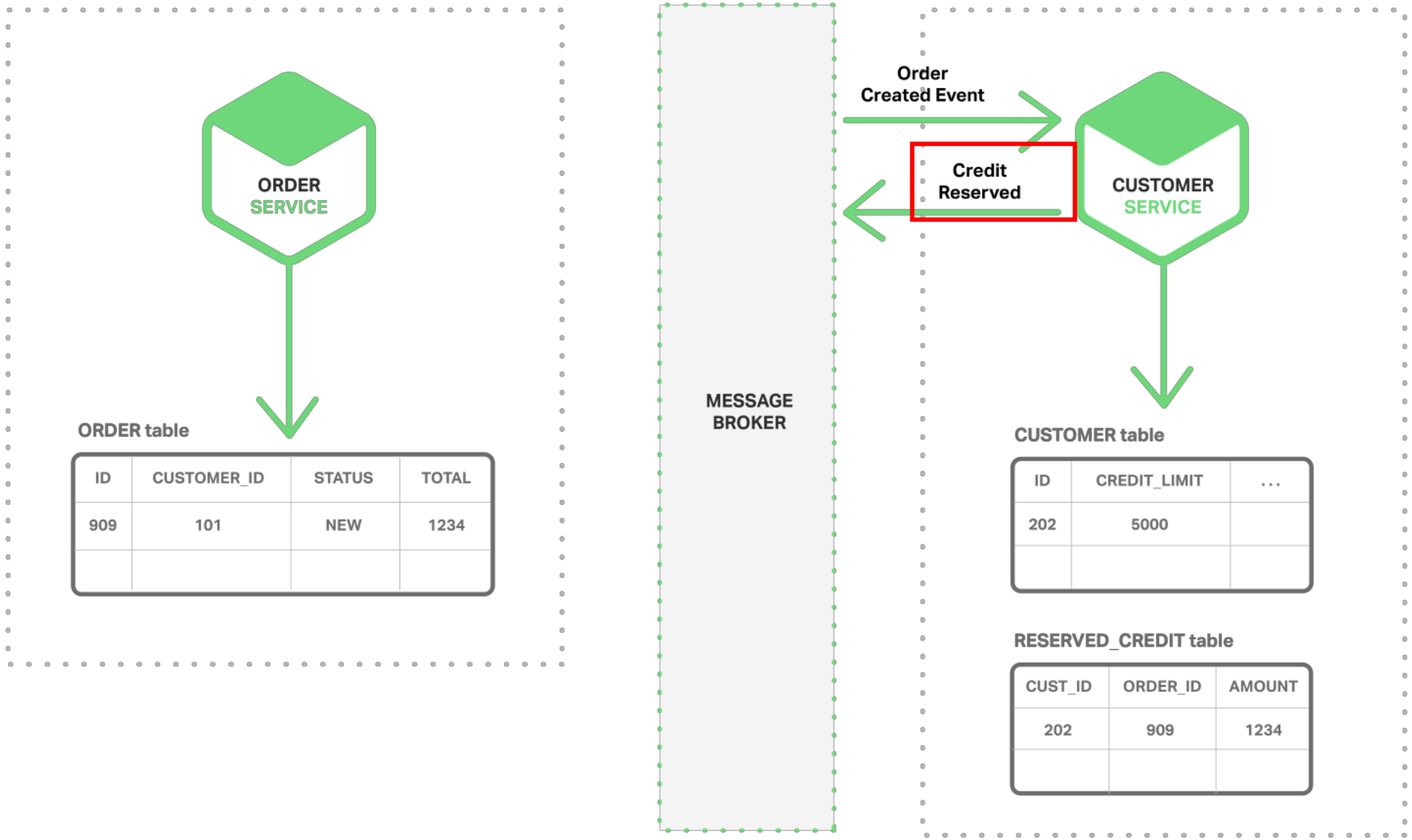


Event Driven Architecture

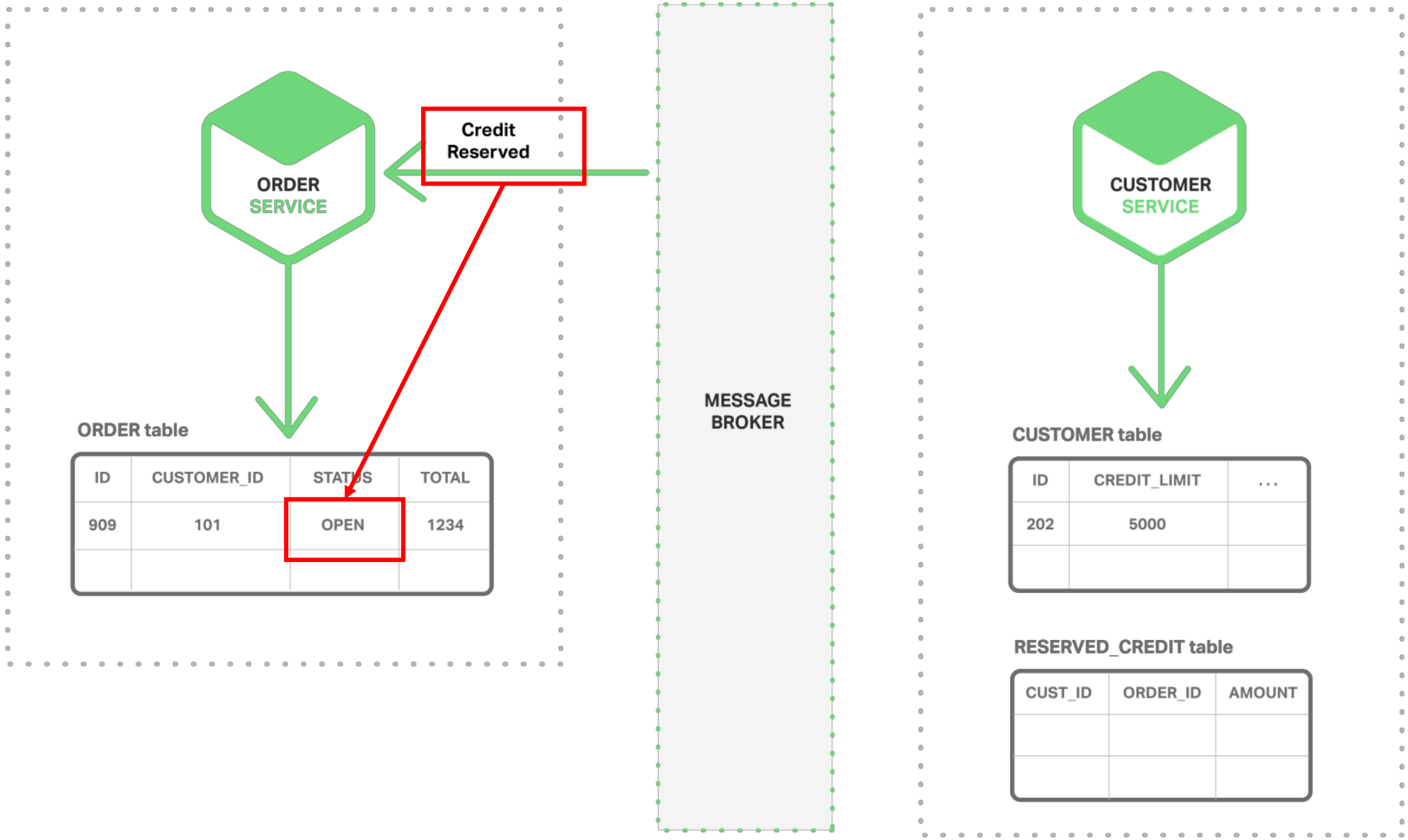
- 일관성



