

ICS 321 Fall 2012

# Transaction Processing (ii)

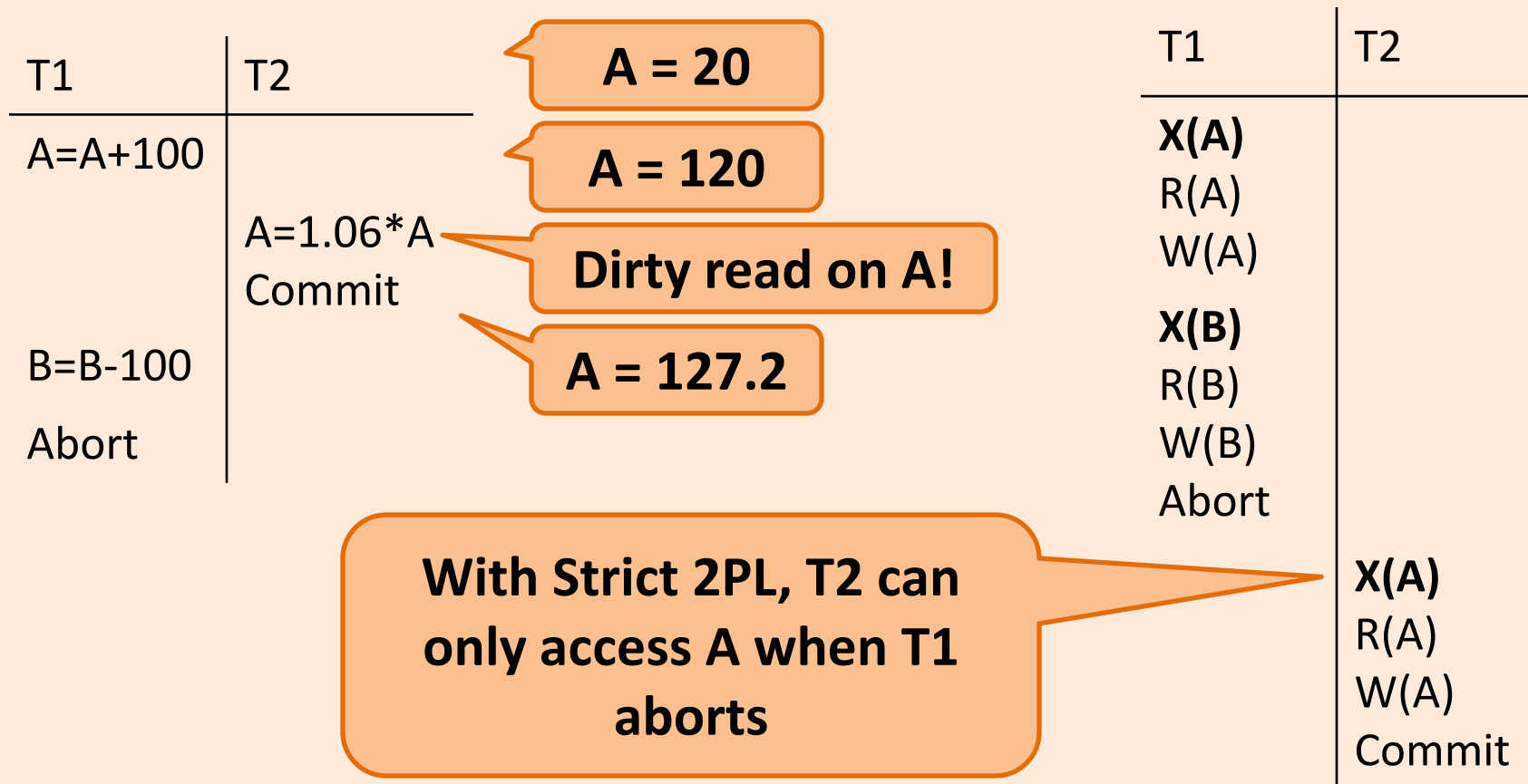
Asst. Prof. Lipyeow Lim  
Information & Computer Science Department  
University of Hawaii at Manoa

# Lock-based Concurrency Control

- Strict Two-phase Locking (Strict 2PL) Protocol:
  - 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
    - *(Non-strict) 2PL Variant:* Release locks anytime, but cannot acquire locks after releasing 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.
- Strict 2PL allows only serializable schedules.
  - Additionally, it simplifies transaction aborts
  - *(Non-strict) 2PL* also allows only serializable schedules, but involves more complex abort processing

