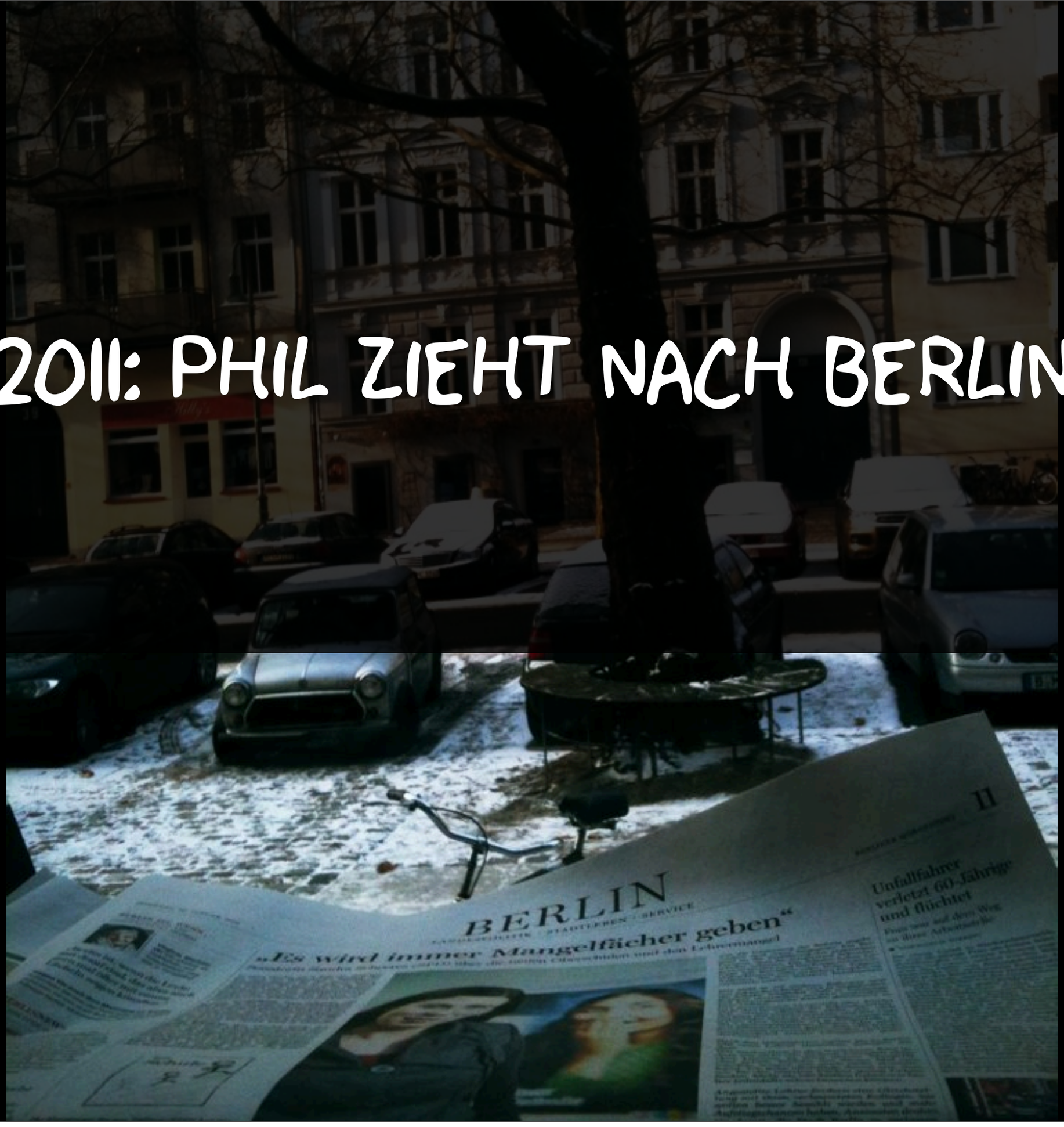
Why code in Node.js often get rejected by SoundCloud

