# .NET

# Alapok, szerelvények

***Albert István***
ialbert@aut.bme.hu
QB. 221, 463-1662

BME, Automatizálási és Alkalmazott Informatikai Tanszék

# Tartalom

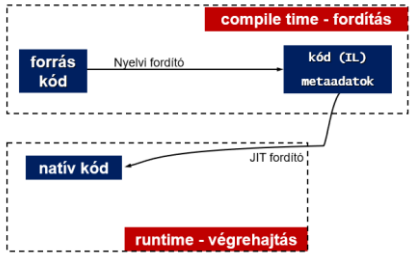
## Miért kellenek programozási környezetek?

- Hardver
  - > CPU, gépikód, assembly
  - > I/O portok, eszközök, időzítés stb.
- Absztrakciós szint növelése
  - > Magasabb szintű fogalmak bevezetése
  - > Szöveg, grid, rekord, reláció stb.

## Miért született meg a .NET?

- A '90-es évek vége: VB és C++ nem elég
  - > Nincs produktív, modern nyelvi megoldás
- J++: MS saját Java implementációja
  - > Új konstrukciók (COM): például események, tulajdonságok
  - > Nyílt levelek, pereskedés stb.
- Internet térnyerése
  - > Minden mindenel össze lesz kötve
  - > Nem asztali számítógép eszközök megjelenése

## Fordítás és végrehajtás




## Egyszerűbb telepítés

- Megoldás a DLL hell-re !
- Assemblyk (szerelvény)
  - > A kód elemi egysége telepítés, verziókezelés és biztonsági/jogosultsági szempontból
  - > Hasonlít a DLL-hez, de önleíró
- Manifest: a metadata hordozója
- Mellékhatások nélküli telepítés
  - > Közös vagy privát alkalmazások és komponensek
  - > Ezt a szerző határozza meg!
- Egymás melletti futtatás
  - > Alkalmazáson / folyamatok belül is futhat egymás mellett több változat

## Futtatókörnyezetek és keretrendszerek

- Futtatókörnyezet: CLR
  - > GC, JIT, ClassLoader, Marshalling, PAL, ...
- Keretrendszer: BLC
  - > List<T>, StringBuilder, FileStream, ...
- Több implementáció
  - > .NET Framework 4.X
  - > .NET Core
  - > (... .NET CF, Rotor, Silverlight, .NET for WinRT)

# Miért kellenek programozási környezetek?

- Hardver
  - CPU, gépikód, assembly
  - I/O portok, eszközök, időzítés stb.
- Absztrakciós szint növelése
  - Magasabb szintű fogalmak bevezetése
  - Szöveg, grid, rekord, reláció stb.

# Nyelvi konstrukciók

- Imperatív vagy deklaratív megközelítés
  - Hogyan, állapot – mit, szabályok
- Metódusok / funkciók (beágyazott funkciók)
  - Változó paraméter lista
- Lokális változók (capture)
- Modulok – láthatóság
- OO koncepciók
  - Öröklés, polimorfizmus, virtuális metódusok
- Jobb rekurzió, lista kezelés stb

# Nyelvi konstrukciók implementációja

- Adatok ábrázolása a memóriában
  - Egész szám? (16 bit / 32 bit)
  - Szöveg? (Karakter kódolás? Hossz/végjel?)
  - Lebegő pontos számok?
- Típusok
  - Szerződések a különböző komponensek között
- OO koncepciók ábrázolása a memóriában
  - Virtuális metódusok címe?
  - Futásidejű típus információ?

# Futásidejű konstrukciók

- Metodus paraméterek, visszatérési értékek?
  - Jobb rekurzió, végtelen stack? (Prolog)
- „this” pointer?
- Ki takarítja a vermet?
  - Pascal vagy C hívási konvenció?
- Kivételkezelés
- Memória kezelés
  - Manuális vagy sem? Pinning?
- Debuggolás, Edit&Continue, ...

# Megvalósítások 1

- A nyelvi szint megvalósítása „össze nő” az **infrastruktúrá**val
  - A C (Pascal, Prolog stb) nyelvi fordító a gépi kód generálásakor meghatározza, hogy milyen hívások konvencióval, szám ábrázolással stb dolgozik
  - A köztes nyelvet használó megoldásoknál (Java, .NET, VB stb) a futtató környezet határozza meg a konvenciókat

# Megvalósítások 2

- A megvalósítások általában erősen kötődnek egy adott **operációs rendszer**hez
  - Tipikusan C API hívások
  - Bootstrapping más
  - Paraméterezés más
  - Koncepciók különbözhetnek (thread, fork)

# Nincs együttműködés

- A programozási környezetek zárt rendszerek
- Se a környezetek se a programnyelvek nincsenek felkészítve az együttműködésre
  - Általában egy C jellegű külső hívási megoldást támogatnak (pl Java)

- Miért?
  - Elvi nehézségek (nyelvi koncepciók)
  - Technikai kihívások (pinning)

# Mi volt a .NET előtt?

- 1985 – Windows 1.0
  - 2015 – Windows 10
- API? – C
- C
  - Nehéz nyelv, könnyen elrontható (memória)
  - Nem produktív, alacsony szintű
  - Hatékony, close to the metal

# COM, Visual Basic

- 1991: Visual Basic =
  vizuális programozás (D&D) + BASIC nyelv
  - Saját futtatókörnyezet, Windows-hoz
- 1993: COM - Component Object Model
  - Bináris kompatibilitás (együttműködés) a különböző nyelven írt komponensek között
    - Visual Basic és C++ (Pascal, Delphi, J++, ... .NET)
  - Rögzített nyelvi interfészek, típusrendszer, memória kezelés és megvalósítási konvenciók
  - OLE, ActiveX, COM+, **WinRT**: mind COM

# ’90-es évek: VB és C++

- VB: gyors, vizuális lapátolás
  - Üzleti alkalmazások felhasználói felülete
  - Nagyon korlátozott nyelv (pl nem OO)
  - Furcsa leágazások, pl ASP (spagetti)
- C++ (ATL, MFC): nehéz
  - Manuális memória kezelés, alacsony absztrakció
  - COM stílusban programozni kihívás (ma is ☺)

**„Csak annak, aki el tudja viselni az igazságot.”**

# 1995: Java

- Modern koncepciók
  - Automatikus memória kezelés (lisp, ‘60)
  - Egyszerű OO (egyszeres öröklés, COM)
  - C jellegű tömör szintaktika
  - Nincs szabvány
- „Írd meg egyszer, ~~debuggold~~ futtasd mindenhol.”
  - Szép cél és majdnem sikerült
- A nyelv és a futtató környezet összenőt
  - Kb 300 nyelv készült hozzá kutatási jelleggel
  - Nehéz más nyelvi környezetekkel összeilleszteni

# Miért született meg a .NET?

- A ’90-es évek vége: VB és C++ nem elég
  - Nincs produktív, modern nyelvi megoldás
- J++: MS saját Java implementációja
  - Új konstrukciók (COM): például események, tulajdonságok
  - Nyílt levelek, pereskedés stb.
- Internet térnyerése
  - Minden mindenel össze lesz kötve
  - Nem asztali számítógép eszközök megjelenése

# .NET

1. Modern futtatókörnyezet
   > Automatikus szemétgyűjtés, biztonság stb
2. Új korszerű, produktív nyelv
   > C#: kompromisszumok nélkül
3. Több nyelv támogatása
   > Kompatilitás: VB, C++, COM, ...
4. Platformfüggetlenség

