Modern Programming Languages - CSC445

Programming in C# Language
Lecture # 3

Course Overview

- Brief introduction to C#.
- Covers the .NET framework, (most of) the C# language and some of the most useful .NET API's.
- Should not be your first programming class.
  - Assume you know C++ and/or Java and basic object-oriented or component-based programming.
- Requires (lots of) practice / reading.
  - C# and .NET cannot be learned thoroughly in this brief course.

Syllabus

- Background, history, CLI, CIL, CLR, CTS, …
- C# Types
  - Primitive types, Classes, Properties, Interfaces, Delegates, Events, Generic types.
- C# language features
  - foreach, yield, events, is/as (type casting), lock.
- Common Interfaces
  - Iterators, equality and comparison
- Base Class Library

Programming in C#

C# History

CSE 4253

Prof. Roger Crawfis

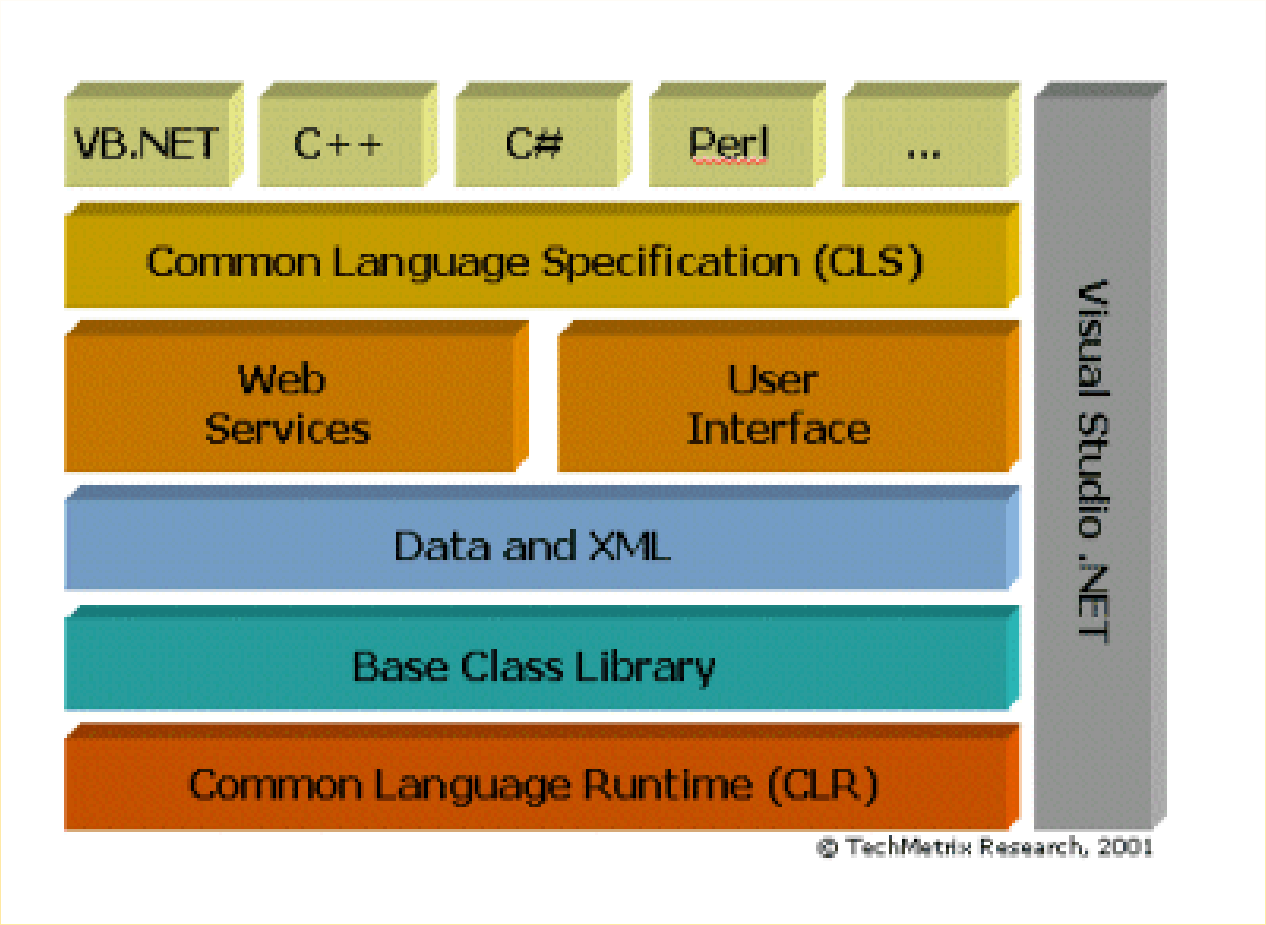
History of C#

- Developed by Microsoft.
- Based on Java and C++, but has many additional extensions.
- Java and C# are both being updated to keep up with each other.
- Cross-development with Visual Basic, Visual C++, F#, IronPython, and many other .NET languages.
  - See: http://en.wikipedia.org/wiki/List_of_CLI_languages

Classification of C#

- Wikipedia.org definition.
  - Object-oriented.
  - Primarily imperative or procedural.
    - LINQ adds some functional programming language capabilities.
  - Structured (as opposed to monolithic).
  - Strongly typed.
  - ISO and ECMA standardized.

