Distributed Database Management Systems

(Distributed, Multi-database, Parallel, Networked and Replicated DBMSs)

Terms of reference:

Distributed Database: A logically interrelated collection of *shared data* and their description, physically distributed over a computer network.

Distributed Processing: A centralized database, which may be accessed from different computer systems, over an underlying network. (NOT distributed! - i.e. - the db "belongs" to one of the computers).

Multidatabase: A special type of DDBMS, where each site maintains complete autonomy,

Replicated DBMS: A DDBMS that keeps and controls replicate data, such as Relations, in multiple databases.

Parallel DBMS: A DBMS that runs on multiple processors, and disks, but may share same memory. The instructions are executed in parallel, for better performance and throughput.

Networked DBMS: DBMS and/or DDBMS computer systems are linked through a Network with a data communication system. Here accessing takes place using various types of protocols: RPC, TCP/IP, SPX,/IPX (Sequenced Packet Exchange/Internetwork Package Exchange), APPC (Advanced Program-to-Program Communications, from IBM), DECNet, AppleTalk, NetBIOS (IBM & Sytek, for PC's).

Distributed DBMS (DDBMS) consists of a collection of sites, each of which maintains a local db system. So it is a network of computers interconnected by a data communication system so that the physical db is distributed on at least two of the system's components:

Each site on the network is able to process **local Transactions** (i.e. - access data only in that single site)
Each site may participate in the execution of **Global Transactions** (i.e. - access data in several sites) which requires communication among the sites.

Note 1: The above can be thought of: **Local Applications & Global Applications**
Note 2: This scheme is transparent to users.

Homogeneous DDBMS: This is the case when the application programs are ***independent*** of how the db is distributed; i.e. if the distribution of the physical data can be altered without having to make alterations to the application programs. Here, all sites use the same DBMS product - same schemata and same data dictionaries.

Heterogeneous DDBMS: This is the case when the application programs are *dependent* **on** the physical location of the stored data; i.e. application programs must be altered if data is moved from one site to another. Here, there are different kinds of DBMSs (i.e. Hierarchical, Network, Relational, Object., etc.), with different underlying data models.

Characteristics of a DDBMS

A DDBMS developed by a single vendor may contain:

- Data independence
- Concurrency Control
- Replication facilities
- Recovery facilities
- Coordinated Data Dictionary
- Authorization System
- Shared Manipulation Language

Also:

- **Transaction Manager (TM)**
- **Data Manager (DM)**
- **Transaction Coordinator (TC)**

NOTE: a **Distributed Data Processing System** is a system where the application programs run on distributed computers which are linked together by a data transmission network.

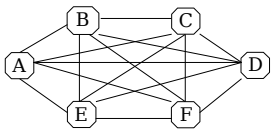
Advantages of DDBMSs	Disadvantages of DDBMSs
More accurately reflects organizational structure	Complexity (Replication overhead, etc)
Shareability and Local Autonomy (enforces global and local policies)	Maintenance Costs (of sites)
Availability and Reliability (failed central db vs failed node)	Security (Network Security)
Performance (process/data migration and speed)	Integrity Control (More complex)
Economics	Lack of Standards
Modular growth	Lack of Experience and Misconceptions
Integration (with older systems)	Database Design more complex

Review of Networking

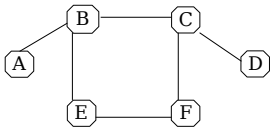
Network: A set of interconnected autonomous computer systems capable of exchanging *data* (broad sense).

Network types include:

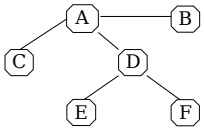
Fully Connected



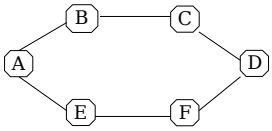
Partially Connected



Tree Structure



Anary/Ring



Considerations when deploying a **DDBMS** (see also above) include:

- *Installation Cost:* physically linking sites in the system.
- *Communication Cost* between sites: both time and money to send a message between two sites
- *Reliability:* The frequency with which a link or a site fails
- *Availability:* The degree to which data can be accessed despite failure of some sites or links

Hardware Links:

- broadband coaxial,
- fiber optics,
- twisted pair,
- baseband coaxial,
- etc.

