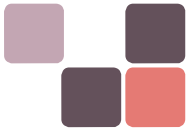
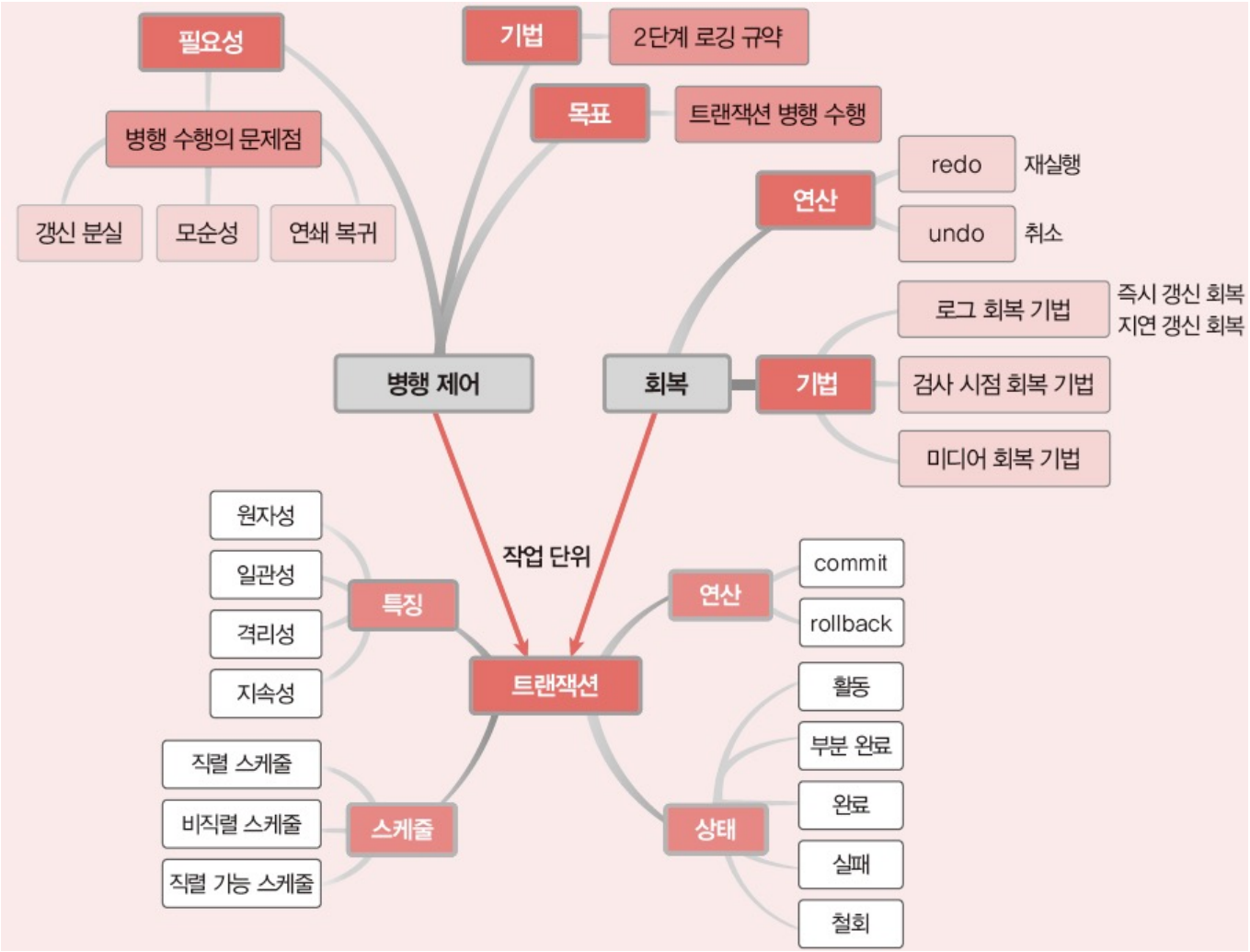


10장. 회복과 병행 제어

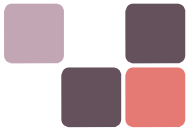
- 트랜잭션
- 장애와 회복
- 병행 제어

학습목표



- 병행 제어와 회복 작업의 기본 단위인 트랜잭션의 개념을 이해한다.
- 데이터베이스를 장애로부터 복구하는 다양한 회복 기법을 익힌다.
- 여러 사용자가 동시에 접근할 수 있도록 트랜잭션 수행을 통제하는 병행 제어 기법을 익힌다.

01 트랜잭션



❖ 트랜잭션(transaction)의 개념

- 하나의 작업을 수행하는데 필요한 데이터베이스 연산들을 모아놓은 것
- 작업 수행에 필요한 SQL 문들의 모임
  - 특히, 데이터베이스를 변경하는 INSERT, DELETE, UPDATE 문의 실행을 관리
- 논리적인 작업의 단위
- 장애 발생 시 복구 작업이나 병행 제어 작업을 위한 중요한 단위로 사용됨
- 데이터베이스의 무결성과 일관성을 보장하기 위해 작업 수행에 필요한 연산들을 하나의 트랜잭션으로 제대로 정의하고 관리해야 함

01 트랜잭션

❖ 트랜잭션의 예 (1)

처리 순서는 중요하지 않지만
두 개의 UPDATE 문이 모두 정상적으로
실행되어야 함

그림 10-1 트랜잭션의 예 1 : 계좌이체 트랜잭션

01 트랜잭션

❖ 트랜잭션의 예 (2)

INSERT 문과 UPDATE 문이 모두 정상적으로 실행되어야 상품주문 트랜잭션이 성공적으로 수행됨

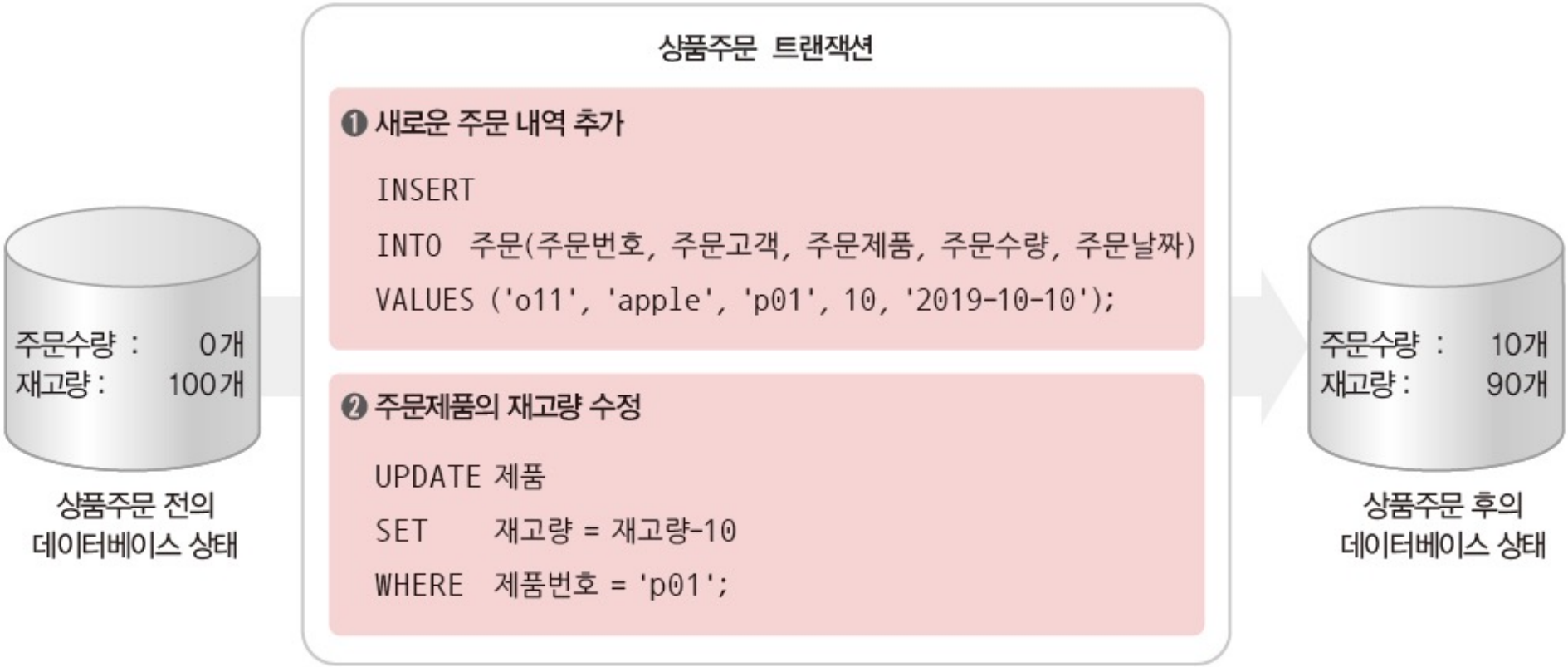


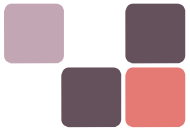
그림 10-2 트랜잭션의 예 2 : 상품주문 트랜잭션

01 트랜잭션

❖ 트랜잭션의 특성(ACID 특성)

그림 10-3 트랜잭션의 특성

01 트랜잭션



❖ 트랜잭션의 특성 – 원자성(atomicity)

- 트랜잭션의 연산들이 모두 정상적으로 실행되거나 하나도 실행되지 않아야 하는 all-or-nothing 방식을 의미
- 만약 트랜잭션 수행 도중 장애가 발생하면?
  - 지금까지 실행한 연산 처리를 모두 취소하고, 데이터베이스를 트랜잭션 작업 전 상태로 되돌려야 함
- 원자성의 보장을 위해 장애 발생 시 회복 기능이 필요

01 트랜잭션

❖ 트랜잭션의 특성 – 원자성

그림 10-4 트랜잭션의 수행 성공 예

01 트랜잭션

❖ 트랜잭션의 특성 – 원자성

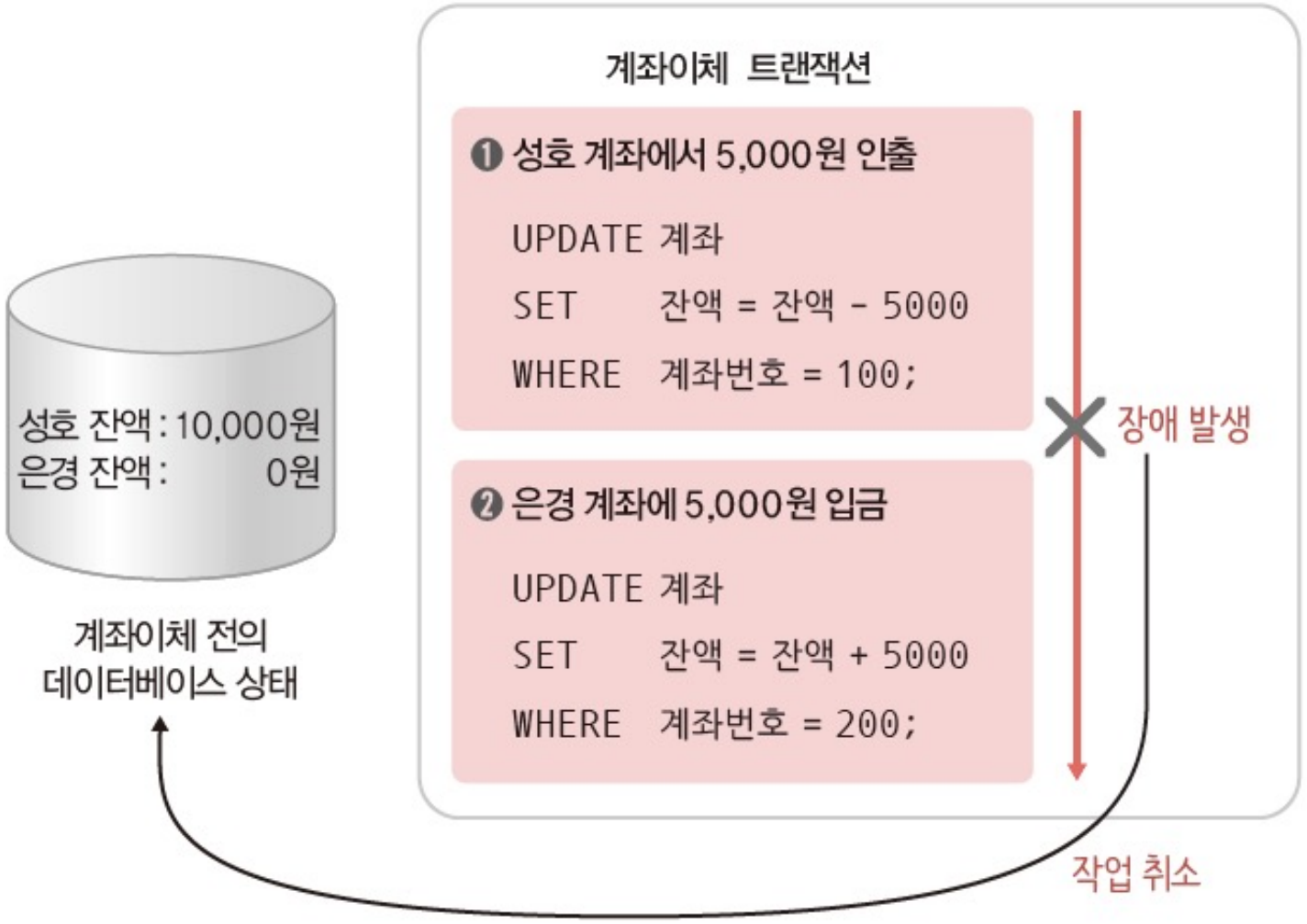


그림 10-5 트랜잭션의 수행 실패 예

01 트랜잭션

❖ 트랜잭션의 특성 – 일관성(consistency)

