

C# 4, 5, 6, 7, 8

{ Albert István }

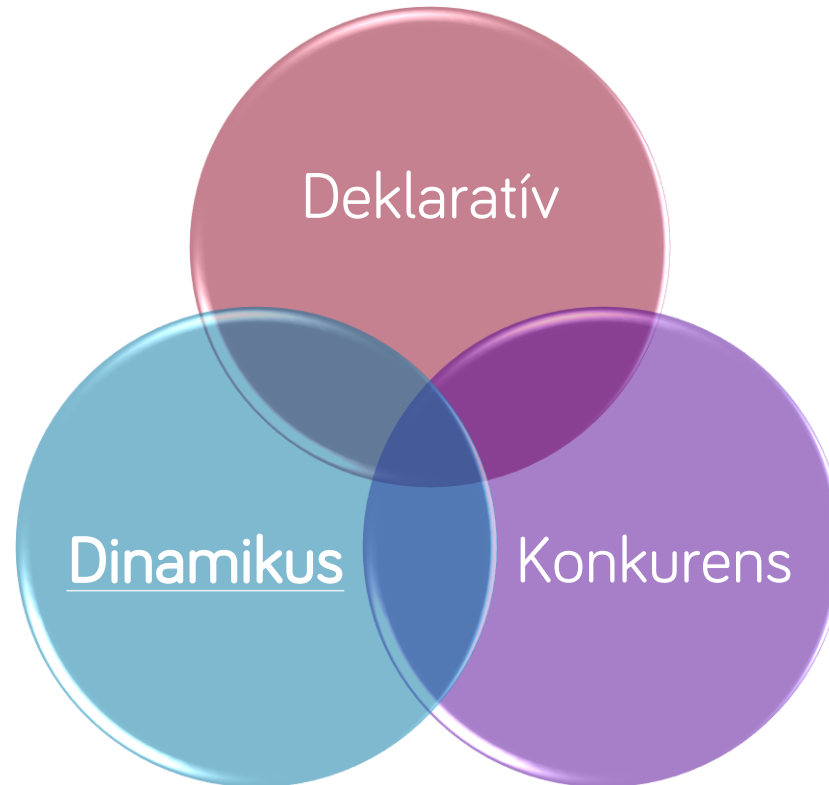


Automatizálási és
Alkalmazott
Informatikai Tanszék

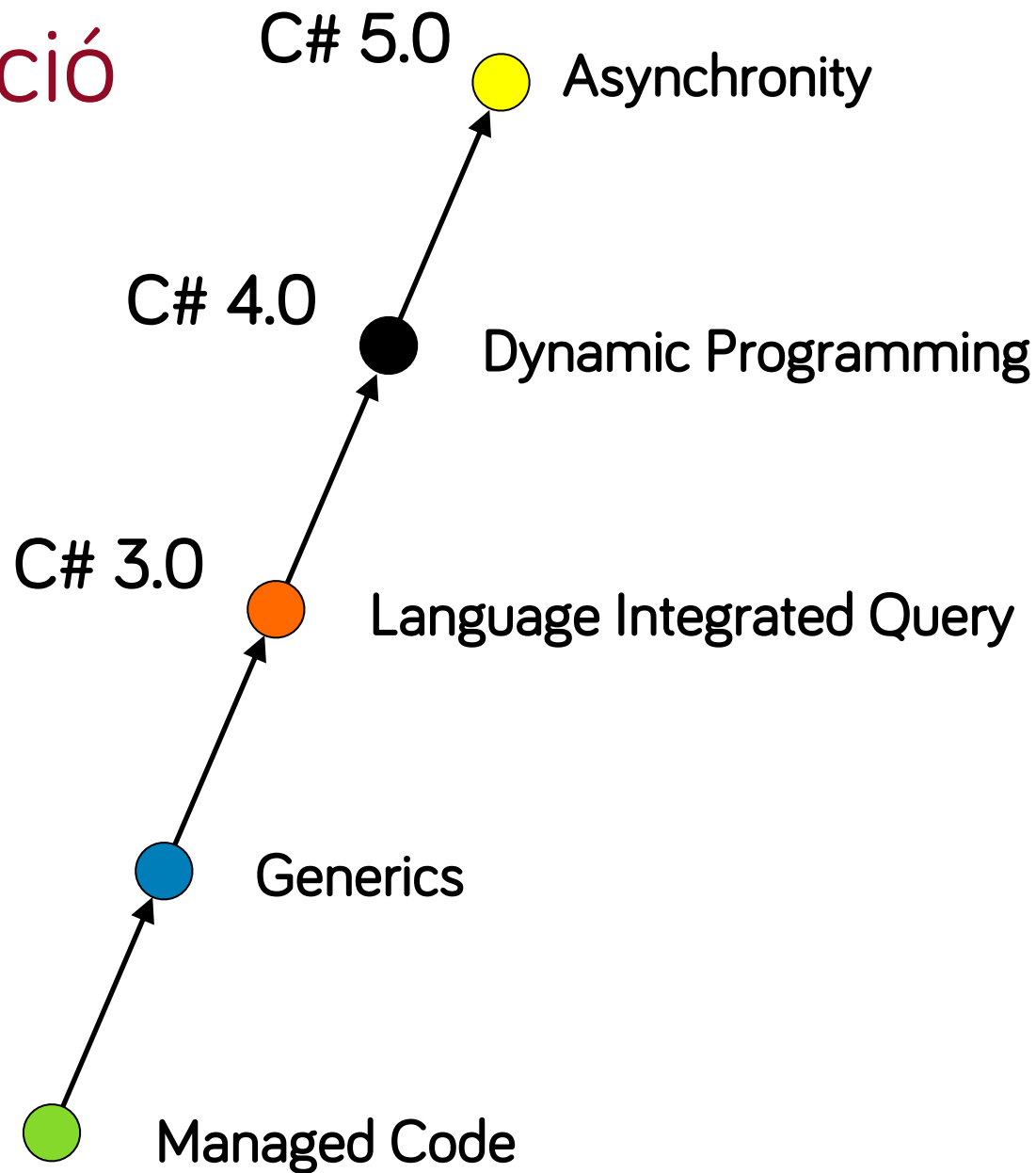
C# : Anders Hejlsberg és a többiek



Trendek



Evolúció



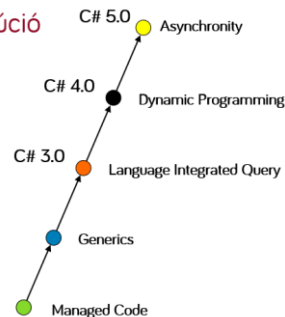
Tartalom

C# 4

1. Dinamikusan tipizált objektumok
2. Opcionális és nevesített paraméterek
3. COM Interop továbbfejlesztése



Evolúció



C# 6

1. null ellenőrzés
2. Automatikus tulajdonság inicializálás
3. Csak olvasható automatikus tulajdonságok
4. nameof operátor
5. Tulajdonság/metódus kifejezések
6. String interpoláció
7. Static type using
8. Gyűjtemény inicializáció
9. Await catch/finally-ban
10. Kivétel kezelő szűrők



C# 7

1. Binary Literals && Digit Separators
2. `out var`
3. `out *` (maybe)
4. Pattern Matching
5. Tuples
6. Local `functions`
7. More Expression Bodied Members



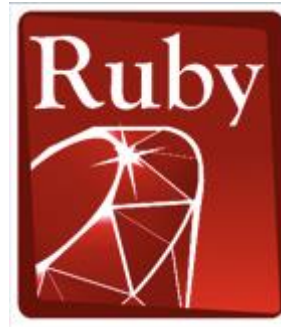
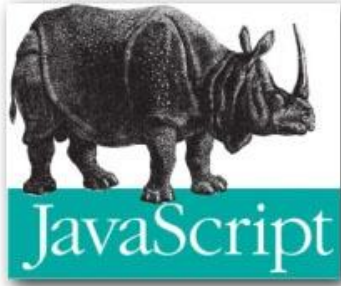
Kérdések?

Albert István
ialbert@aut.bme.hu

C# 4

1. Dinamikusan tipizált objektumok
2. Opcionális és nevesített paraméterek
3. COM Interop továbbfejlesztése

Dinamikus nyelvek



‘Dynamic’

- Dinamikus nyelvek:

A program struktúrája megváltozhat futás-időben

- > Új modulok, típusok jelennek meg
- > Metódusok, mezők, események jönnek létre és törlődnek...

