Die .NET Klassenbibliothek Teil 3

School of Engineering

- Windows Forms
- Low Level Programmierung
- Native Calls
- COM Interoperabilität
- Sicherheit

Windows Forms

Beispiel: GUI-HelloWorld

```
public class HelloWorldForm : Form {
        Label lab;

        HelloWorldForm() {
            this.Text = "HelloWorldForm Titel";
            this.Size = new Size(200,100);
            lab = new Label();
            lab.Text = "HelloWorld";
            lab.Location = new Point(20, 20);
            this.Controls.Add(lab);
        }

        public static void Main(string[] argv) {
            Application.Run(new HelloWorldForm());
        }
}
```

```
csc /t:winexe HelloWorldForm
```

Erstellung von GUI-Anwendungen

School of Engineering

- *Window Forms* ist GUI-Framework für Desktop-Applikationen
  (im Gegensatz zu *Web Forms* für Web-Applikationen)

- Namensräume
  - System.Windows.Forms: GUI-Formulare
  - System.Drawing: Zeichenfunktionalität

Aufbau von Windows Formularen

- Forms
  - Ein Form-Objekt stellt ein Fenster in einer Applikation dar
  - Das Property BorderStyle entscheidet darüber wie ein Fenster erscheint:
    - *Standard*
    - *Tool*
    - *Borderless*
    - *Floating Window*
  - Forms können selbst Fenster enthalten = MDI (Multiple Document Interface)
  - Forms können auch modal (z.b.: als Dialogfenster) geöffnet werden
- Controls
  - Standard Controls wie Button, Label, Radiobutton, TextBox, ...
  - Custom Controls wie DataGrid, MonthCalendar
  - User Controls, welche der Benutzer selbst implementiert

Ereignisgesteuertes Anwendungsmodell

- GUI-Applikation wartet auf Ereignisse von:
  - Benutzern (Tastatur, Maus, ...)
  - Steuerelementen
  - Betriebssystem (Idle, ...)

- Klasse Application bietet statische Methoden für Steuerung einer Anwendung
  - Mit Run wird eine Form registriert und nimmt an Ereignisverarbeitung teil

```
public sealed class Application {
   static void Run(Form mainForm);
   static void Exit();
   static event EventHandler ApplicationExit;
   static event EventHandler Idle;
}
```

GUI-Ereignisse

- Control ändert seinen Zustand -> **Event**
- Registrierung eines EventHandler-Delegates bei der Ereignisquelle (Control)

```
public delegate void EventHandler( object sender, EventArgs e );
```

Beispiel: Registrierung für Button-Click:

```
Button b = new Button();
     b.Click += new EventHandler(clickHandler);
     ...
private void clickHandler(object sender, EventArgs evArgs) { ... }
```

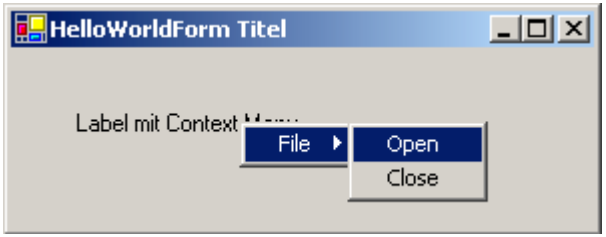
Beispiel: Menüs

- Anlegen eines Menüs für ein Form-Objekt:

```
MainMenu m = new MainMenu();
MenuItem mi = new MenuItem("&File");
mi.MenuItems.Add(new MenuItem("&Open"));
mi.MenuItems.Add(new MenuItem("&Close"));
m.MenuItems.Add(mi);
this.Menu = m;
```

- Anlegen eines Context-Menüs für ein Control

```
ContextMenu m = new ContextMenu();
MenuItem mi = new MenuItem("&File");
mi.MenuItems.Add(new MenuItem("&Open"));
mi.MenuItems.Add(new MenuItem("&Close"));
m.MenuItems.Add(mi);
label.ContextMenu = m;
```



Reichhaltige Bibliothek von Controls

- Es existiert bereits eine reichhaltige Bibliothek von Controls

Label

label1

Button

button1

CheckBox

checkBox1

CheckedListBox

checkedListBox1

ComboBox

RadioButton

radioButton1

ListBox

listBox1

GroupBox

groupBox1

Panel

TextBox

RichTextBox

WebBrowser

. . .

Entwicklung eigener Controls

School of Engineering

- Ähnling zu Java, einfach
- Von System.Windows.Forms.UserControl erben

- OnPaint (PaintEventArgs e) kann überschrieben werden
  - e.Graphics hält das Graphics Object
  - this.DisplayRectangle gibt die aktuelle Grösse an
  - Verwendung von System.Drawings Operationen zum Zeichnen
  - this.Refresh zeichnet neu

- Es können bestehende Controls mittels add hinzugefügt werden

- Anmeldung für Event Verarbeitung (siehe Delegates, Events)

GUI-Layout Gestaltung

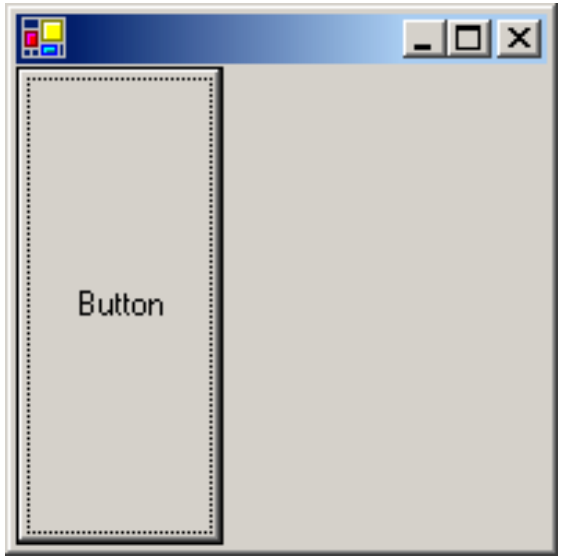
Drei verschiedene Arten:

- ***Docking***: Control bleibt bei Docking an ein Element angekoppelt.
- ***Anchor***: Abstand des Controls zum Container bleibt bei Anchor proportional gleich.
- ***Custom***: Implementierung eines eigenen Layout-Managers

Beispiel Dynamic Layout: Docking

- Anlegen eines Buttons, der links an seinen Container gedockt wird:

```
public class AnchorExample : Form {

  public AnchorExample() {
    InitializeComponent();
  }

  private void InitializeComponent() {
    this.Size = new Size(200,200);
    Button b = new Button();
    b.Text = "Button";
    b.Dock = DockStyle.Left;
    this.Controls.Add(b);
  }
```

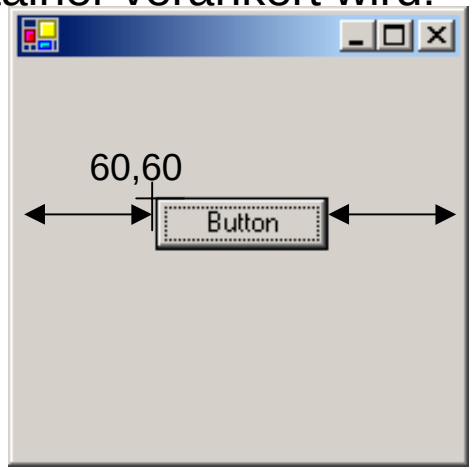


wie Java Borderlayout

Beispiel Dynamic Layout: Anchor

- Anlegen eines Buttons, der links und rechts an seinem Container verankert wird:

```
public class AnchorExample : Form {
    // ...
   private void InitializeComponent() {
     this.Size = new Size(200,200);
     Button b = new Button();
     b.Text = "Button";
     b.Location = new Point(60,60);
     b.Anchor = AnchorStyles.Left | AnchorStyles.Right;
     this.Controls.Add(b);
   }
   // ...
}
```

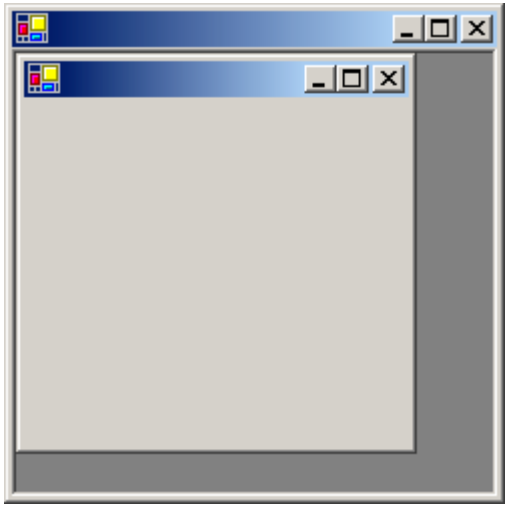


Distanz zum rechten Rand bleibt ebenfalls gleich (optional)

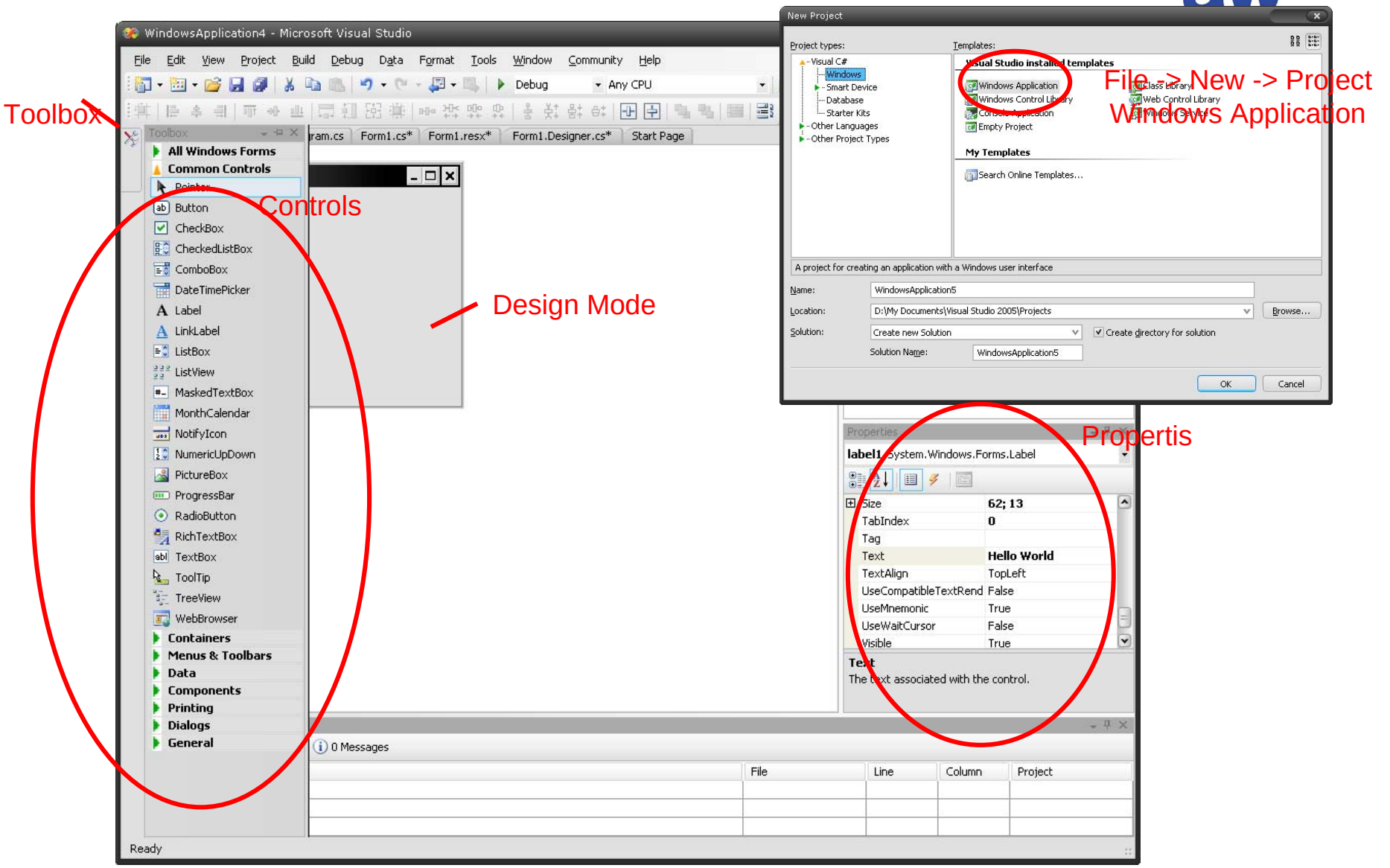
Multiple Document Interface

- Mehrere Forms in einem Fenster darstellen = MDI
- Property *IsMdiContainer* = *true* im Vaterfenster

```
class MdiForm : Form {

    private void InitComponent() {
            this.IsMdiContainer = true;
            this.Size = new Size(250,250);

            Form childForm = new Form();
            childForm.MdiParent = this;
            childForm.Size = new Size(200,200);
            childForm.Show();
    }
...
}
```



