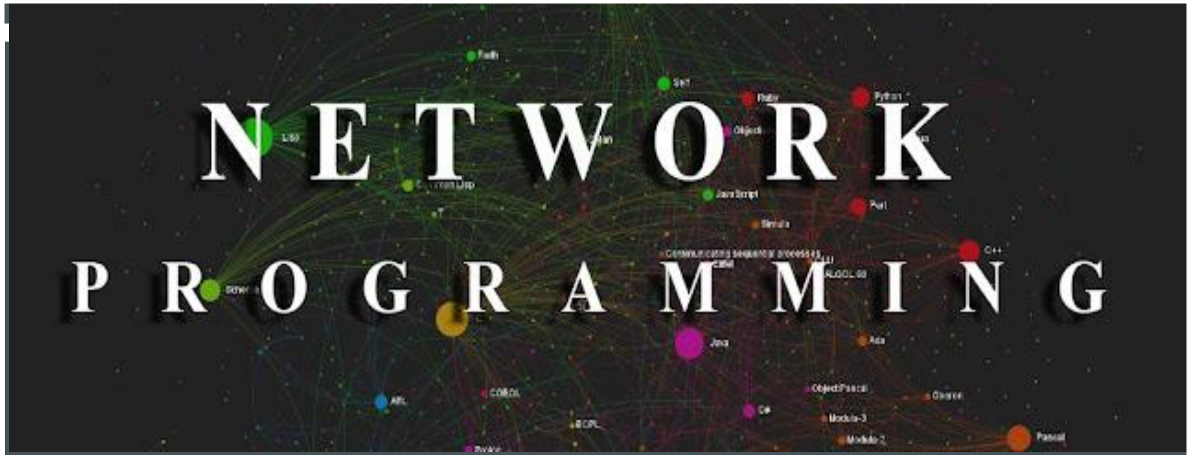


IT432 - NETWORK PROGRAMMING

DR. ALI HUSSEIN AHMED

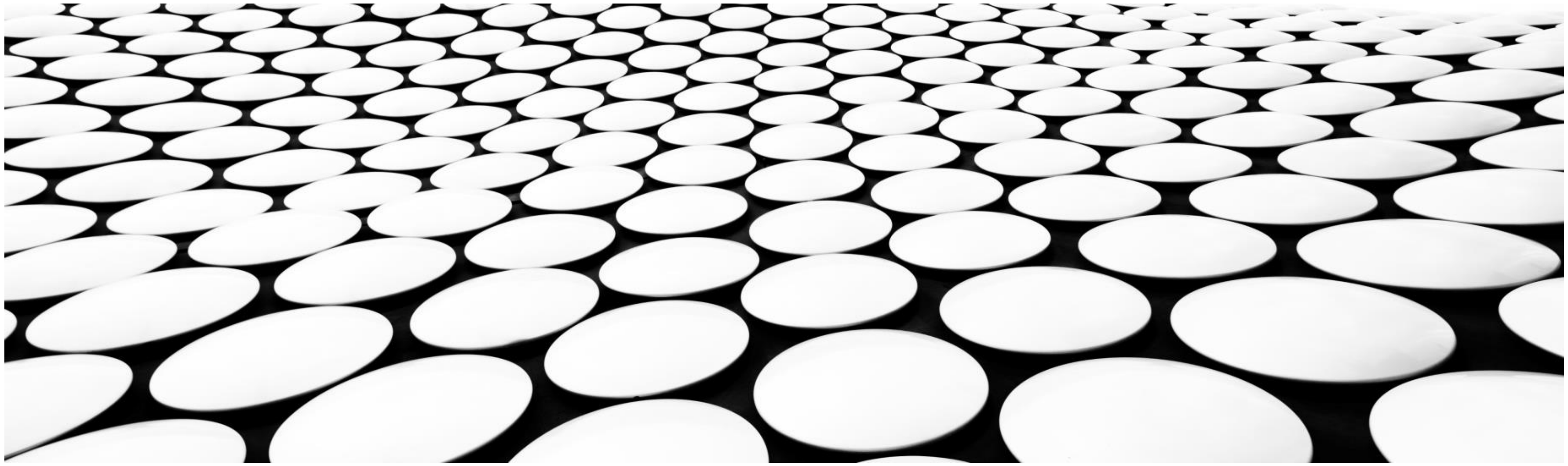
ALI.HUSSEIN@AUN.EDU.EG

IT DEPT - OFFICE NO. 312

2022

كلية معتمدة من الهيئة القومية
لضمان الجودة والاعتماد

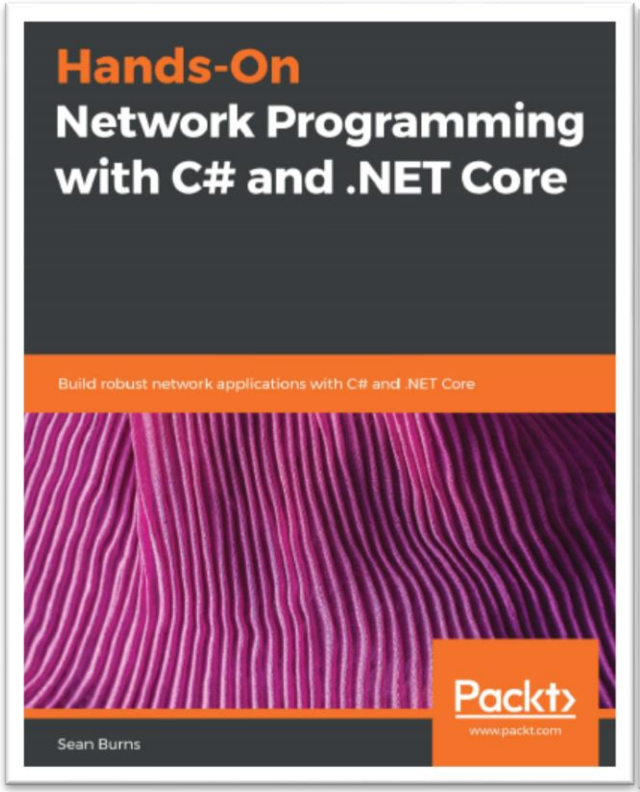
LECTURE 0



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

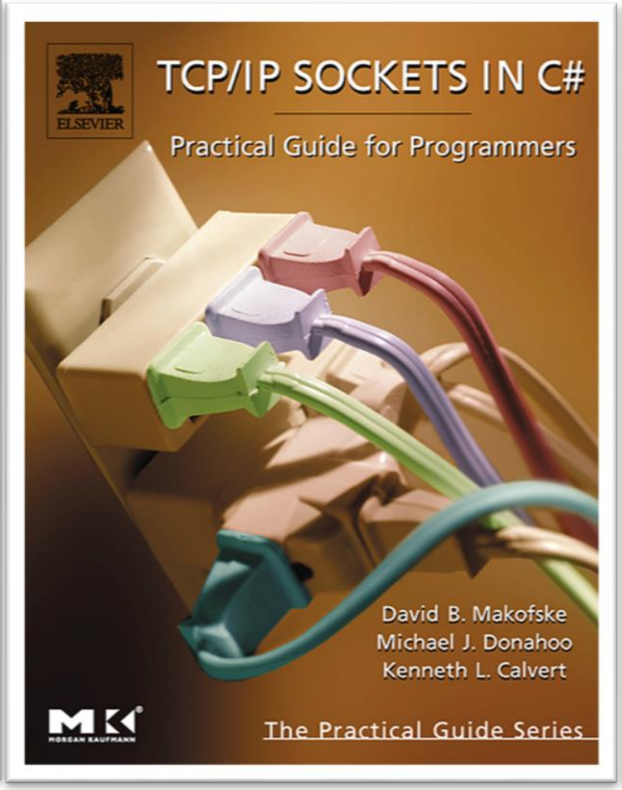


Hands-On
Network Programming with C# and .NET Core

Build robust network applications with C# and .NET Core

Sean Burns

Packt
www.packt.com



ELSEVIER

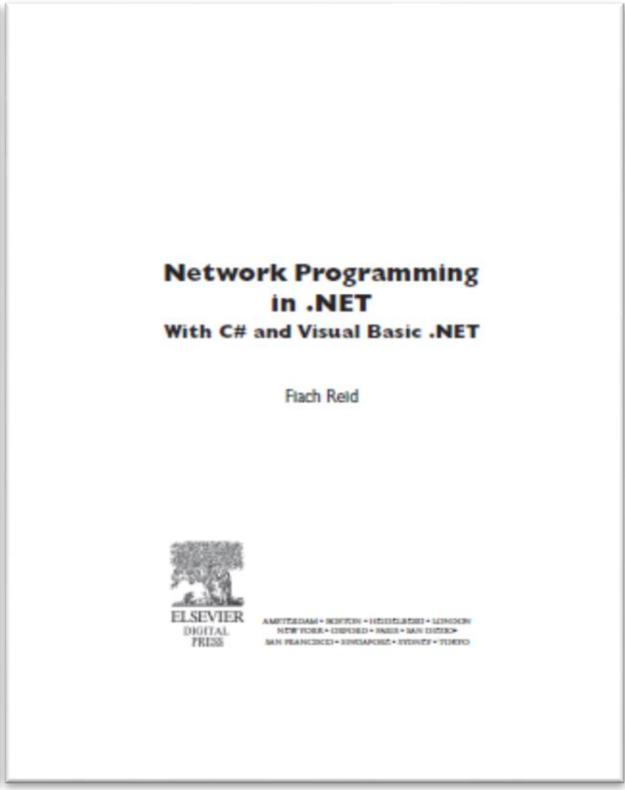
TCP/IP SOCKETS IN C#

Practical Guide for Programmers

David B. Makofske
Michael J. Donahoo
Kenneth L. Calvert

MORGAN KAUFMANN

The Practical Guide Series



Network Programming
in .NET
With C# and Visual Basic .NET

Fiach Reid

ELSEVIER
DIGITAL PRESS

AMSTERDAM • BOSTON • HEIDELBERG • LONDON
NEW YORK • OXFORD • PARIS • SAN DIEGO
SAN FRANCISCO • SINGAPORE • SYDNEY • TOKYO

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 crowed machine
- High reliability
  - Alternative sources for Info.

WIDE VARIETY TYPES OF NETWORKS

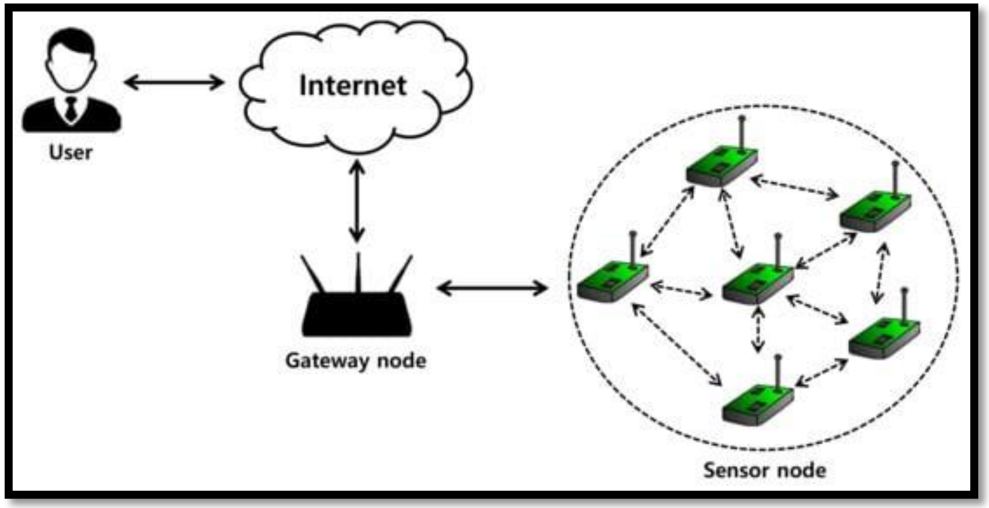
- Circuit switched
  - dedicated circuit per call
  - performance (guaranteed)
  - call setup required
  - telephone system
- Packet switched:
  - data sent through the net in discrete "chunks"
  - user A, B packets *share* network resources
  - resources used *as needed*
  - store and forward: packets 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



