The Common Language Runtime (CLR)

Based on

Mark Sapossnek

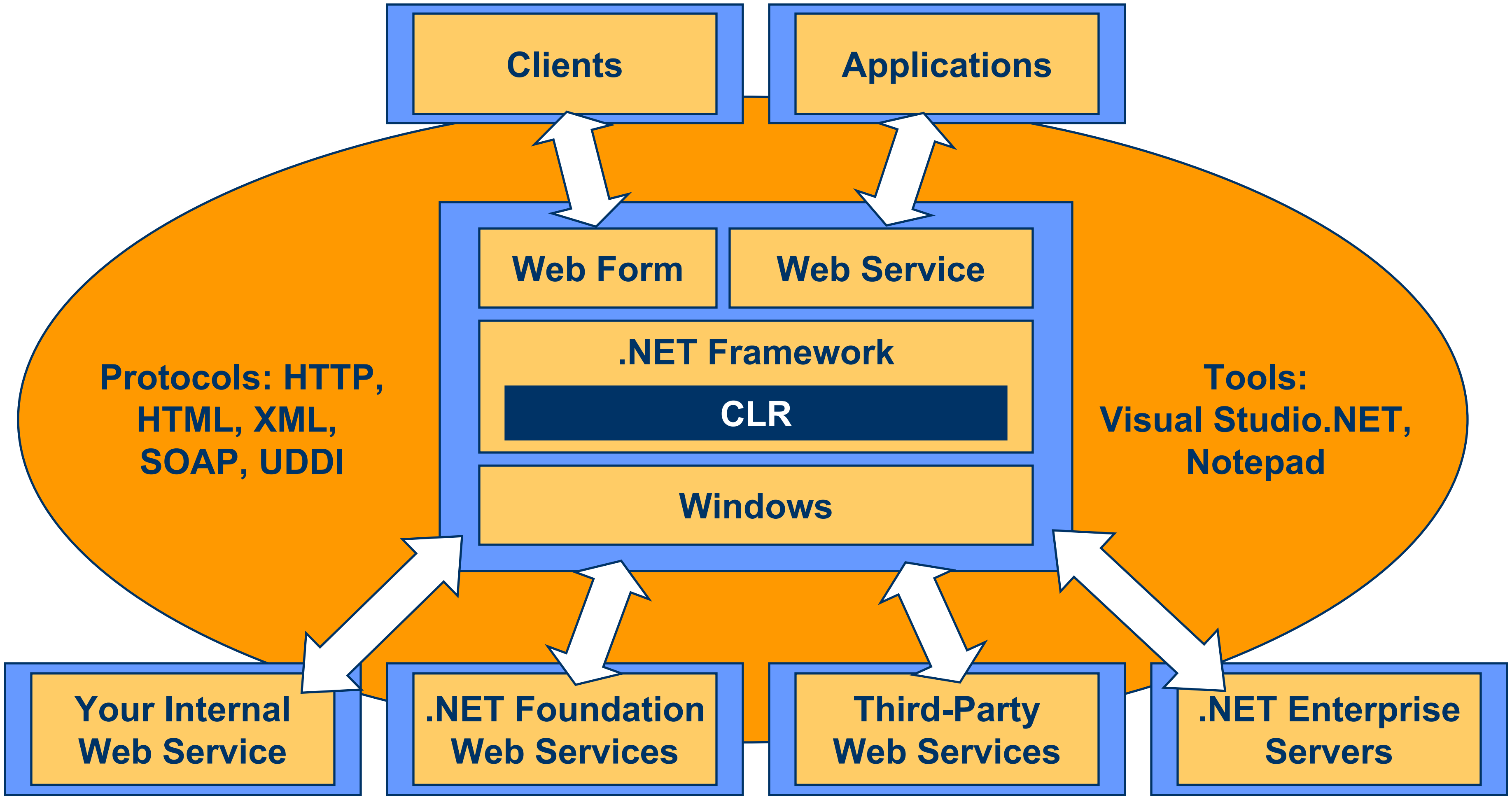
Computer Science Department
Metropolitan College
Boston University

Agenda

- What Is the CLR?
- Assemblies
- Execution Model

What is the CLR?

