Concurrency Control in Distributed Databases

Dr. Mohamed Elhoseny

Outline

- Distributed Database Management system (DDBMS)
- Concurrency Control Models (CC)
- Concurrency Control Protocols
- Deadlock Management in DDBMS

Introduction

- Concurrency control is the activity of coordinating concurrent accesses to a database in a multi-user database management system (DBMS)
- Several problems
  1. *The lost update problem.*
  2. *The temporary update problem*
  3. *The incorrect summary problem*
- Serializability Theory.

Issues in DDBMS

- Data Planning
- Query Optimization and Decomposition
- Distributed Transaction Management
- Fault Tolerance and Reliability
- Networking

Transactions & Transaction Management

- ACID Property is still must be notified in DDBMS
  - Atomicity; Consistency; Isolation; Durability
- Transaction structures : Flat ; Nested

```
Begin_transaction
        T1();
        T2(); ......
End_transaction
```

```
Begin_transaction
  Begin_transaction T1
    Begin_transaction T2
          T3(); ......
    End_transaction T2
  End_transaction T1
End_transaction
```

Introduction … cont.

- A **transaction** is a logical unit of database processing that includes one or more database access operations (read : retrieval or write: insertion, deletion, modification)
- The transaction boundaries
  - **begin transaction** and **end transaction**
- An **application program** may contain several transactions separated by the transaction boundaries.
- The basic database access operations
  - **read_item(*X*).** Reads a database item named *X* into a program variable.
  - **write_item(*X*).** Writes the value of program variable *X* into the database item named *X*.

Two sample transactions

(a)

T_1
read_item(X);
$X := X - N$;
write_item(X);
read_item(Y);
$Y := Y + N$;
write_item(Y);

(b)

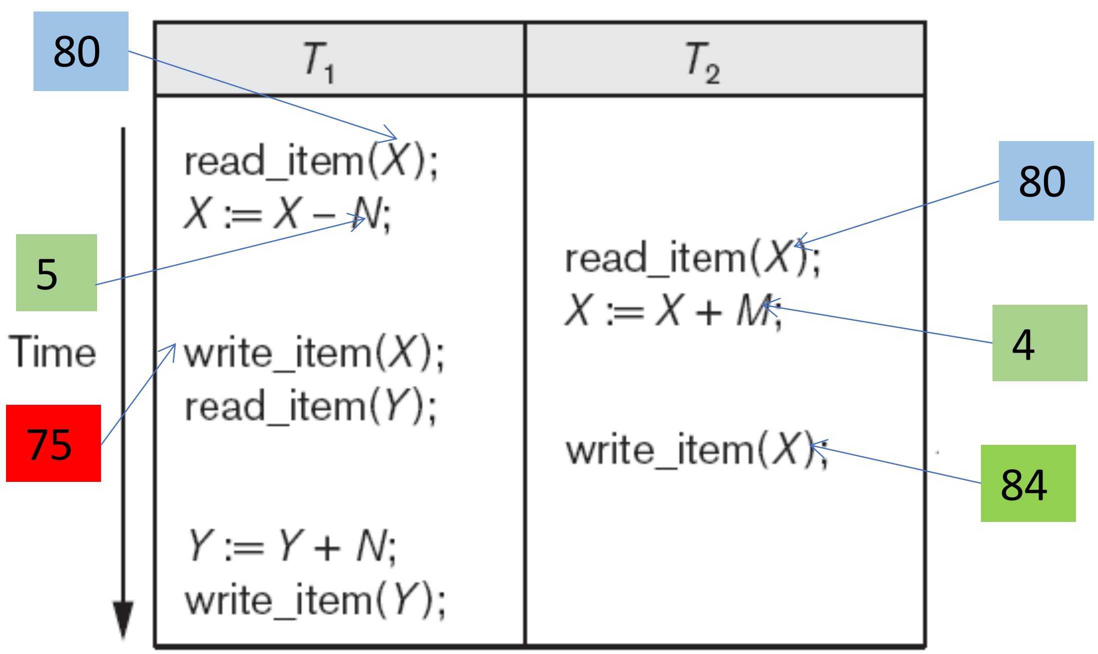
T_2
read_item(X);
$X := X + M$;
write_item(X);

- Two sample transactions.

