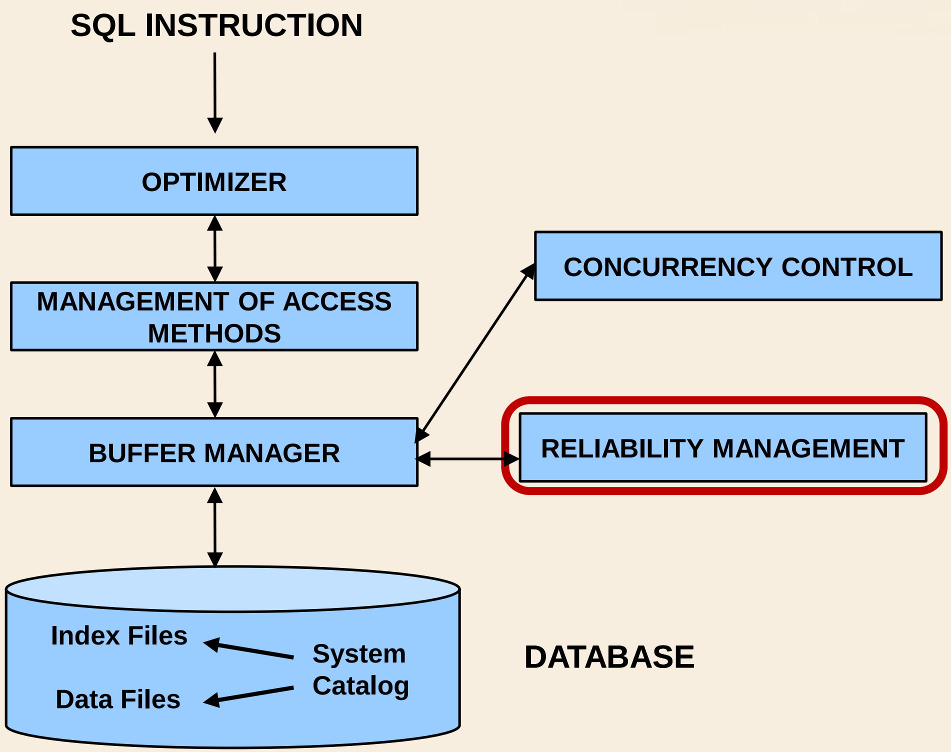
Database Management Systems

Reliability Management

DBMS Architecture



Reliability Manager

- It is responsible of the *atomicity* and *durability* ACID properties
- It implements the following transactional commands
  - begin transaction (B, usually implicit)
  - commit work (C)
  - rollback work (A, for abort)
- It provides the recovery primitives
  - warm restart
    - for main memory failures
  - cold restart

Reliability Manager

- It manages the reliability of read/write requests by interacting with the buffer manager
  - It may generate new read/write requests for reliability purposes
- It exploits the *log file*
  - a persistent archive recording DBMS activity
  - stored on *stable memory*
- It prepares data for performing recovery by means of the operations
  - checkpoint
  - dump

Stable memory

- Memory that is resistant to failure
  - it is an abstraction
  - it is approximated by means of
    - redundancy
    - robust write protocols
- Failures in stable memory are considered catastrophic

Log file

- Sequential file written in stable memory
  - It records transaction activities in chronological order
- Log record types
  - Transaction records
  - System records
- Writing the log
  - Records are written in the current block in sequential order
  - Records belonging to different transactions are interleaved

Transaction records

- Describe the activities performed by each transaction in execution order
  - Transaction delimiters
    - Begin B(T)
    - Commit C(T)
    - Abort/Rollback A(T)

  where T is the Transaction Identifier

DBMG

Transaction records

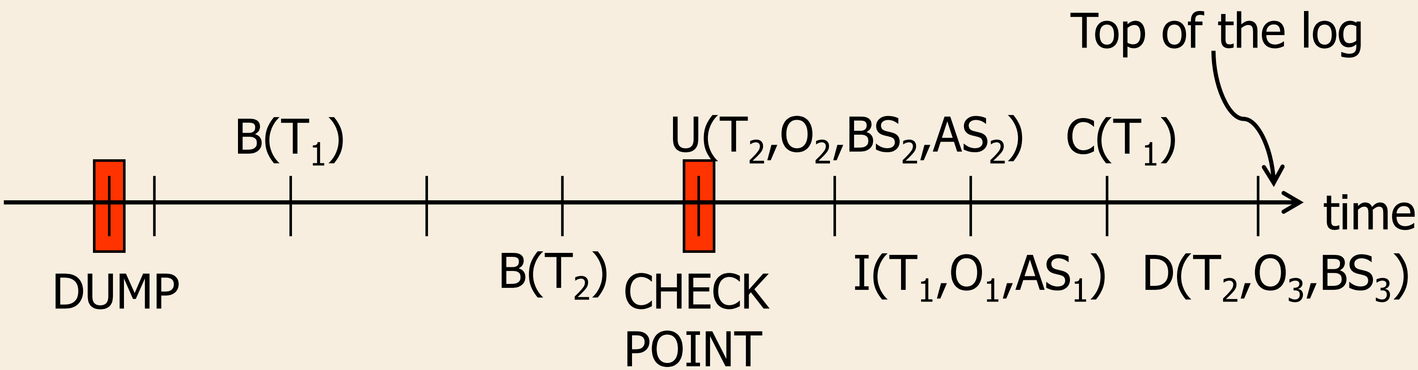
- Data modifications
  - Insert I(T, O, AS)
  - Delete D(T, O, BS)
  - Update U(T, O, BS, AS)

  where
  - O is the written object (RID)
  - AS is the After State (state of object O after the modification)
  - BS is the Before State (state of object O before the modification)

System records

- Record system operations saving data on disk or other tertiary (or off-line) storage
  - Dump
  - Checkpoint CK(L)
    where $L = T_1, T_2, \ldots, T_n$
    is the set of TIDs of active transactions

Log example


Top of the log
B(T_1)
U(T_2,O_2,BS_2,AS_2)
C(T_1)
time
DUMP
B(T_2)
CHECK POINT
I(T_1,O_1,AS_1)
D(T_2,O_3,BS_3)

