

10장. 트랜잭션 처리

- 회복과 병행 제어

트랜잭션

❖ 트랜잭션(transaction)의 개념

- 데이터베이스에서 하나의 기능을 수행하기 위해 필요한 최소한의 논리적 작업 단위
 - 한꺼번에 수행되어야 할 SQL 문들의 집합
 - 회복 작업과 병행 제어 처리를 위한 기본 단위로 사용됨

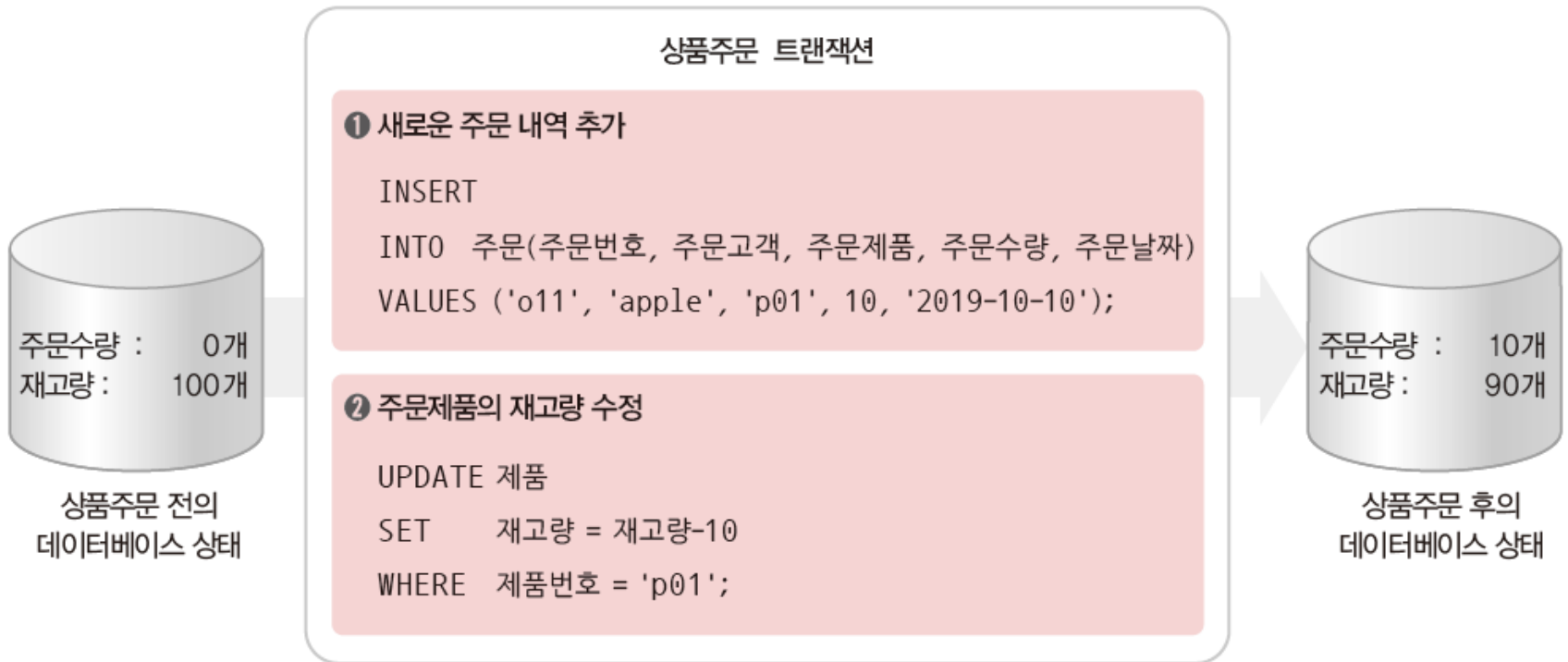
트랜잭션

❖ 트랜잭션의 예 (1)



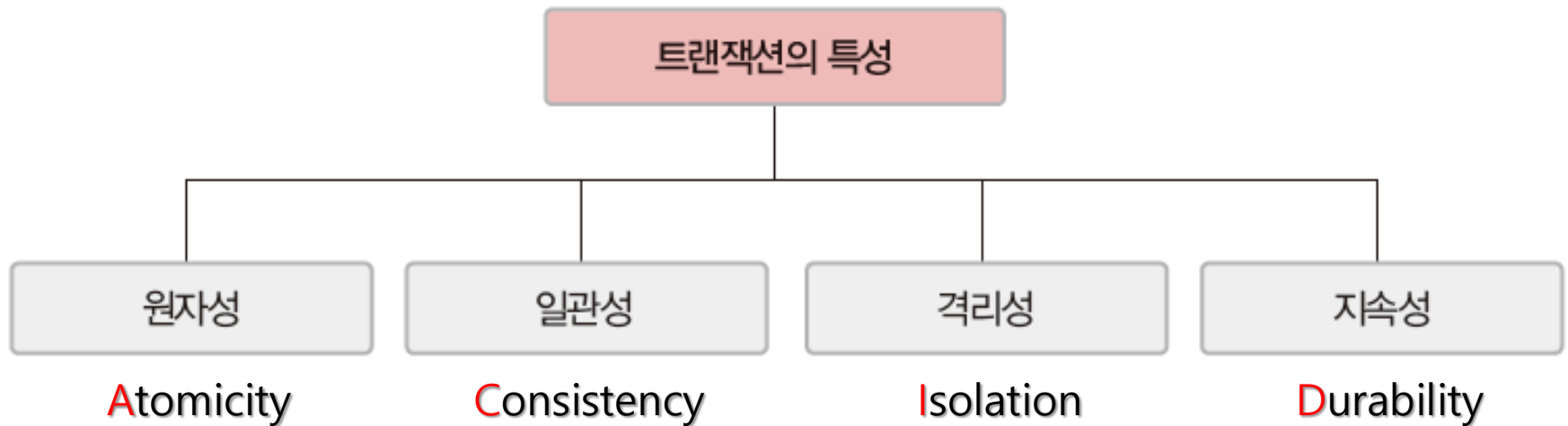
트랜잭션

❖ 트랜잭션의 예 (2)



트랜잭션

❖ 트랜잭션의 특성 (ACID)



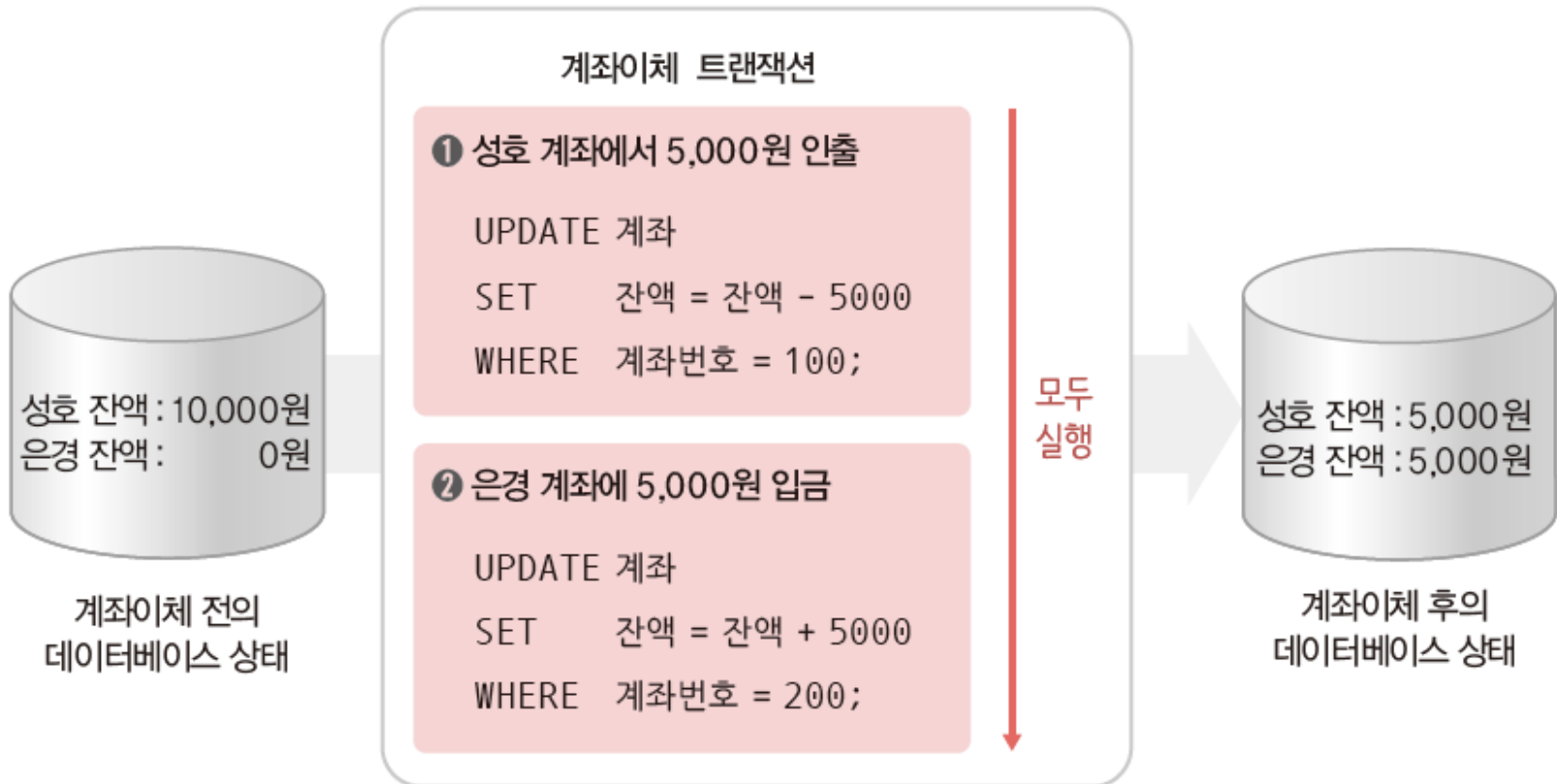
트랜잭션

❖ 트랜잭션의 특성 – 원자성(Atomicity)

- All-or-Nothing
 - 트랜잭션에 속한 모든 연산들이 모두 실행 완료되거나 하나도 실행되지 않아야 함
 - 연산들이 부분적으로 실행되는 것은 허용되지 않음
- 트랜잭션 수행 도중 장애가 발생하는 경우
 - 장애 발생 전까지 실행된 연산 결과를 모두 취소해야 함
 - 데이터베이스를 트랜잭션 작업 전 상태로 되돌려야 함

트랜잭션

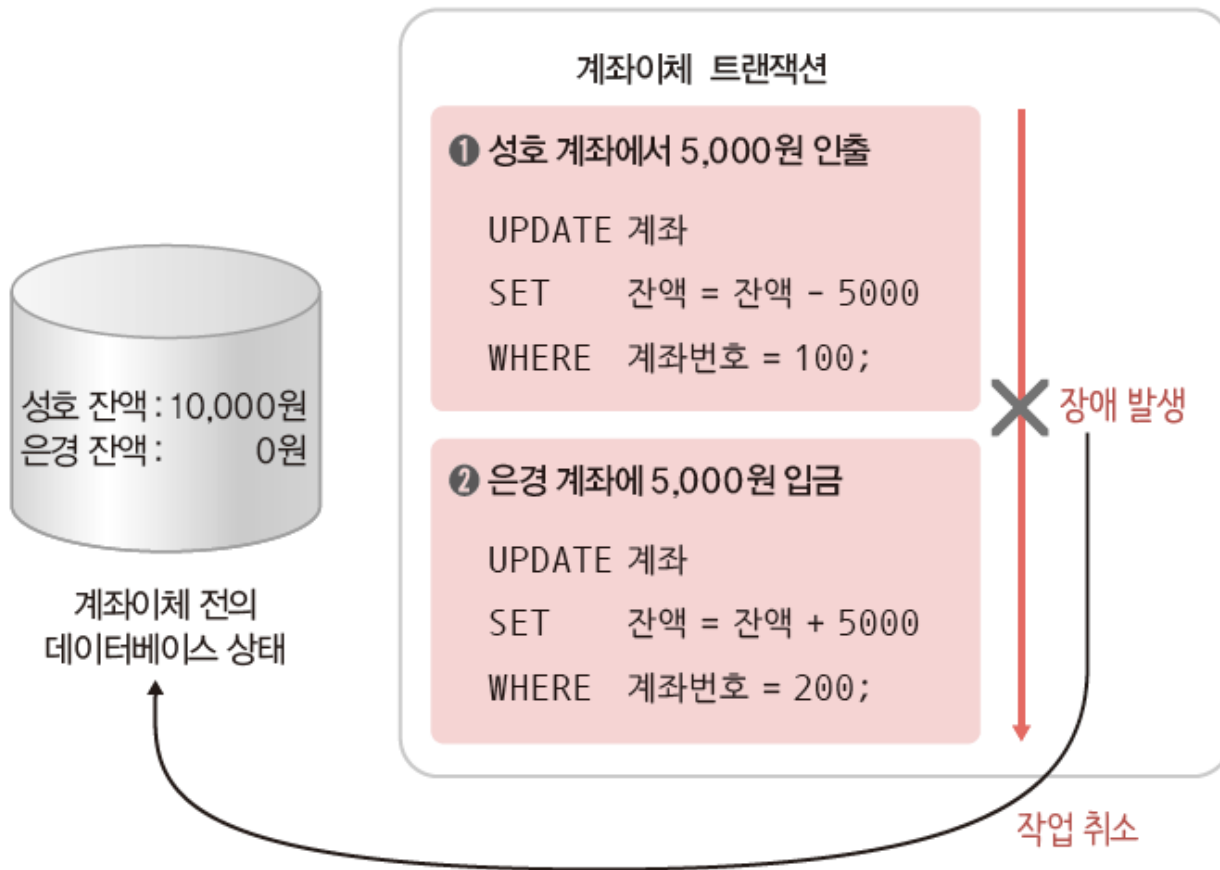
❖ 트랜잭션의 특성 – 원자성 (Atomicity)



< 트랜잭션의 수행 성공 예시 >

트랜잭션

❖ 트랜잭션의 특성 – 원자성 (Atomicity)

