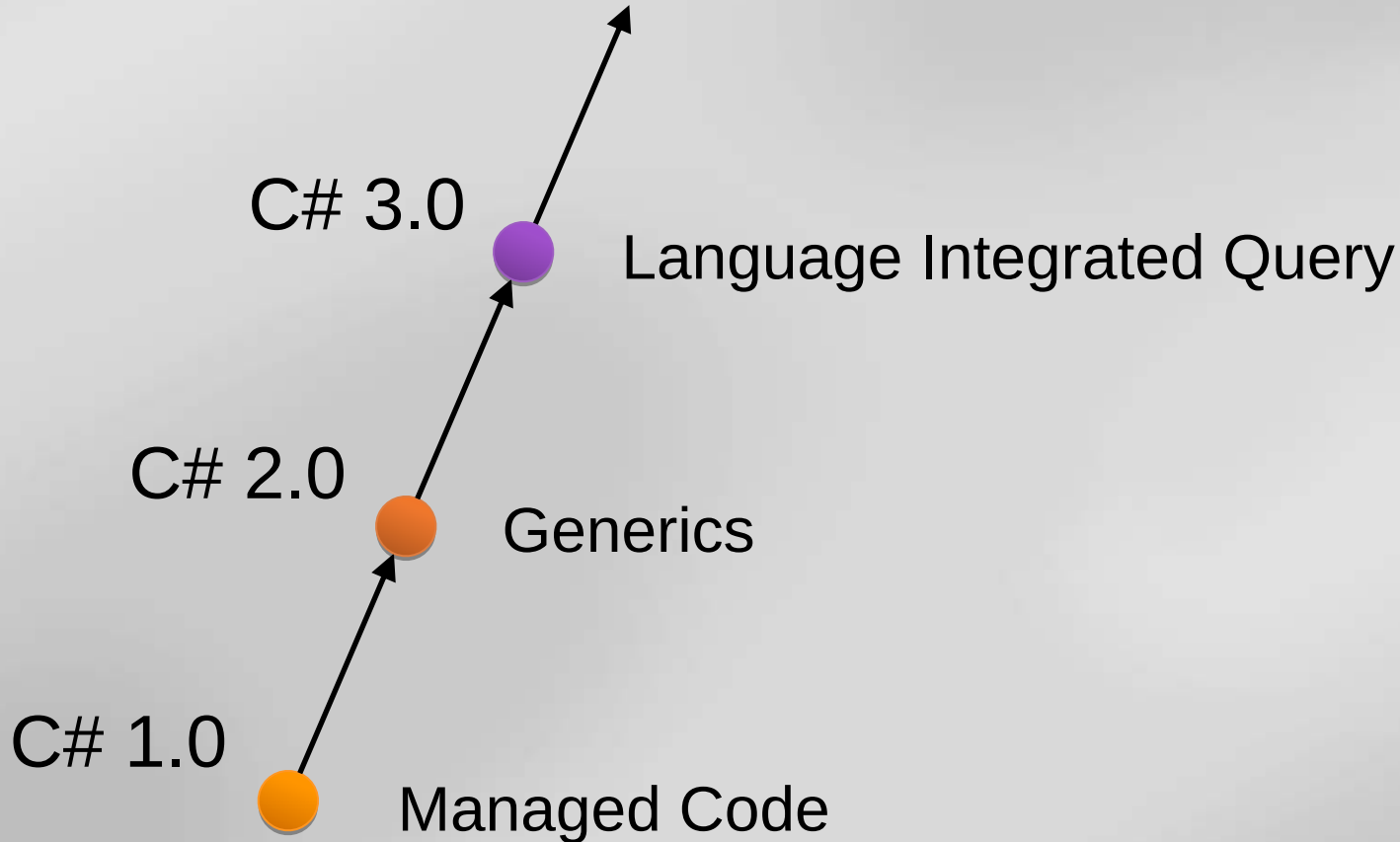
Microsoft

Microsoft tech·days 09
Know-how to win

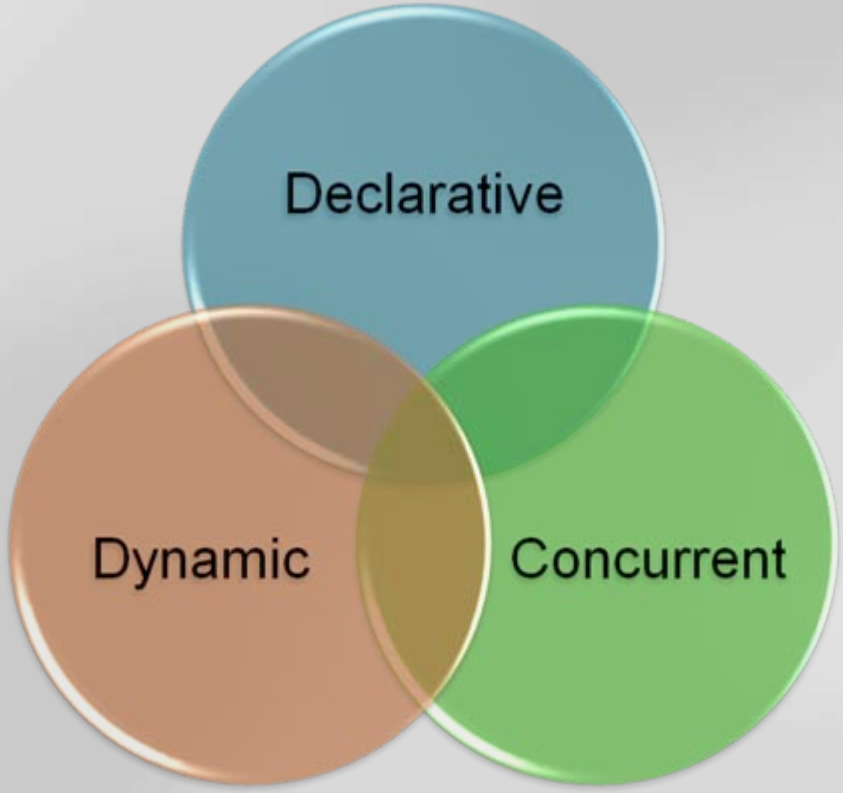
What's new in C# 4.0?

Ken Casada
Developer Evangelist
Microsoft Switzerland
kcasada@microsoft.com
http://blogs.msdn.com/swiss_dpe_team/Default.aspx

