Transactions and Concurrency

In production DBs users perform transactions concurrently.

- DBMS wants to maximize throughput
- Without compromising integrity

Example:

Person tries to remove, at the same time $100 from bank account.

T_1	T_2
Read balance	Read balance
If balance > 100	If balance > 100
subtract 100 from account	subtract 100 from account

time

Can we have reach a state where person gets $200 but bank only records $100 given?

If so, we have lost consistency of data.

Properties of Transactions:

ACID

- Atomicity: A transaction happens in its entirety or not all all

Incomplete transactions must be undone.

- Isolation: A transaction must appear to be executed as if no other transaction is executing at the same time.

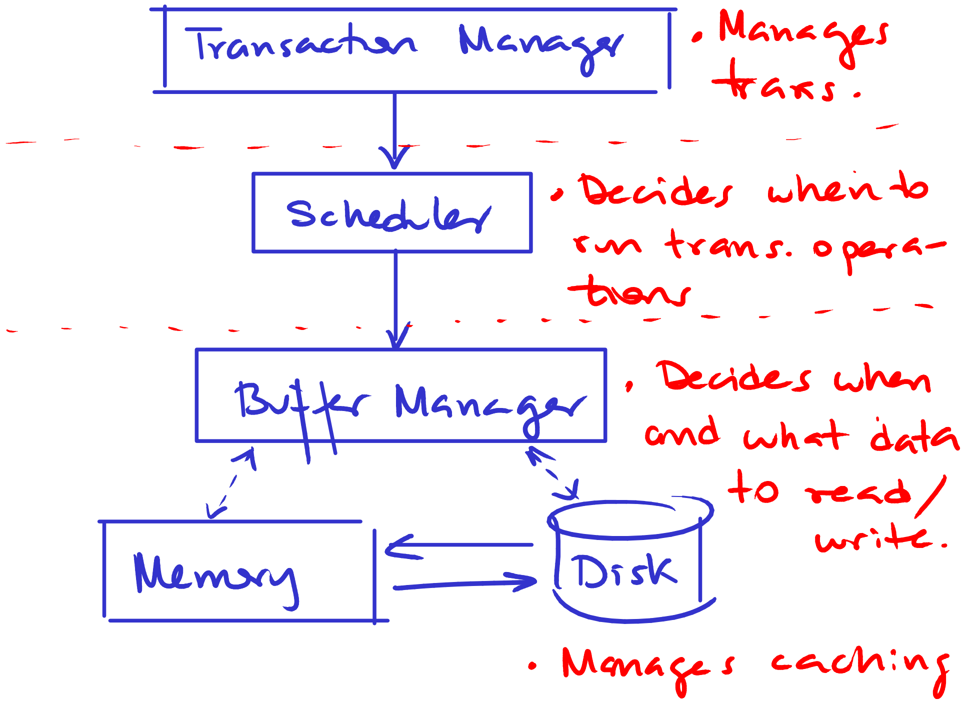
Transactions cannot communicate with each other.

- Durability: The effect on the db of a transaction has succesfully completed must never be lost.
- Even in the event of failures.

Responsability of Programmer.

- Consistency: Transactions are given a DB in a consistent state and are expected to keep it consistent.

$\boxed{\text{The role of the DBMS is to maximize number of concurrent trans. while maintaing ACID.}}$



To maximize throughput, the scheduler might:

- delay transaction operations.
- reorder transactions

To guarantee ACID, the scheduler:

- must make sure transactions are durable
- avoid undesirable interleaving of trans.
- deal with deadlocks

Transactions

Any transaction either
- completes (commits)
or
- aborts (rollback)

Atomicity.

If system crashes: (server or client):
- Non completed transactions must be undone (rollback) Durability.

Correctness Principle

Any transaction, if executed in isolation will transform any consistent state of the DB into another consistent state.

The DBMS <u>must</u> guarantee isolation even when many trans. are executed concurrently

A transaction is a list of actions.
For simplicity sake we will only consider read/write of DB objects.

Notation.

Read (A, v) Reads DB object A into local variable v (local to transaction)

Write (A, v)   Replaces DB object A with value in v.

Ex: T is a transaction that moves \$100 from account A to account B:

T
Read (A, v)
v -= 100
Write (A, v)
Read (B, v)
v += 100
Write (B, v)

schedule of a transaction.

time, implies order of operations.

There might be many copies of the same transaction running.

Ex: Two instances of T are trying to run simultaneously.

Assumption:

Reads and writes are atomic and cannot be interleaved.

Schedule

Sequence of actions taken by one or more transactions.

When two transactions want to be executed 3 options:

1) T_1 executes first, then T_2
denoted:

$T_1 ; T_2$ } Serial schedules.

2) $T_2 ; T_1$

3) The operations of T_1 and T_2 interleave.

Many, many possible interleavings of operations of T_1 and T_2
- Some safe
- Some unsafe (break consistency)

Serial Schedule

A schedule is serial if its actions consists of all the actions of one trans. followed by all the actions of another transaction and so on.

Ex.

T_1	T_2
Read (A, t)	
t += 100	
Write (A, t)	
Commit	
	Read (A, s)
	s *= 1.1
	Write (A, s)
	Commit

$T_1 ; T_2$

T_1	T_2
	Read (A, s)
	s *= 1.1
	Write (A, s)
	Commit
Read (A, t)	
t += 100	
Write (A, t)	
Commit	

$T_2 ; T_1$

Each schedule might have a different impact on DB.

Say A_0 value of A before schedule.

$T_1 ; T_2 \Rightarrow A = 1.1 (A_0 + 100)$

$T_2 ; T_1 \Rightarrow A = 100 + 1.1 A_0$

Serializable Schedule.

A schedule S is serializable if there exists a serial schedule S' of the same transactions such that for every initial state of the DB, the effect of S and S' is the same.

T_1	T_2
Read (A, t)	
	Read (A, s)
t+:=100	
Write (A, t)	
	s *= 1.1
	Write (A, s)
	Commit
Commit;	

Effect of schedule:

$A = 1.1\,A \neq$ effect of $T_1; T_2$ or $T_2; T_1$.

$\Rightarrow$ non-serializable.

Another schedule:

T_1	T_2
Read(A, t)	
t += 100	
Write(A, t)	
	Read(A, s)
	s *= 1.1
	Write(A, s)
	Commit
Commit;	

Serializable:

Equivalent to $T_1; T_2$

To model transactions we only care about Read, Write, Commit, Rollback.

We can rewrite the schedule above as:

$R_1(A), W_1(A), R_2(A), W_2(A), C_2, C_1$

Use A_i for rollback (abort).

The job of the DBMS is to only allow serializable schedules.