

A decorative graphic on the left side of the slide, consisting of a grid of squares in various shades of gray and blue, arranged in a pattern that tapers off to the right.

# .NET: Architektur

© J. Heinzelreiter  
Version 5.8

seit 2000

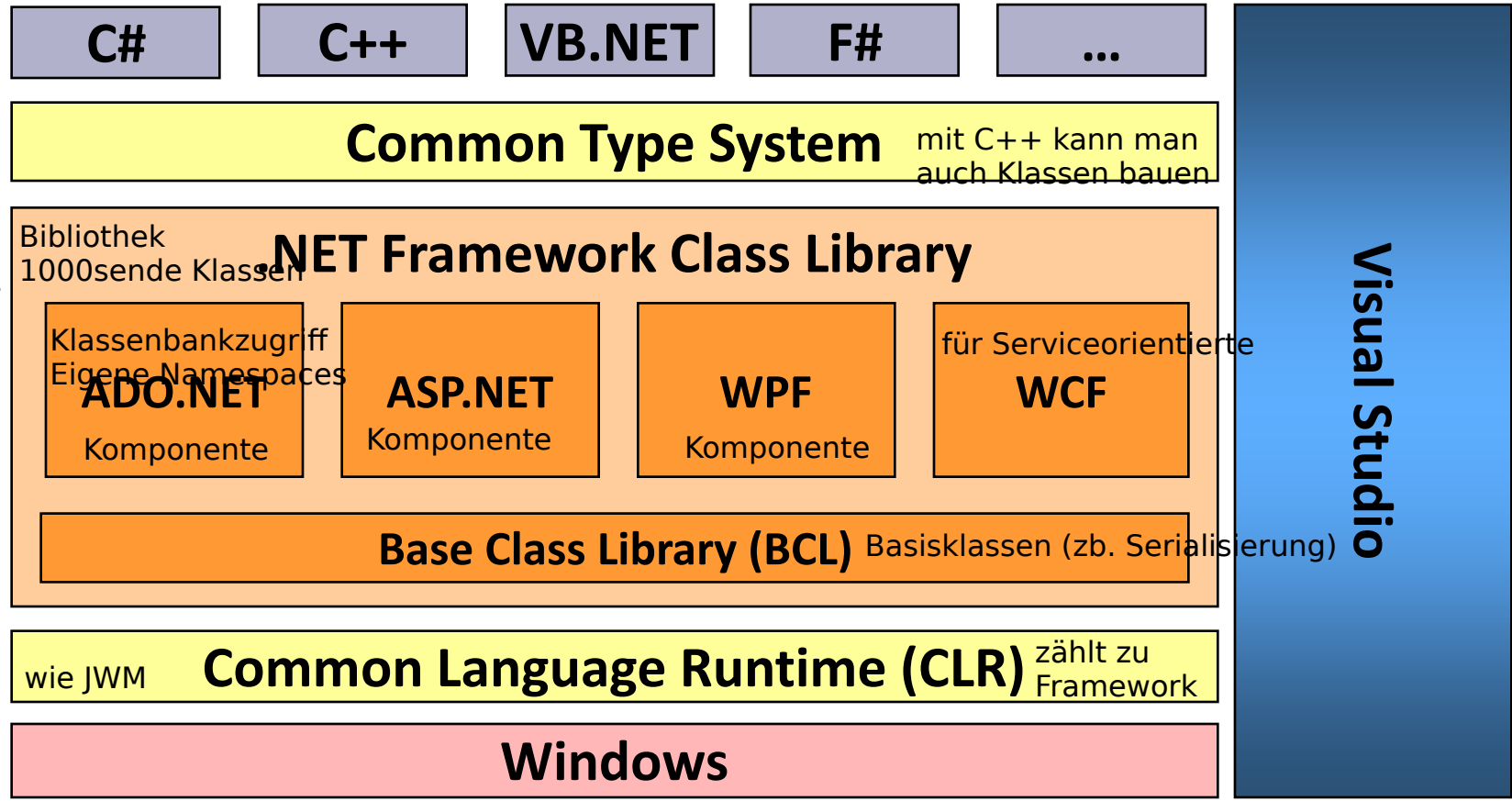
konnte nicht alle anforderungen der modernen Web  
Welt abdecken

# .NET Framework 2.0 – 4.7 (Full Framework)

Als SDK herunterladen

Definition Framework:

in der Industrieverbreitet



Besonders  
ist das CTS  
Alle Klassen  
für verschiedene  
Sprachen nutzbar

# Varianten von .NET

- Das **.NET-Framework** steht ausschließlich für Windows zur Verfügung.
- **.NET Core** (wird in der letzten Übung verwendet)
  - Open-Source-Projekt unter Führung von Microsoft Community hat sich beteiligt sich dran
  - *CoreFX* enthält Basisfunktionalität der .NET-Framework-Bibliothek
  - *CoreCLR* ist die Laufzeitumgebung
  - Unterstützte Plattformen: Linux, Mac OS X, Windows (Full Framework nur unter Windows)
- **Mono** (seit 2001)
  - Open-Source-Projekt, vom Fullframework abgeleitet (umfassender aber nicht Full. kein WPF) Compiler ist Plattform-übergreifend
  - Laufzeitumgebung zu .NET kompatibel,
  - stellt große Teile der Funktionalität des .NET-Framework zur Verfügung
  - Unterstützte Plattformen: Linux, Mac OS X, Windows
- Die **Xamarin-Plattform** (um mobile Applikationen Plattformen unabhängig zu entwickeln)
  - basiert auf Mono und
  - ermöglicht die Entwicklung von nativen mobilen Anwendungen für iOS, Android und Windows Phone.
  - Xamarin wurde Anfang 2016 von Microsoft übernommen.

sehr viele Projekte werden auf GitBub gehostet.

