University of Windsor

11 Transaction Management-DDBMS

Kalyani Selvarajah
School of Computer Science
University of Windsor

Advanced Database Topics
COMP 8157 01 /02/ 03
Fall 2023

Today's Agenda

Transactions in Distributed System

Distributed Concurrency Control

Locking Protocols

Distributed Deadlock Management

Distributed Database Recovery

https://domains.upperlink.ng/elementor-947/

Introductory Questions

How data distribution affects the transaction management protocols?

How centralized concurrency control techniques can be extended to handle data distribution.

How to detect deadlock when multiple sites are involved.

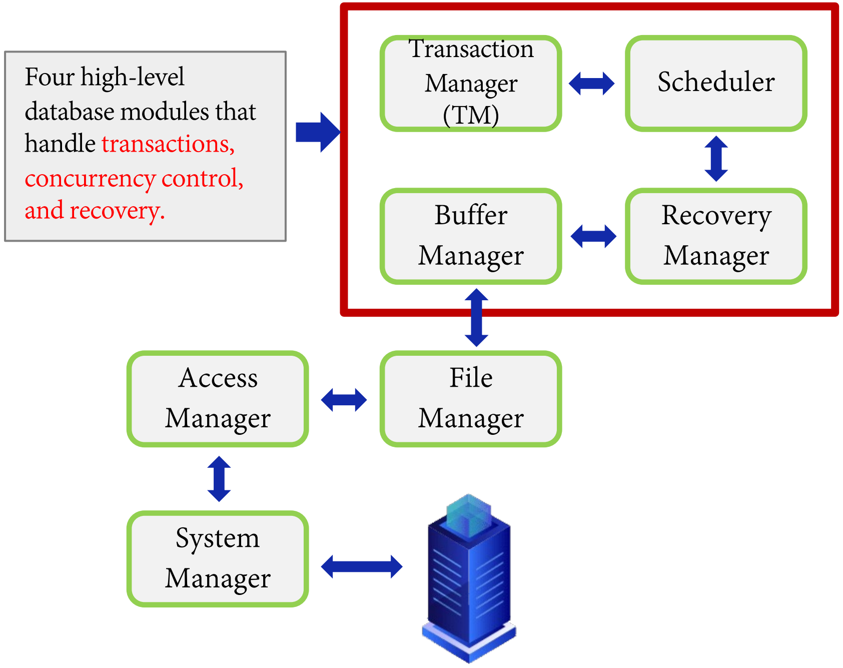
Transactions in Distributed System

The objectives of distributed transaction processing are the same as those of centralized systems, although more complex, because the DDBMS must also ensure the atomicity of the global transaction and each component sub-transaction.

- ✓ In a distributed system, a transaction can either be a local transaction or a global transaction.
- ✓ A local transaction is a transaction that performs all its work at the site where it originates.
- ✓ A global transaction is a transaction that has to perform work at one or more sites different from its originating site.

```
Begin_Tran $T_1$
    Begin_Tran $T_1$LA
        R(X)
        Calculate $X = X + 200$
        W(X)
        Commit T1LA
    End_Tran $T_1$LA
    Begin_Tran $T_1$NY
        R(X)
        Calculate $X = X * 1.10$
        W(X)
        Commit $T_1$NY
    End_Tran $T_1$NY
    Commit $T_1$
End_Tran $T_1$;
```

Coordination of Distributed Transaction



Four high-level database modules that handle transactions, concurrency control, and recovery.

Transaction Manager (TM)