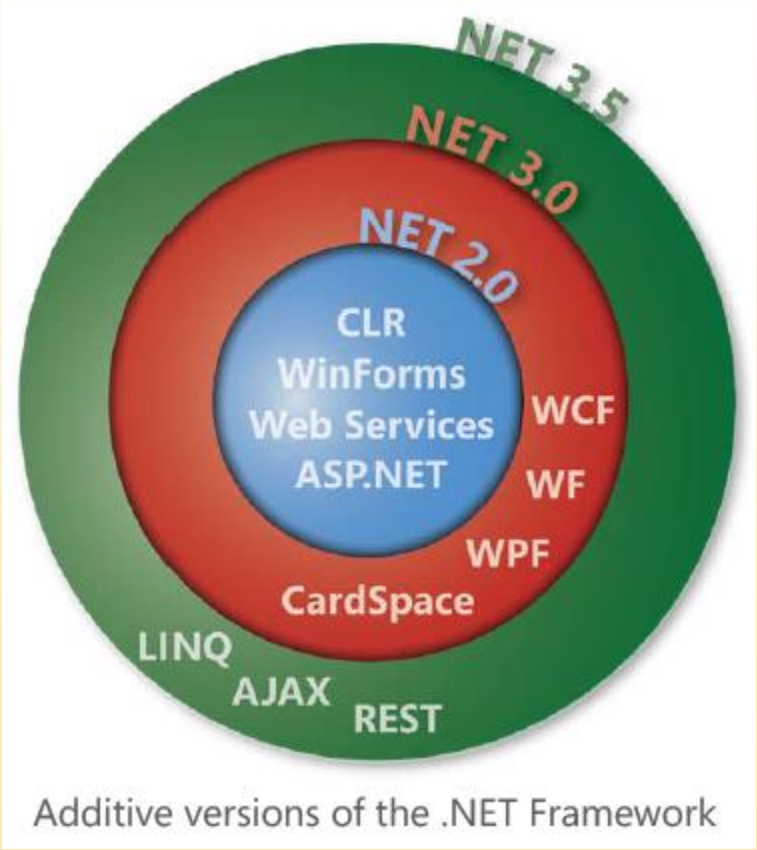
Microsoft's .NET Technologies



The Class Libraries

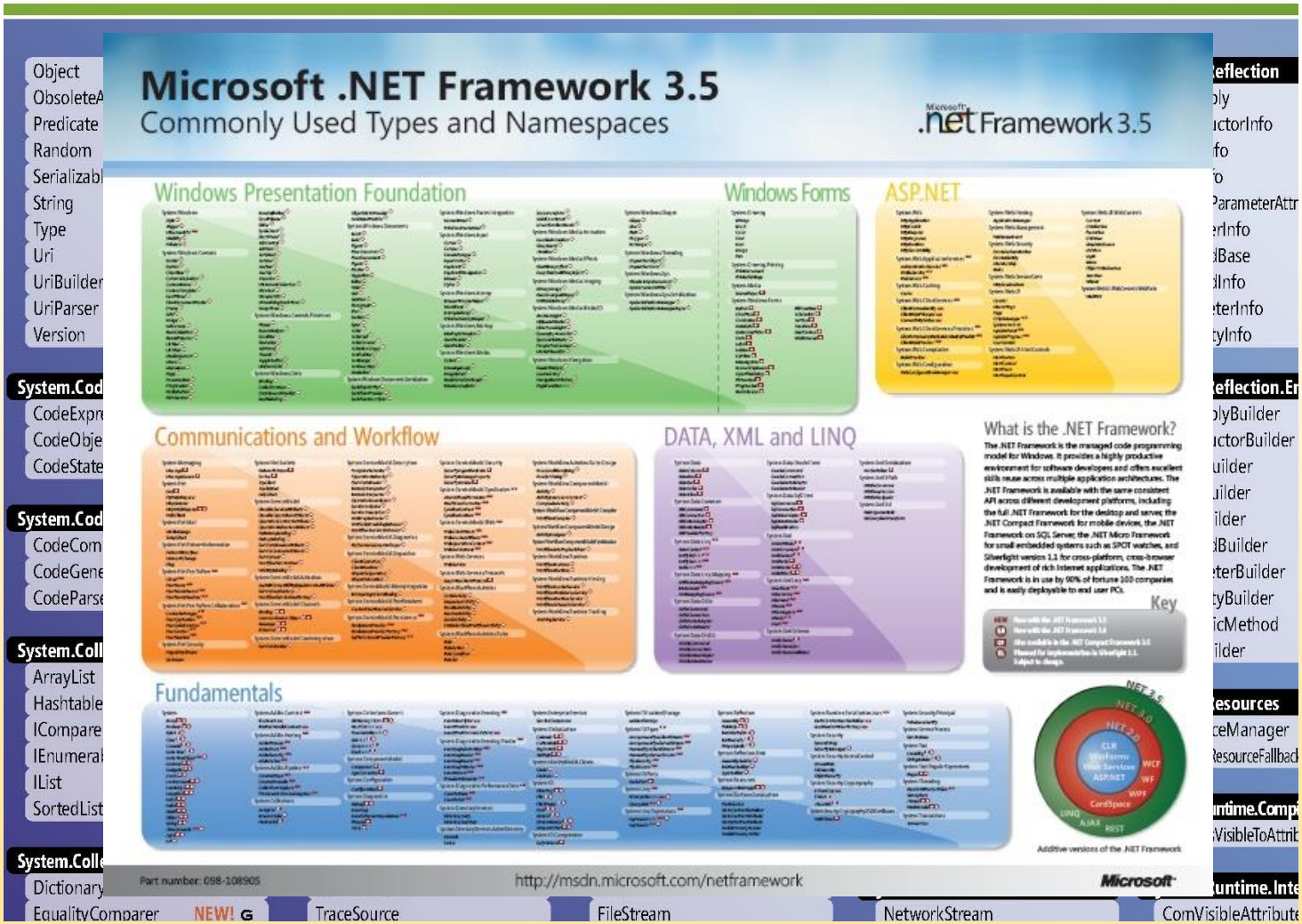
- The common classes that are used in many programs
  - System.Console.WriteLine
  - XML, Networking, Filesystem, Crypto, containers
  - Can inherit from many of these classes
- Many languages run on .NET framework
  - C#, C++, J#, Visual Basic
  - even have Python (see IronPython)

