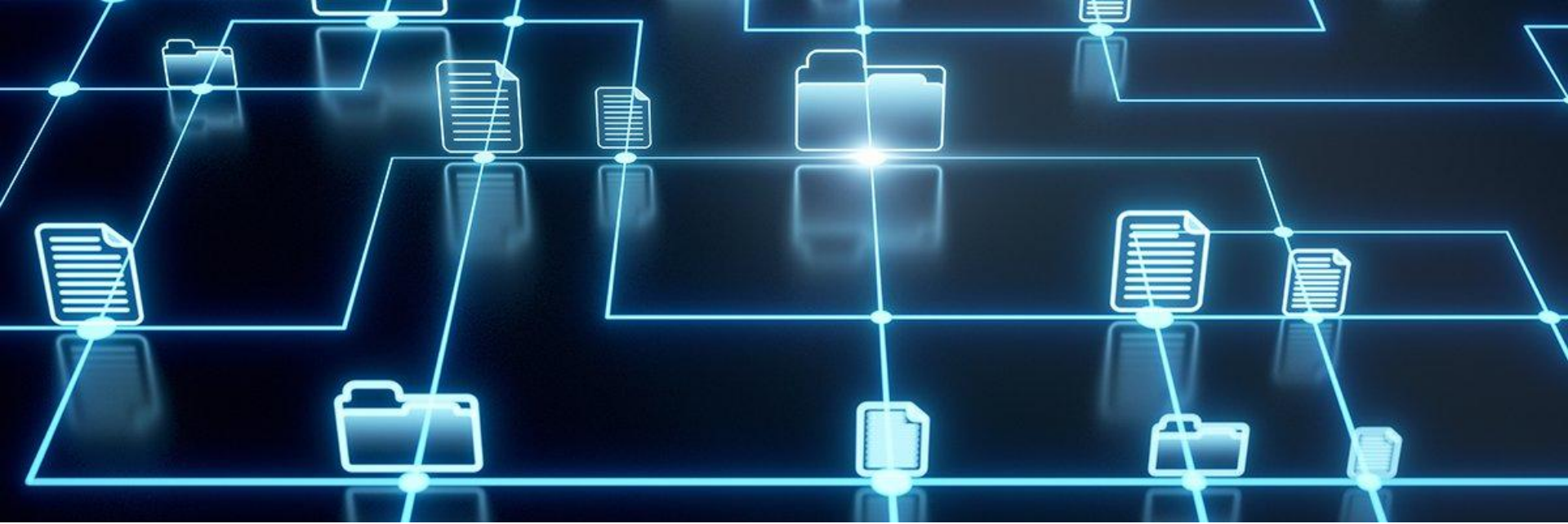


CHAPTER 2-Distributed Objects and File System

BY ANKU JAISWAL
ASST.PROF. IOE,PULCHOWK CAMPUS

DISTRIBUTED OBJECT

Distributed objects are reusable software components that can be distributed and accessed by the user across the network.

–These objects can be assembled into distributed applications.

Example of Distributed Object

- Let's consider a distributed file system, like the **Google File System (GFS).** In this system, a file can be represented **as an object.**
- Each file object contains metadata such as its name, size, location, and permissions, along with the actual data stored in the file.
- The file object exposes methods for operations such as reading from the file, writing to the file, appending data, deleting the file, etc.
- These methods can be invoked by clients distributed across the network.

In a distributed system, objects like files might be distributed across multiple nodes in the network for fault tolerance, scalability, and performance reasons.

Other example of distributed object

Content Delivery Networks (CDNs): CDNs distribute objects such as web pages, images, and videos across multiple edge servers to reduce latency and improve content delivery speed.

Blockchain Networks: Blockchain platforms distribute objects (blocks) across a network of nodes for decentralized and immutable ledger maintenance. Each block contains transactions and references to previous blocks.

Uses of Distributed Objects

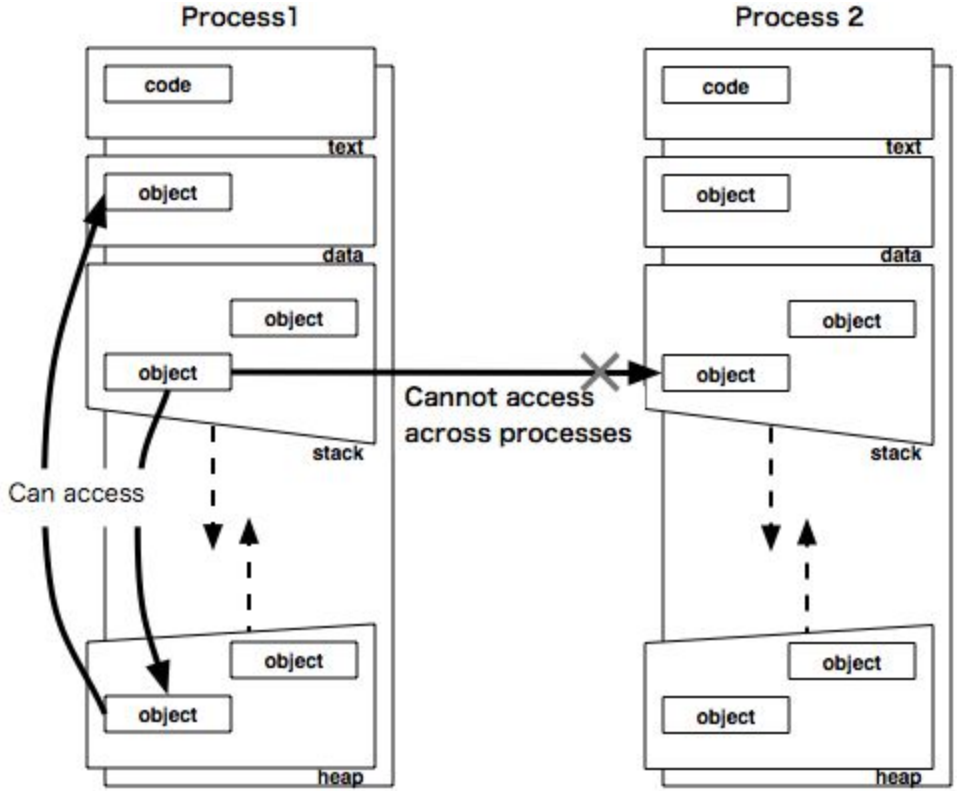
Distributed objects might be used :

1. to share information across applications or users.

