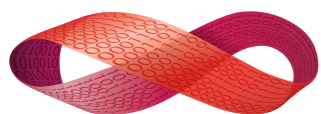


Building Single Page Web Applications with Purescript and Erlang

by @doppioslash

09/06/2017 - EUC2017 - Stockholm



Hi, I'm

Claudia Doppioslash

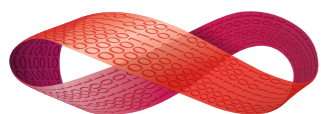
**Functional
Programmer**

&

**Game
Developer**

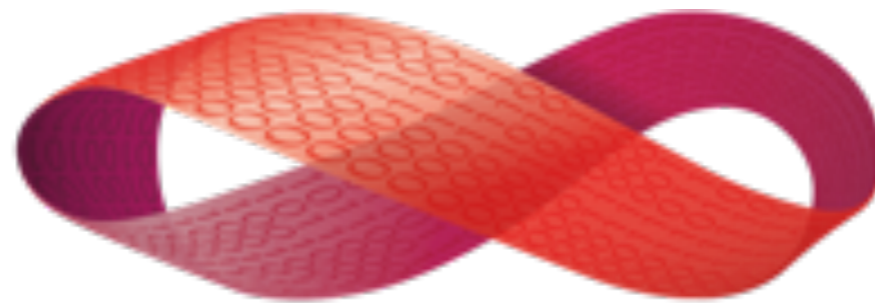
[@doppioslash](#)

www.lambdacat.com



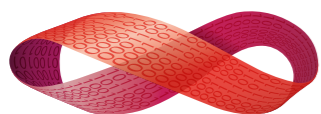
Peer Stritzinger GmbH

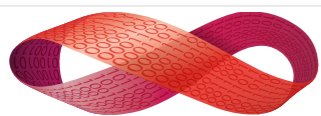
Functional and Failure Tolerant
Programming for Embedded,
Industrial Control and Automotive

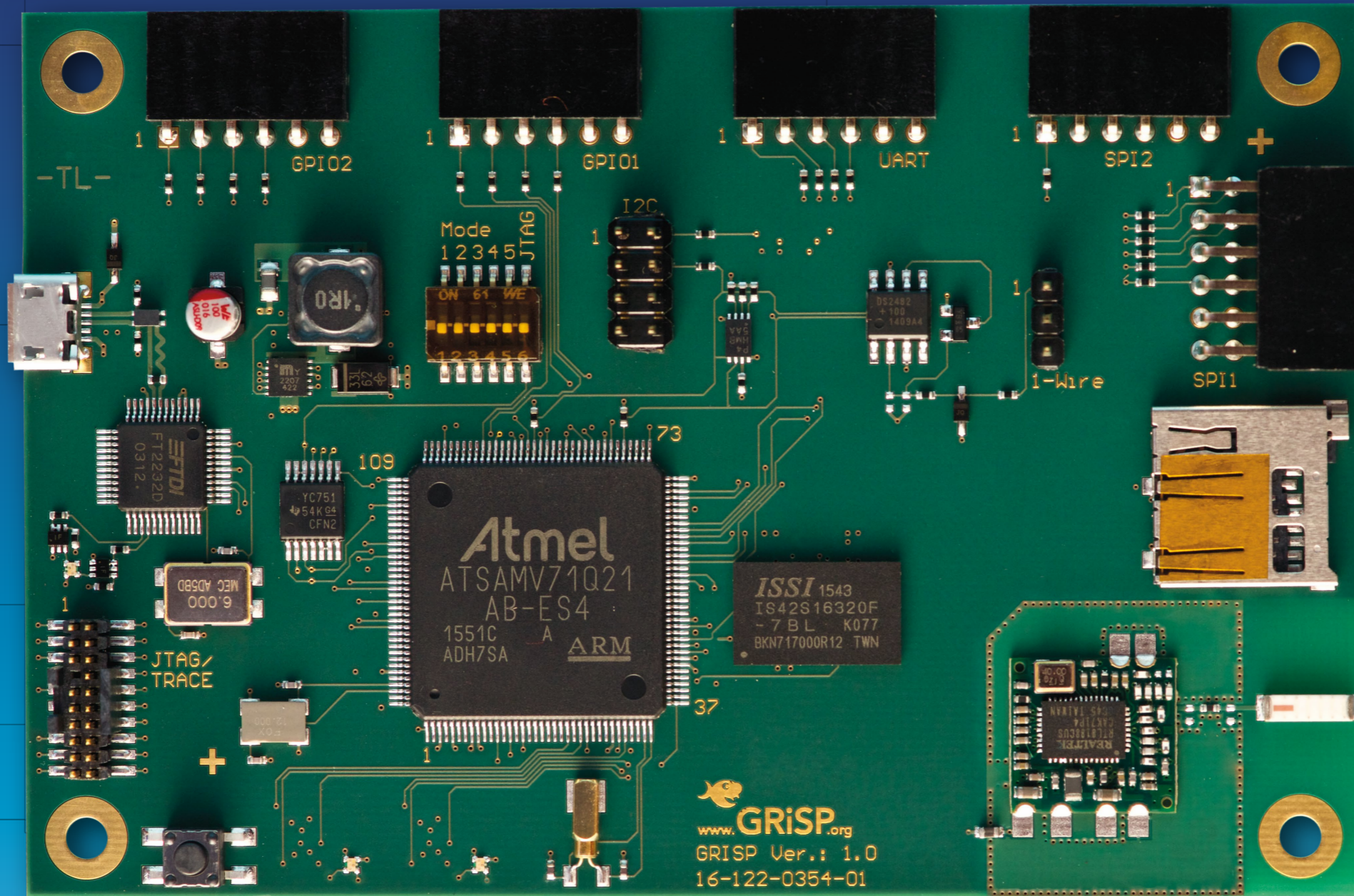


DIPL. PHYS. **PEER STRITZINGER** GMBH

www.stritzinger.com







www.grisp.org

 **LISP**
flavored
ERLANG



GRiSP

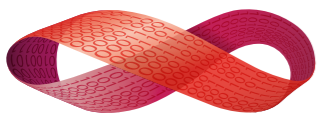


luerl

Why are you here?

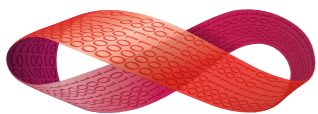
“I need to get some frontend code done,
and I hate Javascript”

Interested in Haskell-like languages



What are you getting

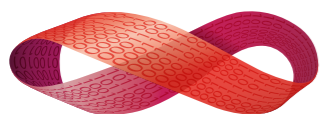
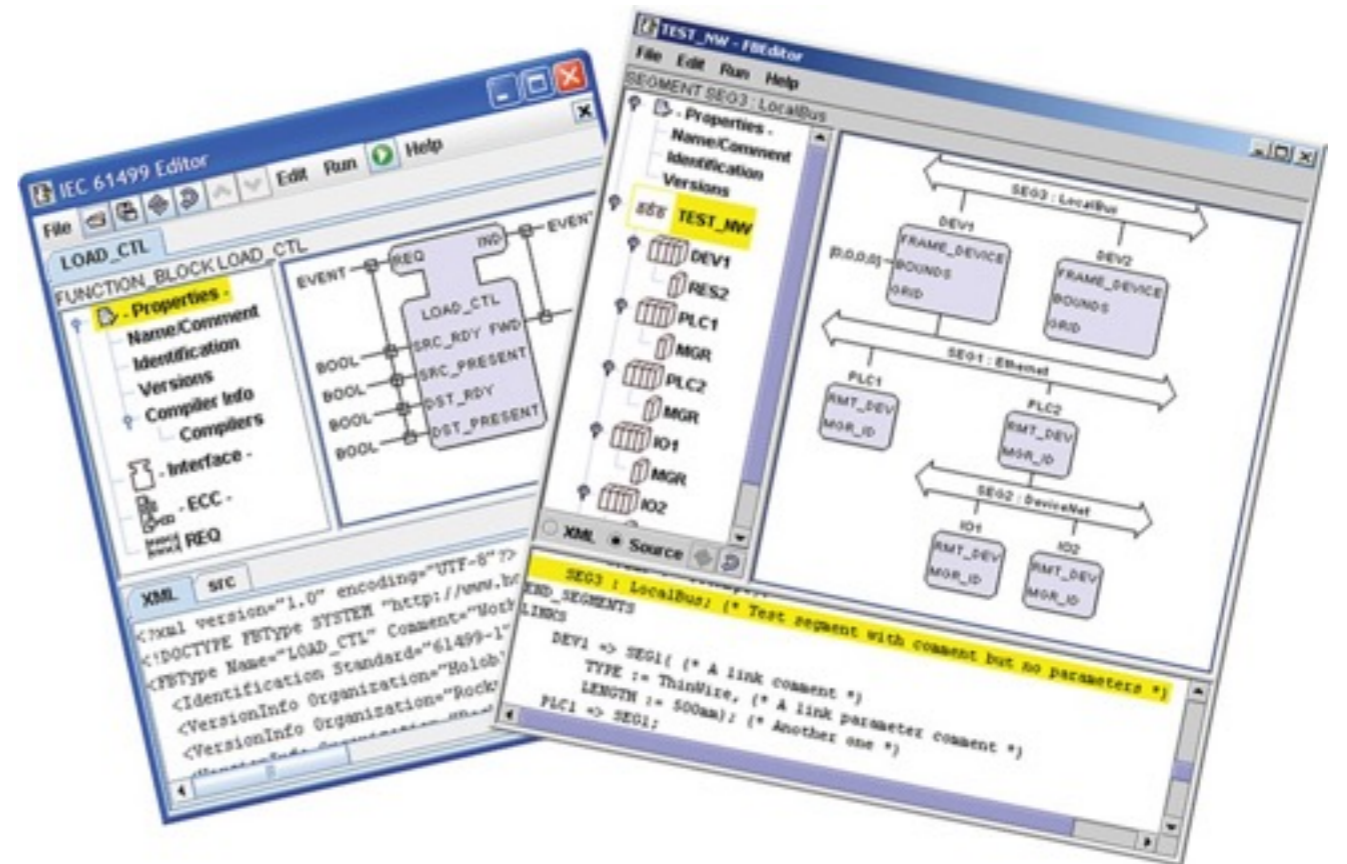
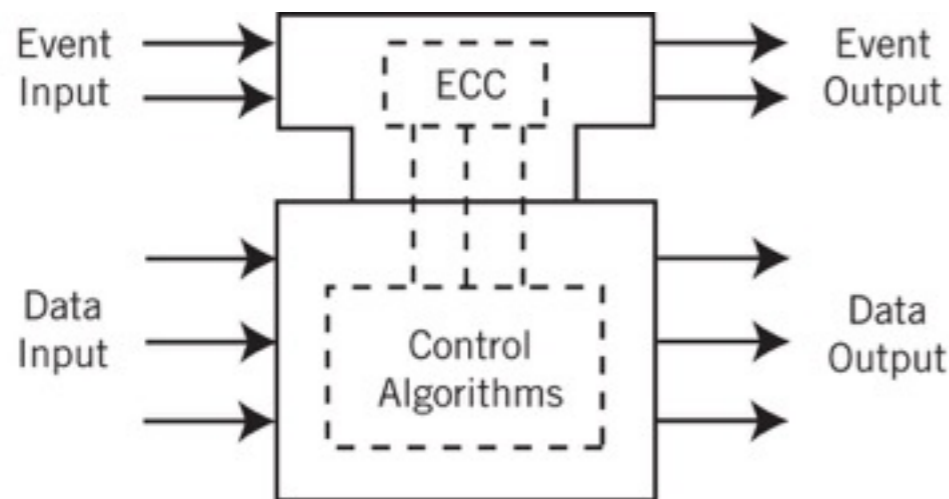
- choices of FP languages that target Javascript
- introduction to Purescript
- overview of framework choices
- thoughts from porting our project from Elm to Purescript