.NET History



Additive versions of the .NET Framework

The Class Libraries



IDE's and CLI Implementations

- Visual C# http://www.microsoft.com/express/2008/
  - in MSDNAA
    - must be version 2008: we need C# 3.0
- Mono: http://www.go-mono.com
  - Open Source for Linux: not quite at 2.0
- Rotor: http://msdn.microsoft.com/net/sscli
  - Shared Source for Windows (through 2.0)
  - Use to work on BSD / OS X, too
- Portable.NET: http://www.dotgnu.org
  - yet another open source implementation

Programming in C#

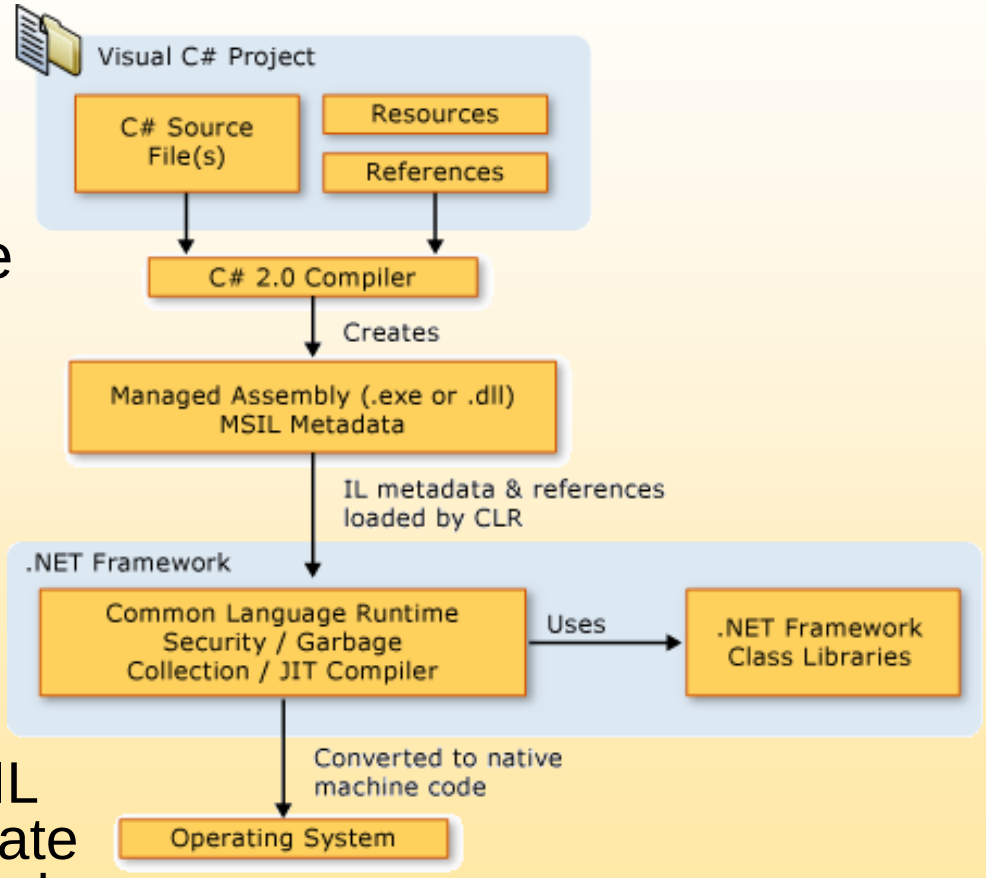
CLR, CLI, oh my!

