The .NET platform

Managed Execution

- Execution of programs takes place under the control of a runtime system. The runtime provides services to the running program.
- Typically, the level of abstraction of the program encoding is higher than machine code.
- Services of the runtime include “garbage collection” (memory management), verification of type safety, enforcement of memory safety and member accessibility.
- Typically, a security model is part of the specification, allowing for “sandboxed” execution.

Managed Execution – pro and con

- Some of the largest attack surfaces for malware, e.g. buffer overflow and indexing out of bounds exploits, simply do not exist for managed execution.
- Garbage Collection eliminates almost all memory leak issues, which are difficult to avoid and costly to find in conventional systems.
- Managed execution adds overheads in memory use, and usually also in runtime performance and start-up delay.
- Managed execution is unsuitable for systems with hard real-time response constraints.

The Microsoft .NET platform

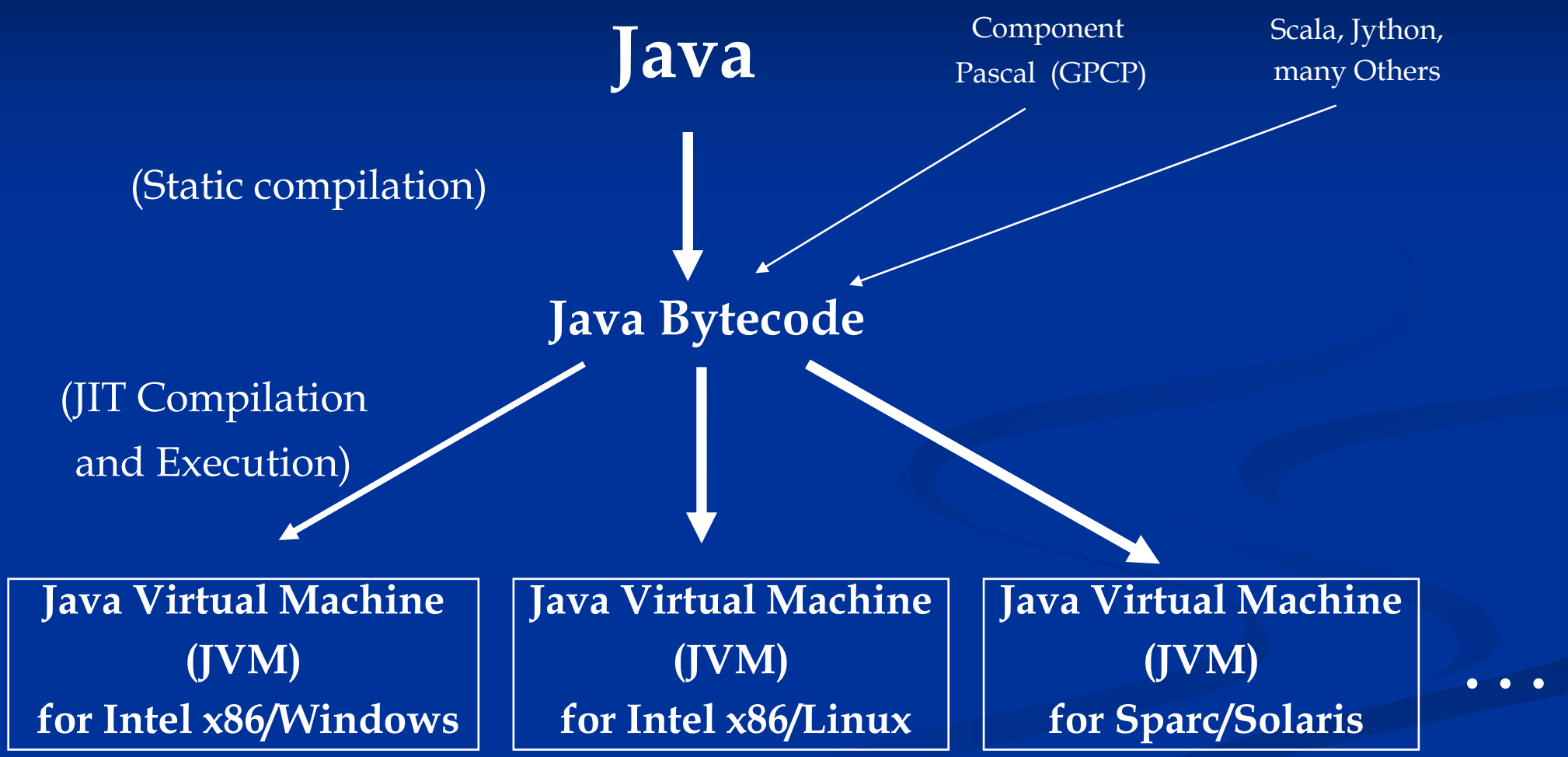
- Moving away from directly executing native code.
  - not enough control
- *Managed code* runs within a new runtime environment
- Code from any source language can be compiled to a common intermediate language (*CIL*)
  - similar to Java Byte Code.
- CIL is Just-In-Time (JIT) compiled into native code
  - similar to how Java Virtual Machines work.
- Primarily used on the Microsoft Windows operating system.
- Announced 13th July 2000, at the PDC in Orlando, Florida.

PDC attendees scramble for .NET preview CD.

Delegate types

- Delegate types may be thought of as type-safe function pointers
- Delegates have a constructor and an *Invoke* method. The signature of the invoke determines the compatible method values
- The constructor may encapsulate a “this” pointer to use in the invocation

