Concurrency Control

continued...

Lock instructions

- New instructions
  - lock-S: shared lock request
  - lock-X: exclusive lock request
  - unlock: release previously held lock

Example schedule:

T1
lock-X(B)
read(B)
B ← B - 50
write(B)
unlock(B)
lock-X(A)
read(A)
A ← A + 50
write(A)
unlock(A)

T2
lock-S(A)
read(A)
unlock(A)
lock-S(B)
read(B)
unlock(B)
display(A + B)

Not serializable

Not enough to take minimum locks when you need to read/write something!
2-Phase Locking Protocol (2PL)

- **Phase 1: Growing phase**
  - Transaction may obtain locks
  - But may not release them

- **Phase 2: Shrinking phase**
  - Only release locks

- **2PL guarantees conflict-serializability**
  - *lock-point*: the time at which a transaction acquired last lock
  - if \( lock-point(T1) < lock-point(T2) \), there can't be an edge from \( T2 \) to \( T1 \) in the precedence graph

Example: T1 in 2PL

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Operation</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>lock-X(B)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>read(B)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B ↵ B-50</td>
<td></td>
</tr>
<tr>
<td></td>
<td>write(B)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>unlock(B)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>lock-X(A)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>read(A)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A ↵ A + 50</td>
<td></td>
</tr>
<tr>
<td></td>
<td>write(A)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>unlock(A)</td>
<td></td>
</tr>
</tbody>
</table>

Growing phase

Shrinking phase
2 Phase Locking

- Guarantees \textit{conflict-serializability},
- not cascade-less recoverability

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>lock-X(A), lock-S(B) read(A) read(B) write(A) unlock(A), unlock(B)</td>
<td>lock-X(A) read(A) write(A) unlock(A) commit</td>
<td>lock-S(A) read(A) commit</td>
</tr>
</tbody>
</table>

- Guaranteeing just recoverability:
  - If T2 performs a dirty read from T1, then:
    - T2 can't commit until T1 either commits or aborts
      - If T1 commits, T2 can proceed with committing
      - If T1 aborts, T2 must abort
  - So cascades still happen
**Strict 2PL**

- Release *exclusive* locks only at the very end, just before commit or abort

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>lock-X(A), lock-S(B)</td>
<td>read(A)</td>
<td>lock-X(A)</td>
</tr>
<tr>
<td></td>
<td>read(B)</td>
<td>write(A)</td>
<td>read(A)</td>
</tr>
<tr>
<td></td>
<td>write(A)</td>
<td>unlock(A), unlock(B)</td>
<td>write(A)</td>
</tr>
<tr>
<td></td>
<td>lock-X(A)</td>
<td>lock-S(A)</td>
<td>unlock(A)</td>
</tr>
<tr>
<td></td>
<td>read(A)</td>
<td>read(A)</td>
<td>Commit</td>
</tr>
<tr>
<td></td>
<td>write(A)</td>
<td>write(A)</td>
<td>Commit</td>
</tr>
<tr>
<td></td>
<td>unlock(A)</td>
<td>unlock(A)</td>
<td>commit</td>
</tr>
<tr>
<td></td>
<td>commit</td>
<td>commit</td>
<td>commit</td>
</tr>
</tbody>
</table>

**Strict 2PL**

- Release *exclusive* locks only at the very end
  - just before commit or abort
  - *Guarantees cascade-less and recoverable schedules*
Strict 2PL

- Release *exclusive* locks only at the very end, just before commit or abort
  - Read locks are ignored

- **Rigorous 2PL**: Release both exclusive *and read* locks only at the very end
  - Makes serializability order == the commit order
  - More intuitive behavior for the users
    - No difference for the system

---

**Strict 2PL**

- **Lock conversion:**
  - Transaction might not be sure what it needs a write lock on
    - Start with a S lock
      - *Upgrade* to an X lock later if needed
    - Doesn’t change any of the other properties of the protocol
Recap so far…

- Concurrency Control Scheme
  - A way to guarantee serializability, recoverability etc

- Lock-based protocols
  - Use *locks* to prevent multiple transactions accessing the same data items

- 2 Phase Locking
  - Locks acquired during *growing phase*, released during *shrinking phase*

- Strict 2PL, Rigorous 2PL