Scheduler

Buffer Manager

Recovery Manager

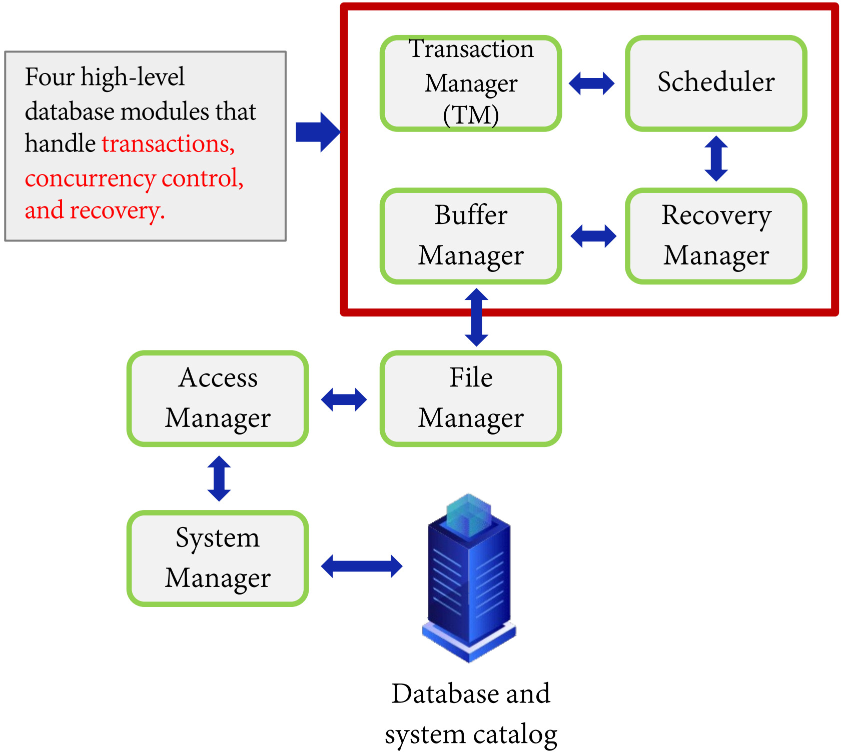
Access Manager

File Manager

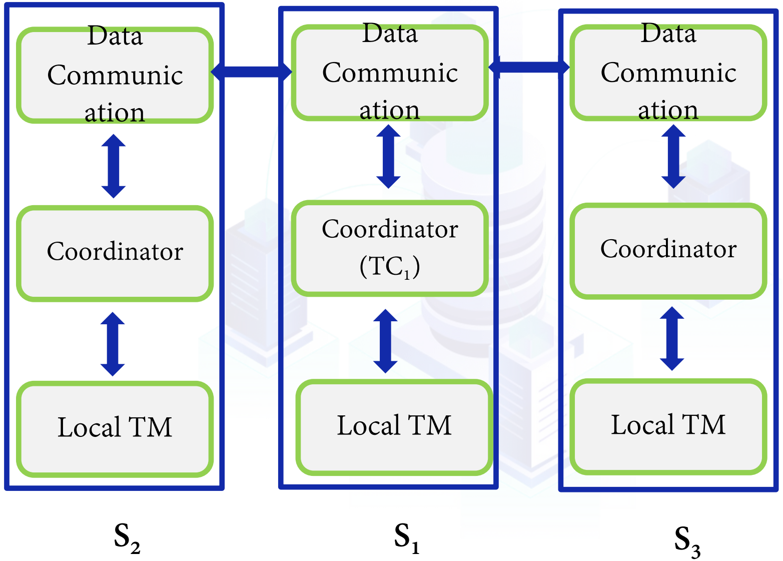
System Manager

Database and system catalog

Coordination of Distributed Transaction



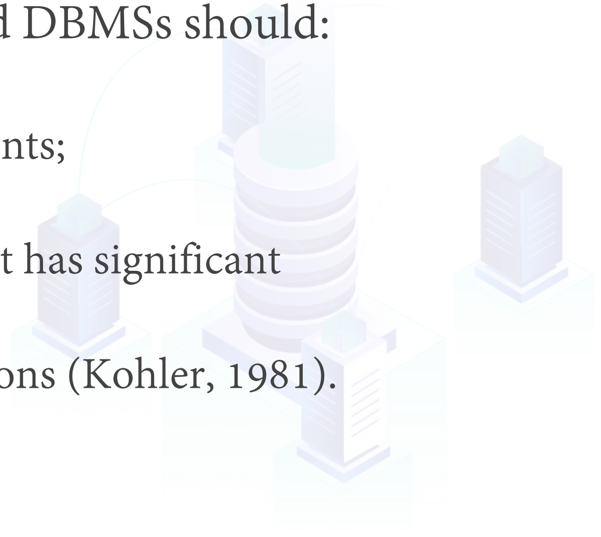
Four high-level database modules that handle transactions, concurrency control, and recovery.



Distributed Concurrency Control

A good concurrency control mechanism for distributed DBMSs should:

- ✓ be resilient to **site and communication** failure;
- ✓ permit parallelism to satisfy performance requirements;
- ✓ incur modest computational and storage overhead;
- ✓ perform satisfactorily in a network environment that has significant communication delay;
- ✓ place few constraints on the structure of atomic actions (Kohler, 1981).



Problems with Distributed Concurrency Control

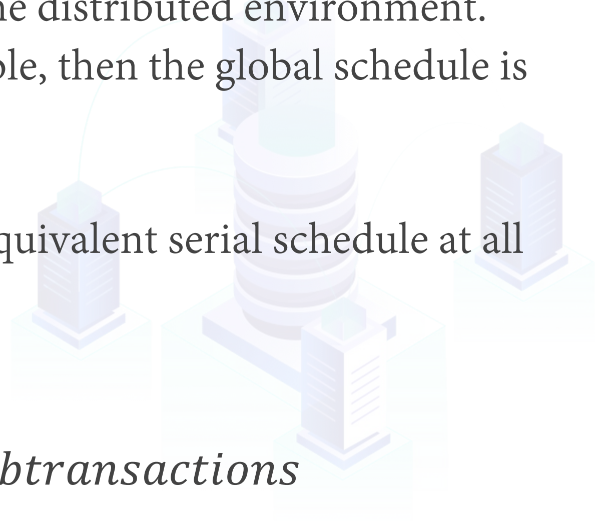
- ✓ Lost update,
- ✓ Uncommitted dependency,
- ✓ Inconsistent analysis
- ✓ **Multiple-copy consistency problem**

We assume in this section that updates to replicated items are carried out **synchronously,** as part of the enclosing transaction.

Distributed Serializability

The concept of serializability in CDBMS can be extended for the distributed environment. If the schedule of transaction execution at each site is serializable, then the global schedule is also serializable.