- Dinamikusan tipizált nyelvek:

A deklarált változó típusa változhat meg végrehajtás közben

Miért használjunk dinamikus nyelveket?

- Gyorsabb prototipizálás
 - > Nincs a kódolás/fordítás/futtatás/debuggolás ciklus
 - > Nincs „felesleges” körbeprogramozás: névterek, osztályok, argumentumok stb.
- Tipikusan interaktív konzol segítségével
- A fordító végzi a munka „nehezét”: a típusok követését stb.

Több érték visszaadása C#-ban és Pythonban

- C#

```
object[] results = new object[2];  
results[0] = 42;  
results[1] = "Hello";  
return results;  
  
return new object [] { 42, "Hello" };
```

```
object[] results = GetValues();  
int x = (int)results[0];  
string s = (string)results[1];
```

- Python

```
def get_values() : return 42, "Hello"  
x, s = get_values()
```

Két változó felcserélése C#-ban és Pythonban

- C#

1 reference

```
void Swap(ref int x, ref int y)  
{ int temp = x; x = y; y = temp; }
```

```
int x = 4, y = 2;  
Swap(ref x, ref y);
```

- Python

```
x, y = 4, 2  
x, y = y, x
```

Dinamikus vs. statikus nyelvek

Dinamikus nyelvek

Egyszerű

Implicit tipizált

Meta programozás

Nincs fordítás

Statikus nyelvek

Robosztus

Gyors

Intelligens eszközök

Jobb skálázódás

Mit ad a C# 4 ?

- A C# nem dinamikus nyelv (és sose lesz az!)
- De jó lenne, ha támogatna néhány szenáriót:
 - > Egyszerű dinamikus problémák megoldása
 - Dinamikus overload feloldás az argumentum futásidejű típusa alapján
 - `object a = 12; cw(a); // hívja meg a cw(int a) overloadot`
 - > Jobb integráció COM-mal
 - Ne kelljen castolni, reflectiont használni
 - > Jobb együttműködés dinamikus nyelvekkel
 - A .NET készen áll a nyelvi együttműködésre!
- Megoldás: dinamikusan tipizált objektumok

Dinamikusan tipizált objektumok

```
Calculator calc = GetCalculator();  
int sum = calc.Add(10, 20);
```

```
object calc = GetCalculator();  
Type calcType = calc.GetType();  
object res = calcType.InvokeMember("Add",  
    BindingFlags.InvokeMethod, null,  
    new object[] { 10, 20 });  
int sum = Convert.ToInt32(res);
```

```
ScriptObject calc = GetCalculator();  
object res = calc.Invoke("Add", 10, 20);  
int sum = Convert.ToInt32(res);
```

*Statikusan
megadott
dinamikus típus*

```
dynamic calc = GetCalculator();  
int sum = calc.Add(10, 20);
```

Dinamikus
konverzió

Dinamikus metódus
hívás

Dinamikusan tipizált objektumok

*Fordítás idejű
dinamikus típusú
objektum*

*Futás idejű típusa:
System.Int32*

```
dynamic x = 1;  
dynamic y = "Hello";  
dynamic z = new List<int> { 1, 2, 3 };
```


Dinamikusan tipizált objektumok

```
...public static class Math
{
    ...public const double PI = 3.1415926535897931;
    ...public const double E = 2.7182818284590451;

    ...public static decimal Abs(decimal value);
    ...public static double Abs(double value);
    ...public static float Abs(float value);
    ...public static int Abs(int value);
    ...public static short Abs(short value);
    ...public static sbyte Abs(sbyte value);
    ...public static long Abs(long value);
}
```

Fordítás időben
választott metódus:
double Abs(double x)

```
double x = 1.75;
double y = Math.Abs(x);
```

```
dynamic x = 1.75;
dynamic y = Math.Abs(x);
```

```
dynamic x = 2;
dynamic y = Math.Abs(x);
```

Futás időben választott
metódus:
double Abs(double x)

Futás időben választott
metódus:
int Abs(int x)

C# 4.0 – DLR

QUIZ

- Mi a különbség, melyik választ az azonos nevű metódusok közül *futás időben*?

```
dynamic a = "apple";  
int l = a.Length;  
Console.WriteLine(l);
```

```
dynamic a = "apple";  
Console.WriteLine(a.Length);
```

```
string a = "apple";  
Console.WriteLine(a.Length);
```

Hogyan működik?

- A DLR-t használja (Dynamic Language Runtime)
 - > Együttműködés ...
 - A dinamikus nyelvek egymással
 - Statikus nyelvekkel és a BCL-lel
 - > Közös kód más dinamikus nyelvekkel
- Bővíthető koncepció...

A DLR fő alaposztálya: DynamicObject

```
...public class DynamicObject : IDynamicMetaObjectProvider
{
    ...protected DynamicObject();

    ...public virtual IEnumerable<string> GetDynamicMemberNames();
    ...public virtual DynamicMetaObject GetMetaObject(Expression parameter);
    ...public virtual bool TryBinaryOperation(BinaryOperationBinder binder, object arg, out object result);
    ...public virtual bool TryConvert(ConvertBinder binder, out object result);
    ...public virtual bool TryCreateInstance(CreateInstanceBinder binder, object[] args, out object result);
    ...public virtual bool TryDeleteIndex>DeleteIndexBinder binder, object[] indexes);
    ...public virtual bool TryDeleteMember>DeleteMemberBinder binder);
    ...public virtual bool TryGetIndex>GetIndexBinder binder, object[] indexes, out object result);
    ...public virtual bool TryGetMember>GetMemberBinder binder, out object result);
    ...public virtual bool TryInvoke>InvokeBinder binder, object[] args, out object result);
    ...public virtual bool TryInvokeMember>InvokeMemberBinder binder, object[] args, out object result);
    ...public virtual bool TrySetIndex>SetIndexBinder binder, object[] indexes, object value);
    ...public virtual bool TrySetMember>SetMemberBinder binder, object value);
    ...public virtual bool TryUnaryOperation(UnaryOperationBinder binder, out object result);
}
```

DynamicObject megvalósítások

- Alapértelmezett nyelvi implementációk
 - > Pont olyan tagok vannak, mint az éppen tárolt objektumnak
 - > Hibaüzenetek, feloldás logika stb. megegyezik
- COM interop
 - > Használja a COM saját reflexiós mechanizmusát
- ExpandoObject
 - > BCL implementáció, kényelmes PropertyBag

ExpandoObject: kényelmes szótár

- Futás időben bővíthető/változtatható objektum
- Tetszőleges tag hozzáadható
 - > metódus, esemény, ...
- Megvalósított interfészek: INPC, IDictionary, ICollection

```
dynamic exp = new ExpandoObject();

exp.Name = "PI";
exp.Age = 13;
exp.Name = 3.14;
exp.Print = (Action)(() =>
{ Console.WriteLine(exp.Name + ": " + exp.Age); });

exp.Print();
```

Az objektum
típusos tagjai
futás időben
változnak

```
34 dynamic exp = new ExpandoObject();
35
36 exp.Name = "PI";
37 exp.Age = 13;
38 exp.Name = 3.14;
39 exp.Print = (Action)(() =>
40 { Console.WriteLine(exp.Name + ": " + e
```

10 %

atch 1

Name	Value	Type
exp	{System.Dynamic.ExpandoObject}	dynamic {System.Dynamic.ExpandoObject}
exp, dynamic	Expanding the Dynamic View will get the dyna	
Name	"PI"	System.String

```
37 exp.Age = 13;
38 exp.Name = 3.14;
39 exp.Print = (Action)(() =>
40 { Console.WriteLine(exp.Name + ": " + exp.Age); });
41
42 exp.Print(); ≤ 8ms elapsed
43 }
```

