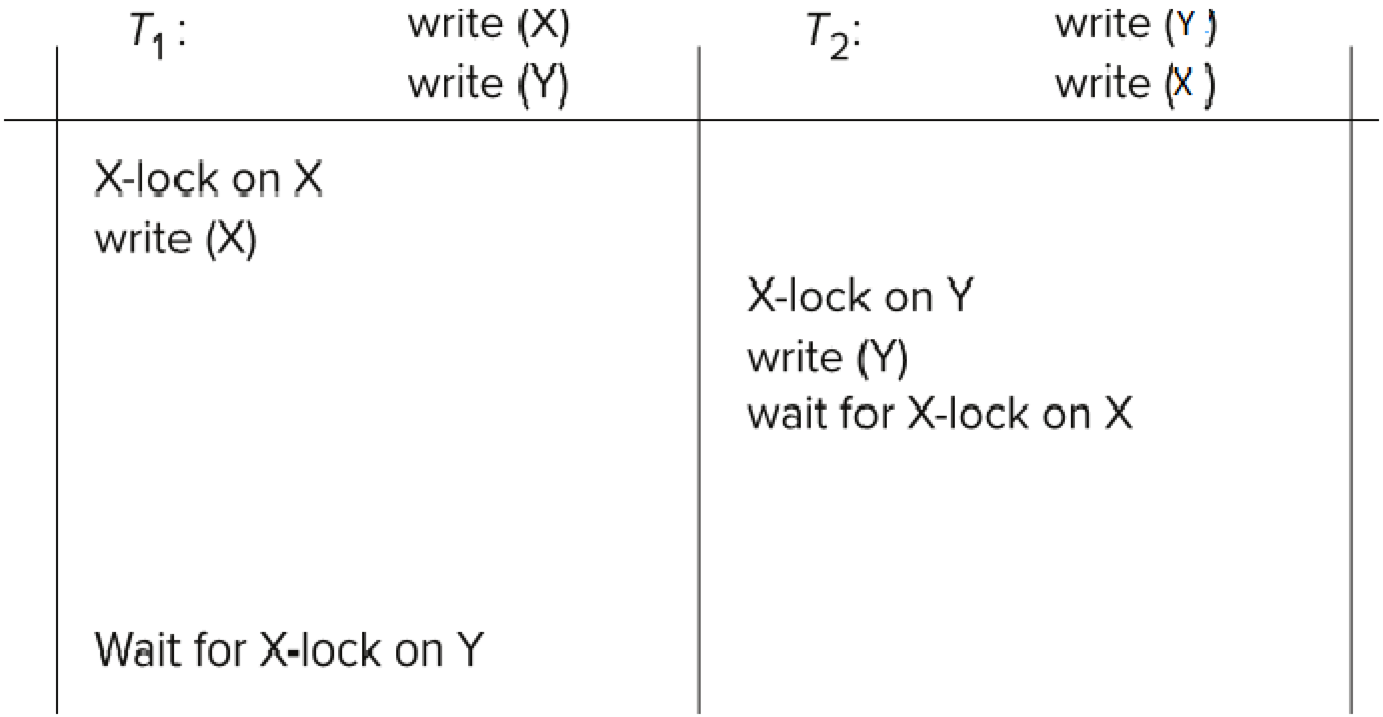
Deadlock Handling in Distributed Transactions

Deadlock Handling

Consider the following two transactions and history, with item X and transaction T_1 at site 1, and item Y and transaction T_2 at site 2:



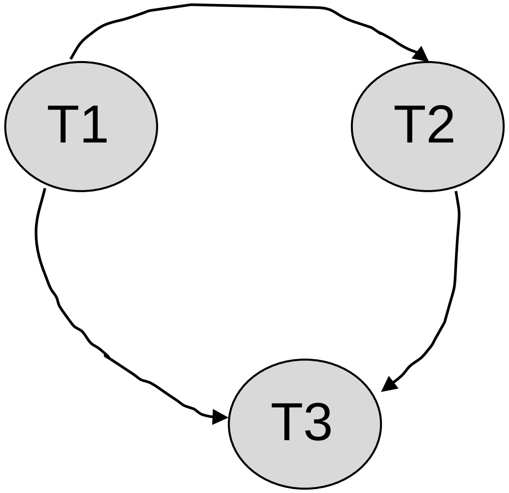
T_1 : write (X) write (Y)	T_2: write (Y) write (X)
X-lock on X write (X)	
	X-lock on Y write (Y) wait for X-lock on X
Wait for X-lock on Y	

