

SpringCamp 2025

**10th Anniversary**

# SpringCamp 2025

Connection, Sharing, Growth, Festival

SpringCamp 2025

# 실전! MSA 트랜잭션 개발 가이드

DB 분리로 트랜잭션 보장이 안된다고?

김용욱. 삼성 SDS

# 소개

마이크로서비스 적용을 가이드하고 함께 개발하고 있습니다.



- 시스템에 적합한지 판단
- 적용 목표에 따라 서비스 설계
- 개발 가이드 및 함께 개발



스프링캠프 2024  
실전! MSA 개발 가이드

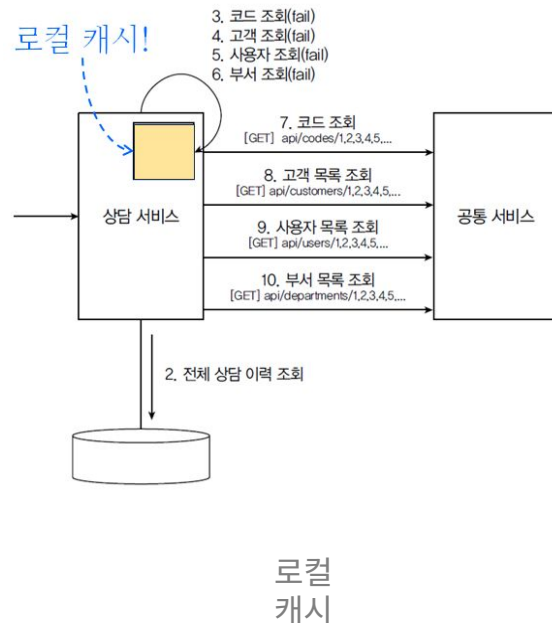
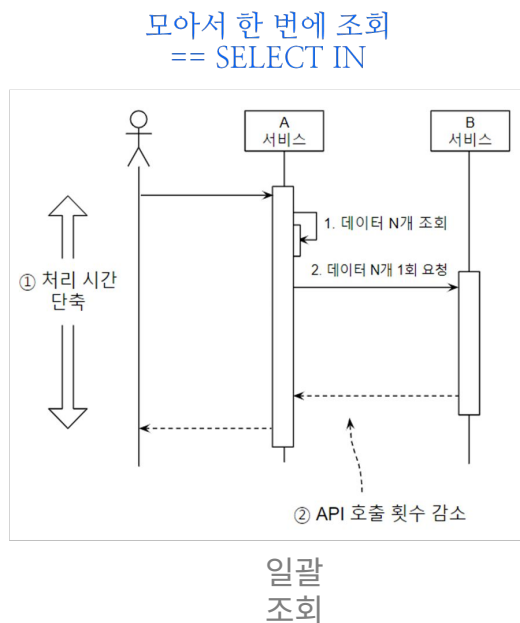
# 스프링캠프 2024 - 실전! MSA 개발 가이드



스프링캠프 2025, 가이드 w/ 예시

## 1. 데이터 참조

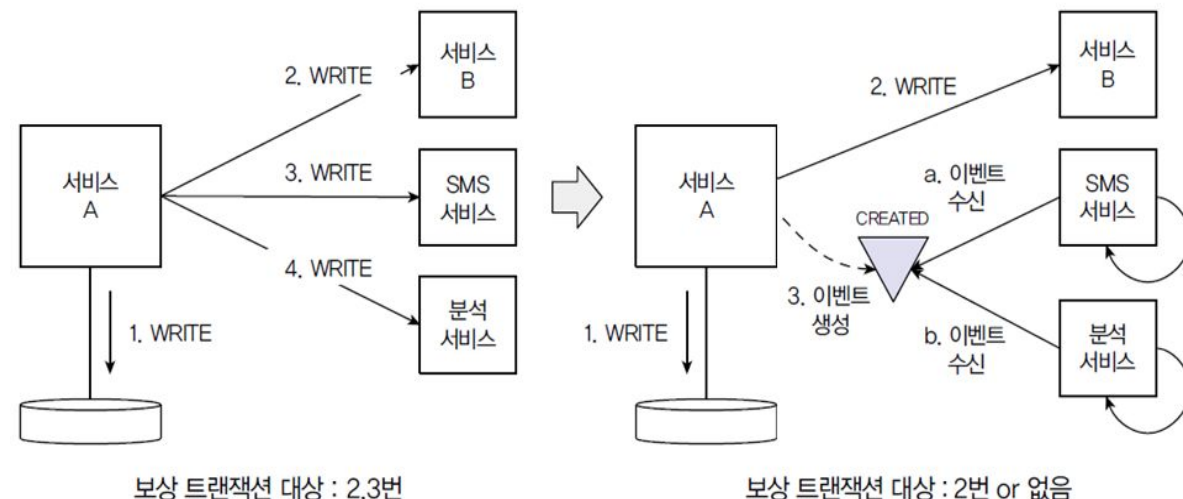
- 구현 이슈
- 패턴: 일괄조회, 로컬캐시, 복제, 모델변경
- MSA vs 모놀리식 성능 비교



## 2. 데이터 쓰기

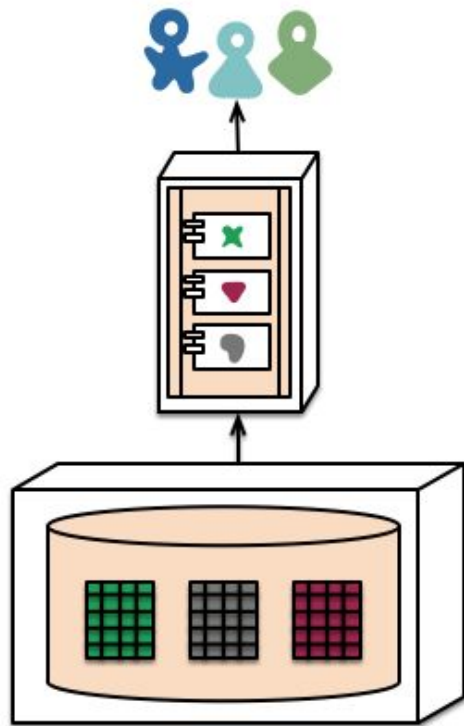
- 구현 이슈: 원자성/독립성
- 패턴: TX 나누기, 역할 분리, 모델/서비스 변경
- 안전한 재시도

- ① 실패해도 전체를 취소할 필요 없다면 이벤트로 분리
- ② 취소할 수 없는 쓰기는 이벤트로 분리

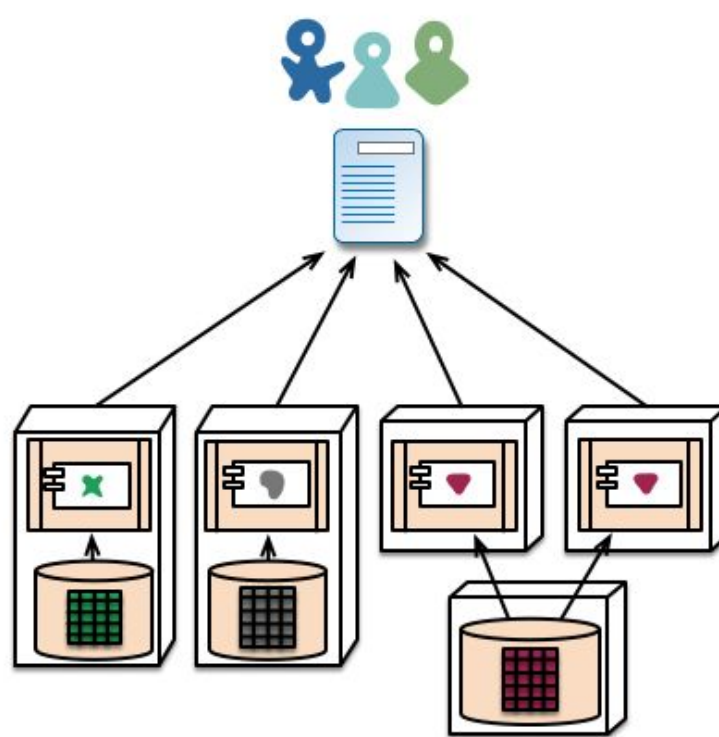


# Microservices

각 팀이 서로 독립적으로 일하는 것이 목적입니다.



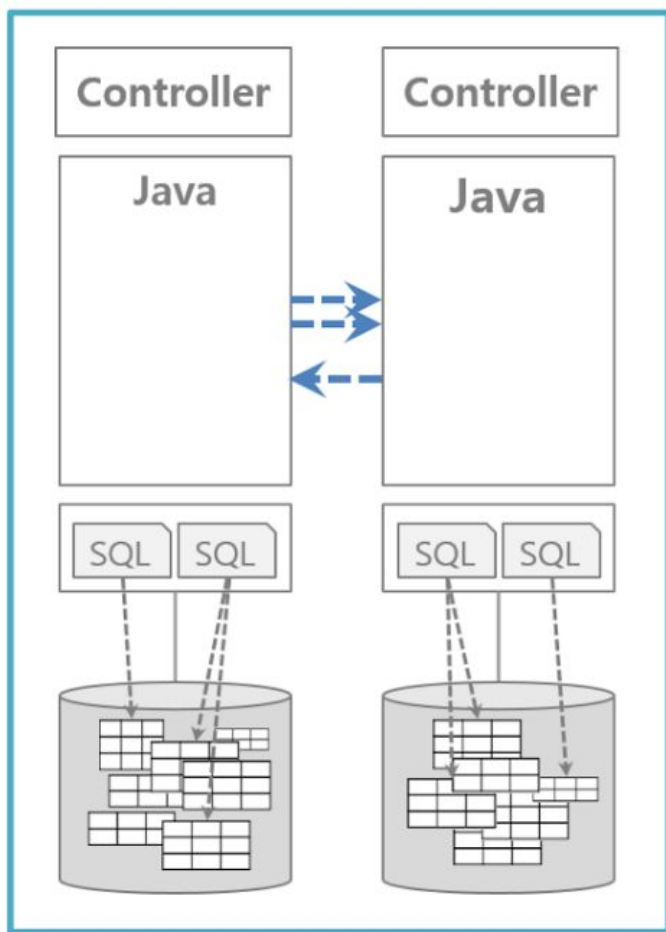
monolith - single database



microservices - application databases

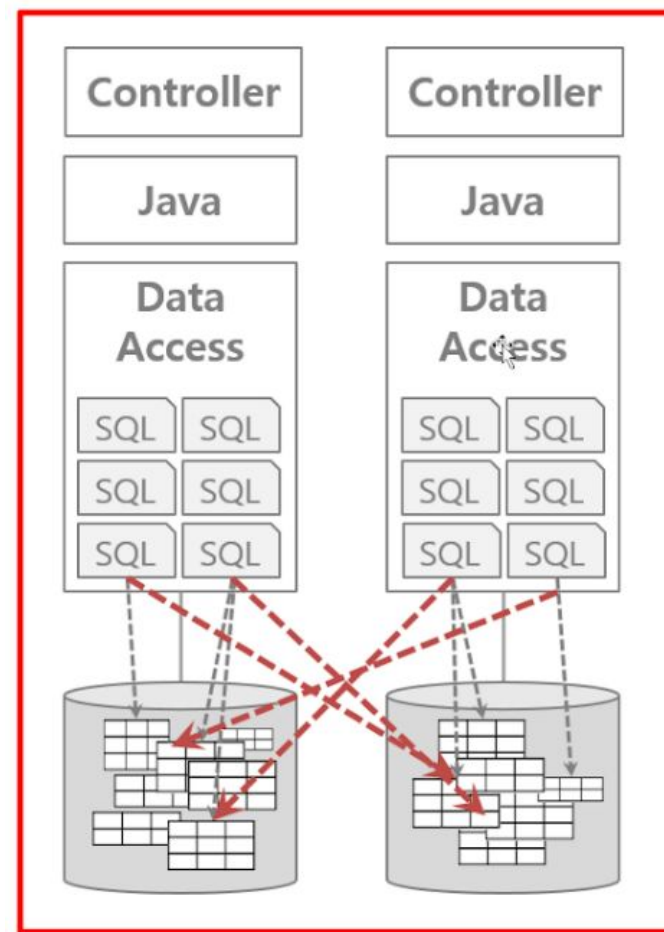
- 서비스 단위의 컴포넌트
- 비즈니스 기능 단위로 구성
- 제품 모델
- 단순한 통신 프로토콜
- 분산된 관리 체계
- 분산된 데이터 관리
- 인프라 자동화
- 실패를 고려한 설계
- 진화하는 설계

다른 서비스의 테이블에 액세스하면 안됩니다!



마이크로서비스

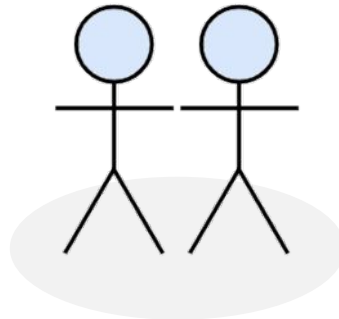
!=



껍데기만 MSA

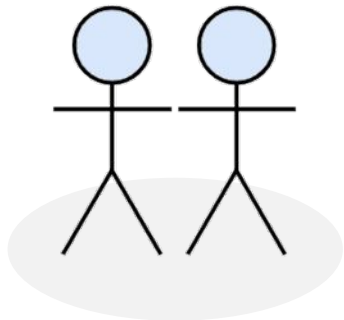
# 무(모)한 도전!

MSA 좋네! 그런데..  
서비스간에 트랜잭션 보장이 안되네?

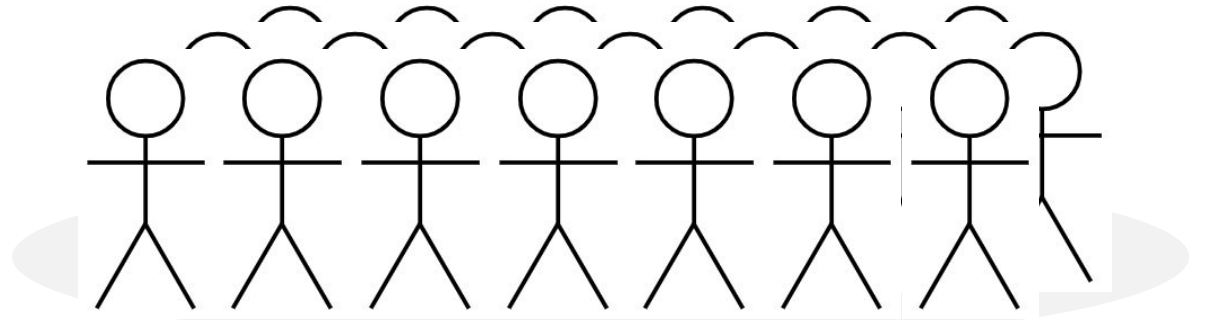


기술 담당

K8s, API GW는 제가 할테니  
트랜잭션은 알아서 하셔야.. ^^;

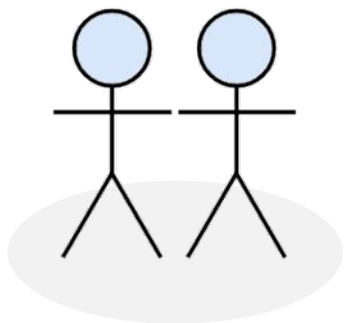
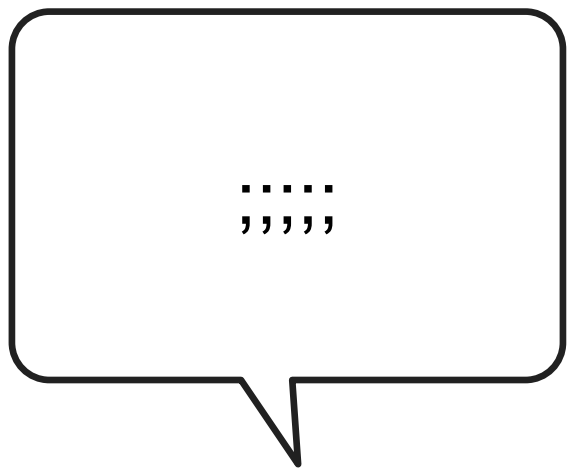


기술 담당

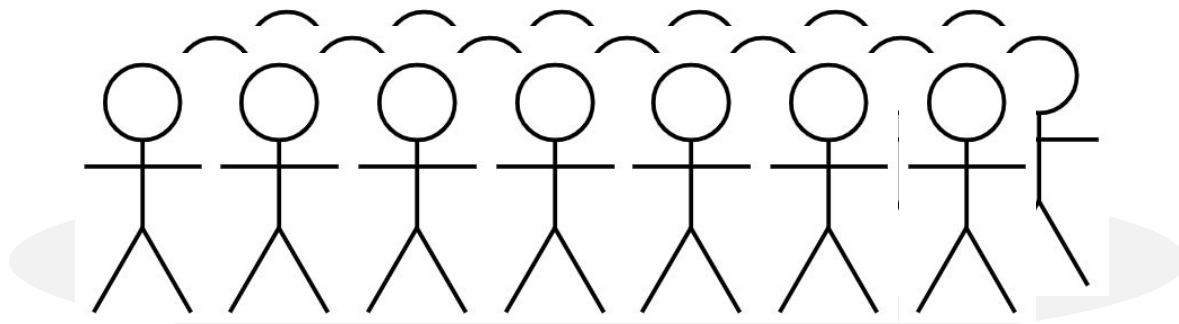
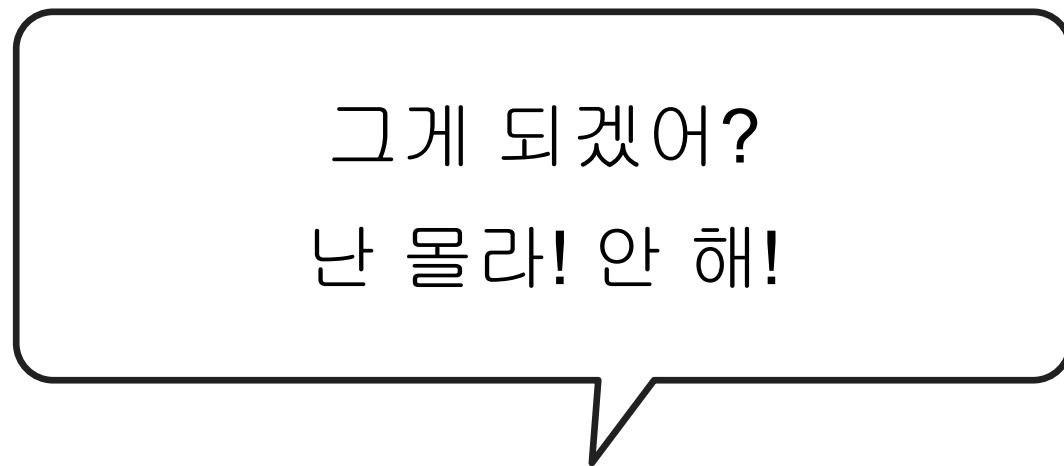


업무 담당



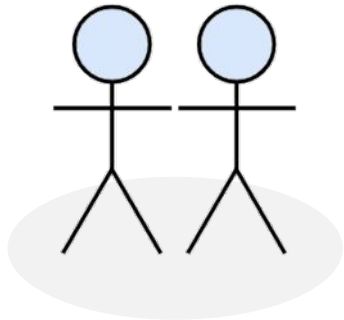


기술 담당



업무 담당

분산 트랜잭션 FW이  
있으면 되겠네!



