Timestamp Ordering
Concurrency Control
**ADMINISTRIVIA**

**Homework #4:** Monday Nov 12\textsuperscript{th} @ 11:59pm

**Project #3:** Monday Nov 19\textsuperscript{th} @ 11:59am
CONCURRENCY CONTROL APPROACHES

**Two-Phase Locking (2PL)**
→ Determine serializability order of conflicting operations at runtime while txns execute.

**Timestamp Ordering (T/O)**
→ Determine serializability order of txns before they execute.
CONCURRENCY CONTROL APPROACHES

Two-Phase Locking (2PL)
→ Determine serializability order of conflicting operations at runtime while txns execute.

Timestamp Ordering (T/O)
→ Determine serializability order of txns before they execute.
T/O CONCURRENCY CONTROL

Use timestamps to determine the serializability order of txns.

If $TS(T_i) < TS(T_j)$, then the DBMS must ensure that the execution schedule is equivalent to a serial schedule where $T_i$ appears before $T_j$. 
TIMESTAMP ALLOCATION

Each txn $T_i$ is assigned a unique fixed timestamp that is monotonically increasing.

→ Let $TS(T_i)$ be the timestamp allocated to txn $T_i$.
→ Different schemes assign timestamps at different times during the txn.

Multiple implementation strategies:
→ System Clock.
→ Logical Counter.
→ Hybrid.
TODAY'S AGENDA

Basic Timestamp Ordering Protocol
Optimistic Concurrency Control
Partition-based Timestamp Ordering
Isolation Levels
BASIC T/O

Txns read and write objects without locks.

Every object $X$ is tagged with timestamp of the last txn that successfully did read/write:

→ $W-\text{TS}(X)$ – Write timestamp on $X$
→ $R-\text{TS}(X)$ – Read timestamp on $X$

Check timestamps for every operation:
→ If txn tries to access an object "from the future", it aborts and restarts.
BASIC T/O – READS

If $TS(T_i) < W-TS(X)$, this violates timestamp order of $T_i$ with regard to the writer of $X$.
→ Abort $T_i$ and restart it with same TS.

Else:
→ Allow $T_i$ to read $X$.
→ Update $R-TS(X)$ to $\max(R-TS(X), TS(T_i))$
→ Have to make a local copy of $X$ to ensure repeatable reads for $T_i$. 
If $\text{TS}(T_i) < R - \text{TS}(X)$ or $\text{TS}(T_i) < W - \text{TS}(X)$
→ Abort and restart $T_i$.

Else:
→ Allow $T_i$ to write $X$ and update $W - \text{TS}(X)$
→ Also have to make a local copy of $X$ to ensure repeatable reads for $T_i$. 
BASIC T/O – EXAMPLE #1

Schedule

\[ TS(T_1) = 1 \]
\[ TS(T_2) = 2 \]

\begin{align*}
&\text{BEGIN} \\
&R(B) \\
&R(A) \\
&\text{COMMIT} \\
&\text{BEGIN} \\
&R(B) \\
&W(B) \\
&R(A) \\
&W(A) \\
&\text{COMMIT}
\end{align*}

Database

<table>
<thead>
<tr>
<th>Object</th>
<th>R-TS</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
BASIC T/O – EXAMPLE #1

**Schedule**

<table>
<thead>
<tr>
<th>Event</th>
<th>Time</th>
<th>Object</th>
<th>R-TS</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN</td>
<td>T1</td>
<td>A</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R(B)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R(A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMIT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- \(TS(T_1)=1\)
- \(TS(T_2)=2\)

**Database**

<table>
<thead>
<tr>
<th>Object</th>
<th>R-TS</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

TIME
BASIC T/O – EXAMPLE #1

**Schedule**

- **TS(T₁)=1**
- **TS(T₂)=2**

**Database**

<table>
<thead>
<tr>
<th>Object</th>
<th>R-TS</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

**EXAMPLE #1**

BEGIN R(B)
R(A)
BEGIN R(B) W(B)
R(A) W(A)
COMMIT

**Example**

<table>
<thead>
<tr>
<th>TIME</th>
<th>Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>T₁</td>
<td>R(B)</td>
</tr>
<tr>
<td>T₂</td>
<td>R(B) W(B) R(A) W(A) COMMIT</td>
</tr>
<tr>
<td></td>
<td>COMMIT</td>
</tr>
</tbody>
</table>
BASIC T/O – EXAMPLE #1

Schedule:

\[ TS(T_1) = 1 \]
\[ TS(T_2) = 2 \]

```
BEGIN
R(B)
R(A)
COMMIT
```

```
BEGIN
R(B)
W(B)
R(A)
W(A)
COMMIT
```

Database:

<table>
<thead>
<tr>
<th>Object</th>
<th>R-TS</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
**BASIC T/O – EXAMPLE #1**

**Schedule**

- **TS(T₁)=1**
  - T₁
  - BEGIN
  - R(B)
  - R(A)
  - COMMIT

- **TS(T₂)=2**
  - T₂
  - BEGIN
  - R(B)
  - W(B)
  - R(A)
  - W(A)
  - COMMIT

**Database**

<table>
<thead>
<tr>
<th>Object</th>
<th>R-TS</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
BASIC T/O – EXAMPLE #1

Schedule:

- $TS(T_1) = 1$
- $TS(T_2) = 2$

<table>
<thead>
<tr>
<th>Time</th>
<th>$T_1$</th>
<th>$T_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R(B)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R(A)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMMIT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BEGIN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R(B)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W(B)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R(A)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W(A)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMMIT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Database:

<table>
<thead>
<tr>
<th>Object</th>
<th>R-TS</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
### BASIC T/O – EXAMPLE #1

**Schedule**

- \(TS(T_1) = 1\)
- \(TS(T_2) = 2\)

**Database**

<table>
<thead>
<tr>
<th>Object</th>
<th>R-TS</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

No violations so both txns are safe to commit.
BASIC T/O – EXAMPLE #2

Schedule

<table>
<thead>
<tr>
<th>T₁</th>
<th>T₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN</td>
<td>BEGIN</td>
</tr>
<tr>
<td>R(A)</td>
<td>W(A)</td>
</tr>
<tr>
<td>W(A)</td>
<td></td>
</tr>
<tr>
<td>R(A)</td>
<td></td>
</tr>
<tr>
<td>COMMIT</td>
<td>COMMIT</td>
</tr>
</tbody>
</table>

Database

<table>
<thead>
<tr>
<th>Object</th>
<th>R-TS</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
BASIC T/O – EXAMPLE #2

Schedule

<table>
<thead>
<tr>
<th></th>
<th>T₁</th>
<th>T₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R(A)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W(A)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R(A)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMMIT</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BEGIN</td>
<td></td>
</tr>
<tr>
<td>W(A)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMMIT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Database

<table>
<thead>
<tr>
<th>Object</th>
<th>R-TS</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

TIME
BASIC T/O – EXAMPLE #2

Schedule

<table>
<thead>
<tr>
<th>Time</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BEGIN R(A)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>BEGIN W(A)</td>
</tr>
<tr>
<td></td>
<td>W(A)</td>
<td>COMMIT</td>
</tr>
<tr>
<td></td>
<td>R(A)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>COMMIT</td>
<td></td>
</tr>
</tbody>
</table>

Database

<table>
<thead>
<tr>
<th>Object</th>
<th>R-TS</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Violation: \( TS(T_1) < W-TS(A) \)

T1 cannot overwrite update by T2, so the DBMS has to abort it!
THOMAS WRITE RULE

If $TS(T_i) < R - TS(X)$:
→ Abort and restart $T_i$.

If $TS(T_i) < W - TS(X)$:
→ **Thomas Write Rule**: Ignore the write and allow the txn to continue.
→ This violates timestamp order of $T_i$.

Else:
→ Allow $T_i$ to write $X$ and update $W - TS(X)$
BASIC T/O – EXAMPLE #2

**Database**

<table>
<thead>
<tr>
<th>Object</th>
<th>R-TS</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Schedule**

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN</td>
<td></td>
<td>BEGIN</td>
</tr>
<tr>
<td>R(A)</td>
<td></td>
<td>W(A)</td>
</tr>
<tr>
<td>W(A)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R(A)</td>
<td>COMMIT</td>
<td></td>
</tr>
<tr>
<td>COMMIT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
BASIC T/O – EXAMPLE #2

**Schedule**

<table>
<thead>
<tr>
<th></th>
<th>T₁</th>
<th>T₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R(A)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BEGIN</td>
<td>W(A)</td>
</tr>
<tr>
<td>W(A)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R(A)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMMIT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Database**

<table>
<thead>
<tr>
<th>Object</th>
<th>R-TS</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Time**

- T₁ starts first with `BEGIN`, then `R(A)`, `W(A)`, `COMMIT`.
- T₂ starts after T₁ with `BEGIN`, then `W(A)`.
We do not update \( W(TS(A)) \).

Ignore the write and allow \( T_1 \) to commit.
**BASIC T/O**

Generates a schedule that is conflict serializable if you do **not** use the Thomas Write Rule.