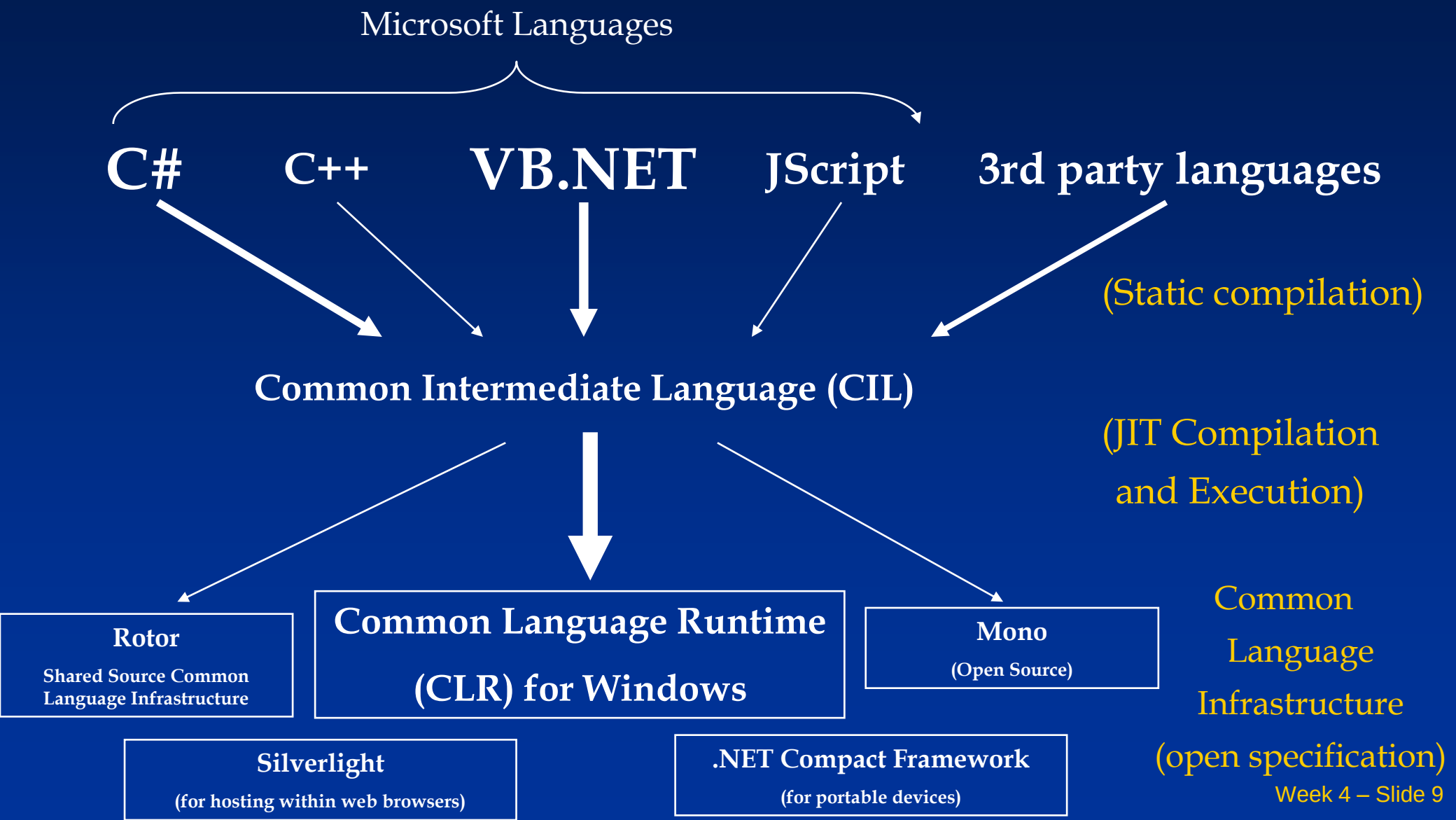
The Java Platform



Java – Historical Notes

- Java had its origins in a small footprint embedded system intended for TV set-top boxes, “Oak”. This was the early 1990s.
- It was retargeted for the then new concept of the Web Browser. The first public release of Java was version 1.02a on 23rd May 1995.
- The current version of Java SE at March 2013 is Version 7 update 17 (1.7.0_17). Version 8 will be out April 2014.

The .NET Platform



CIL Example

```
.class public MyClass extends [mscorlib]System.Object
{
  .method public specialname rtspecialname instance void  .ctor() cil managed
  {
    .maxstack  1
    ldarg.0
    call       instance void [mscorlib]System.Object::.ctor()
    ret
  }

  .method private static void  Main(string[] args) cil managed
  {
    .entrypoint
    .maxstack  1
    ldstr      "Hello World!"
    call       void [mscorlib]System.Console::WriteLine(string)
    ret
  }
}
```

Why .NET ?

- Better support for components (compared to COM).
  - new component "*assemblies*"
  - extensible meta information format
  - improved versioning
  - simpler model for component interaction
  - easier to deploy components (no registry)
  - light weight installation
- Provides better control over security.
  - fine grain access control (*role based* or *code access* security)
- Allows language interworking (not just interoperability)
  - supports inter-language inheritance, exception handling, garbage collection, debugging and profiling.
  - common class libraries (including *WinForms* and ASP.NET).

Assemblies

- An assembly is a collection of one or more "binary" files that together constitute a single software component.
  - contains exactly one main function (DllMain, WinMain or Main).
  - commonly just one (dll) file.
- One of the files contains a manifest that lists the other files in the assembly
  - hash values ensure none of the files can be changed after the assembly is created.
- Deployment
  - all of the files in an assembly must be deployed together.
- Versioning
  - assemblies are the smallest versionable unit
  - also specifies which version of other assemblies it depends on.
- Loading and Security
  - the loader loads and unloads entire assemblies on demand.
  - the assembly requests security permissions which may be granted by the loader.
- Unique type names.
  - every type's identity includes the name of the assembly in which it resides.

