# Unit-3(6hrs)

PROCESS

# Content

## Process

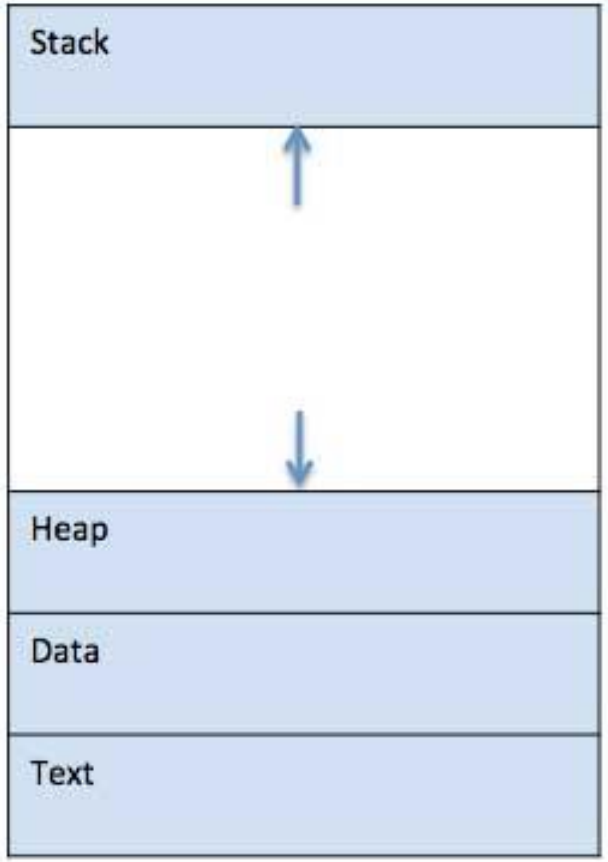
1. Threads
2. Virtualization
3. Clients
4. Servers
5. Code Migration

# Process

A process is basically a program in execution. The execution of a process must progress in a sequential fashion.

Example: we write our computer programs in a text file and when we execute this program, it becomes a process which performs all the tasks mentioned in the program.

When a program is loaded into the memory and it becomes a process, it can be divided into four sections – stack, heap, text and data. The following image shows a simplified layout of a process inside main memory −



# Process cont..

| S. N. | Component & Description |
|---|---|
| 1 | **Stack**<br>The process Stack contains the temporary data such as method/function parameters, return address and local variables. |
| 2 | **Heap**<br>This is dynamically allocated memory to a process during its run time. |
| 3 | **Text**<br>This includes the current activity represented by the value of Program Counter and the contents of the processor's registers. |
| 4 | **Data**<br>This section contains the global and static variables. |

# Process Life Cycle

When a process runs, it goes through many states. Distinct operating systems have different stages, and the names of these states are not standardized. In general, a process can be in one of the five states listed below at any given time.

**Start**

When a process is started/created first, it is in this state.

**Ready**

Here, the process is waiting for a processor to be assigned to it. Ready processes are waiting for the operating system to assign them a processor so that they can run. The process may enter this state after starting or while running, but the scheduler may interrupt it to assign the CPU to another process.

# Process Life Cycle cont..

**Running**

When the OS scheduler assigns a processor to a process, the process state gets set to running, and the processor executes the process instructions.

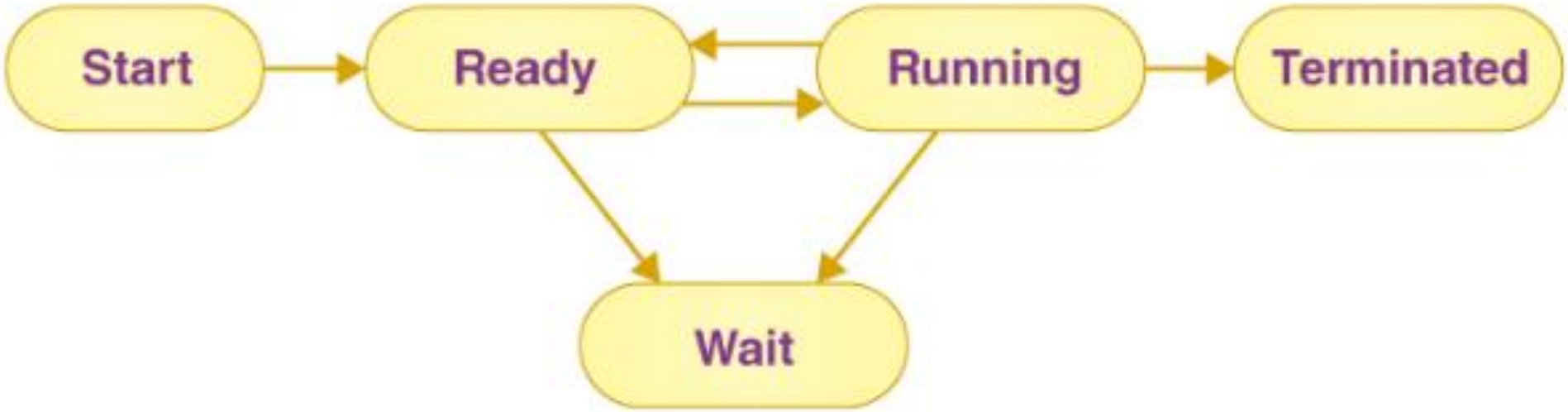
**Waiting**

If a process needs to wait for any resource, such as for user input or for a file to become available, it enters the waiting state.

**Terminated or Exit**

The process is relocated to the terminated state, where it waits for removal from the main memory once it has completed its execution or been terminated by the operating system.

# Process Life Cycle cont..



# Threads

Threads are a fundamental concept in modern computing, enabling concurrent execution within a single process. A thread, sometimes called a "lightweight process," is the smallest unit of execution that can be scheduled and managed by an operating system.

**Key Characteristics of Threads**

**Shared Resources**: Threads within the same process share:
Code section
Data section
Open files
Other resources like global variables

**Individual Resources**: Each thread has its own:
Program Counter (PC): Keeps track of the next instruction to execute.
Stack: Contains the thread's local variables and function call information.
Registers: Hold the thread’s state.

# Threads cont..

**Types of Threads**

**User Threads**: Managed and scheduled by a user-level library rather than the operating system. User threads are efficient but lack the ability to leverage multiple processors since the kernel only sees one process.

**Kernel Threads**: Managed and scheduled directly by the operating system. These threads can take full advantage of multiple processors but are more resource-intensive to manage.

**Thread Lifecycle**

**New**: Thread is created but not yet started.

**Runnable**: Thread is ready to run and waiting for CPU time.

**Running**: Thread is currently executing.

**Blocked/Waiting**: Thread is waiting for an event (e.g., I/O operation completion).

**Terminated**: Thread has finished execution.

# Threads cont..

**Types of threads per process**

**Single thread process:**

In a single-threaded environment, an application or process contains only one thread of execution. This means that tasks are executed sequentially, one after another, without overlap.

**Deterministic Behavior**: Execution order is predictable, which simplifies debugging and testing.

**No Context Switching Overhead**: There's no need to switch between threads, avoiding the overhead associated with context switching.

**Limited Utilization of Multi-Core Processors**: Cannot fully utilize the capabilities of multi-core processors since only one core can be active at a time.

Example Many command-line tools and simple scripts are single-threaded.

# Threads Cont..

**Multi-Threaded Environments**

In a multi-threaded environment, an application or process can contain multiple threads of execution, allowing concurrent execution of tasks.

