C# の現在・過去・未来

えムナウ （児玉宏之）

Microsoft MVP for **Development Tools Visual C#** 2005/01-2008/12

http://mnow.jp/

http://mnow.wankuma.com/

http://blogs.wankuma.com/mnow/

http://www.ailight.jp/blog/mnow/

.NET UX Lab
.Net ユーザーエクスペリエンス研究所

えムナウのC#
プログラミングのページ

わんくま同盟

アジェンダ

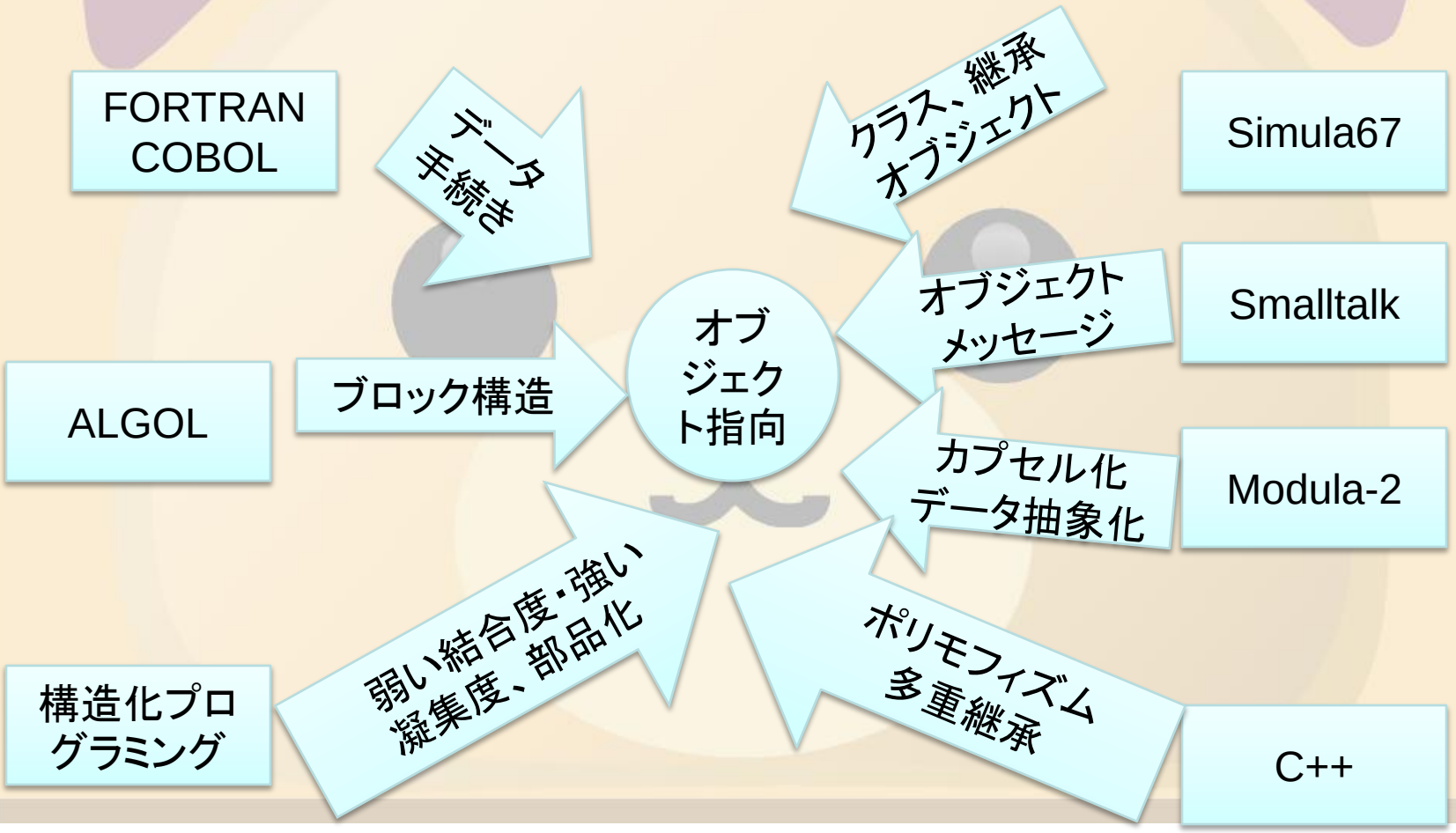
- はじめに
- 捨ててしまったわ、昔のプライドなんて
  - C#の現在過去未来概要
- もしも許されるものなら、きっと生まれ変わる
  - C#の変遷
- ひとつ曲がり角ひとつ間違えて迷い道くねくね
  - 周辺技術の変遷概要
- まとめ