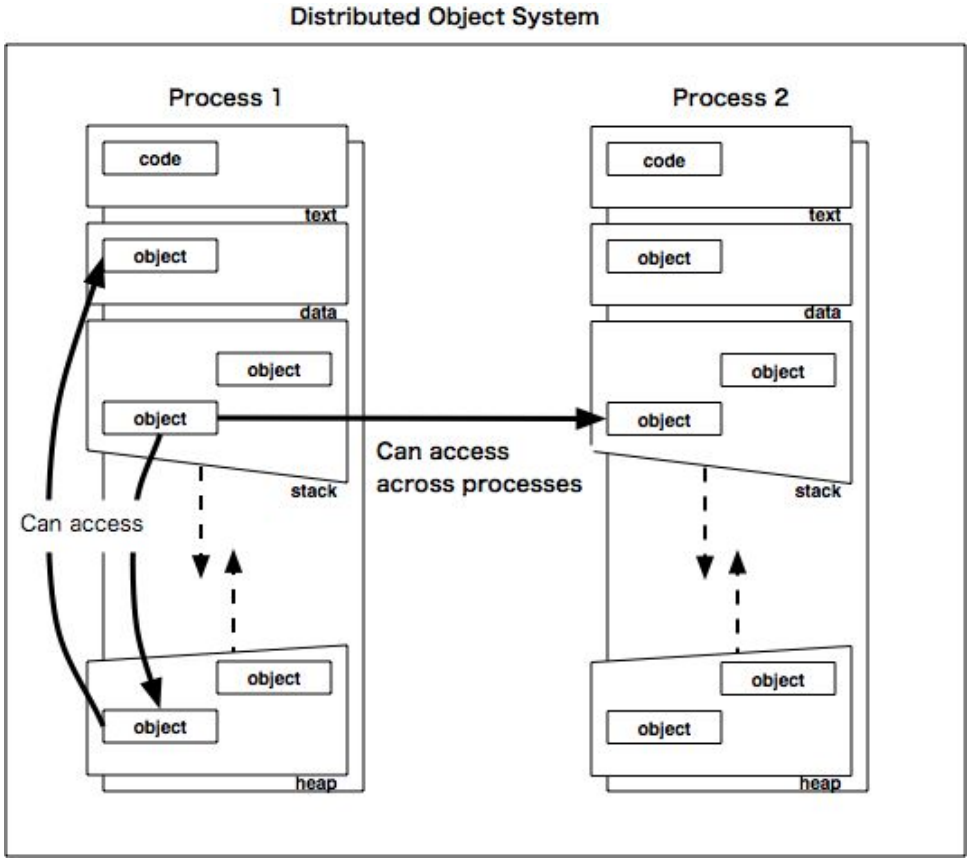
2. to synchronize activity across several machines.

3. to increase performance associated with a particular task.

Location of objects and processes for a normal system



Location of objects and processes within a distributed object systems



DISTRIBUTED OBJECT COMMUNICATION

The main method of distributed object communication is with **remote method invocation**, generally by message-passing:

–one object sends a message to another object in a remote machine or process to perform some task.

–-The results are sent back to the calling object.

RMI(REMOTE METHOD INVOCATION)

- **Distributed objects generally communicate** with Remote Method Invocation (RMI).
- RMI has message passing mechanism in which :

  – one object sends message to another object in a remote machine or process

  –to carry out some task and the results are sent back to the calling object.

RMI uses **stub and skeleton** object for communication with the remote object.

