

Einführung in Microsoft .NET

Martin Saternus

Technical Student Consultant

Microsoft Academic Program

Microsoft Deutschland GmbH

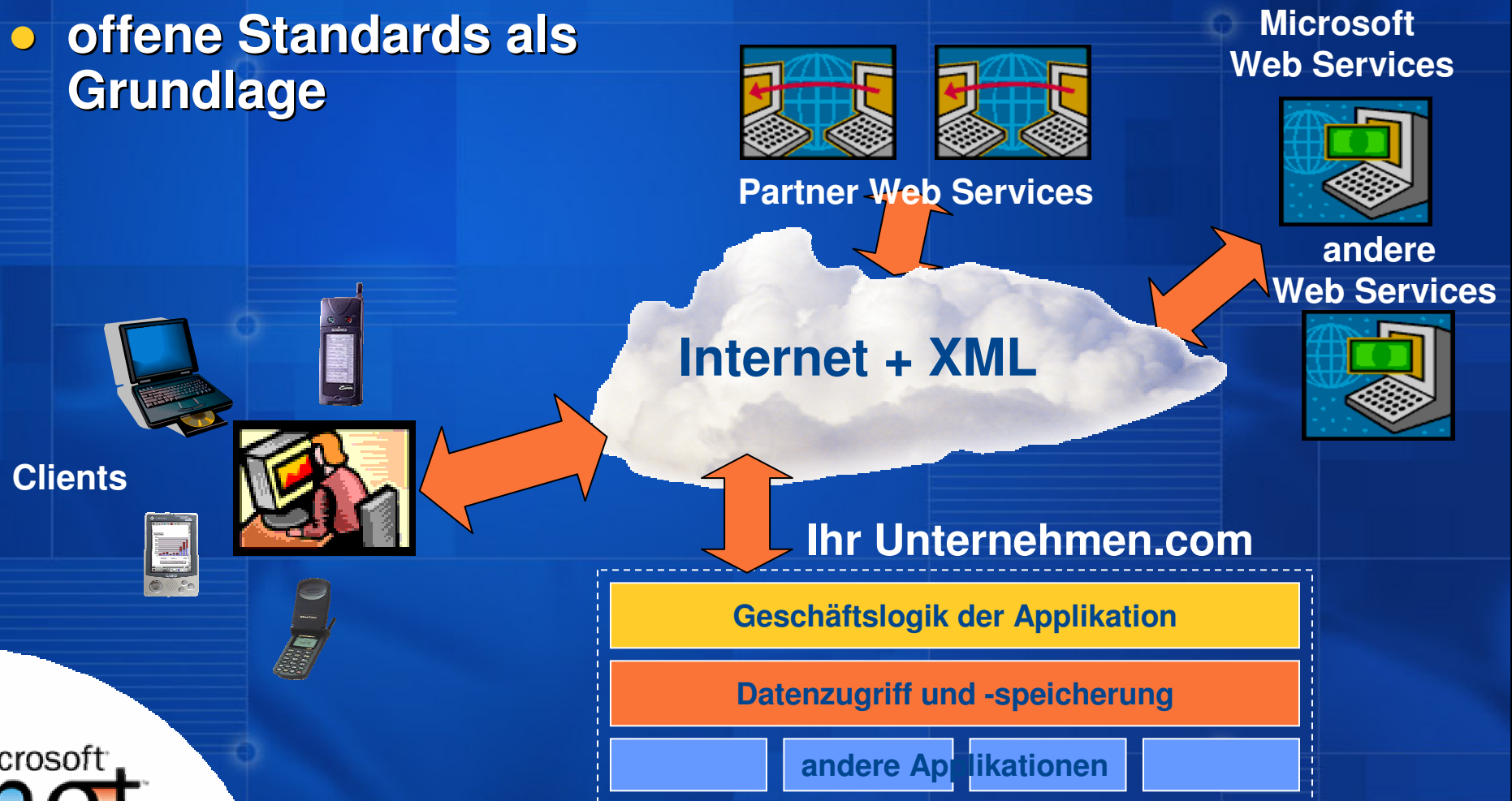
i-martsa@microsoft.com

In diesem Vortrag

- **Der Weg zu Microsoft .NET**
- **Common Language Runtime (CLR)**
- **Base Class Library (BCL)**
- **Deklarative Programmierung mit .NET**

Vision

- Software als Service
- Unterstützung von Smart Devices
- offene Standards als Grundlage



