The C# Equality Operator

Simon Robinson
http://TechieSimon.com
@TechieSimon

hardcore dev and IT training

7. Strings
4. Overriding Equality: Value Types
5. Overriding Equality: Reference Types
8. Comparers and Equality Comparers
10. Structural Equality
3. == in C#
6. Comparisons in .NET
2. Equality in .NET
9. Hash Codes
1.Why Is Equality So Hard?

Module Overview

- We'll compare == and `object.Equals()`.
- For value types, you can't use == unless it's been overloaded.
- If you override `object.Equals()`, *you should overload == too.*
- The == operator doesn't work well with:
  - Inheritance
  - Generics

! = is basically the same as ==,
except it returns the 'opposite' value

The C# == Operator

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

This is not the same as calling `object.Equals()`

(But it often happens to give the same results)

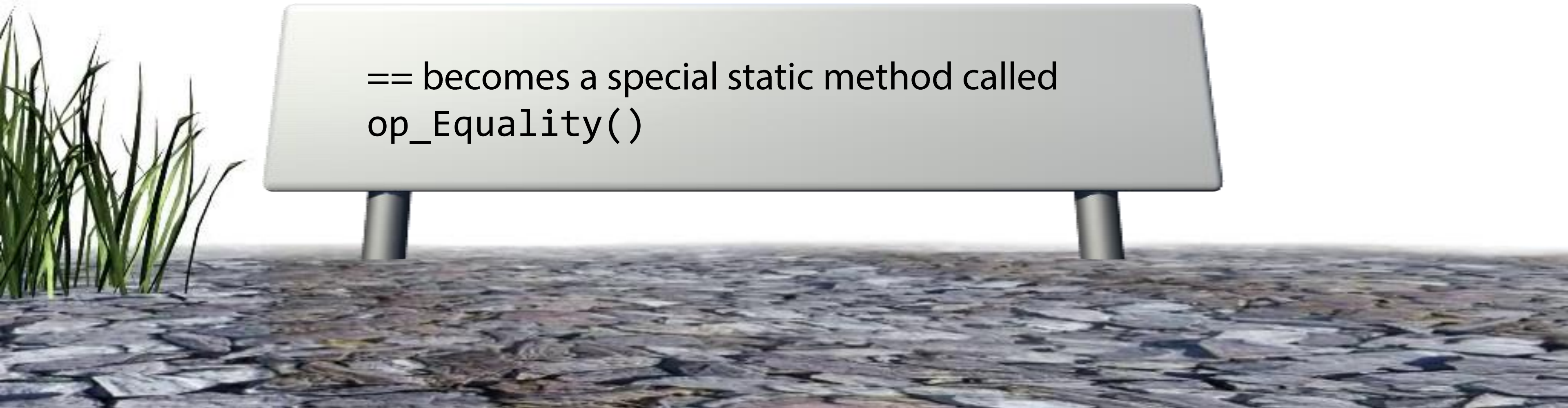
Code Demo

Code Demo

To Overload == ...

```
class MyType
{
    public static bool operator == (MyType lhs, MyType rhs)
    {
    // etc.
    }
```

