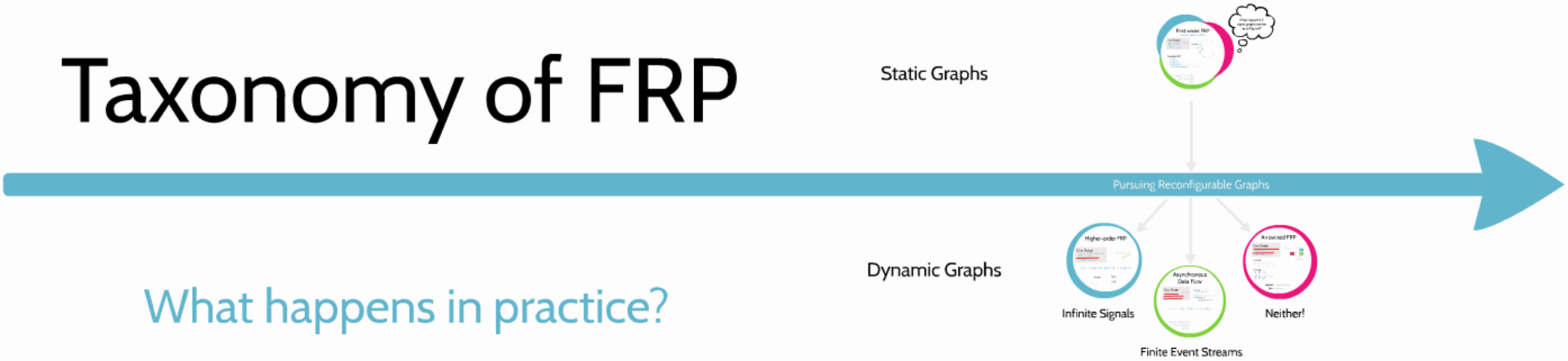
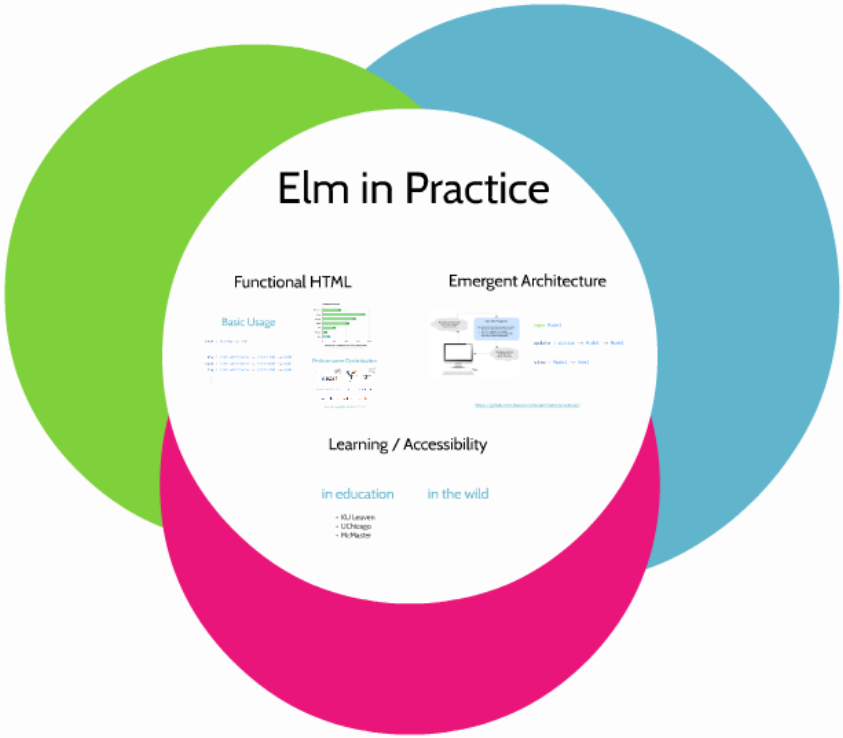
Taxonomy of FRP



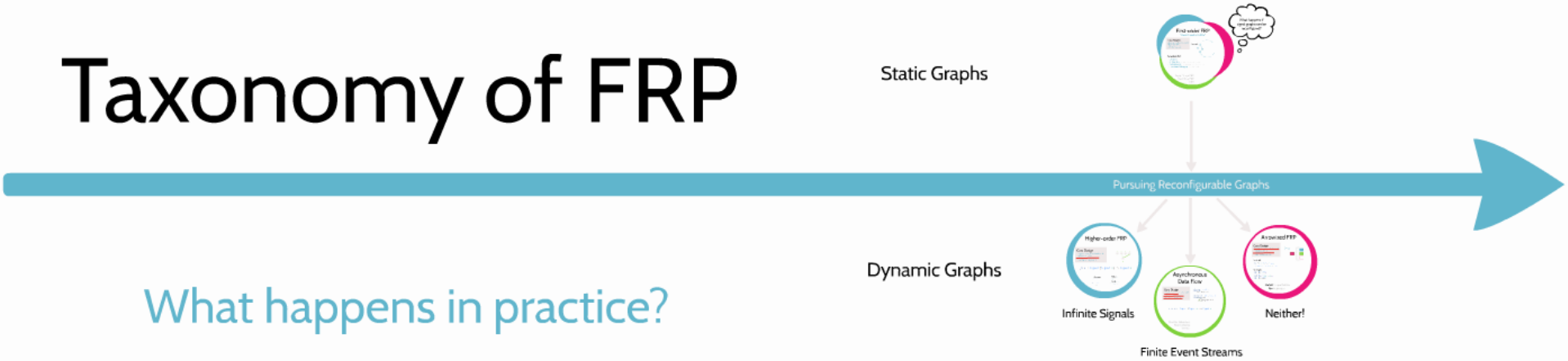
What happens in practice?



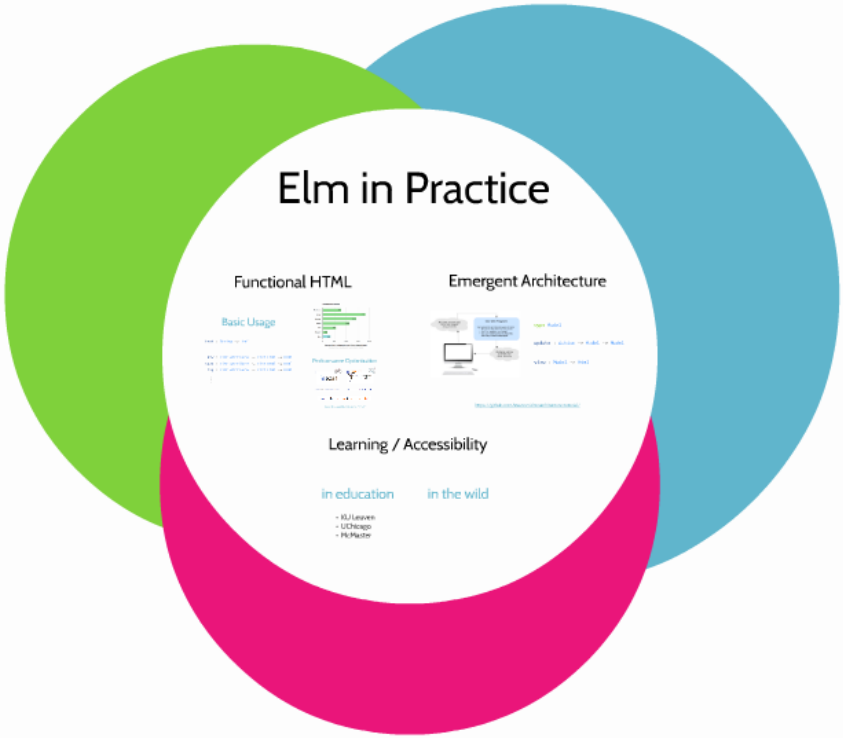
Evan Czaplicki

@czaplic / elm-lang.org / Prezi

Taxonomy of FRP



What happens in practice?



