

The Future of Web Development Looks Functional

I, For One, Welcome Our New FP Overlords



(It's not so bad...)

Fast & Loose With
History

A Mental Moore's Law

- Our notion of what a programmer *should do* has ratcheted up
- Conceptual jumps (trade control for programmer efficiency and/or safety)
 - Assembly → HLLs (1950s)
 - Manual memory allocation → Garbage collection (1960s)

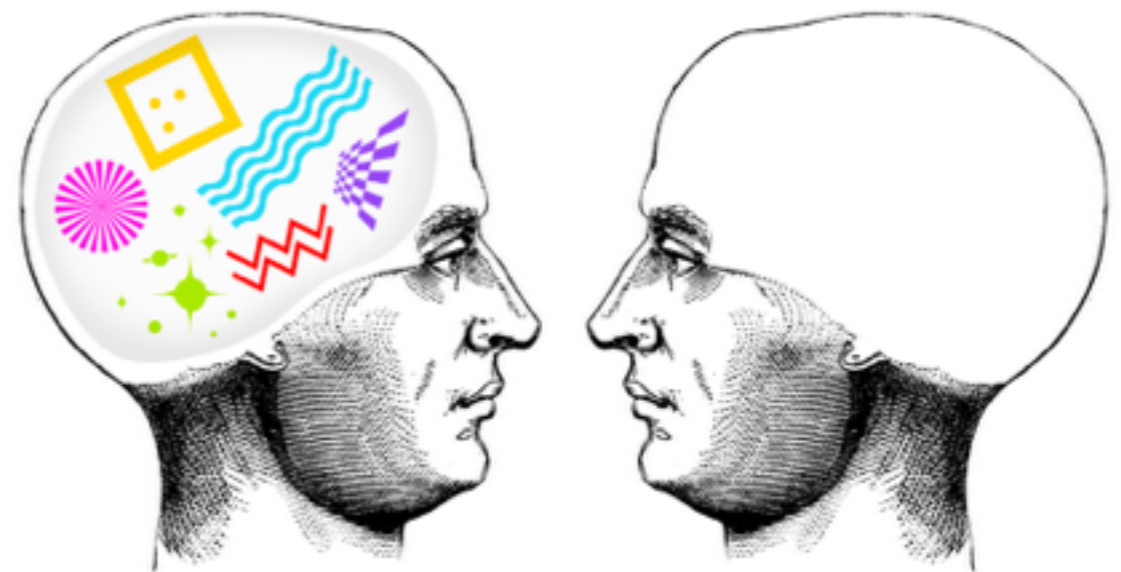
A Mental Model

Yeah, it totally depends on what kind of programming you do. Let's say for "most" programmers

- Our notion of what a programmer *should do* has ratcheted up
- Conceptual jumps (trade control for programmer efficiency and/or safety)
 - Assembly → HLLs (1950s)
 - Manual memory allocation → Garbage collection (1960s)