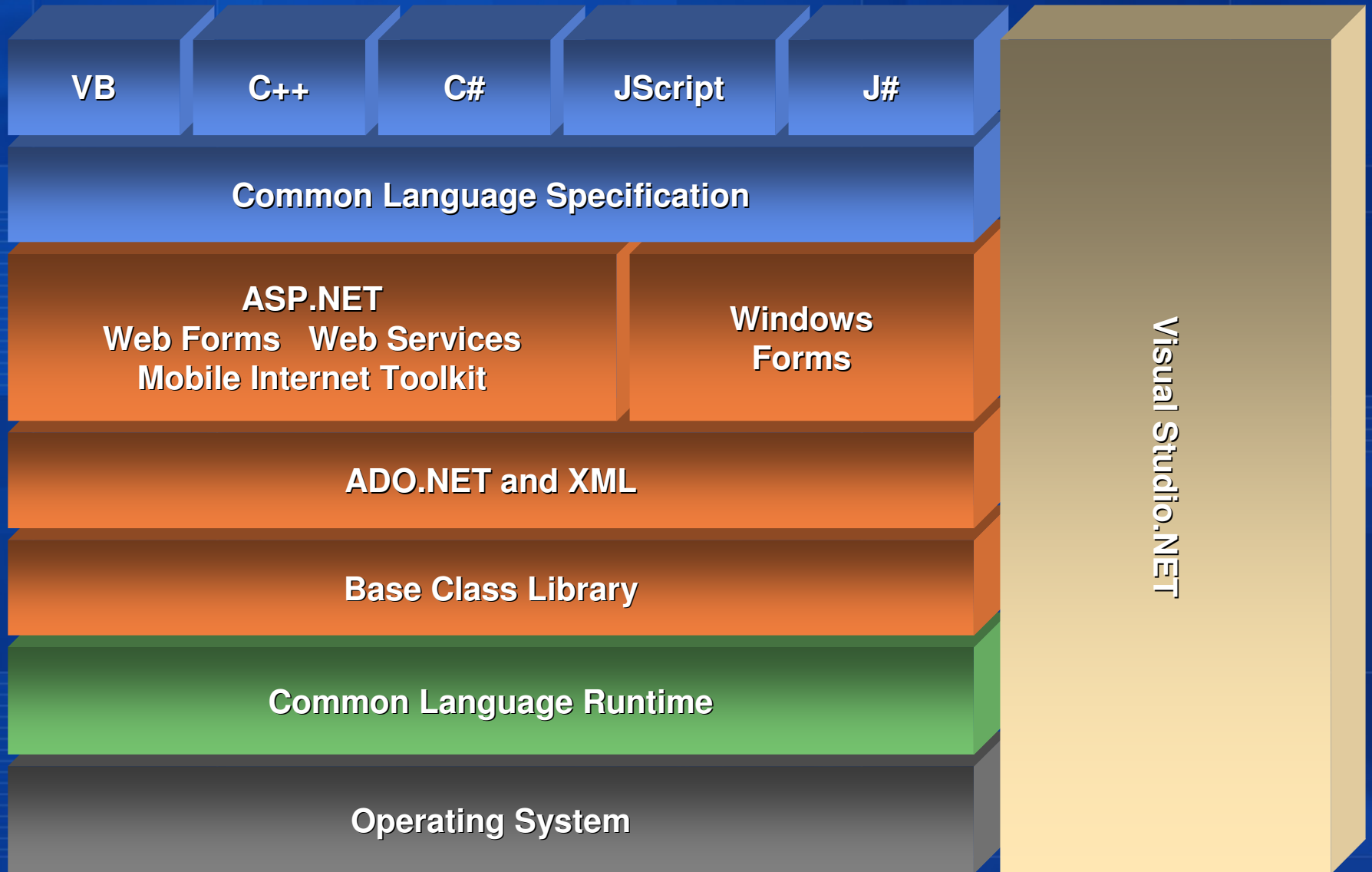
Entwicklungsparadigmen (1/2)

- Vereinfachung von Development und Deployment
 - Selbstbeschreibende Komponenten
 - Hierarchische Namespaces
 - Einheitliches Objektmodell
 - Strukturierte Exceptions
 - Gemeinsames Root Object
- Common Type System (CTS)
 - Einheitliche Vererbungs- und Typdefinitionen

