

CompSci 516  
Database Systems

Lecture 14-15  
Transactions  
– Concurrency Control

Instructor: Sudeepa Roy

## Announcements

- HW2 deadline - **Saturday, 10/21, 5 pm**
  - submit on time
- Project midterm report deadline - **Wednesday, 11/01, 11:55 pm**
  - Keep working on your proposed project

## Reading Material

- [RG]
  - Chapter 17.5.1, 17.5.3, 17.6
- [GUW]
  - Chapter 18.8, 18.9
  - Today's examples are from GUW (lecture slides will be sufficient for this class and exams)

Acknowledgement:  
The following slides have been created adapting the instructor material of the [RG] book provided by the authors Dr. Ramakrishnan and Dr. Gehrke.

## Today's topics

- Dynamic databases and Phantom problem (17.5.1)
- Multiple—granularity locking (17.5.3)
- Optimistic concurrency control (17.6.1)
- Timestamp-based concurrency control (17.6.2)
- Multi-version concurrency control (17.6.3)

## Dynamic Database and Phantom Problem

## Dynamic Databases

- If we relax the assumption that the DB is a fixed collection of objects
- Then even Strict 2PL will not assure serializability
- causes "Phantom Problem" in dynamic databases

### Example: Phantom Problem

- T1 wants to find oldest sailors in rating levels 1 and 2
  - Suppose the oldest at rating 1 has age 71
  - Suppose the oldest at rating 2 has age 80
  - Suppose the second oldest at rating 2 has age 63
- Another transaction T2 intervenes:
  - **Step 1:** T1 locks all pages containing sailor records with rating = 1, and finds oldest sailor (age = 71)
  - **Step 2:** Next, T2 inserts a new sailor onto a new page (rating = 1, age = 96)
  - **Step 3:** T2 locks pages with rating = 2, deletes oldest sailor with rating = 2 (age = 80), commits, releases all locks
  - **Step 4:** T1 now locks all pages with rating = 2, and finds oldest sailor (age = 63)
- No consistent DB state where T1 is "correct"
  - T1 found oldest sailor with rating = 1 before modification by T2
  - T1 found oldest sailor with rating = 2 after modification by T2

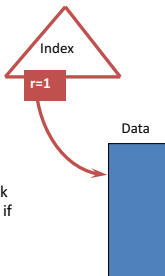
Duke CS, Fall 2017 CompSci 516: Database Systems 7

### What was the problem?

- Conflict serializability guarantees serializability only if the set of objects is fixed
- Problem:
  - T1 implicitly assumed that it has locked the set of all sailor records with rating = 1
  - Assumption only holds if no sailor records are added while T1 is executing
  - Need some mechanism to enforce this assumption
- Index locking and predicate locking

Duke CS, Fall 2017 CompSci 516: Database Systems 8

### Index Locking



- If there is a dense index on the rating field using Alt. (2), T1 should lock the index page containing the data entries with rating = 1
  - If there are no records with rating = 1, T1 must lock the index page where such a data entry would be, if it existed
- If there is no suitable index, T1 must lock all pages, and lock the file/table to prevent new pages from being added
  - to ensure that no new records with rating = 1 are added

Duke CS, Fall 2017 CompSci 516: Database Systems 9

### Predicate Locking

- Grant lock on all records that satisfy some logical predicate, e.g. rating = 1 or, age > 2\*salary
- Index locking is a special case and an efficient implementation of predicate locking
  - e.g. Lock on the index pages for records satisfying rating = 1
- The general predicate locking has a lot of locking overhead and so not commonly used

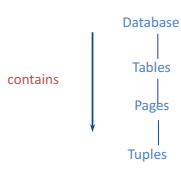
Duke CS, Fall 2017 CompSci 516: Database Systems 10

### Multiple-granularity Locking

Duke CS, Fall 2017 CompSci 516: Database Systems 11

### DB Objects may contain other objects

- A DB contains several files
- A file is a collection of pages
- A page is a collection of records/tuples



