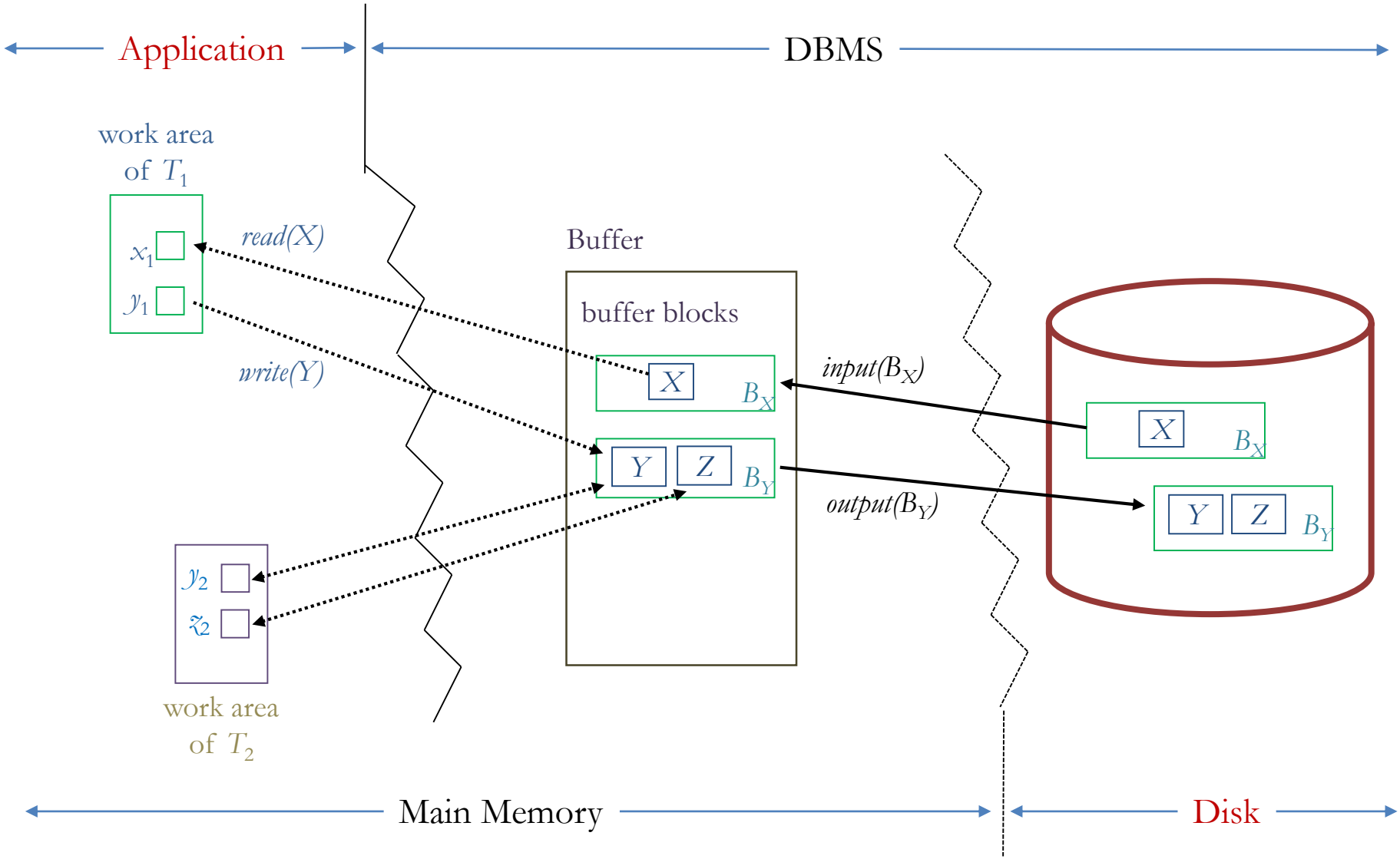
170918

DISCUSSIONS (LOG-BASED RECOVERY)

REVIEW

Transactional Data Access



Log-Based Recovery

- Assume that
  - transactions execute serially
  - log records are written directly to stable storage (not buffered)

- Output of updated blocks can take place at any time
  - before or after transaction commit
  - Order of output can be different from the order or writes

Immediate Database Modification

- Database is updated as soon as **write**(X) is executed
  - Updates of uncommitted transaction may be made to DB
  - Updates to DB does not necessarily mean updates to disk!

