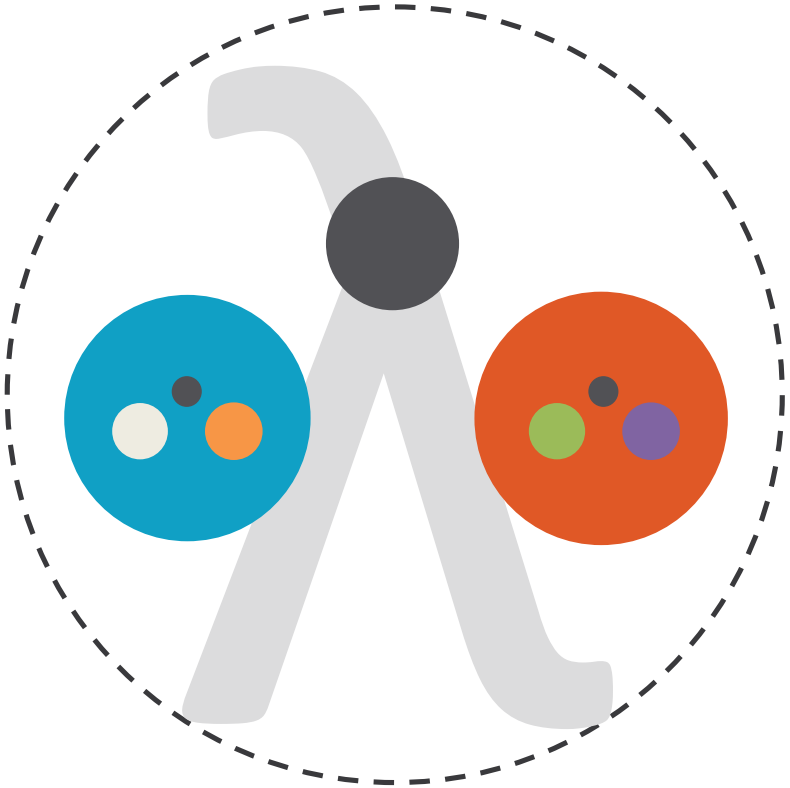
Giving Clarity to LINQ Queries *by* Extending Expressions



Ed Charbeneau

Developer Advocate, Telerik

Code PaLOUsa Co-Chairman

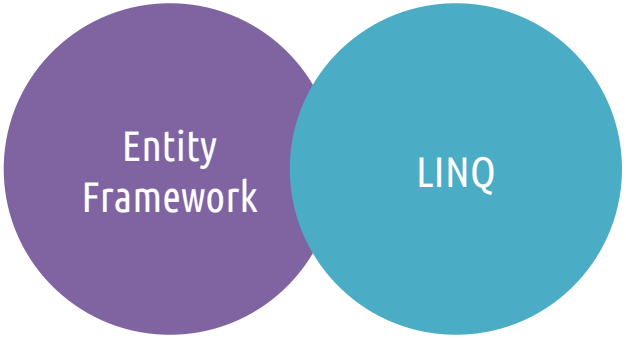
Author: TDN & Simple-Talk

Podcast: Eat Sleep Code the Official Telerik Podcast

Twitter: @EdCharbeneau

Do you use expressions?

Common uses



Expression<Func<TModel, bool>>

People.Where(p => p.Title == “Developer”)

Expression<Func<TModel, TResult>>

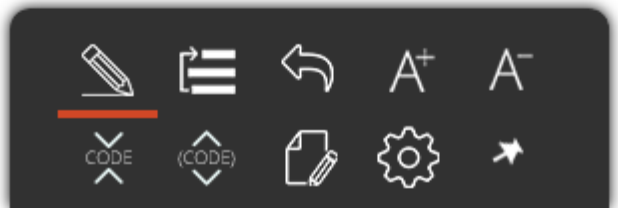
@Html.LabelFor(obj => obj.Prop)