Undoing and redoing actions

- *Undo* of an action on an object O

Action	Undo action
insert O	delete O
update O	write the before state (BS) of O
delete O	write the before state (BS) of O

- *Redo* of an action on an object O

Action	Redo action
insert O	write the after state (AS) of O
update O	write the after state (AS) of O
delete O	delete O

Undoing and redoing actions

- Idempotency property

  Undo or Redo can be repeated an arbitary number of times without changing the final outcome

  UNDO (UNDO(action)) = UNDO(action)

- Useful for managing crashes during the recovery process

Checkpoint

- Operation periodically requested by the Reliability Manager to the Buffer Manager
  - It allows a faster recovery process
- During the checkpoint, the DBMS
  - writes data on disk for all completed transactions
    - *synchronous* write
  - records the active transactions

Execution of a checkpoint

1. The TIDs of all active transactions are recorded
   - after the checkpoint is started, no transaction can commit until the checkpoint ends
2. The pages of concluded transactions (committed or aborted) are *synchronously* written on disk
   - by means of the force primitive
3. At the end of step 2, a checkpoint record is synchronously written on the log
   - it contains the set of active transactions
   - it is written by means of the force primitive

Checkpoint

- After a checkpoint
  - The effect of all committed transactions is *permanently* stored on disk
  - The state of data pages written by active transactions is unknown

Dump

- It creates a complete copy of the database
  - typically performed when the system is offline
  - the database copy is stored in stable memory
    - tertiary storage or off-line storage
  - the copy may be incremental
- At the end, a dump record is written in the log file
  - Date and time of the dump
  - Dump device used

Rules for writing the log

- Designed to allow recovery in presence of failure
  - WAL
  - Commit precedence

Write Ahead Log

- The before state (BS) of data in a log record is written in stable memory before database data is written on disk
  - During recovery, it allows the execution of undo operations on data already written on disk

Commit precedence

- The after state (AS) of data in a log record is written in stable memory before commit
  - During recovery, it allows the execution of redo operations for transactions that already committed, but were not written on disk

DBMG

Practical rules for writing the log

- BS and AS are written together
  - WAL

    The log must be written before the record in the database
  - Commit precedence

    The log must be written before commit

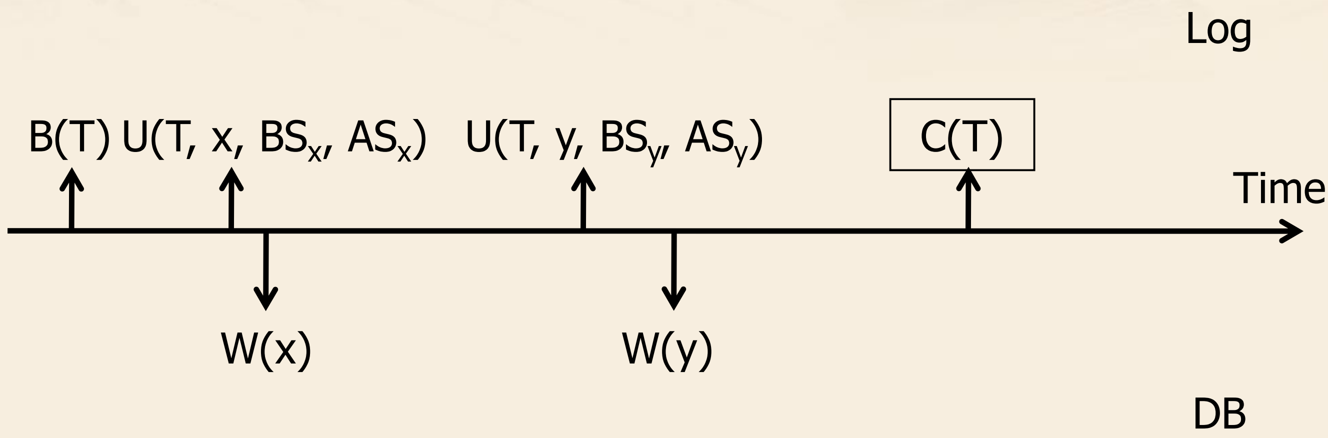
Practical rules for writing the log

- The log is written *synchronously* (force)
  - for data modifications written on disk
  - on commit
- The log is written *asynchronously*
  - for abort/rollback

Commit record

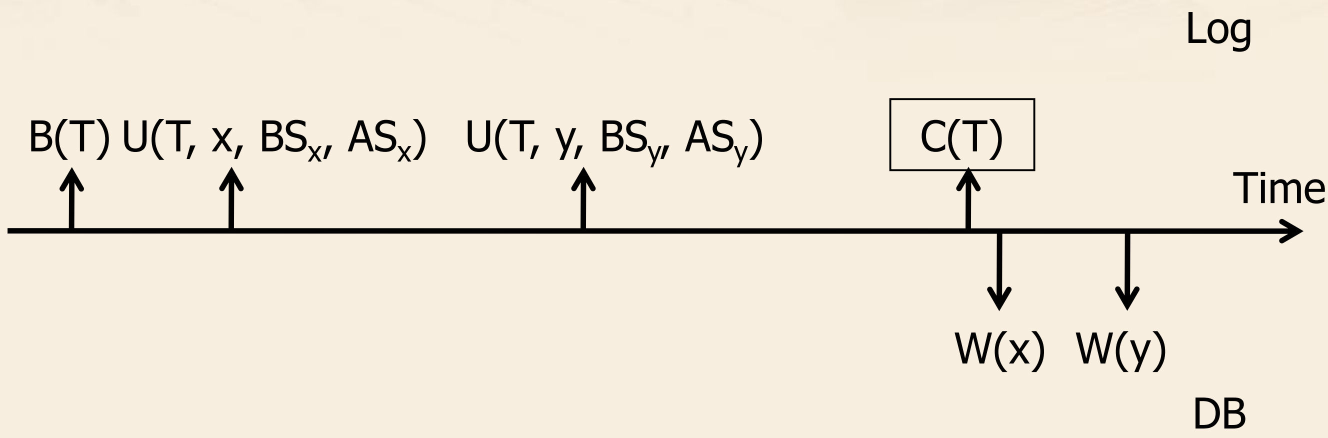
- The commit record on the log is a border line
  - If it is not written in the log, the transaction should be *undone* upon failure
  - If it is written, the transaction should be *redone* upon failure