Versioning

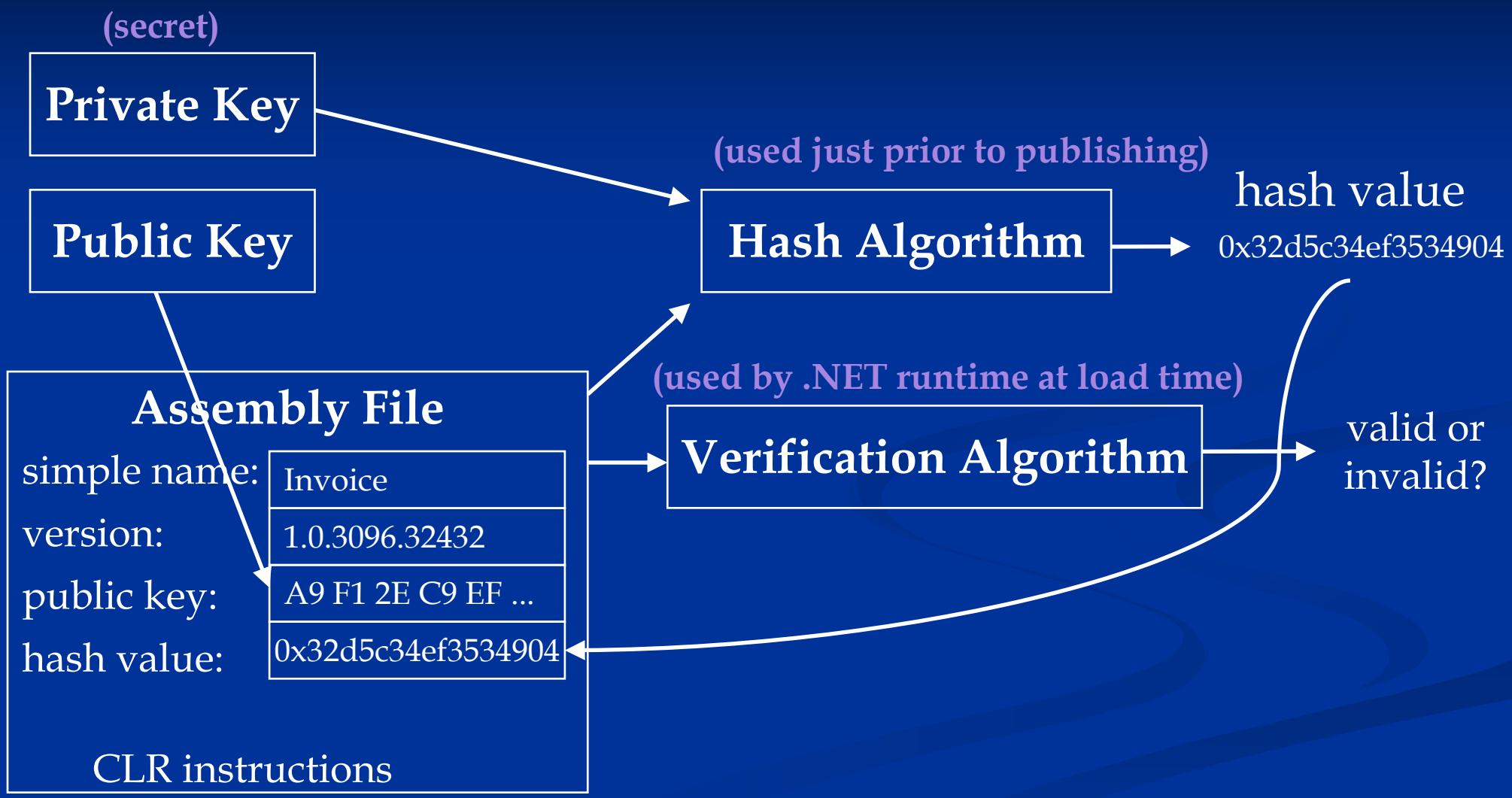
- With the COM component model, whenever a component needed to be updated, an entirely new GUID had to be assigned to it.
  - the new component name bore no relation to the old even though it was still logically the same component.

- In the .NET component model, assemblies are given a simple name that remains constant, plus a version number that changes with each update, eg:
  - `System.Web.Services 1.0.3300.0`
  - This simple name and version is not guaranteed to be globally unique.

Strong Names

- Assemblies can be given a globally unique "*strong*" name by adding a *public key* to the simple name and version:
  - A secret *private key* is used together with the public key to perform a hash function on the contents of the assembly.
  - This hash value is stored in the assembly.
  - The public key is sufficient to check whether the stored hash value corresponds to the contents of the assembly.
- An assembly can specify that it depends on a particular version of some other assembly by specifying its strong name.
  - The calling assembly can be certain that at runtime it will only accept the services of the exact same dependent assembly that was specified when it was compiled.
  - Without the secret private key, no one can create or modify an assembly that spoofs the identity original.
  - Similarly, without the private key, no one can create a subsequent version of such an assembly.

Strong Name Hashing



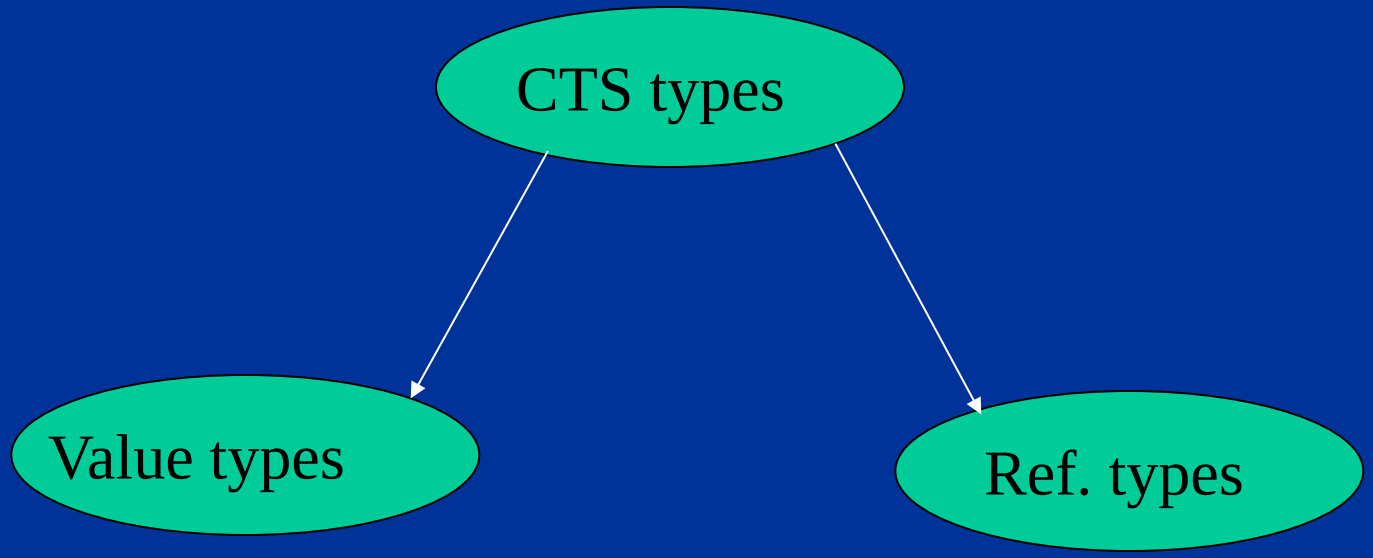
Common Language Runtime (CLR)

- Common Type System
  - a rich type system intended to support the complete implementation of a wide range of programming languages.
- Meta Data
  - a common interchange mechanism for use between tools that manipulate programs (compilers, debuggers, etc.) and the Virtual Execution System.
- Common Language Specification
  - a subset of the Common Type System and a set of usage conventions
- Virtual Execution System
  - responsible for loading and running CIL programs