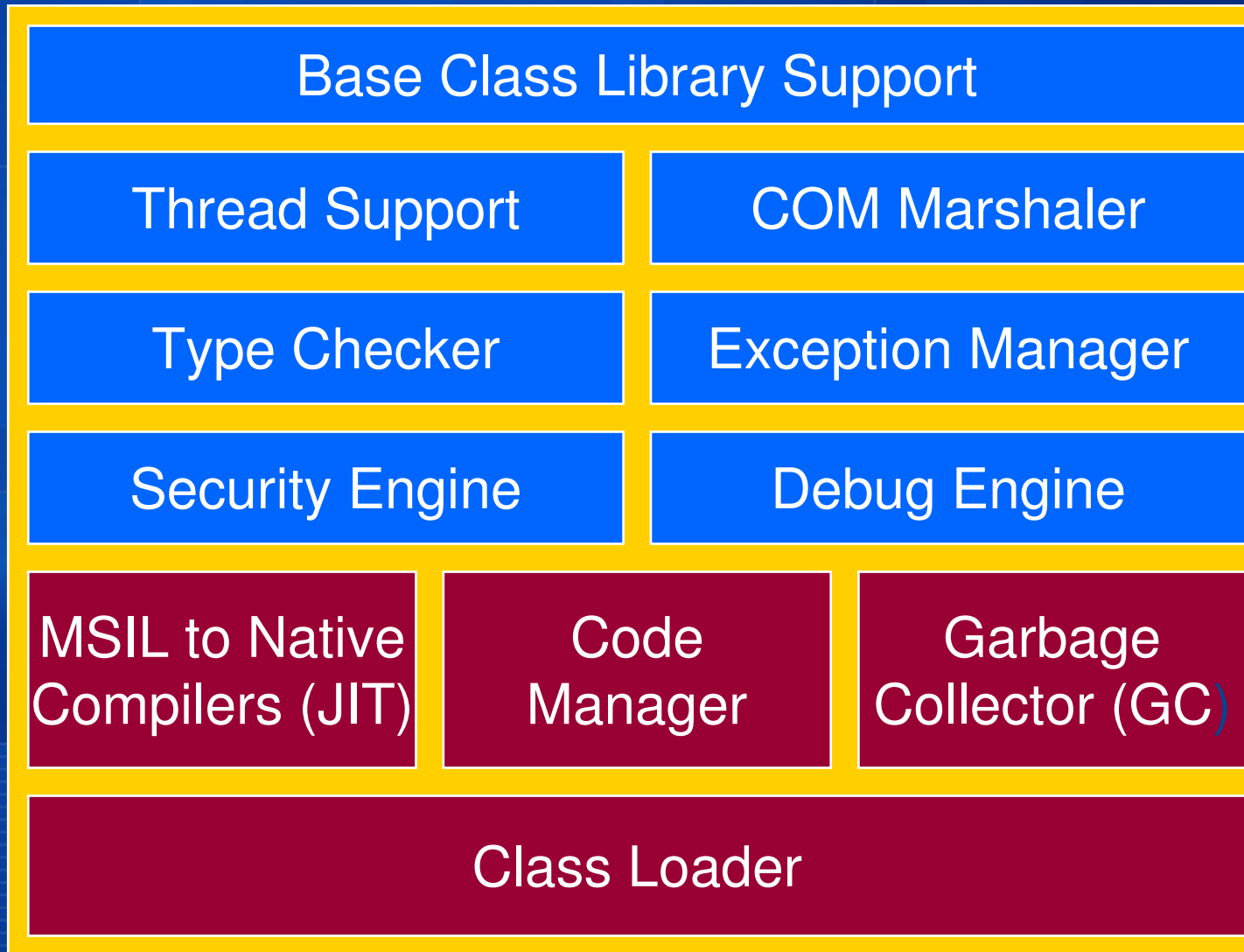
Entwicklungsparadigmen (2/2)

- **Robuste und sichere Ausführungsumgebung**
 - **Code Access Security**
 - Sicherheit basiert auf der Identität des Codes
 - Administrativ über Richtlinien konfigurierbar
 - **ASP.NET: Integrierte Benutzer-Authentifizierung**
 - Windows identity, Passport®, forms-based, ...
 - **Kryptographie Bibliothek für XML**
- **Deployment und Management**
 - **Side-by-side Ausführungen**
 - Verschiedene Versionen derselben Komponente können auf einem System nebeneinander existieren