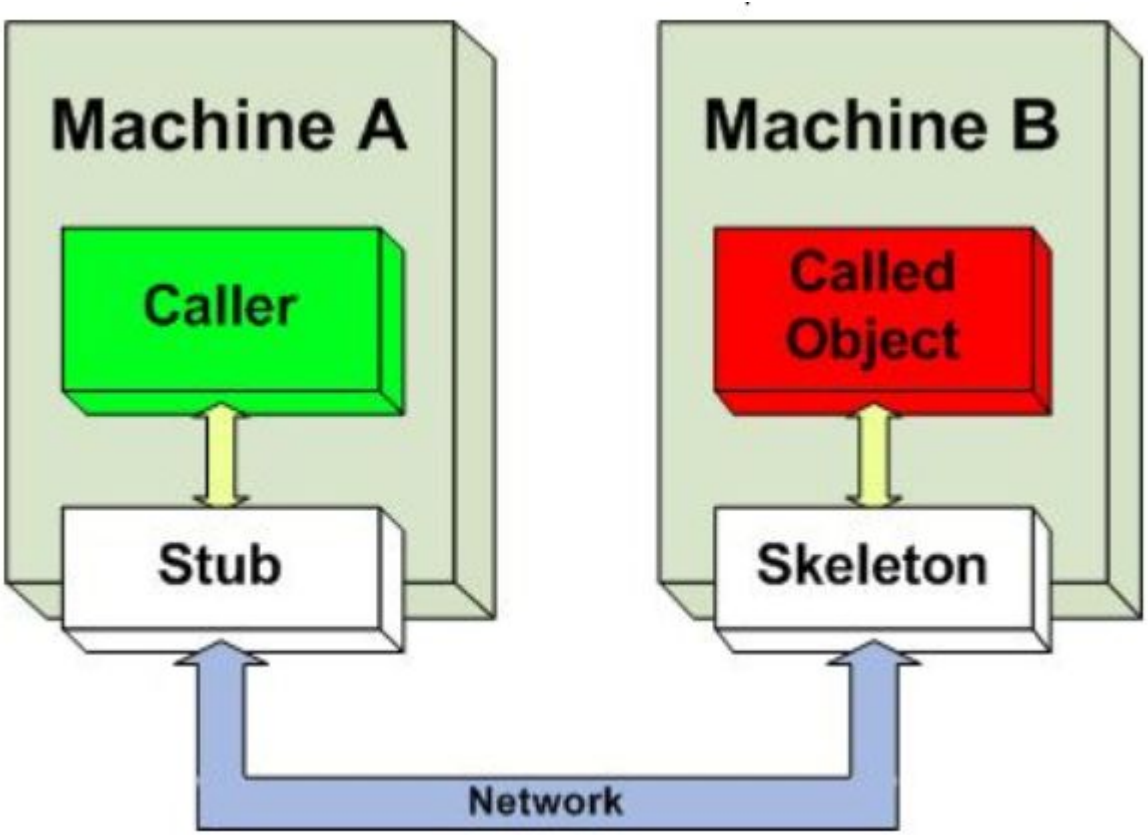
EXAMPLE

Distributed objects are used in Java RMI.

CORBA lets one build distributed mixed object systems.

DCOM(Distributed Component Object Model) is a framework for distributed objects on the Microsoft platform.

Pyro is a framework for distributed objects using the Python programming language.


Machine A
Caller
Stub
Machine B
Called Object
Skeleton
Network

Stub − A stub is a 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.

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 ,
2. It writes and transmits (marshals-pack) the parameters to the remote Virtual Machine
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 unmarshall the parameters
3. It invokes the method on the actual remote object, and
4. It writes and transmits the result to the caller.

DESIGN ISSUES FOR RMI

Retry Request Message: whether to retransmit the request message until either a reply is received or the server is assumed to have a fail.

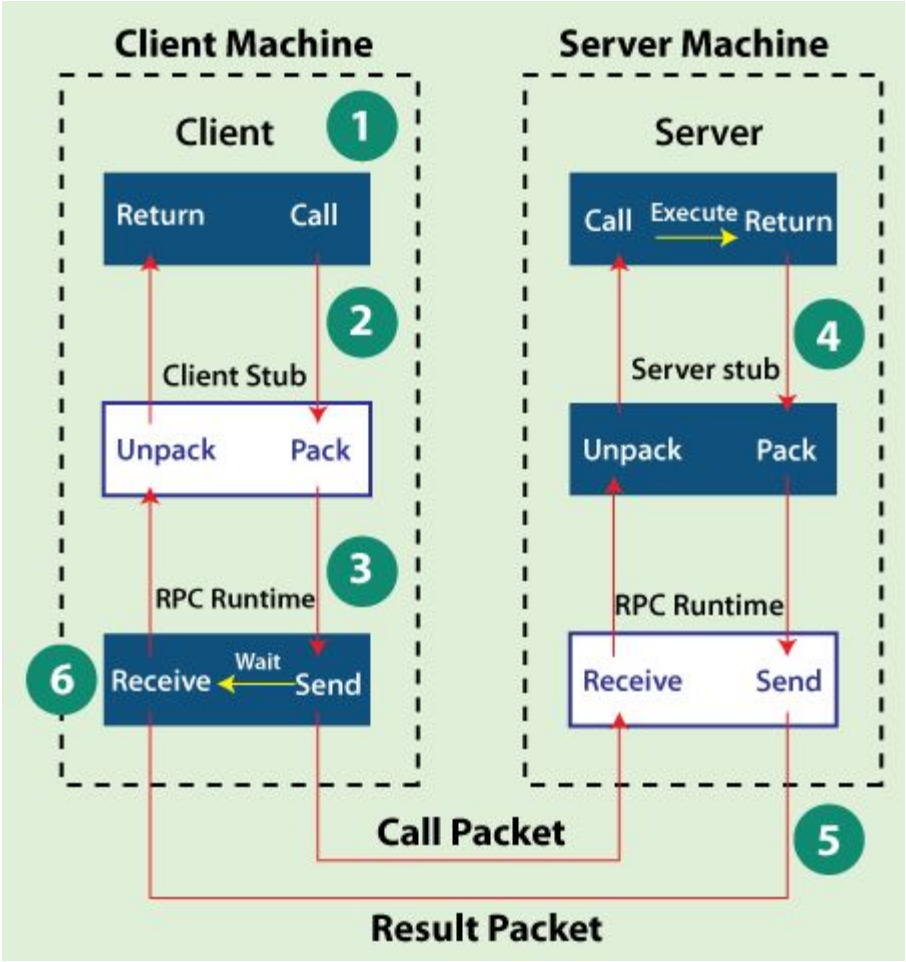
Duplicate Filtering: when retransmissions are used, server has to identify duplicate requests.

Retransmission of results: whether to keep a history of result messages to enable lost results to be retransmitted without re-executing the operations at the server.

RPC (Remote Procedure Call)

Remote Procedure Call (RPC) protocol is generally used to communicate between processes on different workstations.

RPC is a remote communication medium in which a client program calls a procedure in another program running in a server process.


Client Machine
Server Machine
Client
1
Server
Return
Call
Call
Execute
Return
2
4
Client Stub
Server stub
Unpack
Pack
Unpack
Pack
3
RPC Runtime
RPC Runtime
6
Receive
Wait
Send
Receive
Send
Call Packet
5
Result Packet

Step 1: The client, client stub, and RPC runtime execute on the client machine.

