INTRODUCTION TO CONCURRENCY AND IT'S ANOMALIES

- EDDY MWENDA (SCT212-2108/2015)
- GEOFFREY NDUNGU (SCT212-2094/2015)
- CATHERINE NYAMBURA (SCT212-5255/2015)
- TIMOTHY MBAKA (SCT212-2103/2015)
- LORNA MACHARIA (SCT212-2102/2015)
- IMMANUEL MOHOL (SCT212-6248/2015)
- MALINDA KISSAKA (SCT212-5854/2015

DEFINITION

- The ability of different parts or units of a program to be executed out of order or in partial order, without affecting the final outcome: Allowing parallel execution of concurrent units.
- In relation to database, it's the ability of a database to allow multiple users to affect multiple transactions.
- It's one of the main properties that make databases stand out from other forms of storage, e.g. Spreadsheets. They do not offer several users the ability to view and work on the different data at the same time, because once the first user opens the file, it is locked to other users. Others can only read and not alter the data, i.e. edit

IMPORTANCE OF CONCURRENCY

- It allows execution of multiple tasks at the same time
- Offers accessibility to a database to multiple users at real time
- Shortens the work time frame as several tasks are achieved as it isn’t serial

How concurrency works

- How do we allow multiple transactions to be executed simultaneously?
  - Through Interleaving
  - Through parallelism

Through interleaving

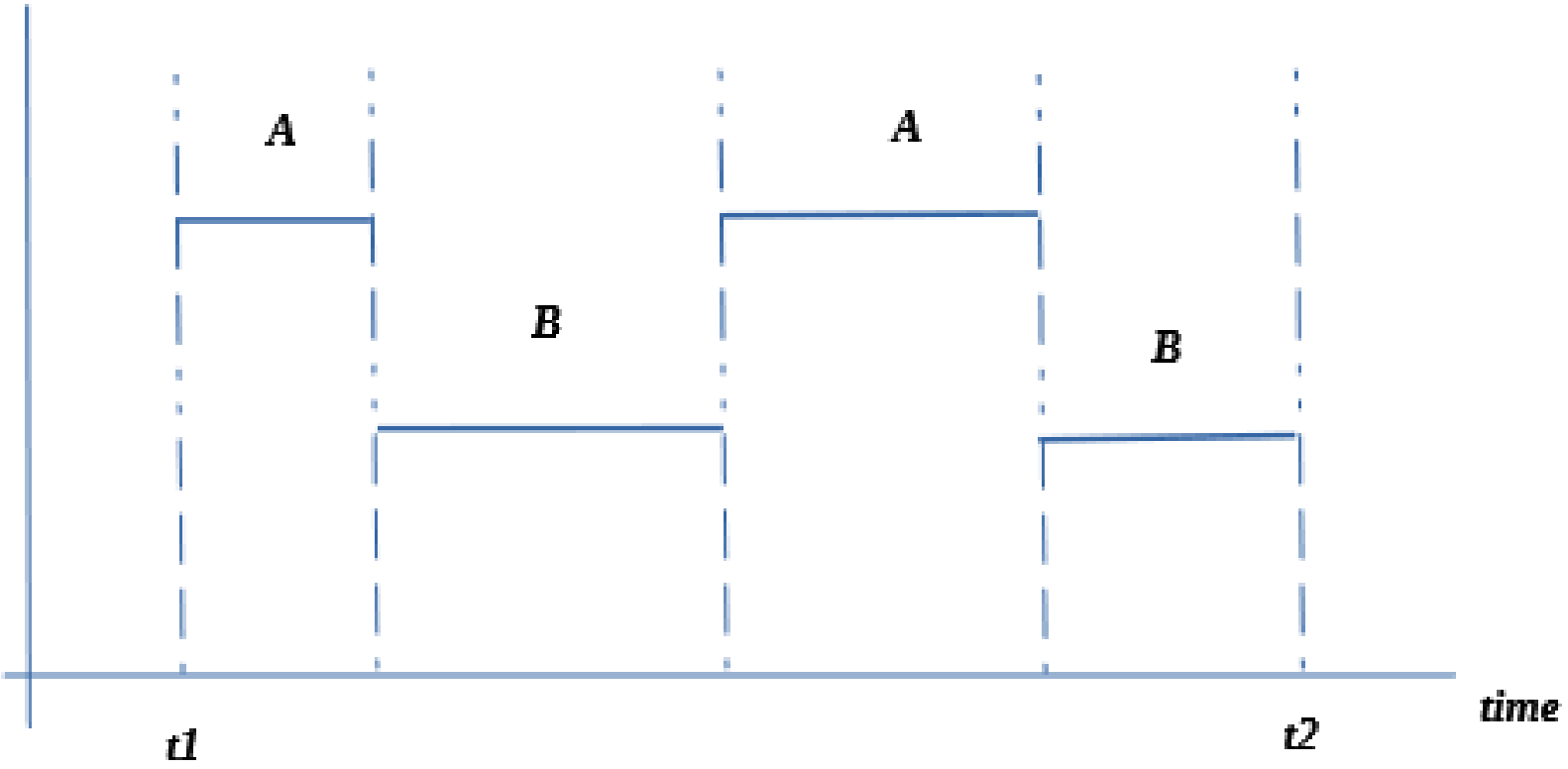
- Interleaving: To allocate things such as successive segments of memory to different tasks. Thereby making a system more efficient, fast and reliable since data is arranged in a non-contiguous manner.