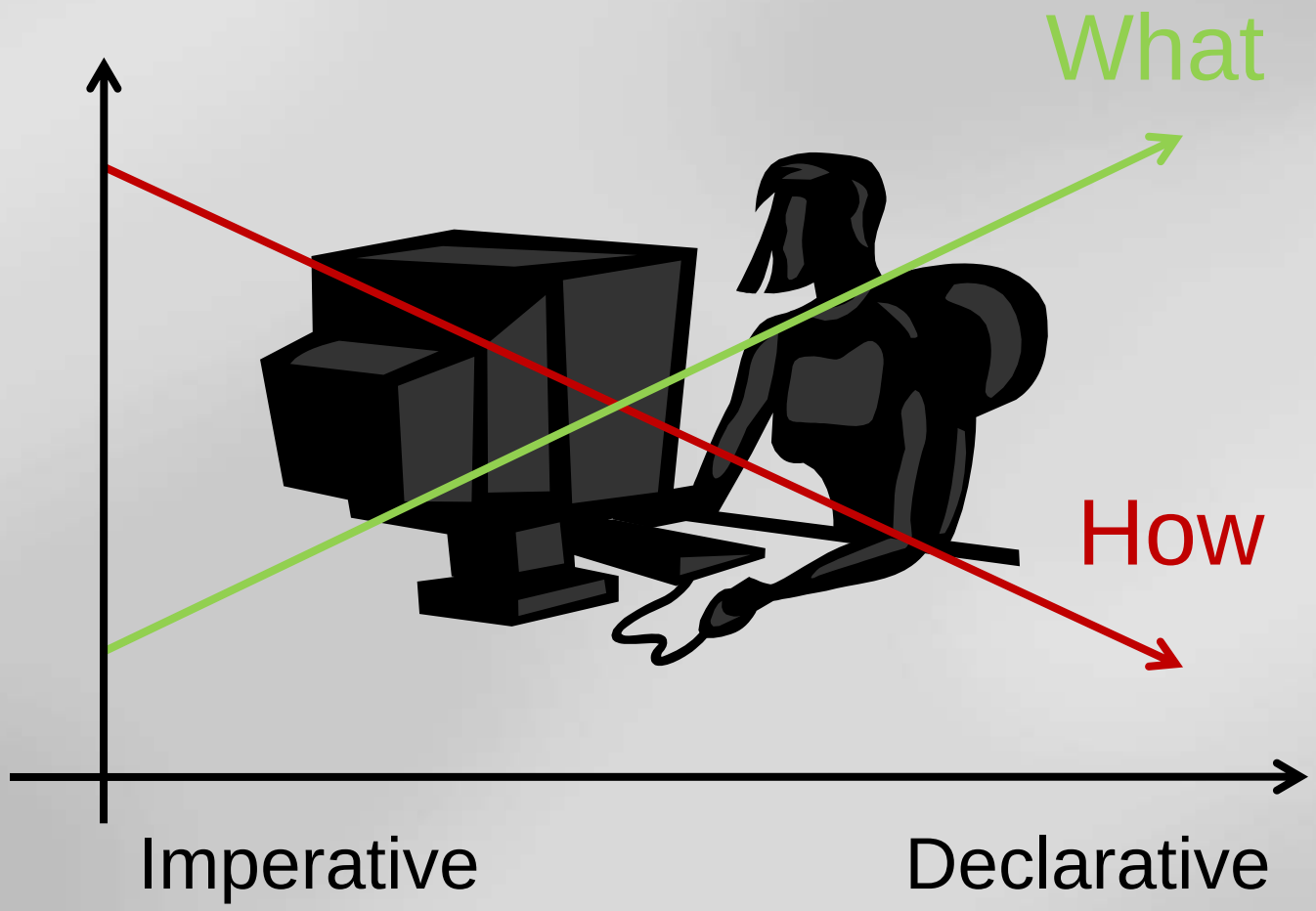
The Evolution of C#



Trends



Declarative Programming



Dynamic vs. Static

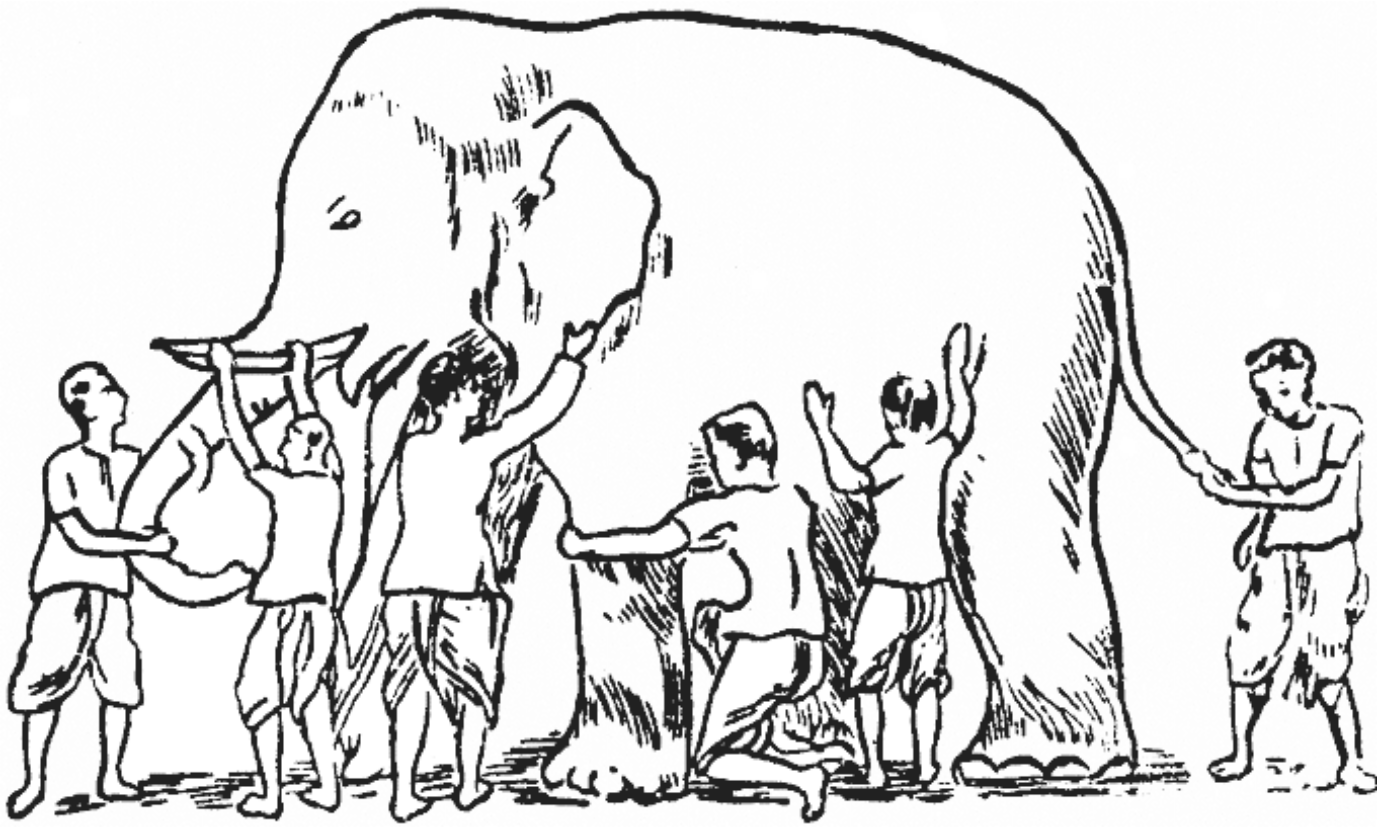
Dynamic Languages

- Simple and succinct
- Implicitly typed
- Meta-programming
- No compilation

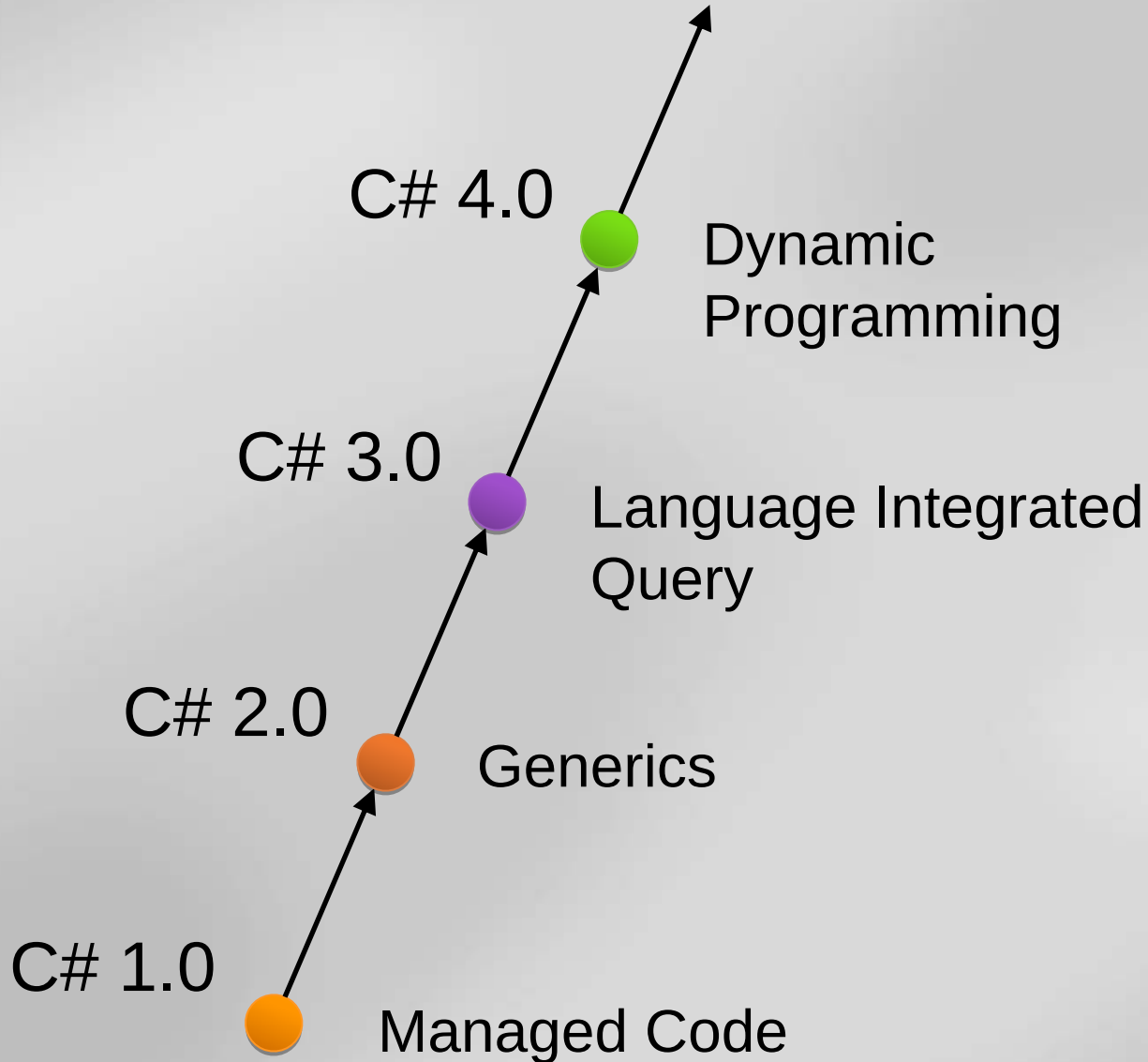
Static Languages

- Robust
- Performant
- Intelligent tools
- Better scaling

Concurrency



The Evolution of C#



C# 4.0 Language Innovations

- Dynamically Typed Objects
- Optional and Named Parameters
- Improved COM Interoperability
- Co- and Contra-variance

.NET Dynamic Programming

IronPython | IronRuby | C# | VB.NET | Others…

Dynamic Language Runtime

Expression Trees | Dynamic Dispatch | Call Site Caching

Object Binder | JavaScript Binder | Python Binder | Ruby Binder | COM Binder

Dynamically Typed Objects

```
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,
int sum = Convert.ToIn
```