Result: deadlock which cannot be detected locally at either site

Deadlock Detection

Wait-for Graph

- Transaction T1 request a lock for data A that is locked by T2
- Transaction T2 request a lock for data B that is locked by T3
- Transaction T1 request a lock for data B that is locked by T3



Question 27-1 Construct wait-for graph for the following.
Transaction T1 request a lock for data A that is locked by T4
Transaction T2 request a lock for data B that is locked by T4
Transaction T3 request a lock for data B that is locked by T4
Transaction T4 request a lock for data C that is locked by T1

Deadlock Detection

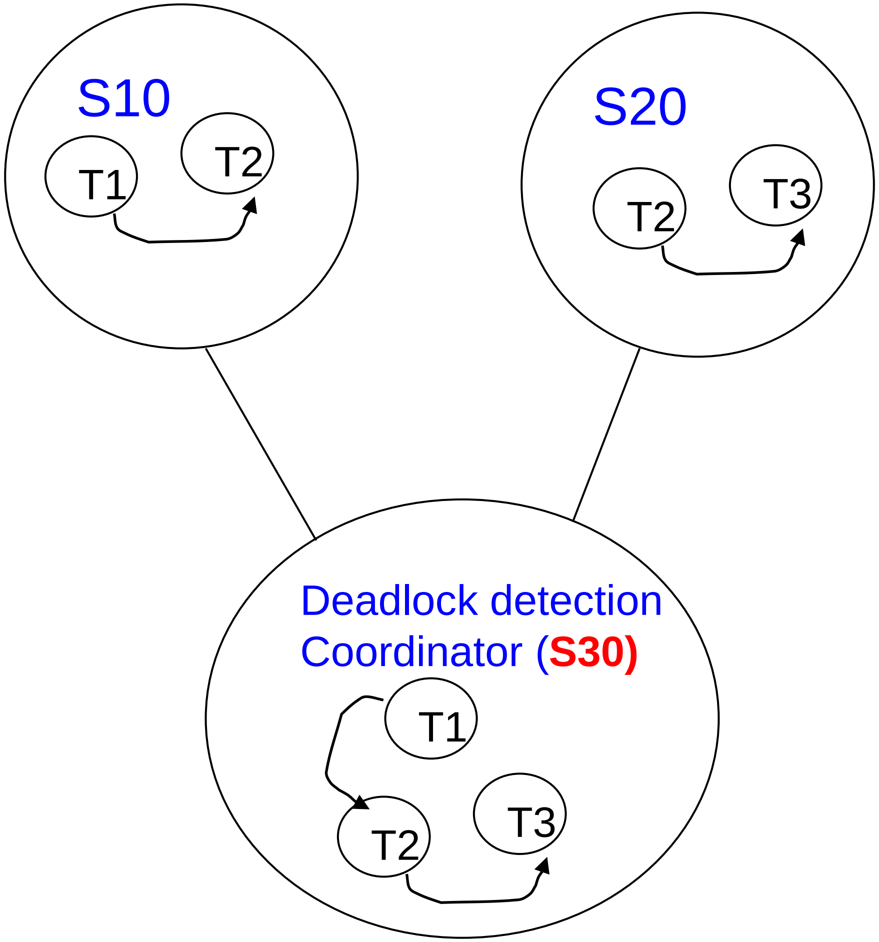
Two sites- S10 and S20

Wait for graph for S10

- Transaction T1 request a lock for data A that is locked by T2

Wait for graph for S20

- Transaction T2 request a lock for data B that is locked by T3



In the **centralized deadlock-detection** approach, a global wait-for graph is constructed and maintained in a *single* site; the deadlock-detection coordinator

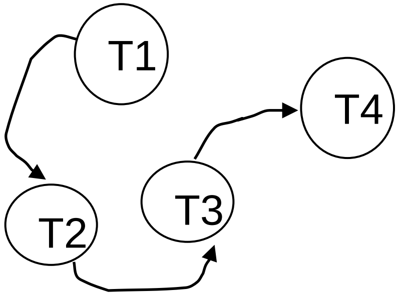
Deadlock Detection

Real graph: Real, but unknown, state of the system.