기술 담당

# 2PC is not an option



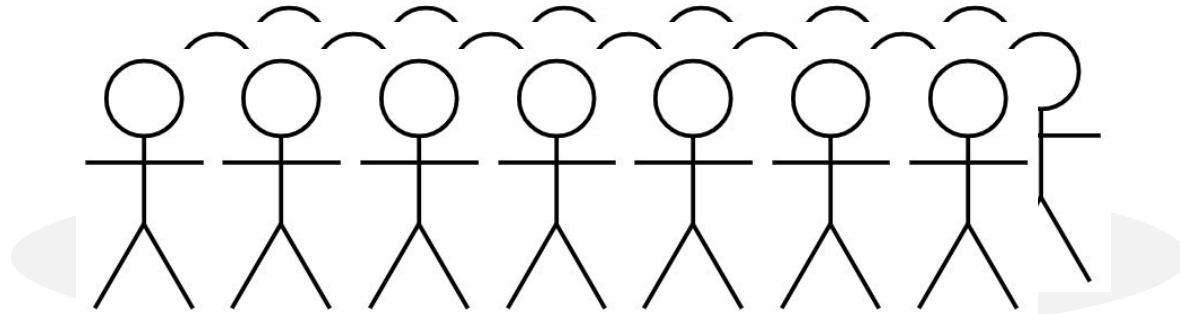
- Guarantees consistency

**BUT**

- 2PC coordinator is a single point of failure
- Chatty: at least  $O(4n)$  messages, with retries  $O(n^2)$
- Reduced throughput due to locks
- Not supported by many NoSQL databases (or message brokers)
- CAP theorem  $\Rightarrow$  2PC impacts availability
- ....

# 가야하는 길!

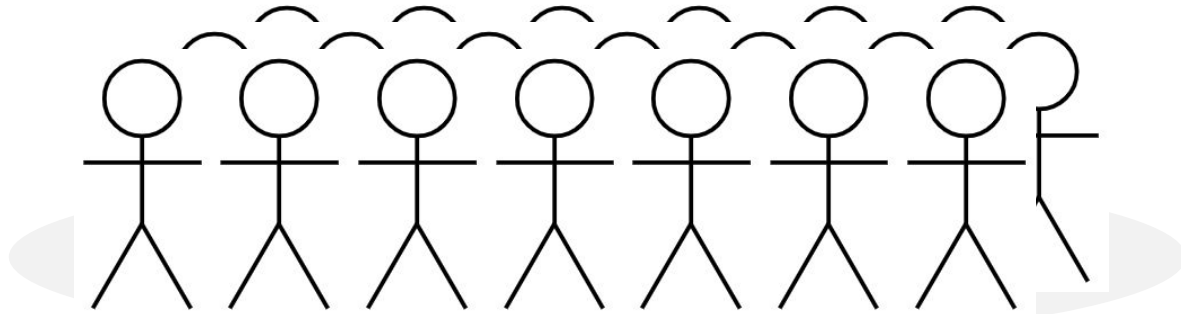
진심이야?  
알아서 해야한다고?



업무 담당

# 가야하는 길!

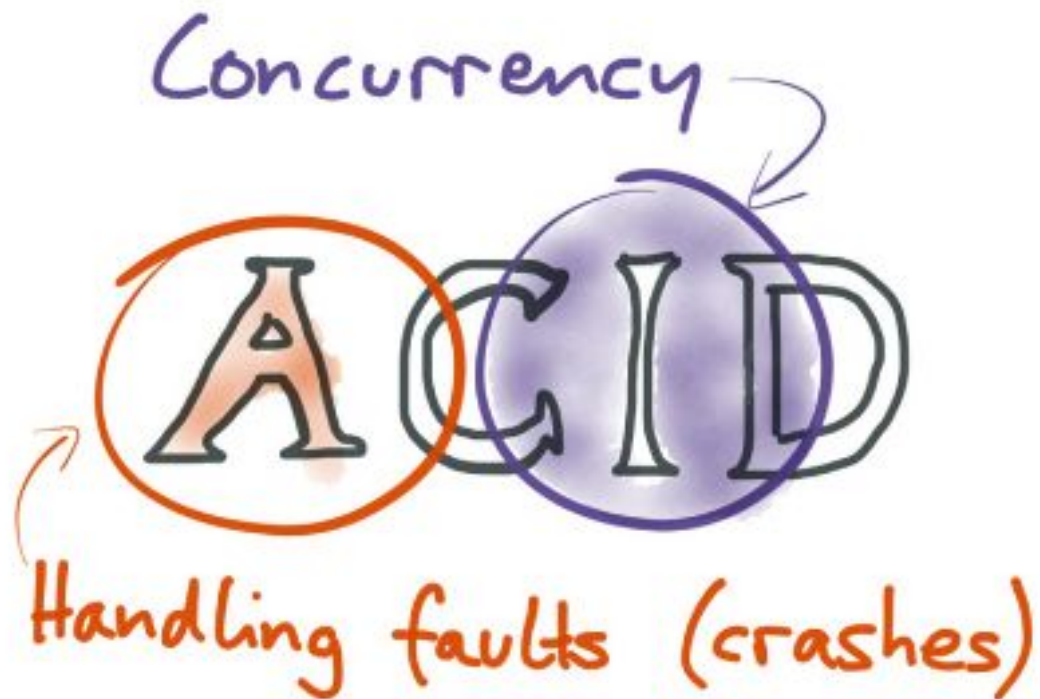
그런데 뭐가 문제지?



업무 담당

# 트랜잭션 보장이 안되?!

여러 서비스에 걸친 쓰기는 트랜잭션 보장이 안되요.



Atomicity (원자성) : DB Rollback

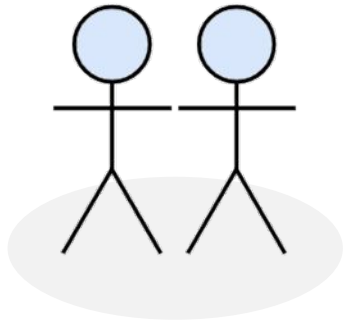
~~Consistency (일관성)~~

Isolation (독립성) : Read Committed

~~Durability (지속성)~~

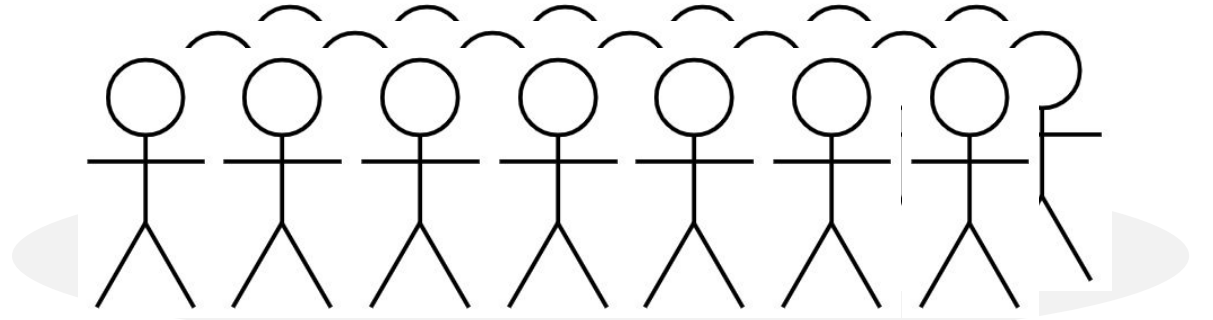
# 가야하는 길!

남들도  
하니까요;;



기술 담당

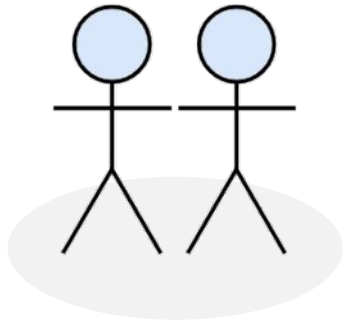
그게 정말 가능해?  
어떻게 해야할 지 모르겠는데?  
위험할 수도 있을 거 아냐?



업무 담당

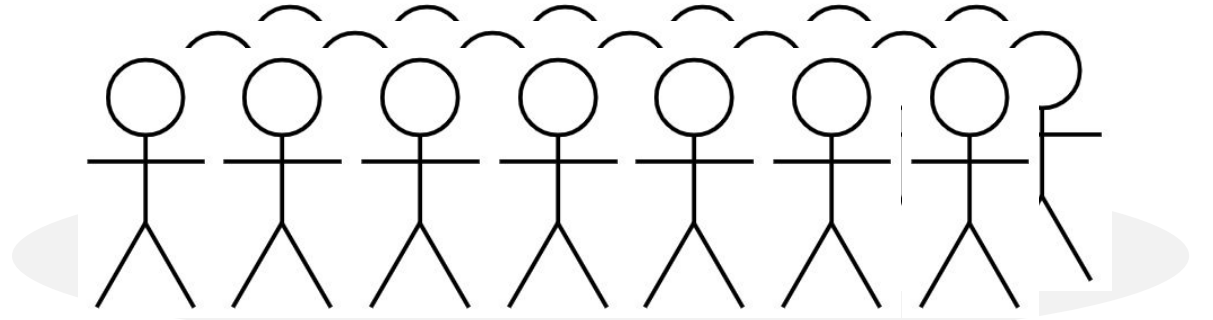
# 가야하는 길!

불가능한 부분도 있지만  
의외로 큰 탈 없이 잘 되더라고요.



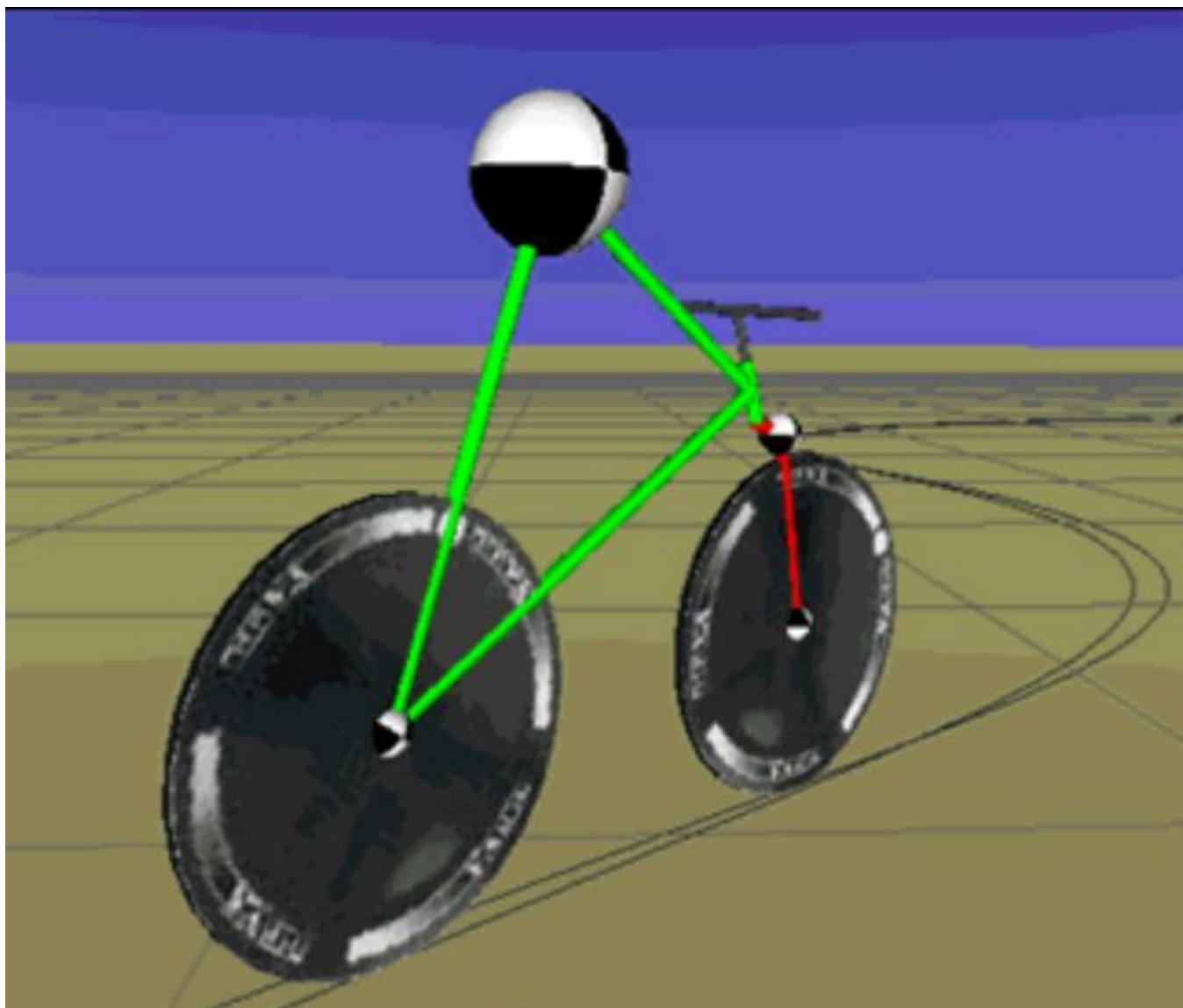
기술 담당

그래..

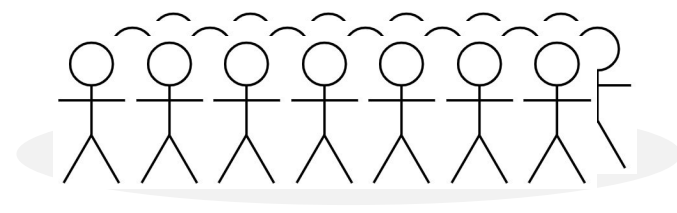


업무 담당





두 발인데  
당연히 넘어지겠지!



# 자전거가 넘어지지 않는 이유?

자전거가 넘어지지 않고 달릴 수 있는 이유는 여러 요인이 복합적으로 작용하기 때문입니다. 주요 원인으로는 회전 관성, 자이로 효과, 캐스터 각, 그리고 운전자의 반사적인 균형 유지 행동 등이 있습니다.

## 1. 회전 관성 (또는 각운동량 보존):

- 자전거 바퀴가 회전할 때 각운동량을 갖게 되는데, 이 각운동량은 바퀴가 회전하는 방향을 유지하려는 경향이 있습니다. 마치 팽이가 계속 회전하는 것과 같습니다.
- 바퀴의 회전 관성은 바퀴가 기울어져도 반대 방향으로 회전하려는 힘을 생성하여 자전거가 똑바로 서 있도록 돕습니다.

## 2. 자이로 효과:

- 자전거의 바퀴는 회전하며 마치 자이로스코프처럼 동작합니다. 바퀴에 힘이 가해지면, 바퀴는 그 힘에 대해 직각 방향으로 회전하려는 경향이 있습니다.
- 이러한 자이로 효과는 바퀴의 회전을 안정시키고, 자전거가 넘어지지 않도록 돕습니다.

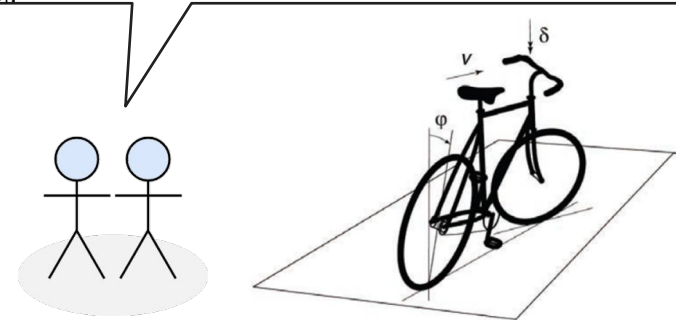
## 3. 캐스터 각:

- 자전거 앞바퀴 축이 앞쪽으로 약간 기울어져 있는데, 이를 캐스터 각이라고 합니다.
- 이 캐스터 각은 자전거가 직진성을 갖도록 도와주고, 앞바퀴가 기울어질 때 반대 방향으로 돌아가도록 하여 자전거가 균형을 유지할 수 있도록 합니다.

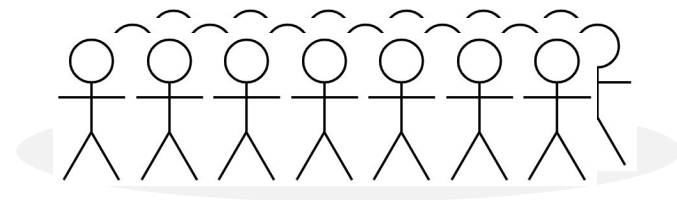
## 4. 운전자의 균형 유지 행동:

- 자전거를 타는 사람은 본능적으로 자전거가 기울어지는 방향 반대로 핸들을 움직여 균형을 잡습니다.
- 이러한 반사적인 행동은 자전거가 넘어지지 않도록 돕는 중요한 요소 중 하나입니다.

결론적으로, 자전거가 넘어지지 않고 달릴 수 있는 것은 회전 관성, 자이로 효과, 캐스터 각, 그리고 운전자의 균형 유지 행동이 복합적으로 작용하는 결과입니다.



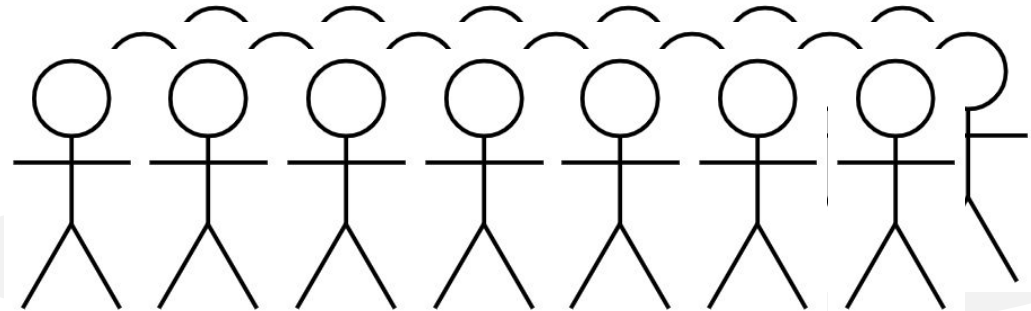
그래서 넘어진 적이  
한 번도 없어?



# 가야하는 길!



좋아! 해볼테니까  
가이드를 줘봐.