The .NET Platform

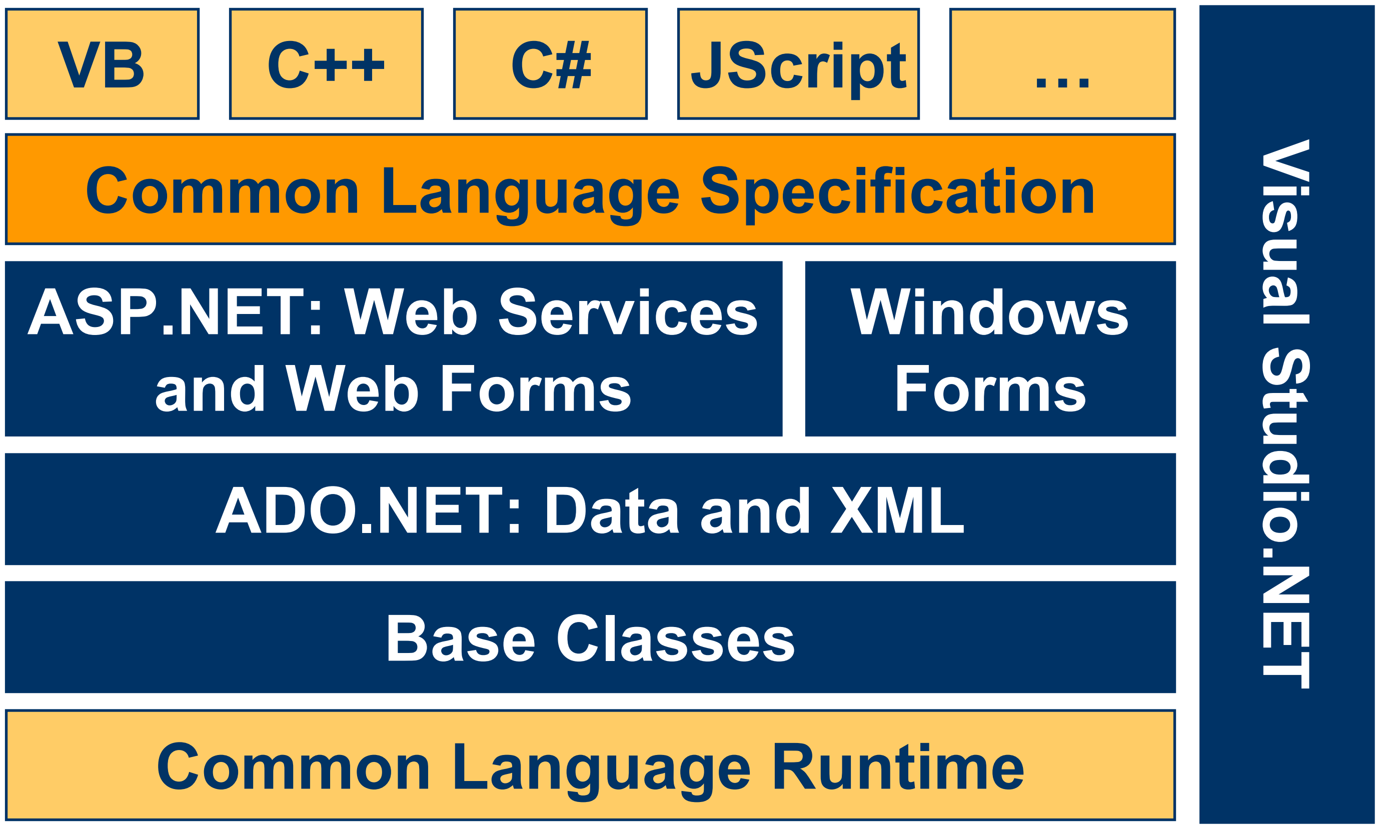

Clients
Applications
Web Form
Web Service
.NET Framework
CLR
Windows
Protocols: HTTP, HTML, XML, SOAP, UDDI
Tools: Visual Studio.NET, Notepad
Your Internal Web Service
.NET Foundation Web Services
Third-Party Web Services
.NET Enterprise Servers

What Is the CLR?
The .NET Framework

- A set of technologies for developing and using components to create:
  - Web Forms
  - Web Services
  - Windows applications
- Supports the software lifecycle
  - Development
  - Debugging
  - Deployment
  - Maintenance

What Is the CLR?

The .NET Framework



What Is the CLR?
Overview

- The CLR provides a run-time environment that manages the execution of code and provides services that improves development, deploy-ment, and run time.
- Code that targets the CLR is called managed code.

What Is the CLR?
Goals

- Development services
  - Deep cross-language interoperability
  - Increased productivity
- Deployment services
  - Simple, reliable deployment
  - Fewer versioning problems – **NO MORE ‘DLL HELL’**
- Run-time services
  - Performance
  - Scalability
  - Availability

What Is the CLR?
Goal: Simpler Development

- Plumbing disappears
  - Metadata
  - Transparent proxies
  - Memory management
  - Consistent exception handling
- Great WYSIWYG tool support
  - Designers and wizards
  - Debuggers
  - Profilers
- Increased productivity

What Is the CLR?
Goal: Simpler, Safer Deployment

- No registration, zero-impact install
  - XCOPY deployment, incremental download
- Side-by-side versions of shared components
  - Capture version at compile time
  - Administrative policy at run time
- Evidence-based security policy
  - Based on code as well as user
  - Code origin (location)
  - Publisher (public key)

DLL Hell

What Is the CLR?
Goal: Scalability

- Smart device to Web Farm
- Automatic memory management
  - Self-configuring
  - Dynamically tuning
- Thread pool
- Asynchronous messaging
  - Object remoting
  - Events
- Smart device version
  - Multiple RTOSes
  - Same tools used for desktop

What Is the CLR?
Goal: Rich Web Clients, Safe Hosting

- WinForms on the client
- ASP.NET Web Forms on the server
- Code is granted permissions
  - Evidence is used by policy to grant permissions
- Application that starts runtime
    - Like Internet Explorer, IIS, SQL Server™, Shell
  - Provides some evidence
  - Controls code loading
  - Maps applications to processes

What Is the CLR?
Goal: Converge Programming Models

- COM, ASP, VB, C++
  - All services available
  - Many services redesigned
    - Ease of use
    - Scalability
    - Consistent API
- Consistent framework raises the abstraction layer
- Gradual transition from simplicity to full power
- Less training, greater productivity

What Is the CLR?
Goal: Multiple Languages

- Common Type System
  - Object-oriented in flavor
  - Procedural languages well supported
  - Functional languages possible
- CLS guides frameworks design
  - Rules for wide reach
  - All .NET Framework functionality available
- Over 15 languages investigated
  - Most are CLS consumers
  - Many are CLS extenders
- Choose the right language for a particular job

