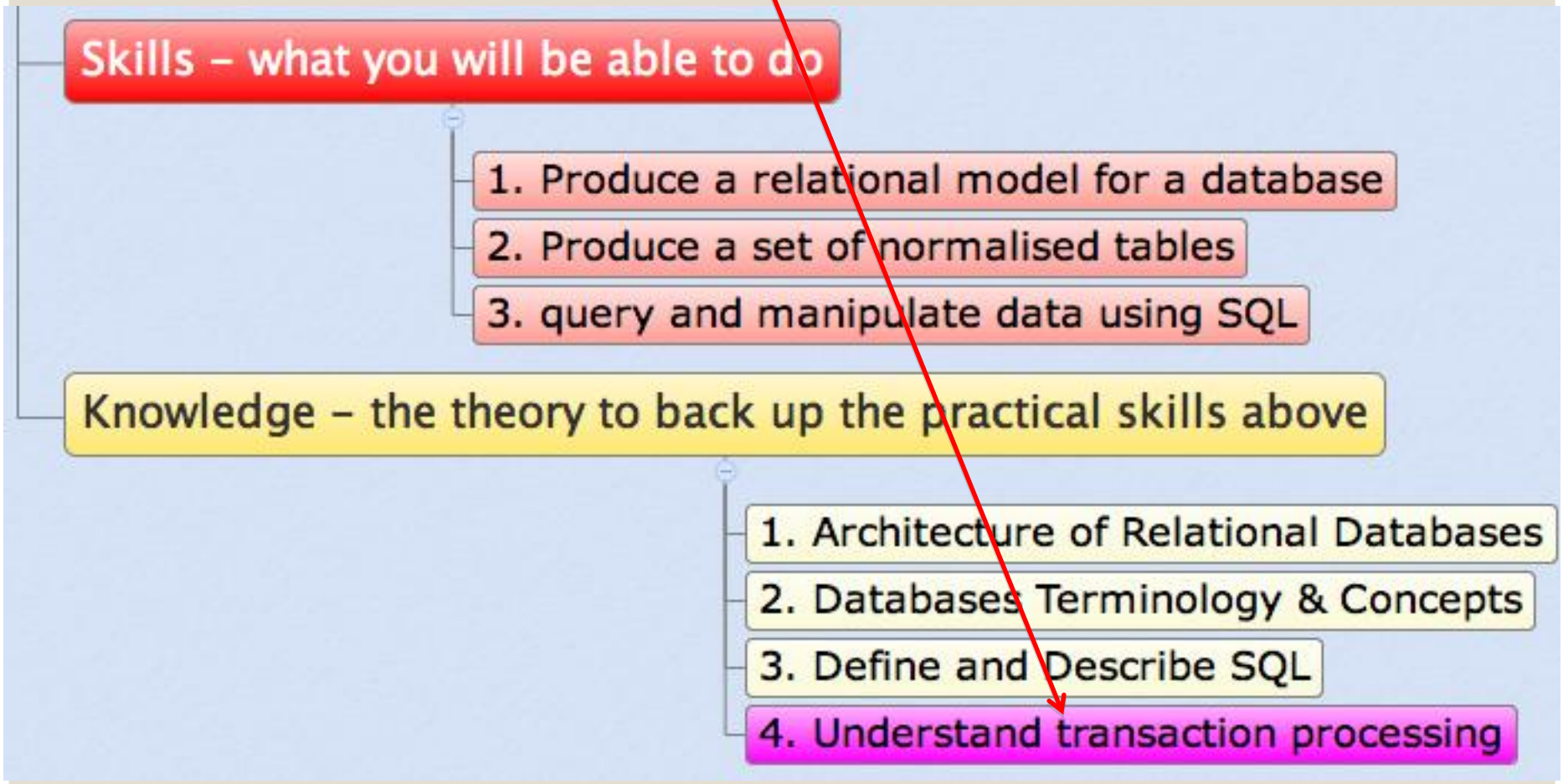
Module: Database Fundamentals

Lecture 10:

Transaction Processing

Learning Outcomes



Topics to Cover:

- ⌘ Concept of a database Transaction
- ⌘ Commit and Rollback
- ⌘ ACID Properties of a Transaction
- ⌘ Three Concurrency Problems
- ⌘ Locking, 2 phase locking, deadlock
- ⌘ Recovery

Ref: Relational Database Principles – Colin Ritchie
Chapter 10

What is a database transaction?

A database transaction is all the data additions, updates and deletes associated with a business event, such as

- placing an order
- purchasing goods
- generating an invoice

A single transaction on a database usually involves a number of reads (SELECT) and writes (UPDATE/INSERT).

Example: A customer orders 10 XBoxes

Business Logic:

Read number of XBoxes in stock
If > 10 then
  reduce number of XBoxes in stock by 10
  Set number ordered to 10
Else
  . . . .
End-If

Database Transaction

Read(qtyInStock**)**
Read(qtyOrdered**)**

qtyInStock = qtyInStock – 10;
qtyOrdered = 10;

write(qtyInStock **)**
write(qtyOrdered**)**
Write(OrderDetails**)**

Stock Table:

	Part ID	desc	QtyIn Stock
Before transcation	X0100	XBox	100
After transaction	X0100	XBox	**90**

OrderDetails Table:

Order Num	Part ID	Qty Ordered
O-5412	**X0100**	**10**

The database transaction is defined as follows:
T={**Read(**qtyInStock**), Read(**qtyOrdered**), Write(**qtyInStock**), Write(**qtyOrdered**), write(**orderDetails**)}**

