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 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: T2 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

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

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

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

Multiple-granularity Locking

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
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

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

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

Transaction in SQL

- SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED ;
- BEGIN TRANSACTION
- … SQL STATEMENTS …
- COMMIT or ROLLBACK

Four isolation levels: performance and serializability

<table>
<thead>
<tr>
<th>Isolation Level</th>
<th>Dirty Read</th>
<th>Unrepeatable Read</th>
<th>Phantom</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ UNCOMMITTED</td>
<td>Maybe</td>
<td>Maybe</td>
<td>Maybe</td>
</tr>
<tr>
<td>READ COMMITTED</td>
<td>No</td>
<td>Maybe</td>
<td>Maybe</td>
</tr>
<tr>
<td>REPEATABLE READS</td>
<td>No</td>
<td>No</td>
<td>Maybe</td>
</tr>
<tr>
<td>SERIALIZABLE</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Approaches to CC other than locking

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

Approaches to Concurrency Control (CC)

- Lock-based CC
- Optimistic CC
- Time-stamp-based CC
- Multi-version CC
**Timestamp**

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

---

**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

---

**Optimistic CC**

- 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

---

**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)

---

**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
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 $T_i$

Validation Tests

- To validate $T_i$
  - 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 $T_i$'s modifications are not visible to the previous transaction $T_j$
- Check yourself: No RW, WR, WW conflicts if any of these tests satisfy

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

Test 2

- 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($T_i$) ∩ ReadSet($T_j$) is empty

Test 3

- 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($T_i$) ∩ ReadSet($T_j$) is empty
  - WriteSet($T_i$) ∩ WriteSet($T_j$) is empty

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 transaction 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)
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

Optimistic CC vs locking

- 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

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

Timestamp-based CC

Main idea:

- Give each object \( O \)
  - A read-timestamp \( RT(O) \), and
  - A write-timestamp \( WT(O) \)
    - RG uses \( RT/WT \), GUW uses \( RT/WT \), either of these is fine
- Give each transaction \( T \)
  - A timestamp \( TS(T) \) when it begins
- If
  - \( ai \) of \( Ti \) conflicts with \( aj \) of \( Tj \)
  - And \( TS(Ti) < TS(Tj) \)
  - Then
    - \( aj \) must occur before \( aj \)
  - Otherwise, abort and restart violating transaction

Timestamp CC

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)
Request for a write: \(W_T(X)\)

1. 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 \(C(X) = false\)  ---> \(T\) not committed yet
2. If \(TS(T) < RT(X)\) 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) \(T\)

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

Timestamp CC and Recoverability

Without “block or delay”, unrecoverable schedules are allowed:
- \(TS(T1) = 1\)
- \(TS(T2) = 2\)

- 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

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
  - \(R1(B), R2(A), R3(C), W1(B), W1(A), W2(C), W3(A)\)
- 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

Initial condition and Steps

<table>
<thead>
<tr>
<th>Step</th>
<th>(T1)</th>
<th>(T2)</th>
<th>(T3)</th>
<th>(A)</th>
<th>(B)</th>
<th>(C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>200</td>
<td>150</td>
<td>175</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(R1(B))</td>
<td>(R2(A))</td>
<td>(R3(C))</td>
<td>(W1(B))</td>
<td>(W1(A))</td>
<td>(W2(C))</td>
</tr>
</tbody>
</table>

After Step 1

<table>
<thead>
<tr>
<th>Step</th>
<th>(T1)</th>
<th>(T2)</th>
<th>(T3)</th>
<th>(A)</th>
<th>(B)</th>
<th>(C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>200</td>
<td>150</td>
<td>175</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(W1(B))</td>
<td>(W1(A))</td>
<td>(W2(C))</td>
<td>(W3(A))</td>
<td>(RT=200)</td>
<td>(C = 1)</td>
</tr>
</tbody>
</table>
### After Step 2

WT of A is <= TS(T2)
C = 1
**Read OK.**

<table>
<thead>
<tr>
<th>Step</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>200</td>
<td>150</td>
<td>175</td>
<td>RT = 150, WT = 0, C = 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>R1(B)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>R1(A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>W1(B)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>W1(A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>W1(C)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>W1(A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### After Step 3