Phil Calçado – SoundCloud
@pcalcado
http://philcalcado.com

2011: PHIL ZIEHT NACH BERLIN




SOUNDCLOUD

> 11 hours of audio uploaded every minute

~ 200 million users / month

The White House*

ladygaga

nineinchnails

Pennywise

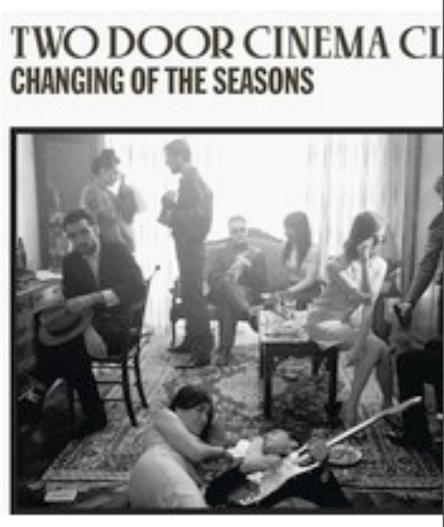
Snoop Dogg

Skrillex

GreenDay

Madonna

TheNational

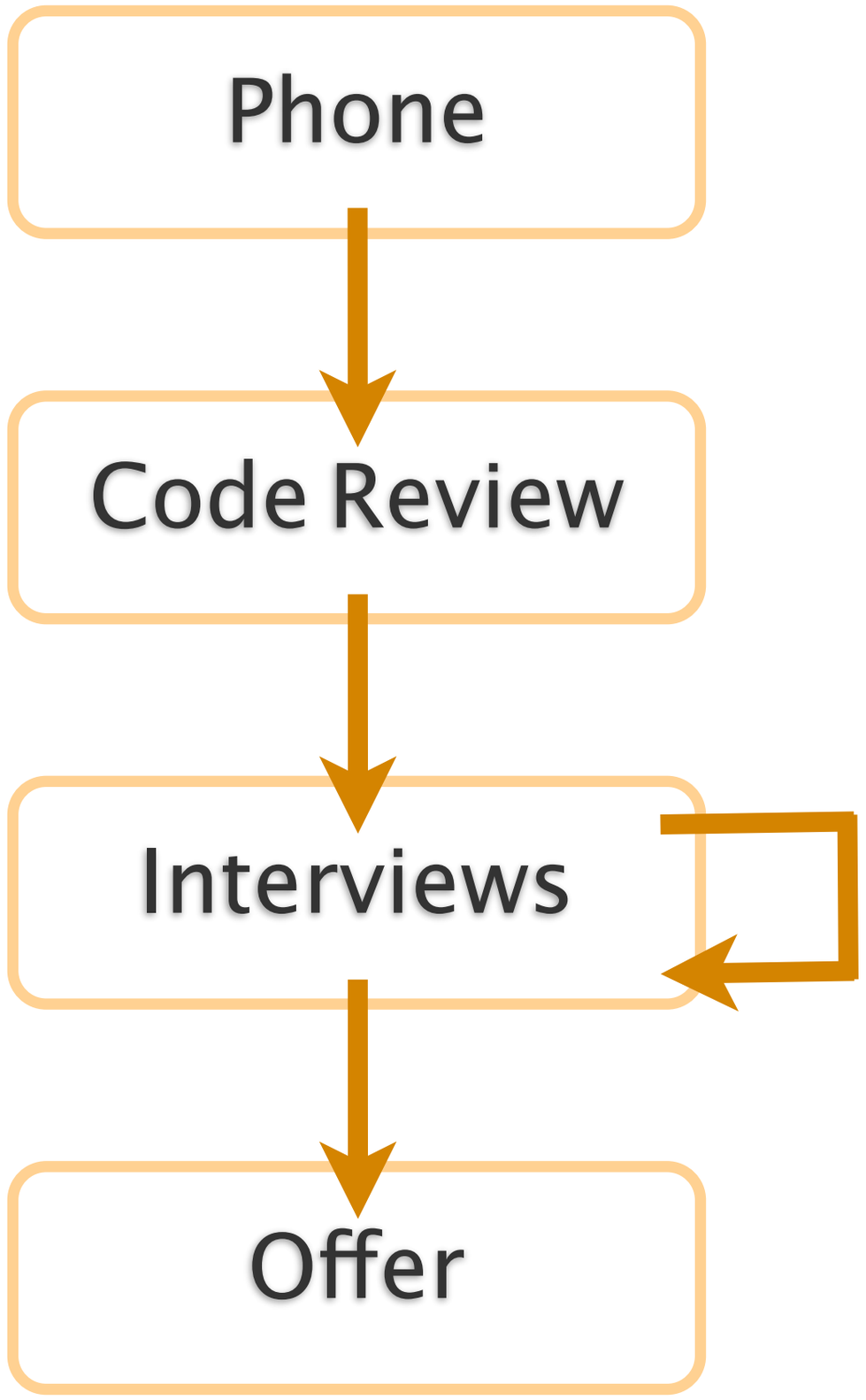


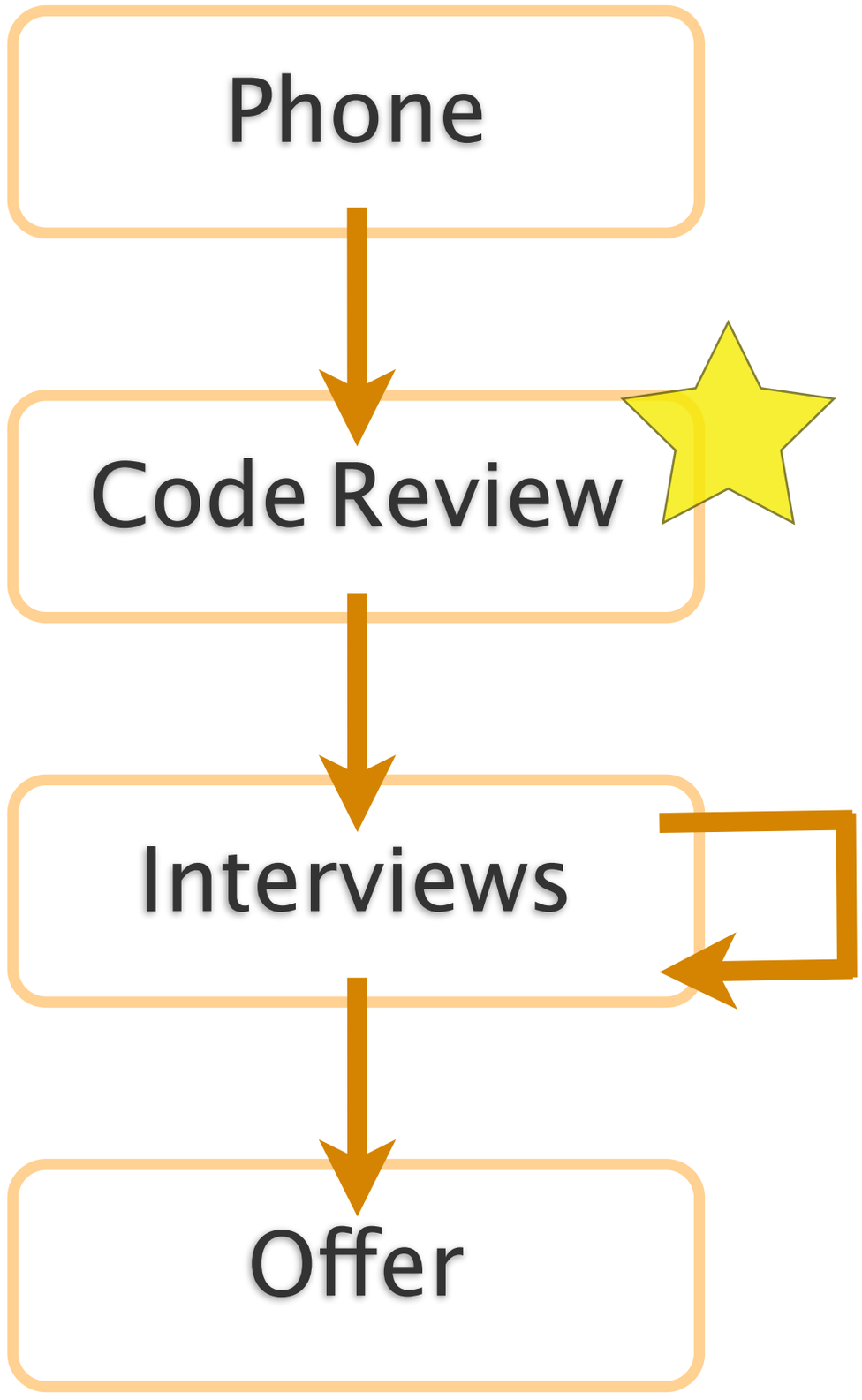
Two Door Cinema Club

hiring @ SoundCloud

GET
EXCITED
AND
MAKE
THINGS

http://bit.ly/15DToNK


Phone
Code Review
Interviews
Offer


Phone
Code Review
Interviews
Offer

challenge until late 2012

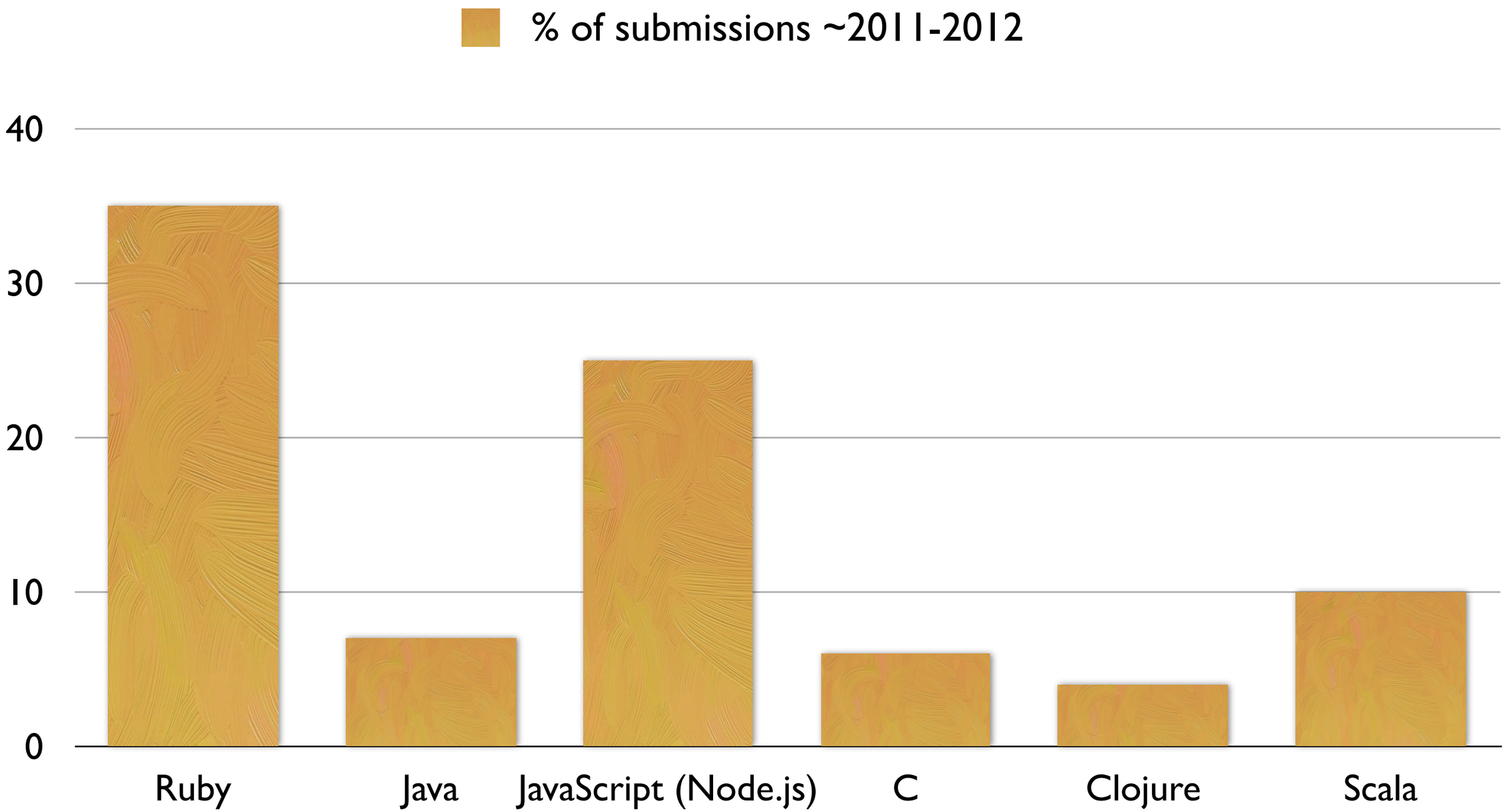
Backend Developer Challenge

Build a system that will accept a multipart form upload while displaying a percentage progress.