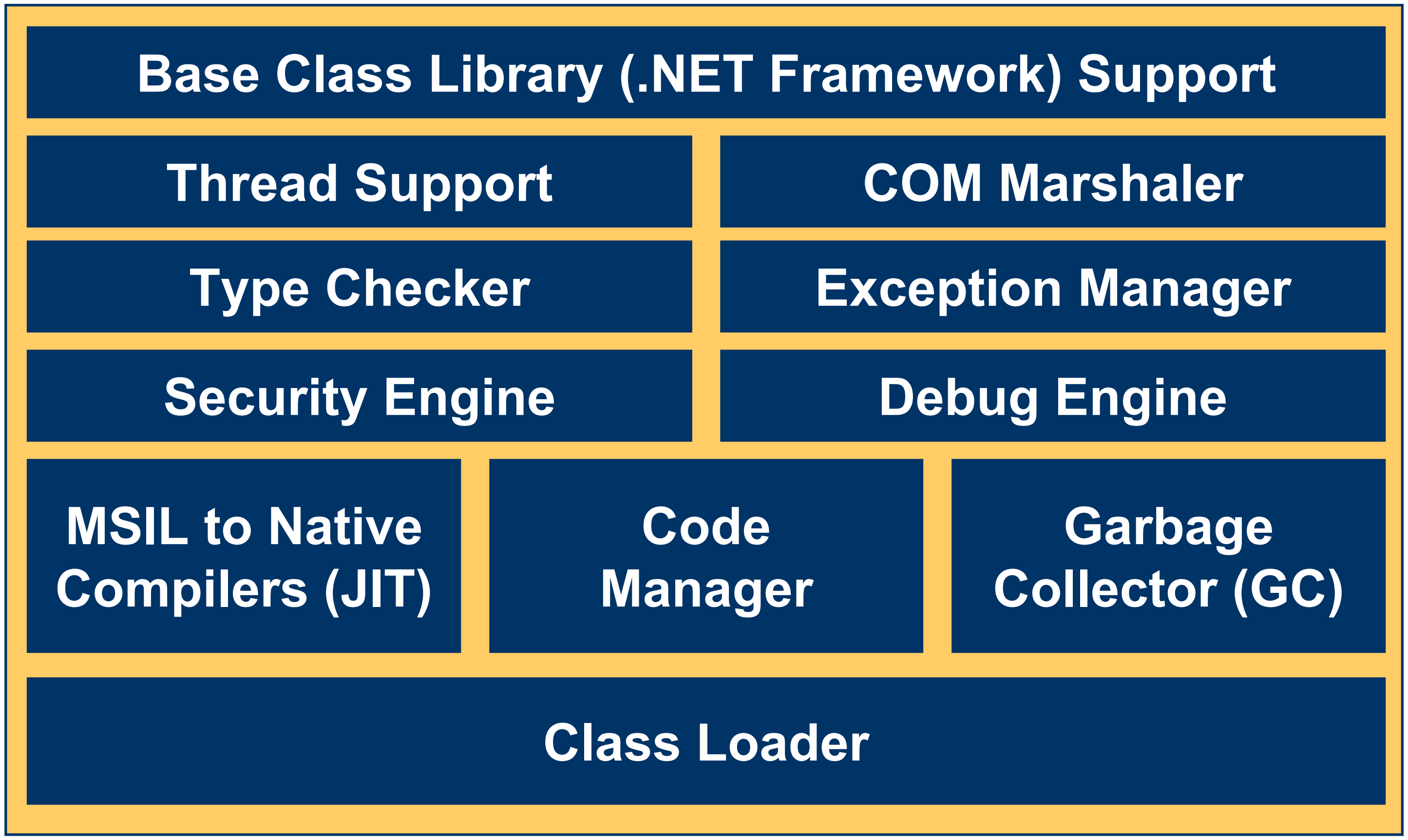
What Is the CLR?
Highlights

- Common Type System
  - Mapping of data types: Programming language ⇨ Framework
- Just-in-time (JIT) compilers
  - JIT compiles intermediate language (MSIL) into native code
  - Highly optimized for platform or device
- Garbage collector
- Permission and policy-based security
- Exceptions
- Threading
- Reflection
- Diagnostics and profiling

What Is the CLR?
Services

- Code management
- Memory management and isolation
- Verification of type safety
- Conversion of MSIL to native code
- Loading and execution of managed code
- Creation and management of metadata
- Insertion and execution of security checks
- Handling cross-language exceptions
- Interoperation between .NET Framework objects and COM objects and Win32 DLLs
- Automation of object layout for late binding
- Developer services (profiling, debugging, etc.)

What Is the CLR?
Architecture



What Is the CLR?
Soon To Be a Standard

- Microsoft, with HP and Intel, submitted proposal to ECMA to standardize:
  - C#
  - Common Language Infrastructure
    - Includes the Common Language Runtime and a subset of the .NET Framework classes
- http://msdn.microsoft.com/net/ecma/
- http://www.ecma.ch

Agenda

- What Is the CLR?
- **Assemblies**
- Execution Model
- Interoperability
- Security

Assemblies
Overview

- Contains code and metadata
- Assemblies function as:
  - Unit of deployment
  - Type boundary
  - Security boundary
  - Reference scope boundary
  - Version boundary
  - Unit of side-by-side execution

Assemblies
Overview

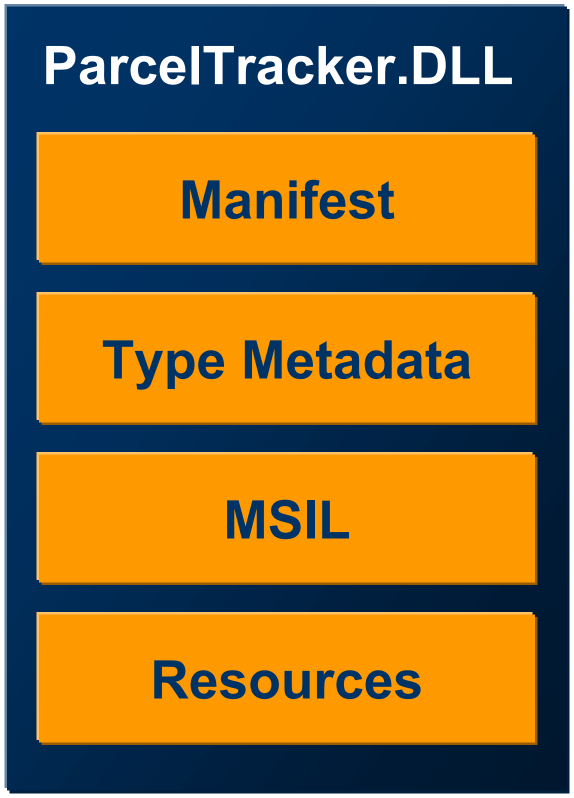
- Assemblies can be:
  - Static: DLL, EXE
    - Uses existing COFF binary format
      - Via existing extension mechanism
  - Dynamic
- Create assemblies with
  - .NET Framework SDK
  - Visual Studio.NET
  - Your own code
    - Dynamic assemblies

Assemblies
Components of an Assembly

- Manifest
  - Metadata about the assembly itself
- Type metadata
  - Completely describes all types defined in an assembly
- Managed code
  - Microsoft Intermediate Language (MSIL)
- Resources
  - For example, .bmp, .jpg