WT of C is <= TS(T3)
C = 1
**Read OK.**

<table>
<thead>
<tr>
<th>Step</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>200</td>
<td>150</td>
<td>175</td>
<td>RT = 150, WT = 0, C = 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>R1(B)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>R1(A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>W1(B)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>W1(A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>W1(C)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>W1(A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### After Step 4

WT & RT of B is <= TS(T1)
**Write OK.**

<table>
<thead>
<tr>
<th>Step</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>200</td>
<td>150</td>
<td>175</td>
<td>RT = 150, WT = 0, C = 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>R1(B)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>R1(A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>W1(B)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>W1(A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>W1(C)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>W1(A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### After Step 5

RT & WT of A is <= TS(T3)
**Write OK.**

<table>
<thead>
<tr>
<th>Step</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>200</td>
<td>150</td>
<td>175</td>
<td>RT = 150, WT = 0, C = 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>R1(B)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>R1(A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>W1(B)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>W1(A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>W1(C)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>W1(A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### After Step 6

RT(C) = 175 < 150 = TS(T1)
**Abort T3**

<table>
<thead>
<tr>
<th>Step</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>200</td>
<td>150</td>
<td>175</td>
<td>RT = 150, WT = 0, C = 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>R1(B)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>R1(A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>W1(B)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>W1(A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>W1(C)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>W1(A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### After Step 7

RT(A) <= TS(T1) – write ok
WT(A) > TS(T1) and C(A) = 0
**Delay T3**

<table>
<thead>
<tr>
<th>Step</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>200</td>
<td>150</td>
<td>175</td>
<td>RT = 150, WT = 0, C = 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>R1(B)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>R1(A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>W1(B)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>W1(A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>W1(C)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>W1(A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Multiversion CC

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

- Ti reads the most recent version whose timestamp precedes TS(Ti)

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.

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

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

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
Example

- Four transactions \( T_1 (TS = 150), T_2 (TS = 200), T_3 (TS = 175), T_4(TS = 225) \)
- One object \( A \)
  - Initial version is \( A_0 \)
- Sequence of actions
  - \( R_1(A), W_1(A), R_2(A), W_2(A), R_3(A), R_4(A) \)
- Q. What is the state of the database at the end if the multiversion CC protocol is followed

**Initial condition and Steps**

\( A_0 \) existed before the transactions started

<table>
<thead>
<tr>
<th>Step</th>
<th>( T_1 )</th>
<th>( T_2 )</th>
<th>( T_3 )</th>
<th>( T_4 )</th>
<th>( A )</th>
<th>( A_{\text{new}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>150</td>
<td>200</td>
<td>175</td>
<td>225</td>
<td>( RT=0 )</td>
<td>( WT=0 )</td>
</tr>
<tr>
<td>2</td>
<td>( R_1(A) )</td>
<td></td>
<td></td>
<td></td>
<td>Read</td>
<td>( RT = 150 )</td>
</tr>
<tr>
<td>3</td>
<td>( W_1(A) )</td>
<td></td>
<td></td>
<td></td>
<td>Create</td>
<td>( RT=150 )</td>
</tr>
<tr>
<td>4</td>
<td>( R_2(A) )</td>
<td></td>
<td></td>
<td></td>
<td>Read</td>
<td>( RT = 200 )</td>
</tr>
<tr>
<td>5</td>
<td>( W_2(A) )</td>
<td></td>
<td></td>
<td></td>
<td>Create</td>
<td>( RT=150 )</td>
</tr>
<tr>
<td>6</td>
<td>( R_3(A) )</td>
<td></td>
<td></td>
<td></td>
<td>Read</td>
<td>( RT = 150 )</td>
</tr>
<tr>
<td>7</td>
<td>( R_4(A) )</td>
<td></td>
<td></td>
<td></td>
<td>Read</td>
<td>( RT = 200 )</td>
</tr>
</tbody>
</table>

**After Step 1**

\( A_0 \) is the newest version with \( WT <= TS(T_1) \)

Read \( A_0 \)