Protocols for writing the log and the database



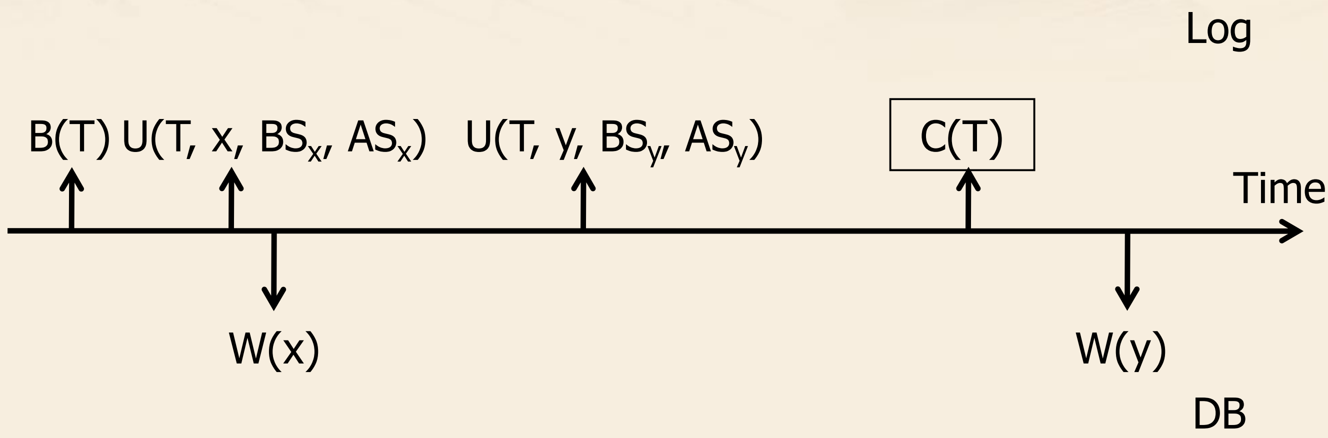
- All database disk writes are performed *before* commit
  - It does not require redo of committed transactions

Protocols for writing the log and the database



- All database disk writes are performed *after* commit
  - It does not require undo of uncommitted transactions

Protocols for writing the log and the database



- Disk writes for the database take place both *before* and *after* commit
  - It requires both the undo and redo operations
- Mixed approach adopted in real systems

Writing the log

- The usage of robust protocols to guarantee reliability is costly
  - Comparable with database update cost
- It is required to guarantee the ACID properties
  - Log writing is optimized
    - Compact format
    - Parallelism
    - Commit of groups of transactions

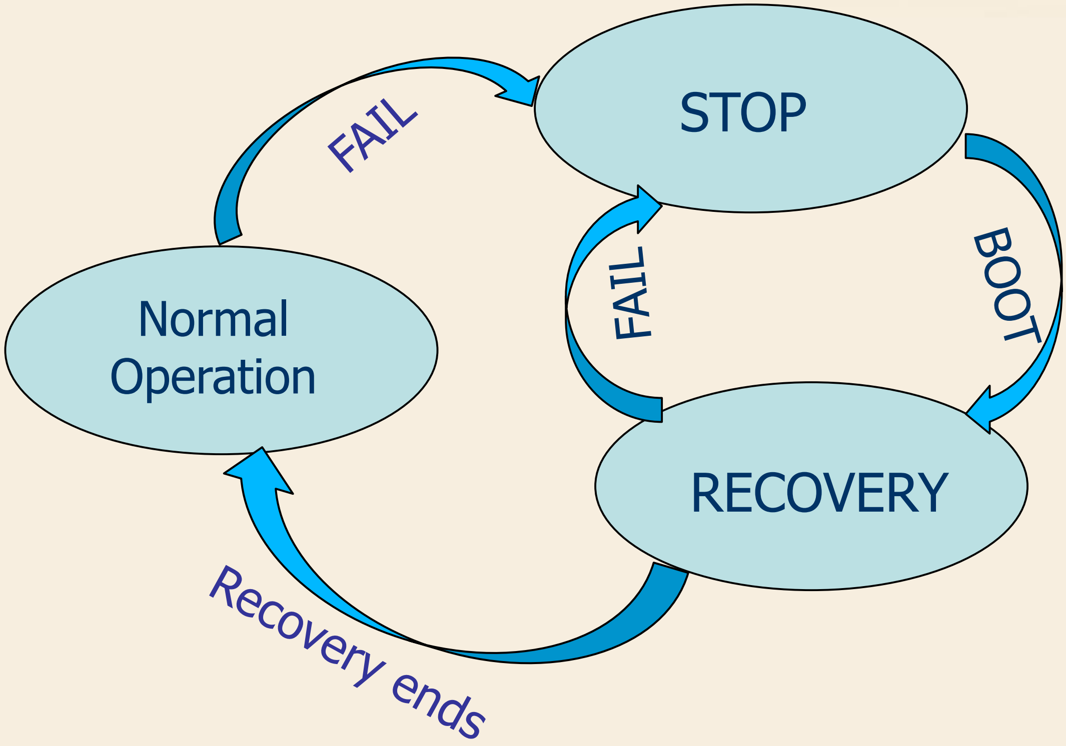
Database Management Systems

Recovery Management

Types of failures

- System failure
  - Caused by software problems or power supply interruptions
  - It causes losing the *main memory content* (DBMS buffer) but *not the disk* (both database and log)
- Media failure
  - Caused by failure of devices managing secondary memory
  - It causes losing the *database content on disk*, but *not the log* content (stored in stable storage)