Communication Time (CT): Time taken to send a message: it depends on the length of the message and the type of network used.
Formula: **CT = Co** + (**#of-bits-in-message/TR**)
where **Co** fixed cost of initiating the message, i.e. the **access** delay and **TR** is the **Transmission Rate**.

Example: Suppose, Co=1 sec, TR=10000 bits/sec then

a) the time it takes to send 100,000 recs of 100 bits is 1001 secs
b) if the recs are sent one at a time,
then CT=100,000 * [1+(100/10,000)] = 101,000 secs.

Example of a Distributed configuration

Suppose we have 4 branches located at different sites. Each branch has its own system with a db consisting of all the accounts maintained at the branch: a site.
There is one single site that maintains information about all of the branches of the bank.

Each **branch** maintains (among others) a relation **deposit,** where
deposit-schema = (branch-name, account#, customer-name, balance)
The centralized site contains a relation **branch** where
branch-schema = (branch-name, assets, branch-city)

Example: A Transaction to add $50 to account# 111 at the Wonderland Branch:

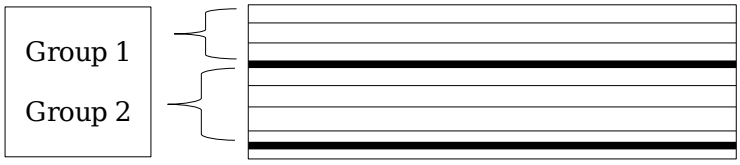
- If the Transaction is initiated at Wonderland then it is a **local transaction, o**therwise it is a **global transaction.**
- A Transaction to transfer $25 from Wonderland account# 111 to Moonland account# 222 at the branch is a **global transaction.**

This a distributed configuration: The various sites are *aware* of one another and each site provides an environment for executing both ***local*** and ***global*** Transactions.

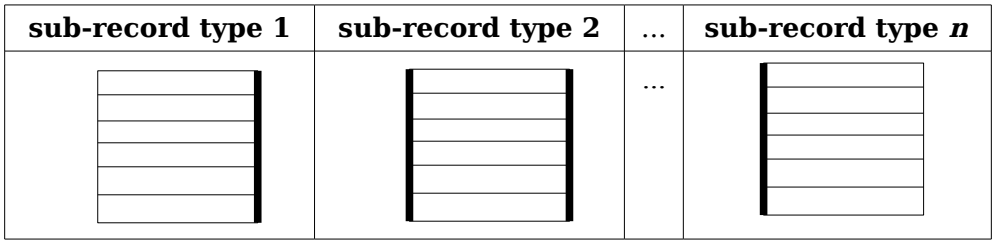
Principles of Data Distribution

The **distribution of data** can be based on *entire files* or on the *subdivisions* of *files.*
There are the following subdivision types: *Horizontal, Vertical and Hybrid subdivisions.*

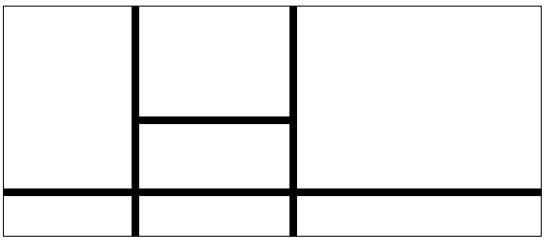
Horizontal: The records of the files are divided into groups which are distributed on different local dbs.



Vertical: A logical record type is divided into two or more physical record types, which can be distributed on different local dbs.



Hybrid: Uses both horizontal and vertical



Example

Suppose we have the relation

Deposit (branch-name, account#, customer-name, balance, tuple-id)

Branch-name	account#	customer-name	balance	tuple-id
Wonderland	305	H.H.	10	1
Wonderland	226	C.B.	5	2
Moonland	177	C.B.	10	3
Moonland	402	B.B.	2	4
Wonderland	155	B.B.	10	5
Moonland	408	B.B.	100	6
Moonland	639	M.M.	1	7