Common Type System (CTS)

- Types are classified as either:
  - Value types
  - Reference types
  - Can convert between value and reference type by boxing and unboxing.
- CTS Types:
  - Classes (with protection mechanisms)
  - Interfaces
  - Pointers (machine addresses)
  - Enumerations
  - Delegates (function pointers for methods)
  - Built-in types: Object, String, bool, char, float32, float64, int16, int32, ...
- Supports:
  - multiple interface inheritance
  - but only single implementation inheritance

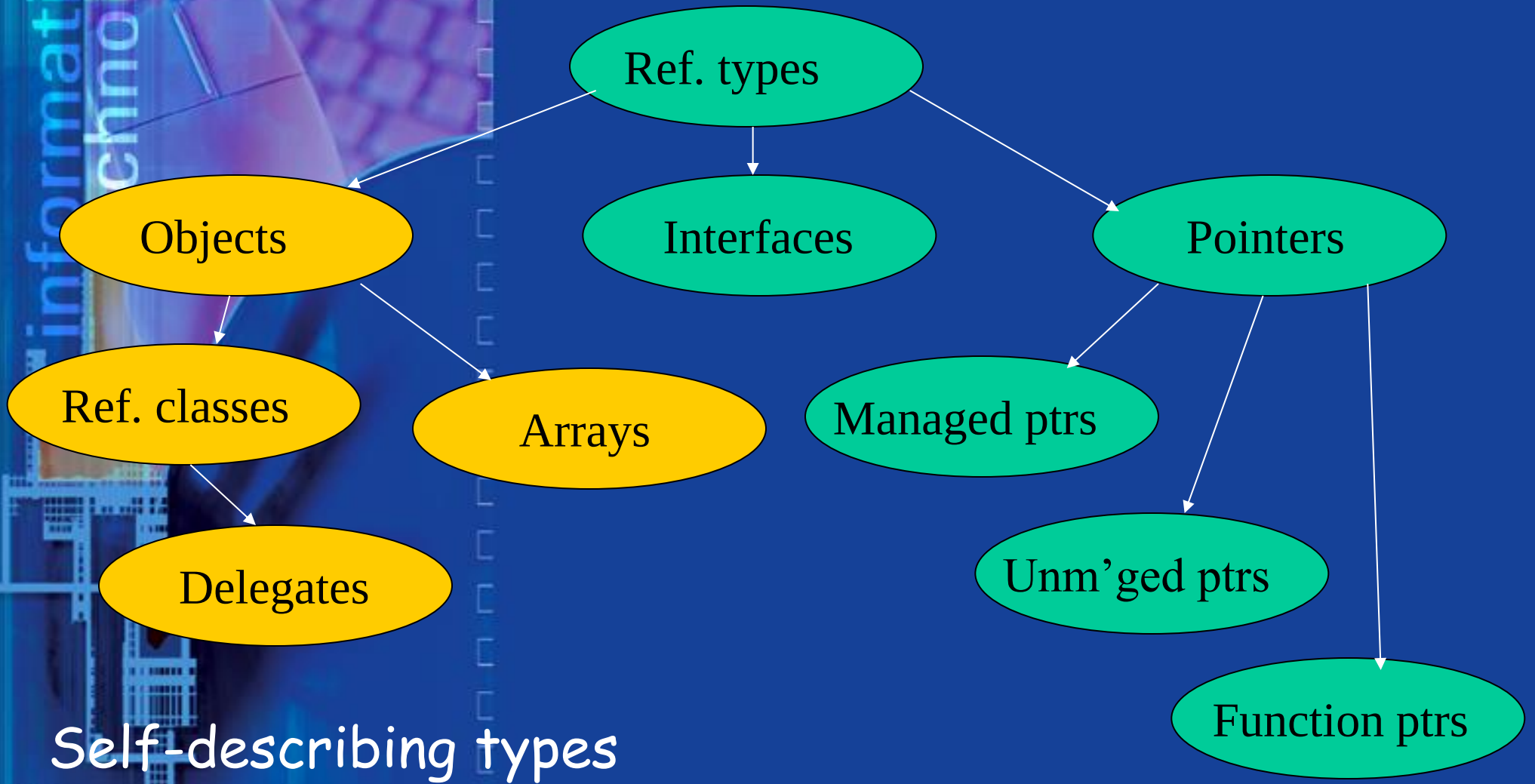
CTS Types



Includes base types such as **int32**, enum types and value classes.

