# Concurrency Control in Centralized Databases

# Atomocity Using Lock-Based Protocols

- A lock is a mechanism to control concurrent access to a data item
- Data items can be locked in two modes :
  - *exclusive (X) mode*. Data item can be both read as well as written. X-lock is requested using **lock-X** instruction.
  - *shared (S) mode*. Data item can only be read. S-lock is requested using **lock-S** instruction.
- Lock requests are made to the concurrency-control manager by the transaction manager. Transaction can proceed only after request is granted.

**Lock-compatibility matrix**

| | S | X |
|---|---|---|
| S | true | false |
| X | false | false |

# Atomocity Using Lock-Based Protocols

| T1 | T2 | T3 | Lock status |
|---|---|---|---|
| LOCK-S(A) | | | GRANT(A,T1) |
| READ(A) | | | |
| | LOCK-S(A) | | GRANT(A,T2) |
| | READ(A) | | |
| | | LOCK-X(A) | WAIT(A,T3) |
| | | WRITE(A) | |
| | | | |

**Lock-compatibility matrix**

| | S | X |
|---|---|---|
| S | true | false |
| X | false | false |

# Atomocity Using Lock-Based Protocols

| T4 | T5 | T6 | Lock status |
|---|---|---|---|
| LOCK-X(A) | | | GRANT(A,T4) |
| WRITE(A) | | | |
| | LOCK-S(A) | | WAIT(A,T5) |
| | READ(A) | | |
| | | LOCK-X(A) | WAIT(A,T6) |
| | | WRITE(A) | |
| | | | |

**Lock-compatibility matrix**

| | S | X |
|---|---|---|
| S | true | false |
| X | false | false |

# Atomocity Using Lock-Based Protocols

| T7 | T8 | T9 | Lock status |
|---|---|---|---|
| LOCK-X(A) | | | GRANT(A,T7) |
| WRITE(A) | | | |
| | LOCK-X(B) | | GRANT(B,T8) |
| | WRITE(B) | | |
| | | LOCK-X(C) | GRANT(C,T9) |
| | | WRITE(C) | |
| | | | |

**Lock-compatibility matrix**

| | S | X |
|---|---|---|
| S | true | false |
| X | false | false |

# Atomocity Using Lock-Based Protocols

**Lock-compatibility matrix**

| | S | X |
|---|---|---|
| S | true | false |
| X | false | false |

**Question 24-1** Put appropriate lock and show lock status

| T1 | T2 | T3 | T4 | LOCK STATUS |
|---|---|---|---|---|
| | | | | |
| READ(P) | | | | |
| | | | | |
| | WRITE(Q) | | | |
| | | | | |
| | | READ(P) | | |
| | | | | |
| | | | WRITE(P) | |

# Lock-Based Protocols

- Example of a transaction performing locking:

  $T_2$: **lock-S**(*A*);
  **read** (*A*);
  **unlock**(*A*);
  **lock-S**(*B*);
  **read** (*B*);
  **unlock**(*B*);
  **display**(*A+B*)

- Locking as above is not sufficient to guarantee serializability — if *A* and *B* get updated in-between the read of *A* and *B*, the displayed sum would be wrong.
- A **locking protocol** is a set of rules followed by all transactions while requesting and releasing locks. Locking protocols restrict the set of possible schedules.

# Pitfalls of Lock-Based Protocols

- Consider the partial schedule

| $T_3$ | $T_4$ |
|---|---|
| lock-x ($B$) | |
| read ($B$) | |
| $B := B - 50$ | |
| write ($B$) | |
| | lock-s ($A$) |
| | read ($A$) |
| | lock-s ($B$) |
| lock-x ($A$) | |

- Neither $T_3$ nor $T_4$ can make progress — executing **lock-S**$(B)$ causes $T_4$ to wait for $T_3$ to release its lock on $B$, while executing **lock-X**$(A)$ causes $T_3$ to wait for $T_4$ to release its lock on $A$.
- Such a situation is called a **deadlock**.
  - To handle a deadlock one of $T_3$ or $T_4$ must be rolled back and its locks released.

# The Two-Phase Locking Protocol

- This is a protocol which ensures conflict-serializable schedules.
- Phase 1: Growing Phase
  - transaction may obtain locks
  - transaction may not release locks
- Phase 2: Shrinking Phase
  - transaction may release locks
  - transaction may not obtain locks
- The protocol assures serializability. It can be proved that the transactions can be serialized in the order of their **lock points** (i.e. the point where a transaction acquired its final lock).
- **Rigorous two-phase locking** is even stricter: here *all* locks are held till commit/abort. In this protocol transactions can be serialized in the order in which they commit.