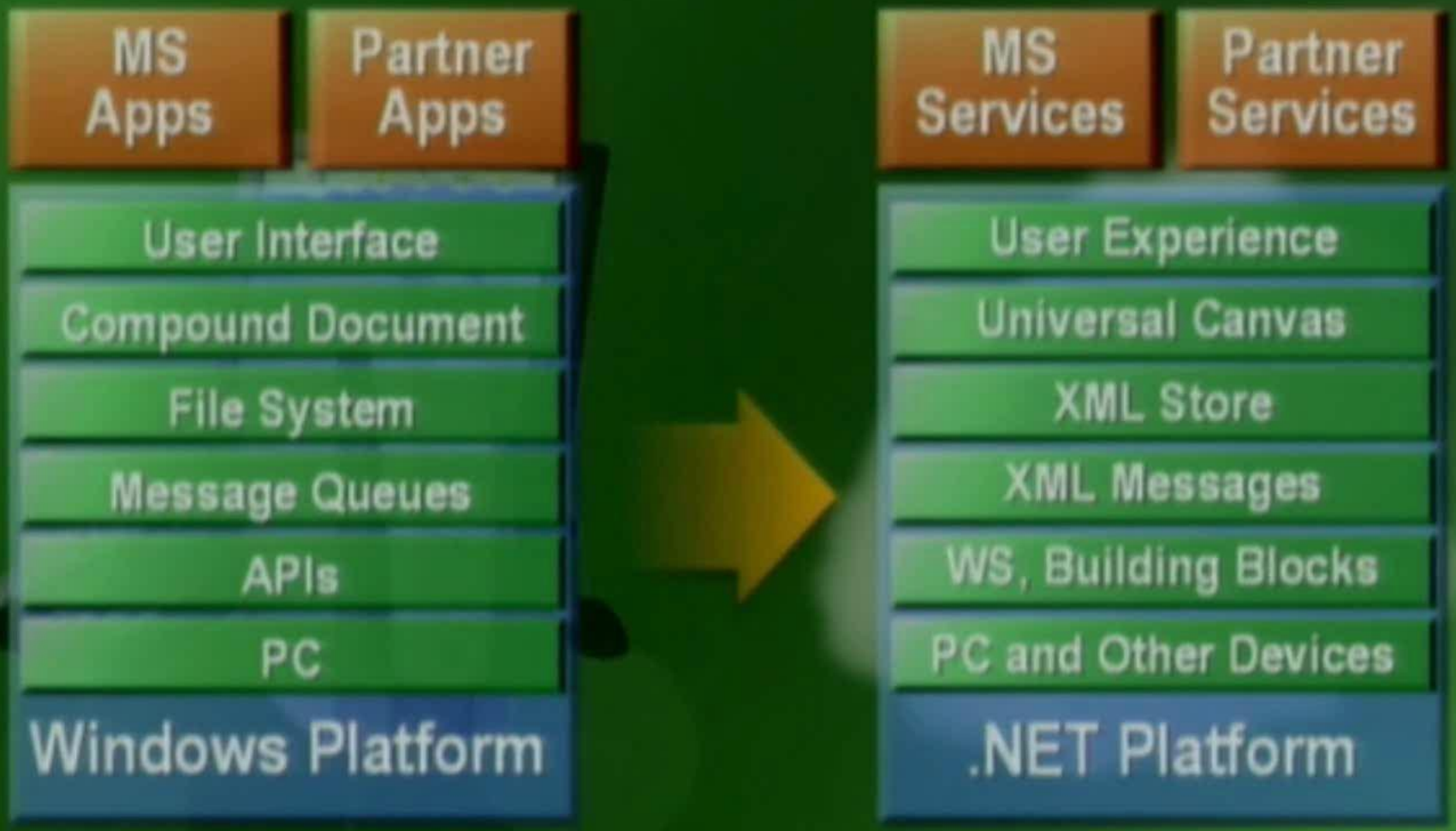
# A .NET már 15 éves!


- .NET Framework 1.0

2002. február 13

*Microsoft .NET provides software developers with the most consistent programming model, enabling them to build applications the way they want across platforms, services, and devices.*

# .NET: A Natural Evolution




*As big a step forward as from MS-DOS to Windows*

# Egyszerűbb, produktív fejlesztés

- Minden objektum-orientált
  - Mindenhol használható osztályok és öröklődés
  - Nyelvek között is!
- Szerveződés
  - Hierarchikus, osztályokba és névterekbe szervezett kód
- Egységes, gazdag típusrendszer
- Komponens-orientált
  - A tulajdonságok, metódusok, események, attribútumok nyelvi szinten jelennek meg
  - Gazdag tervezés idejű funkcionalitás
- Zökkenőmentes integráció

# Moduláris, kiterjeszthető

- Operációs rendszer funkcióinak elérése osztályokon keresztül
- A Framework nem teljesen „fekete doboz”
  - Nagy része C#-ban íródott
  - Nyílt forráskódú
- Bármely .NET osztály kiterjeszthető örökléssel
- Akár nyelvek között is, forráskód nélkül

# Minden osztály / objektum

- Tradicionális megközelítések
  - C++, Java: a primitív típusok mágikusak, a „normál” osztályokkal nem működnek együtt
  - Smalltalk, Lisp: a primitív típusok is osztályok – de nagyon lassúak
- .NET, C#
  - Egységesség, teljesítmény romlás nélkül
  - Bővíthetőség (SQLInteger, Decimal, BigInt, …)
  - A gyűjtemények minden típussal működnek

# Robusztus és biztonságos

## Felügyelt kód

- Felügyelt kód: ellenőrzött
  - Kötelező metainformációk
  - Az IL kód szigorú típusellenőrzésnek vethető alá
  - Megszűnnek gyakori hibaforrások
    - veszélyes típusváltások (type cast)
    - inicializálatlan változók, tömbből kilógó indexek
- Kivételkezelés
  - Nyelvi szinten definiált hibakezelés - sokkal használhatóbb hibajelzés
  - Integrálódik a Windows strukturált kivételkezelésével is (SEH)

# Robusztus és biztonságos

## Felügyelt adatok

- Felügyelt adat: élettartam-felügyelet
  - Minden .NET objektumot a GC takarít el (garbage collector – a szemetes)
  - Nincs elfelejtett pointer, korán felszabadított memória, körkörös hivatkozásokat is kezeli
  - Modern, önhangoló GC algoritmus (mark & compact)

# Nyelvek

- A .NET platform nyelvfüggetlen
  - A Framework minden szolgáltatása hozzáférhető minden nyelv számára
  - Minden nyelv egyformán fordul, egyik sem interpretált
- Common Language Specification
  - Közös nyelv a nyelvek számára, alapfogalmak: például van öröklés, virtuális metódus, stb.
  - Új, tetszőleges nyelvvel kibővíthető