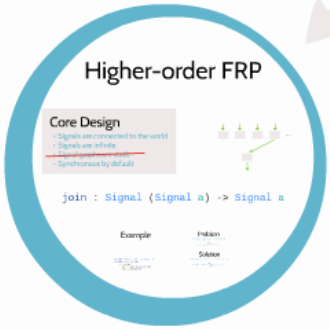
Evan Czaplicki

@czaplic / elm-lang.org / Prezi

Static Graphs

Pursuing Reconfigurable Graphs

Dynamic Graphs



Infinite Signals

Finite Event Streams

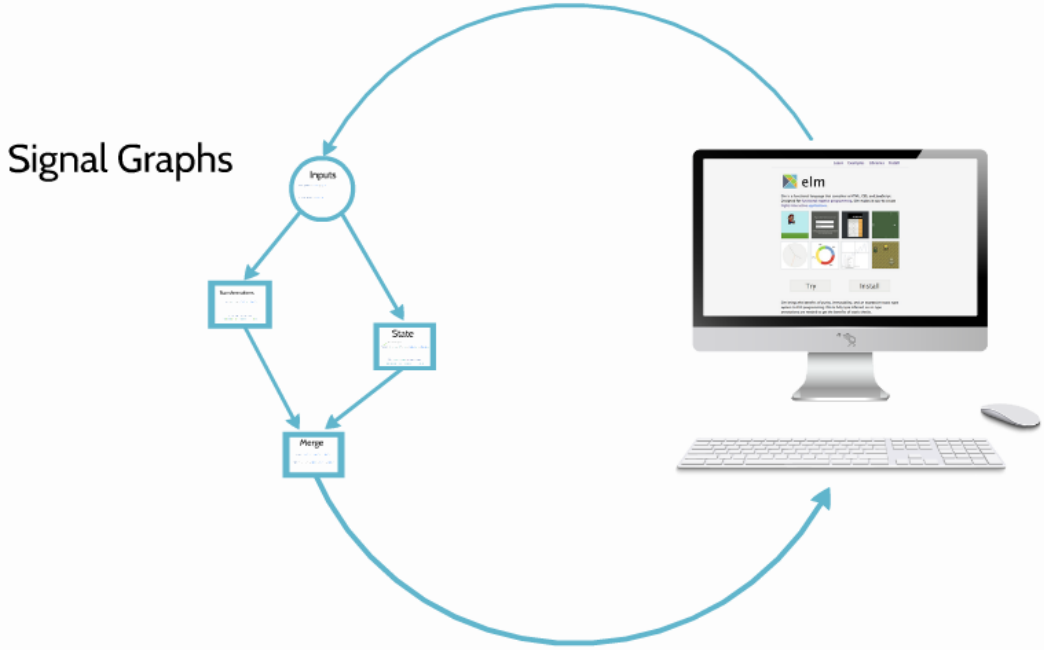
Neither!

First-order FRP

"How it works in Elm"

Core Design

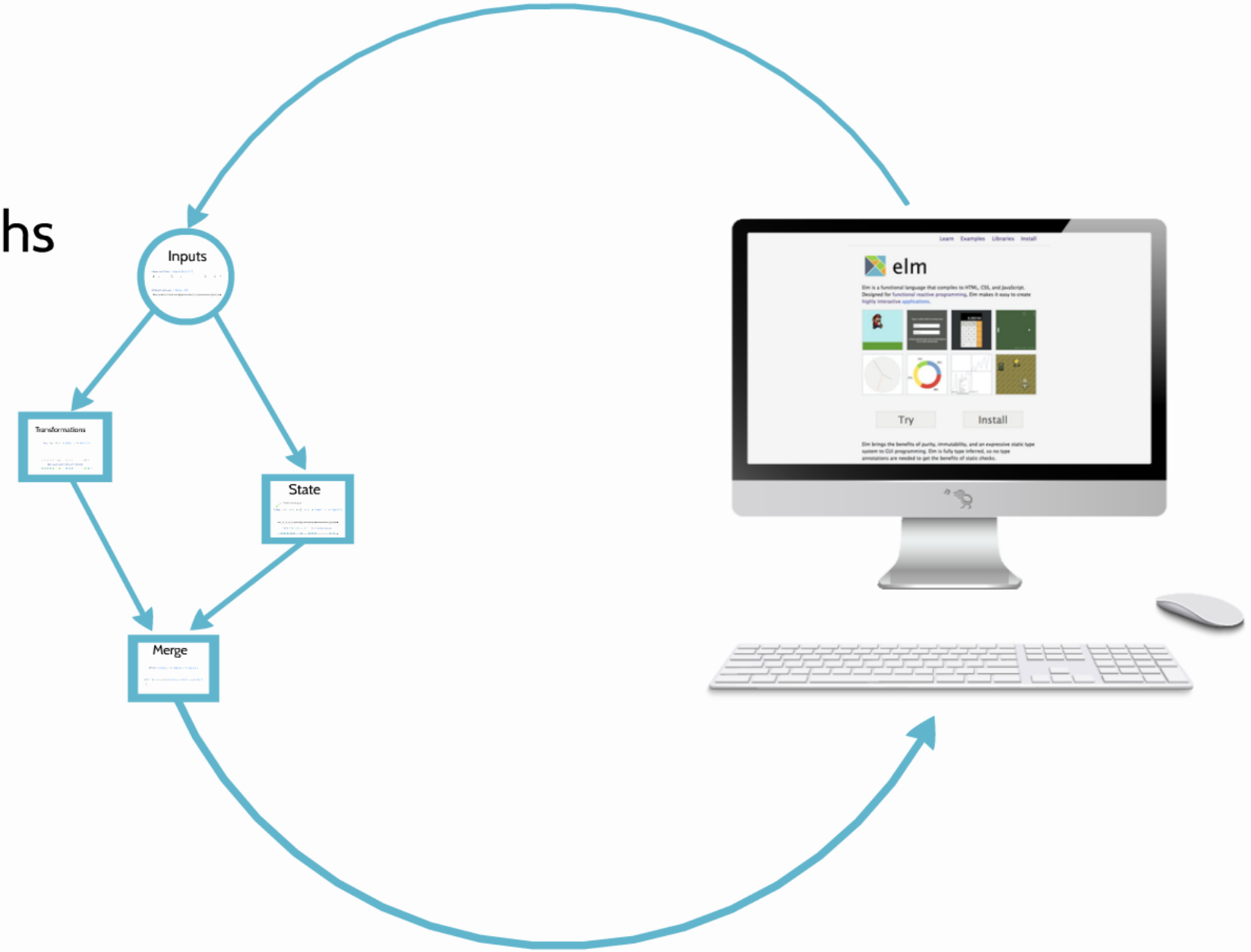
- Signals are connected to the world
- Signals are infinite
- Signal graphs are static
- Synchronous by default



