Data Management

Maurizio Lenzerini

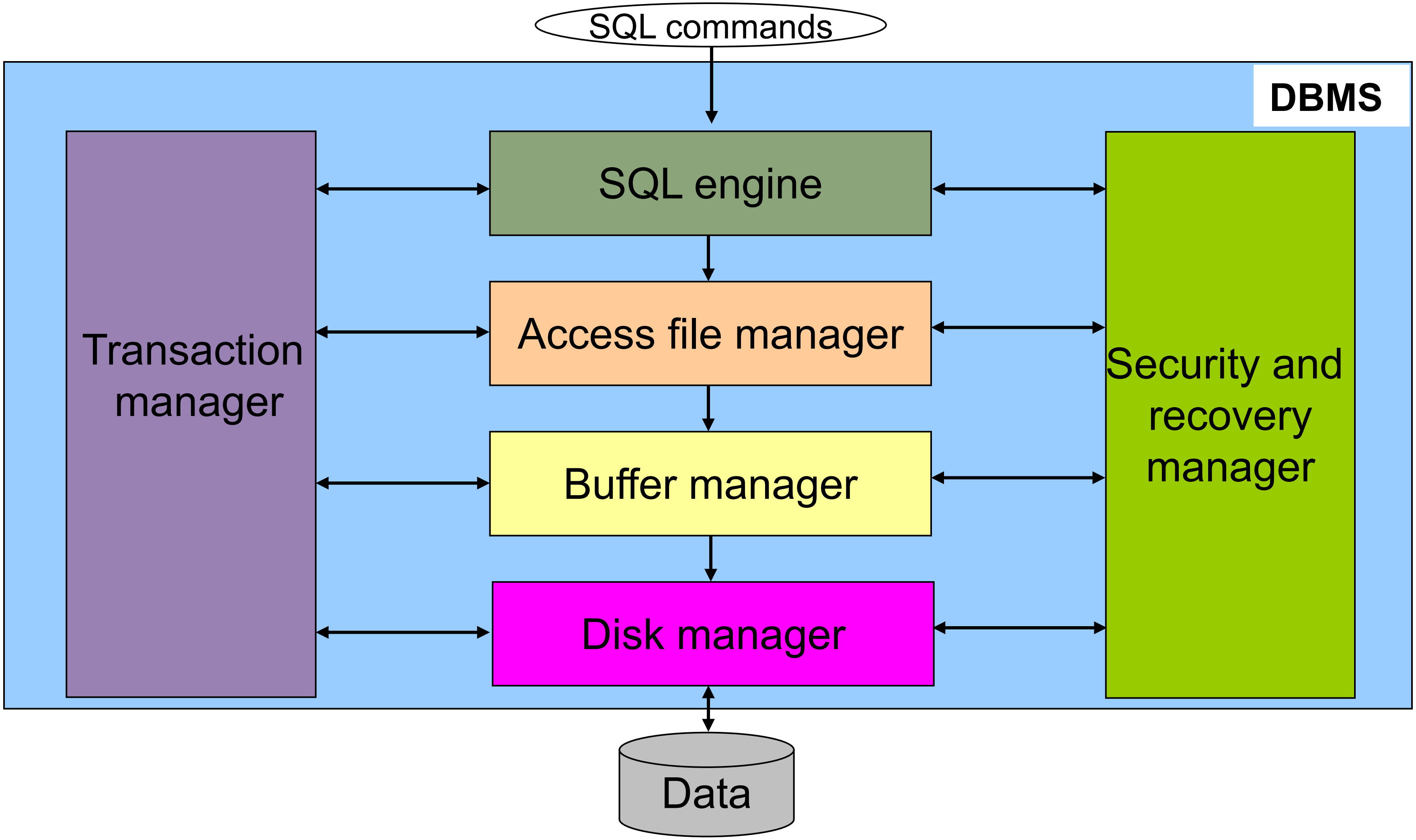
Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
Università di Roma "La Sapienza"

Academic Year 2020/2021

Part 4
Recovery Management

http://www.dis.uniroma1.it/~lenzerin/index.html/?q=node/53

Architecture of a DBMS



4. Recovery management

The recovery manager

The transaction manager is mainly concerned with isolation and consistency, while the recovery manager is mainly concerned with atomicity and persistency.

It is responsible for:

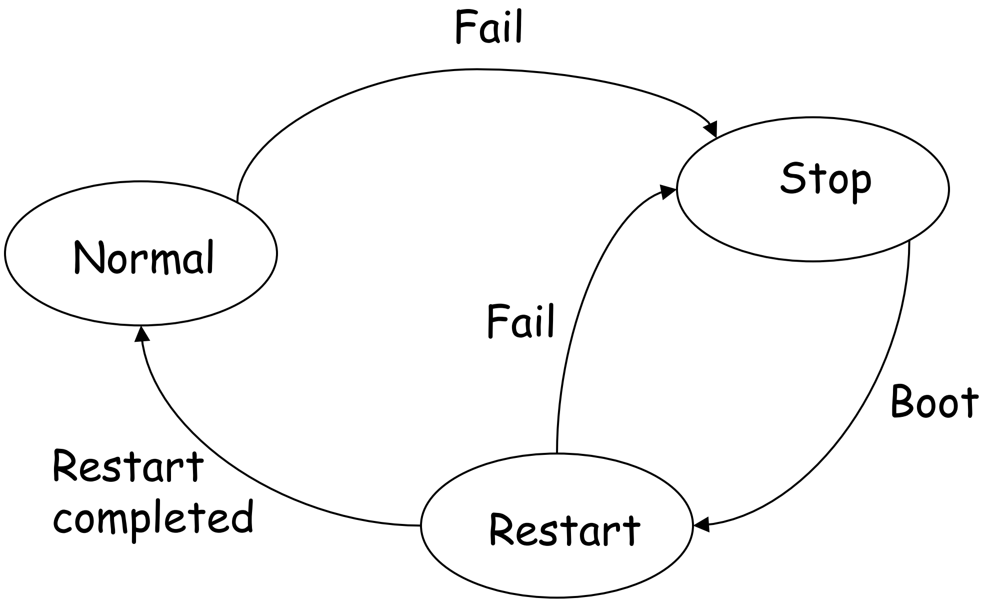
- Beginning the execution of transactions
- Committing transactions
- Executing the rollback of transactions
- Restore a correct state of the database following a fault condition

