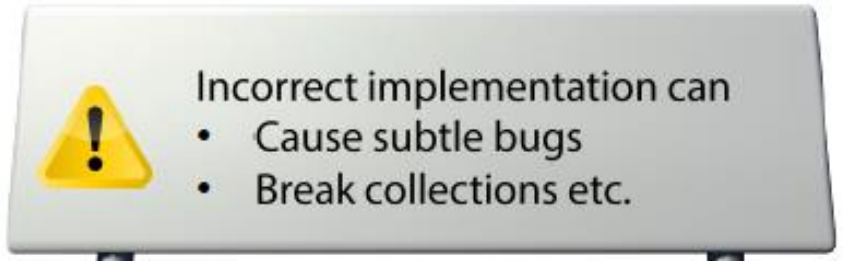
Equality and Comparisons

Why do we need to this

Why is Equality so Hard?

.NET Provides...

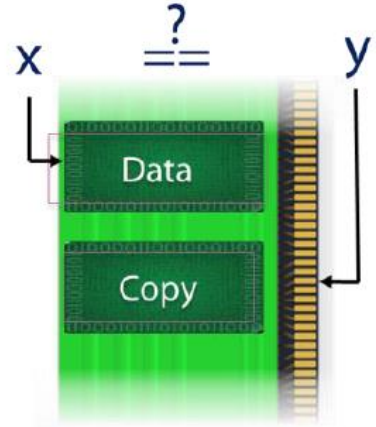
```
System.Object
static Equals()
virtual Equals()
static ReferenceEquals()
virtual GetHashCode()
```



- IEquatable<T>
- IComparable
- IComparable<T>
- IComparer
- IComparer<T>
- IEqualityComparer
- IEqualityComparer<T>
- IStructuralEquatable
- IStructuralComparable

Why is Equality Hard?

1. Reference/Value Equality



(C# syntax doesn't distinguish ref/value)

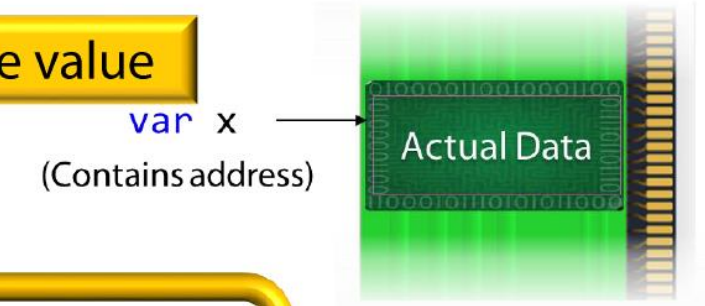
2. Multiple ways to compare values

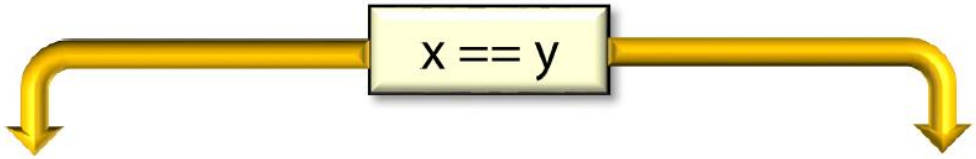
3. Accuracy (for floating points)

4. Conflict with OOP

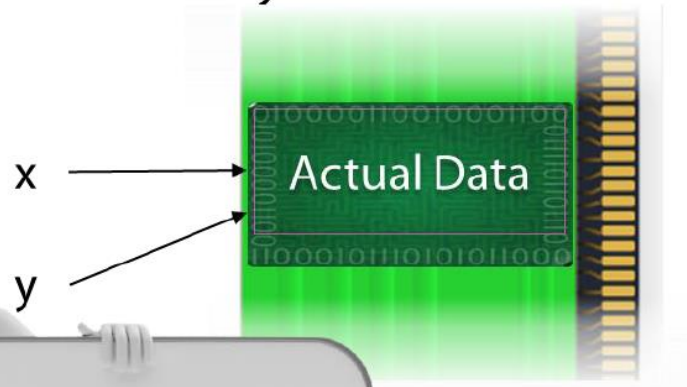
Reference vs Value Equality

Reference types contain a pointer to the value

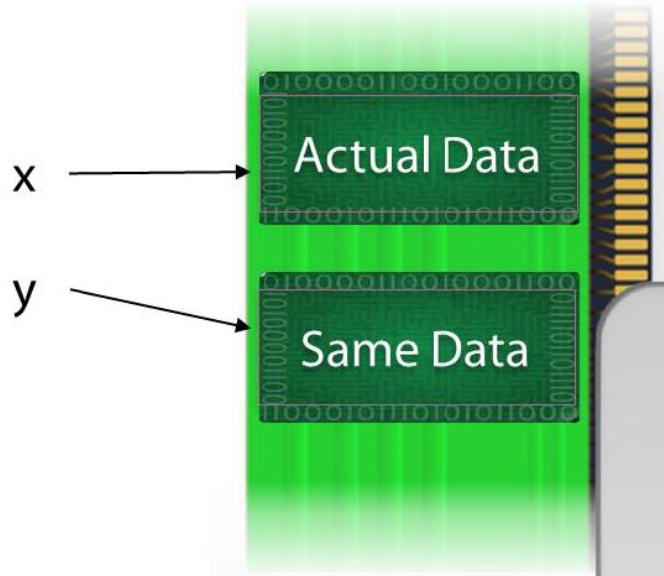




Same memory location?



