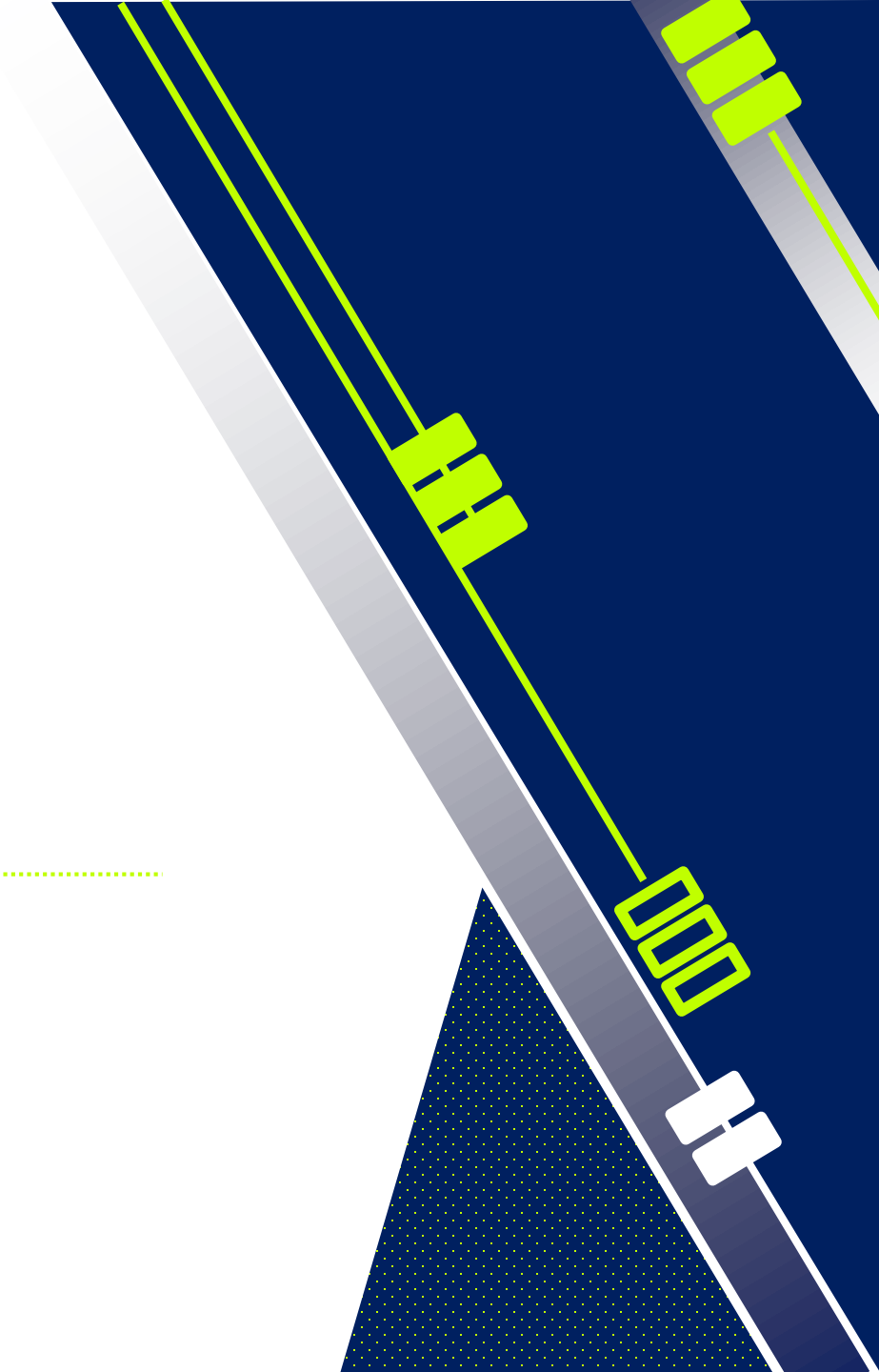


# 14<sup>강</sup> 회복 시스템

■ ■ ■ ■ 컴퓨터과학과 정재화



# 목 차



01. 회복 시스템의 개념

02. 로그 기반 회복

03. 회복 알고리즘



# 01

■ ■ ■ ■ 14강 회복 시스템

## 회복 시스템의 개념

- 회복의 역할
- 회복 데이터의 구성
- 데이터베이스 연산

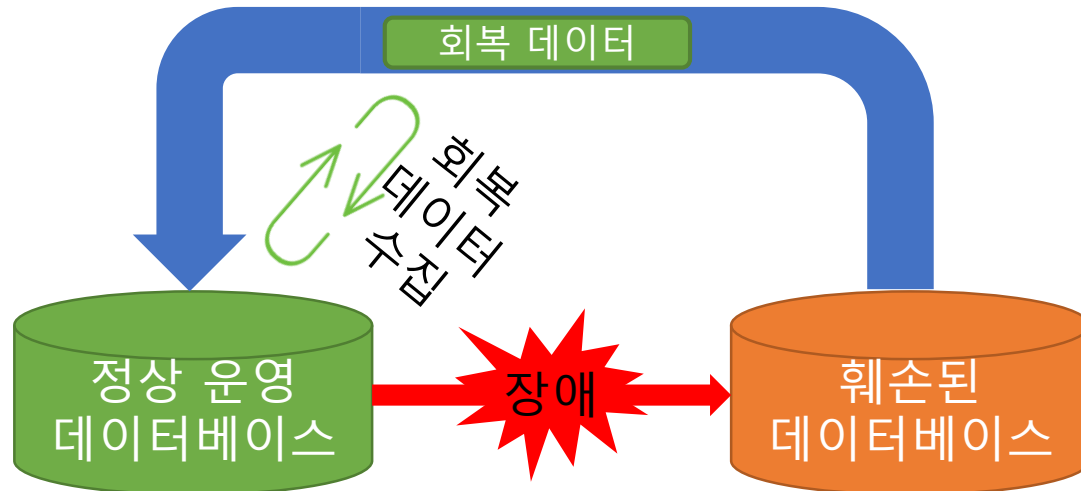
## 1. 예상치 못한 HW 고장 및 SW 오류가 발생

- ▶ 사용자의 작업에 대한 안정적 디스크 반영 여부 보장이 불가능

## 2. 오류 발생 이전의 일관된 상태로 데이터베이스를 복원시키는 기법이 요구

- ▶ 시스템 내의 고장 원인 검출, DBMS의 안전성 및 신뢰성을 보장

## 3. 데이터베이스는 데이터 복원 절차 내재화



## 1. 트랜잭션 실패

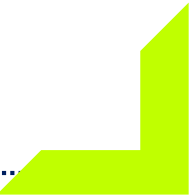
- ▶ 논리적: 잘못된 데이터 입력, 부재, 버퍼 오버플로, 자원 초과 이용
- ▶ 시스템적: 운용 시스템의 교착상태