It uses a special data structure, called log file

Failure types

- System failures
  - System crash:
    - We loose the buffer content, not the secondary storage content
  - System error or application exception
    - E.g. division by zero
  - Local error conditions of a transaction
  - Concurrency control
    - The scheduler forces the rollback of a transaction
- Storage media failures
  - Disk failures
    - We loose secondary storage content, but not the log file content
  - Catastrophic events”
    - Fire
    - Flooding
    - Etc…

Failure recovery: the fail-stop model



Fail

Stop

Normal

Fail

Boot

Restart completed

Restart

The log file

- The log file (or, simply, the log) records the actions of the various transactions in a stable storage (stable means “failure resistant”)
- The stable storage is an abstraction: stability is achieved through replication
- Read and write operations on the log are executed as the operations on the database, i.e., through the buffer. Note that writing on the stable storage is generally done through “force” (synchronously)
- The physical organization of the log can be based on:
  - Tapes
  - Disk (perhaps coupled with tapes)
  - Two replicated disks

The structure of log

- The log is a sequential file (as we said, assumed to be failure-free). The operations on the log are: append a record at the end, scan the file sequentially forward, scan backward.

- The log records the actions of the transactions, in chronologically order.

- Two types of records in the log:
  - Transaction records (begin, insert, delete, update, commit, abort)
  - System records (checkpoint, dump)

- Please, do not confuse the transaction actions with the actions on the secondary storage. In particular, the actions of the transactions are assumed to be executed when they are recorded in the log (even if their effects are not registered yet in the secondary storage)

The transaction records

O = element of the DB

AS = After State, value of O after the operation

BS = Before State, value of O before the operation

For each transaction T, the transaction records are stored in the log as follows:

– **`begin`**:	B(T)
– **`insert:`**	I(T,O,AS)
– **`delete`**:	D(T,O,BS)
– **`update`**:	U(T,O,BS,AS)
– **`commit:`**	C(T)
– **`abort:`**	A(T)

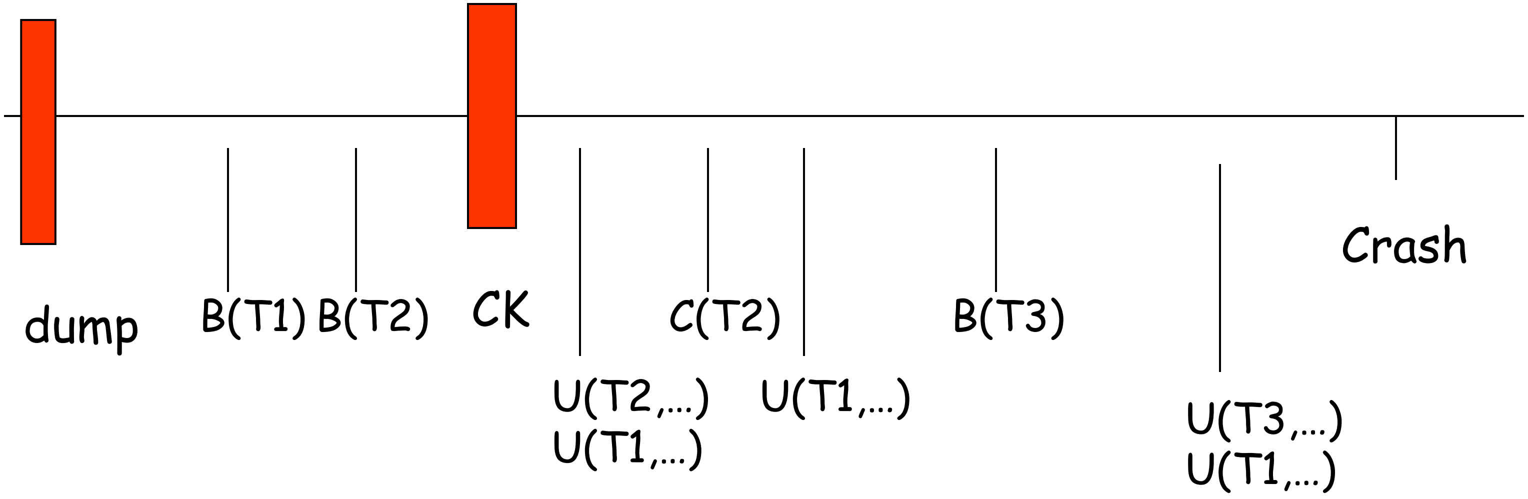
Checkpoint

- The goal of the checkpoint is to register in the log the set of active transactions T1, …, Tn, so as to differentiate them from the committed transactions
- The checkpoint (CK) operation executes the following actions:
  - For each transaction committed after the last checkpoint, their buffer pages are copied into the secondary storage (through flush)
  - A record CK(T1,…Tn) is written on the log (through force), where T1,…Tn identify all active transactions that are uncommitted
- It follows that:
  - For each transaction T such that Commit(T) *precedes* CK(T1,…,Tn) in the log, we can avoid the "redo" in case of failure
- The checkpoint operation is executed periodically, with fixed frequency