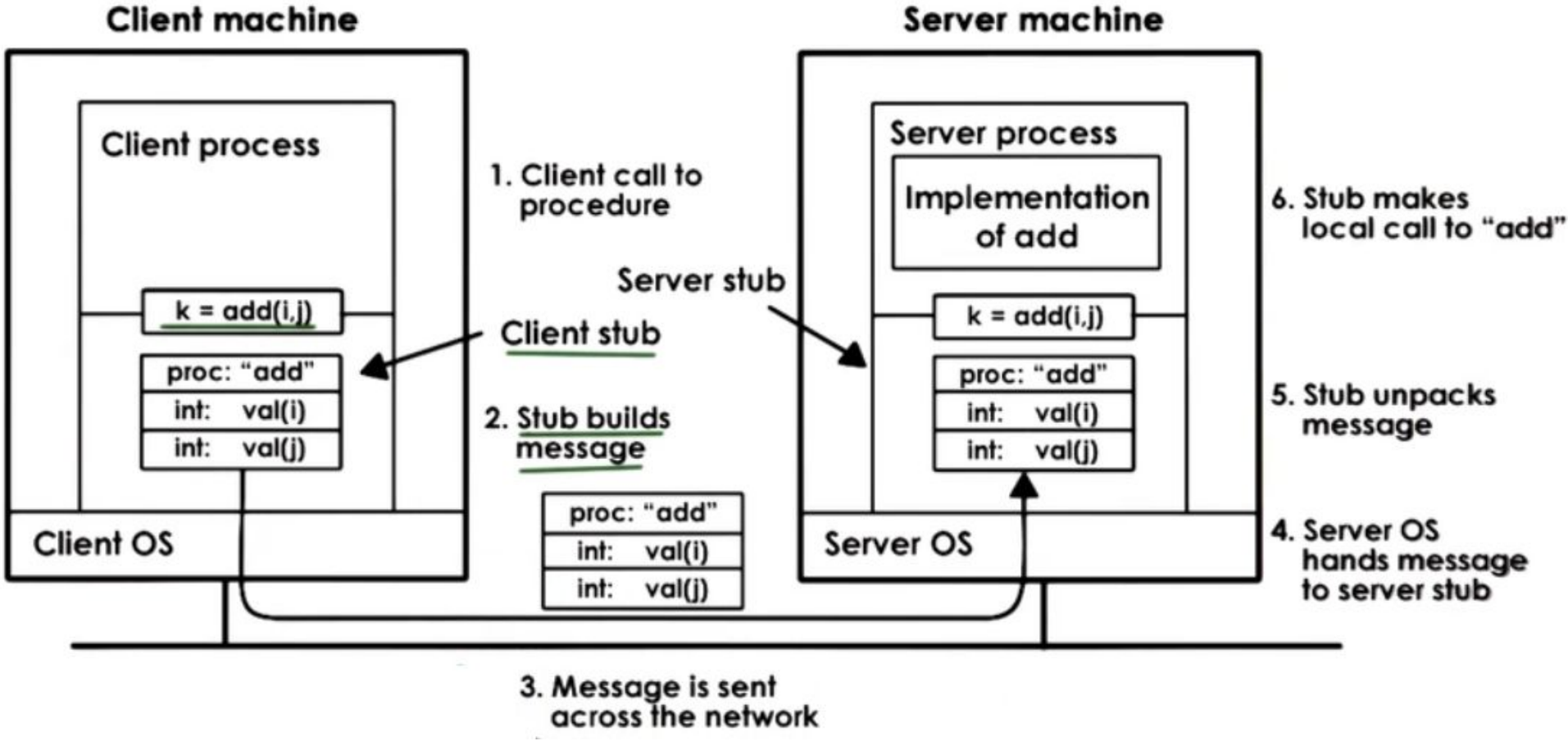
Step 2: A client starts a client stub process by passing parameters in the usual way. The packing of the procedure parameters is called ***marshalling***. The client stub stores within the client's own address space, and it also asks the local RPC Runtime to send back to the server stub.

Step 3: RPC Runtime manages the transmission of messages between the network across client and server, and it also performs the job of retransmission, acknowledgment, routing, and encryption.

Step 4: After completing the server procedure, it returns to the server stub, which packs (marshalls) the return values into a message. The server stub then sends a message back to the transport layer.

Step 5: In this step, the transport layer sends back the result message to the client transport layer, which returns back a message to the client stub.

Step 6: In this stage, the client stub demarshalls (unpack) the return parameters in the resulting packet, and the execution process returns to the caller.


Client machine
Server machine
Client process
Server process
Implementation of add
1. Client call to procedure
Server stub
k = add(i,j)
Client stub
proc: "add"
int: val(i)
int: val(j)
2. Stub builds message
6. Stub makes local call to "add"
5. Stub unpacks message
4. Server OS hands message to server stub
Client OS
Server OS
3. Message is sent across the network

Advantages of RPC

- RPC supports process and thread-oriented models.
- RPC makes the internal message passing mechanism hidden from the user.
- The effort needs to re-write and re-develop the code is minimum.
- Remote procedure calls can be used for distribution and the local environment.
- RPC provides abstraction. For example, the message-passing nature of network communication remains hidden from the user.
- RPC allows the usage of the applications in a distributed environment that is not only in the local environment.
- With RPC code, re-writing and re-developing efforts are minimized.

RPC	RMI
It is an OS and library dependent platform.	It is only a Java platform.
It supports procedural programming.	It supports object oriented programming.
Comparatively RPC is less efficient.	RMI is much more efficient than RPC.
RPC is an older version of RMI.	RMI is the newer successor of RPC.

