南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

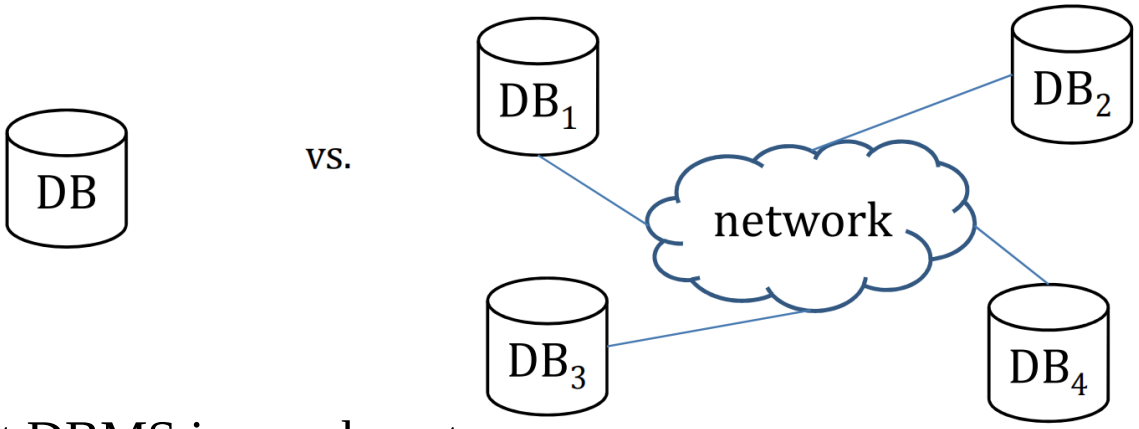
DBGroup

Distributed Databases II

南方科技大学
唐　博
tangb3@sustech.edu.cn

Distributed Database

- Centralized DB vs. Distributed DB



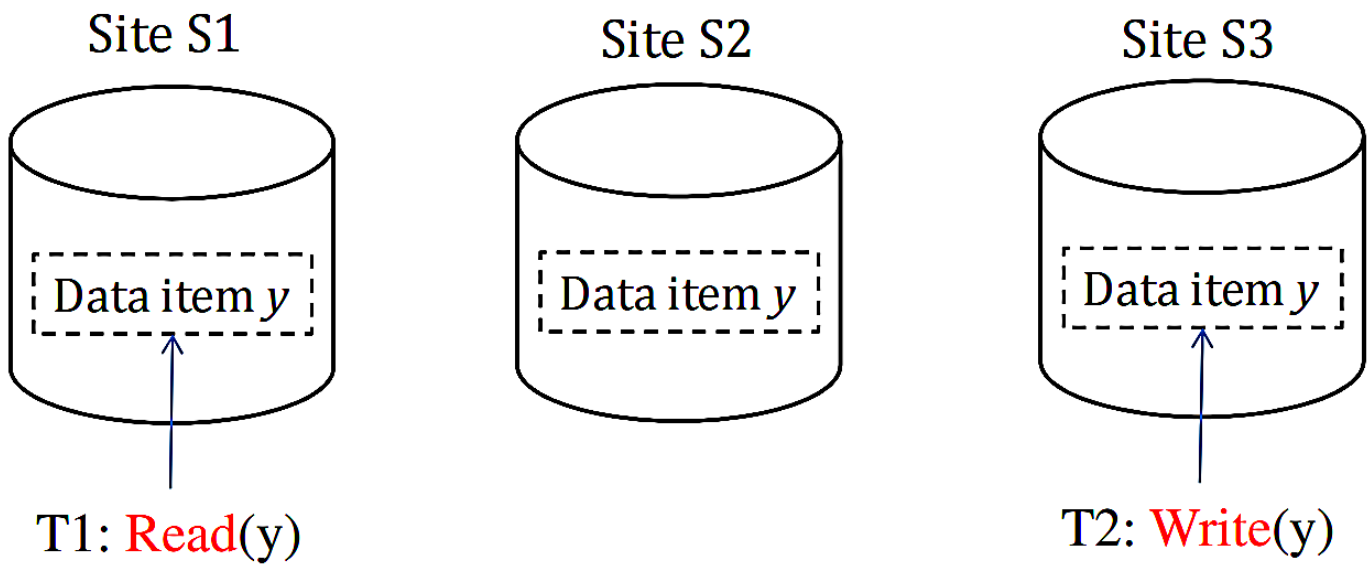
- Revisit DBMS issues, how to:
  - Store data? [covered]
  - Ensure ACID properties?
    - Recovery for ‘A’, ‘D’ [covered]
    - Concurrency for ‘I’ [study today]
  - Process a query fast? [study today]

Lecture Objectives

- Distributed Concurrency Control
  - Distributed Protocols
  - Deadlock Handling
- Availability
- Distributed Query Processing