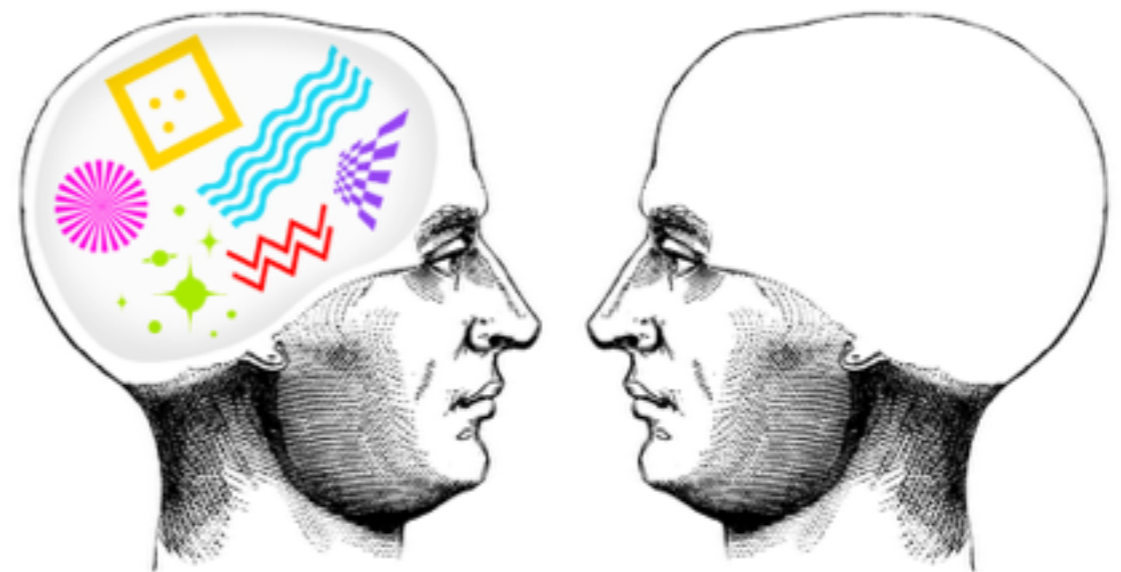
"Abstraction"

- Not a dirty word!
- "The purpose of abstraction is not to be vague, but to create a *new semantic level* in which one can be *absolutely precise*." — Edsger W. Dijkstra
- My emphasis:
"New semantic level"



"Abstraction"

- Not a dirty word!
- "The purpose of abstraction is not to be vague, but to create a *new semantic level* in which one can be *absolutely precise*." — Edsger W. Dijkstra
- My emphasis:
"New semantic level"



Meanwhile, on the
Web

Web

- Designed for high-energy particle physicists by high-energy particle physicists
- A hypertext document system, but a glimmer:

"If one sacrifices portability, it is possible [to] make following a link fire up a special application, so that diagnostic programs, for example, could be linked directly into the maintenance guide."

—Tim Berners-Lee *"Information Management: A Proposal"*

The Web advanced a lot

- Now, essentially:
 - Application programming environment
 - Software delivery system

