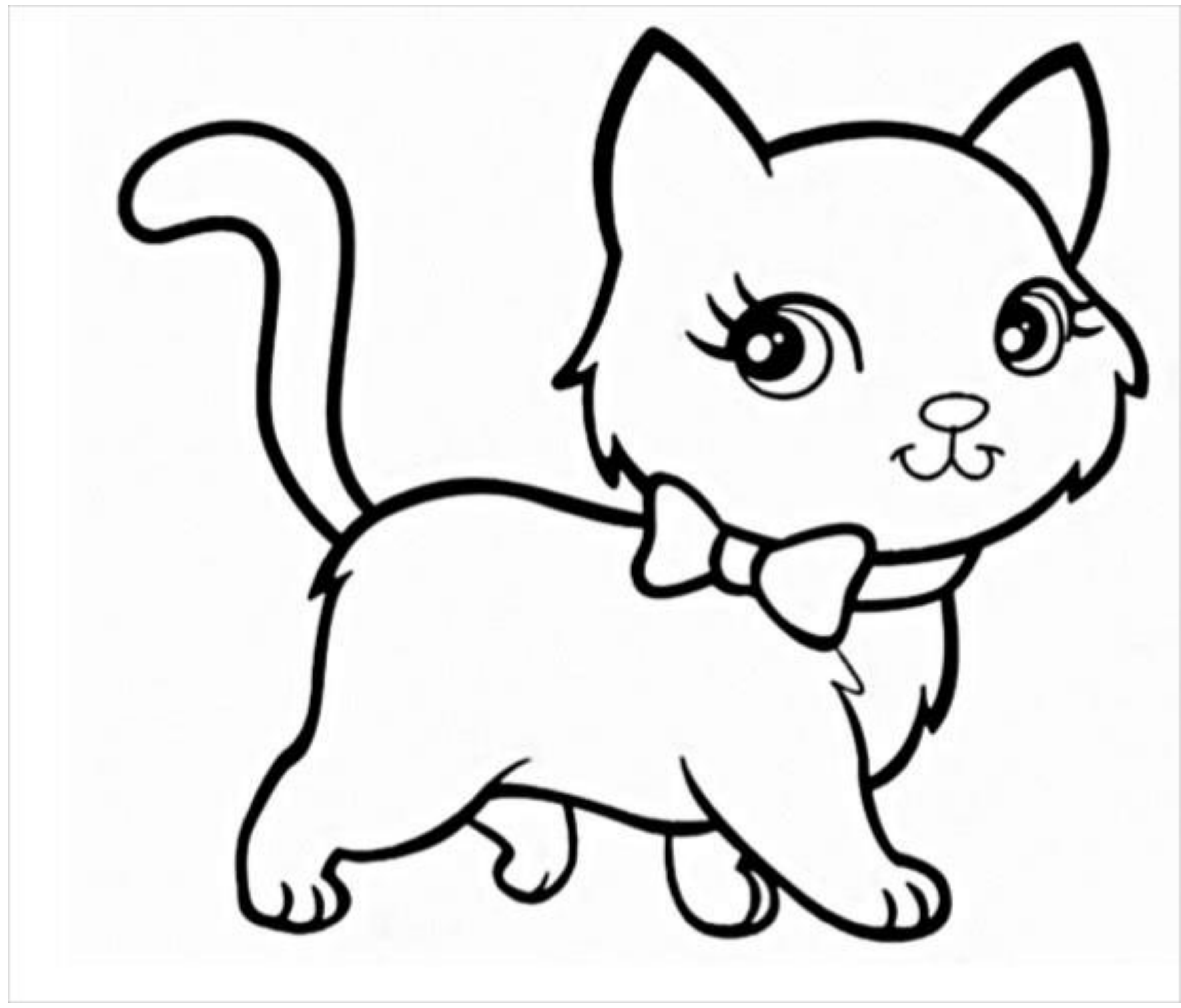
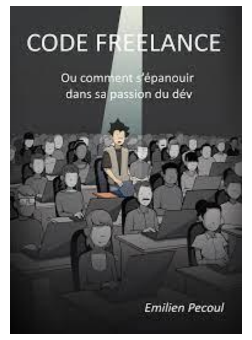
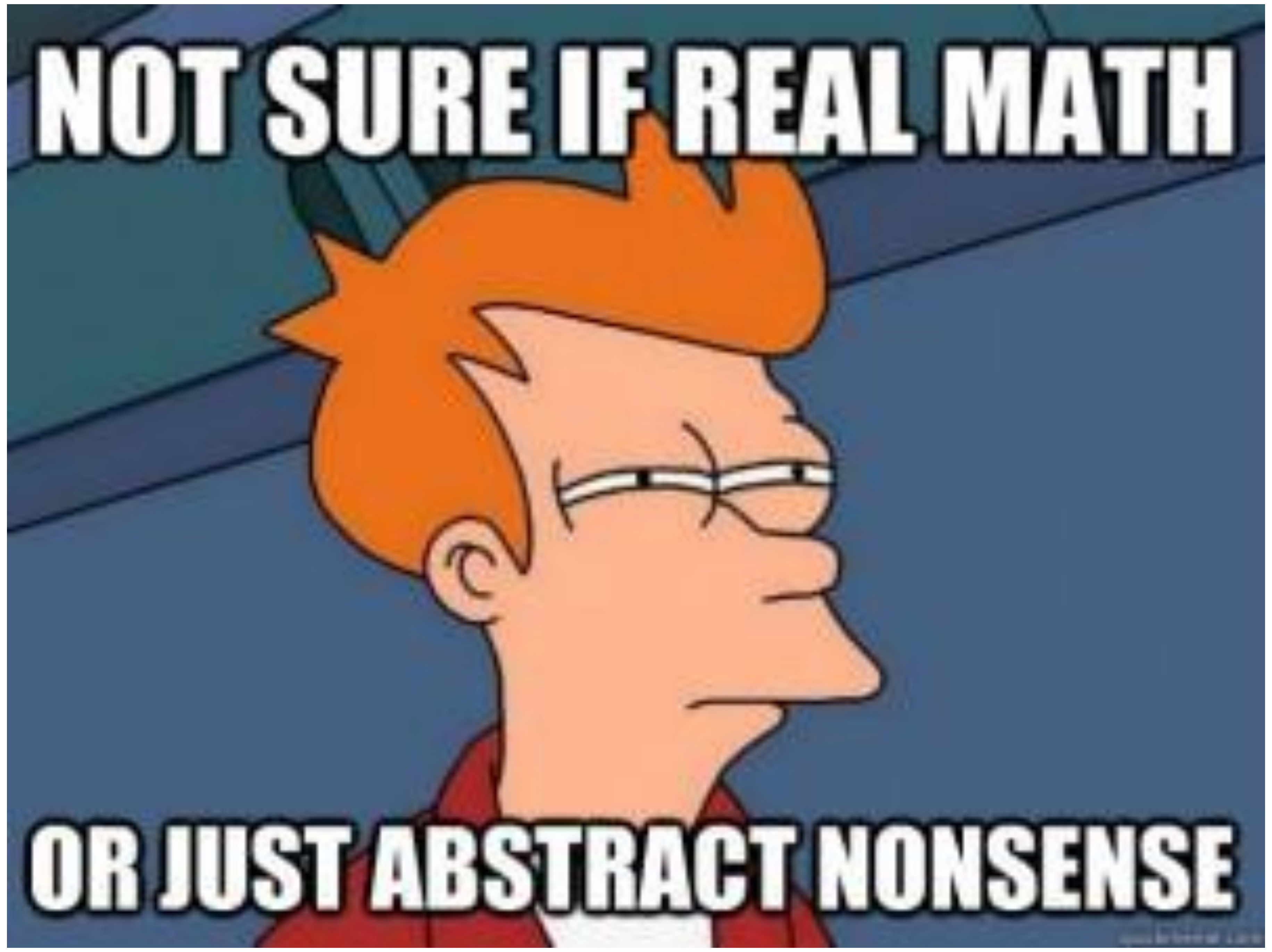
Category Theory:
You already know it



Romain Berthon @romaintrm
Emilien Pecoul @ouarzy



Disclaimer



- We’re not mathematicians
- Just curious programmers

Question?

www.slido.com

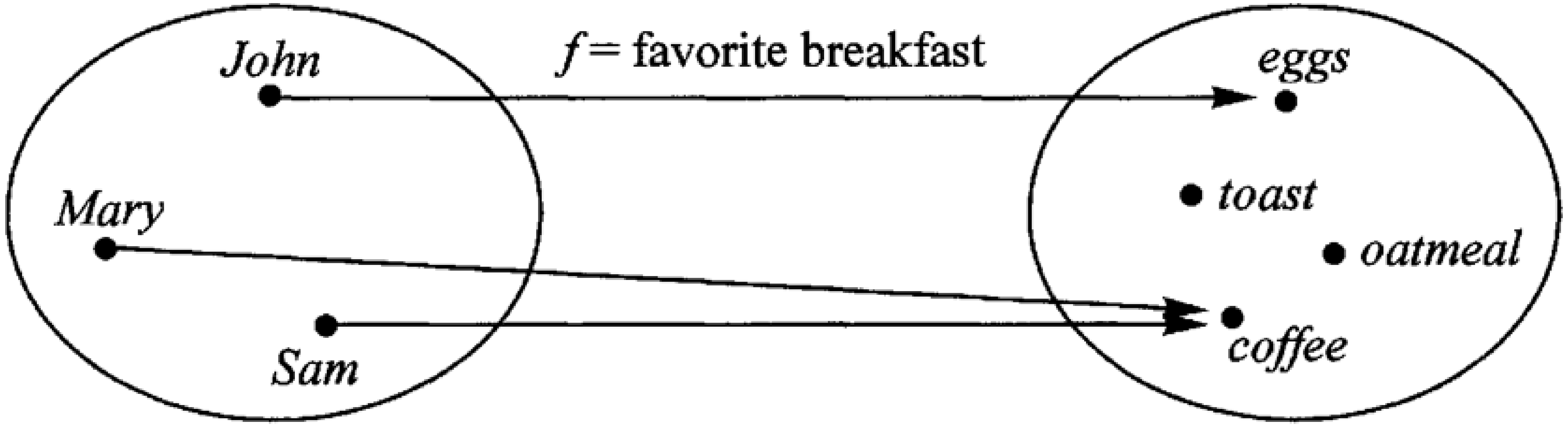
#223494

Why Category Theory ?