GUI Entwicklung mit Visual Studio

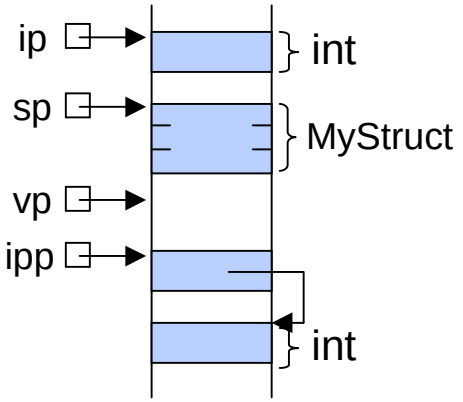


Zeiger
und Low-Level-Programmierung

Zeigertypen

Beispiele

```
int*      ip;   // Zeiger auf eine int-Zelle
MyStruct* sp;   // Zeiger auf ein MyStruct-Objekt
void*     vp;   // Zeiger auf eine beliebige Speicherzelle
int**     ipp;  // Zeiger auf einen Zeiger auf eine int-Zelle
```



Syntax

```
PointerType    = UnmanagedType *
               | void *.
UnmanagedType  = ValueType
               | PointerType.
```

Zeiger auf beliebigen Typ

Wenn Struct, müssen alle Felder von *UnmanagedType* sein

- Zeiger werden vom Garbage-Collector nicht verfolgt
- Zeigertypen sind nicht mit Object kompatibel
- Zeigervariablen können den Wert *null* haben
- Zeiger beliebigen Typs können miteinander verglichen werden (==, !=, <, <=, >, >=)

Unsafe Code

Code, der Zeigerverarbeitung betreibt, ist unsicher
(kann Typregeln brechen und Speicherbereiche zerstören)

- muss in unsafe-Block oder unsafe-Methode eingeschlossen werden

```
unsafe {
  int* p;
  ...
}
```

```
unsafe void foo() {
  int* p;
  ...
}
```

- muss mit unsafe-Option übersetzt werden

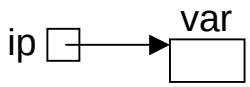
```
csc -unsafe MyProg.cs
```

- Administrator muss dem Code *FullTrust*-Rechte geben

Benutzung von Zeigern

Zeigerdereferenzierung

```
int var;
int* ip = &var;
```

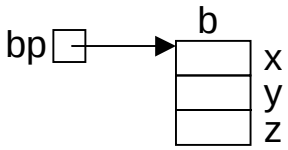



```
*ip = 5;
int x = *ip;
```

- wenn *v* vom Typ *T** ist, hat **v* den Typ *T*
- *void** kann nicht dereferenziert werden

Zugriff auf Struct-Felder

```
struct Block { int x,
               y, z; }
Block b;
Block* bp = &b;
```

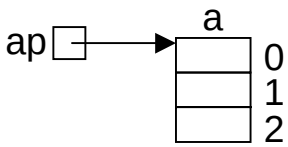



```
bp->x = 1;    // Pfeilschreibweise
*(bp).y = 2;  // alternative Schreibweise
```

- *bp* muss vom Typ *Block** sein
- geht auch für Methodenaufrufe

Zugriff auf Array-Elemente

```
int[] a = new int[3];
int* ap = a;
```



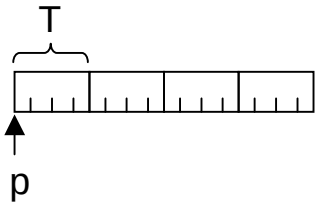

```
ap[0] = 1;    // Arrayschreibweise
*(ap+1) = 2;  // alternative Schreibweise
```

- keine Prüfung auf Indexüberschreitung!

Adressarithmetik

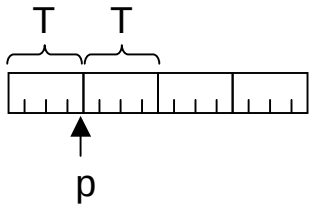
Mit Zeiger kann man rechnen

```
T* p = ...;
```

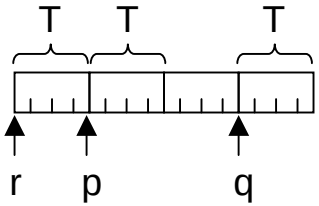



```
p++;
p--;
```

// erhöht p um size(T)




```
T* q, r;
q = p + 2; // q = p + 2*size(T)
r = p - 1; // r = p - size(T)
```



Mit *void** ist keine Zeigerarithmetik möglich

Type Casts mit Zeigern

```
int   i;
int*  ip;
Block*   bp;
void* vp;
```

Implizite Casts
(ohne Cast-Operator möglich)

T* ⇐ null	ip = null;
	bp = null;
	vp = null;

void* ⇐ T*	vp = ip; vp = bp;

Explizite Casts
(benötigt Cast-Operator)

T1* ⇔ T2*	ip = (int*)vp;
	vp = (void*)ip;
	bp = (Block*)vp;

T* ⇔ intType	ip = (int*)i;
	bp = (Block*)i;
	vp = (void*)i;
	i = (int)ip;
	i = (int)bp;
	i = (int)vp;

Fixierte und verschiebbare Variablen

Fixiert — Können vom Garbage-Collector nicht verschoben werden

- lokale Variablen
- value-Parameter
- Felder von Structs, die selbst fixiert sind

Verschiebbar — Können vom Garbage-Collector verschoben werden

- Felder von Klassen (auch statische Felder)
- Arrayelemente
- ref- und out-Parameter

Adressoperator

&designator liefert die Adresse von *designator*

Bedingungen

- *designator* muss eine fixierte Variable bezeichnen (z.B. *&x*, *&s.f*, *&a*[*i*])
- die Variable muss von einem Unmanaged-Typ sein (d.h. keine Klasse, kein Array
- wenn *designator* vom Typ *T* ist, ist *&designator* vom Typ *T**

fixed-Anweisung

Fixiert eine verschiebbare Variable für die Dauer der Anweisung
(d.h. der Garbage-Collector kann sie nicht verschieben)

Syntax

```
FixedStat      = "fixed" "(" PointerType   FixedVarDecl {"," FixedVarDecl} ")" Statement.
FixedVarDecl = (   "&" MoveableVar
                 |   ArrayVar
                 |   StringVar
               ).
```

MoveableVar muss von unmanaged-Typ *T* sein
*T** muss an *PointerType* zuweisbar sein

Arrayelemente müssen von unmanaged-Typ *T* sein
*T** muss an *PointerType* zuweisbar sein

*char** muss an *PointerType* zuweisbar sein

Beispiele

```
int x;
int[] a = new int[10];
Person person = new Person();
string s = "Hello";
```

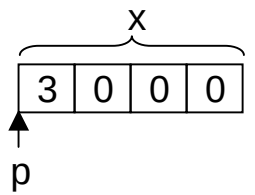
```
fixed (byte* p = (byte*) &x) {
Console.Write(*(p+1)); }
fixed (int* p = a) {...}
fixed (int* p = &a[1]) {...}
fixed (int* p = &person.id) {...}
fixed (char* p = s) {...}
```

Variablen, die im Header einer fixed-Anweisung deklariert werden, sind read-only

Anwendungsbeispiele

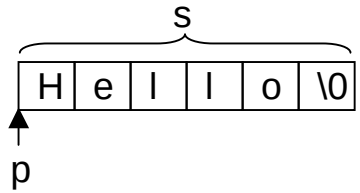
Bytes einer int-Variablen ausgeben

```
int x = 3;
unsafe {
  byte* p = &x;
  for (int i = 0; i < 4; i++) {
          Console.Write("{0:x2} ", *p); p++; }
}
```



String-Verarbeitung

```
string s = "Hello";
unsafe {
  fixed (char* p = s) {
    while (*p != '\0') { Console.Write(*p); p++; }
  }
}
```

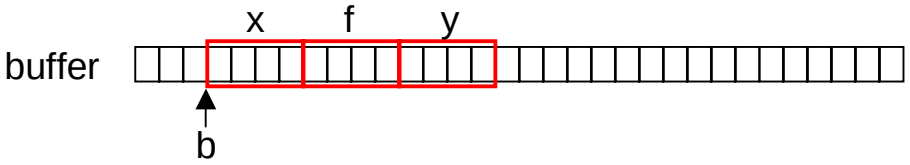


Anwendungsbeispiele (Fortsetzung)

School of Engineering

Struct über Byte-Array legen

```
struct Block {
  int x;
  float f;
  int y;
}
...
byte[] buffer = new byte[1024];
...
unsafe {
  fixed (byte* p = &buffer[3]) {
    Block* b = (Block*) p;
    Console.WriteLine(b->x + " " + b->f + " " + b->y);
  }
}
```



Gefahren der Zeigerverarbeitung

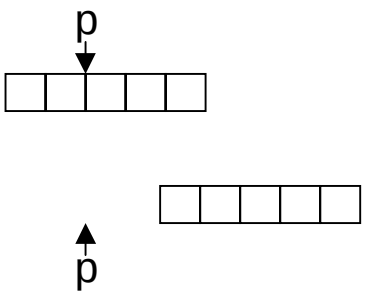
School of Engineering

Man kann beliebige Speicherbereiche zerstören

```
int[] a = new int[3];
unsafe {
  fixed (int* p = a) { p[5] = 0; }
}
```

Man kann Zeiger auf Objekte setzen, die der Garbage-Collector verschiebt

```
int[] a = new int[5];
int* p;
unsafe {
  fixed (int* q = a) { p = a+2; }
  ...
  ...     // GC kann Array-Objekt jetzt verschieben
  *p = 5;// zerstört fremden Speicher
}
```



Gefahren (Fortsetzung)

Man kann Zeiger auf lokale Variablen setzen, die später gar nicht mehr existieren

```
static unsafe int* Foo() {
  int local = 3;
  return &local;
}

static unsafe void Main() {
  int* p = Foo();
  ...
  *p = 5; // greift auf nicht mehr existierende lokale Variable zu!
}
```

Daher

Finger weg von der Zeigerverarbeitung!
Ausser man braucht sie unbedingt zur Systemprogrammierung

Native Calls

Aufruf einer Funktion aus einer WinDLL

School of Engineering

Unmanaged Code

Code der nicht von der CLR verwaltet wird

Signatur der aufzurufenden Win32-Funktion

```
int MessageBox (HWND hWnd, LPTSTR lpText, LPTSTR lpCaption, UINT uType);
```

Aufruf aus C#


```
using System;
using System.Runtime.InteropServices;

class Test {

   [DllImport("user32.dll")]
   static extern int MessageBox(int hWnd, string text, string caption, int type);

   static void Main() {
      int res = MessageBox(0, "Isn't that cool?", "", 1);
      Console.WriteLine("res = " + res);
   }
}
```

http://www.pinvoke.net/

DllImport-Attribut

```
[DllImport ( dll.name
           [, EntryPoint=function-name]
           [, CharSet=charset-enum]         // für String-Marshaling
           [, ExactSpelling= (true|false)]  // Name entsprechend CharSet modifizieren?
           [, SetLastError= (true|false)]   // true: liefert Win32-Fehlerinfo
           [, CallingConvention=callconv-enum]
           )]
```

charset-enum: CharSet.Auto (default), CharSet.Ansi, CharSet.Unicode, ...

callconv-enum: CallingConvention.StdCall (default), CallingConvention.FastCall, ...

