8.4 Microsoft .NET

http://www.microsoft.com/net

.NET Framework =

① *CLR – Common Language Runtime*
ist objektorientierte virtuelle Maschine für Ausführung von „managed cod

② Reichhaltige Klassenbibliotheken für die CLR

③ *C#* als „typische“ Sprache für die CLR

④ Neugestaltete Web-Unterstützung: ASP.NET,
Web Services,
...

8.4.1 Sprachunabhängiges Laufzeitsystem: CLR

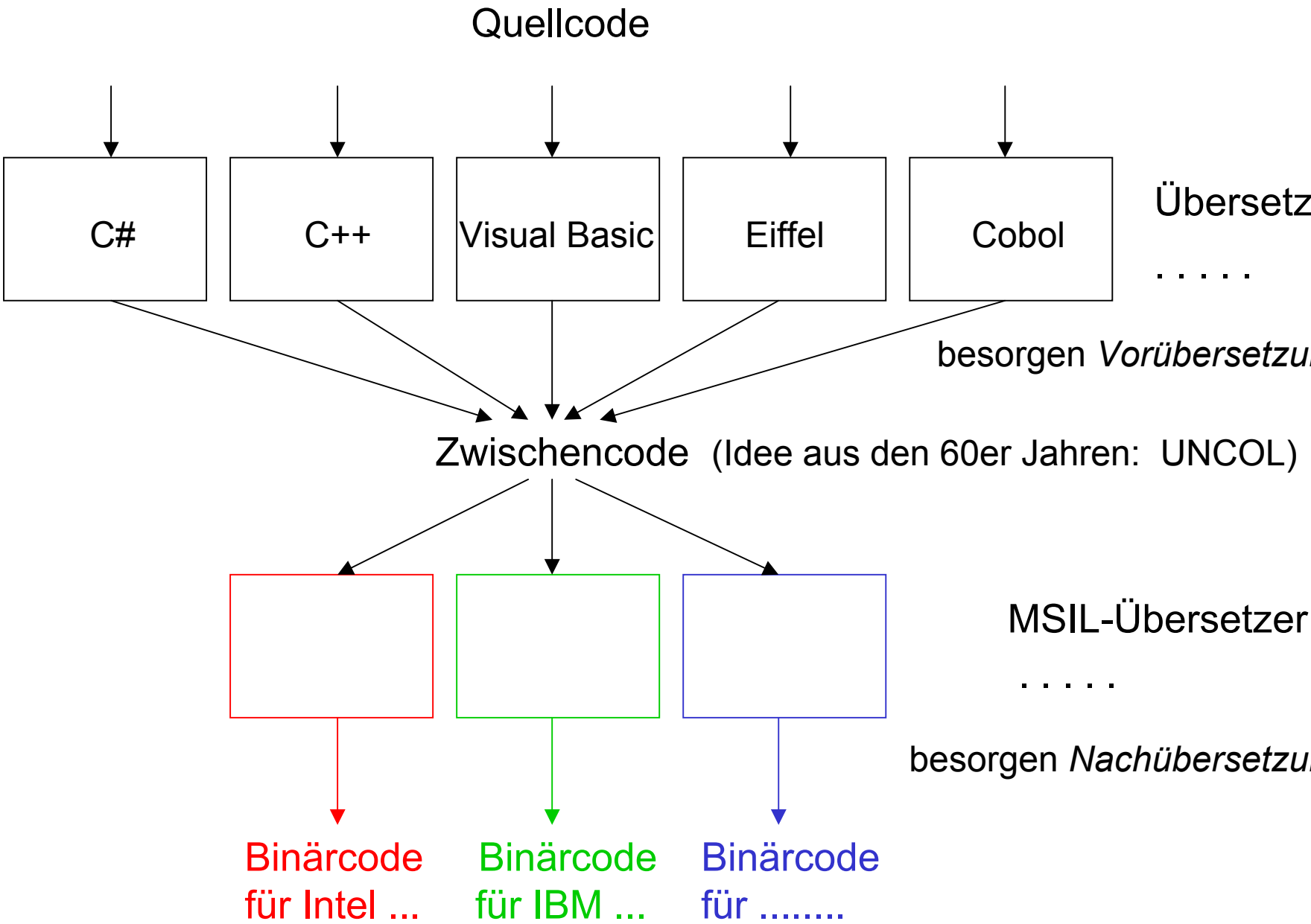
CLR = Common Language Runtime =

virtuelle Maschine für die Ausführung von objektorientiertem Zwischencode in *MSIL – Microsoft Intermediate Language –* (Microsoft‘s Antwort auf Java Bytecode/Virtual Machine)

- Zwischencode zwischen verschiedenartigen Systemen übertragbar und *überall ausführbar*, wo CLR implementiert ist

- dynamisches Laden/*Übersetzen*/Binden von MSIL-Klassen, sobald erstmals benötigt (Objekterzeugung)

- Zwischencode wird vor der Ausführung *immer* in Binärcode für die loka Zielplattform übersetzt *(just-in-time compilation)*
  (auch vorbereitend explizit mittels Befehl `ngen`)


Quellcode
C#
C++
Visual Basic
Eiffel
Cobol
Übersetz
. . . . .
besorgen Vorübersetzu
Zwischencode (Idee aus den 60er Jahren: UNCOL)
MSIL-Übersetzer
. . . . .
besorgen Nachübersetzu
Binärcode für Intel ...
Binärcode für IBM ...
Binärcode für .....