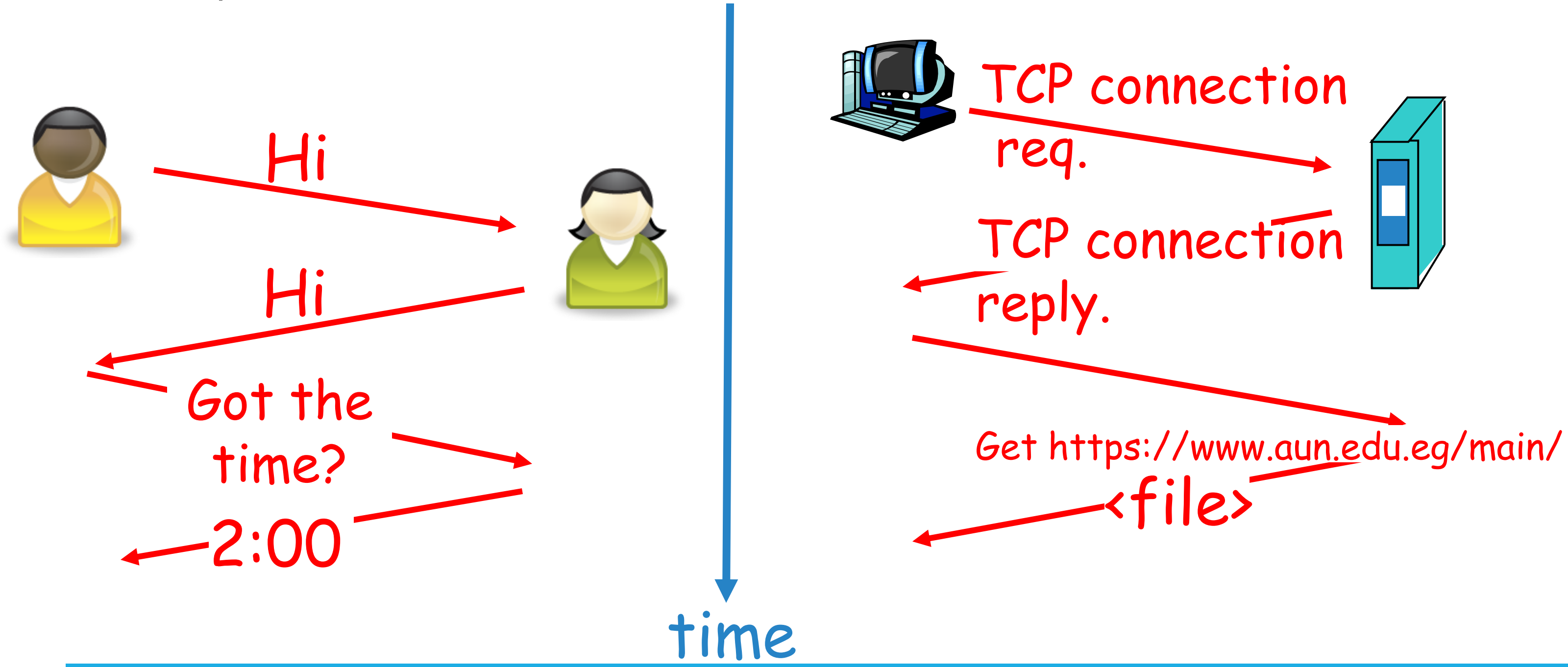
LAN & WAN

- 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:

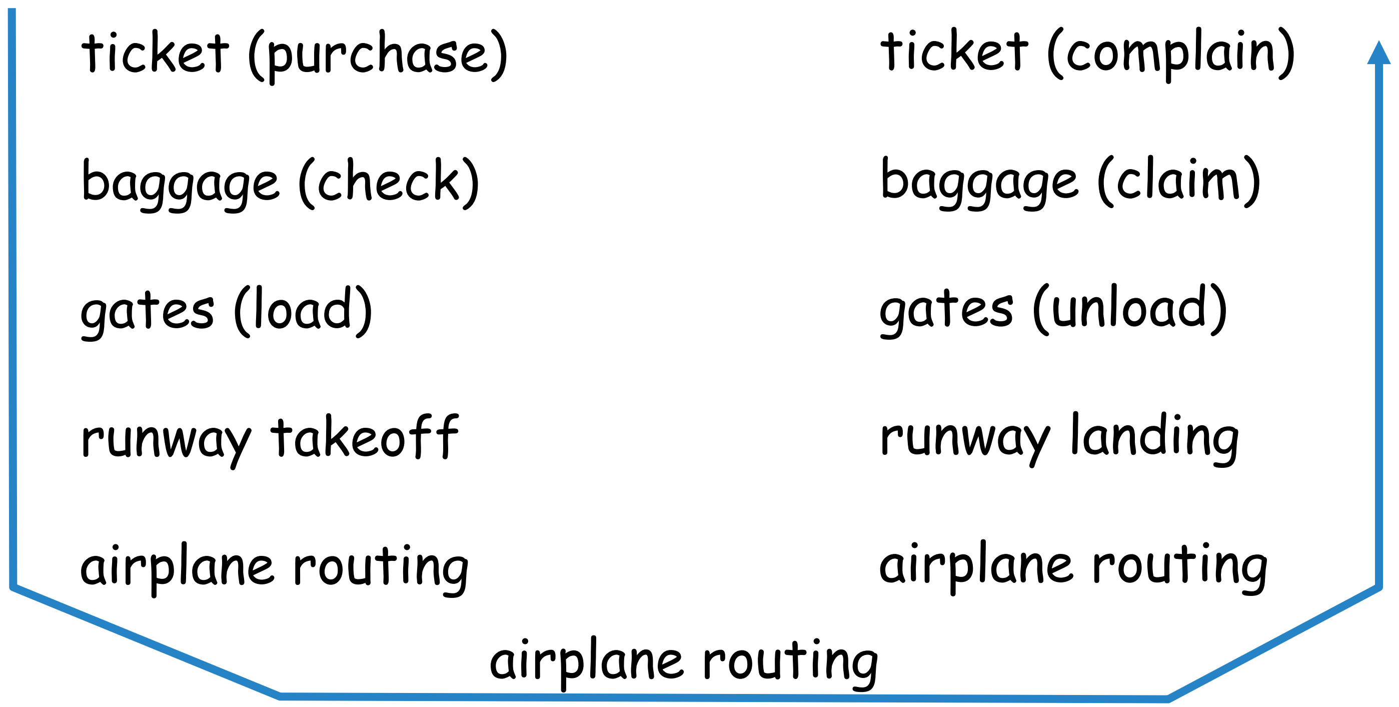


protocols define syntax , semantics and timing information required for entities to communicate (understand each other)

LAYERING

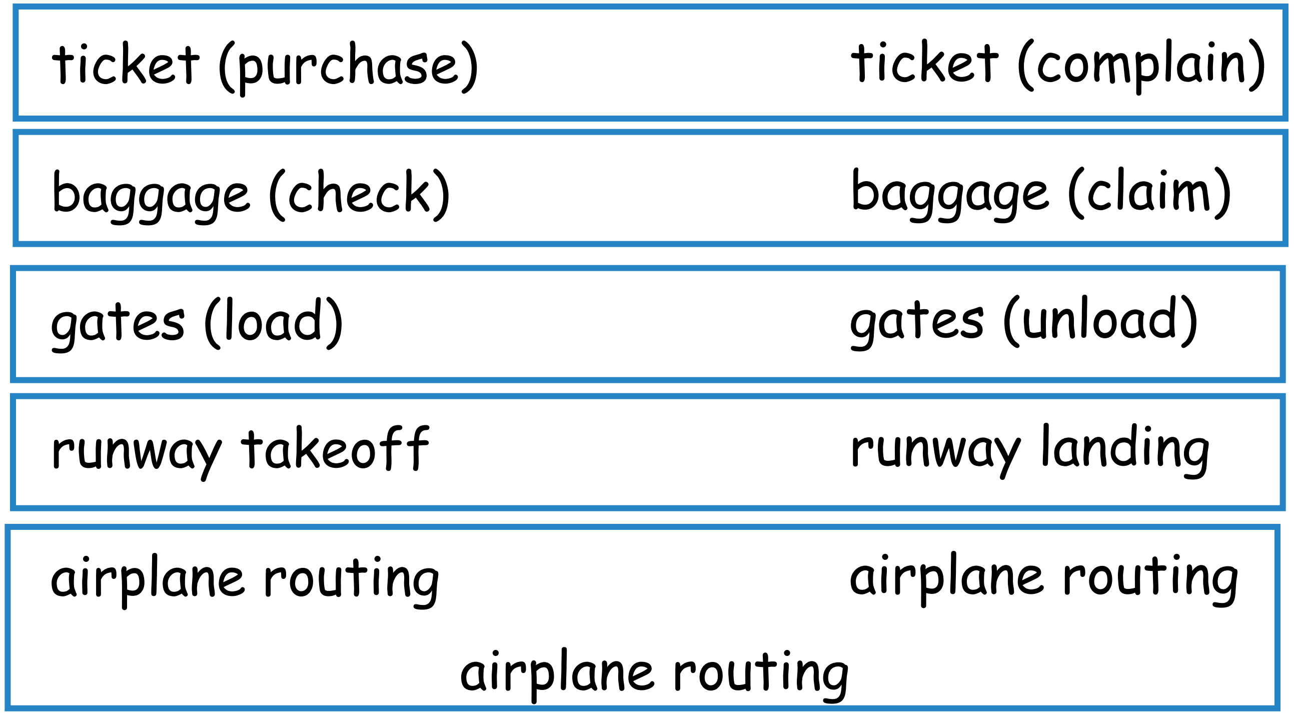
- Lot of functions needs to be organized
- Communicating entities are widely distributes (inherits heterogeneity)
- How to manage and enhances ?
  - Categorized function groups (services) into disjoint sets
  - Assign every group to set of similar entities
  - Each entity can perform service to other (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.

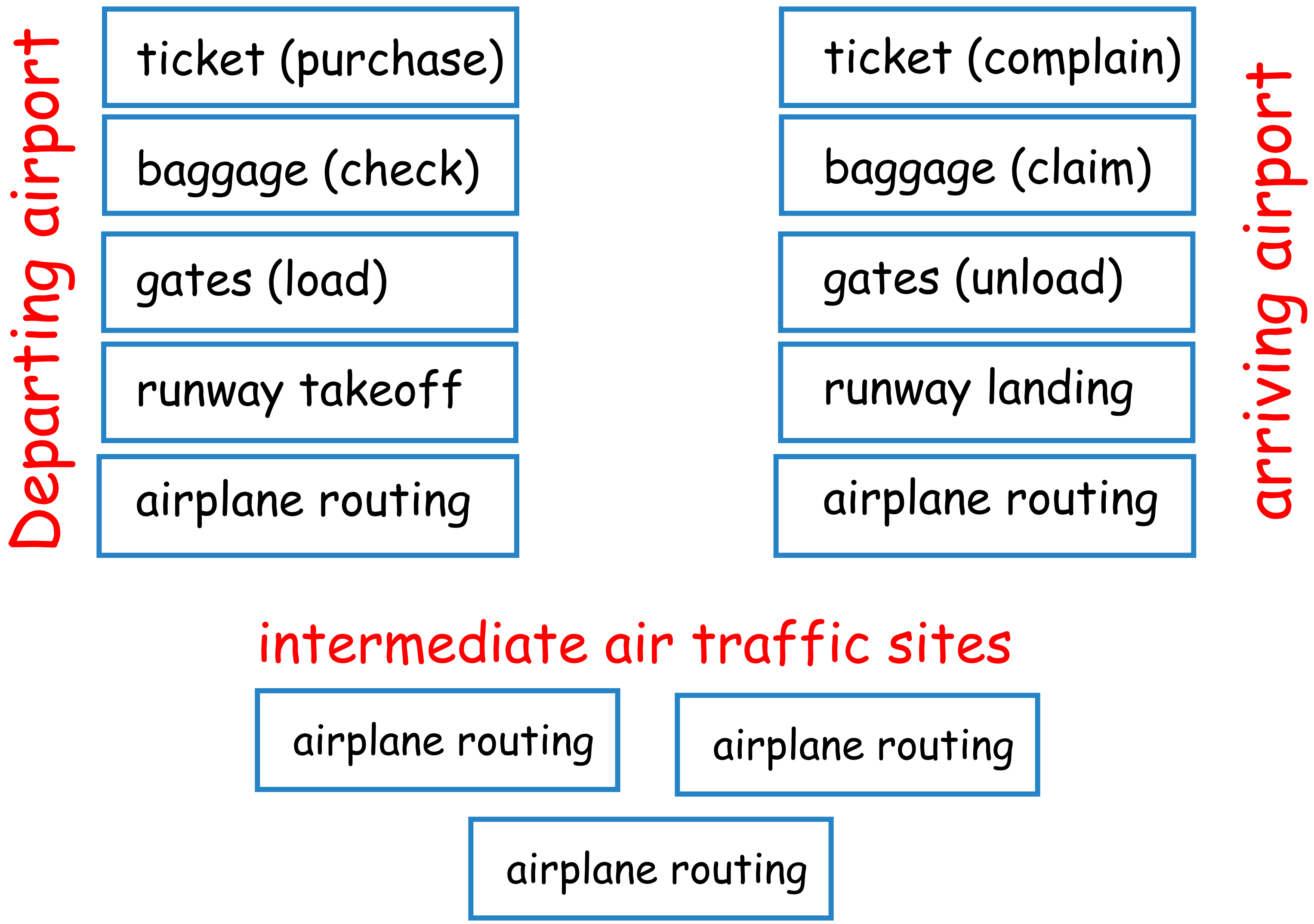
ORGANIZATION OF AIR TRAVEL: PEER LAYER VIEW