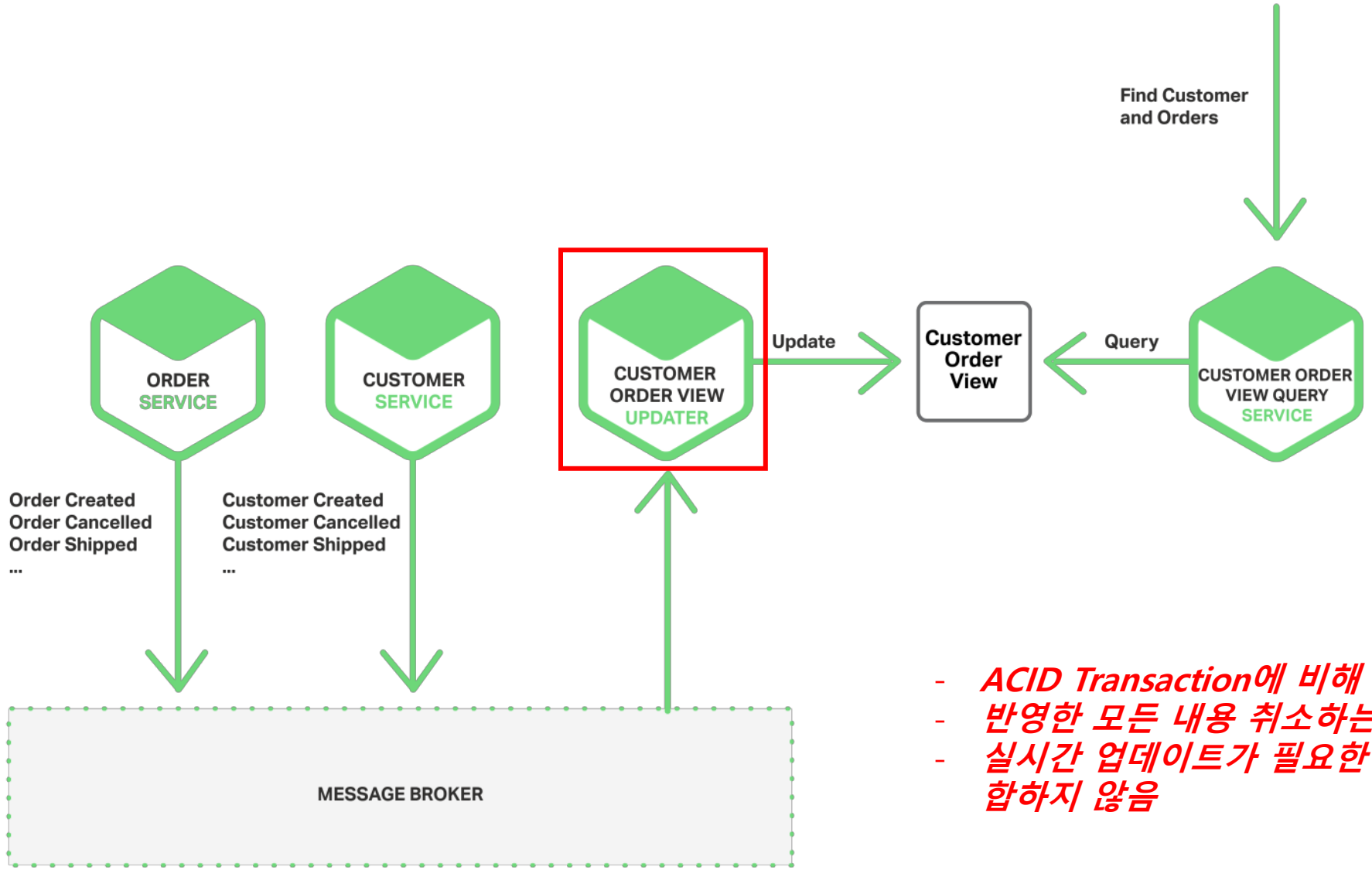
- 일관성


ORDER
SERVICE
ORDER table
ID
CUSTOMER_ID
STATUS
TOTAL
909
101
NEW
1234
MESSAGE
BROKER
Order
Created Event
Credit
Reserved
CUSTOMER
SERVICE
CUSTOMER table
ID
CREDIT_LIMIT
...
202
5000
RESERVED_CREDIT table
CUST_ID
ORDER_ID
AMOUNT
202
909
1234

