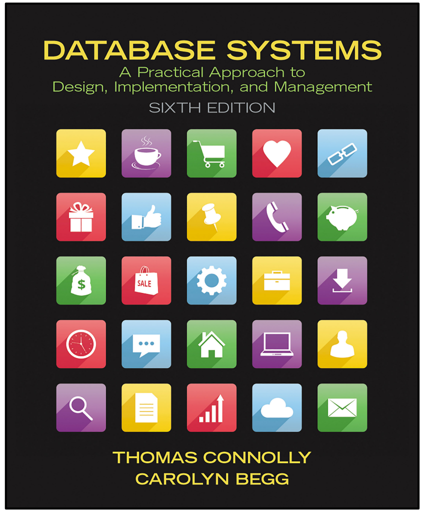
Recoverability

Topic 6, Lesson 8
Algorithms supporting the Durability property

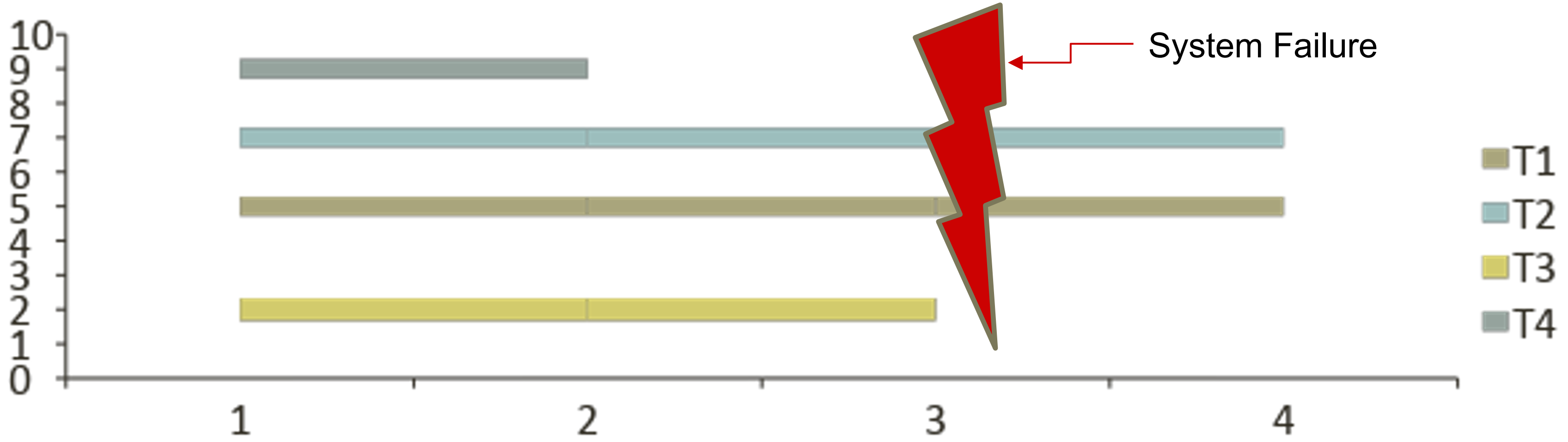
Adapted from Chapter 22



Transaction

- Recovery manager ensures the atomicity and durability properties of ACID
  - Atomicity: either all actions in a transaction are done or none are done
  - Durability: if a transaction is committed, changes persist within the database
- Desired behavior
  - Keep actions of committed transactions
  - Discard actions of uncommitted transaction

Keep the committed transactions



Throw away the active transactions work

- T3 and T4 actions should appear in the database
- T1 and T2 actions should not appear in the database

Database recovery

Process of restoring database to a correct state in the event of a failure.

- Need for Recovery Control
  - Two types of storage: volatile (main memory) and nonvolatile (stable storage).
  - Volatile storage does not survive system crashes.
  - Stable storage represents information that has been replicated in several nonvolatile storage media with independent failure modes.