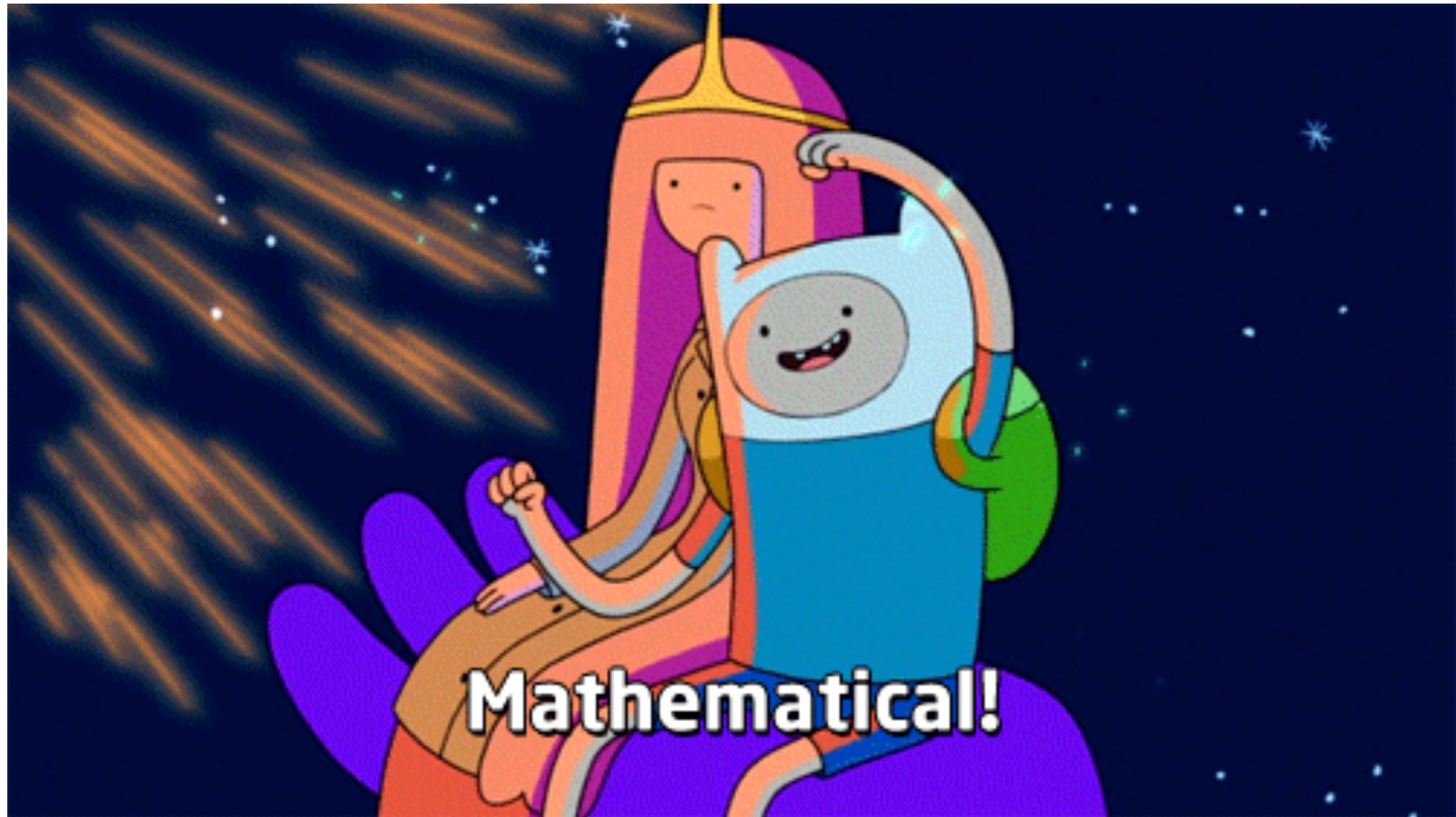
What now?

