Database Recovery Techniques

Chapter 19

Chapter Outline

- Purpose of Database Recovery
- Types of Failure
- Transaction Log
- Transaction Roll-back (Undo) and Roll-Forward
- Data Updates
- Data Caching
- Checkpointing
- Recovery schemes

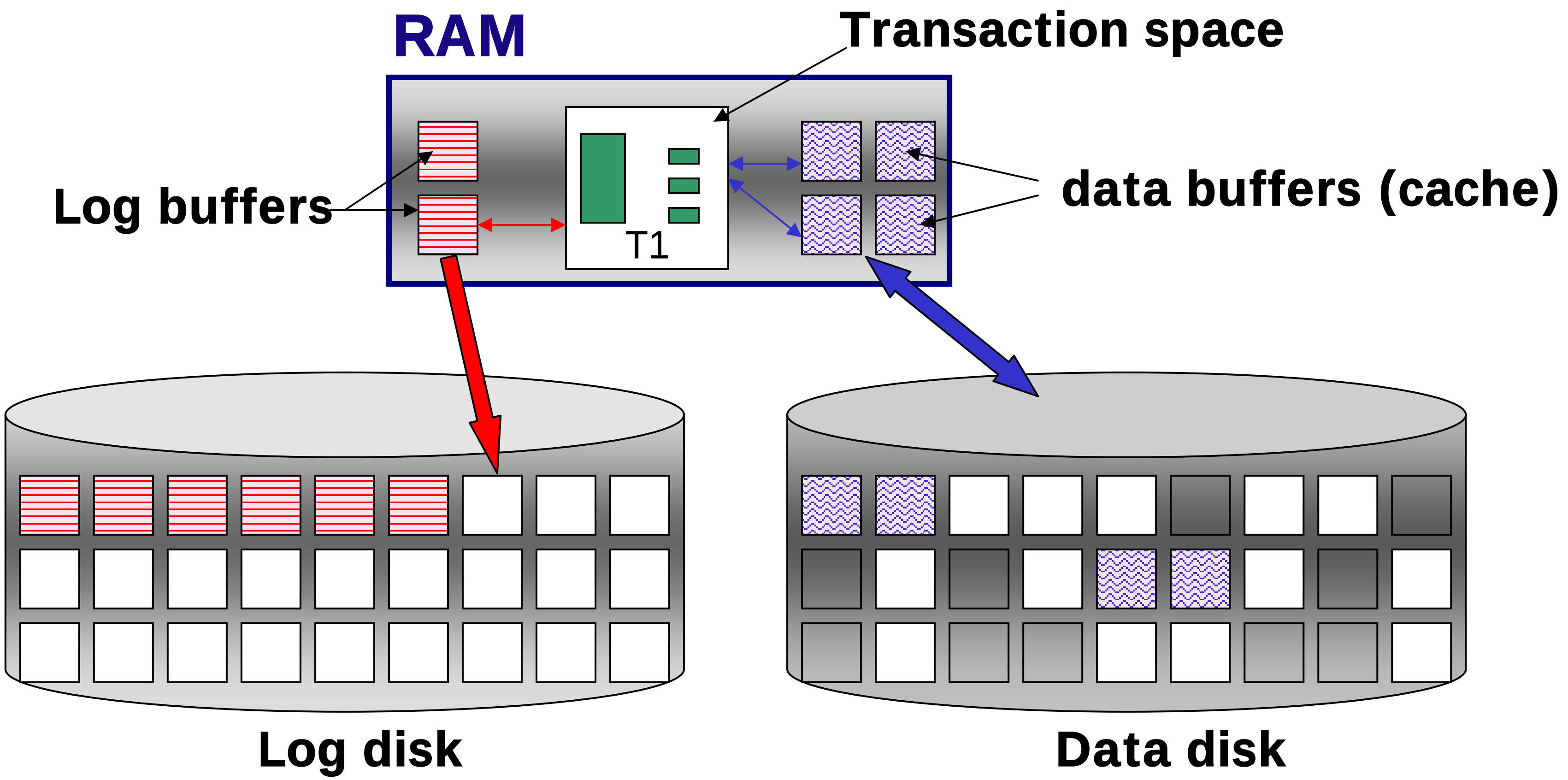
- Purpose of Database Recovery

- To bring the database into the last consistent state, which existed prior to the failure
- To preserve transaction ACID properties.
  - Example: If the system crashes before a fund transfer transaction completes its execution, then either one or both accounts may have incorrect value. Thus, the database must be restored to the state before the transaction modified any of the accounts

- Types of Failure

- The database may become unavailable for use due to:
  - Transaction failure: Transactions may fail because of incorrect input, deadlock, incorrect synchronization
  - System failure: System may fail because of addressing error, application error, operating system fault, RAM failure, etc
  - Media failure: Disk head crash, power disruption, etc.