Types of database failures

- System crashes, resulting in loss of data in the buffer pool of the DBMS.
- Media failures, resulting in loss of parts of secondary storage.
- Application software errors, transaction started but never completed.
- Natural physical disasters resulting in loss of servers.
- Carelessness or unintentional destruction of data.
- Sabotage resulting in loss of data.

Transaction commit

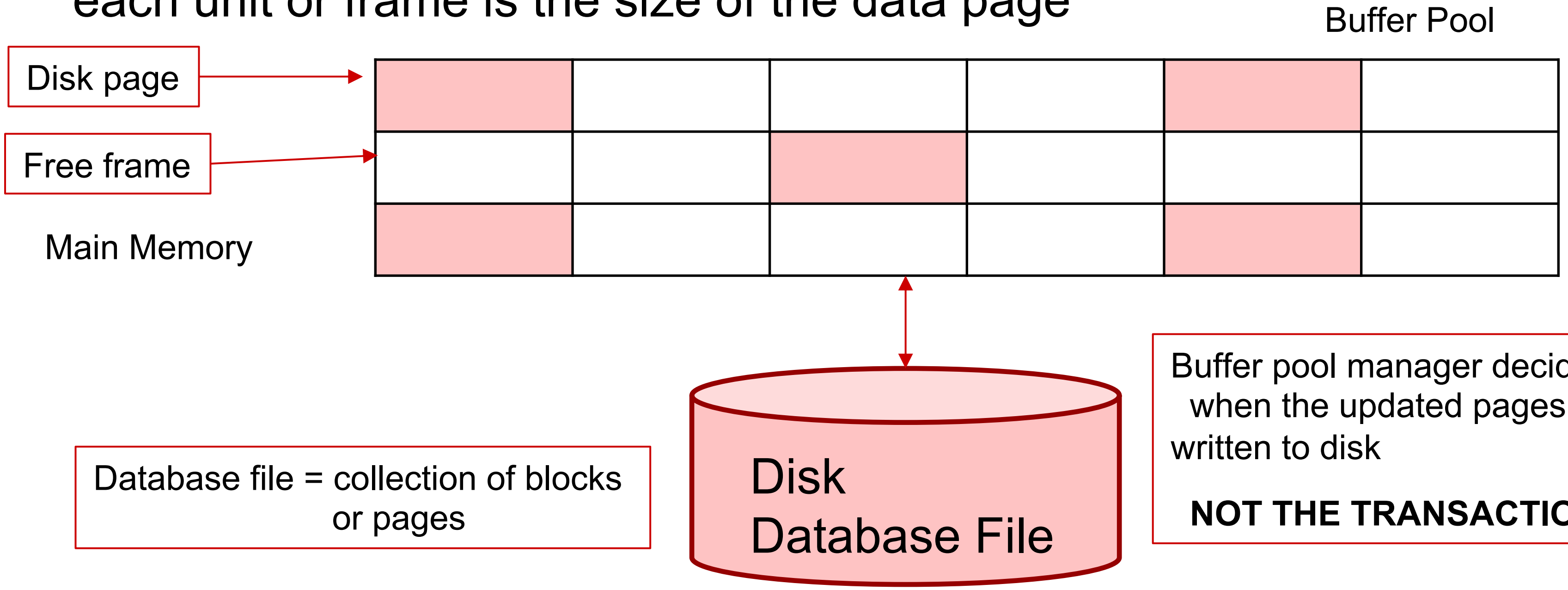
- Transactions represent basic unit of recovery.
- Recovery manager responsible for atomicity and durability.
- If a failure occurs between the time when a transaction issues a commit and the database buffers being flushed to secondary storage then, to to ensure durability, the recovery manager must *redo* (*rollforward*) the transaction's updates.
  - All updates for the committed transaction must be safely stored in the database to ensure the atomicity and durability properties

Transaction: no commit

- If a transaction had not committed at failure time, the recovery manager must *undo* (*rollback*) any effects of that transaction to ensure the atomicity property.
- When a transaction issues a rollback, this scenario is a **partial undo** - only one transaction must be undone in the database files.
- **Global undo** - all uncommitted transactions at failure times needs to be undone.