Dump

- The dump is a copy of the entire state of the DB
- The dump operation is executed offline (all transactions are suspended)
- It produces a backup, i.e., the DB is saved in stable storage
- It writes (through force) a dump record in the log

Example: log with checkpoint and dump



The Undo operation

- Restore the state of an element O at the time preceding the execution of an action
- `update`, `delete`:
  - assigns the *BS value to* O
- `insert`:
  - delete O

The Redo operation

- Restore the state of an element O at the time *following* the execution of an action

- `insert`, `update`:
  - assigns the value AS to O

- `delete`:
  - delete O

Atomicity of transactions

- The outcome of a transaction is established when either the Commit(T) record or the Abort(T) record is written in the log
  - The Commit(T) record is written synchronously from the buffer to the log
  - The Abort(T) record is written asynchronously from the buffer to the log (the recovery manager does not need to know immediately that a transaction is aborted)
- When a failure occurs, for a transaction
  - Uncommitted: since atomicity has to be ensured, in general we may need to undo the actions, especially if there is the possibility that the actions have been executed on the secondary storage → Undo
  - Committed: we need to redo the actions, to ensure durability → Redo

Writing records in the log

The recovery manager follows this rule:

- ***WAL*** **(write-ahead log)**
  - The log records are written from the buffer to the log before the data records involved in the action are written in secondary storage
  - This is important for the effectiveness of the Undo operation, because the old value can always be written back to the secondary storage by using the BS value written in the log. In other words, WAL allows to undo write operations executed by uncommitted transactions

Writing records in the log

The recovery manager follows this rule:

- ***Commit-Precedence***
  - The log records are written from the buffer to the log before the commit of the transaction (and therefore before writing the commit record of the transaction in the log)
  - This is important for the effectiveness of the Redo operation, because if a transaction committed before a failure, but its pages have not been written yet in secondary storage, we can use the AS value in the log to write such pages. In other words, the Commit-Precedence rule allows committed transactions whose effects have not been registered yet in the database to be redone.

Writing in secondary storage

For each operation

- Update
- Insert
- Delete

The recovery manager must decide on the strategy for writing in secondary storage

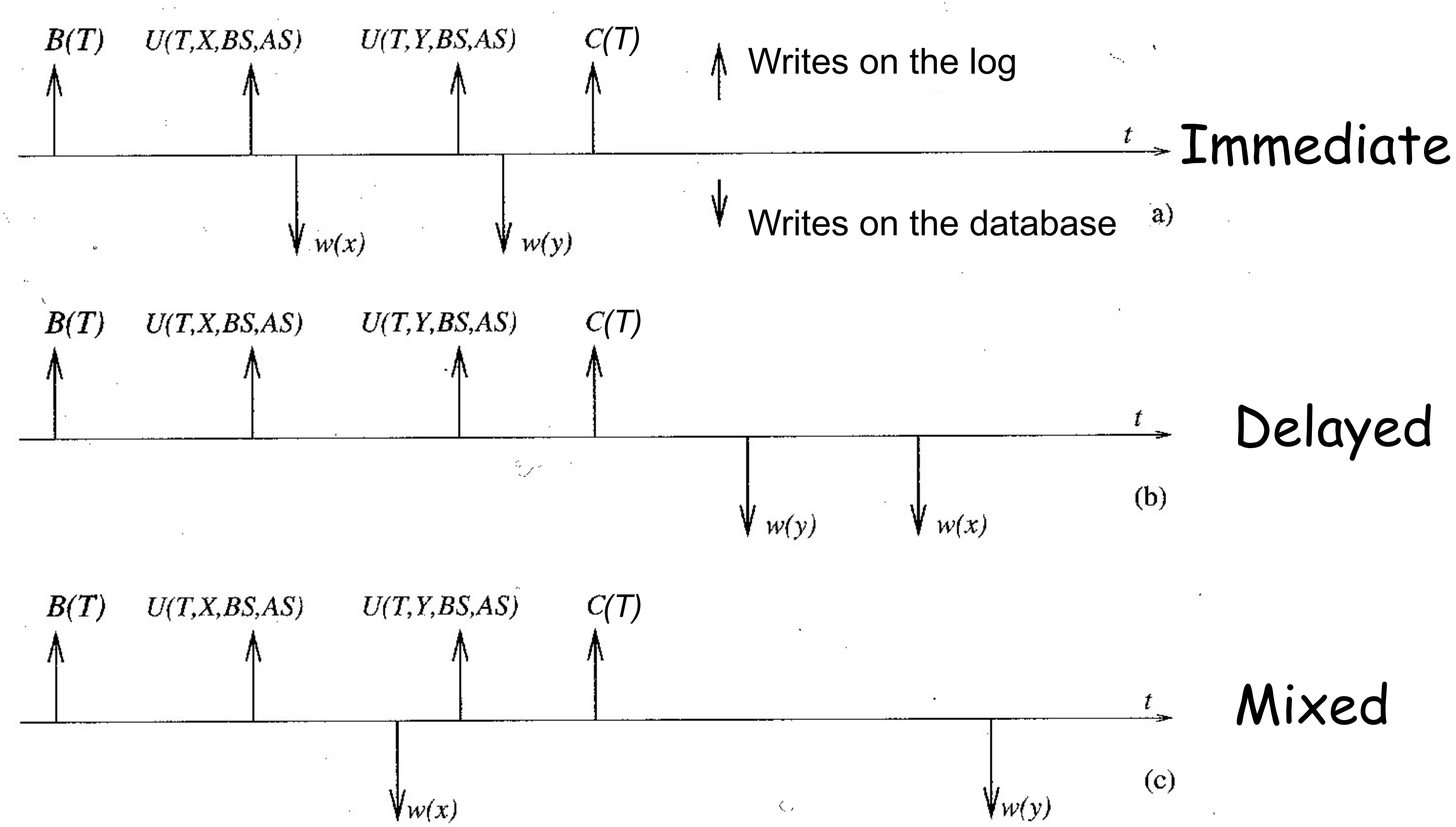
In the following, we concentrate on update, but similar considerations hold for the other operations

