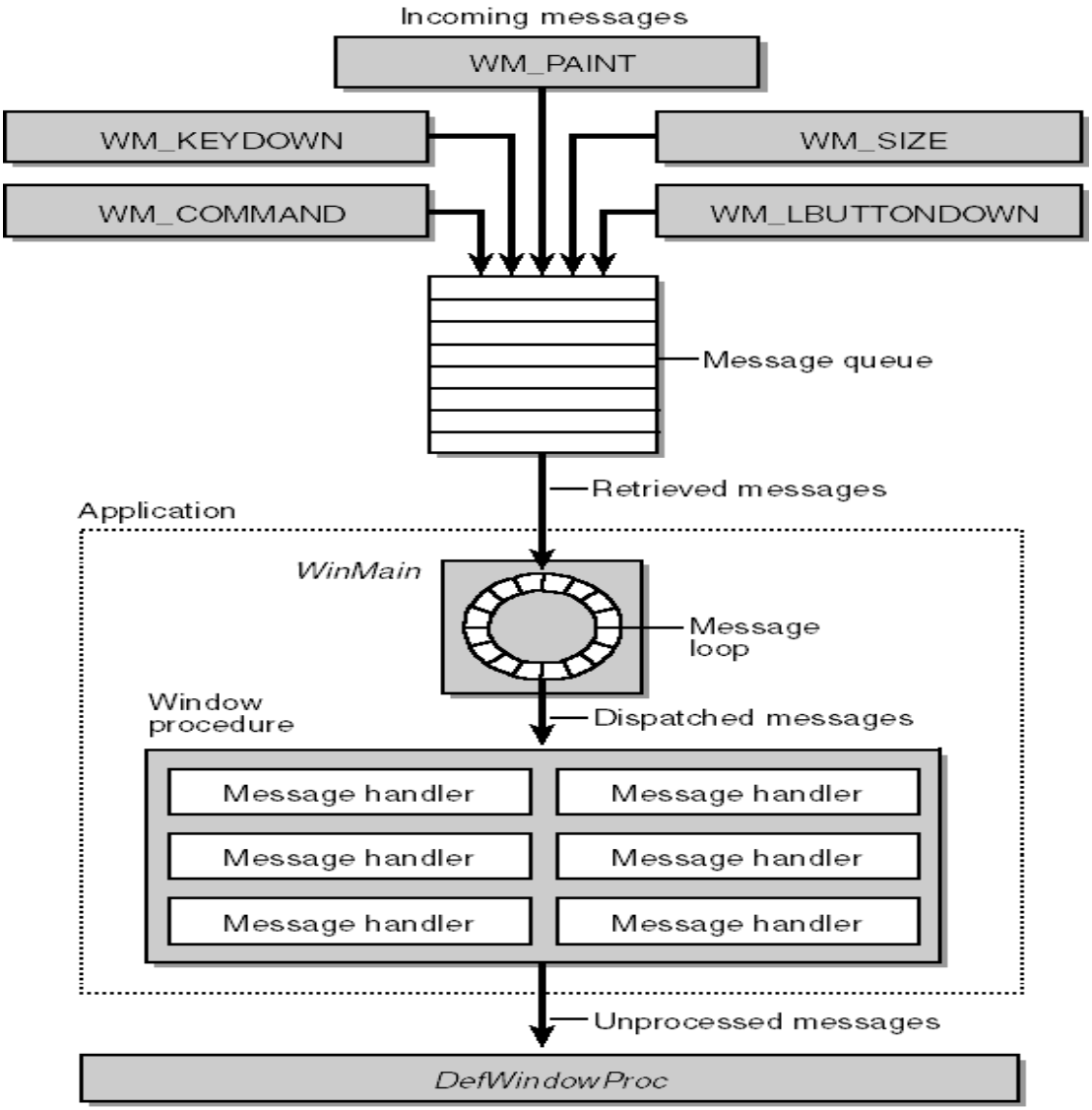
# Windows programming

Windows is an **event driven programming mechanism**, in which applications respond to events by processing messages sent by the operating system. An event could be a keystroke, a mouse click, or a command for a window to repaint itself, among other things.

The entry point for a Windows program is a function named **WinMain**, but most of the action takes place in a function known as the **window procedure**. The window procedure processes messages sent to the window.

WinMain creates that window and then enters a message loop, alternately retrieving messages and dispatching them to the window procedure. Messages wait in a message queue until they are retrieved.




Window Procedure

A window procedure is a **function** that receives and processes all messages sent to the window. Every window class has a window procedure, and every window created with that class uses that same window procedure to respond to messages.

- **Startup of a Simple Windows Program?**

```
LONG WINAPI WndProc(HWND, UINT, WPARAM, LPARAM);
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
   LPSTR lpszCmdLine, int nCmdShow)
{
```

```
   WNDCLASS wc;
   HWND hwnd;
   MSG msg;
   //Fill WNDCLASS structure
   RegisterClass(&wc);
   hwnd = CreateWindow(…);
   ShowWindow(hwnd, nCmdShow);
   UpdateWindow(hwnd);
   while (GetMessage(&msg, NULL, 0, 0)) {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
   }
   return msg.wParam;
}
LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam,
   LPARAM lParam)
{
}
```

# MFC

The Microsoft Foundation Class Library (MFC) is an "application framework" for programming in Microsoft Windows. MFC provides much of the code, which are required for the following –

- Managing Windows.
- Menus and dialog boxes.
- Performing basic input/output.
- Storing collections of data objects, etc.

You can easily extend or override the basic functionality the MFC framework in you C++ applications by adding your application-specific code into MFC framework.

- ## MFC Framework
  - The MFC framework provides a set of reusable classes designed to simplify Windows programming.
  - MFC provides classes for many basic objects, such as strings, files, and collections that are used in everyday programming.
  - It also provides classes for common Windows APIs and data structures, such as windows, controls, and device contexts.
  - The framework also provides a solid foundation for more advanced features, such as ActiveX and document view processing.

- ## Basic features of MFC

1.Application Framework: The MFC library framework includes its own application structure-one that has been proved in many software environments.App wizard generates skeleton code for your entire application, and class wizard generates prototypes and function bodies for message handlers.
2.Message Mapping
3.Runtime class information
4.Serialization

2.Graphics Drawing Support

3. Dialog Support

- ## Principle base class in MFC?

CObject is the principal base class for the Microsoft Foundation Class Library. The majority of MFC classes are derived, either directly or indirectly, from CObject.

CObject provides basic services, including
– Serialization support
– Run-time class information
– Object diagnostic output
– Compatibility with collection classes

- ## classes in MFC which are not derived from CObject class ?