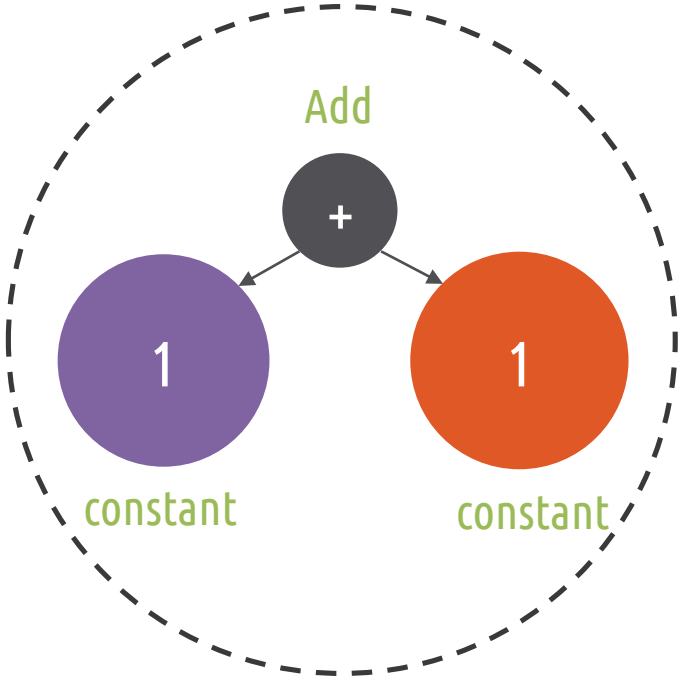
Expressions in C#

- Representation of code as data
- Meta-programming
  - Analyze, rewrite, and translate code at runtime

Expression Factory

```
1    // new Expression() invalid!
2
3    var x = Expression.Constant(1);
4    var y = Expression.Constant(1);
5    var sum = Expression.Add(x,y);
6
7    Console.WriteLine(sum);
8    //(1 + 1)
9
10   Console.WriteLine(sum.GetType());
11   //SimpleBinaryExpression
12
```



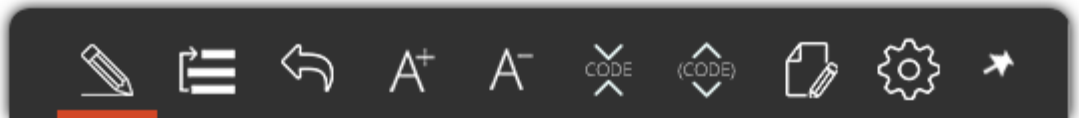


SimpleBinaryExpression

Homo-iconicity

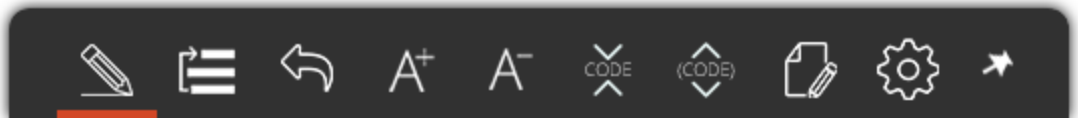
Same syntax for executable code as data representation

```
1   //Executable
2   Func<int, int> plusOne = n => n +1;
3
4   //Data representation
5   Expression<Func<int, int>> plusOne = n => n +1;
6
```



Compile

```
1   Expression<Func<int, int>> plusOne = n => n + 1;
2   Console.WriteLine(plusOne.ToString());
3   //n => (n + 1)
4
5   var x = plusOne.Compile();
6   Console.WriteLine(x(1));
7   //2
8
9
```