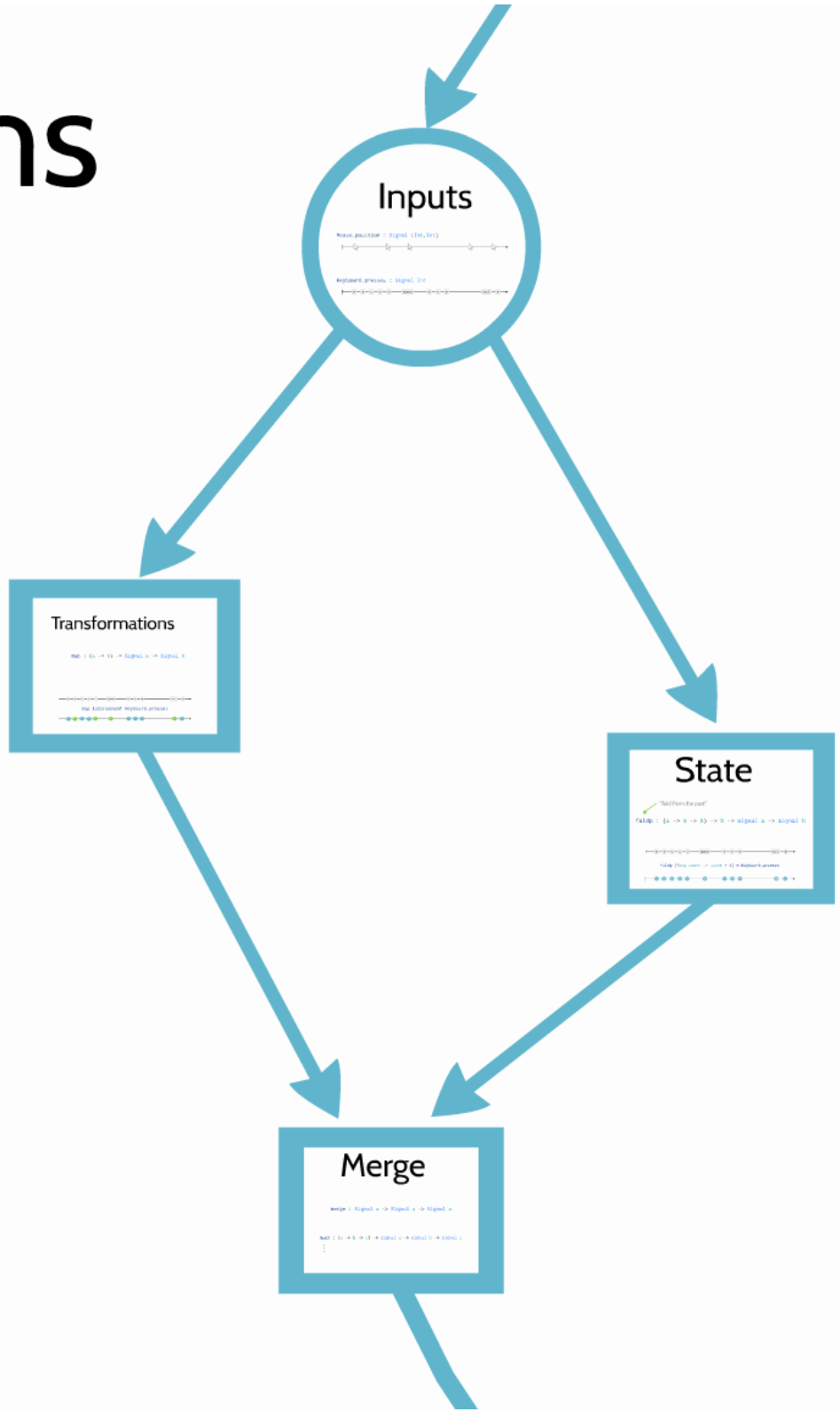
Benefits in Elm

- Efficiency
- Architecture https://github.com/evancz/elm-architecture-tutorial/
- Hot-swapping http://elm-lang.org/edit/examples/Intermediate/Mario.elm
- Time-travel debugging http://debug.elm-lang.org/

Signal Graphs

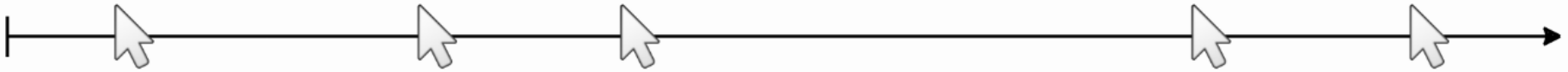


Signal Graphs

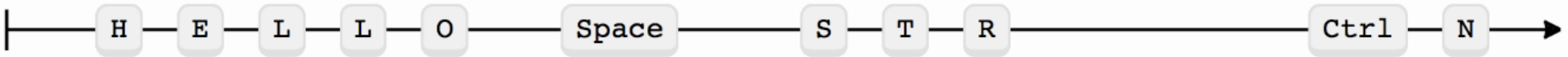


Inputs

```
Mouse.position : Signal (Int,Int)
```

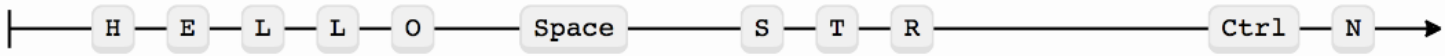


```
Keyboard.presses : Signal Int
```



Transformations

```
map : (a -> b) -> Signal a -> Signal b
```




```
map isConsonant Keyboard.presses
```

State

"fold from the past"

```
foldp : (a -> b -> b) -> b -> Signal a -> Signal b
```


```
foldp (\key count -> count + 1) 0 Keyboard.presses
```

