# The Common Language Infrastructure

- [Ref: Common Language Infrastructure specification, https://www.ecma-international.org/wp-content/uploads/ECMA-335_6th_edition_june_2012.pdf]
- The Common Language Infrastructure (CLI) provides a specification for executable code and the execution environment in which it runs. It is a unifying infrastructure for designing, developing, deploying, and executing distributed components and applications.

# The Common Language Infrastructure

- [Ref: Common Language Infrastructure specification, https://www.ecma-international.org/wp-content/uploads/ECMA-335_6th_edition_june_2012.pdf]
- The Common Language Infrastructure (CLI) provides a specification for executable code and the execution environment in which it runs. It is a unifying infrastructure for designing, developing, deploying, and executing distributed components and applications.
  - The Common Type System (CTS) provides a rich type system that supports the types and operations found in many programming languages.

# The Common Language Infrastructure

- [Ref: Common Language Infrastructure specification, https://www.ecma-international.org/wp-content/uploads/ECMA-335_6th_edition_june_2012.pdf]
- The Common Language Infrastructure (CLI) provides a specification for executable code and the execution environment in which it runs. It is a unifying infrastructure for designing, developing, deploying, and executing distributed components and applications.
  - The Common Type System (CTS) provides a rich type system that supports the types and operations found in many programming languages.
  - Metadata is used to describe and reference the types defined by the CTS. Metadata is stored in a way that is independent of any particular programming language. Thus, metadata provides a common interchange mechanism for use between tools (such as compilers and debuggers) that manipulate programs, as well as between these tools and the VES.

# The Common Language Infrastructure

- [Ref: Common Language Infrastructure specification, https://www.ecma-international.org/wp-content/uploads/ECMA-335_6th_edition_june_2012.pdf]
- The Common Language Infrastructure (CLI) provides a specification for executable code and the execution environment in which it runs. It is a unifying infrastructure for designing, developing, deploying, and executing distributed components and applications.
  - The Common Type System (CTS) provides a rich type system that supports the types and operations found in many programming languages.
  - Metadata is used to describe and reference the types defined by the CTS. Metadata is stored in a way that is independent of any particular programming language. Thus, metadata provides a common interchange mechanism for use between tools (such as compilers and debuggers) that manipulate programs, as well as between these tools and the VES.
  - The Common Language Specification (CLS) is an agreement between language designers and framework (that is, class library) designers. It specifies a subset of the CTS and a set of usage conventions.

# The Common Language Infrastructure

- [Ref: Common Language Infrastructure specification, https://www.ecma-international.org/wp-content/uploads/ECMA-335_6th_edition_june_2012.pdf]
- The Common Language Infrastructure (CLI) provides a specification for executable code and the execution environment in which it runs. It is a unifying infrastructure for designing, developing, deploying, and executing distributed components and applications.
  - The Common Type System (CTS) provides a rich type system that supports the types and operations found in many programming languages.
  - Metadata is used to describe and reference the types defined by the CTS. Metadata is stored in a way that is independent of any particular programming language. Thus, metadata provides a common interchange mechanism for use between tools (such as compilers and debuggers) that manipulate programs, as well as between these tools and the VES.
  - The Common Language Specification (CLS) is an agreement between language designers and framework (that is, class library) designers. It specifies a subset of the CTS and a set of usage conventions.
  - The Virtual Execution System (VES) implements and enforces the CTS model. The VES is responsible for loading and running programs written for the CLI. It provides the services needed to execute managed code and data, using the metadata to connect separately generated modules together at runtime.

# The Common Language Runtime

- [Ref: Smith & Nair, *Virtual Machines: Versatile Platforms for Systems and Processes*, §5.5]
- [Ref: CLR Overview, https://docs.microsoft.com/en-us/dotnet/standard/clr]
- The Common Language Runtime (CLR) is an implementation of the CLI and is part of Miscrosoft's overall .NET framework.
  - Modules are of a standard format, containing metadata and code in the Microsoft intermediate language (MSIL).
  - Supports multiple languages, including C#, Java, Visual Basic .NET, and Managed C++.