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

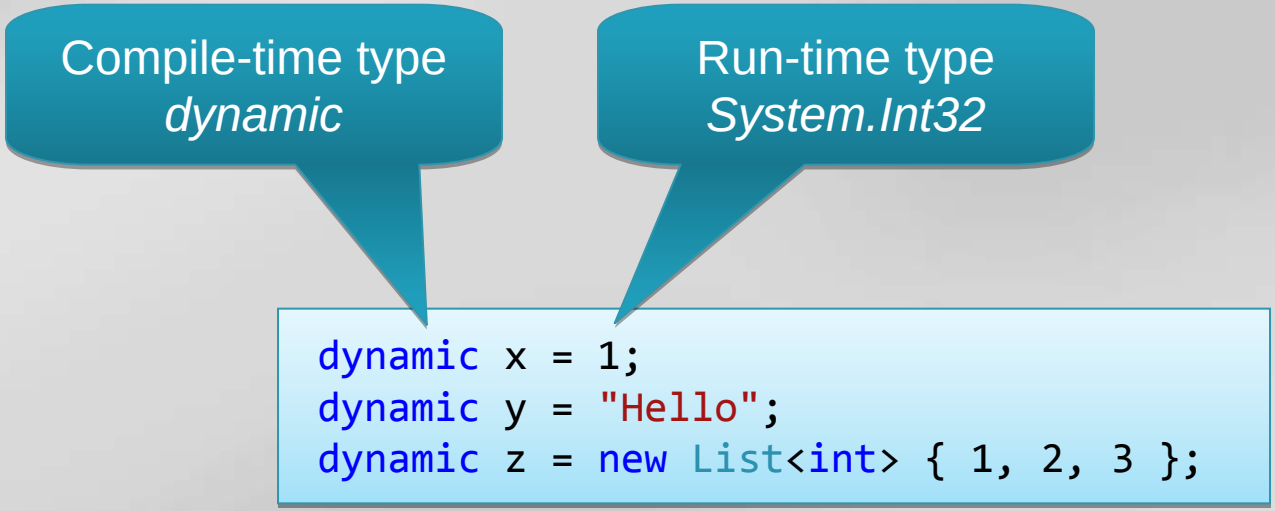
Statically typed to be dynamic

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

Dynamic conversion

Dynamic method invocation

Dynamically Typed Objects



When operand(s) are ***dynamic***…

- Member selection deferred to run-time
- At run-time, actual type(s) substituted for ***dynamic***
- Static result type of operation is ***dynamic***

Dynamically Typed Objects

Microsoft

```
public static class Math
{
    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 long Abs(long value);
    public static sbyte Abs(sbyte value);
    public static short Abs(short value);
```

Method chosen at compile-time: double Abs(double x)

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

Method chosen at run-time: double Abs(double x)

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

Method chosen at run-time: int Abs(int x)

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

IDynamicObject


```
public abstract class DynamicObject : IDynamicObject
{
  public virtual object GetMember(GetMemberBinder info);
  public virtual object SetMember(SetMemberBinder info, object value);
  public virtual object DeleteMember(DeleteMemberBinder info);

  public virtual object UnaryOperation(UnaryOperationBinder info);
  public virtual object BinaryOperation(BinaryOperationBinder info, object arg);
  public virtual object Convert(ConvertBinder info);

  public virtual object Invoke(InvokeBinder info, object[] args);