Same value?



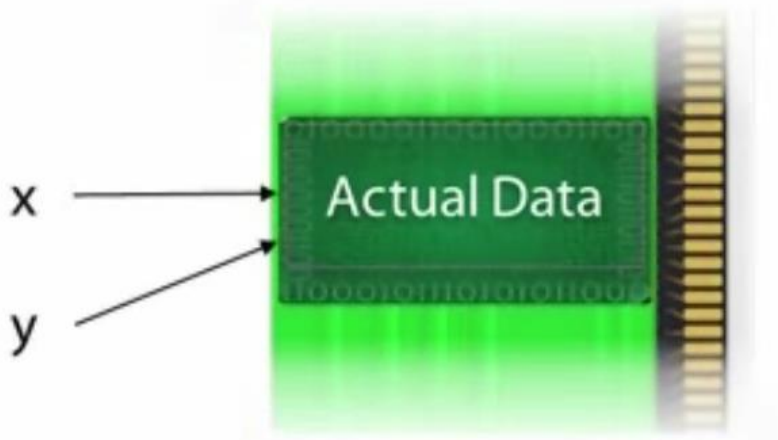
Code Demo

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        Button button1 = new Button();
        button1.Text = "Click me now!";

        Button button2 = new Button();
        button2.Text = "Click me now!";

