

Introduction

The traditional system notation of a process that executes a single activity was found in 1980's to be unequal to the requirement of distributed systems - and also to those of more sophisticated single-computer applications that require internal concurrency. The problem as we shall show, is that the traditional process makes sharing between related activities awkward and expensive.



2.1 Process and threads

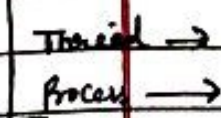
The solution reached was to enhance the notion of process so that it could be associated with multiple activities.

A thread is the operating system abstraction of an activity.

An execution environment consists of:

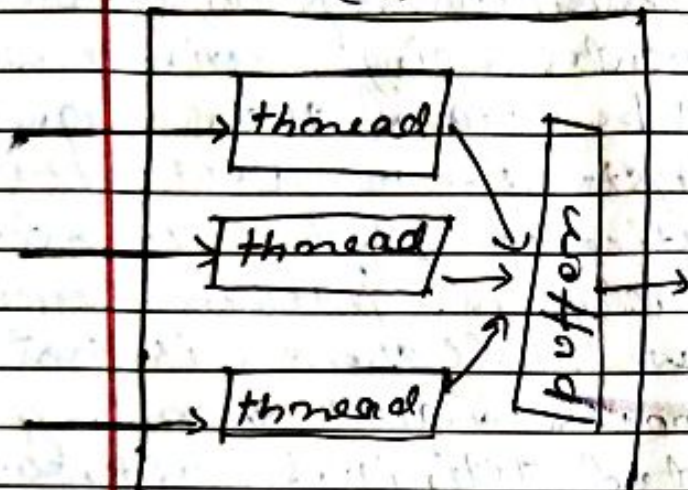
- An address space
- Thread synchronization and communication resources such that semaphores and communication interfaces.
- Higher-level resources such as open files and windows.

relation



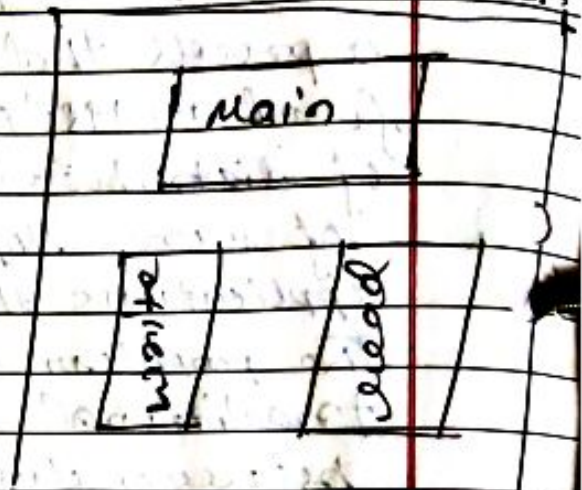
2.1.1 Thread Applications

(a) Terminal



Identical static threads

(b) File Server



Dynamic threads with dispatcher

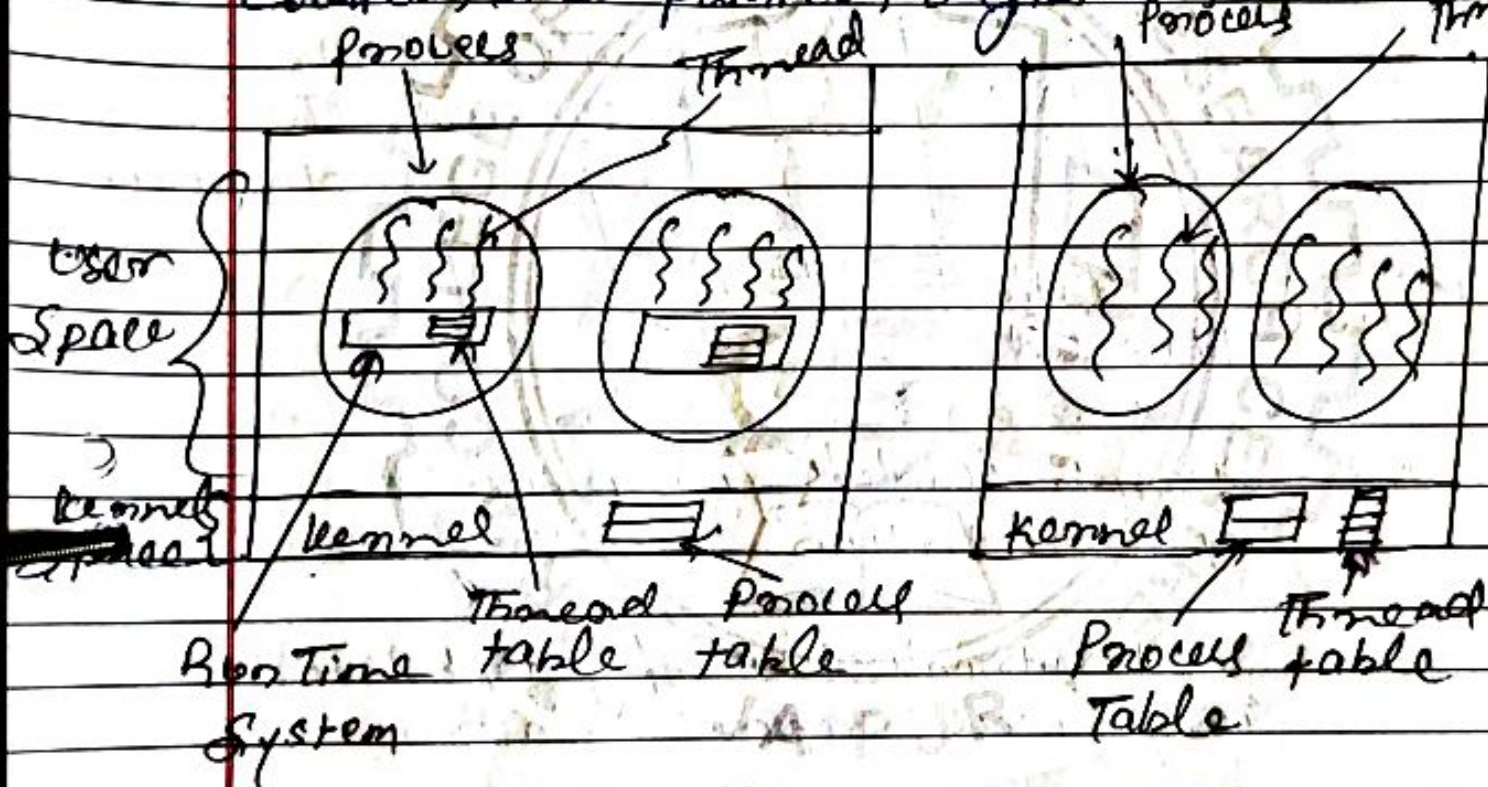
Modern applications & systems

- Multitasking is multiple applications running at once.
- Multithreading is multiple operations performed at the same time.