- Transaction Log ...



... - Transaction Log

- For recovery from any type of failure data values prior to modification (BFIM - BeFore Image) and the new value after modification (AFIM – AFter Image) are required. These values and other information is stored in a sequential file called Transaction log. A sample log is given below. **Back P** and **Next P** point to the previous and next log records of the same transaction.

T ID	Back P	Next P	Operation	Data item	BFIM	AFIM
T1	0	1	Begin			
T1	1	4	Write	X	X = 100	X = 200
T2	0	8	Begin			
T1	2	5	W	Y	Y = 50	Y = 100
T1	4	7	R	M	M = 200	M = 200
T3	0	9	R	N	N = 400	N = 400
T1	5	nil	End			

- Roll-back (Undo) and Roll-Forward (Redo) ...

- To maintain atomicity, a transaction's operations are redone or undone**.**
  - **Undo**: Restore all BFIMs on to disk (Remove all AFIMs).
  - **Redo**: Restore all AFIMs on to disk
- Database recovery is achieved either by performing only Undos or only Redos or by a combination of the two. These operations are recorded in the log as they happen

... - Roll-back and Roll-Forward ...

- We show the process of roll-back with the help of the following three transactions T1, and T2 and T3.

T1	T2	T3
read_item(A)	read_item(B)	read_item(C)
read_item(D)	write_item(B)	write_item(B)
write_item(D)	read_item(D)	read_item(A)
write_item(A)	write_item(A	

... - Roll-back and Roll-Forward ...

- One execution of T1, T2 and T3 as recorded in the log.

		A	B	C	D
		30	15	40	20
	[start_transaction, T3]				
	[read_item, T3, C]				
*	[write_item, T3, B, 15, 12]		12		
	[start_transaction,T2]				
	[read_item, T2, B]				
**	[write_item, T2, B, 12, 18]		18		
	[start_transaction,T1]				
	[read_item, T1, D]				
	[write_item, T1, D, 20, 25]				25
	[read_item, T2, D]				
**	[write_item, T2, D, 25, 26]				26
	[read_item, T3, A]				

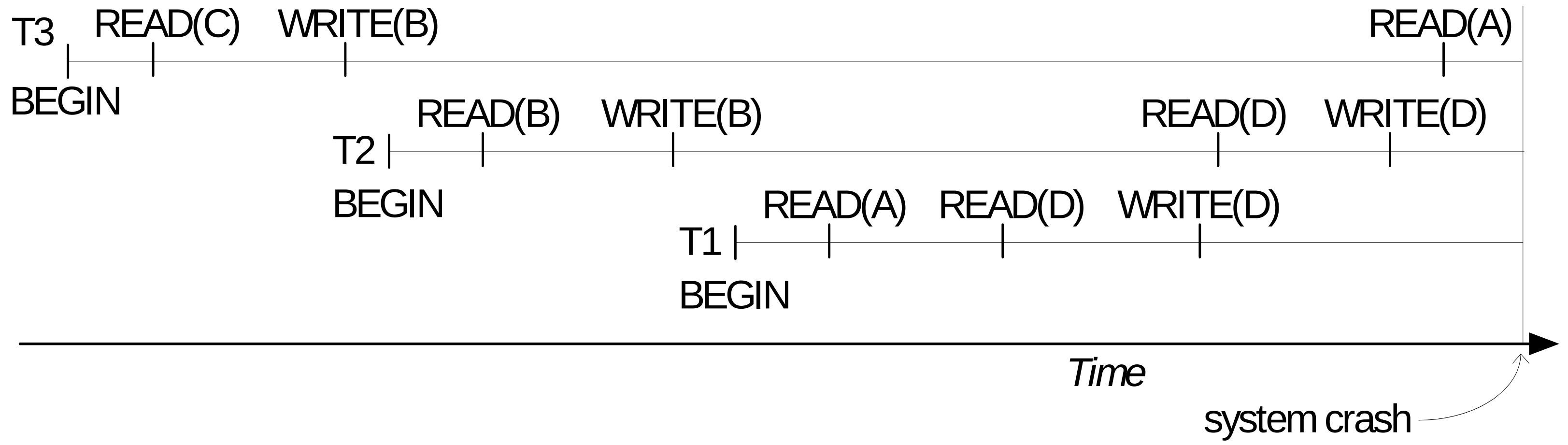
---- system crash ----

* T3 is rolled back because it did not reach its commit point.

