REMOTE PROCEDURE CALL (RPC)

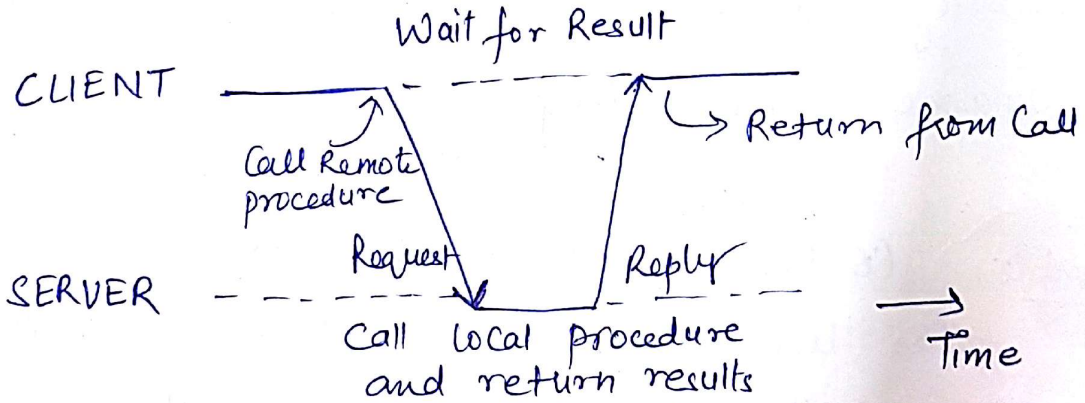
Introduced by Birrell and Nelson (1984)

When a process on machine A calls a procedure on machine B, the calling process on A is suspended and execution of the called procedure takes place on B.

Information can be transported from the caller to the callee in the parameters and can come back in the procedure result. No message passing at all is visible to the programmer.

This method is known as Remote Procedure Call (RPC)

Principle of RPC b/w a client and server Program :



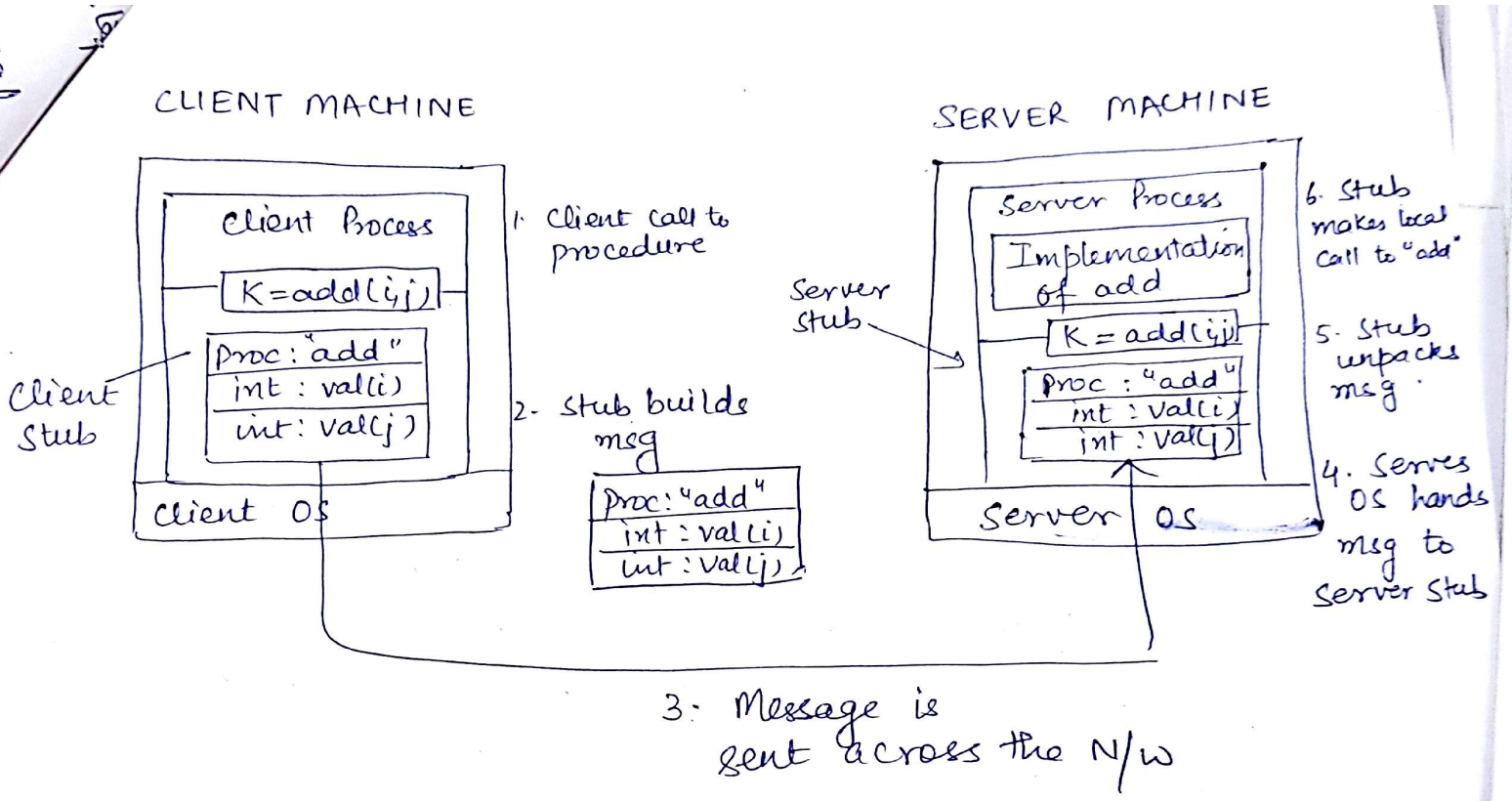
- Following the call to send, the client stub calls receive, blocking itself until the reply comes back.
- The server performs its work & then returns the result to the caller in the usual way.

A Remote Procedure call occurs in the following

1. The client procedure calls the client stub in the normal way.
2. The client stub builds a message & calls the local O.S.
3. The client's OS sends the message to the remote OS
4. The remote OS gives the message to the server stub.
5. The server stub unpacks the parameters & calls the server.
6. The server does the work & returns the result to the stub.
7. The server stub packs it in a message & calls its local OS.
8. The server's O.S. sends the message to the client's OS.
9. The client's OS gives the message to the client stub.
10. The stub unpacks the result & returns to the client

→ Packing parameters into a message is called 'parameter marshaling'.

Example, Consider a remote procedure, add(i, j) that takes two integers parameters i & j and return their arithmetic sum as a result.



Steps Involved in doing a remote Computation through RPC.

