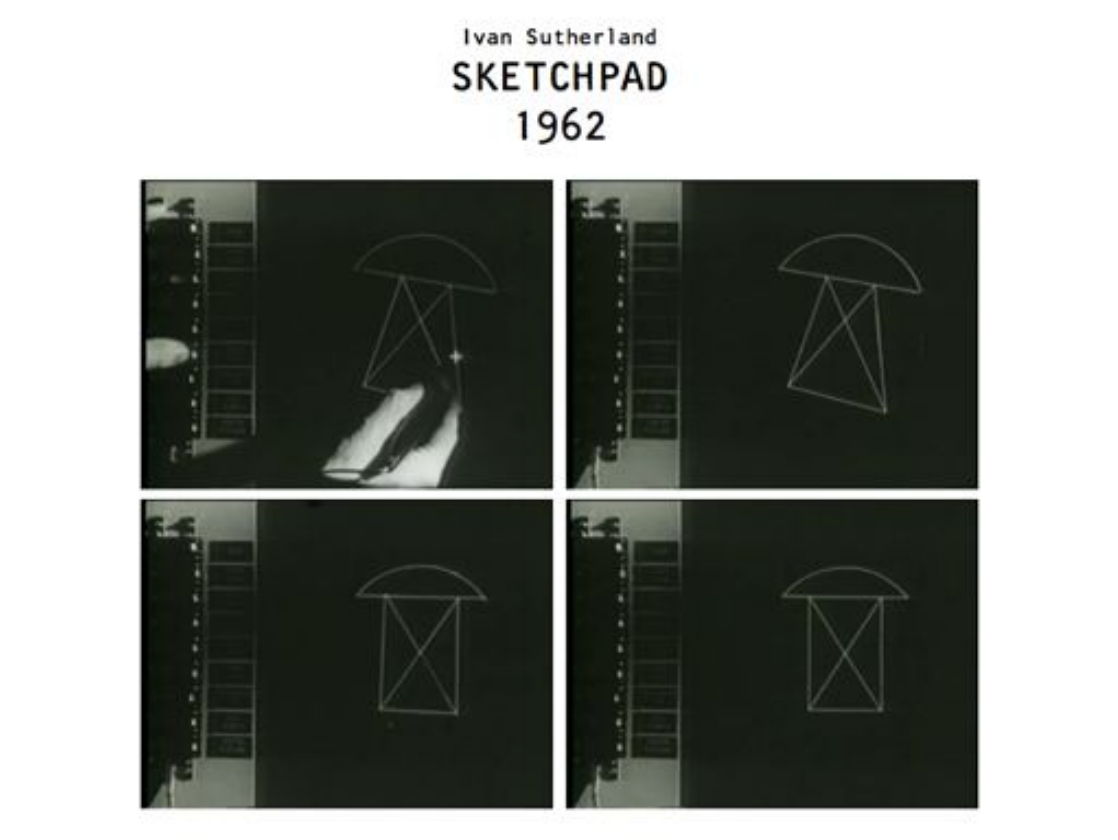
Functional Reactive UIs with Elm

@g0wda

#FunctionalConf

UIs when they began



“The Future of Programming”
Bret Victor, 2013

And then the web happened...

UI development in 2014

HTML + CSS + JavaScript

“...And then suddenly you try to center something vertically...”

-- @evancz

What UIs really are:

```
view = f(data)
```

Elm?

FRP?

FRP?
Functional

FRP?
Purely Functional

FRP?
Purely Functional
Reactive

Why should you be excited about Elm?

