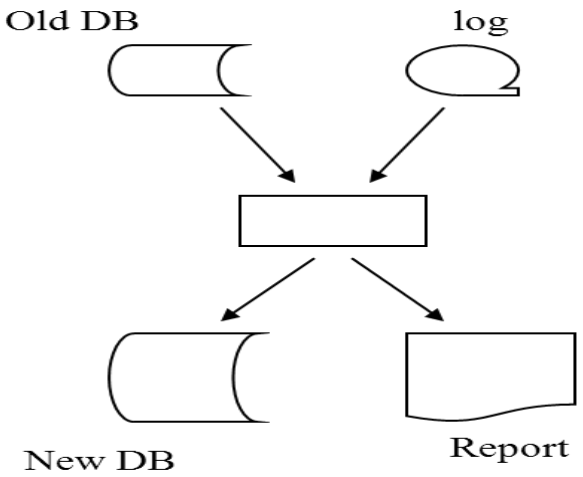
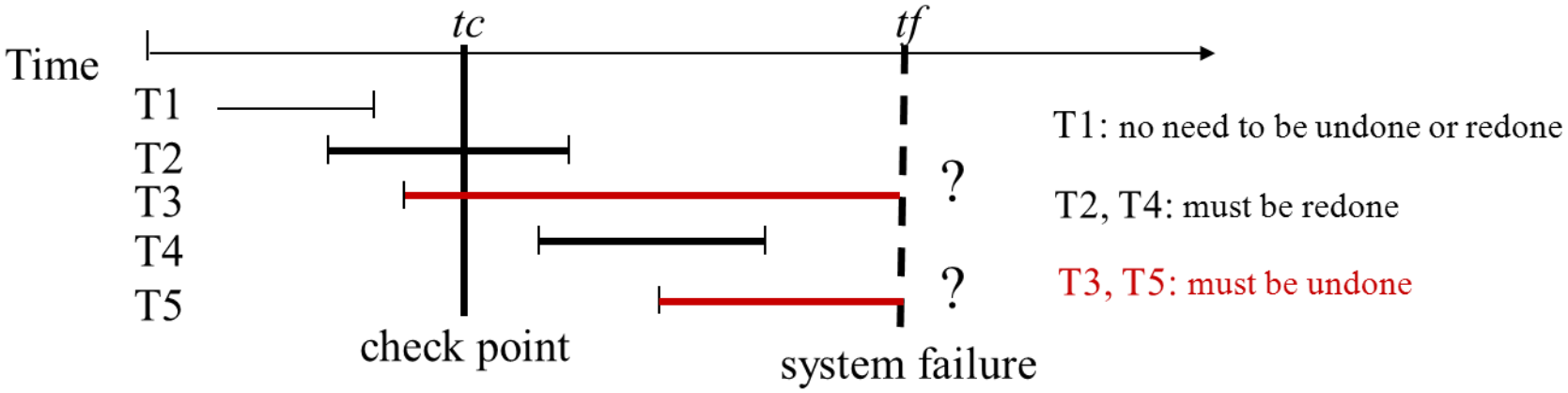
Unit 12

Database Recovery



Contents

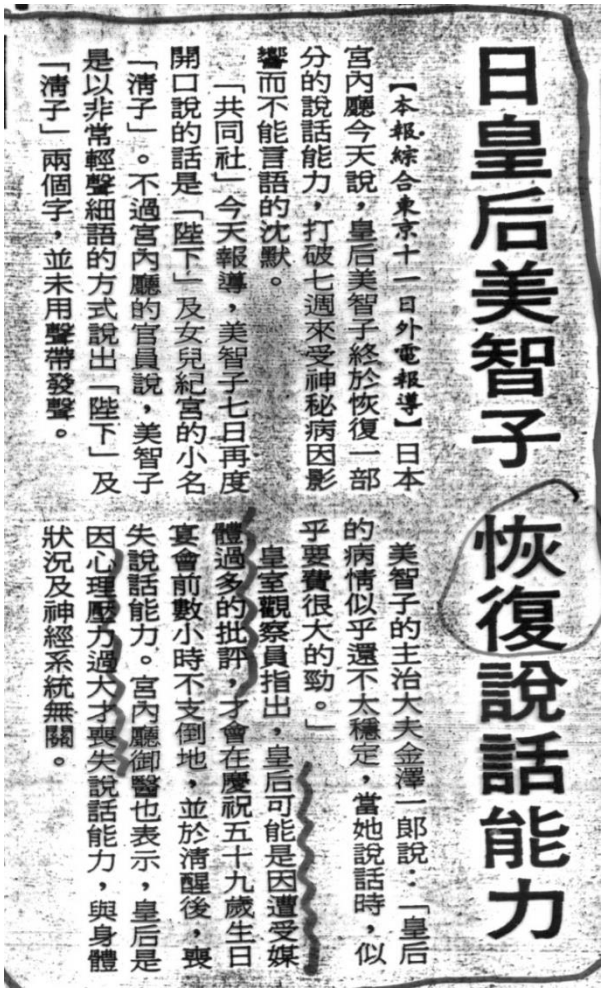
- 12.1 Introduction
- 12.2 Transactions
- 12.3 Transaction Failures and Recovery
- 12.4 System Failures and Recovery
- 12.5 Media Failures and Recovery