Writing in secondary storage

There are three possible methods for writing values into the secondary storage, all coherent with the WAL and the commit-precedence rules

- Immediate effect
  - The update operations are executed immediately on the secondary storage after the corresponding records are written in the log
  - The buffer manager writes the effect of an operation by a transaction T on the secondary storage before writing the commit record of T in log
  - It follows that all the pages of the DB modified by a transaction are certainly written in the secondary storage → REDO not needed
- Delayed effect
  - The update operations by a transaction T are executed on the secondary storage only after the commit of the transaction, i.e., only after the commit record of T has been written in the log → UNDO not needed
  - As usual, the log records are written in the log before the corresponding data are written in secondary storage
- Mixed effect
  - For an operation O, both the immediate effect and the delayed effect are possible, depending on the choice of the buffer manager (both REDO and UNDO needed)

Examples



↑ Writes on the log

↓ Writes on the database

Immediate

Delayed

Mixed

Two types of recovery

Depending on the type of failure…

- In case of system failure:
  - Warm restart

- In case of disk failure:
  - Cold restart

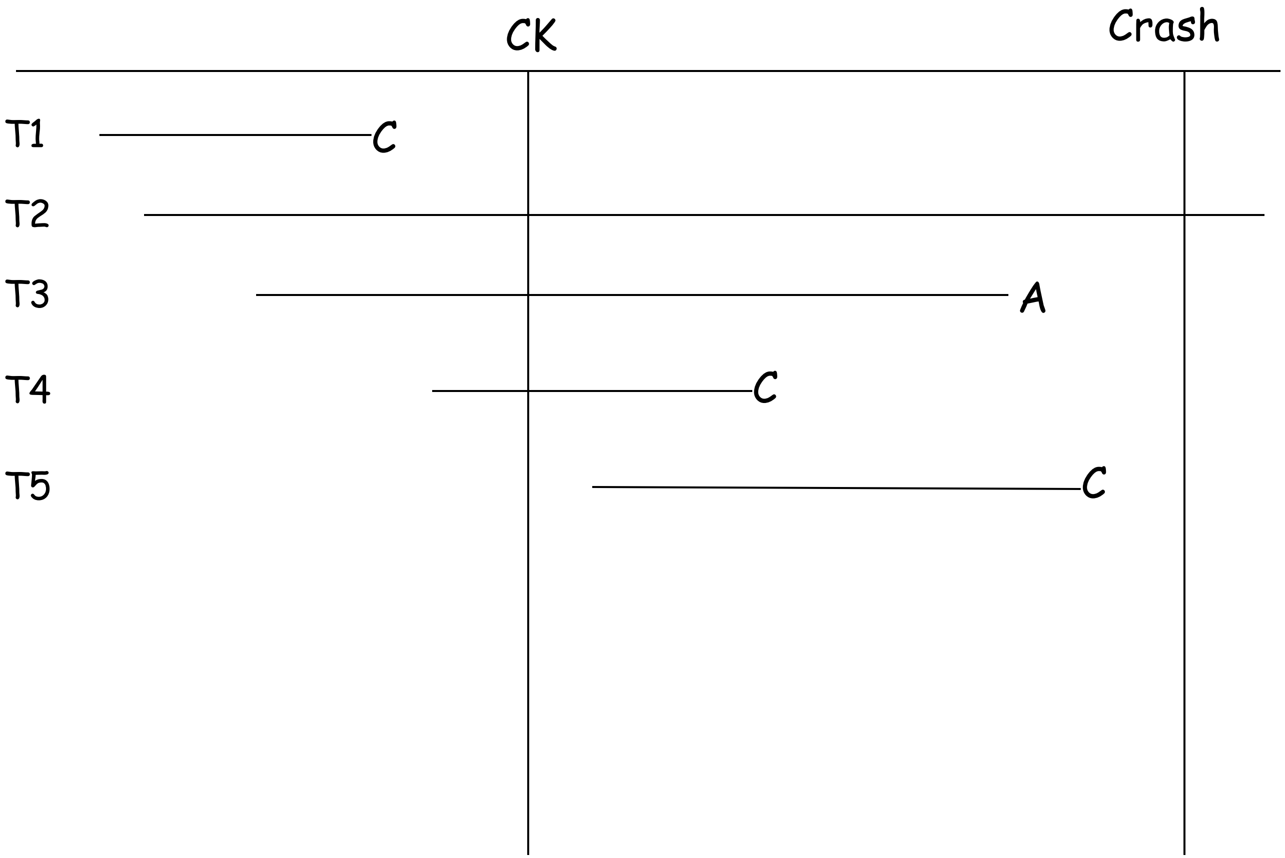
Warm restart

We will assume the mixed effect strategy. The warm restart is constituted by 5 steps:

1. We go backward through the log until the most recent checkpoint record in the log
2. We set s(UNDO) = { active transactions at checkpoint } s(REDO) = { }
3. We go forward through the log adding to s(UNDO) the transactions with the corresponding begin record, and moving those with the commit record to s(REDO)
4. Undo phase: we go backward through the log again, undoing the transactions in s(Undo) until the begin record of the oldest transaction in the set of active transactions at the last checkpoint (note that we may even go before the most recent checkpoint record)
5. Redo phase: we go forward through the log again, redoing the transactions in s(Redo)

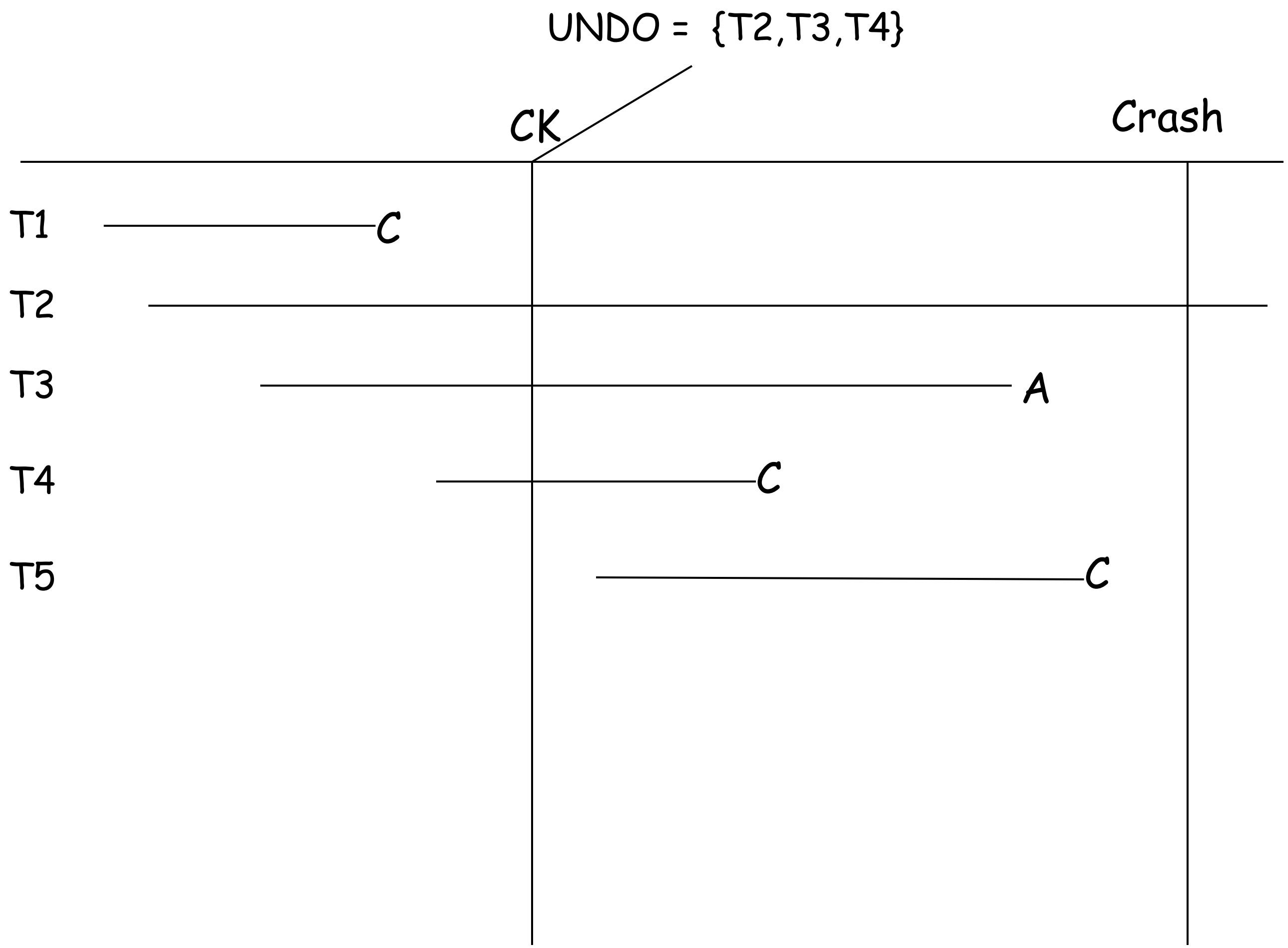
Warm restart: example

B(T1)
B(T2)
U(T2, O1, B1, A1)
I(T1, O2, A2)
B(T3)
C(T1)
B(T4)
U(T3,O2,B3,A3)
U(T4,O3,B4,A4)
CK(T2,T3,T4)
C(T4)
B(T5)
U(T3,O3,B5,A5)
U(T5,O4,B6,A6)
D(T3,O5,B7)
A(T3)
C(T5)
I(T2,O6,A8)



Example: the most recent checkpoint

B(T1)
B(T2)
U(T2, O1, B1, A1)
I(T1, O2, A2)
B(T3)
C(T1)
B(T4)
U(T3,O2,B3,A3)
U(T4,O3,B4,A4)
CK(T2,T3,T4)
C(T4)
B(T5)
U(T3,O3,B5,A5)
U(T5,O4,B6,A6)
D(T3,O5,B7)
A(T3)
C(T5)
I(T2,O6,A8)



Example: s(UNDO) and s(REDO)

B(T1)
B(T2)
8. U(T2, O1, B1, A1)
I(T1, O2, A2)
B(T3)
C(T1)
B(T4)
7. U(T3,O2,B3,A3)
9. U(T4,O3,B4,A4)

