

# NETWORK PROGRAMMING

## IT432 - NETWORK PROGRAMMING

DR. ALI HUSSEIN AHMED

[ALI.HUSSEIN@AUN.EDU.EG](mailto:ALI.HUSSEIN@AUN.EDU.EG)

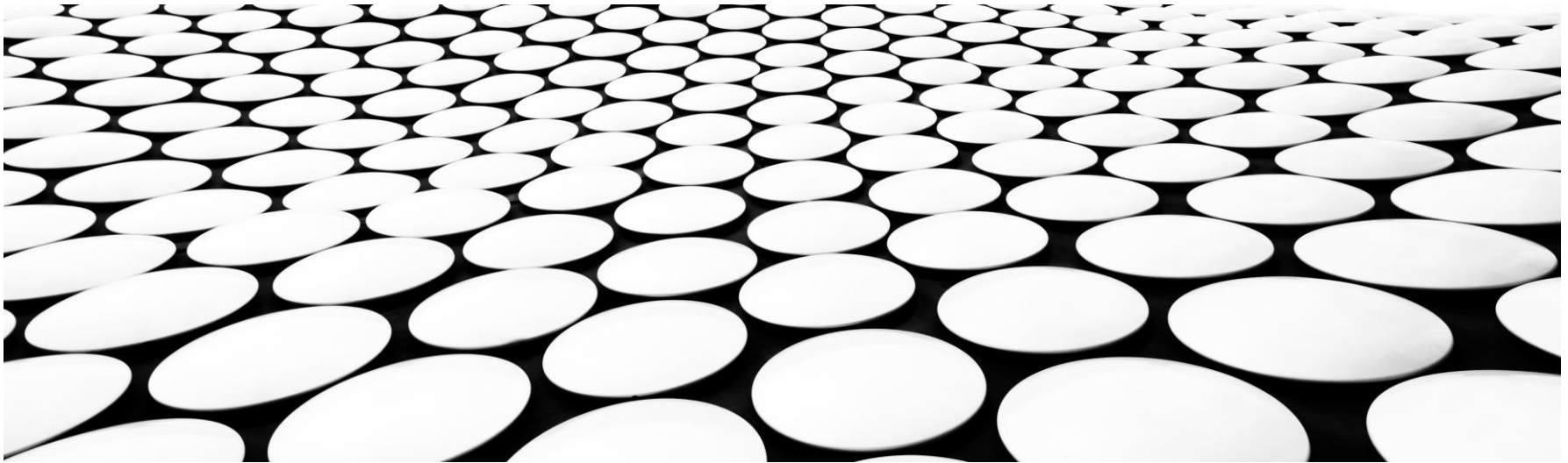
IT DEPT - OFFICE NO. 312

2022



كلية معتمدة من الهيئة القومية  
لضمان الجودة والاعتماد

# LECTURE 0

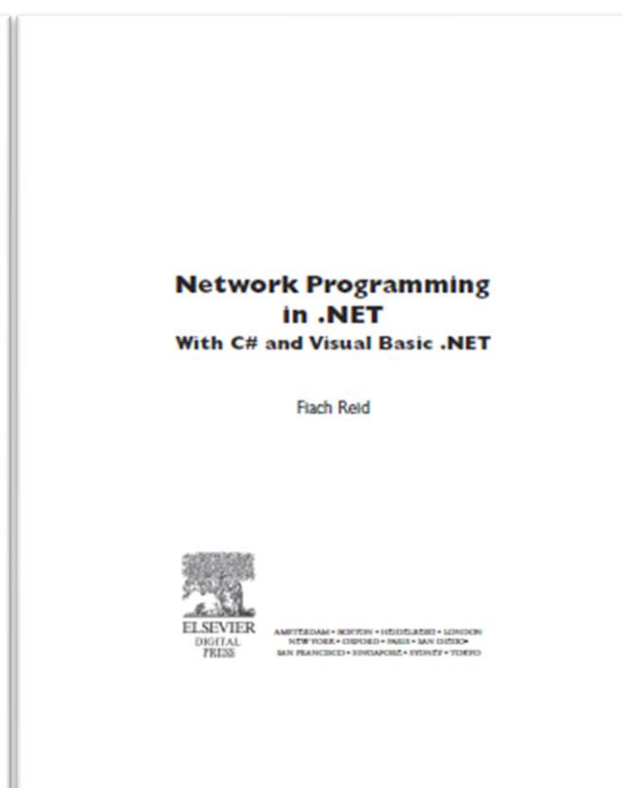
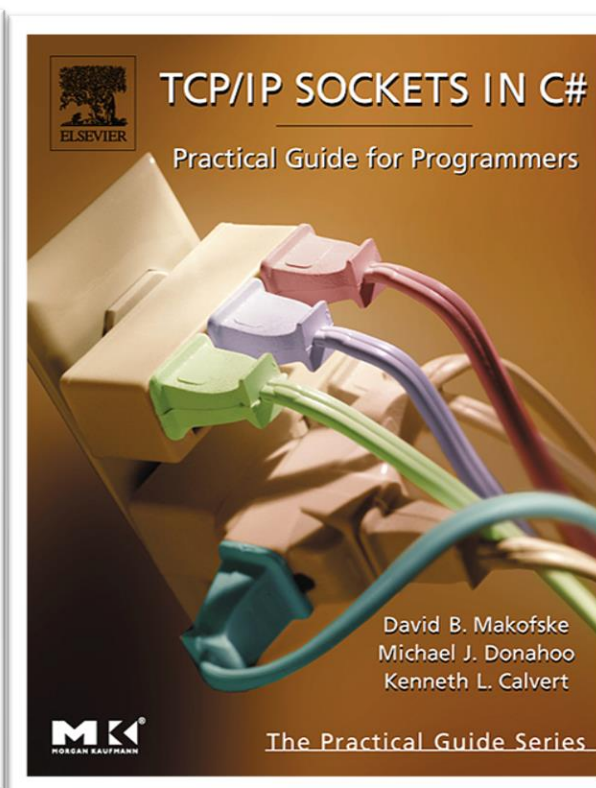
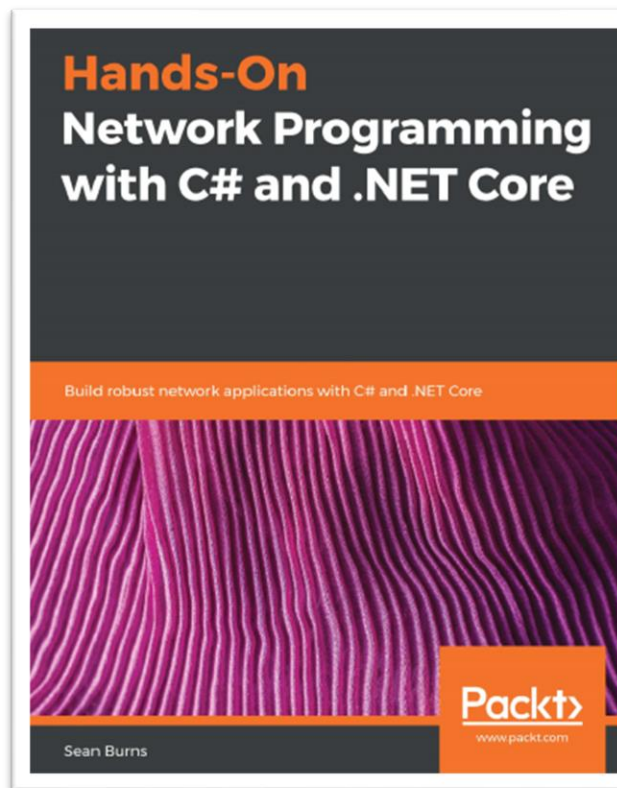




