

Kathleen Dollard
Microsoft

kdollard@microsoft.com
Twitter: @KathleenDollard

Functional Techniques for C#

<https://github.com/KathleenDollard/Slides>

Kathleen Dollard

Microsoft

kdollard@microsoft.com

Twitter: @KathleenDollard

<https://github.com/KathleenDollard/Slides>



You are effective with the imperative, object-oriented core of Java or .NET but you look longingly at the winsome smile of functional languages.

If you play with your language's functional features, you're never quite sure if you're getting it right or taking full advantage of them. This talk is for you.

You'll learn which code to attack with functional ideas and how to do it.

You'll look at code similar to what you write every day, and see it transform from long, difficult-to-follow code to short code that's easy to understand, hard to mess up, and straightforward to debug. Better yet, functional approaches help you apply patterns in a clear and consistent way.

Apply these techniques while leveraging delegates, lambda expressions, base classes and generics.

You are effective with the imperative, object-oriented core of Java or .NET but you look longingly at the **winsome** smile of functional languages.

If you play with your language's functional features, you're never quite sure if you're getting it right or taking full advantage of them. This talk is for you.

You'll learn which code to attack with functional ideas and how to do it.

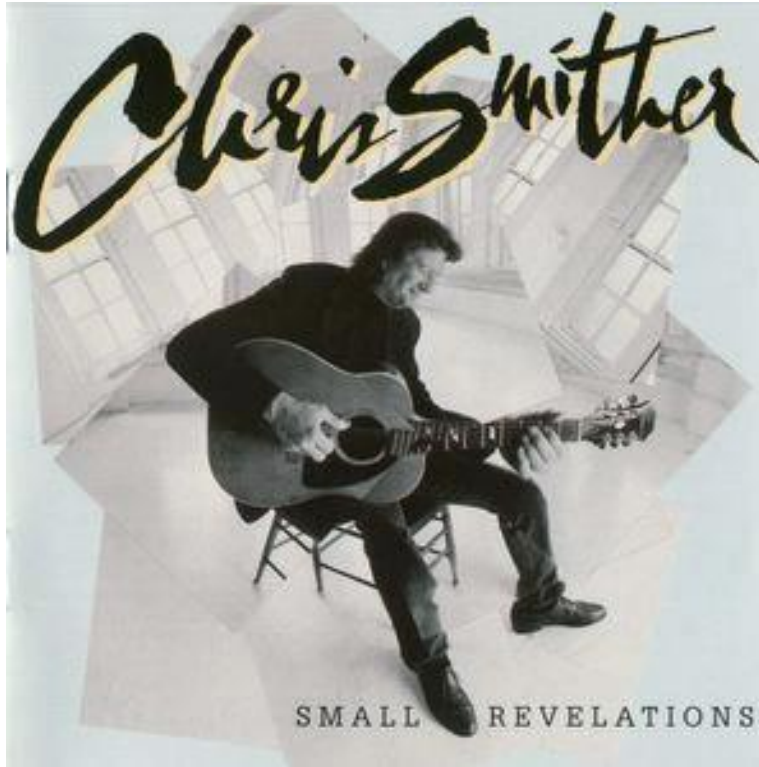
You'll look at code similar to what you write every day, and see it transform from long, difficult-to-follow code to short code that's easy to understand, hard to mess up, and straightforward to debug. Better yet, functional approaches help you apply patterns in a clear and consistent way.

Apply these techniques while leveraging delegates, lambda expressions, base classes and generics.

Winsome Smile

Chris Smither

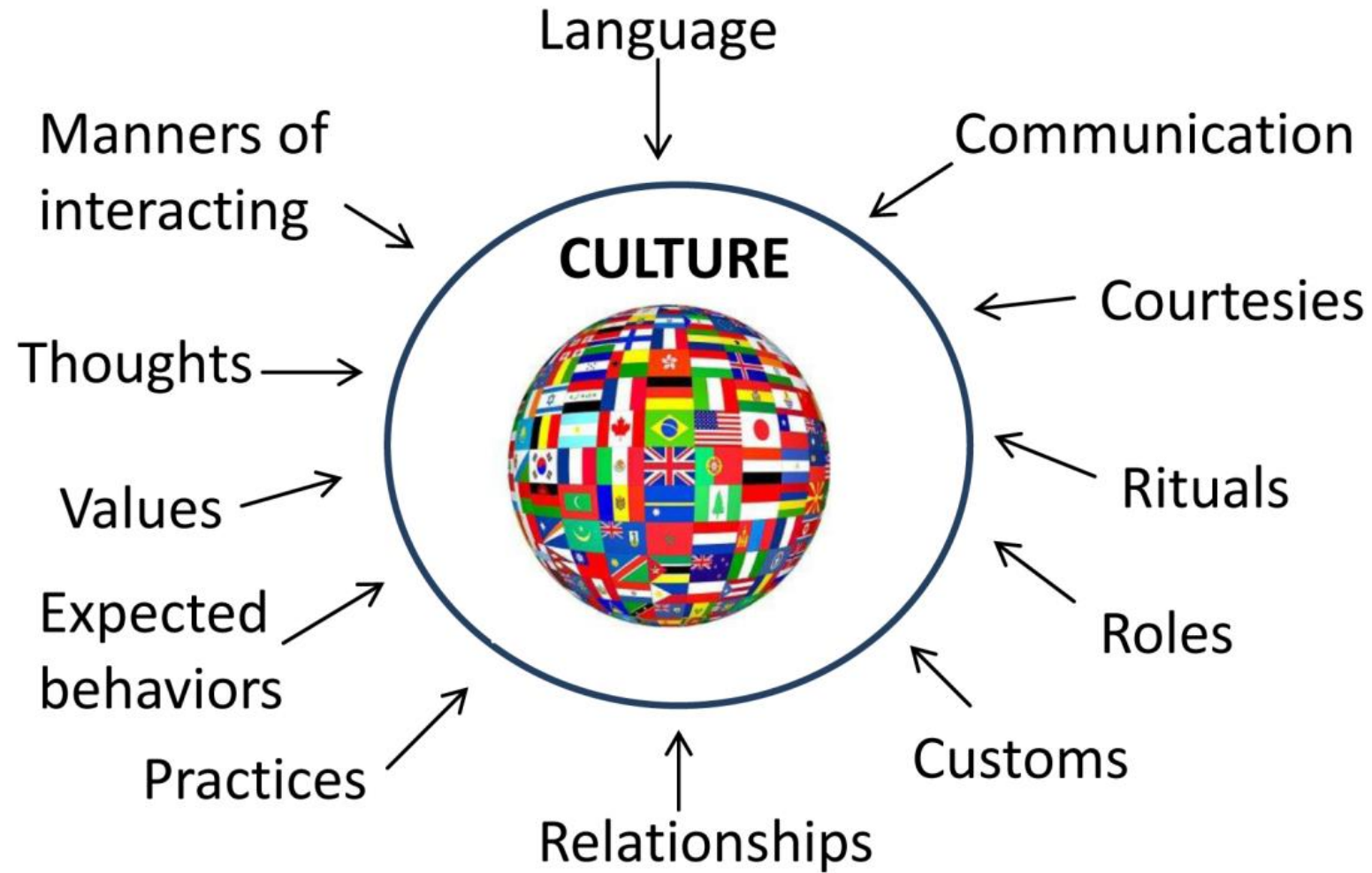
Small Revelations album



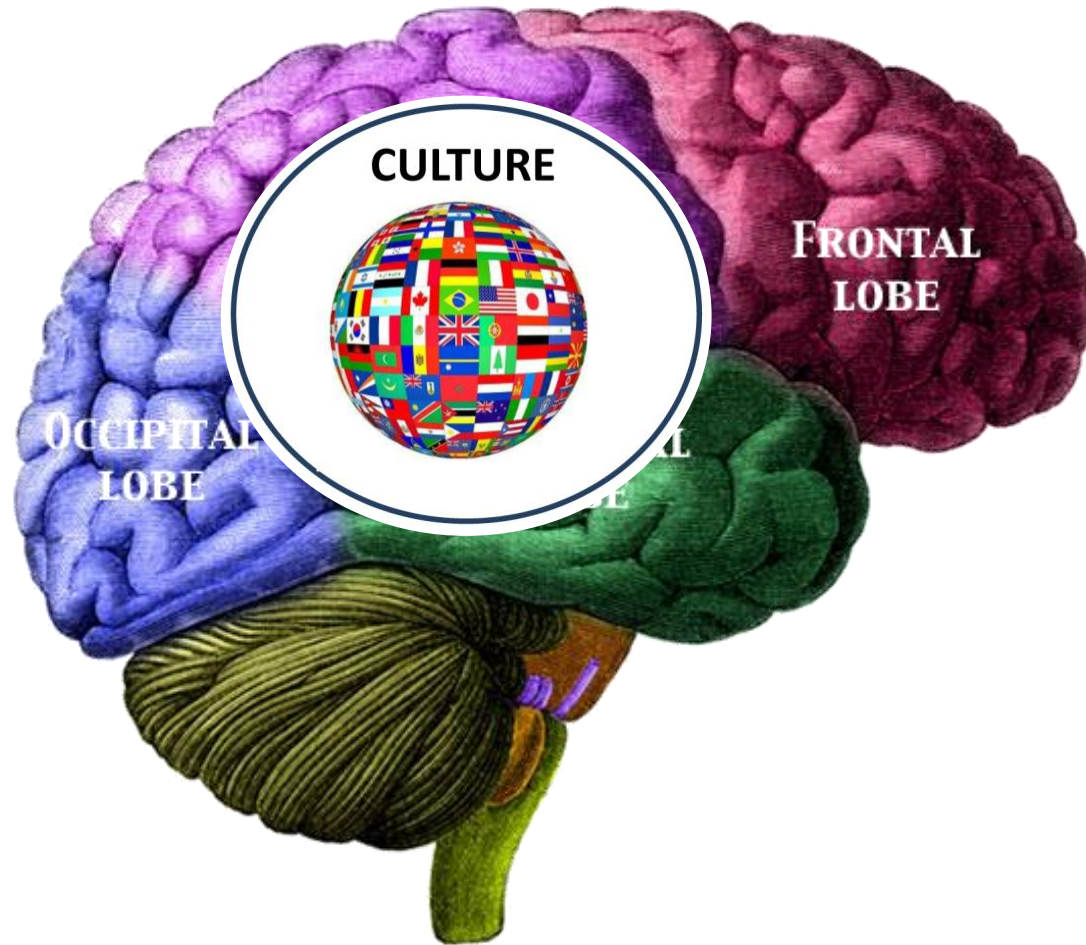
Winsome –

“attractive or appealing in appearance or character.”