        Console.WriteLine(button1 == button2);
    }
}
```


False
Press any key to continue . . .
== evaluates **reference equality** for
System.Windows.Forms.Button

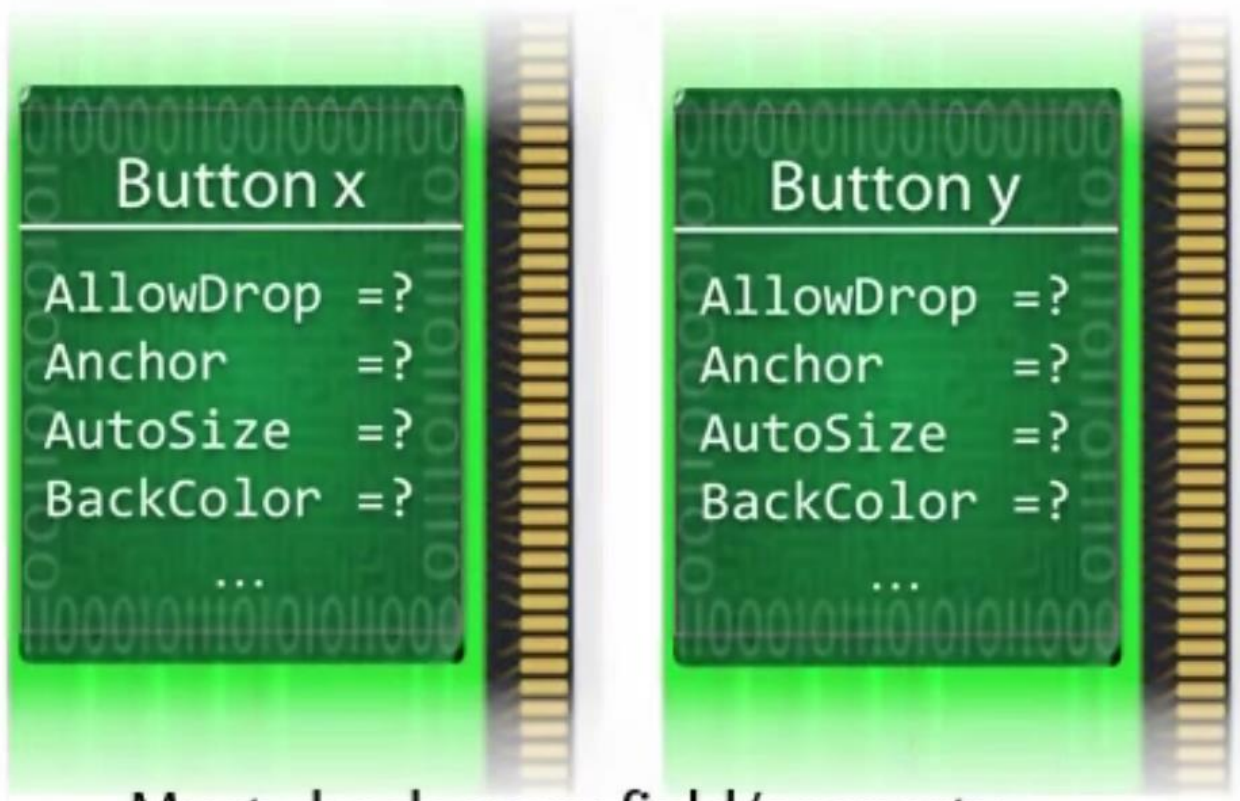
Performance

Reference Equality: Quick



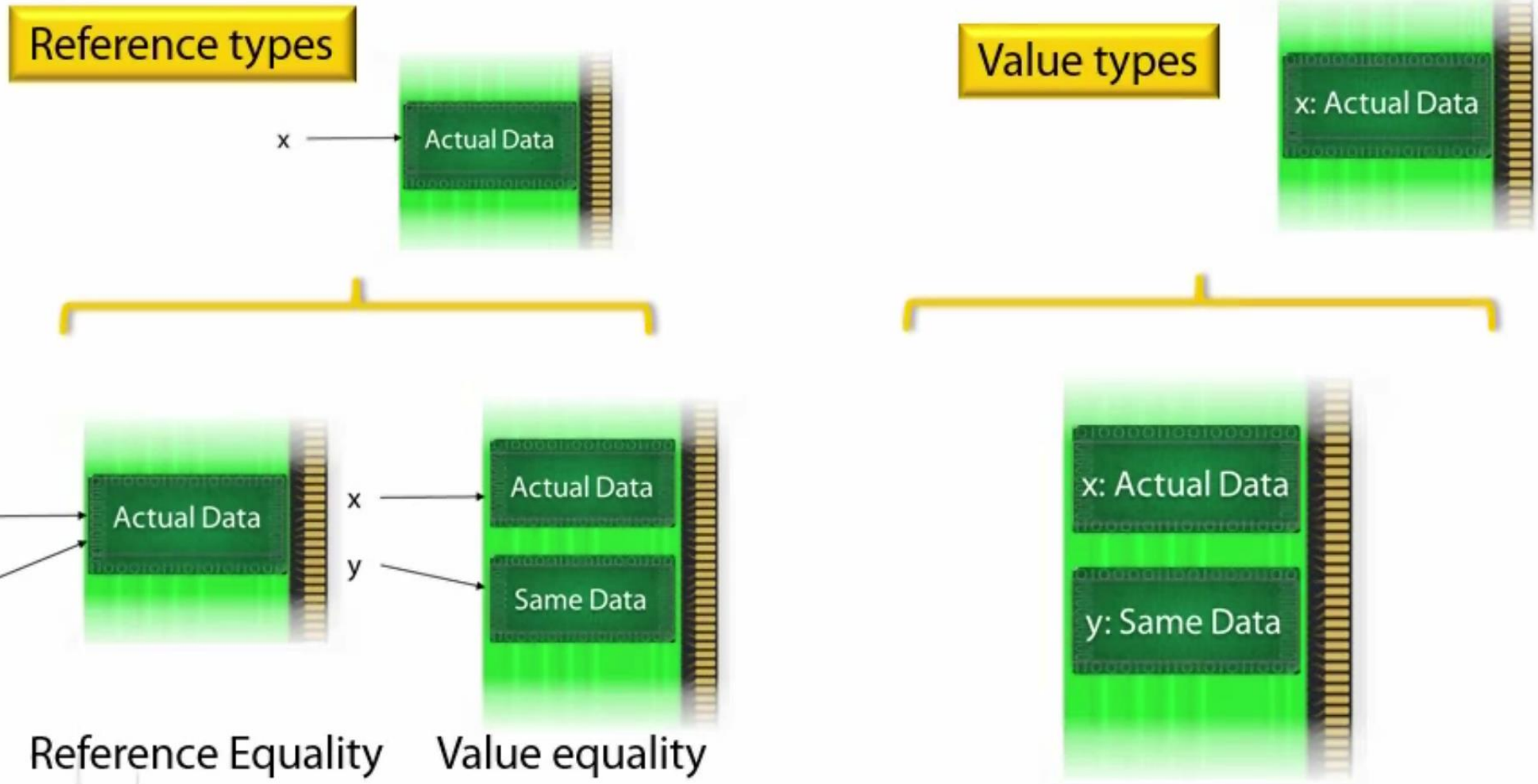
Do x and y contain
the same address?

Value Equality: Slow



Must check every field/property

Reference vs Value Types



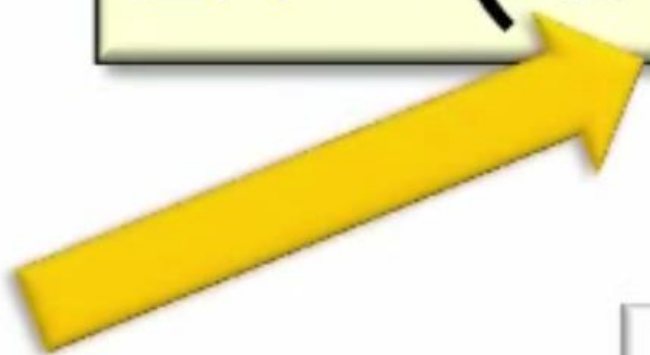
```
class Program
{
    0 references
    static void Main(string[] args)
    {
        int three = 3;
        int threeAgain = 3;
        bool intCmp = (three == threeAgain);
        Console.WriteLine(string.Format("compare ints:        {0}", intCmp));
    }
}
```

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        int three = 3;
        int threeAgain = 3;
        bool intCmp = (three == threeAgain);
        Console.WriteLine(string.Format("compare ints:       {0}", intCmp));

        bool objCmp = ((object)three == (object)threeAgain);
        Console.WriteLine(string.Format("compare objects     {0}", objCmp));
    }
```

```
compare ints:       True
compare objects     False
Press any key to continue . . .
```

C# Has Only One == Operator

```
if (a == b) {
```



What will this do?

You just have to know
what == does for each type

Multiple Ways to Compare Values

"Hello" == "hello" ?

```
string str1 = "apple pie";
string str2 = "apple pie";
if (str1 == str2) {
```

Should we say these are equal?

Clearly, Yes! ✔

```
string str1 = "apple pie";
string str2 = "Apple Pie";
if (str1 == str2) {
```

Should we say these are equal?

C# says they are not:
`(str1 == str2)` evaluates to `false`

But more generally:
It depends on the context

Should You Ignore Case?

