

ICS 321 Spring 2013

# Transactions Processing (i)

Asst. Prof. Lipyeow Lim  
Information & Computer Science Department  
University of Hawaii at Manoa

# Airline Reservation Example

Flights ( fltNo , fltDate , seatNo , seatStatus )

To view available seats:

```
SELECT seatNo
FROM Flights
WHERE fltNo = 123 AND fltDate = DATE '2008-12-25'
      AND seatStatus = ' available ' ;
```

To reserve a particular seat:

```
UPDATE Flights
SET seatStatus = 'occupied'
WHERE fltNo = 123 AND fltDate = DATE '2008-12-25 '
      AND seatNo = '22A';
```

# Transactions

- A transaction is the DBMS's abstract view of a user program: a sequence of reads and writes.
  - Eg. User 1 views available seats and reserves seat 22A.
- A DBMS supports multiple users, ie, multiple transactions may be running concurrently.
  - Eg. User 2 views available seats and reserves seat 22A.
  - Eg. User 3 views available seats and reserves seat 23D.

# Concurrent Execution

- DBMS tries to execute transactions concurrently – why ?

Schedule 1		Schedule 2		Schedule 3	
U1	U2	U1	U2	U1	U2
Finds 22A empty		Finds 22A empty			Finds 22A empty
	Finds 22A empty	Reserves 22A			Reserves 22A
Reserves 22A			Finds 22A taken	Finds 22A taken	
	Reserves 22A		Does not reserve 22A	Does not reserve 22A	

# ACID Properties

4 important properties of transactions

- **Atomicity:** all or nothing
  - Users regard execution of a transaction as atomic
  - No worries about incomplete transactions
- **Consistency:** a transaction must leave the database in a good state
  - Semantics of consistency is application dependent
  - The user assumes responsibility
- **Isolation:** a transaction is isolated from the effects of other concurrent transaction
- **Durability:** Effects of completed transactions persists even if system crashes before all changes are written out to disk

# Atomicity

- A transaction might *commit* after completing all its actions, or it could *abort* (or be aborted by the DBMS) after executing some actions.
- A very important property guaranteed by the DBMS for all transactions is that they are *atomic*. That is, a user can think of a Xact as always executing all its actions in one step, or not executing any actions at all.
  - DBMS *logs* all actions so that it can *undo* the actions of aborted transactions.

# Example (Atomicity)

T1:	BEGIN
	A=A+100
	B=B-100
	END

T2:	BEGIN
	A=1.06*A
	B=1.06*B
	END

- The first transaction is transferring \$100 from B's account to A's account.
- The second is crediting both accounts with a 6% interest payment
- There is no guarantee that T1 will execute before T2 or vice-versa, if both are submitted together. However, the net effect must be equivalent to these two transactions running serially in some order.