Standard Mapping C++ zu CTS,C# Typen

wtypes.h	C++	Common language runtime	C#
HANDLE	void *	IntPtr, UIntPtr	int
BYTE	unsigned char	Byte	byte
SHORT	short	Int16	short
WORD	unsigned short	UInt16	short
INT	int	Int32	int
UINT	unsigned int	UInt32	uint
LONG	long	Int32	int
BOOL	long	Boolean	bool
DWORD	unsigned long	UInt32	uint
ULONG	unsigned long	UInt32	uint
CHAR	char	Char	char
LPSTR	char *	String [in], StringBuilder [in, out]	string [in], StringBuilder [in, out]
LPCSTR	const char *	String	string
LPWSTR	wchar_t *	String [in], StringBuilder [in, out]	string [in], StringBuilder [in, out]
LPCWSTR	const wchar_t *	String	string
FLOAT	float	Single	string
DOUBLE	double	Double	double

Parameter-Marshaling

Standardtypen von C# werden automatisch auf entsprechende Win32-Typen abgebildet.

Standard-Abbildung kann folgendermassen übersteuert werden

```
using System.Runtime.InteropServices;
...

[DllImport("...")]
static extern void Foo (
   [MarshalAs(UnmanagedType.LPStr)] string s,
   int x
);
```

ANSI-String aus Bytes

LPStr	ANSI-String aus einzelnen Bytes
LPWStr	Unicode-String
LPTStr	Default: Windows XP/NT/2000 => Unicode, Windows 98 => ANSI

Übergabe von Klassen und Structs

```
using System.Runtime.InteropServices;

[StructLayout(LayoutKind.Sequential)]// Members werden sequentiell angelegt
class Time {                         // und dicht gepackt (default)
   public ushort year;
   public ushort month;
   ...
}

[StructLayout(LayoutKind.Explicit)]// Members werden an spezifizierte
struct S {                         // Offsets gepackt
   [FieldOffset(0)] ushort x;
   [FieldOffset(2)] int y;
}

[DllImport("...")] static extern void Foo(Time t);
[DllImport("...")] static extern void
Foo1([MarshalAs(UnmanagedType.LPStruct)] Time t);

[DllImport("...")] static extern void Bar(S s);

Foo(new Time());

Bar(new S());
```

StructLayout-Attribut

```
[StructLayout  ( layout-enum                    // Sequential | Explicit | Auto
               [, Pack=packing-size]            // 0, 1, 2, 4, 8, 16, ...
               [, CharSet=charset-enum]         // Ansi | Unicode | Auto
               )]
```

COM Interoperabilität

OLE/COM/ActiveX

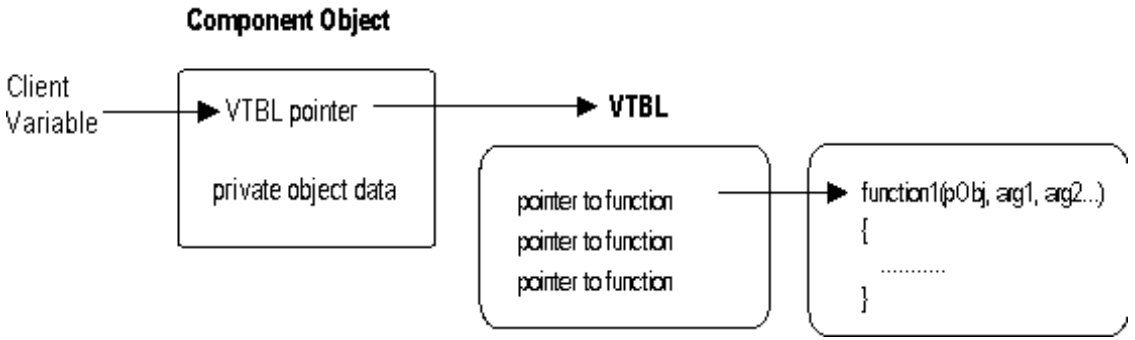
- COM (Component Object Model)
  - *OLE (Object Linking and Embedding) : alter Namen*
  - *ActiveX: visuelle Komponenten (Internet)*
  - Technologie von Microsoft, die es ermöglicht, dass Anwendungen aus Komponenten verschiedener Hersteller interoperieren (sich gegenseitig aufrufen) können.
  - Aktuelle Basis für die Interoperabilität der Office Palette (auch 2007) als auch des Betriebssystems (auch Win 7)
    - *z.B. Excel Spreadsheet in Word Dokument*

- Zu lösende Aufgaben
  - Sicherstellung der **gegenseitigen Aufrufbarkeit** von Komponenten verschiedener Hersteller
  - **Programmiersprachenunabhängigkeit**
  - **Dynamische** (für Script Sprachen) und **statische Typisierung** (für übersetzte Sprachen)
  - **Versionierung** der Komponenten, da unabhängige Lebenszyklen
  - **In-Process**, **Interprocess** und ev. **Remote Aufrufrufbarkeit** sein

http://msdn.microsoft.com/library/default.asp?URL=/library/techart/msdn_comppr.htm

Lösungen in COM

- Interoperabilität: Binärer Standard für den Aufruf und Typen
  - basierend auf C++ Methoden: VTBL (Virtual Function Table)
  - andere Programmierprachen müssen C++/VTBL Modell unterstützten
  - Neues gemeinsames Typensystem -> Abbildung in Prog. Sprache.
    - *Numerische Typen: INT, LONG, UCHAR*
    - *BSTR für String: ("beasters")*
    - *VARIANT für void**
  - Schnittstellen werden in separaten TLB-Dateien (oder in DLL eingebettet) beschrieben (Typelibrary)

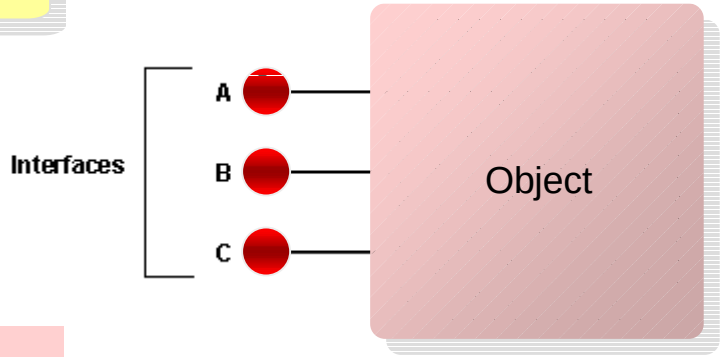


Grundlegendes

- COM Komponenten können nur über Interfaces angesprochen werden.
  - Interface = eine Klasse die von IUnknown erbt (# Interface in C#, C++)
    - *Funktionen für Referenz-Zähl Speicherverwaltung: AddRef & Release*
    - *Funktion um weitere Interfaces der Komponenten zu finden: QueryInterface*
  - Interfaces werden über Global Unique ID (GUID) identifiziert
    - *GUID: Eindeutig über "Raum und Zeit"*
    - *bsp: {c4910d71-ba7d-11cd-94e8-08001701a8a3}*
  - fast alle Methoden liefern i.A. HRESULT zurück

C++

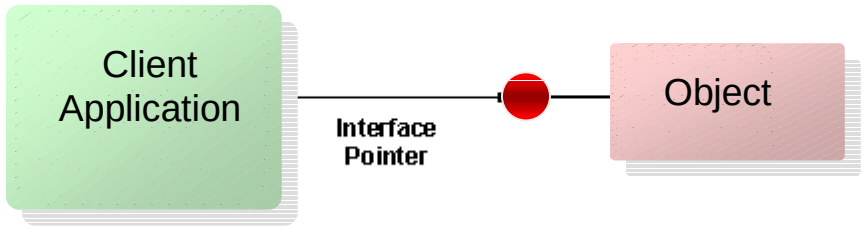
```
class IUnknown {
  public:
  virtual HRESULT QueryInterface(IID& iid, void** ppvObj) = 0;
  virtual ULONG AddRef() = 0;
  virtual ULONG Release() = 0;
}
```



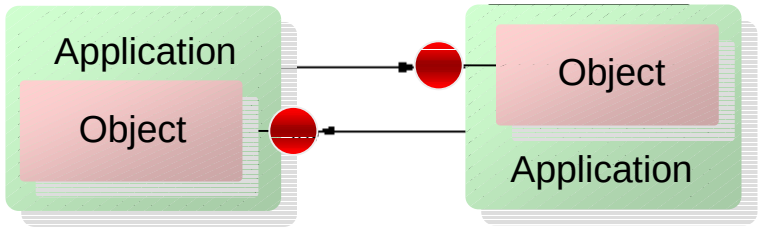

```
class ICalc : public IUnknown {
  public: virtual HRESULT __stdcall Add( int a, int b, int* res) = 0;
}
```

Aufruf

- Der Aufruf geht immer über das Interface
  - 1) finde passendes Interface der Komponente mittels QueryInterface
  - 2) rufe die Methode des Interfaces auf
    - *Interface delegiert den Aufruf an eigentliches "Object"*



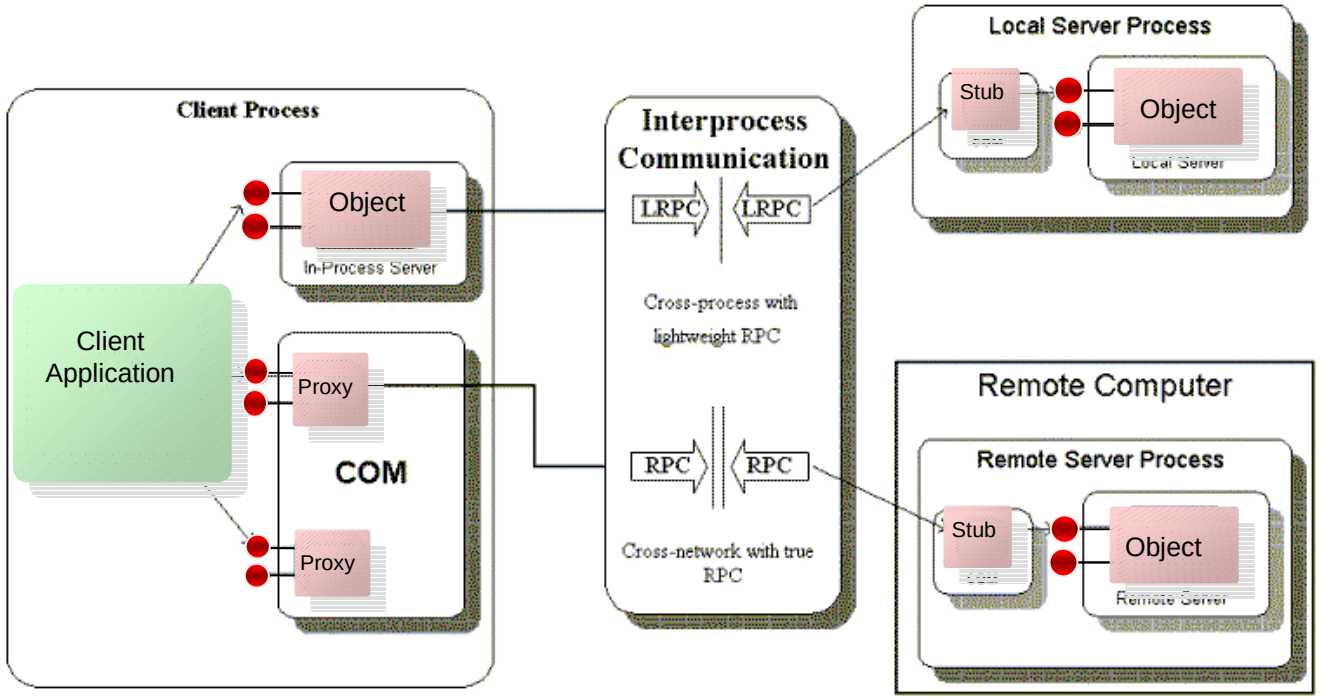
- Anwendungen können sich so auch gegenseitig aufrufen