CASE STUDY- refer site(ezexplanation.com)

Project Code:
https://pypi.org/project/sunrpc/

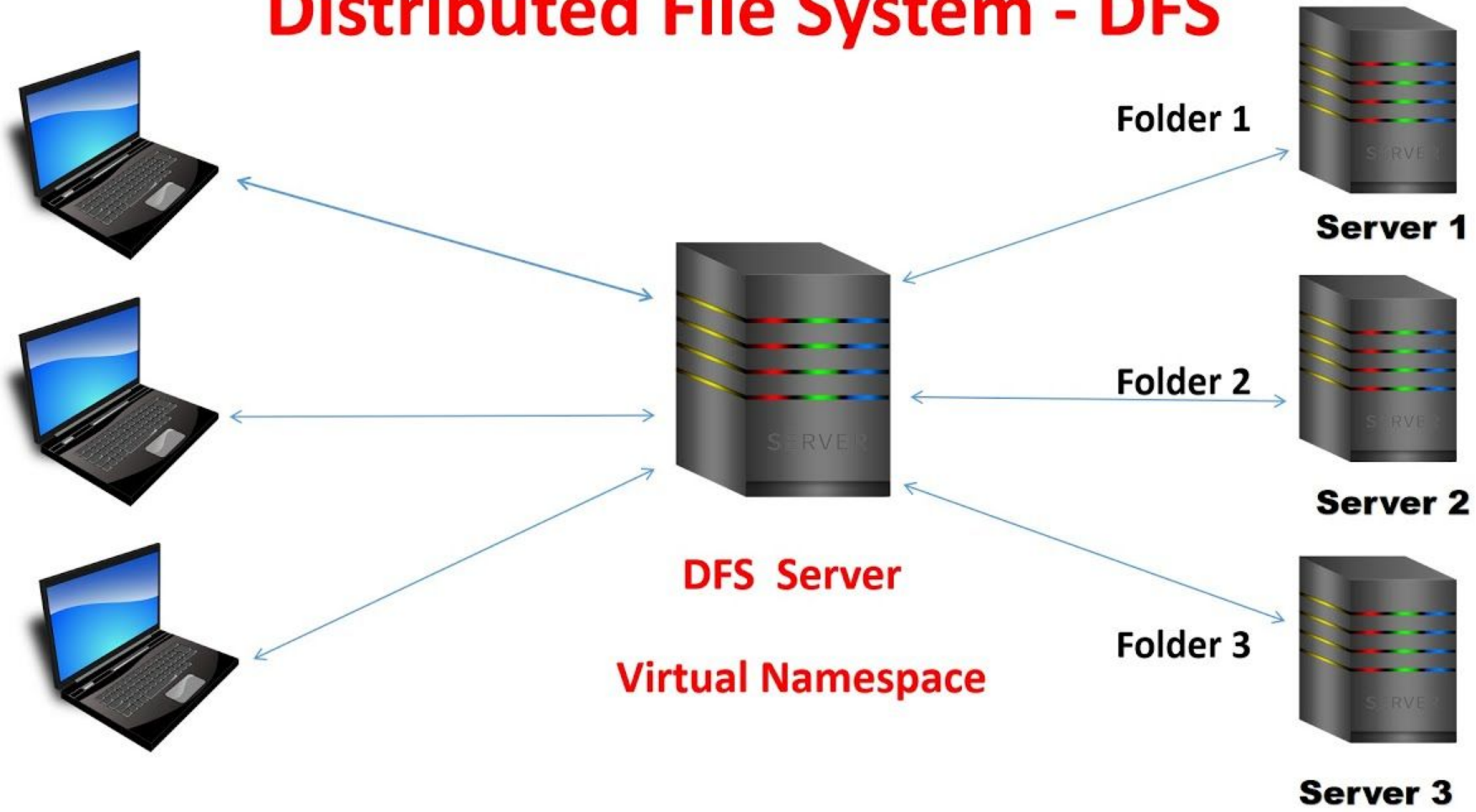
DFS(DISTRIBUTED FILE SYSTEM)

A distributed file system (DFS) is a file system that **spans across multiple file servers or multiple locations,** that are situated in different physical places.

Files are accessible just as if they were stored locally, from any device and from anywhere on the network.

It allows programs to access or store isolated files as they do with the local ones, allowing programmers to access files from any network or computer.

Distributed File System - DFS



WORKING OF DFS

- **Distribution**: First, a DFS distributes datasets across multiple clusters or nodes. Each node provides its own computing power, which enables a DFS to process the datasets in parallel.
- **Replication**: A DFS will also **replicate datasets onto different clusters** by copying the same pieces of information into multiple clusters.
- This helps the distributed file system to achieve fault tolerance
- —to recover the data in case of a node or cluster failure
- —as well as high concurrency, which enables the same piece of data to be processed at the same time.

Clients access data on a DFS using namespaces.

Organizations can group shared folders into logical namespaces.

A DFS namespace is a **virtual shared folder** that contains shared folders from multiple servers.

These present files to users as one shared folder with multiple subfolders.

When a user requests a file, the DFS brings up the first available copy of the file.

TYPES OF DFS

- Windows Distributed File System
- Network File System (NFS)
- Server Message Block (SMB)
- Google File System (GFS)
- Lustre
- Hadoop Distributed File System (HDFS)
- GlusterFS
- Ceph
- MapR File System

PROPERTIES/ADVANTAGE OF DFS

File transparency: users can access files without knowing where they are physically stored on the network.

Load balancing: the file system can distribute file access requests across multiple computers to improve performance and reliability.

Data replication: the file system can store copies of files on multiple computers to ensure that the files are available even if one of the computers fails.

Security: the file system can enforce access control policies to ensure that only authorized users can access files.

Scalability: the file system can support a large number of users and a large number of files.

ADVANTAGES OF DFS

- **Transparent local access** — Data is accessed as if it's on a user's own device or computer.
- **Location independence** — Users may have no idea where file data physically resides.
- **Massive scaling** — Teams can add as many machines as they want to a DFS to scale out.
- **Fault tolerance** — A DFS will continue to operate even if some of its servers or disks fail because machines are connected and the DFS can gracefully failover.

