Functional Techniques for C#

Kathleen Dollard
Microsoft

kdollard@microsoft.com
Twitter: @KathleenDollard

https://github.com/KathleenDollard/Slides

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
- Parallelism
- Reuse
- Expressiveness
- Reasonableness

- ➢Purity
- ➢Immutability
- ➢Inheritance, helper classes
- ➢Less smelly
- ➢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!

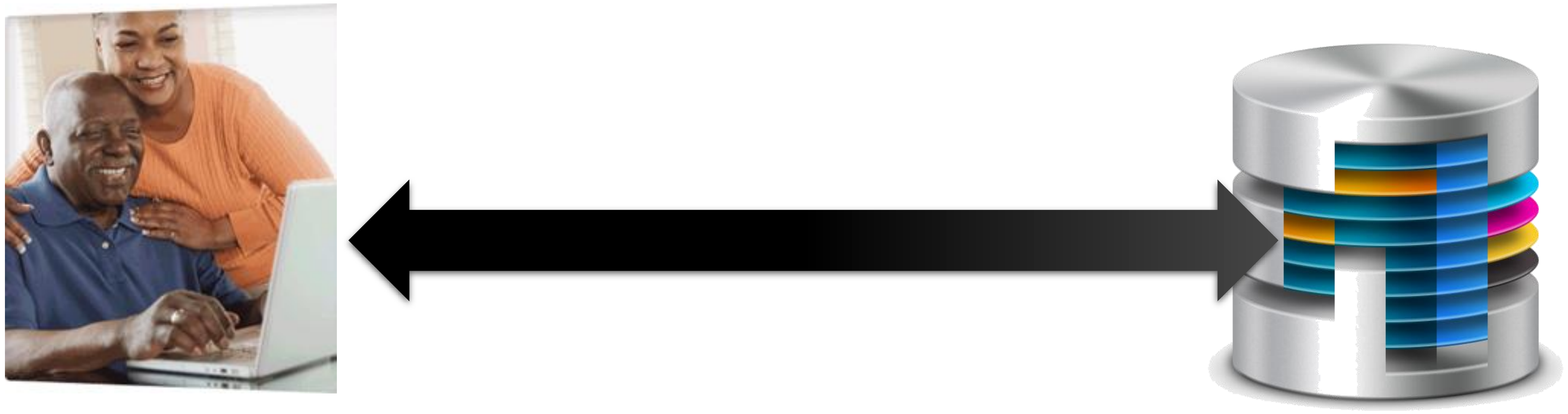
Purity

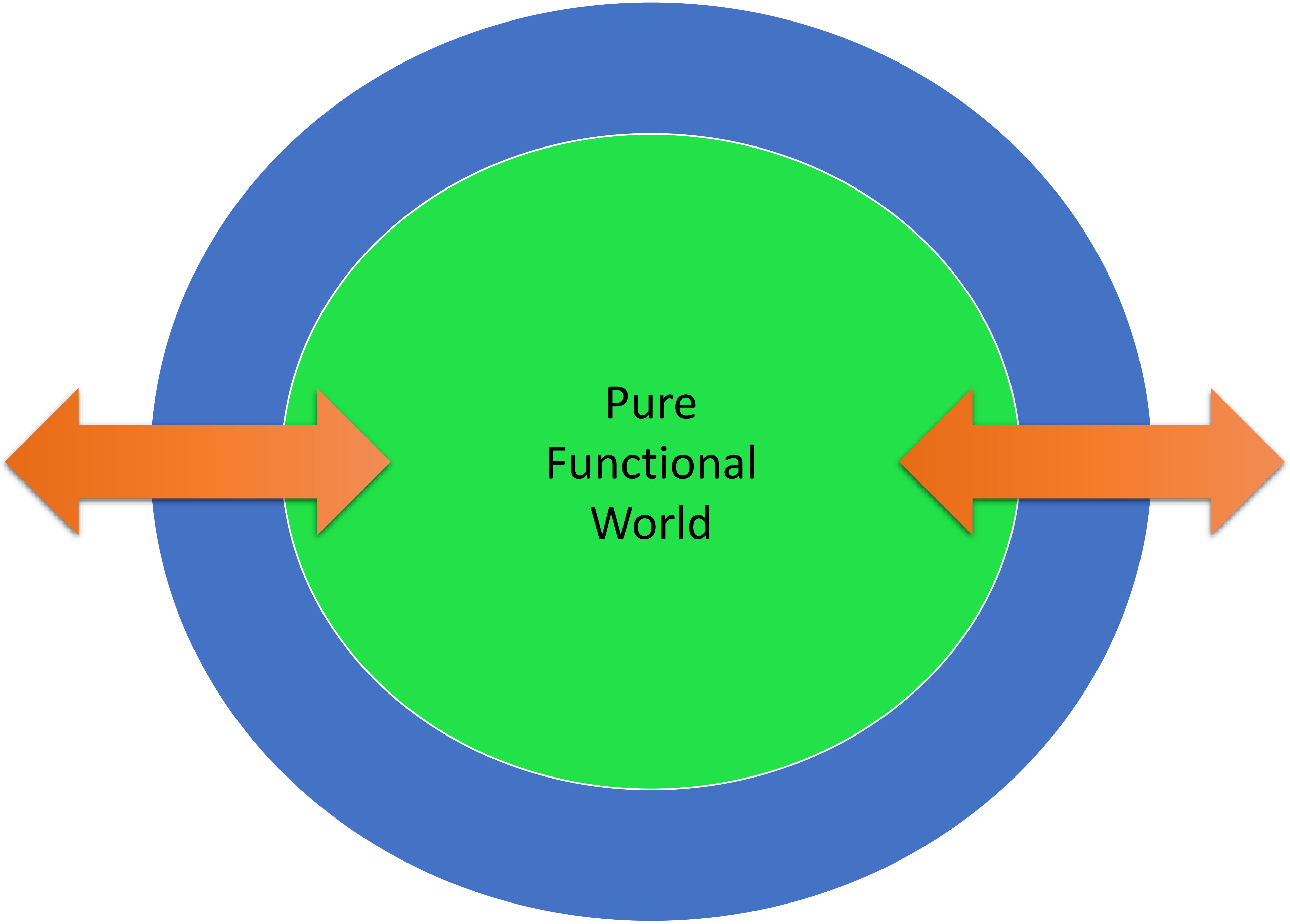
Purity

- 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

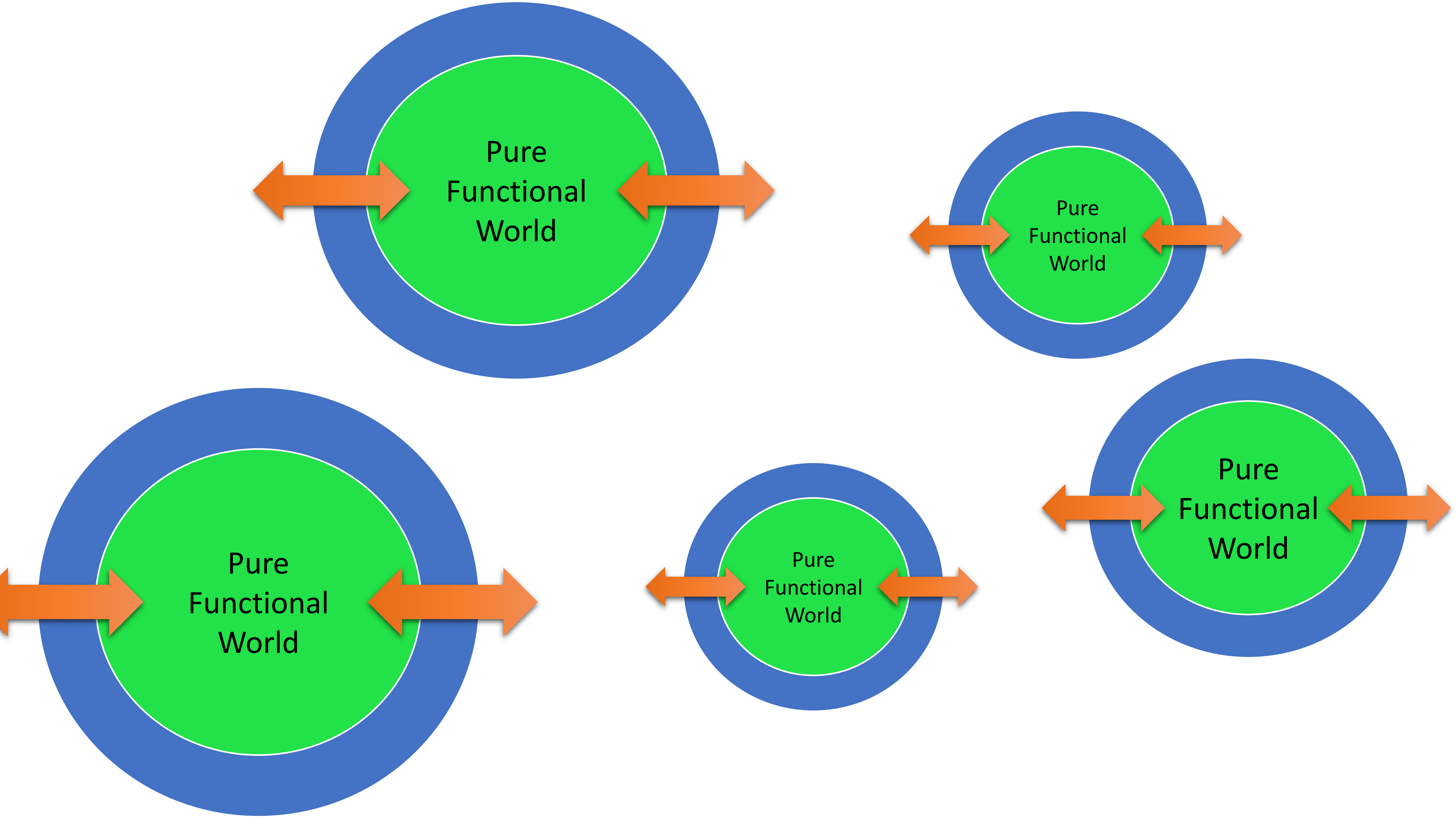