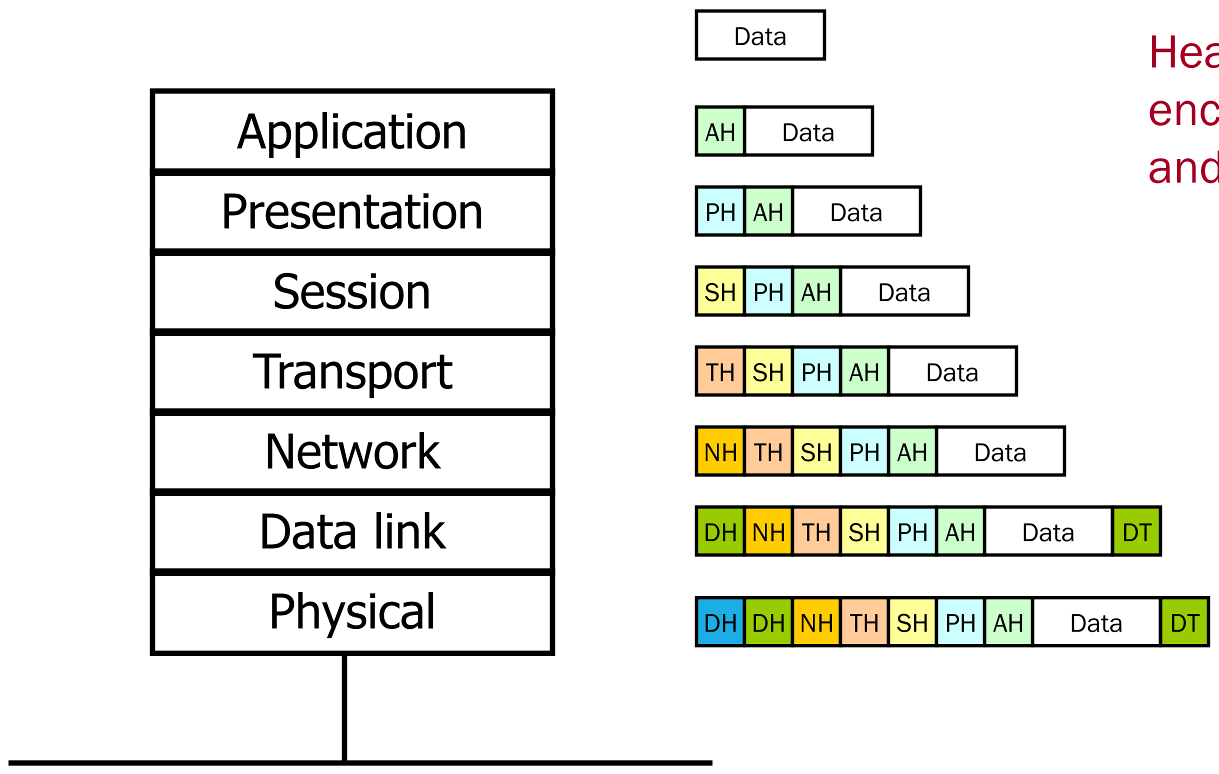
Layers: each layer implements a service

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

DISTRIBUTED IMPLEMENTATION OF LAYER FUNCTIONALITY



PROTOCOL STACK: ISO OSI MODEL

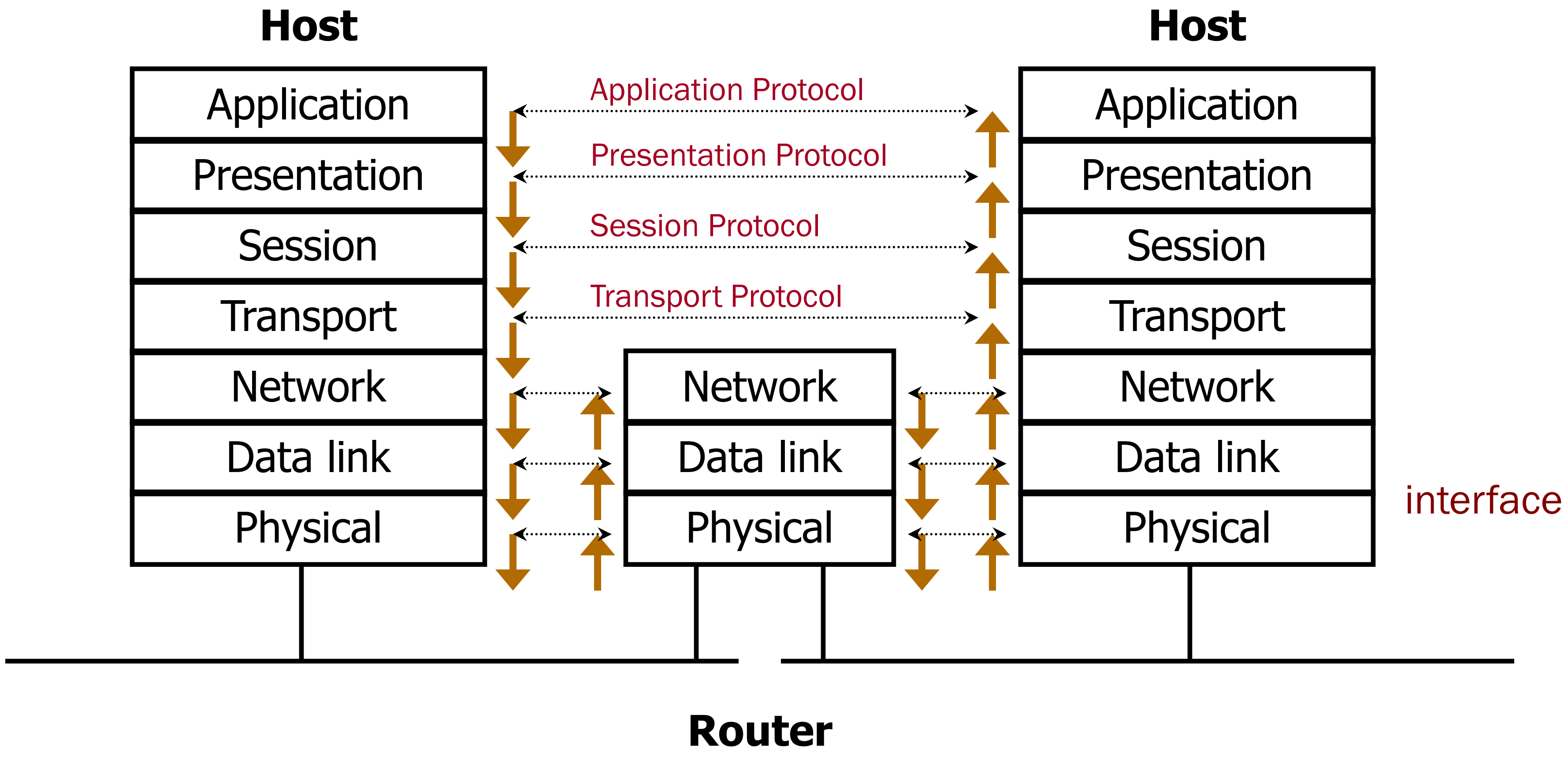


Header encapsulation and stripping

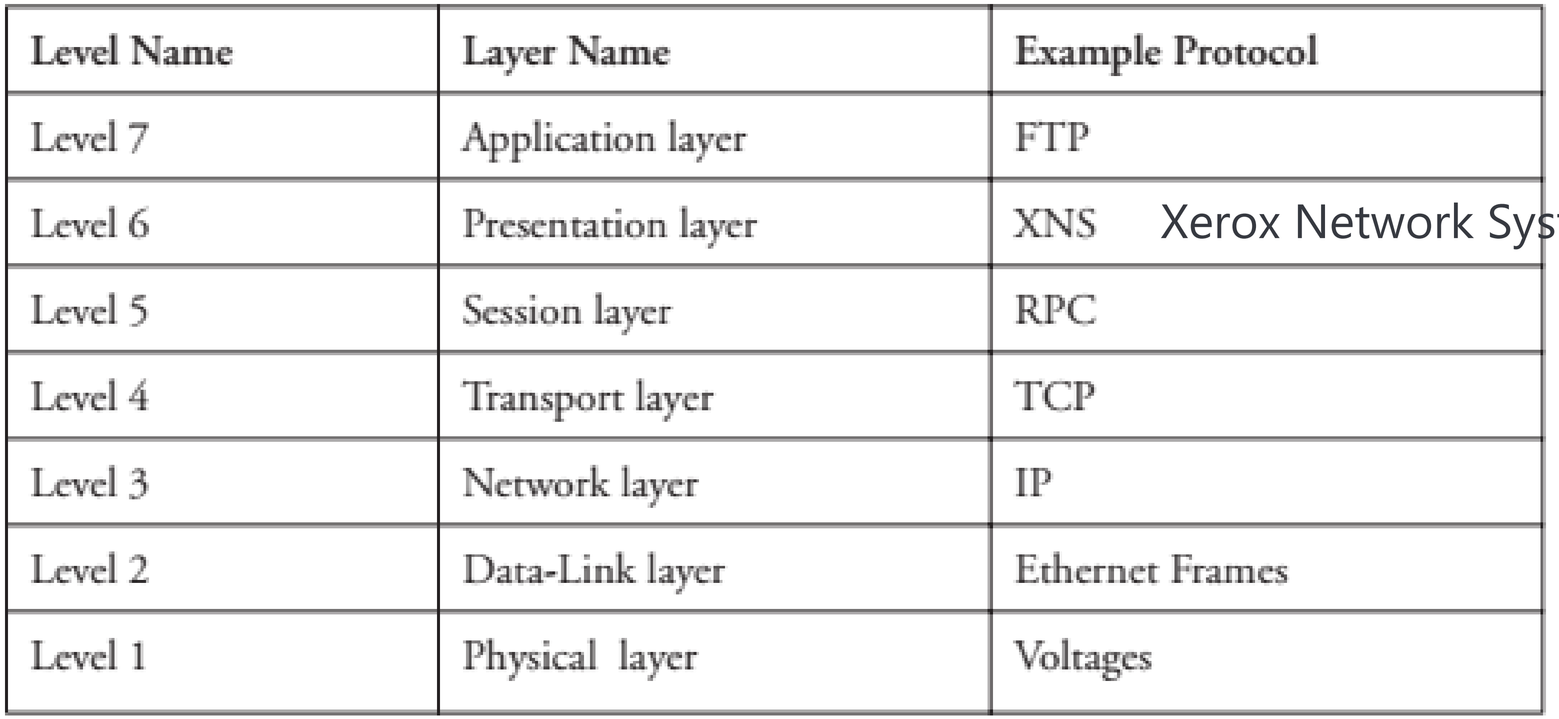
ISO: the International Standards Organization

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
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

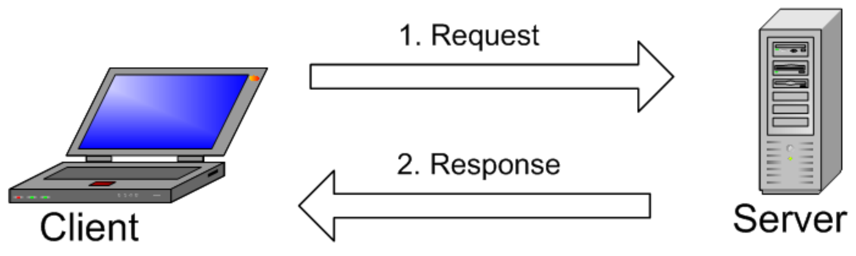
Xerox Network Systems

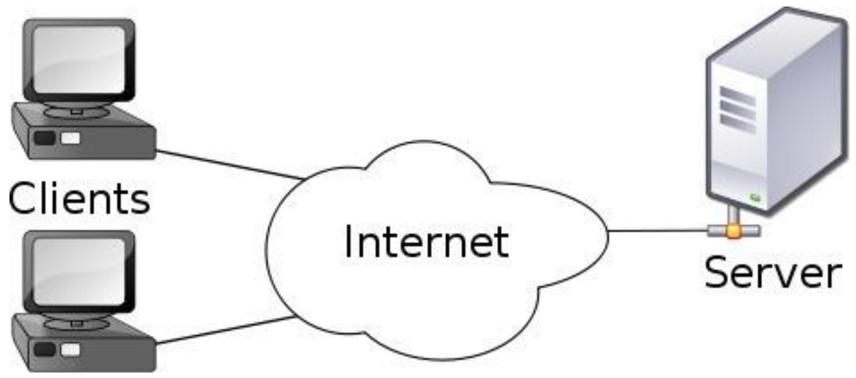
INTEROPERABILITY

- 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**





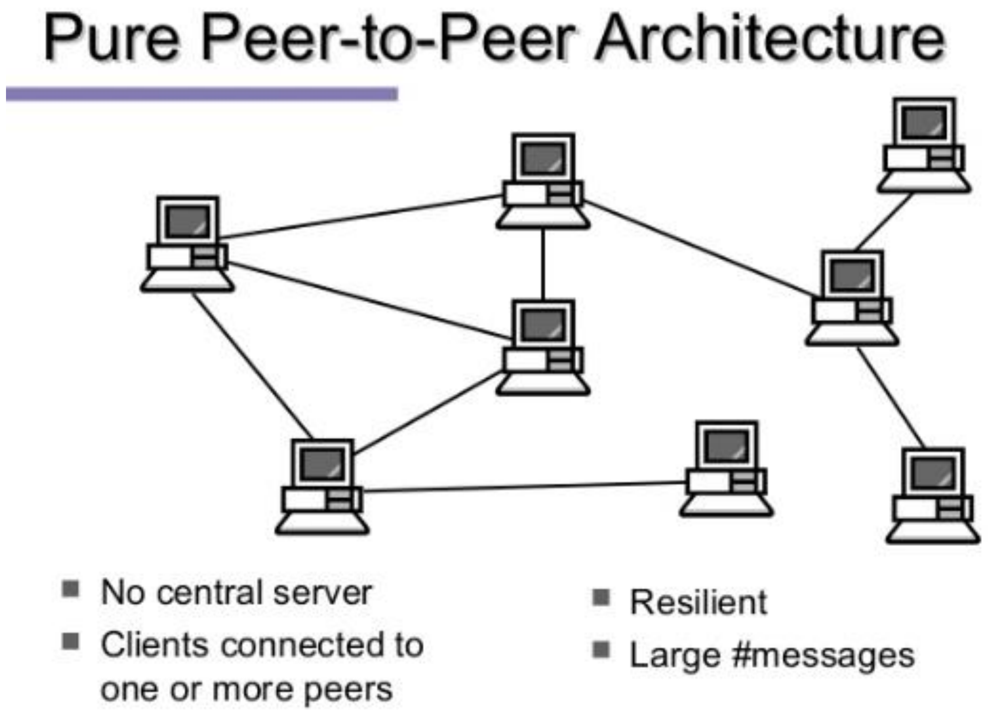
CLIENT - SERVER

- 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.

