

Microsoft® .net™

Principles and Architecture

Agenda

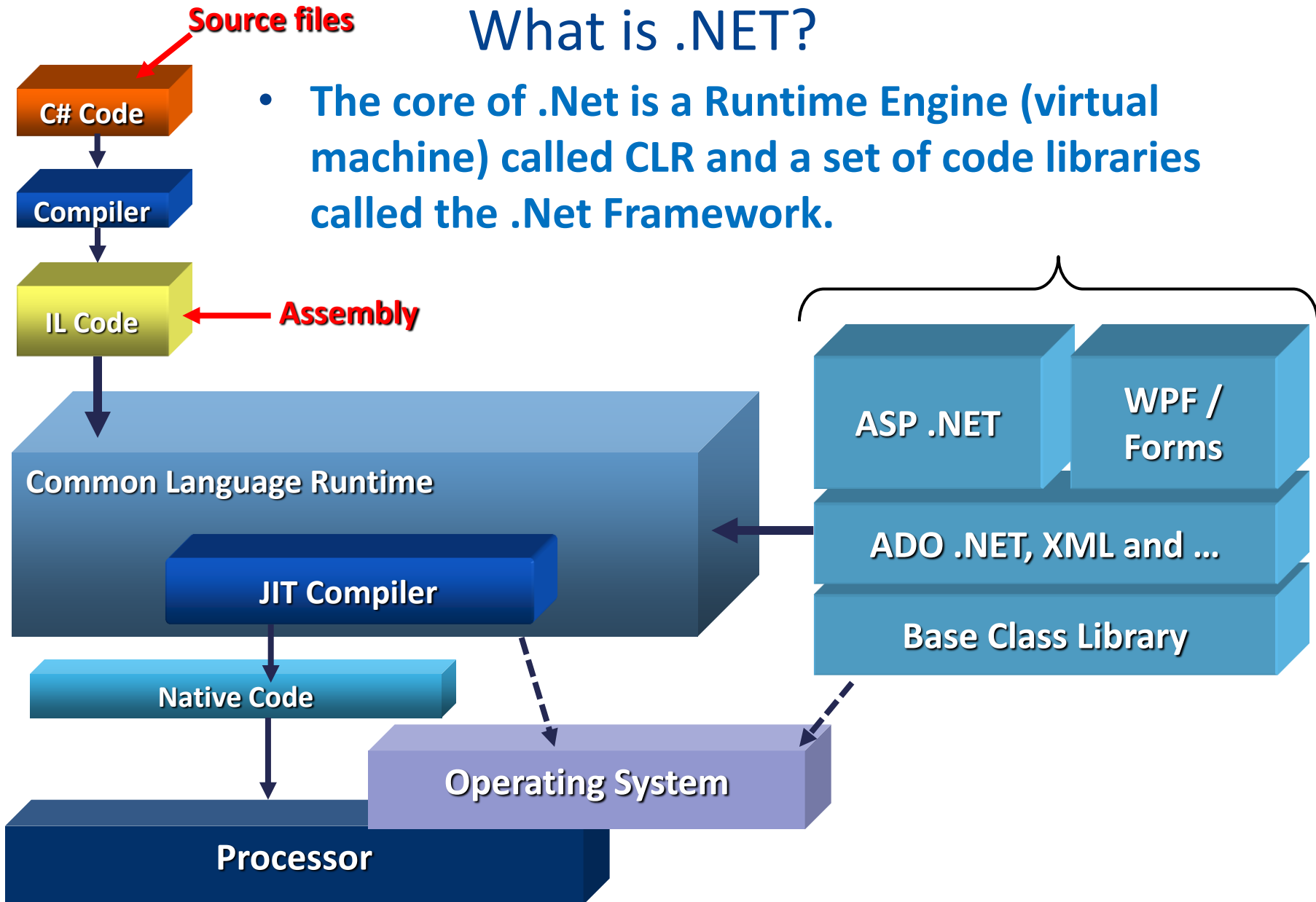
- What is .Net?
- Architecture
- Assemblies
- Garbage Collection

Læringsmål:

Redegøre for principperne i .Net frameworket og dets overordnede arkitektur.

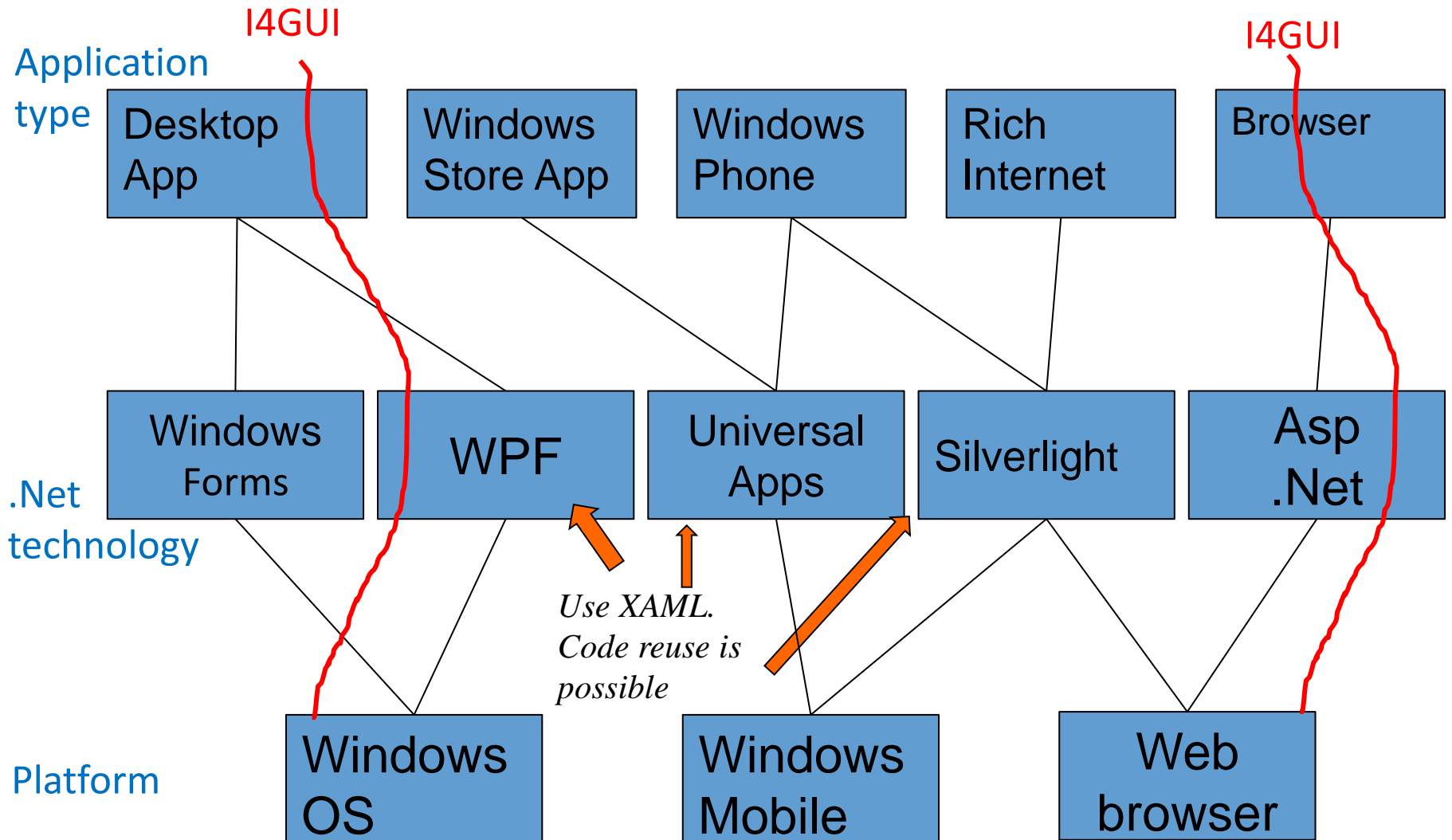
What is .NET?