# Schedules

- **Schedule** – a sequences of instructions that specify the chronological order in which instructions of concurrent transactions are executed
  - A schedule for a set of transactions must consist of all instructions of those transactions
  - Must preserve the order in which the instructions appear in each individual transaction.
- A transaction that successfully completes its execution will have a **commit** instructions as the last statement
  - By default transaction assumed to execute commit instruction as its last step
- A transaction that fails to successfully complete its execution will have an **abort** instruction as the last statement

# Serializability

- **Basic Assumption** – Each transaction preserves database consistency.
- Thus, serial execution of a set of transactions preserves database consistency. (T1, T2, T3, ….., Tn)
- A (possibly concurrent) schedule is serializable if it is equivalent to a serial schedule. Different forms of schedule equivalence give rise to the notions of:
  1. **Conflict serializability**
  2. **View serializability**

# SERIALIZABILITY

Schedule 1 (Serial Schedule)

| T1 | T2 |
|---|---|
| READ(A) | |
| READ(B) | |
| R=A+B | |
| DISPLAY(R) | |
| | READ(A) |
| | READ(C ) |
| | A = A-1000 |
| | C=C+1000 |
| | WRITE(A) |
| | WRITE(C) |

Schedule 2 (Equivalent concurrent Schedule)

| T1 | T2 |
|---|---|
| READ(A) | |
| | READ(A) |
| | READ(C ) |
| | A = A-1000 |
| | C=C+1000 |
| READ(B) | |
| R=A+B | |
| DISPLAY(R) | |
| | WRITE(A) |
| | WRITE(C) |

# Schedule 1

- Let $T_1$ transfer \$50 from $A$ to $B$, and $T_2$ transfer 10% of the balance from $A$ to $B$.
- An example of a **serial** schedule in which $T_1$ is followed by $T_2$ :

| $T_1$ | $T_2$ |
|---|---|
| read ($A$) | |
| $A := A - 50$ | |
| write ($A$) | |
| read ($B$) | |
| $B := B + 50$ | |
| write ($B$) | |
| commit | |
| | read ($A$) |
| | $temp := A * 0.1$ |
| | $A := A - temp$ |
| | write ($A$) |
| | read ($B$) |
| | $B := B + temp$ |
| | write ($B$) |
| | commit |

A =100, B=100, .In schedules 1, the sum "A + B" is preserved.

# Schedule 2

- Let $T_1$ and $T_2$ be the transactions defined previously. The following schedule is not a serial schedule, but it is **equivalent** to Schedule 1.

| $T_1$ | $T_2$ |
|---|---|
| read ($A$) | |
| $A := A - 50$ | |
| write ($A$) | |
| | read ($A$) |
| | $temp := A * 0.1$ |
| | $A := A - temp$ |
| | write ($A$) |
| read ($B$) | |
| $B := B + 50$ | |
| write ($B$) | |
| commit | |
| | read ($B$) |
| | $B := B + temp$ |
| | write ($B$) |
| | commit |

**Question 24-2:**

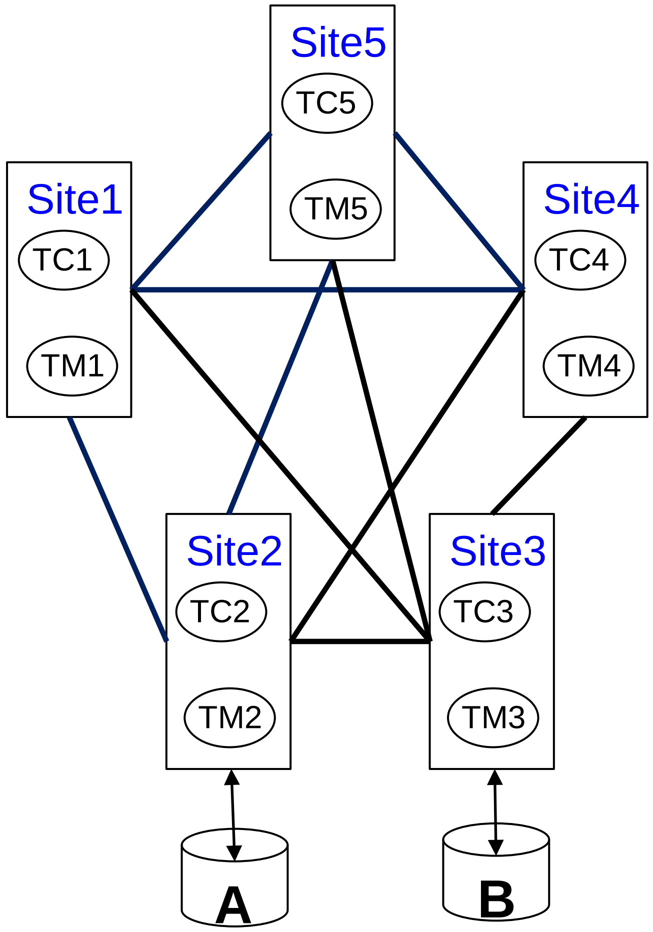
a. Given A = 100 and B= 100. Prove that the above concurrent schedule preserve database consistency.

b. Explain conflict serializability

# Concurrency Control in Distributed Databases

# Concurrency Control

- Concurrency control mechanism maintains the ACID property of transaction by lock based protocol.
- Modify concurrency control schemes for use in distributed environment.
- We assume that each site participates in the execution of a commit protocol to ensure global transaction atomicity.
- We assume all replicas of any item are updated



# Automatic Acquisition of Locks

- A transaction $T_i$ issues the standard read/write instruction, without explicit locking calls.
- The operation **read**(*D*) is processed as:

**if** $T_i$ has a lock on *D*
**then**
read(*D*)
**else begin**
if necessary wait until no other
transaction has a **lock-X** on *D*
grant $T_i$ a **lock-S** on *D*;
read(*D*)
**end**

# Automatic Acquisition of Locks (Cont.)

- **write**(*D)* is processed as:

  **if** $T_i$ has a **lock-X** on *D*
  **then**
  write(*D*)
  **else begin**
  if necessary wait until no other transaction has any lock on *D*,
  if $T_i$ has a **lock-S** on *D*
  **then**
  **upgrade** lock on *D* to **lock-X**
  **else**
  grant $T_i$ a **lock-X** on *D*
  write(*D*)
  **end**;

- All locks are released after commit or abort

# Single-Lock-Manager Approach

In the **single lock-manager** approach, lock manager runs on a *single* chosen site, say Site1. All lock requests sent to central lock manager

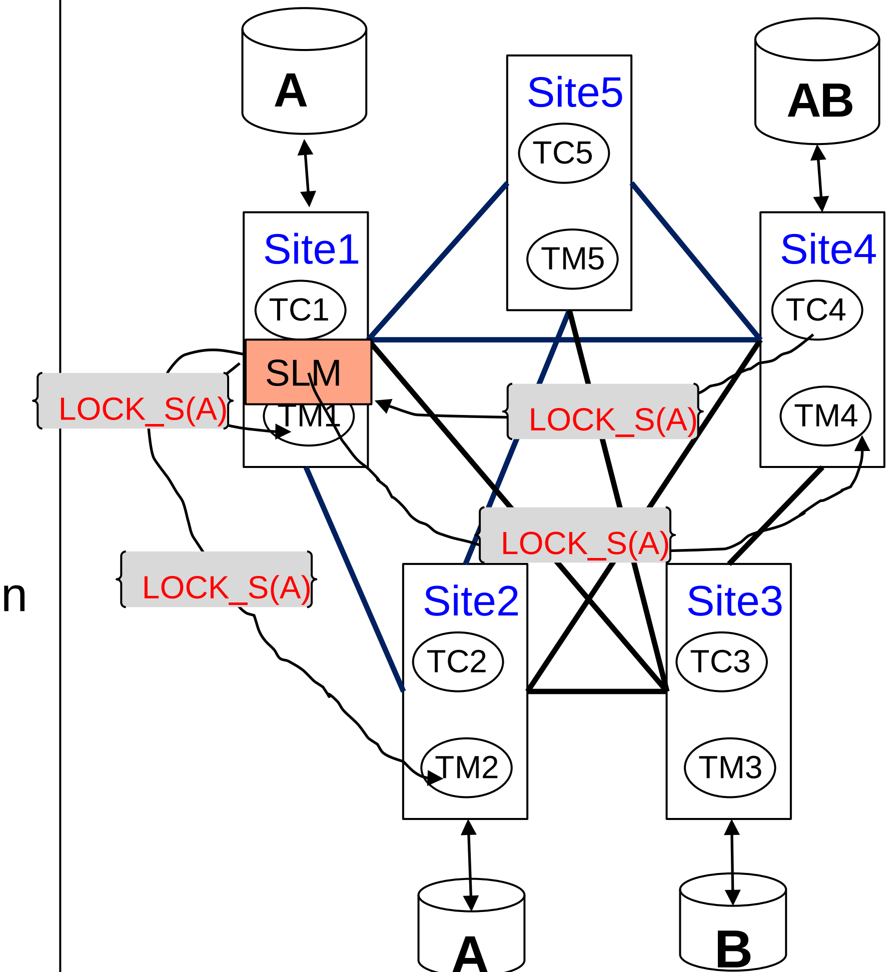
**How to perform READ (A)?**

The transaction can read the data item from *any* one of the sites at which a replica of the data item resides.

**Question 25-1** Let us consider the transaction T41 at site 4. The transaction display of balance of account A. There is single lock manager at site 1.

i. Write the transaction with lock.

ii. Write the steps to obtain the lock by T41.



# Single-Lock-Manager Approach

In the **single lock-manager** approach, lock manager runs on a *single* chosen site, say Site1. All lock requests sent to central lock manager

