Vorlesung
Software aus Komponenten

3. Komponentenmodelle

Prof. Dr. Hans-Gert Gräbe
Wintersemester 2012/13

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

> "... komplette Neudefinition der Art, wie Microsoft in Zukunft Geschäfte machen will ... und wie Software entwickelt werden soll."
>
> **Westphal, 2002**

- Plattform soll bisherige Vorgehensweisen der Windows-Programmierung ersetzen, flexibel auf Betriebssystem- und Basisfunktionen zugreifen und Austausch zwischen Programmen unterstützen.
- Ausgerichtet auf den Einsatz auf verschiedenen Hardware-Plattformen bis hin zu Handys und PDAs. Java-Idee ohne Beschränkung auf Java als Programmiersprache
- Ziele
  - Sicherheit
  - Plattformunabhängigkeit
  - Interoperabilität
  - Homogenität

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Vorgeschichte:

- Rechtsstreit zwischen Sun und Microsoft um Java
  - Microsoft erweitert Java nach eigenen Vorstellungen und Bedürfnissen und gefährdet damit die Java-Kompatibilität
  - Microsoft-Implementierungen J++ und J#
- Weitere Probleme:
  - Auch die für Windowsprogrammierung meist verwendeten Sprachen Visual Basic, C++ und J++ waren untereinander nicht kompatibel
  - String-Datentypen waren sogar nicht binär kompatibel - .NET ist konsequent Unicode basiert
  - kein einheitliches Modell der Speicherverwaltung

- 1996: erste Arbeiten an .NET
- 2000: .NET-Framework 1.0 Beta
- August 2000 – C# und die CLI werden von MS, HP und Intel zur Standardisierung bei der ECMA eingereicht
  - ECMA – European Computer Manufacturers Association
- Dezember 2001 – Fertigstellung des ersten Standards und Weitergabe an die ISO
- Januar 2002: .NET Framework 1.0 und Visual Studio .NET
- April 2003 – .NET Framework 1.1 und Verabschiedung der ISO-Standards ISO/IEC 23270 (C#), ISO/IEC 23271 (CLI)
- 2004: Marktanteil steht noch immer in keinem Verhältnis zur Aufmerksamkeit, die .NET in den Medien findet

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

ECMA-Standardisierung erlaubt Implementierung des Standards auch auf anderen Plattformen.

Versionen jenseits von Windows:

- Microsoft selbst stellte 2002 mit der Shared Source CLI Versionen für Mac OS und Linux bereit. Diese Aktivitäten wurden später wieder aufgegeben.
- Mit dem Mono-Projekt entstehen ab 2004 erste freie Implementierungen von .NET-Konzepten, aktuell zwei Linien:
  - Mono-Projekt von Ximian, dotGNU-Projekt arbeitet an einer Laufzeitumgebung Portable.NET