Our Project

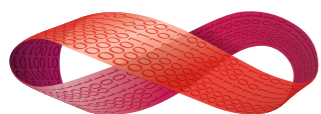
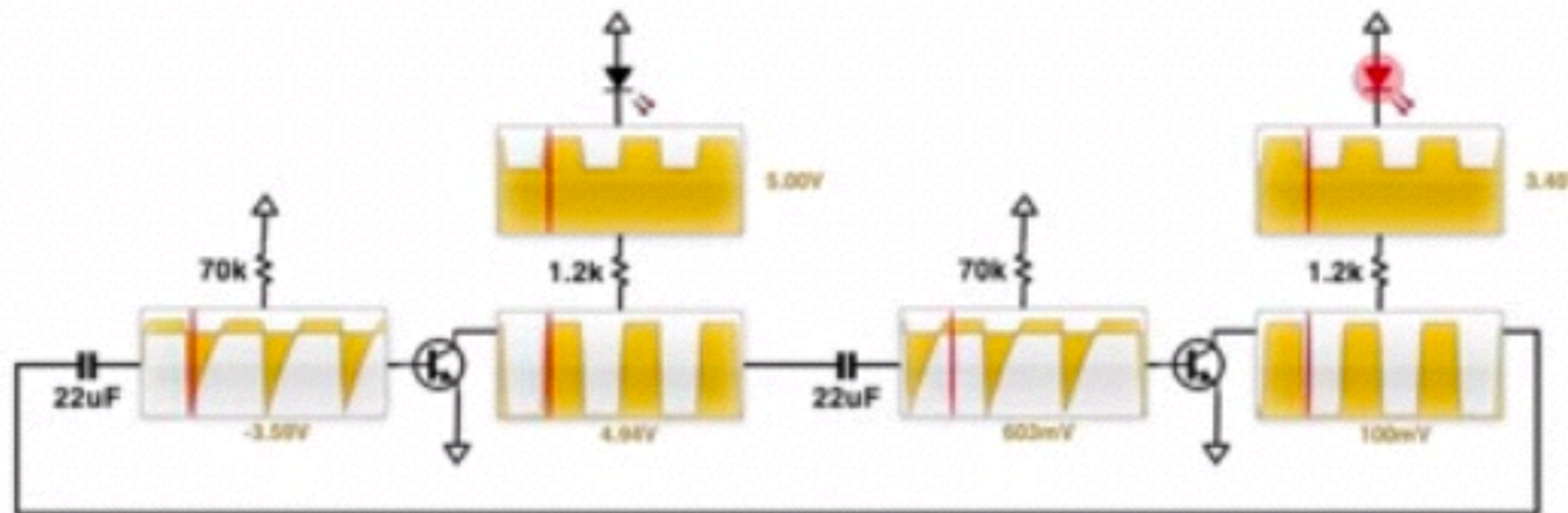
Visual IDE for PLC language **IEC61499**

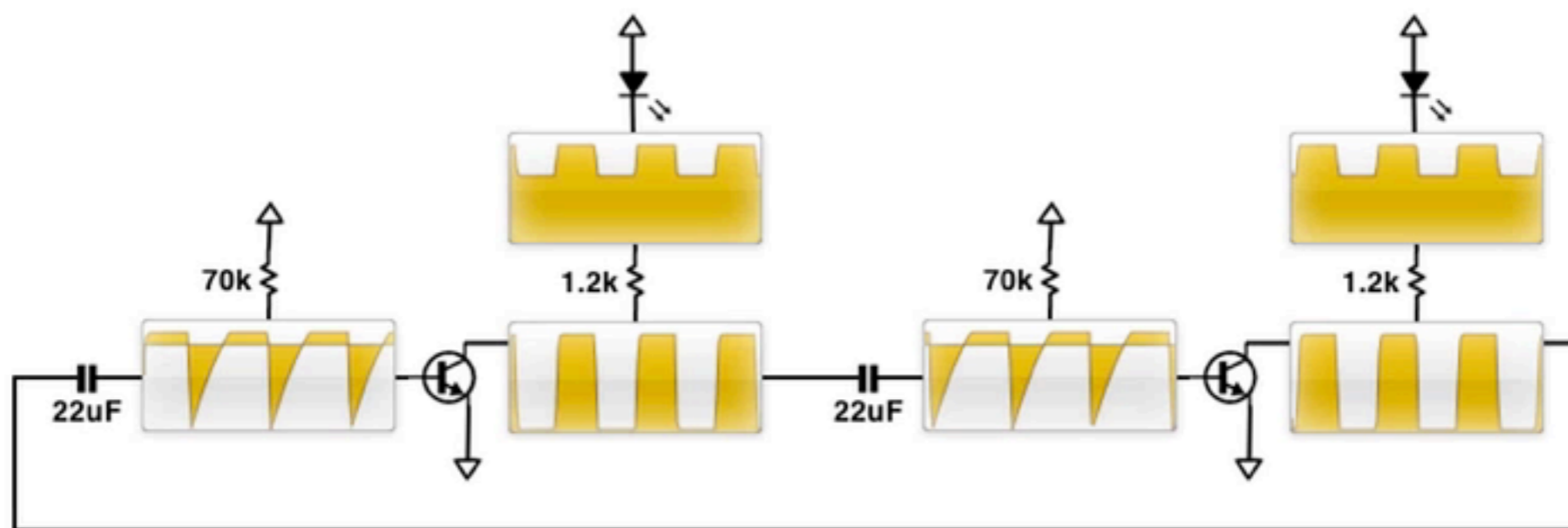
“A programmable logic controller, PLC, or programmable controller is a digital computer used for automation”



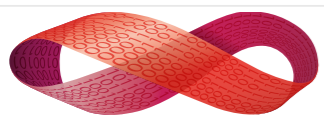
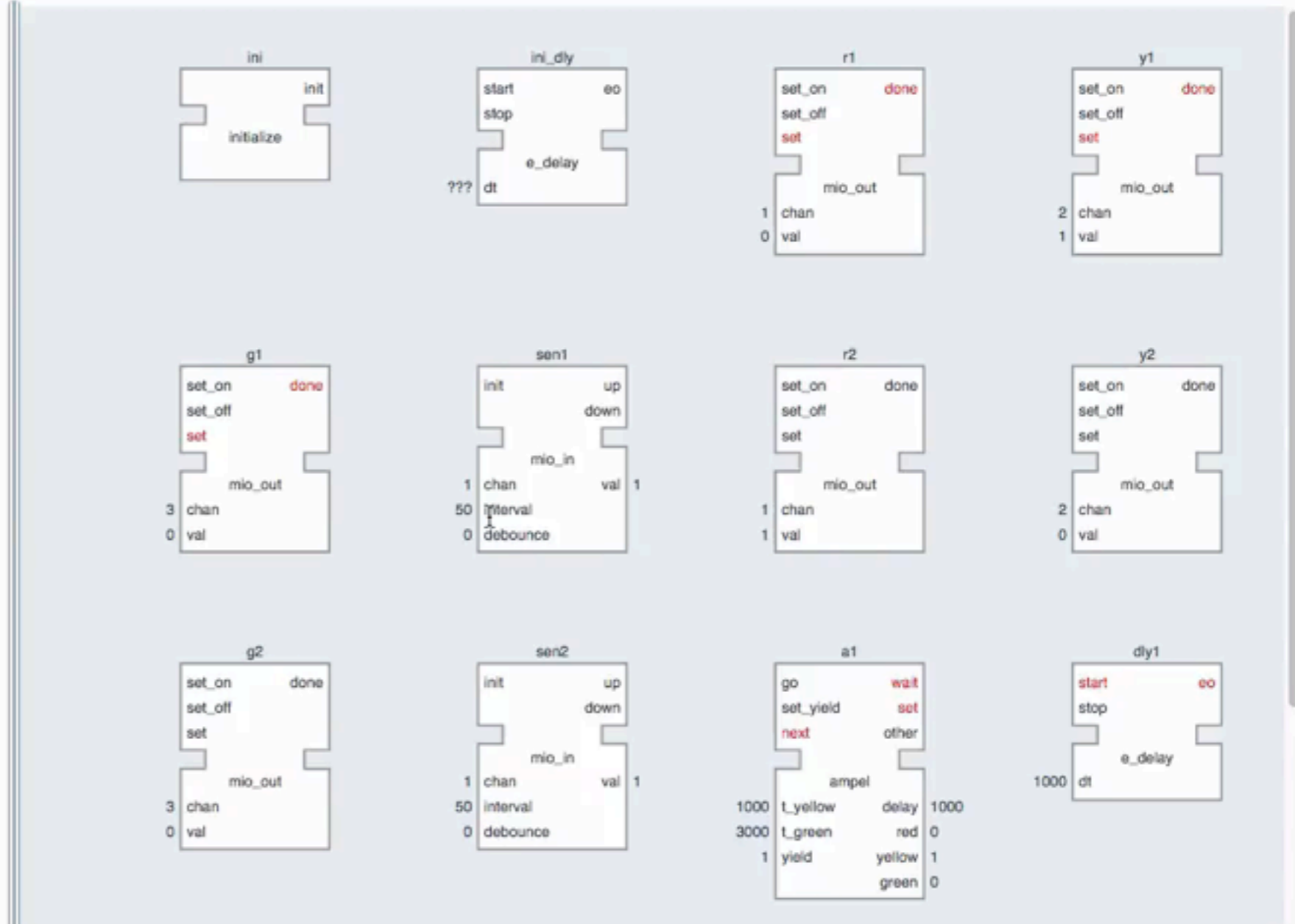
Our Project

Inspired by Bret Victor's "Inventing on Principle" talk
How visualising debugging helps