%

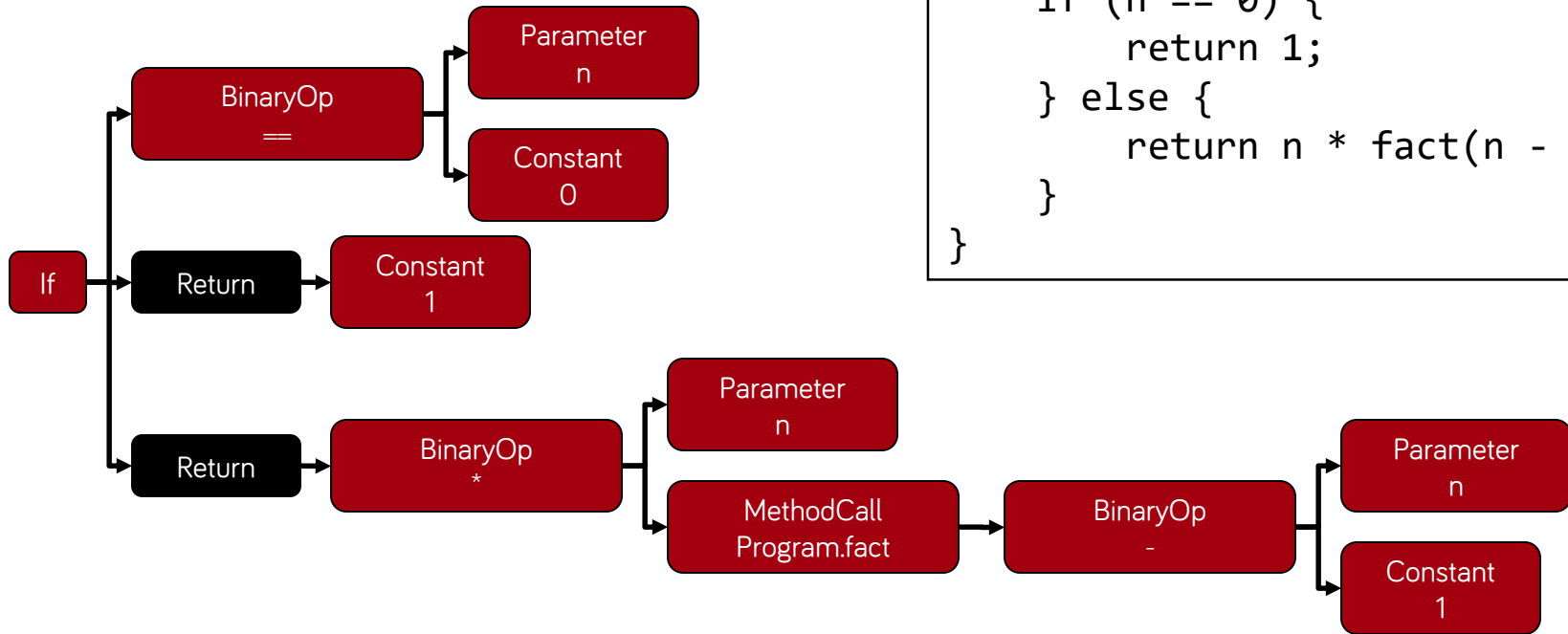
tch 1

ame	Value	Type
exp	{System.Dynamic.ExpandoObject}	dynamic {System.Dynamic.ExpandoObject}
exp, dynamic	Expanding the Dynamic View will get the dyna	
Age	13	System.Int32
Name	3.14	System.Double
Print	{System.Action}	System.Action

Mikor használjuk?

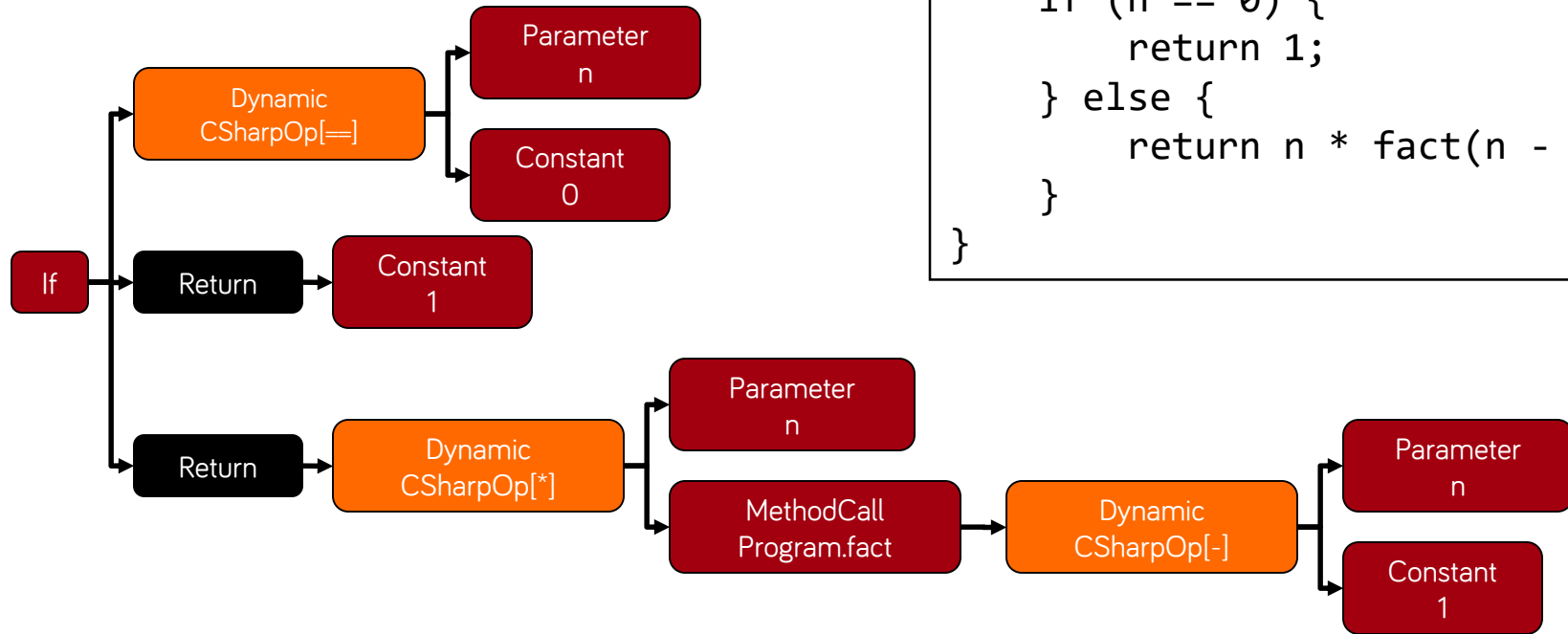
- RITKÁN!
- Meta-programozás
 - > Addinek interfészén
 - > XML/json adatok kezelése
 - > Félig strukturált adatok esetén

Faktoriális C#-ban



```
static int fact(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * fact(n - 1);
    }
}
```