2.1.2 User space thread implementation

- Many kernel provide native support for multi-threaded processes including Windows, Linux, Solaris. These kernel provide thread creation and management system calls, and they schedule individual threads.

When threads are managed in user space, each process needs its own private thread table to keep track of the thread in that process. This table is analogous to the kernel's process table, except that it keeps track only of the per-thread properties such as each thread's program counter, stack pointer, registers, state, etc.

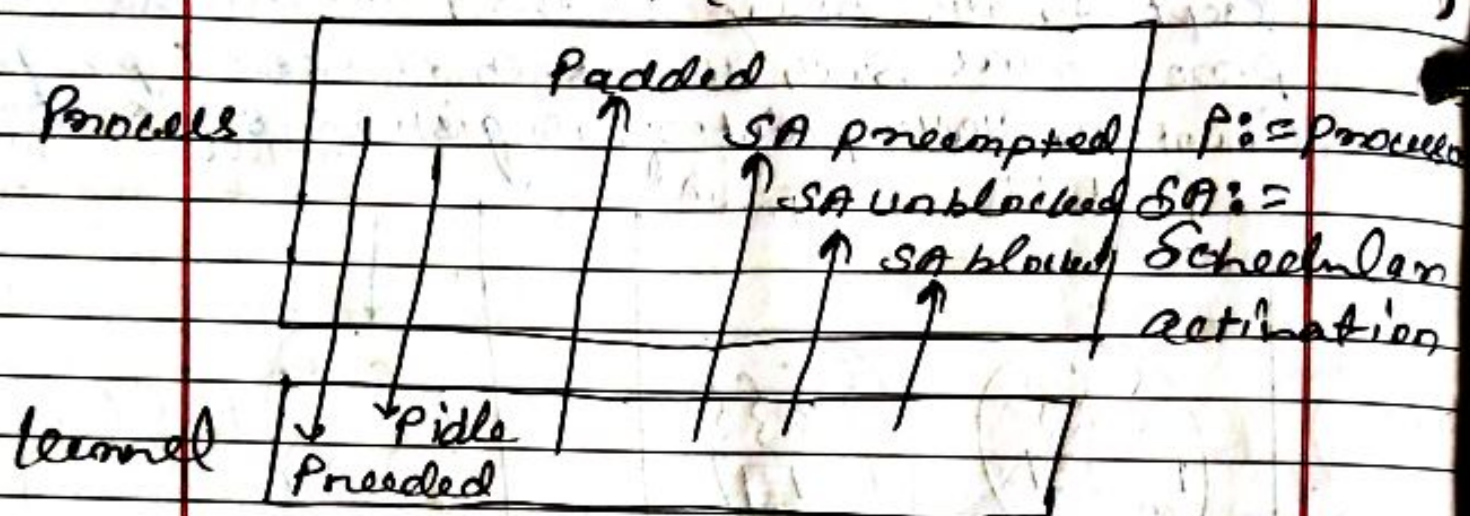


2.1.3 Kernel Space Implementation

- The kernel's thread table holds each thread's registers, state and other information. The information is the same as with user-space threads, but it is now in the kernel instead of in

use on space.

- All calls that might block a thread are implemented as system calls, at considerably greater cost than a call to a run-time system procedure.

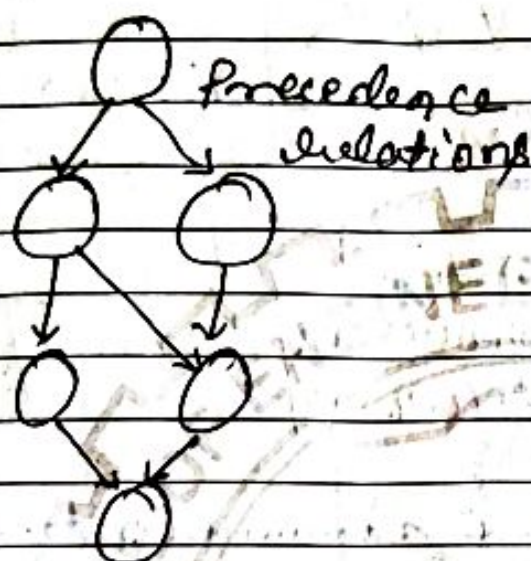


2.2 Graph Models for project representation

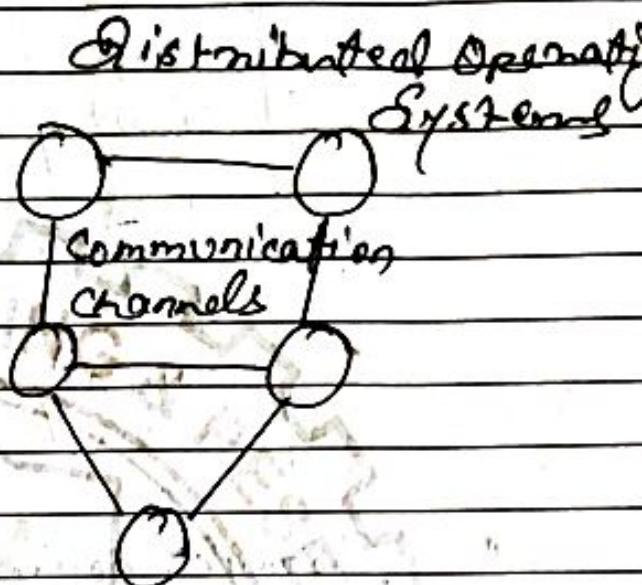
2.2.1 Introduction of graph models for process representation

- Processes are dilated by their need for synchronization and communication.
- A special case of synchronization is the precedence relationship between processes.

T. Seidman



Synchronous process graph (DAG)



Asynchronous process graph

Expression of interaction of processes can be done by taking an existing sequential language.

Peer-to-peer relationship among processes with message passing; implementation of the communication process path.

2.3 Client Server Model

- Interaction through a sequence of requests and responses.

