Transactions

Transaction Properties:

A *transaction* is a unit of program execution that accesses and possibly updates various data items. *Transaction Processing Systems* are systems with large databases and hundreds of concurrent users that are executing database transactions. A transaction usually results from the execution of a user program written in a high level data manipulation language or programming language and is delimited by statements (or function calls) of the form begin transaction and end transaction. To ensure the integrity of the data, we require that the database system maintains the following properties of the transactions. These properties generally called as **ACID rules**.

***Atomicity**: A transaction is an **atomic** unit of processing; it is performed in its entirety or not performed at all.* Either all or non-of the operations of transaction will seen in the database. **(ALL OR NOTHING).** Requires all operations of a transaction must be completed; if not, the transaction is aborted. In other words, a transaction is treated as a single, indivisible logical unit of work.

***Consistency Preservation:** A transaction is **consistency preserving** if its complete execution take the database from one consistent state to another.* At any time, everywhere, to all of the transactions or to all of the users, the answers (result of specific query) must be same.

***Isolation:** A transaction should appear as though it is being executed in isolation from other transactions. That is the execution of a transaction should not be interfered with by any other transactions executing concurrently.* Transactions should never show its results to any one until it performs commit. Means that the data used during the execution of a transaction cannot be used by a second transaction until the first one is completed.

***Durability or permanency:** The changes applied to the database by a committed transaction must persist in the database. These changes must not be lost because of any failure.* Effects of committed transactions must seen in DB, aborted ones must not. (crash recovery component) When a transaction is completed, the database reaches a consistent state, and the state can not be lost, even in the event of the system's failure. Indicates the permanence of the database's consistent state.

Transaction states:

In the absence of failures, all transactions complete successfully (committed). However, transaction may not complete its execution successfully (aborted). If we are to ensure the atomicity property, an aborted transaction must have no effect on the state of the database. Thus, any changes that the aborted transaction made to the database should be undone (rolled back). Once a transaction has committed we can not undo its effects by aborting it. The effects of committed transactions should persist even if there is a system failure. A transaction must be in one of the following states:

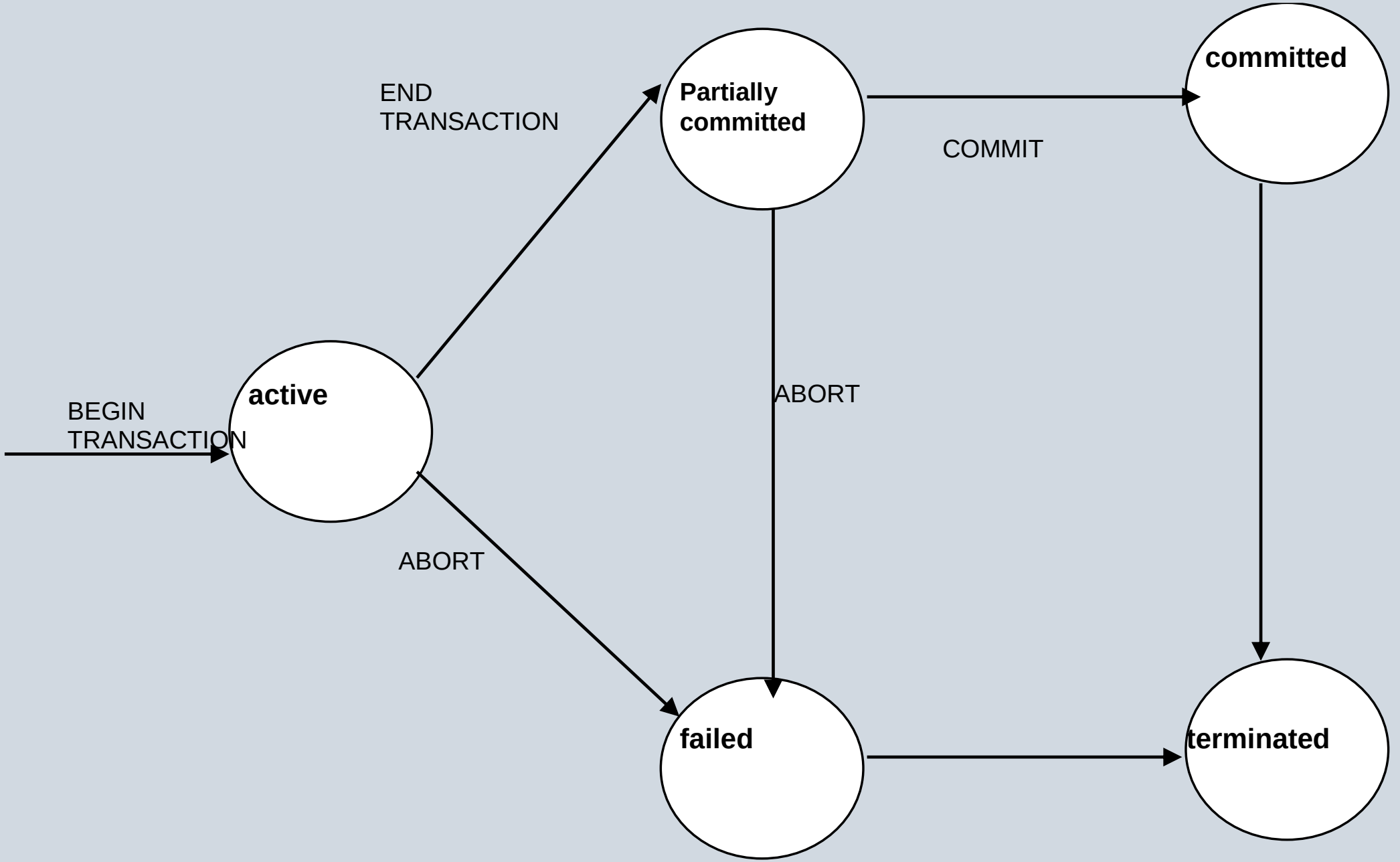
Active, the initial state: The transaction stays in this state while it is executing (Begin transaction)