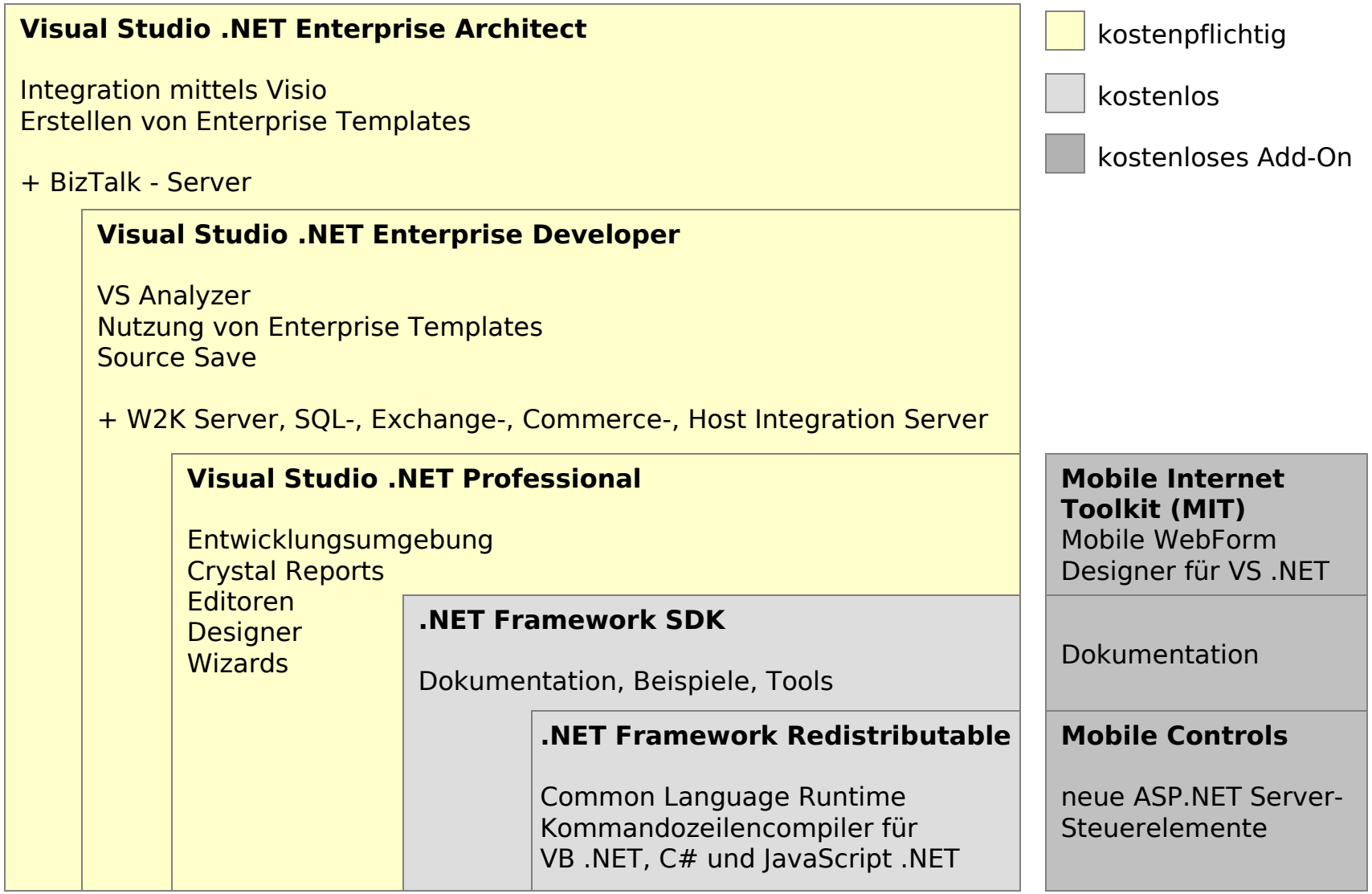
  Bleiben hinter der Leistungsfähigkeit der Windows-Versionen zurück.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Ende 2005: Visual Studio 2005, .NET Framework 2.0
  - Zusammen mit MS SQL Server 2005, MS BizTalk Server 2006
  - Viele Anwendungen verwenden noch starken Durchgriff auf assemblerspezifischen Windows-Code jenseits der IL, damit nur eingeschränkte Plattformunabhängigkeit und Interoperabilität
- Ende 2006: .NET 3.0, später integraler Bestandteil von Windows Vista und Windows Server 2008, mit tiefgreifenden auch konzeptionellen Änderungen an der Architektur
  - WPF – Windows Presentation Foundation: Beschreibungssprache XAML zur Darstellung von Objekten auf dem Bildschirm.
  - WCF – Windows Communication Foundation: Dienstorientierte Kommunikationsplattform für lose gekoppelte verteilte Anwendungen.
  - WWF – Windows Workflow Foundation: Infrastruktur für die einfachere Entwicklung von Workflow-Anwendungen.
  - Windows CardSpace: Identitätsmanagement-Infrastruktur für verteilte Anwendungen.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