- Logging Strategy
  1. Transaction start: <T_i *start*>
  2. **write**(*X*) operation
     a. Write <T_i, X, V_1, V_2> to log
     b. Perform write operation (update DB)
  3. When T_i partially commits,
     Write <T_i **commit**> to log

  - Output of updated blocks can take place at any time

Immediate Database Modification (Cont.)

Log	Write	Output
<T_0 start>		
<T_0, A, 1000, 950>		
	A = 950	
<T_0, B, 2000, 2050>		
	B = 2050	
<T_0 commit>		
<T_1 start>		
<T_1, C, 700, 600>		
	C = 600	
		B_B, B_C
<T_1 commit>		
		B_A

(B_X: block containing X)

Undo & Redo Operations

- **redo**(T_i):
  - set the value of all data items updated by T_i to the new values
  - T_i 관련 첫 log record로부터 아래로 내려가면서 (순방향)

- **undo**(T_i):
  - restore value of all data items updated by T_i to their old values
    - 값을 원래 값으로 변경 시켰으므로 <T_i, X, V_1> log record를 log에 추가
  - T_i 관련 마지막 log record로부터 거꾸로 올라가면서 (역순)

- **redo & undo** operations must be *idempotent*
  - 여러 번 수행되더라도 한번 수행한 효과와 같아야 함
  - 복구 과정에서 몇 번 재수행 되는 경우 발생할 수 있음

Recovery Logic

Example

- The log as it appears at three instances of time.

(a)	(b)	(c)
<T_0 start>	<T_0 start>	<T_0 start>
<T_0, A, 1000, 950>	<T_0, A, 1000, 950>	<T_0, A, 1000, 950>
<T_0, B, 2000, 2050>	<T_0, B, 2000, 2050>	<T_0, B, 2000, 2050>
	<T_0 commit>	<T_0 commit>
	<T_1 start>	<T_1 start>
	<T_1, C, 700, 600>	<T_1, C, 700, 600>
		<T_1 commit>

- If log on stable storage at time of crash is as in case:
  - (a) undo (T_0)
  - (b) undo (T_1) and redo (T_0)
  - (c) redo (T_0) and redo (T_1)