**Characteristics of Multi-Threaded Environments:**

**Responsiveness:** One thread may provide rapid response while other threads are blocked or slowed down.

**Concurrent Execution**: Multiple threads can run at the same time, improving performance, especially on multi-core processors.

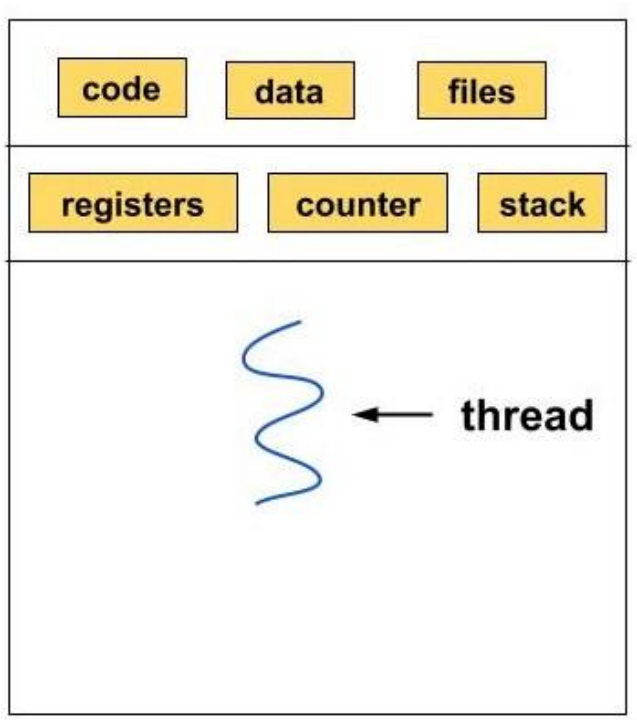
**Complex Design**: Requires careful design to handle synchronization and avoid issues such as race conditions and deadlocks.

**Context Switching Overhead**: There is overhead due to switching between threads, although this is generally less than switching between processes.

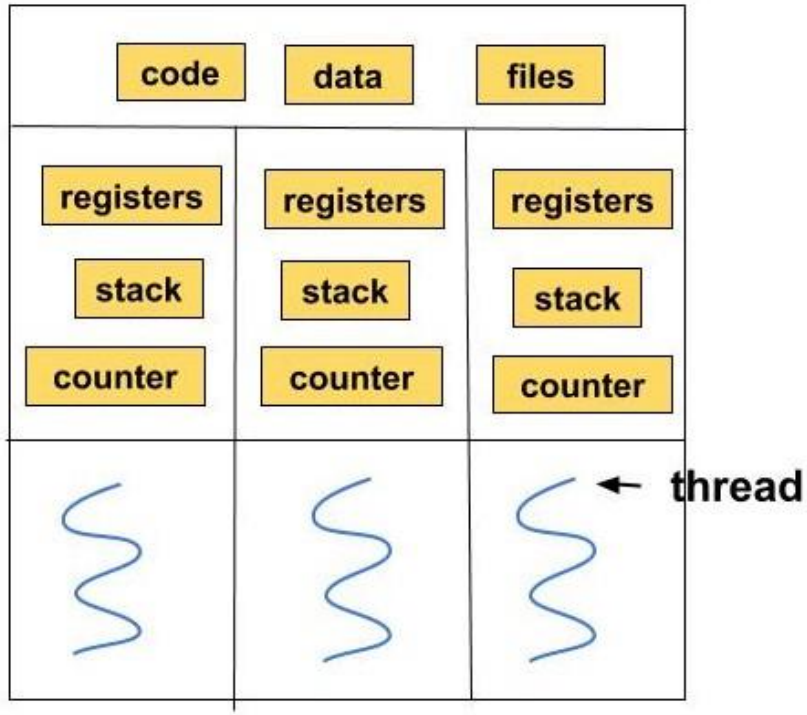
**Efficient Resource Utilization**: Better utilization of system resources, particularly CPU cores.

Example Web servers, interactive GUI applications, and complex simulations often use multi-threading.

# Threads cont..




Single-threaded process




Multi-threaded process

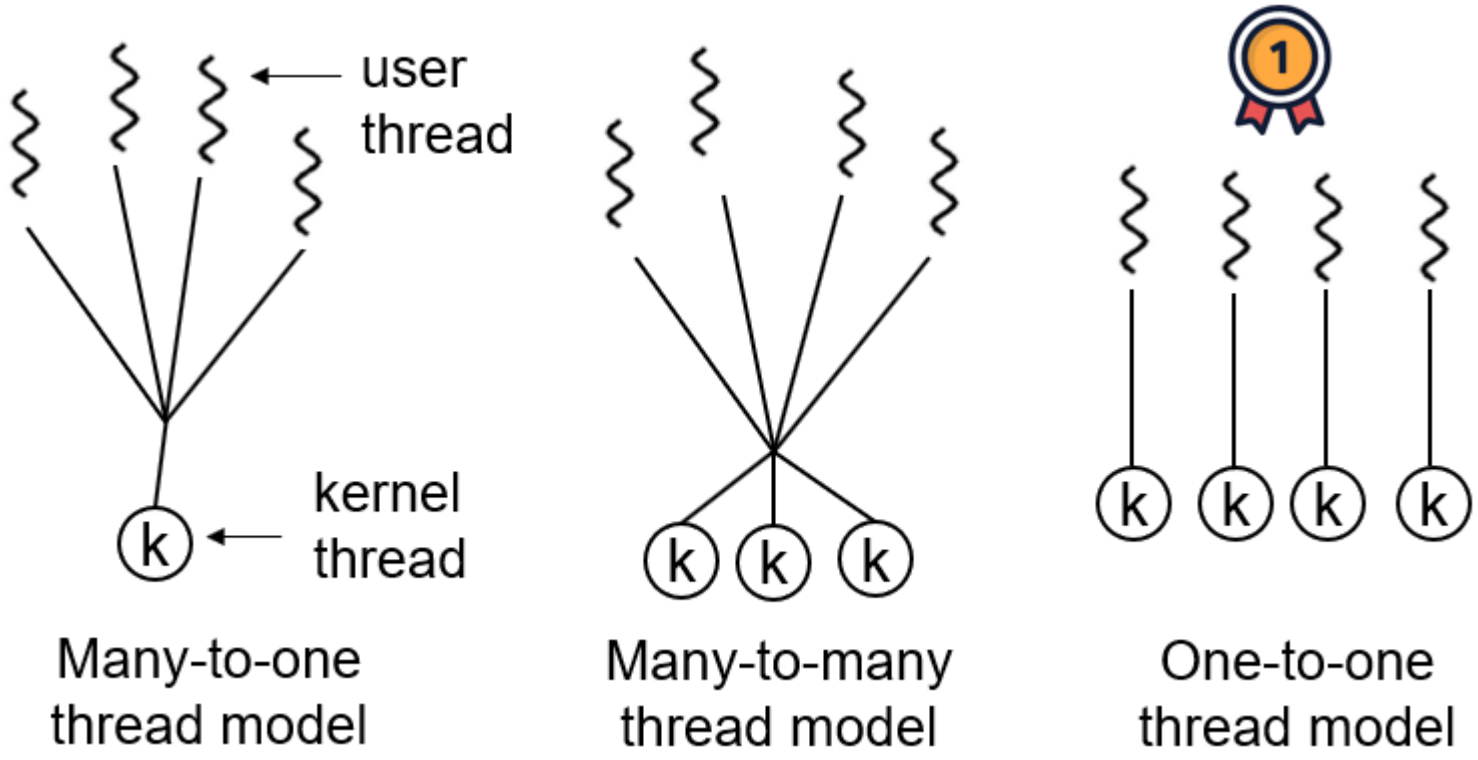
# Thread Models

**Many-to-One Model**: Multiple user-level threads are mapped to a single kernel thread. This model is simple but cannot run threads in parallel on multiple processors.