Running a transaction

T ={**Read(**qtyInStock**),**
Read(qtyOrdered**),**
Write(qtyInStock**),**
Write(qtyOrdered**),**
Write(orderDetails**) }**

These operations MUST either ALL be executed, or none of them executed

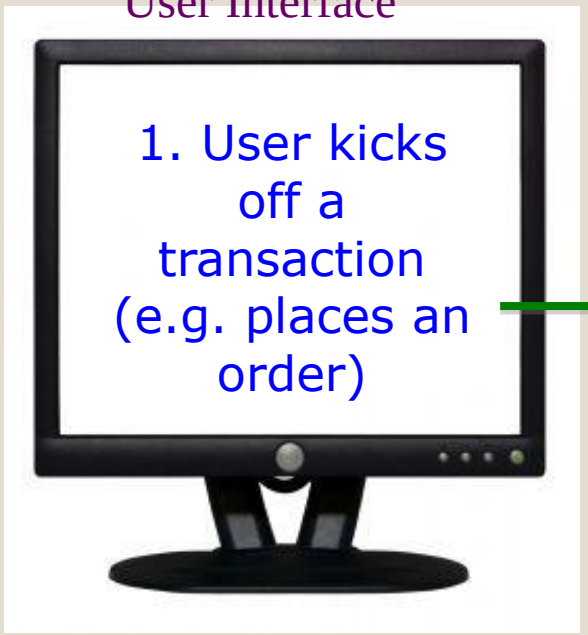
A delay between read and write means data used in calculation may no longer be accurate

If another transaction can read the data between the read and write, they are using out-of -date information

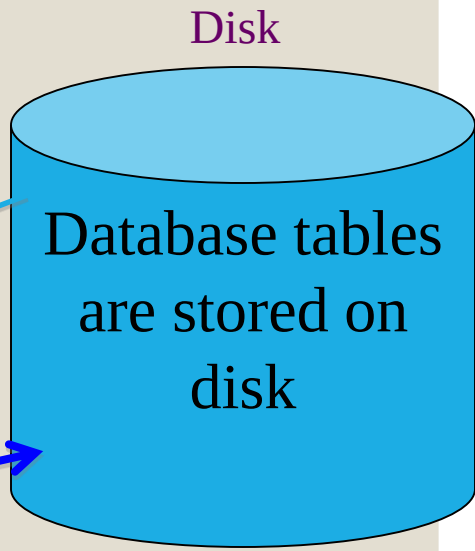
If the database crashed between the two writes, then the order and stock files will be out of sync.

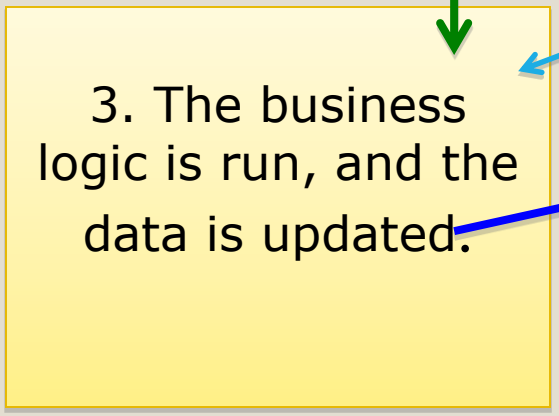
All operations in a transaction must be treated as one, atomic, unit of work

What happens under the cover . . .



2. The data required by the transactions is read into memory





4. Sometime later, the updated data is written back to disk.

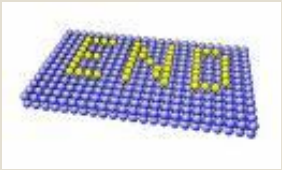
So a Database Transaction is . . .

- . . a group of changes and/or queries to a database which, for the purposes of database integrity, must be performed as a single unit.
- It is the DBMS's responsibility to ensure ALL or NONE of the transactions operations are executed.
- TO do this, the DBMS must be told when a transaction starts and ends.

- In SQL, **BEGIN** denotes the start of a transaction.
- There are two ways to denote the end of a transaction as explained on the next slide. . .

Ending a transactions

COMMIT denotes the end of a transaction if it completes normally.



Sometimes a transaction can not be completed. For example in the following scenarios:

- computer failure / power failure / network failure / item not in stock

In these cases, the transaction must be '**undone**', **by issuing a ROLLBACK command**.

Commit and Rollback

- A ***COMMIT*** statement signals the successful end of a series of updates within one transaction.
- It tells the DBMS to save all the amended data to disk, and so end the current transaction

- ***ROLLBACK*** statement aborts the current transaction.
- Updates are not copied to disk. The DataBase reverts to it's state before the start of the transaction

Commit Transaction Example (Ordering 10 items of product P234)

BEGIN

- Enter Order data into Order Table

```
INSERT INTO OrderDetail
VALUES ('Order001', 'P234', 10);
```

- Amend the Stock Table

```
UPDATE Stock
SET Qty-in-stock = Qty-in-stock - 10
WHERE ProductCode = 'P234';
```

- Complete the Transaction normally

```
COMMIT;
```

Rollback Transaction Example (Ordering 25 items of product P234 when there are only 10 in stock)

BEGIN

Enter Order data into Order Table

```
INSERT INTO OrderDetail
VALUES ('Order001', 'P234', 25);
```

Amend the Stock Table

```
UPDATE Stock
SET Qty-in-stock = Qty-in-stock - 25
WHERE ProductCode = 'P234';
```

***returns error, not enough in stock in to fill the order

Abort the Transaction

```
ROLLBACK;
```

DATABASE INTEGRITY AND THE ACID PROPERTIES

ACID Properties of a Transaction

- The DBMS is responsible for executing transactions correctly

- This is measured by the ACID properties of:
  - Atomicity
  - Consistency
  - Isolation
  - Durability

ACID Properties

- **Atomicity** - a transaction must be treated by the DBMS as an atomic unit of work. All or nothing approach.

