Substituting Inheritance with Discriminated Unions

Zoran Horvat
CEO at Coding Helmet

@zoranh75 https://codinghelmet.com

“The only constant in software is change”

Ancient proverb

Variation at the Function Level

Implementation

```
f(args)
{
  // process args
}
```

Variation at the Function Level

Implementation

```
f(g, args)
{
  // process args
}
```

Varying behavior

This function's responsibility

Variation at the Function Level

Implementation

```
f(g, args)
{
  // process args
  g(); // call dependency
}
```

Varying behavior

This function's responsibility

Some other responsibility

Variation at the Function Level

Implementation

```
f(g, args)
{
  // process args
  g(); // call dependency
}
```

Consumer

```
f(p, args);
f(q, args);
f(r, args);
...
```

Different behaviors

Variation at the Function Level

Implementation

```
f(g, args)
{
  // process args
  g(); // call dependency
}
```

Consumer

```
f(p, args);
f(q, args);
f(r, args);
...
```

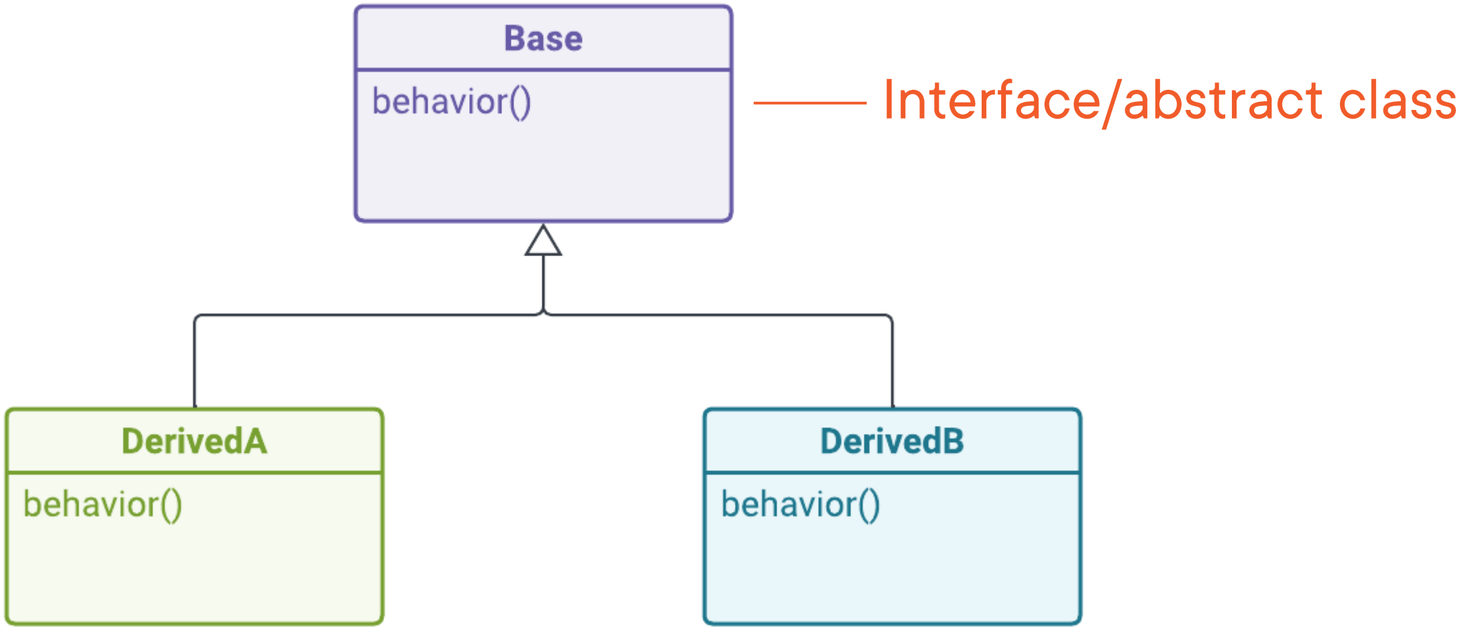
```
h = f(p);
h = f(q);
h = f(r);
...
```

