1. Introduction

Python Programming / Operating Systems

Team

Prof. Federico March

Computer Science

Prof. Marco Cascio

Computer Science

Dr. Flavio Giorgi

Computer Science

Dr. Leonardo Picchiami

Computer Science

Scope and Goal

- The basic concepts of programming
- Programming in Python
- The tools you need to program (as far as necessary)
- Basics of computer networks
- Operating systems and shell programming
- Gaining practical experience with all of the above
- Disclaimer: the actual work starts after this course!

Organization

- We provide you with:
  - Basics of computer networks and Operating systems (the theory)
  - Concepts and tools for Python program development (the theory)
  - Exercises and programming tasks (the practice)

- You need to:
  - Ask questions (at any time) and engage with your classmates! (we are a small group)
  - Work through the tasks (on your device)

- Course Material:
  - Moodle
  - Slides
  - Lecture Notes

Content

Python Programming

- Foundation of Programming
- Primitives (Data Types)
- Operations
- Collections
- Control Statements
- Functions
- Classes
- Threads
- Docker

Basics of Computer Networks

- Network Protocols
- ISO/OSI Model
- TCP/IP Model

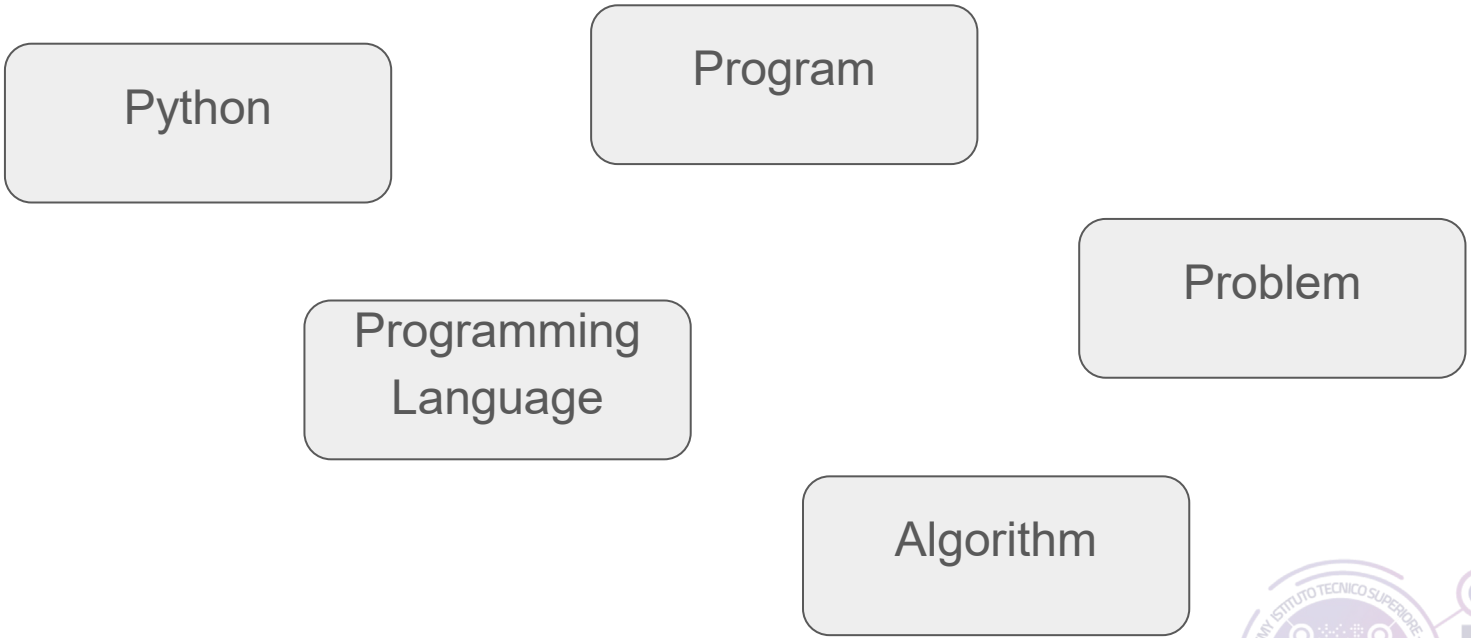
Operating Systems (OS)

- Binary Code
- Computer Architecture
- Fundamentals of OS
- OS services and functions
- Linux
  - Shell Programming

1.1 Getting Started

- Python Program
- Environment

Python Program



Problem 1/2

Description:
A **task** or **question** that requires a solution or answer is a **type of problem** that challenges you to think critically and apply your knowledge to solve it. This can range from a simple question to more complex problems that require **multiple steps** to find a solution.

Problem 2/2

Example:
Calculate the mean of some given numbers. For instance, if you are given the numbers 4, 8, 15, 16, 23, and 42, you would:

- Add them together: 4+8+15+16+23+42=108.
- Count the total numbers: 6.
- Divide 108 by 6: 108/6=18.

Algorithm 1/3

Description:
An **abstract step-by-step procedure** to solve a problem is often called an **algorithm**. Formally, it is a defined **sequence of instructions** or a **systematic approach** used to achieve a particular goal or solve a specific problem.
An algorithm is fundamental in both computer science and everyday problem-solving, as it helps **break down complex challenges into manageable steps** that can be followed to reach a solution.

Algorithm 2/3

A good algorithm is:

- **Clear:** each step is unambiguous, and there is no confusion about how to execute it.
- **Efficient:** it solves the problem in a reasonable amount of time without unnecessary steps.
- **Generalizable:** it can be applied to different sets of data, not just a specific example.

Algorithm 3/3

Example:
Let's take the example of calculating the mean of a set of numbers, and see how this can be expressed in a step-by-step algorithm:

1. Sum up all numbers.
2. Count all numbers.
3. Divide the sum by the count.

Program

Description:

A **set of instructions** telling your computer how to execute an algorithm.

Example:

```
sum = 0
count = 0

for number in numbers:
    sum = sum + number
    count = count + 1

mean = sum / count
```

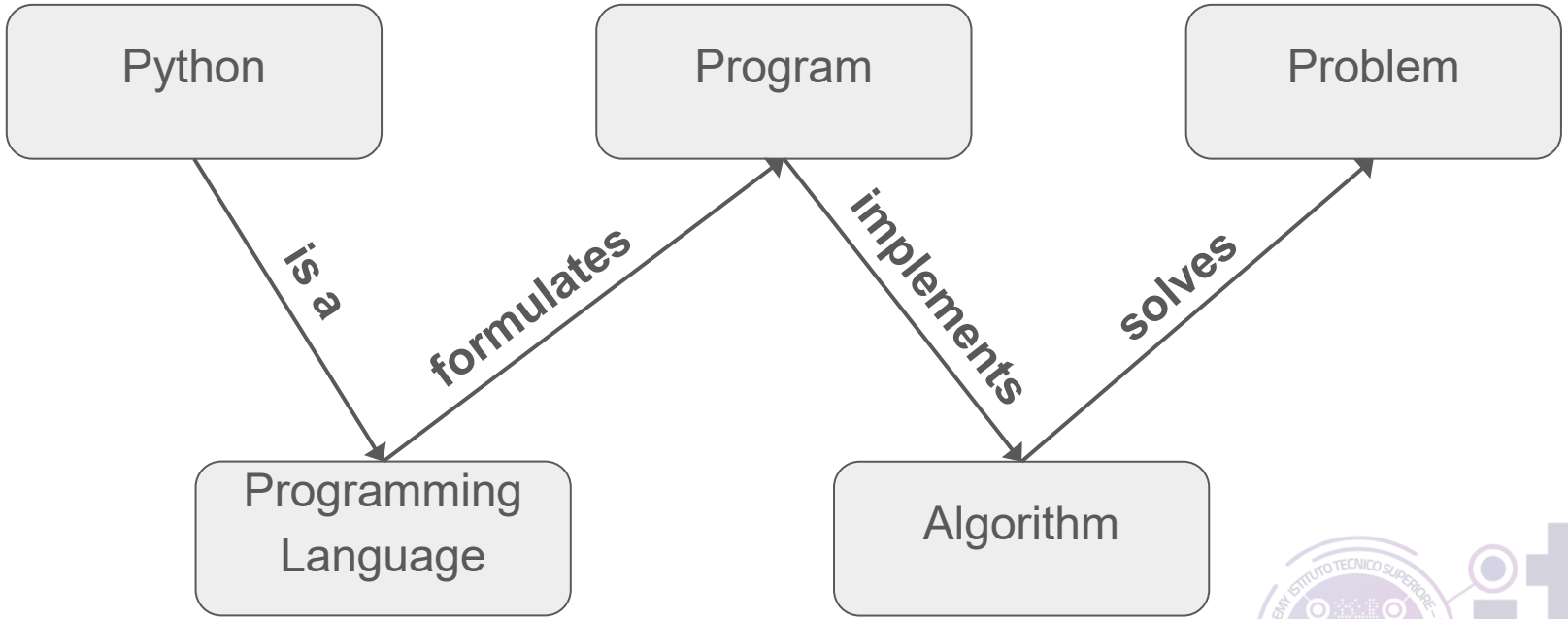
The **programm text** is the **program code**

Programming Language

Description:
A **specific language for formulating programs** is called a **programming language**. Programming languages are used by developers **to communicate instructions to a computer** in a way that it can understand and execute. They are the medium through which we create software, automate tasks, and solve computational problems.

Example:
Python, Java, or C++

Python Program



Environment 1/2

Permanent Storage
- Slow to access
- E.g. HDD, SSD, NAS, cloud storage, etc.

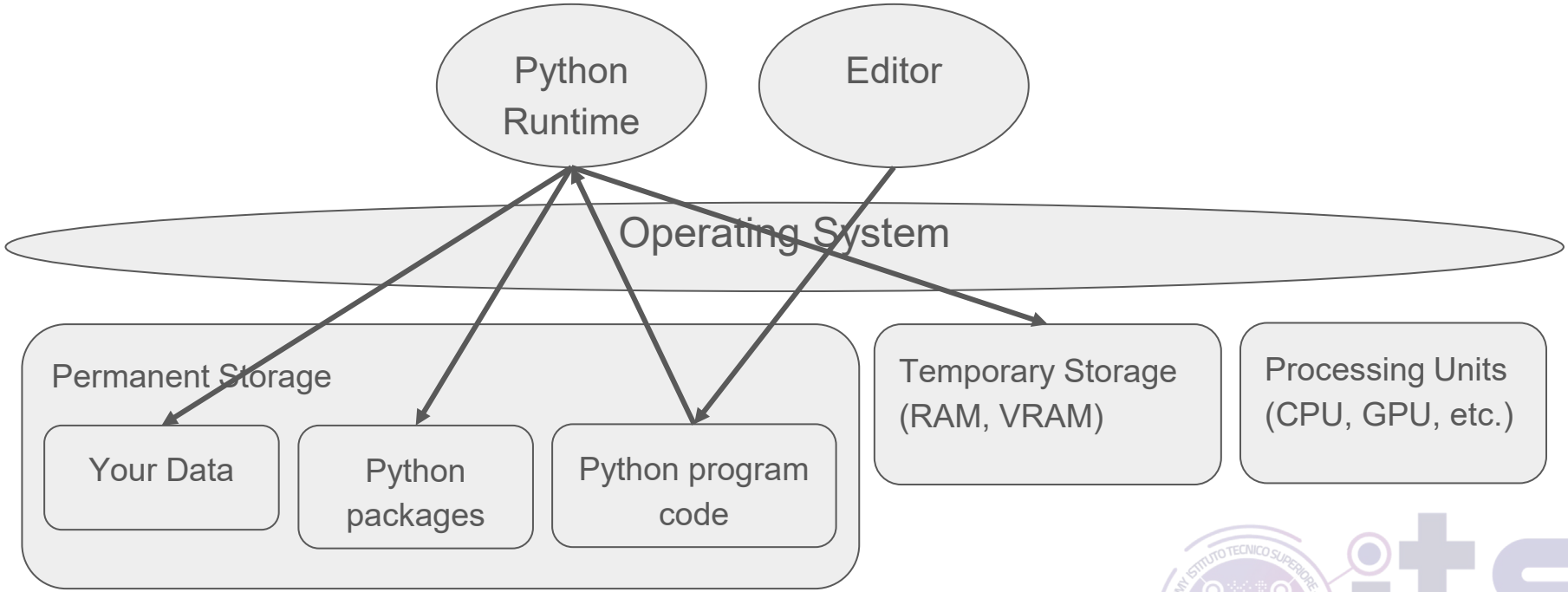
Your Data

Python program code

Processing Units
- Do the actual computation
- Central Processing unit (CPU)
- Graphics Processing Unit (GPU)

Temporary Storage
- Fast to access
- Only temporary
- E.g. RAM, VRAM

Environment 2/2

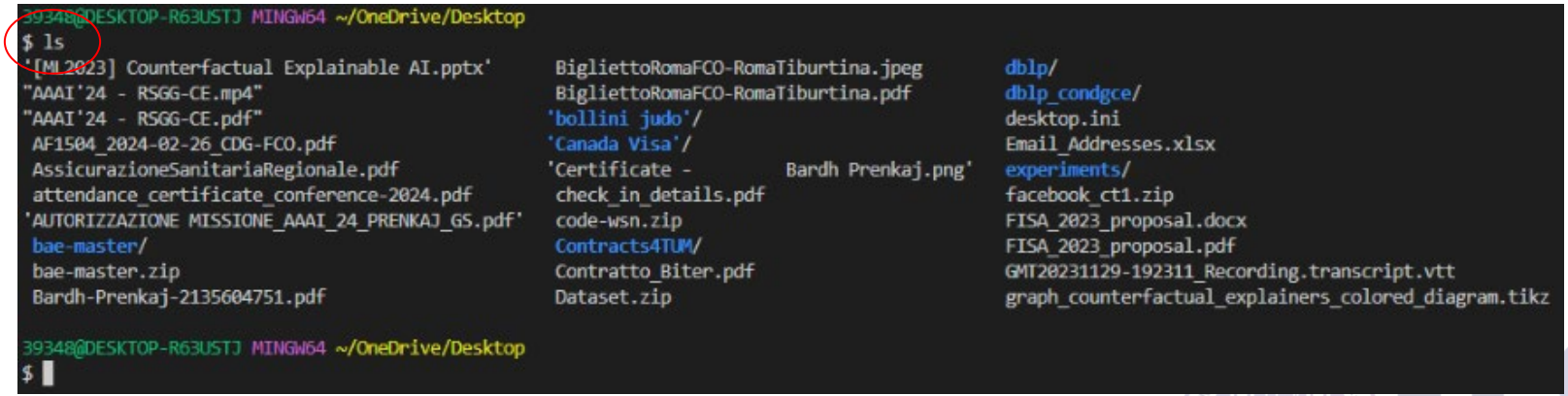


1.2 Setting up our tools: Command-line

- Commands:
  - ls
  - cd
  - mkdir
  - touch
  - rm
  - pwd
  - mv

Command-line

- With the **command-line** (cmd) you can communicate directly with your computer (i.e. operating system)
- Example: For listing the content inside a folder (`ls`)



Command-line - Try it yourself

1. Open command-line: (Use the shortcut **ctrl+alt+t**)
2. Go to your desktop: **`cd`** `Desktop`
3. Print your current working directory: **`pwd`**
4. Create now folder: **`mkdir`** `<name of your folder>`
5. Show your `Desktop` directory content:
    - Linux: `ls`
6. Remove folder: **`rm -r`** `<name of your folder>`

Command-line - Navigating directories

- Change your current directory (**working directory**) with `cd` <target>

```
39348@DESKTOP-R63USTJ MINGW64 ~/OneDrive/Desktop
$ cd dblp

39348@DESKTOP-R63USTJ MINGW64 ~/OneDrive/Desktop/dblp
$ mkdir cartella-di-prova
```

- For going up the folder structure use “..” as target

```
39348@DESKTOP-R63USTJ MINGW64 ~/OneDrive/Desktop/dblp
$ cd cartella-di-prova/

39348@DESKTOP-R63USTJ MINGW64 ~/OneDrive/Desktop/dblp/cartella-di-prova
$
```

```
39348@DESKTOP-R63USTJ MINGW64 ~/OneDrive/Desktop/dblp/cartella-di-prova
$ cd ..

39348@DESKTOP-R63USTJ MINGW64 ~/OneDrive/Desktop/dblp
$
```

Create your first “empty” Python file

You can do that using the command **`touch`** <*file_name*>

```
39348@DESKTOP-R63USTJ MINGW64 ~/OneDrive/Desktop/dblp/cartella-di-prova
$ touch first_python_program.py

39348@DESKTOP-R63USTJ MINGW64 ~/OneDrive/Desktop/dblp/cartella-di-prova
$ ls
first_python_program.py

39348@DESKTOP-R63USTJ MINGW64 ~/OneDrive/Desktop/dblp/cartella-di-prova
$
```

Command-line - Moving and renaming files

The **mv** command is used to move files or directories to a new location or rename files.

- **mv** `file.txt /path/to/destination/`

  This moves file.txt into the specified directory (/path/to/destination)

- **mv** `old_name.txt new_name.txt`

  This renames old_name.txt to new_name.txt

- **mv** `file.txt /new/path/renamed_file.txt`

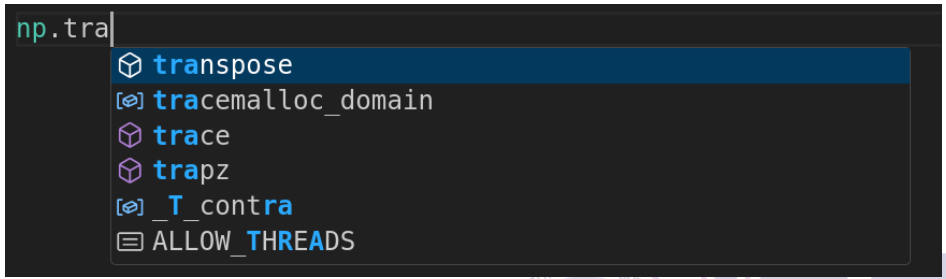
  This moves file.txt to a new directory and renames it

1.3 Setting up our tools: Integrated Development Environment (IDE)

- IDE
- VSCode

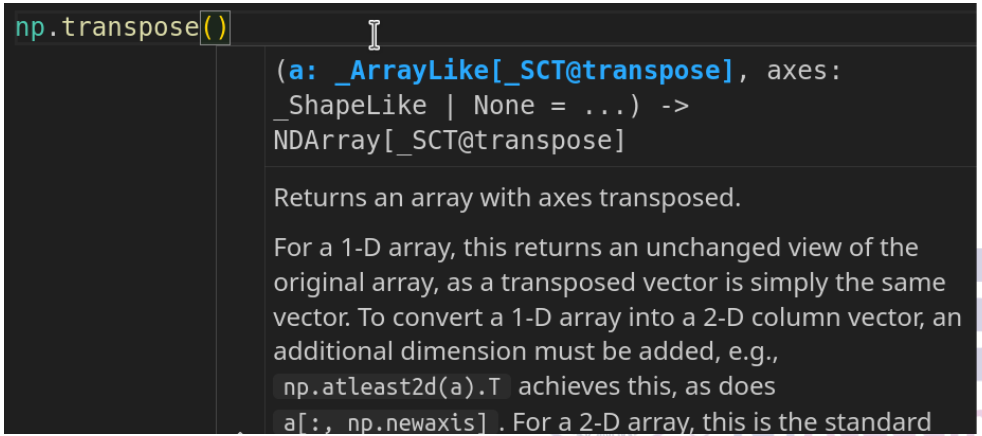
Visual Studio Code - Programming more comfortably!

- **Code editors** are great for developing programs!
  - Visual Studio code is actually more like a *Integrated Development Environment* (IDE)
- Write code and **save** it as a file
- **Highlight errors** in code
- **Autocomplete** and recommendations



Visual Studio Code - Programming more comfortably!

- **Code editors** are great for developing programs!
  - Visual Studio code is actually more like a *Integrated Development Environment* (IDE)
- Write code and **save** it as a file
- **Highlight errors** in code
- **Autocomplete** and recommendations
- **Documentation** and references

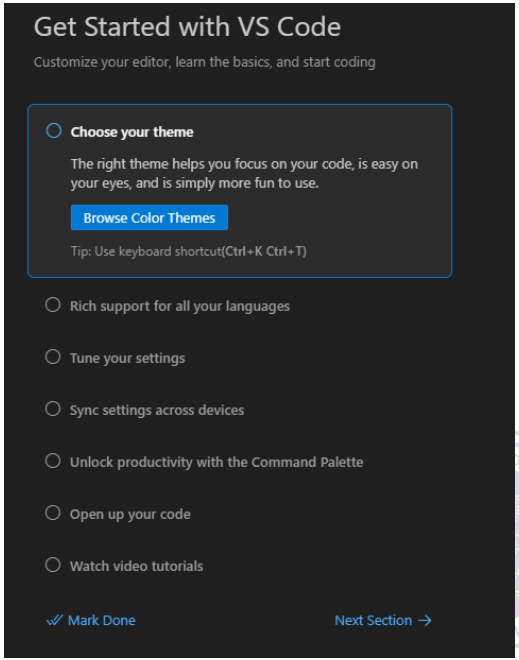


Visual Studio Code - Programming more comfortably!

- **Code editors** are great for developing programs!
  - Visual Studio code is actually more like a *Integrated Development Environment* (IDE)
- Write code and **save** it as a file
- **Highlight errors** in code
- **Autocomplete** and recommendations
- **Documentation** and references
- Code **versioning** control with Git → more on Git later!

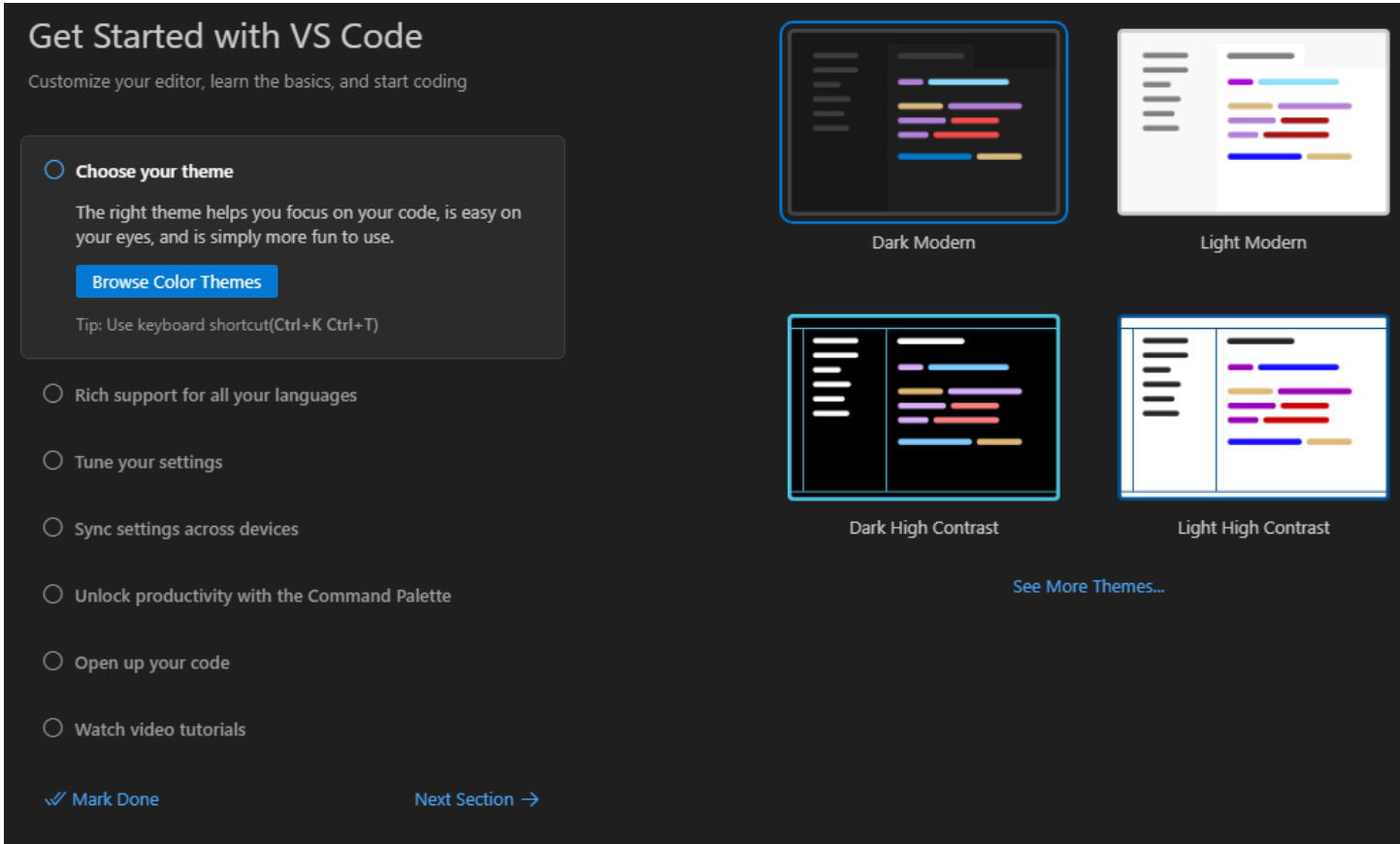
Visual Studio Code - Installation

- Go to code.visualstudio.com
- **Download** installer & **Install** (should be similar for Windows & Mac)
  - Select **all** additional tasks
- **Wait** for the others if you are at "Get Started with VS Code" (We will do the setup together)



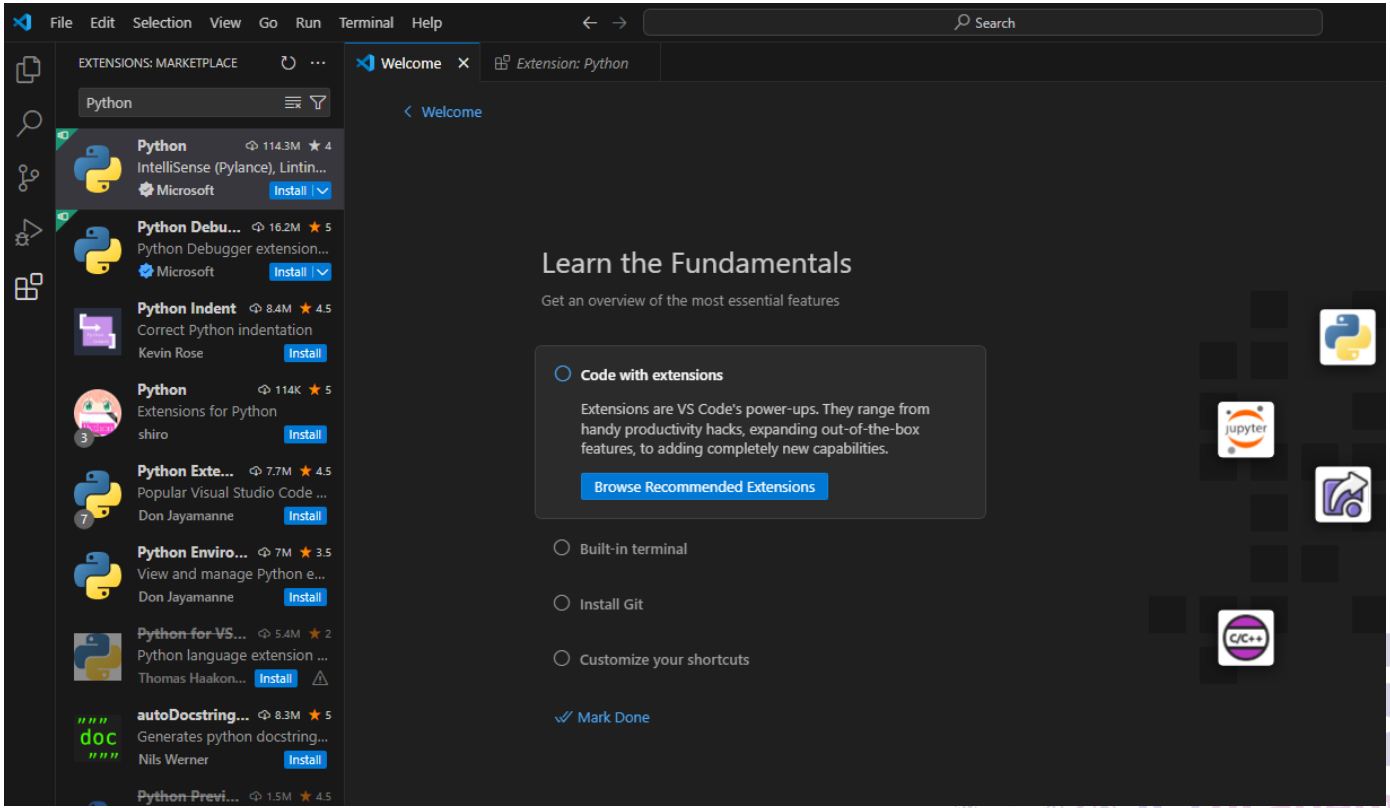
Visual Studio Code - Setup - Theme

1. **Choose** Theme
2. Click “**Next Section**”

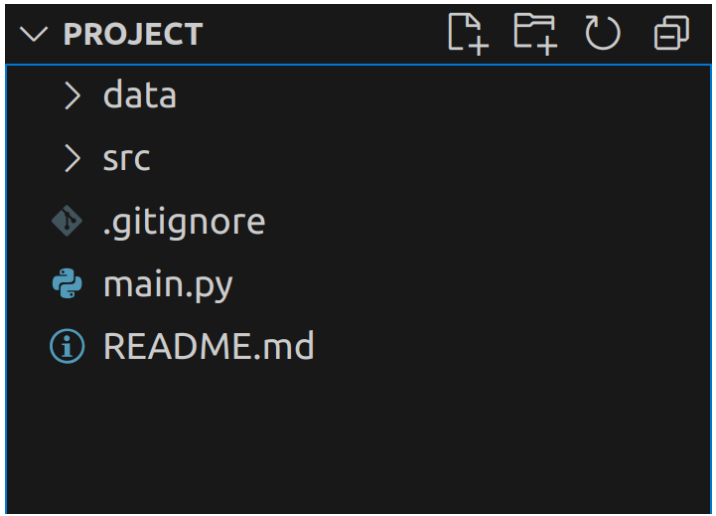


Visual Studio Code - Setup - Python Extension

1. Click “**Browse Recommended Extensions**”
2. Type “**Python**” in the extension **search** bar
3. **Install** the “**Python**” extension
4. If you get the “Get Started with Python Development” Tab opens; **wait** for the others

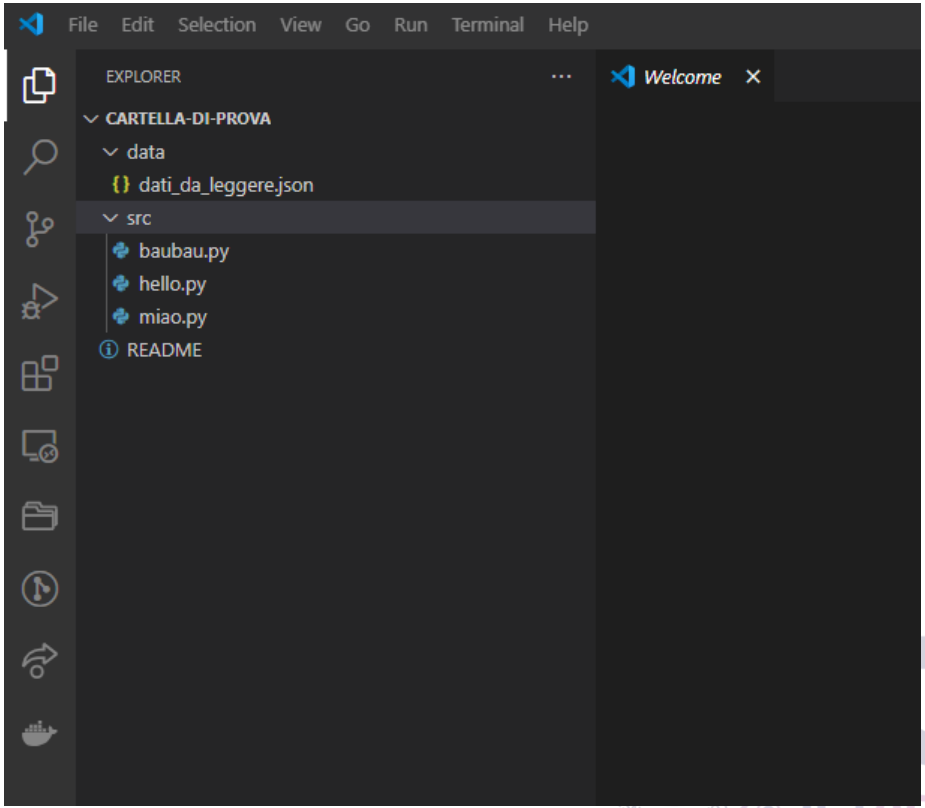


Exercise: Let’s create a “standardized” Python project structure using: mkdir, cd, touch, ls



Visual Studio Code - Setup - Open the project

1. Click “**File > Open Folder**”
2. Choose the parent folder you just created
3. Open the file `hello.py` and write `print(“Hello World”)`



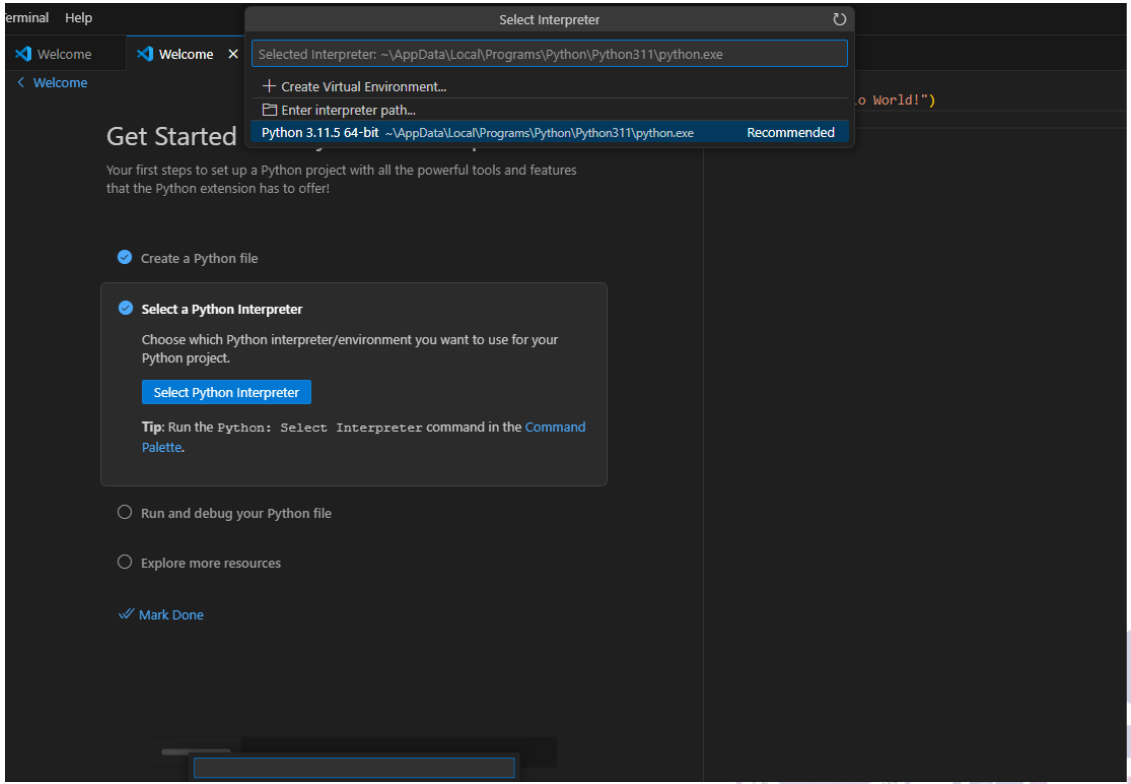
Visual Studio Code - Setup - Running hello.py

1. Click “**Terminal > New terminal**”
2. Navigate to the parent folder of `hello.py`
3. Once there, write `python hello.py`

```
39348@DESKTOP-R63USTJ MINGW64 ~/OneDrive/Desktop/dblp/cartella-di-prova/src
$ python hello.py
Hello World
```

Visual Studio Code - Setup - Python environment

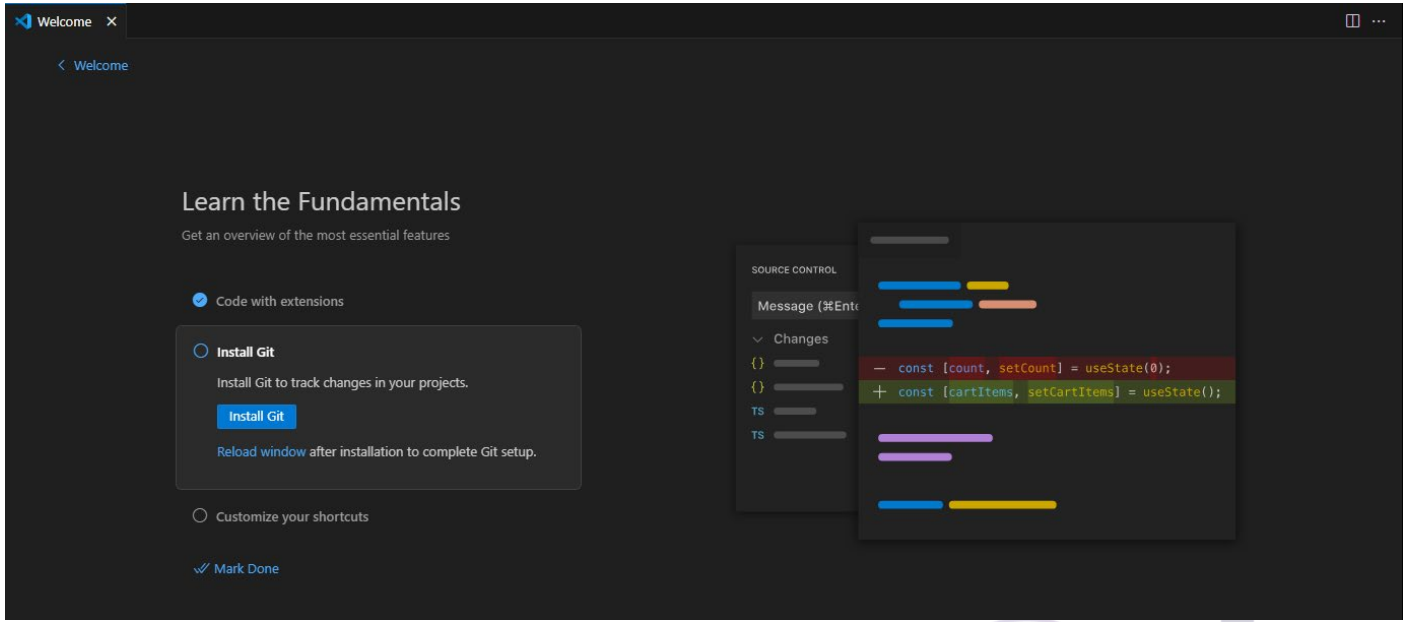
1. Click “**Create Python File**”
2. **Write** in file:
   `print(“Hello World!”)`
3. Go to “**Select** a Python **Interpreter**”
4. **Select** your python **version**.
5. **Go back** into the python **file**



Visual Studio Code - Setup - Git Installation

1. Click “**Install Git**”
2. **Follow** the **instructions** on the pop-up Website.

A. **Download** & **execute** Installer
B. Use **default** settings.
C. **Use VScode** as “**default editor**”
D. Otherwise default options again (there are many)



3. Click “**Reload window**” in VScode

1. Introduction

Understanding Block Diagrams in Algorithms

Algorithm

An **algorithm** is the description of the **problem-solving path** to reach the final results starting from initial data.

We write the algorithm as if addressing an **executor**, capable of performing actions described as **instructions**, written in a specific **language**.

We assume that the executor is available to a user (not necessarily the person who wrote the algorithm) who uses the execution of the algorithm.

Instructions for	Purpose
input	receive data (numbers, expressions, texts...)
output	send messages and communicate results
assignment	store a piece of data by associating it with a variable name
calculation	perform operations on data

Fundamental Structures

Sequence

Start
<instruction>
<instruction>
<instruction>
...
End

Selection

IF <condition>
    THEN
        <instructions1>
    ELSE
        <instructions2>

or

IF <condition>
    THEN
        <instructions1>

Iteration

REPEAT
    <instructions>
UNTIL <condition>

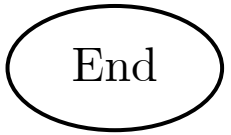
or

WHILE <condition> DO
    <instructions>
END WHILE

or

FOR <index> FROM <start> TO <end>
    DO<instructions>
NEXT <index>

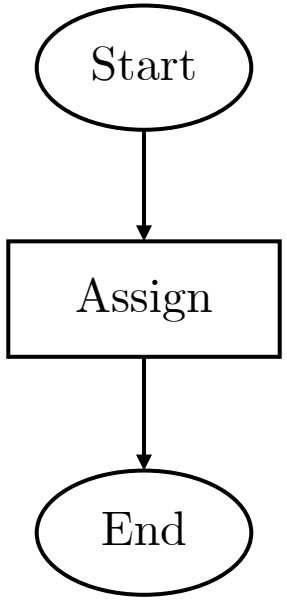
Block Diagram

Opening of the algorithm	Closing of the algorithm	Reading input data (input)	Communicating messages and/or results (output)	Assigning data and/or performing calculations	Controlling the truth value of a condition
					

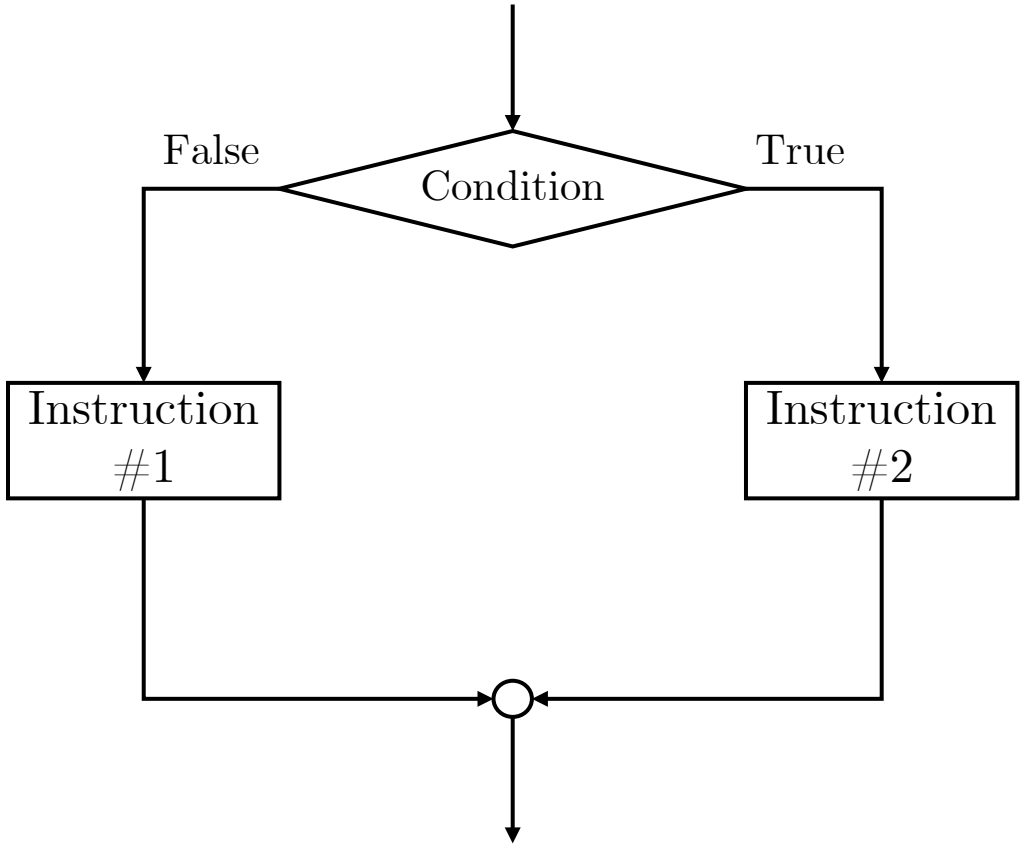
The **three fundamental structures** of algorithms can therefore be described using **flowcharts**. The blocks are represented and connected by arrows, which indicate the flow of execution of the algorithm.

