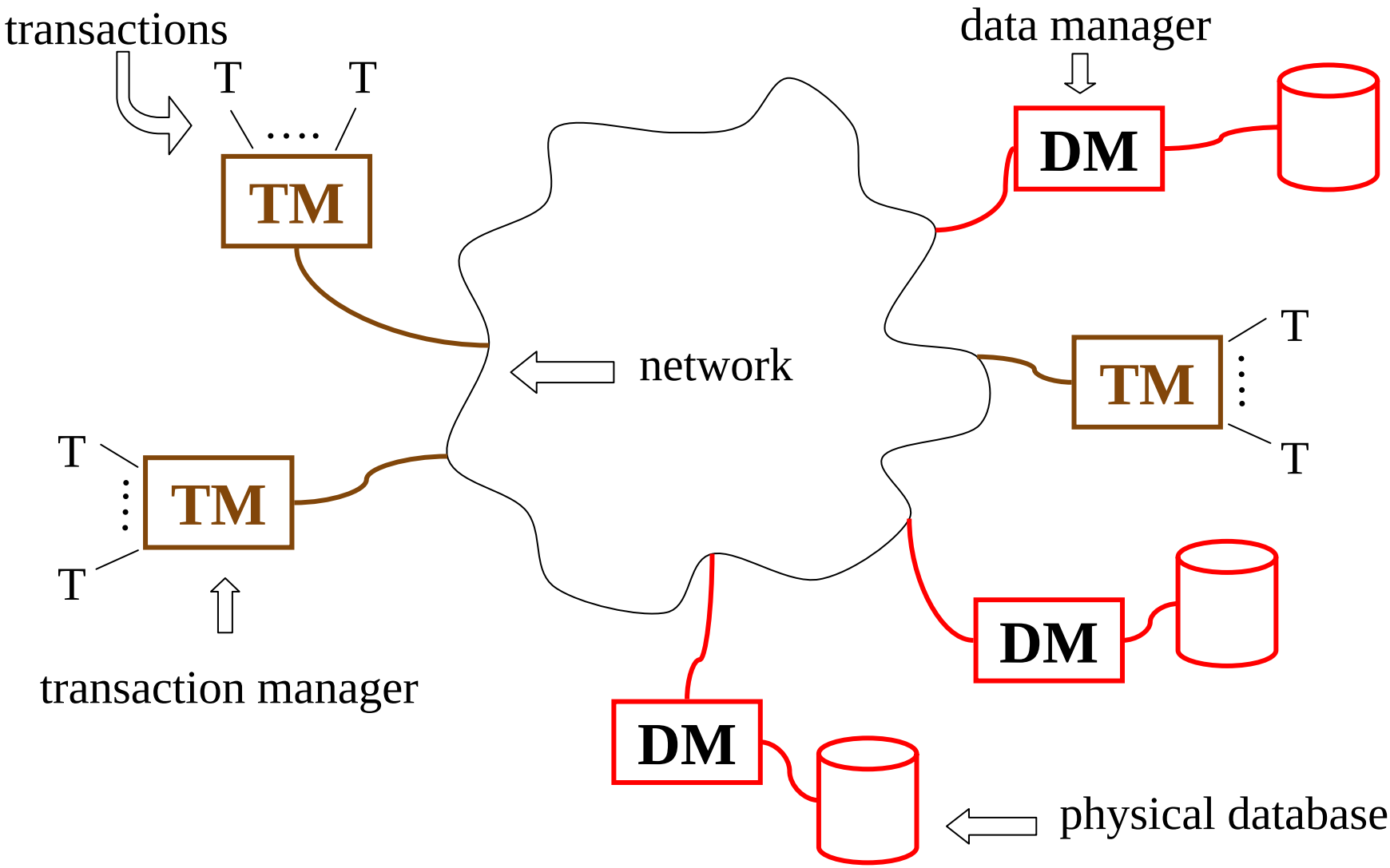
