ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ

Συστήματα Διαχείρισης Βάσεων Δεδομένων

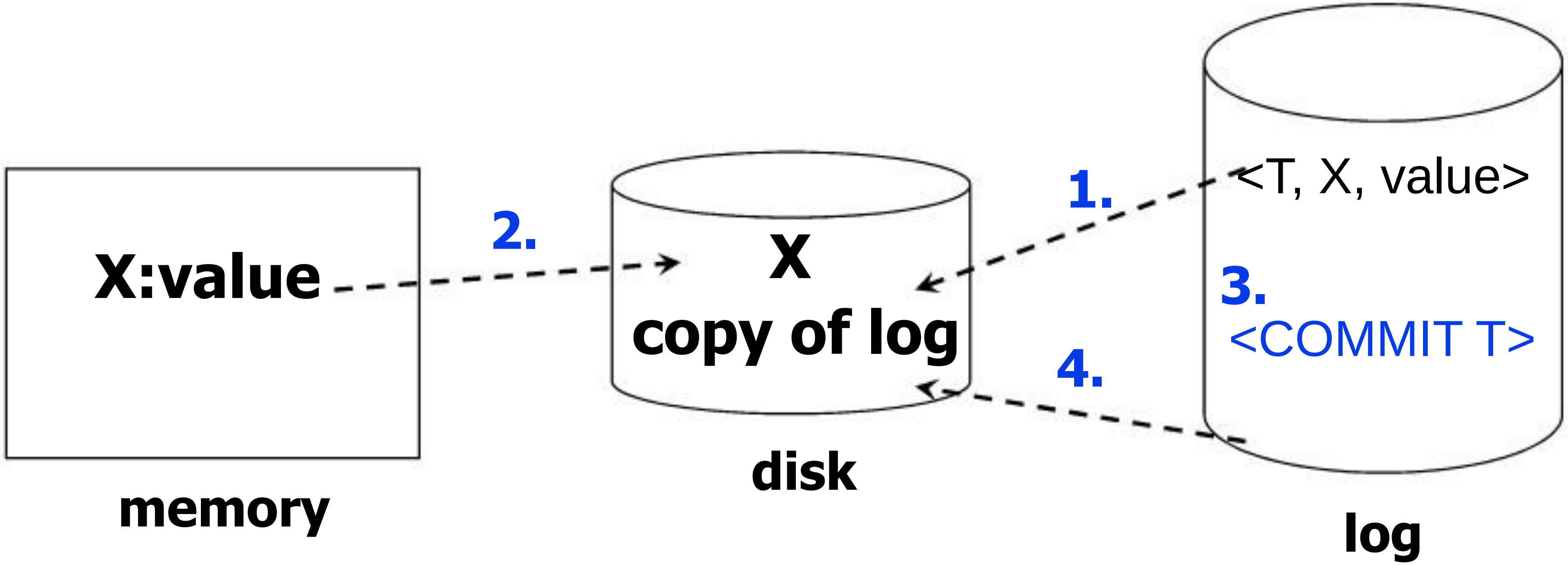
Φροντιστήριο 9: Transactions - part 1

Δημήτρης Πλεξουσάκης
Τμήμα Επιστήμης Υπολογιστών

Tutorial on Undo, Redo and Undo/Redo Logging

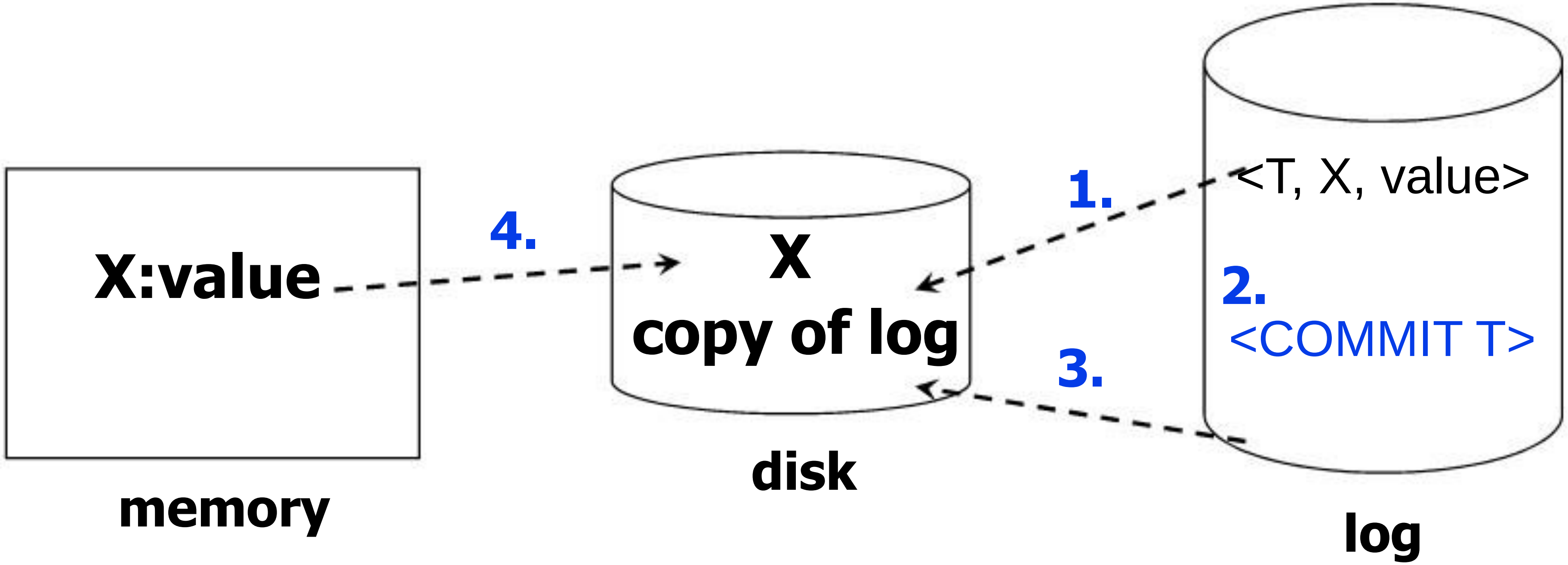
Quick Review: Undo vs. Redo Logging

- General Idea: In case of failure
  - Undo: cancels incomplete, ignores complete transactions
  - Redo: ignores incomplete, re-executes complete transactions
- Methodology: Undo



Quick Review: Undo vs. Redo Logging

- General Idea: In case of failure
  - Undo: cancels incomplete, ignores complete transactions
  - Redo: ignores incomplete, re-executes complete transactions
- Methodology: Redo



Quick Review: Undo vs. Redo Logging

- Checkpointing:

Undo:

1. Write <START CKPT (T_1,…,T_k)>
2. Flush the log.
3. Wait until all T_1,…T_k commit or abort.
4. Write <END CKPT>.
5. Flush the log.

Redo:

1. Write <START CKPT (T_1,…,T_k)>
2. Flush the log.
3. Write to disk all elements of transactions that had already committed before step 1.
4. Write <END CKPT>.
5. Flush the log.

Quick Review: Undo vs. Redo Logging

- Recovery:

Undo:

- Complete checkpoint: scan backwards as far as the START CKPT record.
- Incomplete checkpoint: scan backwards as far as the earliest of $T_1,\ldots T_k$.