## RESOURCES

- Book1: Hands-On Network Programming with C# and .NET Core, 2019.
- Book2: Network Programming in .NET With C# and Visual Basic .NET, Fiach Reid.
- Book3: TCP/IP Sockets in C# Practical Guide for Programmers
- Book:4: C# Network Programming, by Richard Blum
- Lecture notes.

## BOOKS





## **YOUR BEST STRATEGY – ROADMAP TO SUCCESS IN THIS COURSE**

- Come to every lecture
- Read the topics related to network protocols and network programming
- Do not wait till last minute to prepare for exam or work on project
- Enjoy !



# COURSE CONTENT

- Introduction
- Streams
- Serialization
- Threading
- Socket Programming
  - Client/server
  - Web server
  - DNS
  - Peer to peer
  - Multicast and broadcast
- Project

# PLAN :LECTURES AND CHAPTER MAPPING

- Lecture 0: Course Introduction
- Lecture 1 :
- Lecture 2:
- Lecture 3:
- Lecture 4:
- Lecture 5:
- Lecture 6:
- Lecture 7:
- Lecture 8:
- Lecture 9:
- Lecture 9:
- Lecture 10:



## LECTURE 0

- **What is Network Programming?**
- **Network Elements and protocols revision.**
- **C# Introduction**
- **C# Streams**



## NETWORK PROGRAMMING

- **Network Programming** involves writing programs that communicate with other programs across a computer network.
- We use the Socket API .
- It can be achieved by any programming language, but during our course we will use C#.
- Example of programs:
  - Text, Audio and video chat.
  - Video broadcast and multicast.
  - Web browser
  - File exchange over network.
  - ....

# WHAT'S IS A NETWORK?

- The term **network** can refer to any interconnected group or system.
- A **computer network** is composed of multiple computers connected together using a telecommunication system.
- “...communication system for connecting end-systems”
- End-systems or “hosts”
  - PCs
  - dedicated computers
  - network components
- Interconnection may be any medium capable of communicating information:
  - Copper wire
  - Lasers (optical fiber)
  - Radio /Satellite link
  - Cable (coax)
- Example: Ethernet.



## WHY NETWORK?

- Sharing resources
  - Resources become available regardless of the user's physical location (server based, peer2peer)
- Load Sharing/utilization
  - Jobs processed on least crowded machine
- High reliability
  - Alternative sources for Info.

# WIDE VARIETY TYPES OF NETWORKS

## ■ Circuit switched

- dedicated circuit per call
- performance (guaranteed)
- call setup required
- telephone system

## ■ Packet switched:

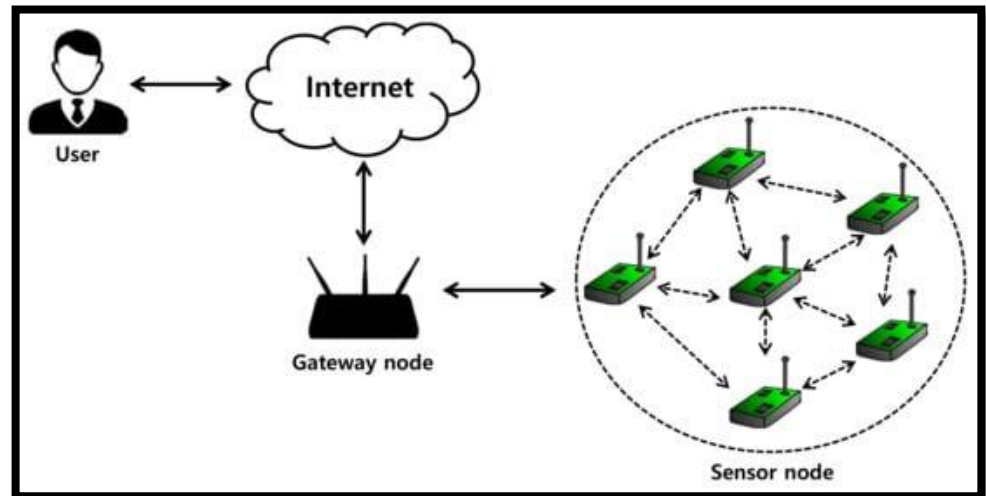
- data sent through the net in discrete “chunks”
- user A, B packets *share* network resources
- resources used *as needed*
- store and forward: packets move one hop at a time
- The Internet (TCP/IP)