## 2. 시스템 장애

- ▶ 시스템의 하드웨어 고장, 소프트웨어의 오류
- ▶ 주기억장치와 같은 휘발성 저장장치의 내용 손실

## 3. 디스크 실패

- ▶ 비휘발성 디스크 저장장치의 손상 및 고장으로 인한 데이터 손실

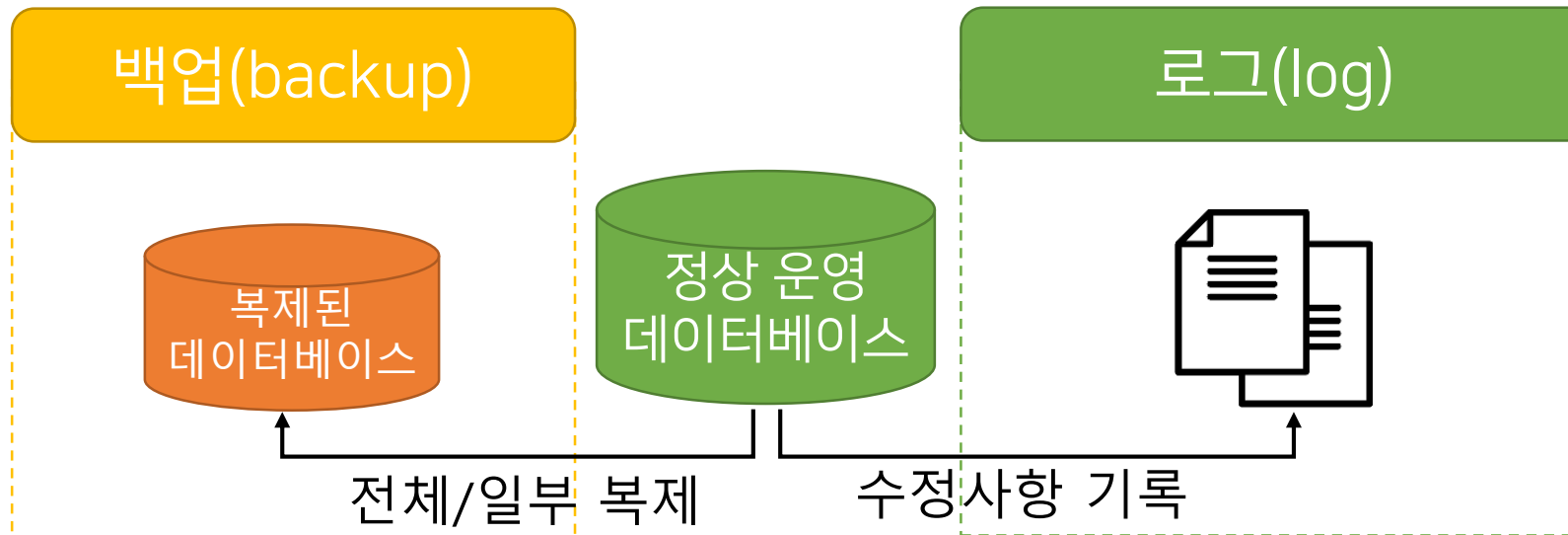


## 1. 백업(backup)

- ▶ 데이터베이스의 일부 또는 전체를 주기적으로 복제하는 방식

## 2. 로그(log)

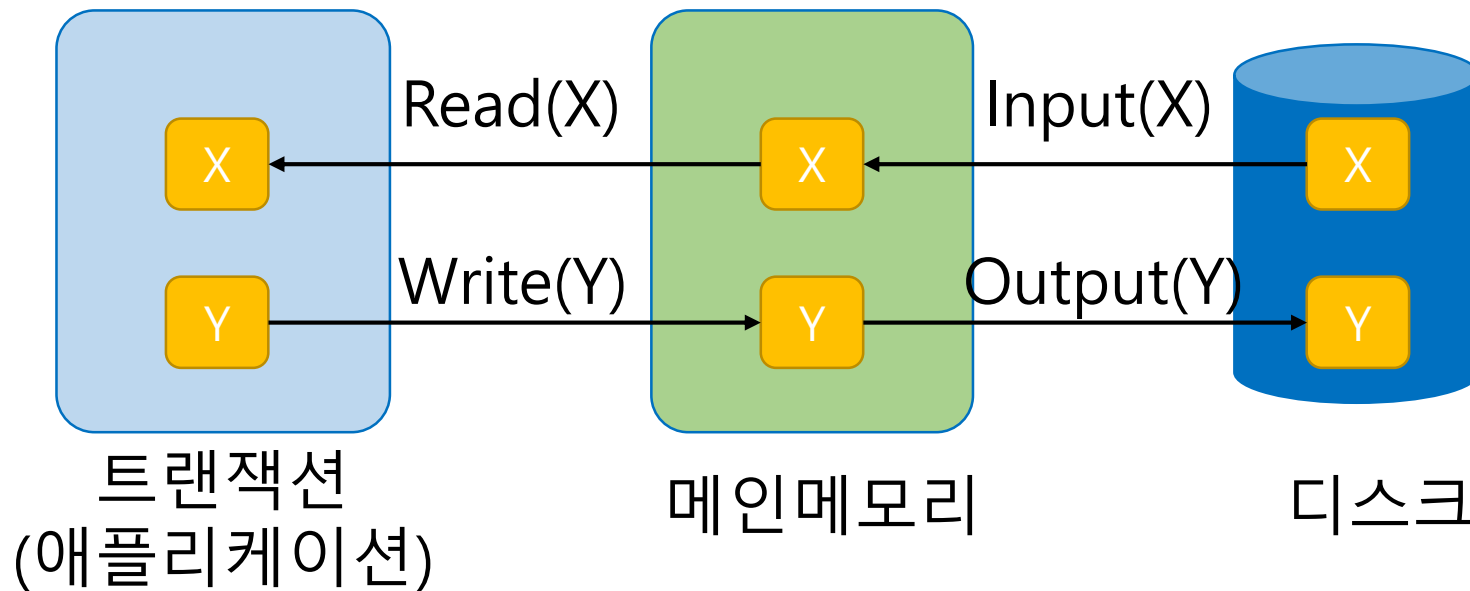
- ▶ 데이터 변경 이전과 이후의 값을 별도의 파일에 기록하는 방식



1. 데이터는 디스크와 같은 비휘발성 저장장치에 저장되며, 전체 데이터의 일부만 주기억장치에 상주
2. 데이터베이스는 데이터를 블록(block) 단위로 전송하고 블록 단위로 기억장소를 분할
3. 트랜잭션은 디스크로부터 주기억장치로 데이터를 가져오며, 변경된 데이터는 다시 디스크에 반영
  - ▶ 가져오기, 내보내기 연산은 블록 단위로 실행
  - ▶ 물리적 블록: 디스크 상의 블록
  - ▶ 버퍼 블록: 주기억장치에 일시적으로 상주하는 블록

## 1. 메인 메모리와 디스크 사이의 연산

- ▶ Input(X): 물리적 블록 X를 메인 메모리에 적재
- ▶ Output(X): 버퍼 블록 X를 디스크에 저장





# 02

■ ■ ■ ■ 14강 회복 시스템

## 로그 기반 회복

- 로그 기반 회복의 개념
- Redo와 Undo 연산
- 체크포인트의 필요

## 1. 데이터베이스가 수행한 모든 수정 작업을 기록한 여러 종류의 로그를 사용하여 회복하는 시스템

## 2. 로그 레코드

- ▶  $\langle T_i, X_j, V_1, V_2 \rangle$ :  $T_i$ 가 데이터 항목 변경 연산을 수행하여  $X_j$ 의 값을  $V_1$ 에서  $V_2$ 로 변경
- ▶  $\langle T_i, start \rangle$ :  $T_i$ 가 시작
- ▶  $\langle T_i, commit \rangle$ :  $T_i$ 가 커밋
- ▶  $\langle T_i, abort \rangle$ :  $T_i$ 가 취소