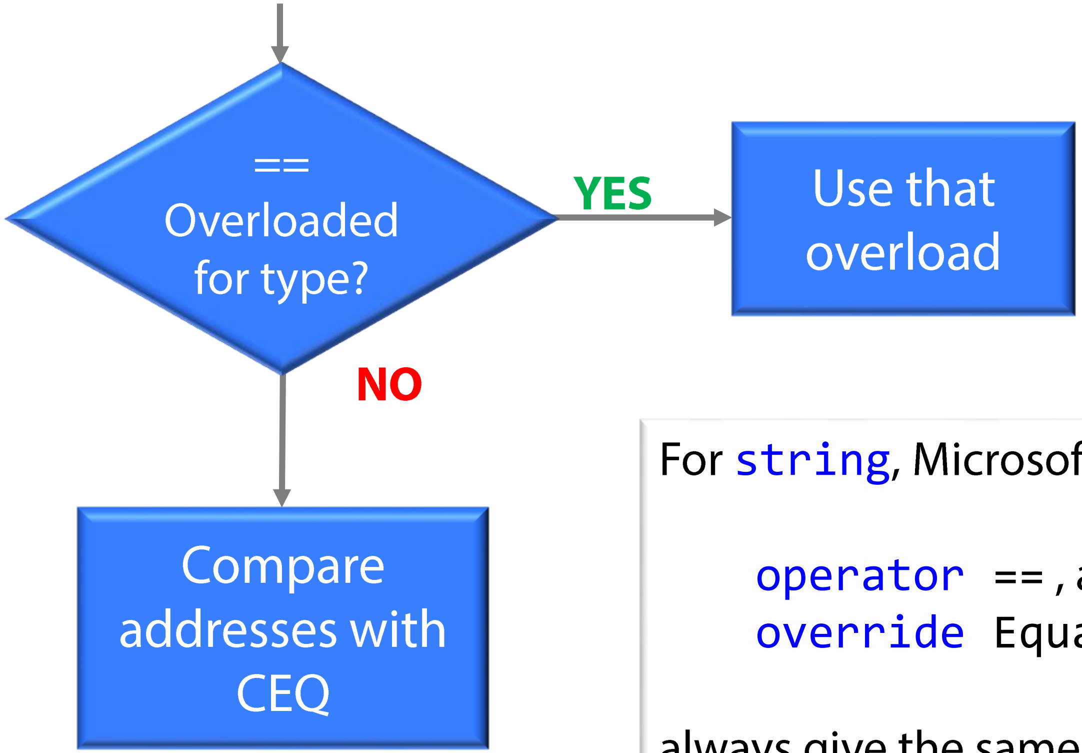
To overload ==
 Declare a 'static method' with the name `operator ==`



== becomes a special static method called `op_Equality()`

Code Demo

How Does == Work for Reference Types?



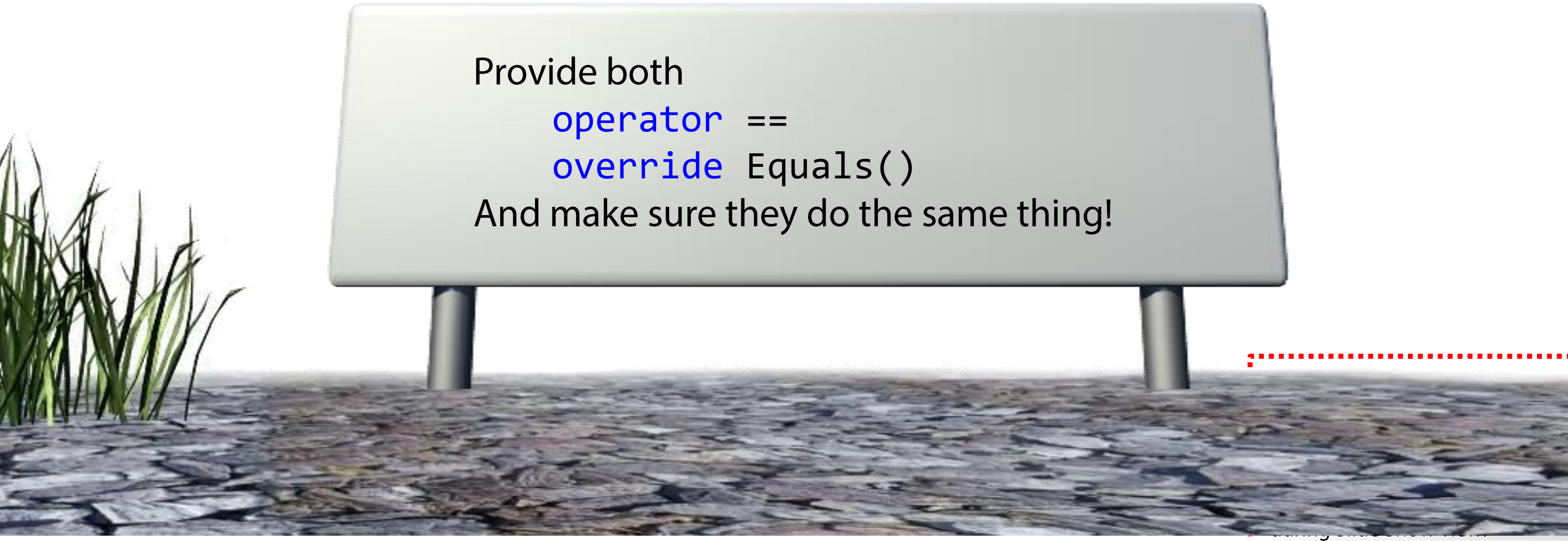
For `string`, Microsoft made sure that:

```
    operator ==, and
    override Equals()
```