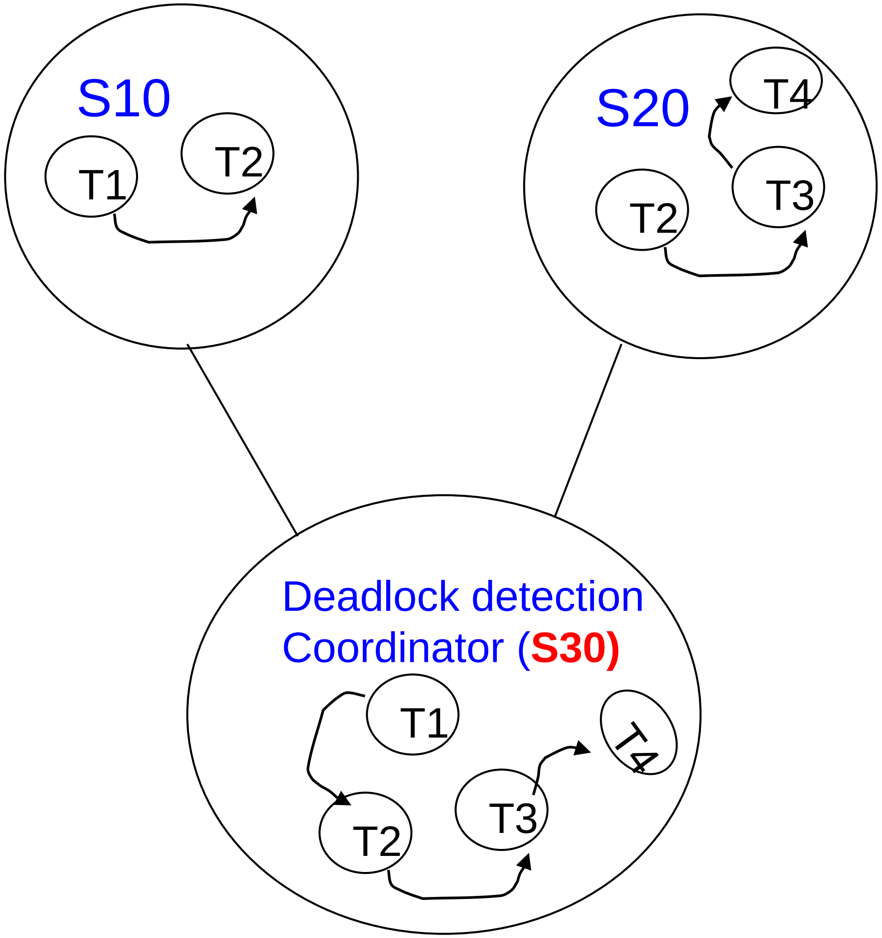
Constructed graph: Approximation generated by the controller during the execution of its algorithm .

T3 → T4 added to S20 but message has not reached to coordinator due to network delay

Real graph and constructed graph are not the same.



Real graph



Constructed graph

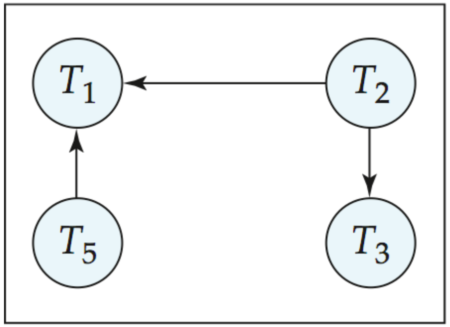
Local and Global Wait-For Graphs

The global wait-for graph can be constructed when:

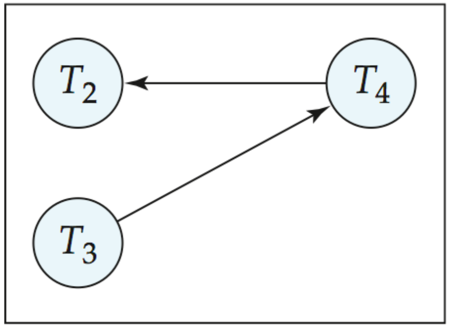
a new edge is inserted in or removed from one of the local wait-for graphs.

a number of changes have occurred in a local wait-for graph.
the coordinator needs to invoke cycle-detection.