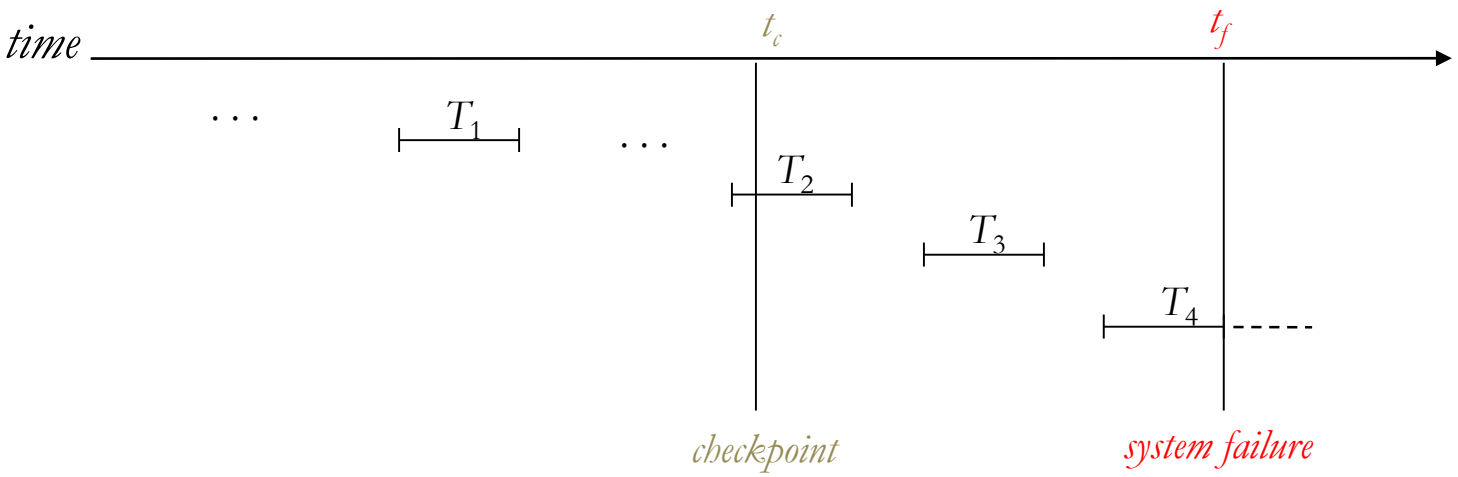
Checkpoints

- 만일 운영 1년만에 system crash가 났다면?
  - Update가 모두 disk에 반영됐다고 확신할 수 있는 transaction은?
  - => 어쩔 수 없이 1년 전 transaction부터 redo 해야!

- 운영 중간중간에 모든 update를 disk에 반영 - **checkpoint**

- Checkpoint process
  1. Output all log records in main memory onto stable storage
  2. Output all modified buffer blocks to disk
  3. Write a log record < **checkpoint**> onto stable storage

Example of Checkpoints



- T_1 can be ignored
  (updates already output to disk due to checkpoint)
- T_4 undone
- T_2 and T_3 redone