- Interfaces sind nicht veränderbar

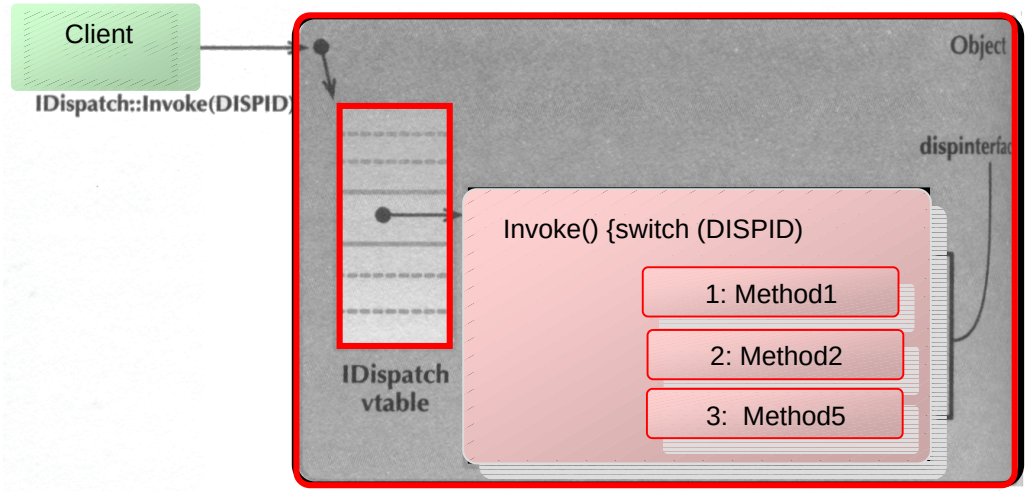
Aufrufmodi

- Verschiedene Aufrufmodi möglich
  - **In-Process**: gleicher Prozess
  - **Local Server**: selbe Maschine anderer Prozess
  - **Remote Server**: andere Maschine (DCOM)



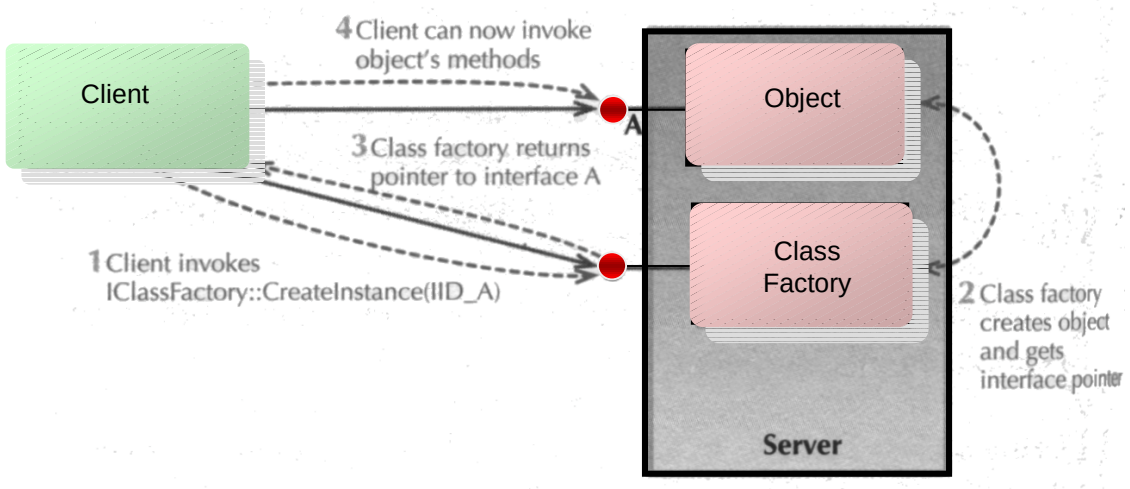
Automation und IDispatch

- Aufruf der Methoden wenn Namen bekannt
  - späte Bindung (nicht kompilierte Sprachen, Scripting)
- Entspricht Aufruf via Reflection Mechanismus in C#
- Spezielle für Scriptingsprachen geeignet (VBA)
- Im Wesentlichen zwei Methoden
  - **GetIDsOfNames:** liefert eine ID der Methode anhand Namen
  - **Invoke:** ruft Methode mit beliebigen Argumenten auf



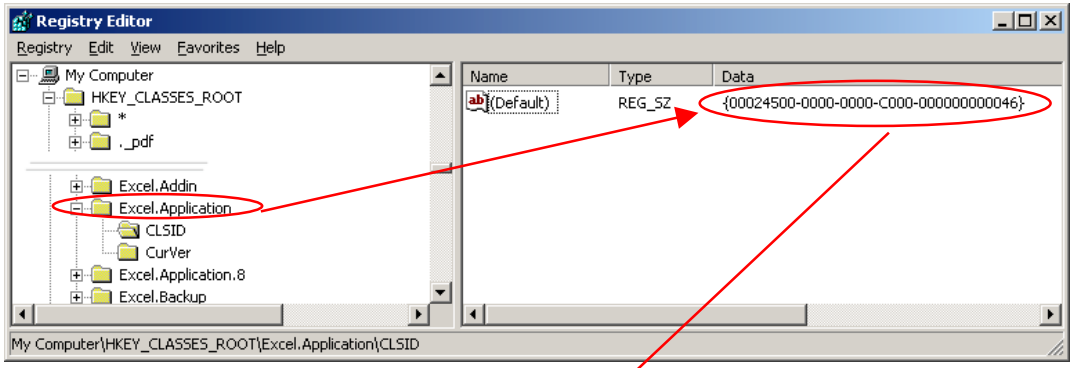
Laden der COM Komponente

- Werden als DLL,OCXs oder EXE implementiert
- bei EXE
  - starte EXE (wenn noch nicht gestartet) und warte bis sie sich mittels **CoRegisterClassFactory** registriert hat
- bei DLLs oder OCXs
  - DLL muss Eingangspunkt **DllGetClassFactory** haben
  - liefert Klasse die **IClassFactory** mit Methode **CreateInstance** implementiert

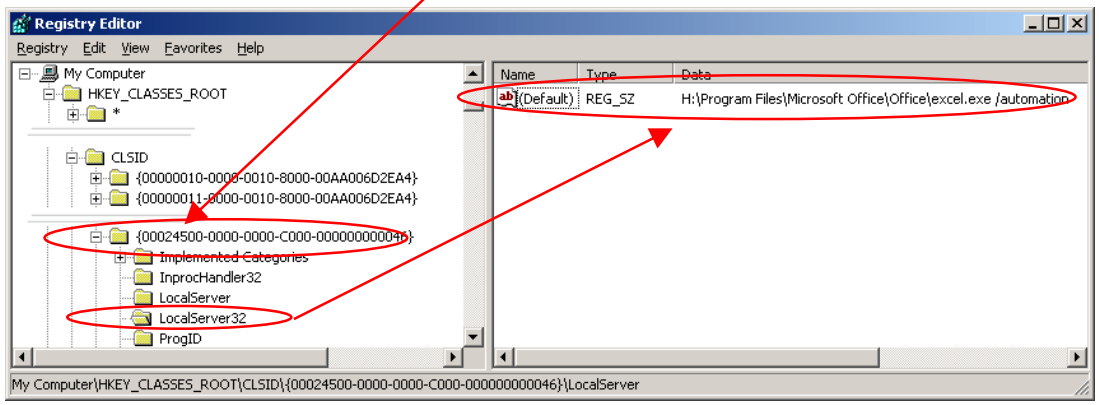


Registry als Verzeichnisdienst für COM

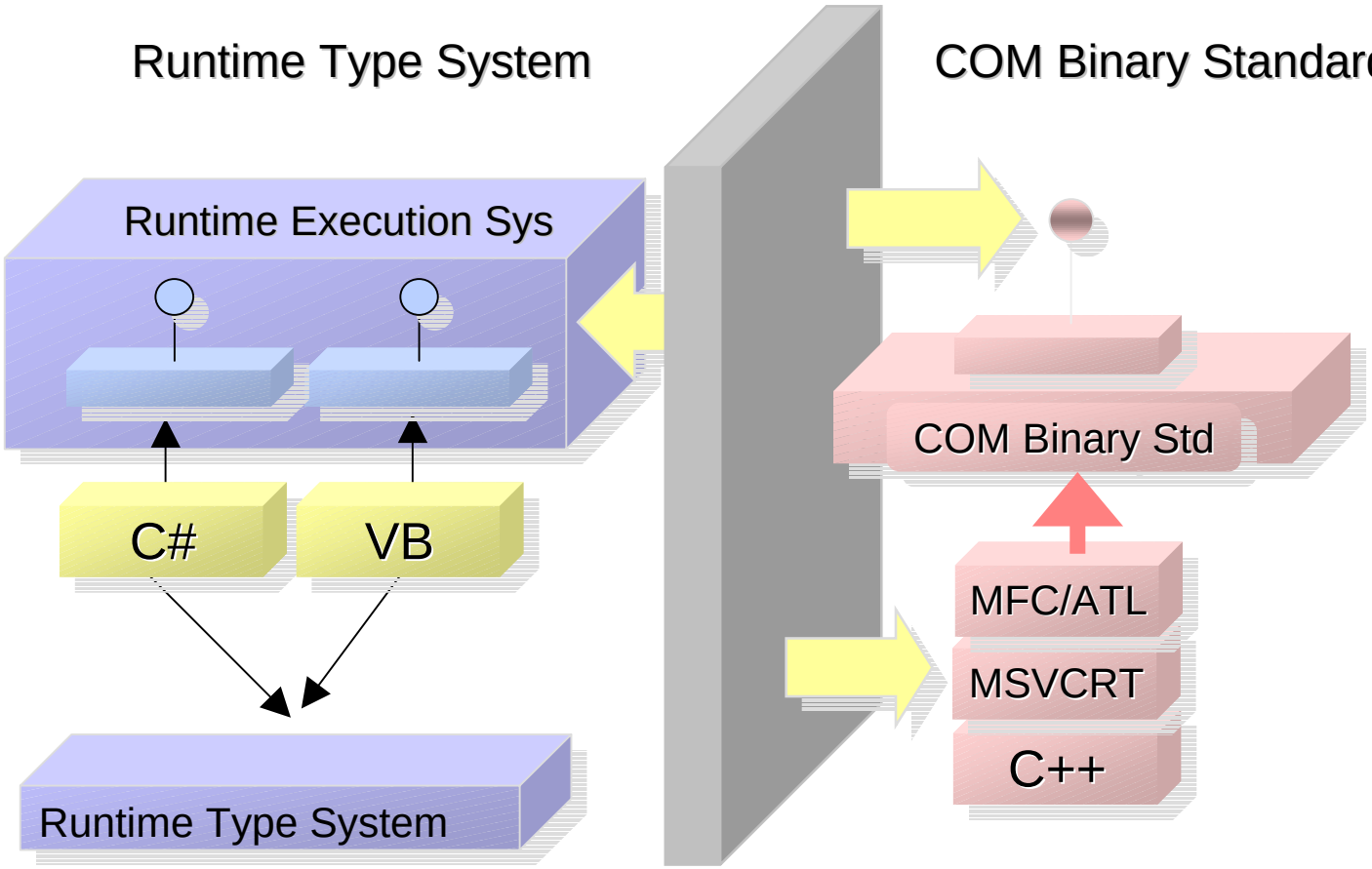
1) Suche Namen in Registry: HKCR
2) bestimme CLSID