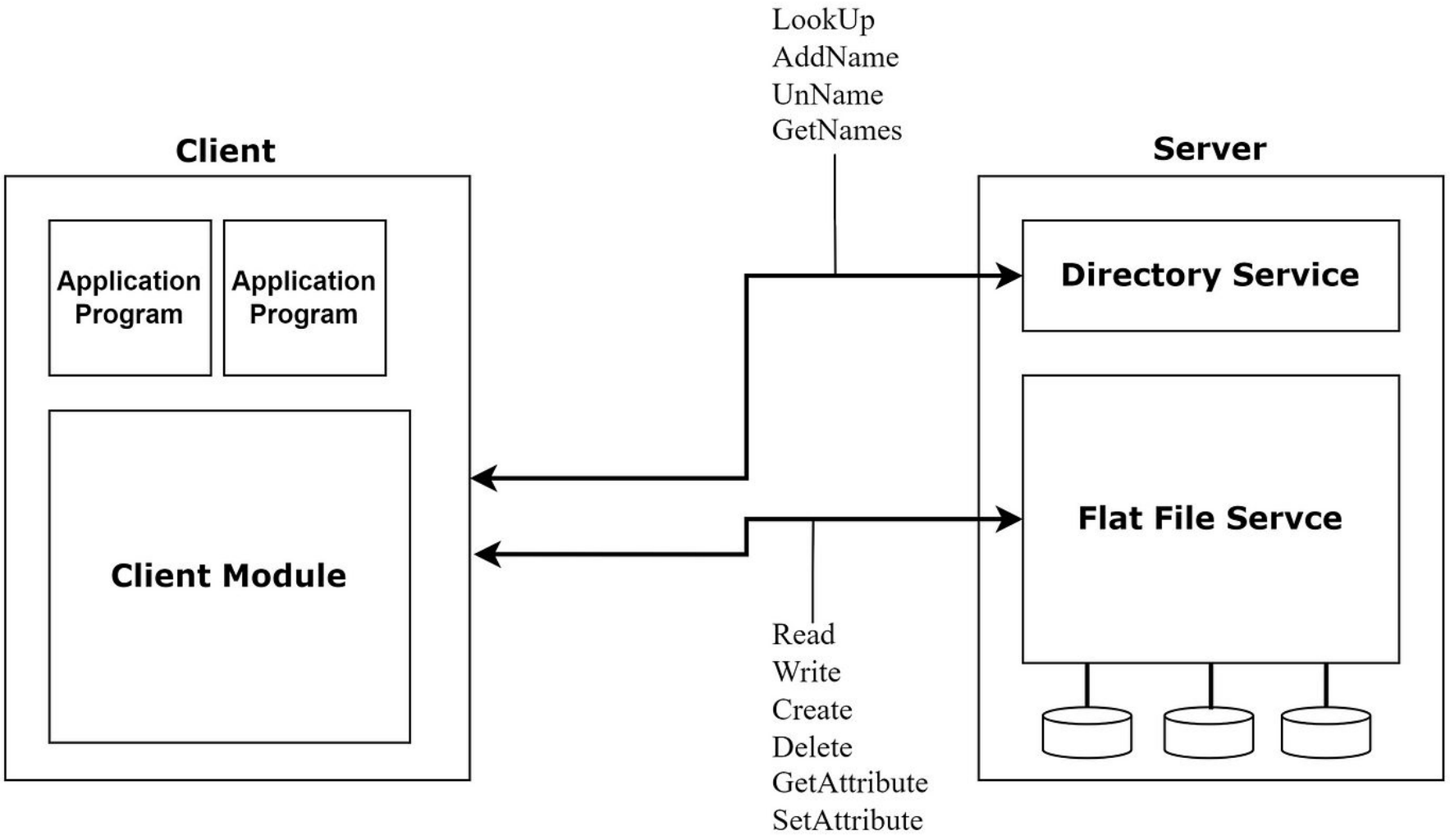
FILE SERVICE ARCHITECTURE

File Service Architecture is an architecture that provides the facility of file accessing by designing the file service as the following three components:

A client module

A flat file service

A directory service


LookUp
AddName
UnName
GetNames
Client
Server
Application Program
Application Program
Directory Service
Client Module
Flat File Servce
Read
Write
Create
Delete
GetAttribute
SetAttribute

1. **Client Module:**

   The client module executes on each computer and **delivers an integrated service (flat file and directory services) to application programs** with the help of a **single API.**

   It stores information about the network locations of flat files and directory server processes.

2. Flat file service:

- A flat file service is used to perform operations on the contents of a file.
- The Unique File Identifiers (UFIDs) are associated with each file in this service.
- For that long sequence of bits is used to uniquely identify each file among all of the available files in the distributed system.
- When a request is received by the Flat file service for the creation of a new file then it generates a new UFID and returns it to the requester.

Flat File Service Model Operations:

Read(FileId, i, n) -> Data: Reads up to n items from a file starting at item ‘i’ and returns it in Data.

Write(FileId, i, Data): Write a sequence of Data to a file, starting at item I and extending the file if necessary.

Create() -> FileId: Creates a new file with length 0 and assigns it a UFID.

Delete(FileId): The file is removed from the file store.

GetAttributes(FileId) -> Attr: Returns the attribute of file.

SetAttributes(FileId, Attr): Sets the attributes of the file.

3. Directory Service:

The directory service serves the purpose of relating file text names with their UFIDs (Unique File Identifiers).

The fetching of UFID can be made by providing the text name of the file to the directory service by the client.

The directory service provides operations for creating directories and adding new files to existing directories.

Directory Service Model Operations:

Lookup(Dir, Name) -> FileId : Returns the relevant UFID after finding the text name in the directory. Throws an exception if Name is not found in the directory.

AddName(Dir, Name, File): Adds(Name, File) to the directory and modifies the file's attribute record if Name is not in the directory. If a name already exists in the directory, an exception is thrown.

UnName(Dir, Name): If Name is in the directory, the directory entry containing Name is removed. An exception is thrown if the Name is not found in the directory.

GetNames(Dir, Pattern) -> NameSeq: Returns all the text names that match the regular expression Pattern in the directory.

SUN NETWORK FILE SYSTEM

CASE STUDY: refer site (ezexplanation.com)

NAME SERVICES

In a Distributed System, a **Naming Service** is a specific service whose aim is to:

provide a consistent and uniform naming of resources(Computers, services, remote objects, and files, as well as users) ,

thus allowing other programs or services to localize them and obtain the required metadata for interacting with them.

