TRANSACTION MANAGEMENT - 2

U08049 Database Design - 2013

Learning outcomes

- Understand how exclusive locks may cause deadlock
- Explain how deadlock may be countered by either prevention, or detection and recovery
- Explain procedures for both
- Understand the concept of ‘non-catastrophic’ database failure
- Explain how recovery from failure may be achieved
- Understand the use of transaction logs and checkpoints in the recovery process

Deadlock

- Arises when two or more transactions are each waiting for locks held by the other.
- CPU may process other concurrent transactions, but the deadlocked could wait forever.

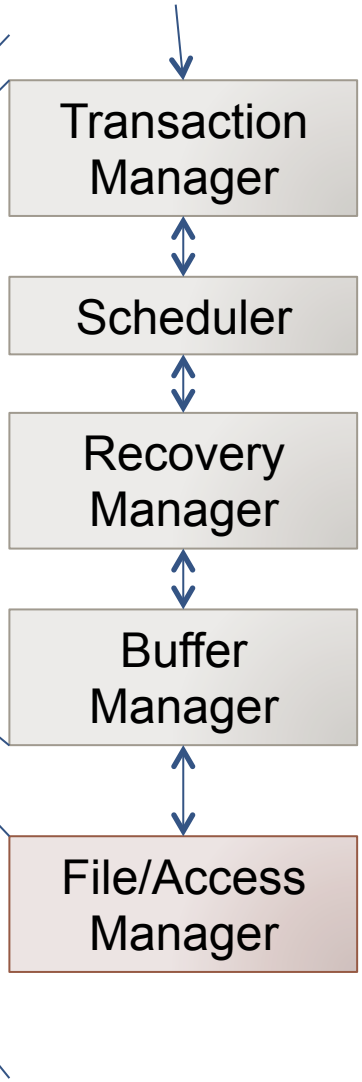
Database Management Systems

APPLICATION LAYER – programs, queries, etc. run in main memory

DBMS– provides support for transactions, including scheduling and recovery

OPERATING SYSTEM – providing file management and access management of secondary storage

SECONDARY STORAGE– stores data and system catalog PERMANENTLY.



Deadlock between 2 transactions

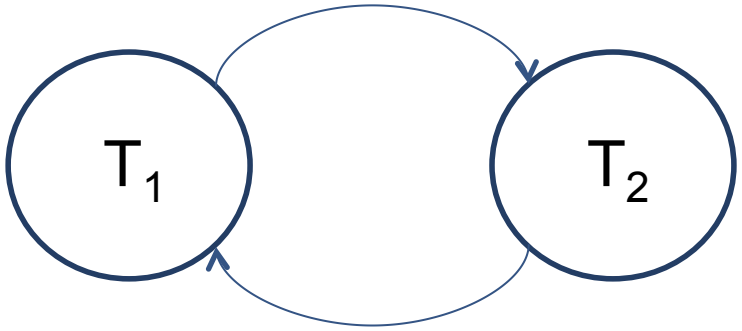
time	T1	T2
t_1	begin_transaction	
t_2	write_lock(bal_x)	
t_3	read(bal_x)	
t_4	$bal_x = bal_x - 10$	
t_5	write(bal_x)	
t_6	write_lock(bal_y)	
t_7	WAIT	begin_transaction
t_8	WAIT	write_lock(bal_y)
t_9	WAIT	read(bal_y)
t_{10}		$bal_y = bal_y + 100$
t_{11}		write(bal_y)
		write_lock(bal_x)
		WAIT
		WAIT
		WAIT

Deadlock prevention

- Timeout
  - Limit the waiting time for a lock
  - Abort transaction after this time (no check for deadlock)
  - Automatically restart the transaction
- Timestamps
  - Recorded for each transaction start
  - Various algorithms may be used to prevent deadlock
  - E.g. ‘wait-die’ where only an older transaction may wait for a younger – a younger transaction dies, and is restarted with the same timestam
- Deadlock prevention (pessimistic – large overhead)

Deadlock Detection

- Wait-for graphs (WFG) cf State Machine
  - Create a node for each transaction
  - Add a directed edge T_i -> T_j if T_i is waiting to lock an item currently locked by T_j;
  - Deadlock exists if and only if the WFG contains a cycle.
- Deadlock detection and recovery (optimistic – recovery is more expensive than prevention, but shouldn't be needed as much.