Fail-stop model

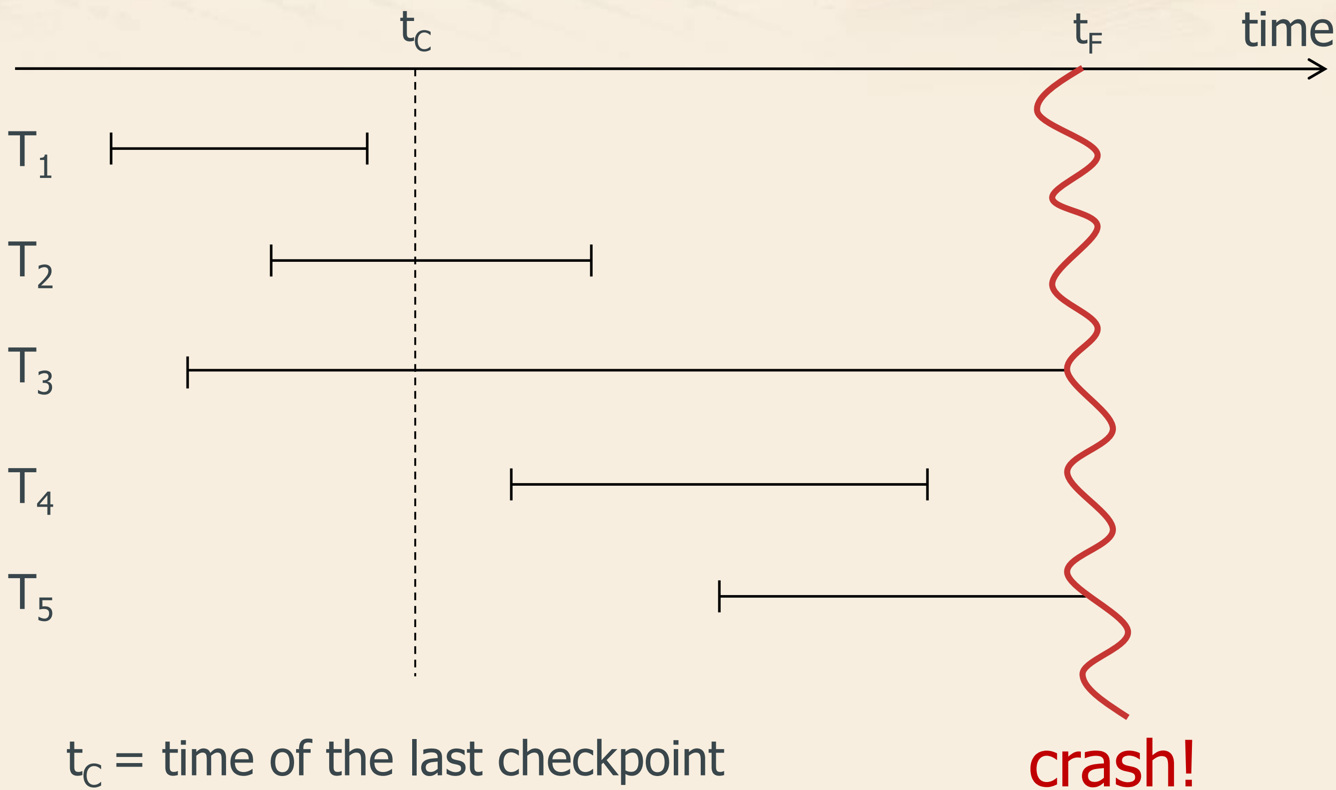


Recovery

- When a failure occurs
  - The system is stopped
- Recovery depends on the failure type
  - Warm restart
    - performed for system failures
  - Cold restart
    - performed for media failures
- When recovery ends
  - the system becomes again available to transactions

Warm Restart

Transaction categories



t_C = time of the last checkpoint

t_F = time of failure

Transaction categories

- Transactions *completed* before the checkpoint (T1)
  - No recovery action is needed
- Transactions which *committed*, but for which some writes on disk may not have been done yet (T2 and T4)
  - redo is needed
- *Active* transactions at the time of failure (T3 and T5)
  - they did not commit
  - undo is needed

Checkpoint record

- The checkpoint record is not needed to enable recovery
  - It provides a faster warm restart
- Without checkpoint record
  - The entire log needs to be read until the last dump

Warm restart algorithm

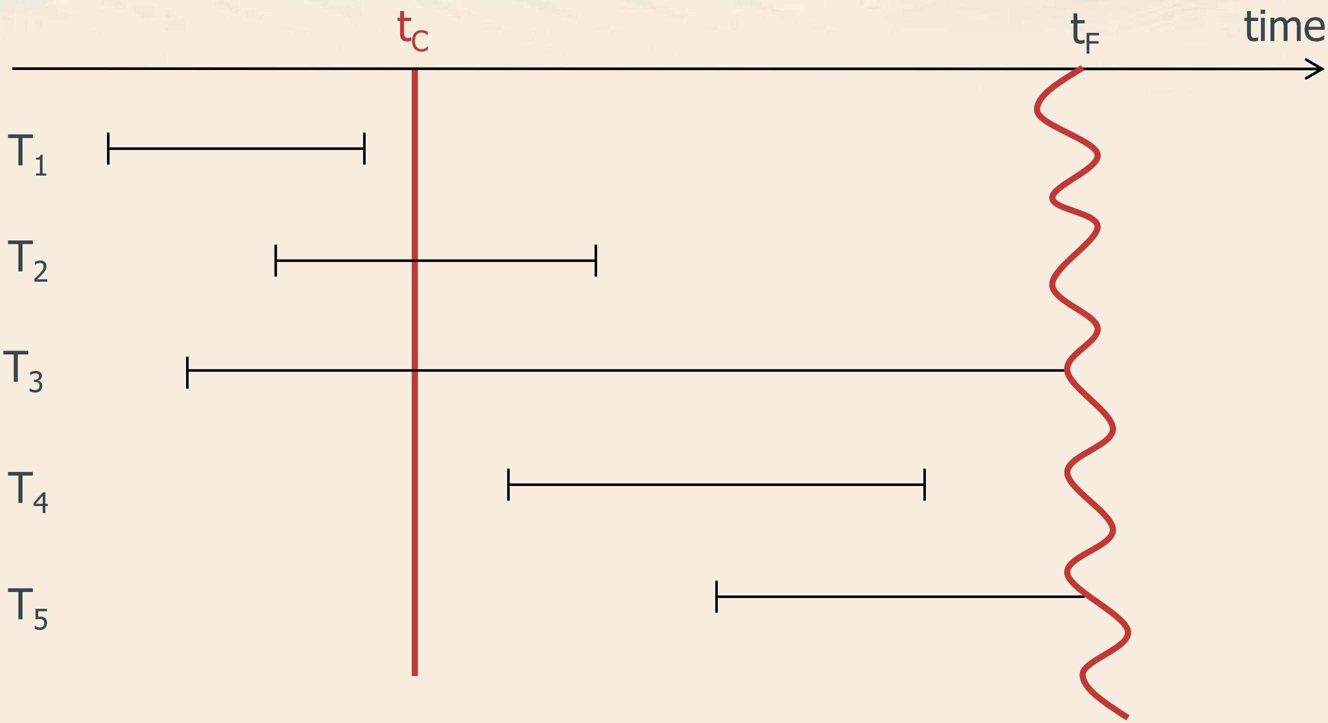
1. Read backwards the log until the last checkpoint record
2. Detect transactions which should be undone/redone
   a) At the last checkpoint
      - UNDO = { Active transactions at checkpoint }
      - REDO = { } (empty)