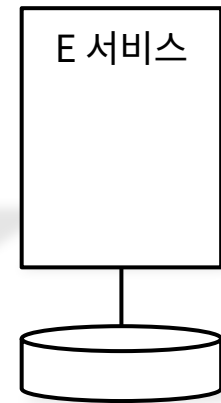
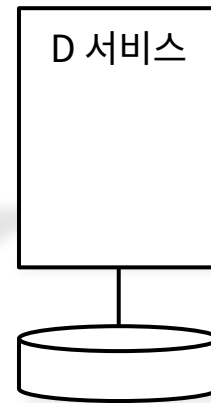
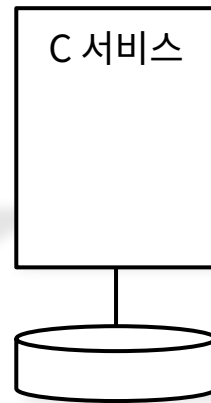
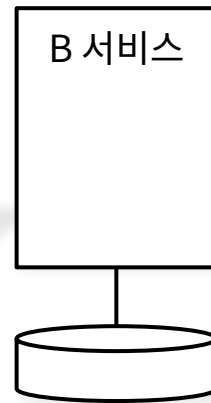
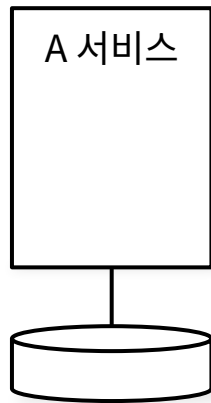
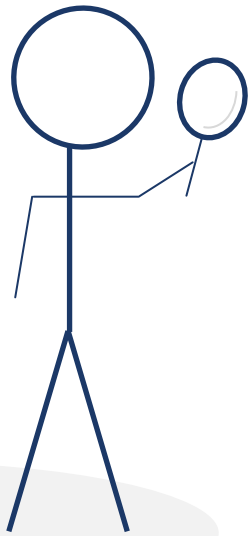


업무 담당

그렇다고...



서비스로 나누면  
어떻게 되는지 보자고



트랜잭션 개발 가이드

# 트랜잭션 개발 가이드

## 1. 데이터 오너십 원칙

## 2. 실패한 트랜잭션의 처리

- 트랜잭션 취소/재시도

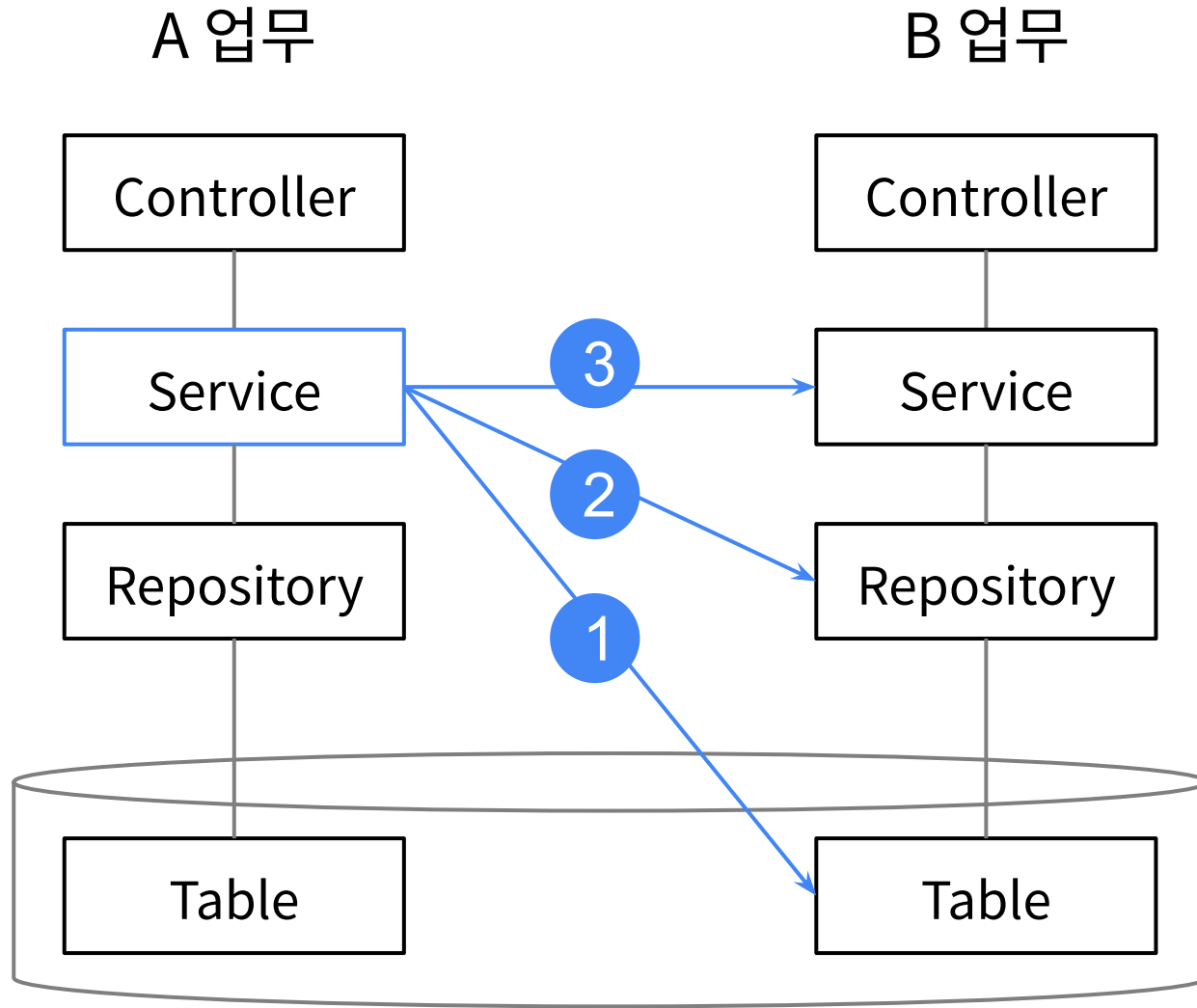
## 3. 트랜잭션 격리성 보완

- 레코드/트랜잭션 잠금

## 4. 신뢰할 수 있는 트랜잭션 구현

- API/Event, 멍등성 처리

# Q: 다른 업무를 참조할 때 익숙한 방식은?



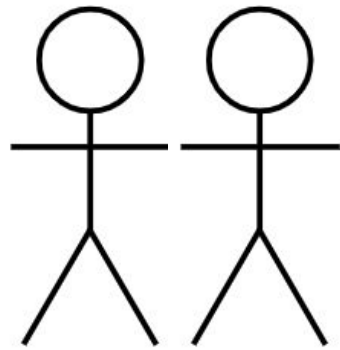
- ① 테이블을 직접 Join
- ② DAO(Repository)를 호출
- ③ ServiceImpl을 호출

# 데이터 오너십 원칙

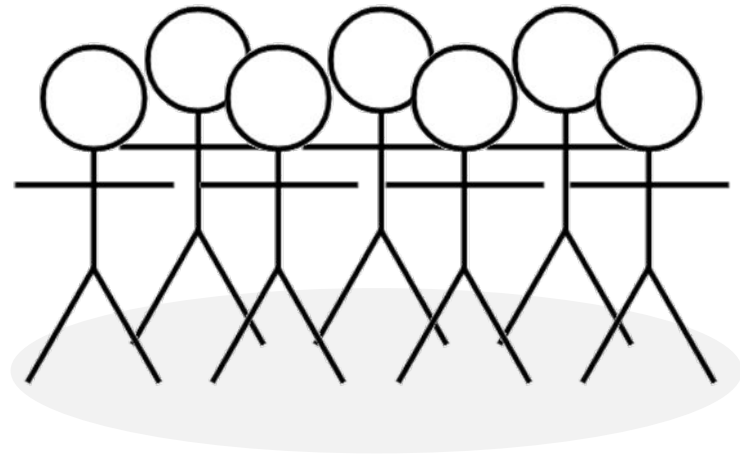


오너십을 가진 서비스만 데이터를 씁니다. 여러 서비스가 얹어 쓰면 안되요!

이 데이터는 저만 변경합니다.  
필요한 분은 말씀하세요!



고객 개발팀





# 속성으로 나눌 수도

고객 정보

ID	A	B	C
001			
002			
003			
004			
005			

고객 개발팀

ID	A	B
001		
002		
003		
004		
005		

상담 개발팀

ID	C
001	
002	
003	
004	
005	

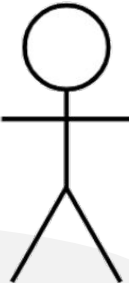
# 레코드로 나눌 수도

통화 이력

ID	Source	A	B
001			
002			
003			
004			
005			
006			
007			

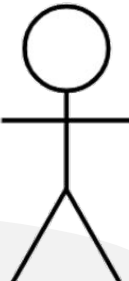
콜센터 개발팀

ID	Source	A	B
001			
003			
005			
006			



VoC 개발팀

ID	Source	A	B
002			
004			
007			



# 트랜잭션 개발 가이드

## 1. 데이터 오너십 원칙

## 2. 실패한 트랜잭션의 처리

- 트랜잭션 취소/재시도

## 3. 트랜잭션 격리성 보완

- 레코드/트랜잭션 잠금

## 4. 신뢰할 수 있는 트랜잭션 구현

- API/Event, 멍등성 처리

(1) 원자성 관련 이슈

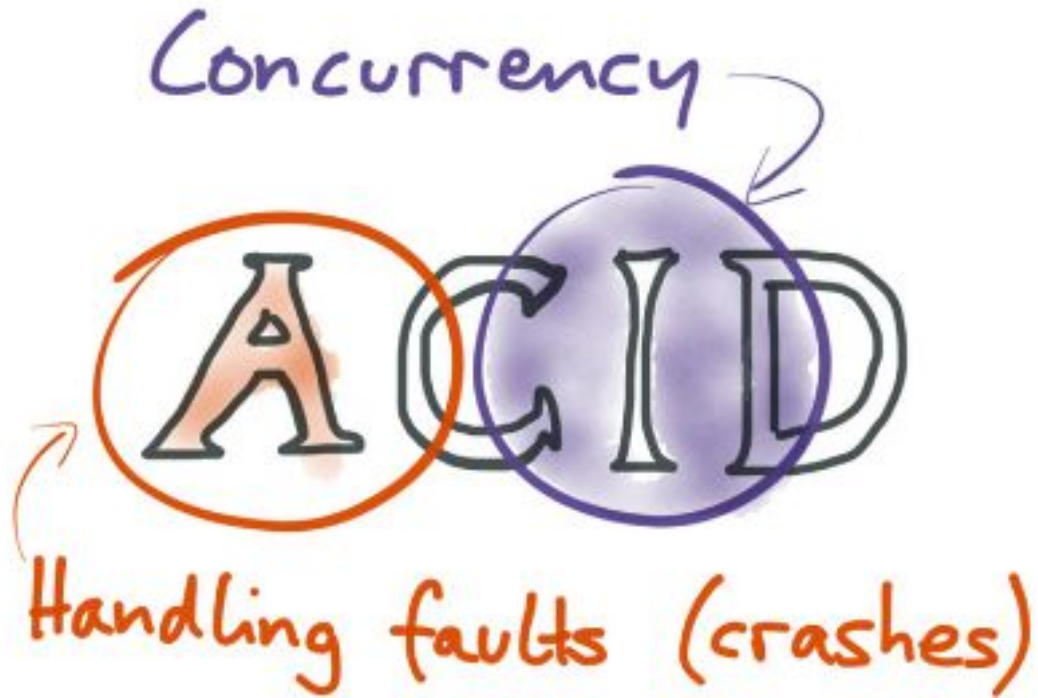
(2) 케이스 별 처리 방법

(3) Best 조합 & 예시

(4) 고급

# (1) 원자성 관련 이슈

① 트랜잭션이 실패했을 때 자동으로 취소가 안되요 (Rollback)