Locking and Granularity

- Different approaches may be used (even multiple levels within one DBMS);
  - Entire database
  - File
  - Page
  - Record
  - Field

Database Recovery

- The process of restoring a database to a ‘stable state’ after failure
  - Stable state is consistent, and with all transactions in a known state
- Recovery from catastrophic failure (e.g. Secondary storage destroyed by fire) relies on appropriate backup procedures or replication.
- System crashes may cause loss of main memory, but recovery will be faster than above.
- Non-catastrophic failure is any failure that allows recovery

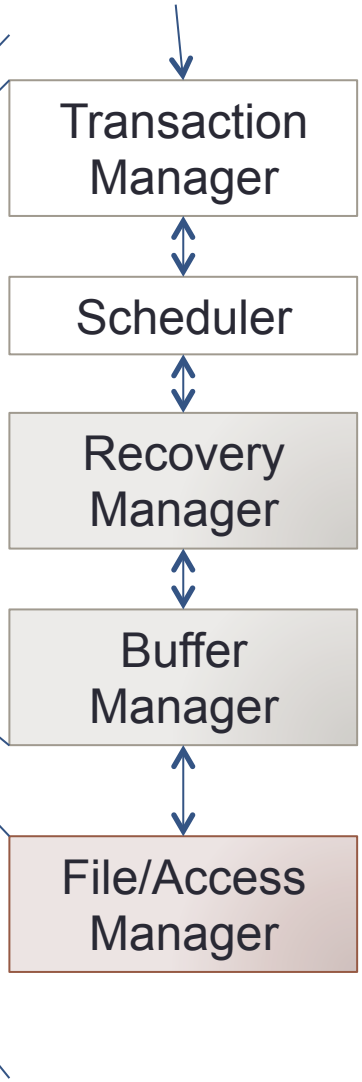
Database Management Systems

APPLICATION LAYER – programs, queries, etc. run in main memory

DBMS– provides support for transactions, including scheduling and recovery

OPERATING SYSTEM – providing file management and access management of secondary storage

