VIA University College

DNP-A18

.NET API

An overview of the .NET API

.NET Standard

What is the .NET Standard and how is it useful?

.NET Core API

Streaming, Serialization, Threading, Networking, Reflection and more!

Exercises

Work with the .NET Core API

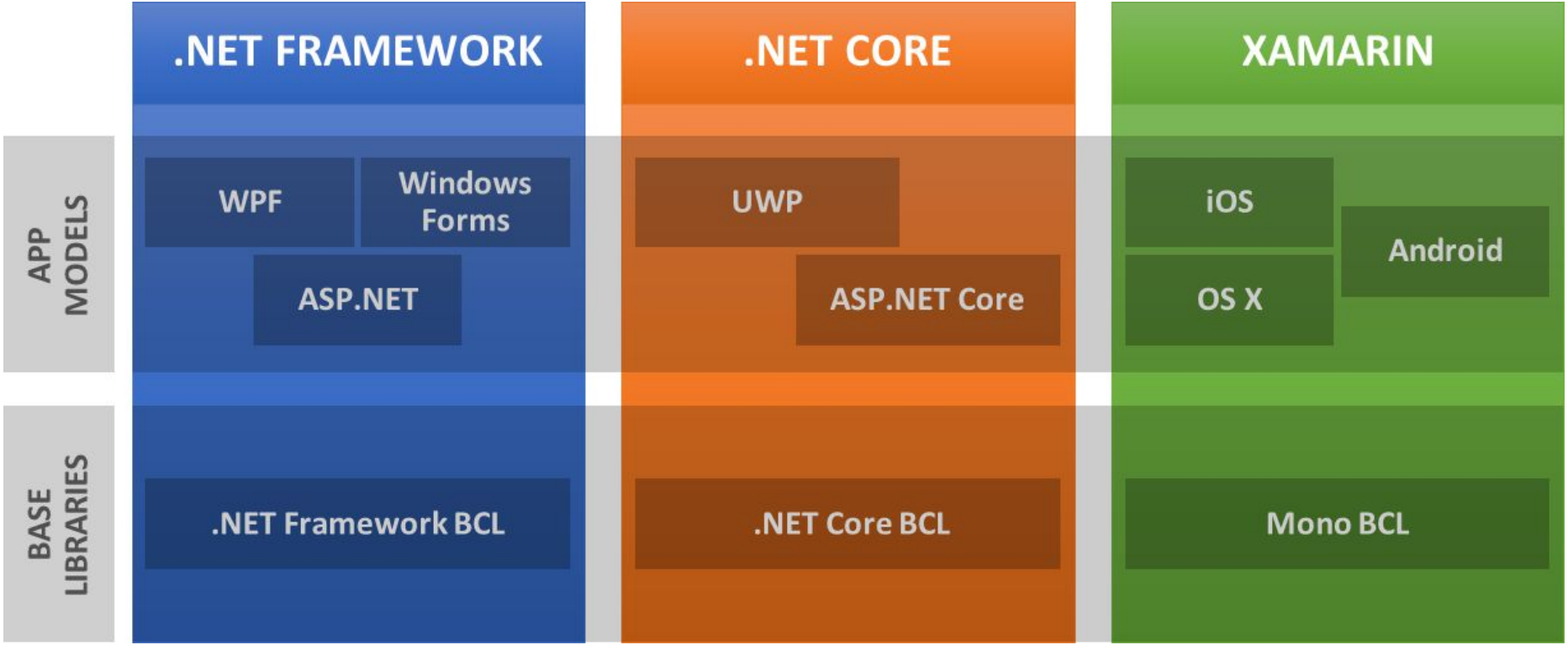
Motivation Class Library

.NET Standard

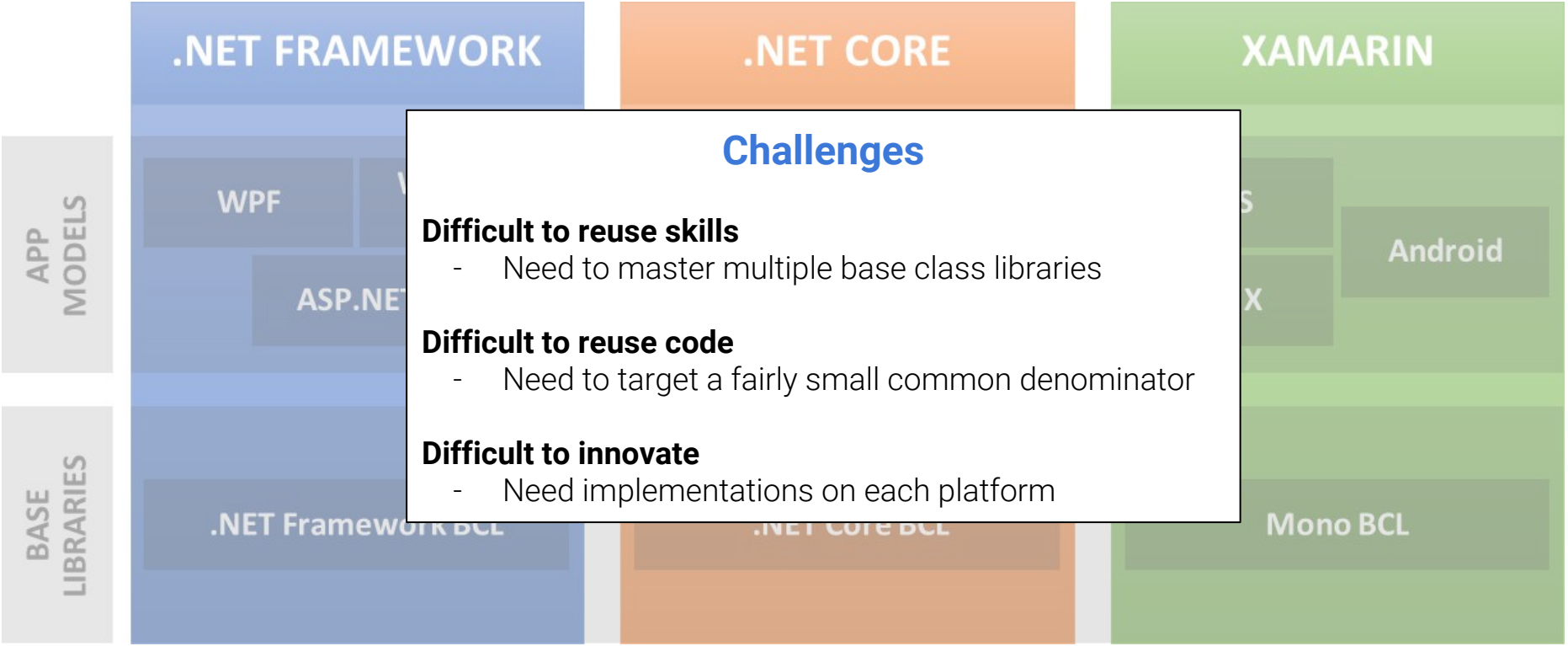
- Building blocks in the form of classes and components
- Avoids code reuse
- Robust, stable and high quality basis
- Enables simple and productive software development

