

Distributed Systems

Processes
Chapter 3

Content

3.1 Threads

3.2 Virtualization


3.3 Clients

3.4 Servers

3.5 Code Migration


Process

- A Program does nothing unless its instructions are executed by a CPU.
- A program in execution is called a process.
- In order to accomplish its task, process needs the computer resources.
- There may exist more than one process in the system which may require the same resource at the same time.

- 
- Some resources may need to be executed by one process at one time to maintain the consistency otherwise the system can become inconsistent and deadlock may occur.

Child Process

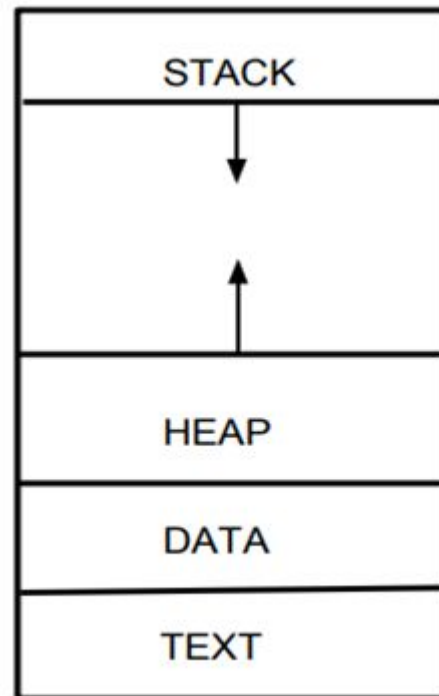
- A process can initiate a sub-process , which is called child process.
- A child process is replica of parent process and share resources of parent process , but cannot exist if parent is terminated.



Consider the scenario where you are working in a Microsoft Word document and you want to insert a chart or a spreadsheet created in Microsoft Excel.

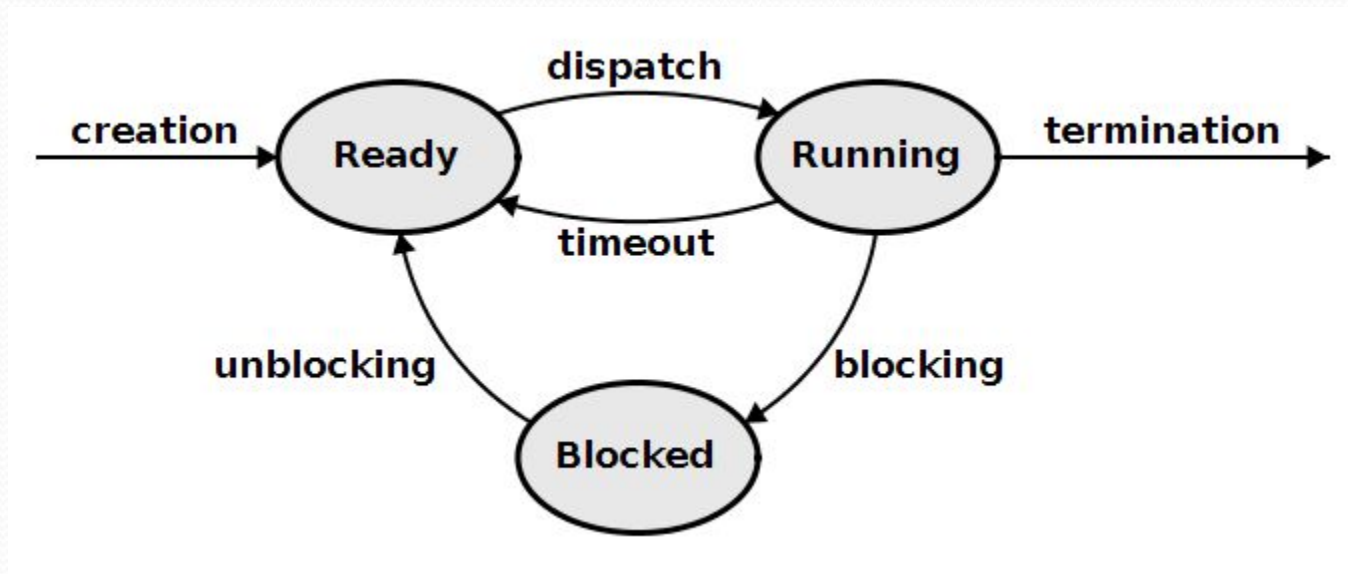
- **Parent Process (Microsoft Word):** You are actively using Microsoft Word (winword.exe).
- **Child Process (Microsoft Excel):** Microsoft Word, as the parent process, then launches a new instance of Microsoft Excel (excel.exe) to handle the creation or editing of the embedded worksheet. This instance of Excel becomes a child process of Word.

Layout of process inside main memory



- Stack: This section contains local variable, function and returns address.
- Heap: This Section is used to provide dynamic memory whenever memory is required by the program during runtime It is provided form heap section.
- Text: This section contains the executable instruction, constants
- Data: This Section contains the global variables and static local variables.

Process Life Cycle




- The life-cycle of a process can be described by a state diagram which has states representing the execution status of the process at various times .
- A process is created .
- Whenever a process is created, it directly enters in the ready state, in which, it waits for the CPU to be assigned.
- One of the processes from the ready state will be chosen by the OS depending upon the scheduling algorithm.
- From the Running state, a process can make the transition to the block or wait state depending upon the scheduling algorithm or the intrinsic behavior of the process.
- A process terminates or ends.