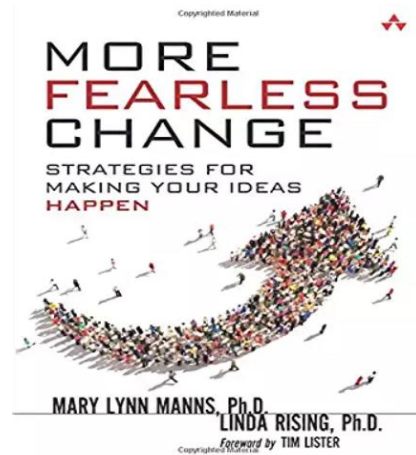
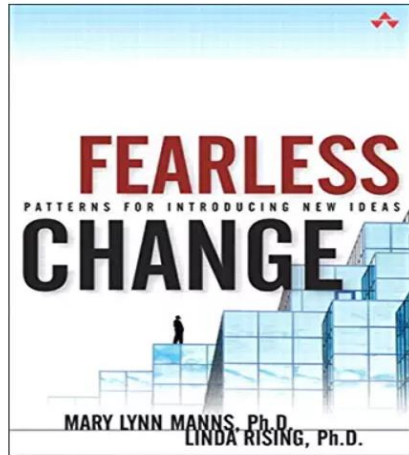
- *Defined by Google*



Culture and language have a great impact on how you think



Linda Rising



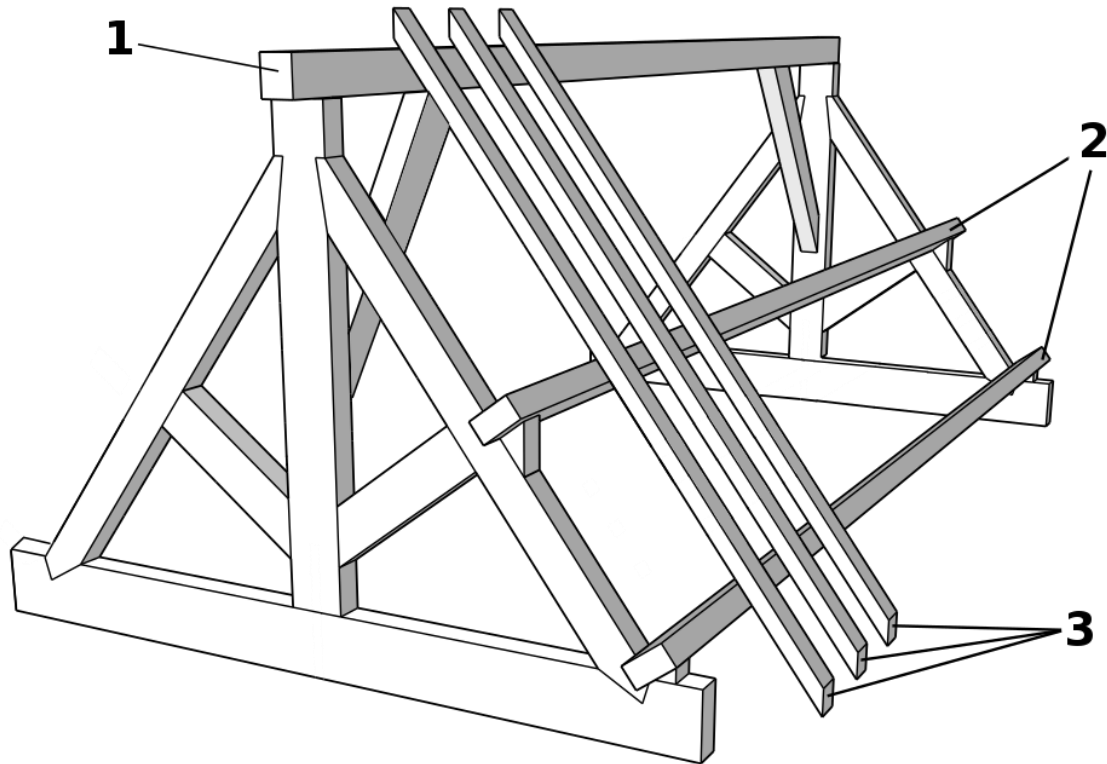
[Fearless Change: Patterns for Introducing New Ideas](#) 4 Oct 2004
by Rising, Linda, Ph.D. and Manns, Mary Lynn, Ph.D.

[More Fearless Change: Strategies for Making Your Ideas Happen](#) 6 Mar 2015
by Mary Lynn Manns and Linda Rising

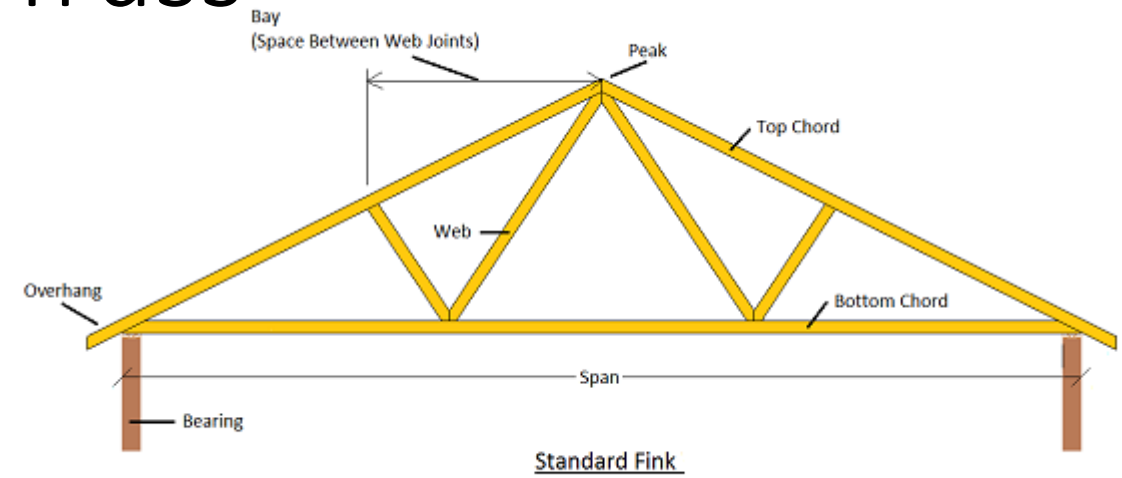
- You, your team, and your organization are complex adaptive systems
- Complex adaptive systems cannot be changed by large forces, at least not effectively
- They change because of small steps – small trials and adjustments

Technology drives fashion

Rafters



Truss



Fashion – Sensible rafter design



Fashion – And when you use trusses





Etsy



Technology drives fashion

- For the stupid
- For the good

Pockets, not truss roofs

- We have a desire for functional
- Functional includes cultural change
- Fashion can add real value, or not



Explore and experiment

What is a Functional Language?

Functional Language

- Central construct is a function
- Functions are first class citizens

Object Oriented Language

- Central concept is a class
- Functions may be (or may not be) first class citizens

Bigger distinctions

- Dynamic vs Strong/static typing
 - JavaScript vs Haskell and C#
- Compiler intensity (policing)
 - JavaScript vs Haskell and C#
- Compiled vs. interpreted
 - C# vs Visual Basic for Applications (VBA)
- Support for REPL
 - PowerShell or F# (C# Interactive)

Why functional in C# (an OO world)?

- Testability ➤ Purity
- Parallelism ➤ Immutability
- Reuse ➤ Inheritance, helper classes
- Expressiveness ➤ Less smelly
- Reasonableness ➤ Craftsmanship (naming, SRP, etc)

***Functional techniques
allow us to up the game in all these areas***

Why C# with Functional

- Lots of usage (your team might be using it)
- Best of strong typing to reduce accidents
 - If you think that's noise, use inference and implicit operators
- Generics to reuse types
- Extension methods to extend types
- Functions first class citizens (strongly typed delegates)
- Expressions trees: a structure to describe delegate contents
- Keep the best of this, add more...

Purity

- No surprises!
 - Should indicate all possible input/output
 - Same input should *always* result in same output

Demo!