Get recipe for Apple Pie

Case doesn't matter

We want
"Apple Pie" = "apple pie"

Username FoodLover2014

Password Apple Pie

Case does matter

We want
"Apple Pie" != "apple pie"

Example: Database Records

Are these equal…?

ID	Name	Price	Last Modified
4382	apple pie	$3.50	1 Dec 2013

ID	Name	Price	Last Modified
4382	apple pie	$3.75	15 May 2014

Accuracy of Data

Some data types are inherently approximate:

`float` `double` `decimal`

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        float six = 6.0000000f;
        float nearlySix = 6.0000001f;

        Console.WriteLine(six == nearlySix);
    }
}
```

```
True
Press any key to continue . . .
```

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        float x = 5.05f;
        float y = 0.95f;

        Console.WriteLine(x + y);
        Console.WriteLine("x + y ==6? " + (x + y == 6.0f));

    }
}
```

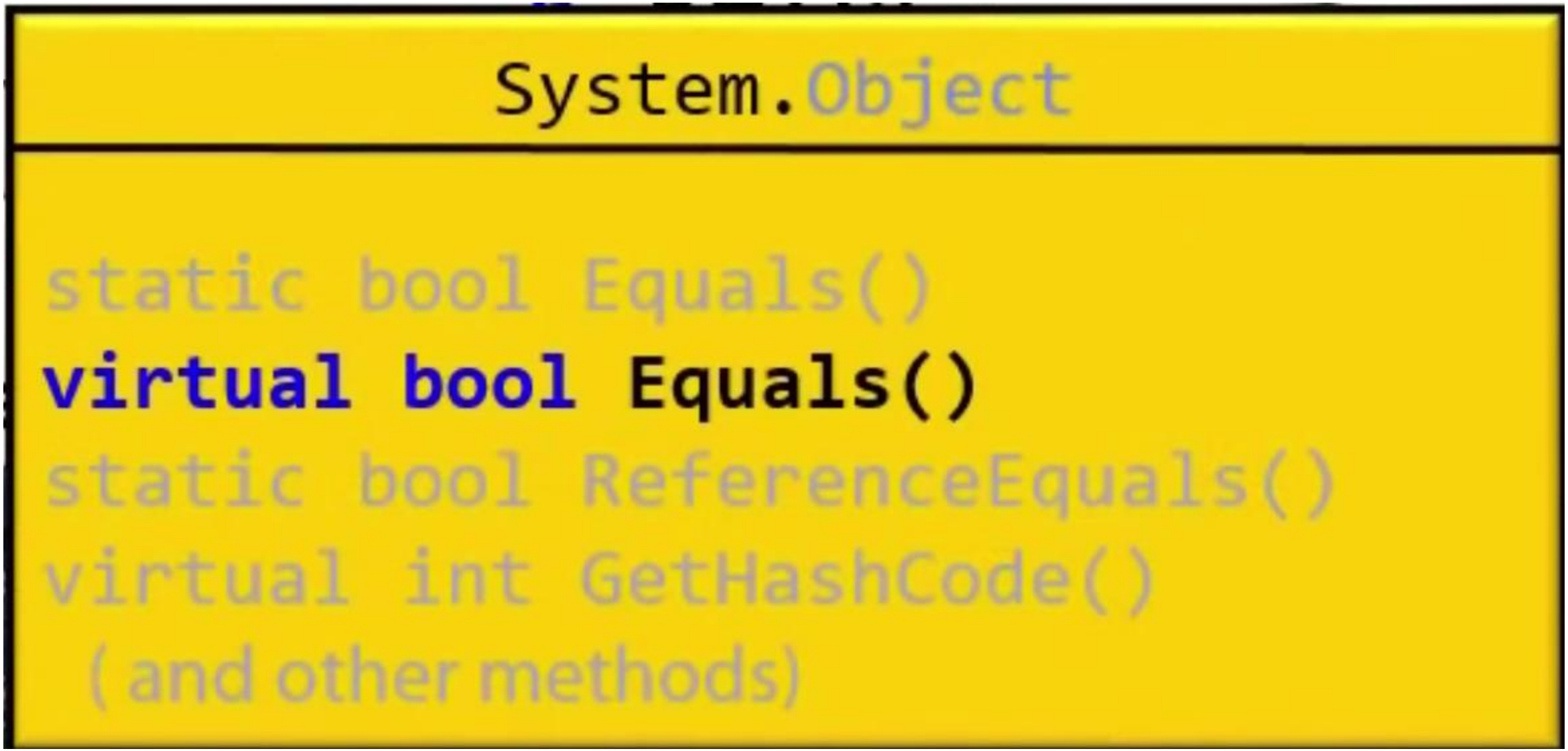
```
6
x + y ==6? False
Press any key to continue . .
```

Rounding errors in floating point variables can cause == to give the 'wrong' answer

The Equality / Type Safety / OOP Conflict

Equality in .NET

- The virtual `object.Equals()` method.
- The static `object.Equals()` method.
- The static `object.ReferenceEquals()` method.