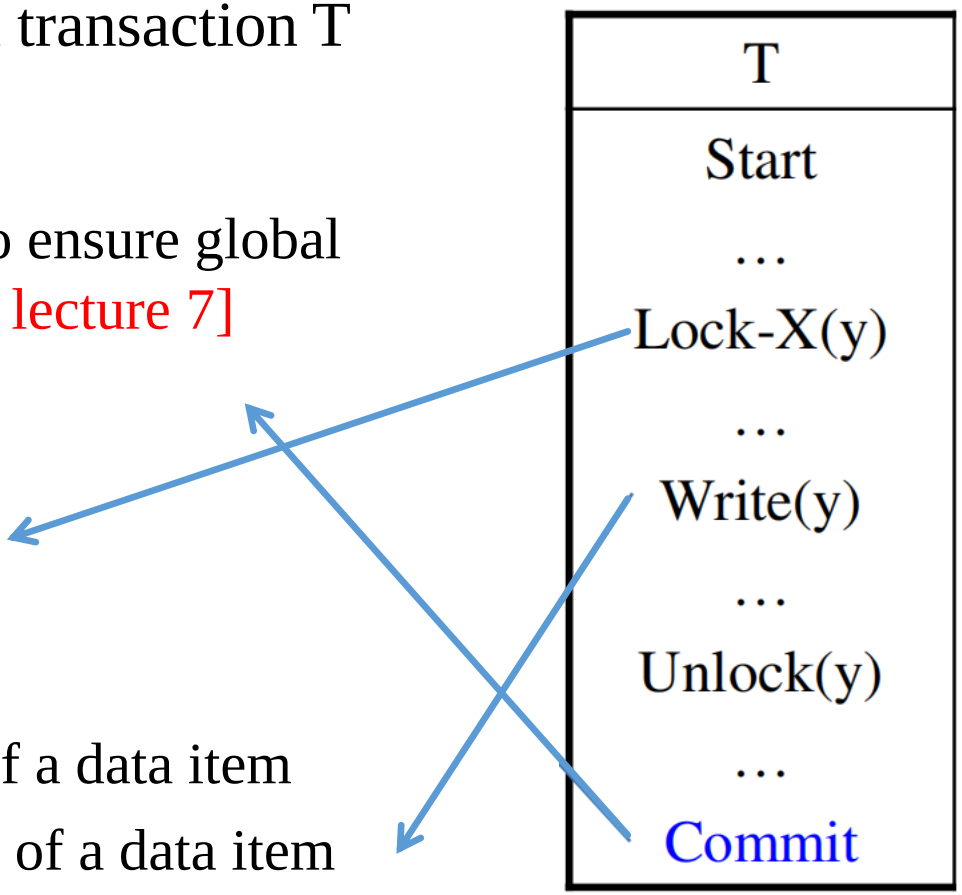
Distributed Database Example

- Distributed environment
  - May not have a centralized server
  - A data item can have replicas at different sites
- How about concurrency control in this setting?
  - Consider two transactions T1 and T2 below



Concurrency Control

- Consider the execution of a transaction T
- Commit / Abort operations
  - Use a **commit protocol** to ensure global transaction atomicity [see lecture 7]
- Lock / Unlock operations
  - How to get a lock?
- Read / Write operations
  - can **read** on ***any*** replica of a data item
  - must **write** on ***all*** replicas of a data item
  - How to deal with site failures?

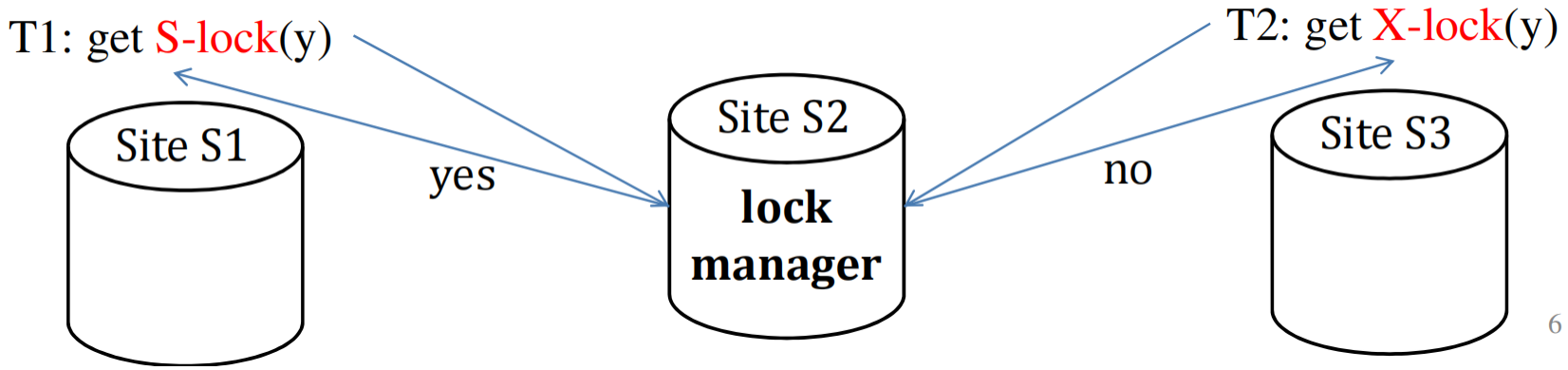


Single-Lock-Manager Approach

- A *single* chosen site S_L maintains a lock manager
- Let S_{init} be the initiator site of transaction T
- When T needs to lock a data item,
  1. it sends a **lock request** to S_L and
  2. the **lock manager** (at S_L) decides whether to grant the lock

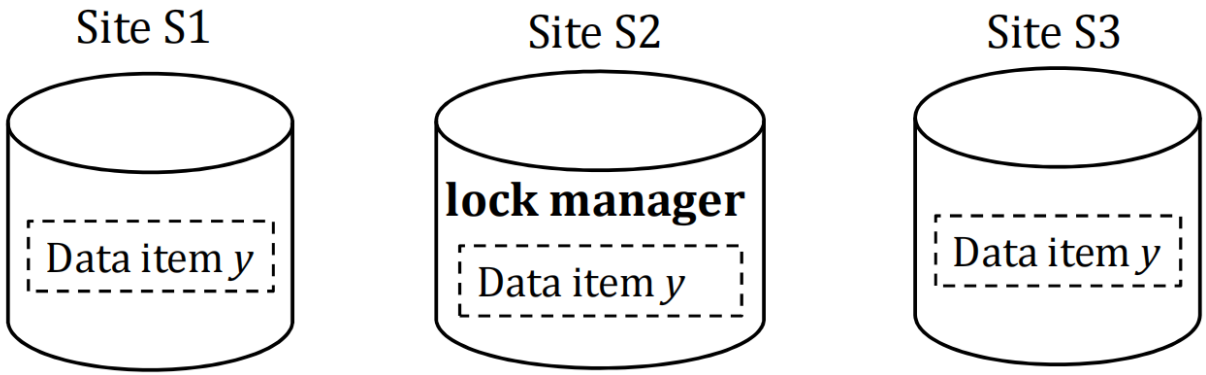
     If yes → it sends a message to S_{init}