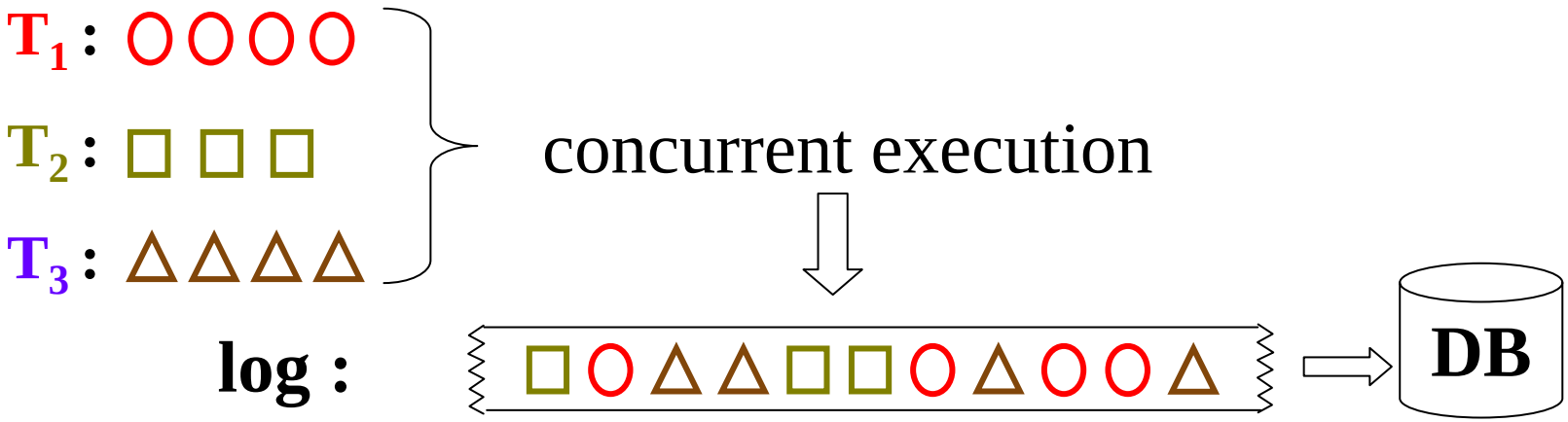
Distributed Transactions

