

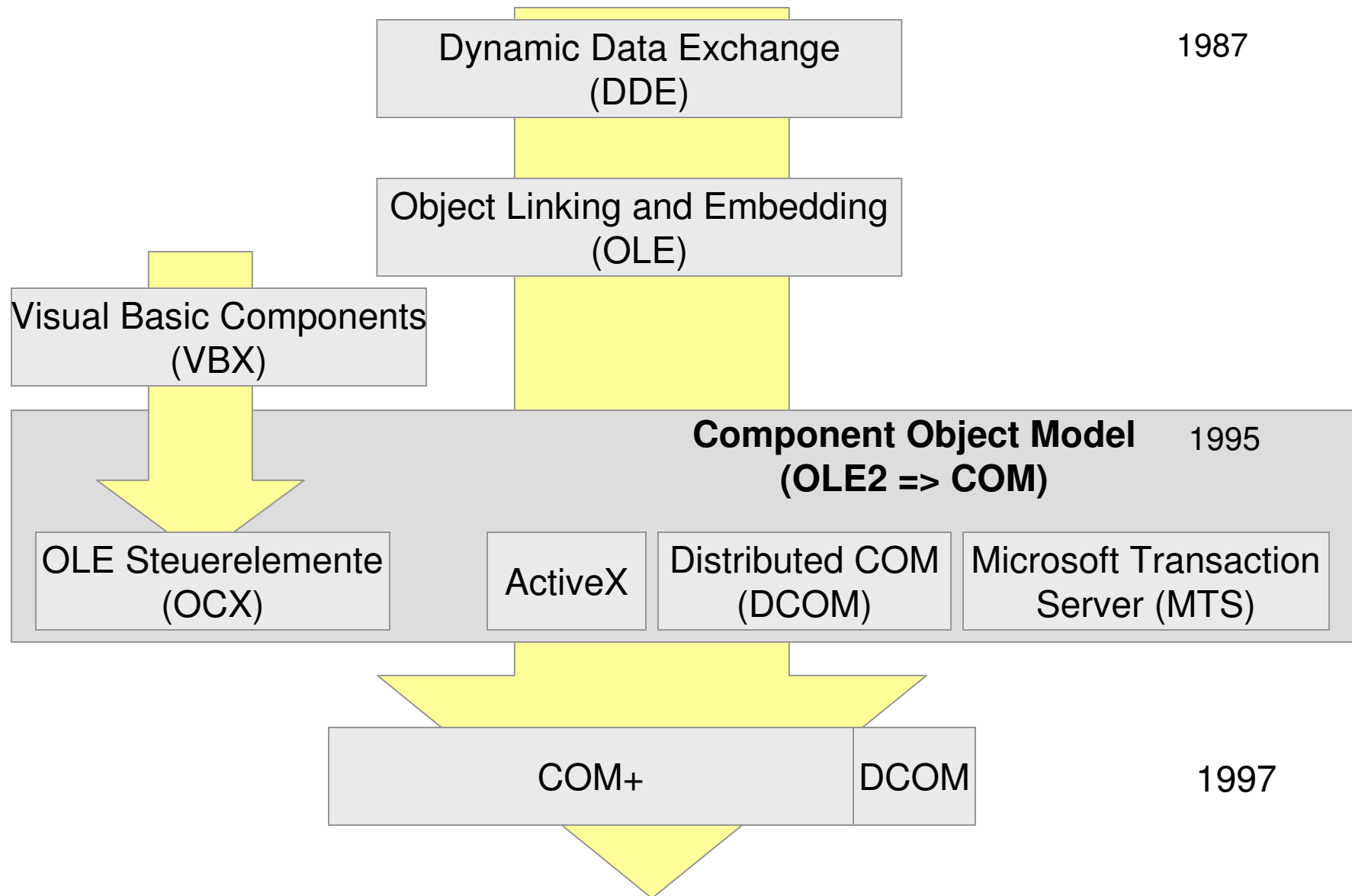
Vorlesung Software aus Komponenten

3. Komponenten-Modelle

Prof. Dr. Hans-Gert Gräbe
Wintersemester 2005/06

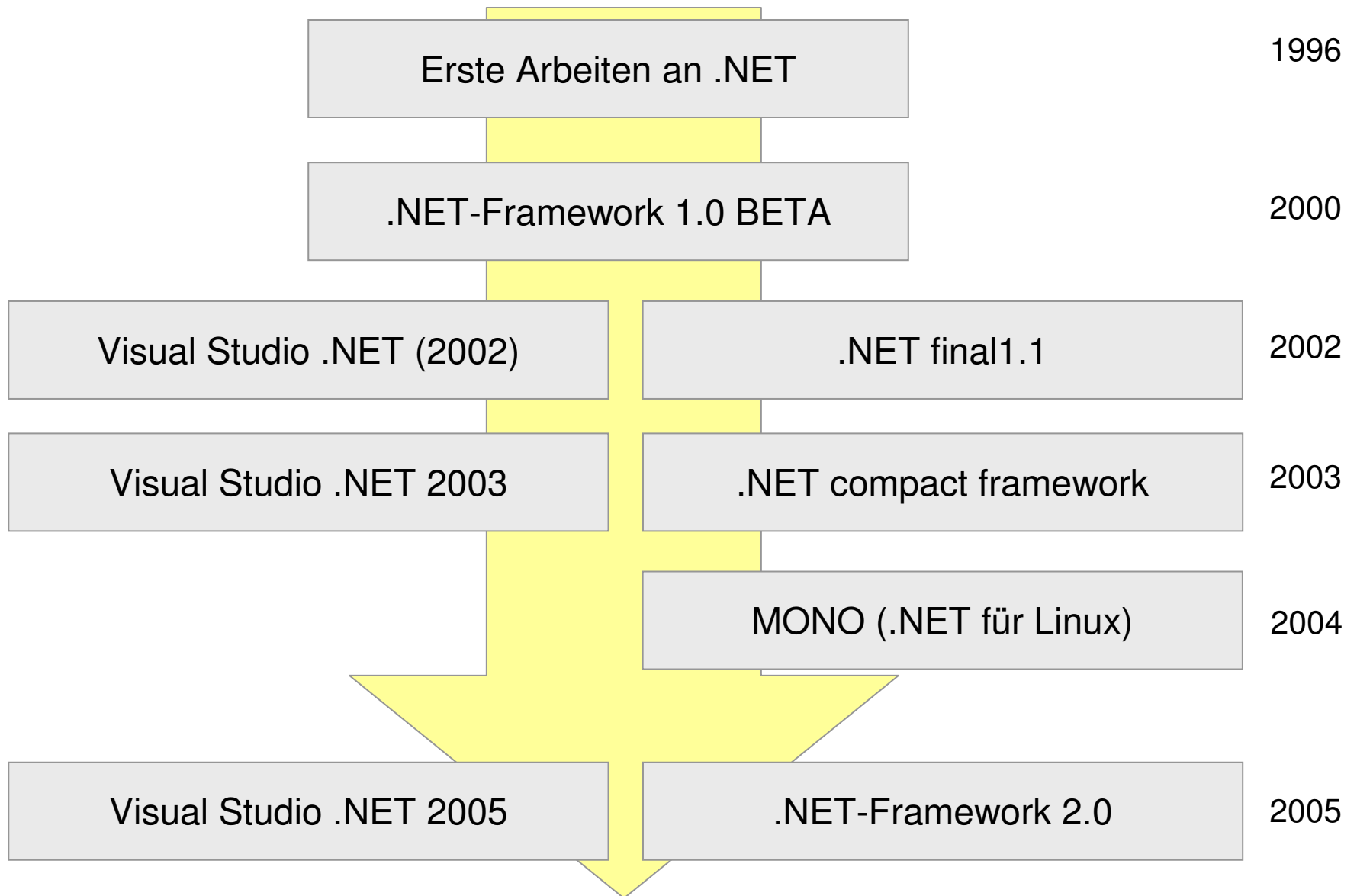
3.4 Microsoft COM und .NET

Geschichtliche Einordnung



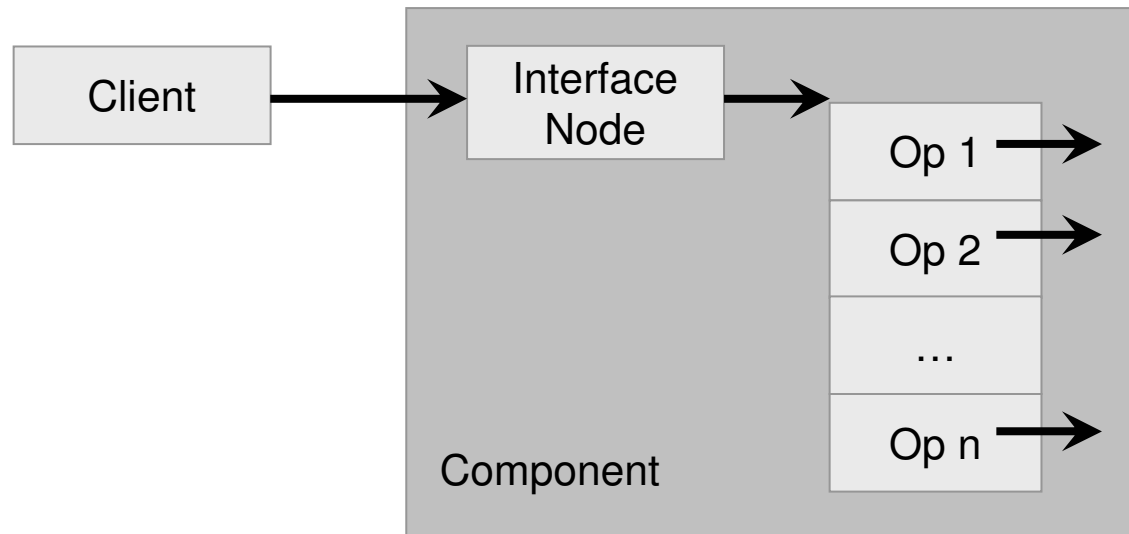
3.4 Microsoft COM und .NET

Geschichtliche Einordnung

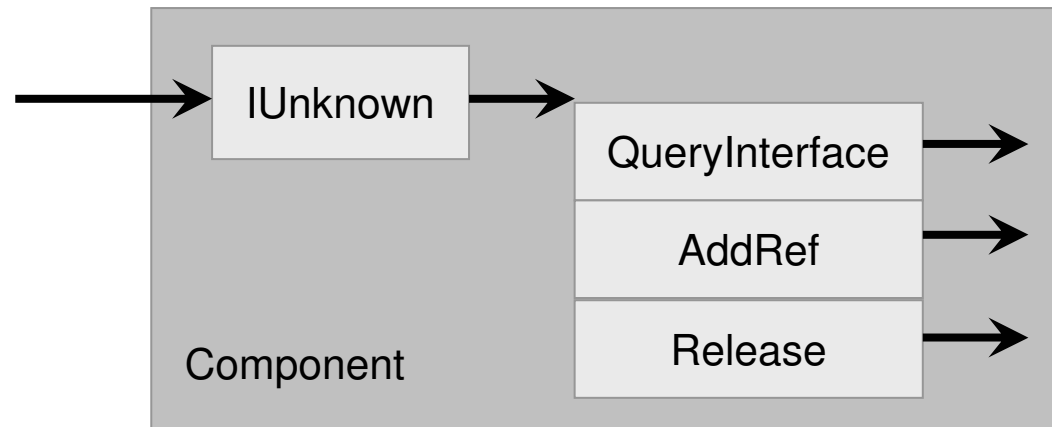


3.4 Microsoft COM und .NET

COM – Die Schnittstelle



- COM ist binärer Standard
- Schnittstelle ist zentraler Punkt von COM
- Schnittstellenvererbung
- Interface Node
 - Methoden eines Objekts => **"this"**-Parameter wird an jede Methode weitergereicht
 - Komponente kann mehrere Schnittstellen implementieren



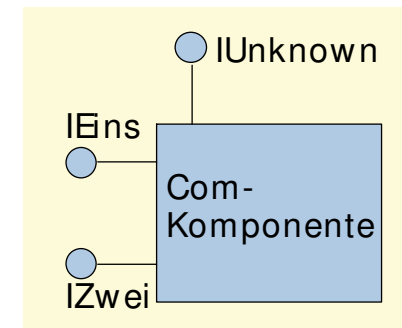
- Schnittstelle IUnknown **muss** von jeder Komponente implementiert werden
 - jede andere Schnittstelle erbt von IUnknown
- QueryInterface