# Database View of Transactions

```
T1:  BEGIN
      A=A+100
      B=B-100
      END
```

```
T1:  BEGIN

      Read A from disk
      A=A+100
      Write A to disk

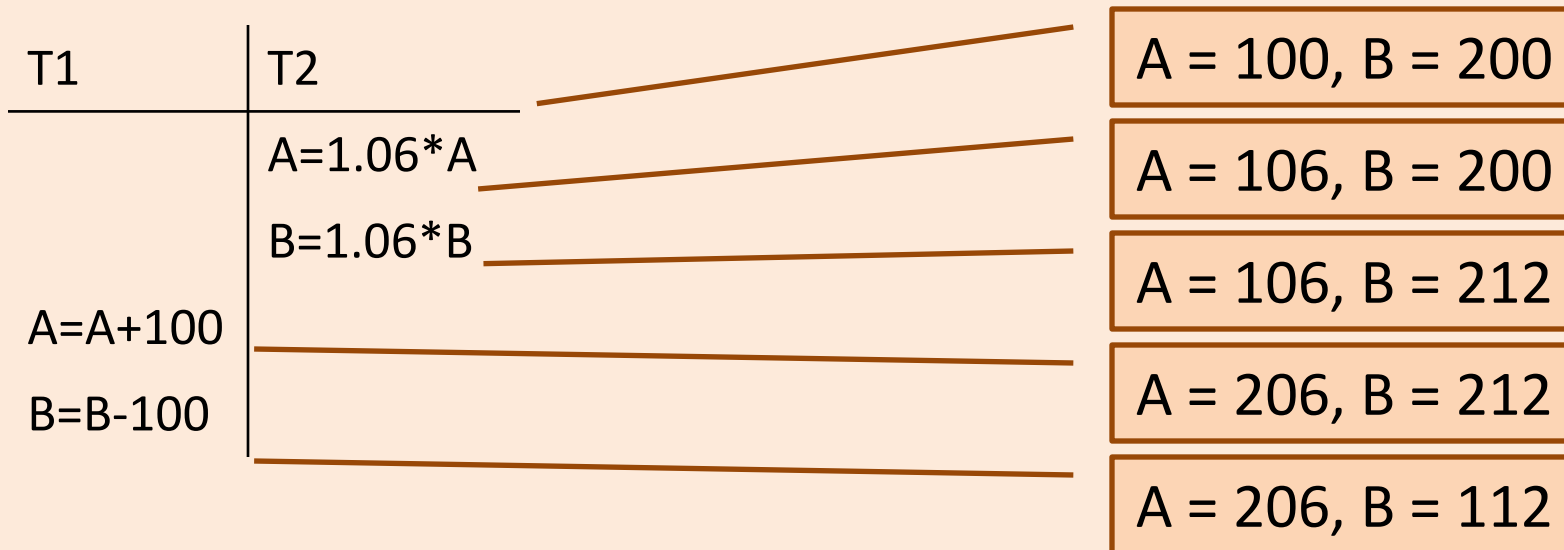
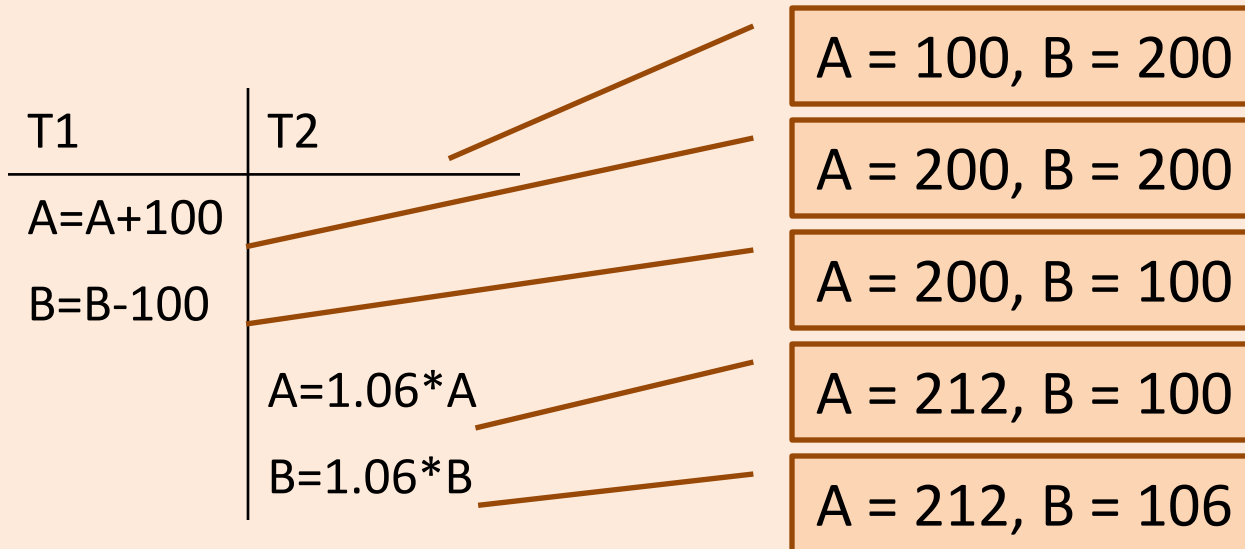
      Read B from disk
      B=B-100
      Write B to disk

      END
```