CArchive
CDumpContext
CRuntimeClass

### Simple Value Types
CPoint
CRect
CSize
CString
CTime
CTimeSpan

**Synchronization Support**

CSingleLock
CMultiLock

**Message Handling and Mapping**

In traditional programs for Windows, Windows messages are handled in a large switch statement in a window procedure. MFC instead uses message maps to map direct messages to distinct class member functions.

**Documents and Views Story**

Most of your MFC library applications will be more complex than the previous examples. Typically, they'll contain **application** and **frame** classes plus two other classes that represent the "**document**" and the "**view**." This document-view architecture is the core of the application framework and is loosely based on the Model/View/Controller classes from the Smalltalk world.

In simple terms, the document-view architecture separates data from the user's view of the data.

- ## ASSERT and VERIFY?

ASSERT evaluates the expression only in the debug version and will throw an exception if the result is 0 and the program termintes.
VERIFY evalutes the expression in Debug and Release version also and if the result is 0, will throw an exception only in Debug mode.

### PostMessage and SendMessage?

The PostMessage function places (posts) a message in the message queue associated with the thread that created the specified window and returns without waiting for the thread to process the message.

The SendMessage function sends the specified message to a window or windows. It calls the window procedure for the specified window and does not return until the window procedure has processed the message.

### PeekMessage and GetMessage?

You can use the PeekMessage function to examine a message queue during a lengthy operation. PeekMessage is similar to the GetMessage function, both check a message queue for a message that matches the filter criteria and then copy the message to an MSG structure. The main difference between the two functions is that GetMessage does not return until a message matching the filter criteria is placed in the queue, whereas PeekMessage returns immediately regardless of whether a message is in the queue.

**what API used to Show/ hide window**

```
ShowWindow(hwnd, SW_HIDE); // to hide window
ShowWindow(hwnd, SW_SHOW); // to show window
```

## hInstance and hWnd?

hWnd is the window's handle and is how the OS defines a window when talking about it inside the PC. hInstance is the OS's handle for the program when running it.

## Modal and Modeless Dialog?

Modal dialog box captures the message loop, whereas model less does not. Call DoModal to create the dialog window and its controls for a modal dialog. If you wish to create a modeless dialog, call Create in the constructor of your Cdialog class.
Example for Model Dialog is Save, Save As Dialog in MS -Word.
Example for Modeless Dialog is Find,Replace dialogs.

### Create modeless dialog

m_pModeless is a variable of type CDialog or any of its descendants.

m_pModeless->Create(IDD_DIALOG1, this);

m_pModeless->ShowWindow(SW_SHOW);

this pointer as a paramter suggest we are creating a child dialog of the current dialog/window.

## hinsrtance and hprevinstance in WinMain function?

hInstance : will be having the handle of the current instance
hPrevInctance : will be having the handle of last instance, hPrevInstance is NULL if only one instance is running

## How Message Map works in an MFC application?

The message map functionality in an MFC application works by the support of 3 Macros, DECLARE_MESSAGE_MAP, BEGIN_MESSAGE_MAP, and END_MESSAGE_MAP and the WindowProc function implementation.

## What is serialization ?which function is responsible for serializing data ?

Searialization is the process of streaming the object data to or from a persistent storage medium. It's useful in Doc-View Architecture. CObject :: Serialize() function is used to do serialization.

**Whats is DDX & DDV in MFC?**

Dialog data exchange (DDX) is an easy way to initialize the controls in your dialog box and to gather data input by the user. Dialog data validation (DDV) is an easy way to validate data entry in a dialog box.

**How to give color for dialog button or static button?**

```
Brush                                                                    *brush;
Initialize   the   brush   pointer   in   the   constructor   of   your   Dialog
Code:
brush               =               new               CBrush(RGB(49,49,49));
Add the WM_CTLCOLR Message handler for the dialog and add the following code
Code:
switch                              (nCtlColor)                              {

case                                                                CTLCOLOR_BTN:
pDC->SetTextColor(RGB(0,                      255,                        0));
pDC->SetBkColor(RGB(0,                        0,                          0));
return                                      (HBRUSH)(brush->GetSafeHandle());
default:
return          CDialog::OnCtlColor(pDC,          pWnd,          nCtlColor);
}
```

# OS Concepts

- ## Virtual memory

***Virtual memory*** is what your program deals with. It consists of all of the addresses returned by ***malloc, new,*** etc. Each process has its own virtual-address space. Virtual address usage is theoretically limited by the address size of your program: 32-bit programs have 4GB of address space; 64-bit programs have vastly more. Practically speaking, the amount of virtual memory that a process can allocate is less than those limits.

*Physical memory* are the chips soldered to your motherboard, or installed in your memory slots. The amount of physical memory in use at any given time is limited to the amount of physical memory in your computer.

The virtual-memory subsystem maps virtual addresses that your program uses to physical addresses that the CPU sends to the RAM chips. At any particular moment, most of your allocated virtual addresses are unmapped; thus physical memory use is lower than virtual memory use. If you access a virtual address that is allocated but not mapped, the operating system invisibly allocates physical memory and maps it in. When you don't access a virtual address, the operating system might unmap the physical memory.

To take your questions in turn:

- **what operations in C++ would inflate virtual memory so much?**

`new`, `malloc`, static allocation of large arrays. Generally anything that requires memory in your program.

- **Is it a problem if my task is using gigs of virtual memory?**

It depends upon the usage pattern of your program. If you allocate vast tracks of memory that you never, ever touch, and if your program is a 64-bit program, it may be okay that you are using gigs of virtual memory.

Also, if your memory use grows without bound, you will eventually run out of some resource.

- **The stack and heap function variables, vectors, etc. - do those necessarily increase the use of physical memory?**

Not necessarily, but likely. The act of touching a variable ensures that, at least momentarily, it (and all of the memory "near" it) is in physical memory. (Aside: containers like `std::vector` may be allocated on either stack or heap, but the **contained** objects are allocated on the heap.)

- **Would removing a memory leak (via delete or free() or such) necessarily reduce both physical and virtual memory usage?**

Physical: probably. Virtual: yes.

# Inter Process Communication

A process can be of two types:

- Independent process.
- Co-operating process.

An independent process is not affected by the execution of other processes while a co-operating process can be affected by other executing processes. Though one can think that those processes, which are running independently, will execute very efficiently, in reality, there are many situations when co-operative nature can be utilized for increasing computational speed, convenience, and modularity. Inter-process communication (IPC) is a mechanism that allows processes to communicate with each other and synchronize their actions. The communication between these processes can be seen as a method of co-operation between them. Processes can communicate with each other through both:

## • Shared Memory

Communication between processes using shared memory requires processes to share some variable, and it completely depends on how the programmer will implement it. One way of communication using shared memory can be imagined like this: Suppose process1 and process2 are executing simultaneously, and they share some resources or use some information from another process. Process1 generates information about certain computations or resources being used and keeps it as a record in shared memory. When process2 needs to use the shared information, it will check in the record stored in shared memory and take note of the information generated by process1 and act accordingly. Processes can use shared memory for extracting information as a record from another process as well as for delivering any specific information to other processes.