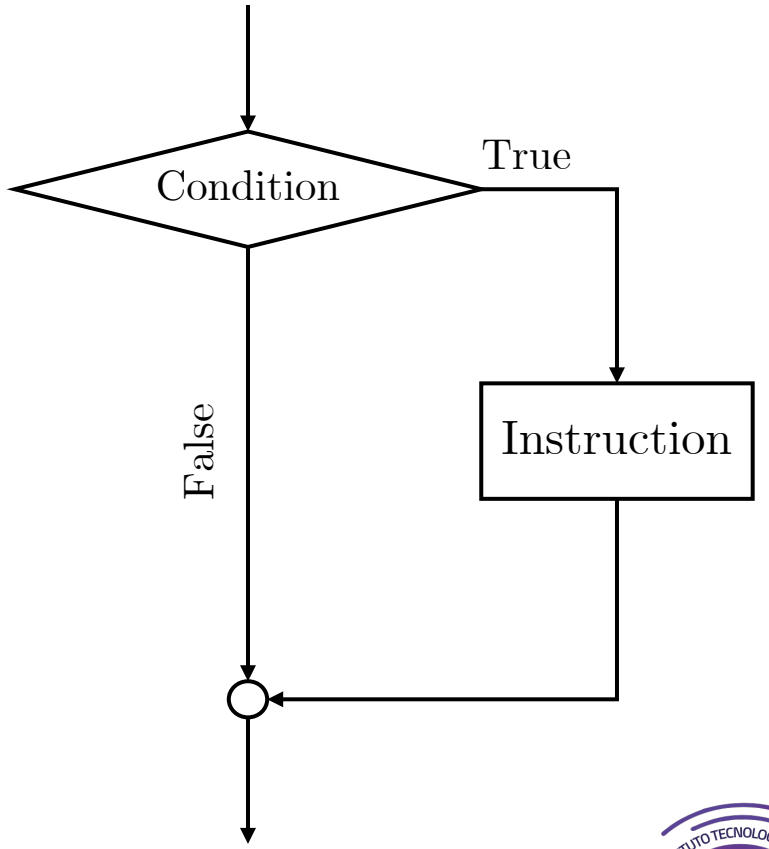
Sequence

The **opening** and **closing blocks** of the algorithm have only **one arrow**, respectively outgoing and incoming, while the **intermediate blocks** generally have **two arrows**: one incoming arrow and one outgoing arrow.

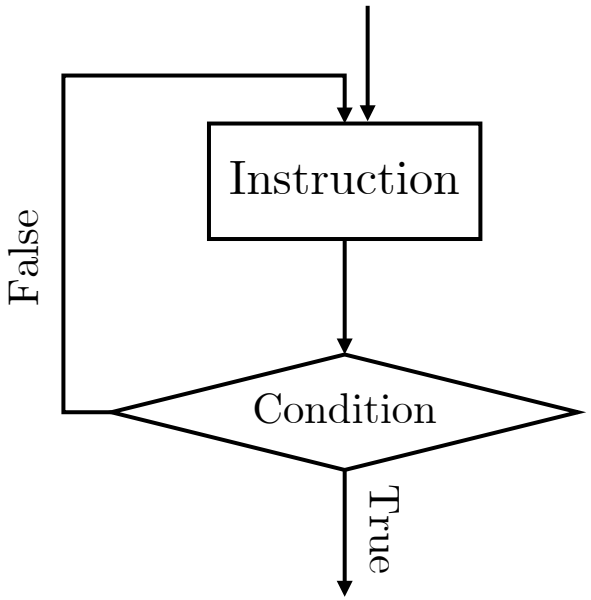


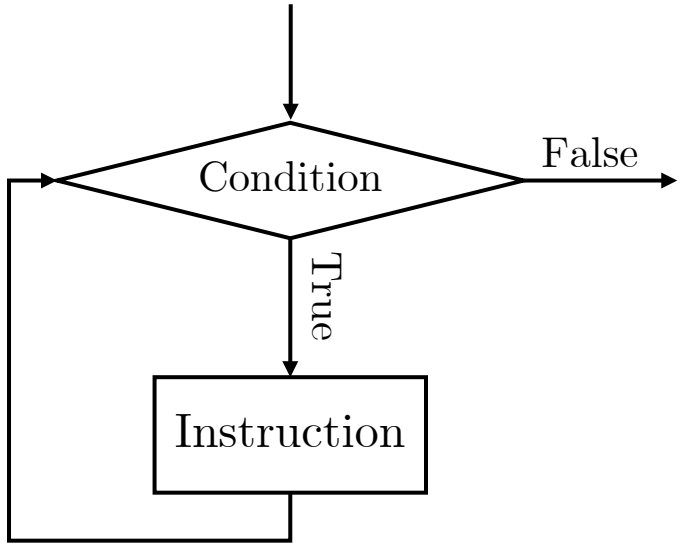
Selection Block

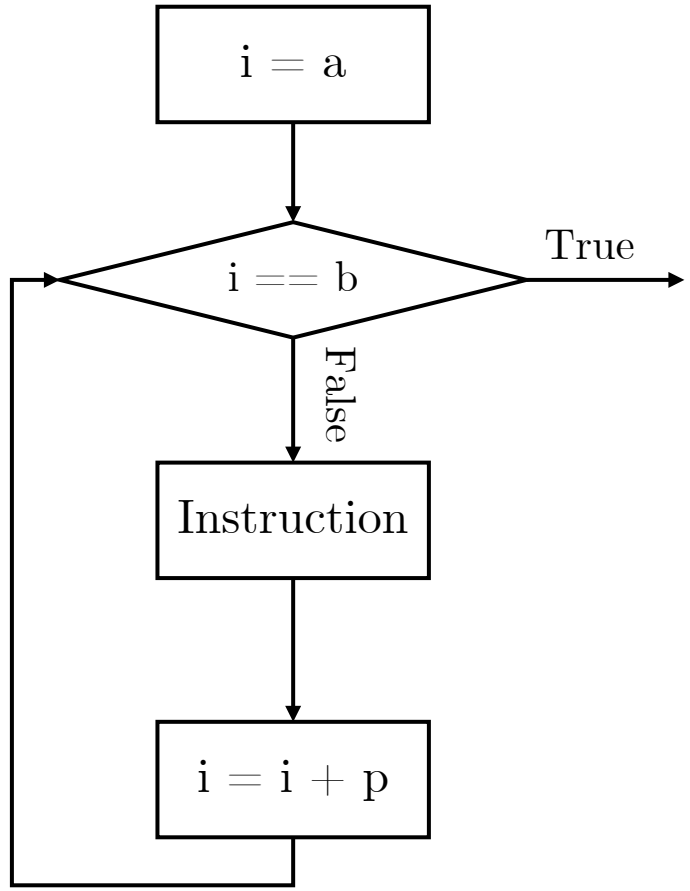




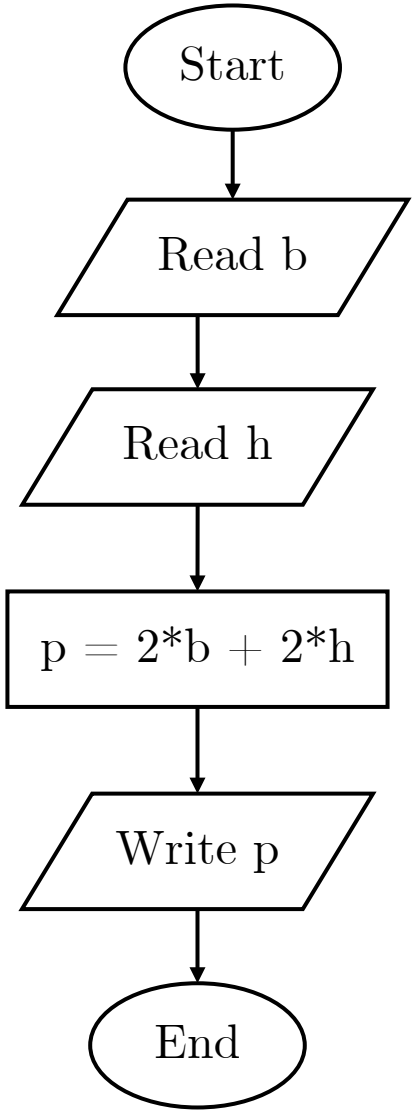
Control Block







Example: perimeter of a rectangle



1. Computer Networks

Foundations and Key Principles

Summary

- Introduction
- Computer Network
- Usage of Computer Networks
- Computer Network Communication
- Computer Network Topology
- Client-Server and Peer-to-Peer (P2P) Models
- Types of Computer Networks

Introduction/1

Evolution of Networking Devices:

- **Mainframes to Personal Computers:** Early reliance on large, centralized systems to modern-day ubiquitous personal computing devices.
- **Rise of Mobile Technology:** Transition from static desktops to mobile smartphones and tablets enabling anytime, anywhere connectivity.

EARLIER

TODAY

Introduction/2

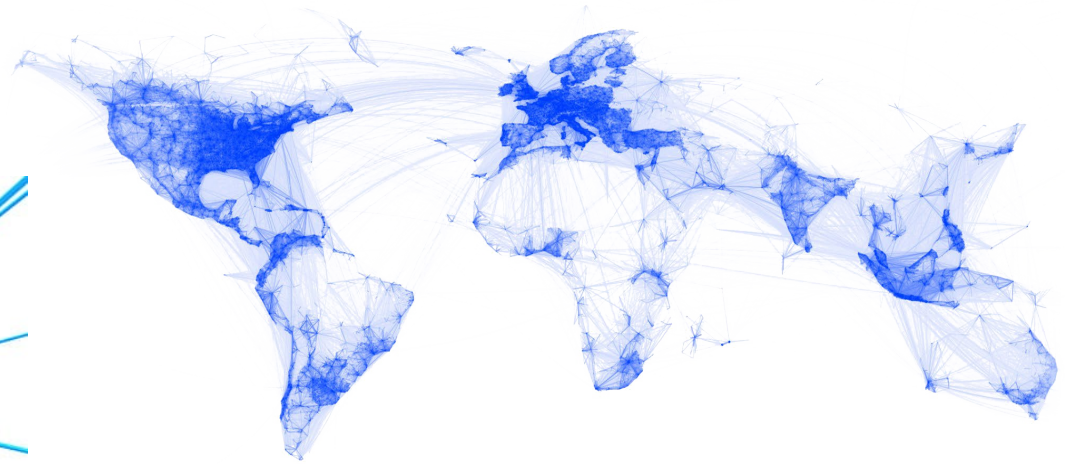
Proliferation of Internet Services:

- **Expansion of Streaming Services:** Impact of high-speed internet on entertainment; from traditional broadcasting to on-demand streaming platforms like Netflix and YouTube.
- **Social Media Networks:** Evolution from basic communication tools to complex platforms driving social interactions, business, and news dissemination.

Introduction/3

Impact on Society:

- **Digital Inclusion and the Global Village:** The role of advanced networking in bridging geographical and socio-economic divides.
- **Challenges:** Addressing cybersecurity threats and privacy concerns in an increasingly interconnected world.



Computer Network

A computer network is a group of interconnected computer systems and other computing devices that communicate with each other to share resources and information. It allows multiple users to exchange data, access shared applications, and utilize common hardware like printers and servers.

Networks can range from simple connections between two computers to vast global networks like the **Internet**. They typically use various technologies, protocols, and physical infrastructures to facilitate seamless communication and data transfer among connected devices.

Usage of Computer Networks/1

Business Applications:

- **Resource Sharing:** Computer networks enable the shared use of essential devices across different computers within a network. This includes printers, scanners, and fax machines, allowing multiple users to utilize the same physical resources efficiently, reducing costs and improving productivity.
- **Information Sharing:** Networks facilitate the seamless exchange of information between individuals, organizations, and technologies. This capability is crucial for collaboration and decision-making, as it ensures that relevant data can be accessed and utilized from multiple locations and platforms.
- **Communication Medium:** Networks serve as vital communication channels. They support various communication tools such as email, video conferencing, and instant messaging, which are essential for modern business operations. These tools help in maintaining effective communication among employees, clients, and partners, regardless of their physical locations.
- **E-commerce:** Networks are fundamental to e-commerce activities. They enable transactions such as online payments, electronic fund transfers, and management of digital marketplaces. This digital infrastructure is key to conducting business in the global market, providing a platform for buying and selling products and services online.

Usage of Computer Networks/2

Home Applications:

- **Remote Information Access:** Computer networks enable remote access to personal and professional information from virtually any location, including one's home. This flexibility allows individuals to work from anywhere, accessing work files, databases, and applications through secure connections like VPNs, enhancing productivity and work-life balance.
- **Person-to-Person Communication:** Networks are crucial in facilitating direct communication between individuals using various methods such as phone calls, text chats, and video calls. These tools help maintain personal relationships and professional collaborations across distances, reducing the need for physical travel and enabling more dynamic and immediate communication.
- **Interactive Entertainment:** Computer networks play a pivotal role in the realm of entertainment, connecting users with a wide array of interactive content. This includes social networking sites that allow for sharing and communication among users, online gaming platforms that enable multiplayer gaming experiences, and streaming services that provide real-time access to movies, music, and television shows. These platforms not only entertain but also create communities and bring people with similar interests together.

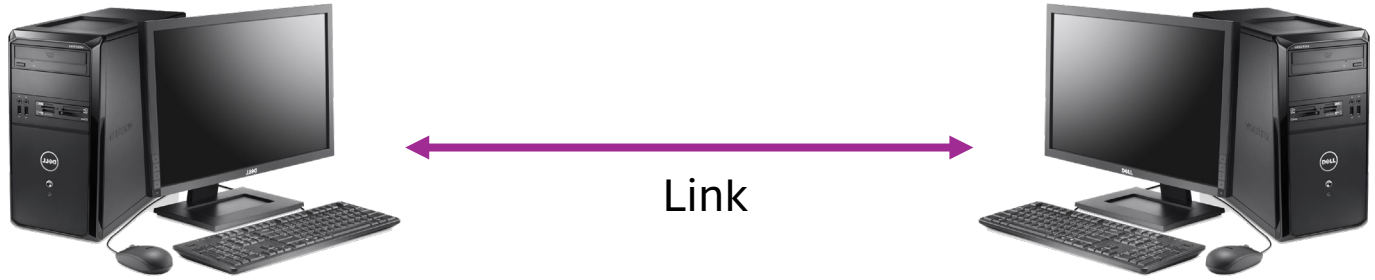
Usage of Computer Networks/3

Mobile Applications:

- **M-commerce:** This form of commerce allows the buying and selling of goods and services through wireless handheld devices, such as smartphones and tablets. M-commerce utilizes mobile networks to provide convenient and fast transactions from anywhere, enabling consumers to shop, manage finances, and access service on-the-go. This includes mobile payments, banking, and shopping through dedicated apps and mobile-friendly websites, thereby enhancing user convenience and broadening market reach for businesses.
- **GPS Navigation System:** Navigation systems use network technology to provide location and time information under all weather conditions and anywhere. These systems offer turn-by-turn navigation instructions to help users reach their destinations efficiently. They are integrated into mobile devices and vehicles to assist with route planning, traffic monitoring, and location tracking, improving travel safety and convenience.
- **Instant Messaging:** Networks support a variety of instant messaging services that allow users to send and receive messages in real-time. These services extend beyond text messaging to include voice calls and the sharing of multimedia files, such as images and documents. Instant messaging has revolutionized communication, offering a quick, cost-effective, and versatile method of staying connected personally and professionally across the globe.

Computer Network Communication/1

- **Point-to-point:** a direct link between two devices.



- **Broadcast:** a link shared among multiple devices.

Computer Network Communication/2

- **Point-to-Point:**

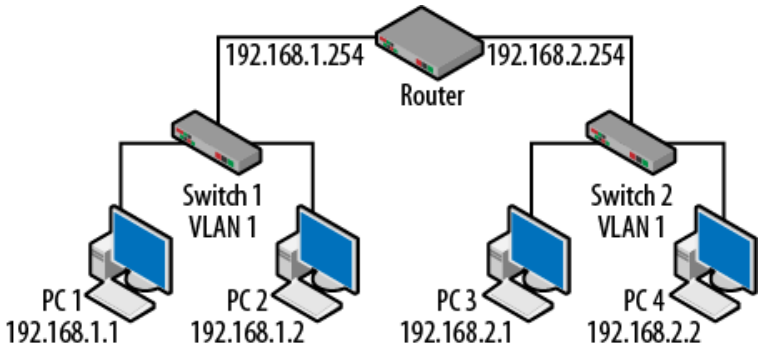
This network is structured through a series of links and nodes.

A node can be any device like a PC, a switch, or a router.

A switch serves to connect devices within the same network, whereas a router links different networks together.

Data packets are transmitted from a source to a designated destination, potentially traversing multiple nodes along the path.

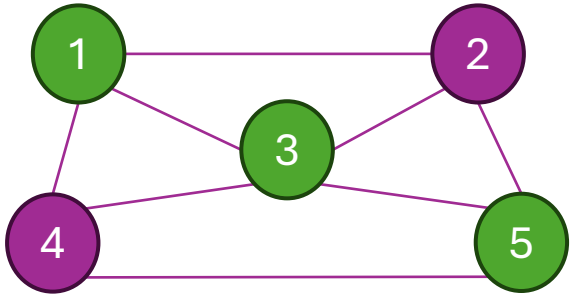
Only the intended recipient, or destination node, processes the message.



Computer Network Communication/3

- **Point-To-Point - Example:**

Let's consider a network consisting of 5 PCs:



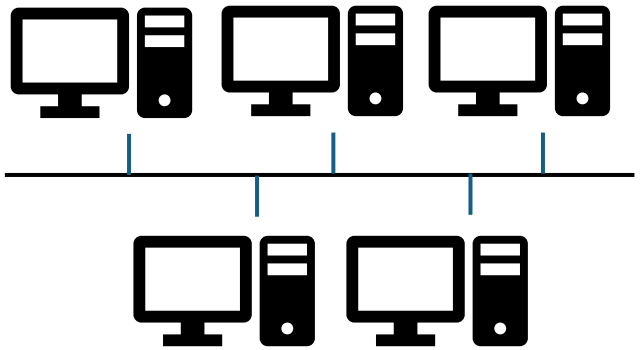
- PC1 wants to send a message to PC5. Several paths are available, e.g., (1,4,5) or (1,2,5).
- The best path selection (i.e., the routing process) can be 'length-based' or 'traffic-based'.
  - If 'length-based', the shortest path is chosen.
  - If 'traffic-based', the less used path is chosen.
- Let's suppose the best path is (1,3,5), i.e., the green path.
  - PC1 sends the message to PC3, which then forwards it to PC5.
  - **What happens if PC3 is offline?** Another path is selected.

Computer Network Communication/4

- **Broadcast:**

A single communication channel is utilized by all computers on the network.

Generally, when any computer sends a message, it broadcasts it to all connected computers.

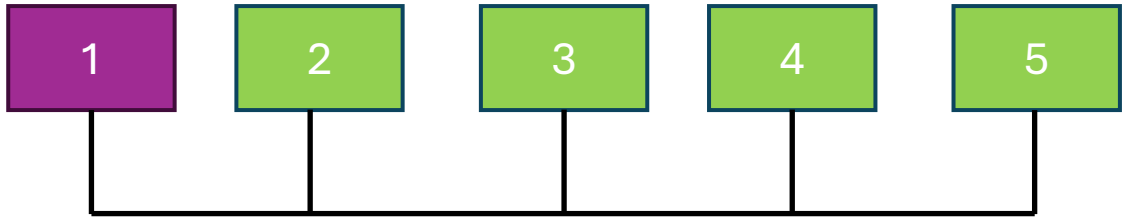


- **Broadcast:** The message is intended for all computers on the network.
- **Multicast:** The message is directed to a specific group of computers within the network.
- **Anycast:** The message can be received by any one of the computers in the network, typically the one nearest to the source.
- **Unicast:** The message is distributed to every computer, only the recipient specified in the message's address field processes it.

Computer Network Communication/5

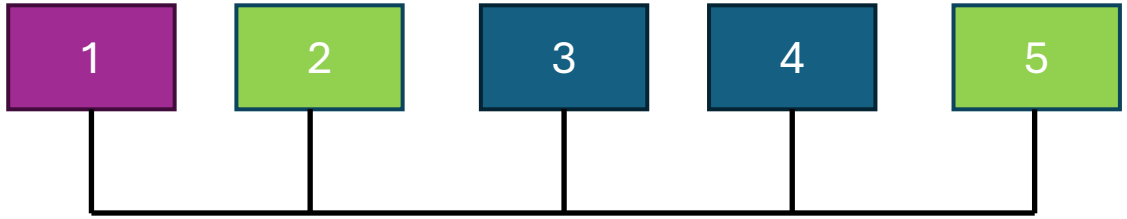
- **Broadcast - Example:**

PC1 sends a message that is received by all PCs.



- **Multicast – Example:**

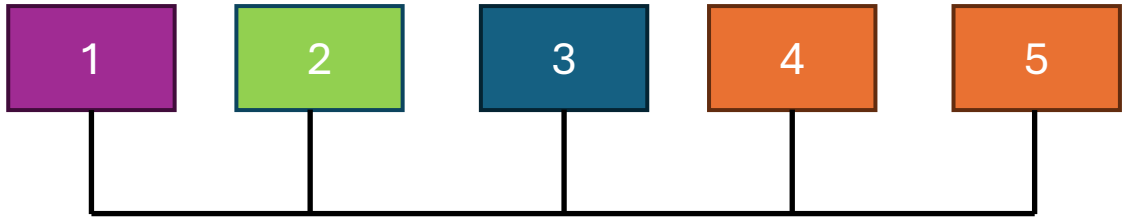
PC1 sends a message that is received by the specified group of PCs (i.e., PC2 and PC5).



Computer Network Communication/6

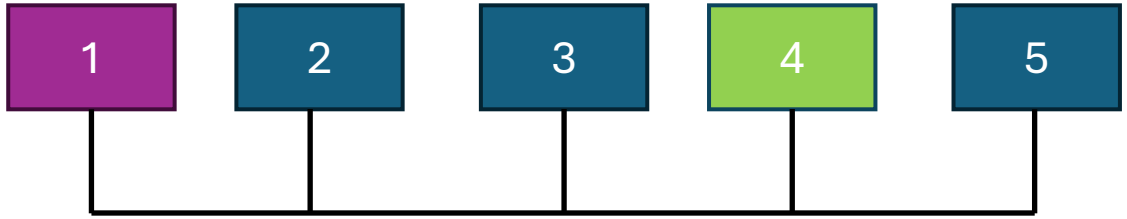
- **Anycast - Example:**

PC1 sends a message that is received by the nearest PC (i.e., PC2) out of a group of potential receivers (i.e., PC4 and PC5).



- **Unicast – Example:**

PC1 sends a message that is received by the specified PC (i,.e., PC4).



Computer Network Topology/1

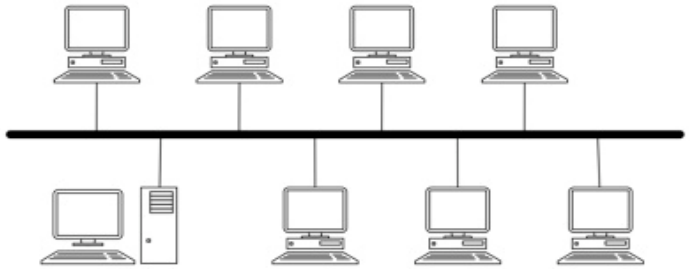
Computer network topology refers to the arrangement of elements (including nodes and connecting lines) within a computer network. It can be visualized as the physical or logical arrangement of how devices like computers, routers, and switches interconnect.

There are 5 different topologies:

- Bus;
- Ring;
- Star;
- Mesh;
- Tree.

Computer Network Topology/2

Bus Topology: All devices share a single communication line or cable. Signals from the source travel in both directions to all machines connected on the network until they find the intended recipient.



Advantages:

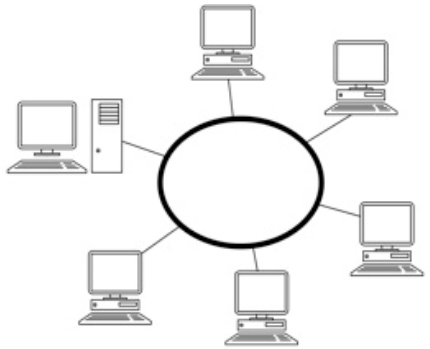
- Cost-effective: Requires fewer cables compared to other topologies.
- Suitable for small networks.
- Straightforward to understand.
- Easily expandable by connecting additional cables.

Disadvantages:

- Single point of failure: If the main cable fails, the entire network goes down.
- Performance degradation: The network slows down as the number of nodes increases or under heavy traffic conditions.
- Limited cable length: The reach of the network is restricted by the cable length.

Computer Network Topology/3

Ring Topology: Devices are connected in a circular format, with each computer linked to two other computers. This setup forms a continuous pathway for signals through each device, with data traveling around the ring in one direction. Data transfer requires a token.



Advantages:

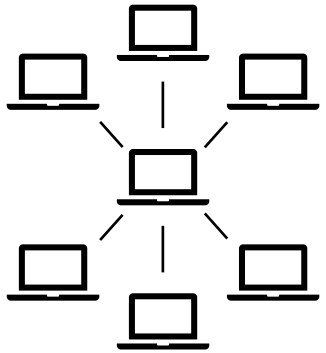
- Consistent performance: Maintains efficiency under high traffic or with many nodes, as nodes transmit data only when they possess the token.
- Cost-effective: Inexpensive to set up and expand.

Disadvantages:

- Complex troubleshooting: Diagnosing issues within this topology can be challenging.
- Disruptive maintenance: Adding or removing computers can interrupt network activity.
- Dependent stability: The failure of a single computer can impact the entire network.

Computer Network Topology/4

Star Topology: Devices are connected to a central hub. Data on a star network passes through the hub before continuing to its destination. The hub acts as a signal repeater.



Advantages:

- Scalability: The network can be expanded with ease.
- Simple troubleshooting: Faults can be diagnosed quickly.
- Resilience of non-central nodes: The failure of individual non-central nodes does not disrupt the entire network.

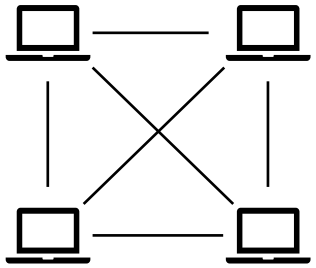
Disadvantages:

- Vulnerability of the central node: The entire network is compromised if the central node fails.
- High load on central node: All network traffic must pass through the central node, which can become a bottleneck.

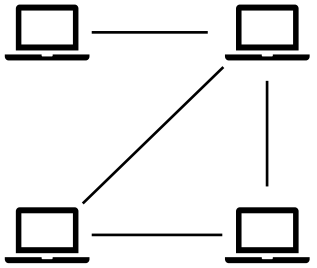
Computer Network Topology/5

Mesh Topology: Devices are interconnected with many redundant interconnections between network nodes. In a full mesh topology, every computer connects directly to every other computer. In a partial mesh topology, some computers connect indirectly through others.

FULL:



PARTIAL:



Advantages:

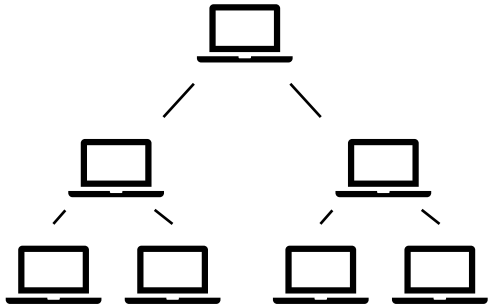
- Independent data paths: Each connection can independently handle its own data traffic.
- High fault tolerance: The network remains operational despite the failure of individual connections.
- Easy fault identification: Problems within the network can be diagnosed swiftly.

Disadvantages:

- Complex setup: Installation and configuration are challenging.
- High cabling costs: Significant expense due to the amount of cabling required.
- Extensive wiring: A large volume of wiring is necessary to connect each node independently.

Computer Network Topology/6

Tree Topology: A variation of the star network setup that branches off other stars. It integrates multiple star topologies together onto a bus, combining the characteristics of both.



Advantages:

- Hybrid structure: Combines elements of bus and star topologies.
- Scalability: Adding new nodes is feasible and straightforward.
- Manageability: The network is easy to oversee and maintain.
- Effective error detection: Issues within the network can be identified promptly.

Disadvantages:

- Extensive cabling: Requires a lot of wiring.
- Elevated cost: The setup and maintenance can be expensive.
- Dependence on root node: The entire network is compromised if the root node fails.
- Maintenance complexity: Managing the network becomes more challenging as more nodes are added.

Client-Server Model

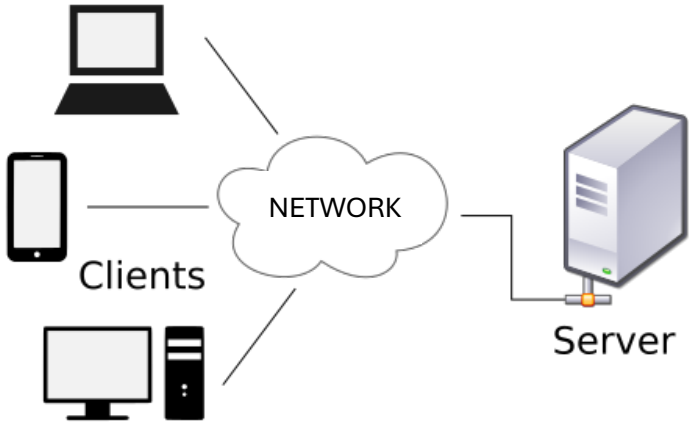
A computing architecture where multiple clients (devices or software) request and receive services from a centralized server. This model centralizes core functions and resources, offering efficiency and scalability by allocating tasks between providers (servers) and requesters (clients).

Client:

- Requests services or resources.

Server:

- Provides resources and services.
- Manages all processes and data storage.
- Sends output back to the client.

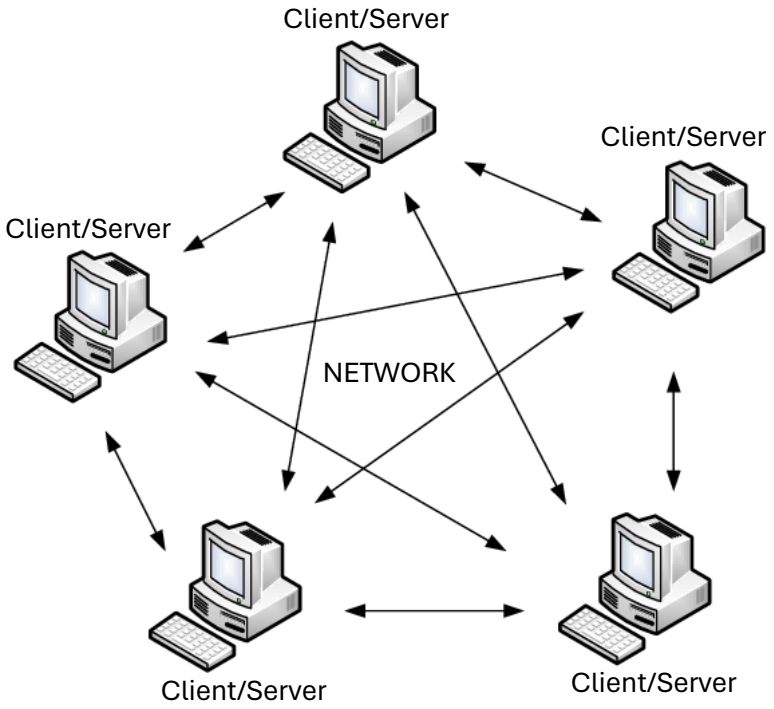


Peer-to-Peer (P2P) Model

A decentralized network architecture where each node, or "peer“, has equivalent capabilities and responsibilities. This model enables direct sharing of resources among peers without the need for centralized coordination, often used for file sharing, communications, and collaborative tasks.

Each peer can be server or client:

- Requests services or resources.
- Provides resources and services.
- Manages processes and data storage.
- Sends output back to the others.



Types of Computer Networks

Networks can be categorized based on their geographic distribution and the distance between connected devices. This classification helps in understanding the scope and the technical requirements of different network setups. From personal devices interacting over mere meters to global communications spanning thousands of kilometers, each network type serves distinct purposes and fulfills specific connectivity needs. There are various network classifications based on their scale: Personal Area Networks (PANs), Local Area Networks (LANs), Metropolitan Area Networks (MANs), Wide Area Networks (WANs), and the global scale of the Internet.

Interprocessor distance	Processors located in same	Example
1 m	Square meter	Personal area network
10 m	Room	Local area network
100 m	Building	
1 km	Campus	
10 km	City	Metropolitan area network
100 km	Country	Wide area network
1000 km	Continent	
10,000 km	Planet	The Internet

1. Computer Architectures

The Management of the Information

Summary

- Raw data
- Information and Knowledge
- The management of the Information
- The management of the Information: Binary Code
- ASCII Code
- Understanding Digital Data Sizes
- Binary Code: Properties
- Converting Binary to Decimal
- Converting Decimal to Binary
- Boolean Algebra
- The Von Neumann Model

Raw data

Data refers to **raw**, **unprocessed facts** that are **collected from different sources**. These facts can be **numbers**, **symbols**, **characters**, or even **observations**.

Data on its own does not have any meaning or context until it is processed or interpreted.

Examples of "data" include:

- a value (e.g., 35),
- a series of measurements,
- a name,
- a color,

... and so on.

Key Point:

Data is unorganized and has no direct significance by itself.

Information and Knowledge

Information is what you get when data is **processed** and **organized**, giving **context** to data.

Example of “information”:

- a value associated with degrees Celsius (e.g., 35°C) represents a temperature value.

Knowledge is the understanding and insight gained from **interpreting** information, making it useful for decision-making. Knowledge answers questions like "what," "who," "when," and "how much."

Example of “knowledge”:

- a temperature value such as 35°C becomes knowledge when it is interpreted as a warm environment.

The Management of the Information: Binary Code/1

Any information **stored** or **retrieved** by a device (such as desktop computers, laptops, or smartphones) must be represented in a language that the device can understand. This language is known as **Binary Code**.

Binary Code is defined as a **coding system** that uses the binary digits **0** and **1** to represent letters, digits, or other characters in a computer or other electronic devices, such as smartphones and tablets.

More specifically, a **BIT (Binary digit)** is the **basic unit of information** used by devices to manage all types of data. A single BIT can exist in one of two states: 0 or 1.

NOTE: With a single BIT, only two elements can be represented.

For example:

1 = "A"

0 = "B"

What about more complex information? BITS COMBINATION.

The Management of the Information: Binary Code/2

Bits and Combinations:

- One bit can have 2 states: 0 or 1.
- With **N bits** we can create $\mathbf{2^N}$ **distinct combinations**.

Examples:

- **1 bit:** $2^1 = 2$ combinations (0, 1)
- **2 bits:** $2^2 = 4$ combinations (00, 01, 10, 11)
- **3 bits:** $2^3 = 8$ combinations (000, 001, 010, 011, 100, 101, 110, 111)

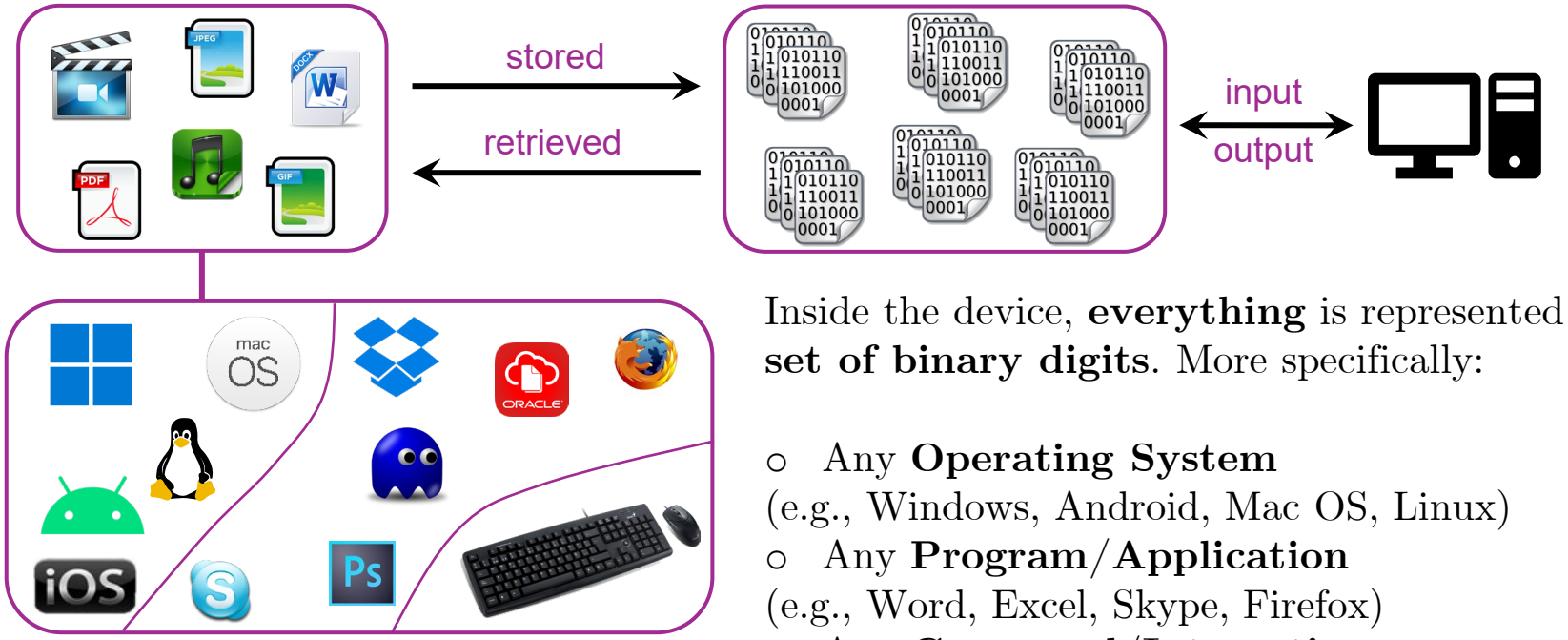
Range of Values:

With **N bits**, the representable values range from **0** to $\mathbf{2^N - 1}$.

Examples:

- **1 bit:** $2^1 - 1 = 1$, representable values in range [0,1]
- **2 bits:** $2^2 - 1 = 3$, representable values in range [0,3]
- **3 bits:** $2^3 - 1 = 7$, representable values in range [0,7]

The Management of the Information: Binary Code/3



Inside the device, **everything** is represented by a **set of binary digits**. More specifically:

- Any **Operating System**
(e.g., Windows, Android, Mac OS, Linux)
- Any **Program/Application**
(e.g., Word, Excel, Skype, Firefox)
- Any **Command/Interaction**
(e.g., Keyboard, Mouse, Touchpad)

ASCII Code

ASCII: The American Standard Code for Information Interchange typically uses **7 bits** to represent each character, which allows for **128 unique character** codes.

Extended ASCII: ASCII is sometimes extended to **8 bits**, which enables an **additional 128 characters**, bringing the total to 256. This extension isn't standardized in the same way as the original 7-bit ASCII, and there are various extended ASCII sets, like ISO 8859-1 or Windows-1252, which include characters for **specific languages** or **graphical symbols**.

Types of Characters:

- **Alphanumeric Characters:** This includes all uppercase and lowercase English letters and numbers (0-9).
- **Symbols:** ASCII includes a set of common punctuation symbols and other miscellaneous symbols like @, #, $, etc.
- **Control Characters:** These are non-printable characters that control the flow of text or its processing in some way. Examples include TAB (horizontal tab), LF (line feed), CR (carriage return), and BEL (bell/alert).

ASCII Code - Examples

Standard ASCII Table (7-bit)

Decimal	Hex	Binary	Character	Description
32	20	00100000	(space)	Space
48	30	00110000	0	Digit Zero
65	41	01000001	A	Uppercase A
97	61	01100001	a	Lowercase a
10	0A	00001010	LF	Line Feed
13	0D	00001101	CR	Carriage Return

Extended ASCII Table (8-bit)

Decimal	Hex	Binary	Character	Description
128	80	10000000	Ç	Latin Capital Letter C with Cedilla
165	A5	10100101	¥	Yen Sign
178	B2	10110010	²	Superscript Two
225	E1	11100001	á	Lowercase a with acute
245	F5	11110101	õ	Lowercase o with tilde

Understanding Digital Data Sizes/1

This table provides a comparison of data sizes using different prefixes, their decimal sizes, binary approximations, and practical examples.