     If no → it delays the request until it can be granted



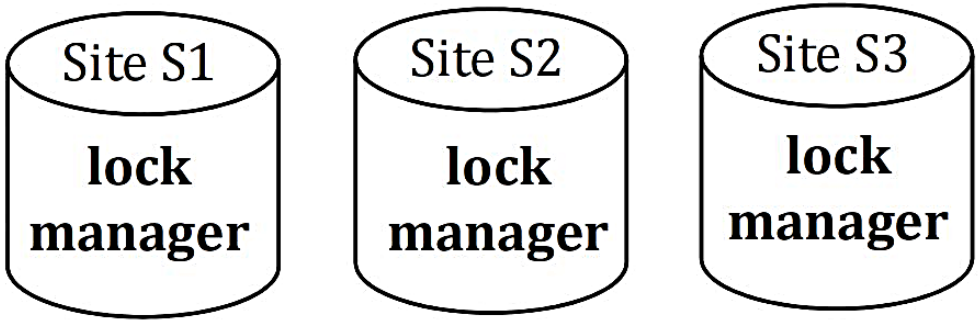
Single-Lock-Manager Approach

- A transaction T
  - can read on any replica of a data item
  - must write on all replicas of a data item
- Advantages:
  - Simple implementation (and few messages per request)
  - Simple deadlock handling
- Disadvantages:
  - Bottleneck at the lock manager site
  - Vulnerable to failure at the lock manager site



Distributed Lock Manager

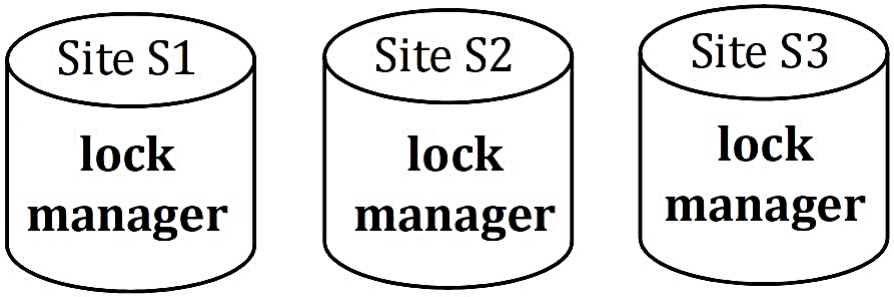
- In this approach, the locking functionality is implemented by lock managers at each site
  - Lock managers control access to **local data items**
  - But require specialized protocols for replicas
- Advantage: robust to failures, no bottleneck at one site
- Disadvantage: complicated deadlock detection
  - Lock managers cooperate to detect deadlock



Distributed Lock Manager

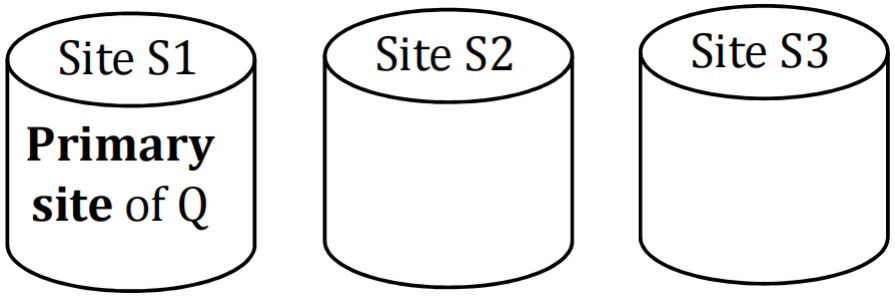
- Variants of this approach (for handling replicas)
  - Details will be provided in the following slides

	Primary copy	Majority protocol	Biased protocol
Robustness to site failures	Bad	Good	Good
S-lock cost	Low	High	Low
X-lock cost	Low	High	Very high



Primary Copy

- Choose one site as the **primary site** of a data item Q
  - Different data items can have different primary sites
- T requests a lock at the primary site of Q [before using Q]
- Benefit
  - Low cost for lock operations
- Drawback
  - Primary site of Q fails → Q becomes inaccessible



Majority Protocol

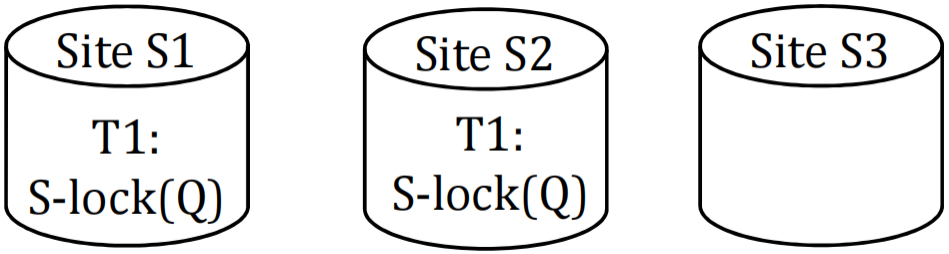
- Local lock manager at each site handles lock/unlock requests for data items stored at that site

- To get lock on data item Q:
  - T sends lock request messages to a **majority** (i.e., more than half) of the sites which store Q
  - T waits until it has obtained locks on a majority of sites

- Write operation: T performs write on ***all*** replicas

Majority Protocol (Cont.)

- Why this protocol works?
  - Suppose that *T1* plans to read *Q* and obtained S-locks of *Q* on a majority of sites
  - *T2* plans to write *Q*
  - Can *T2* obtain X-lock of *Q* (incompatible with S-lock) on a majority of sites?
    - How many site(s) allow *T2* to obtain X-lock of *Q*?



- When *T* issues "write *Q*", why it must write on all replicas?