- GUI programming made pleasant
  - No callbacks, broken stack traces
  - Functionally compose GUIs
  - Code looks like the data-flow
- Types encourage correctness
  - If your code compiles, a fair amount of correctness is already proved.
- Space-age debugging & unlimited Undo
  - Just another day at the purely functional school

Getting Started

- Elm website elm-lang.org
- Try Elm: elm-lang.org/try
- Documentation: library.elm-lang.org/catalog
- Elm platform: github.com/elm-lang/elm-platform
- These slides: bit.ly/ElmFuConf2014
- Code: github.com/shashi/fuconf-talk

Diving in

Values

- `1 : number`
- `1.0 : Float`
- `“1” : String`
- `[markdown|`
  `# Hello FuConf`
  `|] : Element`
- `[1,2,3] : number`
- `(1,2, “3”) : (number, number, String)`
- `{id=007, name=”Bond”} : {id : number, name : String}`

Functions

- ```
  add : number -> number -> number
  add a b = a + b
  ```
- ```
  sum lst = case lst of
      [] -> 0
      x::xs -> x + sum xs
  ```
- ```
  \a b -> a + b
  ```
- ```
  map (\a -> a+1) [1, 2, 3]
  ```
- ```
  sum lst = foldl (+) 0 lst
  ```
  ```

Displaying Values

```
main : Element

plainText : String -> Element
asText : a -> Element
```

What UIs really are:

```
view = f(data)
```

What UIs really are:

$$view' = f(data')$$

What UIs really are:

$$view_t = f(data_t)$$

when data updates with external input, the view should too

`Signal a`

`Signal a` is a time-varying value of type a.

```
lift : (a -> b) -> Signal a -> Signal b
(<~)
```

Some signals

- `Window.dimensions`
  `: (Float, Float)`
- `Mouse.position : (Float, Float)`
- `Keyboard.arrows`
  `: Signal { x:Int, y:Int }`
- `fps 30  : Signal Float`
- `every 1 : Signal Float`

```
Signal b = f <~ Signal a
```

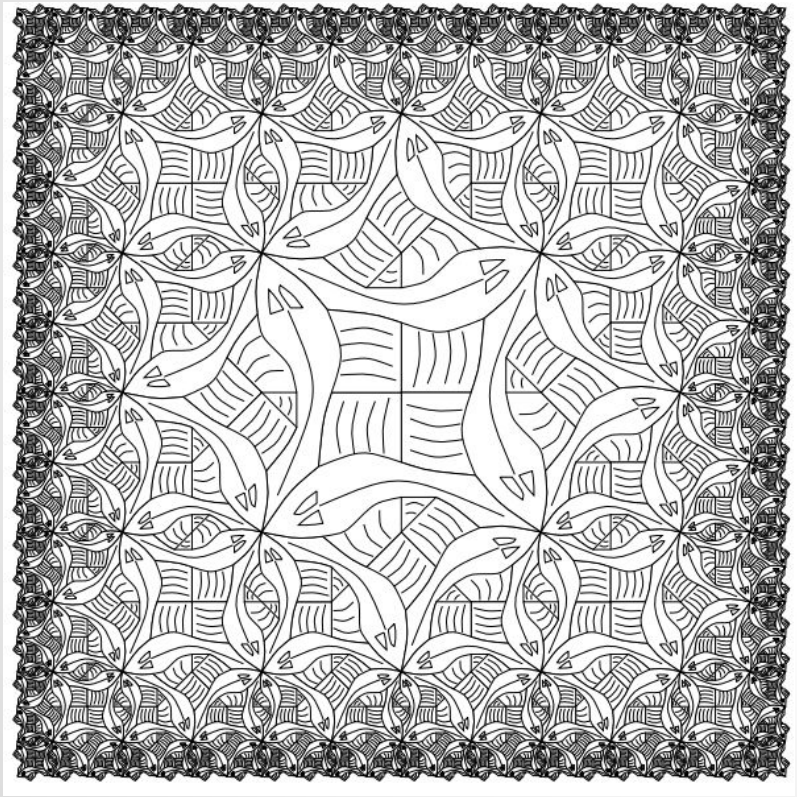
```
Signal view = f <~ Signal data