# WHAT IS INTERNET?

- What is internet?
  - Network of networks
- What is *the* Internet?
  - A global internet based on IP protocol
- Internet applications:
  - Email
  - File transfer
  - File sharing
  - Resource distribution (DNS)
  - World wide web
  - Video conference
  - Gaming

# NEW EMERGING NETWORKS !

- Sensor Networks
  - Small devices equipped by sensor and connected to the internet
  - May include an environmental actuation
- Lots of them (density)
- Cheap unreliable elements
- Run on batteries
- Location becomes a key attribute
- Information sensing around users



## LAN & WAN

- LAN : connects computers that are physically close together ( < 1 mile (1.6 KM)).
  - high speed
  - multi-access
  - common technologies: Ethernet 10 Mbps, 100Mbps
- WAN connects computers that are physically far apart. “long-haul network”.
  - Slower than a LAN.
  - Less reliable than a LAN.
  - point-to-point
  - Technologies: telephone lines , Satellite communications



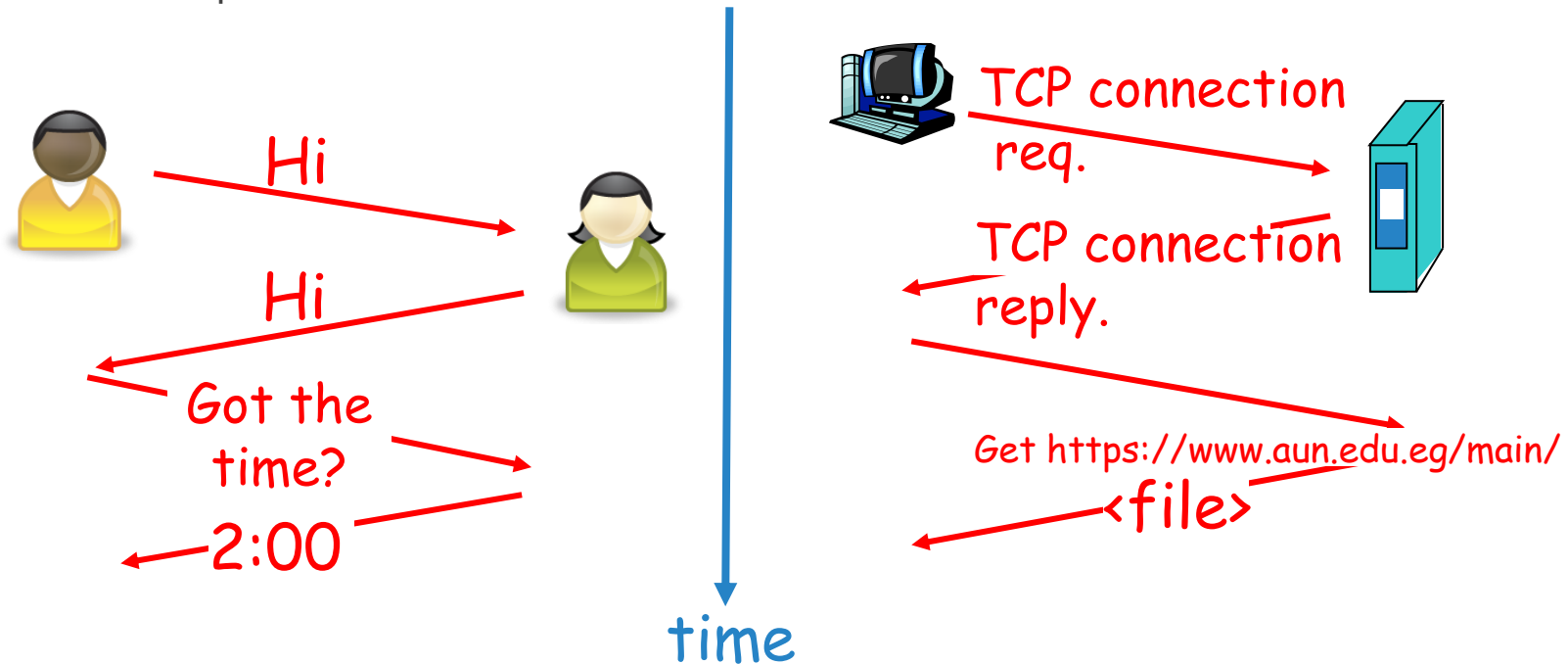
# NETWORK PROTOCOLS





## WHAT'S A PROTOCOL?

- a human protocol and a computer network protocol:

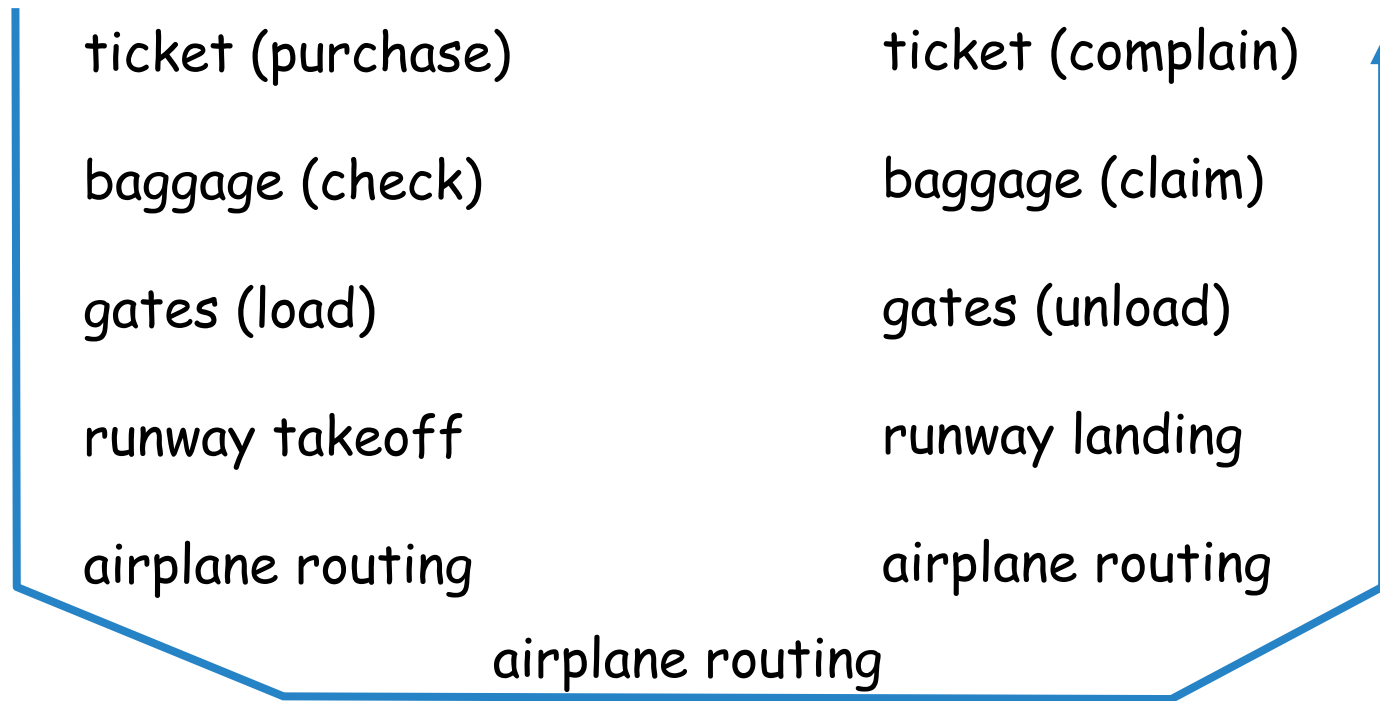


*protocols define syntax , semantics and timing information required for entities to communicate (understand each other)*

## LAYERING

- Lot of functions needs to be organized
- Communicating entities are widely distributes (inherits heterogeneity)
- How to manage and enhances ?
  - Categorized function groups (services) into disjoint sets
  - Assign every group to set of similar entities
  - Each entity can perform service to other (upper) entity
  - Upper entity a summarized command
  - Lower layer performs the details
  - This is called the layered model

## ORGANIZATION OF AIR TRAVEL



Although this course is about network programming  
(and not about networking in general),  
an understanding of a complete reversion on network model is essential.

## ORGANIZATION OF AIR TRAVEL: PEER LAYER VIEW

ticket (purchase)	ticket (complain)
baggage (check)	baggage (claim)
gates (load)	gates (unload)
runway takeoff	runway landing
airplane routing	airplane routing
airplane routing	

**Layers:** each layer implements a service

- via its own internal-layer actions
- relying on services provided by layer below

## DISTRIBUTED IMPLEMENTATION OF LAYER FUNCTIONALITY

Departing airport

ticket (purchase)

baggage (check)

gates (load)

runway takeoff

airplane routing

ticket (complain)

baggage (claim)

gates (unload)