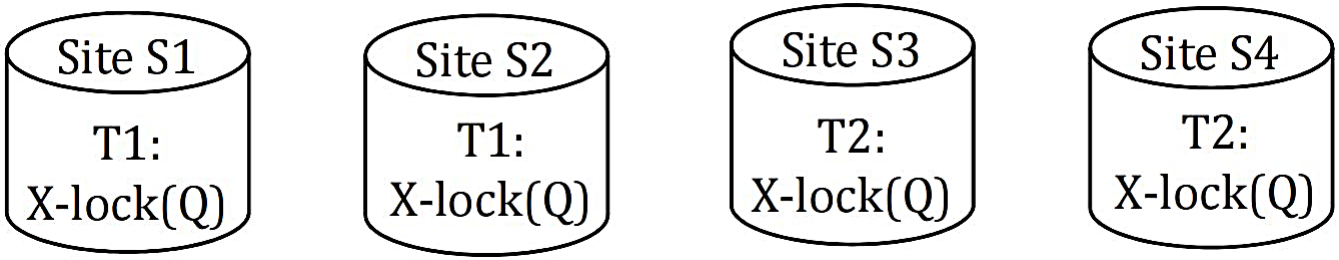
Majority Protocol: Pros and Cons

- Advantage
  - Applicable even when some sites are unavailable
    - Later we discuss how to handle writes in the presence of site failure
- Disadvantage
  - Requires O(n) messages for handling lock/unlock requests
    - n = number of sites that store the replica
  - Potential for deadlock even with single item
    - Why?

Majority Protocol

can have deadlock with a single item!

- Example
  - A data item Q is replicated at four sites: *S1, S2, S3, S4*
  - Suppose that *T1* and *T2* have acquired X-locks as follows



  - Can *T1* or *T2* obtain X-locks of *Q* at a majority of sites?
- Solution
  - Require all transactions to request locks on replicas in the same pre-defined order (e.g., *S1, S2, S3, S4*)

Biased Protocol

- Like the majority protocol, but S-locks and X-locks require different number of lock requests

- To get **S-lock** on data item Q:
  - T requests a lock on Q at 1 site containing a replica of Q
- To get **X-lock** on data item Q:
  - T requests a lock on Q at ALL sites containing a replica of Q

- Advantage: less overhead on **read** operations
- Disadvantage: more overhead on **write** operations

Lecture Objectives

- Distributed Concurrency Control
  - Distributed Protocols
  - Deadlock Handling
- Availability
- Distributed Query Processing

Deadlock Handling

- Given:
  - item A at site S1, item B at site S2
  - transaction T1 starts at site S1
  - transaction T2 starts at site S2
- Consider the following schedule:

T_1	T_2
write (A)	write (B)
write (B)	write (A)

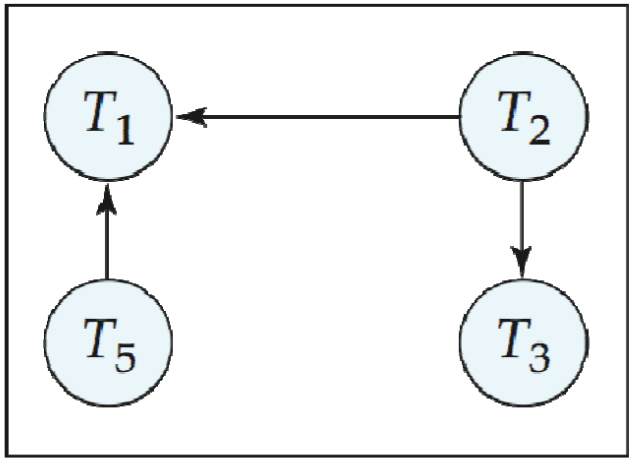
T_1	T_2
X-lock (A)	
write (A)	
	X-lock (B)
	write (B)
	wait for X-lock on A
wait for X-lock on B	

- Problem: this deadlock case cannot be detected locally at either site

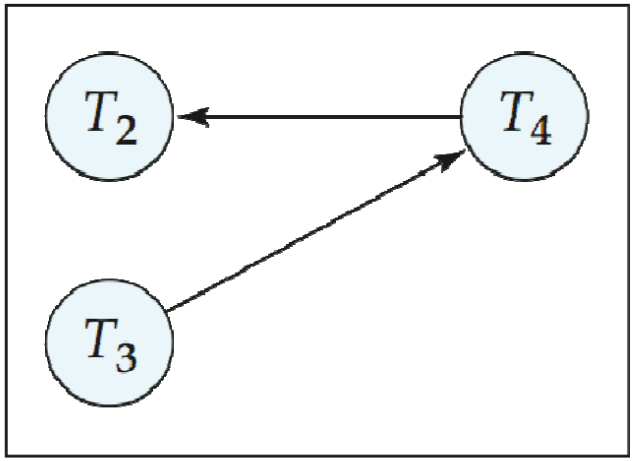
Centralized Approach

- Maintain a global wait-for graph at a *single* site; the deadlock-detection coordinator
- Update the global wait-for graph when:
  - Insert / remove an edge from a local wait-for graph
- The coordinator performs cycle-detection periodically
- If it finds a cycle, then:
  - It selects a victim and notifies all sites, and
  - The sites roll back the victim transaction