 - erste Methode jedes COM-Objektes
 - gibt Zeiger auf angefordertes Interface zurück
 - benutzt Interface Identifier (IID)
- AddRef, Release
 - Verwaltung des Lebenszyklus
 - Referenzzähler

```
[ uuid(12345678-1234-1234-1234-123456789ABC) ]
```

```
interface IUnknown {
```

```
    HRESULT QueryInterface ([in] const IID iid, [out, iid_is(iid)] IUnknown iid);  
    unsigned long AddRef();  
    unsigned long Release();
```

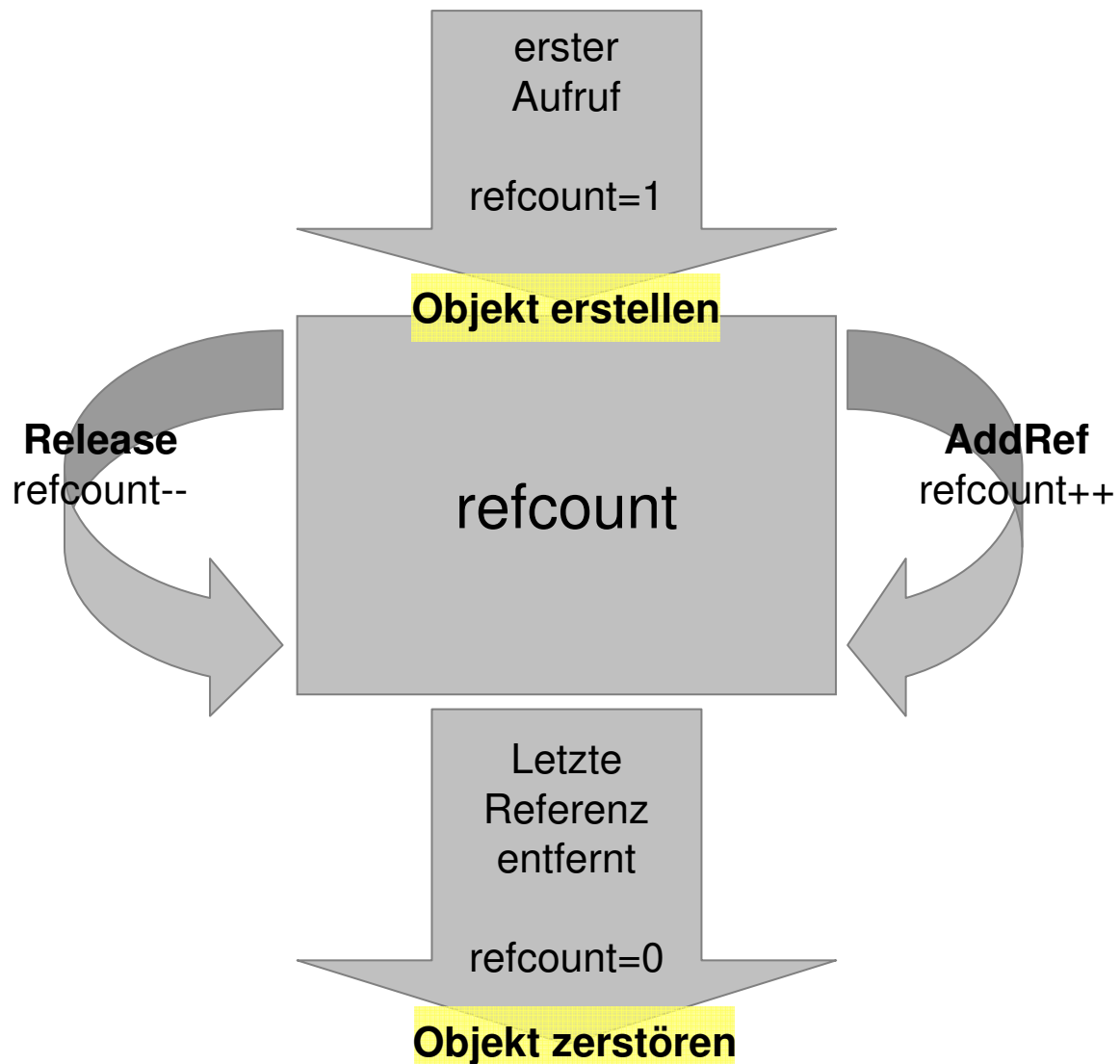
```
}
```



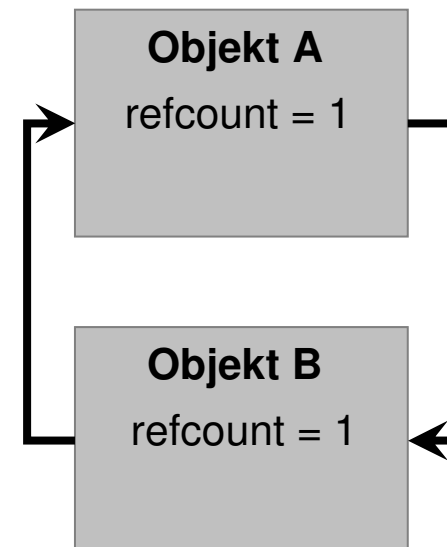
- IID ist eindeutiger Bezeichner/Name
- "sprechender" Name als Alias, generell beginnend mit einem I
- IID ist Global Unique Identifier (GUID)
 - Global eindeutige Nummer, die nach einem speziellen Algorithmus erzeugt wird und in Zeit und Raum eindeutig ist
 - Im Zusammenhang mit Schnittstellen werden GUIDs auch als Interface Identifier (IIDs) bezeichnet