Common abstract language

Behaviors analysis instead of object analysis

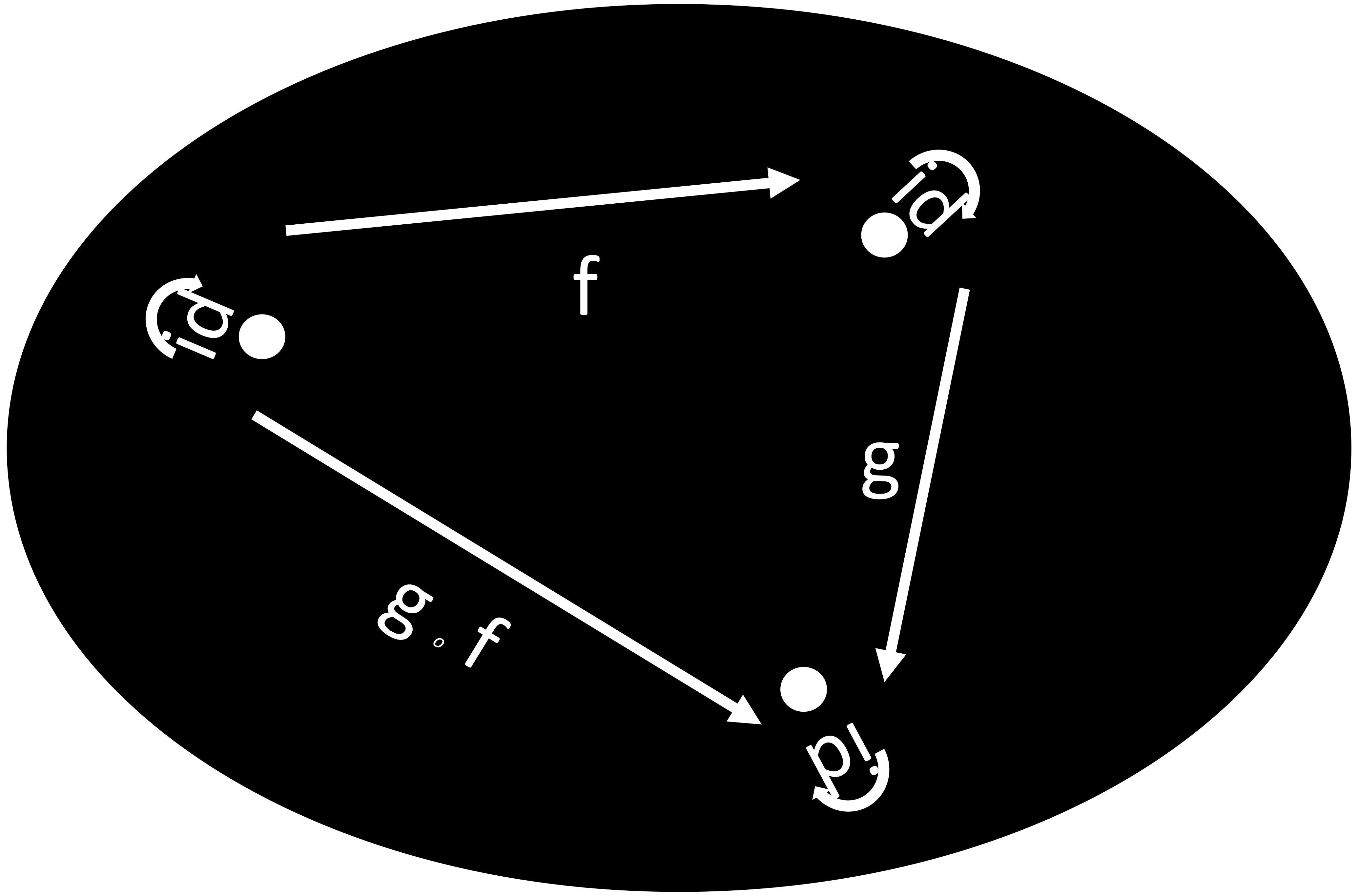
It's all about structure



Code review

```
Romain BERTHON
public Amount GetTotalAmountOfSuspiciousOperations(IReadOnlyList<AccountLine> lines)
{
    var suspiciousOperations :IReadOnlyList<AccountLine> = GetSuspiciousOperations(lines);
    return GetTotalAmount(suspiciousOperations);                          // # 1 Composition
}

1 usage  Romain BERTHON
public IReadOnlyList<AccountLine> GetSuspiciousOperations(IReadOnlyList<AccountLine> lines) =>
    lines // IReadOnlyList<AccountLine>
        .Select(line => (line, isSuspicious: IsSuspiciousAmount(line))) // # 3 Functor & map
                                                                         // Product
        .SelectMany(x :(line,isSuspicious) => x.isSuspicious                         // Where
            ? new List<AccountLine> { x.line }                           // Monad & Bind
            : new List<AccountLine>()) // IEnumerable<AccountLine>
        .ToList();                                                       // # 1 Composition

1 usage  Romain BERTHON
private static bool IsSuspiciousAmount(AccountLine line) =>             // # 2 Morphisms: Loss of information
    line.Amount.Value > 10_000m;

1 usage  Romain BERTHON
public Amount GetTotalAmount(IReadOnlyList<AccountLine> lines) =>
    lines // IReadOnlyList<AccountLine>
        .Select(line => line.Amount)                                     // # 3 Functor & map
        .Aggregate(Amount.Zero, Amount.Add);                             // Monoid
}
```

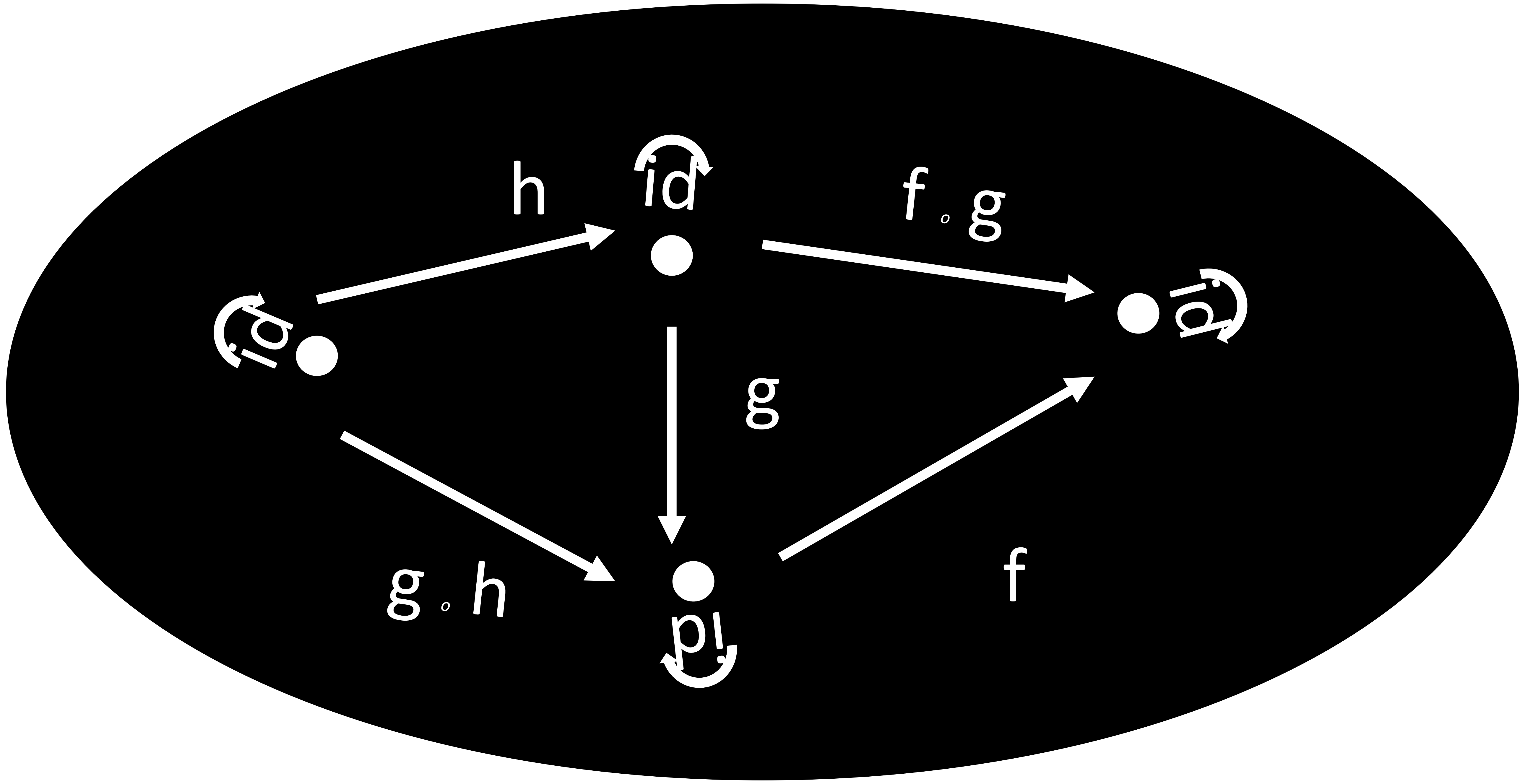
What's Category Theory ?



A bunch of **morphisms,** they start and finish with an **object**

Identity: (a->a)
Composition: (a ->b) -> (b->c) = (a->c)

What's Category Theory ?



Identity law: (id *o* f) = (f *o* id) = f

Associative law: (f *o* g) *o* h = f *o* (g *o* h)

Essence of Category Theory ?

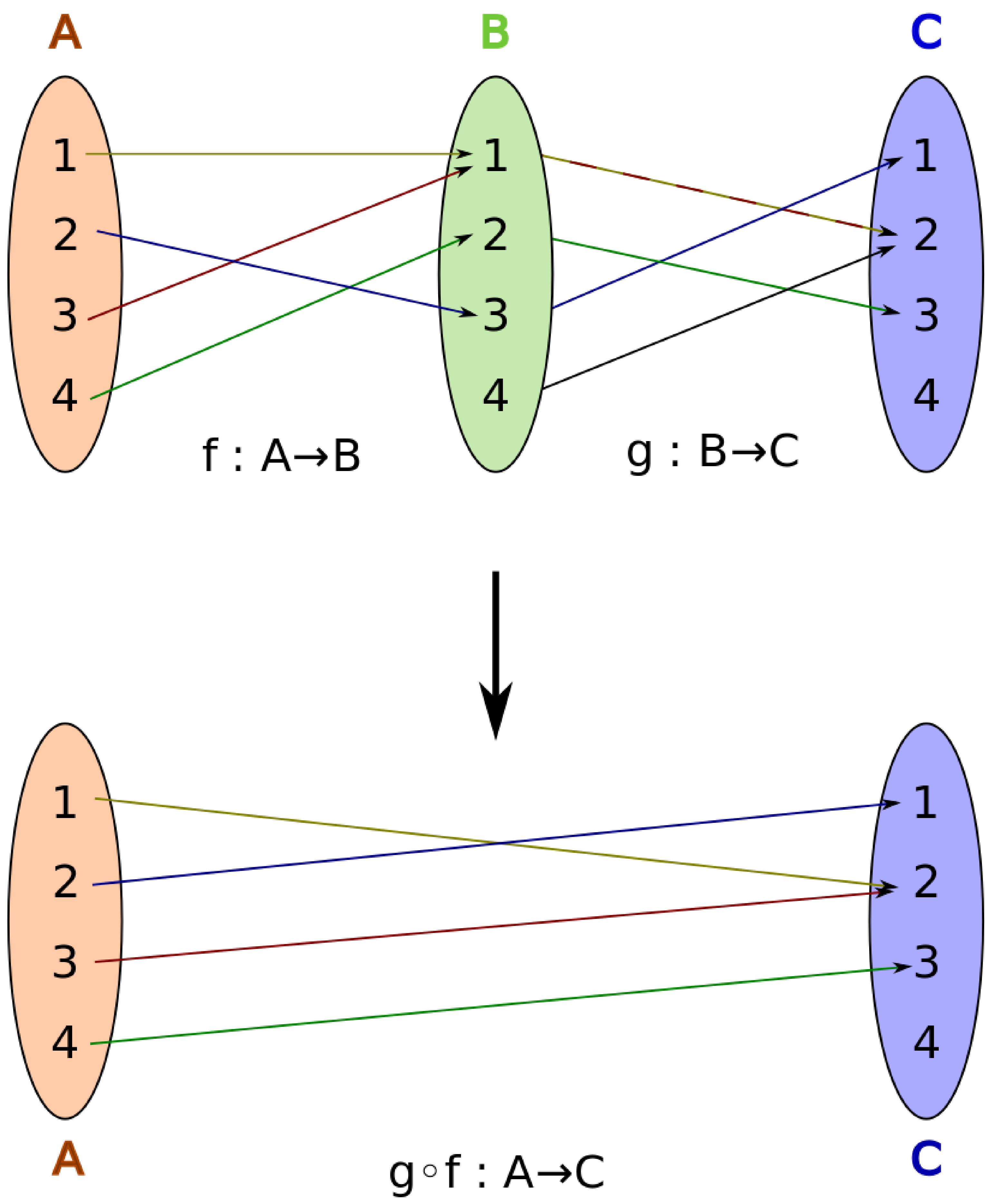
Composition

Identity

Abstraction

(Functional)
Programmers know it!