Purity

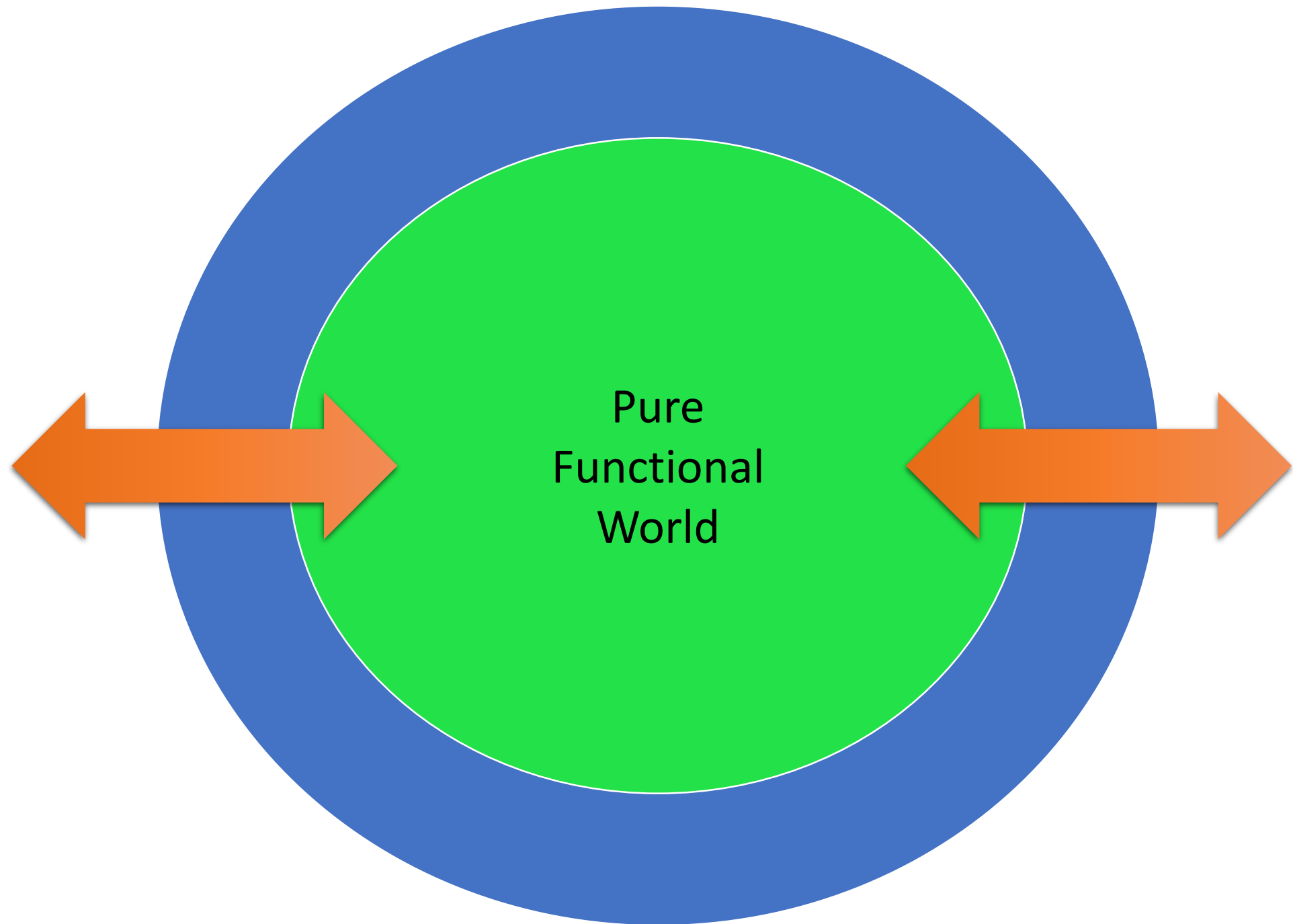
Purity

- No surprises!
 - Should indicate all possible input/output
 - Same input should *always* result in same output
 - Control flow should be entirely predictable
 - Careful planning for exception
 - Void methods (except those doing absolutely nothing) are not pure
- Pure code is easy to test
 - Be clear within your project what “the world can’t change” means

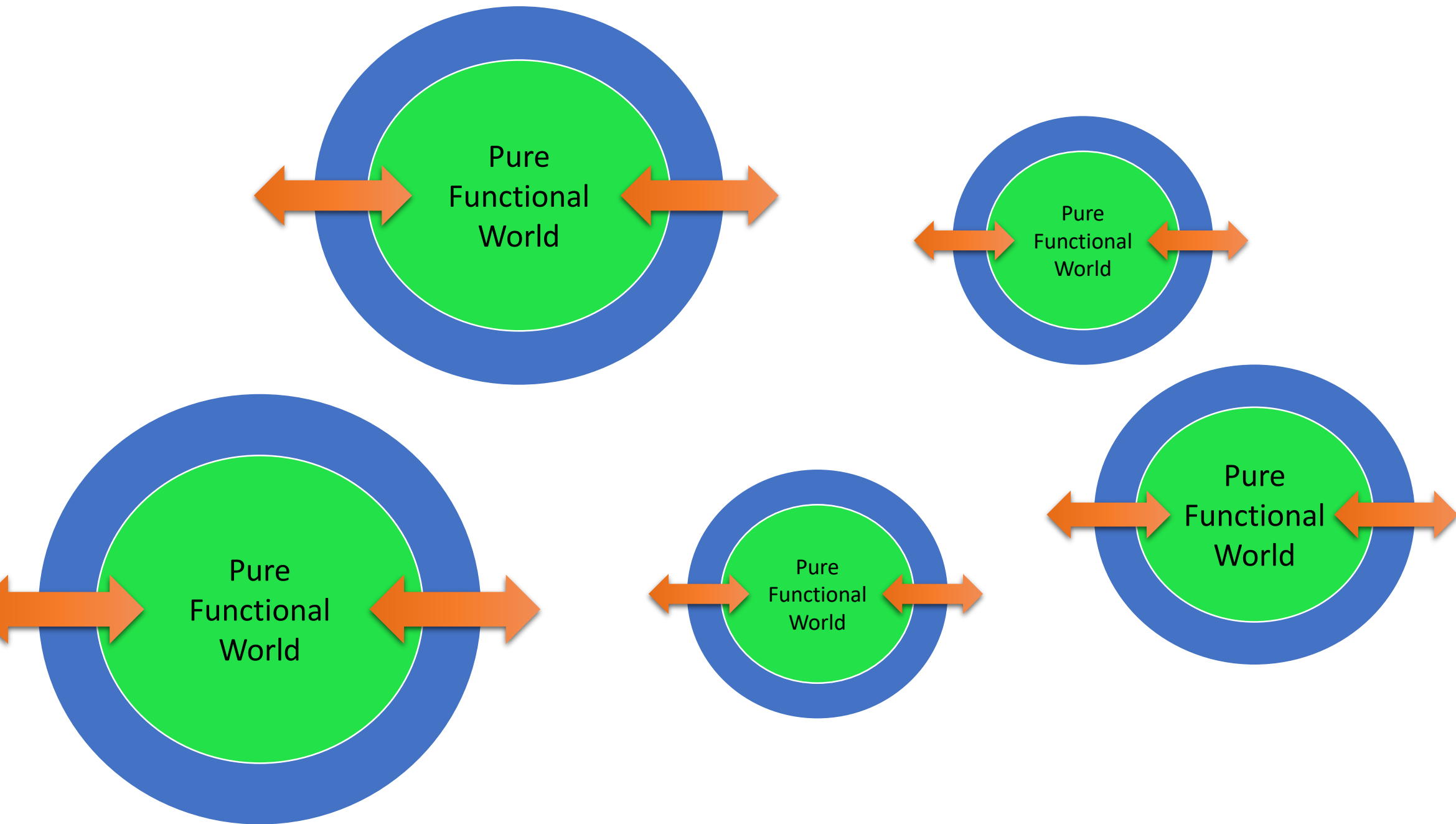
Purity is rather boring

Your app might look like...