  (a) Transaction *T*1. (b) Transaction *T*2.

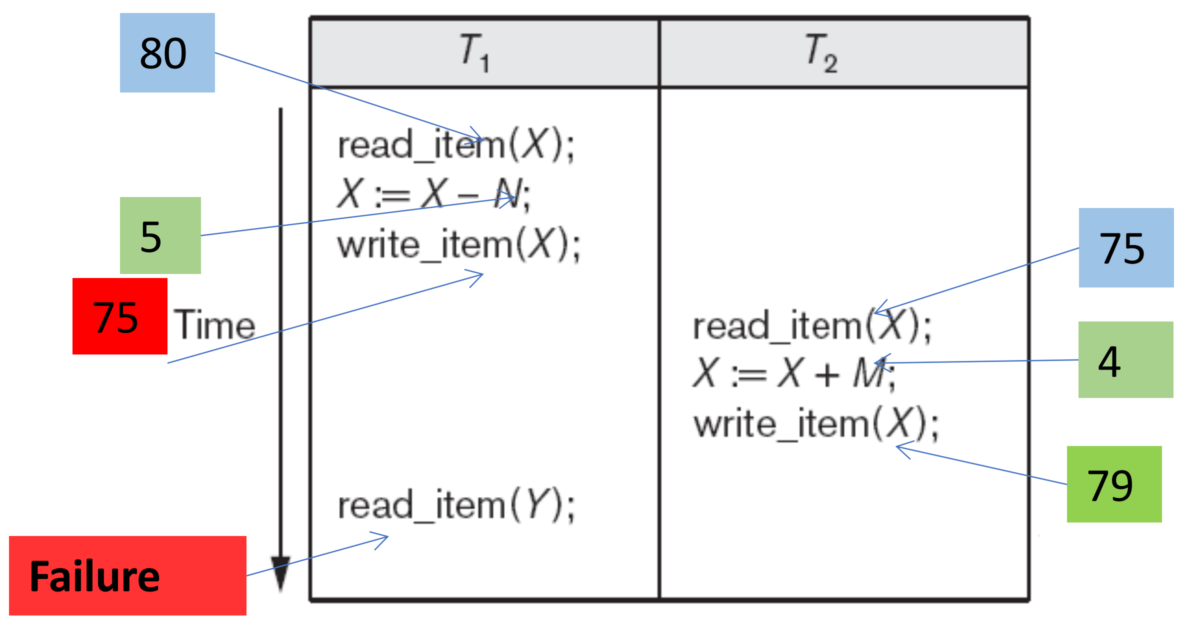
Why Concurrency Control Is Needed

- **The Lost Update Problem.** This problem occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database items incorrect.



Why Concurrency Control Is Needed… cont.

- **The Temporary Update (or Dirty Read) Problem.**
- This problem occurs when one transaction updates a database item and then the transaction fails for some reason. Meanwhile, the updated item is accessed (read) by another transaction before it is changed back to its original value.



Why Concurrency Control Is Needed… cont.

- **The Incorrect Summary Problem.** If one transaction is calculating an aggregate summary function on a number of database items while other transactions are updating some of these items, the aggregate function may calculate some values before they are updated and others after they are updated.

T_1	T_3
	$sum := 0$;
	read_item(A);
	$sum := sum + A$;
	⋮
read_item(X);	
$X := X - N$;	
write_item(X);	
	read_item(X);
	$sum := sum + X$;
	read_item(Y);
	$sum := sum + Y$;
read_item(Y);	
$Y := Y + N$;	
write_item(Y);	

Why Recovery Is Needed

- The system is responsible for making sure that
  - all the operations in the transaction are completed successfully and their effect is recorded permanently in the database, **(committed)**
  - The transaction does not have any effect on the database or any other transactions. **(aborted)**

Why Recovery Is Needed … cont.