12.1 Introduction

Database Recovery: Introduction

- **The Problem of Database Recovery**
  - To restore the database to a **state** that is known to be **correct** after some failures.
- **Possible Failures**
  - programming errors, e.g. divide by 0, QTY < 0
  - hardware errors, e.g. disk crashed
  - operator errors, e.g. mounting a wrong tape
  - power supply, fire, ...
- **Principle of Recovery:**
  Backup is necessary



日皇后美智子 恢復說話能力

【本報綜合東京十一日外電報導】日本宮內廳今天說，皇后美智子終於恢復一部分的說話能力，打破七週來受神秘病因影響而不能言語的沈默。

「共同社」今天報導，美智子七日再度開口說的話是「陛下」及女兒紀宮的小名「清子」。不過宮內廳的官員說，美智子是以非常輕聲細語的方式說出「陛下」及「清子」兩個字，並未用聲帶發聲。

美智子的主治大夫金澤一郎說：「皇后的病情似乎還不太穩定，當她說話時，似乎要費很大的勁。」

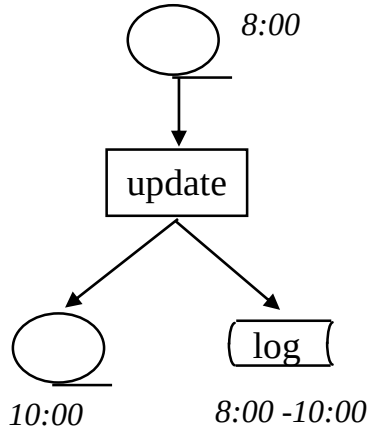
皇室觀察員指出，皇后可能是因遭受媒體過多的批評，才會在慶祝五十九歲生日宴會前數小時不支倒地，並於清醒後，喪失說話能力。宮內廳御醫也表示，皇后是因心理壓力過大才喪失說話能力，與身體狀況及神經系統無關。

Database Recovery (cont.)

- **Basic approach**
  1. Dump database periodically.
  2. Write a log record for every change.
     e.g. E#, old_value, new_value, …
  3. If a failure occurs:

     CASE1 : DB is damaged
     ==> archive copy + redo log = current DB.

     CASE2 : DB is not damaged but contents unreliable
     ==> undo some log.



12.2 Transactions

- unit of Work
- unit of Recovery
- unit of Concurrency (Unit 13)

Transactions: Concepts

- A logical unit of work.
- Atomic from the point of view of the end-user.
- An all-or-nothing proposition.

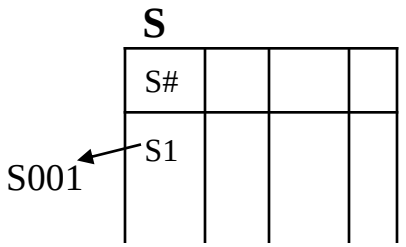
<e.g.>

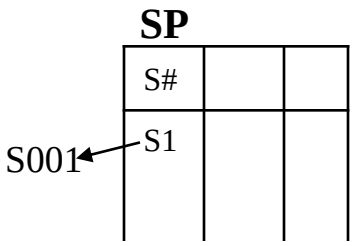
```
TRANSFER : PROC;  /* transfer account */
    GET (FROM, TO, AMOUNT);
    FIND UNIQUE (ACCOUNT WHERE ACC#=FROM);
    ASSIGN (BALANCE - AMOUNT) TO BALANCE;
    IF BALANCE < 0
      THEN
        DO;
        PUT ( 'INSUFFICIENCY FUNDS');
        ROLLBACK;
        END;
      ELSE
        DO;
        FIND UNIQUE (ACCOUNT WHERE ACC# = TO);
        ASSIGN (BALANCE + AMOUNT) TO BALANCE;
        PUT ('TRANSFER COMPLETE' );
        COMMIT;
        END;
END;
```