# Validity and Verifiability

- Code modules can be verified for type safety similar to Java binary classes.
  - Verifiability is desirable but not mandatory.
  - Verification helps establish a run-time protection domain that allows untrusted code to execute safely.
  - User can explicitly permit the use of verified and unverified modules through a security manager.

# Validity and Verifiability

- Code modules can be verified for type safety similar to Java binary classes.
  - Verifiability is desirable but not mandatory.
  - Verification helps establish a run-time protection domain that allows untrusted code to execute safely.
  - User can explicitly permit the use of verified and unverified modules through a security manager.
- However, all programs must be valid.
  - E.g., a stack underflow that is detectable by inspecting the code at load time.

# Validity and Verifiability

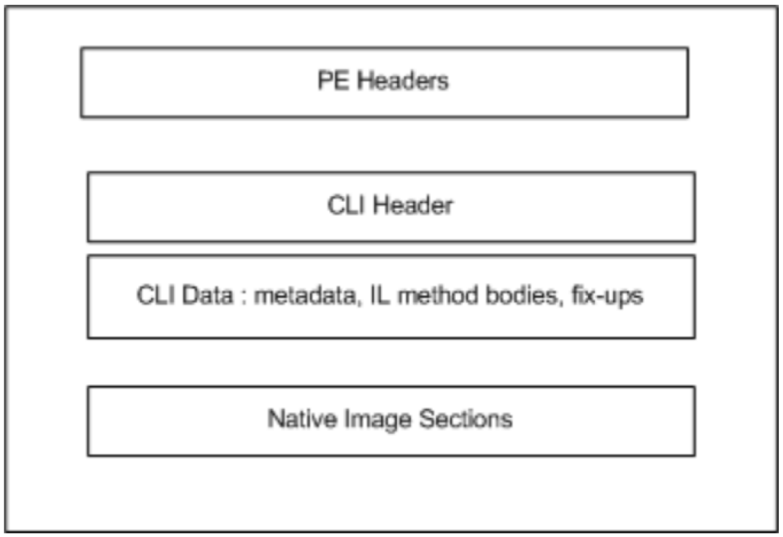
- Code modules can be verified for type safety similar to Java binary classes.
  - Verifiability is desirable but not mandatory.
  - Verification helps establish a run-time protection domain that allows untrusted code to execute safely.
  - User can explicitly permit the use of verified and unverified modules through a security manager.
- However, all programs must be valid.
  - E.g., a stack underflow that is detectable by inspecting the code at load time.
- Three categories of application programs.
  - Those that are verifiable and valid.
  - Those that are unverifiable and valid.
  - Those that are valid.

# Validity and Verifiability

- Code modules can be verified for type safety similar to Java binary classes.
  - Verifiability is desirable but not mandatory.
  - Verification helps establish a run-time protection domain that allows untrusted code to execute safely.
  - User can explicitly permit the use of verified and unverified modules through a security manager.
- However, all programs must be valid.
  - E.g., a stack underflow that is detectable by inspecting the code at load time.
- Three categories of application programs.
  - Those that are verifiable and valid.
  - Those that are unverifiable and valid.
  - Those that are valid.
- Java only allows verifiable and unverifiable classes.