Runtime Modification

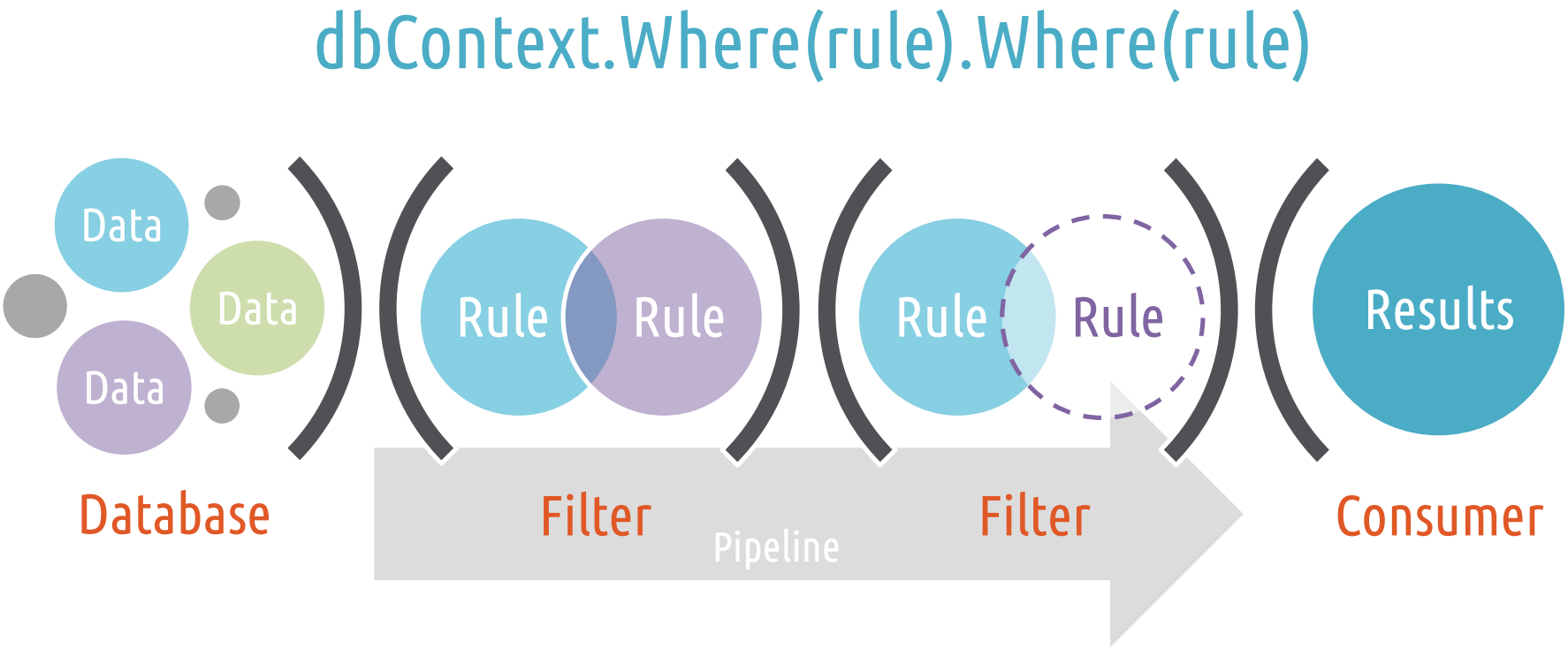
ExpressionVisitor

- used to traverse or rewrite expression trees
- Abstract class
  - Inherit and override
- .Visit(expression)
  - Recursively walks the tree
  - Returns an Expression