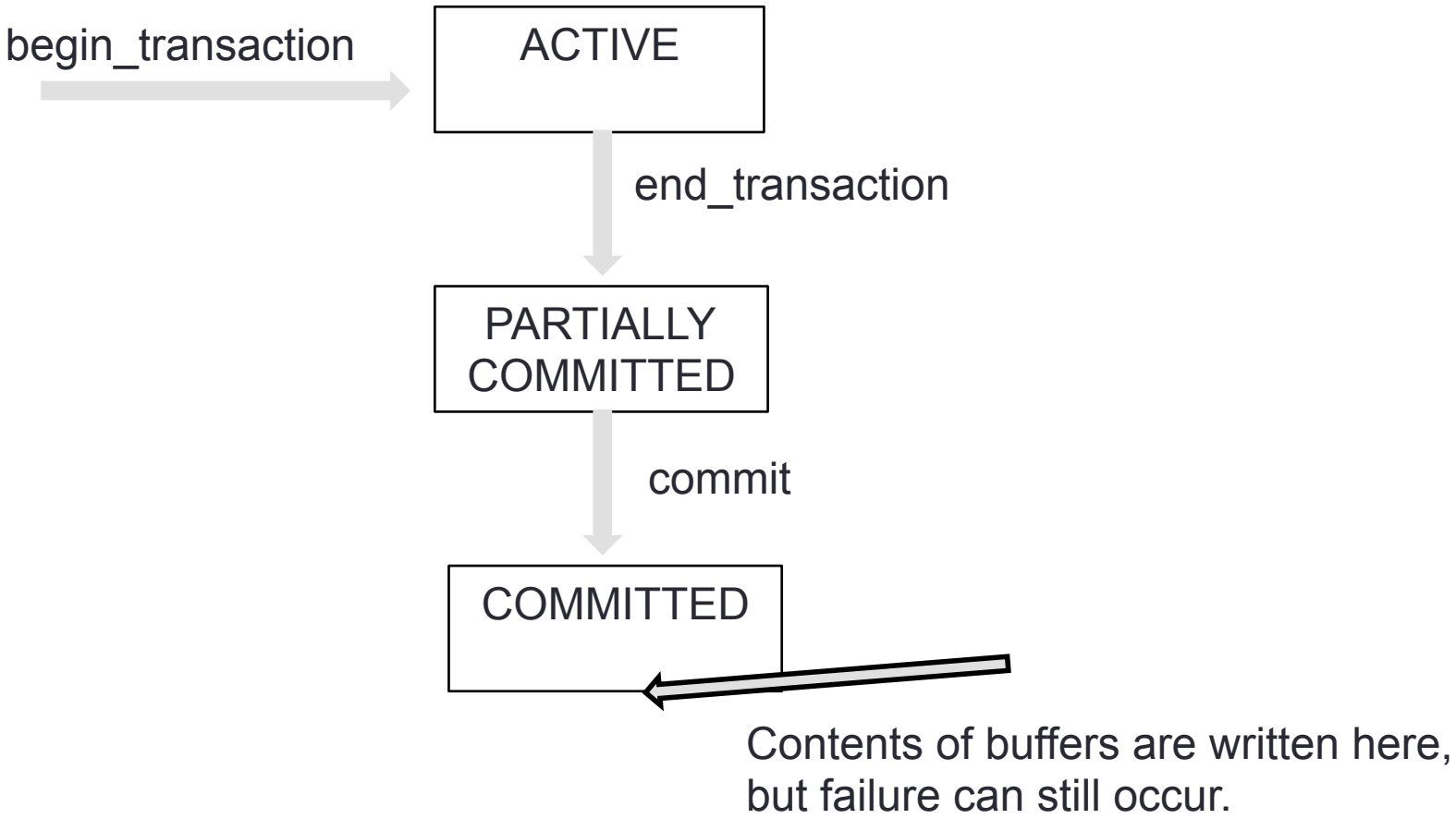
SECONDARY STORAGE– stores data and system catalog PERMANENTLY.



Recovery manager

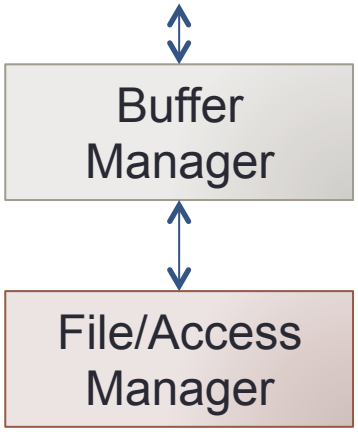
- In the event of failure:
  - Guarantees atomicity
  - Guarantees durability of transactions
- Complicated by the contents of the buffer at fail time
  - ‘Committed’ transactions may not have had updates permanently stored – failure can occur at any time in the life of a transaction

Lifetime of a ‘Successful’ State



Buffer and File/access Managers

- read(bal_x)
  - Find the address of the disk block that contains the required record
  - Transfer the disk block into a database buffer in main memory
  - Copy the salary data from the database buffer into the variable ‘salary’
- write(bal_x)
  - Find the address of the disk block that contains the record
  - Transfer the disk block into a database buffer in main memory
  - Copy the salary data from the variable ‘salary’ into the database buffer
  - Write the database buffer back to disk



UNDO/REDO

- For any recovery mechanism to work, a DBMS needs :
  - An UNDO operation, which will wind-back any updates to disk of a transaction in specific situations (see later)
  - A REDO operation, which will re-make updates to disk made by a transaction what reached its commit point, but did not get its updates permanently stored.

