Haskell eXchange 2015

Programming from Universal Properties

Gershom Bazerman,
S&P/CapitalIQ

Warning: This Talk Contains Lies

Fast and Loose Reasoning is Morally Correct *

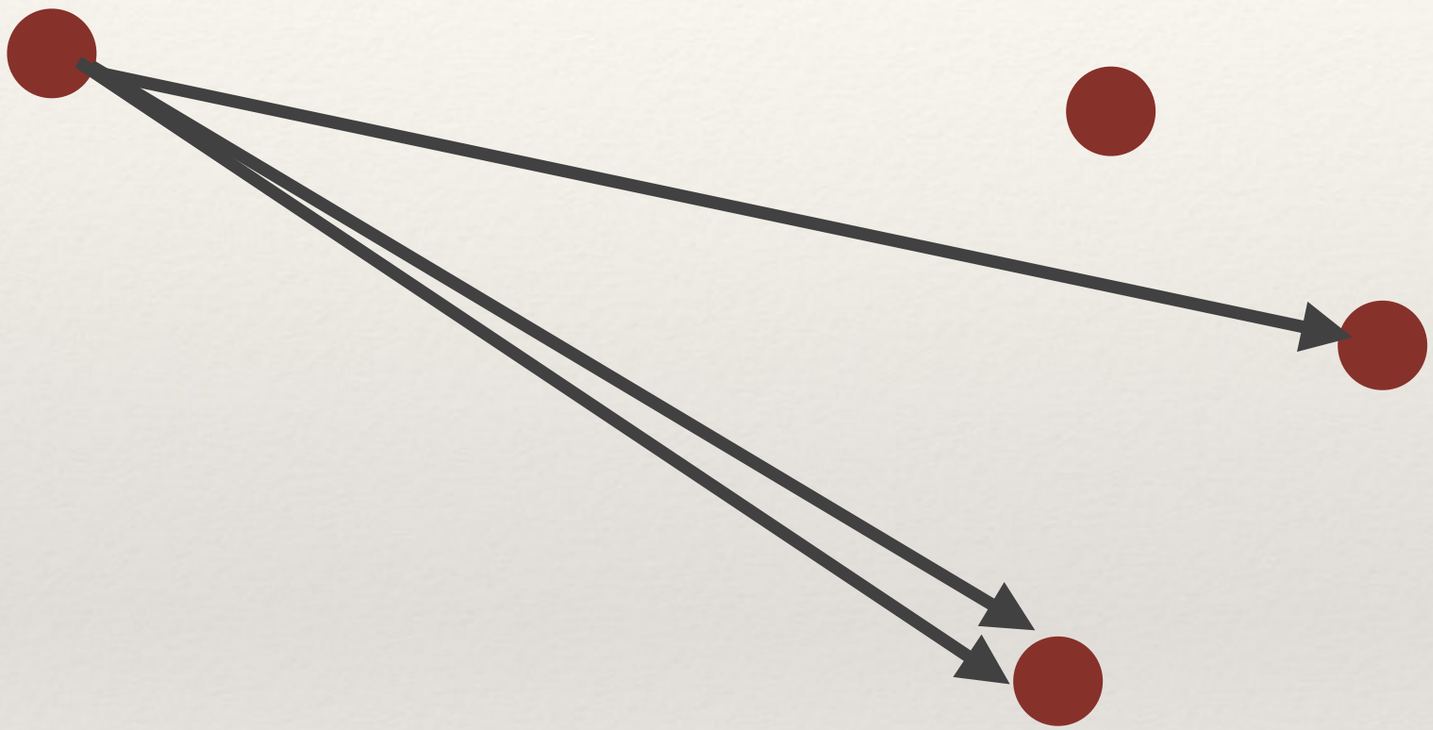
Nils Anders Danielsson John Hughes
Patrik Jansson
Chalmers University of Technology
{nad,rjmh,patrikj}@cs.chalmers.se

Jeremy Gibbons
Oxford University Computing Laboratory
Jeremy.Gibbons@comlab.ox.ac.uk

What is a ~~Universal~~ Property

- Take some notion of a "mathematical object" and define some notion of something (or things) one can do with it. That's a property!
- Why is "color" a property of a wheelbarrow? Because there is a function Wheelbarrow -> Color.