- Flow control is redundant for the majority of invocations, which pass

8

2.3 Client Server Model

- Interaction through a sequence of requests and responses.
- Flow control is redundant for the majority of innovations, which pass

Name of Lecturer :

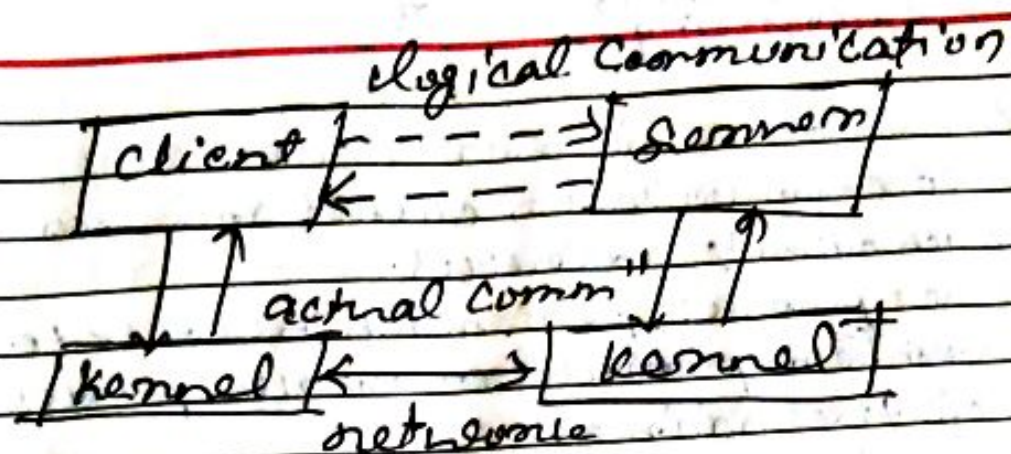
only small arguments and results



- Service oriented communication Model
- Higher-level abstraction of IPC than RPC or message passing communication

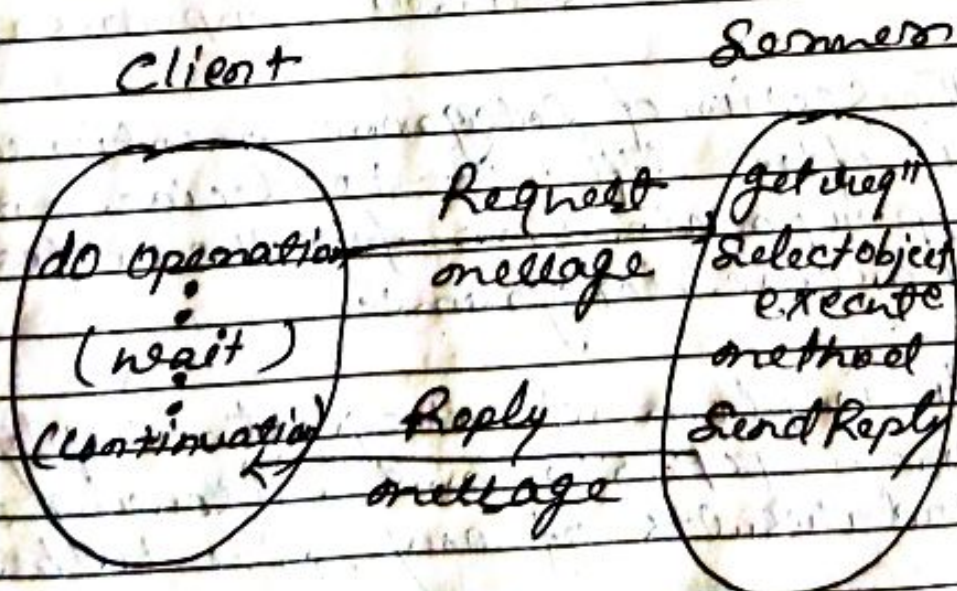
RPC Communication	Message-Passing Communication
Connection oriented or connection-less transport service	

- Standard horizontal or vertical partitioning of modules can be applied to the structure of Services.



2.3.4 Client Server Architecture

- Client/Server architecture is a computing model in which the server hosts, delivers and manages most of the resources and services to be consumed by the client.



- Client Server Architecture may also be referred to as a networking computing model because all services are

2.3.2 Client Server addressing

To communicate, client and server need each other's addresses.

- One process per machine (easy to find out)
- knows process and machine part of the address
- Broadcast requests (too many processes are listening)
- use a unique name server.
- Reality: Most important servers have "well-known" port names. Clients use "ephemeral ports" valid only while the client process is alive.

2.3.3 Client Server Implementation.

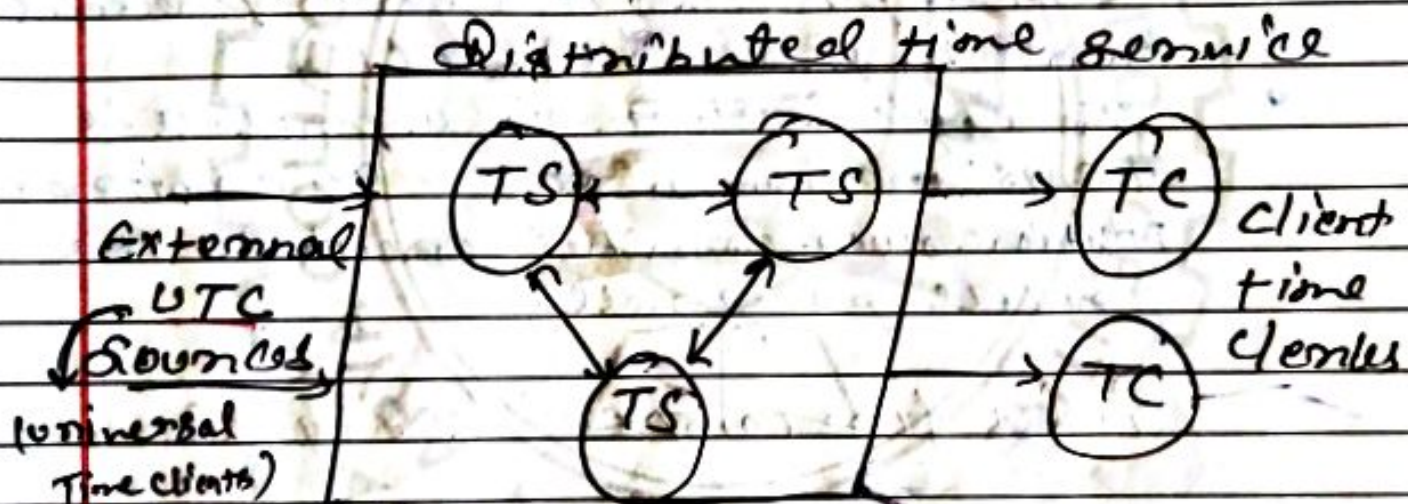
- The section of datagrams mentioned that it is often difficult to decide on an appropriate size for the buffer in which to receive datagrams.
- To implement request-reply protocol over TCP streams, allowing arguments and results of any size to be transmitted.
- Thus, the TCP protocol is chosen for request-reply protocols because it can simplify their implementations.