Warm restart algorithm

b) Read forward the log
  - UNDO = Add all transactions for which the begin record is found
  - REDO = Move transactions from UNDO to REDO list when the commit record is found
    - Transactions ending with rollback remain in the UNDO list

- At the end of step 2
  - UNDO = list of transactions to be undone
  - REDO = list of transactions to be redone

Warm restart algorithm

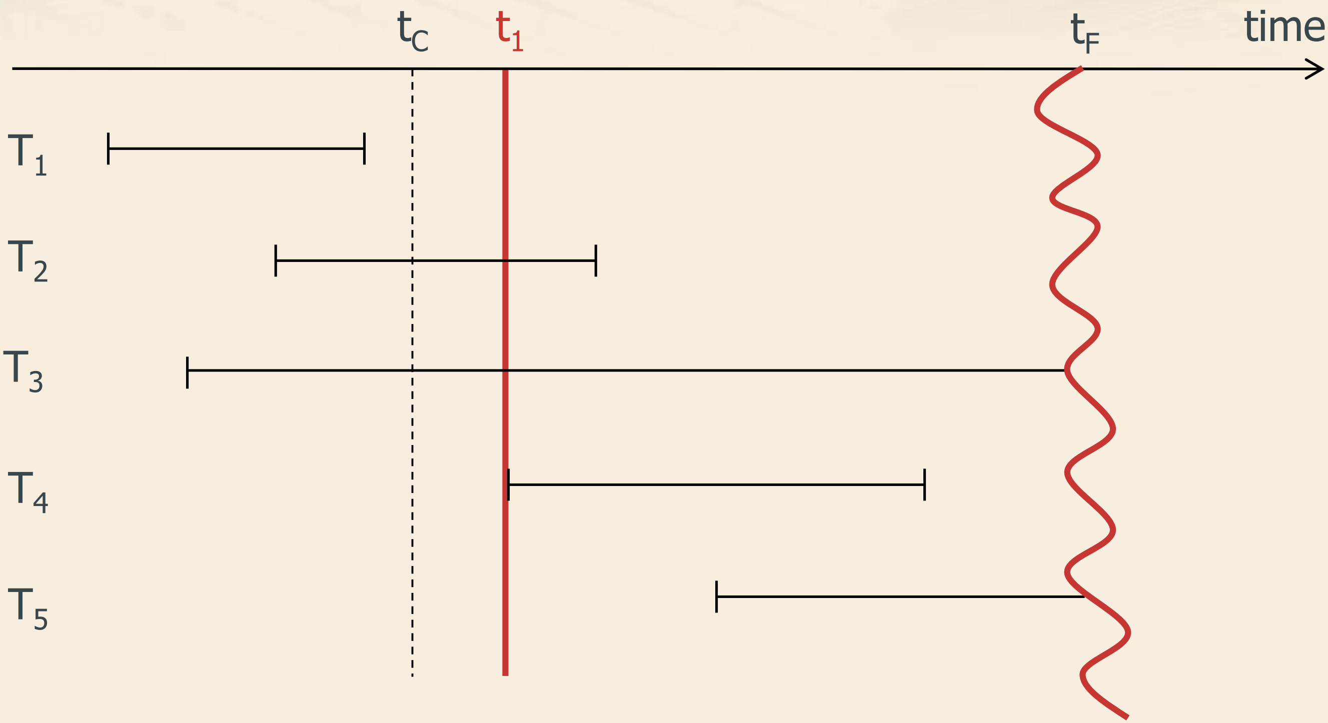


UNDO = $\{T_2, T_3\}$
REDO = $\{ \}$

t_C = time of last checkpoint

t_F = time of failure

Warm restart algorithm

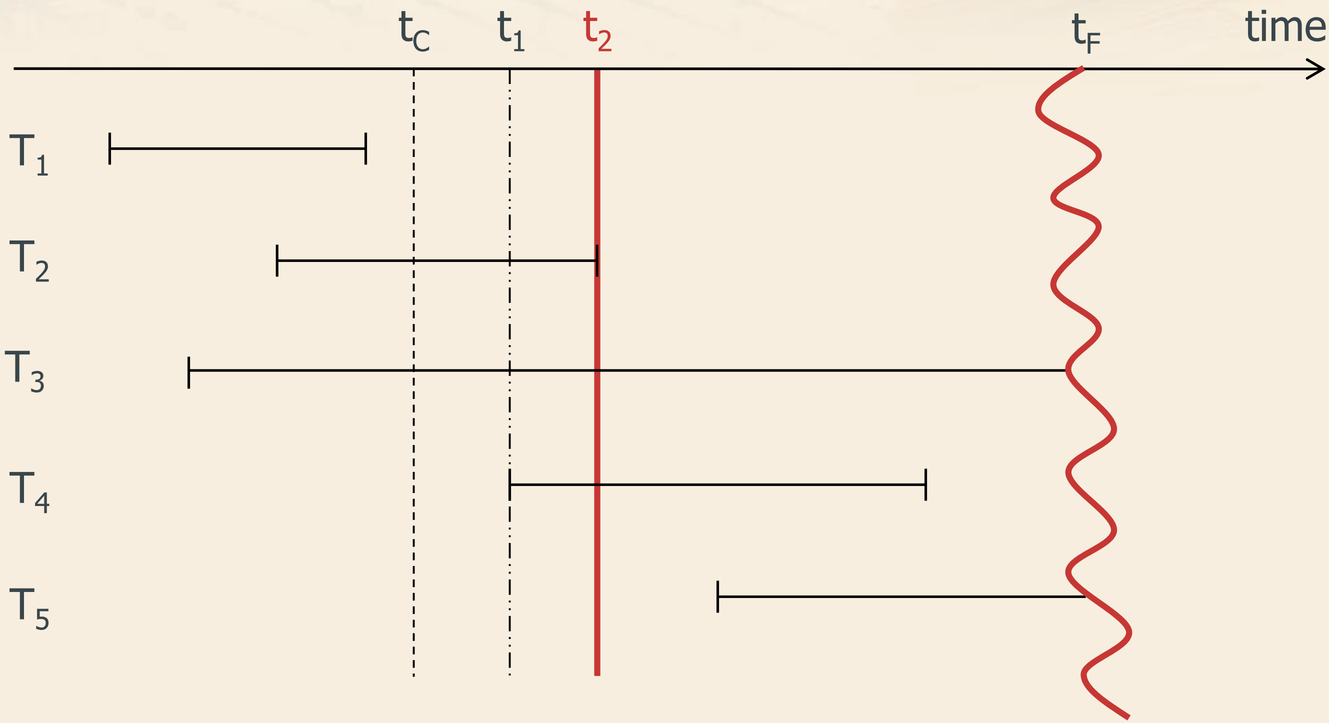


UNDO = $\{T_2, T_3, T_4\}$
REDO = $\{ \}$

t_C = time of last checkpoint

t_F = time of failure

Warm restart algorithm

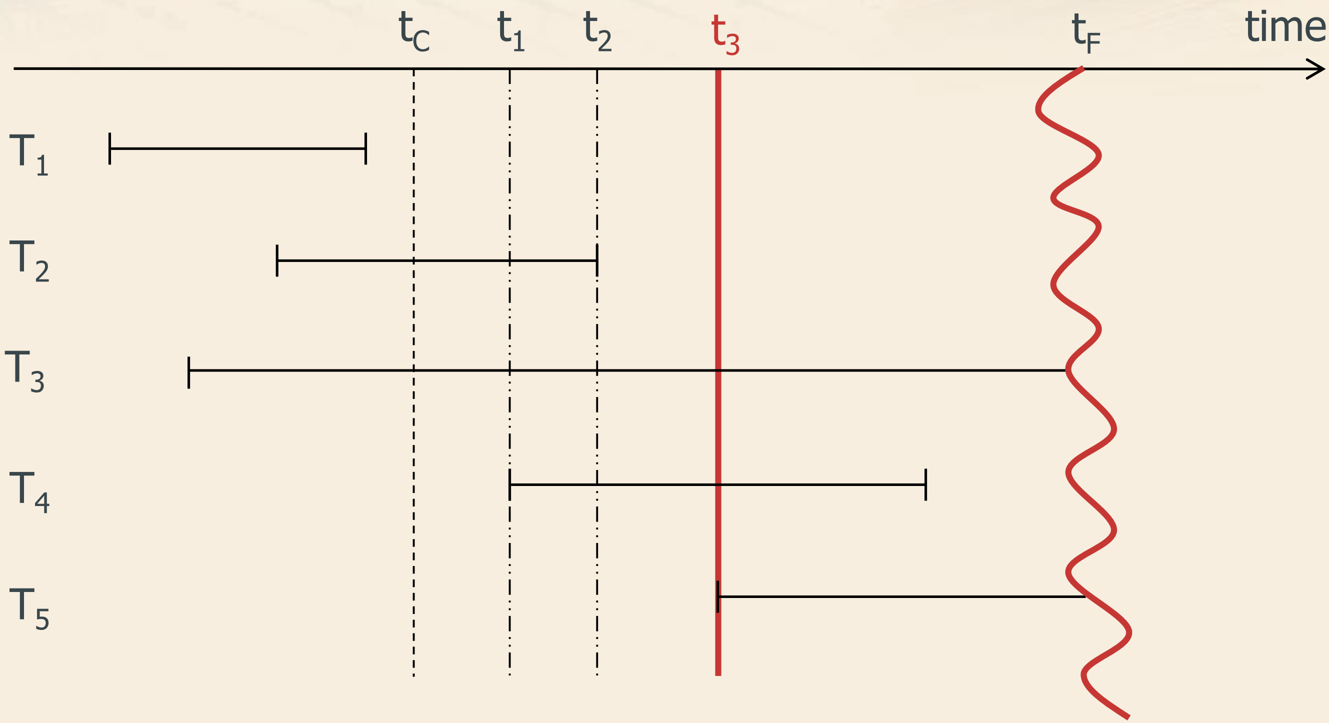


UNDO = $\{T_3, T_4\}$
REDO = $\{T_2\}$

t_C = time of last checkpoint

t_F = time of failure

Warm restart algorithm

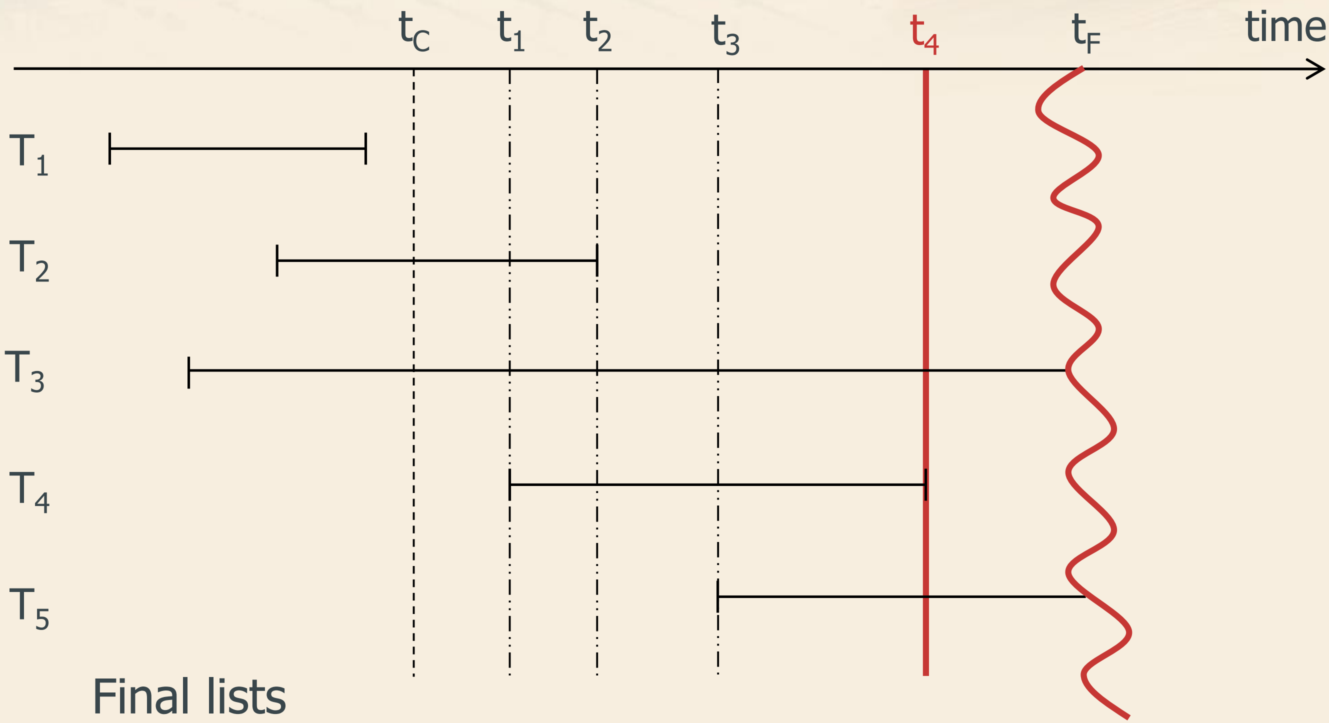


UNDO = $\{T_3, T_4, T_5\}$
REDO = $\{T_2\}$

t_C = time of last checkpoint

t_F = time of failure

Warm restart algorithm