If the coordinator finds a cycle, it selects a victim and notifies all sites. The sites roll back the victim transaction.

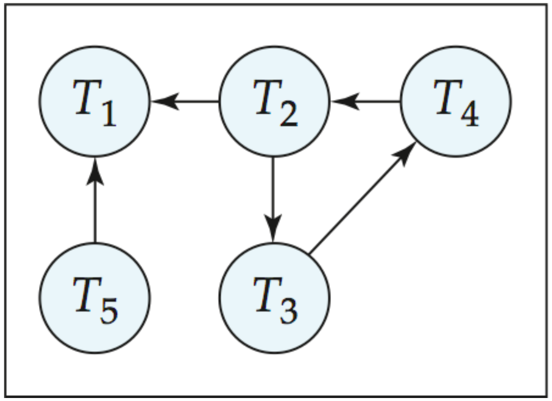


site S_1



site S_2

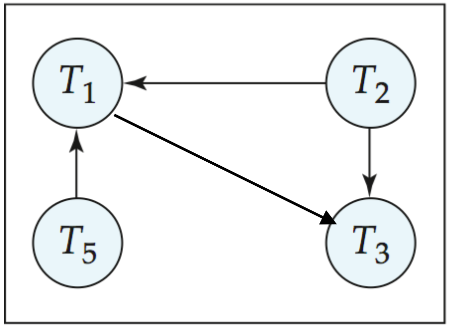
Local



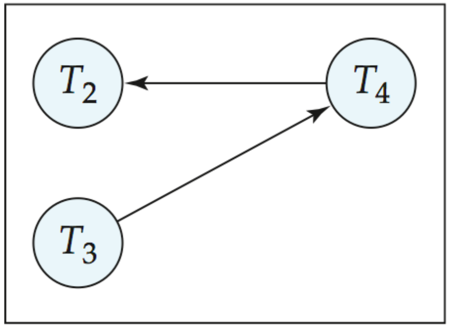
Global

Local and Global Wait-For Graphs

Question 27-2: Construct the global wait-for graph and find the deadlock status



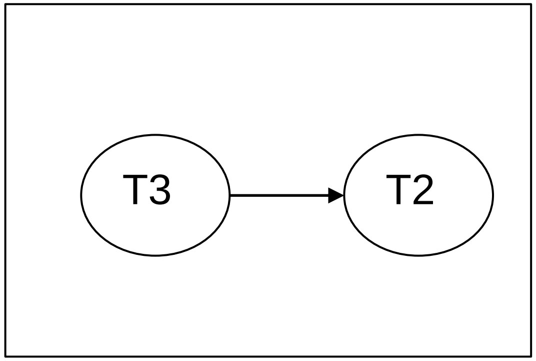
site S_1



site S_2