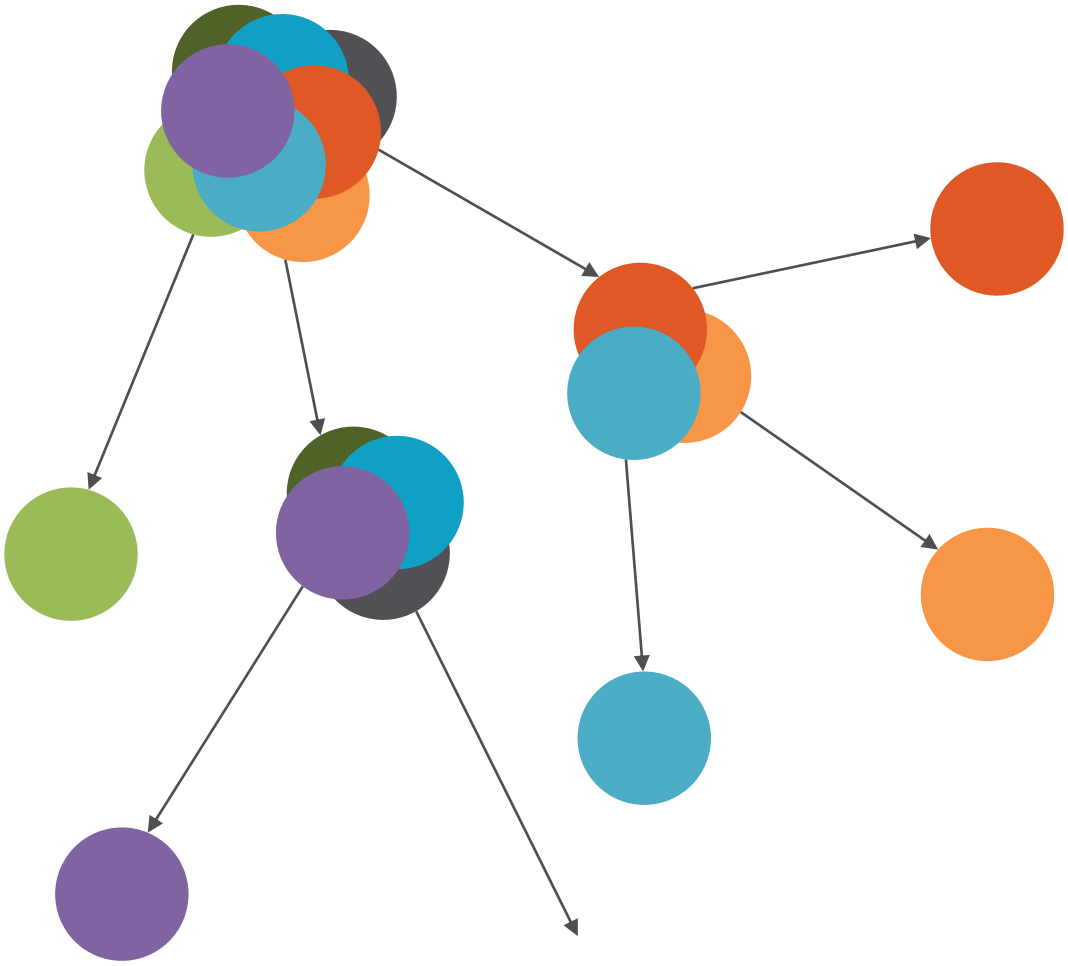
https://msdn.microsoft.com/en-us/library/bb882521(v=vs.90).aspx

How is this useful?

Pipes and filters



Maintainability & Readability



Refactoring

```
1    IQueryable<Post> posts = postRepository.GetAll()
2                    .Where(post => post.IsPublished && post.PostedOn <= today &&
3    (post.PostedOn >= cutoffDate || post.Author == featuredAuthor &&
4    post.PostedOn >= featuredAuthorCutoffDate));
5
```

Let's start simple

```
1   postRepository.GetAll()
2   .Where(post => post.IsPublished && post.PostedOn <= today);
3
```

Distinction

The tale of two extension methods

IEnumerable.Where(q => q.Value == true)

Func (delegate)

Func<T, bool>

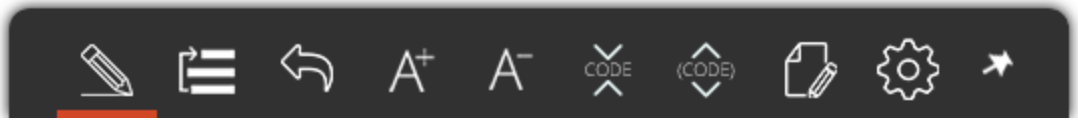
IQueryable.Where(q => q.Value == true)

Expression

Expression<Func<T, bool>

LINQ Where Chaining

```
1   //Both statements produce the same result
2
3   //Concise
4   var query = query.Where(x => x.Value == 1 && x.Name == name);
5
6   //Configurable
7   var query = query.Where(x => x.Value == 1)
8                    .Where(x => x.Name == name);
9
```



Custom Filters


```
IQueryable<MyType> IQueryable<MyType>.CustomMethodName()
```

Return type Extends type Method

```
public static IQueryable<Post> ArePublished(this IQueryable<Post> posts)
{
        return posts.Where(post => post.IsPublished);
}
```

