# Review: Using Timestamps for Concurrency Control

Computer Science E-66
Harvard University

David G. Sullivan, Ph.D.

## Timestamp-Based Concurrency Control

- Transactions are assigned unique timestamps based on when they start.
- The system ensures that all operations are consistent with a serial ordering based on the timestamps.
  - example: TS(T2) < TS(T1), so must be consistent with T2; T1

*actual schedule*

| $T_1$ | $T_2$ |
|---|---|
| | TS = 100<br>r(C) |
| TS = 102<br>**w(A)** | |
| | **r(A)**<br>***denied*** |

*equivalent serial schedule*

| $T_1$ | $T_2$ |
|---|---|
| | TS = 100<br>r(C)<br>**r(A)**<br>… |
| TS = 102<br>**w(A)**<br>… | |

T2's read of A is *too late*:

- if this read were allowed, T2 would read T1's write of A
- in the equiv. serial schedule, it would read A's old value

→ the DBMS denies the read, rolls back T2, and makes it start over

## Timestamp-Based Concurrency Control (cont.)

- Transactions are assigned timestamps based on when they start.
- Each data item D has a:
  - RTS – the largest timestamp of any txn that has read D
  - WTS – the largest timestamp of any txn that has written D

| T1 | T2 | A | B |
|---|---|---|---|
| | | RTS = WTS = 0 | RTS = WTS = 0 |
| TS = 220<br>**w(A)** | | **WTS = 220** | |
| | TS = 230<br>**w(A)** | **WTS = 230** | |
| | **r(B)** | | **RTS = 230** |
| **r(B)** | | | **RTS: no change** |

## Timestamp Rules for Reads and Writes

- When T tries to read A:
  - if TS(T) < WTS(A), roll back T and restart it
    - T's read is too late
  - else allow the read
    - set RTS(A) = max(TS(T), RTS(A))
- When T tries to write A:
  - if TS(T) < RTS(A), roll back T and restart it
    - T's write is too late
  - else if TS(T) < WTS(A), ignore the write and let T continue
    - in the equiv serial sched, T's write would be overwritten
  - else allow the write
    - set WTS(A) = TS(T)

## Ensuring Recoverability and Casecadelessness

- Transactions are assigned timestamps based on when they start.
- Each data item D has a:
  - RTS – the largest timestamp of any txn that has read D
  - WTS – the largest timestamp of any txn that has written D
  - **commit bit – used to prevent dirty reads**
    - **true if the writer of the current value has committed**
    - **false otherwise**

| T1 | T2 | A | B |
| --- | --- | --- | --- |
| | | RTS = WTS = 0; **c = true** | RTS = WTS = 0; **c = true** |
| TS = 220<br>w(A) | | WTS = 220; **c = false** | |
| | TS = 230<br>w(A) | WTS = 230; **c = false** | |
| | r(B) | | RTS = 230 |
| r(B) | | | RTS: no change |
| commit | | c: no change | |
| | **commit** | **c = true** | |

## Timestamp Rules for Reads and Writes

**when using commit bits**

- When T tries to read A:
  - if TS(T) < WTS(A), roll back T and restart it
    - T's read is too late (see our earlier example)
  - else allow the read **(but if c(A) == false, make T wait)**
    - set RTS(A) = max(TS(T), RTS(A))
- When T tries to write A:
  - if TS(T) < RTS(A), roll back T and restart it
    - T's write is too late (see example 2 from last lecture)
  - else if TS(T) < WTS(A), ignore the write and let T continue **(but if c(A) == false, make T wait)**
    - in the equiv serial sched, T's write would be overwritten
  - else allow the write
    - set WTS(A) = TS(T) **(and set c(A) to false)**

## Other Details

- When the writer of *the current value* of data item A commits, we:
  - set A's commit bit to true
  - allow waiting txns try again
- When a txn T is rolled back, we process:
  - all data elements A for which WTS(A) == TS(T)
    - restore their prior state (value and timestamps)
    - set their commit bits based on whether the writer of the prior value has committed
    - make waiting txns try again
  - all data elements A for which RTS(A) == TS(T)
    - restore their prior RTS

## Example of Using Timestamps and Commit Bits

- The balance-transfer example would now proceed differently.

*T1*
```
read balance1
write(balance1 - 500)


read balance2
write(balance2 + 500)
```

*T2*
```
read balance1
read balance2
if (balance1 + balance2 < min)
    write(balance1 - fee)
```

| T1 | T2 | bal1 | bal2 |
|---|---|---|---|
| | | RTS = WTS = 0<br>c = true | RTS = WTS = 0<br>c = true |
| TS = 350<br>r(bal1)<br>w(bal1) | | RTS = 350<br>WTS = 350; c = false | |
| | TS = 375<br>r(bal1)<br>***denied: wait*** | | |
| r(bal2)<br>w(bal2)<br>commit | | c = true | RTS = 350<br>WTS = 350; c = false<br>c = true |
| | r(bal1)<br>*and completes* | RTS = 375 | |