**One-to-One Model**: Each user-level thread maps to a separate kernel thread. This model allows true concurrent execution on multiprocessors but can be resource-intensive.

**Many-to-Many Model**: Multiple user-level threads are mapped to an equal or smaller number of kernel threads. This model strikes a balance between efficiency and concurrency.

# Thread Models




Many-to-one thread model

Many-to-many thread model

One-to-one thread model

# Difference between threads and process

| Aspect | Process | Threads |
|---|---|---|
| Definition | An independent program running on a computer. | A smaller unit of execution within a process. |
| Memory and resources | Each process has its own separate memory space | Threads within the same process share the same memory space. |
| Creation and Management | Creating a new process takes more time and resources. | Creating a new thread is faster and uses fewer resources. |
| Communication | Communicating between processes is more complex and slower. | Communication between threads is easier and faster since they share memory. |
| Overhead | Higher overhead due to separate memory space and resources. | Lower overhead as threads share resources and memory space. |
| Example | Running a web browser and a text editor simultaneously. | In a web browser , one thread handles reading while another handles user input. |

# Virtualization

Virtualization is a technology that allows you to create and manage virtual versions of hardware, software, storage, and networking resources.

It enables you to run multiple operating systems and applications on a single physical machine, known as a host, by abstracting and simulating the underlying physical hardware. Software called a hypervisor connects directly to that hardware and allows you to split 1 system into separate, distinct, and secure environments known as Virtual machines(VMs).

These virtual machines (VMs) depend on software called a hypervisor, which divides the computer's resources, like CPU and memory, from the physical hardware and allocates them correctly to each virtual machine

# Virtualization..

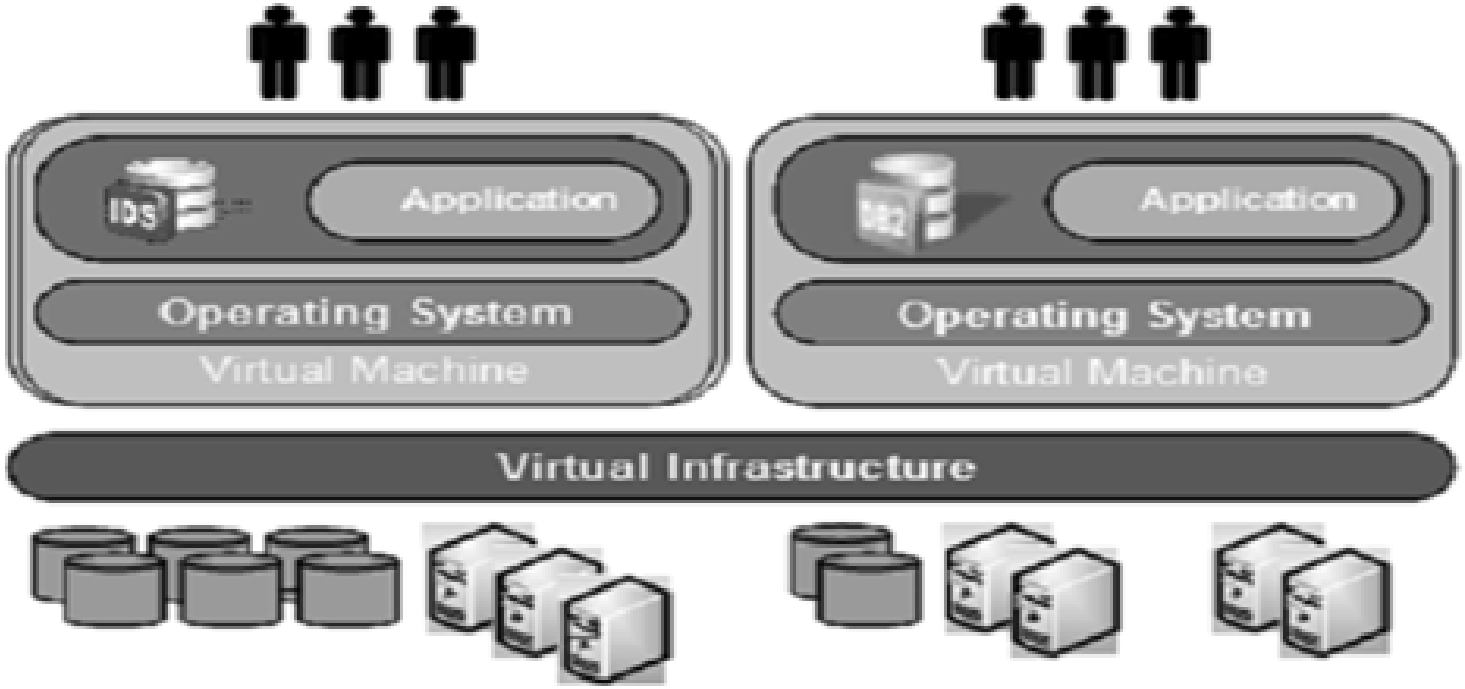



**Fig1: Concept of Virtualization**

# Types of virtualization

- Desktop Virtualization
- Application Virtualization
- Server Virtualization
- Network Virtualization
- Storage Virtualization

# Desktop virtualization

Desktop virtualization is a software technology that separates the desktop environment and associated application software from the physical client device that is used to access it.

Desktop virtualization refers to the process of creating a virtualized environment for desktop computing. This involves running desktop operating systems and applications in virtual machines (VMs) hosted on a central server or cloud infrastructure, which can then be accessed remotely by end users through various devices, such as thin clients, laptops, tablets, or smartphones.

**Benefits of Desktop Virtualization**

**Cost Savings**:

Reduced hardware costs as end users can use thin clients or older PCs to access powerful virtual desktops.

Lower maintenance costs and better resource utiliz

# Cont..

**Security**:

Centralized management and security policies reduce the risk of data breaches.

Data resides on the server, not on the end user's device, minimizing the risk of data loss from theft or hardware failure.

**Flexibility and Mobility**:

Users can access their desktop environment from any device and location, enhancing mobility and remote work capabilities.