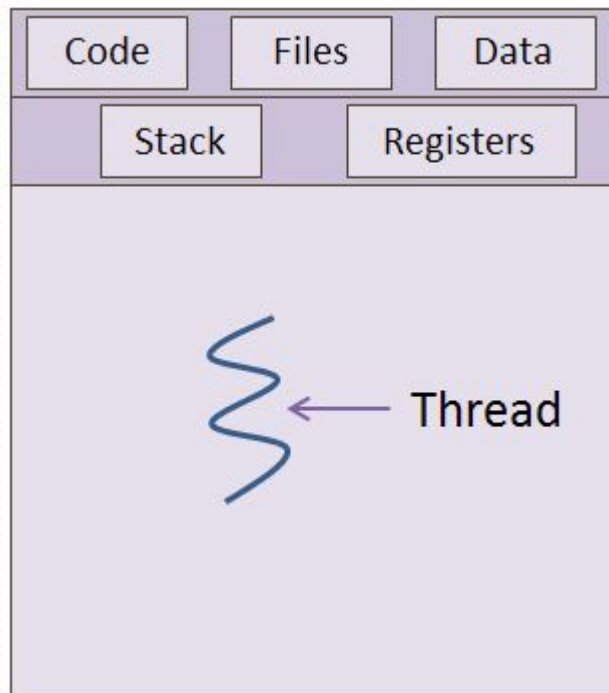
Threads


- A thread is the smallest unit of processing that can be performed in an OS.
- Think of MS Word application, which is a process that runs on computer. But an application can do more than one thing at a time, which means that a given process in an operating system can have one or more **threads**.
-

Advantages of Thread over Process

- 1. *Responsiveness*: If the process is divided into multiple threads, if one thread completes its execution, then its output can be immediately returned.
- 2. *Faster context switch*: Context switch time between threads is lower compared to process context switch. Process context switching requires more overhead from the CPU.

- 
- 3. *Resource sharing*: Resources like code, data, and files can be shared among all threads within a process.
 - 5. *Communication*: Communication between multiple threads is easier, as the threads share common address space.










Processes	Threads
When switching a process, operating system's resources are required	No OS resources are required for thread-switching
If a process is blocked, other processes waiting in the queue are also blocked	If a thread is blocked, another thread in the same process can still execute
Each process uses same code and has its own memory	All threads can share files and share child processes
An application having multiple processes will use more system resources	Processes using multiple threads use less system resources
Each process works on its own	Threads can access data of other threads

Multithreading


- **Multi-threading** is the ability of a process to execute multiple threads at the same time.
- Again, the MS Word example is appropriate in multi-threading scenarios.
- The process can check spelling, auto-save, and read files from the hard-drive, all while you are working on a document.

- 
- Consider the following diagram. The threads share the same code, files, and data.
 - This means that two or more threads can run at the same time (auto-save, grammar check, spell-check, word-count, etc.).

Code	Files		Data
Stack	Stack	Stack	Stack
Registers	Registers	Registers	Registers
			

Benefits of Multithreading

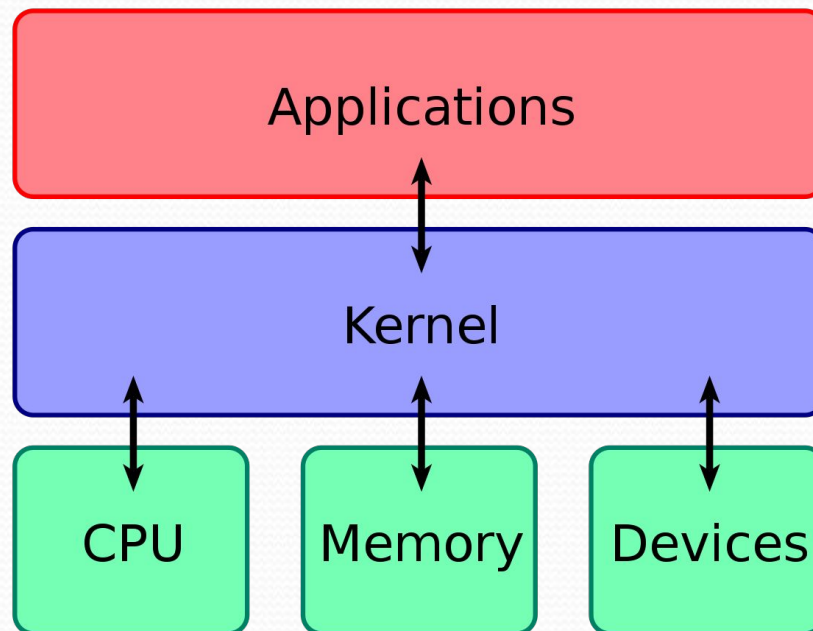
- **Resource Sharing** :All the threads of a process share its resources such as memory, data, files etc.
- **Responsiveness** :Program responsiveness allows a program to run quickly. For example - A web browser with multithreading can use one thread for user contact and another for image loading at the same time.