Purity is rather boring

Your app might look like...



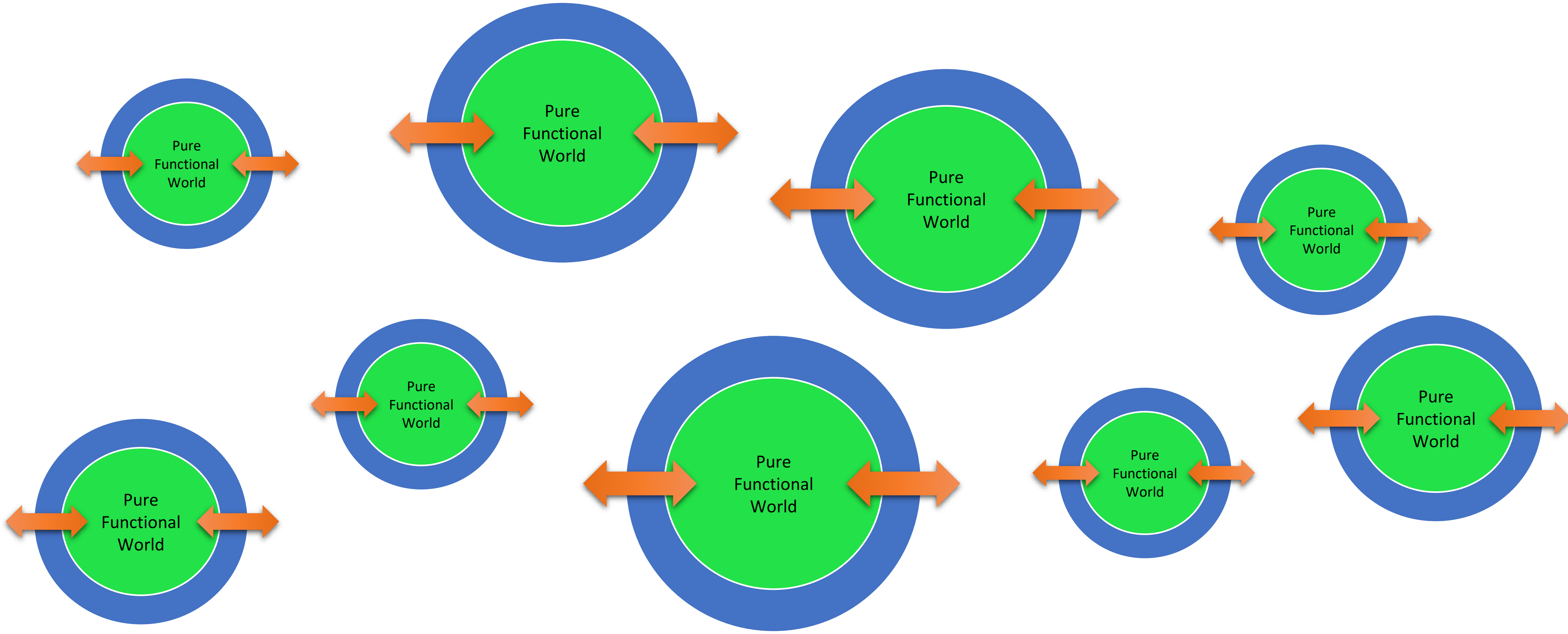

Pure Functional World

Separate pure and not pure code

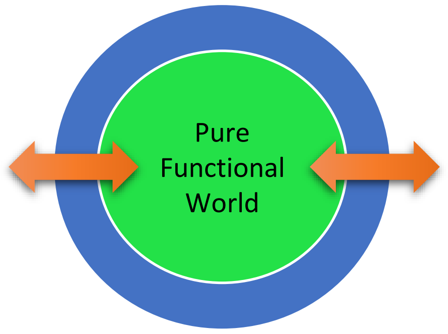

Pure Functional World
Pure Functional World
Pure Functional World
Pure Functional World
Pure Functional World

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

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	*Improving*	B-
OOP		A
Strong typing		A
Generics		A
Pattern matching	*Improving*	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