Separate pure and not pure code

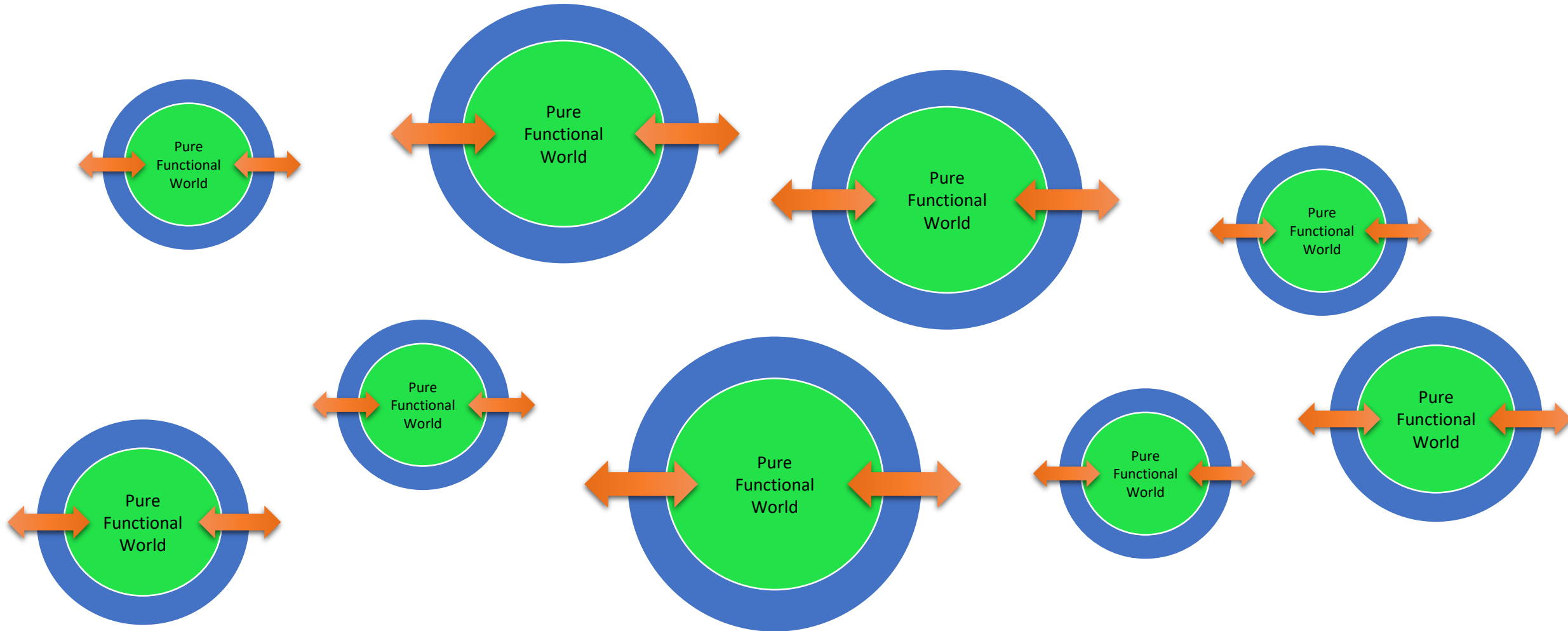


C#



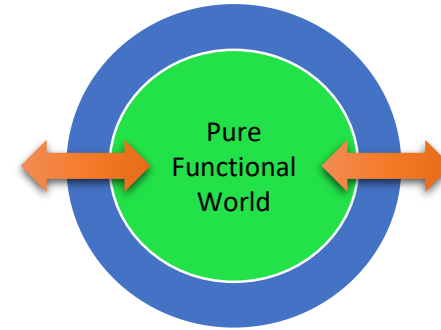
Functional
techniques

C# is statically typed and allows multiple internal functional islands



The ability to test is a measure of architectural sanity

- Unit tests within pure units
- Automated functional tests between units
- Don't mingle pure and impure code
- Don't mingle unit and functional tests



Confusing, since we refer to our automated test tools as unit test tools

Separate pure and not pure code

Separate pure and not pure code

Separate pure and not pure code

C# 7 and functional constructs (opinion)

First class functions		
Purity		
Immutability		
OOP		
Strong typing		
Generics		
Pattern matching		
Expression Trees		
Duck typing		
Records		

C# 7 and functional constructs (opinion)

First class functions		A-
Purity		D
Immutability		B-
OOP		A
Strong typing		A
Generics		A
Pattern matching		C
Expression Trees		A
Duck typing		F
Records		F

Functions as first-class citizens

- Define functions (like data)
- Pass functions around (like data)
- Support higher order functions
 - Functions with delegate parameters or return delegates
- In C# (and Visual Basic) this means Delegates

Delegates – functions as data

- Generic delegate types (Action, Func)
- Type safe function pointers
 - System.Delegate and inherited types
 - “Named” in docs
 - Anonymous methods
 - delegate()
 - Reference to a method (name without parens)
 - Can be a local method
- Lambdas

Delegates – functions as data

- Generic delegate types (Action, Func)
- Type safe function pointers
 - ~~• System.Delegate and inherited types~~
 - ~~• “Named” in docs~~
 - ~~• Anonymous methods~~
 - ~~• delegate()~~
 - Reference to a method (name without parens)
 - Can be a local method
 - Lambdas

f

Delegate

Lambda

func

Are the same in today's context

- Delegates are code fragments that can be stored to execute later
- **Func<T>**
 - **Func<T<T1<T2>>>**
- **Func<T1, T2>**
 - **Func<TParam, T<T1<T2>>>**
 - **Func<int, Task<DataResult<List<Student>>>>**
- **=>**
 - **Func<int, int> f1 = x => x + 2;**
 - **Func<int> f2 = () => 42;**
 - **Func<int, int, int> f3 = (x, y) => x + y;**
- **...Where<T>(Func<T, bool> predicate)**
 - **var y = list.Where(z => z.Id == x);**

LINQ

- Select, Where, OrderBy etc. are higher order functions
- They are pure because they return a new list
- Lambdas are not free, especially with closures

```
var x = 42;
```

```
var y = list.Where(z => z.Id == x);
```

- In memory loops are faster in high performance code (like .NET framework)
 - For your code, they are almost certainly close enough to free
- Expression trees contain code definition
 - Can be understood in different languages
 - Like TSQL

- Action: Delegates that don't return a value
 - **Action<T>**
 - **Action<T<T1<T2>>>**
 - Not pure (unless trivial)
 - Doesn't interchange with Func (not polymorphic)
 - ...RecordTime<T>(Func<T> op)
 - ...RecordTime<T>(Func<T, T2> op)
 - ...
 - ...RecordTime<T>(Action<T> op)
 - ...RecordTime<T>(Action<T, T2> op)
- Arity**

Can convert delegate types to each other

```
public static class ActionExt
{
    public static Func<VoidType> ToFunc(this Action action)
        => () => { action(); return VoidData; };

    public static Func<T, VoidType> ToFunc<T>(this Action<T> action)
        => x => { action(x); return VoidData; };

    public static Func<T1, T2, VoidType> ToFunc<T1, T2>(
        this Action<T1, T2> action)
        => (T1 x, T2 y) => { action(x, y); return VoidData; };
}
```

- Action: Delegates that don't return a value

- **Action<T>**

- **Action<T<T1<T2>>>**

- Not pure (unless trivial)

- Doesn't interchange with Func (not polymorphic)

- ...RecordTime<T>(Func<T> op)

- ...RecordTime<T>(Func<T> op)

Or just generally avoid Action

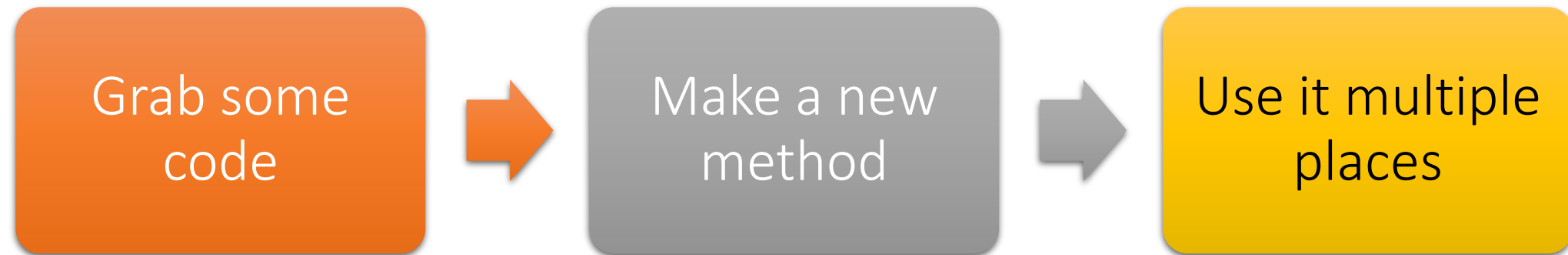
- ...RecordTime<T>(Action<T> op)

- ...RecordTime<T>(Action<T, T2> op)

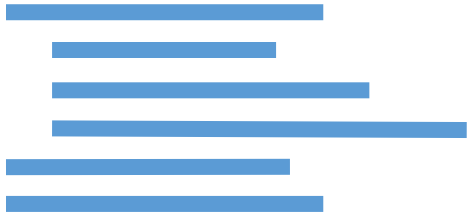
Refactoring to Functional

Imperative (normal) Refactoring

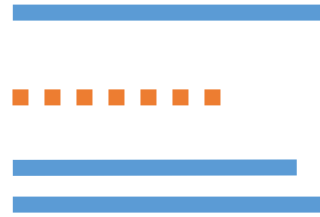
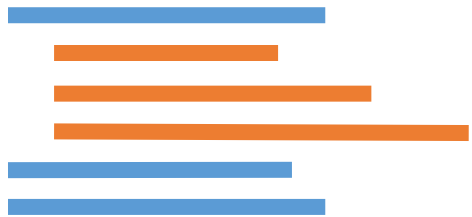
- Inside out refactoring