<table>
<thead>
<tr>
<th>Step</th>
<th>( T_1 )</th>
<th>( T_2 )</th>
<th>( T_3 )</th>
<th>( T_4 )</th>
<th>( A )</th>
<th>( A_{\text{new}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>150</td>
<td>200</td>
<td>175</td>
<td>225</td>
<td>( RT=0 )</td>
<td>( WT=0 )</td>
</tr>
<tr>
<td>2</td>
<td>( R_1(A) )</td>
<td></td>
<td></td>
<td></td>
<td>Read</td>
<td>( RT = 150 )</td>
</tr>
<tr>
<td>3</td>
<td>( W_1(A) )</td>
<td></td>
<td></td>
<td></td>
<td>Create</td>
<td>( RT=150 )</td>
</tr>
<tr>
<td>4</td>
<td>( R_2(A) )</td>
<td></td>
<td></td>
<td></td>
<td>Read</td>
<td>( RT = 200 )</td>
</tr>
<tr>
<td>5</td>
<td>( W_2(A) )</td>
<td></td>
<td></td>
<td></td>
<td>Create</td>
<td>( RT=150 )</td>
</tr>
<tr>
<td>6</td>
<td>( R_3(A) )</td>
<td></td>
<td></td>
<td></td>
<td>Read</td>
<td>( RT = 200 )</td>
</tr>
</tbody>
</table>

**After Step 2**

\( A_0 \) is the newest version with \( WT <= TS(T_1) \)

- \( RT(A_0) <= TS(T_1) \)
- Create a new version \( A_{\text{new}} \)
- Set its \( WT, RT \) to \( TS(T_1) = 150 \) \( (A_{\text{new}} \text{ named accordingly}) \)

<table>
<thead>
<tr>
<th>Step</th>
<th>( T_1 )</th>
<th>( T_2 )</th>
<th>( T_3 )</th>
<th>( T_4 )</th>
<th>( A )</th>
<th>( A_{\text{new}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>150</td>
<td>200</td>
<td>175</td>
<td>225</td>
<td>( RT=150 )</td>
<td>( WT=150 )</td>
</tr>
<tr>
<td>2</td>
<td>( R_1(A) )</td>
<td></td>
<td></td>
<td></td>
<td>Read</td>
<td>( RT = 150 )</td>
</tr>
<tr>
<td>3</td>
<td>( W_1(A) )</td>
<td></td>
<td></td>
<td></td>
<td>Create</td>
<td>( RT=150 )</td>
</tr>
<tr>
<td>4</td>
<td>( R_2(A) )</td>
<td></td>
<td></td>
<td></td>
<td>Read</td>
<td>( RT = 200 )</td>
</tr>
<tr>
<td>5</td>
<td>( W_2(A) )</td>
<td></td>
<td></td>
<td></td>
<td>Create</td>
<td>( RT=150 )</td>
</tr>
<tr>
<td>6</td>
<td>( R_3(A) )</td>
<td></td>
<td></td>
<td></td>
<td>Read</td>
<td>( RT = 200 )</td>
</tr>
</tbody>
</table>

**After Step 3**

\( A_{\text{new}} \) is the newest version with \( WT <= TS(T_2) \)

Read \( A_{\text{new}} \)

Create \( A_1 \)

Update \( RT \)

<table>
<thead>
<tr>
<th>Step</th>
<th>( T_1 )</th>
<th>( T_2 )</th>
<th>( T_3 )</th>
<th>( T_4 )</th>
<th>( A )</th>
<th>( A_{\text{new}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>150</td>
<td>200</td>
<td>175</td>
<td>225</td>
<td>( RT=150 )</td>
<td>( WT=150 )</td>
</tr>
<tr>
<td>2</td>
<td>( R_1(A) )</td>
<td></td>
<td></td>
<td></td>
<td>Read</td>
<td>( RT = 150 )</td>
</tr>
<tr>
<td>3</td>
<td>( W_1(A) )</td>
<td></td>
<td></td>
<td></td>
<td>Create</td>
<td>( RT=150 )</td>
</tr>
<tr>
<td>4</td>
<td>( R_2(A) )</td>
<td></td>
<td></td>
<td></td>
<td>Read</td>
<td>( RT = 200 )</td>
</tr>
<tr>
<td>5</td>
<td>( W_2(A) )</td>
<td></td>
<td></td>
<td></td>
<td>Create</td>
<td>( RT=150 )</td>
</tr>
<tr>
<td>6</td>
<td>( R_3(A) )</td>
<td></td>
<td></td>
<td></td>
<td>Read</td>
<td>( RT = 200 )</td>
</tr>
</tbody>
</table>