はじめに

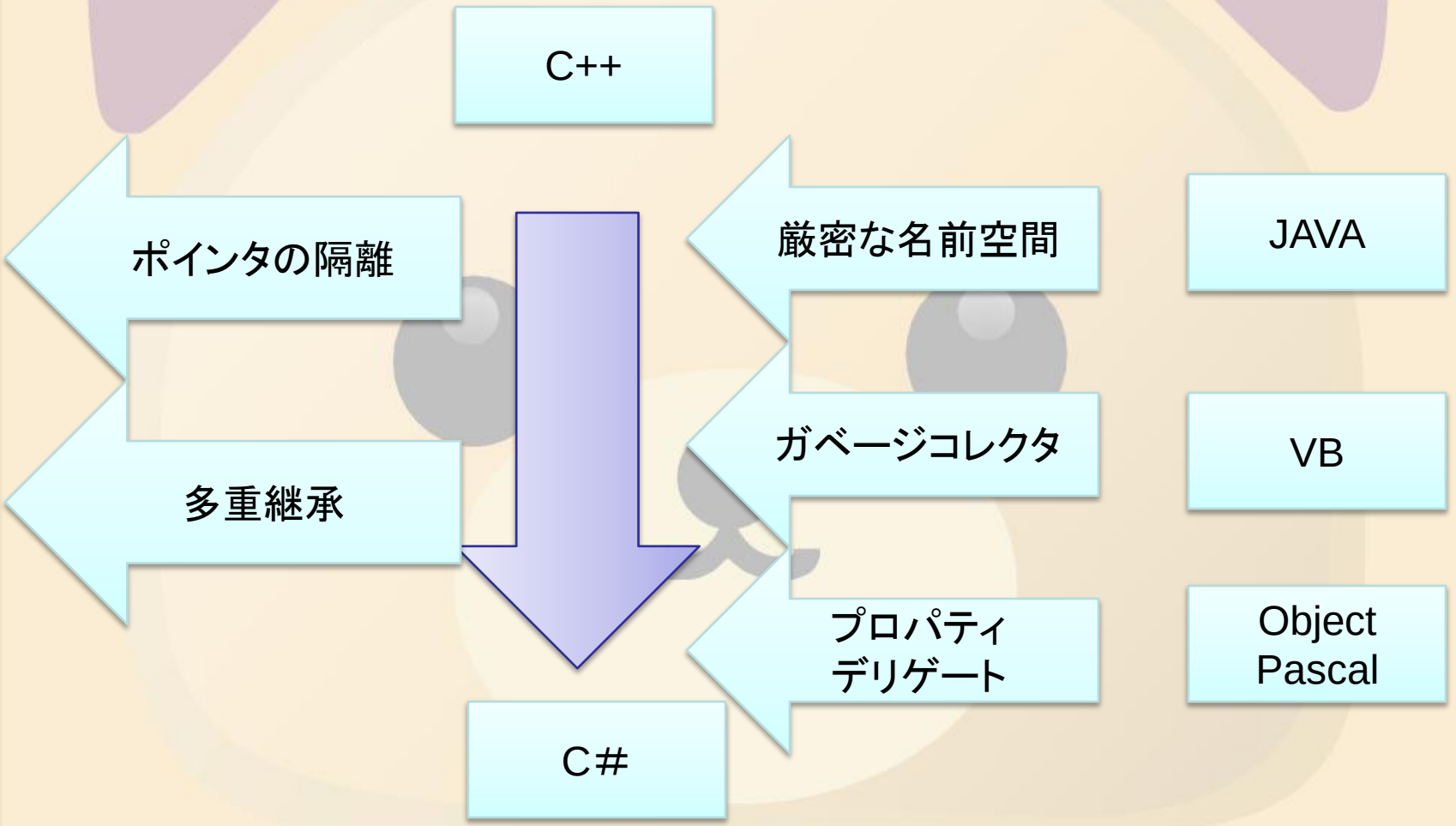
- C#はEcma および ISO によって標準化され、日本においても JIS によって採択されています。
- Delphiの仕事に携わっていた**アンダース・ヘルスバーグ**を中心に開発されたC#ですが、いろいろな要素を吸収しながら現在も成長していってます。
- 今日は、その現在過去未来を追いかけてみましょう。