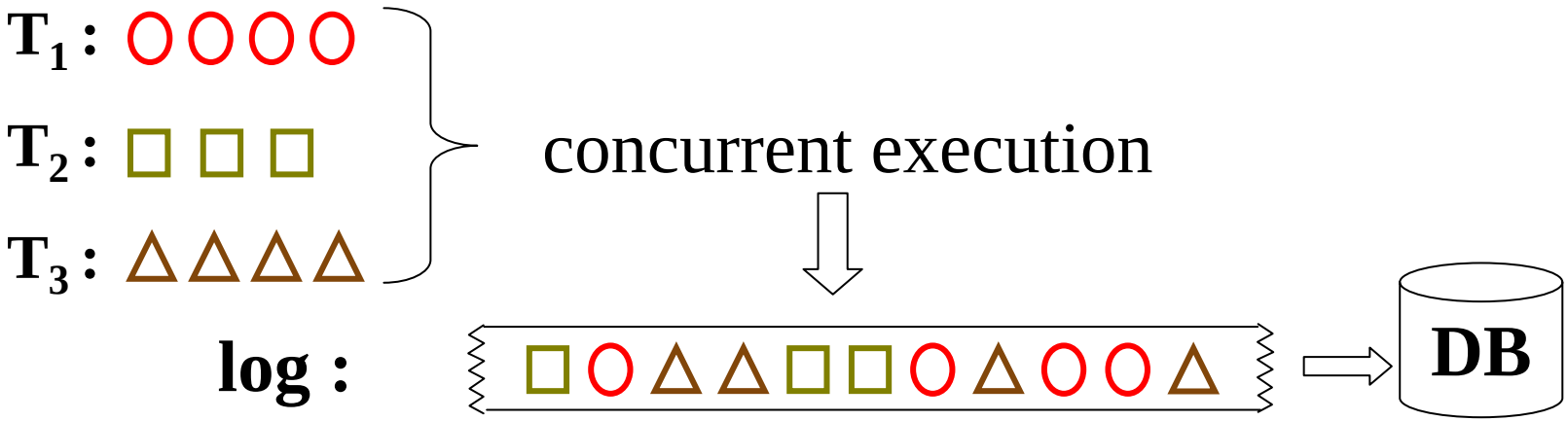
Distributed DBMS Model



Serialization



Serialization



OPERATIONS

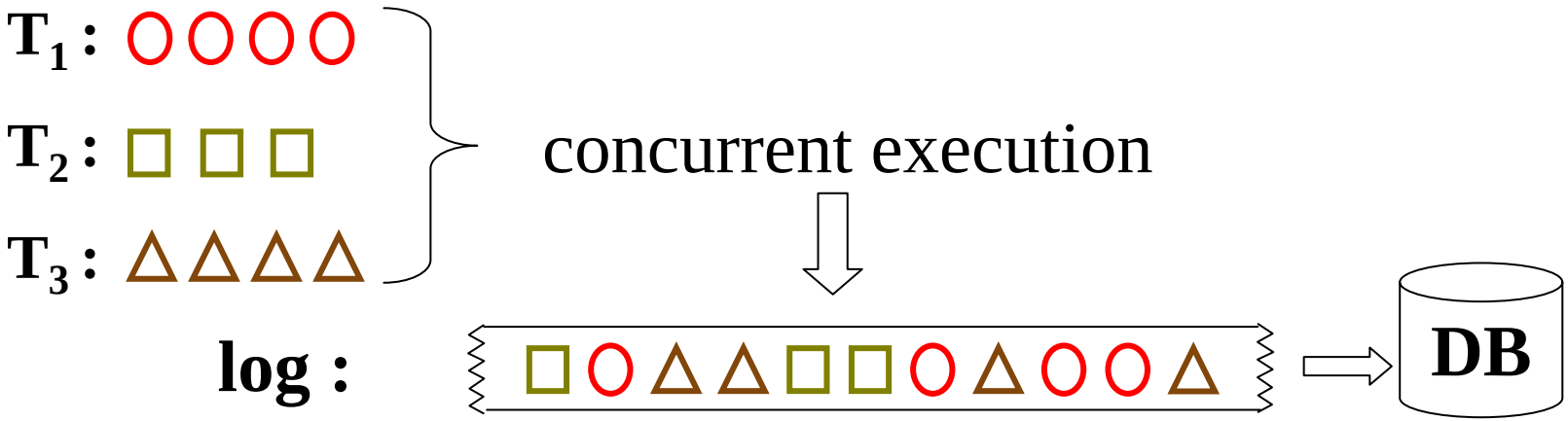
READ(X): read any one copy of X

$R_1 (X_3)$

WRITE (Z): write all copies of Z