Partially applied function

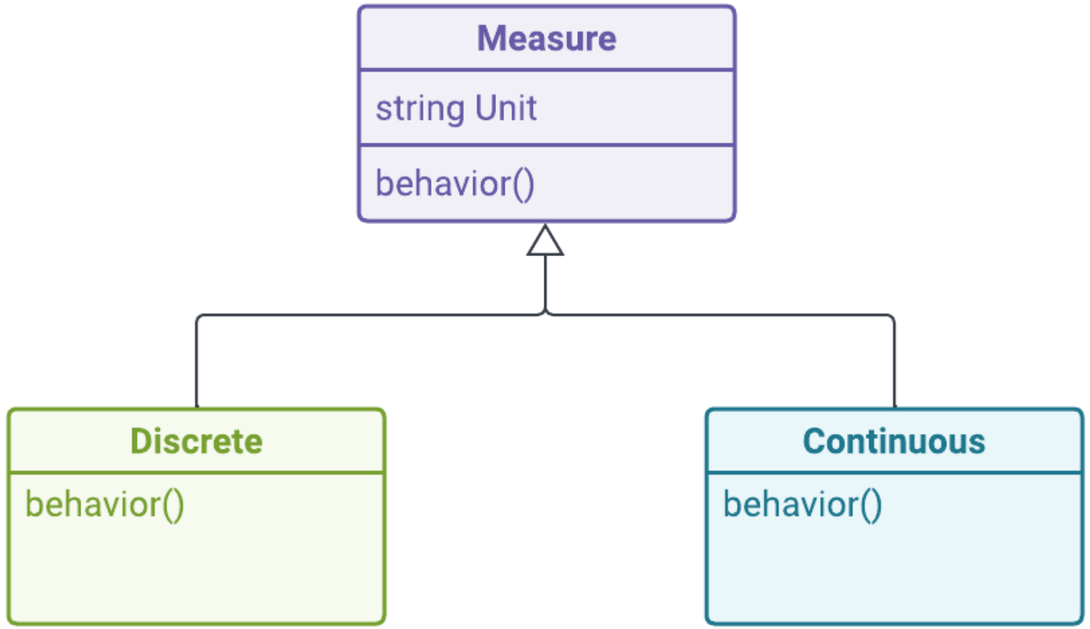
```
h(args);
```

Uniform invocation of a varying behavior

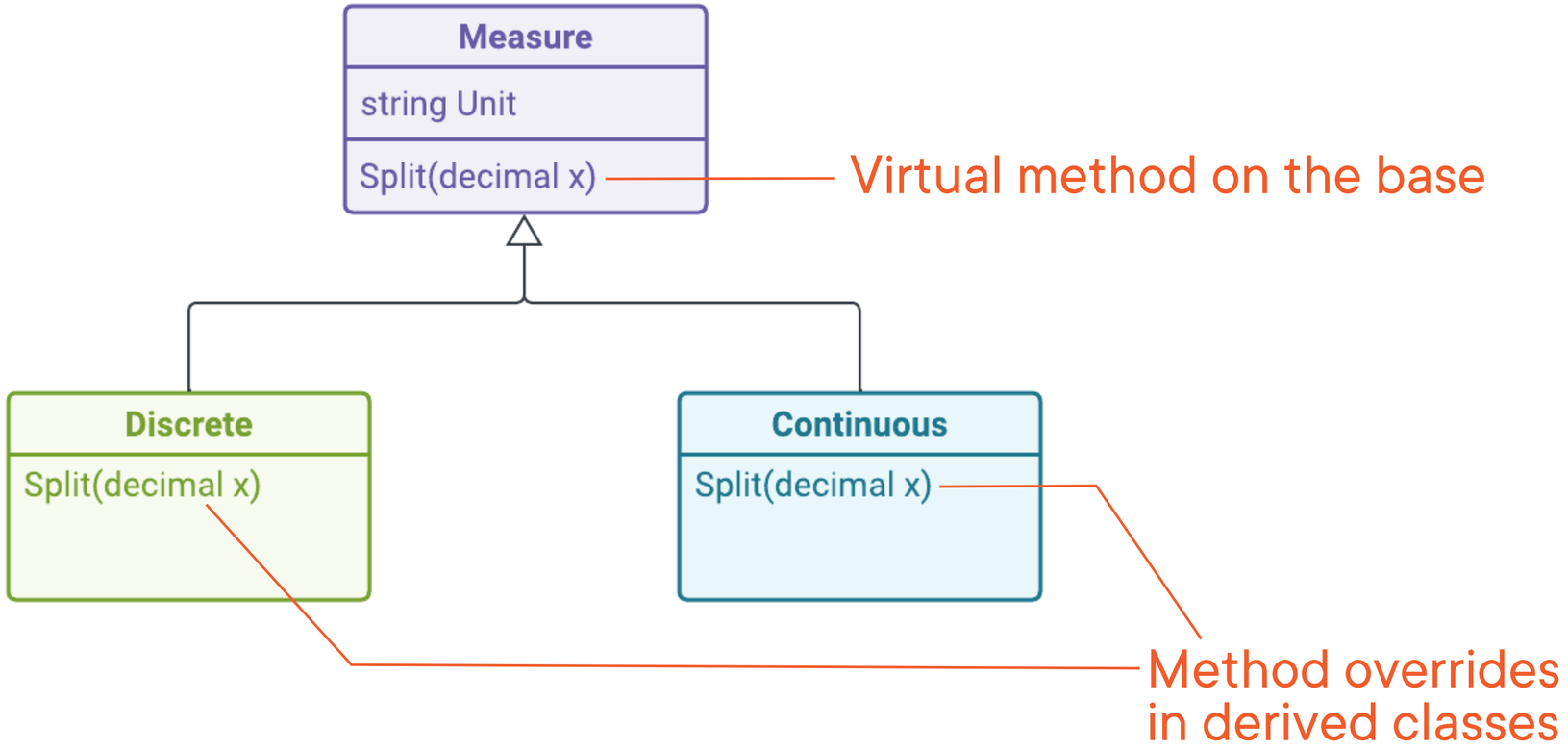
Understanding Polymorphic Execution