- ✓ local serialization orders are **identical**
- ✓ all sub-transactions appear in the same order in the equivalent serial schedule at all sites



$$T_i^x < T_j^x \ \mathit{for\ all}\ S_x\ \mathit{at\ which}\ T_i\ \mathit{and}\ T_j\ \mathit{have\ subtransactions}$$

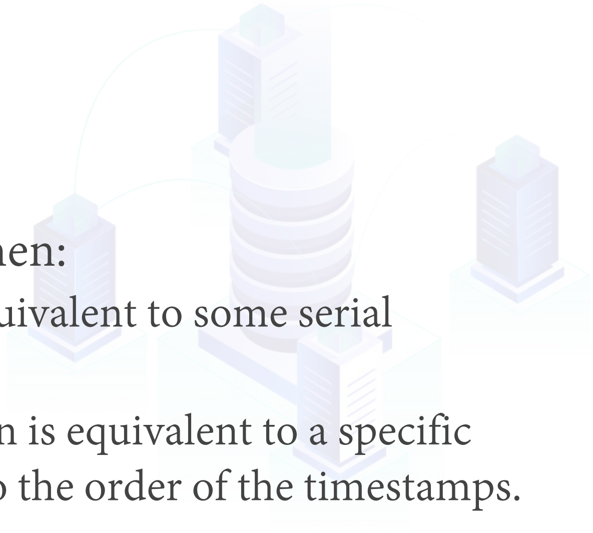
Solutions to Concurrency Control in DDBMS

Solution:

Locking
Timestamping

Given a set of transactions to be executed concurrently, then:

- ✓ **locking guarantees** that the concurrent execution is equivalent to some serial execution of those transactions;
- ✓ **timestamping** guarantees that the concurrent execution is equivalent to a specific serial execution of those transactions, corresponding to the order of the timestamps.



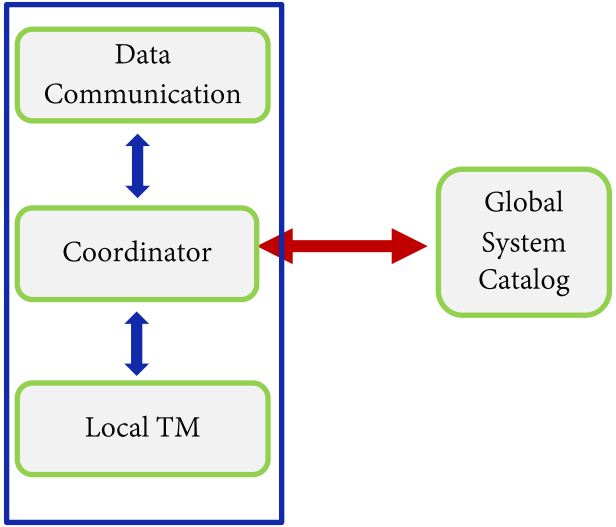
Locking Protocols

Protocols based on two-phase locking (2PL)

- ✓ Centralized 2PL.
- ✓ Primary Copy 2PL.
- ✓ Distributed 2PL.
- ✓ Majority Locking.

Centralized 2PL

- ✓ Single site that maintains all locking information.
- ✓ One lock manager for whole of DDBMS.



- Divides the transaction into a number of sub-transactions.
- Coordinator endure the consistency:
  - If T involves an update of data that is replicated, all copies of the data need to be updated
  - Request X-Lock on all copies of data.
  - Elect one copy to read final update.
- Local TM involved in the GT request and release lock from centralized lock manager.
- The centralized lock manager check for availability, it send an ACK. O/W put the request in the queue