# .NET Core

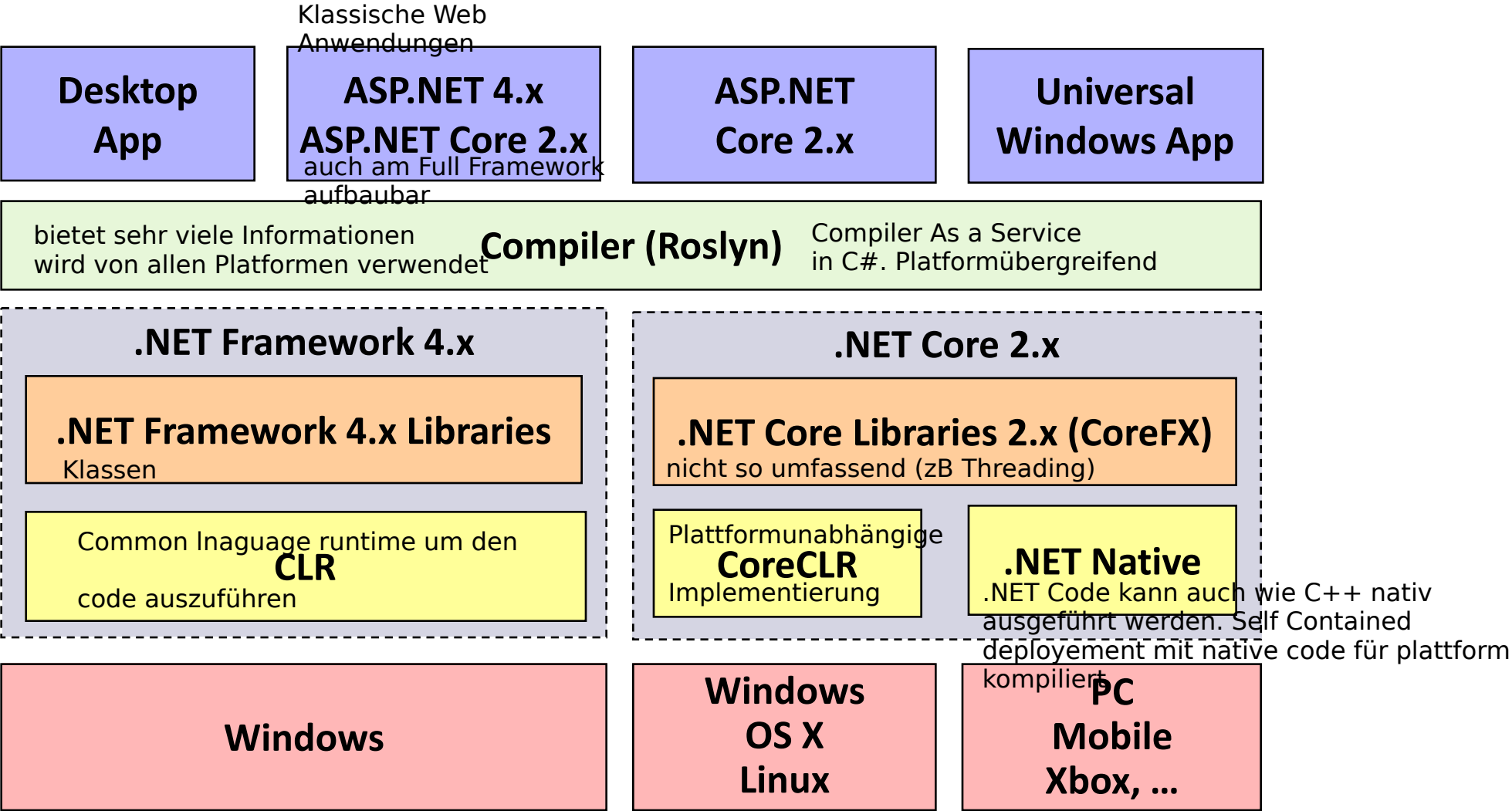
## ■ Motivation

- Verschiedene Varianten des .NET-Frameworks für Desktop, Store Apps, Windows Phone.  
man wollte konsolidierung herstellen weil verschiedene Versionen von .NET
- Entwicklung Framework-übergreifender Anwendung ist schwierig (→ Portable Class Libraries).
- Maschinenweite Installation: Verschiedene Versionen beeinflussen sich gegenseitig.  
Am Webserver zB nur .NET 2.0. Ihc möchte features von .NET 4 nutzen. Entwickler kann self contained deployment nutzen

## ■ Eigenschaften

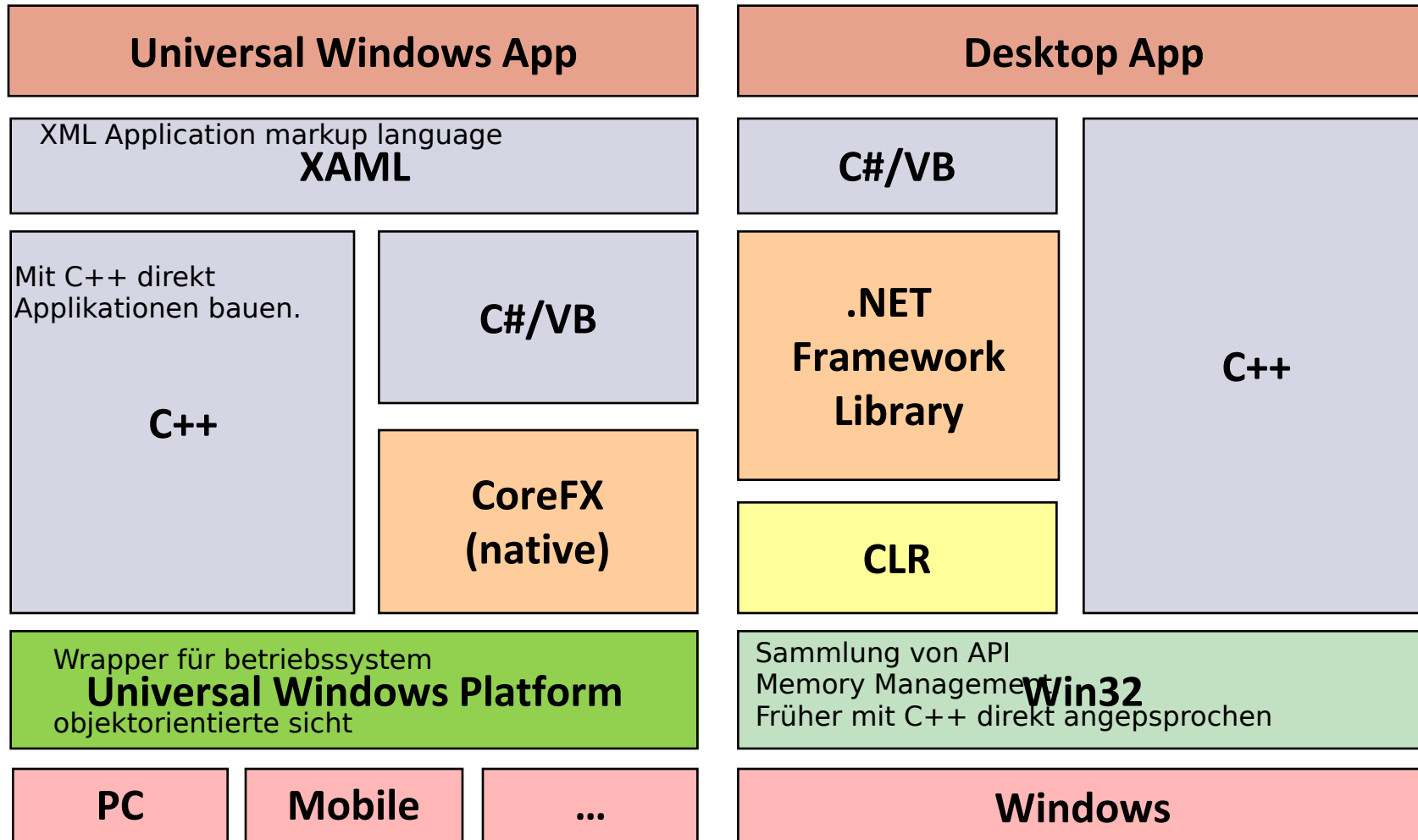
- Plattformübergreifende Implementierung
- Gemeinsame Codebasis für unterschiedliche Anwendungsgebiete (Windows Apps, Web-Anwendungen)
- Komponenten werden über Nuget bereitgestellt nuget Packetmanager für :NET. Sehr viel modularer durch nuget
- Häufigere Releasezyklen (4 Snapshots pro Jahr), einzelne Assemblys können ausgetauscht werden.
- Anwendung und Assemblys können gemeinsam deployt werden. self contained deployment  
esonders festure im web bereich  
um verschiedene features zu nutzen