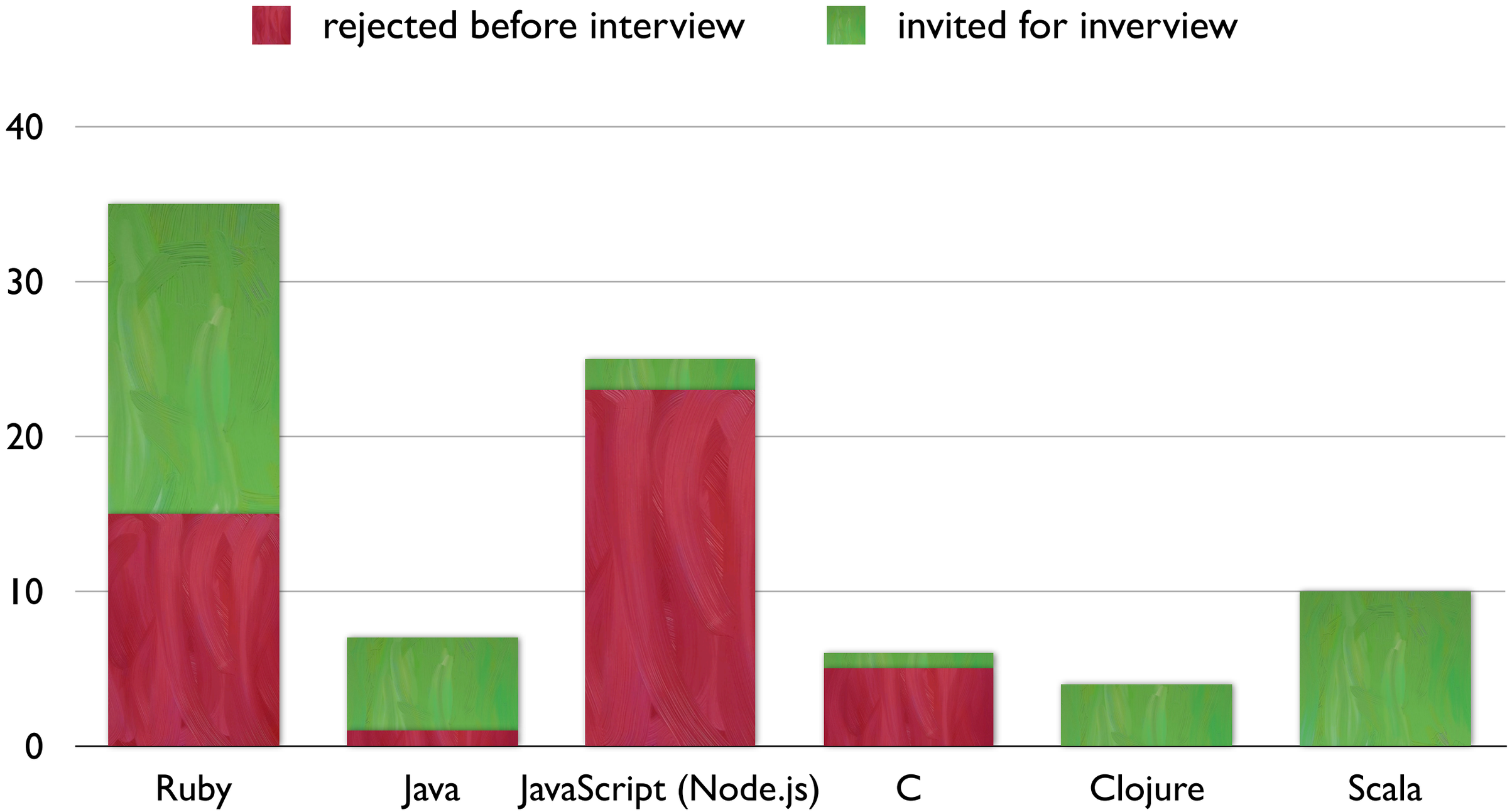
Specification

When a user picks a file from their computer, the upload automatically begins. While uploading, the percentage complete is visible on the page. It should update at least every 2 seconds.

you can choose any language.



you can choose any language.



new challenge

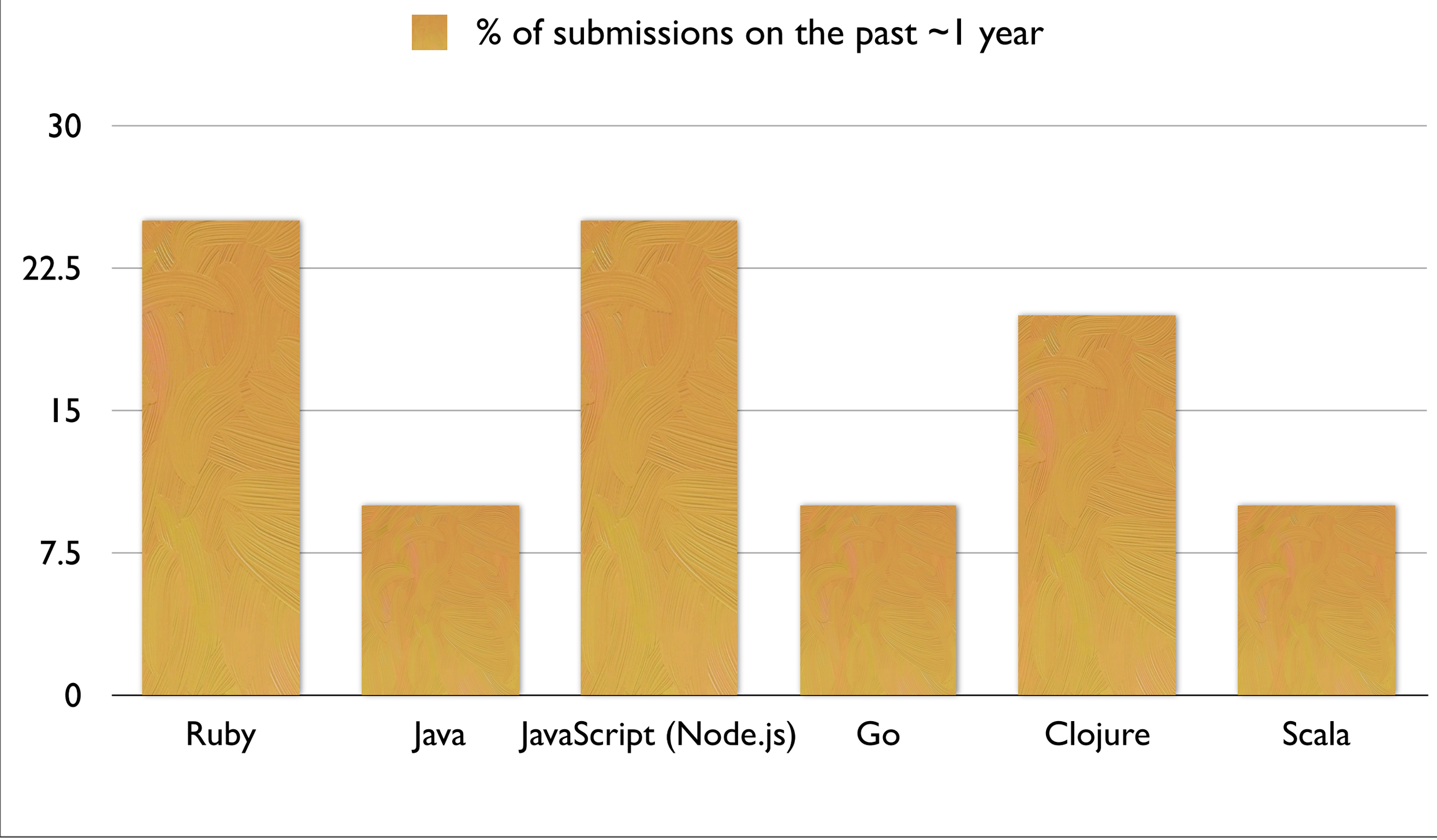
The Challenge

The challenge proposed here is to build a system which acts as a socket server, reading events from an *event source* and forwarding them when appropriate to *user clients*.

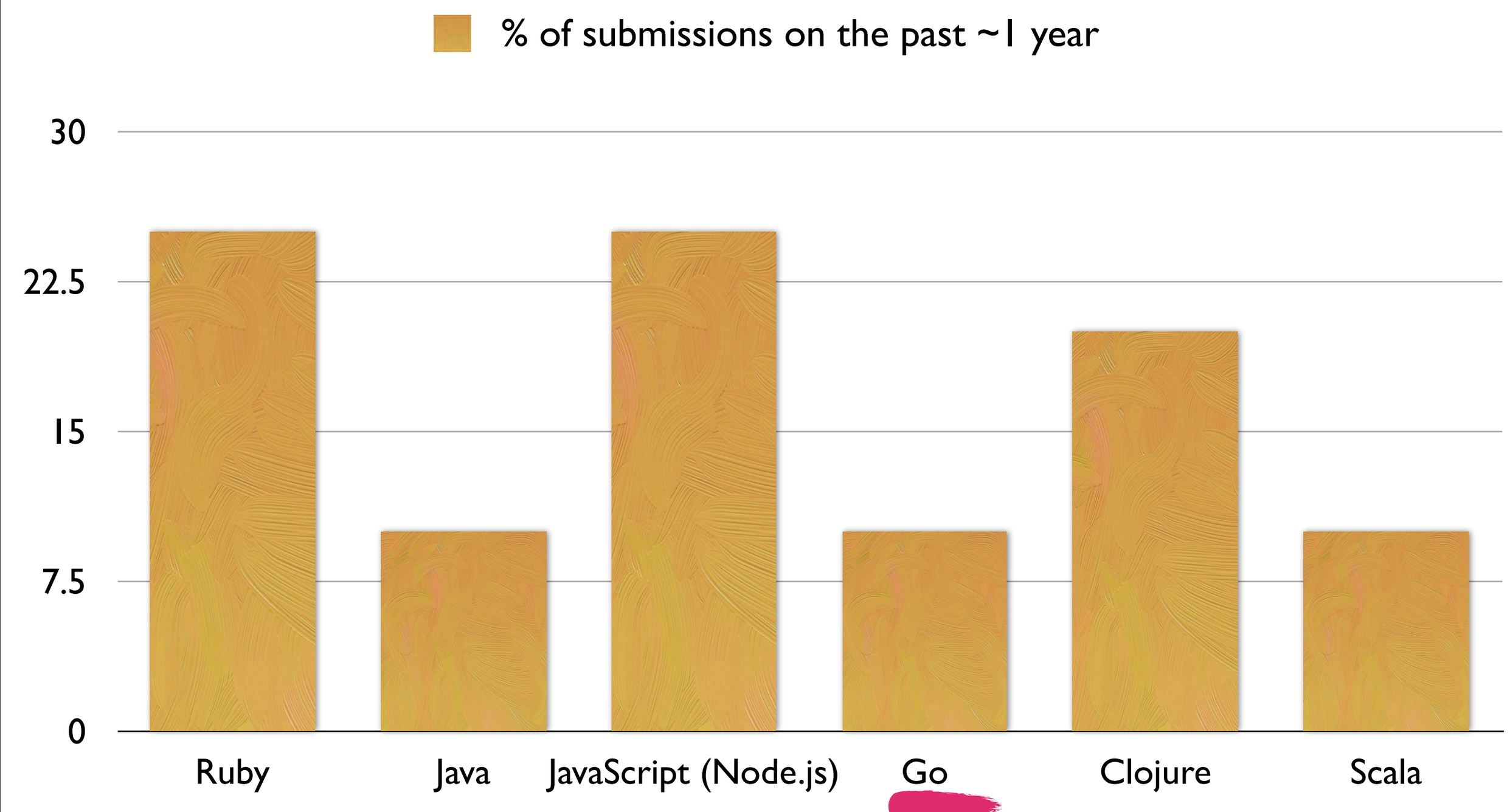
Clients will connect through TCP and use the simple protocol described in a section below. There will be two types of clients connecting to your server:

- **One** *event source*: It will send you a stream of events which may or may not require clients to be notified
- **Many** *user clients*: Each one representing a specific user, these wait for notifications for events which would be relevant to the user they represent

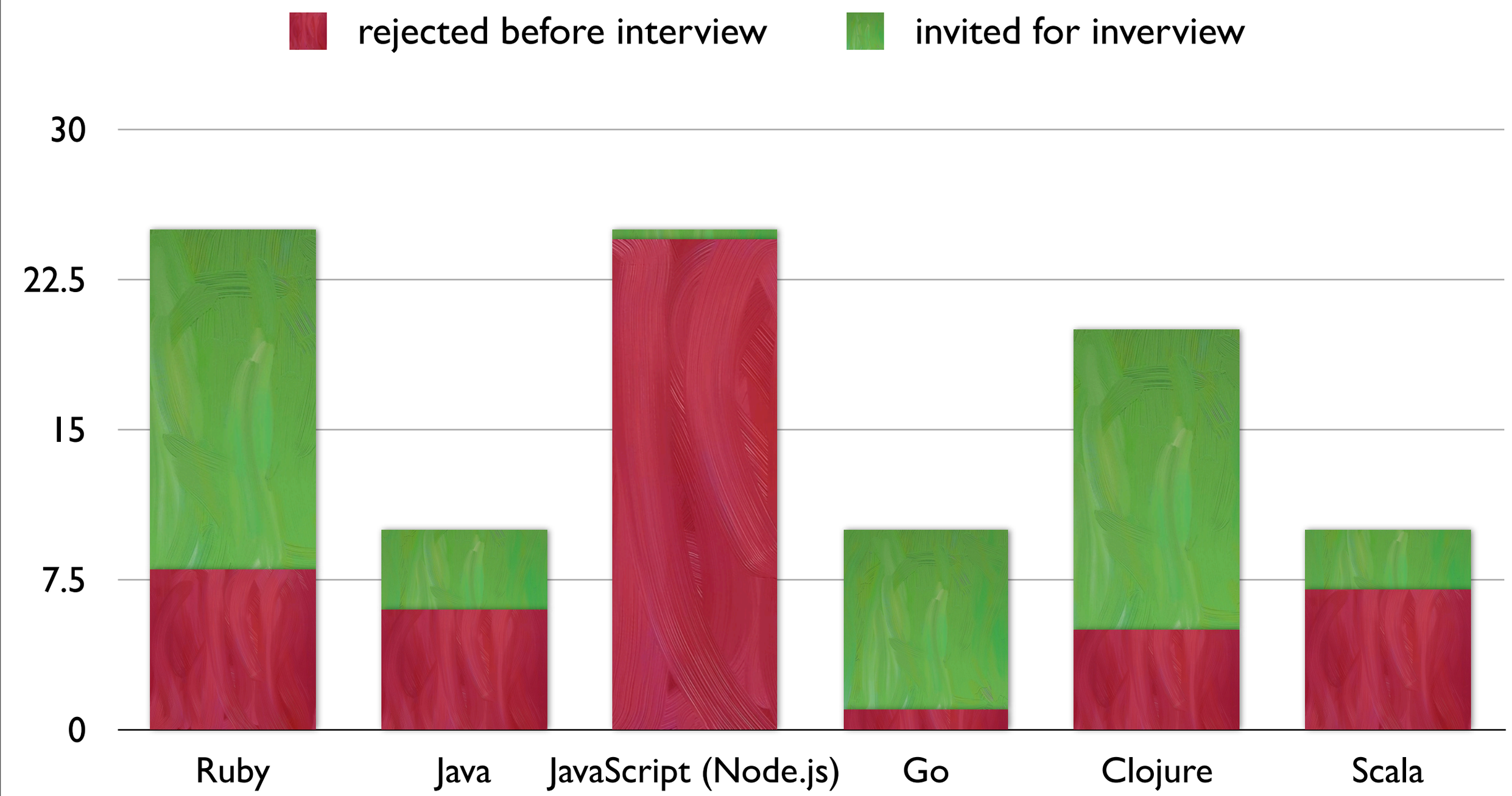
you can choose any language.



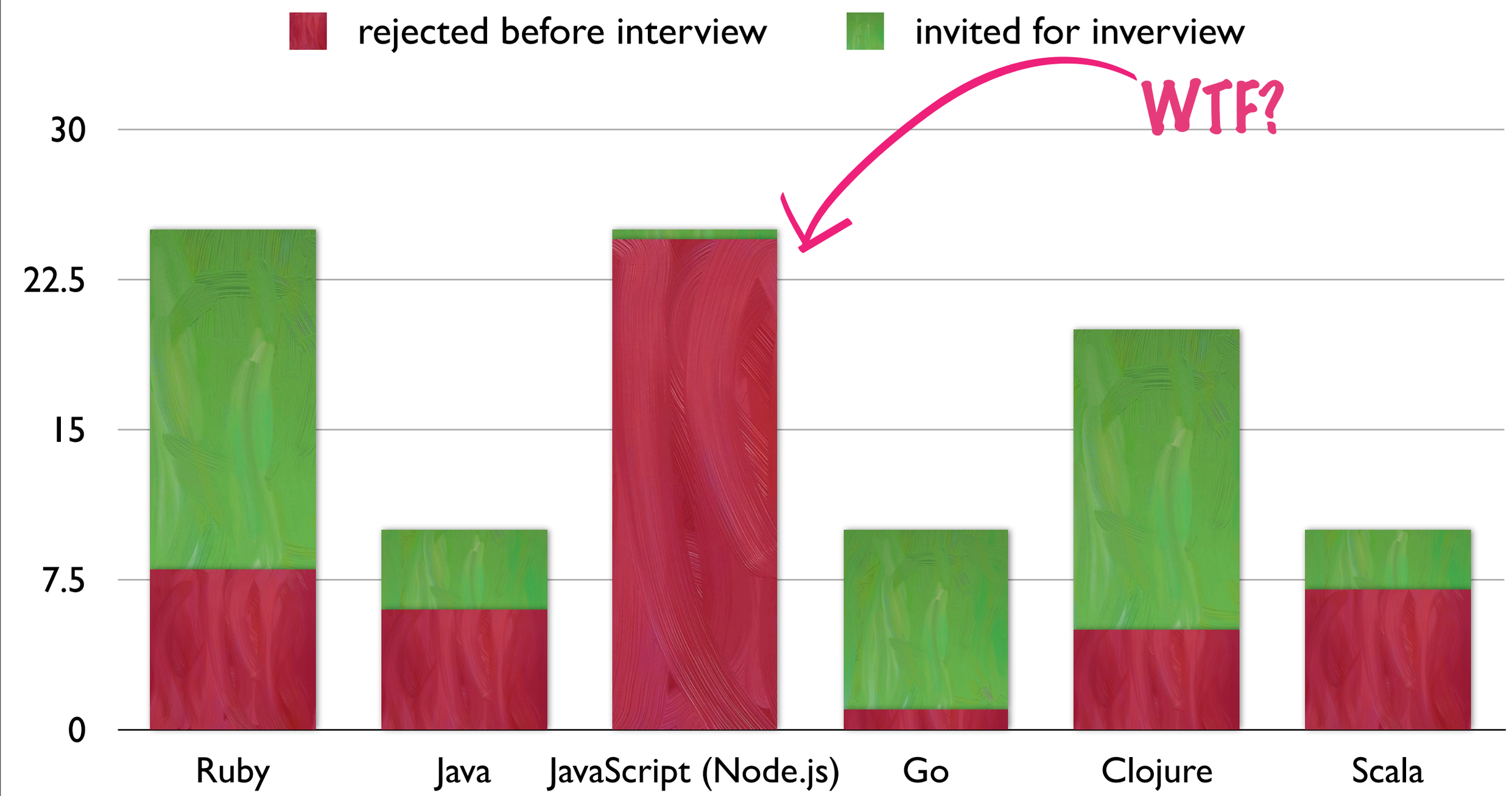
you can choose any language.