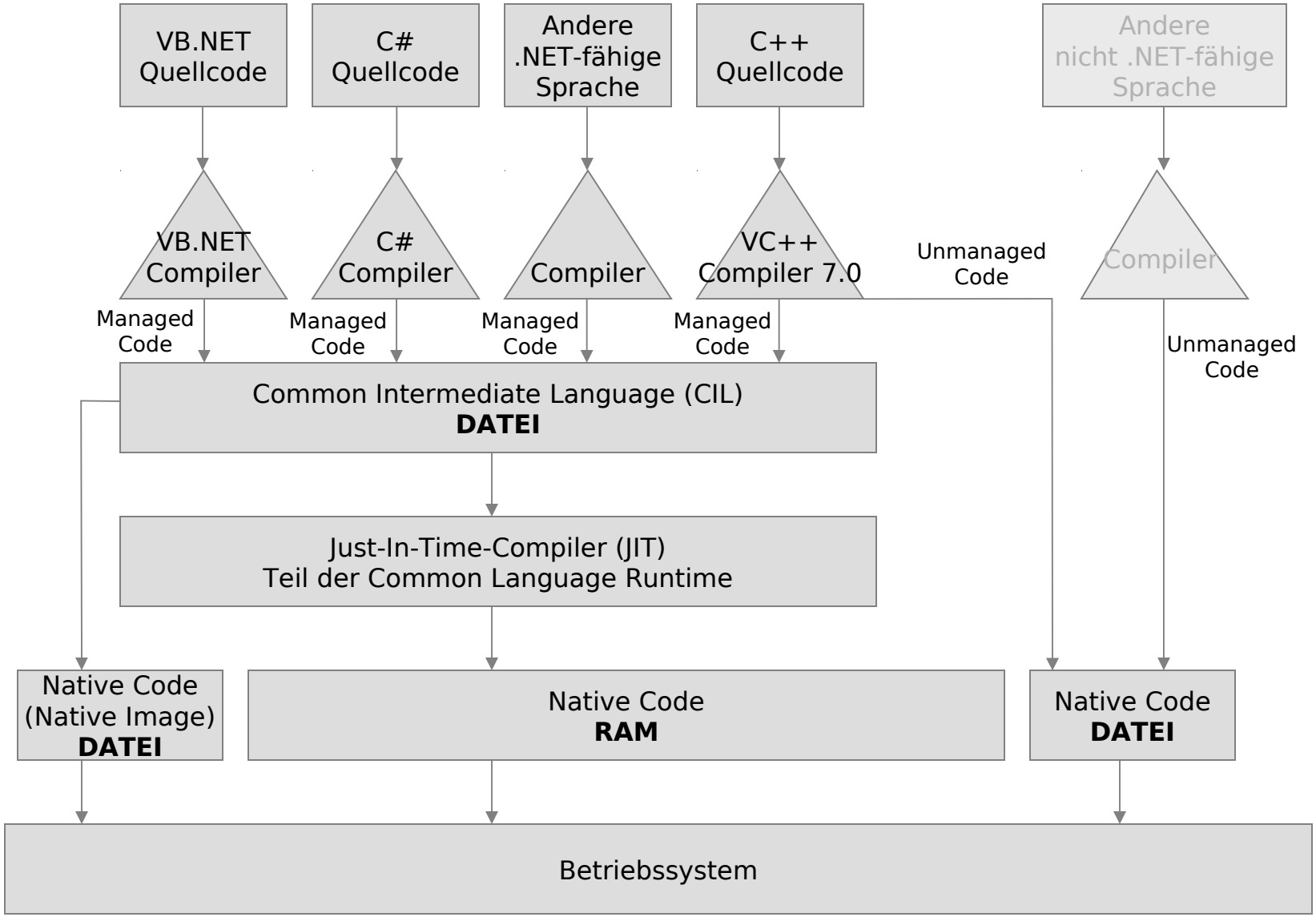
- Ende 2007: Visual Studio 2008 und .NET Framework 3.5
  - Base Class Library (BCL) wird in Framework Class Library (FCL) umbenannt
    - Umfasst fast 12.000 Klassen in 300 Namensräumen
  - Teilweise Freigabe des Quellcodes der Base Class Library unter der restriktiven MS Reference Source License
  - Damit wird die Diskussion um Urheberrechtsverletzungen durch das Mono-Projekt weiter angeheizt
  - Deal zwischen Novell (SuSe-Linux und Mono-Projekt) und Microsoft
- April 2010: .NET 4.0 und Visual Studio 2010
  - Stärkere Ausrichtung auf Mehrkern-Systeme, verteilte Architekturen und Thread-Parallelität
  - Neues Programmiermodell für Multithreading und asynchronen Code
- August 2012: .NET 4.5


Visual Studio .NET Enterprise Architect
Integration mittels Visio
Erstellen von Enterprise Templates
+ BizTalk - Server
Visual Studio .NET Enterprise Developer
VS Analyzer
Nutzung von Enterprise Templates
Source Save
+ W2K Server, SQL-, Exchange-, Commerce-, Host Integration Server
Visual Studio .NET Professional
Entwicklungsumgebung
Crystal Reports
Editoren
Designer
Wizards
.NET Framework SDK
Dokumentation, Beispiele, Tools
.NET Framework Redistributable
Common Language Runtime
Kommandozeilencompiler für
VB .NET, C# und JavaScript .NET
kostenpflichtig
kostenlos
kostenloses Add-On
Mobile Internet Toolkit (MIT)
Mobile WebForm Designer für VS .NET
Dokumentation
Mobile Controls
neue ASP.NET Server-Steuerelemente