Another complication: buffer pool policies

For the DBMS to access data, the data must be read from stable storage into virtual memory. The data or pages are stored in a buffer, where each unit or frame is the size of the data page



Database file = collection of blocks or pages

Buffer pool manager decides when the updated pages are written to disk

NOT THE TRANSACTION!!

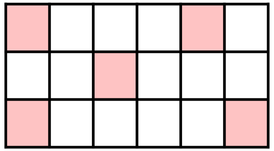
Buffer management

- Database buffer is a limited resource and must be optimally managed
  - Buffer is segmented, each segment can accommodate a page from the database
  - The buffer manager tracks the number of transactions using the page (pin count) as well as if any transaction made a change to the page (dirty bit)
  - Completely fill the buffer with pages from disk
  - When a transaction requests a page - look for the page in the buffer
    - Update the pin count

Buffer management

- When the buffer is full, and a new page is requested
  - Must remove a page to accommodate the new requested page
  - If a page is unpinned and not dirty remove this page and read in the new page from disk
  - If a page in unpinned and dirty , write the change to permanent storage and read in the new page from disk

Buffer manager policies



- **FORCE** – every write to disk at commit?
  - Force: Poor DB performance (many writes clustered on same page) but does guarantee the persistence of the data
  - No Force: buffer manager may delay update of the committed updates
- **STEAL** – allow dirty pages to be written to disk?
  - STEAL: If so, reading data from uncommitted transactions violates atomicity
  - No Steal: If not, poor performance cannot handle the required load

	Force: every committed write to disk	No Force: write when optimal
Steal: use internal DB buffer for read		Desired but complicated
No Steal: always read only committed data	Easy but slow	

Steal & No force policy best combination

Steal and No-force is most complicated but allows for highest database performance.

NO FORCE (complicates enforcing Durability) – What if system crashes before a modified page written by a committed transaction makes it to disk? – Write as little as possible, in a convenient place, at commit time, to support REDOing the modifications.

STEAL (complicates enforcing Atomicity) – What if the transaction that performed the update aborts? – What if system crashes before transaction is finished? – Must remember the old value of the updated data (to support UNDOing the update to the new value).

Recovery facilities

DBMS provides the following facilities to support DB recovery:

1. **Backup mechanism**
   - Makes periodic backup copies of database.
2. **Logging facility**
   - Keeps track of current state of transactions and database changes.
3. **Checkpoint facility**
   - Enables updates to database in progress to be made permanent.
4. **Recovery manager**
   - Allows DBMS to restore database to consistent state following a failure.

Backup mechanism

- DBA must ensure that there are periodic backups of the database
- Many companies store backup copies at an off-site location to minimize the effect of natural disaster catastrophes
- To backup a relational database, the database should be offline, and the prior shutdown of the database needs to be clean.
- Some vendors provide mirroring as a backup facility to avoid the shutdown during backup or some other method to allow the database to remain available to users

Logfile

- Collection of records that represent the history of actions executed by the DBMS
- Contains information about all updates to database:
  - Transaction records.
  - Checkpoint records.
- The log file is often used for other purposes (for example, auditing).

Transaction record

A transaction record in the log must contain:

- Date and timestamp
- Transaction identifier: each transaction requires a unique identifier
- Type of action for the transaction: (transaction start, insert, update, delete, abort, commit).
- Identifier of data item affected by the database action (insert, delete, and update operations).
- Before-image of data item.
- After-image of data item.
- Log management information to easily find other operations for this specific transaction.

Update the log before the data files

- The Write-Ahead Logging Protocol:
    1. Must force the log record to permanent storage ***before*** the corresponding data page gets written to disk.
    2. Must write all log records for a transaction *before commit*.
  - #1 guarantees Atomicity (track all changes).
  - #2 guarantees Durability (do not lose any changes).