.NET Framework



Common Language Runtime



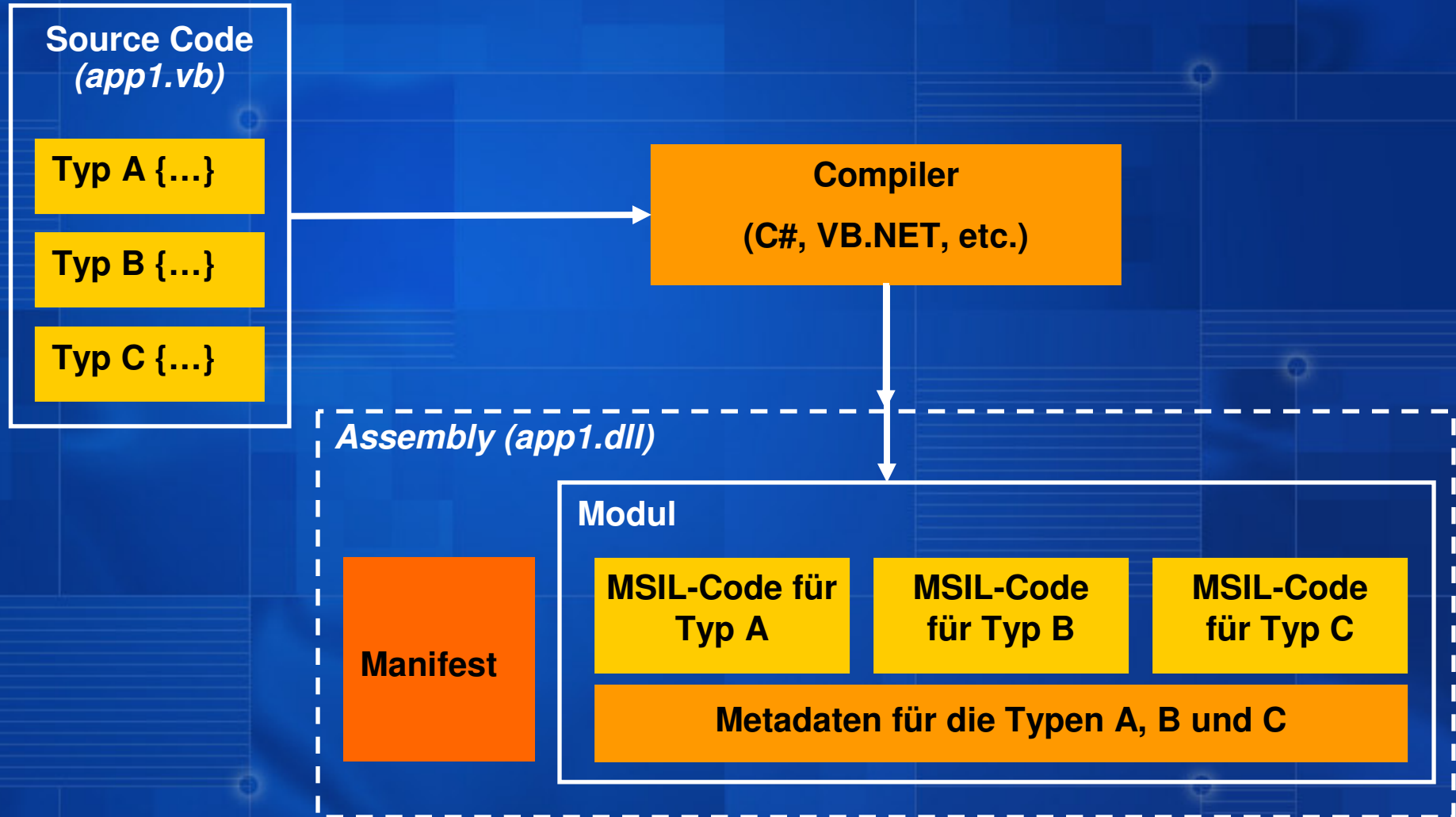
Die CLR

- Jeder Win32-Prozess, der die CLR benutzen möchte, muss diese laden
- Multi-Instance-Singleton
 - genau eine CLR pro Win32 Prozess
 - Viele Instanzen von CorRuntimeHost
 - Alle Instanzen greifen auf gemeinsame Daten zu
- Die Execution Engine der CLR ist in Form einer COM-DLL implementiert
 - **CorRunTimeHost**