## 1. WAL (Write-Ahead Log)

- ▶ 트랜잭션은 데이터베이스 수정 전, 로그 레코드를 생성하여 기록

## 2. 데이터 항목 변경 과정

- ▶ 트랜잭션이 메인 메모리의 개인 영역에서 여러 연산을 수행
- ▶ 트랜잭션이 데이터 항목이 존재하는 메인 메모리에 위치한 버퍼 블록의 데이터를 변경
- ▶ Output 명령을 실행하여 버퍼 블록을 디스크에 기록



## 1. 회복 기법은 로그에 대해 두 연산을 사용

Redo( $T_i$ )

$T_i$ 에 의하여 수정된 새로운 값으로  
데이터베이스의 데이터 항목값을 수정

Undo( $T_i$ )

$T_i$ 에 의해 수정된 모든 데이터 항목을  
이전 값으로 복귀, 완료 후  $\langle T_i, abort \rangle$  기록

## 2. 시스템 장애 발생 시

- ▶ 로그에  $\langle T_i, start \rangle$  가 있지만  $\langle T_i, commit \rangle$  또는  $\langle T_i, abort \rangle$  를 포함하지 않는 경우  $T_i$ 는 Undo
- ▶ 로그에  $\langle T_i, start \rangle$  가 있지만  $\langle T_i, commit \rangle$  또는  $\langle T_i, abort \rangle$  를 포함하는 경우  $T_i$ 를 Redo

## 1. 데이터베이스 변경 시 복구 알고리즘의 고려 사항

- ▶ 트랜잭션의 일부 변경 사항이 버퍼 블록에만 반영되고 물리 블록에 기록되지 않은 상태에서 트랜잭션이 커밋되는 상황
- ▶ 트랜잭션이 동작 상태에서 데이터베이스를 수정했으나 수정 후에 발생한 실패로 취소가 필요한 상황

## 2. 트랜잭션 커밋 상황

- ▶  $\langle T_i, commit \rangle$  로그 레코드가 안정된 저장장치에 기록 완료 시 트랜잭션 커밋으로 간주
- ▶  $\langle T_i, commit \rangle$  로그 레코드가 기록되기 전에 장애가 발생하면 롤백

## 1. 회복은 트랜잭션에 의해 요청된 갱신 작업이 디스크에 반영되는 시점에 따라 구분

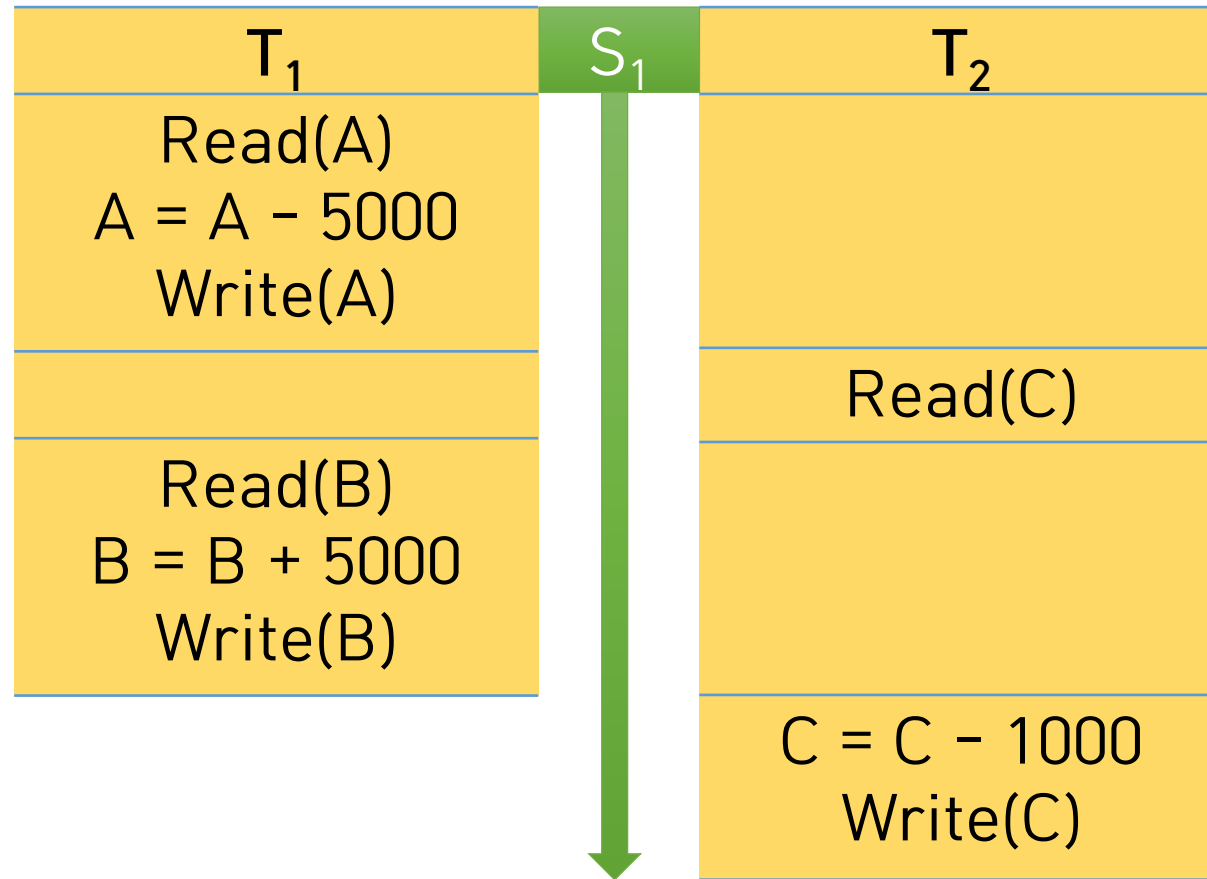
## 2. 지연 갱신 회복(deferred update restore)