- Concurrent systems execute some commands from one program (transaction), then suspends that program and executes some commands from the next program, and so on.

- A program is resumed at the point where it was suspended when it gets its turn to use the CPU again. This is known as interleaving.

Interleaving example

- Consider two transactions A and B executing concurrently in an interleaved manner.



Interleaving (explained)

- Transaction A starts, but does not run to completion, as it awaits for some Input or Output, Transaction B starts until at a time it also requires some resource, therefore it is suspended and Transaction A resumes from it has been suspended.

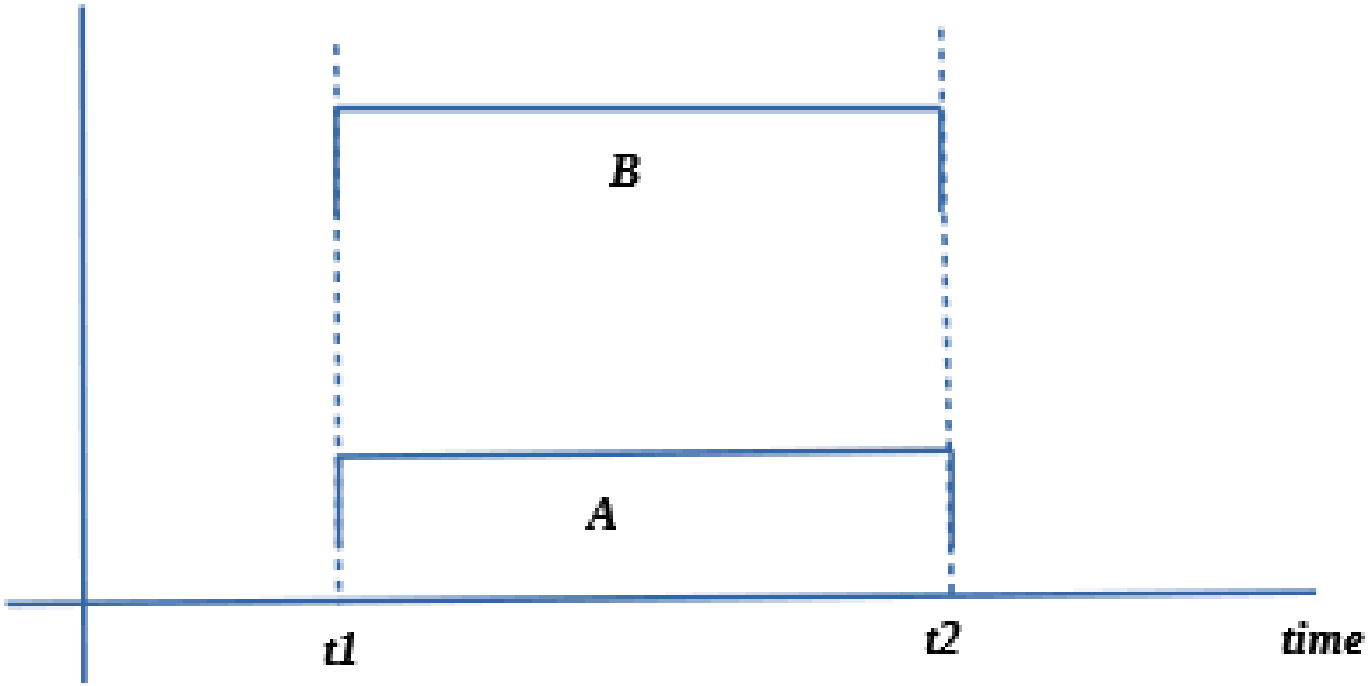
Through parallelism

- Parallelism: Is a technique that performs several computations at the same instance of time(parallel).

- Its is possible to achieve it in an environment where there is multiple processors.

Parallelism example

- Consider two transactions A and B executing in parallel



- Both transaction A and B run together at the same instance of time i.e. t1-t

Concurrency Anomalies

- These are occurrences that violate the constraints and integrity of the database.