- The core of .Net is a Runtime Engine (virtual machine) called CLR and a set of code libraries called the .Net Framework.



WPF, Windows Phone and Silverlight

Unified programming model



.NET on Small Devices

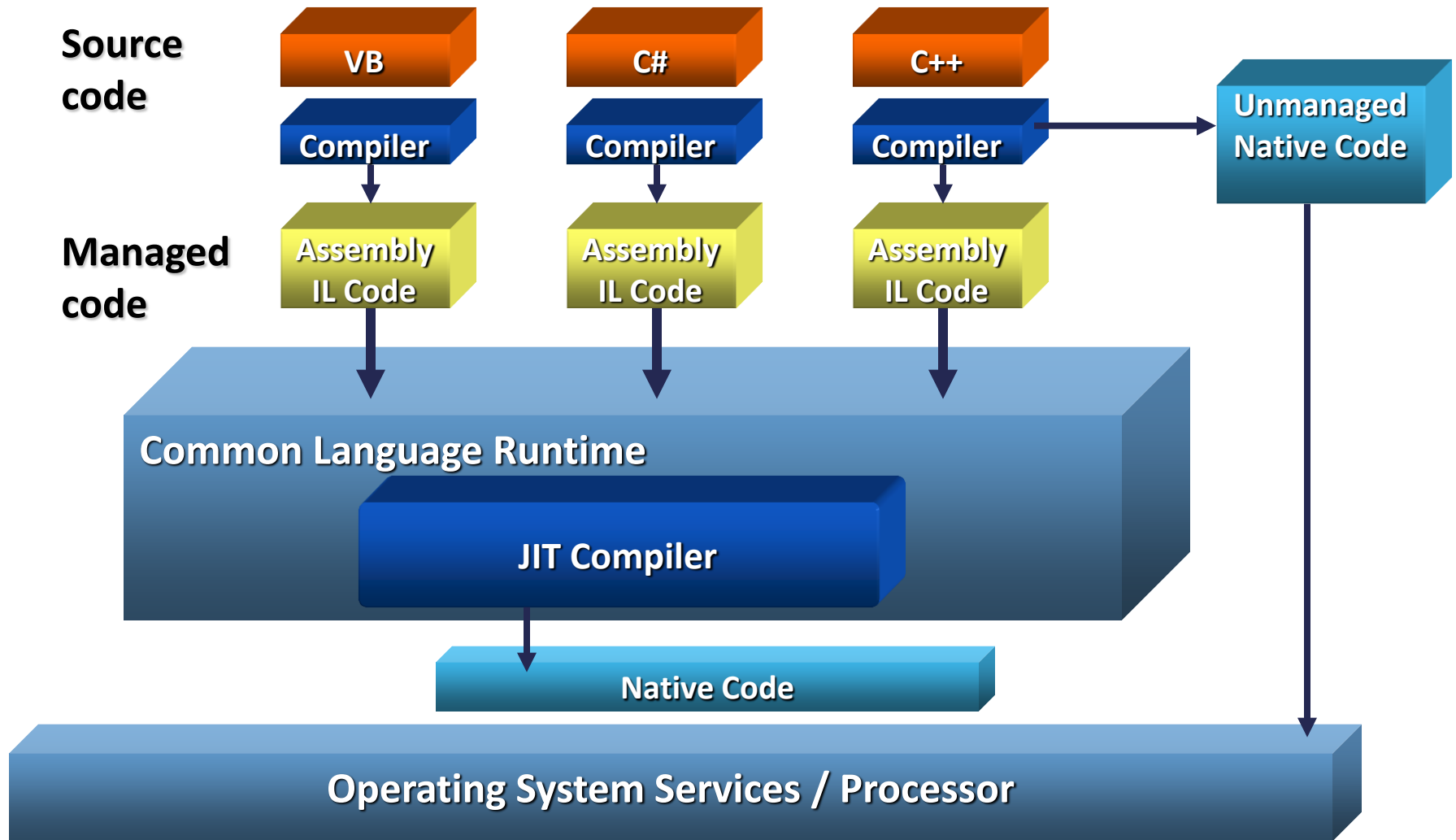
Ikke her i I4GUI, men det i lærer her kan genbruges på de håndholdte

- Other Interfaces
 - .NET Compact Framework
 - Windows Mobile
 - .NET Micro Framework