- ▶ 부분 커밋까지 디스크 반영을 지연시키고 로그에만 기록
- ▶ 실패 시, 별도의 회복 작업 필요 없이 로그만 수정

## 3. 즉시 갱신 회복(immediate update restore)

- ▶ 갱신 요청을 곧바로 디스크에 반영
- ▶ 실패 시, 디스크에 반영된 갱신 내용을 로그를 바탕으로 회복

1.  $A = 30,000$     $B = 10,000$     $C = 50,000$



로그	지연 갱신 DB	즉시 갱신 DB
$\langle T_1, start \rangle$	A=30000 B=10000 C=50000	A=30000 B=10000 C=50000
$\langle T_1, A, 30000, 25000 \rangle$	A=30000 B=10000 C=50000	A=25000 B=10000 C=50000
$\langle T_1, B, 10000, 15000 \rangle$	A=30000 B=10000 C=50000	A=25000 B=15000 C=50000
$\langle T_1, commit \rangle$	A=25000 B=15000 C=50000	A=25000 B=15000 C=50000
$\langle T_2, start \rangle$	A=25000 B=15000 C=50000	A=25000 B=15000 C=50000
$\langle T_2, C, 50000, 40000 \rangle$	A=25000 B=15000 C=50000	A=25000 B=15000 C=40000
$\langle T_2, commit \rangle$	A=25000 B=15000 C=40000	A=25000 B=15000 C=40000



상황 1	상황 2	상황 3
$\langle T_1, start \rangle$	$\langle T_1, start \rangle$	$\langle T_1, start \rangle$
$\langle T_1, A, 30000, 25000 \rangle$	$\langle T_1, A, 30000, 25000 \rangle$	$\langle T_1, A, 30000, 25000 \rangle$
	$\langle T_1, B, 10000, 15000 \rangle$	$\langle T_1, B, 10000, 15000 \rangle$
	$\langle T_1, commit \rangle$	$\langle T_1, commit \rangle$
	$\langle T_2, start \rangle$	$\langle T_2, start \rangle$
	$\langle T_2, C, 50000, 40000 \rangle$	$\langle T_2, C, 50000, 40000 \rangle$
		$\langle T_2, commit \rangle$



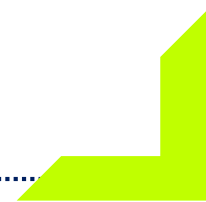
## 1. 로그 기반 회복 시스템의 한계

- ▶ 로그의 크기는 시간이 지남에 따라 계속 증가하므로 대용량 로그의 탐색 비용이 매우 커짐
- ▶ Redo를 해야 하는 트랜잭션 중 대부분은 이미 데이터베이스에 반영
- ▶ 반영된 트랜잭션의 재실행은 시스템 자원의 낭비



➡ 체크포인트 기법 제안

1. 현재 시점에 메인 메모리의 버퍼 블록에 존재하는 모든 로그 레코드를 안정 저장장치로 기록
2. 수정된 모든 버퍼 블록을 디스크에 반영
3. 로그 레코드  $\langle checkpoint ListT \rangle$ 를 안정한 저장장치에 기록
  - ▶  $ListT$ 는 체크포인트 시점에 실행 중인 트랜잭션 목록



## 1. 체크포인트 기법을 이용한 회복 과정

- ▶ 로그의 마지막부터 역방향으로 탐색하여  $\langle checkpoint ListT \rangle$  레코드를 찾음
- ▶  $ListT$ 에 존재하는  $\langle checkpoint ListT \rangle$  이후에 실행된 트랜잭션에 대하여 Redo와 Undo 연산만 수행
  - 로그에  $\langle T_i, commit \rangle$  또는  $\langle T_i, abort \rangle$ 가 없는  $ListT$ 안의 모든 트랜잭션을 Undo
  - 로그에  $\langle T_i, commit \rangle$  또는  $\langle T_i, abort \rangle$ 가 있는  $ListT$ 안의 모든 트랜잭션을 Redo