Before .NET Standard

.NET Standard



Before .NET Standard



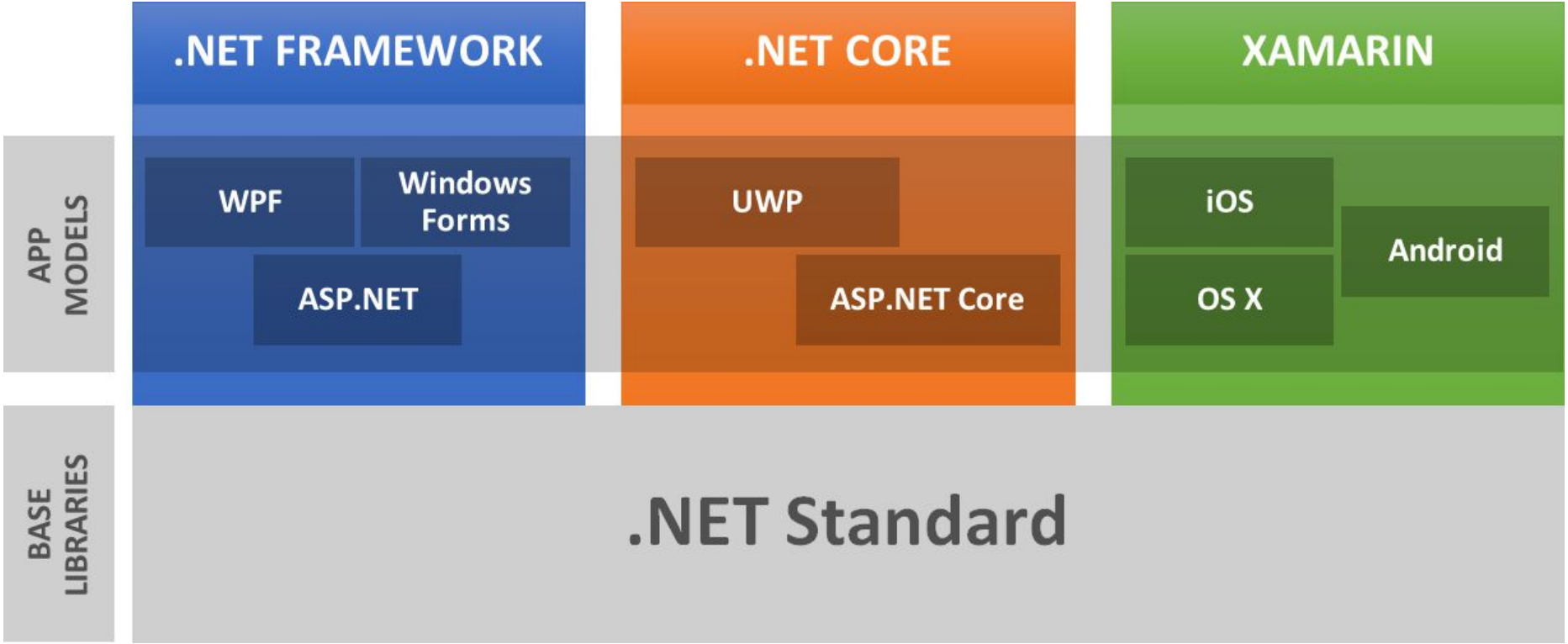
Challenges

Difficult to reuse skills
- Need to master multiple base class libraries

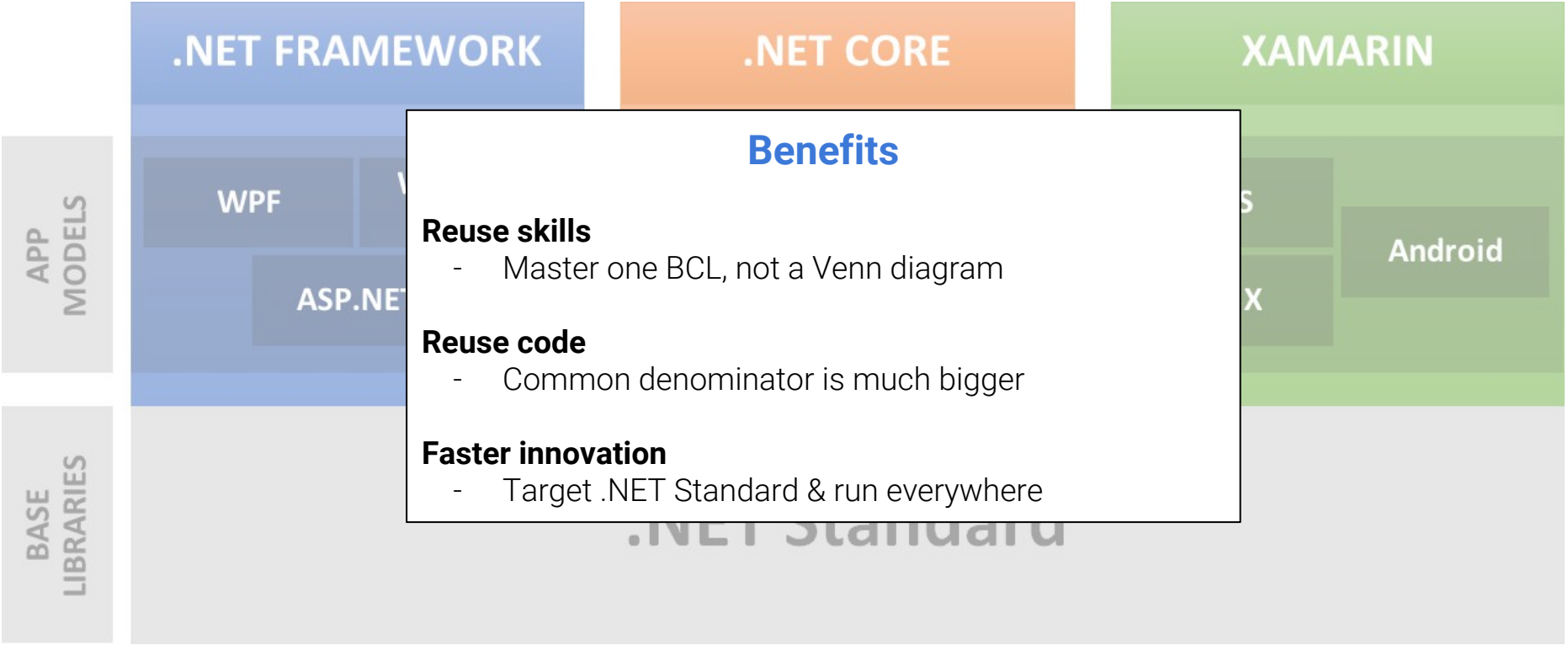
Difficult to reuse code
- Need to target a fairly small common denominator