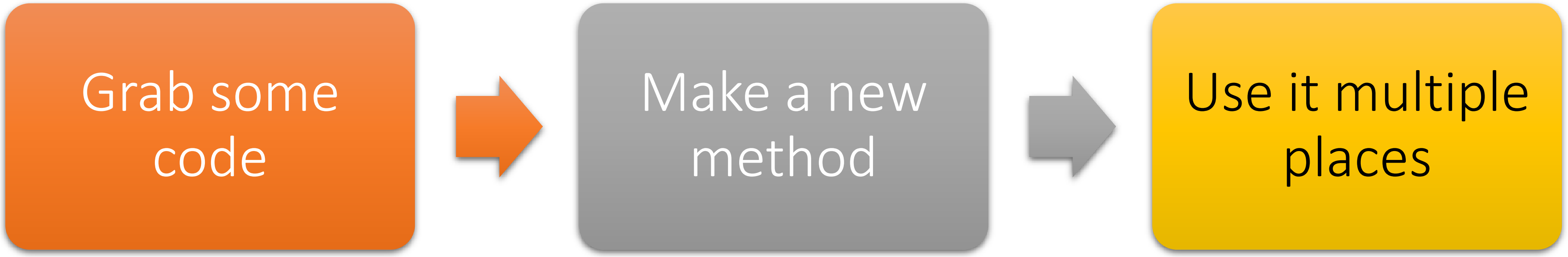
  ```
  var x = 42;
  var y = list.Where(z => z.Id == x);
  ```

  - 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

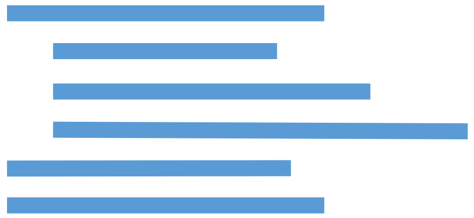
Refactoring to Functional

Imperative (normal) Refactoring

- Inside out refactoring



Imperative Refactoring



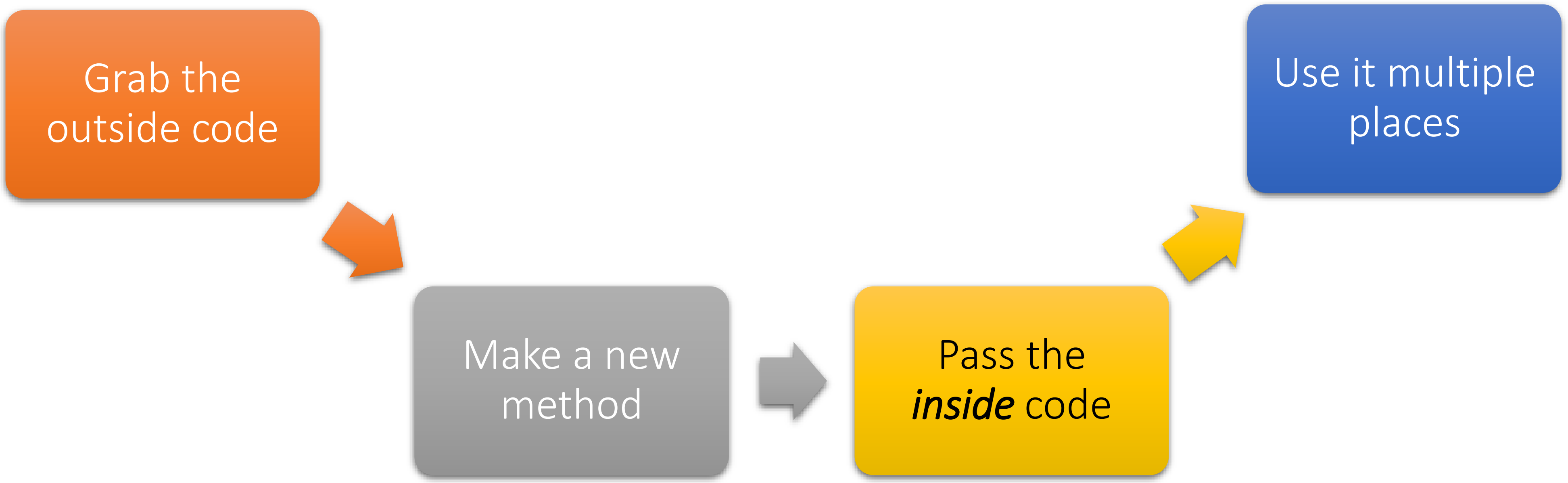
Imperative Refactoring

Demo!

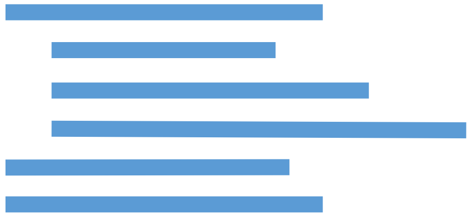
Inside out refactoring (normal)

Functional Refactoring

- Outside in refactoring



Functional Refactoring



Functional Refactoring

Functional Refactoring

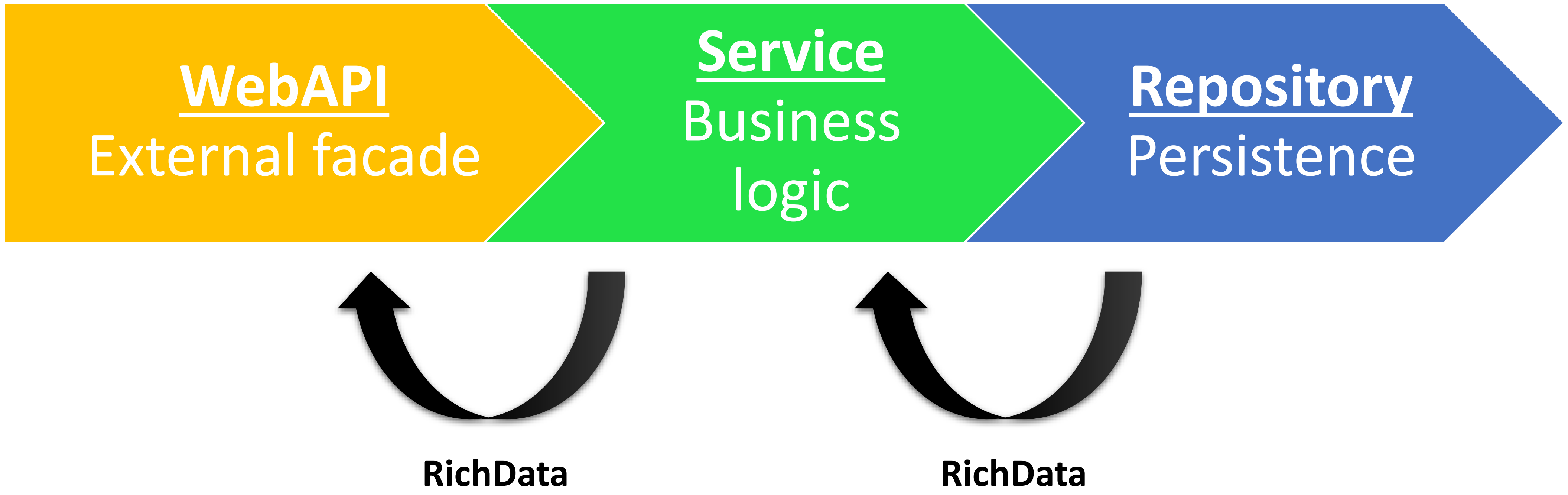
Demo!