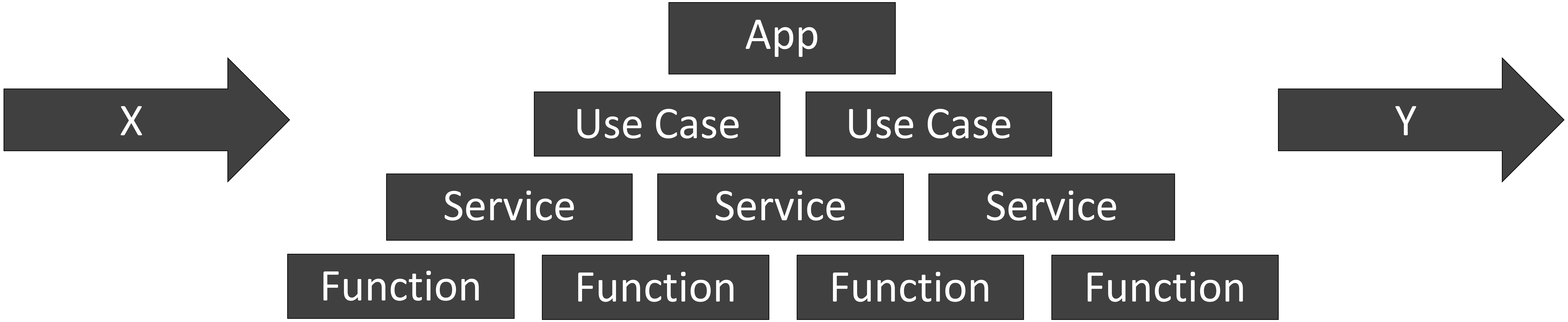
Composition



Composition

Fundamental in functional programming:

- The result of each function is the argument of the next
- The result of the last one is the result of the whole



Composition

Chunk big problems into smaller ones

Requires no side effect

Example ?

```
Romain BERTHON
public class Composition
{
    1 usage   Romain BERTHON
    private int f(decimal value) => (int)value;

    1 usage   Romain BERTHON
    private bool g(int value) => value % 2 == 0;

    Romain BERTHON
    public bool g_after_f(decimal value) => g(f(value));
}
```

```
decimal -> int   Romain BERTHON
let f (x: decimal) = int x
int -> bool   Romain BERTHON
let g (x: int) = x % 2 = 0
decimal -> bool   Romain BERTHON
let g_after_f = f >> g
```

Composition

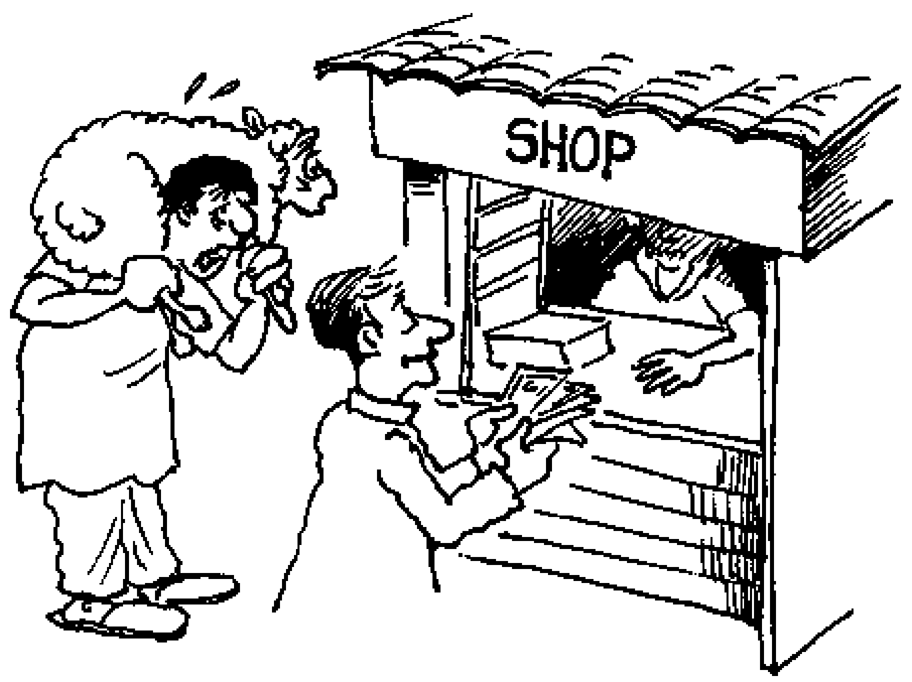
Composition allows to solve big problems with little composed solutions

We need composition because reality is too hard to deal with

Abstraction

How do we build software ?

Reality → Software Model → Code



Product

Money

Person

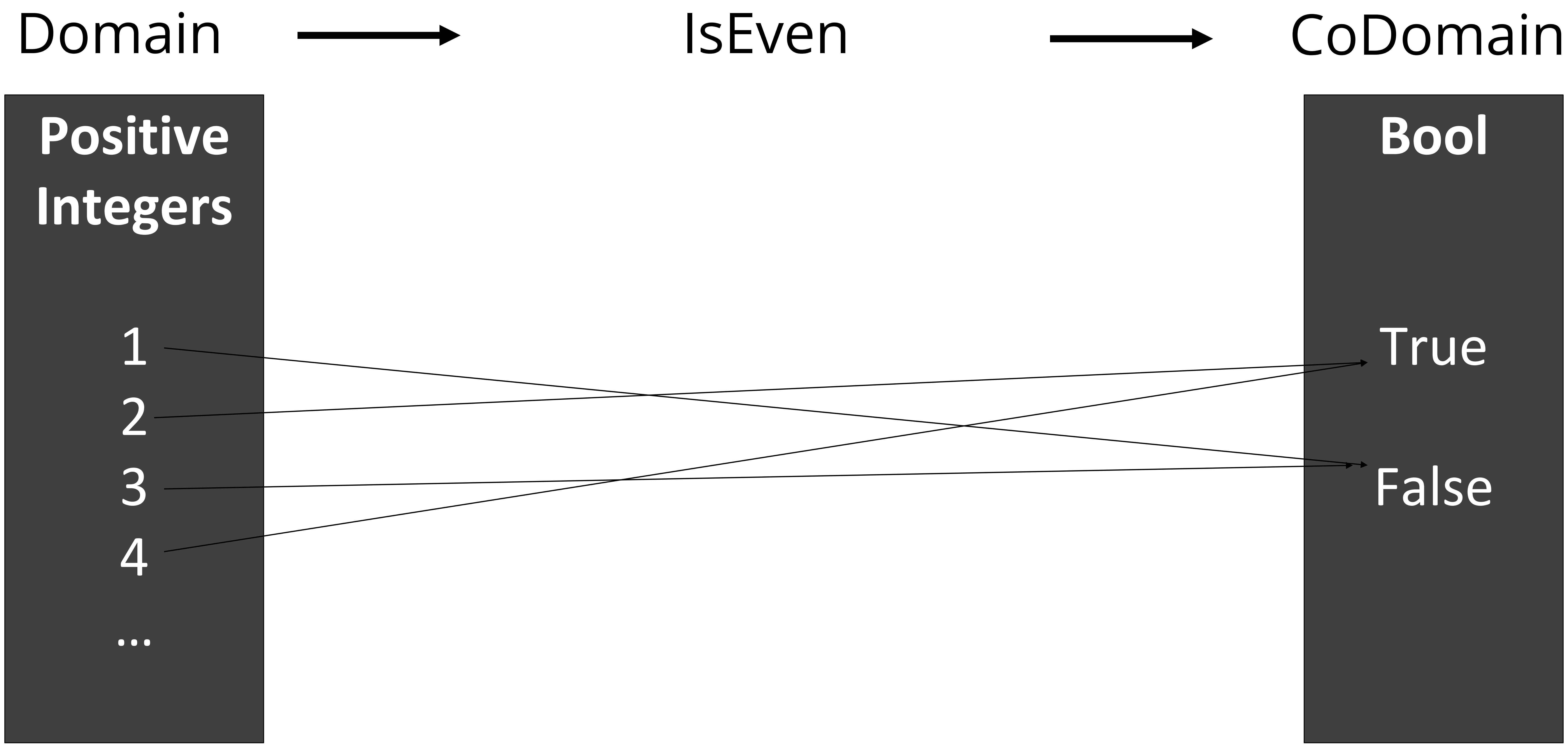
Seller

Buyer

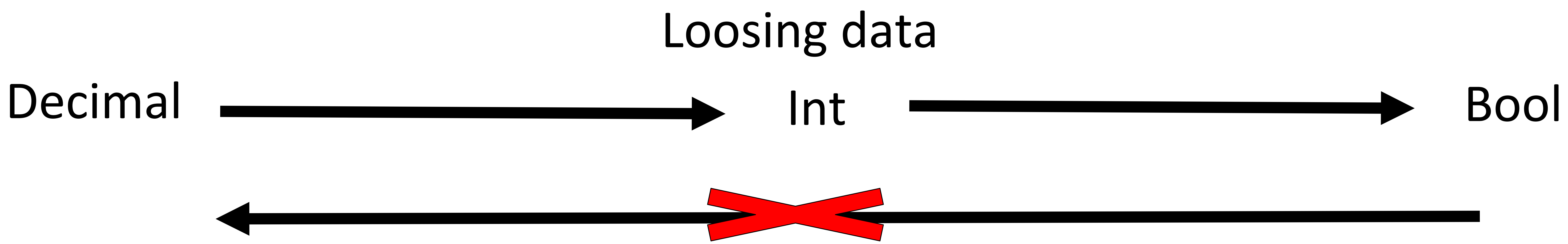
```java
class Checkout {
  private double tax;

  public Checkout(double tax) {
      this.tax = tax;
  }

  double total(final List<Product> products) {
   return products
            .stream()
            .mapToDouble(Product::getPrice)
            .sum() * tax;
  }
}
```

Abstraction

How do we abstract stuff ?