Transactions: Example

<e.g.> [CASCADE CHANGE ON S.S# TO SP.S#]

```
CHANGE: PROC OPTIONS (MAIN)
        EXEC  SQL WHENEVER SQLERROR GOTO UNDO;
        GET LIST (SX, SY);
 (i)    EXEC SQL UPDATE S
                  SET S# =: SY;
                  WHERE S# =:SX;
 (ii)   EXEC SQL UPDATE SP
                  SET S# =: SY;
                  WHERE S# =: SX;
        EXEC SQL COMMIT;
        GO TO FINISH;
UNDO: EXEC SQL ROLLBACK;
FINISH: RETURN;
END
```





Transactions: Structure

- Structure of a Transaction

```
BEGIN TRANSACTION;
    /* application specified sequence of operations*/
                .
                .
COMMIT;             /* signal successful termination    */
(or ROLLBACK;       /* signal unsuccessful termination*/)
```

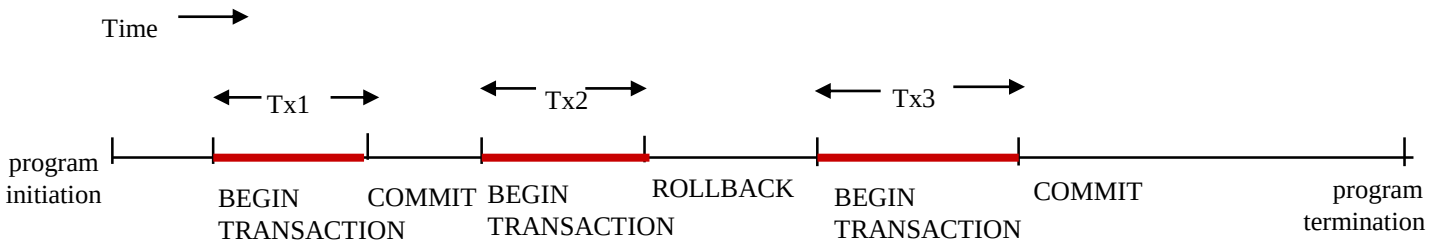
- Implicit

BEGIN TRANSACTION, COMMIT, ROLLBACK may be implicit:

Program initiation → BEGIN TRANSACTION
Normal termination → COMMIT
Abnormal termination → ROLLBACK

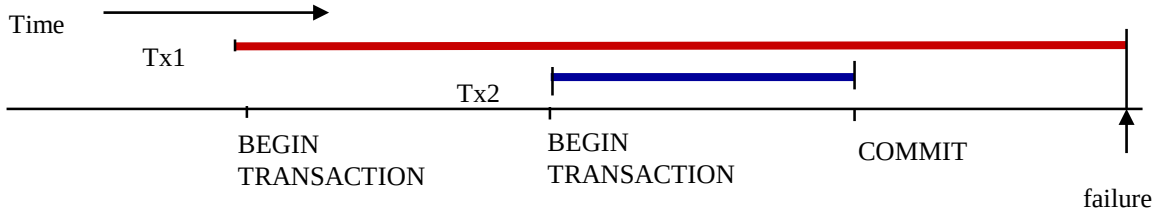
- **Program and Transaction:**

one program may contain several transactions.



Transactions: Manager

- **Transaction cannot be nested:**



  • Does **Tx2** need to be rolled back ?

- **Transaction Manager:**

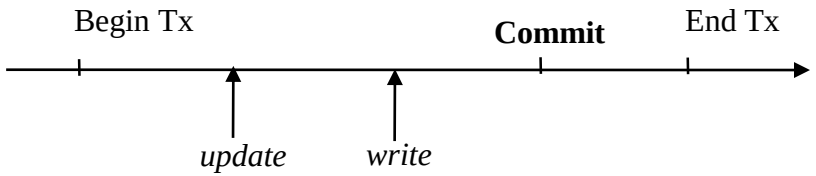
  Transaction should not be lost, or partially done, or done more than once

  <e.g.> Consider the CASCADE example,

  if the system crashed between two updates

  ==> the first update must be **undone** !