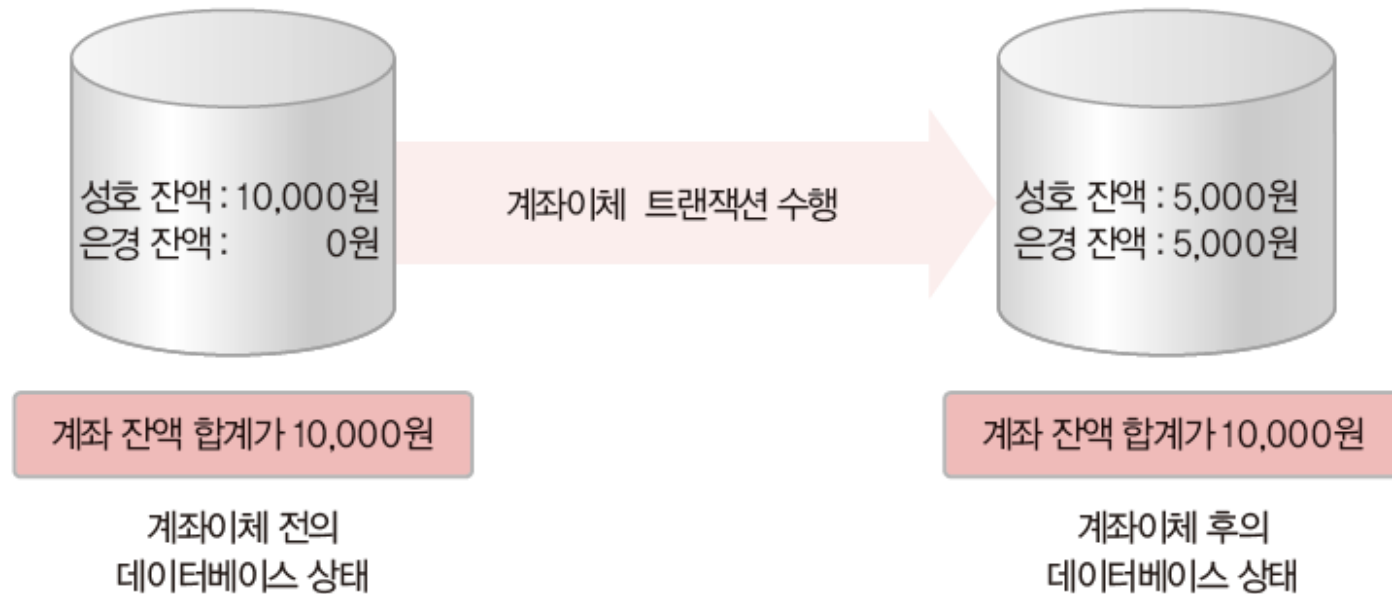


< 트랜잭션의 수행 실패 예시 >

트랜잭션

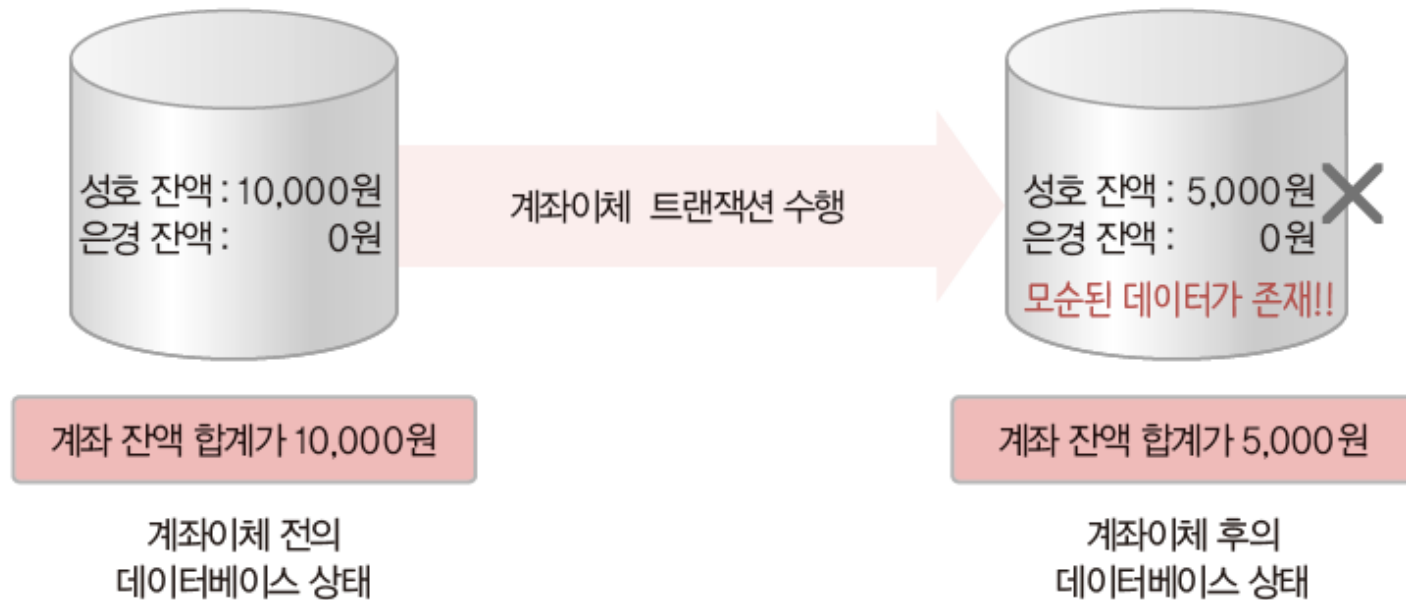
❖ 트랜잭션의 특성 – 일관성 (Consistency)

- 트랜잭션이 성공적으로 수행된 후에도 데이터베이스가 일관된 상태를 유지해야 함



트랜잭션

❖ 트랜잭션의 특성 – 일관성 (Consistency)



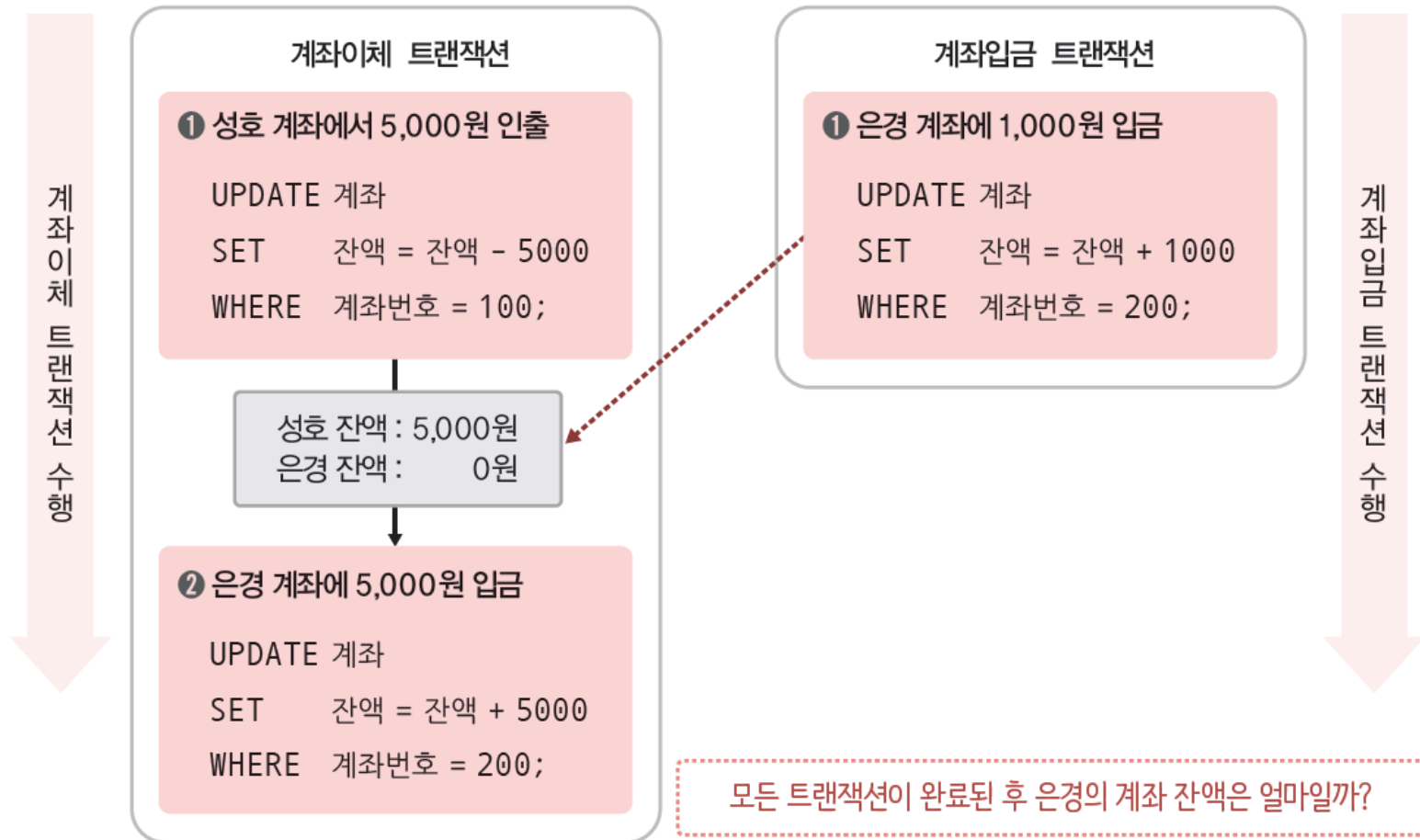
트랜잭션

❖ 트랜잭션의 특성 – 격리성 (Isolation)

- 수행 중인 트랜잭션이 완료될 때까지 다른 트랜잭션들이 중간 연산 결과에 접근할 수 없음
- 여러 트랜잭션이 동시에 수행되더라도 순서대로 하나씩 수행된 결과와 동일한 결과를 제공해야 함

트랜잭션

❖ 트랜잭션의 특성 – 격리성 (Isolation)



트랜잭션

❖ 트랜잭션의 특성 – 격리성

성호 잔액 : 10,000원
은경 잔액 : 0원

계좌이체 전의 데이터베이스 상태

계좌이체 트랜잭션

❶ 성호 계좌에서 5,000원 인출

UPDATE 계좌
SET 잔액 = 잔액 - 5000
WHERE 계좌번호 = 100;

❷ 은경 계좌에 5,000원 입금

UPDATE 계좌
SET 잔액 = 잔액 + 5000
WHERE 계좌번호 = 200;

성호 잔액 : 5,000원
은경 잔액 : 5,000원

계좌이체 후의 데이터베이스 상태

계좌입금 트랜잭션

❶ 은경 계좌에 1,000원 입금

UPDATE 계좌
SET 잔액 = 잔액 + 1000
WHERE 계좌번호 = 200 ;

성호 잔액 : 5,000원
은경 잔액 : 6,000원

계좌입금 후의 데이터베이스 상태

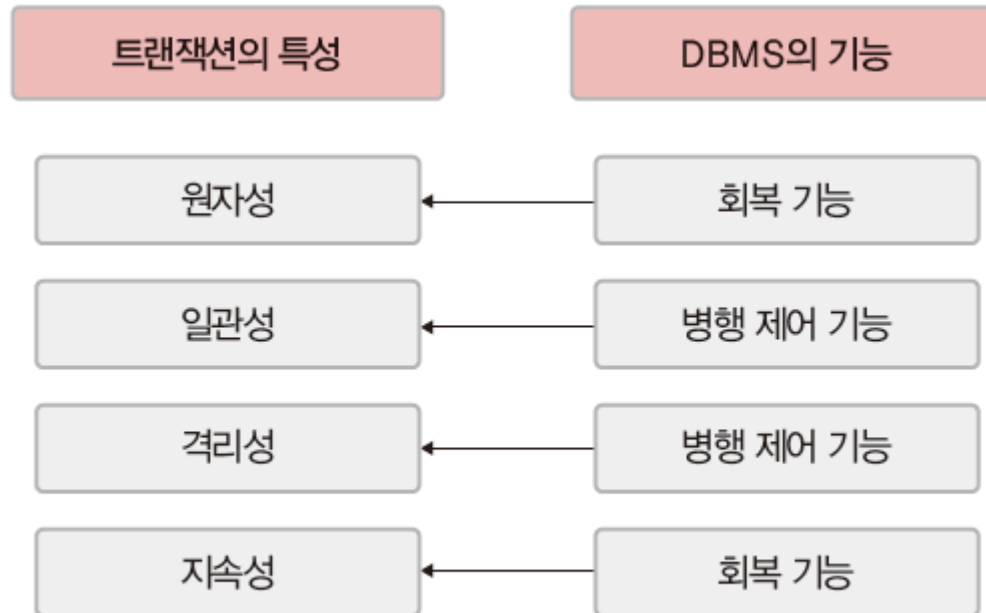
트랜잭션

❖ 트랜잭션의 특성 – 지속성 (Durability)

- 트랜잭션이 성공적으로 완료된 후 데이터베이스에 반영한 수행 결과는 영구적이어야 함 을 의미

트랜잭션

❖ 트랜잭션의 4가지 특성을 보장하기 위한 DBMS의 기능



트랜잭션

❖ 트랜잭션의 주요 연산



트랜잭션

❖ 트랜잭션의 주요 연산 – commit 연산

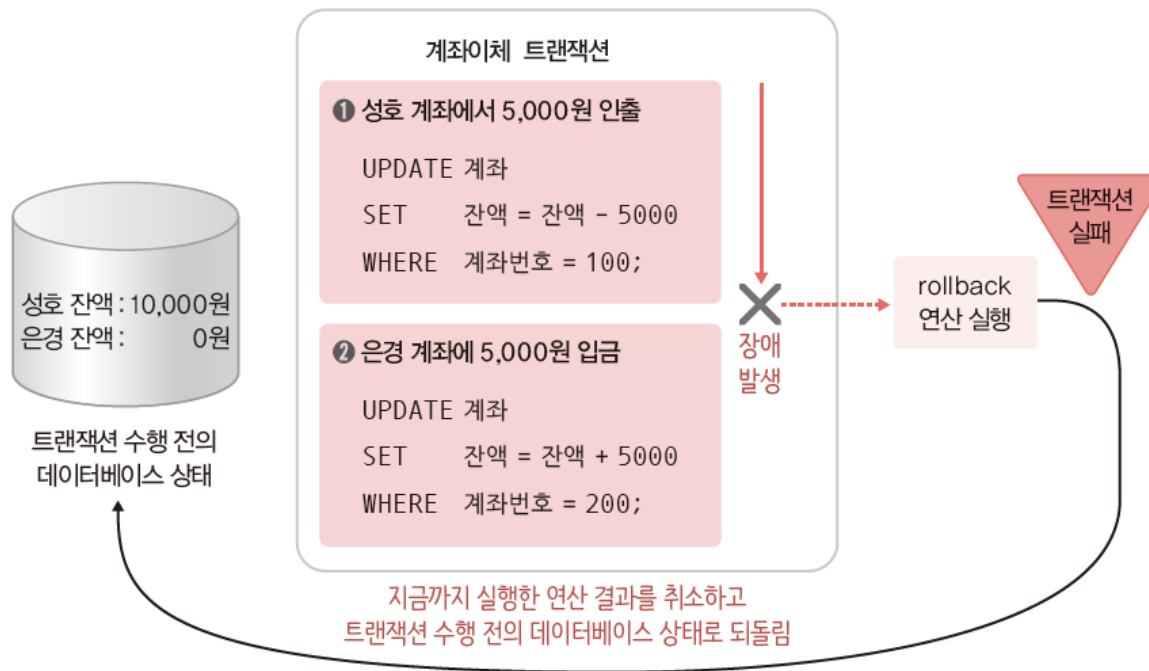
- 트랜잭션의 수행이 성공적으로 완료되었음을 선언하는 연산
- commit 연산이 실행되면 트랜잭션의 수행 결과가 데이터베이스에 반영되고 일관된 상태를 지속적으로 유지하게 됨



트랜잭션

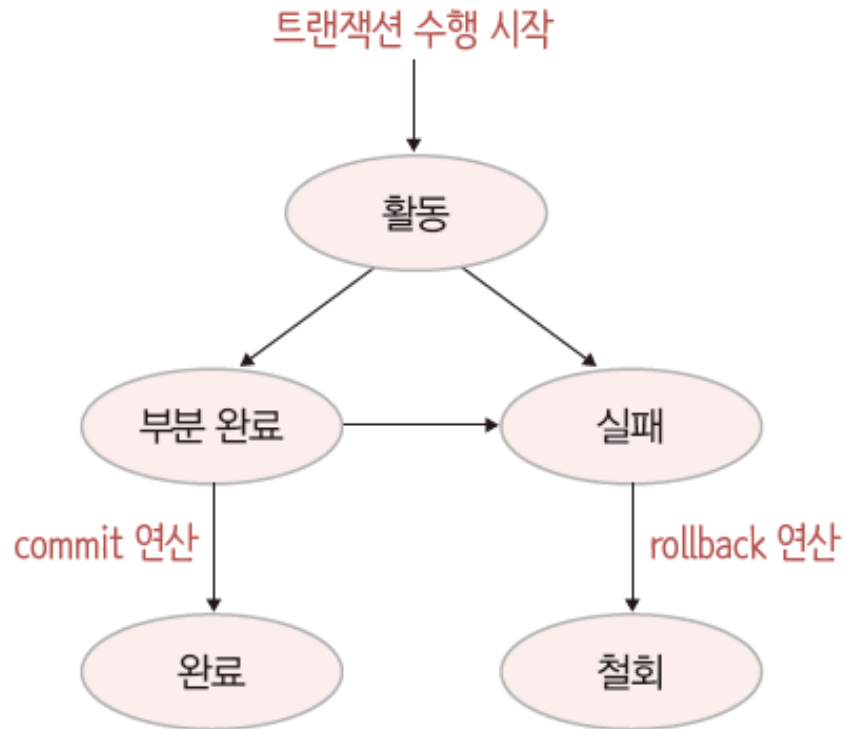
❖ 트랜잭션의 주요 연산 – rollback 연산

- 트랜잭션의 수행이 실패했음을 선언하는 연산
- rollback 연산이 실행되면 트랜잭션이 지금까지 실행한 연산의 결과가 취소되고 데이터베이스가 트랜잭션 수행 전의 일관된 상태로 되돌아감



트랜잭션

❖ 트랜잭션의 상태



트랜잭션

❖ 트랜잭션의 상태

- 활동(active) 상태
 - 트랜잭션이 실행 중인 상태
- 부분 완료(partially committed) 상태
 - 트랜잭션의 마지막 연산이 실행을 끝낸 직후의 상태
- 완료(committed) 상태
 - 트랜잭션이 성공적으로 완료되어 commit 연산을 실행한 상태
 - 트랜잭션이 수행한 최종 결과를 데이터베이스에 반영하고, 데이터베이스가 새로운 일관된 상태가 되면서 트랜잭션이 종료됨

트랜잭션

❖ 트랜잭션의 상태

- 실패(failed) 상태
 - 장애가 발생하여 트랜잭션의 실행이 중단된 상태
- 철회(aborted) 상태
 - 트랜잭션의 실행 실패로 rollback 연산을 실행한 상태
 - 지금까지 실행한 트랜잭션의 연산을 모두 취소하고 트랜잭션이 수행되기 전의 데이터베이스 상태로 되돌리면서 트랜잭션이 종료됨
 - 철회 상태로 종료된 트랜잭션은 상황에 따라 다시 수행되거나 폐기됨

장애와 회복

❖ 장애(failure)

- 시스템이 제대로 동작하지 않는 상태
- 장애 유형

유형	설명	
트랜잭션 장애	의미	트랜잭션 수행 중 오류가 발생하여 정상적으로 수행을 계속할 수 없는 상태
	원인	트랜잭션의 논리적 오류, 잘못된 데이터 입력, 시스템 자원의 과다 사용 요구, 처리 대상 데이터의 부재 등
시스템 장애	의미	하드웨어의 결함으로 정상적으로 수행을 계속할 수 없는 상태
	원인	하드웨어 이상으로 메인 메모리에 저장된 정보가 손실되거나 교착 상태가 발생한 경우 등
미디어 장애	의미	디스크 장치의 결함으로 디스크에 저장된 데이터베이스의 일부 혹은 전체가 손상된 상태
	원인	디스크 헤드의 손상이나 고장 등