CSE 459.24

Prof. Roger Crawfis

CLR and JIT compiling.

- C#, like Java, is executed indirectly through an abstract computer architecture called the CLR.
  - CLR => Common Language Runtime.
  - Abstract, but well defined.
- C# programs are compiled to an IL.
  - Also called MSIL, CIL (Common Intermediate Language) or bytecode.



http://msdn2.microsoft.com/en-us/library/z1zx9t92(VS.80).aspx

CLR and JIT compiling.

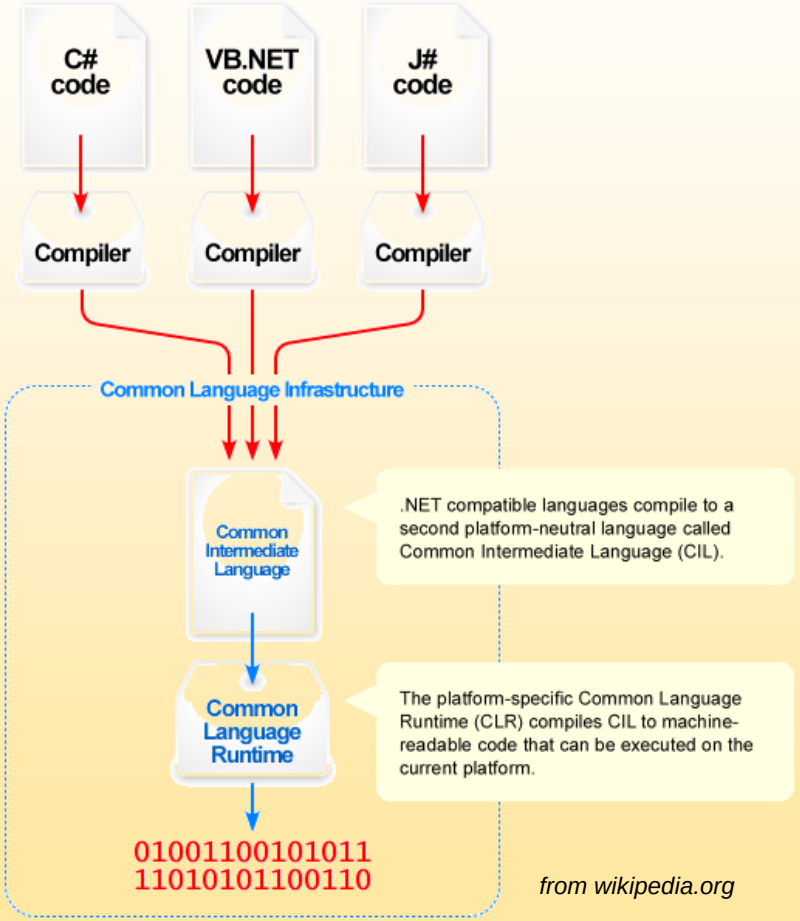
- The CLR transforms the CIL to assembly instructions for a particular hardware architecture.
  - This is termed jit'ing or Just-in-time compiling.
  - Some initial performance cost, but the jitted code is cached for further execution.
  - The CLR can target the specific architecture in which the code is executing, so some performance gains are possible.

CLR and JIT compiling.

- All .NET languages compile to the same CIL.
- Each language actually uses only a subset of the CIL.
- The least-common denominator is the *Common Language Specification* (CLS).
- So, if you want to use your C# components in Visual Basic you need to program to the CLS.