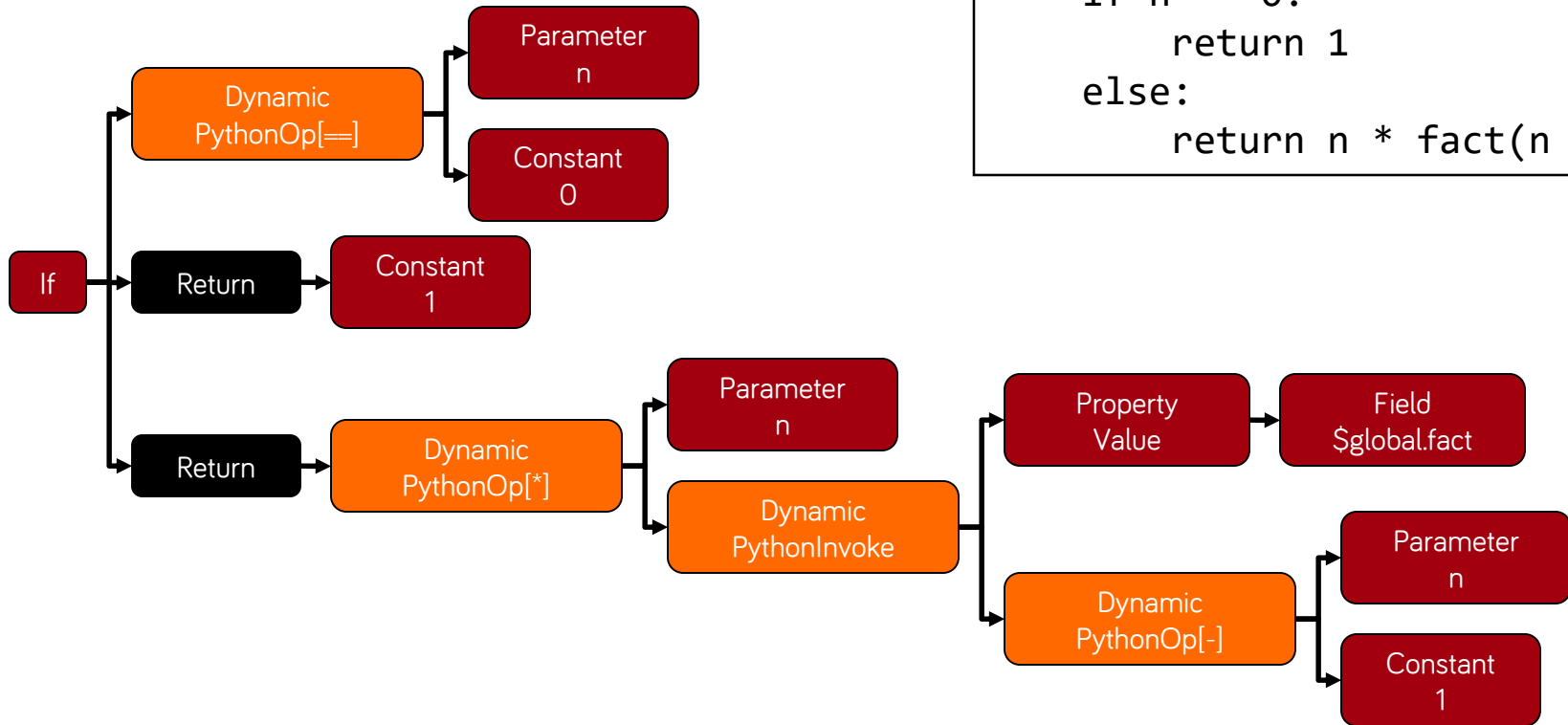
Faktoriális C#-ban, dinamikusan



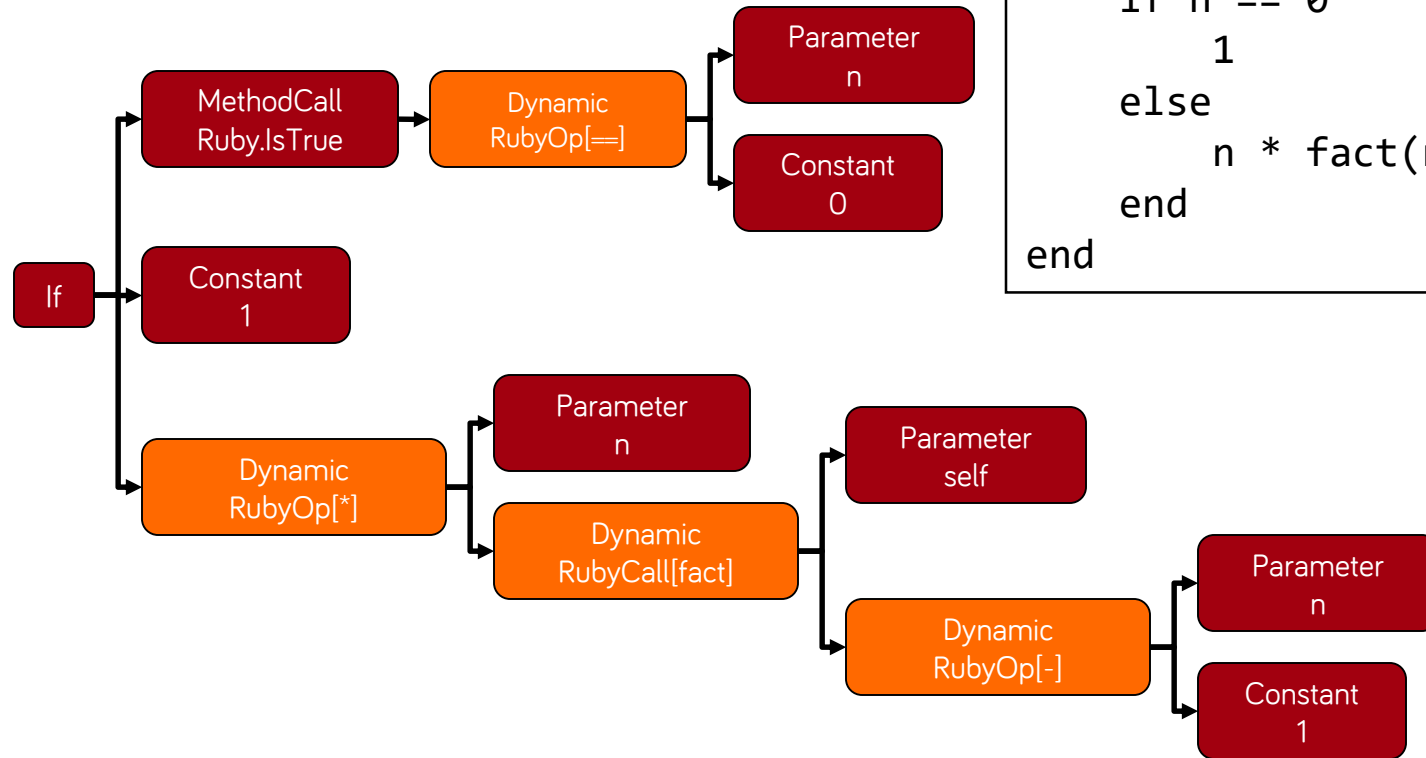
```
static dynamic fact(dynamic n) {
    if (n == 0) {
        return 1;
    } else {
        return n * fact(n - 1);
    }
}
```

Faktoriális Pythonban

```
def fact(n):  
    if n == 0:  
        return 1  
    else:  
        return n * fact(n - 1)
```



Faktoriális Rubyban



```
def fact(n)
  if n == 0
    1
  else
    n * fact(n - 1)
  end
end
```

DLR AST

- AST: Abstract Syntax Tree
- AST később IL-re fordul
- ActionExpression node: egy művelet
 - > Metódus hívás, indexelés stb.
 - > Az ActionExpressionök dinamikus hívási helyekre (call site) fordulnak
- Dinamikus hívási hely: cache-elési mechanizmus
 - > Adott típusokhoz tartozó metódushívást tárol el

Dinamikus hívási hely

- Elsődleges teljesítmény probléma
 - > Minden hívásnál meg kell keresni a megfelelő metódust
 - A paraméterek típusa megváltozhat, ami miatt egy másik metódust kellene hívni
- A statikus fordítás minden ilyen hívást egy dinamikus hívási helyé fordít
 - > A legutolsó metódushívást cache-eli és csak akkor keres, ha szükséges
 - > Típus váltáskor a cache kiürül


FastDynamicSite . UpdateBindingAndInvoke


- Ez fut le minden hívásnál
 1. Az aktuális delegate meghívódik
 2. A delegate ellenőrzi, hogy a típusok azonosak-e az előző híváshoz képest (amihez a delegate elkészült)
 3. Ha IGEN => meghívja a metódust
 4. Ha NEM => call UpdateBindingAndInvoke:
 1. Megkeresi a megfelelő metódust (a nyelv specifikus keresővel - binder)
 2. Létrehozza az új metódust
 3. Az aktuális delegate-et lecseréli az újra

CallSite<T>

```
if (x == 0) { ... }
```

```
static CallSite<Func<CallSite, object, int, bool>> _site = ...;
```

```
if (_site  _site, x, 0)) { ... }
```



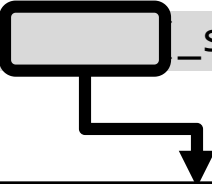
```
static bool _0(Site site, object x, int y) {  
    return site.Update(site, x, y); //tailcall  
}
```


CallSite<T>

```
if (x == 0) { ... }
```

```
static CallSite<Func<CallSite, object, int, bool>> _site = ...;
```

```
if (_site  _site, x, 0)) { ... }
```




```
static bool _1(Site site, object x, int y) {  
    if (x is int) {  
        return (int)x == y;  
    } else {  
        return site.Update(site, x, y); //tailcall  
    }  
}
```