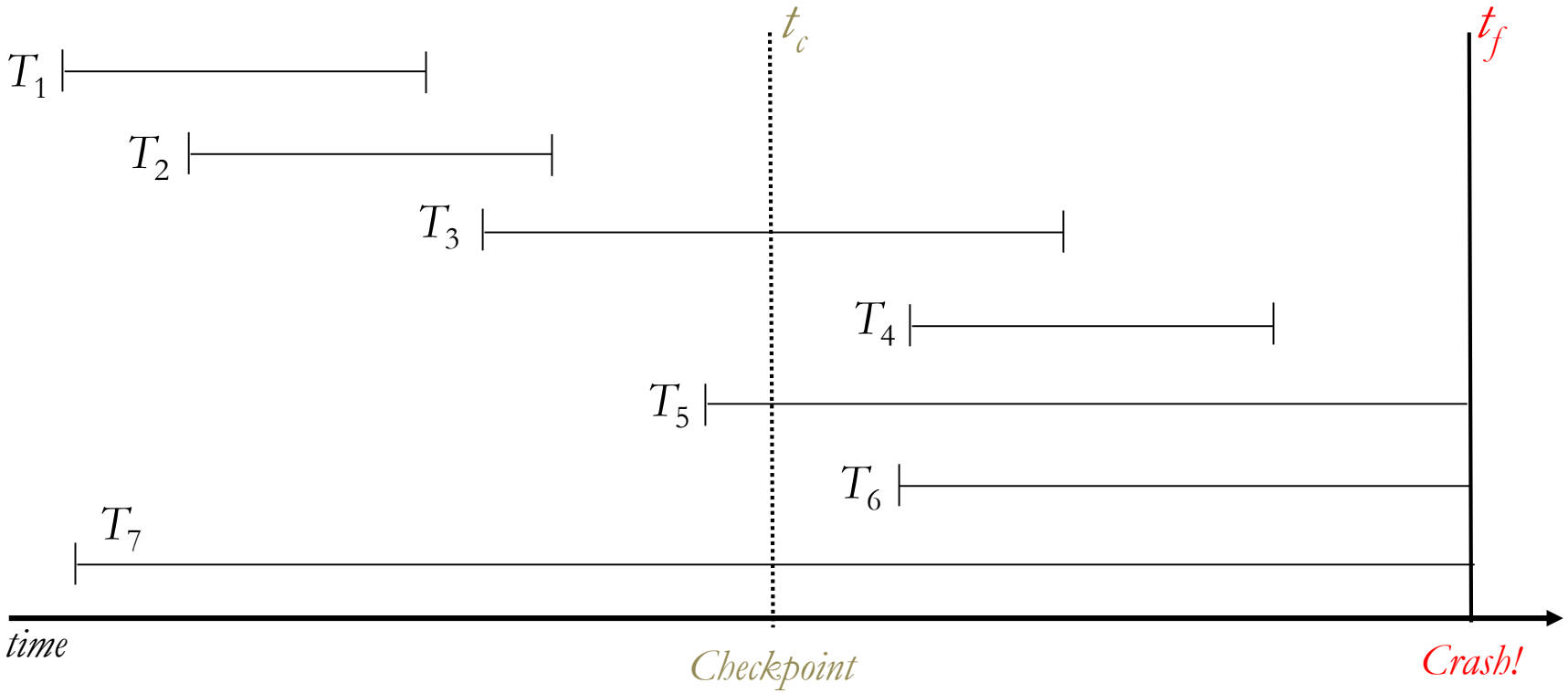
Recovery with Concurrent Transactions

- All transactions share a single disk buffer and a single log
  - 서로 다른 transaction의 log record들이 섞이게 됨 - 실행순서대로 log record 생성/저장
  - A buffer block can have data items updated by one or more transactions

- Assume concurrency control using strict two-phase locking;
  - i.e. updates of uncommitted transactions should not be visible to other transactions => recoverable

Recovery with Concurrent Transactions

- Checkpoint for concurrent transactions
  - Save *L*, list of active transactions at checkpoint time
  - => Checkpoint record: <**checkpoint** *L*>
  - The rest is identical to serial executions



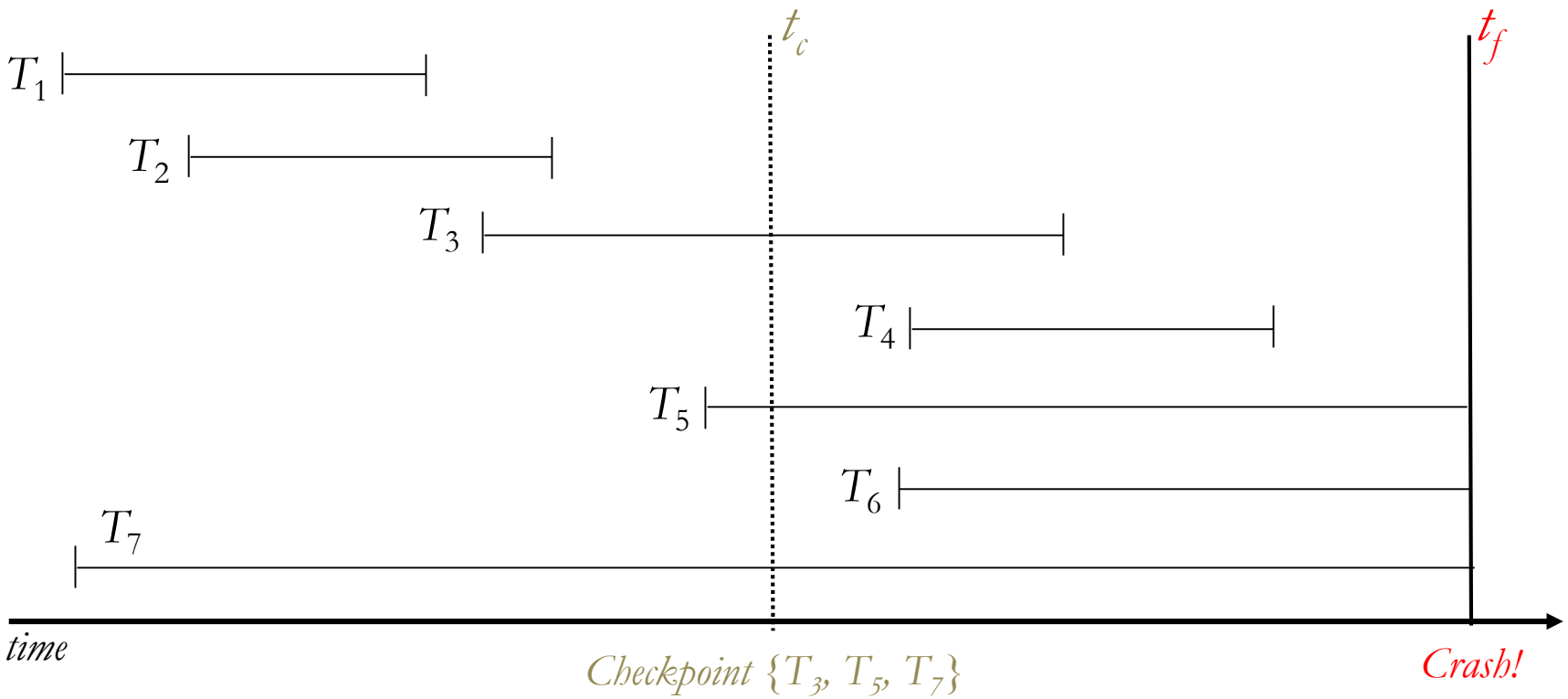
Transaction Rollback – during normal op.