** T2 is rolled back because it reads the value of item B written by T3.

... - Roll-back and Roll-Forward

- One execution of T1, T2 and T3 as recorded in the log.



Illustrating cascading roll-back

- Data updates

- **Immediate Update**: As soon as a data item is modified in cache, the disk copy is updated.
- **Deferred Update**: All modified data items in the cache is written either after a transaction ends its execution or after a fixed number of transactions have completed their execution
- **Shadow update**: The modified version of a data item does not overwrite its disk copy but is written at a separate disk location
- **In-place update**: The disk version of the data item is overwritten by the cache version

- Data Caching

- Data items to be modified are first stored into database cache by the Cache Manager (CM) and after modification they are flushed (written) to the disk.
- The flushing is controlled by **Modified** and **Pin-Unpin** bits.
- **Pin-Unpin**: Instructs the operating system not to flush the data item.
- **Modified**: Indicates the AFIM of the data item.

-- Write-Ahead Logging

- When **in-place** update (immediate or deferred) is used then log is necessary for recovery and it must be available to recovery manager. This is achieved by **Write-Ahead Logging** (WAL) protocol

- WAL states that:
  - Before a data item's AFIM is flushed to the database disk (overwriting the BFIM) its BFIM must be written to the log and the log must be saved on a stable store (log disk).
  - Before a transaction executes its commit operation, all its AFIMs must be written to the log and the log must be saved on a stable store.

- Steal/No-Steal and Force/No-Force

- Possible ways for flushing database cache to database disk:
  - If a cache page updated by a transaction cannot be written to disk before the transaction commits, this is called **no-steal** approach. Otherwise it is **steal**.
  - If all cache pages updated by a transaction are immediately written to disk when a transaction commits, this is called a **force** approach. Otherwise, it is called **no-force**.
- These give rise to four different ways for handling recovery:
  1. Steal/No-Force (Undo/Redo)
  2. Steal/Force (Undo/No-redo)
  3. No-Steal/No-Force (No-undo/Redo) and
  4. No-Steal/Force (No-undo/No-redo).

- Checkpointing

- From time to time (randomly or under some criteria) the database flushes its buffer to database disk to minimize the task of recovery. The following steps defines a checkpoint operation

  1. Suspend execution of transactions temporarily.
  2. Force write modified buffer data to disk.
  3. Write a [checkpoint] record to the log, save the log to disk.
  4. Resume normal transaction execution.

- During recovery **redo** or **undo** is required to transactions appearing after [checkpoint] record.

- Recovery Schemes

- Deferred Update (No Undo/Redo)
- Deferred Update in a single-user system
- Deferred Update with concurrent users
- Recovery Techniques Based on Immediate Update

-- Deferred Update (No Undo/Redo)

- The data update goes as follows:
  - A set of transactions records their updates in the log.
  - At commit point under WAL scheme these updates are saved on database disk.
- After reboot from a failure the log is used to redo all the transactions affected by this failure. No undo is required because no AFIM is flushed to the disk before a transaction commits.

-- Deferred Update in a single-user system

- There is no concurrent data sharing in a single user system. The data update goes as follows
  1. A set of transactions records their updates in the log.
  2. At commit point under WAL scheme these updates are saved on database disk.
- After reboot from a failure the log is used to redo all the transactions affected by this failure. No undo is required because no AFIM is flushed to the disk before a transaction commits.

-- Deferred Update in a single-user system

(a)