- A group of **8 bits** is named **Byte**;

Abbr.	Prefix-Byte	Decimal Size	Size in Thousands	Binary Approximation	An Example
K	Kilo-	10^3	$1,000^1$	$1,024 = 2^{10}$	A text file (e.g., 1 KB)
M	Mega-	10^6	$1,000^2$	$1,024^2 = 2^{20}$	A song (e.g., MP3 file, 4 MB)
G	Giga-	10^9	$1,000^3$	$1,024^3 = 2^{30}$	A HD film (e.g., MP4 file, 1.5 GB)
T	Tera-	10^{12}	$1,000^4$	$1,024^4 = 2^{40}$	A large backup drive (e.g., 1 TB of data)
P	Peta-	10^{15}	$1,000^5$	$1,024^5 = 2^{50}$	Storage of large-scale research data (e.g., climate data, 1 PB)
E	Exa-	10^{18}	$1,000^6$	$1,024^6 = 2^{60}$	Entire data usage of a major tech company (e.g., Google, 1 EB)

Understanding Digital Data Sizes/2

Inside each device, **information** is represented by a fixed set of bytes:

- 16 bits (**2 bytes**)
- 32 bits (**4 bytes**)
- 64 bits (**8 bytes**)

The number of bytes indicates the "power" of a device. The greater the number of bytes:

- The greater the device's ability **to perform complex computations**
- The greater the device's capacity **to manage large amounts of information**
- The greater the device's capability **to process complex instructions**

Binary Code: Properties

Like every numerical system, binary code has its own **intrinsic properties**. One key property, shared with other **positional systems**, is the ability to convert numbers from one numerical base to another.

Furthermore, the **binary system** supports the **four basic operations**:

- Addition;
- Subtraction;
- Multiplication;
- Division.

NOTE:

A number represented using **two digits** is called a **binary number** (**base two**).

A number represented using **ten digits** is called a **decimal number** (**base ten**).

Converting Decimal to Binary

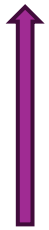
To convert a decimal number (base 10) to a binary number (base 2), follow these steps:

1. **Divide** the number by 2.
2. **Record** the remainder (0 or 1). This will be the least significant bit (LSB).
3. **Continue dividing** the quotient obtained in the previous step by 2, **recording** the remainders.
4. **Repeat** until the quotient is 0.
5. **Write the remainders in reverse order**: from the last obtained to the first.

   This represents the number in binary.

Converting 13_{10} to Binary:

$13 \div 2 = 6$, remainder $= 1$
$6 \div 2 = 3$, remainder $= 0$
$3 \div 2 = 1$, remainder $= 1$
$1 \div 2 = 0$, remainder $= 1$



Result in binary: 1101_2

Converting Binary to Decimal

To convert a binary number (base 2) to a decimal number (base 10), follow these steps:

1. **Write down** the binary number.
2. **List the powers of 2** from right to left, starting with 2^0 under the rightmost bit.
3. **Multiply each bit** by the corresponding power of 2.
4. **Sum** the results to get the decimal number.

Converting 1101_2 to Decimal:

$$\begin{aligned} 1101 &= 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 \\ &= 8 + 4 + 0 + 1 = 13 \end{aligned}$$

Result in decimal: 13_{10}

Binary Addition/1

1. **Take** the two binary numbers you want to add.

   NOTE: Make sure both numbers are of the same length.

   If not, add leading zeros to the shorter number until they are of the same length.

2. **Start adding** the bits from the rightmost side (least significant bit) of the two binary numbers, following these rules:

Case	A + B	Sum	Carry
1	0 + 0	0	0
2	0 + 1	1	0
3	1 + 0	1	0
4	1 + 1	0	1

3. **Proceed by adding** the successive bits and considering any carries from the previous columns. If after adding the last bit there remains a carry, write it in the leftmost column of the result.

Binary Addition/2

Example 1 (1011 + 1101):

```
  1011
+ 1101
----------
 11000 (Result: carry applied)
```

Step-by-step addition with carry:
1 + 1 = 0 (carry 1)
1 + 0 + 1 (carry) = 0 (carry 1)
0 + 1 + 1 (carry) = 0 (carry 1)
1 + 1 + 1 (carry) = 1 (carry 1 to a new column)

Example 2 (1011 + 101):

```
  1011
+ 0101
----------
 10000 (Result: carry applied)
```

Step-by-step addition with carry:
1 + 1 = 0 (carry 1)
1 + 0 + 1 (carry) = 0 (carry 1)
0 + 1 + 1 (carry) = 0 (carry 1)
1 + 0 + 1 (carry) = 0 (carry 1 to a new column)

Binary Subtraction/1

1. **Take** the two binary numbers you want to subtract.

   NOTE: Make sure both numbers are of the same length.

   If not, add leading zeros to the shorter number until they are of the same length.

2. **Start subtracting** the bits from the rightmost side (least significant bit) of the two binary numbers, following these rules:

Case	A - B	Difference	Borrow
1	0 - 0	0	0
2	1 - 0	1	0
3	1 - 1	0	0
4	0 - 1	1	1

3. **Proceed by subtracting** the successive bits and considering any borrows from the previous columns. If after subtracting the last bit there remains a borrow, write it in the leftmost column of the result.

Binary Subtraction/2

Example 1 (1110 – 1001):

```
  1110
- 1001
 --------
  0101 (Result: borrow applied)
```

Step-by-step subtraction with borrow:
0 - 1 = 1 (borrow 1)
1 - 0 - 1 (borrow) = 0
1 - 0 = 1
1 - 1 = 0

Example 2 (1011 - 101):

```
  1011
-  0101
-----------
  0110 (Result: borrow applied)
```

Step-by-step subtraction with borrow:
1 – 1 = 0
1 – 0 = 1
0 - 1 = 1 (borrow 1)
1 - 0 - 1 (borrow) = 0

Boolean Algebra/1

Boolean Algebra is a branch of algebra that deals with **true** or **false** values, typically denoted as **1** and **0**, respectively. It is fundamental in the field of **computer science** and **digital electronics** because it is used to design and analyze the **behavior of digital circuits** and **logic gates**.

Key Concepts:

- **Binary Variables:** Represented as 1 (true) and 0 (false).
- **Logical Operations:**

  AND ($\cdot$): Yields true if both operands are true ($1 \cdot 1 = 1$).

  OR ($+$): Yields true if at least one operand is true ($1 + 0 = 1$).

  NOT ($\neg$): Yields the inverse of the operand ($\neg 1 = 0$).

Boolean Algebra/2

TRUTH TABLES

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

A	NOT A
0	1
1	0

Boolean Algebra/3

- **Commutative:**

A OR B = B OR A

A AND B = B AND A

- **Distributive:**

A OR (B AND C) = (A OR B) AND (A OR C)

A AND (B OR C) = (A AND B) OR (A AND C)

- **De Morgan's Law:**

NOT (A AND B) = (NOT A) OR (NOT B)

NOT (A OR B) = (NOT A) AND (NOT B)

- **Precedence Rules:**

- NOT has the highest precedence;

- followed by AND;

- lastly, OR.

To alter the precedence, use parentheses.

Boolean Algebra/4

Example:

A AND NOT (B OR C)

A	B	C	A AND NOT (B OR C)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

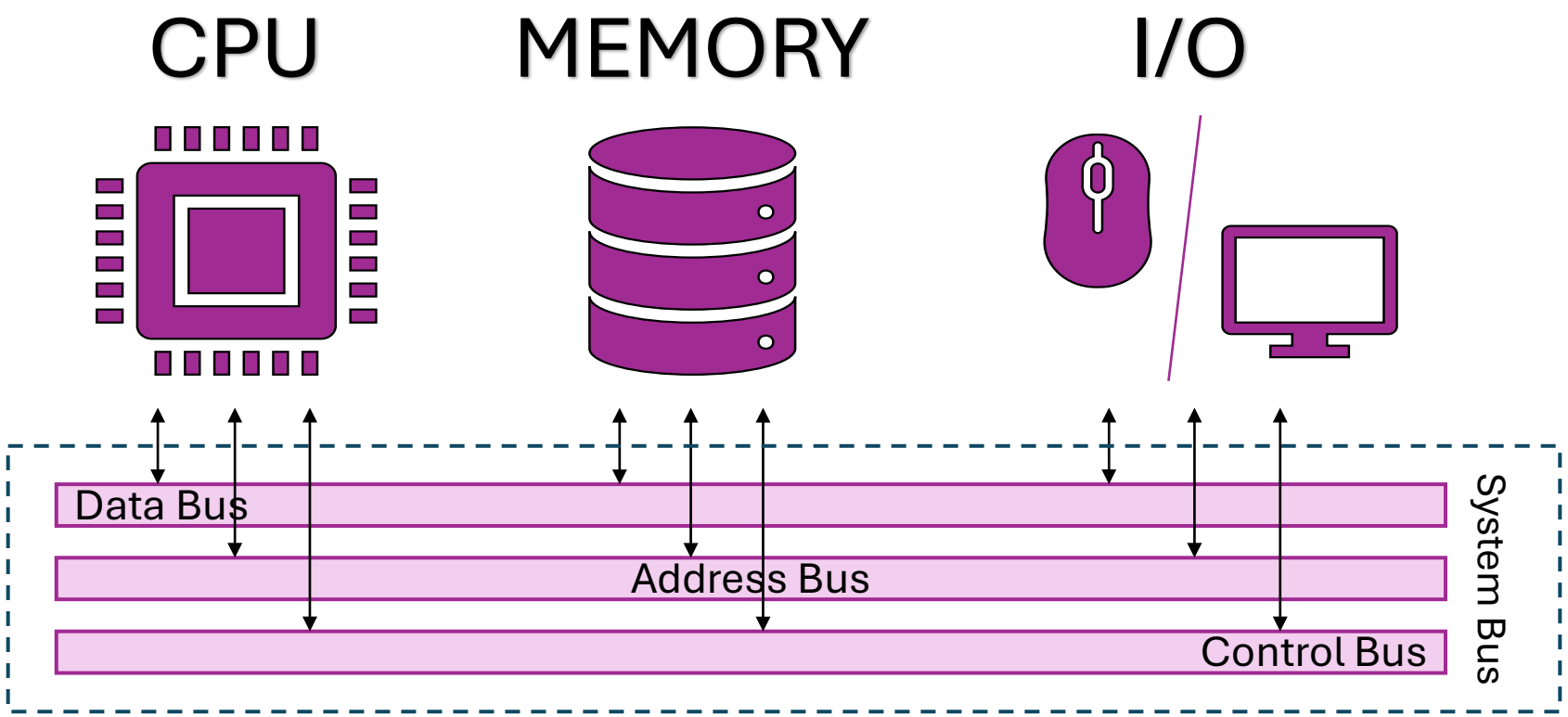
The Von Neumann Model/1

The **Von Neumann Architecture** is a **foundational concept for modern computing systems**, characterized by its structure where data and program code are stored in the same memory space.

The typical **Von Neumann Model** consists of 4 main components:

- **Memory:** Stores both data and instructions.
- **Central Processing Unit:**
    - **Arithmetic Logic Unit (ALU):** Executes all arithmetic and logical operations.
    - **Control Unit (CU):** Decodes program instructions and controls the other components based on these instructions.
- **Input and Output (I/O):** Handle data exchange between the computer system and the external environment.
- **Bus:** Provides a communication system that transfers data between components.

The Von Neumann Model/2



1. Computer Architectures

The Management of the Information

(Additional Resources)

Two’s Complement

Two's Complement is a method used in computer science and electronics to represent **signed integers** in binary systems. It is particularly useful because it **simplifies arithmetic operations**, such as addition and subtraction, allowing both **positive** and **negative numbers** to be handled by the same logical circuits.

Basics Concepts:

- The most significant bit (MSB) is used as the sign bit:
  - 0: indicates a positive number.
  - 1: indicates a negative number.
- **Negative numbers** are represented by **calculating the two's complement** of their corresponding **positive value**. **Positive numbers** are represented by **calculating the two's complement** of their corresponding **negative value**.

Examples:

+3 in 4-bit binary is $\boxed{0}\,011 \longrightarrow$ positive number

MSB

-3 in 4-bit binary is $\boxed{1}\,101 \longrightarrow$ negative number

MSB

ICTAcademy

INFORMATION AND COMMUNICATIONS TECHNOLOGY

How to Calculate Two's Complement 1/2

To find the two's complement of a binary number:

1. **Invert all the bits** of the binary number (one's complement).
2. **Add 1** to the result of the previous step.

Positive -> Negative

Let's assume we are using 4-bit numbers.

Representation of +3:

- The binary representation is: 0011 (4 bits).

Representation of -3:

- **Invert the bits** of +3 (0011): the result is 1100 (one's complement).
- **Add 1 (i.e., 0001)** to 1100.

```
   1100
 + 0001
 ------------
   1101 ⟶ -3 in 4-bit binary
```

How to Calculate Two's Complement 2/2

To find the two's complement of a binary number:

1. **Invert all the bits** of the binary number (one's complement).
2. **Add 1** to the result of the previous step.

Negative -> Positive

Let's assume we are using 4-bit numbers.

Representation of -3:

- The binary representation is: 1101 (4 bits).

Representation of +3:

- **Invert the bits** of -3 (1101): the result is 0010 (one's complement).
- **Add 1 (i.e., 0001)** to 0010.

```
  0010
+ 0001
------
  0011 → +3 in 4-bit binary
```

Two's Complement Properties

- **Single Zero Representation:** In two's complement, positive zero (`0000`) and negative zero (`0000`) share the same binary representation, avoiding ambiguity.

- **Ease of Addition and Subtraction:** To subtract a number, you simply add its two's complement.

  Examples:

  - "5 – 3" can be performed as "5 + (– 3)" in binary.
  - "-5 – 3" can be performed as "-5 + (-3)" or as "(5 + 3) applying the two's complement to the result" in binary.

  By converting subtraction into addition using two's complement, binary arithmetic becomes simpler and more efficient, particularly in digital systems like computers and calculators.

- **Range of Values:** For an N-bit system, two's complement allows representation of numbers from -2^{N-1} to $2^{N-1} - 1$. For example, with 4 bits, from -8 to +7.

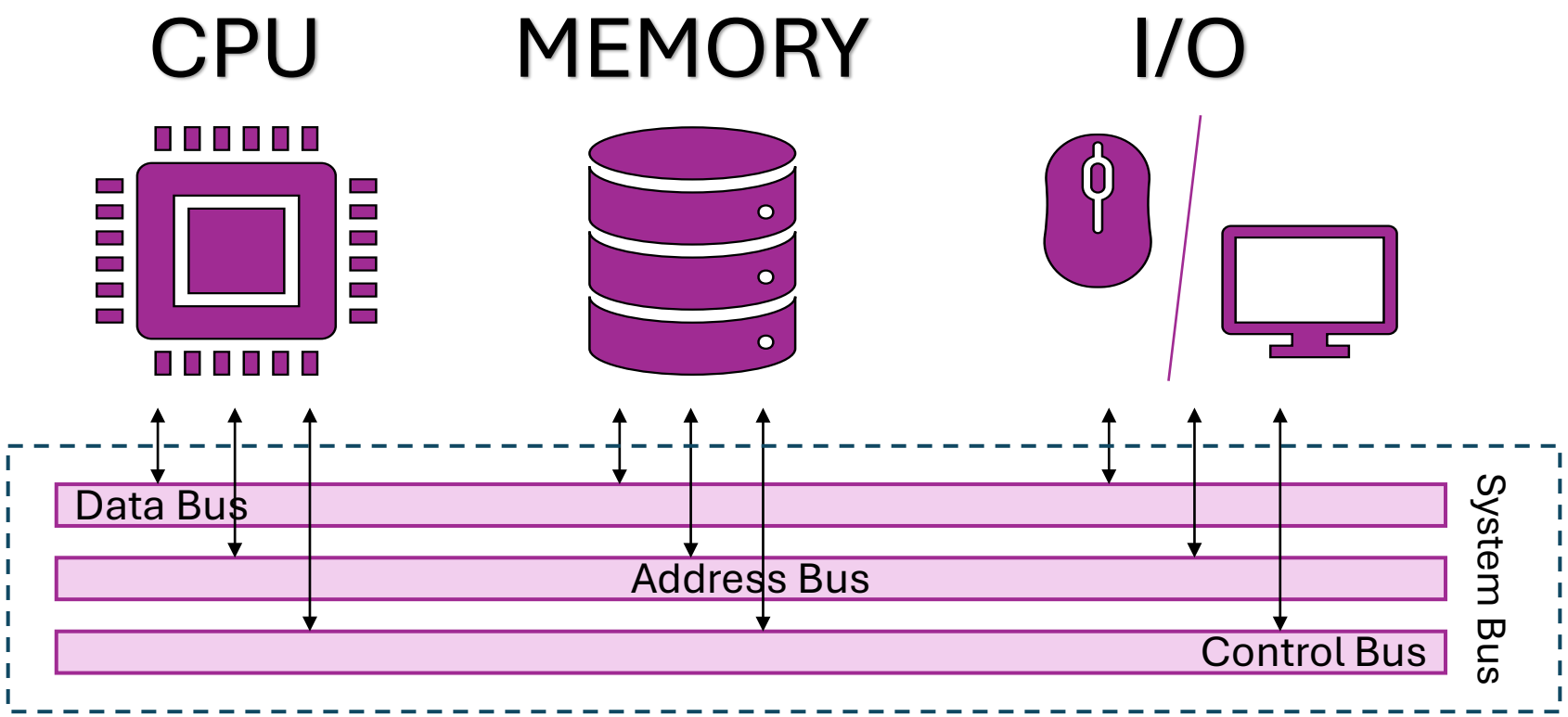
2. Computer Architectures

CPU, Memory, and I/O

Summary

- The Von Neumann Model
- System Bus
- CPU
- Instruction Cycle
- Memory
- MDR and MAR Registers
- FETCH and STORE Operations
- INPUT and OUTPUT Devices
- INPUT/OUTPUT Devices

The Von Neumann Model



System Bus

The **system bus** typically connects the CPU, RAM, I/O peripherals, and other system components. It allows for the **transfer of data**, **instructions**, and **control signals** between these components, enabling them to **collaborate** and **interact**.

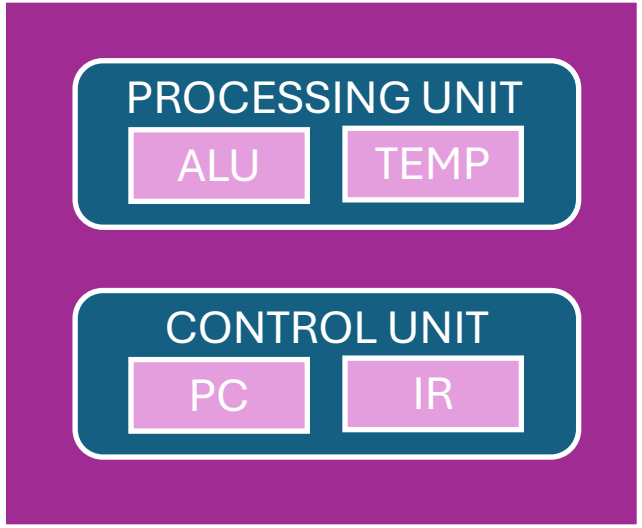
- **Data Bus:** Transmits binary data from one component to another. The data bus lines allow the transfer of information such as program instructions, data to be processed, or calculation results.

- **Address Bus:** Indicates a specific location in memory or I/O peripherals. The address bus allows system components to specify the source or destination of the data to be transferred.

- **Control Bus:** Carries control signals that coordinate and manage the operations of various system components. Control signals can include synchronization signals, interrupt signals, operation enable or disable signals, and status signals.

CPU

The **CPU (Central Processing Unit)** is the primary component of a computer that performs most of the processing inside the system. It **executes instructions** from programs, **performs calculations**, and **manages data flow** to and from other components like memory and peripherals. The CPU is often referred to as the **"brain"** of the computer, as it handles all critical tasks to ensure the system operates efficiently.



CPU – Processing Unit

The **processing unit** is a central part of the CPU responsible for **executing instructions** and managing the **data processing** tasks. It includes several key components that work together to perform **arithmetic** and **logical** operations, **control instruction flow**, and **store intermediate results**.

- **ALU (Arithmetic Logic Unit)**

The ALU is a critical component of the CPU that performs **arithmetic** operations (such as addition, subtraction, multiplication, and division) and **logical** operations (such as AND, OR, NOT, and XOR). It is essential for executing calculations and making decisions based on logical comparisons.

- **TEMP (Temporary Registers)**

Temporary registers (TEMP) are **small**, **fast storage locations** within the CPU used to hold **intermediate data** and **results** during instruction execution. They provide quick access to frequently used values and help optimize the processing speed by reducing the need to access slower main memory.

CPU – Control Unit

The **control unit** is a crucial part of the CPU that **directs the operation of the processor**. It **interprets instructions** from the program, **generates control signals**, and **coordinates the activities** of the CPU's other components, ensuring that instructions are executed in the correct sequence and timing.

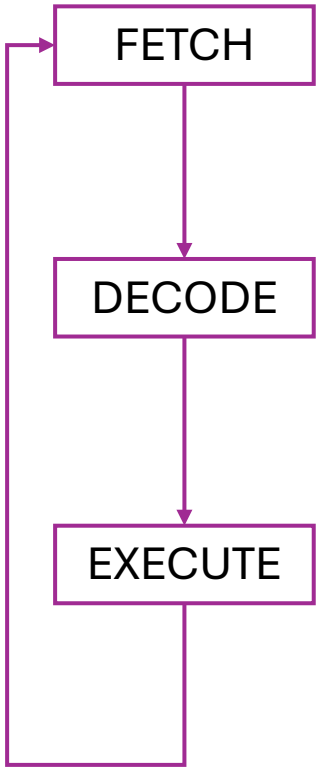
- **PC (Program Counter)**

The Program Counter (PC) is a register within the control unit that holds the **address of the next instruction to be executed**. It increments after each instruction is fetched, ensuring a sequential flow of program execution, unless a jump or branch instruction alters the flow.

- **IR (Instruction Register)**

The Instruction Register (IR) is a component of the control unit that **temporarily** holds the **current instruction being executed**. It stores the fetched instruction from memory, allowing the control unit to decode and process it, directing the necessary actions to other parts of the CPU.

Instruction Cycle



- **Fetch:**

The CPU retrieves the next instruction from memory, as indicated by the Program Counter (PC).

The instruction is loaded into the Instruction Register (IR).

- **Decode:**

The control unit interprets the fetched instruction in the IR.

It determines the operation to be performed and identifies the necessary operands.

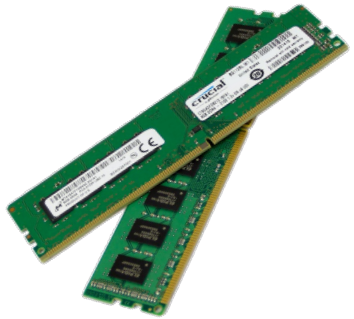
- **Execute:**

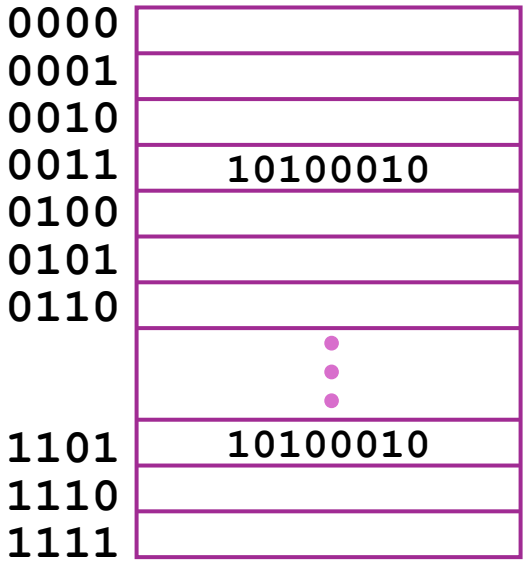
The CPU performs the operation specified by the instruction.

This may involve arithmetic or logical operations carried out by the ALU, data transfer, or control operations.

Memory/1

In the Von Neumann architecture, memory typically refers to **RAM (Random Access Memory)**, which is a critical component that **stores** both **data** and **program instructions temporarily** while the computer is running.





When a device is operating, both the **program** (e.g., Word application) and the **associated data** (e.g., the DOC document you are writing) are **stored in memory (RAM)**.

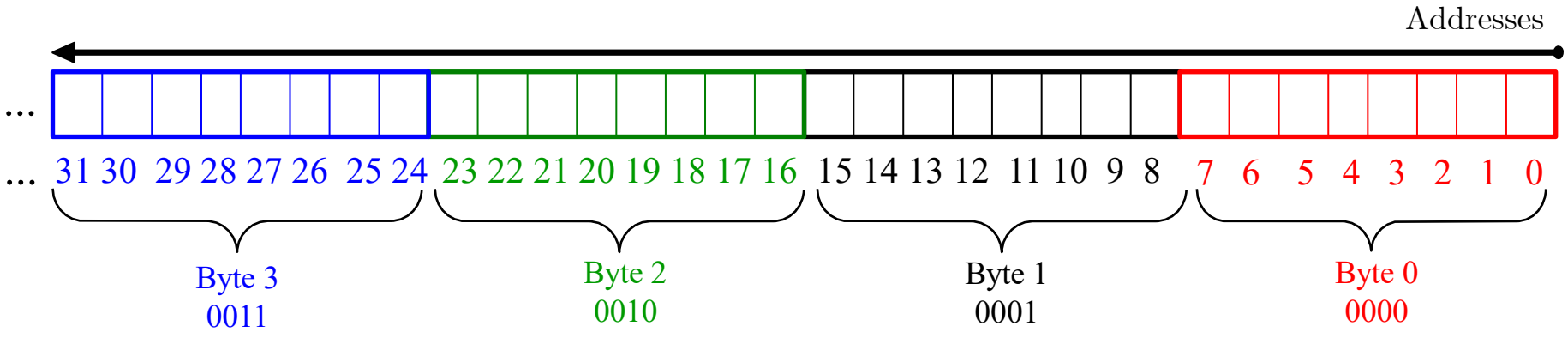
The **basic operations** that can be performed with memory include:

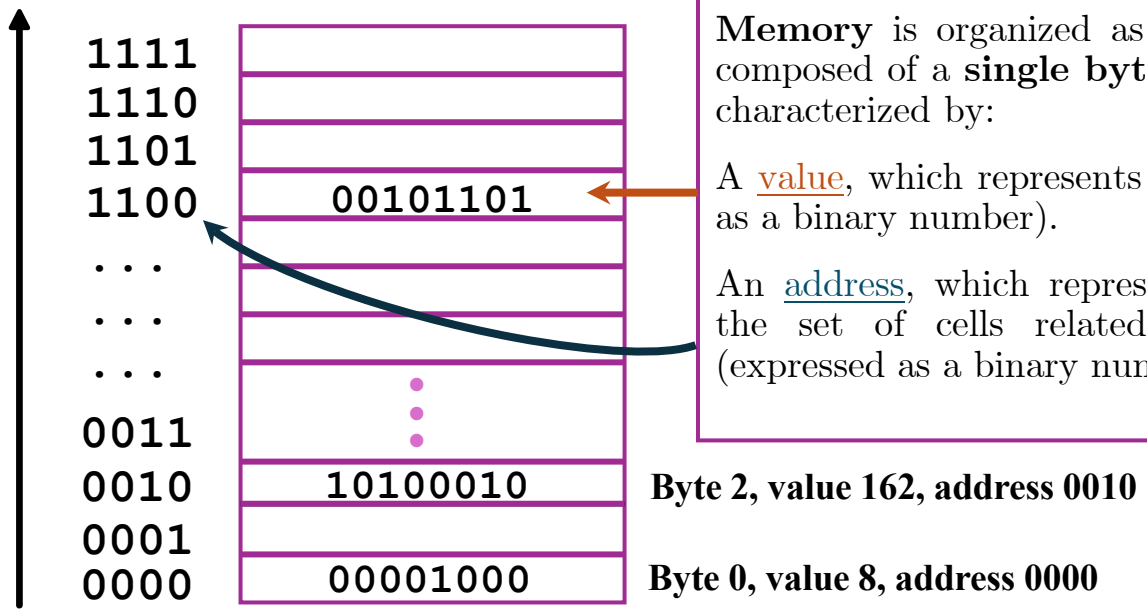
STORE: To save information.
FETCH: To retrieve information.

These operations are facilitated by the addressing mechanism known as **memory addresses**.

ICTAcademy
INFORMATION AND COMMUNICATIONS TECHNOLOGY

Memory/2





Memory is organized as a **sequence of cells**, each composed of a **single byte** (**8 bits**). A memory cell is characterized by:

A value, which represents the cell's content (expressed as a binary number).

An address, which represents the starting address of the set of cells related to a specific application (expressed as a binary number).

Byte 2, value 162, address 0010

Byte 0, value 8, address 0000

MDR and MAR Registers

- **MDR (Memory Data Register)**

The Memory Data Register (MDR), also known as the Memory Buffer Register (MBR), is a register within the CPU that temporarily **holds data being transferred to or from memory**.

When data is read from memory, it is first loaded into the MDR before being processed by the CPU. Similarly, when data is written to memory, it is placed in the MDR before being stored in the specified memory location. The MDR acts as a buffer, facilitating smooth data transfer between the CPU and memory.

- **MAR (Memory Address Register)**

The Memory Address Register (MAR) is a register within the CPU that **holds the address of the memory location** to be accessed next.

During the fetch stage of the instruction cycle, the MAR contains the **address of the instruction to be fetched**.

During data operations, the MAR holds the **address of the data to be read from or written to memory**. The MAR ensures that the correct memory location is accessed, enabling precise data retrieval and storage.

FETCH and STORE Operations

The operation of the Memory module involves the following two steps:

FETCH operation / To read from a location (A):

- Write the address (A) into the MAR.
- Send a "read" signal to the memory.
- Read the data from the MDR.

STORE operation / To write a value (X) to a location (A):

- Write the data (X) to the MDR.
- Write the address (A) into the MAR.
- Send a "write" signal to the memory.

INPUT and OUTPUT Devices

Input and Output (I/O) operations are crucial for allowing a computer system to **interact** with the **external environment** and **peripheral devices**.

Input:

- **Purpose:** To gather data from external sources and provide it to the computer for processing.
- **Devices:** Common input devices include keyboards, mice, scanners, microphones, and cameras.
- **Process:** Input devices convert user actions or external data into digital signals that the computer can process.

Output:

- **Purpose:** To present processed data from the computer to the external environment.
- **Devices:** Common output devices include monitors, printers, speakers, and projectors.
- **Process:** Output devices convert digital signals from the computer into a human-readable or perceivable form, such as text, images, sound, or physical output.

INPUT/OUTPUT Devices

Some devices function as **both input and output** devices, allowing for **two-way interaction** between the user and the computer system. These devices are versatile and can **both send data to** and **receive data from** the computer.

Examples of I/O Devices:

- **Touchscreen Monitors:**

- Input: The user can interact with the screen by touching it, which sends input signals to the computer.

- Output: The screen displays visual information from the computer.

- **External Hard Drives:**

- Input: Data can be written to the drive from the computer.

- Output: Data can be read from the drive to the computer.

- **Modems:**

- Input: Receive data from the internet.

- Output: Send data to the internet.

2. Computer Networks

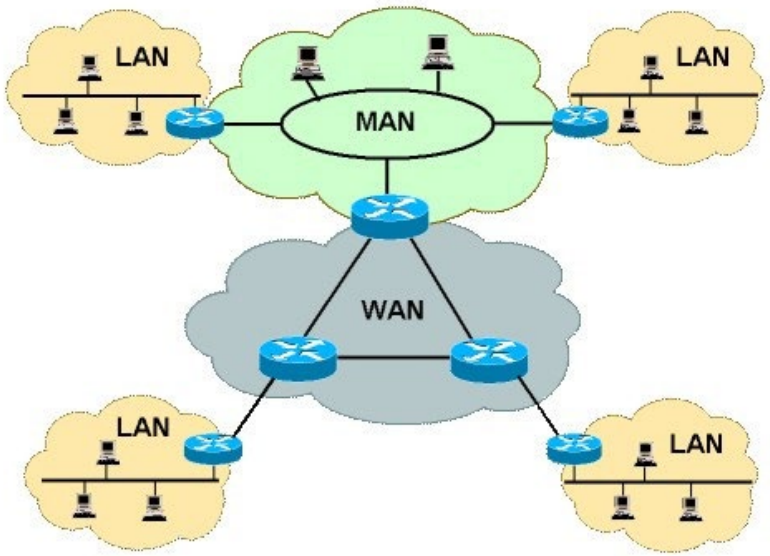
Protocols

Summary

- Internetwork
- Protocols
- Central Role of Software
- Network Communication
- Connection-oriented Services
- Connectionless Services

Internetwork

Internetwork or **Internet**: a collection of different networks connected via a gateway. Each network uses **different hardware** and, more importantly, **different protocols** to manage internal communication.



Protocols

Communication between various entities within networks is managed by **protocols**.

A **network protocol** is a formal, predefined **set of rules** that govern the interactions between two or more connected electronic devices to facilitate communication. These rules primarily address the software components of the devices.

The earliest computer networks were primarily focused on hardware, considering software merely as an adjunct. This approach is outdated. In modern contexts, **network software** is intricately structured and plays a **central role** in network functionality.

Central Role of Software/1

Feature Complexity:

Modern networks must support a wide range of services and applications, from simple data transmission to complex security operations, traffic management, performance optimization, and cloud integration. These functions require sophisticated software capable of dynamically managing various needs and configurations.

Adaptability and Scalability:

Today's networks need to be highly adaptable and scalable to swiftly respond to changing business or consumer demands. Network software enables this flexibility, allowing updates and modifications to be implemented much more quickly and at lower costs compared to hardware changes.

Security:

The security of modern networks heavily relies on advanced software to implement firewalls, intrusion detection systems, encryption, and other essential security measures to protect data and keep the network secure against increasingly sophisticated threats.

Central Role of Software/2

Network software is crucial not only for managing functionality but also for **ensuring compatibility** and **communication** among **diverse hardware** components.

By acting as an **intermediary**, it enables **different devices**, potentially from various manufacturers, to **work together** seamlessly.

This **interoperability** is key to building flexible and efficient network systems that can adapt to **new technologies** and **requirements**.

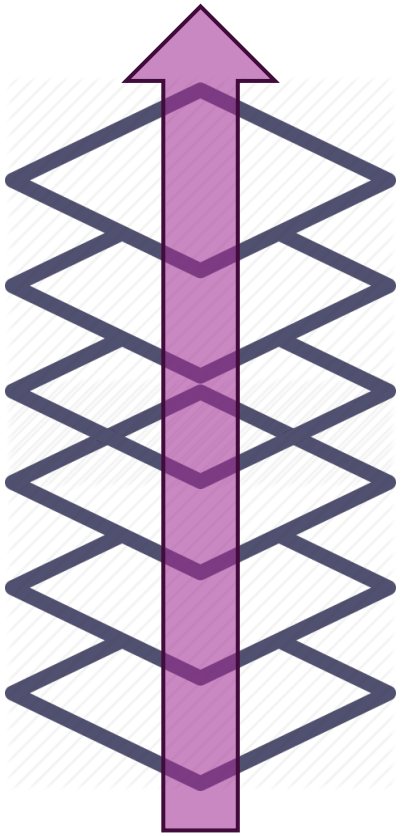
Stack of Layers

To simplify design, networks are structured into **layers** that are built upon one another.

Differences between networks can include:

- The number of layers
- The names of the layers
- The content of each layer
- The functionality of each layer

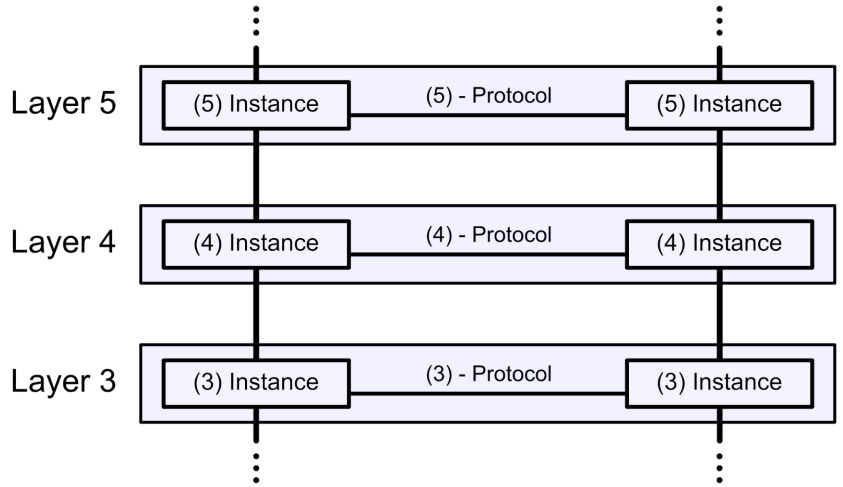
Each layer serves to provide **specific services** to the layers above it, while **abstracting the details** of how those services are implemented.



Layer Protocol

When the same network layer, referred to as **layer n**, communicates across different machines, the rules and conventions guiding this communication are known collectively as the **layer n protocol**.

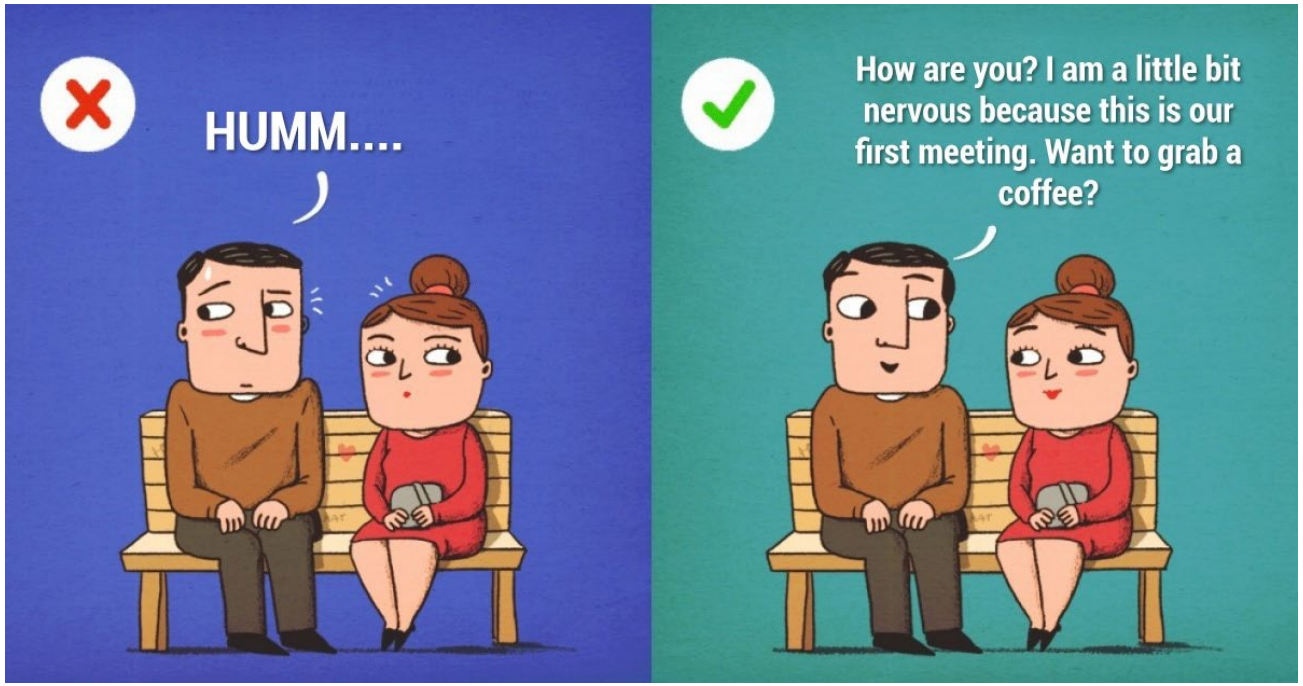
A **protocol** is essentially an agreement that dictates the **procedures for communication** between the parties involved.



Communication between the n-th layer of one machine and the n-th layer of another machine is enabled by the services offered by the underlying layer (n-1).

Protocols - Analogy

- Disregarding the protocol could complicate communication, potentially to the point of impossibility.



Network Communication - Analogy/1

Start and End Points:

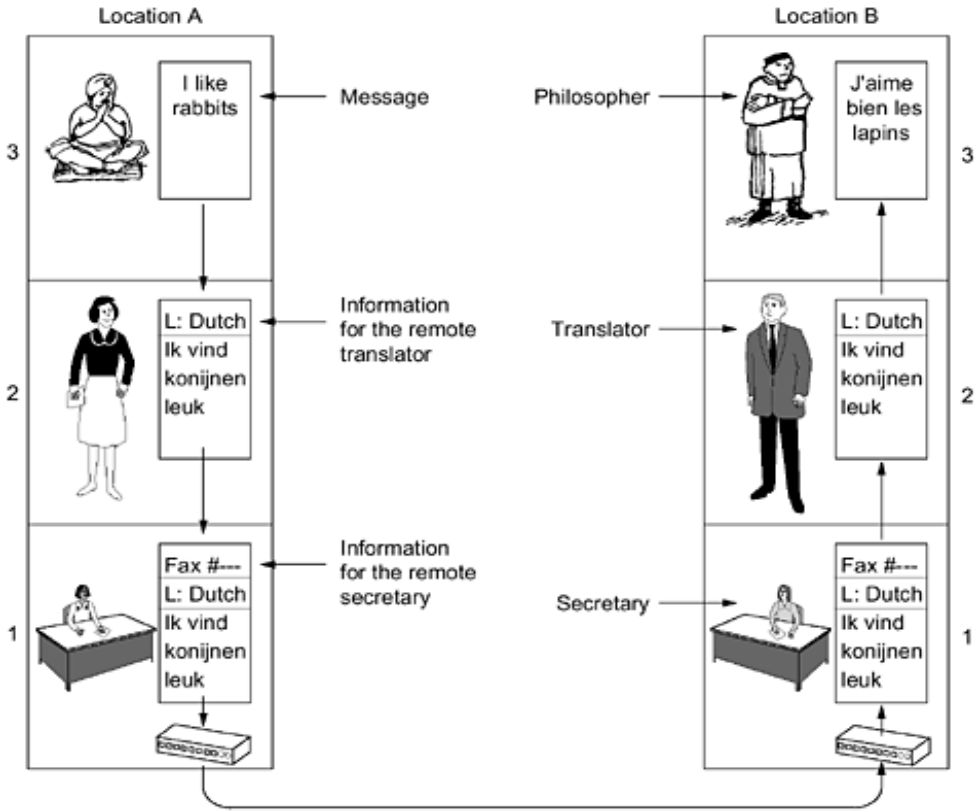
The process begins at Location A where a person (the sender) expresses a message, "I like rabbits." It ends at Location B with a philosopher (the receiver) receiving the message translated into French, "J'aime bien les lapins." In networking, these points represent the source and destination nodes.