# Die .NET-Plattform

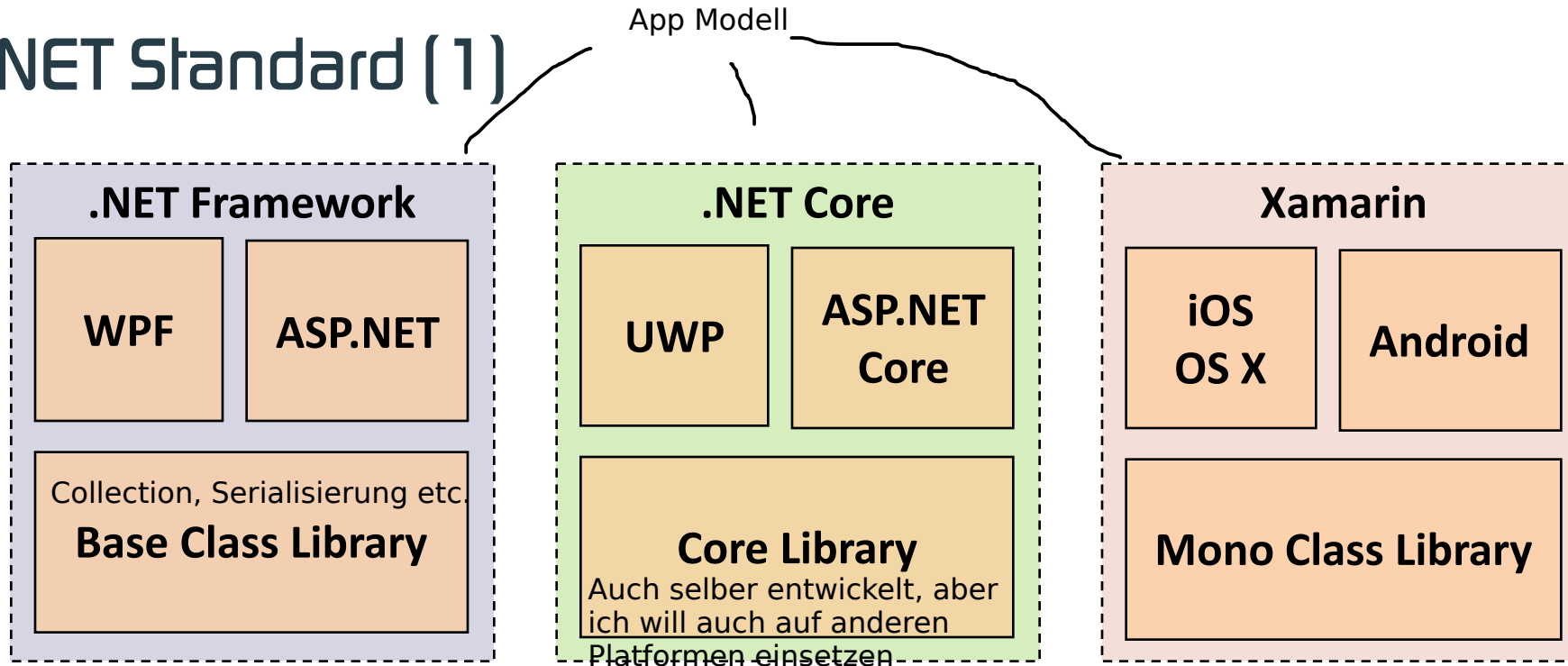


# .NET Framework und die UWP (Windows 10)

Technologiestack anordnung

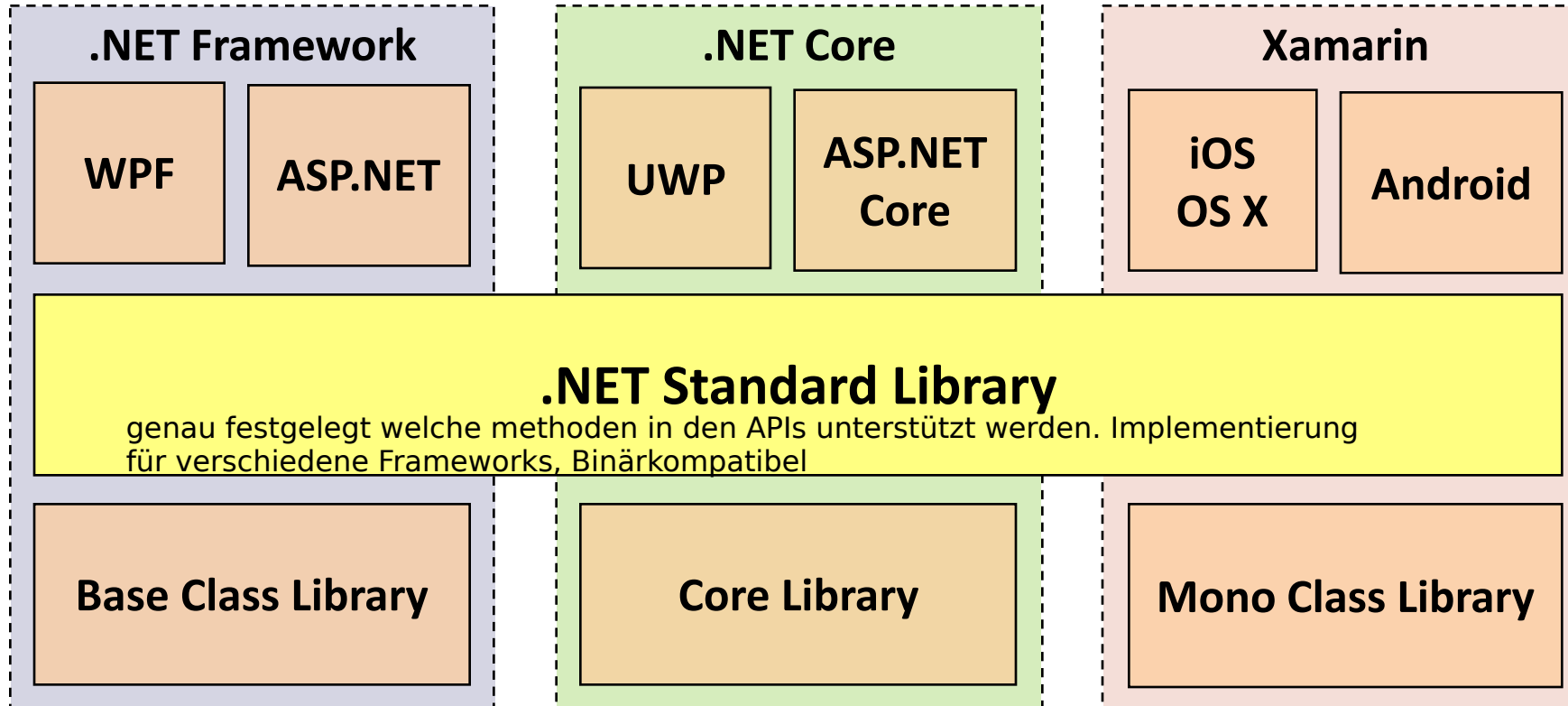


# .NET Standard (1)



- Es existieren viele verschiedene Varianten von .NET.
- Portieren von Code ist aufwändig, da sich auch die Basis-Bibliotheken unterscheiden.
- Portable Class Libraries (PCL) sind unbefriedigend, da Entwickler auf größte gemeinsame Funktionalität eingeschränkt ist.