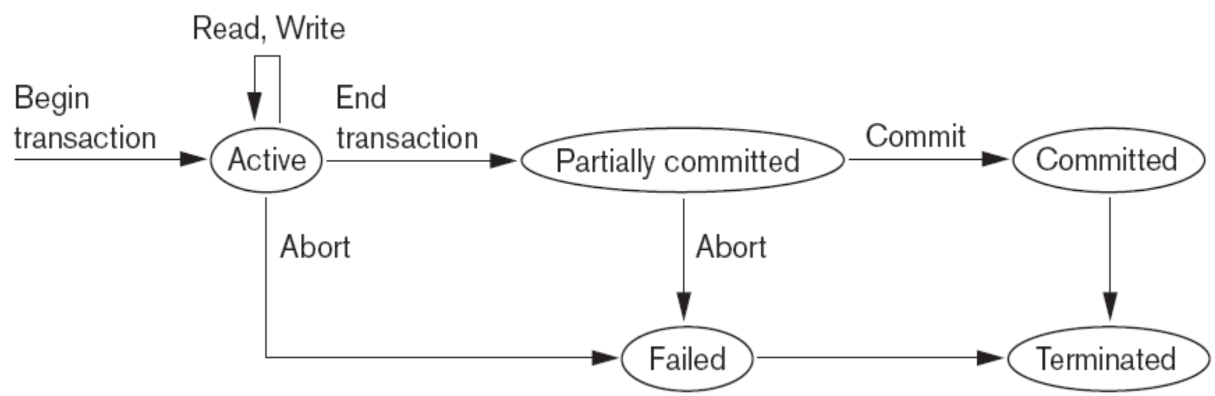
- What causes a Transaction to fail)
  - 1. A computer failure (system crash):
    - If the hardware crashes, the contents of the computer's internal memory may be lost.
  - 2. A transaction or system error:
    - Transaction operation error (integer overflow or division by zero).
    - erroneous parameter values or a logical programming error.
    - The user may interrupt the transaction during its execution.
  - 3. Local errors or exception conditions detected by the Trans. :
    - Ex. data for the transaction may not be found.
    - A programmed abort in the transaction causes it to fail.
  - 4. Concurrency control enforcement:
    - The concurrency control method may decide to abort the transaction, to be restarted later
  - 5. Disk failure:

    Ex. disk read/write head crash.
  - 6. Physical problems and catastrophes:

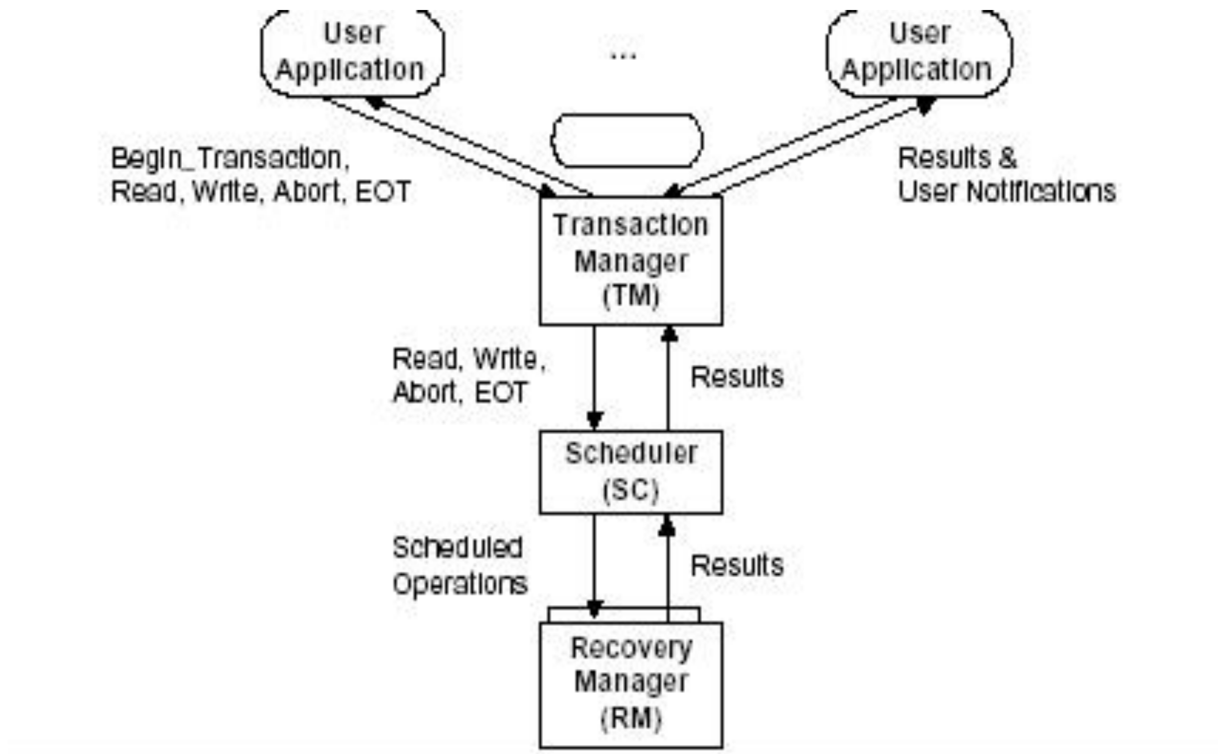
    Ex. power or air-conditioning failure, fire, theft, sabotage, overwriting disks or tapes by mistake, … etc