Atomicity (원자성) : DB Rollback

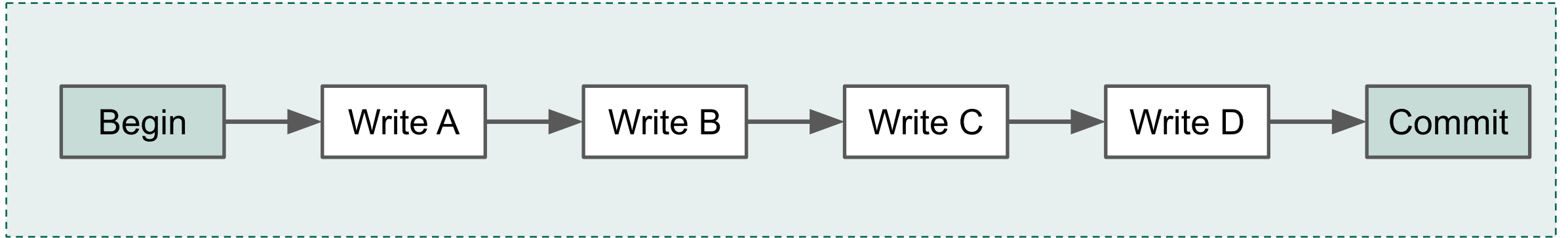
~~Consistency (일관성)~~

~~Isolation (독립성) : Read Committed~~

~~Durability (지속성)~~

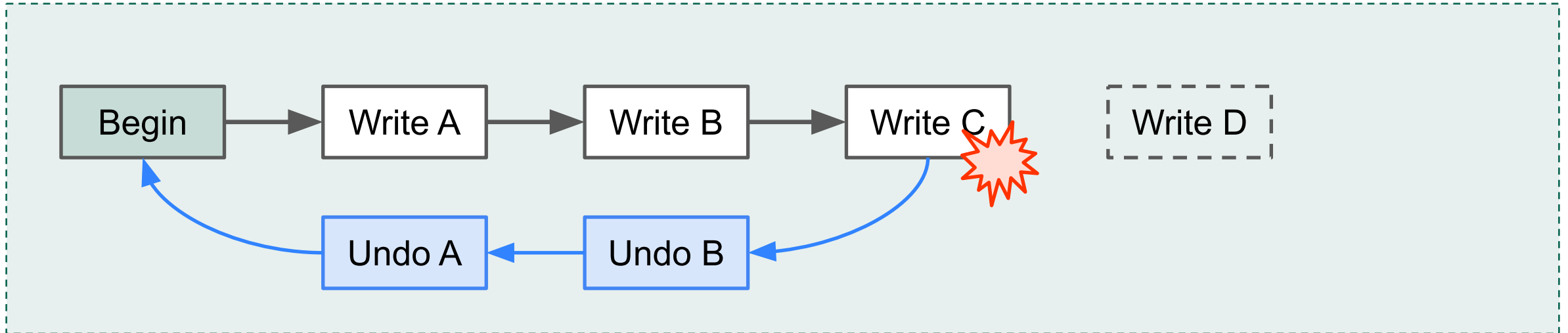
## 트랜잭션 성공

DB



## 실패 & Rollback

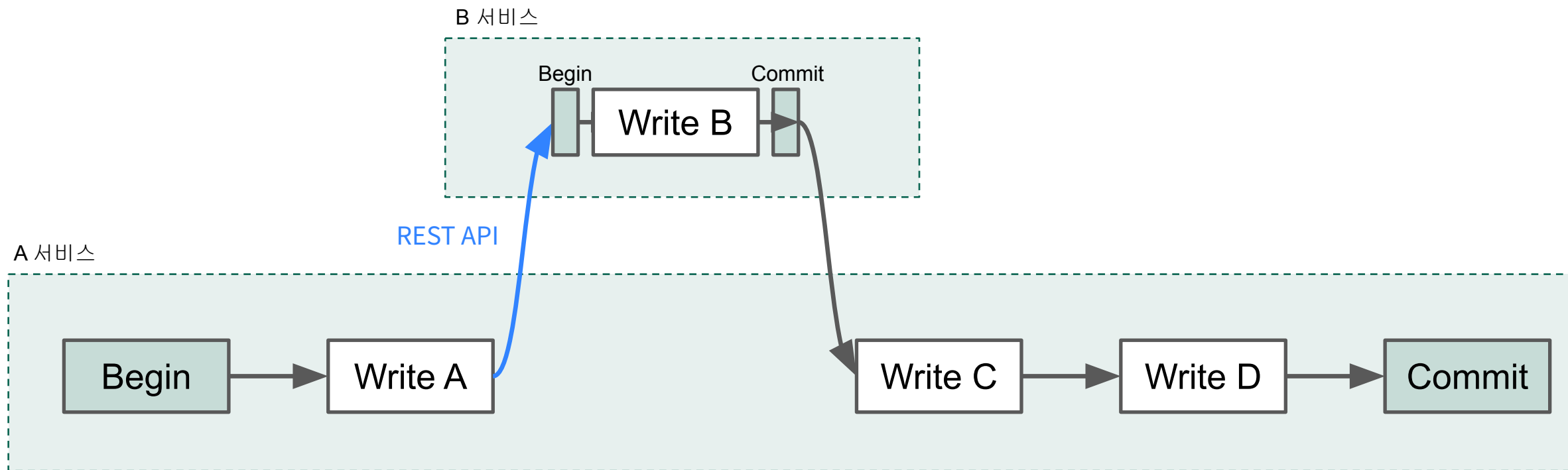
DB



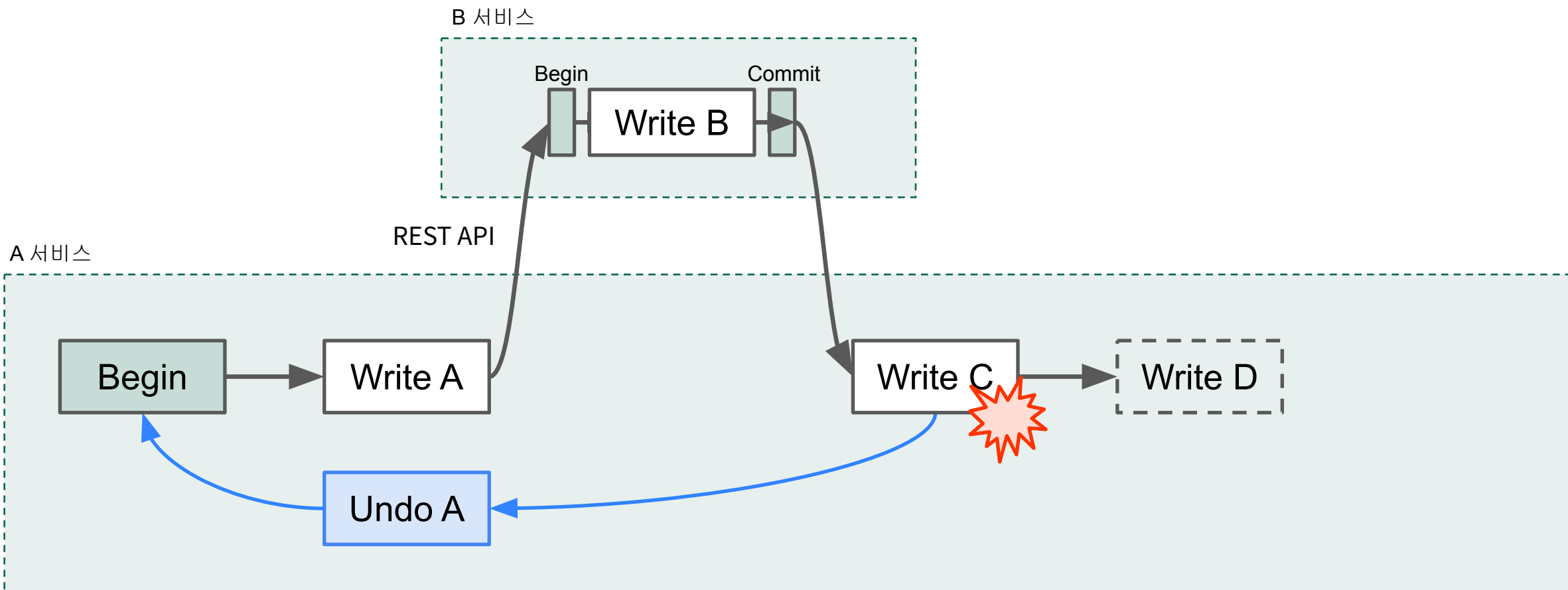
DB

트랜잭션 시작 전과 동일한 상태

# 마이크로서비스 케이스

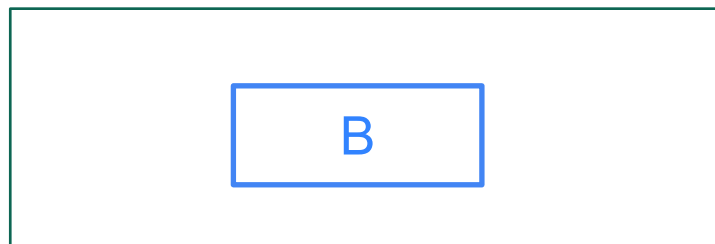




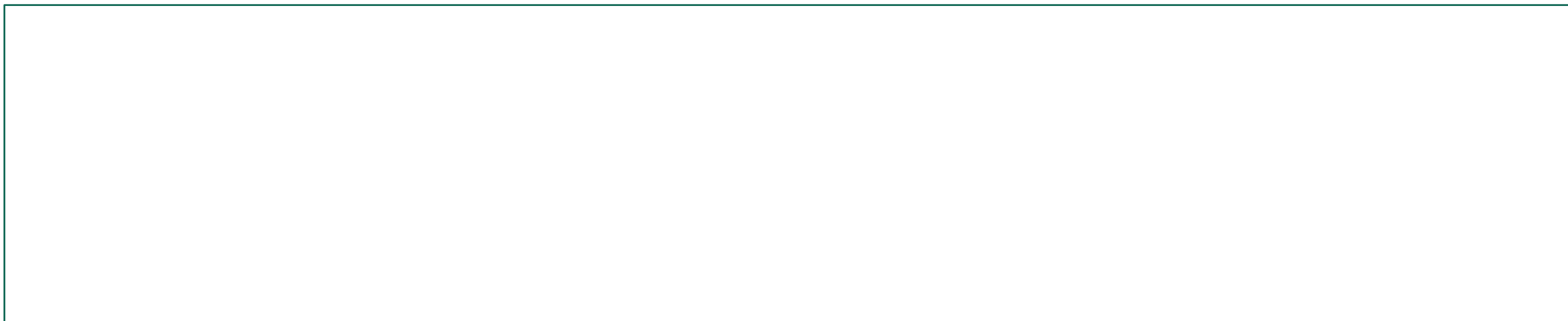


A 서비스의 데이터는 Rollback으로 삭제되지만  
B서비스의 데이터는 남게 됨

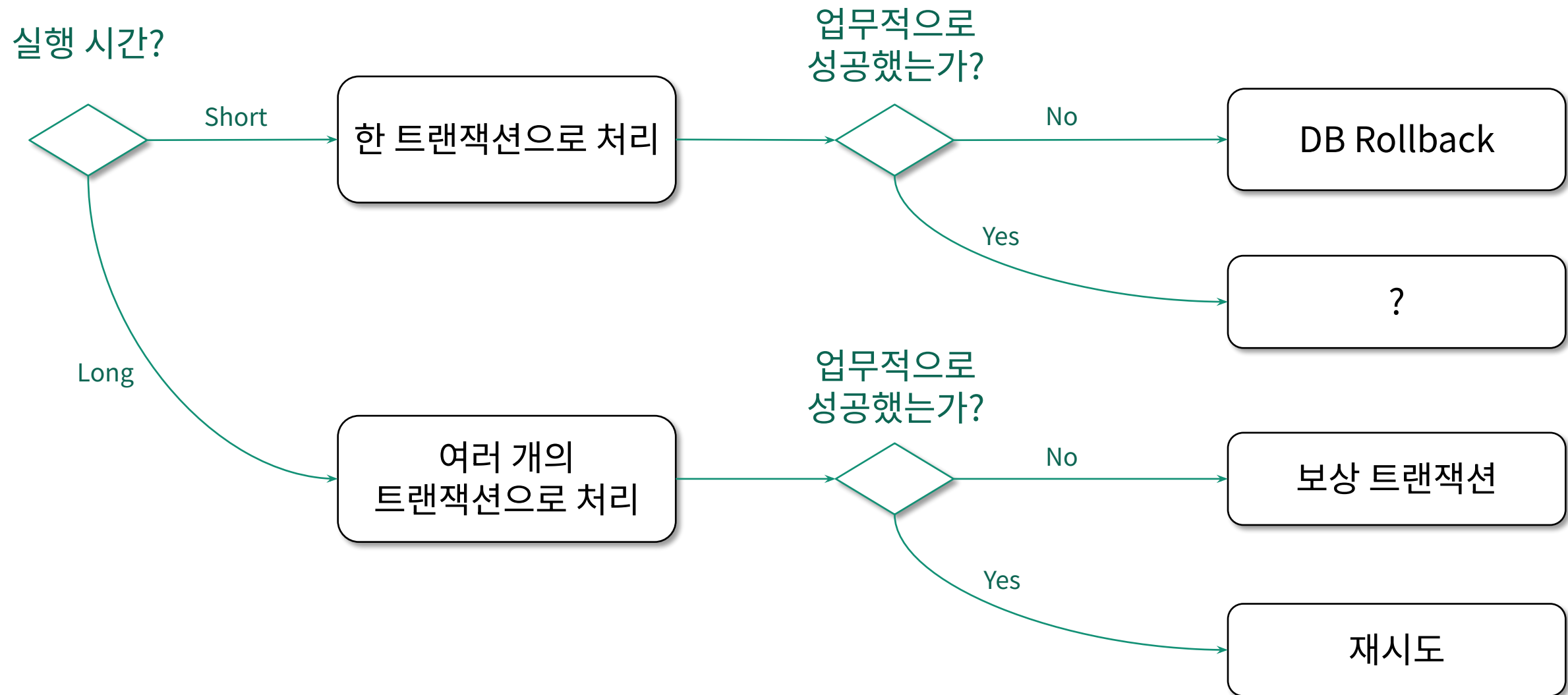
B 서비스



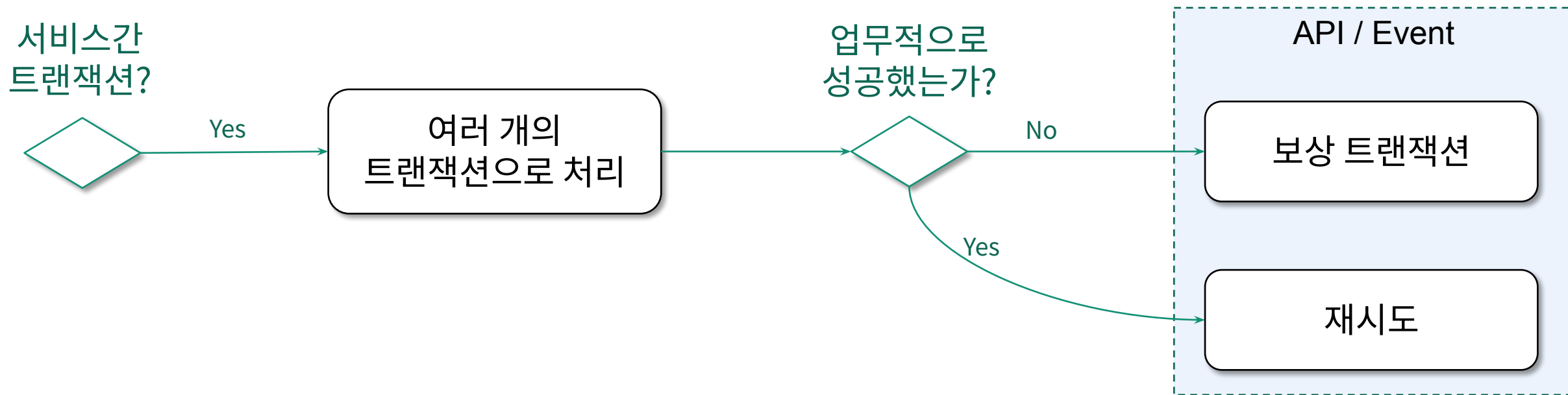
A 서비스



# 모놀리식의 트랜잭션 실패 대응 방법



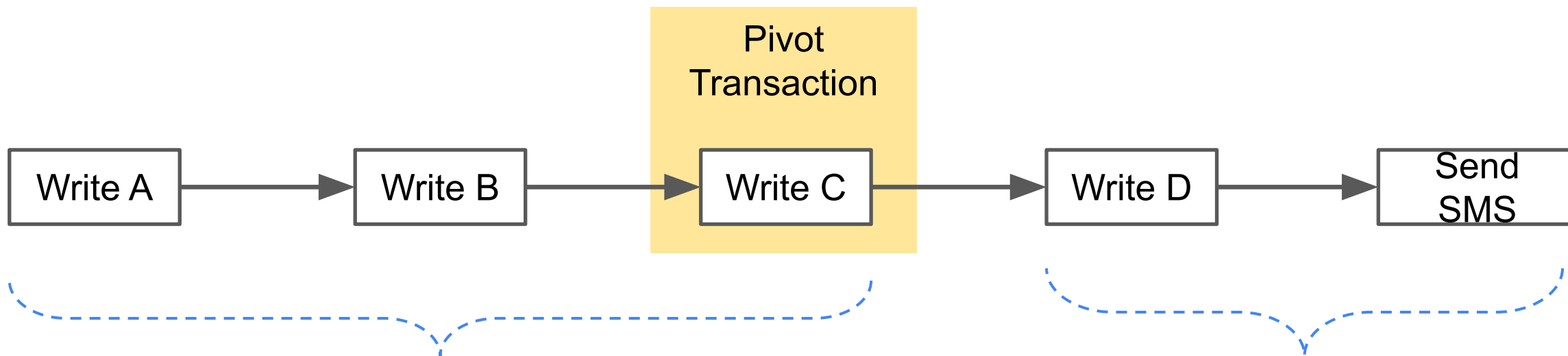
## (2) 서비스 간 트랜잭션 실패 대응 방법



# Pivot Transaction

트랜잭션 중 에러가 발생했을 때 재시도할지 취소할지를 결정하는 기준

업무적으로 성공하는 지점 여부를  
결정하는 지점



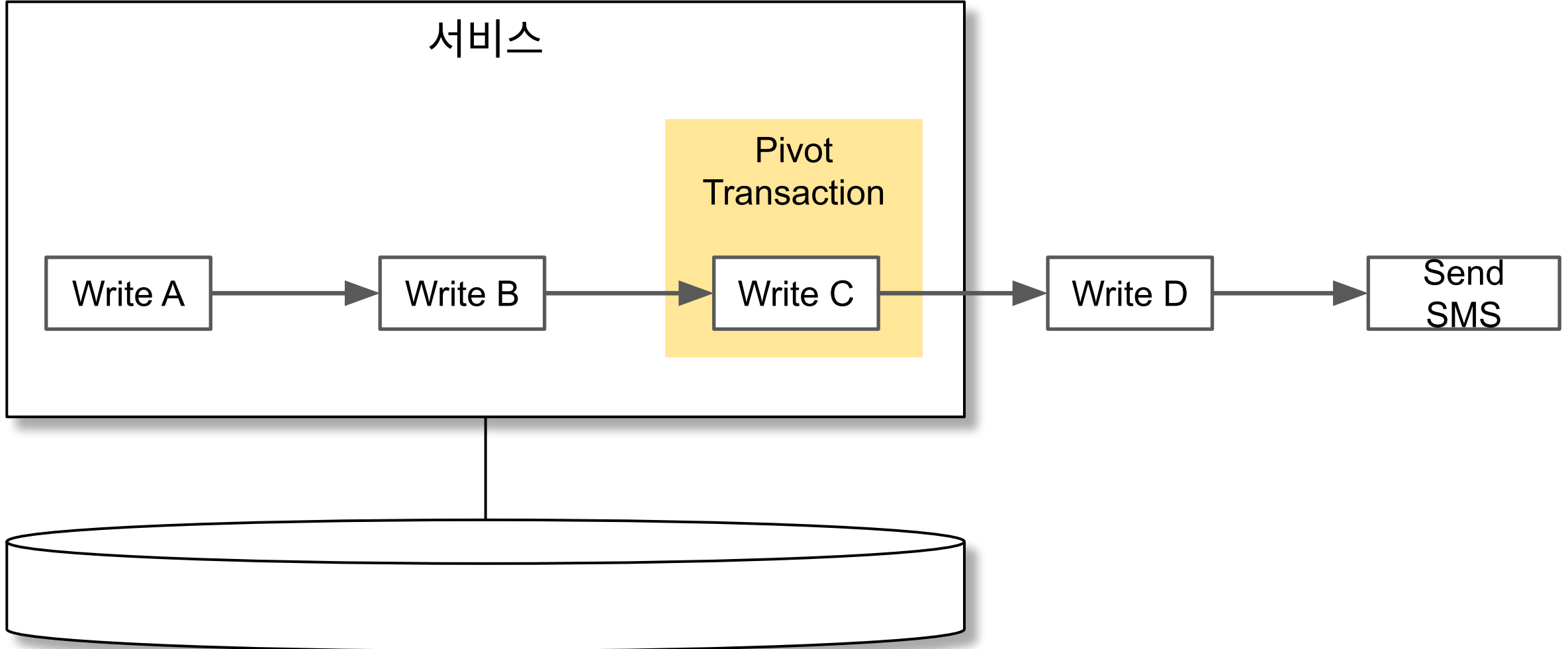
실패하면 트랜잭션 취소  
사용자는 응답 대기  
재시도는 사용자가 주관

실패하면 이벤트로 재시도  
계속 실패하면 시스템 담당자가 처리

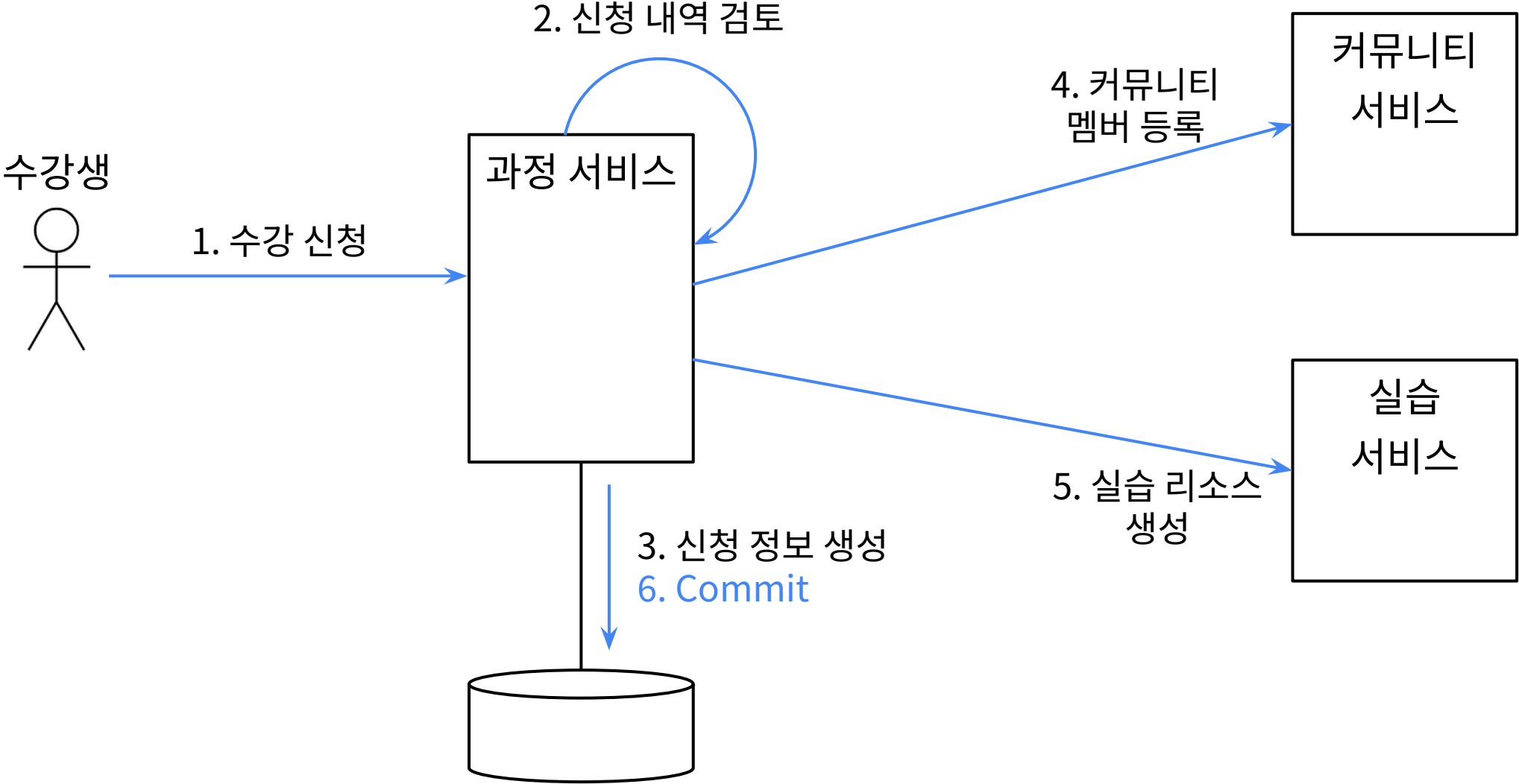
※ API로 재시도하는건 위험할 수도 있어요

# 경험적으로 ...

의미 있는 업무 단위  $\leq$  서비스 업무 범위  
(인 경우가 많습니다...)