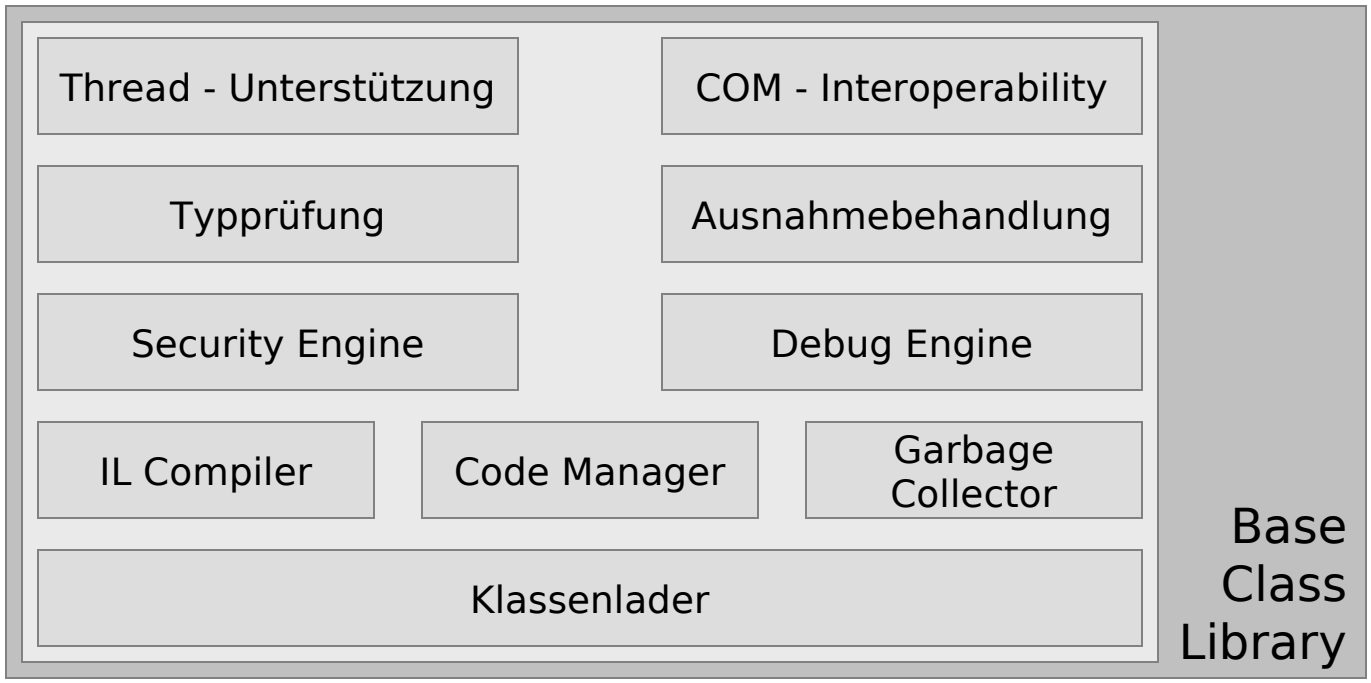
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Das .NET-Konzept

- CLI – **Common Language Infrastructure**
  - Basis zur Ausführung von Programmen, die in unterschiedlichen Programmiersprachen erstellt wurden
  - Zugriff auf eine **virtuelle Maschine** und eine gemeinsame Klassenbibliothek – die **Base Class Library**
- CIL - **Common Intermediate Language**
  - Hochsprachenunabhängige Zwischensprache
- CLR – **Common Language Runtime**
  - Laufzeitumgebung für CIL-Zwischencode
- CTS – **Common Type System**
  - Sicherung der Kompatibilität der Ressourcenzugriffe über einen sprachübergreifenden Standard von OO-Datentypen
  - .NET wurde von Anfang an für den Betrieb mit mehreren Programmiersprachen entwickelt
- Assemblies – **Packungsformat** für Komponenten


VB.NET Quellcode
C# Quellcode
Andere .NET-fähige Sprache
C++ Quellcode
Andere nicht .NET-fähige Sprache
VB.NET Compiler
C# Compiler
Compiler
VC++ Compiler 7.0
Compiler
Unmanaged Code
Managed Code
Managed Code
Managed Code
Managed Code
Unmanaged Code
Common Intermediate Language (CIL)
DATEI
Just-In-Time-Compiler (JIT)
Teil der Common Language Runtime
Native Code (Native Image)
DATEI
Native Code
RAM
Native Code
DATEI
Betriebssystem

- CLR übersetzt Zwischensprachencode (CIL) in Maschinencode
- Speicherverwaltung
- Verwaltung von Prozessen und Threads
- Durchsetzung von Sicherheitsmechanismen
- Laden von Komponenten
- **Alle** .NET-Sprachen setzen auf die CLR als Runtime auf