장애와 회복

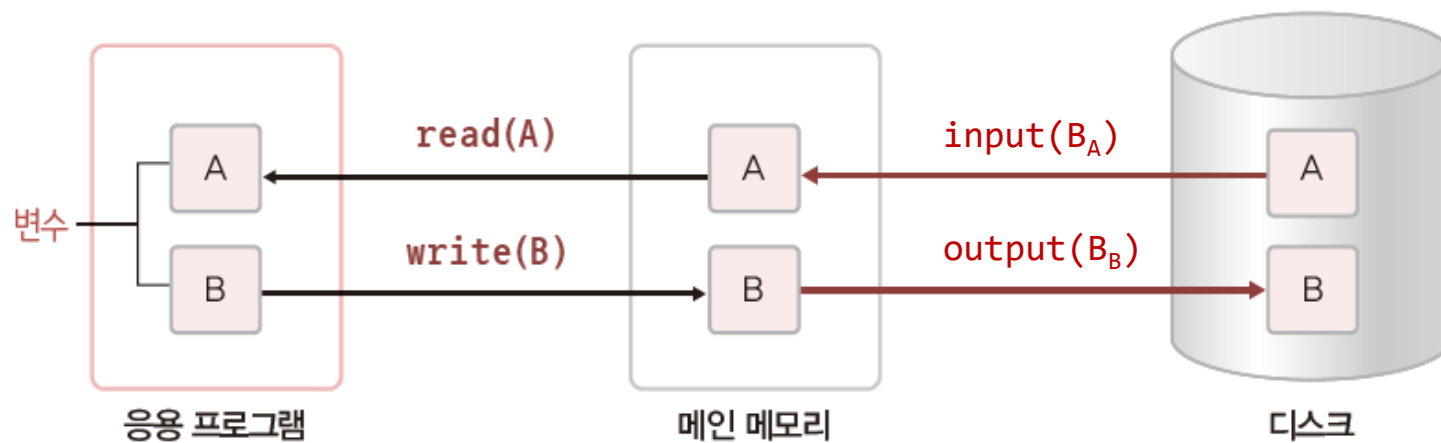
❖ 데이터베이스를 저장하는 저장 장치의 종류

저장 장치	설명	
휘발성 ^{volatile} 저장 장치 (소멸성)	의미	장애가 발생하면 저장된 데이터가 손실됨
	예	메인 메모리 등
비휘발성 ^{nonvolatile} 저장 장치 (비소멸성)	의미	장애가 발생해도 저장된 데이터가 손실되지 않음. 단, 디스크 헤더 손상 같은 저장 장치 자체에 이상이 발생하면 데이터가 손실될 수 있음
	예	디스크, 자기 테이프, CD/DVD 등
안정 ^{stable} 저장 장치	의미	비휘발성 저장 장치를 이용해 데이터 복사본 여러 개를 만드는 방법으로, 어떤 장애가 발생해도 데이터가 손실되지 않고 데이터를 영구적으로 저장할 수 있음

장애와 회복

❖ 트랜잭션의 수행을 위해 필요한 데이터 이동 연산

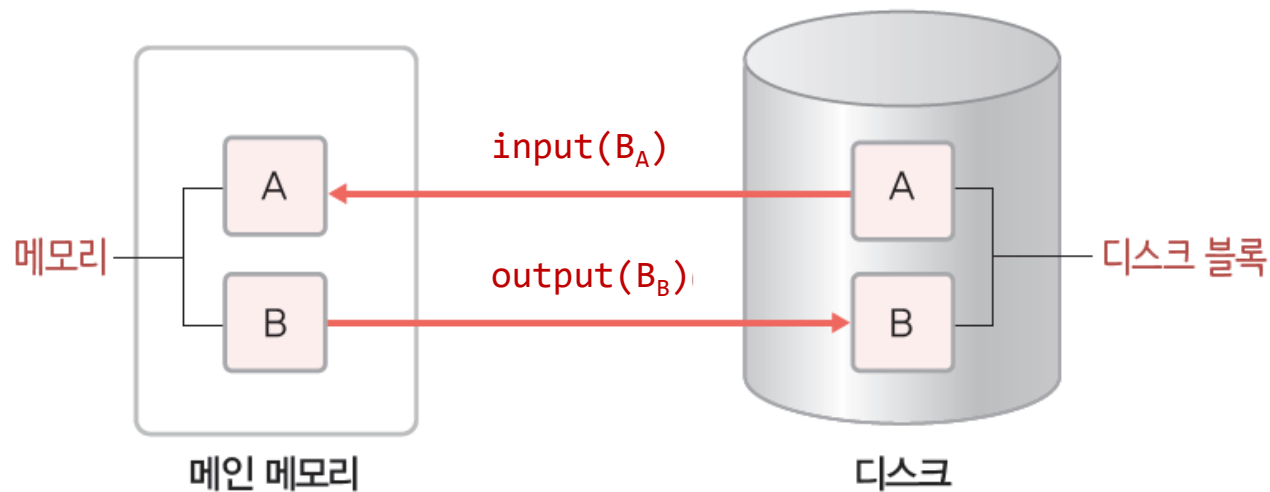
- 디스크와 메인 메모리 간의 블록 이동 연산 : `input()` / `output()`
- 메인 메모리와 응용 프로그램 변수 간의 데이터 이동 연산 : `read()` / `write()`



장애와 회복

❖ 디스크와 메인 메모리 간 데이터 이동 연산의 필요성

- 일반적으로 데이터베이스는 비휘발성 저장 장치인 디스크에 상주
- 트랜잭션이 데이터베이스의 데이터를 처리하기 위해서는 데이터를 디스크에서 메인 메모리로 가져와 처리한 다음 그 결과를 디스크로 보내는 작업이 필요함



장애와 회복

❖ 디스크와 메인 메모리 간의 데이터 이동 연산

- 블록(block) 단위로 수행됨
 - 디스크 블록 : 디스크에 있는 블록
 - 버퍼 블록 : 메인 메모리에 있는 블록
- input(B_x)
 - 데이터 x 를 포함하고 있는 디스크 블록(B_x)을 메인 메모리로 이동시키는 연산
 - 요청에 의해(On demand)
- output(B_x)
 - 데이터 x 를 포함하고 있는 메인 메모리 버퍼 블록(B_x)을 디스크 블록에 이동시키는 연산
 - OS의 버퍼 관리자에 의해(By Buffer Manager)

장애와 회복

❖ 메인 메모리와 변수 간의 데이터 이동 연산

- 응용 프로그램에서 트랜잭션의 수행 지시하는 경우 실행 연산

read(X)

메인 메모리 버퍼 블록에 저장되어 있는 데이터 X를 프로그램의 변수로 읽어오는 연산

write(X)

프로그램의 변수 값을 메인 메모리 버퍼 블록에 있는 데이터 X에 기록하는 연산

- 내부 동작 방식
 - read(X)
 - 내부적으로 $\text{input}(B_x)$ 가 실행
 - write(X)
 - write(X)를 위한 $\text{output}(B_x)$ 는 보통 즉각 실행되지 않음
 - write(X) 연산은 실행되었지만 실제로 $\text{output}(B_x)$ 가 실행되기 전에 시스템이 붕괴
→ 변경된 X 값 상실

장애와 회복

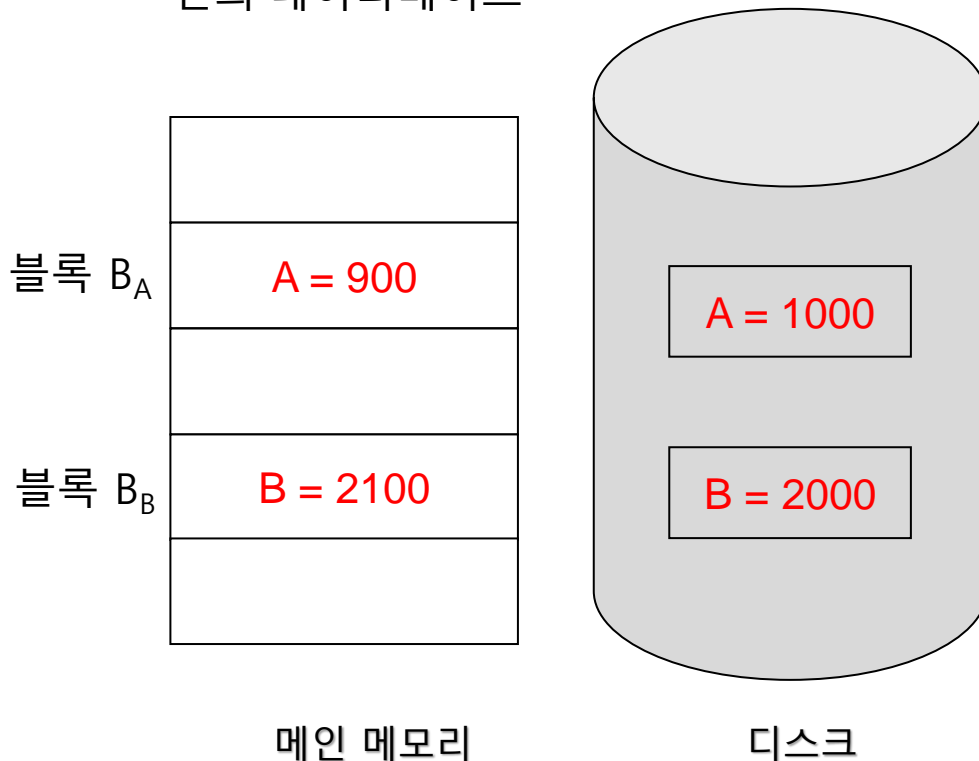
❖ output() 연산들을 실행하는 도중 장애 발생 예시

■ 트랜잭션 T

- 계좌 A에서 100원을 계좌 B에 이체(A=1000, B=2000)

```
T : read(A)
    A = A - 100
    write(A)
    read(B)
    B = B + 100
    write(B)
```

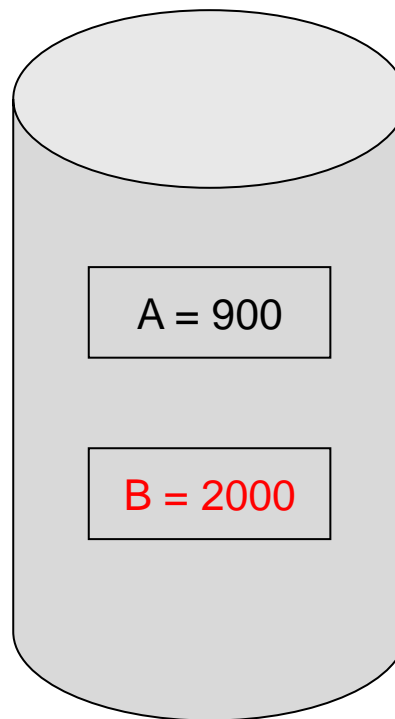
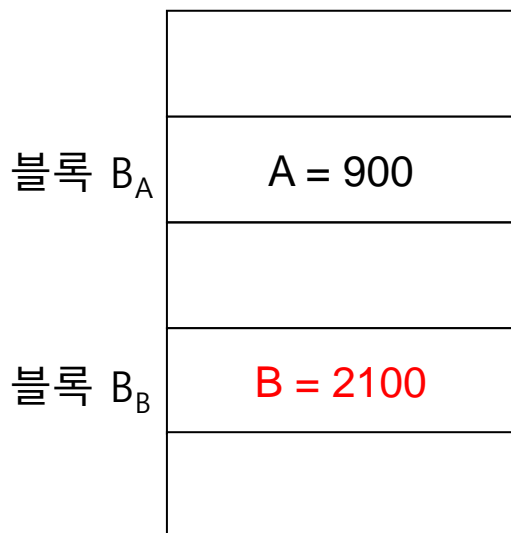
- output(B_A)와 output(B_B) 연산이 실행되기 직전의 데이터베이스



장애와 회복

❖ output() 연산들을 실행하는 도중 장애 발생

- output(B_A)를 수행하고 output(B_B)를 실행하려는 순간 장애 발생하는 경우



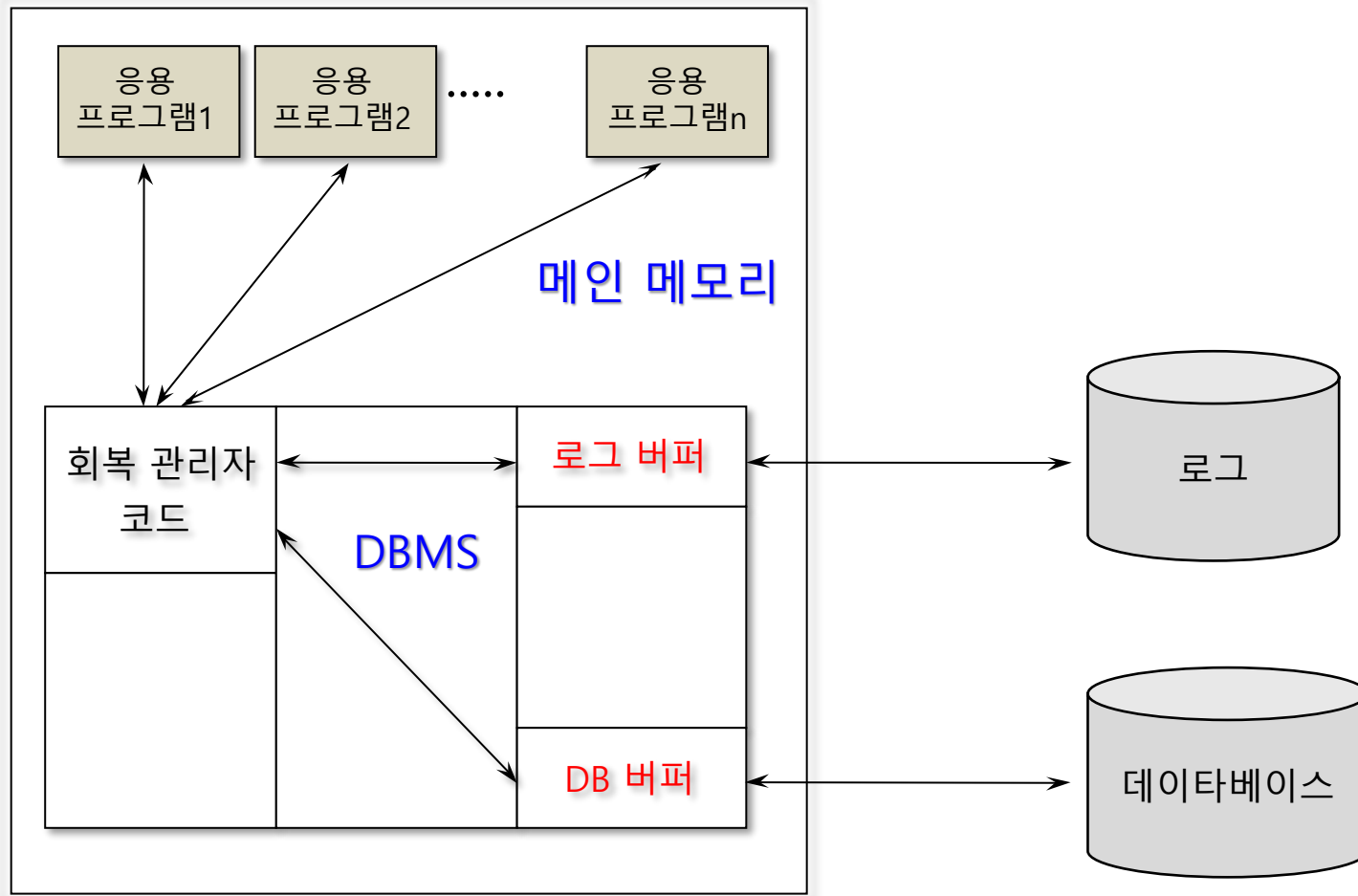
✓ 장애의 결과로 100을 상실 → 모순 상태(inconsistent state)

장애와 회복

❖ 회복(recovery)

- 장애가 발생했을 때 데이터베이스를 장애가 발생하기 전의 일관된 상태로 복구 시키는 것
- DBMS의 서브시스템인 회복 관리자(recovery manager)가 담당
 - 장애 발생을 탐지하고 데이터베이스를 복구 시키는 기능 제공

DBMS의 저장 구조



장애와 회복

❖ 회복의 기본 원리

→ 중복 (Redundancy)

덤프 (dump)	데이터베이스 전체를 다른 저장 장치에 주기적으로 복사하는 방법
로그 (log)	데이터베이스에서 변경 연산이 실행될 때마다 데이터를 변경하기 이전 값과 변경한 이후의 값을 별도의 파일에 기록하는 방법

장애와 회복

❖ 회복을 위한 연산

redo
(재실행)