CallSite<T>

```
if (x == 0) { ... }
```

```
static CallSite<Func<CallSite, object, int, bool>> _site = ...;
```

```
if (_site  _site, x, 0)) { ... }
```



```
static bool _2(Site site, object x, int y) {  
    if (x is int) {  
        return (int)x == y;  
    } else if (x is BigInteger) {  
        return BigInteger.op_Equality((BigInteger)x, y);  
    } else {  
        return site.Update(site, x, y); //tailcall  
    }  
}
```

JVM vs DLR + CLR

- Nincs VM támogatás dinamikus nyelvekhez a Microsoft .NET platformon
 - > A DLR a CLR fölött dolgozik
- A Java VM (Da Vinci Machine) natív támogatással rendelkezik
- Állítólag a Microsoft is tervezi... (évek óta)
- A dinamikus kód generálás miatt talán nem feltétlenül jelentene komolyabb előnyt

C# 4.0 nyelvi fejlesztések

1. Dinamikusan tipizált objektumok
2. Opcionális és nevesített paraméterek
3. COM Interop továbbfejlesztése

Opcionális és nevesített paraméterek

```
public StreamReader OpenTextFile(  
    string path,  
    Encoding encoding,  
    bool detectEncoding,  
    int bufferSize);
```

Elsődleges metódus

```
public StreamReader OpenTextFile(  
    string path,  
    Encoding encoding,  
    bool detectEncoding);
```

Másodlagos overload

```
public StreamReader OpenTextFile(  
    string path,  
    Encoding encoding);
```

Meghívja az
elsődlegesét default
értékekkel

```
public StreamReader OpenTextFile(  
    string path);
```

Opcionális és nevesített paraméterek

```
public StreamReader OpenTextFile(  
    string path,  
    Encoding encoding = null,  
    bool detectEncoding = true,  
    int bufferSize = 1024);
```

Opcionális
paraméterek

```
OpenTextFile("foo.txt", Encoding.UTF8);
```

Nevesített paraméterek

```
OpenTextFile("foo.txt", Encoding.UTF8, bufferSize: 4096);
```

A nevesített
paraméterek sorrendje
mindegy

A paraméterek
kiértékelés a leírás
sorrendje

A nevesített
paraméterek jönnek a
hívás végére

```
OpenTextFile(  
    bufferSize: 4096,  
    path: "foo.txt",  
    detectEncoding: false);
```

Nem-opcionálist
specifikálni kell

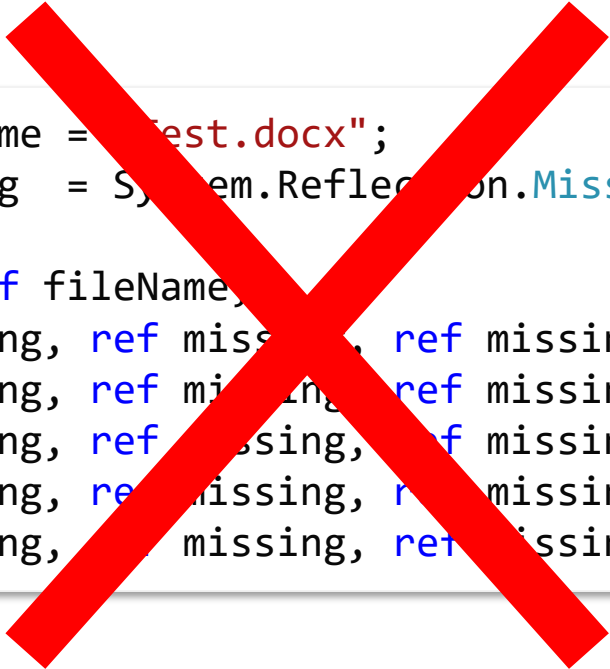
Nem csak nyelvi feature

- Hosszú ideje várunk rá
 - > A CLI és VB már régóta támogatja
- `void ThrowBumeraang`
 `(int distance, bool shouldReturn = true);`
 - > Csak konstans használható
 - > Az alapértelmezett értéket a metaadat tábla tartalmazza
 - > A hívási hely mindig az összes paramétert meg fogja adni – az alapértelmezett értékeket kiolvassa a metaadatok közül
 - Miért? => verzionálás!

C# nyelvi fejlesztések

1. Dinamikusan tipizált objektumok
2. Opcionális és nevesített paraméterek
3. COM Interop továbbfejlesztése

Javított COM együttműködés



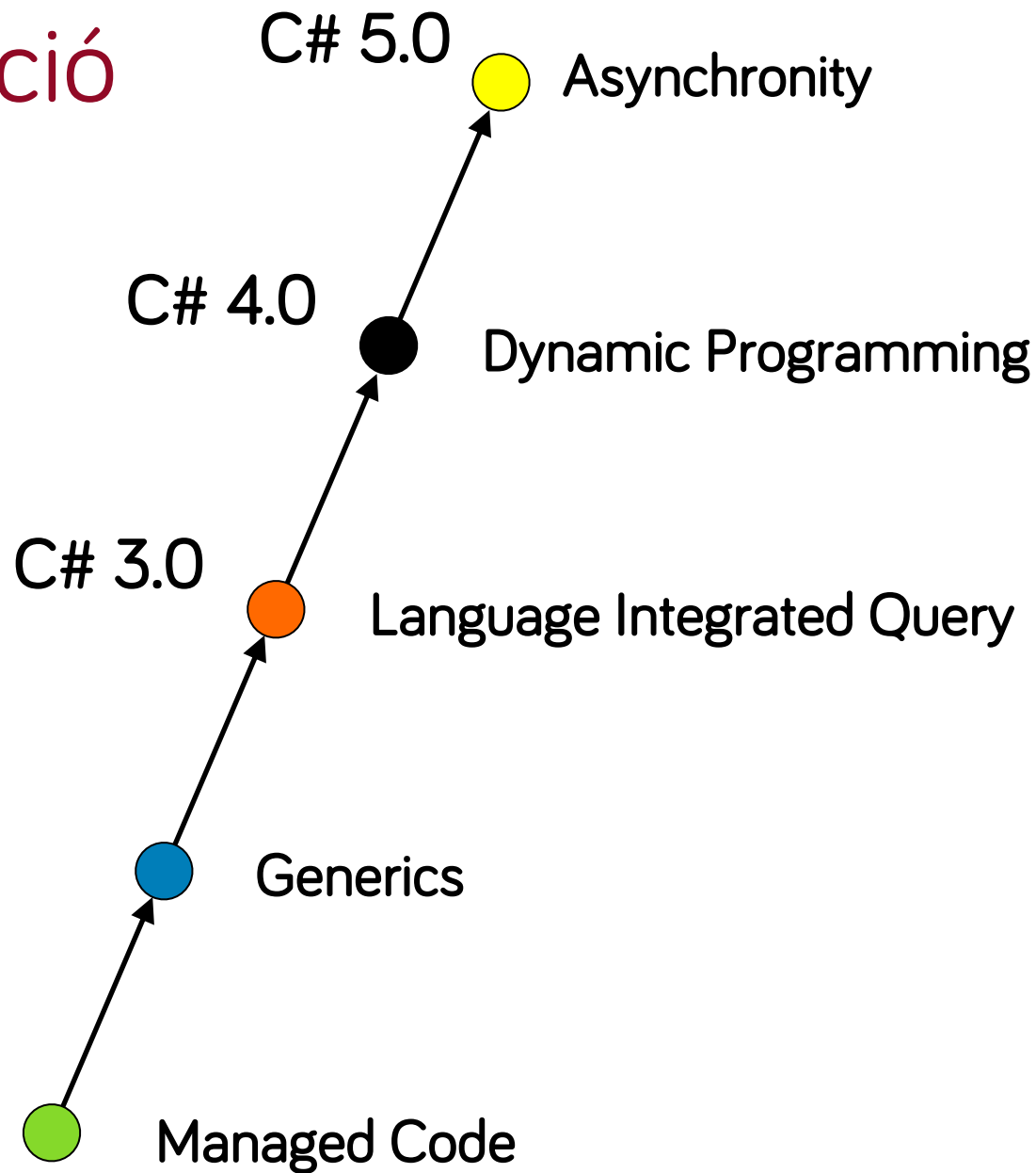
```
object fileName = "Test.docx";  
object missing = System.Reflection.Missing.Value;  
  
doc.SaveAs(ref fileName,  
    ref missing, ref missing, ref missing,  
    ref missing, ref missing, ref missing,  
    ref missing, ref missing, ref missing,  
    ref missing, ref missing, ref missing);
```

```
doc.SaveAs("Test.docx");
```

Javított COM együttműködés

- Automatic object → dynamic leképezés
- Opcionális és nevesített paraméterek
- Indexerek
- Opcionális ref módosítók
- Interop típusok beágyazása (“No PIA”)
 - > Új típusegyezőség vizsgálat van a CLR-ben...