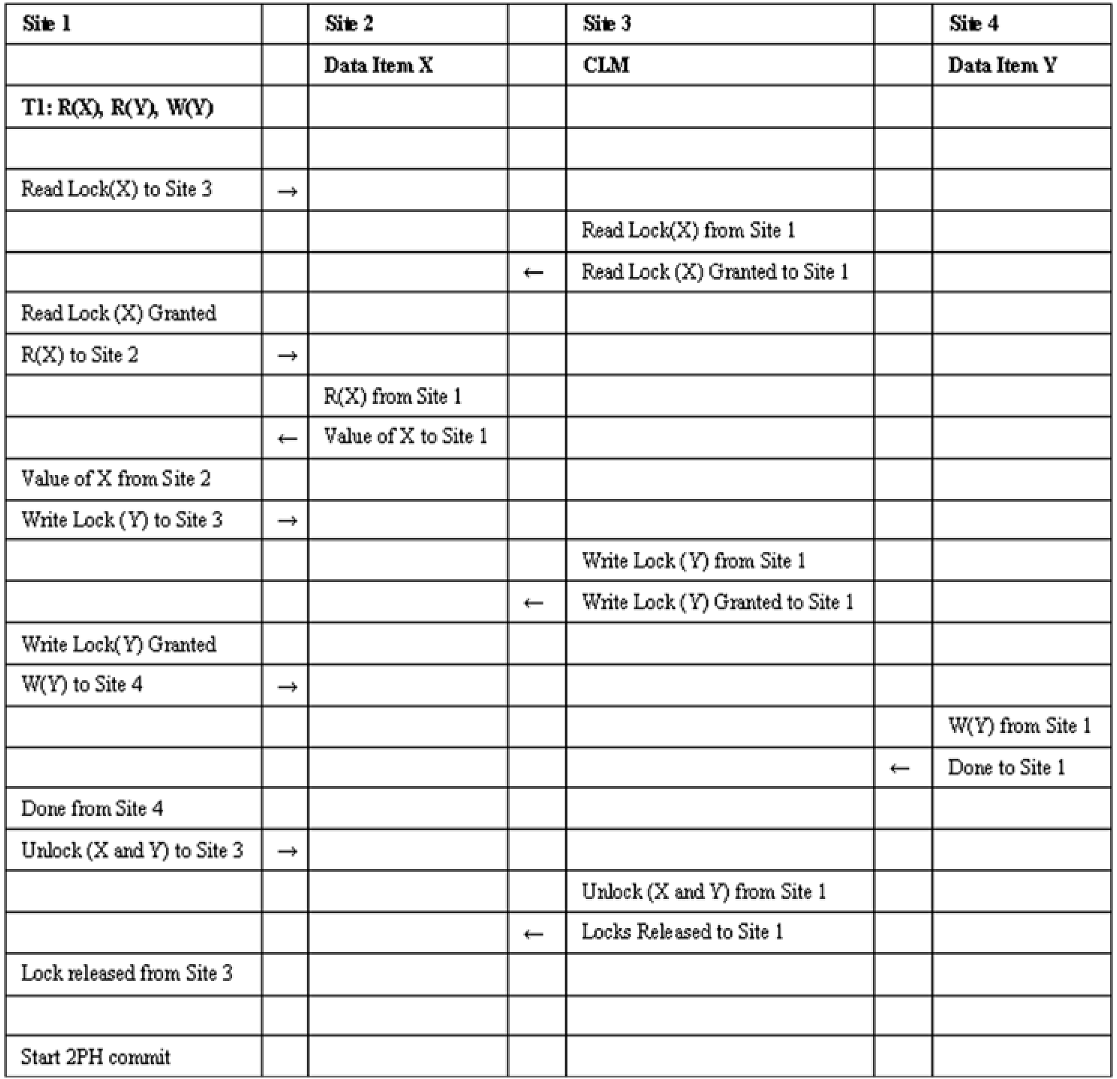
Centralized 2PL

✓ Advantage - easy to implement.

✓ Disadvantages - bottlenecks and lower reliability.

Centralized 2PL

Example: transaction “T1 = R(X), R(Y), W(Y)” enters
the system at Site 1. Data item X is stored at Site 2 and data item Y is stored at Site 4. The central lock manager (CLM) runs at Site 3.



Site 1		Site 2		Site 3		Site 4
		Data Item X		CLM		Data Item Y
T1: R(X), R(Y), W(Y)						
Read Lock(X) to Site 3	→					
				Read Lock(X) from Site 1		
			←	Read Lock (X) Granted to Site 1		
Read Lock (X) Granted						
R(X) to Site 2	→					
		R(X) from Site 1				
	←	Value of X to Site 1				
Value of X from Site 2						
Write Lock (Y) to Site 3	→					
				Write Lock (Y) from Site 1		
			←	Write Lock (Y) Granted to Site 1		
Write Lock(Y) Granted						
W(Y) to Site 4	→					
						W(Y) from Site 1
					←	Done to Site 1
Done from Site 4						
Unlock (X and Y) to Site 3	→					
				Unlock (X and Y) from Site 1		
			←	Locks Released to Site 1		
Lock released from Site 3						
Start 2PH commit						

Primary Copy 2PL

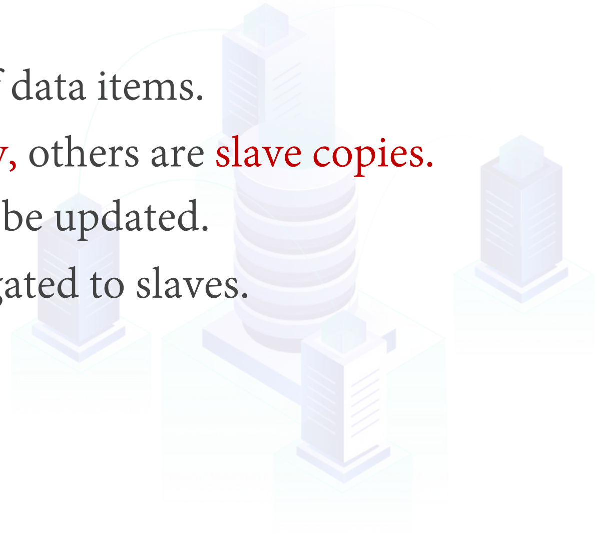
✓ Lock managers distributed to a number of sites.

✓ Each lock manager responsible for managing locks for set of data items.

✓ For replicated data item, one copy is chosen as primary copy, others are slave copies.

✓ Only need to write-lock primary copy of data item that is to be updated.

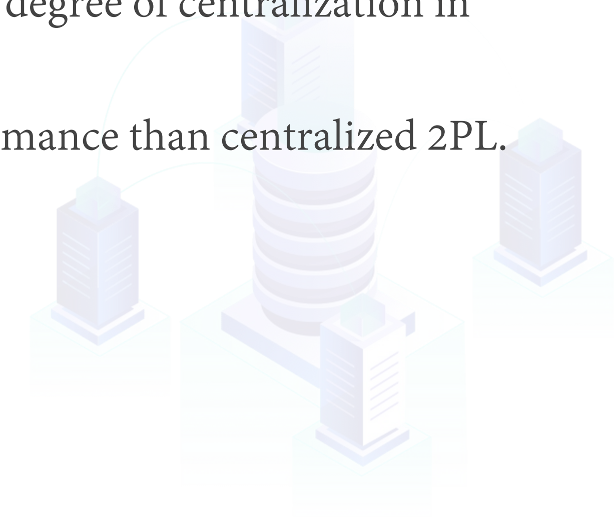
✓ Once primary copy has been updated, change can be propagated to slaves.