3) Mit CLSID suche in HKCR/CLSID Ast die Implementation
4) Lade die Implementation (LocalServer32)



Managed und COM Welt



Zwei sehr unterschiedliche Welten

Entwicklungszeit

.Net Model

- Metadata
- Resilient Bind
- Assemblies
- Object based
- Exceptions
- Strong Names

COM Modell

- Type Libraries
- Immutable Types
- DLLs and EXEs
- Interface based
- HResults
- Guids

Laufzeit

.Net Model

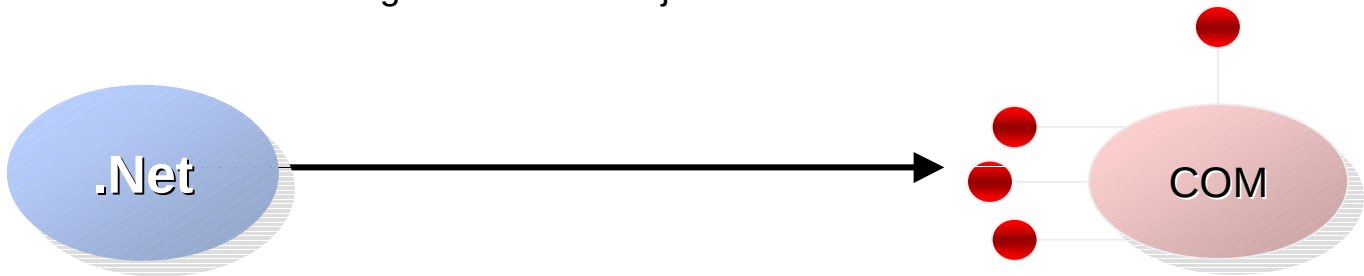
- *new* operator
- Cast operator
- Memory mgmt
- Exceptions

COM Model

- CoCreateInstance
- QueryInterface
- Reference Counting
- HResult

Bi-Directional COM Interop

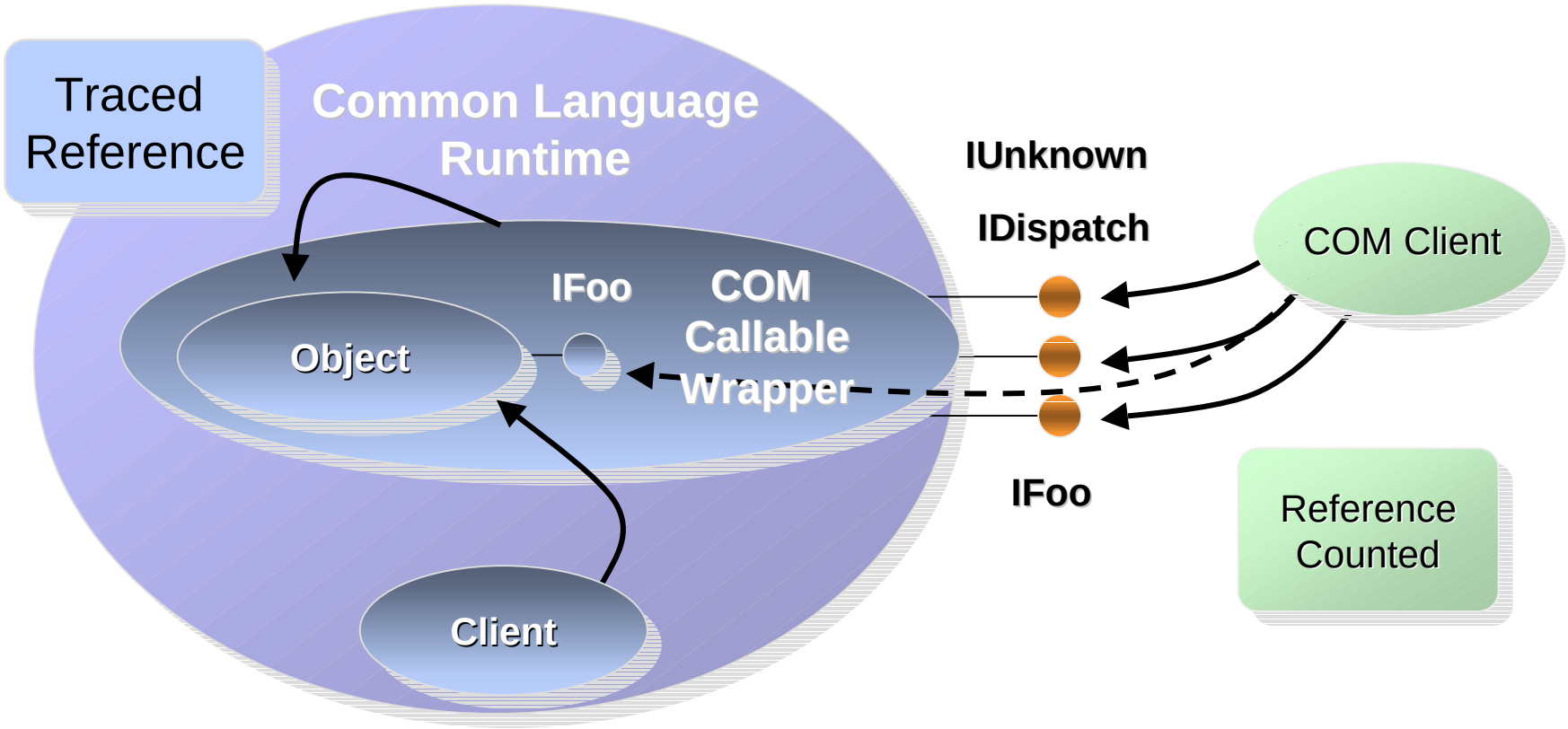
- .Net zu COM
  - Erlaubt von .NET aus Zugriff auf COM Object



- COM zu .Net
  - Ermöglicht von COM aus Zugriff auf .NET Objekte

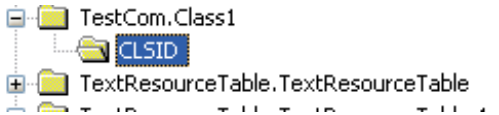
Erstellen einer COM Komponente in C#

COM To .Net Interop: CCW



Erstellen einer COM Komponente in C#

- Eine COM Komponenten ist eine C# Klasse mit [ComVisible(true)]
- die GUId definiert den Eintrag in Registry

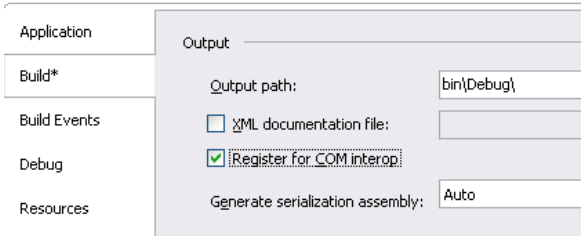



```
using System;
using System.Collections.Generic;
using System.Text;
using System.Runtime.InteropServices;


namespace TestCom {
    [Guid("EF00FB61-2FD8-4ae1-889A-B1254B08F8DC")]
    [IDispatchImpl(IDispatchImplType.CompatibleImpl)]
    [ComVisible(true)]
    public class Class1 {

        public int hello() {
            return 42;
        }
    }
}
```

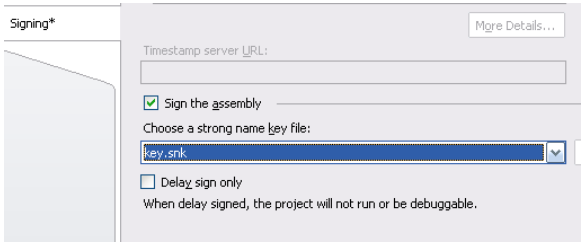
Erstellen einer COM Komponente in C#

School of Engineering

- 1. Funktion der Entwicklungsumgebung
  - Wähle Class Library Projekt
  - Konfiguration unter Properties/ Build Seite
    - *Wähle "Register for COM Interop" aus*
    - *oder später von Hand mit regasm (3 Punkt)*

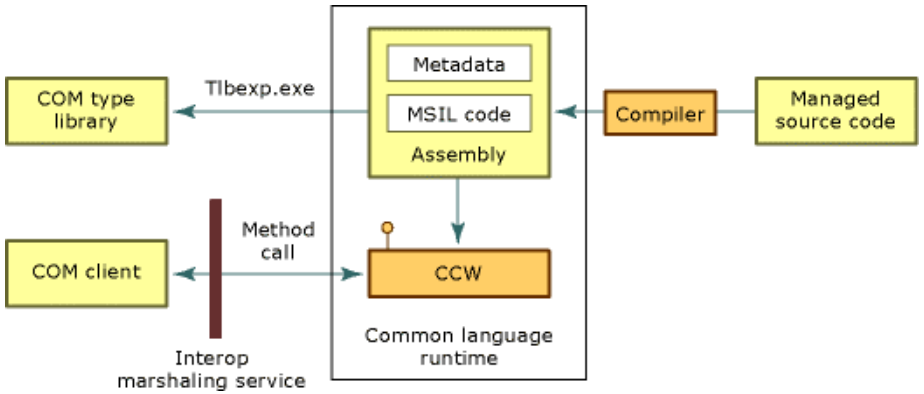


- 2. COM Assembly Erzeugen
  - Erzeuge eine .NET DLL mit starkem Namen
    `sn -k key.snk`
  - AssemblyInfo.cs: `[assembly: AssemblyKeyFile("key.snk")]`



- 3. Eintragen der COM Assembly in Registry
  `regasm <.NET DLL> /codebase`
  ev. `tlbexp <.NET DLL>` wenn TLB benötigt wird