- Log는 atomicity 보장에 중요
  - Fail한 transaction의 rollback과 abort

- Rollback transaction T_i :
  - Scan log backward (역방향)
    - For each log record $<T_i, X, V_1, V_2>$
      - Set value of data item X to V_1
      - $<T_i, X, V_1>$ is written to the log (redo-only log record)
  - Until $<T_i$ **start**> is found
  - Write $<T_i$ **abort**> to log

Recovery after a System Crash

- Transactions that need to be considered for *redo*/*undo* after a crash
  - Transactions **in *L*** of the last checkpoint
  - Transactions that **started after** the last checkpoint



Recovery after a System Crash

1. Redo Phase (repeating history)

> Scan log ***forward*** from last checkpoint log record <**checkpoint** *L*>

- Set *undo-list* to *L*
- For each log record <T_i, X, V_1, V_2> or <T_i, X, V_1> (redo-only log)
  => redo the operation
- For each log record <T_i, **start**>
  => add T_i to *undo-list*
- For each log record <T_i, **abort**> or <T_i, **commit**>
  => remove T_i from undo-list

> Until end of log

Recovery after a System Crash

- After the redo phase
  => undo-list contains list of transactions that are incomplete
  (not *committed* and not *completely rolled back (aborted)*)

2. Undo Phase

> Scan log ***backward*** from the end

- For each log record of a transaction in *undo-list*

  => perform undo actions (same as rollback during normal op.)