Buffer Management 1

- As read and write operations are executed, ‘pages’ are read from the disk, until the buffers are full
- Further read/write operations require pages to be swapped – one out for every one in
- Replacement strategies:
  - First-in-first-out (FIFO)
  - Least-recently-used (LFU)

Buffer Management 2

- Recovery policies:
  - A steal policy, allows the buffer manager to write a buffer to disk before a transaction using the buffer has committed. The alternative is ‘no-steal’.
  - A force policy forces the immediate writing of an updated page in a buffer, to disk. The alternative is no-force.
- Most DBMSs employ a steal, no-force policy
  - Steal avoids the need for large buffer space
  - No-force means that multiple updates to the same page in buffer, do not require multiple writes to disk

Recovery Facilities

- Backup facility – copying the content of a database at regular intervals
- Logging facilities – for recording the state of each transaction
- Checkpoint facilities – for forcing permanent updates
- Recovery manager – for supervising the UNDO and REDO operations necessary for recovery to a stable state.

Backup Facility

- Complete copy or incremental (only backup the changes since the last complete backup)
- Should backup the log file as well as the database
- Made to offline storage
- Frequency of backups is traded against time to recover

Log File

- Complete record of all transactions and their effects
- Typically contains:
  - Transaction ID
  - Record type (start, insert, update, delete, abort, commit)
  - Object ID for the data item altered
  - Before-value of the data item (for update and delete)
  - After-value of the data item (for insert and update)
  - Pointers to previous and next log record for the transaction

Use of the Log File

- Clear to see how the contents can be used to UNDO and REDO transaction operations
- Multiple copies may be maintained to improve the recovery process
- The Log File itself needs writing to disk regularly, hence the need for....

