LuvvieScript

Being An Erlang Dialect That Runs In The Browser

@gordonguthrie
@luvviescript

Why?

By all that is holy, why?

Dev Ops
&
Continuous
Delivery

require

Low
Impedance
Programming

Client Side

NodeJs

Clojure Script

LuvvieScript

Server Side

NodeJs

Clojure

Erlang/OTP

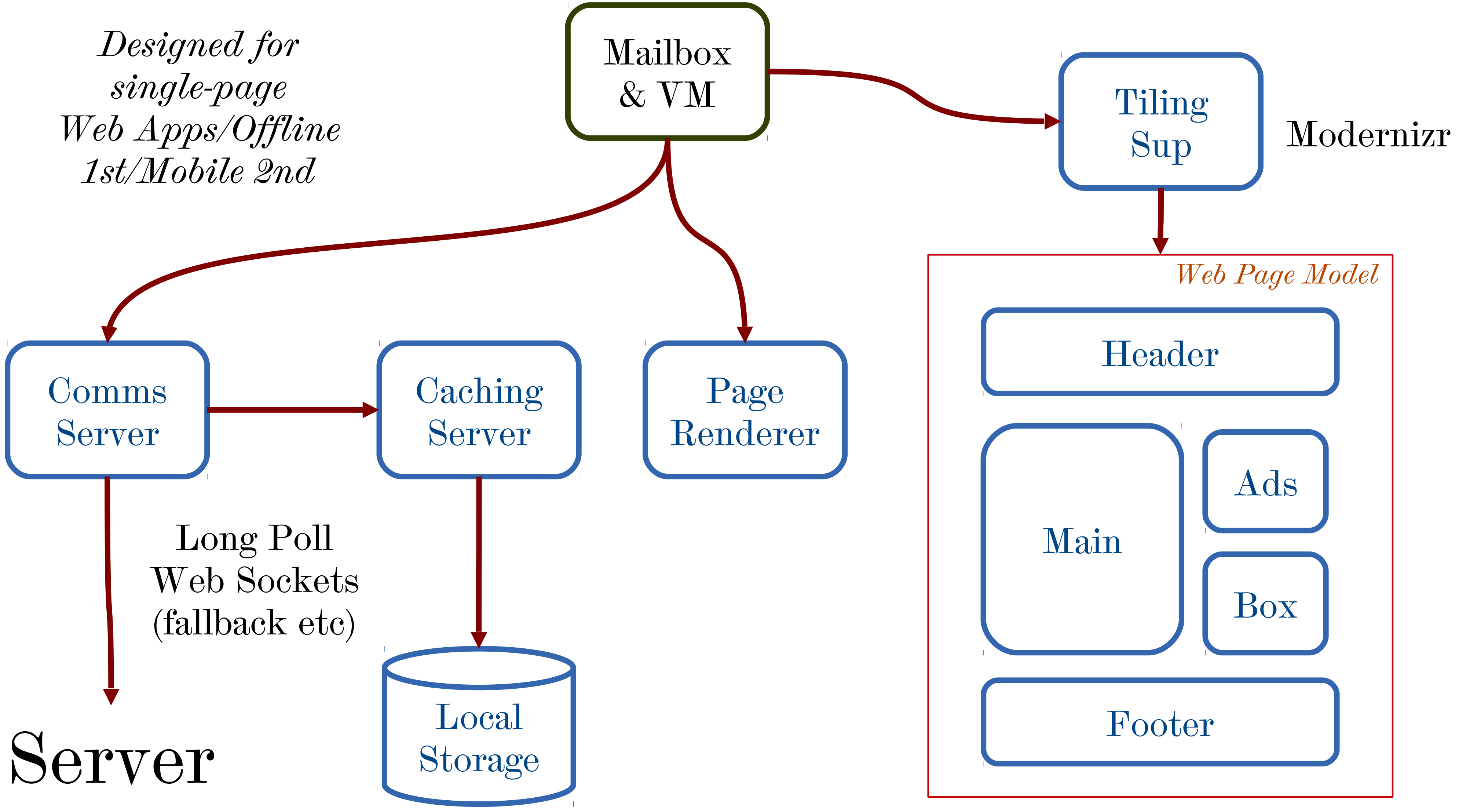
What Is LuvvieScript?