Imperative Refactoring



Imperative Refactoring

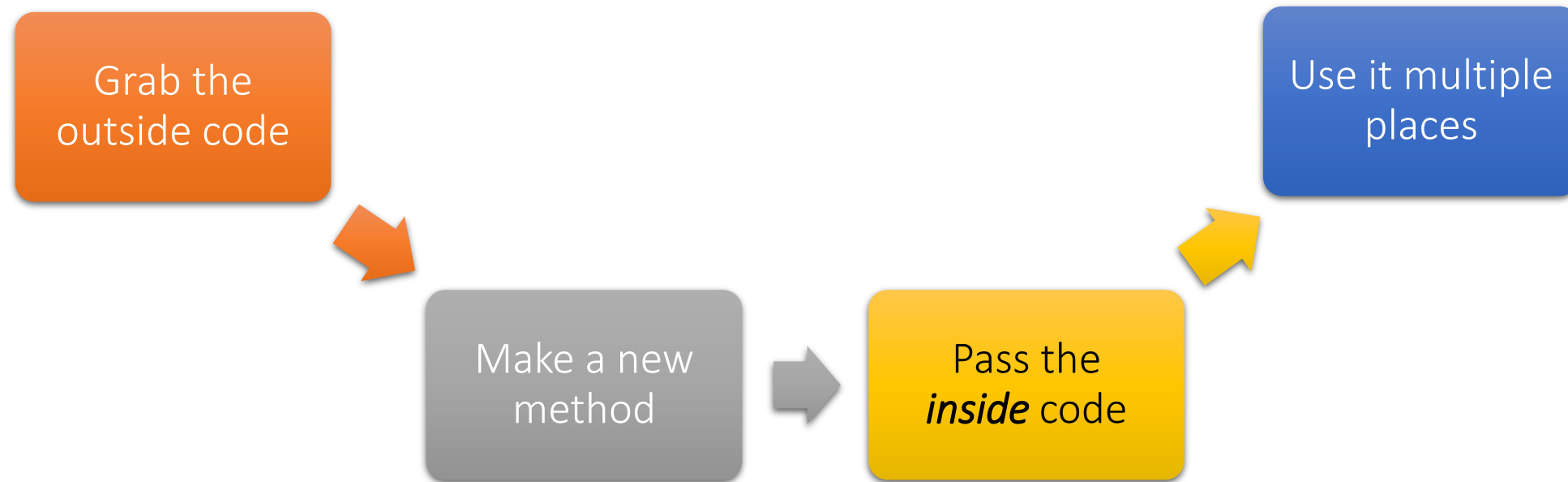


Demo!

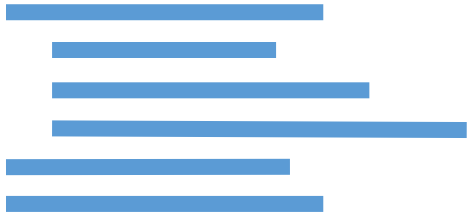
Inside out refactoring (normal)

Functional Refactoring

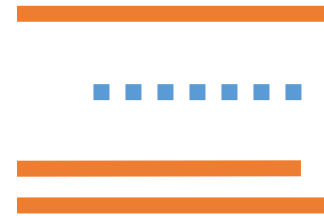
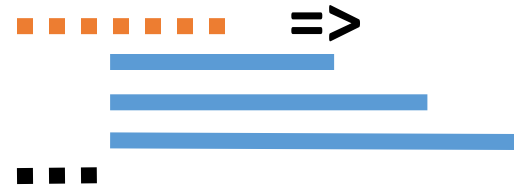
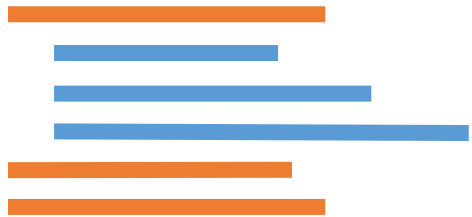
- Outside in refactoring



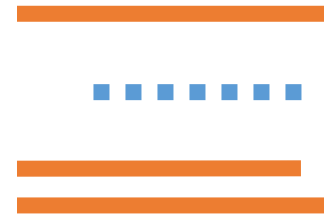
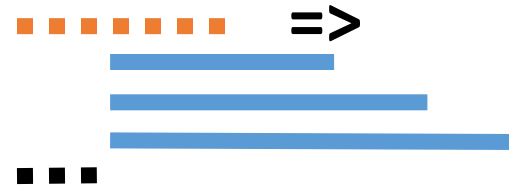
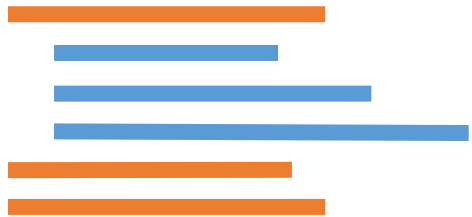
Functional Refactoring



Functional Refactoring



Functional Refactoring



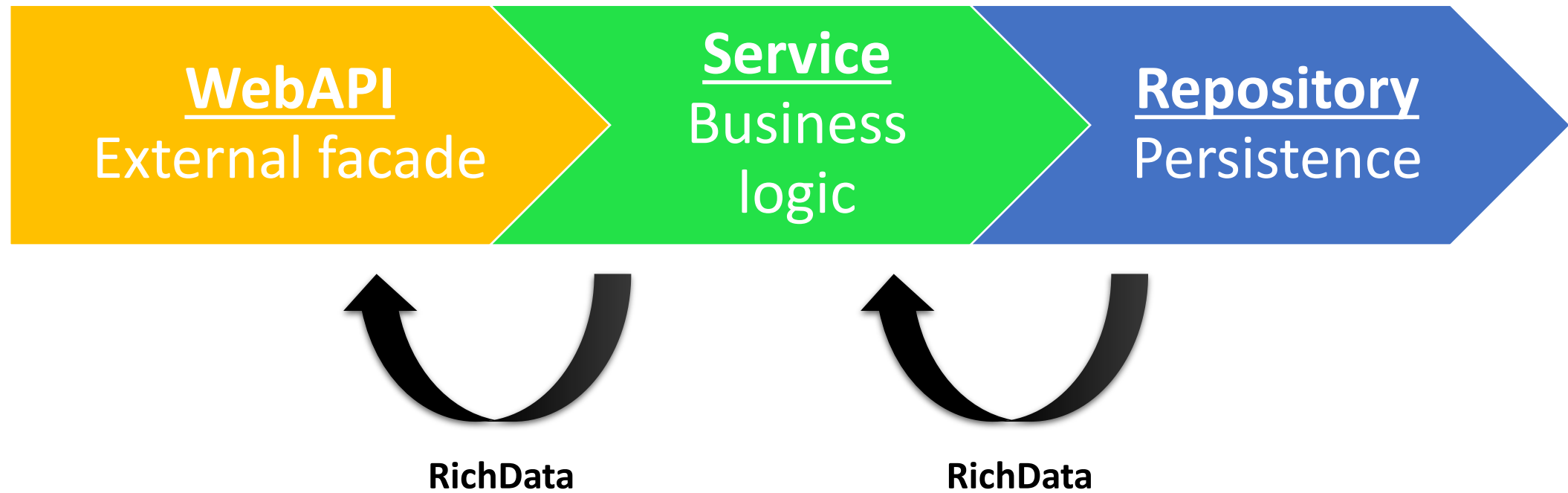
Demo!

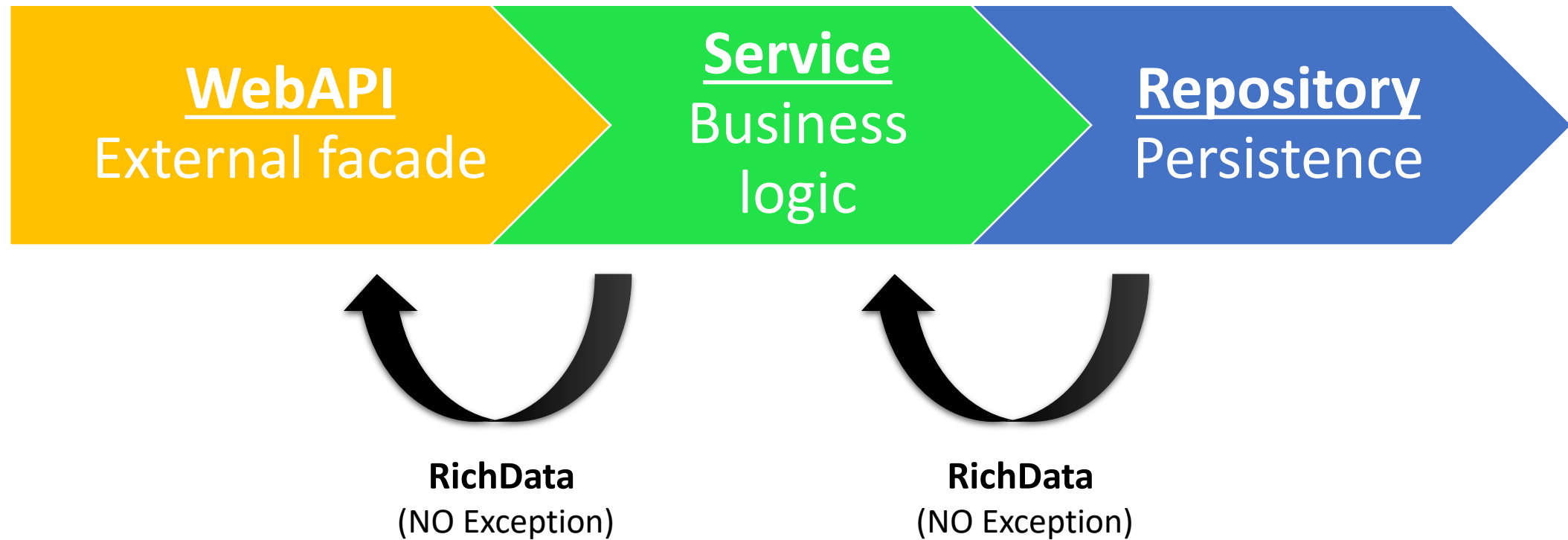
Outside in refactoring

Handling Errors

What could possibly go wrong?