- 트랜잭션이 성공적으로 수행된 후에도 데이터베이스가 일관된 상태를 유지해야 함을 의미

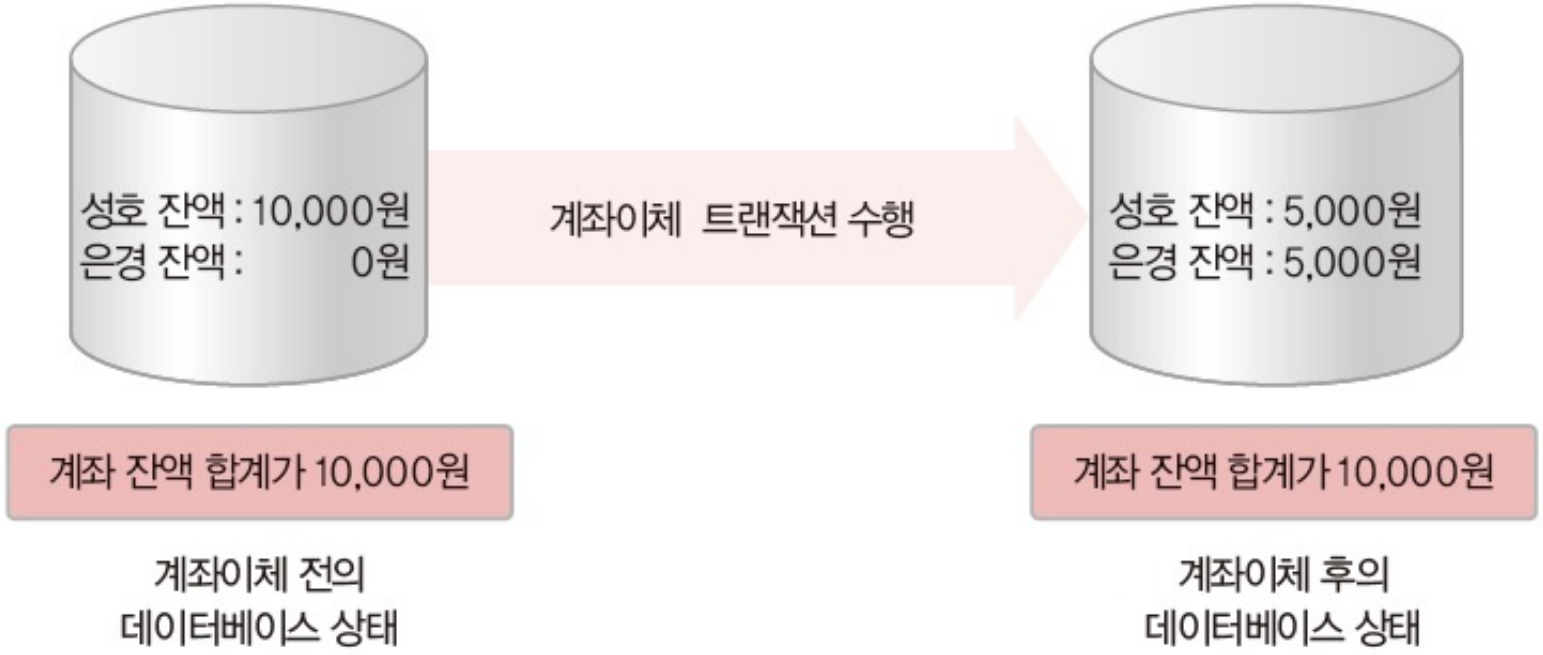


그림 10-6 트랜잭션의 일관성을 만족하는 예

01 트랜잭션

❖ 트랜잭션의 특성 – 일관성

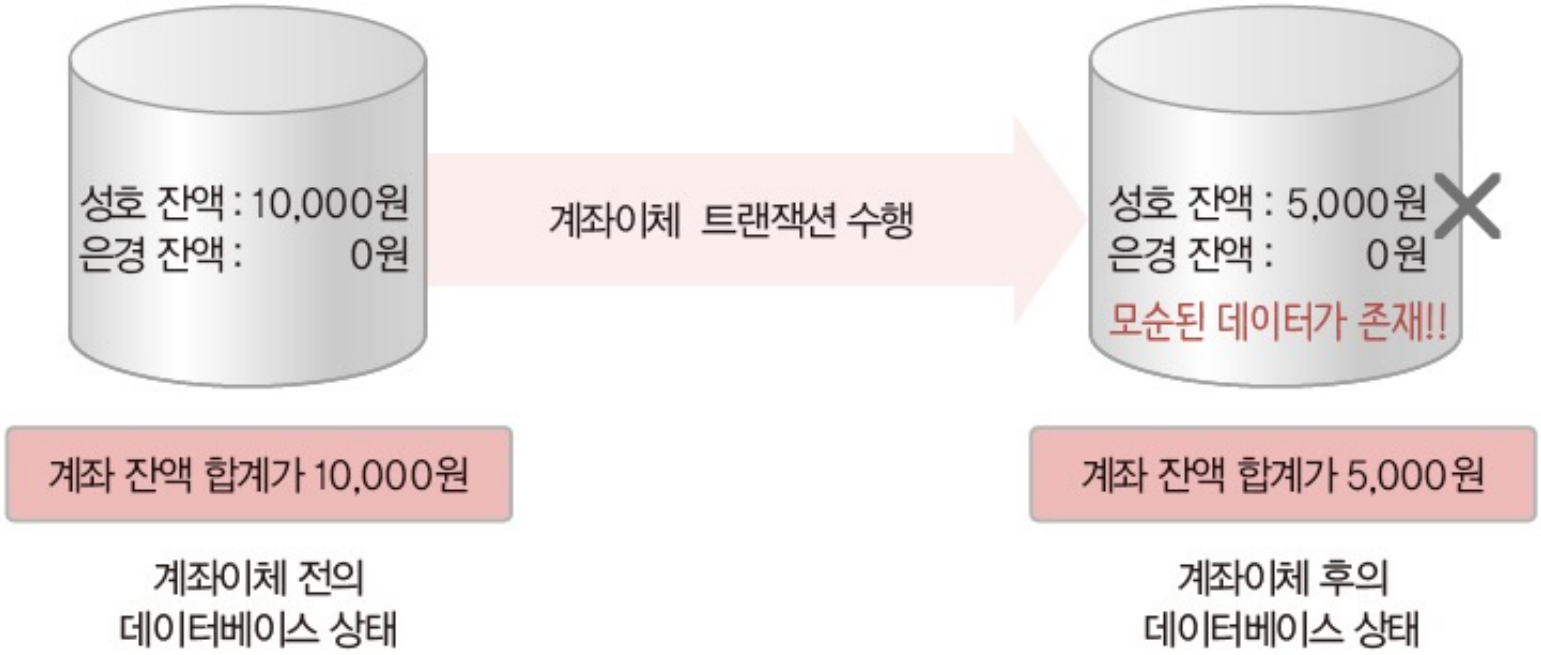
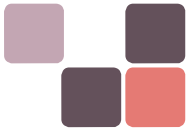


그림 10-7 트랜잭션의 일관성을 만족하지 않는 예

01 트랜잭션



❖ 트랜잭션의 특성 – 격리성(isolation)

- 수행 중인 트랜잭션이 완료될 때까지 다른 트랜잭션들이 중간 연산 결과에 접근할 수 없음을 의미
- 격리성의 보장을 위해서는 여러 트랜잭션이 동시에 수행되더라도 마치 순서대로 하나씩 수행되는 것처럼 정확하고 일관된 결과를 얻을 수 있도록 제어하는 기능이 필요

01 트랜잭션

❖ 트랜잭션의 특성 – 격리성

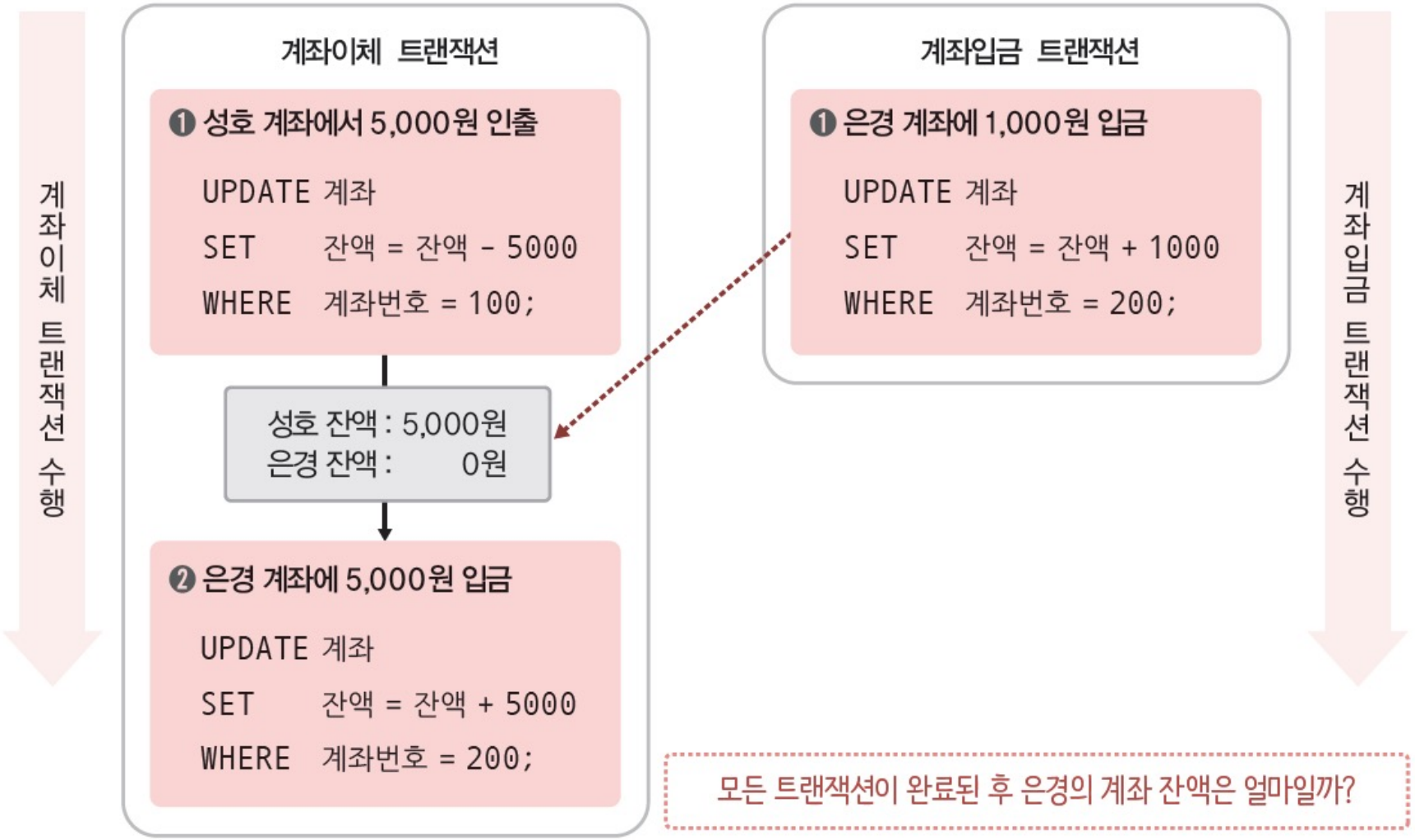


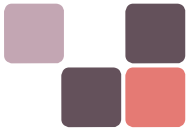
그림 10-8 트랜잭션의 격리성이 만족되지 않는 예

01 트랜잭션

❖ 트랜잭션의 특성 – 격리성

그림 10-9 트랜잭션의 격리성이 만족되는 예

01 트랜잭션



❖ 트랜잭션의 특성 – 지속성(durability)

- 트랜잭션이 성공적으로 완료된 후 데이터베이스에 반영한 수행 결과는 영구적이어야 함을 의미
- 지속성의 보장을 위해서는 장애 발생 시 회복 기능이 필요

01 트랜잭션

❖ 트랜잭션의 4가지 특성을 보장하기 위해 필요한 기능

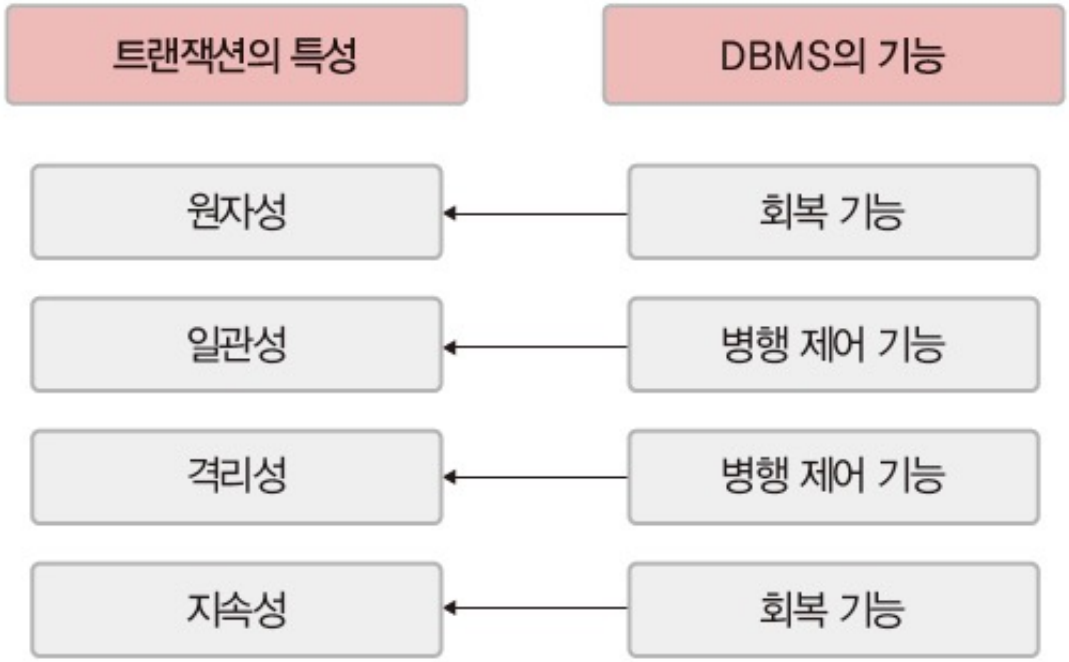


그림 10-10 트랜잭션의 특성과 DBMS의 기능

01 트랜잭션

❖ 트랜잭션의 주요 연산

그림 10-11 트랜잭션의 연산

01 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

그림 10-12 commit 연산을 실행한 예

01 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

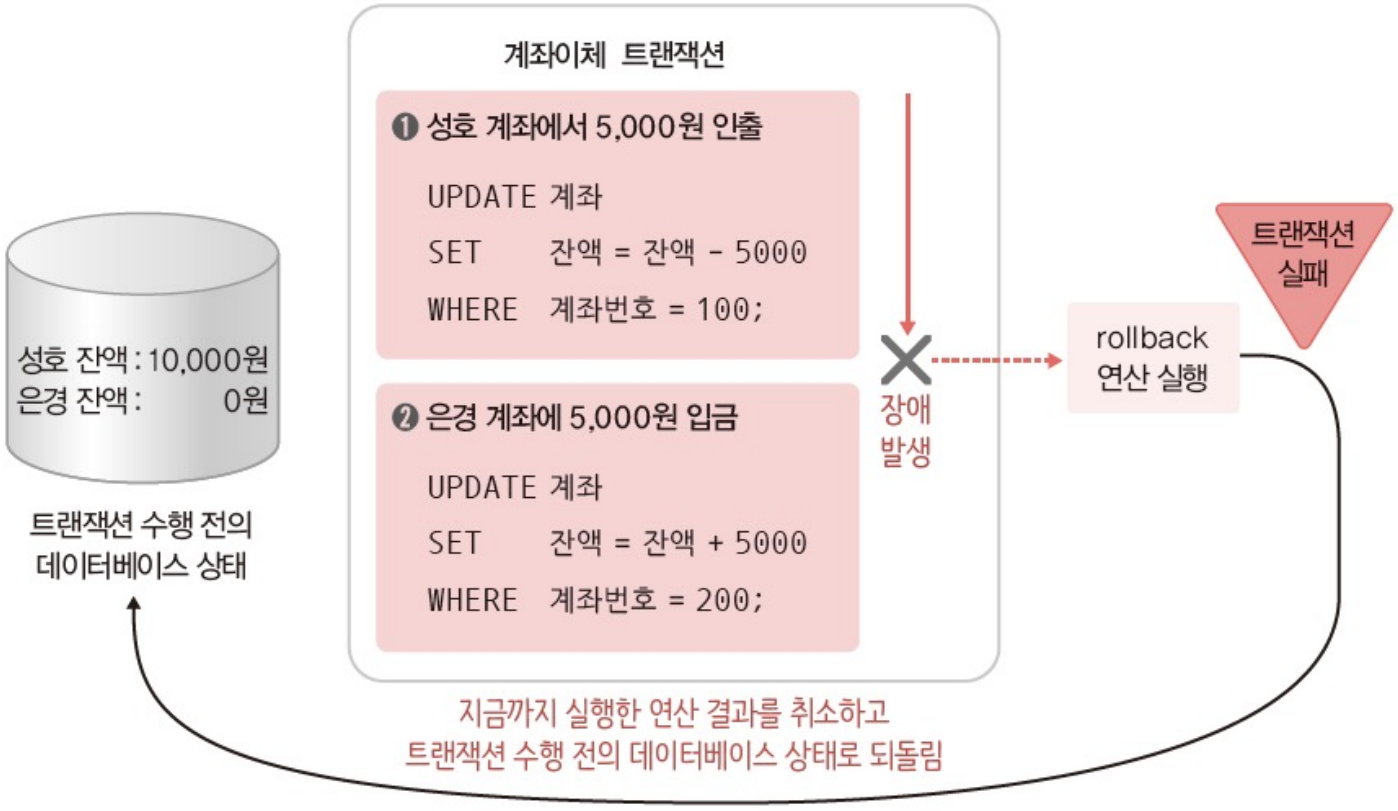


그림 10-13 rollback 연산을 실행한 예

01 트랜잭션

❖ 트랜잭션의 상태

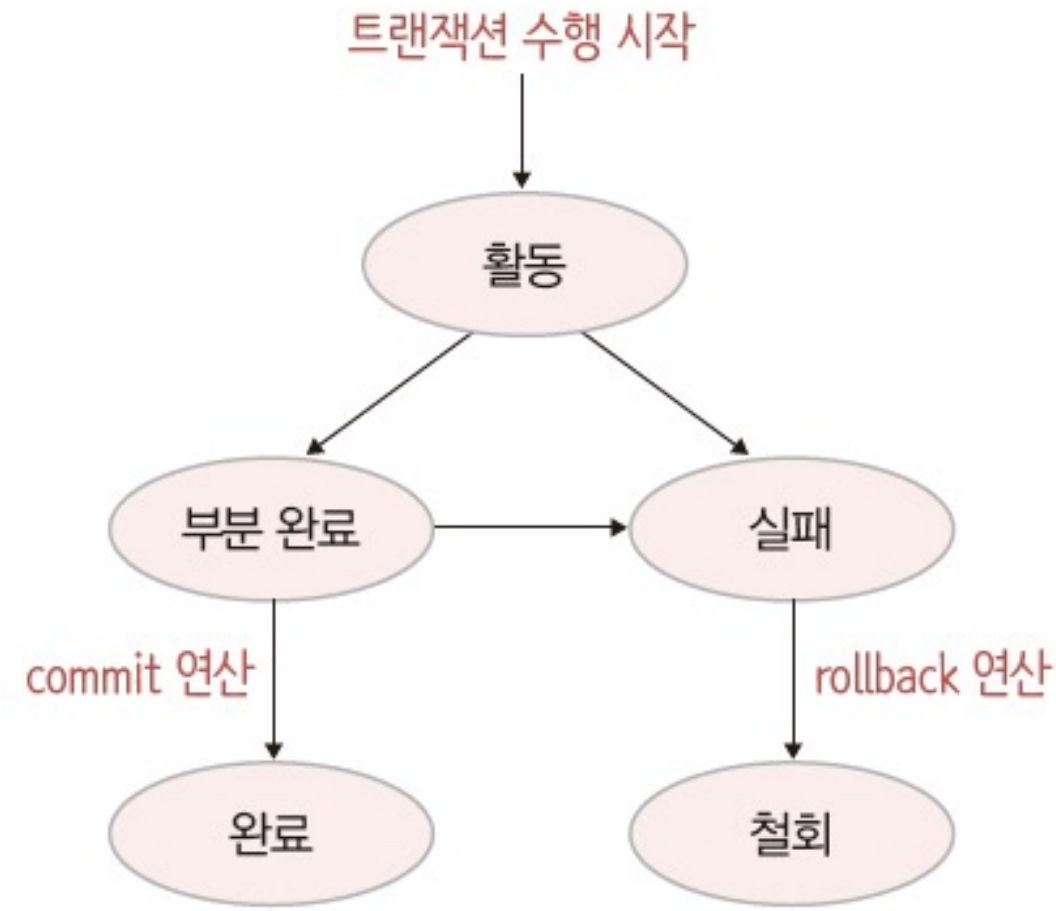
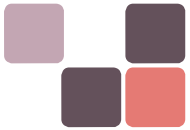