Transmission and Translation Layers:

The message undergoes several steps before reaching the philosopher. Initially, it is translated into Dutch and passed through various intermediaries (secretary and translator) who handle and forward the information, akin to data passing through multiple network devices like routers and switches.

Data Packet Analogy:

The original message "I like rabbits" can be seen as data packets. As it moves from the source, it is processed and repackaged at various stages—translation into another language can represent encoding or encryption in network terms.



ICTAcademy
INFORMATION AND COMMUNICATIONS TECHNOLOGY

Network Communication - Analogy/2

Routing:

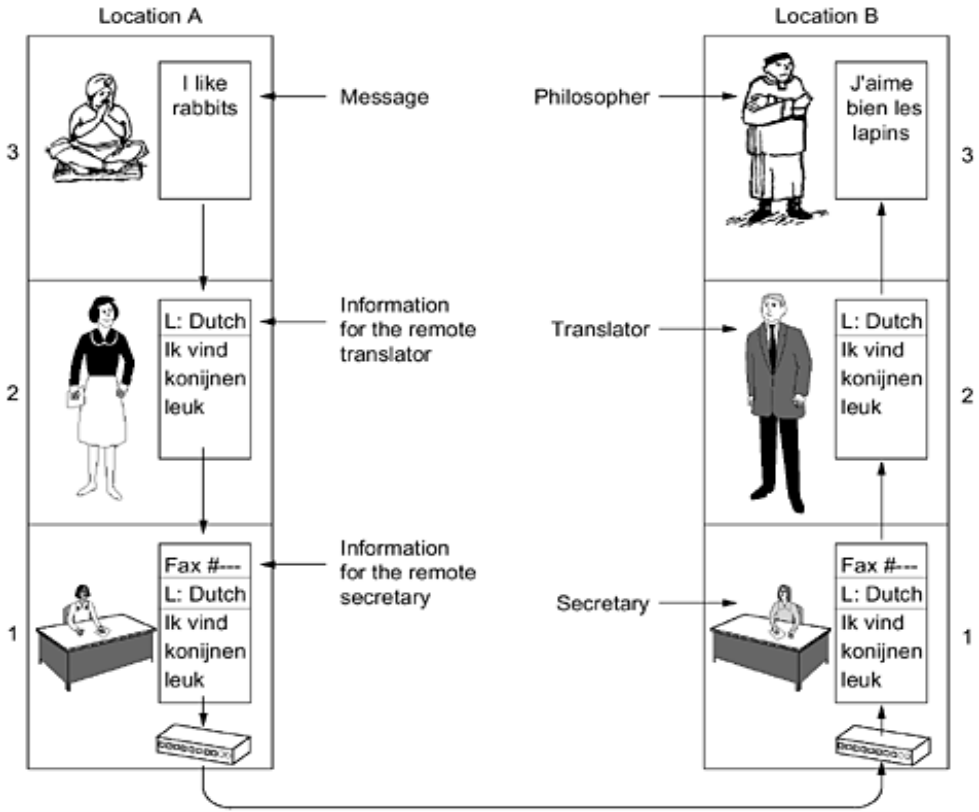
The use of faxes and different personnel for processing the information is similar to how data packets are routed through different network paths. Each node (person or device in the diagram) decides how and where to send the data next based on network protocols.

Protocol Layers:

Each step in the diagram where the message is handled and processed represents different layers in network protocols (like the OSI model). For example, the secretary could be seen as operating in the presentation layer, translating data formats, while the translator operates in the application layer, converting the message contents into different languages.

End-to-End Delivery:

Just as the message starts as a simple expression and ends up being understood by someone who speaks a different language, in networking, data starts from a source and is delivered to a destination across various network segments and possibly transformed or secured in the process, ensuring it is usable upon receipt.



Network Communication

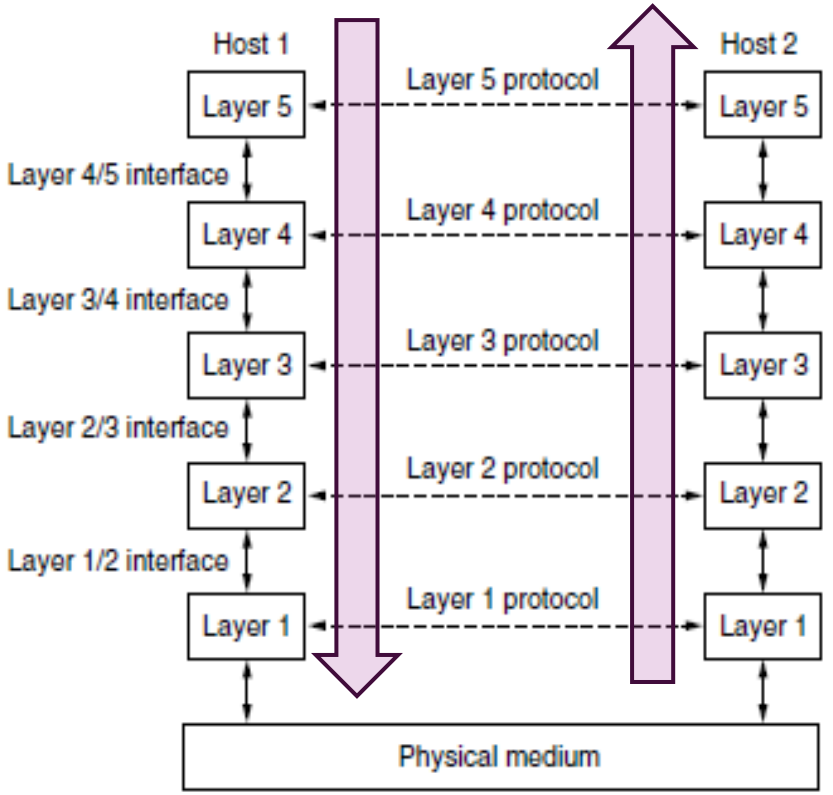
Data is not directly transferred from layer *n* on one machine to layer *n* on another machine.

- **Source (top-down):** each layer passes data and control information down to the immediately lower layer, continuing until it reaches the lowest layer.

- **Destination (bottom-up):** each layer passes data and control information up to the immediately upper layer, continuing until it reaches the highest layer.

Interface: Specifies the operations and services that the lower layer provides to the upper one.

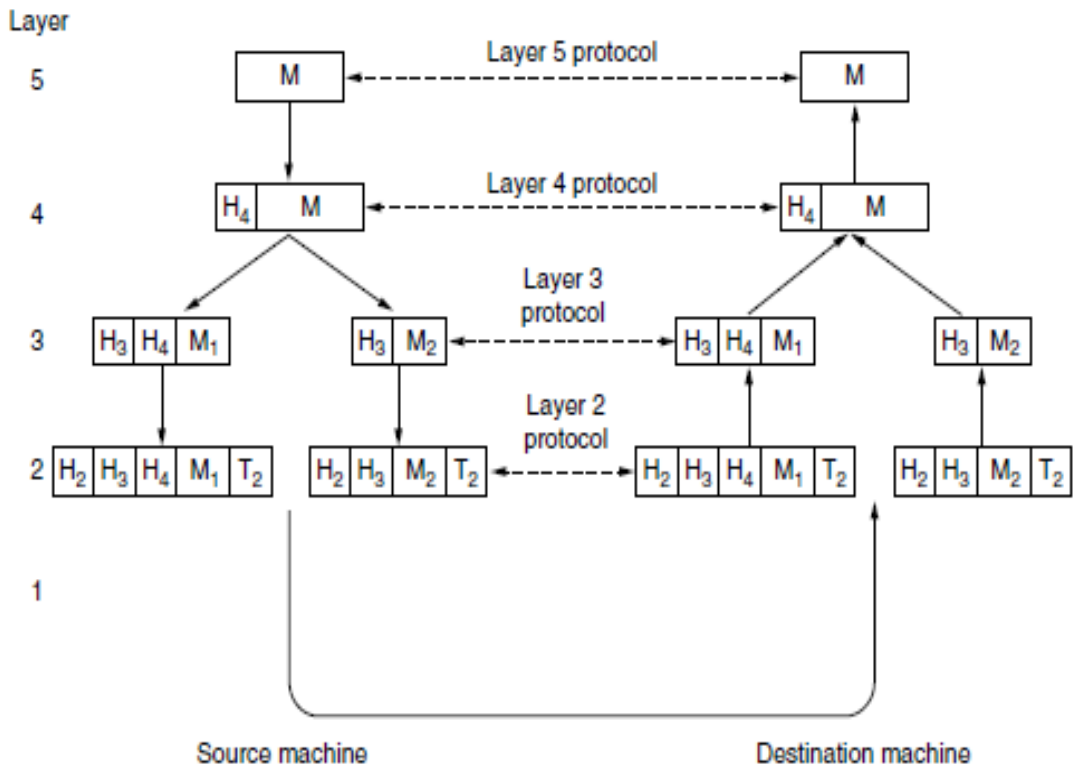
Below level 1, the **physical medium** facilitates data transfer from host 1 to host 2.



Network Communication – Technical Example/1

The application program needs to send the message M to its peer entity (i.e., the same application program on another machine).

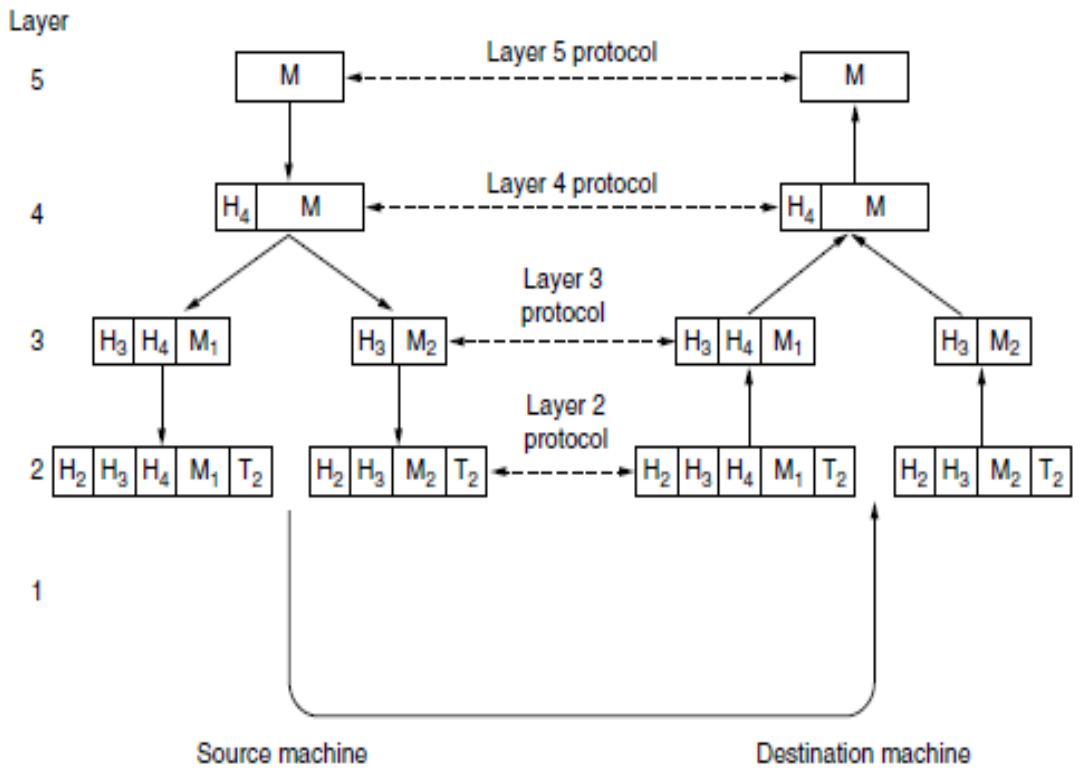
1. **Layer 5** delivers message M to **layer 4.**
2. **Layer 4** adds a header H_4 to the message. The headers contain control information such as sequence number, data offset, acknowledgment number, size, etc. The **layer 4** delivers message M to **layer 3**.
3. **Layer 3** divides the data into smaller units (packets) and adds a header H_3 to each packet. The headers contain source and destination IP addresses, total length, a unique identifier for the packet, header length, TTL etc. The **layer 3** delivers packets to **layer 2**.



ICTAcademy
INFORMATION AND COMMUNICATIONS TECHNOLOGY

Network Communication – Technical Example/2

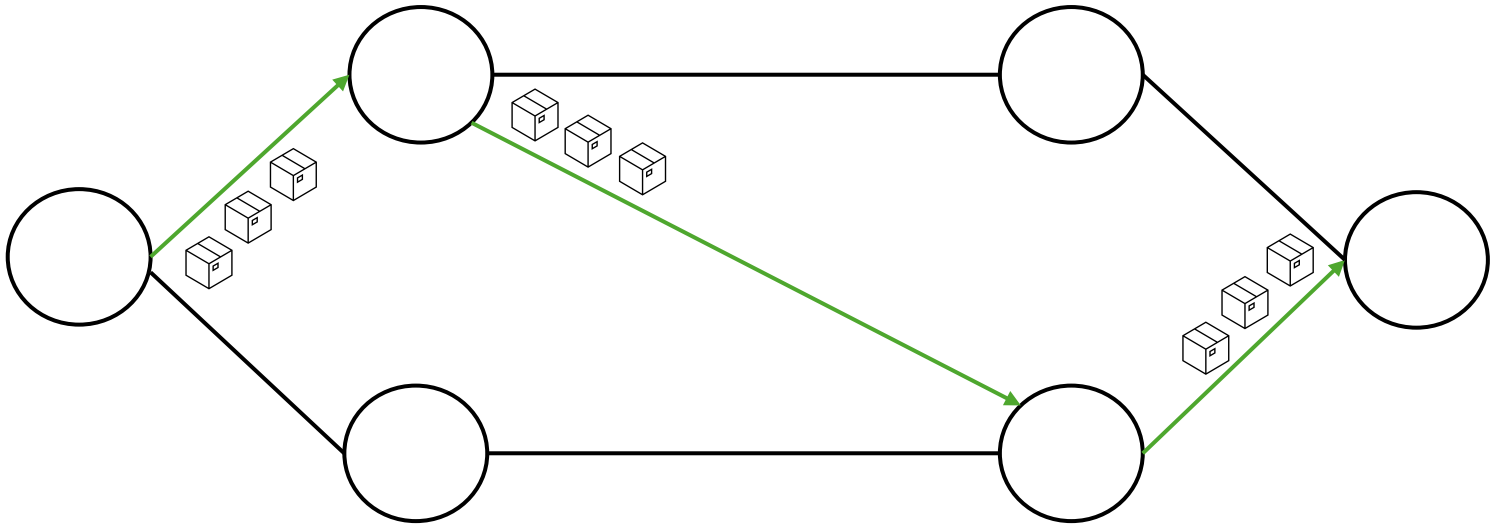
4. **Layer 2** adds a header H_2 and a trailer T_2 to each received packet referred as to frame. The headers contain source and destination MAC addresses, payload length, etc. The trailer is used for transmission error-checking. **Layer 2** delivers packets to **layer 1**.
5. **Layer 1**, the **physical layer**, transfers data to the destination machine.
6. **At the destination machine,** the message moves upward through each layer, with headers and trailers being removed at each step. The message is **fully reconstructed** and becomes readable once it reaches **layer 5**.



ICTAcademy
INFORMATION AND COMMUNICATIONS TECHNOLOGY

Connection-oriented Services/1

Connection-oriented services are those in which a **dedicated connection** is established between the two communication nodes before data transfer begins. This type of service ensures that data is **transmitted reliably** and in the **correct order**.



Connection-oriented Services/2

Key characteristics of connection-oriented services:

- **Reliability:**

Data transmission is guaranteed, and transmission errors are detected and corrected. Used in applications where reliability is more important than speed, such as email and online banking.

- **Ordering:**

Data arrives in the order it was sent.

- **Flow Control:**

Regulates the amount of data sent to prevent overwhelming the receiver.

- **Establishing and Closing the Connection:**

Requires an initial handshake process to establish the connection and a closing process to terminate it.

Connection-oriented Services/3

Handshaking (Opening the Connection)

1. **Connection request:**

Device A sends a request message to Device B to initiate the connection.

2. **Response:**

Device B responds to Device A, confirming that it has received the request and is ready to communicate.

3. **Confirmation:**

Device A sends a confirmation that it has received Device B's response.

Closing the Connection

1. Termination request:

When the communication is complete, Device A sends a termination request to Device B.

2. **Acknowledgment:**

Device B acknowledges the termination request, indicating it is ready to close the connection.

3. **Final confirmation:**

Device A sends a final confirmation, completing the termination process.

Connection-oriented Services/4

The **ACK** (**Acknowledgment**) service is a mechanism used to ensure that the data sent is correctly received by the recipient.

1. **Packet sending:**

Device A sends a data packet to Device B.

2. **Reception and acknowledgment:**

When Device B receives the packet, it sends an acknowledgment message (ACK) to Device A to indicate that the packet has been received correctly.

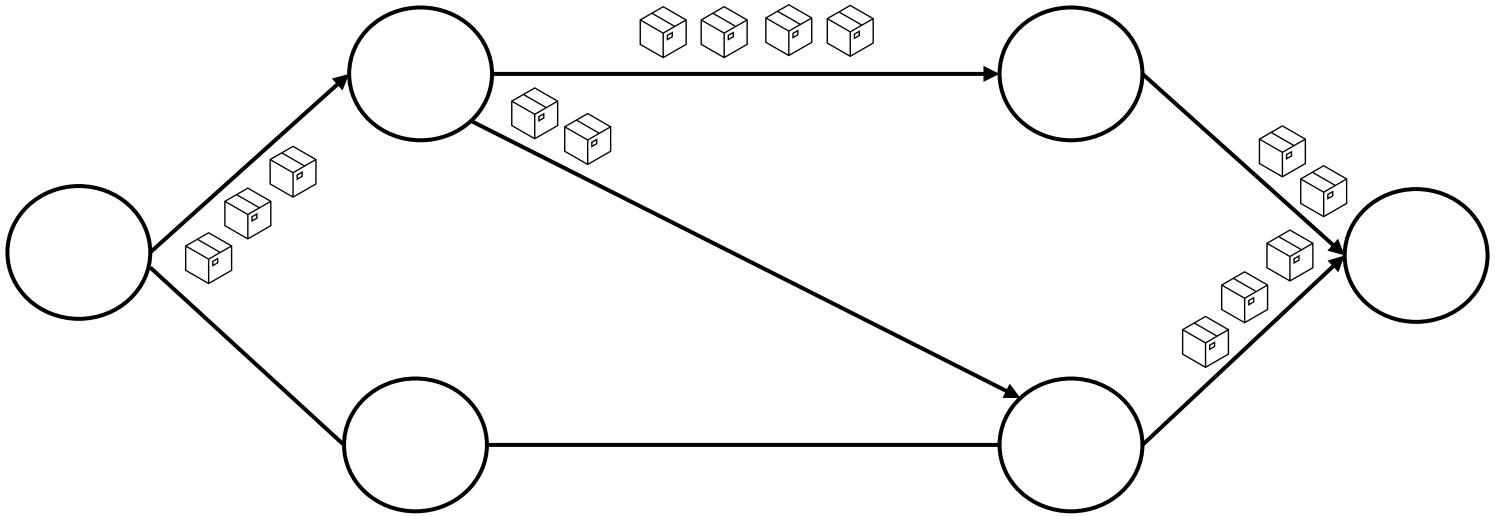
- **Retransmission in case of error:** If Device A does not receive the ACK within a certain period of time, it retransmits the packet, assuming it was lost or damaged during transmission.

This process continues **for each data packet** sent, ensuring that all packets are **received correctly** and in the **right order**.

Connectionless Services/1

Connectionless services do **not require** the establishment of a **dedicated connection** before sending data. Instead, each data packet is **treated independently** and contains all the necessary information to reach its destination.

Two packets with the same source and destination machines can travel through different paths and arrive at different times.



Connectionless Services/2

Key characteristics of connectionless services:

- **Simplicity and Speed:**

Does not require a handshake process, thus having lower latency compared to connection-oriented services.

- **No Delivery Guarantee:**

Packets can be lost, duplicated, or arrive out of order.

- **Packet Independence:**

Each packet is treated separately and contains all the necessary information to reach its destination.

- **Usage in Specific Applications:**

Used in applications where speed is more important than reliability, such as streaming and online gaming.

3. Computer Architectures

Fundamentals of Operating Systems

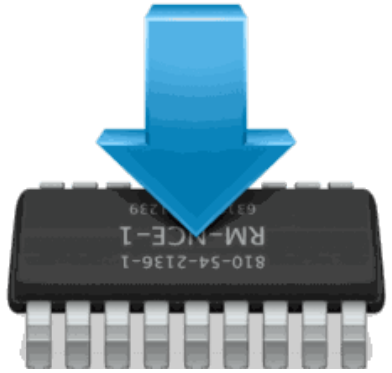
Summary

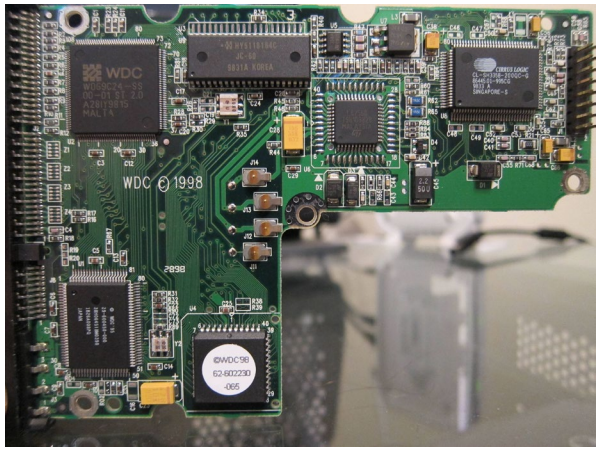
- Firmware
- Non-volatile Memory
- Motherboard
- Basic Input/Output System (BIOS)
- Unified Extensible Firmware Interface (UEFI)
- Drivers
- Operating System
- Operating System Structure
- Operating System Kernel

Firmware/1

Firmware is a type of software that is **embedded** within an **electronic component**. In this context, electronic component refers to any **intelligent integrated circuit** present in various devices, including **processors**, **graphics cards**, **sound cards**, **network interface cards**, and **peripheral devices** like printers and monitors.

The **primary function** of firmware is to **initialize the component** and **facilitate its interaction** with other components within the device. Essentially, firmware provides the component with a **set of communication interfaces** and **protocols**, allowing it to use a common language (encompassing syntax and semantics) to communicate effectively with other components.





Firmware/2

The term **firmware** is a combination of '**firm**' and '**ware**' indicating that firmware is **not easily modified** by the user. It serves as a **bridge** between hardware and software, enabling the **hardware components** to **communicate** and **function** according to **the software instructions**.

The **BIOS (Basic Input/Output System)** is a well-known example of firmware in computers. It resides on the **motherboard** and is crucial for managing **the initial startup process** of the computer, including **hardware initialization** and **booting the operating system**.

Inside every device, whether it is a desktop PC, a laptop, a car's onboard computer, or an airplane's internal computer, there are **similar fundamental components,** such as **motherboards**. These components may **differ in size** and **technical specifications**, but they perform **analogous functions**.

Firmware is typically stored in **non-volatile memory** within the electronic component or device, meaning the memory retains its contents even when the power is turned off.

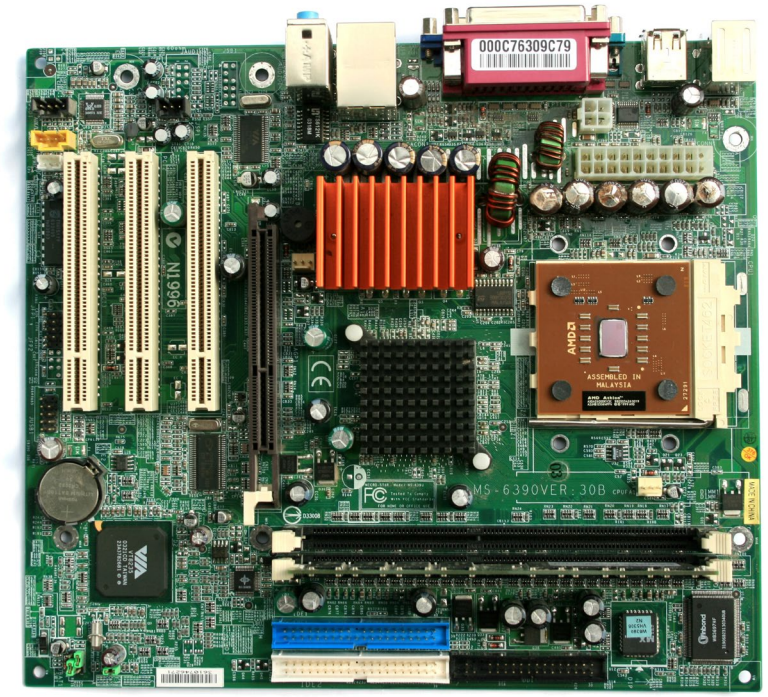
Non-volatile Memory

Common types of non-volatile memory used for storing firmware include:

- **ROM (Read-Only Memory):** Traditionally used for firmware storage, but once written, it cannot be modified.
- **EPROM (Erasable Programmable Read-Only Memory):** Can be erased and reprogrammed using UV light, but this process is not as convenient as modern alternatives.
- **EEPROM (Electrically Erasable Programmable Read-Only Memory):** Can be erased and reprogrammed using electrical charge, making it more flexible than EPROM.
- **Flash Memory:** A type of EEPROM that can be erased and rewritten in blocks, commonly used in modern devices due to its flexibility and higher storage capacity.

These storage types are **embedded** on the device, ensuring that the firmware is **readily accessible** to **initialize** and **control** the device components when powered on.

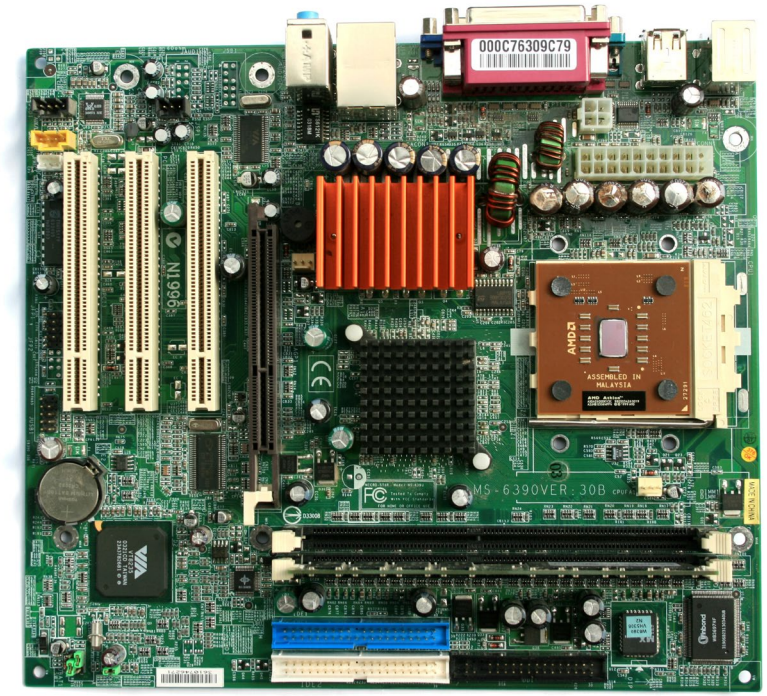
Motherboard/1



A **motherboard** is the main **printed circuit board** (PCB) in a computer or other electronic device that holds and allows communication between many of the crucial electronic components.

- **Central Hub:** It acts as the central hub that connects all the parts of the computer together, allowing them to communicate and work together.
- **CPU Socket:** The motherboard contains a socket that holds the central processing unit (CPU). This is where the CPU is installed.
- **Memory Slots:** It has slots for memory (RAM) modules, allowing the computer to temporarily store and quickly access data.
- **BIOS/UEFI:** The Basic Input/Output System (BIOS) or Unified Extensible Firmware Interface (UEFI) firmware is stored on the motherboard, providing the necessary instructions for the initial boot-up process and hardware initialization.

Motherboard/2



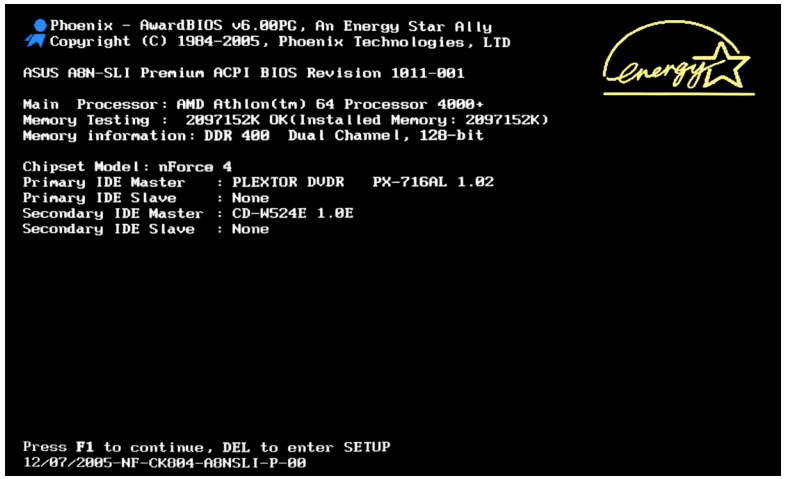
- **Expansion Slots:** These slots allow additional cards to be added, such as graphics cards, sound cards, network cards, and other peripherals.
- **Storage Connectors:** The motherboard has connectors for storage devices, like hard drives and SSDs.
- **Power Connectors:** It includes connectors to supply power to the motherboard and its components from the power supply unit (PSU).
- **Peripheral Connectors:** Ports and connectors for peripherals such as USB devices, audio jacks, network cables, and display connections (HDMI, DisplayPort).

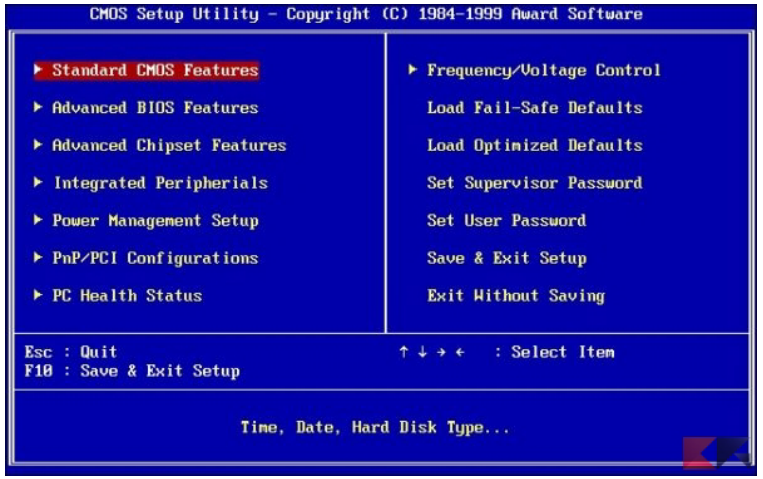
In summary, the motherboard is the **backbone** of a computer, integrating all the critical components and ensuring they can communicate and function together effectively.

Basic Input/Output System (BIOS)

The **Basic Input/Output System (BIOS)** is a set of **software routines** that provides the **foundational software structure** allowing the **Operating System** to **interface** with the device's **hardware**.

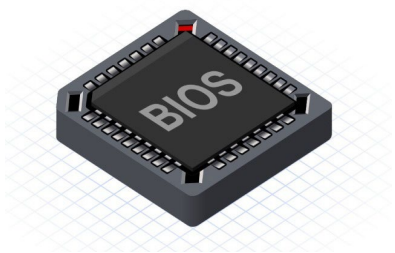
The **BIOS** is stored in a **non-volatile memory** chip, meaning it retains data even without power. Nowadays, the **BIOS** is typically stored on **FLASH** or **EEPROM** memory soldered directly onto the **motherboard**, which can be rewritten using an appropriate upgrade procedure.





BIOS – BOOTSTRAP/1

When a device is started, **before** the Operating System loads, the **BIOS** performs several **key tasks**:



- **Manage Hardware:** It runs various **test routines** called **POST (Power On Self Test)** to verify the proper functioning of different components, including the motherboard, processor, memory, and others.
- **Configure Hardware Settings:** If no serious errors are detected during the POST phase, the BIOS configures the hardware settings, preparing the system for the operating system to take over.

- **Display Startup Screen:** It presents a video screen that allows the user to configure and manage some basic BIOS settings.
- **Initialize Boot Routine:** The BIOS starts the **boot routine** to load the **Operating System**. The boot process is incremental, loading the essential parts of the Operating System from the boot sector to bring the system to a stable operating state.

BIOS – BOOTSTRAP/2

- **Locating MBR:** After executing the basic BIOS routines, the BIOS's first instruction is to locate the **'Starting Point'** of the hard disk (or other mass storage media) where the Operating System is stored. At this location, the **Master Boot Record (MBR)** can be found.

- **Control to Operating System:** The BIOS reads the content of the MBR and transfers control to the Operating System (from this point, the device management is fully handled by the Operating System).

The **MBR** contains information about the **location** of the **BOOT SECTOR** on the hard disk (or other mass storage media hosting the Operating System).

The **BOOT SECTOR** is the **entry point** for starting the **Operating System**.

If the **MBR** table is damaged, the disk might not boot.

Unified Extensible Firmware Interface (UEFI)/1

The **Unified Extensible Firmware Interface (UEFI)** is a modern **firmware** interface designed to **initialize hardware** components and **load the operating system** when a device starts up.

Similar boost process BUT with advantages over BIOS:

- **Graphical User Interface (GUI):** UEFI can offer a more user-friendly graphical interface with mouse support, making it easier to navigate and configure settings.
- **Boot Manager:** A built-in program within UEFI that determines which operating system or boot loader to execute. This helps streamline the boot process and offers more flexibility compared to BIOS.
- **Driver Support:** UEFI can load drivers directly, enabling better hardware compatibility and performance during the boot process.
- **Enhanced Security:** UEFI includes features like Secure Boot, which helps protect the system by ensuring that only trusted software is loaded during the boot process.
- **Extensibility and Updates:** UEFI is modular and can be updated or extended with new functionalities, providing better support for new technologies and improved system stability over time.

Unified Extensible Firmware Interface (UEFI)/2

The **GUID Partition Table (GPT)** is a modern system **for managing partitions** on a hard drive or SSD.

It is part of the **UEFI (Unified Extensible Firmware Interface)** standard and is the successor to the old **MBR (Master Boot Record)**.

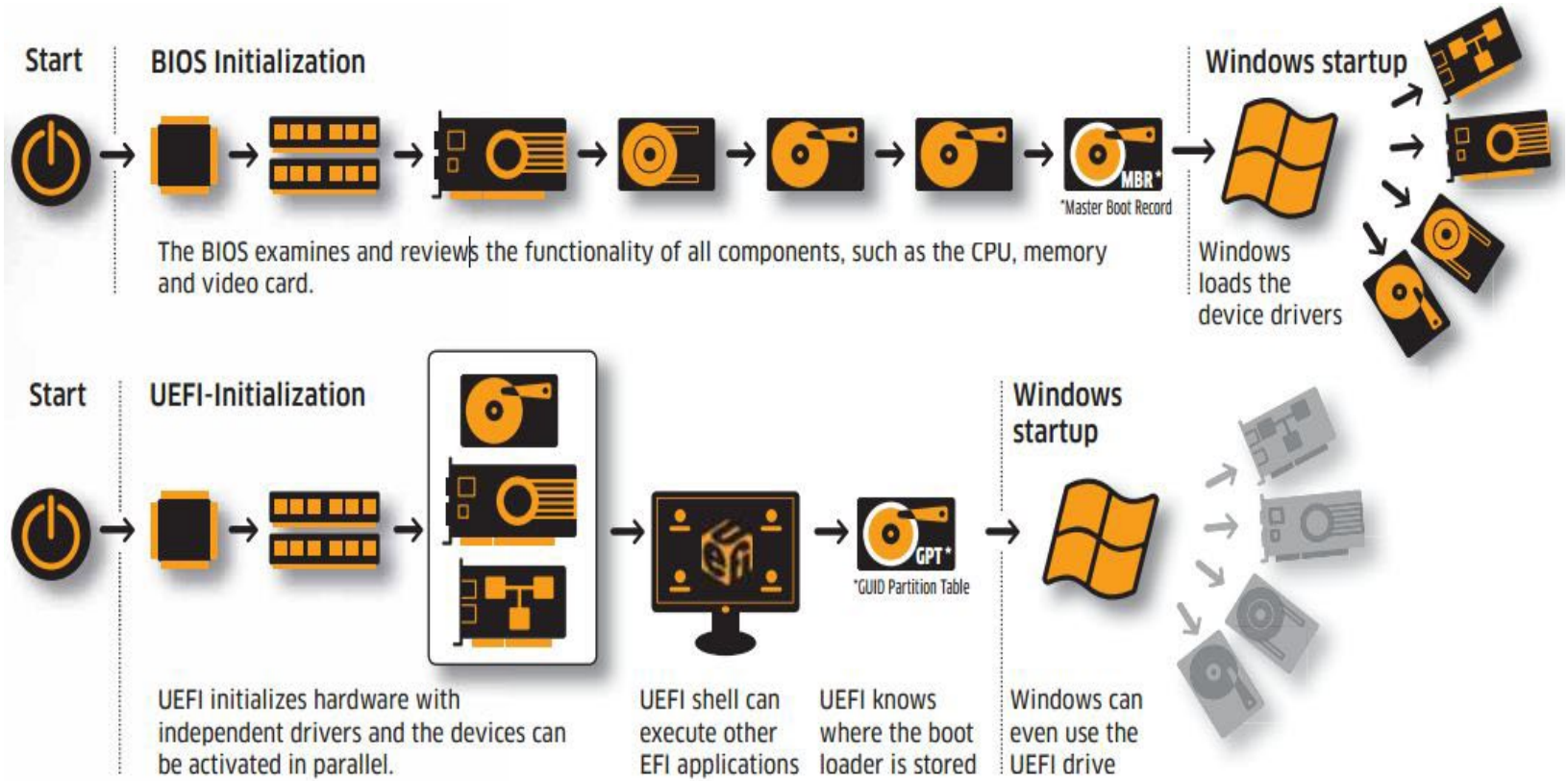
📌 **GUID** means **Globally Unique Identifier**, a globally unique identifier assigned to each partition.

GPT organizes the disk into a more advanced structure compared to MBR, including:

- **Primary GPT header** – Stores the position and structure of the partitions.
- **Partition entries** – Each partition has a GUID and a description.
- **Secondary GPT header (backup)** – A backup copy of the primary header located at the end of the disk.

🚀 This structure provides greater security and flexibility compared to MBR!

Unified Extensible Firmware Interface (UEFI)/3



ICTAcademy
INFORMATION AND COMMUNICATIONS TECHNOLOGY

Drivers

Drivers are software programs that allow the **operating system** and **other software** to communicate with **hardware** devices.

Drivers act as **intermediaries**, translating high-level commands from the OS into low-level commands that the hardware can understand. They **manage the interaction** between the **operating system** and the **hardware**.

Characteristics:

- Installed on the operating system.
- Facilitates communication between OS and hardware.
- Can be updated or replaced more frequently.

Firmware vs Drivers

Key Differences:

- **Location:**

Firmware: Embedded in the hardware.
Drivers: Installed on the operating system.

- **Function:**

Firmware: Manages and controls hardware at a fundamental level.
Drivers: Translates OS commands into actions performed by the hardware.

- **Update Frequency:**

Firmware: Updated less frequently, often through specialized update processes.
Drivers: Can be updated regularly to improve performance or add new features.

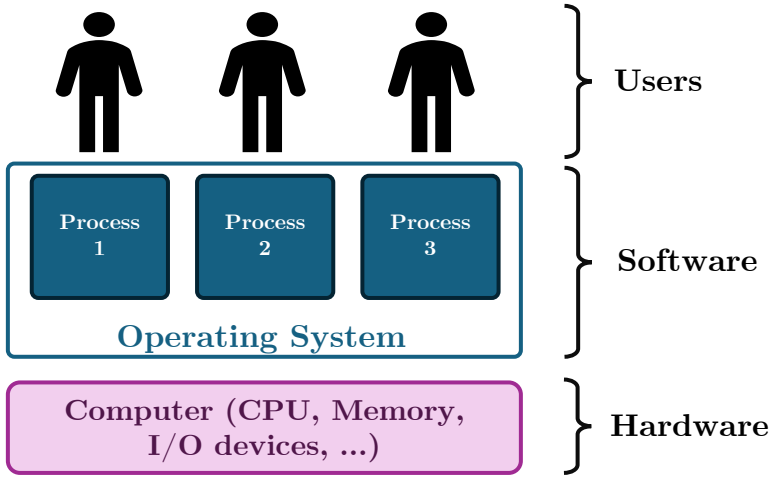
Operating System/1

The **Operating System** is the **fundamental software** responsible for **managing** the entire **device's operations**, including **running applications**, **programs**, and **services**, as well as **interacting** with the user.

Positioned between the **hardware** layer (including firmware) and the **application** layer (applications, programs, services), the Operating System's key role is to abstract the computer's complexities for the application layer.

Specifically, the Operating System handles four **primary tasks**:

1. **Managing** hardware components
2. **Executing** applications, programs, and services
3. **Interfacing** with peripheral devices
4. **Facilitating** user interaction



Operating System/2

In general, the Operating System defines **two main aspects** of a device:

- **Operational Modality of the Device**

This modality determines the purpose and scope of the device, explaining why a specific device is used.