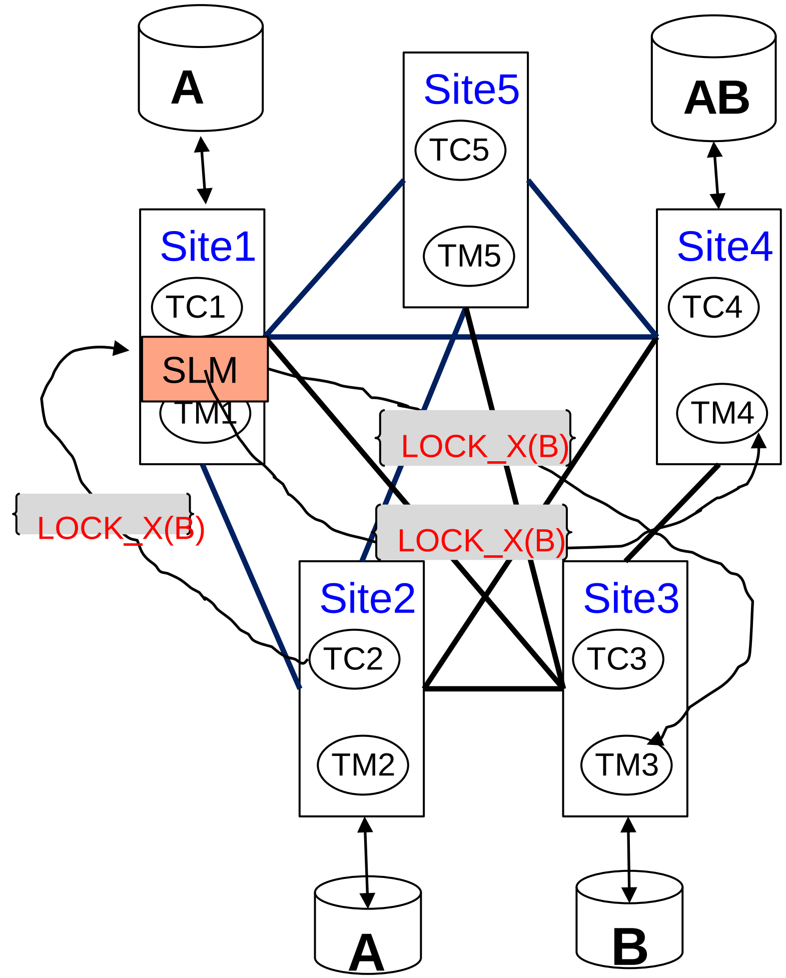
**How to perform write (B)?**

The transaction has to write the data item to all of the sites at which a replica of the data item resides.

**Question 25-2** Let us consider the transaction T21 at site 2. The transaction add Tk. 1000 to account B. There is single lock manager at site 1.

i. Write the transaction with lock.

ii. Write the steps to obtain the lock by T21.



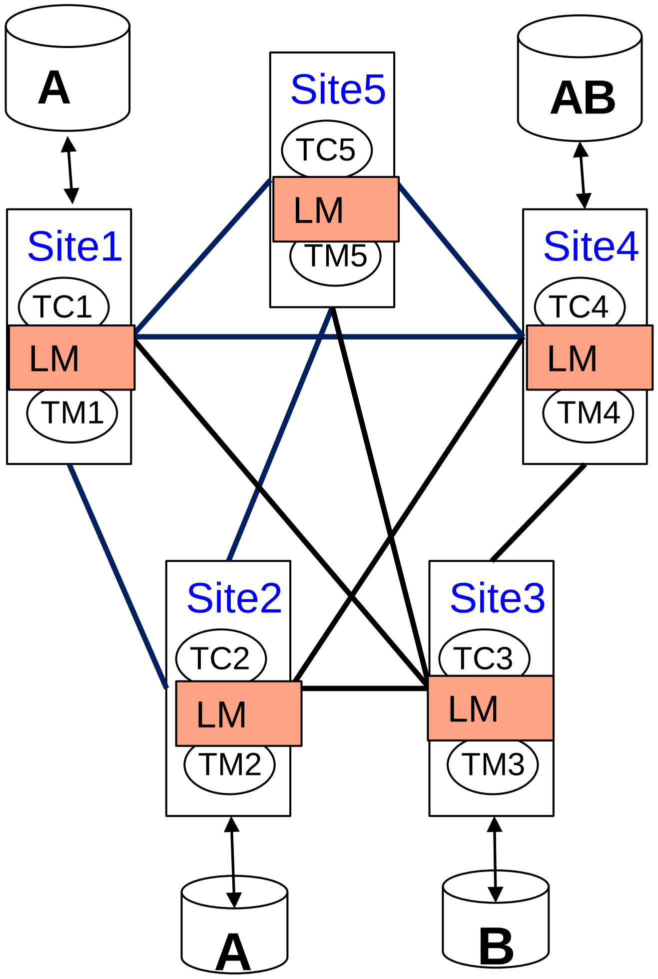

**Writes must be performed on all replicas of a data item**

# Single-Lock-Manager Approach

- Advantages of scheme:
  - Simple implementation
  - Simple deadlock handling
- Disadvantages of scheme are:
  - Bottleneck: lock manager site becomes a bottleneck
  - Vulnerability: system is vulnerable to lock manager site failure.

# Distributed Lock Manager

- In the **distributed lock-manager** approach, functionality of locking is implemented by lock managers at each site
- Lock managers control access to local data items
- Locking is performed separately on each site accessed by transaction
- Every replica must be locked and updated
- But special protocols may be used for replicas (more on this later)