그림 10-14 트랜잭션의 다섯 가지 상태

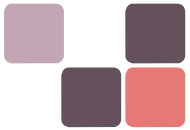
01 트랜잭션



❖ 트랜잭션의 상태

- 활동(active) 상태
  - 트랜잭션이 수행되기 시작하여 현재 수행 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

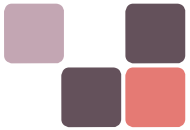
01 트랜잭션



❖ 트랜잭션의 상태

- 실패(failed) 상태
  - 장애가 발생하여 트랜잭션의 수행이 중단된 상태
- 철회(aborted) 상태
  - 트랜잭션의 수행 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

02 장애와 회복

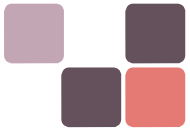


❖ 장애(failure)

- 시스템이 제대로 동작하지 않는 상태

표 10-1 장애의 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



❖ 데이터베이스를 저장하는 저장 장치의 종류

표 10-2 저장 장치의 종류

저장 장치	설명	
휘발성volatile 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
안정stable 저장 장치	의미	비휘발성 저장 장치를 이용해 데이터 복사본 여러 개를 만드는 방법으로, 어떤 장애가 발생해도 데이터가 손실되지 않고 데이터를 영구적으로 저장할 수 있음

02 장애와 회복

❖ 트랜잭션의 수행을 위해 필요한 데이터 이동 연산

- 디스크와 메인 메인 메모리 간의 데이터 이동 연산 : input / output
- 메인 메모리와 프로그램 변수 간의 데이터 이동 연산 : read / write

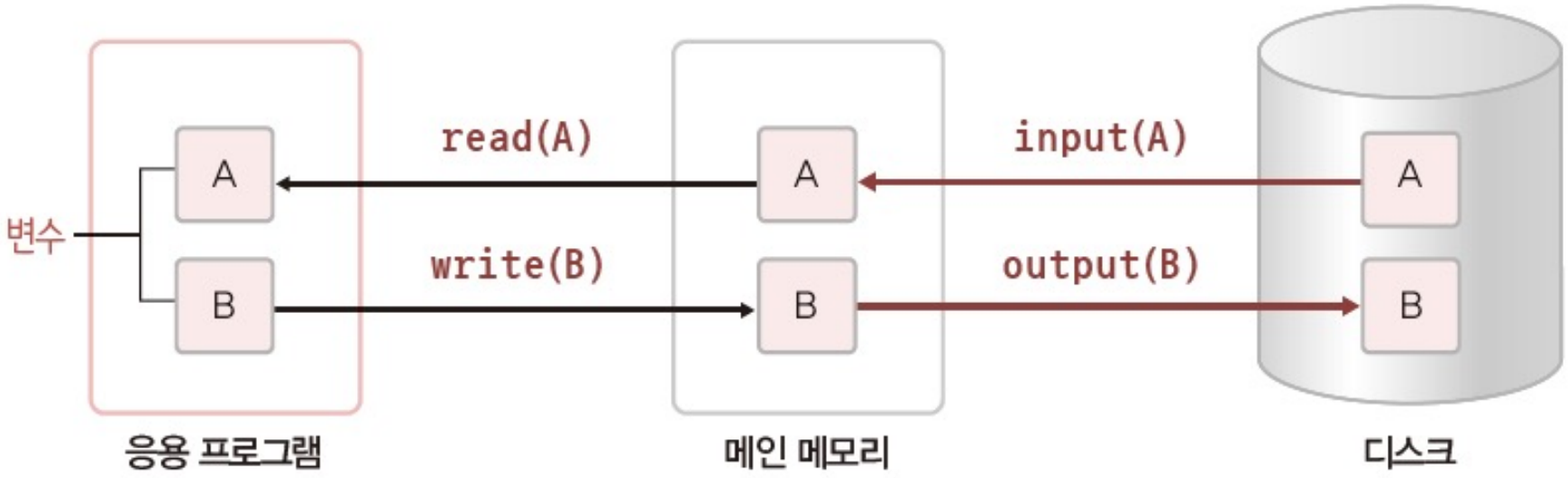


그림 10-18 응용 프로그램이 실행한 트랜잭션의 수행을 위해 필요한 데이터 이동 연산

02 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

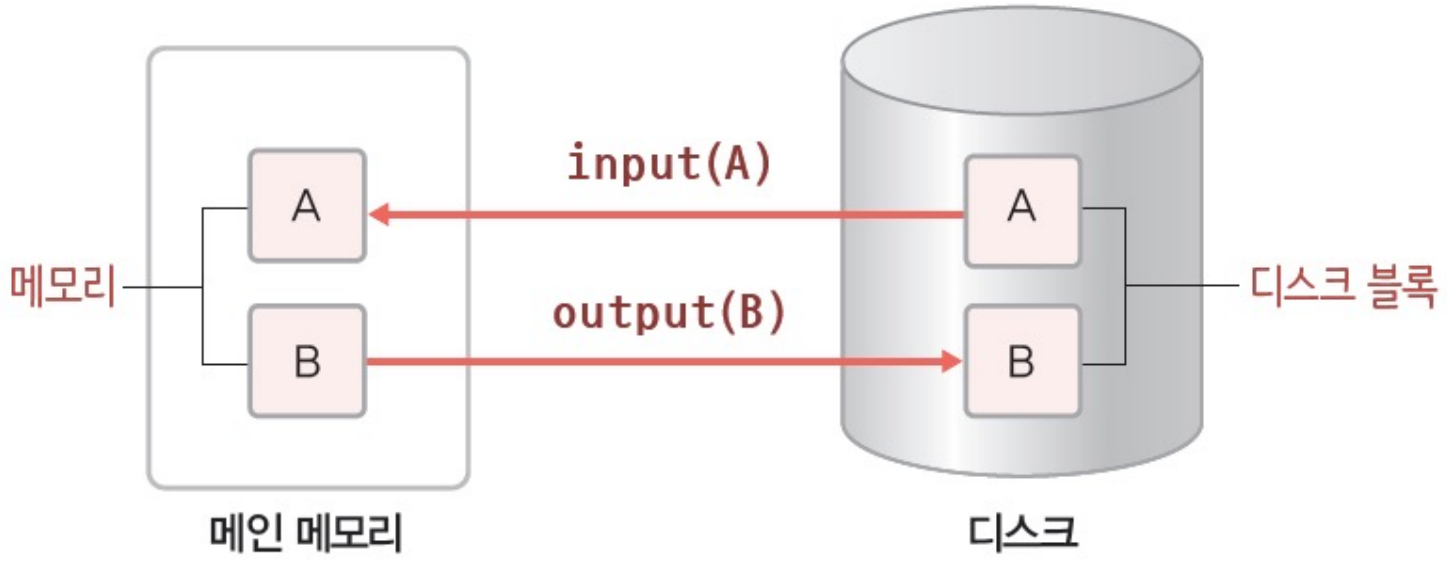
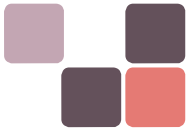


그림 10-15 디스크와 메인 메모리 간의 데이터 이동

02 장애와 회복



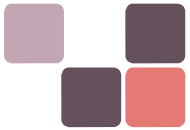
❖ 디스크와 메인 메모리 간의 데이터 이동 연산

- input 연산 / output 연산
- 블록(block) 단위로 수행됨
  - 디스크 블록 : 디스크에 있는 블록
  - 버퍼 블록 : 메인 메모리에 있는 블록

input(X)	디스크 블록에 저장되어 있는 데이터 X를 메인 메모리 버퍼 블록으로 이동시키는 연산
output(X)	메인 메모리 버퍼 블록에 있는 데이터 X를 디스크 블록으로 이동시키는 연산

그림 10-16 디스크와 메인 메모리 간의 데이터 이동 연산

02 장애와 회복



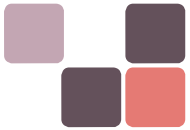
❖ 메인 메모리와 변수 간의 데이터 이동 연산

- read 연산 / write 연산
- 응용 프로그램에서 트랜잭션의 수행을 지시하면 메인 메모리 버퍼 블록에 있는 데이터를 프로그램의 변수로 가져오고, 데이터 처리 결과를 저장한 변수 값을 메인 메모리 버퍼 블록으로 옮기는 작업이 필요

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

그림 10-17 메인 메모리의 버퍼 블록과 프로그램 변수 간의 데이터 이동 연산

02 장애와 회복



❖ 트랜잭션을 데이터 이동 연산을 포함한 프로그램으로 표현한 예

계좌이체 트랜잭션

❶ 성호 계좌에서 5,000원 인출

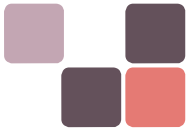
```
read(X);
X = X - 5000;
write(X);
```

❷ 은경 계좌에 5,000원 입금

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

그림 10-19 데이터 이동 연산을 포함한 계좌이체 트랜잭션 표현의 예

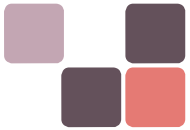
02 장애와 회복



❖ 회복(recovery)

- 장애가 발생했을 때 데이터베이스를 장애가 발생하기 전의 일관된 상태로 복구시키는 것
- 트랜잭션의 특성을 보장하고, 데이터베이스를 일관된 상태로 유지하기 위해 필수적인 기능
- 회복 관리자(recovery manager)가 담당
  - 장애 발생을 탐지하고, 장애가 탐지되면 데이터베이스 복구 기능을 제공

02 장애와 회복



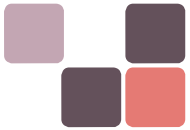
❖ 회복을 위해 데이터베이스 복사본을 만드는 방법

- 데이터베이스 회복의 핵심 원리는 데이터 중복!

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

그림 10-20 데이터베이스 회복을 위해 복사본을 만드는 방법

02 장애와 회복

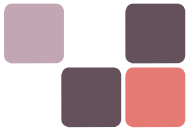


❖ 회복을 위한 기본 연산

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 (전반적으로 손상된 경우에 주로 사용)
undo (취소)	로그를 이용해 지금까지 실행된 모든 변경 연산을 취소하여 데이터베이스를 원래의 상태로 복구 (변경 중이었거나 이미 변경된 내용만 신뢰성을 잃은 경우에 주로 사용)

그림 10-21 회복 연산

02 장애와 회복



❖ 로그 파일

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

표 10-3 로그 레코드의 종류

로그 레코드	설명	
⟨T_i, start⟩	의미	트랜잭션 T_i가 수행을 시작했음을 기록
	예	⟨T_1, start⟩
⟨T_i, X, old_value, new_value⟩	의미	트랜잭션 T_i가 데이터 X를 이전 값$_{old_value}$에서 새로운 값$_{new_value}$으로 변경하는 연산을 실행했음을 기록
	예	⟨T_1, X, 10000, 5000⟩
⟨T_i, commit⟩	의미	트랜잭션 T_i가 성공적으로 완료되었음을 기록
	예	⟨T_1, commit⟩
⟨T_i, abort⟩	의미	트랜잭션 T_i가 철회되었음을 기록
	예	⟨T_1, abort⟩