Architectural Overview of the .NET Framework

.Net Framework Execution Model

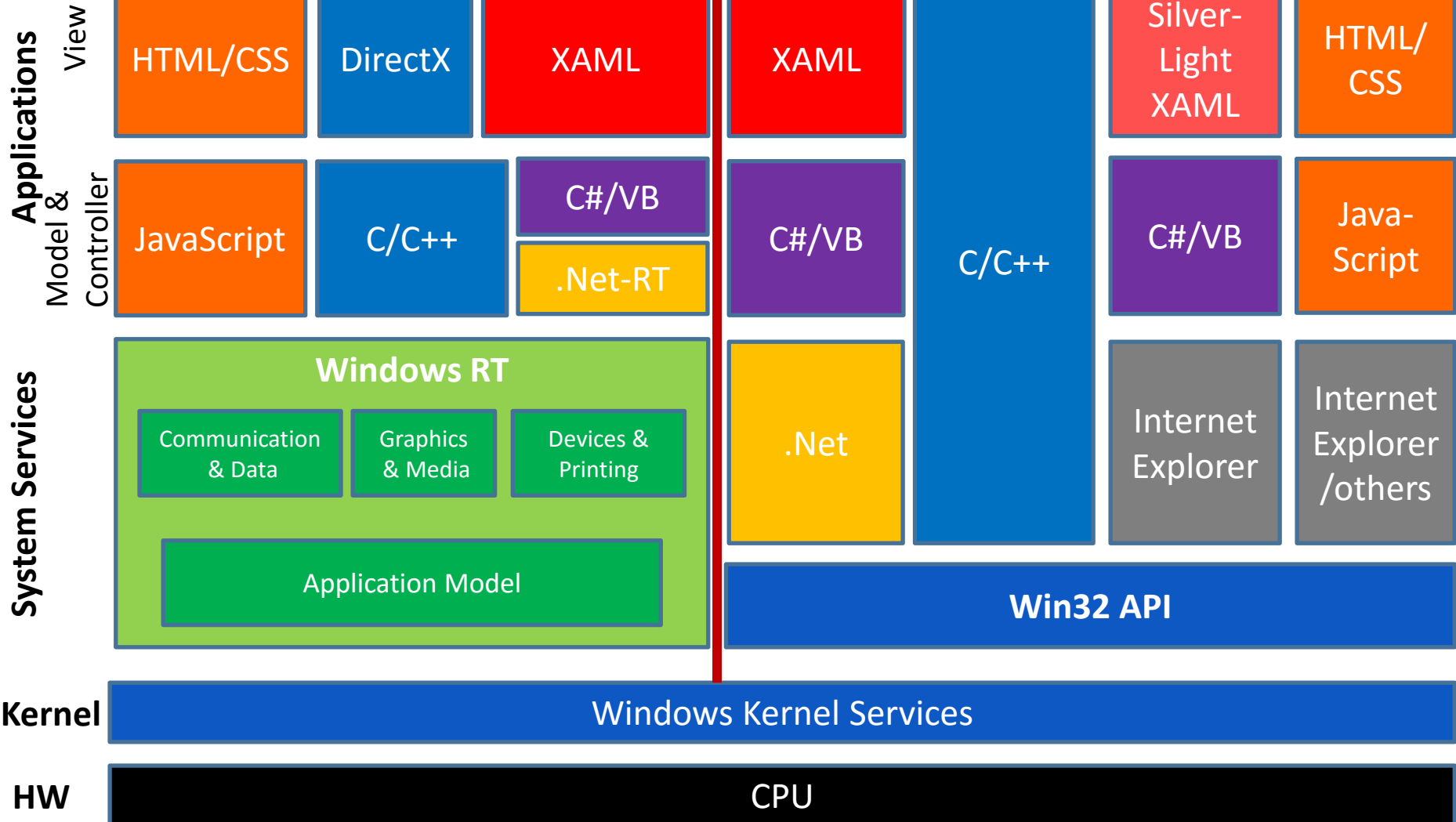


Windows 10 Application Types

Windows Store Apps

Desktop Apps

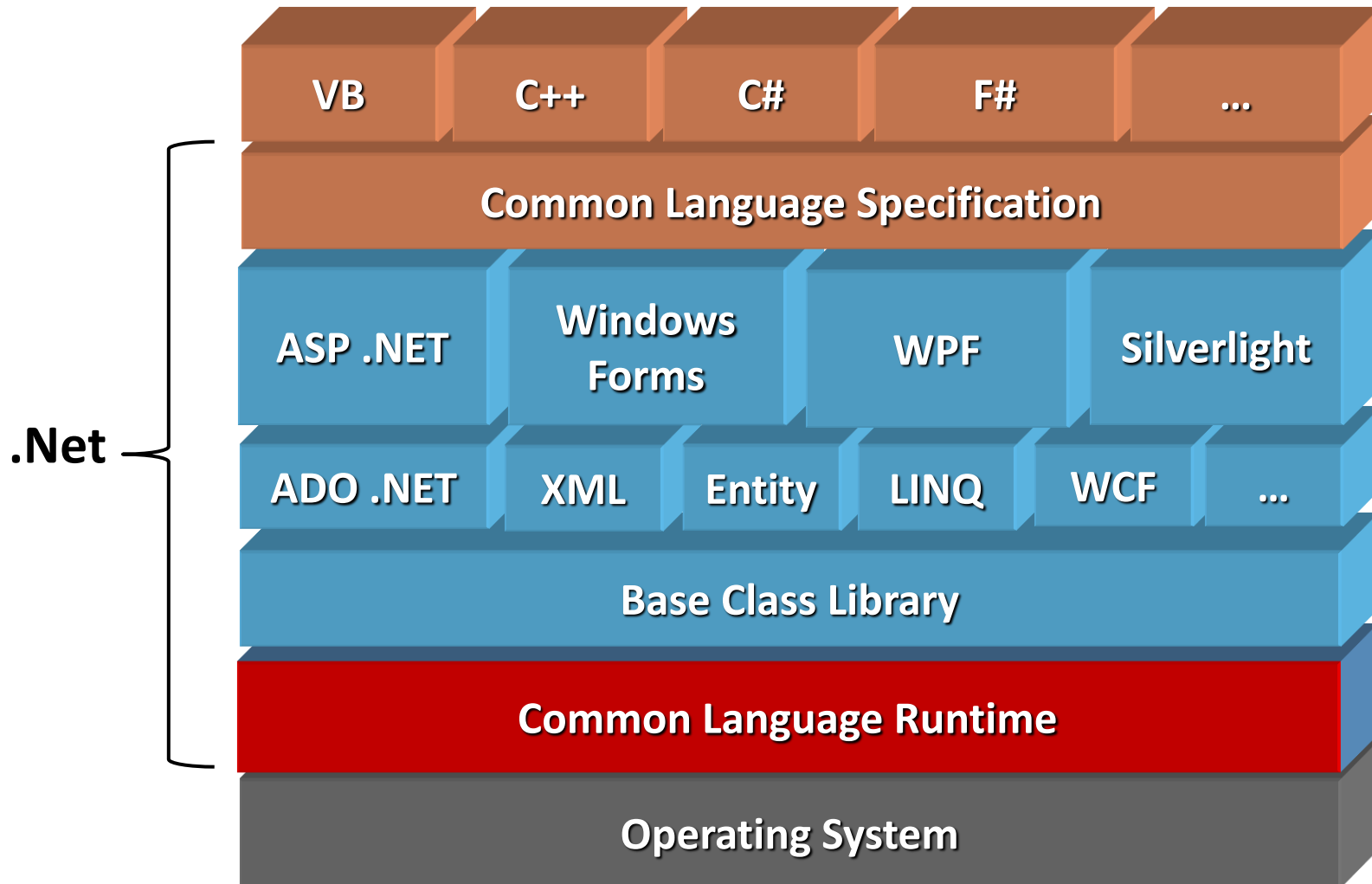
Web Apps



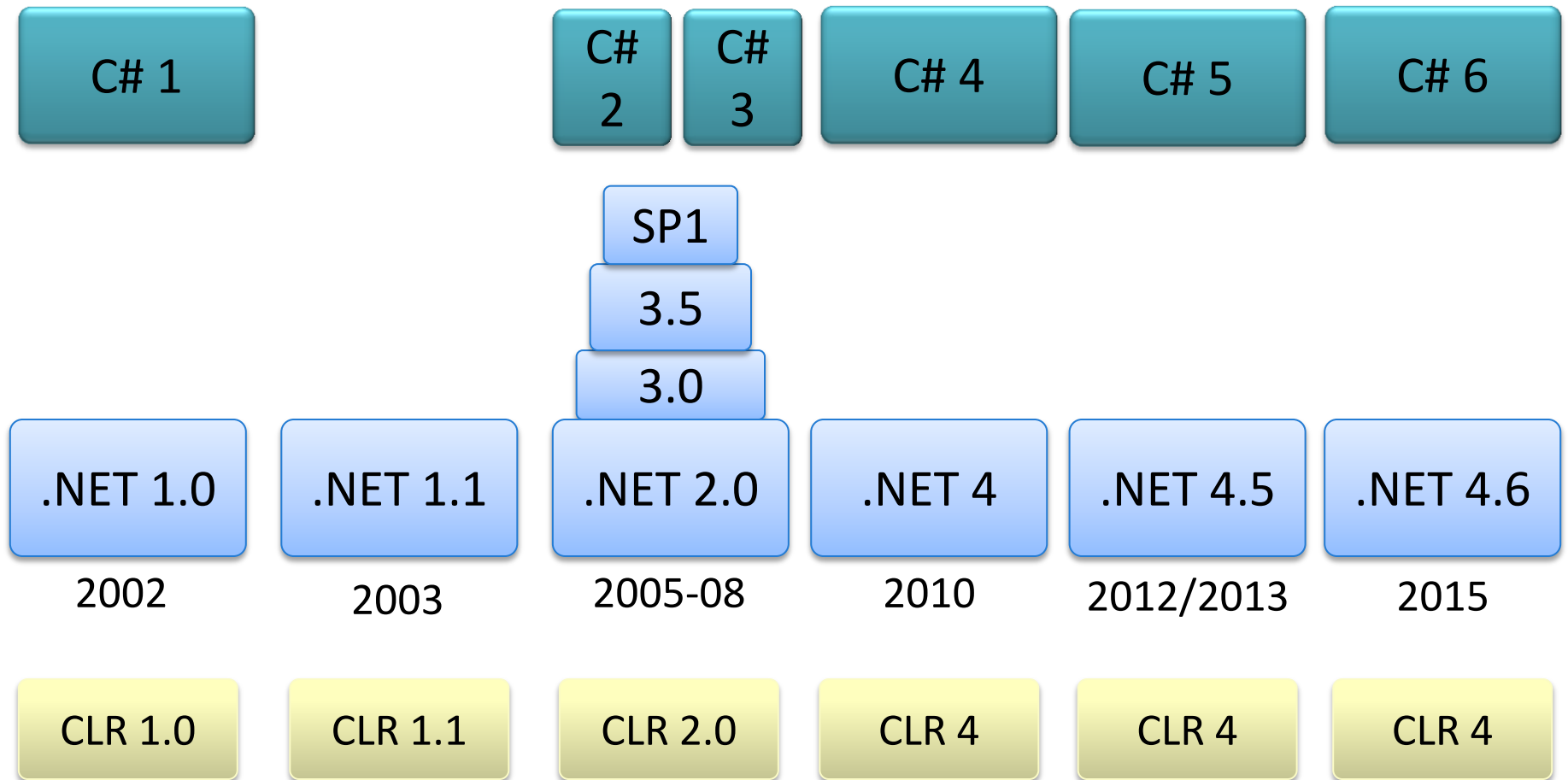
.NET Native

- Typically, apps that target the .NET Framework are compiled to intermediate language (IL)