2.1 Time Services

- clocks are used to represent time (a relative measure of a point of time) and timers (an absolute measure of a time interval)
 - when an event occurs
 - How long it takes
 - which event occurs first

- There is no global time in distributed system

2.1.1 Physical Clocks



- Compensating delay
 - UTC Sources to time servers
 - time servers to clients
- Applications of physical clocks!

- Protocols rely on a time-out for handling exceptions
- Timestamping for secure internet communication

2.1.2 Logical clock

For many applications, events need not be scheduled or synchronized with respect to the real time-clock.

- Each process P_i in the system maintains a logical clock C_i .
- \rightarrow : happens-before relation - to synchronize the logical clocks.
- $a \rightarrow b$: Event a precedes event b
- Within a process, if event a precedes event b , then $C_i(a) < C_i(b)$
- The logical clock in a process is always incremented by an arbitrary positive number when events in the process progress.

Rules for Lamport's logical clock:

1. If $a \rightarrow b$ within the same process then $C_i(a) < C_i(b)$
2. If a is the sending event of P_i and b is the corresponding receiving event of P_j then $C_i(a) < C_j(b)$.

Implementation of the rules

- a) $C(b) = C(a) + d$
b) $C_j(b) = \max(TS_a + d, C_j(b))$, TS_a is the timestamp of the sending event.

2.4.3 Vector Logical clock

For every event a process P_i maintains the vector $VC_i(a) = [TS_1, TS_2, \dots, C_i(a), \dots, TS_n]$ where n is the number of cooperating processes, $C_i(a) = TS_i$ is the logical clock for event a at P_i and TS_k is the best estimate of the logical clock time from process P_k .

- VC_i is initialized to zero vector at system startup.
- The logical clock within a process is incremented according to rule 1.
- Rule 2 is modified: When sending message m from P_i (event a) to P_j , the logical timestamp $VC_i(m)$ is sent along with m to P_j . Let b be the earliest receiving m at P_j . P_j updates its logical clock vector $VC_j(b)$ such that $TS_k(b) = \max(TS_k(a),$

$TS_k(b)$ for every $k=1 \dots n$
and also increments its
logical clock according to the
original clock.

2.4.4 Matrix Logical Clock

A matrix clock is a mechanism
for capturing chronological and
causal relationship in a distributed
system.

Matrix clock are a generalization
of the notion of vector clocks. A
matrix clock maintains a vector
of the vector clock for each
communicating host.

Every time a message is exchanged,
the sending host sends not only
what it knows about the global
state of time, but also the state
of time that it received from
other hosts.

2.5 Language mechanism for synchronization

2.5.1 Language Constructs

A concurrent
language extended from a
sequential language adds

additional constructs to provide:

- Specification of concurrent activities.
- Synchronization of processes
- Interprocess communication
- Nondeterministic execution of process

Synchronization methods	Language facilities
Semaphore	Shared variable and system call
monitor	data type abstraction
Conditional critical region	Control structure
Serializers	data type and control structure
Path expression	data type and program structure
communicating sequential processes	input and output
elementary procedure call	procedure call
rendezvous	procedure call and communication

2.5.2 Shared variable Synchronization

The satisfaction of constraints on the interleaving of the actions of processes (e.g. an action by one process only occurring after an action by another)

Consider two processes updating a shared variable, x , with the alignment: $x := x + 1$

- load the value of x into some register

- increment the value in the register by 1 and

- store the value in the register back to x .

2.5.3 Message passing Synchronization

- The only means of communication in distributed systems.

- Implicit Synchronization: messages can be received only after they have been sent.

- Non-blocking send, blocking receive:

- Asynchronous message passing

- Blocking send, blocking receive:

- Synchronous message passing.

Synchronous message passing:-

- No buffering of messages in the communication channel.
- Rendezvous between send and receive.
- Examples: Communication Sequential Processes (CSP), Remote procedure call (RPC)

Asynchronous message passing:-

- Is an extension of the semaphore concept to distributed systems.
- Send operations assume that the channel has an unbounded buffer.
- Example: pipe and socket.