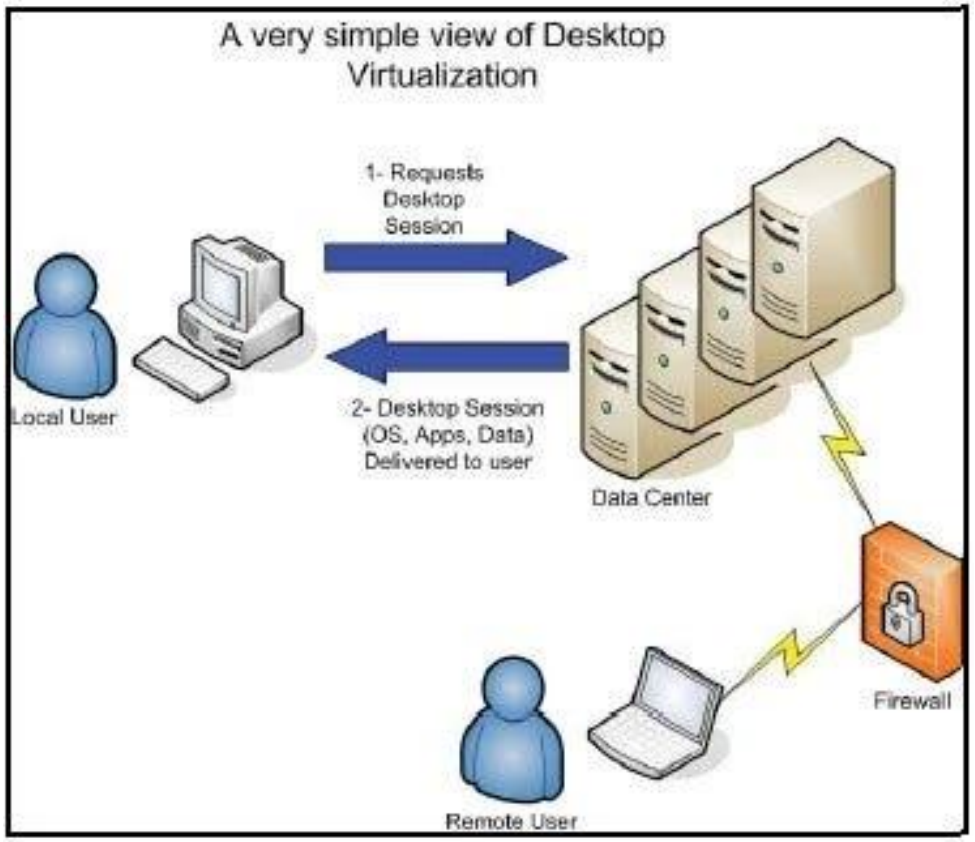
Simplified deployment and updates as changes are made centrally and propagated to all users.

**Scalability**:

Easy to scale up or down based on organizational needs.

Quickly provision new desktops for new employees or temporary workers.

# Cont..


A very simple view of Desktop Virtualization
1- Requests Desktop Session
2- Desktop Session (OS, Apps, Data) Delivered to user
Local User
Data Center
Firewall
Remote User

# Application Virtualization

Application virtualization is a technology that allows applications to be run in environments separate from the underlying operating system on which they are deployed. This separation enables applications to be executed without being installed directly on the end-user device, thus avoiding conflicts with other software and simplifying management.

**Isolation**:

This isolation prevents conflicts between applications and between applications and the operating system.

**Streaming**:

Applications can be streamed on-demand to the user's device, starting to run before the entire application is downloaded.

This reduces waiting times and allows for faster access to applications.

# Application Virtualization cot..

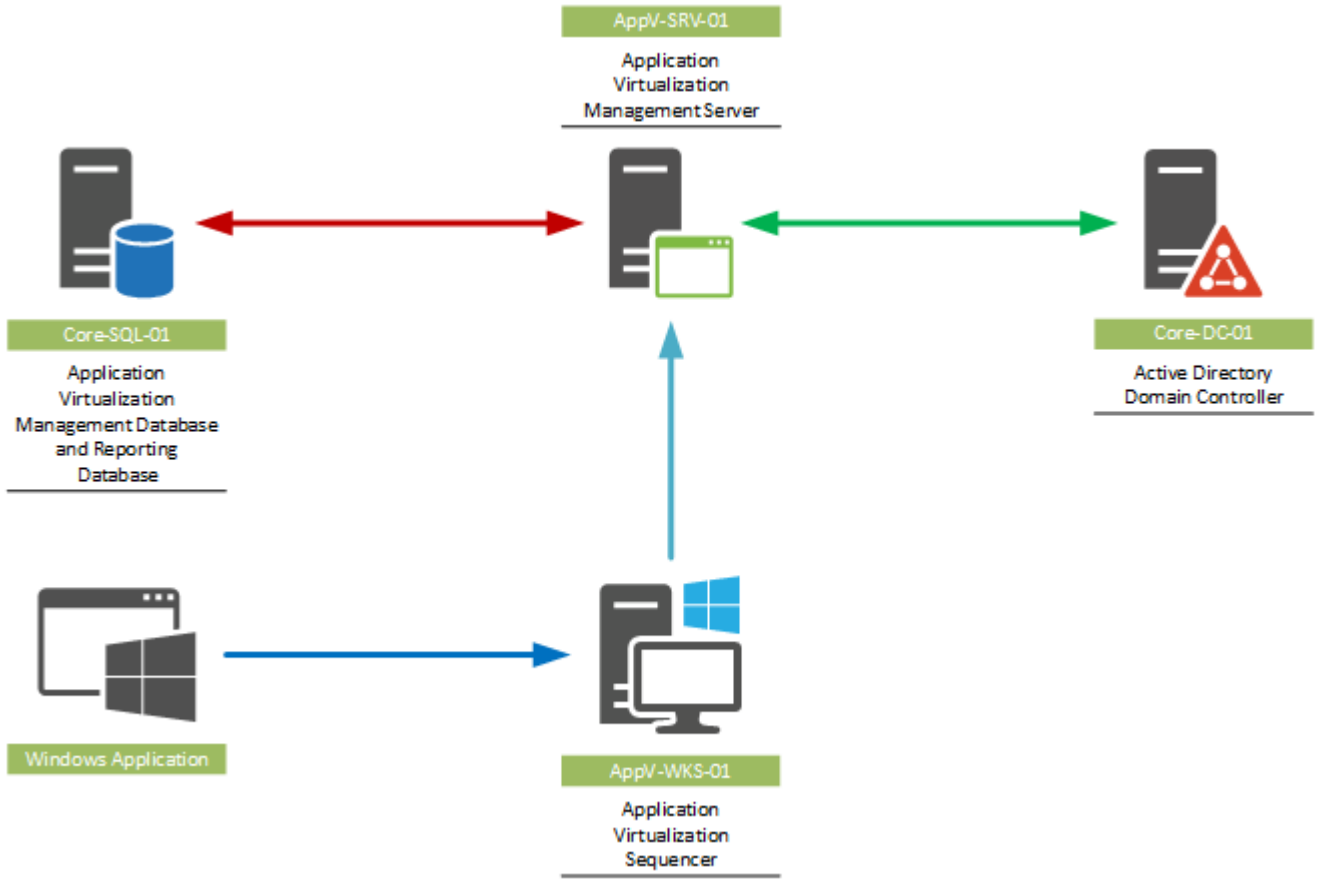
**Centralized Management**:

Applications are managed and updated centrally, simplifying the deployment and maintenance process.

**User Experience**:

Users interact with virtualized applications as if they were installed locally, without noticeable differences in functionality or performance

# Cont..


AppV-SRV-01
Application Virtualization Management Server
Core-SQL-01
Application Virtualization Management Database and Reporting Database
Core-DC-01
Active Directory Domain Controller
Windows Application
AppV-WKS-01
Application Virtualization Sequencer

# Server Virtualization

Server virtualization is a technology that allows multiple virtual servers to run on a single physical server. Each virtual server operates independently, with its own operating system and applications, creating an environment where multiple workloads can be consolidated onto fewer physical machines. This approach maximizes resource utilization, enhances flexibility, and reduces costs.