Evolúció



Miért fontos az aszinkronitás?

- Egyre több hálózati alkalmazás
 - Több késleltetés
 - Több UI ,responsiveness' probléma
 - Több skálázási probléma
- Aszinkron programozás
 - Reszponzív, skálázható alkalmazások alapja
 - Csak aszinkron API-k: JavaScript, Silverlight, **WinRT**

Mitől aszinkron?

- Szinkron → Megvárjuk a visszatérési értéket/választ
 - `string DownloadString(...);`
- Aszinkron → Rögtön visszatér, későbbi feldolgozás
 - `void DownloadStringAsync(..., Action<string> callback);`
- Előnyök
 - UI responsiveness: UI szál válaszol a felhasználónak
 - Szerver skálázódás: a szálát más kérések használhatják
- Van összefüggés a párhuzamossággal és többszálúsággal de nem ugyanaz!
 - Például UI szálon is lehet aszinkron művelet

Szinkron vs. aszinkron

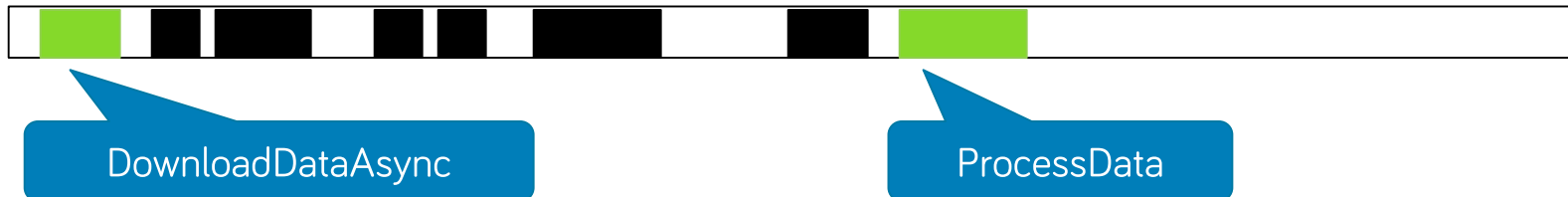
```
var data = DownloadData(...);  
ProcessData(data);
```

Szál



```
DownloadDataAsync(... , data => {  
    ProcessData(data);  
});
```

Szál



Jelenlegi megoldások nehézkeseek

- Háttér szálak használata
- Explicit szálkezelés
- Értesítési mechanizmusok
- Versenyhelyzetek
- **Kódszétदारabolódás**

Ma 1: APM modell

- *BeginMethodName*, *EndMethodName* metódusok
- *IAsyncResult*, *AsyncCallback* értesítési mechanizmus
- Folyamat és megszakítás nehezen implementálható
- Nehezen kombinálható

```
public TResult DownloadData(TParam parameter);  
  
...  
  
public IAsyncResult BeginDownloadData  
    (TParam parameter, AsyncCallback callback, object state);  
  
public TResult EndDownloadData (IAsyncResult asyncResult);
```


Szinkron példa

```
... method1()  
{  
    GetResult();  
    // hibakezeles  
    Textbox1.Text = ...;  
}
```

Aszinkronitás manuálisan

```
... method1()  
{  
    GetResultCompleted += callback1;  
    GetResultAsync( ... );  
    // kilep a metodus  
}  
  
... callback1() ←  
{  
    // háttérszálon fut  
    // hibakezeles stb..  
    Dispatcher( ..., showResult); // áthívás UI szálra  
}  
  
... showResult() ←  
{  
    Textbox1.Text = ...;  
}
```



Ma 2: EAP modell

- Még összetettebb megoldás
- Folyamatjelzés és megszakítás támogatás
- Kivételkezelés
- Nehezen kombinálható

```
public TResult DownloadData(TParam param);  
...  
public void DownloadDataAsync(TParam parameter);  
public event DownloadDataCompletedEventHandler DownloadDataCompleted;  
  
public delegate void DownloadDataCompletedEventHandler(  
    object sender, DownloadDataCompletedEventArgs eventArgs);  
  
public class DownloadDataCompletedEventArgs : AsyncCompletedEventArgs  
{  
    public TResult Result { get; }  
}
```

Szinkron példa

```
... method1()  
{  
    GetResult();  
    // hibakezeles  
    Textbox1.Text = ...;  
}
```

Aszinkron példa

```
... method1()  
{  
    GetResultCompleted += callback1;  
    GetResultAsync( ... );  
    // kilep a metodus  
}  
  
... callback1()  
{  
    // hibakezeles  
    Dispatcher( ..., showResult ); // UI szal  
}  
  
... showResult()  
{  
    Textbox1.Text = ...;  
}
```

Taszkok

- Task osztály: az egységnyi, párhuzamosan vagy aszinkron végrehajtható feladatot tartja nyilván
 - > Van saját egyedi azonosítója (Id)
 - > Lekérdezhető a feladat státusza
 - > A dobott kivétel
 - > Lehet várni a befejezésére (egyre vagy többre)
 - > Folytatások vagy hibakezelő felfűzése
- A Task<TResult> olyan feladat, aminek van visszatérési értéke, a Task-ból származik
 - > Típusosan lekérdezhető az eredmény
- Létrehozás new-val vagy TaskFactory osztállyal

Taszk példa

```
// Create a task
var taskA = new Task(() =>
    Console.WriteLine("Hello from taskA.));

// Start the task.
taskA.Start();

// OR USE THE FACTORY METHOD:
// var taskA = Task.Factory.StartNew(() =>
//     Console.WriteLine("Hello from taskA.));

// Output a message from the joining thread.
Console.WriteLine("Hello from the calling thread.");

/* Output:
 * Hello from the calling thread.
 * Hello from taskA.
 */
```

Folytatásos feladatok

- A feladatok láncba fűzhetőek, az egyik lefutása után automatikusan elindulhat a következő
 - > A egyik művelet eredménye lehet a következő bemenő paramétere
- Több előzmény taszkhhoz is kapcsolható folytatás
 - > Any és All támogatás
 - > Az összes előzmény eredményt megkapja
- Speciális folytatások is beállíthatók a TaskContinuationOptions paraméterrel
 - > Például hiba vagy abortálás esetén más-más folytatás fusson le