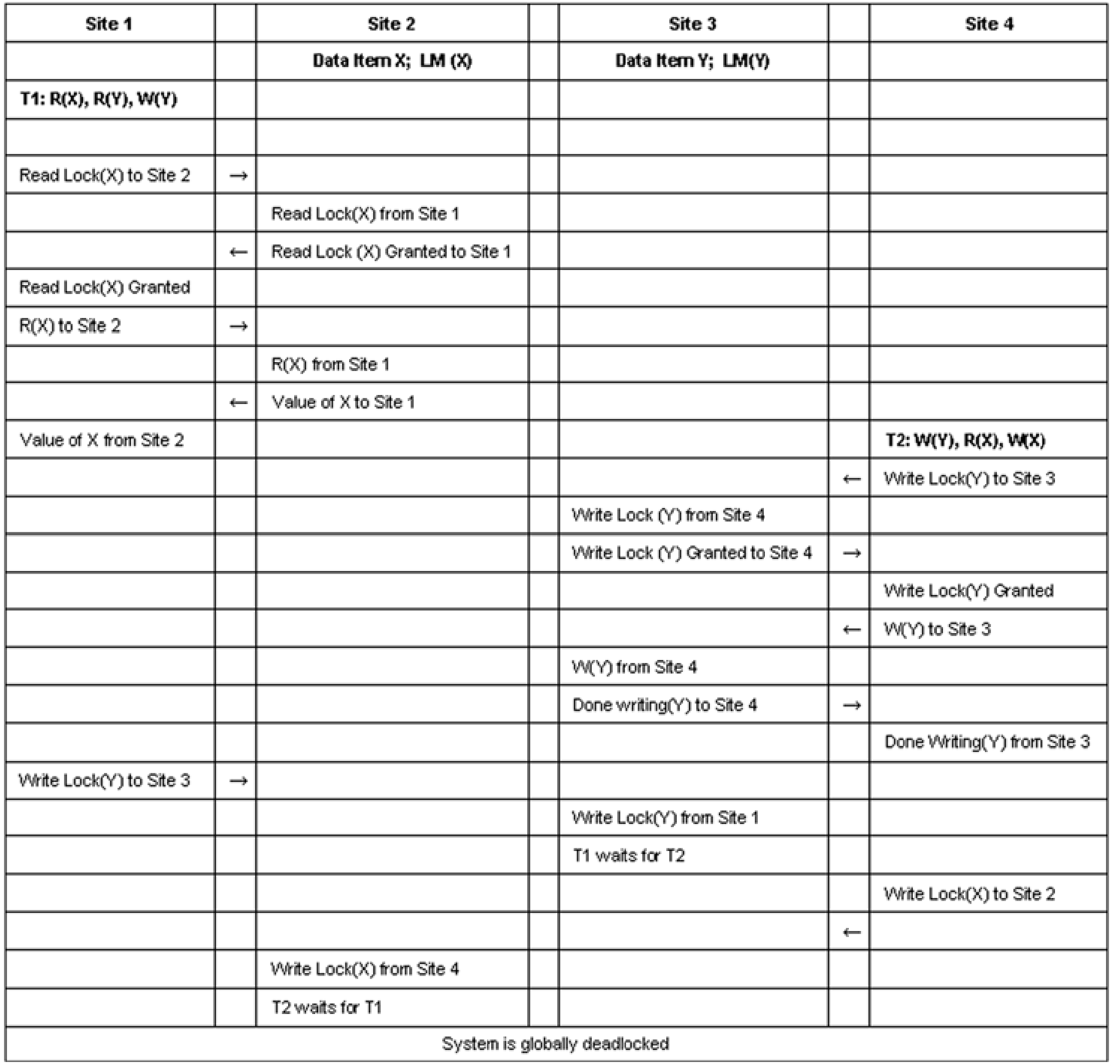
Primary Copy 2PL

✓ Disadvantages - deadlock handling is more complex; still a degree of centralization in system.

✓ Advantages - lower communication costs and better performance than centralized 2PL.



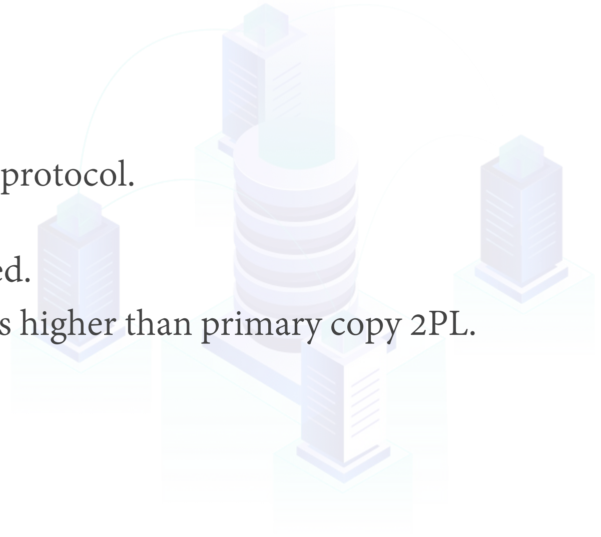
Primary Copy 2PL



Site 1		Site 2		Site 3		Site 4
		Data Item X; LM (X)		**Data Item Y; LM(Y)**		
T1: R(X), R(Y), W(Y)						
Read Lock(X) to Site 2	→					
		Read Lock(X) from Site 1				
	←	Read Lock (X) Granted to Site 1				
Read Lock(X) Granted						
R(X) to Site 2	→					
		R(X) from Site 1				
	←	Value of X to Site 1				
Value of X from Site 2						**T2: W(Y), R(X), W(X)**
					←	Write Lock(Y) to Site 3
				Write Lock (Y) from Site 4		
				Write Lock (Y) Granted to Site 4	→	
						Write Lock(Y) Granted
					←	W(Y) to Site 3
				W(Y) from Site 4		
				Done writing(Y) to Site 4	→	
						Done Writing(Y) from Site 3
Write Lock(Y) to Site 3	→					
				Write Lock(Y) from Site 1		
				T1 waits for T2		
						Write Lock(X) to Site 2
					←	
		Write Lock(X) from Site 4				
		T2 waits for T1				
System is globally deadlocked						