- Some concurrency anomalies:
  - Dirty read / Uncommitted dependency
  - Lost update
  - Non-repeatable read
  - Reading inconsistent states
  - Phantom read

Dirty read / Uncommitted dependency

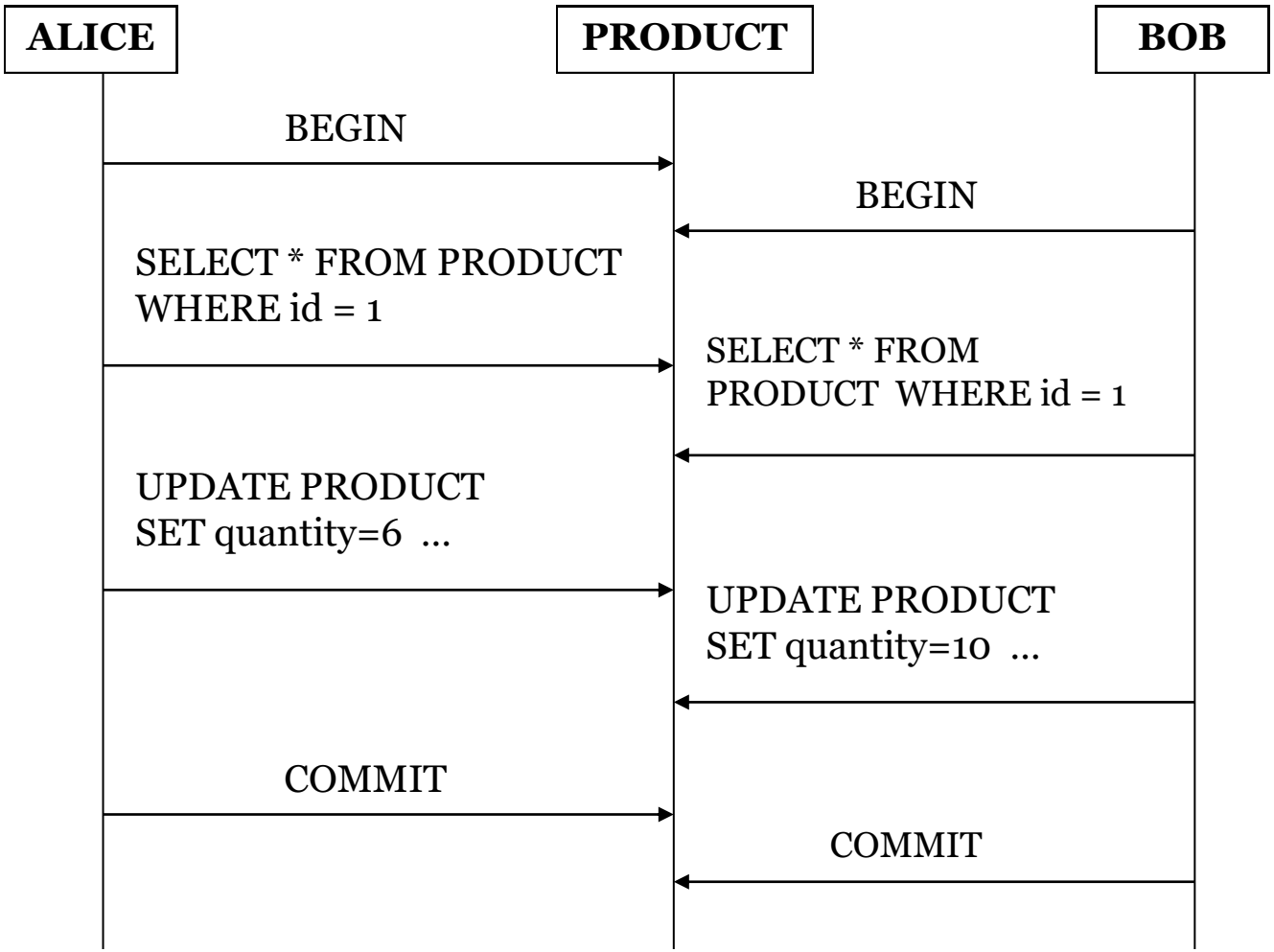
- It occurs when a transaction is allowed to read data from a row that has been modified by another running transaction but not yet committed

- Example:
  We have two transactions X AND Y , X writes any variable value in the Database and still not committed. Same time Y reads the value from X before committing, this is called Dirty Read since X may Rollback and Yet Y will use the value BEFORE THE ROLLBACK

Lost Update

- Occurs when transactions try to concurrently update the same resource without being aware of earlier updates.
- Transaction A and B both read resource X and later after A writes to X, B writes X based on the initial read value of X.
- A’s update is thus lost.

Lost Update (Example)



Final quantity = 10

(ALICE's update of 6 was lost)

Phantom read

- This type of concurrency anomaly occurs when a transaction defines a subset of data items that the transaction wants to work with i.e. by performing a query and obtaining a query result.