WHICH ONE IS EASIER TO REMEMBER ?????

74.125.237.83 or **google.com**

BENEFITS

Resource localization

Uniform naming

Device independent address (e.g., you can move domain name/web site from one server to another server seamlessly).

DNS

The domain name system (DNS) is a naming database in which internet domain names are located and translated into Internet Protocol (IP) addresses.

The domain name system maps the name people use to locate a website to the IP address that a computer uses to locate that website.

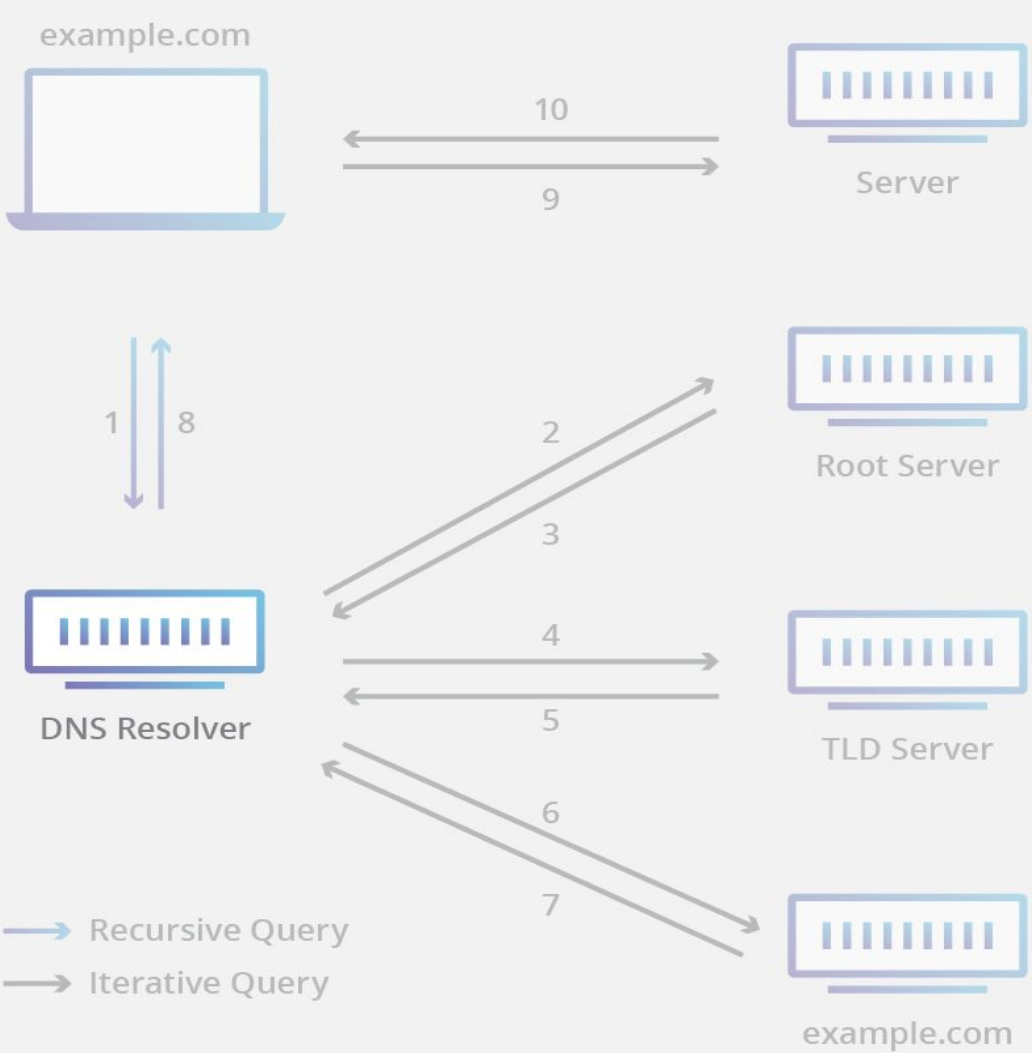
All DNS servers fall into one of four categories:

Recursive resolvers,

root name servers,

TLD nameservers, and

authoritative nameservers.


example.com
10
9
Server
1
8
2
3
Root Server
DNS Resolver
4
5
TLD Server
6
7
example.com
Recursive Query
Iterative Query

DNS RESOLVER

A recursive resolver (also known as a DNS recursor) is the first stop in a DNS query.

After receiving a DNS query from a web client, a recursive resolver will either respond with cached data, or send a request to a root nameserver, followed by another request to a TLD nameserver, and then one last request to an authoritative nameserver.

After receiving a response from the authoritative nameserver containing the requested IP address, the recursive resolver then sends a response to the client.

During this process, the recursive resolver will cache information received from authoritative nameservers.

DNS ROOT SERVER

A root server accepts a recursive resolver’s query which includes a domain name, and the root nameserver responds by directing the recursive resolver to a TLD nameserver, based on the extension of that domain (.com, .net, .org, etc.).

TLD NAMESERVER

A TLD nameserver maintains information for all the domain names that share a common domain extension, such as .com, .net, or whatever comes after the last dot in a URL.