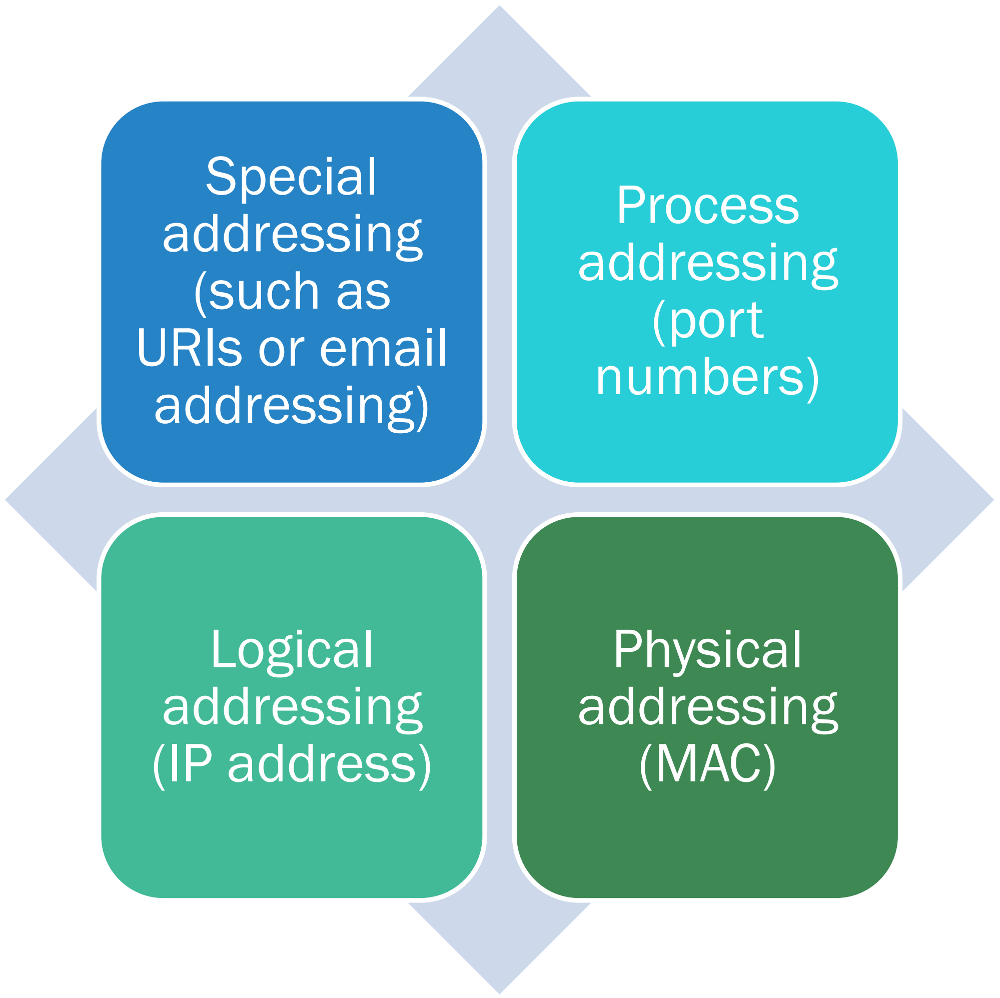
PEER TO PEER



Pure Peer-to-Peer Architecture

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

ADDRESSING


Special addressing (such as URIs or email addressing)
Process addressing (port numbers)
Logical addressing (IP address)
Physical addressing (MAC)

.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.

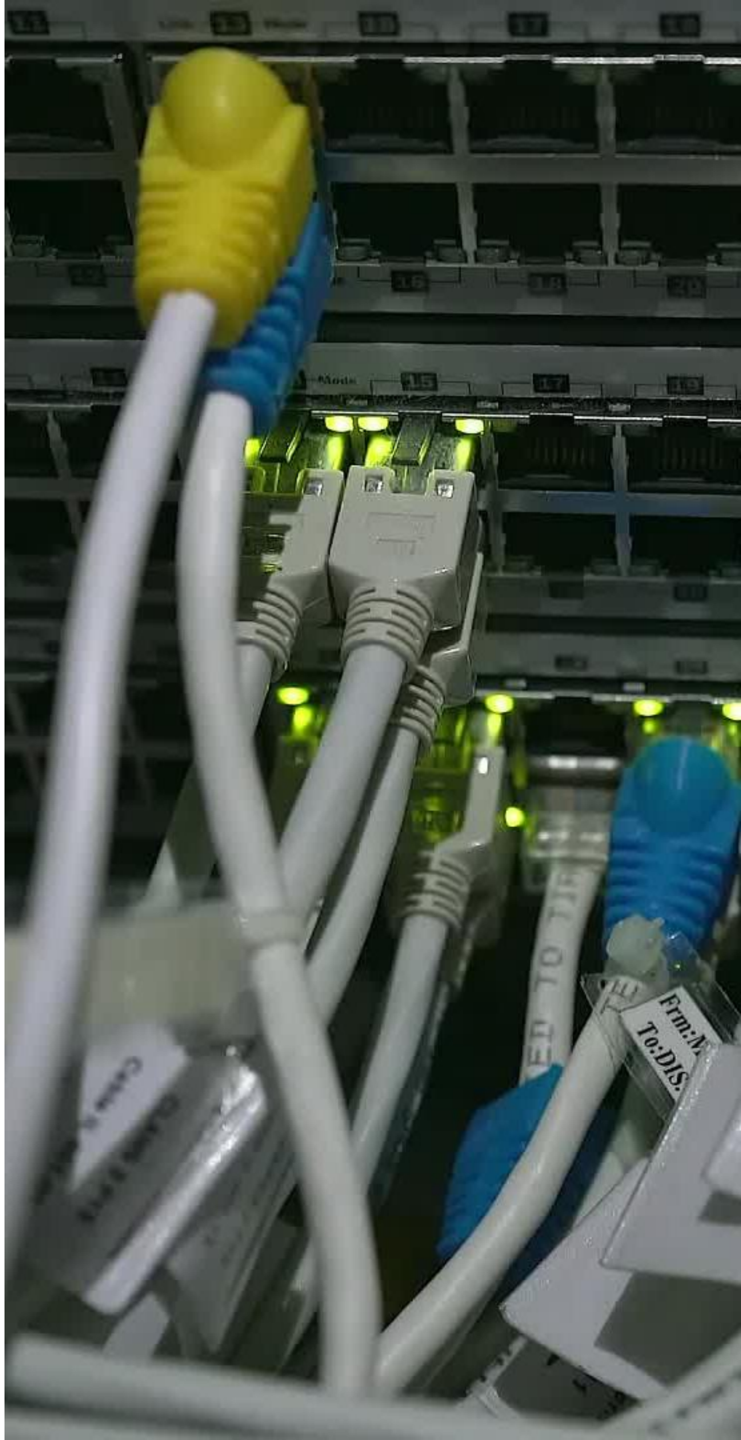
The communication can be either connection oriented or connectionless.

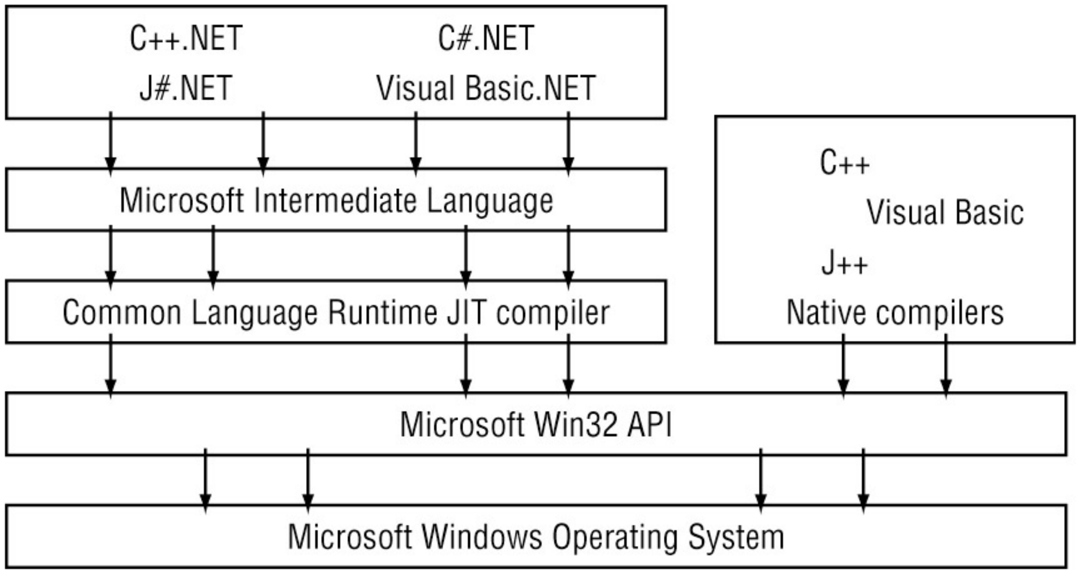
They can also be either stream oriented or datagram based.

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.




C++.NET
C#.NET
J#.NET
Visual Basic.NET
Microsoft Intermediate Language
Common Language Runtime JIT compiler
C++
Visual Basic
J++
Native compilers
Microsoft Win32 API
Microsoft Windows Operating System

.NET MANAGED CODE EXECUTION SCENARIO

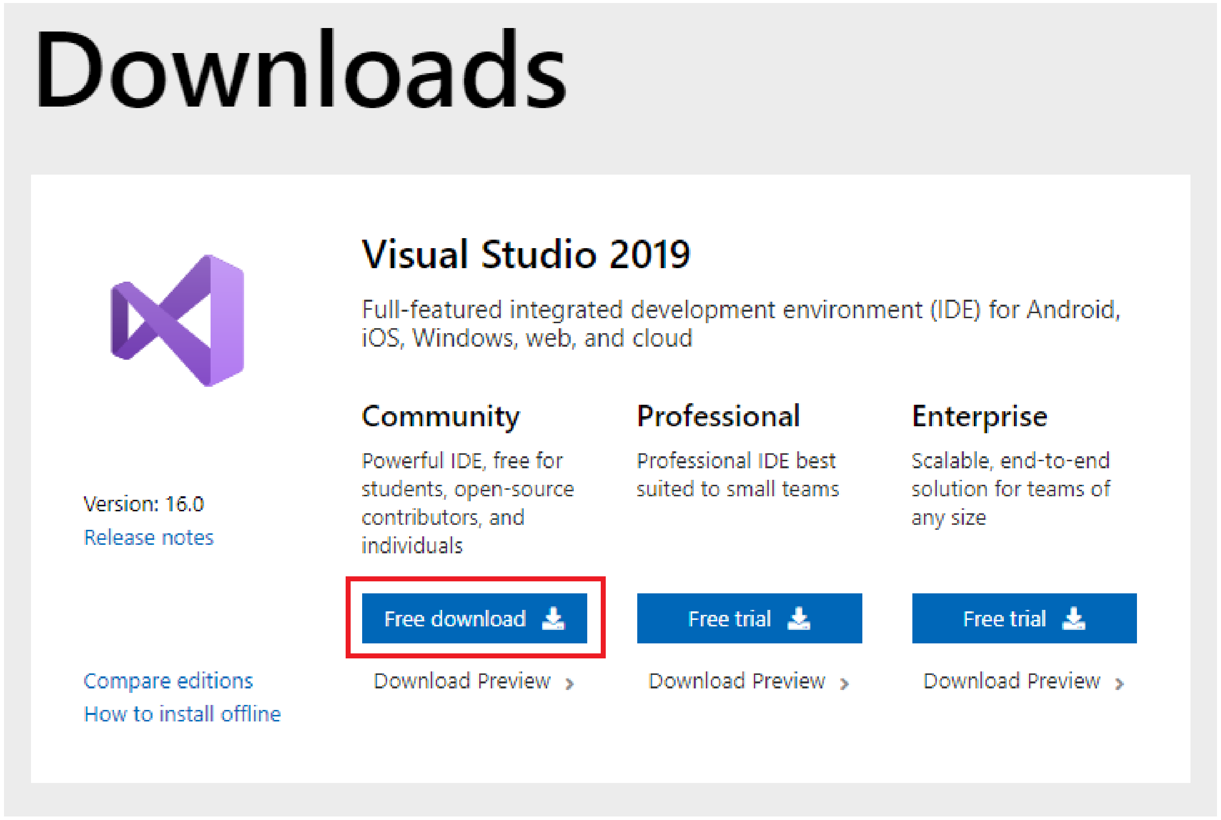
THE COMMON LANGUAGE RUNTIME (CLR) ENVIRONMENT

.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

Visual Studio 2019

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

Version: 16.0
Release notes

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

Professional
Professional IDE best suited to small teams

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

LAB1

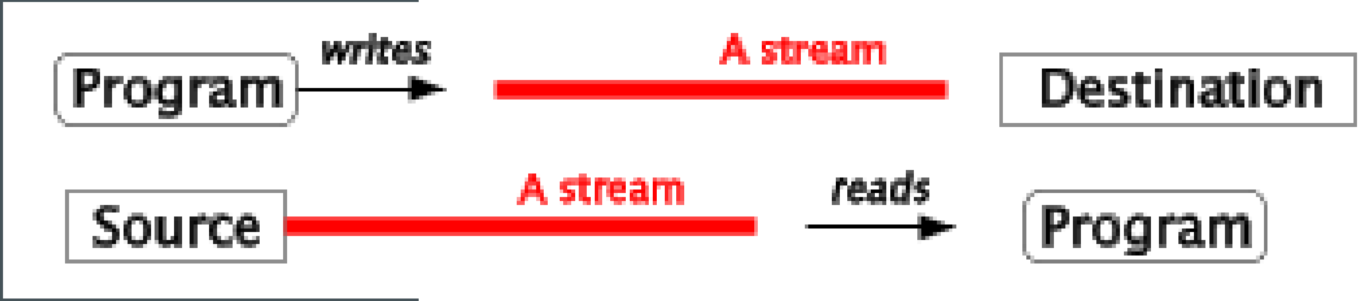
- Please do Sheet 1 with your TA.