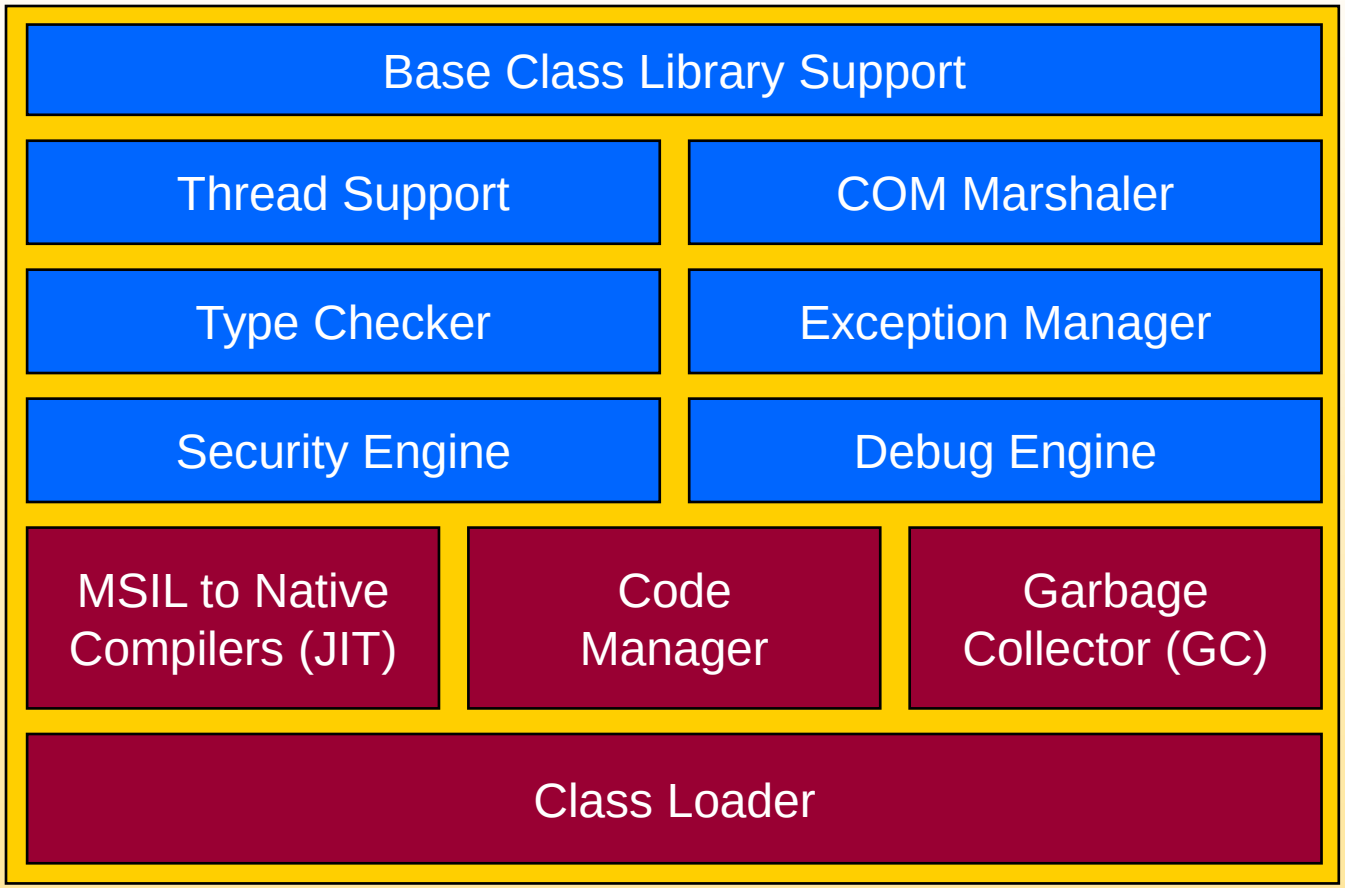
CLR versus CLI.

- CLR is actually an **implementation** by Microsoft of the CLI (Common Language Infrastructure) .
- CLI is an open *specification*.
- CLR is really a platform specific implementation.



from wikipedia.org

The CLR Architecture



From MSDN

Common Language Infrastructure.

- CLI allows for cross-language development.
- Four components:
  - Common Type System (CTS)
  - Meta-data in a language agnostic fashion.
  - Common Language Specification – behaviors that all languages need to follow.
  - A Virtual Execution System (VES).

Common Type System (CTS)

- A specification for *how* types are *defined* and how they *behave*.
  - no syntax specified
- A type can contain zero or more members:
  - Field
  - Method
  - Property
  - Event
- We will go over these more throughout the quarter.

Common Type System (CTS)

- CTS also specifies the rules for visibility and access to members of a type:
  - Private
  - Family
  - Family and Assembly
  - Assembly
  - Family or Assembly
  - Public
- We will go over these more throughout the quarter.