$W_3(Z_2)$ and $W_3 (Z_3)$

Serialization



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

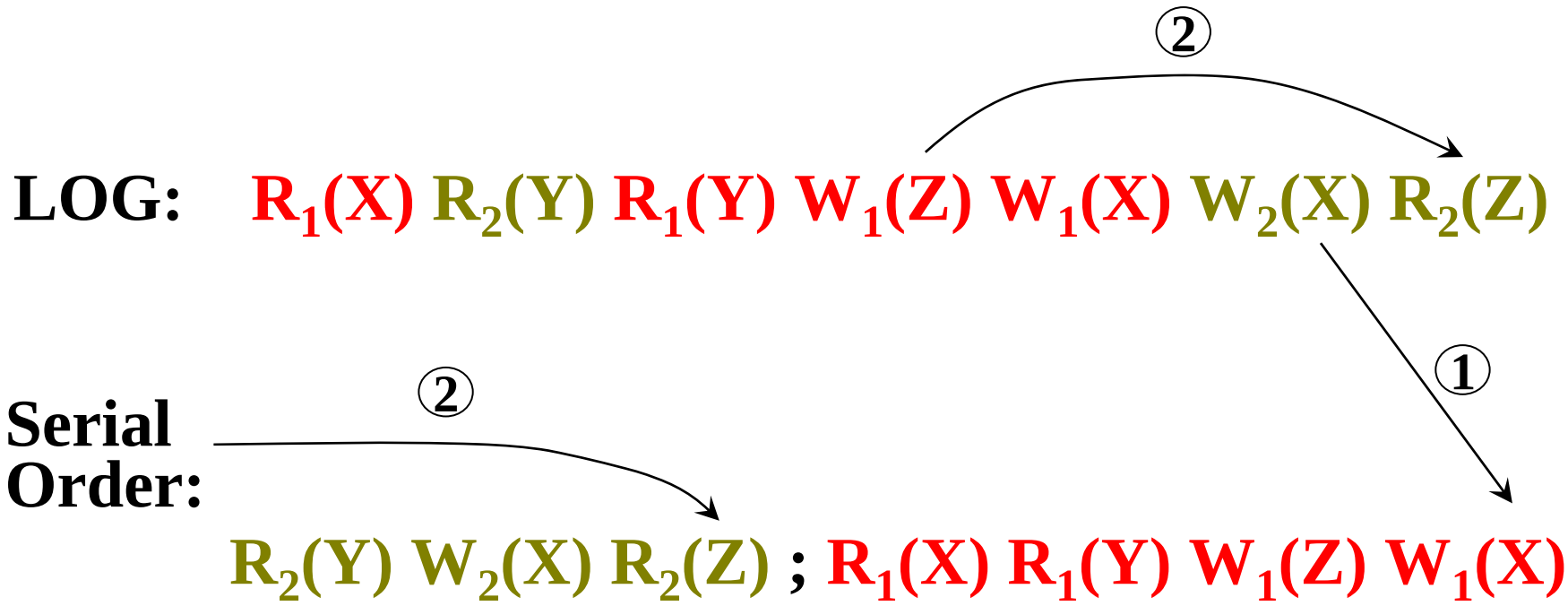
LOG: $R_1(X)\ R_2(Y)\ R_1(Y)\ W_1(Z)\ W_1(X)\ W_2(X)\ R_2(Z)$