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

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

http://people.cs.aau.dk/~normark/oop-csharp/html/notes/io-slide-stream-abstract-class.html

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()**

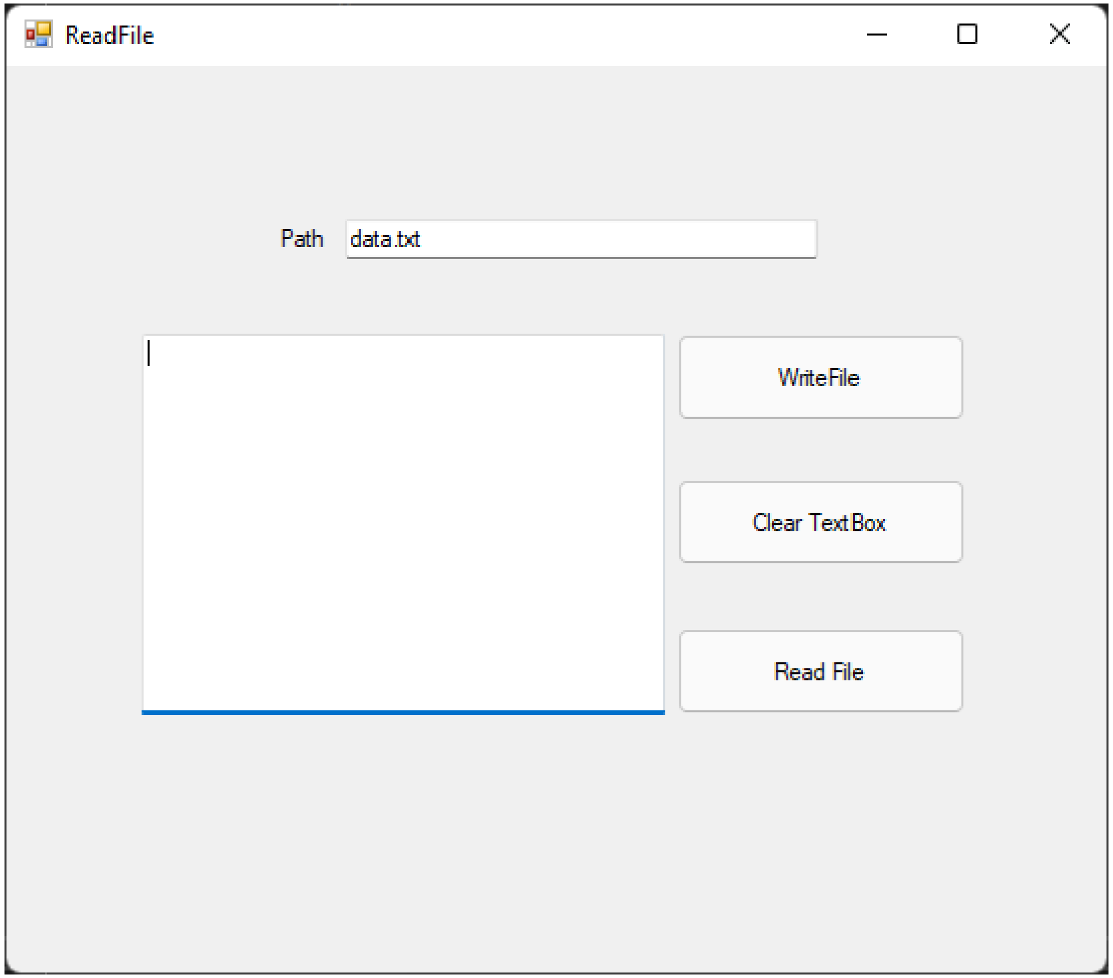
NON-ABSTRACT SUBCLASSES OF *STREAM*

- 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

EXAMPLE: FILESTREAMS (READ AND WRITE)



Our GUI

WRITE 3 CHARS

```
private void button2_Click(object sender, EventArgs e)
        {
            Stream s = new FileStream(txtPath.Text, FileMode.Create);
            s.WriteByte(79);  // O    01001111
            s.WriteByte(79);  // O    01001111
            s.WriteByte(80);  // P    01010000
            s.Close();
        }
```

READ MORE THAN 3 CHARS

```
private void button1_Click(object sender, EventArgs e)
        {
            Stream s = new FileStream(txtPath.Text, FileMode.Open);
            int i, j, k, m, n;
            i = s.ReadByte();  // O   79  01001111
            j = s.ReadByte();  // O   79  01001111
            k = s.ReadByte();  // P   80  01010000

            m = s.ReadByte();  // -1  EOF
            n = s.ReadByte();  // -1  EOF

            txtData.Text = String.Format("{0} {1} {2} {3} {4}", i, j, k, m, n);
            s.Close();
        }
```

LAB

- Extend the code to read and write any number of chars, make the same previous GUI functional (problem 1 sheet2)

NOTE

Class **FileStream** offers binary input and output

The **using** control structure is helpful when we do IO programming

using (*type variable* = *initializer*) *body*

- Semantics
    - In the scope of **using**, bind *variable* to the value of *initializer*
    - The *type* must implement the interface **IDisposable**
    - Execute *body* with the established name binding
    - At the end of *body* do *variable*.Dispose
        - The Dispose methods in the subclasses of Stream call Close

SYNTAX

```
using (Stream s = new FileStream("myFile.txt", FileMode.Create))
                    {
                         s.WriteByte(79);   // O    01001111
                         s.WriteByte(79);   // O    01001111
                         s.WriteByte(80);   // P    01010000
                    }
```

WHY USING

- // The using statement …
- using (type variable = initializer)
- body

- {type variable = initializer;
-   try {
-     body
-   }
-   finally {
-     if (variable != null)
-       ((IDisposable)variable).Dispose();
-   }
- }

CONSOLE APPLICATION TO COPY FILES

- This example copies the content of a file to another.
- A console application, files names are passed as a command line arguments.
  - **`copy-file source-file.txt target-file.txt`**
- During your lab, convert it to WinForm APP
- Make your GUI Attractive, Plz

```
using System;
using System.IO;
public class CopyApp
{
    public static void Main(string[] args) {
        FileCopy(args[0], args[1]);
    }
    public static void FileCopy(string fromFile, string toFile) {
        try {
            using (FileStream fromStream =  new FileStream(fromFile, FileMode.Open))
            {
                using (FileStream toStream = new FileStream(toFile, FileMode.Create))
                {
                    int c;
                    do       {
                        c = fromStream.ReadByte();
                        if (c != -1) toStream.WriteByte((byte)c);
                    } while (c != -1);
                }   }   }
        catch (FileNotFoundException e)
        {
            Console.WriteLine("File {0} not found: ", e.FileName);
            throw;
        }
        catch (Exception)
        {
            Console.WriteLine("Other file copy exception");
            throw;
        }  }  }
```

ADAPTER CLASSES

- A set of classes provided by FCL to support reading and writing in higher level

	Input	Output
Text	*TextReader* StreamReader StringReader	*TextWriter* StreamWriter StringWriter
Binary	BinaryReader	BinaryWriter

ADAPTER CLASSES

- Higher level of abstraction
  - IO of chars and text strings - not just raw bytes
  - IO of values in simple types
- The *TypeReader* and *TypeWriter* classes are not subclasses of the stream classes
  - They are typically built on - and delegates to - a Stream object
  - A Reader or Writer classes has a stream - it is not a stream
  - Reader and Writer classes serve as Stream adapters

THE CLASS ENCODING

- An encoding is a mapping between characters/strings and byte arrays
- An object of class *System.Text.Encoding* represents knowledge about a particular character encoding

THE CLASS ENCODING

- byte[] GetBytes(string) **Instance method**
- byte[] GetBytes(char[]) **Instance method**
  - Encodes a string/char array to a byte array relative to the current encoding
- char[] GetChars(byte[]) Instance method
  - Decodes a byte array to a char array relative to the current encoding
- byte[] Convert(Encoding, Encoding, byte[]) Static method
  - Converts a byte array from one encoding (first parameter) to another encoding (second parameter)

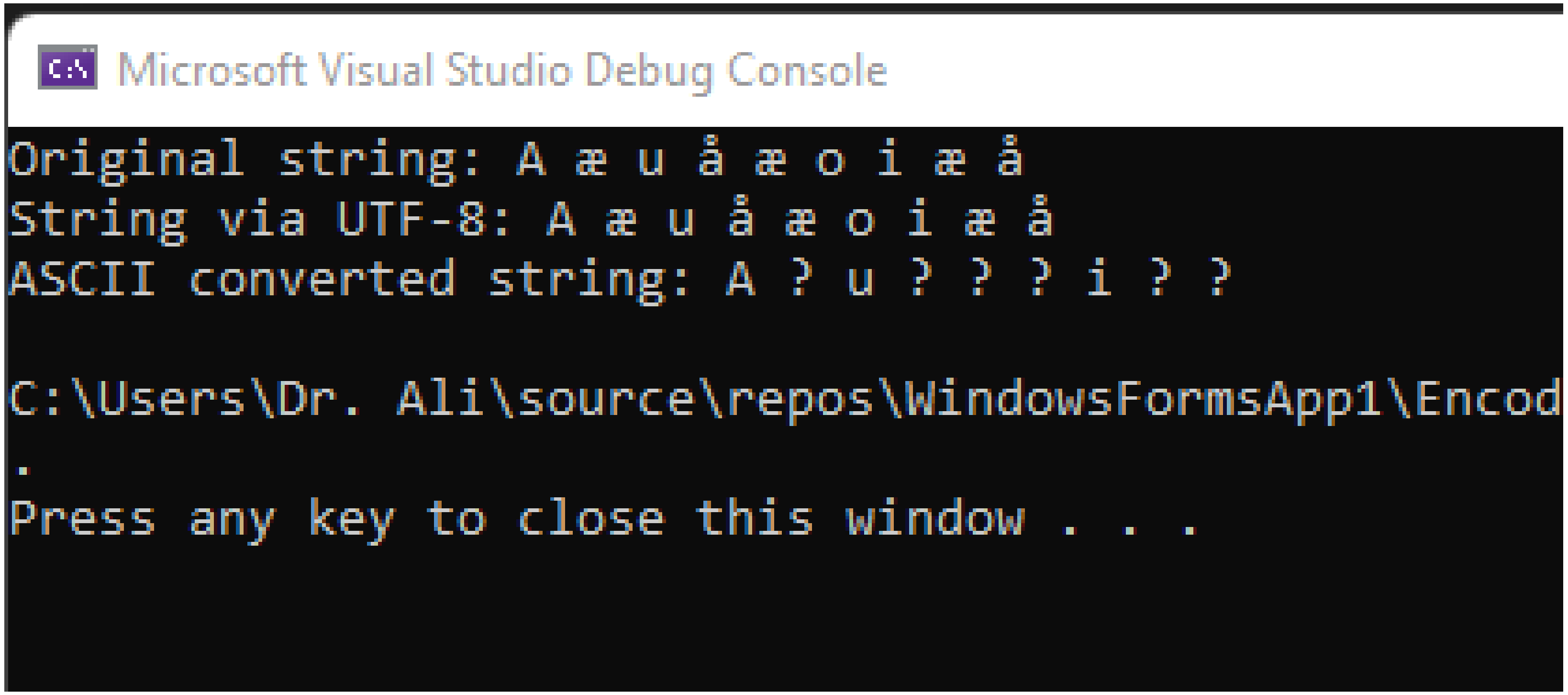
IT TIME TO CODE!
CONVERSIONS IN THE EXAMPLE

- The following example shows how you can use the Encoding class for either converting to and from byte and also convert between different encodings.
- From a unicode string to a byte array in a given encoding *(GetBytes - Encoding).*
- From a byte array in one encoding to a byte array in another encoding (Convert)
- From a byte array in a given encoding to a char array (in unicode)

  (GetChars - Decoding).
- From a char array (in unicode) to a unicode string encoding.