Local and Global Wait-For Graphs

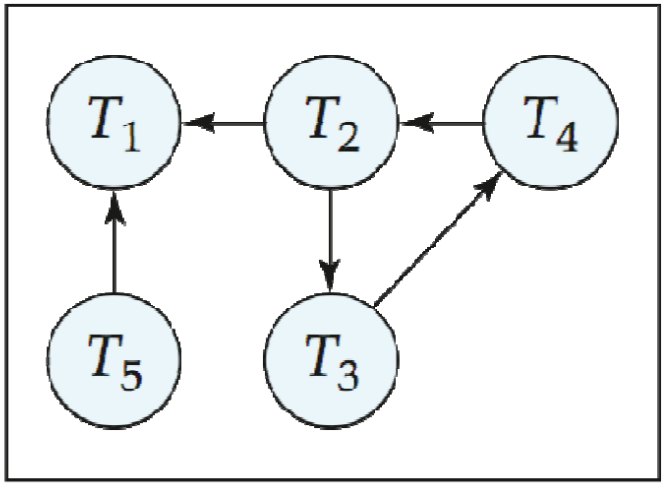


site S_1



site S_2

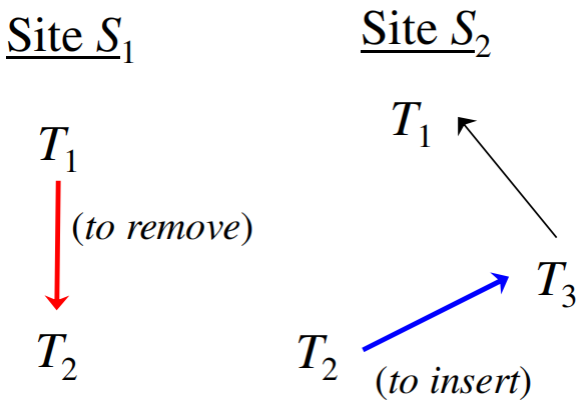
Local



Global

False Cycles

- *Consider this scenario:*
- *1. T_2 releases resources at S_1, coordinator gets "remove $T_1 \rightarrow T_2$"*
- *2. T_2 requests a resource held by T_3 at S_2, coordinator gets "insert $T_2 \rightarrow T_3$"*

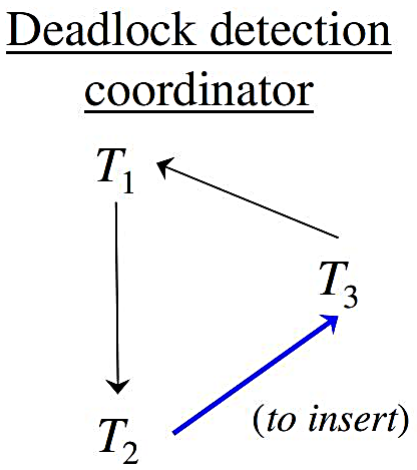


- *Due to network delays, coordinator may get "insert ……" before "delete ……"*
  - *Coordinator finds a **false cycle***

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

  - *But that cycle does not exist in reality*

Solution: avoid false cycles by using two-phase locking



Lecture Objectives

- Distributed Concurrency Control
  - Distributed Protocols
  - Deadlock Handling
- Availability
- Distributed Query Processing

Availability

- Failures may happen in a distributed system
- To be robust, a distributed system must
  - Detect failures
  - Reconfigure the system so computation may continue
  - Recovery/reintegration when a site / link is repaired
- Hard to distinguish link failure from site failure
  - Partial solution:

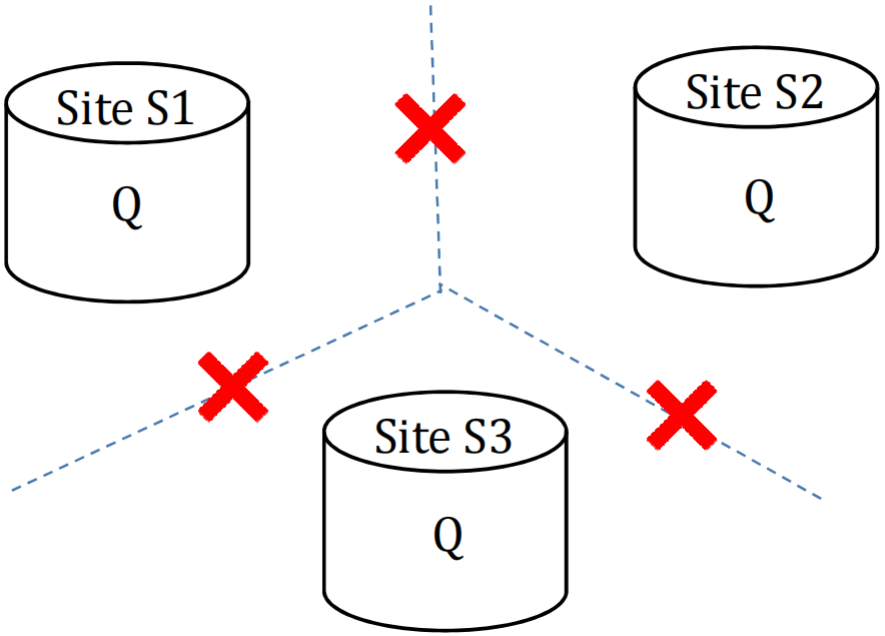
    multiple link failure is likely a site failure

Reconfiguration

- Abort all transactions that were active at a failed site
  - Release the resources (locks) held them by on other sites
- Update the system catalog to exclude replicas stored at a failed site
  - When that site recovers, we will update them (see page 28)
- If a failed site was a central server, conduct **election** to determine the new server
  - E.g., global deadlock detector, concurrency coordinator

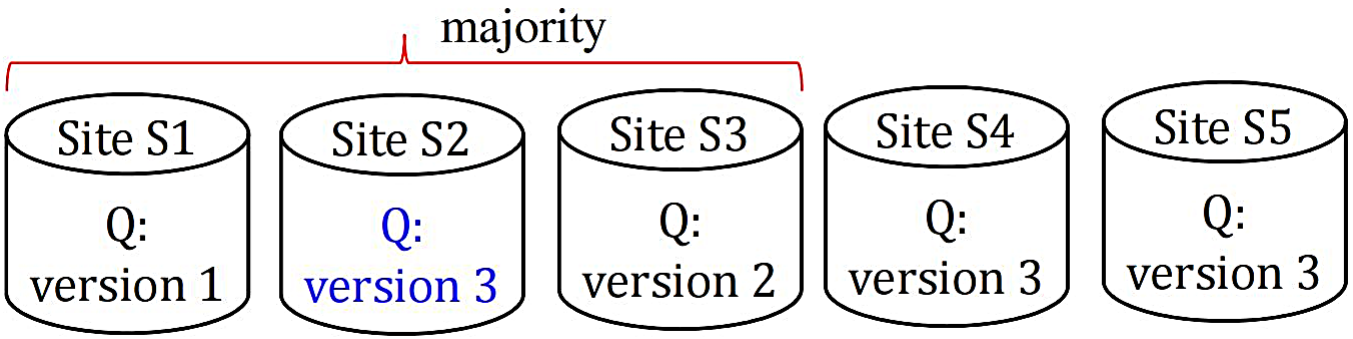
Reconfiguration

- Since network partition may not be distinguishable from site failure, we must avoid these situations:
  - Two or more central servers elected in distinct partitions
  - More than one partition updates a replicated data item