- **User Interaction**:

This modality defines how users can engage with and operate the device.

Examples of Operating Systems include:

- Microsoft Windows,
- GNU/Linux,
- macOS,
- Android,
- iOS

... and many others.

Classification Based on Processing Features

- **Batch Operating Systems**: These Operating Systems run programs in a non-interactive manner. Users load both programs and data, and results are provided only after processing is completed.
- **Interactive Time-Sharing Operating Systems**: These Operating Systems run multiple programs simultaneously, sharing hardware resources (such as CPU time and memory) among them. Users can interact with programs during execution, providing input and receiving output throughout the process.
- **Real-Time Operating Systems**: These Operating Systems ensure that programs execute in real-time (or within a fixed time interval) without delays, latencies, or performance degradation. They are typically used in industrial processes or applied robotics.
- **Embedded Operating Systems**: These Operating Systems are integrated within dedicated hardware, often found in specific devices like cars or airplanes.
- **Hypervisor Operating Systems**: These Operating Systems enable the subdivision of hardware resources into different virtual machines, allowing multiple OSs, programs, and operational modes to run concurrently.

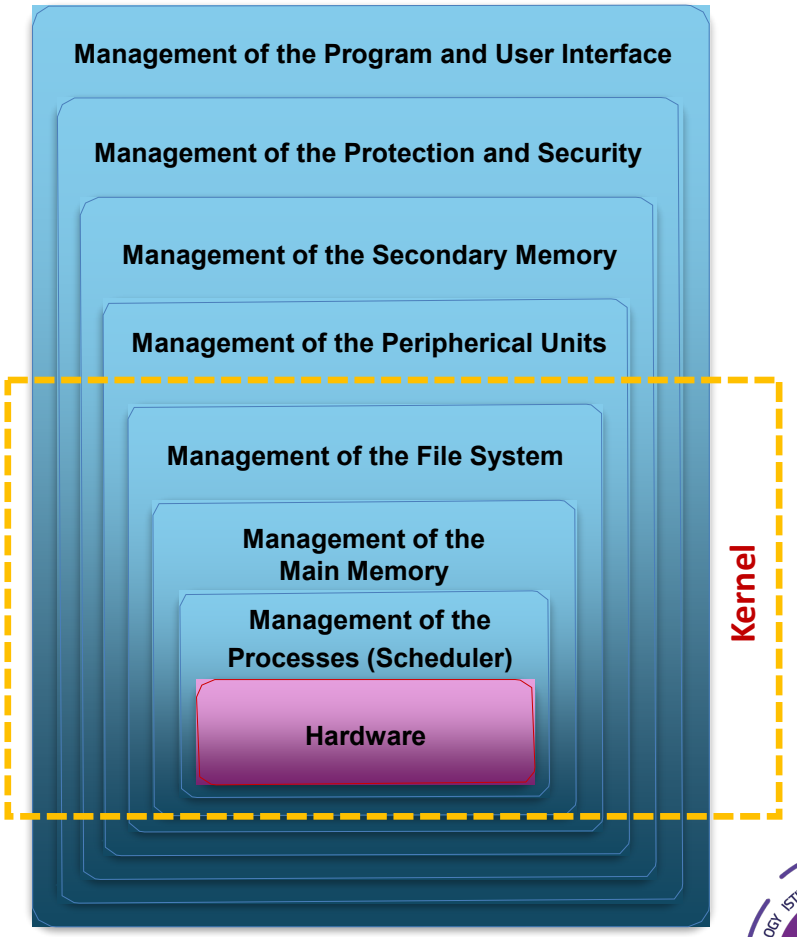
Classification Based on Functional Characteristics

- **Mono-Task Operating Systems:** These operating systems allow only one program (i.e., task) to be executed at a time. A new program can only start once the current one has finished.
- **Multi-Task Operating Systems:** These operating systems enable the parallel execution of multiple programs. Hardware resources (e.g., CPU time, memory) are shared among various programs, allowing them to run simultaneously.
- **Multi-Threading Operating Systems:** In these operating systems, each single program is divided into multiple sub-programs (i.e., threads). Each thread is executed on the CPU (or on a core), enhancing the device's performance.
- **Mono-User Operating Systems:** These operating systems are designed to interact with and execute programs for only one user at a time.
- **Multi-User Operating Systems:** These operating systems can interact with and execute programs for multiple users simultaneously.

Operating System Structure

An **Operating System** can be structured as follows:

1. **Management** of Programs and User Interface
2. **Management** of Protection and Security
3. **Management** of Secondary Memory
4. **Management** of Peripheral Units
5. **Management** of the File System
6. **Management** of Main Memory
7. **Management** of Processes (Scheduler)

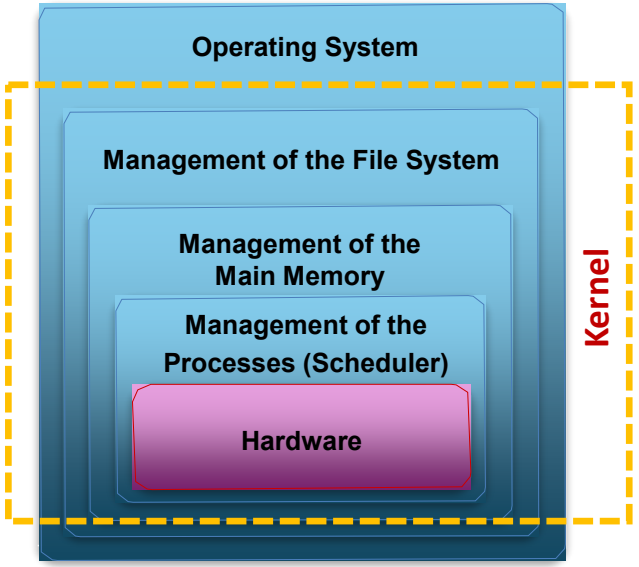


Operating System Kernel

The **kernel** is the **central component** of the **Operating System**. It serves as a **bridge** between **applications** and **physical hardware**, ensuring smooth and coordinated task execution.

Based on their **internal structure** and **kernel design**, Operating Systems can be categorized into three types:

- **Monolithic** kernel Operating System
- **Micro-kernel** Operating System
- **Hybrid** kernel Operating System



Monolithic Kernel Operating System

Monolithic kernel Operating System:

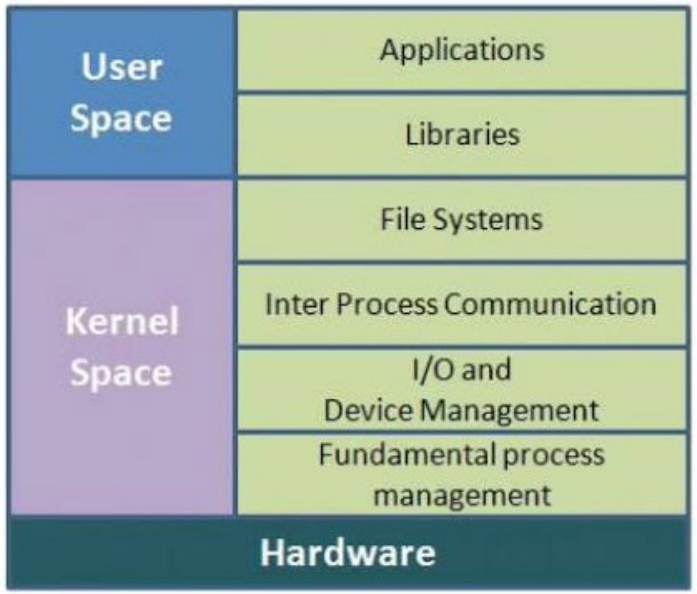
This system implement a **complete abstraction** of the computer within the **kernel**, which runs the operating system itself.

Monolithic kernel includes all the **core functionalities** required for the operating system to manage hardware and system processes within the **Kernel Space**. This includes device drivers, memory management, file system management, and system calls, all running in a single address space. **Applications** and **libraries** run in **User Space**.

By running all essential services in kernel space, monolithic kernels can achieve **high performance** due to minimal context switching between user mode and kernel mode. This **reduces overhead** and **improves efficiency**.

Since all components run in the same address space, a **failure** or **bug** in one part of the **kernel** can potentially **crash** the entire system.

Monolithic kernel is less modular compared to micro-kernel. **Adding new features** or **drivers** often requires **modifying** and **recompiling** the entire kernel, which can be cumbersome and limits flexibility.



Micro-kernel Operating System

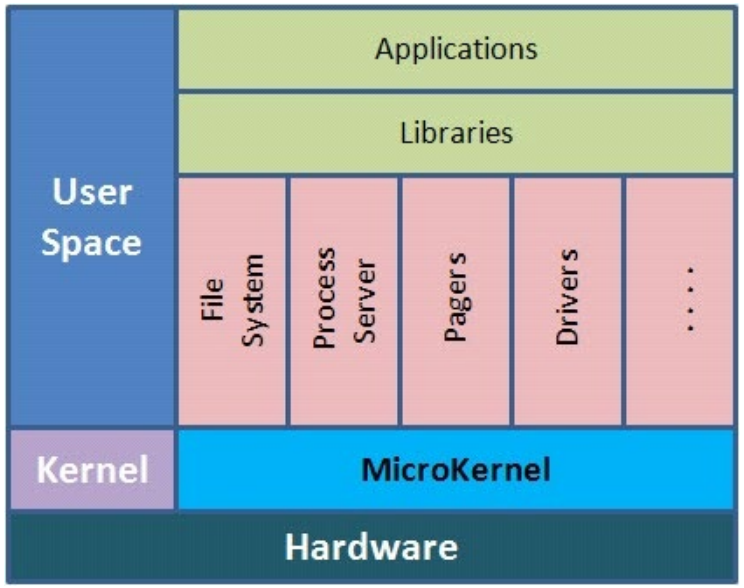
Micro-kernel Operating System:

This system implements only the **essential** Operating System **functions** in the **kernel**. Other functionalities are provided by separate system programs such as device drivers and servers.

Micro-kernel is designed to be **modular**. By keeping only **essential functions** like inter-process communication, basic scheduling, and minimal hardware abstraction in the **Kernel Space**, **other functionalities** are handled by **separate User-Space processes**. This design enhances **modularity** and allows for easier **updates** and **maintenance**.

By **isolating drivers** and **system services** from the kernel, the micro-kernel can **increase** system **stability** and **security**.

Failures or **bugs** in one service **do not** necessarily **crash** the entire system, as they would in a monolithic kernel.



Hybrid kernel Operating System

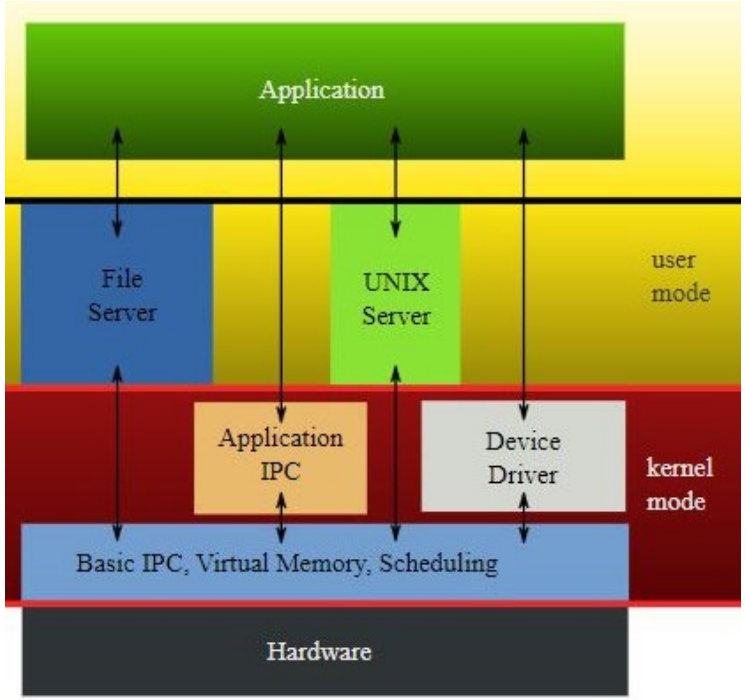
Hybrid kernel Operating System:

This system implements **various** Operating System **functionalities** within the **kernel** but can load **additional modules** at boot time to enhance and complete the kernel's features.

Hybrid kernel provides a **balance** between the monolithic and micro-kernel architectures. By including **essential functionalities** within the kernel and allowing **additional modules** to be **loaded** at **boot time**, they offer a flexible approach to system design.

This architecture aims to achieve the **performance** benefits of monolithic kernels while maintaining a degree of **modularity**.

Operating systems like modern versions of **Windows** and **macOS** use a **hybrid kernel** approach. These systems demonstrate the practical implementation and advantages of hybrid kernels in real-world scenarios.



3. Computer Networks

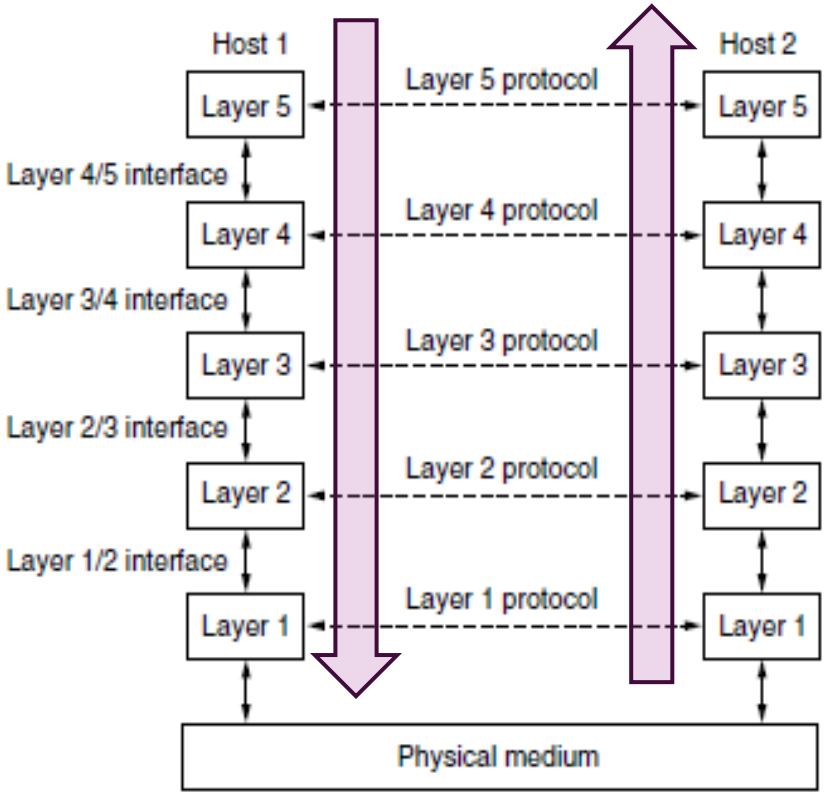
ISO/OSI Model, Physical Layer, Data Link Layer

Summary

- Network Architecture
- Network Ownership and Standards
- ISO/OSI Reference Model
- Physical Layer
- Frequency and Bandwidth
- Bandwidth-delay Product
- Data Link Layer
- Logical Link Control (LLC)
- Media Access Control (MAC)
- Network Switch

Network Architecture

The **network architecture** is a framework that outlines the **structure** and **operation** of a network, consisting of a **set of layers** and **protocols** that dictate how data is transmitted and received.



Network Ownership and Standards

- **Network Ownership:**

Networks where the design and operational decisions are independently made by the manufacturer.

Choices are arbitrary and can vary widely between manufacturers.

- **Standard de facto:**

Public domain specifications widely accepted and used by the industry (TCP/IP model).

- **Standard de iure:**

Specifications approved by international standardization organizations and are in the public domain (ISO/OSI model).

ISO/OSI Reference Model - Introduction

The **OSI (Open Systems Interconnection) model** was established in 1984 and is the result of work by the **ISO (International Organization for Standardization).** It is a **standard de iure**. Its purpose is to:

- Provide a **standard model** against which various network architectures can be compared.

The **ISO/OSI model** is an **abstract description** for layered communication and computer network protocol design. It divides the network architecture into **seven layers**.

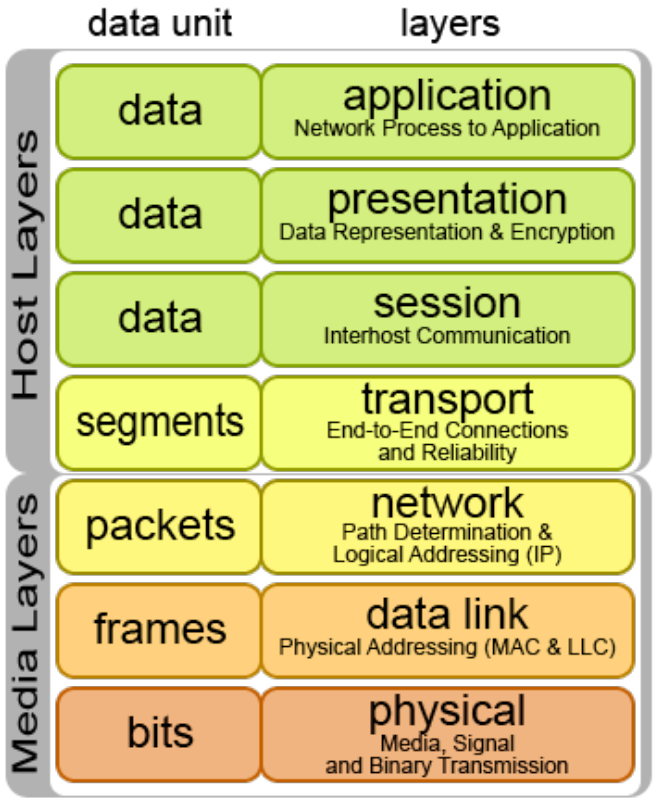
ISO/OSI model was designed according to following principles:

- Each layer must have a different **level of abstraction**;
- Each layer must have a **well-defined function**;

The choice of layers must:

- **Minimize** the information passing between layers;
- **Avoid** too many functions in one layer;
- **Avoid** too many layers.

ISO/OSI Reference Model

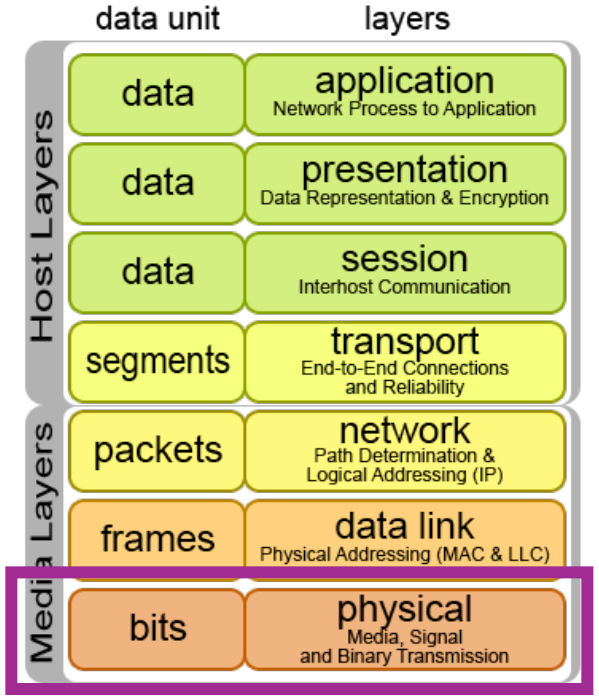


Physical Layer

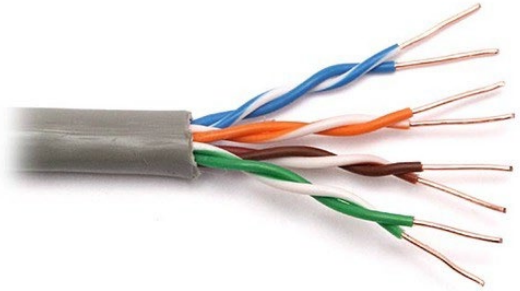
The **physical layer** is responsible for the **physical connection** between devices. It deals with the **transmission** and **reception** of **raw bitstreams** over a physical medium.

Main physical mediums:

- **Copper** cables (e.g., Ethernet, Coaxial cable);
- **Fiber** optic cables;
- **Radio** frequencies (e.g., Wi-Fi, Bluetooth).



Ethernet and Coaxial Cables

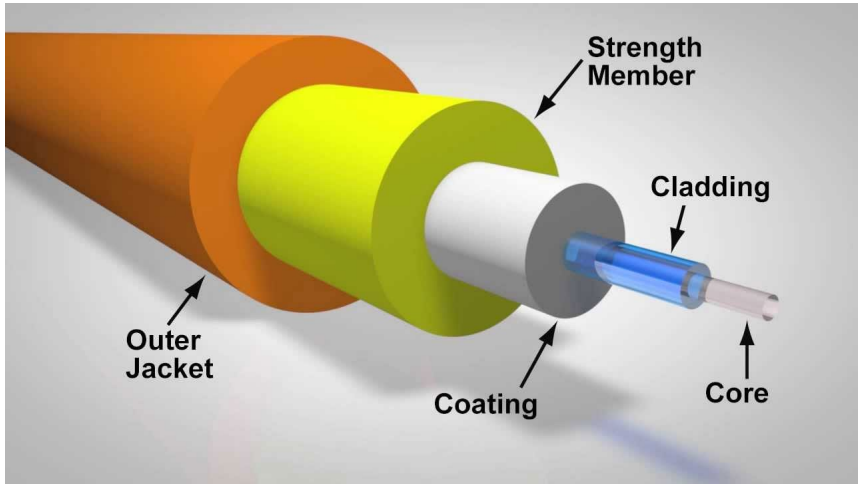


Ethernet cables are types of **electrical cables** used to connect devices within a **local area network (LAN)** for data transmission. They use **twisted pair** wiring to **reduce interferences** and **improve signal quality**.

Coaxial cable is a type of **electrical cable** consisting of a **central copper** conductor, an **insulating layer**, a **metallic shield**, and an **outer insulating layer**. This construction provides **excellent protection** against electromagnetic interferences, ensuring **signal integrity** over **long distances**. It also offers **higher** data transmission **speeds**.

Fiber Optic Cable

Fiber optic cable is a type of network cable that uses **light** to transmit data at **high speeds over long distances**. It consists of **strands of glass** or **plastic** fibers, each capable of carrying data signals in the form of **light pulses**. Fiber optic supports much **higher data rates** and is also **immune** to **electromagnetic interferences**, ensuring clean and reliable data transmission.



- **Core:** The central part of the fiber, made of glass or plastic, through which light signals travel.
- **Cladding:** Surrounds the core and reflects light back into the core, minimizing signal loss.
- **Coating:** Protects the fiber from damage and moisture.
- **Outer Jacket:** Provides additional protection against environmental factors.

Radio Frequencies (Wi-Fi, Bluetooth)

Radio frequency (RF) data transfer involves transmitting data wirelessly through **electromagnetic waves**. **Wi-Fi** and **Bluetooth** are two common technologies that use RF to enable **wireless communication** between devices.

- **Wi-Fi:**

Frequency Bands: Operates primarily in the 2.4 GHz and 5 GHz frequency bands, with newer standards also using the 6 GHz band.
Range: Typically covers a range of up to 100 meters indoors, depending on the environment and obstacles.
Network Type: Typically forms part of a local area network (LAN) where multiple devices connect to a central router or access point.
Applications: Internet access, streaming services, online gaming, smart home connectivity, and enterprise networking.

- **Bluetooth:**

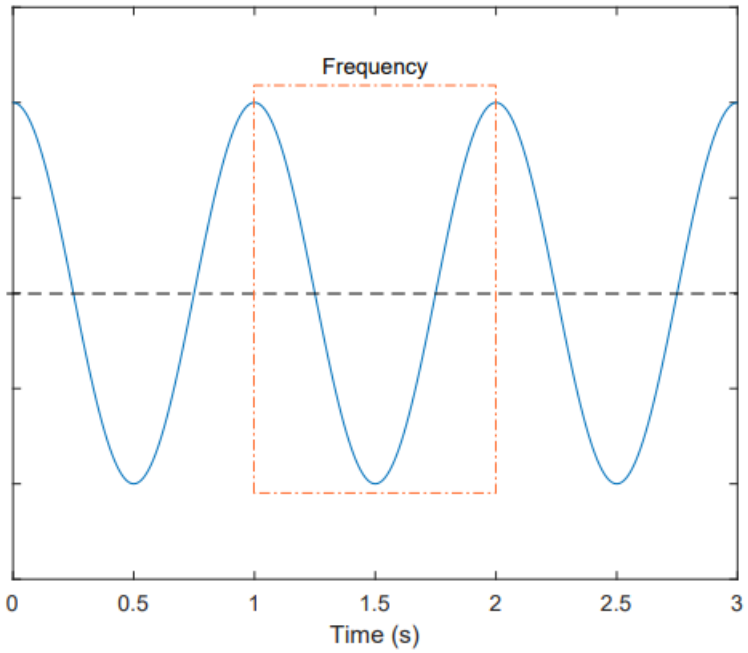
Frequency Bands: Operates in the 2.4 GHz ISM (Industrial, Scientific, and Medical) band.
Range: Typically covers a range of up to 10 meters, with some versions (Bluetooth 5) extending up to 100 meters.
Network Type: Typically forms a personal area network (PAN) where devices connect directly to each other or through a central device in a star topology.
Applications: Connecting peripherals (e.g., headphones, keyboards), file transfer between mobile devices, wireless audio streaming, and smart home device control.

Advantages of RF Data Transfer:

- Eliminates the need for physical cables, providing greater mobility and convenience.
- Simplifies the process of connecting and communicating between devices.

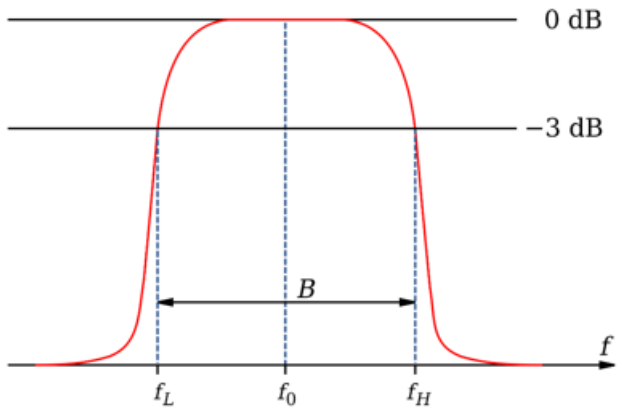
Frequency

Frequency refers to the **number of cycles (oscillations)** of the wave that pass a specific point in **one second**. It is measured in **Hertz (Hz)**, where one Hertz equals one cycle per second.



Bandwidth

Bandwidth is the **range of frequencies** within a given band that a signal occupies. It is the difference between the highest and lowest frequencies in the range and is measured in **Hertz (Hz)**.

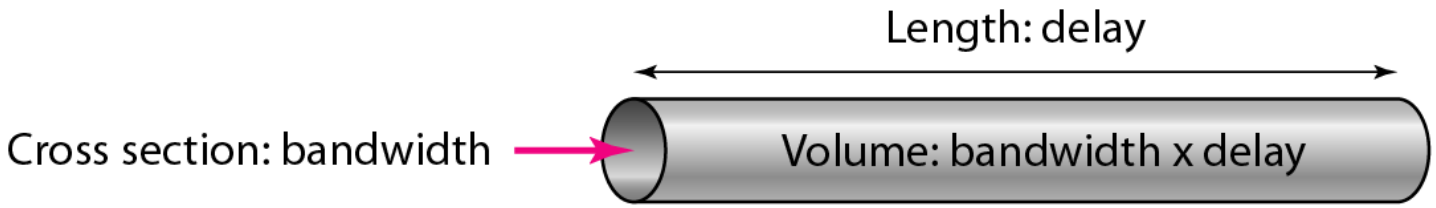


Bandwidth defines the **amount of data** that can be **transmitted** over a network connection in a **given amount of time**, typically measured in **bits per second (bps). Higher bandwidth** indicates a greater capacity to **carry more data**, resulting in **faster data transmission** and **higher performance** in network communications.

The **actual bandwidth** experienced by users can be **lower** due to factors like **network congestion**, **distance from the router**, **interference**, and **device capabilities**.

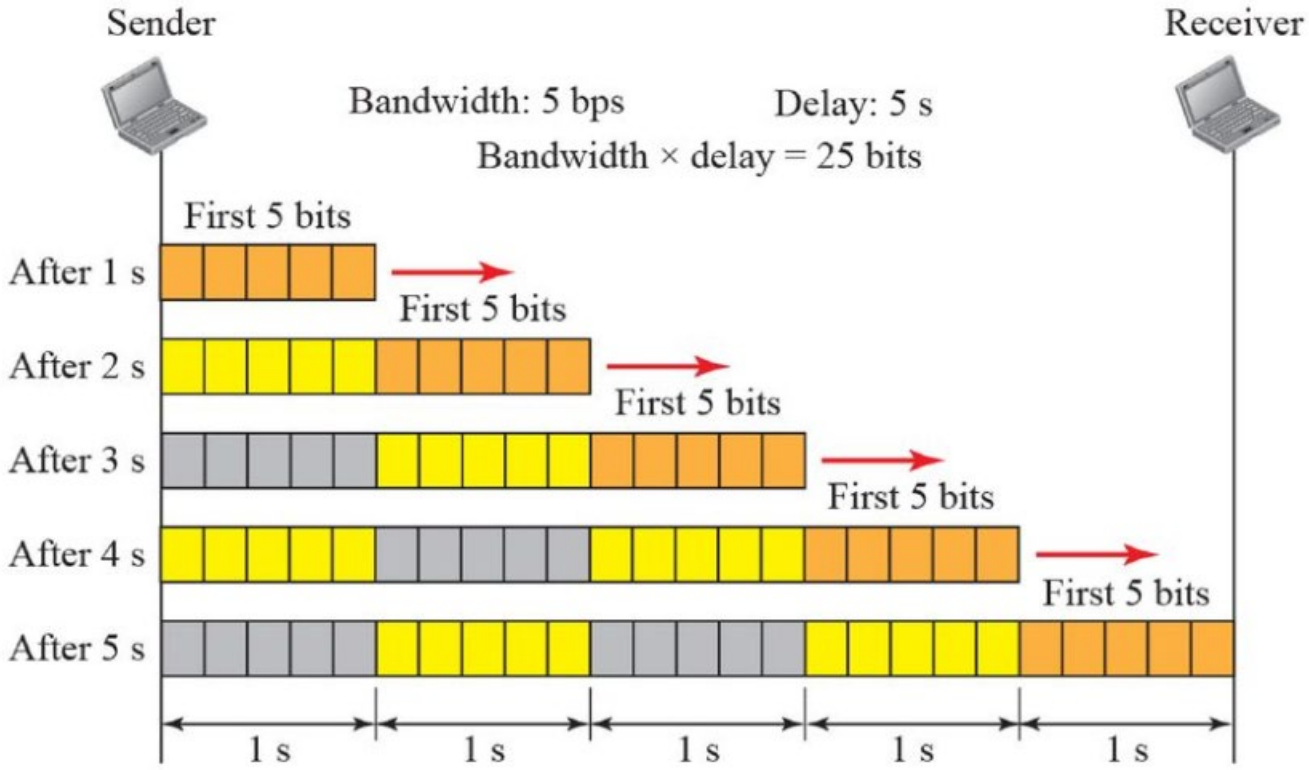
Bandwidth-delay Product/1

The **bandwidth-delay product** defines the **maximum amount of data (in bits)** that can be in **transit** in the network link at **any given time**.



- **Bandwidth:** The cross section of the pipe (how much data can flow through per unit of time).
- **Delay:** The length of the pipe (the time it takes for data to travel from one end to the other).
- **Bandwidth-Delay Product:** The volume of the pipe (total amount of data that can be in transit at any given time). The longer the pipe (greater delay), the longer it takes for data to travel through it.

Bandwidth-delay Product/2

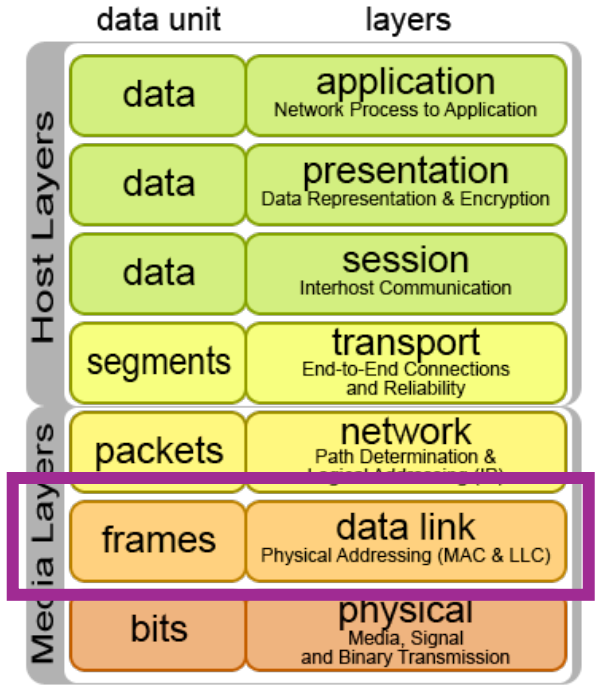


Data Link Layer

The **Data Link** layer is the second layer of the OSI model, responsible for **node-to-node data transfer** and **error detection** and **correction**.

Functions:

- **Establishes** and **terminates logical link** connections.
- Manages **data frames** between devices on the same network.



ICTAcademy
INFORMATION AND COMMUNICATIONS TECHNOLOGY

Functions of the Data Link Layer

- **Framing:**

Divides data into **frames** for easier transmission.

Adds headers and **trailers** to frames to facilitate communication.

- **Error Detection and Correction:**

Detects errors in transmitted frames using techniques like CRC (Cyclic Redundancy Check).

Corrects errors to ensure reliable data transfer.

- **Flow Control:**

Manages data **flow** to prevent congestion and data loss.

- **Addressing:**

Uses **MAC (Media Access Control)** addresses to **identify** devices on the network.

Sub-layers of the Data Link Layer

- **Logical Link Control (LLC):**

Handles **error checking** and **flow control**.

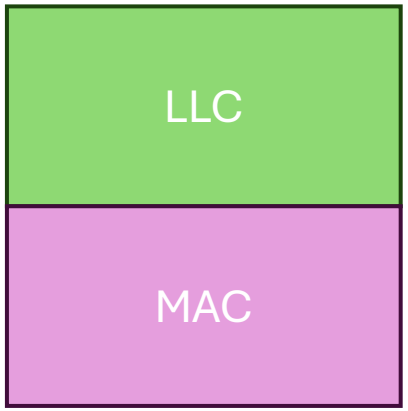
Divides data into **frames** for easier transmission.

Adds **headers** and **trailers** to frames to facilitate communication.

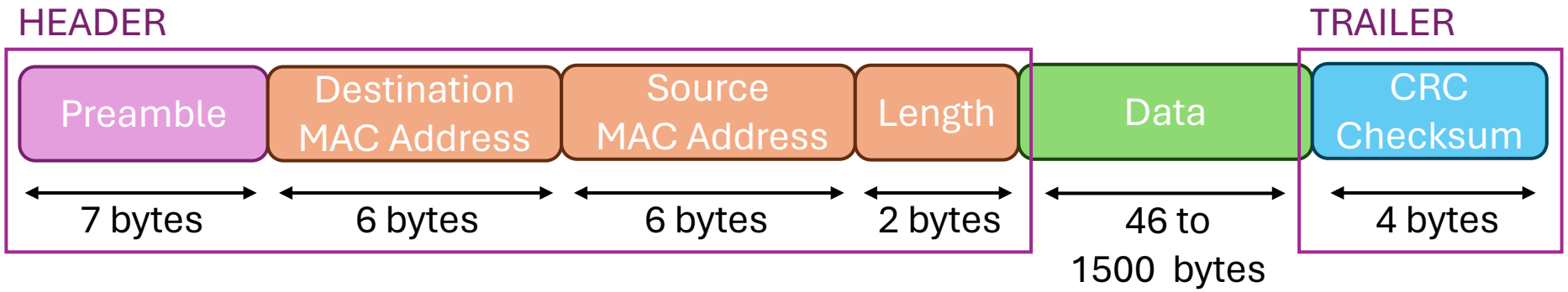
- **Media Access Control (MAC):**

Controls how **devices** on the network **gain access** to the medium.

Determines how data is placed on the medium.



LLC - Framing



- **Preamble:** The preamble is a series of bits that helps the receiving device get ready to understand the incoming data. It ensures both the sender and receiver are in sync before the actual message starts.
- **Destination MAC address:** the physical address of the destination device on the network.
- **Source MAC address:** the physical address of the sending device on the network.
- **Length:** the size of the data payload in the frame.
- **Data:** the field containing the actual payload being transmitted. It can include upper-layer headers.
- **CRC Checksum:** the checksum is a value calculated from the frame's contents to detect errors in transmission. If the calculated checksum at the receiving end does not match the transmitted checksum, it indicates that the frame has been corrupted

LLC - Flow Control Mechanisms

- **Stop-and-Wait**

Description: The sender transmits one frame and waits for an acknowledgment from the receiver before sending the next frame. This ensures that each frame is received and acknowledged before the next one is sent, preventing data overflow and ensuring orderly delivery.

Advantages: Simple and easy to implement.

Disadvantages: Inefficient for high-speed networks due to waiting time.

- **Sliding Window**

Description: Allows multiple frames to be sent before requiring an acknowledgment. The sender can transmit several frames specified by a "window" size. After sending these frames, the sender waits for acknowledgments. As acknowledgments are received, the window slides forward, allowing the sender to transmit more frames.

Advantages: More efficient use of network resources, better performance in high-speed networks.

Disadvantages: More complex to implement.

Media Access Control (MAC) Sub-layer

The **MAC sub-layer** controls how data is **placed onto** and **retrieved from** the network medium.

Functions:

- **Addressing:** Uses MAC addresses to identify sending and receiving devices on the network.
- **Access Control:** Determines how devices share the network medium to avoid collisions (when two devices try to send data simultaneously).

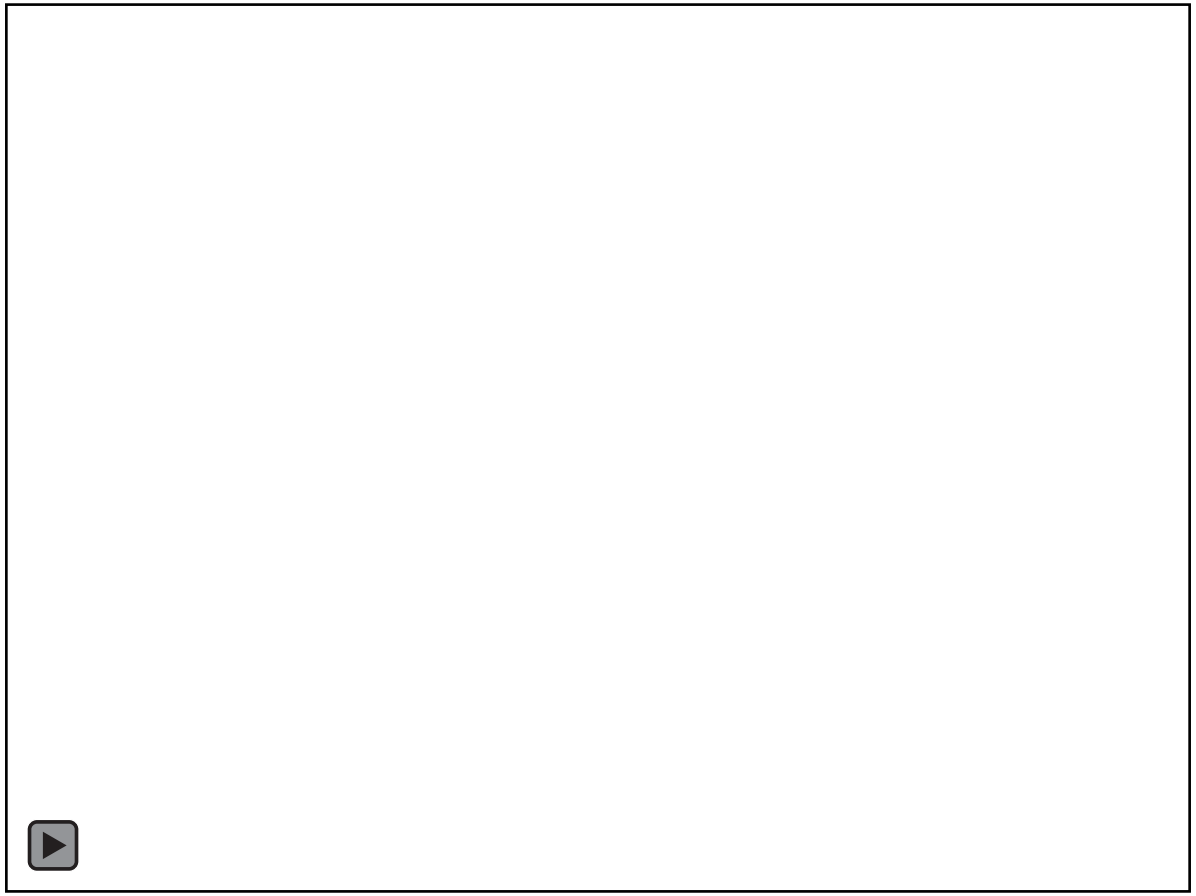
  Example methods:

  - **CSMA/CD (Carrier Sense Multiple Access with Collision Detection):** Used in Ethernet networks to manage data transmission and detect collisions.

  - **CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance):** Used in Wi-Fi networks to avoid collisions by waiting for a clear channel before transmitting.

- **Frame Delimiting:** Defines the start and end of a frame, ensuring that data is correctly interpreted.