System.Object
static bool Equals()
virtual bool Equals()
static bool ReferenceEquals()
virtual int GetHashCode()
(and other methods)

```
public class Food
{
    private string _name;
    0 references
    public string Name { get { return _name; } }

    2 references
    public Food(string name)
    {
        this._name = name;
    }

    0 references
    public override string ToString()
    {
        return _name;
    }

}
```

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        Food banana = new Food("banana");
        Food chocolate = new Food("chocolate");

        Console.WriteLine(banana.Equals(chocolate));
    }
}
```

```
False
Press any key to continue . .
```

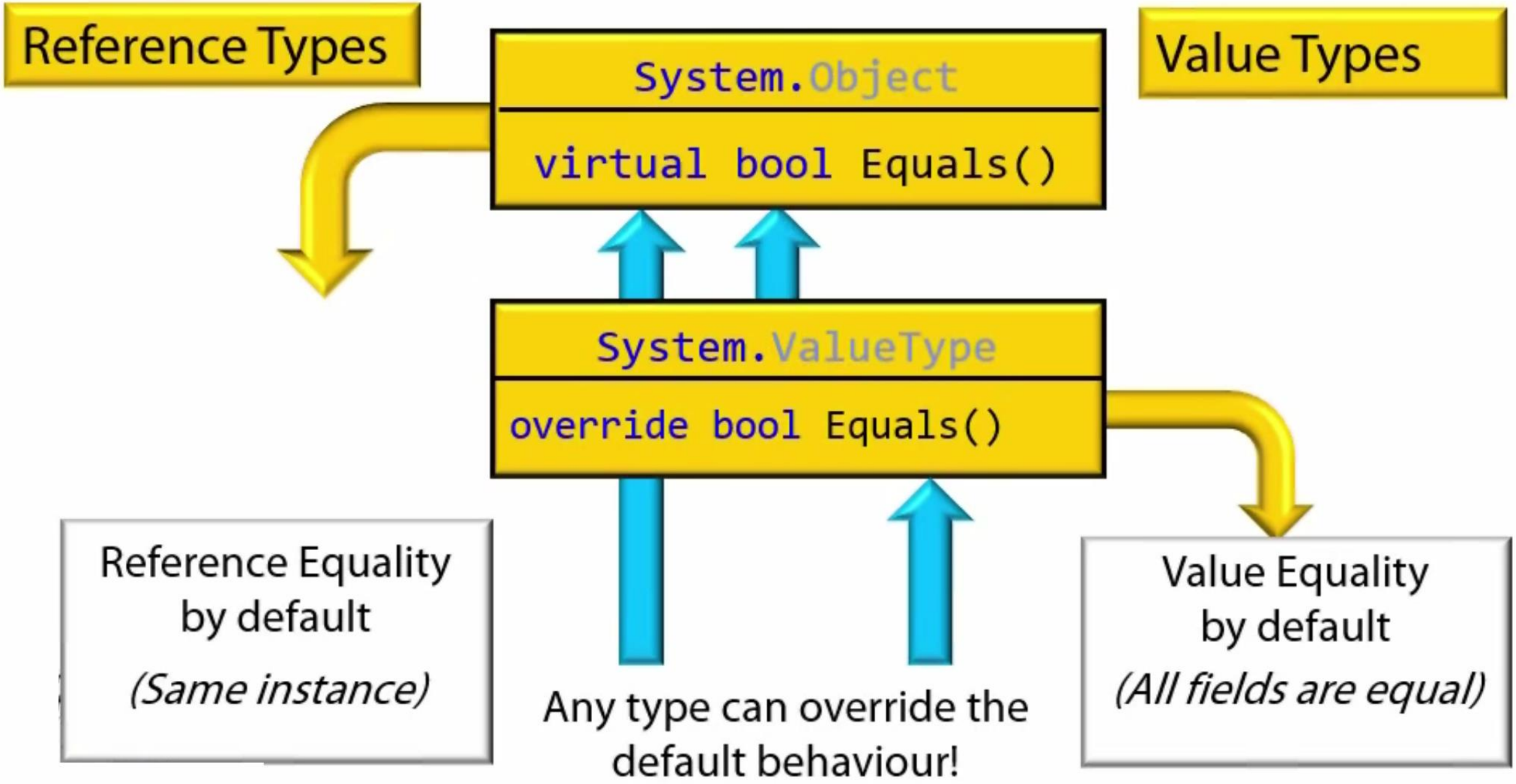
Equals is implemented on `object`
- so available on all types

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        Food banana = new Food("banana");
        Food banana2 = new Food("banana");
        Food chocolate = new Food("chocolate");
        Console.WriteLine(banana.Equals(chocolate));
        Console.WriteLine(banana.Equals(banana2));
    }
```