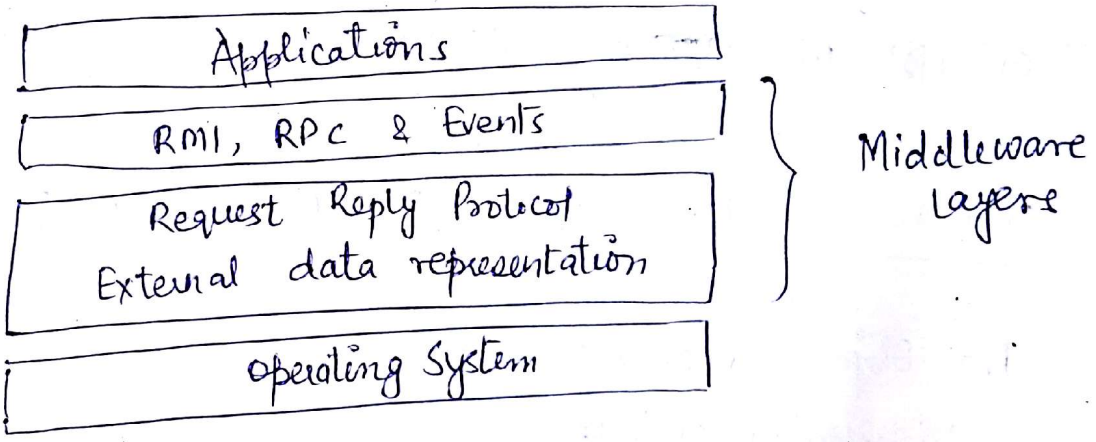
DISTRIBUTED OBJECTS & REMOTE INVOCATION

RMI (Remote Method Invocation) is an extension of local method invocation that allows an object living in one process to invoke the methods of an object living in another process.

S/w that provides a programming model above the basic building blocks of processes & message passing is called middleware.

MIDDLEWARE: uses protocols based on msgs b/w processes to provide its higher-level abstractions such as remote invocations & events.

- The remote method invocation abstraction is based on the 'request-reply' protocol.



Aspects of Middleware:

(1) Location Transparency: In RPC, the client that calls a procedure cannot tell whether the procedure runs in the same process or in a different process. Nor does the client need to know the location of the server. Ily, in RMI the object making the invocation cannot tell whether the object it invokes is local or not & does not need to know its location.

Communication protocols : independent of underlying protocols. eg. the req-reply protocol can be implemented over either UDP or TCP.

Computer H/W : Standards for external data representation. used when marshalling & unmarshalling msgs. They hide the differences due to H/W architectures such as byte ordering.

Operating systems : independent of underlying O.S.

Use of Several Programming languages : Allows clients written in one language to invoke methods in objects that live in server programs written in another language. This is achieved by using an interface definition lang. or IDL to define interfaces.

COMMUNICATION B/W DISTRIBUTED OBJECTS :

- The Object Model &
- Distributed objects
- The Distributed object Model

(1) The Object Model : An object communicates with other objects by invoking their methods, generally passing arguments & receiving results. Objects can encapsulate their data & the code of their methods.

In distributed object system, an object's data should be accessible only via its methods.

OBJECT MODEL

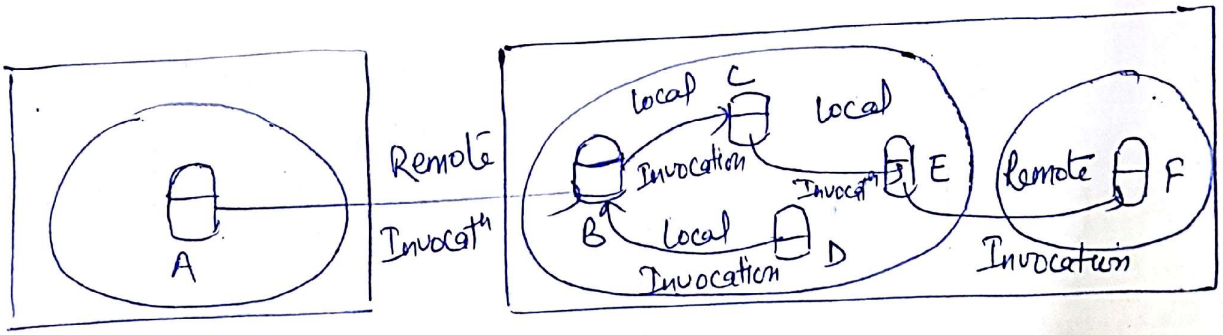
- An object oriented pgm (Java, C++) consists of a collection of interacting objects, each of which consists of a set of data & a set of methods.

- An object can communicates with other objects by invoking their methods, generally passing arguments & receiving results.

- objects can encapsulate their data & the code of their methods.

⇒ In a "distributed object system" an object's data should be accessible only via its methods (or interface).

Distributed object Model



- Each Process contains a collection of objects
- Method Invocation b/w objects in different processes, whether in the same computer or not, are known as "Remote Method Invocation".

- Method invocations b/w objects in the same process are local method Invocation.

REMOTE OBJECTS

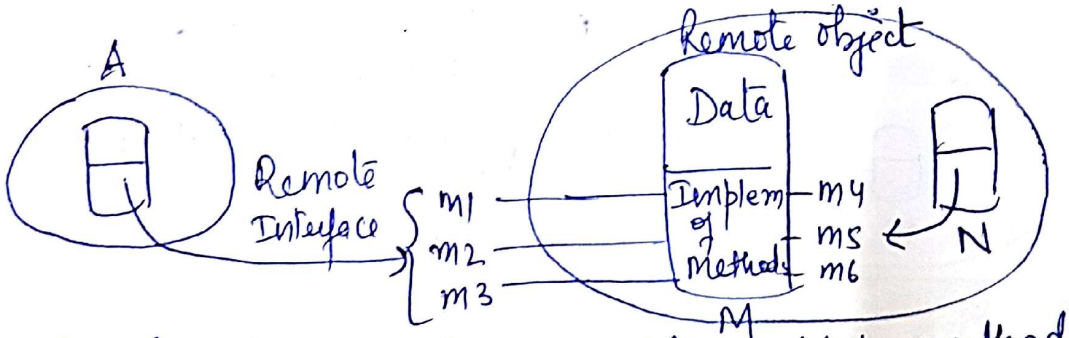
- Objects that can receive remote invocations.

- Objects can invoke the methods of a remote object if they have access to its remote object reference.

- Every remote object has a remote interface that specifies which of its methods can be invoked remotely.

↳ A Remote object Reference is an identifier that can be used throughout a distributed system to refer to a particular unique remote object.

- A remote object reference is passed in the invocation message to specify which object is to be invoked.



- The Remote interface specifies which methods of an object can be invoked remotely.

- objects in other processes can invoke only the methods that belong to the remote interface of a remote object. eg. (A → {m1, m2, m3})

- local objects can invoke the methods in the remote interface as well as other methods implemented by a remote object. (N → {m4, m5, m6})

ELECTION ALGORITHMS

Distributed algorithms require one process to act as coordinator to perform some special role.

ex, the coordinator in the centralized mutual exclusion algorithm.

Assumptions:

1) Each process has a unique number.
2) Every process knows the process number of every other process.
3) What the processes do not know is which ones are currently up & which ones are currently down.

Goal: Ensure that when an election starts, it concludes with all processes agreeing on who the new coordinator is to be.

(I) BULLY ALGORITHM:

When a process notices that the coordinator is no longer responding to requests, it initiates an election.

A process P holds an election as follows:

1. P sends an election msg 'ELECTION' to all processes with higher numbers.
2. If no one responds, P wins the election and becomes coordinator.

3. If higher-ups answers, it takes over. P's job done.

4. If a process can get an ELECTION msg from one of its lower-numbered colleagues.

   (i) message arives → the receiver sends an 'OK' msg back to the sender to indicate that he is alive and will take over.

   (ii) The receiver then holds an election, unless it is already holding one.