  (via String constructor)

```
using System;
using System.Text;
/* Adapted from an example provided by Microsoft */
class ConvertExampleClass{
    public static void Main()    {
        string unicodeStr =          // "A æ u å æ ø i æ å"
            "A \u00E6 u \u00E5 \u00E6 \u00F8 i \u00E6 \u00E5";
        // Different encodings.
        Encoding ascii = Encoding.ASCII,  unicode = Encoding.Unicode,
                 utf8 = Encoding.UTF8,
                 isoLatin1 = Encoding.GetEncoding("iso-8859-1");

        // Encodes the characters in a string to a byte array:
        byte[] unicodeBytes = unicode.GetBytes(unicodeStr),
               asciiBytes = ascii.GetBytes(unicodeStr),
               utf8Bytes = utf8.GetBytes(unicodeStr),
               isoLatin1Bytes = utf8.GetBytes(unicodeStr);

        // Convert from byte array in unicode to byte array in utf8:
        byte[] utf8BytesFromUnicode =
          Encoding.Convert(unicode, utf8, unicodeBytes);
        // Convert from byte array in utf8 to byte array in ascii:
        byte[] asciiBytesFromUtf8 =
          Encoding.Convert(utf8, ascii, utf8Bytes);

        // Decodes the bytes in byte arrays to a char array:
        char[] utf8Chars = utf8.GetChars(utf8BytesFromUnicode);
        char[] asciiChars = ascii.GetChars(asciiBytesFromUtf8);

        // Convert char[] to string:
        string utf8String = new string(utf8Chars),
               asciiString = new String(asciiChars);

        // Display the strings created before and after the conversion.
        Console.WriteLine("Original string: {0}", unicodeStr);
        Console.WriteLine("String via UTF-8: {0}", utf8String);
        Console.WriteLine("ASCII converted string: {0}", asciiString);
    }
}
```



```
Original string: A æ u å æ o i æ å
String via UTF-8: A æ u å æ o i æ å
ASCII converted string: A ? u ? ? ? i ? ?

C:\Users\Dr. Ali\source\repos\WindowsFormsApp1\Encod
.
Press any key to close this window . . .
```

OUTPUT, JUSTIFY THE “?”

THE CLASS TEXTWRITER

- Class TextWriter supports writing of characters and strings via a chosen encoding
- It is also possible to write textual representations of simple types with use of TextWriter.
- Subclasses of the abstract TextWriter class
  - StreamWriter: For character output in a particular character encoding
  - StringWriter: For stream access to a string
- Plz code the related problem in Sheet 2

MEMBERS IN CLASS STREAMWRITER

- **7 overloaded constructors**
  - **Parameters involved: File name, stream, encoding, buffer size**
    - **`StreamWriter(String)`**
    - **`StreamWriter(Stream)`**
    - **`StreamWriter(Stream, Encoding)`**
  - ***others***
- **17/18 overloaded Write / WriteLine operations**
  - **Chars, strings, simple types. Formatted output**
- **`Encoding`**
  - **A property that gets the encoding used for this `TextWriter`**
- **`NewLine`**
  - **A property that gets/sets the applied newline string of this `TextWriter`**
- ***others***

ADAPTER CLASSES CONT.

- Class TextReader supports reading of characters via a chosen encoding
- Class TextReader does not have methods to read textual representation of simple types
- Subclasses of the abstract TextReader class
- StreamReader: For character input in a particular character encoding
- StringReader: For stream access to a string - (discussed later in the lecture)

PROGRAM

- Reading back the text strings encoded in three different ways, with StreamReader.
- In the last half part of the program, the binary contents of the three files are read and reported.

SELF CODE STUDY

- Plz download the cs file from my drive and run it
- plz download and run me !

StreamReader MEMBERS

- 10 StreamReader constructors
  - Similar to the StreamWriter constructors
  - **StreamReader(String)**
  - **StreamReader(Stream)**
  - **StreamReader(Stream, bool)**
  - **StreamReader(Stream, Encoding)**
  - *others*
- **int Read()** Reads a single character. Returns -1 if at end of file
- **int Read(char[], int, int)** Returns the number of characters read
- **int Peek()**
- **String ReadLine()**
- **String ReadToEnd()**
- **CurrentEncoding**
  - A property that gets the encoding of this StreamReader

THE BINARYWRITER, AND BINARYREADER CLASSES

- Class BinaryWriter allows us to write values of simple types in binary format
- Encodings are only relevant if values of type char are written
- Class BinaryReader allows us to read values of simple types in binary format
- Symmetric to BinaryWriter

BinaryWriter MAIN METHODS AND CONSTRUCTORS

- Two public constructors
  - BinaryWriter(Stream)
  - BinaryWriter(Stream, Encoding)
- 18 overloaded Write operations
  - One for each simple type
  - Write(char),
  - Write(char[]), and Write(char[], int, int) - use Encoding
  - Write(string) - use Encoding
  - Write(byte[]) and Write(byte[], int, int)
- Seek(int offset, SeekOrigin origin)

BinaryReader MAIN METHODS AND CONSTRUCTORS

- Two public constructors
  - BinaryReader(Stream)
  - BinaryReader(Stream, Encoding)
- 15 individually name Read*type* operations
  - ReadBoolean, ReadChar, ReadByte, ReadDouble, ReadDecimal, ReadInt16, ...
- Three overloaded Read operations
  - Read() and Read (char[] buffer, int index, int count)
    read characters - using Encoding
  - Read (bytes[] buffer, int index, int count) reads bytes

EXAMPLES

- http://people.cs.aau.dk/~normark/oop-csharp/html/notes/io-book.html#stream-operations

EXAMPLE 1 , BinaryWriter

```
using System;
using System.IO;

public class BinaryWriteSimpleTypes{

 public static void Main(){
  string fn = "simple-types.bin";

  using(BinaryWriter bw =
        new BinaryWriter( new FileStream(fn, FileMode.Create))){
   bw.Write(5);      // 4  bytes
   bw.Write(5.5);    // 8  bytes
   bw.Write(5555M);  // 16 bytes (decimal dt)
   bw.Write(5==6);   // 1  bytes
  }

  FileInfo fi = new FileInfo(fn);
  Console.WriteLine("Length of {0}: {1}", fn, fi.Length);
 }
}
```

```
Length of simple-types.bin: 29
```

EXAMPLE 2 , BinaryReader

```
using System;
using System.IO;

public class BinaryReadSimpleTypes{

 public static void Main(){
  string fn = "simple-types.bin";

  using(BinaryReader br =
       new BinaryReader(   new FileStream(fn, FileMode.Open))){
   int i = br.ReadInt32();
   double d = br.ReadDouble();
   decimal dm = br.ReadDecimal();
   bool b = br.ReadBoolean();

   Console.WriteLine("Integer i: {0}", i);
   Console.WriteLine("Double d: {0}", d);
   Console.WriteLine("Decimal dm: {0}", dm);
   Console.WriteLine("Boolean b: {0}", b);
  } } }
```

```
Integer i: 5
Double d: 5,5
Decimal dm: 5555
Boolean b: False
```

The **File** and **FileInfo** classes

- The class **FileInfo** represents a file
- The class **File** holds static methods for creation, copying, deletion, moving, and opening of files

Example: the **FileInfo** classes

```
using System;
using System.IO;
public class FileInfoDemo{
  public static void Main(){
    // Setting up file names
    string fileName = "file-info.cs",  fileNameCopy = "file-info-copy.cs";

    // Testing file existence
    FileInfo fi = new FileInfo(fileName); // this source file
    Console.WriteLine("{0} does {1} exist",  fileName, fi.Exists ? "" : "not");

    // Show file info properties:
    Console.WriteLine("DirectoryName: {0}", fi.DirectoryName);
    Console.WriteLine("FullName: {0}", fi.FullName);
    Console.WriteLine("Extension: {0}", fi.Extension);
    Console.WriteLine("Name: {0}", fi.Name);
    Console.WriteLine("Length: {0}", fi.Length);
    Console.WriteLine("CreationTime: {0}", fi.CreationTime);

    // Copy one file to another
    fi.CopyTo(fileNameCopy);
    FileInfo fiCopy  = new FileInfo(fileNameCopy);
```

Example (Cont): The **File** and **FileInfo** classes

```
// Does the copy exist?
   Console.WriteLine("{0} does {1} exist", fileNameCopy, fiCopy.Exists ? "" : "not");

   // Delete the copy again
   fiCopy.Delete();

   // Does the copy exist?
   Console.WriteLine("{0} does {1} exist",
                    fileNameCopy, fiCopy.Exists ? "" : "not"); // !!??

   // Create new FileInfo object for the copy
   FileInfo fiCopy1  = new FileInfo(fileNameCopy);
   // Check if the copy exists?
   Console.WriteLine("{0} does {1} exist", fileNameCopy, fiCopy1.Exists ? "" : "not");

   // Achieve a TextReader (StreamReader) from the file info object
   // and echo the first ten lines in the file to standard output
   using(StreamReader sr = fi.OpenText()){
     for (int i = 1; i <= 10; i++)
       Console.WriteLine("  " + sr.ReadLine());
   }   }  }
```

Program: A demonstration of the File class.

```
using System;
using System.IO;
public class FileDemo
{
    public static void Main()
    {
        // Setup file names
        string fileName = "binarystreams.exe", // this source file
        fileNameCopy = "fileCopy.cs";

        // Does this source file exist?
        Console.WriteLine("{0} does {1} exist",
        fileName, File.Exists(fileName) ? "" : "not");

        // Copy this source file
        File.Copy(fileName, fileNameCopy);

        // Does the copy exist?
        Console.WriteLine("{0} does {1} exist", fileNameCopy,
        File.Exists(fileNameCopy) ? "" : "not");

        // Delete the copy again
        Console.WriteLine("Deleting {0}", fileNameCopy);
        File.Delete(fileNameCopy);

        // Does the deleted file exist
        Console.WriteLine("{0} does {1} exist", fileNameCopy,
        File.Exists(fileNameCopy) ? "" : "not");

        // Read all lines in source file and echo
        // one of them to the console
        string[] lines = File.ReadAllLines(fileName);
        Console.WriteLine("Line {0}: {1}", 6, lines[6]);    }}
```

- The class **DirectoryInfo** represents a directory
- The class **Directory** holds static methods for creating, moving, and enumerating through directories

Program: A demonstration of the DirectoryInfo class.

```
using System;
using System.IO;
public class DirectoryInfoDemo
{
    public static void Main()
    {
        string fileName = "directory-info.cs";    // The current source file

        // Get the DirectoryInfo of the current directory
        // from the FileInfo of the current source file
        FileInfo fi = new FileInfo(fileName);     // This source file
        DirectoryInfo di = fi.Directory;

        Console.WriteLine("File {0} is in directory \n   {1}", fi, di);

        // Get the files and directories in the parent directory.
        FileInfo[] files = di.Parent.GetFiles();
        DirectoryInfo[] dirs = di.Parent.GetDirectories();

        // Show the name of files and directories on the console
        Console.WriteLine("\nListing directory {0}:", di.Parent.Name);
        foreach (DirectoryInfo d in dirs)
            Console.WriteLine(d.Name);
        foreach (FileInfo f in files)
            Console.WriteLine(f.Name);
    }
}
```

Do you note anything ?

Program: A demonstration of the Directory class - similar to the DirectoryInfo demo program.

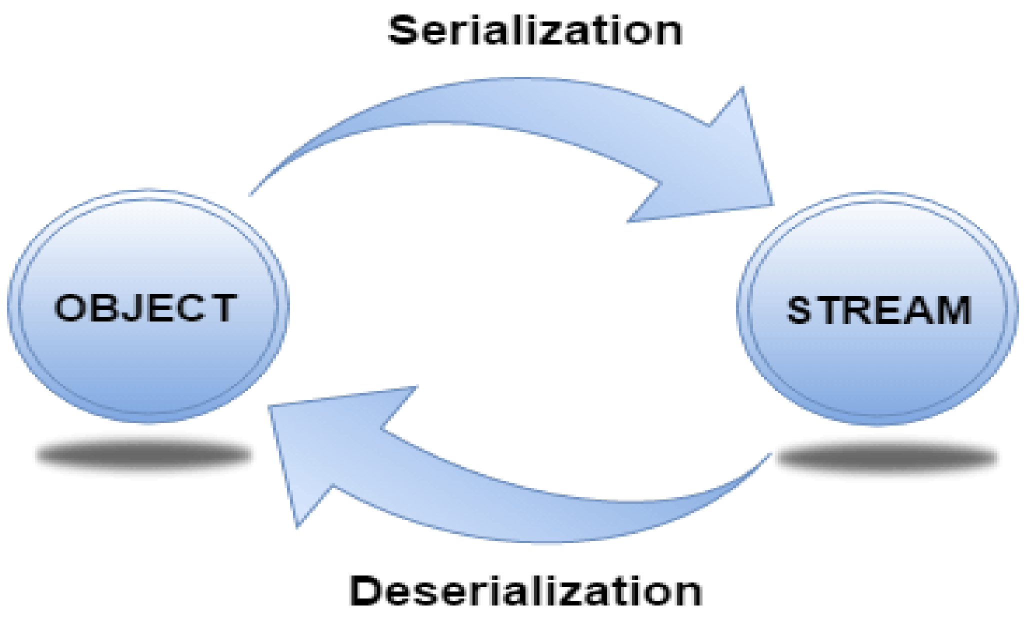
```
using System;
using System.IO;
public class DirectoryDemo
{
    public static void Main()
    {
        string fileName = "binarystreams.exe";         // The current source file
        FileInfo fi = new FileInfo(fileName);    // This source file

        string thisFile = fi.FullName,
               thisDir = Directory.GetParent(thisFile).FullName,
               parentDir = Directory.GetParent(thisDir).FullName;

        Console.WriteLine("This file: {0}", thisFile);
        Console.WriteLine("This Directory: {0}", thisDir);
        Console.WriteLine("Parent directory: {0}", parentDir);

        string[] files = Directory.GetFiles(parentDir);
        string[] dirs = Directory.GetDirectories(parentDir);

        Console.WriteLine("\nListing directory {0}:", parentDir);
        foreach (string d in dirs)
            Console.WriteLine(d);
        foreach (string f in files)
            Console.WriteLine(f);    }   }
```