Redo:

- Completed checkpoint: start scanning from the earliest of $T_1,\ldots T_k$.
- Incomplete checkpoint: search for previous complete checkpoint.

Example 1: Undo Recovery - Case 1

```
<START T1>
<T1, A, 5>
<START T2>
<T2, B, 10>
<START CKPT(T1,T2)>
<T2, C, 15>
<START T3>
<T1, D, 20>
<COMMIT T1>
<T3, E, 25>
<COMMIT T2>
<END CKPT>
→ <T3, F, 30>
```

- System crash after checkpoint
  - ■ Start scanning from the end.
  - ■ T3 is an incomplete transaction and must be undone. We set F = 30.
  - ■ We find an <END CKPT>. Therefore, we will stop scanning at the START CKPT.
  - ■ T2 committed. Do not touch!
  - ■ T3 incomplete. We set E = 25.
  - ■ No other transactions that started, but did not commit, until the START CKPT. End of scanning.

Example 1: Undo Recovery - Case 2

<START T1>
<T1, A, 5>
<START T2>
<T2, B, 10>
<START CKPT(T1,T2)>
<T2, C, 15>
<START T3>
<T1, D, 20>
<COMMIT T1>
→ <T3, E, 25>
<COMMIT T2>
<END CKPT>
<T3, F, 30>

- System crash during checkpoint
  - ■ Start scanning from the end.
  - ■ T3 incomplete. We set E = 25.
  - ■ T1 committed. Do not touch!
  - ■ T2 incomplete. We set C = 15.
  - ■ We find <START CKPT(T1,T2)>. The only possible incomplete are T1, T2. Still, T1 committed. Therefore, we continue until we meet <START T2>.
  - ■ T2 incomplete. We set B = 10.
  - ■ We meet <START T2>. End of scanning.