 - At run time, the just-in-time (JIT) compiler translates the IL to native code
- .NET Native compiles Windows Store apps directly to native code
- .NET Native offers these advantages:
 - Consistently speedy startup times
 - Fast execution times

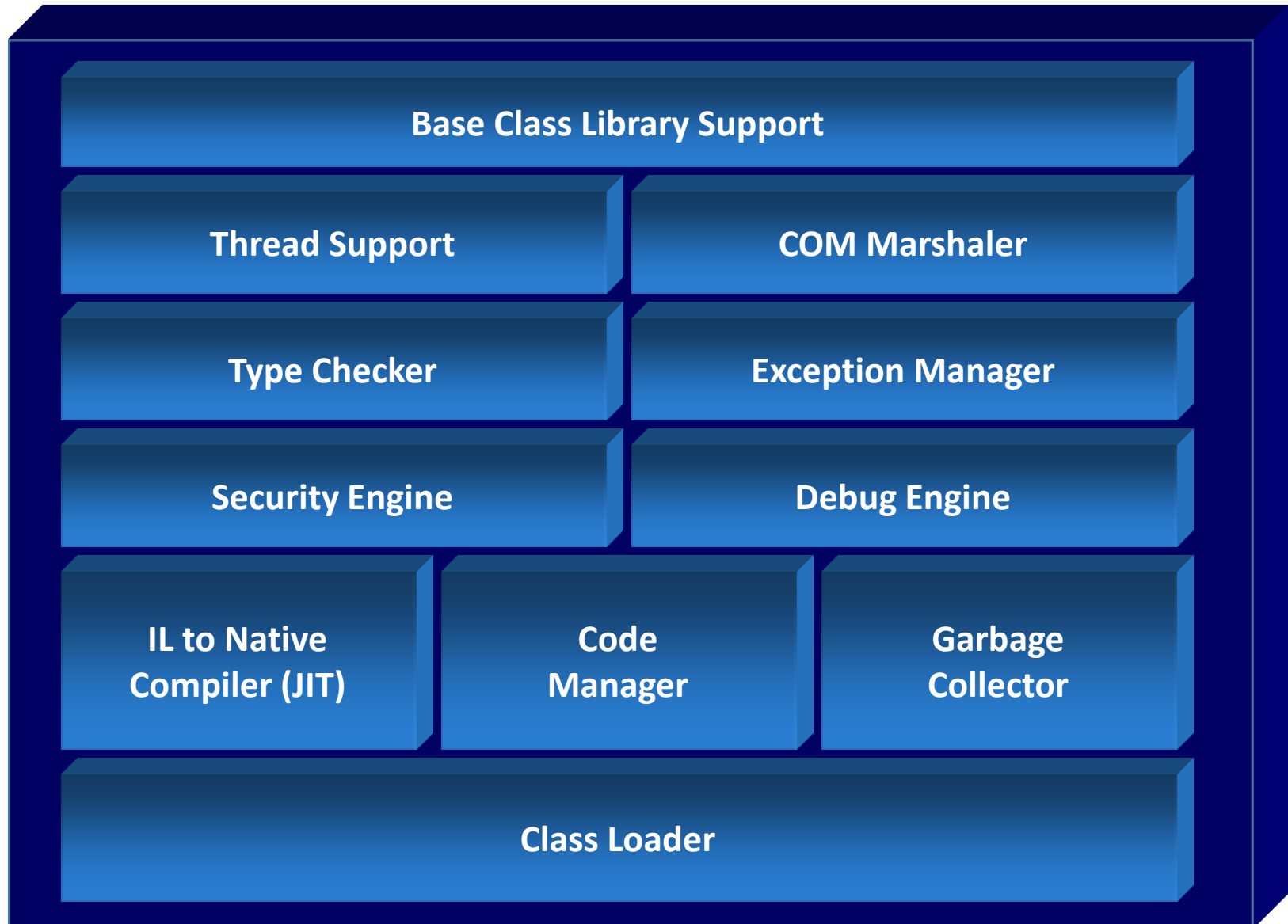
.NET Framework Architecture



A Look Back...