1. C(T4)
2. B(T5)
6. U(T3,O3,B5,A5)
10. U(T5,O4,B6,A6)
5. D(T3,O5,B7)
A(T3)
3. C(T5)
4. I(T2,O6,A8)

0. UNDO = {T2,T3,T4}. REDO = {}

---

1. C(T4) → UNDO = {T2, T3}. REDO = {T4}

2. B(T5) → UNDO = {T2,T3,T5}. REDO = {T4}

3. C(T5) → UNDO = {T2,T3}. REDO = {T4, T5}

Example: the UNDO phase

B(T1)
B(T2)
8. U(T2, O1, B1, A1)
I(T1, O2, A2)
B(T3)
C(T1)
B(T4)
7. U(T3,O2,B3,A3)
9. U(T4,O3,B4,A4)

1. C(T4)
2. B(T5)
6. U(T3,O3,B5,A5)
10. U(T5,O4,B6,A6)
5. D(T3,O5,B7)
A(T3)
3. C(T5)
4. I(T2,O6,A8)

0. UNDO = {T2,T3,T4}. REDO = {}

---

1. C(T4) → UNDO = {T2, T3}. REDO = {T4}

2. B(T5) → UNDO = {T2,T3,T5}. REDO = {T4}

3. C(T5) → UNDO = {T2,T3}. REDO = {T4, T5}

---

4. D(O6)

5. O5 =B7

6. O3 = B5

7. O2 =B3

8. O1=B1

Undo phase

Example: the REDO phase

B(T1)
B(T2)
8. U(T2, O1, B1, A1)
I(T1, O2, A2)
B(T3)
C(T1)
B(T4)
7. U(T3,O2,B3,A3)
9. U(T4,O3,B4,A4)

1. C(T4)
2. B(T5)
6. U(T3,O3,B5,A5)
10. U(T5,O4,B6,A6)
5. D(T3,O5,B7)
A(T3)
3. C(T5)
4. I(T2,O6,A8)

0. UNDO = {T2,T3,T4}. REDO = {}

---

1. C(T4) → UNDO = {T2, T3}. REDO = {T4}

2. B(T5) → UNDO = {T2,T3,T5}. REDO = {T4}

3. C(T5) → UNDO = {T2,T3}. REDO = {T4, T5}

---

4. D(O6)

5. O5 =B7

6. O3 = B5

7. O2 =B3

8. O1=B1

Undo phase

---

9. O3 = A4

10. O4 = A6

Redo phase

Cold restart

It is constituted by three phases:

1. Search for the most recent dump record in the log, and load the dump into the secondary storage (more precisely, we selectively copy the fragments of the DB that have been damaged by the disk failure)
2. Forward recovery of the dump state:
    1. We re-apply all actions in the log, in the order determined by the log
    2. At this point, we have the database state immediately before the crash
3. We execute the warm restart procedure

Exercise: cold restart

- Consider the following log: DUMP, B(T1), B(T2), B(T3), I(T1,O1,A1), D(T2,O2,B2), B(T4), U(T4,O3,B3,A3), U(T1,O4,B4,A4), C(T2), CK(T1,T3, T4), B(T5), B(T6), U(T5,O5,B5,A5), A(T3), CK(T1,T4,T5,T6), B(T7), A(T4), U(T7,O6,B6,A6), U(T6,O3,B7,A7), B(T8), C(T7)

- Suppose that a disk failure occurs. Assume the mixed strategy.