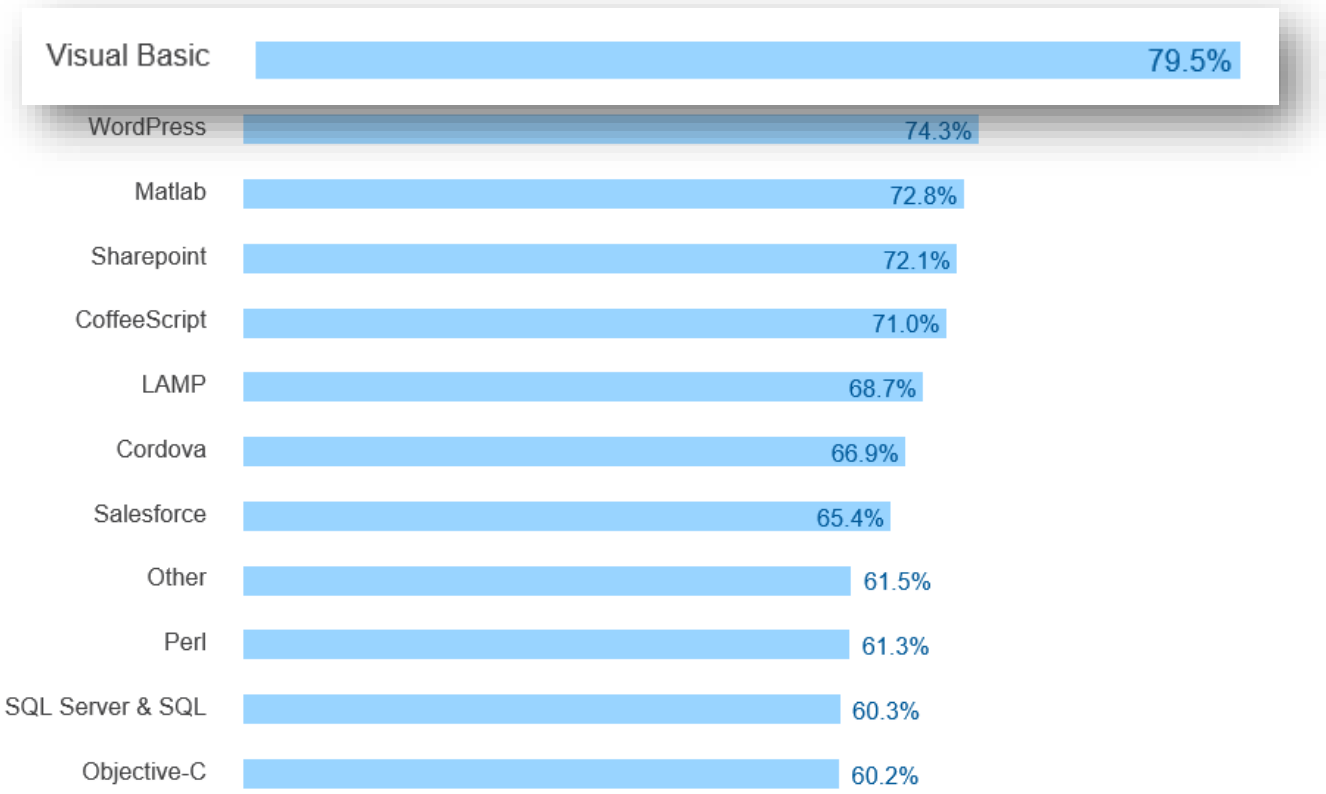
# Nyelvek

- Perl
- *Iron Python*
- COBOL
- **Haskell**
- ML
- JScript
- Ada
- APL
- **Eiffel**
- Pascal
- C
- C++
- *C#*
- Visual Basic
- SmallTalk
- Spec#
- Scheme
- Mercury
- **F#**
- Objective Caml

# Visual Basic .NET

- Egyenrangú nyelv a többivel
  - Egyszerű szintaktika
  - VBRUN helyett .NET Framework
- Jelentős nyelvi fejlődés régi VB-hez képest
  - Osztályok, öröklődés, konstruktorok, stb.
  - Strukturált kivételkezelés (SEH)
  - Egyfajta értékadás!
  - Lehetőség a szigorú típusellenőrzésre
- Új stratégia (2017. február 1)
  - Maradjon egyszerű, lassabb fejlődés!
  - Elsősorban új fejlesztőknek szól!

# Legutáltabb technológiák




http://stackoverflow.com/research/developer-survey-2016#technology-most-loved-dreaded-and-wanted

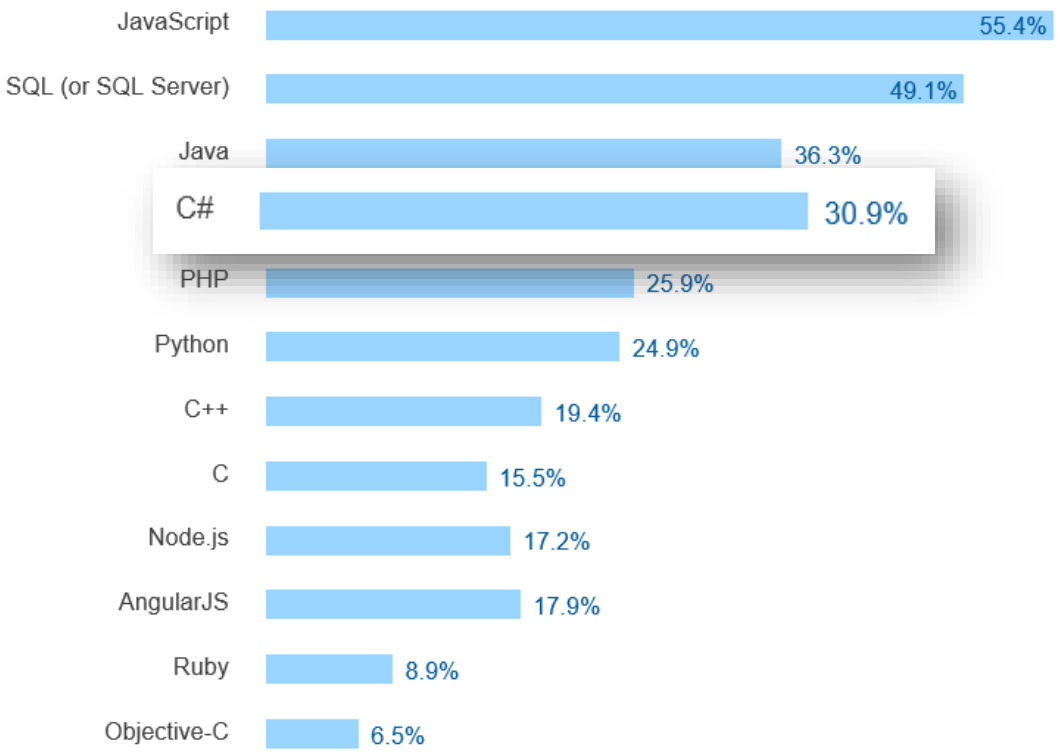
# C++/CLI

- Meglévő C++ kód átvitele .NET alá
  - Egyszerű együttműködés
- Kiterjesztett C++
  - Szabványosított
- Továbbra is „Total Control”
  - Natív és felügyelt kód és adat keverése
  - Lehetővé teszi a fokozatos átállást, más technológiák elérését
- Teljes hozzáférés a .NET CLR funkcióihoz

# C#

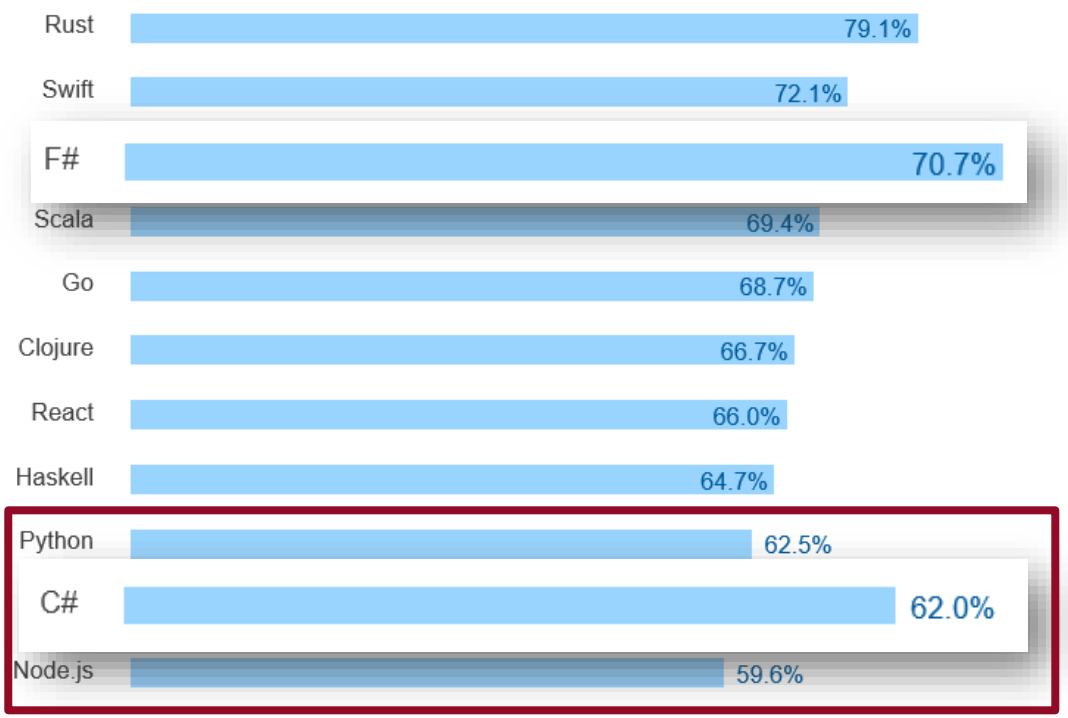
- A C/C++ család első komponens-orientált tagja
  - Properties, Methods, Events, Attributes, XML documentation stb.
  - Minden egy helyen, nincs IDL, stb.
- Minden osztály
  - A primitív típusokhoz sem kell varázsolni
- A .NET Framework is C#-ban van írva