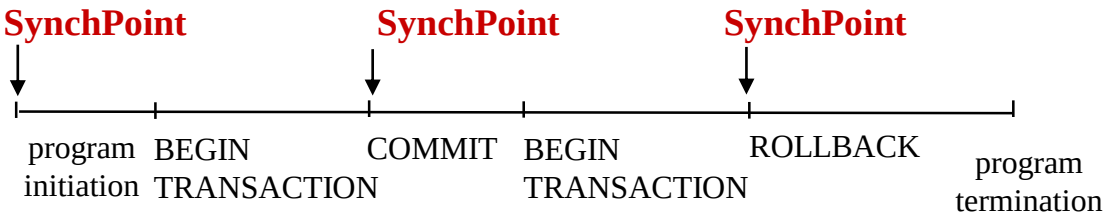
Transactions: Commit and Rollback



- **COMMIT:**
  - signal successful end-of-transaction.
  - all updates made by that transaction can now be made permanent. (e.g. buffer to disk)
- **ROLLBACK:**
  - signal unsuccessful end-of-transaction.
  - the database may be in an inconsistent state.
  - all update made by that transaction so far must be 'rolled back or undone'
- How to undone an update ?
  - system maintain a **log** or **journal** on tape or disk on which details of all update are recorded.

Transactions: Synchronization Point (SynchPoint)

- Represents the boundary between two consecutive transactions.
- Corresponds to the end of logical unit of work.
- A point at which the database is in a **state of consistency**.
- Established by COMMIT, ROLLBACK, and program initiation.



- When a **synchpoint** is established:
  - All updates since the previous **synchpoint** are committed **(COMMIT)** or undone **(ROLLBACK)**
  - All database positioning is lost. (e.g. cursor).
  - All record locks are released.

Types of Transaction Failure

- **Type 1 Transaction Failures:**
  - detected by the application program itself.
    e.g. Insufficient Funds (balance < 0)
  - How to handle ?
    Issue the ROLLBACK command after the detection. (ref. p.12-7)

  } Application program 處理

- **Type2 Transaction Failures:**
  - not explicitly handled by the application
    e.g. divide by zero, arithmetic overflow, ...

  } § 12.3

- **System Failures (Soft crash):**
  - affect all transactions currently in progress,
  - but do not damage the database. e.g. CPU failure.

  } § 12.4

- **Media Failures (Hard crash):**
  - damage the database.
  - affect all transactions currently using that portion.
    e.g. disk head crash.

  } § 12.5

12.3 Type 2 Transaction Failures and Recovery

Transaction Failures and Recovery

- **Transaction Failures:**

  failures caused by unplanned, abnormal program termination**.**

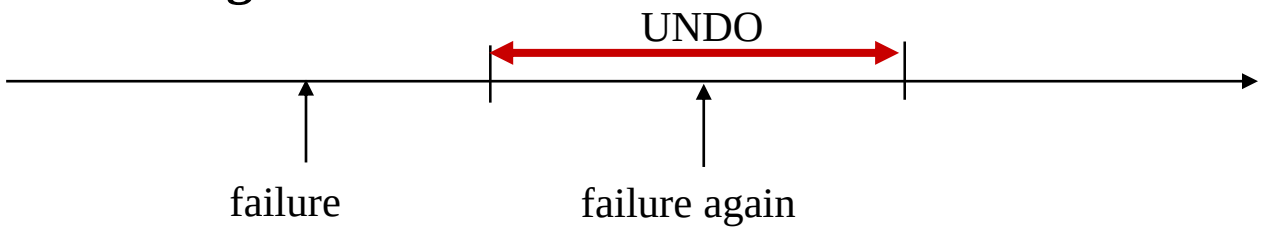
  <e.g.> arithmetic overflow
  divided by zero
  storage protection violation
  log overflow...

- **How to recover transaction failures ?**

  - System force a rollback.
  - the rollback is coordinated by **Recovery Manager**.
  - working backward through the log
    - to undo changes (replace new value by old value)
    - until the "**BEGIN TRANSACTION"** is encountered.