## .NET Standard (2)

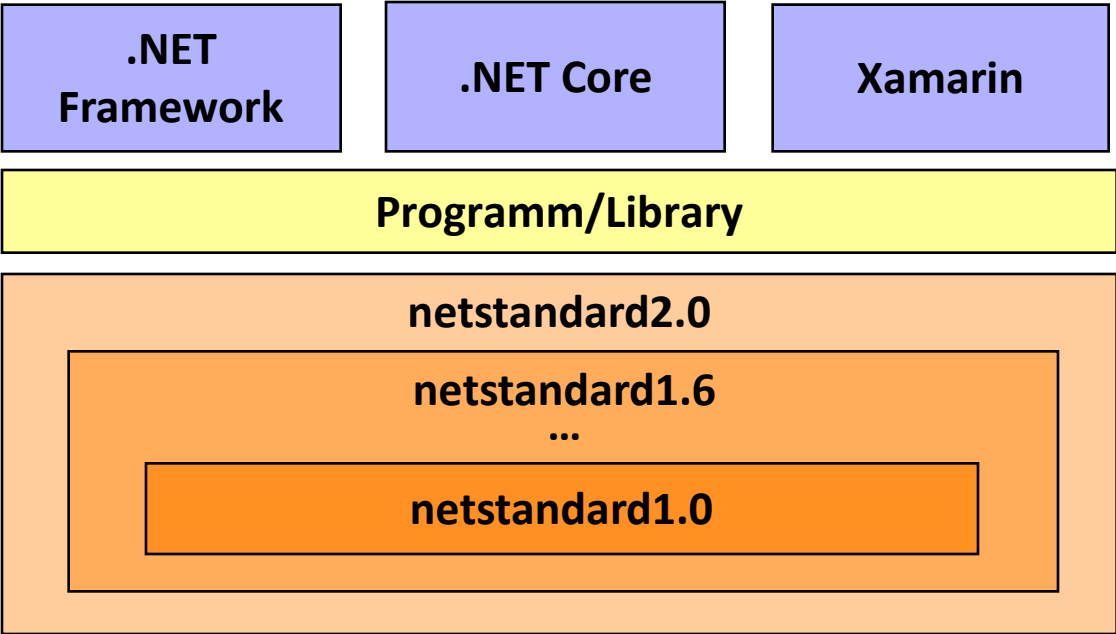


- Die *.NET-Standard-Library* ist eine Spezifikation von .NET-APIs, die von mehreren Laufzeitumgebungen unterstützt werden.



# .NET Standard (3)

- Gemeinsame APIs entstehen nicht zufällig, sondern im Rahmen eines Standardisierungsprozesses.
- Lineare Versionierung:  
1.0 < 1.2 < ... < 2.0
- Binärkompatible Komponenten
- Kompatibilitätsmatrix:



immer mehr erweitert neuer standard ist superset

Plattform-Name	Alias								
.NET Standard	netstandard	1.0	1.1	1.2	1.3	1.4	1.5	1.6	2.0
.NET Core	netcoreapp	1.0	1.0	1.0	1.0	1.0	1.0	1.0	2.0
.NET Framework	net	4.5	4.5	4.5.1	4.6	4.6.1	4.6.1	4.6.1	4.6.1
Mono		4.6	4.6	4.6	4.6	4.6	4.6	4.6	5.4
Universal Windows Platform	uap	10.0	10.0	10.0	10.0	10.0	10.0.16299	10.0.16299	10.0.16299

Kompatibilitätsmatrix verstehen. Komponente weiter vorne. .NET Standardversion auswählen. bei neuen Komponenten wichtig

# NET Framework Compatibility Mode

- Rückwärtskompatibilität

- Anwendungen, die mit Framework x entwickelt wurden, funktionieren mit Laufzeitumgebung von Framework y ( $y > x$ ).  
nuget Sammlung alle pakete für alle.

- Full Framework

- Anwendung läuft auf jenem Framework, mit dem es entwickelt wurde.
- Mehrere Frameworks können parallel installiert sein („side-by-side execution“)
- Anwendungen für .NET-Framework  $\geq 4.5$  sind rückwärtskompatibel.

- .NET Standard 2.0 → **NET Framework Compatibility Mode**

- Beliebige *binäre* .NET-Komponenten können referenziert werden.
- Warnung, dass Komponente möglicherweise nicht kompatibel ist.
- Viele NuGet-Komponenten können unverändert verwendet werden.
- Komponenten, die nicht unterstützte Bibliotheken verwenden (z. B. WPF) verursachen einen Laufzeitfehler.

nimm eine bestehende dll her.  
wenn in dieser dll ausschließlich  
methoden (APIs) wo nur .NET standard  
2.0 komponenten dann. unter mono  
unter linux alte dll verwendbar wenn  
nur alte API verwendet wird  
wenn WPF call dann laufzeit exception

# Common Language Runtime (CLR) (1)

gibt ISO Standard dazu

- Jedes Framework enthält eine Implementierung der Laufzeitumgebung:
  - Full Framework → *CLR*
  - .NET Core → *CoreCLR*
- CLR führt .NET-Anwendungen aus: Code ausführen. Sicherheitsüberprüfungen, Garbage Collection
  - führt Sicherheitsüberprüfungen aus.
  - übernimmt Speicherverwaltung und Fehlerbehandlung.
  - lädt dynamisch Komponenten (richtige Version).
- CLR stellt Verbindung zum Betriebssystem her. stellt fest ob gewisse Betriebssystemfunktionalität zur Verfügung stehen
- CLR versteht eine Zwischensprache, in die alle .NET-Programme übersetzt werden → IL. IL gegenstück zum Bytecode
- Die Typen der Programme müssen sich an gewisse Spielregeln halten → CTS. Compiler muss bestimmtes layout erzeugen.
- Zwischensprache und Informationen über Programme werden in Assemblys verpackt. wird noch genauer behandelt in welcher Form .NET komponenten zur Verfügung stellt

# Common Language Runtime (CLR) (2)

- CLR ist eine *virtuelle Maschine* = mit Software realisierter Prozessor.
- Vorteile:
  - Plattformunabhängigkeit: CLR kann auf andere reale Maschinen portiert werden.
  - Sprachunabhängigkeit: Compiler übersetzen in Sprache der CLR.
  - Kompakter Code.
  - Optimierter Code: CLR kann Spezifika der Zielmaschine berücksichtigen.  
performanceoptimiert auf die plattform
- CLR ist eine Stackmaschine. wie JVM
  - Einfache Codegenerierung