Common Language Runtime



.NET Aim: Robust and Secure

- Automatic lifetime management
 - All .NET objects are garbage collected
 - No stray pointers, no problem with circular references
 - **Multi-generational mark-and-compact GC**
 - Concurrent, self configuring, dynamically tuned
- Exception handling
 - Error handling is a 1st class concept
 - No error return codes – only exceptions
 - Pass exceptions across components & languages

.NET Aim: Robust and Secure

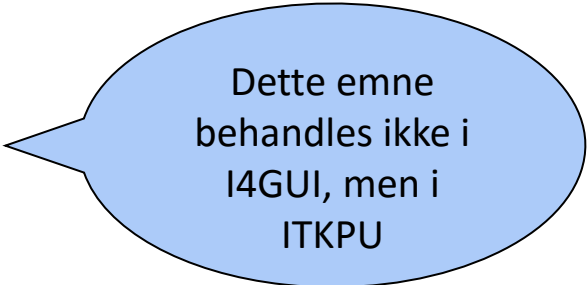
- **Native code**
 - IL (Intermediate Language) ~ CIL (C for Common) ~ MSIL (MS for Microsoft)
 - No interpreter
 - Install-time (Ngen) or run-time IL to native compilation (JIT)
- **Code correctness and type-safety**
 - **IL can be verified to guarantee type-safety**
 - **No unsafe casts**
 - **no uninitialized variables**
 - **no out-of-bounds array indexing**
- **Evidence-based security**
 - Based on origin of code, user, or combination
 - Extensible permissions

.NET Aim: Simplify Deployment

- Assemblies
 - The unit of deployment, versioning, and security
 - Self-describing through manifest
- Zero-impact install
 - Applications and components can be shared or private
- Side-by-side execution
 - Multiple versions of the same component can co-exist, even in the same process

.NET Aim: Seamless Integration

- Interoperability in .Net
 - .NET classes can be used as a COM class
 - COM classes can be imported as .NET classes
- PInvoke
 - Can call DLL entry points (exported functions) in unmanaged DLL's
- C++ managed extensions



Dette emne
behandles ikke i
I4GUI, men i
ITKPU

.NET Aim: Make It Simple To Use

- Organization
 - Code organized in hierarchical namespaces and classes
- Unified type system
 - Eg: one string type!
 - Everything is an object
 - Reference Types or Value Types (primitive types)
- Component Oriented
 - Properties, methods, events, and attributes are first class constructs
 - Design-time functionality

.NET Aim: Factored And Extensible

- The Framework is not a “black box”
- Many .NET class are available for you to extend through inheritance
- Cross-language inheritance!

.NET Aim: Multi-language Platform

- The freedom to choose language
 - All features of .NET platform available to any .NET programming language
 - Application components can be written in multiple languages
- Highly leveraged tools
 - Debuggers, profilers, code coverage analyzers, etc. Work for all languages

Languages

- The .NET Platform is Language Neutral
 - All .NET languages are first class players
- Common Language Specification
 - Consumer Languages (Script languages): Can use the .NET Framework
 - Extender Languages: Can extend the .NET Framework
- Microsoft are providing
 - VB, C++, C#, F# (+ Iron Python, + ...)
- Third-parties offers
 - APL, COBOL, Pascal, Eiffel, Haskell, Perl, Python, Scheme, Smalltalk + ?

.NET is Portable?

- Much of .NET is ECMA standardized
- Available on a variety of platforms:
 - various flavors of Windows (including CE and Mobile versions)
 - Linux / OS X:
 - *Mono* project <http://www.mono-project.com/>
 - [MonoMac](#) for building Cocoa applications on OSX using Mono
 - Unix:
 - *dotGNU* (much less mature) <http://www.dotgnu.org/>
 - FreeBSD / OS X:
 - *Rotor / SSCLI*
 - Microsoft's shared-source release of .NET for research purposes
 - <http://msdn.microsoft.com/net/sscli>
 - iPhone, iPad and Android
 - Xamarin <http://xamarin.com/platform>

.NET Core

- Is a modular version of the .NET Framework designed to be portable across platforms
- Is a subset of the full .NET framework but it provides key functionality to implement the app features you need
- Is released through NuGet in smaller assembly packages
 - Rather than one large assembly that contains most of the core functionality, .NET Core is made available as smaller feature-centric packages
- Is open-source and accept contributions

Main use:

- To build a ASP.Net Web server that also runs on Mac OS and Linux
- To build cross platform mobil apps that runs on ios, Android and Windows Phone (requires use of Xamarin too)

ASSEMBLIES

Assemblies

- Unit of deployment
 - One or more files, independent of packaging
 - Self-describing via metadata and manifest
- Versioning
 - Captured by compiler
- Security boundary
 - Assemblies are granted permissions
 - Methods can demand proof that a permission has been granted to entire call chain
- Mediate type import and export
 - Types named are relative to assembly

Applications

- Applications are configurable units
 - Consists of one or more assemblies
 - Application-specific files or data
- Assemblies are located based on...
 - Their logical name and
 - The application that loads them
- Applications can have private versions of assemblies
 - Private version preferred over shared
- Shared assemblies must be installed in Global Assembly Cache (a subfolder named assembly in the Windows folder)