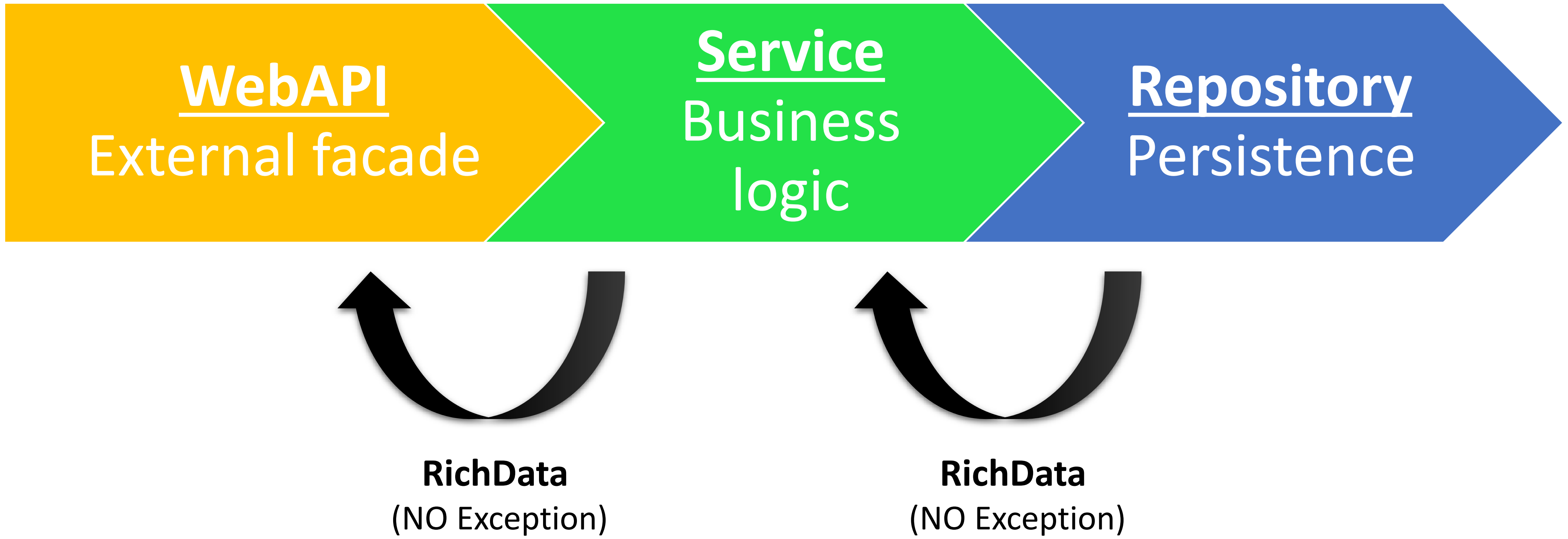
Outside in refactoring

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


WebAPI
External facade
Service
Business logic
Repository
Persistence
RichData
RichData


WebAPI
External facade
Service
Business logic
Repository
Persistence
RichData
(NO Exception)
RichData
(NO Exception)

Naming is hard

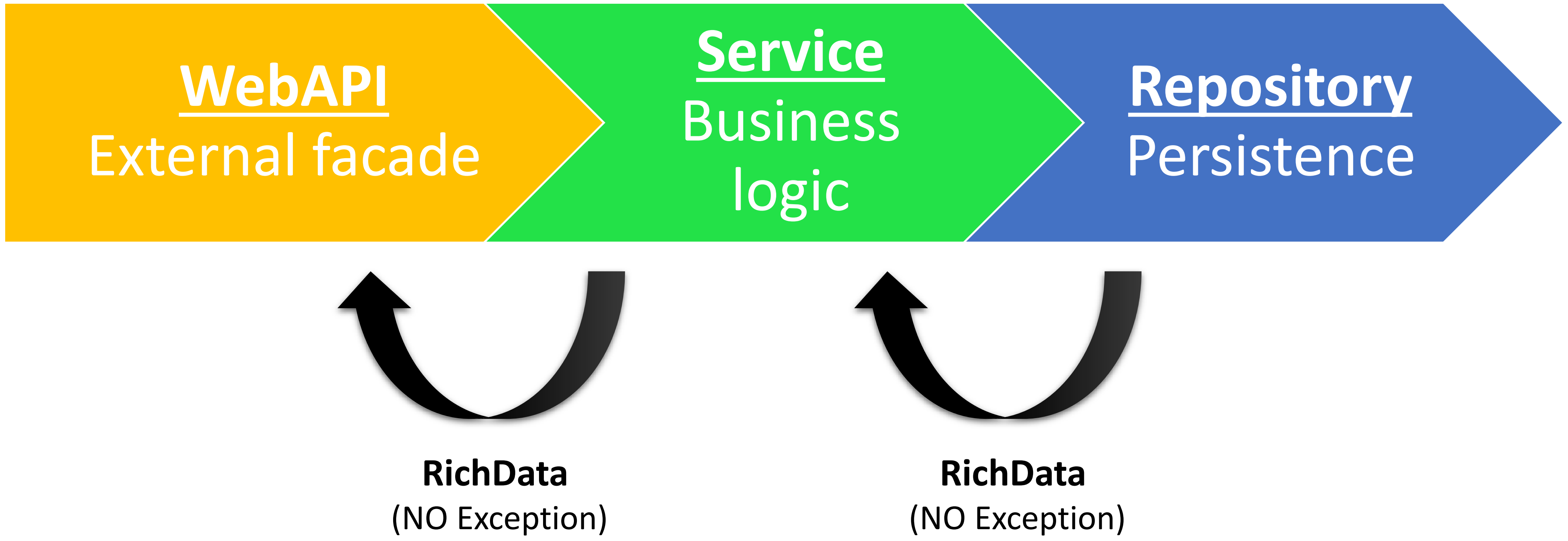
1. Either
2. RichData/RichValue
3. Result
4. Try
5. Validation<Exception<T>>