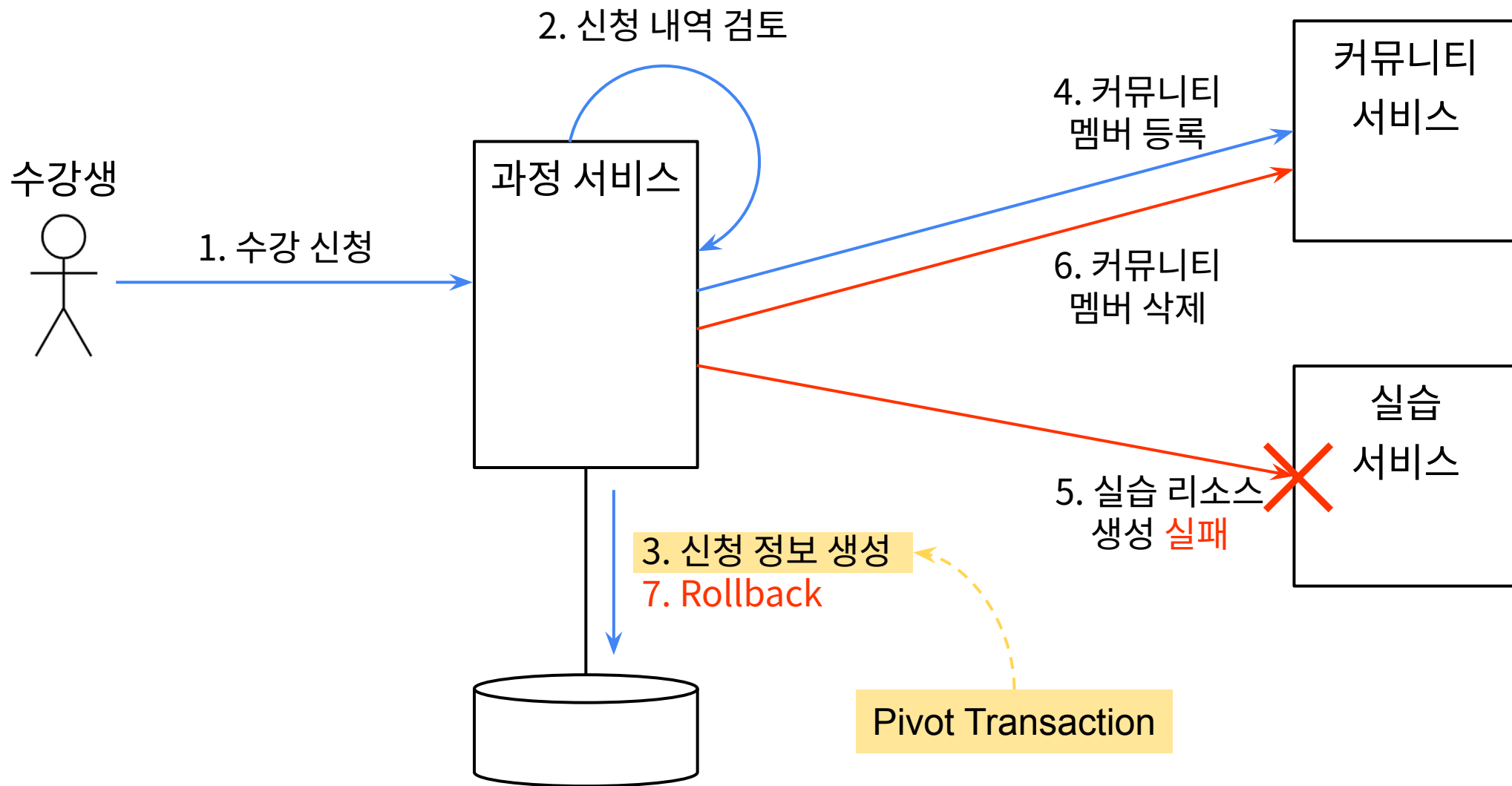


# 예시 - 수강 신청



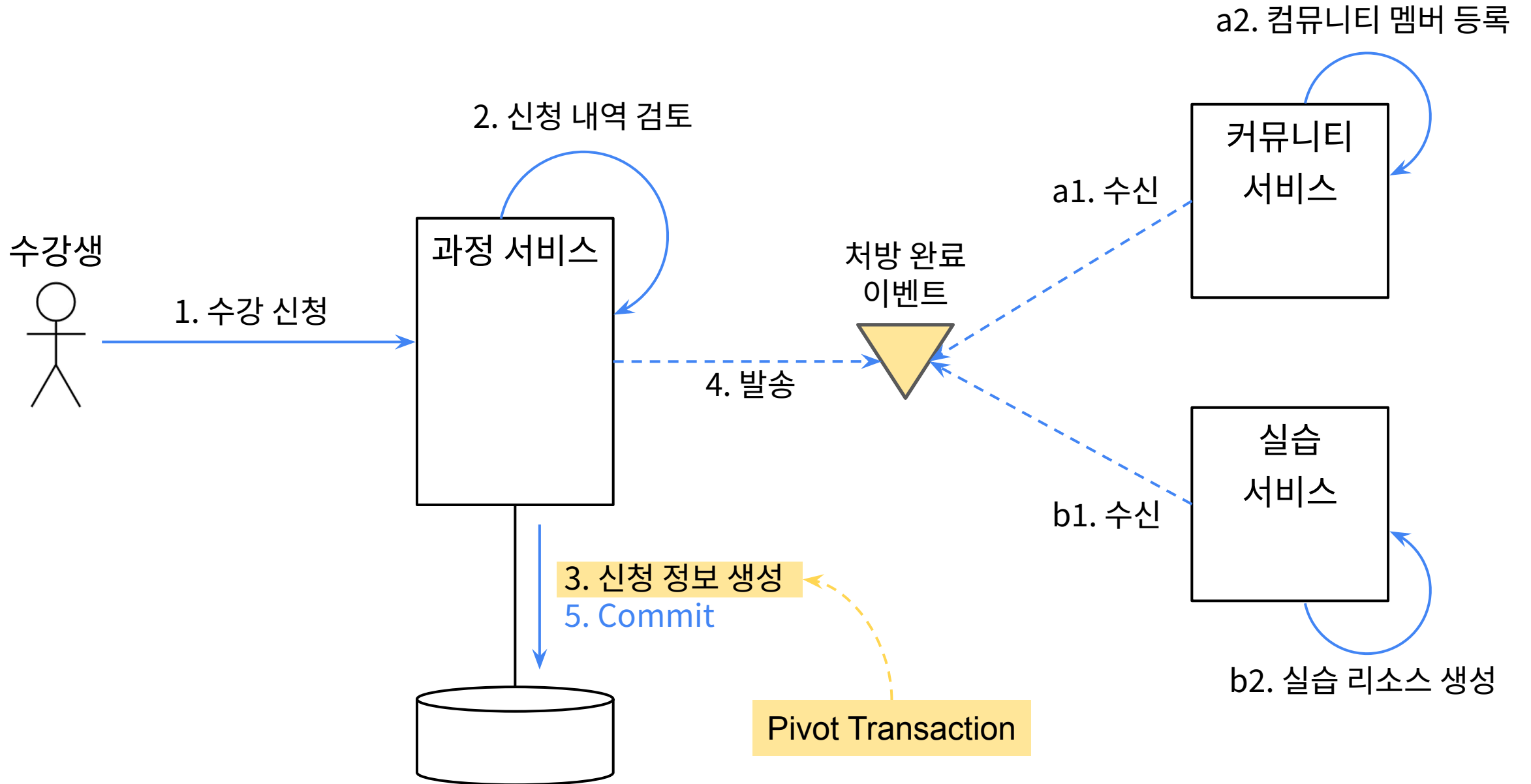
# 수강 신청 실패

실습 리소스 등록이 실패했다고  
수강 신청을 거절하는 게 맞나?

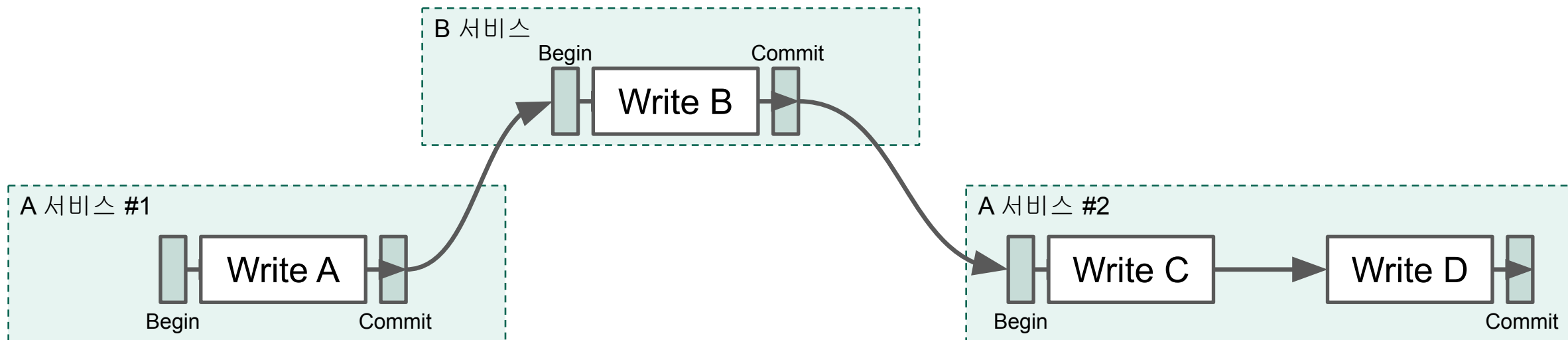
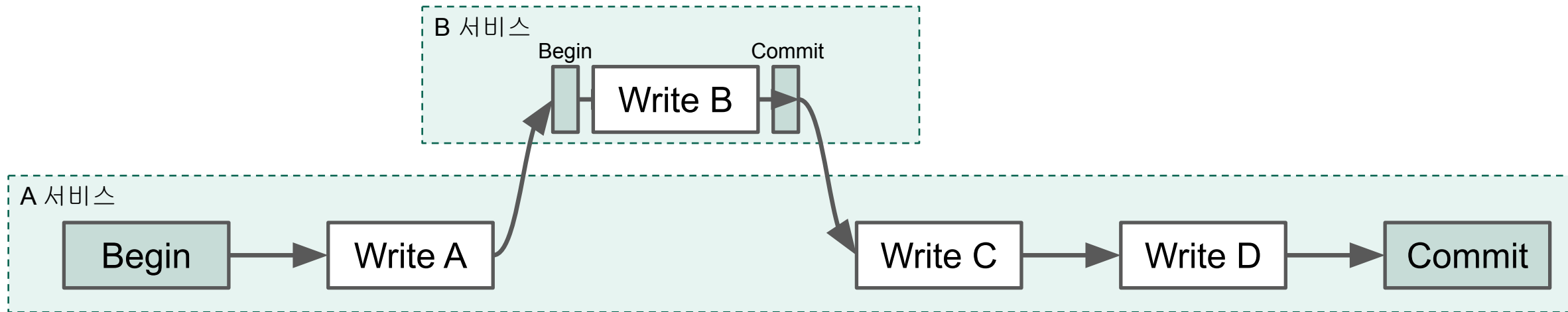




# Pivot Transaction을 기준으로 이벤트로 분리



## (4) 작은 트랜잭션으로 나눈다면?



# 트랜잭션 개발 가이드

1. 데이터 오너십 원칙

2. 실패한 트랜잭션의 처리

- 트랜잭션 취소/재시도

3. 트랜잭션 격리성 보완

- 레코드/트랜잭션 잠금

4. 신뢰할 수 있는 트랜잭션 구현

- API/Event, 멍등성 처리

(1) 발생가능한 이슈

(2) 변경 중인 데이터를 볼 수 있음

(3) 변경 중인 데이터를 덮어 쓸 수 있음

(4) 앞서 조회한 데이터가 변경될 수 있음

# Isolation Level의 변화



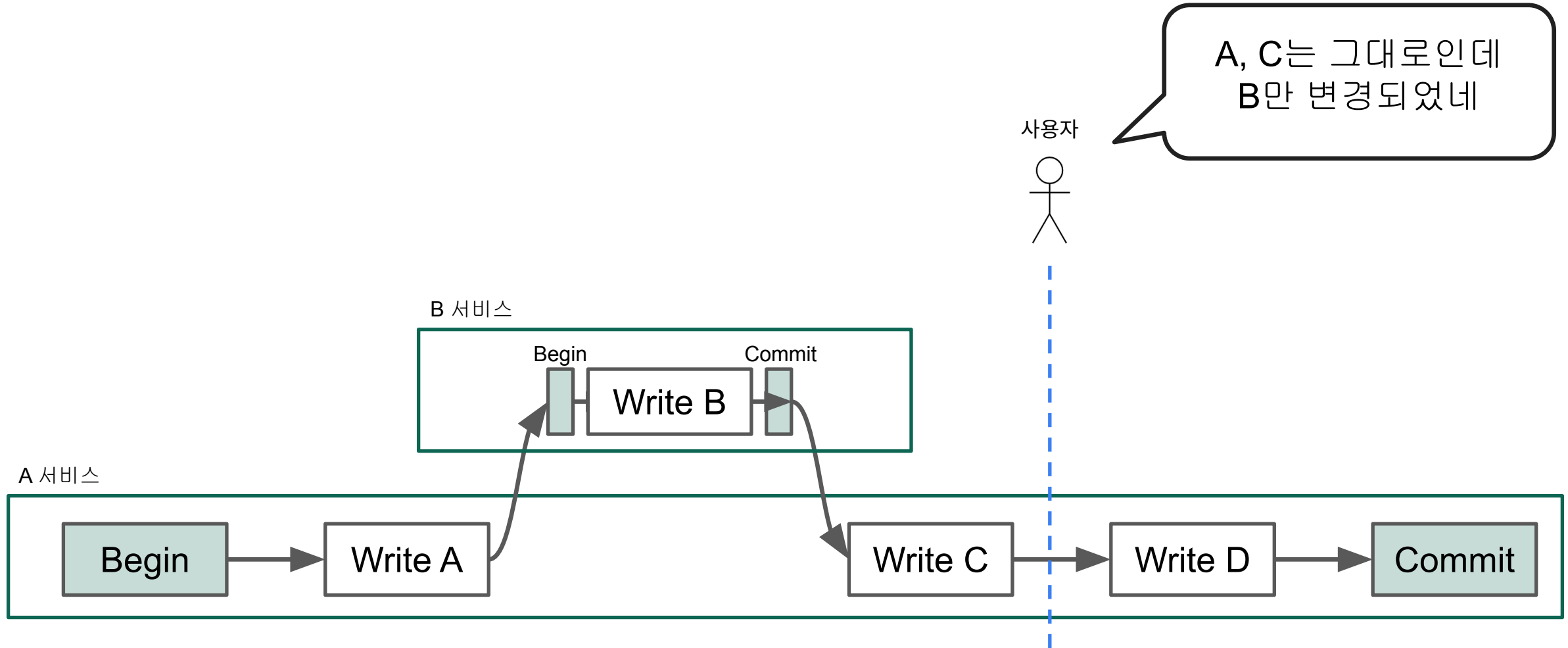
서비스 간의 트랜잭션의 Isolation Level은 Read Uncommitted

	Dirty Read	Non-Repeatable Read	Phantom Read	Write Skew	Default Level
Read Uncommitted	발생 가능	발생 가능	발생 가능	발생 가능	
Read Committed	발생 안함	발생 가능	발생 가능	발생 가능	Oracle, MSSql, Postgre SQL
Repeatable Read	발생 안함	발생 안함	발생 가능	발생 가능	MySQL, Mariadb
Serializable	발생 안함	발생 안함	발생 안함	발생 안함	

# (1) 발생 가능한 이슈와 대처 방법

- ① 변경 중인 데이터를 다른 트랜잭션이 볼 수 있음
- ② 변경 중인 데이터를 후행 트랜잭션이 덮어 쓸 수 있음
- ③ 조회한 값을 다른 트랜잭션이 변경할 수 있음

## (2) 변경 중인 데이터를 다른 트랜잭션이 볼 수 있음



# Eventual Consistency

### (3) 변경 중인 데이터를 다른 트랜잭션이 덮어 쓸 수

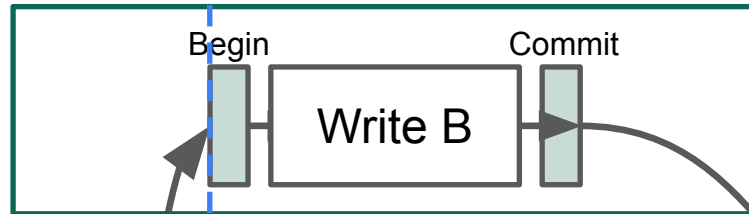
TX #2

B 서비스

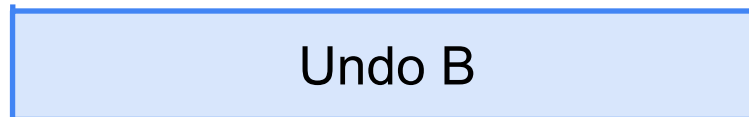


TX #1

B 서비스



A 서비스





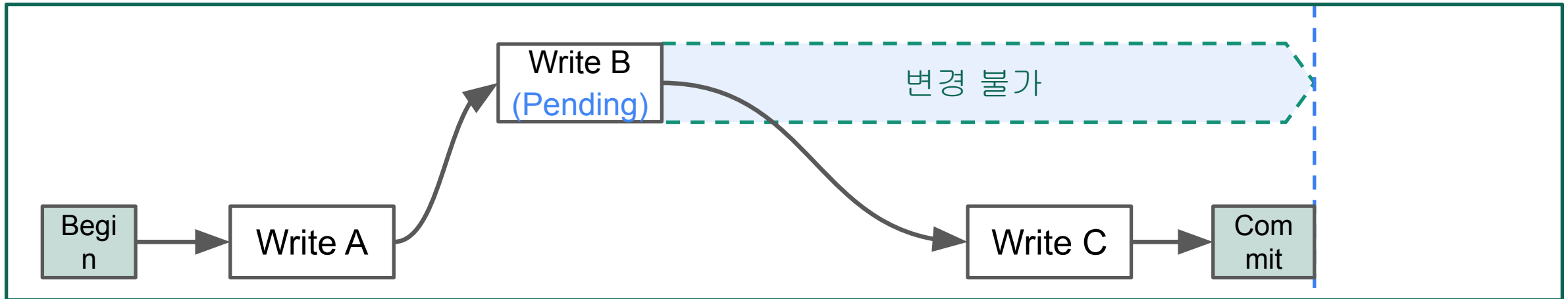
# DB 내부의 트랜잭션이라면

TX #2

B 서비스



TX #1



Semantic Lock

TCC (Try, Confirm, Cancel)

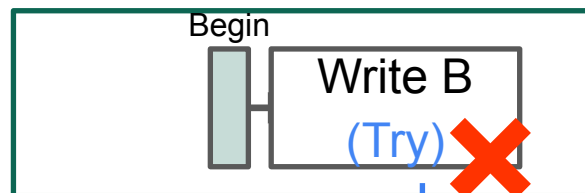
Offline Lock

# 중간 상태를 설정해서 다른 트랜잭션의 쓰기를 차단

※ TCC Pattern  
(Try, Confirm, Cancel)