What is a ~~Universal~~ Property Categorically

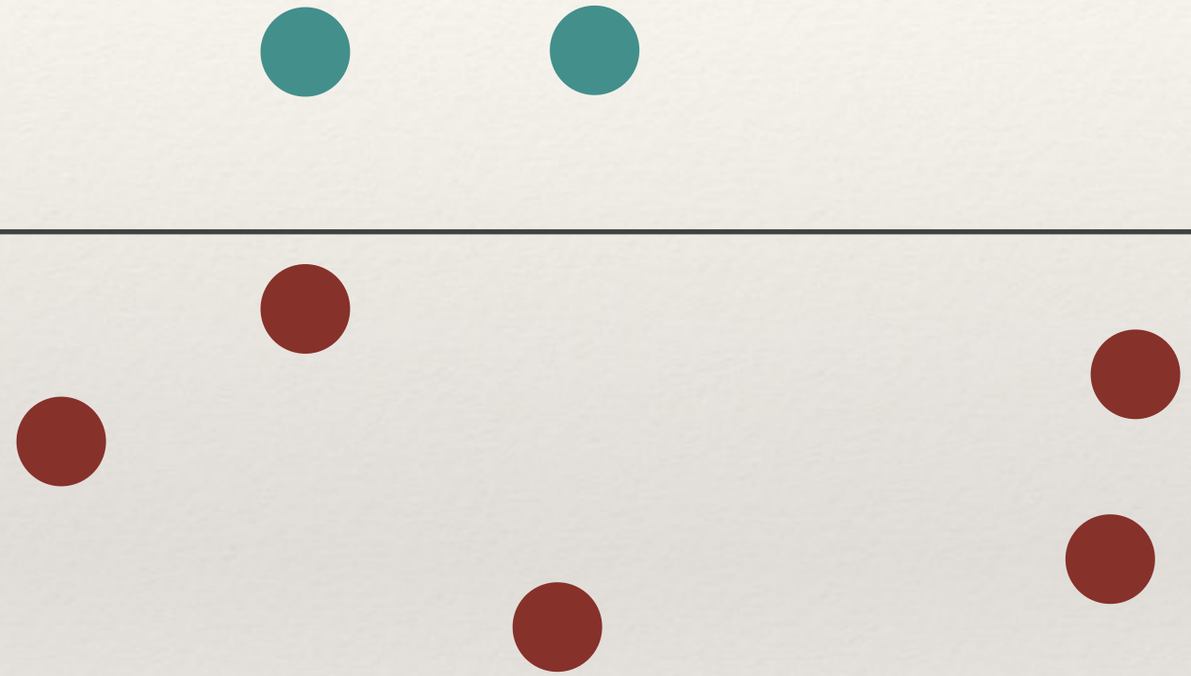


What is a **Universal** Property

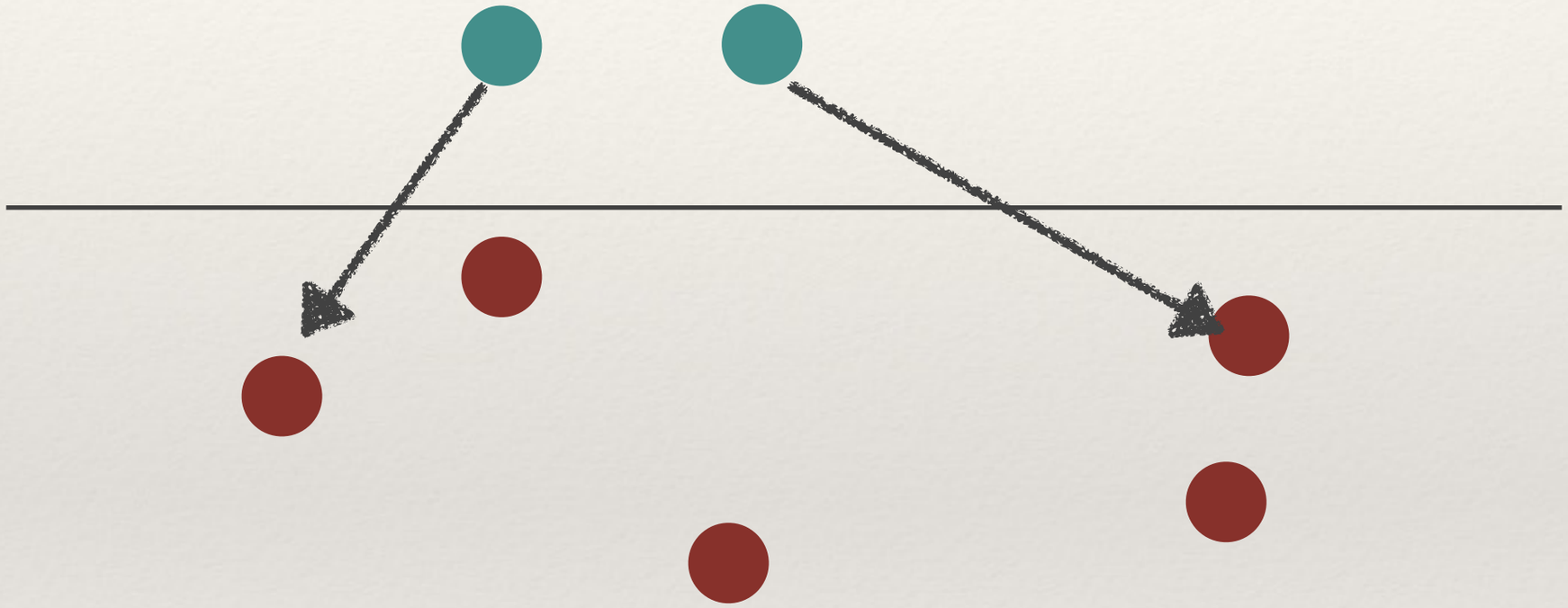
- A universal property of an object is still something you can do with it, but it something you can do with it that encompasses **everything** you can do with it.
- The universal property of Bool is `a -> a -> a`.
- We capture the “essence” of Bool without direct reference to Bool.

What is a **Universal** Property Categorically

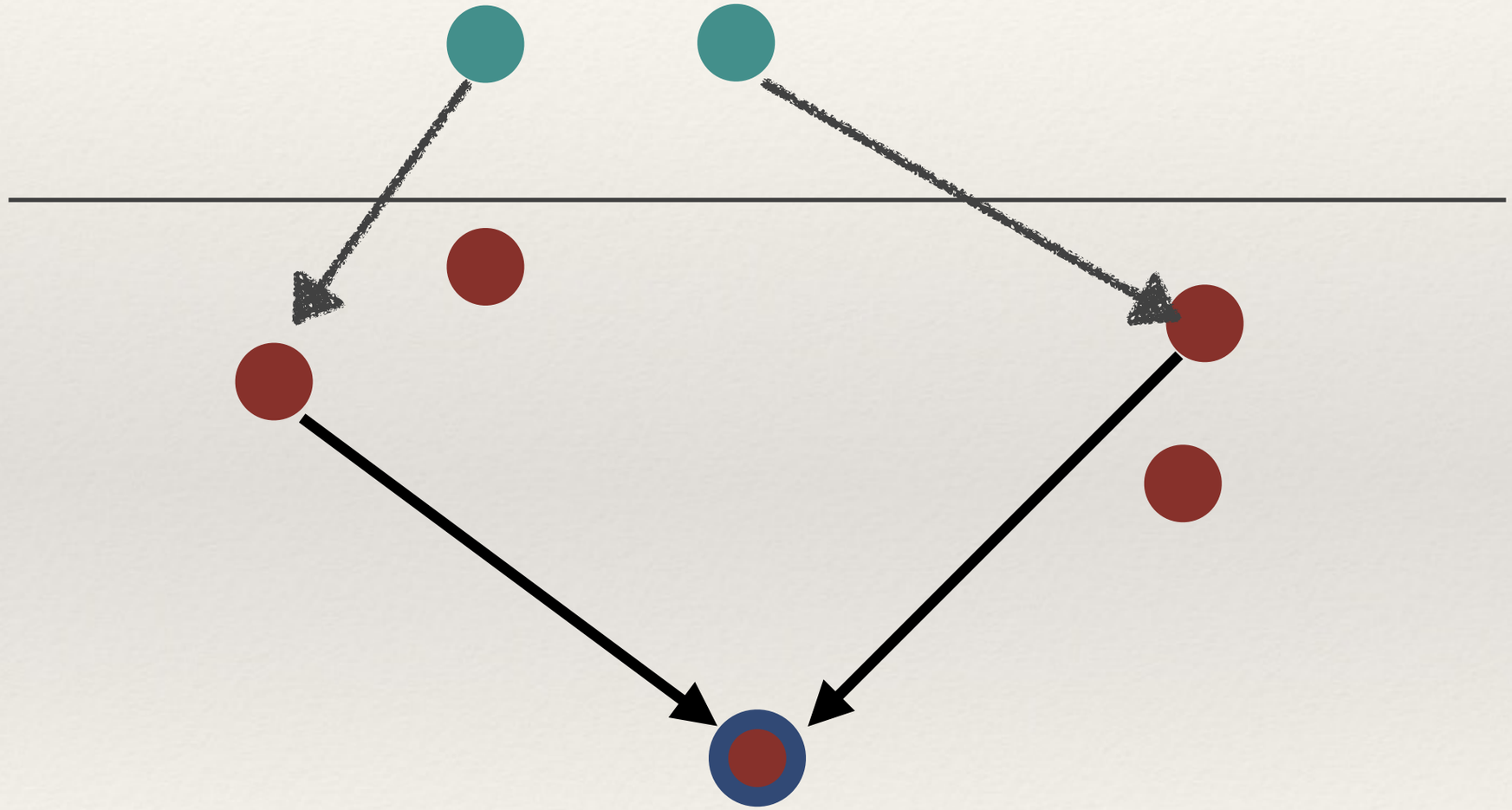
(Ok, only a special case)