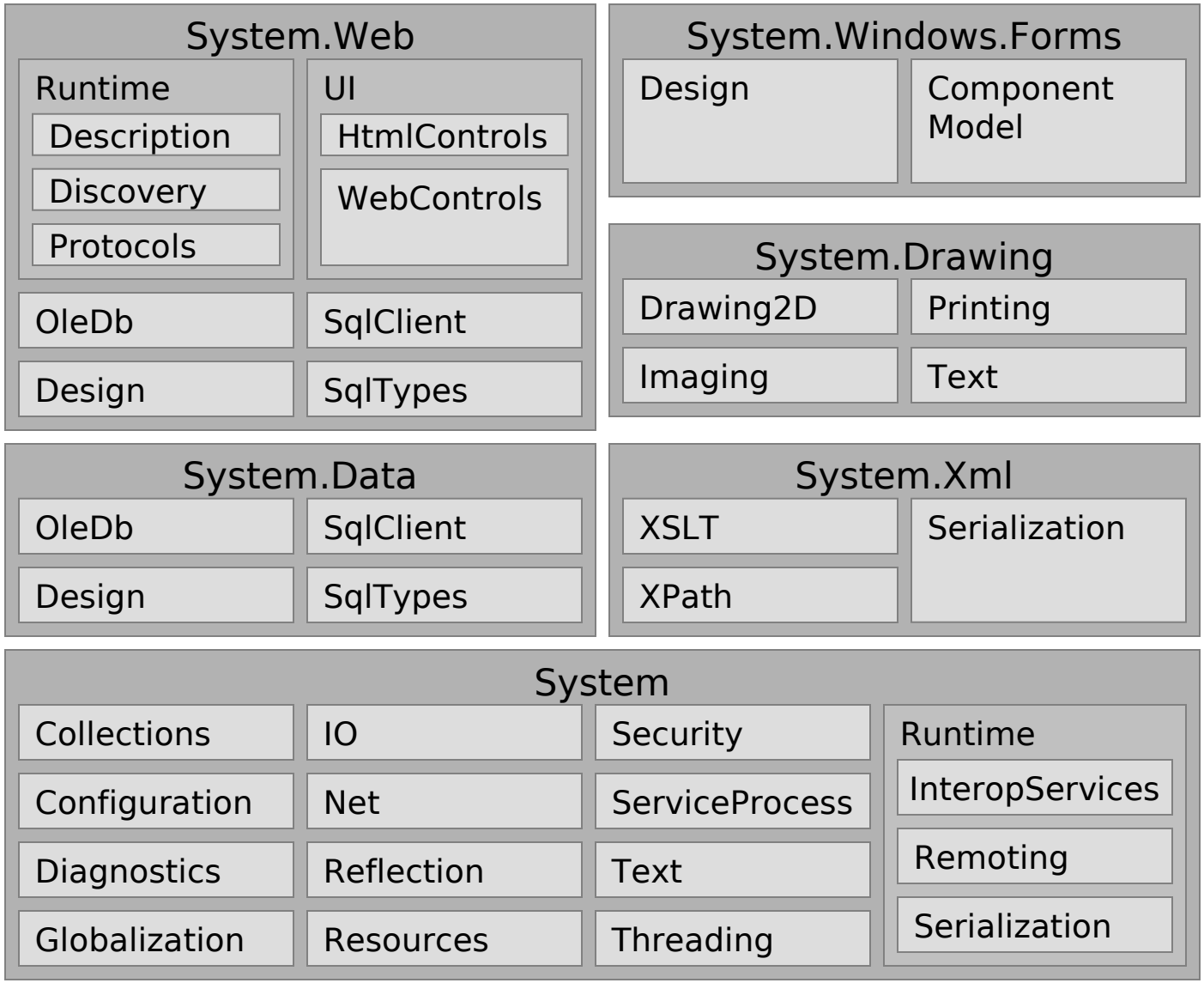


UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Konsistentes Programmiermodell
  - alle Anwendungsdienste als objektorientiertes Programmiermodell
- Vereinfachtes Programmiermodell
  - keine Registrierung und eigenständige Registry-Verwaltung
- Stabile Installationen
  - isolierte Anwendungskomponenten
  - keine „DLL-Hölle“ mehr
  - Versionierung von Komponenten
- Vereinfachte Installationen
  - Anwendungsdateien können einfach in Zielverzeichnis kopiert werden
  - keine Registry-Einträge nötig
- Viele verfügbare Plattformen
  - Compiler generiert IL-Code
  - Ausführbar auf Maschinen, die über ECMA-kompatible Versionen der CLR verfügen
- Integration verschiedener Programmiersprachen
  - Typen, die in unterschiedlichen Sprachen geschrieben wurden
  - Common Type System

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Einfacheres Wiederverwenden von Code
  - durch oben beschriebene Techniken
- Automatische Speicherverwaltung
  - Garbage Collection
- Typsicherheit
  - Zugriff auf Objekte immer auf kompatible Weise
  - Code springt nur an bekannte Stellen (Eintrittspunkt von Methoden)
  - keine Pufferüberläufe
- Komfortables Debuggen
  - Debuggen von Anwendungen unterschiedlicher Sprachen
- Konsistente Fehlerverarbeitung
  - **Alle** Fehler werden über Ausnahmen gemeldet
- Sicherheit
  - basierend auf Herkunft des Codes (code based security) oder der Daten (role based security)
- Interoperabilität
  - Zugriff an der CIL vorbei auf Komponenten möglich, die den älteren COM-Standard implementieren



- Schnittstelle zum Betriebssytem
- komplett Objektorientiert
- Allen .NET Sprachen stehen dieselben Dienste zur Verfügung
- Zugriff auf Dateisystem, Fensteranzeige, Druckfunktionen, Remoting, Grafik, Datenbankzugriff