Duke CS, Fall 2017 CompSci 516: Database Systems 12

## Carefully choose lock granularity

- If a transaction needs most of the pages
  - set a lock on the entire file
  - reduces locking overhead
- If only a few pages are needed
  - lock only those pages
- Need to efficiently ensure no conflicts
  - e.g. a page should not be locked by T1 if T2 already holds the lock on the file

Duke CS, Fall 2017      CompSci 516: Database Systems      13

## New Lock Modes & Protocol

- Allow transactions to lock at each level, but with a special protocol using new “intention locks”:
- Before locking an item (S or X), transaction must set “intention locks” (IS or IX) on all its ancestors
- For unlock, go from specific to general (i.e., bottom-up)
  - otherwise conflicting lock possible at root

		conflicting locks				
	--	IS	IX	S	X	
--		✓	✓	✓	✓	✓
IS	✓		✓	✓	✓	✗
IX	✓	✓		✓	✓	✗
S	✓	✓	✓		✓	✗
X	✓	✓	✓	✓		✗

other tr. cannot have IX or X  
other tr. cannot have any other lock

Duke CS, Fall 2017      CompSci 516: Database Systems      14

## SIX mode = S + IX

- Common situation: a transaction needs to read an entire file and modify a few records
  - S lock
  - IX lock (to subsequently lock some containing objects in X mode)
- Obtain a SIX lock
  - conflict with either S or IX

		conflicting locks				
	--	IS	IX	S	X	
--		✓	✓	✓	✓	✓
IS	✓		✓	✓	✓	✗
IX	✓	✓		✓	✓	✗
S	✓	✓	✓		✓	✗
X	✓	✓	✓	✓		✗

other tr. cannot have IX or X  
other tr. cannot have any other lock

Duke CS, Fall 2017      CompSci 516: Database Systems      15

## Transaction in SQL

- SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED [;]
- BEGIN TRANSACTION
- <... SQL STATEMENTS>
- COMMIT or ROLLBACK
- Four isolation levels : performance and serializability

	Dirty Read	Unrepeatable Read	Phantom
READ UNCOMMITTED	Maybe	Maybe	Maybe
READ COMMITTED	No	Maybe	Maybe
REPEATABLE READS	No	No	Maybe
SERIALIZABLE	No	No	No

Duke CS, Fall 2017      CompSci 516: Database Systems      16

## Approaches to CC other than locking

Duke CS, Fall 2017      CompSci 516: Database Systems      17

## Approaches to Concurrency Control (CC)

- Lock-based CC
  - (so far)
- Optimistic CC
  - today
- Time-stamp-based CC
  - today
- Multi-version CC
  - today

uses “timestamps” in some way

Duke CS, Fall 2017      CompSci 516: Database Systems      18

## Timestamp

- Each transaction gets a unique timestamp
- e.g.
  - system’s clock value when it is issued by the scheduler (assume one transactions issued on one tick of the clock)
  - or a unique number given by a counter (incremented after each transaction)

Duke CS, Fall 2017

CompSci 516: Database Systems

19

## Locking is a “pessimistic or conservative” approach to CC

- Locking is a conservative approach in which **conflicts are prevented**
- Either uses “blocking” (delay) or abort
  - note the several usages of a “block”!
- Disadvantages of locking:
  - Lock management overhead
  - Deadlock detection/resolution
  - Lock contention for heavily used objects
- If only light contention for data objects, still the overhead of following a locking protocol is paid

Duke CS, Fall 2017

CompSci 516: Database Systems

20

## Optimistic CC

Duke CS, Fall 2017

CompSci 516: Database Systems

21

## A second approach to CC: Optimistic CC (Kung-Robinson)

- If conflicts are rare, we might be able to gain **concurrency by not locking**, and instead **checking for conflicts before transactions commit**
- Premise:
  - most transactions do not conflict with other transactions
  - be as permissive as possible in allowing transactions to execute