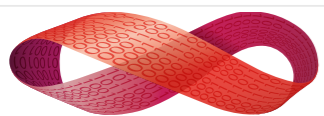




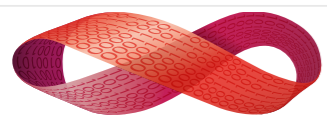
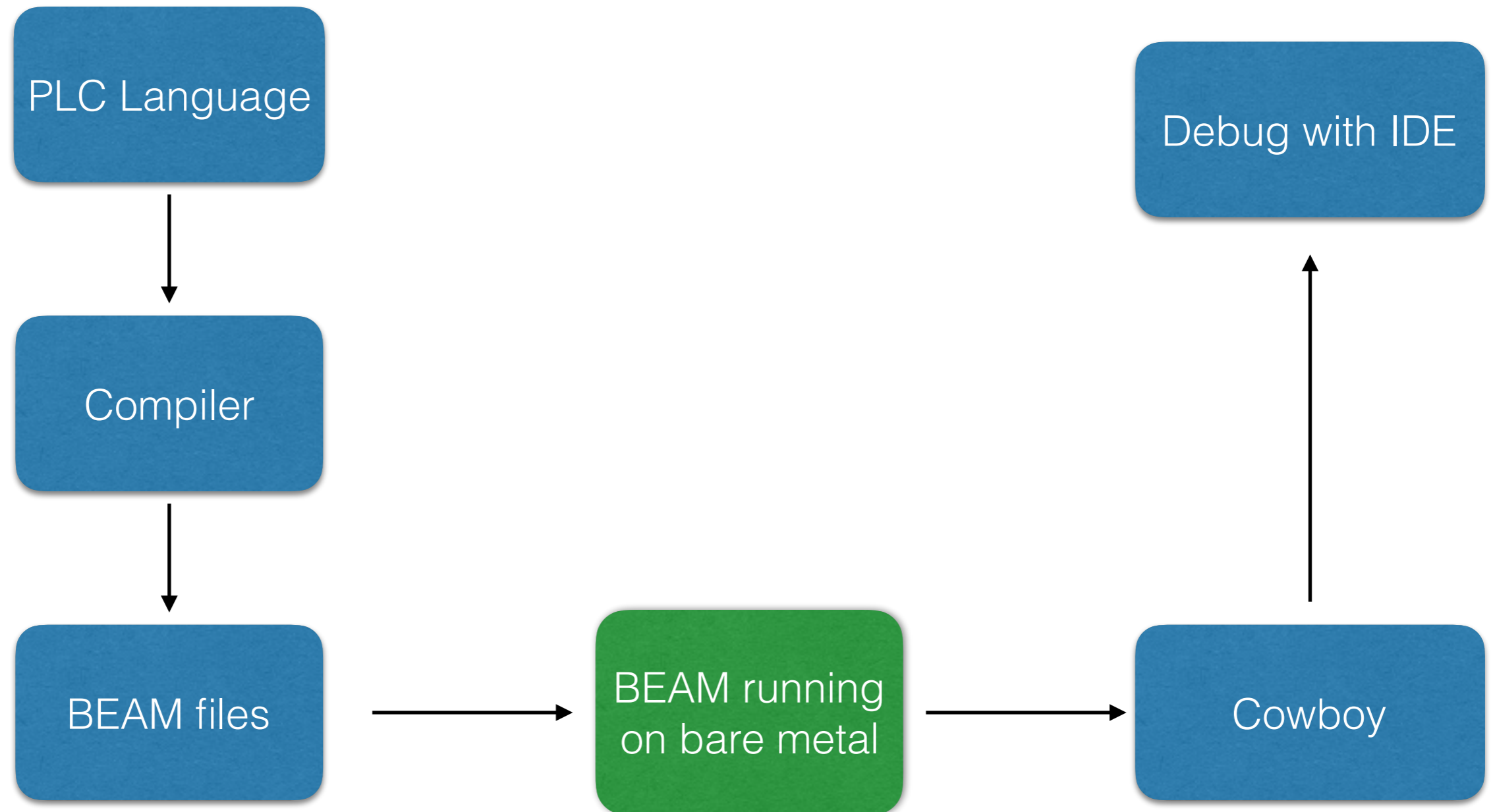
Our Project



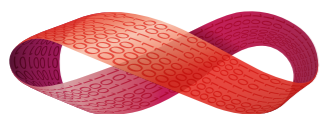
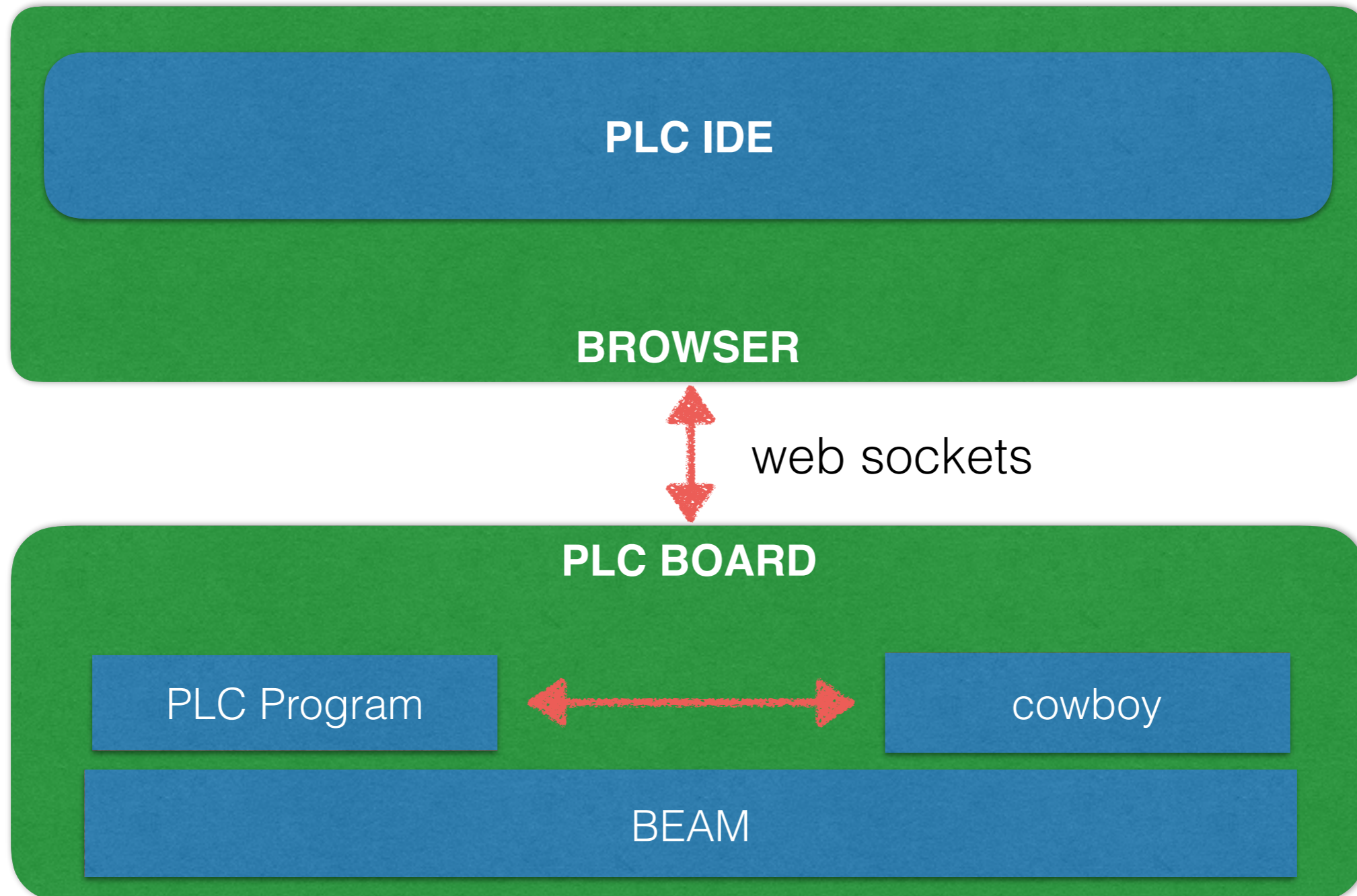
Demo



Deployment



Structure



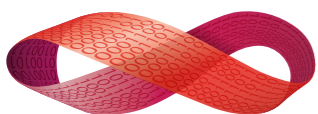
Requirements

Many platforms to support

All PC OSs & iPad Pro

Decent performance

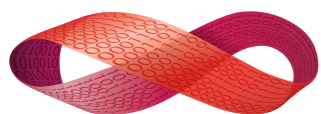
Needs to be interactive
~30fps should be fine



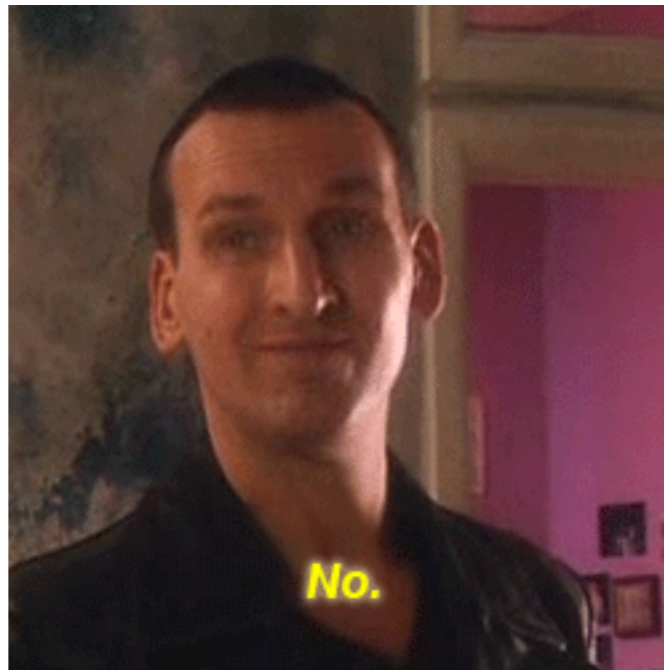
Frontend Tech Choice

Web Technologies because cross-platform

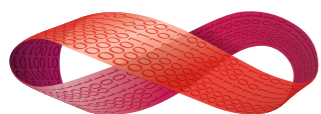
Hence: **Javascript, CSS, Svg**



Wait a minute, Javascript?



...let's not.



Some Possible Choices, Now

Ready now:

Bucklescript

Purescript



Clojurescript



Elm



CoffeScript



Typescript



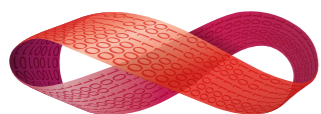
Reason



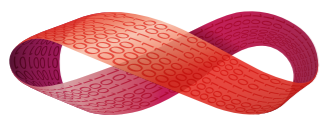
Fable



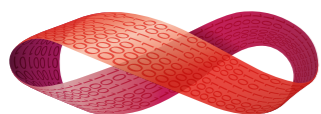
...and more...



They're breeding like rabbits!

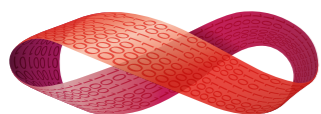


So many choices...





