Topic 09: Transaction Management

ICT285 Databases
Dr Danny Toohey

About this topic

- It is essential to maintain the integrity (consistency) of the database when multiple users are accessing it, or if the system crashes. Central to both concurrency management and recovery is the idea of the transaction as a single logical unit of work (and hence, unit of recovery).
- The most common type of concurrency control is based on locking, which we will look at in this topic. Database recovery is usually based on maintaining a log of transactions which can be used to restore the database to a consistent

- http://claudiofiandrino.altervista.org/Master_degree/Database_management_system/concurrency_control.pdf

Topic learning outcomes

- **After completing this topic you should be able to:**
- Explain what is meant by a transaction, and the possible transaction outcomes of commit and rollback
- Use the SQL commands COMMIT and ROLLBACK
- Define the desirable transaction properties of atomicity, consistency, isolation and durability (ACID)
- Give examples of the problems (including lost update) that can occur during concurrent transaction execution
- Explain what is meant by serial, nonserial and serialisable schedules, and give examples of each
- Describe the use of locking for concurrency management, particularly the two-phase locking protocol
- /

Topic learning outcomes cont'd

- /
- Explain how deadlock may arise, and describe how deadlock may be handled by prevention or detection
- Explain the effect of data item granularity in concurrency control
- Explain the need for database recovery management
- Describe how the transaction log file is used in database recovery management

Resources for this topic

READING

- Text, Chapter 9 p459-470 “Concurrency Control";
  Text, Chapter 9 p477-480 "Database Backup and Recovery"

Other resources:

- Online Chapter 10B p10B-72 - 10B-75:
  “Oracle Database Concurrency Control”,
  “Oracle Database Backup and Recovery”
- Oracle Help:
  http://docs.oracle.com/database/121/CNCPT/consist.htm#CNCPT020 “Data Concurrency and Consistency”

Lab 9 –
Transaction management and concurrency

- The lab for this topic looks at how we can ensure the database stays consistent throughout multi-statement transactions and when multiple concurrent users are updating the database at the same time. To do this we examine the concept of a *transaction* as a single logical unit of work, and the practice of *locking* to manage concurrent users.

Topic outline

Transactions

Problems with concurrent transactions

Lost update problem

Uncommitted data problem

Inconsistent retrieval problem

Concurrency management

Serial, non-serial and serialisable schedules

Locking

Timestamping, optimistic locking techniques

Database recovery management

Log-based recovery using rollback/rollforward

1. Transactions

What is a transaction?

A transaction is a single logical unit of work that must be completed or aborted in its entirety; Does not allow intermediate states

- A transaction takes content of database and if database is to be alerted then database must alter from one consistent database state to another

- To make sure the database is constantly consistent, every transaction has to being from a known consistent state

- Most real-world database transactions are formed by two or more database requests.

  - A database request is the equivalent of a single SQL statement in an application program or transaction.

Transaction support in a DBMS

A transaction can either be a:

- Success – where a transaction *commits* and database is in a new consistent state.
- Failure – is where a transaction *aborts*, and database must be restored to the before state it was in.
  - Such a transaction is rolled back or undone.
- A transaction committed, can't be rolled back.
- An aborted transaction that is rolled back can be restarted later.

Example of a transaction

read(bal_x)	(t=1)
bal_x= bal_x+10	(t=2)
write(bal_x)	(t=3)
commit	(t=4)

This transaction has two database accesses: (t = time)

- At t=1: Reads the balance of x
- At t=3: Writes the new balance of x
- The “commit” statement tells the DBMS the transaction is complete

Another example:

- Suppose we transfer $10 from Fred's bank account to Mary's. The database transaction involves
  - 1. Subtract $10 from Fred's account
  - 2. Add $10 to Mary's account
- If the database is to remain consistent it is essential that either:
  - **Both** steps occur, or
  - **Neither** step occurs
- If only one step happens the database is no longer *consistent*
  - it doesn't represent any actual state in the real world

Action	Bal($_{Fred}$)	Bal($_{Mary}$)	Consistent?
Start	100	50	Yes
Debit Fred	90	50	No
Credit Mary	90	60	Yes
End			Yes

What should happen if there is a failure after Fred’s account is debited but BEFORE Mary’s account is credited?

Transactions in Oracle

- In Oracle a transaction begins implicitly at the first SQL statement and continues until either COMMIT or ROLLBACK is encountered.
  - e.g. Deleting a student and the details of their enrolments should be a single transaction:

```
delete records from ENROLMENT
delete record from STUDENT
COMMIT
```

- Other versions or dialects of SQL may use BEGIN TRANSACTION … END TRANSACTION to explicitly define the transaction, or ABORT instead of ROLLBACK.

Properties of a transaction (ACID)

- **Atomic** – a transaction is completed in its entirety, or not at all (it is not possible to partially complete)
- **Consistent** – a transaction needs to be consistent in that it takes the database from a state of consistency to another
- **Isolated** – other transactions shouldn’t be allowed to access any intermediate values of data used by the transaction
- **Durable** – a durable transaction is one in which all committed changes are written permanently in DBMS

The takeaways…

- A **transaction** is a single logical unit of work that the DBMS must ensure happens in its entirety or not at all
- A transaction moves the database from one consistent state to another
- SQL **commit** signals that a transaction is complete and can be written to the database, and **rollback** returns it to the original state
- A transaction must have the **ACID** properties – **a**tomicity, **c**onsistency, **i**solation, **d**urability

2. Problems with concurrent transactions (think more than one transaction trying to update the database at same time)

Concurrency control

- **Concurrency control** refers to the coordination of concurrent transactions in a multiprocessing database system
- Most large DBMSs are multi-user – many users need to access the system at the same time.

Transaction A___________ _ _ _ _ _______ _ _ _ _

Transaction B_ _ _ _ _ _ _ _________ _ _ _ _ _ _______

___________	program has CPU, is executing
_ _ _ _ _ _ _ _	program is waiting for CPU, suspended or doing I/O

- Transactions are processed **concurrently** because one transaction operating on the DB at one time is very slow, although to the users it appears simultaneous

Concurrency control

- Concurrency control is how the simultaneous operations on the database are managed to stop them from interfering with each other
- Prevents interference when two or more users are accessing the database at the same time and one user is updating data in the database
- Although the two transactions may be correct but transaction interleaved may create an incorrect result

Problems with concurrency

- ~~If two transactions both access the **same database item**, it is possible for their operations to become interleaved in a way that leaves the database inconsistent~~

- There are various potential problems:
  - Lost Update
  - Uncommitted Data
  - Inconsistent Retrieval

(Note that different resources may use different terms for the same problems)

Uncommitted data problem

Occurs when one transaction views intermediate results of another transaction before it has been committed/completed violates the isolation property

Also called *dirty read*

TRANSACTION	COMPUTATION
T1: Purchase 100 units	PROD_QOH = PROD_QOH + 100 (Rolled back)
T2: Sell 30 units	PROD_QOH = PROD_QOH - 30

- Correct execution of the transactions …

Uncommitted data problem (without concurrency)

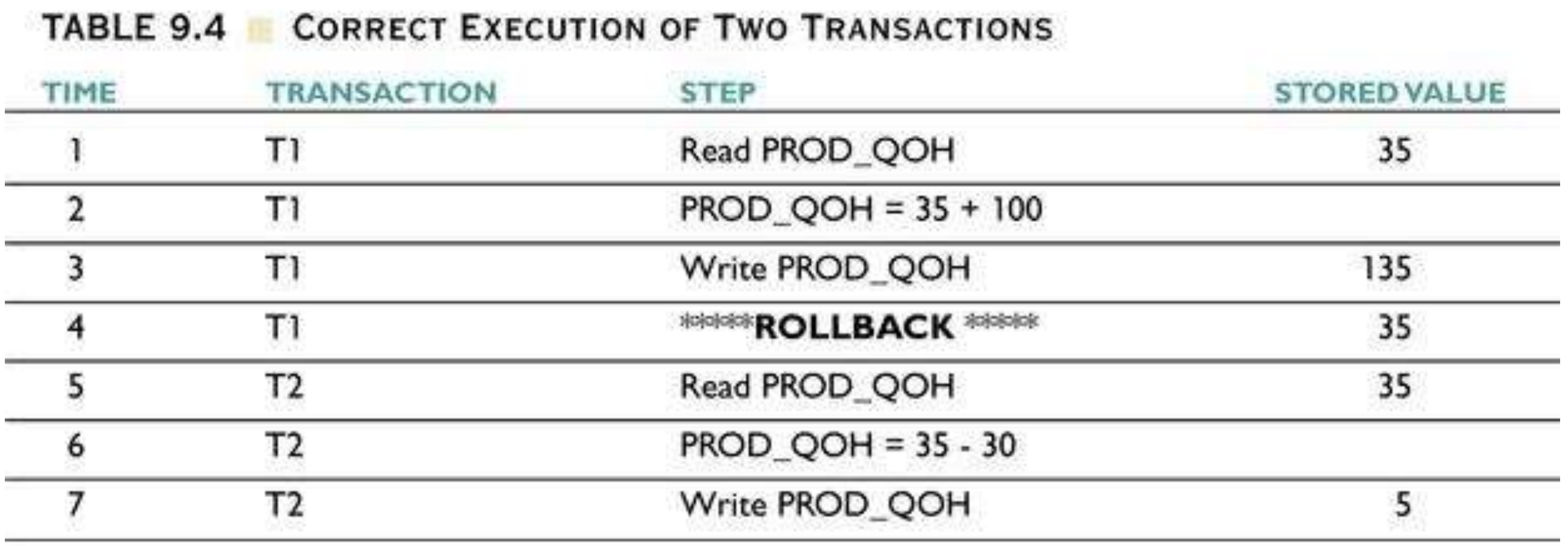


TABLE 9.4 CORRECT EXECUTION OF TWO TRANSACTIONS

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T1	PROD_QOH = 35 + 100	
3	T1	Write PROD_QOH	135
4	T1	*****ROLLBACK *****	35
5	T2	Read PROD_QOH	35
6	T2	PROD_QOH = 35 - 30	
7	T2	Write PROD_QOH	5

But, what if T2 reads T1's data before the rollback and assumes it is correct…

Uncommitted data problem (with concurrency)

