Distributed Transactions
Distributed DBMS Model

transactions

data manager

network

physical database

transaction manager
Serialization

\[ T_1 : \bigcirc \bigcirc \bigcirc \bigcirc \bigcirc \]
\[ T_2 : \square \square \square \]
\[ T_3 : \bigtriangleup \bigtriangleup \bigtriangleup \bigtriangleup \]

concurrent execution

log:

\[ \square \bigcirc \bigtriangleup \bigtriangleup \square \square \bigcirc \bigtriangleup \bigtriangleup \bigcirc \bigcirc \bigcirc \bigtriangleup \]

DB
Transactions

Serialization

\[ T_1 : \begin{array}{c} O \\ O \\ O \\ O \end{array} \]
\[ T_2 : \begin{array}{c} \square \\ \square \end{array} \]
\[ T_3 : \begin{array}{c} \triangle \triangle \triangle \triangle \end{array} \]

concurrent execution

log :
\[ \begin{array}{c} \square \\ O \\ \triangle \triangle \square \square \end{array} \]
\[ \begin{array}{c} O \end{array} \]
\[ \begin{array}{c} \triangle \triangle \triangle \triangle \end{array} \]

DB

OPERATIONS

READ(X): read any one copy of X

\[ R_1 (X_3) \]

WRITE (Z): write all copies of Z

\[ W_3(Z_2) \text{ and } W_3(Z_3) \]
Serialization

T_1:  O O O O
T_2:  □ □ □ □
T_3:  △ △ △ △

concurrent execution

log:

DB is acceptable if it is guaranteed to have resulted from any one of:

T_1  T_2  T_3
T_2  T_1  T_3
T_2  T_3  T_1
T_1  T_3  T_2
T_3  T_1  T_2
T_3  T_2  T_1
Serialization

Consider two concurrent transactions executed at only one DM

LOG: $R_1(X) \ R_2(Y) \ R_1(Y) \ W_1(Z) \ W_1(X) \ W_2(X) \ R_2(Z)$
Serialization

Consider two concurrent transactions executed at only one DM

**LOG:** \( R_1(X) \ R_2(Y) \ R_1(Y) \ W_1(Z) \ W_1(X) \ W_2(X) \ R_2(Z) \)

**Serial Order:**

\( R_2(Y) \ W_2(X) \ R_2(Z) \); \( R_1(X) \ R_1(Y) \ W_1(Z) \ W_1(X) \)
Serialization

Consider two concurrent transactions executed at only one DM

LOG: \( R_1(X) \ R_2(Y) \ R_1(Y) \ W_1(Z) \ W_1(X) \ W_2(X) \ R_2(Z) \)

Serial Order:

\( R_2(Y) \ W_2(X) \ R_2(Z) \); \( R_1(X) \ R_1(Y) \ W_1(Z) \ W_1(X) \)

1. last write conflict
2. read source conflict
Serialization

Consider two concurrent transactions executed at only one DM

LOG: \( R_1(X) \ R_2(Y) \ R_1(Y) \ W_1(Z) \ W_1(X) \ W_2(X) \ R_2(Z) \)

Serial Order:
\( R_1(X) \ R_1(Y) \ W_1(Z) \ W_1(X) \ ; \ R_2(Y) \ W_2(X) \ R_2(Z) \)
Serialization

Consider two concurrent transactions executed at only one DM

LOG: \[ R_1(X) \quad R_2(Y) \quad R_1(Y) \quad W_1(Z) \quad W_1(X) \quad W_2(X) \quad R_2(Z) \]

Serial Order: 
\[ R_1(X) \quad R_1(Y) \quad W_1(Z) \quad W_1(X) \quad ; \quad R_2(Y) \quad W_2(X) \quad R_2(Z) \]
Distributed Transaction Processing

Transactions:

T_1 : READ(X); WRITE(Y);

T_2 : READ(Y); WRITE(Z);

T_3 : READ(Z); WRITE(X);
Transactions:

\[ T_1 : \text{READ}(X); \text{WRITE}(Y); \]
\[ T_2 : \text{READ}(Y); \text{WRITE}(Z); \]
\[ T_3 : \text{READ}(Z); \text{WRITE}(X); \]

LOGS:

\[ L_1 : R_2(Y_1) \ R_1(X_1) \ W_1(Y_1) \ W_3(X_1) \]
\[ L_2 : R_3(Z_2) \ W_2(Z_2) \ W_1(Y_2) \]
\[ L_3 : W_3(X_3) \ W_2(Z_3) \]
Distributed Transaction Processing

Transactions:

\[ T_1 : \text{READ}(X); \text{WRITE}(Y); \]
\[ T_2 : \text{READ}(Y); \text{WRITE}(Z); \]
\[ T_3 : \text{READ}(Z); \text{WRITE}(X); \]

LOGS:

\[ L_1 : \text{R}_2(Y_1) \; \text{R}_1(X_1) \; \text{W}_1(Y_1) \; \text{W}_3(X_1) \]
\[ L_2 : \text{R}_3(Z_2) \; \text{W}_2(Z_2) \; \text{W}_1(Y_2) \]
\[ L_3 : \text{W}_3(X_3) \; \text{W}_2(Z_3) \]

Question: Are these logs equivalent to some serial execution of the transactions?
Serialization of Distributed Logs

Conflict: $P_j(A_x)$ and $Q_i(B_y)$ conflict if

1. $P$ and $Q$ are not both READ, and
2. $A = B$ (same data item), and
3. $i \neq j$ (different transactions), and
4. $X = Y$ (same data manager/log)
Serialization of Distributed Logs