Difficult to innovate
- Need implementations on each platform

Introducing .NET Standard



Introducing .NET Standard



Benefits

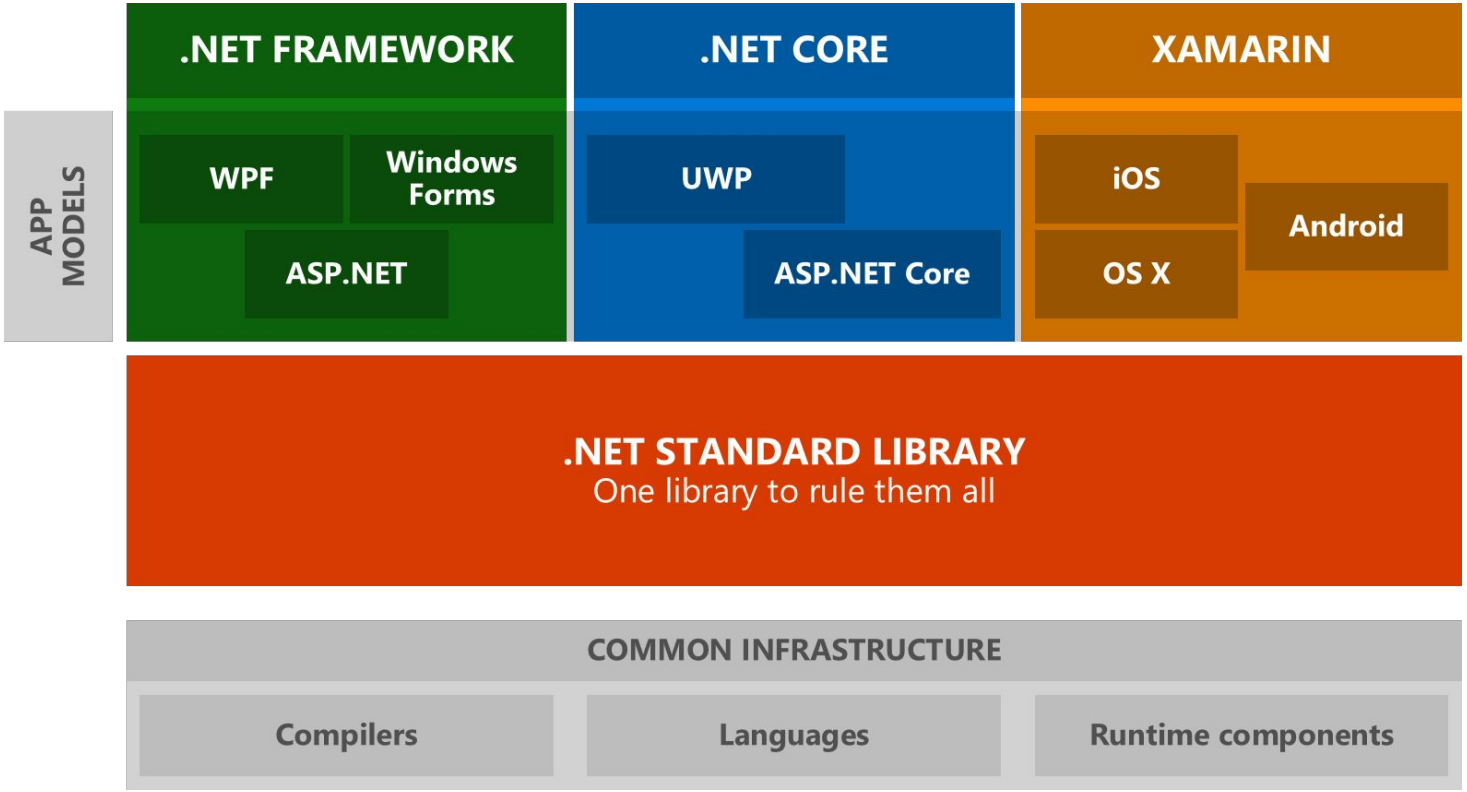
Reuse skills
- Master one BCL, not a Venn diagram

Reuse code
- Common denominator is much bigger

Faster innovation
- Target .NET Standard & run everywhere

.NET Today

.NET Standard



What is .NET Standard?