3.4 Microsoft COM und .NET

Lebenszyklus von COM-Objekten

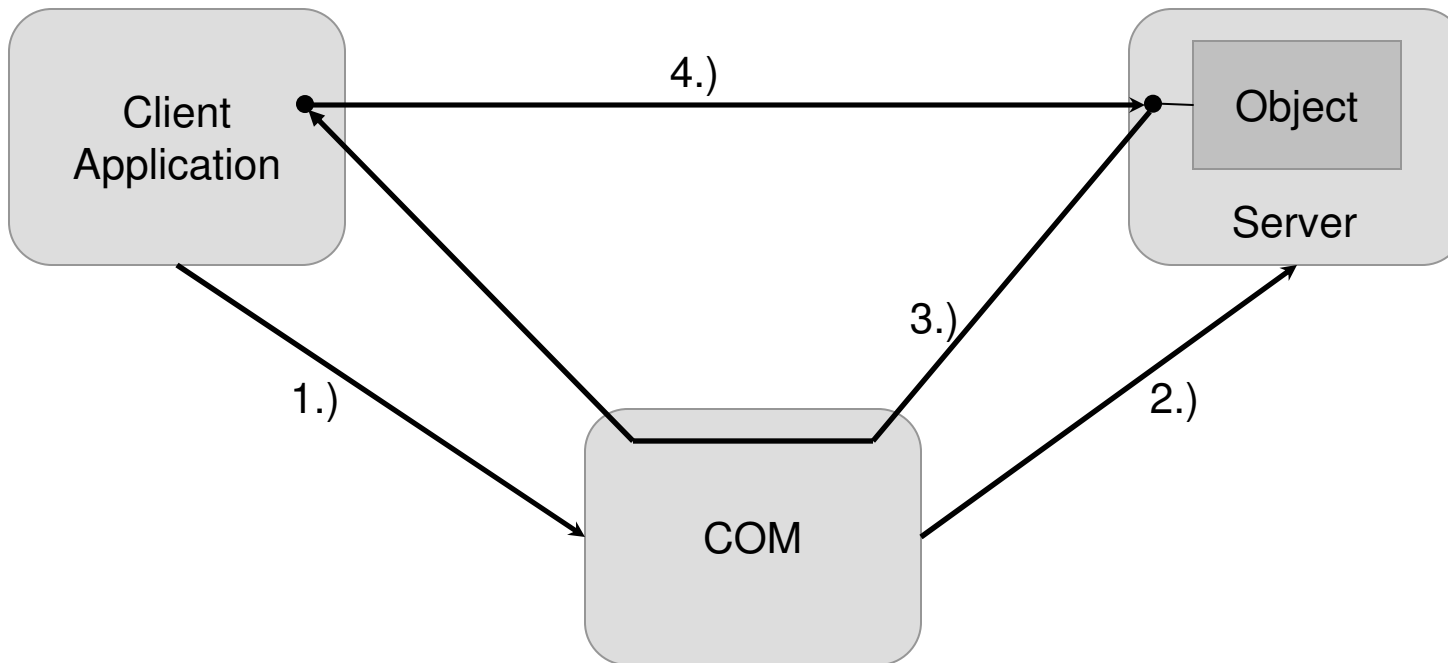


Problem zyklischer Referenzen



3.4 Microsoft COM und .NET

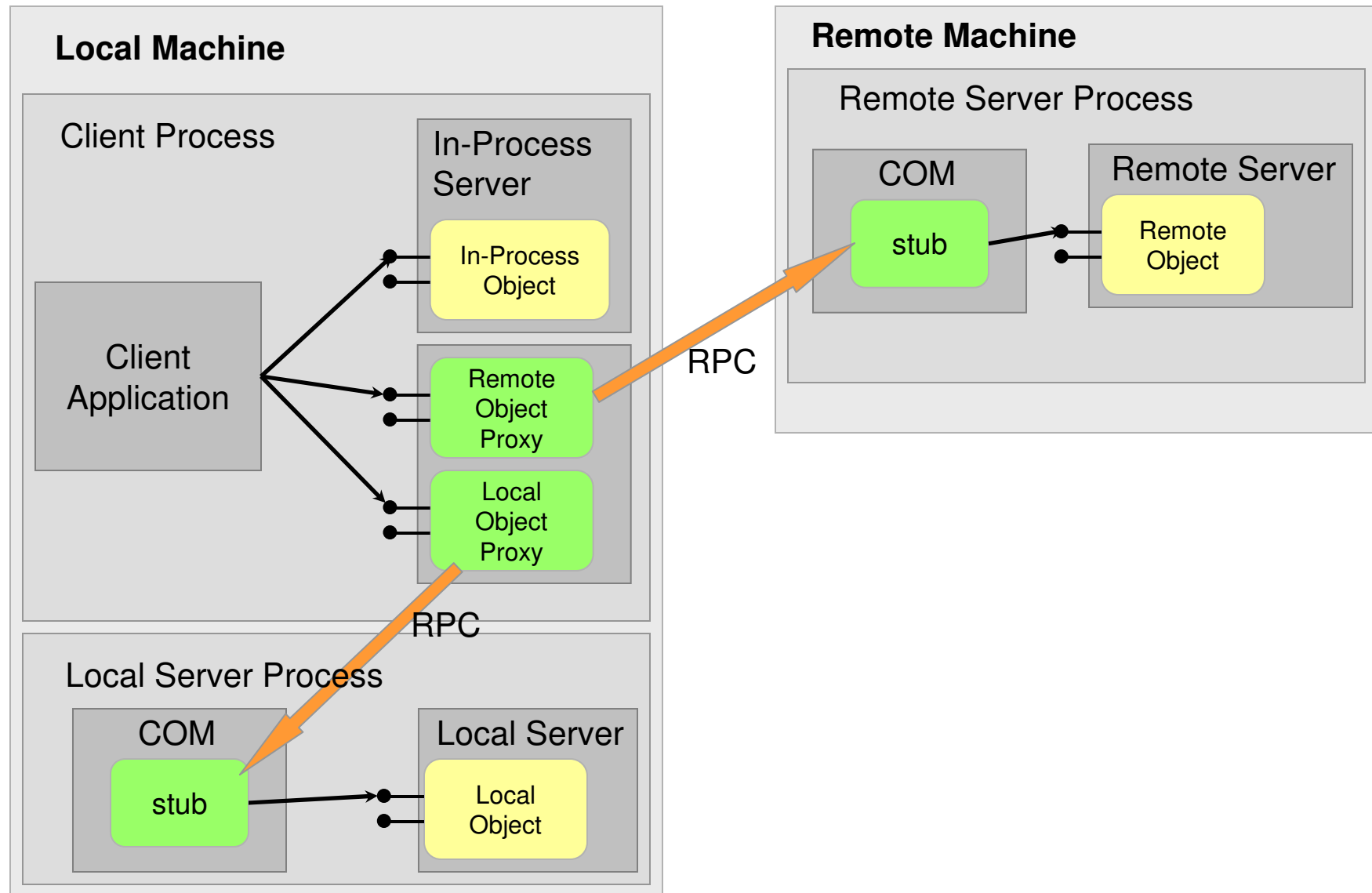
Erstellen eines COM-Objektes



1. Aufruf der Funktion "CreateObject"
2. COM lokalisiert/erzeugt den Server
3. Serverprozess erzeugt das Objekt und gibt einen Schnittstellenzeiger zurück
4. Client kommuniziert über den Schnittstellenzeiger mit dem Serverobjekt