  Either ALL operations in the transaction are executed, or none of the operations are executed.

- **Consistency** - a transaction must transform the DB from one consistent state to another consistent state

- **Isolation** - updates done by a transaction cannot be made visible until the transaction is complete

- **Durability** - when the transaction completes successfully the changes are recorded permanently and the results must not be lost because of subsequent failure.

Implementing Acid Properties

- The following three services ensure ACID properties are maintained by a DBMS
  - COMMIT / ROLLBACK commands
  - Concurrency Control (ensuring multiple transactions executing at the same time do not violate the ACID properties)
  - Recovery (being able to restore the database to a consistent state after a failure)

CONCURRENCY CONTROL

Concurrency Control

- Concurrency Control means controlling concurrent access to an item of data by multiple users

  i.e. two or more users trying to read/update the same data at the same time

- **There are Three Classical Concurrency Problems**
  1. Lost Update
  2. Temporary Update/Uncommitted Dependency
  3. Incorrect Summary Problem

1. Lost Update

- The lost update problem refers to one user over-writing another user's updates.
- This can occur as follows . . .

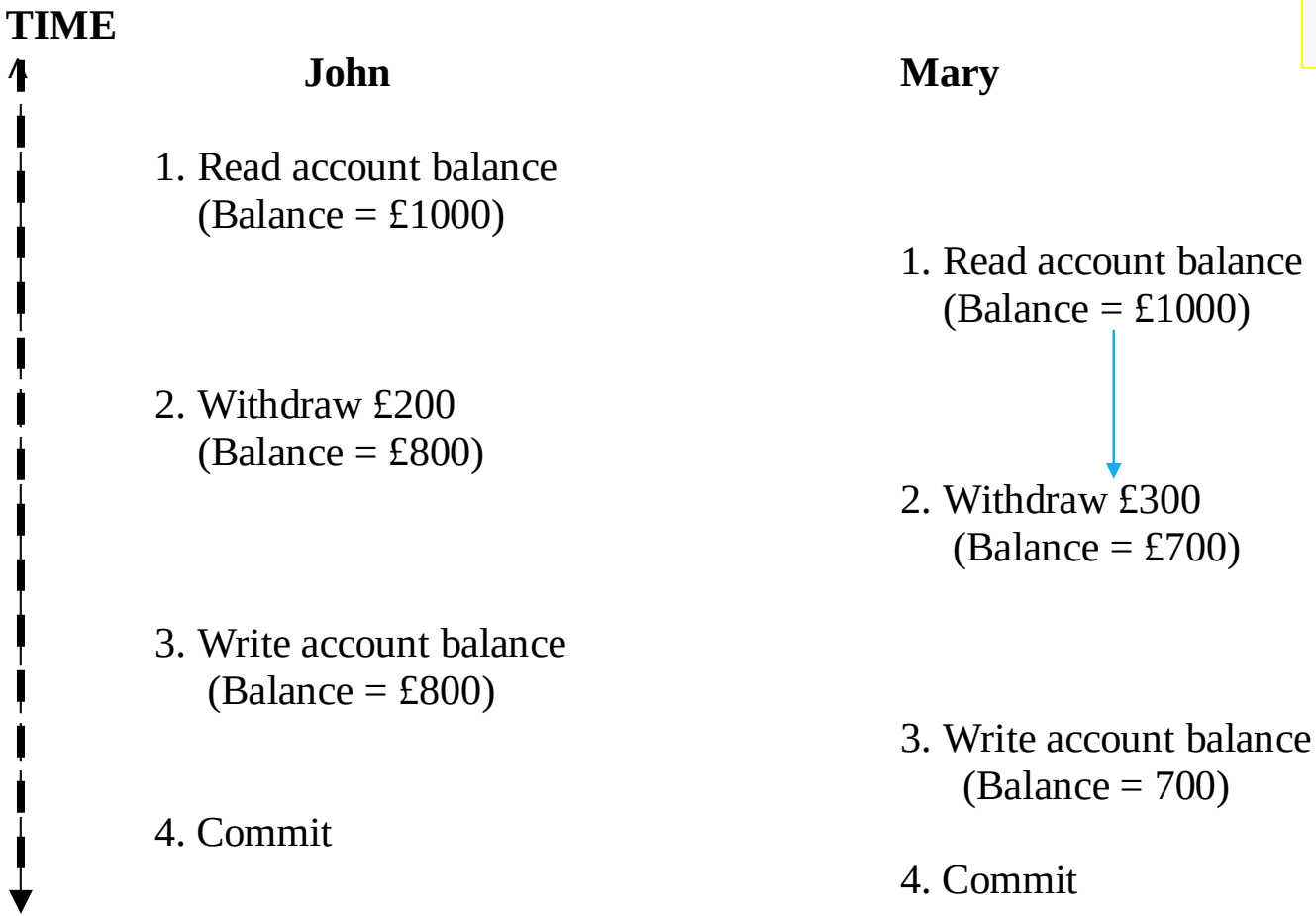
Lost Update

- The Lost Update Problem
  - two transactions that access the same DB items have their operations interleaved in a way that makes the value of some database item **incorrect**.

1. Lost Update

- John and Mary both try to withdraw money from their joint account at the same time . .