- .NET Standard **is a specification**
- A set of APIs that **all .NET platforms have to implement**

.NET Standard	~	**HTML specification**
.NET Framework **.NET Core** **Xamarin**	~	**Browsers**

.NET Core is an implementation of the .NET Standard
They are **fully separated**, e.g. different GitHub repositories

APIs in .NET Standard 2.0

XML	XLinq • XML Document • XPath • Schema • XSL
SERIALIZATION	BinaryFormatter • Data Contract • XML
NETWORKING	Sockets • HTTP • Mail • WebSockets
IO	Files • Compression • MMF
THREADING	Threads • Thread Pool • Tasks
CORE	Primitives • Collections • Reflection • Interop • Linq

.NET API Overview

.NET API Browser

Microsoft Technologies Documentation Resources .NET Search

.NET APIs .NET Core .NET Framework ASP.NET Xamarin Azure

.NET API Browser

Welcome to the .NET API Browser – your one-stop shop for all .NET-based APIs from Microsoft. Start searching for any managed APIs by typing in the box below. You can learn more about the API Browser in our blog post. If you have any feedback, use our UserVoice site.

.NET Core 2.0

Search

.NET Core API Reference version 2.0

Name	Description
Microsoft.CSharp Namespace	The Microsoft.CSharp namespace contains classes that support compilation and code generation using the C# language.
Microsoft.CSharp.RuntimeBinder Namespace	The Microsoft.CSharp.RuntimeBinder namespace provides classes and interfaces that support interoperation between Dynamic Language Runtime and C#.
Microsoft.SqlServer.Server Namespace	The Microsoft.SqlServer.Server namespace contains classes, interfaces, and enumerations that are specific to the integration of the Microsoft .NET Framework common language runtime (CLR) into Microsoft SQL Server, and the SQL Server database engine process execution environment.
Microsoft.VisualBasic Namespace	The Microsoft.VisualBasic namespace contains types that support the Visual Basic Runtime in Visual Basic.

You can **search** for a namespace, class, method, or interface by typing its full or partial name directly in the search bar!

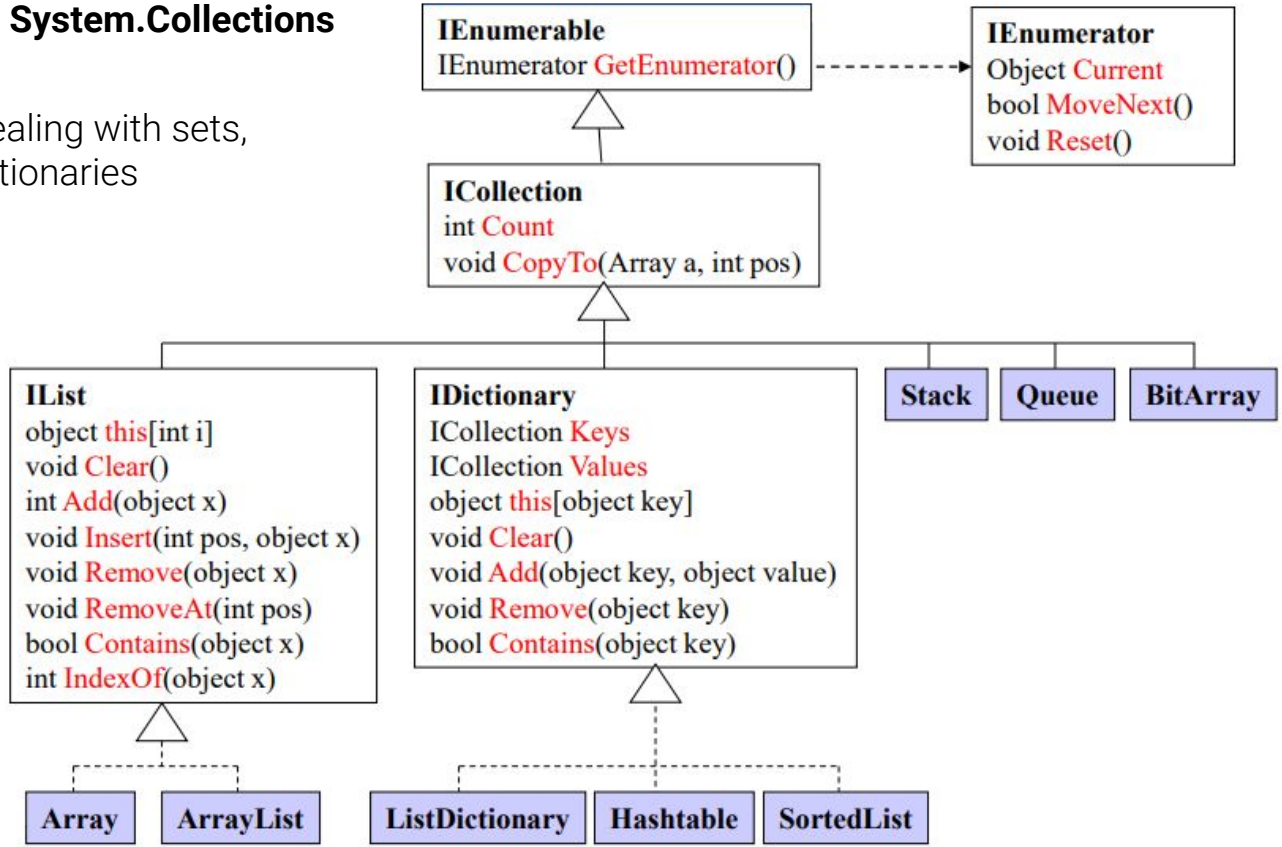
The actual implementation of the .NET Core API (CoreFX) can be found here: .NET Core Foundational Libraries