Understanding Polymorphic Execution

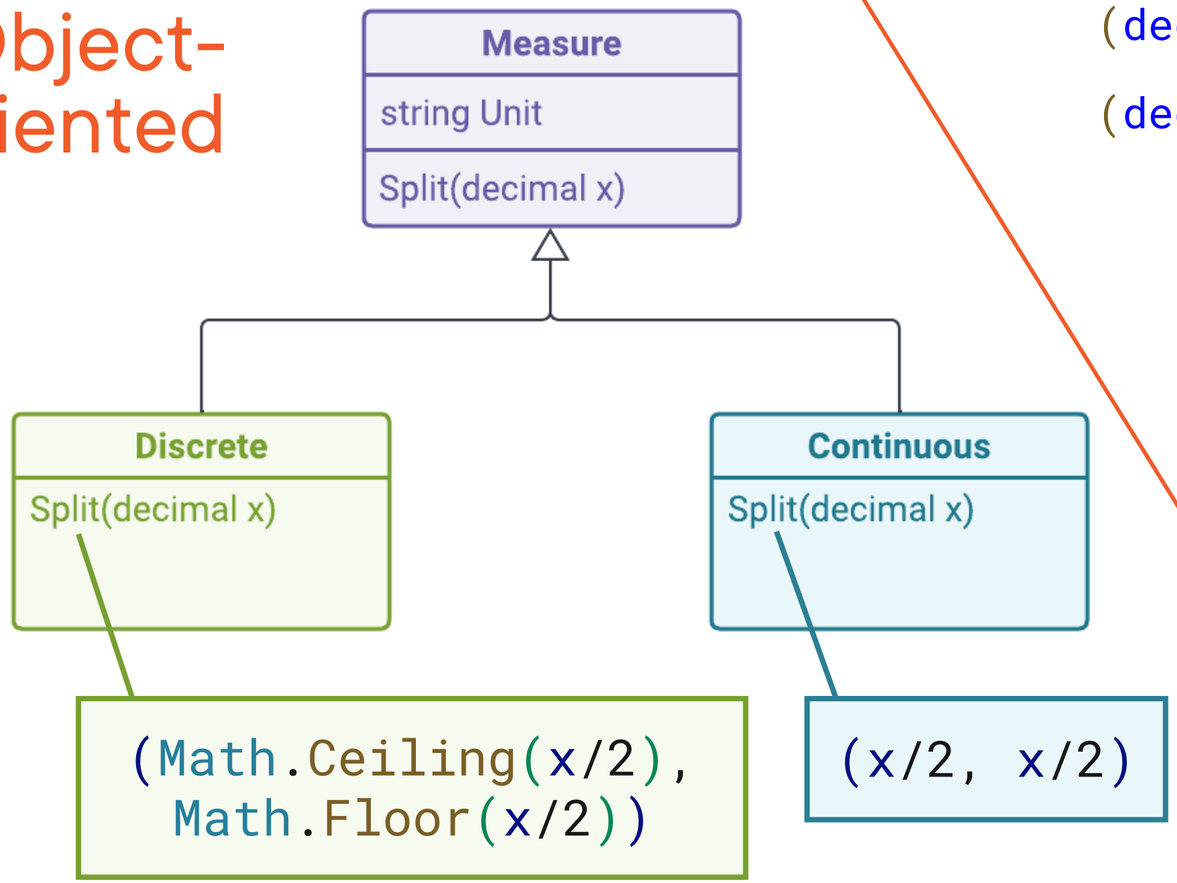


Understanding Polymorphic Execution



Understanding Polymorphic Execution

Object-oriented

