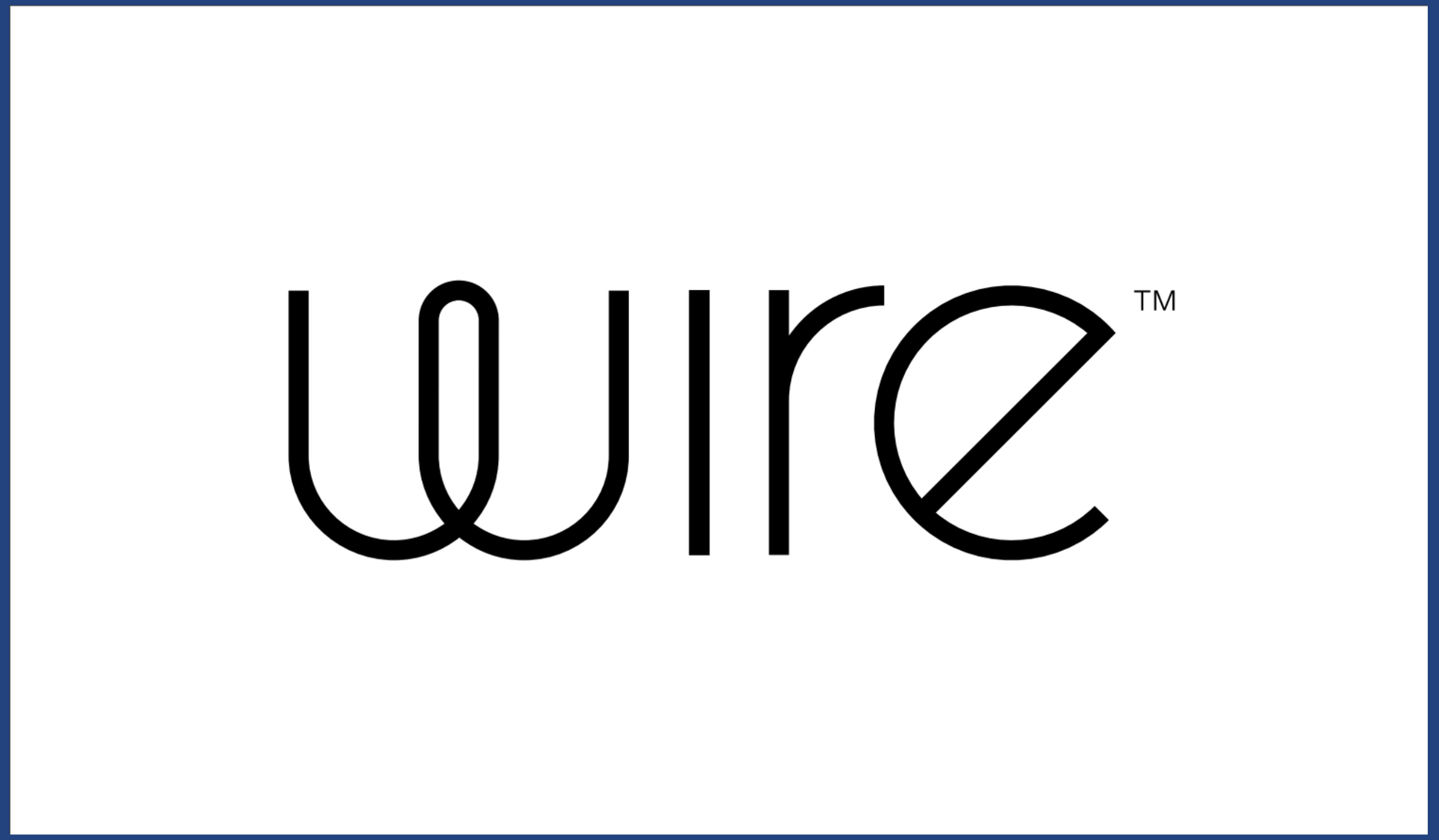
ENV COMONAD



Hiring haskell devs/devops

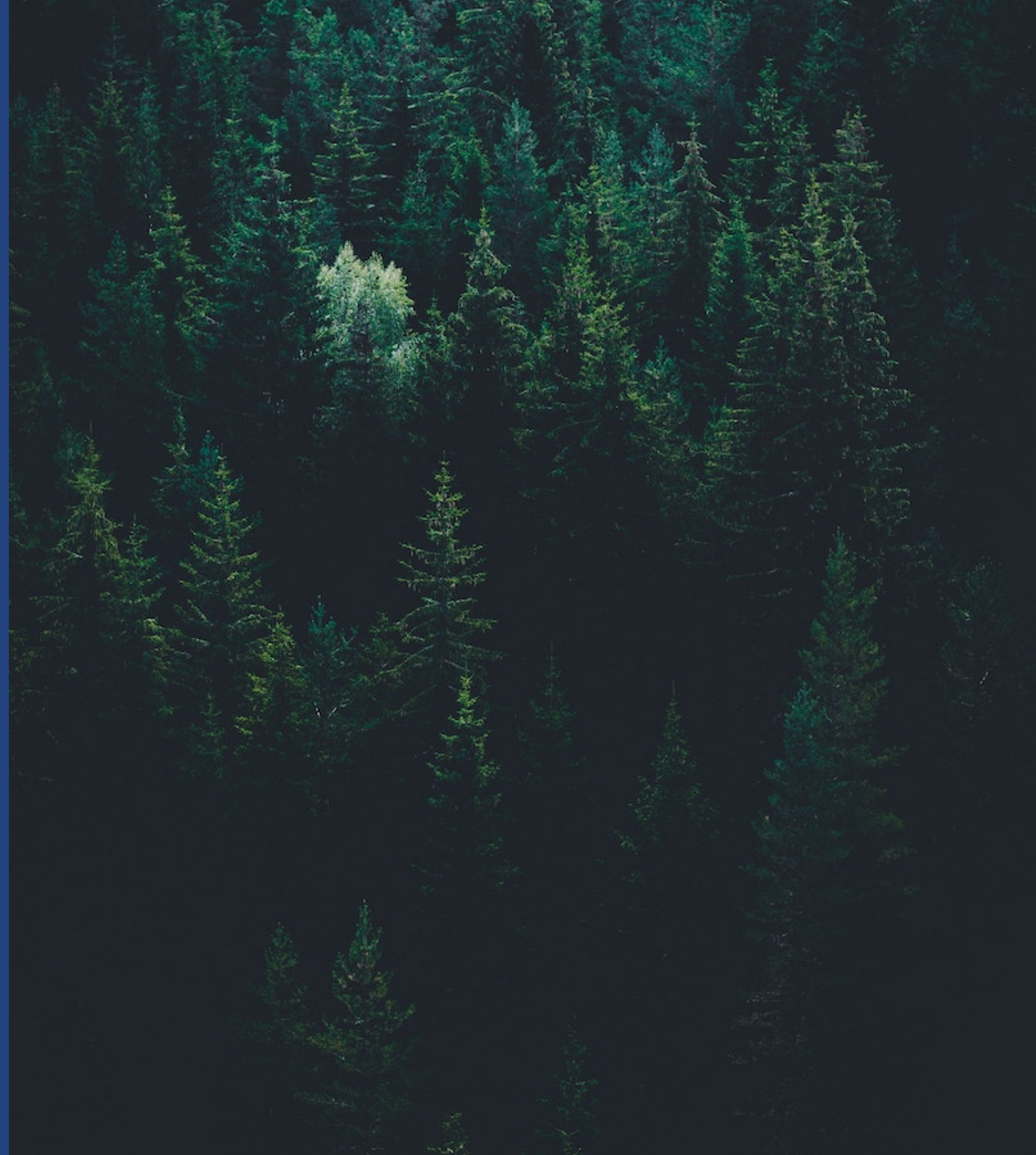
wire.com/jobs

ENV

HAS A SINGLE SLOT

CAN VIEW PROVIDED CONTEXT

READER MONAD BUT OBJECT ORIENTED



ENV A.K.A. CO-READER

a.k.a. (e, a)

```
data Env e a = Env e a
    deriving (Eq, Show, Functor)
```

ENV A.K.A. CO-READER

```
extract :: Env e a -> a
extract   (Env _ a) = a
```

ENV A.K.A. CO-READER

```
duplicate :: Env e a -> Env e (Env e a)
duplicate (Env e a) = ???
```

ENV A.K.A. CO-READER

```
duplicate :: Env e a -> Env e (Env e a)
duplicate (Env e a) = Env e (Env e a)
```

ENV A.K.A. CO-READER

```
extend :: (Env e a -> b)
       -> Env e a
       -> Env e b
extend f  (Env e a) = ???
```

ENV A.K.A. CO-READER

```
extend :: (Env e a -> b)
       -> Env e a
       -> Env e b
extend f  (Env e a) = Env e (f (Env e a))
```