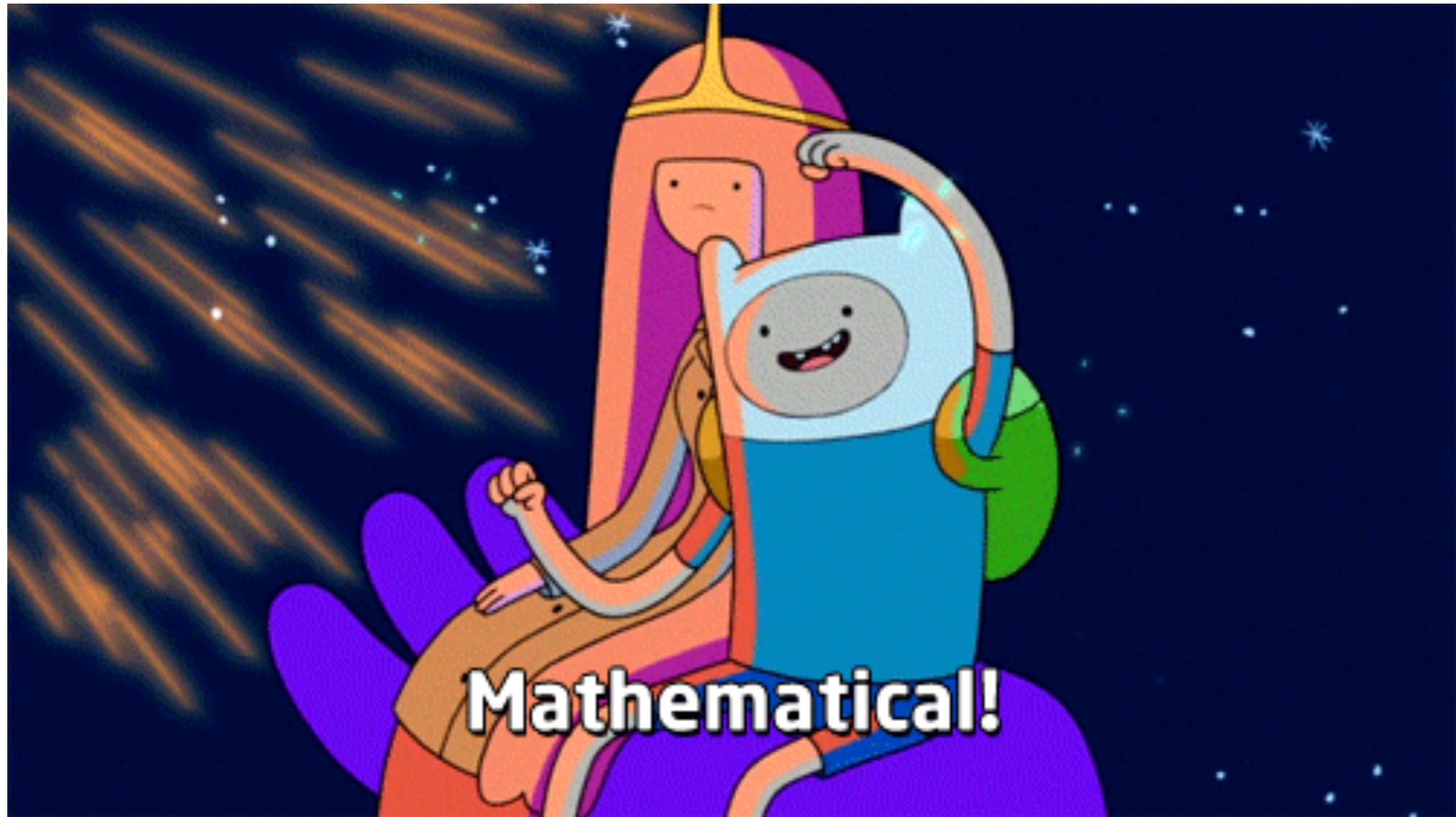
- The Web's changed a lot!
- The way we develop programs should change as well
- Let's look in the toolbox for stuff to help...

Playing to Strengths

Functional Programming

- *High-level* - work with abstractions, saving time & effort
- *Referentially transparent* - same function, same arguments? Same answer
- *Immutable* - values don't change
- Incidental but: *many stylistic differences*





Typed

- Not *synonymous* with FP, but closely linked
- Catch errors early
- Express domain constraints
- Reduce test load


```
fun split [] = ([], [])  
  | split [h] = ([h], [])  
  | split (x::y::t) = let val (s1, s2) = split t  
                        in (x::s1, y::s2)  
                        end
```

```
fun merge ([], x) = x::int list  
  | merge (x, []) = x  
  | merge (h1::t1, h2::t2) =  
    if h1 < h2 then h1::merge(t1, h2::t2)  
    else h2::merge(h1::t1, t2)
```

```
fun sort_1 [] = []  
  | sort_1 x = let val (p, q) = split x  
                in merge (sort_1 p, sort_1 q)  
                end
```

```
fun split [] = ([], [])
  | split [h] = ([h], [])
  | split (x::y::t) = let val (s1, s2) = split t
                       in (x::s1, y::s2)
                       end
```

```
fun merge ([], x) = x::int list
  | merge (x, []) = x
  | merge (h1::t1, h2::t2) = merge(h1::t1, h2::t2)
  | merge (h1::t1, []) = merge(h1::t1, [])
```

Inferred type should be:
int list -> int list

```
fun sort_1 [] = []
  | sort_1 x = let val (p, q) = split x
                in merge (sort_1 p, sort_1 q)
                end
```

```

fun split [] = ([], [])
  | split [h] = ([h], [])
  | split (x::y::t) = let val (s1, s2) = split t
                      in (x::s1, y::s2)
                      end