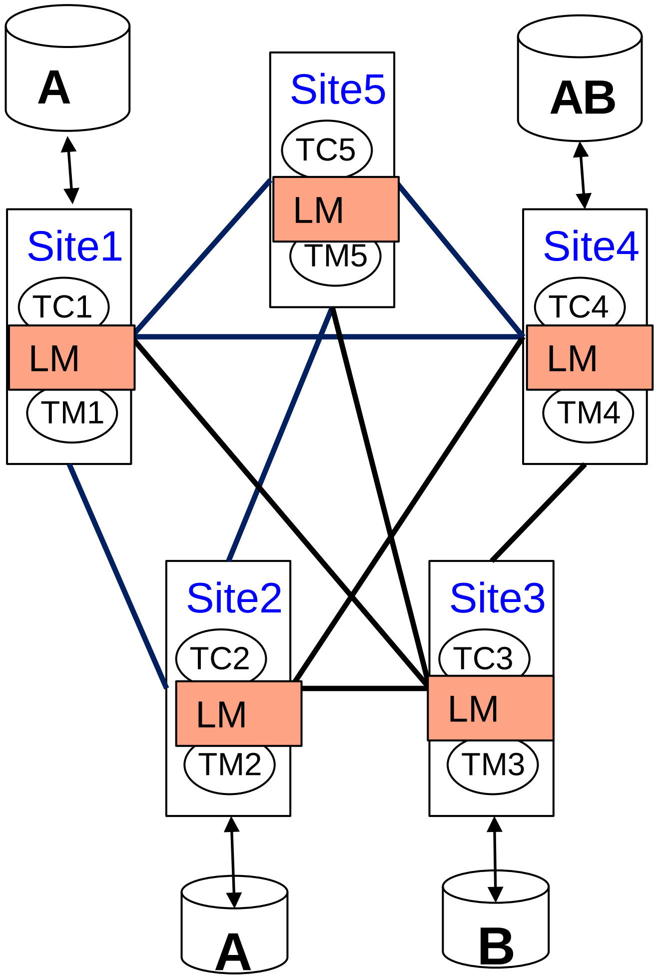


# Distributed Lock Manager

Advantage: work is distributed and can be made robust to failures

Disadvantage:

- Possibility of a global deadlock without local deadlock at any single site
- Lock managers must cooperate for deadlock detection




Compare single lock manager and distributed lock manager approach.

# Replication

- **High availability** is a key goal in a distributed database
  - **Robustness**: the ability to continue function despite failures

- Replication is key to robustness

- Replication decisions can be made at level of data items, or at the level of partitions

# Concurrency Control With Replicas

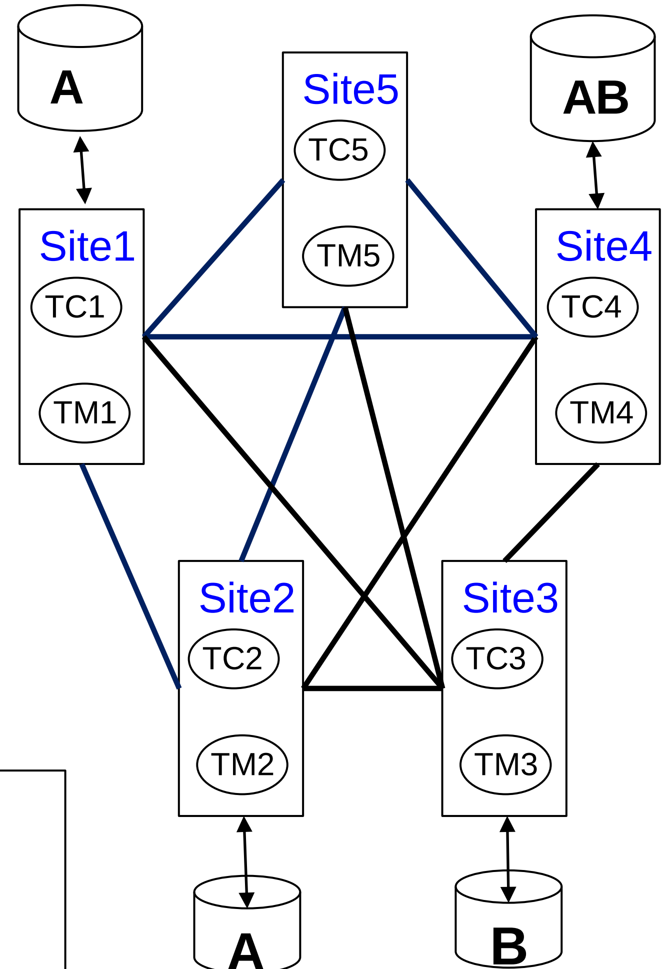
Focus here on concurrency control with locking

Failures addressed later

Ideas described here can be extended to other protocols

**Primary copy**

- one replica is chosen as primary copy for each data item (e.g., A in site 2)
- Node containing primary replica is called **primary node (site 2 for A)**
- concurrency control decisions made at the primary copy only (by site 2)




**Example**: How LOCK-X(A) of T10 at site 3 will be obtained using primary protocol? Site 2 is primary copy for A.

1. site 3 sends LOCK-X(A) request to sites 2.
2. If T10 gets lock on A, then A is locked and T10 proceeds