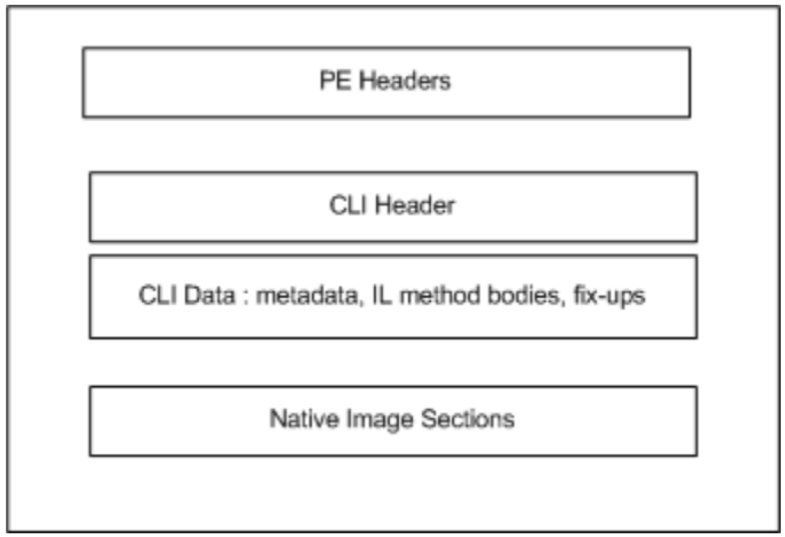
# Assemblies

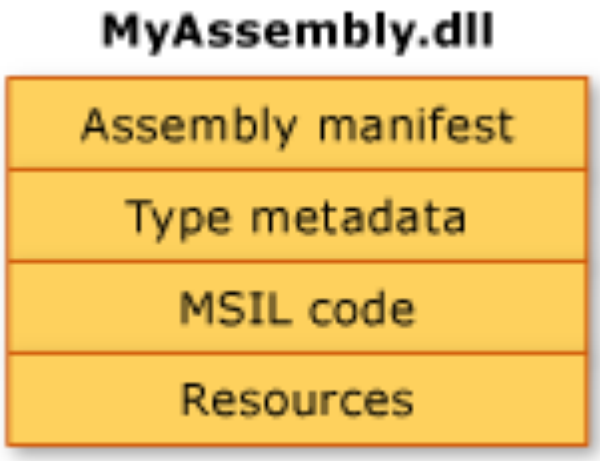
- Assemblies form the fundamental units of deployment, version control, reuse, activation scoping, and security permissions for .NET-based applications.




# Assemblies

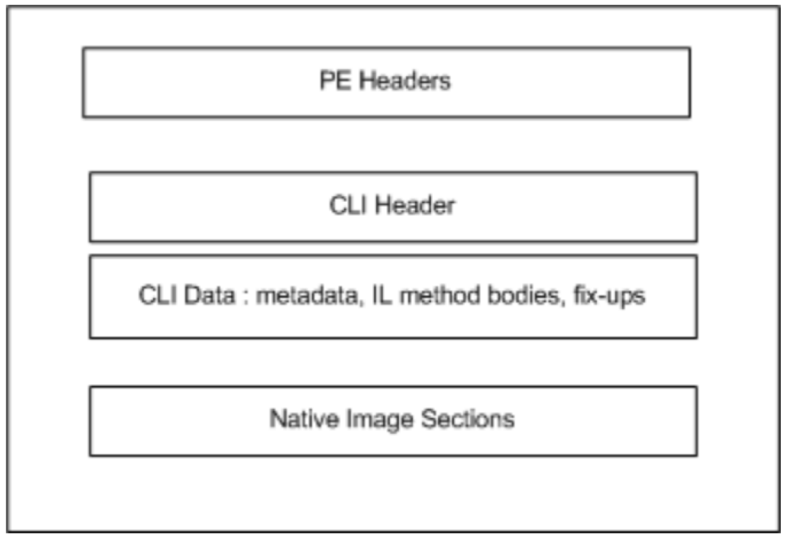
- Assemblies form the fundamental units of deployment, version control, reuse, activation scoping, and security permissions for .NET-based applications.
  - An assembly is a collection of types and resources that are built to work together and form a logical unit of functionality.