5. Eventually, all processes give up but one, & that one is the new coordinator.

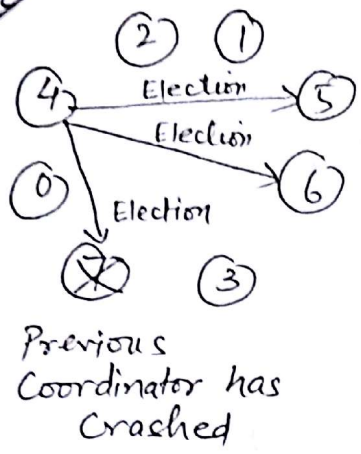
   (i) It announces its victory by sending all processes a msg telling them that starting immediately it is the new coordinator.

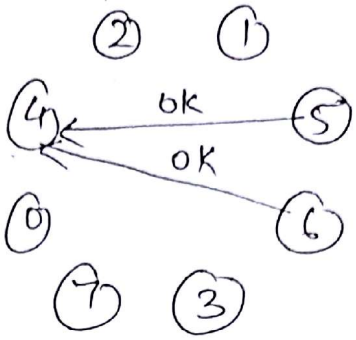
6. If a process that was previously down comes back, it holds an election

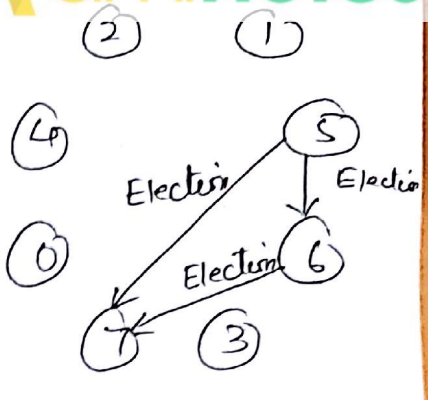
   If it happens to be the highest-numbered process currently running, it will win the election and take over the coordinator's job.

⇒ Thus the biggest guy in town always wins, hence the name "Bully Algorithm".

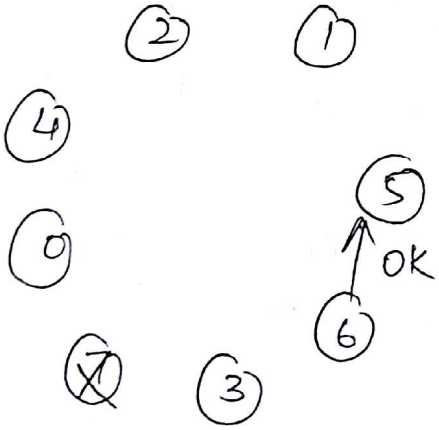
Ex, The group concist of 8 processes.

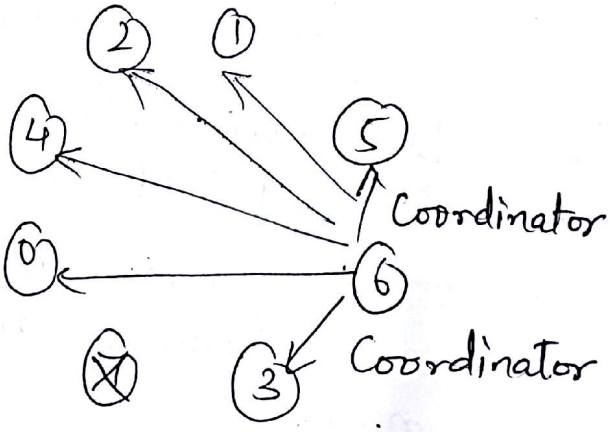






- Previously process 7 was the Coordinator, but it has just crashed
- Process 4 is the first one to notice this, so it sends ELECTION message to all the processes higher than it, namely Processes 5 & 6 both respond with OK.
- Upon getting the first of these responses, 4 knows that its jobs is over.
- Both 5 & 6 hold elections, each one only sending messages to those processes higher than itself.





- Process 6 tells 5 that it will take over.
- 6 knows that 7 is dead & that it is the winner.
- 6 announces this by sending a COORDINATOR message to all running processes.

When 4 gets this, it can now continue with the operation, it was trying to do when it discovered that 7 was dead, but using 6 as the coordinator this time.

If process 7 is ever restarted, it will just send all the others a COORDINATOR message & bully them into submission.

(II) A RING ALGORITHM

- Based on the use of a ring
- Assume: The processes are physically or logically ordered, so that each process knows who its successor is.

1) When any process notices that the coordinator is not functioning, it builds an ELECTION msg containing its own process no. & sends the msg to its successor.

2) If the successor is down, the sender skips over the successor and goes to the next member along the ring.

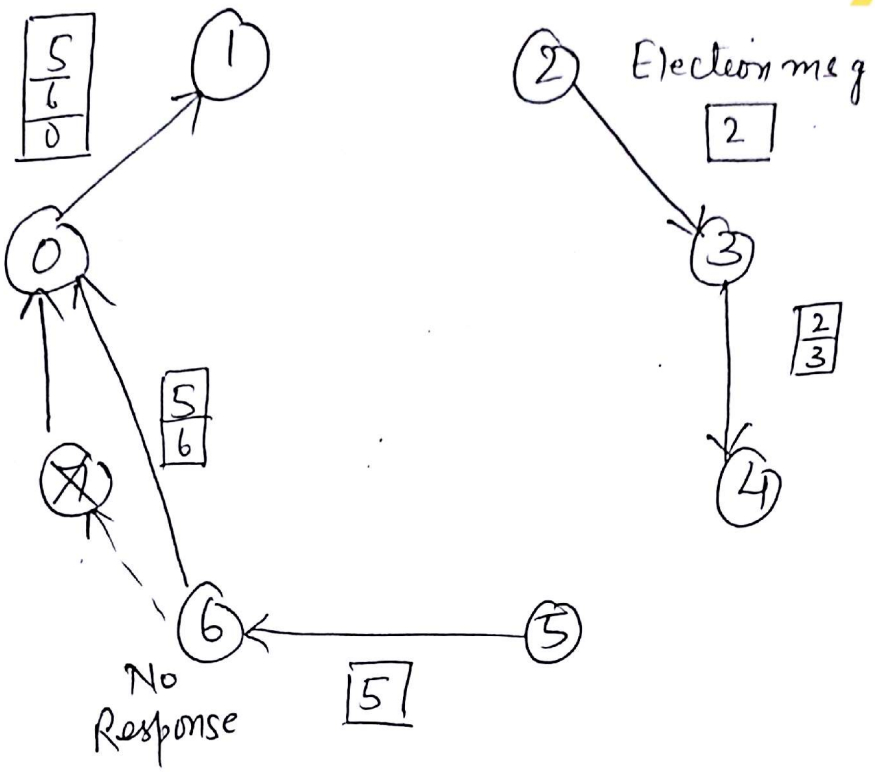
3) At each step, the sender adds its own process no to the list in the message.

4) Eventually, the msg gets backs to the process that started it all.

5) That process recognizes this event when it receives an incoming msg containing its own process number.

6) At that point, the msg type is changed to 'Coordinator' and circulated once again, this time to inform everyone else who the coordinator is (the list member with the highest number) & who the members of the new ring are.

7) When this msg has circulated once, everyone goes back to work.