- For each log record <T_i, **start**> for T_i in *undo-list*

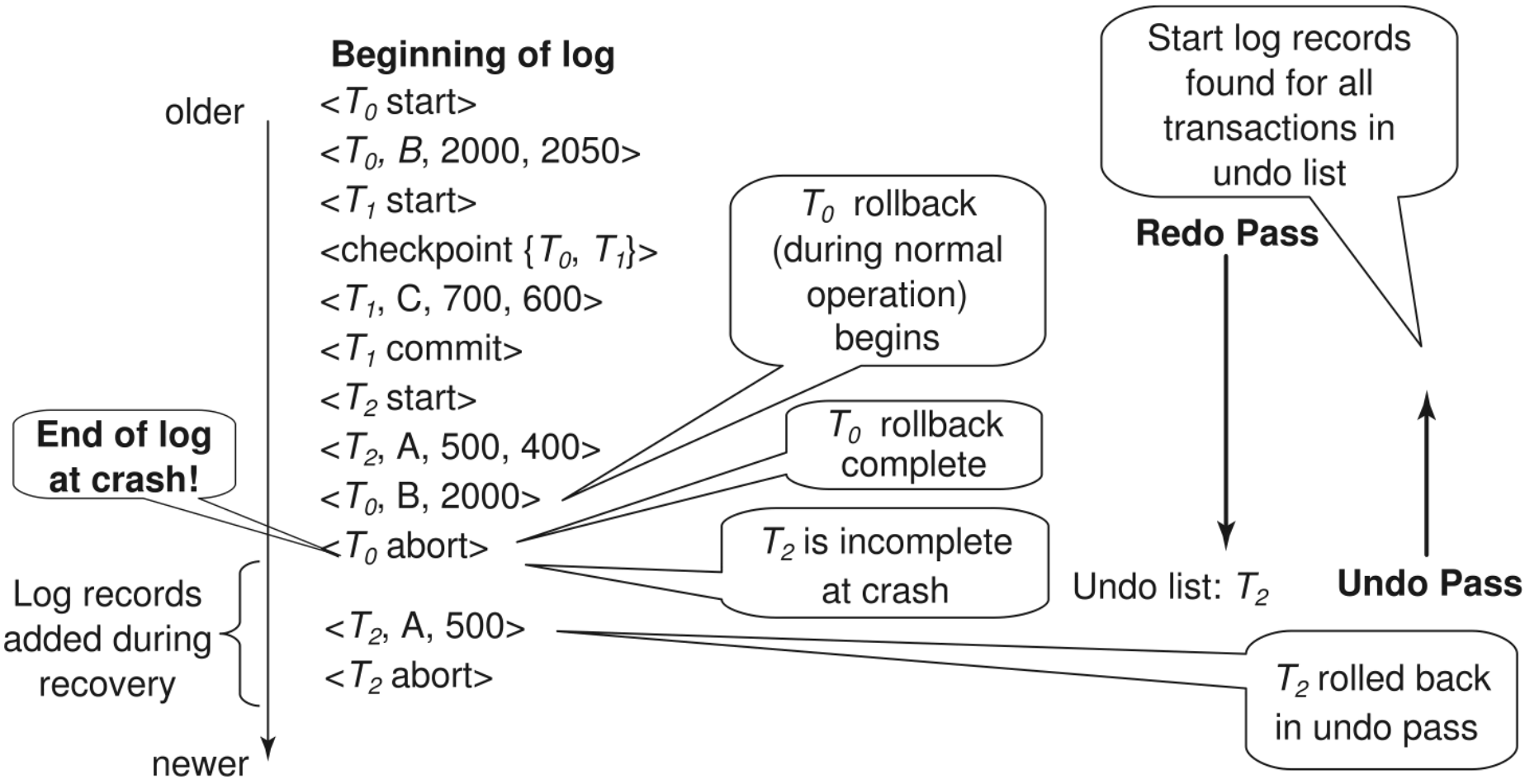
  => write <T_i **abort**> to the log

  => remove T_i from undo-list

> Until *undo-list* is empty

- <End of recovery process>

Example of Recovery



Log Record Buffering

- Log records are buffered in main memory
  - instead of being output directly to stable storage
  - several log records can be output using a single output operation

- Log records are **output** to stable storage when
  - a block of log records in the buffer is full, or
  - A ***log force*** operation is performed to commit a transaction by forcing all its log records (including the commit record) to stable storage.

