mediarithmics

Recursion Schemes

devs@mediarithmics.com

mediarithmics

Our Speakers

Nicolas François
Software Engineer
nfrancois@mediarithmics.com
mediarithmics

Virgile Quintin
Software Engineer
vquintin@mediarithmics.com
mediarithmics

mediarithmics

Contents

mediarithmics

Why recursion scheme?

- Recursive functions can be hard to maintain over time (as any complex control flow).
- Explicit recursion can be hard to test
- Guarantee to finish (if you want too)

mediarithmics

What are recursion schemes?

Recursion schemes: work with recursive data structures without explicit recursion.

More generic, more readable, more testable

One more tool to help manage complex code

mediarithmics

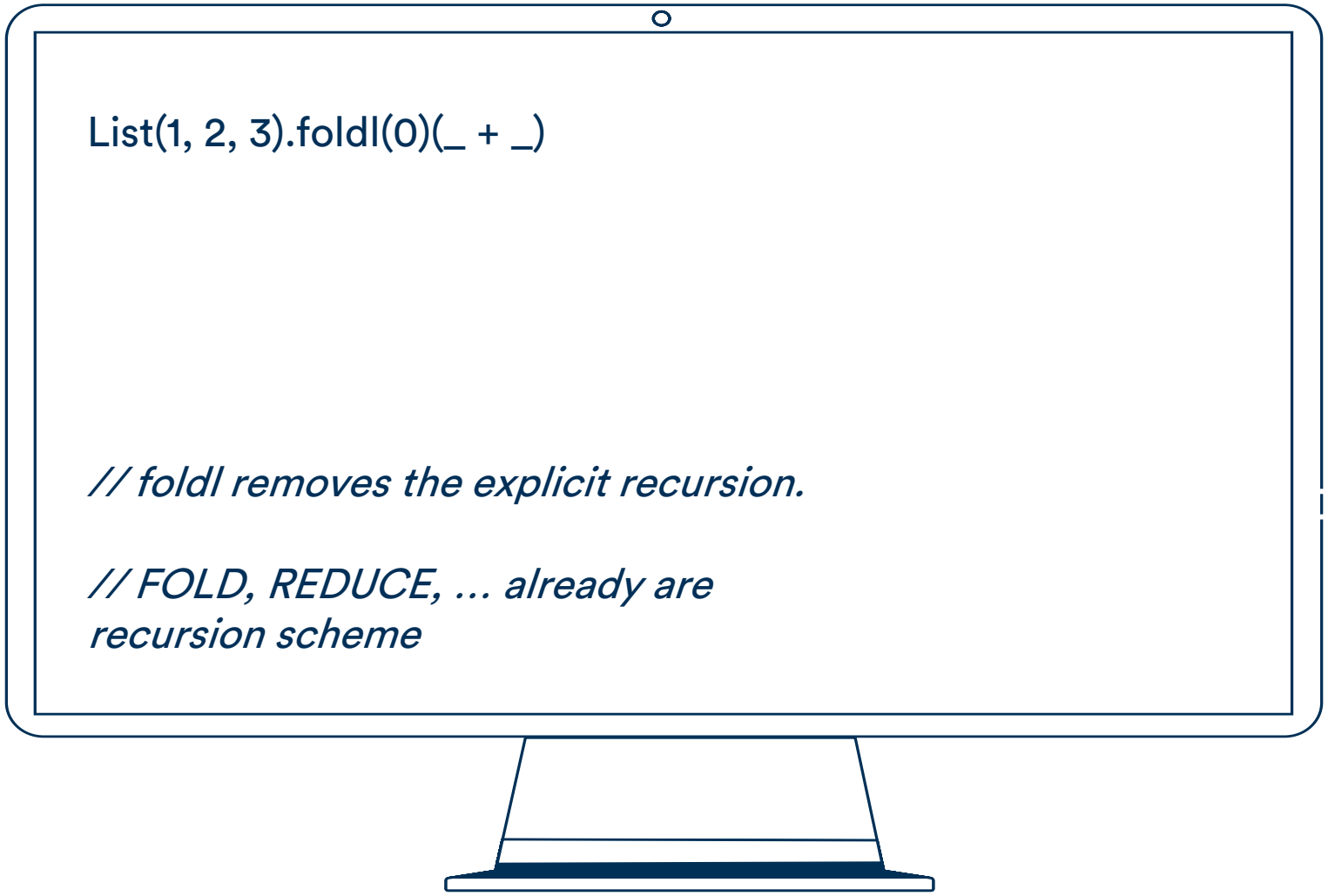
What are recursion schemes?