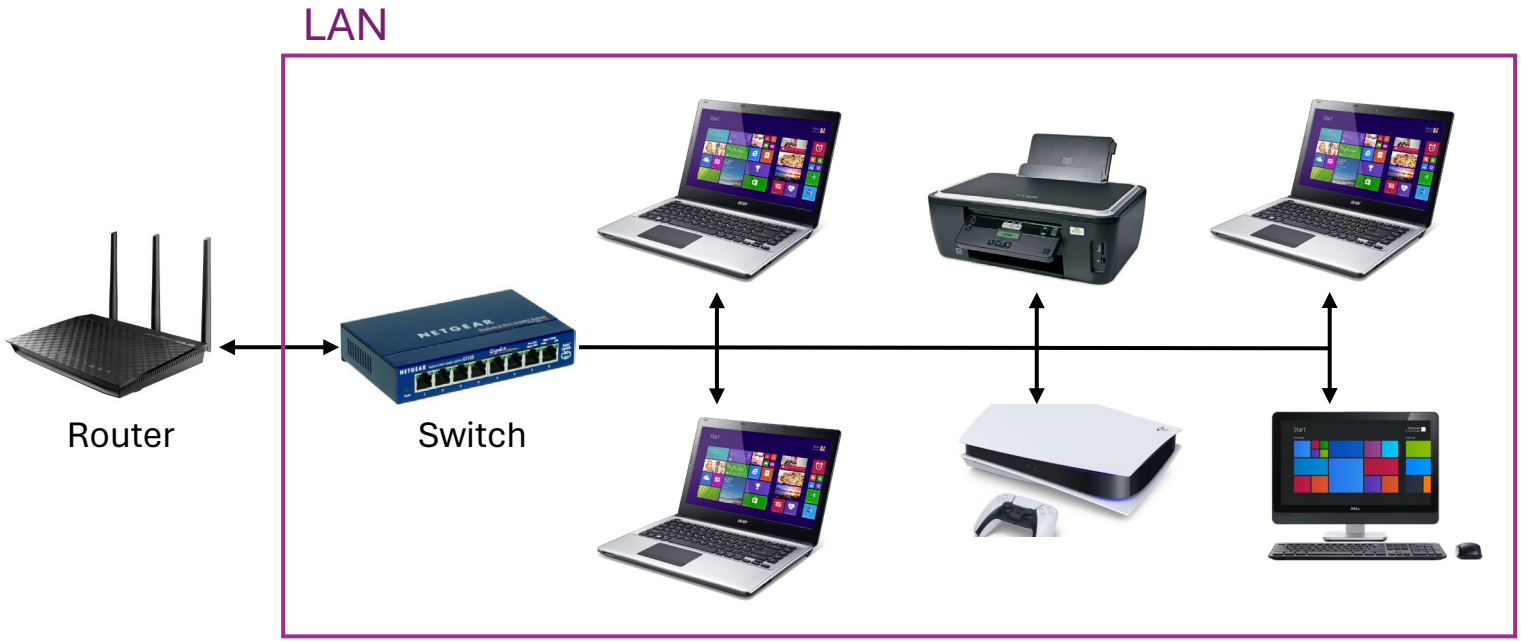
MAC - CSMA/CD and CSMA/CA



Network Switch

Network switches are fundamental devices in a **Local Area Network (LAN)**. They enable **communication** between different devices within the same network.

A **network switch** is a device that **receives**, **processes**, and **forwards** data to specific devices within a local network. It operates at the **Data Link Layer** of the **ISO/OSI model**.



Network Switch – Basic Operations

Basic Operation of a Switch:

- **Receives** a data frame from a connected device.
- **Reads** the destination MAC address of the frame.
- **Looks up** the MAC address in its MAC address table to determine the destination port.
  - The MAC address table maps MAC addresses to the switch port numbers.
  - It is dynamically built by learning MAC addresses from the frames passing through the switch.
- **Sends** the frame to the correct port to reach the destination device.

4. Computer Networks

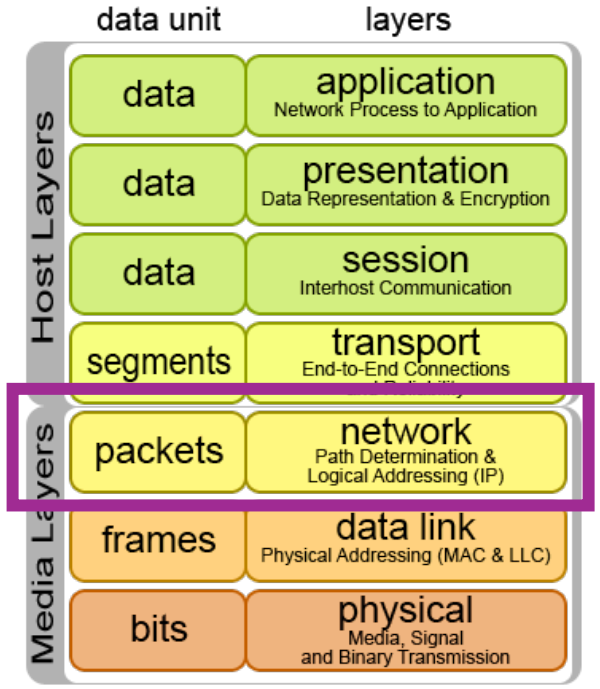
Network, Transport, Session, Presentation, Application Layers

Summary

- Network Layer
- Packet Header Structure
- Routing Process
- Logical Addressing – IP Address
- IPv4 Address Structure
- Subnet Mask Structure
- Transport Layer
- Sockets
- Segment Header Structure
- Session, Presentation, Application Layers

Network Layer

The **Network Layer** is the **third** layer in the **ISO/OSI model**. It determines how data is sent to the receiving devices across multiple networks. Specifically, it is responsible for **routing**, **forwarding**, and **addressing** data packets across different networks.



Functions of the Network Layer

- **Routing:**

Determines the optimal path for data to travel from source to destination.

- **Forwarding:**

Moves packets from the router's input to the appropriate output.

- **Logical Addressing:**

Uses IP addresses to identify devices on a network.

- **Fragmentation and Reassembly:**

Breaks down large packets into smaller fragments and reassembles them at the destination.

Packet Header Structure

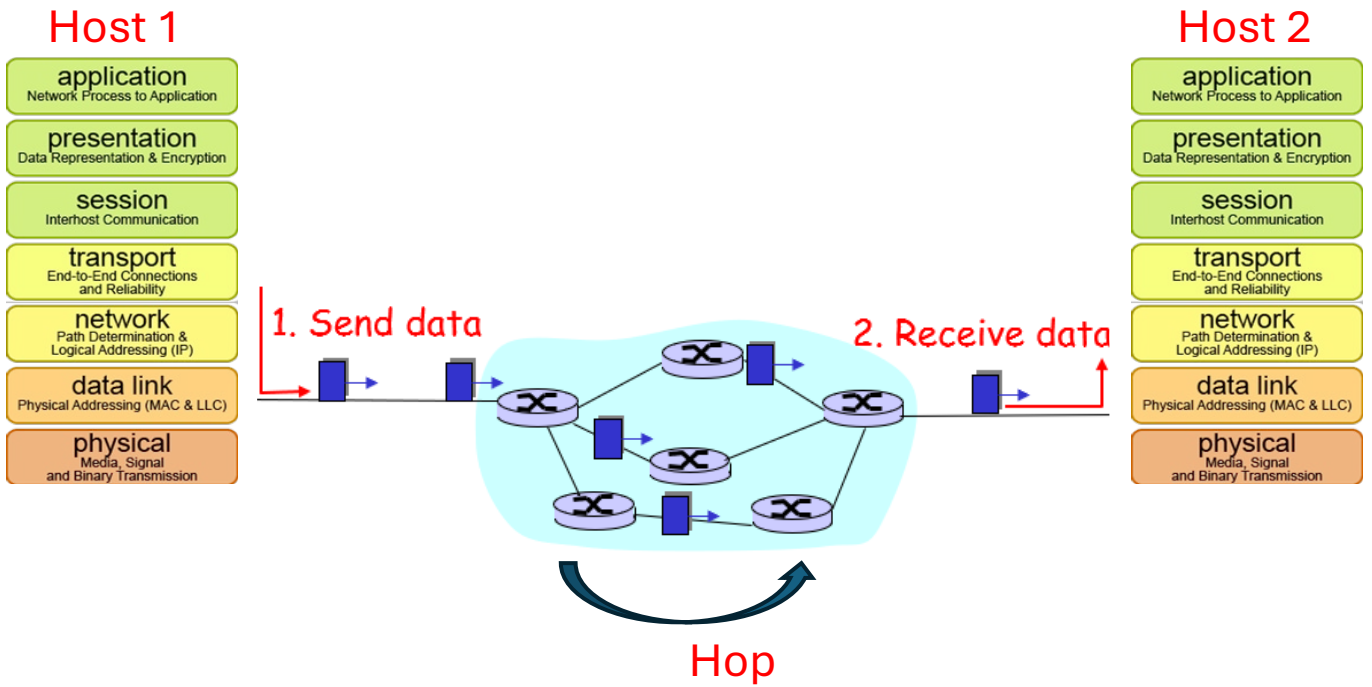
IPv4 Packet Header Structure

Field	Length (bits)	Description
Version	4	IP version (4 for IPv4)
IHL	4	Internet Header Length (in 32-bit words)
Type of Service (ToS)	8	Quality of Service indicators and priority
Total Length	16	Total length of the packet (header + data)
Identification	16	Unique identifier for fragments of the original packet
Flags	3	Control flags for fragmentation
Fragment Offset	13	Position of this fragment in the original packet
Time to Live (TTL)	8	Maximum number of hops the packet can take
Protocol	8	Protocol used in the data portion (e.g., TCP, UDP)
Header Checksum	16	Error-checking of the header
Source IP Address	32	IP address of the sender
Destination IP Address	32	IP address of the receiver
Options	Variable (0-40 bytes)	Optional fields for additional functionality (e.g., security)
Padding	Variable	Extra bytes to ensure the header is a multiple of 32 bits

Routing Process/1

The **routing process** comprises these **3 steps**:

1. **Path Determination:** Routers use routing tables and algorithms to determine the best path for data packets.
2. **Packet Switching:** Data packets are forwarded from one router to another based on the routing table.
3. **Next-Hop Forwarding:** Each router forwards the packet to the next router until it reaches the destination.



Path Determination

Path determination is the process by which routers decide the **best route** for data packets **to travel** from the source to the destination.

- **Routing Tables:** Routers maintain routing tables that contain information about **network topology** and **available routes**.
  - **Static Routing:** Routes are manually configured by network administrators.
  - **Dynamic Routing:** Routes are automatically learned and updated using routing protocols.

- **Routing Algorithms:** Algorithms such as Dijkstra's **Shortest Path First (SPF)** and **Distance Vector** are used to calculate the optimal path.
  - **Shortest Path First (SPF):** Calculates the shortest path to a destination based on cumulative cost metrics.
  - **Distance Vector:** Determines the best path based on distance metrics and updates from neighboring routers.

Packet Switching

Packet switching is the process of **moving data packets** from the **input port** of a router to the appropriate **output port**, based on routing decisions.

- **Switching Fabric:** The internal architecture of a router that connects input ports to output ports.
  - **Store-and-Forward:** Entire packet is received before it is forwarded to the next hop.
  - **Cut-Through:** Packet is forwarded as soon as the destination address is read.

- **Buffering:** Temporary storage of packets in memory if the output port is busy, preventing packet loss.

- **Forwarding Decision:** Based on the destination IP address and the routing table, the router decides the next hop for the packet.

Store-and-Forward Switching

The **entire packet** is received by the router before it is **forwarded** to the next hop.

Process:

1: The router receives the entire data packet.

2: The packet is stored temporarily in memory.

3: Error checking (such as CRC) is performed to ensure data integrity.

4: The packet is forwarded to the next hop based on the destination address.

Advantages: Ensures error-free transmission by checking the entire packet for errors before forwarding. Suitable for networks where data integrity is crucial.

Disadvantages: Higher latency due to the time taken to receive and process the entire packet. Requires more memory to store the packets.

Cut-Through Switching

The packet is **forwarded** as soon as the destination address is read, **without waiting** for the **entire packet** to be received.

Process:

1: The router begins forwarding the packet as soon as it reads the destination MAC address from the packet header.

2: The rest of the packet continues to be forwarded as it is received.

Advantages: Lower latency because forwarding begins almost immediately after the destination address is read. Faster data transfer suitable for high-performance networks.

Disadvantages: Does not check for errors in the packet, which means corrupted packets might be forwarded. Can lead to potential issues if the packet is corrupted during transmission.

Next-Hop Forwarding

Next-hop forwarding refers to the process of **sending** a data packet to the **next router** (or final destination) along the path determined by the routing algorithm.

- **Hop-by-Hop Routing**

Each router along the path makes an independent forwarding decision.

- **Next-Hop Address**

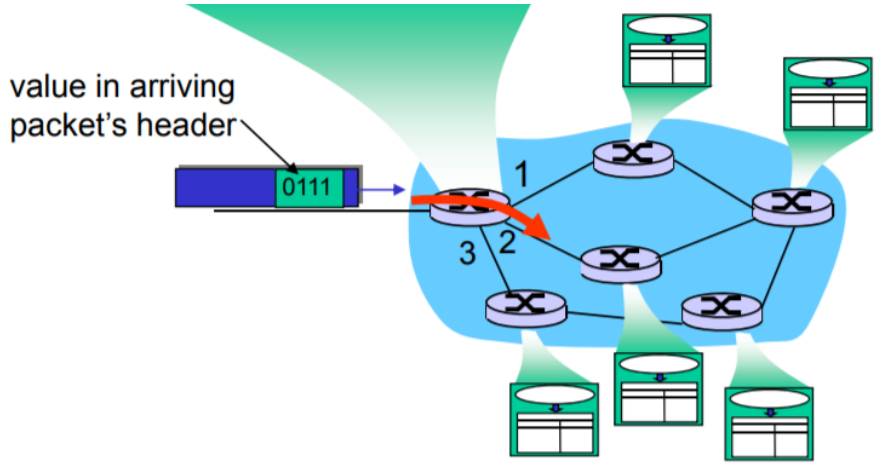
The IP address of the next router to which the packet should be sent.

- **Address Resolution Protocol (ARP)**

Translates IP addresses to MAC addresses for forwarding packets at the Data Link Layer

Routing Process/2

Destinazione IP	Maschera di Sottorete	Next Hop (Prossimo Router)	Interfaccia di Uscita
192.168.1.0	255.255.255.0	192.168.2.1	GigabitEthernet0/1
10.0.0.0	255.0.0.0	10.1.1.1	GigabitEthernet0/2
172.16.0.0	255.255.0.0	172.16.1.1	GigabitEthernet0/3
0.0.0.0	0.0.0.0	192.168.3.1	GigabitEthernet0/4 (Default Route)



Logical Addressing – IP Address

Each device on a network has a unique **IP address** used for **identification** and **communication**.

IPv4 Addresses: 32-bit addresses, e.g., 192.168.1.1

IPv6 Addresses: 128-bit addresses, e.g., 2001:0db8:85a3:0000:0000:8a2e:0370:7334

IPv4 Structure

Format: Dotted decimal notation (e.g., **192.168.1.1**)

Binary Representation: 32 bits divided into four 8-bit octets.

Example: 192.168.1.1 in binary is 11000000.10101000.00000001.00000001.

Classes (to define *network size* and *purpose*):

Class A: 0.0.0.0 to 127.255.255.255

Class B: 128.0.0.0 to 191.255.255.255

Class C: 192.0.0.0 to 223.255.255.255

Class D: 224.0.0.0 to 239.255.255.255 (Multicast)

Class E: 240.0.0.0 to 255.255.255.255 (Reserved)

Subnet Mask Structure

The **Subnet Mask** is a 32-bit number that **divides** the IP address into **network** and **host** portions.

- **Function:**

Determines which part of the IP address is the network address and which part is the host address.

Format: Dotted decimal notation (e.g., 255.255.255.0).

Binary Representation: Corresponds to the IP address, using 1s for the network part and 0s for the host part.

Example: 255.255.255.0 in binary is 11111111.11111111.11111111.00000000.

Combining IP Address and Subnet Mask

Network Address: The part of the IP address identified by the subnet mask's 1s.

Host Address: The part of the IP address identified by the subnet mask's 0s.

Example:

- **IP Address:** 192.168.1.10
- **Subnet Mask:** 255.255.255.0
- **Network Address:** 192.168.1.0
- **Host Address:** 10

IPv4 Class A and Class B

Class A

Range: 0.0.0.0 to 127.255.255.255

First Octet Range: 0 to 127

Default Subnet Mask: 255.0.0.0

Number of Networks: 128 (2^7)

Hosts per Network: Over 16 million (2^{24} - 2)

Usage: Designed for very large networks with many devices, such as large corporations or ISPs.

Example: 10.0.0.1

Class B

Range: 128.0.0.0 to 191.255.255.255

First Octet Range: 128 to 191

Default Subnet Mask: 255.255.0.0

Number of Networks: 16.384 (2^{14})

Hosts per Network: Over 65k (2^{16} - 2)

Usage: Suitable for medium-sized networks, such as universities or large companies.

Example: 172.16.0.1

IPv4 Class C, Class D, and Class E

Class C

Range: 192.0.0.0 to 223.255.255.255

First Octet Range: 192 to 223

Default Subnet Mask: 255.255.255.0

Number of Networks: Over 2 million (2^{21})

Hosts per Network: 254 (2^8 - 2)

Usage: Ideal for small networks, such as small businesses or home networks.

Example: 192.168.1.1

Class D

Range: 224.0.0.0 to 239.255.255.255

First Octet Range: 224 to 239

Default Subnet Mask: 255.255.255.255

Usage: Reserved for multicast groups. Allows a single packet to be sent to multiple destinations.

Example: 224.0.0.1

Class E

Range: 240.0.0.0 to 255.255.255.255

First Octet Range: 240 to 255

Usage: Reserved for experimental purposes and future use. Not used for general networking.

Private and Public IP Addresses

Private IP Addresses

IP addresses used within a private network. Not routable on the internet.

Example:

Home Network: 192.168.1.1

Office Network: 10.0.0.1

Public IP Addresses

IP addresses that are routable on the internet. Assigned by ISPs and regulated by regional internet registries.

Example:

Website IP: 93.184.216.34

Corporate Server: 203.0.113.10

Network Address Translation (NAT)

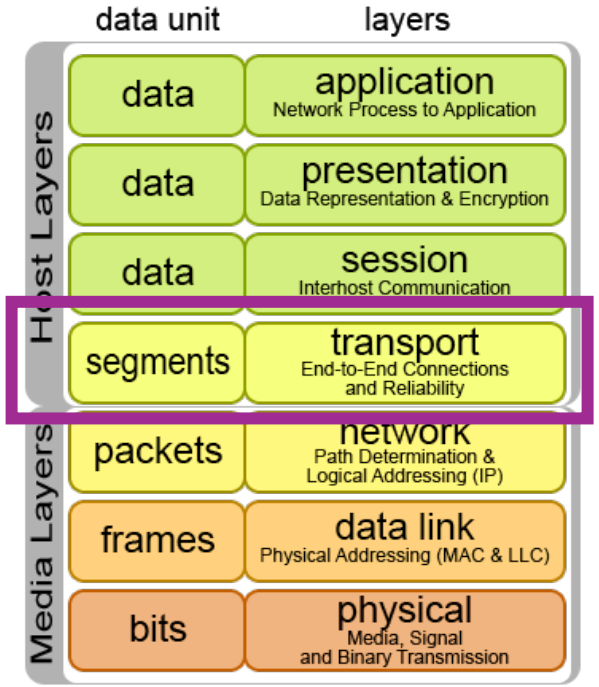
Purpose: Allows multiple private IP addresses to share a single public IP address for internet access.

Function: Translates private IP addresses to a public IP address and vice versa.

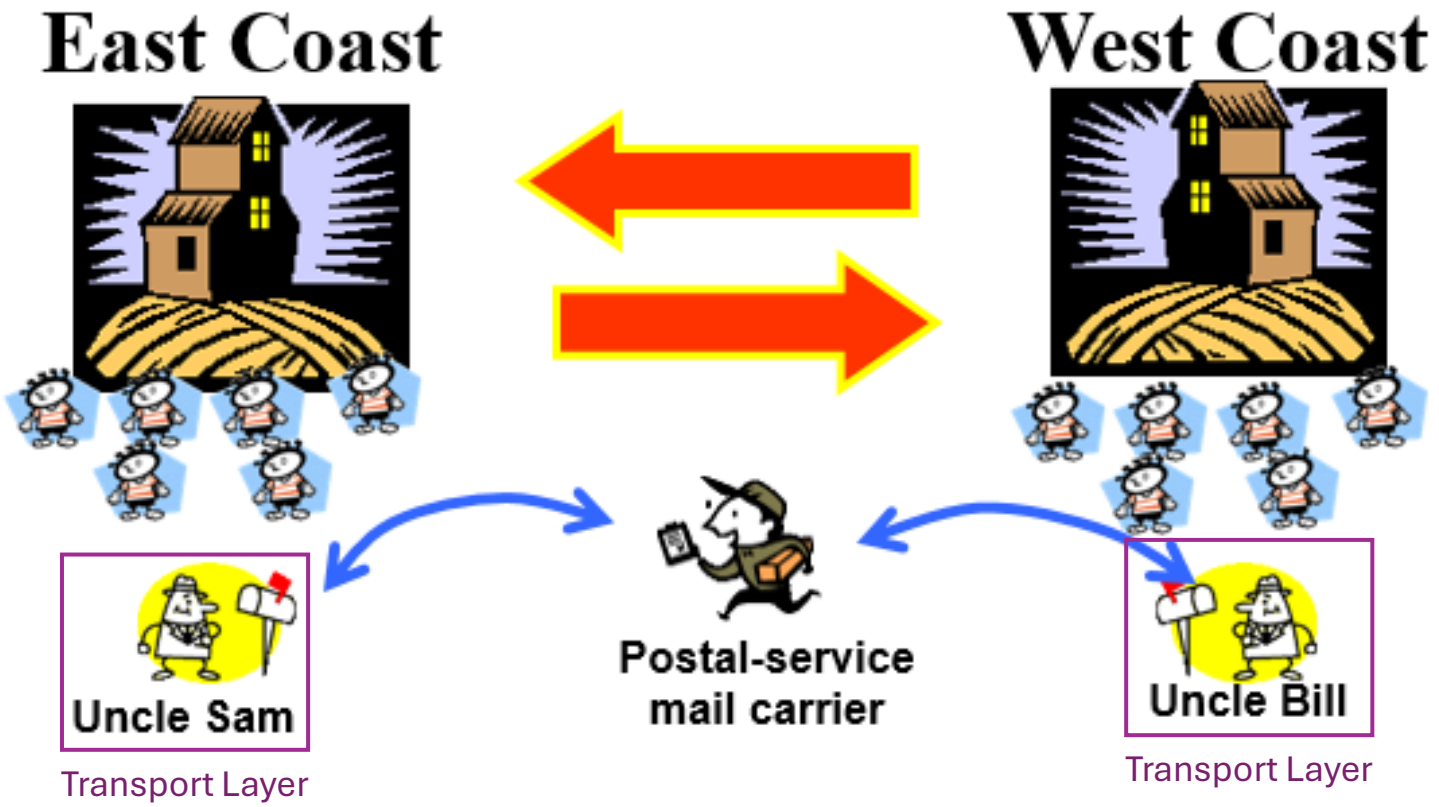
Example: A home router uses NAT to enable all devices in a home network (with private IP addresses) to access the internet using the router's public IP address.

Transport Layer

The **Transport Layer** is crucial for **end-to-end communication** between devices on a network.



Transport Layer - Analogy



Functions of the Transport Layer

- **End-to-End Communication:**

Manages data transfer between devices.

- **Segmentation and Reassembly:**

Splits large data streams into smaller segments and reassembles them at the destination.

- **Error Detection and Correction:**

Ensures data integrity and reliability.

- **Flow Control:**

Manages the rate of data transmission between devices.

Connection Management:

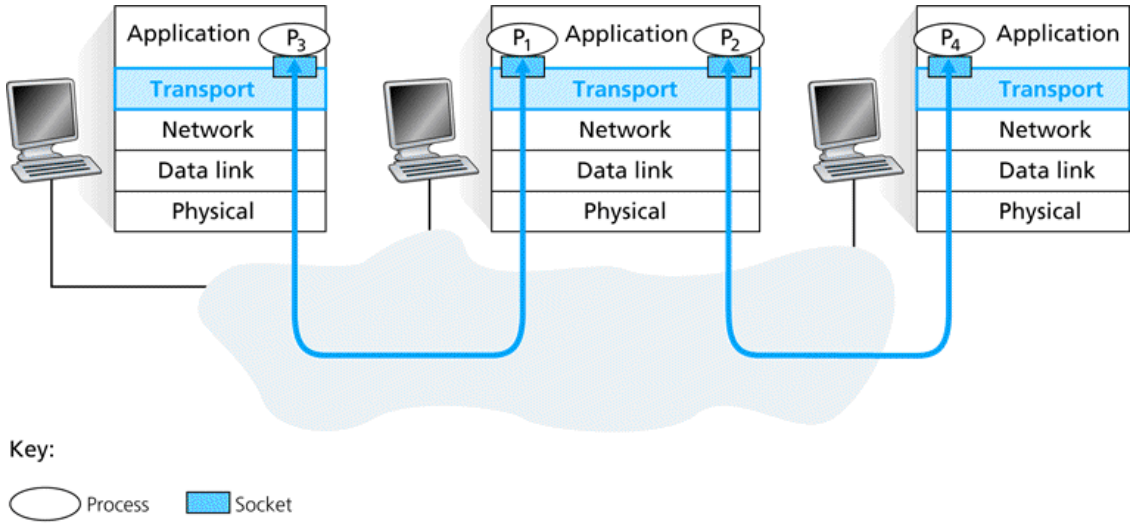
- Establishes, maintains, and terminates connections.

Sockets

Sockets are fundamental for enabling communication between devices over a network. They act as **interfaces** through which **processes (applications)** communicate across a computer network.

Combining **IP addresses** and **port numbers** uniquely identify a network connection (network, host, and applications).

Example: 192.168.1.1:80

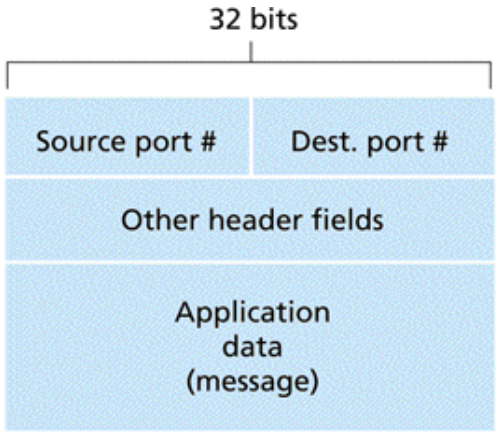


Segment Header Structure

Source Port (16 bits): Port number of the sending application

Destination Port (16 bits): Port number of the receiving application

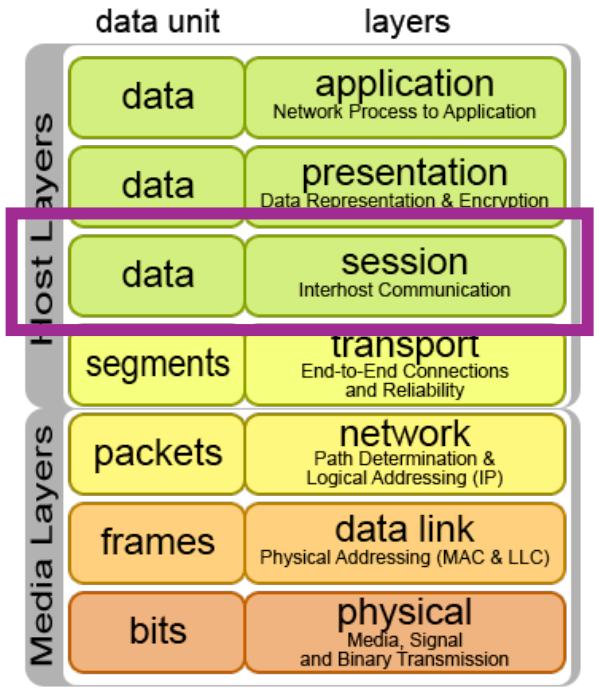
Other header fields: Fields depending on TCP or UDP protocols



Session Layer

The **Session Layer** manages and controls the connections between computers.

- Creates, maintains, and terminates **sessions** between **applications**.
- Manages dialogue (communication) between two devices, allowing them to communicate in either **half-duplex** or **full-duplex** mode.

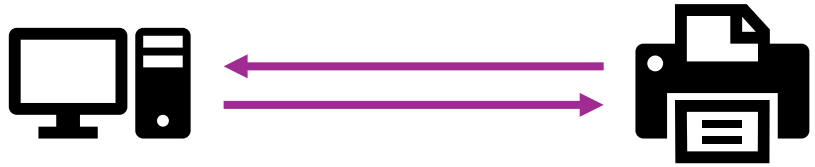


Session Layer – Dialog Control

Half-Duplex Mode:

Communication can occur in both directions, but **not simultaneously**.

Example: A network printer and a computer communicate in half-duplex mode, where the computer sends a print job, and the printer sends an acknowledgment back, but they do not send data at the same time.



Full-Duplex Mode:

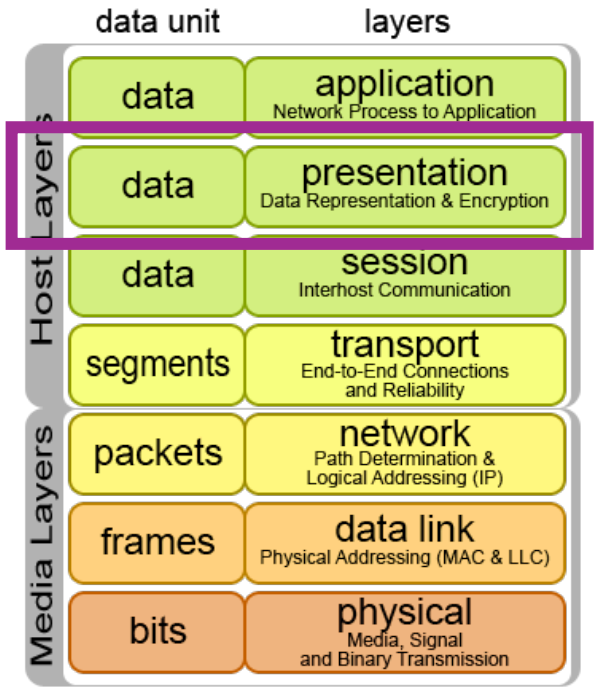
Communication can occur **simultaneously** in both directions.

Example: A video conferencing application where both participants can speak and listen at the same time, ensuring smooth two-way communication.

Presentation Layer

The **Presentation Layer** translates data between the application layer and the network.

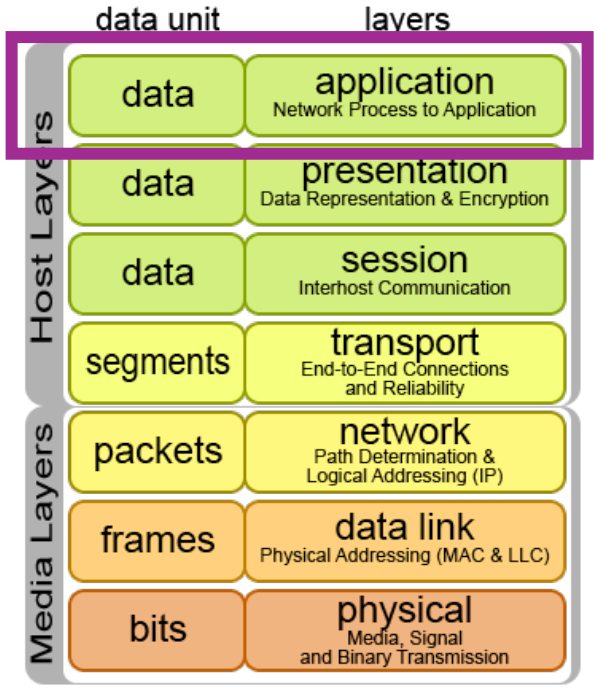
- **Translation:** Converts data formats from application-specific formats to network formats, and vice versa.
- **Encryption/Decryption:** Ensures data security by encrypting data before transmission and decrypting it upon reception.
- **Compression:** Reduces the size of the data to be transmitted to optimize network resource usage.



Application Layer

The **Application Layer** provides network services directly to user applications.

- **Functions:** Network Services: Enables user applications to interact with the network (e.g., file transfers, email, remote login).
- **Resource Sharing:** Facilitates access to network resources.
- **User Interface:** Provides an interface for the user to interact with the network.



4. Computer Architectures

Services and Functions of the Operating System

Summary

- Process Management
- Main Memory Management
- File System Management
- Peripherical Device Mangement
- Secondary Memory Management
- Memory Hierarchy
- Protection and Security Management
- HCIs and Application Management

Process Management/1

A **process** refers to an **instance** of a **running application** or **program**. On a computer or device with a **multi-tasking Operating System (OS)**, numerous processes can run **simultaneously**, whether they belong to different applications or the same one. The OS manages these processes through various activities, including:

- **Creating** and **terminating** processes
- **Suspending** and **resuming** process execution
- **Synchronizing** and **facilitating** communication among processes
- **Managing** deadlocks

Processes can make **system calls** to access **OS services** for their own management, which include:

- **Executing** other processes (exec)
- **Replicating** a running process (fork)
- **Sending signals** between processes (wait/signal)
- **Terminating** a process (kill/terminate)

Process Management/2

Using the **fork** system call, the OS can **create additional processes** based on its configuration, starting with the system's initial boot process.

This results in a **parent/child** tree hierarchy among the running processes on the computer. The **root process**, often referred to as **systemd** in modern Linux distributions or **init** in other UNIX-like systems, is responsible for system initialization and is the **ancestor of all other processes**.

Each process is identified by a unique **Process ID (PID)**, a **Parent Process ID (PPID)**, which references the PID of its parent process, and a **User ID (USERID)**, which indicates the user that is running the process.

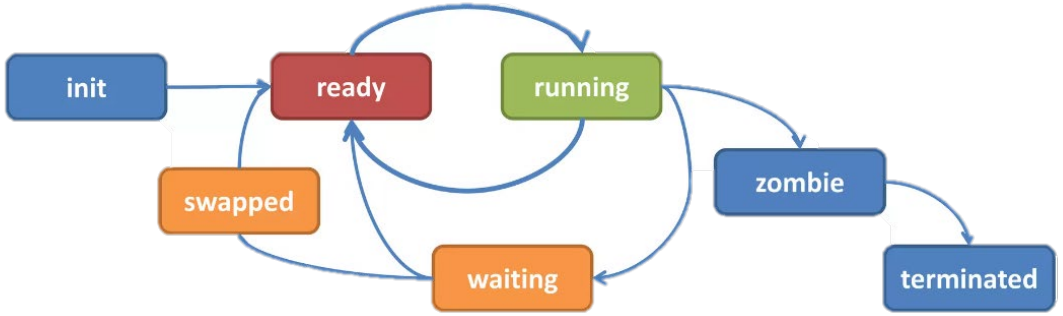
```
tecmint@ubuntu:~$
tecmint@ubuntu:~$ pstree -p
systemd(1)─┬─ModemManager(1156)─┬─{ModemManager}(1197)
           │                    └─{ModemManager}(1200)
           ├─NetworkManager(990)─┬─{NetworkManager}(1072)
           │                     └─{NetworkManager}(1124)
           ├─VGAuthService(916)
           ├─accounts-daemon(979)─┬─{accounts-daemon}(1033)
           │                      └─{accounts-daemon}(1121)
           ├─acpid(980)
           ├─apache2(1382)─┬─apache2(1383)─┬─{apache2}(1393)
           │               │               ├─{apache2}(1395)
           │               │               ├─{apache2}(1397)
           │               │               ├─{apache2}(1398)
           │               │               └─{apache2}(1400)
```

```
tecmint@ubuntu:~$
tecmint@ubuntu:~$ ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 Jan20 ?        00:00:04 /sbin/init auto noprompt spl
root           2       0  0 Jan20 ?        00:00:00 [kthreadd]
root           3       2  0 Jan20 ?        00:00:00 [rcu_gp]
root           4       2  0 Jan20 ?        00:00:00 [rcu_par_gp]
root           5       2  0 Jan20 ?        00:00:00 [slub_flushwq]
root           6       2  0 Jan20 ?        00:00:00 [netns]
root           8       2  0 Jan20 ?        00:00:00 [kworker/0:0H-events_highpri
root          10       2  0 Jan20 ?        00:00:00 [mm_percpu_wq]
root          11       2  0 Jan20 ?        00:00:00 [rcu_tasks_rude_]
root          12       2  0 Jan20 ?        00:00:00 [rcu_tasks_trace]
root          13       2  0 Jan20 ?        00:00:00 [ksoftirqd/0]
root          14       2  0 Jan20 ?        00:00:13 [rcu_sched]
root          15       2  0 Jan20 ?        00:00:00 [migration/0]
```

Process Life Cycle

Each **process**, during its **life cycle**, can be in one of the following states:

1. **Init**: The initial load state of the process in memory. The program is set in an '**execution state**' inside the computer, the main process is created, and the required memory (RAM) is allocated.
2. **Ready**: The process is loaded into memory (RAM) in a '**ready to run**' state, waiting for the CPU allocation.
3. **Running**: The process **is being executed** by the CPU.
4. **Waiting**: The process is **suspended**, awaiting an event (e.g., feedback from a device).
5. **Swapped**: The process, while awaiting an event, has been placed inside the **virtual memory** (i.e., hard drive) and is pending recovery to primary memory to be executed.
6. **Zombie**: The process **has completed its execution** but remains in memory (i.e., it has a PID), waiting for its parent process to release it permanently.
7. **Terminated**: The process is **terminated**, and the OS deallocates the previously assigned memory (RAM).



Main Memory Management

Main memory includes the **CPU registers**, the **CPU cache**, and the **Random Access Memory (RAM)**.

The OS has several responsibilities for managing the main memory:

- **Allocating and deallocating memory:** The OS assigns memory to processes as needed and releases it when it is no longer required.
- **Isolating memory segments:** In multi-tasking systems, the OS ensures that different processes have separate memory spaces to prevent conflicts and unauthorized access.
- **Address mapping:** The OS manages the mapping between logical memory addresses (used by processes) and physical memory addresses (actual locations in RAM).
- **Managing paging and virtual memory:** The OS uses paging to move portions of a program between primary memory (RAM) and secondary storage (swap space) to optimize the use of available memory and extend the effective amount of memory.

Address Mapping

Isolation: Each process (like an application or a background service) running on a computer is given its own **virtual address space**. This space is essentially a set of memory addresses that the process can use to access memory. The key point is that this space is **private** to the process—no other process can see or access this space directly. This isolation protects processes from interfering with each other, which enhances **security** and **stability**.

Consistency: To the process itself, this virtual address space appears as a **continuous** and **consistent range of addresses** starting from zero upwards. It doesn't matter how the physical memory (RAM) is structured or how much physical memory is actually available.

Mapping: When a process requests access to a memory address within its virtual address space, it doesn't access the physical memory directly. Instead, the address it uses is a virtual address. The operating system, with the help of the **memory management unit (MMU)**, translates this virtual address into a physical memory address where the data is actually stored in RAM.

The **system calls** that the processes can invoke to obtain the **memory management services**, from the operating system, are the following:

- **malloc**, **calloc**, **realloc**, for the dynamic allocation of memory blocks;
- **free**, for the deallocation of the previously allocated memory blocks.

Address Mapping - Advantages

- **Security:** By isolating each process's memory, the system prevents one process from accidentally or maliciously interfering with another process's data.
- **Flexibility:** Processes can be given more address space than the actual physical memory available on the system. For example, a process might be allowed to use 4 GB of addresses even if the computer only has 2 GB of physical RAM.
- **Efficiency:** The operating system can manage memory more effectively. It can allocate physical memory where needed and use techniques like paging (storing parts of the virtual memory on disk when RAM is full) to optimize the use of available physical memory.

Virtual Memory and Paging

Virtual memory is a system where the **OS** uses both **physical RAM** and a **portion of the hard disk** called the **swap space** (or **paging file**) to simulate a much larger pool of memory. This system allows each process to have access to a private virtual address space, which is mapped to physical memory and disk as needed.

- **Paging:** The virtual memory is divided into blocks called pages. The corresponding blocks in physical memory are called **page frames**. Not all pages are loaded in physical memory at all times; some are kept on the disk in the swap space.

- **Page Table:** The OS maintains a page table for each process, which maps virtual pages to physical page frames. If a page is not in physical memory (a condition known as a "page fault"), the OS fetches it from the disk to a free page frame in physical memory, updating the table to reflect this.

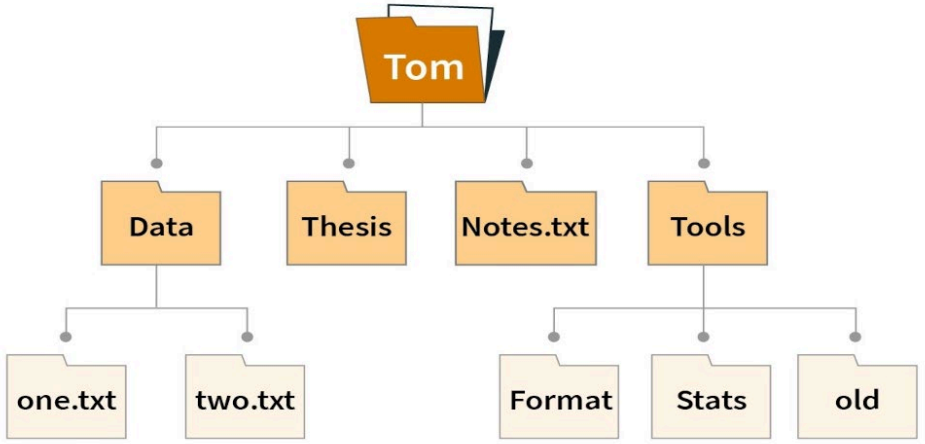
- **Swap In/Swap Out:** When RAM fills up, the OS chooses less frequently used pages and moves them out to disk (swapping out) to make room for active pages that need to be loaded into RAM (swapping in).

File System Management/1

The **file system** is an abstraction layer that the Operating System uses to **manage data** on **secondary storage** (e.g., hard drives and SSDs). This model is independent of the type and number of secondary storage devices.

The **basic element** of the file system is the **file**, which is a sequence of bytes stored on secondary storage. Unlike in-memory data structures, files persist beyond the process that created them and are often terminated with an **EOF (end of file)** marker, though this is more of a **logical concept** rather than a physical symbol stored in the file.

The **file system** provides an abstract framework for **organizing files** on secondary storage, typically in the form of a **hierarchical directory tree**, which includes **directories** and **subdirectories**.