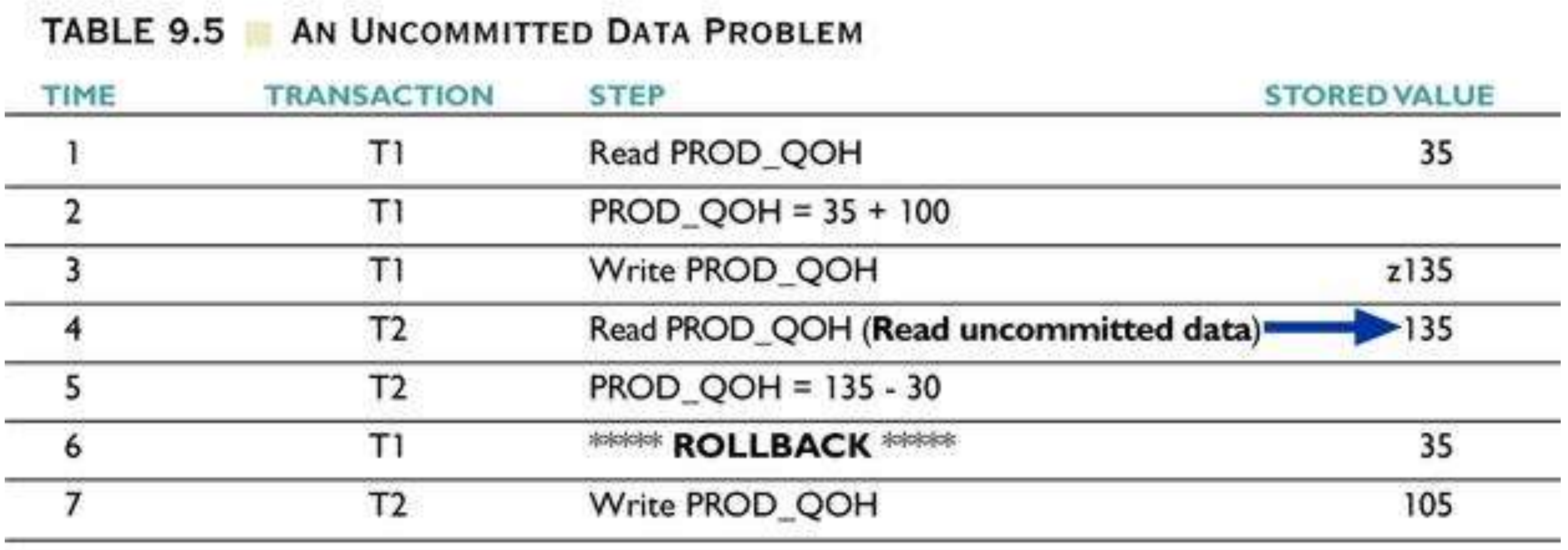


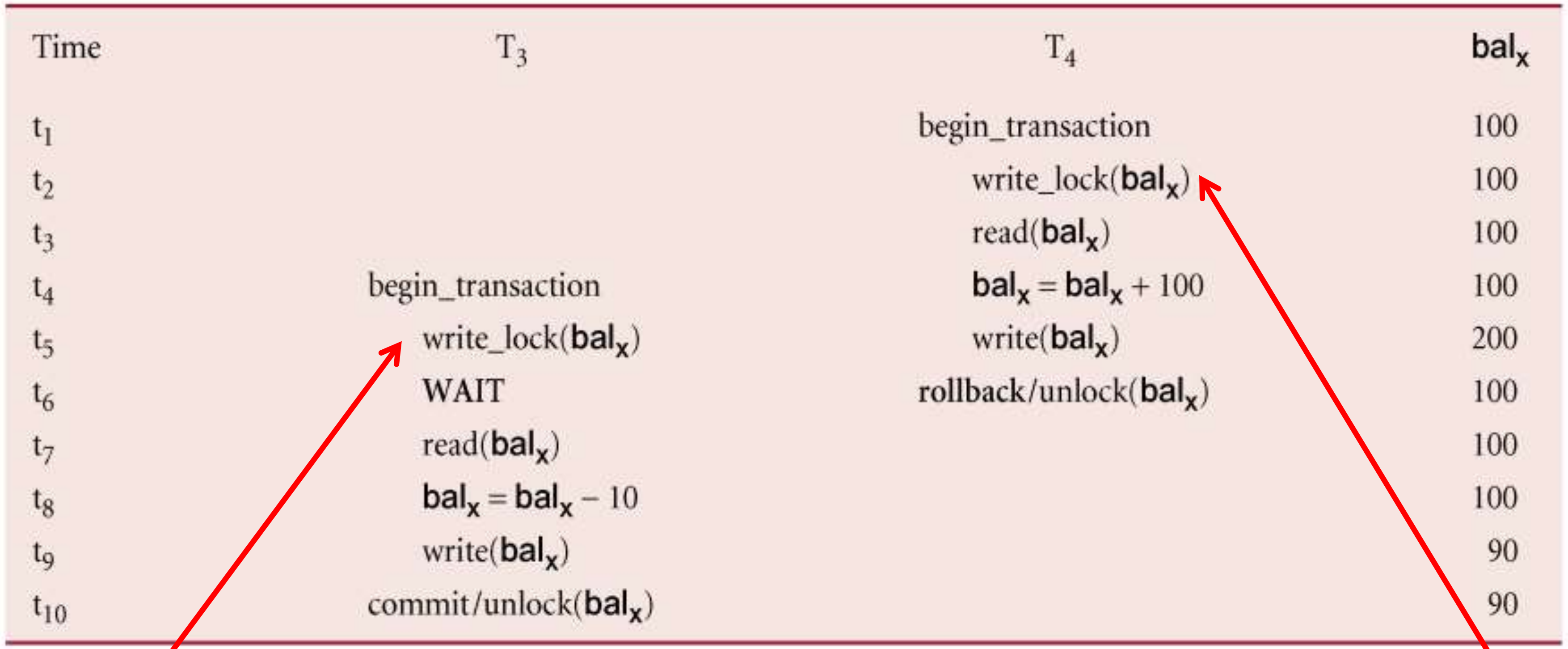
TABLE 9.5 An Uncommitted Data Problem

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T1	PROD_QOH = 35 + 100	
3	T1	Write PROD_QOH	z135
4	T2	Read PROD_QOH (**Read uncommitted data**) →	135
5	T2	PROD_QOH = 135 - 30	
6	T1	***** **ROLLBACK** *****	35
7	T2	Write PROD_QOH	105

T2 reads t1 before t1 is completed. (This table shows what would happen). The rollback is part of transaction 1 thus transaction 2 reads before transaction is completed

The QOH is now 105 – when we know it should be 5!

Preventing uncommitted data problem using TPL



Time	T_3	T_4	bal_x
t_1		begin_transaction	100
t_2		write_lock(bal_x)	100
t_3		read(bal_x)	100
t_4	begin_transaction	$bal_x = bal_x + 100$	100
t_5	write_lock(bal_x)	write(bal_x)	200
t_6	WAIT	rollback/unlock(bal_x)	100
t_7	read(bal_x)		100
t_8	$bal_x = bal_x - 10$		100
t_9	write(bal_x)		90
t_{10}	commit/unlock(bal_x)		90

T4 puts an exclusive lock on bal_x

T3 is forced to wait. Only once T4 is fully completed is T4 read by T3

Lost Update problem

Occurs when an apparently successfully completed update operation by one user is overwritten (ignored) by another user violates the consistency property which is...

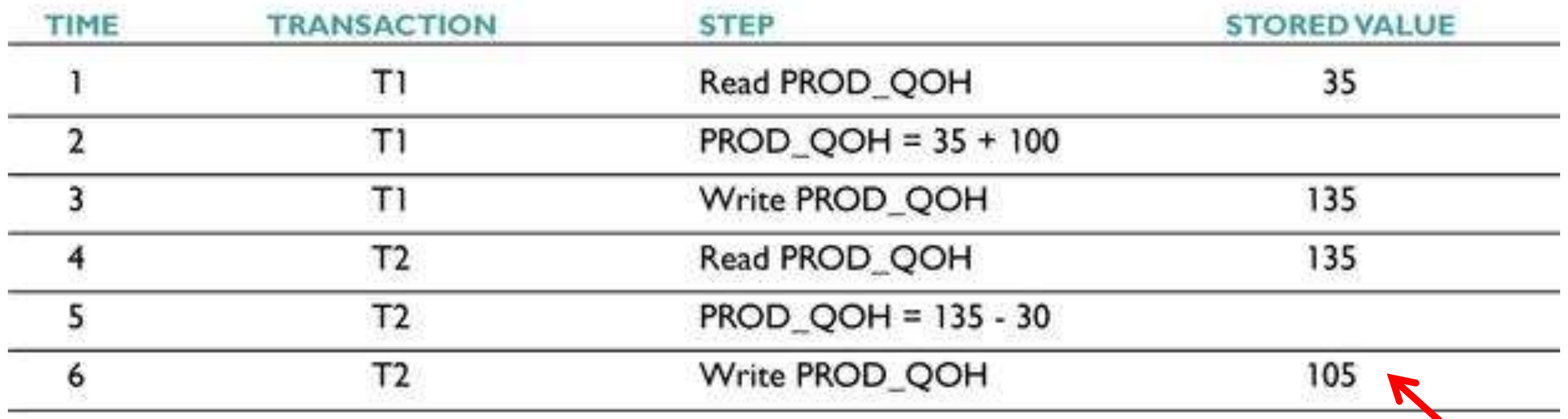
Consider the PRODUCT table, which has an attribute QOH (quantity on hand how many items we got left). The initial QOH of a particular item is 35.

