# FRHTTP & Functional Reactive Programming in NodeJS*

*For non-PhDs

# Unfortunately many people's intro to FRP looks something like this

isomorphism

monad

lift

monoidal functor

flatmap

lambda calculus

lens


I just wanted to make something cool…

Let’s try a gentler approach


THE IMPURITY OF THINE FUNCTOR HATH CAUSED OFFENSE
AND THINE MONAD HATH NO LIFT

FRP isn’t new

# What is FRP?

FRP is a programming methodology
based on **composing pure functions**
that respond to **values over time**

# Characteristics of a pure function

Maintains no state

Operates on only it’s inputs

Does not alter it’s inputs

Produces repeatable output (if the function has the other characteristics this one just happens)

# The alternative to pure functions

Functions who's output is not solely based on inputs

Functions that change the system

These are **Side Effects** (we want to minimize these and keep them at the edges)

# Values over time

We represent this with a concept called “streams”

Something puts a value on a stream, and the stream calls our function with that value as input.

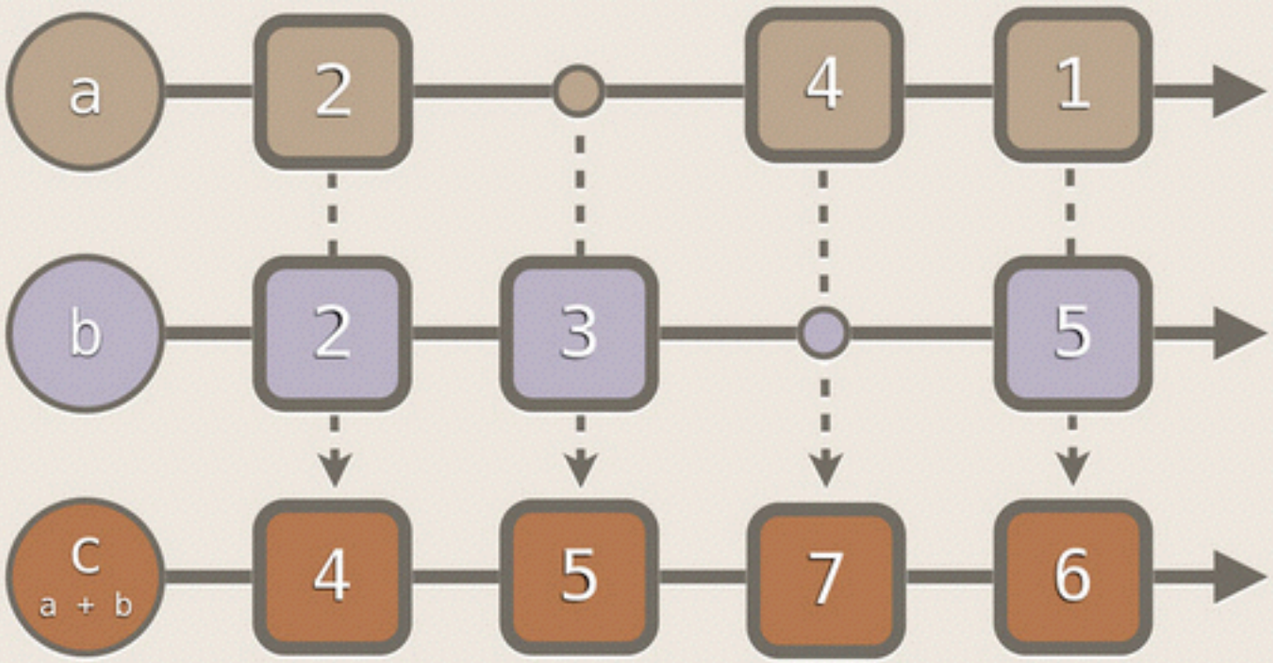
This is what makes us “reactive”

Streams are also where we compose functions to create complex behaviors from simple building blocks

Have you ever used a spreadsheet?

Then you've done FRP!  Let's look…

```
a := 2
b := 2
c := a + b
```



# Benefits

Easy to reason about,
removing state makes lots
of inconsistency go away.

# Composable - We can compose complex behaviors from simple behaviors

Lazy - Everything is lazy evaluated, meaning we don’t pay a runtime cost for things we don’t do

Code reuse becomes much easier as functions are small and perform singular transformations.

Code correctness is provable and tests are easy to write

Errors are easy to reproduce and remedy, even if they happen in production.

# No more of this…

The need for “Future proofing” is minimized/eliminated because adding behavior/ refactoring is so easy.


CAUTION
THIS SIGN HAS
SHARP EDGES
DO NOT TOUCH THE EDGES OF THIS SIGN
ALSO, THE BRIDGE IS OUT AHEAD

# No more Callback Hell!

# FRHTTP

Functional Reactive HTTP

github.com/ayasin/frhttp

“Simplicity is prerequisite for reliability.”

**–Edsger W. Dijkstra**

# FRHTTP Basic Concepts

when - when some data is ready, run me

render - when nothing else can happen,
render the results

# when

**when(name, in, out, fn, advancedOpts);**

name : used for logging and debugging

in : an array of the names of the params you need

out : an array of the params you produce

fn : the function to run

advancedOpts : takeMany, triggerOn, enter, exit

This is chainable : when().when().when()…

# The function in the when block

**function (emit, input);**

emit : an object with a value method and done method

input : an object with each of your requested inputs

# Render

**render(in, fn);**

in : an array of the names of the params you need

fn : the function to run

# The function in the render block

**function (writer, input);**

writer : an object with several methods for sending data

input : an object with each of your requested params

# One more thing…

**inject ({field: val});**

Allows you to “inject” objects into a route’s execution

Remember, we want to not use global vars, this lets us avoid that.

# Making a route

```
var FRHttp = require('frhttp'),
      server = FRHttp.createServer();
      server.GET(path).onValue(function (route) { /* your code here */});
      server.listen(9000);
```

Supported Verbs:

GET

HEAD

PUT

POST

DELETE

NON_REST

# But I have an existing ExpressJS app…

No problem…

```
var server = require(frhttp).createServer();

// set up a bunch of routes

app.get(‘/‘, server.runRoute);
```

# Now that we've got the basics, let's go to the IDE