UNDO Logic and REDO Logic

- **UNDO Logic**



=> **cause the rollback procedure to be restarted from the beginning.**

  - **Idempotent Property** : [Gray '78]

UNDO (UNDO (UNDO (... (x))) = UNDO (x) for all x

i.e. undoing a given change any number of times is the same as undoing it exactly once.

- **REDO Logic**

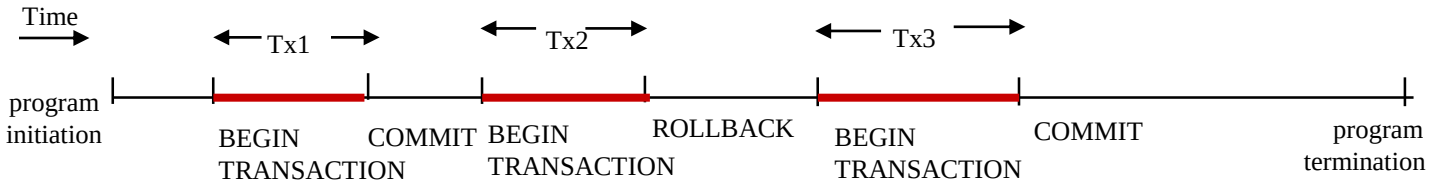
REDO (REDO (REDO (...(x))) = REDO (x) for all x.

Log

- **On-line log (active log) v.s. Off-line log (archive log) :**
  - log data: 200 million byte/day ==> infeasible to be stored entirely on-line
  - active log: stored on disk if full ==> dump to tape ==> archive log.
- **Log Compression**
  - Archive log can be compressed
    => reduce storage, and then increasing efficiency
  - How to compress archive log ?
    - log records for transactions that failed to commit can be deleted (since they have been rolled back).
    - old values are no longer needed for the transactions that did commit (since they will never have to be undone). 只可能做 redo
    - changes can be consolidated (only the final value is kept)

Log: 100 –10 90 r

100 -10 90 cancel



Long Transaction

- Transaction is unit of work, and unit of recovery.
  - Transaction should be short.

    => reduce the amount that has to be undone.
- long transaction => subdivided into multiple transactions.

  <e.g.> T_1: Update all supplier records, S.

  ⇩

  T_{11}: Update all supplier records for supplier name is 'A%'.

  T_{12}: Update all supplier records for supplier name is 'B%'.

  .

  .

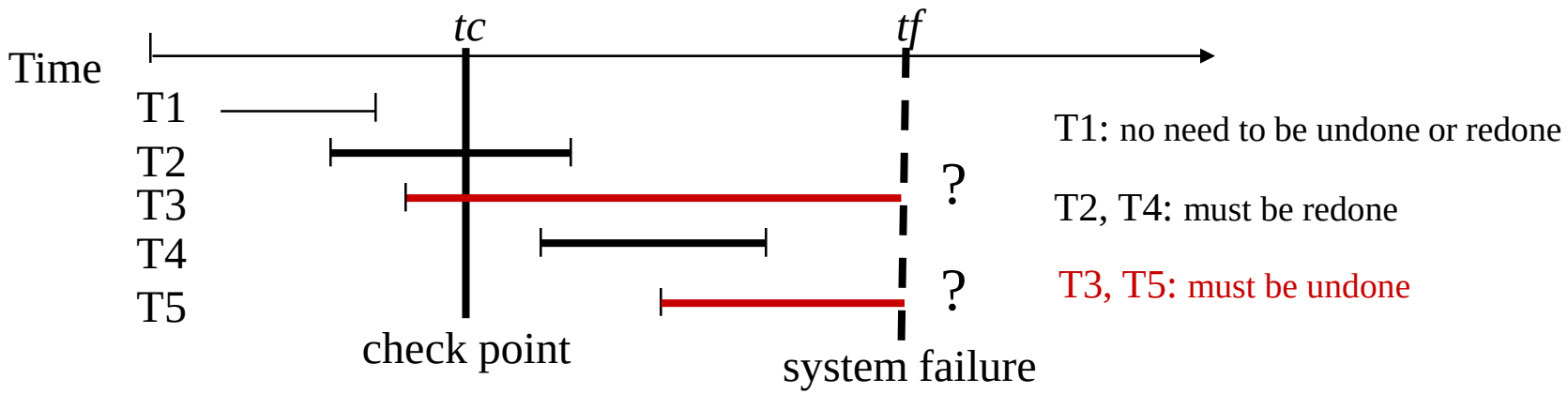
  $T_{1,26}$: Update all supplier records for supplier name is 'Z%'.