Folytatásos taszkok

```
try
{
    var firstTask = new Task(
        () => CopyDataIntoTempFolder(path));

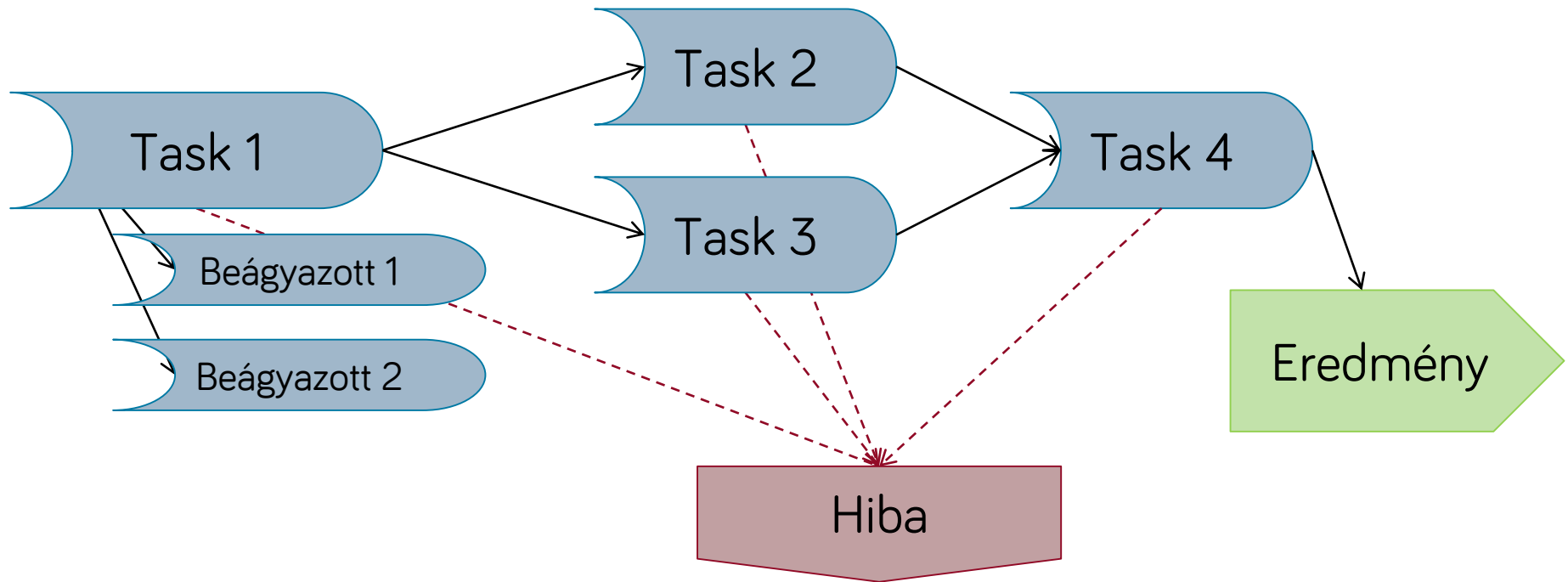
    var secondTask = firstTask.ContinueWith((t) =>
        CreateSummaryFile(path));

    firstTask.Start();
    //...
}
catch (AggregateException e)
{
    Console.WriteLine(e.Message);
}
```

Beágyazott taszkok

- Taszk végrehajtása során létrehozott új műveletet hozzá lehet kötni az éppen futó taszkhhoz
 - `TaskCreationOptions.AttachedToParent`
- A szülő taszk akkor ér véget, amikor az összes gyerek taszk véget ér

Összetett művelet háló



Kivétel kezelés

- A taszkok által dobott kivételek bekerülnek egy AggregateException kivételbe (InnerExceptions)
- Wait hívások, Result property elérése dobja a kivételt
- A Task . Exception propertyn keresztül is lekérdezhető a kivétel
- Ha a Task példány úgy kerül finalizálásra, hogy a fentiek egyike sem hívódik, akkor a teljes folyamat leáll (mint más kivételeknél)

Kivétel kezelés példa

```
var task1 = Task.Factory.StartNew(() =>
{
    throw new MyCustomException("baj van!");
});

try
{
    task1.Wait();
}
catch (AggregateException ae)
{
    // Rethrow anything
    foreach (var e in ae.InnerExceptions)
    {
        if (e is MyCustomException)
            Console.WriteLine(e.Message);
        else
            throw;
    }
}
```

Taszkok megszakítása

- CancellationToken alapú (TPL-től független)
 - > CancellationTokentSource osztály hozza létre a token
 - > A Task példányokhoz a token hozzárendelhető
- Kooperatív: a taszkon belül kell figyelni a token
 - > Polling
 - > Wait...
 - > Callback regisztráció
 - > Több token összeköthető egy tokenbe
- OperationCanceledException kivételt kell dobni
 - > Ebből tudja a TPL, hogy megszakították a taszkot

Művelet megszakítás

```
static void CancelWithThreadPoolMiniSnippet()
{
    //Thread 1:The Requestor - Create the token source.
    CancellationTokenSource cts = new CancellationTokenSource();
    ThreadPool.QueueUserWorkItem(DoSomeWork, cts.Token);

    // Request cancellation
    cts.Cancel();
}
static void DoSomeWork(object obj)
{
    CancellationToken token = (CancellationToken)obj;
    for (int i = 0; i < 100000; i++)
    {
        Thread.SpinWait(5000000); // Simulating work.
        if (token.IsCancellationRequested)
            throw new OperationCanceledException(token);
    }
}
```

Aszinkronitás manuálisan

```
... method1()  
{  
    GetResultCompleted += callback1;  
    GetResultAsync( ... );  
    // kilep a metodus  
}  
  
... callback1() ←  
{  
    // háttérszálon fut  
    // hibakezeles stb..  
    Dispatcher( ..., showResult); // áthívás UI szálra  
}  
  
... showResult() ←  
{  
    Textbox1.Text = ...;  
}
```



Asznkronitás nyelvi szinten

- A fenti kód az új async és await kulcsszavakkal

```
private async void btnDoWork_Click(...)
{
    try
    {
        int result = 0;
        await ThreadPool.RunAsync(
                                delegate { result = Compute(); });
        btnDoWork.Content = result.ToString();
    }
    catch (Exception exc)
    {
        btnDoWork.Content = exc.Message;
    }
}
```

Másik példa ThreadPool nélkül

```
double calculatePI()  
{  
    double pi = 2;  
    const int n = 50;  
    for (int i = n; i > 0; i--)  
        pi = pi * i / (2 * i + 1) + 1;  
    return 2* pi;  
}
```

1 reference

```
private async void Page_Loaded(object sender, RoutedEventArgs e)  
{  
    double result = await Task.Run((Func<double>)calculatePI);  
    txtResult.Text = result.ToString();  
}
```

A nyelvi megoldás előnyei

- Nem kell manuálisan visszahívni a UI szálba
- Nem kell az állapotot hívásonként feldolgozni, hanem használható szokásos a kivételkezelés
- A program struktúrája olvasható marad, nem kell callbackekre feldarabolni
- Az aszinkron hívások egymás után fűzhetők, jól komponálódnak
- A metódus az awaitek mentén feldarabolódik

Aszinkron nyelvi elemek

- “**async**” módosító kulcsszó
 - > A metódus vagy lambda kifejezés aszinkron módon futhat
 - > A fordító állapotgépet készít hozzá
- “**await**” művelet **átadhatja** a vezérlést
 - > Amíg az aszinkron taszk véget nem ér
 - > **await** csak **async** metódusokban használható

.NET 4.5-től az osztályok...

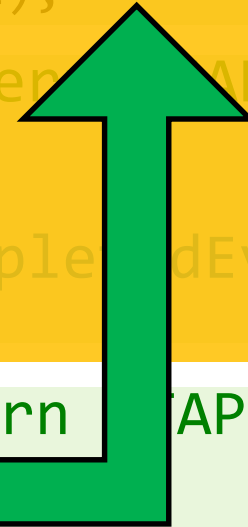
```
// Synchronous  
TResult Foo(...);
```

```
// Asynchronous Programming  
IAsyncResult BeginFoo(...,  
state);  
TResult EndFoo(IAsyncResult asyncresult);
```

```
// Event-based Asynchronous Pattern (EAP)  
public void FooAsync(...);  
public event EventHandler<FooCompletedEventArgs>  
FooCompleted;
```

```
// Task-based Asynchronous Pattern (TAP)  
Task<TResult> FooAsync(...);
```

```
System.IO.Stream:  
Task<int> ReadAsync(...);  
Task<int> WriteAsync(...);  
Task FlushAsync();  
Task<int> CopyToAsync(...);
```



Caller Info attribútumok (C# 5)

- Metódus paraméterre rakhatjuk
- A hívó metódus paramétereit kaphatjuk meg
- 3 attribútum
 - > CallerMember, CallerFilePath, CallerLineNumber
- Példa:

```
public void TraceMessage(string message,  
    [CallerMemberName] string memberName = "",  
    [CallerFilePath] string sourceFilePath = "",  
    [CallerLineNumber] int sourceLineNumber = 0) {  
    Trace.WriteLine("message: " + message);  
    Trace.WriteLine("member name: " + memberName);  
    Trace.WriteLine("source file path: " + sourceFilePath);  
    Trace.WriteLine("source line number: " + sourceLineNumber);  
}
```

Caller Info attriútumok

- INotifyPropertyChanged implementáció:

```
public class MyClass : INotifyPropertyChanged
{
    private string myText;
    public string MyText
    {
        get { return myText; }
        set { myText = value; NotifyPropertyChanged(); }
    }
    public void NotifyPropertyChanged(
        [CallerMemberName] string property = null)
    {
        if (PropertyChanged != null)
            PropertyChanged(this, new PropertyChangedEventArgs(property));
    }
    public event PropertyChangedEventHandler PropertyChanged;
}
```