runway landing

airplane routing

arriving airport

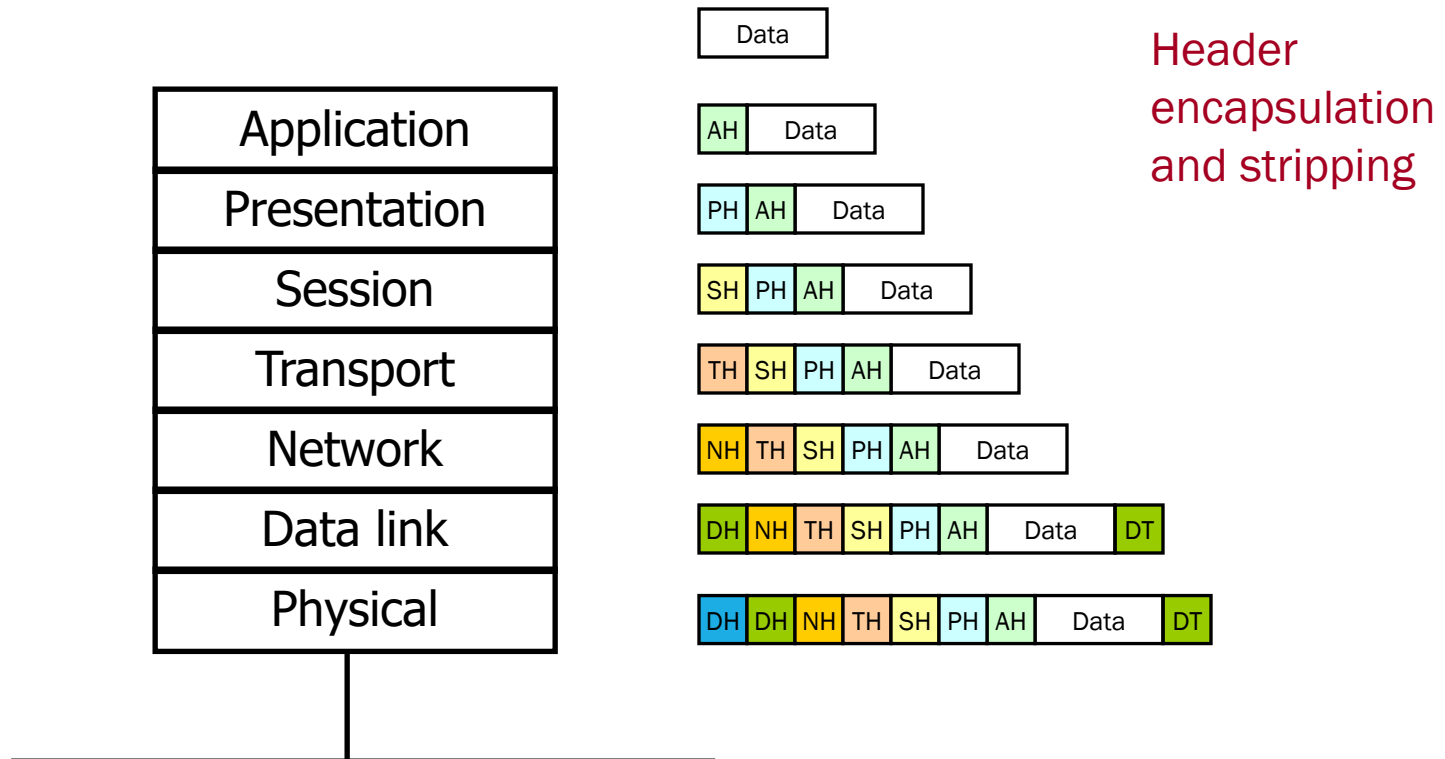
intermediate air traffic sites

airplane routing

airplane routing

airplane routing

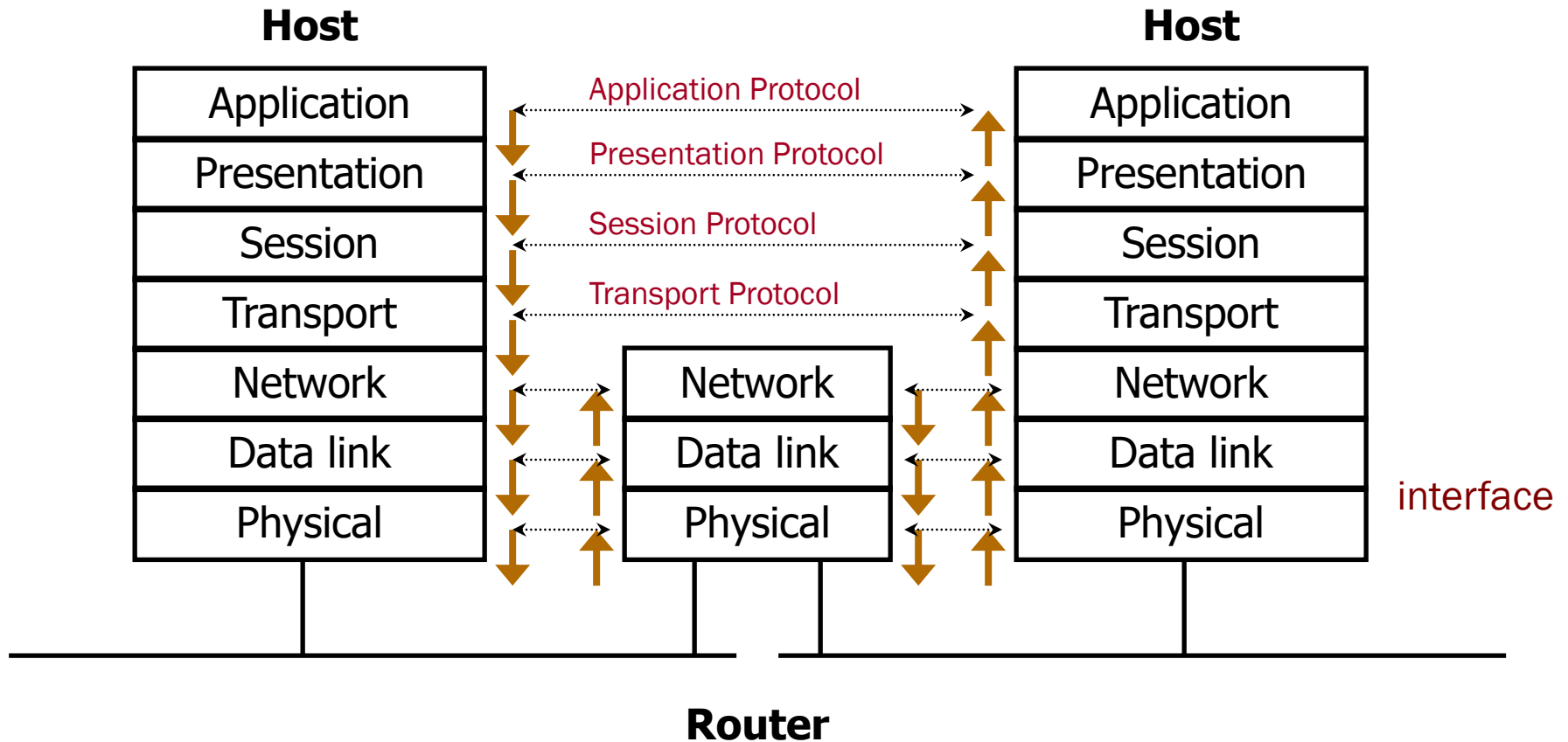
## PROTOCOL STACK: ISO OSI MODEL



ISO: the International Standards Organization

OSI: Open Systems Interconnection Reference Model (1984)

# COMMUNICATING BETWEEN END HOSTS



# PROTOCOLS EXAMPLE

Level Name	Layer Name	Example Protocol
Level 7	Application layer	FTP
Level 6	Presentation layer	XNS Xerox Network Systems
Level 5	Session layer	RPC
Level 4	Transport layer	TCP
Level 3	Network layer	IP
Level 2	Data-Link layer	Ethernet Frames
Level 1	Physical layer	Voltages



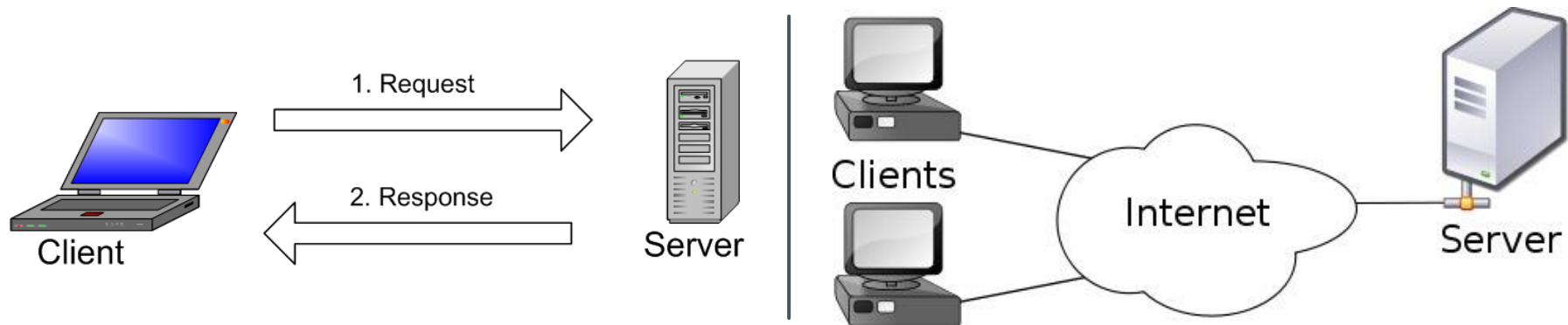
# INTEROPERABILITY

- Divide a task into pieces and then solve each piece independently (or nearly so).
- Establishing a well-defined **interface** between layers makes porting easier.
- Functions of each layer are **independent** of functions of other layers
  - Thus, each layer is like a module and can be developed independently
- Each layer builds on services provided by lower layers
  - Thus, no need to worry about details of lower layers - *transparent* to this layer
- Major Advantages:
  - Code Reuse
  - Eases maintenance, updating of system