Improved server reliability and availability,

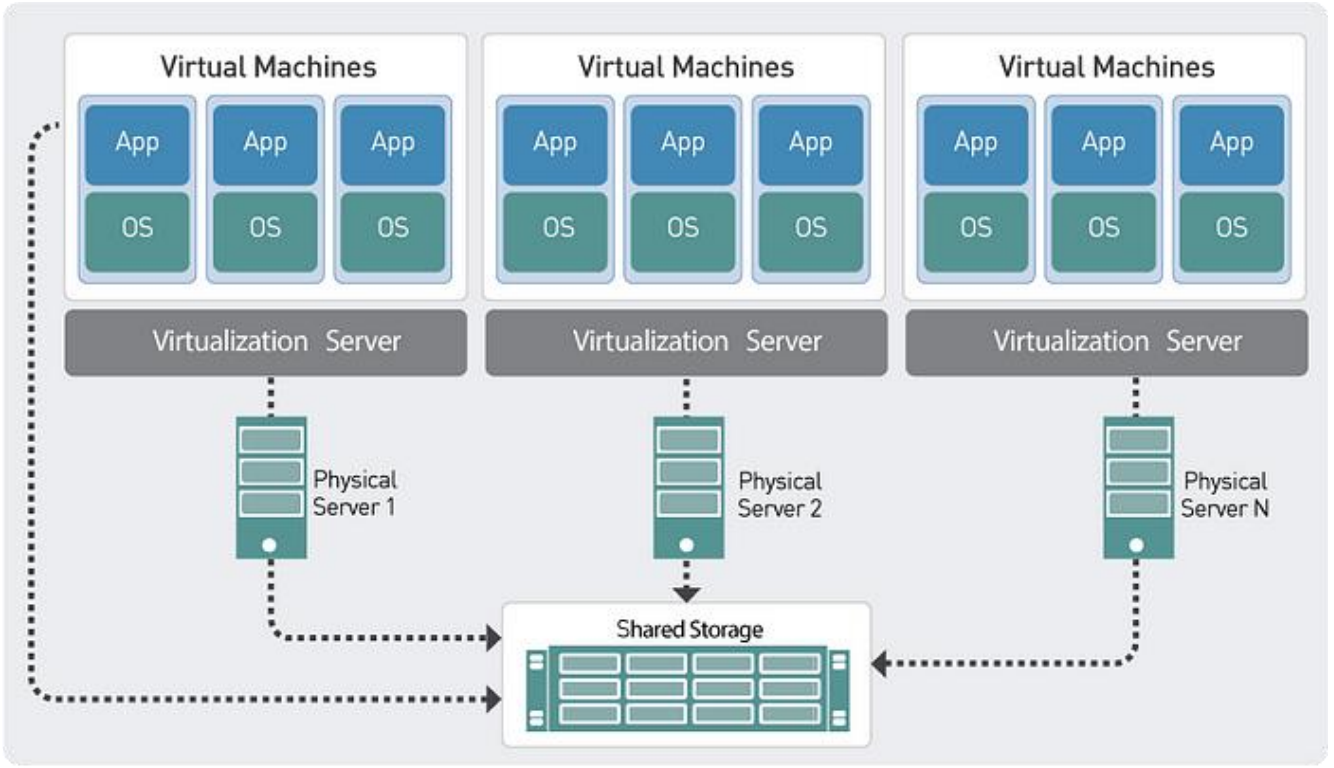
Lower total operational cost.

More efficient utilization of physical servers.

More efficient utilization of power.

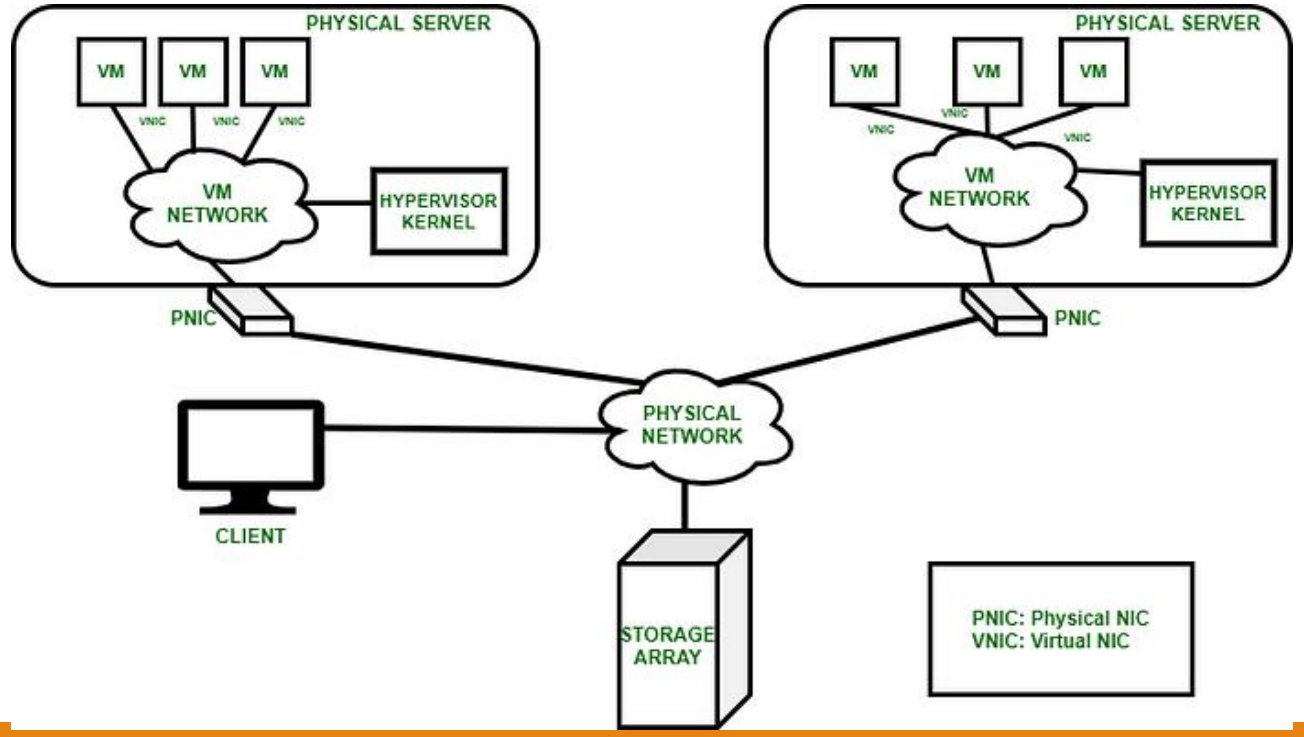
Virtual machine creation: create virtual machine to customer's specifications for memory, CPU reservation, disk space and supported OS.

# Server Virtualization cot..



# Network Virtualization

Network virtualization allows you to create multiple virtual networks on a single physical network. This can improve security and isolation between different applications and services.