Collections

.NET Core API

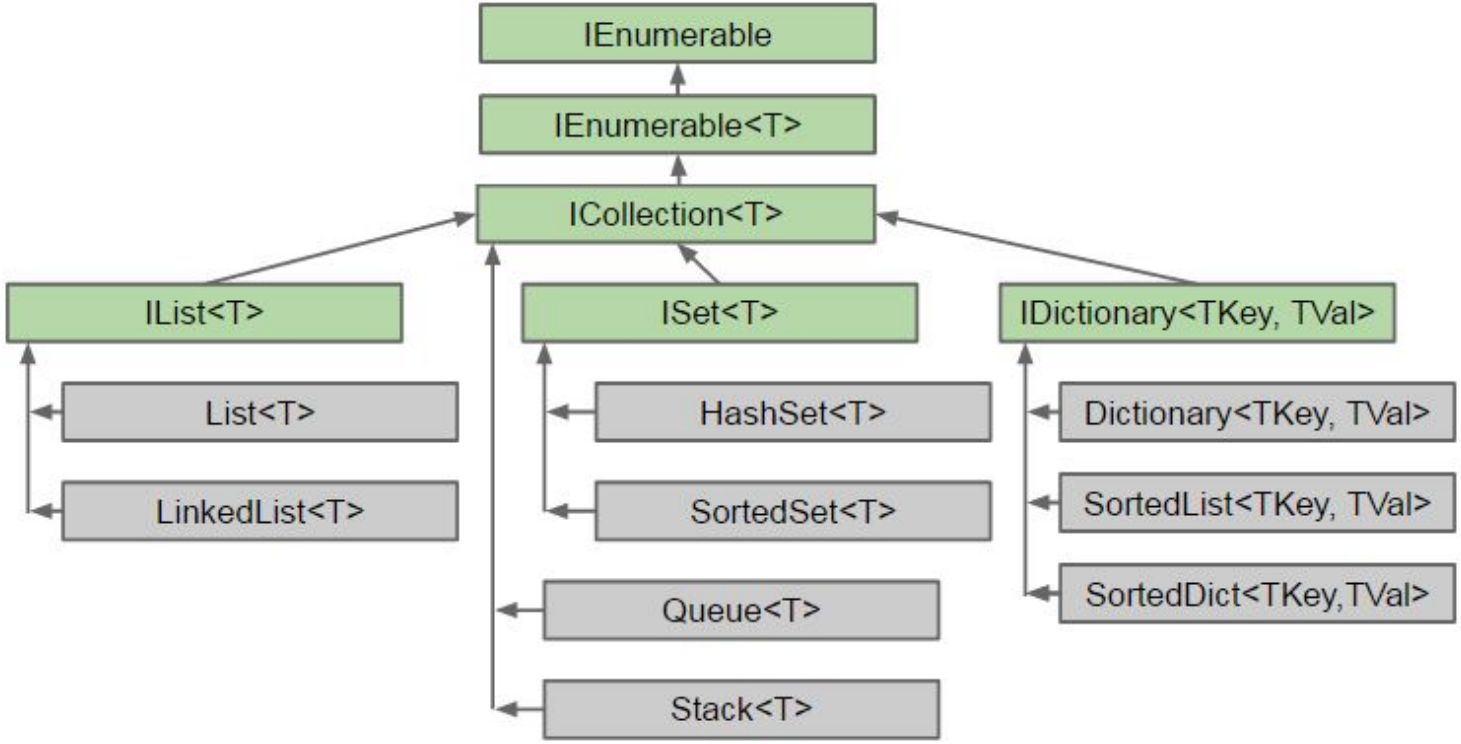
Namespace **System.Collections**

Types for dealing with sets, lists and dictionaries



Generic Collections

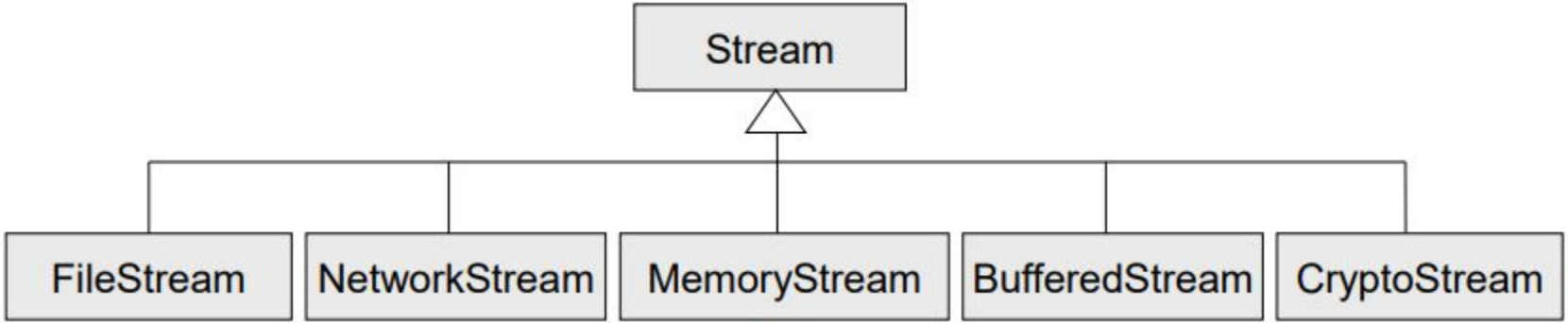
Namespace **System.Collections.Generic**