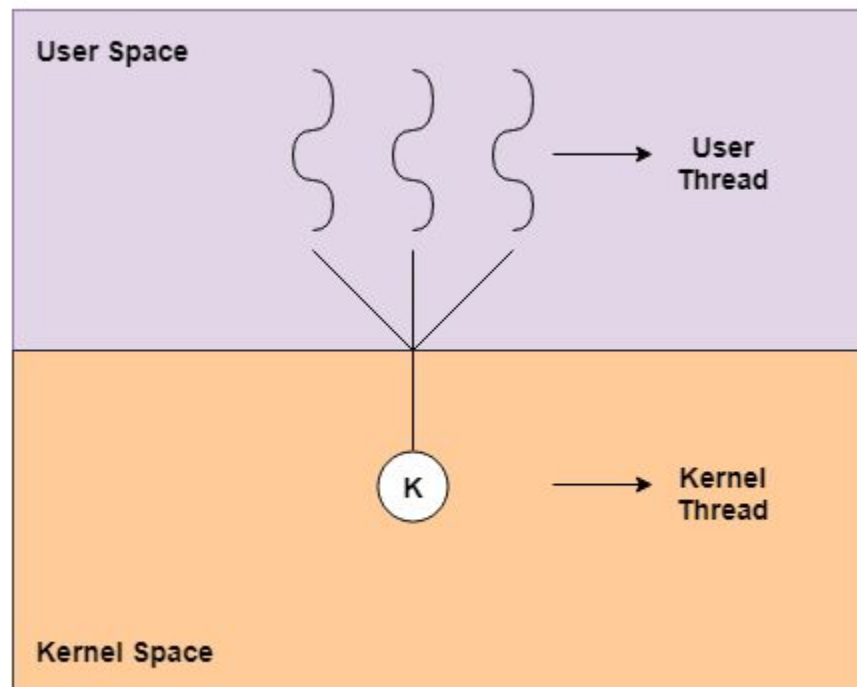
- 
- **Utilization of Multiprocessor Architecture :** In a multiprocessor architecture, each thread can run on a different processor in parallel using multithreading.
 - **Economy :** It is more economical to use threads as they share the process resources.

Types of Thread

- The two main types of threads are user-level threads and kernel-level threads.
- A diagram that demonstrates these is as follows –

- *Note : The kernel is the essential center of a computer operating system (OS). It is the core that provides basic services for all other parts of the OS.*





User-Level

- Managed by application
- Kernel not aware of thread
- Context switching cheap
- Create as many as needed
- Must be used with care

Kernel-Level


- Managed by kernel
- Consumes kernel resources
- Context switching expensive
- Number limited by kernel resources
- Simpler to use


User level thread– Green Threads in Java

kernel level thread– Linux Native POSIX Thread Library (NPTL)

User Level Thread

- The user-level threads are implemented by users and the kernel is not aware of the existence of these threads.
- It handles them as if they were single-threaded processes.
- User-level threads are small and much faster than kernel level threads.

- 
- They are represented by a program counter(PC), stack, registers and a small process control block.
 - Also, there is no kernel involvement in synchronization for user-level threads.
 - examples: Java thread.

- 
- User-level threads are easier and faster to create than kernel-level threads. They can also be more easily managed.
 - User-level threads can be run on any operating system.
 - There are no kernel mode privileges required for thread switching in user-level threads.

Kernel Level Thread

Kernel-level threads are handled by the operating system directly and the thread management is done by the kernel.

The context information for the process as well as the process threads is all managed by the kernel.

Because of this, kernel-level threads are slower than user-level threads. Example: Window .



Problems with Processes

- Creating and managing processes is generally regarded as an *expensive* task .
- Making sure all the processes peacefully co-exist on the system is not easy .

Thread Model

- All modern OS support Kernel Level Thread, allowing kernel to perform multiple simultaneous task.
- Hence , user thread must be mapped to kernel thread.
- Three models:
 - Many to One
 - One to One
 - Many to Many



Many to One

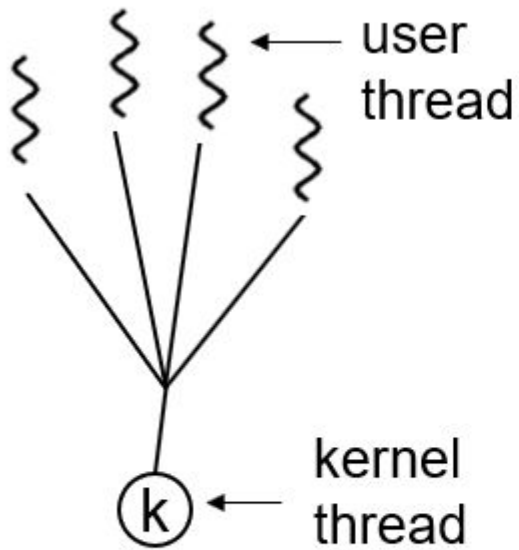
- Maps many user level thread to one kernel level thread

One to One

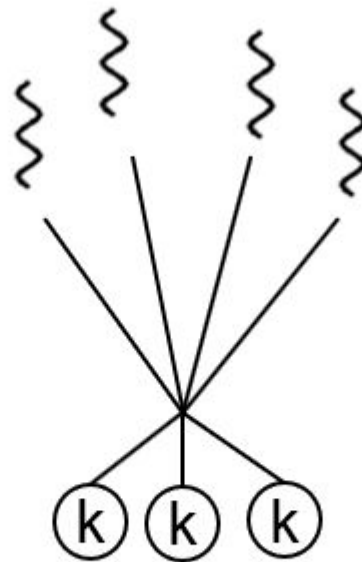
- Maps one user level thread to one kernel level thread