Example ?




```
AmountState EvaluateAmountState(AccountLine line)
```

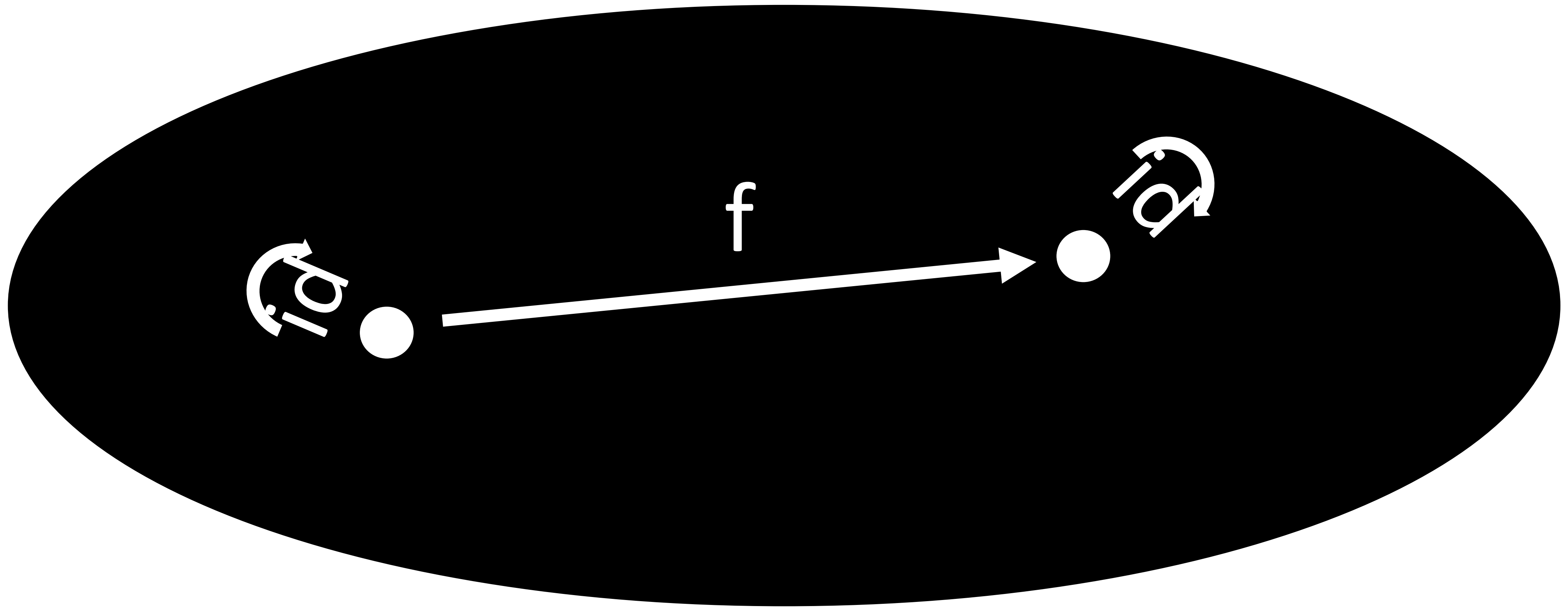
Abstraction

Abstracting is hiding (loosing) useless details in order to apply common rules

We need abstraction because reality is too hard to deal with

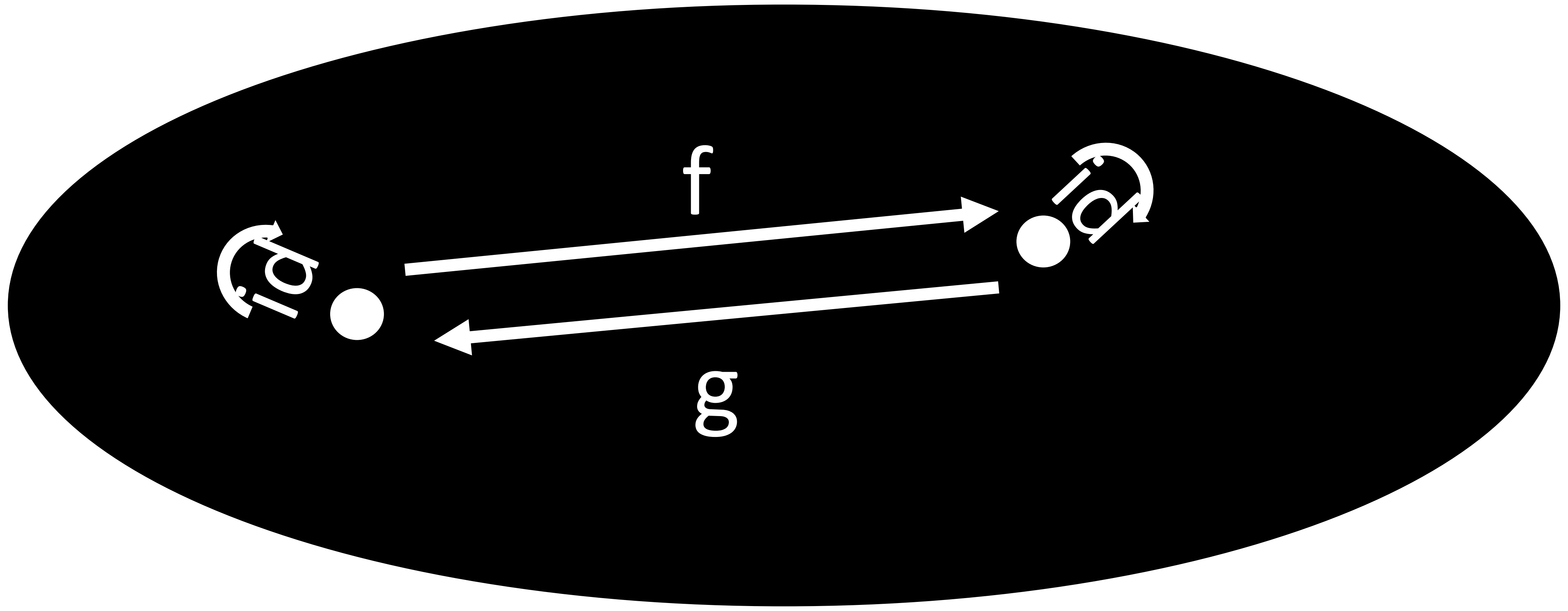
Identity

Identity



Identity law: (id *o* f) = (f *o* id) = f

Identity



Identity law: $(id \circ f) = (f \circ id) = f$

$g \circ f = Id_{Left}$

$f \circ g = Id_{Right}$

Identity

Does (string,int) = (int,string) ?

Isomorphic

$$\begin{tikzcd}
(\text{string, int}) \arrow[loop left, "f \circ g"] \arrow[r, bend left, "f"] \arrow[r, phantom, "\cong"] & (\text{int, string}) \arrow[l, bend left, "g"] \arrow[loop right, "f \circ g"]
\end{tikzcd}$$

$g \circ f = Id_{Left}$

$f \circ g = Id_{Right}$

Example ?

Not loosing data

$$\text{Binary} \underset{g}{\overset{f}{\rightleftarrows}} \text{Bool}$$

$$f \circ g = \text{IdRight}$$

$$g \circ f = \text{IdLeft}$$

$$\text{Model} \rightleftarrows \text{BDD}$$

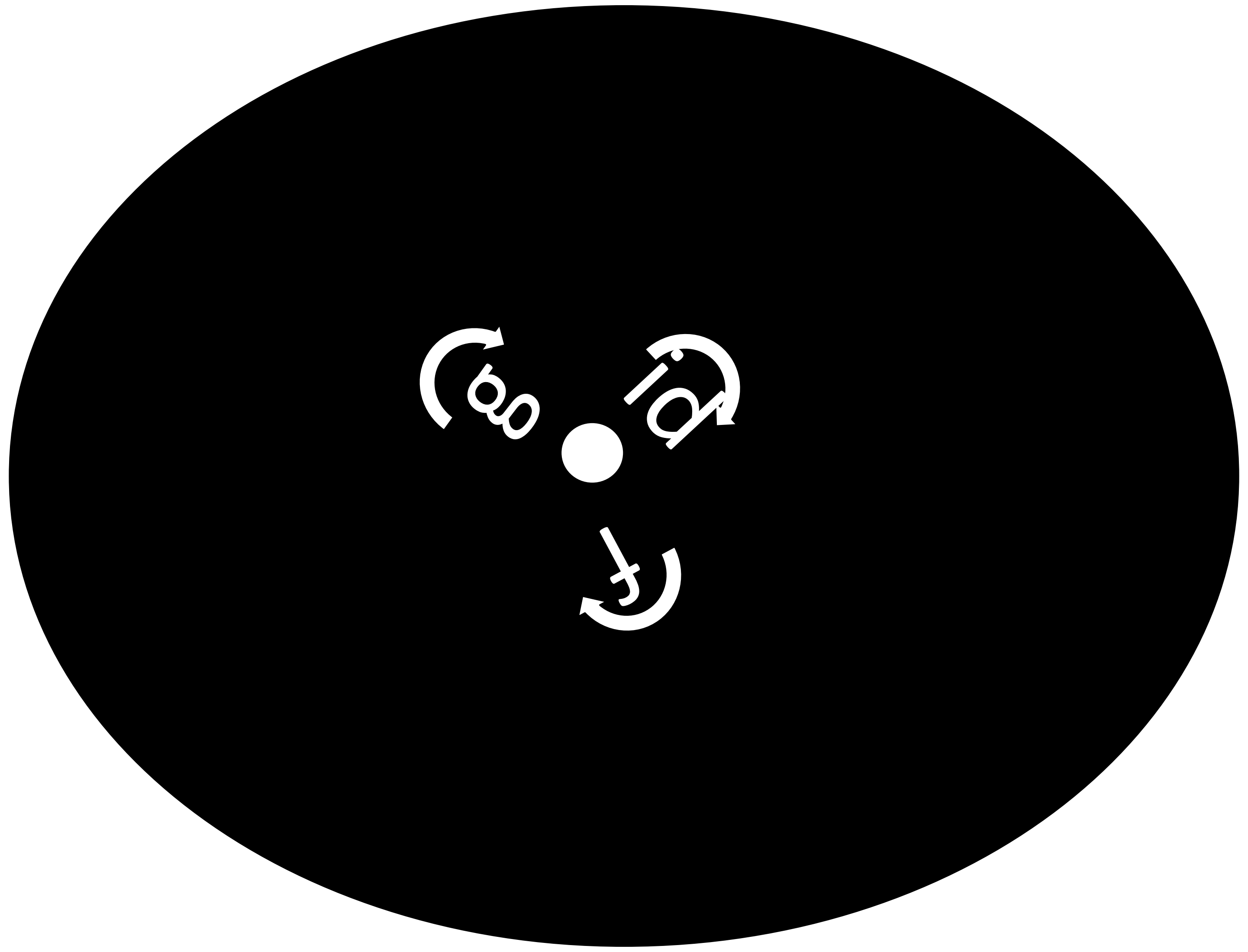
Identity

Identity is important to define an object in a given context

We need identity because reality is too hard to deal with

You already know it!

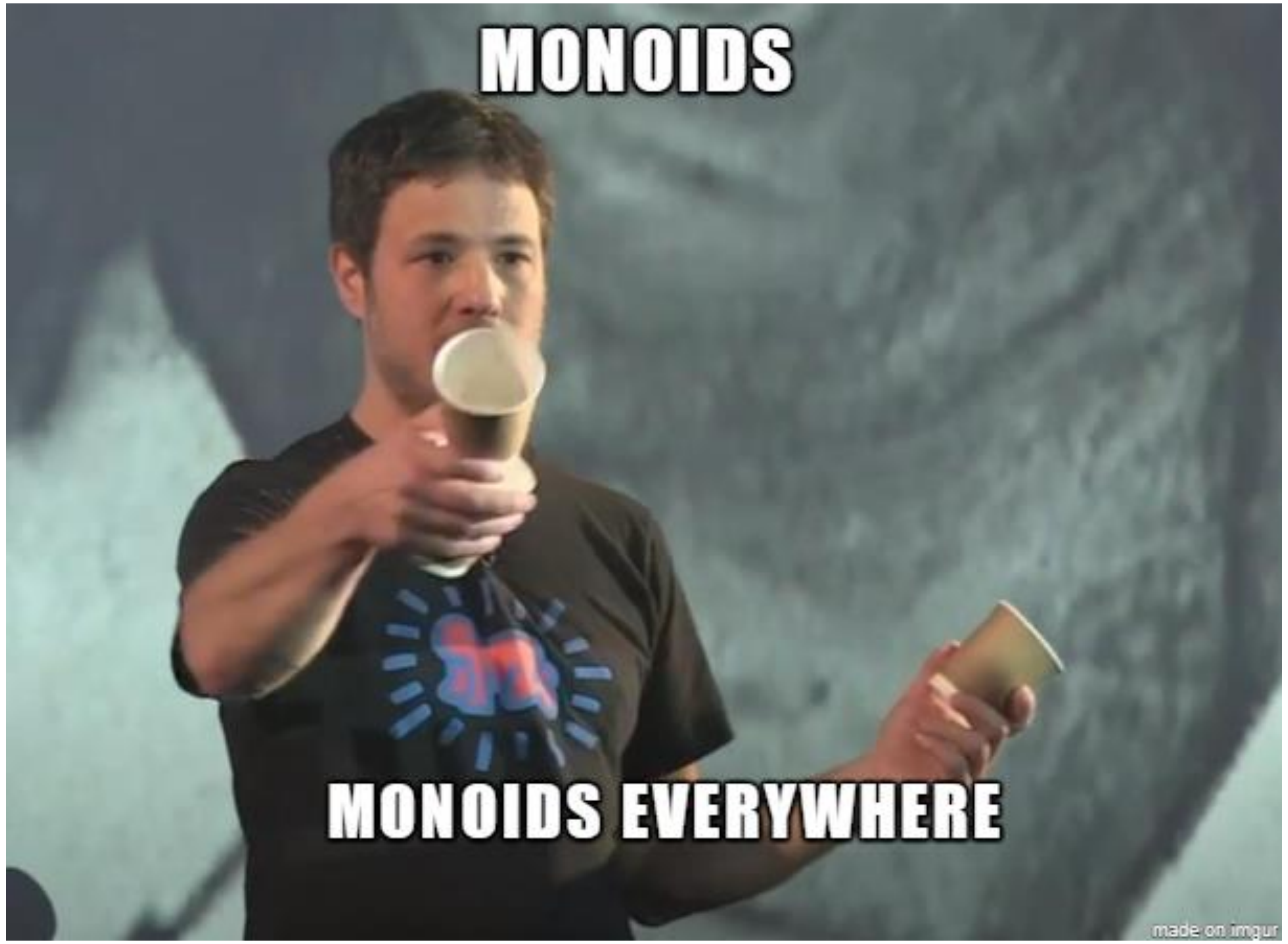
Monoids



A category with **only one** object

=> **Compose**! **Associative !**
=> Structure containing only monoids are monoids !