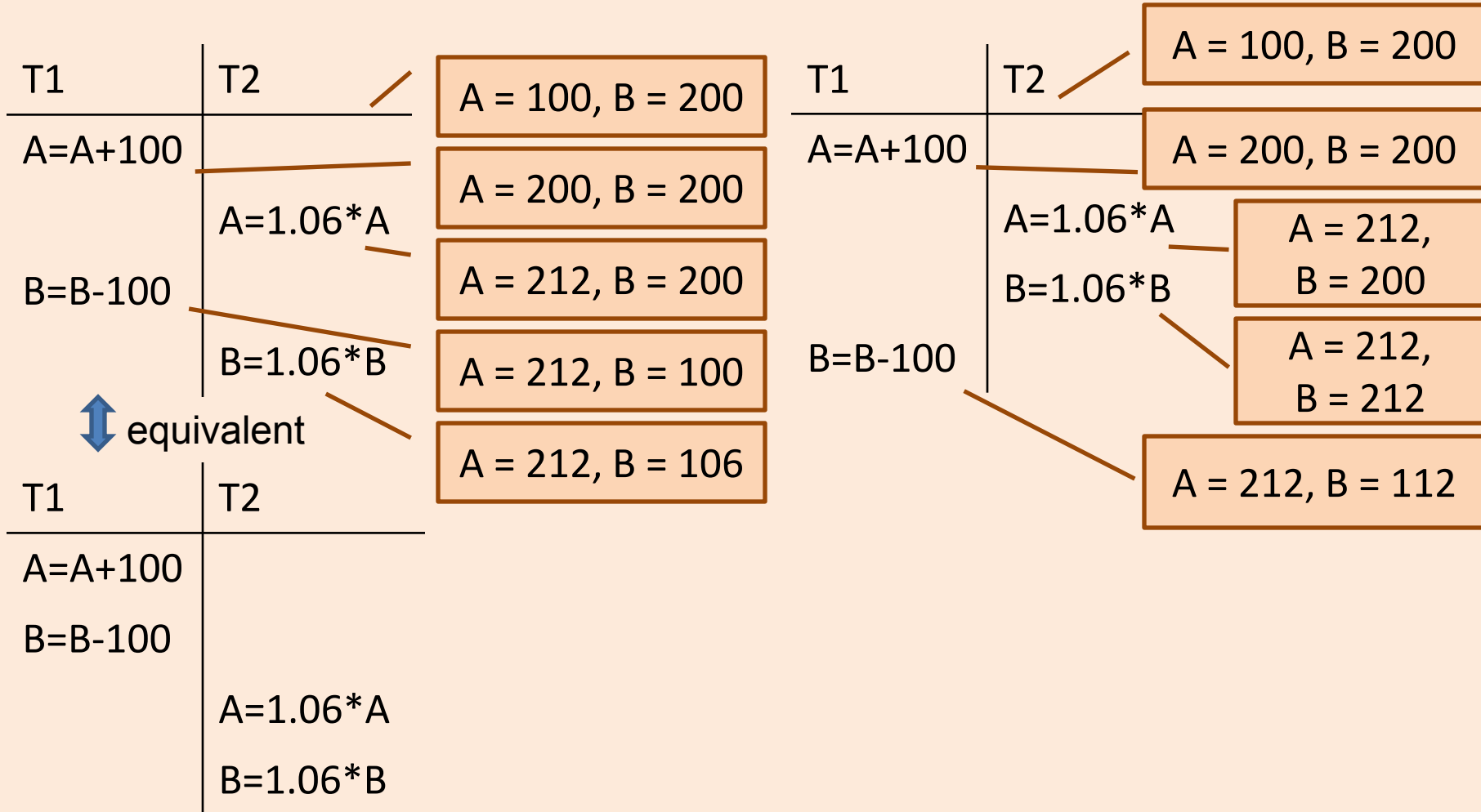
```
T1:  BEGIN
      R(A)
      W(A)
      R(B)
      W(B)
      END
```



# Serial Executions



# Example (Serializability)



# Scheduling Transactions

- *Serial schedule*: Schedule that does not interleave the actions of different transactions.
- *Equivalent schedules*: For any database state, the effect (on the set of objects in the database) of executing the first schedule is identical to the effect of executing the second schedule.
- *Serializable schedule*: A schedule that is equivalent to some serial execution of the transactions.