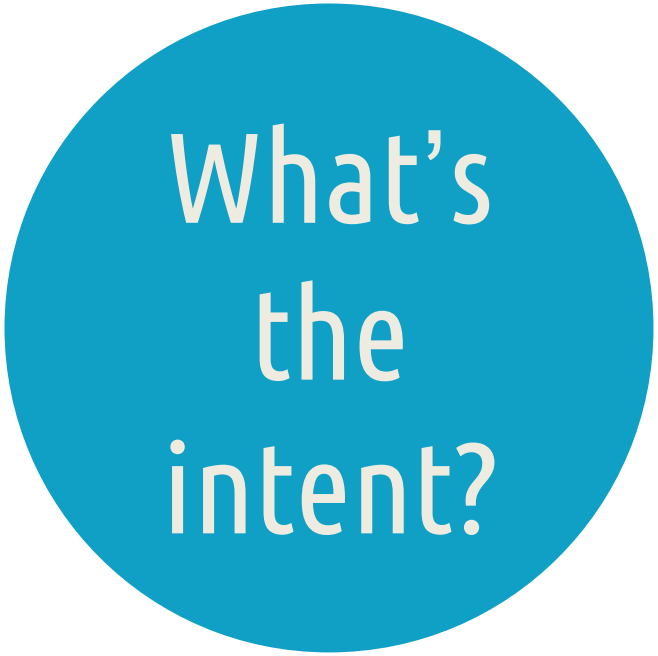
Readability?

Before

```
1   postRepository.GetAll()
2  .Where(post => post.IsPublished &&
3   post.PostedOn <= today);
4
```

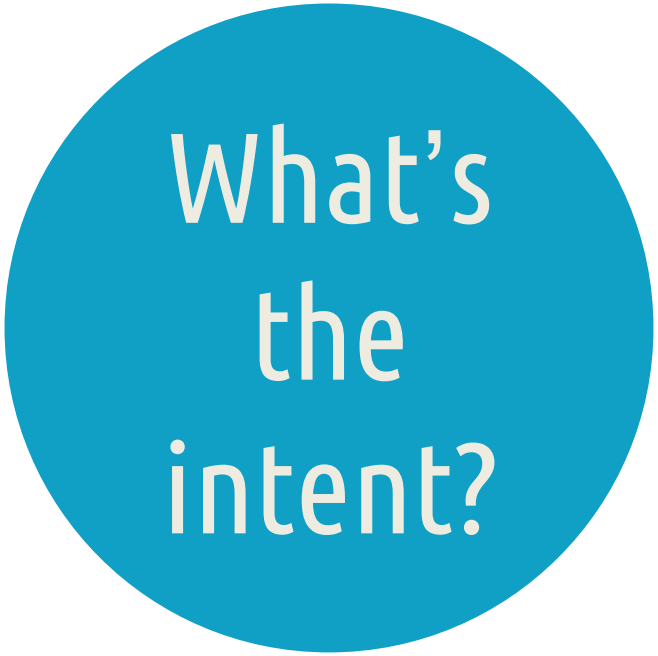
After

```
1   postRepository.GetAll()
2   .ArePublished()
3   .PostedOnOrBefore(today);
4
```