## NETWORK PROGRAMS ARCHITECTURE

- Client/Server
  - **The server hosts, delivers and manages most of the resources and services to be consumed by the client.** This type of architecture has one or more client computers connected to a central server over a network or internet connection
- Peer to peer
  - **Networking architecture in which each workstation, or node, has the same capabilities and responsibilities**



## CLIENT - SERVER

- A server is a process - not a machine !
- A server waits for a request from a client.
- A client is a process that sends a request to an existing server and (usually) waits for a reply.

## CLIENT - SERVER EXAMPLES



Server returns the time-of-day.



Server returns a document (FTP server).



Server prints a file for client.



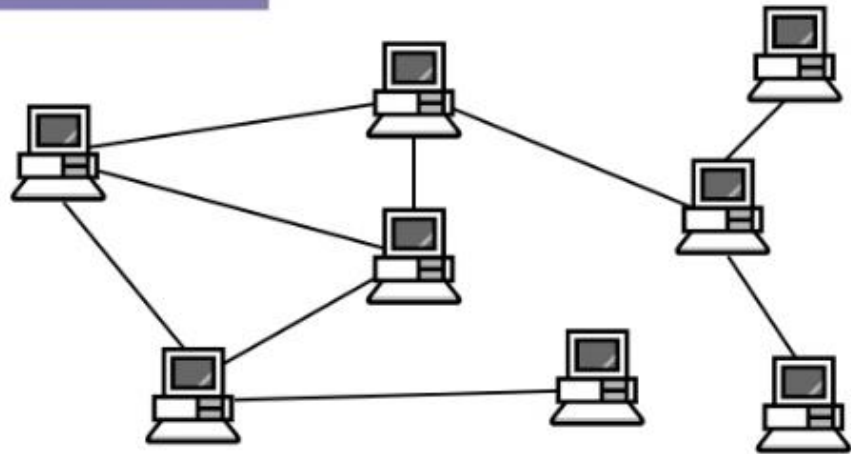
Server does a disk read or write.



Server records a transaction.

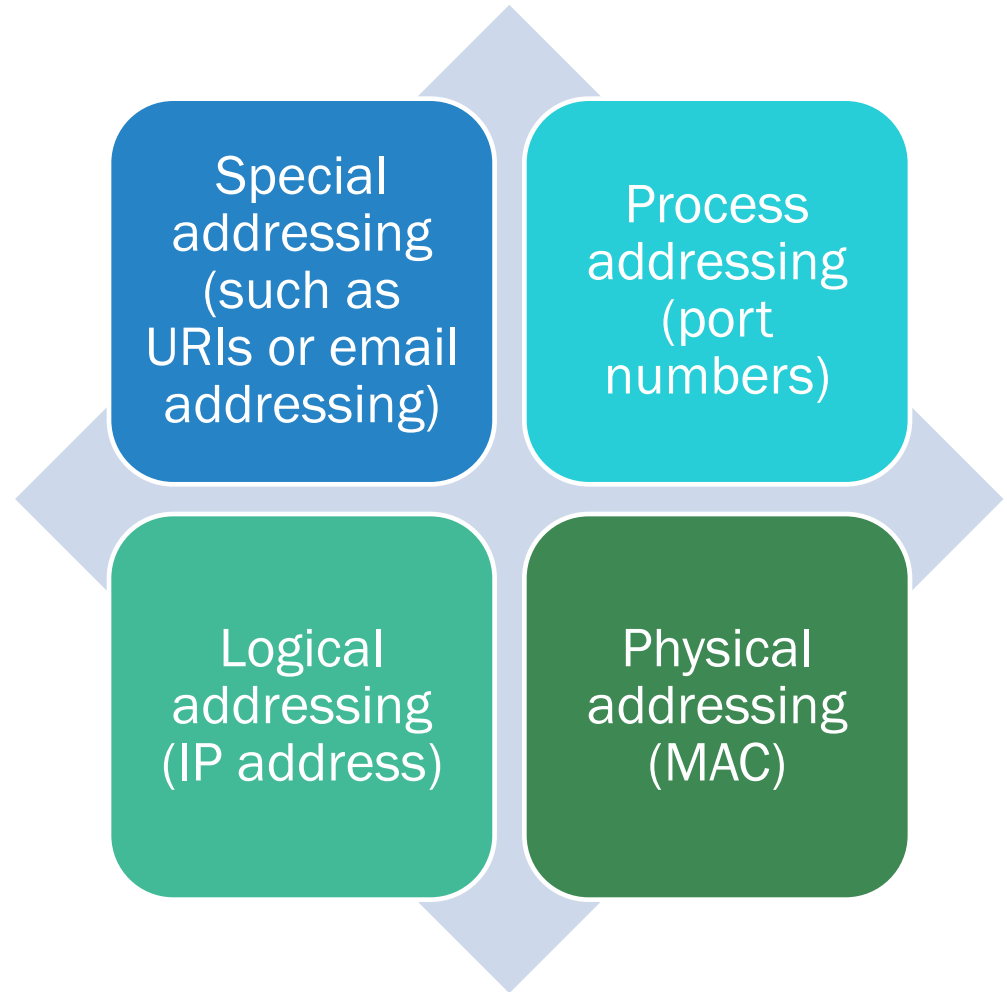
# PEER TO PEER

## Pure Peer-to-Peer Architecture



- No central server
- Clients connected to one or more peers
- Resilient
- Large #messages

## ADDRESSING





# **.NET AND C# FEATURES AND REVISION**

---

# PROGRAMS & PROCESSES

- A **program** is an executable file.
- A **process** or *task* is an instance of a program that is being executed.
- A **single program** can generate multiple processes.