Horizontal fragmentation would produce:

deposit 1

branch-name	account#	customer-name	balance	tuple-id
Wonderland	305	H.H.	10	1
Wonderland	226	C.B.	5	2
Wonderland	155	B.B.	10	5

deposit 2

branch-name	account#	customer-name	balance	tuple-id
Moonland	177	C.B.	10	3
Moonland	402	B.B.	2	4
Moonland	408	B.B.	100	6
Moonland	639	M.M.	1	7

Vertical fragmentation would produce:

Sub1 branch-name	account#	tuple-id
Wonderland	305	1
Wonderland	226	2
Moonland	177	3
Moonland	402	4
Wonderland	155	5
Moonland	408	6
Moonland	639	7

Sub 2 customer-name	balance	tuple-id
H.H.	10	1
C.B.	5	2
C.B.	10	3
B.B.	2	4
B.B.	10	5
B.B.	100	6
M.M.	1	7

NOTE: The tuple-id is a special Attribute added to the relation.
A tuple-id is a physical or logical address for a tuple. Since each tuple in a relation must have a unique address, this Attribute is a key for the augmented scheme.

Replicated Data Bases

A **replicated db** is a special form of db that stores data *redundantly* across different replicated databases.

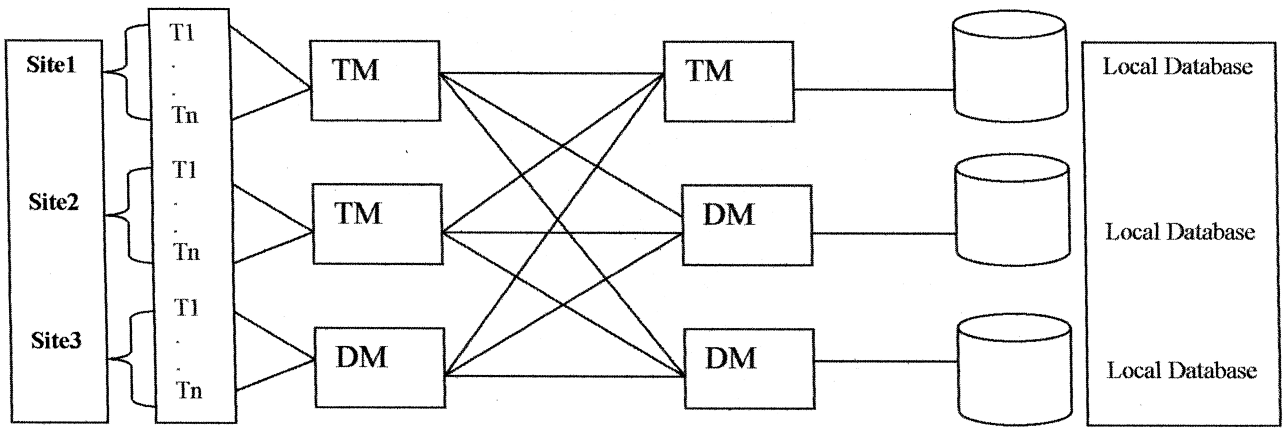
By doing this, we have **shorter response times for queries** are obtained at the expense of **longer processing times for updates**.

- **Availability:** If one of the sites containing relation R fails, then R maybe found at another site so we have continuous operation. Also known as Fault Tolerance.
- **Increased parallelism:** Minimizes the movement of data across sites, but increases overhead for updates, because of replication.

Architecture of a DDBMS

Each computer (site) in a **distributed system** may contain a **Transaction Manager** (**TM**) and a **Data Manager** (**DM**) - as we will see later, there is also a **Transaction Coordinator** (**TC**). The TM is responsible for the Transactions received by the computer. The DM manages the db access on the local computer.

- When a Transaction arrives at the TM, the TM divides the transaction into sub-transactions which are transmitted to those DMs containing the data needed by the Transaction. (In some cases the TC is responsible for this.)
- The TM processes the collected received data from the sub-transactions' responses and produces the final result.
- Any TM can communicate with all DMs and vice versa.



NOTE: The DMs *cannot* transmit data to other DMs and the same applies to TMs, except in certain cases where it is convenient to transfer the total responsibility of a Transaction from one TM to another (i.e. if a Transaction runs as a local Transaction on another computer.)