main : Signal Element
```

In pursuit of

$$f_{view} : a \rightarrow Element$$

functionally composed graphics



“Functional Geometry”
Peter Henderson

Shapes and Paths

A **Shape** is a geometry

A **Path** is a collection of points

- `circle : Float -> Shape`
- `ngon : Int -> Float -> Shape`
- `square, polygon, etc.`
- `path : [(Float, Float)] -> Path`

Forms

A **`Form`** lends itself to styling, grouping & 2D transforms

- `filled : Color -> Shape -> Form`
- `traced : LineStyle -> Path -> Form`
- `group  : [Form] -> Form`

Collages

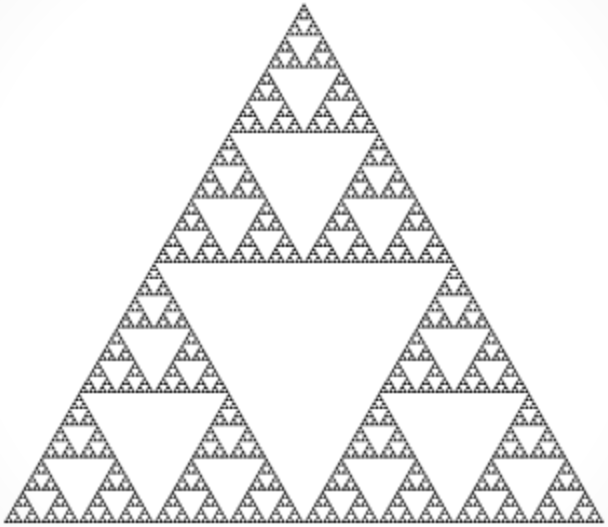
Put `Forms` into a displayable `Element`

```
collage : Int -> Int ->
    [Form] -> Element
```

transforms on Forms

- `move:(Float, Float) -> Form -> Form`
- `rotate : Float -> Form -> Form`
- `scale  : Float -> Form -> Form`

Sierpinski's triangle

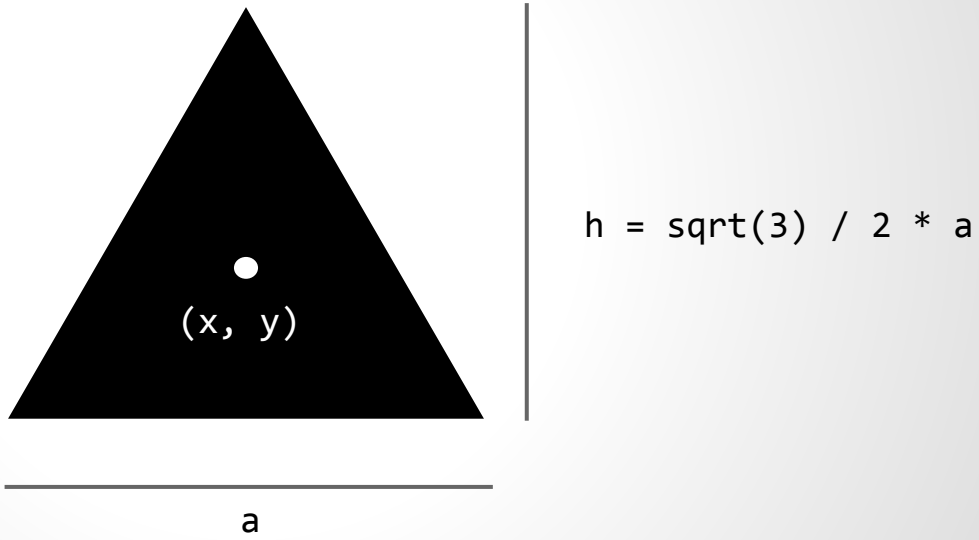


Sierpinski's triangle

- to draw an n-step sierpinski's triangle
  - `if n==0`: draw a triangle
  - `otherwise`: draw (n-1)-step sierpinski's triangle each in the 3 sub-triangles

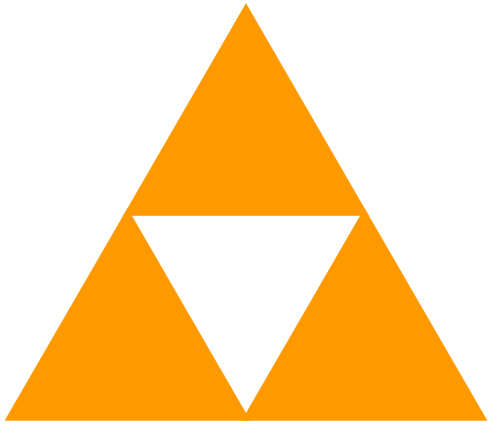
Sierpinski's triangle