Checkpoints

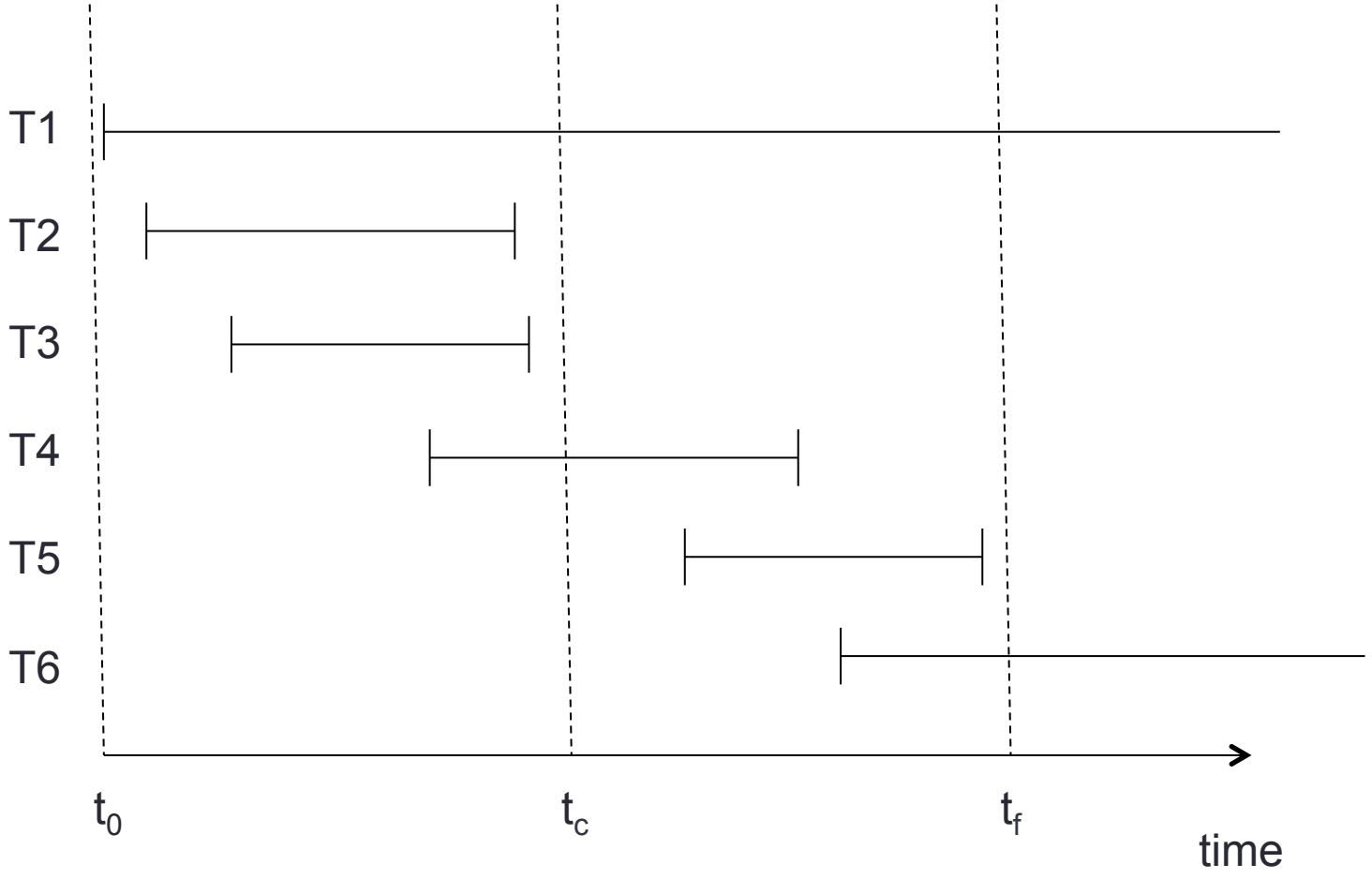
- Checkpoints mark a known state of the buffers, the log file and the database on disk
- Written to the log file, and stored on disk, at the same time as all modified buffers and the log file are written to disk
- Recovery can be made from the disk only – by inspecting the last stored log file

Recording a checkpoint

- The Recovery Manager will execute the following operations:
  1. Write log file to disk
  2. Write all modified pages in buffers to disk
  3. Write a checkpoint record to the log file
  4. Write the log file to disk

- The additional writing of the log file ensures recovery in the case of a failure during checkpointing

Transaction logs with checkpoints



Log records

- Time t_c is the last checkpoint made:
  - The log file on disk will contain all operations from T1 – T4 up until the time t_c
- If failure occurs at time t_f:
  - Recover the log file from ***main memory (contains operations of T1 to T6)***

Recovery (where deferred updates)

- Deferred update means that updates from transactions are not written to the disk until the transaction commits
- At time t_f, any transaction with a start record and a commit record in the log file (e.g. T4 and T5) need to be redone
  - Perform all write operations using the after-value in the log file – in the order they were written
  - Effort wasted but no ill effects if the writing has already been done
- Transactions with a commit before the checkpoint entry can be assumed done (e.g. T2, T3)
- All other can be resubmitted (T1, T6)

Recovery (where immediate updates)

- Immediate update means that updates from transactions are written to the disk as they are executed
- At time t_f, any transaction with a start record and a commit record in the log file (e.g. T4 and T5) need to be redone, as before
- Transactions with a commit before the checkpoint entry can be assumed done (e.g. T2, T3)
- Transactions with start but no commit (e.g. T1, T6) need to be undone
  - Use the before-value from the transaction records in the log file in ***reverse order***

Summary

- Deadlock may occur when two databases are waiting for each other to commit
- DBMS are designed to resolve this issue
- Buffer management
- Log records
- Recovery