3.4 Microsoft COM und .NET

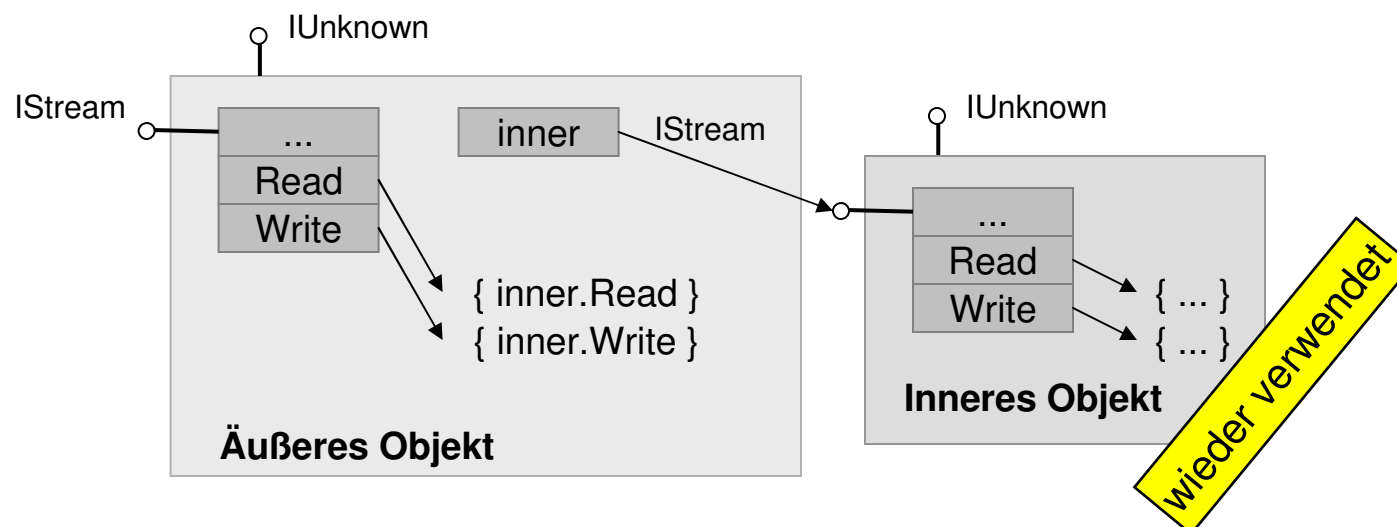
Kommunikation zwischen COM-Objekten



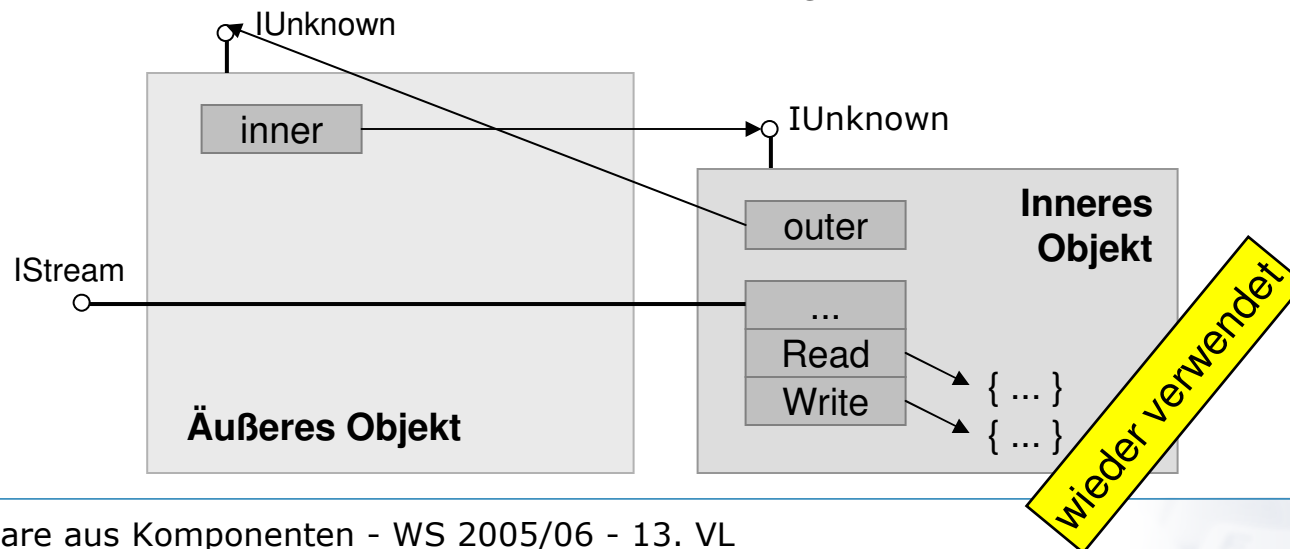
- Typinformationen (Type Information)
 - Laufzeitzugriff auf Typinformationen des COM Objektes
 - wird vom Microsoft IDL Compiler generiert und in einer Typbibliothek gespeichert
 - COM-Schnittstellen zum Interagieren mit dieser Bibliothek
- Strukturiertes Speichermodell (Structured Storage and Persistence)
 - von COM unterstützte Methode zum Speichern von Daten
 - Speicherung erfolgt analog des Dateisystems in einer Datei
- Moniker
 - Dienst, welcher ein einzelnes Objekt in einem genau definierten Zustand erzeugen und initialisieren kann
 - für Clients, die mit exakt dem gleichen Objekt weiterarbeiten müssen
 - Moniker kann an gesamtes Objekt oder an einen Teil gebunden werden

- Einheitlicher Datenaustausch (Uniform Data Transfer)
 - Datentransfer zwischen COM-Objekten
 - Benachrichtigung von Datenänderungen einer Quelle (Datenobjekt) und einem Datenkonsumenten
- Verbindbare Objekte (Connectable Objects)
 - zur Ereignisverarbeitung
 - Objekt definiert ein Interface, welches für das Ereignis genutzt werden soll
 - Client implementiert dieses Interface
- COM+ Ereignisdienst (Event Service)
 - Asynchrone Kommunikation zwischen Komponenten
 - Empfänger abonniert Ereignis beim Dienst
 - Sender schickt Ereignis zum Dienst, ohne den/die Empfänger zu kennen
- COM+ Nachrichtenwarteschlange (Message Queuing)
 - Garantierte Auslieferung auch bei Nichterreichbarkeit des Empfängers
 - Für Systeme, deren Verfügbarkeit nicht immer gewährleistet ist

- COM unterstützt keine Vererbung (nur Schnittstellenvererbung)
- Quellcode der Komponente kann auf Vererbung basieren
- 2 Wege für Wiederverwendung
 - **Kapselung** (containment)
 - Objekt hat exklusive Referenz auf ein anderes
 - Äußeres Objekt "enthält" wieder verwendetes Objekt
 - Aufrufe an Inneres Objekt werden weitergeleitet (simpler Methodenaufruf)
 - Transparent gegenüber Client-Objekten



- COM unterstützt keine Vererbung (nur Schnittstellenvererbung)
- Quellcode der Komponente kann auf Vererbung basieren
- 2 Wege für Wiederverwendung
 - **Aggregation**
 - Referenz des wieder verwendeten Objektes wird herausgereicht
 - Kein Weiterleiten => Zeitersparnis
 - Inneres Objekt muss Aggregation unterstützen
 - Schnittstelle des wieder verwendeten Objektes muss Schnittstelle des Äußeren kennen
 - Mehrere Hierarchieebenen möglich



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

Westphal, 2002

- Bündel von Softwaretechnologien zum Verbinden von Informationen, Menschen, Systemen und Geräten
- Ziele
 - Sicherheit
 - Plattformunabhängigkeit
 - Interoperabilität
 - Homogenität

- August 2000 – Einreichung zur Standardisierung bei ECMA
 - mit Unterstützung von HP und Intel
 - Standardisierung von C#
 - Standardisierung einer Common Language Infrastructure (CLI)
 - Untermenge der Klassenbibliotheken von .NET
 - Dateiformat
 - Typsystem
 - erweiterbares Metadatensystem
 - Intermediate Language (IL)
 - Zugriff auf die zugrunde liegende Plattform
 - skalierbare Basisklassenbibliothek
 - was fehlt:
 - Graphische Benutzerschnittstelle
 - Datenbankzugriff

- Dezember 2001 – Fertigstellung des ersten Standards
 - mit Unterstützung von IBM, Fujitsu, Netscape, Sun u.a.
 - Weitergabe an die ISO
- April 2003 – Verabschiedung des ISO-Standards
 - ISO/IEC 23270 (C#)
 - ISO/IEC 23271 (CLI)
- Oktober 2003 – Standardisierung der Bindung von C++ an die CLI beginnt
 - ECMA-372 (Dezember 2005)
- Aktuelle Implementierungen
 - Microsoft (Windows) <http://msdn.microsoft.com/net>
 - MONO (Linux) <http://www.go-mono.com>

Microsoft .NET Strategie

.NET Framework

Visual Studio .NET
Codeeditor
Fenstereditor
Debugger
Server-Explorer
Entwurfshilfen
C# / VB .NET / J#
ASP.NET

.NET Enterprise Server

Application Center
Exchange Server
BizTalk Server
...
- werden speziell an
.NET angepasst
- Dienste zentral
für .NET Anwen-
dungen

.NET My Services

konkrete
WebServices

.NET Contacts
.NET Wallet
.NET Lists
Microsoft Passport
...