Transaction and System Concepts

- A **transaction** is an atomic unit of work that is either completed in its entirety or not done at all.
  - For recovery purposes, the system needs to keep track of when the transaction starts, terminates, and commits or aborts.
- **Transaction states**:
  - Active state
  - Partially committed state
  - Committed state
  - Failed state
  - Terminated State

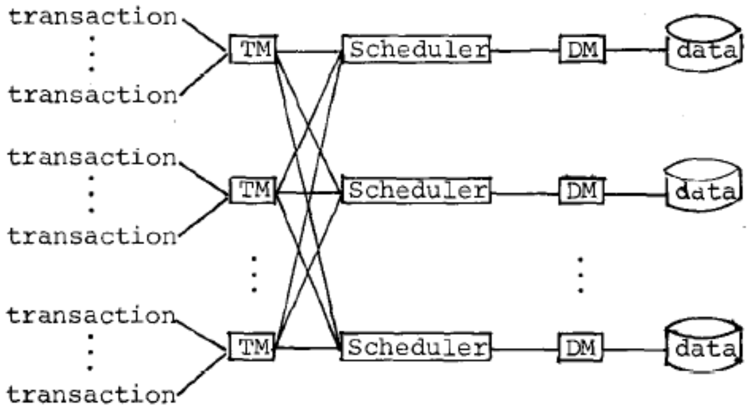


Centralized Transaction Execution

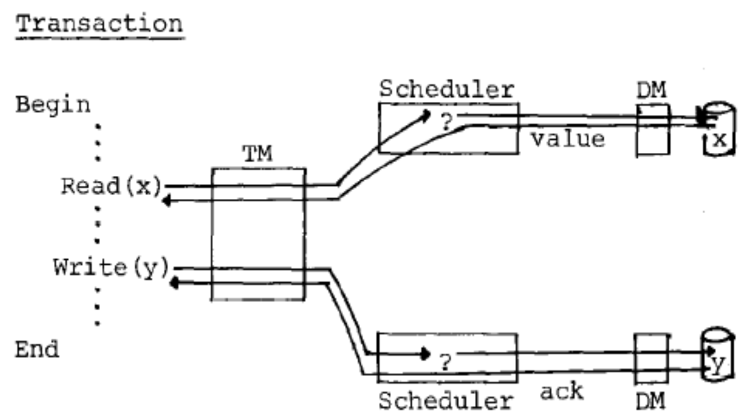


Distributed Transaction Execution

- ➢Transaction Manager
- ➢**<u>Data Manager</u>**
- ➢Scheduler

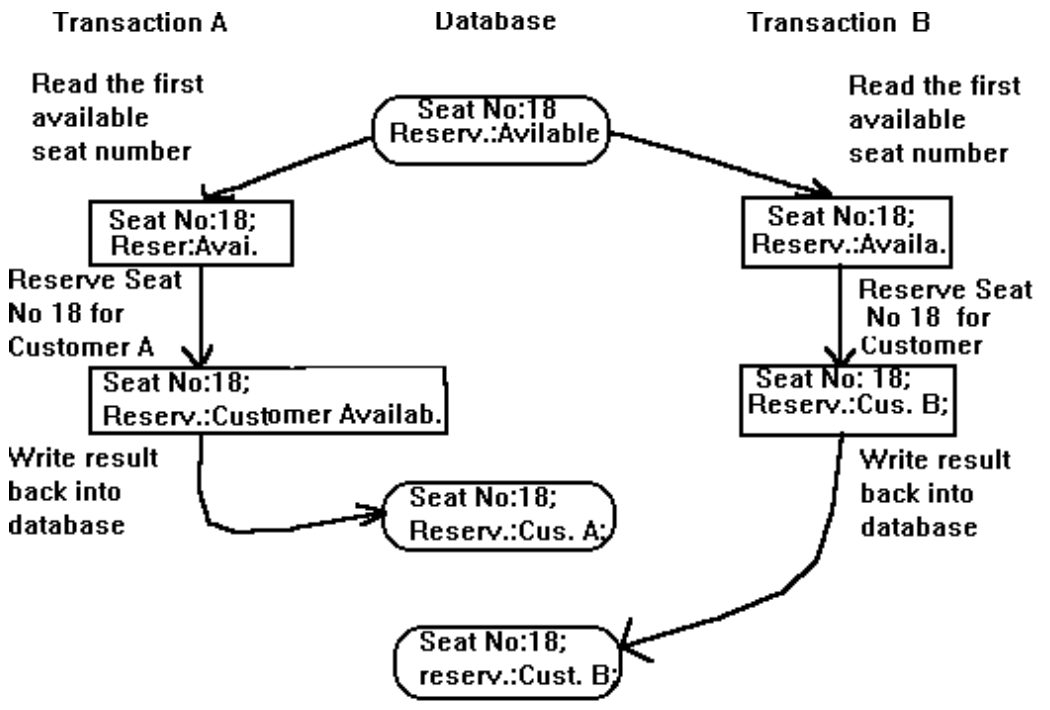


DDBS Architecture



Processing Operation

Anomaly in DB in Absence of Concurrency Control



Types of Schedules

- Serial schedule A: *T*1 followed by *T*2.
- nonserial schedule C with interleaving of operations

Time

T_1	T_2

read_item(X);
$X := X - N$;
write_item(X);
read_item(Y);
$Y := Y + N$;
write_item(Y);	

	read_item(X);
	$X := X + M$;
	write_item(X);

Schedule A

Time

T_1	T_2
read_item(X); $X := X - N$;	
	read_item(X); $X := X + M$;
write_item(X); read_item(Y);	
	write_item(X);
$Y := Y + N$; write_item(Y);	

Schedule C

Scheduling Algorithms

- Modify concurrency control schemes for use in distributed environment. There are 3 basic methods for transaction concurrency control.

➢Locking (Majority, Biased, two phase locking - 2PL).

➢Timestamp ordering

➢Optimistic

➢Hybrid

Locking Protocols

- Majority Protocol

- Local lock manager at each site administers lock and unlock requests for data items stored at that site.
- <u>Un-replicated Data:</u>
  - When a transaction wishes to lock an un replicated data item *Q* residing at site S_i, a message is sent to S_i ‘s lock manager.
  - If *Q* is locked in an incompatible mode, then the request is delayed until it can be granted.
  - When the lock request can be granted, the lock manager sends a message back to the initiator indicating that the lock request has been granted.

Majority Protocol (Cont.)