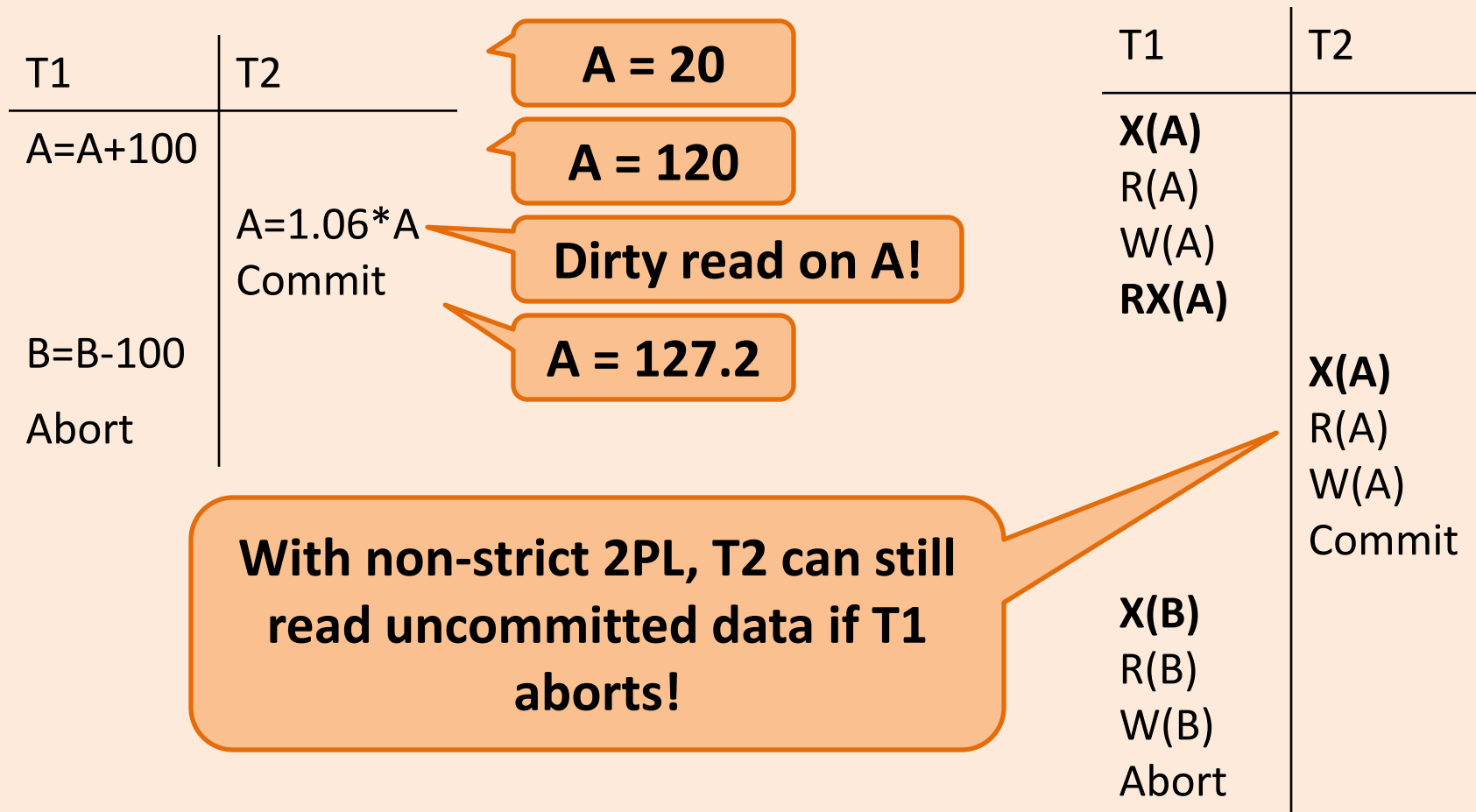
# Example (Strict 2PL)

- Consider the dirty read schedule



# Example (Non-Strict 2PL)

- Consider the dirty read schedule



# Deadlocks

- Cycle of transactions waiting for locks to be released
- DBMS has to either prevent or resolve deadlocks
- Common approach:
  - Detect via timeout
  - Resolve by aborting transactions

T1	T2
Req X(A)	Req X(B)
Gets X(A)	Gets X(B)
...	....
Req X(B)	Req X(A)

# Aborting a Transaction

- If a transaction  $T1$  is aborted, all its actions have to be undone.
  - Not only that, if  $T2$  reads an object last written by  $T1$ ,  $T2$  must be aborted as well!
- Most systems avoid such *cascading aborts* by releasing a transaction's locks only at commit time.
  - If  $T1$  writes an object,  $T2$  can read this only after  $T1$  commits.
- In order to *undo* the actions of an aborted transaction, the DBMS maintains a *log* in which every write is recorded.
  - This mechanism is also used to recover from system crashes: all active Xacts at the time of the crash are aborted when the system comes back up

# Lock Granularity

- What should the DBMS lock ?
  - Row ?
  - Page ?
  - A Table ?

```
UPDATE Sailors
SET      rating=0
WHERE    rating>9
```

```
SELECT *
FROM    Sailors
```

```
SELECT *
FROM    Sailors
WHERE   rating < 2
```

```
UPDATE Boats
SET      color='red'
WHERE    bid=13
```

```
UPDATE Boats
SET      color='blue'
WHERE    bid=100
```

# Crash Recovery

- **Transaction Manager:** DBMS component that controls execution (eg. managing locks).
- **Recovery Manager:** DBMS component for ensuring
  - Atomicity: undo actions of transactions that do not commit
  - Durability: committed transactions survive system crashed and media failures
- Assume atomic writes to disk.



# The Log

- The following actions are recorded in the log:
  - *Ti writes an object*: the old value and the new value.
    - Log record must go to disk before the changed page! (Write Ahead Log property)
  - *Ti commits/aborts*: a log record indicating this action.
- Log records are chained together by Xact id, so it's easy to undo a specific Xact.
- Log is often *duplexed* and *archived* on stable storage.
- All log related activities (and in fact, all CC related activities such as lock/unlock, dealing with deadlocks etc.) are handled transparently by the DBMS.

# Recovering from a Crash

- There are 3 phases in the *Aries* recovery algorithm:
  - Analysis: Scan the log forward (from the most recent *checkpoint*) to identify all Xacts that were active, and all dirty pages in the buffer pool at the time of the crash.
  - Redo: Redoes all updates to dirty pages in the buffer pool, as needed, to ensure that all logged updates are in fact carried out and written to disk.
  - Undo: The writes of all Xacts that were active at the crash are undone (by restoring the *before value* of the update, which is in the log record for the update), working backwards in the log. (Some care must be taken to handle the case of a crash occurring during the recovery process!)