Includes pointers, ref. classes, delegates, arrays

Reference types



Self-describing types

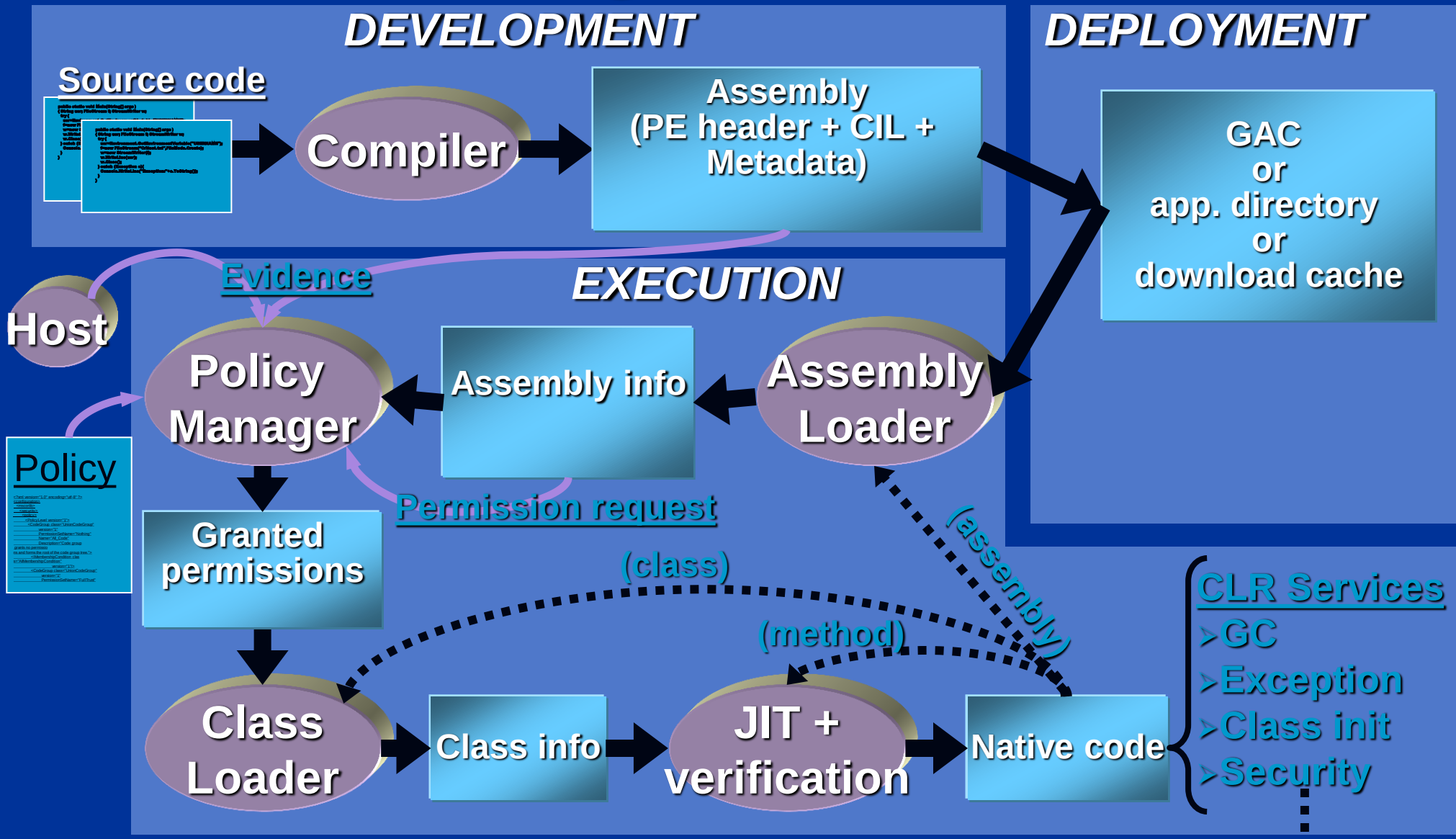
MetaData

- When a compiler produces CIL, it also produces metadata that describes the types the code defines, the assemblies the code references, and other data used by the runtime environment.
- The CIL and metadata are contained in a portable executable (PE) file that is based on and extends the published Microsoft Portable Executable (PE) and Common Object File Format (COFF) used historically for executable content.
- So CIL instructions and metadata are stored in specially formatted .exe or .dll files that can be directly “executed”.
- The presence of metadata in the file along with the CIL enables the code to describe itself, which means that there is no need for type libraries or IDLs.

Virtual Execution System (VES)

- Executes Common Intermediate Language (CIL) programs.
  - Resolves assembly names when loading managed code
  - JIT compiles CIL into native code (when methods are first invoked).
  - Verifies the type safety of methods written in CIL.
  - Garbage collection of memory containing managed data
  - Initiation, propagation, and interception of exceptions.
  - Insertion and management of security tests.
  - Profiling and debugging services.
  - Management of threads, contexts, and remoting.

Managed Code Execution



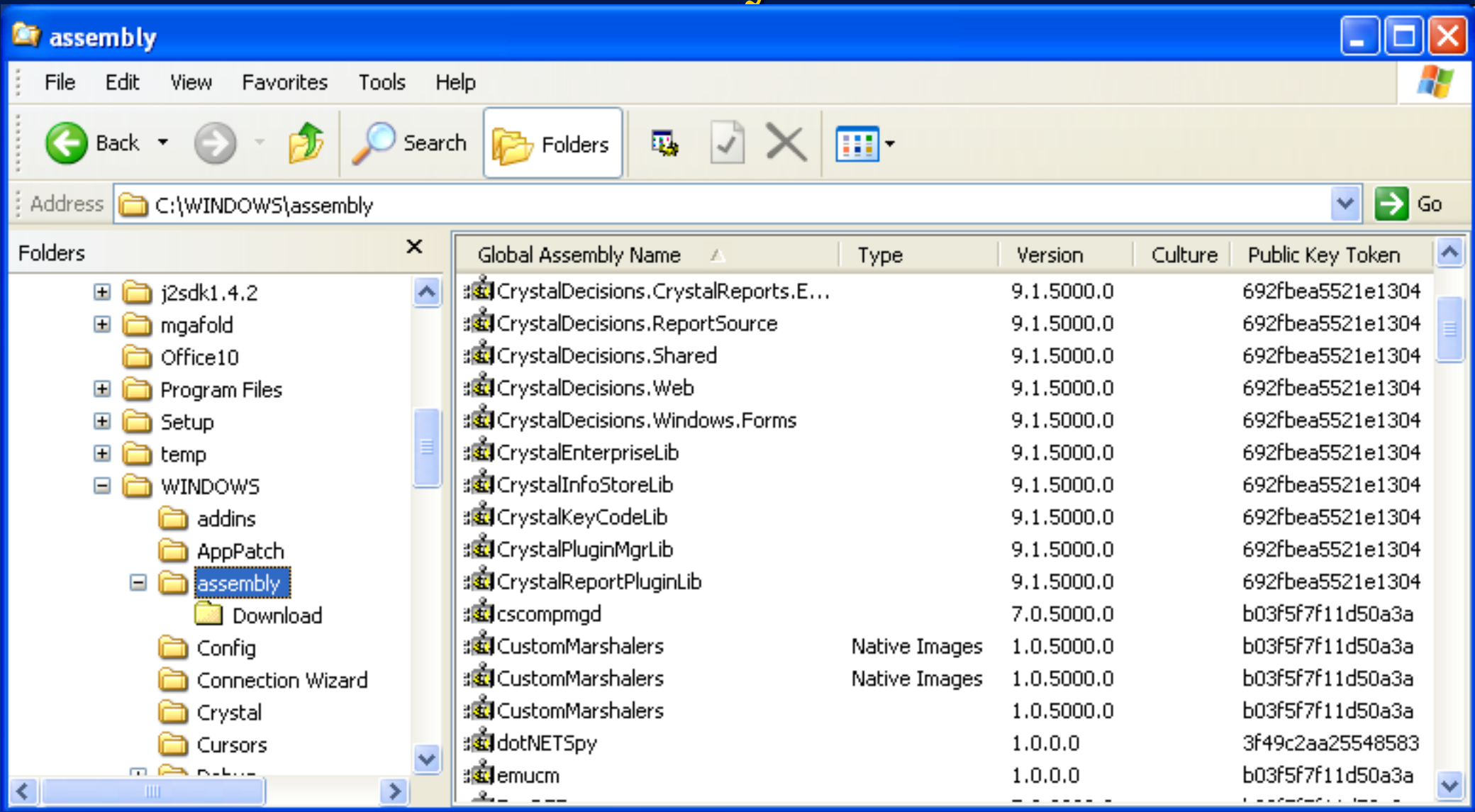
Unmanaged, Managed and Verifiable

- *Managed data*:
  - has a CTS type
  - is laid out in memory in a standard way
  - is garbage collected by the CLR
- *Managed code*:
  - expressed using the CIL,
  - JIT compiled by the CLR
- *Unmanaged code*:
  - native code
  - bridges exist to allow managed and unmanaged code to interact in limited ways
- *Verifiable code*:
  - managed code that the CLR can prove is type safe.
  - it's possible for code to be managed but not verifiable.