Ausführen einer EXE-Datei

- Eine Instanz der CLR wird erzeugt
- CLR erzeugt eine Appdomain
- EXE Datei implementiert einen Stub aus der MSCOREE.DLL
- IL-Code wird durch den JIT Compiler in nativen Code übersetzt
- Assembly der EXE-Datei wird in die AppDomain geladen
- Die CLR sucht nach der statischen Methode Main und führt diese aus
 - ggf. wird eine Exception geworfen

Übersetzen von Sourcen



Kategorien (laut Doku)

- **Private Assembly**
 - Assembly kann nur von genau einer Anwendung benutzt werden
- **Shared Assembly**
 - Assembly kann global von allen Anwendungen benutzt werden

Private Assembly

- Identifikation anhand eines einfachen Namens, z.B. "Reverse"
- Keine Versionsüberprüfung
- Installation per Filecopy
 - Standardmäßig befinden sich Assembly und Anwendung im gleichen Verzeichnis
 - Verzeichnis kann per CFG-Datei definiert werden

Shared Assembly

- Identifikation über einen **Strong Name**
 - Eindeutig per Public-Key-Verschlüsselung
 - Strong Name = Identität + Public Key
- Versionsüberprüfung durch die Runtime
- Installation im Global Assembly Cache (→ SDK-Tool gacutil.exe)
 - systemweiter “Speicherbereich”
 - basiert auf dem Filesystem, normale Dateien
 - keine Registry-Einträge

Strong Name & Codesigning

- **Strong Name garantiert Codeintegrität**
 - **Unbemerktetes Einschleusen von modifiziertem Code wird verhindert**
 - **Aber: Auch böse Menschen arbeiten mit Strong Names**
- **Code-Signing garantiert Codeidentität**
 - **Code ist aufgrund seines X.509-Zertifikats eindeutig identifizierbar**