- 일관성

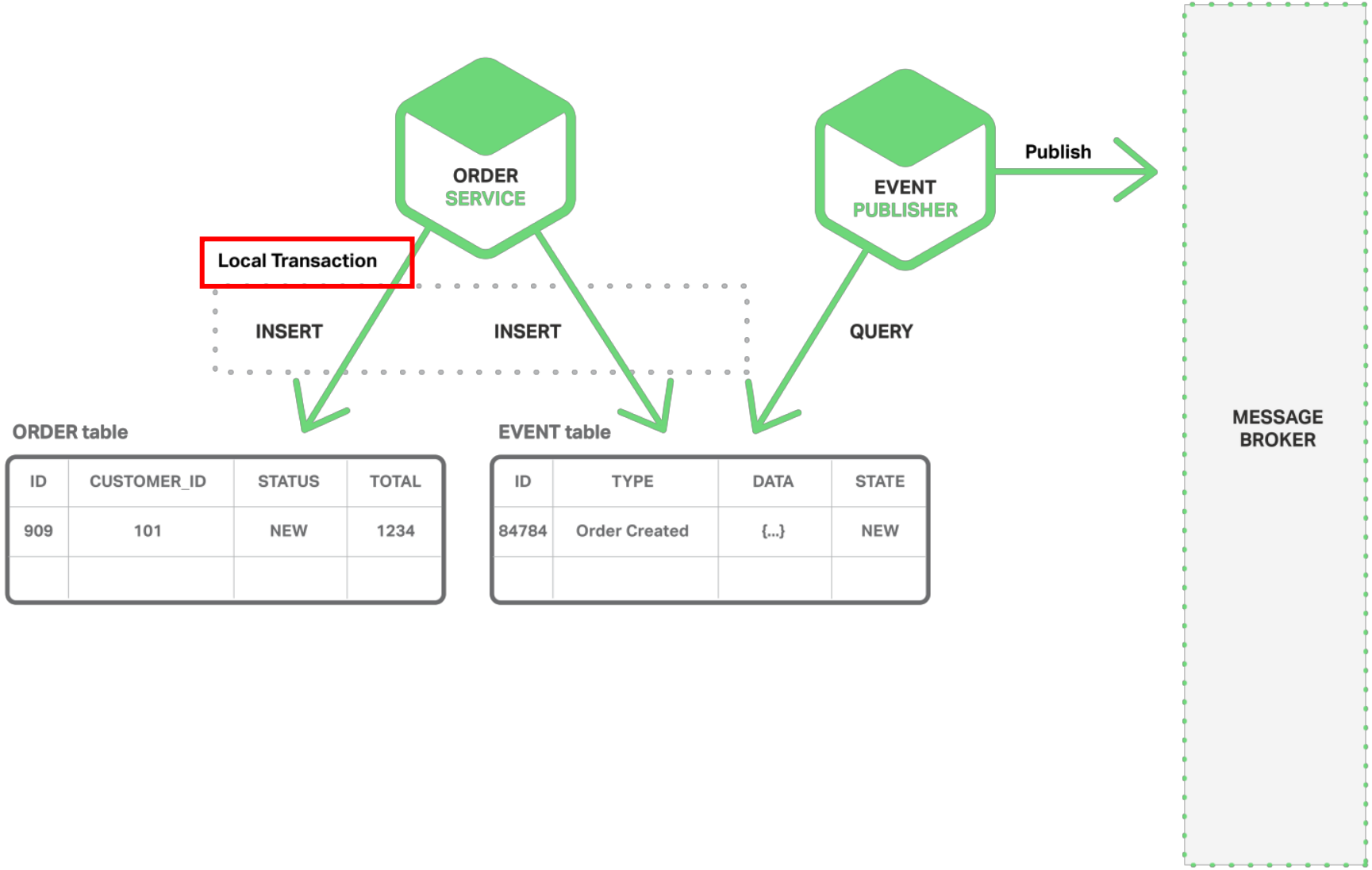

ORDER
SERVICE
Credit
Reserved
MESSAGE
BROKER
CUSTOMER
SERVICE
ORDER table
ID
CUSTOMER_ID
STATUS
TOTAL
909
101
OPEN
1234
CUSTOMER table
ID
CREDIT_LIMIT
...
202
5000
RESERVED_CREDIT table
CUST_ID
ORDER_ID
AMOUNT