Conflict: $P_j(A_X)$ and $Q_i(B_Y)$ conflict if

1. $P$ and $Q$ are not both READ, and
2. $A = B$, and
3. $i \neq j$, and
4. $X = Y$

LOGS: $L_1 : R_2(Y_1) R_1(X_1) W_1(Y_1) W_3(X_1)$

$L_2 : R_3(Z_2) W_2(Z_2) W_1(Y_2)$

$L_3 : W_3(X_3) W_2(Z_3)$
Serialization of Distributed Logs

Conflict: \( P_j(A_X) \) and \( Q_i(B_Y) \) conflict if

1. \( P \) and \( Q \) are not both READ, and
2. \( A = B \), and
3. \( i \neq j \), and
4. \( X = Y \)

LOGS:

1. \( L_1 : R_2(Y_1) \ R_1(X_1) \ W_1(Y_1) \ W_3(X_1) \)
2. \( L_2 : R_3(Z_2) \ W_2(Z_2) \ W_1(Y_2) \)
3. \( L_3 : W_3(X_3) \ W_2(Z_3) \)
Serialization of Distributed Logs

Conflict: $P_j(A_X)$ and $Q_i(B_Y)$ conflict if

1. $P$ and $Q$ are not both READ, and
2. $A = B$, and
3. $i \neq j$, and
4. $X = Y$

LOGS:

$L_1: R_2(Y_1) \quad R_1(X_1) \quad W_1(Y_1) \quad W_3(X_1)$
$L_2: R_3(Z_2) \quad W_2(Z_2) \quad W_1(Y_2)$
$L_3: W_3(X_3) \quad W_2(Z_3)$

Contradictory
\[ \therefore \text{No total order} \]
\[ \therefore \text{Not serializable} \]
Serialization of Distributed Logs

**Theorem:** Distributed logs are serializable if there exists a total ordering of the transactions such that for conflicting operations $P_j$ and $Q_i$ a log shows $P_j \rightarrow Q_i$ only if $T_j \rightarrow T_i$

LOGS:

- $L_1 : R_2(Y_1) \ R_1(X_1) \ W_1(Y_1) \ W_3(X_1)$
- $L_2 : R_3(Z_2) \ W_2(Z_2) \ W_1(Y_2)$
- $L_3 : W_3(X_3) \ W_2(Z_3)$

Contradictory

∴ No total order
∴ Not serializable
Locking

- transactions must use Two Phase Locking (2PL)

**locking phase**  
**release phase**

- only the following lock requests are granted

<table>
<thead>
<tr>
<th>lock request</th>
<th>current lock state</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>not locked</td>
</tr>
<tr>
<td>READ</td>
<td>OK</td>
</tr>
<tr>
<td>WRITE</td>
<td>OK</td>
</tr>
</tbody>
</table>
Transactions

**Locking**

- request lock before accessing a data item
- release all locks at the end of transaction

This guarantees serializability [ESWAREN]
Transactions

**Effects of Locking**

Suppose the transactions have executed to this point:

- \( L_1 : R_2(Y_1) \)
- \( L_2 : R_3(Z_2) \)
- \( L_3 : W_3(X_3) \)

The locks are then:

<table>
<thead>
<tr>
<th>Lock for</th>
<th>Lock state</th>
<th>Waiting for lock</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>write-locked by ( T_3 )</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>read-locked by ( T_2 )</td>
<td>T_1</td>
</tr>
<tr>
<td>Z</td>
<td>read-locked by ( T_3 )</td>
<td>T_2</td>
</tr>
</tbody>
</table>

Only \( T_3 \) is able to execute; it will complete its write and release its locks, leading to …
Effects of Locking

$L_1 : R_2(Y_1), W_3(X_1)$

$L_2 : R_3(Z_2)$

$L_3 : W_3(X_3)$

...these logs, and...

...this lock state:

<table>
<thead>
<tr>
<th>Lock for</th>
<th>Lock state</th>
<th>Waiting for lock</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>unlocked</td>
<td>$T_1$</td>
</tr>
<tr>
<td>Y</td>
<td>read-locked by $T_2$</td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>unlocked</td>
<td>$T_2$</td>
</tr>
</tbody>
</table>

Both $T_1$ and $T_2$ can acquire the locks needed to execute their next actions, resulting in...
Effects of Locking

L₁ : R₂(Y₁), W₃(X₁), R₁(X₁)

...these logs, and...

L₂ : R₃(Z₂), W₂(Z₂)

L₃ : W₃(X₃), W₂(Z₃)

...this lock state:

<table>
<thead>
<tr>
<th>Lock for</th>
<th>Lock state</th>
<th>Waiting for lock</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>read-locked by T₁</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>read-locked by T₂</td>
<td>T₁</td>
</tr>
<tr>
<td>Z</td>
<td>write-locked by T₂</td>
<td></td>
</tr>
</tbody>
</table>

Transaction T₂ completes, releases its lock - resulting in...
Effects of Locking

...this lock state:

<table>
<thead>
<tr>
<th>Lock for</th>
<th>Lock state</th>
<th>Waiting for lock</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>read-locked by T₁</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>unlocked</td>
<td>T₁</td>
</tr>
<tr>
<td>Z</td>
<td>unlocked</td>
<td></td>
</tr>
</tbody>
</table>

At this point, T₁ can acquire the write-lock on Y, perform its write operations and complete, leading to the final serializable logs:

\[ L₁ : R₂(Y₁), W₃(X₁), R₁(X₁), W₁(Y₁) \]
\[ L₂ : R₃(Z₂), W₂(Z₂), W₁(Y₂) \]
\[ L₃ : W₃(X₃), W₂(Z₃) \]