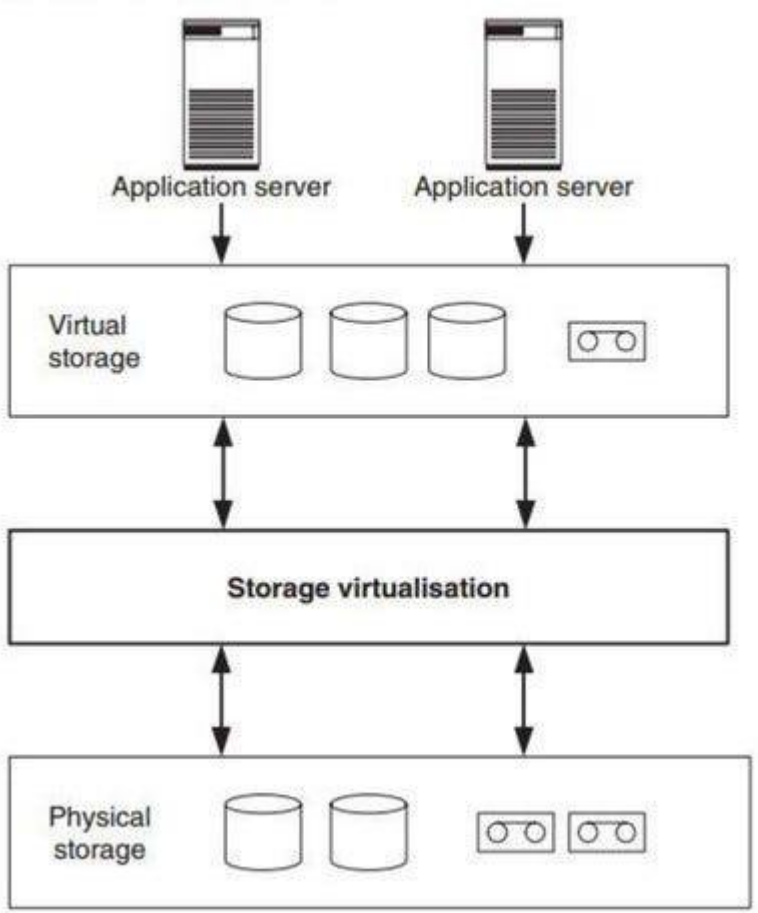


# Storage virtualization.

Storage virtualization abstracts physical storage devices into a unified pool of storage resources. It simplifies storage management, improves flexibility, and can enhance data protection and disaster recovery. Separation of the storage into the physical implementation level and the logical representation level for use by OS, applications and users. At various levels Server, Storage and Network Add an additional layer between storage devices and storage users

storage virtualization organizes all the data stored on different devices into one big virtual container. This container is like a neat storage closet where you can easily find and manage your stuff. It separates the physical storage devices (like hard drives) from the way your computer or applications see and use the storage. So, even if you have data spread across different devices, it looks like it's all in one place to your computer.

# Cont..


Application server
Application server
Virtual storage
Storage virtualisation
Physical storage

# Advantage of Virtualization

**Resource Utilization**: Virtualization allows for better utilization of hardware resources by running multiple virtual machines (VMs) on a single physical server. This optimizes CPU, memory, and storage usage, leading to cost savings and energy efficiency

**Flexibility and Scalability**: Virtualization provides the flexibility to quickly scale up or down based on changing business needs. It enables rapid deployment of new virtual machines, applications, and services without the need for additional physical hardware

**Cost Savings**: By consolidating multiple workloads onto fewer physical servers, virtualization reduces hardware and operational costs. It minimizes the need for purchasing and maintaining multiple physical machines, leading to significant savings in terms of hardware, space, and power consumption.

**Improved Disaster Recovery and High Availability**: Virtualization enables efficient disaster recovery solutions by replicating virtual machines and data across multiple physical hosts or storage systems. In case of hardware failure or disaster, virtual machines can be quickly migrated or restarted on other hosts, minimizing downtime and ensuring business continuity

# Advantage of Virtualization

**Enhanced Management and Automation**: Virtualization centralizes management and simplifies administration tasks through automation. IT administrators can easily provision, configure, monitor, and manage virtual machines using centralized management tools, reducing manual efforts and operational complexity.

**Testing and Development**: Virtualization provides a cost-effective and efficient platform for testing and development environments. Developers can quickly spin up virtual machines to test new applications, configurations, or software updates without impacting production systems.

**Legacy System Support**: Virtualization enables organizations to run legacy applications and operating systems on modern hardware. By virtualizing older systems, businesses can extend the lifespan of legacy applications and avoid costly hardware upgrades or migrations.

# Disadvantage of Virtualization

**Performance Overhead**: Virtualization introduces a layer of abstraction between the hardware and the virtual machines (VMs), which can lead to a slight performance overhead compared to running directly on physical hardware. This overhead may be more noticeable in high-performance computing environments or real-time applications

**Complexity**: Managing virtualized environments can be complex, especially for organizations with limited IT resources or expertise. Configuring, monitoring, and troubleshooting virtual machines, hypervisors, and virtual networks require specialized skills and knowledge.

**Licensing and Compliance**: Virtualization may introduce complexities related to software licensing and compliance. Organizations need to ensure compliance with vendor licensing agreements, especially when running virtual machines across multiple hosts or migrating workloads between hosts.

**Single Point of Failure**: While virtualization offers features like high availability and live migration to improve system reliability, the hypervisor itself becomes a single point of failure. If the hypervisor fails, all virtual machines running on it may be affected, leading to downtime and potential data loss.

**Hardware dependency:**

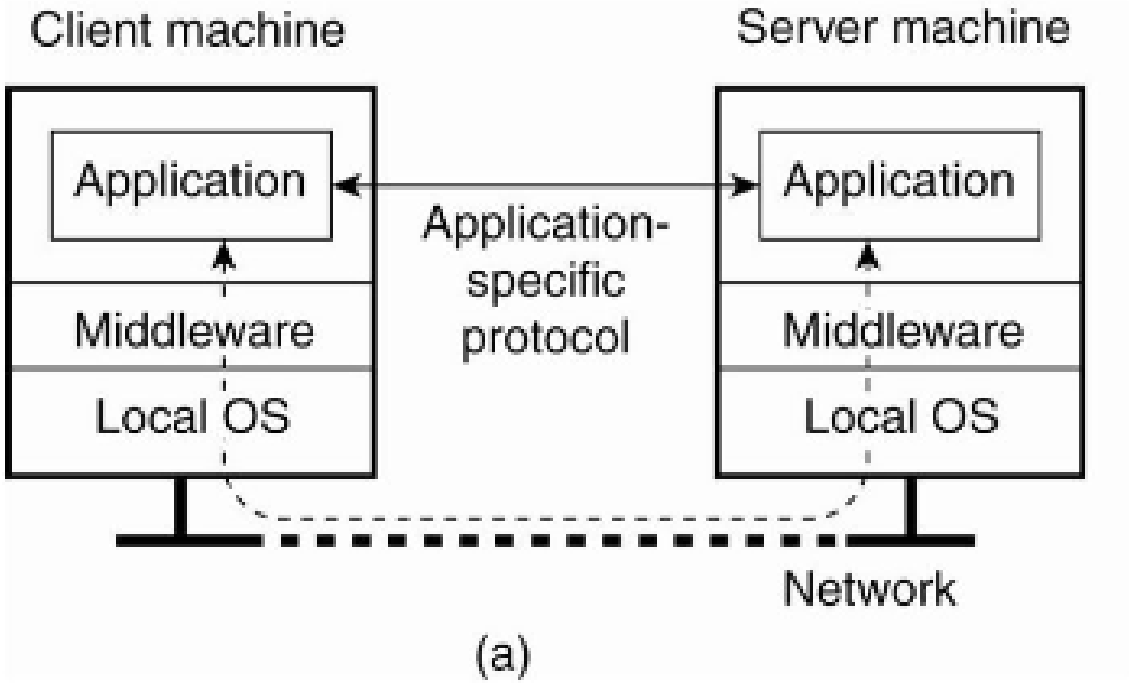
# Client

In the previous chapters we discussed the client server model, the roles of clients and servers, and the ways they interact. In this chapter we are going to discussed about the anatomy of client and server respectively.

**Network User interface:**

- A major task of client machines is to provide the means for users to interact with remote servers.
- A major task of client machine is to provide the means for users to interact with remote servers.
- There are two ways in which this interaction can be supported .
- **First** for each remote service the client machine will have a separate counterpart that can contact the service over the network. A typical example is a calendar running on a users smartphone that needs to synchronize with a remote, possibly shared calendar. In this case an application-level protocol will handle the synchronization as shown in figure.