- 분산된 데이터 조회

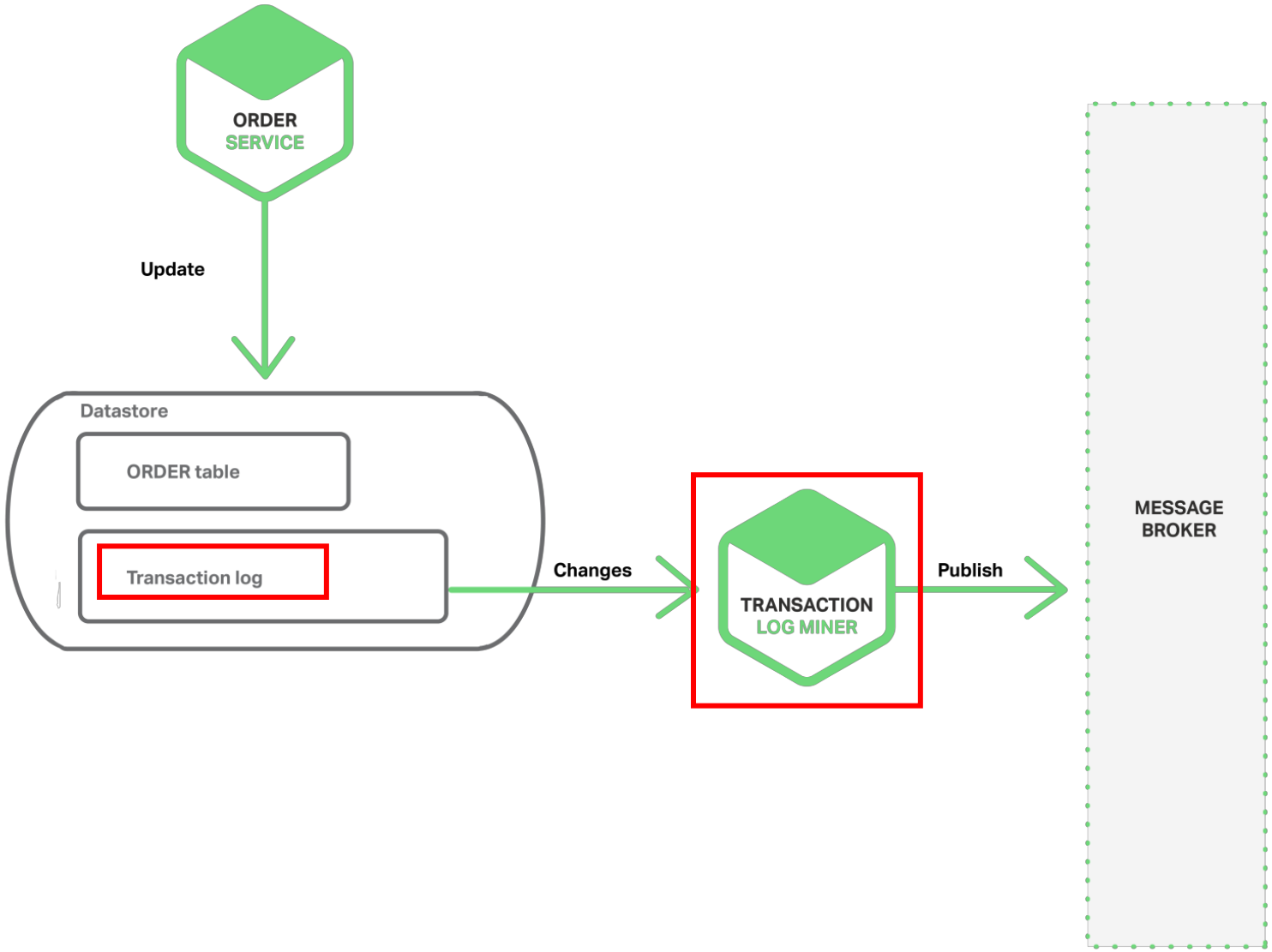


- ACID Transaction에 비해 복잡
- 반영한 모든 내용 취소하는 트랜잭션 구현
- 실시간 업데이트가 필요한 시스템에는 적합하지 않음

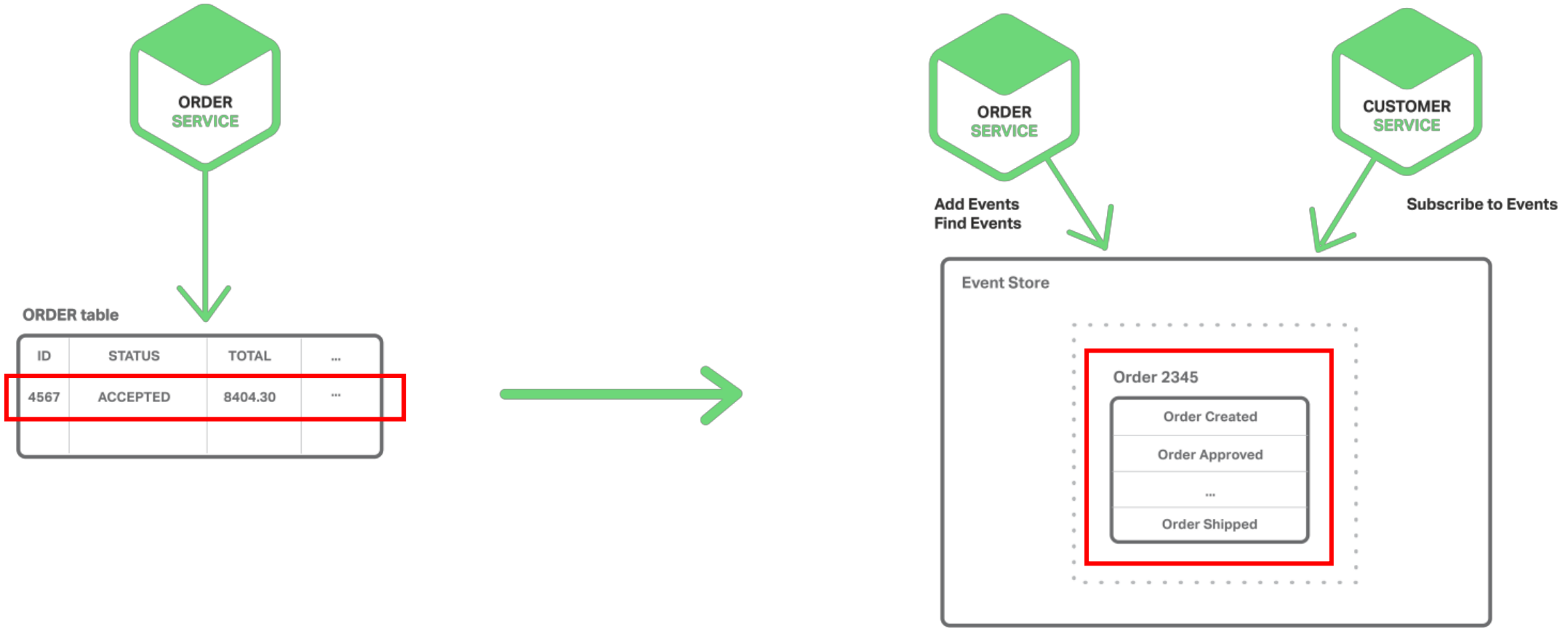
- 원자성
  - 로컬 트랜잭션을 이용해 이벤트 발행



- 원자성
  - DB 트랜잭션 로그를 모아서 이벤트 발행



- 원자성
  - 이벤트 소싱 사용


ORDER
SERVICE
ORDER table
ID
STATUS
TOTAL