# Unterschiede zur Java Virtual Machine

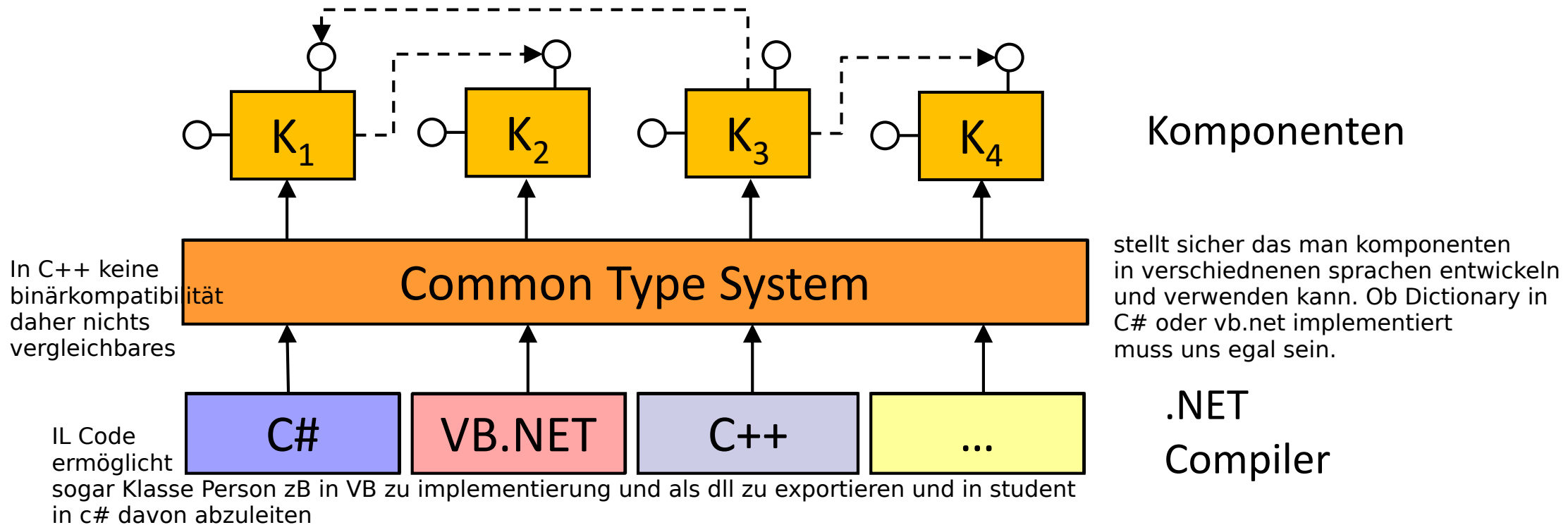
- Kompilation/Interpretation
  - Bytecode wird von JVM interpretiert (Bytecode ist darauf ausgelegt).
  - Hotspot-JVMs übersetzen Teile des Bytecodes, wenn dieser häufig aufgerufen wird.
  - CLR übersetzt Zwischencode immer → JIT-Compiler. nur die methoden die tatsächlich aufgerufen werden. beginnt nach programmstart immer wieder von vorne
- Unterstützung verschiedener Sprachen und –Paradigmen (OO, prozedurale, funktionale Sprachen)
- Selbstdefinierte Wertetype (Strukturen)
- Aufruf per Referenz geht nicht in Java
- Typsichere Methodenzeiger (Delegates)

# Komponenten der CLR

- *CTS*: Common Type System
- *CLS*: Common Language Specification
- *CIL*: Common Intermediate Language common steht fpr standard. IL
- *JIT*: Just in Time Compiler
- *VES*: Virtual Execution System

# Common Type System (CTS)

- CTS legt fest, wie Typen im Speicher repräsentiert werden:
  - Objektorientiertes Programmiermodell,
  - ermöglicht sprachübergreifende Verwendung von Typen.



# Common Type System – Beispiel

## VB.NET

```
Public Class Person
    Private name As String

    Public Sub New(ByVal n
                    As String)
        name = n
    End Sub

    Public Function
        GetName() As String
        GetName = name
    End Function
End Class
```

**vbc** /target:library **Person.vb**

## C#

```
public class Student : Person {
    private string id;

    public Student(string name,
                  String id):
        base(name) {
            this.id = id;
        }

    public string GetId() {
        return id;
    }
}
```

**csc** /r:Person.dll  
/target:exe **Student.cs**



# Common Language Specification (CLS)

unter welchen Voraussetzungen. Ist  
Klasse CLS compiled

- Programmierrichtlinien für *Entwickler*, die garantieren, dass Klassen *sprachübergreifend* eingesetzt werden können:
  - Als Parametertypen bei öffentlichen Methoden darf nur eine Untermenge der CTS-Standardtypen verwendet werden.
  - Groß-/Kleinschreibung nicht ausnutzen.
  - Verschiedene Namen für Methoden und Felder.
- Regeln, die *Compilerbauer* einhalten müssen:
  - Vorschriften, gewisse Metadaten zu generieren.
- Compiler kann CLS-Konformität überprüfen.

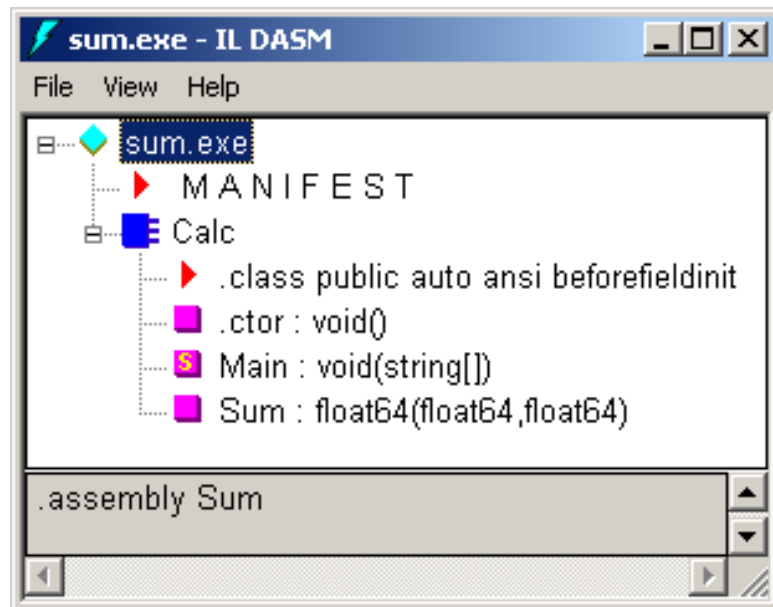
# Intermediate Language (IL)

- Zwischencode, der von allen .NET-Compilern erzeugt wird.  
wird später JIT kompiliert
- Zwischencode wird von CLR ausgeführt.
- IL entspricht dem Bytecode von Java.
- C++-Compiler kann IL- oder nativen Code generieren.
- Andere Bezeichnungen:
  - CIL: Common Intermediate Language (ECMA, ISO).
  - MSIL: Microsoft Intermediate Language.