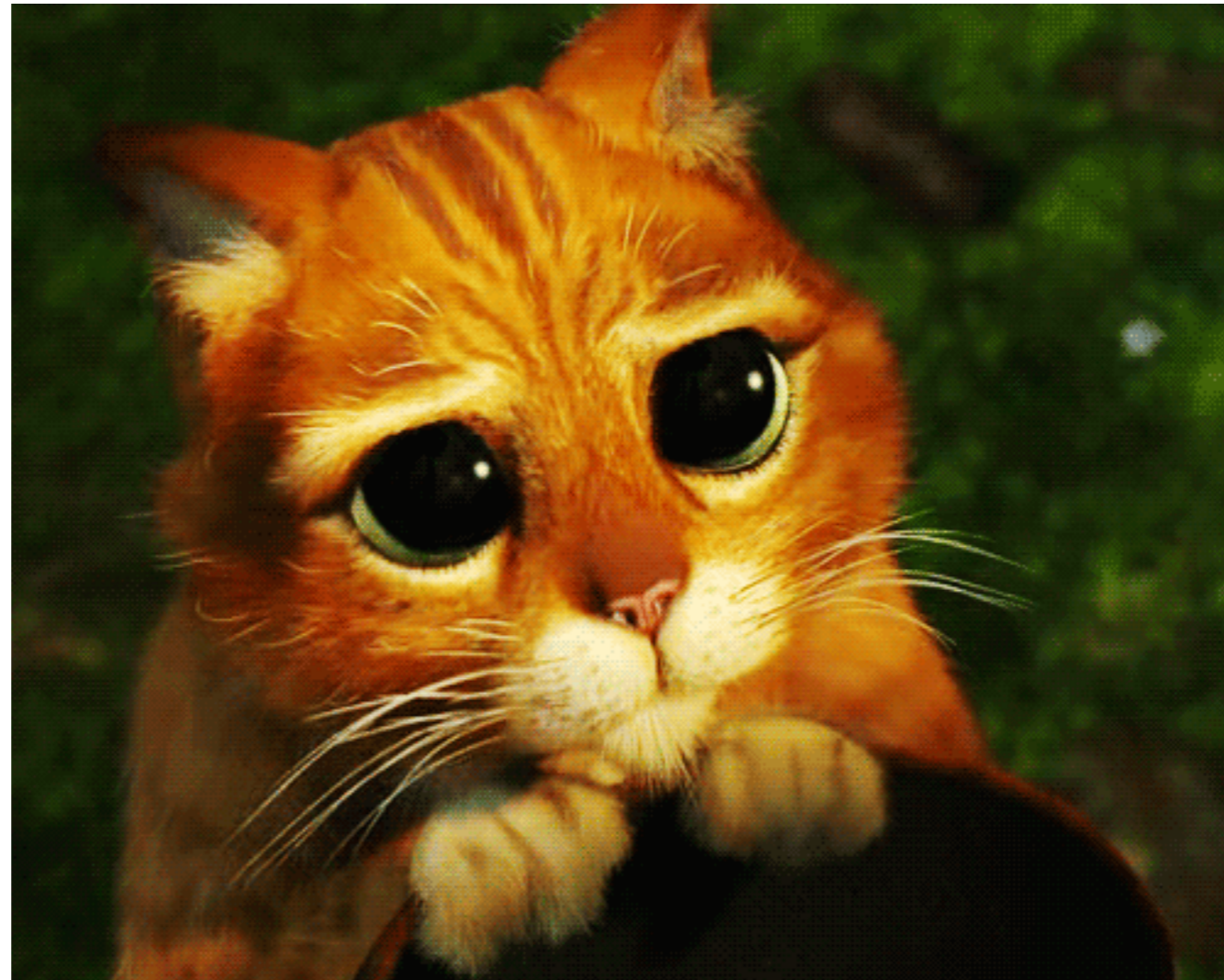
(...or you'll have to port this program again...)



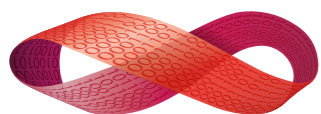
Our First Choice



: **“Please adopt me...”**



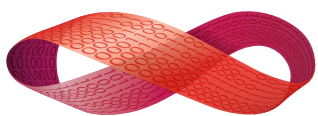
“...I swear I won't mention Monads”



Elm

Is known for:

- very helpful type errors
- opinionated
- a **pure and typed** language, but **simple**



The Elm tradeoff

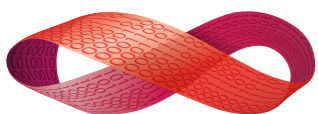
**Preferring simpler types
(unlike e.g. Haskell) begets:**

smooth learning curve
very helpful error messages

but also

more boilerplate
components?

abandoned Functional Reactive Programming

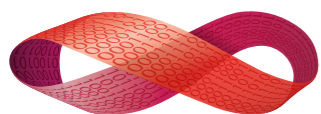


Our Second Choice

Purescript \Leftrightarrow : “Look into the Type Vortex...”



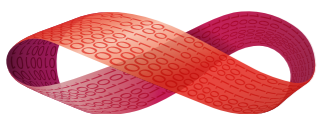
“...to gain Type superpowers (and possibly burn)”



What is Purescript?

Like Elm

Pure Functional
Strongly Typed
Eagerly evaluated
Compiles to Javascript
Advanced Types (Typeclasses, HKT)
Haskell-like syntax (with all the squiggles)
No runtime
Generates readable Javascript
Open community, a bit of a roadmap



Purescript Pros vs JS

If it compiles, it works (90% of the time)

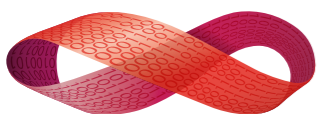
Confident **refactoring** (work in small steps)

Clean

Much fewer LOC

It has error messages

(certainly better than **undefined is not a function**)



Pros compared to Elm

Pursuit (search libs by type signature)

Clearer direction

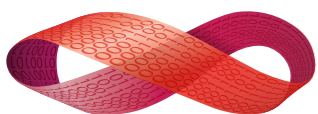
Can work a lot with REPL

Great workflow, (e.g. Type holes)

Many of the higher abstractions

Cons

Takes time to learn the higher abstractions



Pursuit

Pursuit

```
(a -> b) -> f a -> f b
```

Help

Search results

sensor

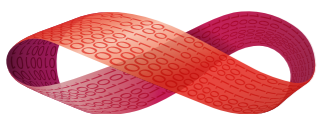
```
sensor :: forall w m a. MonadWriter w m => (w -> w) -> m a -> m a
```

Modify the final accumulator value by applying a function.

P purscript-transformers **M** Control.Monad.Writer.Class

liftA1

```
liftA1 :: forall f a b. Applicative f => (a -> b) -> f a -> f b
```



Elm Search (unofficial)



Elm Search

Search

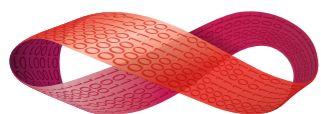
Showing results for: **(a -> b -> b) -> b -> List a -> b**

```
foldl : (a -> b -> b) -> b -> List a -> b
```

Reduce a list from the left.

[elm-lang/core/5.1.1](#)

List



Pros compared to Elm

Pursuit (search libs by type signature)

Clearer direction

Can work a lot with REPL

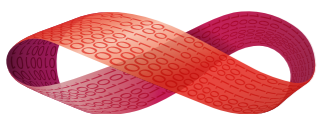
Great workflow, (e.g. Type holes)

Many of the higher abstractions

Cons

Takes time to learn the higher abstractions

Type errors are not as good as Elm



Milestones

purescript / purescript

Unwatch 153

Unstar 3,608

Fork 301

Code

Issues 144

Pull requests 14

Projects 1

Wiki

Insights

1.0

New Issue

No due date 85% complete

Improving error messages, mark DCE as experimental, imperative core improvements, purs bundle performance.

20 Open 118 Closed

[purs ide] Add a `do-nothing` response to the addImport command **psc-ide**

#2926 opened 2 days ago by KRITZCREEK

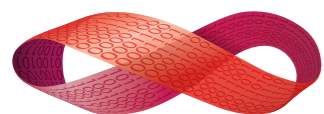
Generic.Rep doesn't derive inner records **bug** **typechecker**

#2911 opened 11 days ago by pkamenarsky

1

Rename classes in Prim **breaking** **modules**

#2903 opened 15 days ago by paf31



Pros compared to Elm

Pursuit (search libs by type signature)

Clearer direction

Can work a lot with REPL

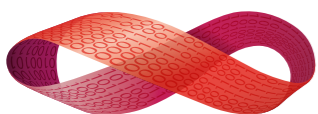
Great workflow, (e.g. Type holes)

Many of the higher abstractions

Cons

Takes time to learn the higher abstractions

Type errors are not as helpful as Elm



Pros compared to Elm

Pursuit (search libs by type signature)

Clearer direction

Can work a lot with REPL

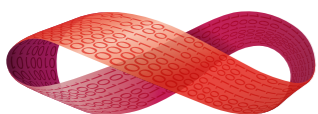
Great workflow, (e.g. Type holes)

Many of the higher abstractions

Cons

Takes time to learn the higher abstractions

Type errors are not as helpful as Elm



Type holes and Search

View Mode Backend Share Compile Show JS Help

```
1 module Main where
2
3 import Prelude
4 import Control.Monad.Eff.Console (log)
5 import Data.Array ((..))
6 import Data.Traversable (traverse)
7 import TryPureScript
8
9 fizzBuzz :: Int -> String
10 fizzBuzz =
11   ?help [ part "fizz" 3
12         , part "buzz" 5
13         ]
14   where
15     part s m n | n `mod` m == 0 = s
16               | otherwise = ""
17
18 main = render =<< withConsole do
19   traverse (log <<< fizzBuzz) (1 .. 100)
```

Hole 'help' has the inferred type

```
Array (Int -> String) -> Int -> String
```

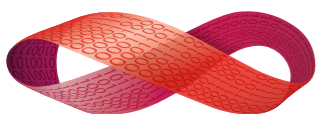
You could substitute the hole with one of these values

```
Data.Foldable.fold           :: forall f m. Foldable t => f -> m -> t -> m
Data.Monoid.mempty          :: forall m. Monoid m => m
Unsafe.Coerce.unsafeCoerce :: forall a b. a -> b
```

in the following context:

```
part :: String -> Int -> Int -> String
```

n value declaration fizzBuzz



Type holes and Search

View Mode Backend Share Compile Show JS Help

```
1 module Main where
2
3 import Prelude
4 import Control.Monad.Eff.Console (log)
5 import Data.Array ((..))
6 import Data.Traversable (traverse)
7 import TryPureScript
8
9 fizzBuzz :: Int -> String
10 fizzBuzz =
11   ?help [ part "fizz" 3
12         , part "buzz" 5
13         ]
14   where
15     part s m n | n `mod` m == 0 = s
16               | otherwise = ""
17
18 main = render =<< withConsole do
19   traverse (log <<< fizzBuzz) (1 .. 100)
```

Hole 'help' has the inferred type

```
Array (Int -> String) -> Int -> String
```

You could substitute the hole with one of these values

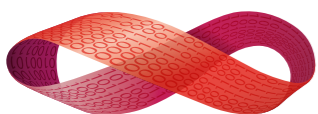
```
Data.Foldable.fold           :: forall f m. Foldable t => f -> m -> t -> m
Data.Monoid.mempty           :: forall m. Monoid m => m
Unsafe.Coerce.unsafeCoerce  :: forall a b. a -> b
```

in the following context:

```
part :: String -> Int -> Int -> String
```

n value declaration fizzBuzz

Type Hole



Type holes and Search

View ModeBackendShare Compile Show JSInferred TypeHelp

```
1 module Main where
2
3 import Prelude
4 import Control.Monad.Eff.Console (log)
5 import Data.Array ((..))
6 import Data.Traversable (traverse)
7 import TryPureScript
8
9 fizzBuzz :: Int -> String
10 fizzBuzz =
11   ?help [ part "fizz" 3
12         , part "buzz" 5
13         ]
14   where
15     part s m n | n `mod` m == 0 = s
16               | otherwise = ""
17
18 main = render =<< withConsole do
19   traverse (log <<< fizzBuzz) (1 .. 100)
```