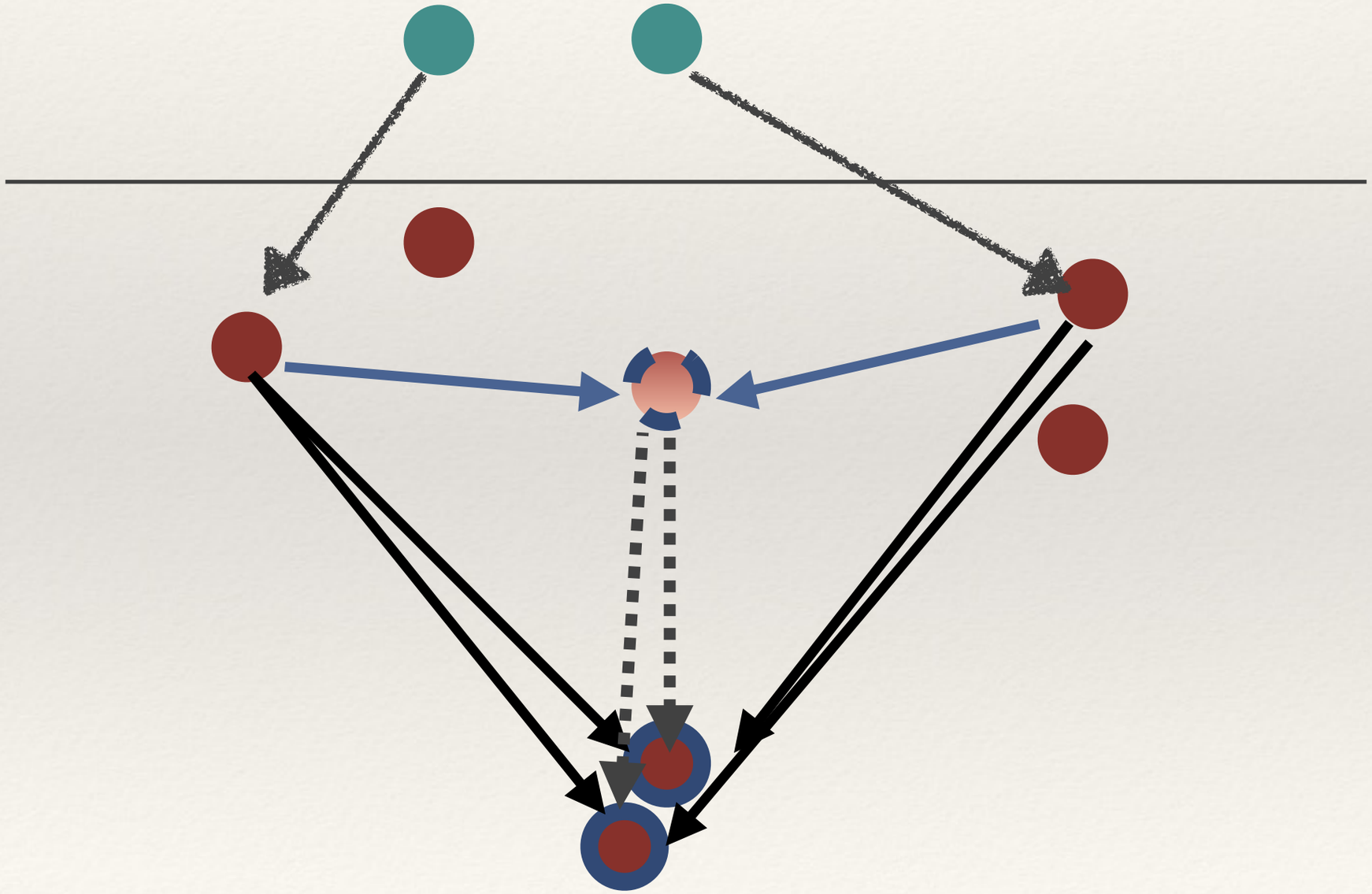
1) A Diagram



2) A Co-Cone



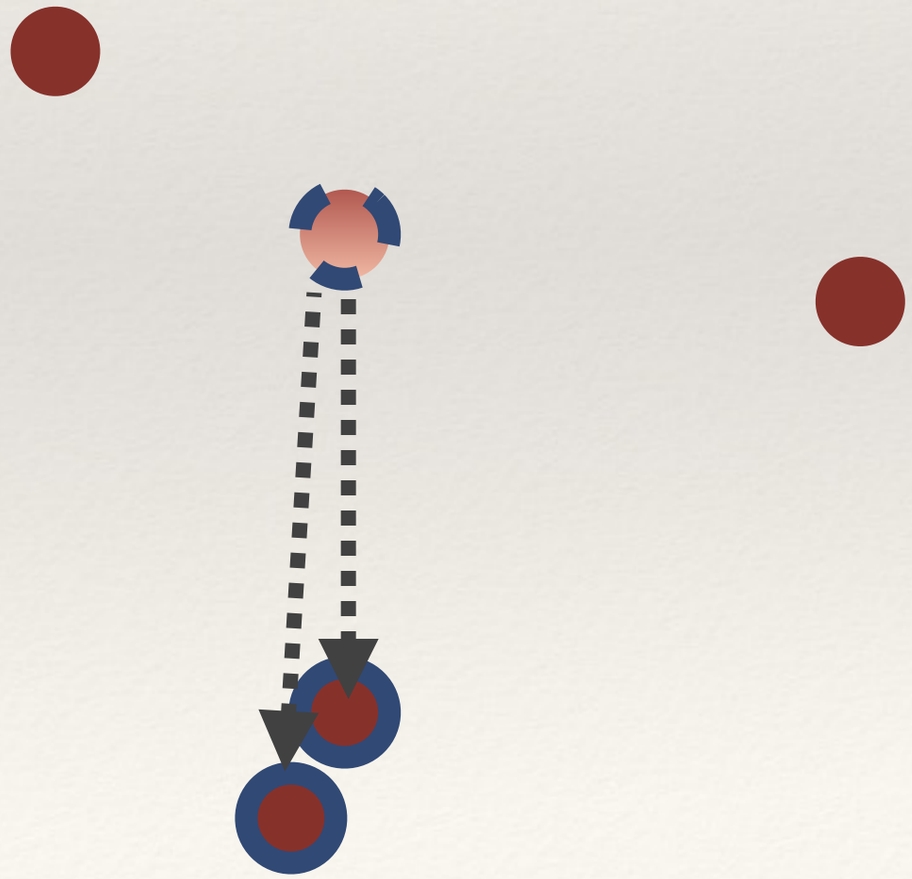
3) A Universal Co-Cone; i.e. A Colimit



Varieties of Colimits

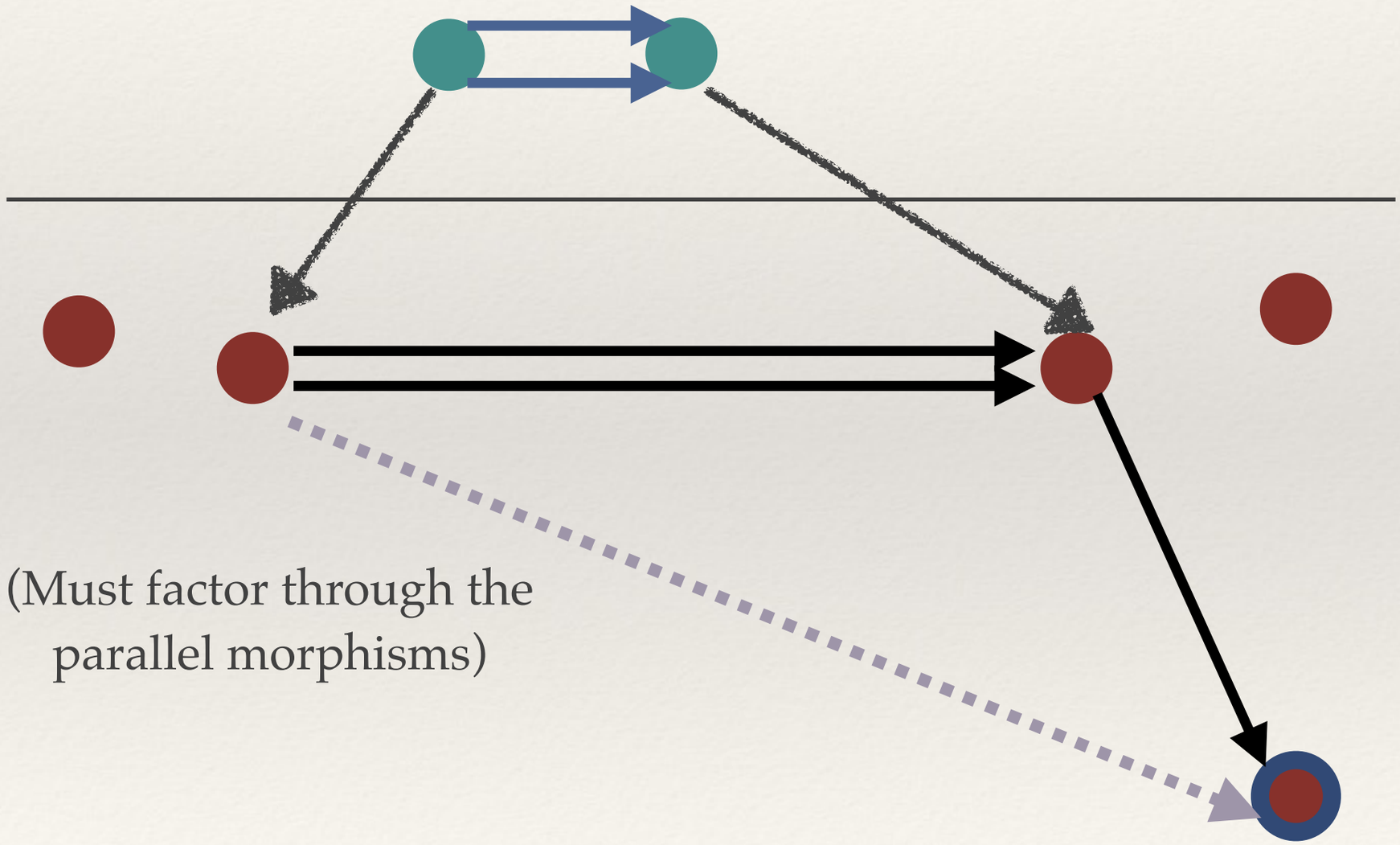
- N discrete objects — N-ary Coproduct (Either).
- 0 discrete objects — Initial Object (Void).

```
newtype Void = Void Void

absurd :: Void -> a
absurd (Void a) = absurd a
```