  public virtual object InvokeMember(InvokeMemberBinder info, object[] args);
  public virtual object CreateInstance(CreateInstanceBinder info, object[] args);

  public virtual object GetIndex(GetIndexBinder info, object[] indices);
  public virtual object SetIndex(SetIndexBinder info, object[] indices, object value);
  public virtual object DeleteIndex(DeleteIndexBinder info, object[] indices);

  public MetaObject IDynamicObject.GetMetaObject();
}
```

Implementing IDynamicObject

demo

Optional and Named Parameters

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

Primary method

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

public StreamReader OpenTextFile(
    string path,
    Encoding encoding);

public StreamReader OpenTextFile(
    string path);
```

Secondary overloads

Call primary with default values

Optional and Named Parameters

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

Optional parameters

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

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

Named argument

Named arguments must be last

Named arguments can appear in any order

Arguments evaluated in order written

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

Non-optional must be specified

Improved COM Interoperability

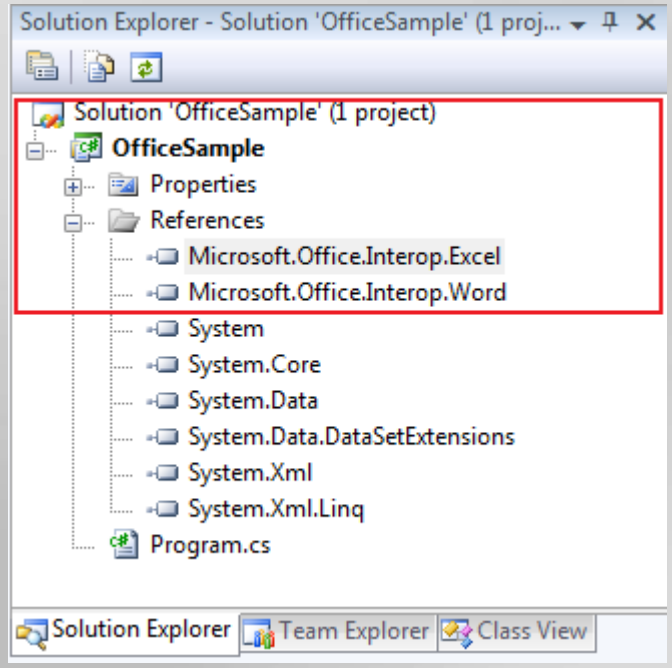
Microsoft

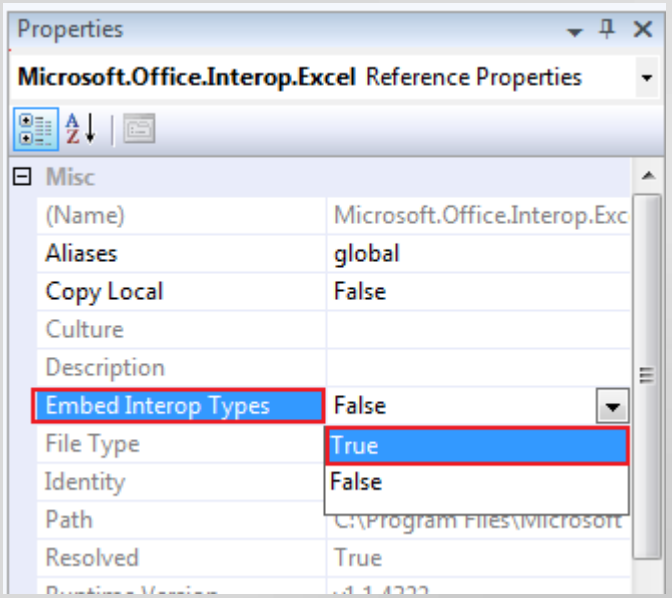
```
object fileName = "Test.docx";
object missing  = System.Reflection.Missing.Value;