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



① **last write conflict**

② **read source conflict**

Serialization

Consider two concurrent transactions executed at only one DM

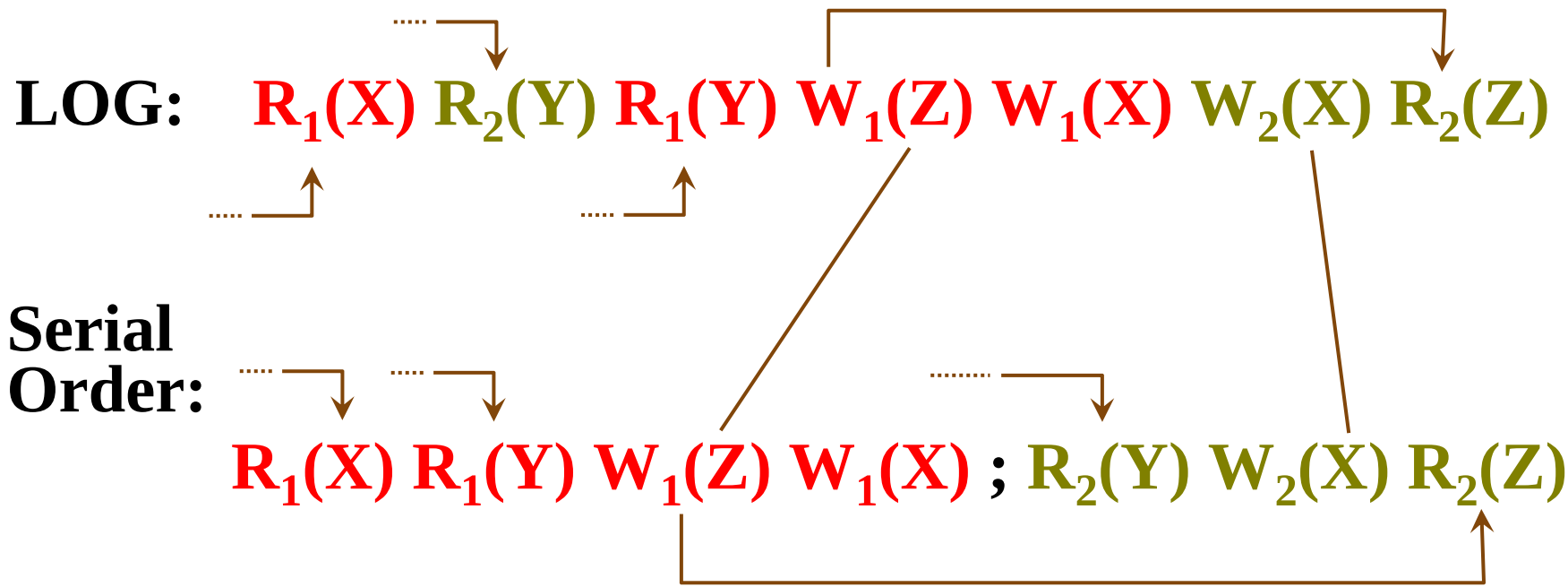
LOG: $R_1(X)\ R_2(Y)\ R_1(Y)\ W_1(Z)\ W_1(X)\ W_2(X)\ R_2(Z)$

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



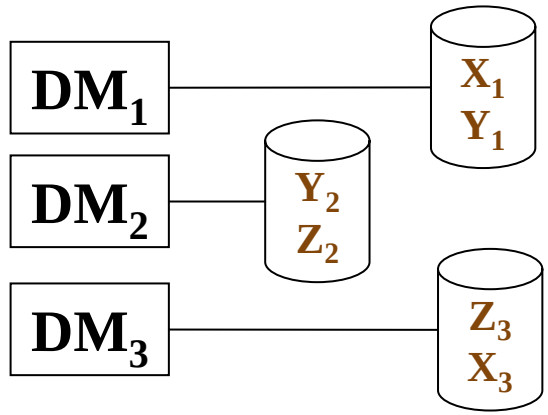
Distributed Transaction Processing

Transactions:

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

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

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



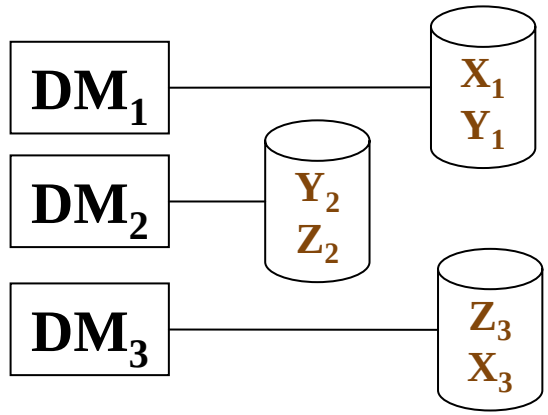
Distributed Transaction Processing

Transactions:

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

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

T_3 : READ(Z); 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 : $W_3(X_3)$ $W_2(Z_3)$

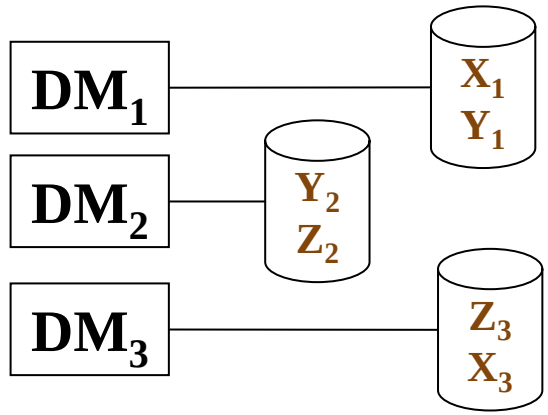
Distributed Transaction Processing

Transactions:

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

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

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



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

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

$L_3 : W_3(X_3)\ W_2(Z_3)$

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

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

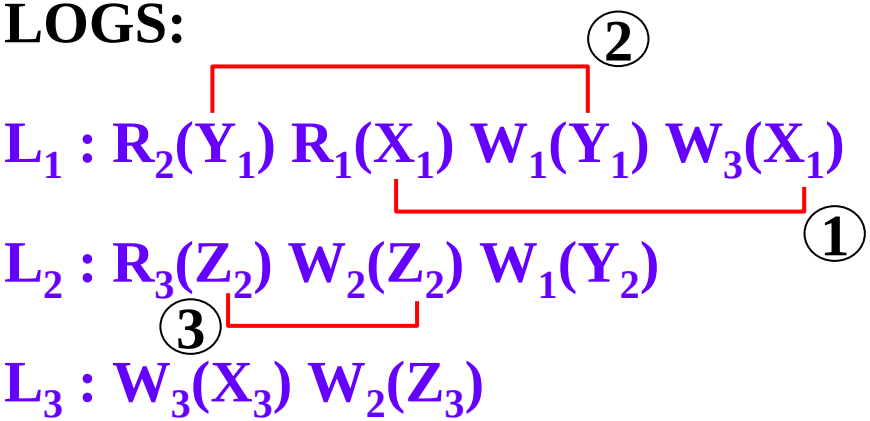
$L_3 : W_3(X_3)\ W_2(Z_3)$

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:

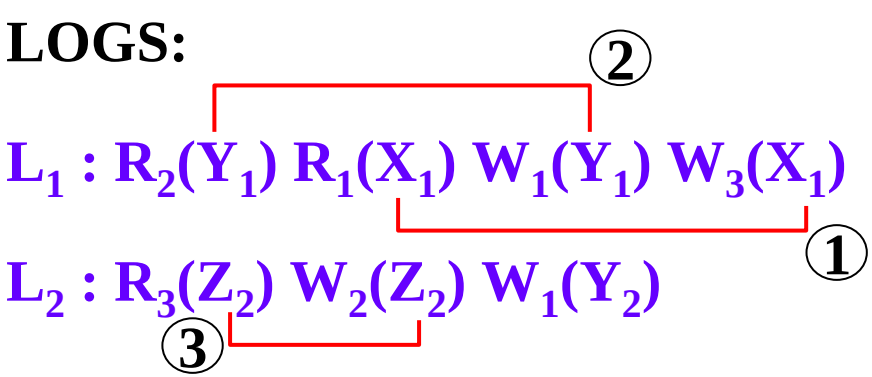


Serialization of Distributed Logs

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:



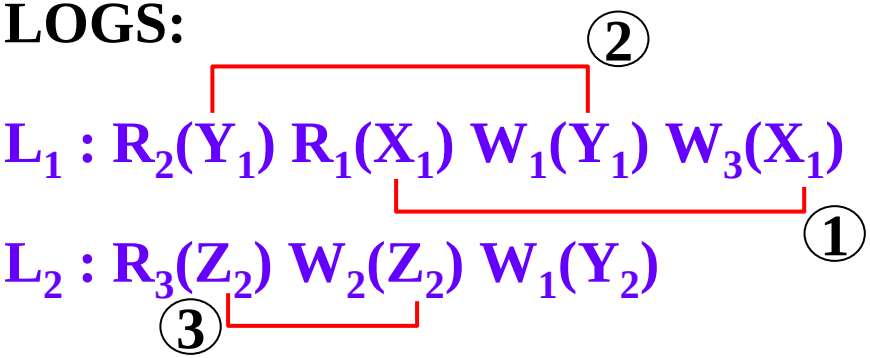
① => $T_1 \rightarrow T_3$
② => $T_2 \rightarrow T_1$
③ => $T_3 \rightarrow T_2$