Distributed Schemata

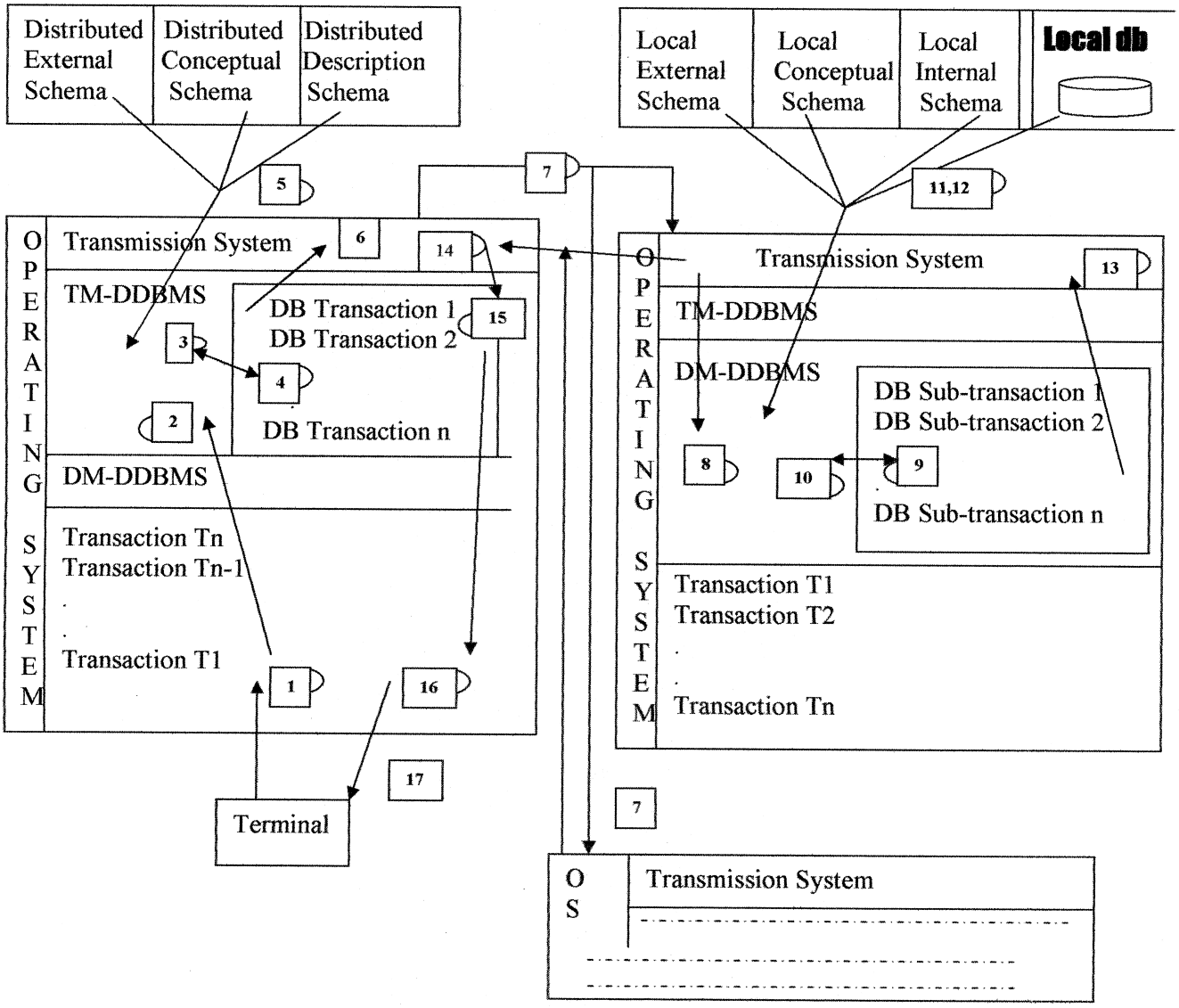
- **Distributed External Schema** contains external schemata for the programs that can run from the local TM.
- **Distributed Conceptual Schema** contains that section of the **Total Conceptual Schema** which is relevant to the Transaction that can be run on the local TM.
- **The Distributed Description:** Describes how the logical files are *divided* (fragmented) into *horizontal or vertical* subdivisions and ***where*** these subdivisions are stored. If the db contains Replication subdivisions so that it is possible to optimize query Transactions and get the redundant data updated automatically.
- **Local External Schema** contains external schemata for the sub-transactions that can be processed on the local db.
- **Local Conceptual Schema** contains the conceptual schema for the local db - the Total Conceptual db can be defined as the union of all the Local Conceptual dbs.
- **Local Internal Schema** contains an internal schema for the local db.