Assemblies

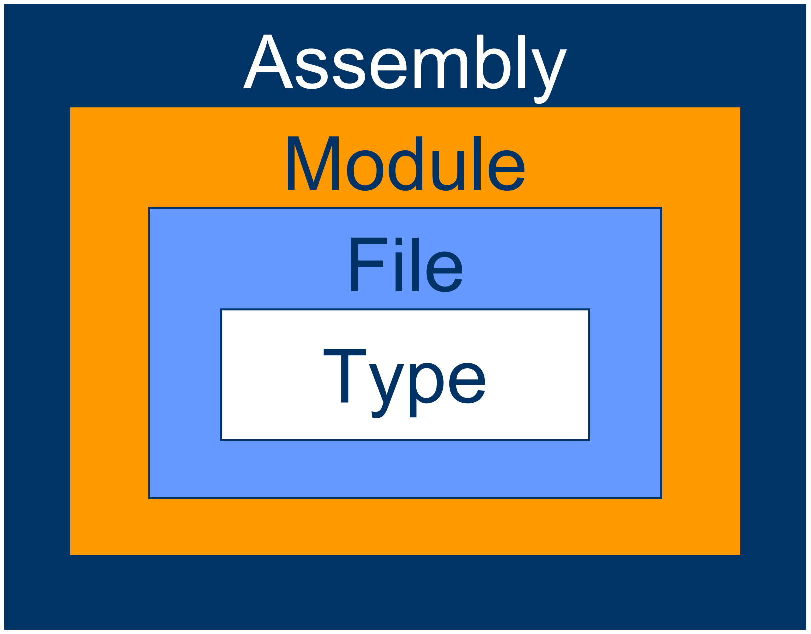
Components of an Assembly



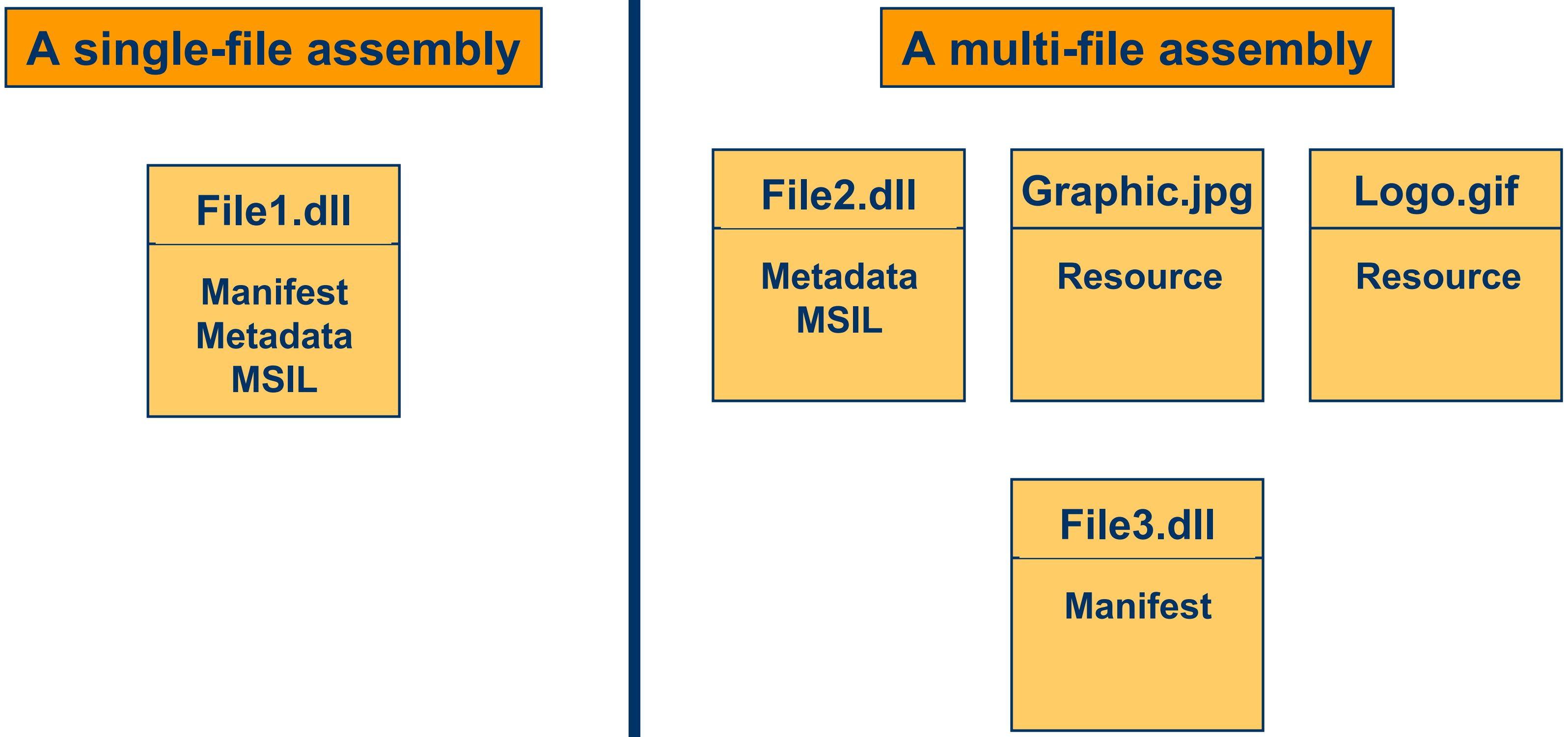
Assemblies
Components of an Assembly

- An assembly is a logical unit, not physical
  - It can consist of multiple modules (.DLL, .JPG, etc.)