Bonus:
It allows us to easily
write performant stack safe code !*

*offer subject to conditions

mediarithmics

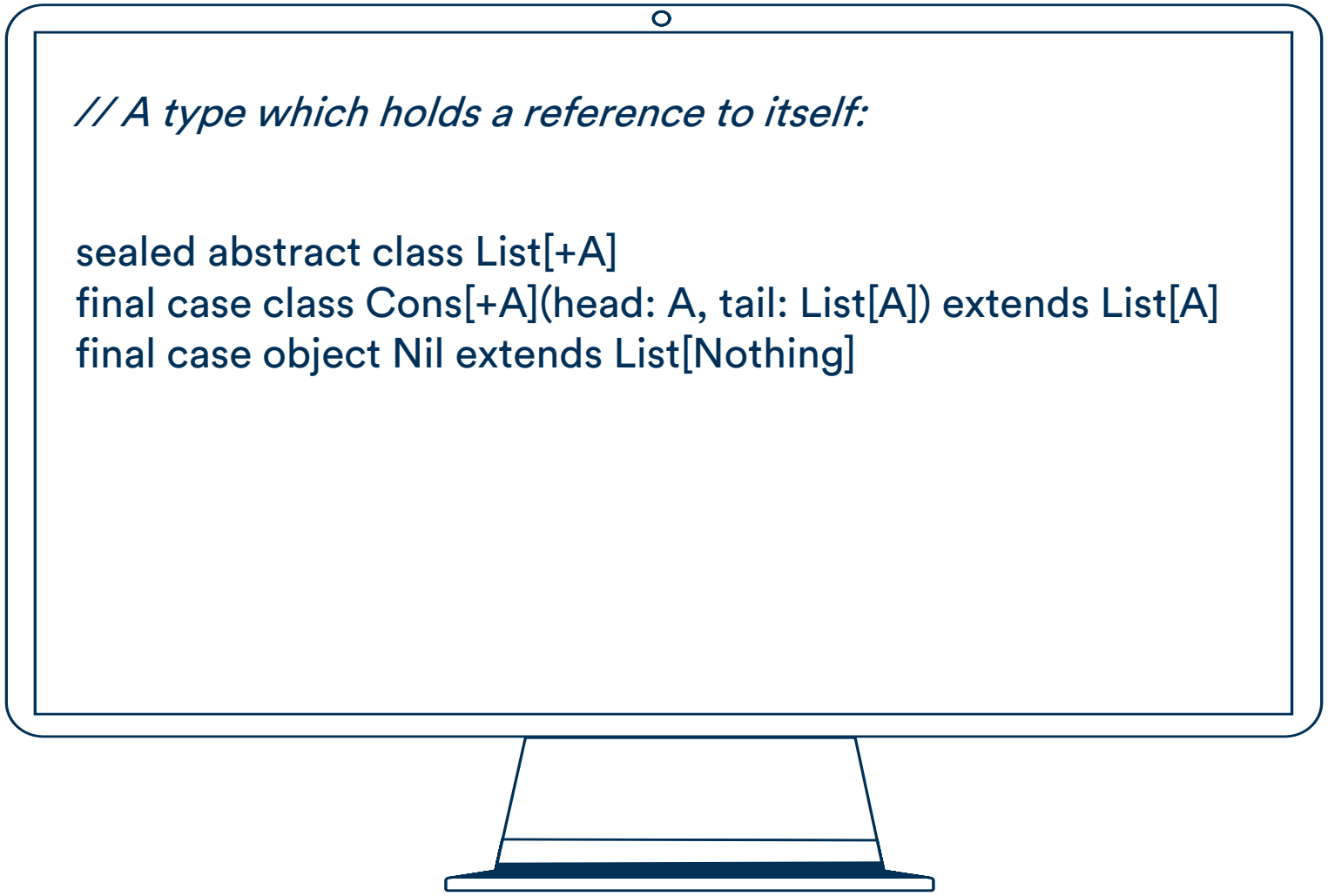
Already known technique



Quick Definitions

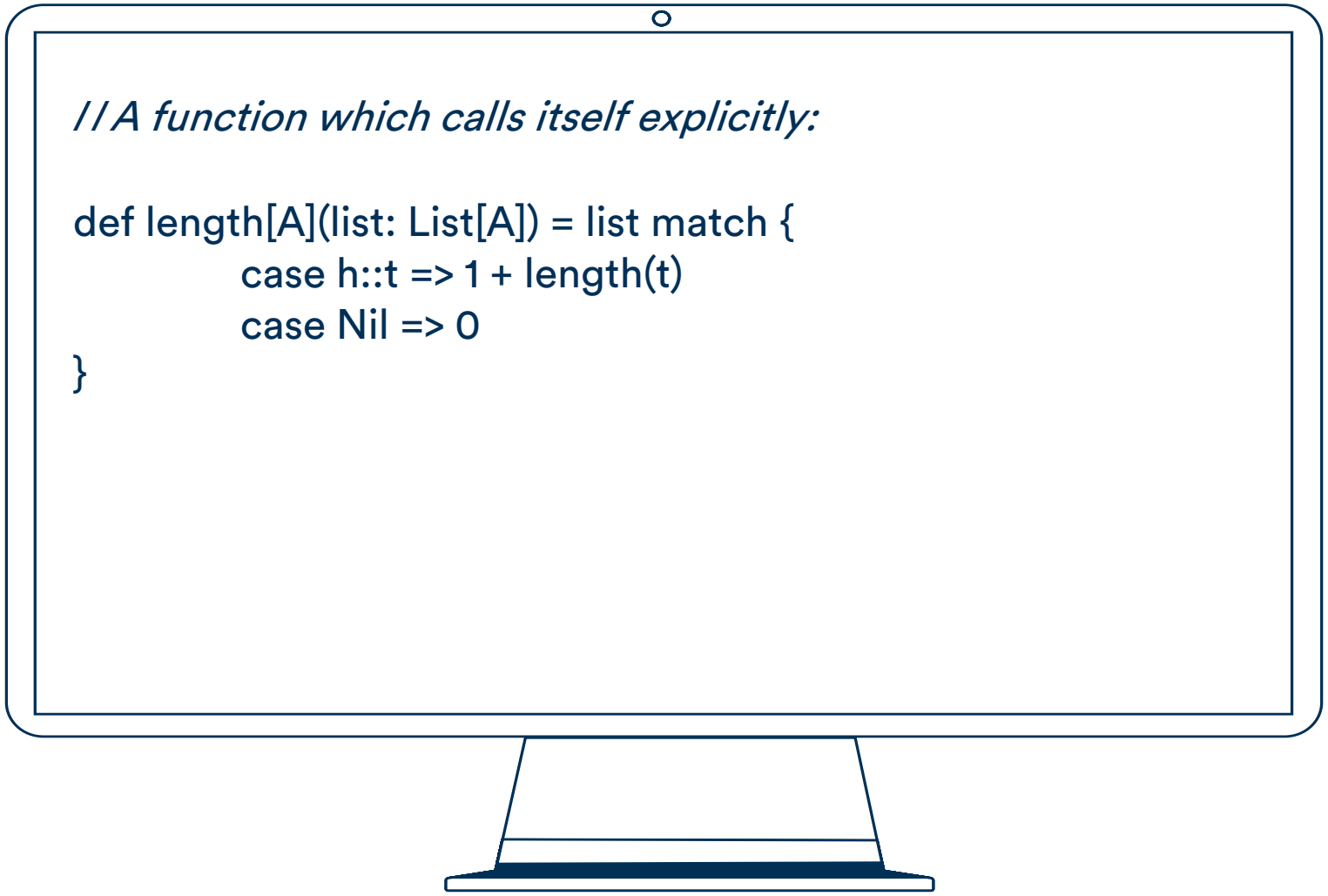
mediarithmics

Recursive data structures



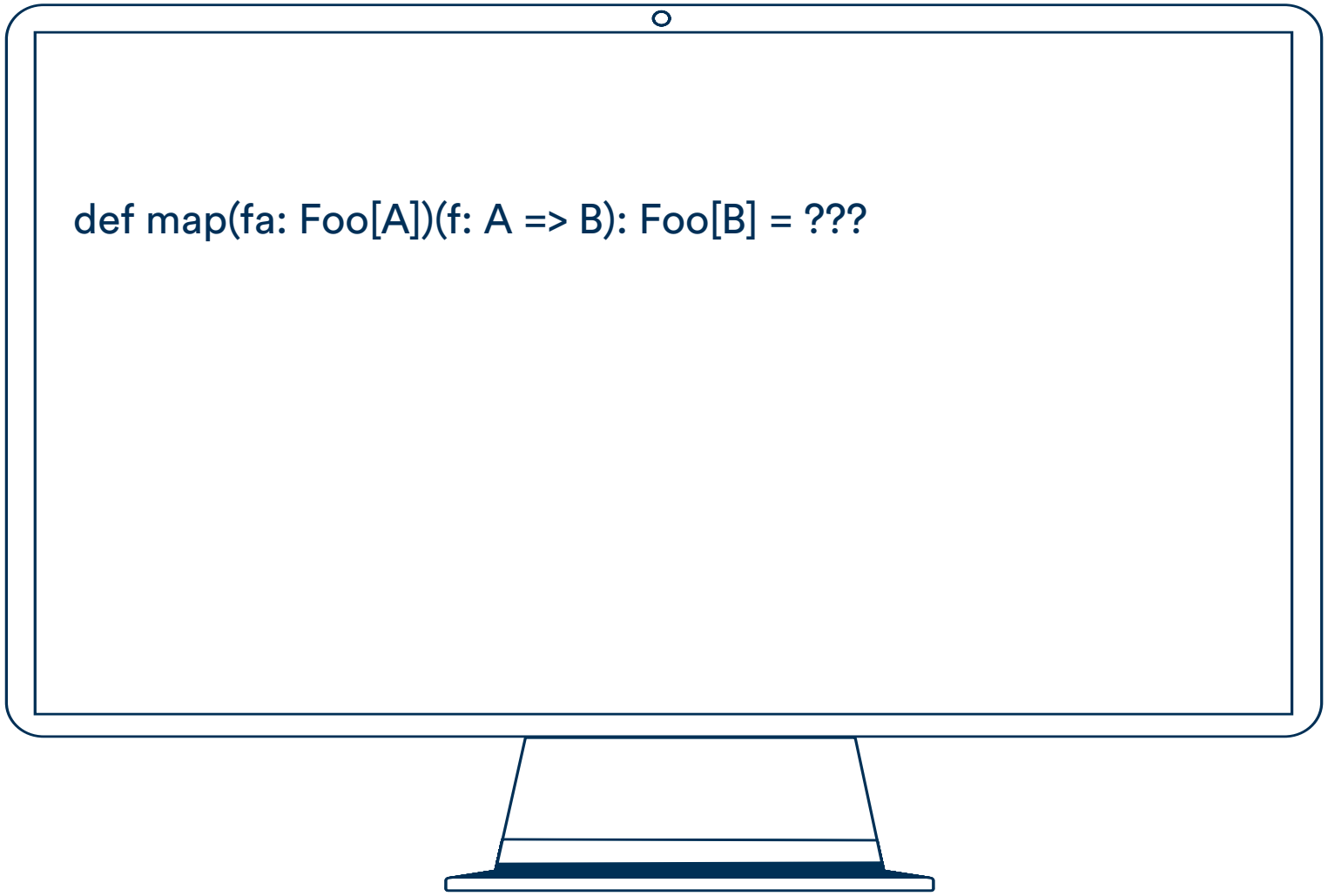
mediarithmics

Explicit recursion



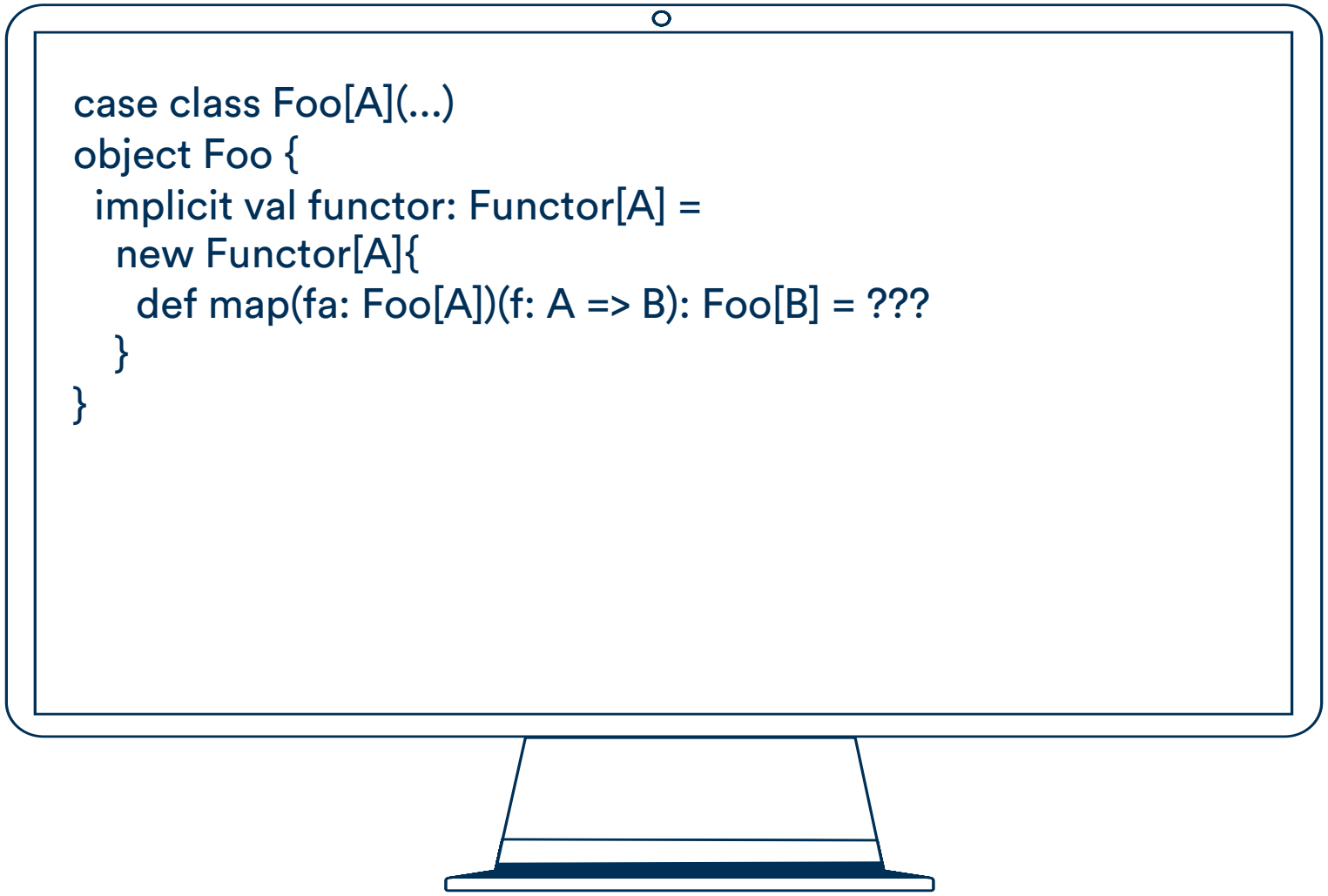
mediarithmics

Functor



mediarithmics

Functor



mediarithmics

Separating concern

A recursive data structure has two things:

- A shape: list, tree
- Recursiveness

mediarithmics

Separating concern

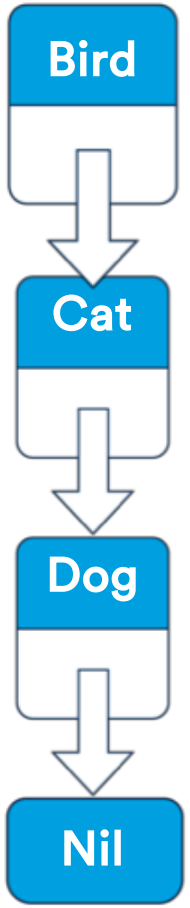
A recursive data structure has two things:

- A shape: list, tree → Pattern functor
- Recursiveness → Fix

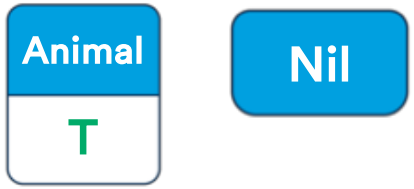
Pattern functor and Fix help separating the concerns in the recursive data structure definition.

mediarithmics

Pattern Functor

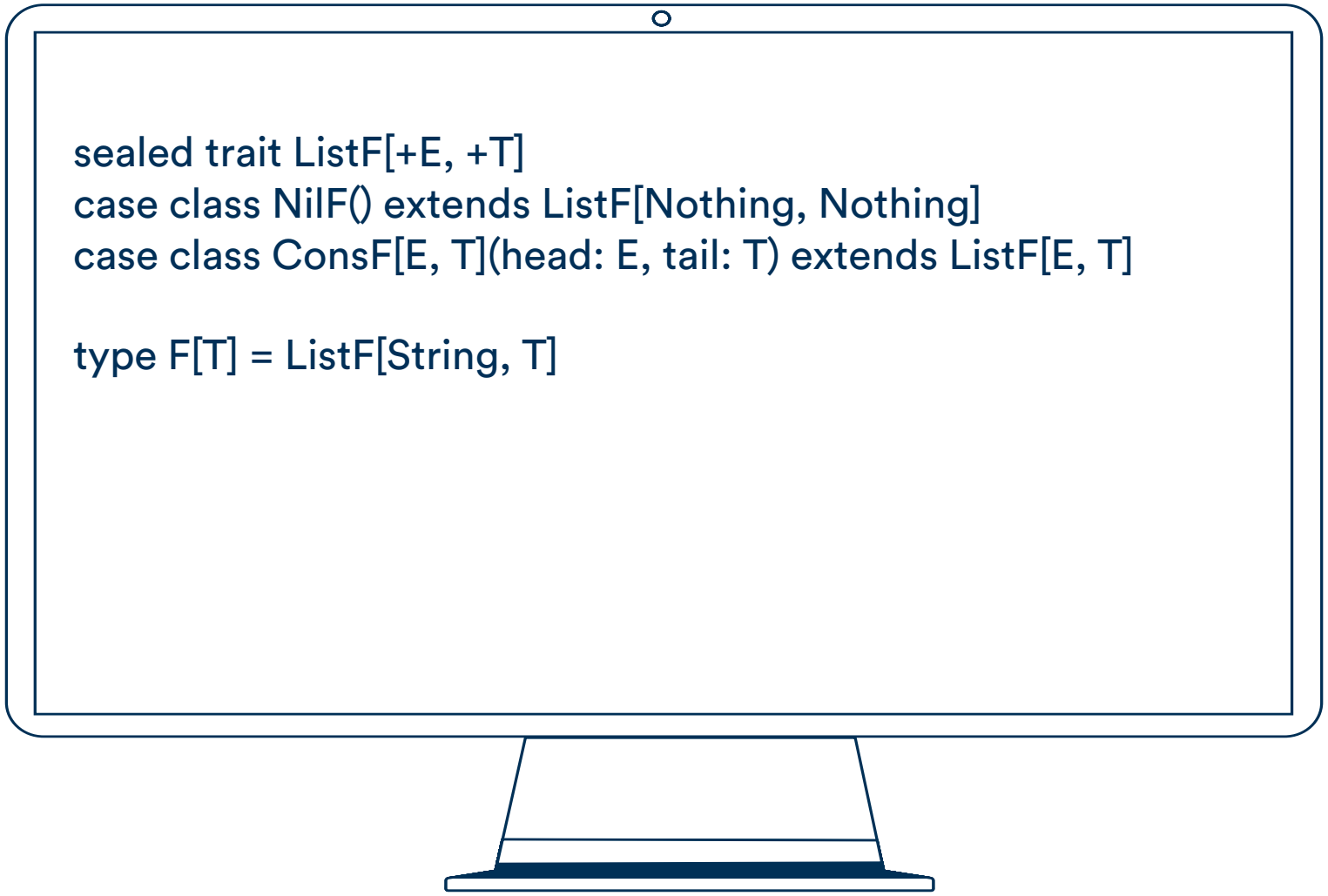


F[T]