TRANSACTION	COMPUTATION
T1 (transaction 1): Purchase 100 units	QOH = QOH + 100
T2: Sell 30 units	QOH = QOH - 30

- Under normal circumstances, the transactions will be executed thus… (next slide)

Lost Update problem (without concurrency)

TABLE 9.2 NORMAL EXECUTION OF TWO TRANSACTIONS



TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T1	PROD_QOH = 35 + 100	
3	T1	Write PROD_QOH	135
4	T2	Read PROD_QOH	135
5	T2	PROD_QOH = 135 - 30	
6	T2	Write PROD_QOH	105

However, if a transaction was able to read a QOH before a previous transaction has been committed, the following will happen …

Lost Update problem (with concurrency)

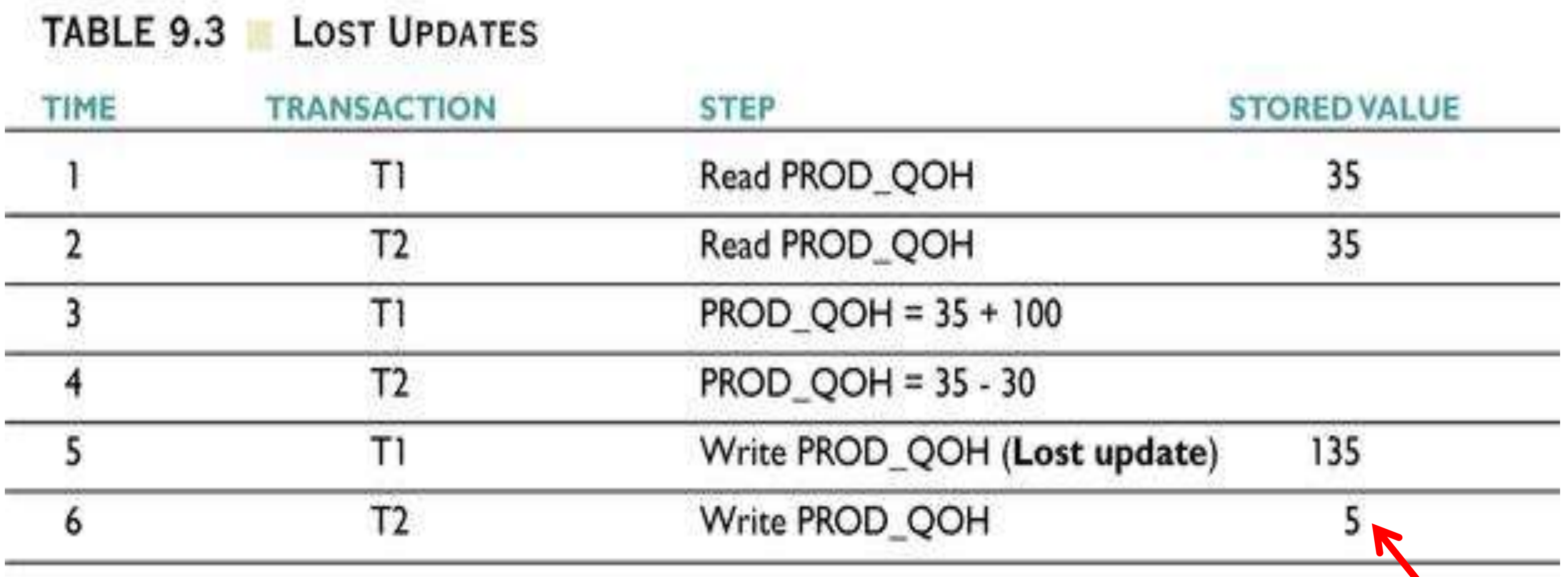


TABLE 9.3 LOST UPDATES

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T2	Read PROD_QOH	35
3	T1	PROD_QOH = 35 + 100	
4	T2	PROD_QOH = 35 - 30	
5	T1	Write PROD_QOH (**Lost update**)	135
6	T2	Write PROD_QOH	5

Transaction 2 Ignores transaction 1 completely. For example, time = 5 is lost in the shuffle. So specifically the update is lost

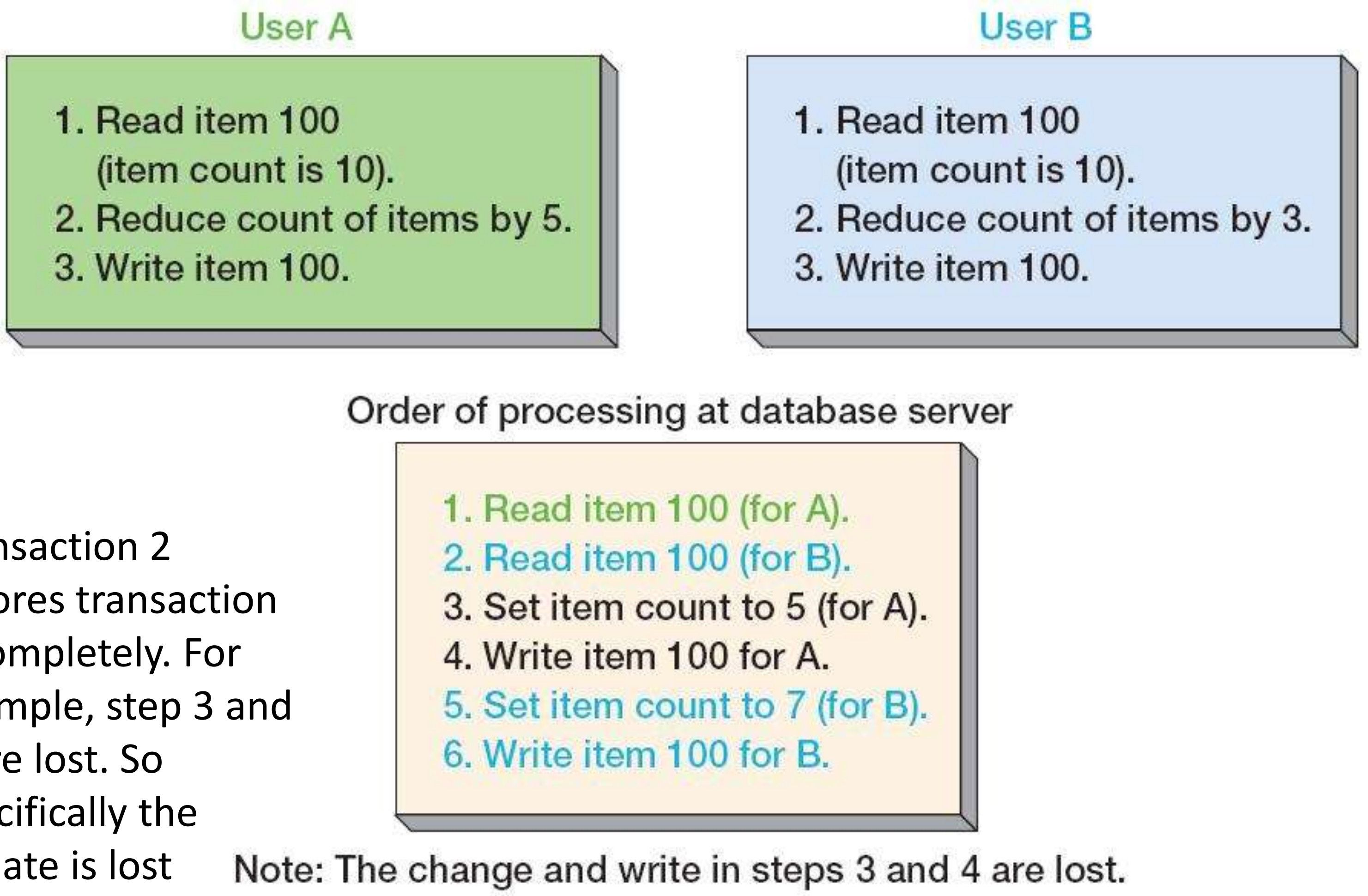
Preventing lost update problem using TPL

Time	T_1	T_2	bal_x
t_1		begin_transaction	100
t_2	begin_transaction	write_lock(bal_x)	100
t_3	write_lock(bal_x)	read(bal_x)	100
t_4	WAIT	$bal_x = bal_x + 100$	100
t_5	WAIT	write(bal_x)	200
t_6	WAIT	commit/unlock(bal_x)	200
t_7	read(bal_x)		200
t_8	$bal_x = bal_x - 10$		200
t_9	write(bal_x)		190
t_{10}	commit/unlock(bal_x)		190

T2 requests an exclusive lock on bal_x so it can read and modify → Write_lock(Balc)

T1 also requests exclusive lock on bal_x. T1 but must wait until t2 is completed. Before it can put an exclusive lock on balx. Therefore, the update from transaction 2 is not lost since transaction 1 will read the update and do something with it

Lost Update problem



User A:

1. Read item 100 (item count is 10).
2. Reduce count of items by 5.
3. Write item 100.

User B:

1. Read item 100 (item count is 10).
2. Reduce count of items by 3.
3. Write item 100.

Order of processing at database server:

1. Read item 100 (for A).
2. Read item 100 (for B).
3. Set item count to 5 (for A).
4. Write item 100 for A.
5. Set item count to 7 (for B).
6. Write item 100 for B.

Note: The change and write in steps 3 and 4 are lost.

Think of these are two transactions instead of users

Transaction 2 Ignores transaction 1 completely. For example, step 3 and 4 are lost. So specifically the update is lost

Inconsistent read problem

Occurs when there are two transactions and one transaction reads the data twice and in between the two reads another transaction updates the data thus causing an inconsistent read violates isolation property

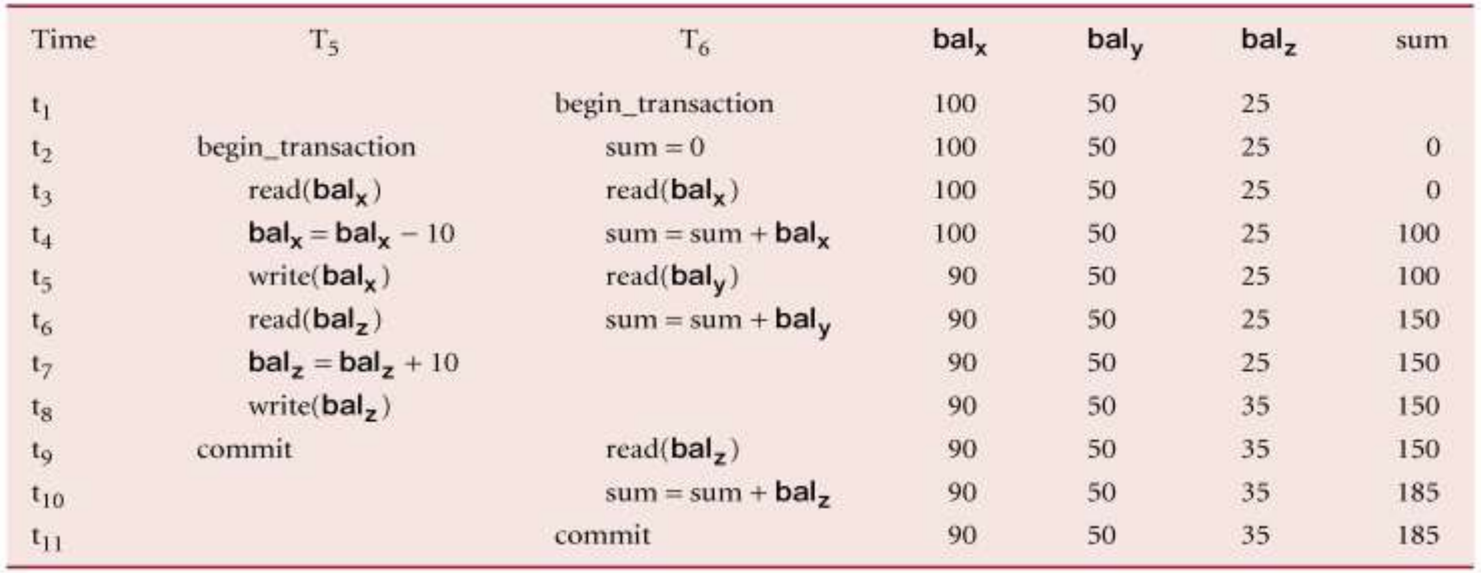
Also known as

Non-repeatable read

Phantom read

Inconsistent read problem (with concurrency)