02 장애와 회복

❖ 로그 기록 예

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

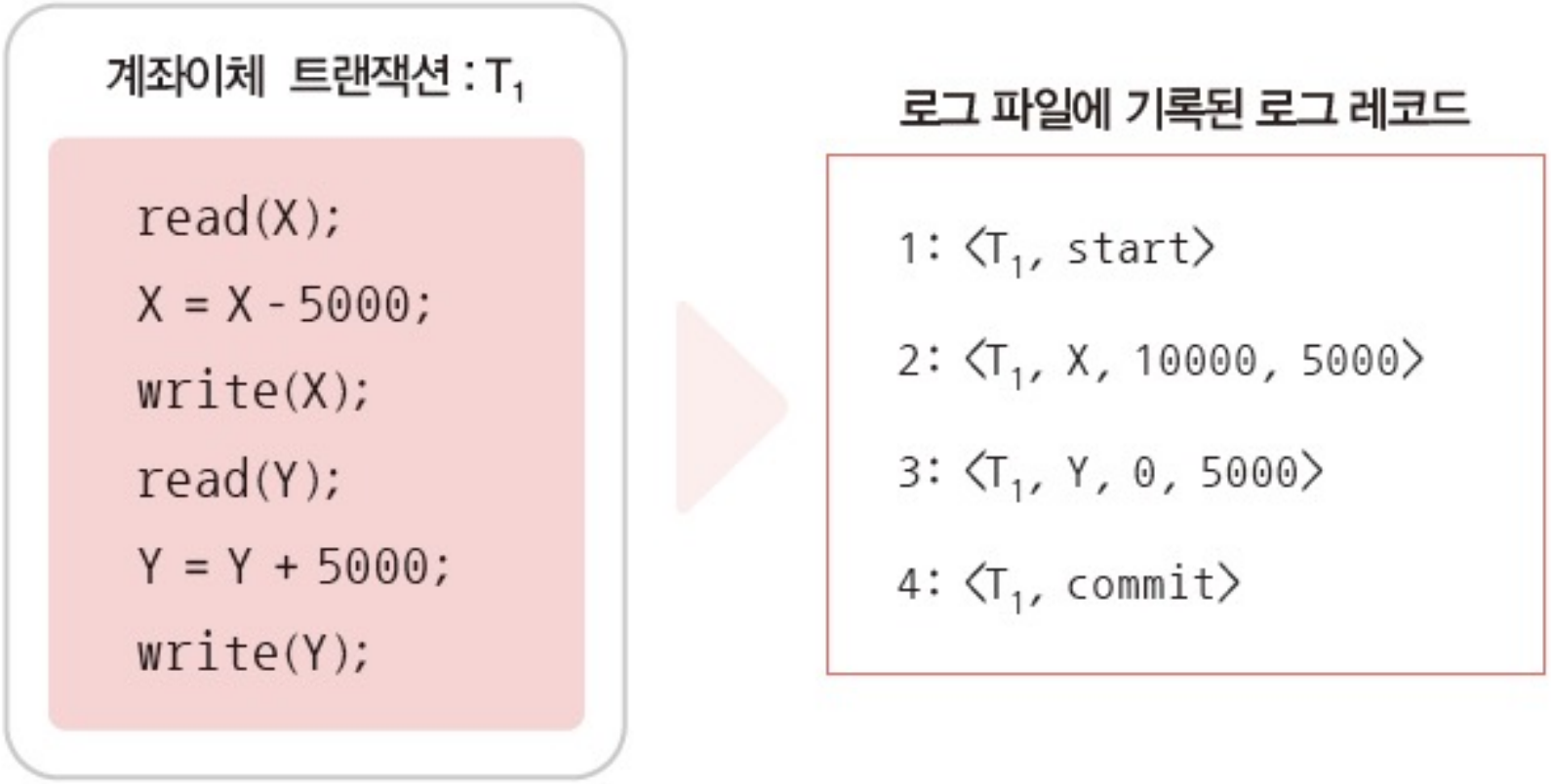


그림 10-22 계좌이체 트랜잭션이 수행되면서 기록된 로그의 예

02 장애와 회복

❖ 회복 기법

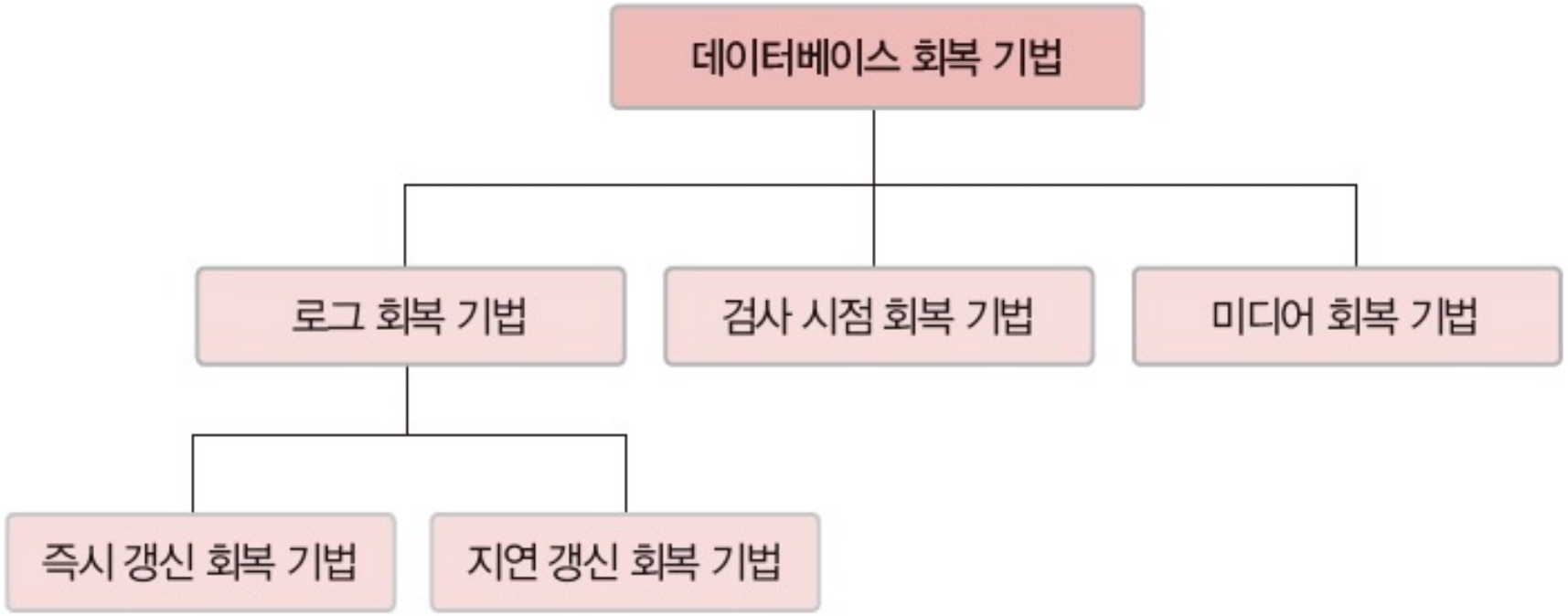
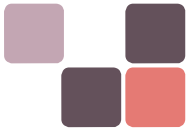


그림 10-23 데이터베이스 회복 기법의 분류

02 장애와 회복



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

- 트랜잭션 수행 중에 데이터 변경 연산의 결과를 데이터베이스에 즉시 반영
- 장애 발생에 대비하기 위해 데이터 변경에 대한 내용을 로그 파일에 기록
  - 데이터 변경 연산이 실행되면, 로그 파일에 로그 레코드를 먼저 기록한 다음 데이터베이스에 변경 연산을 반영
- 장애 발생 시점에 따라 redo나 undo 연산을 실행해 데이터베이스를 복구

02 장애와 회복

❖ 로그 회복 기법 – 즉시 갱신 회복 기법

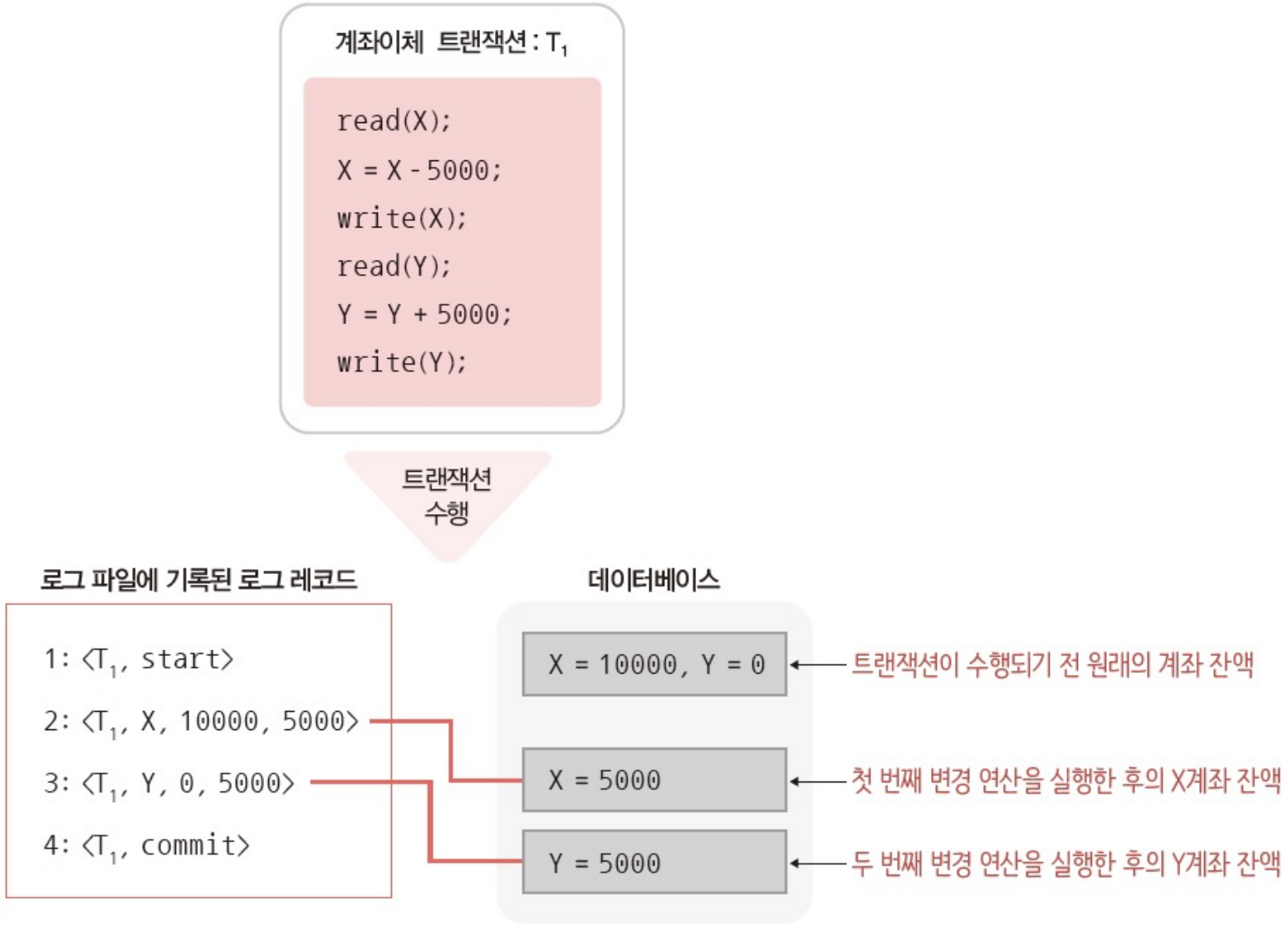


그림 10-24 계좌이체 트랜잭션 수행 중 로그 작성 및 데이터베이스 반영 순서

02 장애와 회복

❖ 로그 회복 기법 – 즉시 갱신 회복 기법

트랜잭션이 완료되기 전 장애가 발생한 경우 로그 파일에 $\langle T_i, start \rangle$ 로그 레코드는 존재하지만 $\langle T_i, commit \rangle$ 로그 레코드는 존재하지 않는 상태	undo 연산
트랜잭션이 완료된 후 장애가 발생한 경우 로그 파일에 $\langle T_i, start \rangle$ 로그 레코드와 $\langle T_i, commit \rangle$ 로그 레코드가 모두 존재하는 상태	redo 연산

그림 10-25 즉시 갱신 회복 기법의 데이터베이스 회복 전략

02 장애와 회복

❖ 로그 회복 기법 – 즉시 갱신 회복 기법 적용 예

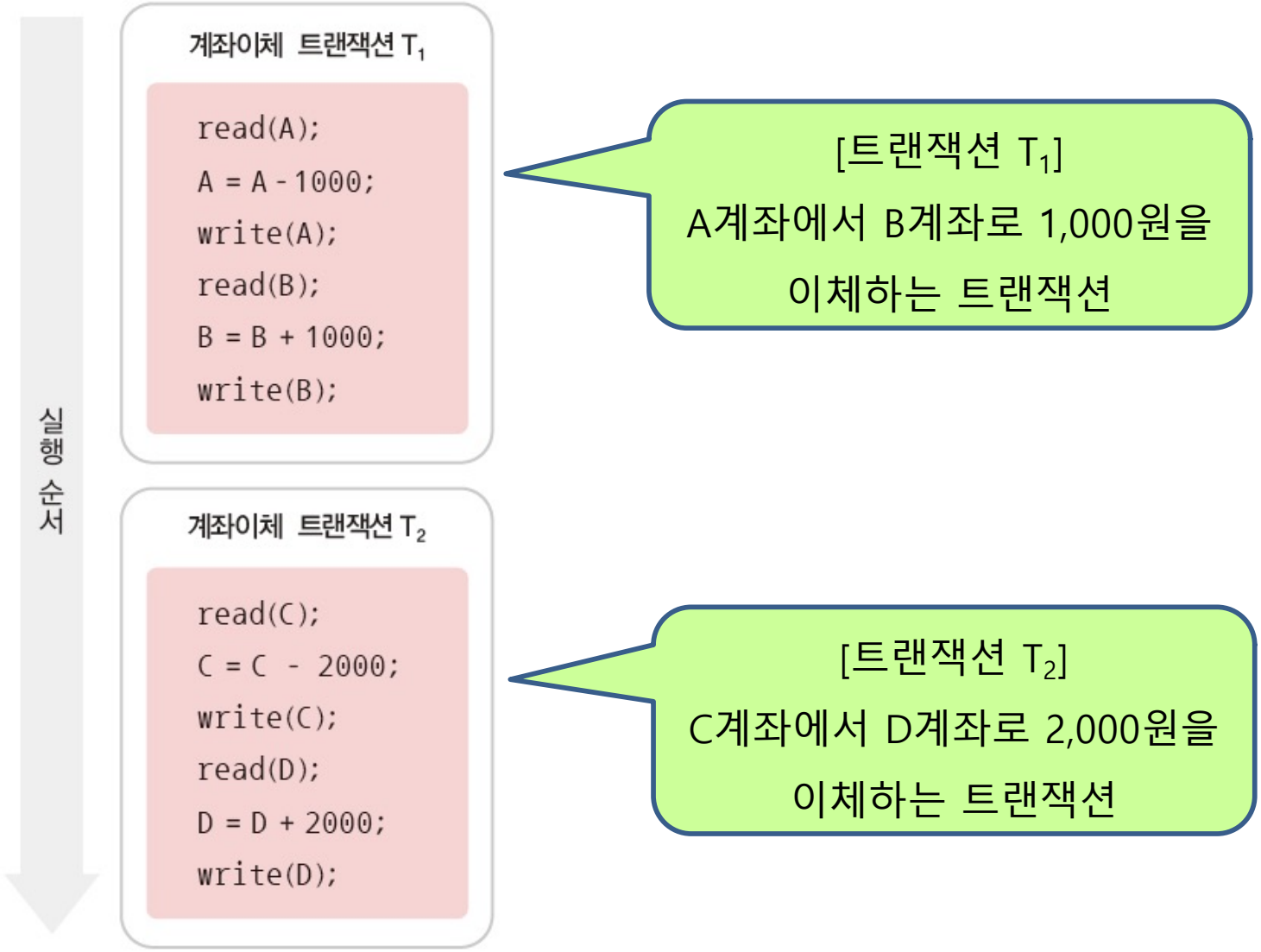


그림 10-26 순차적으로 수행되는 두 트랜잭션의 예

02 장애와 회복

❖ 로그 회복 기법 – 즉시 갱신 회복 기법 적용 예

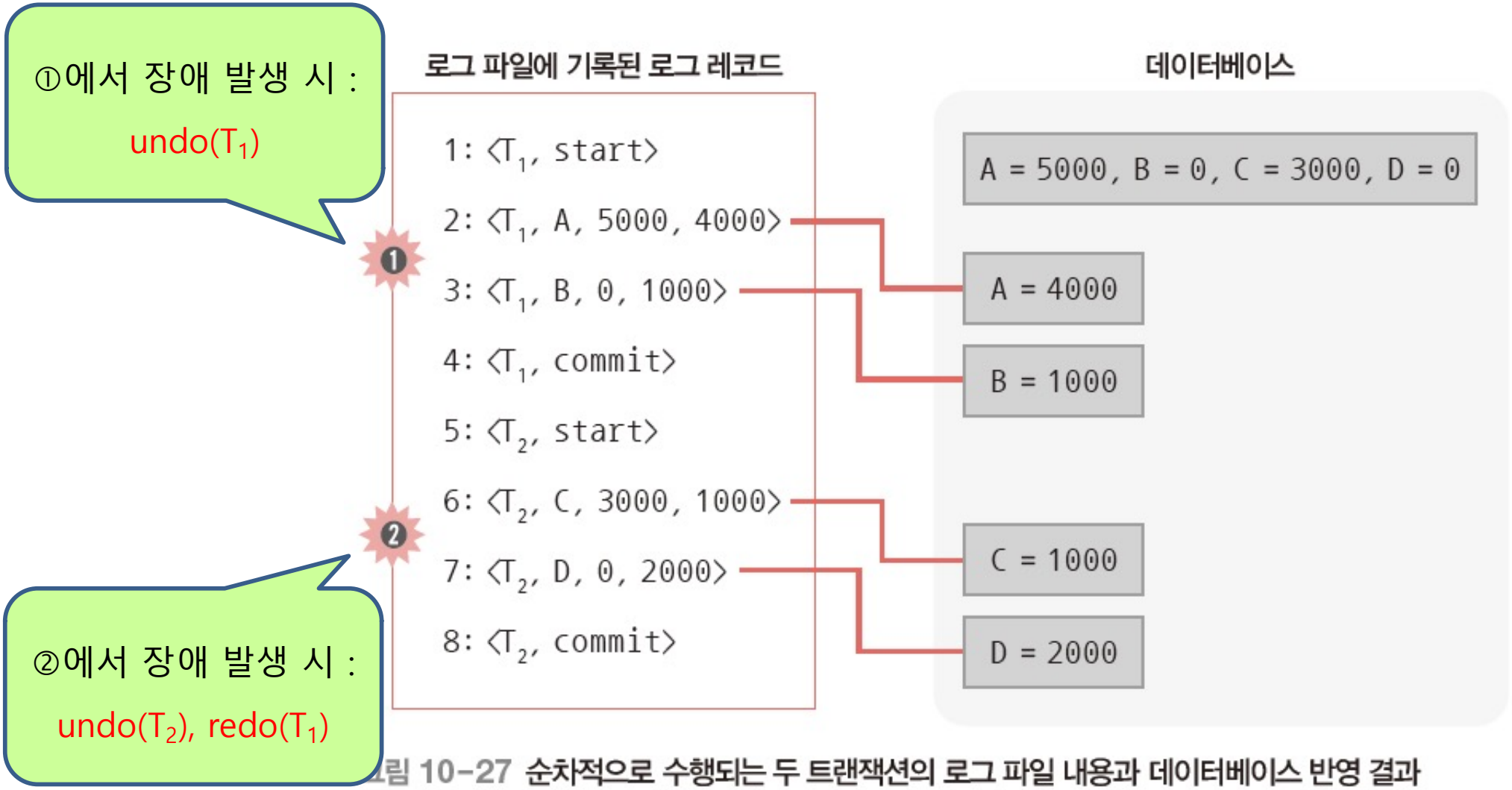
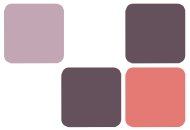