Deployment

- .NET components don't need to be registered in the registry.
- Non-shared components can simply be placed in the application's "Program Files" directory
  - XCOPY deployment
  - No-impact installation (controlled code sharing)
- Shared components are placed in the Global Assembly Cache.
  - a special directory managed using .NET tools (Shfusion.dll).
  - supports side by side execution
- Download Assembly Cache
  - for web based applications.
- Uses XML Configuration files

Global Assembly Cache Viewer



Configuration Files

- XML configuration files control:
  - dependent assembly binding (version redirect)
  - location of dependent code (codebase)
  - security policies
- Application configuration files
  - located in the same directory as the application (`myapp.exe.config`)
- Machine configuration file
  - `WinNT\Microsoft.NTE\Framework\v1.0.3705\Config\Machine.config`
- ASP.NET configuration files (`Web.config`).

.NET Languages

- Microsoft languages
  - C#, Visual Basic.NET, C++ (managed extension), JScript.NET

- Third party languages ("Project 7")
  - Component Pascal (QUT)
  - Scheme (Northwestern University, USA)
  - Oberon (ETH Zurich)
  - Mercury (Univ Melbourne)
  - Mondrian (Univ Massey, NZ)
  - Standard ML (Microsoft Research, Cambridge) ... became F#
  - COBOL and Fortran (Fujitsu)
  - Eiffel (ISE)
  - Perl and Python (ActiveState)
  - Smalltalk (SmallScript inc.)

C# Example

```
using System;

class Hello
{
    public static void Main(string[] args)
    {
        Console.WriteLine("Hello World!");
    }
}
```

Visual Basic Example

```
Imports System

Module Hello

Sub Main()
Console.WriteLine("Hello World!")
End Sub

End Module
```

C++ (Managed extensions) Example

```
#using <mscorlib.dll>

using namespace System;

void main()
{
    Console::WriteLine("Hello World!");
}
```

What's new in Visual Basic 7

- The language:
  - Implementation inheritance
  - Interfaces
  - Structured exception handling
  - Overloading and overriding
  - Constructors and Destructors
  - Delegates
  - Import statements
- Creating GUIs:
  - no more .frm files; user interface created programmatically
  - Programmer creates a form class derived from `System.Windows.Forms.Form`

C++ Extensions for Managed Execution

- Need to use `/CLR` compiler flag
- Revised syntax allows Classes can be marked as:

`abstract`	`__abstract`	(can't be instantiated)
`ref`	`__gc`	(garbage collected)
`interface`	`__interface`	(interface)
`sealed`	`__sealed`	(can't be inherited from)
`value`	`__value`	(a value type)

- Native and managed heaps:

```
Class1^ obj1 = gcnew Class1;
Class2* obj2 = new Class2;
```

- Properties can be defined:

```
property int Size { int get() { return m_size; }}
```

- …
- Note: this is not ANSI standard C++

Common Class Library

`System.Collections (and System.Collections.Generic)`
arrays, hashtables, lists, stored lists, queues, stacks, dictionaries, etc

`System.Data`
database APIs including ADO.Net

`System.Drawing`
GDI graphics functionality

`System.IO`
input and output

`System.Net`
internet protocol APIs including TCP/IP and HTTP

`System.Reflection`
provides view of loaded types, methods, and fields

`System.Security`
underlying structure of .NET security framework

`System.Threading`
enables multi-threaded programming

`System.Web`
for creating web-based applications

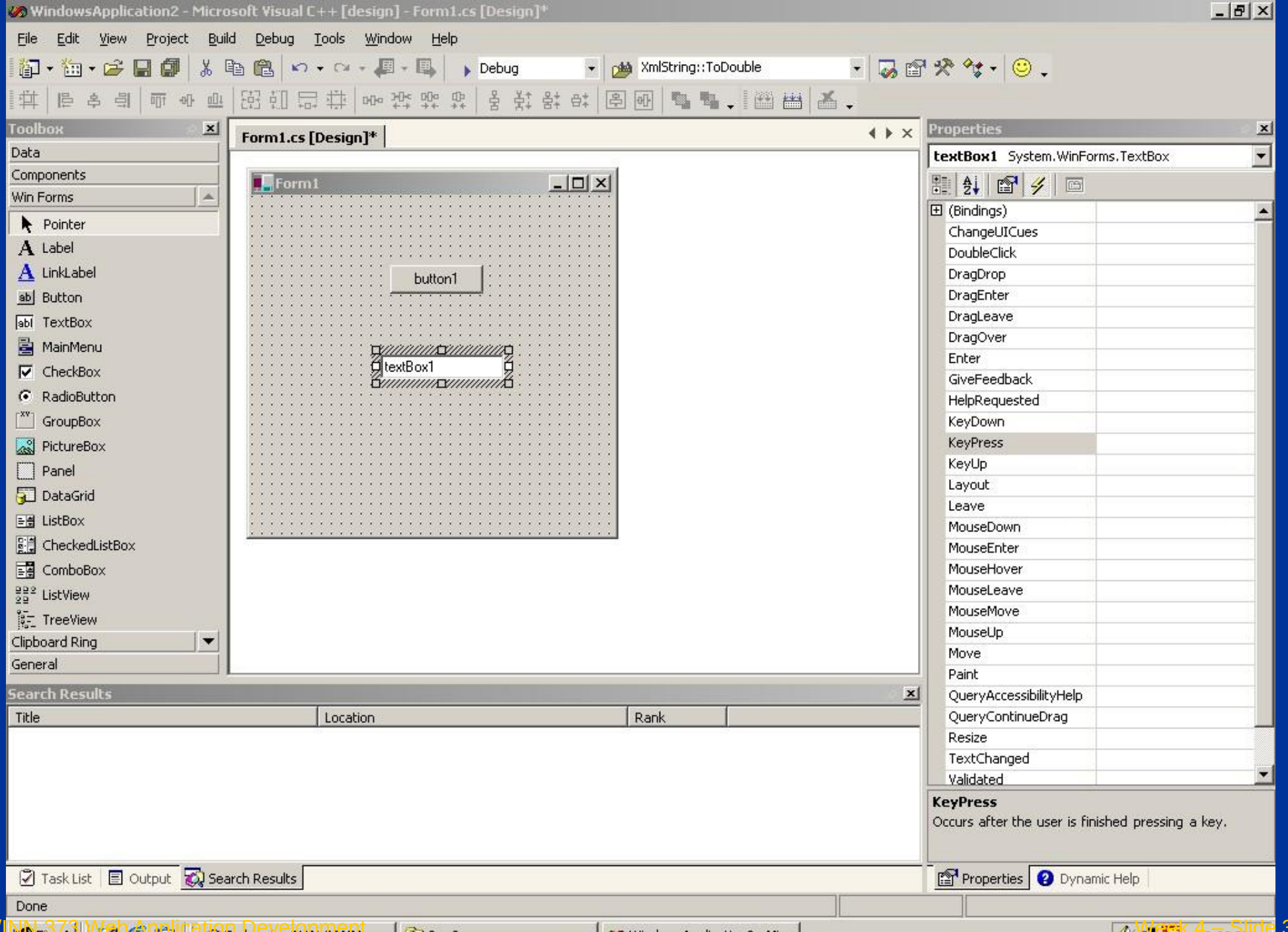
`System.Windows.Forms`
for creating GUI applications