File System Management/2

The concept of a **directory** (or **folder**) is also an abstraction. Precisely, a directory is a file that contains **pointers to other files**, establishing a **parent-child** relationship within the file system.

File names, the set of characters allowed in file names, and the **meta-characters** used to indicate **file placement** within the file system are aspects defined by each specific file system model.

Examples:

"**C:\Tom\Data\one.txt**": This is an absolute path that uniquely identifies a file located on the *C* drive in Microsoft Windows.

NOTE: that it is not case-sensitive and specifies the identifier of the physical drive where the file is stored.

"**~Tom/src/minimumSpanningTree.c**": This path is typical in UNIX-like systems.

Note: It is case sensitive, uses the '***~username***' convention to identify a user's home directory, and the path is independent of the physical location of the file on a specific device.

File System Management/3

Processes can invoke **system calls** to interact with the file system for various operations, including:

- **Creating** and **deleting** files
- **Opening** and **closing** files (**fopen** / **fclose**)
- **Reading** and **writing** files (**fget**/ **fread** / **fwrite**/ etc.)
- **Setting** file attributes (such as read-only, writable, executable)

The file system implements **protection mechanisms** to restrict access to files, ensuring only **authorized users** can interact with them. Additionally, it manages a **queue** of file access requests from processes.

In a **multi-user Operating System**, the file system tracks the ID of the **file owner** and defines **access rules** for other system users, ensuring appropriate access controls are enforced.

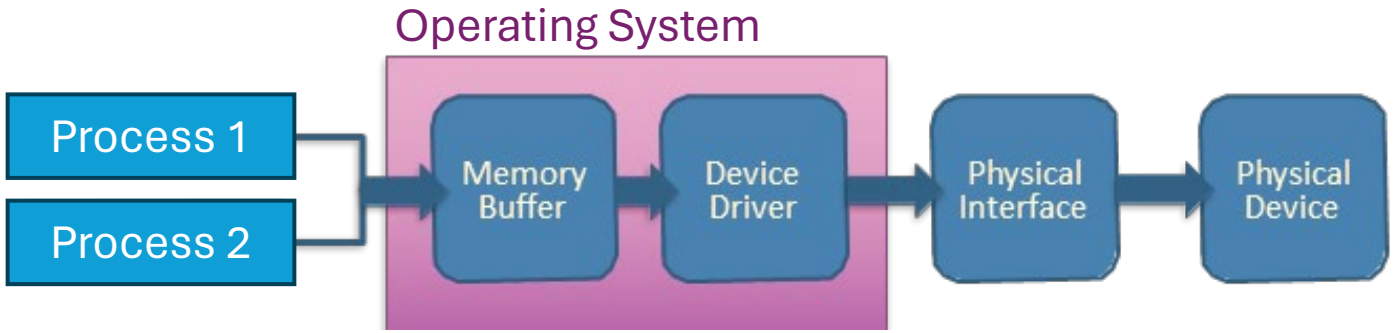
Peripherical Device Management

The **OS** manages the **communication** with **peripheral units** and provides an **abstraction** (i.e., functions) that allows programs to utilize the **communication channel**.

Since **multiple programs** may **simultaneously** need **access** to a particular device (e.g., terminal output, keyboard input, or printer), the OS manages a **queue of requests** (**serialization**) to **avoid conflicts**.

The OS ensures efficient communication with specific peripheral devices. It uses a special type of memory called **buffer memory** to support the communication process. This buffer memory acts like a **parking area** where processes data directed to or coming from the device can be held until they are processed.

Interaction with peripheral devices is facilitated through a specific software module known as a device **driver**.



Secondary Memory Management/1

Secondary memory consists of **persistent storage devices** that can retain recorded information even when the machine is turned off.

Due to physical and technological reasons, secondary memory has a much **higher access time** and **data transfer rate** compared to primary memory (which lacks mechanical components).

Typical secondary storage devices (or mass storage) include **magnetic hard drives**, **optical disks**, **solid-state drives** (**SSDs**), **memory cards**, and more.

Secondary Memory Management/2

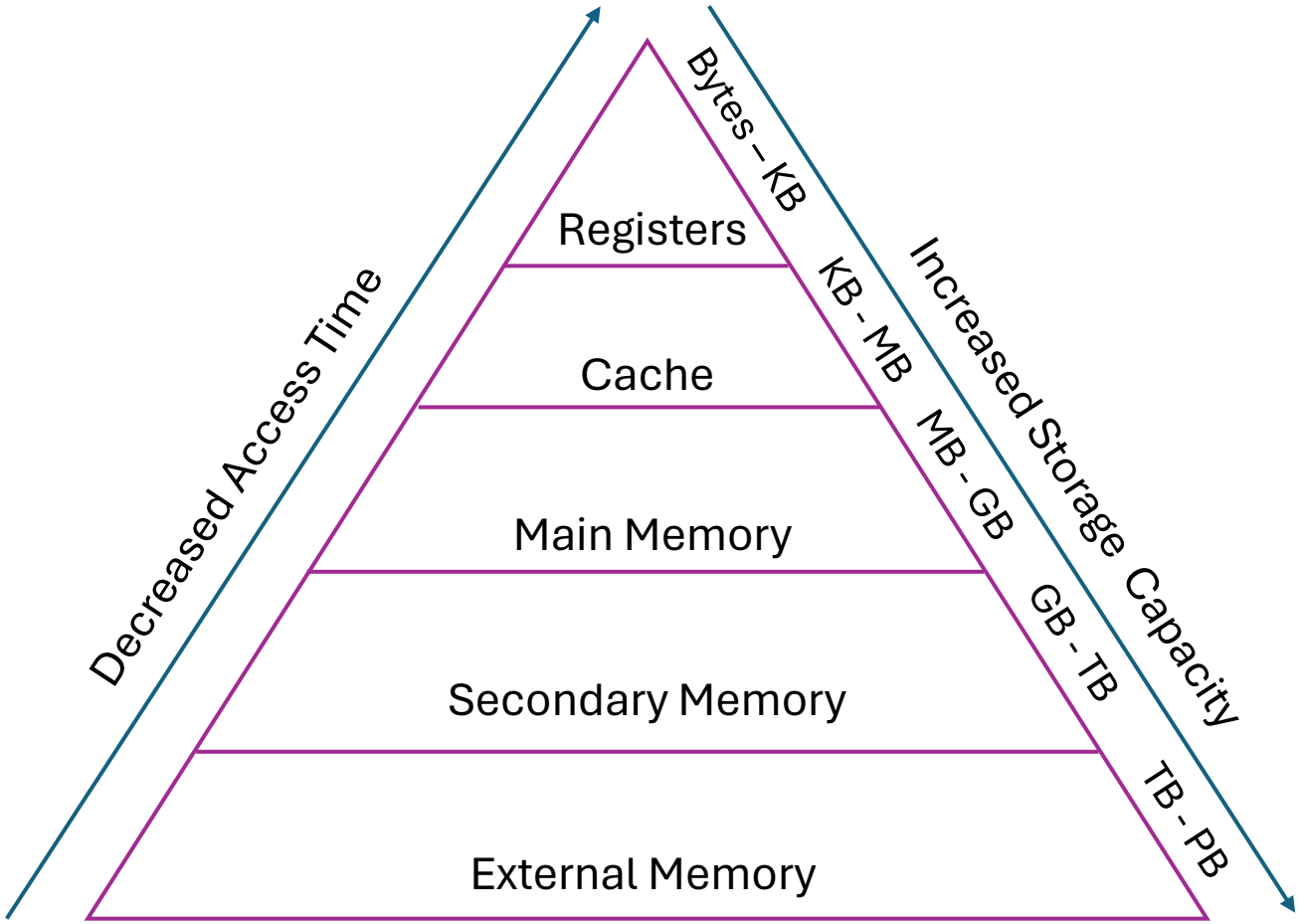
Typical **operations** performed by the **OS** on these devices include:

- **Allocating** and **deallocating** space for storing data (files)
- **Managing** free space within the mass memory unit
- **Optimizing**, **serializing**, and **scheduling** operations within the mass memory unit

File system management and secondary memory management are two distinct but closely related functions of the operating system.

The OS component that manages secondary memory makes the physical structure of the storage device transparent to programs, allowing the same system calls to be used for files stored on different devices.

Memory Hierarchy



Protection and Security Management

In a **multi-tasking** and **multi-user** system, the **OS** manages **resource protection** to ensure the **privacy of resources** and **users**.

The protection process is based on the following elements:

- **Authentication**: A procedure to verify the user's identity.
- **Authorization**: A procedure to determine a user's or process's rights to access a resource.

Authorization

Linking Processes to User Permissions:

- Each process in the system is associated with a specific user.
- Processes inherit the permissions of their associated user for accessing system resources.

High-Level Authorization for OS Processes:

- OS processes are executed by users with the highest level of authorization, such root or administrator.

Resource Security Policies:

- Resource security policies are based on rules that map resource access permissions to system users. For example, file access permissions dictate which users can read, write, or execute specific files.

Grouping Users for Simplified Authorization:

To simplify authorization mapping, the OS allows users to be grouped into clusters or groups. Each member within a group inherits the group's authorizations.

Authentication

A **multi-user OS** implements **login procedures** for system access.

The **login procedure** is crucial for two main reasons:

- It **authenticates** the user based on their credentials (i.e., username and password).
- It **verifies** whether the user is **authorized** to access the system.

The **login process** relies on a repository of **user credentials** and the definition of **user groups**, such as:

- The files **/etc/passwd** , **/etc/shadow ,** and **/etc/group/** in UNIX-like systems.
- Other external systems of authentication and authorization.

NOTE:

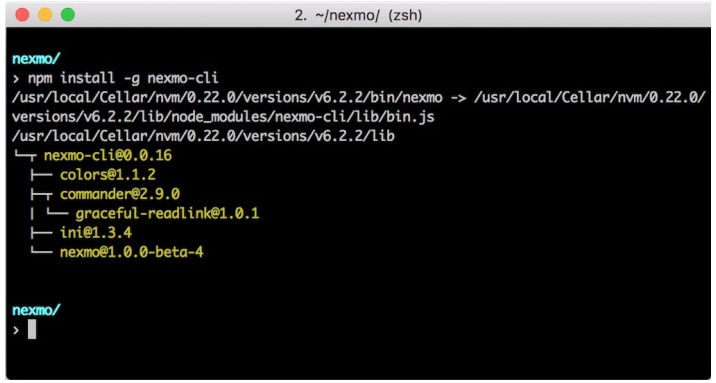
Username and password based authentication is the most common, although not the only or safest method.

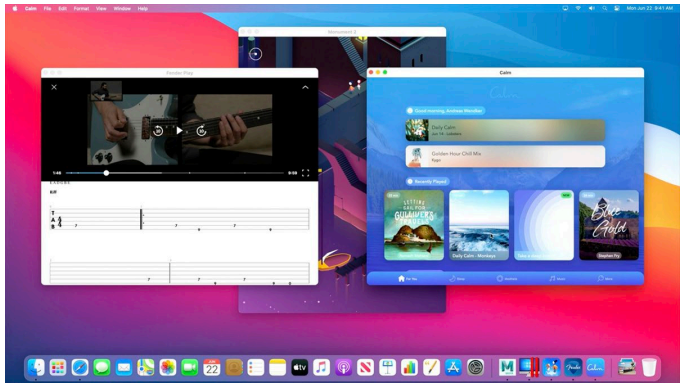
HCIs and Applications Management

The **OS** provides a set of features to enable applications to create **user interaction** tools.

In general, a **User Interface (UI)** can be:

- **Alphanumeric Interface**: The operating system offers an abstract terminal model for presenting information on a screen that displays alphabetic and numeric characters. It also captures input through a keyboard.
- **Graphical User Interface (GUI)**: The operating system provides features for programs to build a GUI using elements such as windows and icons. Input can be done using a mouse or touch-screen, in addition to the keyboard.





ICTAcademy
INFORMATION AND COMMUNICATIONS TECHNOLOGY

5. Computer Networks

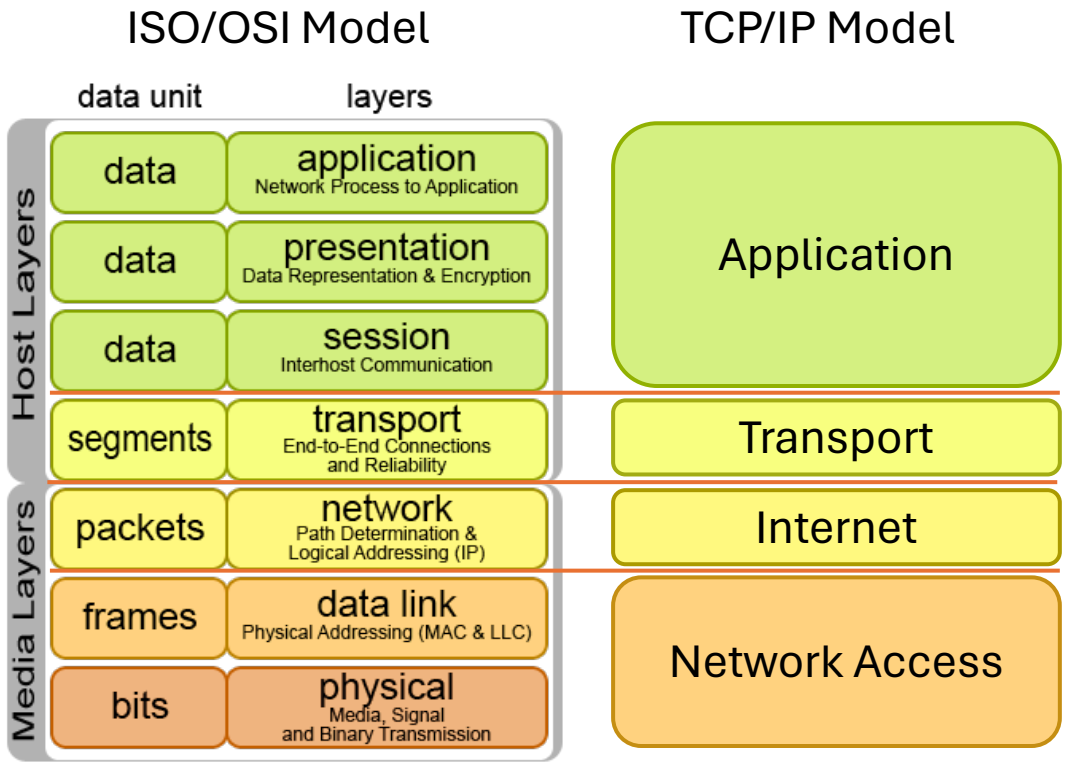
TCP/IP Model

Summary

- ISO/OSI Model vs TCP/IP Model
- Application Layer
- DNS - HTTP - FTP – SMTP/POP3/IMAP - DHCP
- Transport Layer
- TCP - UDP
- Internet Layer
- ICMP - ARP
- Network Access Layer

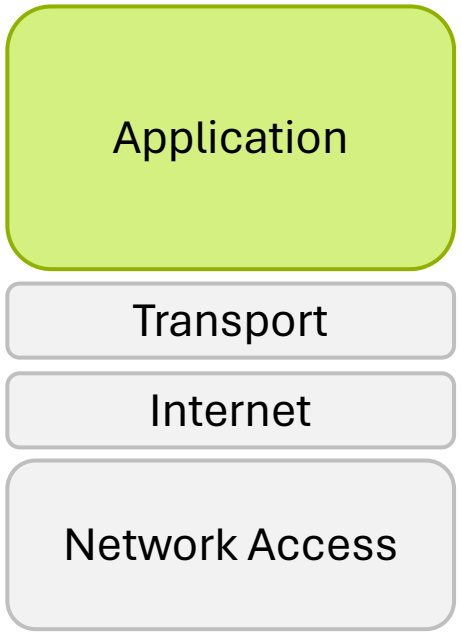
ISO/OSI Model vs TCP/IP Model

The **TCP/IP model** is a concise framework used to standardize and ensure reliable data communication across diverse interconnected networks, including the **internet**. It consists of **four layers**, each responsible for specific functions in the process of data transmission. The TCP/IP model is the **standard de facto** for internet and network communication, widely adopted due to its robustness and flexibility.



ICTAcademy
INFORMATION AND COMMUNICATIONS TECHNOLOGY

Application Layer



Functions:

Provides network services directly to user applications.

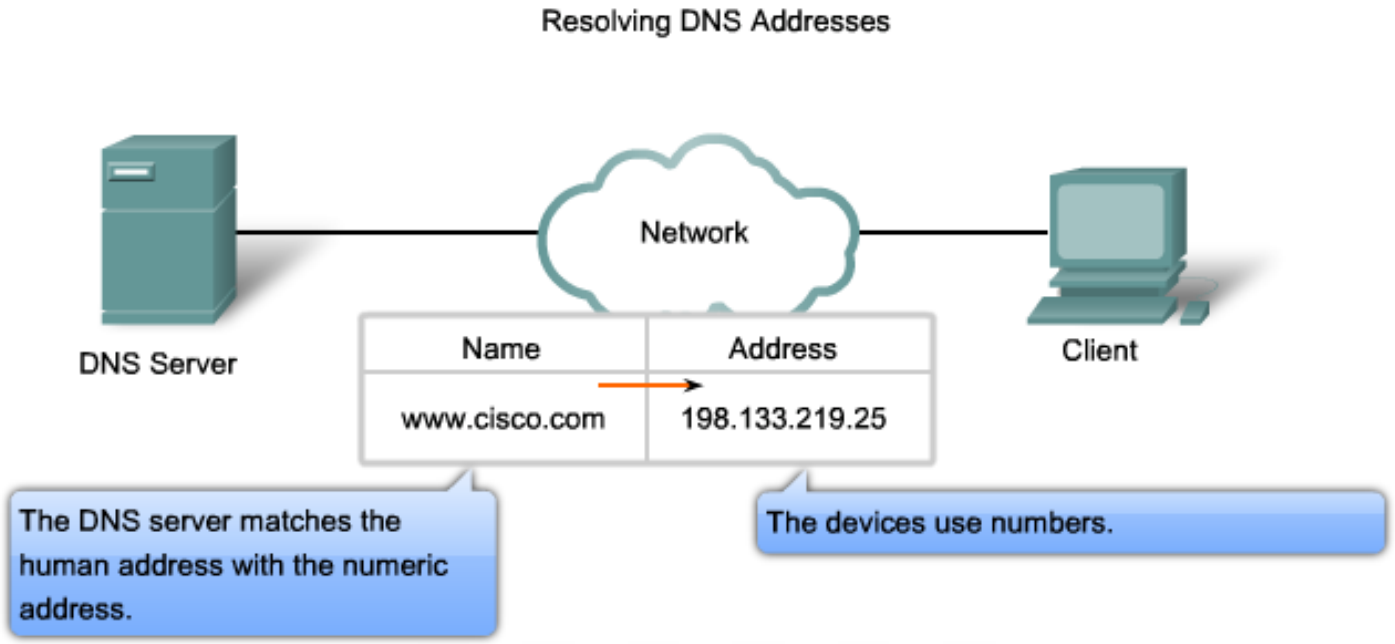
Manages application-level protocols.

Protocols:

- **DNS:** Domain name resolution
- **HTTP/HTTPS:** Web browsing
- **FTP:** File transfer
- **SMTP/POP3/IMAP:** Email
- **DHCP:** Management of IP addresses allocation

Domain Name System (DNS)

The **Domain Name System** (**DNS**) is a hierarchical and decentralized naming system used to **resolve** human-readable **domain names** (like www.cisco.com) into machine-readable **IP addresses** (like 198.133.219.25).



Domain Name System (DNS) – How it works

DNS is a critical component of the internet, enabling the translation of human-readable domain names into IP addresses.

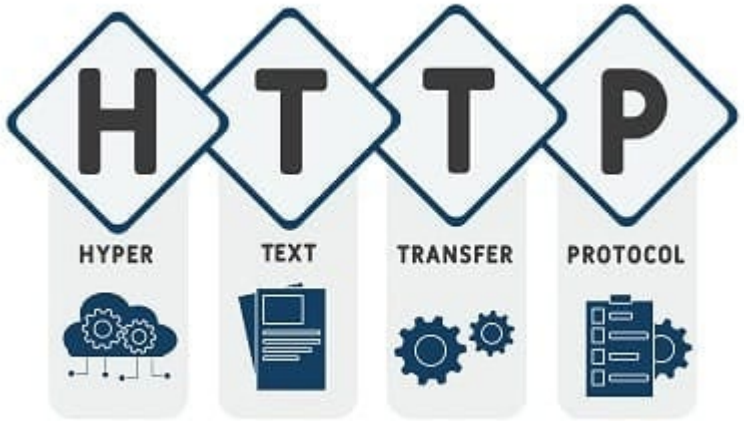
1. **DNS Query:** The process begins when a user types a domain name into a web browser.
2. **Recursive Resolver:** The query is sent to a DNS resolver, which starts the resolution process.
3. **Root Server:** The resolver queries a root server to find the top-level domain (TLD) server (e.g., .com, .org).
4. **TLD Server:** The resolver then queries the TLD server to find the authoritative DNS server for the specific domain.
5. **Authoritative DNS Server:** The authoritative server provides the IP address associated with the domain name.
6. **Response:** The resolver returns the IP address to the user's browser, which then connects to the web server.

HyperText Transfer Protocol (HTTP)

The **HyperText Transfer Protocol** (**HTTP**) facilitates the **transfer** of **hypertext documents**, such as HTML pages, between a **web server** and a **client** (browser).

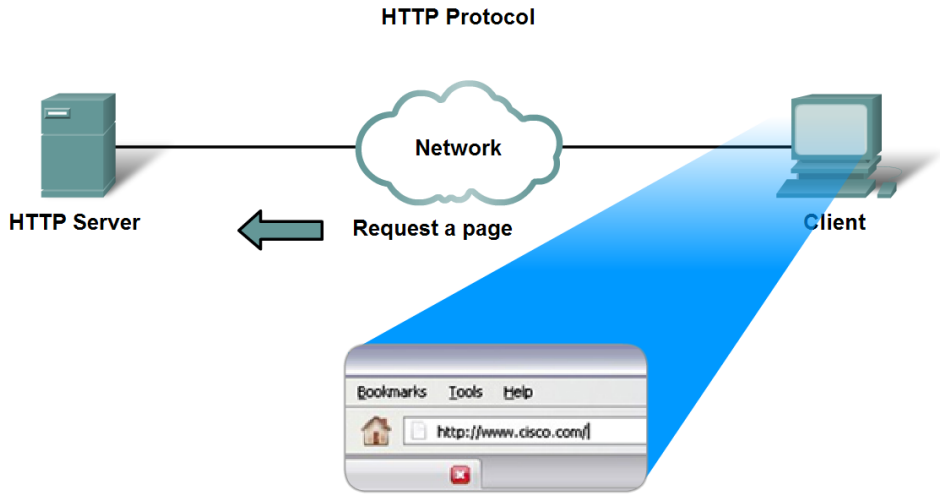
What is HTTPS?

HyperText Transfer Protocol Secure (**HTTPS**) is an extension of HTTP with added **security features**. It provides secure communication over a computer network by **encrypting** the data exchanged between the client and server.



HTTP – Web Browsing Process

1. **Entering the URL:** The user types a URL into the browser's address bar.
2. **DNS Resolution:** The browser queries a DNS server to resolve the URL to an IP address.
3. **Server Connection:** The browser establishes a connection to the identified server.
4. **Sending a Request:** Using HTTP or HTTPS, the browser sends a GET request to the server, typically requesting the default document (e.g., index.html).
5. **Receiving the Response:** The server responds by sending the HTML content of the requested webpage to the browser.
6. **Rendering the Page:** The browser interprets the HTML code and displays the formatted webpage in the browser window.



File Transfer Protocol (FTP)

The **File Transfer Protocol** (**FTP**) is a standard network protocol used for **transferring files** from one host to another over a TCP-based network, such as the internet. It facilitates the transfer of files between a **client** and a **server**.

1. **Connection Establishment:**

Control Connection: Client establishes a control connection to the server for sending commands.

Data Connection: A separate data connection is established for transferring files, which can use various ports.

2. **Authentication:**

Username and Password: The client provides specific credentials to authenticate with the server when accessing protected files. For publicly available files, the default username is 'anonymous' and the password is typically set to 'guest'.

3. **File Transfer:**

Commands: Client sends commands (e.g., RETR, STOR) over the control connection.

Data Transfer: Files are transferred over the data connection.

SMTP/POP3/IMAP – Email

What is SMTP?

The **Simple Mail Transfer Protocol** (**SMTP**) is a protocol used for **sending emails** from a client to a server or between servers.

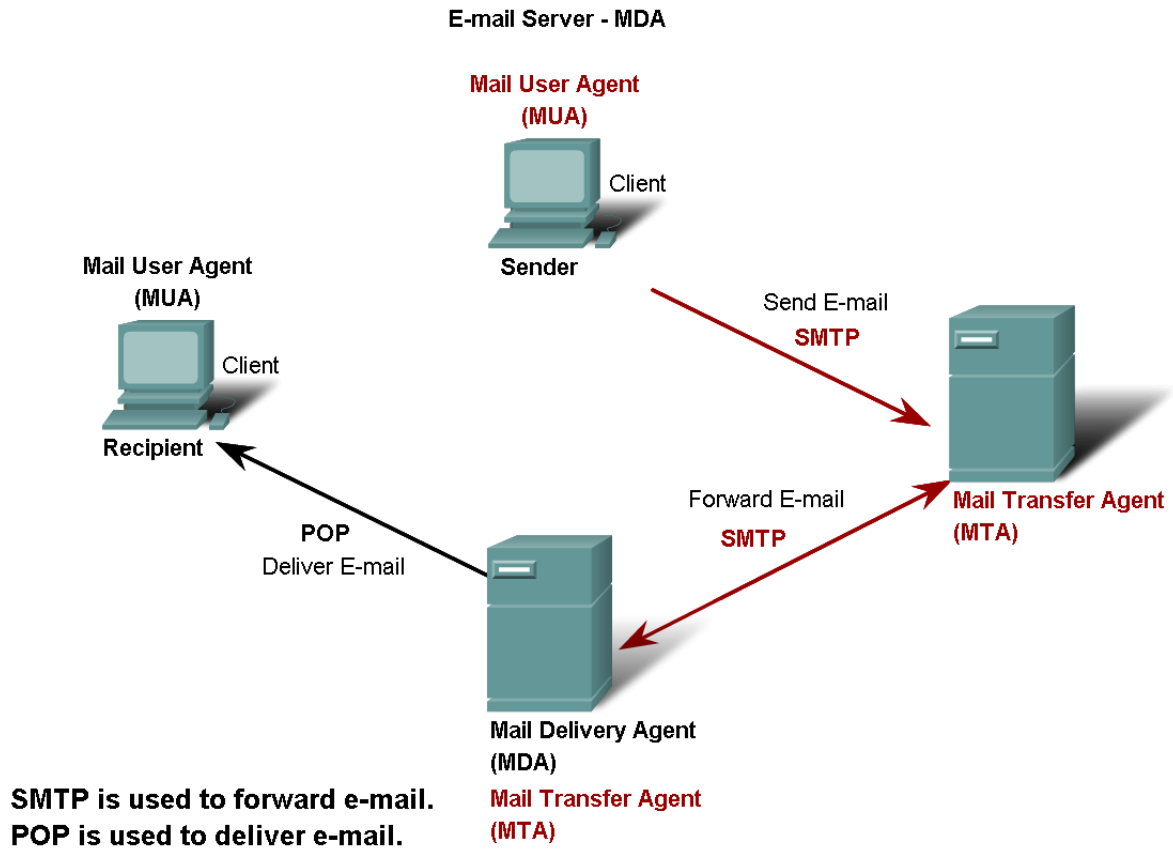
What is POP3?

The **Post Office Protocol version 3** (**POP3**) is a protocol used for **retrieving emails** from a mail server to a client. It **downloads** emails from the server to the client and emails are typically **deleted** from the server after download.

What is IMAP?

The **Internet Message Access Protocol** (**IMAP**) is a standard email protocol used to **access** and **manage** email messages on a **mail server**. Allows users to **view** and **manipulate** their email messages as if they were stored locally on their device, while keeping them on the **server**. Allows **multiple devices** to access the same mailbox, maintaining synchronization across devices.

SMTP/POP3 Example



Dynamic Host Configuration Protocol (DHCP)

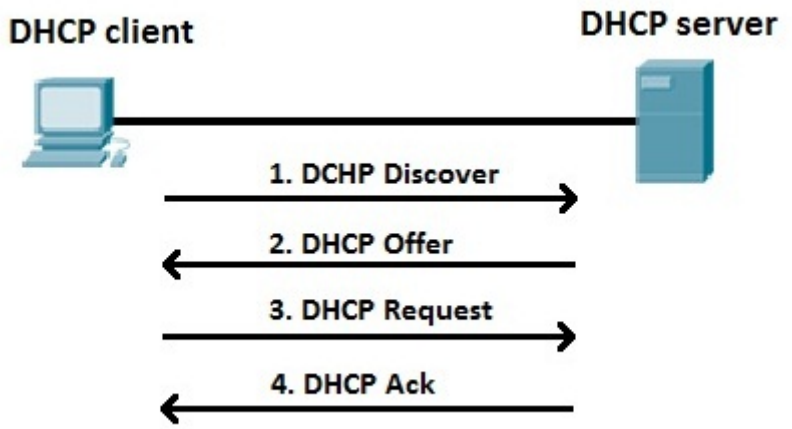
Dynamic Host Configuration Protocol (**DHCP**) is a network **management protocol** used to automatically assign **IP addresses** and other **network configuration parameters** to devices on a network. It reduces the need for manual configuration of IP addresses and ensures unique IP assignment.

1. **DHCP Discovery:** The client sends a DHCPDISCOVER broadcast message to find available DHCP servers.

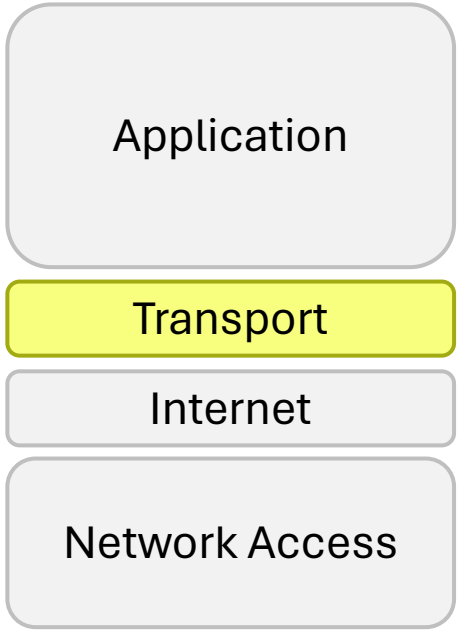
2. **DHCP Offer:** A DHCP server responds with a DHCPOFFER message, offering an IP address and other network configuration settings.

3. **DHCP Request:** The client replies with a DHCPREQUEST message, indicating acceptance of the offer.

4. **DHCP Acknowledgment:** The DHCP server confirms the lease with a DHCPACK message, finalizing the IP address assignment and configuration.



Transport Layer



Functions:

Provides either reliable or unreliable data transmission between hosts.

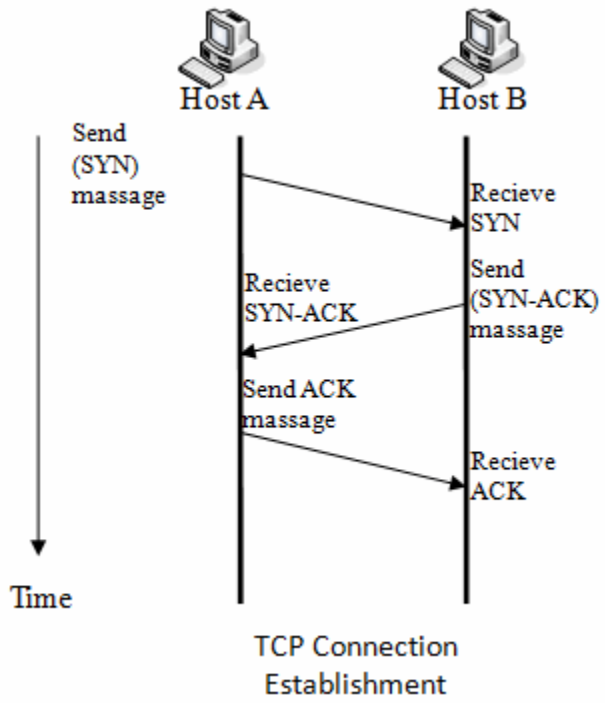
Includes mechanisms to ensure reliable data transmission.

Protocols:

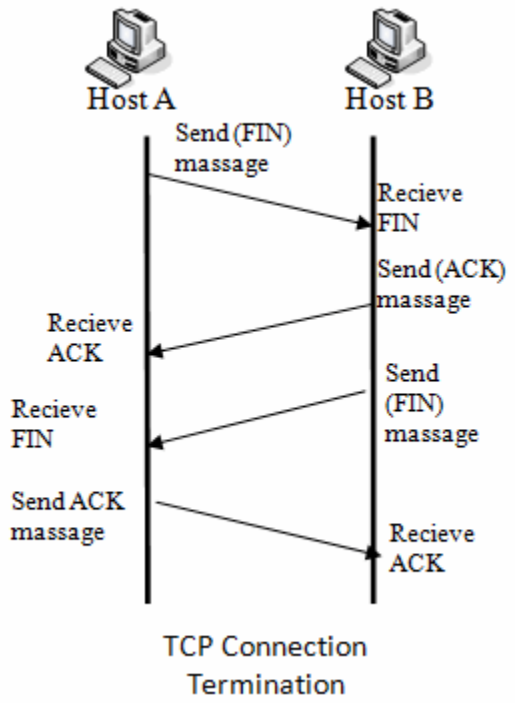
- **TCP:** Reliable, connection-oriented communication.
- **UDP:** Unreliable, connectionless communication.

Transmission Control Protocol (TCP)

The **Transmission Control Protocol** (**TCP**) is a **connection-oriented** protocol that ensures **reliable** data transmission. It provides **error checking**, **flow control**, and **acknowledgment** of data packets, ensuring that data is delivered in the **correct order** without loss or duplication.



TCP Connection Establishment



TCP Connection Termination

TCP Header Structure

Field	Length (bits)	Description
Source Port	16	Port number of the sending application
Destination Port	16	Port number of the receiving application
Sequence Number	32	Position of the first byte of data in the segment
Acknowledgment Number	32	Next expected byte from the sender
Data Offset	4	Size of the TCP header in 32-bit words
Reserved	3	Reserved for future use, must be zero
Flags	9	Control flags (e.g., SYN, ACK, FIN)
Window Size	16	Size of the sender's receive window
Checksum	16	Error-checking of the header and data
Urgent Pointer	16	Points to urgent data (if URG flag is set)
Options	Variable	Optional fields for additional control
Padding	Variable	Ensures header length is a multiple of 32 bits

KEY FIELD

Source and Destination Port:
Identifies the sending and receiving applications.

Sequence Number:
Tracks the position of data in the segment.

Acknowledgment Number:
Confirms receipt of data.

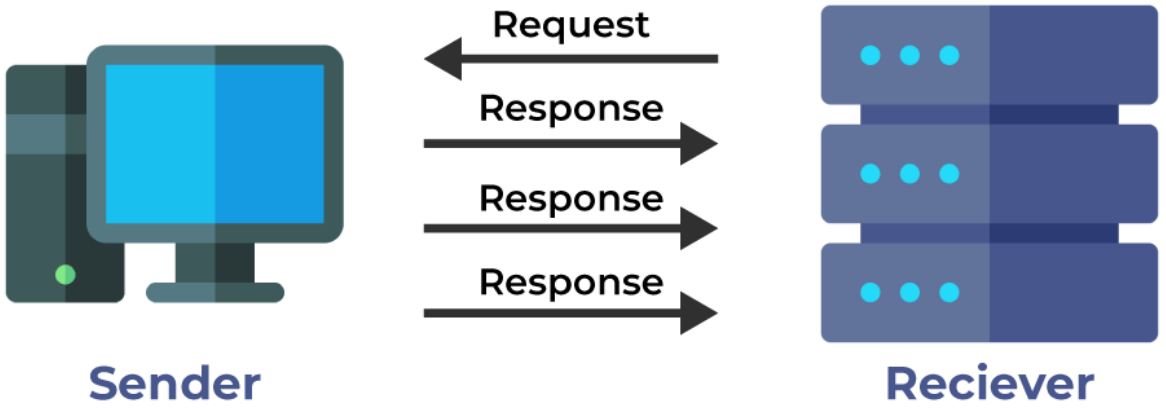
Checksum:
For error-checking of header and data

Flags:

- SYN: Synchronize sequence numbers (connection setup).
- ACK: Acknowledgment field significant.
- FIN: No more data from sender (connection teardown).

User Datagram Protocol (UDP)

The **User Datagram Protocol** (**UDP**) is a **connectionless** protocol that provides **unreliable** data transmission. It does **not guarantee delivery**, **order**, or **error-correction** of the packets. It is used in scenarios where speed is more critical than reliability, such as streaming audio or video.



UDP Header Structure

Field	Length (bits)	Description
Source Port	16	Port number of the sending application
Destination Port	16	Port number of the receiving application
Length	16	Length of the UDP header and data
Checksum	16	Error-checking of the header and data

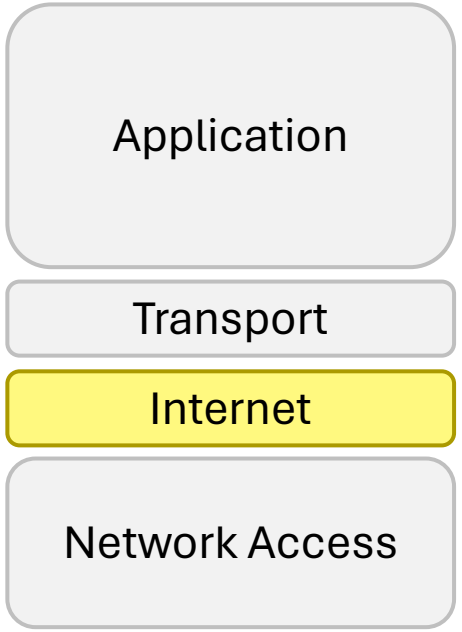
KEY FIELD

Source and Destination Port:
Identifies the sending and receiving applications.

Length:
Specifies the total length of the UDP header and data.

Checksum:
For error-checking of header and data

Internet Layer



Functions:

Manages logical addressing and routing of data packets.

Ensures data reaches the correct destination across multiple networks.

Protocols:

- **IP:** Primary protocol for routing (IPv4 and IPv6).
- **ICMP:** Error messages and operational information.
- **ARP:** Maps IP addresses to MAC addresses.

Internet Control Message Protocol (ICMP)

Internet Control Message Protocol (**ICMP**) is an Internet layer protocol used for sending **error** messages and **operational** information. It helps diagnose **network communication issues** and **report errors** back to the source IP address.

- **Diagnostic Tools:** Used in network utilities like *ping* and *traceroute* to test connectivity and trace paths.
- **Error Reporting:** Communicates network issues such as unreachable hosts, network congestion, and routing problems.

Common ICMP Message Types:

Echo Request (Type 8): Sent by the ping command to request a response from a host.

Echo Reply (Type 0): Sent in response to an echo request, indicating the host is reachable.

Destination Unreachable (Type 3): Indicates that a destination is unreachable for various reasons (e.g., network unreachable, host unreachable).

Time Exceeded (Type 11): Indicates that a packet has expired in transit (used by traceroute).

Redirect (Type 5): Instructs a host to use a different route for packets.

Address Resolution Protocol (ARP)

The **Address Resolution Protocol** (**ARP**) is an Internet layer protocol used to map an **IP address** to a **MAC address**. It facilitates communication within a local network by **linking** layer 2 IP addresses to layer 1 MAC addresses.

How it works:

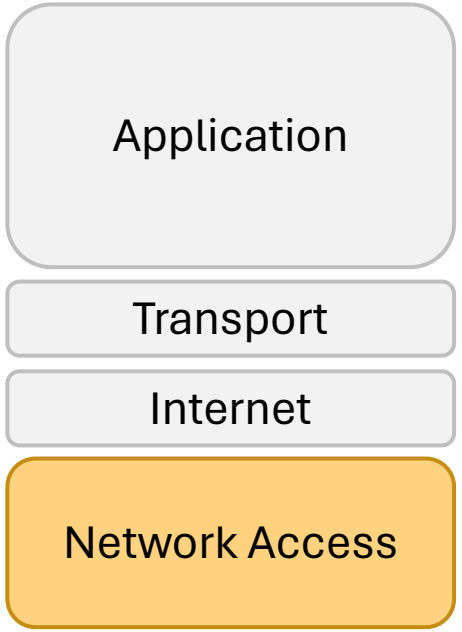
ARP Request: A device sends a broadcast ARP request to all devices on the local network, asking for the MAC address corresponding to a specific IP address.

ARP Reply: The device with the requested IP address responds with an ARP reply, providing its MAC address.

- **ARP Cache:**

A table stored in a device's memory that maintains a record of IP-to-MAC address mappings. It have a limited lifetime and are periodically refreshed.

Network Access Layer



Functions:

Manages data framing, physical addressing, and error detection at the data link level.

Controls hardware and software communication on the physical network.

Protocols:

- **Ethernet:** LAN technology.
- **Wi-Fi:** Wireless networking technology.

2. Linux

Shell Programming

Summary

- Cos'è uno Script
- Shell di esecuzione
- Esecuzione di uno Script
- Variabili locali e d'ambiente
- Read
- Operazioni aritmetiche
- Strutture di controllo (if, case, for, while)

Cos'è uno Script

Uno **script** è una **serie di comandi** di shell scritti in un file di testo.

Sono stati introdotti alcuni **costrutti** di controllo tipici di un vero **linguaggio di programmazione** per rendere gli script di shell più **potenti**.

Perché usare gli script di shell?

- Per **creare** nuovi **comandi personalizzati**
- Per **risparmiare** tempo (esecuzione di uno scipt, invece di molti comandi in sequenza)
- Per **automatizzare** dei task che devono essere **eseguiti frequentemente**
- Per **facilitare** i compiti dell'**amministratore di sistema**.

Come scrivere uno script?

Si può utilizzare un qualunque editor di testo (per esempio **nano**).