Weitere Charakteristika der CLR-Maschine:

- *CLS – Common Language Specification* – der MSIL umfaßt *CTS – Common Type System* –, das zwischen den Typsysteme aller Sprachen vermittelt, *einschließlich Vererbung*

- einheitliche Namensraumverwaltung,
  Ausnahmebehandlung,
  Speicherbereinigung

- Versionsverwaltung

- Sicherheit: Überwachung des Programmverhaltens

➔ für verschiedenartige Programmiersprachen
Interoperabilität im gleichen Adreßraum, ohne Verteilun
mit gemeinsam genutzten Bibliotheken

8.4.2 Programmverwaltung

Programmentwicklung mit .NET SDK (Befehlszeilen)
oder Visual Studio .net

Übersetzung produziert

.dll- oder .exe-Dateien mit **verwaltetem Code** *(managed code):*

- spezieller *CLR Header*
- MSIL-Code
- Schnittstellenbeschreibungen – „unsichtbare IDL“
- weitere Metadaten

Zusammengehörige Code-Dateien können zu einer **Baugruppe** *(assembly)* zusammengefaßt werden, die mit einer **Bekanntmachung** *(manifest)* beschrieben wird.

Die Baugruppe ist die Einheit von *Versionierung*, *Vertrieb* und *Installation*.

8.4.3 C#

„Cis“ *(„C sharp“)*

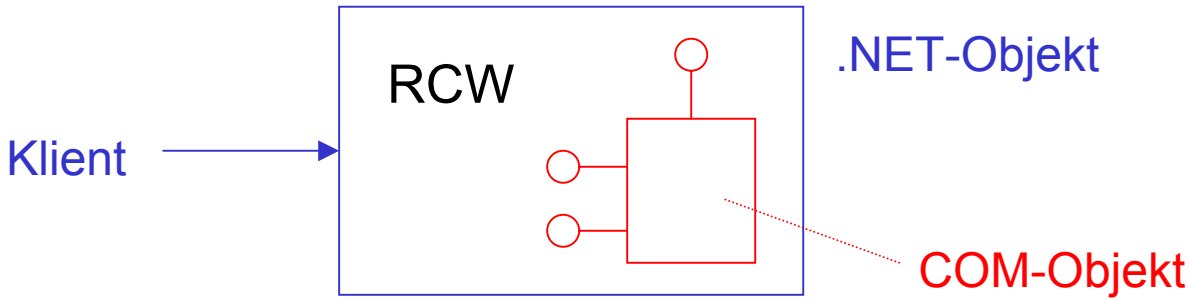
- entstammt wie Java der Sprachfamilie C/C++,
- ist Verbesserung gegenüber Java,
- erlaubt umfassende Ausnutzung der *CLR-Funktionalität* :