Monoids



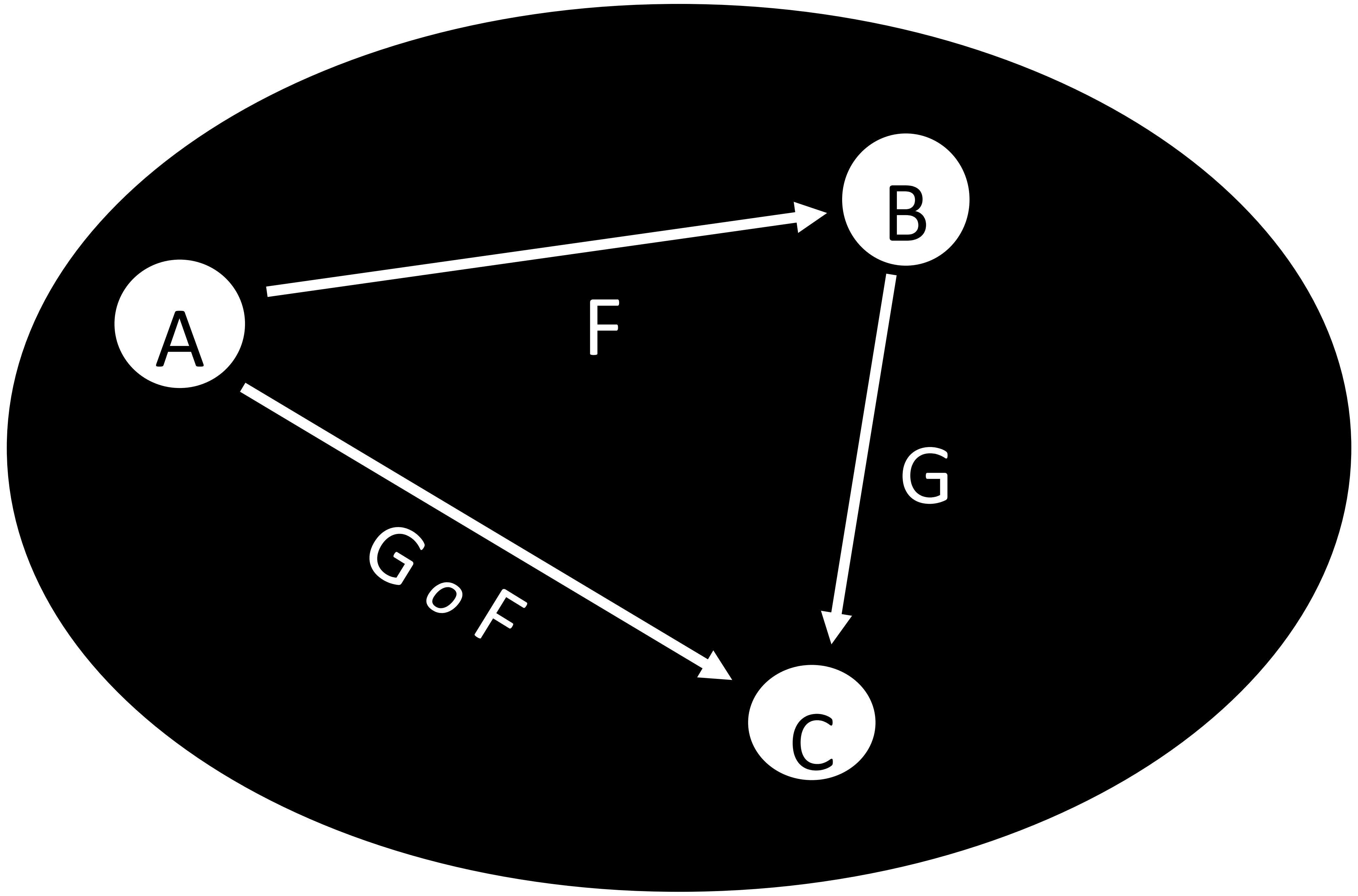
https://youtu.be/J9UwWo2qifg

Example ?

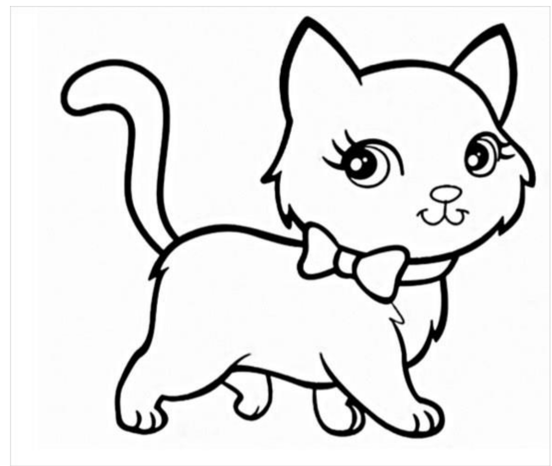
```
public record Amount(decimal Value)
{
    public static Amount Add(Amount left, Amount right) => new(left.Value + right.Value);
    public static readonly Amount Zero = new(0m);                    // Monoid's Neutral element
}
```

Break

Category you already know



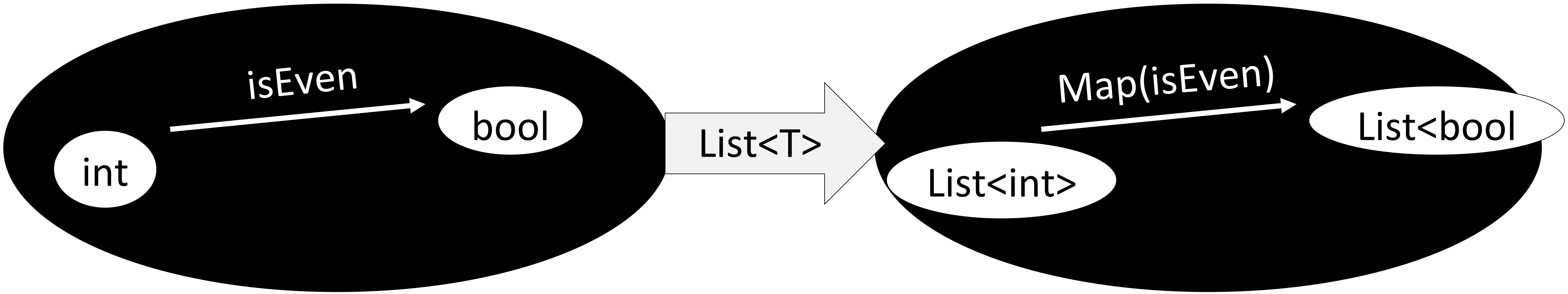
Category of category (cat)



You know category in code

List<T> + map

A functor is a container with a map preserving structure



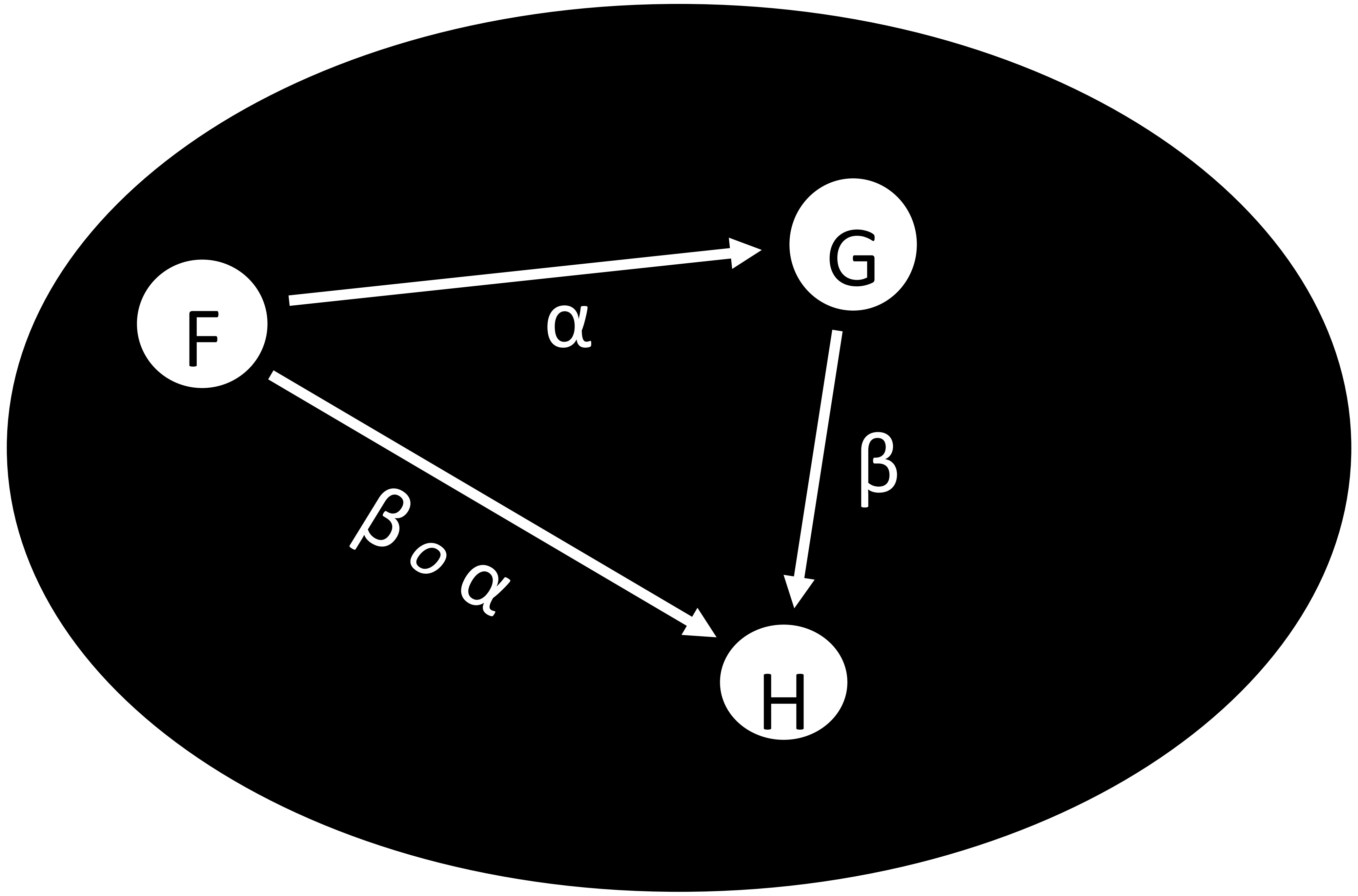
Example ?

List<T>.map(isEven)