Final lists
UNDO = $\{T_3, T_5\}$
REDO = $\{T_2, T_4\}$

t_C = time of last checkpoint

t_F = time of failure

Warm restart algorithm

3. Data Recovery
   a) The log is read *backwards* from the time of failure until the beginning of the oldest transaction in the UNDO list
      - Actions performed by all transactions in the UNDO list are undone
      - For each transaction the begin record should be reached
        - even if it is earlier than the last checkpoint

Warm restart algorithm

b) The log is read *forward* from the beginning of the oldest transaction in the REDO list

- Actions of transactions in the REDO list are applied to the database
- For each transaction, the starting point is its begin record

Warm restart example

- Log snippet

  $B(T_1)$ $B(T_2)$ $U(T_2, O_1, B_1, A_1)$ $I(T_1, O_2, A_2)$ $B(T_3)$ $C(T_1)$ $B(T_4)$ $U(T_3, O_2, B_3, A_3)$ $U(T_4, O_3, B_4, A_4)$ $CK(T_2, T_3, T_4)$ $C(T_4)$ $B(T_5)$ $U(T_3, O_3, B_5, A_5)$ $U(T_5, O_4, B_6, A_6)$ $D(T_3, O_3, B_7)$ $A(T_3)$ $C(T_5)$ $I(T_2, O_6, A_8)$ *failure*