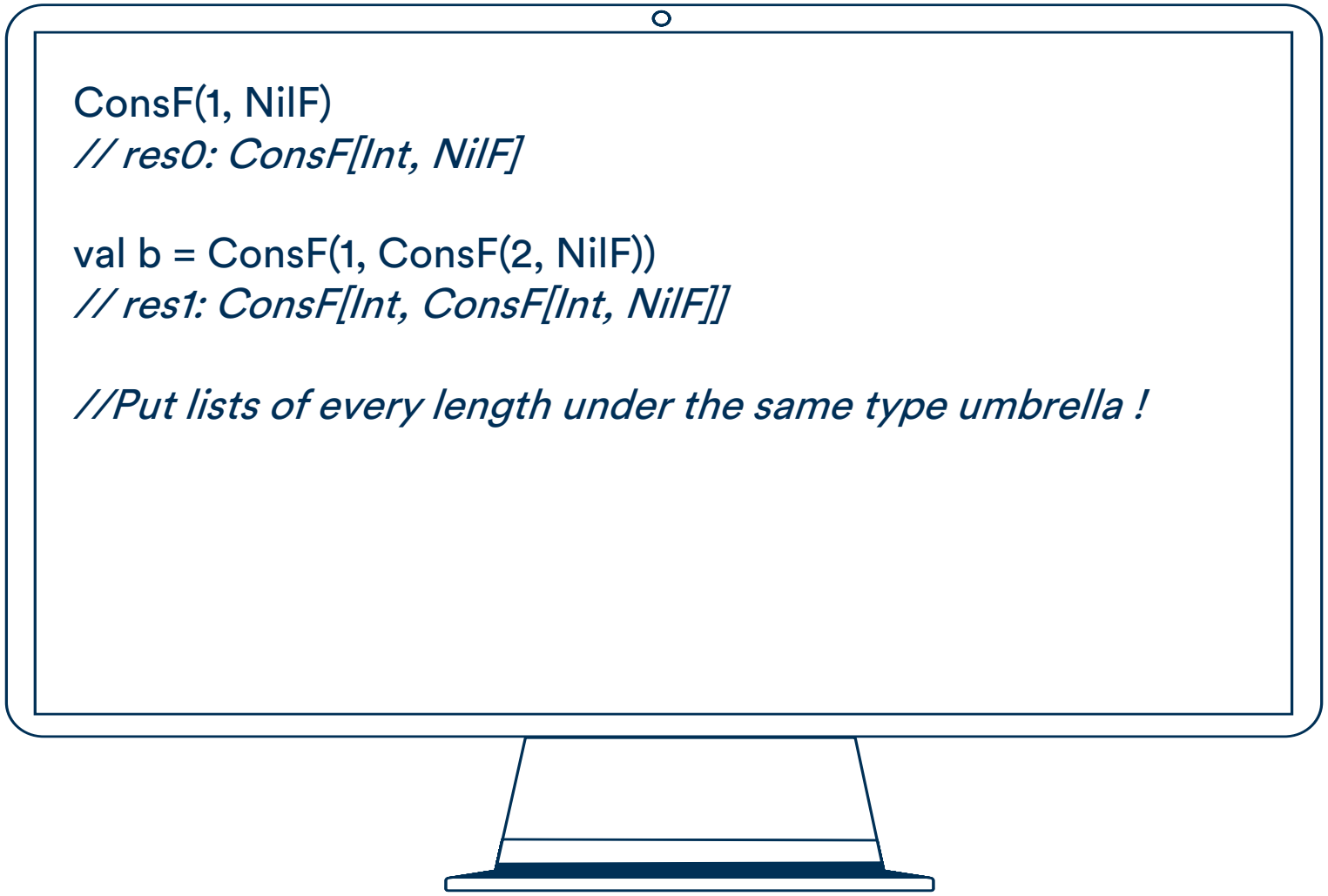
mediarithmics

Pattern Functor



mediarithmics

Fix: type problems



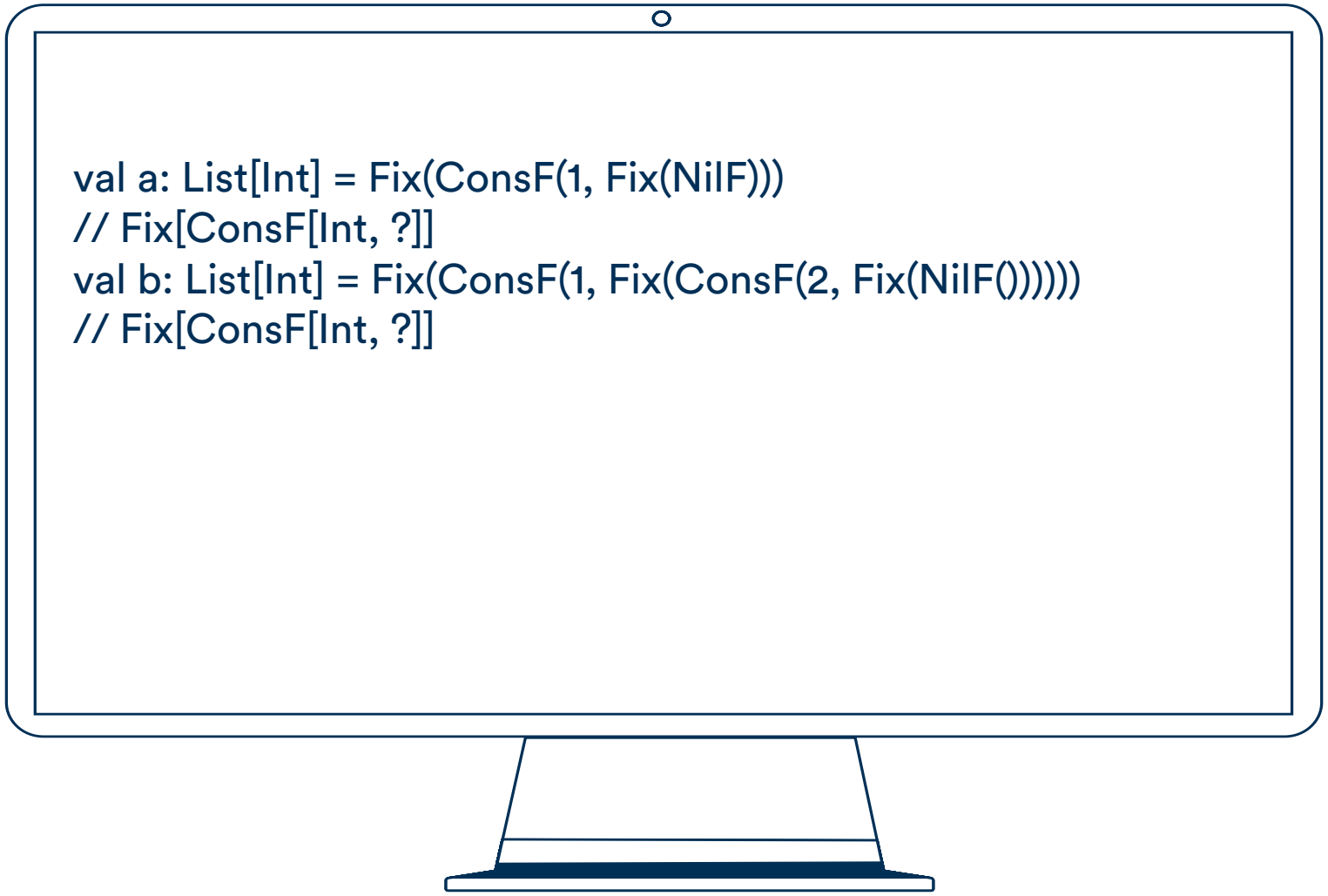
mediarithmics

Fix: the trick

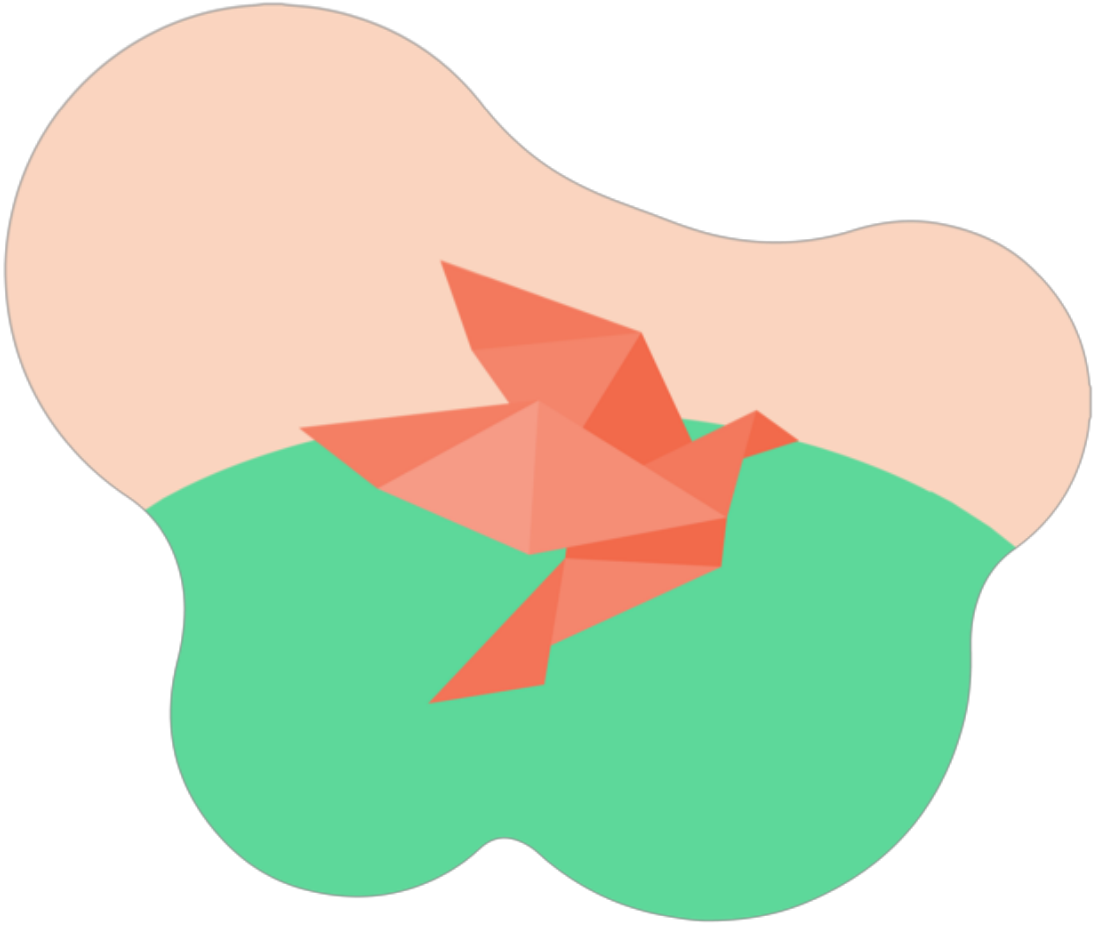
```
case class Fix[F[_]](unFix: F[Fix[F]])

type List[A] = Fix[List[A, ?]]
```

mediarithmics

Fix



Types of recursion schemes



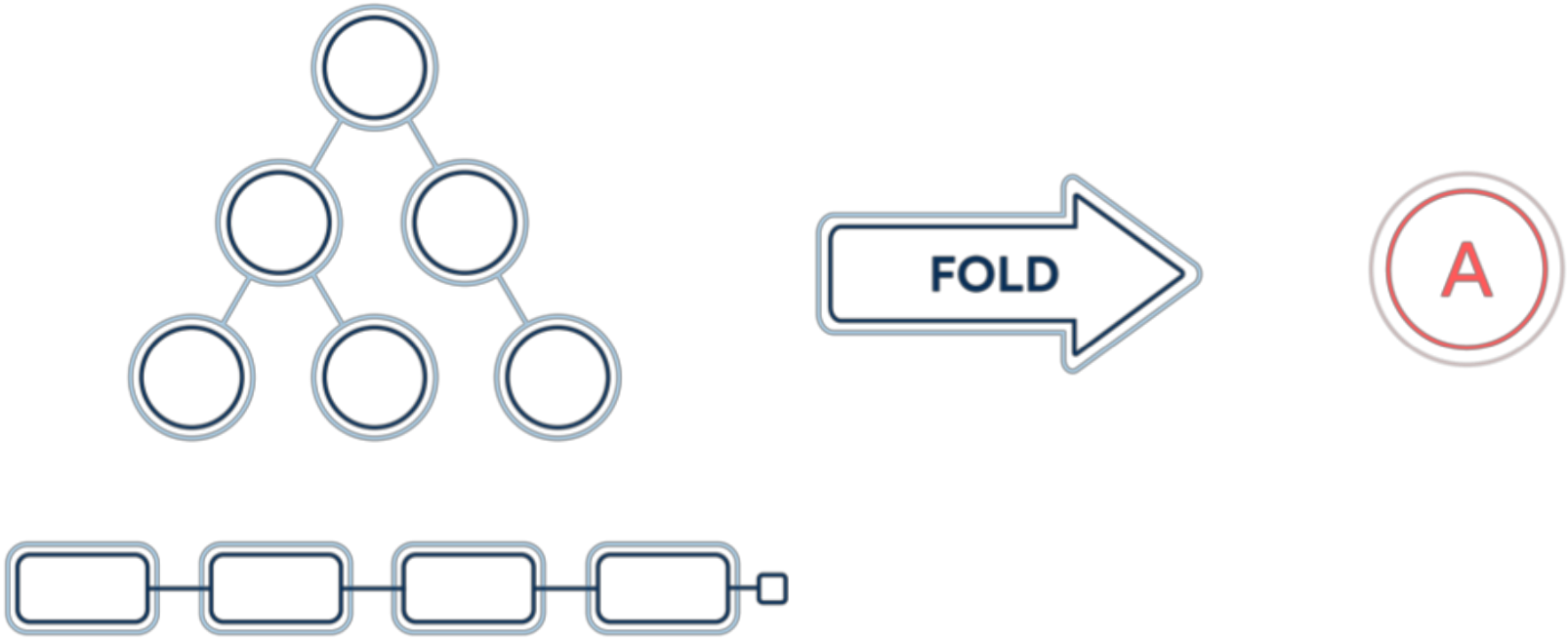
mediarithmics

Types of recursion schemes

- Folds
- Unfolds
- Refolds

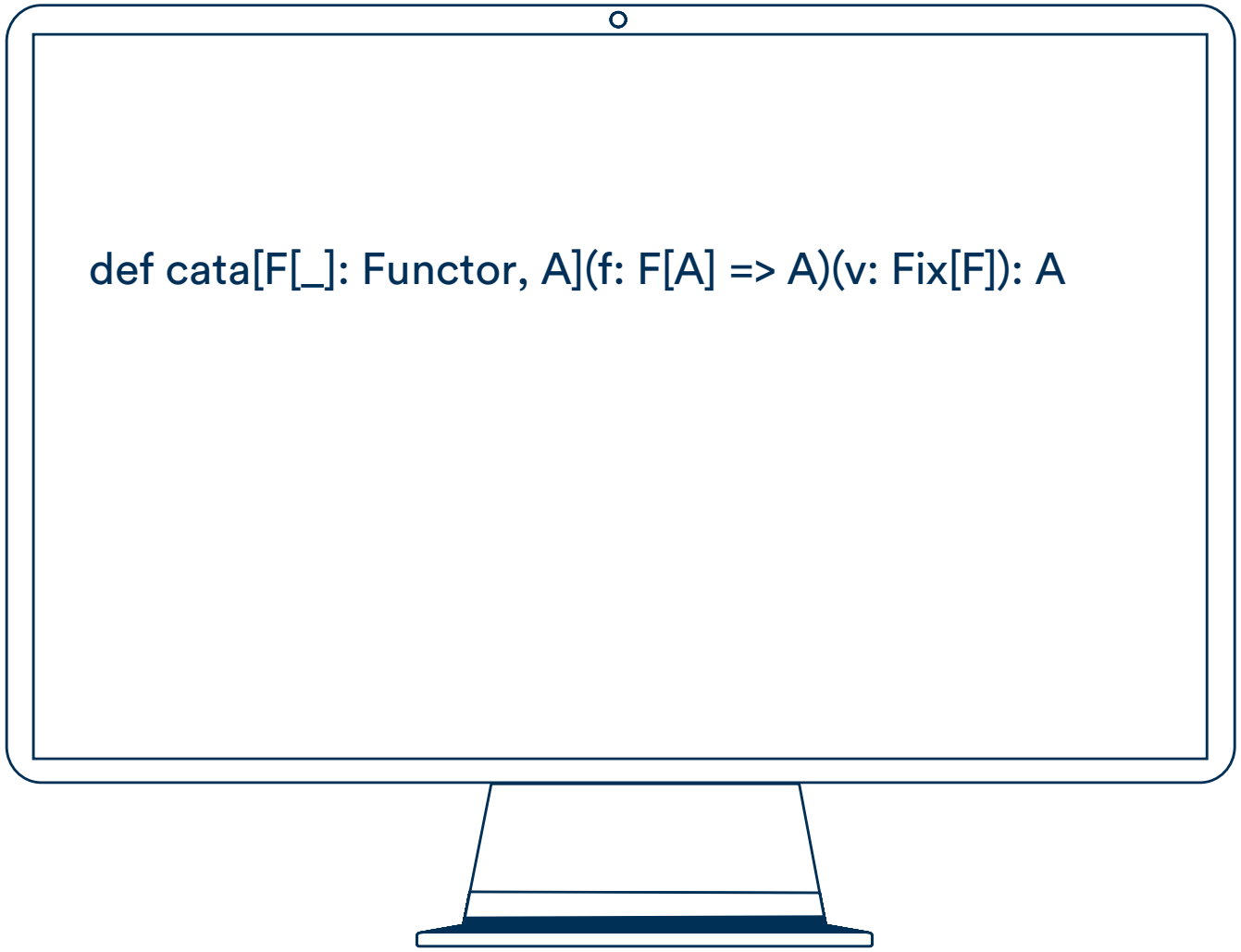
mediarithmics

Fold



mediarithmics

Folds: catamorphism

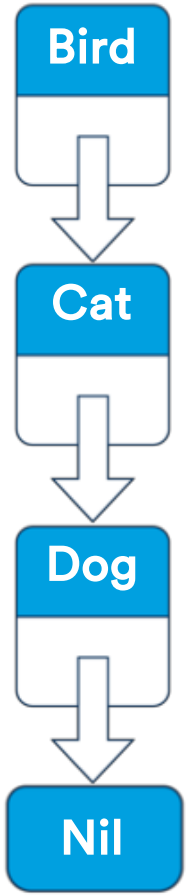


- One level at a time
- From greek κατα as in catastrophe, catalyst, catapult

mediarithmics

Example: length of a list of String

```
def cata[F: Functor, A](f: F[A] => A)(c: Fix[F]): A = ???
cata[ListF[String, ?]], Int]{
 case ConsF(_, previousLength) => 1 + previousLength
 case NilF() => 0
}(list)
```



mediarithmics

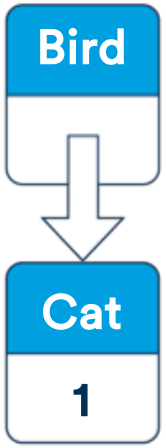
Example: length of a list of String

```
def cata[F: Functor, A](f: F[A] => A)(c: Fix[F]): A = ???
cata[ListF[String, ?]], Int]{
 case ConsF(_, previousLength) => 1 + previousLength
 case NilF() => 0
}(list)
```