Reverse Compiler VB.NET komponente in C# übersetzen  
Disassembler

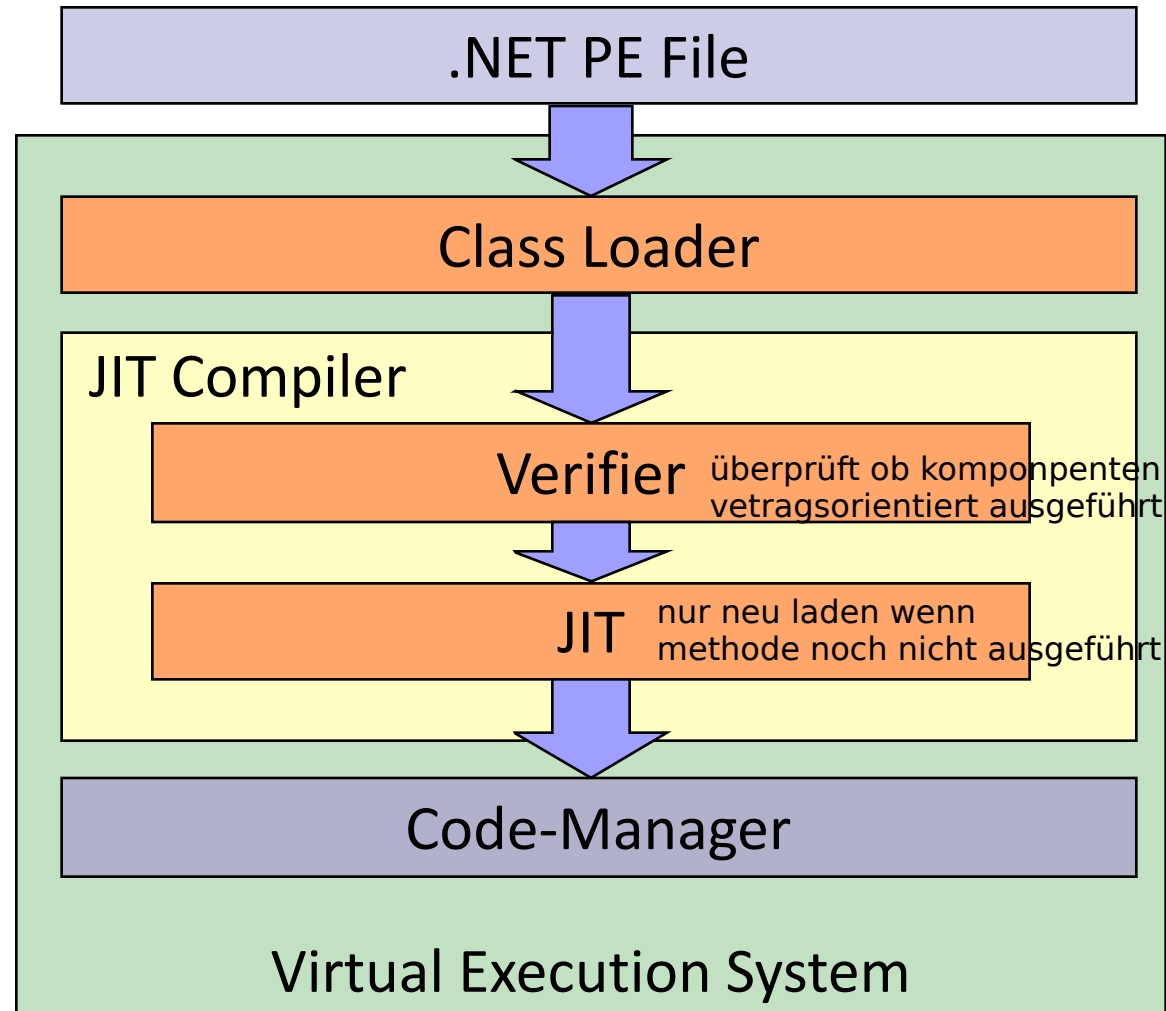
# IL DASM

```
public class Calc {  
    public double Sum(  
        double d1, double d2) {  
        return d1+d2;  
    }  
    ...  
}
```



```
.method public hidebysig instance  
float64 Sum(float64 d1,  
            float64 d2) cil managed {  
    // Code size      8 (0x8)  
    .maxstack 2  
    .locals init (float64 V_0)  
IL_0000:  ldarg.1  
IL_0001:  ldarg.2  
IL_0002:  add  
IL_0003:  stloc.0  
IL_0004:  br.s IL_0006  
IL_0006:  ldloc.0  
IL_0007:  ret  
} // end of method Calc::Sum
```

# Virtual Execution System (VES)



# Aufgaben des VES (1)

- Class-Loader
  - Suchen der Assemblys (Arbeitsverzeichnis oder GAC).
  - Vorbereitung zur Ausführung: Einfügen von Stubs für JIT-Kompilierung.
- Verifier:
  - Überprüfung der Typsicherheit
    - Code darf nur auf berechtigte Bereiche zugreifen.
    - Objekte werden nur über deren Schnittstelle angesprochen.
  - Für typsichere Assemblys kann garantiert werden, dass sie sich nicht gegenseitig beeinflussen (→ *Application Domains*).

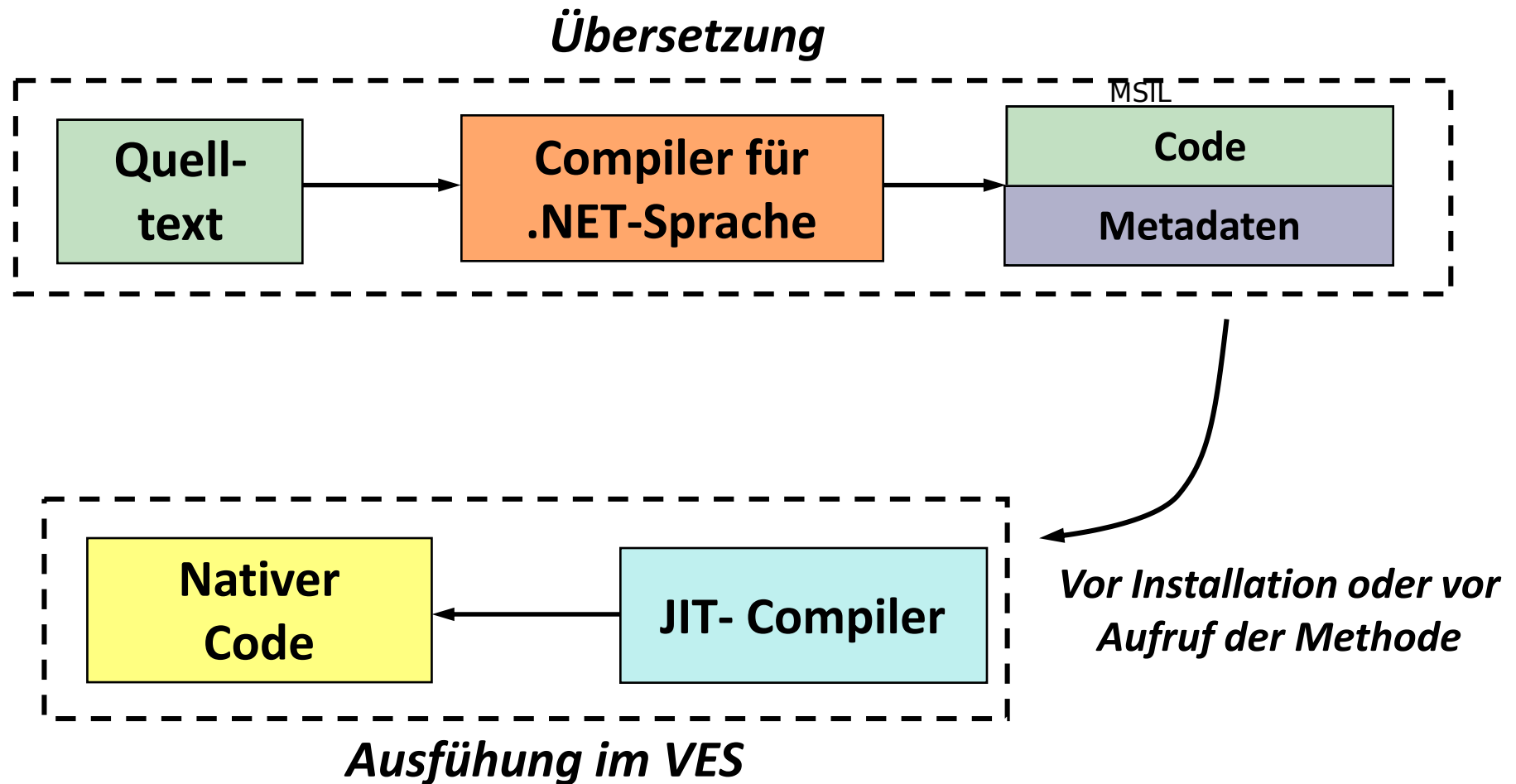
# Aufgaben des VES (2)

- Kategorien von IL-Code
  - Ungültiger Code
    - Code enthält falschen IL-Code.
  - Gültiger Code
    - Code kann nicht typsichere Anweisungen (kann durch Zeigerarithmetik entstehen) enthalten.
  - Typsicherer Code
    - Objekte halten sich an Schnittstellen.
  - Verifizierbarer Code
    - Typsicherheit kann bewiesen werden.
    - Nicht verifizierbarer Code kann aber typsicher sein.
- Manche Compiler garantieren die Erzeugung von typsicherem Code (C#, nicht C++).