가장 최근에 저장한 데이터베이스 복사본을 가져온 후 로그를 이용해 복사본이 만들어진 이후에 실행된 모든 변경 연산을 재실행하여 장애가 발생하기 직전의 데이터베이스 상태로 복구

undo
(취소)

로그를 이용해 지금까지 실행된 모든 변경 연산을 취소하여 데이터베이스를 원래의 상태로 복구

장애와 회복

❖ 로그 파일

- 데이터를 변경하기 이전의 값과 변경한 이후의 값을 기록한 파일
- 레코드 단위로 트랜잭션 수행과 함께 기록됨

로그 레코드	설명	
$\langle T_i, \text{start} \rangle$	의미	트랜잭션 T_i 가 수행을 시작했음을 기록
	예	$\langle T_1, \text{start} \rangle$
$\langle T_i, X, \text{old_value}, \text{new_value} \rangle$	의미	트랜잭션 T_i 가 데이터 X 를 이전 값 _{old_value} 에서 새로운 값 _{new_value} 으로 변경하는 연산을 실행했음을 기록
	예	$\langle T_1, X, 10000, 5000 \rangle$
$\langle T_i, \text{commit} \rangle$	의미	트랜잭션 T_i 가 성공적으로 완료되었음을 기록
	예	$\langle T_1, \text{commit} \rangle$
$\langle T_i, \text{abort} \rangle$	의미	트랜잭션 T_i 가 철회되었음을 기록
	예	$\langle T_1, \text{abort} \rangle$

장애와 회복

❖ 로그 기록 예

- 계좌 잔액이 10,000원인 성호가 계좌 잔액이 0원인 은경에게 5,000원 이체

계좌이체 트랜잭션: T_1

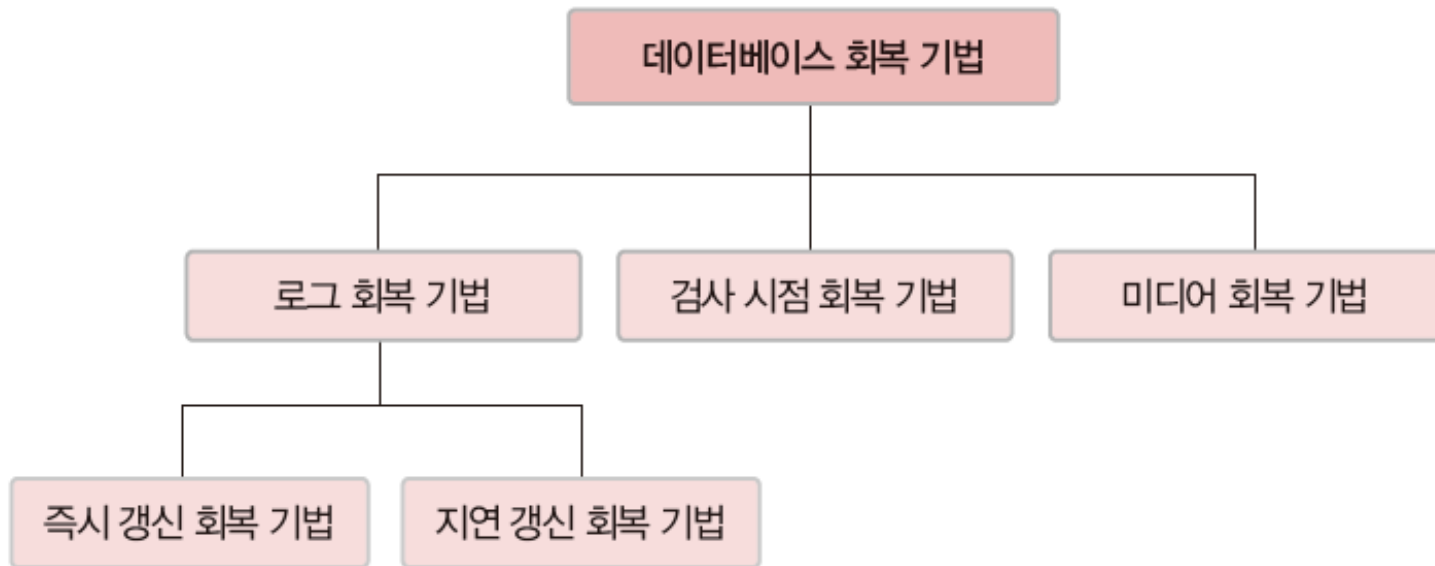
```
read(X);  
X = X - 5000;  
write(X);  
read(Y);  
Y = Y + 5000;  
write(Y);
```

로그 파일에 기록된 로그 레코드

```
1:  $\langle T_1, \text{start} \rangle$   
2:  $\langle T_1, X, 10000, 5000 \rangle$   
3:  $\langle T_1, Y, 0, 5000 \rangle$   
4:  $\langle T_1, \text{commit} \rangle$ 
```

장애와 회복

❖ 회복 기법



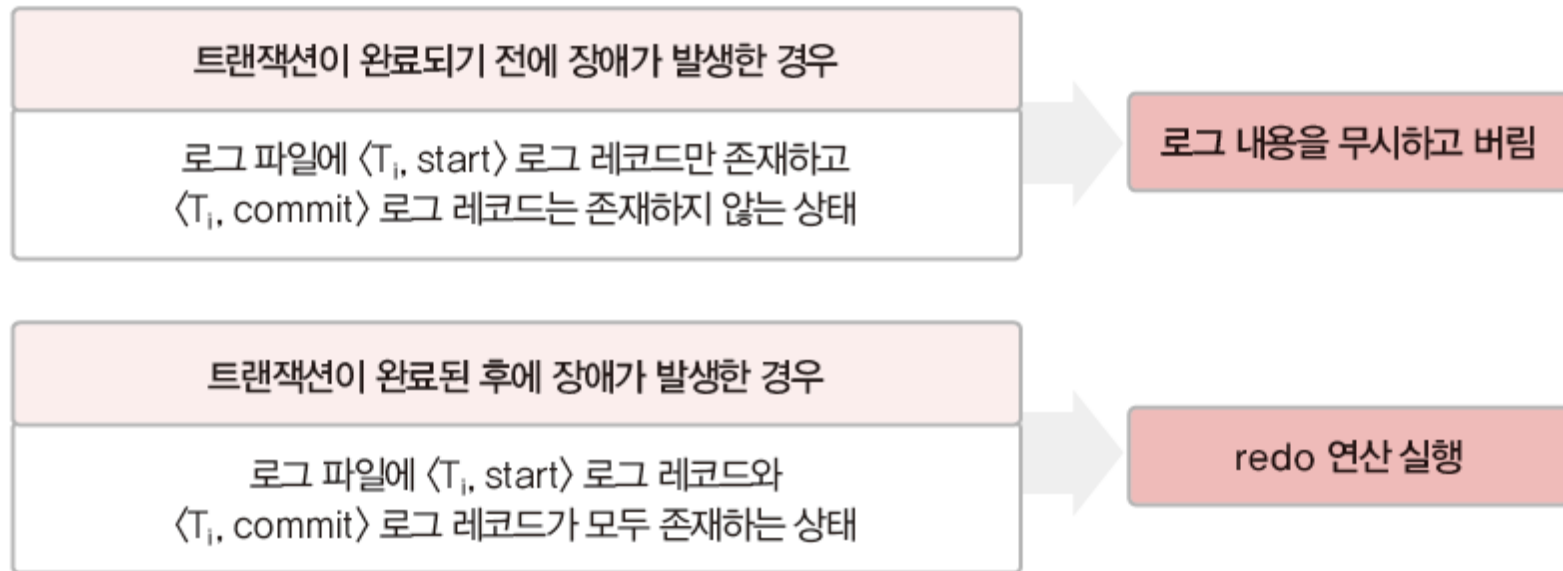
장애와 회복

❖ 로그 회복 기법 – 지연 갱신(deferred update) 회복 기법

- 트랜잭션이 '부분 완료' 상태가 될 때까지 모든 output() 연산을 지연시킴
 - 트랜잭션이 '활동' 상태 중에는 변경 결과를 로그 파일에만 기록함
 - $\langle T_i, \text{Commit} \rangle$ 를 포함하는 로그 레코드를 로그 파일에 기록한 후에 데이터베이스를 갱신하고 '완료' 상태로 감
- 트랜잭션 수행 중에 장애가 발생할 경우
 - 로그 레코드만 삭제하면 데이터베이스가 원래 상태를 돌아 갈 수 있음
 - redo 연산만 수행 (undo 연산은 필요 없음)
 - 로그 레코드에는 변경 이후 값만 기록하면 됨 : $\langle T_1, X, \text{new_value} \rangle$ 형식

장애와 회복

❖ 로그 회복 기법 – 지연 갱신 회복 기법



※ OS가 언제 output() 연산을 실행할 지 알 수 없으므로 redo 연산이 필요함

장애와 회복

❖ 로그 회복 기법 – 지연 갱신 회복 기법 적용 예

①에서 장애 발생 시 :
로그 기록을 무시하
고 별다른 회복 조치
를 하지 않음

로그 파일에 기록된 로그 레코드

1: $\langle T_1, \text{start} \rangle$
2: $\langle T_1, A, 4000 \rangle$
3: $\langle T_1, B, 1000 \rangle$
4: $\langle T_1, \text{commit} \rangle$
5: $\langle T_2, \text{start} \rangle$
6: $\langle T_2, C, 1000 \rangle$
7: $\langle T_2, D, 2000 \rangle$
8: $\langle T_2, \text{commit} \rangle$

데이터베이스

A = 5000, B = 0, C = 3000, D = 0

A = 4000, B = 1000

C = 1000, D = 2000

②에서 장애 발생 시 :
redo(T_1)

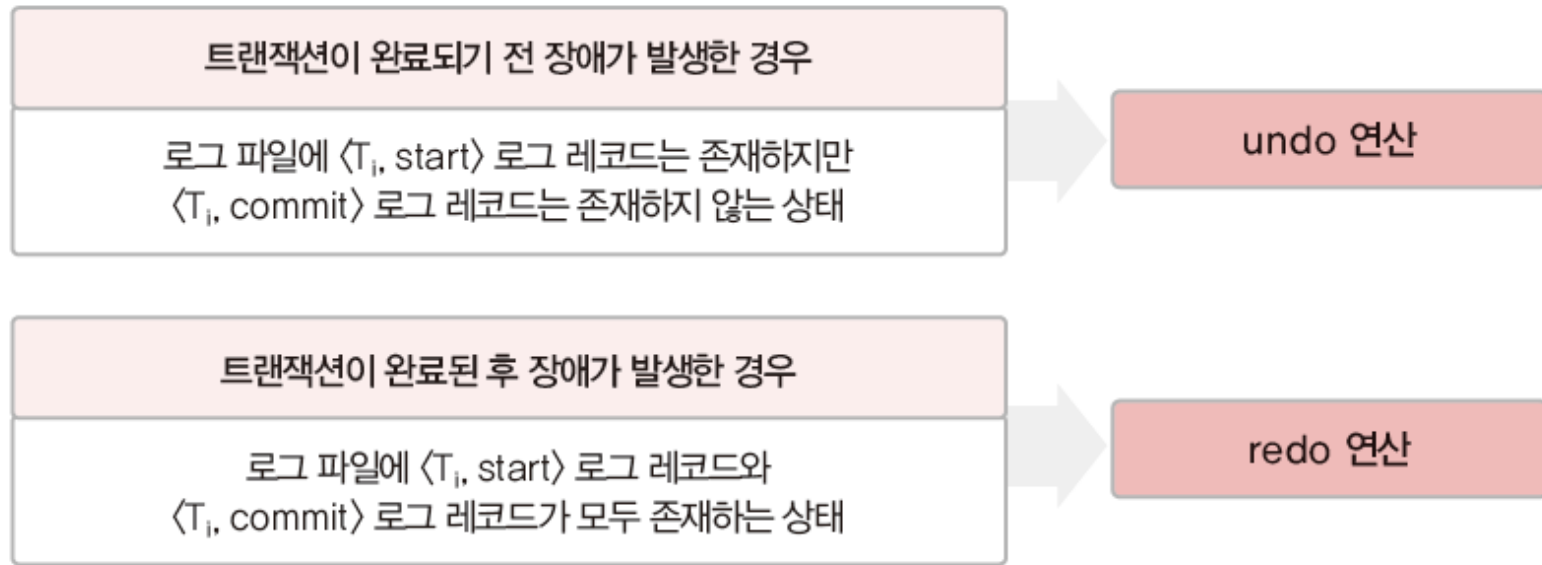
장애와 회복

❖ 로그 회복 기법 – 즉시 갱신(immediate update) 회복 기법

- 트랜잭션 '진행' 상태 중에 변경 연산의 결과를 데이터베이스에 그대로 반영함
 - 변경 내용을 로그 파일에 기록한 후 해당 output() 연산을 즉시 실행함
- 트랜잭션 수행 중에 장애가 발생할 경우
 - 장애 발생 시점에 따라 redo나 undo 연산을 실행해 데이터베이스를 복구
 - 로그 레코드에는 변경 전 후 값을 모두 기록함 : $\langle T_1, X, \text{old_value}, \text{new_value} \rangle$ 형식