Distributed 2PL

- ✓ Lock managers distributed to every site.
- ✓ Each lock manager responsible for locks for data at that site.
- ✓ If data not replicated, equivalent to primary copy 2PL.
- ✓ Otherwise, implements a Read-One-Write-All (ROWA) replica control protocol.
    - o Any copy of a replicated item can be used for a read operation.
    - o all copies must be exclusively locked before an item can be updated.
- ✓ Disadvantages - deadlock handling more complex; communication costs higher than primary copy 2PL.



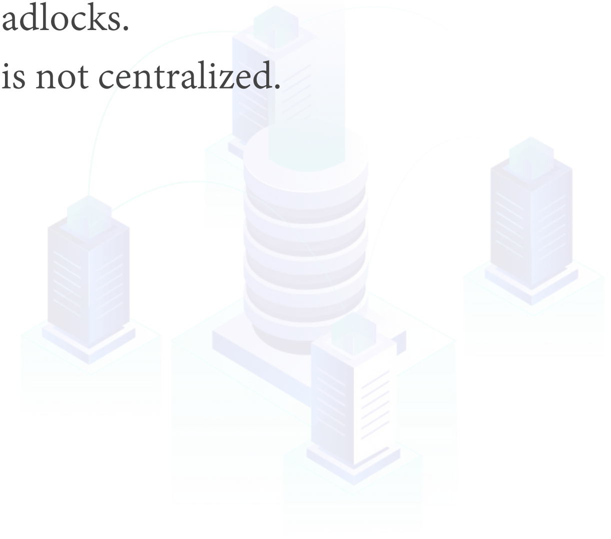
Majority Locking

- ✓ Extension of distributed 2PL.
- ✓ To read or write data item replicated at n sites, sends a lock request to more than half the n sites where item is stored.
- ✓ Transaction cannot proceed until majority of locks obtained.
- ✓ Overly strong in case of read locks.

Distributed Deadlock Management

✓ Any locking-based concurrency control algorithm may result in deadlocks.

✓ Deadlock detection may be more complicated if lock management is not centralized.



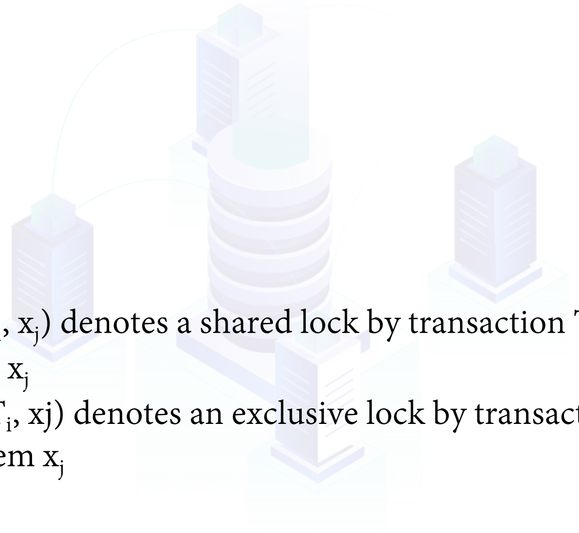
Distributed Deadlock Management

T_1 initiated at site S_1 and creating agent at S_2,

T_2 initiated at site S_2 and creating agent at S_3,

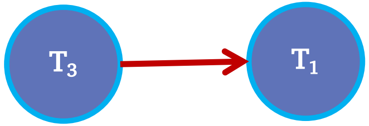
T_3 initiated at site S_3 and creating agent at S_1.

Time	S_1	S_2	S_3
t_1	read_lock(T_1, x_1)	write_lock(T_2, y_2)	read_lock(T_3, z_3)
t_2	write_lock(T_1, y_1)	write_lock(T_2, z_2)	
t_3	write_lock(T_3, x_1)	write_lock(T_1, y_2)	write_lock(T_2, z_3)

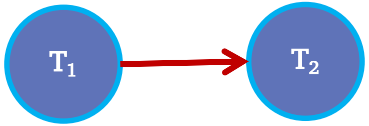


read_lock(T_i, x_j) denotes a shared lock by transaction T_i on data item x_j

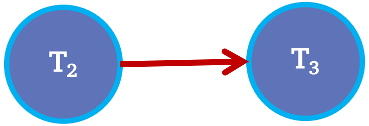
write_lock(T_i, xj) denotes an exclusive lock by transaction T_i on data item x_j



S_1



S_2



S_3

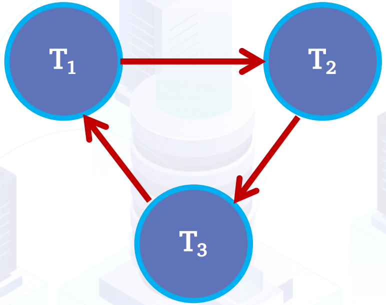
Distributed Deadlock Management

T_1 initiated at site S_1 and creating agent at S_2,

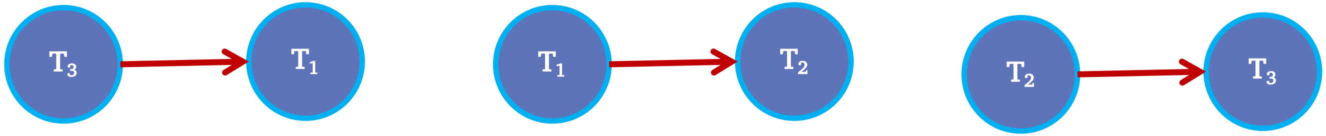
T_2 initiated at site S_2 and creating agent at S_3,

T_3 initiated at site S_3 and creating agent at S_1.