Warm restart example

- Log snippet

  $B(T_1)$ $B(T_2)$ $U(T_2, O_1, B_1, A_1)$ $I(T_1, O_2, A_2)$ $B(T_3)$ $C(T_1)$ $B(T_4)$ $U(T_3, O_2, B_3, A_3)$ $U(T_4, O_3, B_4, A_4)$ $CK(T_2, T_3, T_4)$ $C(T_4)$ $B(T_5)$ $U(T_3, O_3, B_5, A_5)$ $U(T_5, O_4, B_6, A_6)$ $D(T_3, O_3, B_7)$ $A(T_3)$ $C(T_5)$ $I(T_2, O_6, A_8)$

1. At the checkpoint
   - UNDO = $\{T_2, T_3, T_4\}$
   - REDO = { }

Warm restart example

$B(T_1)$ $B(T_2)$ $U(T_2, O_1, B_1, A_1)$ $I(T_1, O_2, A_2)$ $B(T_3)$ $C(T_1)$ $B(T_4)$ $U(T_3, O_2, B_3, A_3)$ $U(T_4, O_3, B_4, A_4)$ $CK(T_2, T_3, T_4)$ $C(T_4)$ $B(T_5)$ $U(T_3, O_3, B_5, A_5)$ $U(T_5, O_4, B_6, A_6)$ $D(T_3, O_3, B_7)$ $A(T_3)$ $C(T_5)$ $I(T_2, O_6, A_8)$

2. Read the log forward

Operation	UNDO	REDO
CK	$\{T_2,T_3,T_4\}$	{ }

Warm restart example

$B(T_1)$ $B(T_2)$ $U(T_2, O_1, B_1, A_1)$ $I(T_1, O_2, A_2)$ $B(T_3)$ $C(T_1)$ $B(T_4)$ $U(T_3, O_2, B_3, A_3)$ $U(T_4, O_3, B_4, A_4)$ $CK(T_2, T_3, T_4)$ $C(T_4)$ $B(T_5)$ $U(T_3, O_3, B_5, A_5)$ $U(T_5, O_4, B_6, A_6)$ $D(T_3, O_3, B_7)$ $A(T_3)$ $C(T_5)$ $I(T_2, O_6, A_8)$