## • Message passing

In this method, processes communicate with each other without using any kind of shared memory. If two processes p1 and p2 want to communicate with each other, they proceed as follows:

- Establish a communication link (if a link already exists, no need to establish it again.)
- Start exchanging messages using basic primitives.
  We need at least two primitives:
  – send(message, destination) or send(message)
  – receive(message, host) or receive(message)

## • Message Passing vs. Shared Memory

**Message** **Passing**

Message passing exchanges data via inter-process communication(IPC) mechanisms.

**Pros**: Generalizes to both local & remote communication.

**Cons**: May incur excess overhead for large messages in a "loop-back" configuration.

**Shared** **Memory**

Shared memory allows apps to access & exchange data as though they were local to address space of each process.

**Pros**: Can be more efficient for large messages in loop-back configuration.

**Cons**: Doesn't generalize efficiently to remote systems & can be more error-prone & non-portable for OO apps.

## ❖ Multiprocessing vs. Multithreading

**Multiprocessing**

A process is the unit of resource allocation & protection.
A process manages certain resources, e.g., virtual memory, I/O handlers, and signal handlers.

**Pros**: Process is protected from other processes via an MMU.

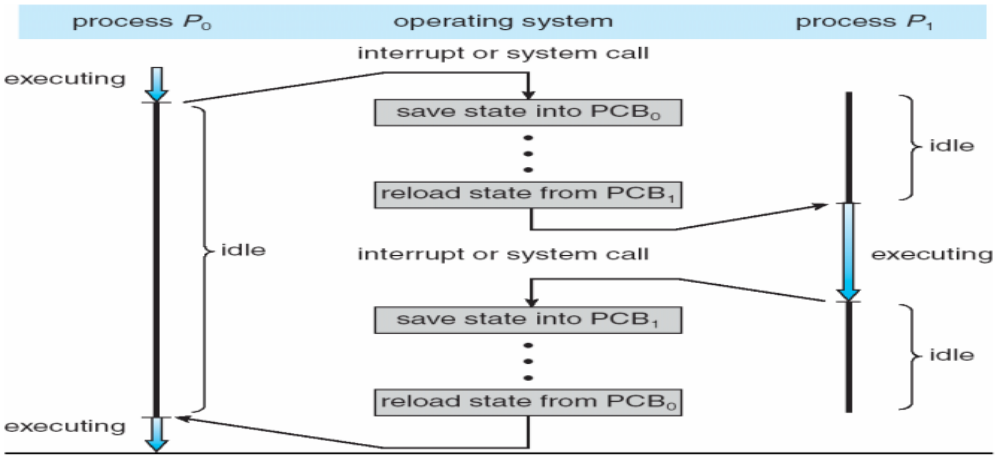
**Cons**: IPC between processes can be complicated and inefficient.

**Multithreading**

A thread is the unit of computation that runs in the context of a process.
A thread manages certain resources, e.g., stack, registers, signal masks, priorities, and thread-specific data
**Pros**: IPC between threads is more efficient that IPC between processes.
**Cons**: Threads can interfere with each other.

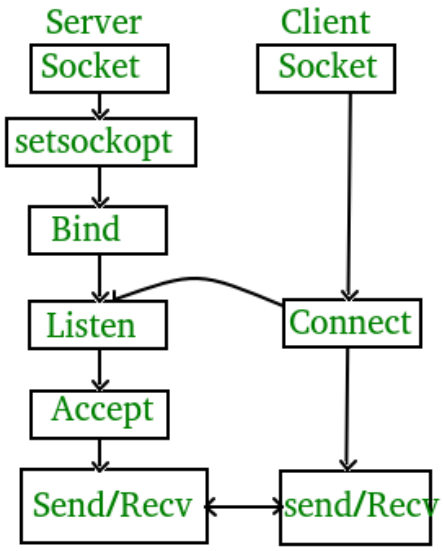
**What is Context Switching?**

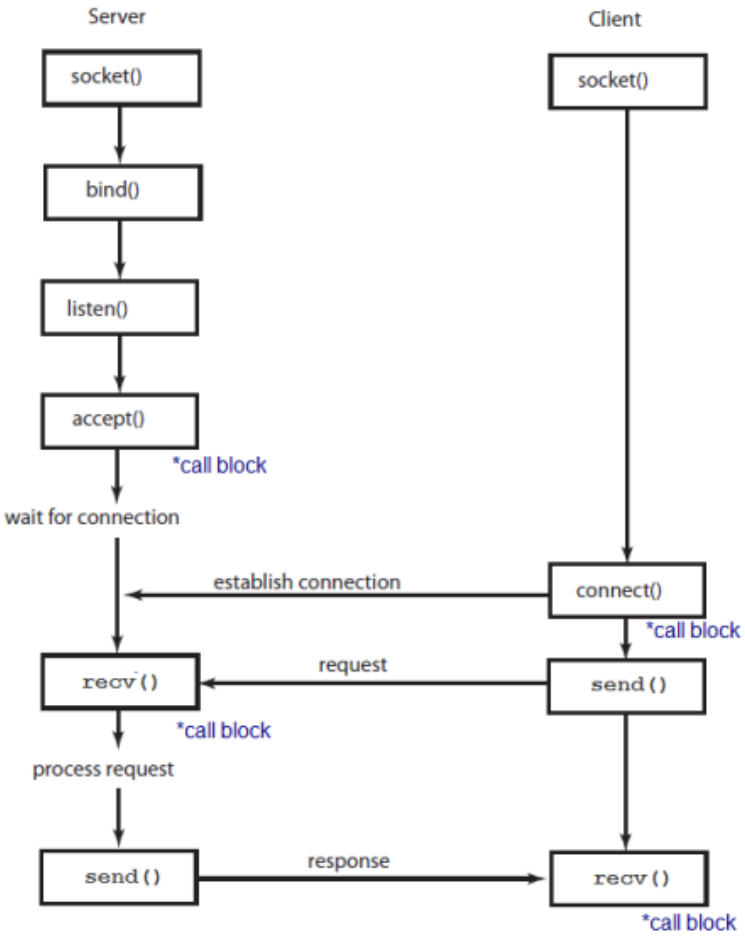
A context switching is a process that involves switching of the CPU from one process or task to another. In this phenomenon, the execution of the process that is present in the running state is suspended by the kernel and another process that is present in the ready state is executed by the CPU.



- TCP communication

❖ What is socket programming?
Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server.