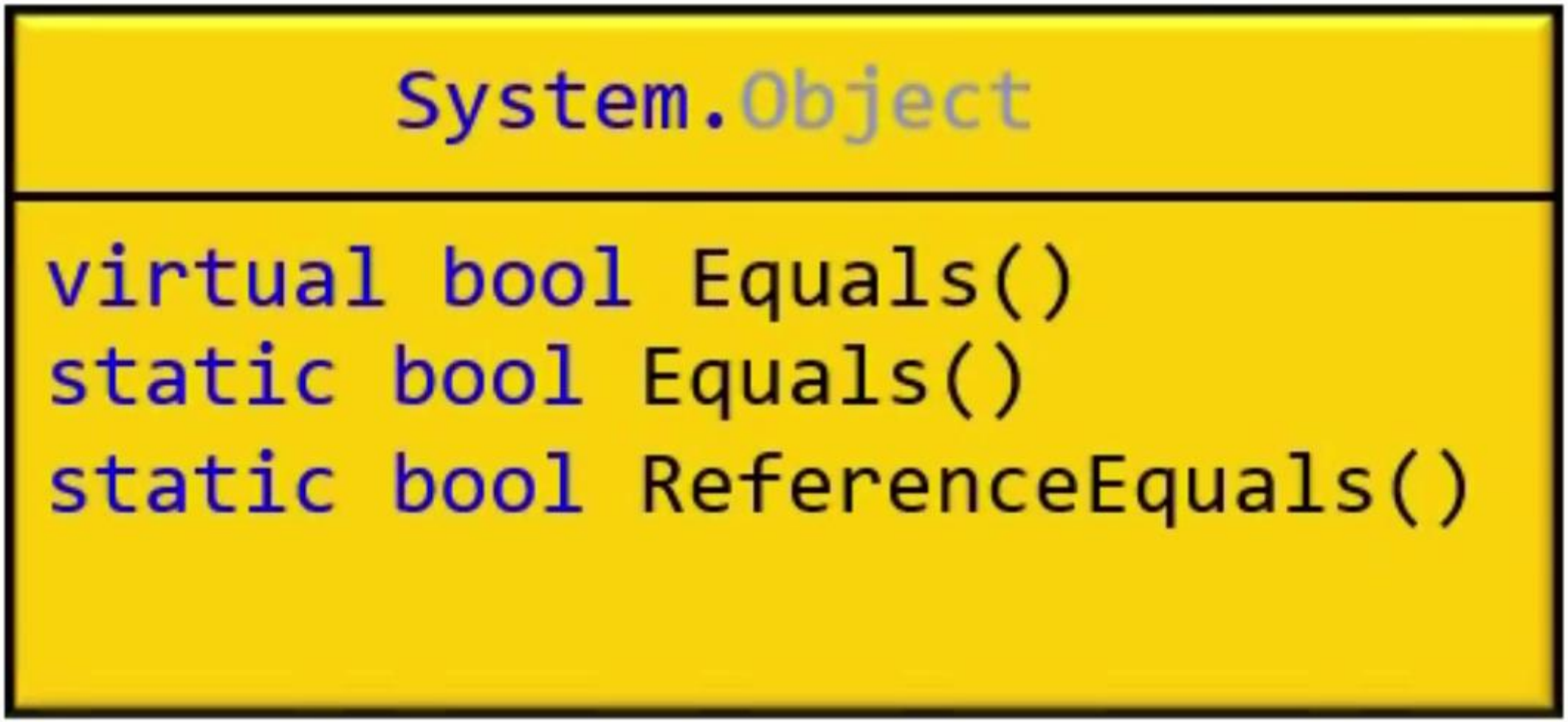
```
False
False
Press any key to continue
```

`object.Equals()` evaluates **reference** equality (unless overridden)

Guiding Principles



Other Equality Methods



```
class Program
{
    0 references
    static void Main(string[] args)
    {
        Food banana = new Food("banana");
        Food banana2 = new Food("banana");
        Food chocolate = new Food("chocolate");

        Console.WriteLine(banana.Equals(chocolate));
        Console.WriteLine(banana.Equals(banana2));
    }


}
```

```
static void Main(string[] args)
{
    Food banana = new Food("banana");
    Food banana2 = new Food("banana");
    Food chocolate = new Food("chocolate");

    Console.WriteLine(banana.Equals(null));
}
```

```
False
Press any key to continue .
```

In .NET:
`null` NEVER equals
any non-null reference

```
static void Main(string[] args)
{
    Food banana = new Food("banana");
    Food banana2 = new Food("banana");
    Food chocolate = new Food("chocolate");

    Console.WriteLine(banana.Equals(null));
}
```

NullReferenceException
if banana is null

```
static void Main(string[] args)
{
    Food banana = new Food("banana");
    Food banana2 = new Food("banana");
    Food chocolate = new Food("chocolate");

    Console.WriteLine(banana.Equals(null));
    Console.WriteLine(object.Equals(banana, null));
    Console.WriteLine(object.Equals(null, banana));
    Console.WriteLine(object.Equals(null, null));
}
```

```
False
False
True
Press any key to continue . . .
```