WE CAN QUERY OUR ENVIRONMENT USING ASK

ASK

```
ask :: Env e a -> e
ask (Env e _) = e

λ> ask (Env 42 "hello")
42
```

WE CAN SELECT A SUBSET OF OUR ENVIRONMENT

USING ASKS

```haskell
asks :: (e -> e') -> Env e a -> e'
asks f (Env e _) = f e

λ> asks fst (Env ("first", "second") 1337)
"first"

λ> asks snd (Env ("first", "second") 1337)
"second"
```

EXAMPLE

```
data Settings =
    Settings
    { padAmount :: Int
    , maxLength :: Int
    , padChar   :: Char
    } deriving (Show)
```

QUERIES USING ASKS

```
data Settings =
     Settings
     { padAmount :: Int
     , maxLength :: Int
     , padChar   :: Char
     } deriving (Show)

getPadChar :: Env Settings a -> Int
getPadChar w = asks padChar w
```

USING 'ASK' QUERIES

```
context :: Env Settings String
context =
  env (Settings{padAmount=3, maxLength=5, padChar='*'})
      "Hello World"

λ> getPadChar context
'*'
```

USING 'ASKS' DIRECTLY

```
context :: Env Settings String
context =
  env (Settings{padAmount=3, maxLength=5, padChar='*'})
      "Hello World"

λ> asks padAmount context
3
```

QUESTIONS?

LET'S WRITE SOME MORE COMPLEX QUERIES

```haskell
trunc :: Env Settings [a] -> [a]
trunc w =
    let mxLngth = asks maxLength w
    in take mxLngth (extract w)
```

```haskell
context :: Env Settings String
context =
  env (Settings{padAmount=3, maxLength=5, padChar='*'})
      "Hello World"

λ> trunc context
"Hello"
```

```haskell
pad :: Env Settings String -> String
pad w =
    let padAmt = asks padAmount w
        c      = asks padChar w
     in    replicate padAmt c
        <> extract w
        <> replicate padAmt c
```

```
context =
  env (Settings{padAmount=3, maxLength=5, padChar='*'})
       "Hello World"

λ> pad context
"***Hello World***"
```

CHAINING ENV QUERIES INTO PIPELINES

MUTATIONS AREN'T SO INTERESTING FOR ENV
BUT WE WANT TO CARRY OUR CONTEXT
THROUGH A SERIES OF COMPUTATIONS

JUST LIKE READER

REMEMBER (=>=)

```
(=>=) :: (w a -> b)
      -> (w b -> c)
      -> (w a -> c)
```

=>=

CHAINS MANY QUERIES INTO ONE

```
pipeline :: Env Settings String -> String
pipeline = trunc =>= pad

λ> pipeline context
"***Hello***"
```

ORDER MATTERS

```
pipeline2 :: Env Settings String -> String
pipeline2 = pad =>= trunc

λ> pipeline2 context
"***He"
```

LOCAL

LETS US ALTER

THE ENVIRONMENT

FOR A QUERY

```
local :: (e -> e') -> Env e a -> Env e' a
```

WE CAN TEMPORARILY CHANGE SETTINGS WITHIN THE PIPELINE

```
pipeline3 :: Env Settings String -> String
pipeline3 = trunc =>= pad . local (setPadChar '_') =>= pad

λ> pipeline3 context
"***___Hello___***"
```

QUESTIONS?

BONUS

ABUSING DO-NOTATION

DO-NOTATION PROVIDES THE CONTEXT ON EVERY LINE

```haskell
pad :: Env Settings String -> String
pad w =
    let padAmt = asks padAmount w
        c      = asks padChar w
        txt    = extract w
        padding = replicate padAmt c
     in padding <> txt <> padding

pad' :: Env Settings String -> String
pad' = do
  padAmt <- asks padAmount
  c      <- asks padChar
  txt    <- extract
  let padding = replicate padAmt c
  return $ padding <> txt <> padding
```

ABUSING EVEN FURTHER

```haskell
pad'' :: Env Settings String -> String
pad'' = do
  padding <- replicate <$> asks padAmount <*> asks padChar
  txt     <- extract
  return $ padding <> txt <> padding
```

 CODE GOLF 

```
pad''' :: Env Settings String -> String
pad''' = do
  let padding = replicate <$> asks padAmount <*> asks padChar
  padding <> extract <> padding
```

chrispenner.ca

github.com/ChrisPenner

 @ChrisLPenner