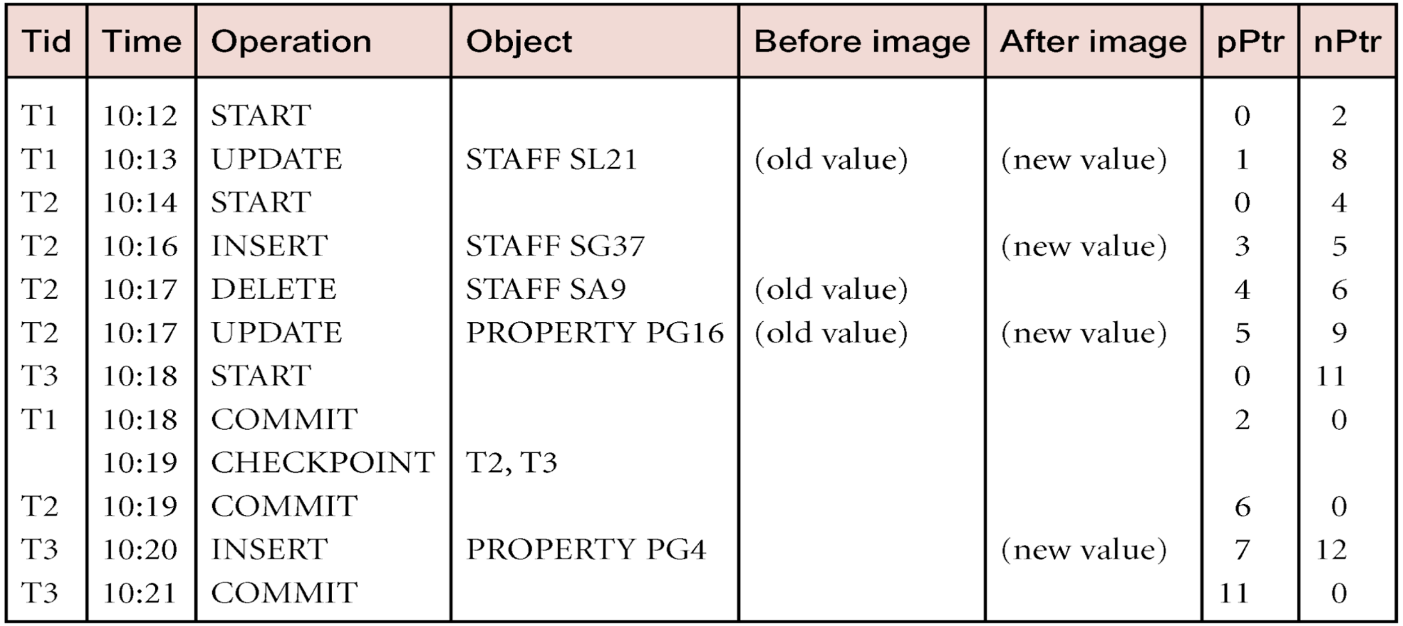
Log file issues

- There must be multiple copies of the log to lower the risk associated with a system failure
- Due to the sheer volume of the log, the log file is typically split into two separate random-access files.
- Log file is a potential bottleneck for a DBMS; it increasingly grows as time passes, so its size is a problem; managing the log file growth is critical to the DBMS' overall performance.

Checkpoint

- Point of synchronization between the database files and the log file. All buffers are force-written to secondary storage.
- A checkpoint record is created in the log, it contains the identifiers of all active transactions.
- When a database failure occurs:
  - Redo all transactions that committed since the checkpoint and
  - Undo all transactions active at time of crash.