Cycle 1: T2→T3 → T2

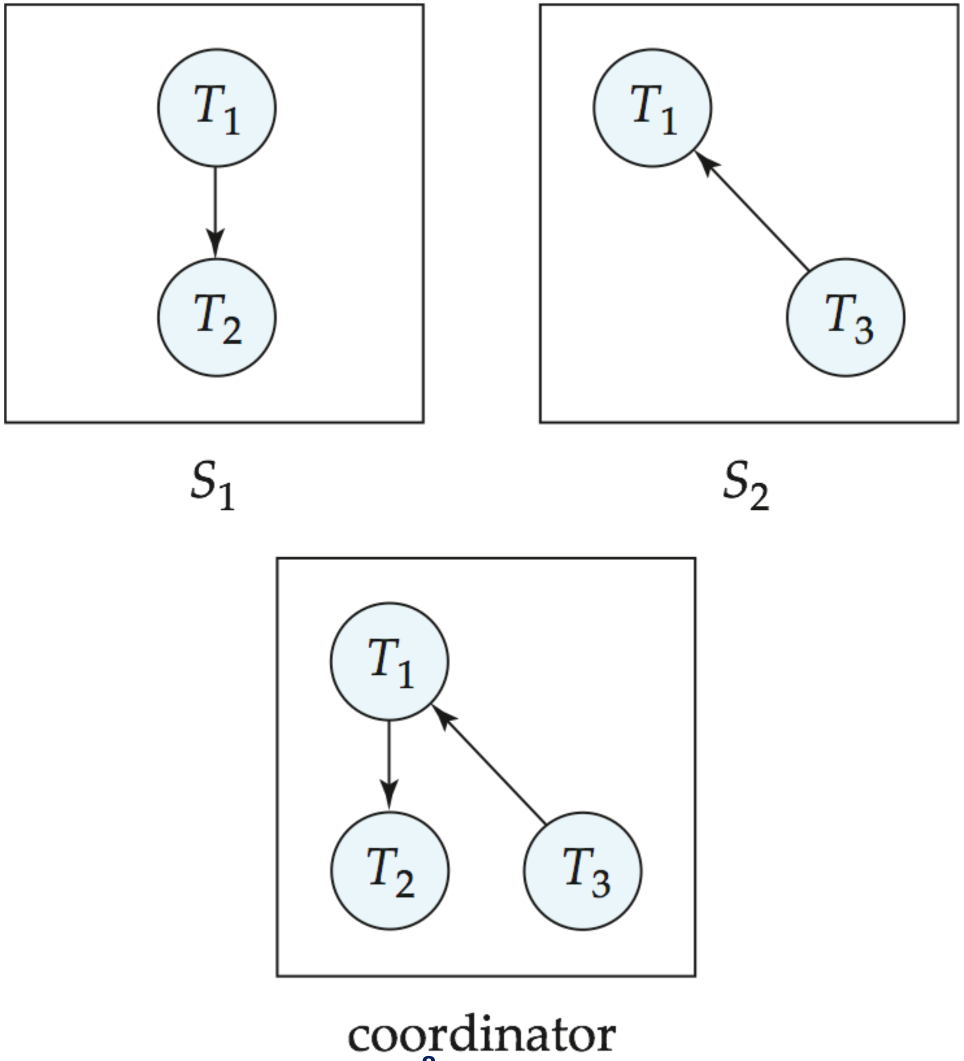
Global



Site 3

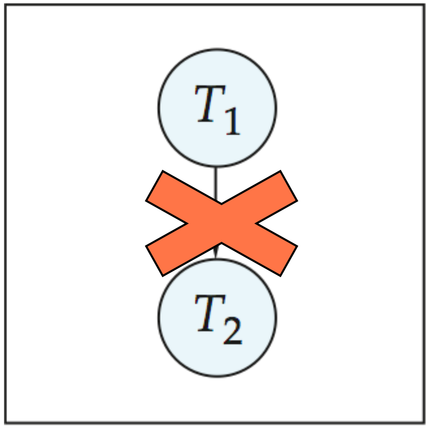
Example Wait-For Graph for False Cycles

Initial state:

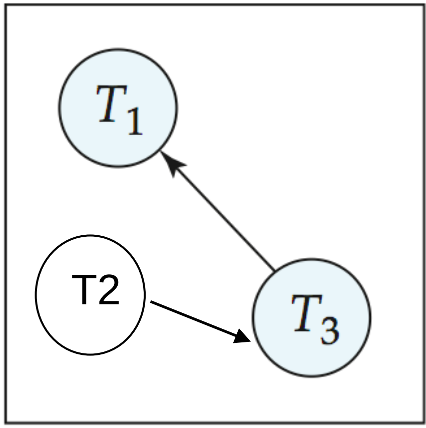


Example Wait-For Graph for False Cycles

Suppose that starting from the state shown in figure,
1. T_2 releases resources at S_1 resulting in a message remove $T_1 \rightarrow T_2$ message from the Transaction Manager at site S_1 to the coordinator)