Multithreaded Distributed DB Management Systems

Rely on an OS which can run several different tasks at the same time, on multiple processors, for parallel processing.

- The **TM-DDBMS** can run several different db statements at the same time, as discussed in **parallel OSs**. The DDBMS divides the individual statements into ***sub-transactions*** and transmits these to the DMs that can process them.
- The **DM-DDBMS** can run several different Transactions transmitted from the different TMs at the same time.

This is shown in the following diagram:



1. A transaction arrives at a local site and an application program is started with the data of the Transaction as input.
2. If the application program has to execute a db statement, this is transmitted to the TM-DDBMS manager system, which can run db statements from all application programs at the same time.

3,4. Working areas are created to run the db statements received.

5. The different **TM** dictionary descriptions are read into memory of the site.
6. The db statement is divided into sub-transactions corresponding to the **DM** machines controlling the data involved.
7. The sub-transactions are transmitted to the relevant distributed site.
8. **DM**-DDBMS receives a sub-transaction from the **TM**.

9,10. Working areas are created to run sub-transactions received.

11. The local schemata are read.
12. Data from the physical db is input into the buffer in DM.
13. An External View from the Local db is transferred to the Transmission System.
14. The External View from the Local db arrives at the TM that managed the original db query.
15. The responses of the original sub-transactions from the focal DMs are corrected for a total response.
16. The result of the db statement is transmitted to .the application program that originally sent the query.
17. Back to the terminal.

LAN, WAN, SAN

A homogeneous DDBMS, where all DBMSs are of the same kind, may be implemented in a Local Area Network (LAN) or in a Wide Area Network (WAN). This depends on the type of application and its solution.

Some examples are:

DDBMS Type	Network Type	
	LAN	**WAN**
Homogeneous	Data Management and Financial Applications	Travel Management and Financial Applications
Heterogeneous	Inter-Divisional	Integrated Systems (Banking)
	Information Systems	Inter-Divisional Systems (inter-banking)

SAN: Storage Area Networks - Storage Architecture

A **SAN** is a world-class enterprise system on **on-line** and **near-line.** With highly available Hardware in combination with *Storage Management Software:*
Characteristics: Precise level of Data availability - Scalability - Control.

Remote replicating Technologies

Categories:

1. **Local Replication.** (Within Data Center): Fully synchronous block level copies from *disk array to disk array.*
2. **Metropolitan Replication.** (Up to 100 km): Synchronous **block level** copies are still possible given the availability of **dark fibre** connectivity.
3. **Long Distance.** (100 km+): Physical limitations make fully *synchronous* replication virtually impossible. (solution: requires an *asynchronous* batched update process).

Asynchronous update process: Ensures data integrity: If a batch update arrives corrupt or out of order, then there is a possibility of damaging the remote replica of the db. Archive logs are employed to copy Transaction data from the primary site to the replicated db.

- **Reliability:** This process ensures that the remote db will always be reliable.
- **Consistency:** Identical point-in-time copy of the source db; as well as, advanced error checking operations for each step of the replication process.
- **Flexibility:** Allows the rate of synchronization to be adjusted depending on the requirements of the system. This is usually done with shell scripts and SQL (Same as in Mastoras' CPS510).

Purpose of SAN

- **Disaster Recovery:** the site to site updates could be scheduled quite frequently (every 5 mins?) so that the data at the recovery site is as up-to-date as possible.
- **Data Warehousing:** usually are scheduled as nightly batch (or day) depending on the system requirements.
- **Data Center migration** aspects of both of the above: Copies may be in longer intervals initially, but later may be more frequent at cut over dates, in order to minimize the overall downtime.