**After Step 4**

\( A_{\text{new}} \) is the newest version with \( WT <= TS(T_2) \)

- \( RT(A_{\text{new}}) <= TS(T_2) \)
- Create a new version \( A_{\text{new}} \)
- Set its \( WT, RT \) to \( TS(T_2) = 200 \) \( (A_{\text{new}} \text{ named accordingly}) \)

<table>
<thead>
<tr>
<th>Step</th>
<th>( T_1 )</th>
<th>( T_2 )</th>
<th>( T_3 )</th>
<th>( T_4 )</th>
<th>( A )</th>
<th>( A_{\text{new}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>150</td>
<td>200</td>
<td>175</td>
<td>225</td>
<td>( RT=150 )</td>
<td>( WT=150 )</td>
</tr>
<tr>
<td>2</td>
<td>( R_1(A) )</td>
<td></td>
<td></td>
<td></td>
<td>Read</td>
<td>( RT = 150 )</td>
</tr>
<tr>
<td>3</td>
<td>( W_1(A) )</td>
<td></td>
<td></td>
<td></td>
<td>Create</td>
<td>( RT=150 )</td>
</tr>
<tr>
<td>4</td>
<td>( R_2(A) )</td>
<td></td>
<td></td>
<td></td>
<td>Read</td>
<td>( RT = 200 )</td>
</tr>
<tr>
<td>5</td>
<td>( W_2(A) )</td>
<td></td>
<td></td>
<td></td>
<td>Create</td>
<td>( RT=200 )</td>
</tr>
<tr>
<td>6</td>
<td>( R_3(A) )</td>
<td></td>
<td></td>
<td></td>
<td>Read</td>
<td>( RT = 200 )</td>
</tr>
</tbody>
</table>
After Step 5

- \( A_{150} \) is the newest version with WT <= TS(T3)
- Read \( A_{150} \)
- DO NOT Update RT

<table>
<thead>
<tr>
<th>Step</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>A0</th>
<th>A150</th>
<th>A200</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>R1(A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RT=150</td>
<td>WT=0</td>
</tr>
<tr>
<td>2</td>
<td>W1(A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RT=150</td>
<td>WT=150</td>
</tr>
<tr>
<td>3</td>
<td>R2(A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RT=150</td>
<td>WT=200</td>
</tr>
<tr>
<td>4</td>
<td>W2(A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RT=200</td>
<td>WT=150</td>
</tr>
<tr>
<td>5</td>
<td>R3(A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RT=200</td>
<td>WT=200</td>
</tr>
<tr>
<td>6</td>
<td>R4(A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RT=225</td>
<td>WT=200</td>
</tr>
</tbody>
</table>

After Step 6

- \( A_{200} \) is the newest version with WT <= TS(T4)
- Read \( A_{200} \)
- Update RT

<table>
<thead>
<tr>
<th>Step</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>A0</th>
<th>A150</th>
<th>A200</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>R1(A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RT=150</td>
<td>WT=150</td>
</tr>
<tr>
<td>2</td>
<td>W1(A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RT=150</td>
<td>WT=150</td>
</tr>
<tr>
<td>3</td>
<td>R2(A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RT=150</td>
<td>WT=150</td>
</tr>
<tr>
<td>4</td>
<td>W2(A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RT=200</td>
<td>WT=200</td>
</tr>
<tr>
<td>5</td>
<td>R3(A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RT=200</td>
<td>WT=200</td>
</tr>
<tr>
<td>6</td>
<td>R4(A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RT=225</td>
<td>WT=200</td>
</tr>
</tbody>
</table>

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