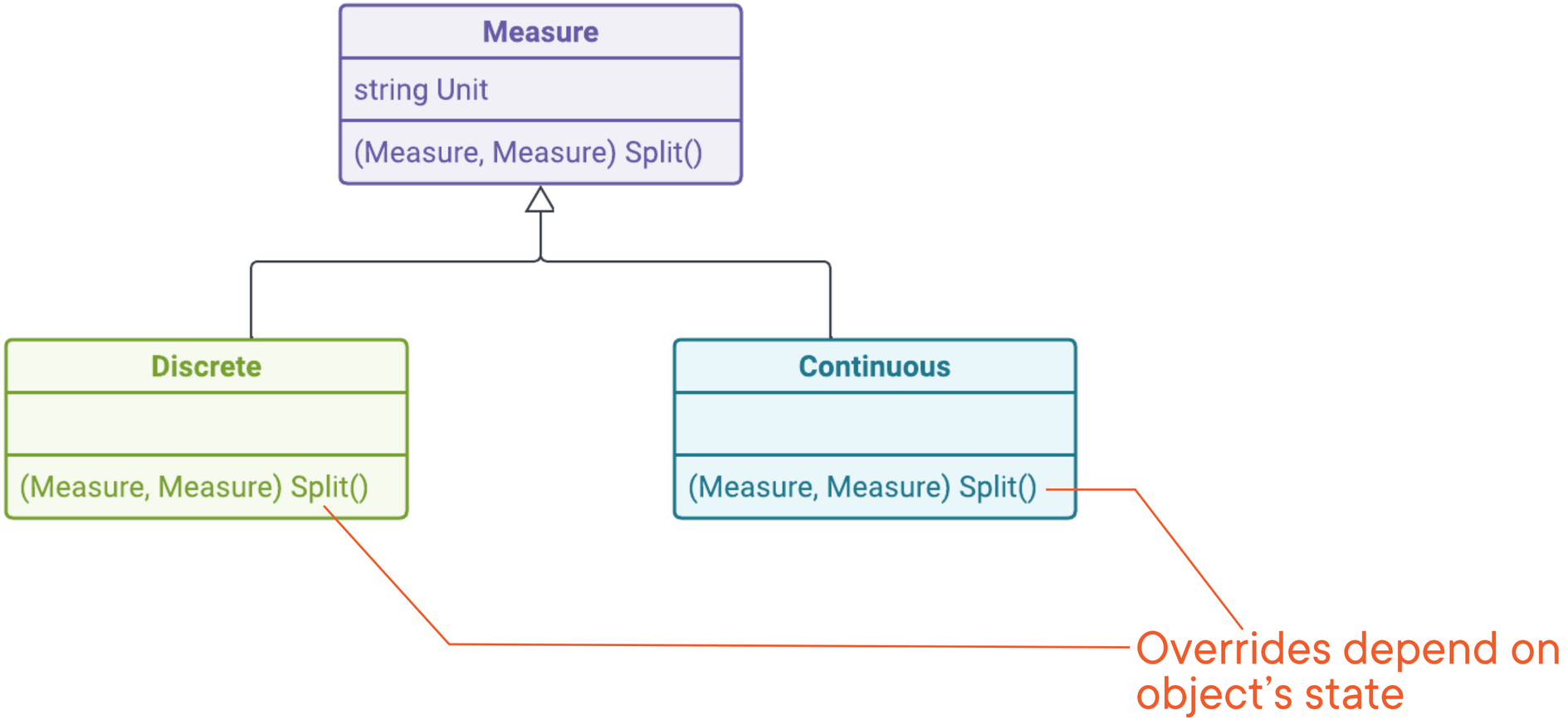



```
(decimal, decimal) SplitDiscrete(decimal x)

(decimal, decimal) SplitContinuous(decimal x)
```

```
var f = SplitDiscrete;
        or
var f = SplitContinuous;

...
f(x);
```