Scrivere uno Script

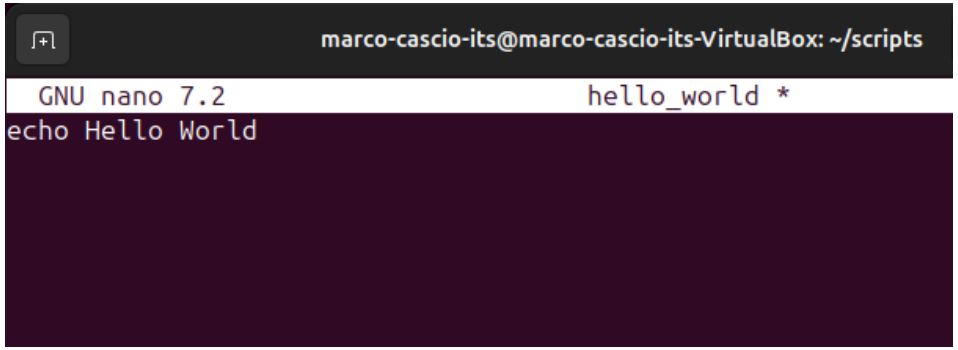
Proviamo a scrivere lo script **hello_world.sh**:

$ mkdir scripts - creiamo la directory in cui verranno salvati i nostri script

$ cd ./scripts - spostiamoci nella directory *scripts* appena creata

$ nano hello_world.sh - apriamo l'editor di testo e creiamo un file *hello_world.sh*

NOTA: l'estensione *.sh* facilmente i file come script di shell



Shell di esecuzione

La **prima riga** di uno script può essere utilizzata per **indicare** quale **shell** interpreta lo script:

#! /bin/bash - indica uno script per la shell bash

Per **conoscere** quale **shell** si sta utilizzando, si utilizza il comando:

$ echo $SHELL

Commenti/1

Tutte le **righe** che cominciano con il simbolo '#' sono considerati **commenti**, e vengono **ignorati** dall'interprete dello script.

Commenti/2

Attenzione!!!

`#!` e `#` sono **differenti**, in quanto:

- `#!` è una **sequenza di caratteri** speciale ed indica al sistema operativo quale **interprete** utilizzare per eseguire lo script e deve essere inserito nella **prima riga** di uno script per poter essere riconosciuto correttamente

- `#` è un simbolo utilizzato per indicare un **commento** all'interno di uno script. I commenti vengono **ignorati** dall'interprete dello script. Questo simbolo può essere usato in **qualsiasi riga** dello script per aggiungere **notazioni** o **spiegazioni**.

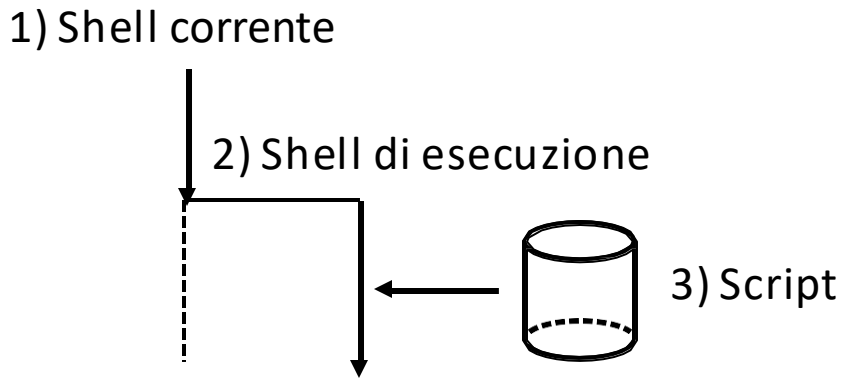
Esecuzione di uno Script/1

Per **eseguire** uno **script**, è possibile utilizzare:

1. **$ bash <** nome_script.sh - redirezione dello standard di input sullo script
2. **$ bash** nome_script.sh
3. **$ ./nome_script.sh**

- La shell **riconosce** che il file di input è uno **script** e **determina** quale **shell** deve essere utilizzata per eseguirlo.

- La shell che **esegue** lo script, lo esegue leggendo i **comandi** come se fossero **digitati direttamente** nella shell stessa.



ICTAcademy
INFORMATION AND COMMUNICATIONS TECHNOLOGY

Esecuzione di uno Script/2

Eseguiamo lo script **hello_world.sh**:

Modo 1.

$ bash < hello_world.sh

Esecuzione di uno Script/3

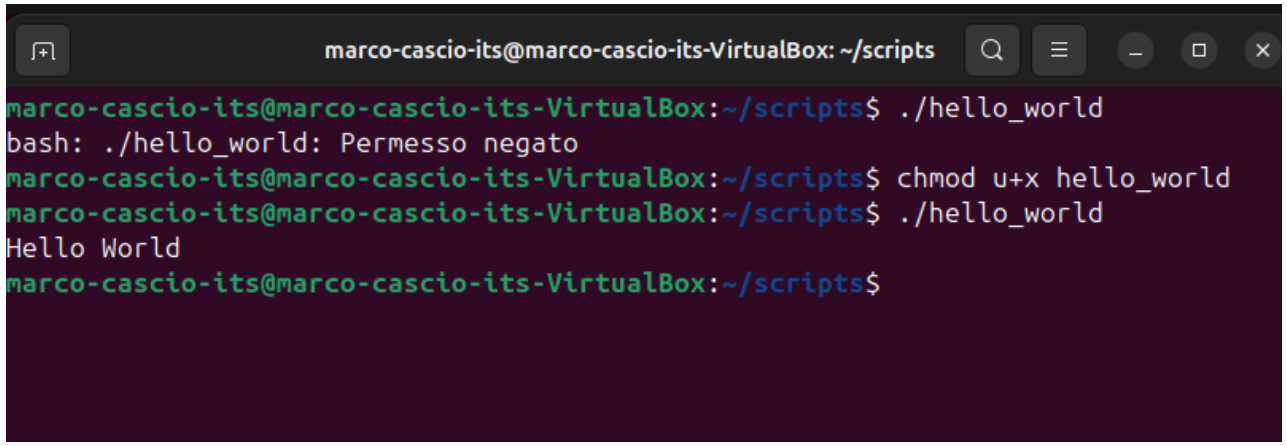
Eseguiamo lo script **hello_world.sh**:

Modo 2.

$ bash hello_world.sh

Esecuzione di uno Script/4

Eseguiamo lo script **hello_world.sh**:



<u>Modo 3.</u>

$ ./hello_world.sh

Output: *Permission denied*

- Cambio permessi:

$ chmod u+x hello_world.sh - assegna all'utente il permesso per eseguire lo script *hello_world.sh*

ICTAcademy
INFORMATION AND COMMUNICATIONS TECHNOLOGY

Variabili di shell

Una **variabile** è un **nome simbolico** al quale è associato un **valore**.

Le variabili sono utilizzate per **memorizzare** dati che possono essere utilizzati all'interno di **script** e **sessioni di shell**.

La shell permette di definire **due tipi** di **variabili**:

- **variabili locali**
- **variabili di ambiente**

Variabili locali/1

Le **variabili locali** non richiedono **alcun comando speciale** per essere **create**. Le variabili locali sono semplicemente definite con un'**assegnazione di valore**.

Le **variabili locali** hanno una **validità limitata** all'ambito della shell stessa, in quanto sono **visibili** ed **esistono** solo all'interno e durante l'esecuzione del **contesto** in cui sono state definite, come una **funzione** o uno **script**.

Il **nome** di una **variabile** può contenere **lettere**, **cifre** e **underscore**, ma il primo carattere non può essere un numero.

La **creazione** di una **variabile** e l'**assegnazione** di un **valore** si ottengono con una dichiarazione del tipo:

nome_variabile=valore (senza spazi)

Variabili locali/2

Assegnazione:

- Se **non** viene fornito il **valore** da assegnare, si intende che la **variabile** è uguale ad una **stringa vuota**
- Se la variabile **non esiste**, allora viene **creata** con il **valore specificato**
- Se la variabile **esiste**, il suo **valore precedente** viene **sovrascritto**

Una **variabile locale** può essere **cancellata** con il comando **unset**:

$ unset nome_variabile

Per **accedere** al **valore** di una **variabile** si utilizza il simbolo **$**:

$ $nome_variabile

Variabili locali/3

Esempi:

$ a=ciao
$ echo $a
Output: *ciao*

- creo la variabile locale *a* con valore *ciao*
- visualizzo/accedo al valore della variabile *a*

$ a=15
$ echo $a
Output: *15*

- sovrascrivo il valore della variabile locale *a* con valore *15*
- visualizzo/accedo al valore della variabile *a*

Variabili locali/4

$ b=ls - definisco la variabile locale *b* con valore il comando *ls*

$ echo $b - visualizzo/accedo al valore della variabile *b*

Output: *ls*

$ $b - esegue il comando **$ ls**

Output:

```
marco-cascio-its@federico-march-its-virtualbox:~/scripts$ $b
hello_world.sh
marco-cascio-its@federico-march-its-virtualbox:~/scripts$ b="ls -l"
marco-cascio-its@federico-march-its-virtualbox:~/scripts$ $b
totale 4
-rwxrw-r-- 1 marco-cascio-its marco-cascio-its 56 mag 21 18:07 hello_world.sh
```

```
marco-cascio-its@federico-march-its-virtualbox:~/scripts$ ls
hello_world.sh
marco-cascio-its@federico-march-its-virtualbox:~/scripts$ ls -l
totale 4
-rw-rw-r-- 1 marco-cascio-its marco-cascio-its 32 mag 21 19:00 hello_world.sh
```

Variabili locali/5

$ unset a - cancella la variabile a

$ unset b - cancella la variabile b

$ echo $a

Output:

$ echo $b

Output:

Variabili di ambiente/1

Le **variabili di ambiente** sono visibili a **tutti i processi figli** della shell. Questo significa che quando un **processo** viene eseguito dalla shell, **erediterà** le **variabili di ambiente** definite nella **shell genitore**.

Le **variabili di ambiente** persistono e sono **accessibili** ai processi figli finché la **shell genitore** è **attiva** o la **variabile** non viene esplicitamente **eliminata**.

Esiste un insieme di **variabili di ambiente predefinite**, che contengono informazioni utili sull'ambiente di esecuzione. Ad esempio:

- **$HOME**: indica il path della home directory dell'utente
- **$PATH**: elenco delle directory in cui il sistema cerca i comandi da eseguire, dopo aver digitato un comando.
- **$MAIL**: indica il path della mailbox personale dell'utente
- **$USER**: username dell'utente attualmente loggato
- **$SHELL**: path della shell di login (/bin/bash)
- **$TERM**: specifica il tipo di terminale utilizzato (utile per compatibilità grafica)

Variabili di ambiente/2

$ echo $HOME

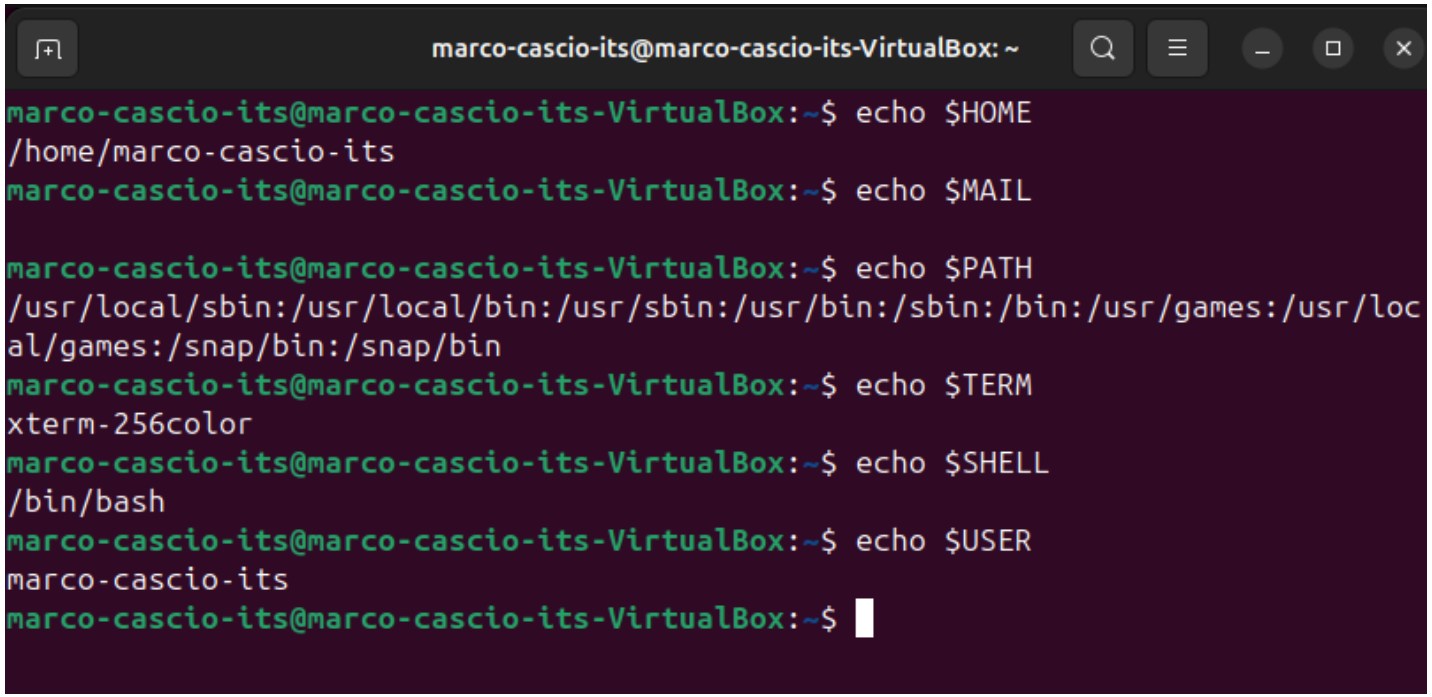
$ echo $MAIL

$ echo $PATH

$ echo $TERM

$ echo $SHELL

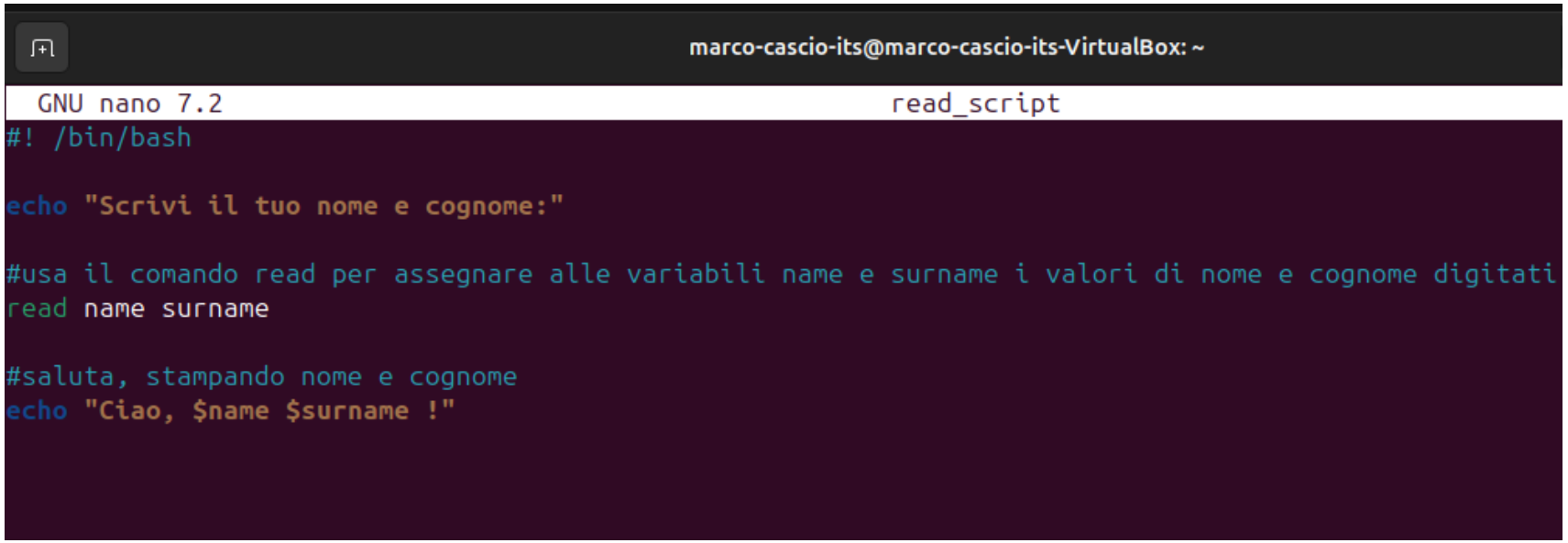
$ echo $USER



Read/1

Il comando **read** legge dallo **standard input** ed assegna alla **prima variabile** specificata la **prima parola** digitata, alla **seconda variabile** specificata la **seconda parola** digitata e così via.

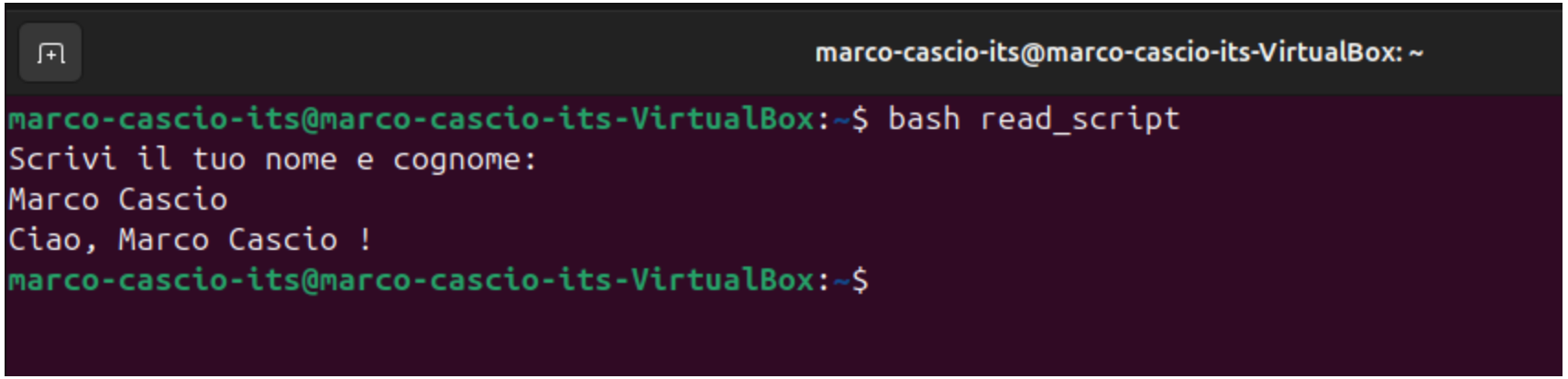
$ nano read_script.sh - crea lo script *read_script.sh*



ICTAcademy
INFORMATION AND COMMUNICATIONS TECHNOLOGY

Read/2

$ bash read_script.sh - esegue lo script *read_script.sh*



Operazioni aritmetiche/1

Operatori aritmetici comuni:

+	**somma**
-	**sottrazione**
*	**prodotto**
/	**divisione intera**
%	**resto**

Operatori relazionali comuni:

>	**maggiore**	<	**minore**
> =	**maggiore o uguale**	< =	**minore o uguale**
=	**uguale**	!=	**diverso**
&&	**AND logico**	||	**OR logico**

Operazioni aritmetiche/2

Il comando ***$ expr <espressione aritmetica>*** permette di usare **variabili numeriche** e di **eseguire** con esse semplici **calcoli**.

Esempio:

$ expr 1 + 2 (attenzione agli spazi)

Output: *3*

Esempio:

$ a=9 b=3

- definisce due variabili numeriche *a* e *b*

Esempio:

$ expr $a - $b (attenzione agli spazi)

- effettua la sottrazione tra *a* e *b*

Output: *6*

ICTAcademy
INFORMATION AND COMMUNICATIONS TECHNOLOGY

Operazioni aritmetiche/3

Le **espressioni aritmetiche** possono essere anche rappresentate come:

- ***$((<espressione>))***

Esempio:

$ echo $(($a+$b))

Output: *12*

oppure come:

- ***$[<espressione>]***

Esempio:

$ echo $[$a - $b]

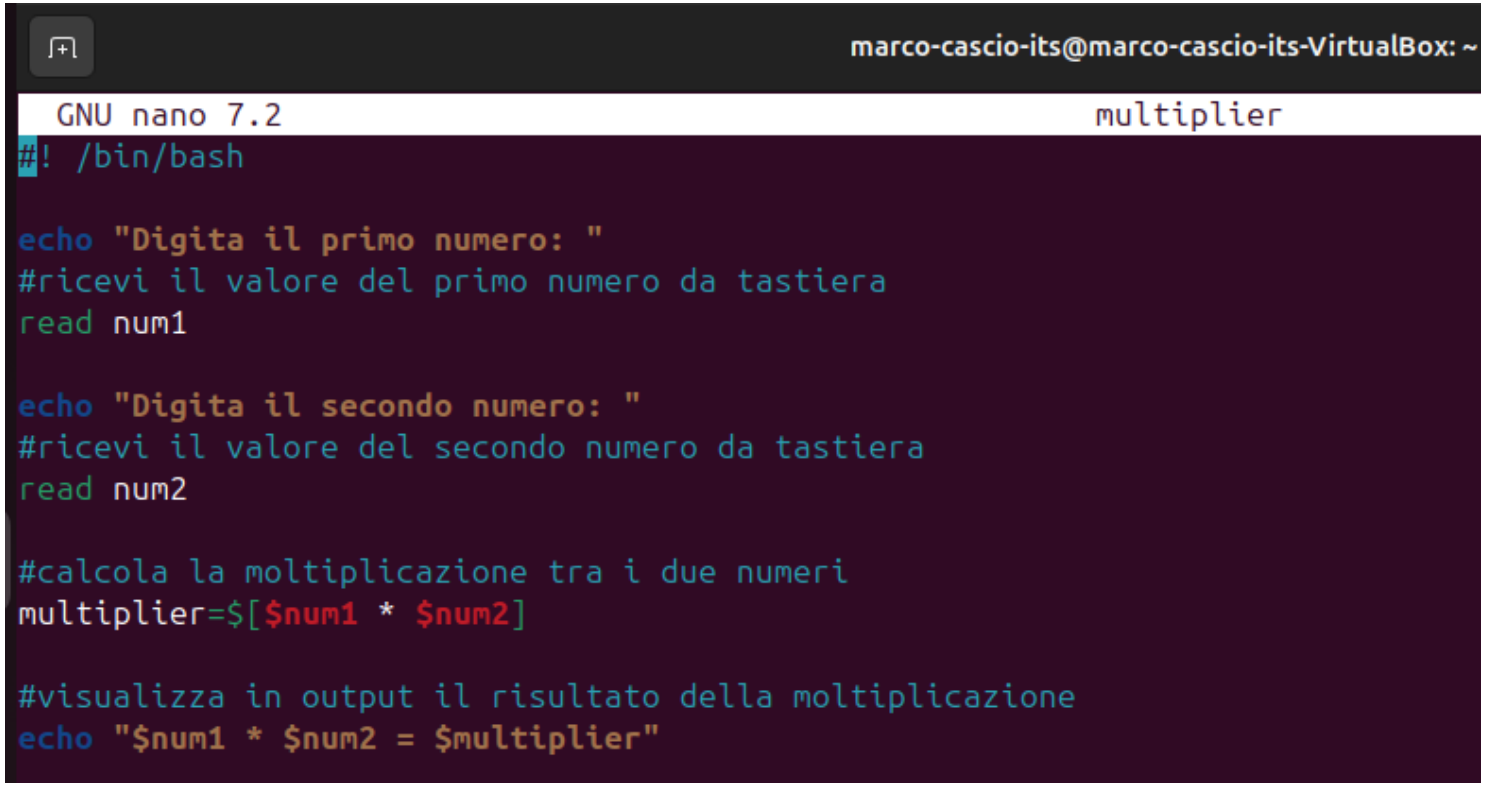
Output: *6*

Operazioni aritmetiche/4

Scrivere uno script *multiplier.sh* che consente di moltiplicare due numeri presi in input da tastiera.

Soluzione:

$ nano multiplier.sh



ICTAcademy
INFORMATION AND COMMUNICATIONS TECHNOLOGY

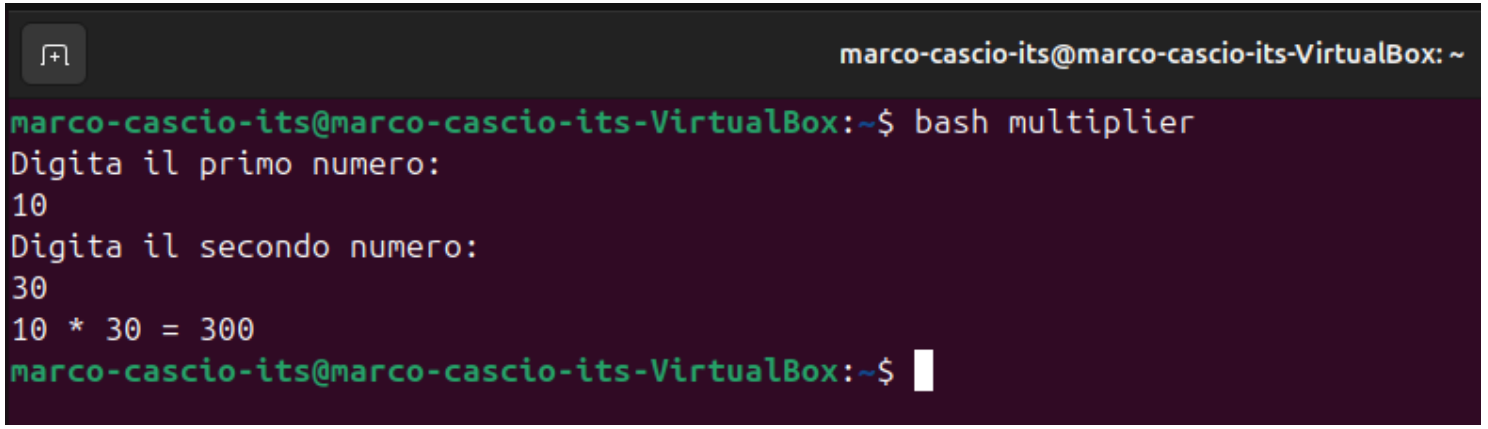
Operazioni aritmetiche/5

Scrivere uno script *multiplier.sh* che consente di moltiplicare due numeri presi in input da tastiera.

Soluzione:

$ bash multiplier.sh

Output:



Strutture di controllo

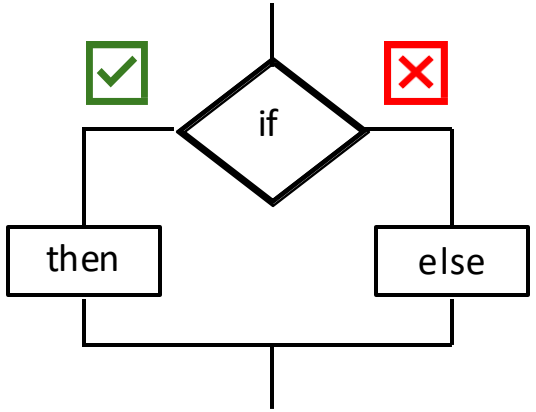
In ambiente Linux (come anche in programmazione) per compiere una determinata azione si usano delle **strutture di controllo**.

Le principali **strutture di controllo** sono:

- **if**
- **case**
- **for**
- **while**

Struttura if/1

if *listacomandi1* (oppure *condizione*)
then *listacomandi2*
else *listacomandi3*
fi



Se *listacomandi1* ha successo oppure la *condizione* è verificata:

- viene eseguito *listacomandi2,*

altrimenti:

- viene eseguita *listacomandi3* in sequenza.

Struttura if/2

- **Operatori matematici**

-eq	uguale a
-ne	diverso da
-lt	minore di
-le	minore o uguale a
-gt	maggiore di
-ge	maggiore o uguale a

- **Operatori logici**

!expr	NOT
expr1 **-a** expr2	AND
expr1 **-o** expr2	OR

Struttura if/3

- **Operatori su file**

-s file	verifica se il file ha dimensioni superiori a 0
-f file	verifica se il file esiste e non è una directory
-d dir	verifica se il dir esiste ed é una directory
-w file	verifica se il file è scrivibile
-r file	verifica se il file é read-only
-x file	verifica se il file é eseguibile

Struttura if/4

Il comando **test** è utilizzato per verificare se una condizione è **true** o **false** a seconda del tipo di test:

$ test condizione

Esempio:

$ test –d scripts - verifica se *scripts* è una directory

$ test $a **–le** $b - verifica se *a* è *minore o uguale* a *b*

Struttura if/5

Esempio 1:

Scrivere uno script *file_dir.sh* che accetti due parametri, ovvero il nome di un file ed il nome di una directory.

Lo script deve verificare se il file e la directory esistono; in caso affermativo, visualizzare un messaggio a riguardo in output.

Soluzione:

$ nano file_dir.sh - crea lo script *file_dir.sh*

```
 GNU nano 7.2                                          file_dir
#! /bin/bash

#testa se il primo argomento passato come input  ($1) è un file
if test -f $1
#se $1 è un file, allora visualizzare un messaggio a riguardo in output
then echo "$1 esiste ed è un file"
else echo "$1 non esiste oppure non è un file"
fi

#testa se il secondo argomento passato come input ($2) è una directory
if test -d $2
then echo "$2 esiste ed è una directory"
else echo "$2 non esiste oppure non è una directory"
fi
```

Struttura if/6

Soluzione

$ touch my_file.txt - crea un file vuoto chiamato *my_file.txt*

$ bash file_dir.sh my_file.txt my_dir - esegue lo script *file_dir.sh* passando come input *my_file.txt* e *my_dir*

Output: *my_file.txt esiste ed è un file*

my_dir non esiste oppure non è una directory

$ mkdir my_dir - crea una directory chiamata

$ bash file_dir.sh my_file.txt my_dir - esegue lo script *file_dir.sh* passando come input *my_file.txt* e *my_dir*

Output: *my_file.txt esiste ed è un file*

my_dir esiste ed è una directory

Struttura if/7

Esempio 2

Scrivere uno script *file_dir.sh* che accetti due parametri, ovvero il nome di un file ed il nome di una directory.

Lo script deve verificare se il file e la directory esistono; in caso affermativo, spostare il file nella directory in questione e visualizzare una scritta con la conferma dell'operazione avvenuta.

Soluzione

$ nano file_dir.sh - crea lo script *file_dir.sh*

```
 GNU nano 7.2                                    file_dir *
#! /bin/bash

#testa se il primo argomento passato come input  ($1) è un file
#e se il secondo argomento passato come input ($2) è una directory
if test -f $1 -a -d $2
#in caso affermativo, sposta il file $1 nella directory $2
then mv $1 $2/$1 ; echo "$1 è stato spostato nella directory $2" ;
else echo "Errore!"
fi
```

Struttura if/8

Soluzione

$ ls - visualizza i file e le directory nella working directory

$ bash file_dir.sh my_file.txt my_dir - esegue lo script file_dir.sh passando come input *my_file.txt* e *my_dir*

Output:

my_file.txt è stato spostato nella directory my_dir

$ ls my_dir - visualizza i file nella directory *my_dir*, per verificare lo spostamento

Struttura case/1

case *test_string* **in**

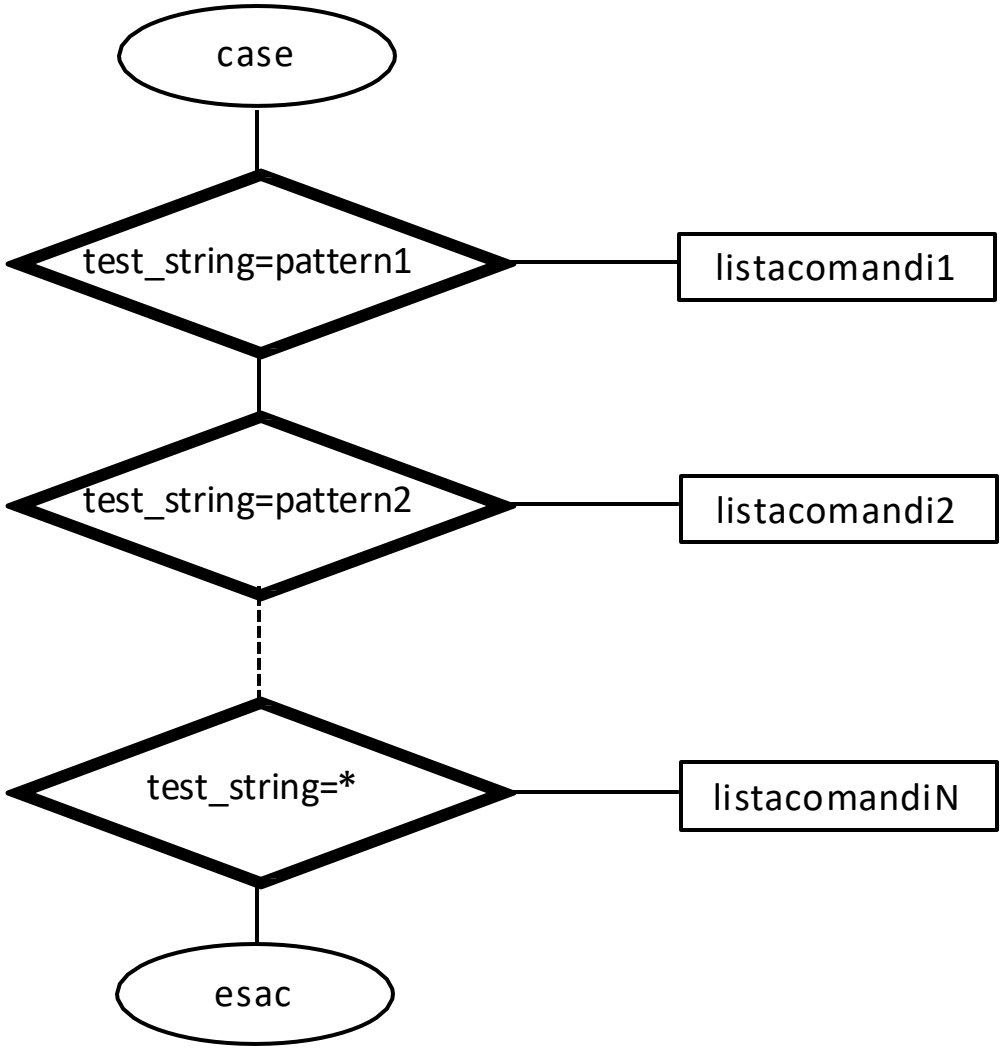
pattern1 **)** *listacomandi1* **;;**

pattern2 **)** *listacomandi2* **;;**

…

* **)** *listacomandiN* **;;**

esac



La struttura **case** consente di **effettuare** una **scelta** nell'**esecuzione** di varie **liste di comandi**.

La **scelta** viene fatta **confrontando** una **stringa** (**test_string**) con una **serie di modelli** (**pattern**) nell'**ordine** in cui esse compaiono.

Se **esiste** una **corrispondenza** con uno dei **pattern**, la relativa **lista di comandi** viene **eseguita**. **Altrimenti**, viene eseguita la **lista di comandi** nel specificata nel **pattern *)**.

Struttura case/2

Esempio 1

Scrivere lo script *tasto.sh* che chieda all'utente di premere un tasto e che visualizzi in output un messaggio che specifica il tipo di tasto premuto.

Ad esempio, visualizzare in output ''Lettera'' se l'utente ha premuto una lettera [a-z], ''Numero'' se l'utente ha premuto un numero [0-9], ''Punteggiatura, spaziatura o altro'' se l'utente ha premuto altri tipi di tasti.

Soluzione

$ nano tasto.sh - crea lo script *tasto.sh*

```
  GNU nano 7.2                                                tasto
#! /bin/bash

#chiedere all'utente di premere un tasto
echo "Premi un tasto e poi invio: "
#salva nella variabile tasto il tasto premuto dall'utente
read tasto

#struttura case per analizzare il tasto premuto
case $tasto in
[a-z] ) echo Lettera ;; #pattern1
[0-9] ) echo Numero ;; #pattern2
* ) echo Punteggiatura, spaziatura o altro ;; #pattern *)
esac
```

Struttura case/3

Soluzione

$ bash tasto.sh - esegue lo script *tasto.sh*

Output:

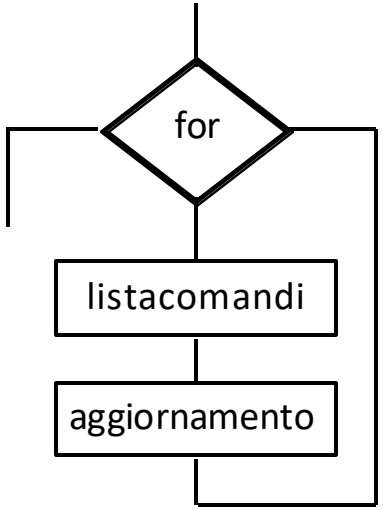
```
marco-cascio-its@marco-cascio-its-VirtualBox:~/scripts$ bash tasto
Premi un tasto e poi invio:
y
Lettera
marco-cascio-its@marco-cascio-its-VirtualBox:~/scripts$ bash tasto
Premi un tasto e poi invio:
9
Numero
marco-cascio-its@marco-cascio-its-VirtualBox:~/scripts$ bash tasto
Premi un tasto e poi invio:
.
Punteggiatura, spaziatura o altro
```

Struttura for/1

for *i* **in** *word1 word2…*

do *listacomandi*

done



La struttura **for** esegue una **scansione di elementi** ed in corrispondenza di questi **esegue** una **lista di comandi**.

Alla variabile **i** sono assegnati a turno i valori di **word1**, **word2**, …, per ogni iterazione del ciclo.

Struttura for/2

Esempio 1

Scrivere lo script *mk_planets.sh*. Questo script deve creare una directory chiamata *planets*. All'interno di tale directory creare 8 file, ognuno dei quali deve essere nominato con il nome di uno dei pianeti del Sistema Solare:

mercurio, venere, terra, marte, giove, saturno, urano, nettuno

Soluzione

$ nano mk_planets.sh - crea lo script *mk_planets.sh*

```
  GNU nano 7.2                                    mk_planets
#! /bin/bash

#crea la directory "planets"
mkdir planets

#per ogni pianeta
for planet in mercurio venere terra marte giove saturno urano nettuno
do
        #crea un file con il nome del pianeta nella directory "planets"
        touch planets/$planet
done
```

ICTAcademy
INFORMATION AND COMMUNICATIONS TECHNOLOGY

Struttura for/3

Soluzione

$ ls - visualizza i file e le directory nella working directory

$ bash mk_planets.sh - esegue lo script *mk_planets.sh*

$ ls - visualizza i file nella working directory per verificare che la directory *planets* sia stata create

$ ls planets - visualizza i file in *planet* per verificare le directory con i nomi dei pianeti siano state create

Output:

```
marco-cascio-its@marco-cascio-its-VirtualBox:~/scripts$ bash mk_planets
marco-cascio-its@marco-cascio-its-VirtualBox:~/scripts$ ls
file_dir  hello_world  mk_planets  multiplier  my  my_dir  planets  tasto
marco-cascio-its@marco-cascio-its-VirtualBox:~/scripts$ ls planets
giove  marte  mercurio  nettuno  saturno  terra  urano  venere
```

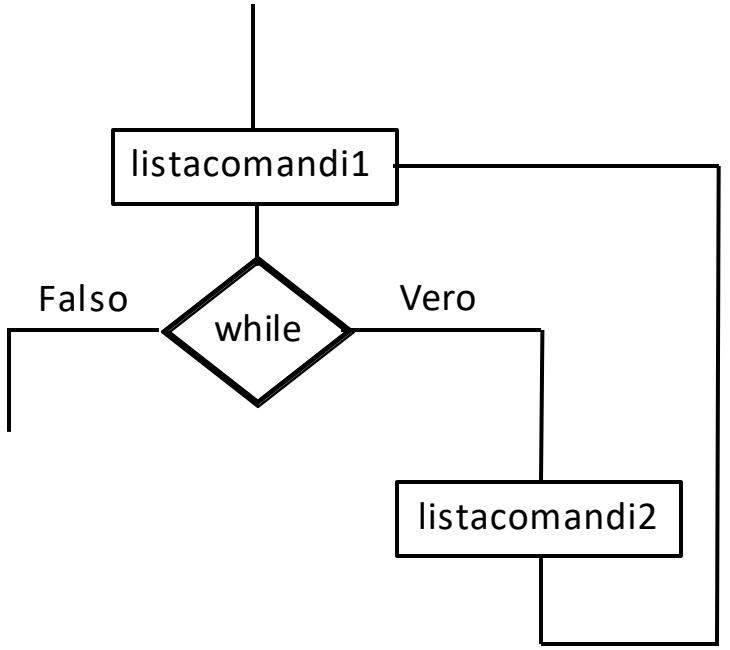
Struttura while/1

while *listacomandi1* (o *condizione*)

do *listacomandi2*

done

Con la struttura **while**, la listacomandi2 viene **eseguita finché** la **listacomandi1** (o una **condizione**) risulta essere **vera**.



Struttura while/2

Esempio 1

Scrivere lo script *numbers_to_file.sh*. Questo script deve contare i numeri da 1 a 30 usando l'istruzione while e se il numero contato è pari, inserire tale numero nel file *pari.txt*.

Soluzione

$ nano numbers_to_file.sh - crea lo script *numbers_to_file.sh*

Struttura while/3

Soluzione

```
 GNU nano 7.2                                    numbers_to_file *
#!/bin/bash

#crea un file chiamato pari
touch pari

#inizializzo il conto, facendo partire la conta da 1
n=1

#struttura while per la conta, finchè n è minore o uguale di 30
while test $n -le 30
do

#mostra in output il numero contato
echo $n

#controlla se il numero contato è pari, ovvero se il calcolo del resto della divisione n/2 è pari a 0
if test $[$n % 2] -eq 0
#se il numero  contato è pari, inseriscilo nel file pari
then echo $n >> pari
fi

#aggiorna il conto
n=$[$n + 1]

done

#notifica all'utente che il conto è finito
echo Conto Finito!
```

Struttura while/3

Soluzione

$ bash numbers_to_file.sh - esegue lo script *numbers_to_file.sh*

Output:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
Conto Finito!
```

Struttura while/4

Soluzione

$ cat *pari.txt* - legge il contenuto del file *pari.txt*

Output:

```
marco-cascio-its@marco-cascio-its-VirtualBox:~/scripts$ cat pari
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
```