In this figure, containment implies a 1:M relationship



Assemblies
Components of an Assembly



Assemblies

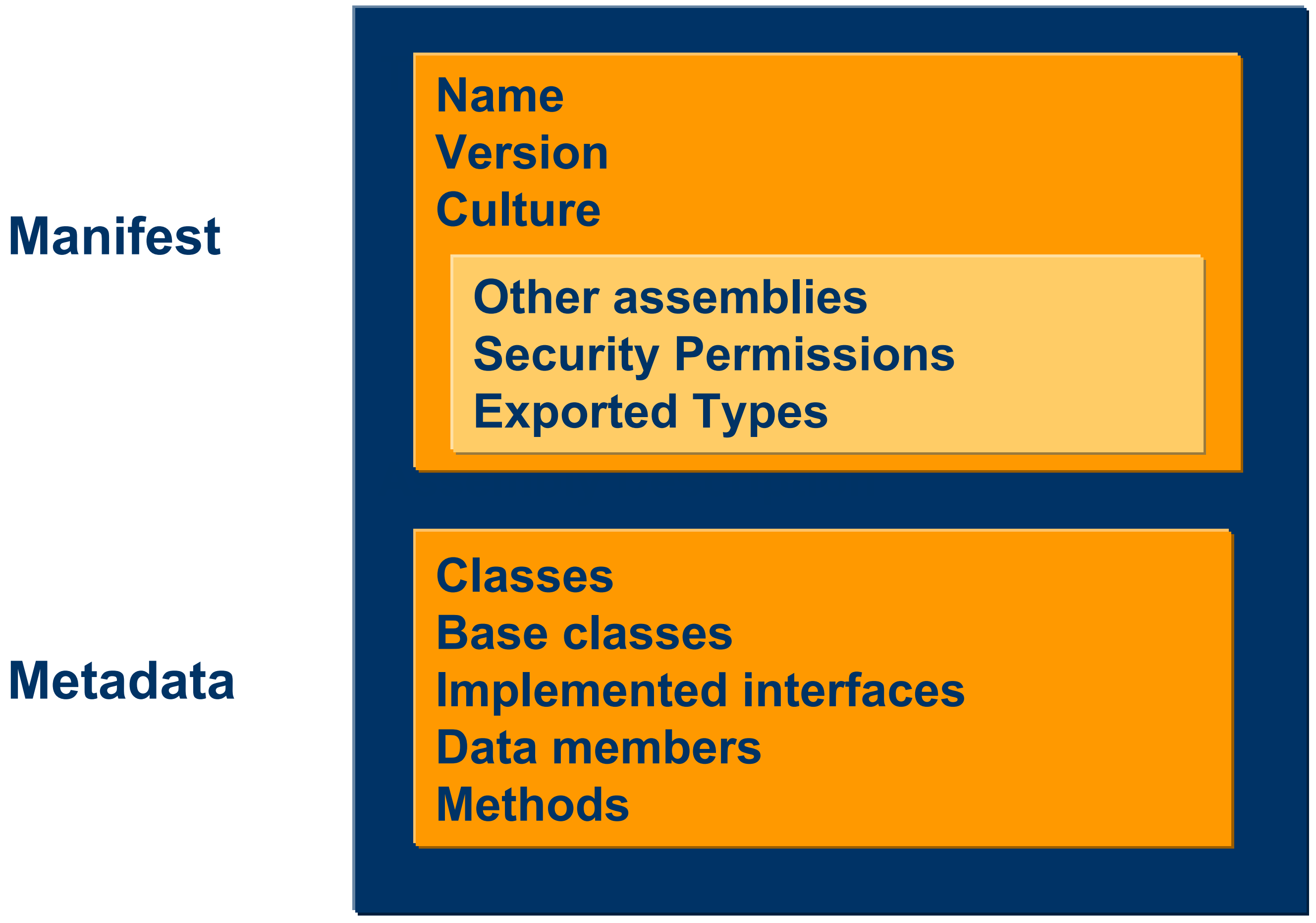
Assembly Generation Tool: `al.exe`

- Takes one or more files (containing either MSIL or resource files) and produces a file with an assembly manifest.
- When compiling a C# file, you can specify that it create a module instead of an assembly by using `/target:module`.

Assemblies
Manifest

- Manifest contains:
  - Identity information
    - Name, version number, culture, strong name
  - List of files in the assembly
  - Map of assembly types to files
  - Dependencies
    - Other assemblies used by this assembly
  - Exported types
  - Security permissions needed to run

Assemblies
Manifest and Metadata



Assemblies
What’s In the Metadata

- Description of types
  - Name, visibility, base class, interfaces implemented
  - Members
    - methods, fields, properties, events, nested types
- Attributes
  - User-defined
  - Compiler-defined
  - Framework-defined