Many to Many

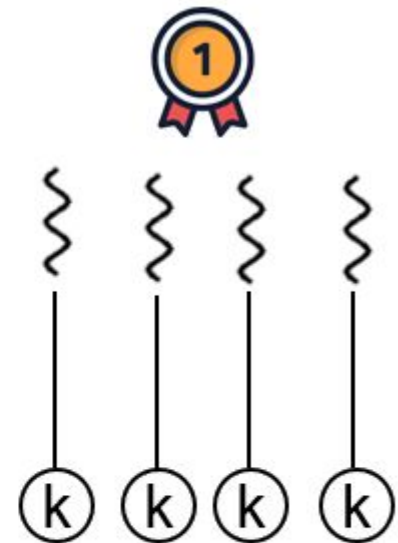
- Maps many user level thread to many kernel level thread



Many-to-one
thread model



Many-to-many
thread model



One-to-one
thread model




Students Work

- Example of each type of model

Virtualization

- **Virtualization** is a technology that allows you to create multiple simulated (virtual) environments or dedicated resources from a single physical hardware system.
- It enables one physical computer (host) to run multiple virtual machines (VMs), each acting like a separate computer with its own operating system and applications.

Virtualization divides a single physical machine into multiple logical systems.


- 
- **Host Machine:** The physical machine that runs the virtualization software.
 - **Guest Machine (Virtual Machine):** The virtual environment created on the host.
 - **Hypervisor:** The software or firmware that enables virtualization by managing virtual machines.

Hypervisor

A **hypervisor**, also called a **Virtual Machine Monitor (VMM)**, is a software or firmware layer that enables **virtualization**.

There are **two main types** of hypervisors:

1. **Type 1 Hypervisor (Bare-Metal Hypervisor)**
2. **Type 2 Hypervisor (Hosted Hypervisor)**



A **Type 1 hypervisor** runs **directly on the physical hardware** (bare metal) of the host machine.

It does **not require a host operating system**.

The hypervisor itself acts as the OS that manages all hardware resources and virtual machines.

Working:

- Installed directly on hardware.
- Allocates CPU, memory, and storage resources to each VM.
- Provides high performance and security.

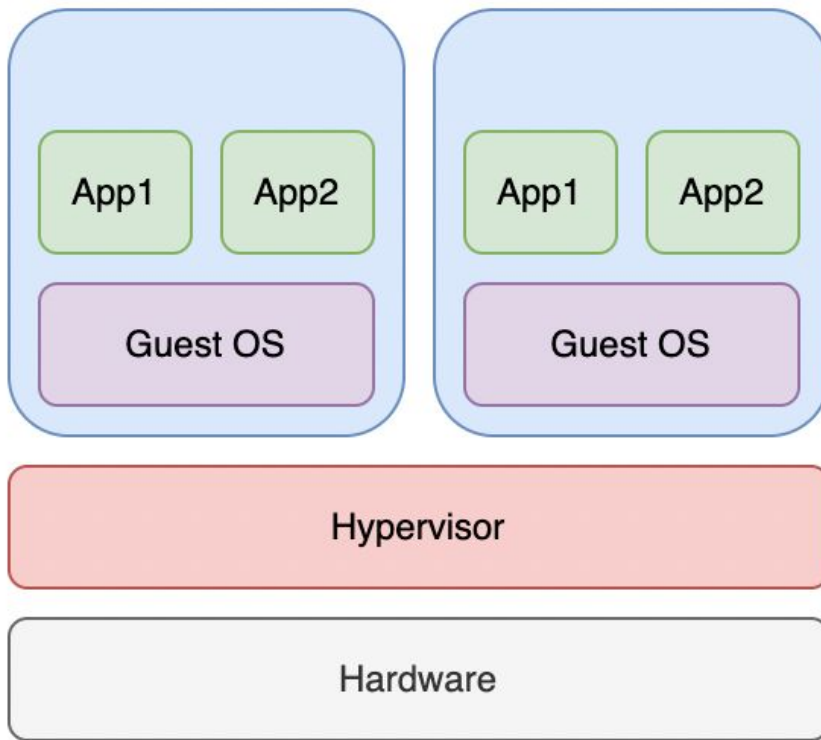


A **Type 2 hypervisor** runs **on top of a host operating system**, just like a regular application.