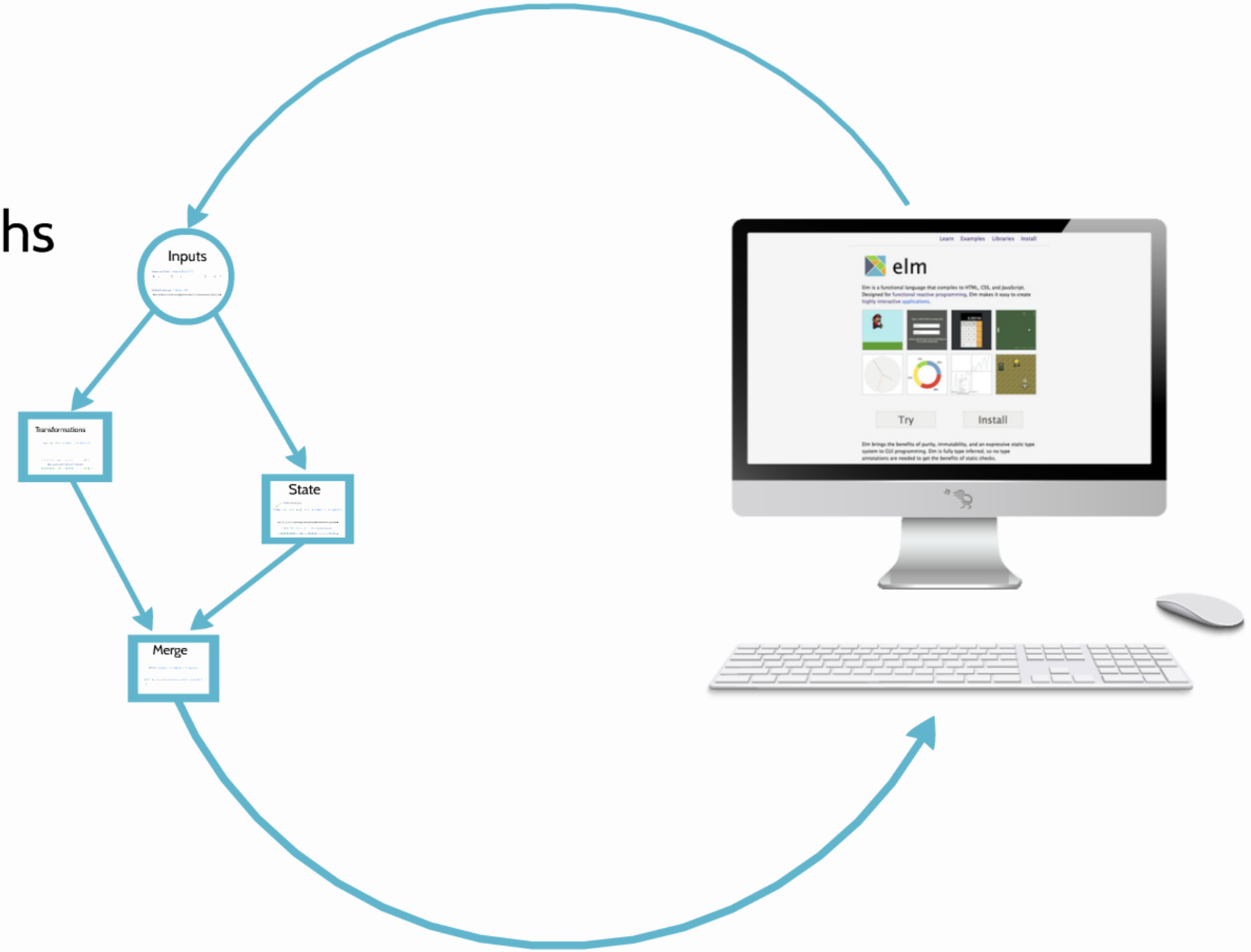
Merge

```
merge : Signal a -> Signal a -> Signal a
```

```
map2 : (a -> b -> c) -> Signal a -> Signal b -> Signal c
```

:
:
:

Signal Graphs

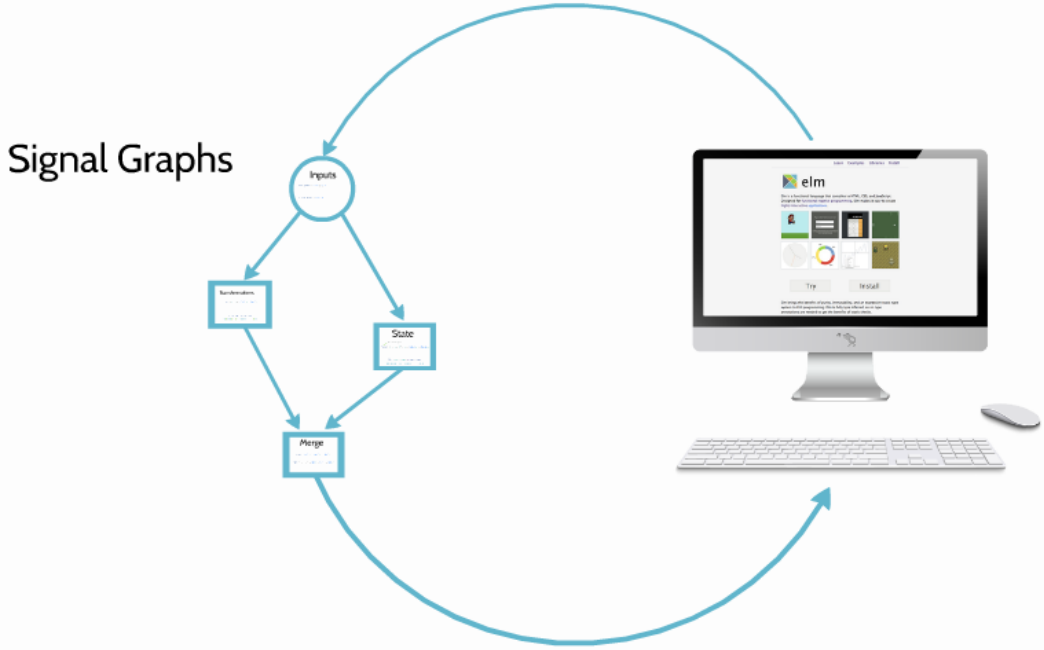


First-order FRP

"How it works in Elm"

Core Design

- Signals are connected to the world
- Signals are infinite
- Signal graphs are static
- Synchronous by default



Benefits in Elm

- Efficiency
- Architecture https://github.com/evancz/elm-architecture-tutorial/
- Hot-swapping http://elm-lang.org/edit/examples/Intermediate/Mario.elm
- Time-travel debugging http://debug.elm-lang.org/

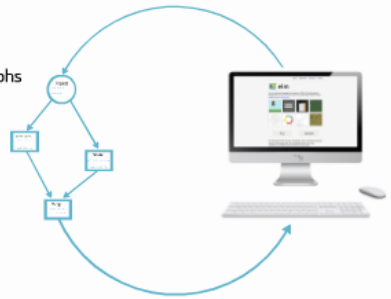
First-order FRP

"How it works in Elm"

Core Design

- Signals are connected to the world
- Signals are infinite
- Signal graphs are static
- Synchronous by default

Signal Graphs



Benefits in Elm

- Efficiency
- Architecture https://github.com/evancz/elm-architecture-tutorial/
- Hot-swapping http://elm-lang.org/edit/examples/Intermediate/Mario.elm
- Time-travel debugging http://debug.elm-lang.org/

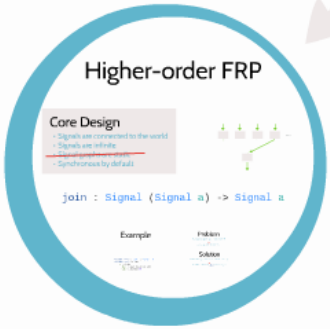
Event-Driven FRP
Real-Time FRP
Elm

What happens if signal graphs can be reconfigured?

Static Graphs

Pursuing Reconfigurable Graphs

Dynamic Graphs

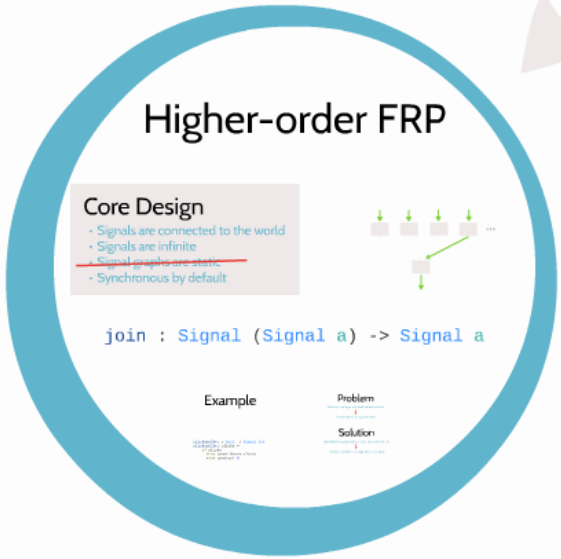


Infinite Signals

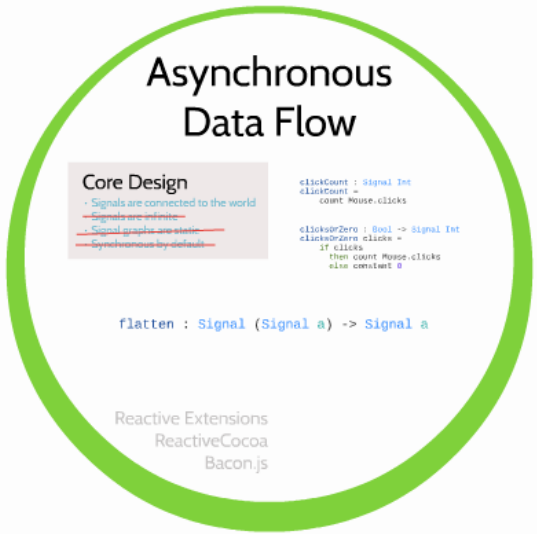
Finite Event Streams

Neither!