- Two processes 2 & 5 discover simultaneously that the previous coordinator, process 7 has crashed
- Each of these builds an ELECTION msg and starts circulating it.
- Eventually both msgs will go all the way around, & both 2 & 5 will convert them into COORDINATOR msg with exactly the same members & in the same order.
- When both have gone around again, both will be removed.
- It does ~~not~~ no harm to have extra msgs circulating at most it wastes a little bandwidth.

TRANSACTION

Sequence of operations on objects to be performed as an indivisible unit by the servers managing those objects. Such that all objects remain in a consistent state.

example, BANKING

- withdrawl
- deposit

ACID Properties of Transactions

(1) Atomicity: A transaction either completes successfully or has no effect at all.

(2) Consistency: A transaction takes the system from one consistent state to another consistent state.

(3) Isolation: Each transaction must be performed w/o interference from other transactions.

(4) Durability: Once a tx commits, the results are saved in permanent storage.

Distributed Transactions: a transact^n that accesses objects managed by multiple servers.

(1) Flat transaction: operations are invoked sequentially.

eg. Begin_transaction
reserve a seat for flight;
reserve a hotel room;
reserve a cab;
End_transaction

(2) Nested Transaction: The top-level transaction can start subtransactions & each subtransaction can start further subtransactions down to any depth of nesting.

eg.

```
Begin_transaction
    Begin_subtransaction
        reserve a seat for flight;
    end_subtransaction
    Begin_subtransaction
        reserve a hotel room;
        -
        -
End_transaction
```

Advantages of Nested tx:

(1) Subtx at the same level can run concurrently

(2) Subtx can commit or abort independently.
- with a flat tx one tx failure would cause the whole tx to be aborted.

LOCKS
- Exclusive → Write-locked item
- Shared → read-locked item

2-Phase Locking scheme

- Growing Phase
- Shrinking Phase

A Transaction is said to follow the 2-Phase locking protocol if all locking operations (read_lock, write_lock) precede the first unlock operations in the tx.

2e Phase

An Expanding or growing phase during which new locks on items can be acquired but none can be released &

A shrinking phase during which existing locks can be released but no new locks can be acquired.

LOCK COMPATIBILITY

LOCK ALREADY SET	LOCK REQUESTED	
	READ	WRITE
NONE	OK	OK
READ	OK	WAIT
WRITE	WAIT	WAIT

Serial Schedule : No interleaving of op^{n}s is permitted

(1) Execute all the op^n of T_X T_1 (in seq) foll. by the op^n of T_X T_2 (in seq).

(2) Execute all the op^n of T_X T_2 (in seq) foll. by all the op^n of T_X T_1 (in seq).

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);

(i) Serial Schedule A

T_1	T_2
	read_item(X);
	X = X + M;
	Write_item(X);
read_item(X);	
X = X - N	
Write_item(X);	
read_item(Y);	
Y = Y + N;	
Write_item(Y);	

(ii) Serial Schedule B

Serializable : which is equivalent to some serial schedule.

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);	

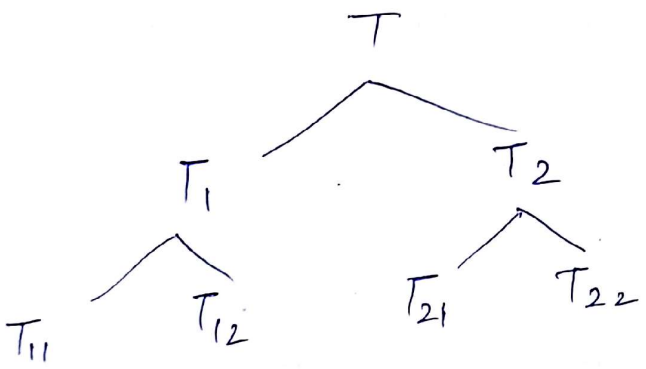
Schedule C

T_1	T_2
read_item(X);	
X = X - N;	
Write_item(X);	
	read_item(X);
	X = X + M;
	Write_item(X);
read_item(Y);	
Y = Y + N;	
Write_item(Y);	

Schedule D SERIALIZABLE SCHEDULE

NESTED TRANSACTIONS

- Transactions composed of other transactions.
- The outermost tx in a set of nested tx is called the top-level tx.
  Transactions other than the top-level tx are called subtransactions.



A subtransaction appears atomic to its parents with respect to transaction failures & to concurrent access.
Subtransaction at same level, such as T_1 & T_2 can run concurrently but their access to common objects is serialized, for ex by locking scheme.

Advantages :

(1) Subtransactions at one level (& their descendants) may run concurrently with other subtransactions at the same level in the hierarchy.
(2) Subtransactions can commit or abort independently.

The Rules of committing of Nested tx :-

(1) A tx may commit or abort only after its child tx have completed.
(2) When a ~~tx~~ subtx completes, it makes an independent decision either to commit provisionally or to abort.

Its decision to abort is final.

(3) When a parent aborts, all of its subtx are aborted.

(4) When a subtx aborts, the parent can decide whether to abort or not.

(5) If the top-level tx commits, then all of the subtx that have provisionally committed can commit too, provided that none of their ancestors has aborted.

TWO-PHASE LOCKING

STRICT TWO-PHASE LOCKING : A Transaction that needs to read or write an object must be delayed until other tx that wrote the same object have committed or aborted.

Use of locks in strict 2-phase locking :-

1. When an opⁿ accesses an object within a tx :
   (a) If the object is not already locked, it is locked & the opⁿ proceeds.
   (b) If the object has a conflicting (write) lock set by another tx, the tx must wait until it is unlocked.
   (c) If the object has a non-conflicting (read) lock set by another tx, the lock is shared & the opⁿ proceeds.
   (d) If the object has already been locked in the same tx, the lock will be promoted if necessary & opⁿ proceeds.

When a tx is committed or aborted, the server unlocks all objects it locked for the tx.

(1) One-phase atomic commit protocol:

The Atomicity of transactions requires that when a distributed transaction comes to an end, either all of its operations are carried out or none of them.

- In the case of a distributed tx, the client has requested the opⁿ at more than one server.

A tx comes to an end when the client requests that a tx be committed or aborted. A simple way to complete the tx in an atomic manner is for the coordinator to communicate the commit or abort request to all of the participants in the tx & to keep on repeating the request until all of them have acknowledged.

(2) But the Two-phase commit protocol is designed to allow any participant to abort its part of a tx. Due to the requirement of atomicity, if one part of a tx is aborted, then the whole tx must also be aborted.

In the first phase, each participant votes for the tx to be committed or aborted. Once a participant has voted to commit, it is not allowed to abort it.

In the second phase, every participant in the tx carries out the joint decision. If any one participant votes to abort, then the decision must be to abort the tx.

⇒ The problem is to ensure that all of the participants vote & that they all reach the same decision.