Write-Ahead Logging (WAL)

Rules that must be followed for log record buffering

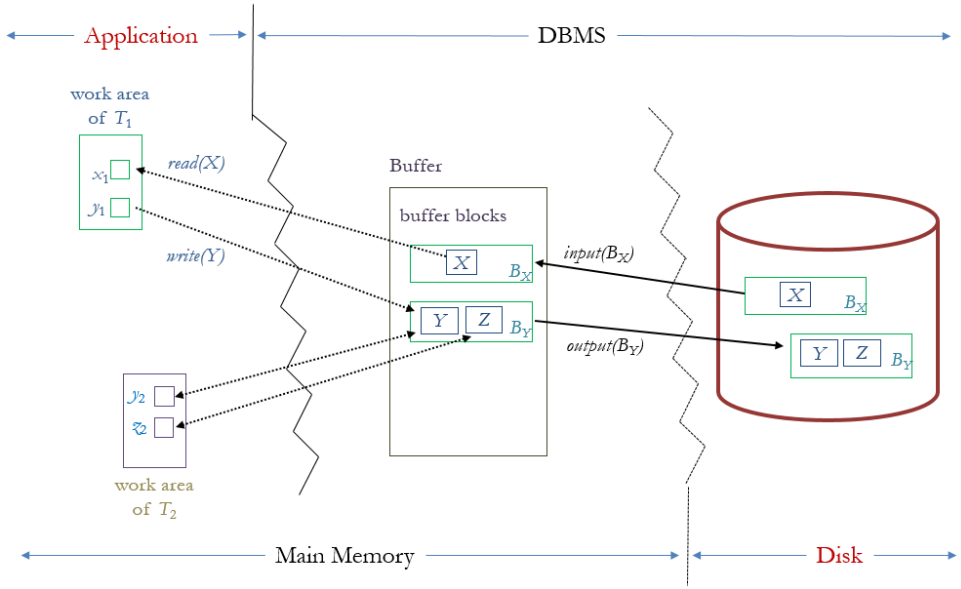
1. Log records are output to stable storage
   in the order they are created.
2. Transaction T_i enters the commit state only when (after) the log record <T_i **commit**> has been output to stable storage
3. Before a block of data in main memory is output to the disk,
   all log records about data in that block must have been output to stable storage.

   => called the ***write-ahead logging*** or ***WAL*** rule

LOG-BASED RECOVERY SIMULATION

Log-based Recovery Simulation

- Log-based Recovery는 본래 DBMS에 의해 자동으로 관리되고 수행되는 기능
- 하지만 Application(Python) programming을 통해 이를 간단히 simulation해 볼 수 있다!
- Requirements: Python + PyMySQL(DB connector) + Log



Recap: PyMySQL

- PyMySQL installation
  - 명령 줄 인터페이스(cmd)에서 (윈도우키 + r, cmd 실행)
  - 명령어 입력: **pip install PyMySQL**
- 또는 빅데이터 프로그래밍 시간에 다뤘던 다른 Python-MySQL DB connection library를 사용해도 무방

Recap: PyMySQL

```
import pymysql.cursors

# Connect to the database
connection = pymysql.connect(
    host=147.46.15.66',
    user='galsang',
    password='bde1234',
    db='galsang',
    charset='utf8',
    cursorclass=pymysql.cursors.DictCursor)

try:
    with connection.cursor() as cursor:
        # Select records from Student table
        sql = "SELECT * FROM student"
        cursor.execute(sql)
        result = cursor.fetchall()
        print(result)
finally:
   connection.close()
```

Recap: PyMySQL

- 학생의 이름을 입력으로 받아서 해당 학생이 들은 모든 과목의 이름과 성적을 출력하는 프로그램

```
import pymysql.cursors

# Connect to the database
connection = pymysql.connect(
    host='147.46.15.66',
    user='galsang',
    password='bde1234',
    db='galsang',
    charset='utf8',
    cursorclass=pymysql.cursors.DictCursor)

name = input("학생의 이름을 입력해주세요: ")

try:
    with connection.cursor() as cursor:
        sql = "SELECT * " ₩
              + "FROM student natural join takes join class on takes.class_id = class.class_id " ₩
              + "natural join course " ₩
              + "WHERE name=%s"
        cursor.execute(sql, name)
        result = cursor.fetchall()
finally:
   connection.close()

print("이름: ", name)
print("과목명₩t성적")
for row in result:
    print(row["title"] + "₩t" + row["grade"])
```

