Concurrency Control can be implemented in different ways. One way to implement it is by using **Locks** second way **Time Stamp Ordering** Protocol.

Transaction Timestamp TS(T_i):

A timestamp is a unique identifier created by the DBMS to identify a transaction. They are usually assigned in the order in which they are submitted to the system, so a timestamp may be thought of as the transaction start time.

There may be different ways of generating timestamps such as

- A simple counter that increments each time its value is assigned to a transaction. They may be numbered ***1, 2, 3…***. Though we'll have to reset the counter from time to time to avoid overflow.

- Using the current date/time from the system clock. Just ensuring that no two transactions are given the same value in the same clock tick, we will always get a unique timestamp. This method is widely used.

Timestamp Ordering Protocol:

The main idea for this protocol is to order the transactions based on their Timestamps.

A schedule in which the transactions participate is then serializable and the only equivalent serial schedule permitted has the transactions in the order of their Timestamp Values.

Stating simply, the schedule is equivalent to the particular Serial Order corresponding to the order of the Transaction timestamps.

An algorithm must ensure that, for each item accessed by Conflicting Operations in the schedule, the order in which the item is accessed does not violate the ordering. To ensure this, use two Timestamp Values relating to each database item X.

- **W_TS(X)** is the largest timestamp of any transaction that executed write(X) successfully.
- **R_TS(X)** is the largest timestamp of any transaction that executed read(X) successfully.

Basic Timestamp Ordering:

Every transaction is issued a timestamp based on when it enters the system. Suppose, if an old transaction Ti has timestamp TS(Ti), a new transaction Tj is assigned timestamp TS(Tj) such that TS(Ti) < TS(Tj). The protocol manages concurrent execution such that the timestamps determine the serializability order. The timestamp ordering protocol ensures that any conflicting read and write operations are executed in timestamp order.

Whenever some Transaction T tries to issue a R_item(X) or a W_item(X), the Basic TO algorithm compares the timestamp of T with R_TS(X) & W_TS(X) to ensure that the Timestamp order is not violated. This describes the Basic TO protocol in the following two cases.

1. Whenever a Transaction *T* issues a **W_item(X)** operation, check the following conditions:
    - If ***R_TS(X) > TS(T)*** or if ***W_TS(X) > TS(T)***, then abort and rollback T and reject the operation. else,
    - Execute W_item(X) operation of T and set W_TS(X) to TS(T).

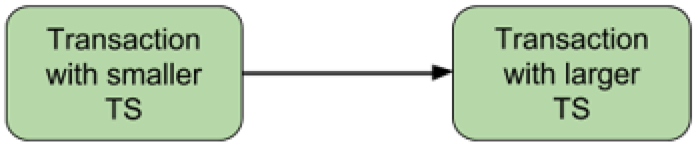
2. Whenever a Transaction *T* issues a **R_item(X)** operation, check the following conditions:
    - If ***W_TS(X) > TS(T)***, then abort and reject T and reject the operation, else
    - If W_TS(X) <= TS(T), then execute the R_item(X) operation of T and set R_TS(X) to the larger of TS(T) and current R_TS(X).

Whenever the Basic TO algorithm detects two conflicting operations that occur in an incorrect order, it rejects the latter of the two operations by aborting the Transaction that issued it. Schedules produced by Basic TO are guaranteed to be conflict serializable. Already discussed that using Timestamp can ensure that our schedule will be deadlock free.

One drawback of the Basic TO protocol is that Cascading Rollback is still possible. Suppose we have a Transaction T1 and T2 has used a value written by T1. If T1 is aborted and resubmitted to the system then, T2 must also be aborted and rolled back. So the problem of Cascading aborts still prevails.

Let's gist the Advantages and Disadvantages of Basic TO protocol:

- Timestamp Ordering protocol ensures serializability since the precedence graph will be of the form:



- Timestamp protocol ensures freedom from deadlock as no transaction ever waits.
- But the schedule may not be cascade free, and may not even be recoverable.

Strict Timestamp Ordering:

A variation of Basic TO is called Strict to ensure that the schedules are both Strict and Conflict Serializable. In this variation, a Transaction T that issues a R_item(X) or W_item(X) such that TS(T) > W_TS(X) has its read or write operation delayed until the Transaction T' that wrote the values of X has committed or aborted.

Advantages :

High Concurrency: Timestamp-based concurrency control allows for a high degree of concurrency by ensuring that transactions do not interfere with each other.

Efficient: The technique is efficient and scalable, as it does not require locking and can handle a large number of transactions.

No Deadlocks: Since there are no locks involved, there is no possibility of deadlocks occurring.

Improved Performance: By allowing transactions to execute concurrently, the overall performance of the database system can be improved.

Disadvantages:

Limited Granularity: The granularity of timestamp-based concurrency control is limited to the precision of the timestamp. This can lead to situations where transactions are unnecessarily blocked, even if they do not conflict with each other.

Timestamp Ordering: In order to ensure that transactions are executed in the correct order, the timestamps need to be carefully managed. If not managed properly, it can lead to inconsistencies in the database.

Timestamp Synchronization: Timestamp-based concurrency control requires that all transactions have synchronized clocks. If the clocks are not synchronized, it can lead to incorrect ordering of transactions.

Timestamp Allocation: Allocating unique timestamps for each transaction can be challenging, especially in distributed systems where transactions may be initiated at different locations.