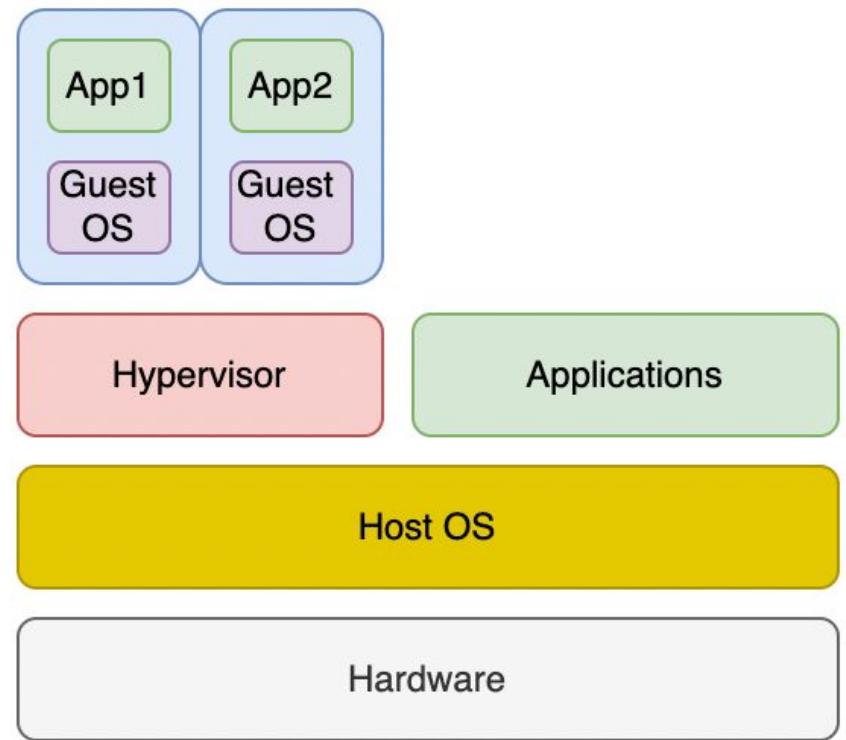
It uses the host OS to access hardware resources (CPU, RAM, Disk, etc.).

Working:

- Installed as software on an existing OS.
- Relies on host OS drivers for hardware access.
- Easier to install and use for testing and development.



Type1 Hypervisor



Type2 Hypervisor



Advantage of Virtualization

Efficient utilization of hardware resources

Reduced hardware and maintenance costs

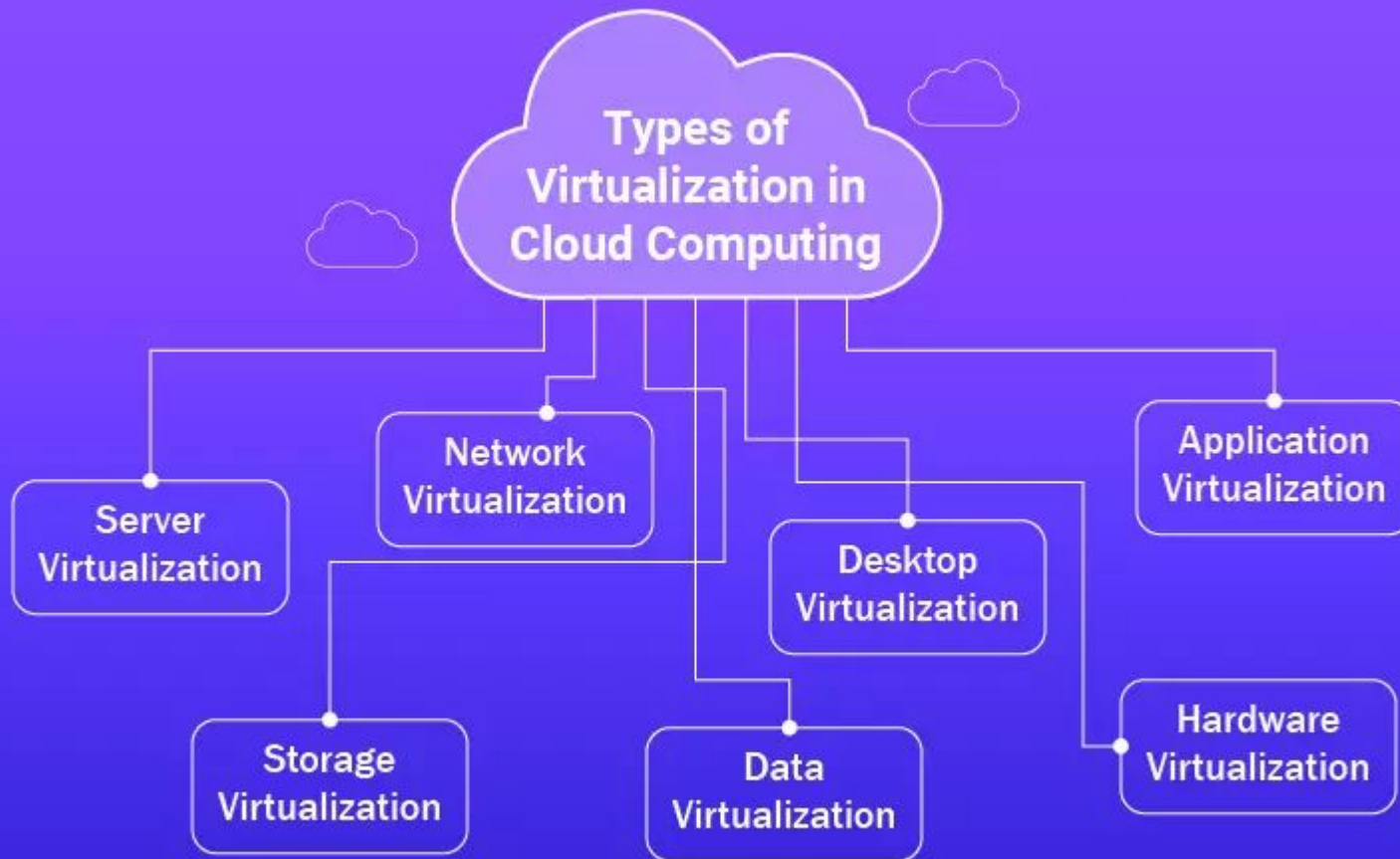
Lower power and cooling requirements

Easier system management and maintenance

Isolation between virtual machines (improved security)

Simplified data backup and recovery

Types of Virtualization



1. Hardware Virtualization

Hardware virtualization allows multiple virtual machines (VMs) to run on a single physical machine by abstracting the hardware resources.