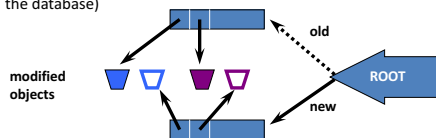
Duke CS, Fall 2017

CompSci 516: Database Systems

22

## Kung-Robinson Model

- Transactions have three phases:
  1. **READ**: Read from the database, but make changes to “private copies” of objects (assume private workspace)
  2. **VALIDATE**: When decide to commit, also check for conflicts with concurrently executing transactions
    - if a possible conflict, abort, clear private workspace, restart
  3. **WRITE**: If no conflict, make local copies of changes public (copy them into the database)



Duke CS, Fall 2017

CompSci 516: Database Systems

23

## Validation

- Test conditions that are **sufficient** to ensure that no conflict occurred
- Each transaction  $T_i$  is assigned a numeric id
  - Use a **timestamp  $TS(T_i)$**
- Transaction ids assigned **at end of READ phase, just before validation begins**
- Validation checks whether the timestamp ordering has an equivalent serial order

Duke CS, Fall 2017

CompSci 516: Database Systems

24

### Notation

- $TS(T_i)$ : Transaction id or timestamp of  $T_i$   
BEFORE the validation step starts
- $ReadSet(T_i)$ : Set of objects read by transaction  $T_i$
- $WriteSet(T_i)$ : Set of objects modified by transaction  $T_i$

next, three tests used for validation

Duke CS, Fall 2017      CompSci 516: Database Systems      25

### Validation Tests

- To validate  $T_j$ 
  - for each committed transactions  $T_i$
  - such that  $TS(T_i) < TS(T_j)$
  - one of the three validation tests (TEST 1, TEST 2, TEST 3) must be satisfied
  - (see the tests next)
- Ensures that  $T_j$ 's modifications are not visible to the previous transaction  $T_i$
- Check yourself: No RW, WR, WW conflicts if any of these tests satisfy

Duke CS, Fall 2017      CompSci 516: Database Systems      26

### Test 1

- For all  $i$  and  $j$  such that  $TS(T_i) < TS(T_j)$ , check that  $T_i$  completes (all three phases) before  $T_j$  begins

Duke CS, Fall 2017      CompSci 516: Database Systems      27

### Test 2

figure not exactly correct since V and W steps cannot overlap

- For all  $i$  and  $j$  such that  $TS(T_i) < TS(T_j)$ , check that:
  - $T_i$  completes before  $T_j$  begins its Write phase +
  - $WriteSet(T_i) \cap ReadSet(T_j)$  is empty

Does  $T_j$  read dirty data? Does  $T_i$  overwrite  $T_j$ 's writes?

- Allows  $T_j$  to read objects while  $T_i$  is still modifying objects
- But no conflict because  $T_j$  does not read any object modified by  $T_i$
- $T_j$  can overwrite some writes by  $T_i$  (ok since  $T_j$  starts later)

Duke CS, Fall 2017      CompSci 516: Database Systems      28

### Test 3

figure not exactly correct since V and W steps cannot overlap

- For all  $i$  and  $j$  such that  $T_i < T_j$ , check that:
  - $T_i$  completes Read phase before  $T_j$  completes its Read +
  - $WriteSet(T_i) \cap ReadSet(T_j)$  is empty +
  - $WriteSet(T_i) \cap WriteSet(T_j)$  is empty

i.e.  $T_j$  does not write any object that  $T_i$  reads or writes

Does  $T_j$  read dirty data? Does  $T_i$  overwrite  $T_j$ 's writes?