1) Core FP concept, 2 & 3) Kathleen 4) Sander 5) Enrico

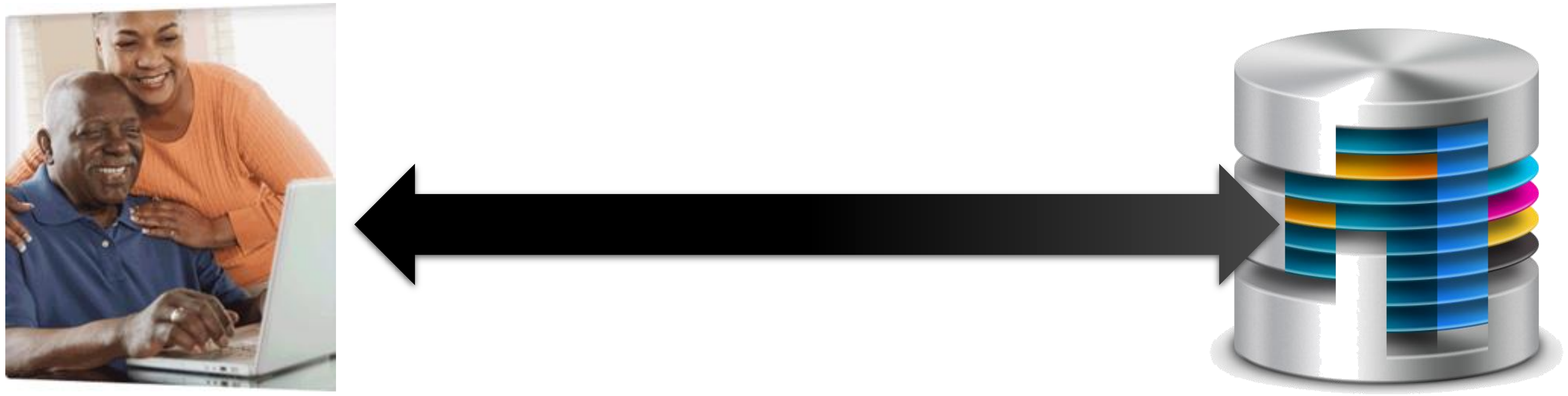
Naming is hard

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

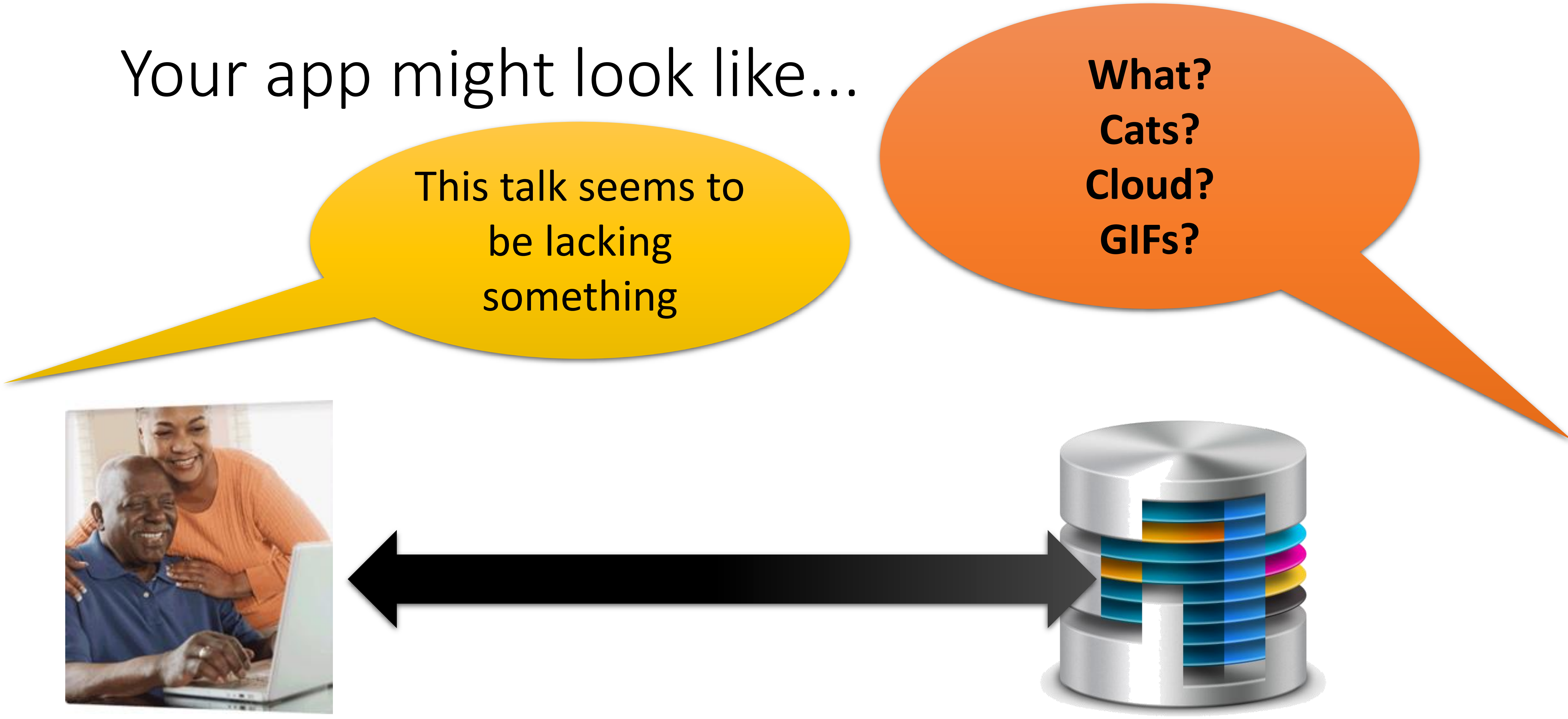
1) Core FP concept, 2 & 3) Kathleen 4) Sander 5) Enrico


WebAPI
External facade
Service
Business logic
Repository
Persistence
RichData
(NO Exception)
RichData
(NO Exception)

Your app might look like...

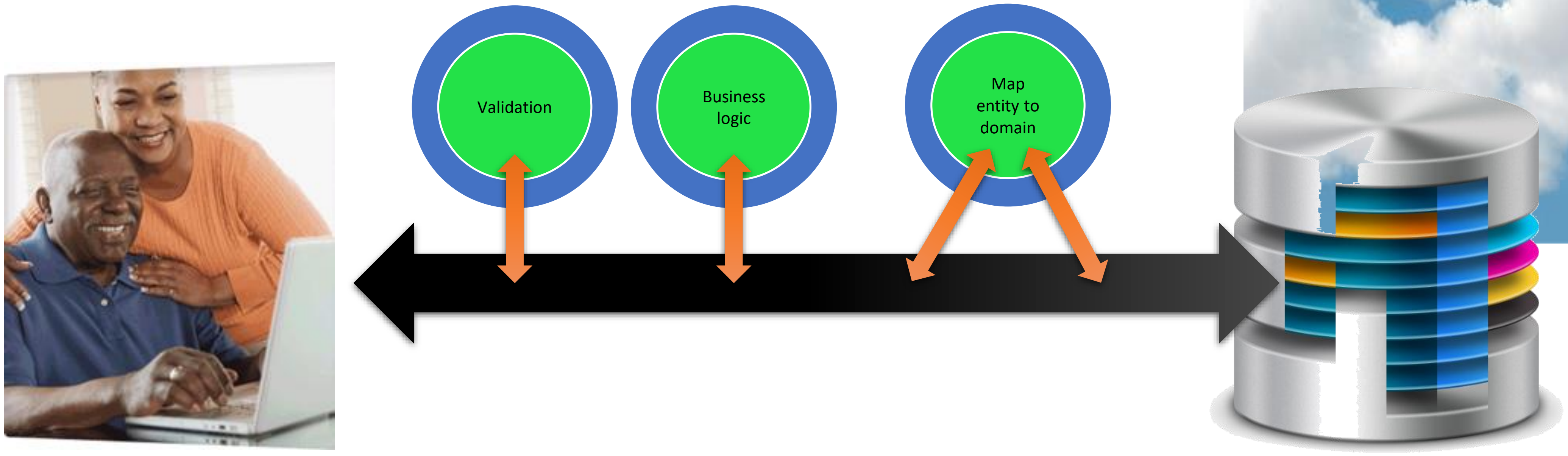


Your app might look like...