→ No deadlocks because no txn ever waits.
→ Possibility of starvation for long txns if short txns keep causing conflicts.

Permits schedules that are not **recoverable**.
RECOVERABLE SCHEDULES

A schedule is **recoverable** if txns commit only after all txns whose changes they read, commit.

Otherwise, the DBMS cannot guarantee that txns read data that will be restored after recovering from a crash.
RECOVERABLE SCHEDULES

\[ T_2 \text{ is allowed to read the writes of } T_1. \]
RECOVERABLE SCHEDULES

Schedule

T₁

BEGIN
W(A)

⋮

ABORT

T₂

BEGIN
R(A)
W(B)
COMMIT

T₂ is allowed to read the writes of T₁.

This is not recoverable because we cannot restart T₁.

T₁ aborts after T₂ has committed.
BASIC T/O – PERFORMANCE ISSUES

High overhead from copying data to txn's workspace and from updating timestamps.

Long running txns can get starved.
→ The likelihood that a txn will read something from a newer txn increases.
OBSERVATION

If you assume that conflicts between txns are **rare** and that most txns are **short-lived**, then forcing txns to wait to acquire locks adds a lot of overhead.

A better approach is to optimize for the no-conflict case.
OPTIMISTIC CONCURRENCY CONTROL

The DBMS creates a private workspace for each txn.
→ Any object read is copied into workspace.
→ Modifications are applied to workspace.

When a txn commits, the DBMS compares workspace write set to see whether it conflicts with other txns.

If there are no conflicts, the write set is installed into the "global" database.
# OCC PHASES

#1 – Read Phase:
→ Track the read/write sets of txns and store their writes in a private workspace.

#2 – Validation Phase:
→ When a txn commits, check whether it conflicts with other txns.

#3 – Write Phase:
→ If validation succeeds, apply private changes to database. Otherwise abort and restart the txn.
OCC – EXAMPLE

Schedule

T₁

BEGIN
READ R(A)
W(A)
VALIDATE
WRITE
COMMIT

T₂

BEGIN
READ R(A)
VALIDATE
WRITE
COMMIT

Database

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
OCC – EXAMPLE

**Schedule**

- **T₁**
  - BEGIN
  - READ
  - R(A)
  - W(A)
  - VALIDATE
  - WRITE
  - COMMIT

- **T₂**
  - BEGIN
  - READ
  - R(A)
  - VALIDATE
  - WRITE
  - COMMIT

**Database**

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Example**

BEGIN
READ R(A)
VALIDATE
WRITE
COMMIT

BEGIN
READ R(A)
VALIDATE
WRITE
COMMIT
OCC – EXAMPLE

Schedule

\[ \begin{array}{c|c|c|}
\text{T_1} & \text{T_2} \\
\hline
\text{BEGIN} & \text{BEGIN} \\
\text{READ} & \text{READ} \\
\text{R(A)} & \text{R(A)} \\
\text{VALIDATE} & \text{VALIDATE} \\
\text{WRITE} & \text{WRITE} \\
\text{W(A)} & \text{W(A)} \\
\text{COMMIT} & \text{COMMIT} \\
\end{array} \]

Database

\[
\begin{array}{c|c|c|}
\text{Object} & \text{Value} & \text{W-TS} \\
\hline
\text{A} & 123 & 0 \\
\text{-} & \text{-} & \text{-} \\
\end{array}
\]

T_1 Workspace

\[
\begin{array}{c|c|c|}
\text{Object} & \text{Value} & \text{W-TS} \\
\hline
\text{-} & \text{-} & \text{-} \\
\text{-} & \text{-} & \text{-} \\
\end{array}
\]
OCC – EXAMPLE

Schedule

<table>
<thead>
<tr>
<th>T₁</th>
<th>T₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN</td>
<td>BEGIN</td>
</tr>
<tr>
<td>READ</td>
<td>READ</td>
</tr>
<tr>
<td>R(A)</td>
<td>R(A)</td>
</tr>
<tr>
<td>WRITE</td>
<td>VALIDATE</td>
</tr>
<tr>
<td>VALIDATE</td>
<td>WRITE</td>
</tr>
<tr>
<td>W(A)</td>
<td>COMMIT</td>
</tr>
<tr>
<td>COMMIT</td>
<td></td>
</tr>
</tbody>
</table>

Database

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>0</td>
</tr>
<tr>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
</tbody>
</table>