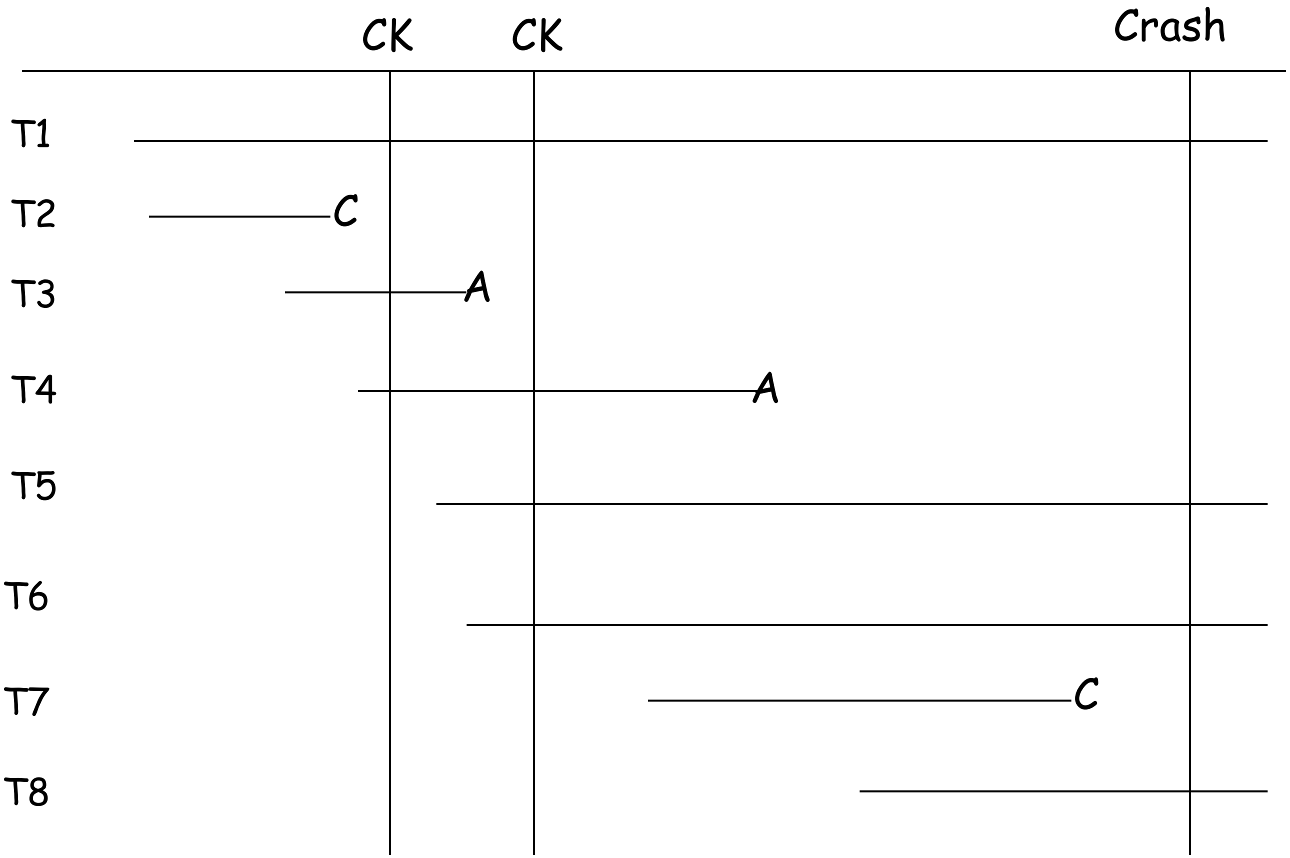
Solution: reconstruct the DB from DUMP

- DUMP, B(T1), B(T2), B(T3), I(T1,O1,A1), D(T2,O2,B2), B(T4), U(T4,O3,B3,A3), U(T1,O4,B4,A4), C(T2), CK(T1,T3, T4), B(T5), B(T6), U(T5,O5,B5,A5), A(T3), CK(T1,T4,T5,T6), B(T7), A(T4), U(T7,O6,B6,A6), U(T6,O3,B7,A7), B(T8), C(T7)

- We go to the most recent dump record in the log (the first record), and load the dump into the secondary storage
- We scan the log forward starting from B(T1), and we execute all actions in the log, until C(T7)
- We execute the warm restart procedure