mediarithmics

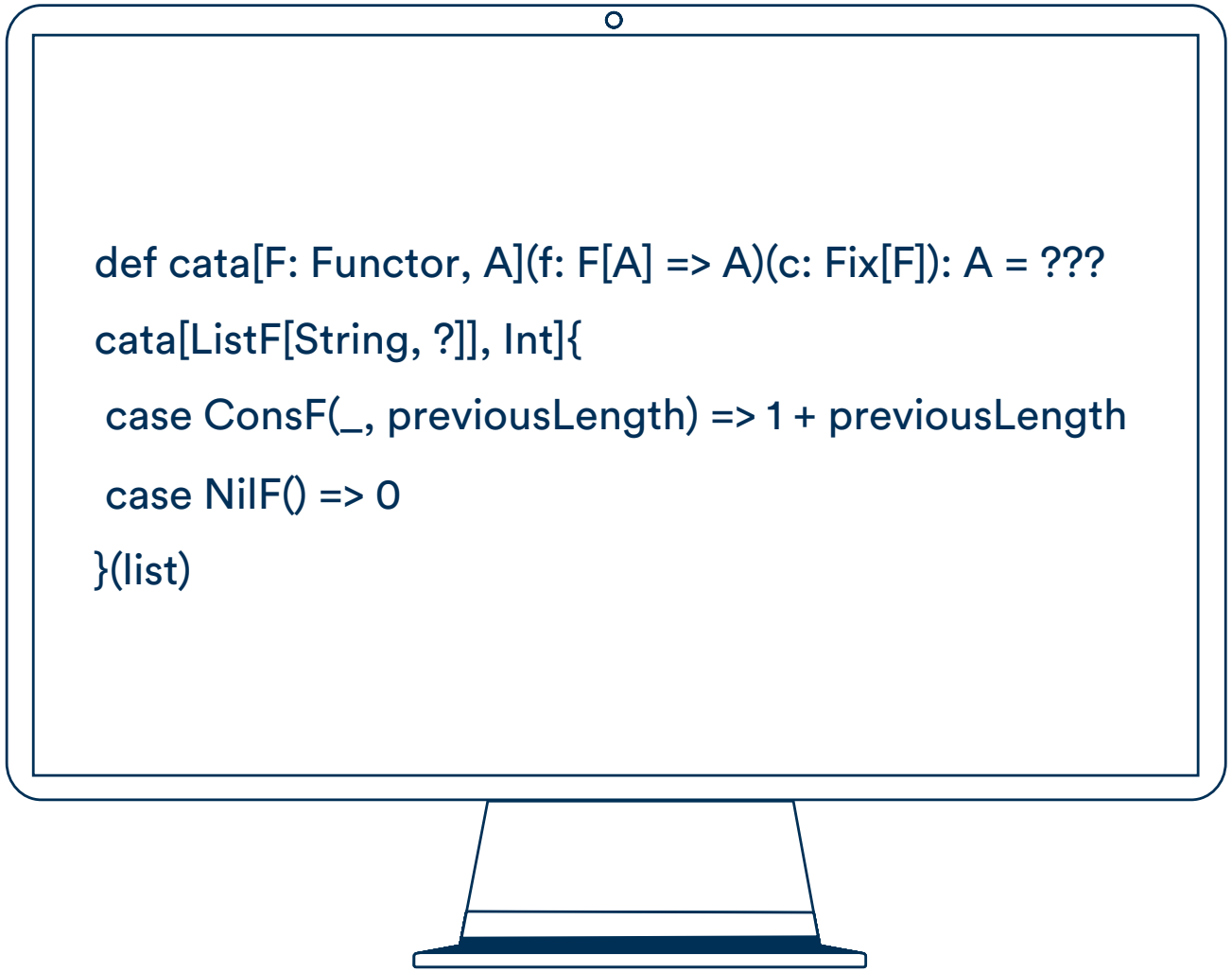
Example: length of a list of String

```
def cata[F: Functor, A](f: F[A] => A)(c: Fix[F]): A = ???
cata[ListF[String, ?]], Int]{
 case ConsF(_, previousLength) => 1 + previousLength
 case NilF() => 0
}(list)
```



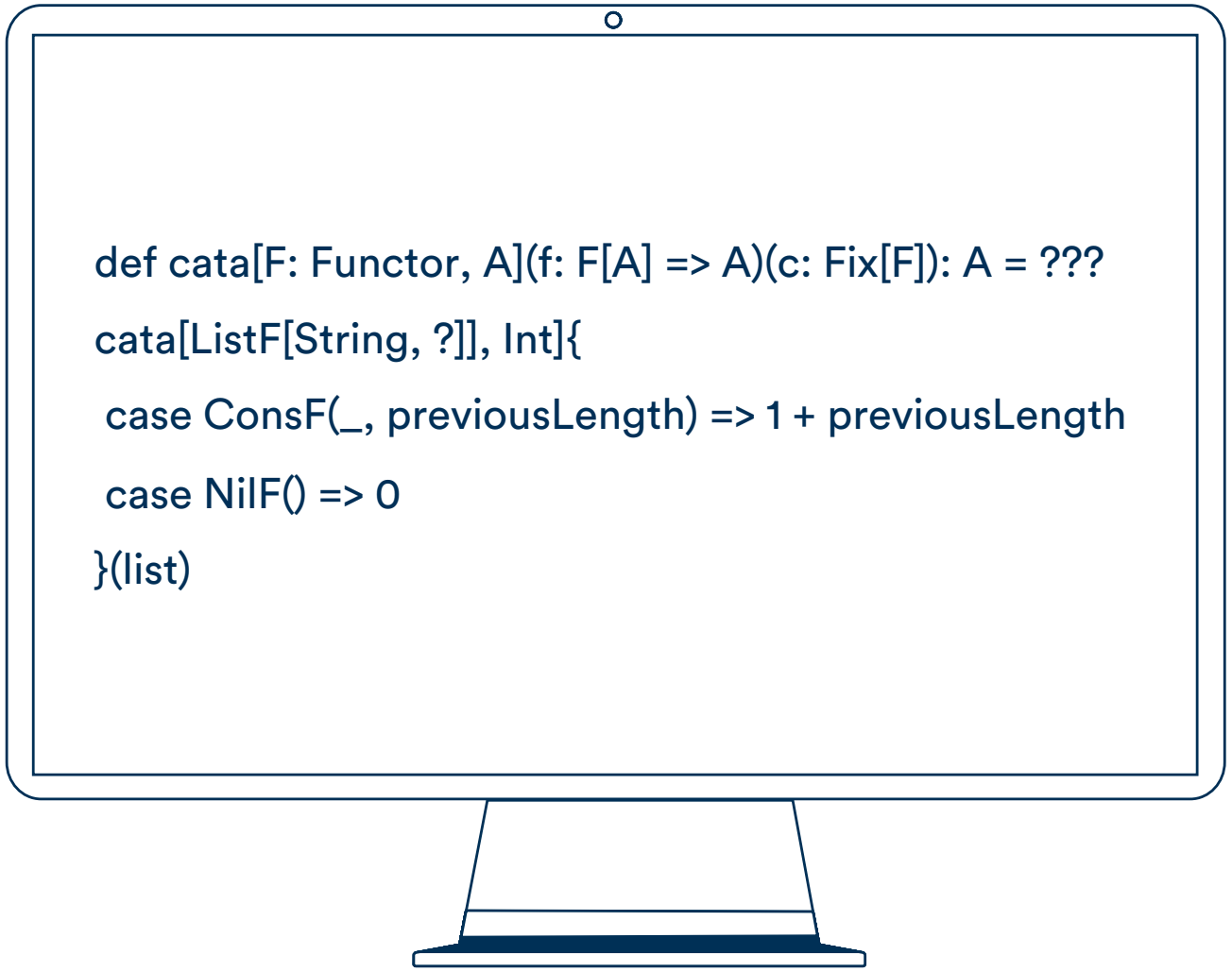
mediarithmics

Example: length of a list of String



mediarithmics

Example: length of a list of String



2

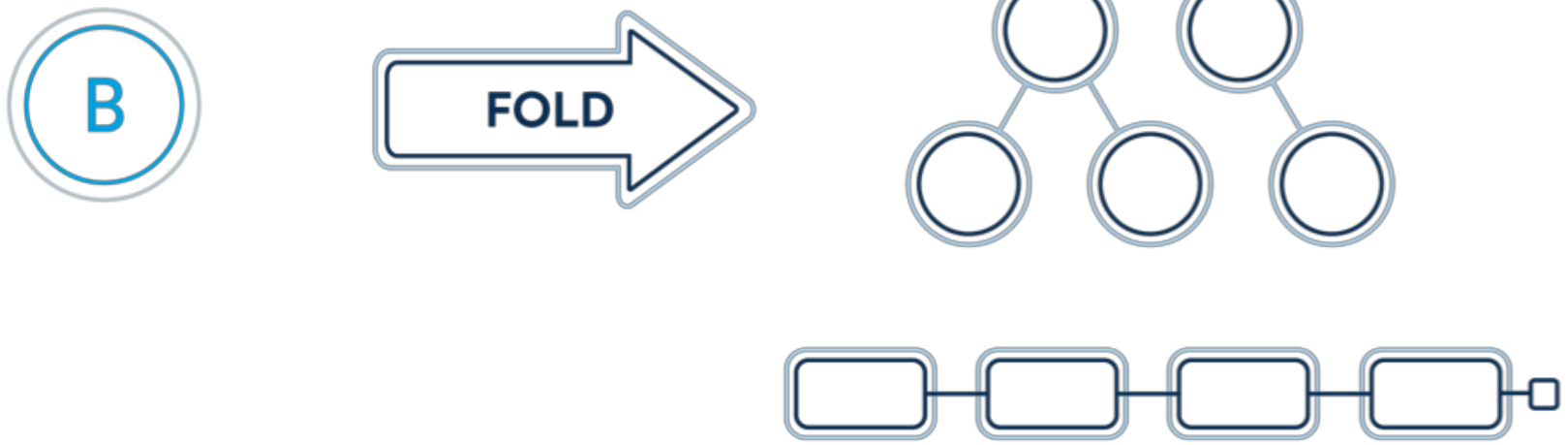
mediarithmics

Testability through Pattern functor

mediarithmics

Unfolds

Picture of a value on the left
and a tree on the right



mediarithmics

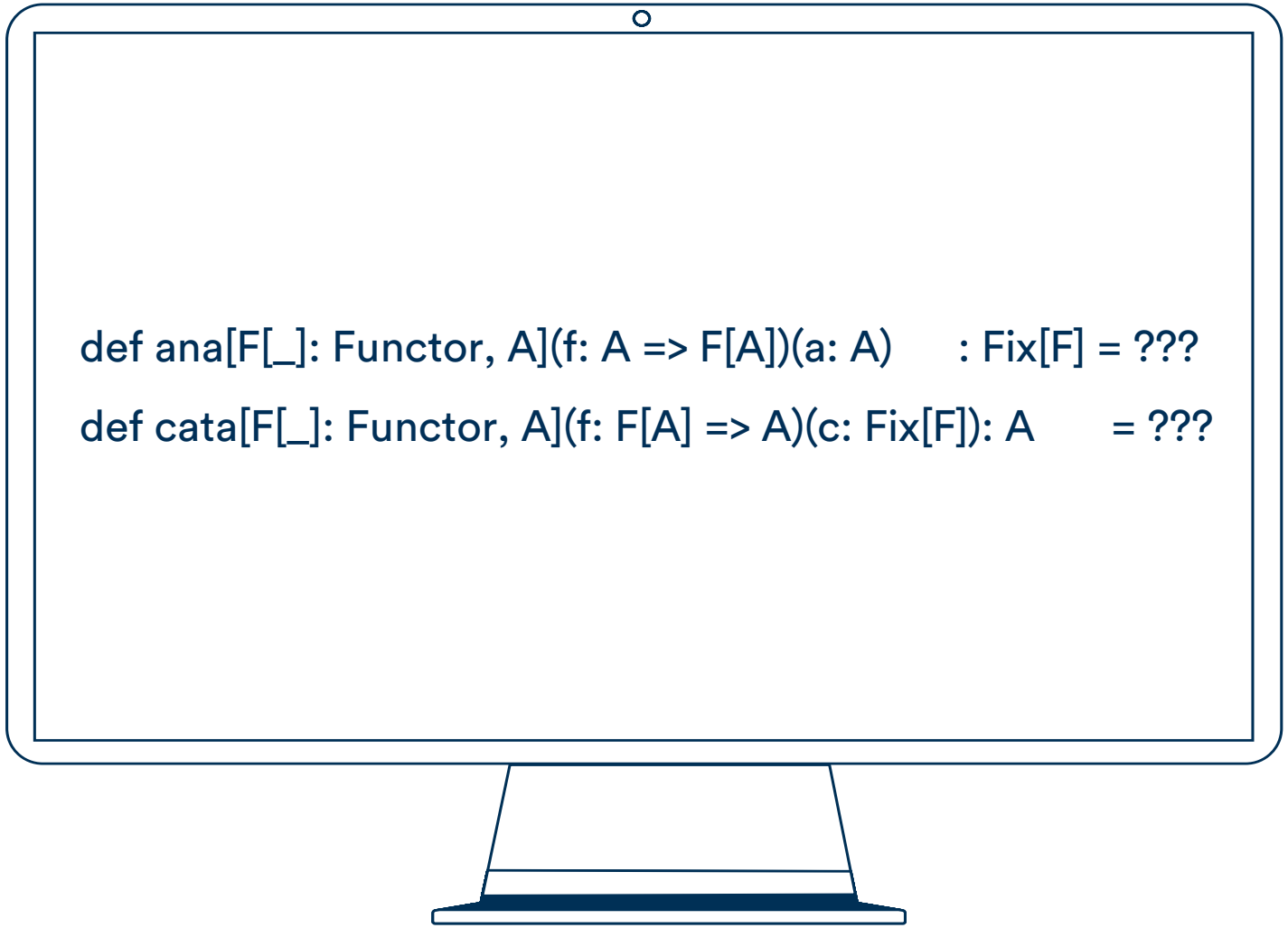
Unfolds: anamorphism

```
def ana[F[_]: Functor, A](f: A => F[A])(a: A): Fix[F] = ???
```