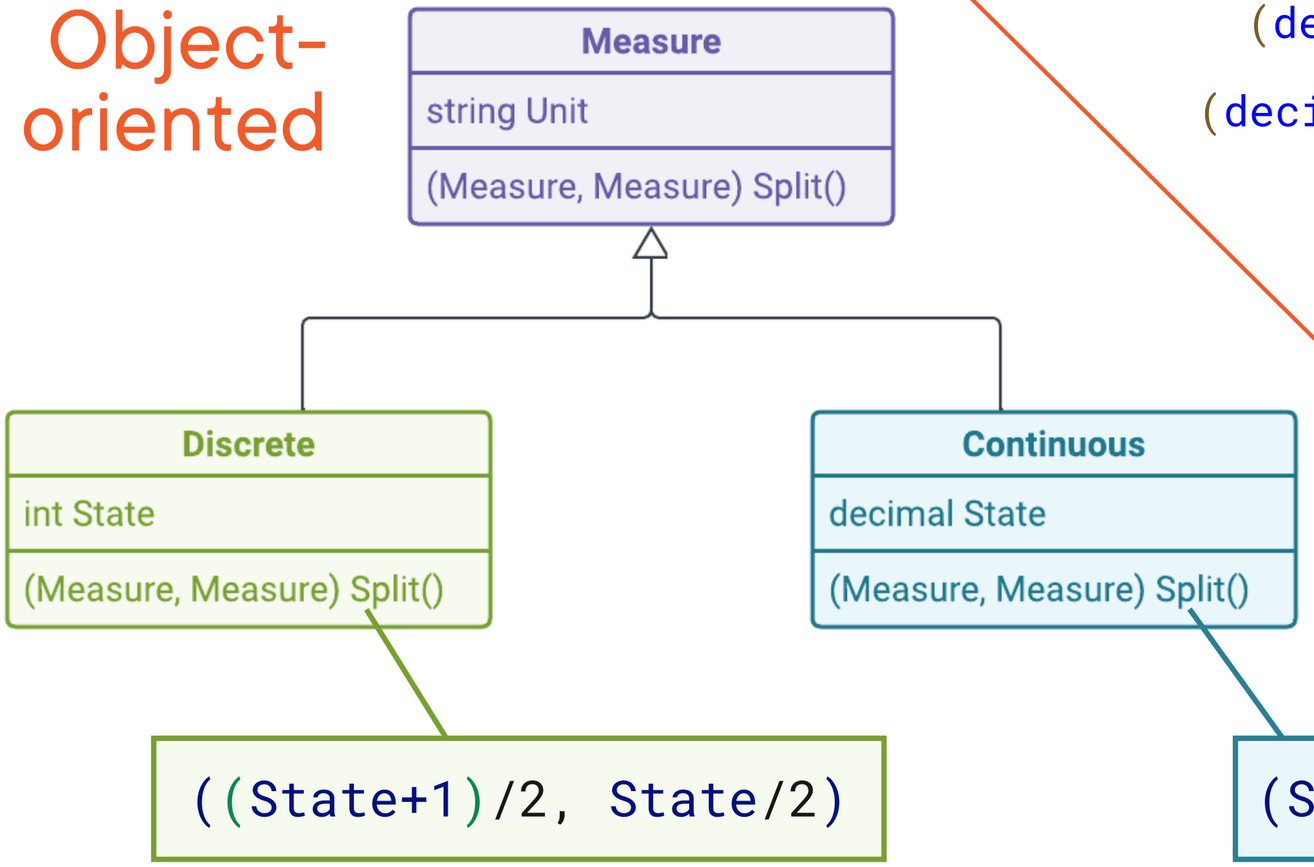
Functional

Understanding Polymorphic Execution



Understanding Polymorphic Execution

Object-oriented



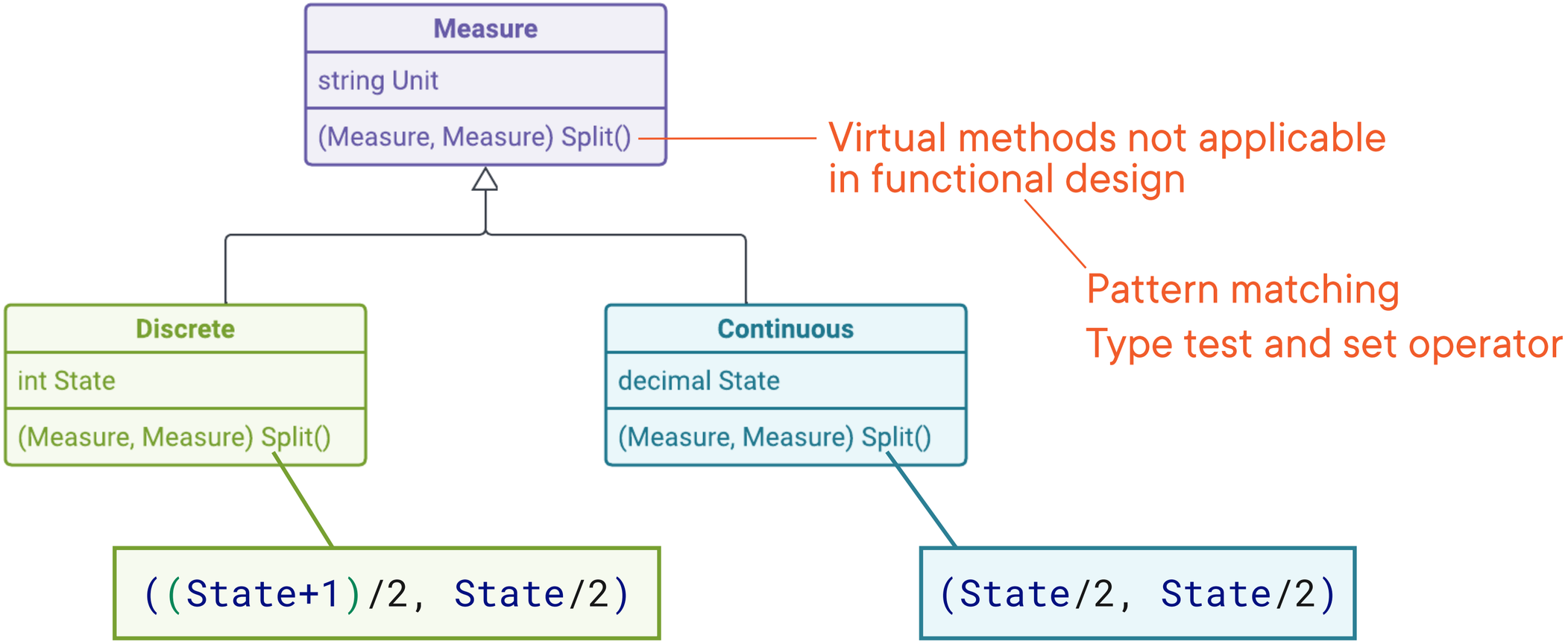

```
(decimal, decimal) SplitDiscrete(decimal x)
(decimal, decimal) SplitContinuous(decimal x)
```