OCC Example

BEGIN READ R(A)
VALIDATE WRITE
W(A)
COMMIT

BEGIN
READ R(A)
VALIDATE WRITE
123
0
A

T₁ Workspace

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>0</td>
</tr>
<tr>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
</tbody>
</table>
OCC – EXAMPLE

Schedule

T₁  T₂
BEGIN
READ R(A)
VALIDATE
WRITE
W(A)
COMMIT
BEGIN
READ R(A)
VALIDATE
WRITE

Database

Object  Value  W-TS
A  123  0
-
-

T₁ Workspace

Object  Value  W-TS
A  123  0
-
-

T₂ Workspace

Object  Value  W-TS
-
-
-
-
-
OCC – EXAMPLE

Schedule

T₁  T₂

BEGIN

READ
R(A)

W(A)

VALIDATE
WRITE

COMMIT

BEGIN

READ
R(A)

VALIDATE
WRITE

COMMIT

Database

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

T₁ Workspace

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

T₂ Workspace

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
OCC – EXAMPLE

Schedule

\[ T_1 \]
BEGIN
READ R(A)
W(A)
VALIDATE
WRITE
COMMIT

\[ T_2 \]
BEGIN
READ R(A)
VALIDATE
WRITE
COMMIT

Database

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ TS(T_2) = 1 \]

\[ T_1 \] Workspace

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ T_2 \] Workspace

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

BEGIN
READ R(A)
VALIDATE
WRITE
COMMIT
OCC – EXAMPLE

**Schedule**

- **T₁**
  - BEGIN
  - READ R(A)
  - W(A)
  - VALIDATE
  - WRITE
  - COMMIT

- **T₂**
  - BEGIN
  - READ R(A)
  - VALIDATE
  - WRITE
  - COMMIT

**Database**

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**TS(T₂) = 1**

**T₁ Workspace**

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**T₂ Workspace**

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
OCC – EXAMPLE

Schedule

T₁

BEGIN
READ R(A)
W(A)
VALIDATE
WRITE
COMMIT

T₂

BEGIN
READ R(A)
VALIDATE
WRITE
COMMIT

Database

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Ts(T₂) = 1

T₁ Workspace

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
OCC - EXAMPLE

Schedule

<table>
<thead>
<tr>
<th>T₁</th>
<th>T₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN</td>
<td>BEGIN</td>
</tr>
<tr>
<td>READ R(A)</td>
<td>READ R(A)</td>
</tr>
<tr>
<td>W(A)</td>
<td>VALIDATE</td>
</tr>
<tr>
<td>VALIDATE</td>
<td>WRITE</td>
</tr>
<tr>
<td>COMMIT</td>
<td>COMMIT</td>
</tr>
</tbody>
</table>

Database

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

TS(T₂)=1

T₁ Workspace

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>456</td>
<td>∞</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

CMU 15-445/645 (Fall 2018)
### OCC – EXAMPLE

**Schedule**

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>BEGIN</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>READ R(A)</td>
<td>123</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>W(A)</td>
<td>456</td>
<td>∞</td>
</tr>
<tr>
<td></td>
<td>COMMIT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Database**

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**T₁ Workspace**

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>456</td>
<td>∞</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TS(T₁)=2**

**TS(T₂)=1**
OCC – EXAMPLE

Schedule

<table>
<thead>
<tr>
<th>T₁</th>
<th>T₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN</td>
<td>BEGIN</td>
</tr>
<tr>
<td><strong>READ</strong> R(A)</td>
<td><strong>READ</strong> R(A)</td>
</tr>
<tr>
<td><strong>VALIDATE</strong></td>
<td><strong>VALIDATE</strong></td>
</tr>
<tr>
<td><strong>WRITE</strong></td>
<td><strong>WRITE</strong></td>
</tr>
<tr>
<td><strong>W(A)</strong></td>
<td><strong>W(A)</strong></td>
</tr>
<tr>
<td>COMMIT</td>
<td>COMMIT</td>
</tr>
</tbody>
</table>

Database

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>456</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

T₁ Workspace

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>456</td>
<td>∞</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

TS(T₁) = 2

TS(T₂) = 1

Example

BEGIN
READ R(A)
W(A)
VALIDATE
WRITE
COMMIT

BEGIN
READ R(A)
VALIDATE
WRITE
COMMIT

456
1
456
2
123
0
A
456
∞

TS(T₂) = 1

TS(T₁) = 2
OCC – VALIDATION PHASE

The DBMS needs to guarantee only serializable schedules are permitted.

$T_i$ checks other txns for RW and WW conflicts and makes sure that all conflicts go one way (from older txns to younger txns).
OCC – SERIAL VALIDATION

Maintain global view of all active txns.