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,
    ref missing, ref missing, ref missing);
```

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

Improved COM Interoperability

- Optional and named parameters
- Optional “ref” modifier
- Interop type embedding (“No PIA”)





Improved COM Interoperability

Microsoft

- object → dynamic mapping

```
… you can now say
excel.Cells[1, 1].Value = "Hello";
  instead of
((Excel.Range)excel.Cells[1, 1]).Value2 = "Hello";


… and
Excel.Range range = excel.Cells[1, 1];
  instead of
Excel.Range range = (Excel.Range)excel.Cells[1, 1];
```

Co- and Contra-variance

```
string[] strings = GetStringArray();
Process(strings);
```

.NET arrays are co-variant

```
void Process(object[] objects) {
   objects[0] = "Hello";        // Ok
   objects[1] = new Button();   // Exception!
}
```

…but ***not safely*** co-variant

```
List<string> strings = GetStringList();
Process(strings);
```

Until now, C# generics have been ***invariant***

```
void Process(IEnumerable<object> objects) {
   // IEnumerable<T> is read-only and
   // therefore safely co-variant
}
```

C# 4.0 supports ***safe*** co- and contra-variance

Safe Co- and Contra-variance

```
public interface IEnumerable<out T>
{
    IEnumerator<T> GetEnumerator();
}
```

out = Co-variant
Output positions only

```
public interface IEnumerator<out T>
{
    T Current { get; }
    bool MoveNext();
}
```

Can be treated as less derived

```
IEnumerable<string> strings = GetStrings();
IEnumerable<object> objects = strings;
```

```
public interface IComparer<in T>
{
    int Compare(T x, T y);
}
```

in = Contra-variant
Input positions only

Can be treated as more derived