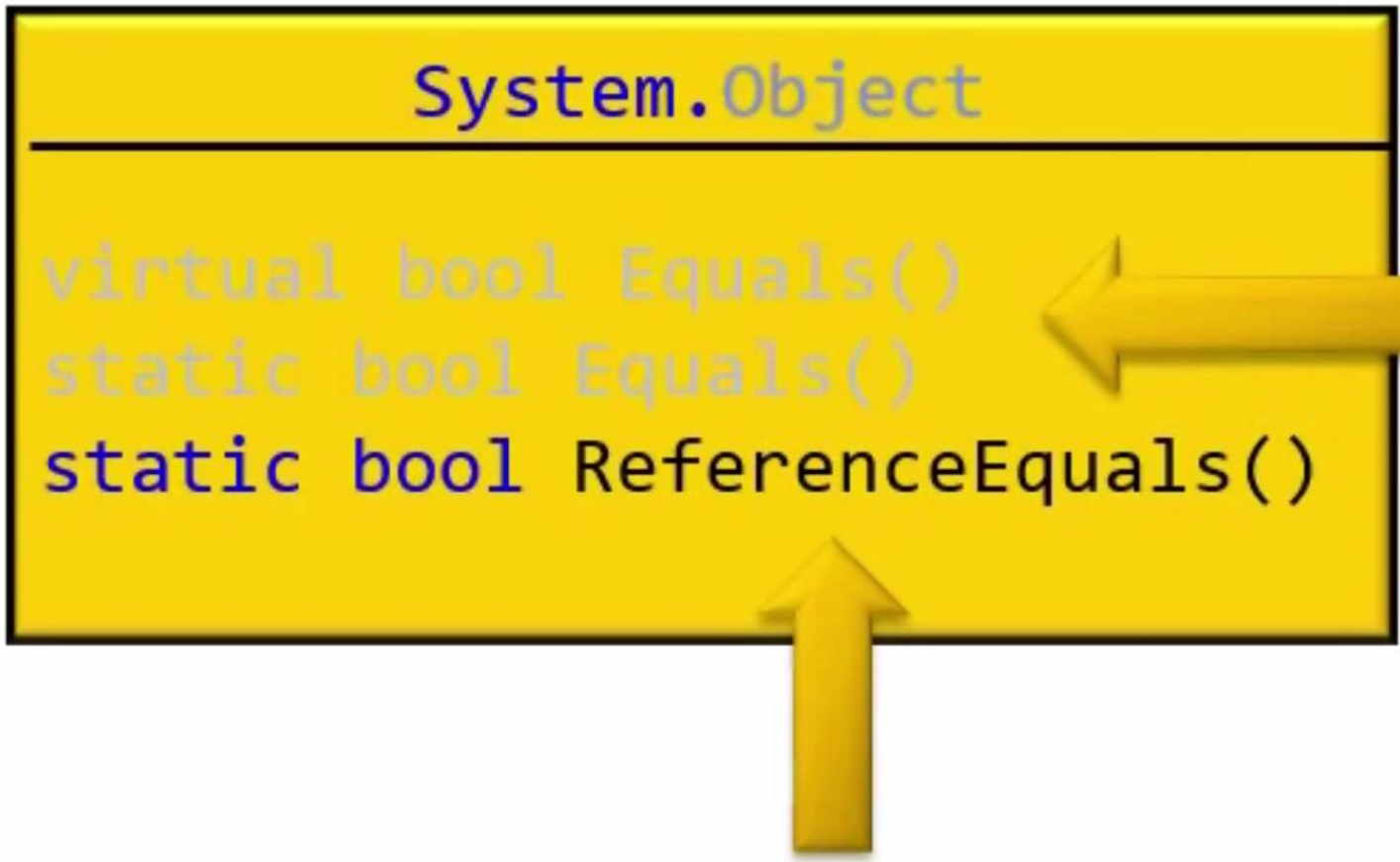
In .NET:
null ALWAYS equals null

What Static Equals() Does

```
public static bool Equals(object obj1, object obj2)
{
    if (obj1 == obj2)
        return true;
    if (obj1 == null || obj2 == null)
        return false;
    else
        return obj1.Equals(obj2);
}
```

Static method gives same
results as the virtual method
(except for extra null checking)

ReferenceEquals()