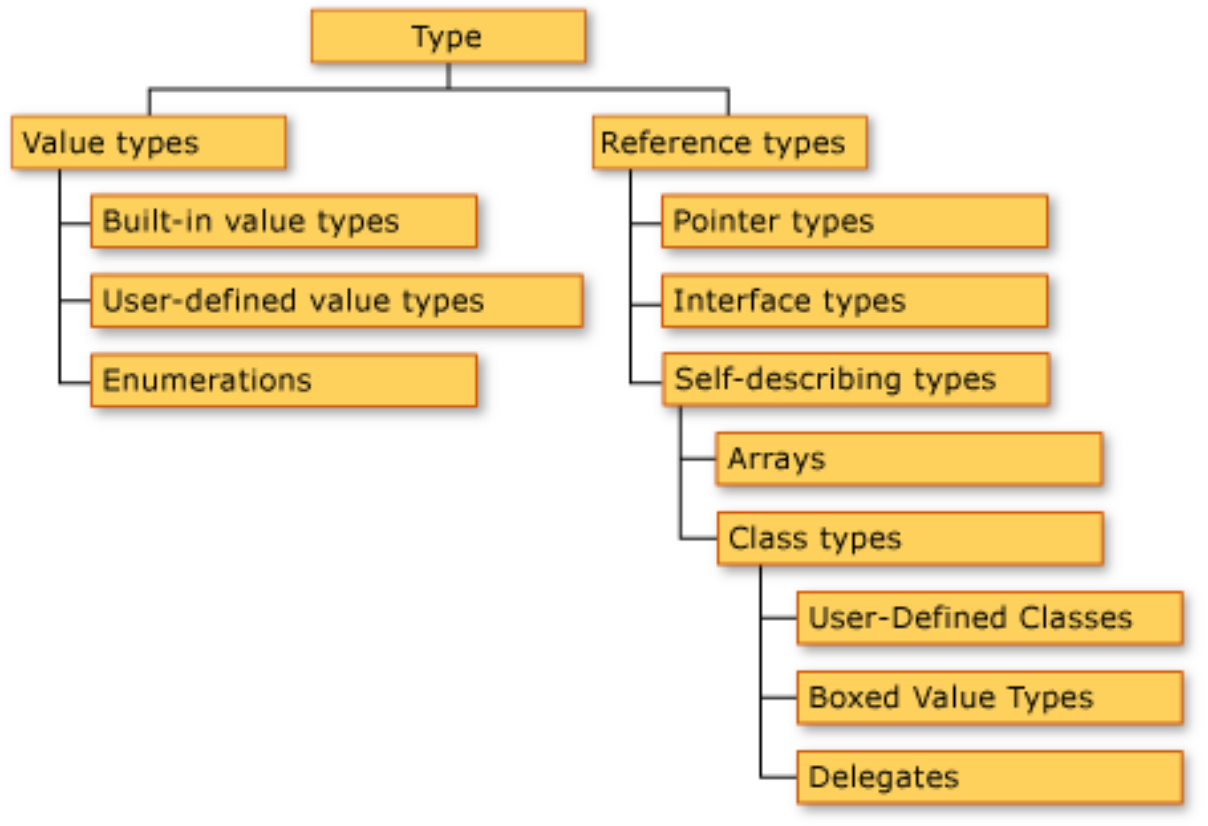
Common Type System (CTS)

- Other rules
  - Object life-time
  - Inheritance
  - Equality (through System.Object)

Common Type System (CTS)

- Languages often define aliases
- For example
  - CTS defines System.Int32 – 4 byte integer
  - C# defines *int* as an alias of System.Int32
  - C# aliases System.String as *string*.

Common Type System (CTS)

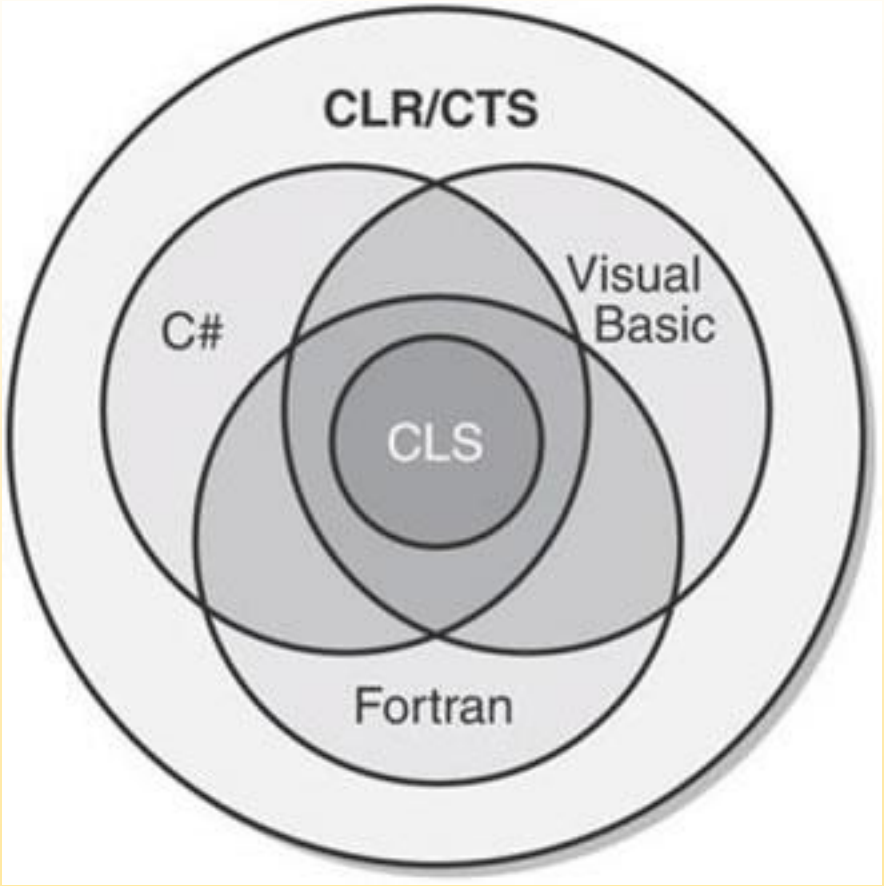


From MSDN

Common Language System

- A specification of language features
  - how methods may be called
  - when constructors are called
  - subset of the types in CTS which are allowed
- For example
  - Code that takes UInt32 in a public method
  - UInt32 is not in the CLS
- Can ***mark*** classes as CLS-compliant
  - not marked is assumed to mean not compliant