Varieties of Colimits

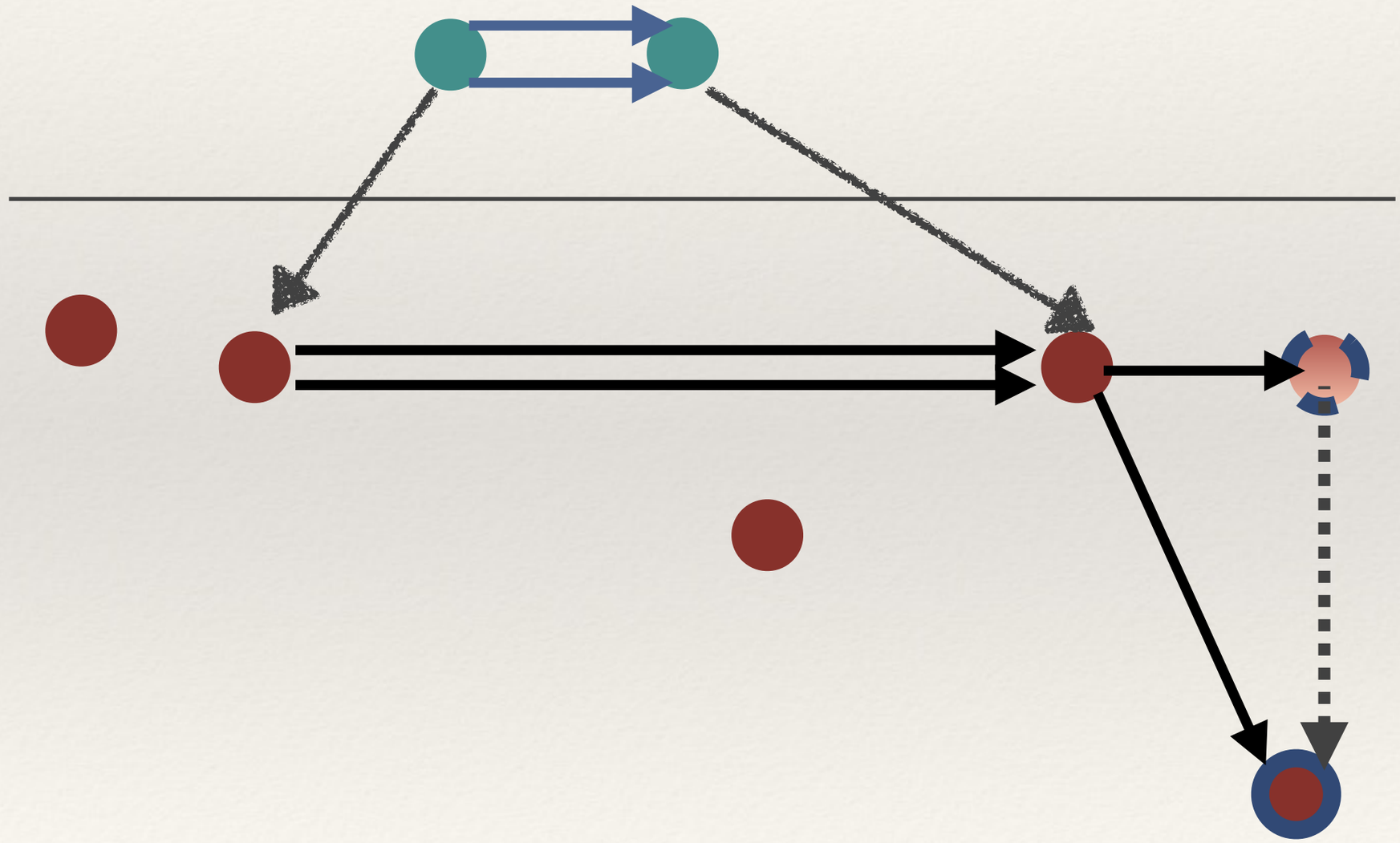
- Two objects, parallel morphisms — Coequalizer



(Must factor through the parallel morphisms)

Varieties of Colimits

- Two objects, parallel morphisms — Coequalizer



Some Coequalizers

- Objects are Sets, Morphisms are Set-theoretic functions

Coequalizers are quotients — i.e. f, g : A -> B then

Coeq(f,g) ~=

[f(x) | x <- a, f(x)=g(x)] +

[(f(x),g(x)) | x <- a, f(x) /= g(x)] (sort of)

Some Coequalizers

- Take Matr_Int — objects, given by N, are sets of all NxN matrices, morphisms are NxM matrices that act by multiplication and we have a zero object.

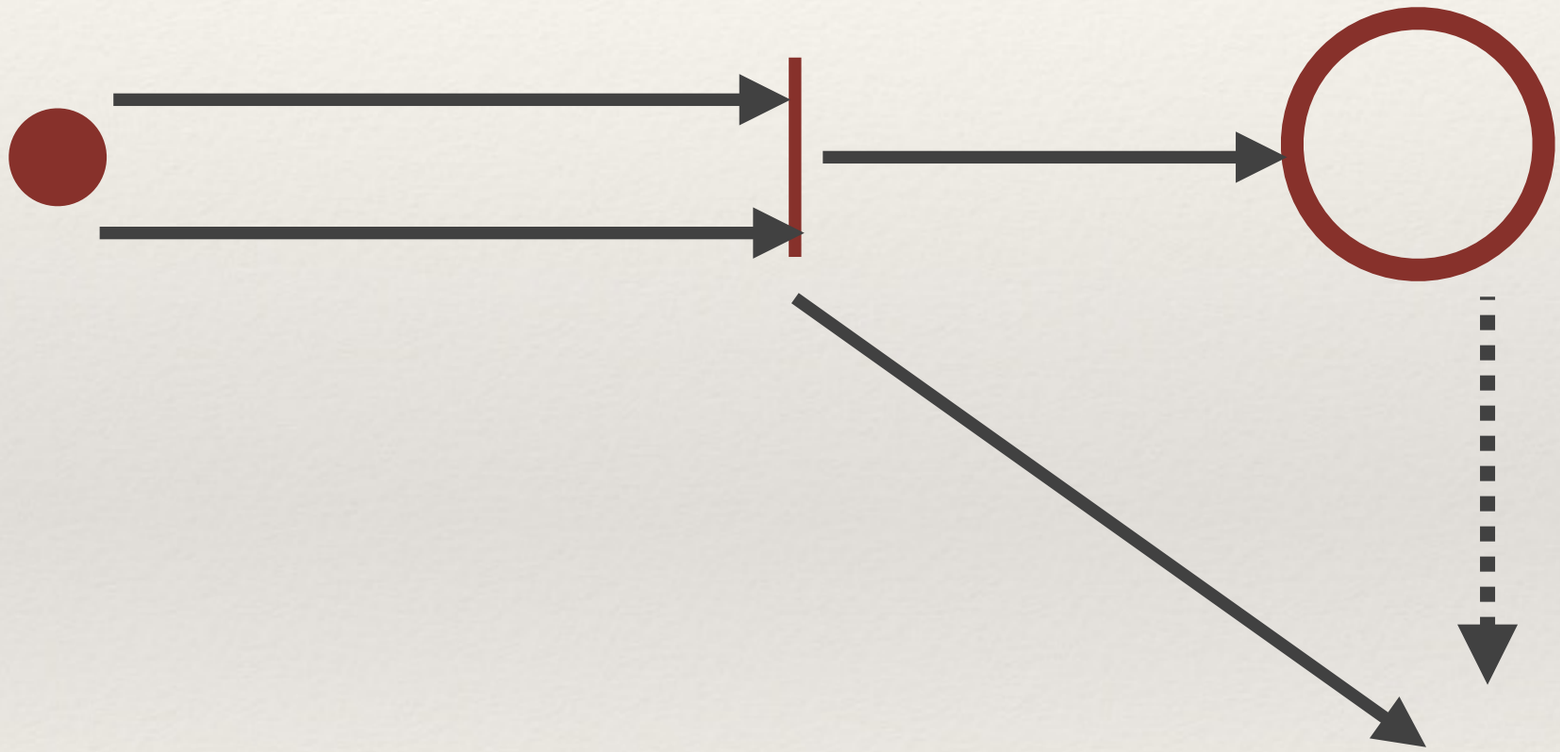
  Coeq(M,0) is the cokernel — aka the left null space.