```

```

fun merge ([], x) = x::int list
  | merge (x, []) = x
  | merge (h1::t1, h2::t2) = merge(h1::t1, h2::t2)
  | merge (h1::t1, []) = merge(h1::t1, [])

```

Inferred type should be:
int list -> int list

```

fun sort_1 [] = []
  | sort_1 x = let val (p, q) = split x
                in merge (sort_1 p, sort_1 q)

```

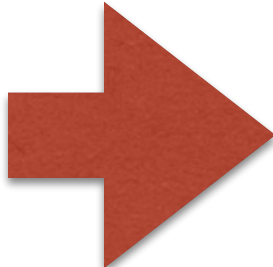
BUT! It is:
'a list -> int list

"Diff"

```
fun sort_2 [] = []  
+ | sort_2 [x] = [x]  
  | sort_2 x = let val (p, q) = split x  
               in merge (sort_2 p, sort_2 q)  
               end
```

REAL WORLD ACADEMIA


WEDNESDAY, JUNE 13, 2012



Unit testing isn't enough. You need static typing too.

When I was working on my research for my Masters degree I promised myself that I would publish my paper online under a free license, as

SUBSCRIBE TO

 Posts 

 Comments 

- Moral: static typing & tests catch *different bugs*
- You need both

Unifying Platforms

Idea

- Let's tailor the languages we use to the task at hand
 - Writing apps for the web
- This should pay off in terms of simplicity of development...

"Unifying"

- Combine or collapse heterogeneous tech platforms
 - CSS + HTML + JS (**Elm**)
 - Client + Server (**Opa**, Ur/Web)
 - OS + Server + Application (MirageOS)

"Unifying"

- I won't have time to cover all this!
- Introduce the ideas & get you thinking.

Elm



+



+



- *FRP* - Functional Reactive Programming
- Combine HTML, CSS, and JavaScript into a unified UI layer

"Hello World"


 **evancz / elm-todomvc**

Proper implementation of the TodoMVC app

 **25** commits

 **3** branches



 branch: **master** ▾

elm-todomvc / +

Merge branch '0.15'



evancz authored 3 days ago

 [.gitignore](#)

Make changes for 0.14

 [LICENSE](#)

Initial commit

How it works

- Architecture:* Model, Update, View
 - *Model* holds all app state
 - *Update* combines an action & the current model to produce a new model
 - *View* displays this all to the user

*<http://elm-lang.org/learn/Architecture.elm>

Model

Example

Update

```
type alias Model =  
  { tasks : List Task  
  , field : String  
  , uid : Int  
  , visibility : String  
  }
```

```
type alias Task =  
  { description : String  
  , completed : Bool  
  , editing : Bool  
  , id : Int  
  }
```

```
type Action  
  = NoOp  
  | UpdateField String  
  | EditingTask Int Bool  
  | UpdateTask Int String  
  | Add  
  | Delete Int  
  | DeleteComplete  
  | Check Int Bool  
  | CheckAll Bool  
  | ChangeVisibility String
```

Example

```
update : Action -> Model -> Model
```

```
update action model =
```

```
  case action of
```

```
    NoOp -> model
```



Adding a new
TODO item

```
Add ->
```

```
  { model |
```

```
    uid <- model.uid + 1,
```

```
    field <- "",
```

```
    tasks <-
```

```
      if String.isEmpty model.field
```

```
        then model.tasks
```

```
        else model.tasks ++ [newTask model.field  
                              model.uid]
```

```
  }
```

Example

```
update : Action -> Model -> Model
```

```
update action model =
```

```
  case action of
```

```
    NoOp -> model
```