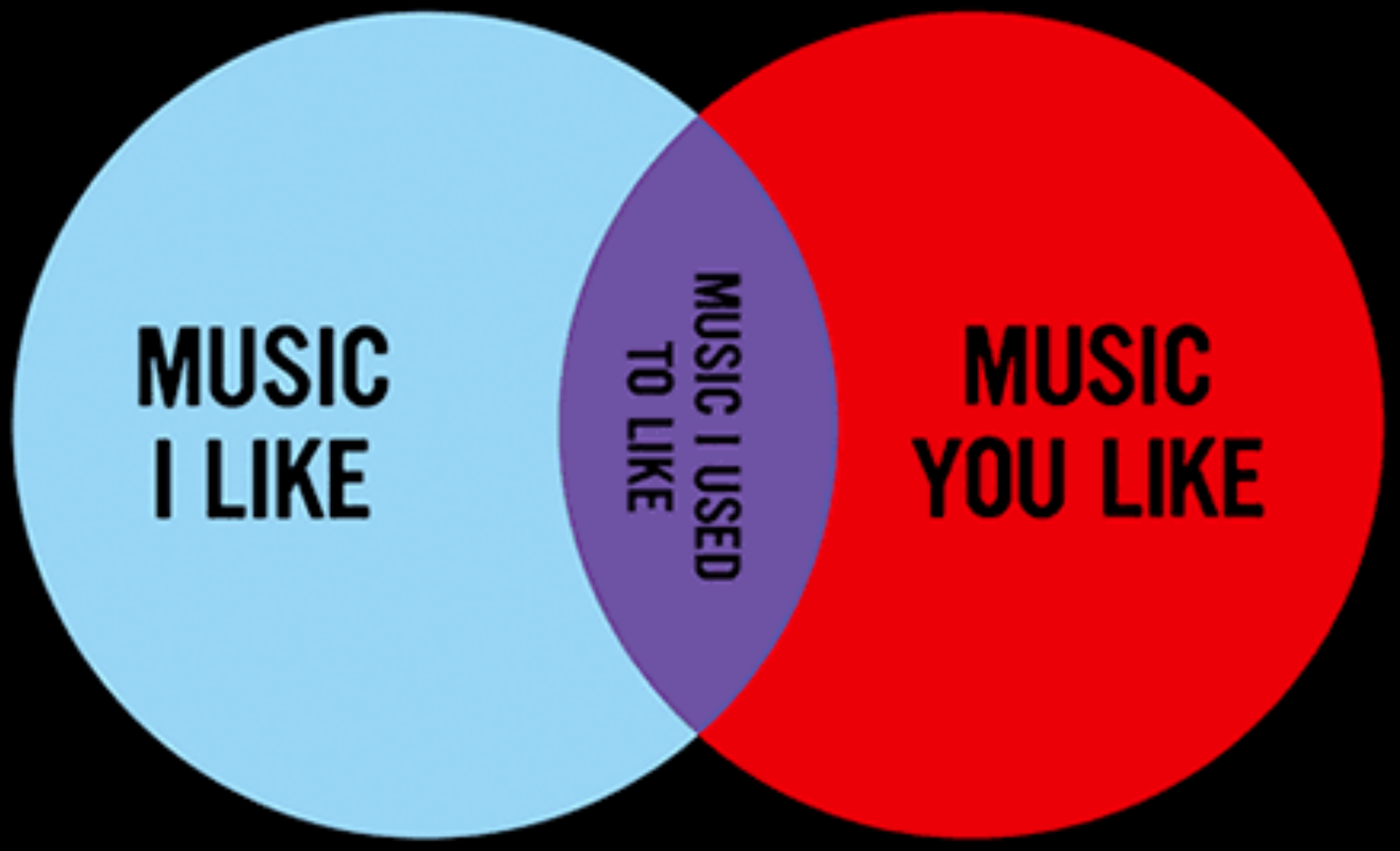
you can choose any language.



you can choose any language.



don’t get me wrong



we are all hipsters

http://bit.ly/J1aLNn

More 1–24 of 24

an, Sean, Peter (4)	Inbox	dev-discuss	[dev-discuss] Queuing: Bloat, Delays, and Active Queue Management - Nice, that paper has been q	16:47
y Conroy	Inbox	dev-discuss	[dev-discuss] Recursive Drawing - http://recursivedrawing.com/ Marcus just sent this to the design t	11:40
, Kim, Sebastian, me (6)		dev-discuss	[dev-discuss] the gospel of simplicity - hit harder: https://twitter.com/#!/PLT_Borat/status/197679102105362	9 May
omás (3)	Inbox	dev-discuss	Interview: Philip Wadler on Functional Programming - Thanks for the links! On Mon, May 7, 2012 at	7 May
		dev-discuss	Herlin Haskell User Group on meetup - Sounds interesting: http://www.meetup.com/berlinhug	4 May
Sebastian (2)	Inbox	dev-discuss	[dev-discuss] Fwd: interesting article about probablistic data structures for data mini... - On Thu, Ma	3 May
omás (2)	Inbox	dev-discuss	Generating Compiler Optimizations from Proofs - I found this useful as overview of category theory a	3 May
		dev-discuss	Rich Hickey keynote @ RailsConf - Keynote: Simplicity Matters Rich Hickey Rich Hickey, the author of Clo	2 May
, me, Matthias (15)	Inbox	dev-discuss	[dev-discuss] ruby + structural types - I know, I'm just saying it doesn't fit very well with the nature c	2 May
stian Ohm	Inbox	dev-discuss	[dev-discuss] Extending Ruby with Ruby - Here's a railsconf presentation, demonstrating how to met	30 Apr
mélie (2)	Inbox	dev-discuss	Mendeley's "personalised research advisor" using mahout - If we need to know more about the Musi	27 Apr
stian Ohm	Inbox	dev-discuss	[dev-discuss] John Carmack on Functional Programming - http://www.altdevblogaday.com/2012/04/2	27 Apr
ias Georgi		dev-discuss	[dev-discuss] mruby - a lightweight implementation of ruby - https://github.com/mruby/mruby mruby is the li	20 Apr
		dev-discuss	A proof that Unix utility "sed" is Turing complete - http://www.catonmat.net/blog/proof-that-sed-is-turing-com	20 Apr
		dev-discuss	ScalaDays VIdeos... - Are already being posted (<3 skillsmatter, wish we had it in Berlin): http://skillsmatter	18 Apr
stian Ohm		dev-discuss	[dev-discuss] Celluloid: Actor-based concurrent object framework for Ruby - Hey, wondering if anybody has	18 Apr
nder, Kim (2)		dev-discuss	[dev-discuss] Truth about scala - Inline image 1 On Fri, Apr 13, 2012 at 11:10 AM, Alexander Simmerl <alx(	13 Apr
		dev-discuss	Ebook: Data-Intensive Text Processing with MapReduce - http://lintool.github.com/MapReduceAlgorithms/M	12 Apr
		dev-discuss	Cute intro to Lambda calculus, Peano, Y combinator, and many other concepts using Ruby ... - bit.ly/HjPf14	6 Apr
		dev-discuss	Hamster - Efficient, Immutable, Thread-Safe Collection classes for Ruby - https://github.com/harukizaemon	30 Mar
stian Ohm		dev-discuss	[dev-discuss] Ruby 2.0: New GC and Enumerable::Lazy - Here's an article providing some insight into Ruby	24 Mar
		dev-discuss	Slides: The Typeclassopedia - http://typeclassopedia.bitbucket.org	16 Mar
		dev-discuss	Skillsmatter's Functional Programming exchange Videos Being published - Skillsmatter is already publishing	16 Mar

```
this.fs.stat(fileToServe, function(err, stats) {
  if (stats) {
    if(stats.isDirectory()) {
      if(settings['directory_listing'] === true && !self.path.existsSync(self.path.jo
        data = self.print_directory_listing({'parent' : fileToServe, 'files' : self.f
        content_type = 'text/html; charset=utf-8';
      }
      else {
        fileToServe = self.path.join(fileToServe, settings['directory_index']);
      }
    }
  }
  else {
    stats = false;
    try {
      stats = self.fs.statSync(fileToServe + '.xml');
    }
    catch(e) {
      console.log(e);
    }

  if(stats) {
    fileToServe += '.html';
  }
  else {
    fileToServe = '404.xml');
    status_code = 404;
  }
```

Not OO, not Functional...
Just (bad) Procedural code.

NOTHING IS ANY GOOD IF OTHER PEOPLE LIKE IT

http://bit.ly/JiECRp

we don’t need this complicated stuff.

this is not java, you know.

so how do we structure our app?