(Note: If each transaction preserves consistency, every serializable schedule preserves consistency.)

# Transactions in SQL

- After connection to a database, a transaction is automatically started
  - Different connections -> different transactions
- Within a connection, a transaction is ended by
  - **COMMIT** or **COMMIT WORK**
  - **ROLLBACK** (= “abort”)
- DBMS can also initiate rollback and return an error.
- **SAVEPOINT** <savepoint name>
- **ROLLBACK TO SAVEPOINT** <savepoint name>
  - Locks obtained after savepoint can be released after rollback to that savepoint
- Using savepoints vs sequence of transactions
  - Transaction rollback is to last transaction only

# Isolation levels in SQL

- SQL supports 4 isolation levels

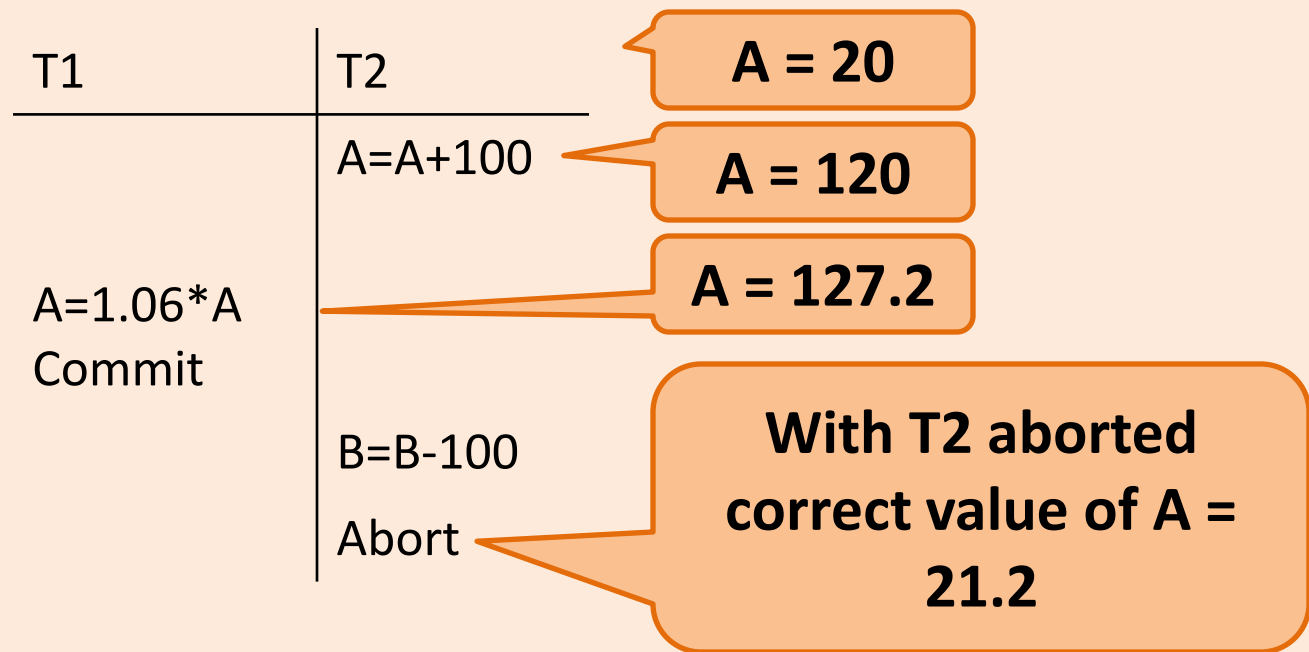
SQL Isolation Levels	DB2 Isolation Levels	Dirty read	Unrepeatable Read	Phantom
READ UNCOMMITTED	UNCOMMITTED READ (UR)	Maybe	Maybe	Maybe
READ COMMITTED	CURSOR STABILITY * (CS)	No	Maybe	Maybe
REPEATABLE READ	READ STABILITY (RS)	No	No	Maybe
SERIALIZABLE	REPEATABLE READ (RR)	No	No	No