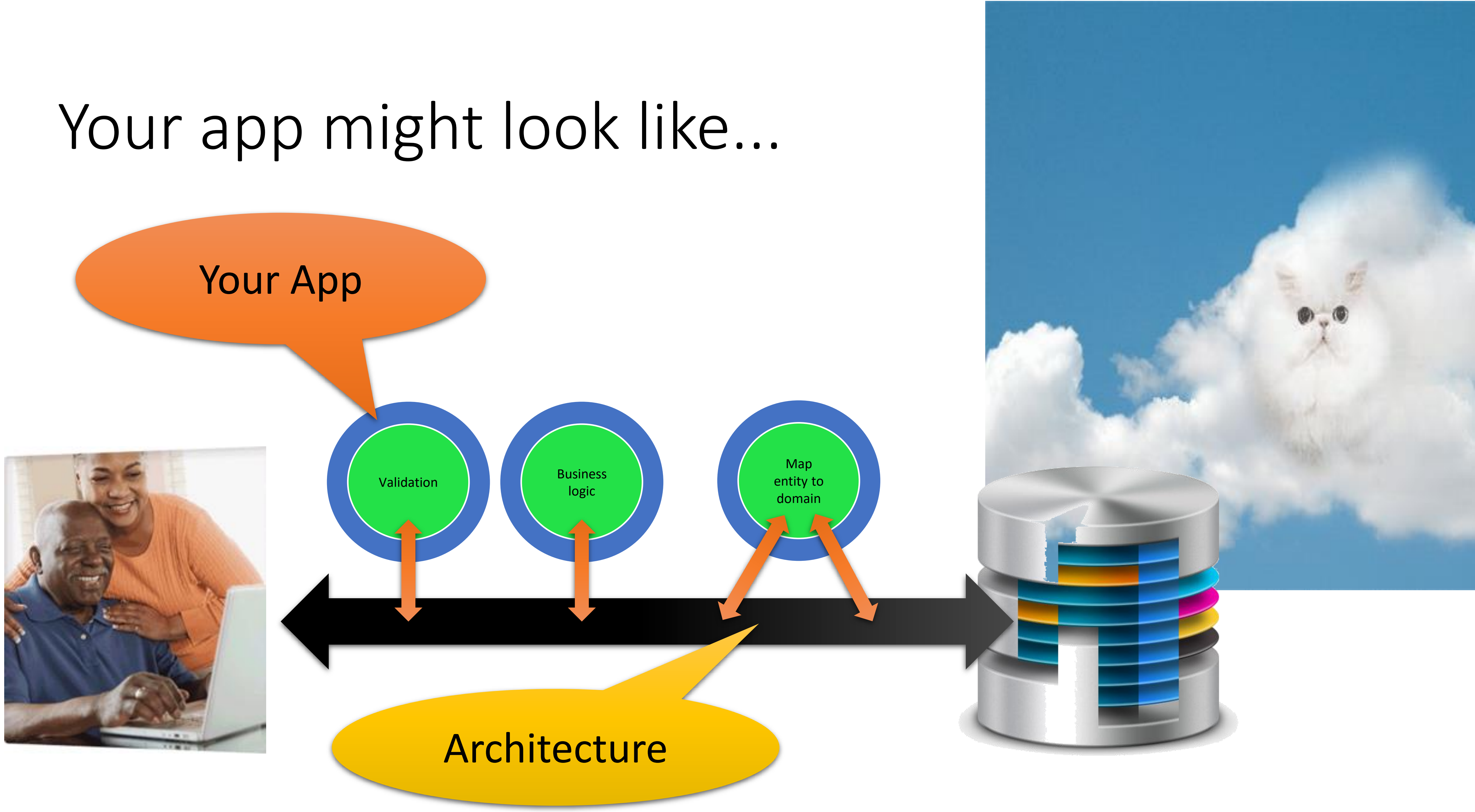


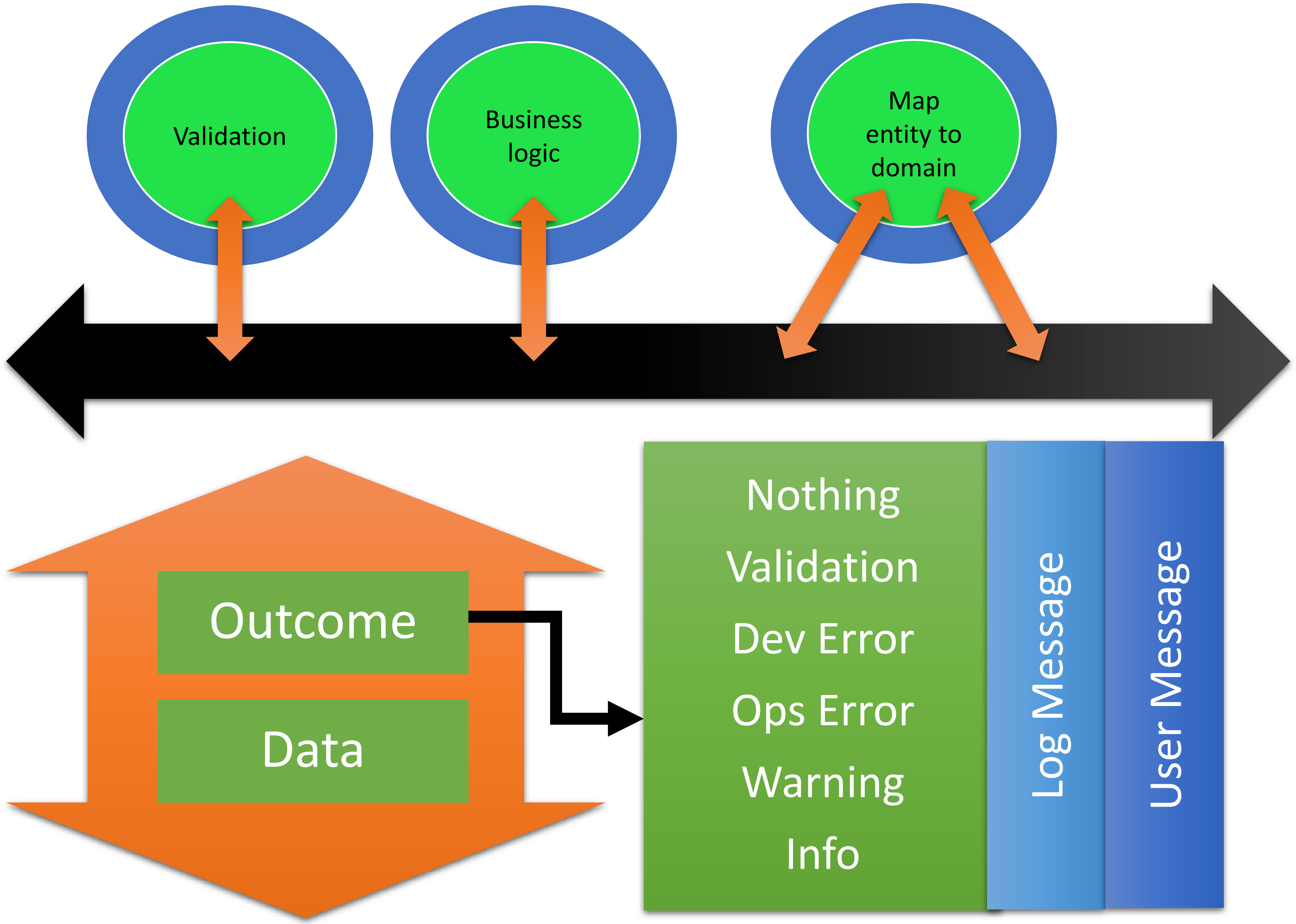
Your app might look like...