Adding a new
TODO item

```
Add ->
```

```
  { model |
```

```
    uid <- model.uid + 1,
```

```
    field <- "",
```

```
    tasks <-
```

```
      if String.isEmpty model.field
```

```
        then model
```

```
        else model
```

cases for other
actions
follow...

```
[newTask model.field  
 model.uid]
```

```
  }
```

Example

model : Signal Model

model =

Signal.foldp update initialModel actions.signal

update : Action -> Model -> Model

update = -- as before...

view : Address Action -> Model -> Html

view address model = -- renders html...

main : Signal Html

main =

Signal.map (view actions.address) model

Example

*blank model
(i.e. no TODOs)*

model : Signal Model

model =

Signal.foldp update initialModel actions.signal

update : Action -> Model -> Model

update = -- as before...

view : Address Action -> Model -> Html

view address model = -- renders html...

main : Signal Html

main =

Signal.map (view actions.address) model

Example

blank model
(i.e. no TODOs)

Incoming
actions from the
view

model : Signal Model

model =

Signal.foldp update initialModel actions.signal

update : Action -> Model -> Model

update = -- as before...

view : Address Action -> Model -> Html

view address model = -- renders html...

main : Signal Html

main =

Signal.map (view actions.address) model

Example

model : Signal Model

model =

Signal.foldp update initialModel actions.signal

blank model
(i.e. no TODOs)

Incoming
actions from the
view

update : Action -> Model -> Model

update = -- as before...

Big **case** over
all input
actions

view : Address Action -> Model -> Html

view address model = -- renders html...

main : Signal Html

main =

Signal.map (view actions.address) model

Example

```
model : Signal Model
```

```
model =
```

```
Signal.foldp update initialModel actions.signal
```

blank model
(i.e. no TODOs)

Incoming
actions from the
view

Place to send UI
actions (e.g.
UpdateTask)

```
Action -> Model -> Model
```

```
-- as before...
```

Big **case** over
all input
actions

```
view : Address Action -> Model -> Html
```

```
view address model = -- renders html...
```

```
main : Signal Html
```

```
main =
```

```
Signal.map (view actions.address) model
```

Example

```
model : Signal Model  
model =
```

blank model
(i.e. no TODOs)

Incoming
actions from the
view

```
Signal.foldp update initialModel actions.signal
```

Place to send UI
actions (e.g.
UpdateTask)

```
Action -> Model -> Model  
-- as before...
```

Big **case** over
all input
actions

```
view : Address Action -> Model -> Html  
view address model = -- renders html...
```

```
main : Signal Html  
main =
```

Model -> Html

```
Signal.map (view actions.address) model
```

Go check it out

- But maybe not now: <https://github.com/evancz/elm-todomvc/blob/master/ToDo.elm>
- Live version: <http://evancz.github.io/elm-todomvc/>