## ❖ Socket Functions (Server)

1. int socket(int domain, int type, int protocol)

Used to create a new socket, returns a file descriptor for the socket or -1 on error.
It takes three parameters:
int sockfd = socket(domain, type, protocol);
**sockfd:** socket descriptor, an integer (like a file-handle)
**domain:** integer, communication domain e.g., AF_INET (IPv4 protocol) , AF_INET6 (IPv6 protocol)
**type:** communication type
SOCK_STREAM: TCP(reliable, connection oriented)
SOCK_DGRAM: UDP(unreliable, connectionless)
**protocol:** Protocol value for Internet Protocol(IP), which is 0. This is the same number which appears on protocol field in the IP header of a packet.(man protocols for more details)

2. Bind:

int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
After creation of the socket, bind function binds the socket to the address and port number specified in addr(custom data structure). In the example code, we bind the server to the localhost, hence we use **INADDR_ANY** to specify the IP address.

3. Listen:
int listen(int sockfd, int backlog);
It puts the server socket in a **passive mode**, where it **waits for the client to approach the server** to make a connection. The backlog, defines the maximum length to which the queue of pending connections for sockfd may grow. If a connection request arrives when the queue is full, the client may receive an error with an indication of **ECONNREFUSED**.

4. Accept:
int new_socket= accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
It extracts the first connection request on the queue of pending connections for the listening socket, sockfd, creates a new connected socket, and **returns a new file descriptor referring to that socket**. At this point, **connection is established between client and server**, and they are **ready to transfer data**.

### ❖ Socket Functions (Client)

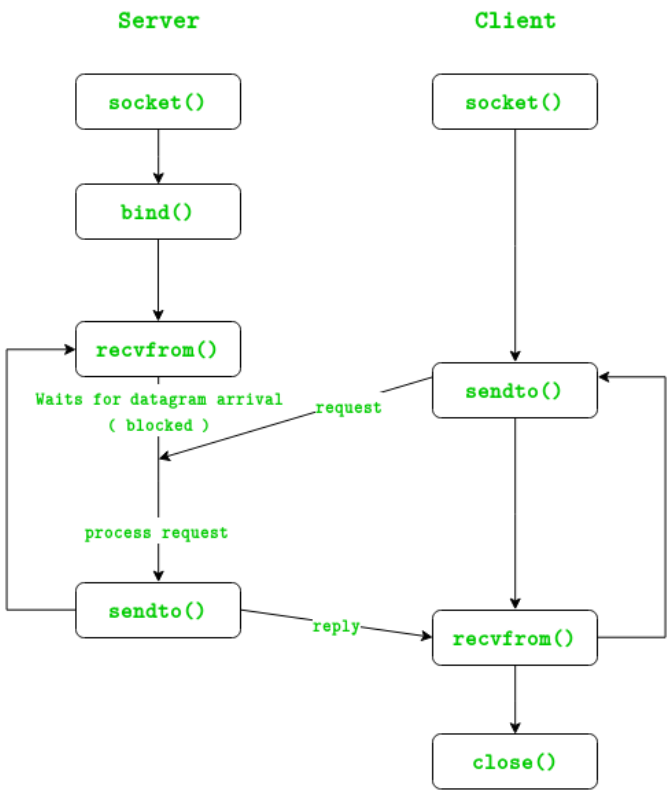
1. Connect:
int connect(int sockfd, const struct sockaddr *addr,
socklen_t addrlen);
The connect() system call **connects the socket referred to by the file descriptor sockfd** to the address specified by addr. Server's address and port is specified in addr.

### ❖ Socket Functions (Server & Client both)

1. int send(int fd, void *buffer, size_t n, int flags)
Sends n bytes from *buffer to socket fd.Returns the number of bytes sent or -1 on error.
2. int receive(int fd, void *buffer, size_t n, int flags)
Reveives n bytes from **socket fd** into ***buffer**. Returns the number of bytes received or -1 on error.

## • UDP Communication

In UDP, the client does not form a connection with the server like in TCP and instead just sends a datagram. Similarly, the server need not accept a connection and just waits for datagrams to arrive. Datagrams upon arrival contain the address of the sender which the server uses to send data to the correct client.




## ❖ TCP vs UDP

1. Stream Sockets:

- Stream sockets provide **reliable two-way** communication similar to when we call someone on the phone.
- One side initiates the connection to the other, and after the connection is established, either side can communicate to the other.
- In addition, there is immediate confirmation that what we said actually reached its destination.
- Stream sockets use a **Transmission Control Protocol (TCP)**, which exists on the transport layer
- Webservers, mail servers, and their respective client applications all use TCP and stream socket to communicate.

2. Datagram Sockets

- Communicating with a datagram socket is more like mailing a letter than making a phone call. The connection is **one-way** only and **unreliable**.
- If we mail several letters, we can't be sure that they arrive in the same order, or even that they reached their destination at all.
- Datagram sockets use User Datagram Protocol (UDP).

## ❖ Boost.Asio

Boost.Asio is a cross-platform C++ library for network and low-level I/O programming that provides developers with a consistent asynchronous model using a modern C++ approach.

Why Boost.Asio?

Writing networking code that is portable is easy to maintain has been an issue since long. C++ took a step to resolve this issue by introducing boost.asio. It is a cross-platform C++ library for

network and low-level I/O programming that provides developers with a consistent asynchronous model using a modern C++ approach.

### ❖ select function (winsock2.h)

The **select** function **determines the status of one or more sockets**, waiting if necessary, to perform synchronous I/O.

The **shutdown** function is used on all types of sockets **to disable reception, transmission, or both**.

Return value

The **select** function returns the **total number of socket handles that are ready** and contained in the **fd_set structures**, zero if the time limit expired, or SOCKET_ERROR if an error occurred.

### ❖ shutdown function (winsock2.h)

The **shutdown** function disables sends or receives on a socket.

Return value

If no error occurs, **shutdown** returns **zero**. Otherwise, a value of **SOCKET_ERROR** is returned,