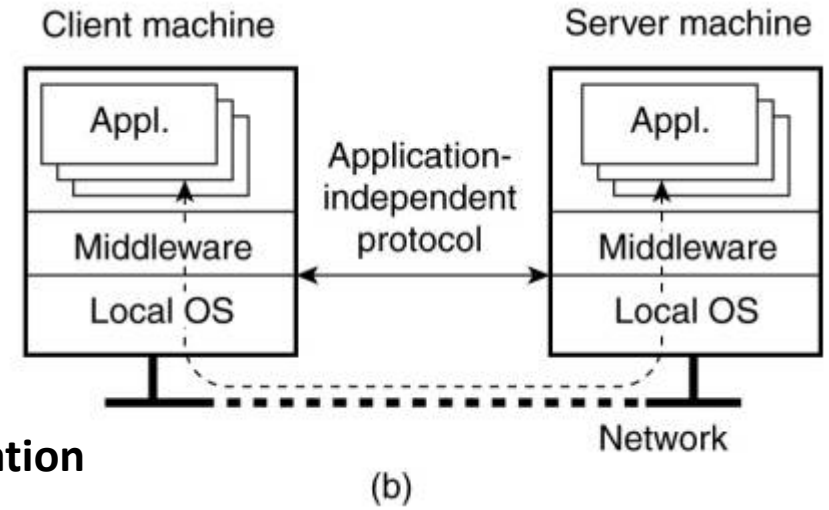
# Cont..




**A networked application with its own protocol**

# Cont..

- A **Second solution** is to provide direct access to remote services by offering only a convenient user interface. Effectively, this means that the client machine is used only as a terminal with no need for local storage leading to an application-neutral solution in figure below.
- In the case of networked user interface , everything is processed and stored at the server. This thin client approach has received much attention with the increase of Internet connectivity and the use of mobile devices. Thin client solution are also popular as they ease the task of system management.




**A general solution to allow access to remote application**

# Client-Side Software for Distribution Transparency

- Client software comprises more than just user interfaces. In many cases, parts of the processing and data level in a client-server application are executed on the client side as well. A special class is formed by embedded client software, such as for automatic teller machines (ATMs), cash registers, barcode readers, TV set-top boxes, etc. In these cases, the user interface is a relatively small part of the client software, in contrast to the local processing and communication facilities.

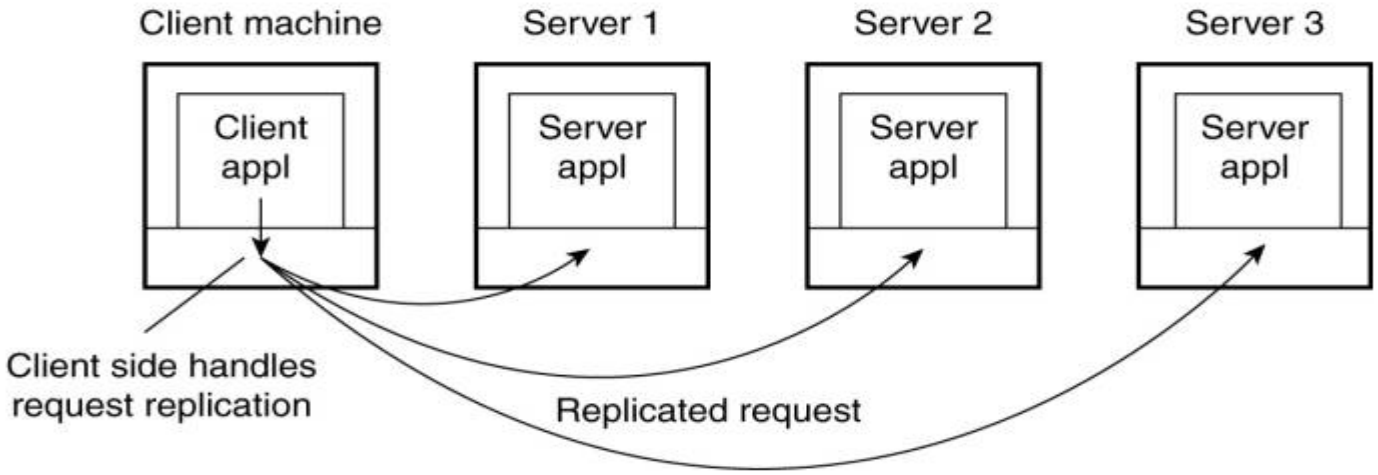
- Besides the user interface and other application-related software, client software comprises components for achieving distribution transparency. Ideally, a client should not be aware that it is communicating with remote processes. In contrast, distribution is often less transparent to servers for reasons of performance and correctness. For example, in Chap. 6 we will show that replicated servers sometimes need to communicate in order to establish that operations are performed in a specific order at each replica.

- Access transparency is generally handled through the generation of a client stub from an interface definition of what the server has to offer. The stub provides the same interface as available at the server, but hides the possible differences in machine architectures, as well as the actual communication.

# cont…

- There are different ways to handle location, migration, and relocation transparency.
- many distributed systems implement replication transparency by means of client-side solutions. For example, imagine a distributed system with replicated servers, Such replication can be achieved by forwarding a request to each replica, as shown in Fig. 3-10. Client-side software can transparently collect all responses and pass a single response to the client application.

Figure 3-10. Transparent replication of a server using a client-side solution.



# Cont..

Finally, consider failure transparency. Masking communication failures with a server is typically done through client middleware. For example, client middle-ware can be configured to repeatedly attempt to connect to a server, or perhaps try another server after several attempts. There are even situations in which the client middleware returns data it had cached during a previous session, as is sometimes done by Web browsers that fail to connect to a server.

Concurrency transparency can be handled through special intermediate servers, notably transaction monitors, and requires less support from client software. Likewise, persistence transparency is often completely handled at the server.

# Servers

A server is a piece of computer hardware or software that provides functionality for other programs or devices, called clients. This architecture is called client server model.

**General server design issue in distributed system**

A server is a process implementing a specific service on behalf of a collection of clients. In essence, each server is organized in the same way: it waits for an incoming request from a client and subsequently ensures that the request is taken care of, after which it waits for the next incoming request.

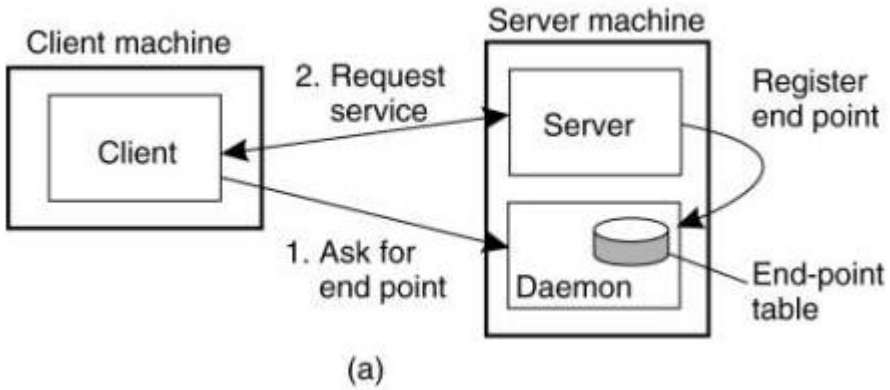
1. **Concurrent versus iterative servers:** There are two types of server choices:

**Iterative Server:** It is a single threaded server which processes the requests in a queue. While processing the current request it adds incoming request in a wait queue. Essentially, request are not executed in parallel at any time on the server.