...
4567
ACCEPTED
8404.30
...
ORDER
SERVICE
CUSTOMER
SERVICE
Add Events
Find Events
Subscribe to Events
Event Store
Order 2345
Order Created
Order Approved
...
Order Shipped

CQRS (Command Query Responsibility Segregation)

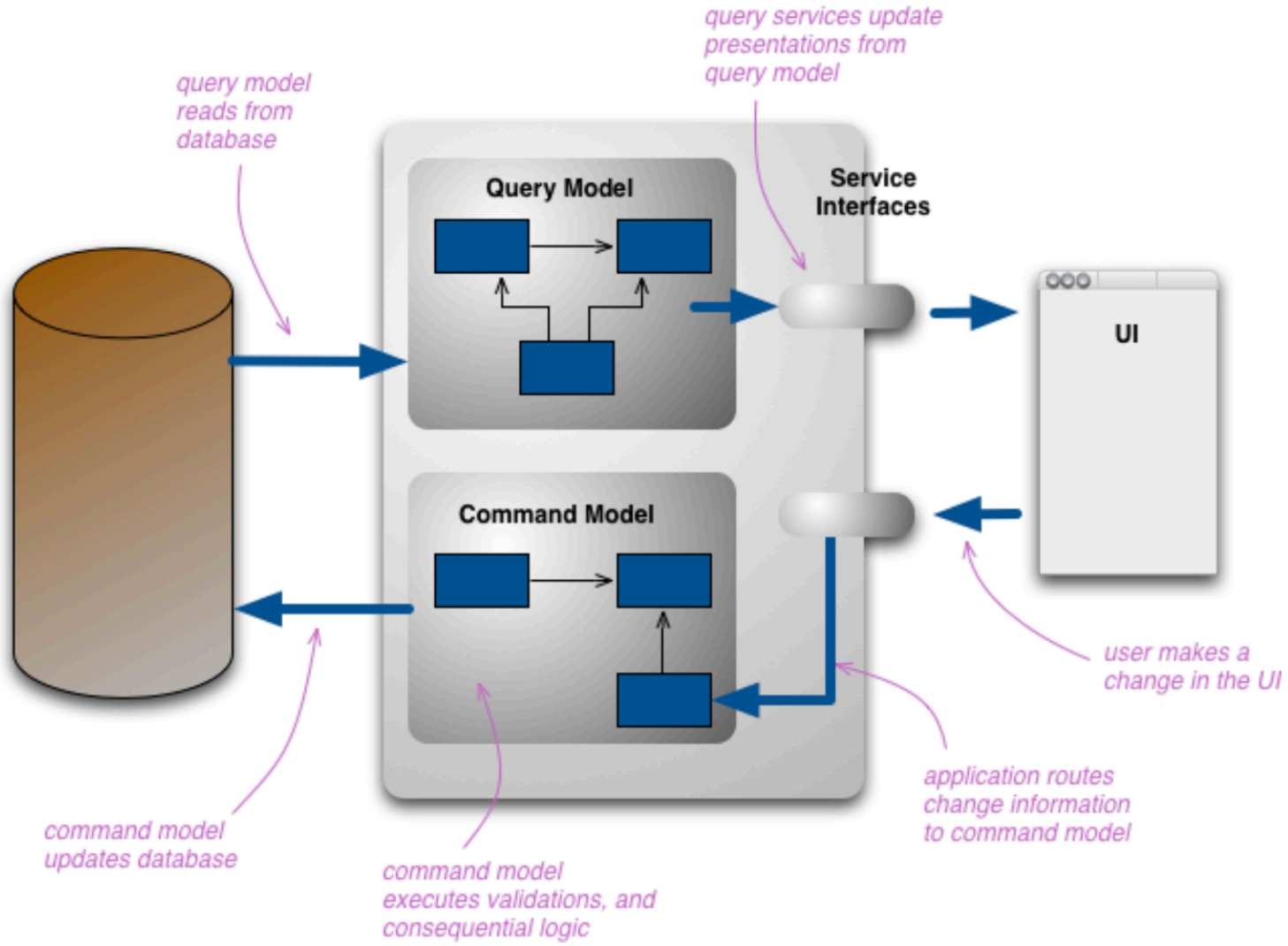
- 단일 모델을 사용할 때 발생하는 복잡도를 해결
  - 상태 변경
  - 상태 정보 조회