# Concurrency Control With Replicas

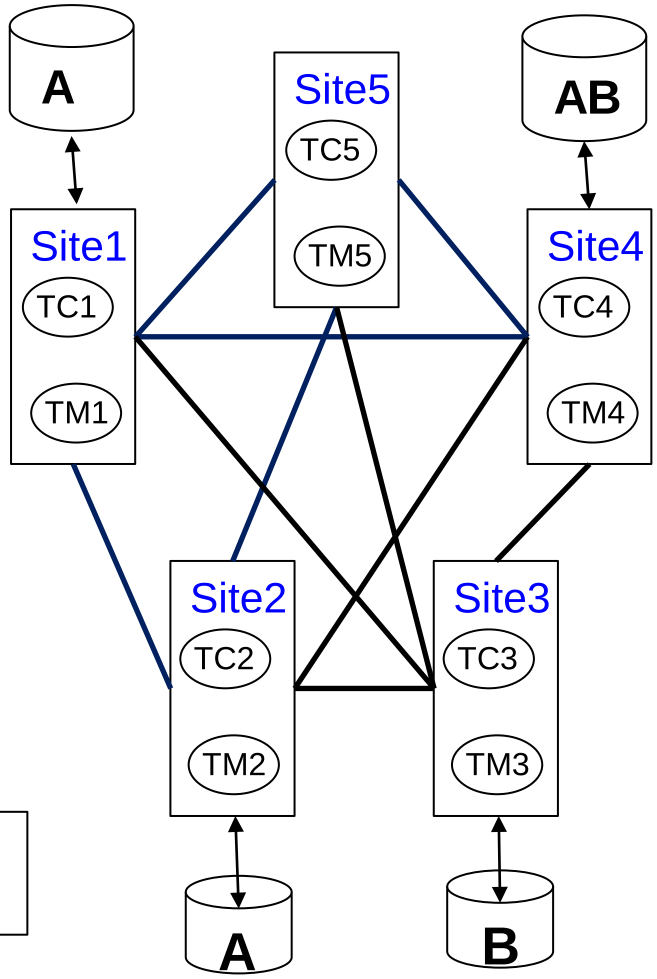
Focus here on concurrency control with locking

Failures addressed later

Ideas described here can be extended to other protocols

**Primary copy**

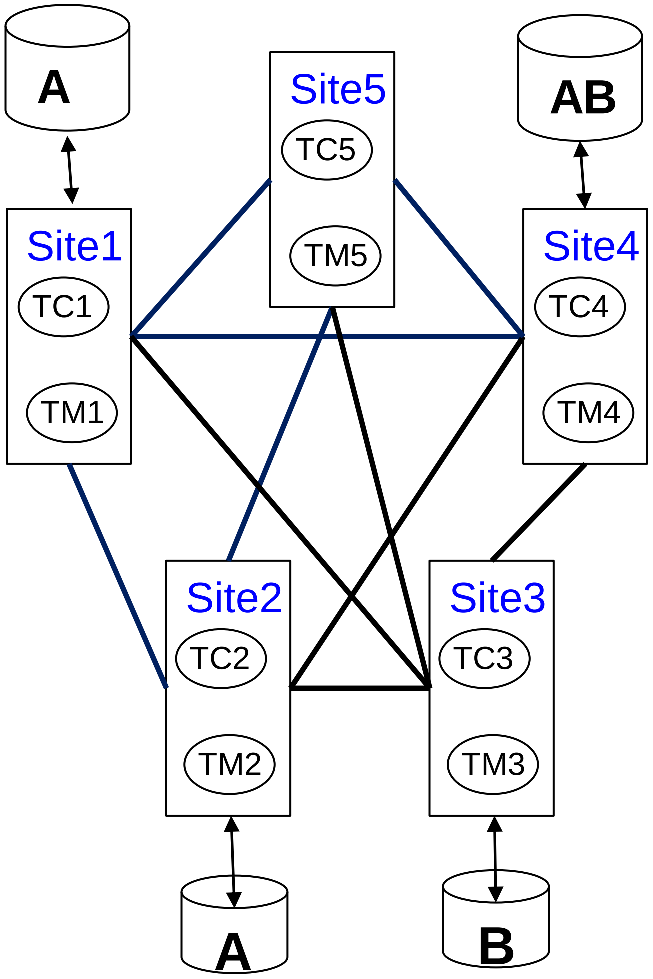
- one replica is chosen as primary copy for each data item (e.g., A in site 2)
- Node containing primary replica is called **primary node (site 2 for A)**
- concurrency control decisions made at the primary copy only (by site 2)




**Question**: How LOCK-X(B) of T2 at site 2 will be obtained using primary copy protocol? Site 4 is primary copy for B.

# Concurrency Control With Replicas

- Benefit: Low overhead of locking **(How?)**
- Drawback: primary copy failure (site 2 for A) results in loss of lock information and non-availability of data item (A), even if other replicas (site 1, site 4) are available
- Extensions to allow backup server to take over possible, but vulnerable to problems on network partition