捨ててしまったわ、昔のプライドなんて

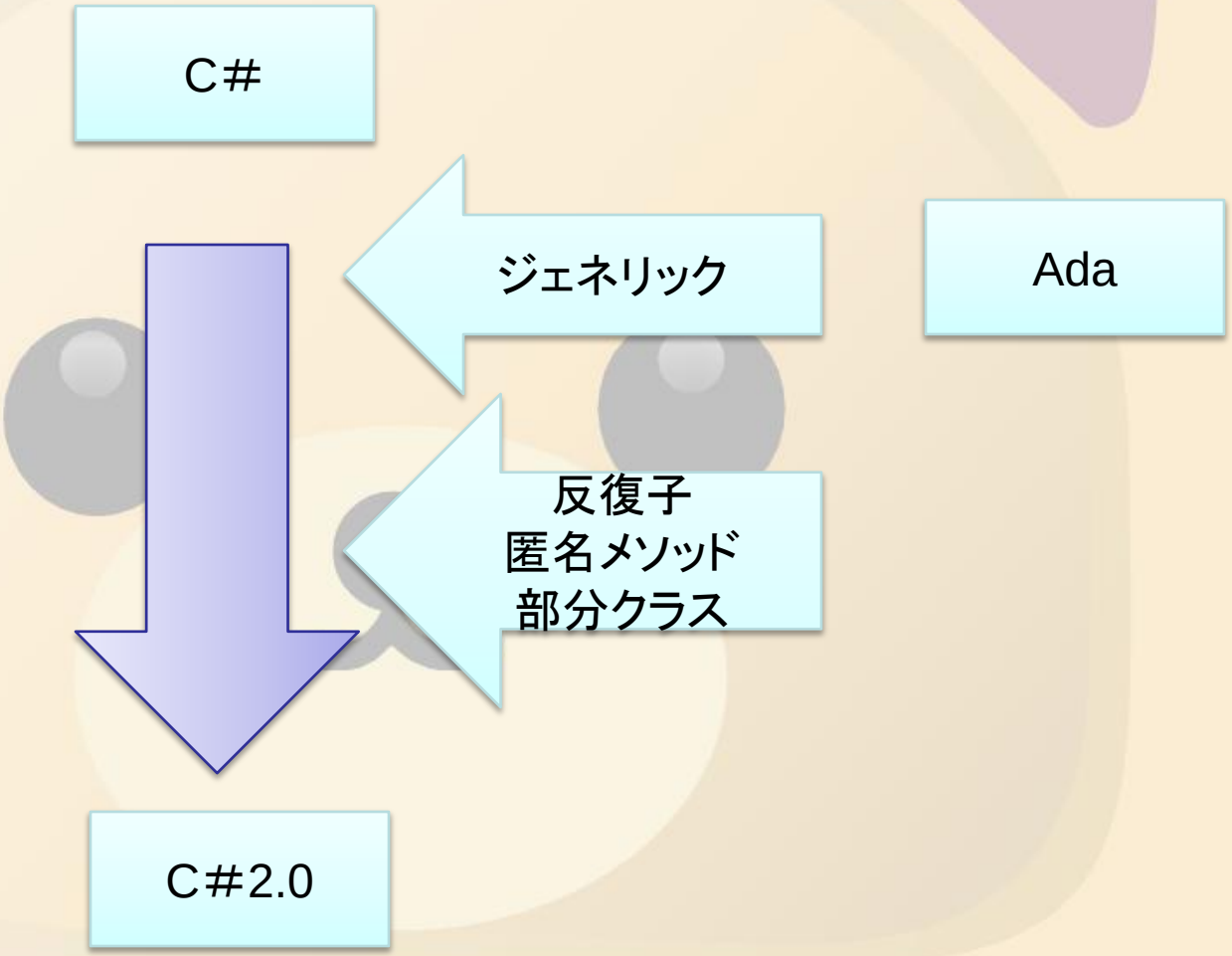
- オブジェクト指向言語 C++ の成り立ち



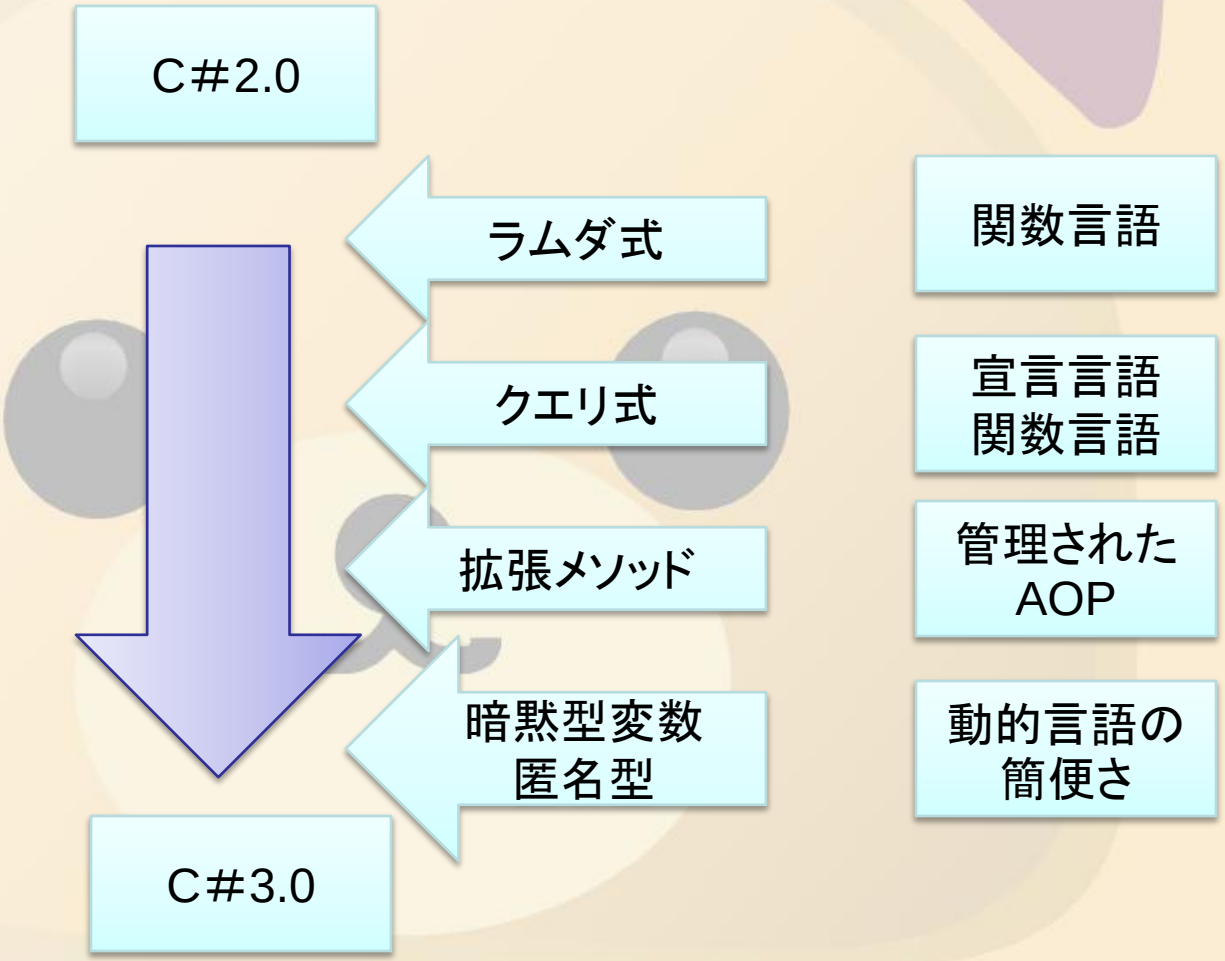
捨ててしまったわ、昔のプライドなんて



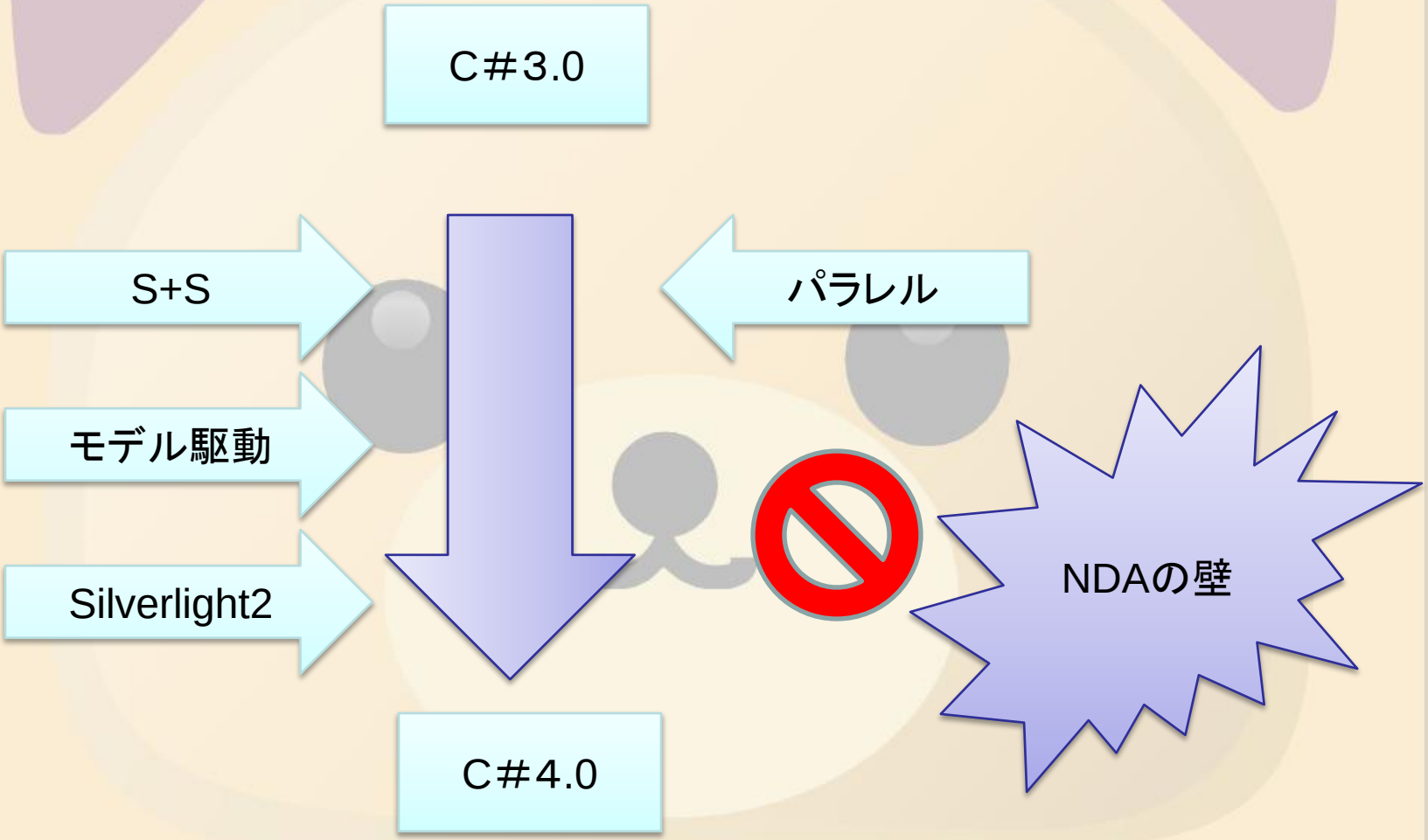
捨ててしまったわ、昔のプライドなんて



捨ててしまったわ、昔のプライドなんて



捨ててしまったわ、昔のプライドなんて



もしも許されるものならきっと生まれ変わる

- C/C++ から C# で何が変わったか
  - global の排除、(シングルトンやスタティックは残されているが・・・)
  - 厳格な名前空間 と 階層構造
  - ポインタの隔離(unsafe の中でだけ利用できる)
  - ガベージコレクタ組み込み と Dispose の追加
  - 多重継承の削除・複数のインタフェースの使用
  - 暗黙の変換を安全なものに限った
  - プロパティの追加(ゲッター・セッターを含めた)

もしも許されるものならきっと生まれ変わる

C/C++

- 厳格な名前空間
- ポインタの隔離
- ガベージコレクタ組み込み
- プロパティ

C#

メモリーリーク
バッファーオーバーラン

もしも許されるものならきっと生まれ変わる

- C#2.0で何が変わったか
  - ジェネリック
  - 反復子 と yield
  - 部分クラス partial
  - null許容型とnull結合演算子
  - 匿名メソッド
  - 名前空間のエイリアス修飾子
  - 静的クラス static class

外部アセンブリのエイリアス extern
プロパティ アクセサのアクセシビリティ
デリゲートの共変性と反変性