Pursuing Reconfigurable Graphs



Infinite Signals



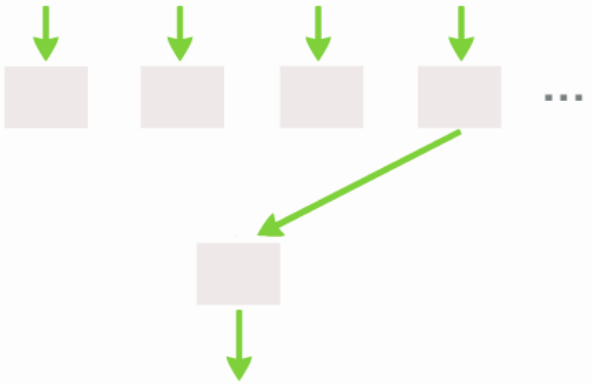
Neither!

Finite Event Streams

Higher-order FRP

Core Design

- Signals are connected to the world
- Signals are infinite
- ~~Signal graphs are static~~
- Synchronous by default



```
join : Signal (Signal a) -> Signal a
```

Example

```
clicksOrZero : Bool -> Signal Int
clicksOrZero clicks =
    if clicks
      then count Mouse.clicks
      else constant 0
```

Problem

Creating a new signal may need infinite look back!

Memory growth is linear with time!

Solution

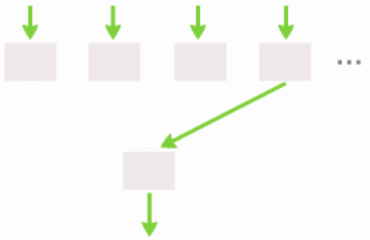
Only switch to signals that have "safe" amounts of history.

Restrict the definition of `join` with fancier types.

Higher-order FRP

Core Design

- Signals are connected to the world
- Signals are infinite
- ~~Signal graphs are static~~
- Synchronous by default



```
join : Signal (Signal a) -> Signal a
```

Example

```
clicksOrZero : Bool -> Signal Int
clicksOrZero clicks =
    if clicks
      then count Mouse.clicks
      else constant 0
```

Problem

Creating a new signal may need infinite look back!

↓

Memory growth is linear with time!

Solution

Only switch to signals that have "safe" amounts of history

↓

Restrict the definition of `join` with fancier types

Asynchronous Data Flow

Core Design

- Signals are connected to the world
- ~~Signals are infinite~~
- ~~Signal graphs are static~~
- ~~Synchronous by default~~

```
clickCount : Signal Int
clickCount =
    count Mouse.clicks


clicksOrZero : Bool -> Signal Int
clicksOrZero clicks =
    if clicks
      then count Mouse.clicks
      else constant 0
```

```
flatten : Signal (Signal a) -> Signal a
```

Reactive Extensions
ReactiveCocoa
Bacon.js

Asynchronous Data Flow

Core Design

- Signals are connected to the world
- ~~Signals are infinite~~
- ~~Signal graphs are static~~
- ~~Synchronous by default~~

```
clickCount : Signal Int
clickCount =
    count Mouse.clicks
```

```
clicksOrZero : Bool -> Signal Int
clicksOrZero clicks =
    if clicks
      then count Mouse.clicks
      else constant 0
```

```
flatten : Signal (Signal a) -> Signal a
```

```
clickCount : Signal Int
clickCount =
    count Mouse.clicks


clicksOrZero : Bool -> Signal Int
clicksOrZero clicks =
    if clicks
      then count Mouse.clicks
      else constant 0
```

Asynchronous Data Flow

Core Design

- Signals are connected to the world
- ~~Signals are infinite~~
- ~~Signal graphs are static~~
- ~~Synchronous by default~~

```
clickCount : Signal Int
clickCount =
    count Mouse.clicks


clicksOrZero : Bool -> Signal Int
clicksOrZero clicks =
    if clicks
      then count Mouse.clicks
      else constant 0
```

```
flatten : Signal (Signal a) -> Signal a
```

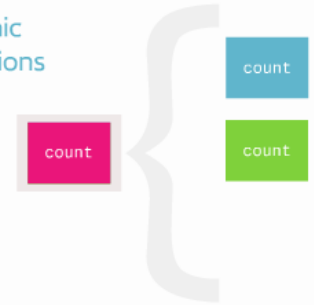
Reactive Extensions
ReactiveCocoa
Bacon.js

Arrowized FRP

Core Design

- ~~Signals are connected to the world~~
- Signals are infinite
- ~~Signal graphs are static~~
- Synchronous by default

Dynamic Collections



Basic API

```
pure : (a -> b) -> Automaton a b

state : s -> (a -> s -> s) -> Automaton a s

(>>>) : Automaton a b -> Automaton b c -> Automaton a c
```

Examples

```
plus1 : Automaton Int Int
plus1 = pure (\n -> n + 1)

count : Automaton a Int
count = state 0 (\a total -> total + 1)

plus2 : Automaton Int Int
plus2 = plus1 >>> plus1
```

Haskell: Yampa, Netwire

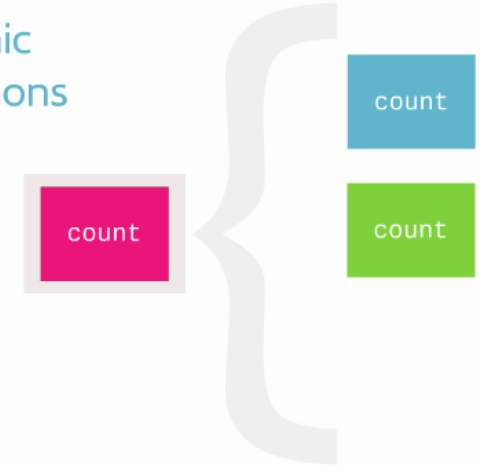
Elm: Automaton

Arrowized FRP

Core Design

- ~~Signals are connected to the world~~
- Signals are infinite
- ~~Signal graphs are static~~
- Synchronous by default

Dynamic Collections



Basic API

```
pure : (a -> b) -> Automaton a b

state : s -> (a -> s -> s) -> Automaton a s

(>>>) : Automaton a b -> Automaton b c -> Automaton a c
```

Examples

```
plus1 : Automaton Int Int
plus1 = pure (\n -> n + 1)

count : Automaton a Int
count = state 0 (\a total -> total + 1)

plus2 : Automaton Int Int
plus2 = plus1 >>> plus1
```