Your app might look like...



Your app might look like...





Validation
Business logic
Map entity to domain
Outcome
Data
Nothing
Validation
Dev Error
Ops Error
Warning
Info
Log Message
User Message

Demo!

Outcome

RichData<TData>

Useful things in C#

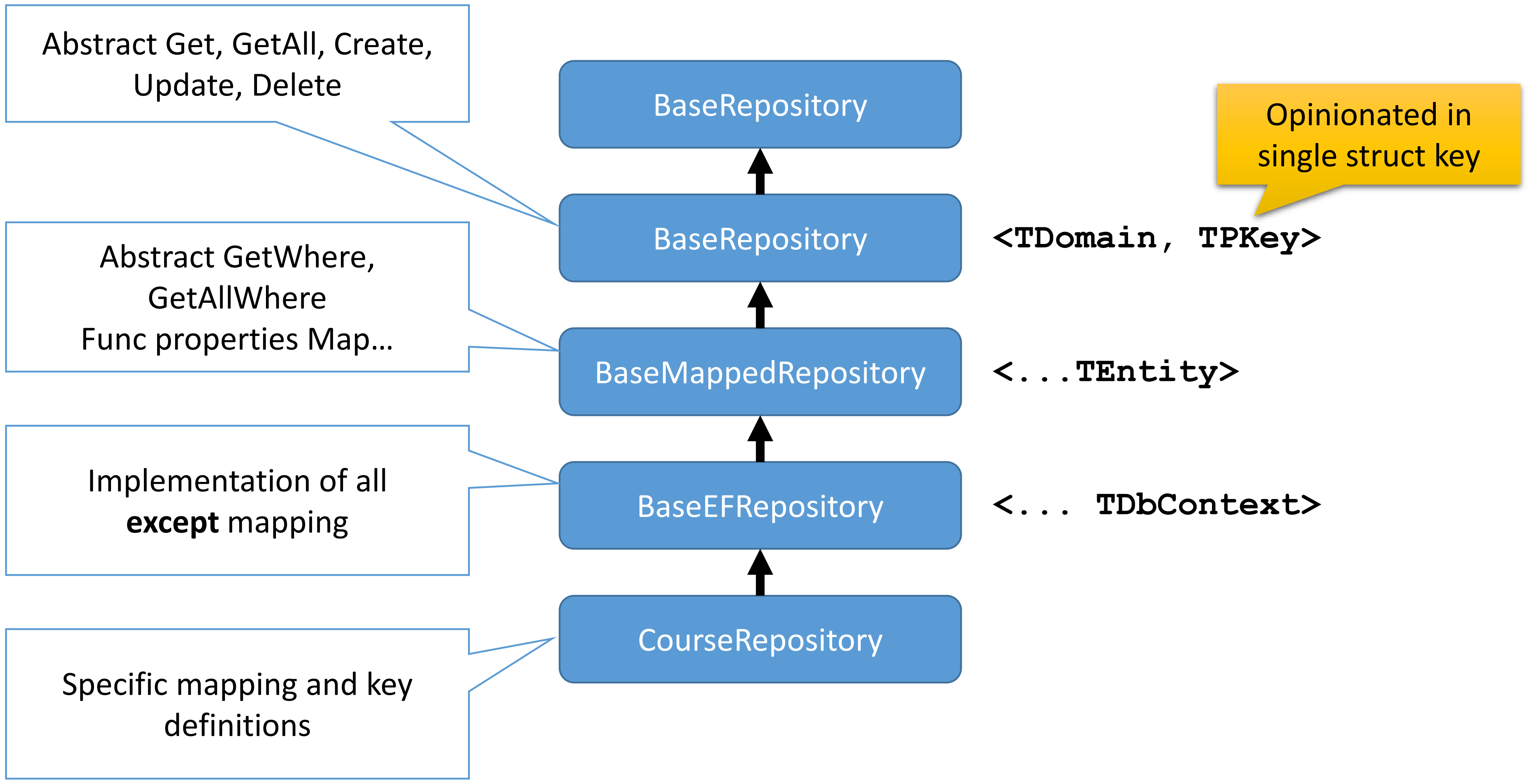
- Expression-body members
- Generic inheritance hierarchies
- Expression trees
- Local functions
- Pattern matching (enhanced switch statement)
- Tuples
- Throw expression