Output in the next Slide

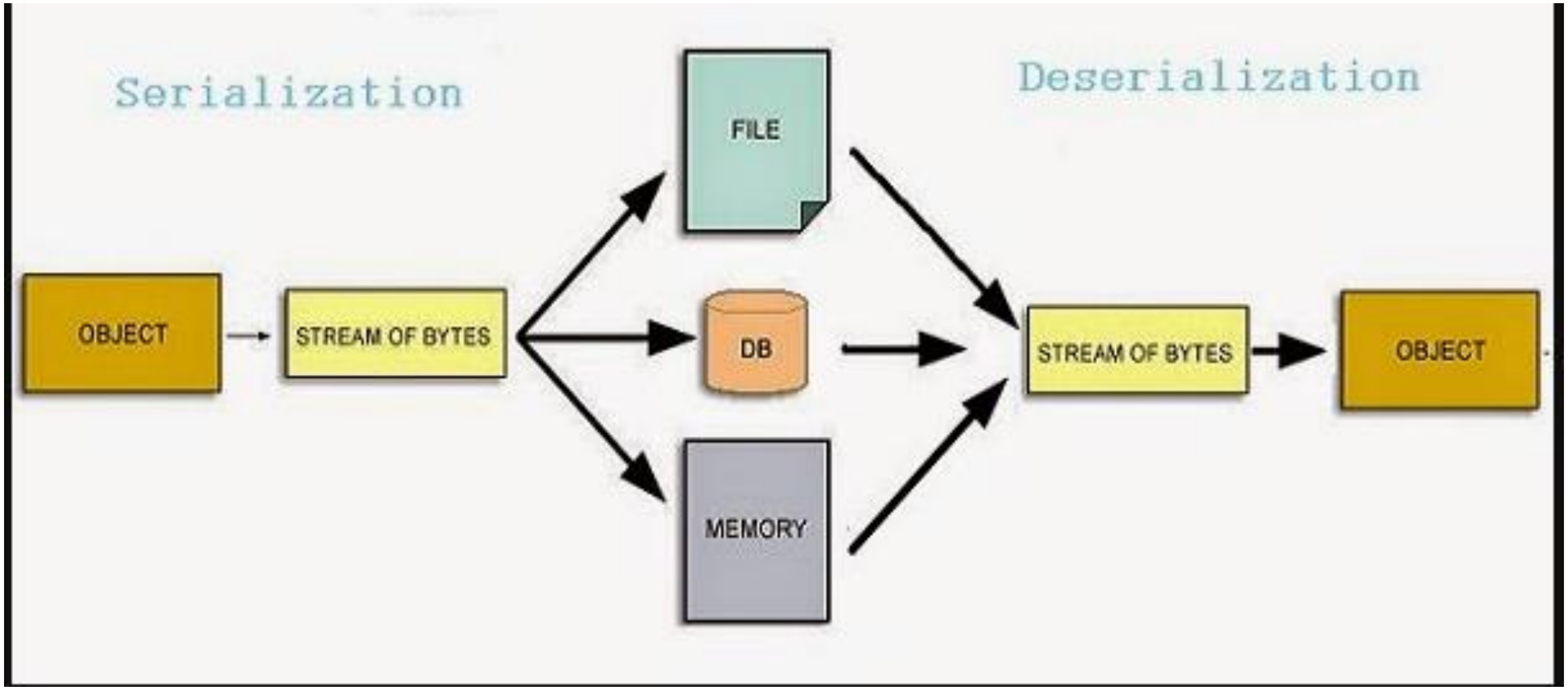
Output

```
This file: C:\Users\Dr. Ali\source\repos\binarystreams\binarystreams\bin\Debug\net6.0\binarystreams.exe
This Directory: C:\Users\Dr. Ali\source\repos\binarystreams\binarystreams\bin\Debug\net6.0
Parent directory: C:\Users\Dr. Ali\source\repos\binarystreams\binarystreams\bin\Debug

Listing directory C:\Users\Dr. Ali\source\repos\binarystreams\binarystreams\bin\Debug:
C:\Users\Dr. Ali\source\repos\binarystreams\binarystreams\bin\Debug\net6.0
```

Serialization



What is object serialization ?



Serialization

- Serialization
  - Writes an object o to a file or any type of streams
  - Also writes the objects referred from o
- Deserialization
  - Reads a serialized file in order to reestablish the serialized object o
  - Also reestablishes the network of objects originally referred from o
- Serialization and deserialization is supported via classes that implement the Iformatter interface:
  - **BinaryFormatter** and **SoapFormatter**
- Methods in **Iformatter**:
  - **Serialize** and **Deserialize**
- **During this course we will study only BinaryFormatter for binary object serialization**

WHY SERIALIZATION ?

- Serialization is the best way to save objects in streams so that we can “send” any where.
- Object state can be resorted after deserialization
- Privacy and object state issues !!!

Example of Serialization in C#

- In this example we will serialize and restore a person object compositing a date object.
- We will define class Date and Person.
- A controller program will make instance from the person class and serialize it to a files and then restore/deserialize it back again into memory

THE USED NAMESPACES

```
using System;
using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
```

STEP 1: DEFINING THE CLASSES

```
[Serializable]
public class Date
{
    private ushort year;
    private byte month, day;
    private DayOfWeek nameOfDay;

    public Date(int year, int month, int day)
    {
        this.year = (ushort)year;
        this.month = (byte)month;
        this.day = (byte)day;
        this.nameOfDay = (new DateTime(year, month, day)).DayOfWeek;
    }

    public Date(Date d)
    {
        this.year = d.year; this.month = d.month;
        this.day = d.day; this.nameOfDay = d.nameOfDay;
    }
```

STEP 1: DEFINING THE CLASSES

```
public int Year { get { return year; } }
 public int Month { get { return month; } }
 public int Day { get { return day; } }

 // return this minus other, as of usual birthday calculations.
 public int YearDiff(Date other)
 {
     if (this.Equals(other))
         return 0;
     else if ((new Date(other.year, this.month, this.day)).IsBefore(other))
         return this.year - other.year - 1;
     else
         return this.year - other.year;
 }
```

STEP 1: DEFINING THE CLASSES

```
public override bool Equals(Object obj)
  {
     if (obj == null)
        return false;
     else if (this.GetType() != obj.GetType())
        return false;
     else if (ReferenceEquals(this, obj))
        return true;
     else if (this.year == ((Date)obj).year &&
           this.month == ((Date)obj).month &&
           this.day == ((Date)obj).day)
        return true;
     else return false;
  }
```

STEP 1: DEFINING THE CLASSES

```
public bool IsBefore(Date other)
  {
    return
     this.year < other.year ||
     this.year == other.year && this.month < other.month ||
     this.year == other.year && this.month == other.month && this.day < other.day;
  }
  public static Date Today
  {
    get
    {
      DateTime now = DateTime.Now;
      return new Date(now.Year, now.Month, now.Day);
    }
  }

  public override string ToString()
  {
    return string.Format("{0} {1}.{2}.{3}", nameOfDay, day, month, year);
  }
}
```

STEP 1: DEFINING THE CLASSES

```
[Serializable]
public class Person
{
    private string name;
    private int age;    // Redundant
    private Date dateOfBirth, dateOfDeath;
    public Person(string name, Date dateOfBirth)
    {
        this.name = name;
        this.dateOfBirth = dateOfBirth;
        this.dateOfDeath = null;
        age = Date.Today.YearDiff(dateOfBirth);
    }
    public Date DateOfBirth
    {
        get { return new Date(dateOfBirth); }
    }
    public int Age
    {
        get { return Alive ? age : dateOfDeath.YearDiff(dateOfBirth); }
    }
```

STEP 1: DEFINING THE CLASSES

```
public bool Alive
  {
     get { return dateOfDeath == null; }
  }

  public void Died(Date d)
  {
     dateOfDeath = d;
  }

  public void Update()
  {
     age = Date.Today.YearDiff(dateOfBirth);
  }

  public override string ToString()
  {
     return "Person: " + name +  "  *" + dateOfBirth +   (Alive ? "" : "  +" + dateOfDeath) +
            "  Age: " + age;
  }
```

STEP 1: DEFINING THE CLASSES

```
[OnSerializing()]
   internal void OnSerializingMethod(StreamingContext context)
   {
       Console.WriteLine("serializing . . . .");
   }

   [OnSerialized()]
   internal void OnSerializedMethod(StreamingContext context)
   {
       Console.WriteLine("Done !");
   }
   [OnDeserializing()]
   internal void OnDeserializingMethod(StreamingContext context)
   {
       Console.WriteLine("deserializing . . . .");
   }
   [OnDeserialized()]
   internal void OnDeserializedMethod(StreamingContext context)
   {
       Console.WriteLine("obj Ready");
   }}
```

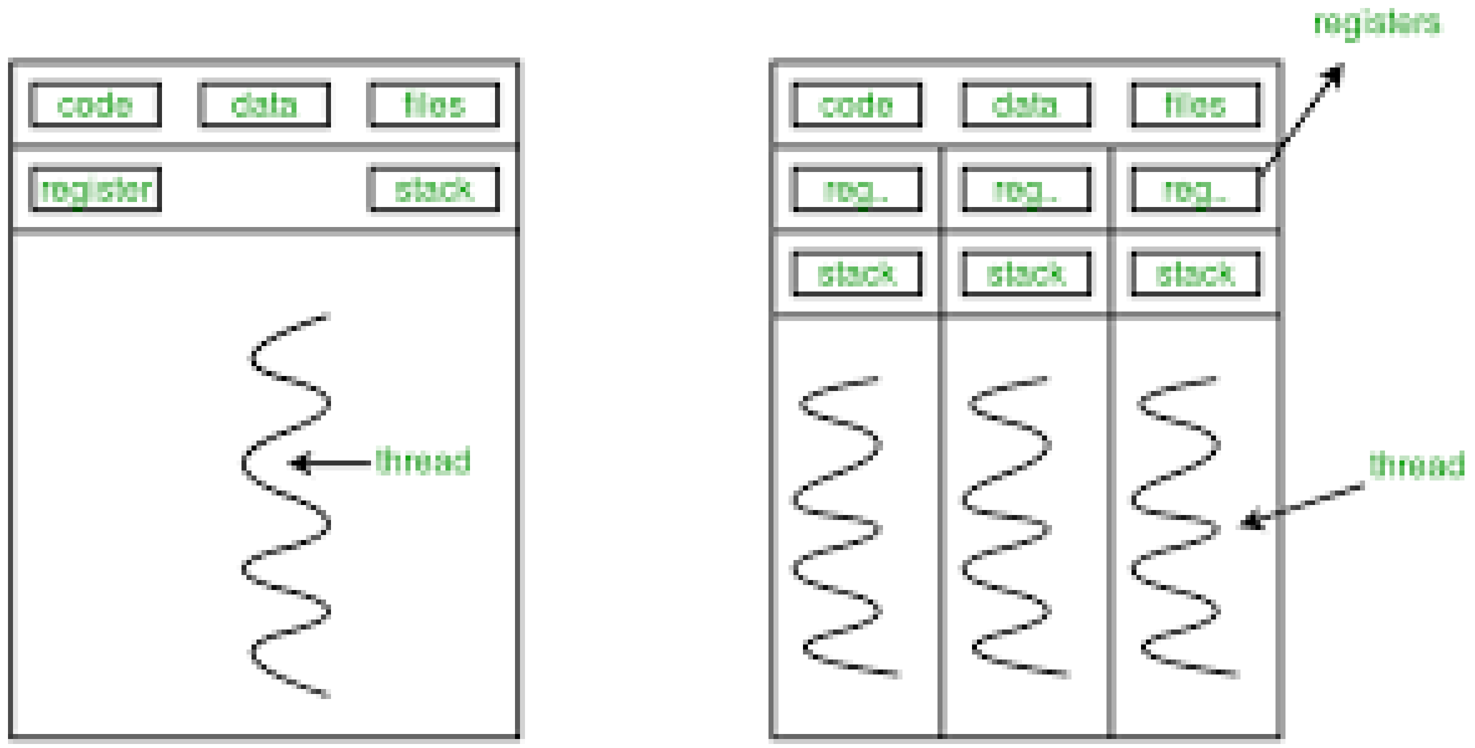
STEP 2: ATTEMPTING TO SERIALIZE AND DESERIALIZE

```
class Client
{
   public static void Main()
   {
      Person p = new Person("Peter", new Date(1936, 5, 11));
      p.Died(new Date(2007, 5, 10));
      Console.WriteLine("{0}", p);
      using (FileStream strm =    new FileStream("person.dat", FileMode.Create))
      {
         IFormatter fmt = new BinaryFormatter();
         fmt.Serialize(strm, p);
      }
      // ------------------------------------------------------------
      p = null;
      Console.WriteLine("Reseting person");
      // ------------------------------------------------------------
      using (FileStream strm =   new FileStream("person.dat", FileMode.Open))
      {
         IFormatter fmt = new BinaryFormatter();
         p = fmt.Deserialize(strm) as Person;
      }
      Console.WriteLine("{0}", p);    }}
```

OUTPUT

```
Person: Peter  *Monday 11.5.1936  +Thursday 10.5.2007  Age: 85
serializing . . . .
Done !
Resetting person
deserializing . . . .
obj Ready
Person: Peter  *Monday 11.5.1936  +Thursday 10.5.2007  Age: 85

Press any key to close this window . . .
```

Threading



single-threaded process

multithreaded process

WHAT IS A THREAD?