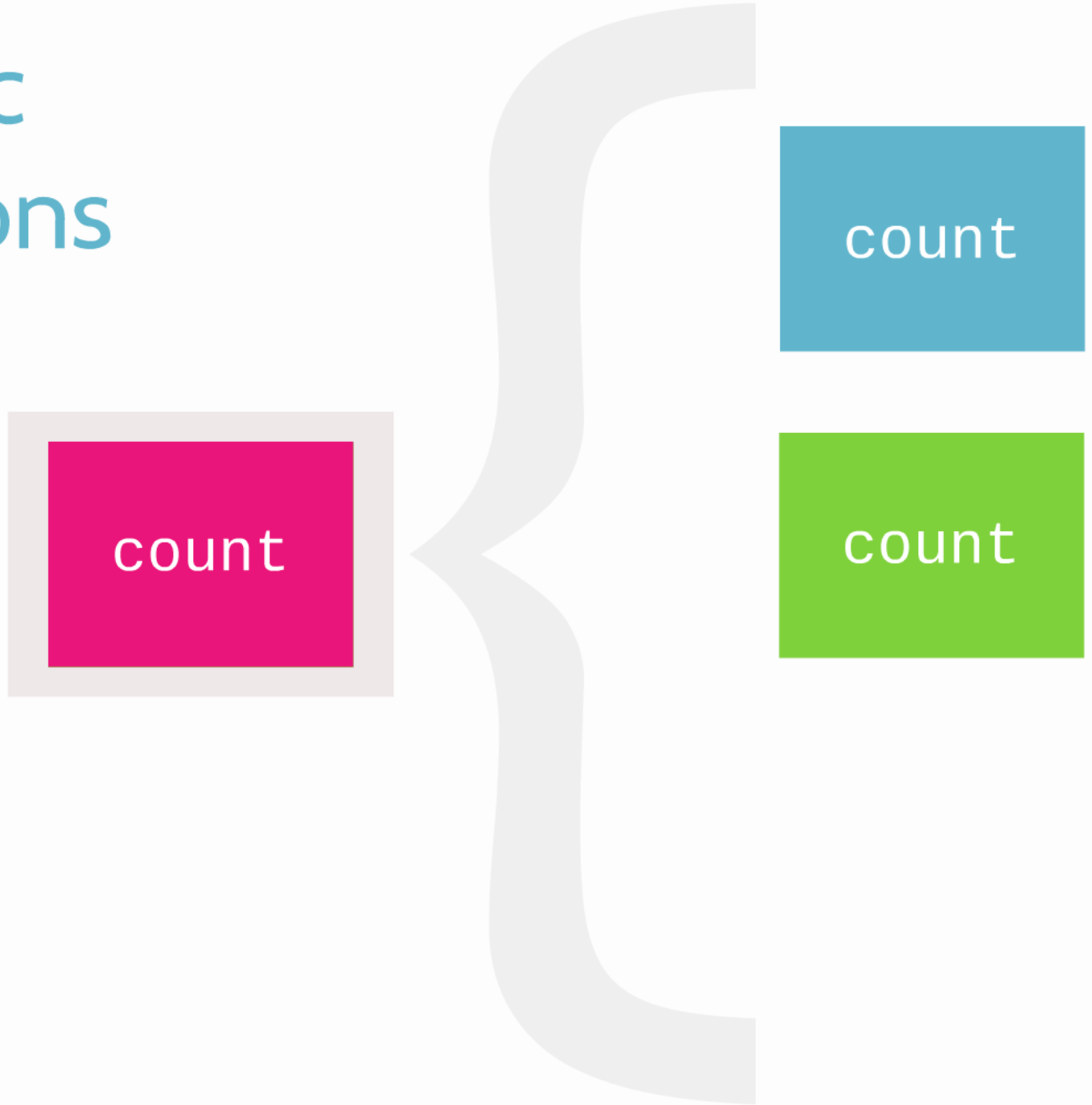
Basic API

```
pure : (a -> b) -> Automaton a b

state : s -> (a -> s -> s) -> Automaton a s

(>>>) : Automaton a b -> Automaton b c -> Automaton a c
```

Examples

```
plus1 : Automaton Int Int
plus1 = pure (\n -> n + 1)

count : Automaton a Int
count = state 0 (\a total -> total + 1)

plus2 : Automaton Int Int
plus2 = plus1 >>> plus1
```

Dynamic Collections

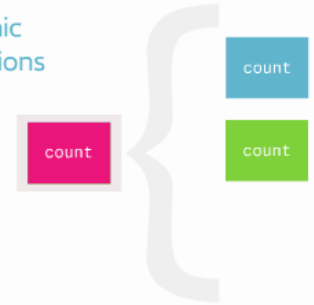


Arrowized FRP

Core Design

- ~~Signals are connected to the world~~
- Signals are infinite
- ~~Signal graphs are static~~
- Synchronous by default

Dynamic Collections



Basic API

```
pure : (a -> b) -> Automaton a b

state : s -> (a -> s -> s) -> Automaton a s

(>>>) : Automaton a b -> Automaton b c -> Automaton a c
```

Examples

```
plus1 : Automaton Int Int
plus1 = pure (\n -> n + 1)

count : Automaton a Int
count = state 0 (\a total -> total + 1)

plus2 : Automaton Int Int
plus2 = plus1 >>> plus1
```