RECOVERY CONTROL in DDBMS

We have seen the **TM**, **TC**, and **DM**. Also, a T is a program unit whose execution preserves the consistency of the db so a T must be executed *atomically* (i.e. - either all of the T or none of the T is executed - indivisible).

In the DDBMS all sites are participating in the execution of a T (even as sub-Transactions). So the failure of one of the sites or the failure of a communication link connecting these sites may result in erroneous computations!!

The **TM** must ensure that the Ts are executing properly:

- The site **TM** must cooperate to execute global Transactions
- In order that they cooperate, we define a **Transaction Coordinator** (**TC**) whose job is to coordinate execution of all Ts (local or global) initiated at the site (No need for a TC on a centralized environment)

The **TM** at each site is responsible for:

- Maintaining a log for recovery purposes.
- Participating in an appropriate **concurrency** control scheme to coordinate the concurrent execution of the Ts executing at that site.

The **TC** at each site is responsible for:

- Starting the execution of a T.
- Breaking the T into a number of sub-Ts and distributing the sub-Ts to the appropriate site for execution.
- Coordinating the termination of the Ts which may result in the T being **committed or aborted** *at all sites!*

NOTE: if no **TC** is defined the **TM** takes these responsibilities.

Distributed Atomic Transactions

Because the Ts are indivisible. the TC must execute a **commit protocol.**
We have: 2-phase and 3-phase commit protocols.

Two Phase Commit Protocol with TC

Suppose T is initiated at site Si with TC as Ci When T completes its execution, i.e., all sites at which T has executed, inform Ci that T has completed; then the TC or Ci starts the 2-Phase Commit:

Phase 1:

- Ci adds the record **<prepare T>** to the log and forces it into stable storage.
- It then sends a **prepare T** message to all sites at which T executed.
- When the TM at that site receives that message it determines whether it is willing to commit its portion of the T or not:

  * If the answer is **no**: adds the statement **<no T>** to the log and then it responds by sending an **abort T** message to Ci.
  - If the answer is **yes**: adds the statement **<ready T>** to the log and forces all log records corresponding to T onto stable storage. The TM replies with a **ready T** message to Ci.

Phase 2:

- When Ci receives the response to prepare T message from all of the sites or when a pre-specified time interval has elapsed since the prepare T message was sent out., Ci can determine whether the T can be committed or aborted:
  - T can be committed if Ci receives a ready T message from all participants.
  - Otherwise T must be aborted.
- So Ci either a statement <commit T> or <abort T> is added in the log.
- Now the T has either committed or aborted!! So the coordinator sends either a commit T or an abort T message to all participating sites.
- When a site receives the message, it records it to the log.

NOTES:

1) T can be aborted at any time prior to sending the **ready T** message to the TC.
2) **3-phase commit:** In some implementations, a site sends an **acknowledge T** message to TC at the end of the second phase protocol. When the TC receives this message from all the sites, it adds the statement **<complete T>** to the log.
3) If only a TM and DM are involved:
   i. First the TM transmits a "**ready to commit**" message to the DMs involved in the T.
   ii. When the TM has received an "**acceptance to commit**" message from *every* DMs involved, the TM transmits a "**commit**" statement.
   iii. If the TM does not receive the "**acceptance to commit** " message from every DM involved in the T, T has failed and must be recovered by backing out.
   iv. If recovery is to occur **after** the **commit** statement has been transmitted, the updating must be forced through, possibly by means of the TM log.

Types of Failures

1. **Failure of a participating site:**
   - If and when a participating site Si recovers from a failure it must examine its log to determine what happened to T:
     - If the log has a **<commit T>** record the site executes a **redo(T)**.
     - If the log has an **<abort T>** record the site executes an **undo(T)**.
     - If the log has a **<ready T>** record the site consults Ck to see what has happened to T.
     - If Ck is up, then notifies Si and whether T *committed* (site executes a **redo(T)** or *aborted* (site executes an **undo(T)**).
     - If none of the above then it implies that the failure *happened before* the **<prepareT >** message, i.e. Si failed before responding to **prepare T** message from Ci. **<abort t>**, in which case site executes an **undo(T)**.
2. **Failure of the Coordinator**
3. **Failure of a link**
4. **Failure of a network partition** where the TC remains in one partition (no problem) or in several partitions: messages are lost.