  - Objekteigenschaft aller Daten (`4711.ToString()`!)
  - *Verweise auf objektbezogene Operationen (delegates)* und Mengen von solchen,
  - darauf basierend *Ereignisse* (events) und Beobachter-Muster,
  - *Annotationen (attributes)* für Klassen, Operationen, ... , in die *Metadaten* übernommen und über *Reflexion* abfragbar
  - . . . . .

8.4.4 .NET und COM

Aufruf .NET → COM

verwendet Hülle (wrapper), die dem Aufrufer ein .NET-Objekt präsentiert:



RCW = *runtime-callable wrapper*

new impliziert `CoCreateInstance`

RCW-Code generierbar aus der COM-IDL-Beschreibung:

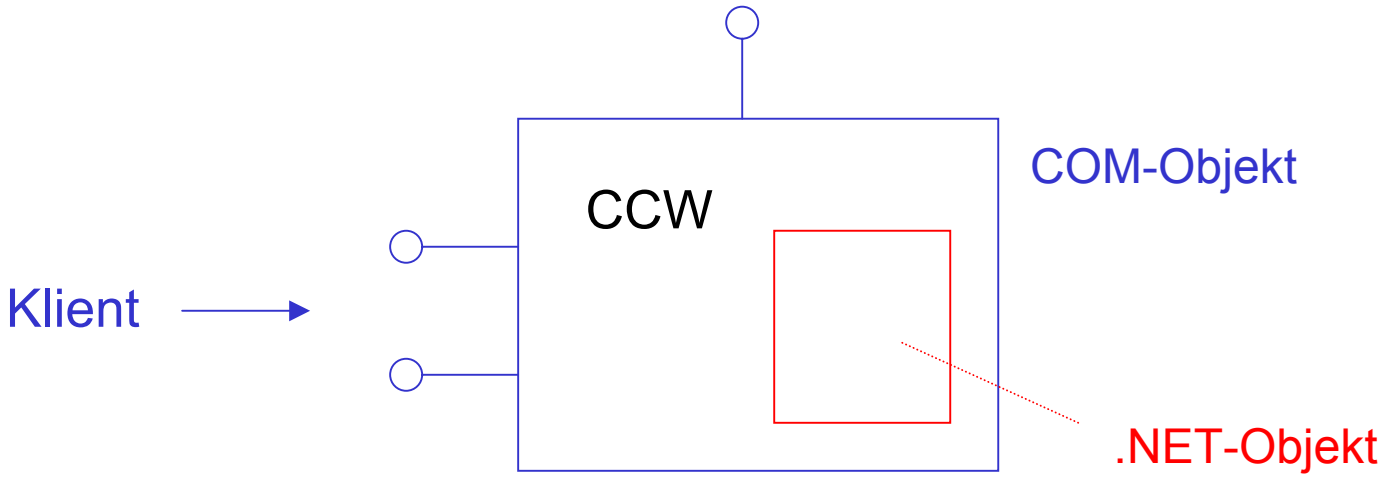
IDL-Text → Typbibliothek `example.tlb` → Namensraum `exampleL`

`mktyplib` (COM tool)

`tlbimp` (in .NET SDK)

Aufruf COM → .NET

setzt voraus, daß die Klasse argumentlosen Konstruktor hat,
wird vermittelt über einen *COM-callable wrapper - CCW*



`CoCreateInstance` impliziert **`new`** mit *argumentlosem* Konstruktor

CCW-Code aus Metadaten generieren und registrieren mit `regasm`

! RCW und CCW vermitteln zwischen `ComException` und `HRESULT` !