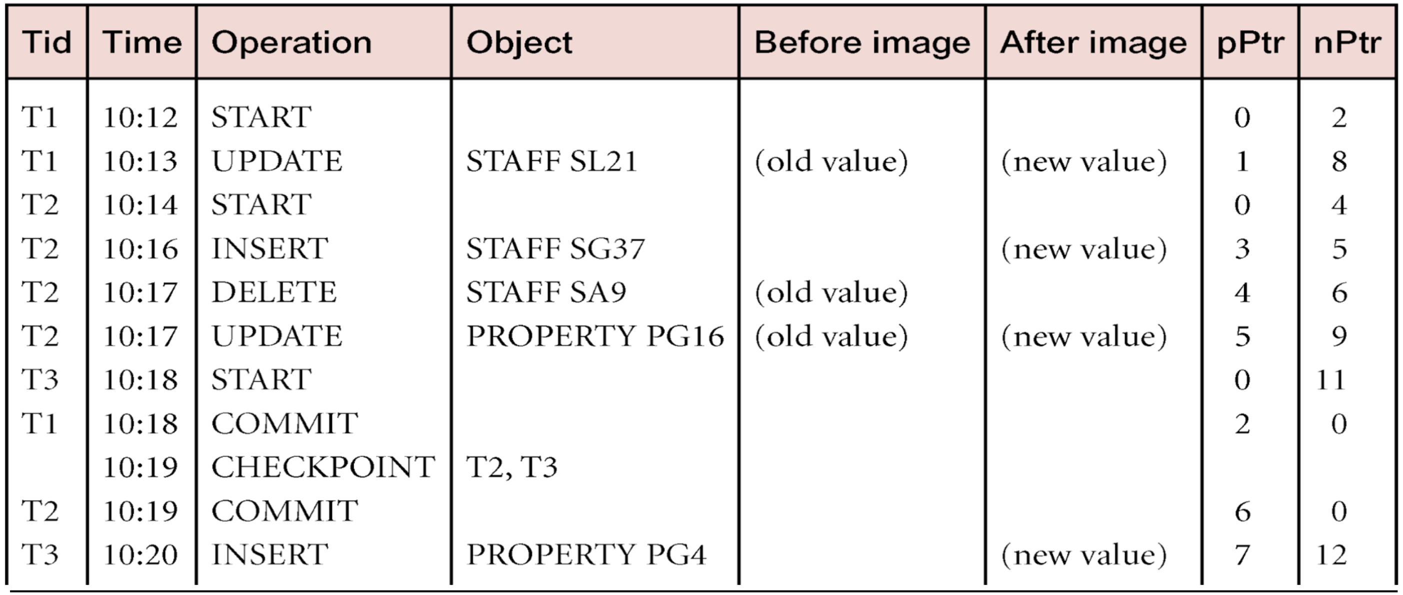
Sample log file (1)



Tid	Time	Operation	Object	Before image	After image	pPtr	nPtr
T1	10:12	START				0	2
T1	10:13	UPDATE	STAFF SL21	(old value)	(new value)	1	8
T2	10:14	START				0	4
T2	10:16	INSERT	STAFF SG37		(new value)	3	5
T2	10:17	DELETE	STAFF SA9	(old value)		4	6
T2	10:17	UPDATE	PROPERTY PG16	(old value)	(new value)	5	9
T3	10:18	START				0	11
T1	10:18	COMMIT				2	0
	10:19	CHECKPOINT	T2, T3				
T2	10:19	COMMIT				6	0
T3	10:20	INSERT	PROPERTY PG4		(new value)	7	12
T3	10:21	COMMIT				11	0

Given the checkpoint, can be guaranteed that T1 data has been flushed to disk. Must make sure T2, T3 changes in DB.