Distributed Concurrency Control

Concurrency on a distributed system is maintained by:

1. Locking Protocol: Two Phase Locking:
    a) No replicated data
    b) Single Coordinator
    c) Multiple Coordinator
2. Time Stamping order
    a) Centralized (single site)
    b) Distributed
3. Deadlock handling and prevention.

1. Two Phase Locking

a. No replicated data

Each site S maintains a **Local Lock Manager** (**LM**): The LM manages the lock and unlock procedures (shared and exclusive) and data items at that site.

- When a T wants to lock a DI at site Si, it simply sends a message to the Si's LM requesting a lock (SL or XL).
- If DI is locked in an incompatible mode, then the request is delayed until it can be granted.
- When granted, the LM sends a message to back to the initiator saying the lock request has been granted.

N.B. Simple implementation:

Only *two* message transfers for Lock and *two* messages for unlock requests.

b. Single Coordinator.

The system maintains a **Single Lock Manager** (**SLM**): SLM resides in a *single chosen site* Si. All lock and unlock requests are made to this site Si.

- If T wants to lock DI, it sends a Lock request to Si.
- The SLM determines whether the lock can be granted immediately.
    - If so, it sends a message to the site of the request.
    - If the request is delayed until it can be granted: message to the initiator site Si.
- T can read the DI from **any** one of the sites at which a replica of the DI resides.
- In case of a write, all the sites where a replica of DI exists must be written.

Advantages:

- Simple implementation: requires *two* messages for handling lock requests, and *one* message for unlock requests.
- Simple deadlock handling: Since all lock and unlock requests are made at one site; the deadlock handling algorithm for a single site applies here too.

Disadvantages:

- *Bottleneck:* This site becomes a bottleneck since ***all*** requests are processed here.
- *Vulnerability and Reliability:* If this site Si fails, the Concurrency Controller is lost: Hence, processing must stop or **Recovery Scheme** must be used.

c. Multiple Coordinator.

The system, in this scheme, maintains a Lock Manager, which is distributed, over several sites.

- Reliability is increased
- Bottleneck and Vulnerability disadvantages are reduced!
- However, it complicates deadlock handling, since lock and unlock requests are not made to a single site.

2.Time Stamp Ordering

T is given a unique Timestamp for serialization order purpose. We have two schemes: Centralized and Distributed.

a) Centralized (Single Site) Time Stamp Order

- A single site is designated for distributing the timestamps (TS).
- The site can use a logical counter or its own local clock.

b) Distributed Time Stamp Order

Each site generates a unique timestamp using either a logical counter or a local clock.

Global timestamps must be unique. In order to achieve this, a **global timestamp** is constructed consisting of:

unique local timestamp + site identifier*,* (where '+' denotes concatenation).

NOTE: the order of site identifiers is very important:
We must ensure that TSs generated at one site are not always greater than those generated at another site.

PROBLEM: Some sites generate TSs *faster* than other sites. Hence, we need a fair system to generate a TS.

SOLUTION: A logical counter, which is incremented after a new local TS is generated, advancing whenever a T visits that site Si.

3. Deadlock Handling and Prevention

The tree protocol for centralized systems can be extended to a **Global tree** protocol.

wait-for graphs
Each site keeps a *local wait-for graph:*

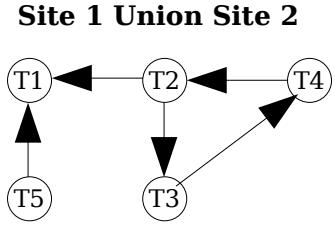
> The nodes of the graph correspond to all Ts (local and global) that are currently either holding or requesting any DI local to that site.

Example
Suppose S1 and S2 are two sites, the wait-for graphs may look like:

NOTE:

- T2 and T3 appear in both sites, this means they are requesting data items from both sites.
- When Ti on S1 needs a resource held by Tj in site S2, a request message is sent by Ti to S2.
- The edge of Ti -> Tj is then inserted in the local *wait-for* graph at S1.
- Each *wait-for* graph is **acyclic** (no cycles).

A *deadlock* may exist in the system because the **union** of S1 and S2 contains a cycle.



There are algorithms to prevent this from happening