Example 1: Undo Recovery - Case 2 1/2

```
<START T1>
<T1, A, 5>
<START T2>
<T2, B, 10>
<START CKPT(T1,T2)>
<T2, C, 15>
<START T3>
<T1, D, 20>
<COMMIT T1>
<T3, E, 25>
<COMMIT T2>
<END CKPT>
<T3, F, 30>
```

- System crash during checkpoint
  - It is the same case as before.
  - We find <START CKPT(T1,T2)>. The only possible incomplete are T1, T2. Therefore, we continue until we meet all <START Ti>, where i = 1,2.

Example 2: Redo Recovery - Case 1

<START T1>

<T1, A, 5>

<START T2>

<COMMIT T1>

<T2, B, 10>

<START CKPT(T2)>

<T2, C, 15>

<START T3>

<T3, D, 20>

<END CKPT>

<COMMIT T2>

<COMMIT T3>

- System crash after checkpoint
  - ■ We make a quick scan from the end.
  - ■ We find <END CKPT> so we only need to care with those mentioned in the beginning record of the checkpoint and the ones started after that. That is T2, T3, and not T1.
  - ■ We start from the earliest transaction mentioned in the beginning record of the checkpoint and continue downwards.
  - ■ T2 committed, it must be redone. B = 10.
  - ■ T2 committed, it must be redone. C = 15.
  - ■ T3 committed, it must be redone. D = 20.

Example 2: Redo Recovery - Case 1 1/2

<START T1>

<T1, A, 5>

<START T2>

<COMMIT T1>

<T2, B, 10>

<START CKPT(T2)>

<T2, C, 15>

<START T3>

<T3, D, 20>

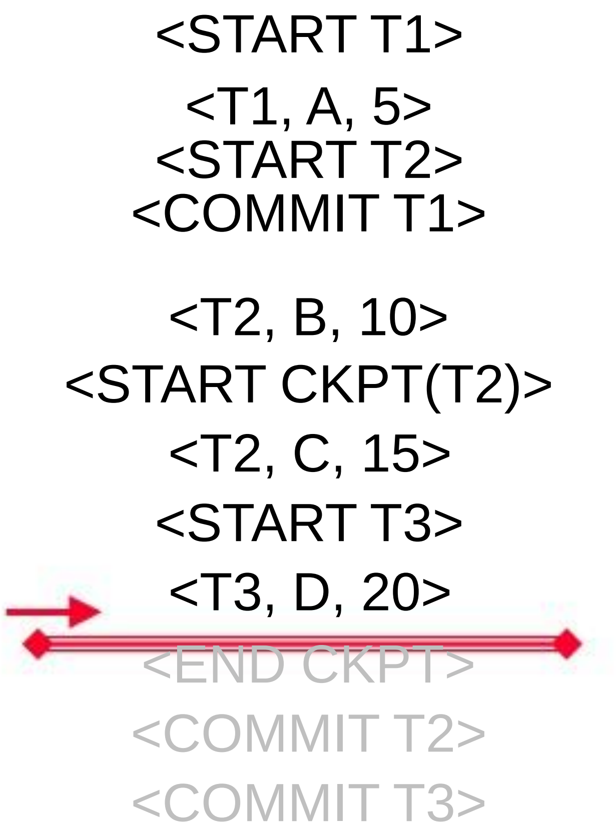
<END CKPT>

<COMMIT T2>

<COMMIT T3>

- System crash after checkpoint
  - ■ Now T3 is not a committed transaction and, as a result, we must not redo it.
  - ■ At the end of the recovery process, we add an <ABORT T3> record to the log.

Example 2: Redo Recovery - Case 2



<START T1>
<T1, A, 5>
<START T2>
<COMMIT T1>
<T2, B, 10>
<START CKPT(T2)>
<T2, C, 15>
<START T3>
<T3, D, 20>
<END CKPT>
<COMMIT T2>
<COMMIT T3>

- System crash during checkpoint
  - ■ We must search back to the previous checkpoint and find its list of active transactions.
  - ■ In this case there is no previous checkpoint. We start from the beginning of the log.
  - ■ Only T1 is committed and must be redone. A = 5.
  - ■ At the end of the recovery process, we add <ABORT T2>, <ABORT T3> to the log.

Example 3

```
<START T1>
<T1, C, 35>
<T1, D, 450>
<START T2>
<T2, C, 18>
<T2, B, 12>
<T1, D, 500>
<COMMIT T1>
<START CKPT (T2)>
<END CKPT>
<T2, D, 18>
<START T3>
<T3, C, 45>
<T3, E, 2>
<T2, A, 10>
<COMMIT T3>
<COMMIT T2>
```

- The following values are stored in the disk: A=10, B=12, C=45, D=65, E=2.
- Given the log shown
  - could this be an undo log?
  - No, because, for an undo log, all transactions mentioned at the start of the checkpoint must commit before its ending.
  - could this log result in the previously mentioned values for A, B, C, D and E?