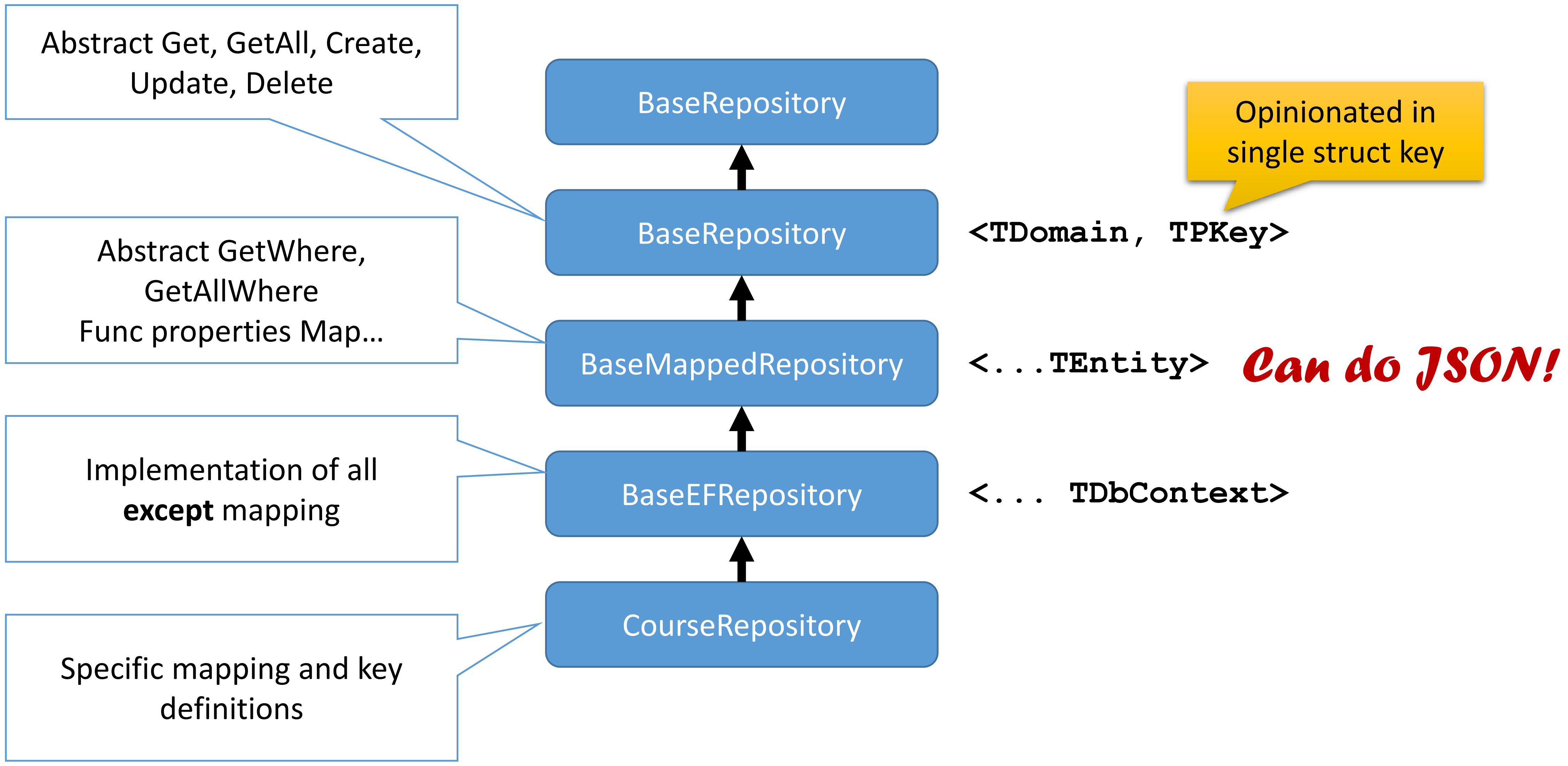
8

- Pattern matching (switch expression)
- Default interface implementation (rich interfaces)

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


Abstract Get, GetAll, Create, Update, Delete
Abstract GetWhere, GetAllWhere Func properties Map…
Implementation of all **except** mapping
Specific mapping and key definitions
BaseRepository
BaseRepository
BaseMappedRepository
BaseEFRepository
CourseRepository
Opinionated in single struct key
<TDomain, TPKey>
<...TEntity>
<... TDbContext>

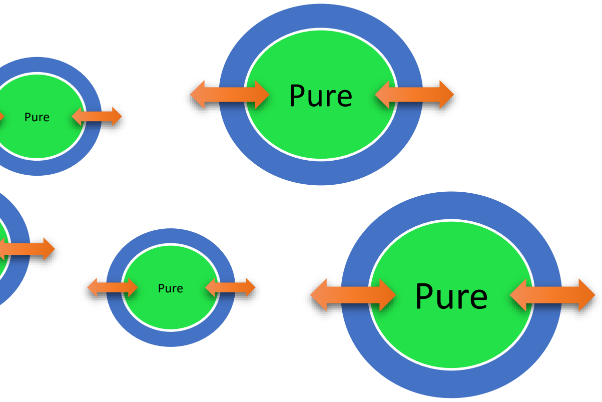

Abstract Get, GetAll, Create, Update, Delete
Abstract GetWhere, GetAllWhere Func properties Map…
Implementation of all **except** mapping
Specific mapping and key definitions
BaseRepository
BaseRepository
BaseMappedRepository
BaseEFRepository
CourseRepository
Opinionated in single struct key
<TDomain, TPKey>
<...TEntity>
Can do JSON!
<... TDbContext>

Recap

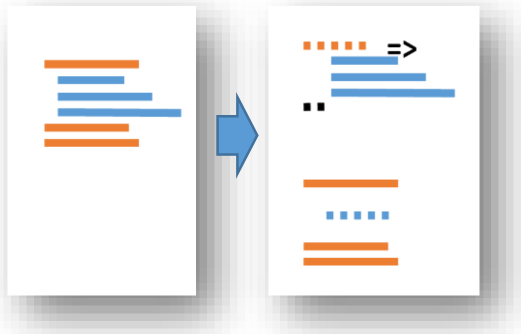
- We have a desire for functional
- Functional includes cultural change
- Fashion can add real value, or not

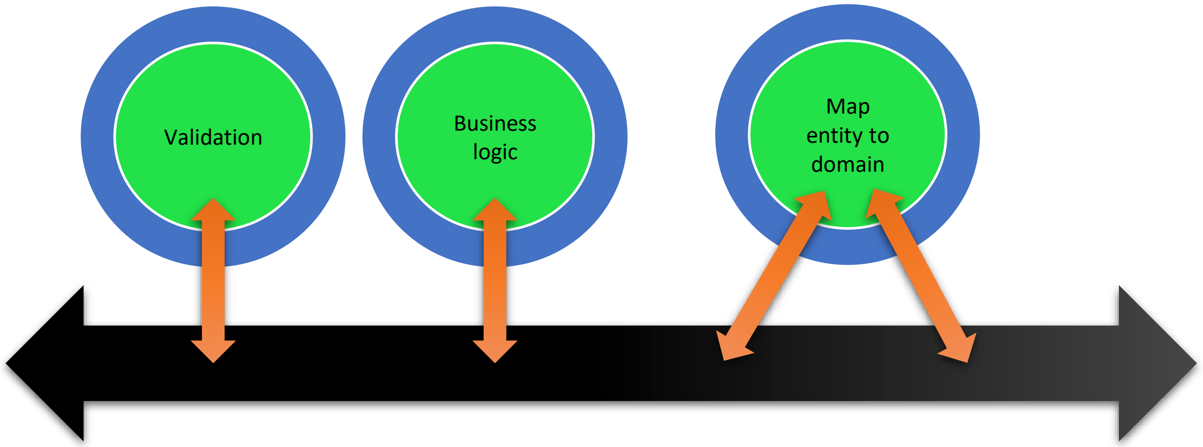
Explore and experiment

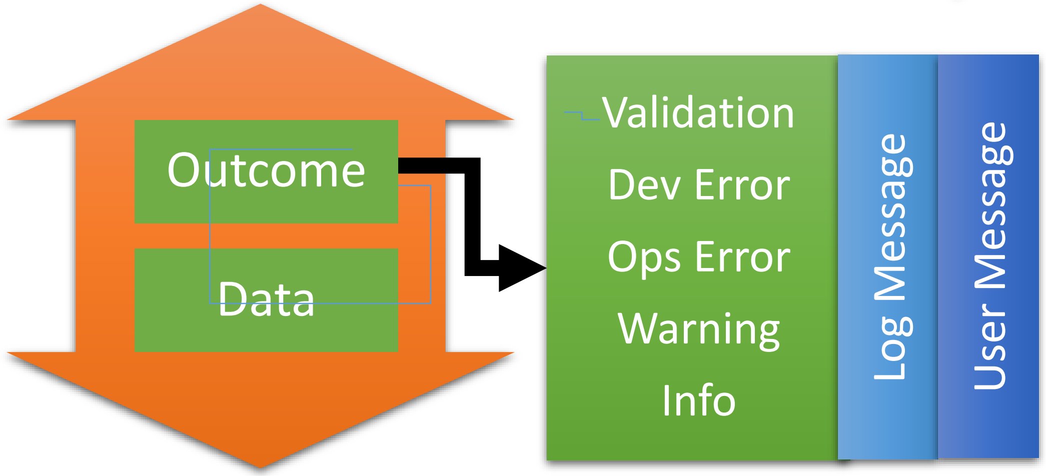
C# ❤ Functional techniques



- 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