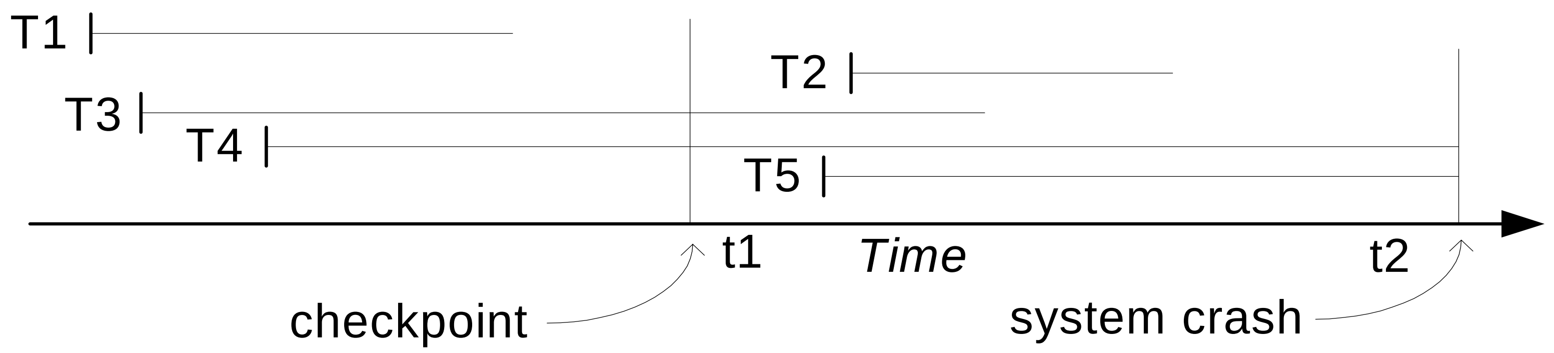
T1	T2
read_item (A)	read_item (B)
read_item (D)	write_item (B)
write_item (D)	read_item (D)
	write_item (A)

(b)

[start_transaction, T1]
[write_item, T1, D, 20]
[commit T1]
[start_transaction, T1]
[write_item, T2, B, 10]
[write_item, T2, D, 25] ← system crash

The [write_item, ...] operations of T1 are redone.
T2 log entries are ignored by the recovery manager.

-- Deferred Update with concurrent users ...



- T1: No need to do anything for it committed before checkpoint.
- T2 and T3: are redone
- T4 and T5: Ignored

Note: In **deferred updates**, data disk are updated after transaction commit

...-- Deferred Update with concurrent users ...

(a)

T1	T2	T3	T4
read_item (A)	read_item (B)	read_item (A)	read_item (B)
read_item (D)	write_item (B)	write_item (A)	write_item (B)
write_item (D)	read_item (D)	read_item (C)	read_item (A)
	write_item (D)	write_item (C)	write_item (A)

(b)

```
[start_transaction, T1]
[write_item, T1, D, 20]
[commit, T1]
[checkpoint]
[start_transaction, T4]
[write_item, T4, B, 15]
[write_item, T4, A, 20]
[commit, T4]
[start_transaction T2]
[write_item, T2, B, 12]
[start_transaction, T3]
[write_item, T3, A, 30]
[write_item, T2, D, 25]   ← system crash
```

T2 and T3 are ignored because they did not reach their commit points.
T4 is redone because its commit point is after the last checkpoint.

...-- Deferred Update with concurrent users ...

- Two tables are required for implementing this protocol:
  - **Active table**: All active transactions are entered in this table.
  - **Commit table**: Transactions to be committed are entered in this table.
- During recovery, all transactions of the **commit** table are redone and all transactions of **active** tables are ignored since none of their AFIMs reached the database. It is possible that a **commit** table transaction may be **redone** twice but this does not create any inconsistency because of a redone is "**idempotent**", that is, one redone for an AFIM is equivalent to multiple redone for the same AFIM.

-- Recovery Techniques Based on Immediate Update ...

- **Undo/No-redo Algorithm**
  - In this algorithm AFIMs of a transaction are flushed to the database disk under WAL before it commits. For this reason the recovery manager **undoes** all transactions during recovery.
  - No transaction is **redone**.
  - It is possible that a transaction might have completed execution and ready to commit but this transaction is also **undone**.

... -- Recovery Techniques Based on Immediate Update ...

- **Undo/Redo Algorithm (Single-user environment)**
  - Recovery schemes of this category apply **undo** and also **redo** for recovery.
  - In a single-user environment no concurrency control is required but a log is maintained under WAL.
  - Note that at any time there will be one transaction in the system and it will be either in the commit table or in the active table.
  - The recovery manager performs:
    1. **Undo** of a transaction if it is in the **active** table.
    2. **Redo** of a transaction if it is in the **commit** table.

… - Recovery Techniques Based on Immediate Update

- **Undo/Redo Algorithm (Concurrent execution)**
  - Recovery schemes of this category applies undo and also redo to recover the database from failure.
  - In concurrent execution environment a concurrency control is required and log is maintained under WAL.
  - Commit table records transactions to be committed and active table records active transactions.
  - To minimize the work of the recovery manager checkpointing is used.
  - The recovery performs:
    1. **Undo** of a transaction if it is in the active table.
    2. **Redo** of a transaction if it is in the commit table.

END