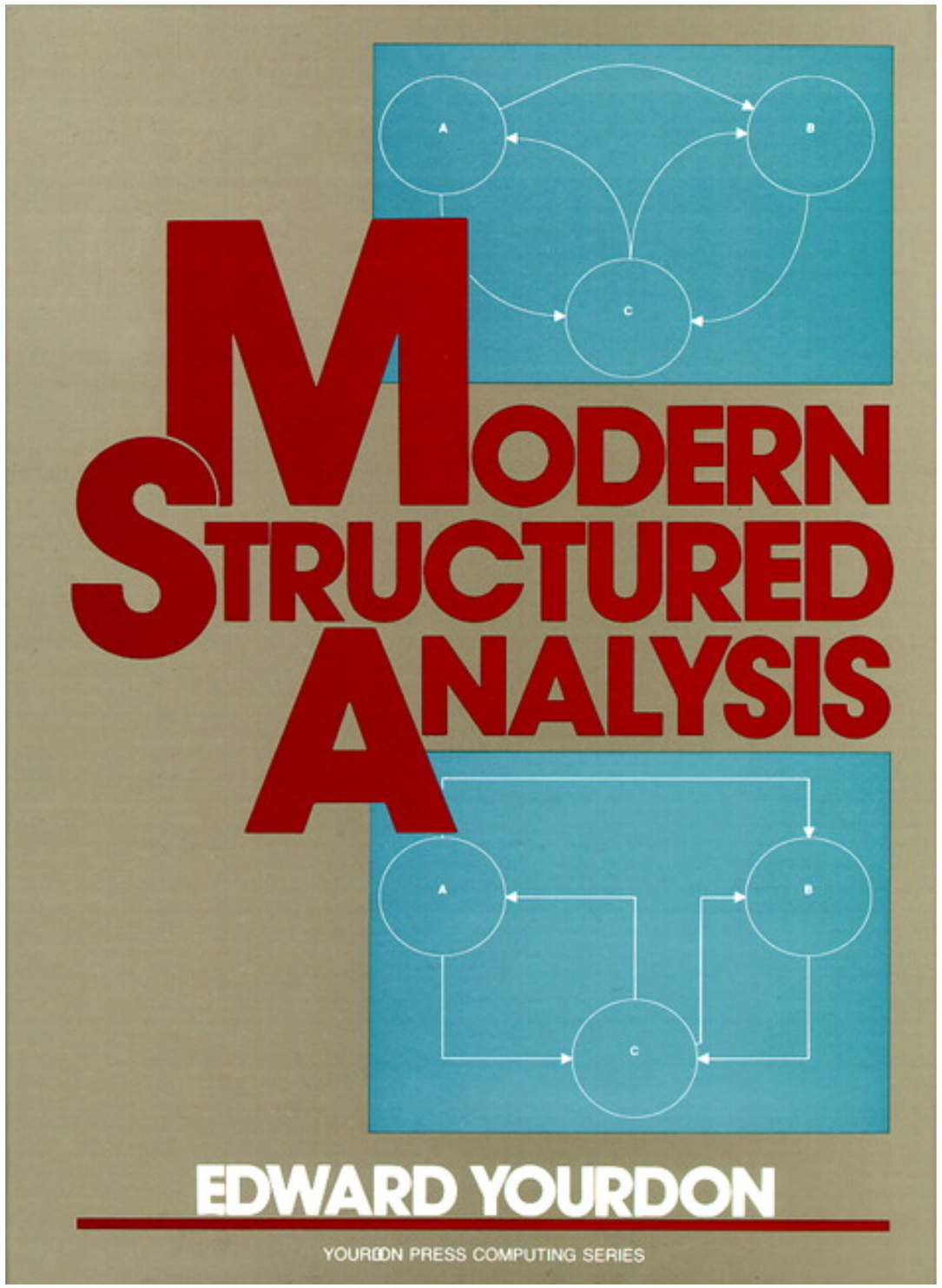
IT'S
COMPLICATED

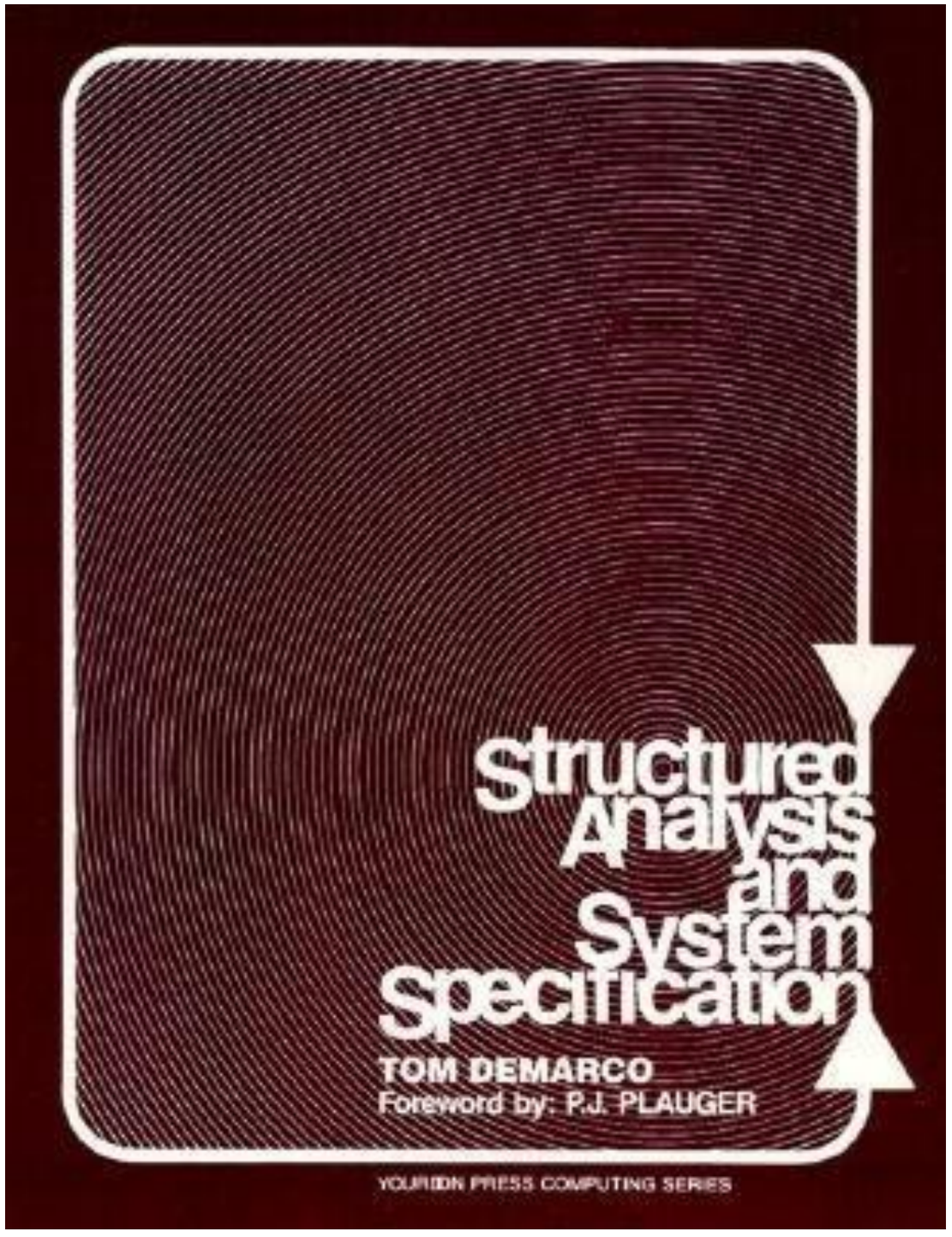
http://bit.ly/JiFSnq

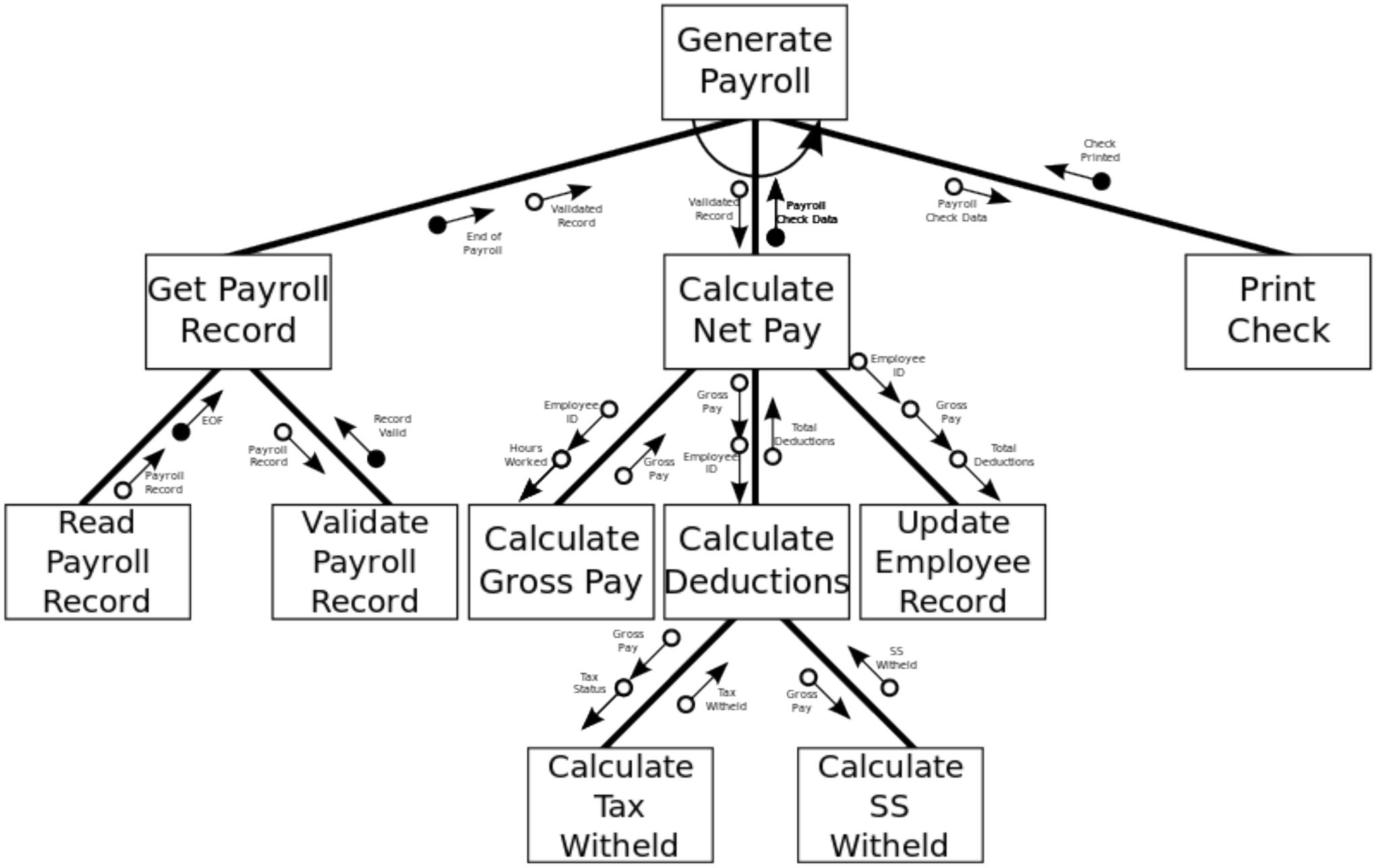
what to use?

(decent)

procedures?


A
B
C
MODERN
STRUCTURED
ANALYSIS
A
B
C
EDWARD YOURDON
YOURDON PRESS COMPUTING SERIES


Structured
Analysis
and
System
Specification
TOM DEMARCO
Foreword by: P.J. PLAUGER
YOURDON PRESS COMPUTING SERIES


Generate Payroll
End of Payroll
Validated Record
Validated Record
Payroll Check Data
Payroll Check Data
Check Printed
Get Payroll Record
Calculate Net Pay
Print Check
EOF
Payroll Record
Payroll Record
Record Valid
Employee ID
Hours Worked
Gross Pay
Employee ID
Gross Pay
Total Deductions
Employee ID
Gross Pay
Total Deductions
Read Payroll Record
Validate Payroll Record
Calculate Gross Pay
Calculate Deductions
Update Employee Record
Gross Pay
Tax Status
Tax Witheld
Gross Pay
SS Witheld
Calculate Tax Witheld
Calculate SS Witheld

what to use?

objects?

Grab File Edit Capture Window Help

Monsters Calling Home

https://soundcloud.com/runrivernorth/monsters-calling-home

Read Later | SC | APIs | DE | compsci | Reading | Viewportmarklet | Stanford Encycloped | linkpichos | motivate | Other Bookmarks

Stream | Explore | Search | Upload | Monsters Calling Home | More

Run River North

1 year

Monsters Calling Home

0.26

Write a comment ...

Repost | Add to set | Share

28,454 | 166 | 19 | 21

• Indie Rock • Gangster Folk Oriental • Monsters Calling Home • ollaboration • K • Korean • Korean American • Folk • arcade fire

Recorded by Stephen JW Lee. Mixed by Daniel Chae & Stephen JW Lee.
Jennifer.Daniel.Alex.Sally.John.Joe

21 comments View all

Daphney Chang at 3
Ur songs r life chang

Report copyright infringement

Run River North
1,145 | 7
Following

Related

unseenmusic
[Indie / Folk / Thoughtful] - "Wild...

SOUNDCLOUD

Nick Fisher

June 14th, 2012

Building The Next SoundCloud

> This article is also available in Serbo-Croatian: Pravljenje novog SoundCloud.

The front-end team at SoundCloud has been building upon our experiences with the HTML5 widget to make the recently-released Next SoundCloud beta as solid as possible. Part of any learning also includes sharing your experiences, so here we outline the front-end architecture of the new site.

Building a single-page application

what to use?

. . . functions?

▲ BrendanEich 295 days ago | link | parent

Dumb luck, as in winning the lottery? Not really. What did Ben Kenobi say to Han Solo about luck?

Subtle chains of cause and effect were at play among people involved, going back years to Silicon Graphics (Netscape drew from UIUC and SGI, plus montulli from Kansas, and jwz). Also going back through the living history of programming languages. SICP and some of the Sussman & Steele "Lambda the ..." papers made a big impression on me years before, although I did not understand their full meaning then.