### ❖ Real Time Systems

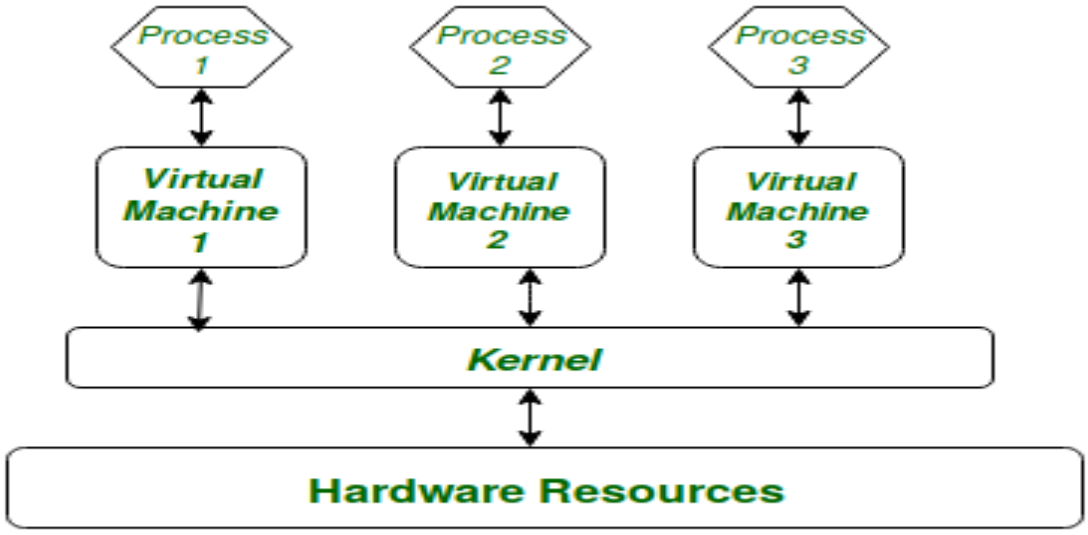
Real time system means that the system is subjected to real time, i.e., response should be guaranteed within a specified timing constraint or system should meet the specified deadline. For example: flight control system, real time monitors etc.

# Linux

**Linux** is an open source multi-tasking, multi-user operating system. It was initially developed by Linus Torvalds in 1991. Linux OS is widely used in desktops, mobiles, mainframes etc.

The main purpose of a computer is to run a predefined sequence of instructions, known as a program. A program under execution is often referred to as a process. Now, most special purpose computers are meant to run a single process, but in a sophisticated system such a

general purpose computer, are intended to run many processes simulteneously. Any kind of process requires hardware resources such are Memory, Processor time, Storage space, etc. In a General Purpose Computer running many processes simulteneously, we need a middle layer to manage the distribution of the hardware resources of the computer efficiently and fairly among all the various processes running on the computer. This middle layer is referred to as the **kernel**.




Per-Process Hardware Virtualization

- ## Core Subsystems

The **Core Subsystems** of the **Linux Kernel** are as follows:

1. The Process Scheduler

This kernel subsystem is responsible for fairly **distributing the CPU time among all the processes** running on the system simulteneously.

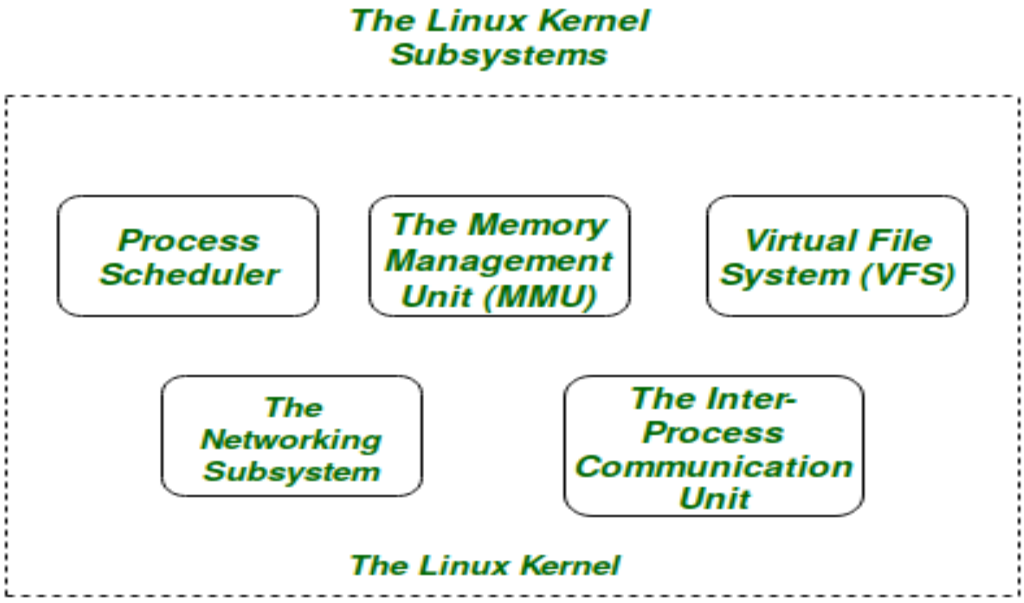
2. The Memory Management Unit (MMU)

This kernel sub-unit is responsible for proper **distribution of the memory resources** among the various processes running on the system. The MMU does more than just simply provide separate virtual address spaces for each of the processes.

3. The Virtual File System (VFS)

This subsystem is responsible for providing a unified interface to access stored data across different filesystems and physical storage media.

4. The Networking Unit
5. Inter-Process Communication Unit




- Linux vs Windows Commands

| SNo. | Windows | Linux | Description |
|---|---|---|---|
| 1. | dir | ls -l | Directory listing |
| 2. | ren | mv | Rename a file |
| 3. | copy | cp | Copying a file |
| 4. | move | mv | Moving a file |
| 5. | cls | clear | Clear Screen |
| 6. | del | rm | Delete file |
| 7. | fc | diff | Compare contents of files |
| 8. | find | grep | Search for a string in a file |

| SNo. | Windows | Linux | Description |
|---|---|---|---|
| 9. | command /? | man command | Display the manual/help details of the command |
| 10. | chdir | pwd | Returns your current directory location |
| 11. | time | date | Displays the time |
| 12. | cd | cd | Change the current directory |
| 13. | md | mkdir | To create a new directory/folder |
| 14. | echo | echo | To print something on the screen |
| 15. | edit | vim(depends on editor) | To write in to files. |
| 16. | exit | exit | To leave the terminal/command window. |
| 17. | format | mke2fs or mformat | To format a drive/partition. |
| 18. | free | mem | To display free space. |
| 19. | rmdir | rm -rf/rmdir | To delete a directory. |
| 20. | taskkill | kill | To kill a task. |
| 21. | tasklist | ps x | To list running tasks. |
| 22. | set var=value | export var=value | To set environment variables. |
| 23. | attrib | chown/chmod | To change file permissions. |
| 24. | tracert | traceroute | To print the route packets trace to network host. |

| SNo. | Windows | Linux | Description |
|---|---|---|---|
| 25. | at | cron | daemon to execute scheduled commands. |
| 26. | type | cat | To print contents of a file. |
| 27. | ping | ping | To send ICMP ECHO_REQUEST to network hosts. |
| 28. | nslookup | nslookup | To query Internet name servers interactively. |
| 29. | chdisk | du -s | For disk usage. |
| 30. | tree | ls -R | To list directory recursively. |

# Agile Methodology

- ## What Is Agile Methodology in Project Management?

- The Agile methodology is a way to manage a project by breaking it up into several phases.
- It involves constant collaboration with stakeholders and continuous improvement at every stage.
- Once the work begins, teams cycle through a process of planning, executing, and evaluating.
- Continuous collaboration is vital, both with team members and project stakeholders.