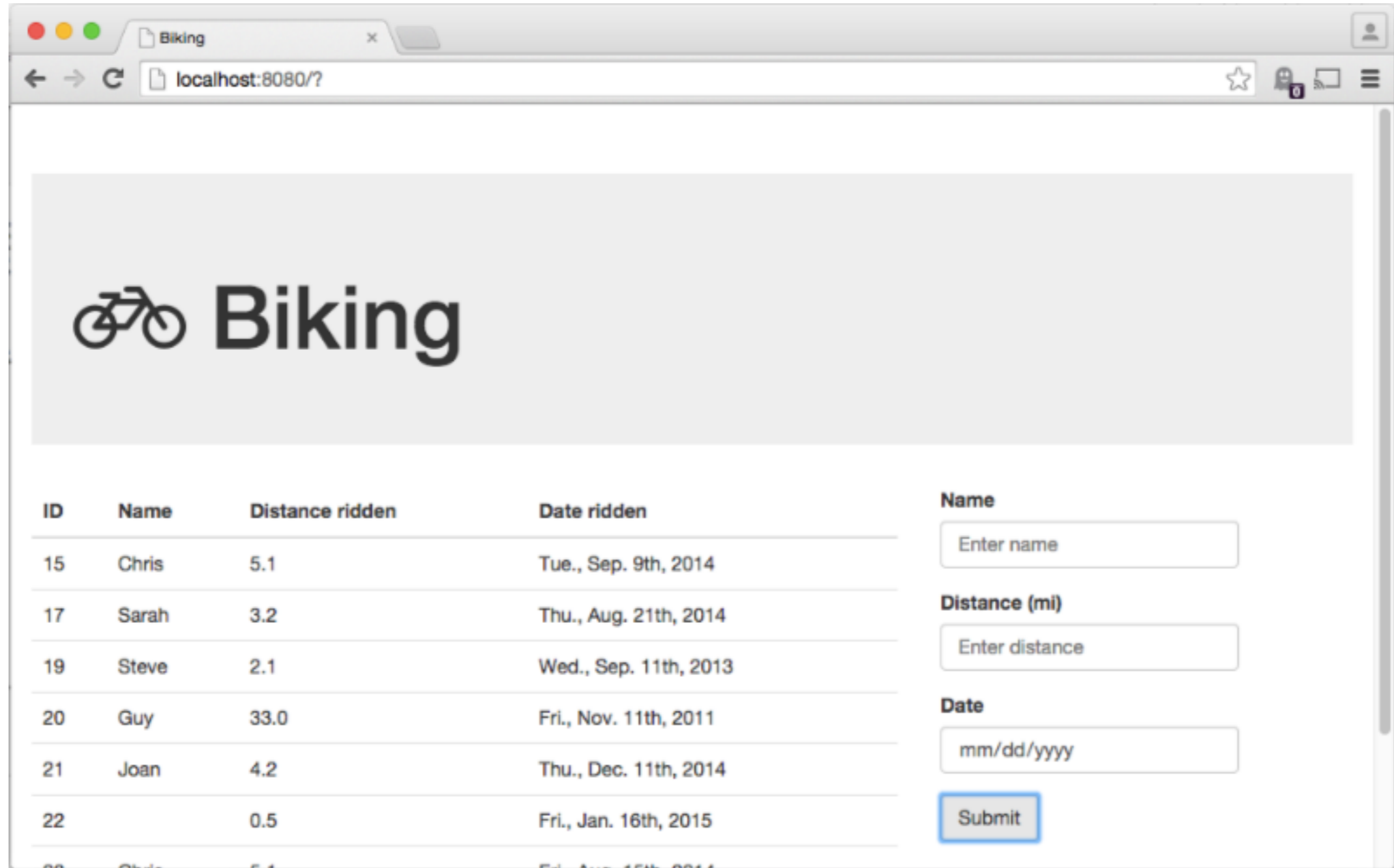
Opa

- Idea: combine client & server into one codebase
- Code is similar to JS but is statically typed / inferred

Compile to...

- On the server:
 - JavaScript for Node.js/Mongo
- On the client:
 - JavaScript for HTML & client-side JS

Biking!



The screenshot shows a web browser window with the address bar displaying 'localhost:8080/?'. The page content includes a header with a bicycle icon and the title 'Biking'. Below the header is a table of biking records and a form to add a new record.

ID	Name	Distance ridden	Date ridden
15	Chris	5.1	Tue., Sep. 9th, 2014
17	Sarah	3.2	Thu., Aug. 21th, 2014
19	Steve	2.1	Wed., Sep. 11th, 2013
20	Guy	33.0	Fri., Nov. 11th, 2011
21	Joan	4.2	Thu., Dec. 11th, 2014
22		0.5	Fri., Jan. 16th, 2015

The form on the right side of the page includes the following fields:

- Name**:
- Distance (mi)**:
- Date**:
- Submit**:

MVC

```
database biking {  
  ride /rides[/{id}]  
  int /index = 0  
}
```

```
module Model {  
  function create_ride(string user_name,  
                      string dist,  
                      string date) {  
    // parse arguments...  
    /biking/rides[~{id}] <- ~{id, user_name, distance, date}  
    /biking/index <- id + 1  
  }  
}
```