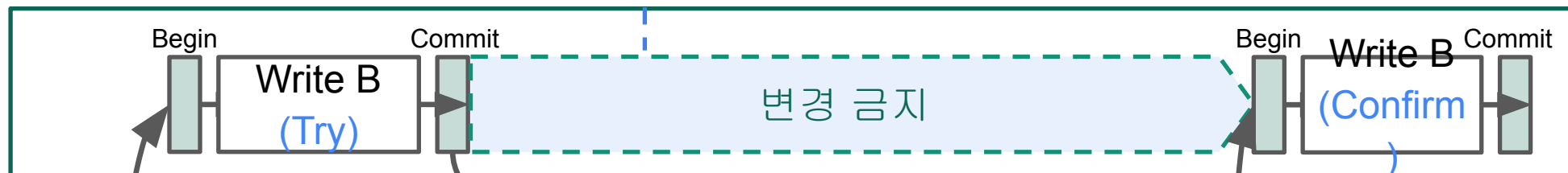
TX #2

B 서비스



TX #1

B 서비스



A 서비스



# (4) 앞서 조회한 데이터를 다른 트랜잭션이 변경할 수

TX #2

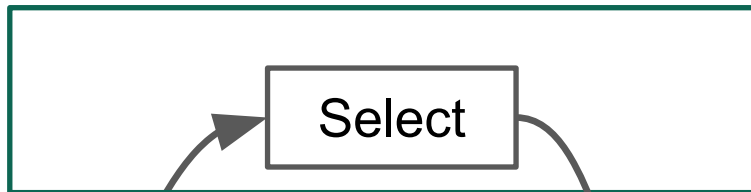
B 서비스



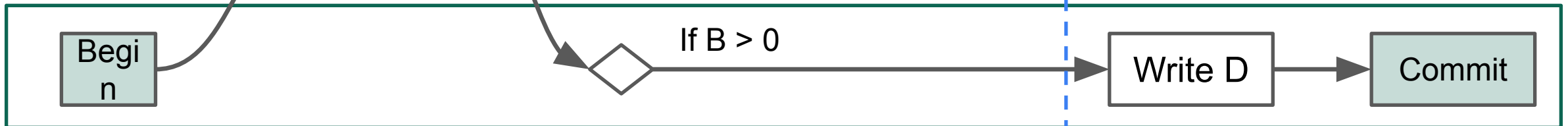
B = 0

TX #1

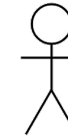
B 서비스



A 서비스



사용자



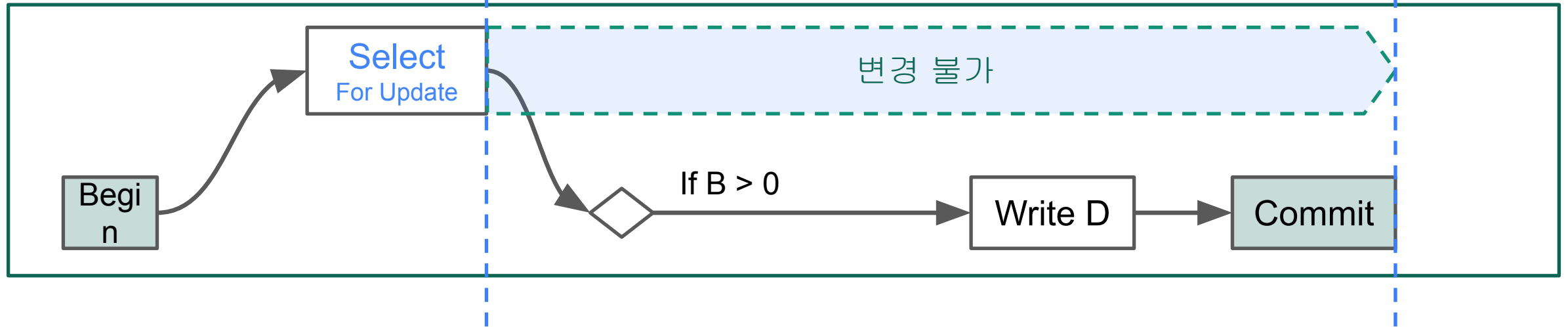
변경하면 안되는데  
변경해버렸네;;