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- CIL ist eine Art "objektorientierter Maschinencode"
- arbeitet Stack-orientiert, keine Register
- unabhängig von CPU
- Verifizierung des Codes durch die CLR bei der Übersetzung in nativen Code
  - nur Speicheradressen lesen, in die vorher Daten geschrieben wurden
  - Methoden mit der korrekten Anzahl von Argumenten aufrufen
  - jedes Argument hat richtigen Typ
- wird zur Laufzeit vom Just-In-Time-Compiler kompiliert
- Caching von bereits übersetzten Typen
- Optimierung anhand ausführender Architektur

```
.method private hidebysig static void Main() cil managed {
        .entrypoint
        .maxstack 3
        .locals ([0] int32 v, [1] object o)
        IL_0000:  ldc.i4.5
        IL_0001:  stloc.0
        IL_0002:  ldloc.0
        IL_0003:  box        [mscorlib]System.Int32
        ...
```

Beispiel für IL-Code

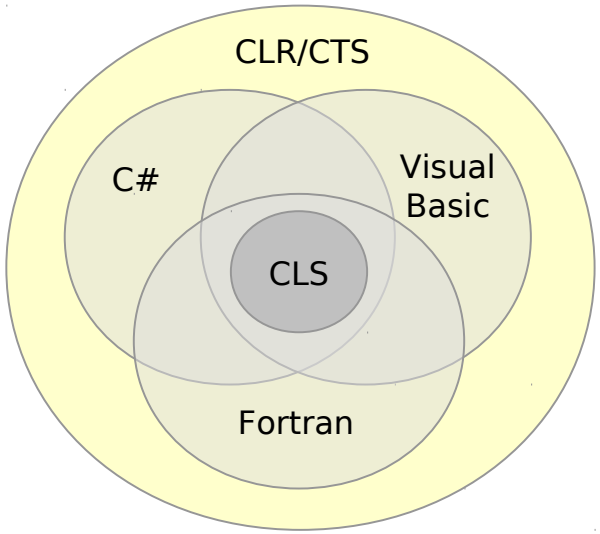
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Common Language Specification

kleinster gemeinsamer Nenner der .NET-Sprachen

- standardisierte Typen
- selbstbeschreibende Typinformationen (Metadaten)
- gemeinsame Ausführungsumgebung

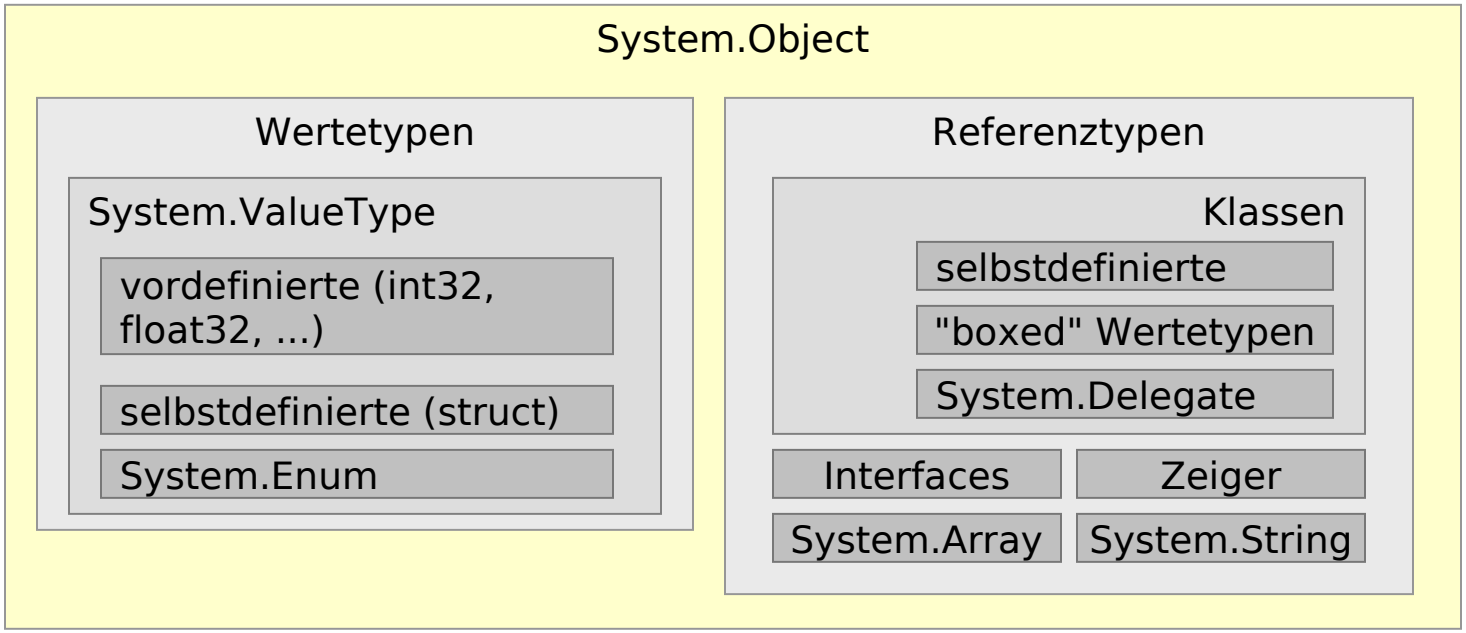
```
using System;
[assembly:CLSCompliant(true)]
// Compiler soll CLS-Kompatibilität prüfen

// Fehler, weil Klasse öffentlich ist
public class App {
    // Fehler, weil UInt32 nicht CLS-Kompatibel
    public UInt32 Abc() { return 0; }
    // Fehler, weil keine Unterscheidung zwischen
    // Groß- und Kleinschreibung in CLS
    public void abc() {}
    //Kein Fehler, da Methode privat ist
    private UInt32 ABC() { return 0;}
```