- Allows  $T_i$  and  $T_j$  write objects at the same time
- More overlap than Test 2
- But the sets of objects written cannot overlap

Duke CS, Fall 2017      CompSci 516: Database Systems      29

### Comments on Serial Validation

- List of objects written/read by each transaction has to be maintained
- While one transaction is validating, no transaction can commit
  - otherwise some conflicts may be missed
- Assignment of transaction id, validation, and the Write phase are inside a critical section
  - i.e., Nothing else goes on concurrently
  - If Write phase is long, major drawback
- The write phase of a validated transactions must be completed before other tr. s are validated
  - i.e. changes should be reflected to the DB from private workspace
- Optimization for Read-only transactions:
  - Don't need critical section (because there is no Write phase)

Duke CS, Fall 2017      CompSci 516: Database Systems      30

## Overheads in Optimistic CC

- Must record read/write activity in ReadSet and WriteSet per transaction
  - Must create and destroy these sets as needed
- Must check for conflicts during validation, and must make validated writes “global”
  - Critical section can reduce concurrency
- Optimistic CC restarts transactions that fail validation
  - Work done so far is wasted; requires clean-up

Duke CS, Fall 2017

CompSci 516: Database Systems

31

## Optimistic CC vs locking

End of Lecture 14

- If there are few conflicts and validation is efficient
  - optimistic CC is better than locking
- If many conflicts
  - cost of repeatedly restarting transactions hurts performance significantly

Duke CS, Fall 2017

CompSci 516: Database Systems

32

Start of Lecture 15

## Timestamp-based CC

Duke CS, Fall 2017

CompSci 516: Database Systems

33

## A third approach to CC

So far...

- Lock-based CC
  - conflicting actions of different transactions are ordered by the order in which locks are obtained
  - locking protocols ensure serializability
- Optimistic CC
  - A timestamp ordering is imposed on transactions
  - Validation checks that all conflicting transactions occurred in the same order
- Next: Timestamp-based CC
  - another use of timestamp

Duke CS, Fall 2017

CompSci 516: Database Systems

34

## Timestamp CC

Main Idea:

- Give each object  $O$ 
  - a read-timestamp  $RT(O)$ , and
  - a write-timestamp  $WT(O)$ 
    - RG uses  $RTS/WTS$ , GUV uses  $RT/WT$ , either of these is fine
- Give each transaction  $T$ 
  - a timestamp  $TS(T)$  when it begins
- If
  - action  $a_i$  of  $T_i$  conflicts with action  $a_j$  of  $T_j$ ,
  - and  $TS(T_i) < TS(T_j)$
- then
  - $a_i$  must occur before  $a_j$
- Otherwise, abort and restart violating transaction

Duke CS, Fall 2017

CompSci 516: Database Systems

35

## Request for a read: $R_T(X)$

1. If  $TS(T) \geq WT(X)$ 
  - last written by a previous transaction — OK (i.e. “physically realizable”)
  - If  $C(X)$  is true — check if previous transaction has committed
    - Grant the read request by  $T$ 
      - if  $TS(T) > RT(X)$ 
        - set  $RT(X) = TS(T)$
    - If  $C(X)$  is false
      - Delay  $T$  until  $C(X)$  becomes true, or the transaction that wrote  $X$  aborts
2. If  $TS(T) < WT(X)$ 
  - write is not realizable — already written by a later trans.
  - Abort (or, Rollback)  $T$  — i.e. abort and restart with a larger timestamp

Duke CS, Fall 2017

CompSci 516: Database Systems

36

### Request for a write: $W_T(X)$

- If  $TS(T) \geq RT(X)$  and  $TS(T) \geq WT(X)$ 
  - last written/read by a previous transaction – OK
  - Grant the write request by T
    - write the new value of X
    - Set  $WT(X) = TS(T)$
    - Set  $C(X) = \text{false}$  -- T not committed yet
- If  $TS(T) \geq RT(X)$  but  $TS(T) < WT(X)$ 
  - write is still realizable -- but already a later value in X
  - If  $C(X)$  is true
    - previous writer of X has committed
    - simply ignore the write request by T
    - but allow T to proceed without making changes to the database
  - If  $C(X)$  is false
    - Delay T until  $C(X)$  becomes true, or the transaction that wrote X aborts