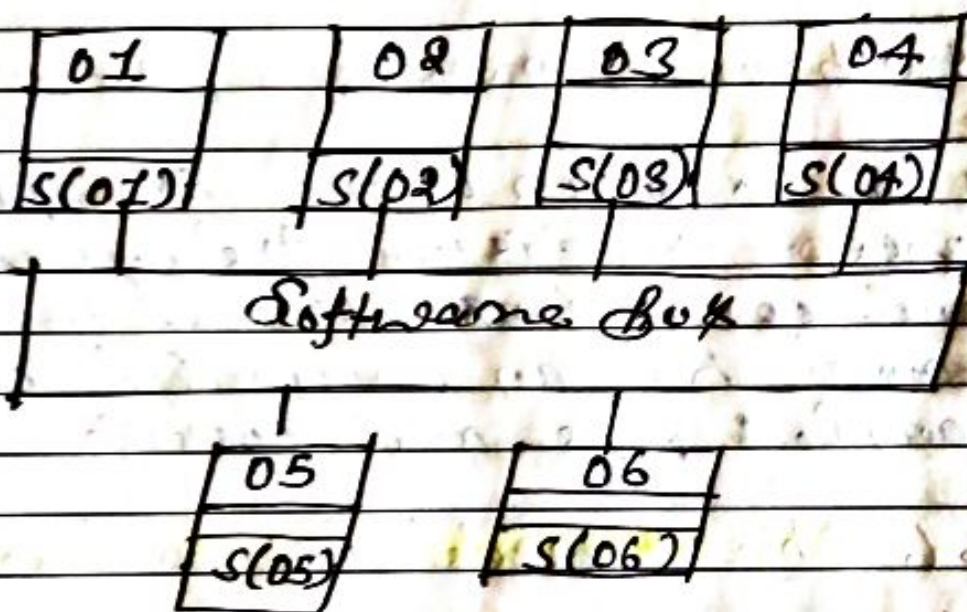
2.6 Object Model Resource Semantics

2.6.1 Introduction of Object Model Resource Semantics

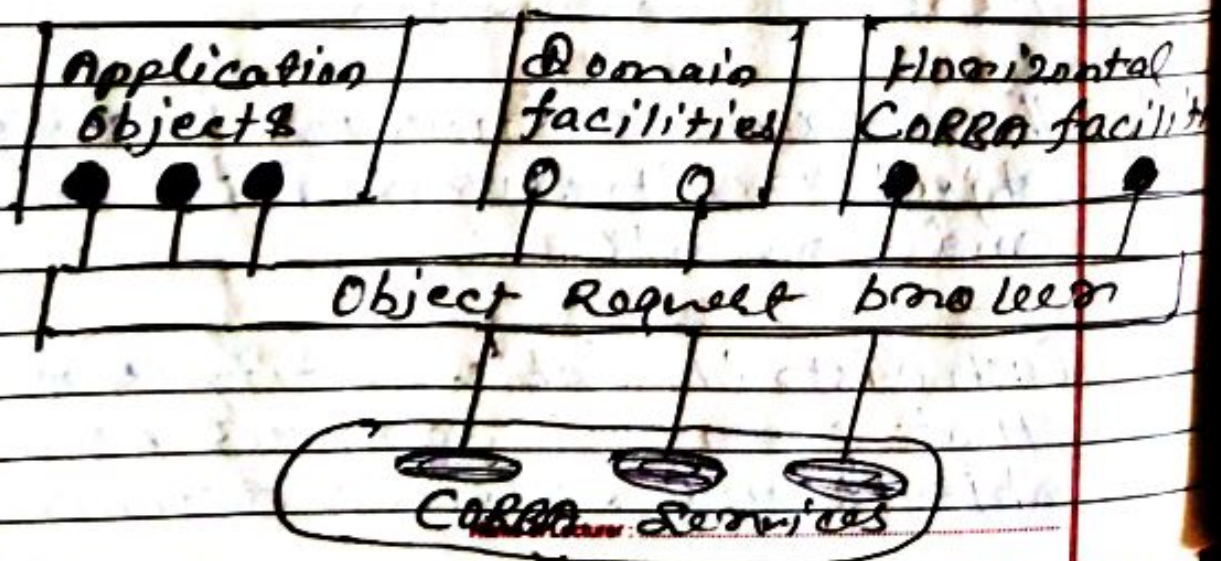
- There is no distinction in a distributed object architectures between clients and servers.
- Each distinguishable entity is an object that provides services to other objects and receives services from

other objects.

- Object communication is through a middleware system called an object request broker (Software bus).
- Homogeneous more complex to design than C/S systems.



CORBA application Structure :-



2.7 Characteristics of Concurrent Programming Language

Concurrent computing is a form of computing in which several computations are executing during overlapping time periods concurrently - instead of sequentially. This is a property of a system - this may be an individual program, a computer, or a network - and there is a separate execution unit and "thread of control" from each computation.

- Increased application throughput - parallel execution of a concurrent program allows the number of tasks completed in certain time period to increase.
- High responsiveness for input/output:- Input/output intensive applications mostly wait for input or output operations to complete. Concurrent programming allows the time that would be spent waiting to be used for another task.

- More appropriate program structure -

Some problems and problem domains are well-suited to representation as concurrent tasks or processes.

2.8 Inter process communication and coordination

- Distributed TPC and process coordination are based on message passing.
- Dependent on the ability to locate communication entities: deal of the name service.
- Three fundamental message passing communication models:
 - message passing
 - request/reply (RPC)
 - transaction communication
- Distributed process coordination examples:
 - Distributed mutual exclusion
 - leader election

2.3.1 Message passing communication

2.3.1.1 ~~Box~~ • Messages are collections of data objects

- Their structure and interpretation are defined by the peer applications
- Communicating processes pass composed messages to the system transport service.

Inter process communication	transaction request/reply (RPC) message passing
network operating system	Transport connection
communication network	packet switching

2.3.1.1 Basic Communication Primitive

- send (destination, message)
- receive (source, message)

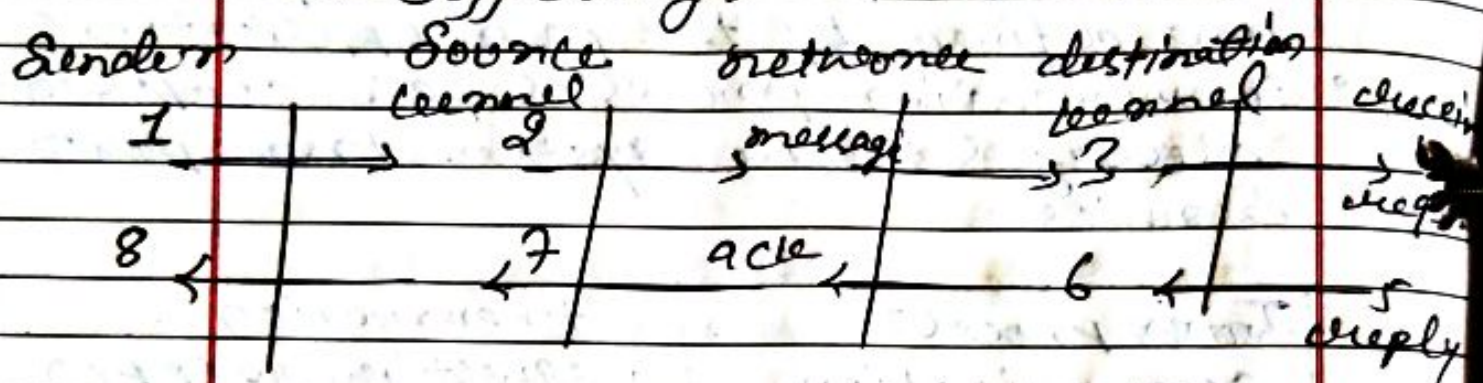
where source or destination = (Process name, link, mailbox, port)

Process name (global PID) - direct communication primitive link (connection)

mailbox - indirect communication primitive many-to-many

Port - indirect communication
Primitive many-to-one

2.8.1.2 Message Synchronization
and Buffering:



- Nonblocking send: 1+8
- blocking send: 1+2+7+8
- Reliable blocking send: 1+2+3+6+7+8
- Explicit blocking send:
1+2+3+4+5+6+7+8
- Request and reply: 1-4, service, 5-8