Query Database

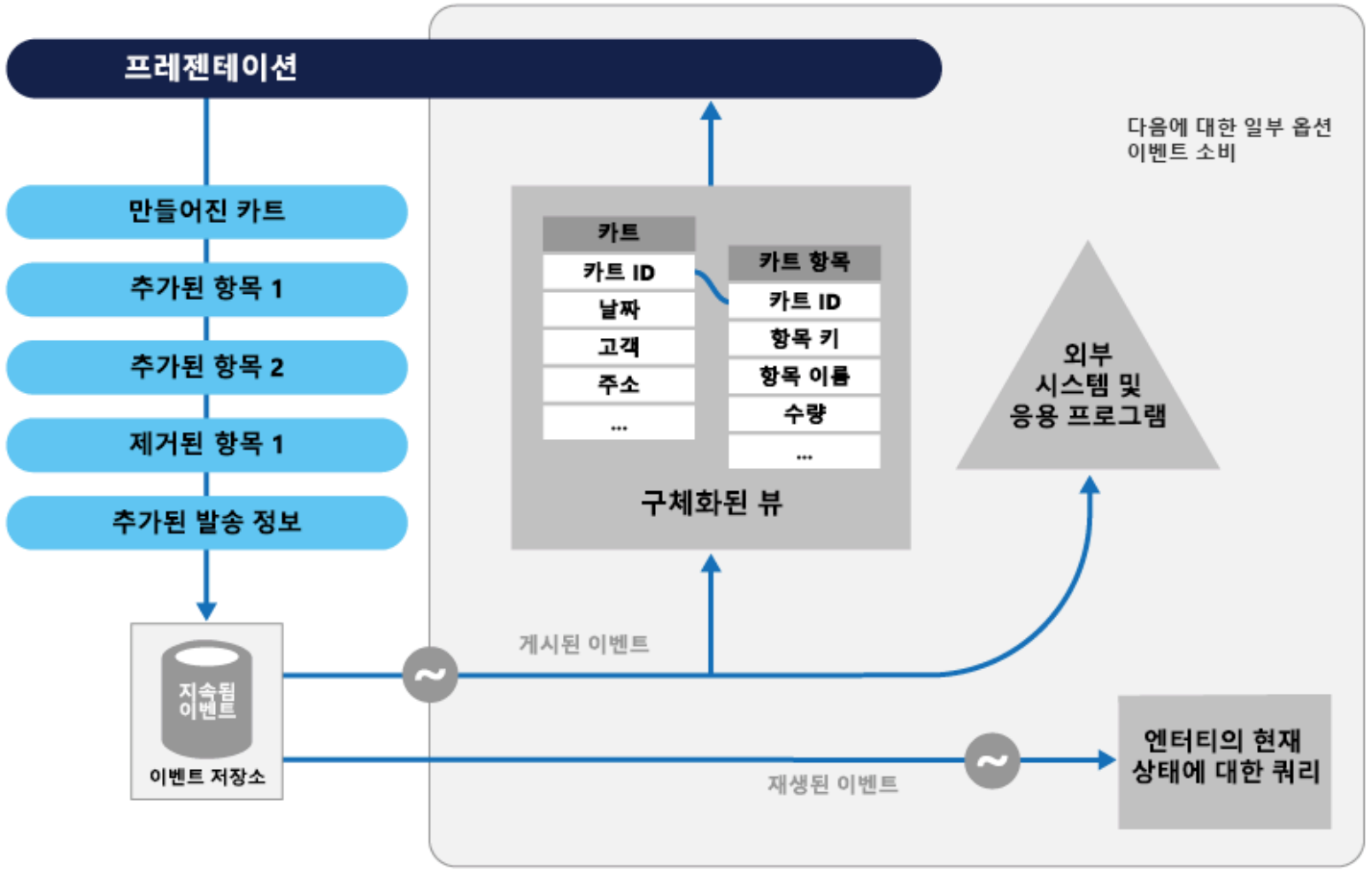
Message Broker

Command Database

- 각 모델에 적합한 구조 사용 가능
  - RDB, NoSql, Cache ...