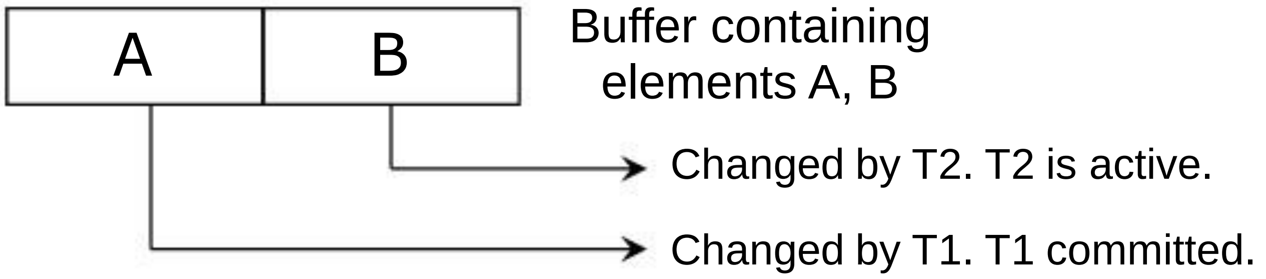
Example 4

```
<START T1>
<T1, C, 35>
<T1, D, 450>
<START T2>
<T2, C, 18>
<T2, B, 12>
<T1, D, 500>
<COMMIT T1>
<START CKPT (T2)>
<END CKPT>
<T2, D, 18>
<START T3>
<T3, C, 45>
<T3, E, 2>
<T2, A, 10>
<COMMIT T3>
<COMMIT T2>
```

- The following values are stored in the disk: A=10, B=12, C=45, D=65, E=2.
- Given the log shown
  - could this be a redo log?
  - Yes.
  - could this log result in the previously mentioned values for A, B, C, D and E?
  - No. The problem is the value of D. Since T1 committed before the checkpoint and is not mentioned as active, we are sure that D = 500 for the moment. T2 also accesses D. Maybe the changes were written or maybe not. In either case, D □ 65.

A Point of Caution

- What if the size of the elements are not equal to the size of memory buffers?
- For instance, if a buffer contains element A that was changed by a committed transaction and another element B that was changed by a transaction that has not yet had its COMMIT record written to disk.
- During checkpointing both undo and redo put contradictory requirements: the buffer must be copied to disk because of A, but also forbidden because of B.
- Solution: Undo/Redo Logging


A
B
Buffer containing elements A, B
Changed by T2. T2 is active.
Changed by T1. T1 committed.

Undo/Redo Logging

- Rule: Before modifying any element on disk, the log records must first be flushed.
- Checkpointing: Remember that we write an <END CKPT> only after all dirty buffers are written to disk (i.e., we flush all buffers, not just those written by committed transactions as in redo).
- Recovery: We proceed first backward to find checkpoints, forward to redo history and backward to undo uncommitted transactions, as appropriate.

Example 5: Undo/Redo Recovery - Case 1

```
<START T1>
<T1, A, 4, 5>
<START T2>
<COMMIT T1>
<T2, B, 9, 10>
<START CKPT(T2)>
<T2, C, 14, 15>
<START T3>
<T3, D, 19, 20>
<END CKPT>
<COMMIT T2>
<COMMIT T3>
```

- System crash after checkpoint
  - There is no need to look prior to the <START CKPT …> record
  - T1 is assumed completed and stored. We ignore it.
  - T2 and T3 are redone.

Example 5: Undo/Redo Recovery - Case 2

```
<START T1>
<T1, A, 4, 5>
<START T2>
<COMMIT T1>
<T2, B, 9, 10>
<START CKPT(T2)>
<T2, C, 14, 15>
<START T3>
<T3, D, 19, 20>
<END CKPT>
<COMMIT T2>
<COMMIT T3>
```

- System crash after checkpoint
  - As before but at the end we redo T2 and undo T3

Τέλος Ενότητας

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ
2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Κρήτης**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ
2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Σημειώματα

Σημείωμα αδειοδότησης

Σημείωμα Αναφοράς

Copyright Πανεπιστήμιο Κρήτης, Δημήτρης Πλεξουσάκης. «**Συστήματα Διαχείρισης Βάσεων Δεδομένων. Φροντιστήριο 9: Transactions - part 1**». Έκδοση: 1.0. Ηράκλειο/Ρέθυμνο 2015. Διαθέσιμο από τη δικτυακή διεύθυνση: http://www.csd.uoc.gr/~hy460/