  I.e. x such that x*M = 0.

- http://blog.functorial.com/posts/2012-02-19-What-If-Haskell-Had-Equalizers.html

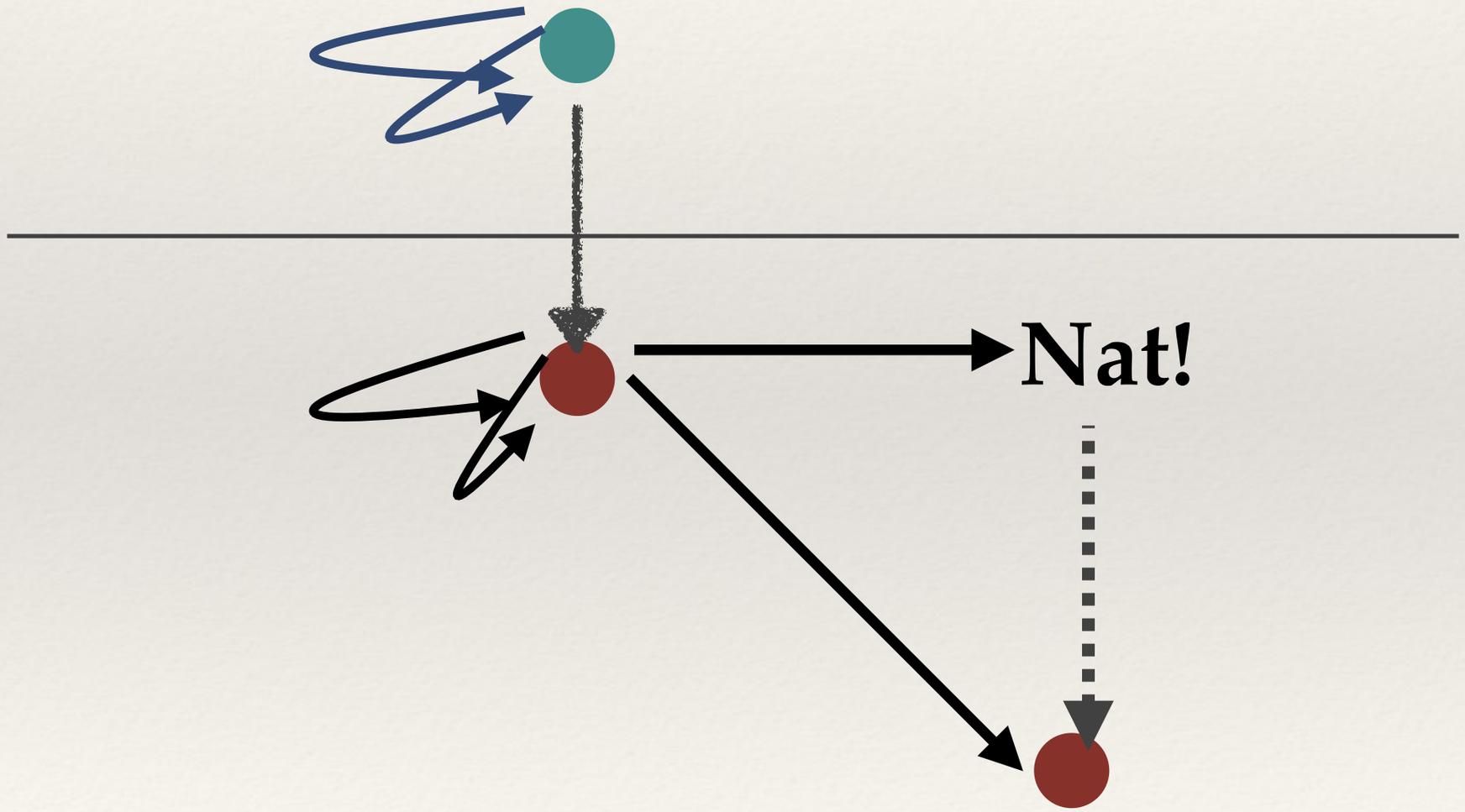
Some Coequalizers

- Topologically



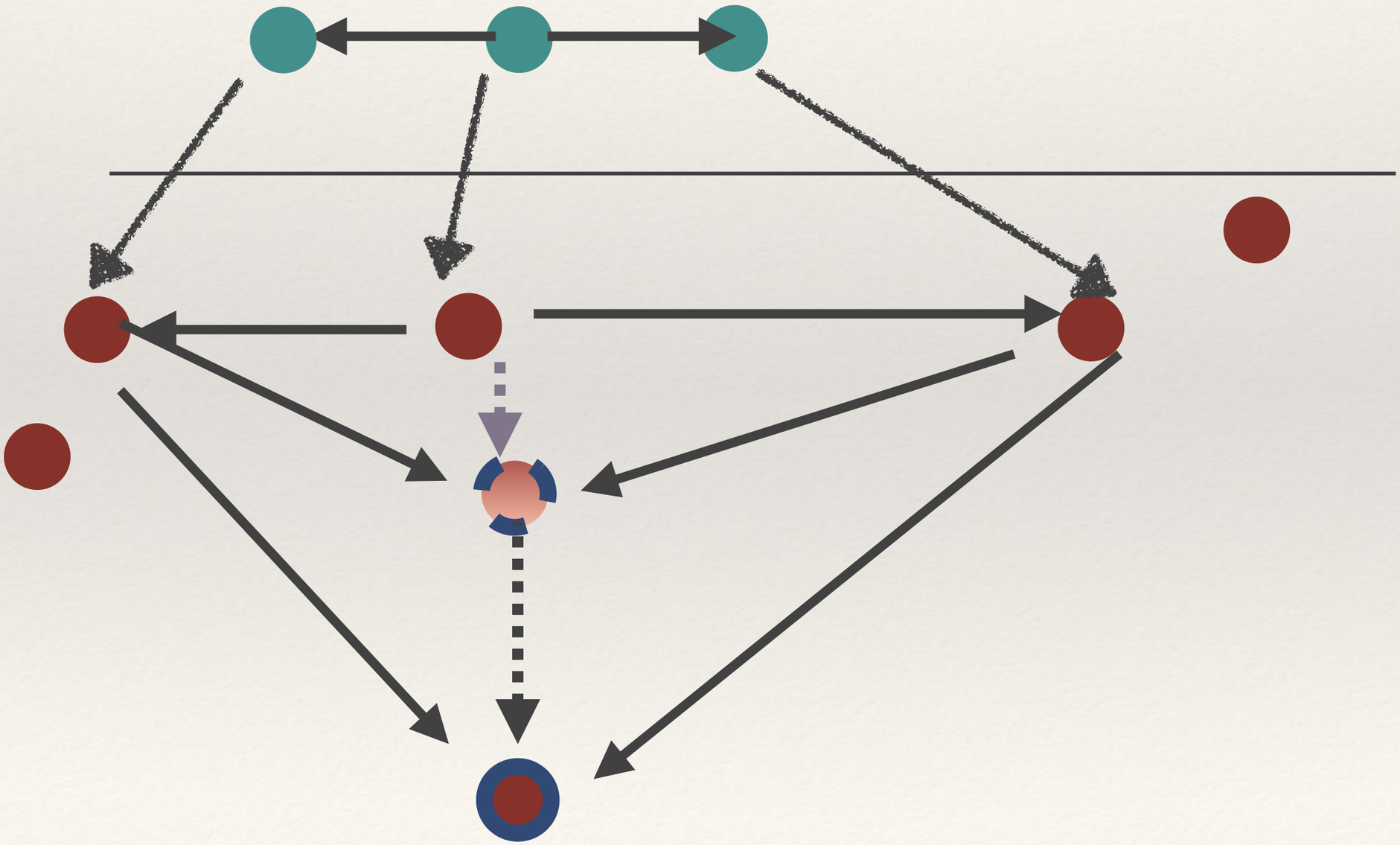
Parallel arrows on the same object

- Object with a loop and a constant arrow. (Result of Freyd)



Varieties of Colimits

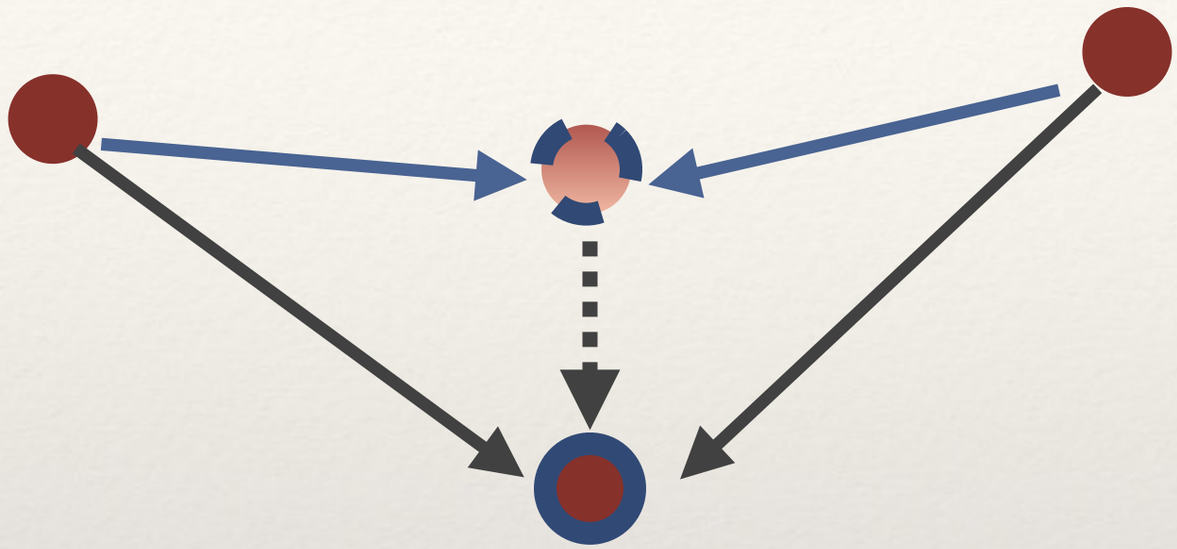
- Three objects in a *span* — Pushout



Cocompleteness