- At this point, it is possible that data items are concurrently changed by another transaction so that they no longer qualify for inclusion in the query result or vice versa. The same applies to objects that are inserted or deleted.

Phantom read

- When we look at an example of a situation where objects are inserted or deleted, a phantom read occurs when, in the course of a transaction, two identical queries are executed, and the collection of rows returned by the second query is different from the first.

- This happens because in between the two queries, another transaction takes place that either inserts or deletes a row(s) thus leading to a phantom read.

Phantom read example

- Transaction A begins
  - SELECT * FROM EMPLOYEE WHERE SALARY > 10000;
- Transaction B begins
  - INSERT INTO EMPLOYEE 9EMP_ID, FIRST_NAME, DEPT_ID, SALARY) VALUES (‘111’, ‘Jamie’, 10, 35000);

✓ Transaction B inserts a row that would satisfy the query in transaction A if it were issued again and this leads to a phantom read anomaly.

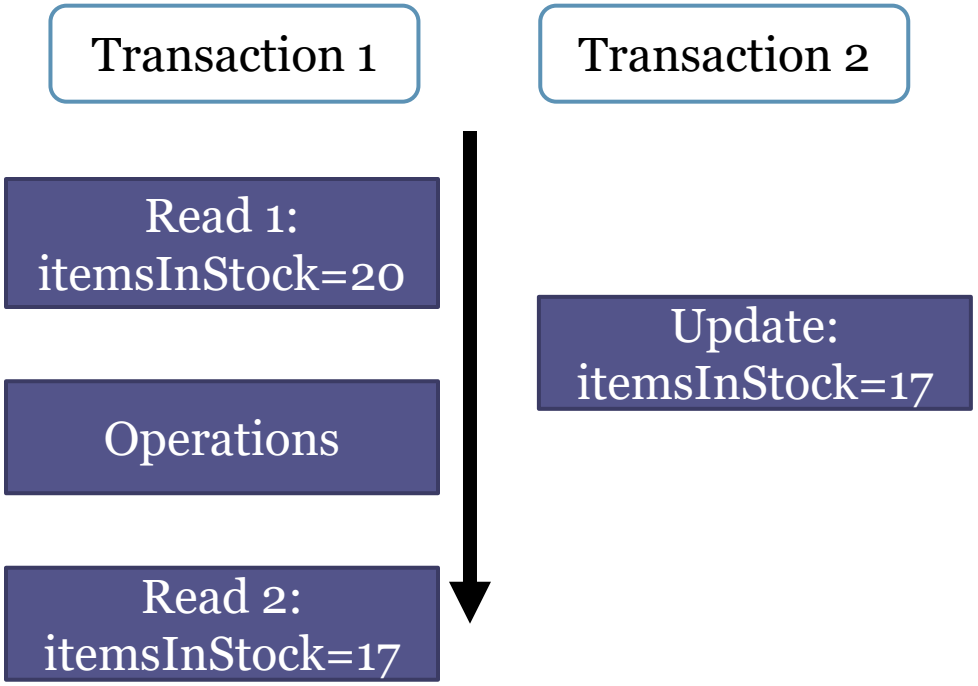
NON-REPEATABLE READ

- This Concurrency anomaly happens when one transaction reads the same data twice, and in between those reads, another transaction updates the data.

- So at the second read, a different value is returned.

NON-REPEATABLE READ

ID	Name	itemsInStock
1	Chapati	20



READ INCONSISTENT STATE

- When a transaction reads an object X twice and X has different values, the problem is called *inconsistent read*.
- This involves scenario whereby a transaction 2 reads the value X before transaction 1 has changed it hence yielding inconsistent results.
- There are two types of inconsistent reads:
  - Ghost update (a) if two transactions access concurrently to the same object and they view their modification each other; notice that all objects are already existent in the database.
  - Ghost update (b) if one of two transaction insert a new object into the database and another transaction access use that data

- The following is a scenario

Transaction 1	**Transaction 2**
Scope1.Transaction.Begin();	Scope2.Transaction.Begin();
Customer c2=GetCustomer(2);	
Scope1.Refresh(c2);	Customer c2=GetCustomer(2);
c2.DsicountRate is 1.0	c2.DiscountRate=1.2;
	Customer c4= GetCustomer(4);
	c4.DiscountRate=1.2;
	Scope2.Transaction.Commit();
Customer c4=GetCustomer (4);	
Scope1.Refresh(c4);	
c4.DiscountRate is 1.2	
Scope1.Transaction.Commit();	

- Transaction1 shows an inconsistent states in which the discount rates are different.
- Based on this inconsistent state, Transaction1 could make incorrect calculations.

- It is mostly seen to be the same as non-repeatable read with the only difference being that this concurrency anomaly violates certain or constraints. E.g. In the above example, it is a requirement that the customers of interest have the same discount rate

QUESTIONS?