- Protocol failure like routing (one, not seen by app)
- Unpacking behavior like bad JSON format (one, seen by infrastructure)
- Validation like string too long (many)
- Anticipated environment issues like database missing (one, ☞ ops)
- Unexpected app failure like null reference (one, ☞ programmers)
- Batch process, last 3 above for each item
 - All succeed
 - Some succeed
 - None succeed





Naming is hard



1. Either
2. RichData/RichValue
3. Result
4. Try
5. Validation<Exception<T>>

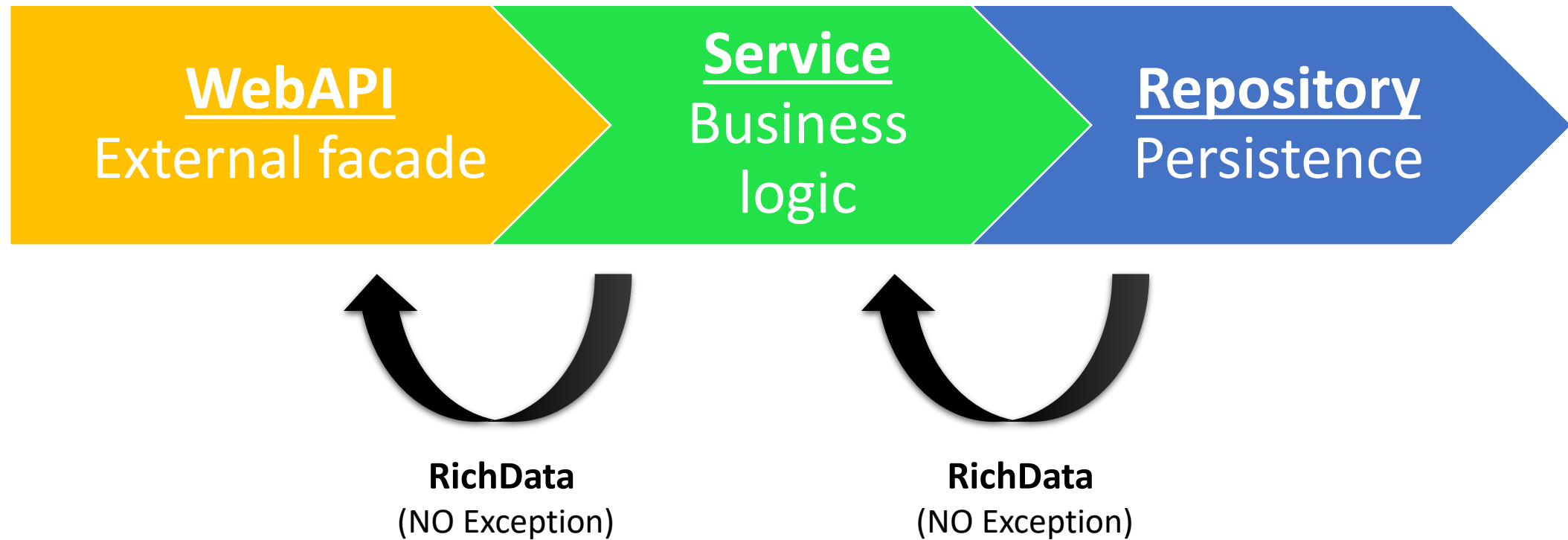
1) Core FP concept, 2 & 3) Kathleen 4) Sander (if I understand his abstract) 5) Enrico

Naming is hard



1. Either
2. **RichData/RichValue**
3. Result
4. Try
5. Validation<Exception<T>>

1) Core FP concept, 2 & 3) Kathleen 4) Sander (if I understand his abstract) 5) Enrico



Your app might look like...



Your app might look like...

This talk seems to
be lacking
something

**What?
Cats?
Cloud?
GIFs?**



Your app might look like...



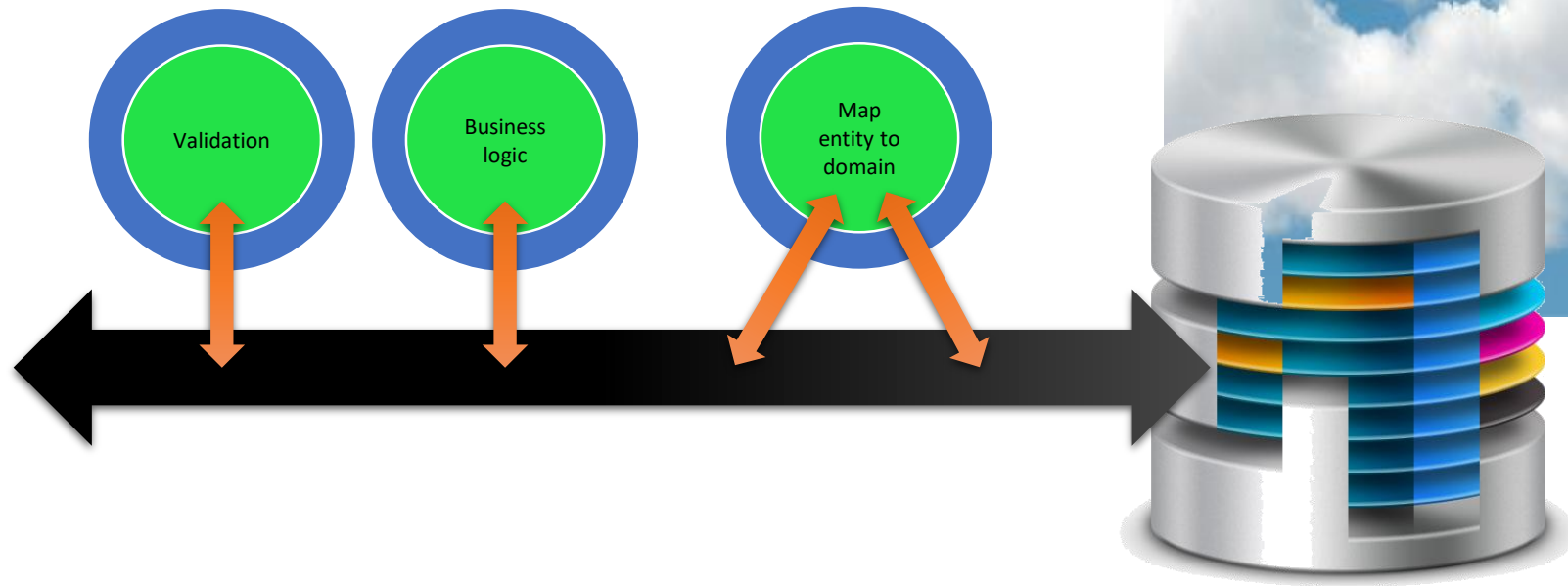
[This Photo](#) by Unknown Author is licensed under [CC BY](#)



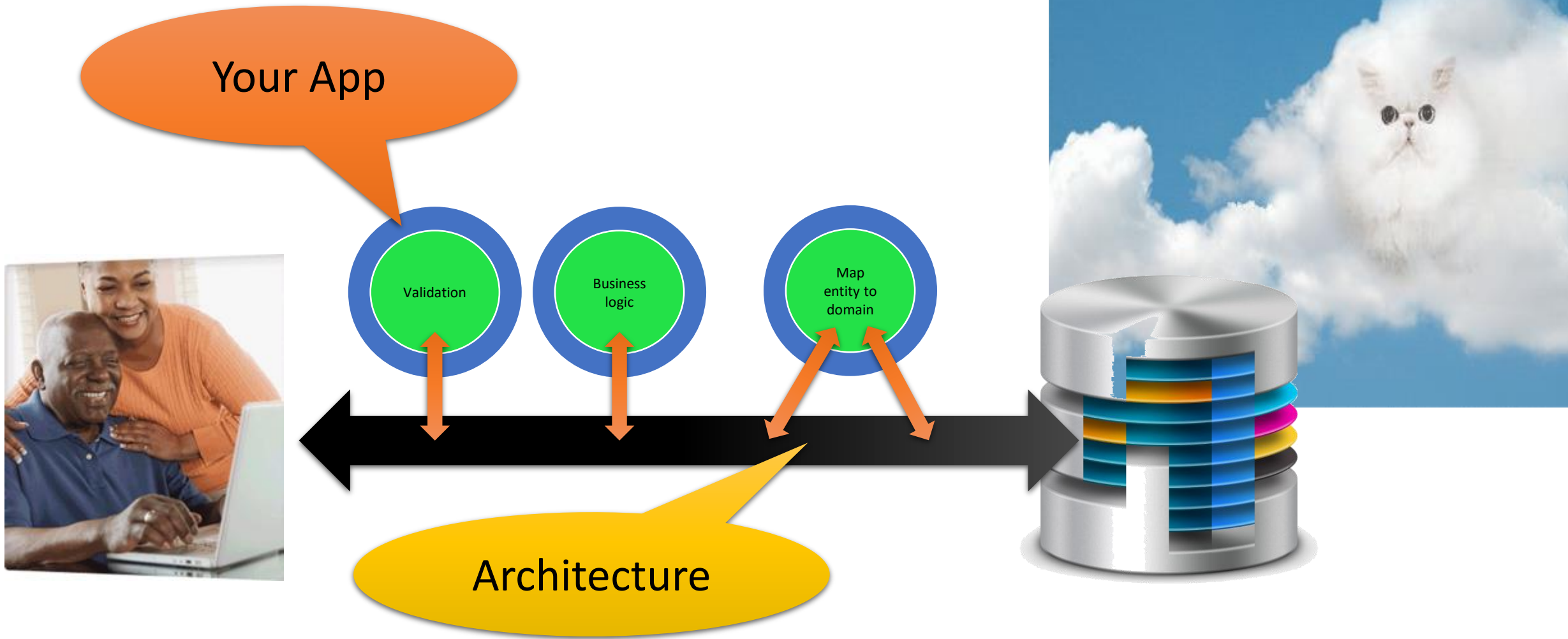
[This Photo](#) by Unknown Author is licensed under [CC BY](#)
[This Photo](#) by Unknown Author is licensed under [CC BY](#)