Hole 'help' has the inferred type

```
Array (Int -> String) -> Int -> String
```

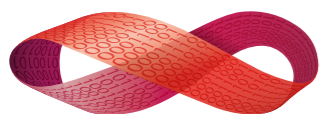
You could substitute the hole with one of these values:

```
Data.Foldable.fold           :: forall f m. Foldable t => f -> m -> t -> m
Data.Monoid.mempty          :: forall m. Monoid m => m
Unsafe.Coerce.unsafeCoerce :: forall a b. a -> b
```

in the following context:

```
part :: String -> Int -> Int -> String
```

n value declaration fizzBuzz



Type holes and Search

View Mode Backend Share Compile Show JS Help

```
1 module Main where
2
3 import Prelude
4 import Control.Monad.Eff.Console (log)
5 import Data.Array ((..))
6 import Data.Traversable (traverse)
7 import TryPureScript
8
9 fizzBuzz :: Int -> String
10 fizzBuzz =
11   ?help [ part "fizz" 3
12         , part "buzz" 5
13         ]
14   where
15     part s m n | n `mod` m == 0 = s
16               | otherwise = ""
17
18 main = render =<< withConsole do
19   traverse (log <<< fizzBuzz) (1 .. 100)
```

Hole 'help' has the inferred type

```
Array (Int -> String) -> Int -> String
```

You could substitute the hole with one of these values

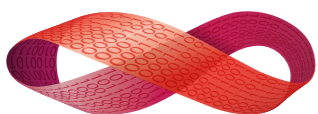
```
Data.Foldable.fold           :: forall f m. Foldable t => f -> m -> t -> m
Data.Monoid.mempty           :: forall m. Monoid m => m
Unsafe.Coerce.unsafeCoerce  :: forall a b. a -> b
```

Suggested Functions

in the following context:

```
part :: String -> Int -> Int -> String
```

n value declaration fizzBuzz



Pros compared to Elm

Pursuit (search libs by type signature)

Clearer direction

Can work a lot with REPL

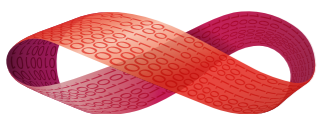
Great workflow, (e.g. Type holes)

Many of the higher abstractions

Cons

Takes time to learn the higher abstractions

Type errors are not as helpful as Elm



Pros compared to Elm

Pursuit (search libs by type signature)

Clearer direction

Can work a lot with REPL

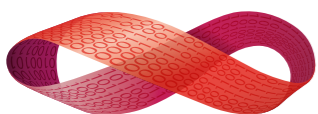
Great workflow, (e.g. Type holes)

Many of the higher abstractions

Cons

Takes time to learn the higher abstractions

Type errors are not as helpful as Elm



Pros compared to Elm

Pursuit (search libs by type signature)

Clearer direction

Can work a lot with REPL

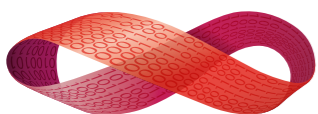
Great workflow, (e.g. Type holes)

Many of the higher abstractions

Cons

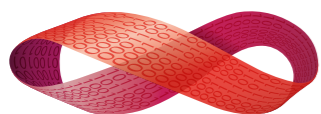
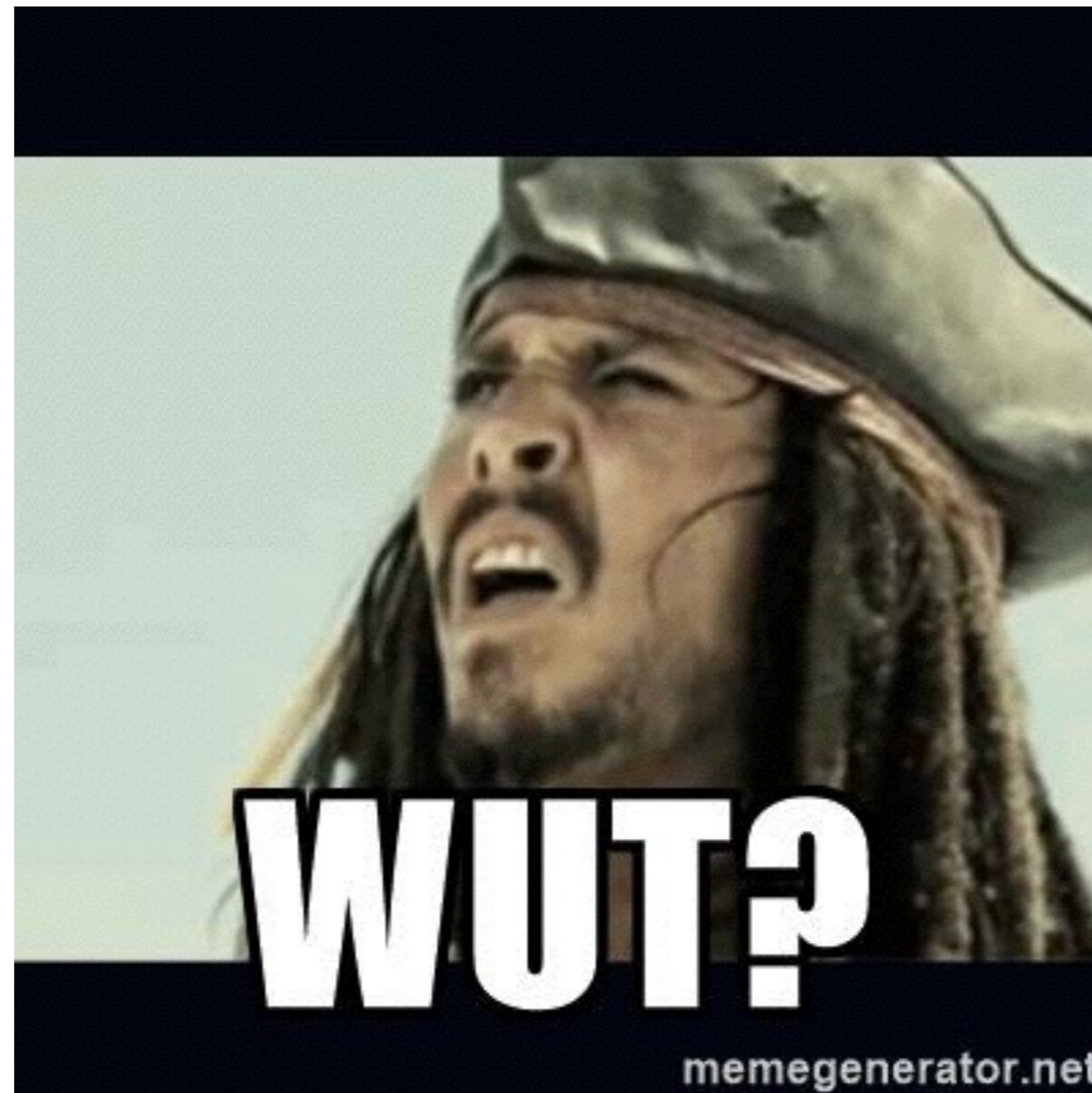
Takes time to learn the higher abstractions

Type errors are not as helpful as Elm



Too hard?

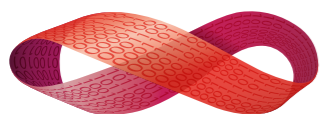
“A monad is just a monoid in the category of endofunctors”



Too hard?

Definitely you don't have to know **everything** to start

Coming from a language like Elm you only need to learn how to **use a few Monads** (use, not write), and get **familiar with Typeclasses** to start getting productive in Purescript

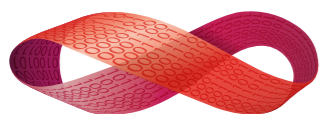


Keeping up with the Haskell type treadmill

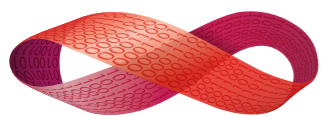
Researchers are inventing and discovering **new ideas all the time**, you'll never learn them all.

Just **go at your own pace**

The higher abstractions will still be there tomorrow

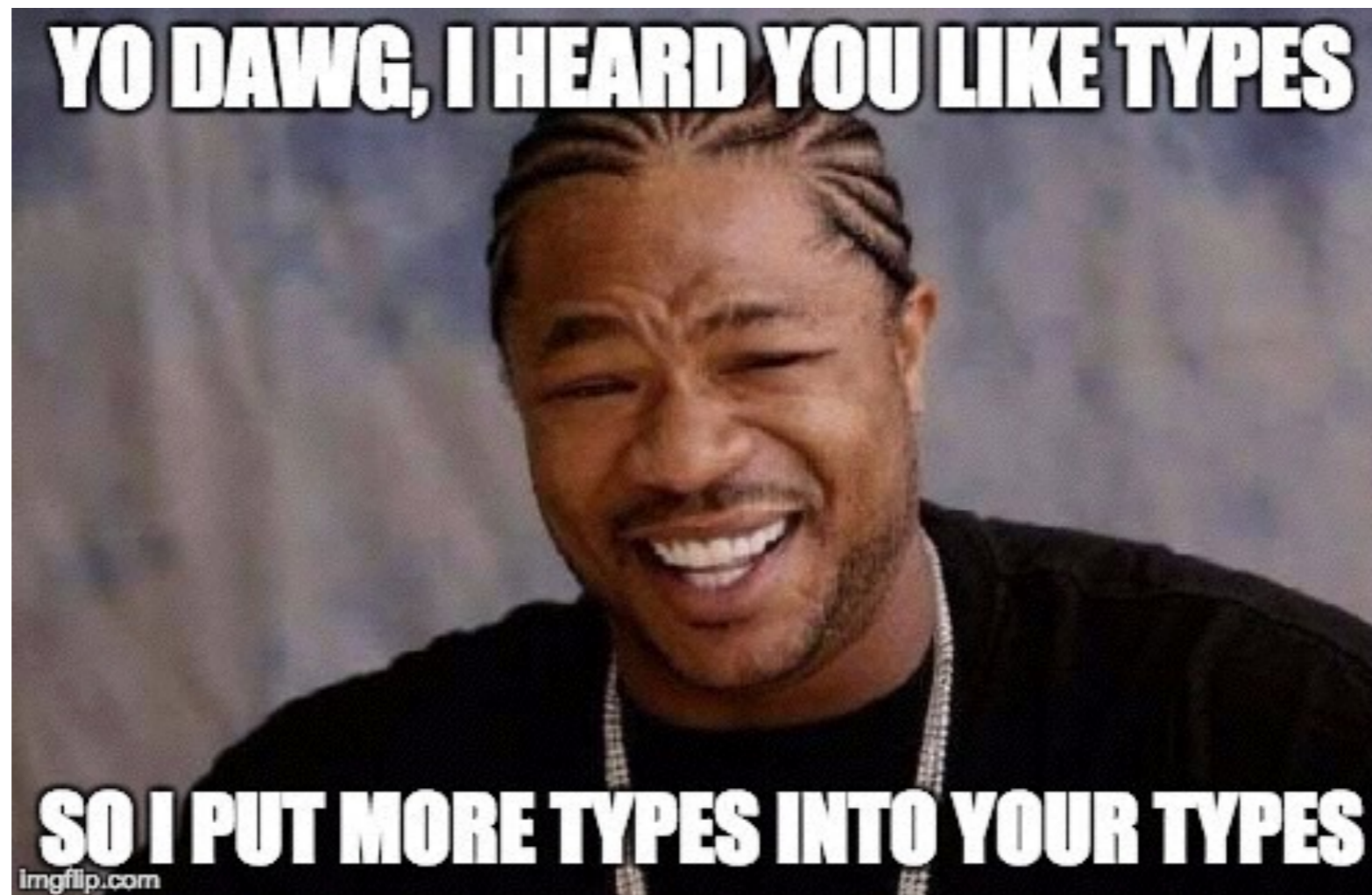


Why Purescript after Elm?