- ## Principles:

1. Highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. It welcomes changing requirements, even late in development.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shortest timescale.

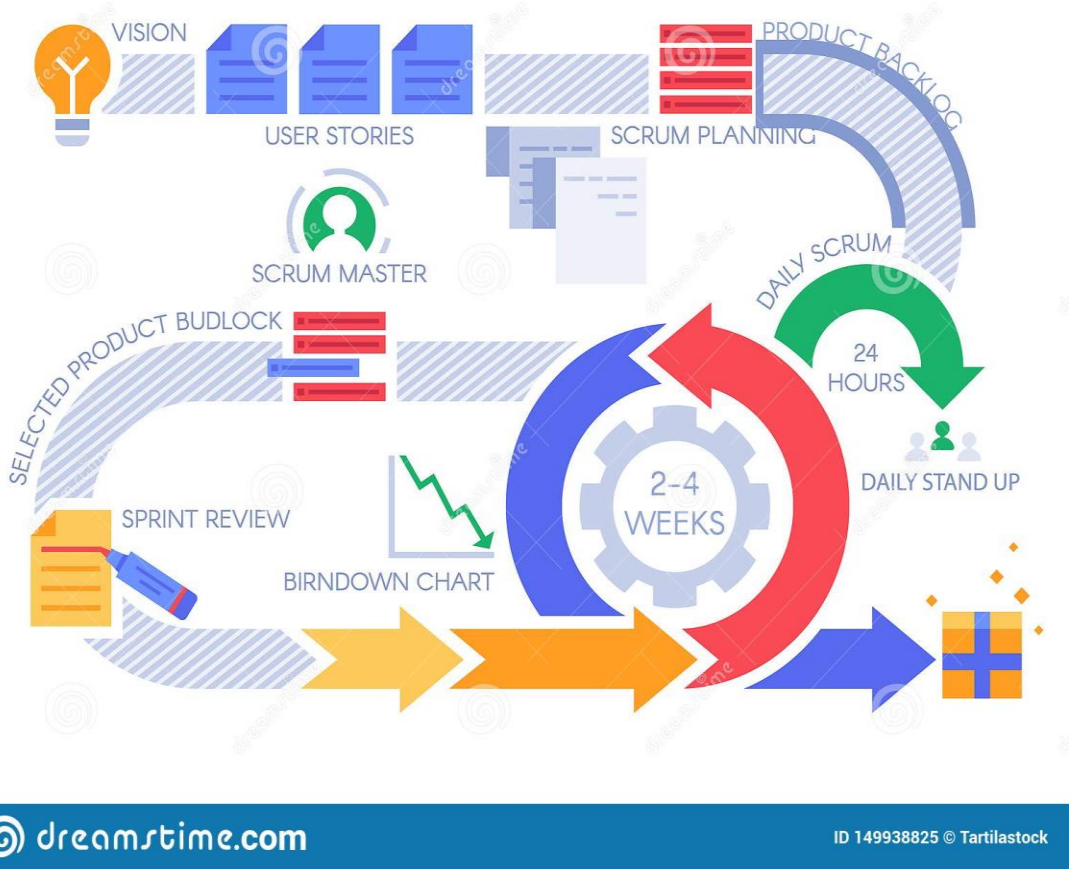
❖ ## Advantages:

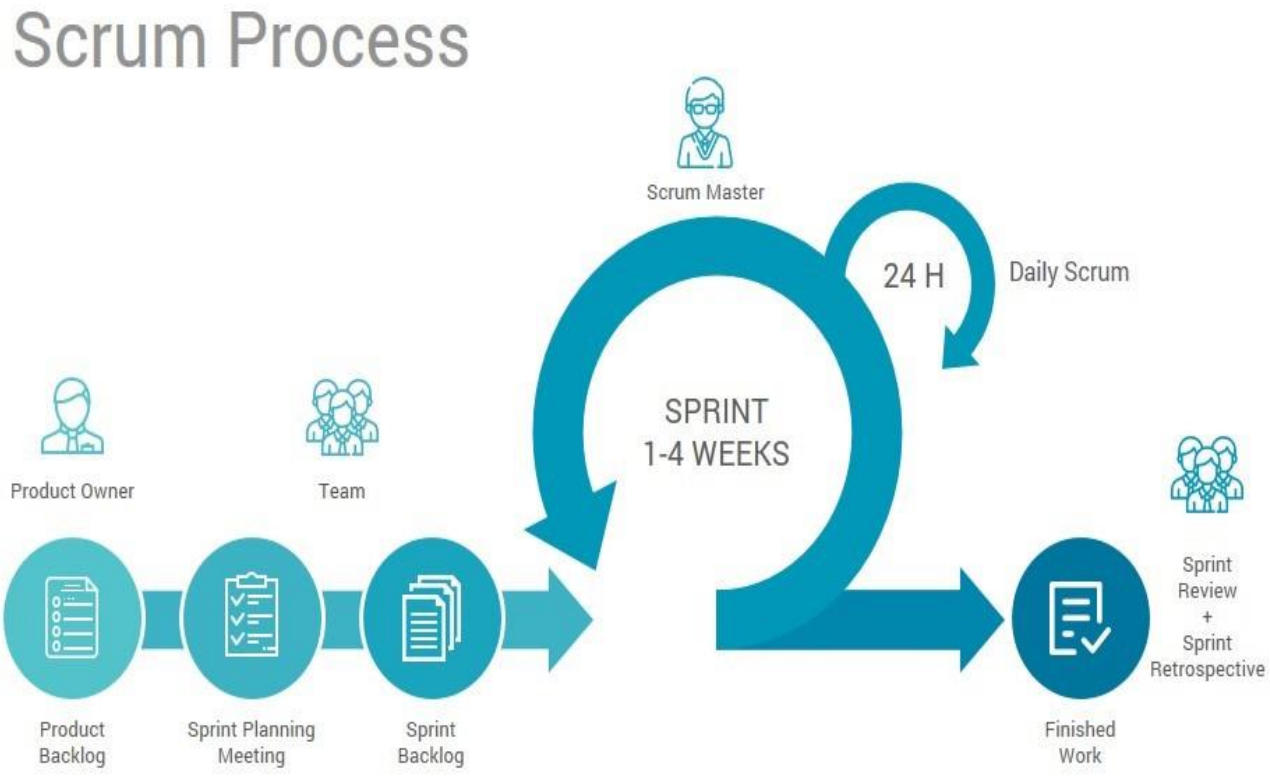
- Deployment of software is quicker and thus helps in increasing the **trust of the customer.**
- Can better adapt to rapidly changing requirements and respond faster.

- Helps in getting **immediate feedback** which can be used to improve the software in the next increment.
- People – Not Process. People and interactions are given a higher priority rather than process and tools.
- Continuous attention to technical excellence and good design.