Streaming

The **System.IO** namespace contains

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



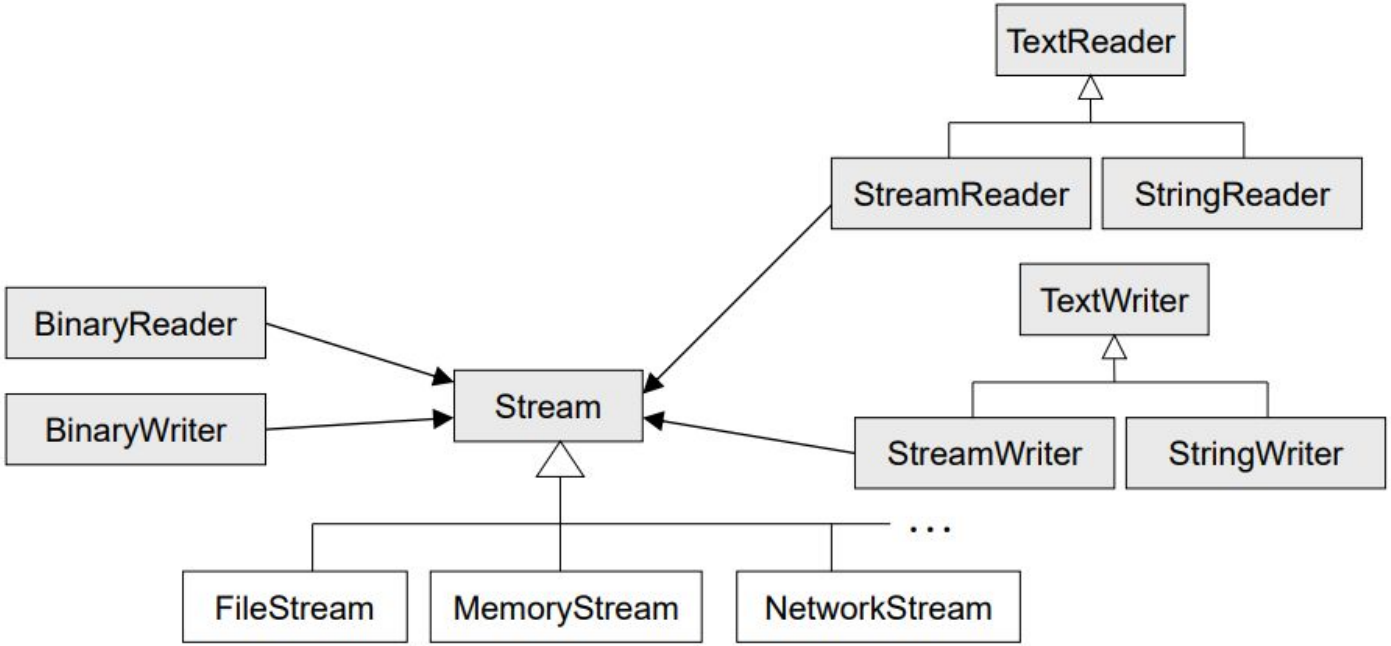
- Readers and Writers for formatting
- Streams support synchronous and asynchronous protocols

Stream class

Readers and Writers

Readers and Writers overtake formatting tasks

- BinaryReader and BinaryWriter for binary data
- TextReader and TextWriter for character data



Example StreamReader


```
static void Main(string[] args)
{
    try
    {
        using(var sr = new StreamReader("file.txt"))
        {
            string line;
            // Read lines from the file until the end of the file is reached.
            while ((line = sr.ReadLine()) != null)
            {
                // Do stuff with line...
            }
        }
    }
    catch (Exception e)
    {
        Console.WriteLine("The file could not be read:");
        Console.WriteLine(e.Message);
    }

}
```

Working with Files

Provides **instance** methods:

FileInfo

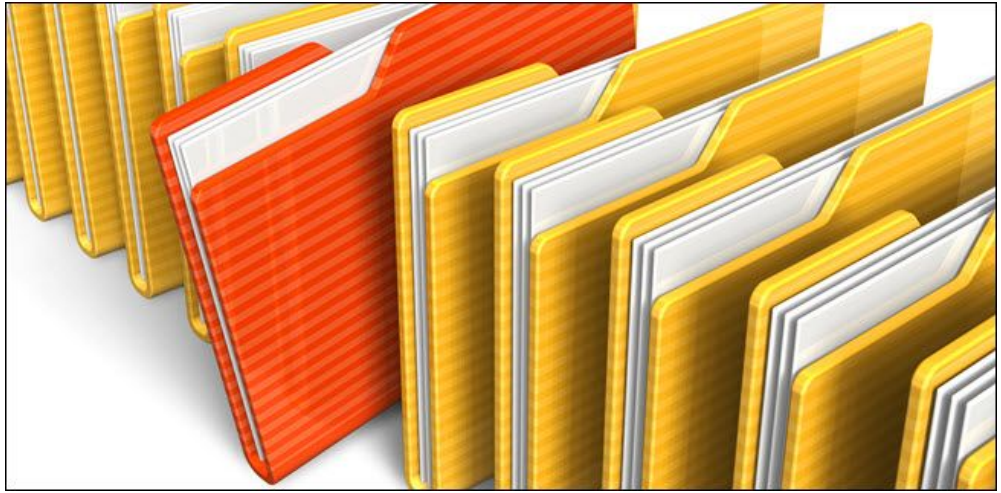
DirectoryInfo

Path

Provides **static** methods:

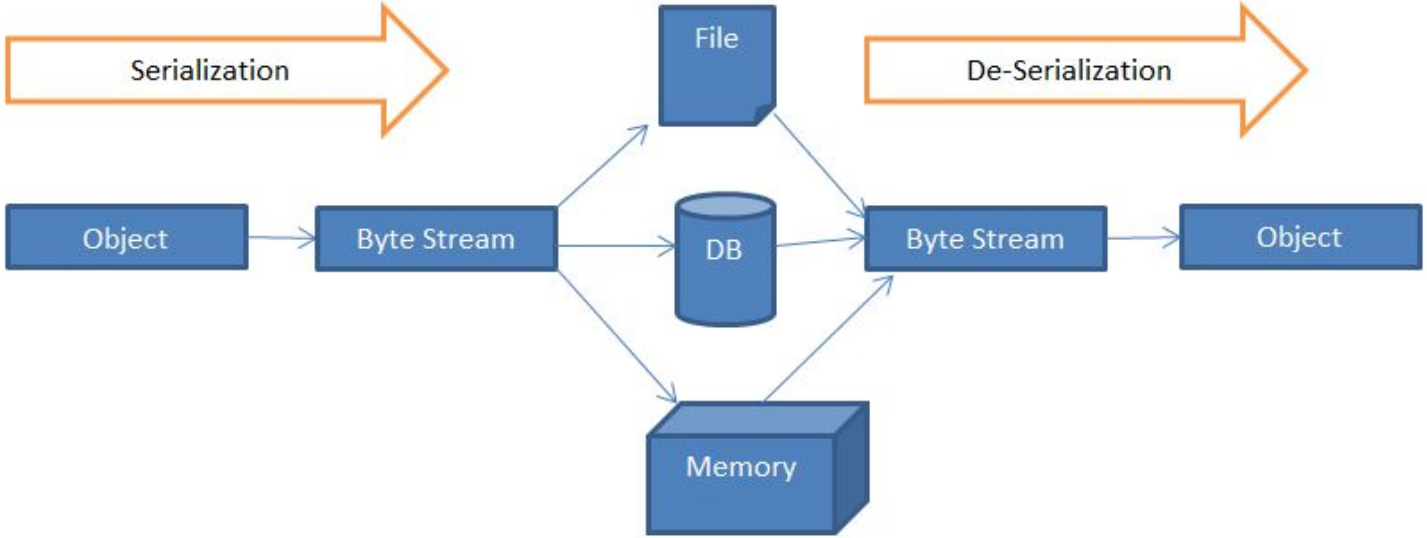
File

Directory



Serialization

Serialization is the process of converting the state of an object into a form that can be **persisted** or transported.