A **hypervisor** (or Virtual Machine Monitor, VMM) is installed on the physical server to create and manage these virtual environments.

Types of Hardware Virtualization:

- **Full Virtualization:** Guest OS runs unmodified; hypervisor emulates complete hardware (e.g., VMware, VirtualBox).
- **Para Virtualization:** Guest OS is aware of virtualization; it communicates with the hypervisor for better performance (e.g., Xen).



Examples:

- VMware vSphere
- Microsoft Hyper-V
- Oracle VirtualBox

Used in **data centers and cloud computing** to run multiple servers on one hardware system.

2. Network Virtualization:

Network virtualization combines hardware (routers, switches, etc.) and software resources into a single, software-based virtual network.

It abstracts network services (like routing, security, or bandwidth) from physical network devices.

Examples:

- VMware NSX
- Cisco ACI (Application Centric Infrastructure)
- GNS3 (for simulation)

3. Desktop Virtualization:

Desktop virtualization allows a user's desktop environment (including OS, files, and applications) to run on a remote server instead of the local computer.

Users access the virtual desktop via thin clients or web interfaces.

Examples:

- VMware Horizon View
- Citrix Virtual Apps and Desktops
- Microsoft Remote Desktop Services (RDS)



Used in organizations to manage multiple desktops centrally, especially in remote or hybrid work setups.

Benefit:

- Easier IT management and updates
- Access desktops from anywhere
- Enhanced security (data stays on server)

● 4. Storage Virtualization:

Storage virtualization combines multiple physical storage devices (like HDDs, SSDs, or SANs) into a single logical storage pool.

It makes storage management easier by presenting virtualized storage as a single unit to users or applications.

⚙️ Examples:

- IBM SAN Volume Controller
- NetApp ONTAP
- VMware vSAN



Use Case:

Used in **enterprise storage systems** and **cloud storage** to improve performance, flexibility, and backup management.

Benefit:

Simplifies data migration, backup, and replication processes.

5. Server Virtualization:


Server virtualization divides a physical server into multiple virtual servers.

Each virtual server runs its own operating system and applications independently.

It's the **most common form** of virtualization used in cloud data centers.

Examples:

- VMware ESXi
- Citrix XenServer
- Microsoft Hyper-V



Used to consolidate workloads, reduce server sprawl, and increase resource utilization.

Benefit:

Improves **efficiency, scalability**, and **reduces hardware costs**.



6. Data virtualization:

This is the kind of virtualization in which the data is collected from various sources and managed that at a single place without knowing more about the technical information like how data is collected, stored & formatted then arranged that data logically.

Many big giant companies are providing their services like Oracle, IBM, At scale, Cdata, etc.

7. Operating System Virtualization

In OS virtualization, the **operating system kernel** allows multiple isolated user-space instances, called **containers**, to run on a single host.

Each container behaves like an independent OS environment but shares the same kernel.

Examples:

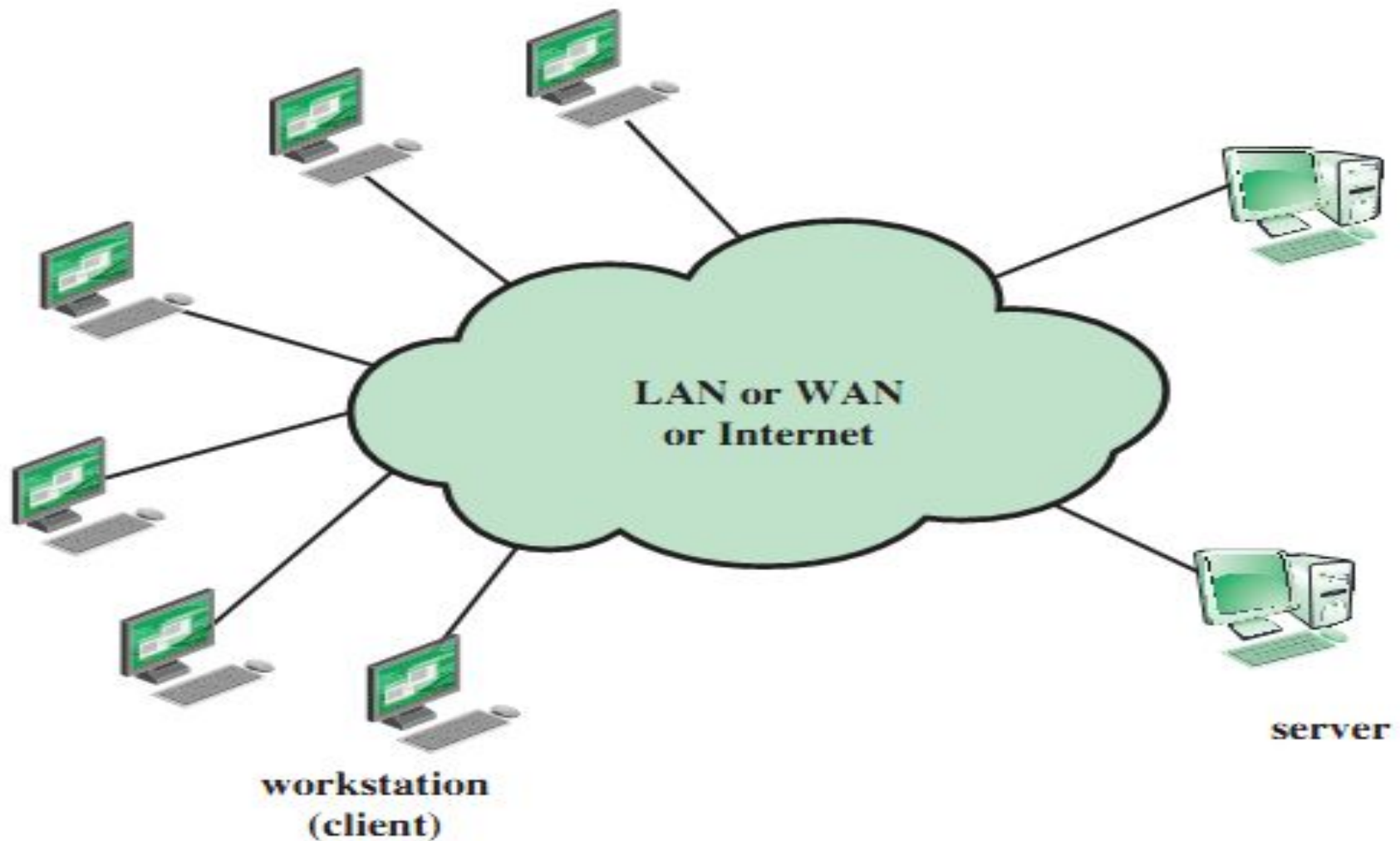
- Docker
- Linux Containers (LXC)
- OpenVZ