- Coproducts & Equalizers —> Pushouts
- Pushouts & Initial Object —> Coproducts & Equalizers
- either of these —> “All Finite Colimits”; i.e. co-complete

Categories to Haskell

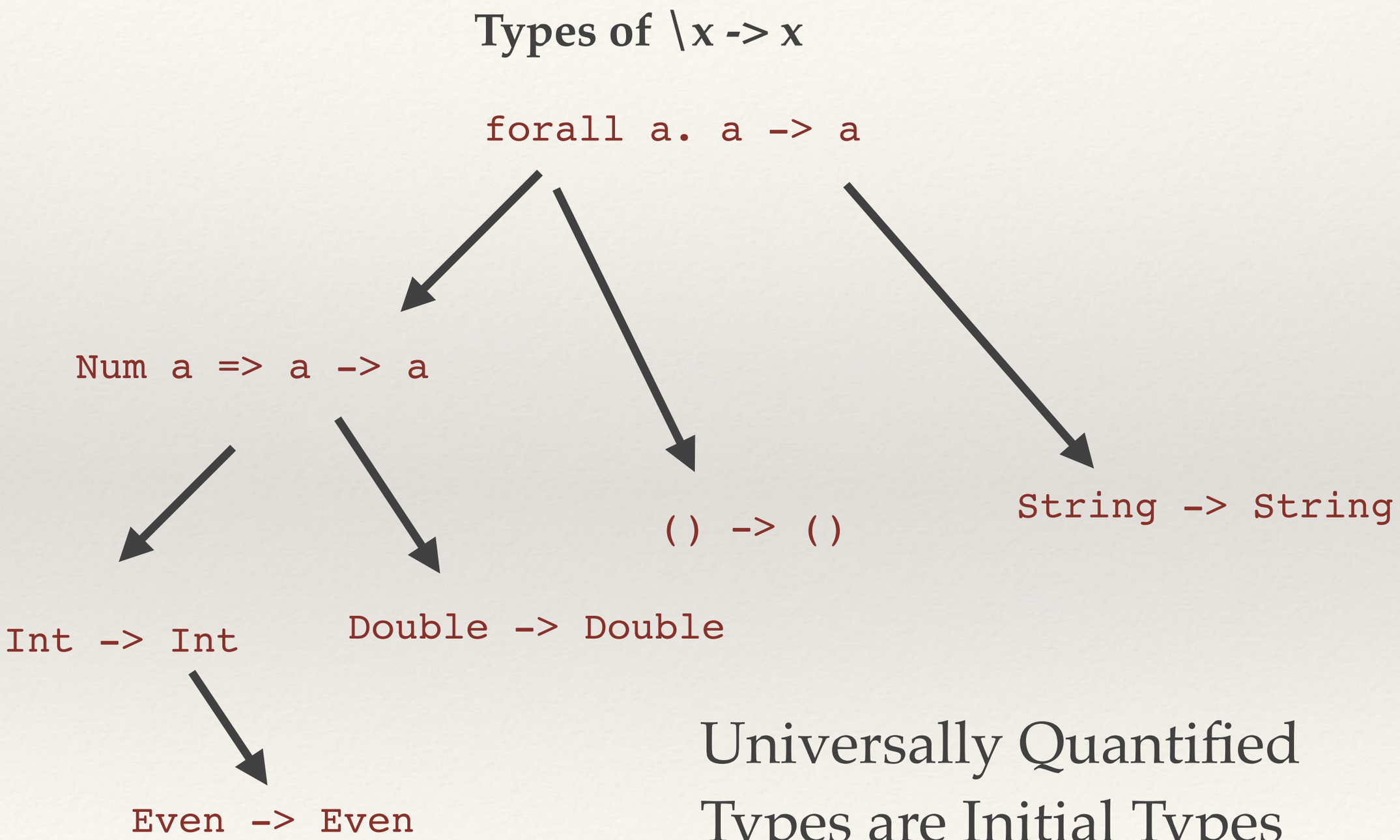


```
data ConeOverTwo a b t = CoT (a -> t) (b -> t)

data Colim f = Colim (forall t. f t -> t)

-- Colim (ConeOverTwo a b) === Either

data ConeOverZero t = CoZ

-- Colim ConeOverZero === Void
```

Universal Properties from Universal Quantification!