# Legelterjedtebb technológiák




http://stackoverflow.com/research/developer-survey-2016#technology-most-popular-technologies

# Legkedveltebb technológiák

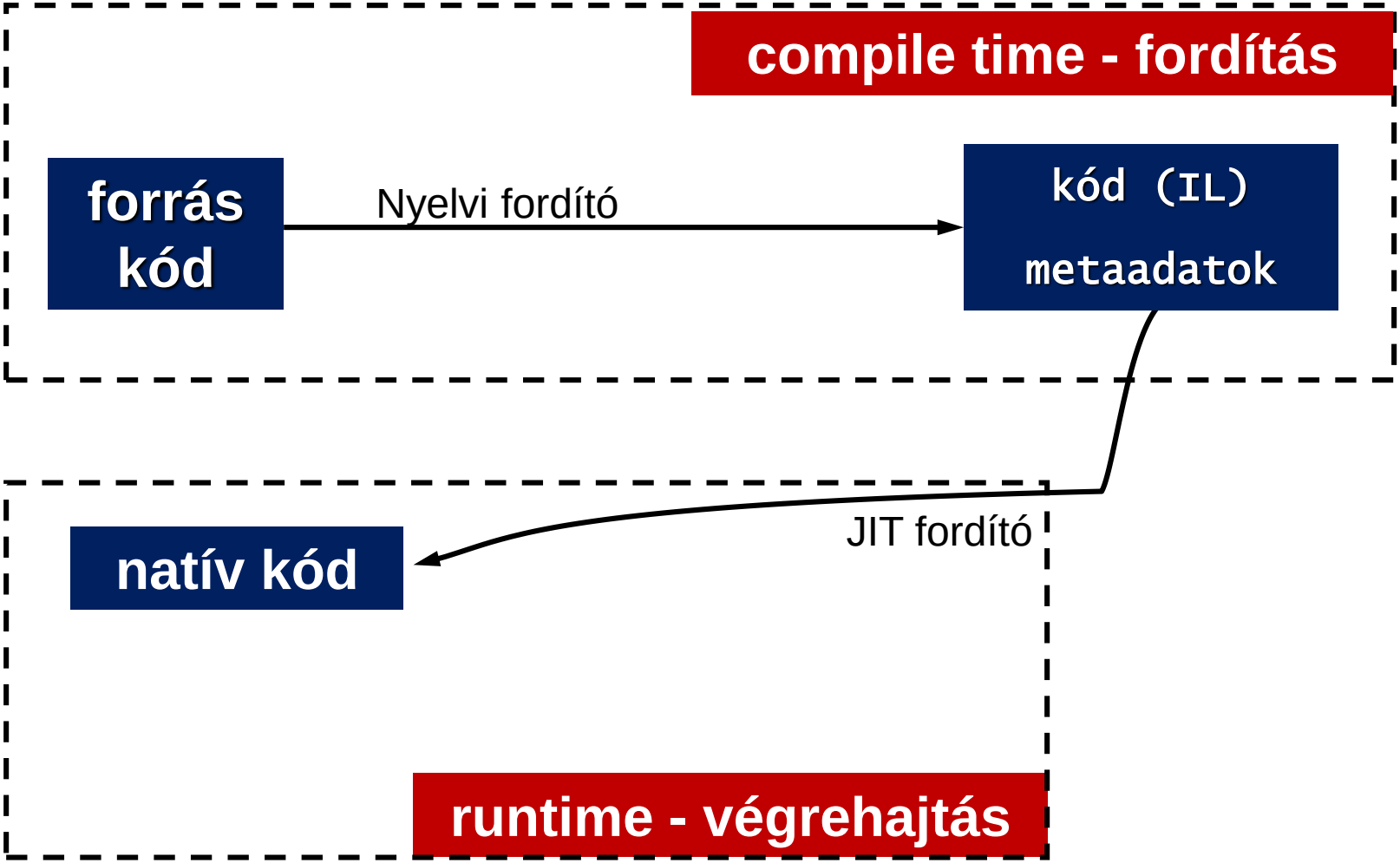



http://stackoverflow.com/research/developer-survey-2016#technology-most-loved-dreaded-and-wanted

# Nyelvi funkciók nyílt fejlesztése

| Feature | Example | C# | VB |
|---|---|---|---|
| Auto-property initializers | `public int X { get; set; } = x;` | Added | Exists |
| Getter-only auto-properties | `public int Y { get; } = y;` | Added | Added |
| Ctor assignment to getter-only autoprops | `Y = 15` | Added | Added |
| Parameterless struct ctors | `Structure S : Sub New() : End Sub : End Structure` | Added | Added |
| Using static members | `using System.Console; … Write(4);` | Added | Exists |
| Dictionary initializer | `new JObject { ["x"] = 3, ["y"] = 7 }` | Added | No |
| Await in catch/finally | `try … catch { await … } finally { await … }` | Added | No |
| Exception filters | `catch(E e) if (e.Count > 5) { … }` | Added | Exists |
| Partial modules | `Partial Module M1` | N/A | Added |
| Partial interfaces | `Partial Interface I1` | Exists | Added |
| Multiline string literals | `"Hello<newline>World"` | Exists | Added |
| Year-first date literals | `Dim d = #2014-04-03#` | N/A | Added |
| Line continuation comments | `Dim addrs = From c in Customers ' comment` | N/A | Added |

# Fordítás és végrehajtás



# Fordítás

- Közös nyelvi specifikáció
- Minden nyelvhez más fordító
- Azonos kimenet: MSIL (CIL)
- Azonos szolgáltatások:
  - Nyelvek közti származtatás, kivételek, …
  - Kód ellenőrzés, típus kompatibilitás
  - Debug szimbólumok
  - Teljesítmény számlálók

# CIL

- Processzor független, evaluation-stack alapú
- Továbbfordításra tervezték
- Nyelvfüggetlen
- Objektumorientáltság jellemzi
- Metaadat:
  - típusok leírása
  - tagváltozók, metódusok leírása
  - ...
- Egyéb információk: optimalizás, GC, ...

# CIL-ből natív kód

- JIT compiler: Just-In-Time fordító
  - Lusta: csak akkor fordít, amikor egy metódusra szükség van
  - Debug JIT: optimalizálatlan, Edit&Continue
  - Standard JIT: optimalizált(abb) kód
- NGEN:
  - NEM JIT, a teljes szerelvényt fordítja már telepítéskor
    - .NET frissítéskor újra kell fordítani mindent
  - Gyors indulás
  - Optimálisabb fordítás
    - De fordítási egységeken kívüli hívás lassabb

# Mit jelent a típusosság?

- A memória egy összefüggő területe **egység**et képez (***objektum***), struktúrája megfeleltethető a **fordítás időben** megadott ***típus***nak, amely megadja, hogy az objektum milyen alaptípusokból és más típusokból áll (milyen műveletek végezhetők rajta stb.)
- A memória (***program állapot***) csak a típusok leírásának megfelelően kezelhető, változtatható
  - > Az állapot átmenetek ellenőrizhetők!