Used in **software development and deployment** to package and run applications in isolated environments.

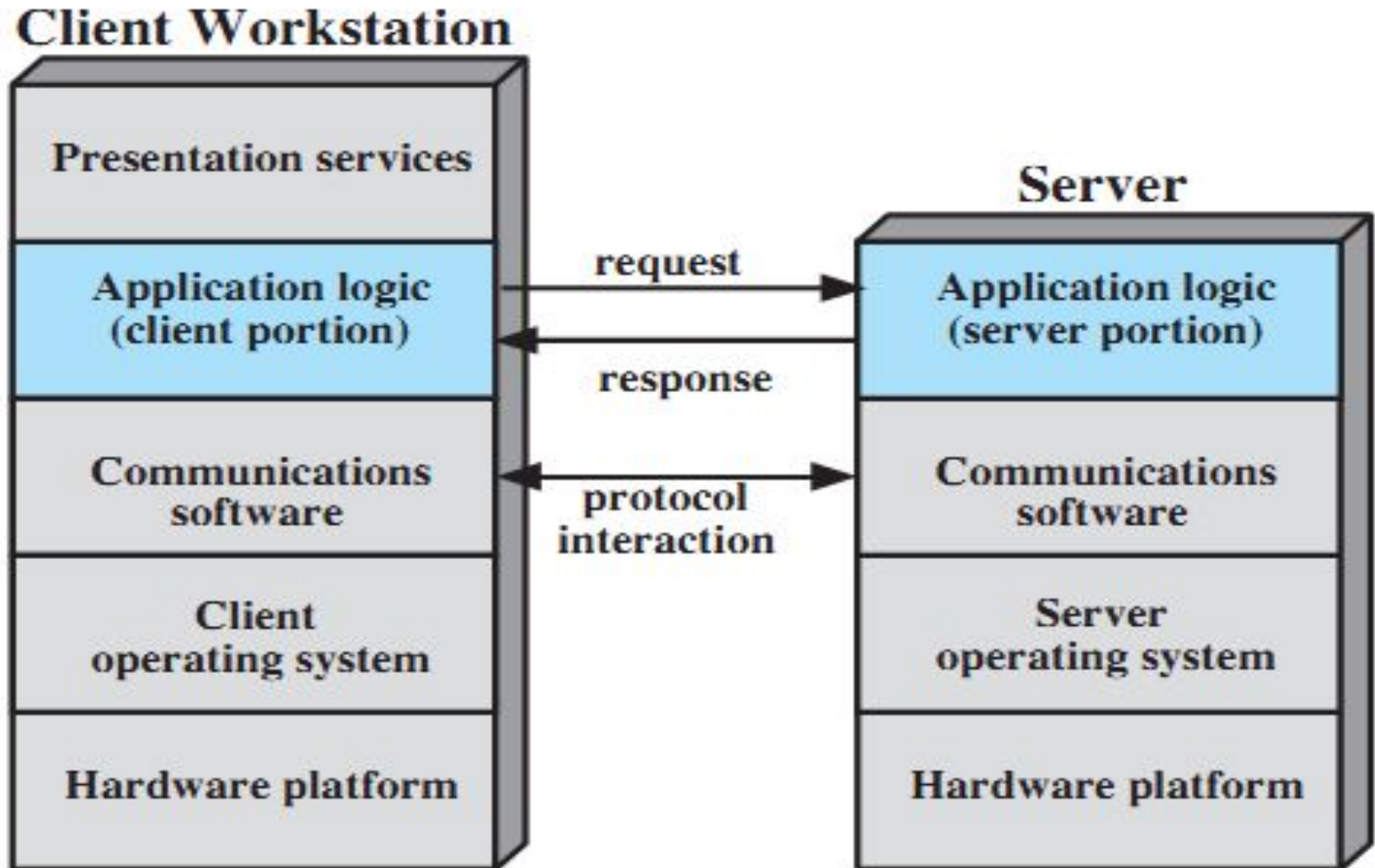
Clients

- What's a client?
- *Definition:* “A program which **interacts** with a human user and a remote server.”
- Typically, the user interacts with the client via a GUI.
- Of course, there's more to clients than simply providing a UI. Remember the multi-tiered levels of the Client/Server architecture from earlier