Remember, I was recruited to "do Scheme", which felt like bait and switch in light of the Java deal brewing by the time I joined Netscape. My interest in languages such as Self informed a subversive agenda re: the dumbed down mission to make "Java's kid brother", to have objects without classes. Likewise with first-class functions, which were inspired by Scheme but quite different in JS, especially JS 1.0.

Apart from the "look like Java" mandate, and "object-based" as a talking point, I had little direction. Only a couple of top people at Netscape and Sun really grokked the benefit of a dynamic language for tying together components, but they were top people (marca, Rick Schell [VP Eng Netscape], Bill Joy).

Rather than dumb luck, I think a more meaningful interpretation is that I was a piece of an evolving system, exploring one particular path in a damn hurry. That system contains people playing crucial parts. Academic, business, and personal philosophical and friendship agendas all transmitted an analogue of genes: ideas and concrete inventions from functional programming and Smalltalk-related languages.

You might think "it's still luck, it could have been Forth, or TCL". Not likely. There were not years or even months to spare. I had hacked language implementations for fun since I was an undergrad, and for SGI's packet sniffing tools earlier my career. I was a C/Unix fanboy, I knew the C grammar by heart. Independent of me, the "Make it look like Java" order was not just lucky, it was congruent as a consequent, even predictable, given the rise of C in the '80s and C++ in the '90s, and the direct influence of C++ on Java.

My point is simple: the likelihood of any other language syntax than C (C++ -> Java, but really: C) was low. The likelihood of something without "objects" was also low. Netscape recruited me in part because I could hack quickly, and in part because I had some language implementation chops (not enough, in hindsight). I was "that guy", not in any brag-worthy sense, just the only person who was in the position to do the deed, with (barely) enough skills to pull it off.

Many hackers could have done a better job with more time, or perhaps a better job had they been in my shoes. Who knows? But no one at Netscape could have, and the opportunity was there and then.

The path dependence part is spot on. Netscape's business plan for 1.0 was getting out in six months or someone else would kill Mosaic and take over. The entire platform push in 1.1 (plugins) and 2 (frames, JS) was about getting on first. We knew Microsoft was coming, because Netscape had rejected a low-ball offer from them in late '94.

/be

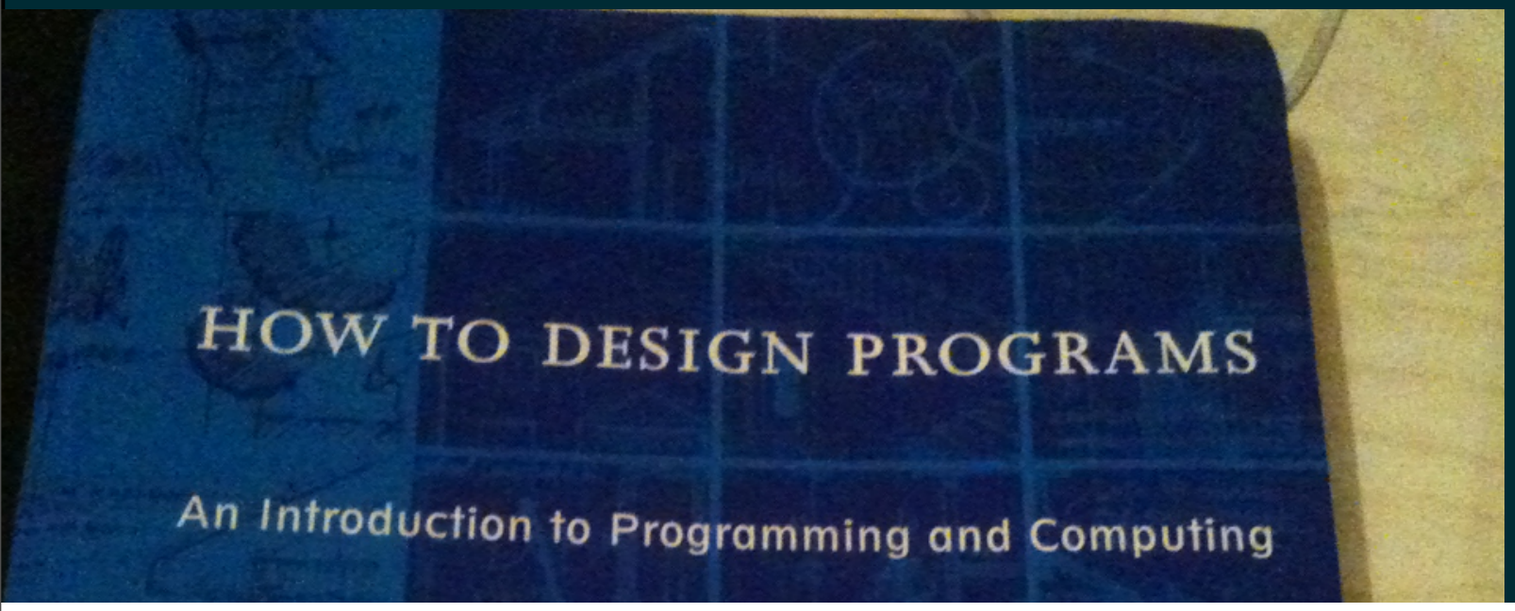
“SICP and some of the Sussman & Steele "Lambda the ..." papers made a big impression on me years before, although I did not understand their full meaning then.