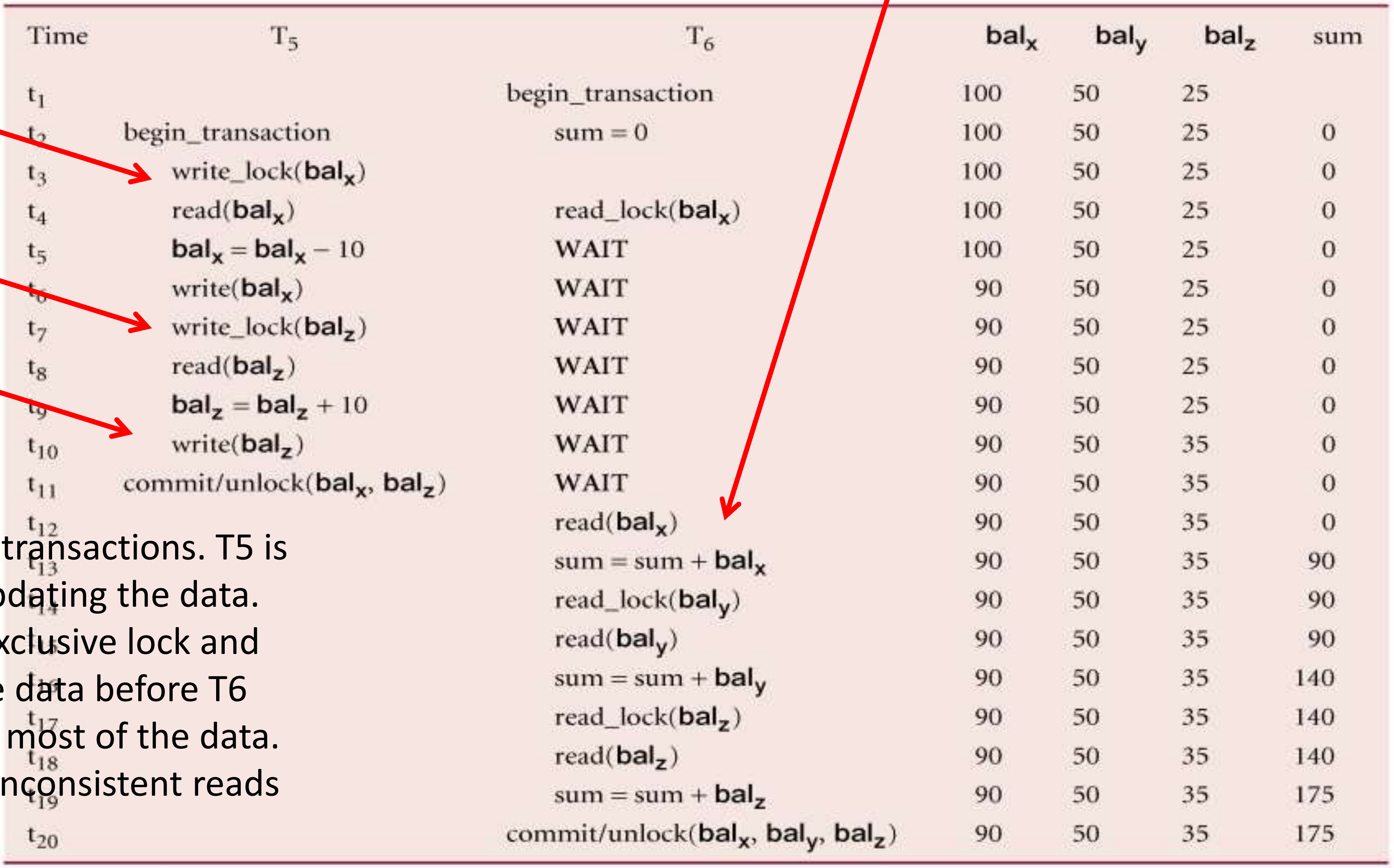
- T_6 is totaling balances of account x (\$100), account y (\$50), and account z (\$25).
- Meantime, T_5 has transferred \$10 from bal_x to bal_z, so T_6 now has wrong result (\$10 too high).



Time	T_5	T_6	bal_x	bal_y	bal_z	sum
t_1		begin_transaction	100	50	25	
t_2	begin_transaction	sum = 0	100	50	25	0
t_3	read(bal_x)	read(bal_x)	100	50	25	0
t_4	$bal_x = bal_x - 10$	sum = sum + bal_x	100	50	25	100
t_5	write(bal_x)	read(bal_y)	90	50	25	100
t_6	read(bal_z)	sum = sum + bal_y	90	50	25	150
t_7	$bal_z = bal_z + 10$		90	50	25	150
t_8	write(bal_z)		90	50	35	150
t_9	commit	read(bal_z)	90	50	35	150
t_{10}		sum = sum + bal_z	90	50	35	185
t_{11}		commit	90	50	35	185

- Problem avoided by preventing T_6 from reading bal_x and bal_z until after T_5 completed updates.

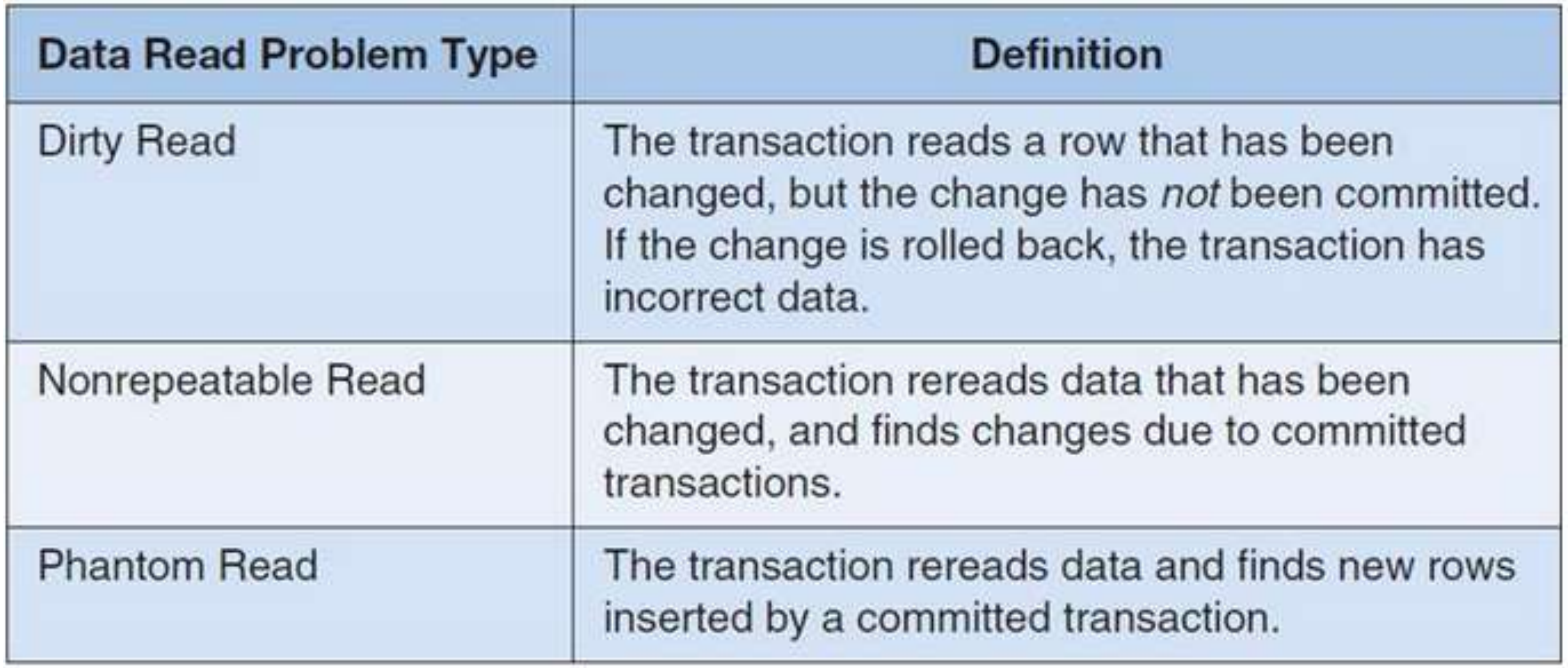
Preventing inconsistent retrievals using 2PL



Time	T_5	T_6	bal_x	bal_y	bal_z	sum
t_1		begin_transaction	100	50	25	
t_2	begin_transaction	sum = 0	100	50	25	0
t_3	write_lock(bal_x)		100	50	25	0
t_4	read(bal_x)	read_lock(bal_x)	100	50	25	0
t_5	$bal_x = bal_x - 10$	WAIT	100	50	25	0
t_6	write(bal_x)	WAIT	90	50	25	0
t_7	write_lock(bal_z)	WAIT	90	50	25	0
t_8	read(bal_z)	WAIT	90	50	25	0
t_9	$bal_z = bal_z + 10$	WAIT	90	50	25	0
t_{10}	write(bal_z)	WAIT	90	50	35	0
t_{11}	commit/unlock(bal_x, bal_z)	WAIT	90	50	35	0
t_{12}		read(bal_x)	90	50	35	0
t_{13}		sum = sum + bal_x	90	50	35	90
t_{14}		read_lock(bal_y)	90	50	35	90
t_{15}		read(bal_y)	90	50	35	90
t_{16}		sum = sum + bal_y	90	50	35	140
t_{17}		read_lock(bal_z)	90	50	35	140
t_{18}		read(bal_z)	90	50	35	140
t_{19}		sum = sum + bal_z	90	50	35	175
t_{20}		commit/unlock(bal_x, bal_y, bal_z)	90	50	35	175

There are two transactions. T5 is in charge of updating the data. T5 places an exclusive lock and updates all the data before T6 begins to read most of the data. This prevents inconsistent reads arising