## Aufgaben des VES (3)

- JIT: Übersetzung von IL- in nativen Code.
- Code-Manager: Ausführung des nativen Codes
  - Garbage Collection von „Managed Types“,
  - Ausnahmebehandlung,
  - Security:
    - Code-Access Security: Berechtigungen sind von der „Herkunft“ des Codes abhängig.
    - Große Veränderungen in .NET 4.0
  - Debugging und Profiling,
  - Platform Invoke (P/Invoke):
    - Aufruf von Win32 API-Funktionen

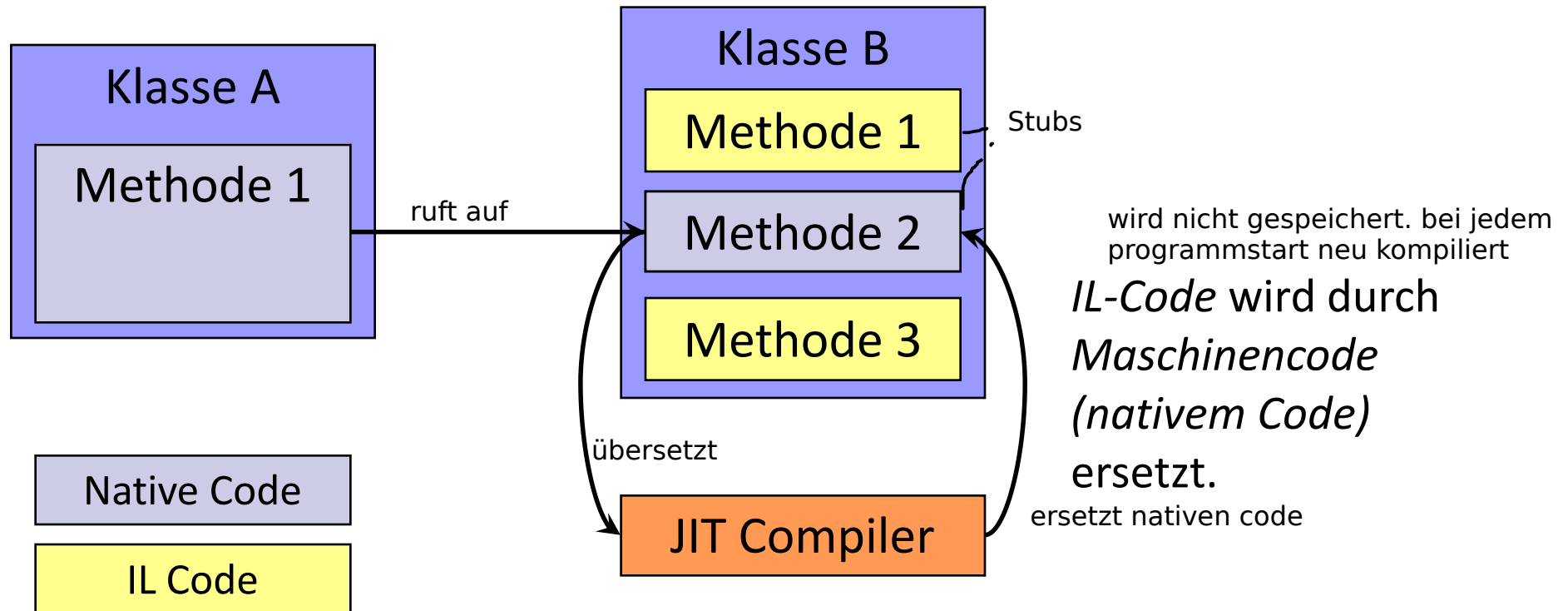
# Übersetzung und Ausführung





# Just in Time Compiler (JIT)

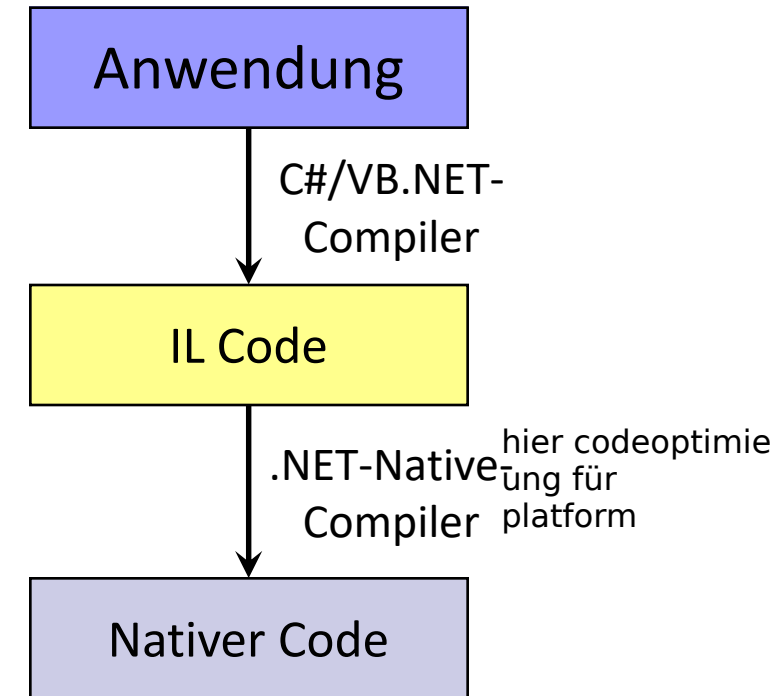
- IL-Code wird immer kompiliert.
- Code-Generierung bei erstem Methodenaufruf.