At the receiving site blocking
is quite explicit: blocked for message
arrival

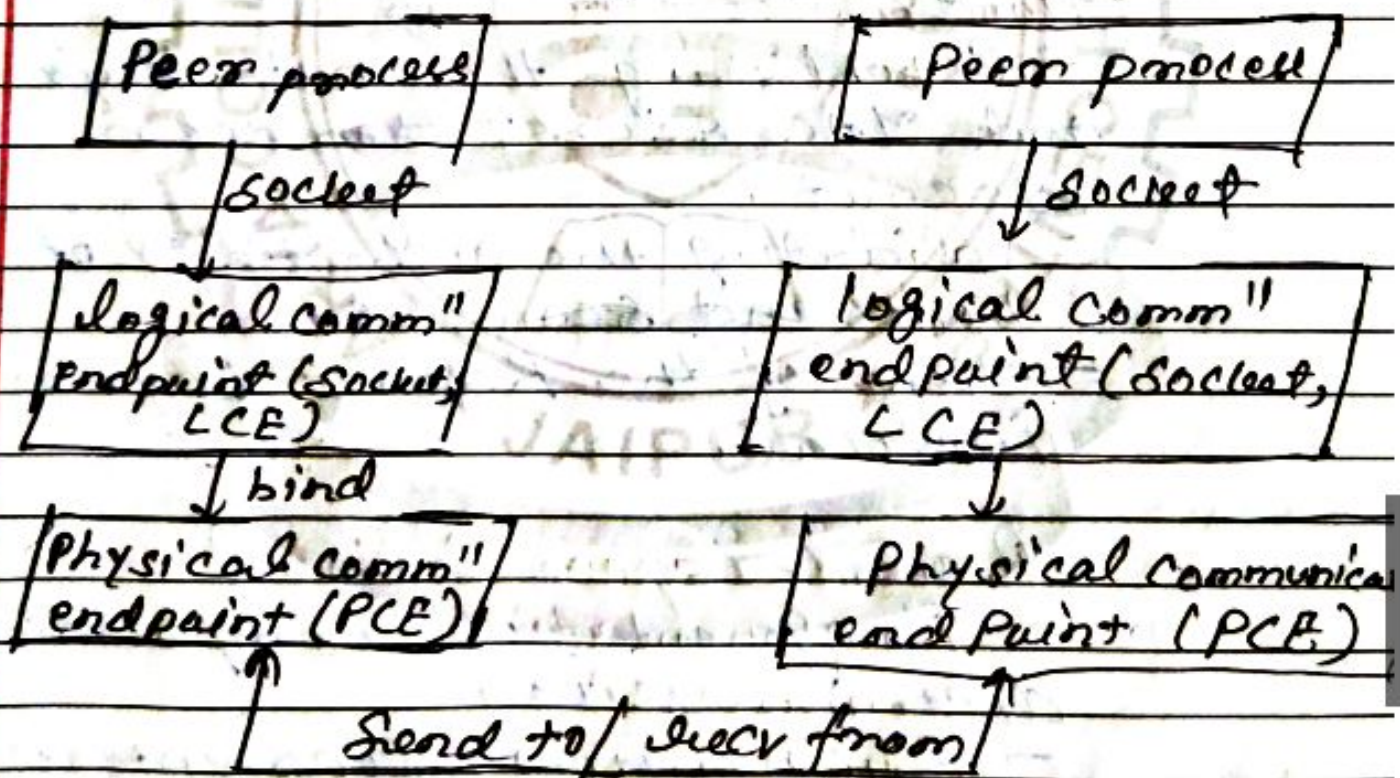
Implicit buffer space:

- In sender's channel
- In receiver's channel
- In the communication
network

2.8.1.3 Pipe and Sockets API's

- Pipe: implemented with finite-size, FIFO byte stream buffer maintained by OS kernel
 - Created with the pipe system call, which returns two descriptors.
 - Data in pipes are uninterpreted byte sequences and are anonymous.

• Socket is a communication endpoint of communication link managed by the OS's transport system.



Connectionless socket communication

- modeling network I/O based on conventional file I/O
- created by the socket system call
- Used for file-oriented read/write operations.
- Used for communication specific send/receive operations.

2.8.1.4 Secure Sockets Layer:-

- Privacy in socket communication
- Integrity of socket data
- Handshake protocol
 - Establishing the write keys and MAC secret \rightarrow master secret.
 - Validating the authenticity of clients and servers.
 - Client of the Record Layer Protocol.
- Record Layer Protocol
 - Fragmentation, compression/decompression.
 - Encryption/decryption of message stream.

2.8.1.5 Group communication and multicast

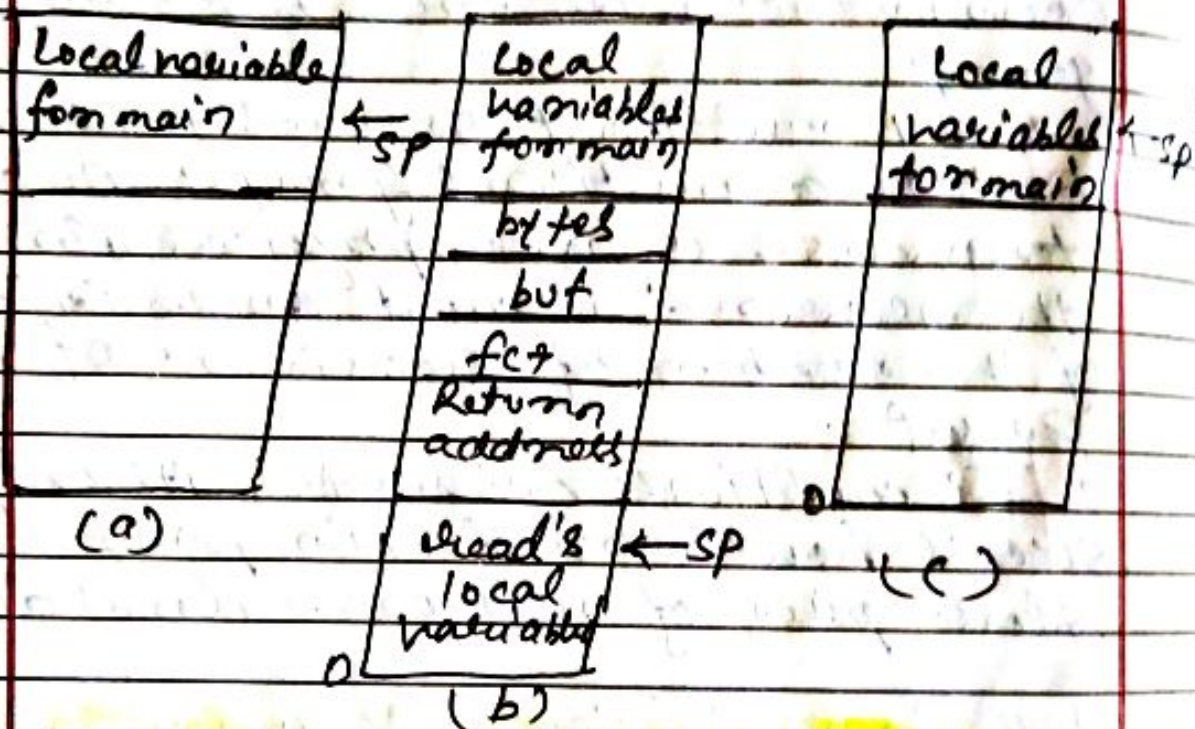
- Multicast is a means of one-to-many communication. It is essential for scalable group communications as it allows a group member to communicate with one, with an abstract group.
- IP multicast ensures that a packet sent to a multicast group will only traverse a given physical link in the network at most once, independent of the number of recipients of that packet.
- IP multicast has met with only moderate success however, due in part to the slow pace of network-level deployment.