- 4. Rest übernimmt der CCW



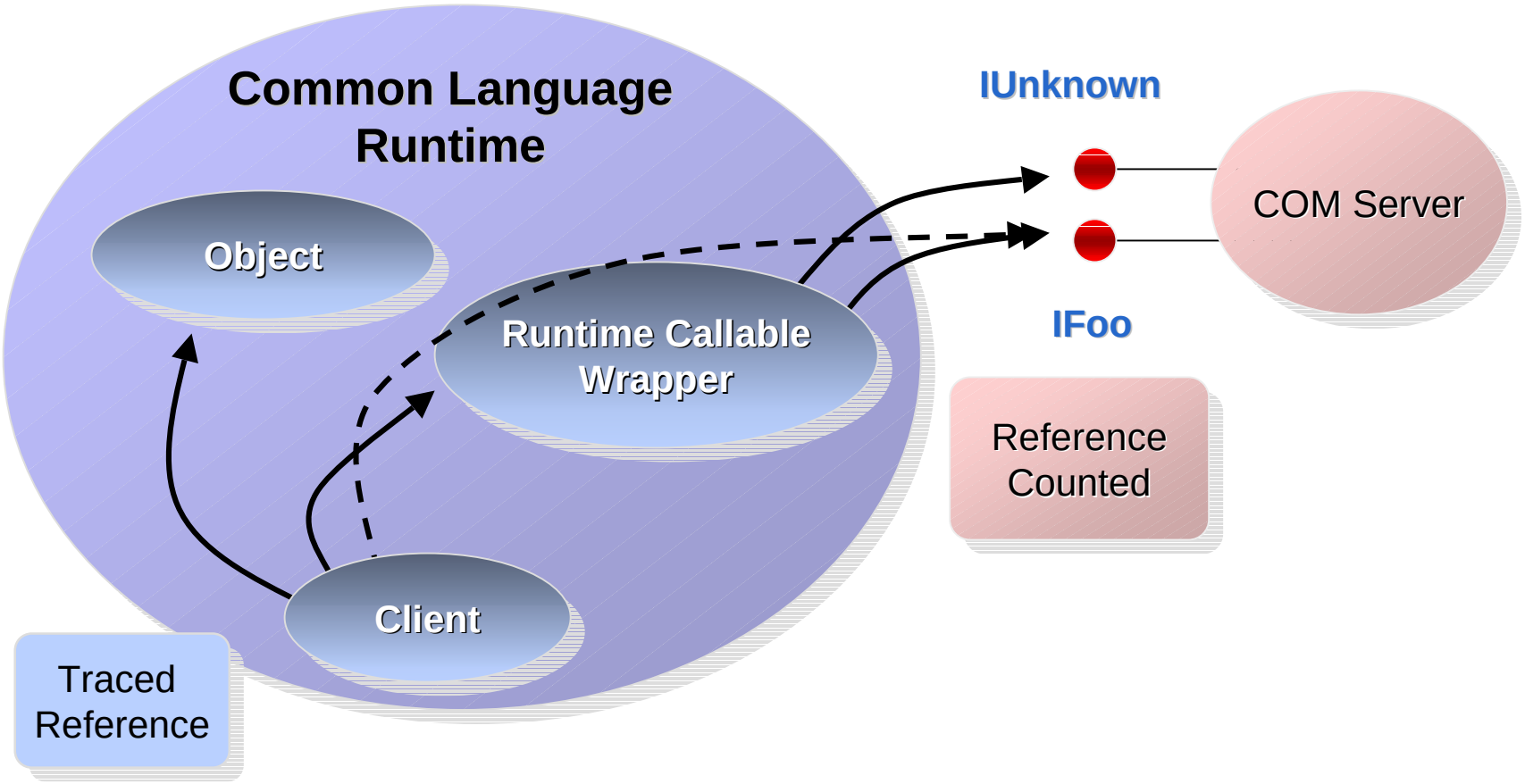
Verwendung in z.B. Excel

- Dieses Objekt kann dann z.B. in Excel mittels VBA folgendermassen angesprochen werden

```
Public Sub Test()
    Dim Test As Object
    Dim i As Integer
    Set Test = CreateObject("TestCom.Class1")
    i = Test.hello
    ActiveSheet.Cells(1, 1) = i
End Sub
```

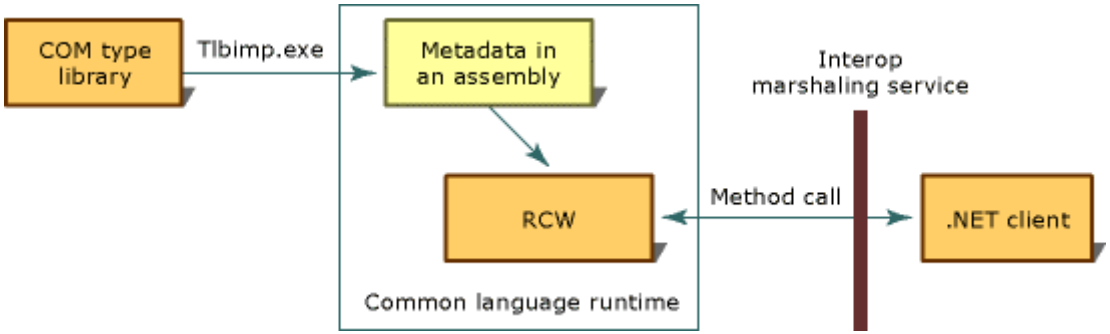
Verwenden einer COM Komponente in .NET

.NET zu COM Interop



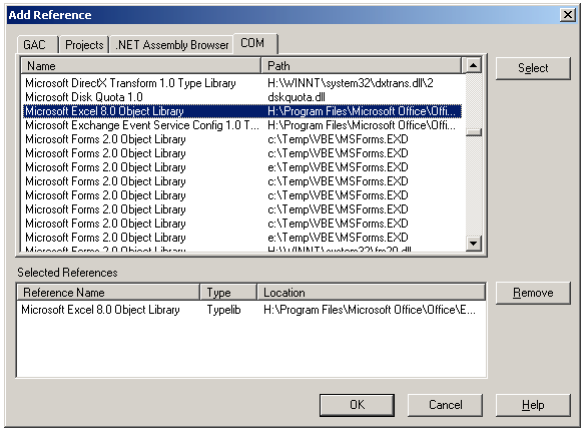
Verwendung von COM in .NET: RCW

- Beschaffe eine *Interop Assembly* welche die COM Typen enthält
- Primary Interop Assembly (PIA)
  - Erste Wahl wenn verfügbar (MSDN)
  - Enthält auch spezifische Anpassungen
- Erzeuge eine eigene
  - Hinzufügen einer Referenz in Entwicklungswerkzeug
  - Unter Verwendung des TLB Importierers (**TlbImp)**
  - Bei graphischen (**ActiveX**) Verwendung von **AxImp**
  - Von Hand



Beispiel: Steuerung von Excel

- Erzeuge Wrapper Klasse für Excel Object Library
- Entweder via tlbimp oder VS
  - Reference in Entwicklungstool und importiere
    ```
    using Excel;
    ```
  - Füge Code Hinzu




```
Console.WriteLine ("Creating new Excel.Application");
System.Threading.Thread.CurrentThread.CurrentCulture
            = new System.Globalization.CultureInfo("en-GB");
Application app = new Application();
if (app == null) {
  Console.WriteLine("ERROR: EXCEL couldn't be started!");
  return 0;
}
app.Visible = true; // Making application visible"
Workbooks workbooks = app.Workbooks; // Getting the workbooks collection
// The following line is the temporary workaround for the LCID problem
Workbook workbook = workbooks.Add(XlWBATemplate.xlWBATWorksheet);

Excel._Worksheet  oSheet = (Excel._Worksheet)workbook .ActiveSheet;
//Add table headers going cell by cell.
oSheet.Cells[1, 1] = "First Name";
oSheet.Cells[1, 2] = "Last Name";
oSheet.Cells[1, 3] = "Full Name";
oSheet.Cells[1, 4] = "Salary";

app.Quit;
```

Je nach Version Fix Bug in Excel Wrapper

c:\Program Files\Microsoft.NET\SDK\v1.1\Samples\Technologies\Interop\Applications\Office\Excel

Beispiel: Fernsteuerung von Merlin

- Erzeuge Wrapper Klasse für Agent COM Object

```
H:\WINNT\msagent>tlbimp agentctl.dll AgentObjects.dll
```

- Kopiere erzeugte dll in Project Directory
- Referenziere (in Tool) und importiere dll

```
using AgentObjects;
```

Oder in VS References
->Add Reference
->COM
->Microsoft Agent Control

- Definiere und initialisiere Agent

```
Agent agent = new AgentObjects.Agent);
agent.Connected = true;
string name =  "Merlin";
agent.Characters.Load(name, name+".acs");
IAgentCtlCharacterEx merlin = agent.Characters[name];
```

- Führe Operationen aus.

```
merlin.Show(false);
merlin.MoveTo(300, 300, 1000);
merlin.Play("Greet");
merlin.Speak("Hallo, I am "+name,null);
merlin.Play("Congratulate");
merlin.Speak("I am an ActiveX control programmed in \Map=seesharp=C#\!", null);
merlin.Play("Announce");
...
merlin.Hide(false);
```

Acknowledge Alert Announce Blink Confused Congratulate Decline DoMagic1 DontRecognize Explain GestureDown GestureLeft GestureRight GestureUp GetAttention GetAttentionContinued GetAttentionReturn Greet Hearing_1 Hide Idle1_1 Idle2_1 Idle3_1 LookDown LookLeft LookRight LookUp MoveDown MoveLeft MoveRight MoveUp Pleased Process Processing Read ReadContinued ReadReturn Reading RestPose Sad Search Searching Show StartListening StopListening Suggest Surprised Think Thinking Uncertain Wave Write WriteContinued WriteReturn Writing

Beispiel: Einbettung von Browser

- Erzeuge Wrapper Klasse für IE-ActiveX

```
H:\WINNT\system32>AxImp shdocvw.dll
                       /out:c:\tmp\axshdocvw.dll
tlbimp MSHTML.TLB /out:c:\tmp\mshtml.dll
```

- Kopiere erzeugte dlls in Project Directory
- Referenziere und importiere dll

```
using AxSHDocVw;
using mshtml;
```

- Definiere und initialisiere Browser

```
private AxWebBrowser axWebBrowser1;
....
this.axWebBrowser1 = new AxWebBrowser();
((System.ComponentModel.ISupportInitialize)(this.axWebBrowser1)).BeginInit();
this.axWebBrowser1.Enabled = true;
this.axWebBrowser1.Location = new System.Drawing.Point(10, 10);
this.axWebBrowser1.Size = new System.Drawing.Size(200, 200);
this.axWebBrowser1.TabIndex = 2;
this.Controls.Add(this.axWebBrowser1);
```

- Navigiere zu Seite

```
object o= null;
axWebBrowser1.Navigate(textBox1.Text,ref o,ref o,ref o,ref o);
```

In VS gibt es auch
ein fertiges WebBrowser
Control in der Toolbox
Vorsicht: andere Schnittstelle!!