Both of these transactions are occurring at the same time!



1. Lost Update

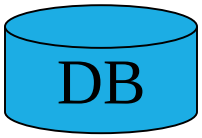
- When a user accesses a row of data from the DB, the actual data is read from the DB storage and held in a temporary buffer area in main memory

- Updating the DB involves modifying the in-memory copy and then writing it back to the DB storage on disk

1. Lost Update

What's happening behind the scenes!!

John reads a row memory

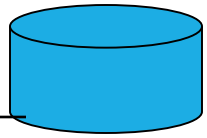


row copied to

John's process gets a copy of the data

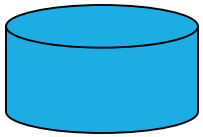
Mary's process gets a copy of the data

Row read



Mary reads the same row

John amends his copy of the row and writes it to the DB

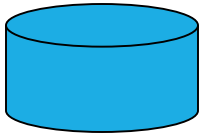


row written

Data updated in main memory

Data updated in main memory

Row written



Mary amends her copy of the row and writes to the DB, overwriting John's update

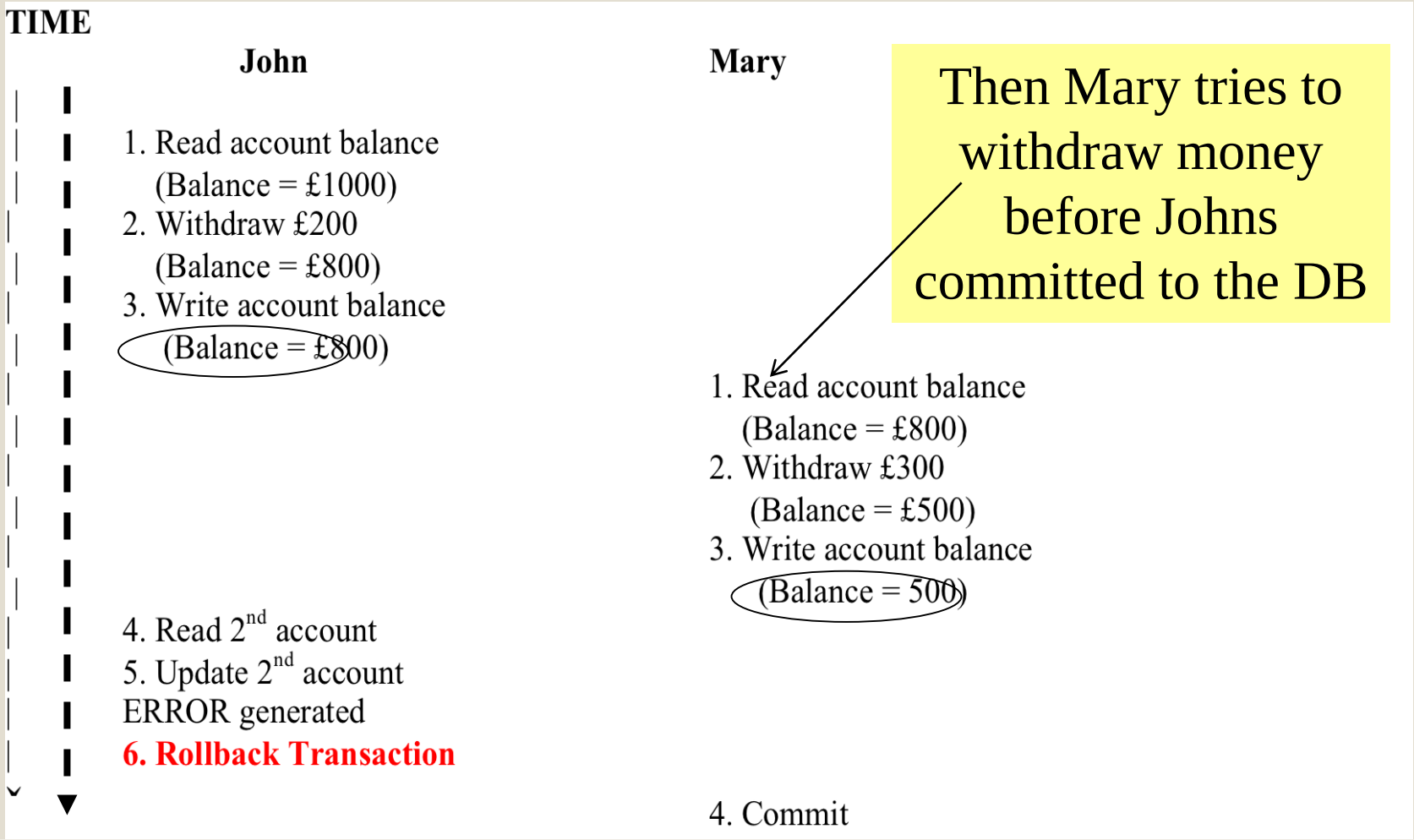
Temporary Update

- The Temporary Update Problem
  - one transaction updates a DB item and then the transaction **fails** for some reason.
  - The updated item is accessed by another transaction before it is changed back to its original value.

2. Temporary Update or Uncommitted Dependency

- The temporary update problem refers to one transaction viewing data that has been updated by another transaction, but will be rolled back by that other transaction at a later date, and so is not accurate data. It can occur as follows . . .