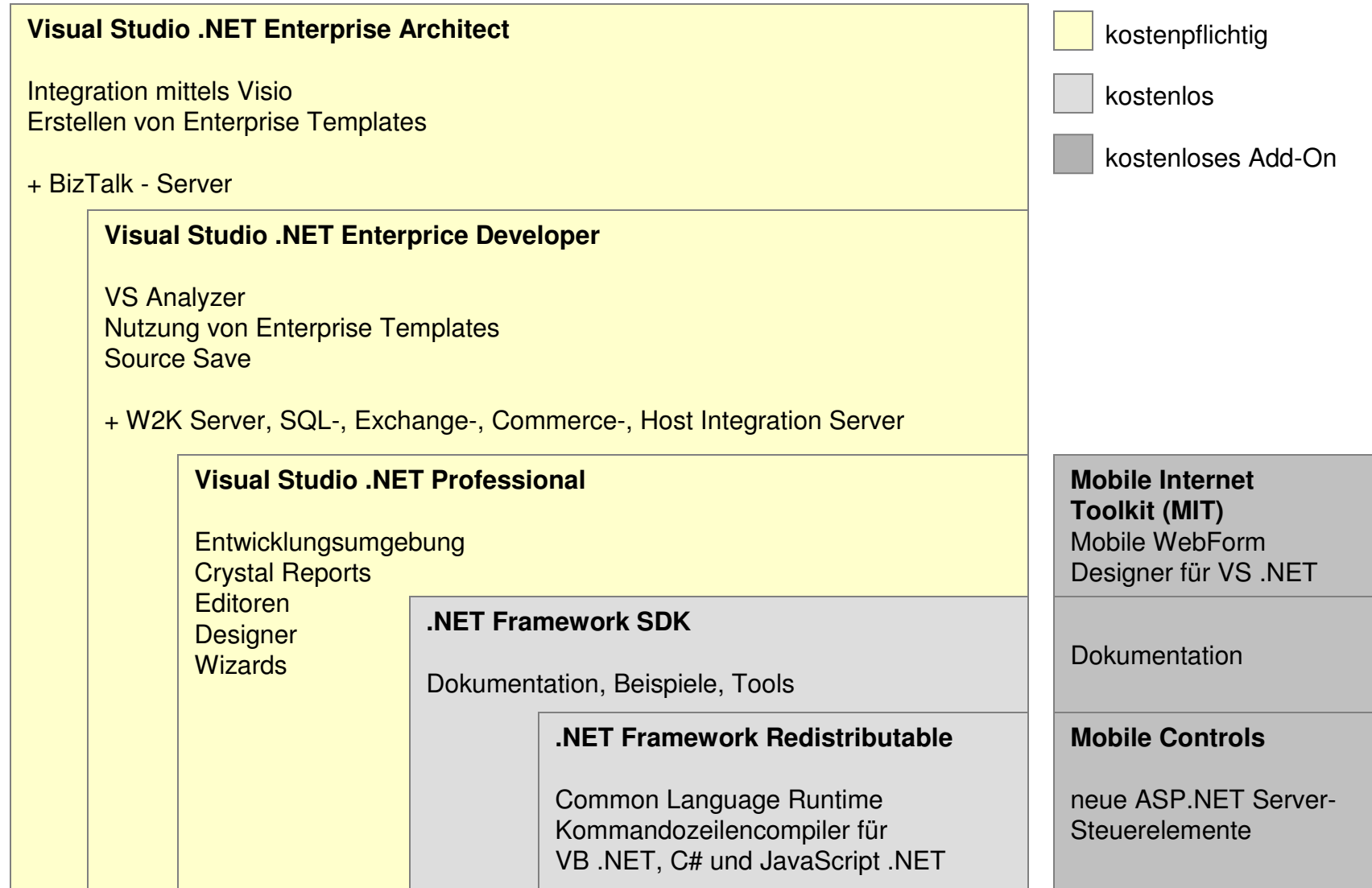
.NET Devices

Hardware wie PDA,
Handys, Tablet,
PC, Auto-PC ...

- Mobile Internet
Toolkit (MIT)
- .NET Compact
Framework (CF)