A Single File Application

Foo.exe

Assembly



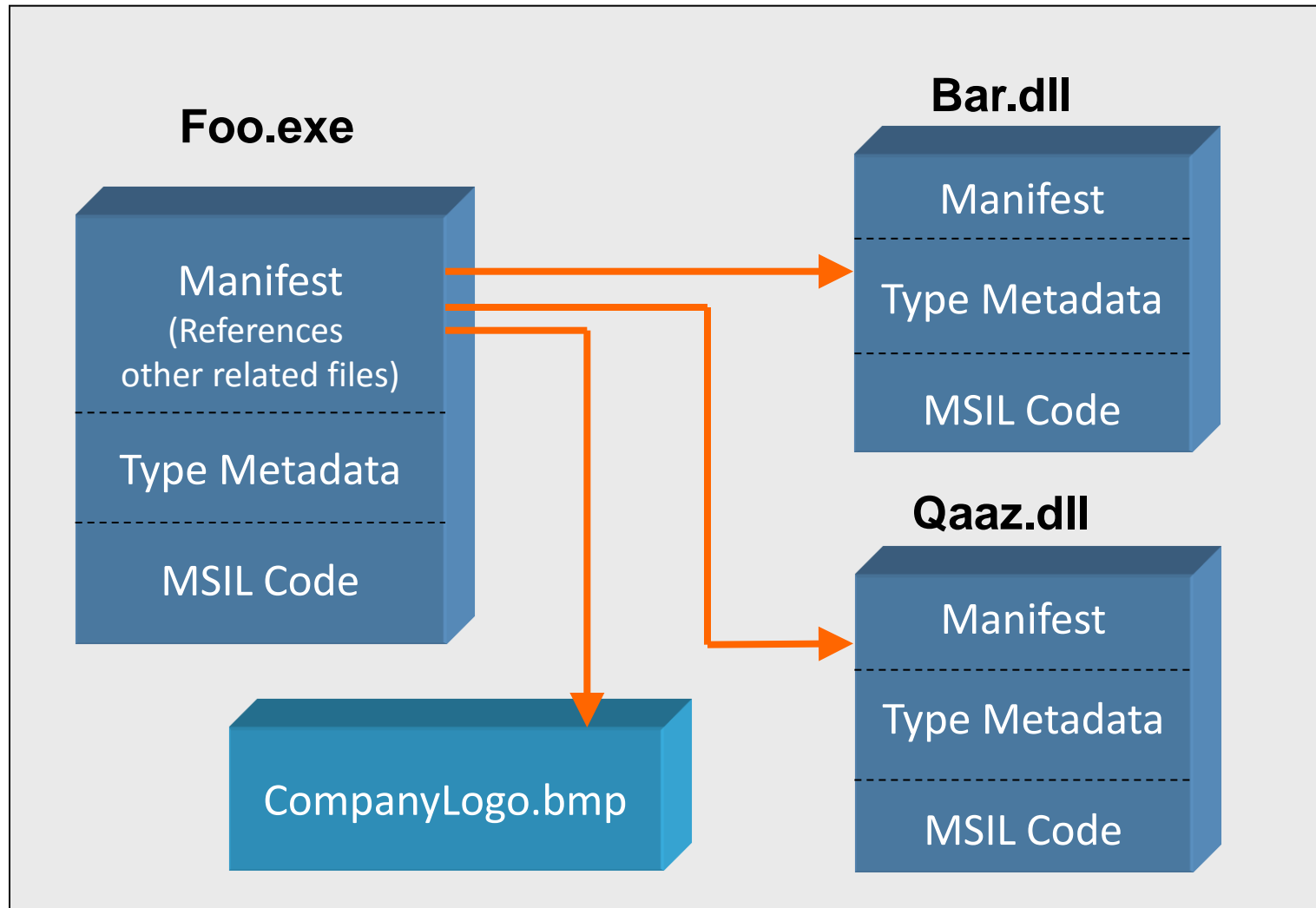
Manifest

Type Metadata

MSIL Code

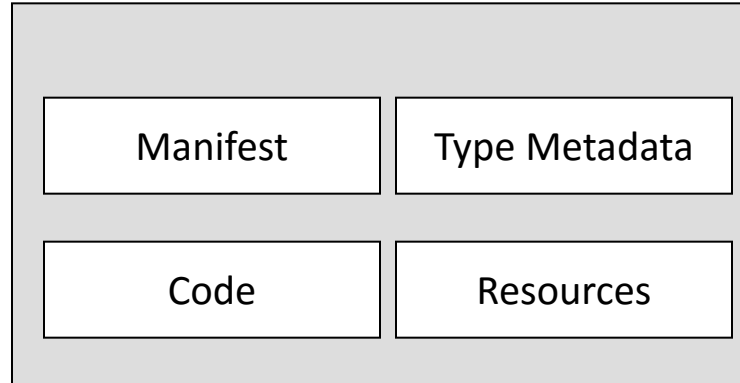
(Optional) Resources

A Multi File Application

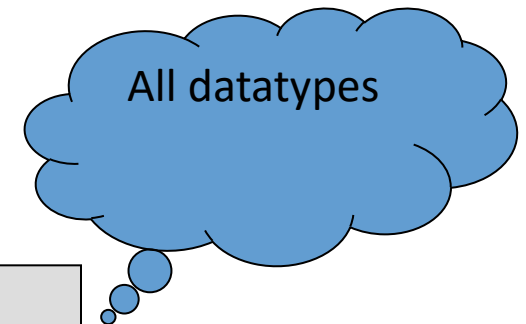
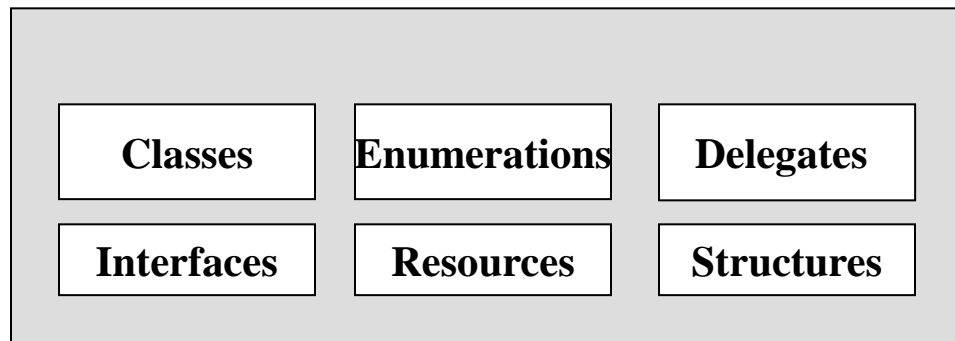


Two conceptual 'views'

Physical view of an Assembly



Logical view of an Assembly



Garbage Collection

AKA
Managed Heap

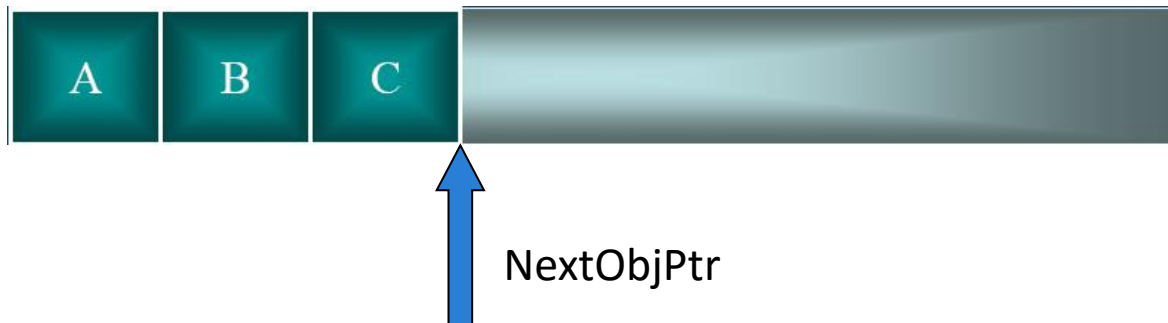
<http://channel9.msdn.com/posts/Maoni-Stephens-CLR-45-Server-Background-GC>

Memory Overview

- Every program uses memory
 - Files, memory buffers, screen space, network connections, database resources, and so on.
- A class identifies some type of resource
- Usual programming paradigm
 1. Allocate memory for resource (new operator)
 2. Initialize memory to make resource usable (constructor)
 3. Use the resource (access members) – repeat as necessary
 4. Tear-down the resource (destructor)
 5. Free the memory
- Common problems
 - Forgetting to free memory → **memory leak**
 - Using memory after freeing it → **memory corruption**
- These bugs are worse than most other bugs
 - Consequence and timing is unpredictable

Solution

- **Garbage Collection** makes these bugs a thing of the past.
- All reference types are allocated on the managed heap
 - Your code never frees an object.
 - The GC frees objects when they are no longer reachable.
- Each process gets its own managed heap
 - Virtual address space region.
- The new operator always allocates objects at the end.
- If heap is full, a GC occurs
 - Reality: GC occurs when generation 0 is full.