All variants must have the same signature

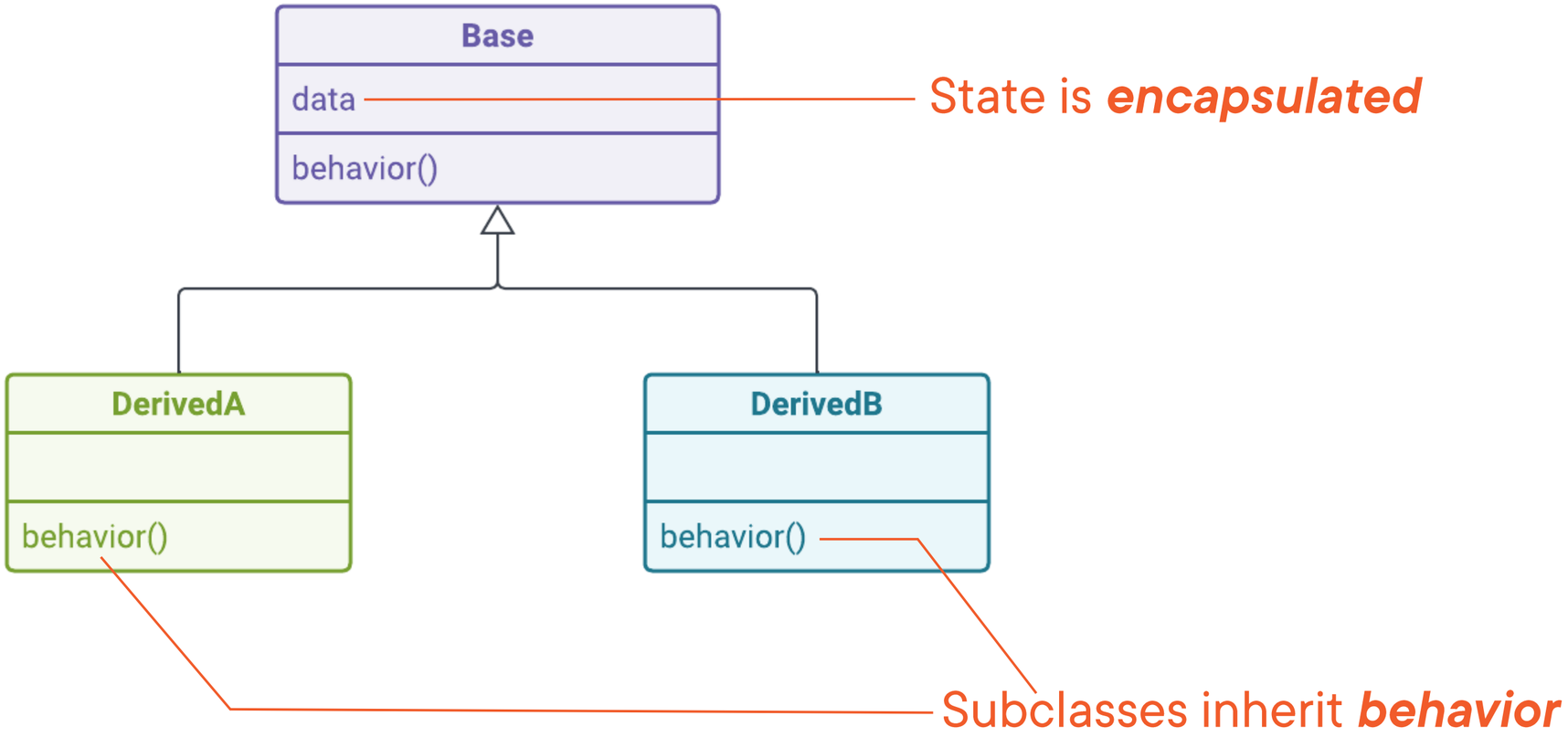
```
var f = SplitDiscrete;
var f = SplitContinuous;
```