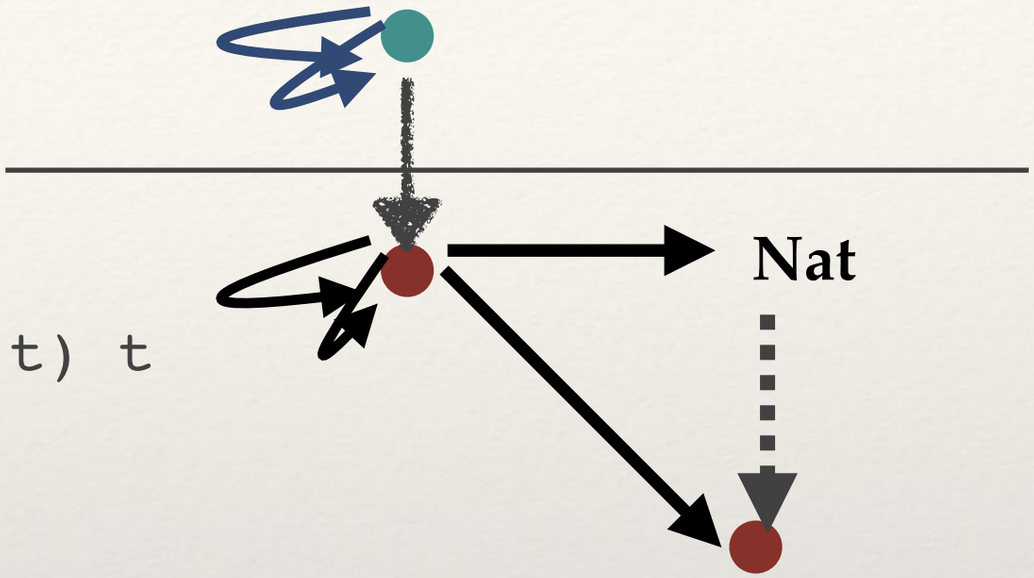
```
data ConeOverTwo a b t = CoT (a -> t) (b -> t)

data Colim f = Colim (forall t. f t -> t)

{-
either :: Either a b ->
     (a -> c) -> (b -> c) -> c

either2 :: Colim (ConeOverTwo a b) ->
     (a -> c) -> (b -> c) -> c
either2 (Colim h) f g = h (CoT f g)

Exercise — work the isomorphism through by hand.
-}
```

Why does this work!?



Universally Quantified
Types are Initial Types

Categories In Haskell

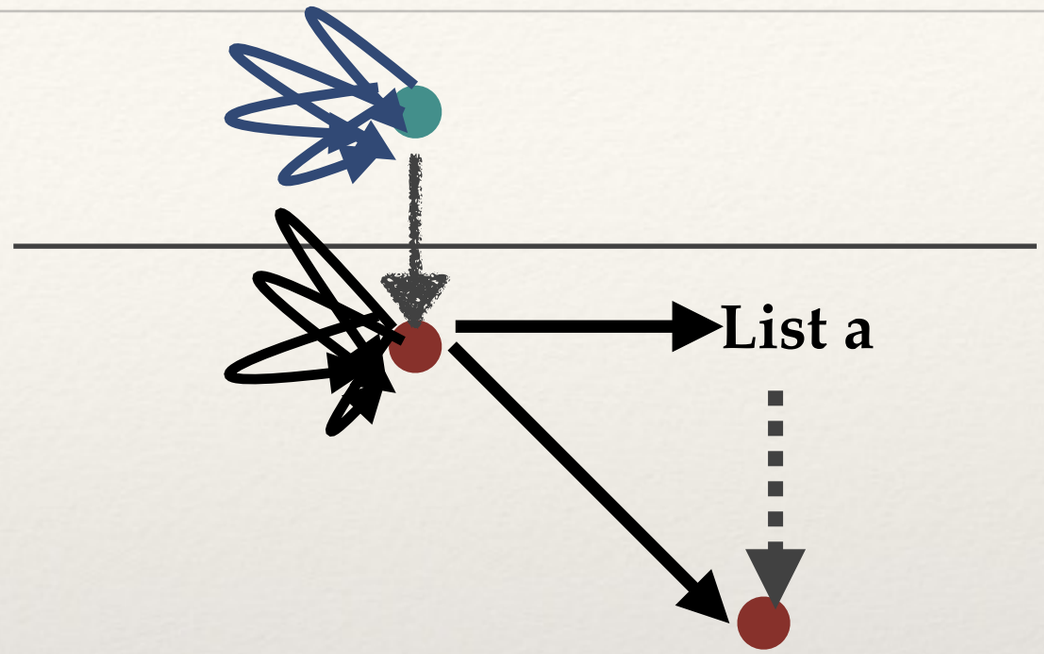



```
data ConeOverLoop t = CoL (t -> t) t

type Nat = Colim ConeOverLoop

fromN :: Nat -> Int
fromN (Colim f) = f (CoL succ 0)

toN :: Int -> Nat
toN n = Colim (\(CoL s z) -> last $ take (n+1) $ iterate s z)
```