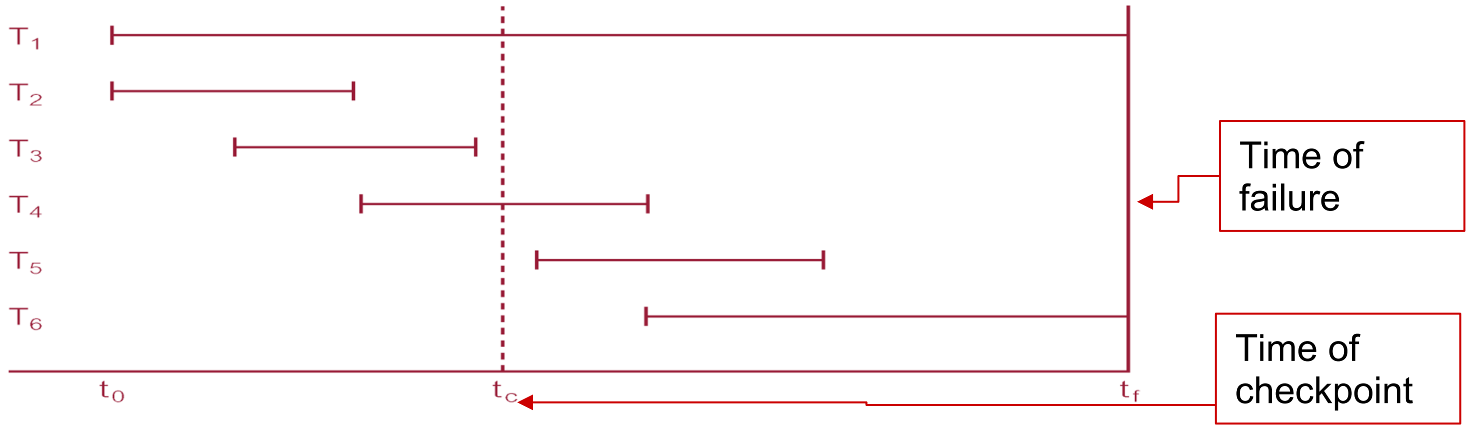
Sample log file (2)



Tid	Time	Operation	Object	Before image	After image	pPtr	nPtr
T1	10:12	START				0	2
T1	10:13	UPDATE	STAFF SL21	(old value)	(new value)	1	8
T2	10:14	START				0	4
T2	10:16	INSERT	STAFF SG37		(new value)	3	5
T2	10:17	DELETE	STAFF SA9	(old value)		4	6
T2	10:17	UPDATE	PROPERTY PG16	(old value)	(new value)	5	9
T3	10:18	START				0	11
T1	10:18	COMMIT				2	0
	10:19	CHECKPOINT	T2, T3				
T2	10:19	COMMIT				6	0
T3	10:20	INSERT	PROPERTY PG4		(new value)	7	12

Given the checkpoint, can be guaranteed that T1 data has been flushed to disk. Must make sure T2 changes to DB and T3 changes are not in database.