```
sierpinsky (x, y) a n =
    if n==0 then
```



Sierpinski's triangle

```
if n>0 then
```



where  = sierpinski <new center> (h/2) (n-1)

Layouts

```
flow : Direction -> [Element] -> Element
```

Directions can be:

`up, down, right, left, inward, outward`

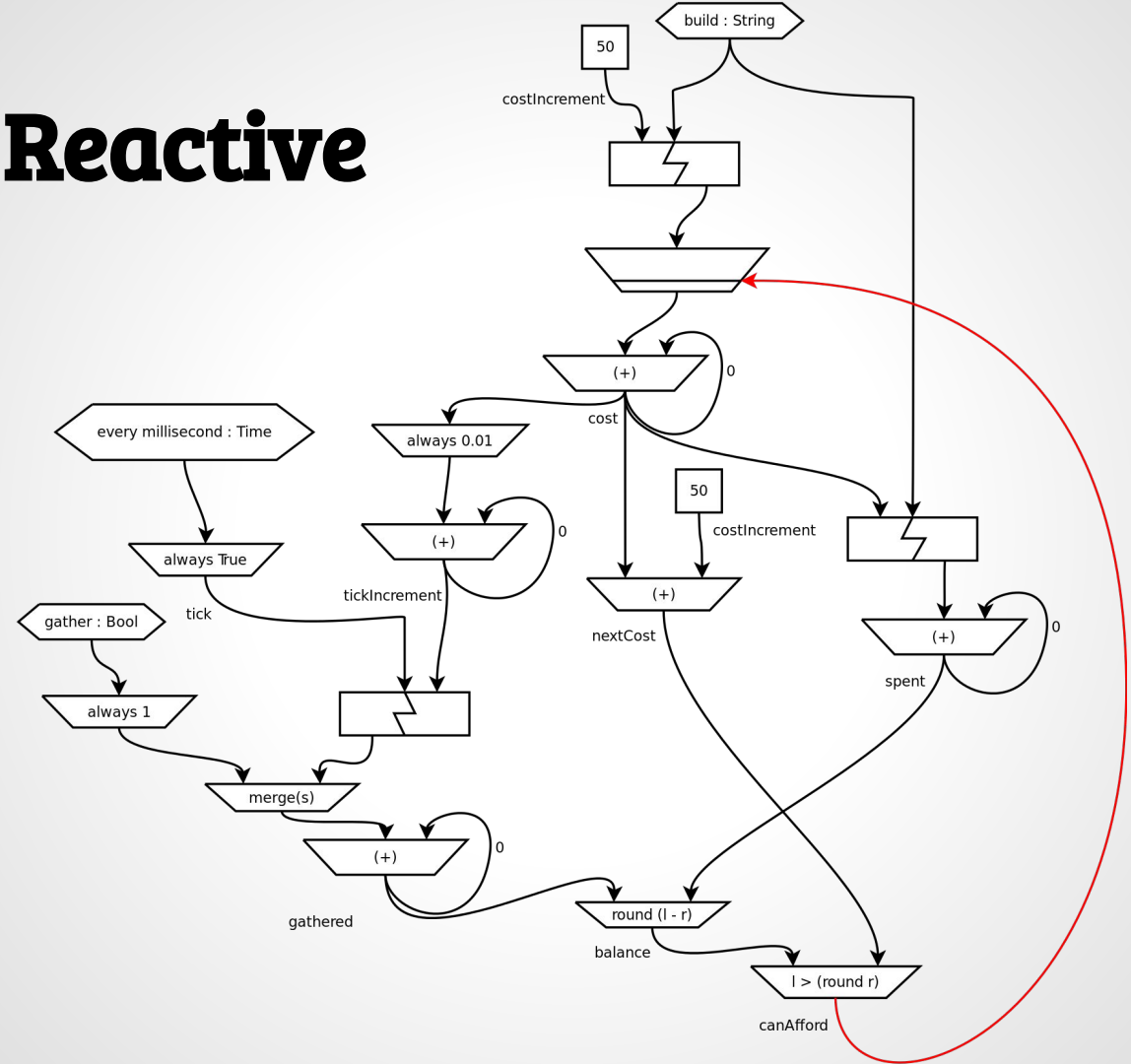
Layouts