- If  $TS(T) < RT(X)$ 
  - write is not realizable -- already read by a later transaction
  - Abort (or, Rollback) T

Duke CS, Fall 2017      CompSci 516: Database Systems      37

### Thomas Write Rule

- If  $TS(T) < WT(O)$  and a write request comes
  - violates timestamp order of T w.r.t. writer of O

Thomas Write Rule:

- But we can safely ignore such outdated writes
- no need to restart T
- T's write is effectively followed by another write, with no intervening reads
- Allows some serializable, but NOT conflict serializable schedules

Duke CS, Fall 2017      CompSci 516: Database Systems      38

### Timestamp CC and Recoverability

Without "block or delay", unrecoverable schedules are allowed:

- $TS(T_1) = 1$
- $TS(T_2) = 2$

T1 (1)	T2 (2)
W(A); WT(A)=1	R(A): RT(A)=2 W(B): WT(B)=2 Commit

- Timestamp CC with "delays" allows only recoverable schedules:
  - "Block" readers T (where  $TS(T) > WT(O)$ ) until writer of O commits
  - a full example from GUW next
- Similar to writers holding X locks until commit, but still not quite 2PL

Duke CS, Fall 2017      CompSci 516: Database Systems      39

### Example

- Three transactions T1 ( $TS = 200$ ), T2 ( $TS = 150$ ), T3 ( $TS = 175$ )
- Three objects A, B, C
  - initially all have  $RT = WT = 0, C = 1$  (i.e. true)
- Sequence of actions
  - $R_1(B), R_2(A), R_3(C), W_1(B), W_1(A), W_2(C), W_3(A)$
- Q. What is the state of the database at the end if the timestamp-based CC protocol is followed
  - i.e. report the RT, WT, C

Duke CS, Fall 2017      CompSci 516: Database Systems      40

### Initial condition and Steps

Step	T1	T2	T3	A	B	C
	200	150	175	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1
1	$R_1(B)$					
2		$R_2(A)$				
3			$R_3(C)$			
4	$W_1(B)$					
5	$W_1(A)$					
6		$W_2(C)$				
7			$W_3(A)$			

Duke CS, Fall 2017      CompSci 516: Database Systems      41

### After Step 1

WT of B is  $\leq TS(T_1)$   
C = 1  
Read OK.

Step	T1	T2	T3	A	B	C
	200	150	175	RT = 0, WT = 0, C = 1	RT = 200, WT = 0, C = 1	RT = 0, WT = 0, C = 1
1	$R_1(B)$				RT=200	
2		$R_2(A)$				
3			$R_3(C)$			
4	$W_1(B)$					
5	$W_1(A)$					
6		$W_2(C)$				
7			$W_3(A)$			

Duke CS, Fall 2017      CompSci 516: Database Systems      42

### After Step 2

WT of A is  $\leq$  TS(T<sub>2</sub>)  
C = 1  
Read OK.

Step	T1	T2	T3	A	B	C
	200	150	175	RT = 150, WT = 0, C = 1	RT = 200, WT = 0, C = 1	RT = 0, WT = 0, C = 1
1	R <sub>1</sub> (B)				RT=200	
2		R <sub>2</sub> (A)		RT=150		
3			R <sub>3</sub> (C)			
4	W <sub>1</sub> (B)					
5	W <sub>1</sub> (A)					
6		W <sub>2</sub> (C)				
7			W <sub>3</sub> (A)			

Duke CS, Fall 2017

CompSci 516: Database Systems

43

### After Step 3

WT of C is  $\leq$  TS(T<sub>3</sub>)  
C = 1  
Read OK.