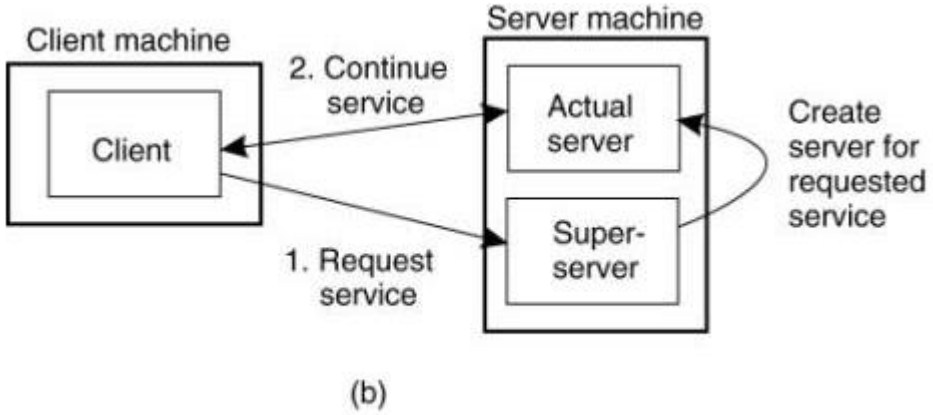
# Servers cont..

- **Concurrent server:** In this case when a new request comes in the server provides a new thread to service the request. Thus all process are executed in parallel in this scenario. This the thread pre-request model.

## 2. Contacting the server: end points




Client to server binding using Daemon




Client to server binding using a super-server

# Servers cont..

3. **Interrupting a server**
4. **Stateful servers stateless servers**

**Stateful servers:** are those , which keep a state of their connected clients. For example push server are stateful servers since the server need to know the list of clients it needs to be send.

**Stateless servers:** on the other hand don't keep any information on the state of its connected clients, and can change its own state without having to inform any client. In this case client will keep its own session information. Pull server are the example of stateless server where the client sends the request to the servers as and when required.

**Soft state servers:** maintain the state of the client for a limited period of time on the server and when the session timeouts it discards the information.

# Object Server

An object server is a type of server specifically designed to support and manage distributed objects.

Some of the key points of object server:

1. **General Purpose**: Unlike traditional servers, which are built to provide specific services (like a web server delivering web pages), an object server doesn't offer a specific service on its own.
2. **Hosting Objects:** The object server hosts various objects. These objects are pieces of code that provide specific services or functionalities.
3. **Service Implementation by Objects**:The services offered by the server are actually implemented by the objects that reside within it. Each object can perform different tasks or provide different services.

# Cont..

4. **Invocation of Local Objects**:When a remote client (a user or another system) makes a request, the object server processes this request by invoking (calling) the appropriate local object to perform the required task.
5. **Flexibility**:Since the services are tied to the objects, you can easily change what services the server offers by simply adding new objects or removing existing ones. This makes the server very flexible and adaptable

**Activation Policies:**

**Activation Policies**: These are the rules or methods used to decide how and when to activate (bring into the server's active memory) an object so it can perform its tasks. Think of it like turning on a machine only when you need to use it.

# Cont..

**Object Adapters (or Object Wrappers)**

**Object Adapter**: This is a tool or software component that helps manage the activation policies. It decides how to handle objects based on the given rules.

**Purpose of Object Adapter**: It organizes and manages objects according to the activation policies. You can think of it as a manager that makes sure the right objects are ready to work when needed.

**How It Works**

**Grouping Objects**:

Objects are grouped based on the activation policies. For example, some objects might need to be always ready, while others are activated only when a request comes in.

# Cont..

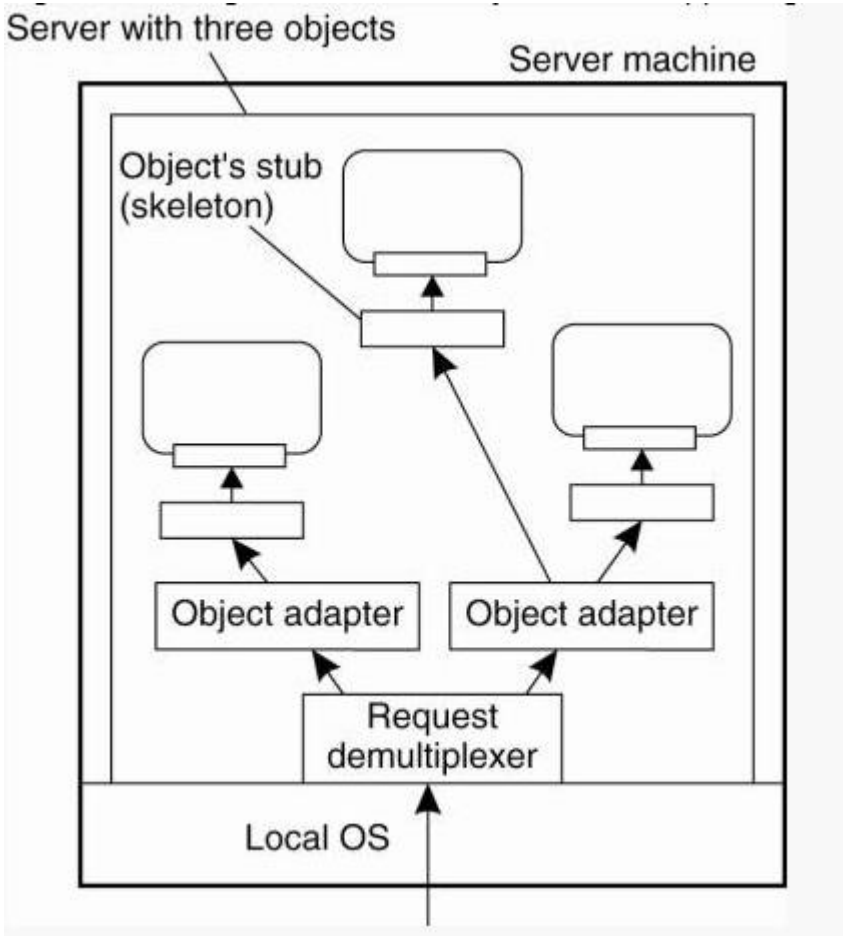
**Activation**:

When a client requests a service from the server, the object adapter ensures the correct object is activated (loaded into memory) to handle the request.

**Configuration**:

The object adapter itself is generic, meaning it can be used for different activation policies. Developers only need to configure it to fit the specific rules or needs of their application.

# Cont..


Server with three objects
Server machine
Object's stub
(skeleton)
Object adapter
Object adapter
Request
demultiplexer
Local OS

# Code Migration

Code migration in a distributed system refers to the process of moving code or programs from one location (node) to another within a distributed network. This concept is crucial for load balancing, resource optimization, and improving system performance. There are different types of code migration, each serving different purposes.

Types of Code Migration

**Process Migration**:

Moving a whole running program (with all its data) from one computer to another.

Imagine moving a game you are playing from one computer to another without stopping the game.

**Strong Mobility**: The program and its running state (like current data and what it's doing) move together.Example: Moving a video call from your laptop to your phone without interruption.

# Cont..

**Why is Code Migration Useful?**

**Balancing Work**:If one computer is too busy, we can move some tasks to a less busy computer.

**Better Performance**:Moving tasks closer to where the data is can make things run faster.

**Handling Failures**:If a computer is about to fail, we can move tasks to another computer to keep everything running smoothly.

**Efficient Use of Resources**:Using the idle capacity of computers to perform tasks efficiently.

**Example Scenario**

**Load Balancing Web Servers**:

If one web server is handling too many visitors, we can direct new visitors to another less busy server.

# *Thank you*