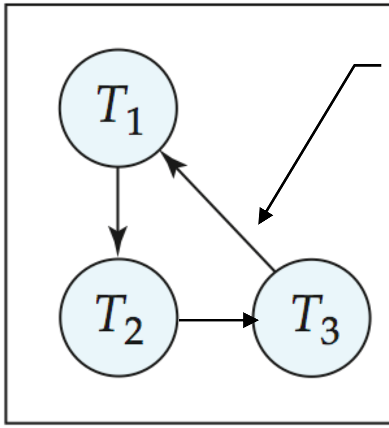


S_1



S_2

2. And then T_2 requests a resource held by T_3 at site S_2 resulting in a message insert $T_2 \rightarrow T_3$ from S_2 to the coordinator



coordinator

False Cycle

Suppose further that the insert message reaches before the **delete** message this can happen due to network delays

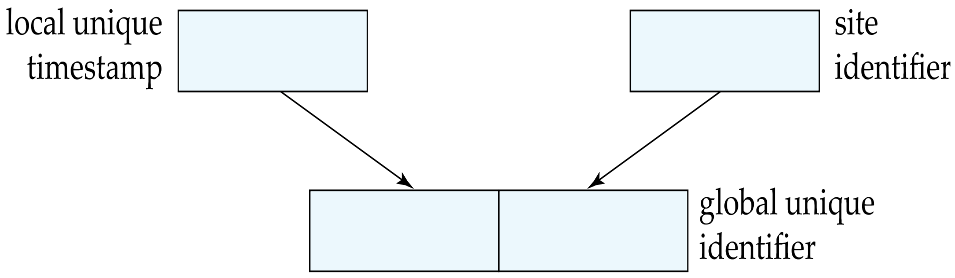
The coordinator would then find a false cycle
$T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_1$

Distributed Deadlocks

- The false cycle above never existed in reality.
- False cycles cannot occur if two-phase locking is used.
- Unnecessary rollbacks may result
  - When deadlock has indeed occurred and a victim has been picked, and meanwhile one of the transactions was aborted for reasons unrelated to the deadlock.
  - Due to false cycles in the global wait-for graph
- In the **distributed deadlock-detection** approach, sites exchange wait-for information and check for deadlocks
  - Expensive and not used in practice

Distributed Timestamp-Based Protocols

- Timestamp based concurrency-control protocols can be used in distributed systems
- Each transaction must be given a *unique* timestamp
- Main problem: how to generate a timestamp in a distributed fashion
- Each site generates a unique local timestamp using either a logical counter or the local clock.
- Global unique timestamp is obtained by concatenating the unique local timestamp with the unique identifier.

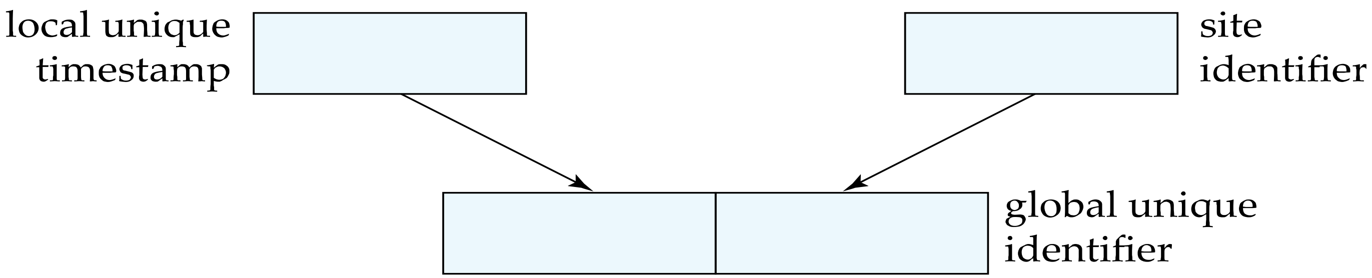


Discussion:

Why site identifier in LSB and local unique timestamp in MSB?

What would happen if it would be reverse?