Summary of data read problem types

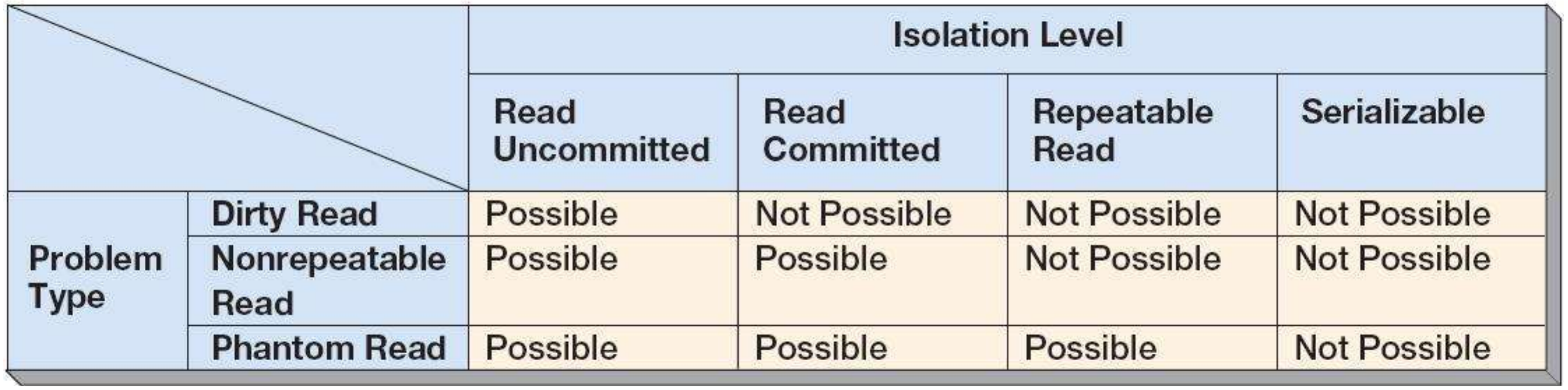


Data Read Problem Type	Definition
Dirty Read	The transaction reads a row that has been changed, but the change has *not* been committed. If the change is rolled back, the transaction has incorrect data.
Nonrepeatable Read	The transaction rereads data that has been changed, and finds changes due to committed transactions.
Phantom Read	The transaction rereads data and finds new rows inserted by a committed transaction.

- The SQL standard defines different **isolation levels** that control these. The programmer declares the isolation level required and the DBMS manages locks to achieve it

Kroenke & Auer, Figure 9-11

Transaction isolation levels in the SQL standard and data read problems



		Isolation Level			
		Read Uncommitted	Read Committed	Repeatable Read	Serializable
Problem Type	Dirty Read	Possible	Not Possible	Not Possible	Not Possible
	Nonrepeatable Read	Possible	Possible	Not Possible	Not Possible
	Phantom Read	Possible	Possible	Possible	Not Possible

- The ‘serialisable’ level of isolation prevents all of the potential data read problems
- DBMSs vary in their support for these levels
- Oracle supports Serialisable, Read Committed (default) and Read Only

Summary of concurrency problems

- **Lost update**
  - Occurs when an apparently successfully completed update operation by one user is overwritten by another user
- **Uncommitted data (dirty read)**
  - Occurs when one transaction can see intermediate results of another transaction before it has committed
- **Inconsistent read**
  - Occurs when a transaction reads several values but a second transaction updates some of them during execution of first

The takeaways…

- Concurrent transactions are those that are executed ‘at the same time’, by interleaving their steps
- When one transaction is able to access data being used by another transaction before that transaction is complete, it is possible for the database to become inconsistent
- There are several potential problems: lost update, uncommitted data, and inconsistent read
- -- There needs to be some way of ensuring that transactions can be interleaved but in such a way that the database remains consistent

3. Concurrency management

Concurrency management

- How to manage concurrent transactions?

Method 1:

- A **schedule** is a sequence of the operations by a set of concurrent transactions that preserves the order of the operations in each of the individual transactions
- Schedules can be
  - Serial
  - non-serial
  - Serialisable schedule

Serial schedule

- Every statements in each transaction are performed consecutively (serially) with no interleaving operations from other transactions
  - Data isolation is guaranteed since we wait for all the transaction to complete

Disadvantage

- Processing time is wasted because CPU waits for a write or read operation to be complete, thus losing several cycles. This results in unacceptable response times

Examples of serial schedules

Consider the concurrent transactions:

• Transaction A :	read X;	X := X+10;
• Transaction B:	read X;	X := X*2;

These could be scheduled as:

- A then B, or
- B then A

So If X = 100

- A then B results in a final value of	X = 220
- B then A results in a final value of	X = 210

Either result is considered correct. If each individual transaction is correct, then the **serial schedule is correct**

Non-serial schedule

- Operations in different transactions are **interleaved**

Advantage-

- Better use of the CPU and improved response times

Disadvantage-

- BUT as we have seen, this can cause problems (e.g. lost update, uncommitted data, inconsistent retrievals)

Serialisability

- **Serialisable** schedule.
    - Guarantees correctness of a serial schedule + benefits in performance through interleaving

We need to ensure that if a set of transactions executes concurrently it produces *the same results as some serial execution*

Guaranteeing serialisability

- For concurrency control, DBMS needs to have techniques to guarantee serialisability
- Typical techniques are:

Pessimistic Locking- predict conflicts will occur, and thus we need to manages schedule to avoid conflict

Optimistic Locking-

Timestamping

Locks

- ~~Concurrency can be controlled using **locks**~~
- ~~A lock guarantees exclusive use of a data item to a current transaction~~
- ~~A transaction acquires a lock prior to data access; the lock is released (unlocked) when the transaction is completed~~
- ~~Locks can be either placed by command (*explicit lock*) or by the DBMS (*implicit lock*)~~
- ~~All lock information is managed by a lock manager~~

Type of lock– binary locks

- A binary lock has either locked (1) or unlocked (0) state
- Locked state is where a transaction locks onto an object so other transaction can't use the object
- Unlocked state is where no transaction are currently locked onto the object so any transaction can lock onto it and use it
- Transaction must unlock the object after termination

- ~~Every transaction requires a lock and unlock operation for each data item that is accessed~~
- ~~Very restrictive~~

Type of lock - shared/exclusive locks

- **Shared** lock on an item/object by a transaction means the transaction(s) can **read but NOT modify** the item/object

- **Exclusive** lock on an item/object by a transaction means the transaction can **read and modify** the item/object
  - **An item can only be exclusively locked by one transaction at one** time. Thus other transactions must wait until this is released before they can gain any lock on the item:

Current Lock on Item		**shared**	**exclusive**	**no lock**
Request	**shared**	Yes	No	Yes
for Lock	**exclusive**	No	No	Yes

Advantage-

- More flexible in that many transaction can hold a shared lock on an item thus allowing an item to be read by many transactions at one time

Shared/exclusive locks

- However, *locking on its own* doesn't guarantee serialisability -
  - Also need some form of protocol or rules about **when** to lock/unlock

Two-Phase Locking (TPL)

TPL defines how transactions acquire and relinquish locks

- In the **growing** phase, a transaction gets all the locks it needs but it will not be unlocking any locks during this phase. When a transaction gets all its locks, the transaction is in its locked point
- In the **shrinking** phase, a transaction releases all its locks and cannot get any new locks

- Guarantees serialisability but comes at the price of having locks created earlier and held for longer than is strictly necessary
- Does <u>not</u> prevent deadlocks

Two-Phase Locking Protocol

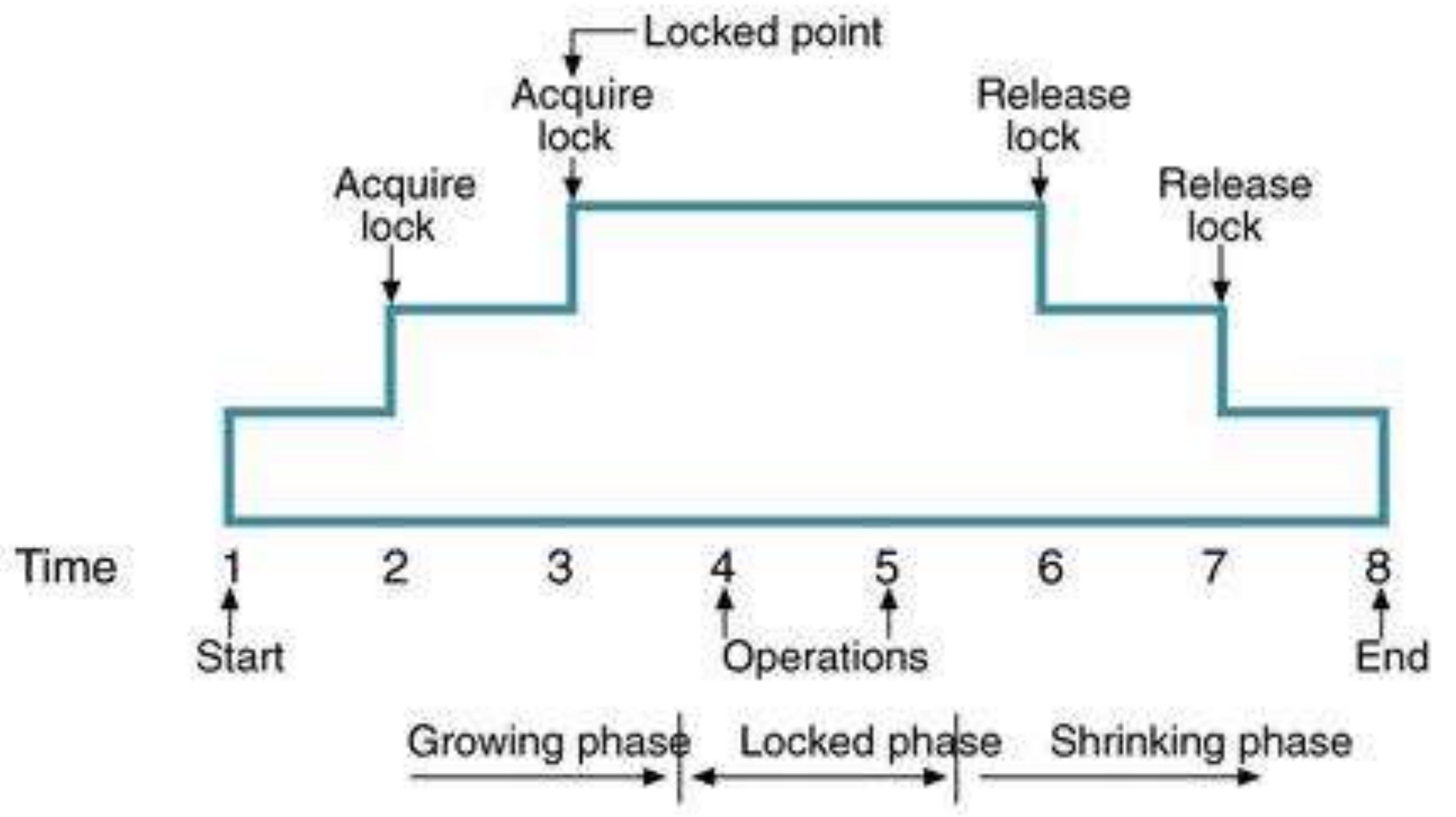


FIGURE 9.6 Two-Phase Locking Protocol

Note growing phase and shrinking phase

Deadlock (deadly embrace)

- Dead locks is the period where there are two transactions and each transaction waits for locks to be released that are held by the other.