그림 10-27 순차적으로 수행되는 두 트랜잭션의 로그 파일 내용과 데이터베이스 반영 결과

02 장애와 회복



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

- 트랜잭션 수행 중에 데이터 변경 연산의 결과를 로그에만 기록해두고, 트랜잭션이 부분 완료된 후에 로그에 기록된 내용을 이용해 데이터베이스에 한번에 반영
- 트랜잭션 수행 중에 장애가 발생할 경우 로그에 기록된 내용을 버리기만 하면 데이터베이스가 원래 상태를 그대로 유지하게 됨
  - undo 연산은 필요없고 redo 연산만 사용
  - 로그 레코드에는 변경 이후 값만 기록하면 됨 : $<T_1, X, new\_value>$ 형식

02 장애와 회복

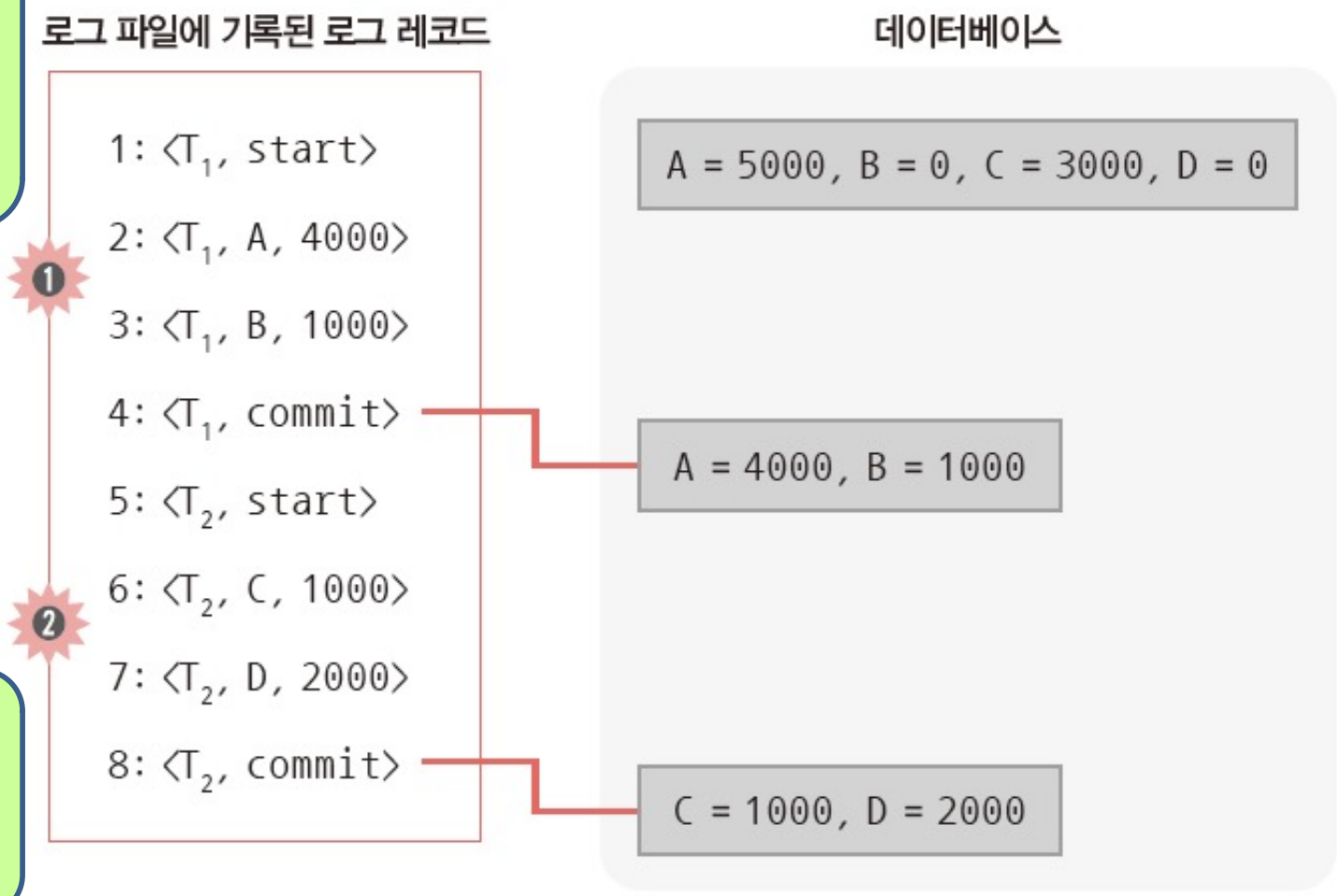
❖ 로그 회복 기법 – 지연 갱신 회복 기법

그림 10-28 지연 갱신 회복 기법의 데이터베이스 회복 전략

02 장애와 회복

❖ 로그 회복 기법 – 지연 갱신 회복 기법 적용 예

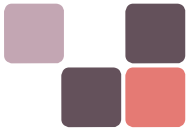
①에서 장애 발생 시 :
로그 기록을 무시하고
별다른 회복 조치를
하지 않음



②에서 장애 발생 시 :
redo(T_1)

그림 10-29 순차적으로 수행되는 두 트랜잭션의 로그 파일 내용과 데이터베이스 반영 결과

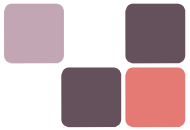
02 장애와 회복



❖ 검사 시점 회복 기법

- 로그 기록을 이용하되, 일정 시간 간격으로 검사 시점(checkpoint)을 만듦
  - 검사 시점이 되면 모든 로그 레코드를 로그 파일에 기록하고, 데이터 변경 내용을 데이터베이스에 반영한 후 검사 시점을 표시하는 <checkpoint L> 로그 레코드를 로그 파일에 기록함
    - <checkpoint L>에서 L은 현재 실행되고 있는 트랜잭션의 리스트
- 장애 발생 시 가장 최근 검사 시점 이후의 트랜잭션에만 회복 작업 수행
  - 가장 최근의 <checkpoint L> 로그 레코드 이후 기록에 대해서만 회복 작업 수행
    - 회복 작업은 즉시 갱신 회복 기법이나 지연 갱신 회복 기법을 이용해 수행
- 로그 전체를 대상으로 회복 기법을 적용할 때 발생할 수 있는 비효율성의 문제를 해결
  - 검사 시점으로 작업 범위가 정해지므로 불필요한 회복 작업이 없어 시간이 단축됨

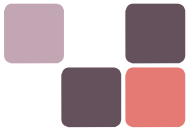
02 장애와 회복



❖ 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 발생하면?
  - 가장 최근에 복사해둔 덤프를 이용해 장애 발생 이전의 데이터베이스 상태로 복구하고 필요에 따라 redo 연산을 수행

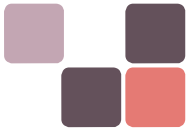
03 병행 제어



❖ 병행 수행과 병행 제어

- 병행 수행(concurrency)
  - 여러 사용자가 데이터베이스를 동시 공유할 수 있도록 여러 개의 트랜잭션을 동시에 수행하는 것을 의미
  - 여러 트랜잭션이 차례로 번갈아 수행되는 인터리빙(interleaving) 방식으로 진행됨
- 병행 제어(concurrency control) 또는 동시성 제어
  - 병행 수행 시 같은 데이터에 접근하여 연산을 실행해도 문제가 발생하지 않고 정확한 수행 결과를 얻을 수 있도록 트랜잭션의 수행을 제어하는 것을 의미

03 병행 제어



❖ 병행 수행 시 발생할 수 있는 문제점

- 갱신 분실(lost update)
  - 하나의 트랜잭션이 수행한 데이터 변경 연산의 결과를 다른 트랜잭션이 덮어써 변경 연산이 무효화되는 것
  - 여러 트랜잭션이 동시에 수행되더라도 갱신 분실 문제가 발생하지 않고 마치 트랜잭션들을 순차적으로 수행한 것과 같은 결과 값을 얻을 수 있어야 함

03 병행 제어

❖ 병행 수행 시 발생할 수 있는 문제점

- 갱신 분실

트랜잭션 T_1에 대해 갱신 분실이 발생함

(T_1의 변경 연산 결과가 데이터베이스에 반영되지 않음)

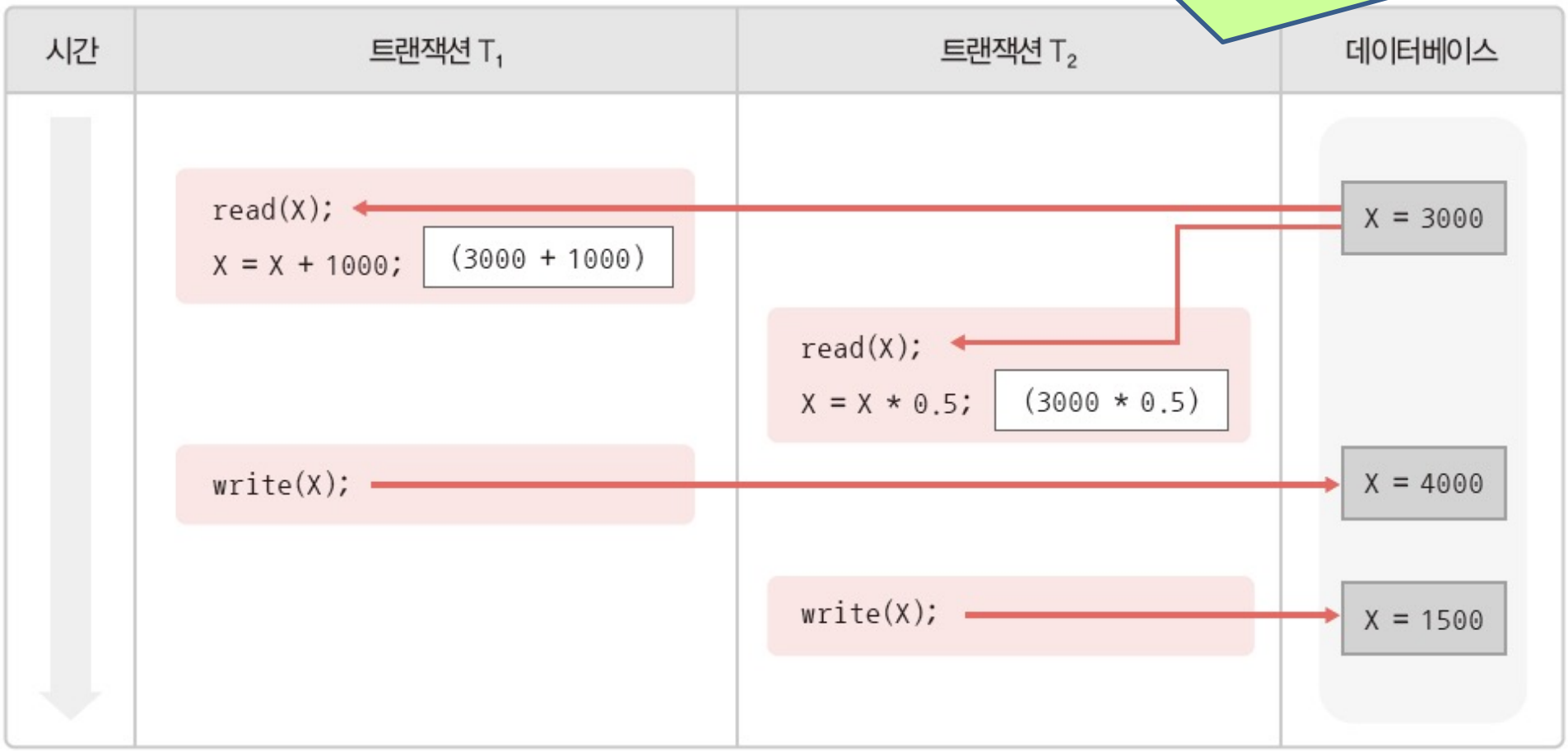


그림 10-30 두 트랜잭션의 병행 수행으로 발생한 갱신 분실의 예

03 병행 제어

❖ 병행 수행 시 발생할 수 있는 문제점

- 갱신 분실

트랜잭션을 순차적으로 수행해서
갱신 분실 문제가 발생하지 않는 경우

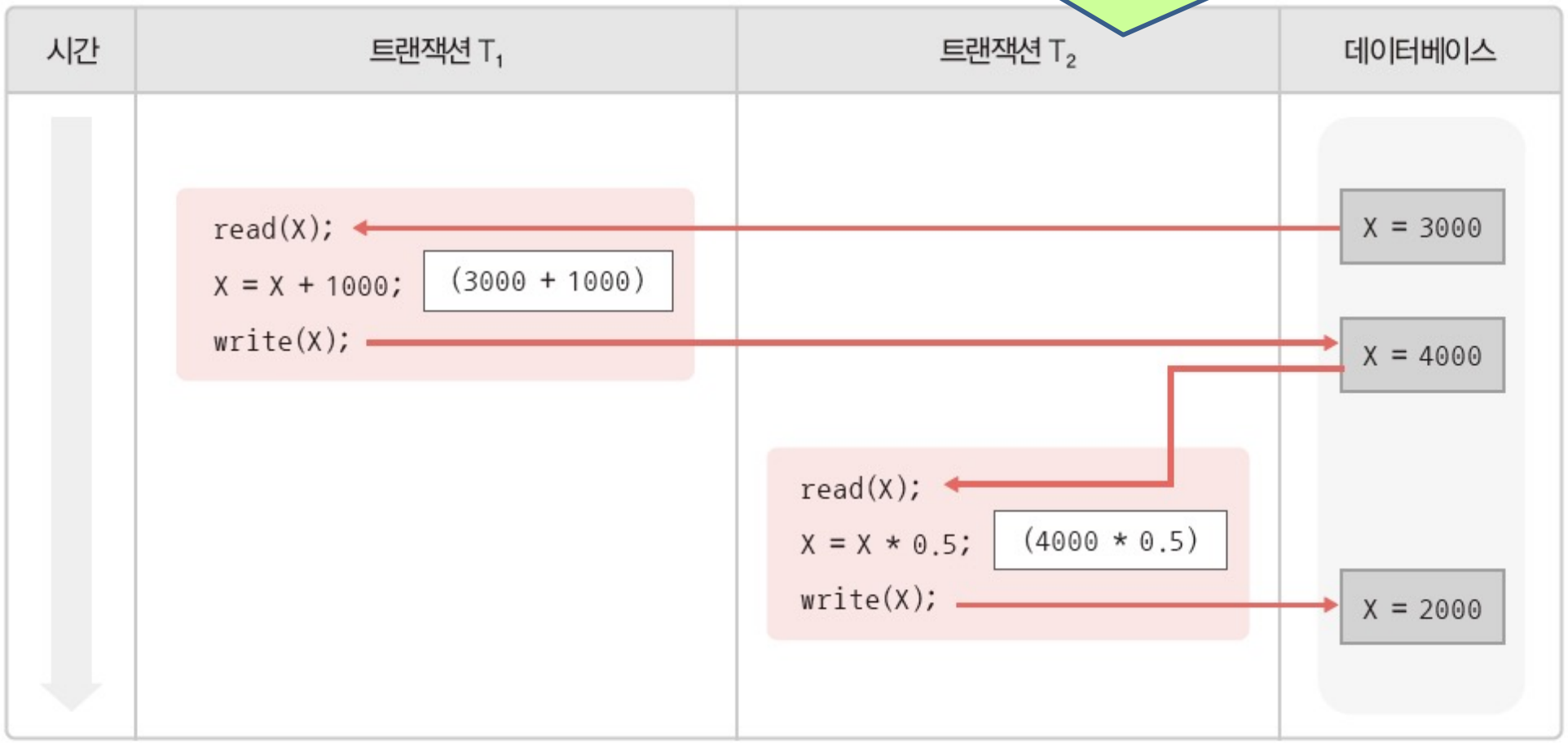
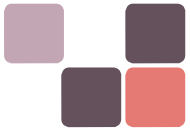