固定サイズ バッファ fixed
フレンド アセンブリ
インライン警告制御
　　#pragma warning
volatile の拡張
　　IntPtr・UIntPtr

もしも許されるものならきっと生まれ変わる

- ジェネリック

```
class SimpleList : IList
{
    public SimpleList()
    public int Add(SimpleClass value)
    public void Clear()
    public bool Contains(SimpleClass value)
    public int IndexOf(SimpleClass value)
    public void Insert(int index, SimpleClass value)
    public bool IsFixedSize
    public bool IsReadOnly
    public void Remove(SimpleClass value)
    public void RemoveAt(int index)
    public SimpleClass this[int index]
    public void CopyTo(Array array, int index)
    public int Count
    public bool IsSynchronized
    public object SyncRoot
    public IEnumerator GetEnumerator()
}
```

ぜんぶ実装しろと
大変な苦労が・・・

こんだけ

```
class   SimpleList : List<SimpleClass>
{
}
```

もしも許されるものならきっと生まれ変わる

- 反復子 と yield
  - foreach が便利に

```
int number;
int exponent;
public static System.Collections.IEnumerator GetEnumerator()
{
    int counter = 0;
    int result = 1;
    while (counter++ < exponent)
    {
        result = result * number;
        yield return result;
    }
}
```

もしも許されるものならきっと生まれ変わる

- 部分クラス partial
  - 同じクラスのファイルを分割

Employee_a.cs

```
public partial class Employee
{
    public void DoWork()
    {
    }
}
```

Employee_b.cs

```
public partial class Employee
{
    public void GoToLunch()
    {
    }
}
```

- null許容型とnull結合演算子
  - int? a; でnullを許容する int の宣言
  - int b = a ?? -1; でnullだったら−1を代入

もしも許されるものならきっと生まれ変わる

- 匿名メソッド
  - 簡単なイベントハンドラ関数は不要

```
button1.Click += delegate(object o, EventArgs e)
{ System.Windows.Forms.MessageBox.Show("Click!"); };
```