```
container : Int -> Int -> Position ->
        Element -> Element
```

Position can be:

```
middle, midTop, midBottom, midLeft,
  midRight, topLeft, topRight,...
    midTopAt, bottomRightAt, ...
```

Going Reactive



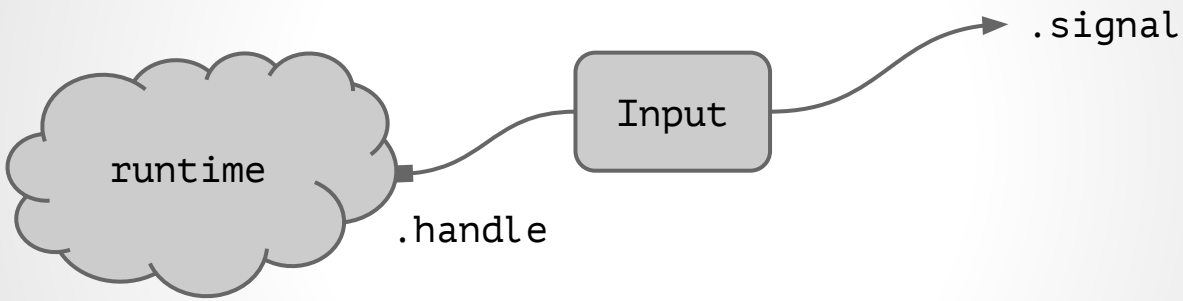
Making more signals

- `foldp : (a -> b -> b) -> b -> Signal a -> Signal b`
- `lift2 : (a -> b -> c) -> Signal a -> Signal b -> Signal c`
- `merge : Signal a -> Signal a -> Signal a`
- `sampleOn : Signal a -> Signal b`

Making more signals

- `keepIf`
- `keepWhen`
- `dropIf`
- `dropWhen`
- `dropRepeats`
- `count`
- `countIf`

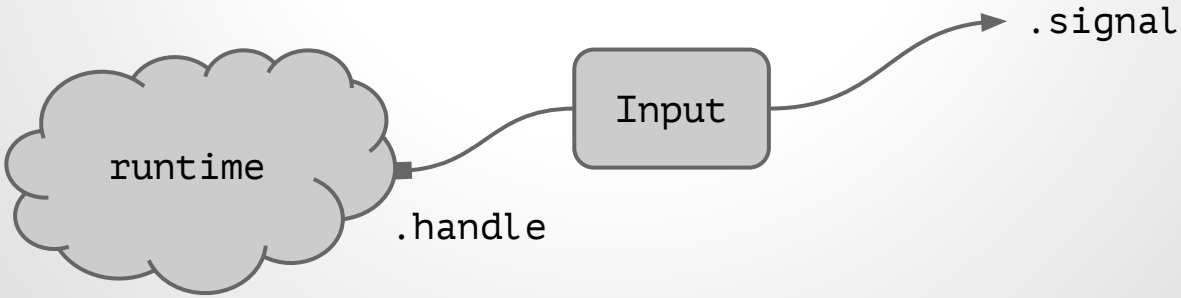
Interactive UI elements



Input

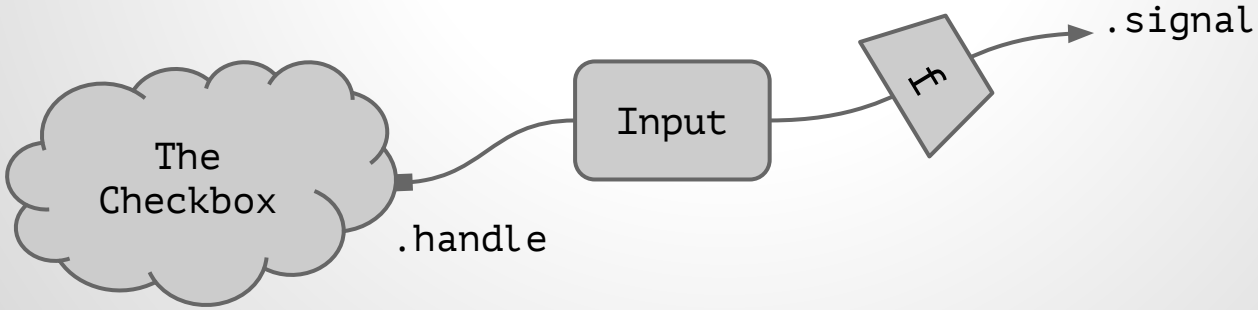
- `type Input a =`
  `{ signal : Signal a`
  `, handle : Handle a }`

- `input : a -> Input a`



Interactive UI elements

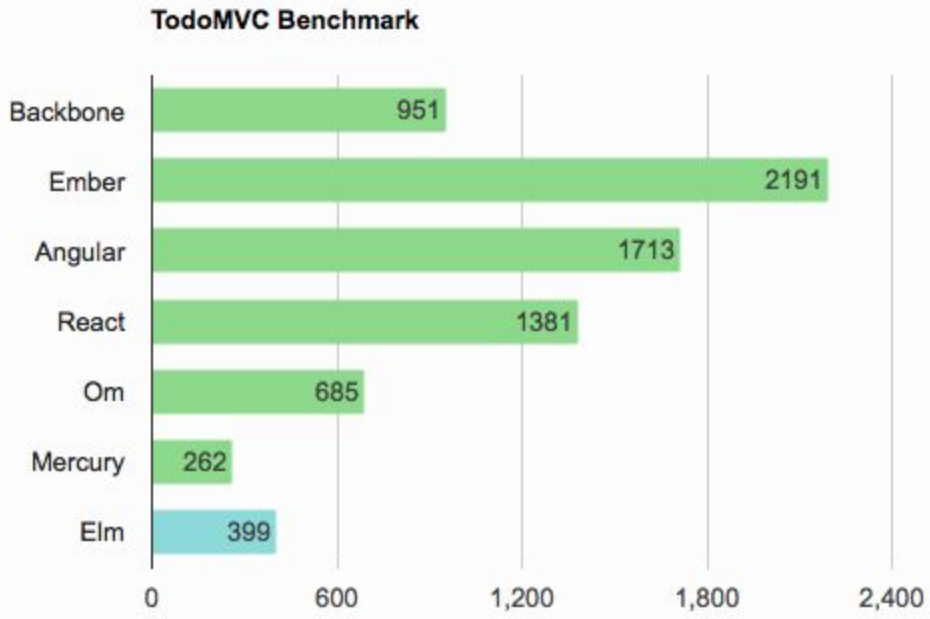
```
check = input False
checkbox check.handle f default
```



elm-html