- One level at a time
- From greek ανα (up) as in anatomy, analogy, analyse, anabolic

mediarithmics

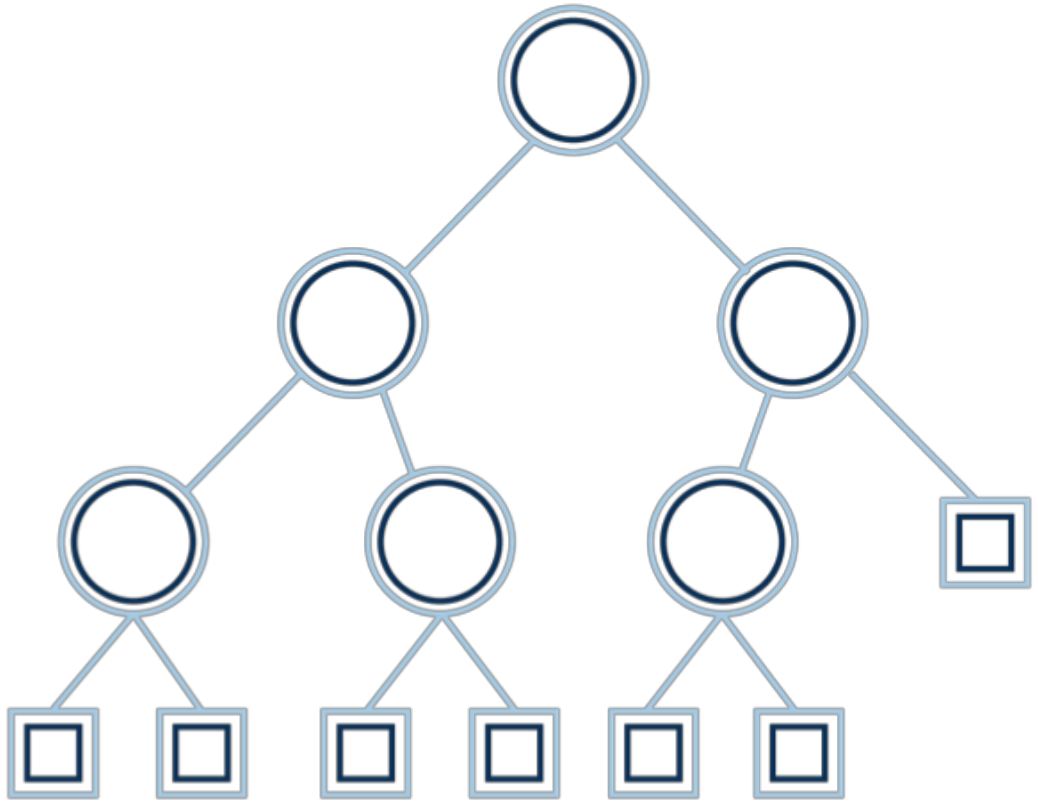
Unfolds: anamorphism



- One level at a time
- From greek ανα (up) as in anatomy, analogy, analyse, anabolic

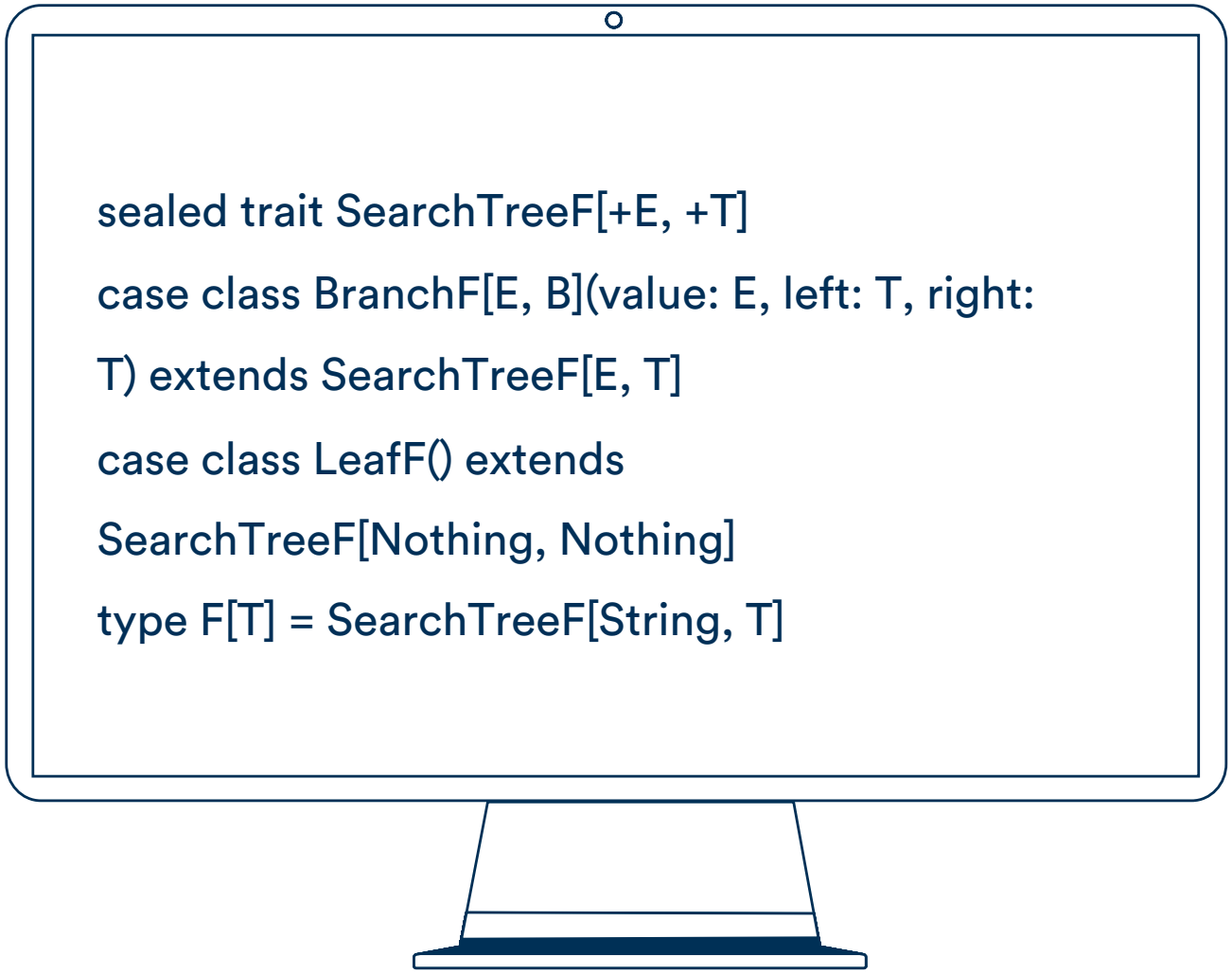
mediarithmics

Example: balanced bst from sorted string List



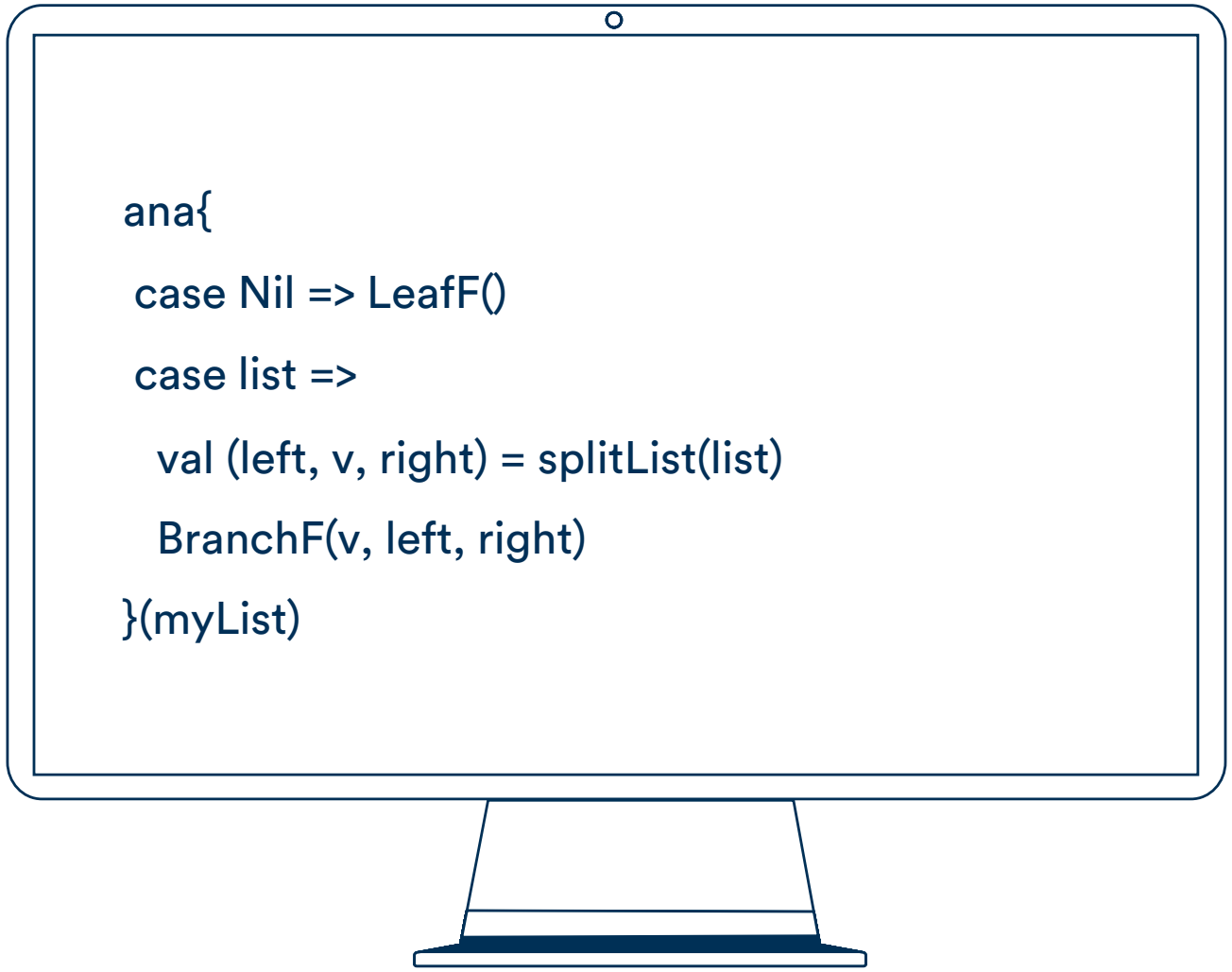
mediarithmics

Example: balanced bst from sorted string List



mediarithmics

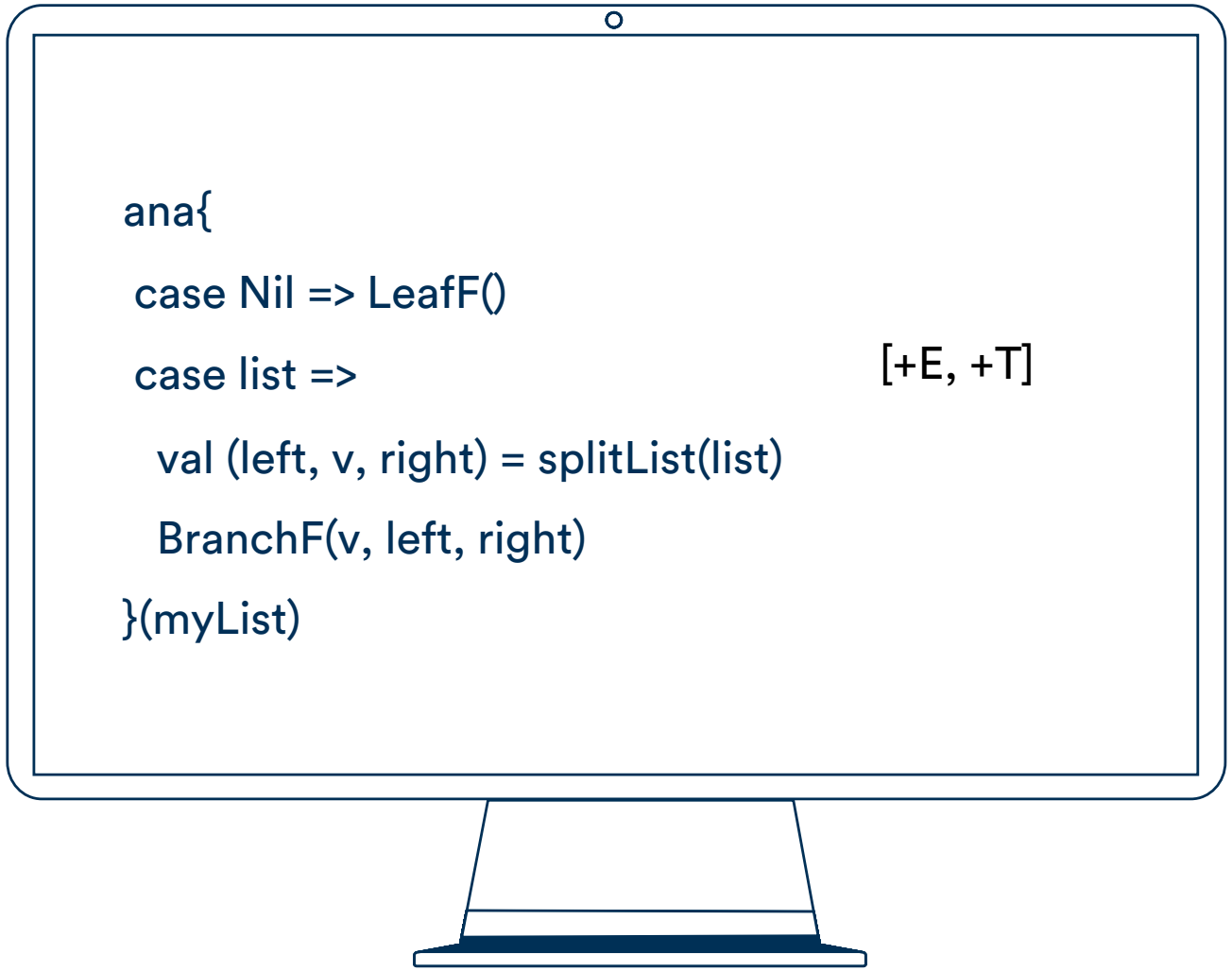
Example: balanced bst from sorted string List