$$\text{int} \xrightarrow{\text{isEven}} \text{bool}$$

Select in LINQ = map

Category you already know

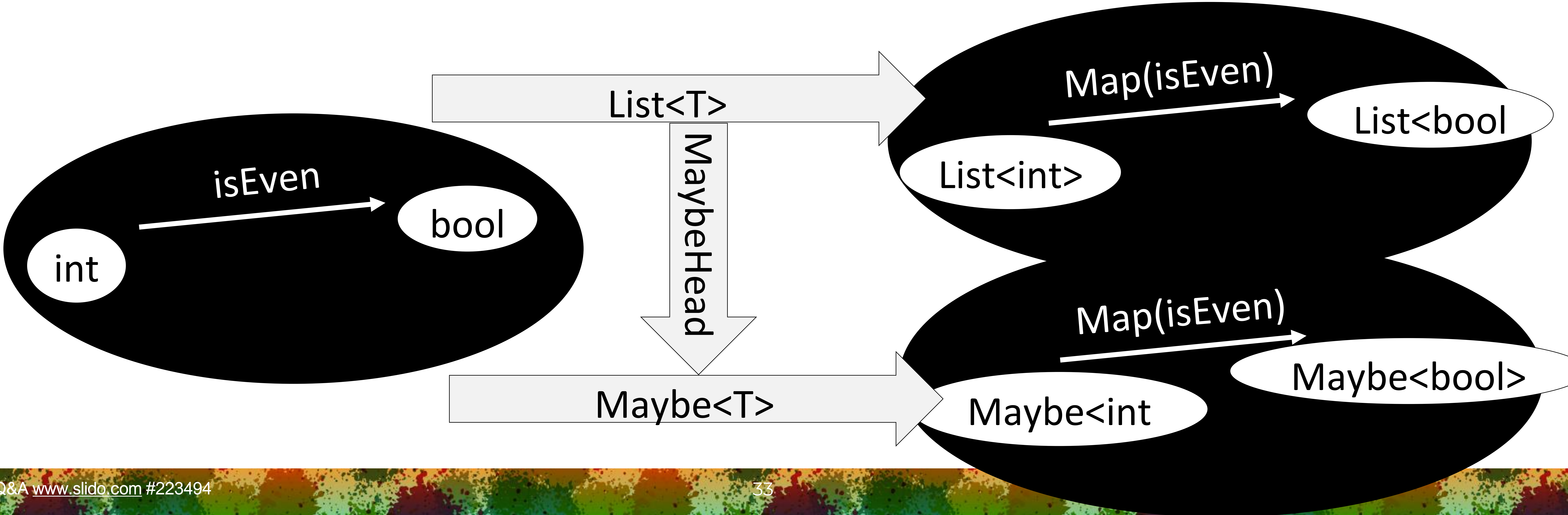


Category of functors

You know category in code

List<T> -> Maybe<T>

A natural transformation is a generic function between two functors



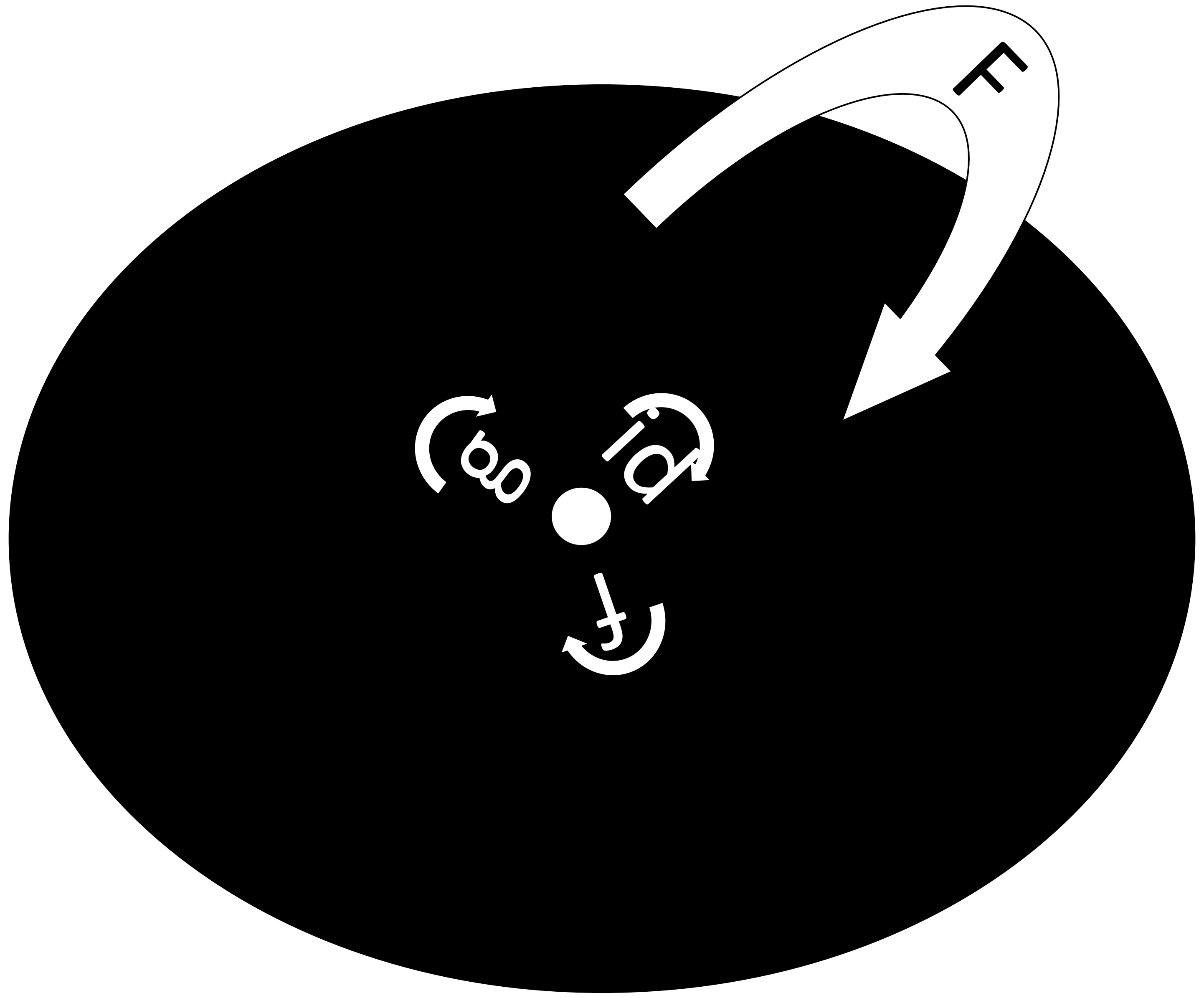
Example ?

$$\text{List<T>} \xrightarrow[\text{MaybeHead}]{} \text{Maybe<T>}$$

```
public static Maybe<T> MaybeHead<T>(List<T> list) =>
    list.Any() ? Maybe<T>.Some(list.First()) : Maybe<T>.None();
```

```
let maybeHead = function
    | [] -> None
    | head::tail -> Some head
```

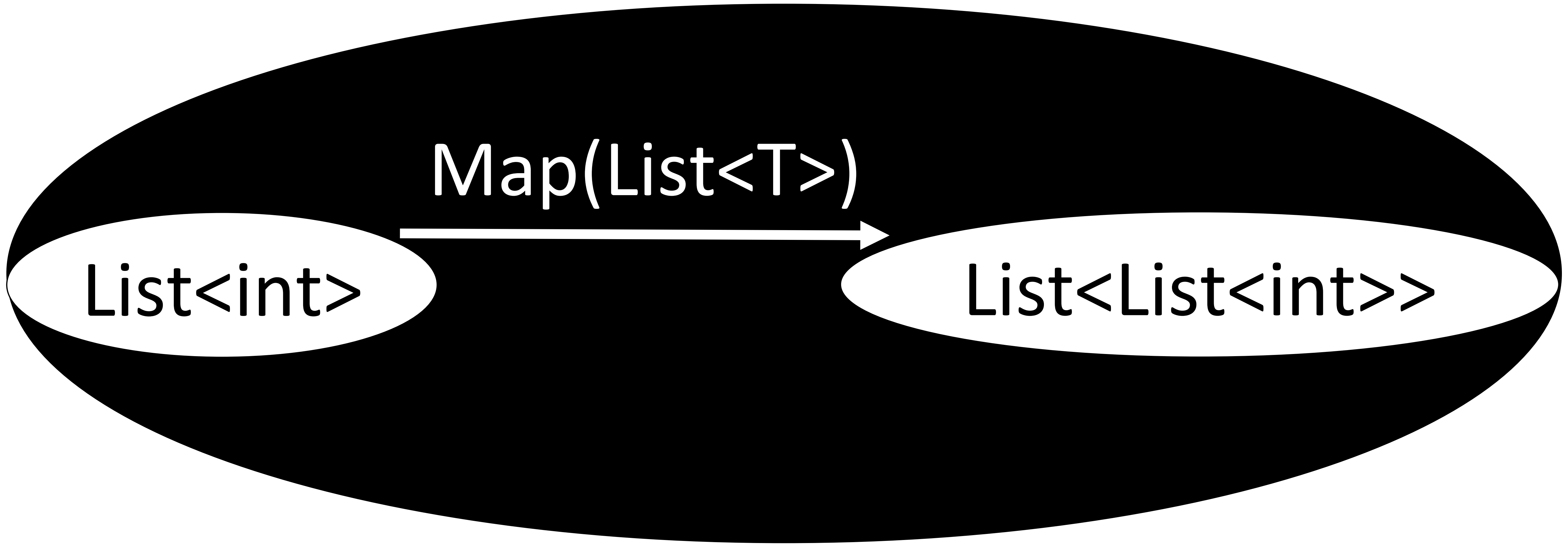
Endofunctor



A functor that map a **category to itself**

Endofunctors are interesting because they do a good job of representing **structures** inside categories that work for **any object**

Endofunctor



An endofunctor can apply to itself

"All the **Functors** we are dealing with in functional **programming** are **Endofunctors**" - https://blog.softwaremill.com/monoid-in-the-category-of-endofunctors-b85bab43587b

Example ?

$$\text{Maybe<T>} \xrightarrow{\text{Map(Maybe<T>)}} \text{Maybe<Maybe<T>>}$$

```
1 usage
static Maybe<TOut> Map<TValue, TOut>(Maybe<TValue> maybe, Func<TValue, TOut> morphism) =>
    maybe.Match(
        someMorphism: value => Maybe<TOut>.Some(morphism(value)),
        Maybe<TOut>.None); // Maybe<TOut>

public static Maybe<Maybe<T>> F<T>(Maybe<T> maybe) => Map(maybe, Maybe<T>.Some);
```

How can I come back ?

Select(List<T>)

List<T>

List<List<T>>

SelectMany(List<List<T>>)

Map(Maybe<T>)

Maybe<T>

Maybe<Maybe<T>>

Join(Maybe<Maybe<T>>)

Monads

A monad is a monoid in the category of endofunctors

Example ?

```
let safeStringIsPositiveInt = safeStringToInt >> bind safeIsPositive >> map intToString
```

How to compose function with **side effects** ?

=> By **encapsulation** in a monad!

No Silver bullet

Mathematical abstraction are « **easy** to build »

Abstraction from real world will be **harder** to build.

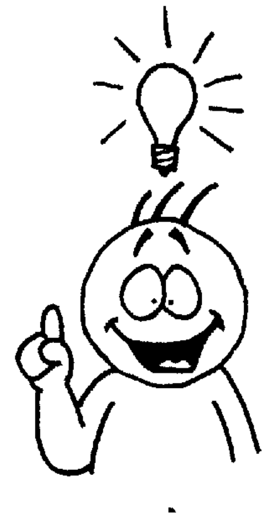
Conclusion

Category theory is **not mandatory** to code...

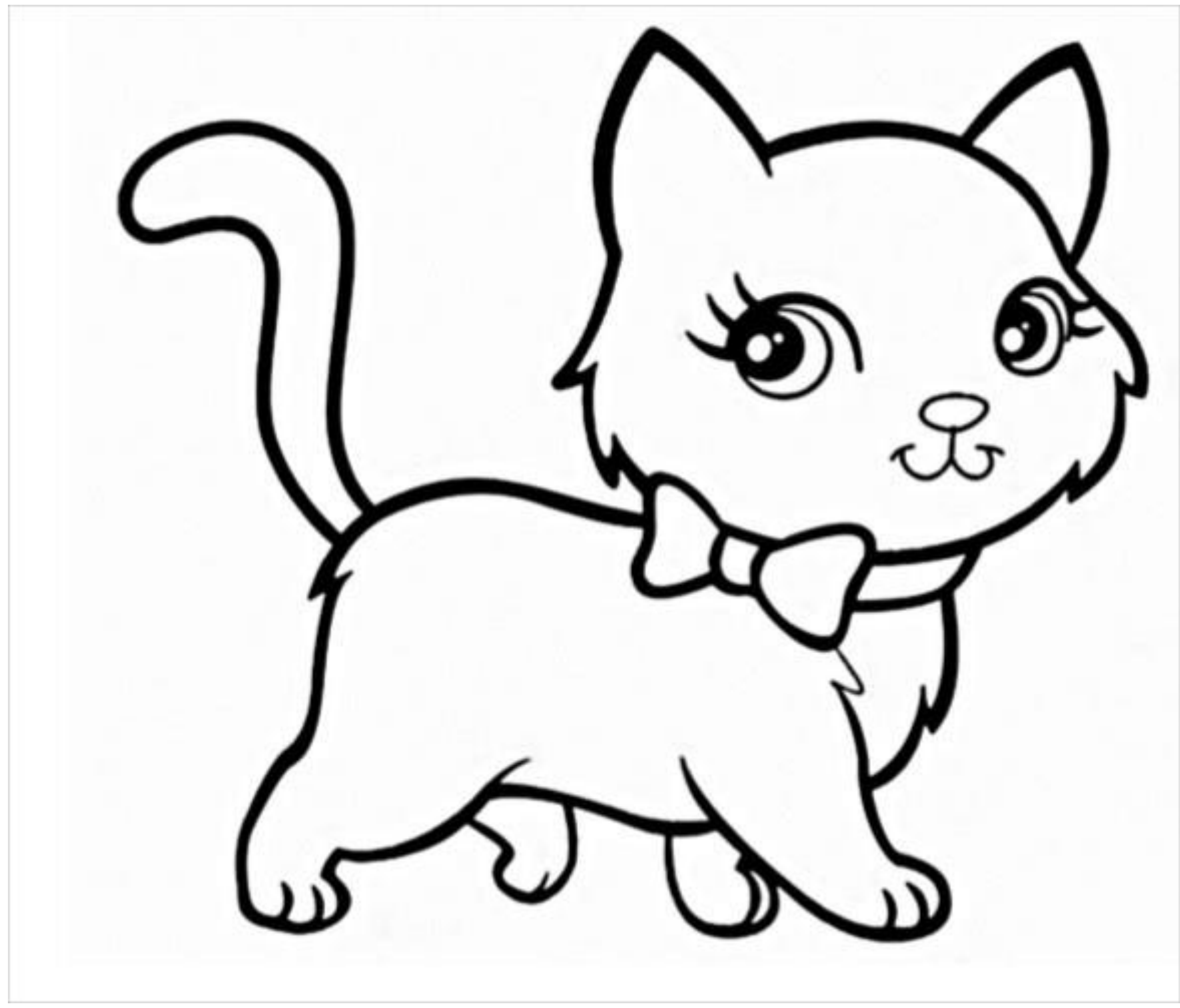
..But it could help to find **clever solution** to complex programming problems.