그림 10-31 트랜잭션 T_1을 수행한 후 트랜잭션 T_2를 수행한 결과

03 병행 제어



❖ 병행 수행 시 발생할 수 있는 문제점

- 모순성(inconsistency)
  - 하나의 트랜잭션이 여러 개 데이터 변경 연산을 실행할 때 일관성 없는 상태의 데이터베이스에서 데이터를 가져와 연산함으로써 모순된 결과가 발생하는 것
  - 여러 트랜잭션이 동시에 수행되더라도 모순성 문제가 발생하지 않고 마치 트랜잭션들을 순차적으로 수행한 것과 같은 결과 값을 얻을 수 있어야 함

03 병행 제어

❖ 병행 수행 시 발생할 수 있는 문제점

- 모순성

트랜잭션 T_1이 데이터 X와 Y를
서로 다른 상태의 데이터베이스에서
가져와 연산을 실행하는 모순이 발생

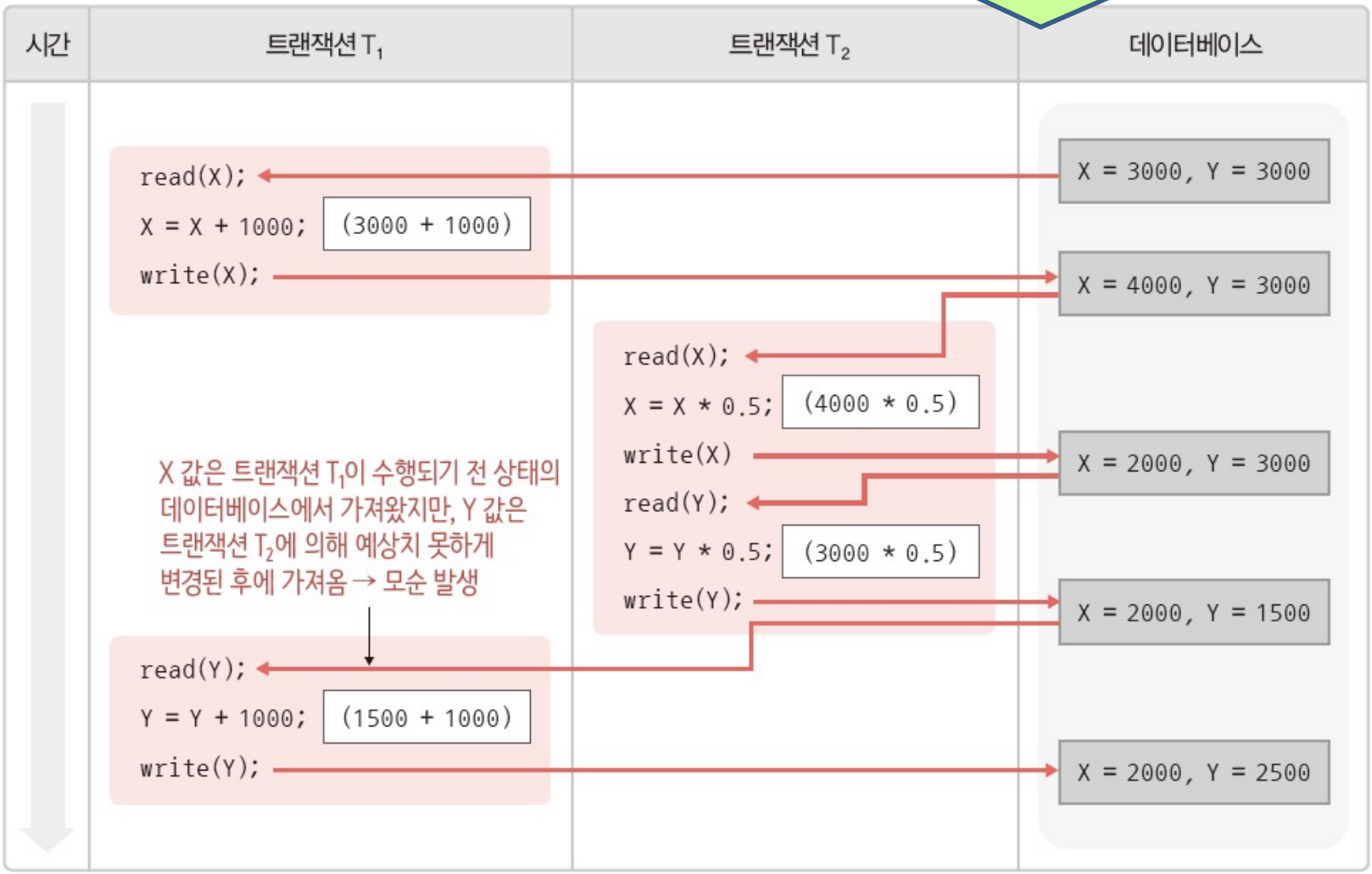


그림 10-32 두 트랜잭션의 병행 수행으로 발생한 모순성의 예

03 병행 제어

❖ 병행 수행 시 발생할 수 있는 문제점

- 모순성

트랜잭션을 순차적으로 수행해서 모순성 문제가 발생하지 않는 경우

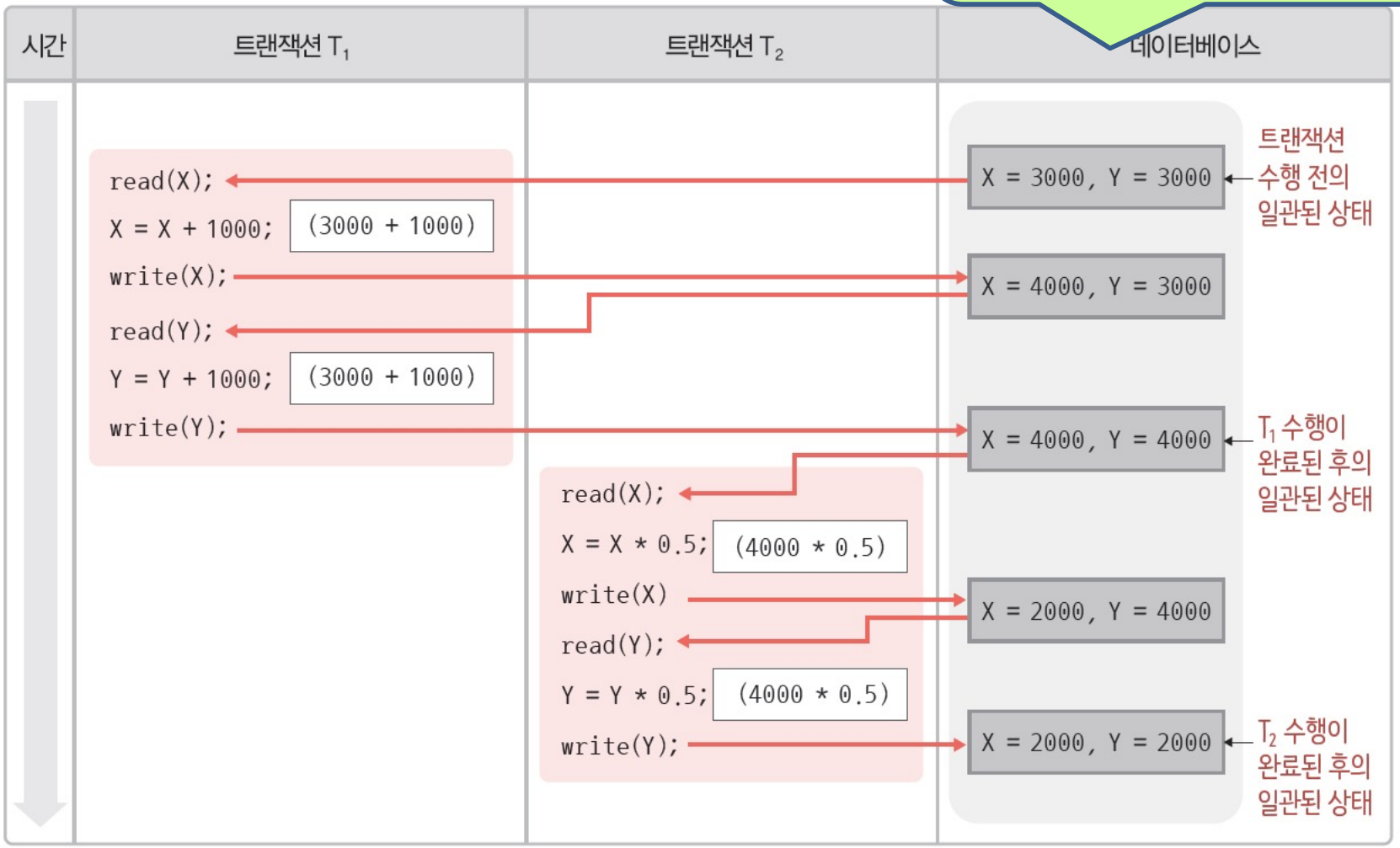
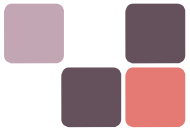


그림 10-33 트랜잭션 T_1을 수행한 후 트랜잭션 T_2를 수행한 결과

03 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
  - 여러 트랜잭션이 동시에 수행되더라도 연쇄 복귀 문제가 발생하지 않고 마치 트랜잭션들을 순차적으로 수행한 것과 같은 결과 값을 얻을 수 있어야 함

03 병행 제어

❖ 병행 수행 시 발생할 수 있는 문제점

- 연쇄 복귀

트랜잭션 T_1이 변경한 데이터 X를 가져가 연산을 수행한 트랜잭션 T_2도 rollback 연산이 연쇄적으로 실행되어야 하지만 이미 완료된 상태라 rollback 연산을 실행할 수 없는 문제가 발생

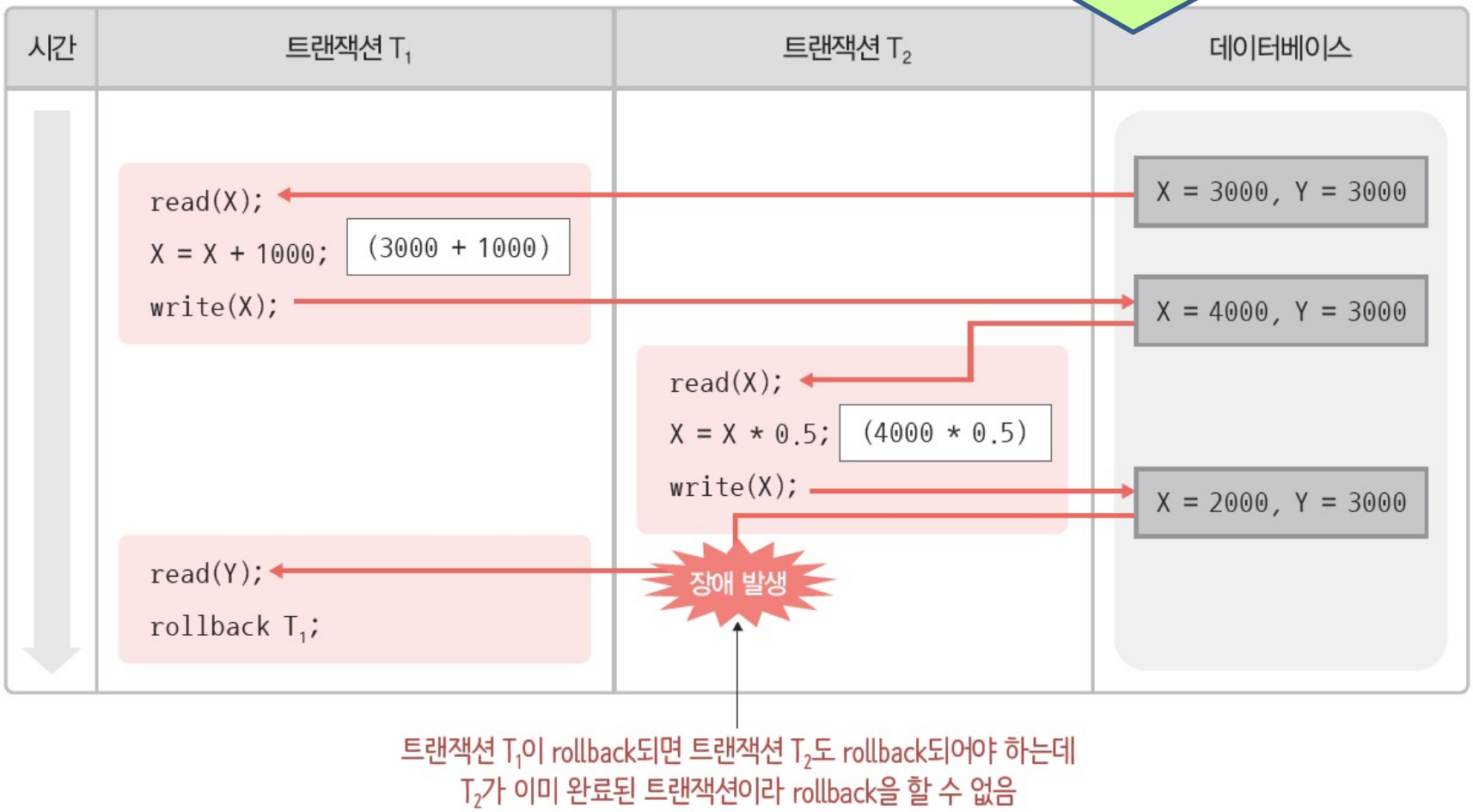


그림 10-34 두 트랜잭션의 병행 수행으로 발생한 연쇄 복귀의 예

03 병행 제어

❖ 병행 수행 시 발생할 수 있는 문제점

- 연쇄 복귀

트랜잭션을 순차적으로 수행해서
연쇄 복귀 문제가 발생하지 않는 경우

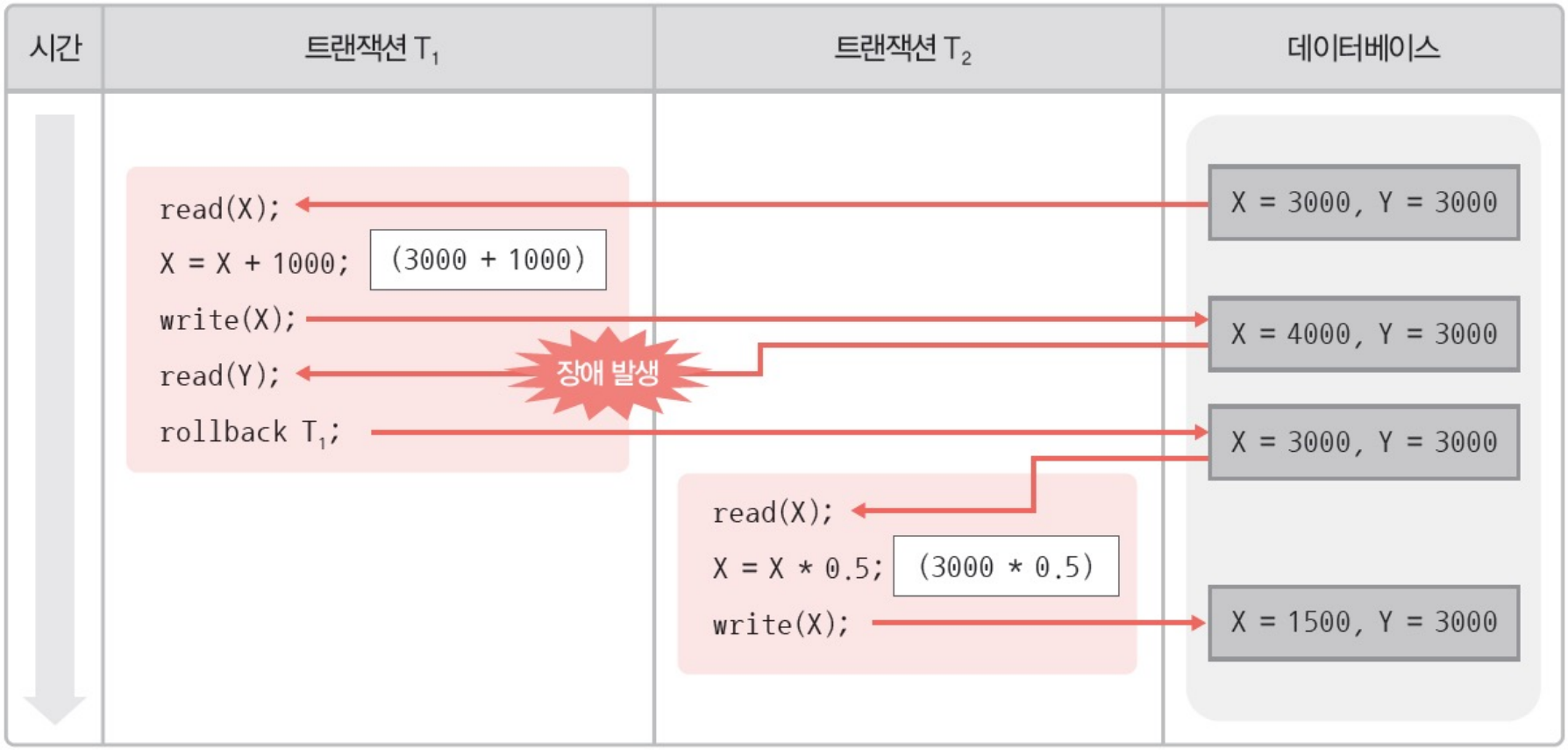
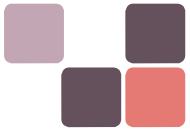