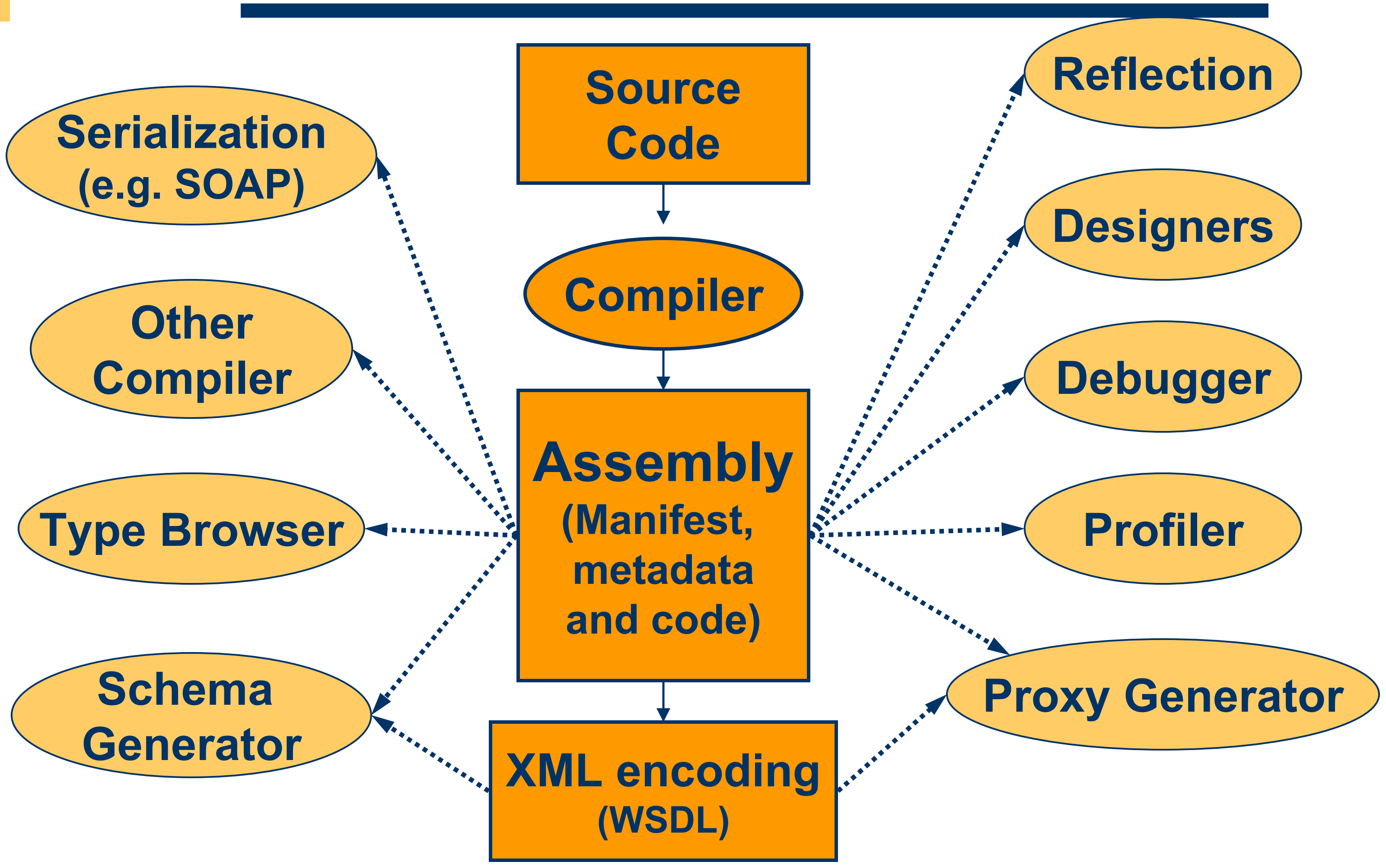
Assemblies
Demo: ILDASM.EXE

- Allows you to inspect the metadata and disassembled IL code in an assembly
- Great way to see what's really going on
- Use `ildasm /?` to see the various options

Assemblies
Metadata

- Key to simpler programming model
- Generated automatically
    - Stored with code in executable file (.dll or .exe)

Assemblies
Metadata: Creation and Use



Assemblies
Compilers Use Metadata

- For cross-language data type import
- Emit metadata with output code
  - Describe types defined and used
  - Record external assemblies referenced
  - Record version information
- Custom attributes can be used
  - Obsolete
  - CLS compliance
  - Compiled for debugging
  - Language-specific markers

Assemblies
Other Tools Use Metadata

- Designer behavior
  - Controlled by user-supplied attributes
    - Category
    - Description
- Designer extensibility
  - User-supplied attributes specify code to use
    - Type converters
    - Editors
- Web methods marked by custom attribute
- Type viewer

Assemblies
Global Assembly Cache

- A set of assemblies that can be referenced by any application on a machine
- Should be used only when needed
  - Private assemblies are preferred
- Located at `%SystemRoot%\assembly`
  - `(c:\winnt\assembly)`
- Add assemblies by
  - Installer program
  - `gacutil.exe`
  - Windows Explorer
    - Assembly Cache Viewer (`shfusion.dll`) is a shell extension for GAC that is installed with the .NET Framework SDK
  - .NET Framework Configuration Tool (`mscorcfg.msc`)
- Assembly must have a strong name

Assemblies
Strong Names

- Strong names identify an assembly
  - Contains text name, version, culture, public key, and digital signature
- Generated from an assembly using a private key
- Benefits
  - Guarantees name uniqueness
  - Protect version lineage
    - No one else can create a new version of your assembly
  - Provides strong integrity check
    - Guarantees that contents of an assembly didn’t change since it was built

Assemblies
Strong Names

- To sign an assembly with a strong name:
  - Use Assembly Generation tool: `al.exe`
  - Use assembly attributes (`AssemblyKeyFileAttribute` or `AssemblyKeyNameAttribute`)
- Requires a key pair (private and public)
  - To generate a key pair use the Strong Name tool: `sn.exe`

Assemblies

Demo: Installing an Assembly in GAC

- Create assembly
- Sign assembly with key from sn.exe
- Install into GAC via gacutil.exe, Assembly Cache Viewer and .NET Framework Configuration Tool

Assemblies
Signcode

- A strong name identifies an assembly but it does not authenticate an assembly
    - Strong names do NOT imply a level of trust
- Signcode allows the embedding of a certificate in an assembly
    - Now your assembly can be authenticated

Assemblies
Signcode

- To use signcode:
  - Obtain a Software Publisher Certificate (`.spc`)
  - Use `signcode.exe` to sign the assembly
- Signcode can only sign one file at a time
  - For an assembly, you sign the file containing the manifest

Assemblies
How Do You Obtain a Certificate?

- Purchase one from a well known Certificate Authority (such as Verisign)
- Create your own
  - For testing purposes only
  - Use `Makecert.exe` to create a X.509 certificate
  - Use `cert2spc.exe` to generate an SPC from a X.509 certificate

Assemblies
Strong Names and Signcode

- Strong names and signcode provide different, complimentary levels of protection
- You can assign a strong name or assign a signcode signature to an assembly, or both
- When using both, the strong name must be assigned first

Assemblies
Signcode

- Specify what permissions your assembly needs
  - Only specify required permissions
  - Handle optional permissions dynamically
- Set security policy on run-time machine

Assemblies

Deployment

- Unit of deployment
  - One or more files, independent of packaging
  - Self-describing via manifest and metadata
- Versioning
  - Captured by compiler
  - Policy per-application as well as per-machine
- 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 relative to assembly

Assemblies
Deployment

- Applications are configurable units
  - 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 to shared version
  - Version policy can be per application

Assemblies
MSIL

- Microsoft Intermediate Language

```
.assembly hello {}
.assembly extern mscorlib {}
.method static public void main() il managed {
  .entrypoint
  .maxstack 1
  ldstr "Hello World from IL!"
  call void [mscorlib]System.Console::WriteLine(class
                                                  System.String)
  ret
}
```