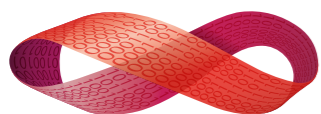


Why Purescript after Elm?

Exhibit 1: the **type system** is a great feature of Elm

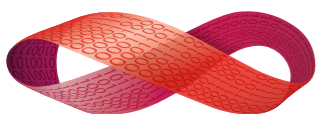


Purescript's has more features. (Simplicity vs Power)



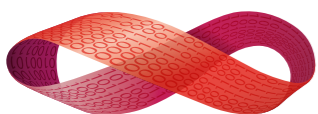
Why Purescript after Elm?

- once you get **restless with Elm's boilerplate**, you're likely ready for more powerful abstractions
- it's similar enough that **porting code is relatively straightforward**
- it's possible to implement Elm in it, but not the other way around (**general purpose**)
- it benefits from the hindsight of **following Haskell from a time distance**



Why Purescript after Elm?

- once you get **restless with Elm's boilerplate**, you're likely ready for more powerful abstractions
- it's similar enough that **porting code is relatively straightforward**
- it's possible to implement Elm in it, but not the other way around (**general purpose**)
- it benefits from the hindsight of **following Haskell from a time distance**



purescript-elm

← → ↻ 🔒 GitHub, Inc. [US] | <https://github.com/rgrempel/purescript-elm> 🔍 ☆ ⌵

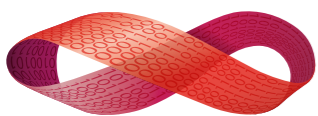
purescript-elm

Having done some [Elm](#) programming, I wanted to give [Purescript](#) a try. I thought I would port one of my Elm apps to Purescript, but quickly realized that there were a variety of little differences between the Elm core libraries and their Purescript equivalents. One possible approach would have been to modify my app. However, it seemed to me that it might be more interesting to port the Elm libraries to Purescript -- at least as a first step. I could then change the app to use more idiomatic Purescript at my leisure.

Having started down that rabbit hole, I became fascinated by how Purescript does things -- and also fascinated by some of the inner workings of Elm. One of the things I've tried to do is rewrite as much as possible of the Javascript used by Elm in plain-old-Purescript. This has been more time-consuming than just wrapping the Javascript, but it has been a nice way to teach myself idiomatic Purescript techniques.

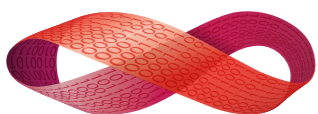
I have now broken out some of the core Elm modules into a [purescript-elm-compatible](#) library, which deals with "basic" core modules, such as, well, `Basics`, and `Array`, `Bitwise`, `Char`, `Date`, `Debug`, `Dict`, `Json.Encode`, `Json.Decode`, `List`, `Maybe`, `Random`, `Regex`, `Result`, `Set`, `String`, and `Trampoline`. So, you might find those useful already.

The main things remaining to do are:



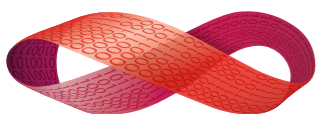
Why Purescript after Elm?

- once you get **restless with Elm's boilerplate**, you're likely ready for more powerful abstractions
- it's similar enough that **porting code is relatively straightforward**
- it's possible to implement Elm in it, but not the other way around (**general purpose**)
- it benefits from the hindsight of **following Haskell from a time distance**



Why Purescript after Elm?

- once you get **restless with Elm's boilerplate**, you're likely ready for more powerful abstractions
 - it's similar enough that **porting code is relatively straightforward**
 - it's possible to implement Elm in it, but not the other way around (**general purpose**)
- it benefits from the hindsight of **following Haskell from a time distance**



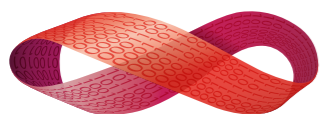
Philosophy Differences



Elm gives you only **one possible program structure** (Elm arch)



In Purescript there are **many possible ways of structuring your app**



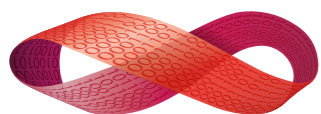
Philosophy Differences



Elm is made to be **simple** above anything else, have a quick learning curve



In Purescript you have most of the type features you have in Haskell, **longer learning curve**



Reflection on Elm - Purescript - Haskell



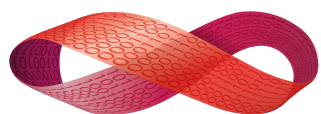
- Simplest
- Focused on UX
- One way to do things
- Removes all historical baggage
- Great entry level language
- only targets web browsers



- Sensible
- UX is fairly good
- Still a lot of power
- Eagerly evaluated, hence simpler
- general purpose
- many backends (C++, Erlang, Js)



- Research language
- Most powerful
- Least good UX
- Most historical baggage
- Laziness adds complexity
- Compiles to native code, Ivm, C, etc



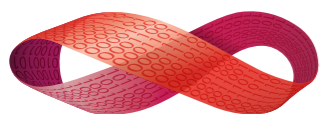
Frameworks Overview

Wrapping React

- Pux
- Thermite
- purescript-react

Pure

- Halogen
- Flare
- Optic UI



Frameworks

Type Complexity continuum

Easy

**Here be
lenses**

**Here be
free monads**

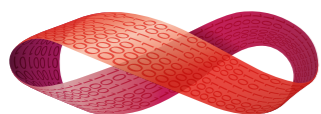


Flare

Pux

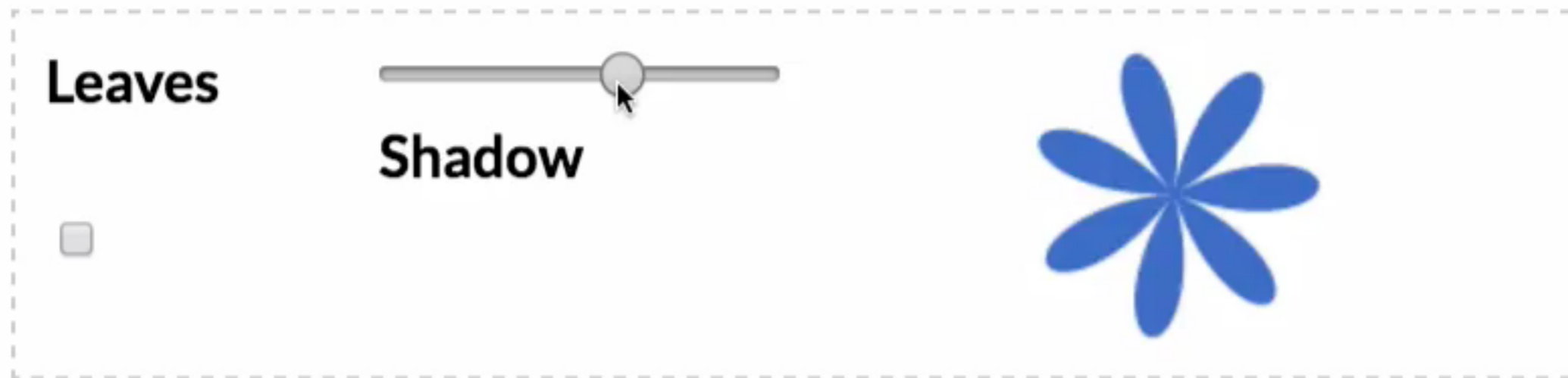
Thermite
Optic UI

Halogen

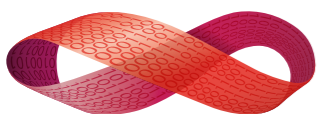


Why Flare?

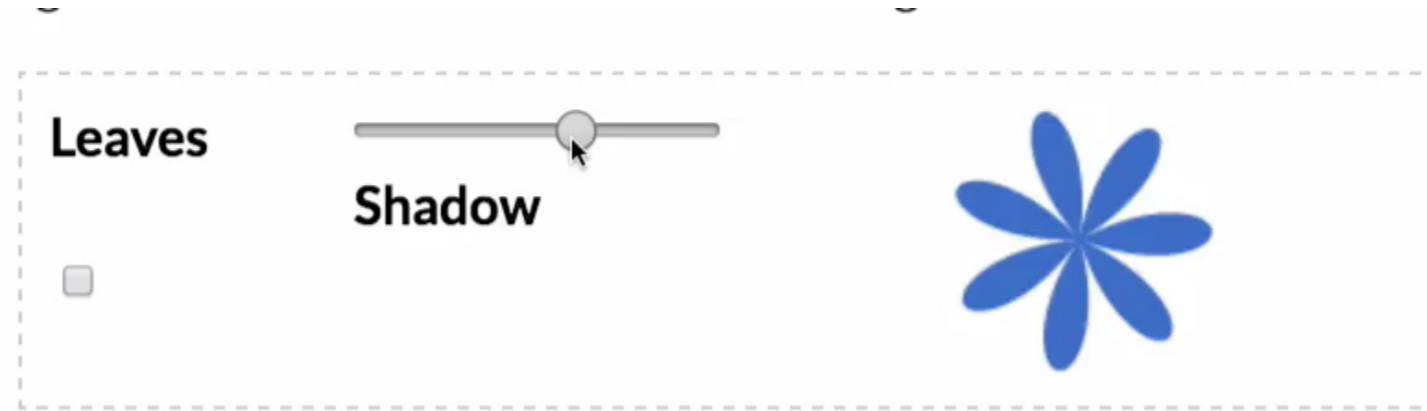
- Great to start with
- Easy to make cool interactive-graphs



- Limited to a specific use case
- Need to understand applicative functor syntax:
thing <\$> thing <*> thing



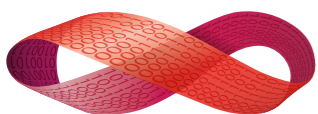
Why Flare?



```
plot :: Int -> Boolean -> Number -> Drawing
plot n s time = shadow (style s) $
    filled (fillColor (hsl 220.0 0.6 0.5)) $
        path (map point angles)

where point phi = { x: 50.0 + radius phi * cos phi
                  , y: 50.0 + radius phi * sin phi }
  angles = map (\i -> 2.0 * pi / toNumber points * toNumber i) (0 .. points)
  points = 200
  radius phi = 48.0 * abs (cos (0.5 * toNumber n * (phi + phi0)))
  phi0 = 0.001 * time
  style false = mempty
  style true = shadowColor black <> shadowOffset 2.0 2.0 <> shadowBlur 2.0

ui4 = lift3 plot (intSlider "Leaves" 2 10 6)
          (boolean "Shadow" false)
          (lift animationFrame)
```



Why Pux?