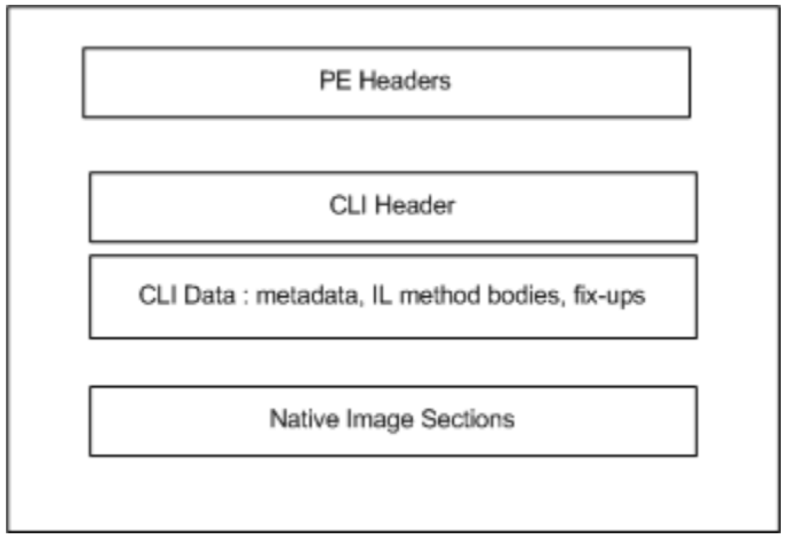
# Assemblies

- Assemblies form the fundamental units of deployment, version control, reuse, activation scoping, and security permissions for .NET-based applications.
  - An assembly is a collection of types and resources that are built to work together and form a logical unit of functionality.
  - Assemblies take the form of executable (`.exe`) or dynamic link library (`.dll`) files in PE/COFF format.



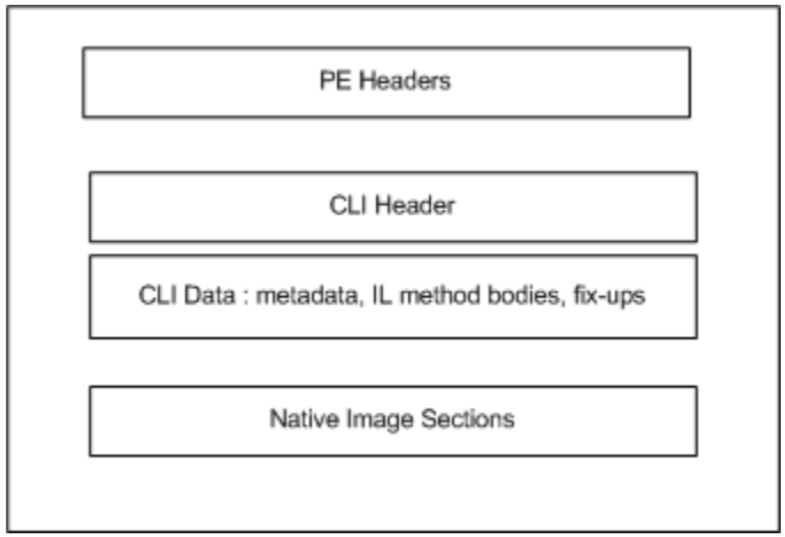

# Assemblies

- Assemblies form the fundamental units of deployment, version control, reuse, activation scoping, and security permissions for .NET-based applications.
  - An assembly is a collection of types and resources that are built to work together and form a logical unit of functionality.
  - Assemblies take the form of executable (`.exe`) or dynamic link library (`.dll`) files in PE/COFF format.
  - They provide the CLR with the information it needs to be aware of type implementations.




# Assemblies

- Assemblies form the fundamental units of deployment, version control, reuse, activation scoping, and security permissions for .NET-based applications.
  - An assembly is a collection of types and resources that are built to work together and form a logical unit of functionality.
  - Assemblies take the form of executable (`.exe`) or dynamic link library (`.dll`) files in PE/COFF format.
  - They provide the CLR with the information it needs to be aware of type implementations.
  - Assemblies are only loaded into memory if they are required.




# MSIL

- Object-oriented, stack-based bytecode.
  - Similar in many ways to Java bytecode.
  - Typically JIT-compiled into native code.

# MSIL

- Object-oriented, stack-based bytecode.
  - Similar in many ways to Java bytecode.
  - Typically JIT-compiled into native code.
- Differences from Java bytecode.
  - The local data area and argument area for an MSIL method are not defined to be part of a stack frame (although they might be implemented as such).
  - Rather than a single constant pool, MSIL supports a number of metadata tables called streams.
    - These tables are accessed by tokens identifying the stream and the entry within it.
  - MSIL instructions are type-generic (only one `add` instruction, with the type being inferred from the operand types).
  - Certain MSIL instructions are unverifiable, in order to support C-like memory operations and control flow.

# Cross-Language Interoperability

- Interoperability is supported in a more integrated way than in Java, extending to data.
  - A type defined in one language can be used across other languages.
    - Made possible by common metadata.
    - Hidden cost: may require significant changes to existing language implementations.
  - The CLS component of the CLI contains a set of standard rules intended to ensure interoperability.