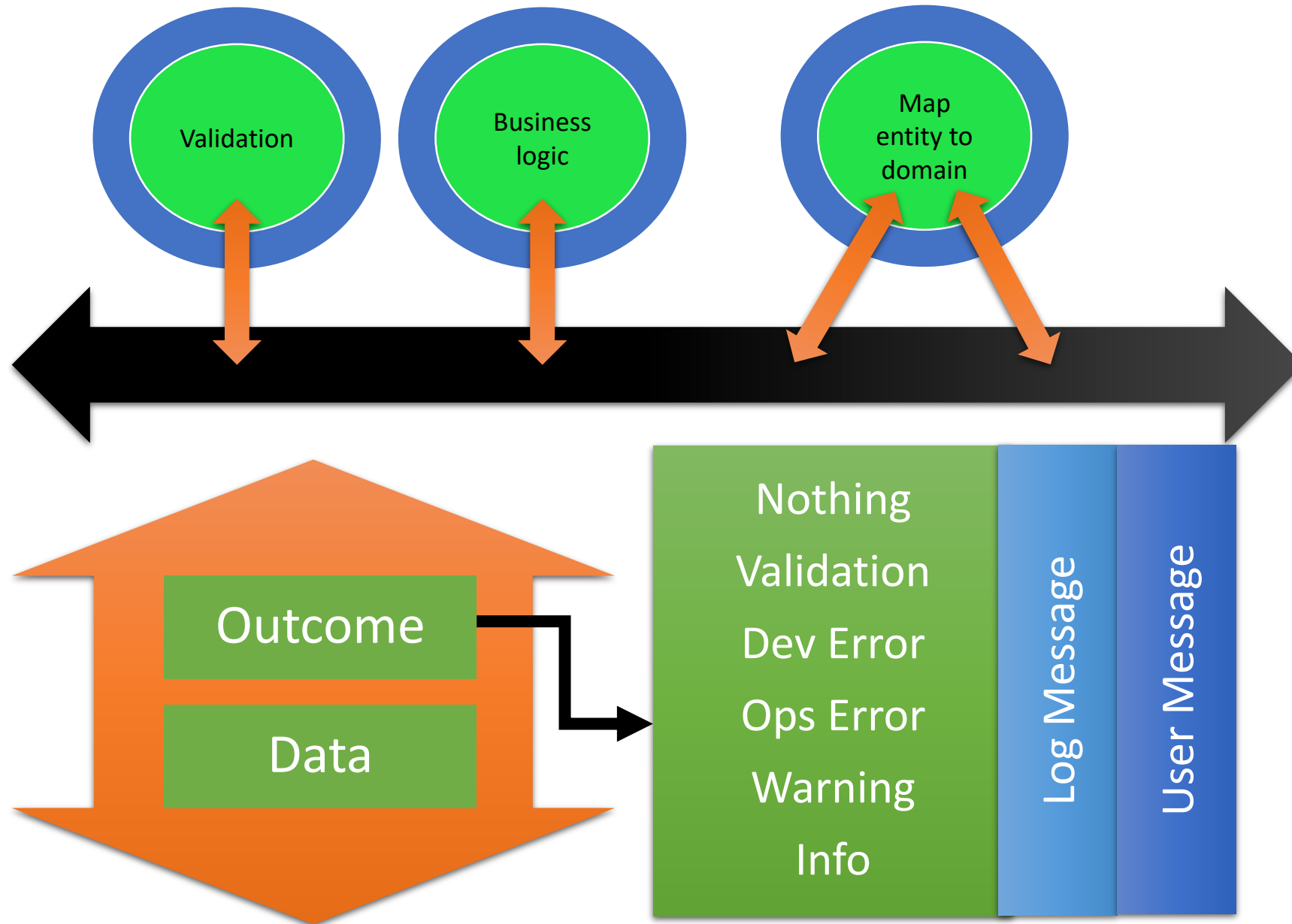


Your app might look like...



Your app might look like...





Demo!

Outcome

RichData<TData>

Useful things in C#

- Expression-body members
 - Generic inheritance hierarchies
 - Expression trees
 - Local functions
 - Pattern matching (enhanced switch statement)
 - Tuples
 - Throw expression
-
- Pattern matching (switch expression)
 - Default interface implementation (rich interfaces)

Expression-body members

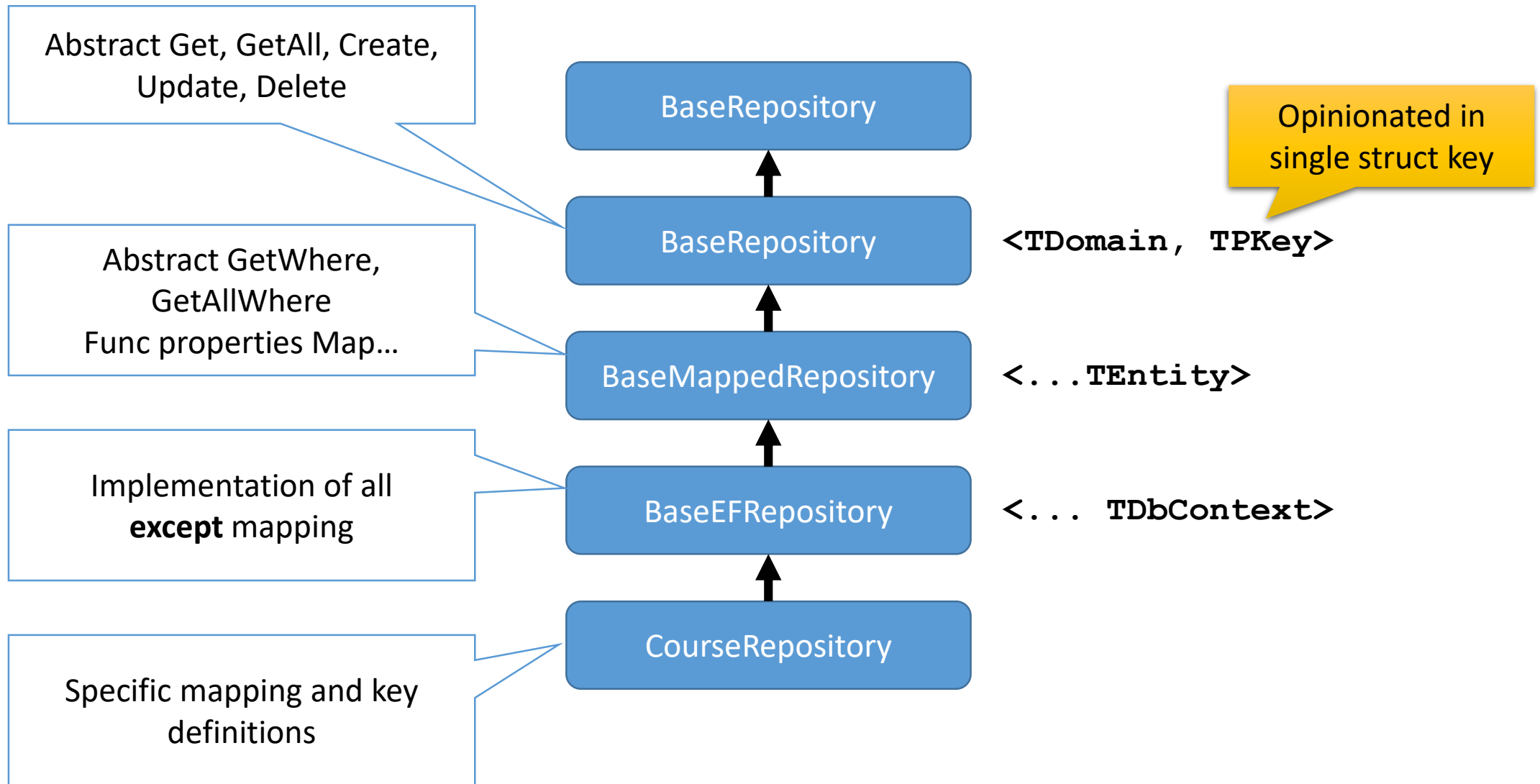
```
public (IEnumerable<string> messages,  
        int staffCount,  
        int) GetThankYouMessages(IEnumerable<Person> persons)  
=>  
(  
    persons.Select(x => GetThankYouMessage(x)),  
    persons.Where(x => x is Staff).Count(),  
    42  
);
```