RF

Operations for Two-phase commit protocol

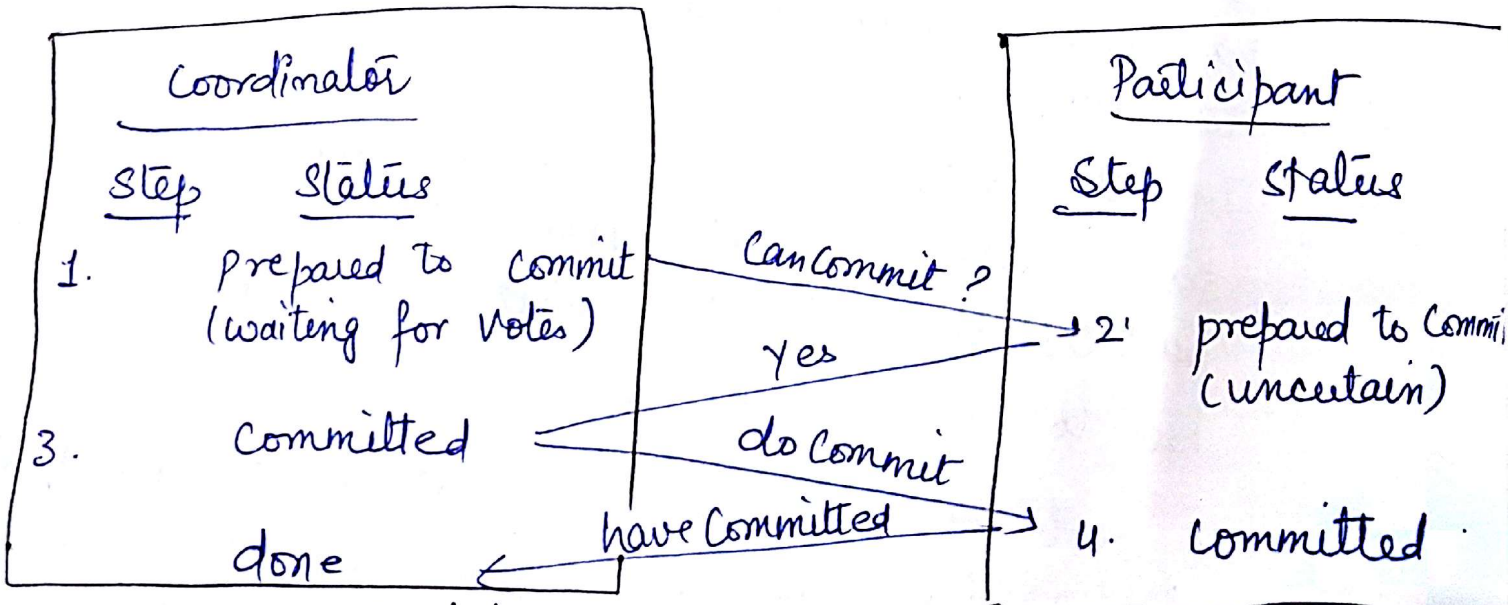
① CanCommit? (trans) → Yes/No
Call from coordinator to participant to ask whether it can commit a tx. Participant replies with its vote.

② doCommit (trans)
Call from coordinator to participant to tell participant to commit its part of a tx.

③ doAbort (trans)
_____ to abort its part of a tx.

④ haveCommitted (trans, participant)
Call from participant to coordinator to confirm that it has committed the tx.

⑤ getDecision (trans) → Yes / No
call from participant to coordinator to ask for the decision on a tx after it has voted Yes but has still had no reply after some delay.
Used to recover from server crash or delayed messages.



Communication in 2-phase Commit Protocol.

TIMESTAMP ORDERING

In concurrency control schemes based on timestamp ordering, each op^n in a tx is validated when it is carried out. If the op^n is cannot be validated, the tx is aborted immediately & can be restarted by the client.

- Each tx is assigned a unique timestamp value when it starts.
- Requests from tx can be totally ordered acc. to their TS.

The basic TS ordering rule is based on op^n conflicts :-

A tx's request to write an object is valid only if that object was last read & written by earlier tx.
A tx's req. to read an object is valid only if that object was last written by an earlier tx.

Operation Conflicts for TS ordering

Rule	T_c	T_i	
1.	Write	Read	T_c must not write an object that has been read by any T_i where $T_i > T_c$ this requires that $T_c \geq$ max. read TS of the object.
2.	Write	Write	T_c must not write an object that has been written by any T_i where $T_i > T_c$ this requires that $T_c >$ write TS of the committed object.

	T_c	T_i	
3.	Read ~~Red~~	Write	T_c must not read an object that has been written by an T_i where $T_i > T_c$ this requires that $T_c >$ Write TS of the Committed object.

Timestamp Ordering Write Rule :

a) T_3 write

Before T_2

After T_2 T_3

Time

b) T_3 write

Before T_1 T_2

After T_1 T_2 T_3

Time

c) T_3 write

Before T_1 T_4

After T_1 T_3 T_4

Time

d) T_3 write

Before T_4

After T_4

Time

Transcation Aborts

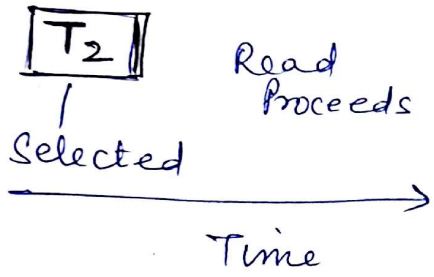
T_3 write TS on the committed version of the object & the tx is aborted.

T_i → Committed

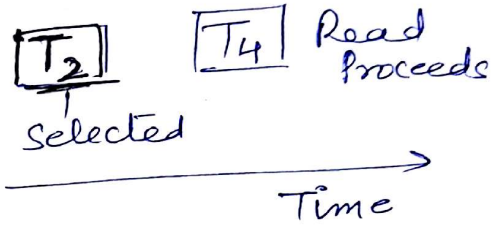
T_i → Tentative

Timestamp ordering Read Rule:

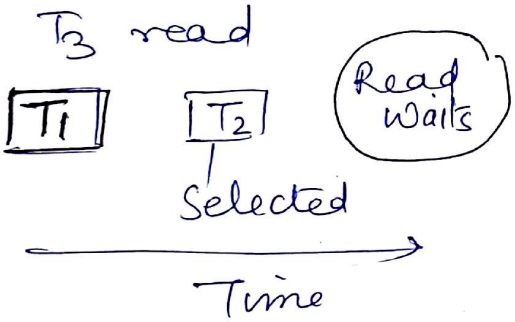
a) T_3 read



b) T_3 read



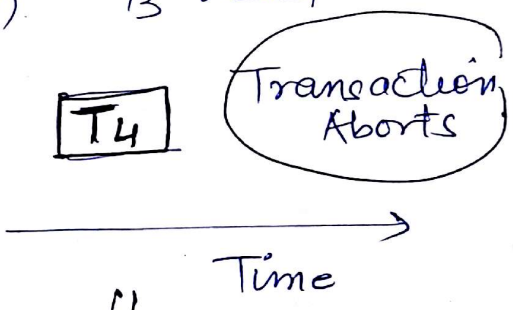
c) T_3 read



⇓

T_3 waits until T_2 is committed or abort.

d) T_3 read



⇓

There is no suitable version to read & tx T_3 is aborted

SECURITY

Security mechanism in DS arises from the desire to share resources.

1) Processes encapsulate resources & allow clients to access them through interfaces. Resources must be protected against unauthorised access.

2) Processes interact through a n/w i.e. shared among many users. Enemies can access the n/w. They can copy as well as read any msg.

Overview of SECURITY TECHNIQUES:

Worst Case Assumptions & Design Guidelines

(1) Interfaces are exposed (communicat" interfaces)

(2) Networks are insecure (Msg sources can be falsified)

(3) Limit the lifetime & scope of each secret (The longer we use the secret key & more widely it is know the greater the risk).