## ❖ Disadvantages:

- In case of large software projects, it is difficult to assess the effort required at the initial stages of the software development life cycle.
- The Agile Development is more code focused and produces less documentation.
- Agile development is **heavily depended on the inputs of the customer**. If the customer has ambiguity in his vision of the final outcome, it is highly likely for the project to get off track.






# Networking concepts

- ## Domain Name System (DNS) in Application Layer

DNS is a host name to IP address translation service. DNS is a distributed database implemented in a hierarchy of name servers. It is an application layer protocol for message exchange between clients and servers.

Every host is identified by the IP address but remembering numbers is very difficult for the people and also the IP addresses are not static therefore a mapping is required to change the domain name to IP address. So DNS is used to convert the domain name of the websites to their numerical IP address.

### Domain :

There are various kinds of DOMAIN :
**Generic domain** : .com(commercial) .edu(educational) .mil(military) .org(non profit organization) .net(similar to commercial) all these are generic domain.

**Country domain** .in (india) .us .uk
**Inverse domain** if we want to know what is the domain name of the website. Ip to domain name mapping.So DNS can provide both the mapping for example to find the ip addresses of geeksforgeeks.org then we have to type nslookup www.geeksforgeeks.org.

- ## Difference between Ping and Traceroute

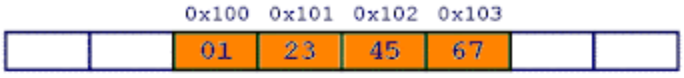
**Ping** – It is a utility that helps one to check if a particular IP address is accessible or not. Ping works by sending a packet to the specified address and waits for the reply. It also measures round trip time and reports errors.

**Traceroute** – It is a utility that traces a packet from your computer to the host, and will also show the number of steps (hops) required to reach there, along with the time by each step.

The main difference between Ping and Traceroute is that Ping is a quick and easy utility to tell if the specified server is reachable and how long will it take to send and receive data from the server whereas Traceroute finds the exact route taken to reach the server and time taken by each step (hop).

- ## Little and Big Endian Mystery

**What are these?**
Little and big endian are two ways of storing multibyte data-types ( int, float, etc). In little endian machines, last byte of binary representation of the multibyte data-type is stored first. On the other hand, in big endian machines, first byte of binary representation of the multibyte data-type is stored first.
Suppose integer is stored as 4 bytes (For those who are using DOS-based compilers such as C++ 3.0, integer is 2 bytes) then a variable x with value 0x01234567 will be stored as following.



Big Endian

Little Endian

```
unsigned int i = 1;
char* c = (char*)&i;
if (*c)
cout << "Little endian";
else
```

```
cout << "Big endian";
```

In the above program, a character pointer c is pointing to an integer i. Since size of character is 1 byte when the character pointer is de-referenced it will contain only first byte of integer. If machine is little endian then *c will be 1 (because last byte is stored first) and if the machine is big endian then *c will be 0.

#### Does endianness matter for programmers?

Most of the times ***compiler takes care of endianness***, however, endianness becomes an issue in following cases.
***It matters in network programming***: Suppose you write integers to file on a ***little endian*** machine and ***you transfer this file to a big-endian*** machine. Unless there is little endian to big endian transformation, big endian machine will read the file in reverse order. You can find such a practical example here.
Standard byte order for networks is big endian, also known as network byte order. Before transferring data on network, data is first ***converted to network byte order (big endian).***

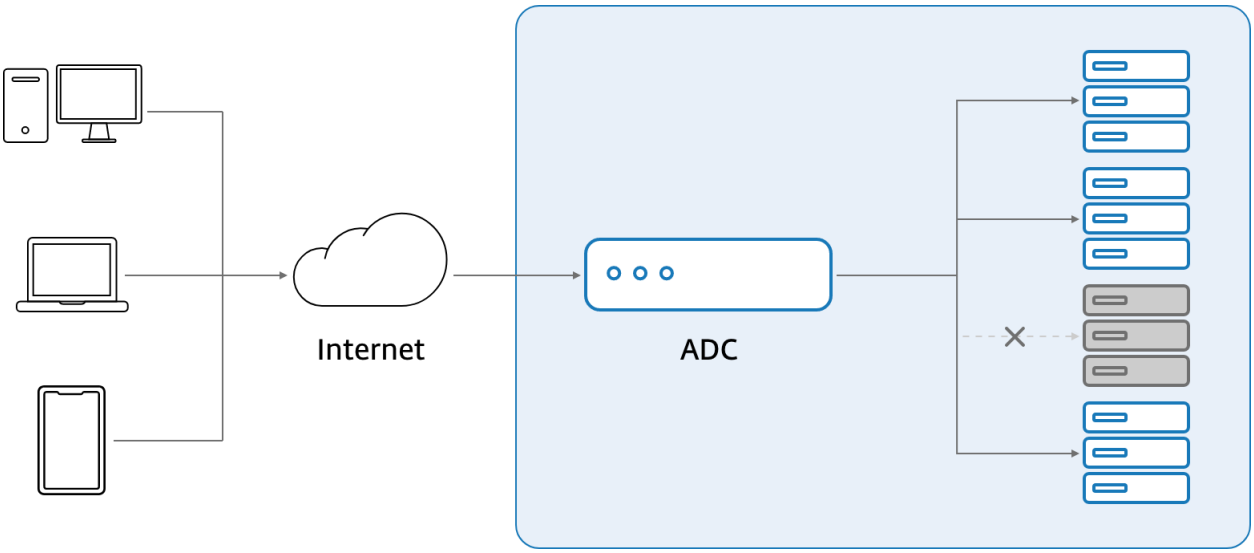
- ## Load Balancing

Load balancing is defined as the methodical and efficient distribution of network or application traffic across multiple servers in a server farm. Each load balancer sits between client devices and backend servers, receiving and then distributing incoming requests to any available server capable of fulfilling them.

What are load balancers and how they work?

A load balancer may be:

- A physical device, a virtualized instance running on specialized hardware or a software process.
- Incorporated into application delivery controllers (ADCs) designed to more broadly improve the performance and security of three-tier web and microservices-based applications, regardless of where they’re hosted.
- Able to leverage many possible load balancing algorithms, including round robin, server response time and the least connection method to distribute traffic in line with current requirements.




Regardless of whether it's hardware or software, or what algorithm(s) it uses, a load balancer disburses traffic to different web servers in the resource pool to ensure that no single server becomes overworked and subsequently unreliable. Load balancers effectively minimize server response time and maximize throughput.

Indeed, the role of a load balancer is sometimes likened to that of a traffic cop, as it is meant to systematically route requests to the right locations at any given moment, thereby preventing costly bottlenecks and unforeseen incidents. Load balancers should ultimately deliver the performance and security necessary for sustaining complex IT environments, as well as the intricate workflows occurring within them.