[“bird”, “cat”,
“dog”, “fish”]

mediarithmics

Example: balanced bst from sorted string List

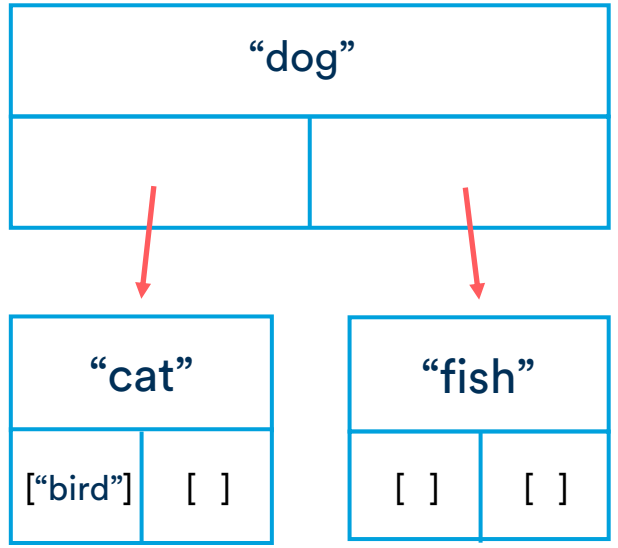


“dog”	
[“bird”, “cat”]	[“fish”]

mediarithmics

Example: balanced bst from sorted string List

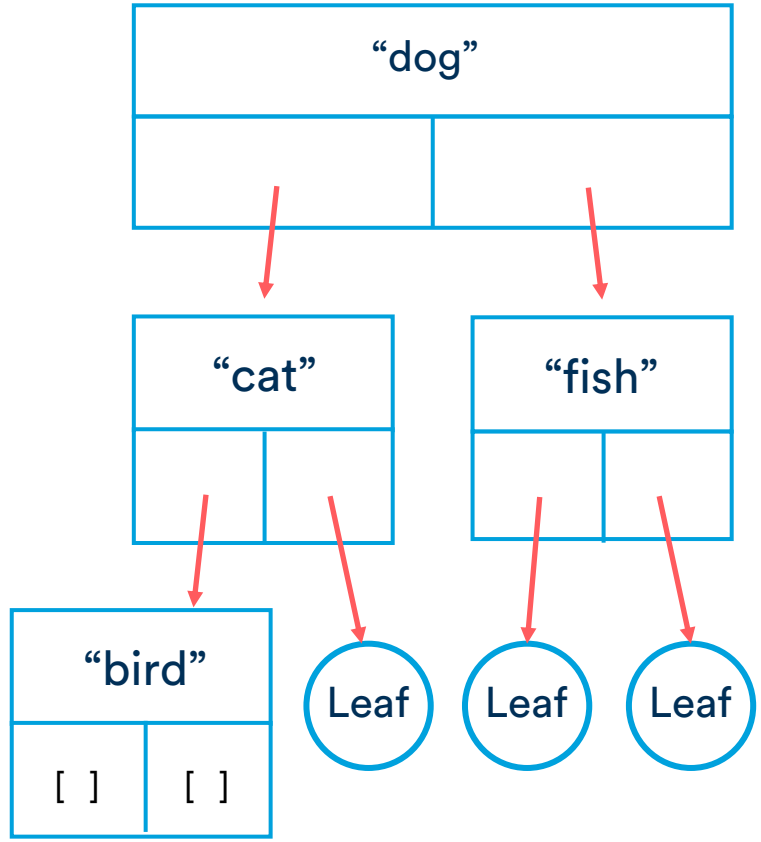
```
ana{
 case Nil => LeafF()
 case list =>                                    [+E, +T]
  val (left, v, right) = splitList(list)
  BranchF(v, left, right)
}(myList)
```



mediarithmics

Example: balanced bst from sorted string List

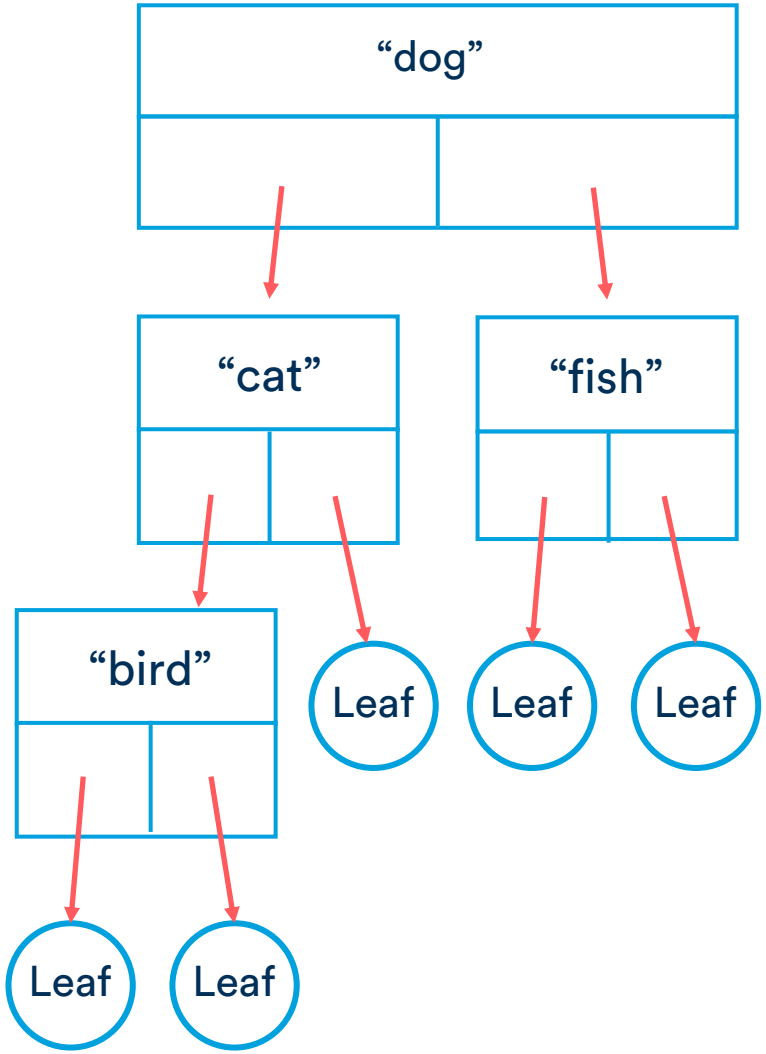
```
ana{
 case Nil => LeafF()
 case list =>                                    [+E, +T]
  val (left, v, right) = splitList(list)
  BranchF(v, left, right)
}(myList)
```



mediarithmics

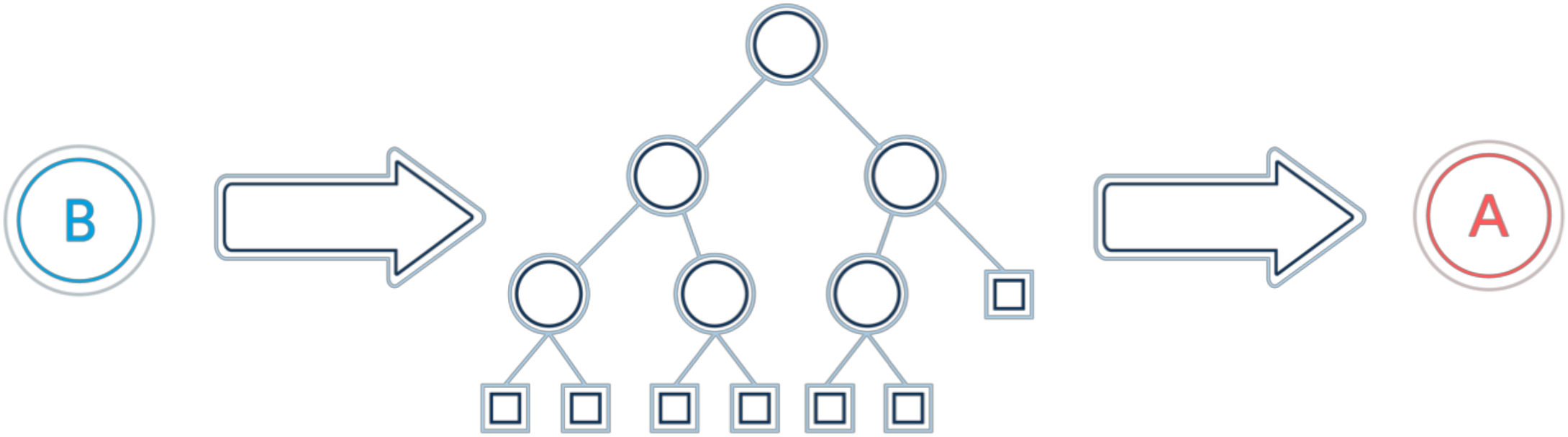
Example: balanced bst from sorted string List

```
ana{
 case Nil => LeafF()
 case list =>                                    [+E, +T]
  val (left, v, right) = splitList(list)
  BranchF(v, left, right)
}(myList)
```



mediarithmics

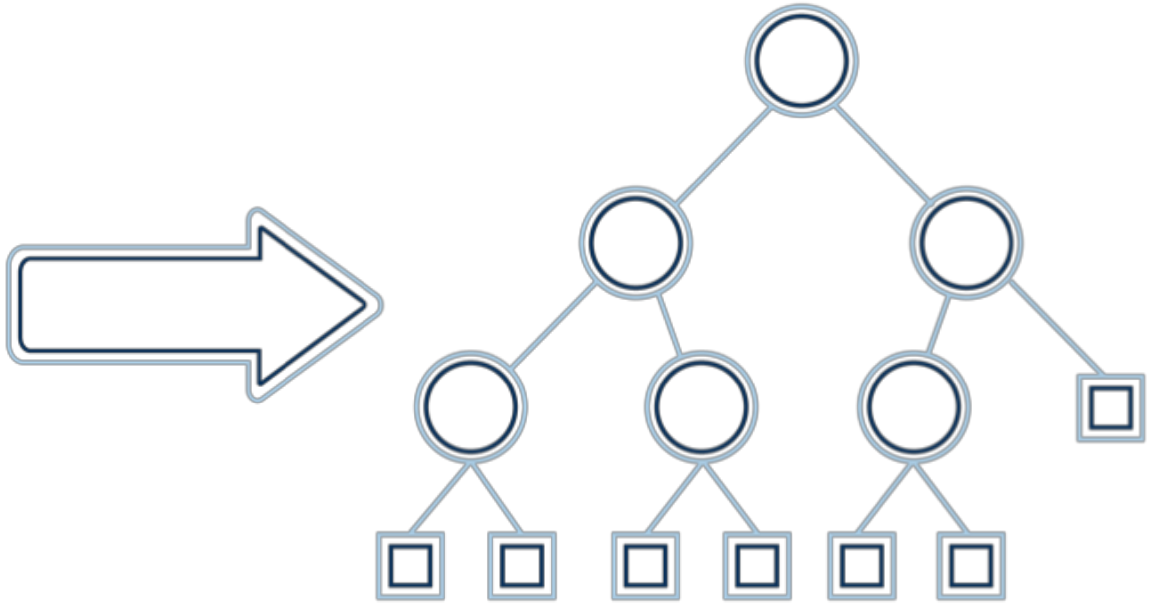
Refold



mediarithmics

Refold

https://url.com



42

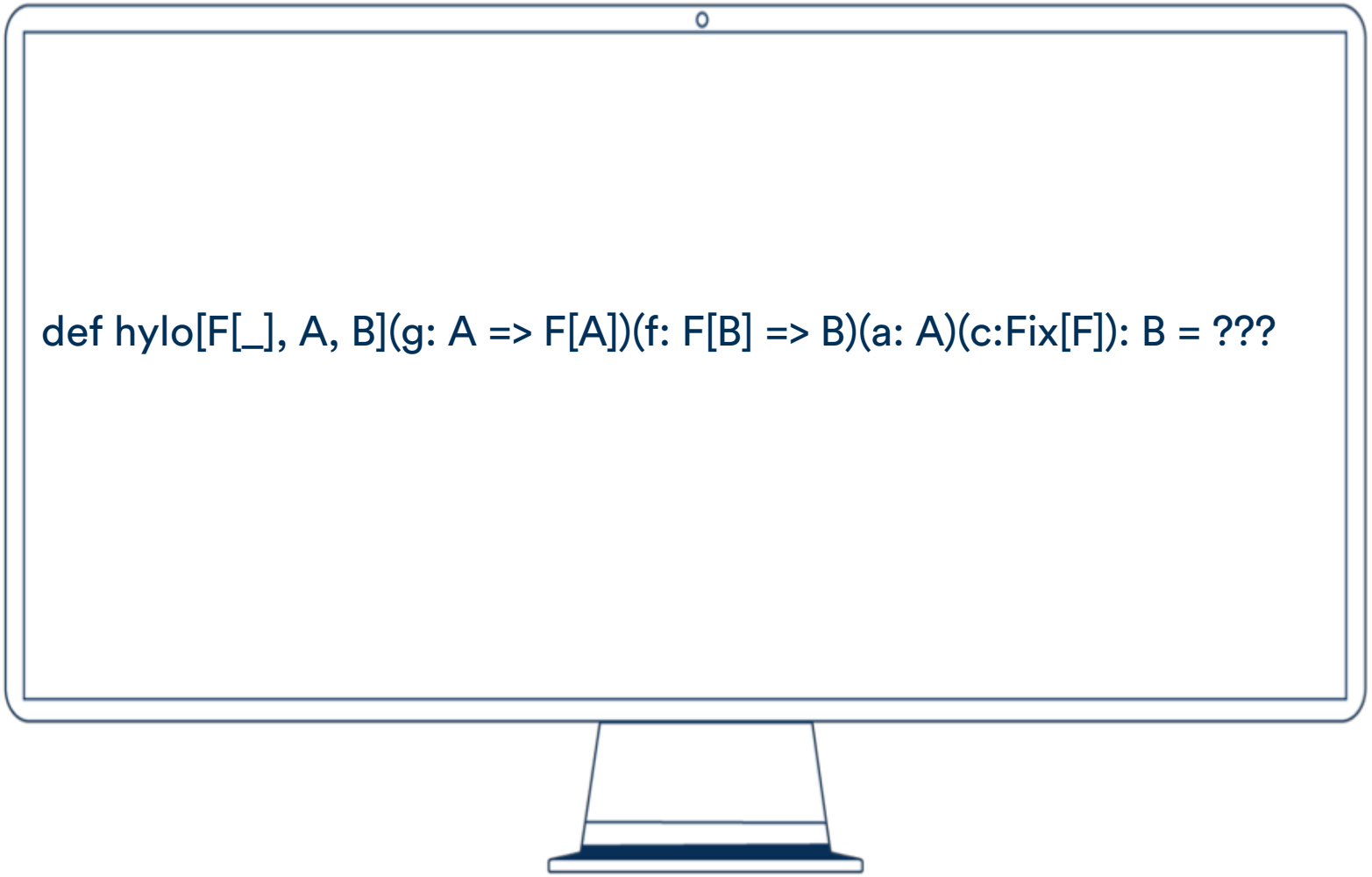
mediarithmics

Refolds: hylomorphism

- Building one step at a time
- Consuming one step at time
- It is just ana and then cata !
- But more efficient