(4) Algorithms & Program code are available to attackers (secret encryption algorithms are totally inadequate for today's large scale n/w env.)

(5) Attackers may have access to large resources

(6) Minimize the trusted base

CRYPTOGRAPHY

Encryption is the process of encoding a message in such a way as to hide its contents. Encryption & Decryption are also based on the use of secrets called 'keys'.

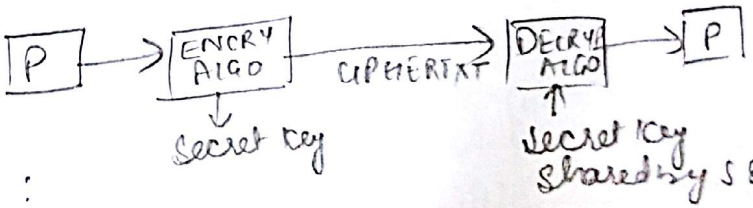
A plaintext is an intelligible msg i.e. to be converted into an unintelligible (encrypted) form.
A ciphertext is a message in encrypted form.

Encryption, the process of converting a plaintext to a ciphertext, generally involves a set of transformation that uses a set of algo. & a set of I/P parameters.

Decryption is the process of converting a ciphertext to a plaintext.

Generally, both encryption & decryption require a 'key' parameter whose secrecy is absolutely essential to the functioning of the entire process.

If the key is the same for both encryption & decryption the system is referred to as the symmetric. otherwise it is asymmetric



POTENTIAL THREATS :

(1) Leakage : the acquisition of inf^m by unauthorized recipients.

(2) Tampering : the unauthorized alteration of information

(3) Vandalism : interference with the proper operation of a sys w/o gain to the perpetrator

(5) Eavesdropping : obtaining copies of msgs w/o authority

(5) Masquerading : sending or receiving msgs using the identity of another principal w/o their authority.

(6) Message Tampering : Intercepting msgs & altering their contents

(7) Replaying : Storing intercepted msgs & sending them at a later date.

(8) Denial of Service : flooding a channel or other resource with msgs in order to deny access for others.

There are 2 main classes of Encryption algo :-

Symmetric (1) The first uses 'shared ~~&~~ secret keys' — the sender & the recipient must share a knowledge of the key & it must not be revealed to anyone else.

Asymmetric (2) uses public / private key pairs - the sender of a msg uses a public key - one that has already been published by the recipient to encrypt the message.
The recipient uses a corresponding private key to decrypt the message.

CRYPTOGRAPHIC ALGORITHMS

- SYMMETRIC (Secret Key)
- ASYMMETRIC (Public Key)

Public key Cryptography Algo

RSA (Rivest, Shamir, Adi)

key Generation

(1) Select p & q such that p & q both prime p = 7 q = 11

(2) Calculate $n = p \times q$ $n = 77$

(3) Calculate $\phi(n) = (p-1)(q-1)$ $\phi(n) = 6 \times 10 = 60$

(4) Select integer e such that $\gcd(\phi(n), e) = 1$; $1 < e < \phi(n)$ $2 \times 5 \times 3 = 20$ e = 13

(5) Calculate d $d = e^{-1} \bmod \phi(n)$ $d \times e \bmod \phi = 1$, $d \times 13 \bmod 60 = 1$

(6) Public key $KU = \{e, n\}$

(7) Private key $KR = \{d, n\}$

Encryption

Plaintxt : M

Ciphertxt : $C = M^e \pmod{n}$

Decryption

Ciphertxt C

Plaintxt $M = C^d \pmod{n}$

Digital Signatures

- emulates the role of conventional signatures, verifying to a third party that a message or a document is an unaltered copy of one produced by the signer.

→ based upon an irreversible binding to the message or document of a secret known only to the signer.

This can be achieved by encrypting the message - or better, a compressed form of the message called a digest, using a key i.e. known only to the signer.

A digest is a fixed-length value computed by applying a secure digest function.

Alice wants to sign a document M so that any subsequent recipient can verify that she is the originator of it. Thus when Bob later accesses the signed document after receiving it by any route & from any source he can verify that Alice is the originator.

(1) Alice computes a fixed-length digest of the document Digest(M).

(2) Alice encrypts the digest in her private key, appends it to M & makes the result M, $\{Digest(M)\}K_{Apriv}$ available to the intended users.

(3) Bob obtains the signed document, extracts M & computes Digest(M).

(4) Bob decrypts $\{Digest(M)\}K_{Apriv}$ using Alice's public key K_{Apub} & compares the result with his calculated Digest(M). If they match, the signature is valid.

Remote Method Invocation

RMI stands for **Remote Method Invocation**. It is a mechanism that allows an object residing in one system (JVM) to access/invoke an object running on another JVM.

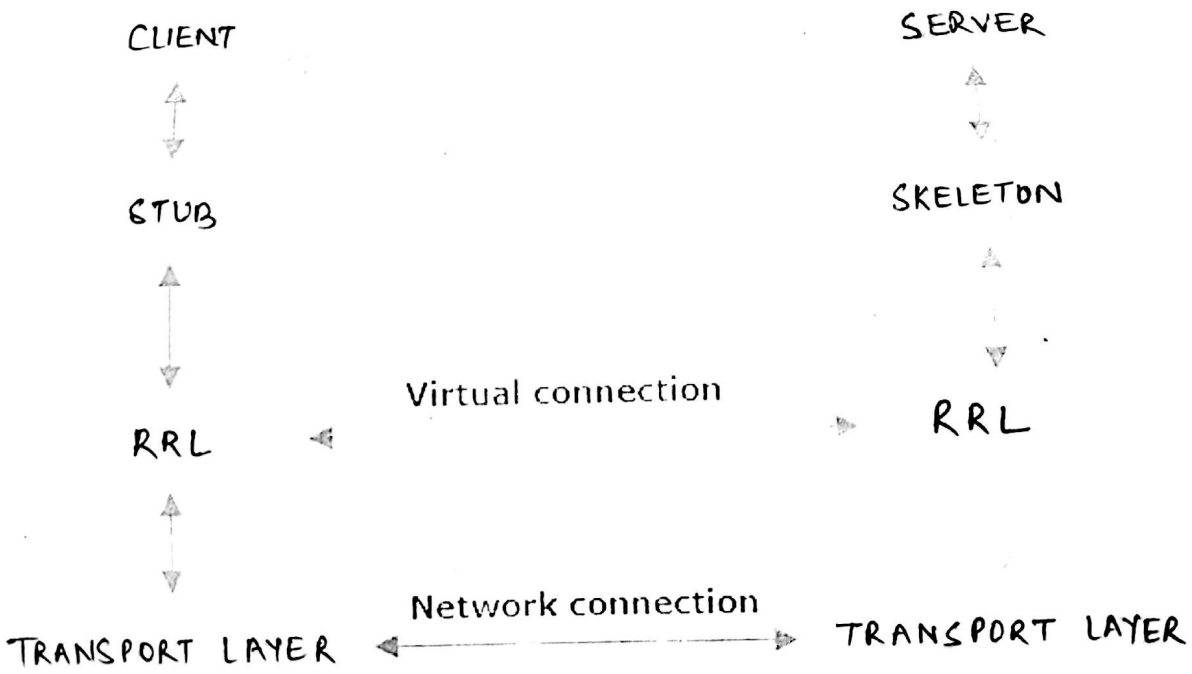
RMI is used to build distributed applications; it provides remote communication between Java programs. It is provided in the package **java.rmi**.