always give the same result

Important Principle…

If you are changing how equality works for a type….



Provide both

```
    operator ==
    override Equals()
```

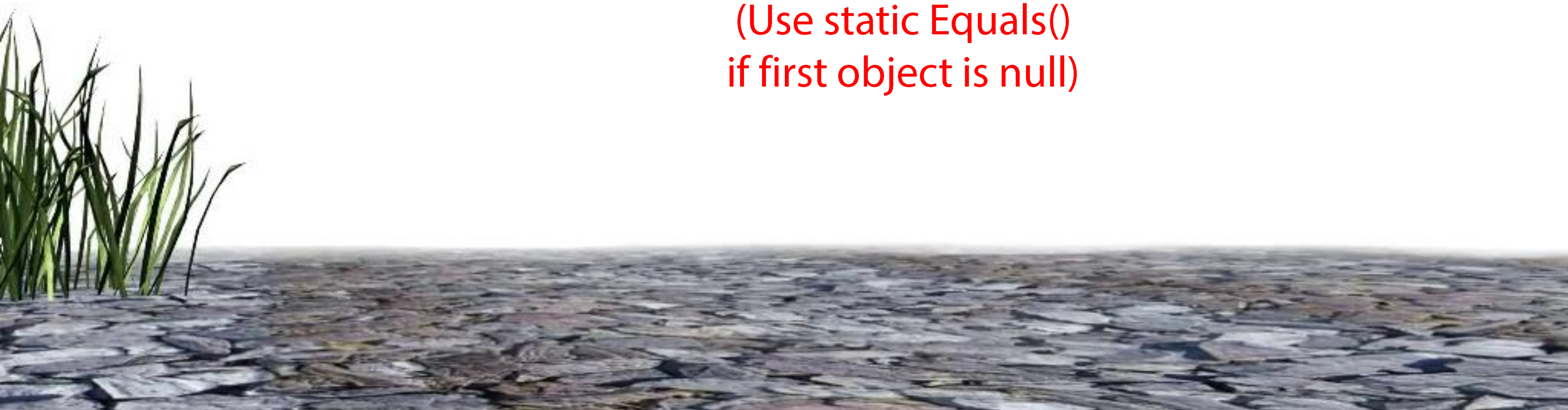
And make sure they do the same thing!

Comparing == and Object.Equals()