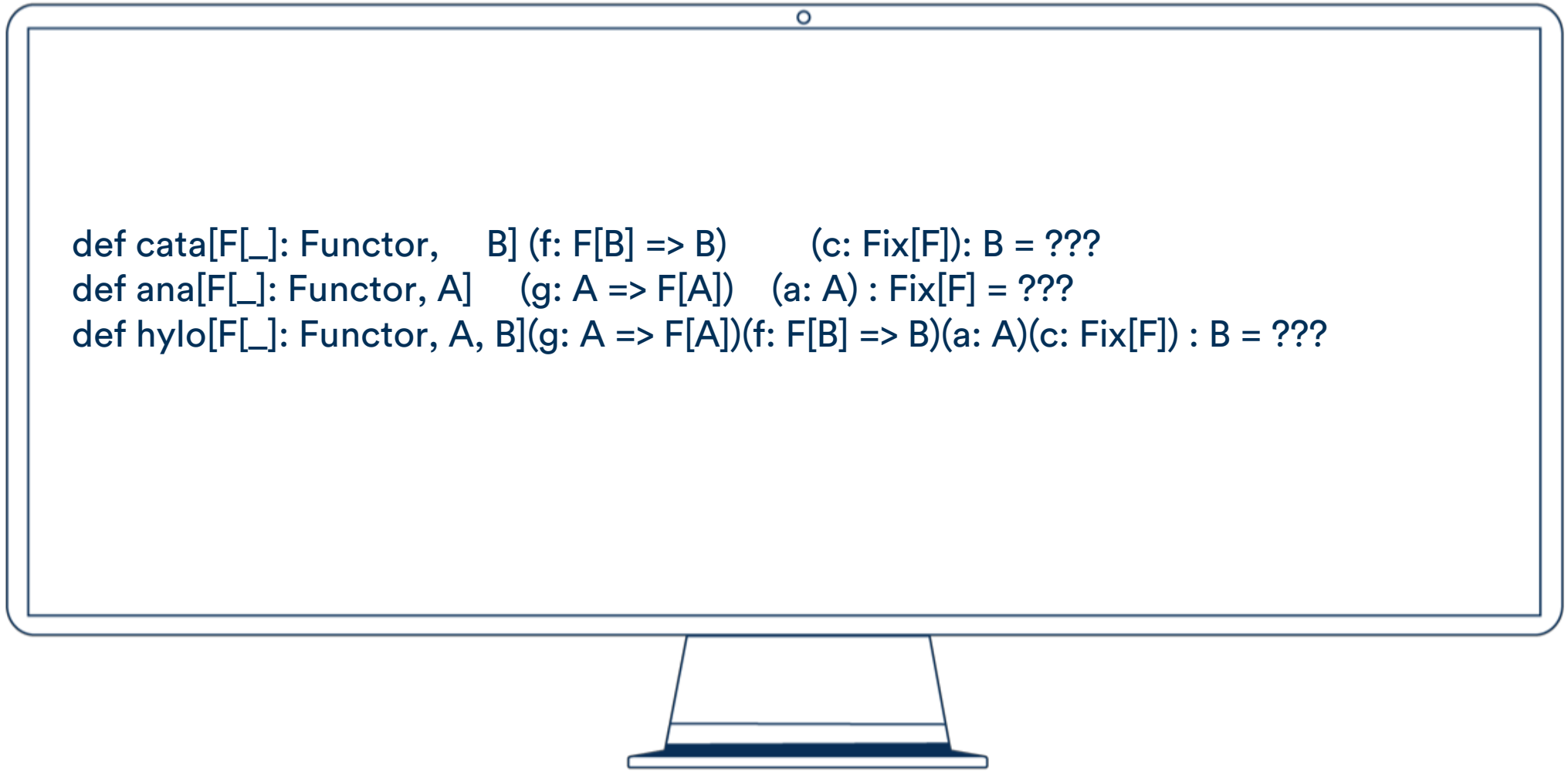
mediarithmics

Refolds: hylomorphism

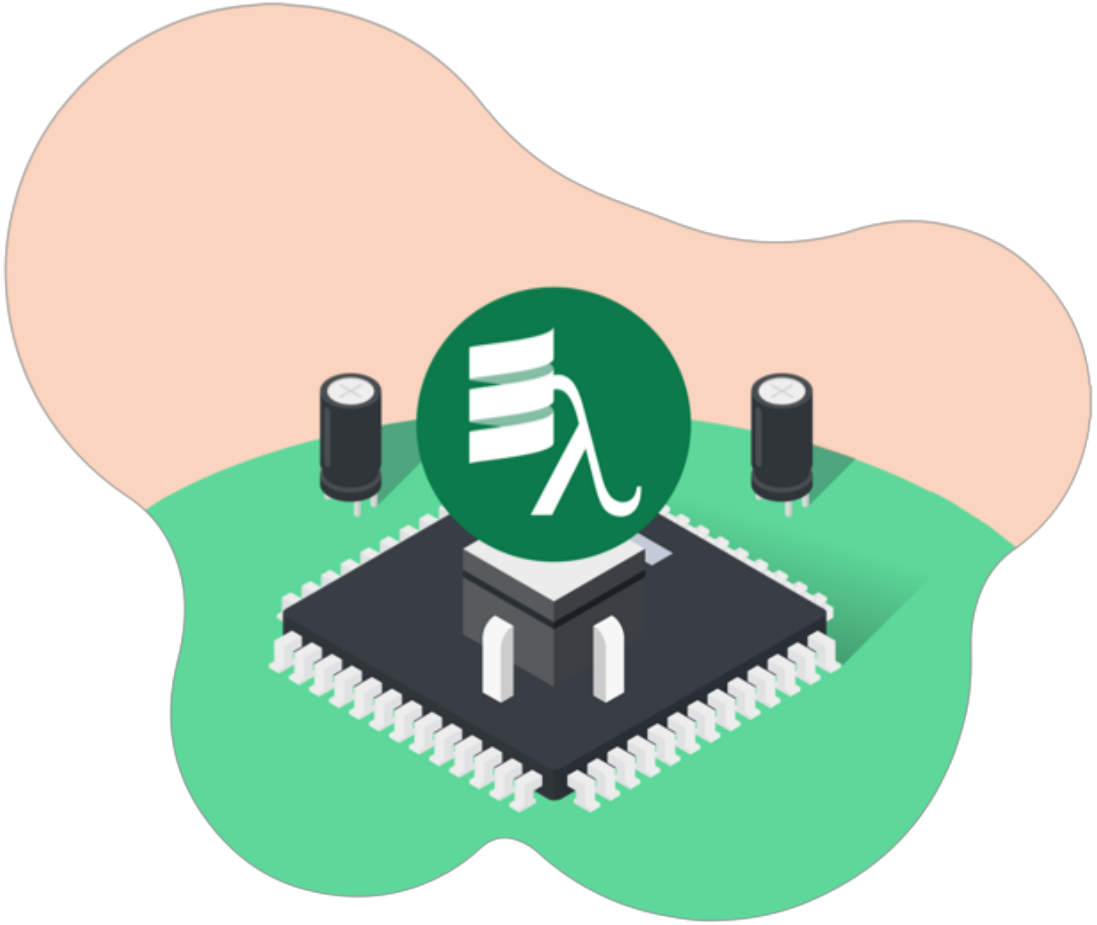


mediarithmics

Refolds: hylomorphism



Types of recursion schemes

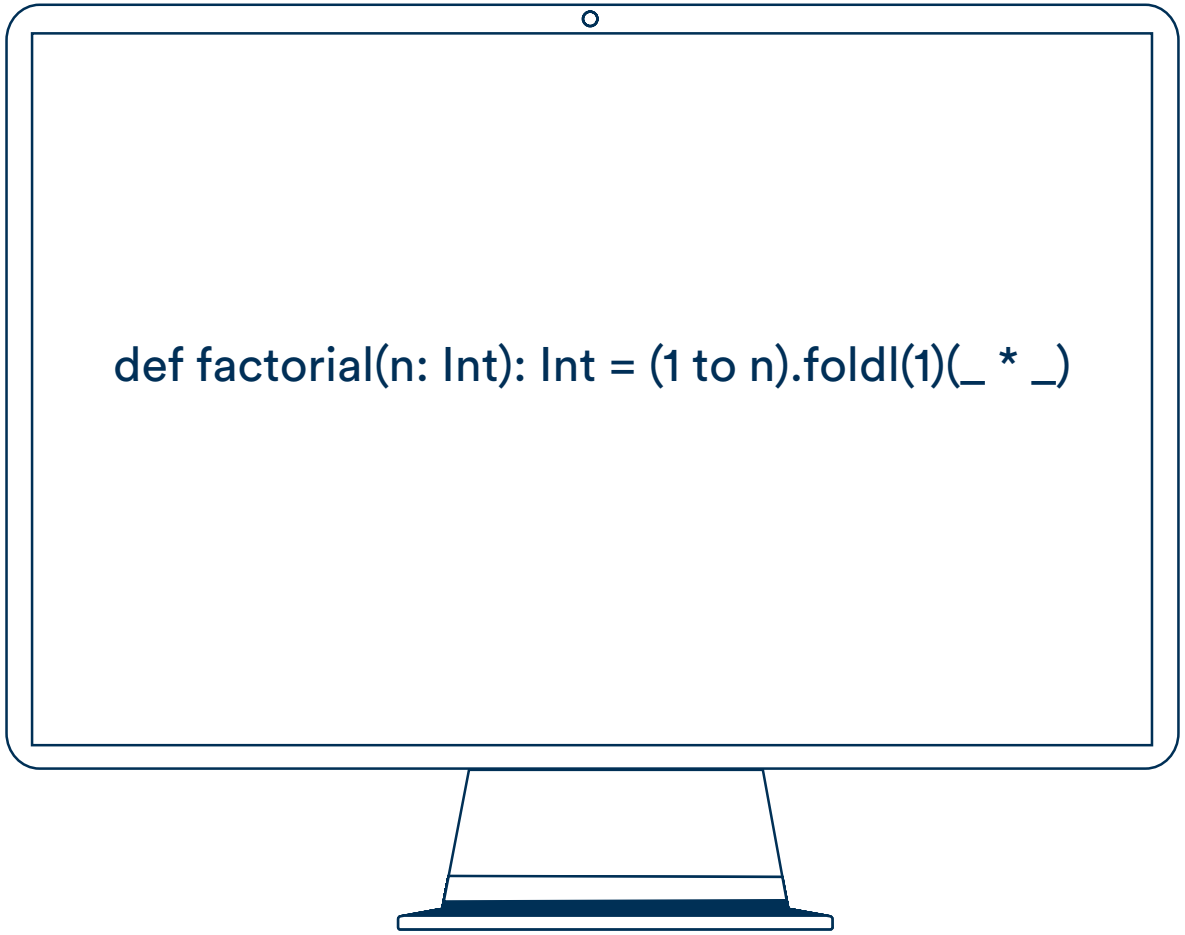


mediarithmics

Example: Factorial

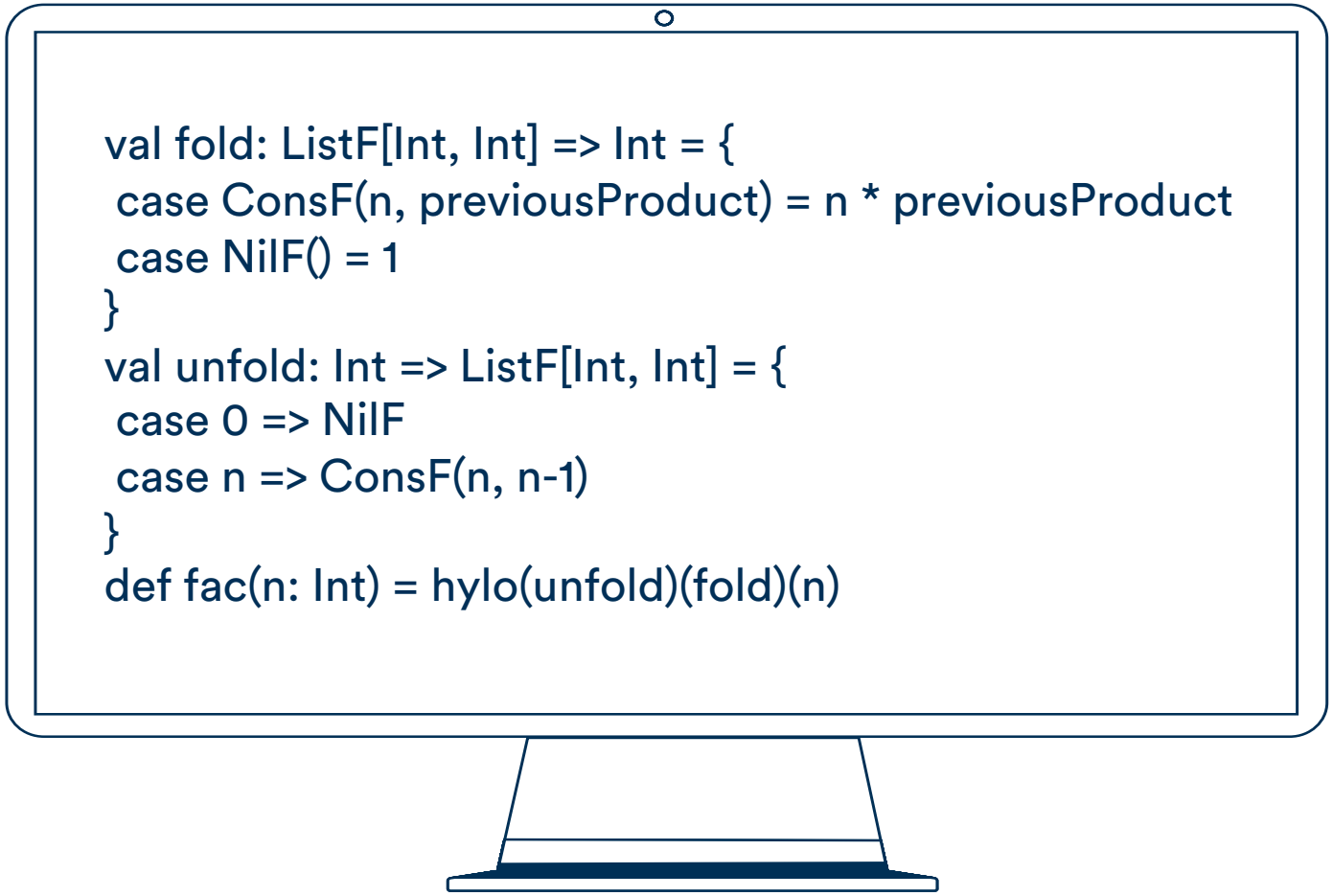
Computing a factorial:

- building a list: List(n, …, 3, 2, 1)
- computing the product of all the numbers in the list



mediarithmics

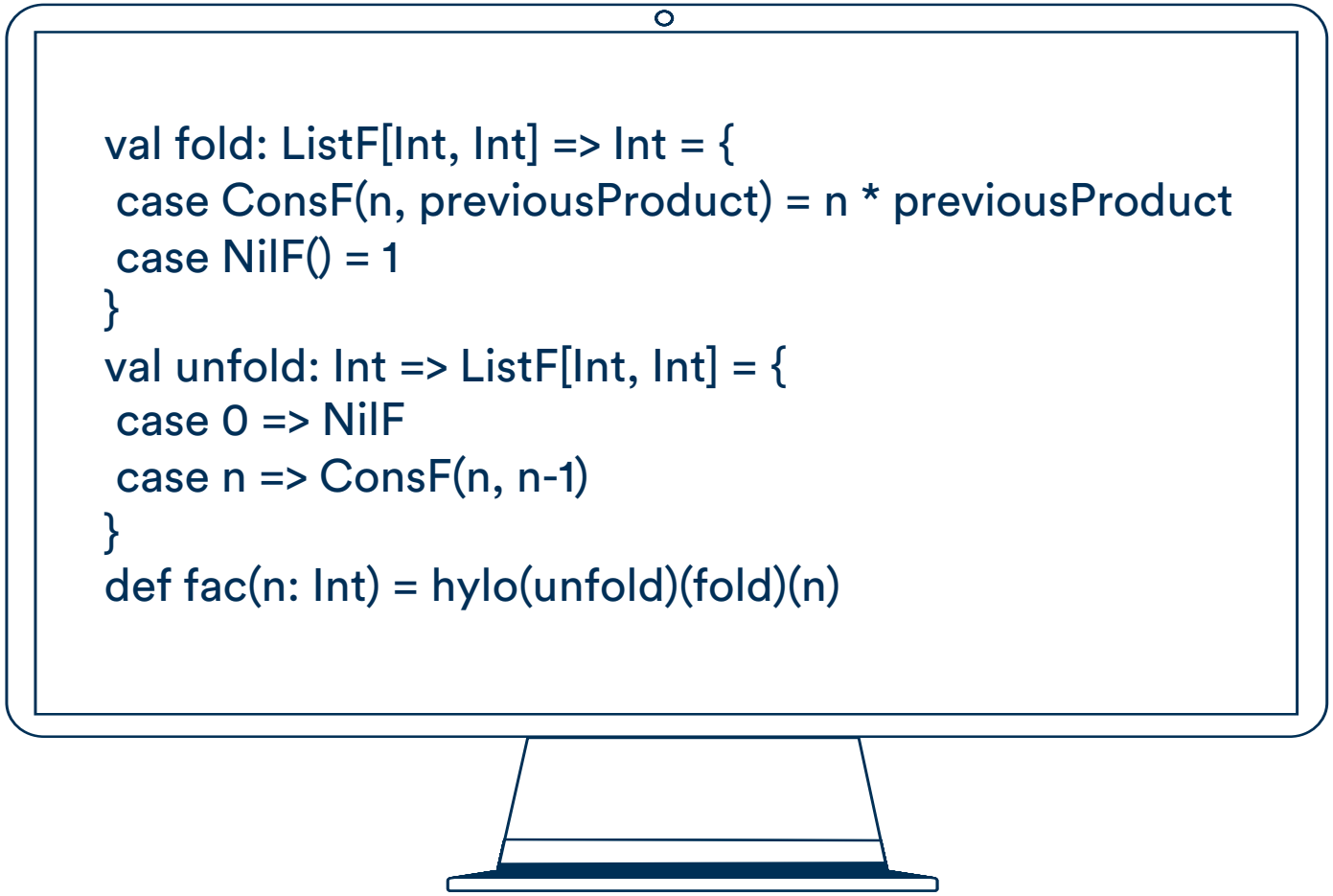
Example: Factorial



3

mediarithmics

Example: Factorial

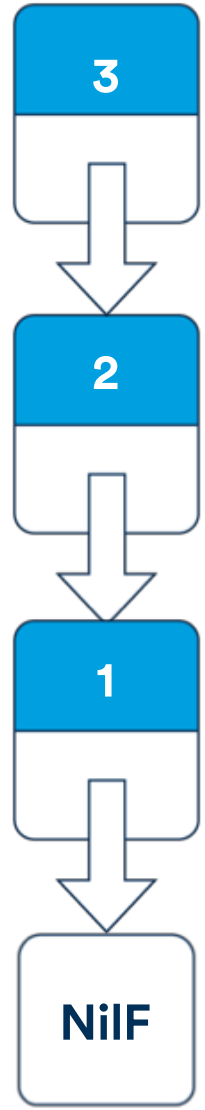


```
val fold: ListF[Int, Int] => Int = {
 case ConsF(n, previousProduct) = n * previousProduct
 case NilF() = 1
}
val unfold: Int => ListF[Int, Int] = {
 case 0 => NilF
 case n => ConsF(n, n-1)
}
def fac(n: Int) = hylo(unfold)(fold)(n)
```

mediarithmics

Example: Factorial

```
val fold: ListF[Int, Int] => Int = {
 case ConsF(n, previousProduct) = n * previousProduct
 case NilF() = 1
}
val unfold: Int => ListF[Int, Int] = {
 case 0 => NilF
 case n => ConsF(n, n-1)
}
def fac(n: Int) = hylo(unfold)(fold)(n)
```



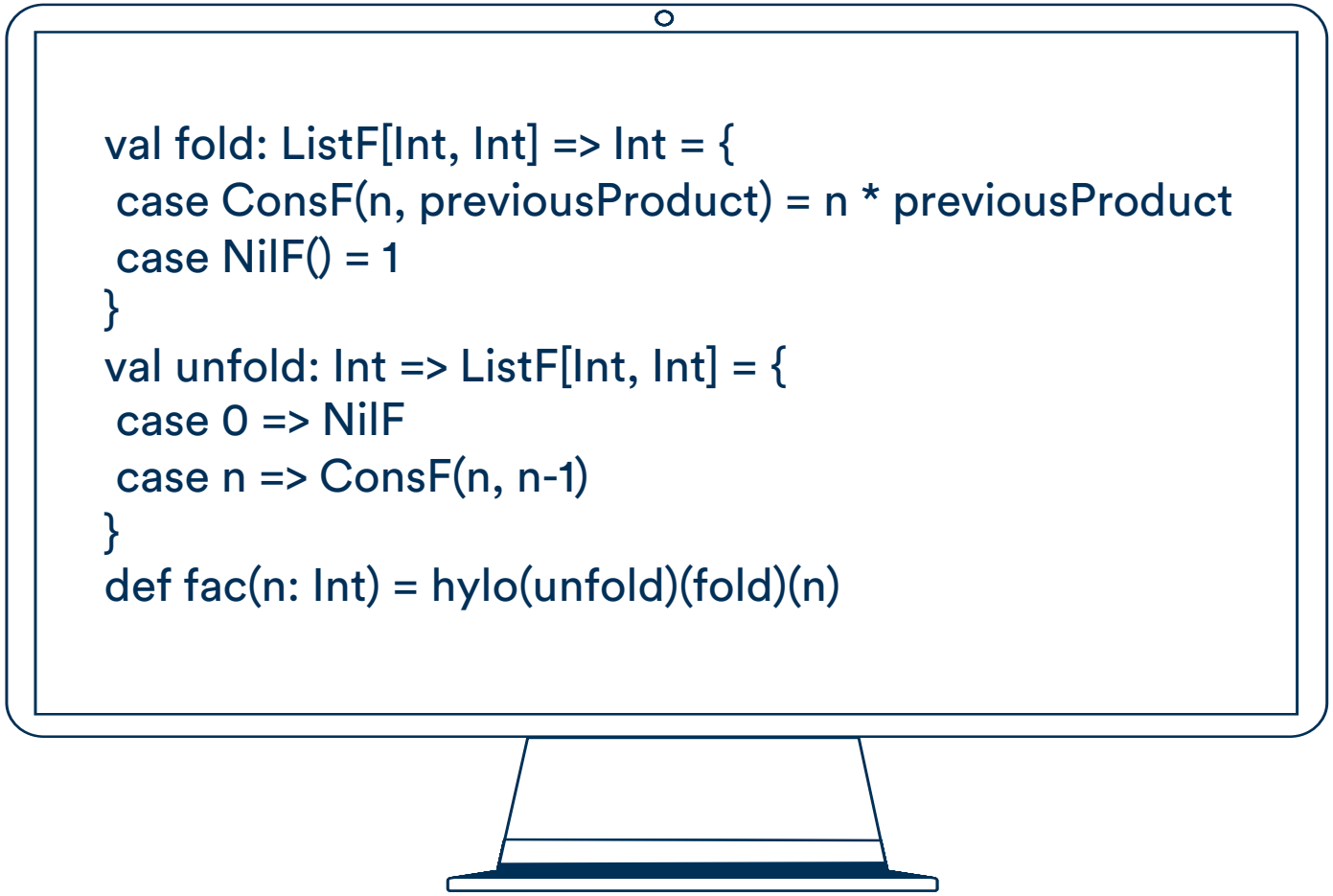
mediarithmics

Example: Factorial

```
val fold: ListF[Int, Int] => Int = {
 case ConsF(n, previousProduct) = n * previousProduct
 case NilF() = 1
}
val unfold: Int => ListF[Int, Int] = {
 case 0 => NilF
 case n => ConsF(n, n-1)
}
def fac(n: Int) = hylo(unfold)(fold)(n)
```

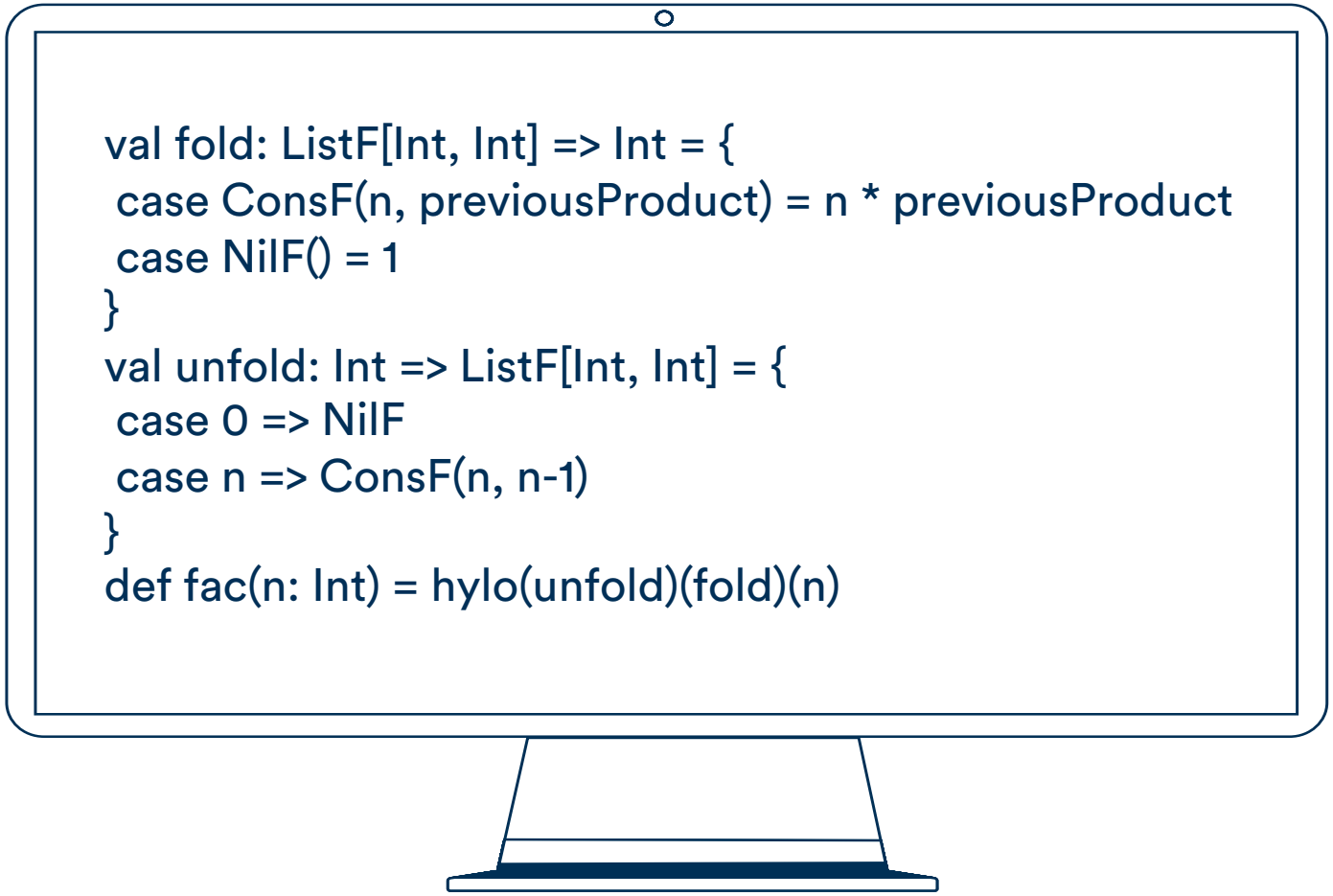
mediarithmics

Example: Factorial



mediarithmics

Example: Factorial



6

Demo

mediarithmics

Main Libraries

Matryoshka

(ScalaZ)

Droste

(Typelevel)

mediarithmics

Useful knowledge when working with libraries

- The data structure are encapsulate in Algebra(fold) and CoAlgebra(unfold)
- Some other (un)folding function exist to embody final stop or other special behavior.
  However this can be simpler to embed this behavior in the data structure
- Also exist with an effect:

```
def cataM[F[_] : Traverse, M[_] : Monad, A](f: F[A] => M[A])(v: Fix[F]): M[A]
```

mediarithmics

Useful Resources

- https://github.com/passy/awesome-recursion-schemes
- https://blog.sumtypeofway.com/an-introduction-to-recursion-schemes/
- http://free.cofree.io/2017/11/13/recursion/ (https://github.com/zliu41/zliu41.github.io/blob/master/_posts/2017-11-13-recursion.md)
- Both Droste and matryoshka have an awesome gitter
- https://www.youtube.com/watch?v=oRLkb6mqvVM&list=PL9KXdMOfekvYYX53KHR-28e3BjlTIEjbT&index=5
- https://www.youtube.com/watch?v=XZ9nPZbaYfE
- https://www.youtube.com/watch?v=zkDVCQiveEo

mediarithmics

Advantages and drawbacks

Advantages:

- Better readability for complexe cases
- More efficient than naive implementations (Template Driven Development)
- Stack safe (if the data structure is stack safe)

Drawbacks:

- Learning curve
- Documentation is very *academic*
- Lot of boilerplate for simple recursion

mediarithmics

Thank you !

devs@mediarithmics.com