The GC Algorithm Overview

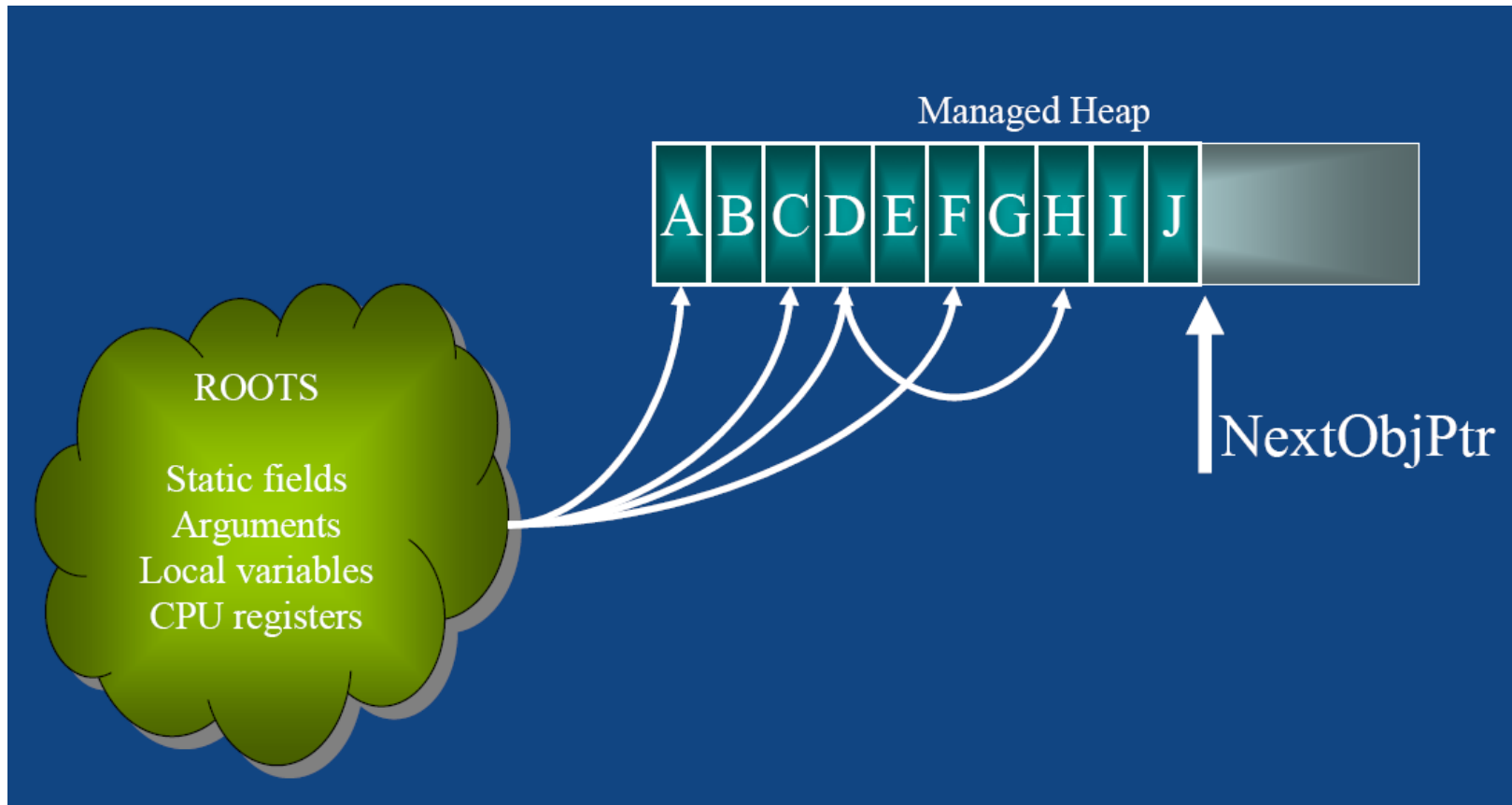
- Allocating objects:
 - new operator always allocates objects at the end
 - Almost as fast as a stack allocation
 - Much faster than unmanaged new/malloc/HeapAlloc
 - Large objects are allocated from a special heap
 - Large objects aren't moved in memory
 - Large objects are $\geq 85,000$ bytes (*may change*)
- Garbage Collection:
 - Marking:
 - Objects referred to by app's variables (the Roots) are marked.
 - Compacting:
 - Marked objects are shifted down over unmarked objects
 - If all objects are marked, no compacting occurs
 - new throws OutOfMemoryException

This algorithm is called: “*Mark and Compact*”