장애와 회복

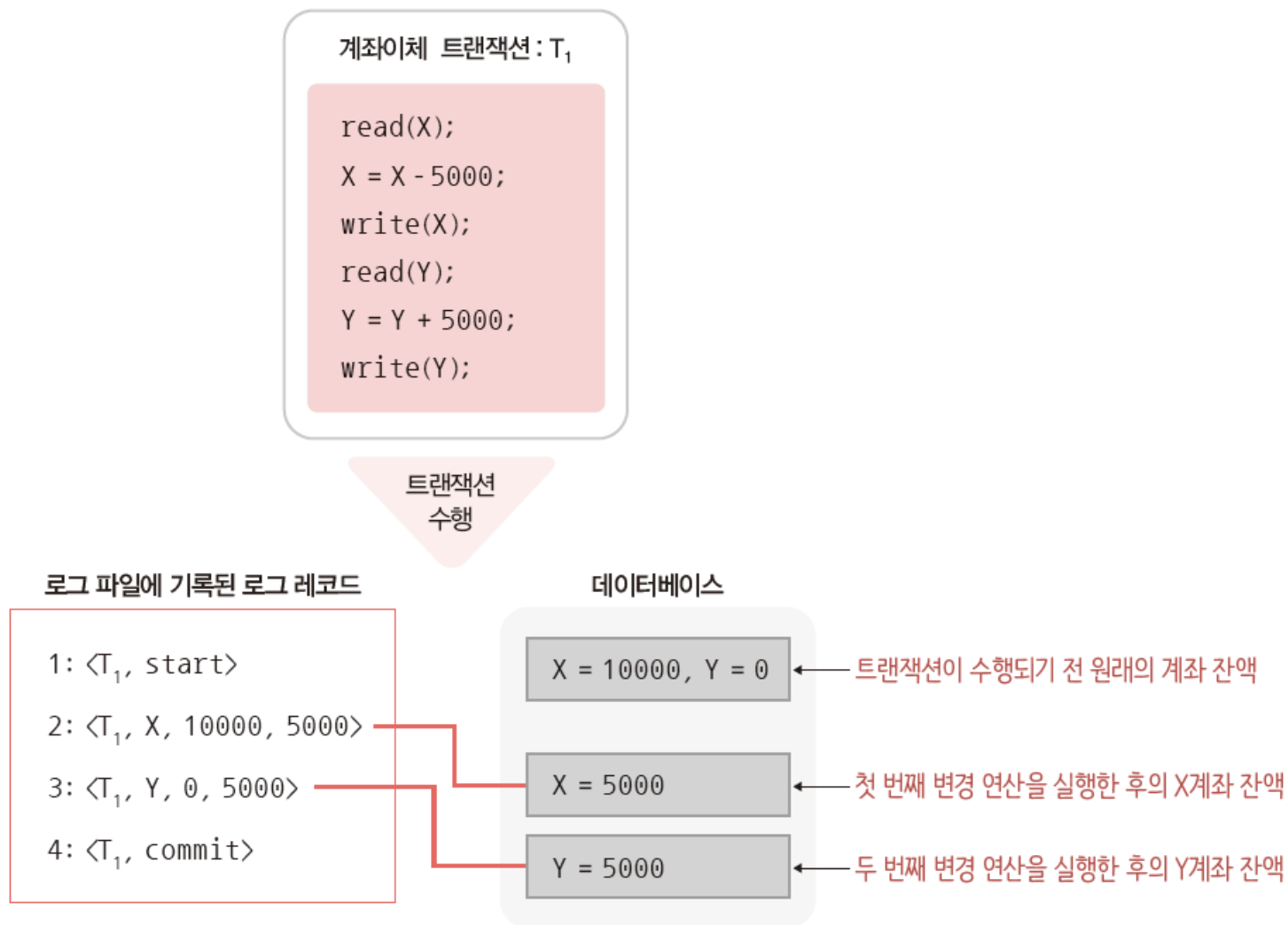
❖ 로그 회복 기법 – 즉시 갱신 회복 기법



※ OS가 언제 output() 연산을 실행했는지 혹은 실행할 지 알 수 없으므로 undo 및 redo 연산 모두 필요함

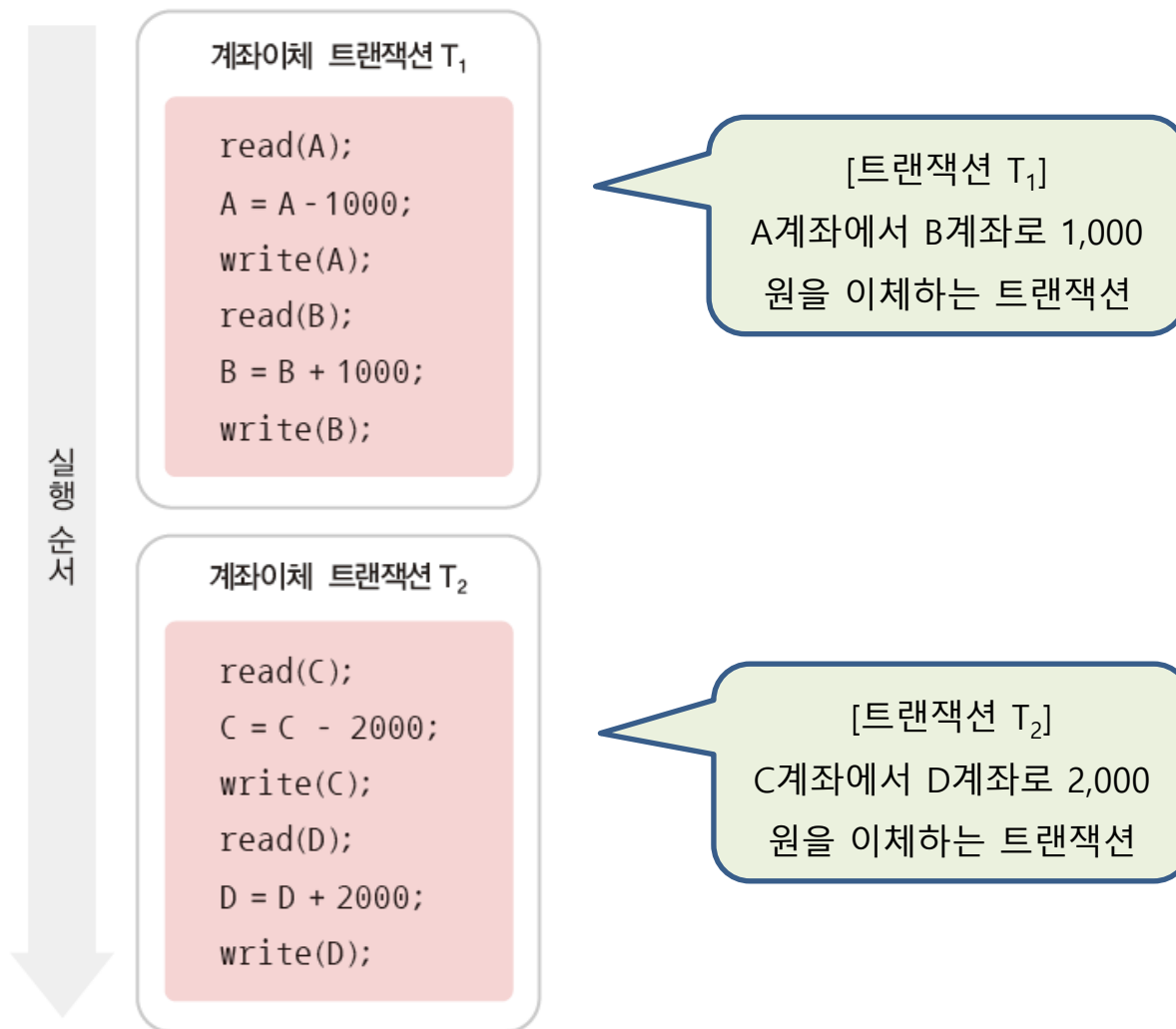
장애와 회복

❖ 로그 회복 기법 – 즉시 갱신 회복 기법



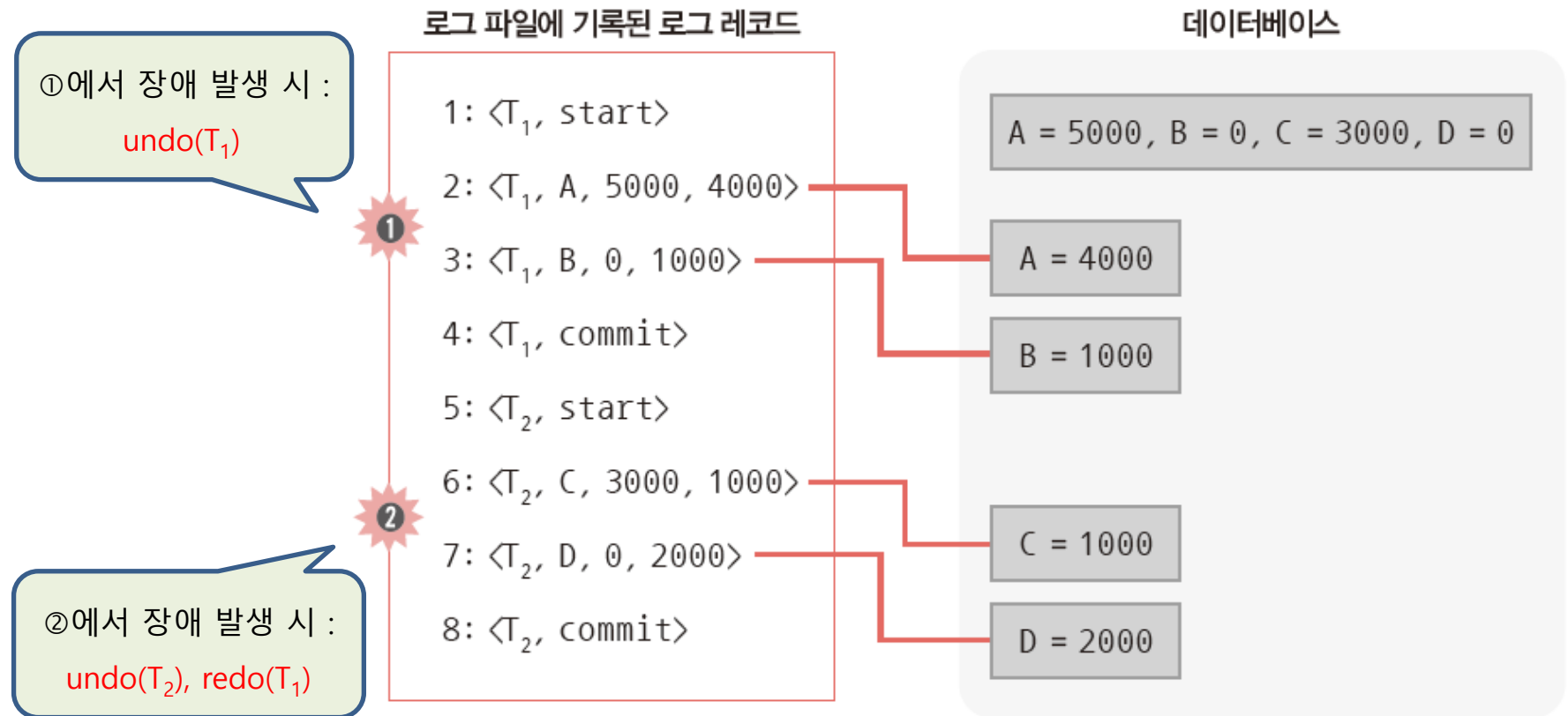
장애와 회복

❖ 로그 회복 기법 – 즉시 갱신 회복 기법 적용 예



장애와 회복

❖ 로그 회복 기법 – 즉시 갱신 회복 기법 적용 예



장애와 회복

❖ 검사 시점 회복

- 로그 전체를 대상으로 redo와 undo를 해야 될 트랜잭션을 결정해야 하는 비효율성 문제를 해결함
 - 작업 범위가 한정되어 회복 작업 시간이 단축됨
- 일정 시간 간격으로 검사 시점(checkpoint)을 설정하고 검사 시점에서 다음 작업을 수행함
 - 1) 메인 메모리(로그 버퍼)에 있는 모든 로그 레코드를 디스크에 있는 로그 파일에 기록함
 - 2) 메인 메모리(DB 버퍼)에 있는 모든 변경 연산의 결과를 디스크에 있는 데이터베이스에 반영함
 - 3) 검사 시점을 표시하는 <checkpoint L> 로그 레코드를 로그 파일에 기록함
 - L은 현재 실행되고 있는 트랜잭션의 리스트
- 장애 발생 시 가장 최근 검사 시점 이후의 트랜잭션에만 회복 작업 수행

장애와 회복

❖ 미디어 회복 기법

- 디스크에 발생할 수 있는 장애에 대비한 회복 기법
- 덤프(복사본) 이용
 - 전체 데이터베이스의 내용을 일정 주기마다 다른 안전한 저장 장치에 복사
- 디스크 장애가 발생하는 경우
 - 가장 최근에 복사해둔 덤프를 이용해 장애 발생 이전의 데이터베이스 상태로 복구하고 필요에 따라 redo 연산을 수행

병행 제어

❖ 병행 수행과 병행 제어

■ 병행 수행(concurrency)

- 여러 사용자가 데이터베이스를 동시 공유할 수 있도록 여러 개의 트랜잭션을 동시에 수행하는 것
- 여러 트랜잭션이 차례로 번갈아 수행되는 인터리빙(interleaving) 방식으로 진행됨

■ 병행 제어(concurrency control)

- 여러 트랜잭션이 같은 데이터에 접근하여 연산을 실행하더라도 정확한 수행 결과를 얻을 수 있도록 트랜잭션의 수행을 제어하는 것

병행 제어

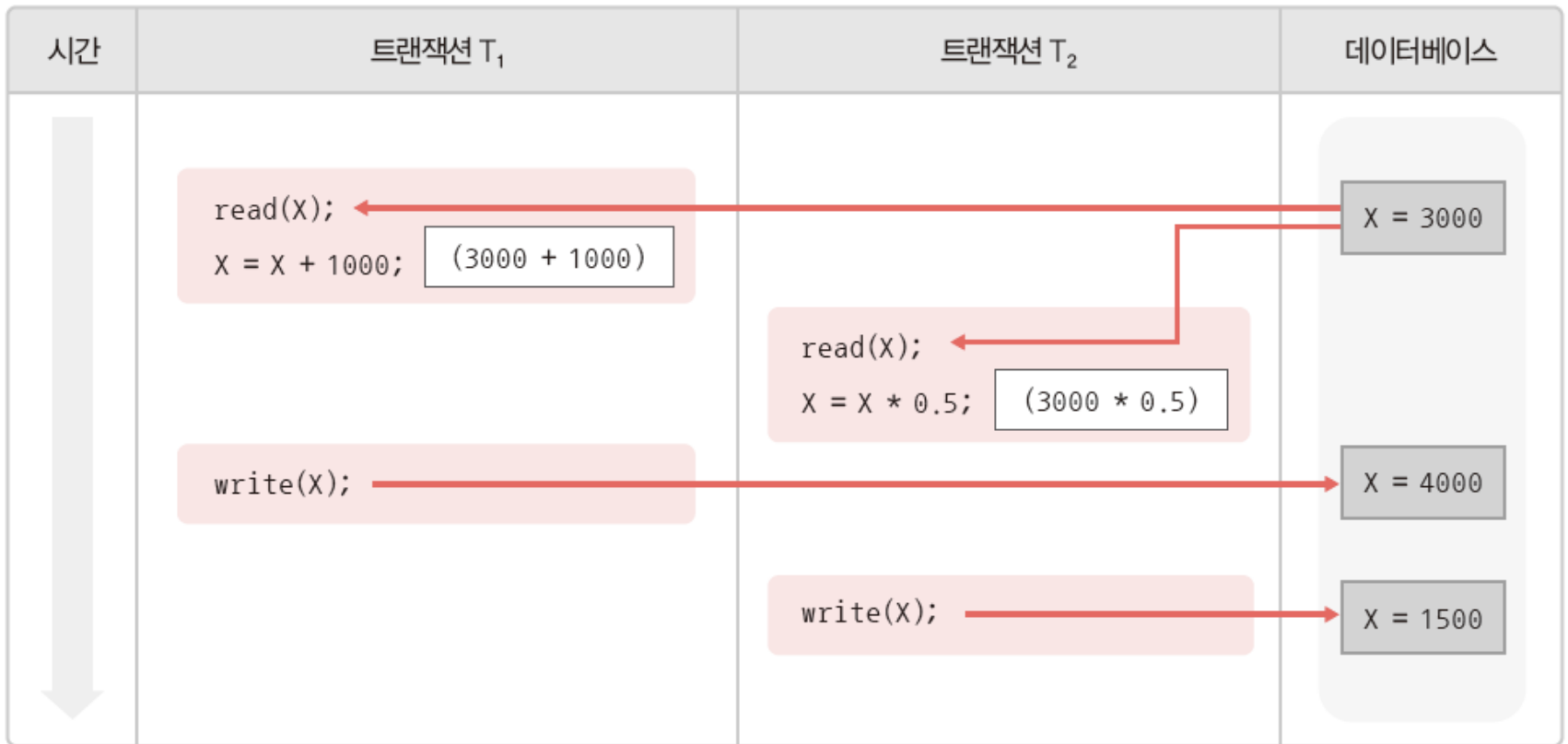
❖ 병행 수행 시 발생할 수 있는 문제점

■ 갱신 분실(lost update)