Colimits ~= Initial Algebras ~= Church Encodings

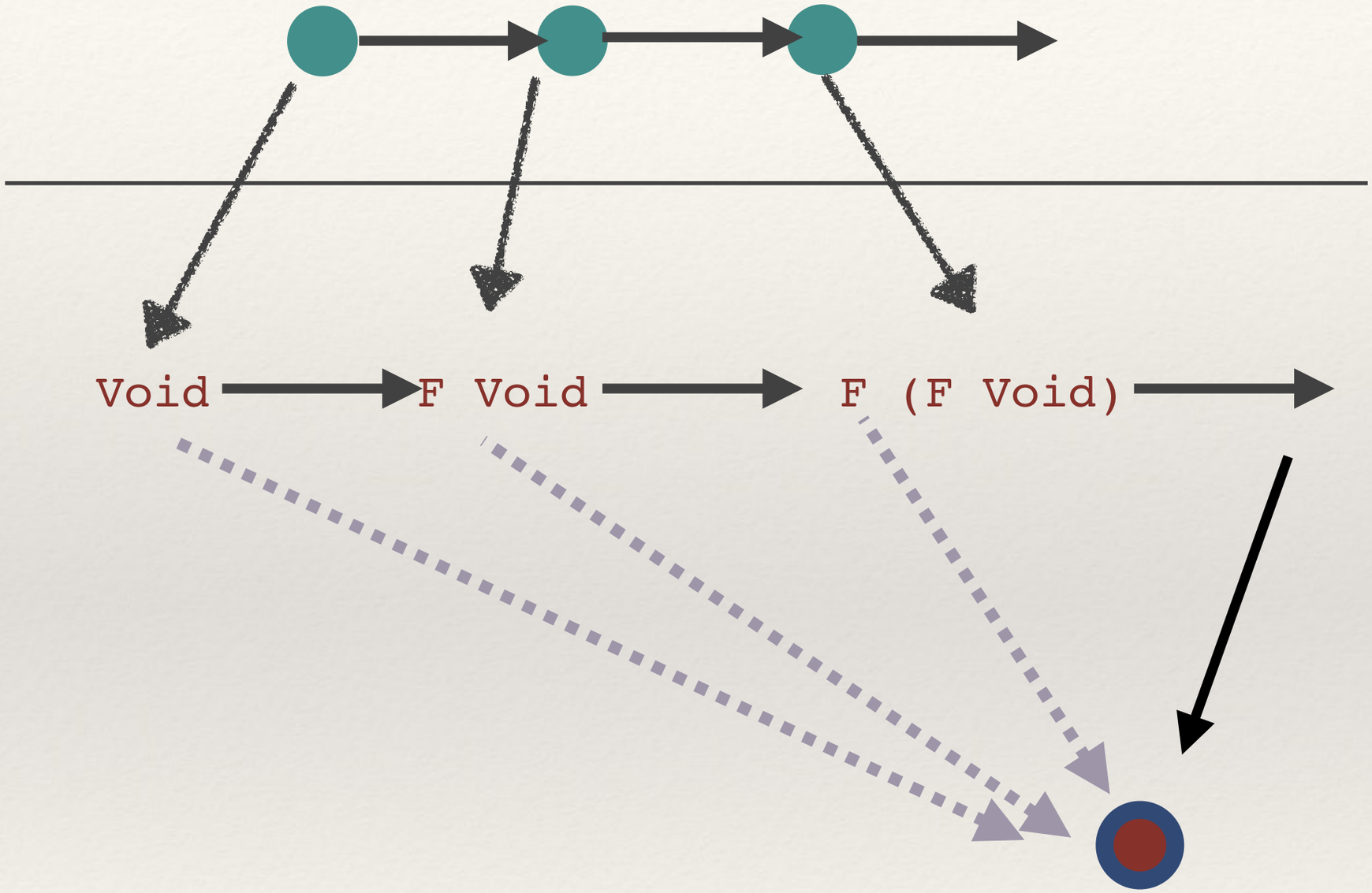
Data ~= Functions

Whoa

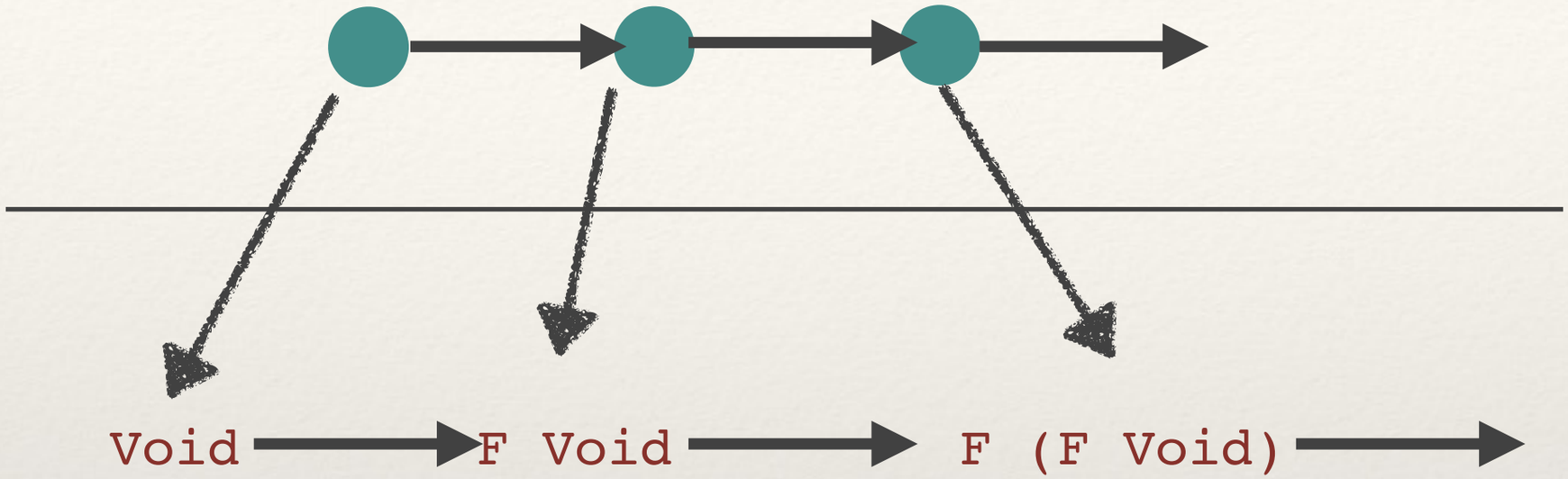



```
data IndexedConeOverLoop a t = ICoL (a -> t -> t) t

type List a = Colim (IndexedConeOverLoop a)

fromList :: List a -> [a]
fromList (Colim f) = f (ICoL (:) [])

toList :: [a] -> List a
toList xs = Colim (\(ICoL c z) -> foldr c z xs)
```

In Fact…



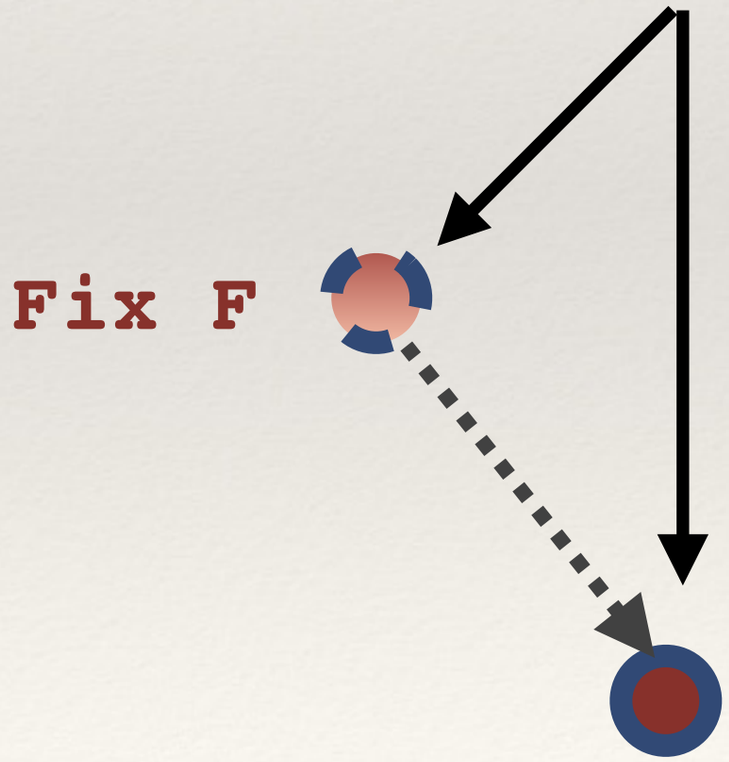
In Fact…



Adamek's Theorem:

Under certain conditions
(F is co-continuous, etc) then:

The colimit of the chain
induced by iteration of F
is the initial F algebra.



Thinking with Universal Properties

- Date recurrence rules (calendar appointments, meeting schedules, scheduled batch procedures and reports, scheduled bond payments)

- `data Sched = Daily | Weekly [1-7] | MonthlyAbsolute [1-31] | MonthlyRelative [1-7] [1-4] | JointSchedule Sched Sched | …`

- `interpSched :: Sched -> Day -> Day`

- `interpSched :: Sched -> Day -> Nat`

- `type GenSched = (Day -> Nat)`

- `data GenSched = GenSched (Day -> Maybe (Nat, GenSched))`

Thinking with Universal Properties

- `data GenSched = GenSched (Day -> Maybe (Nat, GenSched))`
- `data GenSched a = GenSched (a -> Maybe (Nat, GenSched a))`
- … by universal nonsense …
- `type GenSched a = a -> [Nat]`
- A universal schedule representation.

Clojure's Transducers

- `type Reducer a = forall z. (a -> z -> z)`
- `type Transducer a b = forall z. (a -> z -> z) -> (b -> z -> z)`

- … By abstract nonsense

- `Transducer a b === b -> [a]`
- (https://oleksandrmanzyuk.wordpress.com/2014/08/09/transducers-are-monoid-homomorphisms/

  http://tel.github.io/posts/typing-transducers/)

Just One More Thing…

- Take some category C, now look at "things 'containing' C but that have all colimits." Now take the initial such thing… what is it?

The Yoneda Embedding

Extreme Haskell

Take it to the (co-)Limit!