Assemblies
MSIL

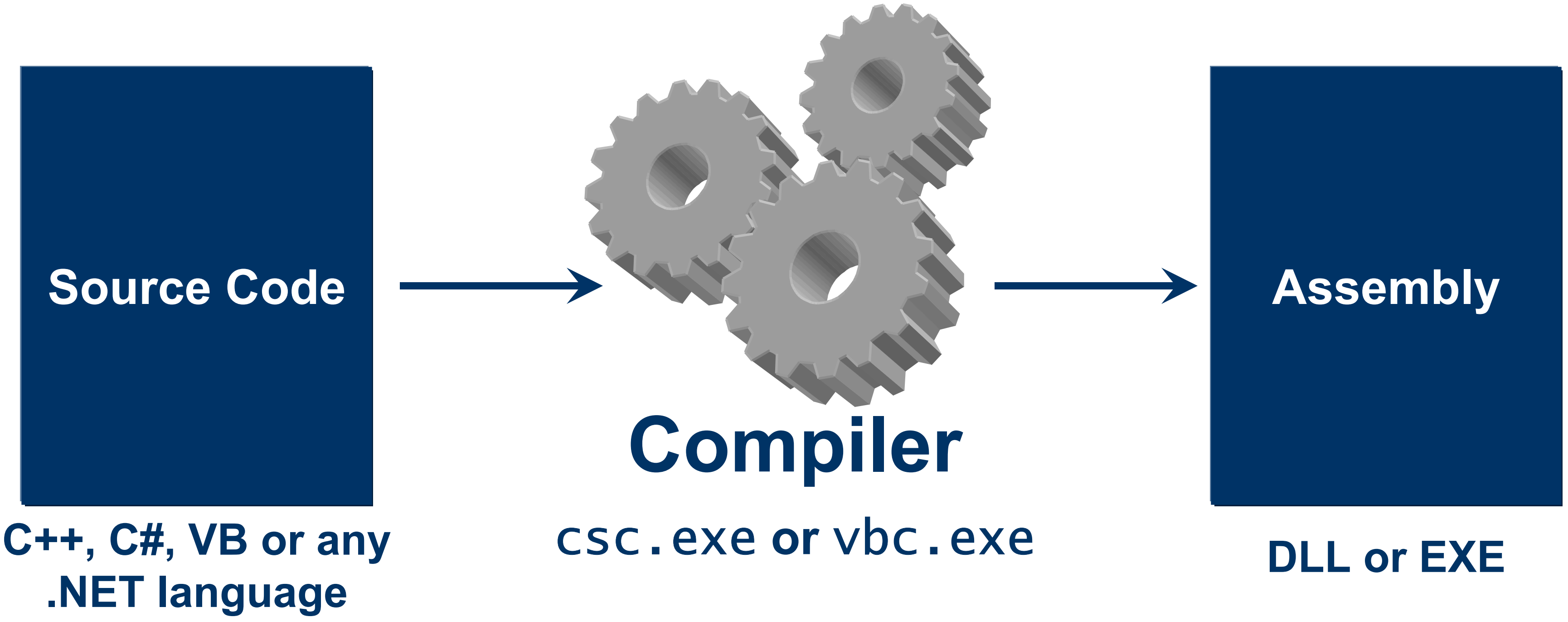
- Compiled with `ilasm.exe`
- MSIL was designed for the CLR
  - Object-oriented (primitives are not special)
  - Designed for the Common Type System
  - Does not embed type information
- See documentation in `\FrameworkSDK\Tool Developers Guide\docs`

Agenda

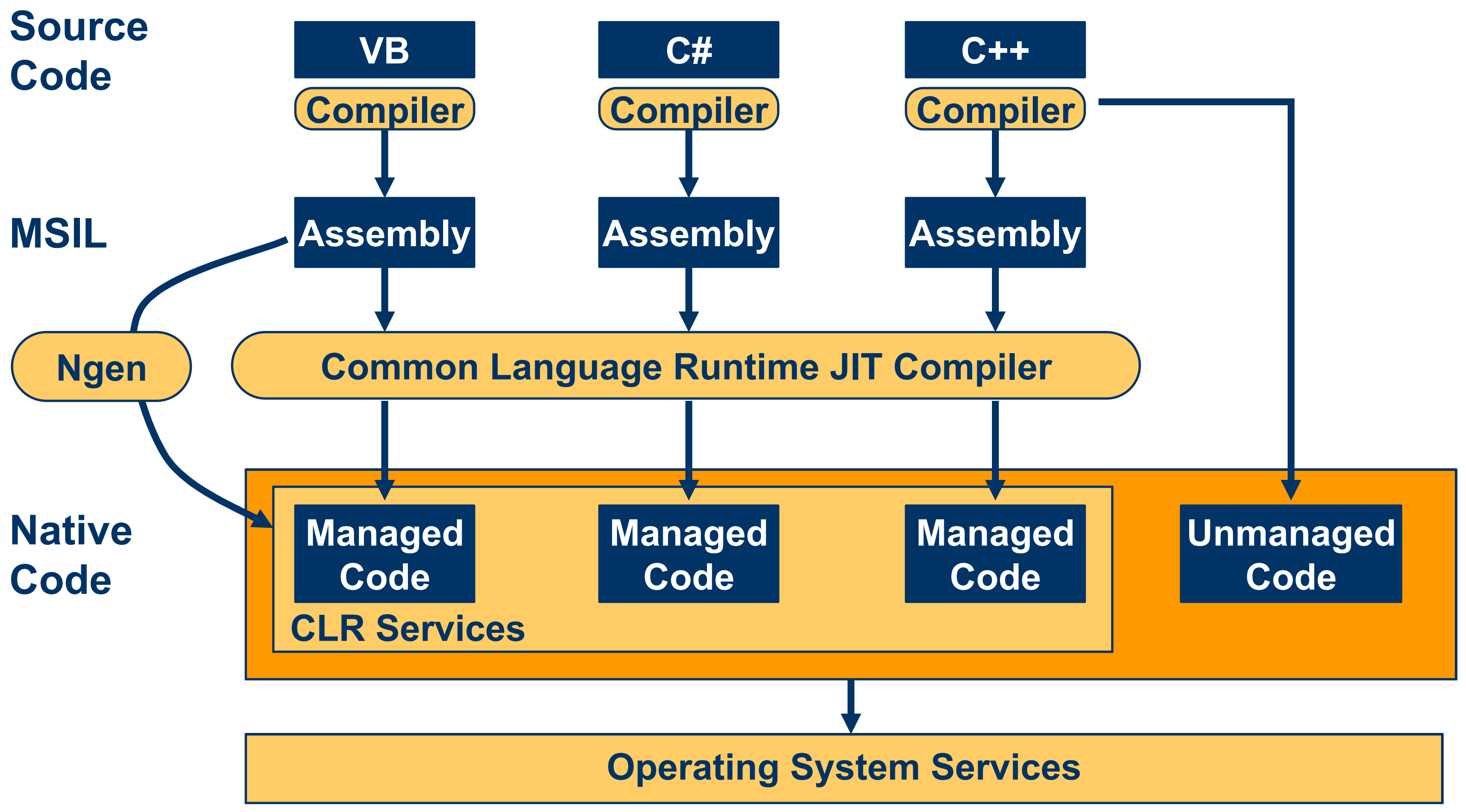
- What Is the CLR?
- Assemblies
- **Execution Model**
- Interoperability
- Security