# 03

■ ■ ■ ■ 14강 회복 시스템

## 회복 알고리즘

- 트랜잭션 롤백 알고리즘
- 시스템 장애 후 회복 알고리즘

1단계: 로그를 역방향으로 탐색

2단계:  $T_i$ 의 로그 레코드  $\langle T_i, X_j, V_1, V_2 \rangle$ 에 대하여

- ▶ 데이터 항목  $X_j$ 에  $V_1$ 기록
- ▶ 로그 레코드  $\langle T_i, X_j, V_1 \rangle$ 을 로그에 기록

3단계:  $\langle T_i, start \rangle$ 를 찾은 이후

- ▶ 역방향 탐색을 중단
- ▶ 로그 레코드  $\langle T_i, abort \rangle$ 를 로그에 기록

$T_0$

```
Read(A);  
A = A - 5000;  
Write(A);  
Read(B);  
B = B + 5000;  
Write(B);
```

트랜잭션

```
<T0 start>  
<T2, C, 750, 60>  
<T0, A, 30000, 25000>  
<T5, D, 54900, 10>  
<T0, B, 10000, 15000>  
<T0, B, 10000>  
<T0, A, 30000>  
<T0, abort>
```

로그

## 1. 시스템 장애 이후 재시작 시 다음 두 단계를 거침

- ▶ Redo, Undo 단계

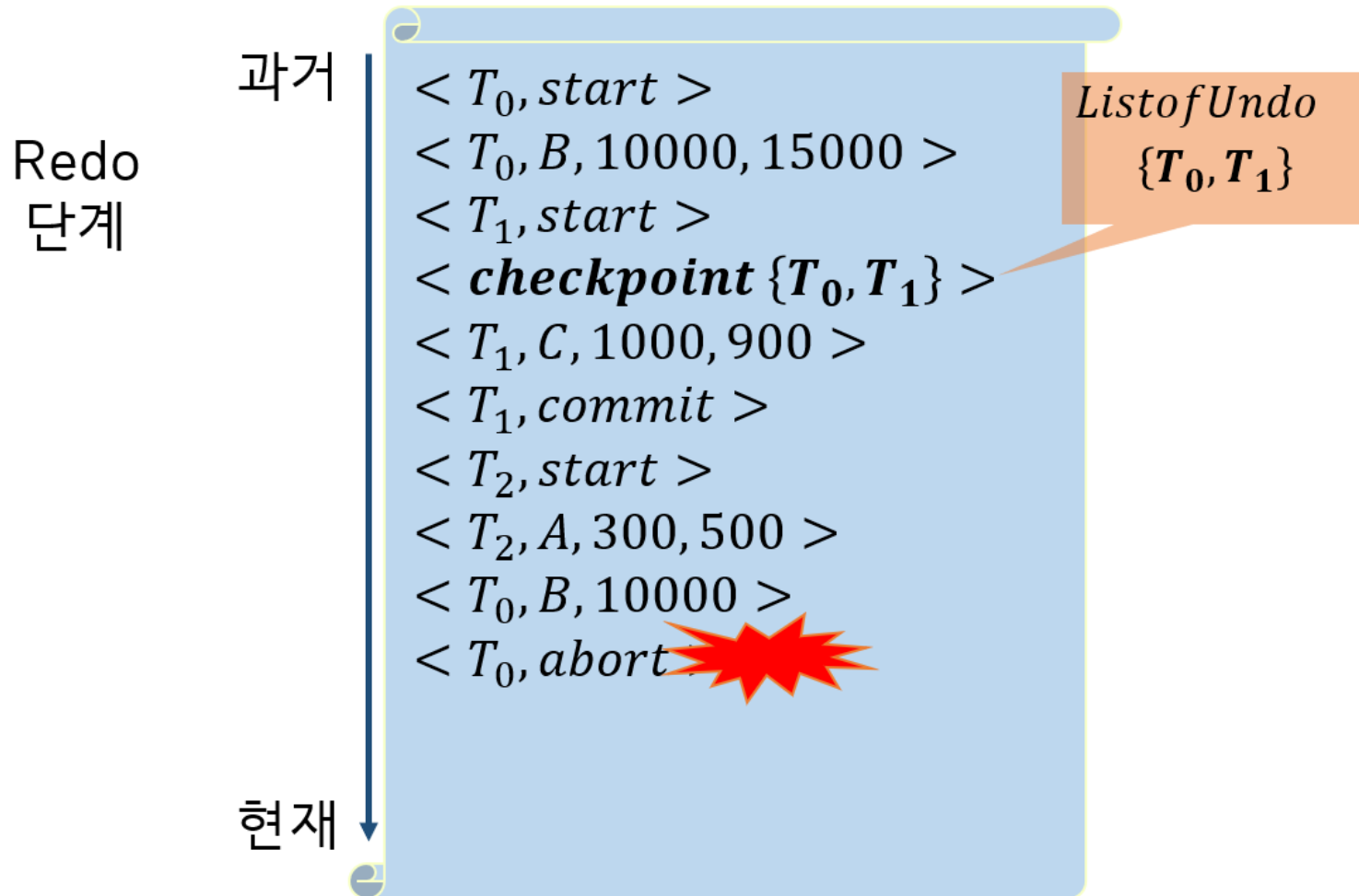
## 2. Redo 단계

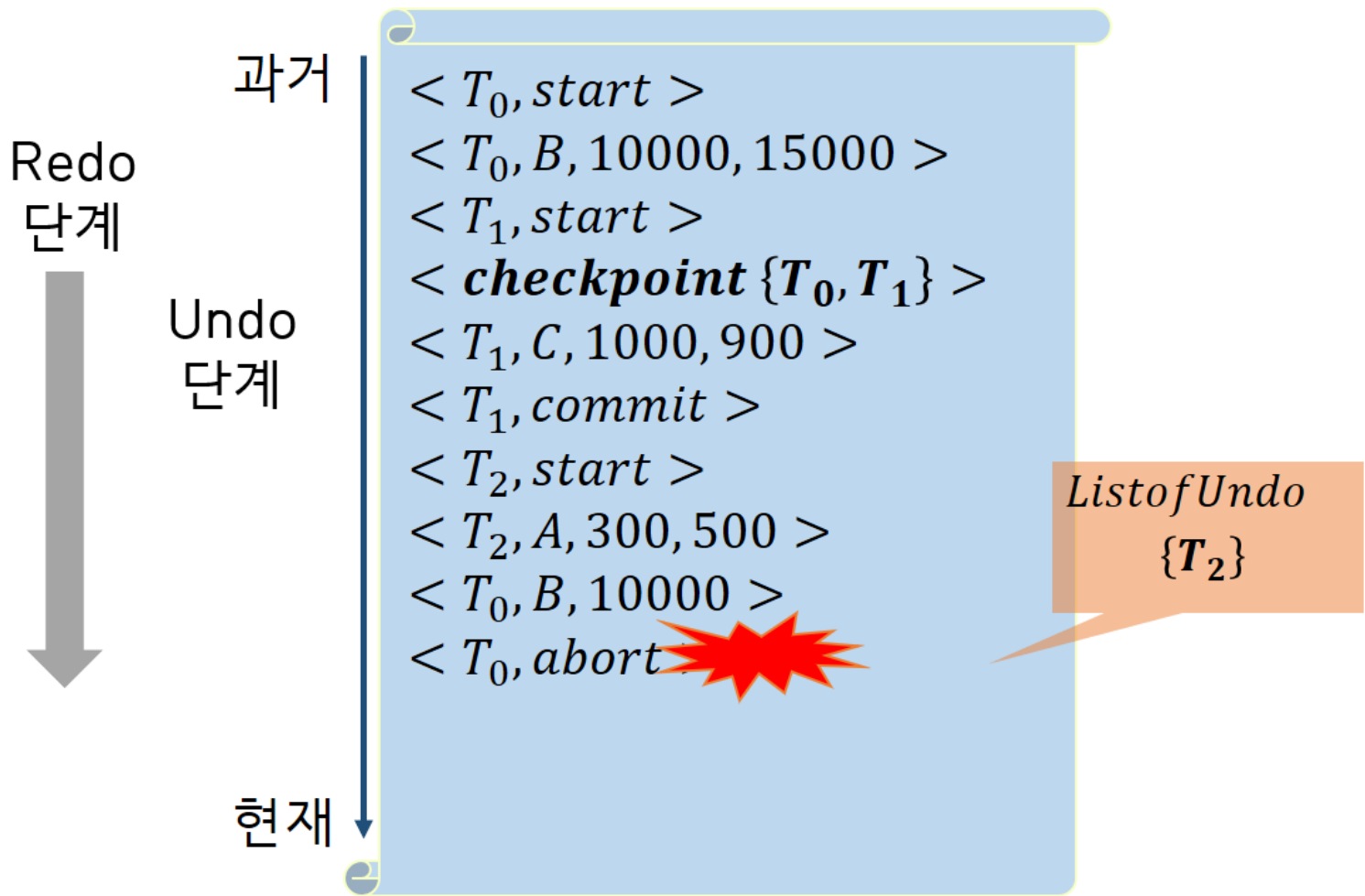
- ▶ 최근의 체크포인트에서부터 순방향 로그 탐색
- ▶ 롤백 대상할 트랜잭션의 Undo 리스트인 *ListofUndo*를 *ListT*로 초기화
- ▶  $\langle T_i, X_j, V_1, V_2 \rangle, \langle T_i, X_j, V_1 \rangle$  형태의 모든 레코드를 재실행
- ▶  $\langle T_i, start \rangle$  발견 시,  $T_i$ 를 *ListofUndo*에 추가
- ▶  $\langle T_i, abort \rangle, \langle T_i, commit \rangle$  발견 시  $T_i$ 를 Undo 리스트에서 제거