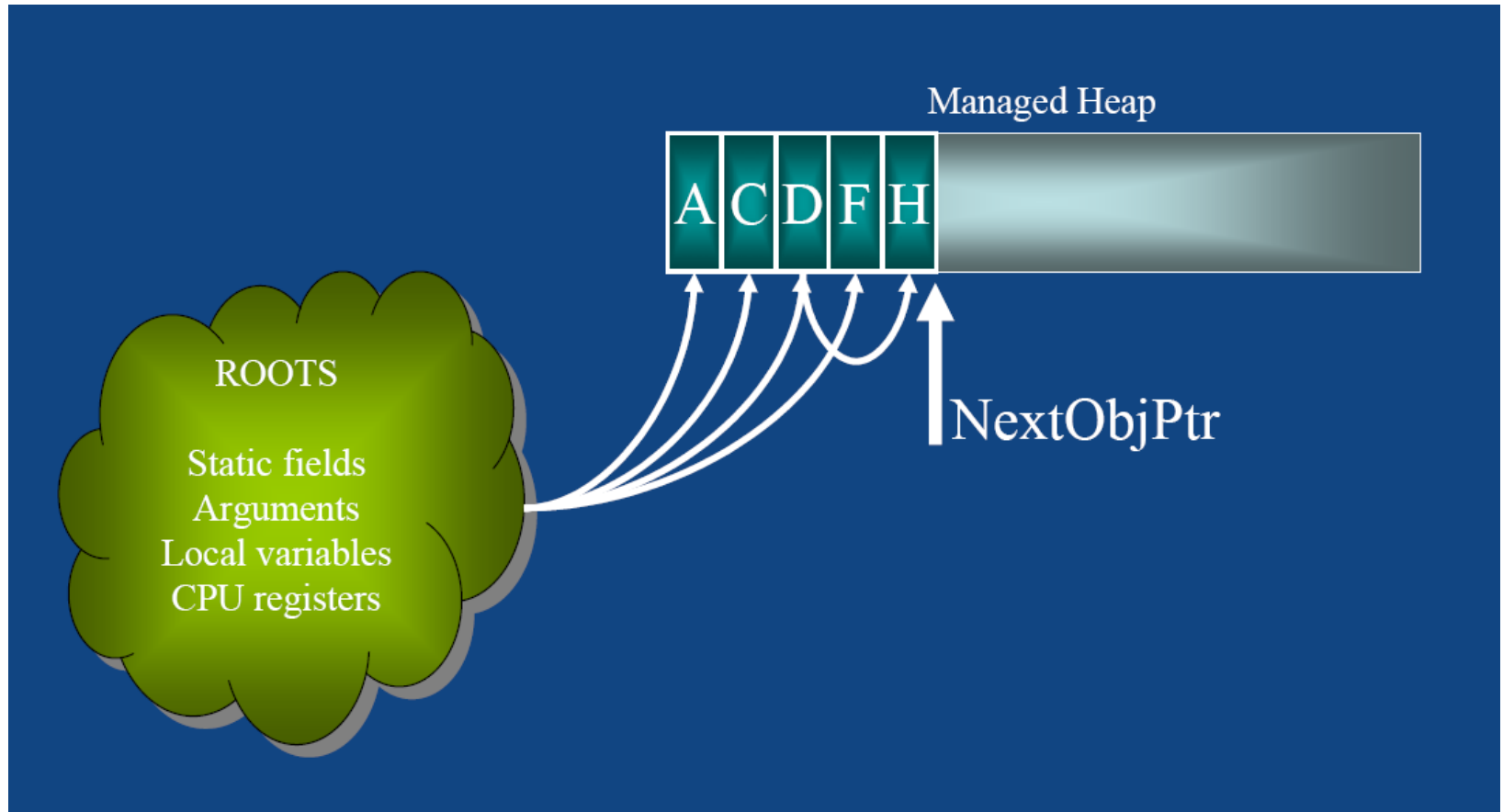
Using Roots to Mark Objects

- When a garbage collection starts all objects in heap are considered garbage
- Marking Phase:
 1. Marks objects reachable from Roots
 - A Root is a memory location that can refer to an object (or be null)
 - Static fields defined within a type
 - Arguments passed to a method
 - Local variables declared within a method
 - CPU registers (enregistered fields, arguments, or variables)
 - Roots are always reference types (never value types)
 - Each method has a root table (produced by JIT compiler)
 2. Each marked object has its fields checked; these objects are marked too (recursively)
 3. GC walks up the thread's call stack determining roots for the calling methods by accessing each method's root table
 - Already marked objects are skipped
 - Improves performance
 - Prevents infinite loops due to circular references
 - Static fields are checked; these objects are marked too

Before a Collection



After a Collection



Compacting Objects in the Heap

- **Compacting Phase:** Compacts marked objects
 - Marked objects are shifted down (simple memory copy)
- No address space fragmentation unlike the unmanaged heap!
 - Each root is updated to point to object's new memory address
- After compacting the heap...
 - The NextObjPtr pointer is positioned after last surviving object
 - The 'new' operation that caused the GC is tried again
 - Heap should have free space & new should succeed
 - If not, an OutOfMemoryException is thrown

Generations

- Makes assumptions about your code
 - The newer an object is, the shorter its lifetime will be
- A collection is likely to free a lot of memory
 - The older an object is, the longer its lifetime will be
- A collection is unlikely to free a lot of memory
 - New objects have a strong relationship are accessed together
- Allocated adjacent: locality of reference, reduced working set
- Studies show that assumptions are valid for many apps
- Generational GC improves perf by collecting new objects only
 - Old objects are not marked and walked recursively
 - Only new surviving objects are compacted
- Only new objects' roots need updating

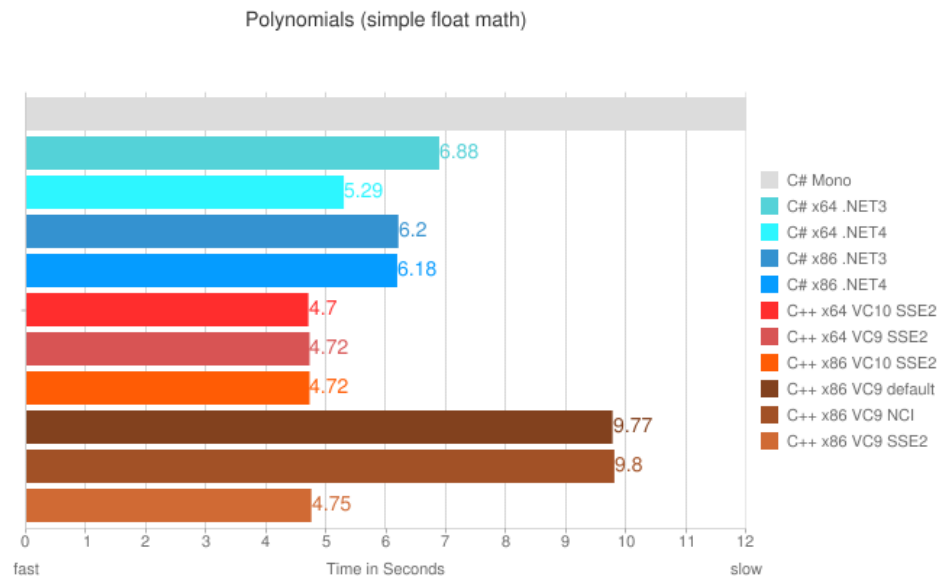
Improves Performance!

Tools for Monitoring the GC

- PerfMon
 - Graphs many .NET-related objects/counters
 - Comes with Windows
- CLR Profiler
 - Shows objects by: most-allocated, size
 - Show function call graphs, loaded types, etc.
 - Download from <http://MSDN.Microsoft.com>
- Search for:
“Writing High-Performance Managed Applications :
A Primer” by Gregor Noriskin
 - Article has a link to download the CLR Profiler
- Plus several commercial products like Red Gates Ants Profiler:
<http://www.red-gate.com/>

IS C# Slow?

- It depends heavily on the algorithm.
- In general you can expect C# programs to run approximately 10 % slower than C++ programs, due to Garbage Collections and range checks on arrays etc.
- But Head-to-head benchmark: C++ vs. .NET shows that C# sometimes perform better, and sometimes are 2 – 3 times slower than C++.



Ref.: <http://www.codeproject.com/KB/cross-platform/BenchmarkCppVsDotNet.aspx>

References

- Garbage Collection
<http://msdn.microsoft.com/en-us/magazine/bb985010.aspx>
- Common Type System
<http://msdn.microsoft.com/en-us/library/zcx1eb1e.aspx>
- Automatic Memory Management
<http://msdn.microsoft.com/en-us/library/f144e03t.aspx>

Resources

- Microsoft Developer Network: <http://msdn.microsoft.com/>
- Microsoft's sourceforge-like project site: <http://www.codeplex.com/>
- Good sites for .NET code, discussions, etc:
 - <http://stackoverflow.com/>
 - <http://wpfdisciples.wordpress.com/>
 - <http://www.codeproject.com>
 - <http://www.asp.net/>