Vollständige Liste der CLS-Regeln im Abschnitt „Cross-Language Interoperability“ in der Dokumentation des .NET Framework SDK

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Common Type System



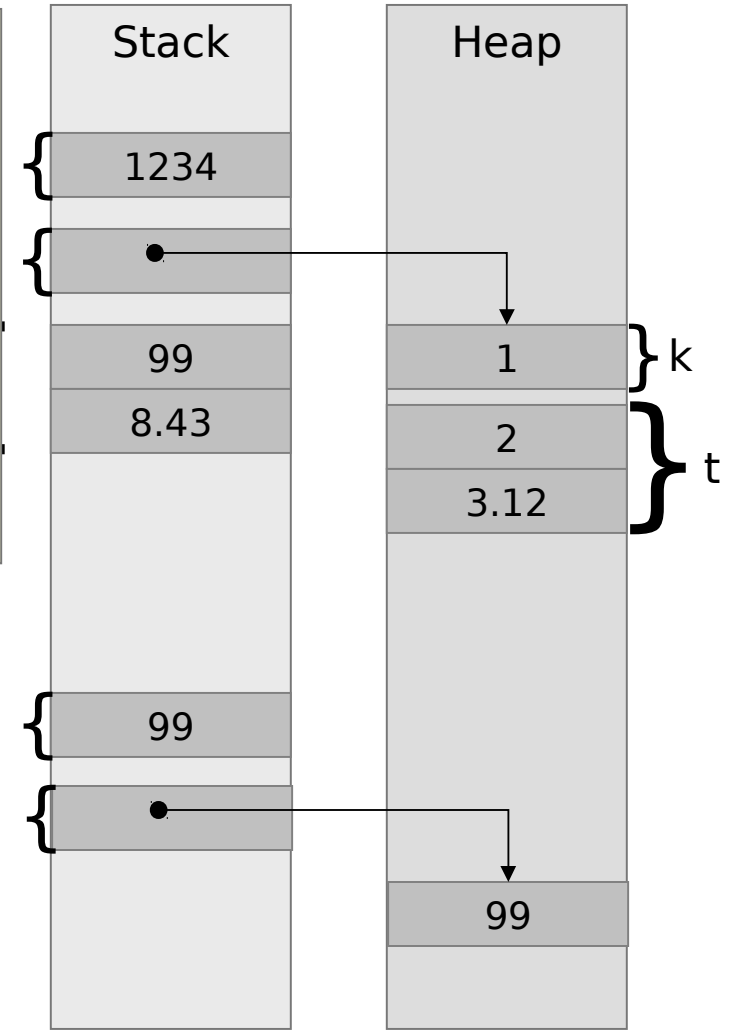
- Wertetypen enthalten Werte (liegen auf dem Stack)
- Referenztypen zeigen auf Werte (Werte liegen auf dem Heap)
- Wertetypen können ausdrücklich als Objekte behandelt (und damit auf dem Heap abgelegt) werden: Boxing

