

View serializable

- ▶ Every **Conflict Serializable** schedule is also **View Serializable**.
- ▶ A **View Serializable** schedule **may be** a **Conflict Serializable** schedule.
- ▶ Below is a schedule which is View-Serializable but not Conflict Serializable.

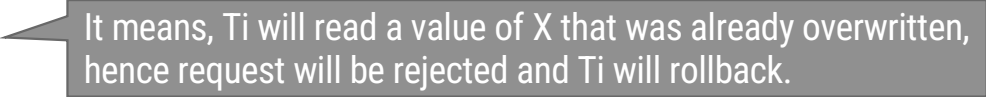
T_{27}	T_{28}	T_{29}
read (Q)	write (Q)	
write (Q)		write (Q)

- ▶ What serial schedule is above equivalent to?
- ▶ Every **View Serializable** schedule **having Blind Writes** will **not** be **Conflict Serializable**.

Time stamp based protocol

- ▶ This protocol **uses either system time or logical counter** to be used as a time-stamp.
- ▶ Typically, time-stamp values are assigned in the order in which the transactions are submitted to the system, so a time-stamp can be thought of as the transaction **start time**.
- ▶ A transaction 'T1' created at 0002 clock time would be older than all other transaction, which come after it.
 - Time-stamp of Transaction T_i is denoted as $TS(T_i)$.
- ▶ In addition, **every data item is given the latest read and write time-stamp**.
 - Read time-stamp of data-item X is denoted by $R\text{-timestamp}(X)$. It is the timestamp of the transaction who has performed $Read(X)$.
 - Write time-stamp of data-item X is denoted by $W\text{-timestamp}(X)$. It is the timestamp of the transaction who has performed $Write(X)$.
- ▶ The algorithm **allows inter-leaving** of transactions operations, but it must ensure that for each **pair of conflicting operations** in the schedule, **the order in which the item is accessed must follow the timestamp order**.

Time stamp ordering protocol

- ▶ This is the responsibility of the protocol system that the conflicting pair of tasks should be executed according to the timestamp values of the transactions.
 - ↪ Time-stamp of **Transaction** T_i is denoted as $TS(T_i)$.
 - ↪ **Read** time-stamp of **data-item** X is denoted by $R\text{-timestamp}(X)$. It is the timestamp of the transaction who has performed $Read(X)$.
 - ↪ **Write** time-stamp of **data-item** X is denoted by $W\text{-timestamp}(X)$. It is the timestamp of the transaction who has performed $Write(X)$.
- ▶ Timestamp ordering protocol works as follows:
 - ↪ If a transaction T_i requests for $read(X)$ operation:
 - If $TS(T_i) < W\text{-timestamp}(X)$  It means, T_i will read a value of X that was already overwritten, hence request will be rejected and T_i will rollback.
 - Operation rejected.
 - If $TS(T_i) \geq W\text{-timestamp}(X)$
 - Operation executed.
 - ↪ If a transaction T_i requests for $write(X)$ operation:
 - If $TS(T_i) < R\text{-timestamp}(X)$
 - Operation rejected.
 - If $TS(T_i) < W\text{-timestamp}(X)$
 - Operation rejected and T_i rolled back.
 - Otherwise, operation executed.

Time stamp ordering protocol (Example)

TS = 10	TS = 30	TS = 20
T1	T2	T3
Read (A)	Write (A) Read (B)	Write (A) Write (B)

TS = 30	TS = 20	TS = 10
T1	T2	T3
Write (A) Write (B) Read (C)	Read (B) Read (C)	Write (C) Write (A)

Thomas Write Rule

- ▶ Modified Time-Stamp Protocol to make some improvements and may generate those schedules which are View Serializable but not conflict serializable and provide better concurrency.
- ▶ It modify time-stamp protocol in obsolete write case when
 - ➔ Transaction T_i requests for Write(Q) operation, if $TS(T_i) < W\text{-timestamp}(Q)$ [Case 1].

Case 1

$TS(T_i) = 5$	$TS(T_x) = 10$
Write (Q)	Write (Q)

Case 2

$TS(T_i) = 5$	$TS(T_x) = 10$
Write (Q)	Write (Q)

- Here, T_i attempts to write obsolete value of Q. It should have been written earlier than T_x and now it is outdated.
- Hence, Write (Q) operation of T_i can be ignored.

TS = 10	TS = 20	TS = 30
T1	T2	T3
Read (Q)	Write (Q)	Write (Q)
Write (Q)		

It is view serializable
But not conflict serializable.