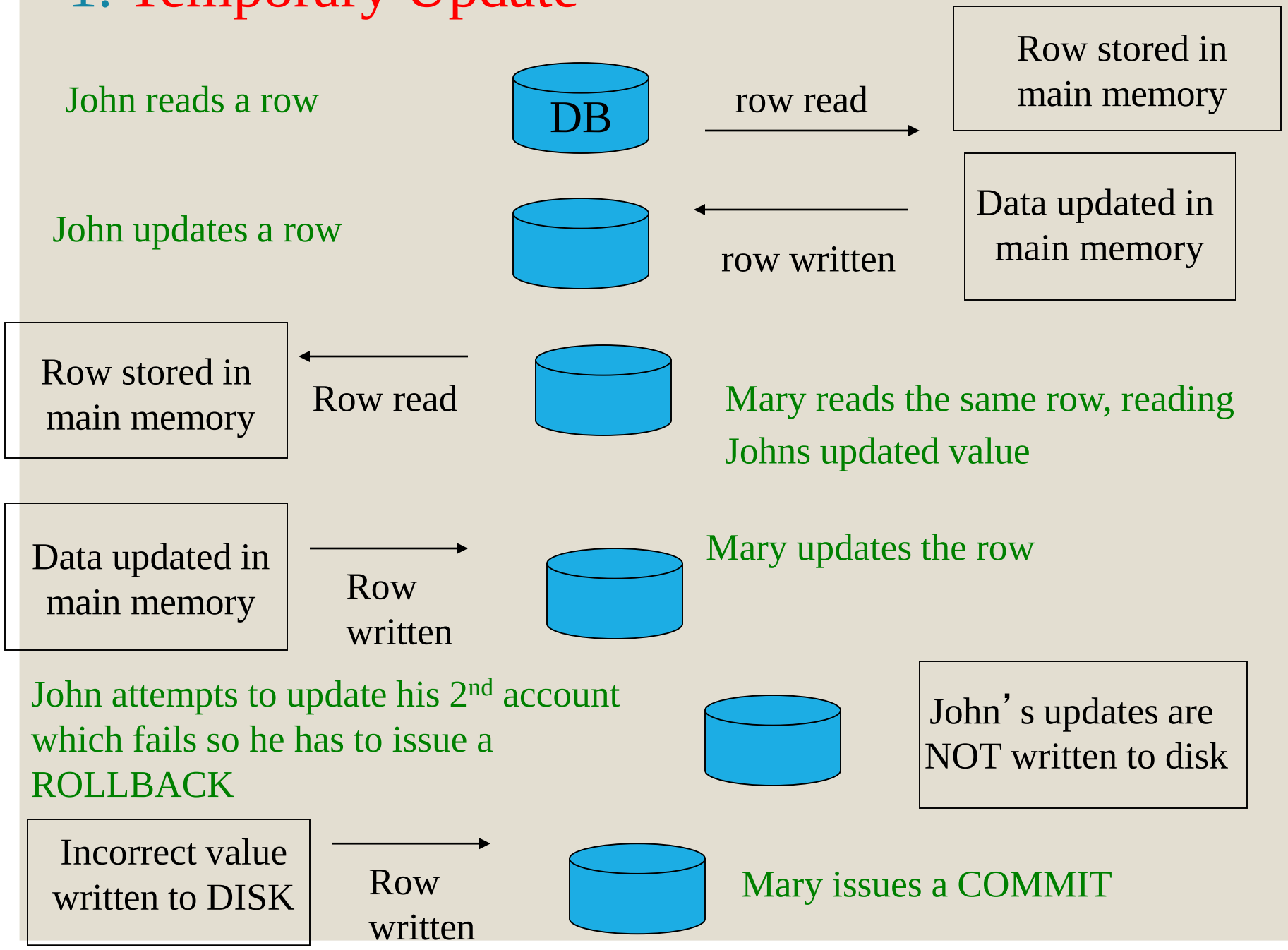
2. Temporary Update: Mary tries to withdraw money from their joint account at the same time as John tries to move money from the account to another account.



2. Temporary Update

- Mary's transaction read data that had been updated by a transaction that was not yet complete, i.e. it had not yet issued a COMMIT.
- John's transaction was later ROLLED BACK, meaning Marys transaction read an incorrect value from the database.

1. Temporary Update



Incorrect Summary

- The Incorrect Summary Problem:
  - one transaction is calculating an **aggregate** summary function on a *number of records* while other transactions are **updating** some of these records, the aggregate function may calculate some values before they are updated and others after they are updated.
  - Incorrect data

3. Incorrect Summary Problem

- This problem arises when an aggregate query is running over a number of rows in the database
  - e.g. sum(balance) or avg(balance)
- At the same time, a second query is updating the same values:
  - Add 50 to balance of account A.
  - Subtract 50 from balance of account b
- In such a scenario, the result of the aggregate query can be incorrect as illustrated next . . .

3. Incorrect Summary Problem

- It is caused by the incorrect interleaving of transaction operations.
- It demonstrates how erroneous results can be obtained by a transaction which is only reading the database.

- The following Example shows:
  - Trans A is reading 3 accounts and summing up the balances. Assume
  - ACC1=€100
  - ACC2=€200
  - ACC3 = €300

Sum(acc1+acc2+acc3)= €600

At the same time…….

3. Incorrect Summary Problem

- Trans B, over the same time interval, is transferring €50 from ACC2 and depositing it in ACC3.
- Trans A gets a result of 650 instead of 600
- DB has not been corrupted, but query result is wrong

Account Balances

Tran A	Tran B	Acct 1	Acct 2	Acct 3
read(acc 1)		100	200	300
sum = 100				
read (acc 2)				
sum = 200				
	read acc 2			
	balance = 200			
	subtract 50 from balance			
	balance = 150	100	150	300
	read acc 3			
	balance = 300			
	add 50 from balance			
	balance = 350	100	150	350
read (acc3)				350

Final sum = 650 but it should be 600

Consequences of no concurrency control

- Inconsistent data
- Undermining customers trust, for example:
  - Inaccurate account balances
  - Double booking a plane seat
  - Errors in customers invoices
  - Shipping the wrong quantity of goods

Solution to the problem . . .

