Lock Conversion

- A refinement of the basic 2PL
- Upgrade – from shared (S) to exclusive (X)
  - in Growing phase only
- Downgrade – from X to S
  - in Shrinking phase only
2. Timestamp-Based Protocols

- Another serializability method

- Ensures conflict serializability – any conflicting read and write operations are executed in timestamp order

- A transaction rolled back by concurrency scheme is assigned a new timestamp and is restarted

- Therefore, free from deadlocks

---

Timestamp

- Each transaction associates a unique fixed timestamp TS(Ti) assigned by the system before Ti starts execution

- Either by system clock or by a logical counter

- If TS(Ti) < TS(Tj), then the system ensures the produced schedule is equivalent to a serial schedule in which transaction Ti appears before transaction Tj

- W-timestamp(Q) – the largest timestamp of any transaction that successfully executed write(Q)

- R-timestamp(Q) – the largest timestamp of any transaction that successfully execute read(Q)
**Timestamp-Ordering Protocol**

- To ensure that any conflicting *read* and *write* operations are executed in timestamp order

1. Transaction $T_i$ issues read($Q$)

   - If $TS(T_i) < W$-timestamp($Q$)
     - I.e. $T_i$ read a value of $Q$ overwritten by another transaction since $T_i$ started
     - **Result:** read($Q$) is rejected and $T_i$ rollback

   - If $TS(T_i) \geq W$-timestamp($Q$),
     - read is executed, and $R$-timestamp($Q$) is set to the maximum of $R$-timestamp($Q$) and $TS(T_i)$

2. $T_i$ issues write($Q$)

   1. If $TS(T_i) < R$-timestamp($Q$)
      - I.e. the value of $Q$ produced was previously needed
      - Therefore, the write is rejected and $T_i$ is rolled back

   2. If $TS(T_i) < W$-timestamp($Q$)
      - I.e. $T_i$ is attempting to write an obsolete value of $Q$
      - Therefore, the write is rejected and $T_i$ is rolled back

   - Otherwise, the write operation is executed and $W$-timestamp is set to the maximum of $W$-timestamp($Q$) and $TS(T_i)$
Review Exercise

- John and Mary joint account has an initial balance of $2000. Mary deposits $1000 through one ATM while John withdraws $500 from another location. Their transactions are performed concurrently as follows:

<table>
<thead>
<tr>
<th></th>
<th>Mary</th>
<th>John</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Read Balance</td>
<td>Balance = Balance – 500</td>
</tr>
<tr>
<td>Read Balance</td>
<td></td>
<td>Write Balance</td>
</tr>
<tr>
<td>Balance = Balance + 1000</td>
<td>Write Balance</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- A. Identify, if any, problems(?) with this banking system
- B. Apply Timestamping Protocol to this schedule and illustrate by highlighting all intermediate results

Set Initial Timestamps

<table>
<thead>
<tr>
<th>Mary</th>
<th>John</th>
<th>$T_{Mary}$</th>
<th>$T_{John}$</th>
<th>$R_{Bal}$</th>
<th>$W_{Bal}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Read Bal</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read Bal</td>
<td></td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bal=Bal-500</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write Bal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bal=Bal+1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write Bal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Result: Mary deposits $1000, then John withdraws $500

### 3. Validation (Optimistic) Technique

- If the majority of transactions are read-only, the rate of conflicts may be very low
- In this case, there is a need for a lower overhead scheme and less delay of transactions
- Each transaction consists of three phases

<table>
<thead>
<tr>
<th>Mary</th>
<th>John</th>
<th>$T_{Mary}$</th>
<th>$T_{John}$</th>
<th>$R_{Bal}$</th>
<th>$W_{Bal}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Bal</td>
<td></td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read Bal</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read Bal</td>
<td></td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bal = Bal - 500</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write Bal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bal = Bal + 1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write Bal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bal = Bal - 500</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write Bal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Three Phases

- **1. Read phase**
  - Items are read and stored in variables local to $T_i$
  - Write operations take place on temporary local variables without actual database update

---

Read

---

Three Phases

- **1. Read phase**
  - Items are read and stored in variables local to $T_i$
  - Write operations take place on temporary local variables without actual database update

- **2. Validation phase**
  - To test if the copy of write results from temporary to the database causes a violation of serializability

---

Read  Validation
Three Phases

1. Read phase
   - Items are read and stored in variables local to $T_i$
   - Write operations take place on temporary local variables without actual database update

2. Validation phase
   - To test if the copy of write results from temporary to the database causes a violation of serializability

3. Write phase
   - If validation succeeds, then apply actual database update, otherwise $T_i$ is rolled back

Three Timestamps

- Start($T_i$) – start of execution
- Validation($T_i$) – start of validation phase
- Finish($T_i$) – write phase complete

$TS(T_i) = Validation(T_i)$ rather than $Start(T_i)$ for better response provided conflict rates among transactions are low
Validation Test

- For $T_j$ requires that for all $T_i$ with $TS(T_i) < TS(T_j)$, either one of the two must hold:
  
  1. $Finish(T_i) < Start(T_j)$

  2. The set of data items written by $T_i$ does not intersect with the set of data items read by $T_j$ and $T_i$ completes its write phase before $T_j$ starts its validation phase
     
        $(Start(T_j) < Finish(T_i) < Validation(T_j))$

Summary

- Serializability

- Three Common Approaches
  
  1. Lock
  2. Timestamp
  3. Validation (Optimistic)