A dom scripting language for web pages
that talk to Erlang/OTP servers
not a general purpose language

A strict sub-set of Erlang
capable of running client & server side

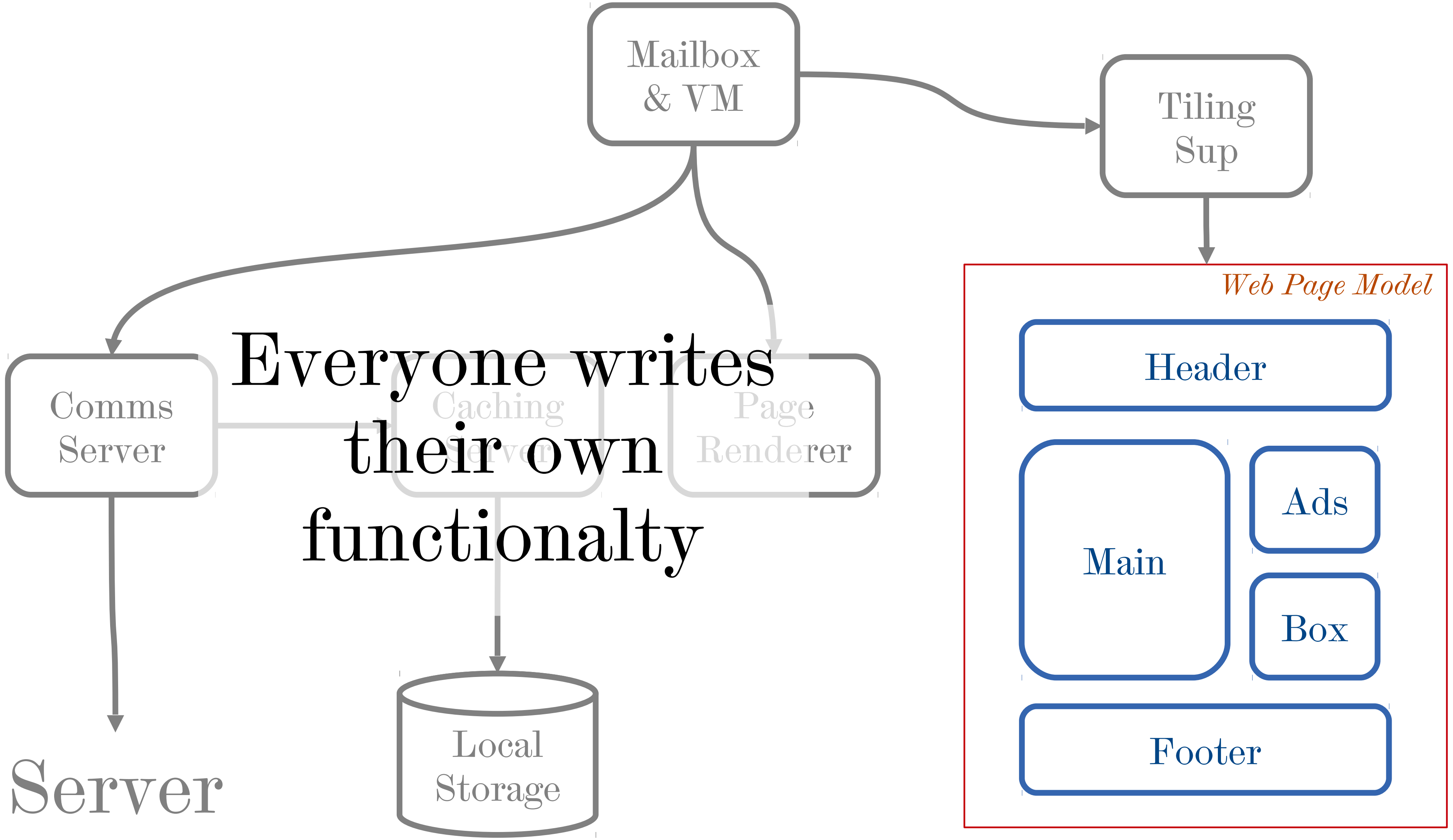
Not an implementation of the Erlang
VM in the browser – Not Erlang on V8

Client Side	*Server Side*
Low concurrency *approx 9*	High concurrency *tens of thousands*
Heavy-weight currency by Web Workers	Light weight concurrency by Erlang processes
No shared code between concurrent processes	Shared code via the Erlang VM and code loader
Code change by page refresh	Hot swapping and code management
Little supervision & no restart	OTP supervision & restart