8.4.5 Fernaufrufe

werden von CLR + Bibliotheken unterstützt, ähnlich zu Java, aber

interoperabel *ohne IDL*,

Metadaten ersetzen IDL,

automatische Vertretererzeugung bei Bedarf,

Transport über TCP oder HTTP (SOAP, siehe 9)

Bibliotheken für Fernaufrufe:

```
System
System.Runtime.Remoting
System.Runtime.Remoting.Activation
System.Runtime.Remoting.Channels
. . .
```

Klasse für fernaufrufbare Objekte muß wie folgt erben:

```
class RemTable : System.MarshalByRefObject, I1, I2, ...
      .....
}
```

Lokale Objekterzeugung: `RemTable t = new RemTable(arg1,..)`

Fernerzeugung privater Objekte *(„client-activated objects):*

```
object[] attr = {new System.Runtime.Remoting.Activation.
                  UrlAttribute("tcp://host/Example")}
RemTable t = (RemTable)System.Activator.CreateInstance(
                    typeof(RemTable),
                    {arg1, arg2, ...}
                    attr);
```

oder mit *Ortsabstraktion* (!) bei Verwendung einer *Konfigurationsdatei:*

```
RemTable t = new RemTable(arg1, arg2, ...);
```

Beachte: `RemTable t = ...` liefert Vertreterobjekt,
dessen Klasse anonyme Unterklasse von `RemTable` ist
(„transparent proxy“)

Fernbenutzung:

Fernaufruf

```
result = t.op(arg1,...);
```

Fernzugriff – Lesezugriff auf Attribut oder Eigenschaft

```
result = t.attr;
```

Dabei wird für ein Argument- oder Ergebnis-Objekt, dessen Klasse von `MarshalByRefObject` erbt, wie im lokalen Fall ein Verweis übergeben, andernfalls eine Kopie des Objekts
(Klasse muß in diesem Fall mit `Serializable` annotiert sein)

8.4.5.1 Gemeinsame Objekte

mehrerer Klienten: durch *Registrierung* beim CLR

```
System.Runtime.Remoting.RemotingConfiguration.
      RegisterWellKnownServiceType(
            typeof(RemTable),
            "SharedTable",          // local URI
            System.Runtime.Remoting.
            WellKnownObjectMode.Singleton)
```

registriert ein lokales `RemTable`-Objekt unter dem Namen `SharedTable` - mit verzögerter Objekterzeugung (beim ersten Fernaufruf !) über *parameterlosen Konstruktor*

„server-activated, well-known object“

Klientenverhalten:

```
RemTable t = (RemTable)System.Activator.GetObject(
                       typeof(RemTable),
                       "tcp://host:8711/SharedTable
```

siehe 8.4.5.2

liefert (ohne Fernaufruf!) Vertreterobjekt (alternativ: mit `new` nach `RemotingConfiguration.RegisterWellKnownClientType(.`

```
t.op(...)
```

bewirkt Fernaufruf – gegebenenfalls mit Fernerzeugung

`WellKnownObjectMode.SingleCall` (statt `Singleton`)

bewirkt, daß für jeden Aufruf *neues Wegwerf*-Objekt erzeugt wird

Alternative: Objekt sofort erzeugen und bekanntmachen:

```
System.Runtime.Remoting.
ObjRef dummy = System.Runtime.Remoting
               RemotingServices.Marshal(
                                  new RemTable(arg1,...),
                                  "SharedTable");
```

- und der Klient:

```
RemTable t = (RemTable)System.Activator.GetObject(
                       typeof(RemTable),
                       "tcp://host:8711/SharedTable");
```

8.4.5.2 Bereitstellung von Transportkanälen

muß vor Benutzung des Fernaufrufsystems vorgenommen werden,
sowohl klientenseitig als auch serverseitig

Server:

```
System.Runtime.Remoting.Channels.Tcp.
        TcpChannel ch = new TcpChannel(8711);
System.Runtime.Remoting.Channels.
        ChannelServices.RegisterChannel(ch);
```

Klient:

```
System.Runtime.Remoting.Channels.Tcp.
        TcpChannel ch = new TcpChannel();
System.Runtime.Remoting.Channels.
        ChannelServices.RegisterChannel(ch);
```