## Scheduler

42

- □ The scheduler is the module that schedules the transaction's actions, ensuring serializability.
- □ How ?
  - Locks
  - Time stamps

42

# Locking Scheduler

Simple idea:

- Each element has a unique lock
- □ Each transaction must first acquire the lock before reading/writing that element
- $\hfill\Box$  If the lock is taken by another transaction, then wait
- $\hfill\Box$  The transaction must release the lock(s)

Adapted from M. Balazinska

# Lock Based Concurrency Control

43

- □ DBMS aims to allow only recoverable and serializable schedules
- □ Ensure committed transactions are not un-done while aborting transactions
- □ Use a *locking protocol*: a set of rules to be followed by each transaction to ensure serializable schedules
- Lock: a mechanism to control concurrent access to a data object

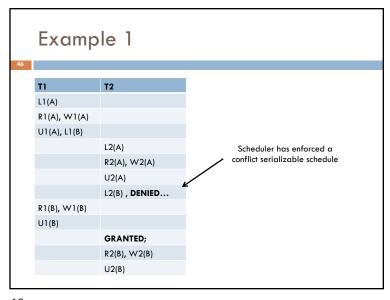
43

## **Notation**

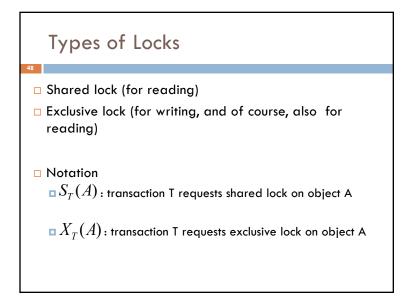
45

- □ Li(A) = transaction Ti acquires lock for element A
- □ Ui(A) = transaction Ti releases lock for element A

44 45

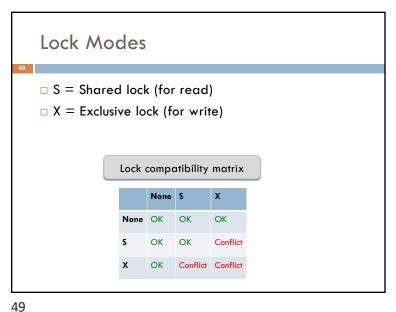


46



T1 T2
L1(A)
R1(A), W1(A)
U1(A)
L2(A)
R2(A), W2(A)
U2(A)
L2(B)
R2(B), W2(B)
U2(B)
L1(B)
R1(B), W1(B)
U1(B)

47



48

Z

# Strict Two Phase Locking (Strict 2PL)

- Most widely used locking protocol
- □ Two rules:
  - Each Xact must obtain a S (shared) lock on object before reading, and an X (exclusive) lock on object before writing.
- All locks held by a transaction are released when the transaction completes
- If an Xact holds an X lock on an object, no other Xact can get a lock (S or X) on that object.
- Strict 2PL allows only schedules whose precedence graph is acyclic (i.e., serializable)
- □ Recoverable and ACA

50

## **Implications**

- □ The locking protocol only allows safe interleavings of transactions
- □ If T1 and T2 access different data objects, then no conflict and each may proceed
- Otherwise, if same object, actions are ordered serially.
  - The Xact who gets the lock first must complete before the other can proceed

### 

R(A), W(A)

R(B), W(B)U(A), U(B)

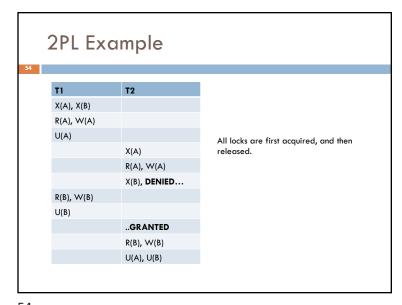
51

53

## Two Phase Locking Protocol (2PL)

- Variant of Strict 2PL
- □ Relaxes the 2<sup>nd</sup> rule of Strict 2PL to allow Xacts to release locks before the end (commit/abort)
- Two rules:
  - Each Xact must obtain a S (shared) lock on object before reading, and an X (exclusive) lock on object before writing.
  - A transaction cannot request additional locks once it releases any lock.
  - If an Xact holds an X lock on an object, no other Xact can get a lock (S or X) on that object.

52



54

56

# Strict 2PL makes 2PL "strict" Recall: a strict schedule is one where a value written by T is not read/overwritten until T commits/aborts Strict 2PL makes T hold locks until commit/abort No other transaction can see or modify the data object until T is complete

**2PL Implications** 

 In every transaction, all lock requests must precede all unlock requests.

□ This ensures conflict serializability

Why? (Think of order Xacts enter their shrinking phase)

■ This induces a sort ordering of the transactions that can be serialized

55

57

## Remarks

What if a transaction releases its locks and then aborts?

□ Recall: conflict serializable definition only considers committed transactions

## **Phantom Problem**

- So far we have assumed the database to be a static collection of elements (=tuples)
- □ If tuples are inserted/deleted then the *phantom* problem appears

58

60

# Phantom Problem

T1 T2

SELECT \*
FROM Product
WHERE color='blue'

INSERT INTO Product(name, color) VALUES ('gizmo', 'blue')

SELECT \*
FROM Product
WHERE color='blue'

Suppose there are two blue products, X1, X2:

R1(X1),R1(X2),W2(X3),R1(X1),R1(X2),R1(X3)

This is conflict serializable! What's wrong??

Phantom Problem

T1

T2

SELECT \*
FROM Product
WHERE color='blue'

INSERT INTO Product(name, color) VALUES ('gizmo', 'blue')

SELECT \*
FROM Product
WHERE color='blue'

Is this schedule serializable?

59

## Phantom Problem

T1

61

T2

SELECT \*
FROM Product
WHERE color='blue'

INSERT INTO Product(name, color) VALUES ('gizmo', 'blue')

SELECT \*
FROM Product
WHERE color='blue'

Suppose there are two blue products, X1, X2:

R1(X1),R1(X2),W2(X3),R1(X1),R1(X2),R1(X3)

## Phantom Problem

62

□ A "phantom" is a tuple that is invisible during part of a transaction execution but not all of it.

□ In our example:

- T1: reads list of products
- □ T2: inserts a new product
- □ T1: re-reads: a new product appears!

62

# **Dealing With Phantoms**

64

- $\hfill\Box$  Lock the entire table, or
- □ Lock the index entry for 'blue'
  - □ If index is available

Dealing with phantoms is expensive!

**Phantom Problem** 

63

- □ In a static database:
  - Conflict serializability implies serializability
- □ In a *dynamic* database, this may fail due to phantoms