그림 10-35 트랜잭션 T_1을 수행한 후 트랜잭션 T_2를 수행한 결과

03 병행 제어



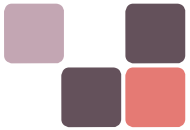
❖ 트랜잭션 스케줄

- 트랜잭션에 포함되어 있는 연산들을 수행하는 순서

표 10-4 트랜잭션 스케줄의 유형

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

03 병행 제어



❖ 직렬 스케줄(serial schedule)

- 의미
  - 인터리빙 방식을 이용하지 않고 각 트랜잭션별로 연산들을 순차적으로 실행시키는 것
- 특징
  - 직렬 스케줄에 따라 트랜잭션이 수행되면, 다른 트랜잭션의 방해를 받지 않고 독립적으로 수행되므로 항상 모순이 없는 정확한 결과를 얻게 됨
  - 다양한 직렬 스케줄이 만들어질 수 있고, 직렬 스케줄마다 데이터베이스에 반영되는 최종 결과가 다를 수 있지만 직렬 스케줄의 결과는 모두 정확함
  - 각 트랜잭션을 독립적으로 수행하기 때문에 병행 수행으로 볼 수 없음

03 병행 제어

❖ 직렬 스케줄 예

트랜잭션 T_1, T_2를 대상으로 하는 첫 번째 직렬 스케줄

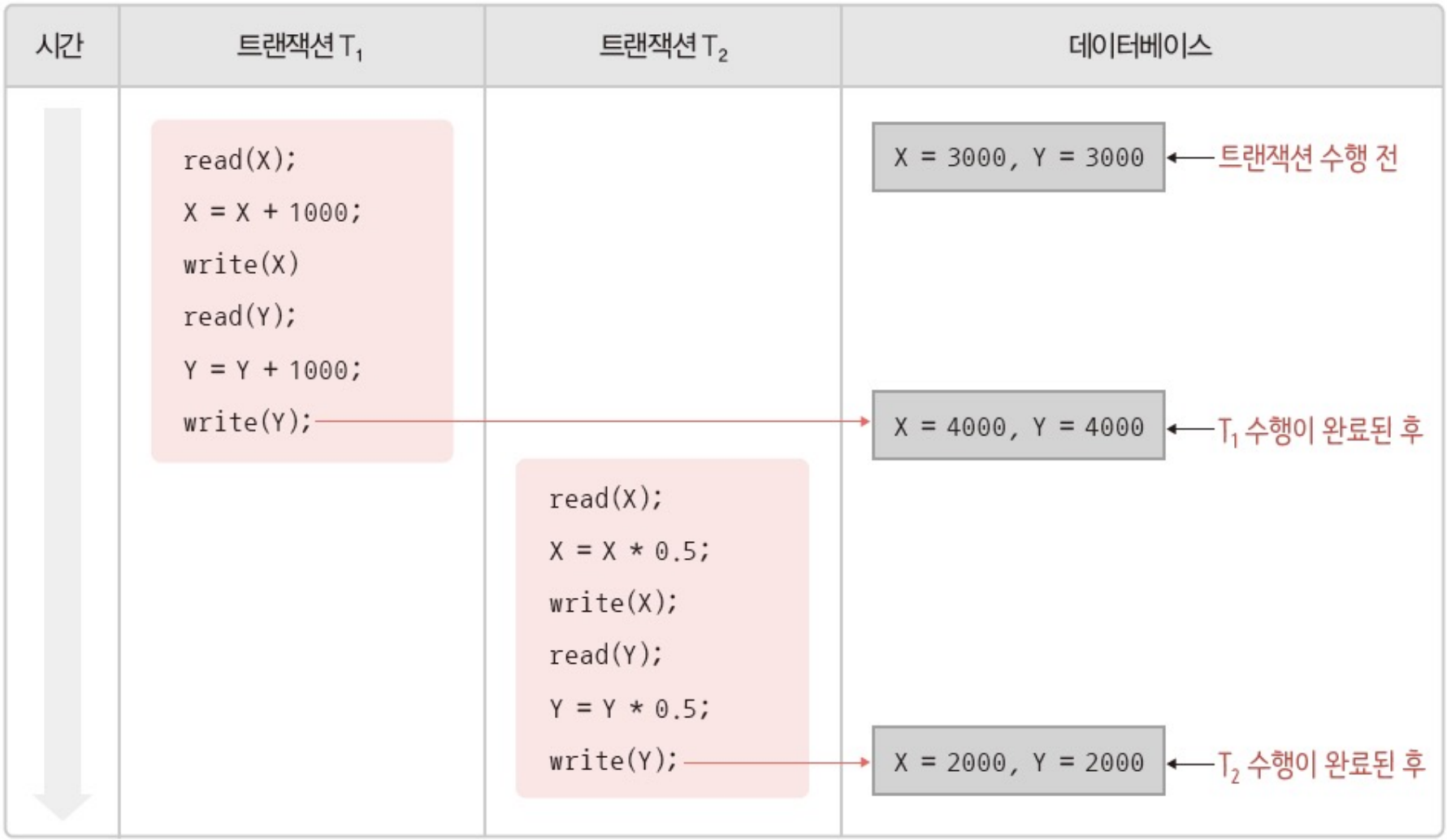


그림 10-36 트랜잭션 T_1을 수행한 후에 트랜잭션 T_2를 수행하는 직렬 스케줄의 예 : T_1 수행 → T_2 수행

03 병행 제어

❖ 직렬 스케줄 예

트랜잭션 T_1, T_2를 대상으로 하는 두 번째 직렬 스케줄

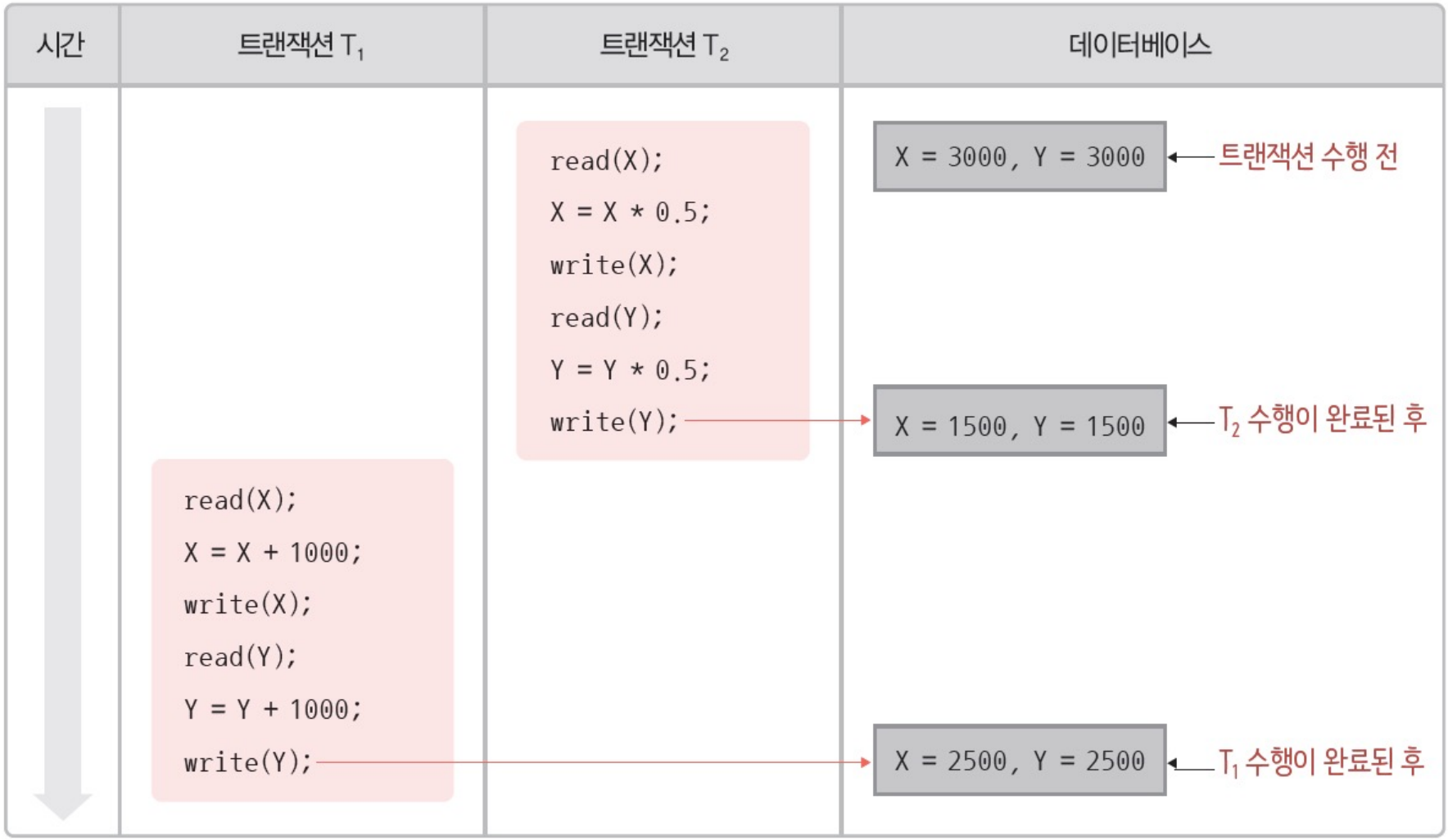
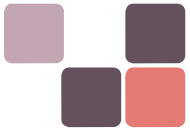


그림 10-37 트랜잭션 T_2를 수행한 후에 트랜잭션 T_1을 수행하는 직렬 스케줄의 예 : T_2 수행 → T_1 수행

03 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 그 중에는 잘못된 결과를 생성하는 것도 있음

03 병행 제어

❖ 비직렬 스케줄 예

트랜잭션 T_1, T_2를 대상으로 하는 첫 번째 비직렬 스케줄
(병행 수행에 성공하여 정확한 트랜잭션 수행 결과를 생성)

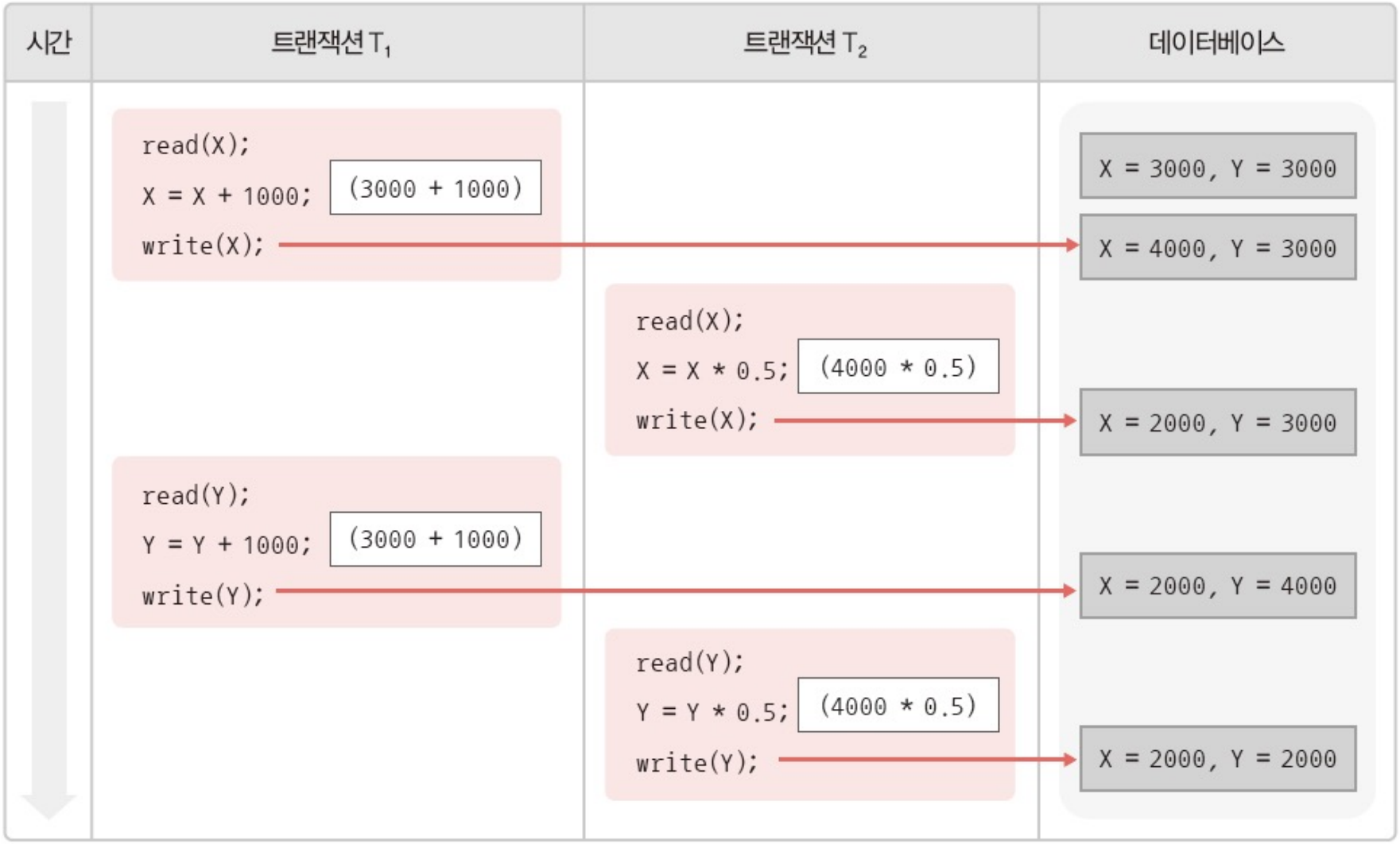


그림 10-38 트랜잭션 T_1과 T_2에 대한 비직렬 스케줄의 예 1 : 정확한 결과 생성

03 병행 제어

❖ 비직렬 스케줄 예

트랜잭션 T_1, T_2를 대상으로 하는 두 번째 비직렬 스케줄
(병행 수행에 실패하여 잘못된 트랜잭션 수행 결과를 생성)