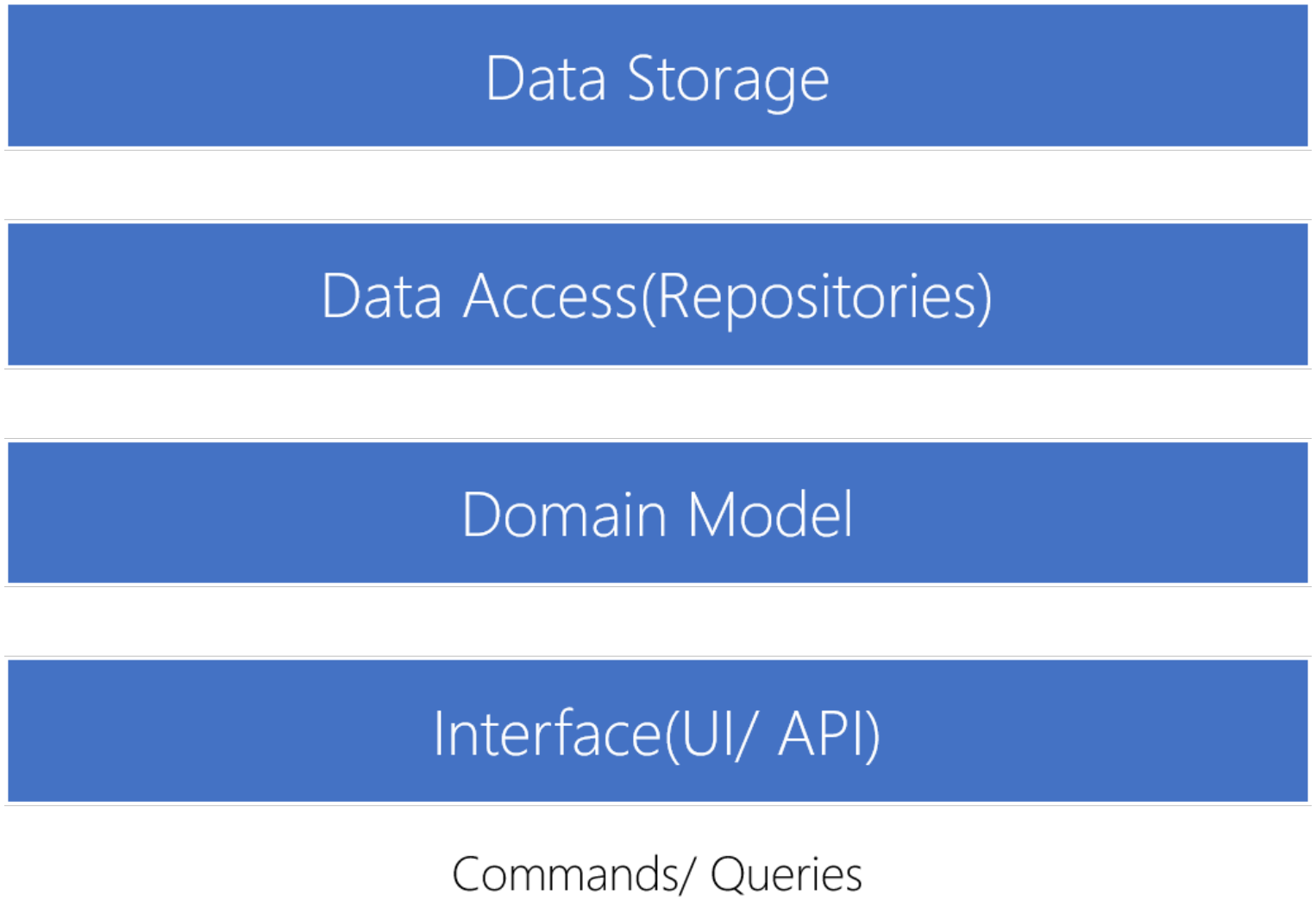
- *Event Sourcing*



id	root_id	event
1	1	카트 생성
2	1	상품1 추가
3	1	상품2 추가
4	1	상품2 삭제
5	1	배송정보 추가

- *Transaction 처리 용이*
- *실시간 업데이트가 필요한 시스템에는 적합하지 않음*

- **CQRS 미적용 다계층 아키텍처**



- **CQRS 적용 다계층 아키텍처**