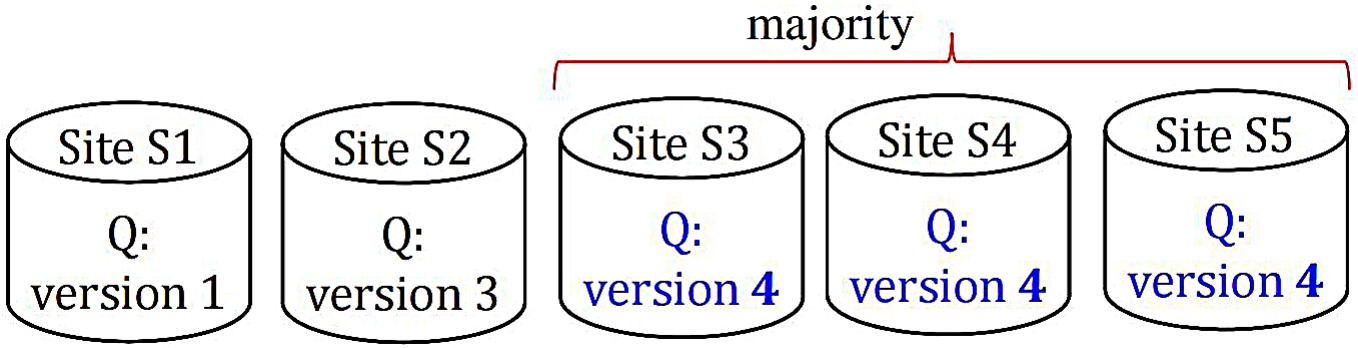
Majority-Based Approach

- The majority protocol can be modified to work even if some sites are unavailable
  - Each item's replica has a **version number**
  - T obtains locks from a majority of sites
    - **majority** means > ½ of sites which store Q (including failed sites)
- Read operations examine all locked replicas, and use the latest replica (i.e., with the largest version number)
  - Optional: update replicas to the latest copy
    (no need to obtain X-locks for this update)

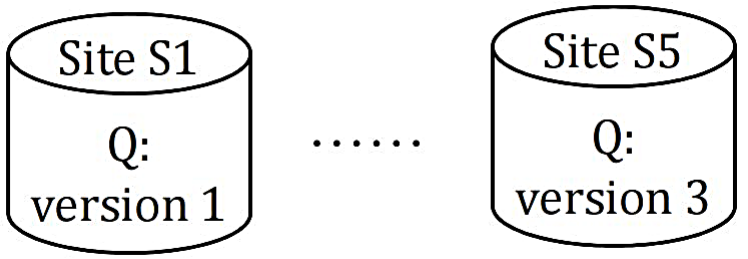


Majority-Based Approach

- Write operations
  - Find the highest version number like read
  - Set the new version number = old highest version + 1
  - **Write** on all locked replicas, and update their version number
- Site/network failures cause no problems as long as
  - Sites at commit have a majority of replicas of any updated data items
  - Each read used a majority of sites to find the latest version



Read-One Write-All



- Read-One Write-All
  - This approach is correct
  - But, when there are failed sites, we cannot write (for any data item with replica on failed sites)
- Read-One Write-All-Available (ignoring failed sites) is ***incorrect***!
  - Temporary link failure → 0001_ a site has old values
  - Reads from that site would return incorrect values
  - With network partitioning, sites in each partition may update same item concurrently
    - believing sites in other partitions have all failed

Site Reintegration

- While a site was down, it may have missed some updates
- When a failed site recovers, it must catch up with those updates that it missed
- Problem:
  - There may be updates to items whose replica is stored at the site while the site is recovering
- Solution:
  - Lock all replicas of all data items at the site
  - Update them to latest version, then release locks

Lecture Objectives

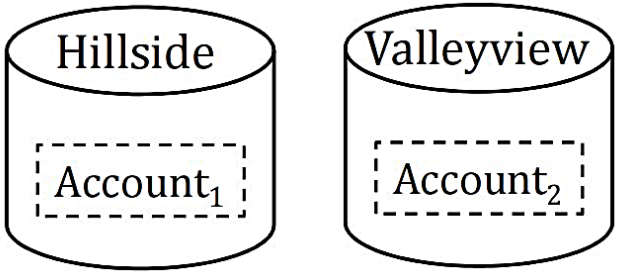
- Distributed Concurrency Control
  - Distributed Protocols
  - Deadlock Handling
- Availability
- Distributed Query Processing

Distributed Query Processing

- For a centralized system,
  cost of a query plan = number of disk block accesses
- In a distributed system, the "best" plan also depends on the following factors:
  - Total amount of data sent over the network
  - Communication cost between sites
  - Processing time at each site (with consideration on processing on multiple sites in parallel)
- In subsequent slides, we only consider the
  total communication cost
  - In assignments/exam, we will state the cost function clearly

Selection Operation

Replace r by its fragments



- Consider horizontal fragmentation of *"account"* relation

$$account_1 = \sigma_{branch\_name = \text{"Hillside"}}(account)$$

$$account_2 = \sigma_{branch\_name = \text{"Valleyview"}}(account)$$

- Given the query: $\sigma_{branch\_name = \text{"Hillside"}}(account)$

$$= \sigma_{branch\_name = \text{"Hillside"}}(account_1 \cup account_2)$$

$$= \sigma_{branch\_name = \text{"Hillside"}}(account_1) \cup \sigma_{branch\_name = \text{"Hillside"}}(account_2)$$

eliminate this operation because $account_1$ has only tuples for "Hillside"