## WHY C#.NET

- The Microsoft .NET Framework provides a layered, extensible, and **managed implementation of Internet services** that can be quickly and easily integrated into your applications.
- Your network applications can build on **pluggable** protocols to automatically take advantage of new Internet protocols, **or** they can use a managed implementation of the Windows socket interface to work with the network on the socket level.
- In addition to that you did take a visual C# course before.



## WHY C#.NET

The .NET framework provides two namespaces, System.Net and System.Net.Sockets for network programming.

The classes and methods of these namespaces help us to write programs, which can communicate across the network.

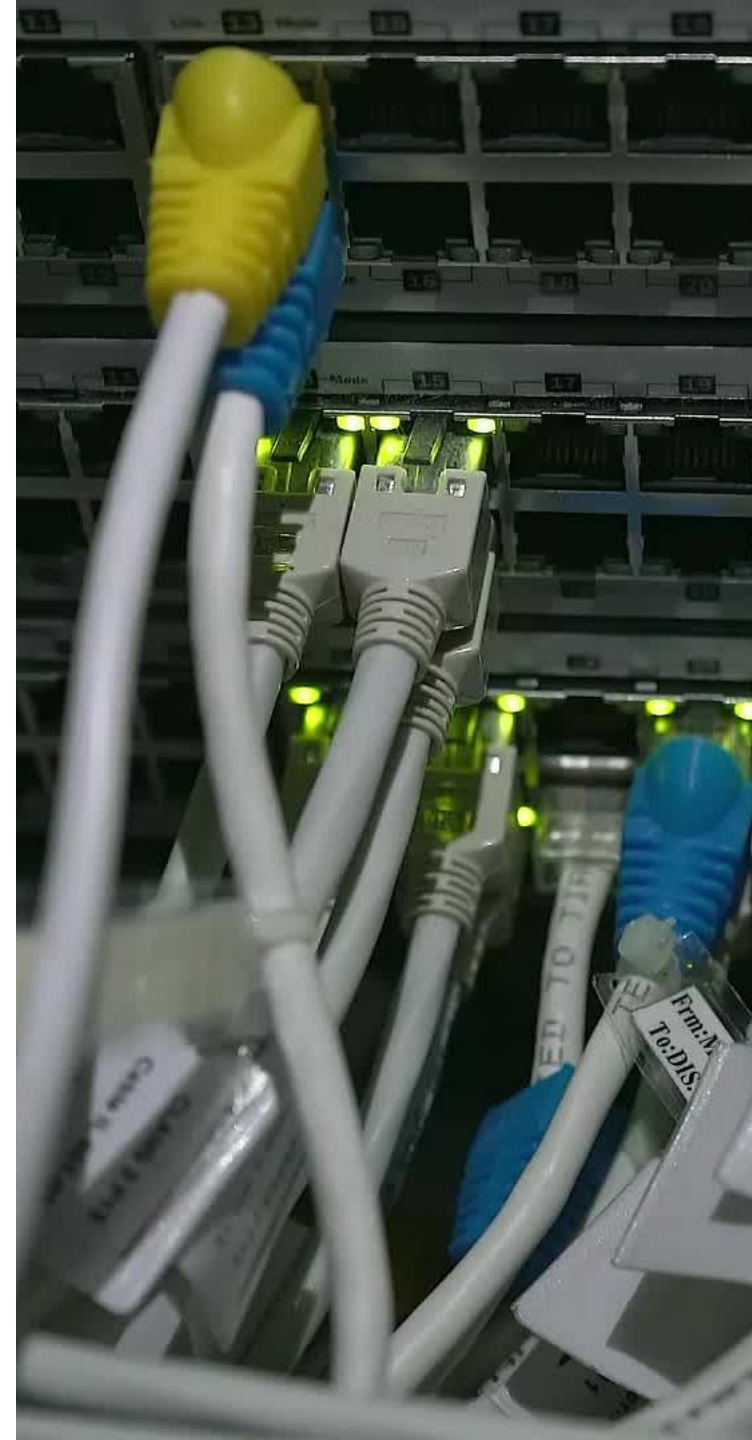
The communication can be either connection oriented or connectionless.

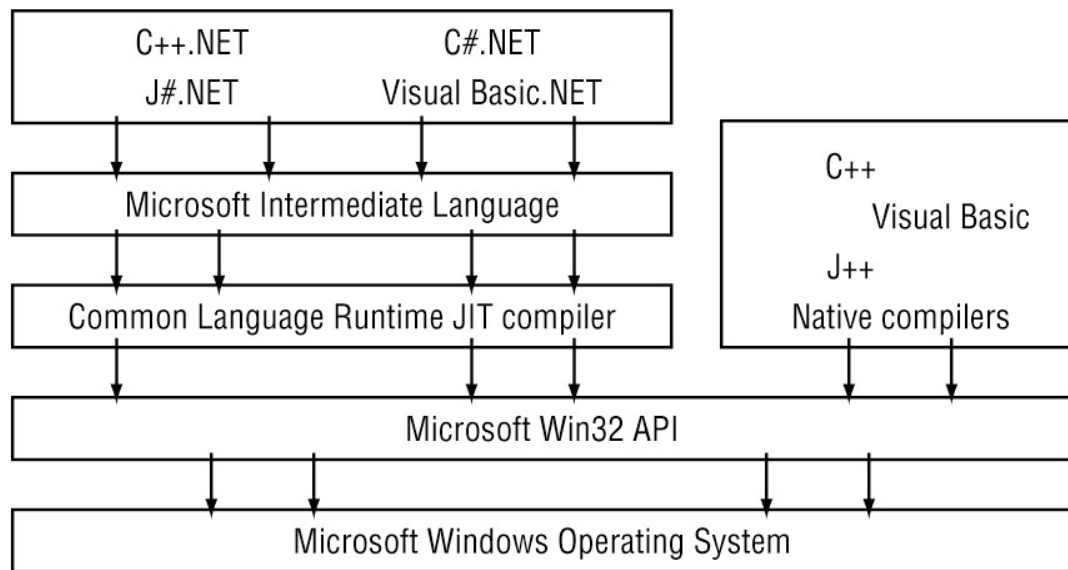
They can also be either stream oriented or datagram based.