Together, these processes allow data to be easily stored and transferred.

How to Serialize

The easiest way to make a class serializable is to mark it with the Serializable attribute as follows

```
[Serializable]
public class MyObject {
    public int n1 = 0;
    public int n2 = 0;
    public string str = null;
}
```

```
MyObject obj = new MyObject();
obj.n1 = 1;
obj.n2 = 24;
obj.str = "Some String";
IFormatter formatter = new BinaryFormatter();
Stream stream = new FileStream("MyFile.bin", FileMode.Create,
FileAccess.Write, FileShare.None);
formatter.Serialize(stream, obj);
stream.Close();
```

This will write all fields to stream!

How to Deserialize


```
IFormatter formatter = new BinaryFormatter();
Stream stream = new FileStream("MyFile.bin", FileMode.Open,
FileAccess.Read, FileShare.Read);
MyObject obj = (MyObject) formatter.Deserialize(stream);
stream.Close();
// Here's the proof.
Console.WriteLine($"n1: {obj.n1}");
Console.WriteLine($"n2: {obj.n2}");
Console.WriteLine($"str: {obj.str}");
```

JSON Serialization

.NET Supports serialization into various formats (Binary, JSON, XML)

JSON Serialization

How to serialize and deserialize JSON

What is JSON?

https://www.newtonsoft.com ← industry standard for working with JSON

```
dotnet add package Newtonsoft.Json
```

Threading

.NET Core API

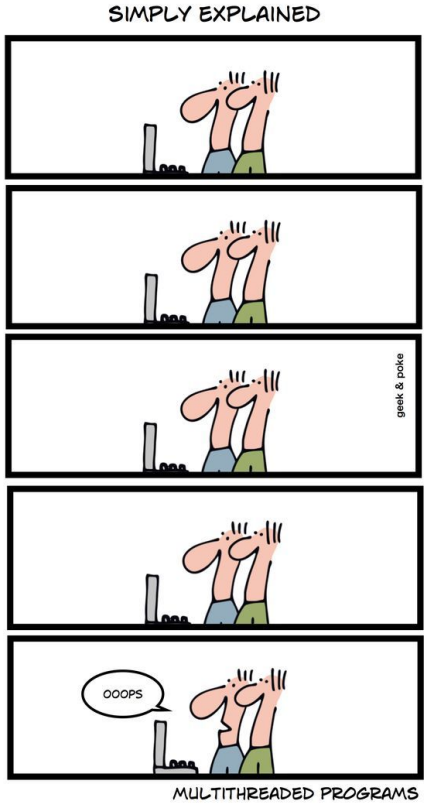
Namespace **System.Threading**

Provides classes and interfaces that enable multithreaded programming

Multithreading - parallel execution of code, leveraging threads

Example Usage

- Separate heavy calculations from UI (to avoid freezes)
- Constantly query external service and notify application if new data arrived
- To avoid stopping processing when waiting for user's input
- Separating processing workflow by threads
- There are a few independent, lightweight tasks that do not intersect



Class Thread

.NET Core API

```
public sealed class Thread
{
    public Thread(ThreadStart start) {}

    public ThreadPriority Priority { get; set; }
    public ThreadState ThreadState { get; }

    public bool IsAlive { get; }
    public bool IsBackground { get; set; }

    public void Start() {}
    public static void Sleep(int time) {}
    public void Suspend() {}
    public void Resume() {}
    public void Join() {}
    public void Abort() {}

    public static Thread CurrentThread { get; }
}
```

constructor with ThreadStart delegate

setting/getting priority

current state

properties liveness, background

methods for controlling thread

gets the currently running thread

ThreadStart, ThreadPriority and ThreadState

.NET Core API

```
public delegate void ThreadStart();

public sealed class Thread
{
    public Thread(ThreadStart start) {}

    public ThreadPriority Priority { get; set; }
    public ThreadState ThreadState { get; }
```

```
public enum ThreadPriority
{
    Highest,
    AboveNormal,
    Normal,
    BelowNormal,
    Lowest
}
```

```
public enum ThreadState
{
    Unstarted,
    Running,
    Background,
    WaitSleepJoin,
    SuspendRequested,
    Suspended,
    AbortRequested,
    Stopped
}
```

Creating a New Thread

Implementing method for **ThreadStart**

```
public static void RunT0 () {
    for (int i = 0; i < 10000; i++) {
        Console.Write ('x');
        Thread.Sleep (100);
    }
}
```

Creating a Thread with **delegate** to method RunT0 and starting it

```
public static void Main(string[] args) {
    //main thread starts a new thread which runs RunT0 method
    Thread t0 = new Thread( new ThreadStart(RunT0));
    t0.Start();
}
```

Thread.Start Method

Task-based Asynchronous Programming

Creating and running tasks implicitly:

```
Parallel.Invoke(() => DoSomeWork(), () => DoSomeOtherWork());
```

Creating and running tasks explicitly:

```
public static void Main()
{
    Thread.CurrentThread.Name = "Main";

    // Create a task and supply a user delegate by using a lambda expression.
    Task taskA = new Task( () => Console.WriteLine("Hello from taskA."));
    // Start the task.
    taskA.Start();

    // Output a message from the calling thread.
    Console.WriteLine("Hello from thread '{0}'.",
                      Thread.CurrentThread.Name);
    taskA.Wait();
}
```

```
// The example displays output like the following:
//       Hello from thread 'Main'.
//       Hello from taskA.
```