# Típusbiztonság

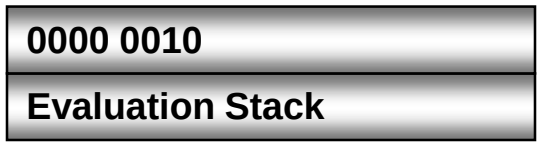
- Ez egy nyelvi jellemző!
  - Például Java-ra, C#-ra (kivéve a pointer kezelést) igaz
- Az x86 gépi kódra általánosságban nem igaz
  - Túl általános, memória műveletekkel dolgozik
  - De lehet olyan gépi kódot írni, ami ellenőrizhető!
- Az IL kód, Java byte kód is ellenőrizhető
  - Kiértékelési verem alapú (evaluation stack)

# IL végrehajtási modell

- A végrehajtási modell kiértékelési verem alapú
  - Az összes nyelv erre fordul le
  - Csak egy modell, az igazi végrehajtás a gép regisztereit használja!
- Jó modell ellenőrzéshez
  - RPN 1920, Burks, Dijkstra, Hamblin, 195x

10 + 20 - 5

```
IL_0001:  ldc.i4  10
IL_0002:  ldc.i4  20
IL_0003:  add
IL_0004:  ldc.i4.5
IL_0005:  sub
```

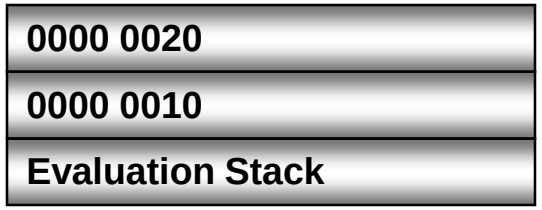


# IL végrehajtási modell

- A végrehajtási modell kiértékelési verem alapú
  - Az összes nyelv erre fordul le
  - Csak egy modell, az igazi végrehajtás a gép regisztereit használja!
- Jó modell ellenőrzéshez
  - RPN 1920, Burks, Dijkstra, Hamblin, 195x

10 + 20 - 5

```
IL_0001:  ldc.i4  10
IL_0002:  ldc.i4  20
IL_0003:  add
IL_0004:  ldc.i4.5
IL_0005:  sub
```

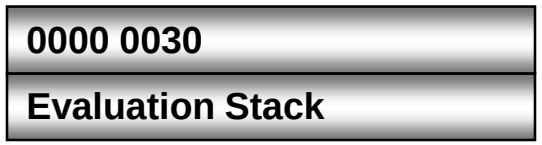


# IL végrehajtási modell

- A végrehajtási modell kiértékelési verem alapú
  - Az összes nyelv erre fordul le
  - Csak egy modell, az igazi végrehajtás a gép regisztereit használja!
- Jó modell ellenőrzéshez
  - RPN 1920, Burks, Dijkstra, Hamblin, 195x

10 + 20 - 5

```
IL_0001:  ldc.i4  10
IL_0002:  ldc.i4  20
IL_0003:  add
IL_0004:  ldc.i4.5
IL_0005:  sub
```

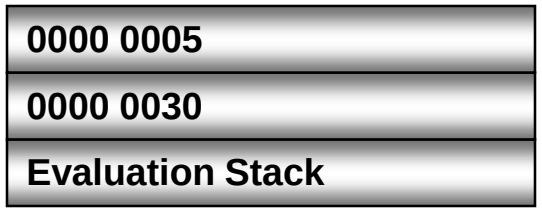


# IL végrehajtási modell

- A végrehajtási modell kiértékelési verem alapú
  - Az összes nyelv erre fordul le
  - Csak egy modell, az igazi végrehajtás a gép regisztereit használja!
- Jó modell ellenőrzéshez
  - RPN 1920, Burks, Dijkstra, Hamblin, 195x

10 + 20 - 5

```
IL_0001:  ldc.i4  10
IL_0002:  ldc.i4  20
IL_0003:  add
IL_0004:  ldc.i4.5
IL_0005:  sub
```

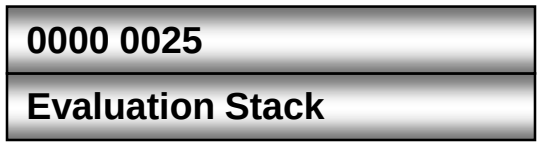


# IL végrehajtási modell

- A végrehajtási modell kiértékelési verem alapú
  - Az összes nyelv erre fordul le
  - Csak egy modell, az igazi végrehajtás a gép regisztereit használja!
- Jó modell ellenőrzéshez
  - RPN 1920, Burks, Dijkstra, Hamblin, 195x

10 + 20 - 5

```
IL_0001:  ldc.i4  10
IL_0002:  ldc.i4  20
IL_0003:  add
IL_0004:  ldc.i4.5
IL_0005:  sub
```



# IL végrehajtási modell

```
int seed = 0;

0 references
int Calculate( int a, int b )
{
    return a + b + seed;
}
```

```
.method private hidebysig instance int32
        Calculate(int32 a,
                  int32 b) cil managed
{
  // Code size       11 (0xb)
  .maxstack  8
  IL_0000:  ldarg.1
  IL_0001:  ldarg.2
  IL_0002:  add
  IL_0003:  ldarg.0
  IL_0004:  ldfld      int32 ILshow.Program::seed
  IL_0009:  add
  IL_000a:  ret
} // end of method Program::Calculate
```

# IL végrehajtási modell

```
int seed = 0;


0 references
double Calculate( double a, double b )
{
    return a + b + seed;
}
```

```
.method private hideby
        Calculate(int3
                  int3
{
  // Code size       1
  .maxstack  8
  IL_0000:  ldarg.1
  IL_0001:  ldarg.2
  IL_0002:  add
  IL_0003:  ldarg.0
  IL_0004:  ldfld
  IL_0009:  add
  IL_000a:  ret
} // end of method Pro
```

```
.method private hidebysig instance float64
        Calculate(float64 a,
                  float64 b) cil managed
{
  // Code size       12 (0xc)
  .maxstack  8
  IL_0000:  ldarg.1
  IL_0001:  ldarg.2
  IL_0002:  add
  IL_0003:  ldarg.0
  IL_0004:  ldfld      int32 ILshow.Program::seed
  IL_0009:  conv.r8
  IL_000a:  add
  IL_000b:  ret
} // end of method Program::Calculate
```

# Egyszerűbb telepítés

- Megoldás a DLL hell-re !
- Assemblyk (szerelvény)
  - A kód elemi egysége telepítés, verziókezelés és biztonsági/jogosultsági szempontból
  - Hasonlít a DLL-hez, de önleíró
- Manifest: a metadata hordozója
- Mellékhatások nélküli telepítés
  - Közös vagy privát alkalmazások és komponensek
  - Ezt a szerző határozza meg!
- Egymás melletti futtatás
  - Alkalmazáson / folyamatok belül is futhat egymás mellett több változat

# Assemblyk (=szerelvények)

- Funkciók logikai egysége ( logikai .dll )
  - metaadatok a .NET-es osztályokról
  - IL kód ( PE fájl – Portable Executable )
  - erőforrások ( .jpg, .txt, … )
- Minden alkalmazás assemblykből épül fel
  - Egy névtér több assemblyben
  - Egy assemblyben több névtér
- Hivatkozhat más assemblykre

# A manifeszt információk

- Név
- Verzió: major, minor, build number, revision
- Támogatott culture, processzor és OS
- Megosztott (erős - strong) nevek
- Assembly referenciák :
  - név (nyilvános kulcs részlet, ha megosztott)
  - verziószám
- Típus referenciák
- Egyéb attribútumok: cím, leírás, ...