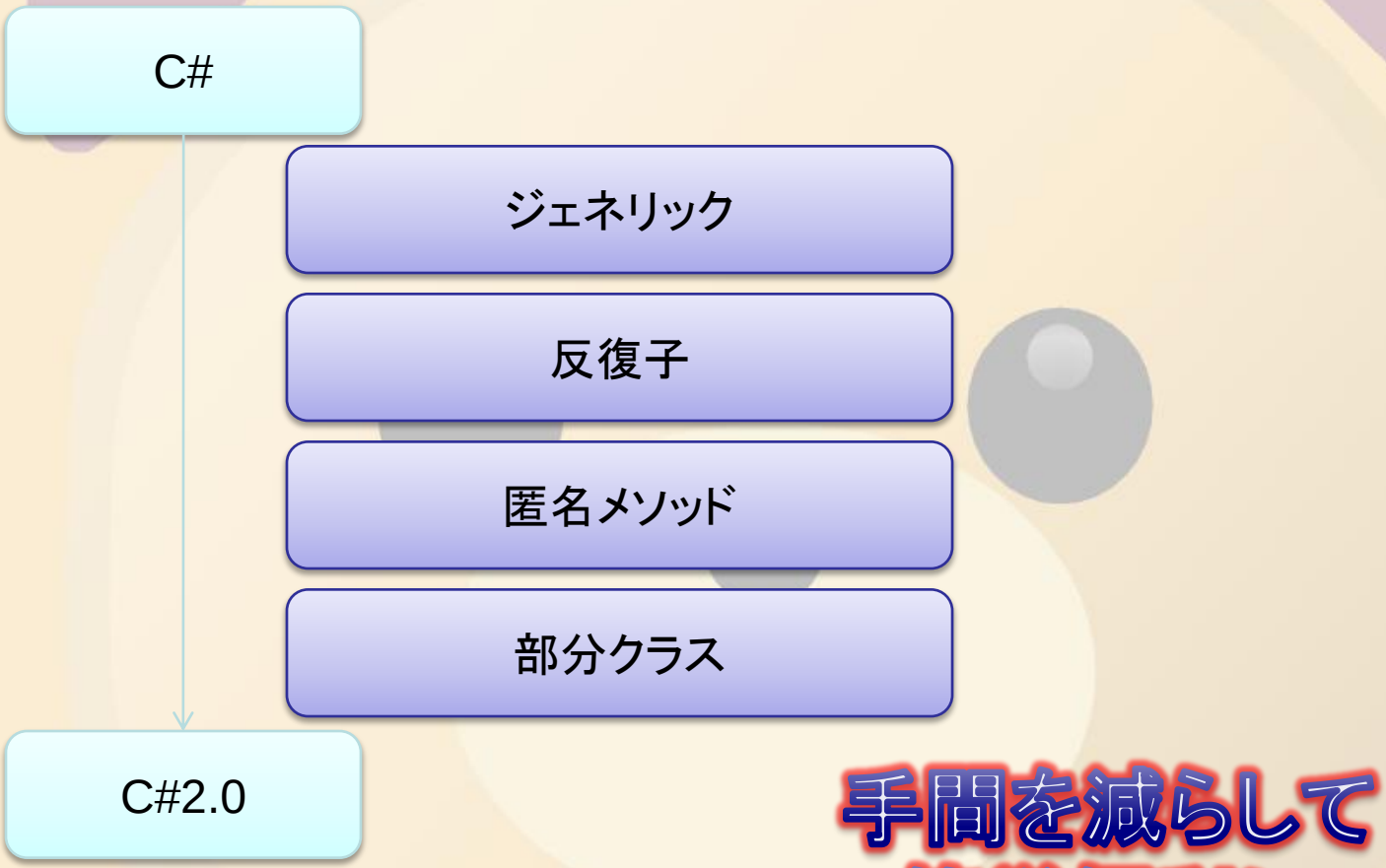
- 名前空間のエイリアス修飾子
  - 同じ名前の名前空間の解決

```
global::System.Console.WriteLine(number);
```

- 静的クラス static class
  - インスタンスを作らないで利用可能

```
static class CompanyInfo {
        public static string GetCompanyName() { return "CompanyName"; }
}
```

もしも許されるものならきっと生まれ変わる



手間を減らして
簡単便利に

もしも許されるものならきっと生まれ変わる

- C#3.0で何が変わったか
  - 暗黙に型付けされたローカル変数
  - オブジェクト初期化子
  - コレクション初期化子
  - 拡張メソッド
  - 匿名型
  - ラムダ式
  - クエリ式
  - 自動実装プロパティ
  - 部分メソッド定義 partial メソッド

もしも許されるものならきっと生まれ変わる

- 暗黙に型付けされたローカル変数

```
var al = new List<MyAccount>();
al.Add(new MyAccount("hnaka", "553-0001", "大阪府"));
al.Add(new MyAccount("hkodama", "168-0064", "東京都"));

var accounts =
    EnumerableExtensions<MyAccount, MyAccount2>.Select(
        EnumerableExtensions<MyAccount, MyAccount2>.Where
            (al, delegate(MyAccount a) { return a.ZipCode == "168-0064"; }),
        delegate(MyAccount a) { return new MyAccount2(a.Name, a.ZipCode); }
    );

Console.WriteLine("C#3.0 暗黙に型付けされたローカル変数");
foreach (var account in accounts)
{
    Console.WriteLine(account.Name + "(" + account.ZipCode + ")");
}
```