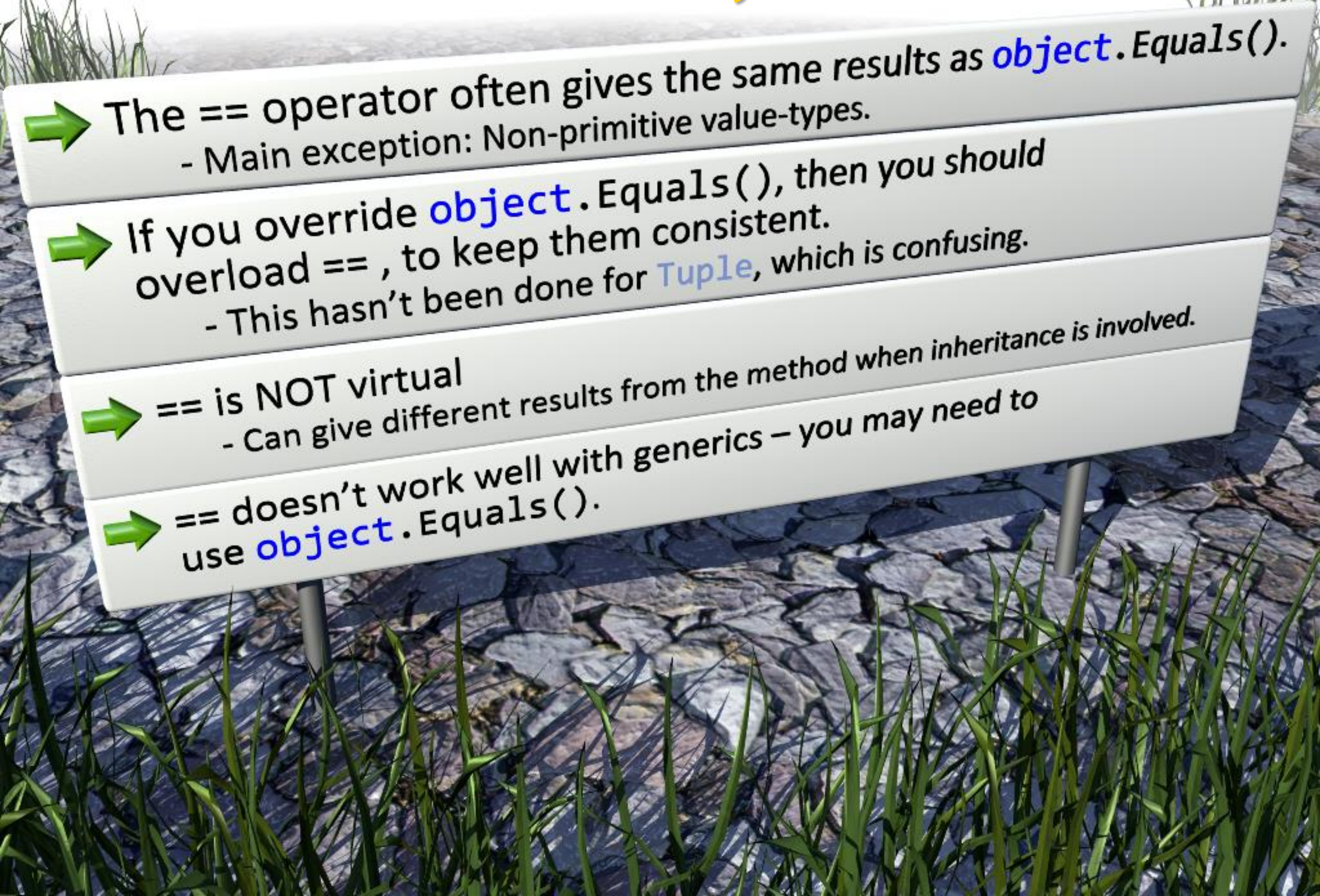
	== Operator	object.Equals()
Primitive Types	Compare Values	Compare Values
Reference Types (by default)	Compare References	Compare References
Value Types (by default)	❌ Not Available	Compares Values (but slow)

Can overload == and override `Equals()`

(Use static Equals() if first object is null)



Summary



- The == operator often gives the same results as `object.Equals()`.
  - Main exception: Non-primitive value-types.
- If you override `object.Equals()`, then you should overload == , to keep them consistent.
  - This hasn't been done for `Tuple`, which is confusing.
- == is NOT virtual
  - Can give different results from the method when inheritance is involved.
- == doesn't work well with generics – you may need to use `object.Equals()`.