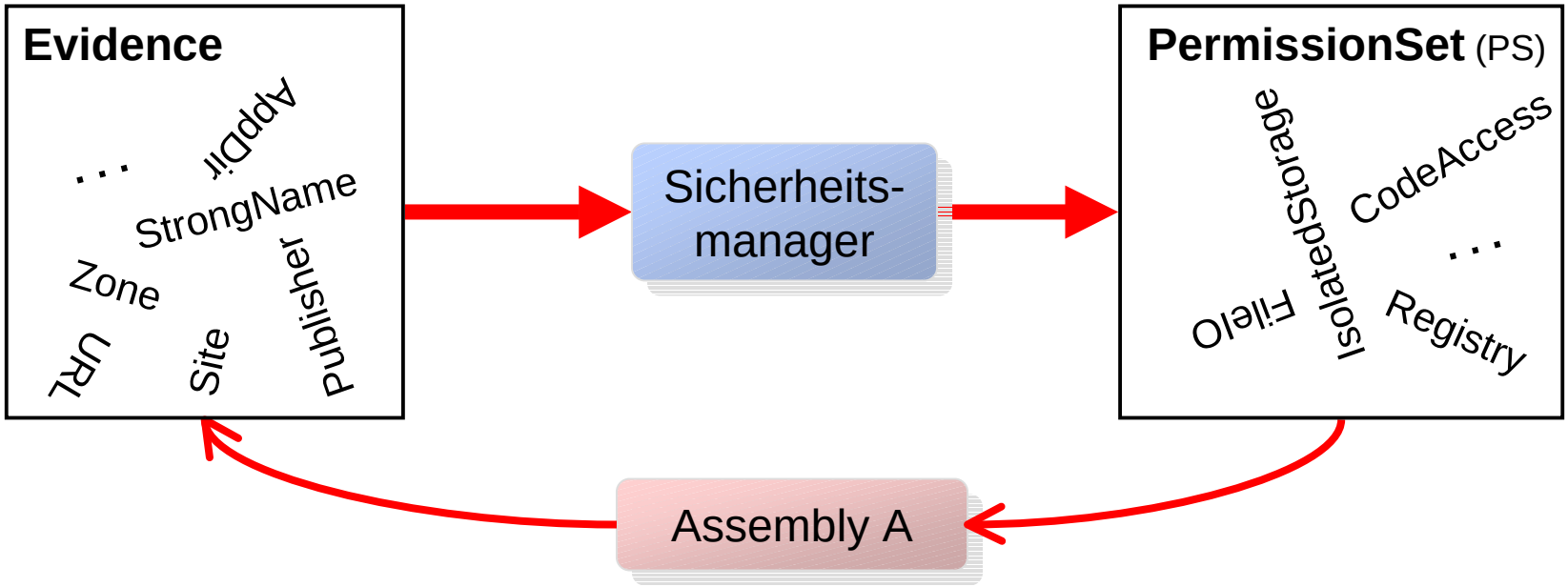
Sicherheit unter .NET

Sicherheit unter .NET

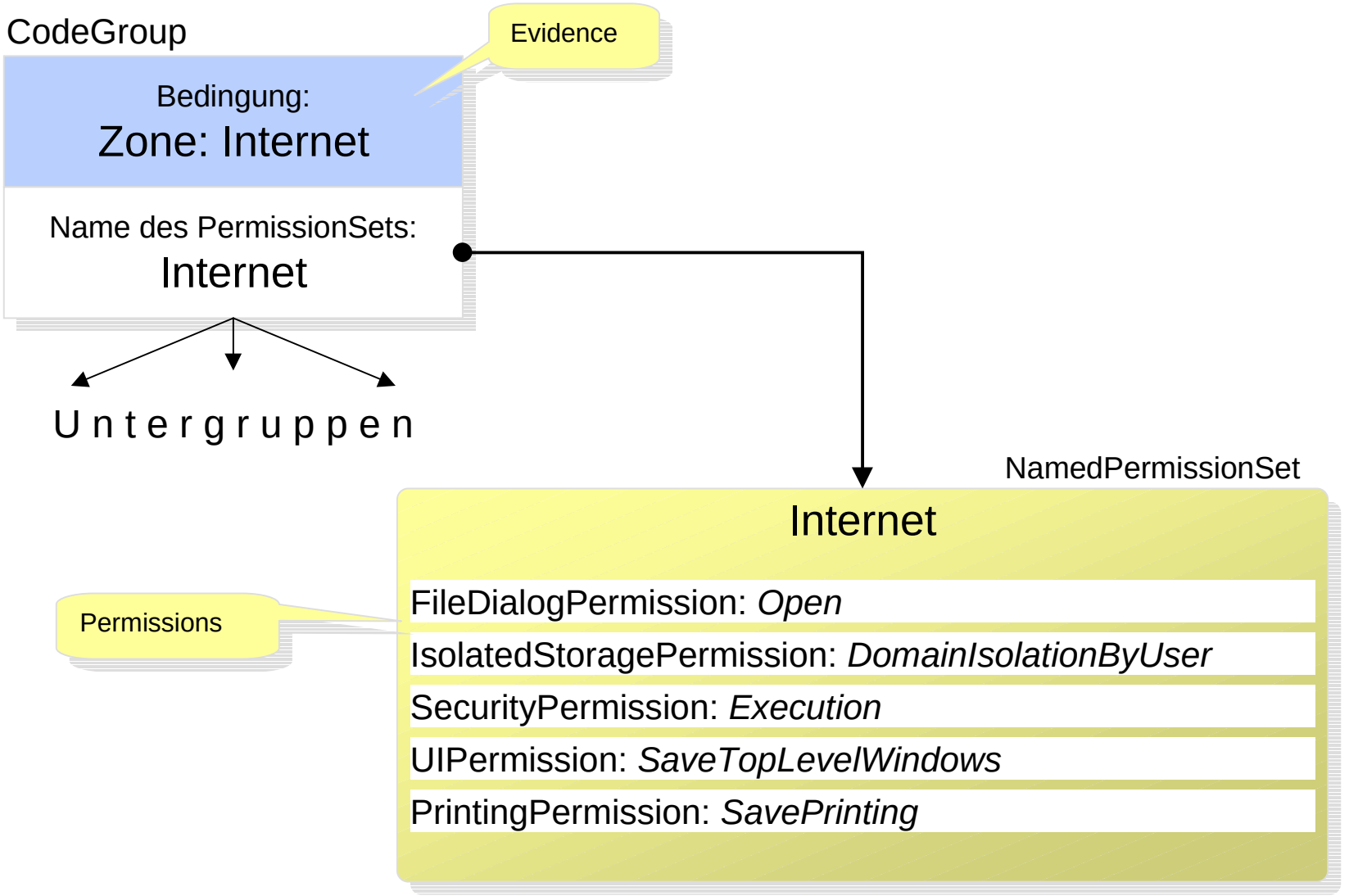
- **Rollenbasierte Sicherheit** (Teil des Betriebssystems)
  - .NET Zugriffsklassen
  - Rechte werden an ***Benutzer*** vergeben
  - abhängig von ***Gruppenzugehörigkeit***
  - Namensraum System.Security.Principal bietet unterstützende Interfaces an:
    - *System.Security.Principal.IPrincipal = Rolle, Gruppe*
    - *System.Security.Principal.IIdentity = Benutzer*

- **Codebasierte Sicherheit** (*Code Access Security (CAS)*)
  - Rechte werden an ***Assemblies*** vergeben
  - abhängig von ***Assemblyinformationen*** (*evidence*) und ***Sicherheitspolitik***

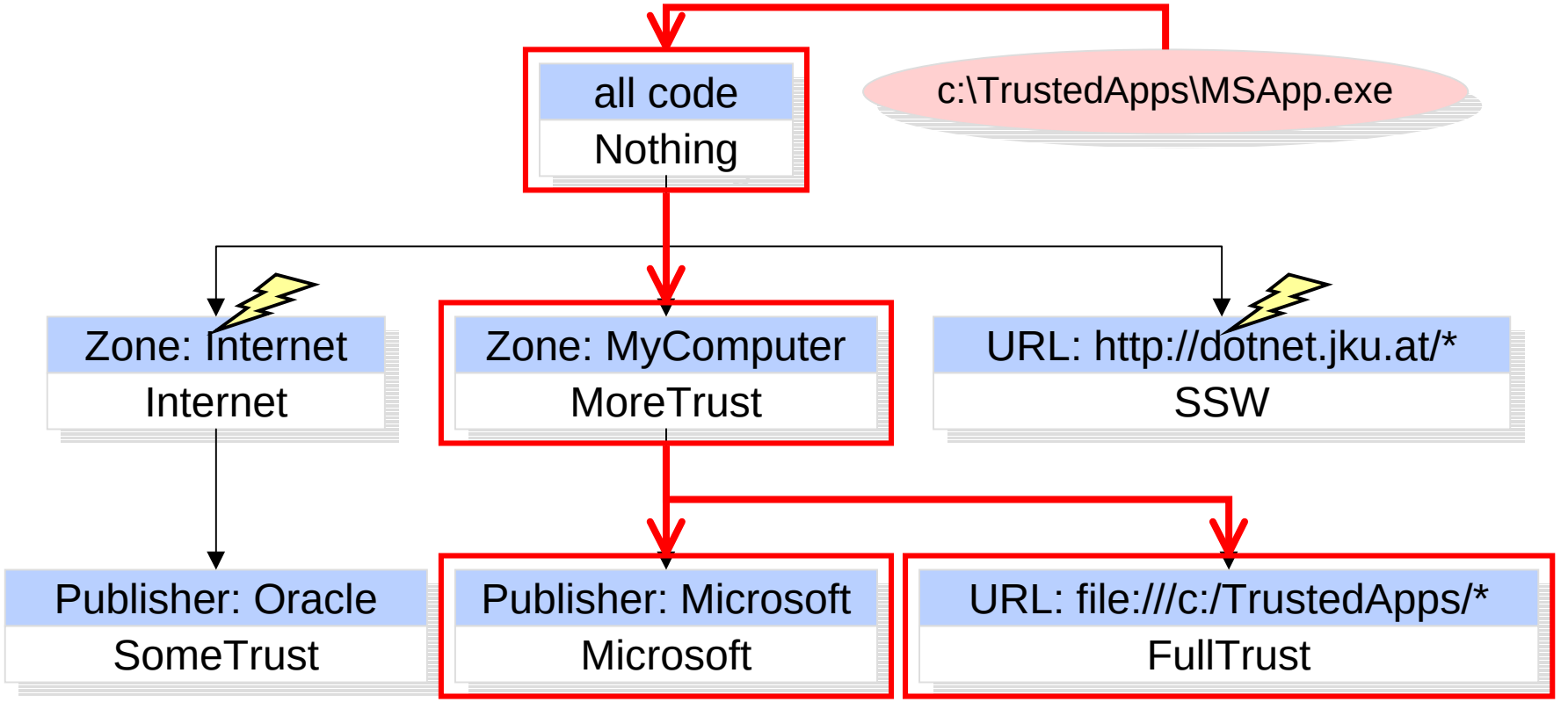
Code Access Security (CAS)