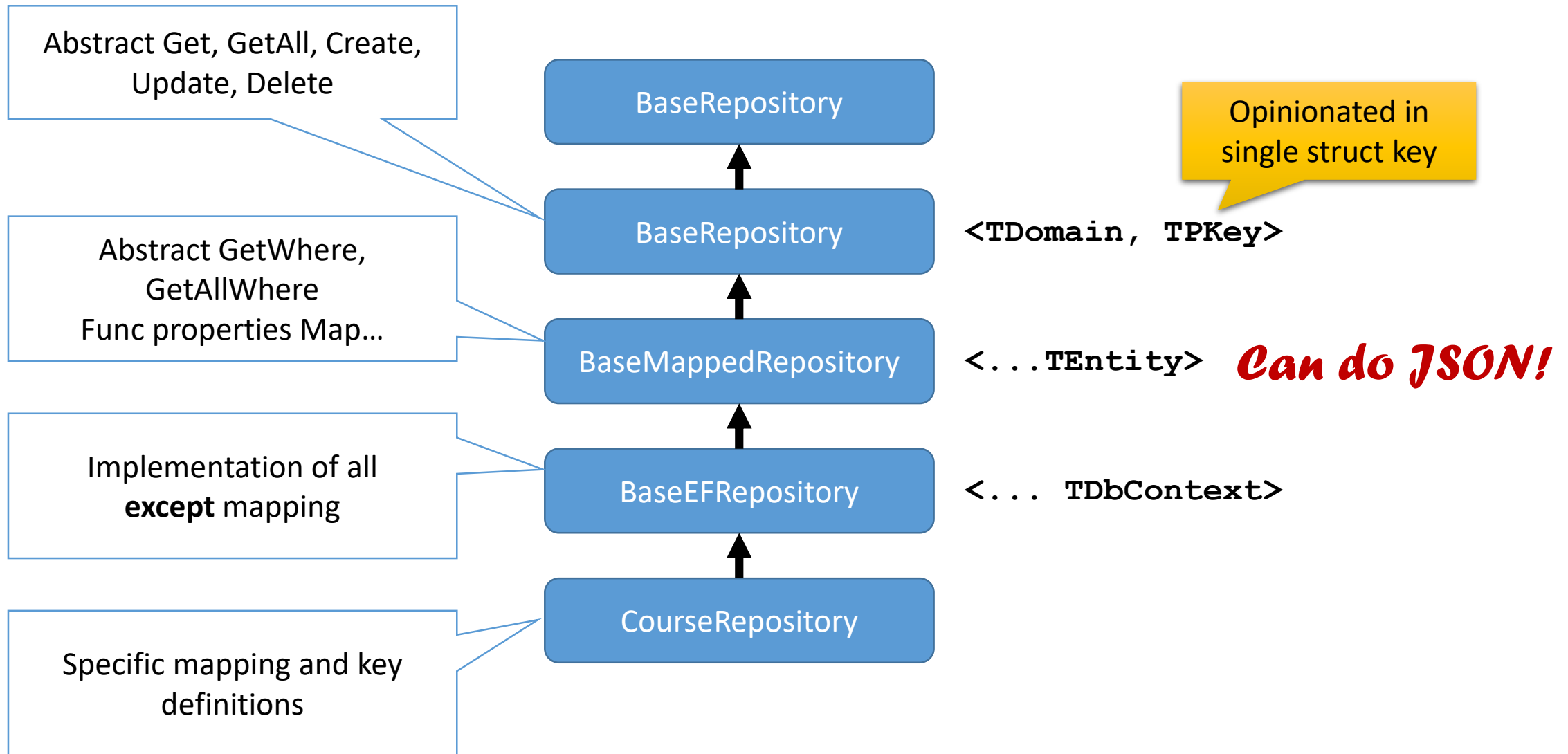
We need to be communicate intended purity

\Rightarrow Use expression body members
only when code is pure

Generic inheritance hierarchies from a *partial application* perspective

- Base class has no generic type
- Leaf class has all types
- Each intervening class has a purpose and adds generic types





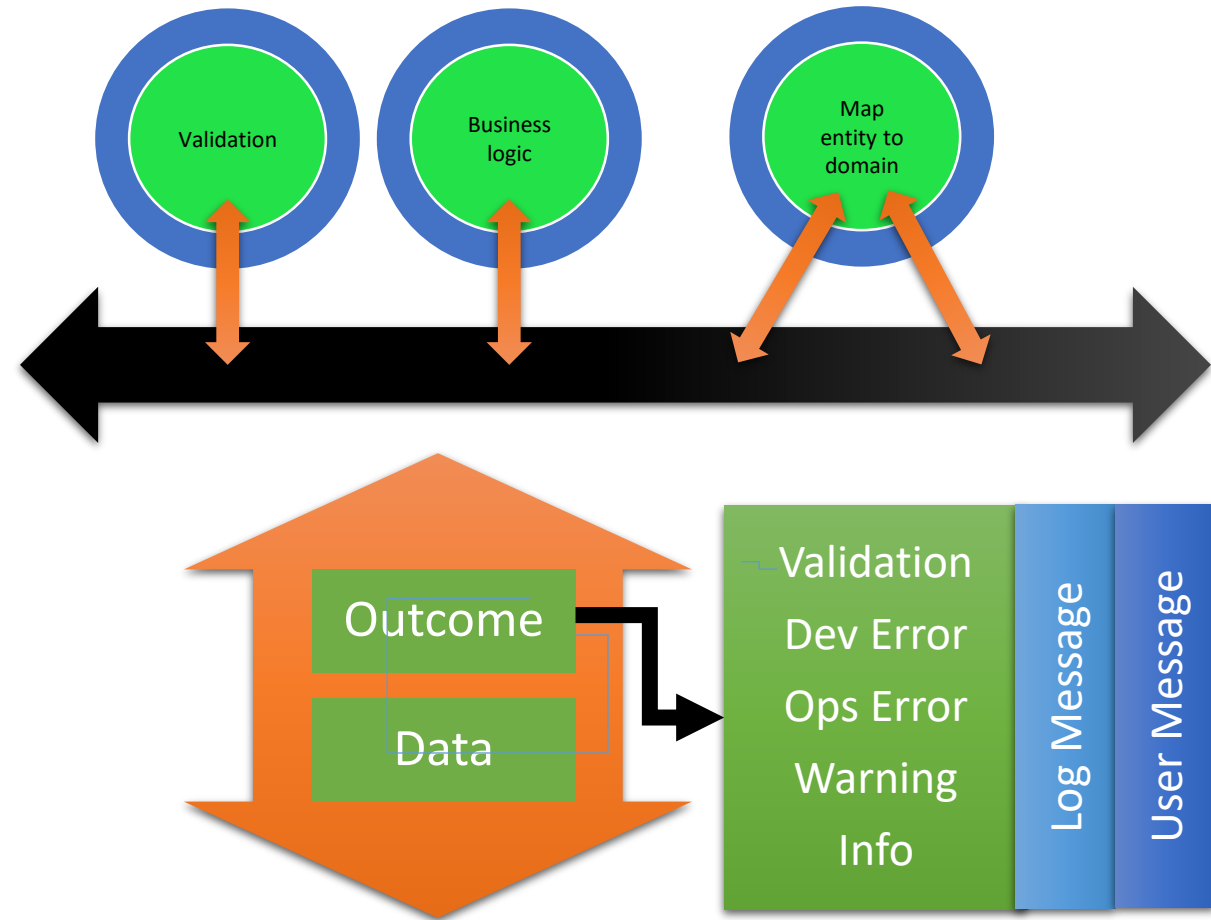
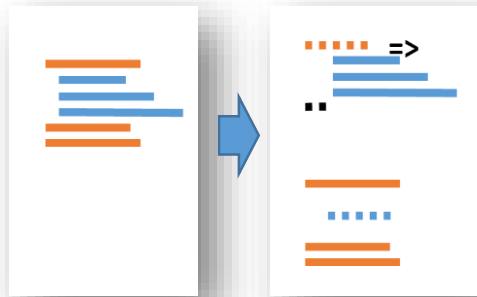
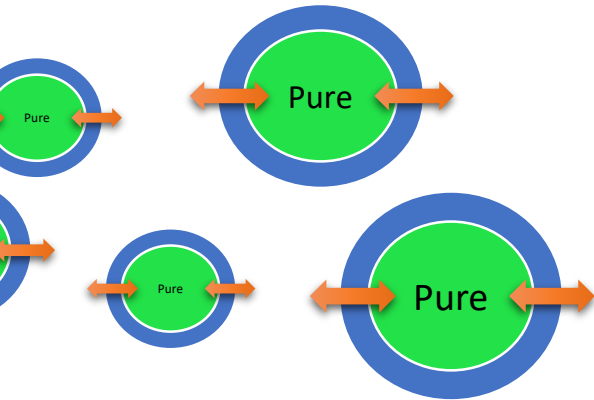
Recap

- We have a desire for functional
- Functional includes cultural change
- Fashion can add real value, or not

} Explore and experiment

C# Functional techniques

- Pattern matching
- Generic hierarchies
- And much more



Questions?

Functional Techniques for C#

@kathleendollard
kdollard@gmail.com

References

- Today's code: <https://github.com/KathleenDollard/Slides>
- Functional Programming in C#: How to write better C# code
 - Enrico Buonanna
 - Manning, 2017
- Pluralsight : *Applying Functional Principles in C#*, Vladimir Khorikov
- Pluralsight : *Functional Programming with C#*, Dave Fancher
- Review of Bacus's paper: <https://medium.com/luteceo-software-chemistry>