```
node : String ->
       [Attribute] ->
       [CssProperty] ->
       [Html] ->
       Html
```

```
patch = diff(view', view)
DOM = apply_patch(patch, DOM)
```


TodoMVC Benchmark
Backbone
951
Ember
2191
Angular
1713
React
1381
Om
685
Mercury
262
Elm
399
0
600
1,200
1,800
2,400
Average time in milliseconds over 16 runs (lower is better)
Firefox 30 on MacBook Air with OSX 0.10.9.4

Ports
“Hello, JS!”

```
module Chat where

-- incoming messages typed by your chat parter
port messageIn : Signal String

myTextInput = … -- Singal String
-- outgoing messages typed in by you
port messageOut : Signal String
port messageOut = myTextInput
```

```
var chat = Elm.embed(
    Elm.Chat,
    div,
    { messagesIn : "" });
```

```
chat.ports.messageIn.send("hey");

chat.ports.messageOut .subscribe(
    handler);

chat.ports.messageOut .unsubscribe(
    handler);
```

FRP is the new MVC

- Model
- Update
- Display

Learning more

- http://elm-lang.org/Examples.elm
- WebGL!
- Extensible Records
- Google Group elm-discuss
- Twitter @elmlang

Me

Shashi Gowda
@g0wda
github.com/shashi

Thank you! <3