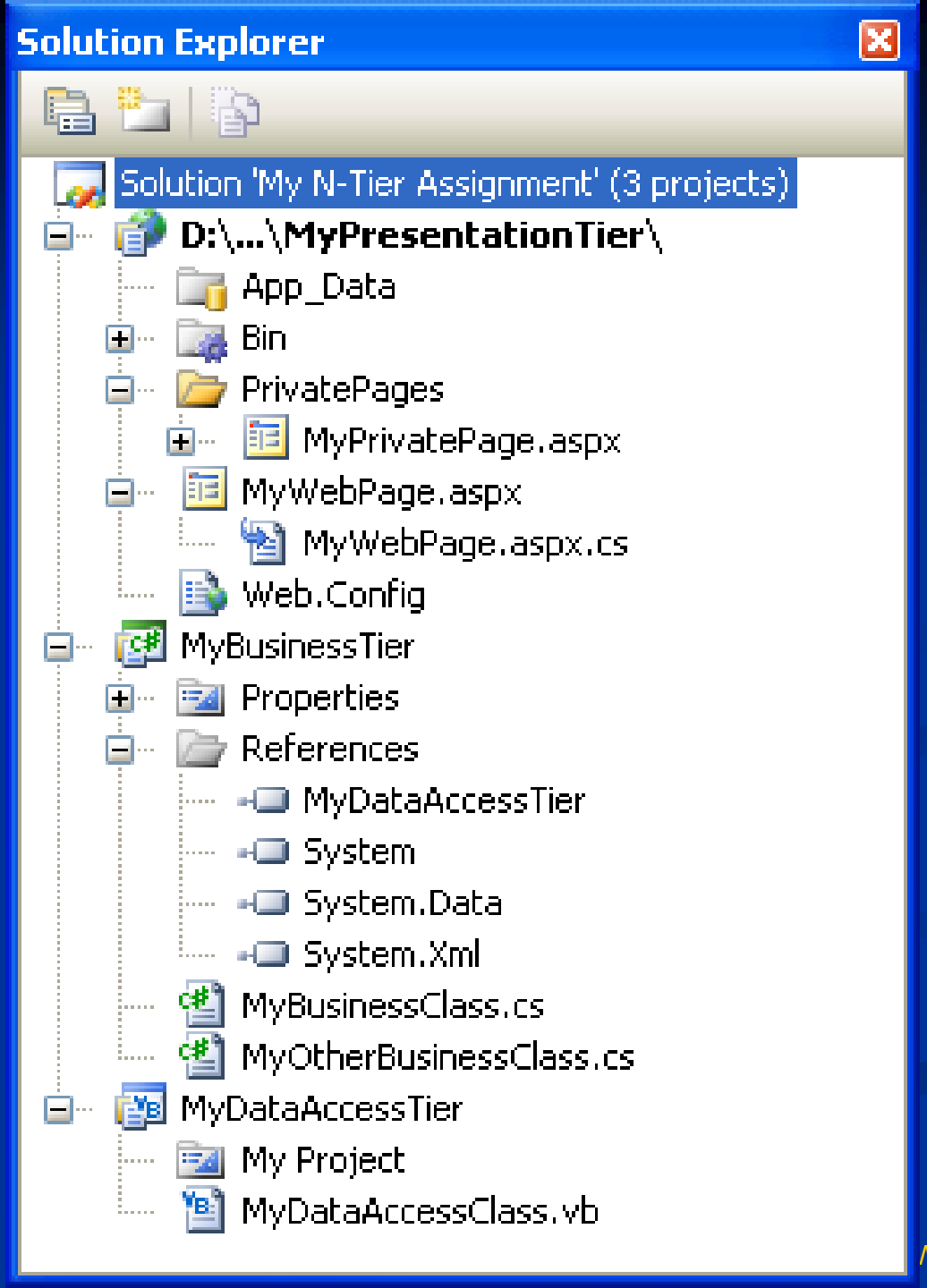
`System.Xml`
standards-based support for XML processing