Exercise 1

- 주어진 (학사 정보) DB와 log 파일이 있을 때, log 파일에 알맞게 DB recovery를 simulation하는 Python 프로그램을 작성해보자.
  - 단, 주어진 DB의 상태는 오류가 난 후 불완전한 상태이며, recovery.log 파일에 저장된 log를 순서대로 실행하여 recovery를 완료하였을 경우 제대로 복구된다고 가정하자.
- 초기 세팅: DB schema SQL, 예시 log 파일 (**recovery.log**)
  - 과목 게시판 참조
- 문제 간소화를 위하여 다음과 같이 조건을 제한함
  - 사용되는 SQL문은 반드시 다음의 형태를 따름
    - UPDATE <table_name> SET <column_name> = <value> WHERE <primary_key> = <value>
  - 값을 변경하려는 column은 foreign key가 아닌 일반 column.
  - TAKES 테이블은 제외(primary key가 pair로 이루어져 있는 경우)

Exercise 1

- recovery.log 파일 포맷

포맷	설명	예시
<Transaction ID> start	해당 transaction 시작	<T1> start
<Transaction ID> <Table>.<key>.<column>, <oldValue>, <newValue>	해당 table의 <key> 값에 해당하는 레코드의 <column>의 <oldValue>를 <newValue>로 변경	<T3> student.1292001.address, 서울, 대구 <T2> course.C304.title, 객체지향언어, 기계학습
<Transaction ID> <Table>.<key>.<column>, <value>	해당 table의 <key> 값에 해당하는 레코드의 <column>에 해당하는 부분을 <value>로 변경	<T1> course.C01.title, 컴퓨터의개념및실습 <T2> student.1292001.year, 1
<Transaction ID> commit	해당 transaction commit	<T3> commit
<Transaction ID> abort	해당 transaction rollback	<T1> abort
checkpoint <Transaction ID>,…, <Tran ID>	checkpoint 에서의 active transaction	checkpoint <T1>, <T2>, <T3>

Tips

- **def read_log():**
  - log 파일의 문자열을 읽어서 Python으로 가져오는 함수
  - log 파일은 항상 Python 프로그램과 같은 경로에 위치하며,
  - 파일 이름은 **recovery.log**로 고정되어 있다고 가정
- **def parse_log():**
  - 읽어온 log의 각 라인을 parsing한 후,
  - 각 라인이 어떠한 기능을 하고 있는 log인 지 파악하고,
  - 각 log에 알맞은 logic을 수행하도록 하는 함수
- **def connect_db():**
  - DB에 접속하는 함수
- **def execute_sql():**
  - 각 log를 수행하기 위해 알맞게 만들어 낸 SQL(UPDATE 문)을 DB에서 실행
- And so on…

Tips

- Redo phase / Undo phase를 순서대로 실행
- Transaction의 undo을 관리하는 undo list를 항상 고려

Submission

- 제출: lecture@europa.snu.ac.kr
  - 제목: [bde2_recovery] <이름>
    - ex) [bde2_recovery] 김태욱
  - 반드시 개인 별로 제출 (단, 실습을 하는 동안 discussion 권장)
- 제출물: 아래를 압축하여 제출 (파일 이름: <이름>.zip)
  - 주어진 recovery.log를 수행한 후의 DB 상태에서 **course, professor, department, employee** table의 상태를 select한 결과를 제출
    - 결과 캡쳐 혹은 결과를 담고 table 형태의 text
  - 작성 완료한 프로그램 코드
    - **주어진 recovery.log 파일 뿐만 아니라 다른 log로도 테스트할 수 있음**

Exercise 2 (Optional)

- Exercise 1의 Python 프로그램을 완성했다면, recovery 과정에서 log의 수정(추가)이 필요한 경우를 생각해보자.
  - Undo phase?
- 만약 필요하다면 해당 log를 recovery 도중에 recovery.log에 수정(추가) 하도록 프로그램을 수정해 보자.