12.4 System Failures and Recovery

System Failures and Recovery

- **Critical point : contents of ~~main storage~~ are lost, in particular, the database buffers are lost. e.g. CPU failure.**
- **How to recover ?**
  (1) UNDO the transactions in progress at the time of failure. e.g. T_3 ,T_5
  (2) REDO the transactions that successfully complete but did not write to the physical disk.

- <e.g.>



System Failures and Recovery

e.g. T_3, T_5

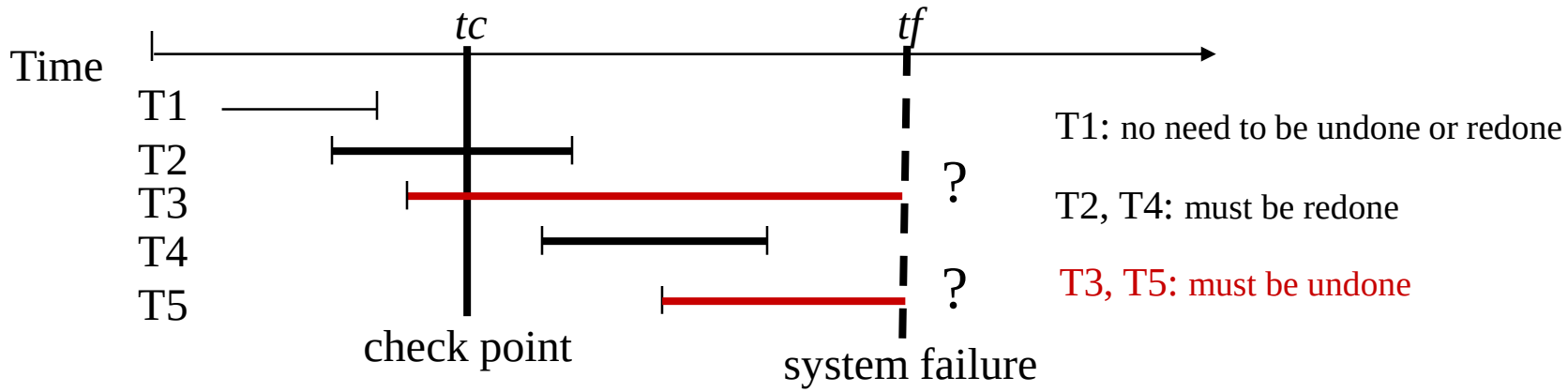
- **How does the system know: which transaction to redo and which to undo?**
  - **<1> Taking a ~~check point~~:**
    - at certain prescribed intervals
    - involves:

      (1) writing the contents of the database buffers out to the physical database. e.g. disk

      e.g. T_1

      (2) writing a special checkpoint record (contains a list of transactions which are in progress) e.g. {T2, T3} in progress

<e.g.>



System Failures and Recovery (cont.)

<2> Decide undo and redo list

Decide the undo list and redo list by the following procedure :

STEP1:

UNDO-list = list of transactions given in the checkpoint record = {T2, T3}

REDO-list = { }

STEP2:

Search **forward** through the log, starting from the checkpoint, to the end of log:

- if a 'BEGIN TRANSACTION' is found => add to UNDO-list {T2, T3, T4, T5}
- if a 'COMMIT' is found => remove from UNDO-list to REDO-list

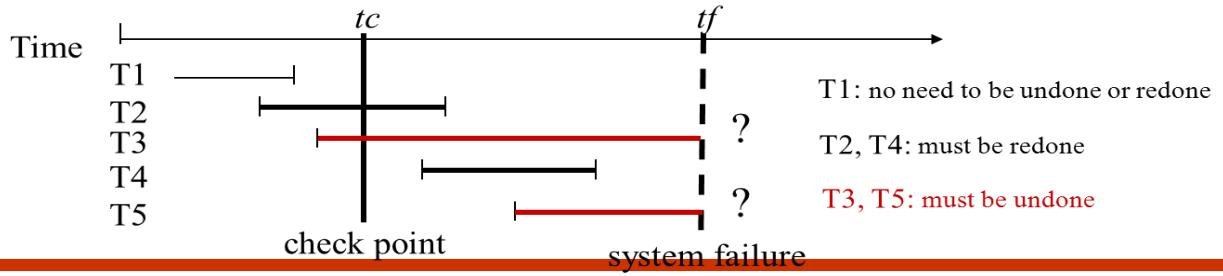
UNDO-list = {T3, T5} 做一半的，要**undo**

REDO-list = {T2, T4} 應該已做完，不確定有無 **write to disk**

<3> Undo: System works **backward** through the log, **undoing** the UNDO-List.