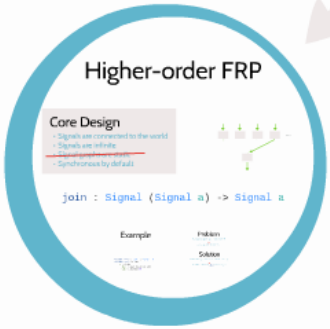
Haskell: Yampa, Netwire

Elm: Automaton

Static Graphs

Pursuing Reconfigurable Graphs

Dynamic Graphs

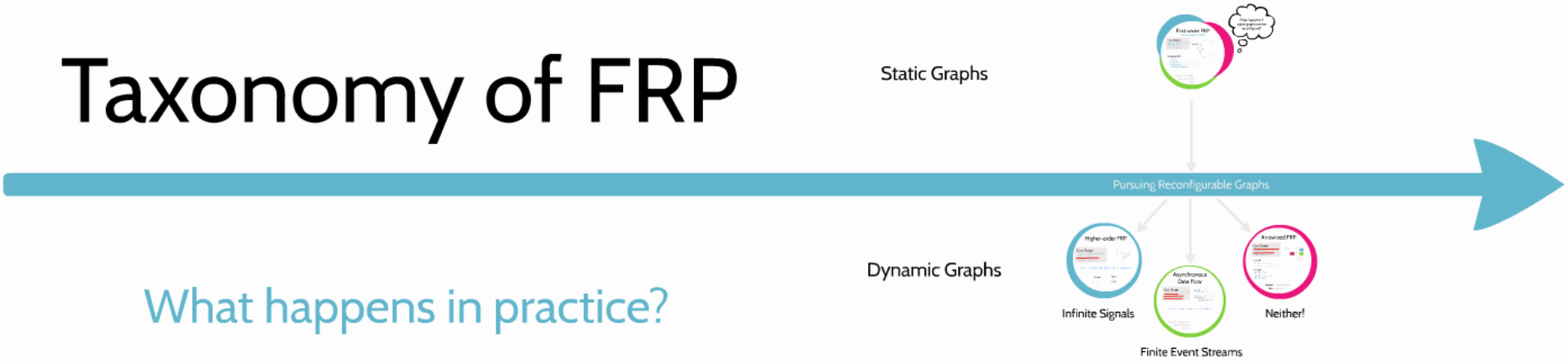


Infinite Signals

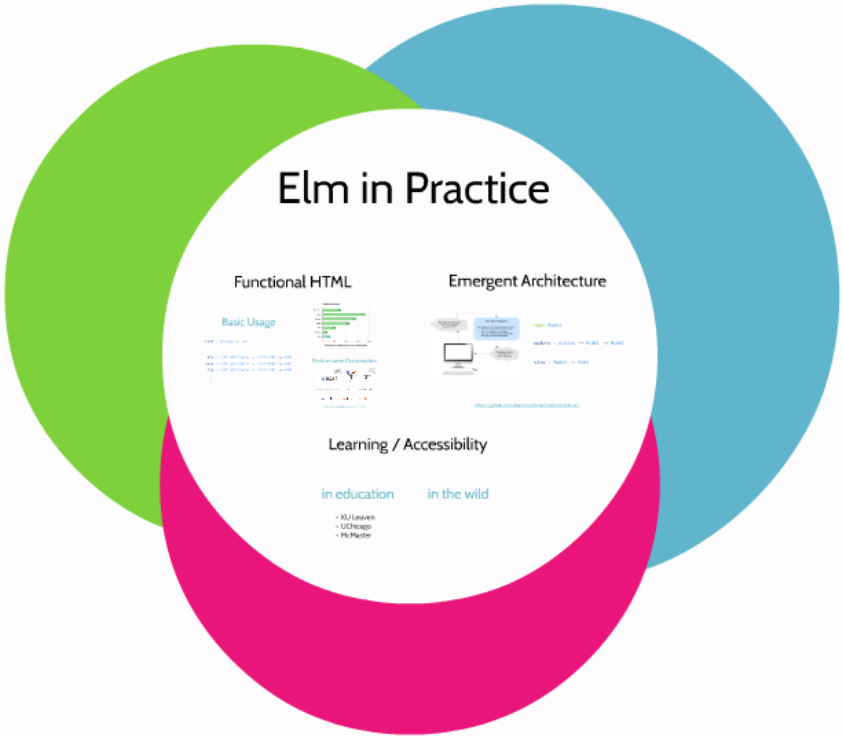
Finite Event Streams

Neither!

Taxonomy of FRP



What happens in practice?



Evan Czaplicki

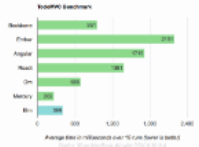
@czaplic / elm-lang.org / Prezi

Elm in Practice

Functional HTML

Basic Usage

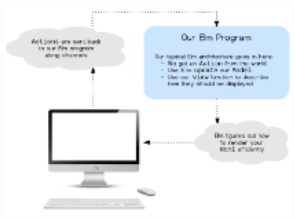
```
text : String -> Html

div : List Attribute -> List Html -> Html
span : List Attribute -> List Html -> Html
img : List Attribute -> List Html -> Html
 :
 :
```



Performance Optimization

```
lazy : (a -> Html) -> a -> Html
```

Emergent Architecture




```
type Model

update : Action -> Model -> Model

view : Model -> Html
```

https://github.com/evancz/elm-architecture-tutorial/

Learning / Accessibility

in education

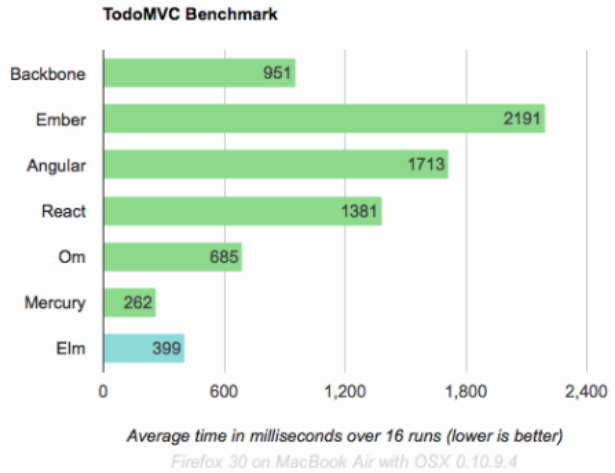
- KU Leuven
- UChicago
- McMaster

in the wild

Functional HTML

Basic Usage

```
text : String -> Html

 div : List Attribute -> List Html -> Html
span : List Attribute -> List Html -> Html
 img : List Attribute -> List Html -> Html
  :
  :
  :
```



Performance Optimization

```
view : Twitter -> Html
view twitter =
  div []
    [ TopBar.view twitter.top
    , SideBar.view twitter.side
    , Tweets.view twitter.feed
    , ...
    ]
```

```
view : Twitter -> Html
view twitter =
  div []
    [ lazy TopBar.view twitter.top
    , lazy SideBar.view twitter.side
    , lazy Tweets.view twitter.feed
    , ...
    ]
```

```
SideBar.view : SideBar.Model -> Html
```

```
TopBar.view : TopBar.Model -> Html
```

```
lazy : (a -> Html) -> a -> Html
```

http://elm-lang.org/blog/Blazing-Fast-Html.elm

Basic Usage

```
text : String -> Html


 div : List Attribute -> List Html -> Html
span : List Attribute -> List Html -> Html
 img : List Attribute -> List Html -> Html

  :
  :
```

Performance Optimization

```
view : Twitter -> Html
view twitter =
  div []
    [ TopBar.view twitter.top
    , SideBar.view twitter.side
    , Tweets.view twitter.feed
    , ...
    ]
```

```
view : Twitter -> Html
view twitter =
  div []
    [ lazy TopBar.view twitter.top
    , lazy SideBar.view twitter.side
    , lazy Tweets.view twitter.feed
    , ...
    ]
```

```
SideBar.view : SideBar.Model -> Html
```

```
TopBar.view : TopBar.Model -> Html
```

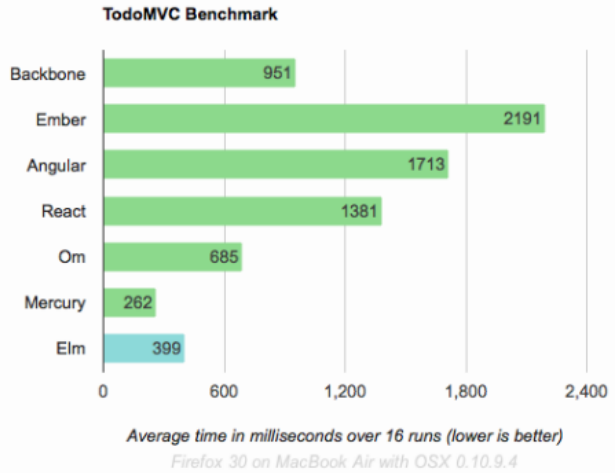
```
lazy : (a -> Html) -> a -> Html
```

http://elm-lang.org/blog/Blazing-Fast-Html.elm

Functional HTML

Basic Usage

```
text : String -> Html

 div : List Attribute -> List Html -> Html
span : List Attribute -> List Html -> Html
 img : List Attribute -> List Html -> Html
  :
  :
  :
```



Performance Optimization

```
view : Twitter -> Html
view twitter =
  div []
    [ TopBar.view twitter.top
    , SideBar.view twitter.side
    , Tweets.view twitter.feed
    , ...
    ]
```

```
view : Twitter -> Html
view twitter =
  div []
    [ lazy TopBar.view twitter.top
    , lazy SideBar.view twitter.side
    , lazy Tweets.view twitter.feed
    , ...
    ]
```

