Deadlocks

- A deadlock is an impasse that may result when two or more transactions are waiting for locks to be released which are held by each other.
  - For example: T1 has a lock on X and is waiting for a lock on Y, and T2 has a lock on Y and is waiting for a lock on X.

- Given a schedule, we can detect deadlocks which will happen in this schedule using a wait-for graph (WFG).

Precedence/Wait-For Graphs

- Precedence graph
  - Each transaction is a vertex
  - Arrows from T1 to T2 if
    - T1 reads X before T2 writes X
    - T1 writes X before T2 reads X
    - T1 writes X before T2 writes X

- Wait-for Graph
  - Each transaction is a vertex
  - Arrows from T2 to T1 if
    - T1 read-locks X then T2 tries to write-lock it
    - T1 write-locks X then T2 tries to read-lock it
    - T1 write-locks X then T2 tries to write-lock it

Example

T1: Read(X)
T2: Read(Y)
T1: Write(X)
T2: Read(X)
T3: Read(Z)
T3: Write(Z)
T1: Read(Y)
T3: Read(X)
T1: Write(Y)
Deadlock Prevention

- Deadlocks can arise with 2PL
  - Deadlock is less of a problem than an inconsistent DB
  - We can detect and recover from deadlock
  - It would be nice to avoid it altogether
- Conservative 2PL
  - All locks must be acquired before the transaction starts
  - Hard to predict what locks are needed
  - Low 'lock utilisation' - transactions can hold on to locks for a long time, but not use them much
Deadlock Prevention

- We impose an ordering on the resources
  - Transactions must acquire locks in this order
  - Transactions can be ordered on the last resource they locked

- This prevents deadlock
  - If T1 is waiting for a resource from T2 then that resource must come after all of T1’s current locks
  - All the arcs in the wait-for graph point ‘forwards’ - no cycles

Example of resource ordering

- Suppose resource order is: X < Y
- This means, if you need locks on X and Y, you first acquire a lock on X and only after that a lock on Y
  - (even if you want to write to Y before doing anything to X)
- It is impossible to end up in a situation when T1 is waiting for a lock on X held by T2, and T2 is waiting for a lock on Y held by T1.

Timestamping

- Transactions can be run concurrently using a variety of techniques
- We looked at using locks to prevent interference

- An alternative is timestamping
  - Requires less overhead in terms of tracking locks or detecting deadlock
  - Determines the order of transactions before they are executed

Timestamping Example

- Each transaction has a timestamp, TS,
  - TS(T1) < TS(T2) if T1 starts before T2
  - Can use the system clock or an incrementing counter to generate timestamps

Each resource has two timestamps
  - R(X), the largest timestamp of any transaction that has read X
  - W(X), the largest timestamp of any transaction that has written X

Timestamp Protocol

- T tries to read X
  - If TS(T) < W(X) T is rolled back and restarted with a later timestamp
  - Otherwise the read succeeds and we set R(X) = max(R(X), TS(T))

- T tries to write X
  - If TS(T) < W(X) or TS(T) < R(X) then T is rolled back and restarted with a later timestamp
  - Otherwise the write succeeds and we set W(X) = TS(T)

Timestamping Example

- Given T1 and T2 we will assume
  - The transactions make alternate operations
  - Timestamps are allocated from a counter starting at 1
  - T1 goes first
  - T1 Read(X)
  - T2 Read(Y)
  - Y = Y + X
  - T2 Read(Y)
  - T2 Write(Z)
  - T2 Write(Z)
Timestamp Example

T1
Read(X)
Read(Y)
Y = Y + X
Write(Y)
T2
Read(X)
Read(Y)
Z = Y - X
Write(Z)
**Timestamp Example**

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read(X)</td>
<td>Read(X)</td>
</tr>
<tr>
<td>Read(Y)</td>
<td>Read(Y)</td>
</tr>
<tr>
<td>( Y = Y + X )</td>
<td>( Z = Y - X )</td>
</tr>
<tr>
<td>Write(Y)</td>
<td>Write(Z)</td>
</tr>
</tbody>
</table>

\[
\begin{array}{ccc}
R & X & Y \\
W & 3 & 3 \\
TS & T1 & T2 \\
& 3 & 2
\end{array}
\]

**Timestamp Example**

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read(X)</td>
<td>Read(X)</td>
</tr>
<tr>
<td>Read(Y)</td>
<td>Read(Y)</td>
</tr>
<tr>
<td>( Y = Y + X )</td>
<td>( Z = Y - X )</td>
</tr>
<tr>
<td>Write(Y)</td>
<td>Write(Z)</td>
</tr>
</tbody>
</table>

\[
\begin{array}{ccc}
R & X & Y \\
W & 3 & 3 \\
TS & T1 & T2 \\
& 3 & 2
\end{array}
\]

---

**Timestamping**

- The protocol means that transactions with higher times take precedence
  - Equivalent to running transactions in order of their final time values
  - Transactions don't wait - no deadlock
- Problems
  - Long transactions might keep getting restarted by new transactions - starvation
  - Rolls back old transactions, which may have done a lot of work

---

**Next Lecture**

- Database Security
  - Aspects of security
  - Access to databases
  - Privileges and views
- Database Integrity
  - View updating, Integrity constraints
- For more information
  - Connolly and Begg chapters 6 and 19