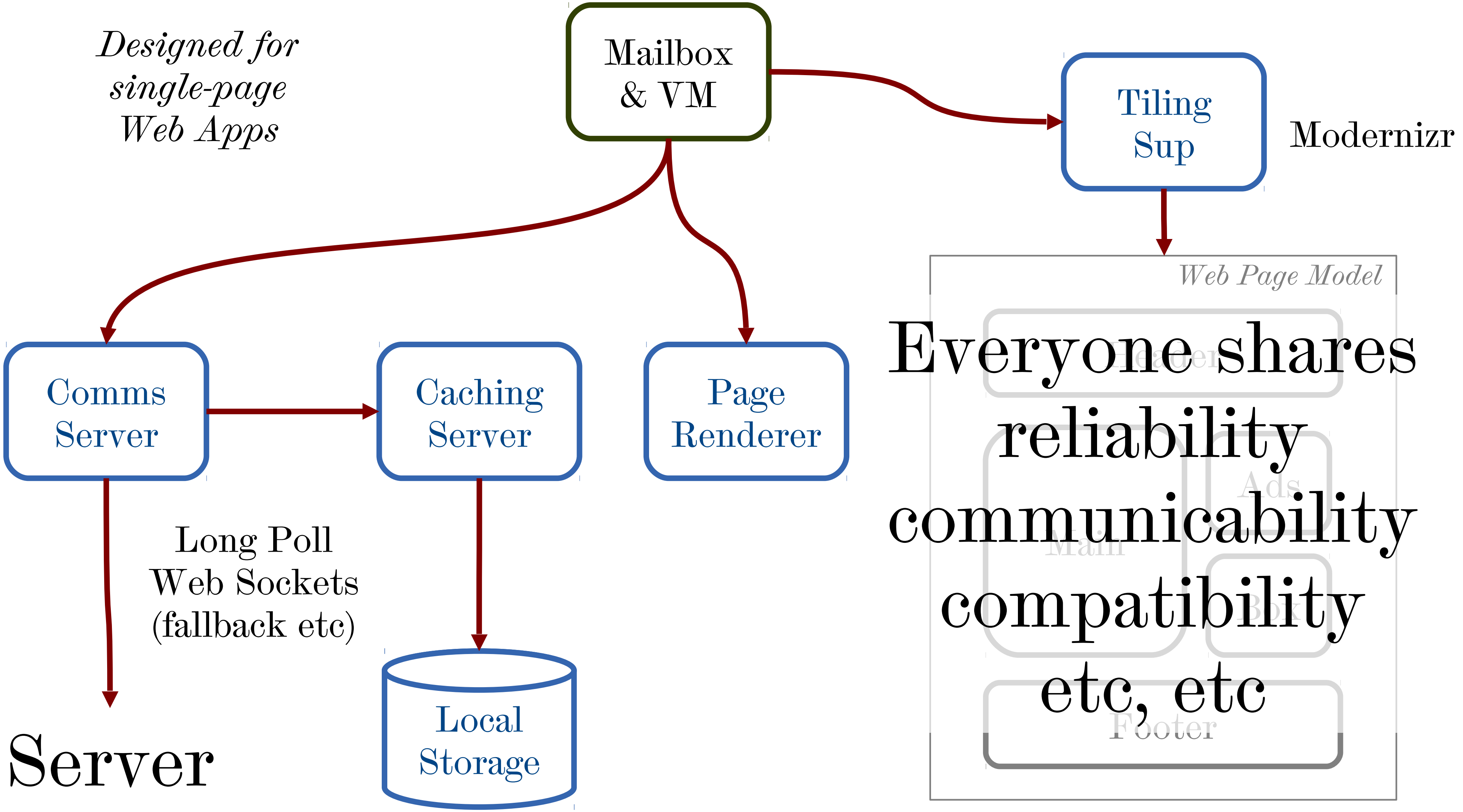
Designed for single-page Web Apps/Offline 1st/Mobile 2nd
Mailbox & VM
Tiling Sup
Modernizr
Web Page Model
Header
Main
Ads
Box
Footer
Comms Server
Caching Server
Page Renderer
Long Poll
Web Sockets
(fallback etc)
Local Storage
Server

-ilities
-alities



Everyone writes their own functionalty

Designed for single-page Web Apps



Everyone shares
reliability
communicability
compatibility
etc, etc

Server-style: Msg Driven Loops

```
handle_call(Msg, From, State) →
    {NewState, SideEffects} = manipulate_state(Msg, State),
    NewState2 = handle_side_effects(SideEffects, State),
    Reply = "whatevs",
    {reply, Reply, NewState2};
```

Normalised Event Handling

```
msg_from_dom
msg_from_server
msg_from_other_web_worker
```

...and Source Maps...

No Source Maps
- no debugging in the browser
- no bloody use...

How?
Exactly?