Distributed Timestamp-Based Protocols



Example on Timestamp Generation

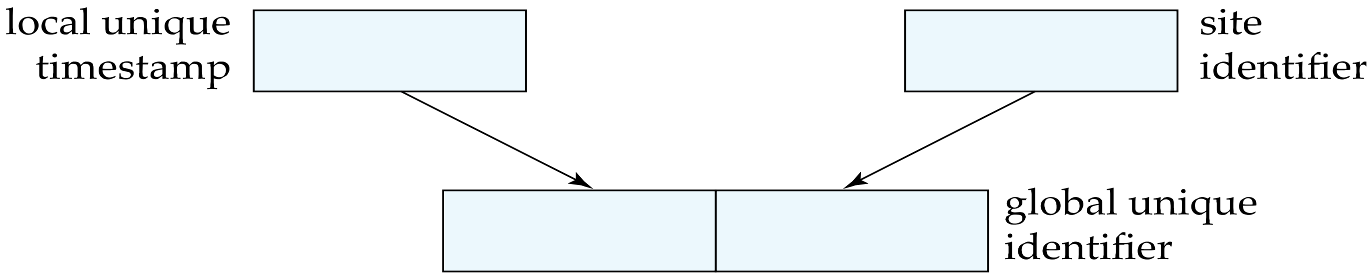
The site identifiers of S1, S2 and S3 are 10, 20 and 30 respectively. The transactions T5 is in site 2, T6 in site 3 and T7 in site 1 respectively. Local timestamp for T5, T6 and T7 are 99, 88 and 99 respectively. Find the timestamp of T5, T6 and T7

Global timestamp for T5 = T5 local timestamp + identifier of S2
= concat (99 + 20) = 9920
Global timestamp for T6 = T6 local timestamp + identifier of S3
= concat (88 + 30) = 8830
Global timestamp for T5 = T7 local timestamp + identifier of S1
= concat (99 + 10) = 9910

Question:

The site identifiers of S3, S4 and S5 are 111, 555 and 333 respectively. The transactions T5 is in S5, T6 in S4 and T7 in S3 respectively. Local timestamp for T5, T6 and T7 are 101, 101 and 102 respectively. Find the timestamp of T5, T6 and T7.

Distributed Timestamp-Based Protocols



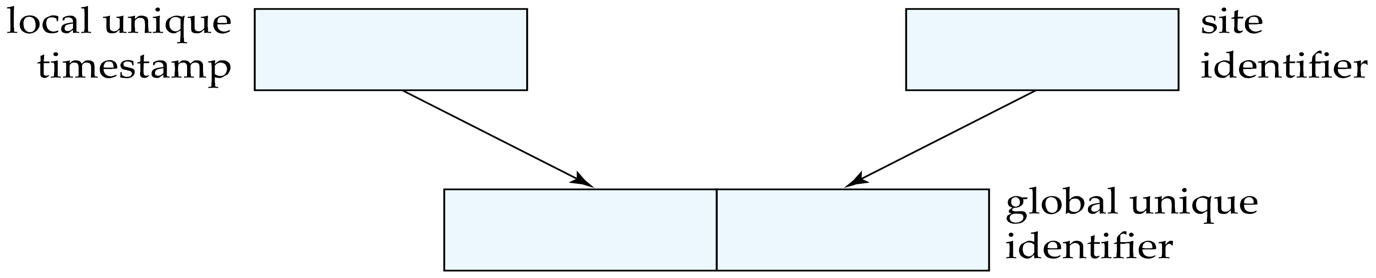
Question:

The site identifiers of S3, S4 and S5 are 111, 555 and 333 respectively. The transactions T5 is in S5, T6 in S4 and T7 in S3 respectively. Local timestamp for T5, T6 and T7 are 101, 101 and 102 respectively.
Find the timestamp of T5, T6 and T7.

Distributed Timestamps

Problem of Slow Local Clock

- A node with a slow clock will assign smaller timestamps (**HOW?)**
    - Because of local unique timestamp
- Still logically correct: serializability not affected
    - But: “disadvantages” transactions



Distributed Timestamps

To fix the problem of slow clock

- Keep clocks synchronized using network time protocol
- Or, define within each node N_i a **logical clock** (LC_i), which generates the unique local timestamp
  - Require that N_i advance its logical clock whenever a request is received from a transaction Ti with timestamp < *x,y*> and x is greater that the current value of LC_i.
  - In this case, site N_i advances its logical clock to the value $x + 1$

EXAMPLE

Site S1 has LC1 with value = 88

T2 has timestamp <99, S3> and T2 access data at S1

What will be the new value of LC1?