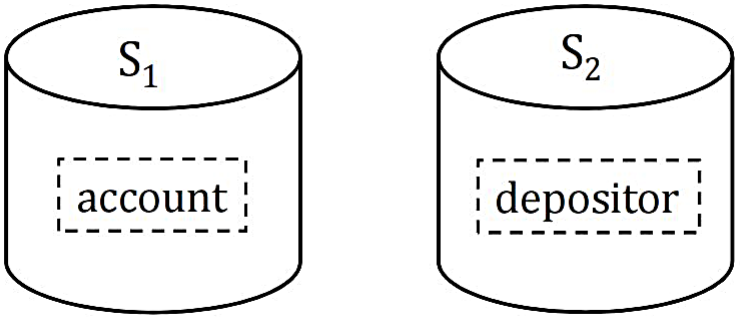
always empty result

- So, just ask the "Hillside" site to return $account_1$

Join Operation

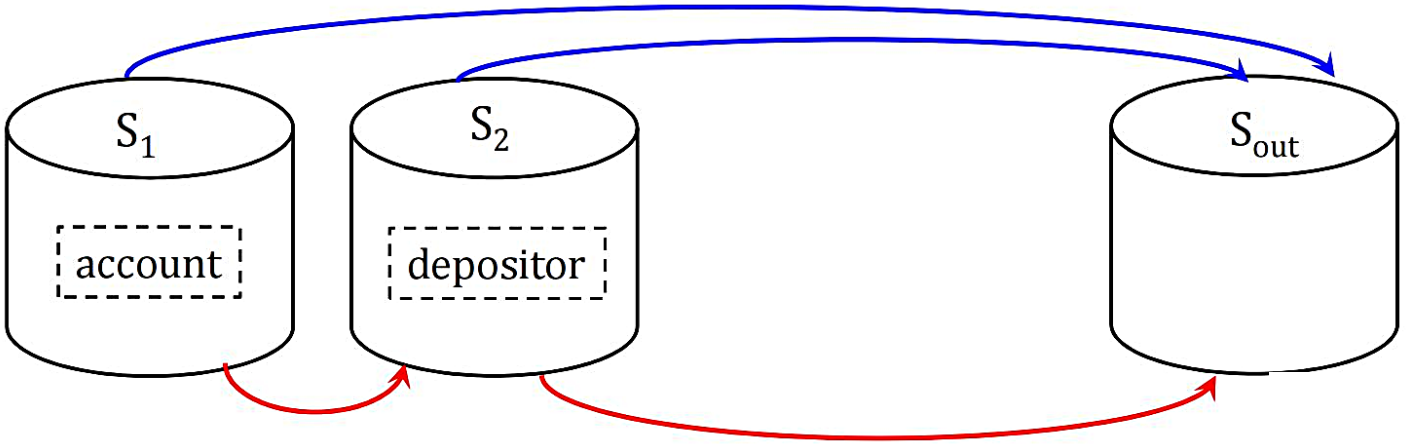
- Example: *account* $|X|$ *depositor*
  - *account at site* S_1, *depositor at* S_2

- Different algorithms to implement joins
  - Method 1: send a copy then join
  - Method 2: semi-join



Method 1: Send a copy then join

- Let site S_{out} be the site to obtain the output
  - S_{out} can be the initiator site S_{init} or decided by the query optimizer
- Consider different ways to send relation(s), choose the **cheapest** way
- Way 1. Send all relations to site S_{out} , then join them at site S_{init}
- Way 2. Send *account* from S_1 to S_2,
  compute *temp* = *account* |X| *depositor* at S_2,
  Send *temp* to S_{out}
- Way 3. Like 2, but we reverse the role of S_1 and S_2



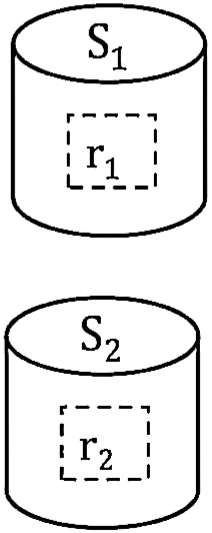
Method 2: Semijoin Method

Let r_1 be a relation (with schema R_1) stored at site S_1

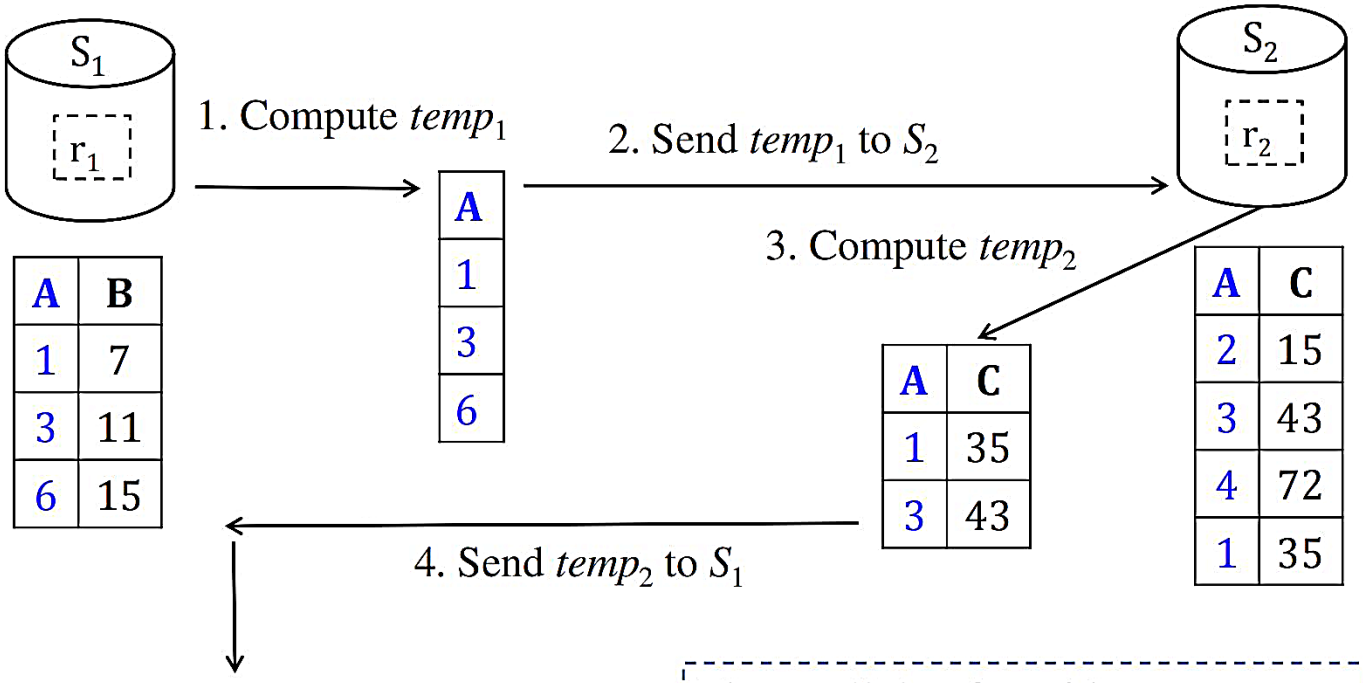
Let r_2 be a relation (with schema R_2) stored at site S_2