**SET TRANSACTION ISOLATION LEVEL SERIALIZABLE**

**SELECT \*  
FROM Reserves  
WHERE SID=100  
WITH UR**

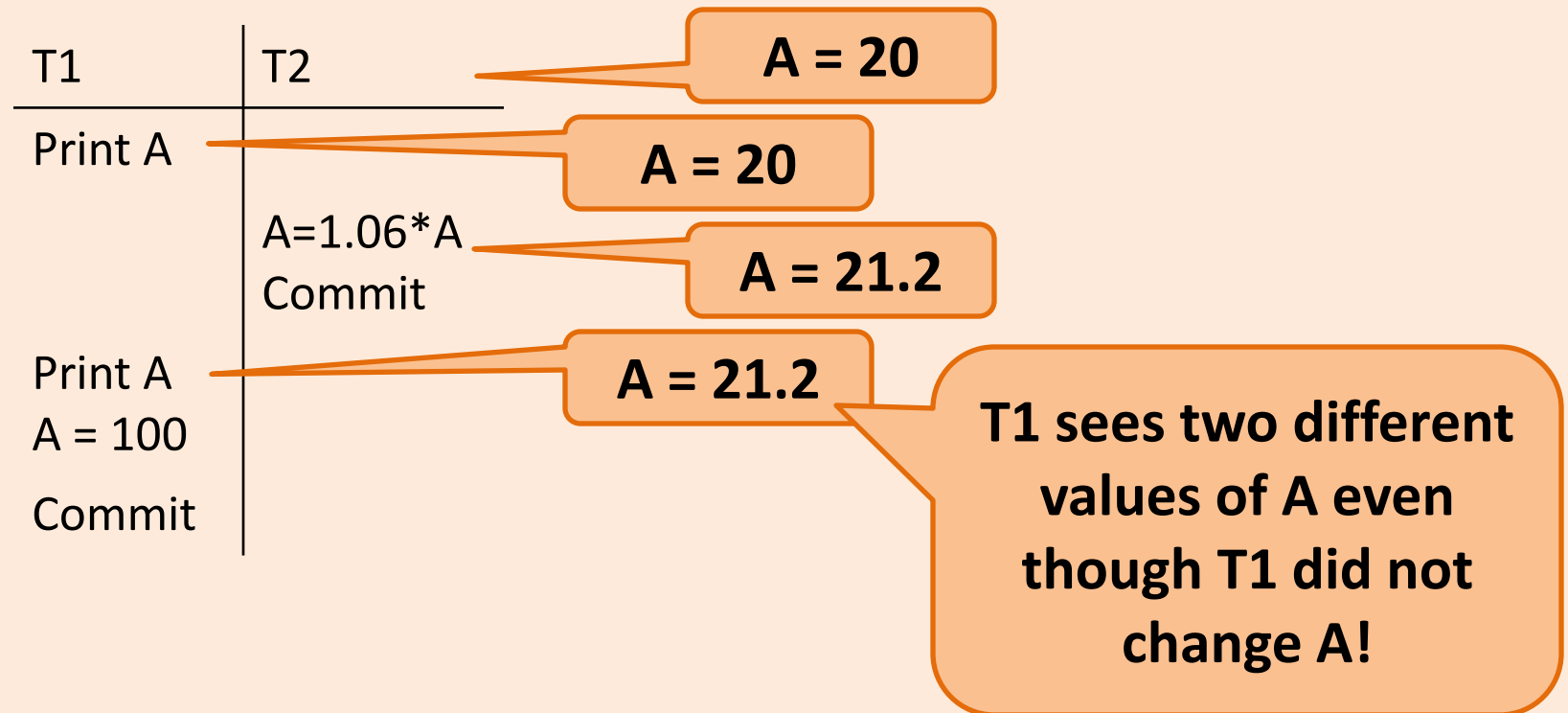
# Anomaly: Dirty Reads

- T1 reads uncommitted data from T2 which may abort



# Anomaly: Unrepeatable Reads

- T1 sees two different values of A, because updates are committed from another transaction (T2)



# Anomaly: Phantom Reads

- Multiple reads from the same transaction sees different set of tuples