Assemblies



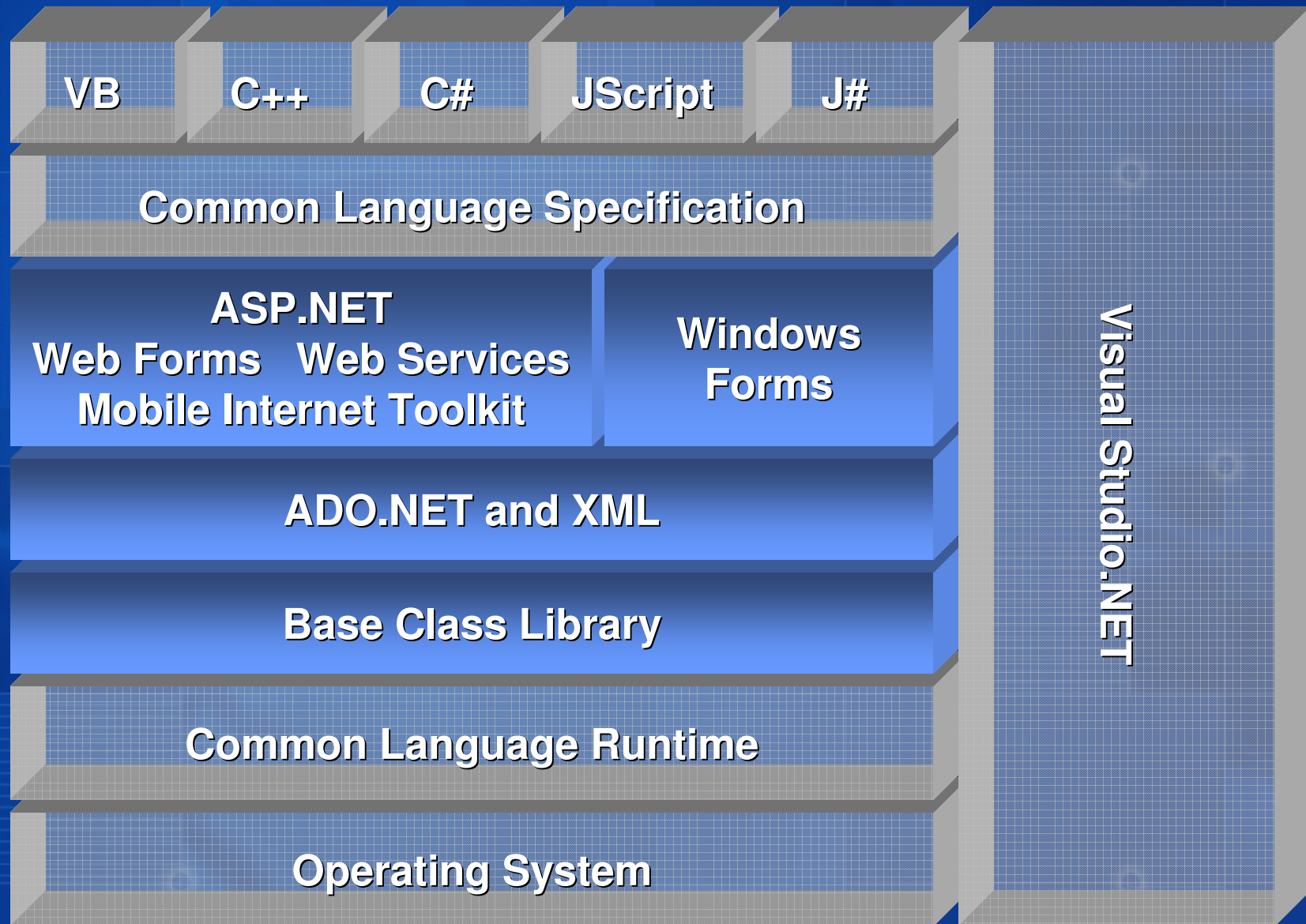
Kategorien in der Praxis

- **Private Assemblies**
 - Alle Assemblies, die im nicht-produktiven Betrieb eingesetzt werden
- **Published Assemblies**
 - Alle Assemblies, die im produktiven Betrieb eingesetzt werden
 - Diese Assemblies sollten grundsätzlich mit einem Strong Name versehen werden
 - Shared Assemblies setzen einen Strong Name voraus!

Zwischenstation...

- **Assemblies dienen zur physikalischen Organisation von Code**
 - Flexibles Gruppieren von Modulen
 - Mobiler Code
- **Shared Assemblies benutzen keine Registry**
- **Setzen Sie bei allen Assemblies, die Sie ausliefern Strong Names ein!**
 - Nach Möglichkeit sollten Sie auch Codesigning einsetzen

Framework, Languages und Tools



Einheitliches Programmier Model

Gleiche API ungeachtet der
Sprache und des Modells

.NET Framework

VB Forms

MFC/ATL

ASP

Windows API



Wieviel einfacher?

Windows API

```
HWND hwndMain = CreateWindowEx(  
    0, "MainWClass", "Main Window",  
    WS_OVERLAPPEDWINDOW | WS_HSCROLL | WS_VSCROLL,  
    CW_USEDEFAULT, CW_USEDEFAULT,  
    CW_USEDEFAULT, CW_USEDEFAULT,  
    (HWND)NULL, (HMENU)NULL, hInstance, NULL);  
ShowWindow(hwndMain, SW_SHOWDEFAULT);  
UpdateWindow(hwndMain);
```

.NET Framework

```
Form form = new Form()  
form.Text = "Main Window"  
form.Show()
```