Generic Client/Server Environment



Generic Client/Server Architecture



Servers

- What's a server?
- *Definition:* “A process that implements a specific service **on behalf of** a collection of clients”.
- Typically, servers are organized to do one of two things:
 1. Wait
 2. Service... wait ... service ... wait ... service ... wait ...

Servers: Iterative and Concurrent

- *Iterative*: server handles request, then returns results to the client; any new client requests *must wait* for previous request to complete (also useful to think of this type of server as *sequential*).
- *Concurrent*: server does not handle the request itself; a separate thread or sub-process handles the request and returns any results to the client; the server is then free to immediately service the next client (i.e., there's no waiting, as service requests are processed in *parallel*).

Server “States”- Stateful & Stateless

- A Stateful server remember client data (state) from one request to the next. Stateful servers, do store session state. They may, therefore, keep track of which clients have opened which files, current read and write pointers for files, which files have been locked by which clients, etc.
- A Stateless server keeps no state information.Stateless file servers do not store any session state.

Code Migration

Code Migration is the process of **transferring code (or a program segment)** from one machine (node) to another in a **distributed system** for execution.

In simple words, it means **moving a running program or its components** (code, data, and execution state) **from one computer to another** within a network.

Example

Imagine a web application where a client requests a complex computation.

Instead of performing it locally, the **code is sent to a more powerful server** that executes it and returns the result.

This transfer and execution of code is called **code migration**.

Process and Code Migration

Under certain circumstances, in addition to the usual passing of data, *passing code* (even while it is executing) can greatly simplify the design of a DS.

However, code migration can be inefficient and very costly.

So, why migrate code?

Reasons for Migrating Code

Why? Biggest single reason: **better performance.**

The big idea is to move a compute-intensive task from a *heavily loaded* machine to a *lightly loaded* machine

“on demand” and “as required”.

Advantages of Code Migration

Load distribution: Balance workload among servers by moving tasks dynamically.

Performance improvement: Execute tasks closer to the required data or on faster machines.

Fault tolerance :Migrate code away from a failing node to another available node.

Flexibility and scalability :Enables dynamic system expansion and adaptive resource usage.



SOME SOLVED QUESTIONS

USER LEVEL VS KERNEL LEVEL

User level thread

User thread are implemented by users.

OS doesn't recognize user level threads.

Implementation of User threads is easy.

Context switch time is less.

Example : Java thread, POSIX threads.

Kernel level thread


kernel threads are implemented by OS.

Kernel threads are recognized by OS.

Implementation of Kernel thread is complicated.

Context switch time is more.

Example : Window Solaris.

- 
- WHY ULT: User-level threads are **easier and faster to create than kernel-level threads**. They can also be more easily managed. User-level threads can be run on any operating system. There are no kernel mode privileges required for thread switching in user-level threads.
 - WHY KLT: If one kernel thread perform blocking operation then another thread can continue execution.


- 
- **Actions taken by a kernel to context-switch between kernel-level threads are-**


Context switching between kernel threads typically requires saving the value of the CPU registers from the thread being switched out and restoring the CPU registers of the new thread being scheduled.





**What resources are used when a thread created?
How do they differ from those when a process is created?**


- When a thread is created the threads does not require any new resources .The thread shares the resources like memory of the process to which they belong to.
- Where as if a new process creation is very heavyweight because it always requires new address space to be created .

- 
- **The actions taken by a thread library to context switch between user-level threads.**
 - Answer: In general, context switching between user threads involves taking a user thread off its Light Weight Process and replacing it with another thread. This act typically involves saving and restoring the state of the registers.

- 
- Under what circumstances does a multithreaded solution using multiple kernel threads provide better performance than a single-threaded solution on a single-processor system?

- 
- When a kernel thread suffers a page fault, another kernel thread can be switched in to use the interleaving time in a useful manner. A single-threaded process, on the other hand, will not be capable of performing useful work when a page fault takes place. Therefore, in scenarios where a program might suffer from frequent page faults or has to wait for other system events, a multi-threaded solution would perform better even on a single-processor system.

- 
- Can a multithreaded solution using multiple user-level threads achieve better performance on a multiprocessor system than on a single processor system? Explain.

- 
- A multithreaded system comprising of multiple user-level threads cannot make use of the different processors in a multiprocessor system simultaneously. The operating system sees only a single process and will not schedule the different threads of the process on separate processors. Consequently, there is no performance benefit associated with executing multiple user-level threads on a multiprocessor system.

- Provide two programming examples in which multithreading does not provide better performance than a single-threaded solution.

- Any kind of sequential program is not a good candidate to be threaded. An example of this is a program that calculates an individual tax return. (2) Another example is a "shell" program such as the C-shell or Korn shell. Such a program must closely monitor its own working space such as open files, environment variables, and current working directory.