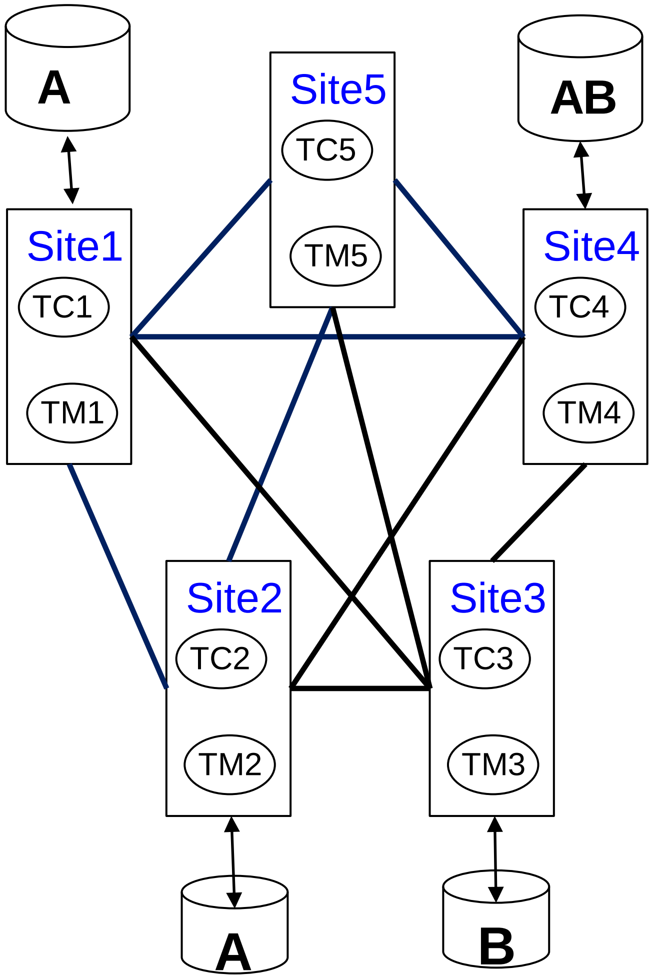
# Concurrency Control With Replicas (Cont.)

**Majority protocol**:

- Transaction requests locks at majority/all replicas
- Lock is successfully acquired on the data item only if lock obtained at a majority of replicas

**Example**: How LOCK-X(A)/ LOCK-S(A) of T10 at site 3 will be obtained using majority protocol?

1. site 3 sends LOCK-X(A)/ LOCK-S(A) request to sites 1, 2 or 2,4 or 1,4 or 1,2,4.
2. If T1 gets lock on 2 sites, then A is locked and T1 proceeds



# Concurrency Control With Replicas (Cont.)

## Number of messages in Majority Protocol

Benefit:

- Resilient to node failures
- Processing can continue as long as at least a majority of replicas are accessible

Overheads

- Higher cost due to multiple messages
- Possibility of deadlock even when locking single item

**Example:** A data item Q is replicated to 10 sites. Transaction T1 has LOCK-S(A). What is the minimum number of messages required to obtain this lock using majority protocol?

Messages for lock request
= 10/2 + 1 = 6

Messages for lock grant
= 10/2 + 1 = 6

Messages for unlock
= 10/2 + 1 = 6

Total = 6+6+6 = 18

# Concurrency Control With Replicas (Cont.)

## Number of messages in Majority Protocol

Benefit:

- Resilient to node failures and node failures
- Processing can continue as long as at least a majority of replicas are accessible

Overheads

- Higher cost due to multiple messages
- Possibility of deadlock even when locking single item

**Question 26-1:** A data item P is replicated to 12 sites. Transaction T2 has LOCK-X(P).
Find  the minimum number of messages required to

a. obtain this lock  using majority protocol?
b. unlock

# Concurrency Control With Replicas (Cont.)

## Number of messages in Majority Protocol

Benefit:

- Resilient to node failures and node failures
- Processing can continue as long as at least a majority of replicas are accessible

Overheads

- Higher cost due to multiple messages
- Possibility of deadlock even when locking single item

How can you avoid such deadlocks?

A data item Q is replicated to ***n*** sites. Transaction T1 has LOCK-S(A).
What is the minimum number of messages required to obtain this lock?

Messages for lock request
= n/2 + 1

Messages for lock grant
= n/2 + 1

Messages for unlock
= n/2 + 1

Total = 3(n/2 + 1)

# Concurrency Control With Replicas (Cont.)

## Number of messages in Primary Copy Protocol

**Example**: A data item Q is replicated to 10 sites. Transaction T1 has LOCK-S(A).
What is the number of messages required to obtain this lock using primary copy protocol?

Messages for lock request
= 1

Messages for lock grant
=1

Messages for unlock
= 1

Total = 1+1+1 = 3

# Concurrency Control With Replicas (Cont.)

**Biased protocol**

- Shared lock can be obtained on any replica
- Reduces overhead on reads
- Exclusive lock must be obtained on *all* replicas
- Blocking if any replica is unavailable

**Example:** A data item Q is replicated to 10 sites. Transaction T1 has LOCK-X(A).
What is the minimum number of messages required to obtain this lock?

No. of messages = 3*10 = 30

**Example:** A data item Q is replicated to 10 sites. Transaction T1 has LOCK-S(A).
What is the minimum number of messages required to obtain this lock?

Messages for lock request
= 1
Messages for lock grant
=1
Messages for unlock
= 1
Total = 1+1+1 = 3

# Concurrency Control With Replicas (Cont.)

**Biased protocol**

- Shared lock can be obtained on any replica
- Reduces overhead on reads
- Exclusive lock must be obtained on *all* replicas
- Blocking if any replica is unavailable

**Question26-2:** Data item A and B are replicated to 15 sites. Transaction T11 has LOCK-X(A) and LOCK-S(B).

a. What are the minimum number of sites required to obtain these locks using biased protocol?
b. Find the number of messages to obtain lock and release lock in each case.
c. Compare biased protocol with majority protocol in terms of performance.