Functional

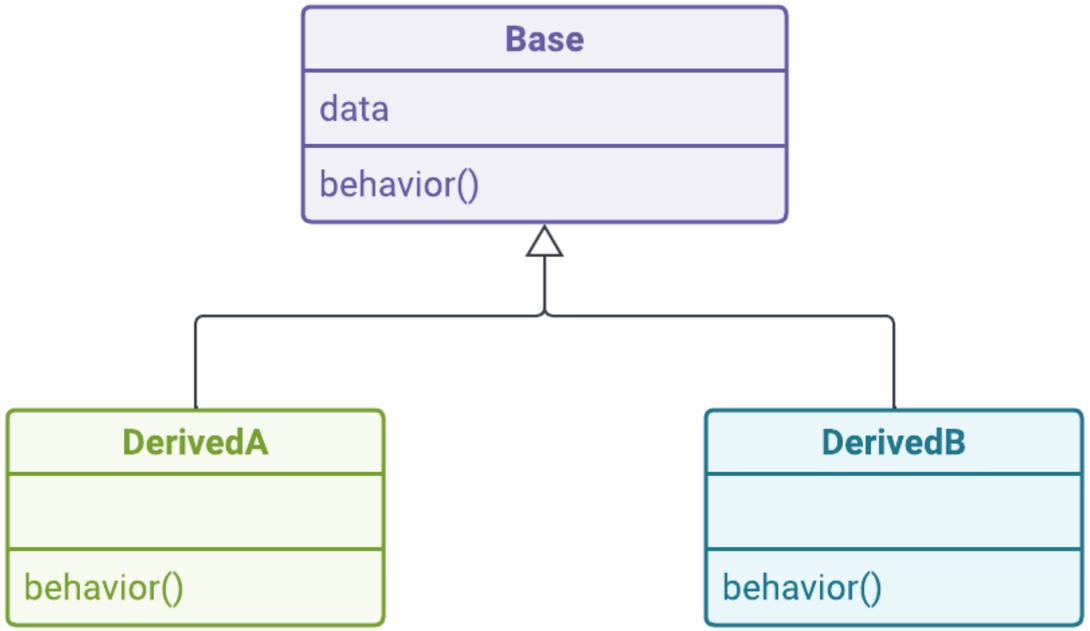
Understanding Polymorphic Execution



Varying Behavior



Varying Behavior



Object-oriented types

Functional types

Public components

Accessed by ***functions***

Extending a type

```
1  namespace Models.Common;
2
3  using Models.Types.Common;
4
5  public static class MeasureTransforms
6  {
7      public static (Measure a, Measure b) SplitInHalves(this Measure m) =>
8          m switch
9          {
10
11         };
12 }
```

`switch` expression (C# 8)

- Each branch is an expression
- Each branch is ***evaluated***
- Entire switch expression becomes an ***expression***
- Use switch expression in expression-bodied methods
- Assign switch expression to a variable

```
m switch
{

};
```

vs. “old style” `switch`

- Blocks of code to execute
- Mandatory `break` in every `case`
  (prevents unwanted “fall-through”, as in C++)
- Requires `return` inside `case` to return a result

DEMO
Application
Models
Common
Code39.cs
MeasureTransforms.cs
Media
Time
Types
Common
Measure.cs
Month.cs
Year.cs
Components
ExternalPart.cs
ExternalSku.cs
ExternalSkuPhoto.cs
Part.cs
StockKeepingUnit.cs
Vendor.cs
Media
Models.csproj
TestPersistence
Web
Demo.sln
TIMELINE

```
1 namespace Models.Types.Components;
2
3 public abstract record InventoryItem(Guid Id, string Name, StockKeepingUnit Sku);
4
5 public record Part(Guid Id, string Name, StockKeepingUnit Sku)
6     : InventoryItem(Id, Name, Sku);
7
8 public record Material(Guid Id, string Name, StockKeepingUnit Sku)
9     : InventoryItem(Id, Name, Sku);
```

Part: Name="BC547", SKU="ELTRBC547"

There is no such thing as dividing a part into two

Material: Name="Soldering alloy ingot", SKU="SLD10"

Dividing a material results in portions
that retain the qualities of original material