Architecture of an RMI Application

In an RMI application, we write two programs, a **server program** (resides on the server) and a **client program** (resides on the client).

- Inside the server program, a remote object is created and reference of that object is made available for the client (using the registry).
- The client program requests the remote objects on the server and tries to invoke its methods.

The following diagram shows the architecture of an RMI application.



Let us now discuss the components of this architecture.

- **Transport Layer** – This layer connects the client and the server. It manages the existing connection and also sets up new connections.
- **Stub** – A stub is a representation (proxy) of the remote object at client. It resides in the client system; it acts as a gateway for the client program.
- **Skeleton** – This is the object which resides on the server side. **stub**communicates with this skeleton to pass request to the remote object.
- **RRL(Remote Reference Layer)** – It is the layer which manages the references made by the client to the remote object.

stub

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
3. It waits for the result
4. It reads (unmarshals) the return value or exception, and
5. It finally, returns the value to the caller.

skeleton

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

1. It reads the parameter for the remote method
2. It invokes the method on the actual remote object, and
3. It writes and transmits (marshals) the result to the caller.

Working of an RMI Application

The following points summarize how an RMI application works –

- When the client makes a call to the remote object, it is received by the stub which eventually passes this request to the RRL.

- When the client-side RRL receives the request, it invokes a method called **invoke()** of the object **remoteRef**. It passes the request to the RRL on the server side.
- The RRL on the server side passes the request to the Skeleton (proxy on the server) which finally invokes the required object on the server.
- The result is passed all the way back to the client.

Marshalling and Unmarshalling

Whenever a client invokes a method that accepts parameters on a remote object, the parameters are bundled into a message before being sent over the network. These parameters may be of primitive type or objects. In case of primitive type, the parameters are put together and a header is attached to it. In case the parameters are objects, then they are serialized. This process is known as **marshalling**.

At the server side, the packed parameters are unbundled and then the required method is invoked. This process is known as **unmarshalling**.

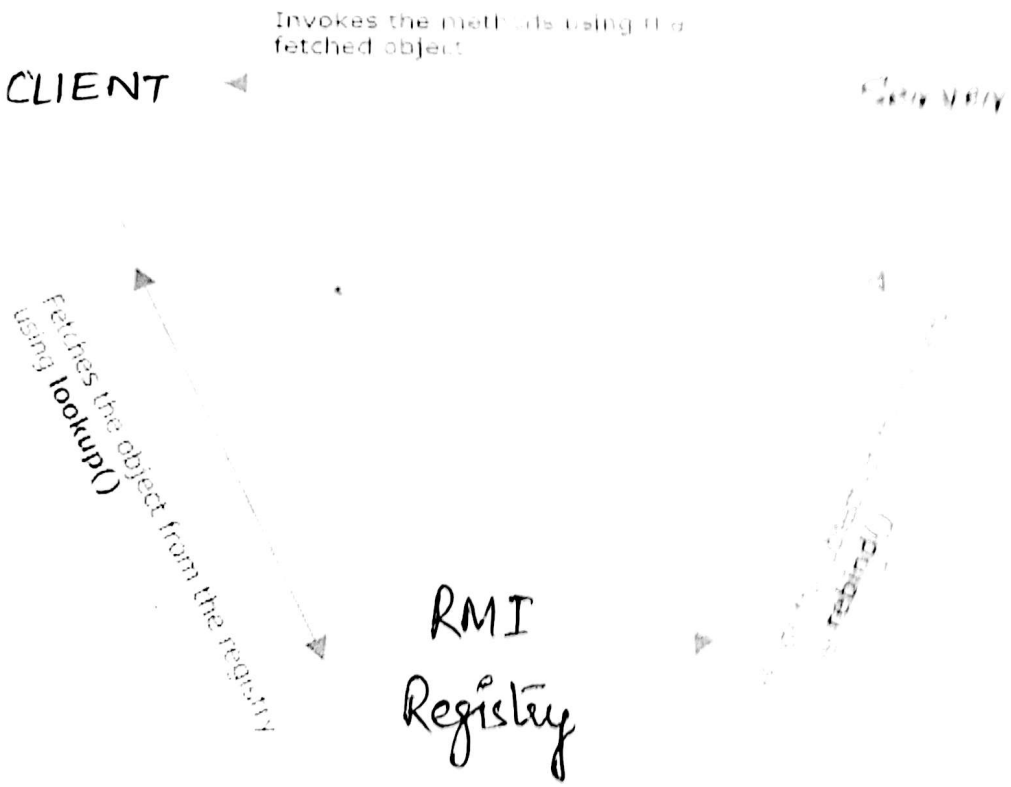
In few words, "**marshalling**" refers to the process of converting the data or the objects inbto a byte-stream, and "**unmarshalling**" is the reverse process of converting the byte-stream beack to their original data or object. The conversion is achieved through "serialization".

RMI Registry

RMI registry is a namespace on which all server objects are placed. Each time the server creates an object, it registers this object with the RMIregistry (using **bind()** or **reBind()** methods). These are registered using a unique name known as **bind name.**

To invoke a remote object, the client needs a reference of that object. At that time, the client fetches the object from the registry using its bind name (using **lookup()** method).

The following illustration explains the entire process –



Goals of RMI

Following are the goals of RMI –

- *To minimize the complexity of the application.*
- *To preserve type safety.*
- *Distributed garbage collection.*
- *Minimize the difference between working with local and remote objects.*

- Define the remote interface
- Develop the implementation class (remote object)
- Develop the server program
- Develop the client program
- Compile the application
- Execute the application

DISTRIBUTED FILE SYSTEMS

A distributed file system is a resource management component of a distributed operating system.

It implements a common file system that can be shared by all the autonomous computers in the system.

2 imp. Goals of a DFS follow:

(1) Network Transparency: To provide the same functional capabilities to access files distributed over a n/w as the file system of a timesharing mainframe system does to access files residing at one location.

→ Ideally users do not have to be aware of the location of files to access them.
This property of a DFS is known as N/w transparency.

(2) High Availability: Users should have the same easy access to files, irrespective of their physical location.

→ System failures or regularly scheduled activities such as backups or maintenance should not result in the unavailability of files.

FILE ATTRIBUTES

File length
Creation Timestamp
Read Timestamp
Write Timestamp
Attribute Timestamp
Reference Count
Owner
File Type
Access control list

FILE SYSTEM MODULES.