AST to AST Transpiler

Generate a Core Erlang AST
Transpile to the Mozilla Javascript AST
Monkey about with JS Syntax Tools
Generate JS with **`escodegen.js`**

Erlang

```
-module(demo).

-export([test/0]).

test() ->
    A = first(),
    B = second(),
    C = third(),
    A + B / C.

first() -> 1.

second() -> 2.

third() -> 3.
```

Erlang Core

```
module 'demo' ['module_info'/0,
       'module_info'/1,
       'test'/0]
    attributes []
'test'/0 =
    fun () ->
let <A> =
    apply 'first'/0
()
in  let <B> =
apply 'second'/0
    ()
    in  let <C> =
    apply 'third'/0
()
in  let <_cor3> =
```

Core - Smaller And More Regular

21 primitives - `literal`, `binary`, `bitstr`, `cons`, `tuple`, `var`, `values`, `fun`, `seq`, `let`, `letrec`, `case`, `clause`, `alias`, `receive`, `apply`, `call`, `primop`, `try`, `catch`, `module`

Core AST

```
{c_module,[],
    {c_literal,[],demo},
    [{c_var,[],{module_info,0}},
     {c_var,[],{module_info,1}},
     {c_var,[],{test,0}}],
    [],
    [{{c_var,[],{test,0}},
      {c_fun,
          [7,{file,"test/passing/src/demo.erl"}],
          [],
          {c_let,[],
              [{c_var,[8,{file,"test/passing/src/demo.erl"}],'A'}],
              {c_apply,
                  [8,{file,"test/passing/src/demo.erl"}],
                  {c_var,[8,{file,"test/passing/src/demo.erl"}],{first,
                  []},
              {c_let,[],
                  [{c_var,[9,{file,"test/passing/src/demo.erl"}],'B'}],
                  {c_apply,
```

JS AST

```
"body": [{
        "kind": "var",
        "declarations": [{
                "init": {
                    "type": "ObjectExpression",
                    "properties": []
                },
                "type": "VariableDeclarator",
                "id": {
                    "type": "Identifier",
                    "name": "exports"
                }}],
        "type": "VariableDeclaration"
    },
    {
        "expression": {
            "operator": "=",
            "right": {
                "body": {
```

Node-y JS

```
var exports = {};
exports.test = function () {
    _args = arguments.length;
    switch (_args) {
    case 0:
        return test();
        break;
    default:
    }
};
first = function () {
    _args = arguments.length;
    switch (_args) {
    case 0:
        return 1;
        break;
    default:
        return 'throw error';
    }
```

Source Maps

```
{"version":3,"file":"test/passing/src/.
   ./psrc/demo.erl","sources":
   ["test/passing/src/../psrc/demo.erl"]
   ,"names":
   [],"mappings":";;;;;;;;;;;;;;eAaK,C;;
   ;;;;;;;;eAGA,C;;;;;;;;;;;;;;QATA,C;QA
   CA,C;QACA,C;;;;;;;;;;;;eAUA,C"}
```

```
-module(demo).

-export([test/0]).

test() ->
    A = first(),
    B = second(),
    C = third(),
    A + B / C.

first() -> 1.

second() -> 2.

third() -> 3.
```

Module Declaration

Attributes

Function Declarations

Attributes – A Moveable Feast

Arbitrary erlang terms which have meaning (or no meaning) at different points in the software cycle

```
define    Macros                        Pre-compile Expansion
include                                 Pre-compile Expansion

record    Syntactic Sugar               Compile Time
export                                  Compile Time
spec          Limited Type Checking        Compile Time
type          Limited Type Checking        Compile Time

spec          Full Type Checking           Dialyzer
type          Full Type Checking           Dialyzer

author                                  Run Time Tool Support
vsn                                     Run Time Tool Support
date                                       Run Time Tool
Support
```

Luvvie Script Attributes

```
-dialect({luvviescript, {version, 1}}).
-require({javascript, [_underscore,
  jquery]}).
-require({luvviescript, [lists, sets]}).
```

Model Definition

```
-record(tag, {
              type                           :: html()| pseudo_html(),
              id            = get_id()  :: opaque(),
              class         = []             :: [strings()],
              subscriptions = []             :: [dom_events()| pseudo_events()],
              attrs         = []             :: list(),
              inner         = []             :: [#tag{}]
             }).
```

Type-To-Type Transforms

Not entirely clear how type transforms will be done.

Might use the BERT wire interchange format as the basis.

Not clear how integers will be handled

Roadmap

Get Primitives Working
(With Test Suites)

JS Transforms For Recursion

Implement 'Servers'

Build Runtime Servers

Join us in the beam community in the Google Summer Of Code

@gordonguthrie
gordon@vixo.com
@luvviescript
http://luvv.ie

Goodbye