<4> Redo**:** System then works **forward** through the log, **redoing** the REDO-List

<e.g.>

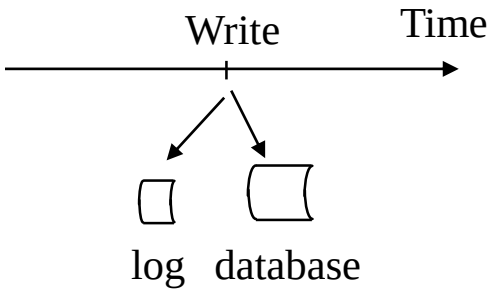


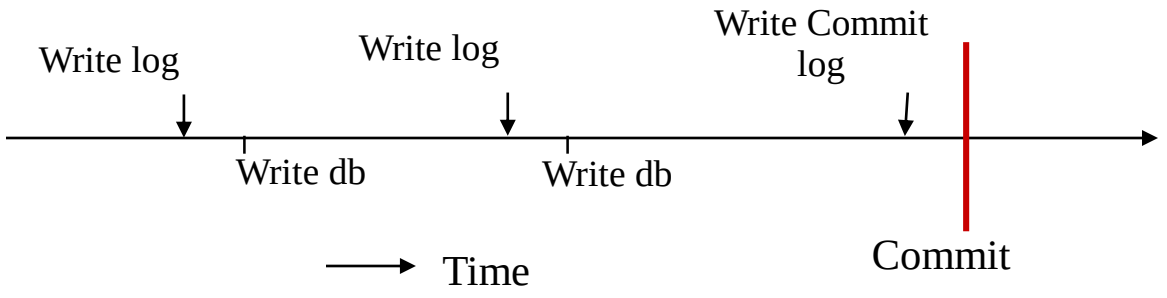
Write-Ahead Log Protocol

- **Write-Ahead Log Protocol (i.e. Log first protocol)**

  Note: '**write a change to database**' and '**write the log record to log**' are two distinct operations
  => **failure** may occur between them!

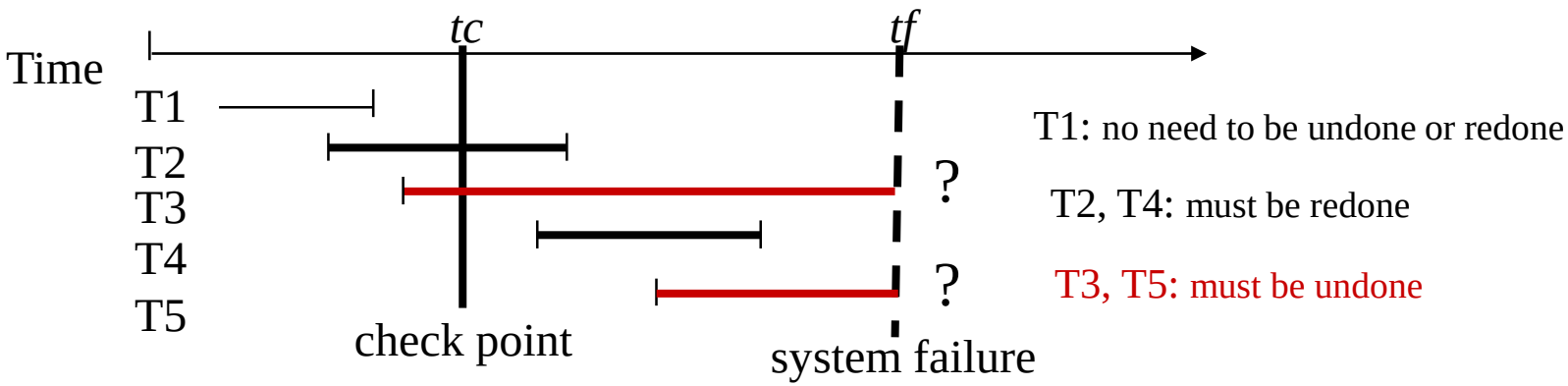
  - Before writing a record to physical database, the **log** record must first be written to physical log.
  - Before committing a transaction, all **log** records must first be written to physical log.