- For example, when two transactions T1 and T2 exist in the following mode:

  T1 has data item X and needs Y

  T2 has data item Y and needs X

- If T1 does not unlock data item X, T2 cannot begin; and, if T2 does not unlock data item Y, T1 cannot continue.
  But neither can unlock as are still in growing phase!

How A Deadlock Condition Is Created

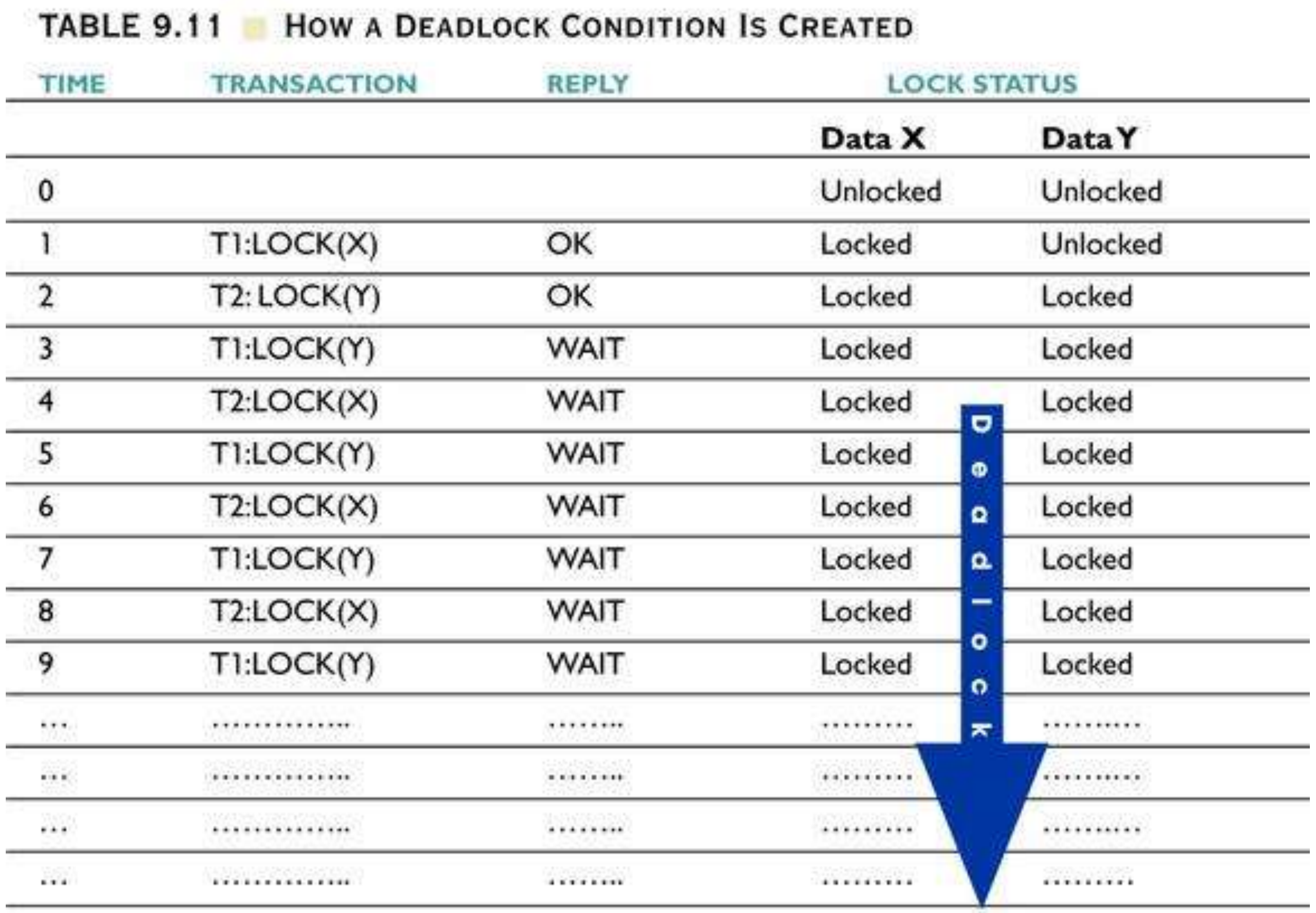
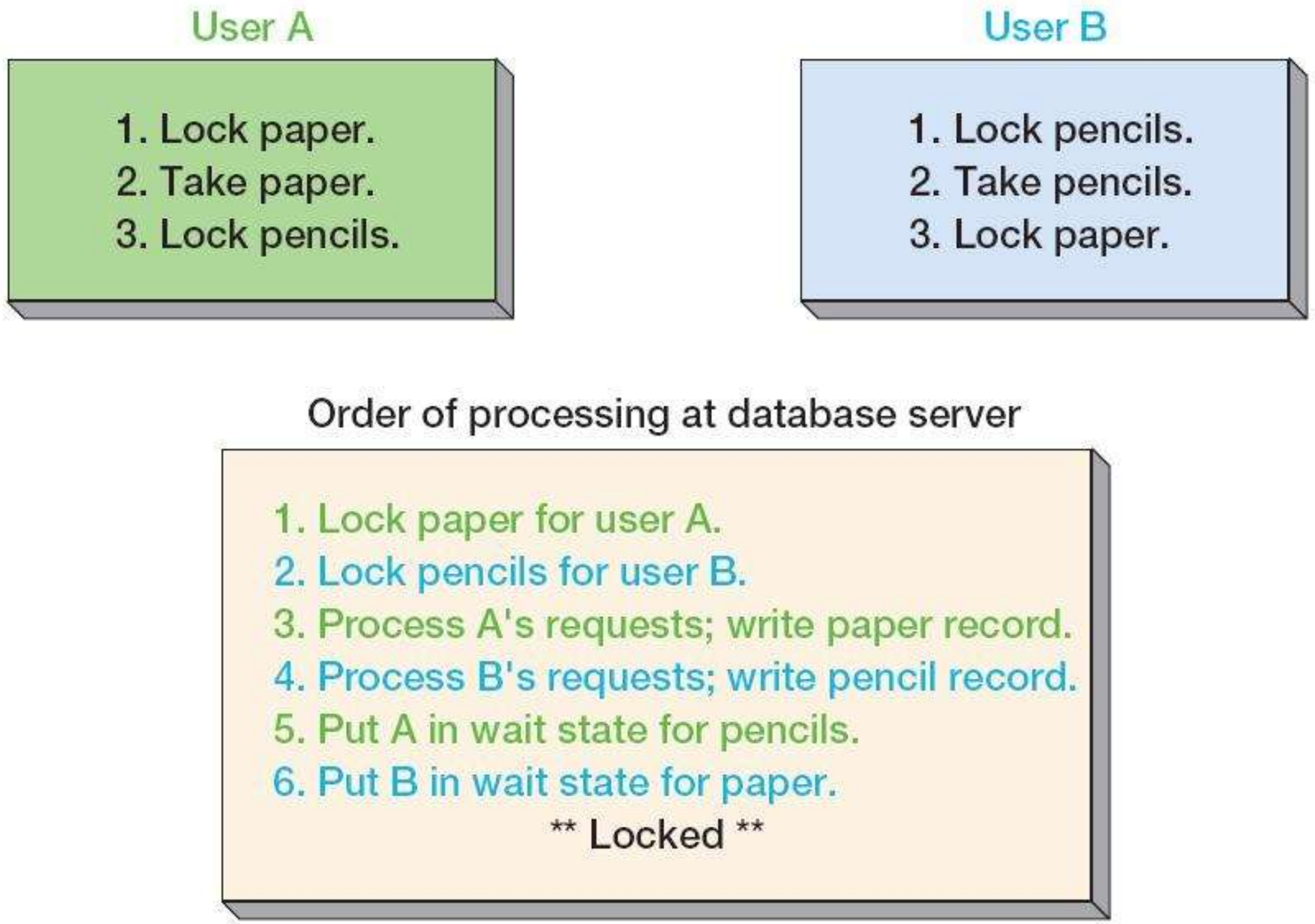


TABLE 9.11 How a Deadlock Condition Is Created

TIME	TRANSACTION	REPLY	LOCK STATUS	
			Data X	Data Y
0			Unlocked	Unlocked
1	T1:LOCK(X)	OK	Locked	Unlocked
2	T2: LOCK(Y)	OK	Locked	Locked
3	T1:LOCK(Y)	WAIT	Locked	Locked
4	T2:LOCK(X)	WAIT	Locked	Locked
5	T1:LOCK(Y)	WAIT	Locked	Locked
6	T2:LOCK(X)	WAIT	Locked	Locked
7	T1:LOCK(Y)	WAIT	Locked	Locked
8	T2:LOCK(X)	WAIT	Locked	Locked
9	T1:LOCK(Y)	WAIT	Locked	Locked
...	....	....	....	....
...	....	....	....	....
...	....	....	....	....
...	....	....	....	....

Deadlock


User A
1. Lock paper.
2. Take paper.
3. Lock pencils.
User B
1. Lock pencils.
2. Take pencils.
3. Lock paper.
Order of processing at database server
1. Lock paper for user A.
2. Lock pencils for user B.
3. Process A's requests; write paper record.
4. Process B's requests; write pencil record.
5. Put A in wait state for pencils.
6. Put B in wait state for paper.
** Locked **

Deadlock

Only one way to break deadlock: abort one or more of the transactions.