```
IComparer<object> objComp = GetComparer();
IComparer<string> strComp = objComp;
```

Variance in C# 4.0

- Supported for interface and delegate types
- Value types are always invariant
  - IEnumerable<int> *is not* IEnumerable<object>

Variance in .NET Framework 4.0

Microsoft

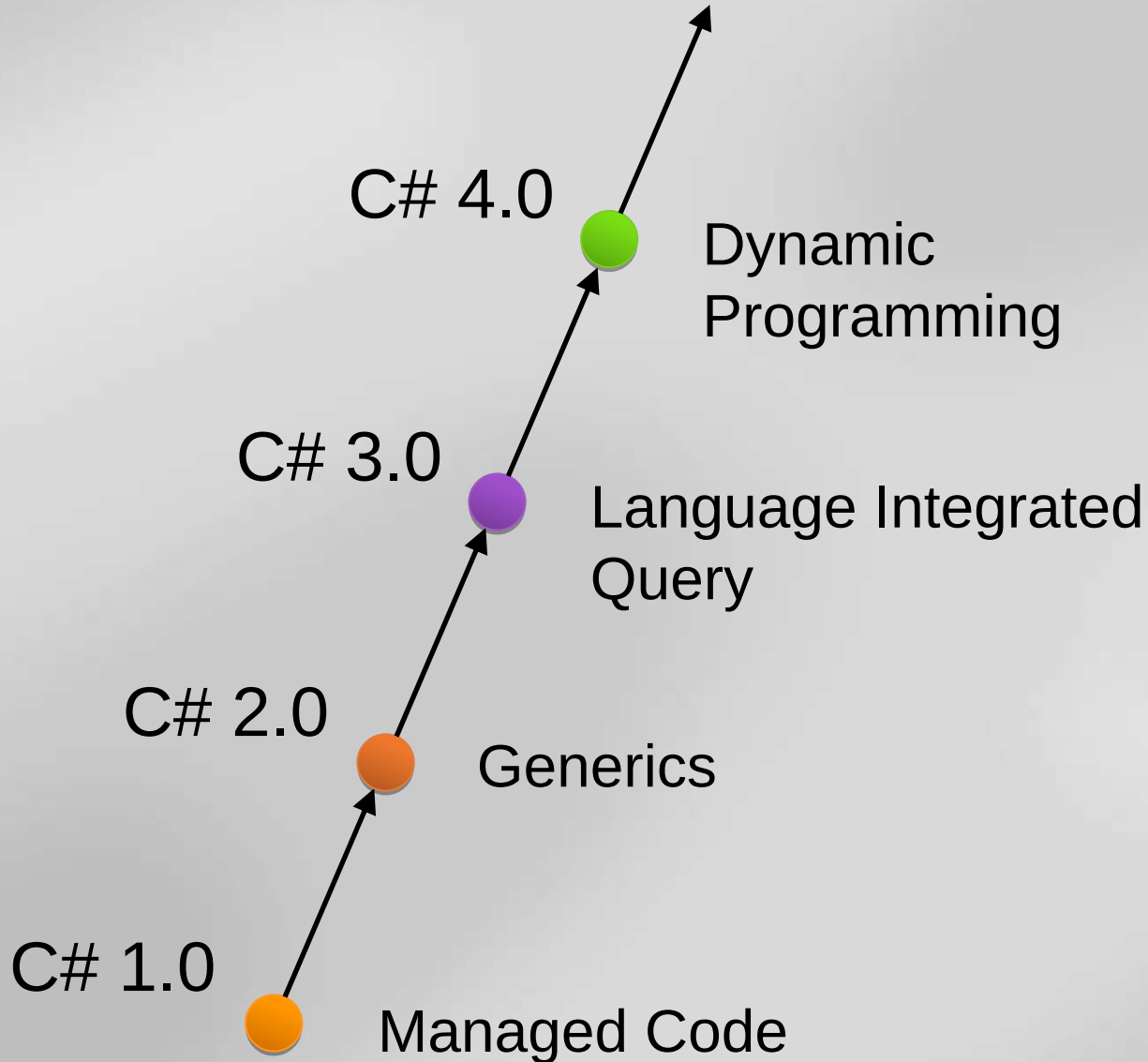
Interfaces

System.Collections.Generic.IEnumerable<out T>
System.Collections.Generic.IEnumerator<out T>
System.Linq.IQueryable<out T>
System.Collections.Generic.IComparer<in T>
System.Collections.Generic.IEqualityComparer<in T>
System.IComparable<in T>

Delegates

System.Func<in T, …, out R>
System.Action<in T, …>
System.Predicate<in T>
System.Comparison<in T>
System.EventHandler<in T>

The Evolution of C#



Additional Resources

- C# 4.0 Samples and Whitepaper
  - http://code.msdn.microsoft.com/csharpfuture
- Visual C# Developer Center
  - http://csharp.net
- C# team member's blogs
  - http://blogs.msdn.com/ericlippert/
  - http://blogs.msdn.com/cburrows/
  - http://blogs.msdn.com/samng/
  - http://blogs.msdn.com/sreekarc/
  - http://blogs.msdn.com/mattwar/
  - http://blogs.msdn.com/ed_maurer/
  - http://blogs.msdn.com/davsterl/
  - http://blogs.msdn.com/alexghi

Microsoft®

Your potential. Our passion.™