Contradictory
$\therefore$ No total order
$\therefore$ 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:



① => $T_1 \rightarrow T_3$

② => $T_2 \rightarrow T_1$

③ => $T_3 \rightarrow T_2$

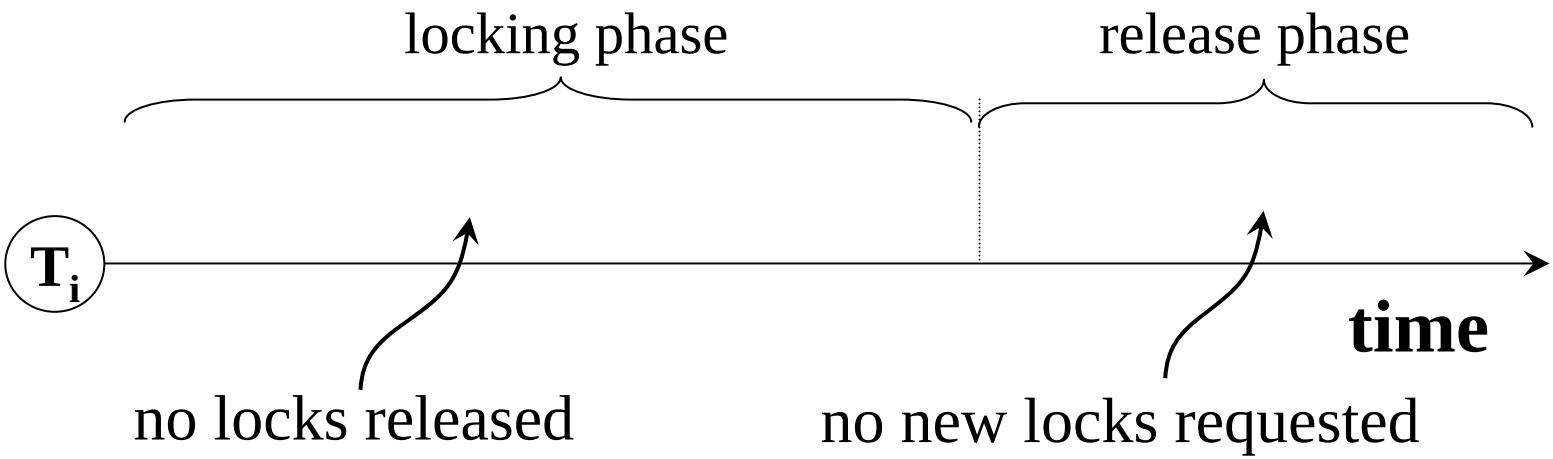
Contradictory

∴ No total order

∴ Not serializable

Locking

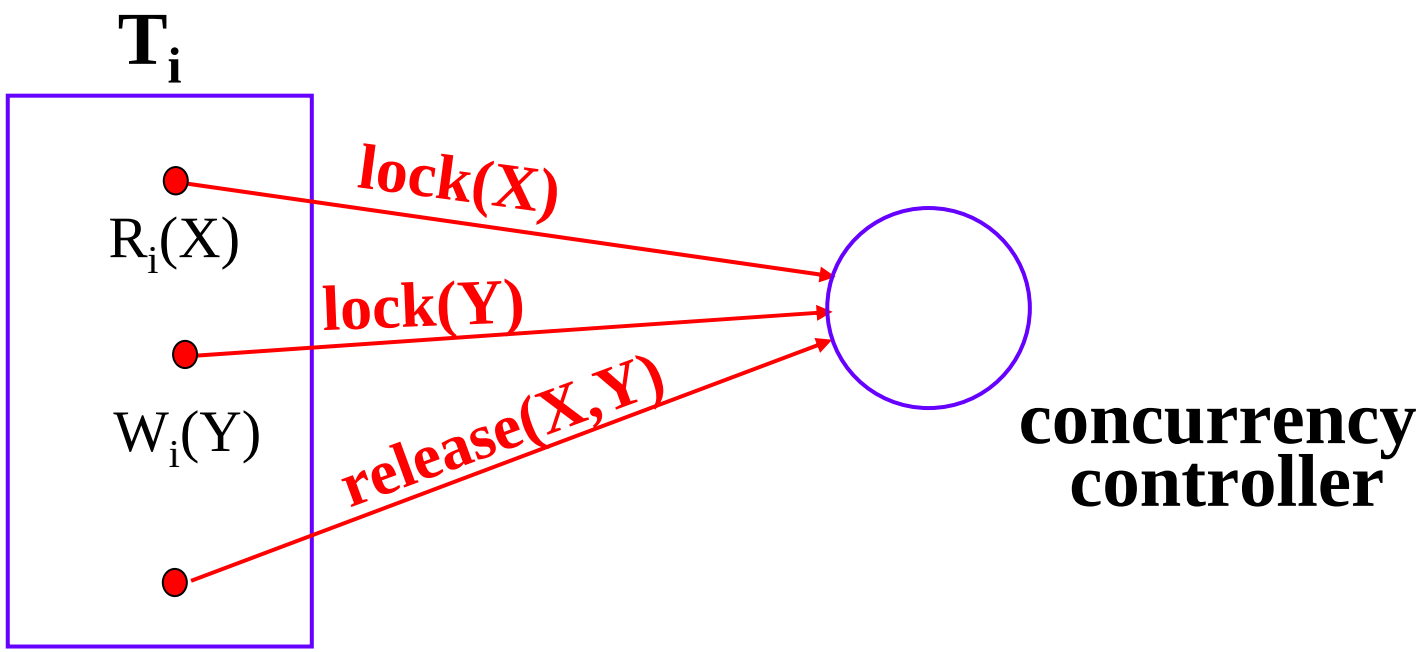
- **transactions must use Two Phase Locking (2PL)**



- **only the following lock requests are granted**

	current lock state		
lock request	not locked	READ locked	WRITE locked
READ	OK	OK	DENY
WRITE	OK	DENY	DENY

Locking



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

This guarantees serializability [ESWAREN]

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:

Lock for	Lock state	Waiting for lock
X	write-locked by T_3	T_1
Y	read-locked by T_2	
Z	read-locked by T_3	T_2

Only T_3 is able to execute; it will complete its write and release its locks, leading to …

Effects of Locking

…these logs, and…

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