2. Read the log forward

Operation	UNDO	REDO
CK	$\{T_2,T_3,T_4\}$	$\{\ \}$
$C(T_4)$	$\{T_2,T_3\}$	$\{T_4\}$
$B(T_5)$	$\{T_2,T_3,T_5\}$	$\{T_4\}$
$A(T_3)$	$\{T_2,T_3,T_5\}$	$\{T_4\}$
$C(T_5)$	$\{T_2,T_3\}$	$\{T_4,T_5\}$

Final lists

DBMG

Warm restart example

$B(T_1)$ $B(T_2)$ $U(T_2, O_1, B_1, A_1)$ $I(T_1, O_2, A_2)$ $B(T_3)$ $C(T_1)$ $B(T_4)$ $U(T_3, O_2, B_3, A_3)$ $U(T_4, O_3, B_4, A_4)$ $CK(T_2, T_3, T_4)$ $C(T_4)$ $B(T_5)$ $U(T_3, O_3, B_5, A_5)$ $U(T_5, O_4, B_6, A_6)$ $D(T_3, O_3, B_7)$ $A(T_3)$ $C(T_5)$ $I(T_2, O_6, A_8)$

3. Undo transactions in UNDO = $\{T_2, T_3\}$
   a) DELETE O_6
   b) INSERT $O_3 = B_7$
   c) $O_3 = B_5$
   d) $O_2 = B_3$
   e) $O_1 = B_1$

DBMG

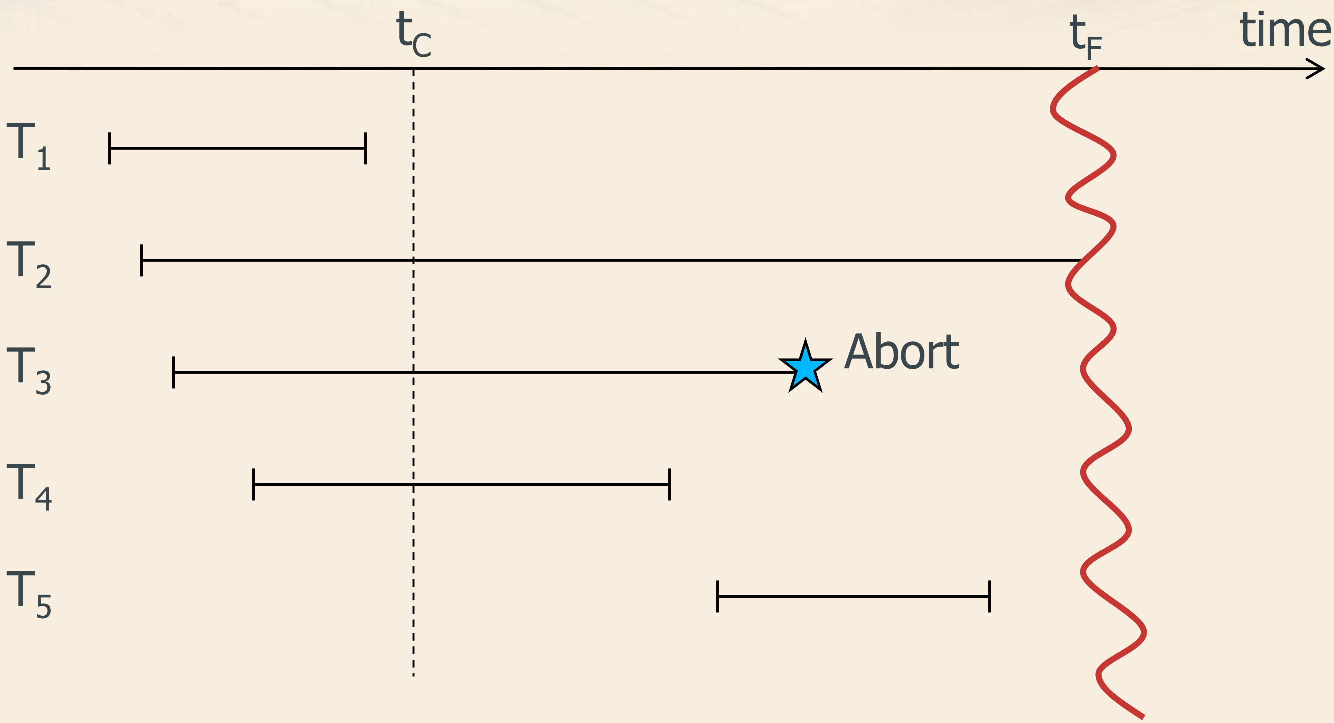
Warm restart example

$B(T_1)$ $B(T_2)$ $U(T_2, O_1, B_1, A_1)$ $I(T_1, O_2, A_2)$ $B(T_3)$ $C(T_1)$ $B(T_4)$ $U(T_3, O_2, B_3, A_3)$ $U(T_4, O_3, B_4, A_4)$ $CK(T_2, T_3, T_4)$ $C(T_4)$ $B(T_5)$ $U(T_3, O_3, B_5, A_5)$ $U(T_5, O_4, B_6, A_6)$ $D(T_3, O_3, B_7)$ $A(T_3)$ $C(T_5)$ $I(T_2, O_6, A_8)$

4. Redo transactions in REDO = $\{T_4, T_5\}$
   a) $O_3 = A_4$
   b) $O_4 = A_6$

DBMG

Warm restart example



REDO = $\{T_4, T_5\}$
UNDO = $\{T_2, T_3\}$

t_F = time of failure

Cold Restart

Cold restart

- It manages failures damaging (a portion of) the database on disk
- Main steps
  1. Access the last dump to restore the damaged portion of the database on disk
  2. Starting from the last dump record, read the log forward and redo all actions on the database and transaction commit/rollback
  3. Perform a warm restart

Cold restart

- Alternative to steps 2 and 3
  - Perform only actions of committed transactions
  - It requires two log reads
    - Detect committed transactions
      - build a REDO list
    - Redo actions of transactions in REDO list