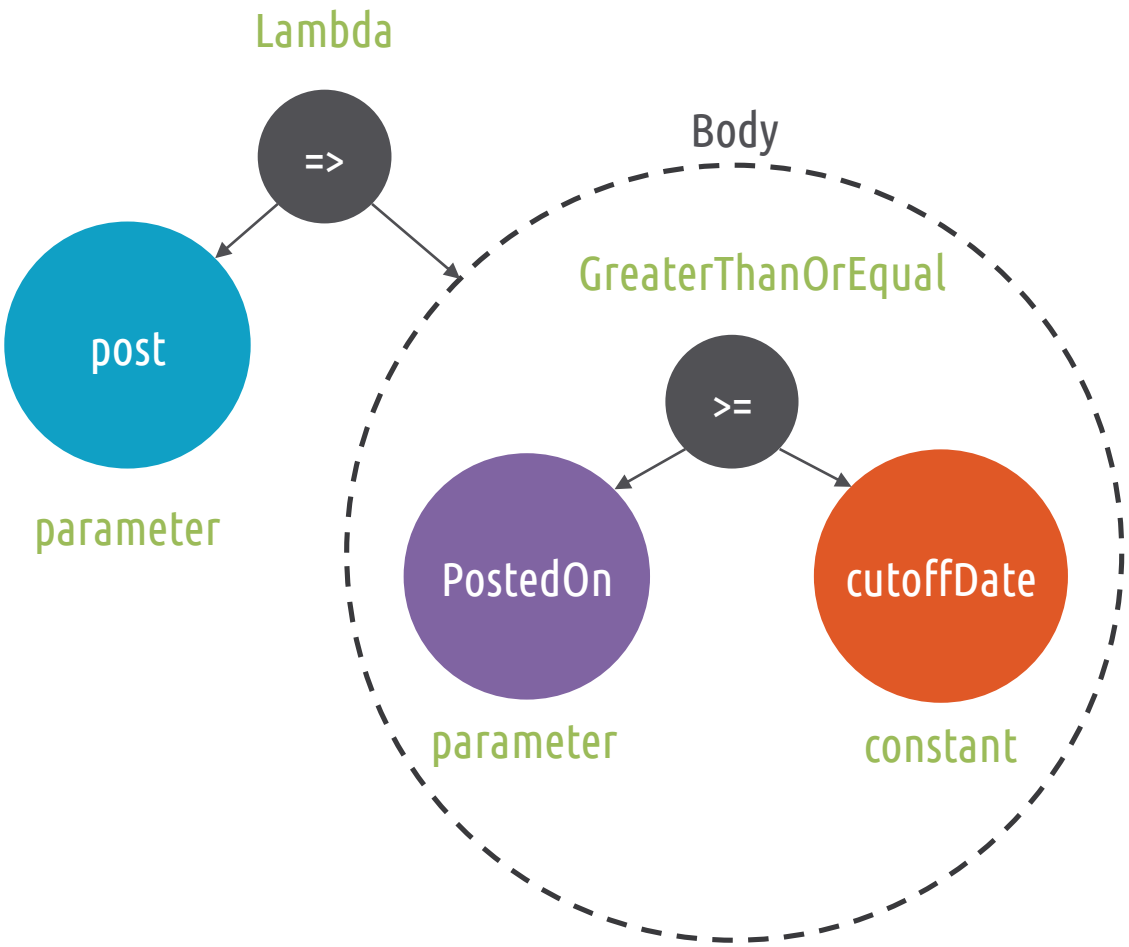
Now what?

```
1    IQueryable<Post> posts = postRepository.GetAll()
2                    .Where(post => post.IsPublished && post.PostedOn <= today &&
3    (post.PostedOn >= cutoffDate || post.Author == featuredAuthor &&
4    post.PostedOn >= featuredAuthorCutoffDate));
5
```