Load balancing is the most scalable methodology for handling the multitude of requests from modern multi-application, multi-device workflows. In tandem with platforms that enable seamless access to the numerous different applications, files and desktops within today's digital workspaces, load balancing supports a more consistent and dependable end-user experience for employees.

**Hardware- vs software-based load balancers**

Hardware-based load balancers work as follows:

- They are typically high-performance appliances, capable of securely processing multiple gigabits of traffic from various types of applications.
- These appliances may also contain built-in virtualization capabilities, which consolidate numerous virtual load balancer instances on the same hardware.
- That allows for more flexible multi-tenant architectures and full isolation of tenants, among other benefits.

In contrast, software-based load balancers:

- Can fully replace load balancing hardware while delivering analogous functionality and superior flexibility.
- May run on common hypervisors, in containers or as Linux processes with minimal overhead on bare-metal servers and are highly configurable depending on the use cases and technical requirements in question.
- Can save space and reduce hardware expenditures.

**What are some of the common load balancing algorithms?**

A load balancer, or the ADC that includes it, will follow an algorithm to determine how requests are distributed across the server farm. There are plenty of options in this regard, ranging from the very simple to the very complex.

- Round Robin

Round robin is a simple technique for making sure that a virtual server forwards each client request to a different server based on a rotating list. It is easy for load balancers to implement, but does don't take into account the load already on a server. There is a danger that a server may receive a lot of processor-intensive requests and become overloaded.

- Least Connection Method

Whereas round robin does not account for the current load on a server (only its place in the rotation), the least connection method does make this evaluation and, as a result, it usually delivers superior performance. Virtual servers following the least connection method will seek to send requests to the server with the least number of active connections.

- Least Response Time Method

More sophisticated than the least connection method, the least response time method relies on the time taken by a server to respond to a health monitoring request. The speed of the response is an indicator of how loaded the server is and the overall expected user experience. Some load balancers will take into account the number of active connections on each server as well.

- Least Bandwidth Method

A relatively simple algorithm, the least bandwidth method looks for the server currently serving the least amount of traffic as measured in megabits per second (Mbps).
Least Packets Method
The least packets method selects the service that has received the fewest packets in a given time period.

- Hashing Methods

Methods in this category make decisions based on a hash of various data from the incoming packet. This includes connection or header information, such as source/destination IP address, port number, URL or domain name, from the incoming packet.

- Custom Load Method

The custom load method enables the load balancer to query the load on individual servers via SNMP. The administrator can define the server load of interest to query – CPU usage, memory and response time – and then combine them to suit their requests.

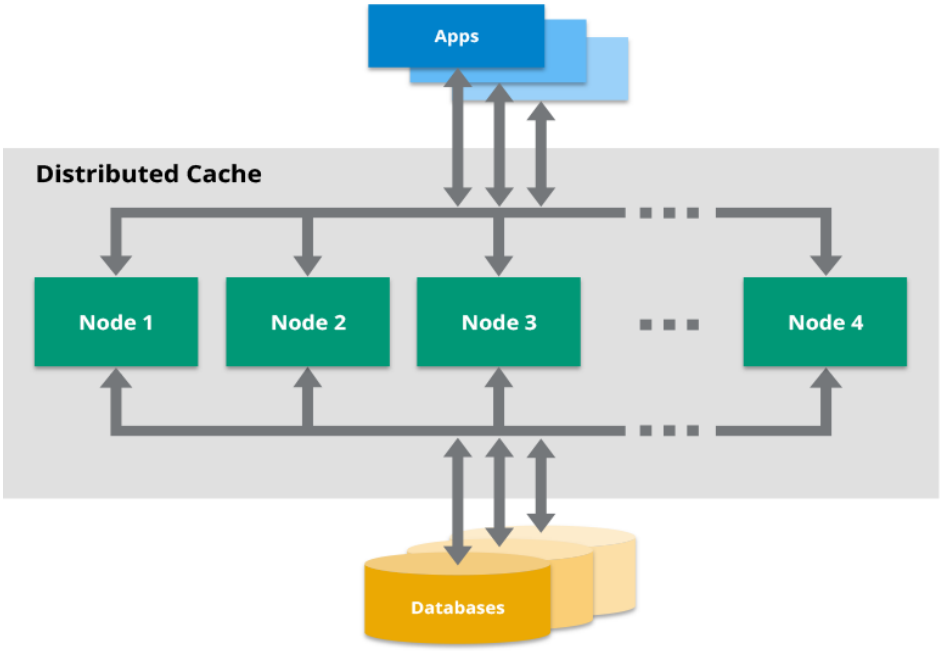
## Why is load balancing necessary?

An ADC with load balancing capabilities helps IT departments ensure scalability and availability of services. Its advanced traffic management functionality can help a business steer requests more efficiently to the correct resources for each end user. An ADC offers many other functions (for example, encryption, authentication and web application firewalling) that can provide a single point of control for securing, managing and monitoring the many applications and services across environments and ensuring the best end-user experience.

- ## What Is a Distributed Cache?

A **distributed cache** is a system that pools together the random-access memory (RAM) of multiple networked computers into a single in-memory data store used as a data cache to provide fast access to data. While most caches are traditionally in one physical server or hardware component, a distributed cache can grow beyond the memory limits of a single computer by linking together multiple computers–referred to as a distributed architecture or a distributed cluster–for larger capacity and increased processing power.

Distributed caches are especially useful in environments with high data volume and load. The distributed architecture allows incremental expansion/scaling by adding more computers to the cluster, allowing the cache to grow in step with the data growth.




### ❖ What Are Popular Use Cases for a Distributed Cache?

There are many use cases for which an application developer may include a distributed cache as part of their architecture. These include:

- Application acceleration. Applications that rely on disk-based relational databases can't always meet today's increasingly demanding transaction performance requirements. By storing the most frequently accessed data in a distributed cache, you can dramatically reduce the I/O bottleneck of disk-based systems. This ensures your applications run much faster, even with a large number of transactions when usage spikes.
- **Storing web session data**. A site may store user session data in a cache to serve as inputs for shopping carts and recommendations. With a distributed cache, you can have a large number of concurrent web sessions that can be accessed by any of the web application servers that are running the system. This lets you load balance web traffic over several application servers and not lose session data should any application server fail.

- **Decreasing network usage/costs**. By caching data in multiple places in your network, including on the same computers as your application, you can reduce network traffic and leave more bandwidth available for other applications that depend on the network.
- **Reducing the impact of interruptions**. Depending on the architecture, a cache may be able to answer data requests even when the source database is unavailable. This adds another level of high availability to your system.
- **Extreme scaling**. Some applications request significant volumes of data. By leveraging more resources across multiple machines, a distributed cache can answer those requests.