Very similar to the Elm architecture (0.16)

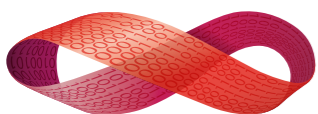
Svg support already included

Interactive React debugger can be wired in

Probably the simplest Purescript framework

Why not?

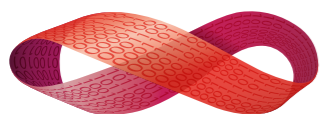
React dependencies /0\



On the pain of installing React

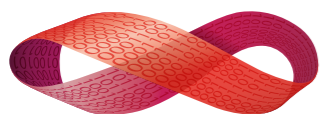


(Though the React interactive debugger is nice)



But!

Now it can use PReact instead of React



Pux Structure

State

Action

update
view

inputs
Aff

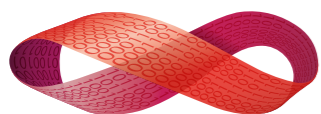
Compare with the Elm Architecture (0.16)

Model

Action

update
view

inputs
Effects



Counter Code

```
data Action = Increment | Decrement
```

Action

State

```
type State = Int
```

```
update :: Action -> State -> State
```

```
update Increment state = state + 1
```

```
update Decrement state = state - 1
```

update

view

```
view :: State -> Html Action
```

```
view state =
```

```
  div []
```

```
    [ button [ onClick (const Increment) ]
```

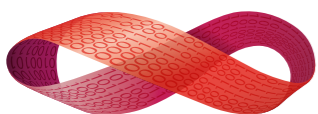
```
            [ text "Increment" ]
```

```
    , span [] [ text (show state) ]
```

```
    , button [ onClick (const Decrement) ]
```

```
            [ text "Decrement" ]
```

```
    ]
```



Thermite

Wraps React

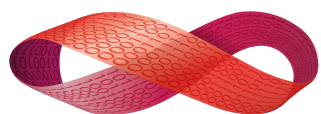
Lenses and stuff

Written by Phil Freeman,
Purescript's author

Optic UI

Pure Purescript

Lenses and stuff



Why Halogen?

Doesn't depend on React

It's used in production by Slamdata, on a pretty impressive app

> 1 people developing it

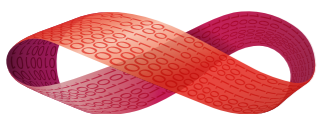
Nice Html DSL

v1.0.0 has arrived!

Why not?

Argh, the types!! My eyes burn!

aka it's just a bit hard



Slamdata

slamdata / slamdata

Watch 34

Unstar 278

Fork 65

Code

Issues 173

Pull requests 9

Projects 0

Wiki

Insights

Contributors

Commits

Code frequency

Punch card

Network

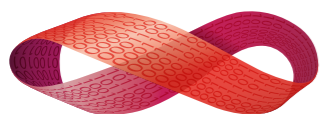
Members

Dependents

Feb 8, 2015 – Jun 6, 2017

Contributions: Commits

Contributions to master, excluding merge commits



Why Halogen?

Doesn't depend on React

It's used in production by Slamdata, on a pretty impressive app

> 1 people developing it

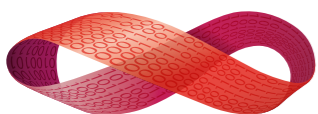
Nice Html DSL

v1.0.0 has arrived!

Why not?

Argh, the types!! My eyes burn!

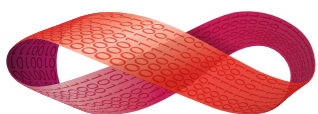
aka it's just a bit hard



Halogen Structure



Compare with Pux



Halogen Structure

State

```
-- | The state of the component
type State = Boolean

-- | The query algebra for the component
data Query a
  = ToggleState a
  | IsOn (Boolean -> a)
```

Query
action
request

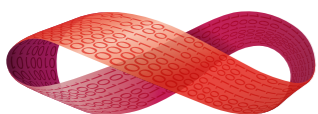
```
data Message = Toggled Boolean
```

```
type Input = Unit
```

Component

```
-- | The component definition
myButton :: forall m. H.Component HH.HTML Query Input Message
myButton =
  H.component
    { initialState: const initialState
    , render
    , eval
    , receiver: const Nothing
    }
```

```
where
```



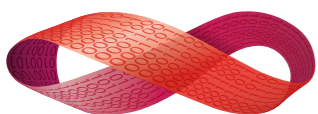
Halogen Structure



render

```
initialState :: State
initialState = false
```

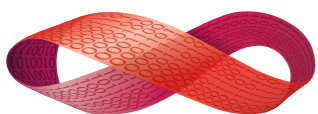
```
render :: State -> H.ComponentHTML Query
render state =
  let
    label = if state then "On" else "Off"
  in
    HH.button
      [ HP.title label
      , HE.onClick (HE.input_ Toggle)
      ]
      [ HH.text label ]
```



Halogen Structure

eval

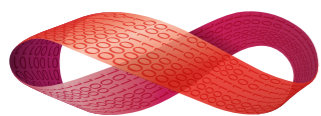
```
eval :: Query ~>
      H.ComponentDSL State Query Message m
eval = case _ of
  Toggle next -> do
    state <- H.get
    let nextState = not state
    H.put nextState
    H.raise $ Toggled nextState
    pure next
  IsOn reply -> do
    state <- H.get
    pure (reply state)
```



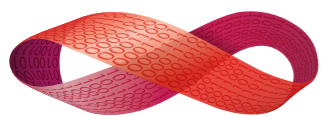
Porting choices

1. Which tools?

2. Which framework?



Which tools?

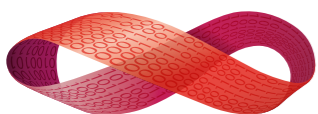


Package Management

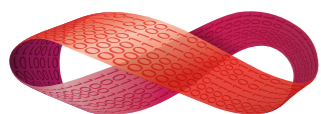
At the moment Purescript is relying on **bower**, which makes the time after a new release particularly annoying

But Purescript's community is working on a new package manager: **psc-package**

And there is also **purify**, which wants to be like Haskell's stack (reproducible builds)



Bower: **The day after a new release**



Bower:

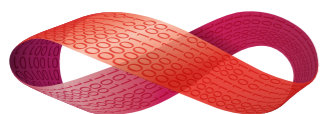
The day after a new release

Bower **has no clue about Purescript**

Bower always gets the **latest version** of a library

Until all libraries are **updated to latest**, chaos

Solution: manually tell bower the version you want

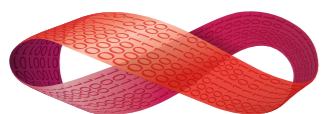


Package Management

At the moment Purescript is relying on **bower**, which makes the time after a new release particularly annoying

But Purescript's community is working on a new package manager: **psc-package**

And there is also **purify**, which wants to be like Haskell's stack (reproducible builds)

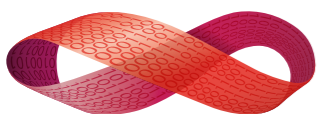


Package Management

At the moment Purescript is relying on **bower**, which makes the time after a new release particularly annoying

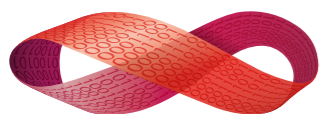
But Purescript's community is working on a new package manager: **psc-package**

And there is also **purify**, which wants to be like Haskell's stack (reproducible builds)



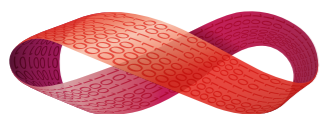
Package Management

Bower



Project Setup

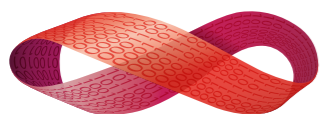
Pulp



Backend Communication

JSON

BERT

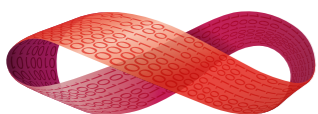


Why BERT?

our server runs on **embedded**

it's better to move as much **computation as possible to the client side**

about **one order of magnitude faster**,
compared to jsx (pure Erlang library)



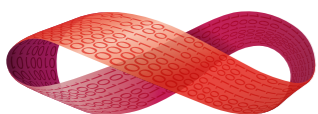
Why BERT?

Time taken

(term to binary) BERT = **1x**

(C NIF) jiffy = **10x**

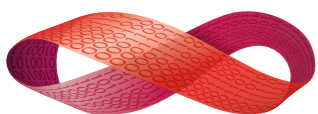
(Erlang) jsx = **43x**



Why NOT BERT?

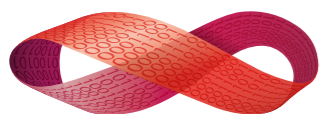
Doesn't support **Erlang maps** yet
(not a problem for us)

less of an ecosystem than JSON



Which Framework?

**we went with
Halogen**

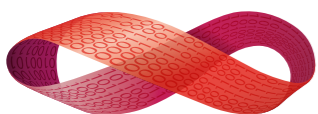


Getting our bearings

Effects are now **within a monad**
instead of the tuple (model, effects)

We can have **components**, and
they can have **state**

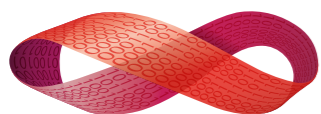
How to **break down into component?**
Matches elm arch (0.16) up to a point