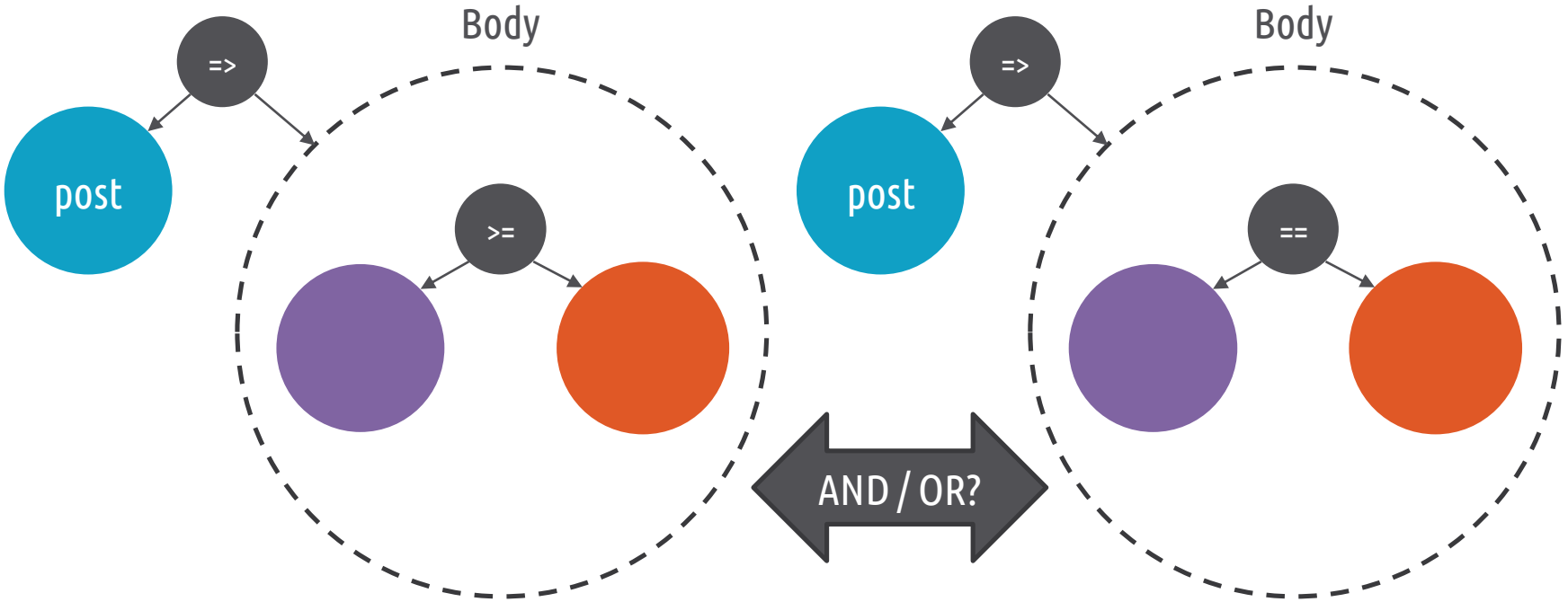


Lambda as an Expression tree

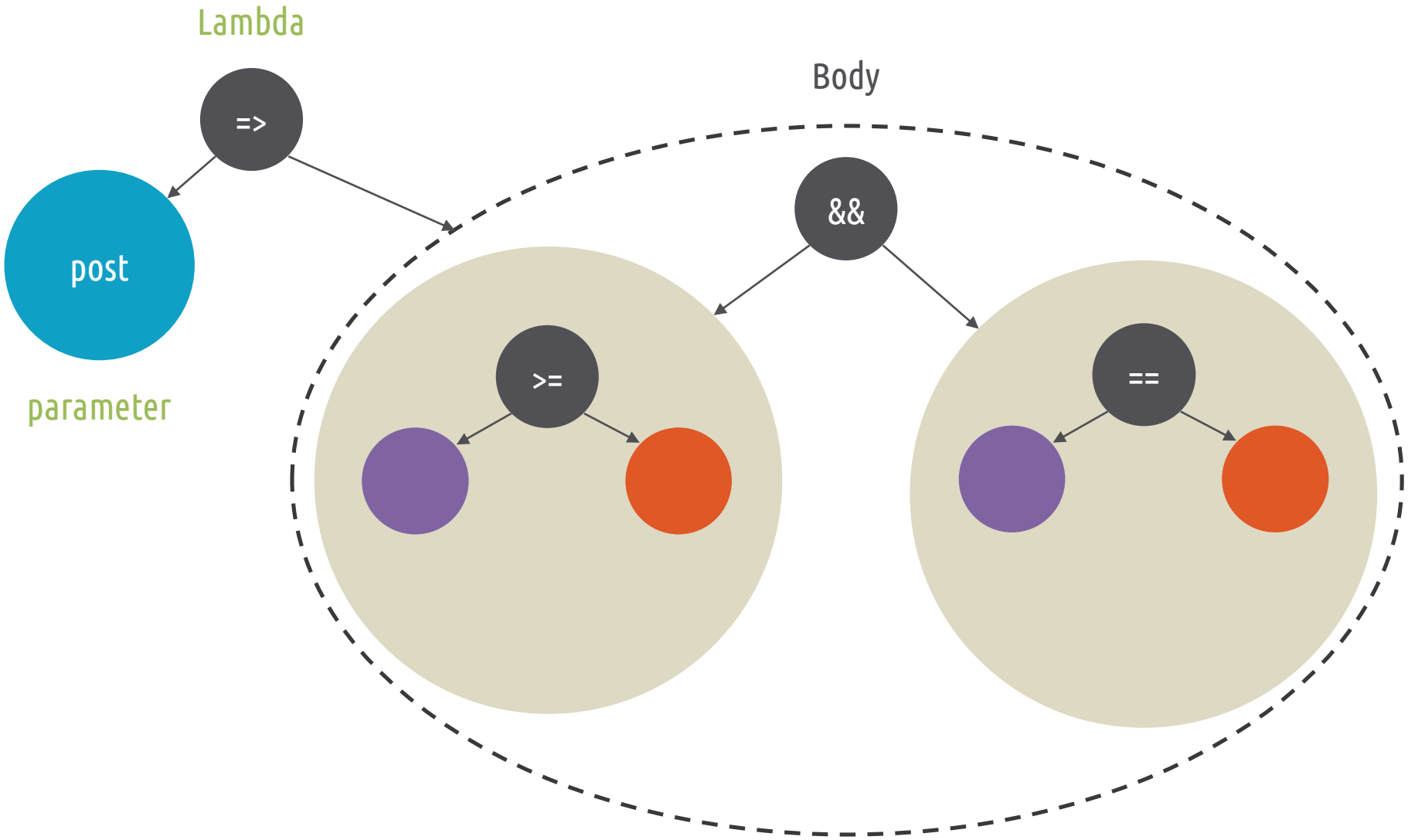
post => post.PostedOn >= cutoffDate



Combine Like Expressions?



Combined Expressions



Readability?

Before

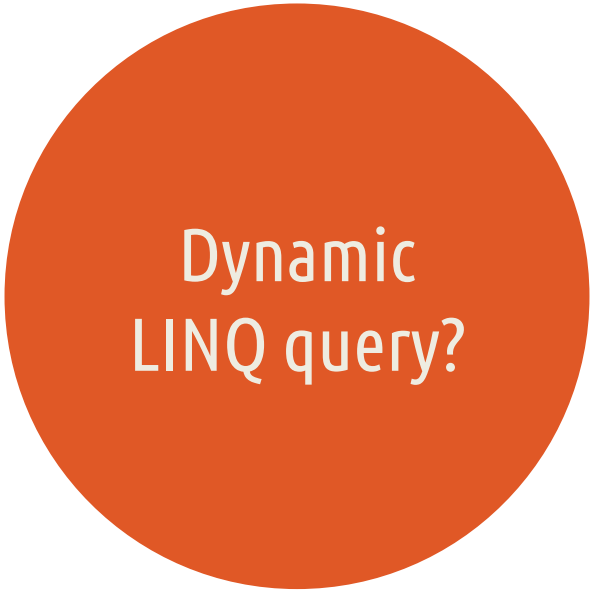
```
1   IQueryable<Post> posts = postRepository.GetAll()
2   .Where(post => post.IsPublished && post.PostedOn <= today &&
3   (post.PostedOn >= cutoffDate || post.Author == featuredAuthor &&
4   post.PostedOn >= featuredAuthorCutoffDate));
5
```

After

```
1   IQueryable<Post> posts = postRepository.GetAll()
2   .ArePublished()
3   .PostedOnOrBefore(today)
4   .Where(PostedOnOrAfter(cutoffDate)
5   .Or(FeaturedAuthorPostedrOnOrAfter(featuredAuthor,featuredAuthorCutoffDate)));
6
```

New Requirements!

Now we must support multiple featured authors



No problem, now we're equipped for it!

Dynamic Queries

```
1   var rule = PredicateExtensions.PredicateExtensions.Begin<Post>();
2   foreach (var authorName in featuredAuthorNames)
3   {
4       rule = rule.Or(FeaturedAuthorPostedOnOrAfter(authorName, featuredAuthorCutoffDate));
5   }
6   return rule;
7
```

Try this without expressions.

Questions?

Resources

- Article
  - https://goo.gl/kAM05P
- GitHub
  - https://github.com/EdCharbeneau/PredicateExtensions
- Nuget
  - https://www.nuget.org/packages?q=predicate+extensions
- Twitter
  - @EdCharbeneau