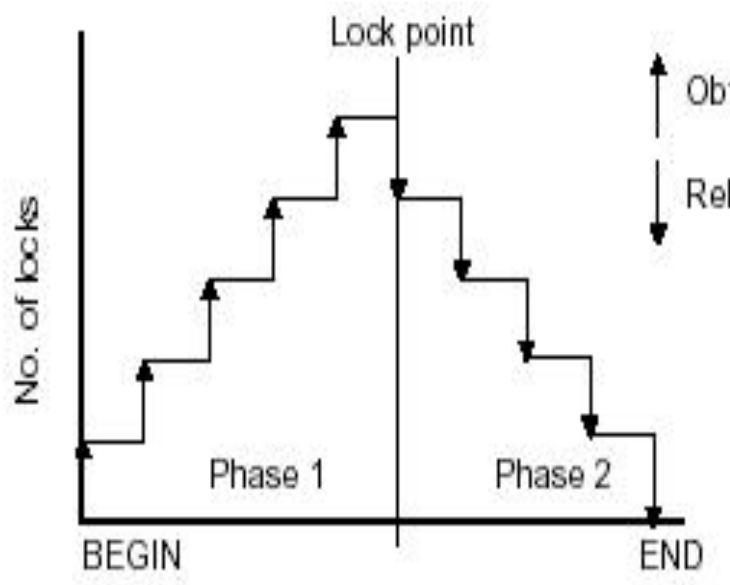
- In case of replicated data
  - If *Q* is replicated at n sites, then a lock request message must be sent to more than half of the n sites in which *Q* is stored.
  - The transaction does not operate on *Q* until it has obtained a lock on a majority of the replicas of *Q*.
  - When writing the data item, transaction performs writes on *all* replicas.
- Benefit
  - Can be used even when some sites are unavailable
- Drawback
  - Requires **2(*n*/2 + 1**) messages for handling lock requests, and **(*n*/2 + 1)** messages for handling unlock requests.
  - Potential for deadlock even with single item - e.g., each of 3 transactions may have locks on 1/3rd of the replicas of a data.

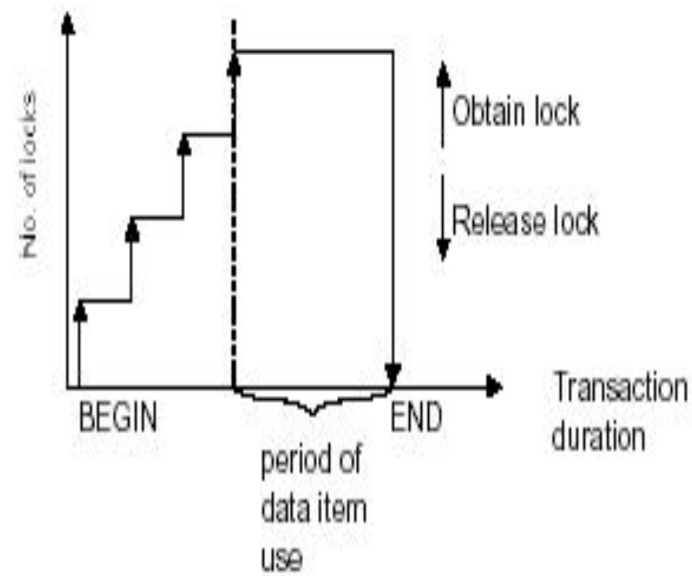
Biased Protocol

- Local lock manager at each site as in majority protocol, however, requests for shared locks are handled differently than requests for exclusive locks.
- **Shared locks.** When a transaction needs to lock data item *Q*, it simply requests a lock on *Q* from the lock manager at one site containing a replica of *Q*.
- **Exclusive locks.** When transaction needs to lock data item *Q*, it requests a lock on *Q* from the lock manager at all sites containing a replica of *Q*.
- Advantage - imposes less overhead on **read** operations.
- Disadvantage - additional overhead on writes