もしも許されるものならきっと生まれ変わる

- 拡張メソッド

```
static class EnumerableExtension
{
    public delegate TD SelectFunc<TS, TD>(TS t);
    public static IEnumerable<TD> Select<TS, TD>
                (this IEnumerable<TS> e, SelectFunc<TS, TD> f)
    {
        foreach (TS i in e) {
            yield return f(i);
        }
    }
    public static IEnumerable<TS> Where<TS>
                (this IEnumerable<TS> e, Predicate<TS> p)
    {
        foreach (TS i in e)  {
            if (p(i)) yield return i;
        }
    }
}
```

もしも許されるものならきっと生まれ変わる

- ラムダ式

```
var al = new List<MyAccount>();
al.Add(new MyAccount("hnaka", "553-0001", "大阪府"));
al.Add(new MyAccount("hkodama", "168-0064", "東京都"));

var accounts = al
    .Where(a => a.ZipCode == "168-0064")
    .Select(a => new MyAccount2(a.Name, a.ZipCode));

Console.WriteLine("C#3.0 ラムダ式");
foreach (var account in accounts)
{
    Console.WriteLine(account.Name + "(" + account.ZipCode + ")");
}
```

もしも許されるものならきっと生まれ変わる

- オブジェクト初期化子および コレクション初期化子

```
var al = new List<MyAccount> {
    new MyAccount{Name="hnaka",ZipCode="553-0001",Prefecture="大阪府"},
    new MyAccount{Name="hkodama",ZipCode="168-0064",Prefecture="東京都"}
};

var accounts = al
    .Where(a => a.ZipCode == "168-0064")
    .Select(a => new MyAccount2 { Name = a.Name, ZipCode = a.ZipCode });

Console.WriteLine("C#3.0 オブジェクト初期化子および コレクション初期化子");
foreach (var account in accounts)
{
    Console.WriteLine(account.Name + "(" + account.ZipCode + ")");
}
```

もしも許されるものならきっと生まれ変わる

- 自動実装プロパティ

```
class LightweightCustomer
{
    public double TotalPurchases { get; set; }
    public string Name { get; private set; } // read-only
    public int CustomerID { get; private set; } // read-only
}
```

- 部分メソッド定義 partial メソッド

```
Employee_a.cs
public partial class Employee
{
    // 宣言だけ
    partial void onNameChanged();
}
```

```
Employee_b.cs
public partial class Employee
{
    partial void onNameChanged()
    {
        // 実装コード
    }
}
```

もしも許されるものならきっと生まれ変わる

- 匿名型

```
var al = new [] {
    new {Name="hnaka",ZipCode="553-0001",Prefecture="大阪府"},
    new {Name="hkodama",ZipCode="168-0064",Prefecture="東京都"}
};

var accounts = al
    .Where(a => a.ZipCode == "168-0064")
    .Select(a => new { Name = a.Name, ZipCode = a.ZipCode });

Console.WriteLine("C#3.0 匿名型");
foreach (var account in accounts)
{
    Console.WriteLine(account.Name + "(" + account.ZipCode + ")");
}
```

もしも許されるものならきっと生まれ変わる

- クエリ式

```
var al = new[] {
    new {Name="hnaka",ZipCode="553-0001",Prefecture="大阪府"},
    new {Name="hkodama",ZipCode="168-0064",Prefecture="東京都"}
};
var accounts = from a in al
               where a.ZipCode == "168-0064"
               select new { Name = a.Name, ZipCode = a.ZipCode };
Console.WriteLine("C#3.0 クエリ式");
foreach (var account in accounts)
{
    Console.WriteLine(account.Name + "(" + account.ZipCode + ")");
}
```

もしも許されるものならきっと生まれ変わる

C#2.0

- 暗黙に型付けされたローカル変数
- ラムダ式
- オブジェクト初期化子
  コレクション初期化子
- クエリ式
- 拡張メソッド
- 自動実装プロパティ
- 匿名型
- partial メソッド

C#3.0

すべてはLinqのために

もしも許されるものならきっと生まれ変わる

- パラレルLinq・パラレルFor

```
int[] data  = new int[] { 0, 1, 2, 3 };
int[] data2 = (from x in data.AsParallel() select x * x).ToArray();
```

```
int[] data  = new int[] { 0, 1, 2, 3 };
Int[] data2 = new int[] { 0, 0, 0, 0 };
Parallel.For(0, 3, delegate(int i) {data2[i] = data[i] * data[i]; });
```