Three general techniques for handling deadlock:

- Timeouts
- Deadlock prevention
- Deadlock detection and recovery

Deadlock should be transparent to user, so DBMS should restart transaction(s).

Deadlock - Timeouts

- The Transaction that wants to put an exclusive lock on an object that already has been exclusively locked by another object, will only wait for a system-defined period of time.
- Lock requests will time out when the locks isn’t granted within a system-defined period of time. And this result in the transactions being aborted and restarted

Deadlock - Prevention

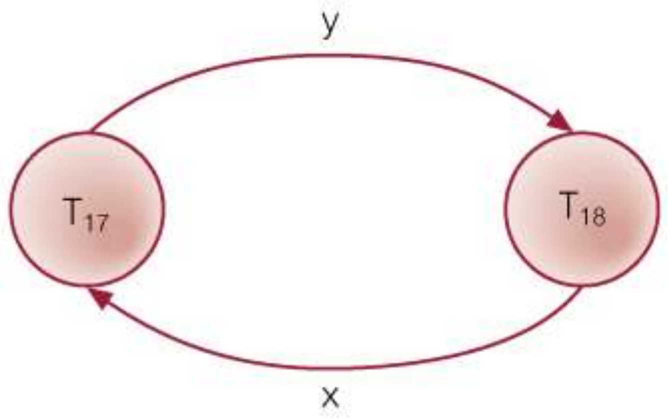
DBMS predicts whether transactions in the future would create a deadlock. If they do then steps to help prevent the creation of a deadlock are implemented.

These steps normally include the abortion and restart of certain transactions.

The transactions to abort and restart are influenced by age of the transaction as determined by the timestamp when it entered the system

Deadlock - Detection and Recovery

- DBMS allows deadlock to happen but identifies it and breaks it.
- Usually handled by construction of wait-for graph (WFG) showing transaction dependencies:
  - Deadlock exists if and only if WFG contains cycle
- WFG is created at regular intervals.



Recovery from deadlock detection - issues

~~Choice of deadlock victim~~

- ~~Which transaction caused the deadlock?~~
- ~~How long has the transaction been running?~~
- ~~How many data items have been updated by the transaction?~~
- ~~How many data items are still to be updated by the transaction?~~

~~How far to roll a transaction back~~

- ~~Easier to simply rollback entire transaction, but not necessarily the most efficient~~

~~Avoiding 'starvation'~~

- ~~The algorithm must ensure same transaction should not always be chosen as the victim~~

Lock granularity

- Determines how much of the database is locked
    - Can be database, table, page, row, and column (field)

- ~~In general there is a tradeoff between the amount of concurrency permitted and the amount of overhead needed to implement~~
- ~~The best solution depends on the particular transaction and what it does~~
    - ~~Ideally a DBMS would support a mixture of granularities~~

Lock granularity - levels

- **Database** level
  - Whole database is locked and other transactions can’t use any of the table while the whole database is locked
  - Advantage
    - Good for batch processing of 100,000 rows of updates for instance, but not suitable for online multi-user databases because access to database would be to slow
- **Table** level
  - Whole table is locked when a transaction wants to access the table. A single transaction can request to access multiple tables which would result in multiple tables being locked as well

  Disadvantage-
  - restrictive in a multi-user environment since some transaction needs to access the same table thus waiting time is required

Lock granularity - levels

- **Page** level (common)
  - Tables may span many pages
  - The page holding the record that is required for the update is locked
- **Row** level
  - For different rows in the same table allows for concurrent access
  - Disadvantage
    - high level of overhead in maintaining the lock thus high maintenance cost
  - Advantage is increase speed of updates from allowing concurrent access

- **Field** level
  - Allows concurrent access to different fields in the same row
  - Disadvantage is VERY HIGH level of overhead in maintaining the lock thus very high maintenance cost.
  - Advantage is increase speed of updates from allowing concurrent access

Locking in Oracle

- ~~Oracle is intended for serious multi-user applications and can handle thousands of concurrent transactions~~
- Data definition (create table, alter table..) and insert, delete, update require **exclusive** locks
- Share locks are required for some operations but not select - select queries need no locking
- ~~The default lock level is **row** (record locking) so two transactions can update different rows in the same table with no contention~~

Locking in Oracle

- ~~Oracle only reads committed changes; it will never read dirty data.~~
- ~~Oracle deals with deadlock by detection and recovery~~

Optimistic Locking

- Predict conflicts will not occur thus transactions proceed on that assumption.
- Prior the commitment of a transaction, a check is made to view whether there were any conflicts that might affect the consistency of the database, and the transaction is aborted and restarted if so
- Optimistic locking is ideal for the Internet and for many intranet applications since those require bulk loading of transactions thus try to reduce time

Timestamping

- Locks will not be used thus we can’t get any deadlock
- A global unique timestamp is given to each transaction and the transaction uses it to organise concurrent transactions in timestamp order
- If any conflicts were to arise older transaction gets priority

The takeaways…

- The aim of concurrency management is to enable transactions to be interleaved for performance reasons while remaining correct
- **Serialisable** schedules are interleaved schedules equivalent to a serial schedule
- There are various methods for ensuring serialisable schedules:
  - Locking methods, principally **Two-Phase Locking (TPL)**
  - Timestamping
- **Deadlock** can be a problem in any method that uses locking

4. Database recovery management

Database recovery management

- Recovery management is utilized when a transaction fails and it restores database back to a consistent state

- In high volume systems, it is vital that the database is able to recover within a very short time
- Recovery techniques are generally based on the **atomic transaction property:**

  ALL portions of the transaction must be applied and completed to produce a consistent database. If, for some reason, any transaction operation cannot be completed, the transaction must be aborted, and any changes to the database must be rolled back

Database recovery management

When a transaction writes to the database:

1. The item is updated in a buffer in main memory
2. The contents of the buffer are then written back to disk

- Only after step 2 is the write permanently recorded on disk
- So if there is a failure involving loss of memory between steps 1 and 2, it is possible that even though all the operations in the transaction have completed, the write is lost

Database recovery management

- The **commit** statement in a transaction means that all the operations have been completed and it should now be written to the database
- No commit transaction not completed and so all operations must be rolledback
- Basis for recovery management:
  - If a transaction has issued a commit statement, its effects should not be lost, whether or not the buffers have actually been written out (ensures **durability** property), and
  - If there is no commit statement, the transaction has not completed, so all its operations must be undone (rollback) to ensure **atomicity**

Techniques for recovery management

Recovery via reprocessing

- Database returns to a identified save point and the workload gets reprocessed from there
- If the computer is strictly scheduled then the recovered system may not catch up therefore not an ideal strategy
- Reprocessing may not be in the original order

Shadow paging

- Whenever there is an updates to a page in database copies of the pages will be produced. And if transaction is to be rolled back the original page is restored.

Log-based recovery (Rollback/rollforward)

Log based recovery

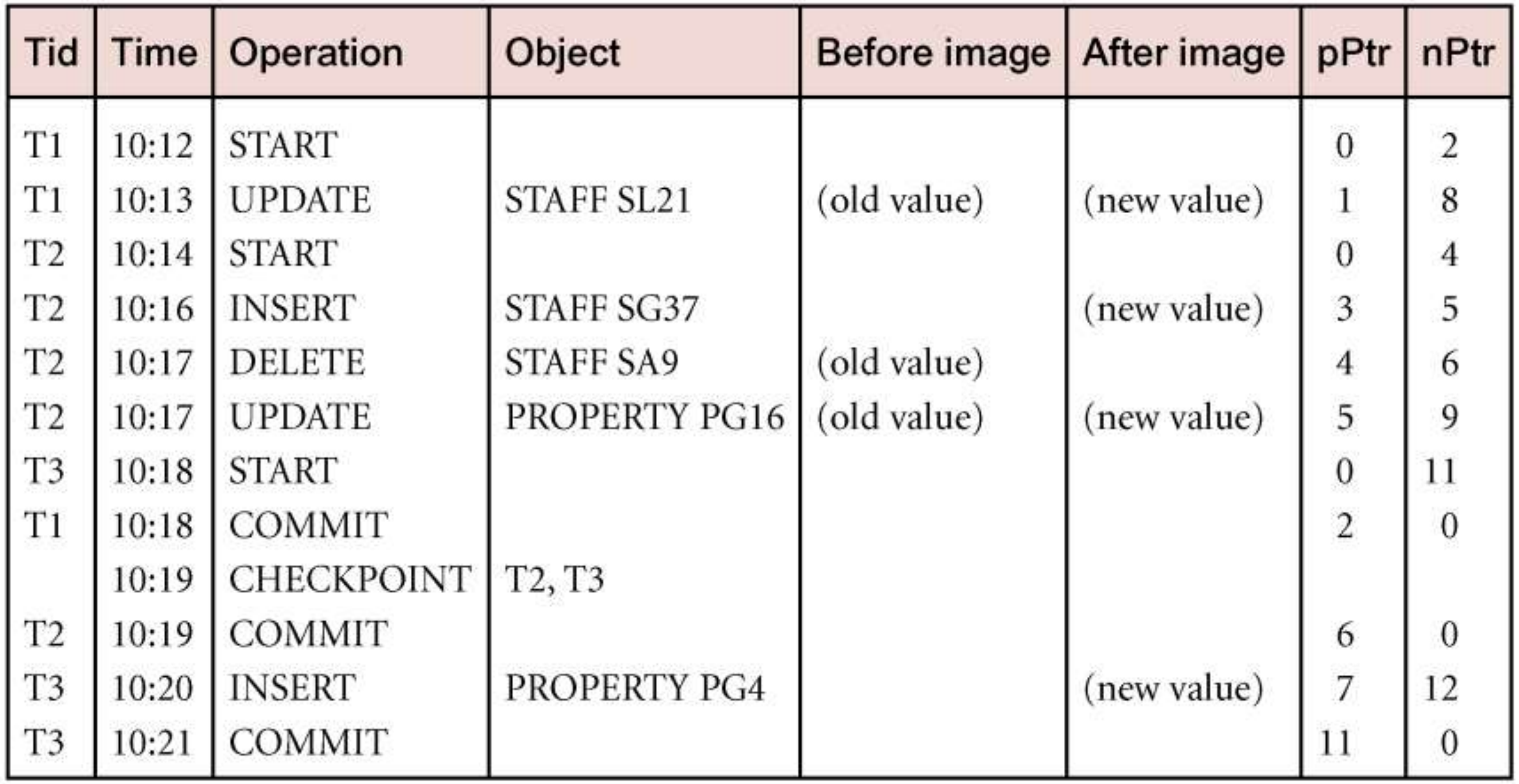
- The system records progress of each transactions in a log, and undoes/redoes effect of transactions from the log
- Information about any updates the database is doing is always written to log prior to the database is updated
  - As a result if a crash arises, there is no possibility that the database has been updated without the log having a record of it
- The log keeps track of before and after images in the log so if there is a failure we can utilize the images to return the database back to a consistent state
- Note that the **database itself** may be updated either:
  - At the commit point (deferred update)
  - During the transaction (immediate update)