Record read set and write set while txns are running and write into private workspace.

Execute Validation and Write phase inside a protected critical section.
OCC – VALIDATION PHASE

Each txn's timestamp is assigned at the beginning of the validation phase.

Check the timestamp ordering of the committing txn with all other running txns.

If $\text{TS}(T_i) < \text{TS}(T_j)$, then one of the following three conditions must hold...
OCC – VALIDATION STEP #1

Ti completes all three phases before Tj begins.
OCC – VALIDATION STEP #1

Schedule

T₁
BEGIN
READ
VALIDATE
WRITE
COMMIT

T₂
BEGIN
READ
VALIDATE
WRITE
COMMIT

TIME

Schedule

T₁
BEGIN
READ
VALIDATE
WRITE
COMMIT

T₂
BEGIN
READ
VALIDATE
WRITE
COMMIT

TIME
OCC – VALIDATION STEP #2

\( T_i \) completes before \( T_j \) starts its Write phase, and \( T_i \) does not write to any object read by \( T_j \).

\[ \text{WriteSet}(T_i) \cap \text{ReadSet}(T_j) = \emptyset \]
OCC – VALIDATION STEP #2

Schedule

<table>
<thead>
<tr>
<th>T₁</th>
<th>T₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN</td>
<td>BEGIN</td>
</tr>
<tr>
<td>READ R(A) W(A)</td>
<td>WRITE COMMIT</td>
</tr>
<tr>
<td>VALIDATE</td>
<td>VALIDATE</td>
</tr>
</tbody>
</table>

Database

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>∞</td>
</tr>
</tbody>
</table>

T₁ Workspace

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>∞</td>
</tr>
</tbody>
</table>

T₂ Workspace

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>0</td>
</tr>
</tbody>
</table>
OCC – VALIDATION STEP #2

Schedule

T₁
BEGIN
READ R(A)
W(A)
VALIDATE
READ R(A)
VALIDATE
WRITE
COMMIT

T₂
BEGIN

Database

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

T₁ Workspace

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>∞</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

T₂ Workspace

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

T₁ has to abort even though T₂ will never write to the database.
OCC – VALIDATION STEP #2

**Schedule**

<table>
<thead>
<tr>
<th>TIME</th>
<th>Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>T₁</td>
<td>BEGIN</td>
</tr>
<tr>
<td></td>
<td>READ</td>
</tr>
<tr>
<td></td>
<td>R(A)</td>
</tr>
<tr>
<td></td>
<td>W(A)</td>
</tr>
<tr>
<td></td>
<td>VALIDATE</td>
</tr>
<tr>
<td></td>
<td>WRITE</td>
</tr>
<tr>
<td></td>
<td>COMMIT</td>
</tr>
<tr>
<td>T₂</td>
<td>BEGIN</td>
</tr>
<tr>
<td></td>
<td>READ</td>
</tr>
<tr>
<td></td>
<td>R(A)</td>
</tr>
<tr>
<td></td>
<td>VALIDATE</td>
</tr>
<tr>
<td></td>
<td>WRITE</td>
</tr>
<tr>
<td></td>
<td>COMMIT</td>
</tr>
</tbody>
</table>

**Database**

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>0</td>
</tr>
<tr>
<td>⋯</td>
<td>⋯</td>
<td>⋯</td>
</tr>
</tbody>
</table>

**T₁ Workspace**

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>456</td>
<td>∞</td>
</tr>
<tr>
<td>⋯</td>
<td>⋯</td>
<td>⋯</td>
</tr>
</tbody>
</table>

**T₂ Workspace**

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>0</td>
</tr>
<tr>
<td>⋯</td>
<td>⋯</td>
<td>⋯</td>
</tr>
</tbody>
</table>
OCC – VALIDATION STEP #2

Schedule

T₁

BEGIN
READ
R(A)
W(A)
VALIDATE
WRITE
COMMIT

T₂

BEGIN

Database

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

T₁ Workspace

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>456</td>
<td>∞</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

T₂ Workspace

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Safe to commit T₁ because we know that T₂ will not write.
**OCC – VALIDATION STEP #3**

\[ T_i \] completes its \textbf{Read} phase before \[ T_j \] completes its \textbf{Read} phase

And \[ T_i \] does not write to any object that is either read or written by \[ T_j \]:

\[ \rightarrow \text{WriteSet}(T_i) \cap \text{ReadSet}(T_j) = \emptyset \]

\[ \rightarrow \text{WriteSet}(T_i) \cap \text{WriteSet}(T_j) = \emptyset \]
OCC – VALIDATION STEP #3