# DB 내부의 트랜잭션이라면

TX #2

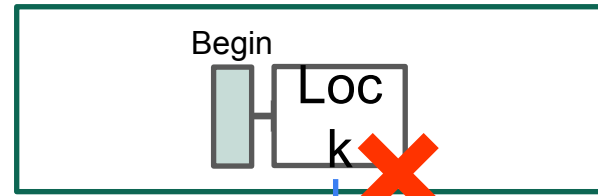


TX #1



TX #2

B 서비스



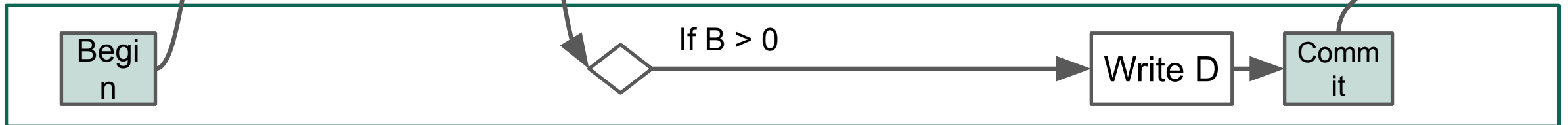
TX #1

B 서비스

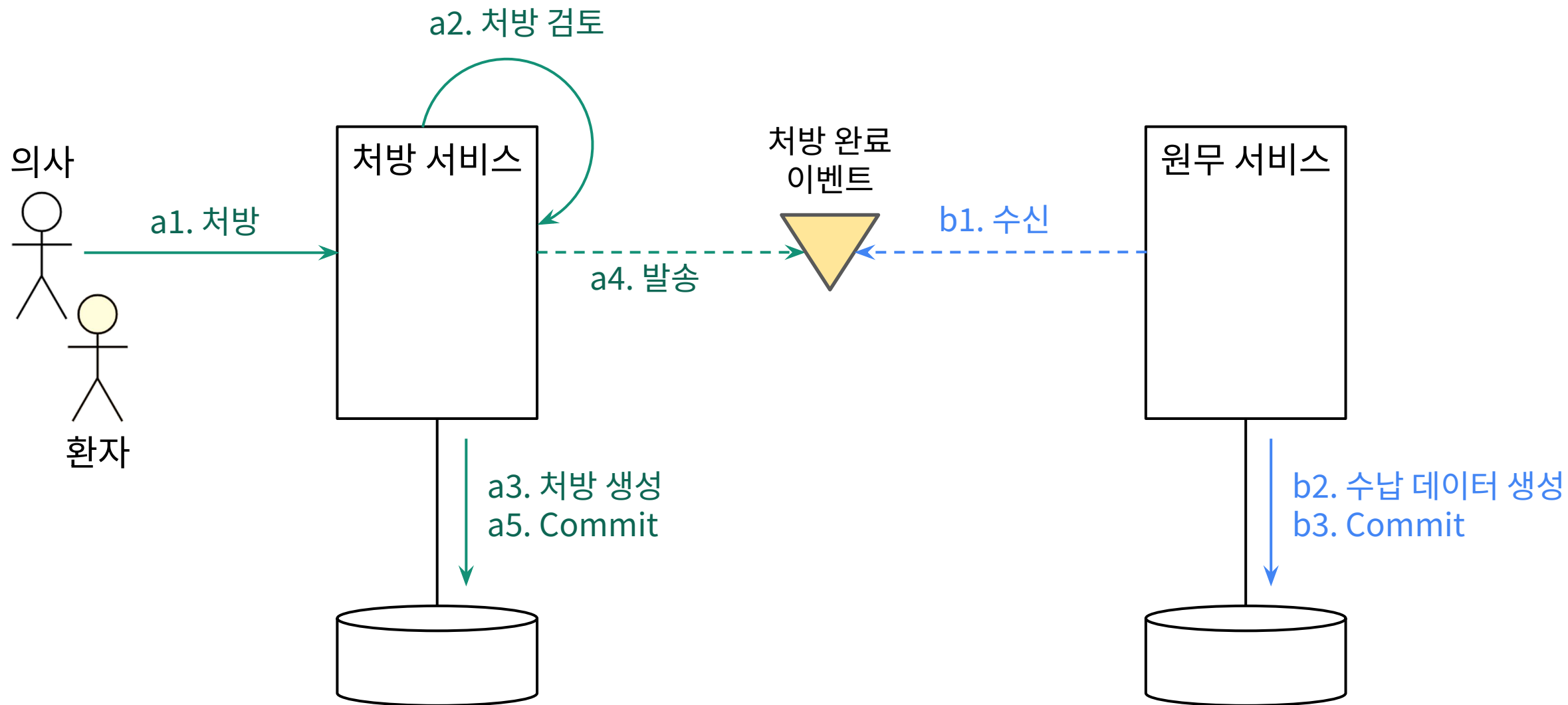
B 서비스



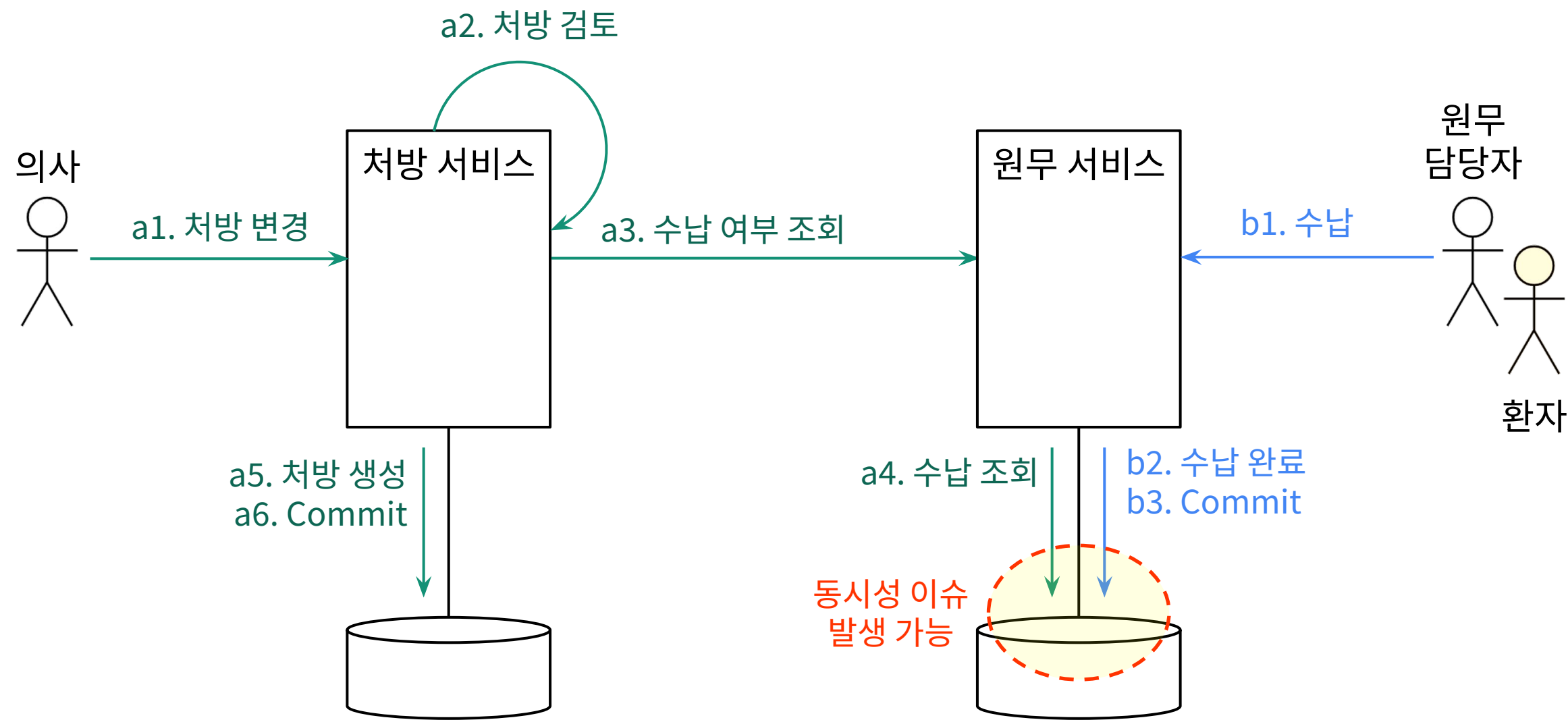
A 서비스



### (3) 예시 - 외래진료 처방

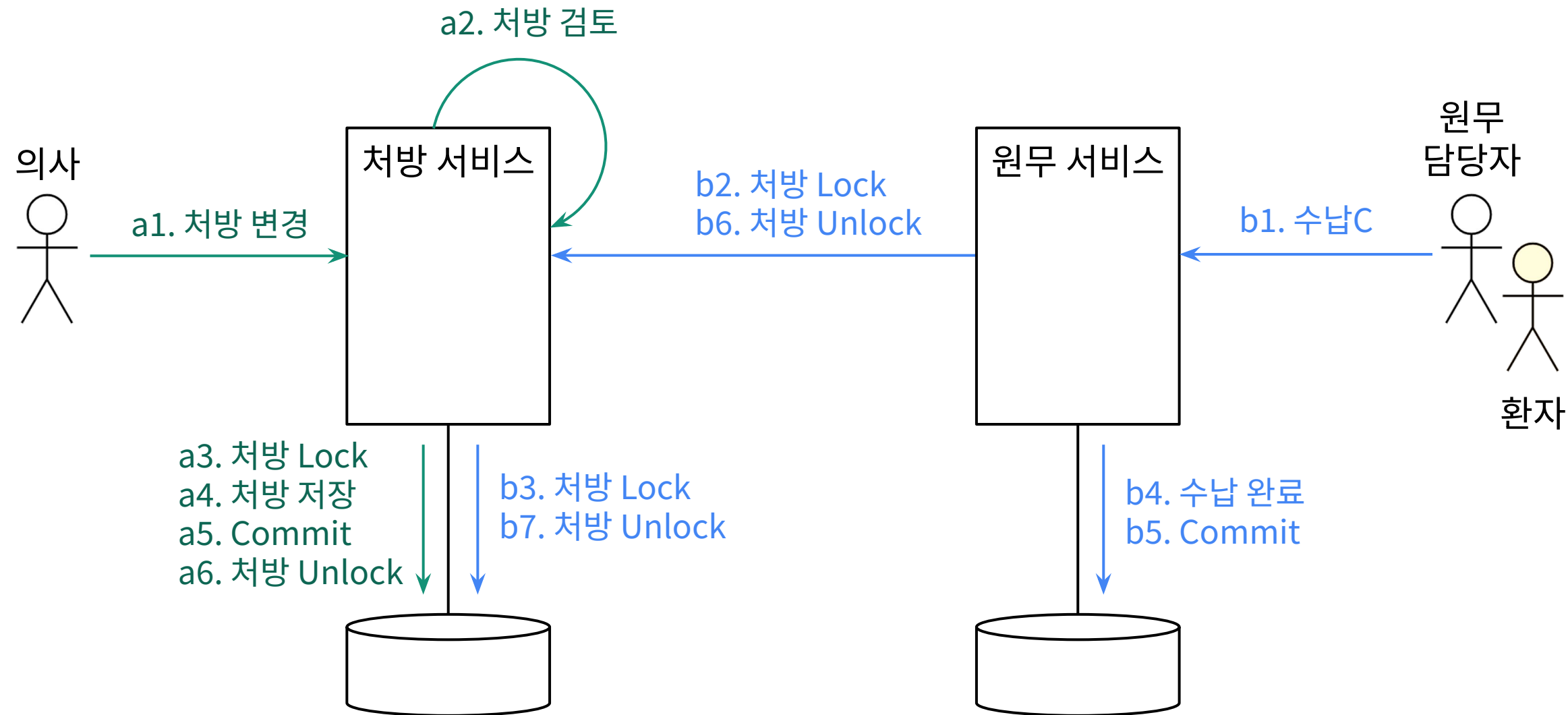


# 외래진료 - (a)처방 변경, (b)수납





# 외래진료 - (a)처방 변경, (b)수납 (+ Lock)



# 트랜잭션 개발 가이드

1. 데이터 오너십 원칙

2. 실패한 트랜잭션의 처리

- 트랜잭션 취소/재시도

3. 트랜잭션 격리성 보완

- 레코드/트랜잭션 잠금

4. 실회할 수 있는 트랜잭션 구현

- API/Event, 멍등성 처리

(1) 두 저장소 쓰기의 어려움

(2) 진짜 실패 vs 가짜 실패

(3) 실패를 복구하는 방법

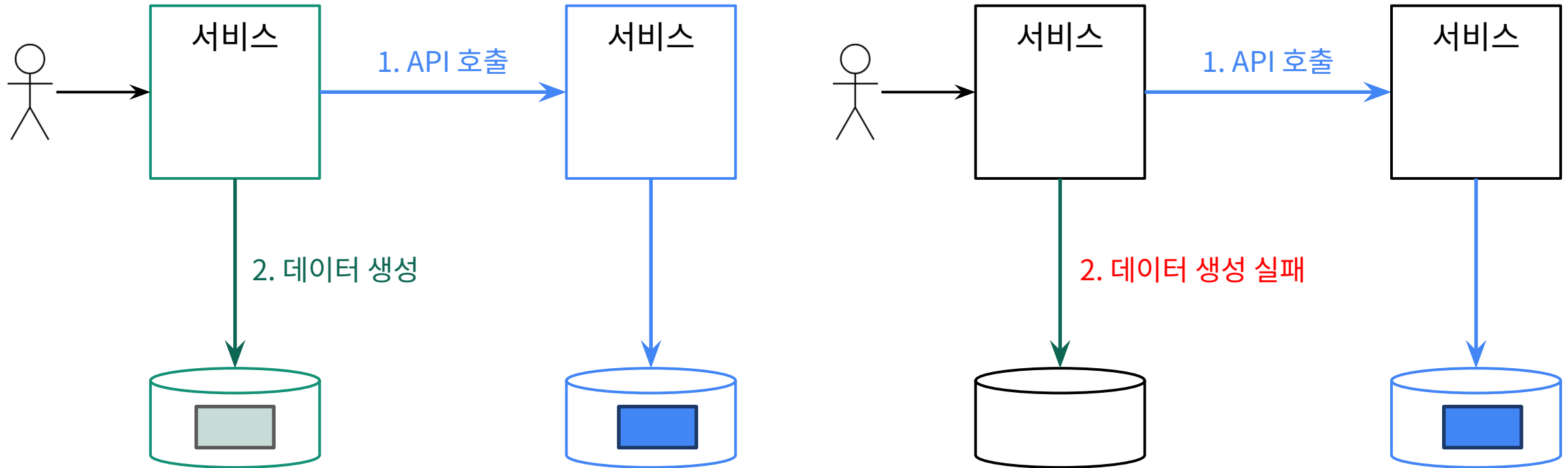
(4) 멍등성

# (1) 두 저장소에 쓰기의 어려움

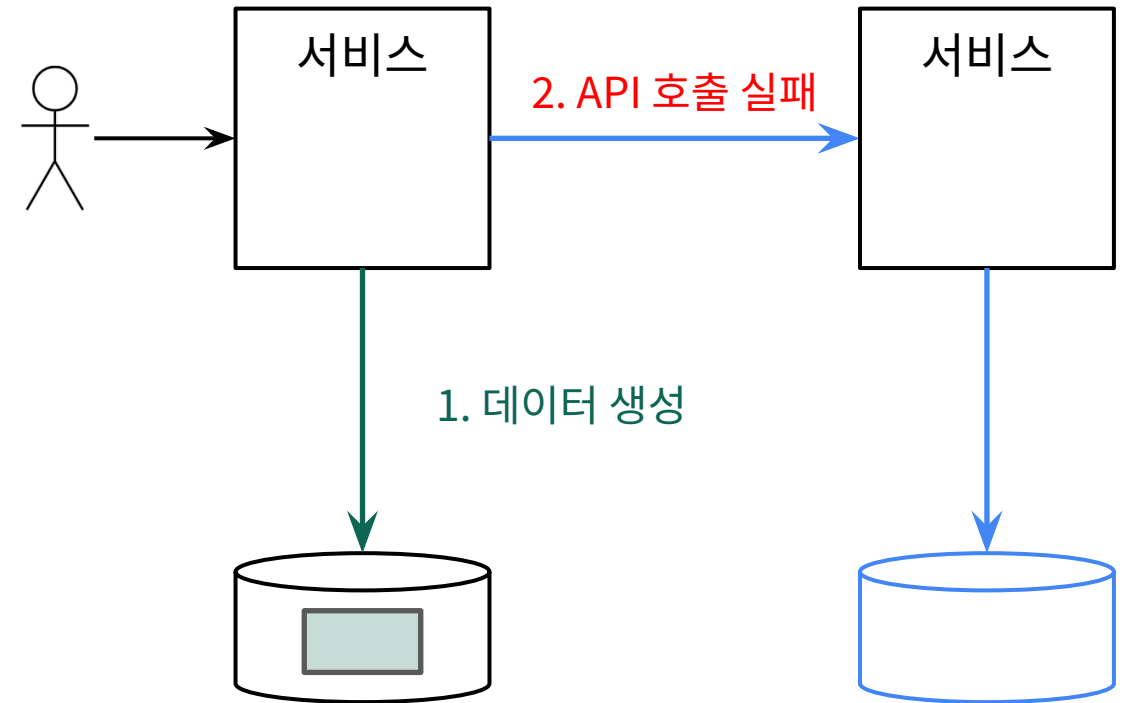
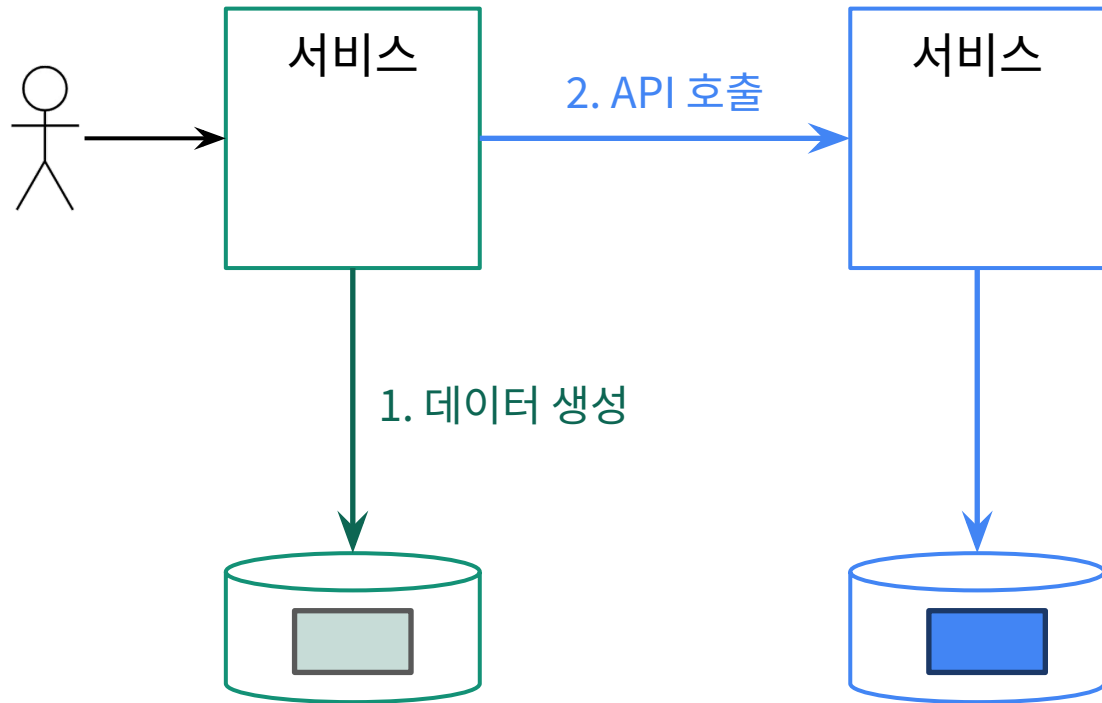


특정 사건을 데이터로 저장하고 API/이벤트를 호출하는 것은 의외로 까다로움

순서1: 이벤트 전송 후 데이터 생성

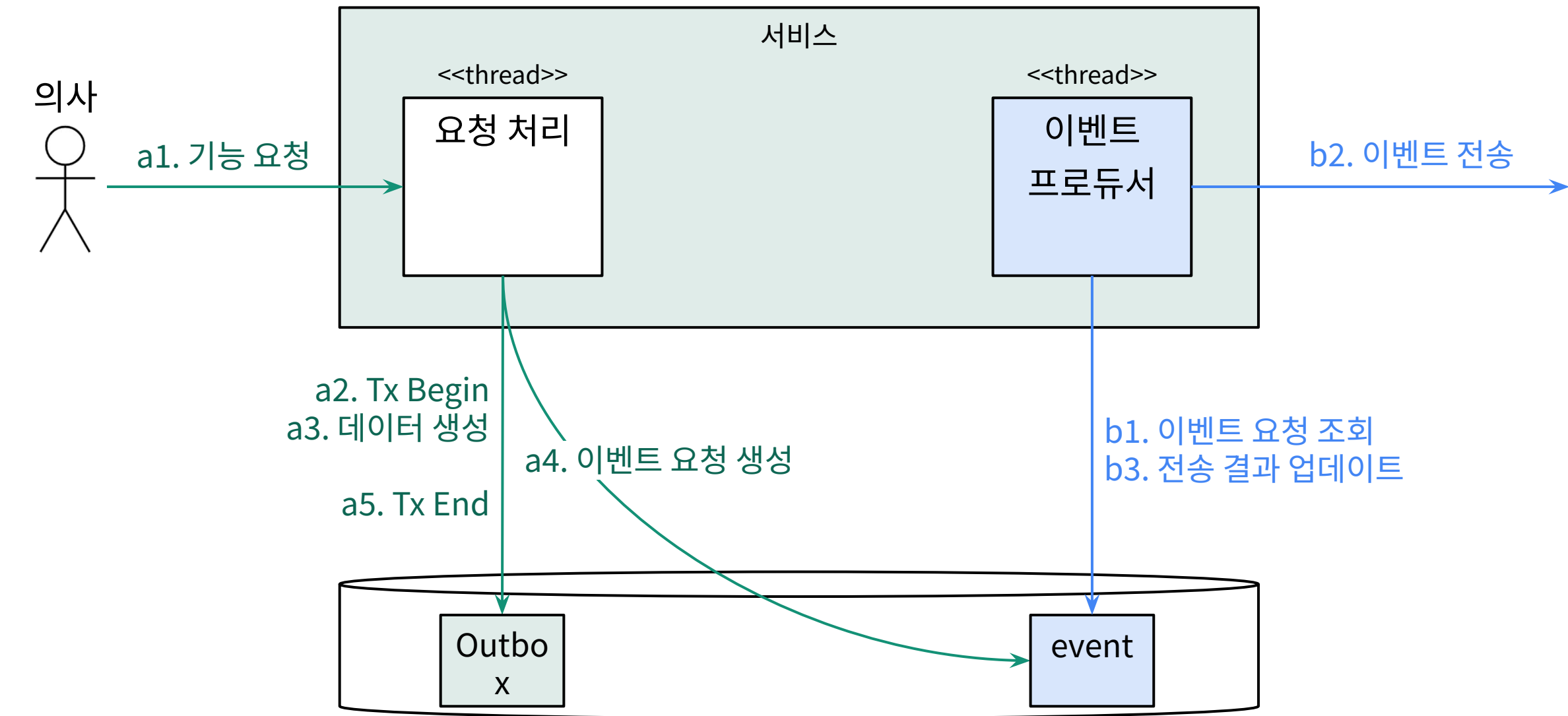


## 순서2: 데이터 생성 후 이벤트 전송



# 실패할 수 있는 이벤트 전송

※ Transactional Outbox Pattern



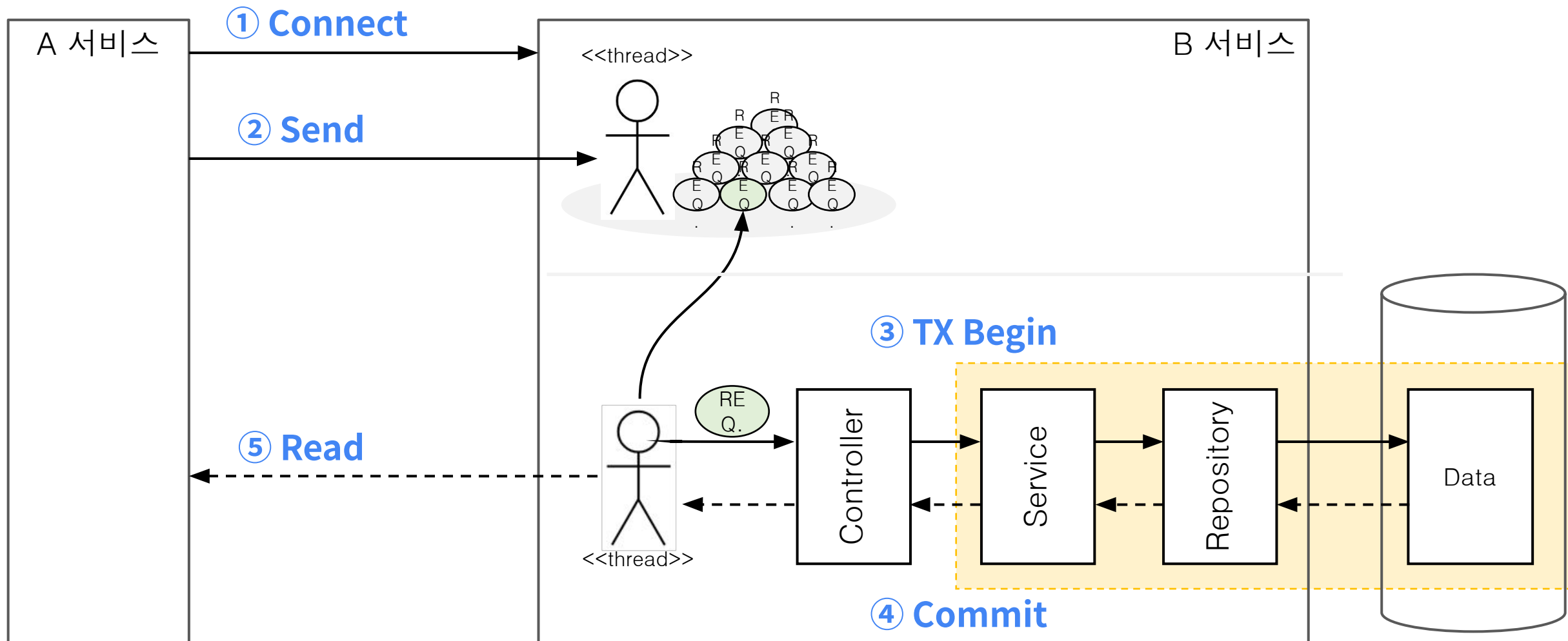
# 실패했는데 성공했다고?



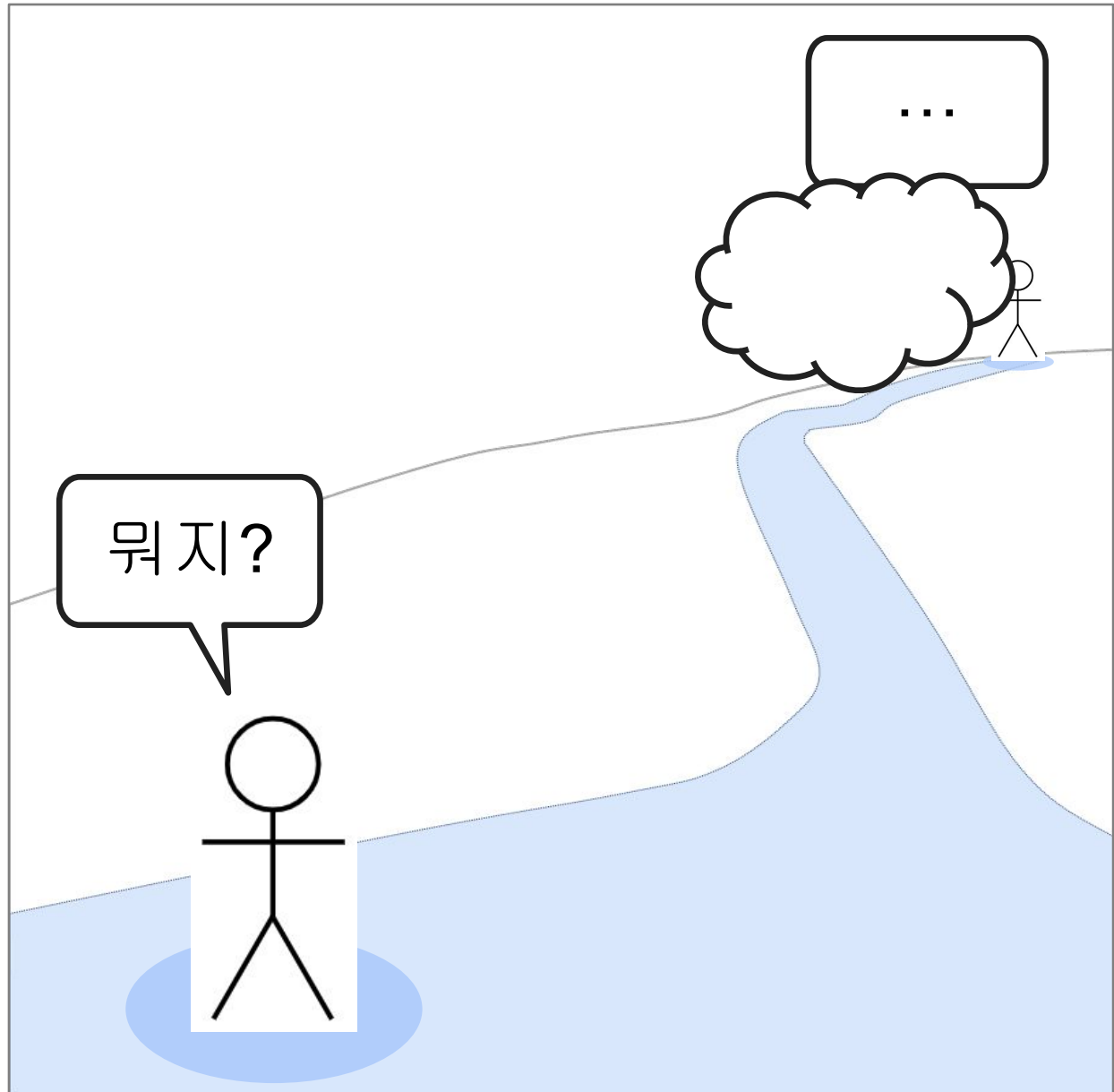
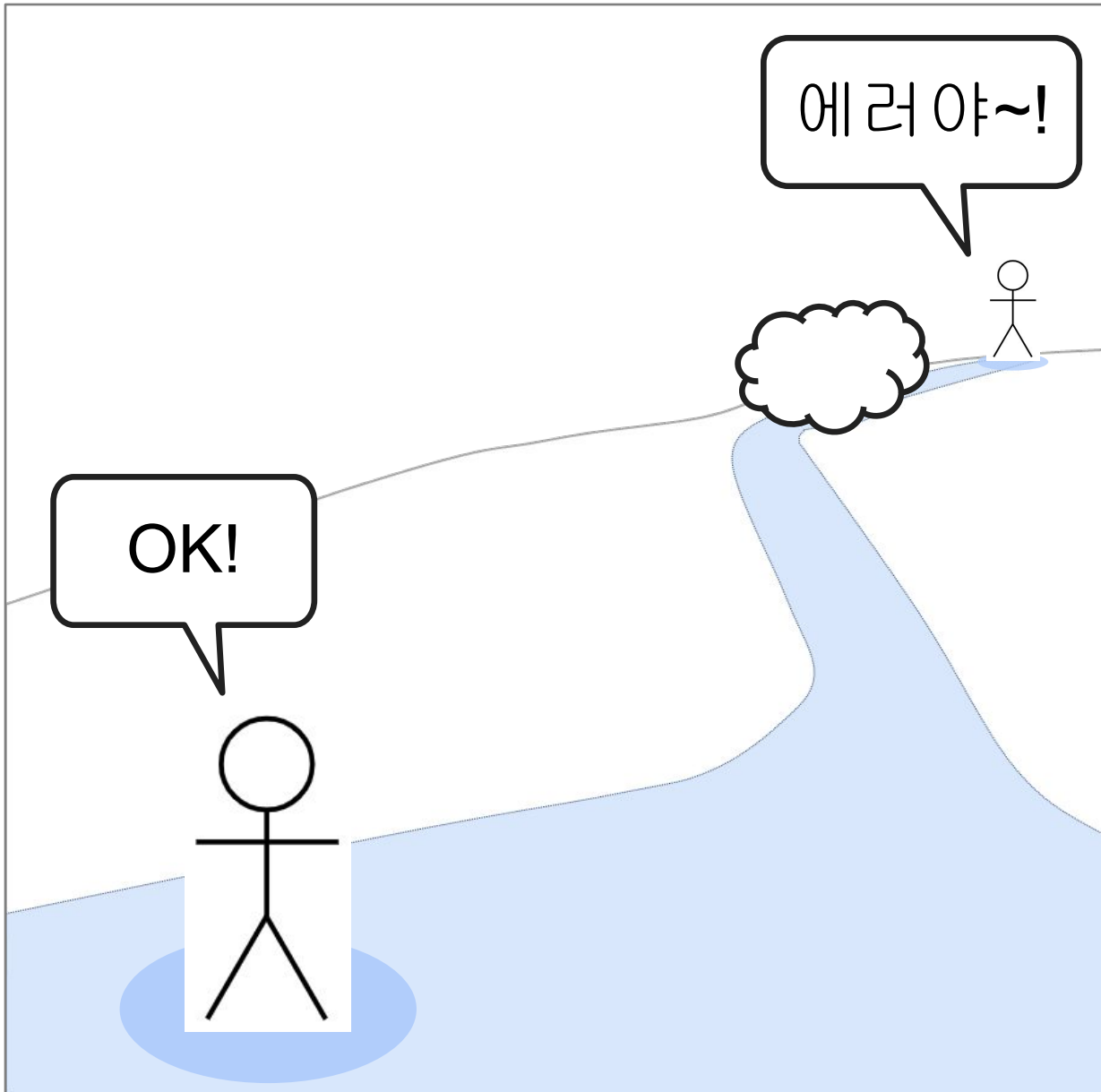
실패했다고 다시 하면  
두 번 하는 것일 수도...

두 번 실행해도  
동일한 결과가 나와야 함

# 불안정한 네트워크 통신



# (1) 진짜 실패 vs 가짜 실패

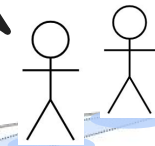




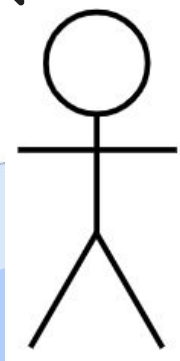
# QUIZ: 진짜 실패를 나타내는 예외는?