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Verweis- und Referenztypen

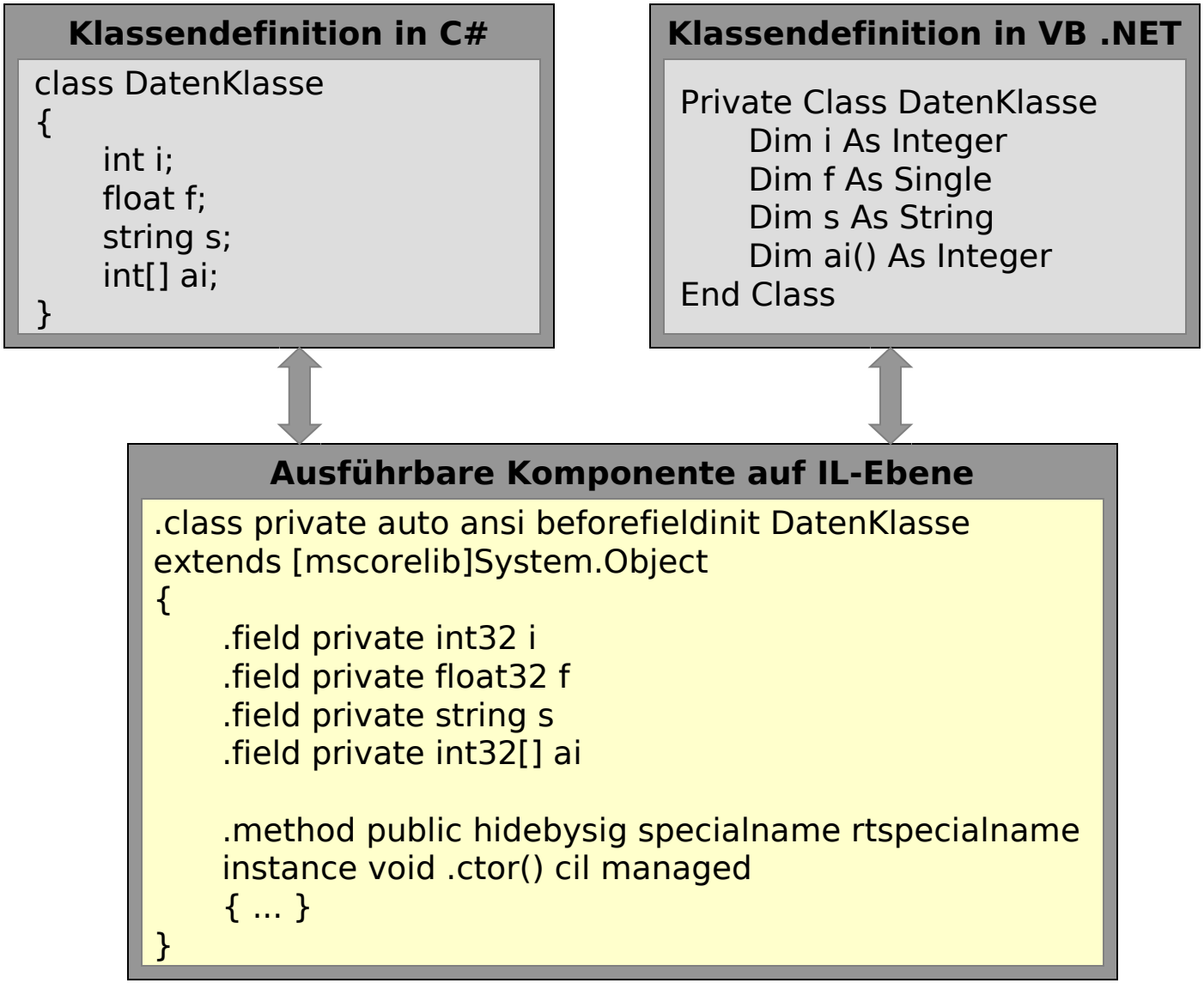
```
struct MyStruct {
      int i;
      float f;
}
class MyClass {
      int k;
      MyStruct t
}
```

```
int j;
j = 1234;

MyClass c = new MyClass();
c.k = 1; c.t.i = 2; c.t.f = 3.12;

MyStruct s;
s.i = 99; s.f = 8.43
```



Boxing

```
int i;
i = 99;

object o;
o = i;
```


Klassendefinition in C#
class DatenKlasse
{
int i;
float f;
string s;
int[] ai;
}
Klassendefinition in VB .NET
Private Class DatenKlasse
Dim i As Integer
Dim f As Single
Dim s As String
Dim ai() As Integer
End Class
Ausführbare Komponente auf IL-Ebene
.class private auto ansi beforefieldinit DatenKlasse
extends [mscorelib]System.Object
{
.field private int32 i
.field private float32 f
.field private string s
.field private int32[] ai
.method public hidebysig specialname rtspecialname
instance void .ctor() cil managed
{ ... }
}

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Typsicherheit
  - CLR weiß zur Laufzeit **immer** um den Typen eines Objektes
  - Objekt kann seinen Typ nicht manipulieren
- Konvertierung (Type Casting)
  - Objekt kann in einen seiner Basistypen konvertiert werden (z.B. Int32 -> Object) – implizite Konvertierung
  - wenn ein Objekt in einen abgeleiteten Typen umgewandelt werden soll, muss explizit konvertiert werden

```
class SHK : Student { ... }
class App {
   public static void Main() {
      SHK x = new SHK();
      EvaluateStudent(x); // Ok, da SHK vom Typ Student abgeleitet wurde

      DateTime newYear = new DateTime(2001, 1, 1);
      EvaluateStudent(newYear);
      // Ausnahme in EvaluateStudent, da DateTime nicht von Student abgeleitet wurde
   }
   public void EvaluateStudent(Object o) {
      Student s = (Student) o;
      // Compiler weiß nicht, ob Konvertierung erfolgen kann, erst CLR zur Laufzeit
      ...
   }
```