Halogen Structure



Compare with Elm arch

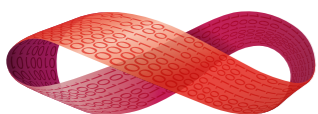


Halogen Structure

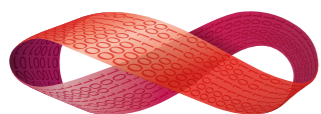
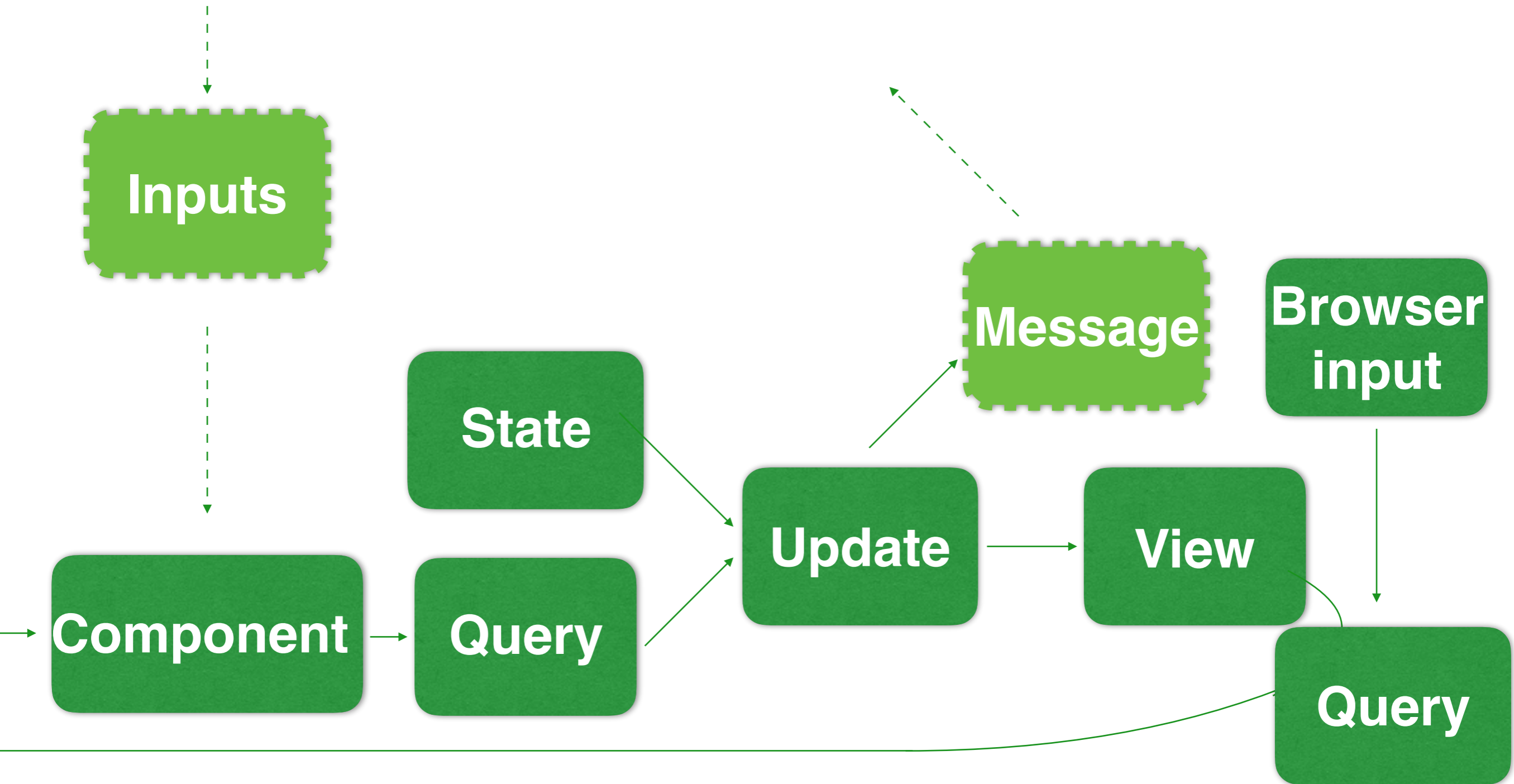
Inputs values: the parent can send information to the children

Messages: components can send information to the parent

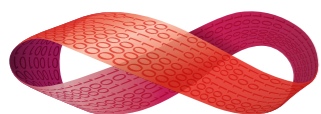
Requests: the parent can request information from the children



Halogen Component



Porting Experience



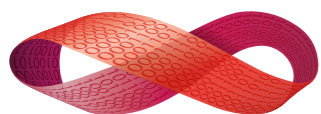
Things that happen while porting from Elm to Purescript

copy pasting will partially work

types will tend naturally to get **more abstracted**

you will spend a fair chunk of time worrying
about **what your monads are doing**

type errors can get cryptic, work in small
chunks, so you can track them down



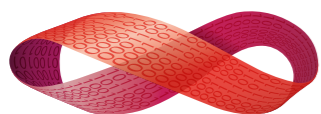
e.g. Html syntax

Elm

```
[ ]
, div [ class "plots"
      / attribute "role" "region"
      / attribute "aria-label" "Data plots"
      / attribute "style" "width: 35%"
      [ ]
, hr [ class "vertical-separator"
     / tabIndex 0
     / attribute "role" "separator"
     / attribute "aria-valuemin" "0"
     / attribute "aria-valuemax" "100"
     / attribute "aria-valuenow" "30"
     [ ]
, div [ class "blocks"
      / attribute "role" "region"
      / attribute "aria-label" "Function blocks"
      / tabIndex 0
      / attribute "style" "width: 65%"
      [ ]
, ... ..
```

Purescript

```
, HH.main_
  [ HH.div
    [ HP.class_ (HH.ClassName "plots")
    , HA.role "region"
    , HA.label "Data plots"
    , HC.style do
      C.width (C.pct 35.0)
    ]
  ]
, HH.hr
  [ HP.class_ (HH.ClassName "vertical-separator")
  , HP.tabIndex 0
  , HA.role "separator"
  , HA.valueMin "0"
  , HA.valueMax "100"
  , HA.valueNow "30"
  ]
, HH.div
  [ HP.class_ (HH.ClassName "blocks")
  , HA.role "region"
  , HA.label "Function blocks"
  , HP.tabIndex 0
  , HC.style do
    C.width (C.pct 65.0)
  ]
]
```



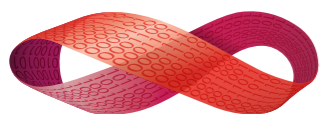
Things that happen while porting from Elm to Purescript

copy pasting will partially work

types will tend naturally to get **more abstracted**

you will spend a fair chunk of time worrying
about **what your monads are doing**

type errors can get cryptic, work in small
chunks, so you can track them down



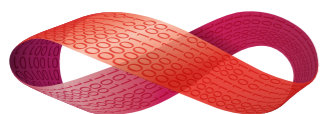
Things that happen while porting from Elm to Purescript

copy pasting will partially work

types will tend naturally to get **more abstracted**

you will spend a fair chunk of time worrying
about **what your monads are doing**

type errors can get cryptic, work in small
chunks, so you can track them down



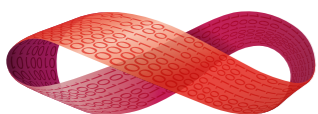
Things that happen while porting from Elm to Purescript

copy pasting will partially work

types will tend naturally to get **more abstracted**

you will spend a fair chunk of time worrying
about **what your monads are doing**

type errors can get cryptic, work in small
chunks, so you can track them down



Purescript Conclusion

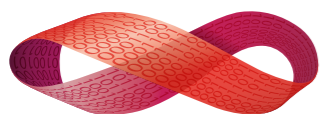
Powerful

No limits on abstractions

It **will take time to learn**, but if you know Elm (or other typed FP) you get a headstart

You **don't** have to know everything to start

It's not obsessed about language UX,
but it's still good

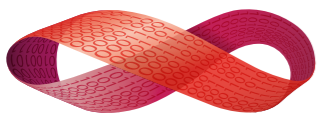


Higher Abstractions in Erlang

Erlando

Erlando is a set of syntax extensions for Erlang. Currently it consists of three syntax extensions, all of which take the form of [parse-transformers](#).

- **Cut:** This adds support for *cuts* to Erlang. These are inspired by the [Scheme form of cuts](#). *Cuts* can be thought of as a light-weight form of abstraction, with similarities to partial application (or currying).
- **Do:** This adds support for *do*-syntax and monads to Erlang. These are heavily inspired by [Haskell](#), and the monads and libraries are near-mechanical translations from the Haskell GHC libraries.
- **Import As:** This adds support for importing remote functions to the current module namespace with explicit control of the local function names.



Higher Abstractions in Elixir

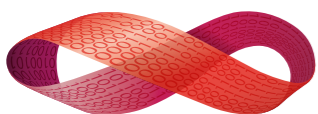
Algae

Bootstrapped
algebraic data types
for Elixir



witchcraft

Witchcraft is a library providing common algebraic and categorical abstractions to Elixir. (Monoids, functors, monads, arrows, and categories)



Steps to get started with Purescript

1. Get Purescript from NPM

Notorious Pug Mafia

npm Enterpri



find packages

purescript public



npm v0.11.5 build passing coverage 100% dependencies up to date

devDependencies up to date

PureScript binary wrapper that makes it seamlessly available via **npm**

Installation

Steps to get started with Purescript

...or **psvm** (version manager)

Neurotic Pink Mongooses

npm Enterprise



find packages

psvm public



Purescript Version Manager

Installation

Steps to get started with Purescript

2. Start reading “Purescript by Example”



Store Read Write Support Blog

Search Leanpub



Sign In

Sign Up

PureScript by Example



PureScript by Example

Buy on Leanpub

Table of Contents

1. Introduction

1.1 Functional JavaScript

1.2 Types and Type Inference

1.3 Polyglot Web Programming

1.4 Prerequisites

1. Introduction

1.1 Functional JavaScript

Functional programming techniques have been making appearances in JavaScript for some time now:

- Libraries such as [UnderscoreJS](#) allow the developer to leverage tried-and-trusted functions such as `map`, `filter` and `reduce` to create larger programs from smaller programs by composition:

Steps to get started with Purescript

3. read purescript-elm-compat

purescript-elm-compat

This package is the first fruits of an effort aimed at people who know [Elm](#) well and wish to give [Purescript](#) a try. The idea is to make it as easy as possible to take Elm code (and Elm knowledge) and use it in Purescript.

The modules in this package are Purescript equivalents of Elm core modules, with `Elm.` tacked on to the beginning. So, Elm's `Maybe` becomes `Elm.Maybe`, Elm's `List` becomes `Elm.List`, etc.

With a few exceptions, the implementation wraps some existing Purescript module, making whatever adjustments are necessary to maintain the Elm API as closely as possible. Thus, this package is unlikely to be of interest to people who do not know Elm -- there is already a more direct way to do everything this package does.

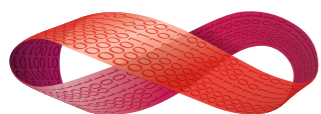
The [larger project](#), still in progress, will also deal with things such as tasks, signals, graphics, HTML, etc. However, I thought that this package might already be of some help to someone.

Compatibility

The modules are based on Elm 0.16, or version 3.0 of the Elm core libraries.

Steps to get started with Purescript

4. Try out Flare



Steps to get started with Purescript

...or Pux

PUX

- Introduction
- Architecture
- Events
- Markup
- Rendering
- Components
- Forms
- Routing
- CSS

- API Reference
- Examples
- Devtool Extension

- GitHub
- Chat (Gitter)
- Learn PureScript

Build type-safe web applications with PureScript

Pux is a PureScript library for building web applications. Interactive UI is modeled as a single state transition function, `Event -> State -> (State, HTML)` which is run for every event. Pux also provides tooling such as:

- ★ Isomorphic routing and rendering
- ★ Hot reloading
- ★ Render to React (or any virtual DOM library)
- ★ Time-travelling debug extension

Quick start

The [starter app](#) provides everything you need to get started:

```
git clone git://github.com/alexmingoia/pux-starter-app.git my-awesome-pux-app
cd my-awesome-pux-app
npm install
npm start
```

Steps to get started with Purescript

5. meet the community

#irc

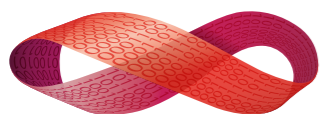
#purescript
@freenode



#purescript
@fpchat



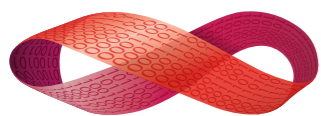
purescript git repo



www.stritzinger.com



@doppioslash



**Win One of 3 Boards by
subscribing to the Newsletter
during the conference
until June 11th**



GRiSP

www.grisp.org