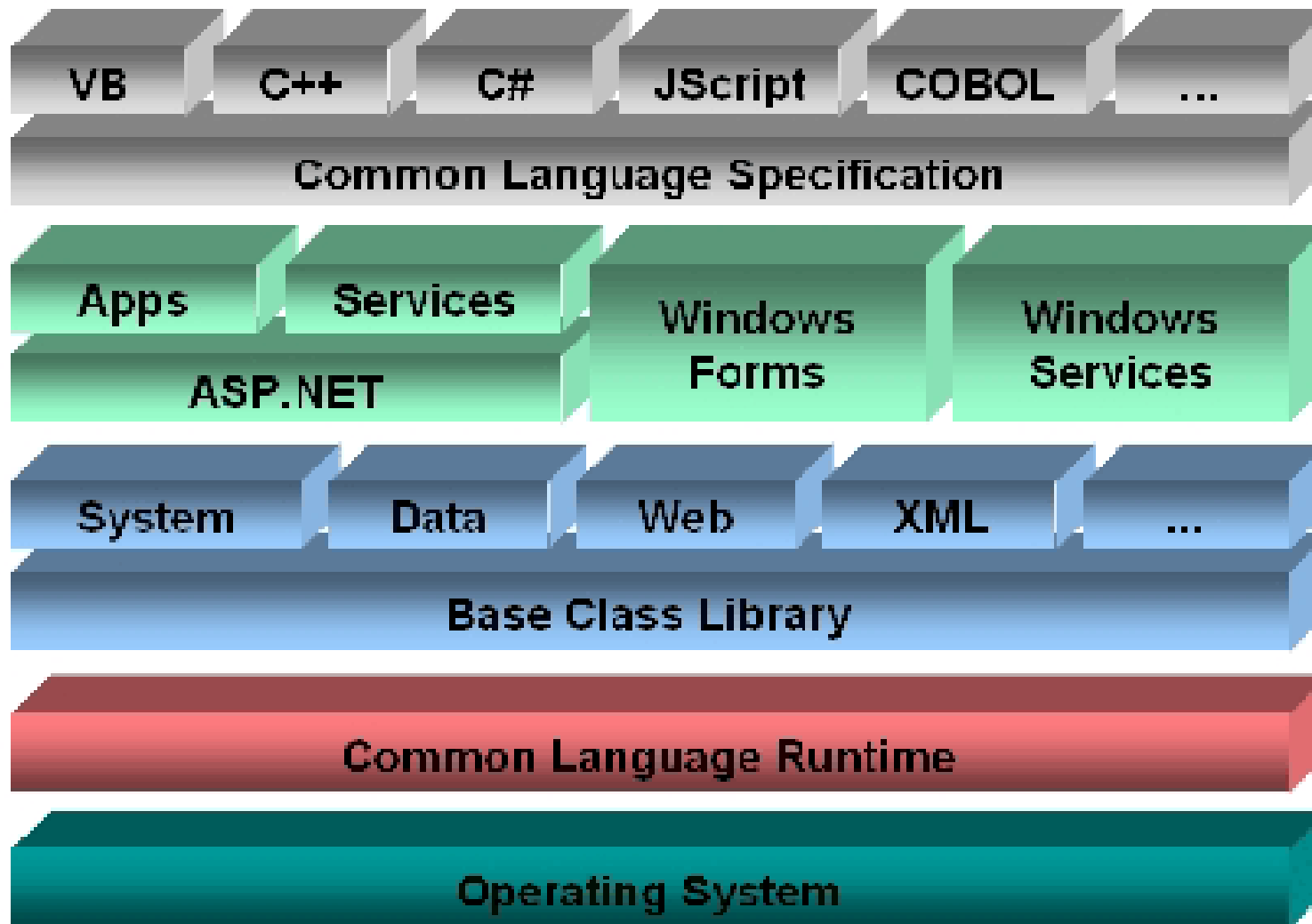
3.4 Microsoft COM und .NET

Die Entwicklungsumgebung von Microsoft



3.4 Microsoft COM und .NET

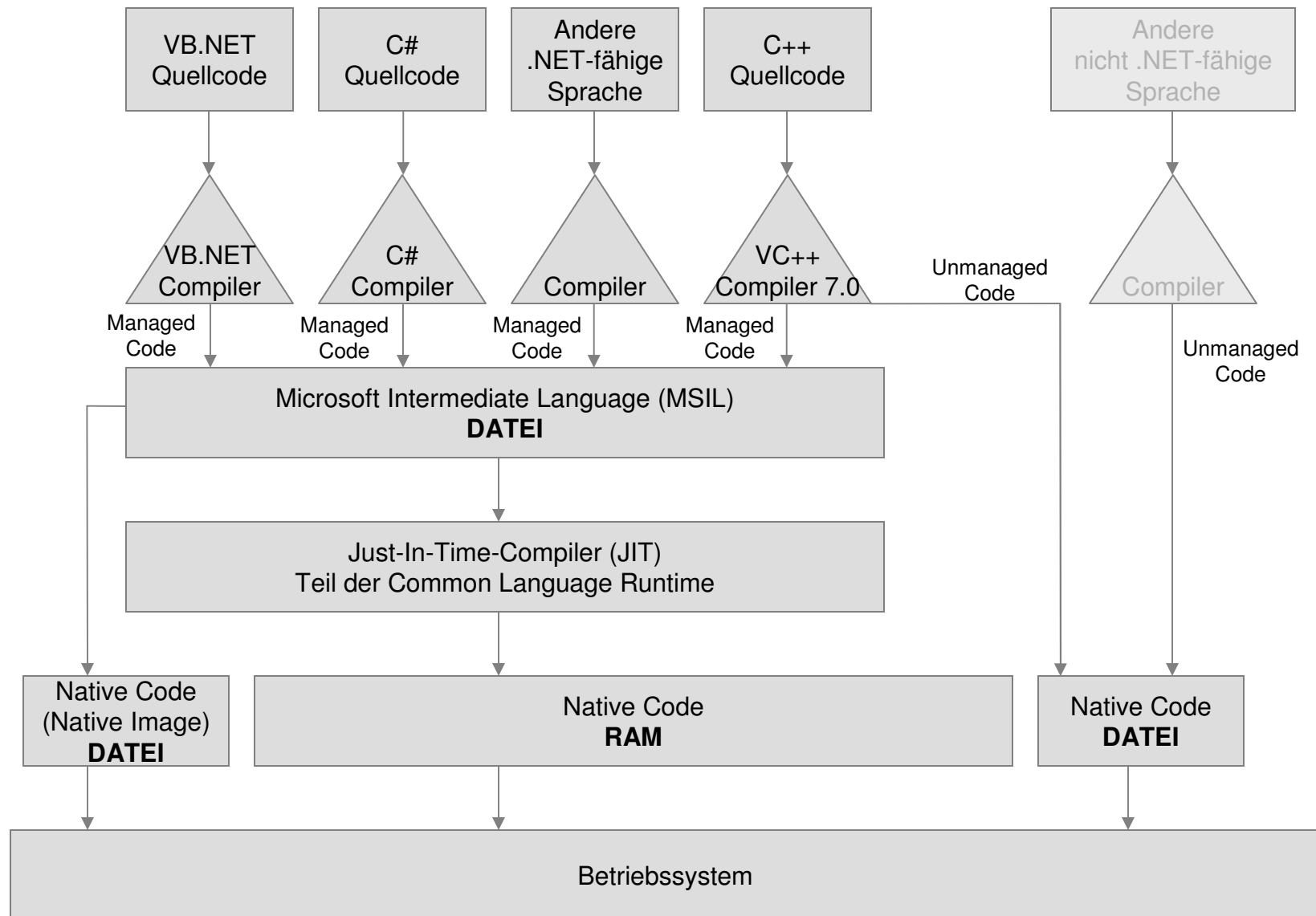
Das .NET - Framework



Quelle: Microsoft

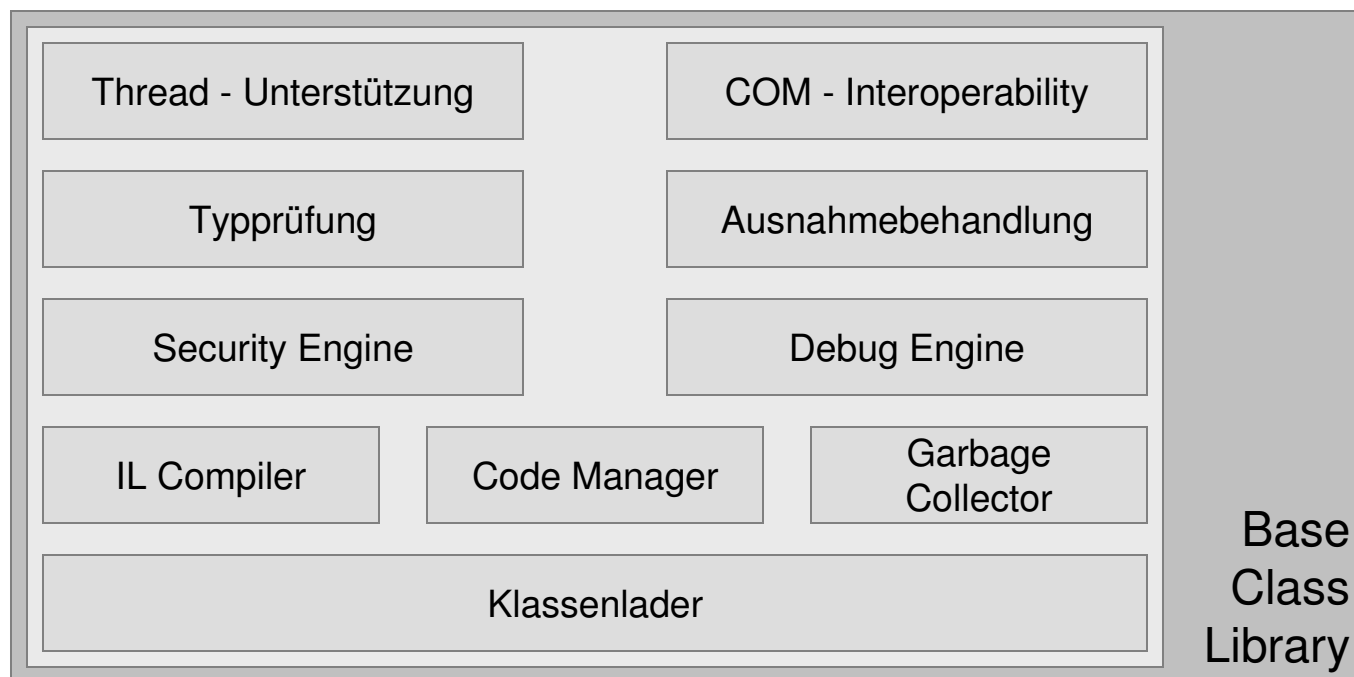
3.4 Microsoft COM und .NET

Das .NET - Framework



3.4 Microsoft COM und .NET Common Language Runtime

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



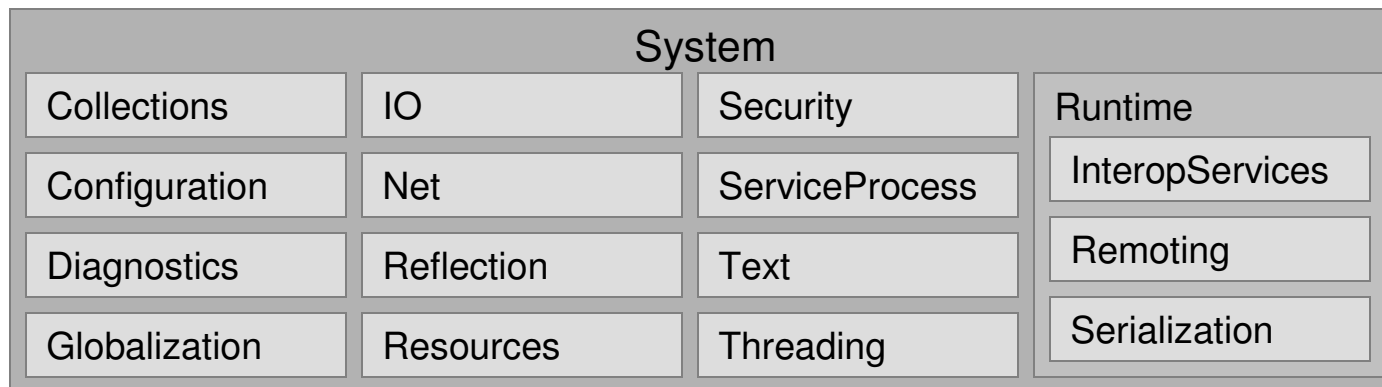
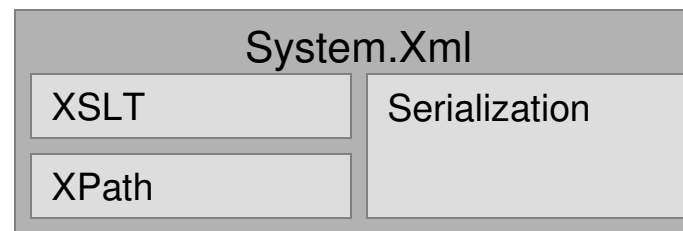
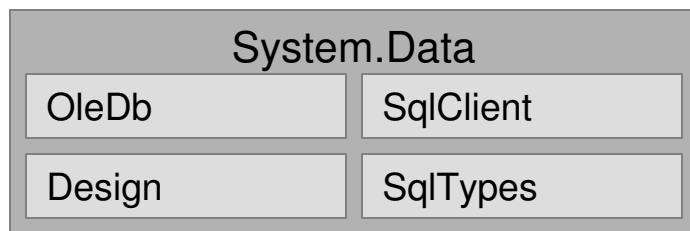
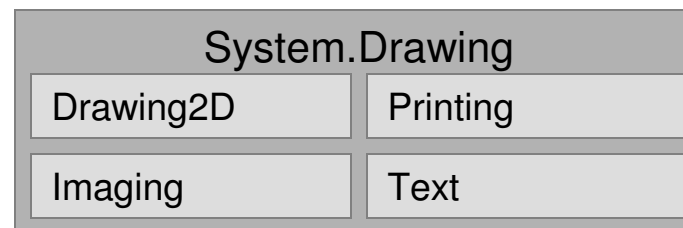
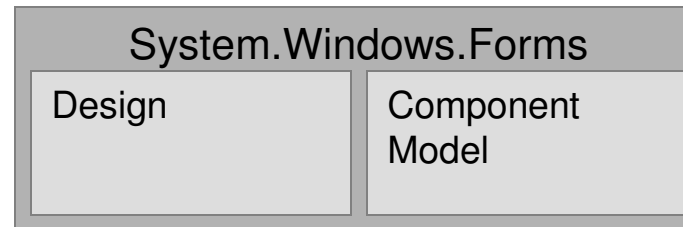
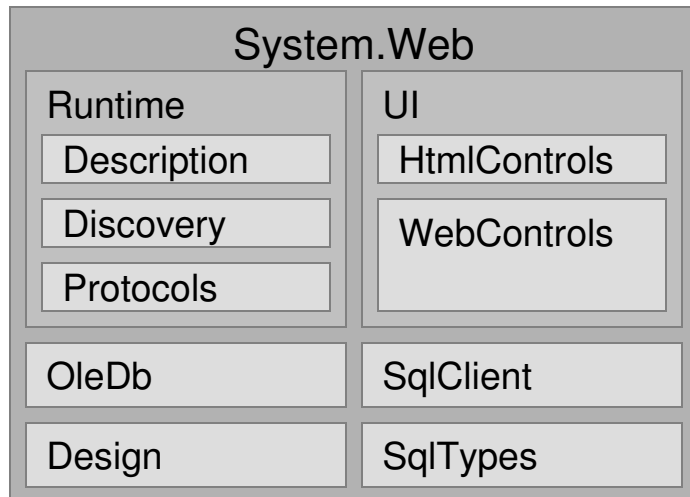
- Konsistentes Programmiermodell
 - alle Anwendungsdienste als objektorientiertes Programmiermodell
- Vereinfachtes Programmiermodell
 - keine Registrierung, COM-Schnittstellen, HRESULTs ...
- Stabile Installationen
 - isolierte Anwendungskomponenten
 - keine >DLL-Hölle< mehr
 - Versionierung von Komponenten
- Vereinfachte Installationen
 - Anwendungsdateien einfach in Zielverzeichnis kopieren
 - 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

3.4 Microsoft COM und .NET

Common Language Runtime

- 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 / der Daten
- Interoperabilität
 - Zugriff auf COM-Komponenten