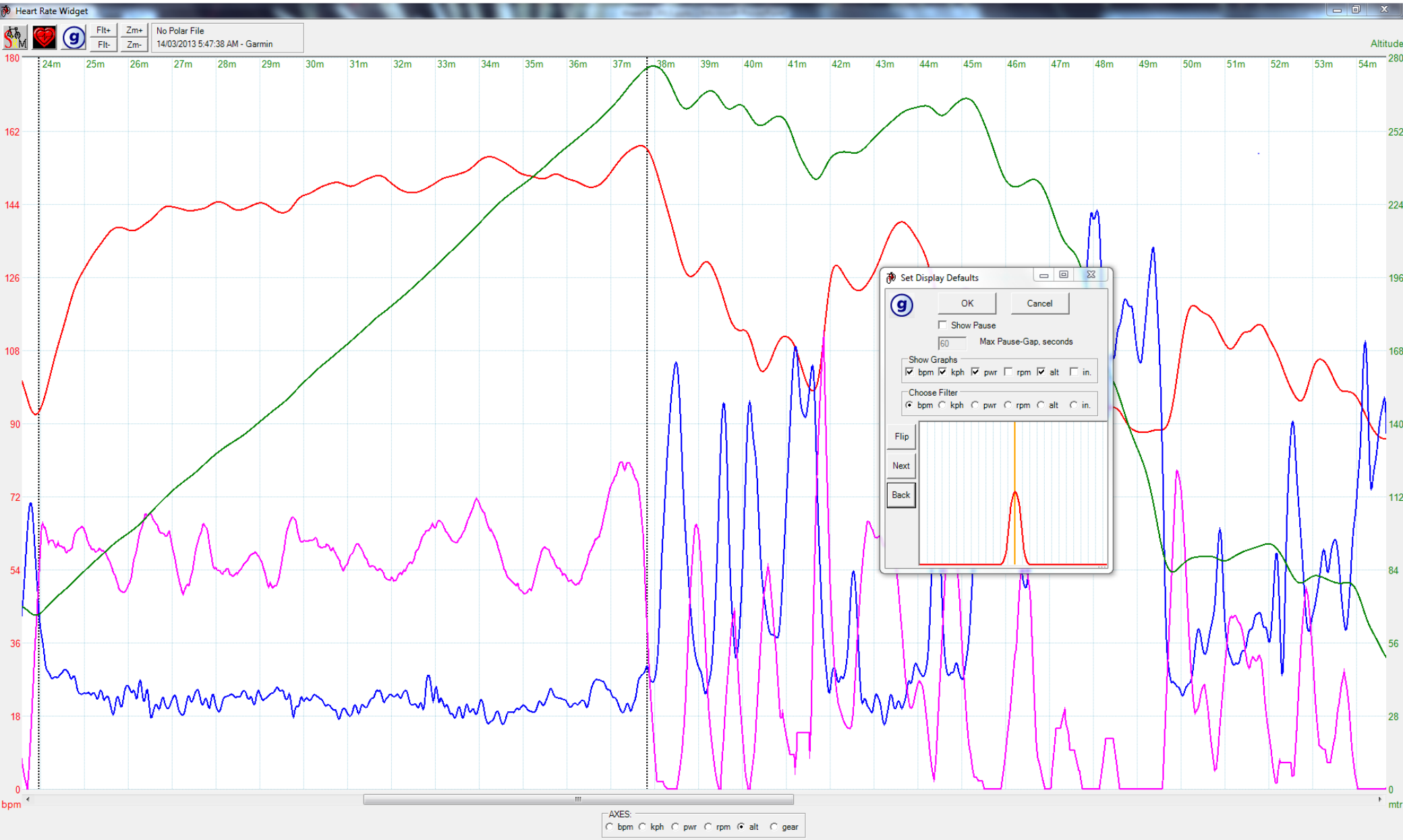
Visual Design in Visual Studio.NET

- Visual Studio.NET supports a visual interface for:
  - positioning and sizing controls
  - setting properties
  - creating menus
  - and specifying event handlers

  at design time, by automatically inserting initialization code that performs those activities.


WindowsApplication2 - Microsoft Visual C++ [design] - Form1.cs [Design]*
File Edit View Project Build Debug Tools Window Help
Debug
XmlString::ToDouble
Toolbox
Data
Components
Win Forms
Pointer
Label
LinkLabel
Button
TextBox
MainMenu
CheckBox
RadioButton
GroupBox
PictureBox
Panel
DataGrid
ListBox
CheckedListBox
ComboBox
ListView
TreeView
Clipboard Ring
General
Form1.cs [Design]*
Form1
button1
textBox1
Properties
textBox1 System.WinForms.TextBox
(Bindings)
ChangeUICues
DoubleClick
DragDrop
DragEnter
DragLeave
DragOver
Enter
GiveFeedback
HelpRequested
KeyDown
KeyPress
KeyUp
Layout
Leave
MouseDown
MouseEnter
MouseHover
MouseLeave
MouseMove
MouseUp
Move
Paint
QueryAccessibilityHelp
QueryContinueDrag
Resize
TextChanged
Validated
KeyPress
Occurs after the user is finished pressing a key.
Search Results
Title
Location
Rank
Task List
Output
Search Results
Properties
Dynamic Help
Done
Start
Eudora by QUALCOMM
One2
WindowsApplication2 - Mi...
2:20 PM


Solution Explorer
Solution 'My N-Tier Assignment' (3 projects)
D:\...\MyPresentationTier\
App_Data
Bin
PrivatePages
MyPrivatePage.aspx
MyWebPage.aspx
MyWebPage.aspx.cs
Web.Config
MyBusinessTier
Properties
References
MyDataAccessTier
System
System.Data
System.Xml
MyBusinessClass.cs
MyOtherBusinessClass.cs
MyDataAccessTier
My Project
MyDataAccessClass.vb


Heart Rate Widget
Flt+
Flt-
Zm+
Zm-
No Polar File
14/03/2013 5:47:38 AM - Garmin
Altitude
24m 25m 26m 27m 28m 29m 30m 31m 32m 33m 34m 35m 36m 37m 38m 39m 40m 41m 42m 43m 44m 45m 46m 47m 48m 49m 50m 51m 52m 53m 54m
180 162 144 126 108 90 72 54 36 18 0
bpm
280 252 224 196 168 140 112 84 56 28 0
mtr
Set Display Defaults
OK
Cancel
Show Pause
60
Max Pause-Gap, seconds
Show Graphs
bpm kph pwr rpm alt in.
Choose Filter
bpm kph pwr rpm alt in.
Flip
Next
Back
AXES:
bpm kph pwr rpm alt gear

Evolution of Type Systems in Managed Execution Systems

- Introduction of Parametric Polymorphism (Generics) Java and .NET have equivalent expressive power.
- Anonymous methods and/or classes.
- Type inference.
- Lambda abstraction (already in .NET, coming in Java Version 8).
- Support for dynamically typed languages "invokedynamic" instruction.