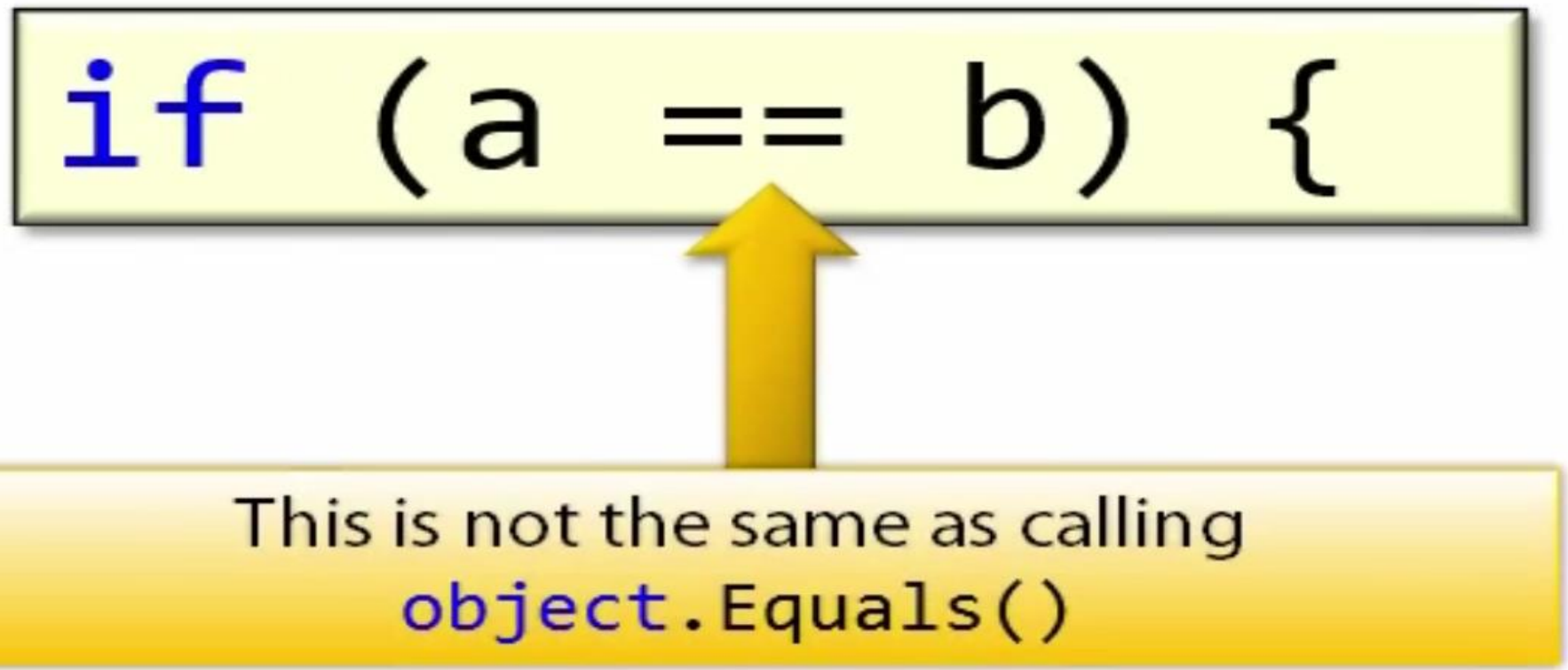


Usually compare references
– But not if overridden

Used to check whether two variables refer to the same instance

The C# Equality Operator

The C# == Operator



(But it often happens to give the same results)

Comparing == and Object.Equals()

	== Operator	object.Equals()
Primitive Types	Compare Values	
Reference Types (by default)	Compare References	

Can overload == and override `Equals()`

Implementing Equality for Reference Types

- Why implement equality for a *reference type*?
- Demo implementing equality:
  - Override `object.Equals()`.
  - Override `object.GetHashCode()`.
  - Implement == and != overloads.

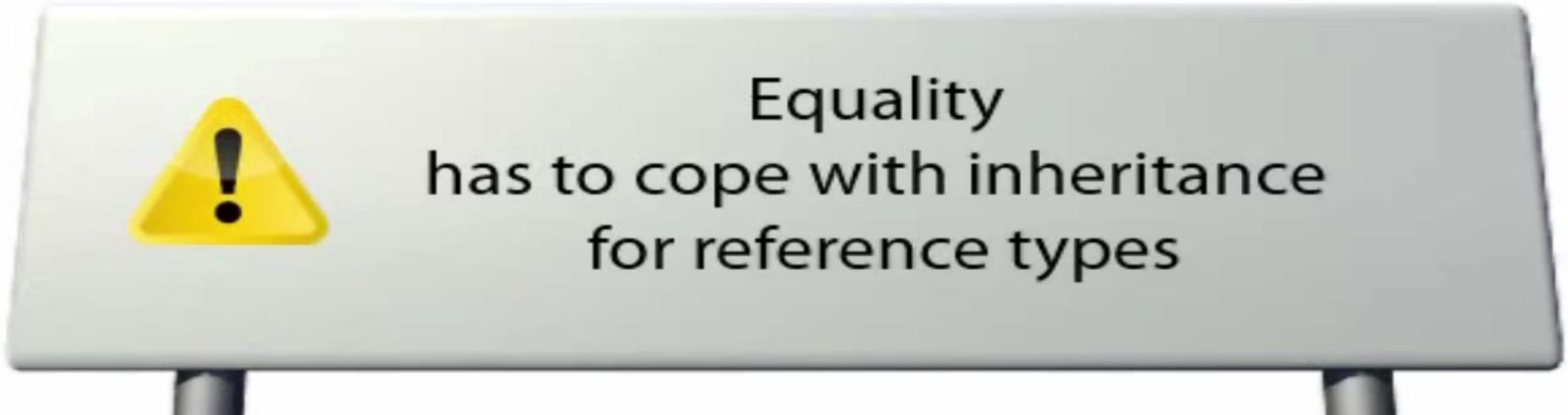
Equality for Reference Types

Most Reference Types

Any Value Type

Can inherit from

Implicitly sealed



Demo
Food and CookedFood

How Equality works in Inheritance

Overriding the Equals()

```
public class Food
{
    0 references
    public override bool Equals(object obj)
    {
        if (obj == null)
            return false;
        if (ReferenceEquals(obj, this))
            return true;
        if (obj.GetType() != this.GetType())
            return false;
        Food rhs = obj as Food;
        return this._name == rhs._name && this._group == rhs._group;
    }
```

Also Have to Override the GetHashCode()

To make the GetHashCode implementation match with what Equals() is doing

```
public override int GetHashCode()
{
    return this._name.GetHashCode() ^ this._group.GetHashCode();
}
```

GOOD PRACTICE
If you override the Equals() Also Overload the == Operator

```
public class Food
{
    public static bool operator ==(Food x, Food y)
    {
        return object.Equals(x, y);
    }
```

static object.Equals()
Does null-checking then calls
virtual Equals()

Implementing Equality for the Sub class CookedFood

```
public sealed class CookedFood : Food
{
    2 references
    public override bool Equals(object obj)
    {
        if (!base.Equals(obj))
            return false;
        CookedFood rhs = (CookedFood)obj;
        return this._cookingMethod == rhs._cookingMethod;
    }

    private string _cookingMethod;
```

If `base.Equals()` returns `true`
- We just need to check derived type fields

Return `true`
if `base.Equals()` returns `true`
and derived type fields are equal

Also Have to Override the GetHashCode() for Sub class

```
public override int GetHashCode()
{
    return base.GetHashCode() ^ this._cookingMethod.GetHashCode();
}
```

If you override the Equals() Also Overload the == and != Operator for

```
public static bool operator ==(CookedFood x, CookedFood y)
{
    return object.Equals(x, y);
}

0 references
public static bool operator !=(CookedFood x, CookedFood y)
{
    return !object.Equals(x, y);
}
```