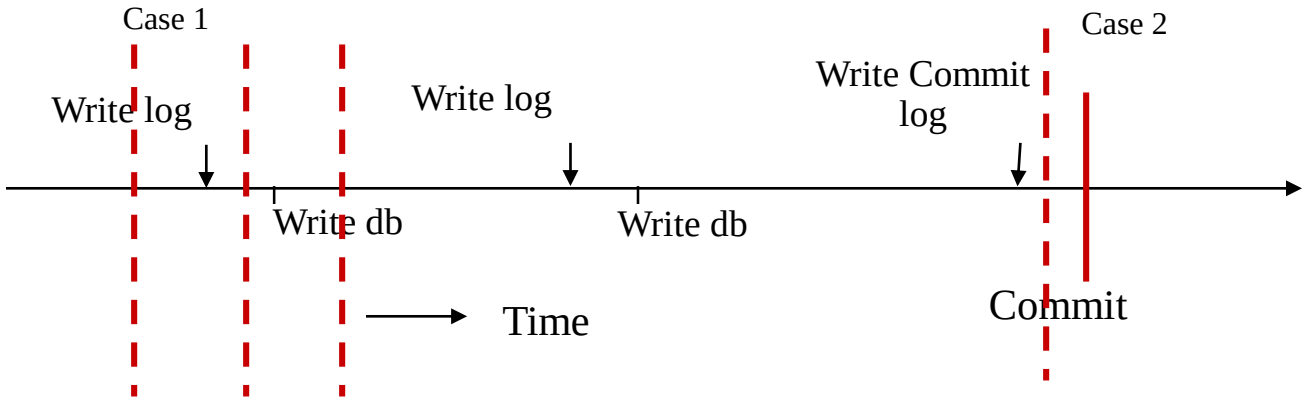




Write-Ahead Log Protocol (cont.)

- **Why log need to write ahead?** (Think!)





12.5 Media Failures and Recovery

Types of Transaction Failure

- **Type 1 Transaction Failures:**
  - detected by the application program itself.
    e.g. Insufficient Funds (balance < 0)
  - How to handle ?
    Issue the ROLLBACK command after the detection. (ref. p.12-7)

  } Application program 處理

- **Type2 Transaction Failures:**
  - not explicitly handled by the application
    e.g. divide by zero, arithmetic overflow, ...

  } § 12.3

- **System Failures (Soft crash):**
  - affect all transactions currently in progress,
  - but do not damage the database. e.g. CPU failure.

  } § 12.4

- **Media Failures (Hard crash):**
  - damage the database.
  - affect all transactions currently using that portion.
    e.g. disk head crash.

  } § 12.5

Media Failures and Recovery

- **Critical point:**

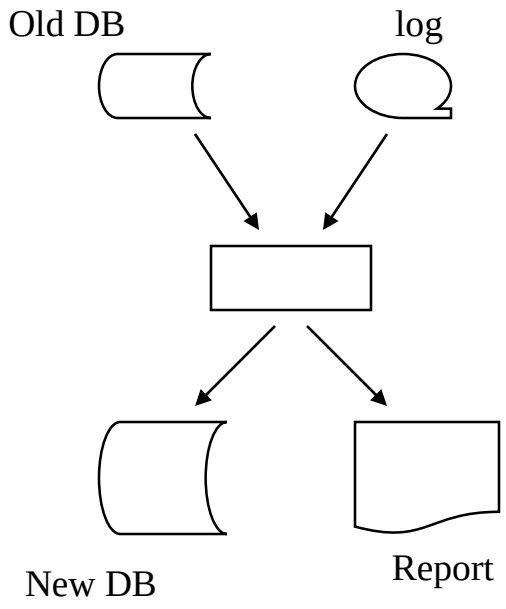
  Some portion of the **secondary storage** is damaged.

- **How to recover?**

  (1) load the database to new device from the most recent **archive copy** (old DB.)

  (2) use the log (both active and archive) to **redo** all the transactions that are completed since that dump was taken.

  Note: Assume **log** dose not fail.
  (Duplex log to avoid log failure.)



end of unit 12