Checkpoint Example

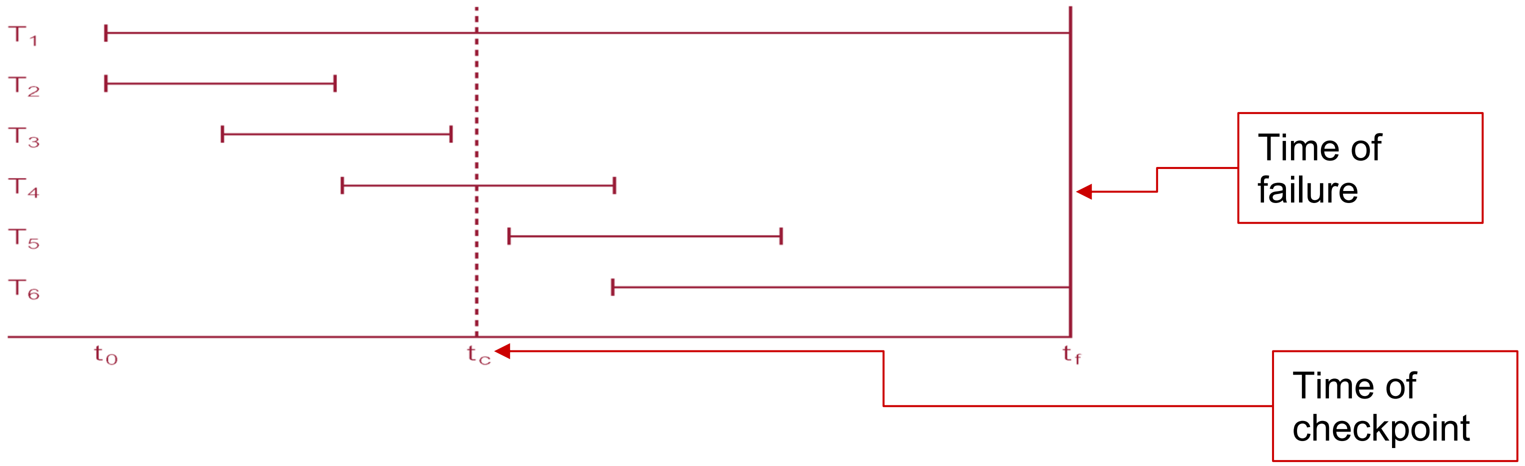


- DBMS starts at time t_0 but fails at time t_f. Assume data for transactions T_2 and T_3 have been written to secondary storage due to the checkpoint at t_c. T_1 and T_6 must be undone.
- In the absence of any other information (such as the checkpoint), recovery manager must redo T_2, T_3, T_4, and T_5.

Redo/Undo with checkpoint

In example below, with checkpoint at time t_c, changes made by T_2 and T_3 have been written to secondary storage.

Thus: redo T_4 and T_5, undo transactions T_1 and T6.



Recovery Techniques

Recovery Manager determines:

If database has been damaged:

Need to restore last backup copy of database and reapply updates of committed transactions using log file.

If database is only inconsistent:

Need to undo changes that caused inconsistency. May also need to redo some transactions to ensure updates reach secondary storage.

Do not need backup but can restore database using before- and after-images in the log file.

Recovery Techniques

Three main recovery techniques:

- Deferred Update
- Immediate Update
- Shadow Paging

Deferred Updates

- Updates are not written to the database until after a transaction has reached its commit point.
- If transaction fails before commit, it will not have modified database and so no undoing of changes required.
- May be necessary to redo updates of committed transactions as their effect may not have reached database.

Steps for deferred update

- Transaction start: create a transaction start record to the log
- For any write operation: create a log record (excluding before image). Do not write the changes to the database buffers or database or the log on disk.
- Transaction commits: create a commit record to the log, write all log records to disk, commit the transaction. **Use the log records to perform the updates**.
- If a transaction aborts, ignore the created log records for the transaction; do not perform the writes to the DB.

Immediate Update

Updates are applied to database at any point after the log is updated.

Need to **redo** updates of committed transactions following a failure.

May need to **undo** effects of transactions that had not committed at time of failure.

Essential that log records are written before write to database, since we do not want to lose an update. This means a **write-ahead log protocol is needed for immediate update.**

Immediate Update

If no “**transaction commit**” record in log, then that transaction was active at failure and must be undone.

Undo operations are performed **in reverse order in which they were written to log.**

Typical protocol used by most relational database systems

Steps: immediate update

- Transaction start: write a transaction start record to the log
- For any write operation: create a log record to the log file (to disk) containing all necessary information.
- Once the log record is written, write the update to the database buffers.
- The DB is updated once the database flushes the buffers to secondary storage
- Transaction commits: write a transaction commit record to the log.
- Transaction aborts: log must be used to undo the transaction. (Writes undone in reverse order).

Shadow Paging

- Maintain two-page tables during the life of a transaction: the **current** page table and the **shadow** page table.
- When transaction starts, the current page and the shadow page are the same.
- Shadow page table is never changed and is used to restore database in event of failure (before commit).
- During the transaction, the current page table records all updates.
- When transaction completes, current page table becomes shadow page table (new version of the page).
- An unpopular, old solution also referred to as old-master new-master.

Summary

- There are four necessary recovery facilities
  - Recovery Manager
  - Database backup facility
  - Database logging facility
  - Checkpoint facility
- Recovery Manager guarantees Atomicity and Durability.
- The recovery of a database is dependent on the type of failure the database encountered
  - If the current version of the database is not recoverable use the log and a backup version of the database to get the database to a consistent state
  - If the current version of the database is recoverable and in an inconsistent state, then use the log with the current version of the database to recover from the failure.
- Checkpointing: A quick way to limit the amount of recovery work that needs to be done during recovery