3.4 Microsoft COM und .NET Base Class Library



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

- IL 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
```

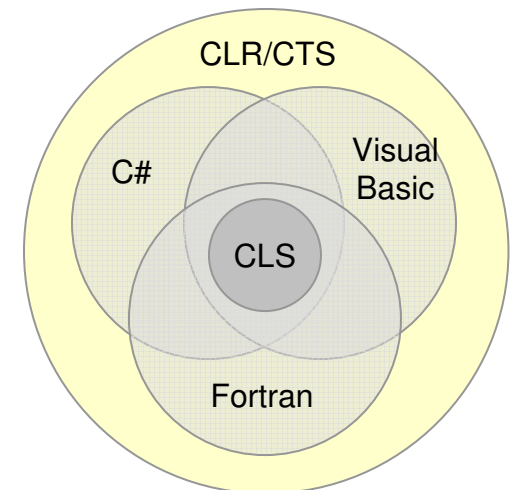
Beispiel für IL-Code

3.4 Microsoft COM und .NET 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

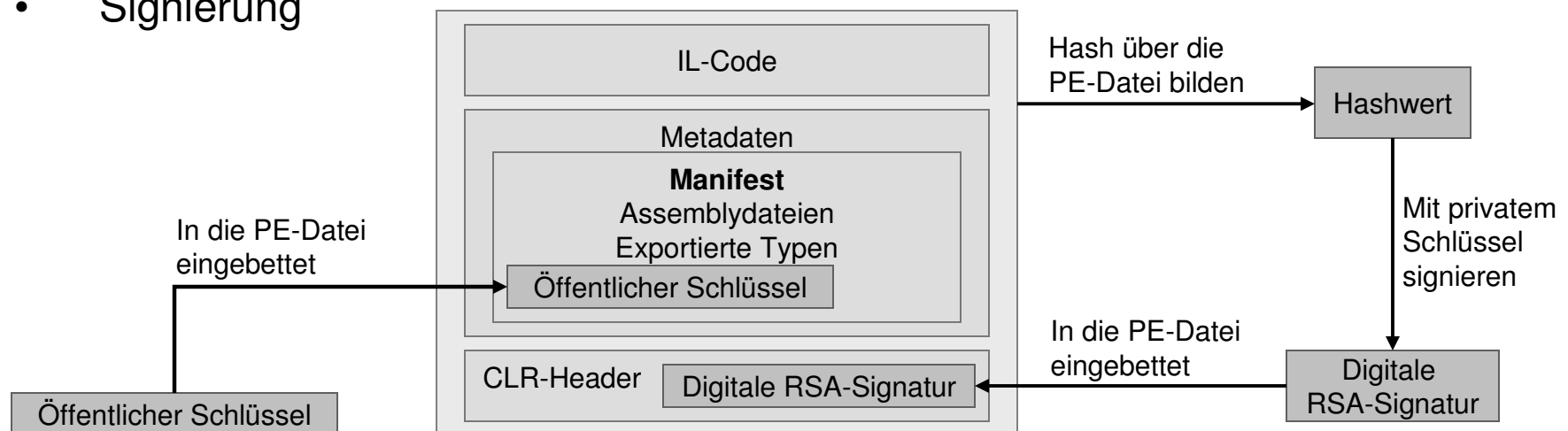
// Fehler, weil Klasse öffentlich ist
public class App {
    // Fehler, weil UInt32 nicht CLS-Kompatibel
    public UInt32 Abc() { return 0; }
    // Fehler, weil keine Unterscheidung zwischen Groß- und Kleinschreibung in CLS
    public void abc() {}
    //Kein Fehler, da Methode privat ist
    private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



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

- atomare, selbstbeschreibende Einheit, inklusive Metadaten
- Metadatenstruktur wird **Manifest** genannt
- "single file assembly"
 - Manifest ist Teil des eigentlichen Codes
 - z.B. bei DLLs
- "multi file assembly"
 - Manifest ist eigenständige Einheit
- ein Manifest enthält ...
 - Identität der Assembly (Name, Version, ...)
 - die Namen aller Dateien im Assembly
 - kodierter Hashwert aller Dateien im Assembly
 - Details der vorhandenen Klassen, Methoden und Eigenschaften
 - Namen und Hashwerte aller referenzierten Assemblies
 - Sicherheitseinstellungen

- Starke Namen (strong names)
 - zur eindeutigen Identifizierung einer Assembly
 - Dateinamen (ohne Erweiterung)
 - Versionsnummer
 - Kultur
 - Token für einen öffentlichen Schlüssel
 - kann global weitergegeben werden (Global Assembly Cache)
 - Möglichkeit zur Sicherstellung der Unversehrtheit des Codes
- Signierung



3.4 Microsoft COM und .NET

Global Assembly Cache (GAC)

- genau definiertes Verzeichnis, in der globale Assemblies abgelegt werden
 - C:\Windows\Assembly\GAC
 - niemals von Hand Dateien in GAC kopieren
- festdefinierte Struktur:
 - Name
 - Versionsnummer_Kultur_Schlüsseltoken
- Vorteil: Assembly kann global genutzt werden
- Nachteil: Keine einfache Installation mehr möglich

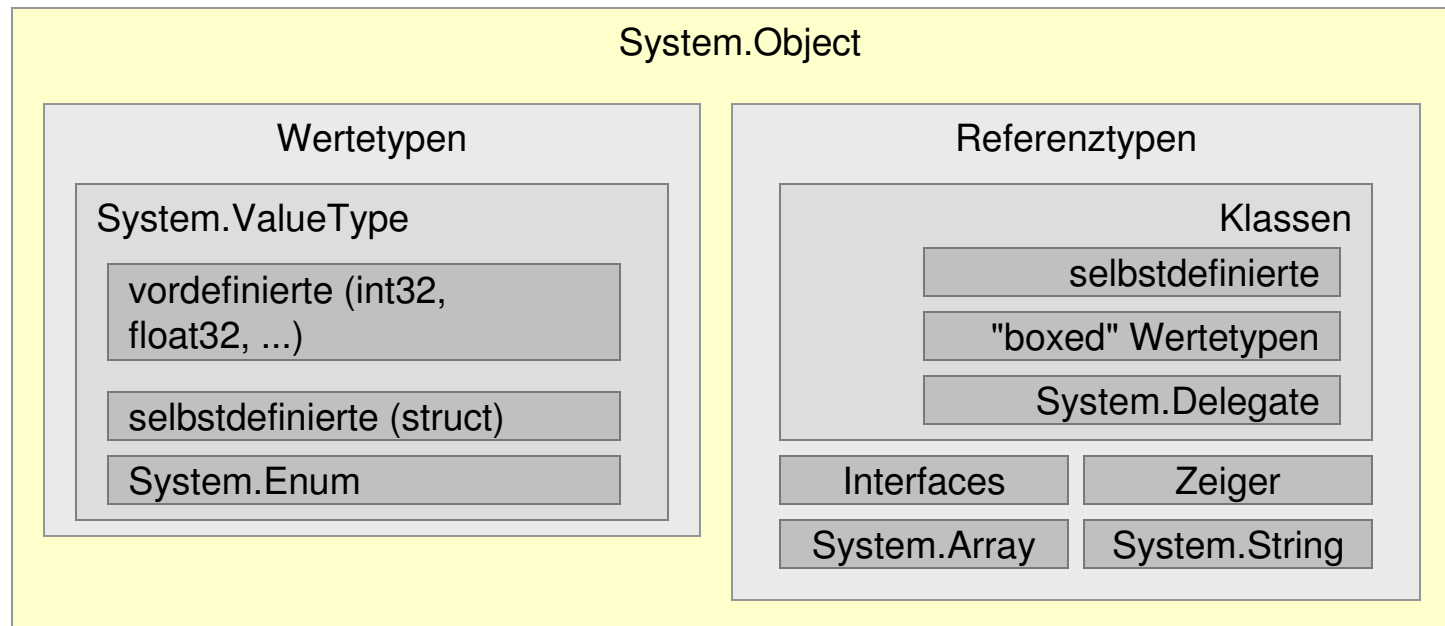
3.4 Microsoft COM und .NET

Elemente eines verwalteten Moduls

- PE-Header
 - Windows-Standardheader für PE-Dateien
 - definiert Dateityp (GUI, CUI, DLL)
 - Zeitstempel
 - Informationen über speziellen CPU-Code
- CLR-Header
 - benötigte Version der CLR
 - spezielle Flags
 - Metadaten token MethodDef (Einstiegspunkt)
 - Adresse und Größe von Metadaten des Moduls
 - Ressourcen
 - strong names

- Metadaten
 - Typen und Member, die im Quellcode definiert sind
 - Typen und Member, auf die aus dem Quellcode zugegriffen wird
- IL-Code
 - vom Compiler generierter Code
 - Programmiersprachenunabhängig
 - wird bei der Ausführung von der CLR in CPU-spezifische Befehle kompiliert

3.4 Microsoft COM und .NET Common Type System (CTS)



- 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

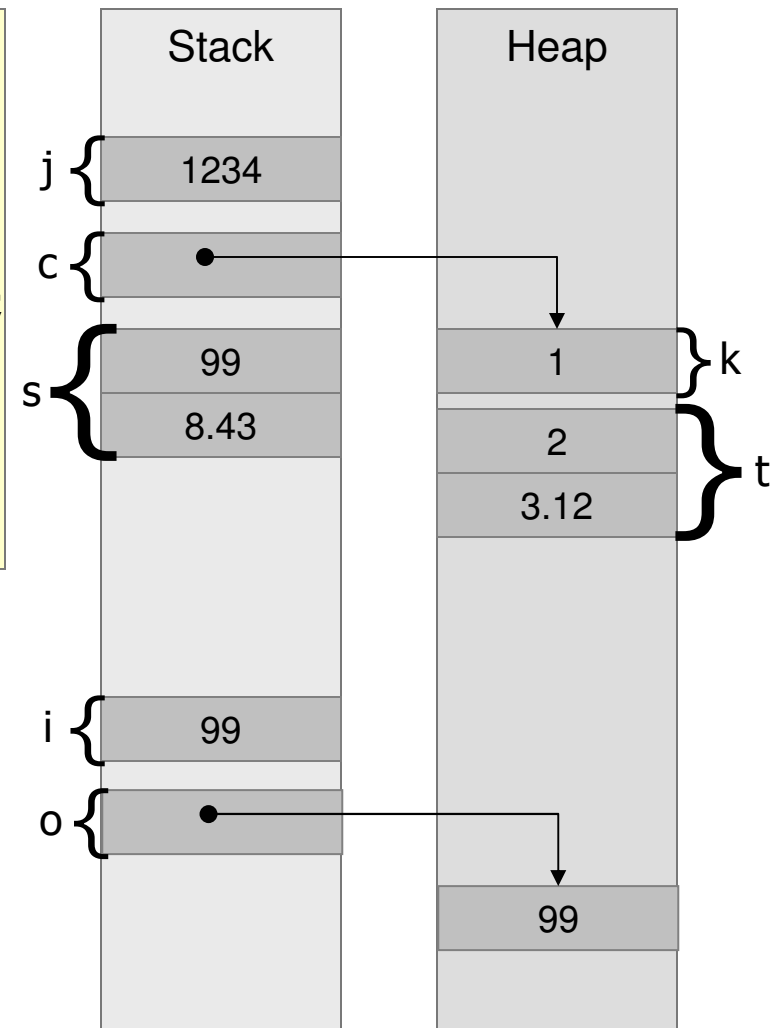
Verweis- und Referenztypen

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

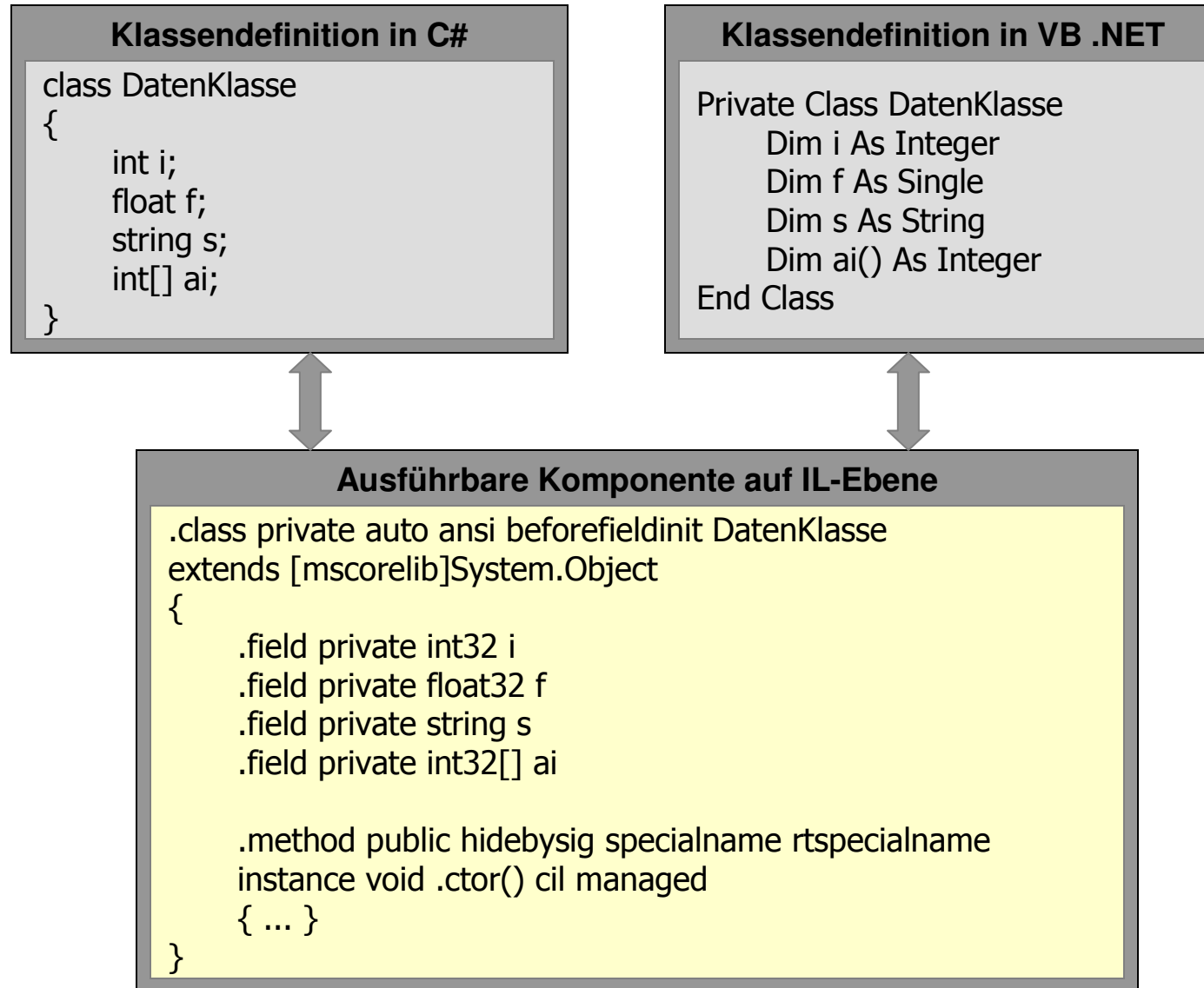
```
int j;  
j = 1234;  
  