Partially committed, after the final statement has been executed

Failed, after the discovery that normal execution can no longer proceed

Aborted, after the transaction has been rolled back and the database has been restored to its state prior to the start of the transaction.(ROLLBACK)

Committed, after successful completion (COMMIT)


END
TRANSACTION
Partially
committed
committed
COMMIT
active
BEGIN
TRANSACTION
ABORT
ABORT
failed
terminated

WHY CONCURRENCY CONTROL IS NEEDED

Concurrency control is the procedure in DBMS for managing simultaneous operations without conflicting with each another. Concurrent access is quite easy if all users are just reading data. There is no way they can interfere with one another. Though for any practical database, would have a mix of reading and WRITE operations and hence the concurrency is a challenge. *Concurrency control* is used to address such conflicts which mostly occur with a multi-user system. It helps you to make sure that database transactions are performed concurrently without violating the data integrity of respective databases. (https://www.guru99.com/dbms-concurrency-control.html)

Several problems can occur while transactions are working concurrently. In order to solve and prevent these problems concurrency control mechanisms are used. To explain some of these problems assume that we have 2 transaction with the following operations:

T1:	read item(X)	--- r1(x)	T2:	read item (X)	--- r2(x)
	X:=X-N	--- op1(x)		X:=X+M	--- op2(x)
	write item (X)	--- w1(x)		write item (X)	--- w2(x)
	read item (Y)	--- r1(y)			
	Y:=Y+N	--- op1(y)			
	write item (Y)	--- w1(y)			

a. The lost update problem:

T1	T2	
r1(x) op1(x)		
	r2(x) op2(x)	
w1(x) r1(y)		
	w2(x)	Item X has an incorrect value because its update by T1 is LOST
op1(y) w1(y)		

b. The temporary update problem: (uncommitted data)(uncommitted dependency- dirty read)

T1	T2
r1(x) op1(x) w1(x)	
	r2(x) op2(x) w2(x)
r1(y) transaction fails	

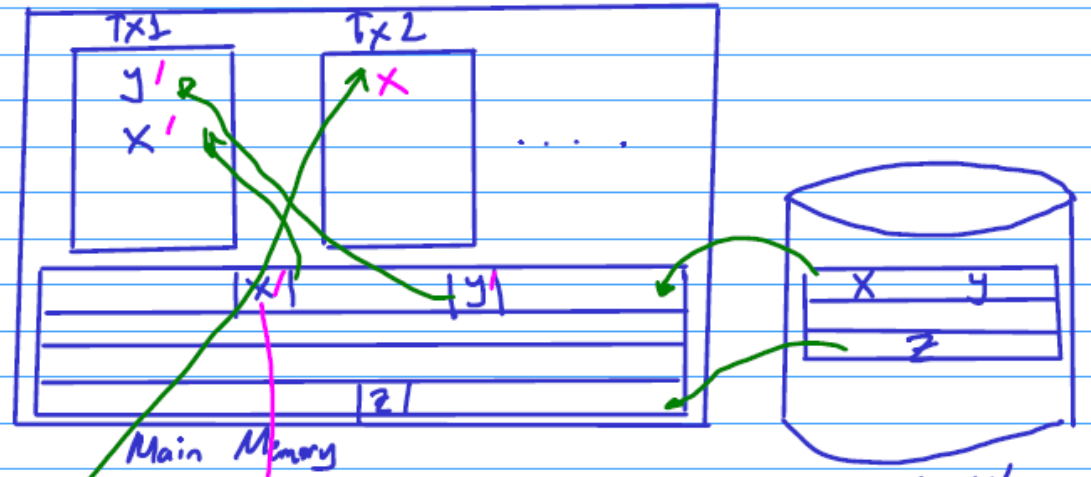
Since transaction T1 fails, must change the value of X back to its old value; but meanwhile, T2 has read the temporary incorrect value of X.

c. The incorrect summary problem: (inconsistent retrievals)(inconsistent analysis problem)

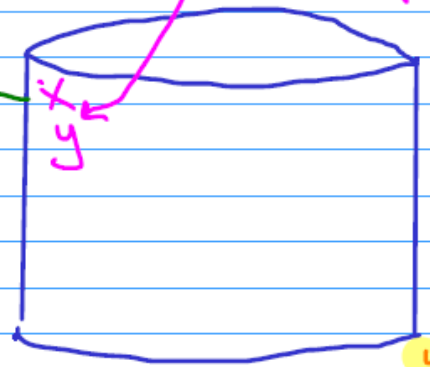
T1	T3
	Sum:=0 r3(A) sum := sum+A
r1(x) op1(x) w1(x)	
	r3(x) sum:=sum+X r3(y) Sum:=sum+Y
r1(y) op1(y) w1(y)	

T3 reads X after N is subtracted and reads Y before N is added, so a wrong summary is the result.

CONSISTENT READ → undo table space used!



consistent read
old version kept here



Tx1: read x
write x
read y
write y

Tx2: read x consistent read
write y Waits

UNDO TABLE SPACE
used for 1. roll back 2. consistent read

Tx1	Tx2	
reader	reader	✓
reader	writer	✓
writer	reader	✓ consistent read (old version read)
writer	writer	✗ Tx2 waits until Tx1 commit or rollback