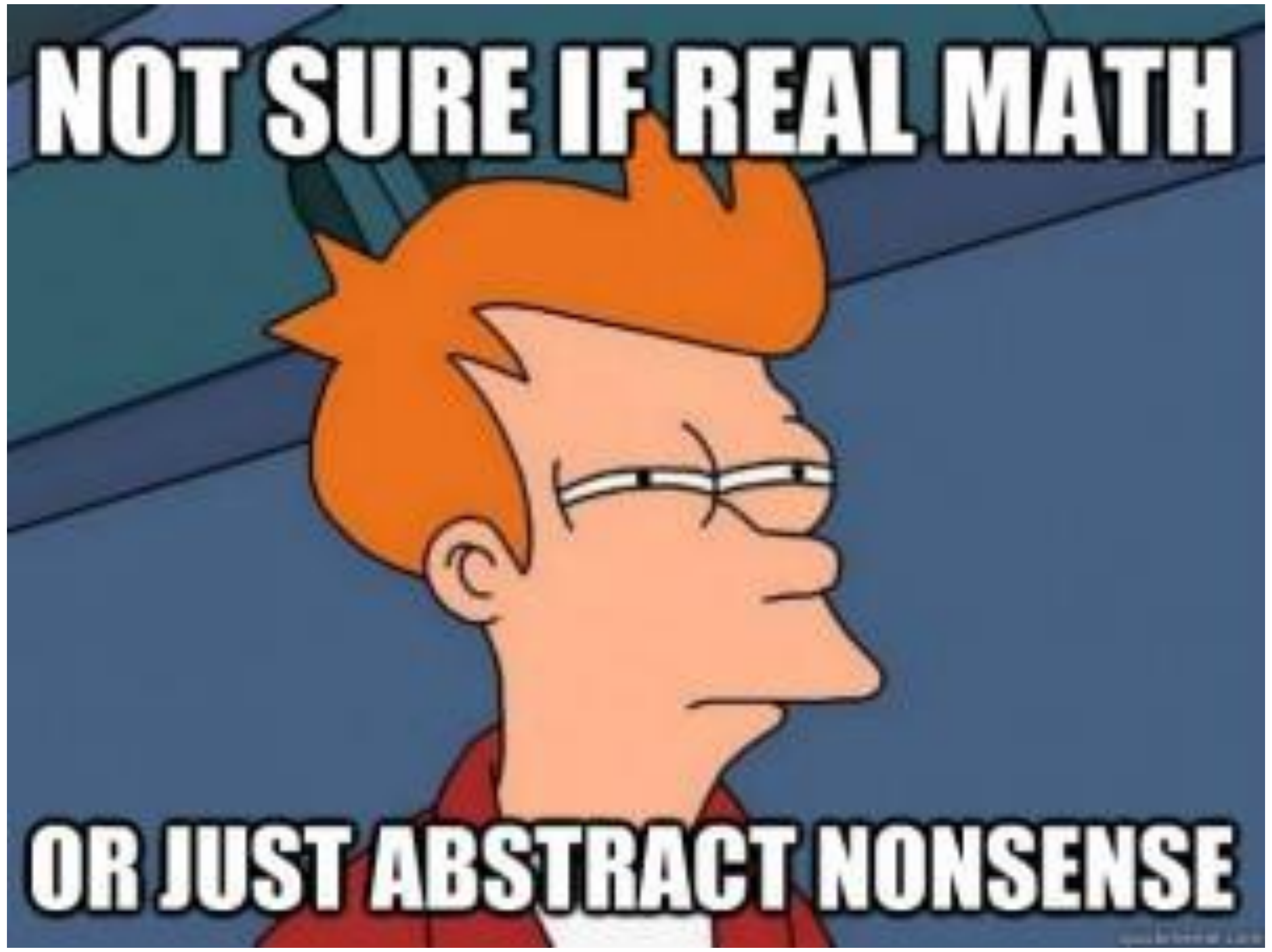
[Category theory] does not itself solve hard problems [...] It puts the hard problems in clear relief and makes their solution possible.

—*The Last Mathematician (Hilbert Gottingen)*



Thanks





References

- Category Theory for beginners (Ken Scambler)
- Category Theory for programmers (Bartosz Milewski)
- Philip Wadler 's Blog: http://homepages.inf.ed.ac.uk/wadler/
- Robb Seaton's Blog: http://rs.io/why-category-theory-matters/
- Category Theory for the working mathematicians (MacLane, Saunders)

History

- 1940s: Einlenberg and MacLane formalize Category Theory
- 1958: Monads discovered by Godement
- 1990: Moggi, Wadler apply monads to functional programming

You know category in math

Product

$$A^C * B^C = (A * B)^C$$

$$(C \to A\ ,\ C \to B) = C \to (A,B)$$

$$\begin{tikzcd}
 & c \arrow[dl, "f"'] \arrow[d, "\langle f,g \rangle"'] \arrow[dr, "g"] & \\
a & (a,b) \arrow[l, "first"] \arrow[r, "second"'] & b
\end{tikzcd}$$

You know category in math

Co Product

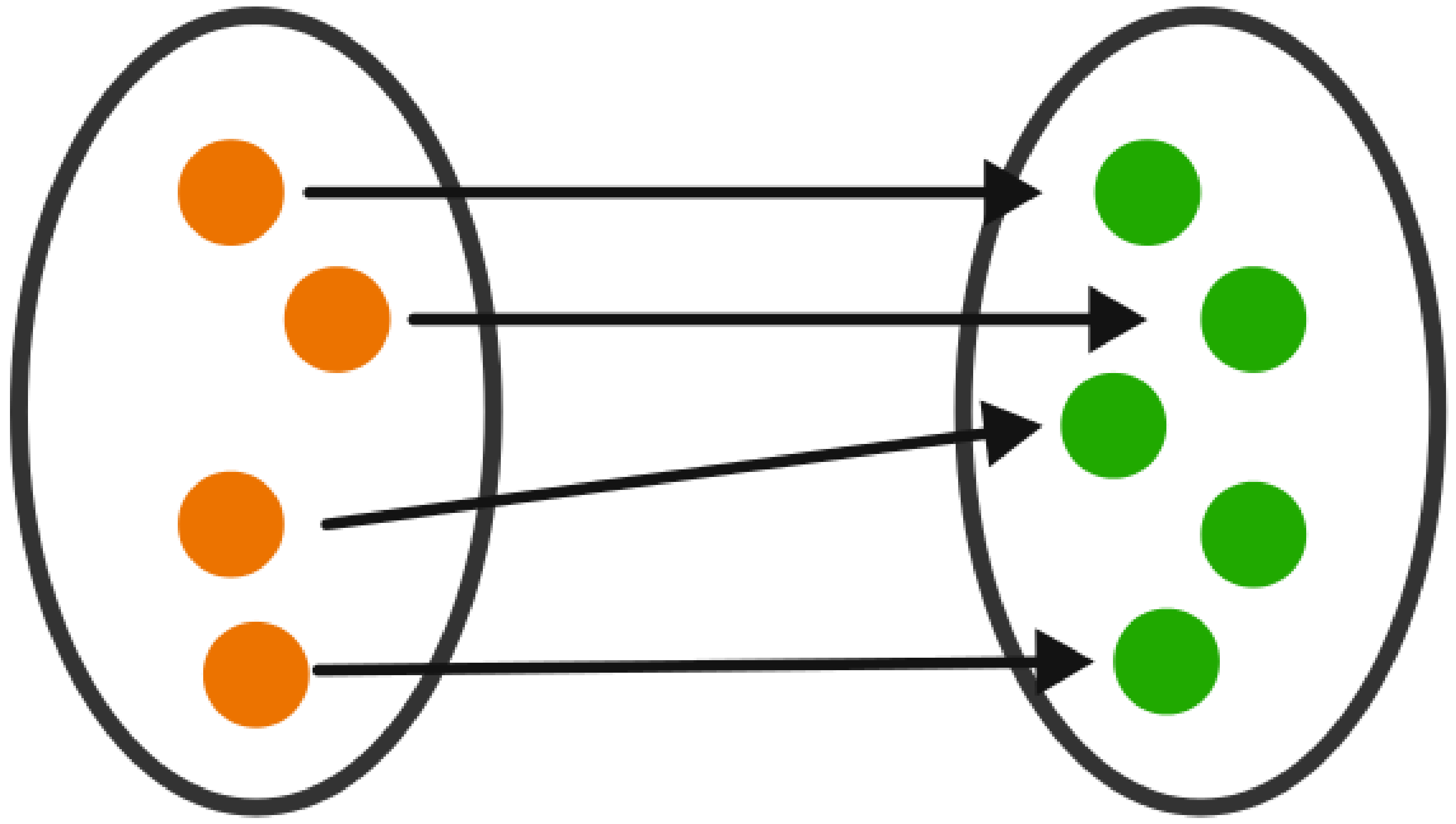
$$(A + B) * C = A * C + B * C$$

$$(A|B , C) = (A,C) \mid (B,C)$$

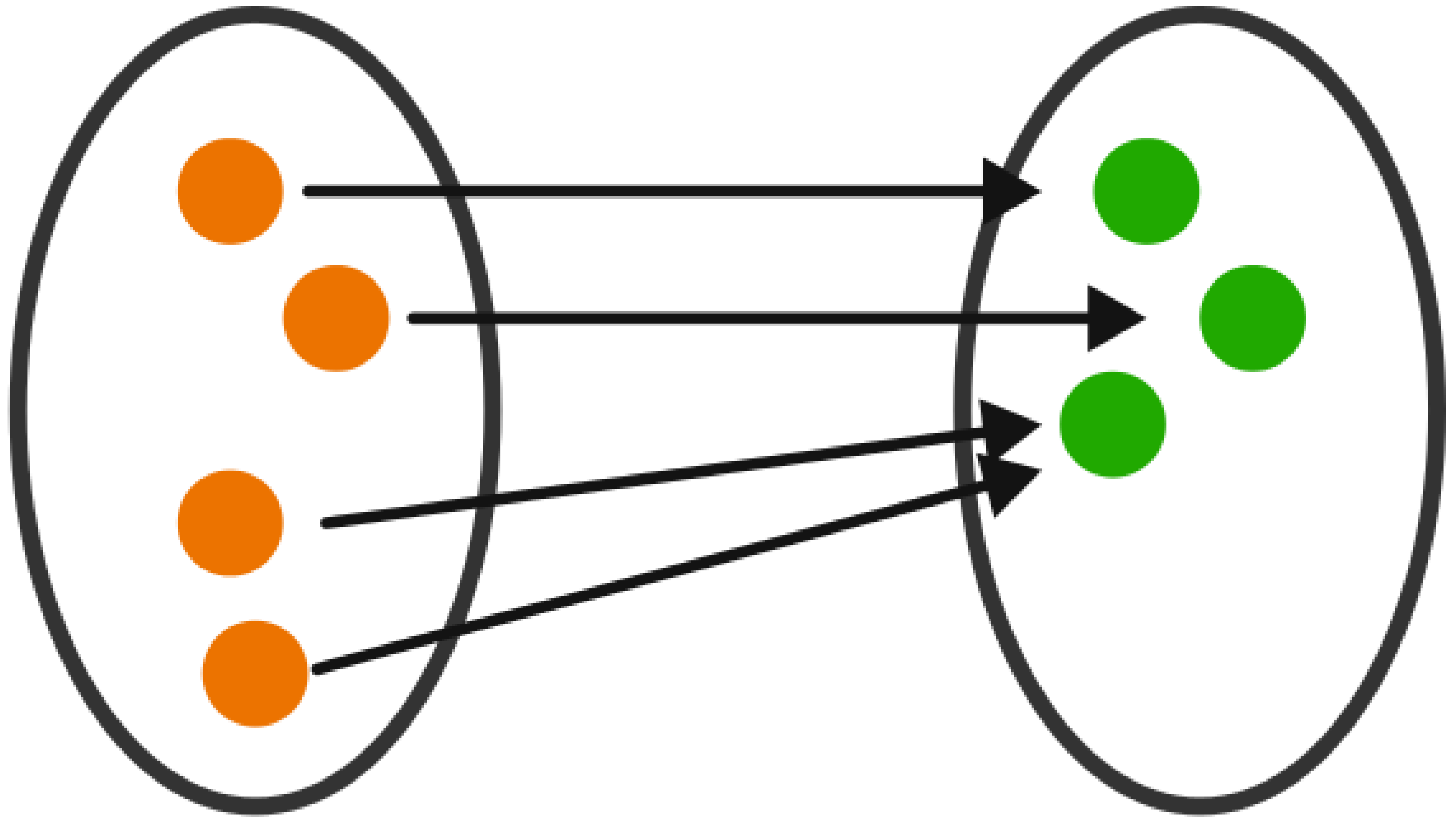
$$\begin{tikzcd}
a \arrow[r, "ind"] \arrow[dr, "f"'] & a|b \arrow[d, "{[f,g]}"] & b \arrow[l, "inr"'] \arrow[dl, "g"] \\
& c &
\end{tikzcd}$$