- The objective of the DBMS is to:
  - Allow multiple transactions to run at the same time to improve efficiency (i.e. allow concurrency)
  - Make sure the effect of each transaction IS THE SAME AS if the transaction was running on its own.

- This is achieved using **LOCKING**

Locking

- Locking used by all commercial DB to implement concurrency
- Transactions lock parts of the DB data while it is being amended, preventing other transactions from amending/reading the same data
- When the amendment is complete, the lock is released enabling other transactions to continue

Lost Update Problem Revisited

- John reads DB row into memory buffer
- John's transaction locks the database row
- Mary tries to read the same row but is prevented by the lock – her transaction enters a 'wait' state
- John writes the updated row to disk and releases the lock
- Mary's transaction now proceeds and reads the updated row
- Mary's transaction locks the row to prevent access by other users
- Mary updates the row and writes to disk, releasing the lock

Lost Update Problem Solved

- The two updates are carried out successfully

- Downside: Mary's transaction is delayed slightly
  - Duration of locks could be significant, e.g. on-line clerks dealing with telephone orders

Temporary Update Revisited

1. John starts transaction and reads a row of a table after applying a lock
2. Mary tries to access the same row, but can't because its locked
3. John updates the row, writes to disk and releases lock
4. Mary now can read the row, locks it and updates it. Mary's update is based on data *after update by John.*
5. Mary now writes row to disk and releases the lock.
6. John's acquires a lock for the 2nd account, but the transaction fails and is rolled back.

- Problem – 1st lock is release too early
- Solution - Two-phase locking

Two-phase locking

Every transaction has 2 phases

- 1. **Growing phase -** locks acquired
- 2. **Shrinking phase -** locks released

Within a transaction, all locks must be acquired BEFORE the first lock is released.

- Returning to Step 3 on previous slide - lock would not be released as the transaction is not yet complete.
- Mary could not do steps 4 and 5
- When John's rollback occurs, Mary's update is not lost because it never took place!

Granularity of Locking

- **A lock can be applied at varying levels of 'severity' in terms of the amount of data that is locked**

- **Database Level**
- Lock the entire database
- Used only during a database backup or restore.
- Most severe

- **Table Level**
- Lock an entire table.
- Used where a particular transaction is to be applied to all of the rows in a table e.g. an increase of 10% is to be applied to all products in the Product table.

Granularity of Locking (cont.)

- **Block or Page Level**

A physical storage **block** or **page** in which a requested row is held is locked. Not used frequently..

- **Row Level**
  - Requested row is locked. **Most often used**.
  - All other records are available.
  - Where many records are involved in an update there may be an overhead incurred.

- **Field Level**
  - Only a single field or column is locked.
  - Useful where a single field gets updated often and locking the entire row would affect performance.

e.g. quantity-on-hand field in an inventory database.

  - Requires a considerable overhead.

BEWARE of Deadlock

Deadlock occurs when each of two transactions are waiting for the other to release the lock on an item

1. Anne locks AB123

2. Bill locks AB456

3. Anne requests AB456 and ……..
WAITS !…….

Deadlock

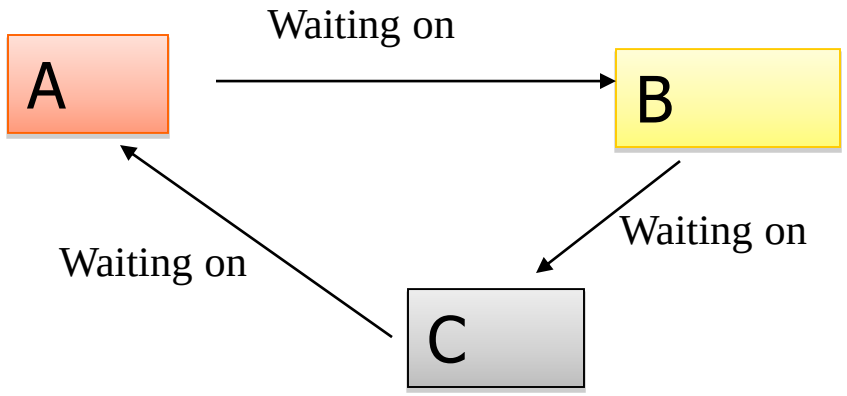
4. Bill requests AB123 and ……
WAITS!

- The Result is Circular Wait
- Both users are halted waiting for the other to proceed.

One user needs to rolls back their transaction

Solution to Deadlock – deadlock detection.

- The DBMS maintains a directed graph of all transactions waiting on locks, and what transactions they are waiting on.
- Deadlock is detected by a loop in the graph.
- The newest transaction is rolled back and restarted, allowing the other transactions to continue.



Exercise

- We have now covered:
  - COMMIT & ROLLBACK
  - LOCKING
- Using both of these, how many of the ACID properties can be guaranteed?
  - ATOMICITY?
  - CONSISTENCY?
  - ISOLATION?
  - DURABILITY?

RECOVERY

Recovery

Why recovery is needed (reasons for transaction failure):

1. **A computer failure (crash):** A hardware or software error occurs in the computer system during transaction execution. If the hardware crashes, the contents of the computer's memory may be lost.