Step	T1	T2	T3	A	B	C
	200	150	175	RT = 150, WT = 0, C = 1	RT = 200, WT = 0, C = 1	RT = 175, WT = 0, C = 1
1	R <sub>1</sub> (B)				RT=200	
2		R <sub>2</sub> (A)		RT=150		
3			R <sub>3</sub> (C)			RT=175
4	W <sub>1</sub> (B)					
5	W <sub>1</sub> (A)					
6		W <sub>2</sub> (C)				
7			W <sub>3</sub> (A)			

Duke CS, Fall 2017

CompSci 516: Database Systems

44

### After Step 4

WT & RT of B is  $\leq$  TS(T<sub>2</sub>)  
Write OK.

Step	T1	T2	T3	A	B	C
	200	150	175	RT = 150, WT = 0, C = 1	RT = 200, WT = 200, C = 0	RT = 175, WT = 0, C = 1
1	R <sub>1</sub> (B)				RT=200	
2		R <sub>2</sub> (A)		RT=150		
3			R <sub>3</sub> (C)			RT=175
4	W <sub>1</sub> (B)				WT=200 C=0	
5	W <sub>1</sub> (A)					
6		W <sub>2</sub> (C)				
7			W <sub>3</sub> (A)			

Duke CS, Fall 2017

CompSci 516: Database Systems

45

### After Step 5

RT & WT of A  $\leq$  TS(T<sub>2</sub>)  
Write ok.

Step	T1	T2	T3	A	B	C
	200	150	175	RT = 150 WT = 200 C = 0	RT = 200 WT = 200 C = 0	RT = 175 WT = 0 C = 1
1	R <sub>1</sub> (B)				RT=200	
2		R <sub>2</sub> (A)		RT=150		
3			R <sub>3</sub> (C)			RT=175
4	W <sub>1</sub> (B)				WT=200 C=0	
5	W <sub>1</sub> (A)			WT=200 C=0		
6		W <sub>2</sub> (C)				
7			W <sub>3</sub> (A)			

Duke CS, Fall 2017

CompSci 516: Database Systems

46

### After Step 6

RT(C) = 175 < 150 = TS(T<sub>2</sub>)  
Abort T<sub>2</sub>

Step	T1	T2	T3	A	B	C
	200	150	175	RT = 150 WT = 200 C = 0	RT = 200 WT = 200 C = 0	RT = 175 WT = 0 C = 1
1	R <sub>1</sub> (B)				RT=200	
2		R <sub>2</sub> (A)		RT=150		
3			R <sub>3</sub> (C)			RT=175
4	W <sub>1</sub> (B)				WT=200 C=0	
5	W <sub>1</sub> (A)			WT=200 C=0		
6		W <sub>2</sub> (C) Abort				
7			W <sub>3</sub> (A)			

Duke CS, Fall 2017

CompSci 516: Database Systems

47

### After Step 7

RT(A)  $\leq$  TS(T<sub>3</sub>) -- write ok  
WT(A) > TS(T<sub>3</sub>) and C(A) = 0  
Delay T<sub>3</sub>

Step	T1	T2	T3	A	B	C
	200	150	175	RT = 150 WT = 200 C = 0	RT = 200 WT = 200 C = 0	RT = 175 WT = 0 C = 1
1	R <sub>1</sub> (B)				RT=200	
2		R <sub>2</sub> (A)		RT=150		
3			R <sub>3</sub> (C)			RT=175
4	W <sub>1</sub> (B)				WT=200 C=0	
5	W <sub>1</sub> (A)			WT=200 C=0		
6		W <sub>2</sub> (C) Abort				
7			W <sub>3</sub> (A) Delay			

Duke CS, Fall 2017

CompSci 516: Database Systems

48



## Multiversion CC

Duke CS, Fall 2017      CompSci 516: Database Systems      49

## A fourth approach to CC

- **Multiversion CC**
  - another way of using timestamps
  - ensures that a transaction never has to be restarted (aborted) to read an object
    - unlike timestamp-based CC
- **The idea is to make several copies of each DB object**
  - each copy of each object has a **write timestamp**
- **T<sub>i</sub> reads the most recent version whose timestamp precedes TS(T<sub>i</sub>)**

Duke CS, Fall 2017      CompSci 516: Database Systems      50

## Multiversion Timestamp CC

- **Idea: Let writers make a “new” copy while readers use an appropriate “old” copy:**

MAIN SEGMENT  
(Current versions of DB objects)

VERSION POOL  
(Older versions that may be useful for some active readers.)