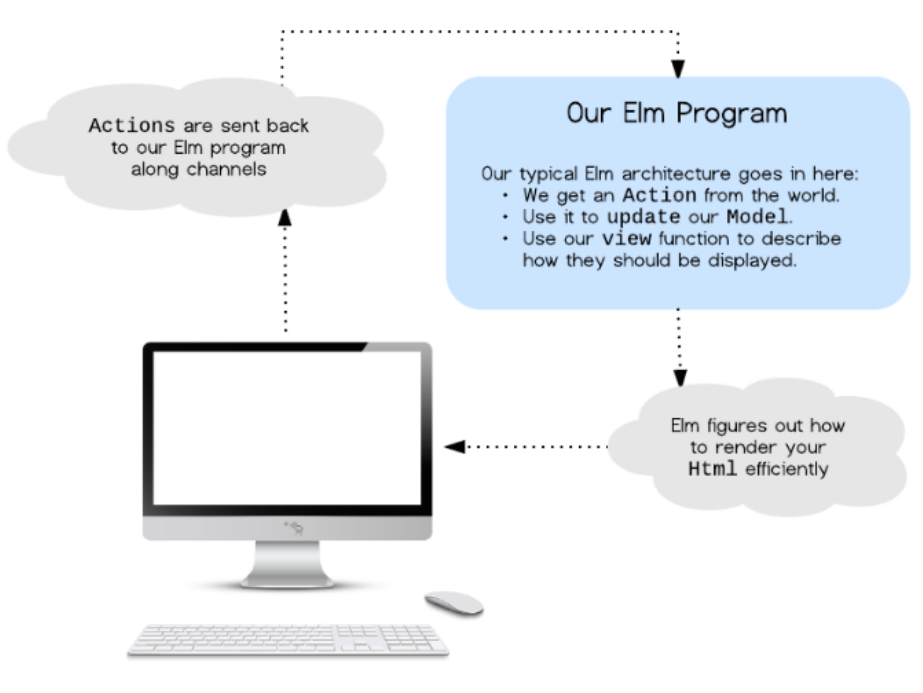
```
SideBar.view : SideBar.Model -> Html
```

```
TopBar.view : TopBar.Model -> Html
```

```
lazy : (a -> Html) -> a -> Html
```

http://elm-lang.org/blog/Blazing-Fast-Html.elm

Emergent Architecture




```
type Model

update : Action -> Model -> Model

view : Model -> Html
```

https://github.com/evancz/elm-architecture-tutorial/

Learning / Accessibility

in education

- KU Leuven
- UChicago
- McMaster

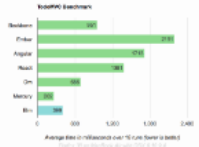
in the wild

Elm in Practice

Functional HTML

Basic Usage

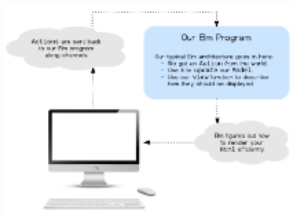
```
text : String -> Html

div : List Attribute -> List Html -> Html
span : List Attribute -> List Html -> Html
img : List Attribute -> List Html -> Html
 :
 :
```



Performance Optimization

```
lazy : (a -> Html) -> a -> Html
```

Emergent Architecture




```
type Model

update : Action -> Model -> Model

view : Model -> Html
```

https://github.com/evancz/elm-architecture-tutorial/

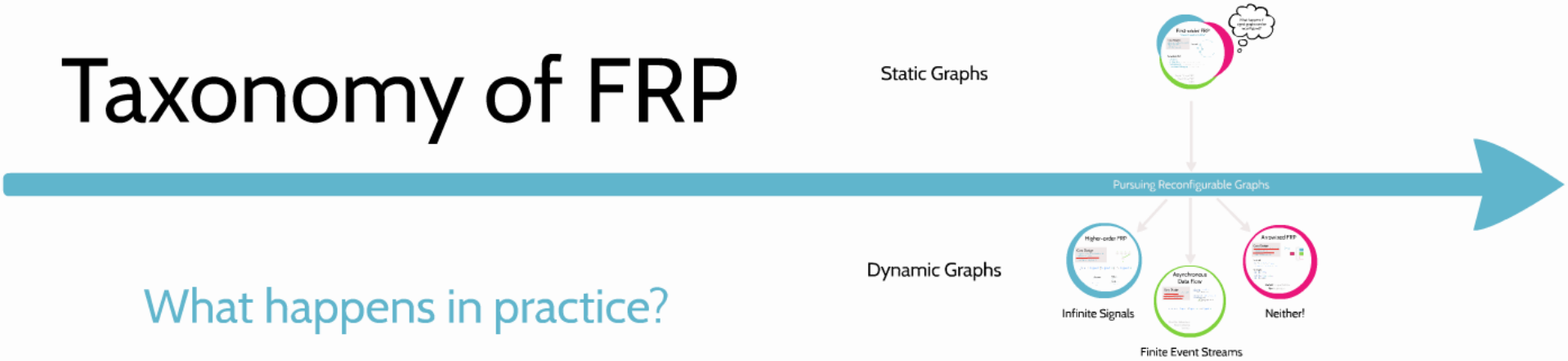
Learning / Accessibility

in education

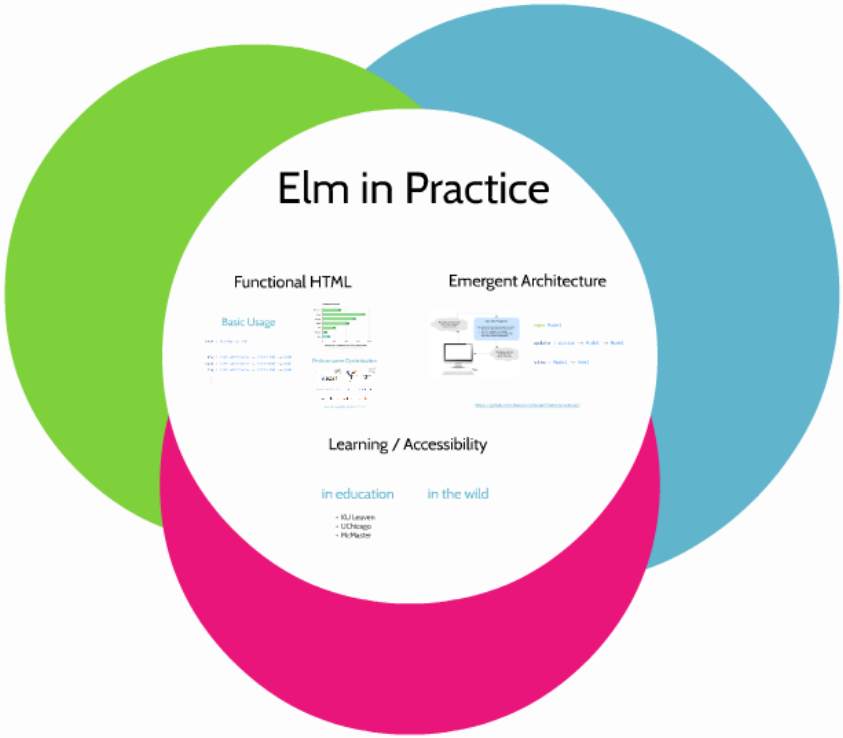
- KU Leuven
- UChicago
- McMaster

in the wild

Taxonomy of FRP



What happens in practice?



Evan Czaplicki

@czaplic / elm-lang.org / Prezi