2.9 Request/Reply Communication

- The Request/Reply Communication method is a very common technique for one application to request the services of another.
- Most widely used request/reply communication model is Remote Procedure Call (RPC).

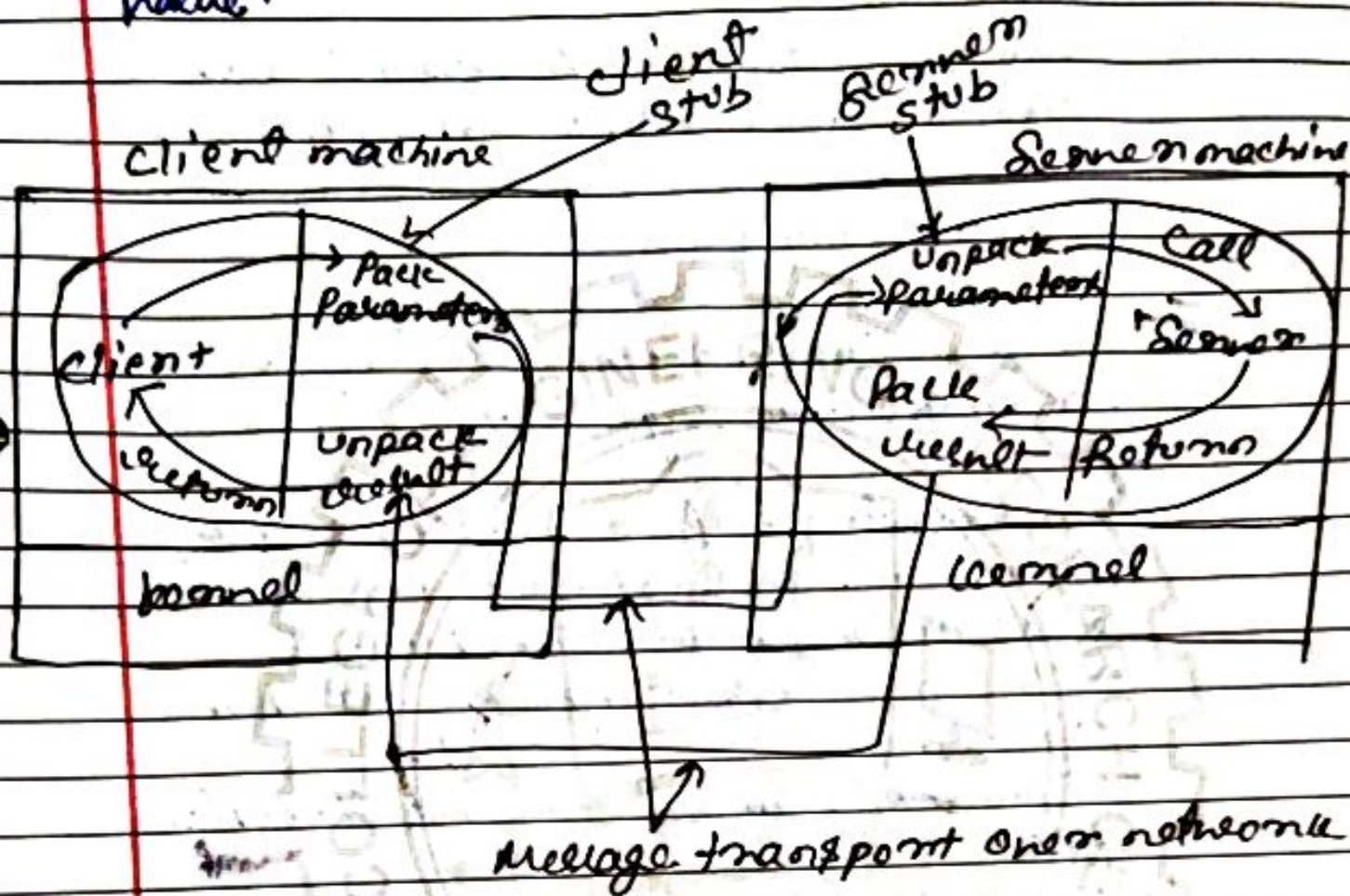
2.9.1 RPC Operations

RPC works, it is important first to fully understand how a conventional procedure call works.



- A reference parameter in C is a pointer to a variable (i.e. the address of the variable), rather than the value of the variable.
- It consists of having the variable copied to the stack by the caller, as in call-by-value, and then copied back after the call, overwriting the caller's original

value:



When the message arrives at the server, the kernel passes it up to a server stub that is bound with the actual server.

2.9.2 RPC Exception and failure Handling

- Exceptions
- Abnormal conditions raised by the execution of stub and callee procedures.
- Example: Overflow / underflow, protection violation.

- Failures
- Problems caused by crashes of clients, servers, or the communication

Exception Handling: ↗

- In local procedure call: global variables and signals.
- In computer network, the exchange of control and status information must rely on a data channel
 - In band signaling, or out-band signaling
 - Separate channel - more flexible for RPC
 - It is implemented as part of the stub library support and should be transparent.