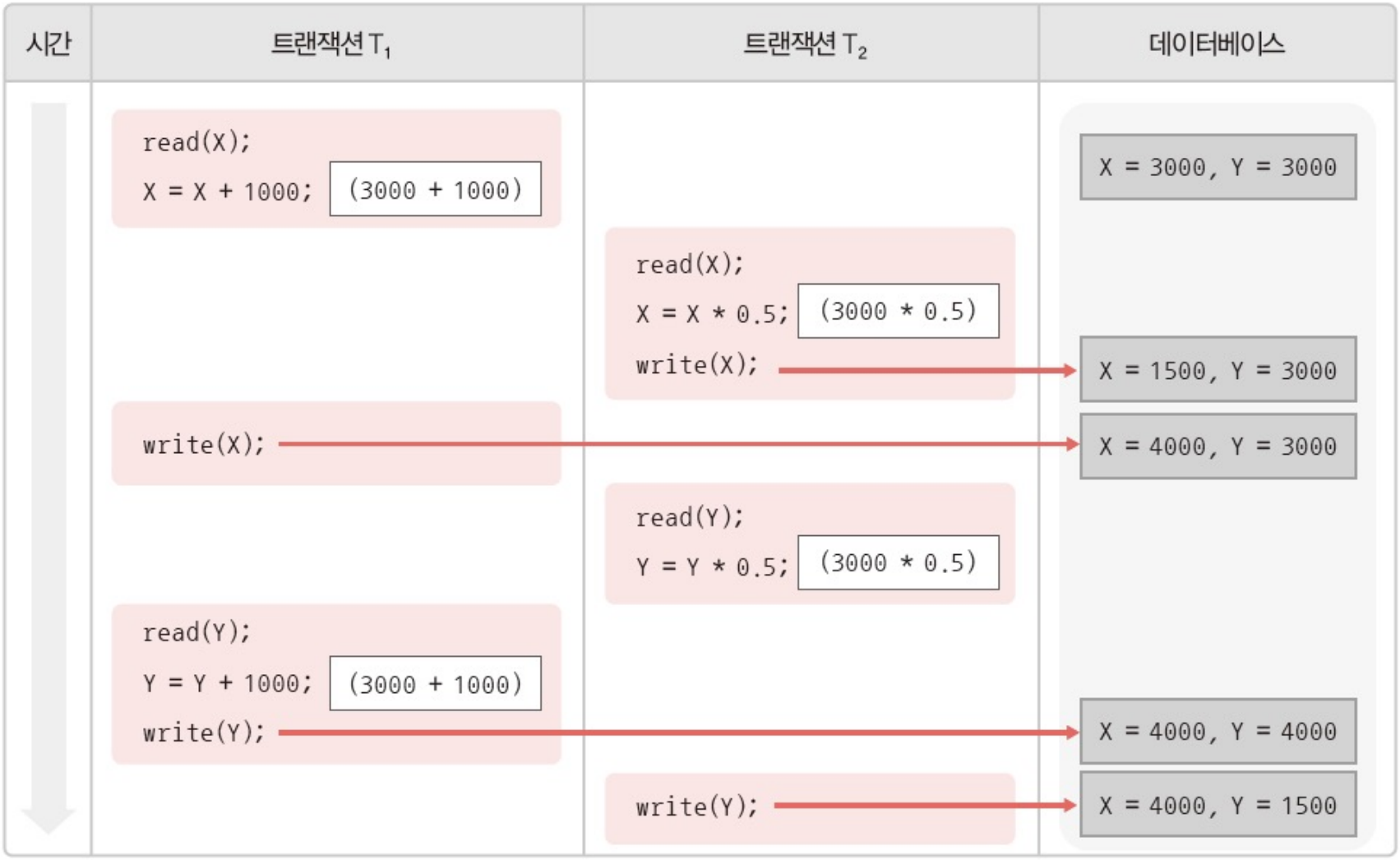
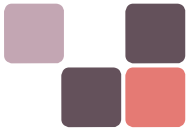


그림 10-39 트랜잭션 T_1과 T_2에 대한 비직렬 스케줄의 예 2 : 잘못된 결과 생성

03 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

03 병행 제어

❖ 직렬 가능 스케줄 예

트랜잭션 T_1, T_2를 대상으로 하는 비직렬 스케줄이면서 정확한 수행 결과를 생성하기 때문에 **직렬 가능 스케줄**임

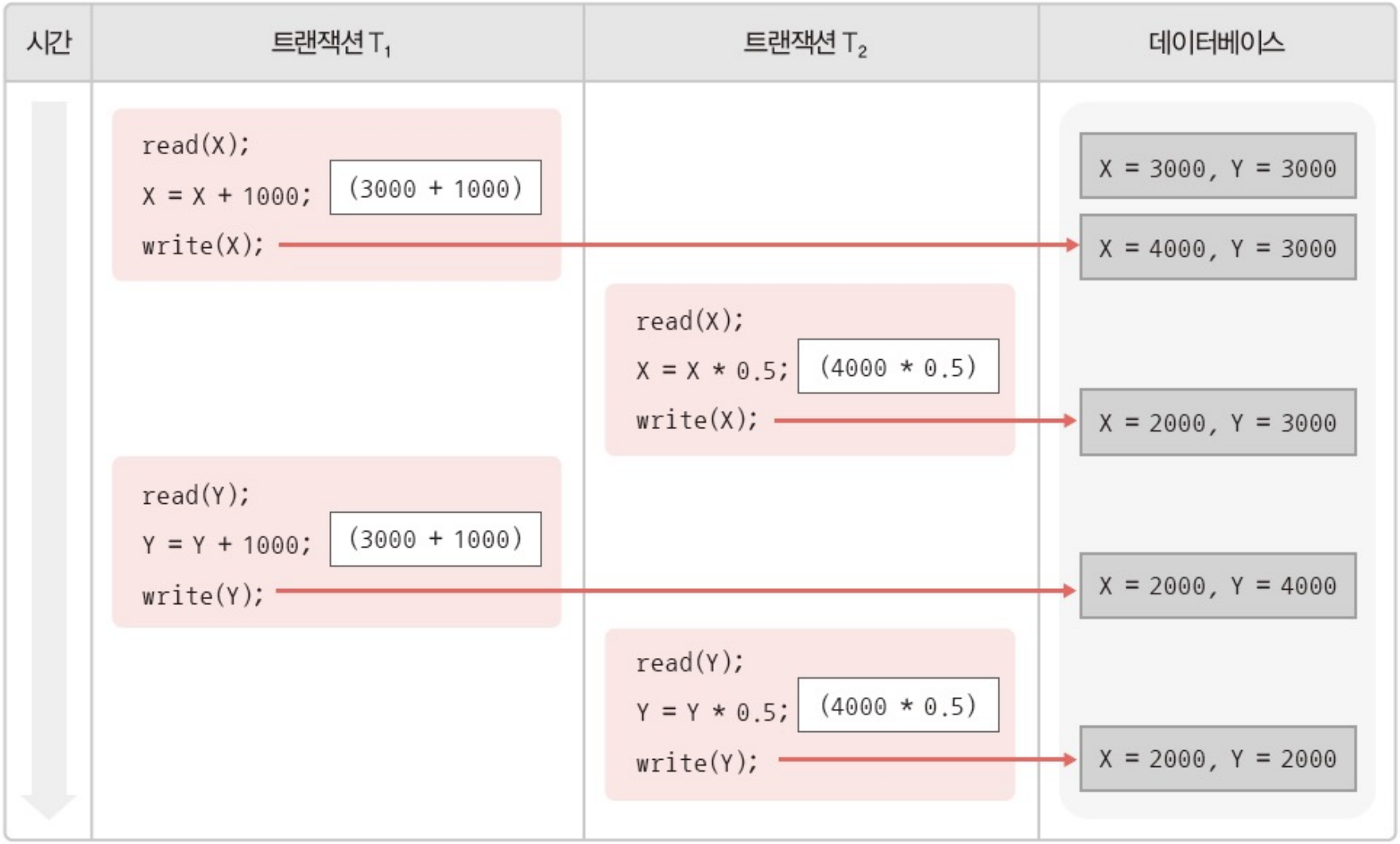
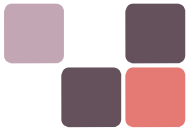


그림 10-38 트랜잭션 T_1과 T_2에 대한 비직렬 스케줄의 예 1 : 정확한 결과 생성

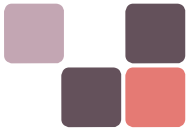
03 병행 제어



❖ 병행 제어 기법

- 의미
  - 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

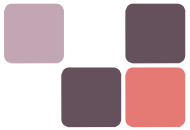
03 병행 제어



❖ 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

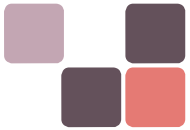
03 병행 제어



❖ 로킹 기법

- 기본 로킹 규약
  - 트랜잭션은 데이터에 접근하기 위해 먼저 lock 연산을 실행해 독점권을 획득함
    – read 또는 write 연산을 실행하기 전 lock 연산을 실행
  - 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

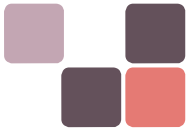
03 병행 제어



❖ 로킹 기법

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

03 병행 제어



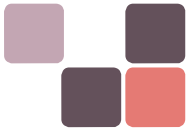
❖ 기본 로킹 규약의 효율성을 높이기 위한 방법

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

표 10-5 lock 연산

연산	설명
공용shared 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)
전용exclusive 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)

03 병행 제어



❖ 기본 로킹 규약의 효율성을 높이기 위한 방법

표 10-6 lock 연산의 양립성

	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

03 병행 제어

❖ 기본 로킹 규약으로 직렬 가능성이 보장되지 않는 스케줄 예

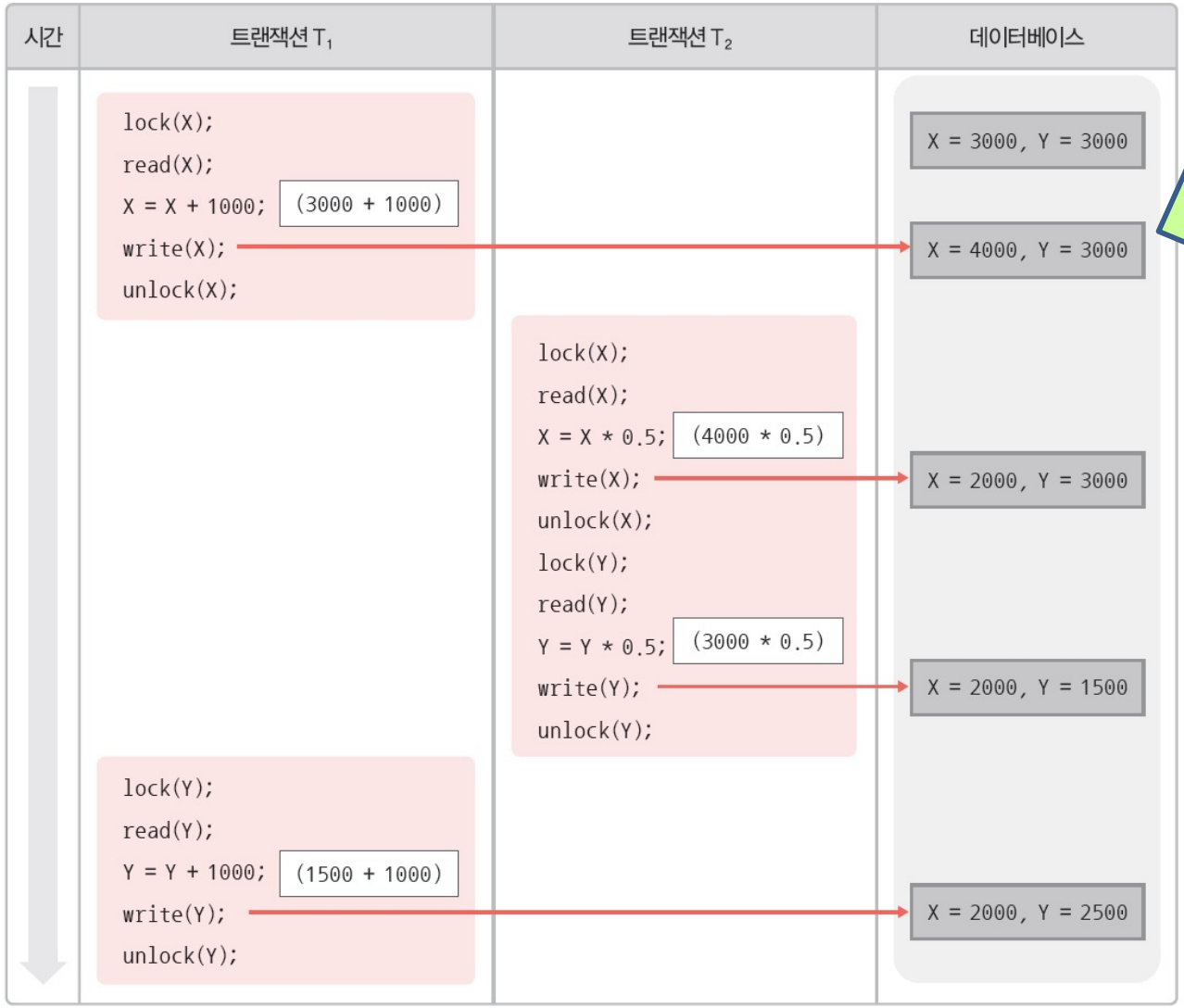
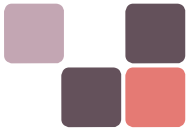


그림 10-40 기본 로킹 규약으로 직렬 가능성이 보장되지 않는 스케줄의 예



❖ 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
    - 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

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

그림 10-41 2단계 로킹 규약의 전략

03 병행 제어

❖ 2단계 로킹 규약

2단계 로킹 규약을 준수하여 직렬 가능성이 보장된 스케줄 예

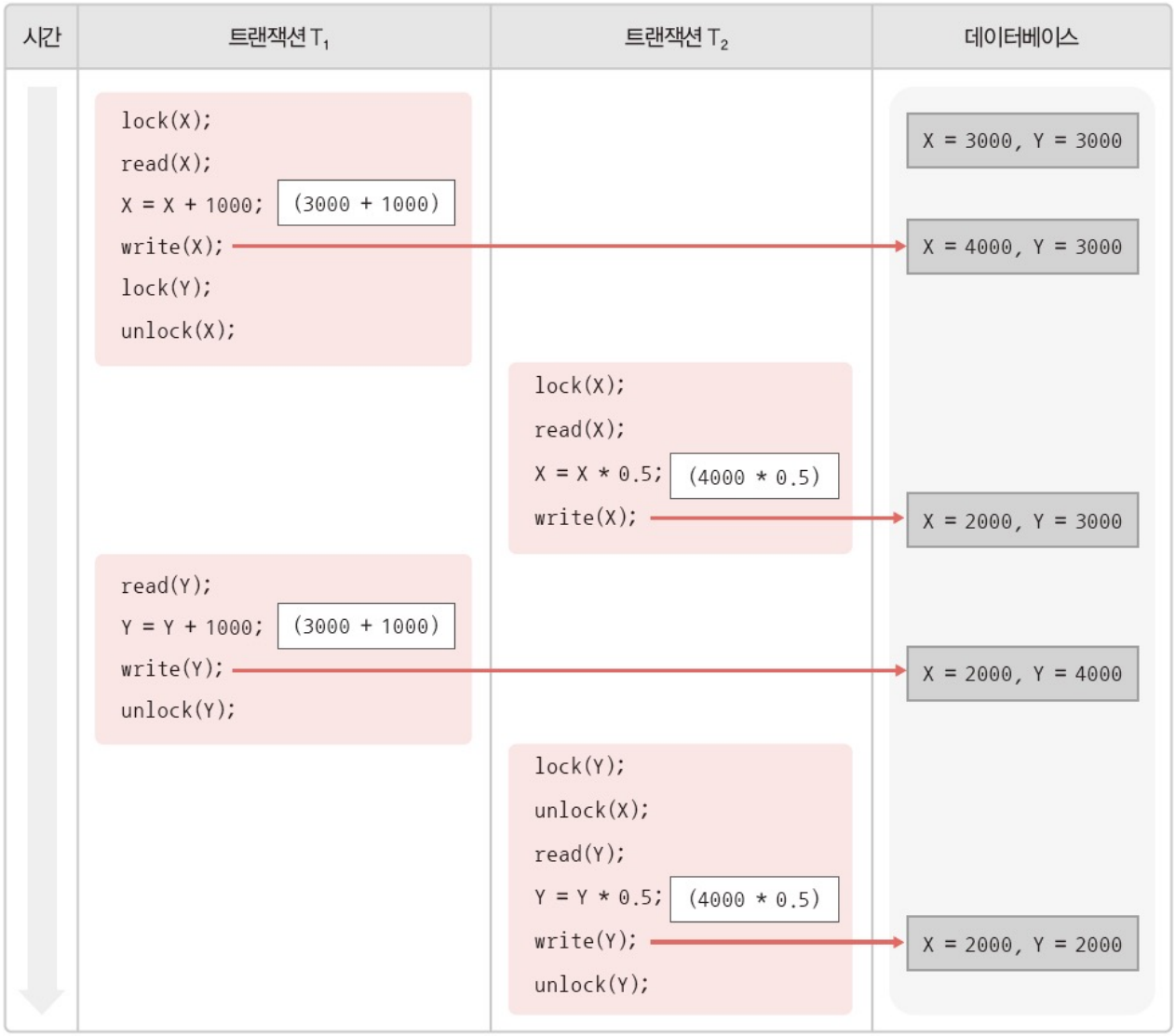
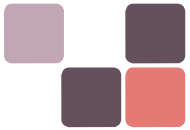


그림 10-42 2단계 로킹 규약으로 직렬 가능성이 보장된 스케줄의 예

03 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

Thank You