- 하나의 트랜잭션이 수행한 데이터 변경 연산의 결과를 다른 트랜잭션의 덮어쓰기로 인해 변경 연산이 무효화되는 것

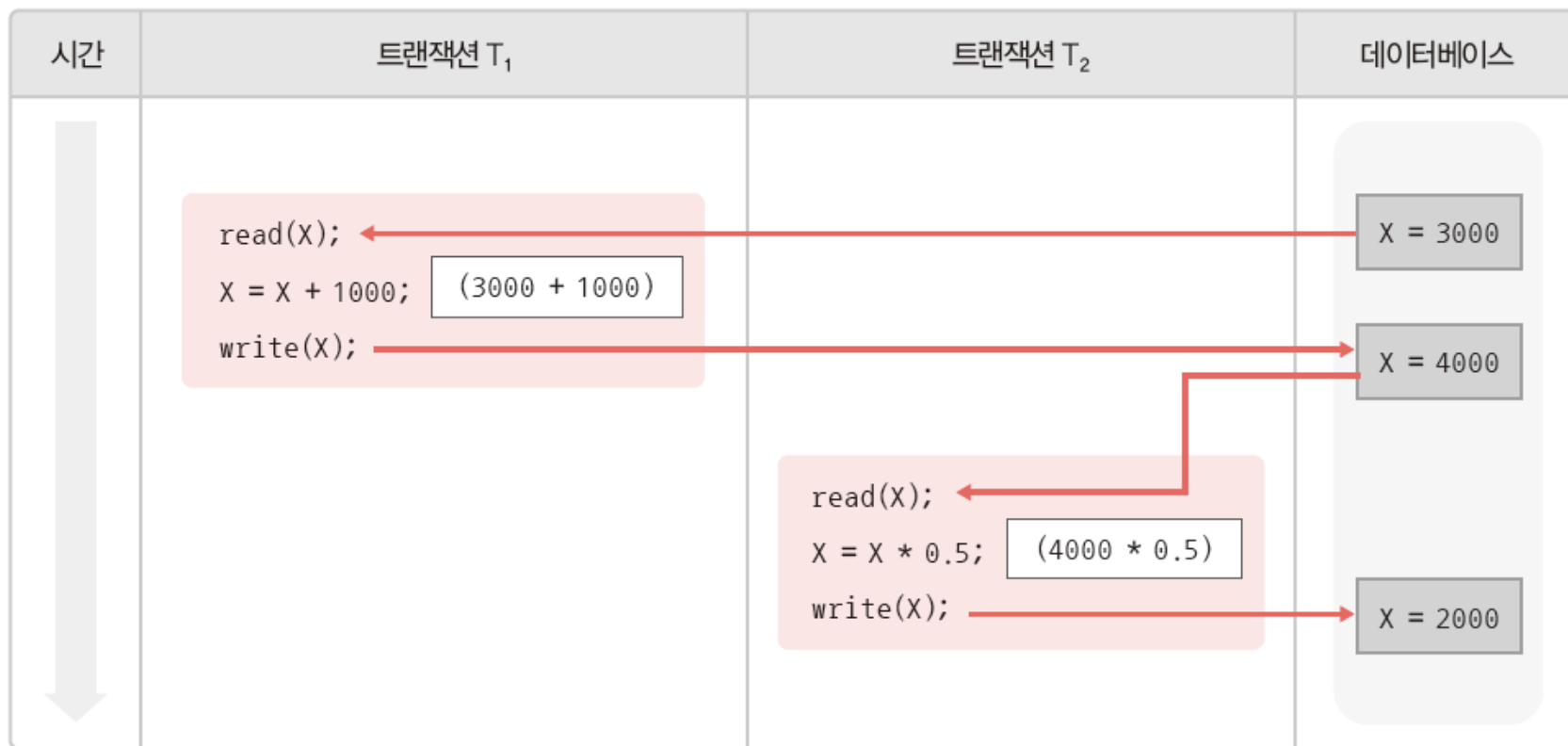
병행 제어

❖ 갱신 분실 예시



병행 제어

❖ 정상 동작 예시



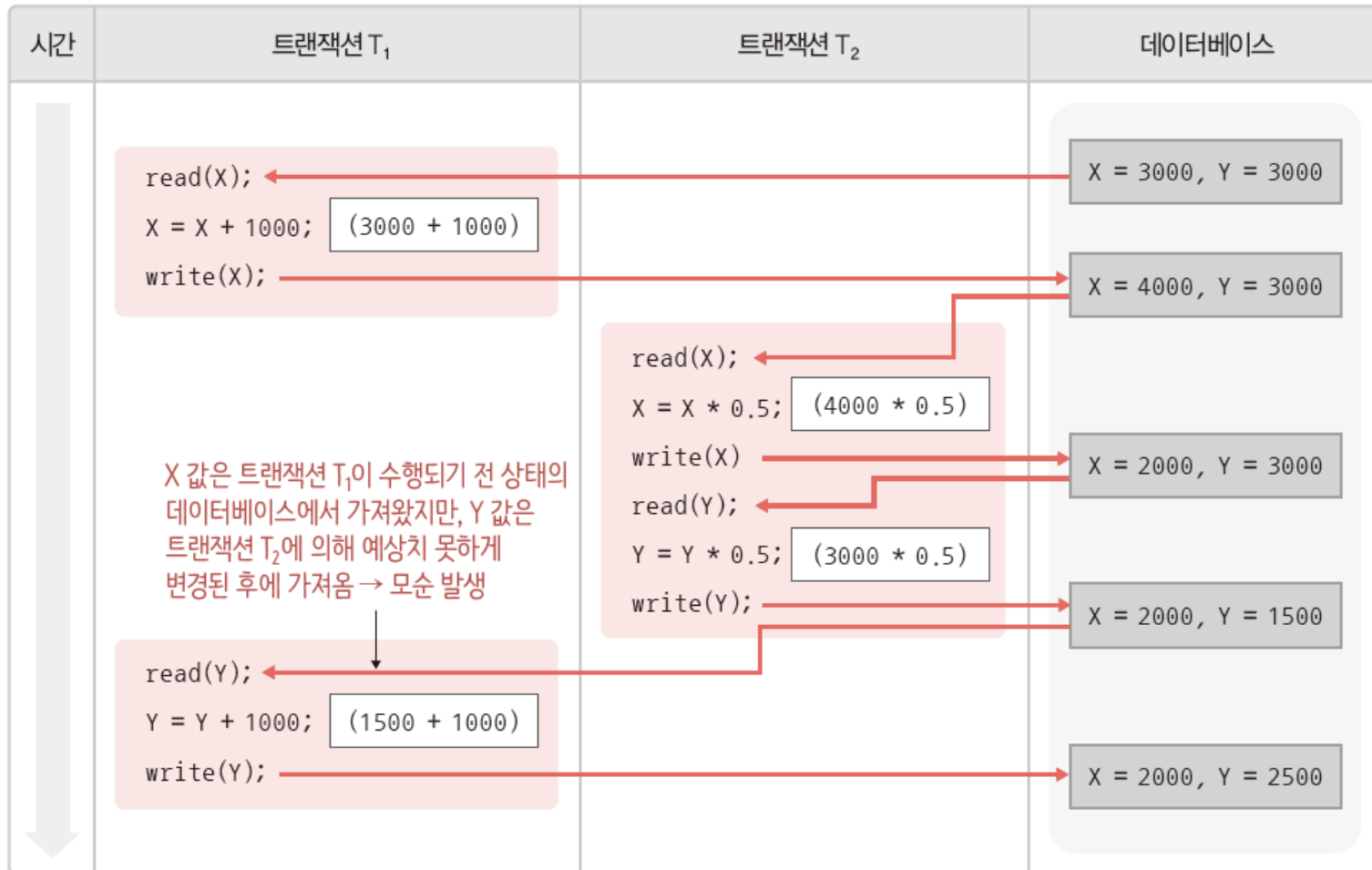
병행 제어

❖ 병행 수행 시 발생할 수 있는 문제점

- 모순성(inconsistency)
 - 하나의 트랜잭션이 일관성 없는 상태의 데이터베이스에서 데이터를 가져와서 연산함으로써 모순된 결과가 발생하는 것

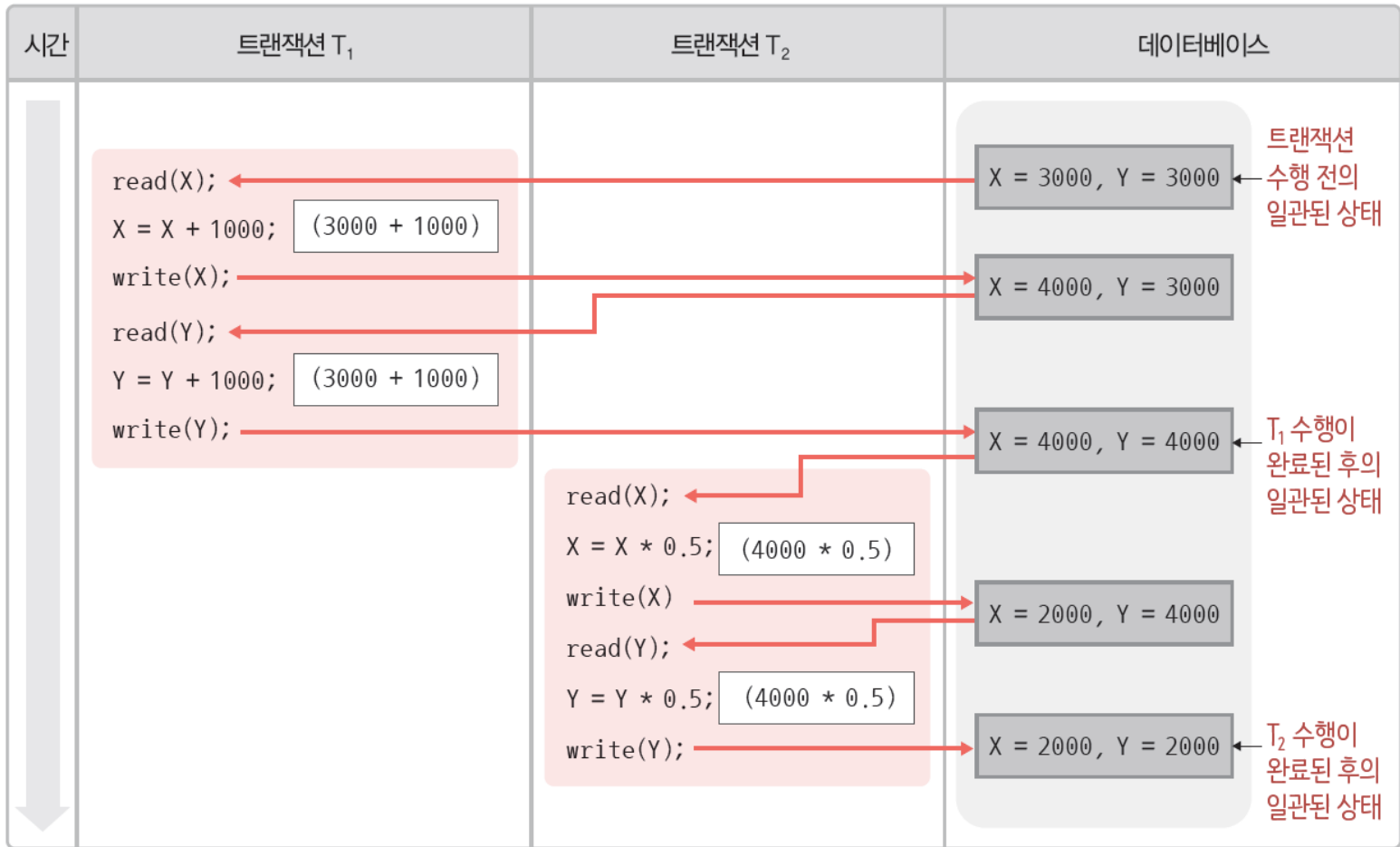
병행 제어

❖ 모순성 예시



병행 제어

❖ 정상 동작 예시



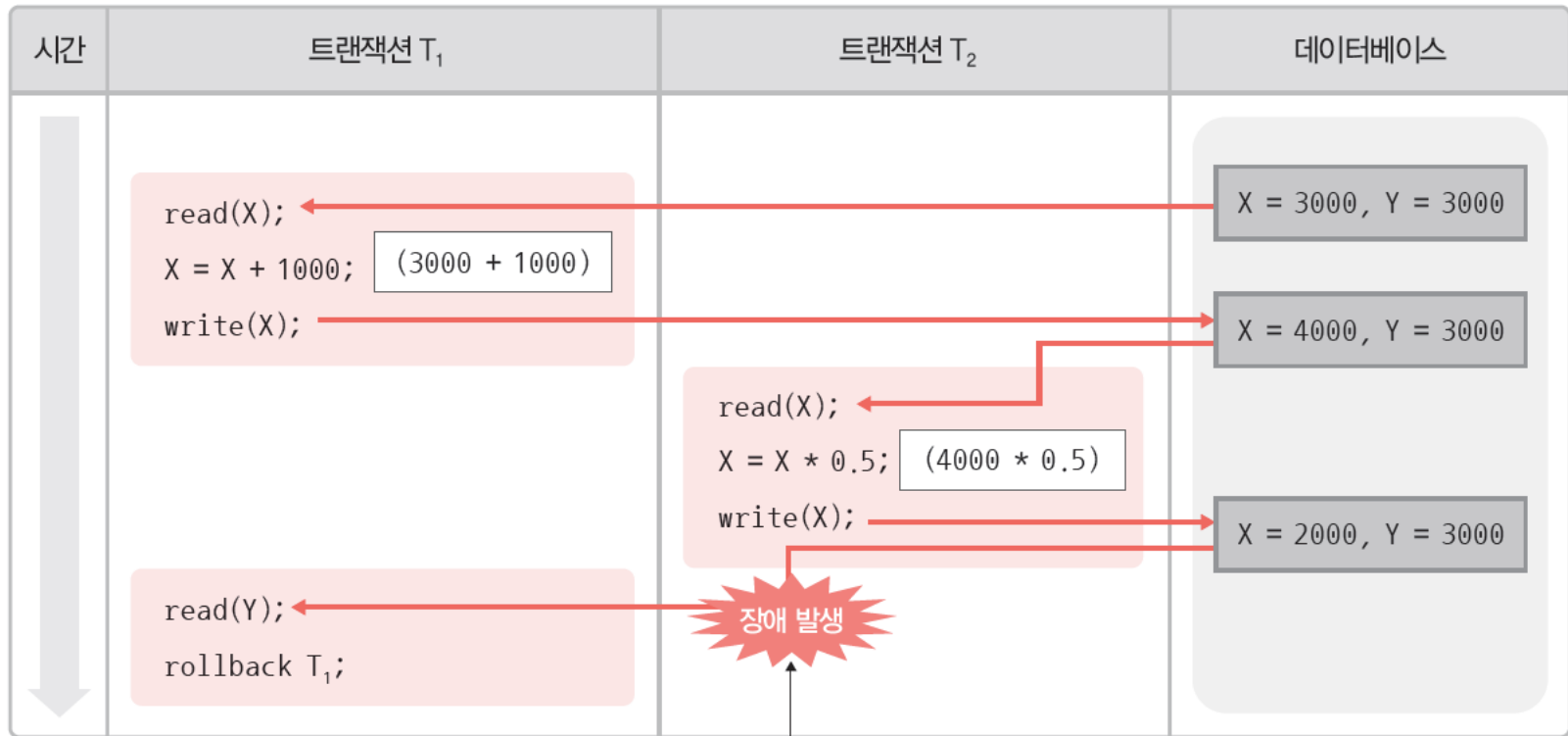
병행 제어

❖ 병행 수행 시 발생할 수 있는 문제점

- 연쇄 복귀(cascading rollback)
 - 트랜잭션이 완료되기 전 장애가 발생하여 rollback 연산을 수행하면, 장애 발생 전에 이 트랜잭션이 변경한 데이터를 가져가서 변경 연산을 실행한 다른 트랜잭션에도 rollback 연산을 연쇄적으로 실행해야 한다는 것

병행 제어

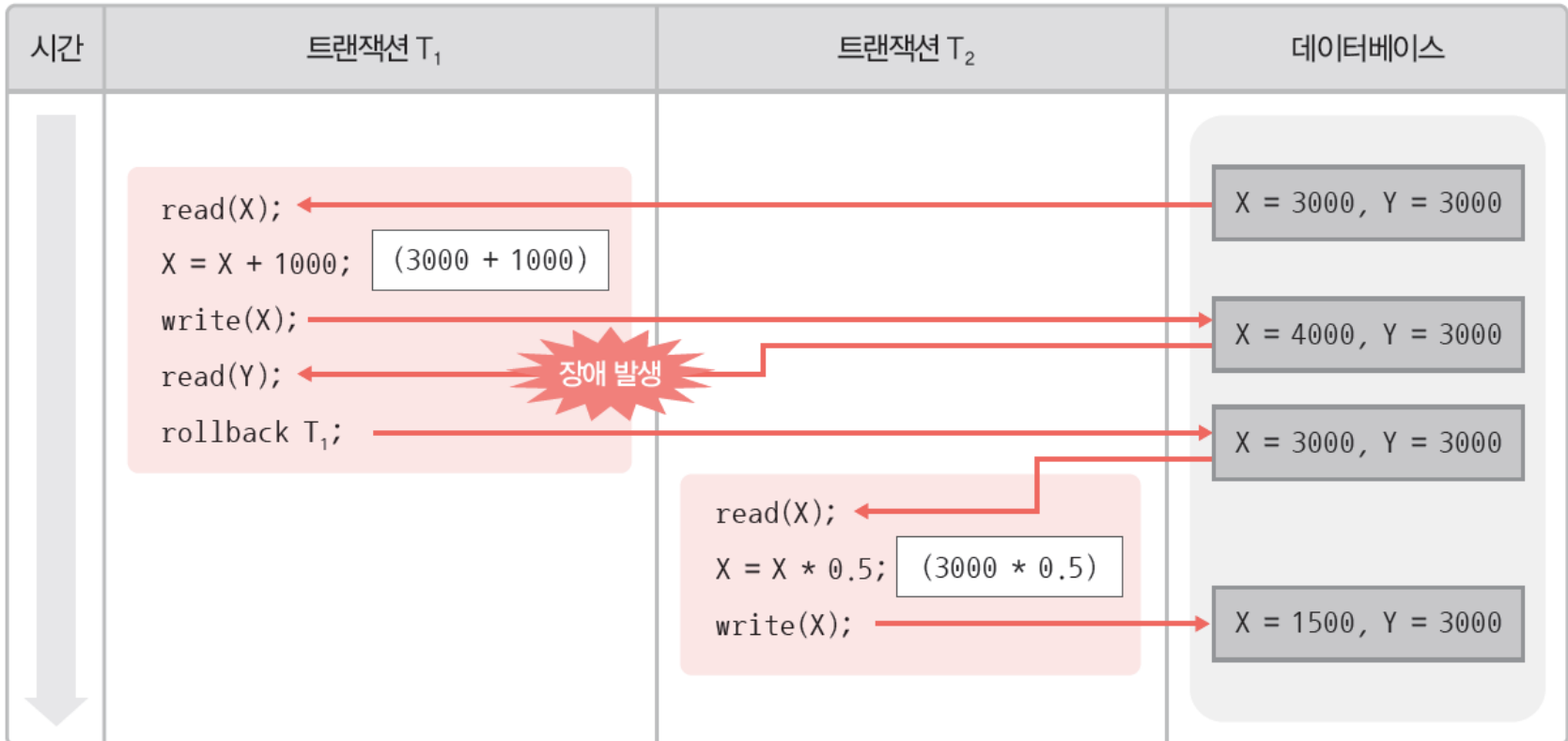
❖ 연쇄 복구 예시



트랜잭션 T_1 이 rollback되면 트랜잭션 T_2 도 rollback되어야 하는데
 T_2 가 이미 완료된 트랜잭션이라 rollback을 할 수 없음

병행 제어

❖ 정상 동작 예시



병행 제어

❖ 연쇄 복귀의 원인

■ 충돌 (conflict)

- ① 상이한 트랜잭션에 속하고 있으면서 동일한 데이터 아이템을 처리 대상으로 하는 두 연산
- ② 최소한 하나의 기록(write) 연산
 - 공용하는 충돌된 데이터를 통해 트랜잭션 사이에 간섭이 일어나기 때문