Readers are always allowed to proceed

- But may be “blocked” until writer commits.

Duke CS, Fall 2017      CompSci 516: Database Systems      51

## Multiversion CC (Contd.)

- **Each version of an object has**
  - its writer’s TS as its **WT**, and
  - the timestamp of the transaction that most recently read this version as its **RT**
- **Versions are chained backward**
  - we can discard versions that are “too old to be of interest”
- **Each transaction is classified as Reader or Writer.**
  - Writer **may** write some object; Reader never will
  - Transaction declares whether it is a Reader when it begins

Duke CS, Fall 2017      CompSci 516: Database Systems      52

## Reader Transaction

- **For each object to be read:**
  - Finds **newest version** with  $WT < TS(T)$
  - Starts with current version in the main segment and chains backward through earlier versions
  - Update RT if necessary (i.e. if  $TS(T) > RT$ , then  $RT = TS(T)$ )
- **Assuming that some version of every object exists from the beginning of time, Reader transactions are never restarted**
  - However, might block until writer of the appropriate version commits

WTS timeline

Duke CS, Fall 2017      CompSci 516: Database Systems      53

## Writer Transaction

- **To read an object, follows reader protocol**
- **To write an object:**
  - must make sure that the object has not been read by a “later” transaction
  - Finds **newest version V** s.t.  $WT(V) <= TS(T)$ .
- **If  $RT(V) <= TS(T)$** 
  - T makes a copy **CV** of V, with a pointer to V, with  $WT(CV) = TS(T)$ ,  $RT(CV) = TS(T)$
  - Write is buffered until T commits; other transactions can see TS values but can’t read version CV
- **Else**
  - reject write

Duke CS, Fall 2017      CompSci 516: Database Systems      54

### Example

- Four transactions T1 (TS = 150), T2 (TS = 200), T3 (TS = 175), T4(TS = 225)
- One object A
  - Initial version is A<sub>0</sub>
- Sequence of actions
  - R<sub>1</sub>(A), W<sub>1</sub>(A), R<sub>2</sub>(A), W<sub>2</sub>(A), R<sub>3</sub>(A), R<sub>4</sub>(A)
- Q. What is the state of the database at the end if the multiversion CC protocol is followed

### Initial condition and Steps

A<sub>0</sub> existed before the transactions started

Step	T1	T2	T3	T4	A <sub>0</sub>		
	150	200	175	225	RT=0, WT=0		
1	R <sub>1</sub> (A)						
2	W <sub>1</sub> (A)						
3		R <sub>2</sub> (A)					
4		W <sub>2</sub> (A)					
5			R <sub>3</sub> (A)				
6				R <sub>4</sub> (A)			

### After Step 1

A<sub>0</sub> is the newest version with WT <= TS(T<sub>1</sub>)  
Read A<sub>0</sub>

Step	T1	T2	T3	T4	A <sub>0</sub>		
	150	200	175	225	RT=0, WT=0		
1	R <sub>1</sub> (A)				Read RT = 150		
2	W <sub>1</sub> (A)						
3		R <sub>2</sub> (A)					
4		W <sub>2</sub> (A)					
5			R <sub>3</sub> (A)				
6				R <sub>4</sub> (A)			

### After Step 2

- A<sub>0</sub> is the newest version with WT <= TS(T<sub>1</sub>)
- RT(A<sub>0</sub>) <= TS(T<sub>1</sub>)
- Create a new version A<sub>150</sub>
- Set its WT, RT to TS(T<sub>1</sub>) = 150 (A<sub>150</sub> named accordingly)