C# 6

1. null ellenőrzés
2. Automatikus tulajdonság inicializálás
3. Csak olvasható automatikus tulajdonságok
4. nameof operátor
5. Tulajdonság/metódus kifejezések
6. String interpoláció
7. Static type using
8. Gyűjtemény inicializáció
9. Await catch/finally-ban
10. Kivétel kezelő szűrők

Null ellenőrzés

```
int v1()  
{  
    if (points != null)  
    {  
        var next = points.FirstOrDefault();  
        if (next != null && next.X != null)  
            return next.X;  
    }  
    return -1;  
}
```

```
int v1()  
{  
    var bestValue = points?.FirstOrDefault()?.X ?? -1;  
}
```

Null ellenőrzés delegate hívás esetén

```
If( PropertyChanged != null )
```

```
    PropertyChanged( this, args );
```

```
PropertyChanged?.Invoke( this, args );
```

Automatikus tulajdonság inicializálás

```
public class Person
{
    public string FirstName { get; set; } = „Alma”;
    public string LastName { get; set; } = „Csíkos”;
}
```

Csak olvasható automatikus tulajdonságok

```
private readonly int x;
```

```
public int X { get { return x; } }
```

```
public int X { get; } = x;
```

Property expressions

```
public double Distance {  
    get { return Math.Sqrt((X * X) + (Y * Y)); }  
}
```

```
public double Distance =>  
    Math.Sqrt((X * X) + (Y * Y));
```

Method expressions

```
public Point Move(int dx, int dy) {  
    return new Point(X + dx1, Y + dy1);  
}
```

```
public Point Move(int dx, int dy)  
    => new Point(X + dx, Y + dy);
```

Nameof operátor

```
void func(string param1) {  
    throw new ArgumentNullException( "param1" ); }  

```

```
void func(string param1) {  
    throw new ArgumentNullException(  
        nameof(param1) ); }  

```

String interpolation

```
var s = String.Format("{0} is {1} year{{s}} old",  
                      p.Name, p.Age);
```

```
var s = $"{p.Name} is {p.Age} year{{s}} old";
```

```
var s = $"{p.Name} is {p.Age} "  
        "year{(p.Age == 1 ? "" : "s")} old";
```


Static type using statements

```
public double A { get { return  
    Math.Sqrt(Math.Round(5.142)); } }
```

```
using System.Math;
```

```
...
```

```
public double A { get { return  
    Sqrt(Round(5.142)); } }
```

Gyűjtemény inicializáció

```
var numbers = new Dictionary<int, string> {  
    { 7, "hét" },  
    { 9, "kilenc" },  
    { 13, "tizenhárom" }  
};
```

```
var numbers = new Dictionary<int, string> {  
    [7] = "hét",  
    [9] = "kilenc",  
    [13] = "tizenhárom"  
};
```

Await catch/finally-ban

```
var input = new StreamReader(fileName);  
var log = new StreamWriter(logFileName);  
try {  
    var line = await input.ReadLineAsync();  
}  
catch (IOException ex) {  
    await log.WriteLineAsync(ex.ToString());  
}  
finally {  
    if (log != null) await log.FlushAsync();  
}
```

Kivétel kezelő szűrők

```
try
{
    person = new Person(„Csíkos", null);
}
catch(ArgumentNullException e) if(e.ParamName=="firstName")
{
    Console.WriteLine("First name is null");
}
catch(ArgumentNullException e) if(e.ParamName=="lastName")
{
    Console.WriteLine("Last name is null");
}
```

Kivétel kezelő szűrők például naplózáshoz

```
try { ... }
```

```
catch (Exception ex) if ( Log(ex) ) { ... }
```

```
private static bool Log(Exception exception)  
{  
    Console.WriteLine(exception.Message);  
    return false;  
}
```

C# 7

1. Binary Literals && Digit Separators
2. `out var`
3. `out *` (maybe)
4. Pattern Matching
5. Tuples
6. Local `functions`
7. More Expression Bodied Members

Binary Literals && Digit Separators

```
var flags = new int[00_01,  
                    00_20,  
                    03_00,  
                    40_00,  
                    0___0];
```

```
var binaryFlags = new int[0b000_001,  
                           0b000_010,  
                           0b000_100,  
                           0b001_000,  
                           0b010_000,  
                           0b100_000,  
                           0b111_111];
```

out var

```
var dict = new Dictionary<string, string>();

string value;
if (dict.TryGetValue("key", out value))
{
    Console.WriteLine(value);
}
```



```
var dict = new Dictionary<string, string>();

string value;
if (dict.TryGetValue("key", out var value))
{
    Console.WriteLine(value);
}
```


out * (maybe)

```
if (p.HasCoordinates(out var x, out *))    // Only x matters!  
{  
    ...  
}
```

Variable Extraction && is type Pattern Matching

```
public abstract class Animal { }

public class Dog : Animal
{
    public string Name { get; set; }

    public void BarkAt(Animal animal)
    {
        if (animal is Dog d)
            WriteLine($"Sniff-sniff {d.Name}!");
        else
            WriteLine("Bark!");
    }
}
```

Simple Variable Extraction Pattern Matching

```
public abstract class Animal { }

public class Dog : Animal
{
    public string Name { get; set; }

    public void BarkAt(Animal animal)
    {
        if (animal is var a)
            WriteLine($"Sniff-sniff {a}!");
        else
            WriteLine("Woof?!");    // Not happening!
    }
}
```

out && is Pattern Matching

```
if (o is int i || (o is string s && int.TryParse(s, out i))  
{  
    /* use i */  
}
```

Type switch Pattern Matching

```
switch (shape)
{
    case Circle c:
        WriteLine($"circle with radius {c.Radius}");
        break;
    case Rectangle s when (s.Length == s.Height):
        WriteLine($"{s.Length} x {s.Height} square");
        break;
    case Rectangle r:
        WriteLine($"{r.Length} x {r.Height} rectangle");
        break;
    default:
        WriteLine("<unknown shape>");
        break;
    case null:
        throw new ArgumentNullException(nameof(shape));
}
```

Tuple Types

```
// tuple return type
(string, string, string) LookupName(long id)
{
    // retrieve first, middle and last from data storage
    return (first, middle, last); // tuple literal
}
```

```
var names = LookupName(id);
WriteLine($"found {names.Item1} {names.Item3}.");
```

Tuple Literals

```
// tuple literal return type
(string first, string middle, string last) LookupName(long id)
{
    // retrieve first, middle and last from data storage
    return (first, middle, last); // tuple literal
}
```

```
var names = LookupName(id);
WriteLine($"found {names.first} {names.last}.");
```

Tuple Deconstruction

```
(string first, object middle, string last) = LookupName(id);  
WriteLine($"found {first} {last}.");
```

```
var (first, middle, last) = LookupName(id);
```


Type Deconstruction

```
class Point
{
    public int X { get; }
    public int Y { get; }

    public Point(int x, int y) { X = x; Y = y; }
    public void Deconstruct(out int x, out int y)
    {
        x = X;
        y = Y;
    }
}
```

```
// calls Deconstruct(out var myX, out var myY);
var (myX, myY) = new Point(2, 3);
```

```
// calls Deconstruct(out var myX, out *);
var (myX, *) = new Point(2, 3);
```

Local functions

```
public int Fibonacci(int x)
{
    if (x < 0)
        throw new ArgumentException("Less negativity!", nameof(x));

    return Fib(x).current;

    (int current, int previous) Fib(int i)
    {
        if (i == 0) return (1, 0);
        var (p, pp) = Fib(i - 1);
        return (p + pp, p);
    }
}
```

More Expression Bodied Members

```
class Person
{
    public Person(string name) => names.TryAdd(id, name);

    ~Person() => names.Remove(id, out *);

    public string Name
    {
        get => names[id];
        set => names[id] = value;
    }
}
```

throw Expressions

```
class Person
{
    public string Name { get; }
    public Person(string name)
        => Name = name ?? throw new ArgumentNullException(name);

    public string GetFirstName()
    {
        var parts = Name.Split(' ');
        return (parts.Length > 0)
            ? parts[0]
            : throw new InvalidOperationException("No name!");
    }

    public string GetLastName()
        => throw new NotImplementedException();
}
```