Klassenbibliothek

System.Web

Services

Description

Discovery

Protocols

Caching

Configuration

UI

HtmlControls

WebControls

Security

SessionState

System.Windows.Forms

Design

ComponentModel

System.Drawing

Drawing2D

Imaging

Printing

Text

System.Data

OleDb

Common

SqlClient

SQLTypes

System.Xml

XSLT

XPath

Serialization

System

Collections

Configuration

Diagnostics

Globalization

IO

Net

Reflection

Resources

Security

ServiceProcess

Text

Threading

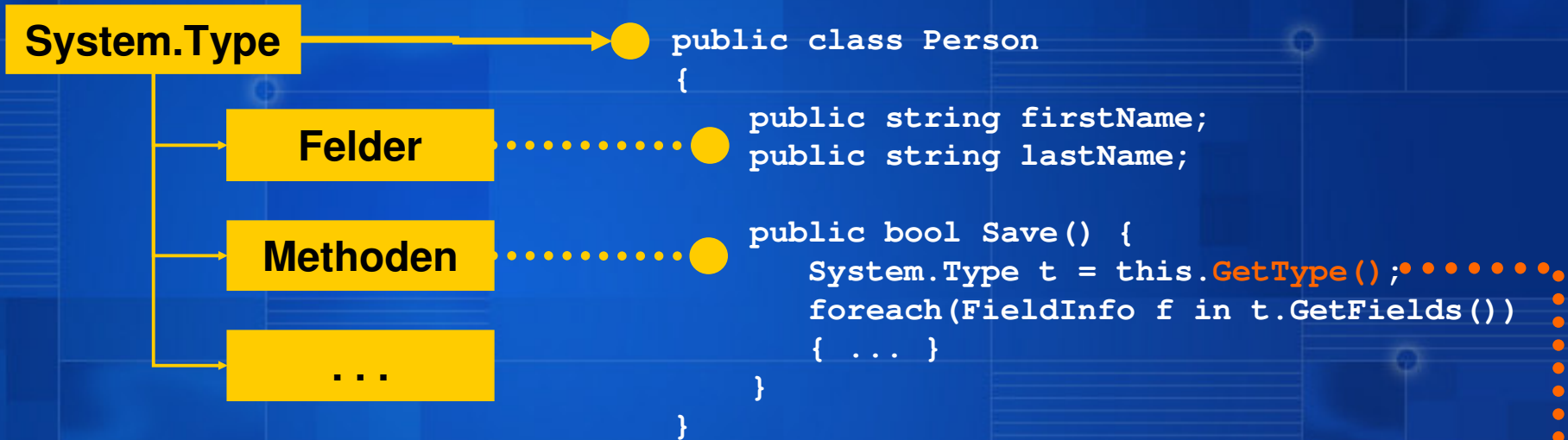
Runtime

InteropServices

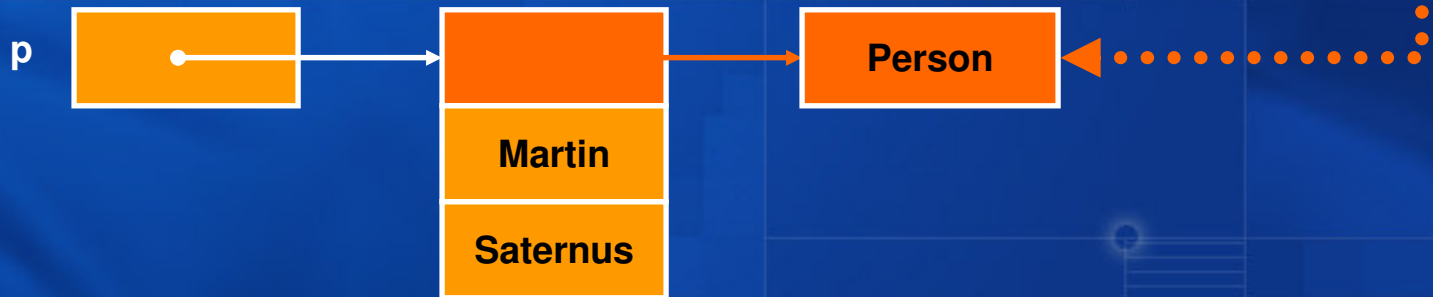
Remoting

Serialization

Metadaten - Reflection



Person p = new Person("Martin", "Saternus");



typeof() vs. GetType()

- Alle Typinformationen können über einen Typ `Type` besorgt werden.
- `GetType` liefert Typ zur Laufzeit
 - Auflösung über Name
 - `Type t = Type.GetType("System.Int32")`
- `typeof` liefert Typ zur Compilezeit
 - Auflösung über "type token"
 - `Type t = typeof(System.Int32)`
 - In VB.NET `TypeOf`

Attribute

- **Klassen und Methoden können über Attribute mit Metadaten versehen werden**
 - Deklarative Definition von Objektverhalten (?)
- **Attribute sind Klassen**
 - Einheitliche Vorgehensweise erleichtert die Definition eigener Attribute
 - Attribute können an Typen und Member „angehängt“ werden
- **Auch Attribute können per Reflection ausgelesen werden**

Definieren von Attributen

- Attribute werden über Klassen definiert, die von `System.Attribute` abgeleitet sind
 - Klasse wird nur „bei Bedarf“ instanziiert
- C# Syntax für Attribute
 - Attribute werden einem Subjekt mit eckigen Klammern vorangestellt
 - Mehrere Attribute werden durch ein Komma voneinander getrennt
- Der Geltungsbereich von Attributen wird über ein eigenes Attribut definiert
 - `System.AttributeUsageAttribute`

Attribute



Fragen !?