Codegruppen & Genehmigungsmengen

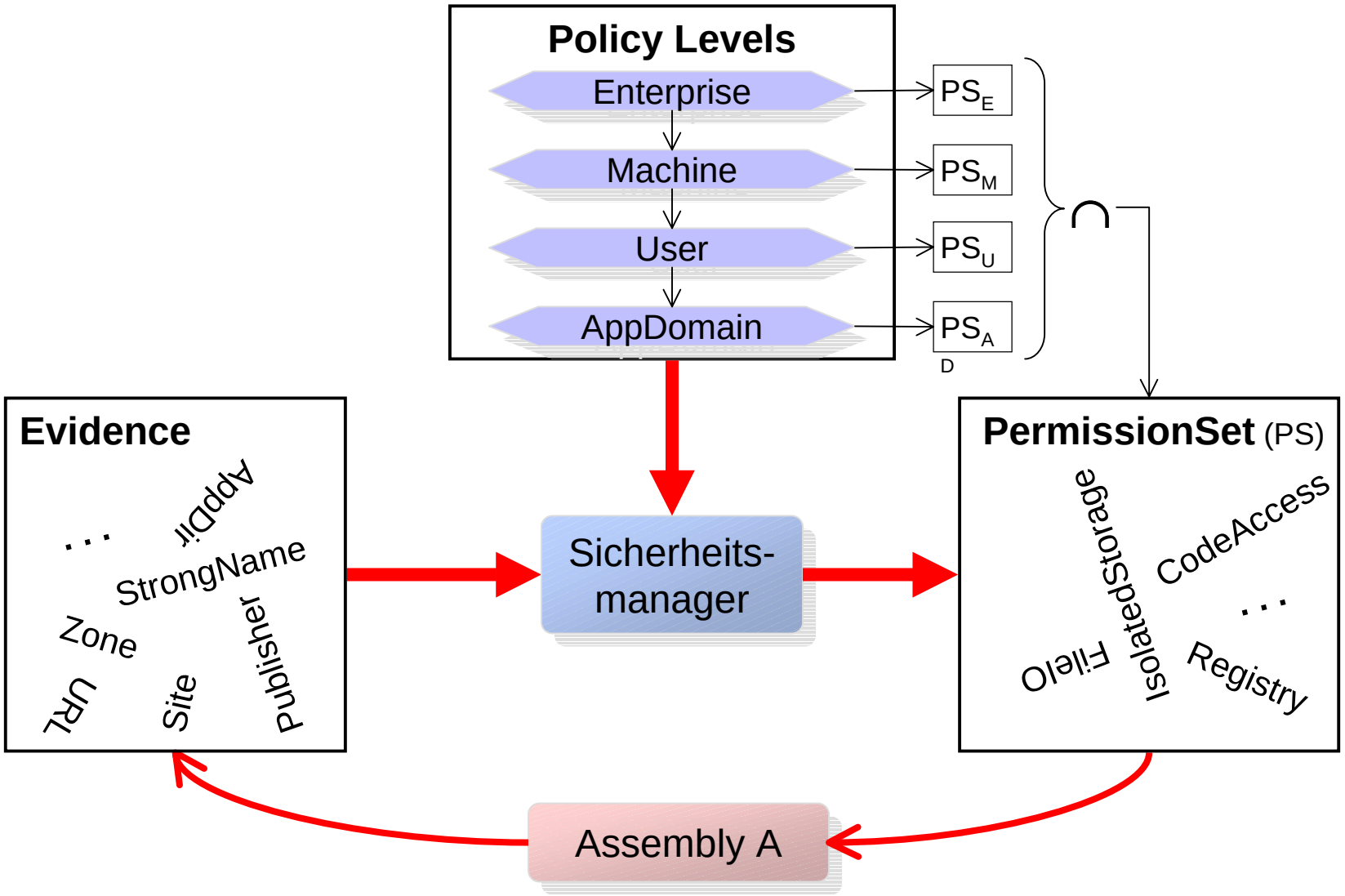


Ebenen der Sicherheitspolitik (Policy)



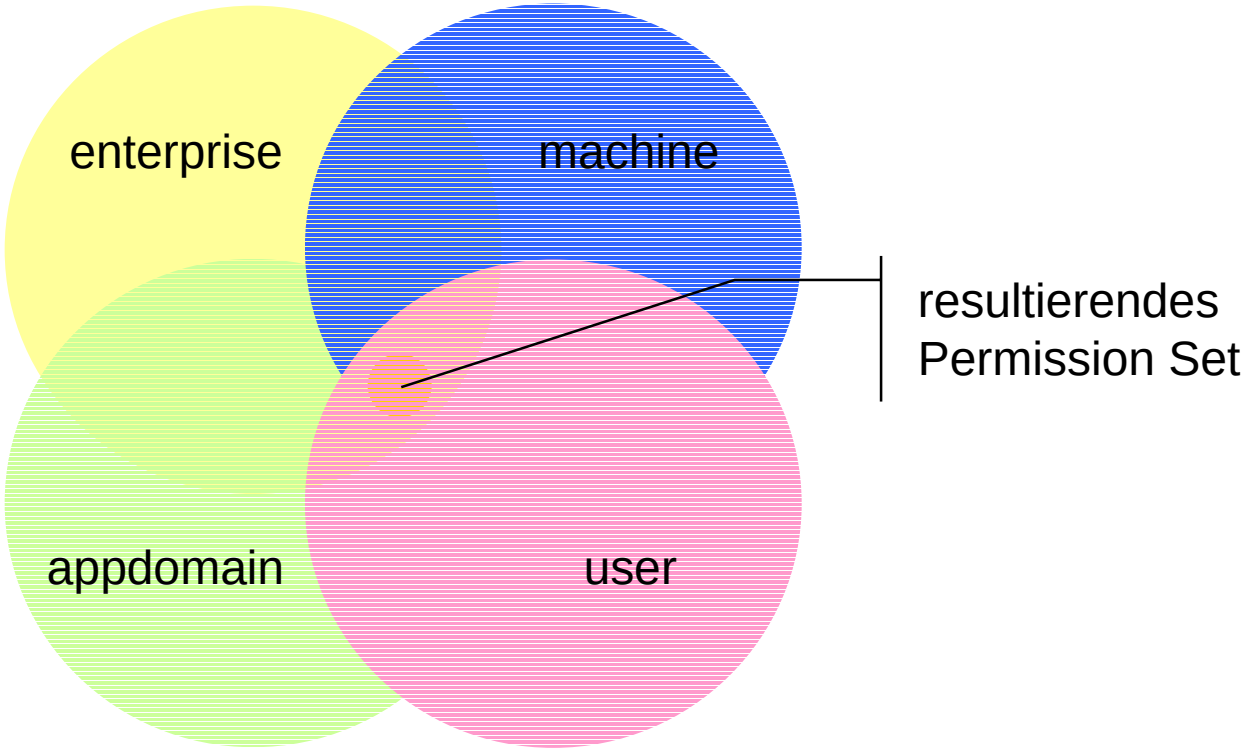
Nothing ∪ MoreTrust ∪ Microsoft ∪ FullTrust

Code Access Security (CAS)

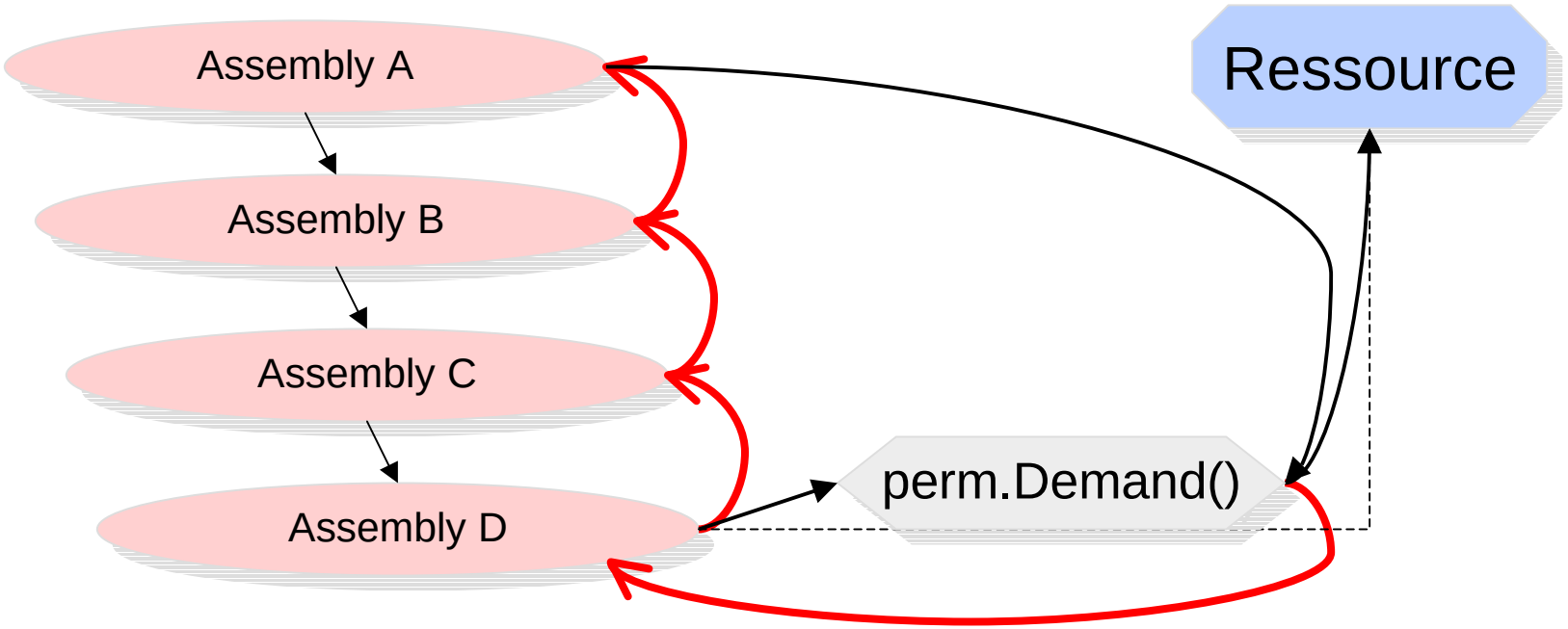


Policy Levels

- Policies sind auf verschiedenen Ebenen administrierbar



Security Stack Walk



- alle Rufer werden überprüft
- nur wenn jeder die Genehmigung perm besitzt, darf auf die Ressource zugegriffen werden

Permissions anfordern

- Es wird überprüft ob die Permission vorhanden ist
  - beim Start und nicht erst beim Zugriff
- Kann auf Methoden oder Block-Ebene durchgeführt werden
  - Declarativ oder Imperativ

Deklarative Permissions

- … über Attribute
- Spezielle Permissions für Assembly, Klasse oder Methode
  - Lower Lever Security überschreibt Higher Level Security!
- Zur Ladezeit wird entschieden ob Permission gewährt wird

```
using System.Security.Permissions;

[FileIOPermissionAttribute(SecurityAction.Demand)]
public static string ReadData() {
   // lese File ein
}
```

Imperative Permissions

School of Engineering

- … über expliziten Code
- Erzeuge ein Permission Objekt und rufe seine Methoden auf
- Schutz bezieht sich auf Methode
- Zur Laufzeit wird entschieden ob Permission gewährt wird

```
using System.Security.Permissions;

String fullPath = Directory.GetFullPathInternal(fileName);
FileIOPermission p = new FileIOPermission(
                   FileIOPermissionAccess.Read, fullPath);
p.Demand();
```

Stack Walk Modifikatoren

- Modifikatoren überschreiben das Ergebnis des Stack-Walks
- **Assert**
  - Ich verbürge mich für meine Aufrufer. Permission nicht weiter prüfen
  - Security Loch
- **Deny**
  - Permission wird explizit verweigert
- **PermitOnly**
  - Erlaubt Zugriff auf eine spezielle Ressource

Fragen?

Anhang A: Konfiguration der Sicherheitspolitik

School of Engineering

- Microsoft .NET Framework Configuration Tool
  - MMC (Microsoft Management Console) Snap-In
  - Starten
    - *mit* *Start | Control Panel | Administrative Tools | Microsoft .NET Framework Configuration*
    - *oder von der Kommandozeile mit mscorcfg.msc*
  - bietet Unterstützung (Wizards) für
    - *Erzeugen/Verändern von neuen Codegruppen*
    - *Erzeugen/Verändern von neuen Genehmigungsmengen*
    - *Feststellen der Genehmigung/Codegruppen eines Assemblies*

- Code Access Security Policy Tool (caspol.exe)
  - Kommandozeilewerkzeug
    - *z.B. Ein-/Ausschalten der Sicherheitsprüfungen*

      caspol ***-security*** *(on | off)*

… Konfiguration der Sicherheitspolitik

Code Access Security Policy Tool (caspol.exe)

Kommandozeilen-Werkzeug zum

- Ein-/Ausschalten der Sicherheitsprüfungen
  > caspol **-security** (on | off)

Ab .NET 2 kann Security nicht mehr permanent ausgeschalten werden

- Hinzufügen/Entfernen/Verändern/Anzeigen von Codegruppen
  > caspol **-addgroup**
  > caspol **-remgroup**
  > caspol **-chggroup**
  > caspol **-listgroups**
- Hinzufügen/Entfernen/Verändern/Anzeigen von PermissionSets
  > caspol **-addpset**
  > caspol **-rempset**
  > caspol **-chgpset**
  > caspol **-listpset**
- ...
- Kommandos wirken auf Enterprise-/Maschinen-/Benutzerebene
  > caspol **-enterprise** / **-machine** / **-user** / **-all** . . .
- ...