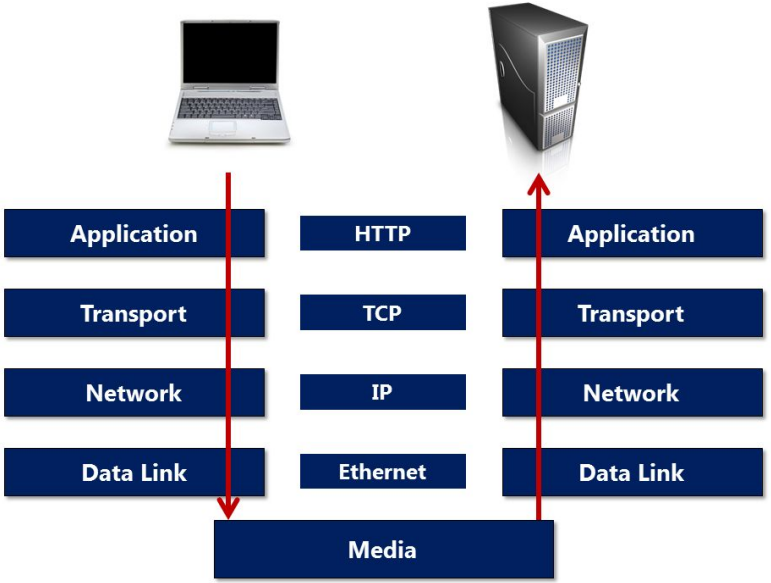
[Task-based async programming]

Networking

Namespace **System.Net**
Supports the implementation of typical client/server applications

System.Net offers implementation of

- Internet protocols, e.g. **TCP**, **UDP**, **HTTP**
- Internet services, e.g. **DNS** (Domain Name System)



System.Net.Sockets offers support for the creation of data streams over networks

Addressing

Addressing is done by classes

- **IPAddress** represents IP address
- **IPEndPoint** represents end point with IP address and port

Example

```
byte[] adr = { 254, 10, 120, 4 };
IPAddress ipAdr = new IPAddress( adr );
// Create a new IPEndPoint with port number 80 (HTTP)
IPEndPoint ep = new IPEndPoint( ipAdr, 80 );
```

DNS (Domain Name System)

DNS offers an “IP into domain name”-mapping service

- **Dns** supports DNS mapping
- **IPHostEntry** is a container class for address information

Example

```
// Get all the addresses of a given DNS name
IPHostEntry host = Dns.GetHostEntry("via.dk");
foreach (IPAddress ip in host.AddressList)
    Console.WriteLine(ip.ToString());
```

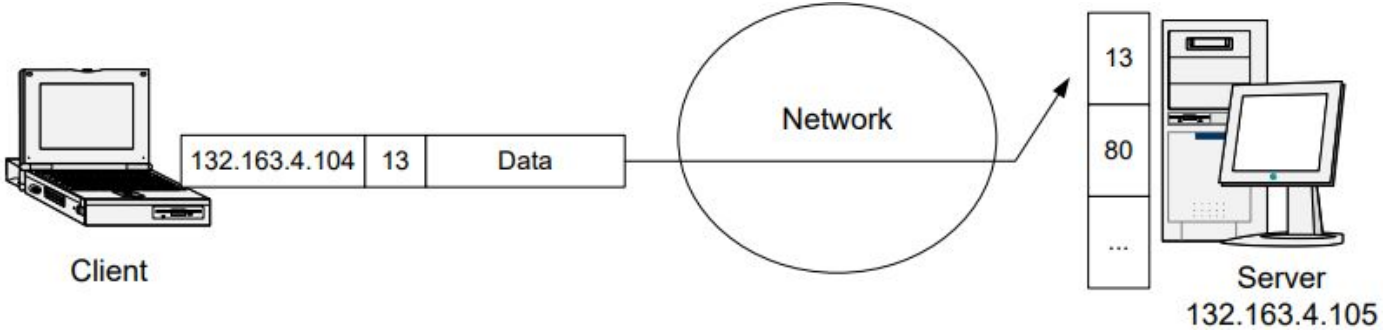
Sockets

- Sockets represent **bidirectional communication** channels, which allow sending and receiving streamed data

- **Client**/**server** architectures
  - client sends request to the server
  - server handles request and
  - sends back response

- Addressing by IP addresses and ports

- Data exchange by **streams**



Creating a Server


```
byte[] adr = { 127, 0, 0, 1 };
IPAddress ipAdr = new IPAddress(adr);
TcpListener listen = new TcpListener(ipAdr, 5000);
listen.Start();

// Wait for connection
TcpClient client = listen.AcceptTcpClient();

// Communicate with client
NetworkStream stream = client.GetStream();
byte[] abyString = Encoding.ASCII.GetBytes("Hi from server");
stream.Write(abyString, 0, 14);
```

Creating a Client


```
byte[] adr = { 127, 0, 0, 1 };
TcpClient client = new TcpClient("127.0.0.1", 5000);

// Connect to end point
client.Connect(new IPEndPoint(new IPAddress(adr), 5000));

// Communicate with server
NetworkStream networkStream = client.GetStream();
byte[] abyString = Encoding.ASCII.GetBytes("Hi from client");
networkStream.Write(abyString, 0, 14);
```

Reflection

Namespace **System.Reflection**

Permits access to meta-information of types at **run-time**

- Getting meta-information about **assemblies**, **modules** and **types**
- Getting meta-information about the **members** of a type
- **Dynamic** creation of instances of a type at run-time
- **Search** for methods and their dynamic invocation at run-time
- Accessing values of properties and fields of an object
- Design of **new types** at runtime (via System.Reflection.Emit)

Reflection Example

C# Program “HelloWorld”

```
namespace Hello {
    using System;
    public class HelloWorld {
        public static void Main (string[] args) {
            Console.WriteLine ("HelloWorld");
        }
        public override string ToString () {
            return "Example HelloWorld";
        }
    }
}
```

Compile and create assembly **`HelloWorld.dll`**

Loading the assembly **`HelloWorld.dll`**

```
Assembly a = Assembly.Load("HelloWorld");

// Print all existing types in a given assembly
Type[] types = a.GetTypes();
foreach (Type t in types)
    System.Console.WriteLine(t.FullName);

// Print all existing methods of a given type
Type hw = a.GetType("Hello.HelloWorld");
MethodInfo[] methods = hw.GetMethods();
foreach (MethodInfo m in methods)
    System.Console.WriteLine(m.Name);
```