Execution Model

Create Assembly



Execution Model



Execution Model
Compiling IL to Native Code

- JIT compiler
  - Generates optimized native code
  - Compiled when a method is first called
  - Includes verification of IL code
- `Ngen.exe`
  - Install-time native code generation
  - Used when assembly is installed on machine
  - Reduces start-up time
  - Native code has version checks and reverts to run-time JIT if they fail

Execution Model
Run-Time Hosts

- ASP.NET
- Internet Explorer
- Shell executables
- More in future
  - For example: SQL Server (Yukon)
- Can create your own run-time hosts

Execution Model
Binding to Assemblies

- An application consists of one or more assemblies.
- How does one assembly bind to another?
  - Based upon metadata and policy
    - Local (preferred)
    - Assembly Global Cache
- Multiple versions of an assembly may exist on the same machine.
  - Easier software deployment, updates and removal
  - Multiple versions of an assembly can even be used by the same application

Execution Model
Application Domains

- Traditionally, processes were used to isolate applications running on the same computer
  - Isolates failure of one application
  - Isolates memory
- Problems
  - Uses more resources
  - If needed, inter-process calls can be expensive

Execution Model
Application Domains

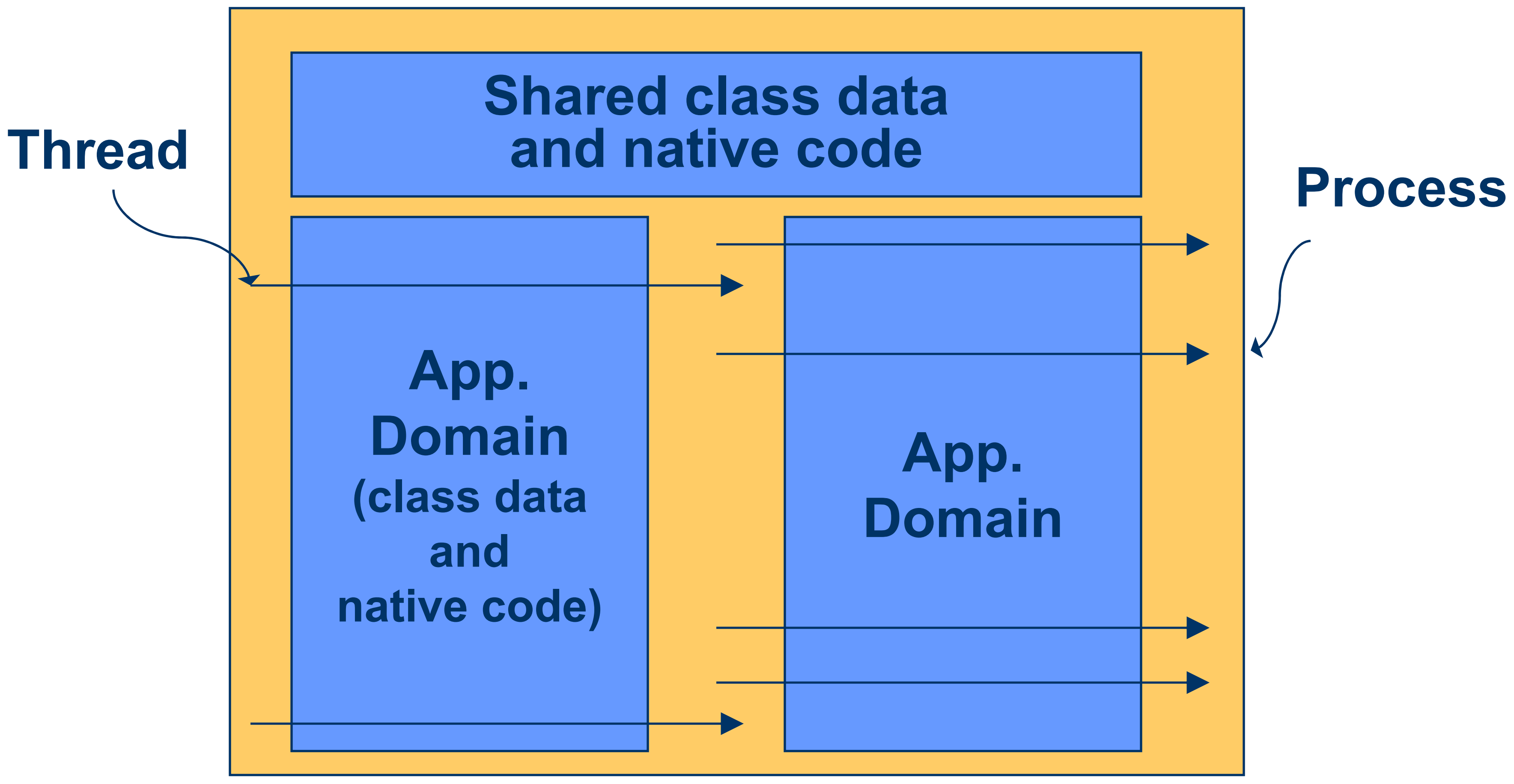
- .NET introduces Application Domains, which allow you to run multiple applications within the same process
- Enabled by code verification
  - No code will crash the process
- Managed by the `System.AppDomain` class
- Common assemblies can be shared across domains or can be specific to a domain

Execution Model
Application Domains

- Benefits:
  - Application domains are isolated
  - Faults are isolated
  - Individual applications can be stopped without stopping the process
  - Can configure each application domain independently
  - Can configure security for each domain
  - Cross-domain calls can be done through proxies
    - More efficient than cross-process calls

Execution Model

Application Domains


Shared class data
and native code
Thread
Process
App.
Domain
(class data
and
native code)
App.
Domain