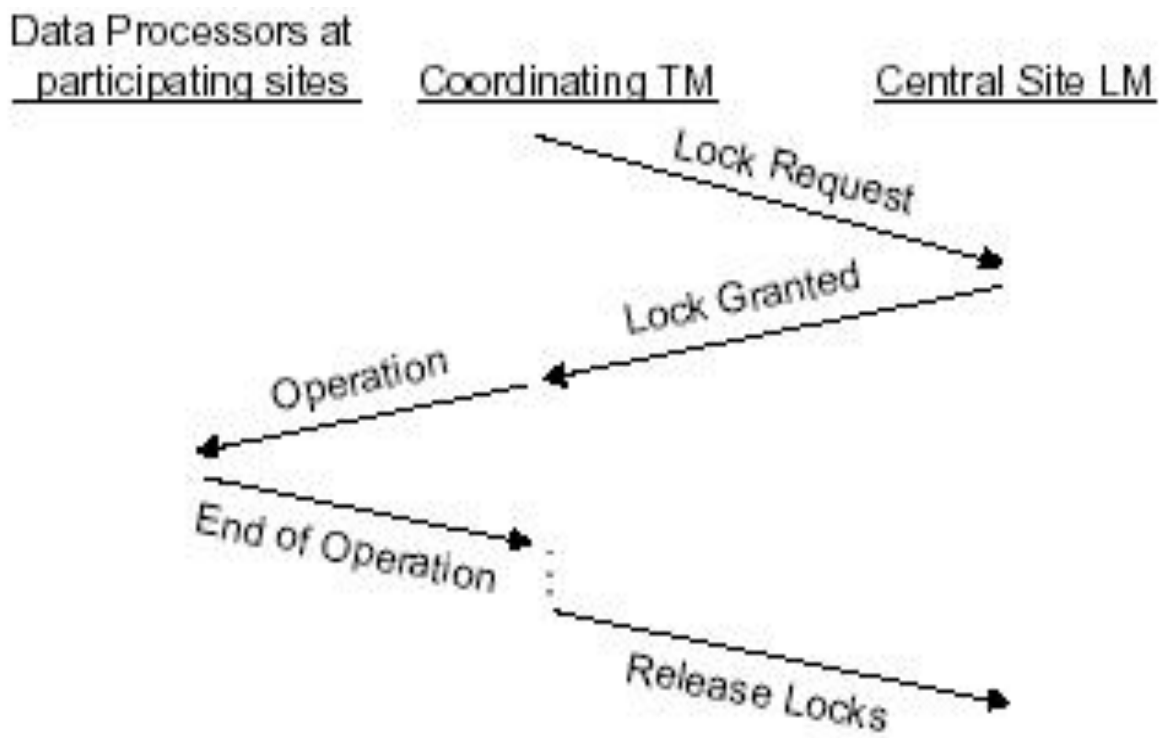
2 Phase Locking (2PL)