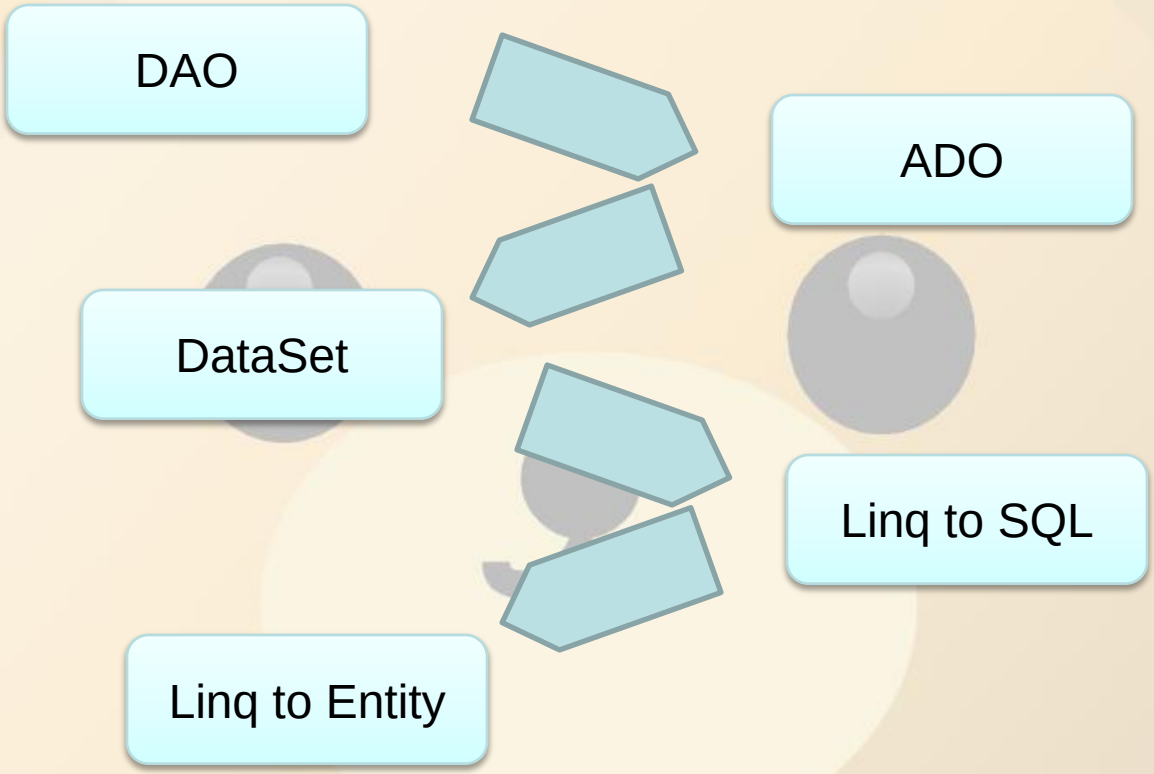
ここでいきなりクイズです。
結果として得られる data2 は同じものですか？

- データアクセス

バージョン	便利なアクセス手段	データ表現
Microsoft Visual Studio 2002 .NET Framework 1.0	DataAdapter	DataSet DataTable
Microsoft Visual Studio 2005 .NET Framework 2.0	TableAdapter	DataSet DataTable
Microsoft Visual Studio 2008 .NET Framework 3.5	Linq to SQL DataContext メソッド	DataContext object
Microsoft Visual Studio 2008 .NET Framework 3.5　SP1	Linq to Entity ObjectContext メソッド	ObjectContext EntityObject

ひとつ曲がり角ひとつ間違えて迷い道くねくね

- データアクセス



ひとつ曲がり角ひとつ間違えて迷い道くねくね

- Windowsアプリケーション

バージョン	テクノロジ	表の表現
Microsoft Visual Studio 2002 .NET Framework 1.0	Windows Forms	DataGrid
Microsoft Visual Studio 2005 .NET Framework 2.0	Windows Forms	DataGridView
Microsoft Visual Studio 2005 .NET Framework 3.0	WPF	ListView
Microsoft Visual Studio 2008 .NET Framework 3.5　SP1？	WPF	DataGrid

ひとつ曲がり角ひとつ間違えて迷い道くねくね

- Webアプリケーション

バージョン	テクノロジ
Microsoft Visual Studio 2002 .NET Framework 1.0	ASP.NET
Microsoft Visual Studio 2002 .NET Framework 1.0	SOAP
Microsoft Visual Studio 2005 .NET Framework 2.0	Ajax
	Silverlight1.0
Microsoft Visual Studio 2008 .NET Framework 3.5　SP1？	Silverlight2.0
	ソフトウェア+サービス

ひとつ曲がり角ひとつ間違えて迷い道くねくね

- クライアントOS

西暦		
1981年	MS-DOS	
1986年	Windows	1.0
1993年		3.1
1995年		95
1998年		98
2000年		Me
2001年		XP
2007年		Vista
2009年？		Windows7（仮称）
????年	Midori（コードネーム）	

まとめ

- C#の現在過去未来を駆け足で見てきましたが、いろいろな技術を貪欲に取り入れながら、進化し続けていっています。
- 同様にOSやフレームワークの技術も進化し続けていっています。
- 世の中、まだ、VB6やMFCやASPを使った仕事が多いとも聞きますが、新しい技術も勉強していかなければならないと思っています。