※ 병행 제어

- 충돌 데이터의 관리

병행 제어

❖ 트랜잭션 스케줄

- 트랜잭션에 포함되어 있는 연산들의 실행 순서

트랜잭션 스케줄	의미
직렬 스케줄	인터리빙 방식을 이용하지 않고 각 트랜잭션별로 연산들을 순차적으로 실행시키는 것
비직렬 스케줄	인터리빙 방식을 이용하여 트랜잭션들을 병행해서 수행시키는 것
직렬 가능 스케줄	직렬 스케줄과 같이 정확한 결과를 생성하는 비직렬 스케줄

병행 제어

❖ 직렬 스케줄(serial schedule)

■ 의미

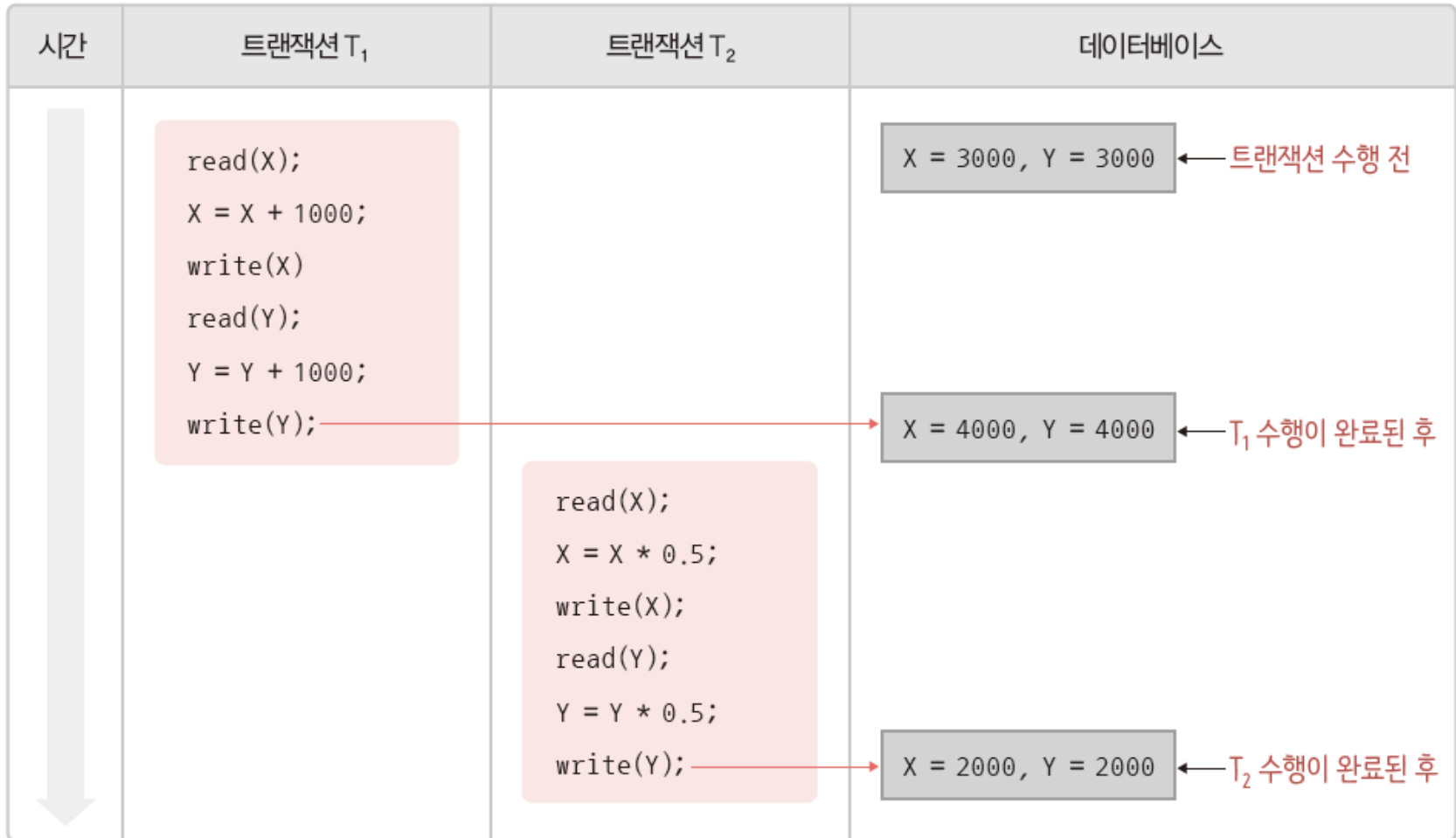
- 인터리빙 방식을 이용하지 않고 각 트랜잭션별로 연산들을 순차적으로 실행시키는 것

■ 특징

- 다른 트랜잭션의 방해를 받지 않고 독립적으로 수행되므로 항상 모순이 없는 정확한 결과를 얻게 됨
- 직렬 스케줄마다 데이터베이스에 반영되는 최종 결과가 다를 수 있지만 직렬 스케줄의 결과는 모두 정확함

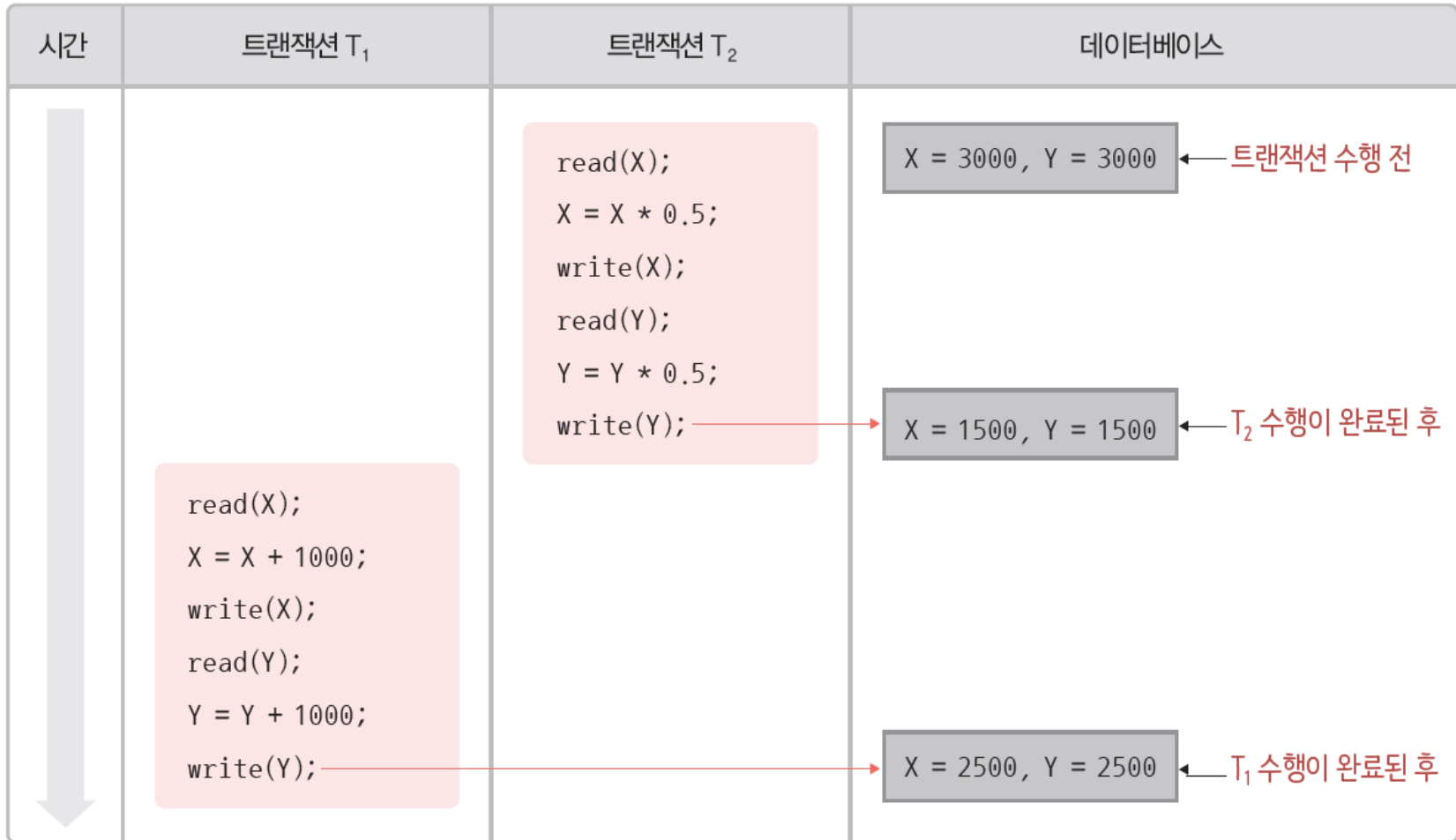
병행 제어

❖ 직렬 스케줄 예시 (1)



병행 제어

❖ 직렬 스케줄 예시 (2)



병행 제어

❖ 비직렬 스케줄(nonserial schedule)

■ 의미

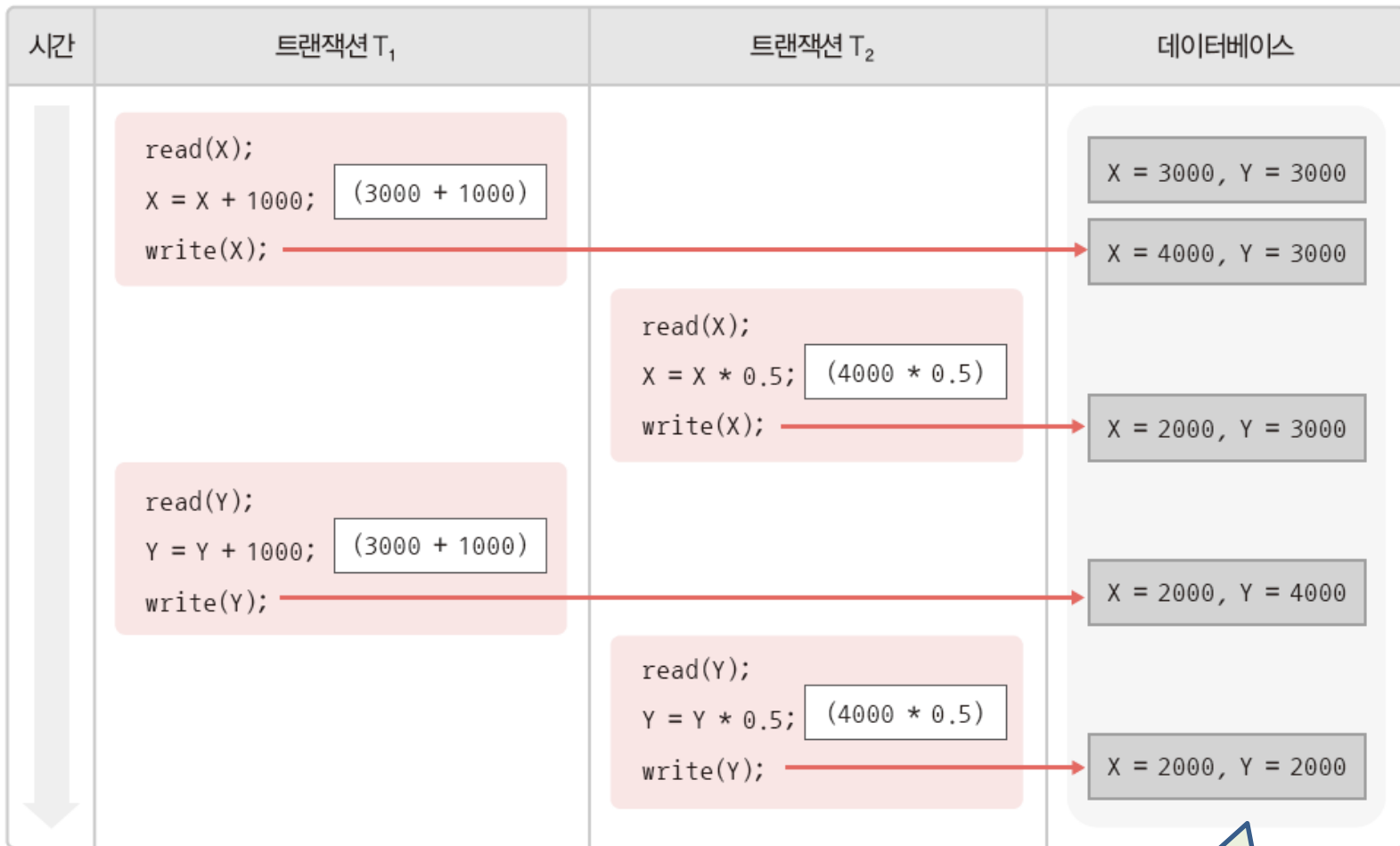
- 인터리빙 방식을 이용하여 트랜잭션을 병행 수행하는 것

■ 특징

- 트랜잭션이 번갈아 연산을 실행하기 때문에 하나의 트랜잭션이 완료되기 전에 다른 트랜잭션의 연산이 실행될 수 있음
- 비직렬 스케줄에 따라 병행 수행하면 갱신 분실, 모순성, 연쇄 복귀 등의 문제가 발생할 수 있어 결과의 정확성을 보장할 수 없음
- 다양한 비직렬 스케줄이 만들어질 수 있고 정확한 결과 혹은 잘못된 결과를 생성할 수 있음

병행 제어

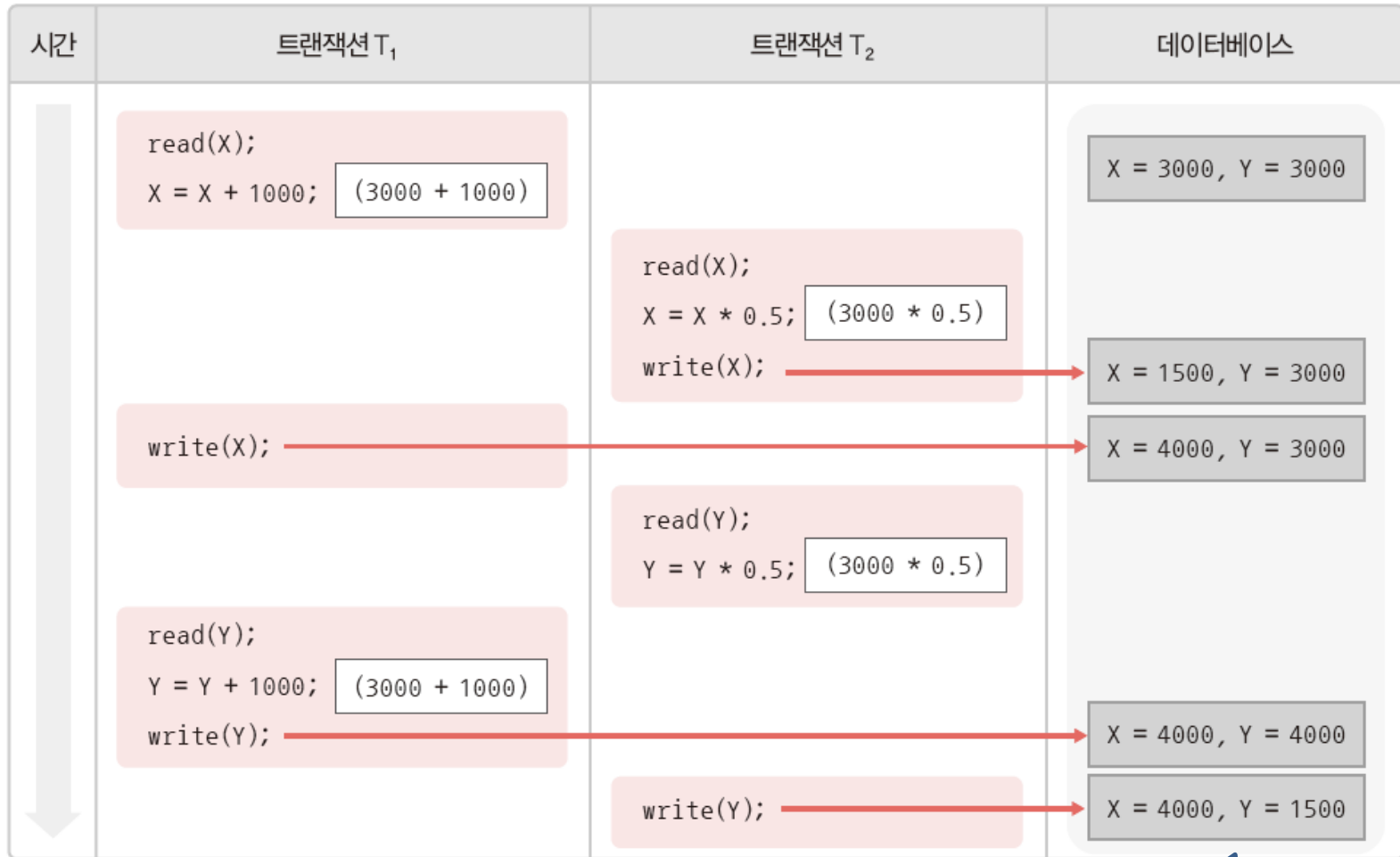
❖ 비직렬 스케줄 예시 (1)



정확한 결과

병행 제어

❖ 비직렬 스케줄 예시 (2)



잘못된 결과

병행 제어

❖ 직렬 가능 스케줄(serializable schedule)