- *Centralized 2PL.*
- *Primary copy 2PL.*
- *Distributed 2PL.*
- *Voting 2PL.*

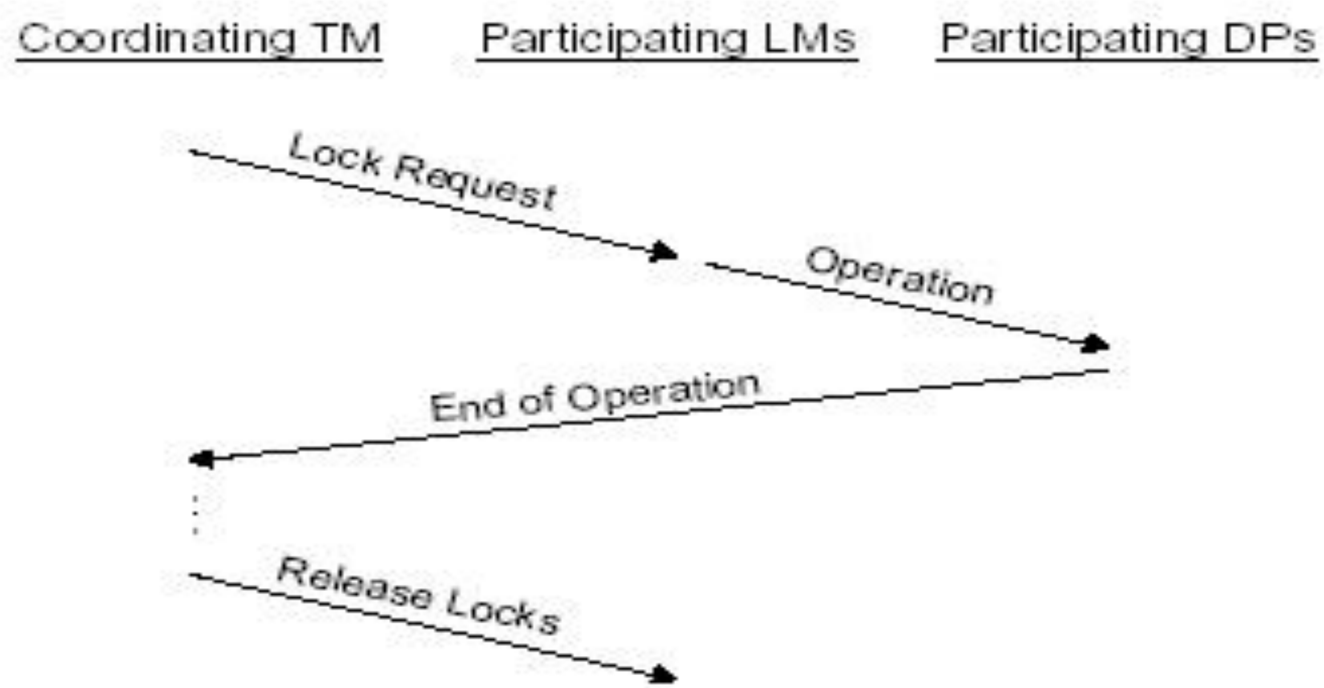




Centralized 2PL



Distributed 2PL



Timestamp Ordering

- Timestamp (TS): a number associated with each transaction
  - Not necessarily real time
    - Can be assigned by a logical counter
  - Unique for each transaction
  - Should be assigned in an increasing order for each new transaction

Timestamp Ordering

- Timestamps associated with each database item
  - Read timestamp (RTS) : the largest timestamp of the transactions that read the item so far
  - Write timestamp (WTS) : the largest timestamp of the transactions that write the item so far
- After each successful read/write of object *O* by transaction T the timestamp is updated
  - RTS(O) = max(RTS(O), TS(T))
  - WTS(O) = max(WTS(O), TS(T))

Timestamp Ordering

- Given a transaction T
- If T wants to read(X)
  - If TS(T) < WTS(X) then read is rejected, T has to abort
  - Else, read is accepted and RTS(X) updated.
- For a write-read conflict, which direction does this protocol allow?

Timestamp Ordering

- If T wants to write(X)
  - If TS(T) < RTS(X) then write is rejected, T has to abort
  - If TS(T) < WTS(X) then write is rejected, T has to abort
  - Else, allow the write, and update WTS(X) accordingly

New Approaches to Concurrency Control

- Total Ordering
- Total ordering in networking terms describes the property of a network guaranteeing that all messages are delivered in the same order across all destinations.
- In combination with the concept of transactions, one can make use of this property to ensure that transactions are received in the same order at all sites — called the ORDER CC technique.
- Algorithm

➢ Each transaction is initiated by sending its reads and write predeclares to the corresponding schedulers as a single atomic action in totally ordered fashion.

➢ Each scheduler stores the received operation requests in a FIFO-type queue.

➢ If read is at the head of the queue, it is immediately executed.

➢ transaction can now issue the write requests in accordance with the previously given predeclares.

➢ Upon commit, the committed values are send in non-ordered fashion to the schedulers, which re-place the corresponding predeclare statements in the queue with the received committed writes.

Timestamp Ordering Revisited

- Whenever a network layout provides predictability regarding the time at which a message will arrive at its destination, such as interconnection networks, this property can be exploited for concurrency control .
- Algorithm

➢ The transaction manager initiates a transaction by sending its reads and write predeclares to the corresponding schedulers as a single atomic action.

➢ This atomic action is assigned a timestamp t, denoting the time by which all operations will have arrived at their respective schedulers.

➢ When a scheduler receives an operation o, it can either wait until time t has arrived .

➢ The alternative option is to process o ahead of time t, and causing conflicting operations that arrive afterwards, but with a lower timestamp, to abort.

Thank You…

Any Questions…???