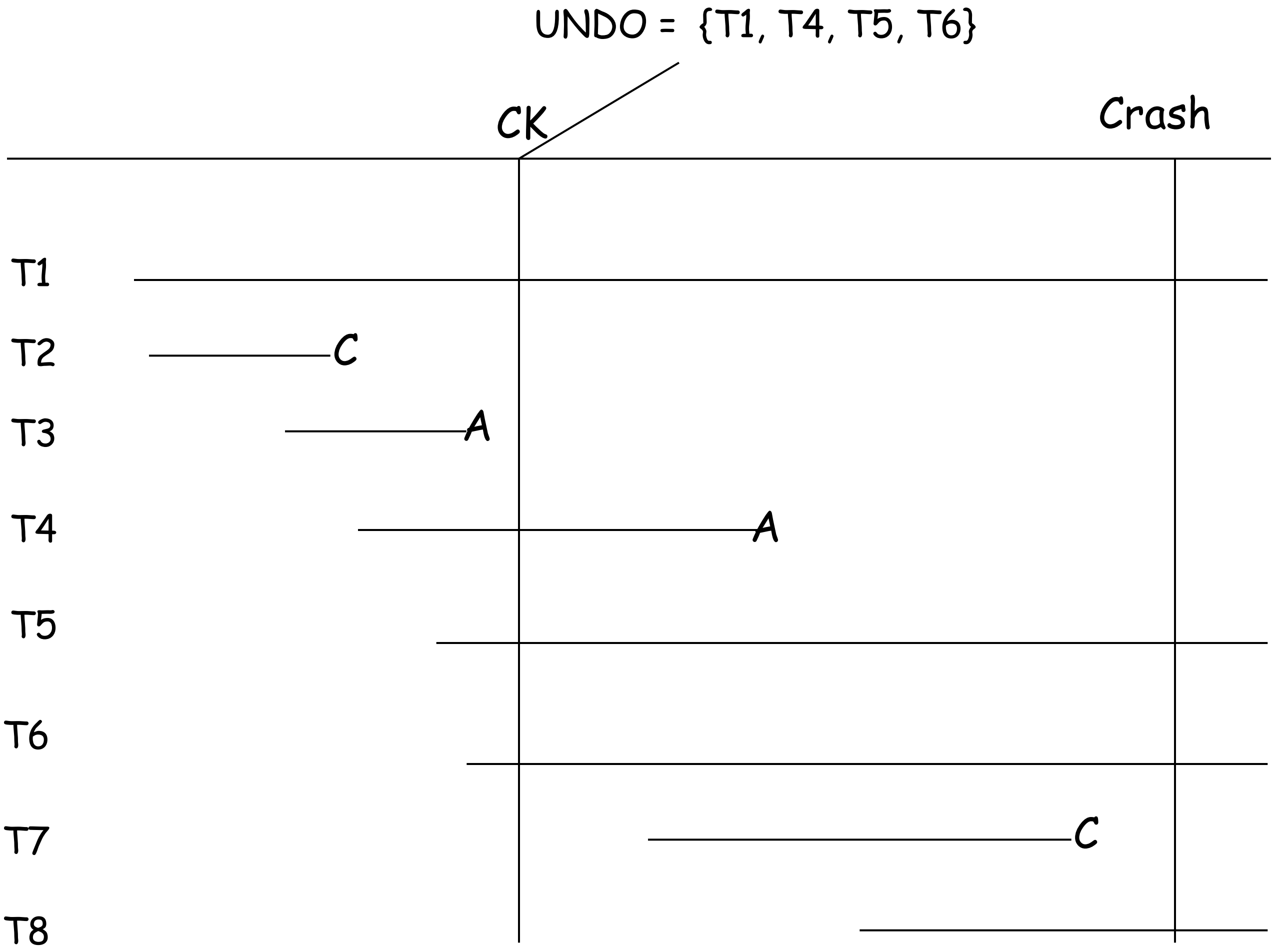
Solution: warm restart

B(T1),
B(T2),
B(T3),
I(T1,O1,A1),
D(T2,O2,B2),
B(T4),
U(T4,O3,B3,A3),
U(T1,O4,B4,A4),
C(T2),
CK(T1,T3, T4),
B(T5),
B(T6),
U(T5,O5,B5,A5),
A(T3),
CK(T1,T4,T5,T6),
B(T7),
A(T4),
U(T7,O6,B6,A6),
U(T6,O3,B7,A7),
B(T8),
C(T7)



Solution: most recent checkpoint

B(T1),
B(T2),
B(T3),
I(T1,O1,A1),
D(T2,O2,B2),
B(T4),
U(T4,O3,B3,A3),
U(T1,O4,B4,A4),
C(T2),
CK(T1,T3, T4),
B(T5),
B(T6),
U(T5,O5,B5,A5),
A(T3),
CK(T1,T4,T5,T6),
B(T7),
A(T4),
U(T7,O6,B6,A6),
U(T6,O3,B7,A7),
B(T8),
C(T7)



Solution: the UNDO and REDO sets

B(T1),
B(T2),
B(T3),
I(T1,O1,A1),
D(T2,O2,B2),
B(T4),
U(T4,O3,B3,A3),
U(T1,O4,B4,A4),
C(T2),
CK(T1,T3, T4),
B(T5),
B(T6),
U(T5,O5,B5,A5),
A(T3),
CK(T1,T4,T5,T6),
B(T7),
A(T4),
U(T7,O6,B6,A6),
U(T6,O3,B7,A7),
B(T8),
C(T7)

0. UNDO = {T1, T4, T5, T6}. REDO = {}

1. B(T7) → {T1, T4, T5, T6, T7}. REDO = {}

2. B(T8) → {T1, T4, T5, T6, T7, T8}. REDO = {}

3. C(T7) → {T1, T4, T5, T6, T8}. REDO = {T7}

Solution: the UNDO phase

B(T1),
B(T2),
B(T3),
I(T1,O1,A1),
D(T2,O2,B2),
B(T4),
U(T4,O3,B3,A3),
U(T1,O4,B4,A4),
C(T2),
CK(T1,T3, T4),
B(T5),
B(T6),
U(T5,O5,B5,A5),
A(T3),
CK(T1,T4,T5,T6),
B(T7),
A(T4),
U(T7,O6,B6,A6),
U(T6,O3,B7,A7),
B(T8),
C(T7)

0. UNDO = {T1, T4, T5, T6}. REDO = {}

---

1. B(T7) → {T1, T4, T5, T6, T7}. REDO = {}
2. B(T8) → {T1, T4, T5, T6, T7, T8}. REDO = {}
3. C(T7) → {T1, T4, T5, T6, T8}. REDO = {T7}

---

4. O3 = B7
5. O5 = B5
6. O4 = B4
7. O3 = B3
8. D(O1)

Undo phase

Solution: the REDO phase

B(T1),
B(T2),
B(T3),
I(T1,O1,A1),
D(T2,O2,B2),
B(T4),
U(T4,O3,B3,A3),
U(T1,O4,B4,A4),
C(T2),
CK(T1,T3, T4),
B(T5),
B(T6),
U(T5,O5,B5,A5),
A(T3),
CK(T1,T4,T5,T6),
B(T7),
A(T4),
U(T7,O6,B6,A6),
U(T6,O3,B7,A7),
B(T8),
C(T7)

0. UNDO = {T1, T4, T5, T6}. REDO = {}

---

1. B(T7) → {T1, T4, T5, T6, T7}. REDO = {}

2. B(T8) → {T1, T4, T5, T6, T7, T8}. REDO = {}

3. C(T5) → {T1, T4, T5, T6, T8}. REDO = {T7}

---

4. O3 = B7

5. O5 = B5

6. O4 = B4

7. O3 = B3

8. D(O1)

Undo phase

---

9. O6 = A6

Redo phase

Exercise

Suppose that at time T the last checkpoint record in the log is C1, the checkpoint record preceeding C1 in the log is C2, and the last dump record in the log appears between C2 and C1. Prove or disprove the following statements:

1. “At time T, we can delete all records in the log before C1, without affecting the effectiveness of the recovery procedure for a system failure.”

2. “At time T, we can delete all records in the log before C2, without affecting the effectiveness of the recovery procedure for a media failure.”

Solution

To refute claim (1), it is sufficient to consider the following configuration of the log:

B(T)   I(T,O,AS)   C2   DUMP   C1

Suppose that, with this configuration of the log, at time T we delete all records in the log before C2, i.e., B(T) and I(T,O,AS). Note that T is in the list of the active transactions corresponding to C1. Therefore, deleting the action I(T,O,AS) is obviously wrong. Indeed, should the warm restart procedure need to be executed, it would need to undo such an operation. However, this would be impossible, because the corresponding record would not appear in the log anymore. Obviously, the cold restart procedure would have the same problem, because it uses the warm restart procedure. Therefore, claim (2) is also refuted.