Schedule

<table>
<thead>
<tr>
<th></th>
<th>T₁</th>
<th>T₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>READ</td>
<td>R(A)</td>
<td></td>
</tr>
<tr>
<td>W(A)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VALIDATE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRITE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMMIT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Database

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>XYZ</td>
<td>0</td>
</tr>
</tbody>
</table>

T₁ Workspace

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>456</td>
<td>∞</td>
</tr>
</tbody>
</table>

T₂ Workspace

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>XYZ</td>
<td>0</td>
</tr>
</tbody>
</table>
**OCC – VALIDATION STEP #3**

**Schedule**

<table>
<thead>
<tr>
<th>T₁</th>
<th>T₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>\begin{itemize} \item \textit{BEGIN} \item \textit{READ} R(A) \item \textit{W(A)} \item \textit{VALIDATE} \item \textit{WRITE} \item \textit{COMMIT} \end{itemize}</td>
<td>\begin{itemize} \item \textit{BEGIN} \item \textit{READ} R(A) \item \textit{R(B)} \item \textit{VALIDATE} \item \textit{WRITE} \item \textit{COMMIT} \end{itemize}</td>
</tr>
</tbody>
</table>

**Database**

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>XYZ</td>
<td>0</td>
</tr>
</tbody>
</table>

**T₁ Workspace**

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>456</td>
<td>∞</td>
</tr>
</tbody>
</table>

**T₂ Workspace**

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>XYZ</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ \text{TS}(T₁) = 1 \]

*Safe to commit T₁ because T₂ sees the DB after T₁ has executed.*
**OCC – VALIDATION STEP #3**

**Schedule**
- $T_1$:
  - BEGIN
  - READ
  - R(A)
  - W(A)
  - VALIDATE
  - WRITE
  - COMMIT
- $T_2$:
  - BEGIN
  - R(A)
  - VALIDATE
  - WRITE
  - COMMIT

**Database**
- **Workspace $T_1$**
  - A: 456, W-TS: 1
  - B: XYZ, W-TS: 0

- **Workspace $T_2$**
  - A: 456, W-TS: $\infty$
  - B: XYZ, W-TS: 0

- $TS(T_1)=1$
OCC – VALIDATION STEP #3

**Schedule**

- **T₁**
  - BEGIN
  - **READ**
  - **W(A)**
  - **VALIDATE**
  - **WRITE**
  - **COMMIT**

- **T₂**
  - BEGIN
  - **READ**
  - **W(A)**
  - **WRITE**
  - **R(A)**
  - **VALIDATE**
  - **COMMIT**

**Database**

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>456</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>XYZ</td>
<td>0</td>
</tr>
</tbody>
</table>

**T₂ Workspace**

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>XYZ</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
OCC – VALIDATION STEP #3

Schedule

\[ T_1 \]
BEGIN
READ
R(A)
W(A)
VALIDATE
WRITE
COMMIT

\[ T_2 \]
BEGIN
READ
R(B)
VALIDATE
WRITE
COMMIT

Database

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>456</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>XYZ</td>
<td>0</td>
</tr>
</tbody>
</table>

T₂ Workspace

<table>
<thead>
<tr>
<th>Object</th>
<th>Value</th>
<th>W-TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>XYZ</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>456</td>
<td>1</td>
</tr>
</tbody>
</table>
OCC – OBSERVATIONS

OCC works well when the # of conflicts is low:
→ All txns are read-only (ideal).
→ Txns access disjoint subsets of data.

If the database is large and the workload is not skewed, then there is a low probability of conflict, so again locking is wasteful.
OCC – PERFORMANCE ISSUES

High overhead for copying data locally.

Validation/Write phase bottlenecks.

Aborts are more wasteful than in 2PL because they only occur after a txn has already executed.
OBSERVATION

When a txn commits, all previous T/O schemes check to see whether there is a conflict with concurrent txns.
→ This requires latches.

If you have a lot of concurrent txns, then this is slow even if the conflict rate is low.
PARTITION-BASED T/O

Split the database up in disjoint subsets called *horizontal partitions* (aka shards).