man kann auch alles vorübersetzen. zB systemkomponenten, zentrale type

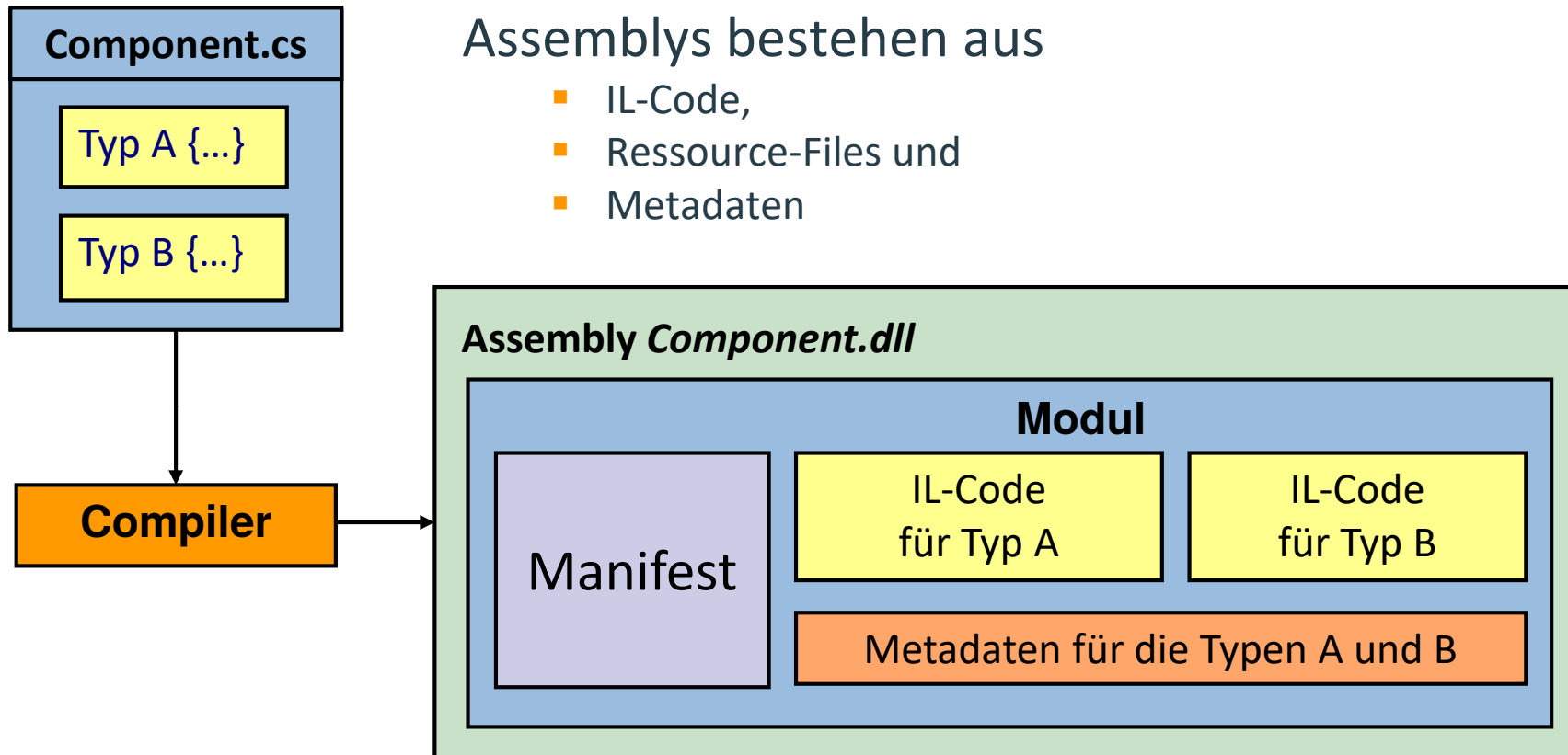
# .NET Native (.NET Core) andere Methode als vorhin

- liegt in binärer form vor. Für UWP Apps
- Der JIT-Compiler übersetzt IL-Code zur Laufzeit des Programms → Just In Time.
- Mit .NET Native kann IL-Code bereits zur Übersetzungszeit des Programms in nativen Code kompiliert werden.
- geht nur eingeschränkt .NET Native nutzt das Compiler-Backend von C++.
- Vorteile
  - Schnellere Ausführungszeiten durch kürzeren Programmstart
  - Schnellerer Programmstart (weniger code muss geladen werden)
  - Geringerer Hauptspeicherbedarf
  - Kleinere Deployment-Pakete (kompakter)
- Produktiv wird .NET Native dzt. nur für Universal-  
Windows-Plattform-Apps (Windows 10) genutzt. besonders für heterogene Plattformen (z:B versch. .NET methoden)  
Am Zielsystem nur minimale runtime. sonst würde auf bestimmten platformen programm nicht laufen.



# Assembllys

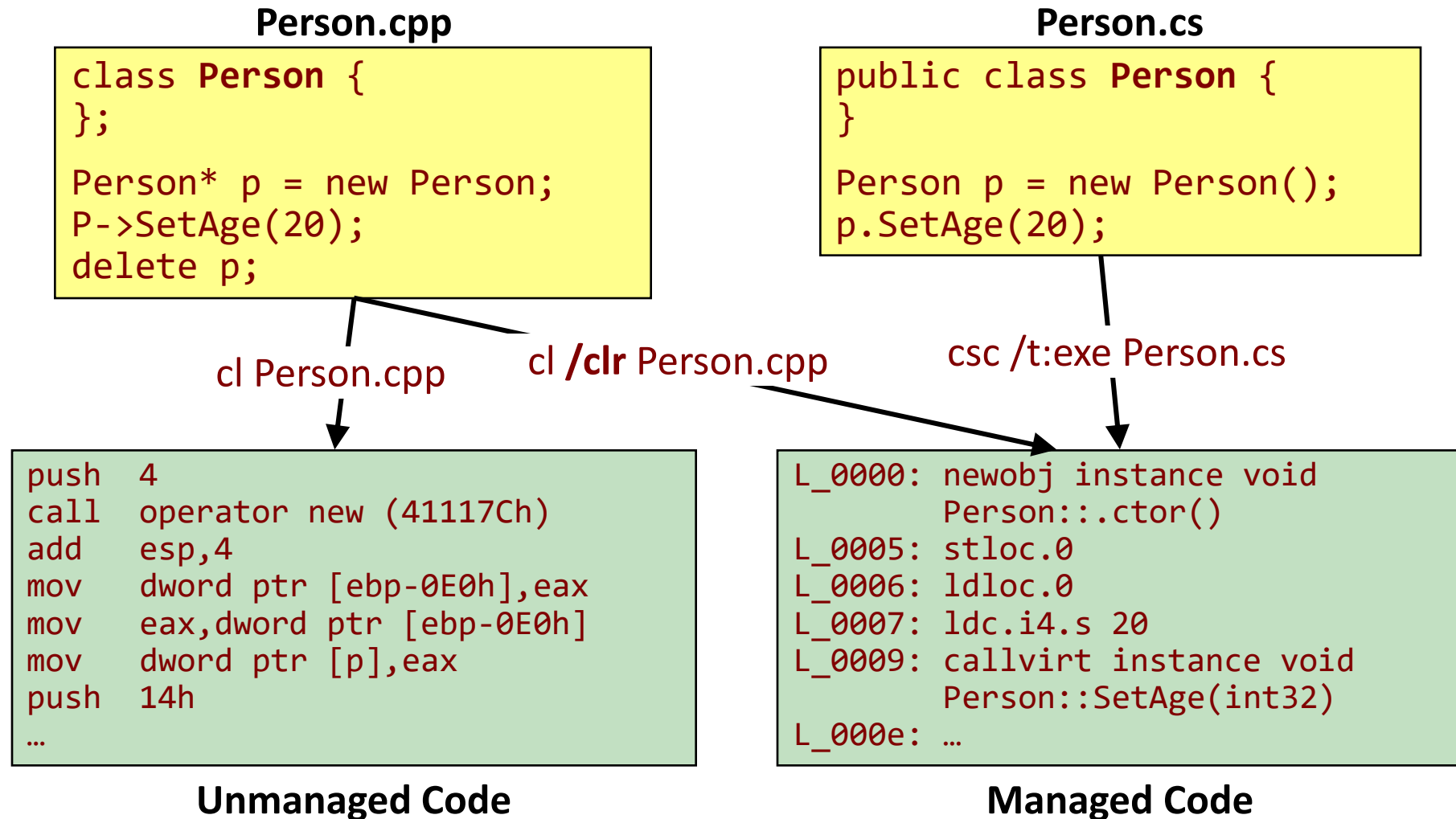
link zum nächsten foliensatz  
Assemblies sind exe und dlls.



# Managed/Unmanaged Code/Types

- *Managed Code* wird von der CLR ausgeführt.
- *Unmanaged Code* wird direkt vom Prozessor ausgeführt.
- Alle .NET-Sprachen, außer C++, werden in *Managed Code* übersetzt.
- C++-Code kann entweder in nativen oder *Managed Code* übersetzt werden.
- *Managed Code* darf nicht mit *Managed Types* verwechselt werden. Nur *Managed Types* werden vom Garbage Collector automatisch freigegeben.
- Nur *Managed Types* können über Assembly-Grenzen hinweg verwendet werden.

# Managed/Unmanaged Code



# Managed/Unmanaged Types