- ① `java.net.ConnectException`
- ② `java.net.SocketTimeoutException`
- ③ `org.springframework.web.client.HttpClientErrorException`
- ④ `org.springframework.web.client.ResourceAccessException`
- ⑤ `java.net.SocketException`

## (2) 가짜 실패를 복구하는 방법



뭐였어?  
같이 맞춰보자

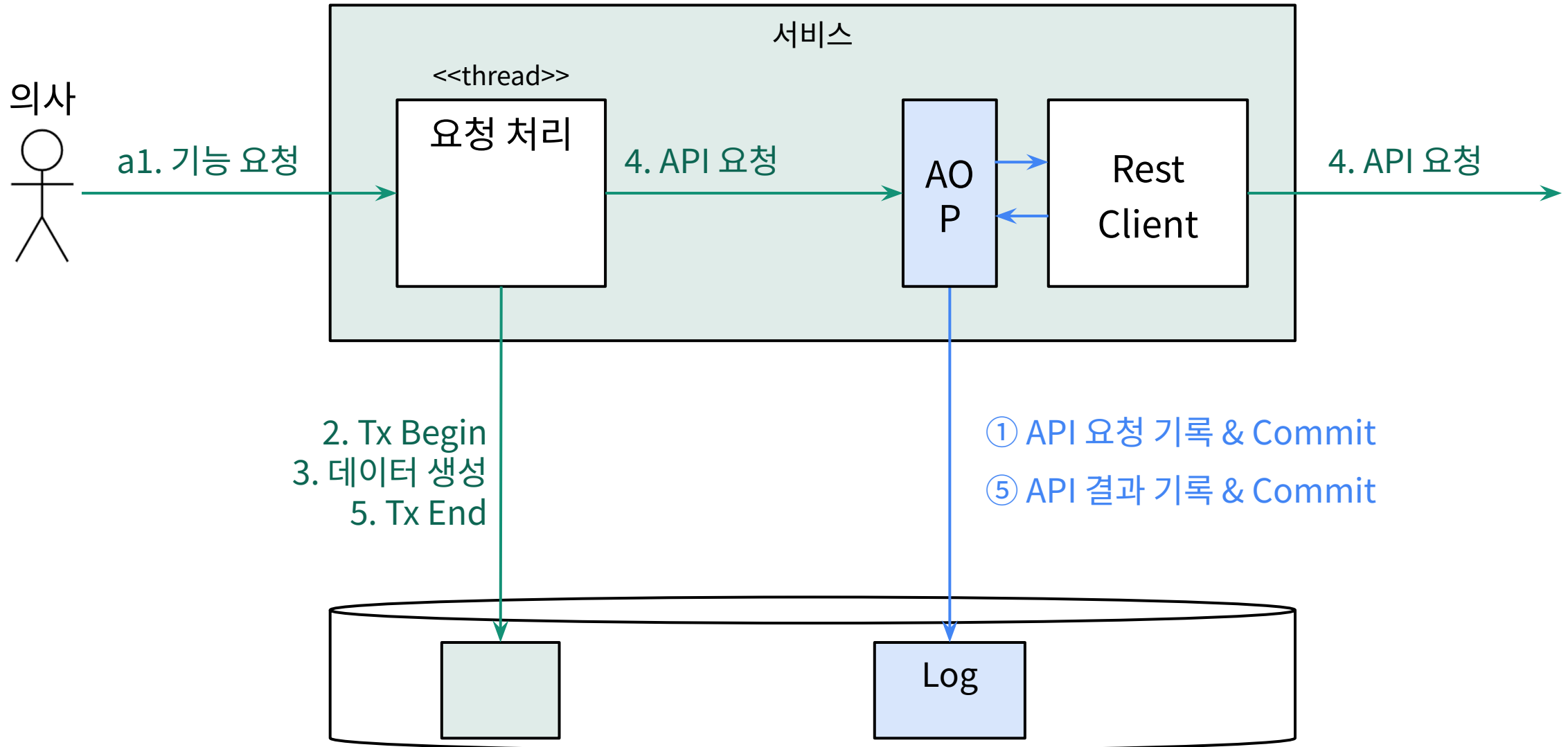


괜찮음  
다시 해보자고!

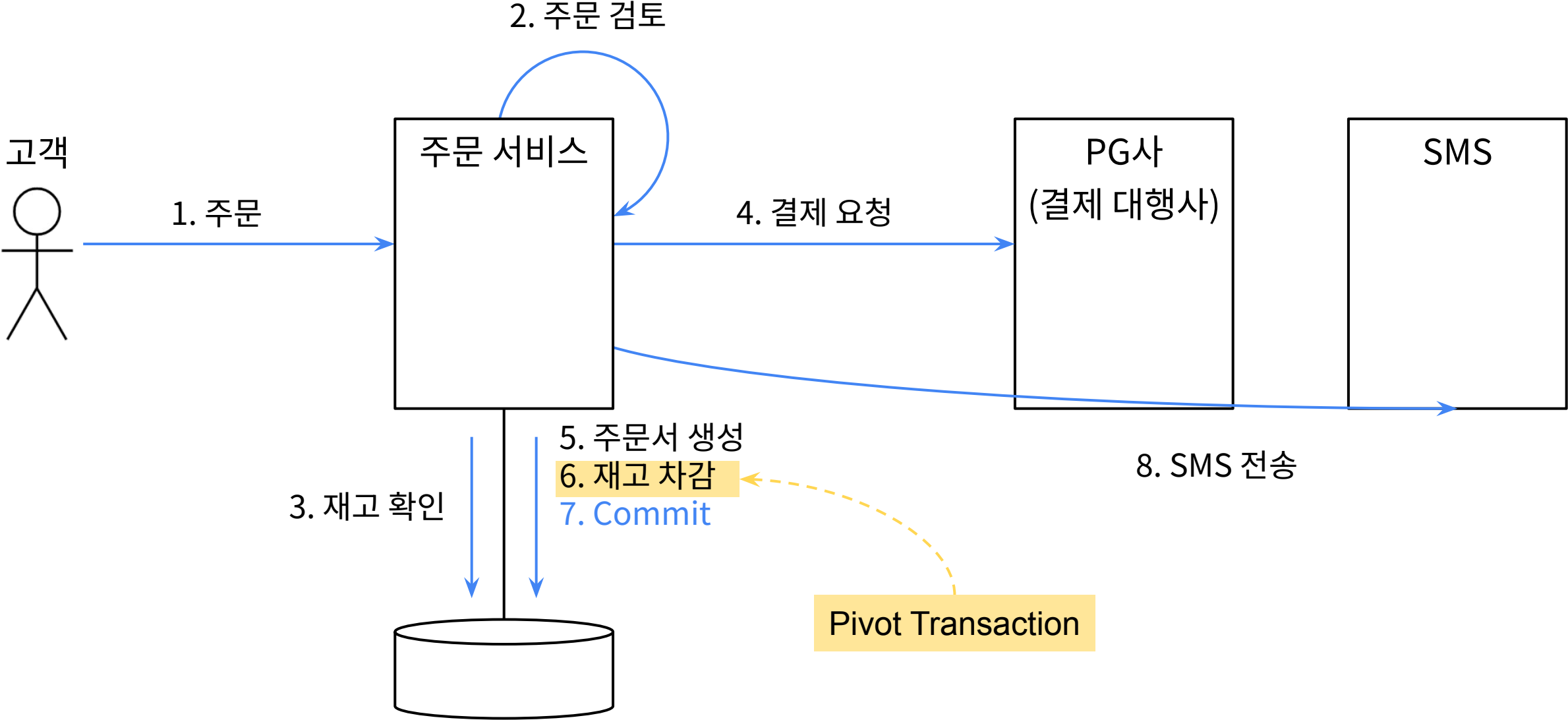


OK~!

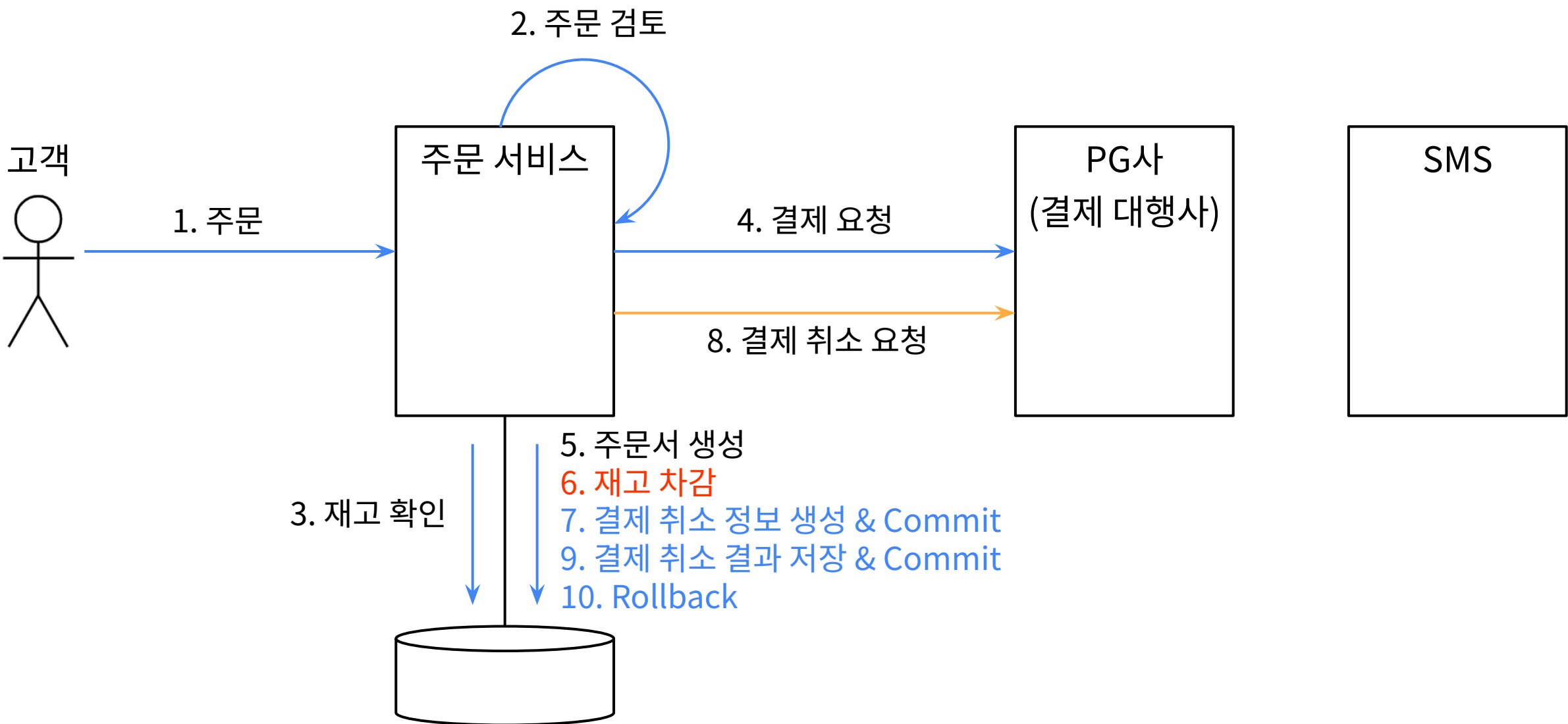
# 실패 정보의 유실 방지



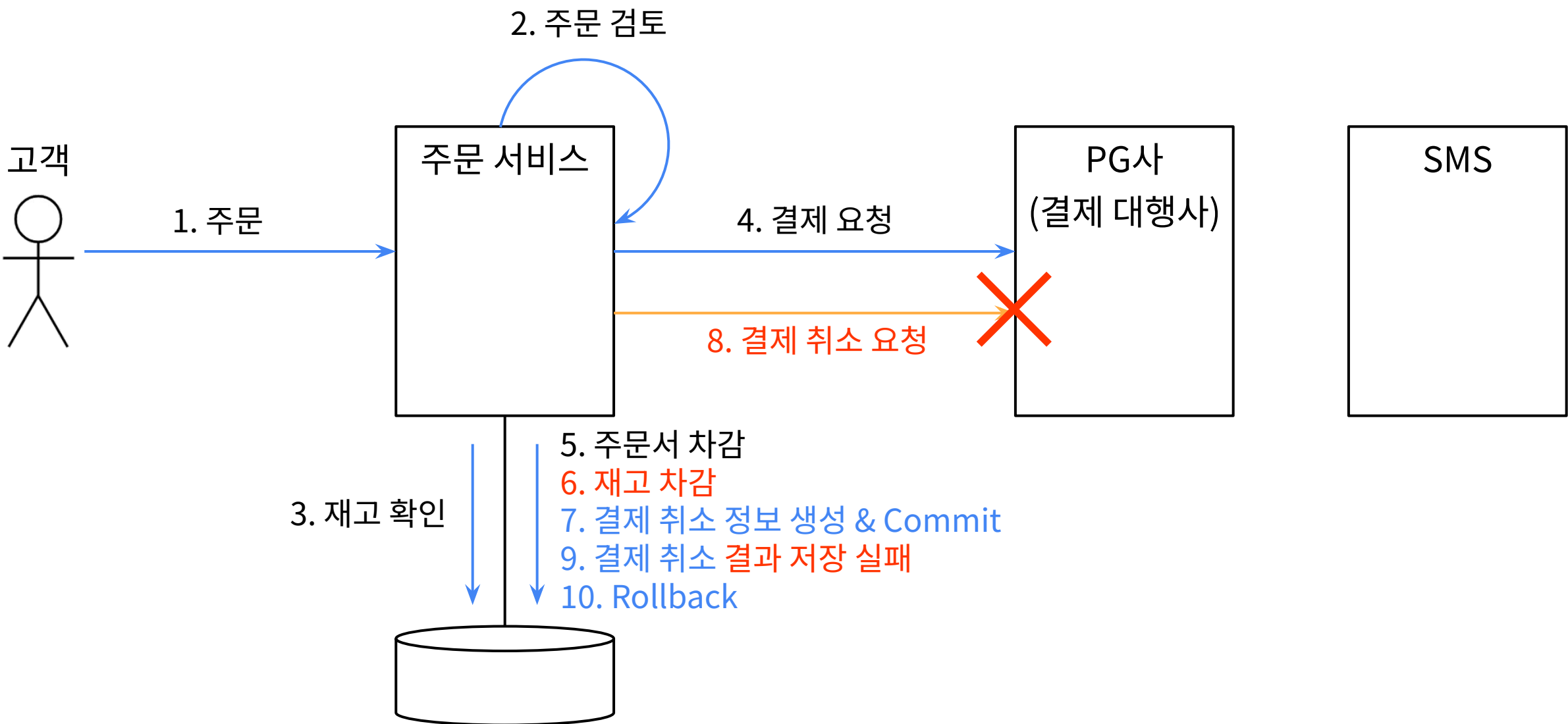
# 예시 - 주문 결제 예시



# 주문 결제 - 6번 재고 차감 중 오류



# 주문 결제 - 6번 오류로 8번 결제 취소 중 오류



## (4) 역등성

역등성 있게 구현하면 실패해도 더 편리하게 조치 가능

### 1. 중복 실행

- 유니크 값으로 식별 (없으면 채번)
- 중복 체크: 사전 체크 or 사후 에러 무시

### 2. 순서 꼬임 처리

- 같은 데이터를 여러 번 수정하는 경우
- 순서가 꼬여도 문제가 없도록 구현

ID  
이름  
휴대폰  
사무실

원본	
#001	
홍길동	
1111-1111	
1111-1111	



Updated #1	
#001	
2222-2222	



Updated #2	
#001	
3333-3333	
3333-3333	



최종	
#001	
홍길동	
3333-3333	
3333-3333	



ID  
이름  
휴대폰  
사무실

원본	
#001	
홍길동	
1111-1111	
1111-1111	



Updated #2	
#001	
3333-3333	
3333-3333	



Updated #1	
#001	
2222-2222	



최종	
#001	
홍길동	
2222-2222	
3333-3333	

같은 필드 목록 전송

ID  
이름  
휴대폰  
사무실  
버전

원본	
#001	
홍길동	
1111-1111	
1111-1111	
v0	



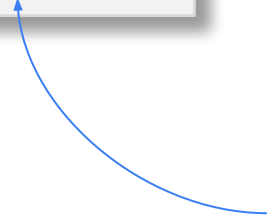
Updated #1	
#001	
홍길동	
2222-2222	
1111-1111	
v1	



Updated #2	
#001	
홍길동	
3333-3333	
3333-3333	
v2	



최종	
#001	
홍길동	
3333-3333	
3333-3333	
v2	



버전 정보를 함께 전송

ID  
이름  
휴대폰  
사무실  
버전

원본	
#001	
홍길동	
1111-1111	
1111-1111	
v0	



Updated #2	
#001	
홍길동	
3333-3333	
3333-3333	
v2	

(O)  $V2 > V0$



Updated #1	
#001	
홍길동	
2222-2222	
1111-1111	
v1	

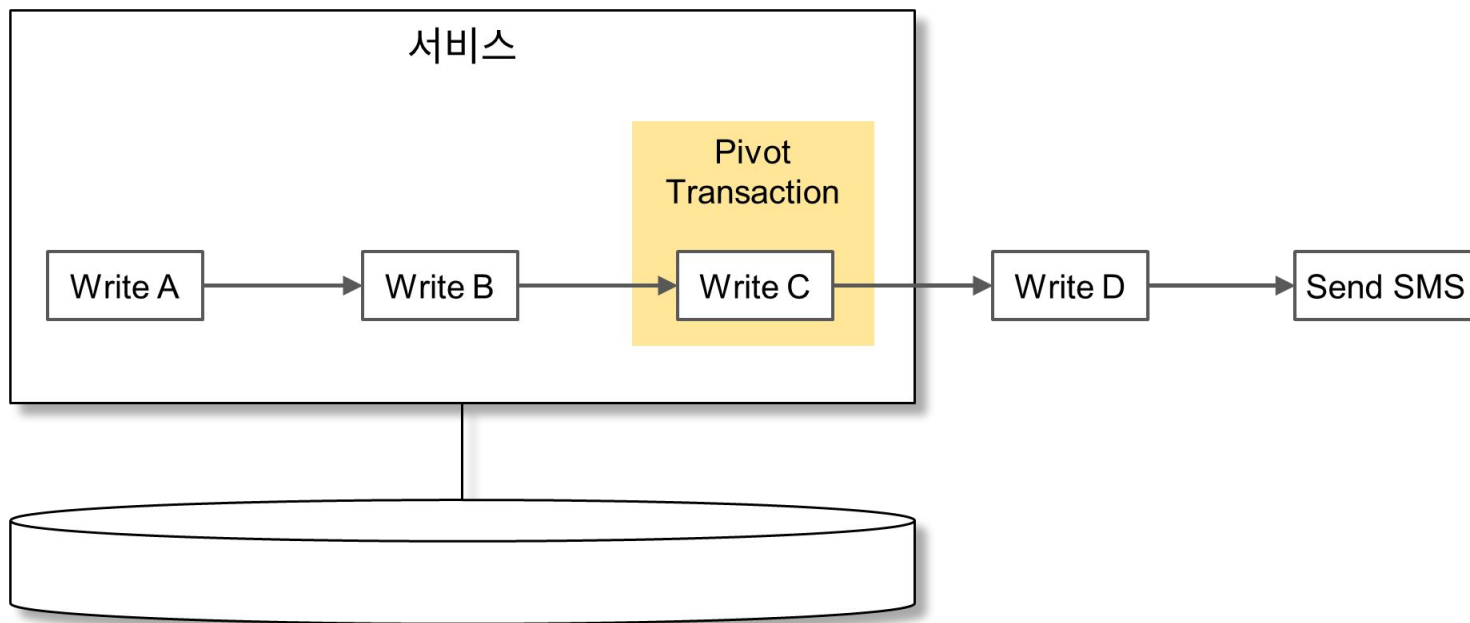
(X)  $V1 < V2$



최종	
#001	
홍길동	
3333-3333	
3333-3333	
v2	

# 요약

1. 피봇 트랜잭션 이후는 이벤트로 처리
2. 격리성을 보완할 때는 어플리케이션 레벨 락 설정
3. 왜 실패했는지 모르면 대사 작업 필요





**SpringCamp 2025**

**E.O.D**