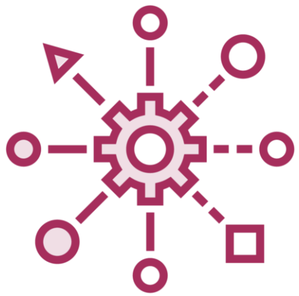
```
4
5  public abstract record InventoryItem(Guid Id, string Name, StockKeepingUnit Sku);
6
7  public record Part(Guid Id, string Name, StockKeepingUnit Sku)
8      : InventoryItem(Id, Name, Sku);
9
10 public record Material(Guid Id, string Name, StockKeepingUnit Sku,
11                        ContinuousMeasure Quantity)
12     : InventoryItem(Id, Name, Sku);
13
14 public record Portion(Guid Id, string Name, StockKeepingUnit Sku,
15                       ContinuousMeasure Quantity)
16     : InventoryItem(Id, Name, Sku);
17
18 public static class InventoryItemExtensions
19 {
20     public static InventoryItem AsDiscriminatedUnion(this InventoryItem item) =>
21         item switch
22         {
23             Part or Material => item,
24             _ => throw new InvalidOperationException(
25                 $"Not defined for object of type {item?.GetType().Name ?? "<null>"}")
26         };
27
28     public static TResult MapAny<TResult>(this InventoryItem item,
```

The language of purchasing and stockkeeping

```
public record Part(Guid Id, string Name, StockKeepingUnit Sku)

public record Material(Guid Id, string Name, StockKeepingUnit Sku,
                       ContinuousMeasure Quantity)
```

```
public record Portion(Guid Id, string Name, StockKeepingUnit Sku,
                      ContinuousMeasure Quantity)
```

The language of processes in the production plant



Disparate data incur special cases

Persistent state becomes corrupt

“Make illegal states unrepresentable”

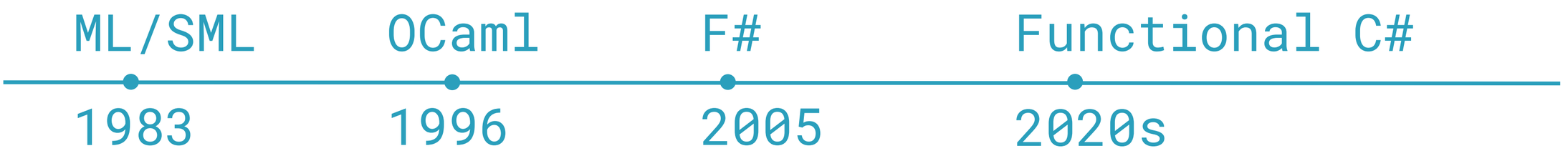
Scott Wlaschin
https://fsharpforfunandprofit.com/posts/
designing-with-types-making-illegal-states-unrepresentable/

Yaron Minsky
https://blog.janestreet.com/effective-ml-revisited/

“Make illegal states unrepresentable”

Scott Wlaschin
https://fsharpforfunandprofit.com/posts/
designing-with-types-making-illegal-states-unrepresentable/

Yaron Minsky
https://blog.janestreet.com/effective-ml-revisited/



```
4
5  public abstract record InventoryItem(Guid Id, string Name, StockKeepingUnit Sku);
6
7  public record Part(Guid Id, string Name, StockKeepingUnit Sku)
8      : InventoryItem(Id, Name, Sku);
9
10 public record Material(Guid Id, string Name, StockKeepingUnit Sku,
11                        ContinuousMeasure Quantity)
12     : InventoryItem(Id, Name, Sku);
13
14 public record Portion(Guid Id, string Name, StockKeepingUnit Sku,
15                       ContinuousMeasure Quantity)
16     : InventoryItem(Id, Name, Sku);
17
18 public static class InventoryItemExtensions
19 {
20     public static InventoryItem AsDiscriminatedUnion(this InventoryItem item) =>
21         item switch
22         {
23             Part or Material => item,
24             _ => throw new InvalidOperationException(
25                 $"Not defined for object of type {item?.GetType().Name ?? "<null>"}")
26         };
27
28     public static TResult MapAny<TResult>(this InventoryItem item,
```

Summary

Working with discriminated unions in C#

- Make possible to vary data representation
- Applies where function variation does not

Defining a discriminated union

- Base type (interface, abstract class)
- Multiple direct inheritors
- Inheriting type carries additional meaning
- An inheritor may add specific properties
- There can only be one level of inheritance

Summary

Using discriminated unions in C#

- Heavy use of type test and set pattern matching constructs
- Often using the `switch` expression

Summary

Comparing with method overriding

- Functions on discriminated unions are different from object-oriented methods
- Easy to add a new function on a discriminated union
- Hard to add a new type to a discriminated union

Managing a class hierarchy

- Adding a class to a hierarchy is cheap
- Adding a new abstract function to base is a breaking change

Up Next:
Modeling Missing Objects