MVC

```
database biking {  
  ride /rides[/{id}]  
  int /index = 0  
}
```



Define
database

```
module Model {  
  function create_ride(string user_name,  
                      string dist,  
                      string date) {  
    // parse arguments...  
    /biking/rides[~{id}] <- ~{id, user_name, distance, date}  
    /biking/index <- id + 1  
  }  
}
```

MVC

```
database biking {  
  ride /rides[/{id}]  
  int /index = 0  
}
```

Define
database

```
module Model {  
  function create_ride(string user_name,  
                      string dist,  
                      string date) {  
    // parse arguments...  
    /biking/rides[~{id}] <- ~{id, user_name, distance, date}  
    /biking/index <- id + 1  
  }  
}
```

Store new bike
ride in database

MVC

```
function page_template(title, content) {  
  html =  
    <div>  
      <div class="navbar navbar-fixed-top">  
        <div class=navbar-inner>  
          <div class=container>  
            <a class=brand href="./index.html"></a>  
          </div>  
        </div>  
      </div>  
      <div id=#main class=container-fluid>  
        {content}  
      </div>  
    </div>  
  Resource.page(title, html)  
}
```

MVC

literal
HTML

```
function page_template(title, content) {  
  html =  
    <div>  
      <div class="navbar navbar-fixed-top">  
        <div class=navbar-inner>  
          <div class=container>  
            <a class=brand href="./index.html"></a>  
          </div>  
        </div>  
      </div>  
      <div id=#main class=container-fluid>  
        {content}  
      </div>  
    </div>  
  Resource.page(title, html)  
}
```

MVC

literal
HTML

```
function page_template(title, content) {  
  html =  
    <div>  
      <div class="navbar navbar-fixed-top">  
        <div class=navbar-inner>  
          <div class=container>  
            <a class=brand href="./index.html"></a>  
          </div>  
        </div>  
      </div>  
      <div id=#main class=container-fluid>  
        {content}  
      </div>  
    </div>  
  Resource.page(title, html)  
}
```

Other templates...
Just use functions to
modularize!

MVC

```
function input_form() {  
  <form>  
    <!-- input fields -->  
    <button type=submit class="btn btn-default"  
  
      onclick={function(_) {  
        name = Dom.get_value(#name)  
        distance = Dom.get_value(#distance)  
        date = Dom.get_value(#date)  
        Model.create_ride(name, distance, date) }}>  
      Submit  
    </button>  
  </form>  
}
```


MVC

```
function input_form() {  
  <form>  
    <!-- input fields -->  
    <button type=submit class="btn btn-default"  
      onclick={function(_) {  
        name = Dom.get_value(#name)  
        distance = Dom.get_value(#distance)  
        date = Dom.get_value(#date)  
        Model.create_ride(name, distance, date) }}>  
      Submit  
    </button>  
  </form>  
}
```

View calls
Model function
(which uses
the DB)

MVC

```
module Controller {  
  dispatcher = {  
    parser {  
      case (.*): View.default_page()  
    }  
  }  
}
```

```
Server.start(Server.http, [  
  { register:  
    // set doctype & etc.  
  },  
  { custom: Controller.dispatcher }  
])
```

Opa

- Most of the code is specifying the view templates
- Callbacks from the view naturally tie into Model functions
- These calls work across the client/server divide

Lots more!

- Ur/Web applies a single statically typed language to:
 - Client/server code, DB & HTML
- Mirage OS - unikernel/library operating system
 - Compiles whole app to run "bare metal" on the Xen hypervisor

Summary

- We don't have to live with the complexity that we've inherited!
- With some hard work & open minds we can *unify* accumulated layers of platforms.

Thanks!

- I'm Chris Wilson
- @twopoint718
- chris@sencjw.com