# Assembly mint egység …

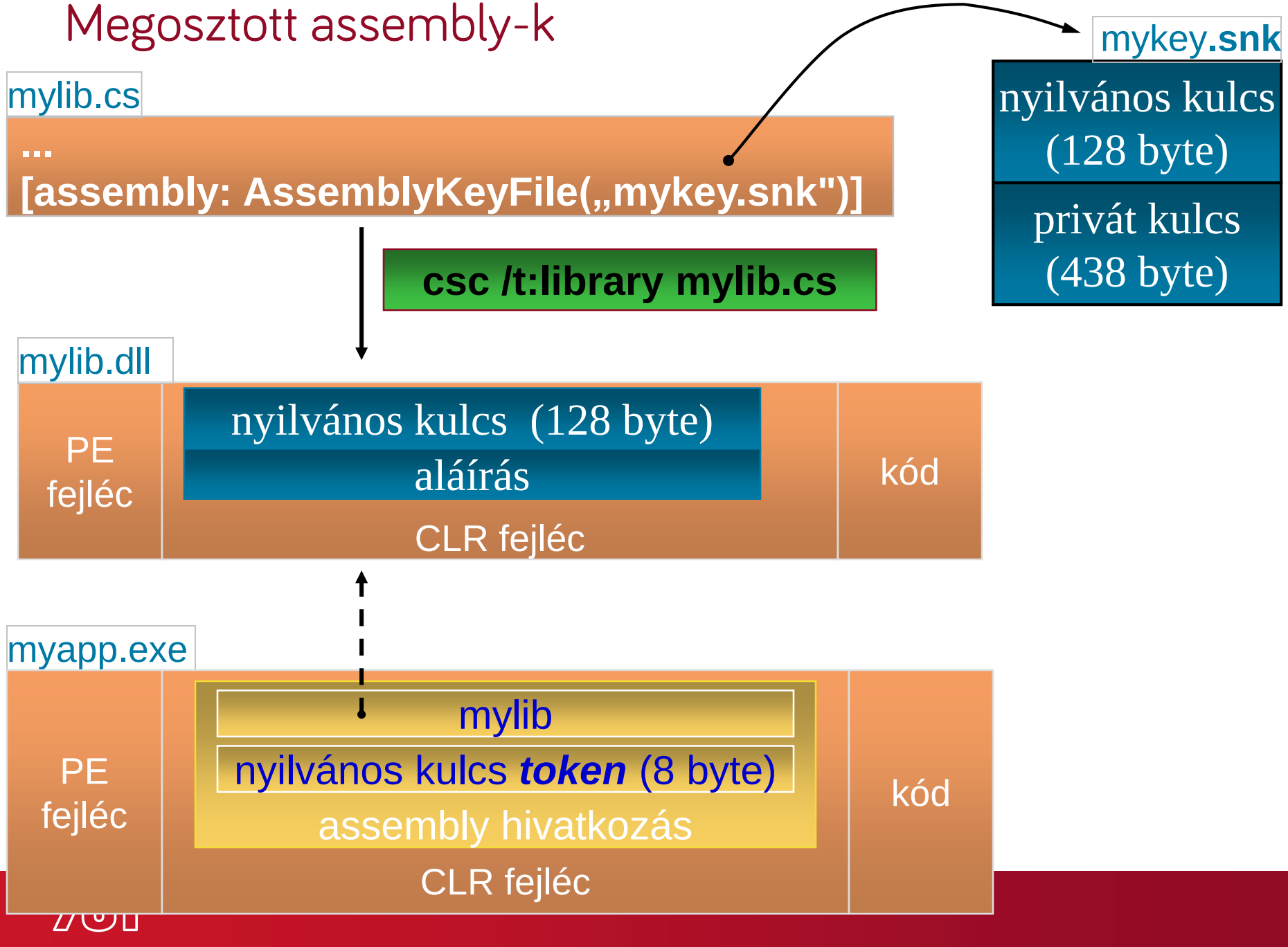
- Típus egység
  - a típusok assemblyhez kötődnek és nem névterekhez !
- Egymás-melletti végrehajtás egysége
  - Több verzió futhat egymás mellett, akár egy folyamaton belül is

# Verziókezelés

**major . minor . buildnumber . revisionnumber**

- major, minor: inkompatibilis verziók
  - teljesen új verzió született
- buildnumber: talán még kompatibilisek
  - kisebb módosítások – szervíz csomag
  - biztonságos/normál mód konfigurálható
- revisionnumber: kompatibilisnak szánt verziók
  - csak hibajavítás
  - QFE: Quick Fix Engineering

# Megosztott assembly-k



# Globális gyorsítótár

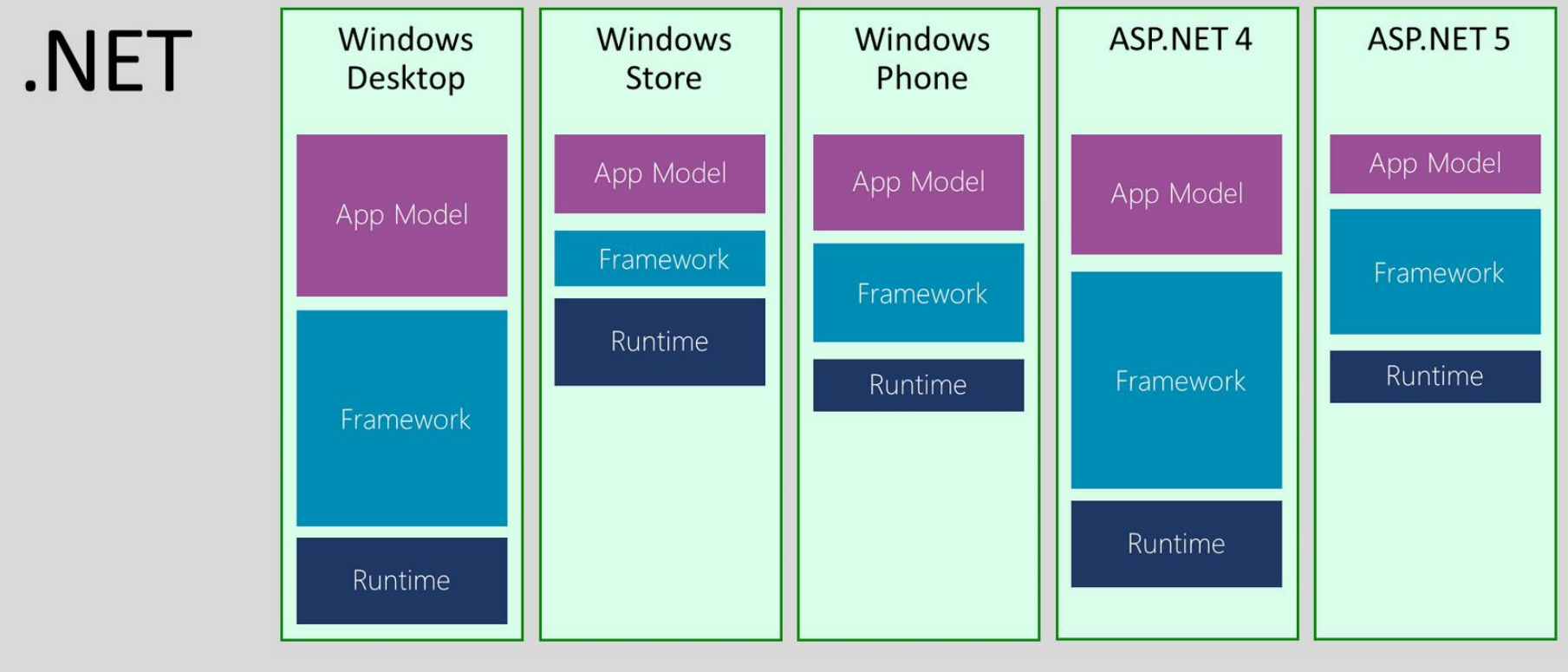
- Több alkalmazás által használt komponensek gyűjtőhelye (system32/*.dll)
- Teljesítménynövekedés – ellenőrzés, betöltés
- Integritás – minden hivatkozott típus, assembly megvan
- Automatikus QFE
- Csak megosztott assembly-k telepíthetők – egyediség, biztonság
- Csak az adminisztrátor törölhet
- Eszköz: shell kiterjesztése

# Futtatókörnyezetek és keretrendszerek

- Futtatókörnyezet: CLR
  - GC, JIT, ClassLoader, Marshalling, PAL, …
- Keretrendszer: BLC
  - List<T>, StringBuilder, FileStream, …

- Több implementáció
  - .NET Framework 4.X
  - .NET Core
  - (… .NET CF, Rotor, Silverlight, .NET for WinRT)