The most widely used protocol is TCP which is used for stream-based communication and UDP is used for data-grams based applications.

## LOT OF EASY FEATURES

- There are some other helper classes like `IPEndPoint`, `IPAddress`, `SocketException` etc, which we can use for Network programming.
- The .NET framework supports both synchronous and asynchronous communication between the client and server.
- A synchronous method is operating in blocking mode, in which the method waits until the operation is complete before it returns.
- But an asynchronous method is operating in non-blocking mode, where it returns immediately, possibly before the operation has completed.





# **.NET MANAGED CODE EXECUTION SCENARIO**

## **THE COMMON LANGUAGE RUNTIME (CLR) ENVIRONMENT**



## **.NET “EXECUTABLE” FILE**

- Don't contain ready to run machine code but it has:
  - A *stub* assembly language program to start the CLR compiler
  - The MSIL code of the compiled application

# INSTALLING A C# DEVELOPMENT ENVIRONMENT

- Visual studio 2019 community edition

## Downloads



Version: 16.0  
[Release notes](#)


[Compare editions](#)  
[How to install offline](#)

### Visual Studio 2019

Full-featured integrated development environment (IDE) for Android, iOS, Windows, web, and cloud

#### Community


Powerful IDE, free for students, open-source contributors, and individuals

[Free download](#) 

[Download Preview](#) >

#### Professional


Professional IDE best suited to small teams

[Free trial](#) 

[Download Preview](#) >

#### Enterprise

Scalable, end-to-end solution for teams of any size

[Free trial](#) 

[Download Preview](#) >



# LAB1

- Please do Sheet 1 with your TA.



# C# STREAMS

- The Stream Concept
- System.IO namespace
  - Identify the stream classes
  - Types of streams
  - File streams
  - Memory streams



## THE STREAM CONCEPT

- A **stream** is a flow of data from a program to a backing store, or from a backing store to a program.
- The program can either write to a stream or read from a stream.





## STREAMS AND STREAM PROCESSING

- Reading from or writing to files in secondary memory (disk)
- Reading from or writing to primary memory (RAM)
- Connection to the Internet
- Socket connection between two programs

# SYSTEM.IO NAMESPACE

- Contains types that allow reading and writing to files and data streams, and types that provide basic file and directory support.

CLASSES	
<u>BinaryReader</u>	Reads primitive data types as binary values in a specific encoding.
<u>BinaryWriter</u>	Writes primitive types in binary to a stream and supports writing strings in a specific encoding.
<u>Directory</u>	Exposes static methods for creating, moving, and enumerating through directories and subdirectories. This class cannot be inherited.
<u>File</u>	Provides static methods for the creation, copying, deletion, moving, and opening of a single file, and aids in the creation of <u>FileStream</u> objects.
<u>FileStream</u>	Provides a <u>Stream</u> for a file, supporting both synchronous and asynchronous read and write operations.
<u>FileStreamOptions</u>	Defines a variety of configuration options for <u>FileStream</u> .

<u>IOException</u>	The exception that is thrown when an I/O error occurs.
<u>MemoryStream</u>	Creates a stream whose backing store is memory.
<u>Path</u>	Performs operations on <u>String</u> instances that contain file or directory path information. These operations are performed in a cross-platform manner.
<u>Stream</u>	Provides a generic view of a sequence of bytes. This is an abstract class.
<u>StreamReader</u>	Implements a <u>TextReader</u> that reads characters from a byte stream in a particular encoding.
<u>StreamWriter</u>	Implements a <u>TextWriter</u> for writing characters to a stream in a particular encoding.
<u>StringReader</u>	Implements a <u>TextReader</u> that reads from a string.
<u>StringWriter</u>	Implements a <u>TextWriter</u> for writing information to a string. The information is stored in an underlying <u>StringBuilder</u> .
<u>TextReader</u>	Represents a reader that can read a sequential series of characters.
<u>TextWriter</u>	Represents a writer that can write a sequential series of characters. This class is abstract.

# The Abstract class *Stream* in C#

- The abstract class *Stream* is the most general stream class in C#
- Provides a generic view on data sources and data destinations
- Isolates the programmer from operating system details
- Offers reading and writing of raw, binary data
- Chunked and sequenced as bytes
- No char encoding is involved
- **Synchronous** reading and writing
  - Read and Write transfer byte arrays
  - A subclass must implement the operations Read and Write
- **Asynchronous** reading and writing -
  - BeginRead and EndRead
  - BeginWrite and EndWrite
  - Rely on Read and Write
- A stream can be queried for its capabilities
  - CanRead, CanWrite, CanSeek

## THE MOST IMPORTANT MEMBERS IN CLASS STREAM

- **int *Read*** (byte[] buf, int pos, int len)
- **int *ReadByte***()
- **void *Write*** (byte[] buf, int pos, int len)
- **void *WriteByte***(byte b)
- **bool *CanRead***
- **bool *CanWrite***
- **bool *CanSeek***
- **long *Length***
- **void *Seek*** (long offset, SeekOrigin org)
- **void *Flush*** ()
- **void *Close***()

## NON-ABSTRACT SUBCLASSES OF *STREAM*

- `System.IO.FileStream`
  - Provides a stream backed by a **file** from the operating system
- `System.IO.BufferedStream`
  - Encapsulates buffering around another stream
- `System.IO.MemoryStream`
  - Provides a stream backed by RAM memory
- **`System.Net.Sockets.NetworkStream` (this is ours !!)**
  - **Encapsulates a socket connection as a stream**
- `System.IO.Compression.GZipStream`
  - Provides stream access to compressed data
- `System.Security.Cryptography.CryptoStream`



**ITS TIME TO CODE !**

- Be ready for some examples