Surjective/Bijective

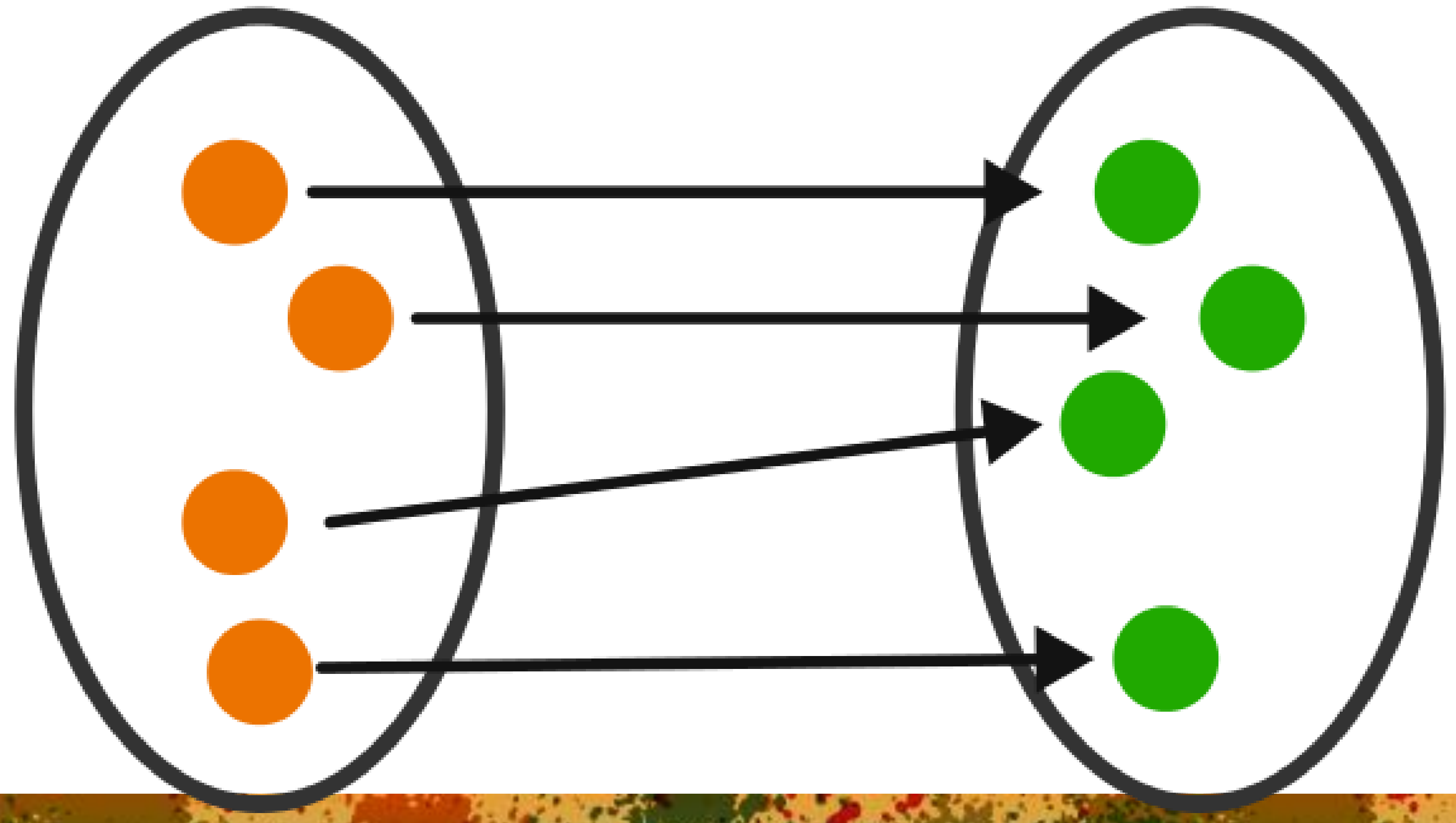
Injection (One-to-One)



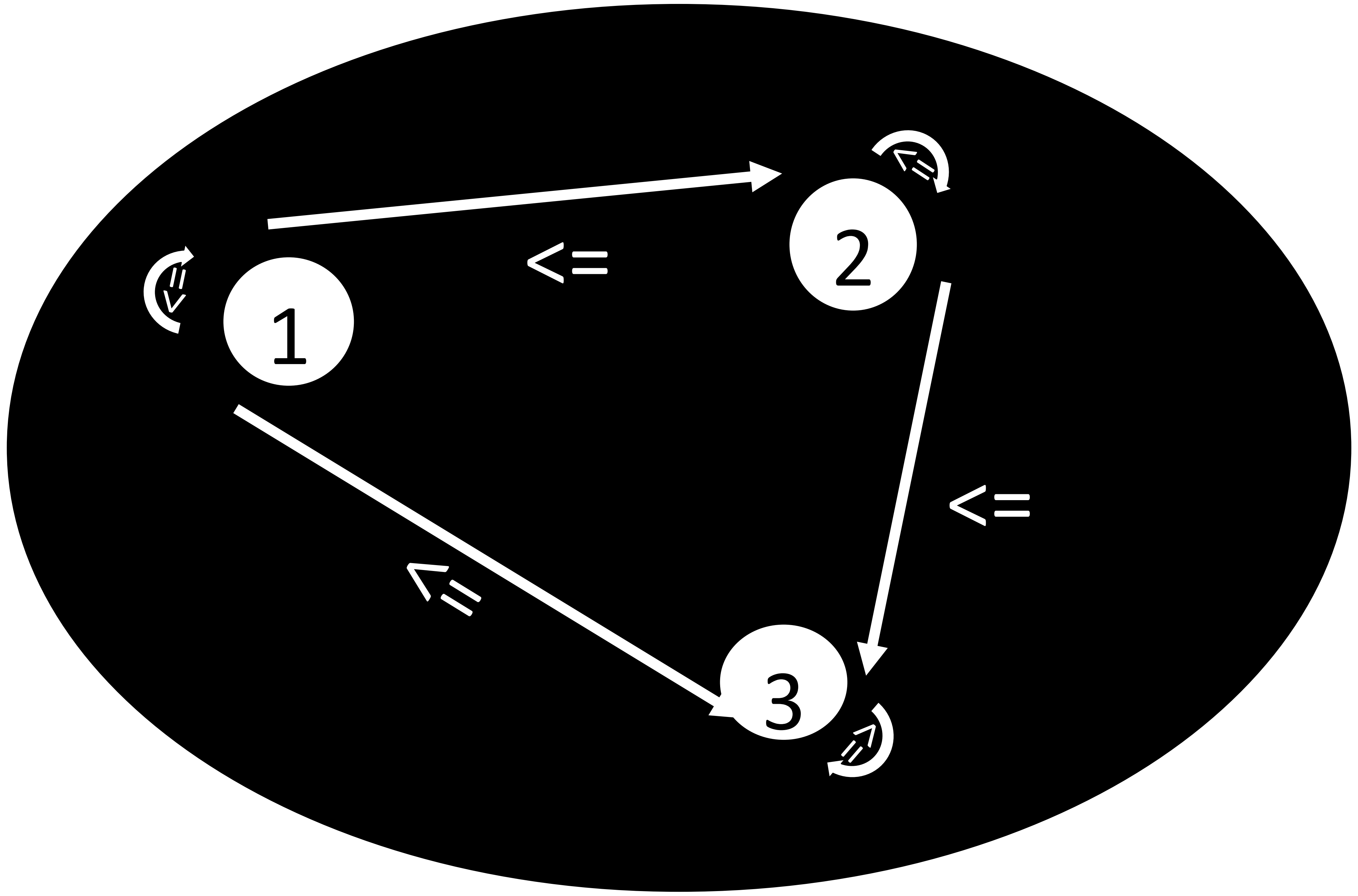
Surjection (Onto)



Bijection (One-to-One and Onto)

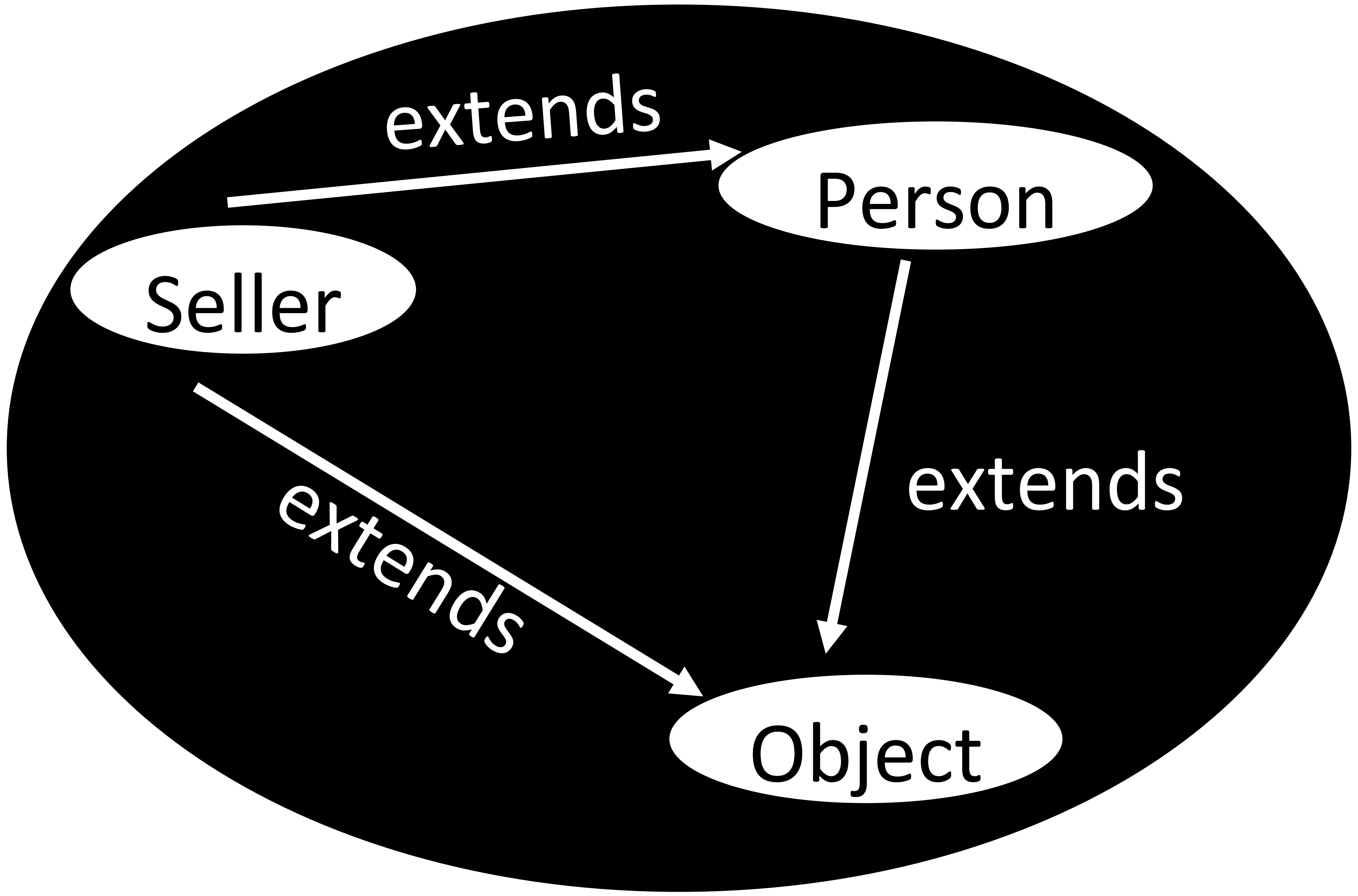


Category you already know



Ordered set

Category you already know



Class hierarchy