$L_2 : R_3(Z_2)$

$L_3 : W_3(X_3)$

…this lock state:

Lock for	Lock state	Waiting for lock
X	unlocked	T_1
Y	read-locked by T_2	
Z	unlocked	T_2

Both T_1 and T_2 can acquire the locks needed to execute their next actions, resulting in…

Effects of Locking

…these logs, and…

$L_1 : R_2(Y_1), W_3(X_1), R_1(X_1)$

$L_2 : R_3(Z_2), W_2(Z_2)$

$L_3 : W_3(X_3), W_2(Z_3)$

…this lock state:

Lock for	Lock state	Waiting for lock
X	read-locked by T_1	
Y	read-locked by T_2	T_1
Z	write-locked by T_2	

Transaction T_2 completes, releases its lock - resulting in…

Effects of Locking

…this lock state:

Lock for	Lock state	Waiting for lock
X	read-locked by T_1	
Y	unlocked	T_1
Z	unlocked	

At this point, T_1 can acquire the write-lock on Y, perform its write operations and complete, leading to the final serializable logs:

$$\mathbf{L_1 : R_2(Y_1),\ W_3(X_1),\ R_1(X1),\ W_1(Y_1)}$$

$$\mathbf{L_2 : R_3(Z_2),\ W_2(Z_2),\ W_1(Y_2)}$$

$$\mathbf{L_3 : W_3(X_3),\ W_2(Z_3)}$$