Time	S_1	S_2	S_3
t_1	read_lock(T_1, x_1)	write_lock(T_2, y_2)	read_lock(T_3, z_3)
t_2	write_lock(T_1, y_1)	write_lock(T_2, z_2)	
t_3	write_lock(T_3, x_1)	write_lock(T_1, y_2)	write_lock(T_2, z_3)



Combined wait-for graphs for sites S_1, S_2, and S_3.



S_1

S_2

S_3

Deadlock Detection in DDBMSs

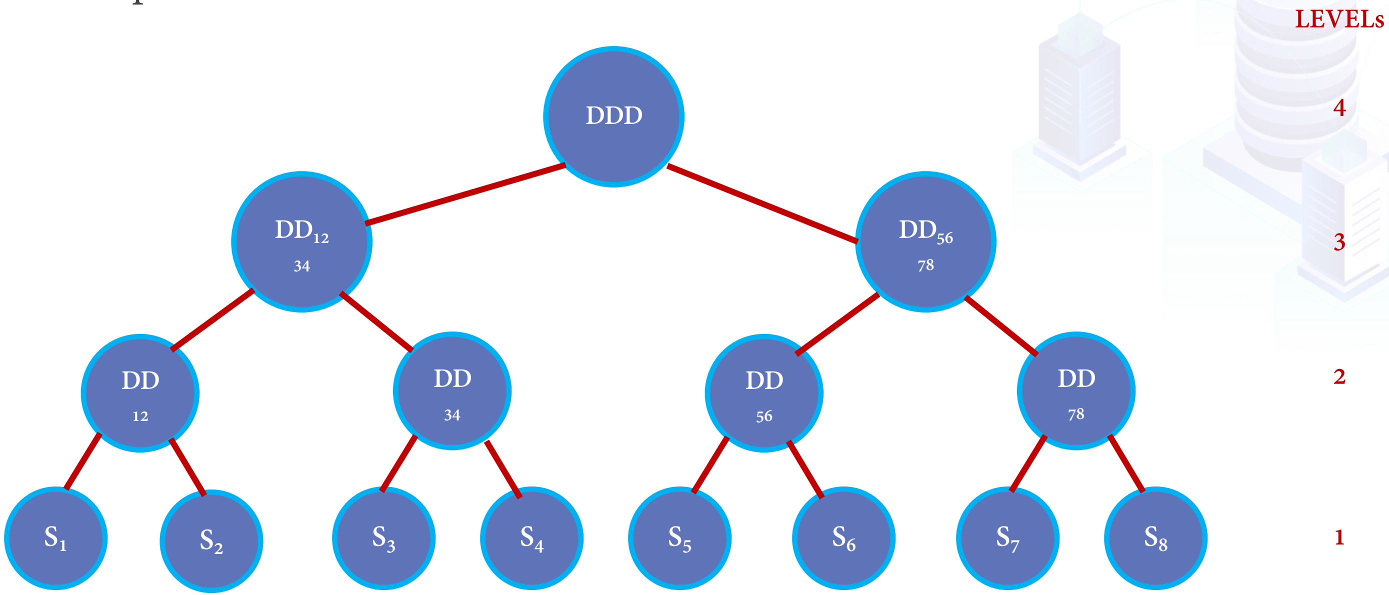
- ✓ Centralized Deadlock Detection.
- ✓ Hierarchical Deadlock Detection.
- ✓ Distributed Deadlock Detection.

Centralized Deadlock Detection.

- ✓ Single site appointed deadlock detection coordinator (DDC).
- ✓ DDC has responsibility for constructing and maintaining global WFG.
- ✓ If one or more cycles exist, DDC must break each cycle by selecting transactions to be rolled back and restarted.

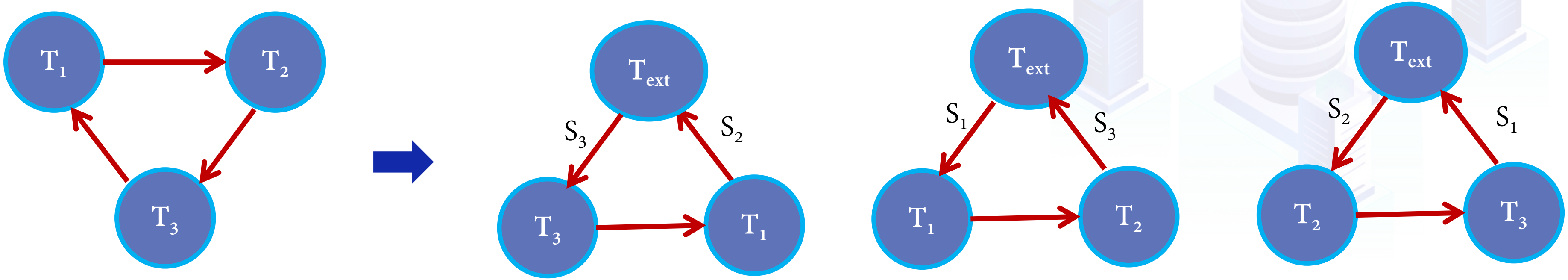
Hierarchical Deadlock Detection

- ✓ Sites are organized into a hierarchy.
- ✓ Each site sends its local WFG to detection site above it in hierarchy.
- ✓ Reduces dependence on centralized detection site.