# Különböző .NET stackek



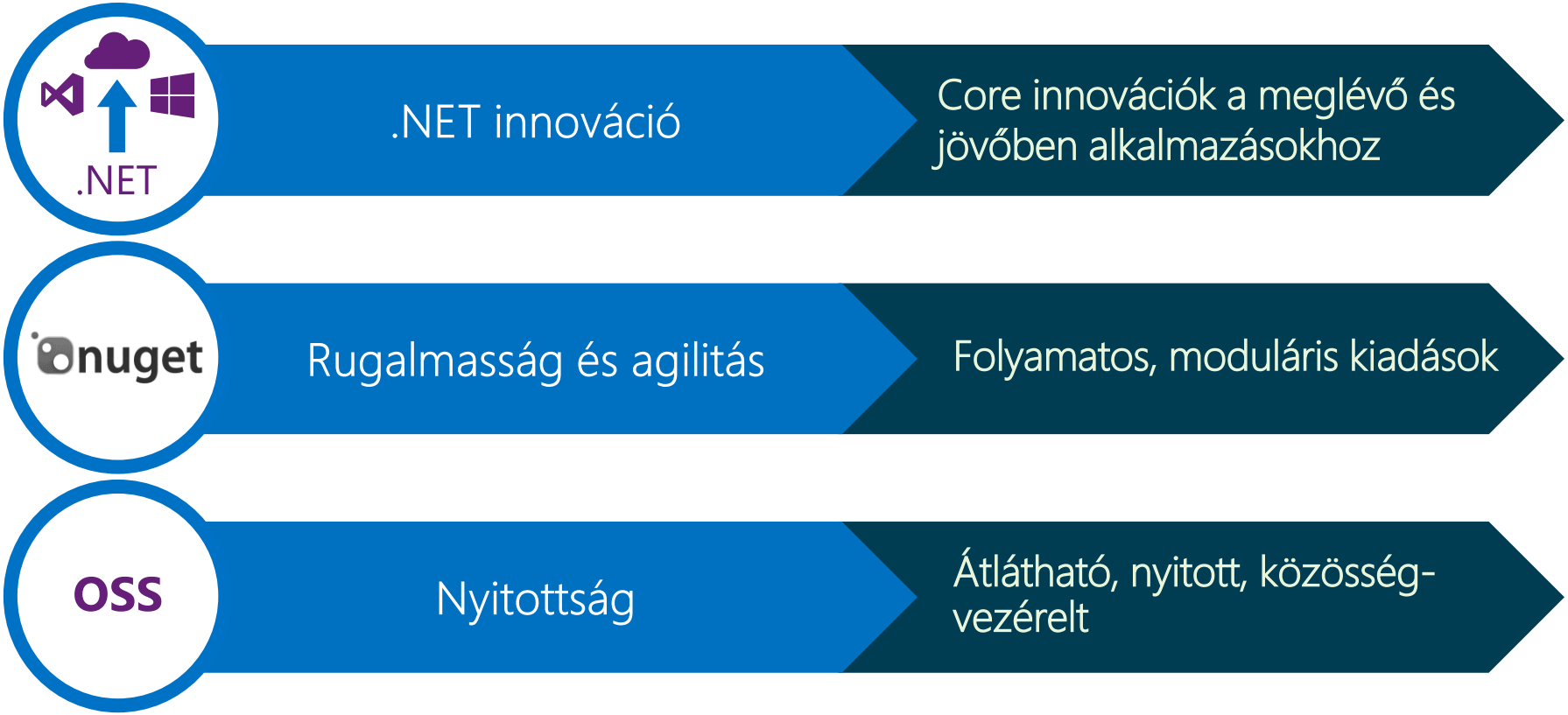
# .NET Framework 4.7

- A meglévő, klasszikus asztali .NET keretrendszer következő verziója
  - Néhány új feature, bugfixes
- Monolitikus
  - ASP.NET, WCF, WPF, WF, WinForms, …
- Windows 10-től előtelepítve!

# Gép szintű monolitikus keretrendszer

- Előnyei
  - Központilag kezelhető – vállalati környezet
  - Kisebb helyigény (mintha alkalmazásonként lenne)
  - Natív kód megosztása az alkalmazások között
- Hátrányai
  - Közös függőség bevezetése a gépen futó különböző alkalmazások kötött
  - Frissítés esetén kompatibilitási problémák
  - Adminisztrátor jogosultságok kellenek

# .NET Foundation: új megközelítés



# Ami változik mostanában...

| | | |
|---|---|---|
| Run on Windows |  | Run everywhere |
| .NET as system component |  | Deploy with app |
| Run on VM (CLR) |  | Compile to native |
| Black box compilers |  | Open compiler APIs |
| Edit in Visual Studio |  | Use your favorite editor |
| Proprietary |  | Open source |

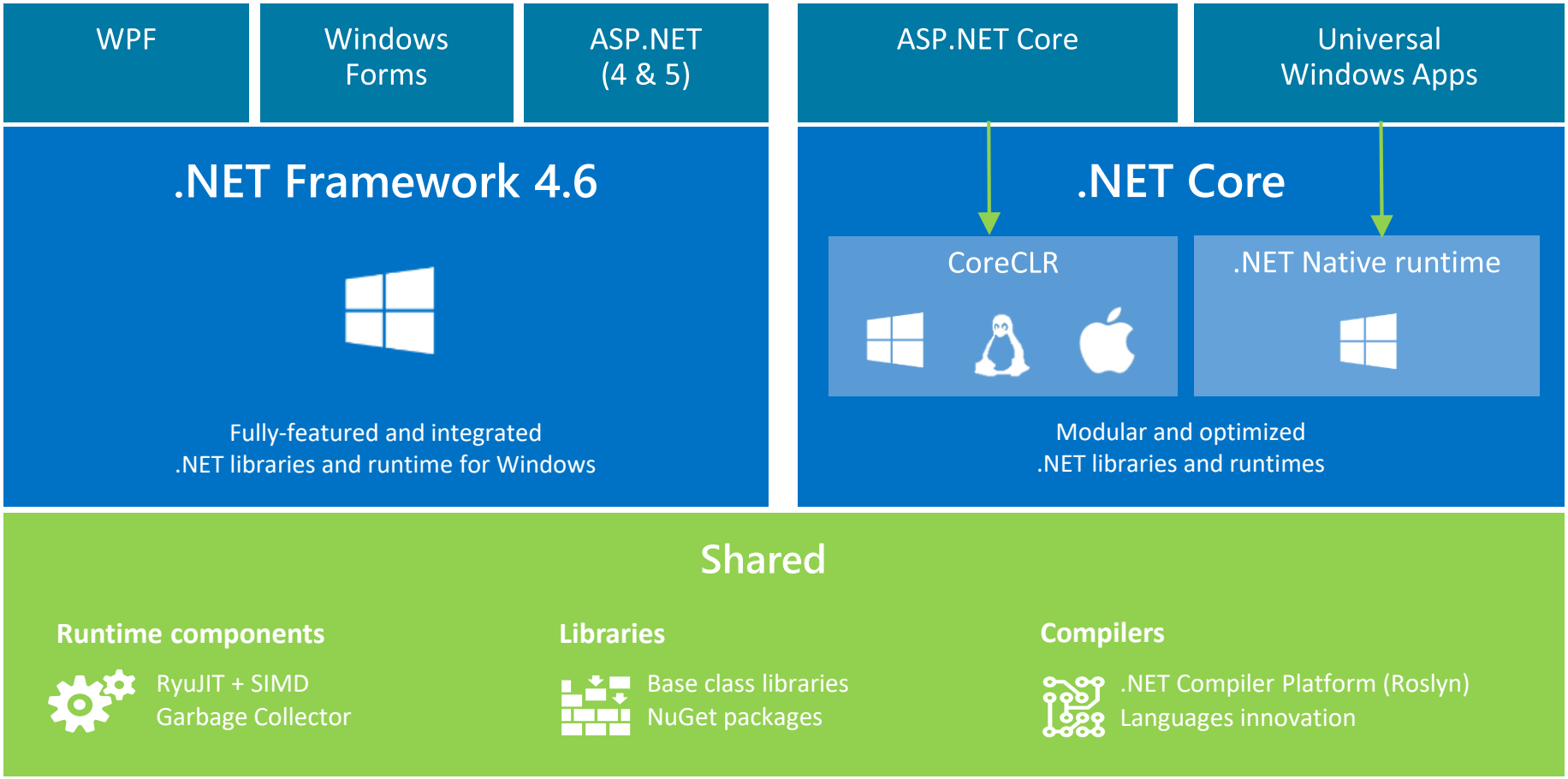
# .NET Core 1.0

- A nagy keretrendszer egy forkja, amit erősen refaktoráltak
- Csak egy minimális BCL tartozik hozzá
  - Gyorsabb: pl hideg indítás: 13 sec helyett 3 sec
- Moduláris
  - Minden további funkció NuGet csomagokban
  - A csomag neve = a szerelvény neve
- Nyílt forráskódú
- Cross-platform: Windows, Linux, Mac OSX
- Side-by-side telepítés