CLS versus CLR



CLR via C#, *Jeffrey Richter*

Built-in Types

C#	*CTS type (FCL name)*	*CLS compliant*
int	System.Int32	yes
uint	System.UInt32	no
sbyte	System.SByte	no
byte	System.Byte	yes
short	System.Int16	yes
ushort	System.UInt16	no
long	System.Int64	yes
ulong	System.UInt64	no
float	System.Single	yes
double	System.Double	yes
decimal	System.Decimal	yes
char	System.Char	yes
string	System.String	yes
object	System.Object	yes

Blittable types

- Most of these types are *blittable*, meaning their memory layout is consistent across languages and hence, support interoperability.
- The types bool, char, object and string are not and must be ***Marshaled*** when using these between languages.
- Single dimensional arrays of blittable types are also blittable.

Programming in C#

Assemblies

CSE 494R

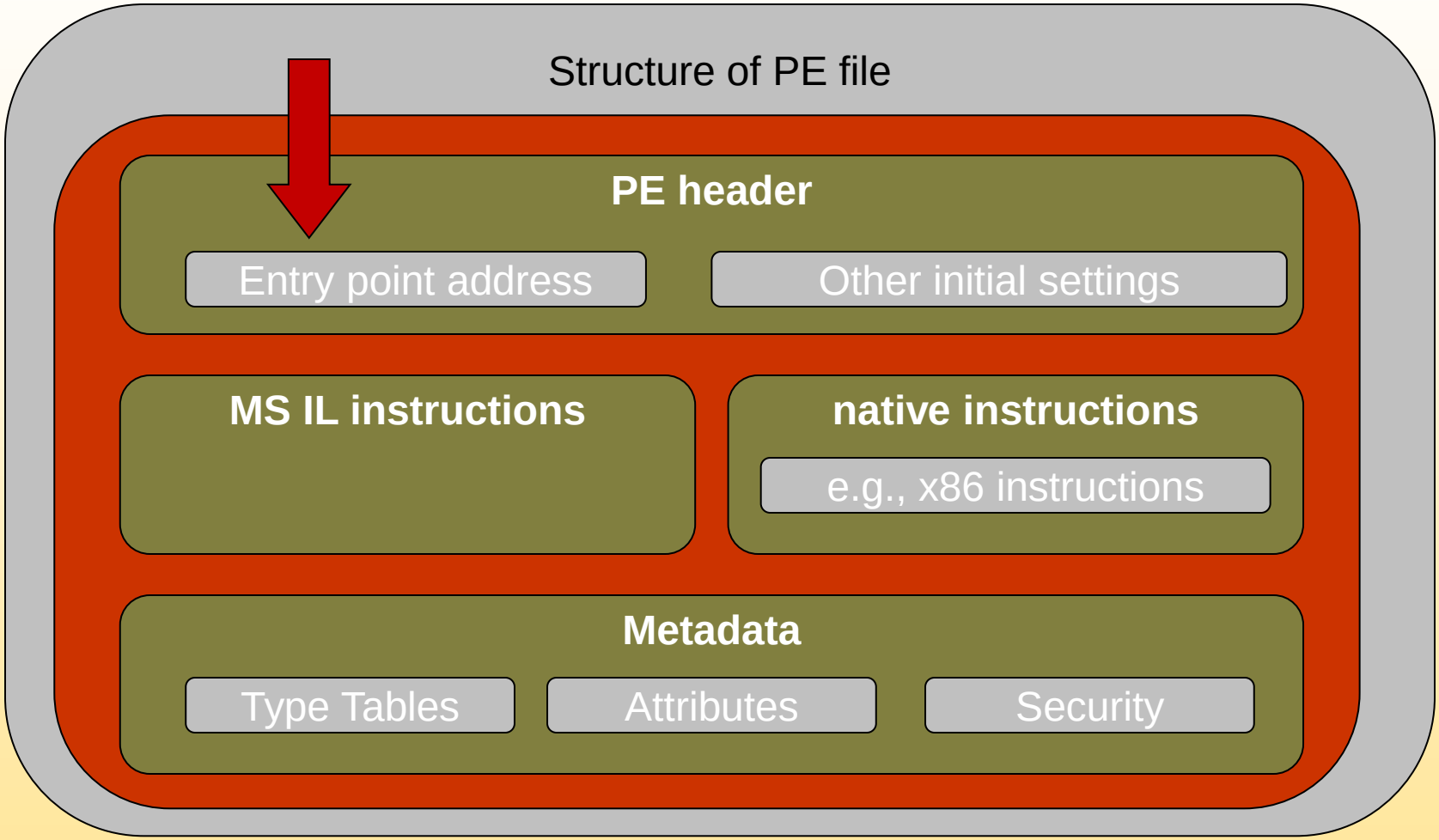
(proposed course for 459 Programming in C#)