MyClass c = new MyClass();  
c.k = 1; c.t.i = 2; c.t.f = 3.12;  
  
MyStruct s;  
s.i = 99; s.f = 8.43
```

Boxing

```
int i;  
i = 99;  
  
object o;  
o = i;
```



3.4 Microsoft COM und .NET Common Type System (CTS)



3.4 Microsoft COM und .NET

Common Type System (CTS)

- 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
        ...
    }
}
```

- Kopieren von Dateien in Zielverzeichnis, z.B. per Batch
 - alle abhängigen Verweise und Typen sind in Assembly enthalten
 - referenzierte Assemblies im Anwendungsverzeichnis
- Konventionelle Installation möglich (cab, msi ...)
 - Verknüpfungen auf Desktop, Startleiste ...
 - Einbindung in Softwareverwaltung von Windows
 - zukünftig eventuell Automation von Verknüpfung
- Private Assemblies
 - werden in Anwendungsverzeichnis installiert
 - werden **nur** von jeweiliger Anwendung benutzt
 - es werden nur Typen gebunden, die für die Anwendung passen

- Konfigurationsdatei im Anwendungsverzeichnis zur Steuerung der Anwendung
 - Pfadeinstellung für Zusatzassemblies

Anwendungsverzeichnis
App.exe
App.exe.config
Zusatzassembly
zusatz.dll
xxx.netmodule
yyy.netmodule

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <probing privatePath="Zusatzassembly" />
    </assemblyBinding>
  </runtime>
</configuration>
```

- Suche nach Assemblydateien

AppBase\culture\AsmName.dll
AppBase\culture\AsmName\AsmName.dll
AppBase\culture\privatePath1\AsmName.dll
AppBase\culture\privatePath1\AsmName\AsmName.dll
AppBase\culture\privatePath2\AsmName.dll
AppBase\culture\privatePath2\AsmName\AsmName.dll

- bei Misserfolg wird nach .exe gesucht
- Konfigurationsdatei kann während der Laufzeit durch Klassen aus dem Namespace "System.Configuration" manipuliert werden

- Systemweite Konfiguration möglich (Machine.config)
 - *C:\Windows\Microsoft.NET\Framework\version\CONFIG*
 - Einstellungen in Machine.config überschreiben anwendungsspezifische Einstellungen
 - Angaben zu den von der CLR verwendeten Dateien
 - Einstellungen für den Zugriff vom Internet aus
 - Einstellungen für die Kompilierung von IL-Code

- ".NET kompakt" von Ralf Westphal
- "Visual Basic .NET Class Design Handbook" von Andy Olsen et al
- Microsoft .NET Framework Programmierung von Jeffrey Richter
- Softwareentwicklung mit C# von Hanspeter Mössenböck
 - <http://dotnet.jku.at>.
- Microsoft Website
 - <http://www.microsoft.com/net>
- it-Visions Website
 - <http://www.it-visions.de>
- .netXtreme
 - <http://www.dotnetextreme.com/articles/assemblies.asp>
- "Lehrbuch der Software-Technik" von Helmut Balzert
- "Komponentenmodelle" von Volker Gruhn, Andreas Thiel