Failure Handling: ↗

- Cannot locate the server
 - Nonexistent server, or outdated program
 - handle client as exception
- Message can be delayed or lost
 - message can be retransmitted

§9.3 Secure RPC

- Security is important for RPC, since
 - RPC introduces vulnerability because it opens doors for attacks.
 - RPC becomes a cornerstone of client/server computation.
 - Security features should be built on top of a secure RPC.
- Authentication protocol for RPC should establish:-
 - Mutual authentication
 - Message integrity, confidentiality, and originality.
- Design of a secure authentication protocol
 - How strong the security goals?
 - What possible attacks?
 - Some inherent limitations of system.
 - Short term solution: additional security features.

§10 Transaction Communication

If all users that people are connected through communication; they engage in transaction.

First, it is recognized that each of us is a sender-receiver not merely a sender or a receiver.

2.10.1 The ACID properties

The ACID is a set of properties that guarantee that database transactions are processed reliably. In the context of databases a single logical operation on the data is called transaction.

- **Atomicity**: - Atomicity requires that each transaction be "all or nothing". If one part of the transaction fails, and the database state is left unchanged.
- **Consistency**: - The consistency property ensures that any transaction will bring the database from valid state to another.

Isolation: - The isolation property ensures that the concurrent execution of transactions results in a system state that would be obtained if transactions were executed serially.

- ^{Commit} Durability:- Durability means that once a transaction has been committed. It will remain so, even in the event of power loss, crashes, or errors. In a relational database, for instance, once a group of SQL statements executes.

2.10.2 Two phase commit Protocol

During the progress of the transaction, there is no communication between the Coordinator and the participants apart from the participants informing the Coordinator when they join the transaction.

Phase 1 (Voting Phase):

1. The coordinator sends a can commit? request to each of the participants in the transaction.

2. When a participant receives a can commit? request it replies with its vote (Yes or No) to the coordinator. Before voting Yes, it prepares to commit by saving objects in the permanent storage.

If the vote is no the participants aborts immediately.

Phase 2 (Completion according to outcome of vote):-

3. The coordinator collects the votes (including its own)

a) If there are no failures and all the votes are Yes the coordinator decides to commit the transaction and sends a do Commit request to each of the participants.

b) Otherwise, the coordinator decides to abort the transaction and sends do Abort requests to all participants that voted Yes.

4. Participants that voted Yes are waiting for a do Commit or do Abort request from the coordinator. When a participant receives one of these messages, it acts accordingly and in case of commit, sends a have Committed call as confirmation to the coordinator.

2.11 Name and Directory Services

Clients find out about the services provided in a smart space using a discovery service. A discovery service is a directory service in which services in a smart space are registered and looked up by their attributes, but one whose implementation takes account of relative system properties.

2.11.1 Name and Address Resolution

A name in the directory service consists of one or more parts, technically called labels, that are conventionally concatenated and delimited by the dots.

a) A network host is configured with an initial cache (so called hints) of the known addressees of the root name servers. Such a hint file is updated periodically by an administrator from a reliable source.

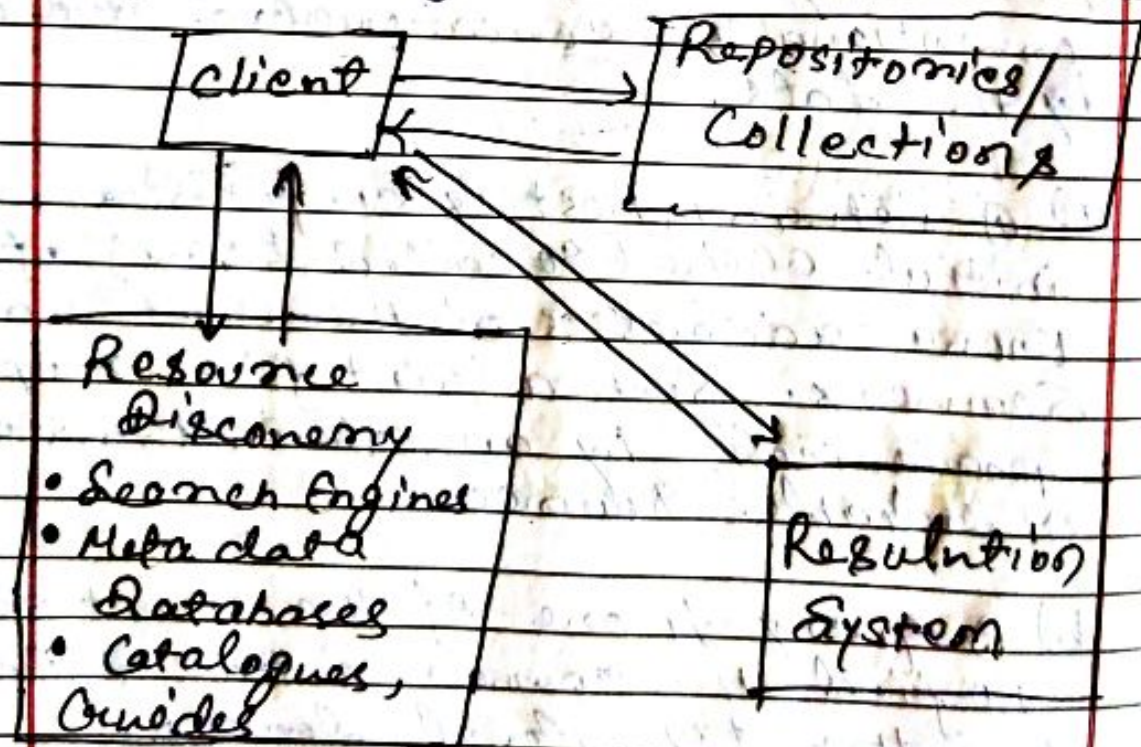
b) A query to one of the root servers to find the server authoritative for the top-level domain.

e) A query to the obtained TLD
Server for the address of a
DNS server authoritative for
the second-level domain.

d) Repetition of the previous step
to process each domain name
label in sequence, until the final
step which returns the IP
address of the host sought.

2.11.2 Object Attributes and Name Structure

→ framework for managing
digital objects:



- Go from name to attributes
- Fundamental indication system for Digital Object management on the net.
- Frees digital content from constraints of specific technologies.

2.11.3 Name space and Information Base

A namespace is a container for a set of identifiers. Namespaces provide a level of direction to specific identifiers, thus making it possible to distinguish between identifiers with the same exact name.

- For efficient name resolution and management, there is a need for an information model to serve as a basis of the name space database implementation.

Directory Service Agent (DSA) :-

- The servers for the name service
- Naming contexts are the basic unit.

Naming Context

— Partial subtree of the DIT

- The resolution request is sent from one DSA to another until the object is found in DIT.
(Directory Information Tree)
(DUAL Directory User Agent)