# Quorum Consensus Protocol

**Quorum consensus** protocol for locking
Each site is assigned a weight; let S be the total of all site weights
Choose two values **read quorum** $Q_R$ and **write quorum** $Q_W$
Such that $Q_r + Q_w > S$ and $2 * Q_w > S$
Each read must lock enough replicas that the sum of the site weights is $\geq Q_r$
Each write must lock enough replicas that the sum of the site weights is $\geq Q_w$
Can choose $Q_r$ and $Q_w$ to tune relative overheads on reads and writes

Suitable choices result in majority and biased protocols.

**What are they?**

**Example:** The data item P is replicated in 9 sites and the weight of each site is 1. $Q_w$ is 7 and $Q_r$ is 3. S=9
Find the number of replicas to be locked for

i. LOCK-S
ii. LOCK-X

**For Lock-S, Qr = 3,**

Sum of site weight = 3

Number of sites = 3

**For Lock-X, Qw = 7**

Sum of site weight = 7

Number of sites = 7

# Quorum Consensus Protocol

**Quorum consensus** protocol for locking

Each site is assigned a weight; let S be the total of all site weights

Choose two values **read quorum** $Q_R$ and **write quorum** $Q_W$

Such that $Q_r + Q_w > S$ and $2 * Q_w > S$

Each read must lock enough replicas that the sum of the site weights is ≥ $Q_r$

Each write must lock enough replicas that the sum of the site weights is ≥ $Q_w$

Can choose $Q_r$ and $Q_w$ to tune relative overheads on reads and writes

Suitable choices result in majority and biased protocols.

**What are they?**

**Example:** The data item P is replicated in 9 sites S1, S2, S3, S4, S5, S6, S7, S8, S9 and the weight of the sites are1,1,1,2,2,2,3,3,3 respectively. $Q_w$ is 12 and $Q_r$ is 7.

Find the replicas to be locked for

i. LOCK-S(P)

ii. LOCK-X(P)

a. Starting the sites with lowest weight with ascending order.

b. Starting the sites with highest weight with descending order.

**a. Ascending order**

Weights S1 + S2 + S3 + S4 + s5
= 1+1+1+2+2= 7 = $Q_r$
For LOCK-S(P), sites to have locked is S1, S2, S3, S4, s5

Weights S1+S2+S3+S4+s5+S6+S7
= 1+1+1+2+2+2+3= 12= $Q_w$
For LOCK-X(P), sites to have locked is S1, S2, S3, S4, s5,S6,S7

# Quorum Consensus Protocol

**Quorum consensus** protocol for locking

Each site is assigned a weight;  let S be the total of all site weights

Choose two values **read quorum** $Q_R$ and **write quorum** $Q_W$

Such that $Q_r + Q_w > S$ and $2 * Q_w > S$

Each read must lock enough replicas that the sum of the site weights is $\geq Q_r$

Each write must lock enough replicas that the sum of the site weights is $\geq Q_w$

Can choose $Q_r$ and $Q_w$ to tune relative overheads on reads and writes

Suitable choices result in majority and biased protocols.

**What are they?**

**Example:** The data item P is replicated in 9 sites S1, S2, S3, S4, S5, S6, S7, S8, S9 and the weight of the sites are1,1,1,2,2,2,3,3,3 respectively. $Q_w$ is 12 and $Q_r$ is 7.

Find the replicas to be locked for

i. LOCK-S(P)

ii. LOCK-X(P)

a. Starting the sites with lowest weight with ascending order.

b. Starting the sites with highest weight with descending order.

**b. Descending order**

Weights S9+S8+S7
= 3+3+3= 9 > $Q_r$
For LOCK-S(P), sites to have locked are S9, S8, S7

Weights S9+S8+S7+S6+S5
= 3+3+3+2+2= 13 > $Q_w$
For LOCK-X(P), sites to have locked are S9,S8,S7,S6,S5

# Quorum Consensus Protocol

**Quorum consensus** protocol for locking
Each site is assigned a weight; let S be the total of all site weights
Choose two values **read quorum** $Q_R$ and **write quorum** $Q_W$
Such that $Q_r + Q_w > S$ and $2 * Q_w > S$
Each read must lock enough replicas that the sum of the site weights is $\geq Q_r$
Each write must lock enough replicas that the sum of the site weights is $\geq Q_w$
Can choose $Q_r$ and $Q_w$ to tune relative overheads on reads and writes

Suitable choices result in majority and biased protocols.

**What are they?**

**Question:** The data item P is replicated in 8 sites S1, S2, S3, S4, S5, S6, S7, S8 and the weight of the sites are 2,2,2,3,3,3,4,4 respectively.

a. Find $Q_w$ and $Q_r$.

Find the replicas to be locked for

i. LOCK-S(P)
ii. LOCK-X(P)

b. Starting the sites with lowest weight with ascending order.

c. Starting the sites with highest weight with descending order.