Prof. Roger Crawfis

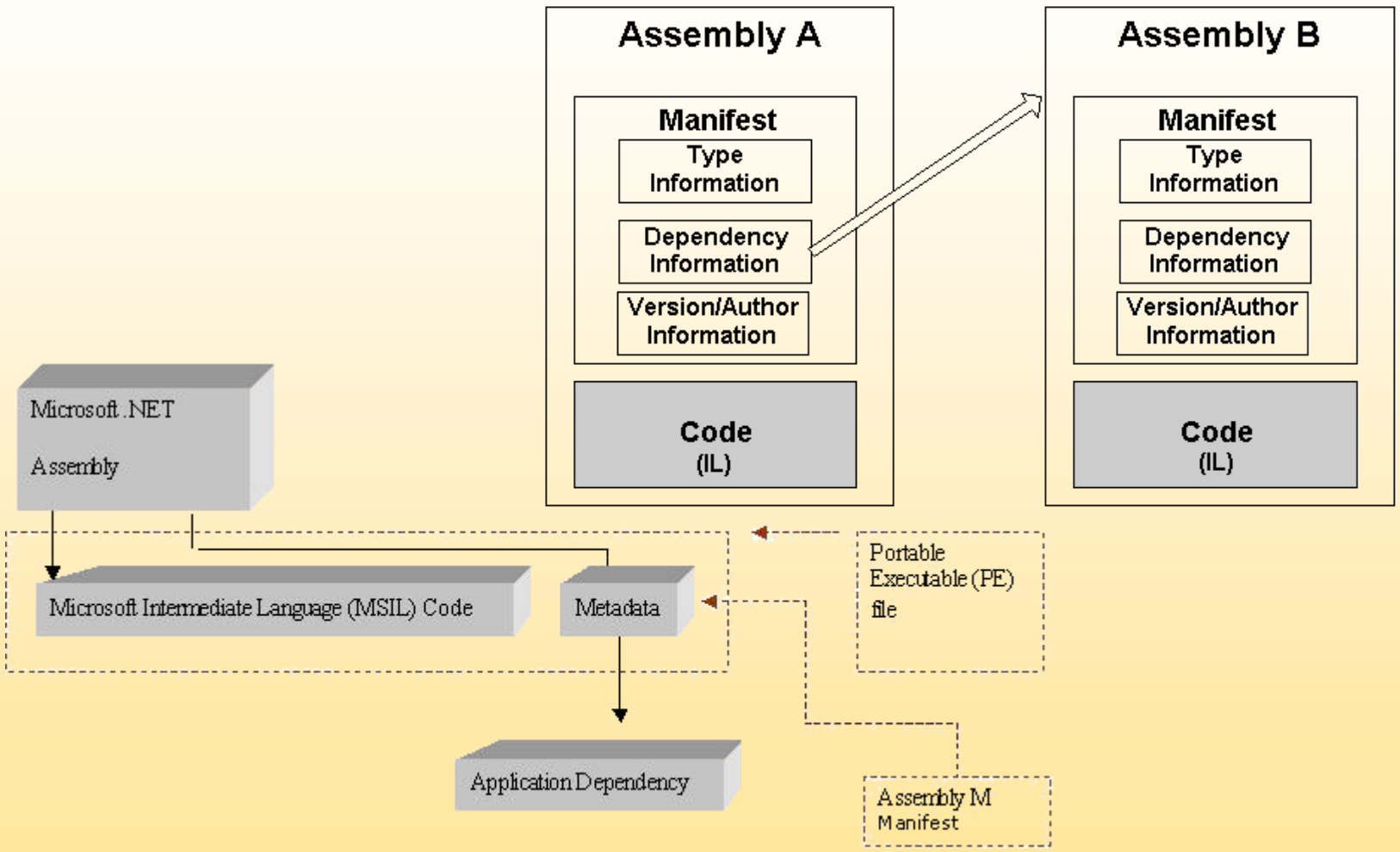
Assemblies

- Code contained in files called “assemblies”
  - code and ***metadata***
  - .exe or .dll as before
  - Executable needs a class with a “Main” method:
    - `public static void Main(string[] args)`
  - types
    - local: local assembly, not accessible by others
    - shared: well-known location, can be ***GAC***
    - strong names: use crypto for signatures
      - then can add some versioning and trust

PE executable file



Manifests and Assemblies



First C# Program

```
using System;
namespace Test
{
    class ExampleClass
    {
        static void Main()
        {
            System.Console.WriteLine("Hello, world!");
        }
    }
}
```

Constructions of Note

- `using`
  - like import in Java: bring in namespaces
- `namespace`
  - disambiguation of names
  - like Internet hierarchical names and C++ naming
- `class`
  - like in C++ or Java
  - single inheritance up to object

Constructions of Note

- static void Main()
  - Defines the entry point for an assembly.
  - Four different overloads – taking string arguments and returning int's.
- Console.Write(Line)
  - Takes a formatted string: "Composite Format"
  - Indexed elements: e.g., {0}
    - can be used multiple times
    - only evaluated once
  - {index [,alignment][:formatting]}