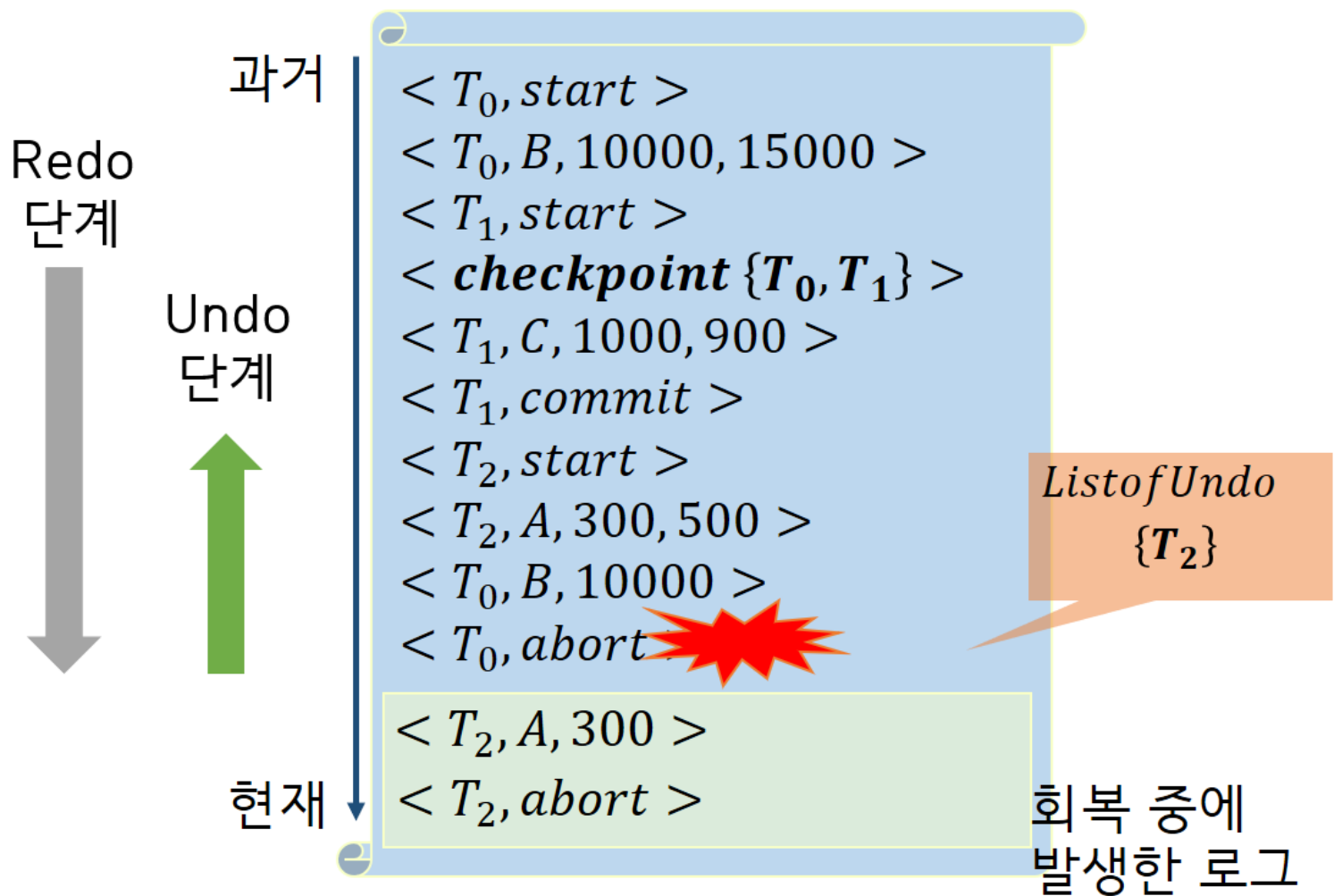


## 1. Undo 단계 (역방향 로그 탐색)

- ▶ *ListofUndo*의 트랜잭션의 로그레코드를 찾으면 트랜잭션 롤백 알고리즘 1단계 수행
- ▶ *ListofUndo*의 트랜잭션  $T_i$ 에 대해  $\langle T_i, start \rangle$ 를 만나면 로그에  $\langle T_i, abort \rangle$ 를 기록하고 *ListofUndo*에서 제거
- ▶ *ListofUndo*에 트랜잭션이 존재하지 않는 상태가 되면 Undo 단계를 종료









다음 시간에는

**15강 연습문제 풀이 2**를

학습하겠습니다.