Step	T1	T2	T3	T4	A <sub>0</sub>	A <sub>150</sub>
	150	200	175	225	RT=150 WT=0	RT=150 WT=150
1	R <sub>1</sub> (A)				Read RT = 150	
2	W <sub>1</sub> (A)					Create RT=150 WT=150
3		R <sub>2</sub> (A)				
4		W <sub>2</sub> (A)				
5			R <sub>3</sub> (A)			
6				R <sub>4</sub> (A)		

### After Step 3

- A<sub>150</sub> is the newest version with WT <= TS(T<sub>2</sub>)
- Read A<sub>150</sub>
- Update RT

Step	T1	T2	T3	T4	A <sub>0</sub>	A <sub>150</sub>
	150	200	175	225	RT=150 WT=0	RT=200 WT=150
1	R <sub>1</sub> (A)				Read	
2	W <sub>1</sub> (A)					Create RT=150 WT=150
3		R <sub>2</sub> (A)				Read RT=200
4		W <sub>2</sub> (A)				
5			R <sub>3</sub> (A)			
6				R <sub>4</sub> (A)		

### After Step 4

- A<sub>150</sub> is the newest version with WT <= TS(T<sub>2</sub>)
- RT(A<sub>150</sub>) <= TS(T<sub>2</sub>)
- Create a new version A<sub>200</sub>
- Set its WT, RT to TS(T<sub>2</sub>) = 200 (A<sub>200</sub> named accordingly)

Step	T1	T2	T3	T4	A <sub>0</sub>	A <sub>150</sub>	A <sub>200</sub>
	150	200	175	225	RT=150 WT=0	RT=200 WT=150	RT=200 WT=200
1	R <sub>1</sub> (A)				Read		
2	W <sub>1</sub> (A)					Create RT=150 WT=150	
3		R <sub>2</sub> (A)				Read RT=200	
4		W <sub>2</sub> (A)					Create RT=200 WT=200
5			R <sub>3</sub> (A)				
6				R <sub>4</sub> (A)			

### After Step 5

- $A_{150}$  is the newest version with  $WT \leq TS(T_3)$
- Read  $A_{150}$
- DO NOT Update RT

Step	T1	T2	T3	T4	$A_0$	$A_{150}$	$A_{200}$
	150	200	175	225	RT=150 WT=0	RT=200 WT=150	RT=200 WT=200
1	$R_1(A)$				Read		
2	$W_1(A)$					Create RT=150 WT=150	
3		$R_2(A)$				Read RT=200	
4		$W_2(A)$					Create RT=200 WT=200
5			$R_3(A)$			Read	
6				$R_4(A)$			

Duke CS, Fall 2017 Compsci 516: Database Systems 61

### After Step 6

- $A_{200}$  is the newest version with  $WT \leq TS(T_4)$
- Read  $A_{200}$
- Update RT

Step	T1	T2	T3	T4	$A_0$	$A_{150}$	$A_{200}$
	150	200	175	225	RT=150 WT=0	RT=200 WT=150	RT=200 WT=200
1	$R_1(A)$				Read		
2	$W_1(A)$					Create RT=150 WT=150	
3		$R_2(A)$				Read RT=200	
4		$W_2(A)$					Create RT=200 WT=200
5			$R_3(A)$			Read	
6				$R_4(A)$			Read RT=225

Duke CS, Fall 2017 Compsci 516: Database Systems 62

### Summary

- “Phantom Problem” and why serializability/2PL fails
- New requirements and mechanisms for multiple-granularity locks
- Note the key ideas for three timestamp-based alternative approaches (to Lock-based approaches) to CC
  - Optimistic: validation tests
  - Timestamp:  $RT(O)$  &  $WT(O)$  on each object  $O$
  - Multiversion: multiple versions of each object  $O$  with different  $WT$  and  $RT$
- Note: a new action (block or delay) in addition to commit or abort

Duke CS, Fall 2017 Compsci 516: Database Systems 63