Log based recovery

The DBMS maintains a log which records at least:

- Transaction ID
- Type of log record, e.g. transaction start, commit, insert, delete, update
- Identifier of data item involved
- **Before image** (value) of data item
- **After image** of data item
- Pointers to previous and next operations in a transaction

Sample log file



Tid	Time	Operation	Object	Before image	After image	pPtr	nPtr
T1	10:12	START				0	2
T1	10:13	UPDATE	STAFF SL21	(old value)	(new value)	1	8
T2	10:14	START				0	4
T2	10:16	INSERT	STAFF SG37		(new value)	3	5
T2	10:17	DELETE	STAFF SA9	(old value)		4	6
T2	10:17	UPDATE	PROPERTY PG16	(old value)	(new value)	5	9
T3	10:18	START				0	11
T1	10:18	COMMIT				2	0
	10:19	CHECKPOINT	T2, T3				
T2	10:19	COMMIT				6	0
T3	10:20	INSERT	PROPERTY PG4		(new value)	7	12
T3	10:21	COMMIT				11	0

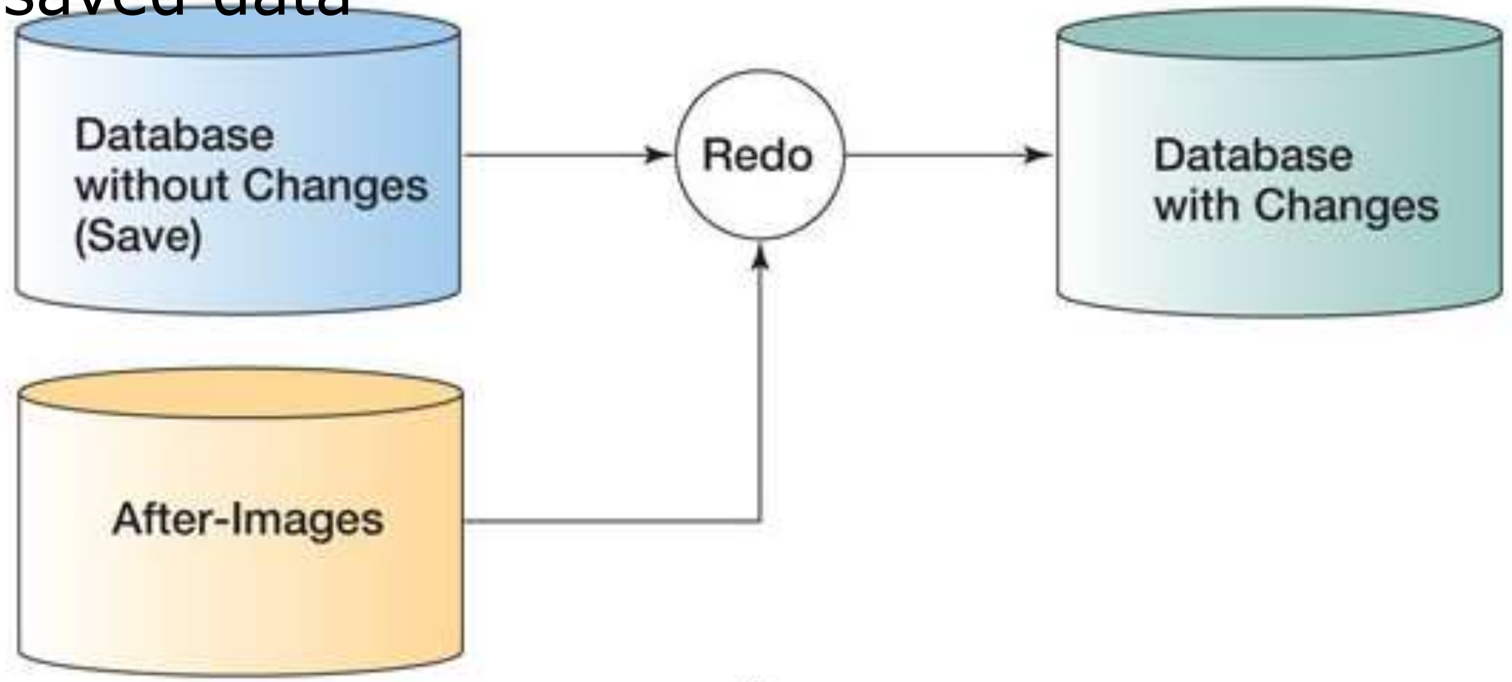
Image from Connolly & Begg

Rollback/Rollforward

- Recovery via rollback/rollforward:
  - Periodically save the database at a check point and keep the the log since the last save
  - Database log contains records of the changes to the data in chronological order
- The before and after images of the log are used to rollback or roll forward the database when a failure occurs

Rollforward

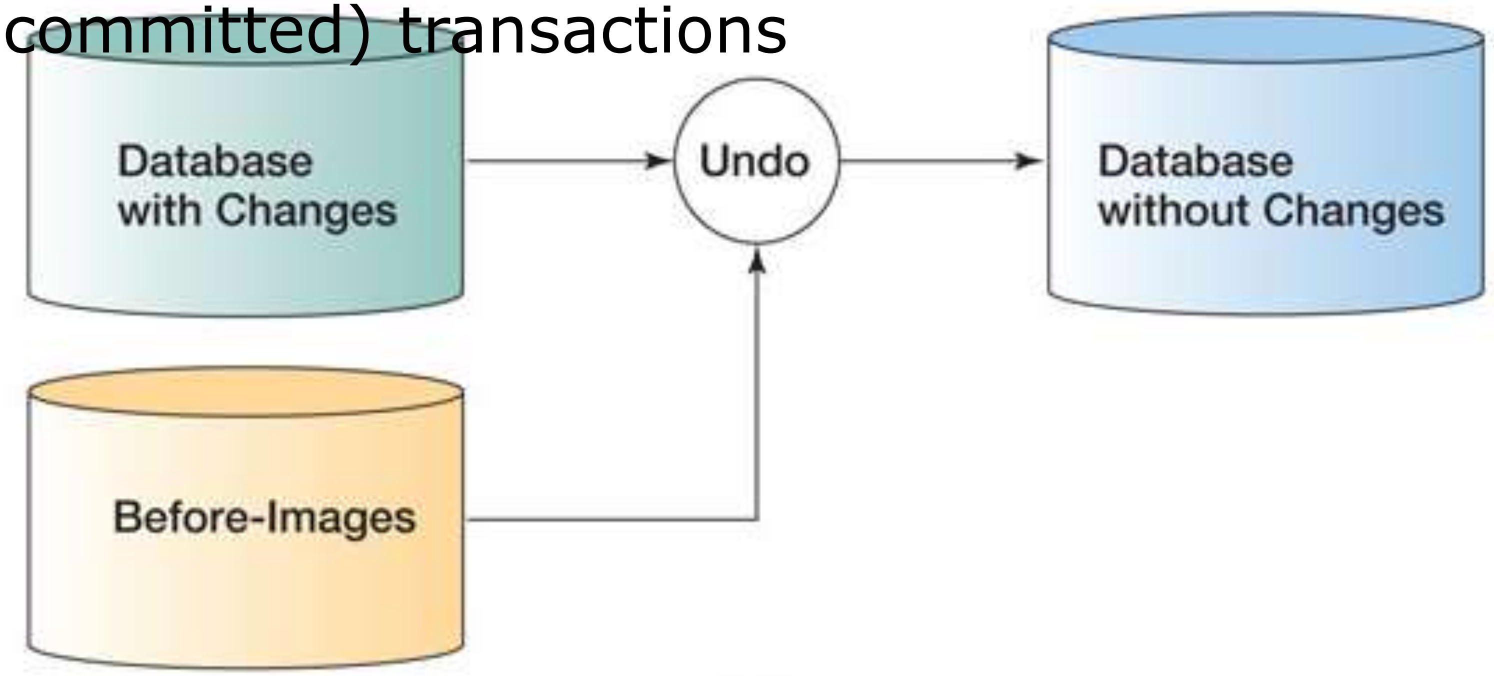
Recovers a database by **reapplying** valid transactions recorded in the log and using saved data



After-image: a copy of every database record (or page) after it was changed

Rollback

Recovers a database by **undoing** the erroneous modifications done to the database (by uncommitted transactions) and then **redoing** valid (committed) transactions



Before-image: a copy of every database record (or page) before it was changed

Checkpoint

- A **checkpoint** is a save point of where the database and transaction log are guaranteed to be synchronized
  - DBMS stops processing new transactions, completes processing outstanding transactions, and make sure all buffers are written out to disk
  - Once all of the writing is successfully completed → the log and the database are synchronized

Advantages-

- Checkpoints accelerate database recovery process
  - Since database can be restored using after-images since the last checkpoint
  - Checkpoint can be done several times per hour

The takeaways…

- If the system fails, the database must be returned to a consistent state
- Usually the DBMS handles this by maintaining a **log** of transactions that have been applied to the database
- Recovery is done by **rollback/rollforward** - *rolling back* transactions *uncommitted* at the point of failure, and *rolling forward committed* ones
- **Checkpoints** are a point of synchronisation between the log and the database, and are used to reduce the amount work required to recover the database

Topic learning outcomes revisited

- **After completing this topic you should be able to:**
- Explain what is meant by a transaction, and the possible transaction outcomes of commit and rollback
- Use the SQL commands COMMIT and ROLLBACK
- Define the desirable transaction properties of atomicity, consistency, isolation and durability (ACID)
- Give examples of the problems (including lost update) that can occur during concurrent transaction execution
- Explain what is meant by serial, nonserial and serialisable schedules, and give examples of each
- Describe the use of locking for concurrency management, particularly the two-phase locking protocol
- /

Topic learning outcomes revisited

- /
- Explain how deadlock may arise, and describe how deadlock may be handled by prevention or detection
- Explain the effect of data item granularity in concurrency control
- Explain the need for database recovery management
- Describe how the transaction log file is used in database recovery management

What’s next?

- In the next topic we'll explore some further concepts to do with database implementation, including distributed database architectures, and database programming using embedded SQL in Oracle.