Use timestamps to order txns for serial execution at each partition.
→ Only check for conflicts between txns that are running in the same partition.
CREATE TABLE customer ( 
  c_id INT PRIMARY KEY, 
  c_email VARCHAR UNIQUE, 
  ...
);

CREATE TABLE orders ( 
  o_id INT PRIMARY KEY, 
  o_c_id INT REFERENCES customer (c_id), 
  ...
);

CREATE TABLE oitems ( 
  oi_id INT PRIMARY KEY, 
  oi_o_id INT REFERENCES orders (o_id), 
  oi_c_id INT REFERENCES orders (o_c_id), 
  ...
);
HORIZONTAL PARTITIONING

Application Server

Partitions

BEGIN

Customers 1-1000

Customers 1001-2000
HORIZONTAL PARTITIONING

Application Server

Partitions

CUSTOMERS
ORDERS
OITEMS

CUSTOMERS
ORDERS
OITEMS

Customers 1-1000

Customers 1001-2000

COMMIT
PARTITION-BASED T/O

Txns are assigned timestamps based on when they arrive at the DBMS.

Partitions are protected by a single lock:
→ Each txn is queued at the partitions it needs.
→ The txn acquires a partition’s lock if it has the lowest timestamp in that partition’s queue.
→ The txn starts when it has all of the locks for all the partitions that it will read/write.
PARTITION-BASED T/O – READS

Txns can read anything that they want at the partitions that they have locked.

If a txn tries to access a partition that it does not have the lock, it is aborted + restarted.
PARTITION-BASED T/O — WRITES

All updates occur in place.
→ Maintain a separate in-memory buffer to undo changes if the txn aborts.

If a txn tries to write to a partition that it does not have the lock, it is aborted + restarted.
PARTITION-BASED T/O

Server #1

Server #2

Txn Queue

Partitions

Customers 1-1000

Customers 1001-2000

CUSTOMERS

ORDERS

OITEMS

CUSTOMERS

ORDERS

OITEMS
PARTITION-BASED T/O

Server #1

Server #2

BEGIN

BEGIN

Txn Queue

Server1: 100
Server2: 101

Partitions

Customers
1-1000

Customers
1001-2000
PARTITION-BASED T/O

BEGIN

Server #1

Server #2

BEGIN

Txn Queue

Server1: 100
Server2: 101

Partitions

Customers
1-1000

Customers
1001-2000

Server1: 100
Server2: 101

Txn #100
PARTITION-BASED T/O

Server #1: Get C_ID=1

Server #2: BEGIN

Txn Queue:

Server1: 100
Server2: 101

Partitions:

- Customers 1-1000
- Customers 1001-2000

Txn #100: Get C_ID=1
PARTITION-BASED T/O

Server #1

Server #2

BEGIN

COMMIT

Server1:
Server2:

Txn Queue

Server1: 100
Server2: 101

Partitions

CUSTOMERS
ORDERS
OITEMS

CUSTOMERS
ORDERS
OITEMS

Txn #100

Customers 1-1000

Customers 1001-2000
PARTITION-BASED T/O

Server #1

Server #2

BEGIN

Txn Queue

Server2: 101

Partitions

Customers 1-1000

Customers 1001-2000
PARTITION-BASED T/O

Server #1

BEGIN

Server #2

Txn Queue

Server2: 101

Partitions

Customers
1-1000

Customers
1001-2000

Txn #101

CUSTOMERS
ORDERS
OITEMS

CUSTOMERS
ORDERS
OITEMS
Partition-based T/O protocol is fast if:

→ The DBMS knows what partitions the txn needs before it starts.
→ Most (if not all) txns only need to access a single partition.

Multi-partition txns causes partitions to be idle while txn executes.
Recall that so far we have only been dealing with transactions that read and update data.

But now if we have insertions, updates, and deletions, we have new problems...
THE PHANTOM PROBLEM

CREATE TABLE people (id SERIAL, name VARCHAR, age INT, status VARCHAR);

BEGIN
    SELECT MAX(age) FROM people WHERE status='lit'
    INSERT INTO people (age=96, status='lit')
    SELECT MAX(age) FROM people WHERE status='lit'
    COMMIT
END

BEGIN
    SELECT MAX(age) FROM people WHERE status='lit'
    COMMIT

72
96
WTF?

How did this happen?
→ Because $T_1$ locked only existing records and not ones under way!

Conflict serializability on reads and writes of individual items guarantees serializability only if the set of objects is fixed.
PREDICATE LOCKING

Lock records that satisfy a logical predicate:
→ Example: `status='lit'`

In general, predicate locking has a lot of locking overhead.

Index locking is a special case of predicate locking that is potentially more efficient.
INDEX LOCKING

If there is a dense index on the status field then the txn can lock index page containing the data with status='lit'.