Distributed Deadlock Detection

- ✓ Most well-known method developed by Obermarck (1982).
- ✓ An external node, Text, is added to local WFG to indicate remote agent.
- ✓ If a local WFG contains a cycle that does not involve Text, then site and DDBMS are in deadlock.



Global WFG

S_1

S_2

S_3

local WFGs at sites S_1, S_2, and S_3

Distributed Deadlock Detection.

S_1: Text → T_3 → T_1 → Text

S_2: Text → T_1 → T_2 → Text

S_3: Text → T_2 → T_3 → Text

Transmit local WFG for S_1 to the site for which transaction T_1 is waiting, site S_2.

Local WFG at S_2 is extended and becomes:

- S_2: Text → T_3 → T_1 → T_2 → Text

Still contains potential deadlock, so transmit this WFG to S3:

- S_3: Text → T_3 → T_1 → T_2 → T_3 → Text

Global WFG contains cycle not involving Text, so deadlock exists.

- $T_3 \rightarrow T_1 \rightarrow T_2 \rightarrow T_3$

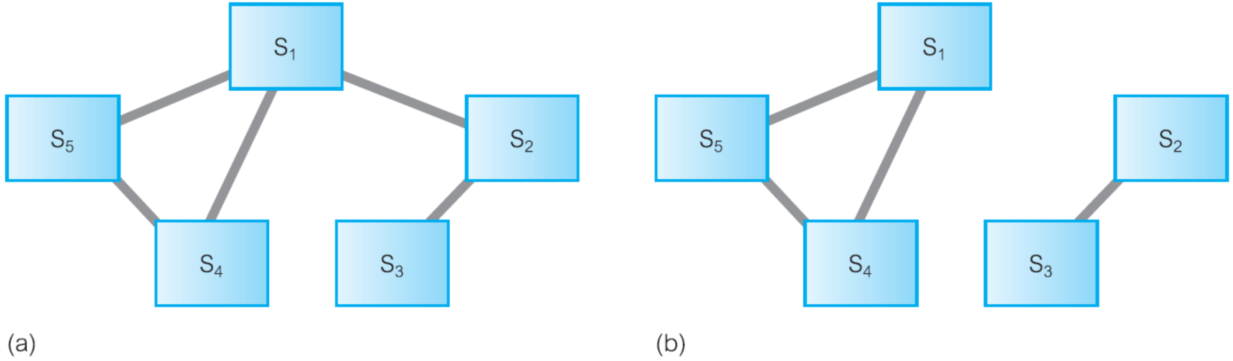
Distributed Database Recovery

Failures in a Distributed Environment:

- ✓ Loss of a message.
- ✓ Failure of a communication link.
- ✓ Failure of a site.
- ✓ Network partitioning.

Distributed Database Recovery

- ✓ DDBMS is highly dependent on ability of all sites to be able to communicate reliably with one another.
- ✓ Communication failures can result in network becoming split into two or more partitions.
- ✓ May be difficult to distinguish whether communication link or site has failed.



Partitioning a network: (a) before failure; (b) after failure.

Quiz!

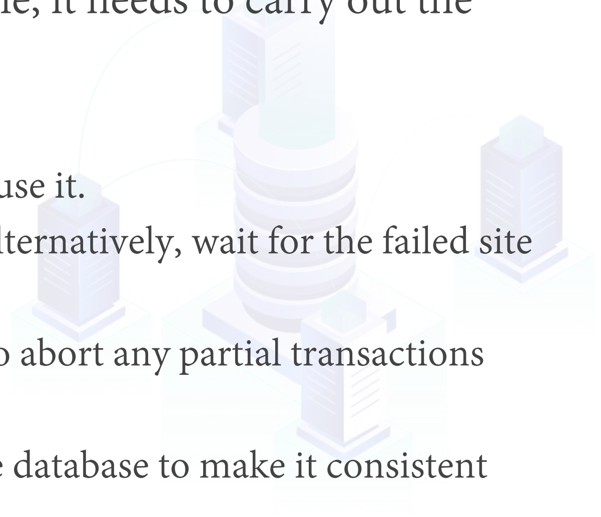
Suppose that site S1 cannot communicate with site S2 within a fixed (timeout) period. What could be the reason ?

1. site S2 has crashed, or the network has gone down;
2. the communication link has failed;
3. the network is partitioned;
4. site S2 is currently very busy and has not had time to respond to the message.
5. All of the above

Distributed Database Recovery

If the DDBMS detects that a site has failed or become inaccessible, it needs to carry out the following steps:

- ✓ **Abort any transactions** that are affected by the failure.
- ✓ **Flag the site** as failed to prevent any other site from trying to use it.
- ✓ **Check periodically** to see whether the site has recovered or, alternatively, wait for the failed site to broadcast that it has recovered.
- ✓ On restart, the failed site must initiate a recovery procedure to abort any partial transactions that were active at the time of the failure.
- ✓ After local recovery, the failed site must update its copy of the database to make it consistent with the rest of the system.



Summary

We enhance the knowledge of transaction management in CDBMS to DDBMS.