Task: evaluate $r_1 \ltimes r_2$ and obtain result at S_1

1. Compute $temp_1 \leftarrow \prod_{R1 \cap R2} (r_1)$ at S_1
2. Send $temp_1$ from S_1 to S_2
3. Compute $temp_2 \leftarrow r_2 \ltimes temp_1$ at S_2
4. Send $temp_2$ from S_2 to S_1
5. Compute $r_1 \ltimes temp_2$ at S_1
   - This is the same as $r_1 \ltimes r_2$



Semijoin Method: Example



A	B
1	7
3	11
6	15

A
1
3
6

A	C
1	35
3	43

A	C
2	15
3	43
4	72
1	35

A	C	B
1	35	7
3	43	11

The **semijoin** of r_2 with r_1:

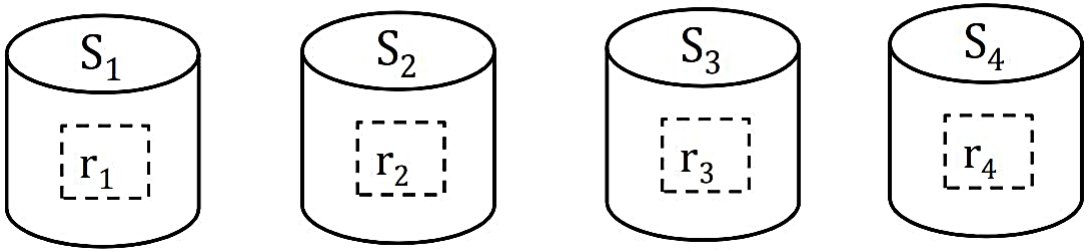
Notation: $r_2 \ltimes r_1$

Meaning: $\prod_{R2} (r_2 \ltimes r_1)$

where R_2 is the schema of r_2

Joining Multiple Tables

- Consider r_1 |X| r_2 |X| r_3 |X| r_4 where relation r_i is stored at site S_i
  - We need to report the result at the initiator site (S_1)
- Two approaches
  - Join the relations one-by-one
  - Parallel execution strategy
    - May not reduce the total communication cost
    - May reduce the total response time



Parallel Execution Strategy

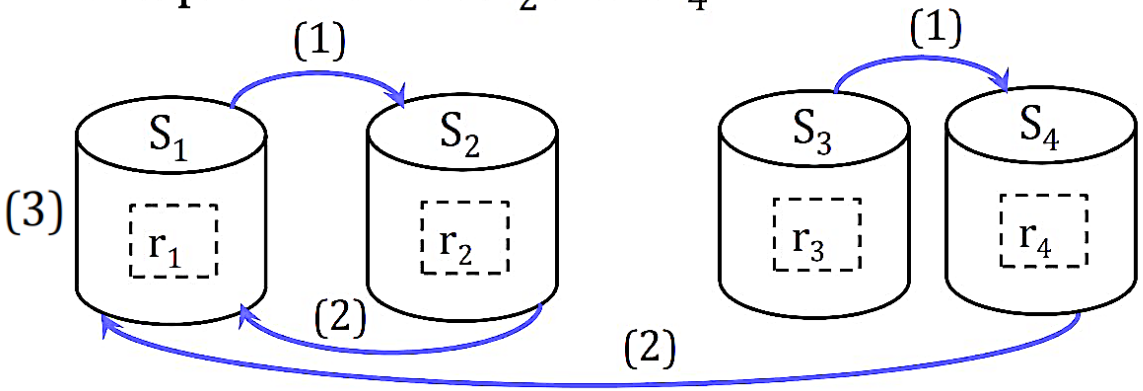
Consider r_1 |X| r_2 |X| r_3 |X| r_4 where relation r_i is stored at site S_i

- Suppose that the result must be reported at site S_1

(1) Send r_1 to S_2 and compute r_1 |X| r_2 at S_2; simultaneously send r_3 to S_4 and compute r_3 |X| r_4 at S_4

(2) S_2 sends tuples of (r_1 |X| r_2) to S_1;
S_4 sends tuples of (r_3 |X| r_4) to S_1

(3) When tuples of (r_1 |X| r_2) and (r_3 |X| r_4) arrive at S_1 then (r_1 |X| r_2) |X| (r_3 |X| r_4) is computed at S_1, in parallel with the computations at S_2 and S_4



Note: Step 1 in the above strategy can be further improved by using semijoin

Summary

- Distributed Concurrency Control
  - Distributed Protocols
  - Deadlock Handling
- Availability
- Distributed Query Processing

Readings after the class

- Chapters 7 and 11 in the book
  Ozsu, and Valuriez. Principles of Distributed Database System, 3rd Ed, Springer, 2011 (free online)

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

DBGroup

谢谢！

DBGroup @ SUSTech

Dr. Bo Tang (唐博)

tangb3@sustech.edu.cn