- A thread is nothing more than a process.
- On the computer, a thread is a process moving through time.
- The process performs sets of sequential steps, each step executing a line of code.
- Because the steps are sequential, each step takes a given amount of time.
- The time it takes to complete a series of steps is the sum of the time it takes to perform each programming step.

WHAT ARE MULTITHREADED APPLICATIONS (NETWORK APPLICATIONS)?

- For a long time, most programming applications were single-threaded.
- That means there was only one thread in the entire application.
- You could never do computation A until completing computation B.
- A multithreaded application allows you to run several threads, each thread running in its own process. So theoretically you can run step 1 in one thread and at the same time run step 2 in another thread.
- At the same time you could run step 3 in its own thread, and even step 4 in its own thread
- Theoretically, if all four steps took about the same time, you could finish your program in a quarter of the time it takes to run a single thread (assuming you had a 4 processor machine)
- An Unusual Analogy
  - Human Body

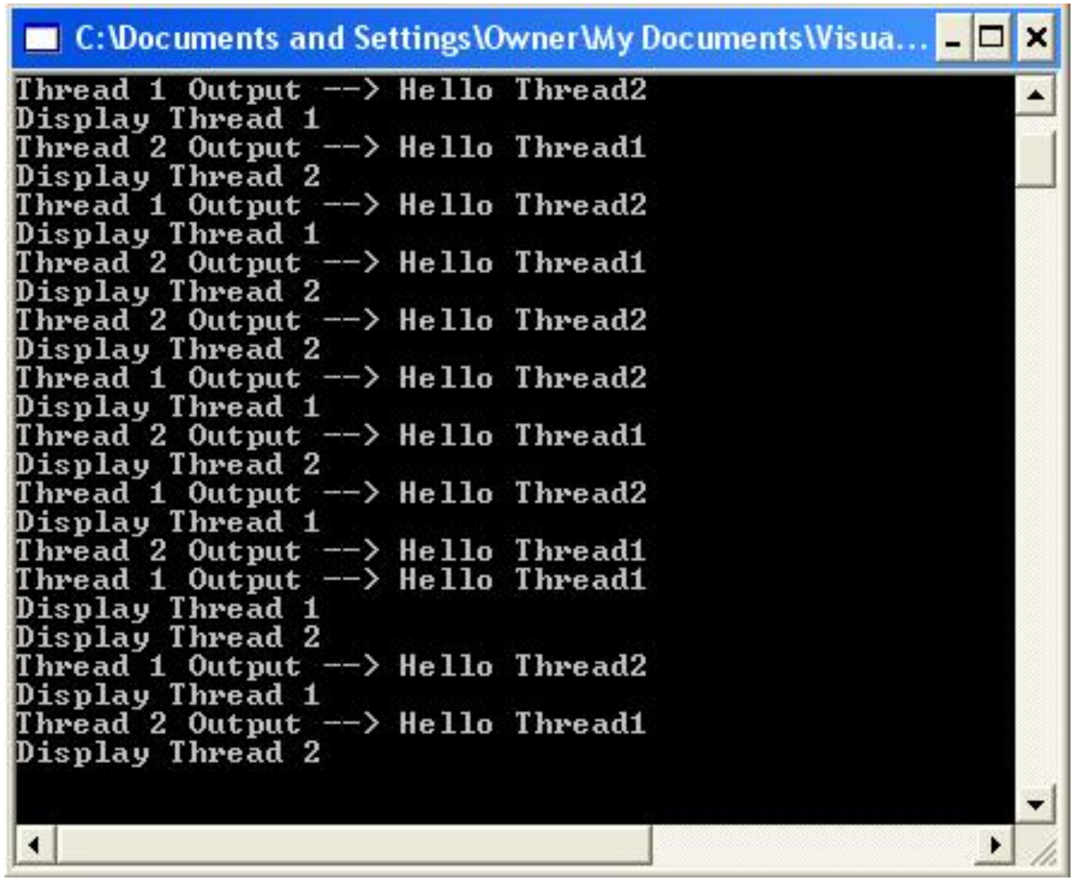
CONCURRENCY !!

Can’t wait,
lets Code !!

- We are aiming to Create two threads sharing a common variable in memory

```
using System;
using System.IO;
class MThread_APP
{
    // used to indicate which thread we are in
    private static string _threadOutput = "";
    private static bool _stopThreads = false;
    static void DisplayThread1()
    {
        while (_stopThreads == false)
        {
            Console.WriteLine("Display Thread 1");
            // Assign the shared memory to a message about
thread #1
            _threadOutput = "Hello Thread1";
            Thread.Sleep(1000);  // simulate a lot of
processing
            // tell the user what thread we are in thread #1,
and display shared memory
            Console.WriteLine("Thread 1 Output --> {0}",
_threadOutput);
        }    }
static void DisplayThread2()
   {
       while (_stopThreads == false)
       {
           Console.WriteLine("Display Thread 2");
     // Assign the shared memory to a message about thread #2
           _threadOutput = "Hello Thread2";
           Thread.Sleep(1000);  // simulate a lot of processing
           // tell the user we are in thread #2
           Console.WriteLine("Thread 2 Output --> {0}",
            _threadOutput);   }
   }
   public static void Main()
   {
       Thread thread1 = new Thread(new
      ThreadStart(DisplayThread1));
       Thread thread2 = new Thread(new
      ThreadStart(DisplayThread2));
       // start them
       thread1.Start();
       thread2.Start();}}
```

OUTPUT Any comment !!



```
Thread 1 Output --> Hello Thread2
Display Thread 1
Thread 2 Output --> Hello Thread1
Display Thread 2
Thread 1 Output --> Hello Thread2
Display Thread 1
Thread 2 Output --> Hello Thread1
Display Thread 2
Thread 2 Output --> Hello Thread2
Display Thread 2
Thread 1 Output --> Hello Thread2
Display Thread 1
Thread 2 Output --> Hello Thread1
Display Thread 2
Thread 1 Output --> Hello Thread2
Display Thread 1
Thread 2 Output --> Hello Thread1
Thread 1 Output --> Hello Thread1
Display Thread 1
Display Thread 2
Thread 1 Output --> Hello Thread2
Display Thread 1
Thread 2 Output --> Hello Thread1
Display Thread 2
```

EXPLANATION

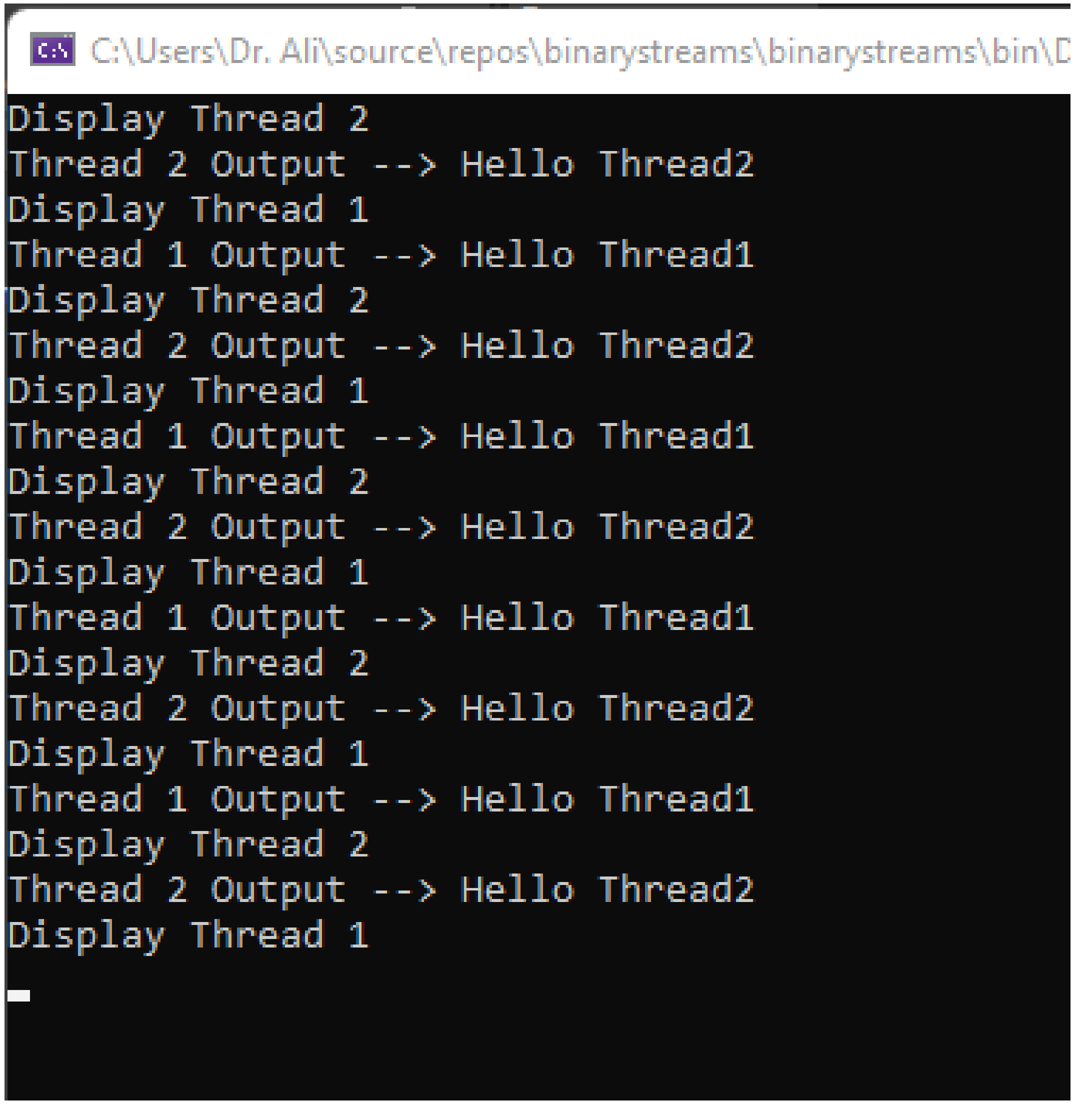
- The code theoretically is executing the two methods DisplayThread1 and DisplayThread2, simultaneously.
- Each method shares the variable, _threadOutput.
- Race condition !
  - Each method shares the variable, _threadOutput. So it is possible that although _threadOutput is assigned a value "Hello Thread1" in thread #1 and displays _threadOutput two lines later to the console, that somewhere in between the time thread #1 assigns it and displays it, thread #2 assigns _threadOutput the value "Hello Thread2"
  - Thread#1 assigns and thread#2 displays !!
  - How to organize that !!
  - Use **Lock**

THREAD SAFE CODE

- The best way to avoid race conditions is to write thread-safe code
- Make tow instances (don’t share)
- Let any one to win the race!
- The way we prevent one thread from affecting the memory of the other class while one is occupied with that memory is called *locking.*
- So any thread trying to modify a field of the class while inside the lock will be *blocked*
- Blocking means that the thread trying to change the variable will sit and wait until the lock is released on the locked thread.
- The thread is released from the lock upon reaching the last bracket in the lock { } construct.

```
using System;
using System.IO;
class MThread_APP
{
    // used to indicate which thread we are in
    private  string _threadOutput = "";
    private  bool _stopThreads = false;
    public MThread_APP()    {
        Thread thread1 = new Thread(new ThreadStart(DisplayThread1));
        Thread thread2 = new Thread(new ThreadStart(DisplayThread2));
        // start them
        thread1.Start();
        thread2.Start();
    }
     void DisplayThread1()
    {
        while (_stopThreads == false)    {
            // lock on the current instance of the class for thread #1
            lock (this)      {
                Console.WriteLine("Display Thread 1");
                _threadOutput = "Hello Thread1";
                Thread.Sleep(1000);  // simulate a lot of processing
// tell the user what thread we are in thread #1
                Console.WriteLine("Thread 1 Output --> {0}",
_threadOutput);            }// lock released  for thread #1 here
        }    }
void DisplayThread2()    {
        while (_stopThreads == false)       {

// lock on the current instance of the class for thread #2
            lock (this)          {
                Console.WriteLine("Display Thread 2");
                _threadOutput = "Hello Thread2";
                Thread.Sleep(1000);  // simulate a lot of
processing
                                      // tell the user what
thread we are in thread #1
Console.WriteLine("Thread 2 Output --> {0}", _threadOutput);
            } // lock released  for thread #2 here
        }    }}
class MainClass
{
    public static void Main()
    {
      new MThread_APP();
    }}
```

OUTPUT


C:\Users\Dr. Ali\source\repos\binarystreams\binarystreams\bin\D

```
Display Thread 2
Thread 2 Output --> Hello Thread2
Display Thread 1
Thread 1 Output --> Hello Thread1
Display Thread 2
Thread 2 Output --> Hello Thread2
Display Thread 1
Thread 1 Output --> Hello Thread1
Display Thread 2
Thread 2 Output --> Hello Thread2
Display Thread 1
Thread 1 Output --> Hello Thread1
Display Thread 2
Thread 2 Output --> Hello Thread2
Display Thread 1
Thread 1 Output --> Hello Thread1
Display Thread 2
Thread 2 Output --> Hello Thread2
Display Thread 1
```

https://www.c-sharpcorner.com/article/introduction-to-multithreading-in-C-Sharp/