If there are no records with status='lit', the txn must lock the index page where such a data entry would be, if it existed.
LOCKING WITHOUT AN INDEX

If there is no suitable index, then the txn must obtain:
→ A lock on every page in the table to prevent a record’s status='lit' from being changed to lit.
→ The lock for the table itself to prevent records with status='lit' from being added or deleted.
REPEATING SCANS

An alternative is to just re-execute every scan again when the txn commits and check whether it gets the same result.
→ Have to retain the scan set for every range query in a txn.
→ Andy doesn't know of any commercial system that does this (only just Silo?).
WEAKER LEVELS OF ISOLATION

Serializability is useful because it allows programmers to ignore concurrency issues. But enforcing it may allow too little concurrency and limit performance. We may want to use a weaker level of consistency to improve scalability.
ISOLATION LEVELS

Controls the extent that a txn is exposed to the actions of other concurrent txns.

Provides for greater concurrency at the cost of exposing txns to uncommitted changes:
→ Dirty Reads
→ Unrepeatable Reads
→ Phantom Reads
**ISOLATION LEVELS**

**SERIALIZABLE**: No phantoms, all reads repeatable, no dirty reads.

**REPEATABLE READS**: Phantoms may happen.

**READ COMMITTED**: Phantoms and unrepeatable reads may happen.

**READ UNCOMMITTED**: All of them may happen.
<table>
<thead>
<tr>
<th>Isolation Levels</th>
<th>Dirty Read</th>
<th>Unrepeatable Read</th>
<th>Phantom</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serializable</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Repeatable Read</td>
<td>No</td>
<td>No</td>
<td>Maybe</td>
</tr>
<tr>
<td>Read Committed</td>
<td>No</td>
<td>Maybe</td>
<td>Maybe</td>
</tr>
<tr>
<td>Read Uncommitted</td>
<td>Maybe</td>
<td>Maybe</td>
<td>Maybe</td>
</tr>
</tbody>
</table>
ISOLATION LEVELS

**SERIALIZABLE**: Obtain all locks first; plus index locks, plus strict 2PL.

**REPEATABLE READS**: Same as above, but no index locks.

**READ COMMITTED**: Same as above, but S locks are released immediately.

**READ UNCOMMITTED**: Same as above, but allows dirty reads (no S locks).
You set a txn's isolation level *before* you execute any queries in that txn.

Not all DBMS support all isolation levels in all execution scenarios → Replicated Environments

The default depends on implementation…
## Isolation Levels (2013)

<table>
<thead>
<tr>
<th>Database</th>
<th>Default</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actian Ingres 10.0/10S</td>
<td><strong>SERIALIZABLE</strong></td>
<td><strong>SERIALIZABLE</strong></td>
</tr>
<tr>
<td>Aerospike</td>
<td>READ COMMITTED</td>
<td>READ COMMITTED</td>
</tr>
<tr>
<td>Greenplum 4.1</td>
<td>READ COMMITTED</td>
<td><strong>SERIALIZABLE</strong></td>
</tr>
<tr>
<td>MySQL 5.6</td>
<td>REPEATABLE READS</td>
<td><strong>SERIALIZABLE</strong></td>
</tr>
<tr>
<td>MemSQL 1b</td>
<td>READ COMMITTED</td>
<td>READ COMMITTED</td>
</tr>
<tr>
<td>MS SQL Server 2012</td>
<td>READ COMMITTED</td>
<td><strong>SERIALIZABLE</strong></td>
</tr>
<tr>
<td>Oracle 11g</td>
<td>READ COMMITTED</td>
<td><strong>SNAPSHOT ISOLATION</strong></td>
</tr>
<tr>
<td>Postgres 9.2.2</td>
<td>READ COMMITTED</td>
<td><strong>SERIALIZABLE</strong></td>
</tr>
<tr>
<td>SAP HANA</td>
<td>READ COMMITTED</td>
<td><strong>SERIALIZABLE</strong></td>
</tr>
<tr>
<td>ScaleDB 102</td>
<td>READ COMMITTED</td>
<td>READ COMMITTED</td>
</tr>
<tr>
<td>VoltDB</td>
<td><strong>SERIALIZABLE</strong></td>
<td><strong>SERIALIZABLE</strong></td>
</tr>
</tbody>
</table>

Source: Peter Bailis
SQL-92 ACCESS MODES

You can provide hints to the DBMS about whether a txn will modify the database during its lifetime.

Only two possible modes:

→ READ WRITE (Default)
→ READ ONLY

Not all DBMSs will optimize execution if you set a txn to in READ ONLY mode.
CONCLUSION

Every concurrency control can be broken down into the basic concepts that I've described in the last two lectures.

I'm not showing benchmark results because I don't want you to get the wrong idea.
NEXT CLASS

Multi-Version Concurrency Control