Recovery

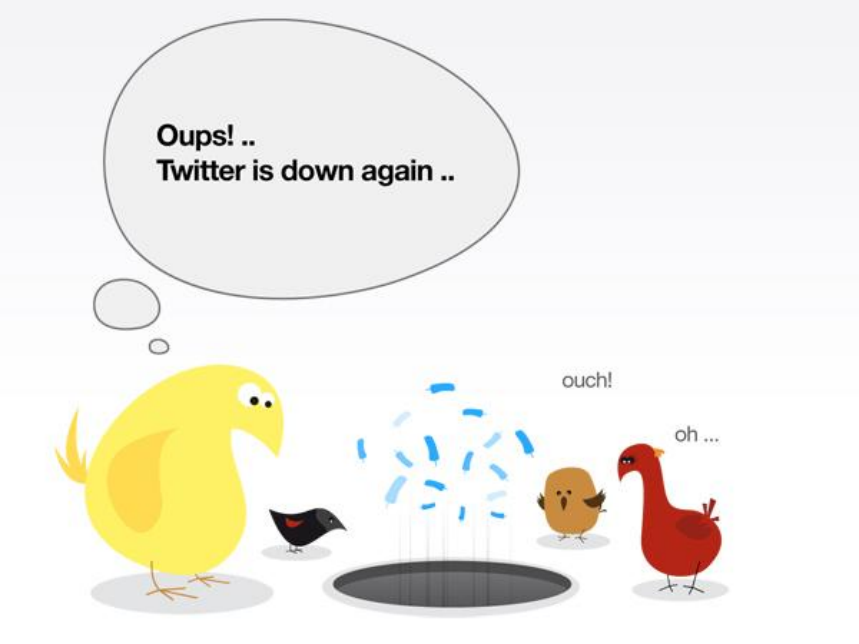
2. **Local errors** or exception conditions detected by the transaction:
   - certain conditions necessitate cancellation of the transaction. For example, data for the transaction may not be found. A condition, such as insufficient account balance in a banking database, may cause a transaction (e.g. withdrawal) to be canceled.
   - a programmed abort in the transaction

Transaction Processing - 11

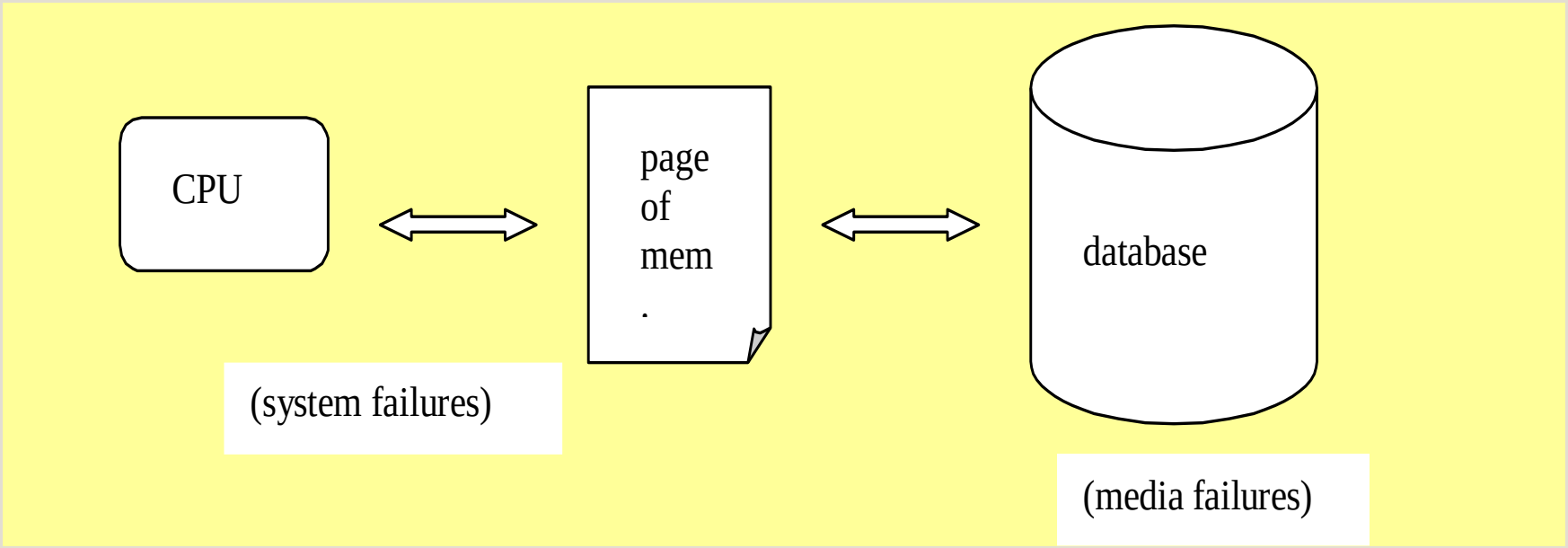
5. **Disk failure:** Some disk blocks may lose their data because of a read or write malfunction or because of a disk read/write head crash. This may happen during a read or a write operation of the transaction.
6. **Physical problems and catastrophes:** This refers to an endless list of problems that includes power or air-conditioning failure, fire, theft, sabotage, overwriting disks or tapes by mistake, and operator error.

Recovery

- Inevitably in any computer system things are going to go wrong.
  - The machine crashes and data is lost . . .
- The Recovery Manager in the DBMS handles recovery.



Reasons for failure fall into two categories



1. System Failures

- Affects all transactions currently in progress but does not physically damage the database.
- The critical point about them is that the contents of main memory (the database buffers) are lost.