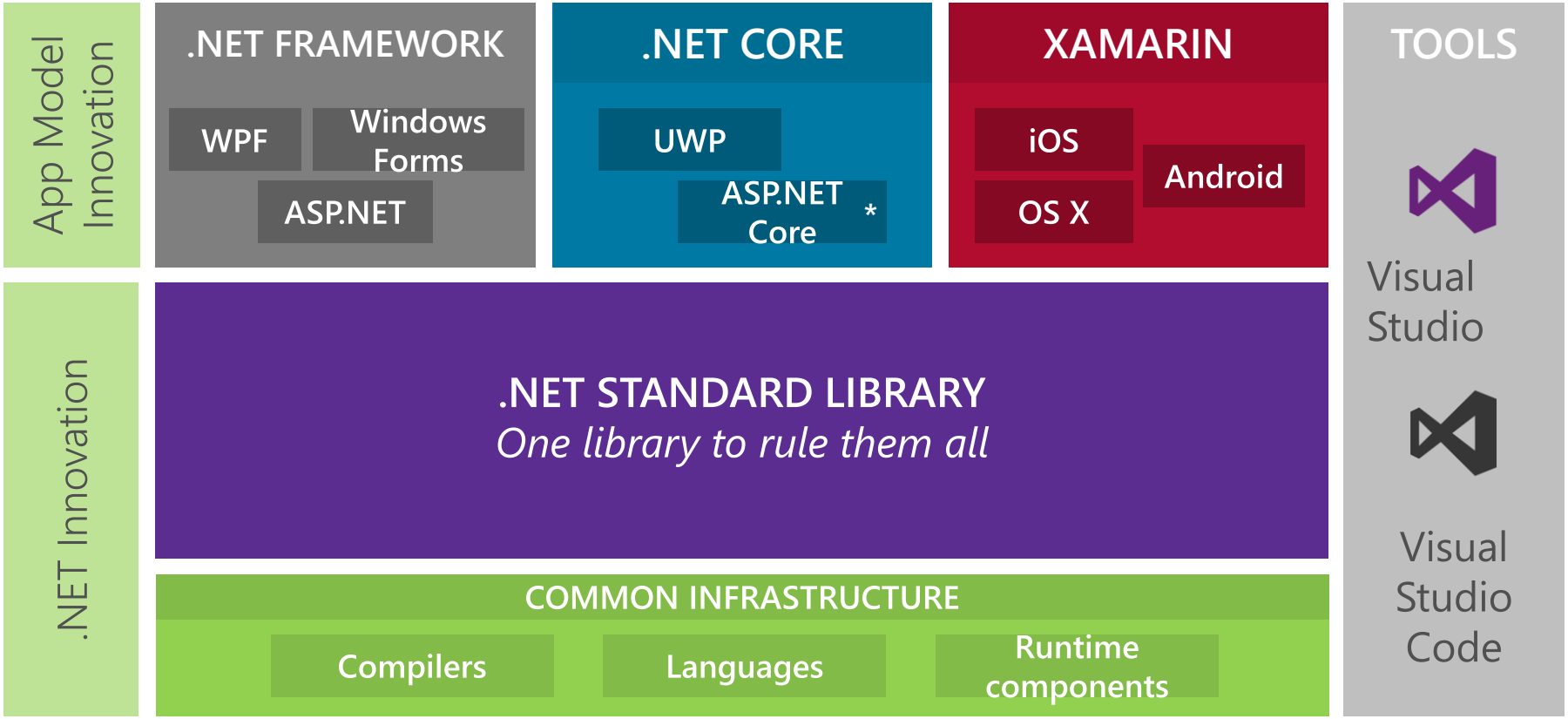
# Ami közös a két implementációban

- Új JIT: RyuJIT
  - SIMD támogatás, egyéb optimalizációk
- Fordító infrastruktúra: Roslyn
  - Nyelvi feature-ök
- Könyvtárak
  - BCL közös részei – azonos implementáció!
  - Entity Framework
  - ...
- A .NET Framework lényegében a .NET Core + NuGet csomagok egy összetesztelt snapshotja

# .NET


WPF
Windows Forms
ASP.NET (4 & 5)
ASP.NET Core
Universal Windows Apps
.NET Framework 4.6
Fully-featured and integrated
.NET libraries and runtime for Windows
.NET Core
CoreCLR
.NET Native runtime
Modular and optimized
.NET libraries and runtimes
Shared
Runtime components
RyuJIT + SIMD
Garbage Collector
Libraries
Base class libraries
NuGet packages
Compilers
.NET Compiler Platform (Roslyn)
Languages innovation

# .NET innovációs terek



# .NET Platform Standard

- .NET Standard Library
  - Like C++ standard lib
- Egy API minden .NET környezetben és platformon
  - Csak API contract, nincs implementáció
  - A megírt kód mindenhol újra felhasználható
  - **A ráépülő könyvtárak mindenütt használhatóak**
- Megvalósítás
  - netstandard.dll, referencia szerelvény (assembly)
  - Type-forwarding: a platform adja az implementációt

# .NET Standard 2.0 – 32k API

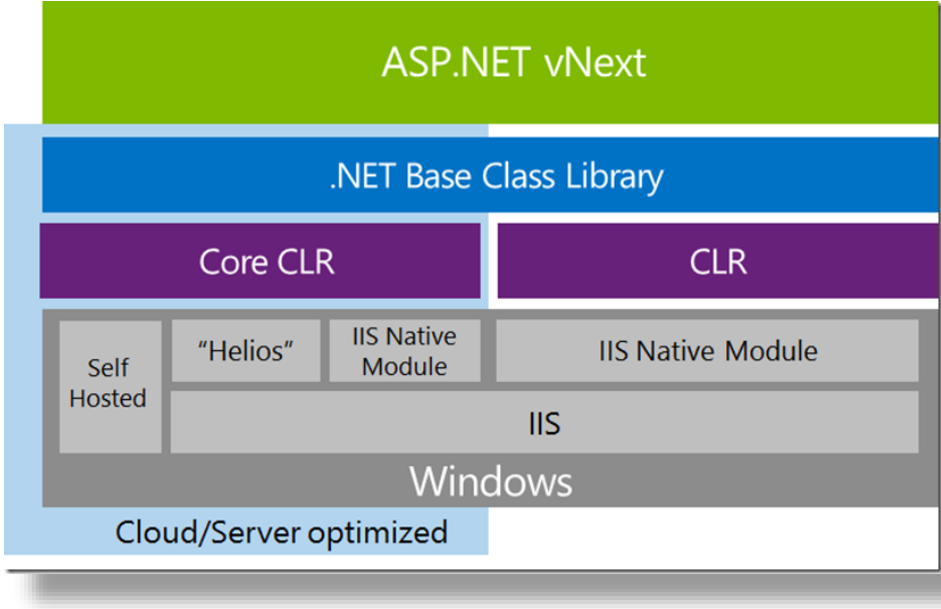
| | |
|---|---|
| **XML** | XLinq • XML Document • XPath • Schema • XSL |
| **SERIALIZATION** | BinaryFormatter • Data Contract • XML |
| **NETWORKING** | Sockets • HTTP • Mail • WebSockets |
| **IO** | Files • Compression • MMF |
| **THREADING** | Threads • Thread Pool • Tasks |
| **CORE** | Primitives • Collections • Reflection • Interop • Linq |

# Miért lett nyílt forráskódú?

- Cross-platform
  - Ez nem egy technikai követelmény, de tipikus
  - „Olcsóbb”
- Gyorsabb bugfixing, jobb dizájn
  - Rögtön kapnak visszajelzést: hibák, API struktúra, …
  - Több lehetőség próbálkozásra
- Hatékonyabb a Microsoft belső működése
  - Minden csapat egyszerűen hozzáfér a kódhoz…
- A .NET Framework miért csak „source open”?
  - Túl sok gépen van rajta, a kompatibilitás túl fontos

# Összefoglalás

ASP.NET Core 1.0
- tisztább,
- gyorsabb,
- egyszerűbb




## Cloud ready, and cross-platform

Felhőre tervezett, cross-platform megoldás

## Modular and Open

Moduláris, nyílt forráskódú, agilis fejlesztés

## Improved tooling and frameworks

Gyorsabb kiadási ciklusok