■ 의미

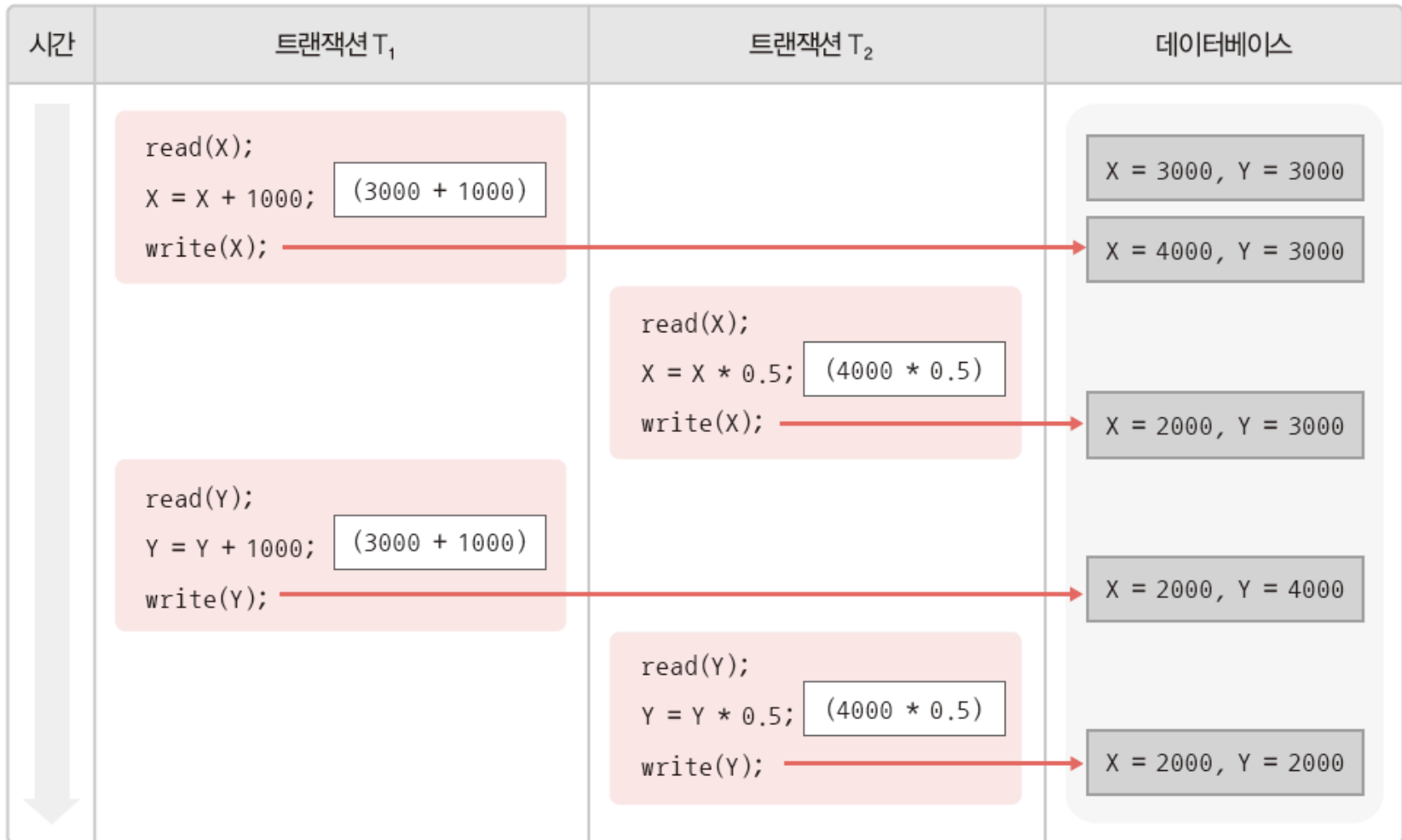
- 직렬 스케줄에 따라 수행한 것과 같이 정확한 결과를 생성하는 비직렬 스케줄
- 비직렬 스케줄 중에서 수행 결과가 동일한 직렬 스케줄이 있는 것

■ 특징

- 인터리빙 방식으로 병행 수행하면서도 정확한 결과를 얻을 수 있음
- 직렬 가능 스케줄인지 판단하는 것은 간단한 작업이 아니므로 직렬 가능성을 보장하는 병행 제어 기법을 사용하는 것이 일반적임

병행 제어

❖ 직렬 가능 스케줄 예시



병행 제어

❖ 병행 제어 기법

- 의미
 - 비직렬 스케줄로 병행 수행하면서도 직렬 가능성을 보장하기 위한 기법
- 방법
 - 모든 트랜잭션이 준수하면 직렬 가능성이 보장되는 규약을 정의하고, 트랜잭션들이 이 규약을 따르도록 함
- 대표적인 병행 제어 기법
 - 로킹 기법

병행 제어

❖ 로킹(locking) 기법

■ 기본 원리

- 한 트랜잭션이 먼저 접근한 데이터에 대한 연산을 끝낼 때까지는 다른 트랜잭션이 그 데이터에 접근하지 못하도록 상호 배제(mutual exclusion)함

■ 방법

- 병행 수행되는 트랜잭션들이 같은 데이터에 동시에 접근하지 못하도록 lock과 unlock 연산을 이용해 제어
 - lock : 트랜잭션이 데이터에 대한 독점권을 요청하는 연산
 - unlock : 트랜잭션이 데이터에 대한 독점권을 반환하는 연산

병행 제어

❖ 로킹 기법

■ 기본 로킹 규약

- 트랜잭션은 데이터에 접근하기 위해 먼저 lock 연산을 실행해 독점권을 획득함
 - read 또는 write 연산을 실행하기 전 lock 연산을 실행함
- 다른 트랜잭션에 의해 이미 lock 연산이 실행된 데이터에는 다시 lock 연산을 실행할 수 없음
- 독점권을 획득한 데이터에 대한 모든 연산의 수행이 끝나면 트랜잭션은 unlock 연산을 실행해서 독점권을 반납해야 함

병행 제어

❖ 로킹 기법

■ 로킹 단위

- lock 연산을 실행하는 대상 데이터의 크기
- 전체 데이터베이스부터 릴레이션, 튜플, 속성까지도 가능함
- 로킹 단위가 커질수록 병행성은 낮아지지만 제어가 쉬움
- 로킹 단위가 작아질수록 제어가 어렵지만 병행성은 높아짐

병행 제어

❖ 기본 로킹 규약의 효율성을 높이기 위한 방법

- 트랜잭션들이 같은 데이터에 동시에 read 연산을 실행하는 것을 허용
 - lock 연산을 두 가지 종류로 구분하여 사용

연산	설명
공용 _{shared} lock	트랜잭션이 데이터에 대해 공용 lock 연산을 실행하면, 해당 데이터에 read 연산을 실행할 수 있지만 write 연산은 실행할 수 없다. 그리고 해당 데이터에 다른 트랜잭션도 공용 lock 연산을 동시에 실행할 수 있다. (데이터에 대한 사용권을 여러 트랜잭션이 함께 가질 수 있음)
전용 _{exclusive} lock	트랜잭션이 데이터에 전용 lock 연산을 실행하면 해당 데이터에 read 연산과 write 연산을 모두 실행할 수 있다. 그러나 해당 데이터에 다른 트랜잭션은 공용이든 전용이든 어떤 lock 연산도 실행할 수 없다. (전용 lock 연산을 실행한 트랜잭션만 해당 데이터에 대한 독점권을 가질 수 있음)

병행 제어

❖ Lock 연산의 양립성

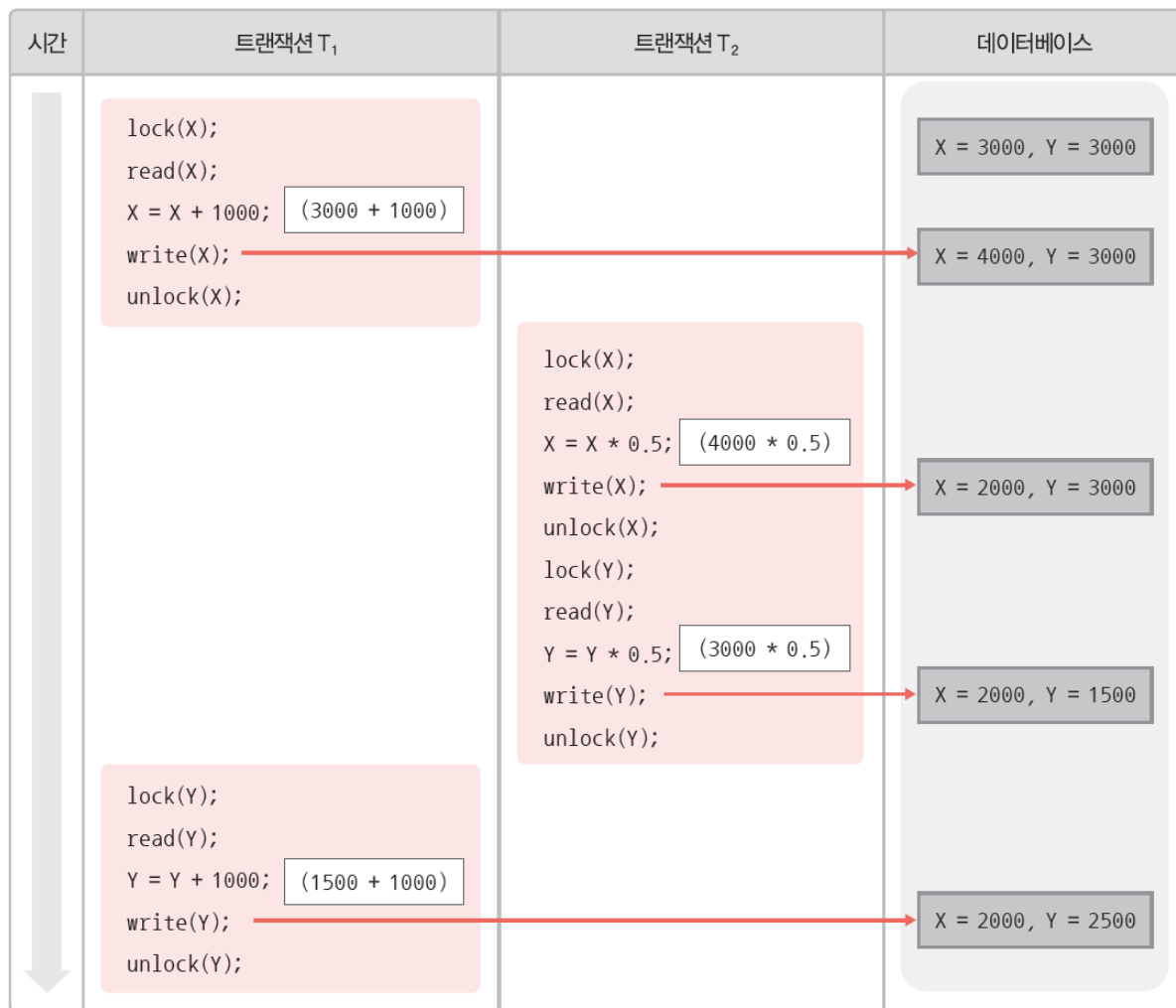
- 기본 로킹 규약의 효율성을 높이기 위한 방법

	공용 lock	전용 lock
공용 lock	가능	불가능
전용 lock	불가능	불가능

- 서로 다른 트랜잭션이 같은 데이터에 공용 lock 연산을 동시에 실행할 수 있음
- 다른 트랜잭션이 전용 lock 연산을 실행한 데이터에는 공용 lock, 전용 lock 모두 실행 불가

병행 제어

❖ 기본 로킹 규약으로 직렬 가능성이 보장되지 않는 스케줄 예



트랜잭션 T_1 이 데이터 X에
너무 빨리 unlock 연산을
실행하여 트랜잭션 T_2 가
일관성 없는 데이터에
접근했기 때문

[해결책: 2단계 로킹 규약]

병행 제어

❖ 2단계 로킹 규약(2PLP; 2 Phase Locking Protocol)

■ 의미

- 기본 로킹 규약의 문제를 해결하고 트랜잭션의 직렬 가능성을 보장하기 위해 lock과 unlock 연산의 수행 시점에 대한 새로운 규약을 추가한 것

■ 방법

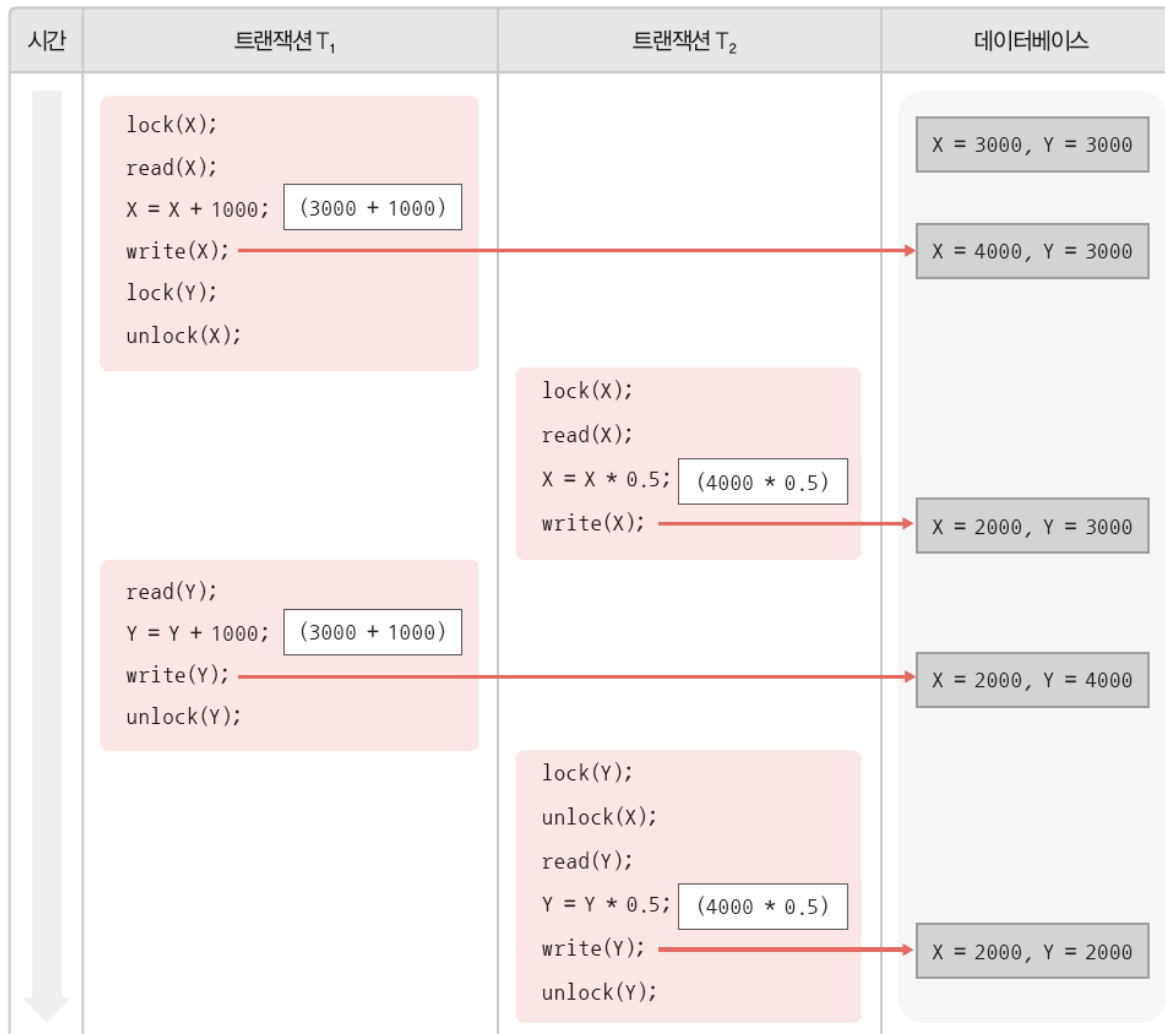
- 트랜잭션이 lock과 unlock 연산을 확장 단계와 축소 단계로 나누어 실행

확장 단계	트랜잭션이 lock 연산만 실행할 수 있고, unlock 연산은 실행할 수 없는 단계
축소 단계	트랜잭션이 unlock 연산만 실행할 수 있고, lock 연산은 실행할 수 없는 단계

- 트랜잭션이 처음 수행되면 확장 단계로 들어가 lock 연산만 실행 가능
- unlock 연산을 실행하면 축소 단계로 들어가 unlock 연산만 실행 가능
- 트랜잭션은 첫 번째 unlock 연산 실행 전에 필요한 모든 lock 연산을 실행해야 함

병행 제어

❖ 2단계 로킹 규약



병행 제어

❖ 교착 상태(deadlock)

- 트랜잭션들이 상대가 독점하고 있는 데이터에 unlock 연산이 실행되기를 서로 기다리면서 트랜잭션의 수행을 중단하고 있는 상태
- 교착 상태가 발생하지 않도록 예방하거나, 발생 시 빨리 탐지하여 필요한 조치를 취해야 함