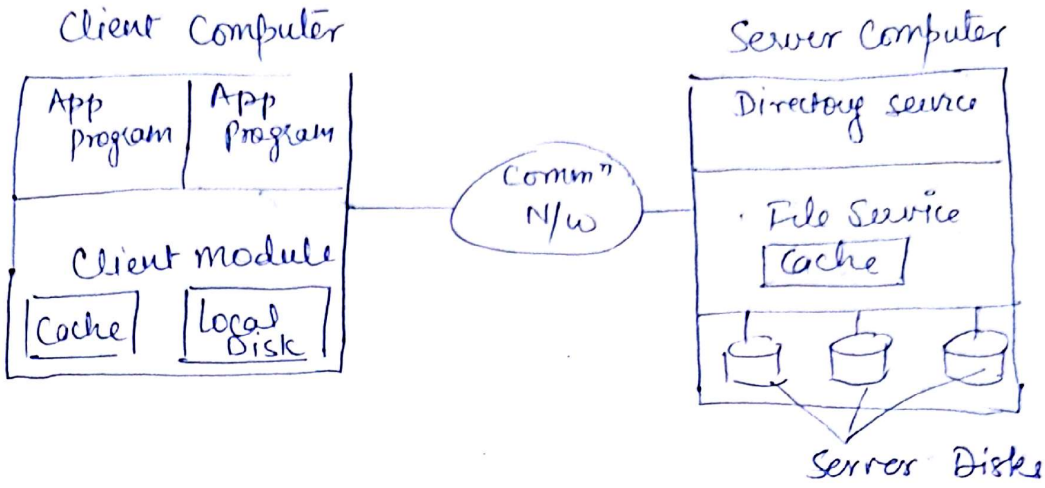
Directory module : relate the names to file IDs.
File module : relates file IDs to particular file
Access Control module : checks permission for operation requested.
File Access Module : reads or writes the data or attributes
Block Module : accesses & allocates disk blocks
Device Module : disk I/O & buffering.

DISTRIBUTED FILE SYSTEM REQUIREMENTS

(1) Transparency
- Access (unaware of distribution of files)
- Location (uniform file name space)
- Mobility (files can be moved without any changes made in admin tables)
- Performance (Even load on service raises — perform satisfaction)
- Scaling (can be expanded with a wide range of loads & n/w sizes)

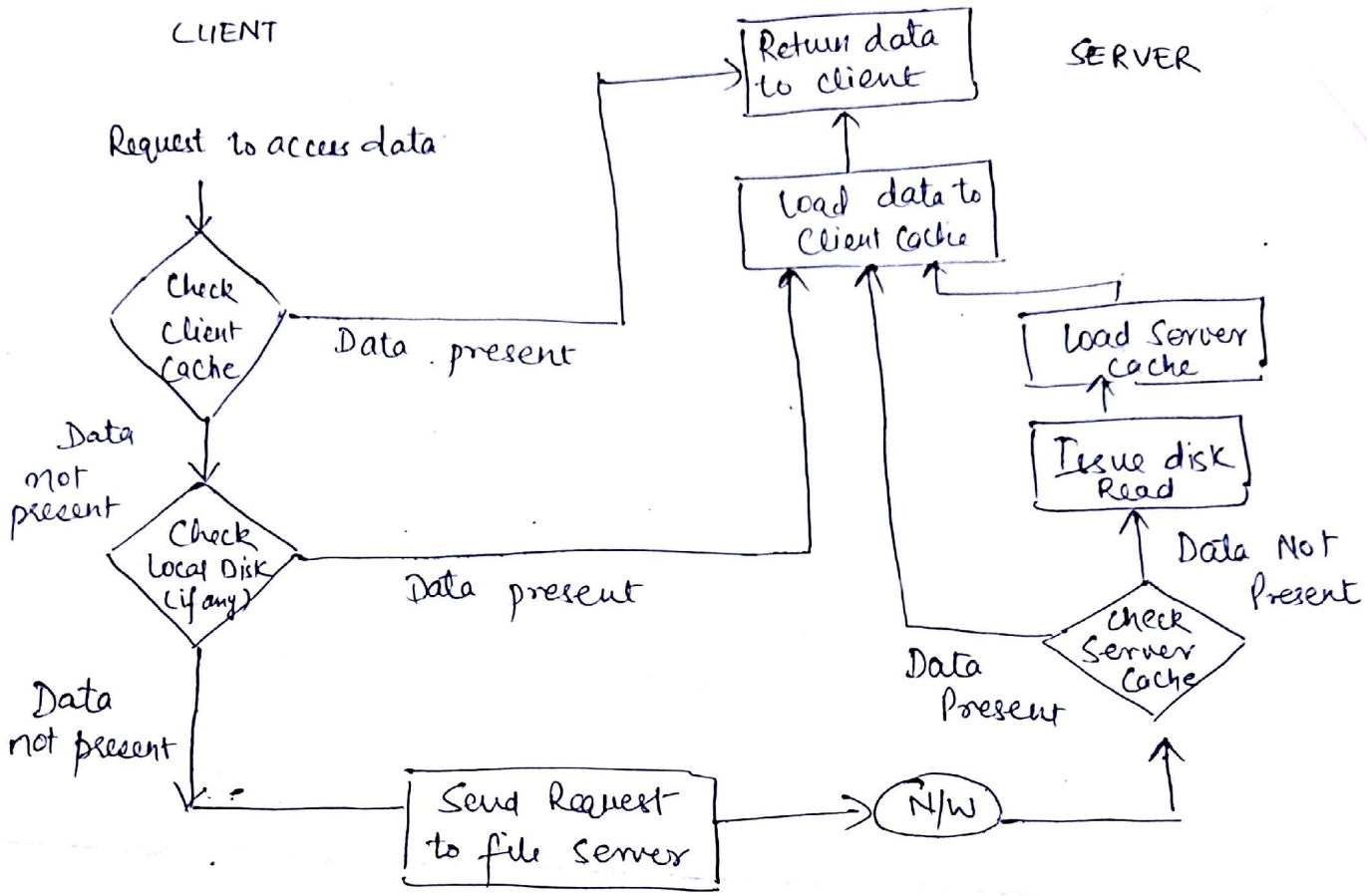
(2) Concurrent File Updates
(3) File Replication
(4) Consistency
(5) Fault tolerance
(6) H/w & OS Heterogeneity
(7) Security (Digital Signatures & Encryption)
(8) Efficiency

ARCHITECTURE



→ In a dist. File sys, files can be stored at any m/c and the computation can be performed at any m/c.

- When a m/c needs to access a file stored on a remote m/c, the remote m/c performs the necessary file op^n & returns data.
- Some client m/c may also be equipped with a local disk storage that can be used for caching remote files.
- The 2 imp. services present in DFS are:
  - 'Name Server'
  - 'Cache Manager'
- A name server is a process that maps names specified by clients to stored objects such as files & directories.
- A cache manager is a process that implements f caching. In file caching, a copy of data stored at a remote file server is brought to the client's m/c when referenced by the client.



Typical Data Access in DFS

DESIGN ISSUES

(1) Naming & Name Resolution :

A name in file system is associated with an Object (such as a file or a directory).

Name resolution refers to the process of mapping a name to an object or (in case of replication) to multiple objects.

A name space is a collection of names which may or may not share an identical resolution mechanism.

Naming Structures

- location Transparency :- file name does not reveal the file's physical storage location.
- location Independence :- File name does not need to be changed when the file's physical storage location changes.

Name Server : Responsible for name resolution i.e. a process that maps names specified by clients to stored objects (files, directories)

(2) Caching : commonly employed in DFS to reduce delays in the accessing of data.

- A copy of data stored at a remote file server is brought to the client when referenced by the client. Subsequent access to the data is performed locally at the client.
- Caching exploits the temporal locality of reference (a file recently accessed is liked to be accessed again in the near future.
- Data can be either cached in the main m/m or on the local disk.

Write Policy:

(1) Write through : The request made by the client are carried to the server immediately.

(2) Write Back : This policy is concerned with the delay of writing server when the data is modified & the modification is reflected to server after time delay.