Examples of System Failure (5)

1. Power failure,
2. Software failure,
3. Concurrency control enforcement,
4. An Exception detected by the Transaction
   e.g. A purchase order for a product not in stock
5. A System Error,
   This may be due to communications problems,
   hardware failure etc.

2. Media failures

- Results in physical damage to the database, and affects at least those transactions currently using that portion (e.g. disk head crash).
- System failures are much more common than media failures.
- The means of recovering to a consistent state is different for the different types of failures.

Recovering from Failure

- A DBMS has a number of tools to enable it to recover from system and media failure. These include:

1. Regular Backups
2. Transaction log
3. Checkpoint

What does the DBMS need to restore a database?

1. Regular Backups

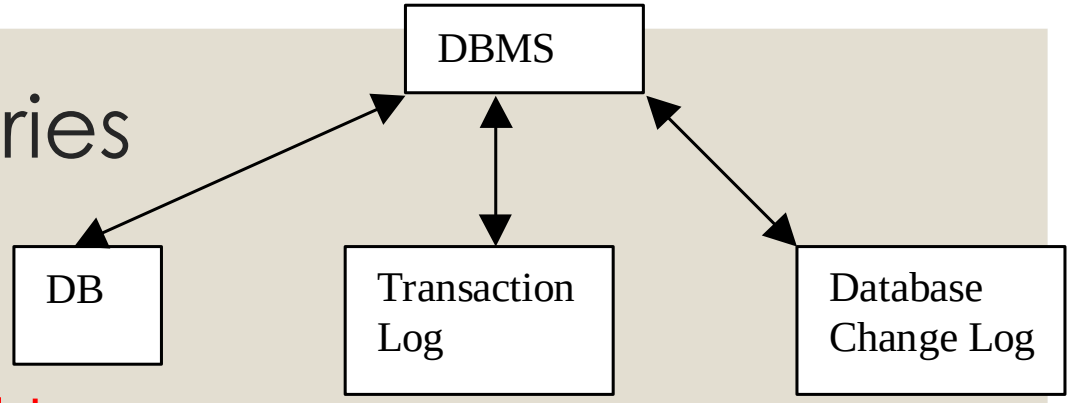
   The entire database is backed up regularly (e.g. once a day)

   - This means that in the event of media failure, corrupting some or all of the database, the database can be restored to it's previous correct status (at most 1 day old).
   - If there is a media failure a switch can be performed

What does the DBMS need to restore a database?

2. Transaction log
   - A transaction log contains the details of all transactions processed, and so provides an audit trail of all transactions and database changes.

Transaction log entries



- A transaction log contain 4 types of entries:
  1. [Start, T], which signals the start of transaction T.
  2. [Write, T, X, old-value, new-value], which signals that transaction T has updated the value of X from old-value to new-value.
  3. [Read, T, X] records the fact that T read the value of X.
  4. [Commit, T] records the successful completion of T.

- Sample from Transaction Log (using John & Mary examples from lecture)
  - [start, John]
  - [read, John, Balance]
  - [start, Mary]
  - [read, Mary, Balance]
  - [write, Mary, balance, 1000, 800]
  - . . . .

This log is kept on disk, and so is not effected by system failure.

Using a transaction log to recover from Media Failure

- Recap: With media failure, the database itself is lost, so an old copy of the database is restored from a backup

- The transaction log is then used to re-run transactions that have committed since the last backup was taken, bringing the data in the database up to date.

Using a transaction log to recover from System Failure

- If there is a system failure, the contents of main memory are lost.
- In memory, there will be two categories of transactions:
  - Transactions not yet complete and so need to be rolled back and started again.
  - Transactions that were complete, but their changes were not yet written to disk.
- Transaction Log will show which transactions had completed, and which were in progress.

Using a transaction log to recover from System Failure

- However, for committed transactions, how do you know if the details were still in the database buffers, or had already been written to disk?
- Ans: Checkpoints

Transaction Log

[start, 1]
[read, 1, balance_1]
[read, 1, balance_2]
[write, 1,balance_1, 10,20]
[write, 1, balance_2, 50,40]
[commit,1]
[checkpoint]

3. Check Points

A check point is a point in time at which the updates of committed transactions are written from **memory** to **disk**.

- Taking a checkpoint involves:
  - writing a checkpoint record to the transaction log, and
  - "force-writing" or flushing the contents of the database buffers to disk.
- Any transactions which issue a COMMIT AFTER the last checkpoint have not yet been written to DISK, and so their updates are only in memory buffers.

Returning to ACID properties:

- Atomicity:
  - Guaranteed by using COMMIT & ROLLBACK
- Consistency:
  - Guaranteed by combined used of LOCKING and COMMIT & ROLLBACK (and integrity constraints)
- Isolation:
  - Guaranteed by using LOCKING
- Durability:
  - Guaranteed if the database can Recover committed transactions from Media & System failure.

Revision Questions:

- What is a transaction?
- Describe the ACID properties?
- What are the 3 classical concurrency control problems?
- What is meant by the terms: lost update, uncommitted dependency problem, and the inconsistent analysis problem?
- Describe each of the problems briefly.
- What is meant by the term 'granularity of locking'?
- Provide 3 examples of different levels of granularity in locking.
- What is deadlock?
- How can deadlock be dealt with in a DBMS?
- What are the 2 phases in 2-phase locking?
- What is the difference between system failure and media failure.
- Give 5 examples of system failures.
- What is a Checkpoint?
- How does the Recovery Manager recover transactions when a system fails?
- Describe the entries in the Transaction Log?
- How does a database recover from media failure?