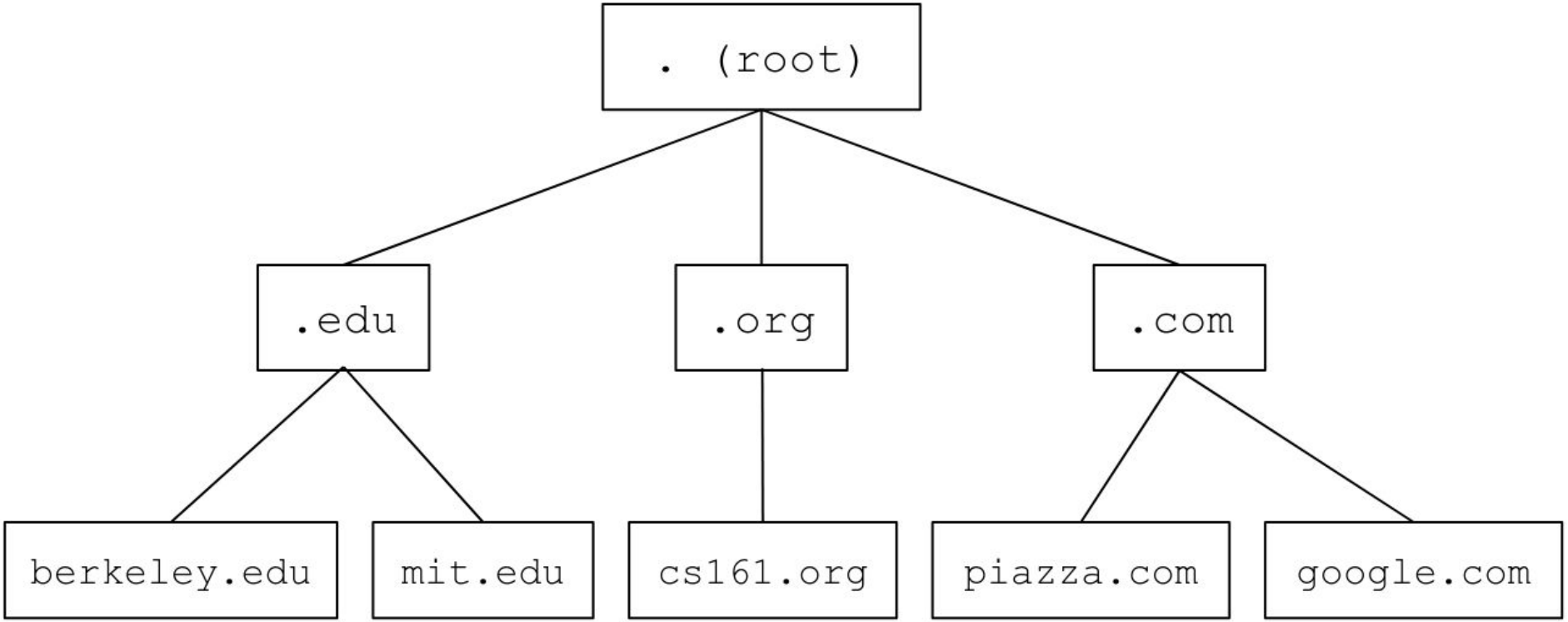
For example, a .com TLD nameserver contains information for every website that ends in ‘.com’.

If a user was searching for google.com, after receiving a response from a root nameserver, the recursive resolver would then send a query to a .com TLD nameserver, which would respond by pointing to the authoritative nameserver for that domain.

Management of TLD nameservers is handled by the **Internet Assigned Numbers Authority (IANA)**. The IANA breaks up the TLD servers into two main groups:

Generic top-level domains: These are domains that are not country specific, some of the best-known generic TLDs include .com, .org, .net, .edu, and .gov.

Country code top-level domains: These include any domains that are specific to a country or state. Examples include .uk, .us, .ru, and .jp.


. (root)
.edu
.org
.com
berkeley.edu
mit.edu
cs161.org
piazza.com
google.com

AUTHORITATIVE NAME SERVER

When a recursive resolver receives a response from a TLD nameserver, that response will direct the resolver to an authoritative nameserver.

The authoritative nameserver is usually the resolver’s last step in the journey for an IP address.

The authoritative nameserver contains information specific to the **domain name** it serves (e.g. google.com) and it can provide a recursive resolver with the IP address of that server

DNS records

DNS record types are records that provide important information about a hostname or domain. These records include the current IP address for a domain.

Also, DNS records are stored in text files (zone files) on the authoritative DNS server. The content of a DNS record file is a string with special commands that the DNS server understands.

The following are the five major DNS record types:

- A record
- AAAA record
- CNAME record
- Nameserver (NS) record
- Mail exchange (MX) record

1. A record

The A record is the most important DNS record type. The "A" in A record stands for "address."

An A record shows the IP address for a specific hostname or domain. For example, a DNS record lookup for the domain example.com returns the following result:

Type	Domain Name	IP Address	TTL
A	example.com	93.184.216.34 Verizon Business (AS15133)	24 hrs

2. AAAA record

AAAA record, just like A record, point to the IP address for a domain.

However, this DNS record type is different in the sense that it points to IPV6 addresses.

IPV6 is an upgrade over IPV4 as it offers more IP addresses.

As a result, IPV6 solves the issue of running out of unique IP addresses.

3. CNAME record

CNAME—or, in full, "canonical name"—is a DNS record that points a domain name (an alias) to another domain.

For example, the subdomain ng.example.com can point to example.com using CNAME.

4. NS record

NS(Nameserver) record helps point to where internet applications like a web browser can find the IP address for a domain name.

5. MX record

A mail exchange (MX) record, is a DNS record type that shows where emails for a domain should be routed to. In other words, an MX record makes it possible to direct emails to a mail server.

Name	Type	Priority	RDATA
@	MX	10	mx.zoho.com
@	MX	20	mx2.zoho.com

Directory and Discovery Services

Directory:

A directory, in computing, is a specialized database optimized for reading, searching, and often updating.

It organizes and provides access to information in a hierarchical or logical structure.

Directories can contain various types of data, including user profiles, system configurations, network resources, and more.

Directories are commonly used for authentication, authorization, and information lookup.

Directory Services:

Directory services refer to the software systems or protocols that manage access to directories.

They provide a centralized and standardized way for applications and users to access directory information.

Directory services typically offer features such as authentication, authorization, and access control.

One of the most well-known directory services is the **Lightweight Directory Access Protocol (LDAP)**, which is widely used for accessing and managing directory information.

Other examples include Active Directory (used in Windows environments) and OpenLDAP (an open-source implementation of LDAP).