Remember, I was recruited to "do Scheme" [...] My interest in languages such as Self informed a subversive agenda [...]. Likewise with first-class functions, which were inspired by Scheme [...]”

http://bit.ly/JFbZt8

PLAY OLD VIDEO GAMES

http://bit.ly/JiEQYM



That is, *f* stands for *x-adder5*, a function, which adds 5 to its argument.

Using this example, we can write *add*'s contract and a purpose statement:

```
;; add : number → (number → number)
;; to create a function that adds x to its input
(define (add x)
  (local ((define (x-adder y) (+ x y)))
    x-adder))
```

The most interesting property of *add* is that its result "remembers" the value of *x*. For example, every time we use *f*, it uses 5, the value of *x*, when *add*

```
(define (add x)
  (local ((define (x-adder y) (+ x y)))
    x-adder))
```

```
(define (add x)
  (local ((define (x-adder y) (+ x y)))
    x-adder))
```

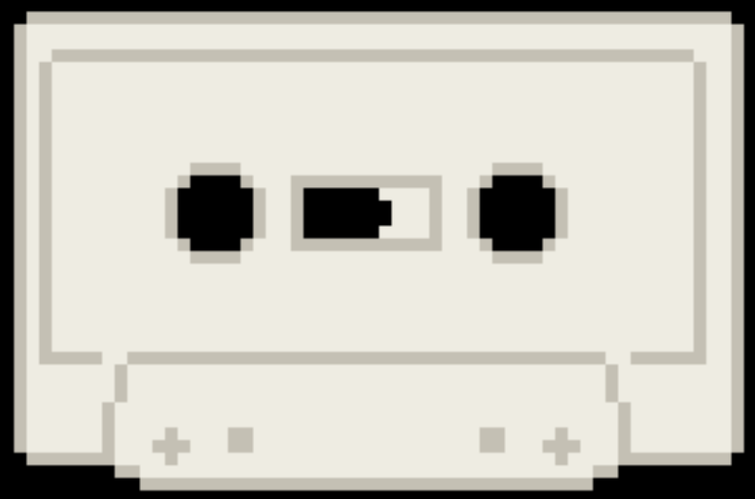
```
Welcome to Racket v5.2.1.
> > (define adder (add 7))
> adder
#<procedure:x-adder>
> (adder 10)
17
>
```

```
function add(x) {
  return function (y) {
    return y + x;
  };
}

var adder = add(7);
console.log(adder);
console.log(adder(10));
```

```
function add(x) {
  return function (y) {
    return y + x;
  };
}

var adder = add(7);
console.log(adder);
console.log(adder(10));
```

```
> var adder = add(7);
  undefined
> console.log(adder);
  function (y) {
      return y + x;
    }
< undefined
> console.log(adder(10));
  17
< undefined
>
```

going old school



http://bit.ly/JiF7ef

e.g. passing functions around

we see a lot of this

```
function addUser(user) {
  try {
    var db = connectToDatabase(credentials());
    execute(db, asSqlInsert(user));
  } catch (e) {
    handleDatabaseError(e);
  }
}
```

```
function addTrack(track) {
  try {
    var db = connectToDatabase(credentials());
    execute(db, asSqlInsert(track));
  } catch (e) {
    handleDatabaseError(e);}
}
```

1st, we extract what is common

```
function executeWithConnection(functionToExecute, argument){
  try {
    var db = connectToDatabase(credentials());
    functionToExecute(db, argument);
  } catch (e) {
    handleDatabaseError(e);
  }
}
```

then we refactor our functions

```
function addUser(db, user) {
  execute(db, asSqlInsert(user));
}


function addTrack(db, track) {
  execute(db, asSqlInsert(track));
}

function executeWithConnection(functionToExecute, argument){
  try {
    var db = connectToDatabase(credentials());
    functionToExecute(db, argument);
  } catch (e) {
    handleDatabaseError(e);
  }
}
```

and we send them as arguments

```
function addUser(db, user) {
  execute(db, asSqlInsert(user));
}


function addTrack(db, track) {
  execute(db, asSqlInsert(track));
}

function executeWithConnection(functionToExecute, argument){
  try {
    var db = connectToDatabase(credentials());
    functionToExecute(db, argument);
  } catch (e) {
    handleDatabaseError(e);
  }
}

executeWithConnection(addUser, {'name':'Phil'});
executeWithConnection(addTrack, {'title':'The JavaScript Blues'});
```

e.g. closures not only for objects

we see a lot of this

```
function writeComment(count, author, text) {
  if(count > 3){
    throw new Error(author + " executed too many actions!!");
  }
  saveComment(author, text);
  return count + 1;
}
```

```
var counter = 0;
counter = writeComment(counter, 'pcalcado', "Check my stuff");
counter = writeComment(counter, 'pcalcado', "Check my stuff");
counter = writeComment(counter, 'pcalcado', "Check my stuff");
counter = writeComment(counter, 'pcalcado', "Check my stuff");
counter = writeComment(counter, 'pcalcado', "Check my stuff");
// -> Error: pcalcado executed too many actions!!
```

we can keep writeComment as it is

```
function writeComment(count, author, text) {
  if(count > 3){
    throw new Error(author + " executed too many actions!!");
  }
  saveComment(author, text);
  return count + 1;
}
```

and we add a function with two closures

```
function writeComment(count, author, text) {
  if(count > 3){
    throw new Error(author + " executed too many actions!!");
  }
  saveComment(author, text);
  return count + 1;
}

function makeWriteCommentFunction(author){
  var counter = 0;
  return function (text) {
    counter = writeComment(counter, author, text);
  }
}
```

and we add a function with two closures

```
function writeComment(count, author, text) {
  if(count > 3){
    throw new Error(author + " executed too many actions!!");
  }
  saveComment(author, text);
  return count + 1;
}

function makeWriteCommentFunction(author){
  var counter = 0;
  return function (text) {
    counter = writeComment(counter, author, text);
  }
}
```

immutable

and we add a function with two closures

```
function writeComment(count, author, text) {
  if(count > 3){
    throw new Error(author + " executed too many actions!!");
  }
  saveComment(author, text);
  return count + 1;
}

function makeWriteCommentFunction(author){
  var counter = 0;
  return function (text) {
    counter = writeComment(counter, author, text);
  }
}
```

mutable

immutable

now we don't need to pass in the kitchen sink

```
function writeComment(count, author, text) {
  if(count > 3){
    throw new Error(author + " executed too many actions!!");
  }
  saveComment(author, text);
  return count + 1;
}

function makeWriteCommentFunction(author){
  var counter = 0;
  return function (text) {
    counter = writeComment(counter, author, text);
  }
}

var currentUserWritesComment = makeWriteCommentFunction('pcalcado');
currentUserWritesComment("Check my stuff");
currentUserWritesComment("Check my stuff");
currentUserWritesComment("Check my stuff");
currentUserWritesComment("Check my stuff");
currentUserWritesComment("Check my stuff");
// -> Error: pcalcado executed too many actions!!
```

e.g. functions all the way down

we see a lot of this

```
function deleteUser(currentUser, userToDelete){
  if(currentUser == userToDelete || currentUser.admin) {
    deleteRecord(userToDelete);
  } else {
    throw new Error(currentUser.login + " trying to delete "+ userToDelete.login);
  }
}

function activateUser(currentUser, userToActivate){
  if(currentUser == userToActivate || currentUser.admin) {
    activate(userToActivate);
  } else {
    throw new Error(currentUser.login + " trying to activate "+ userToActivate.login)
  }
}
deleteUser(admin, pcalcado);
deleteUser(pcalcado, pcalcado);
deleteUser(tiga, pcalcado);
// Error: tiga trying to delete pcalcado

activateUser(admin, pcalcado);
activateUser(pcalcado, pcalcado);
activateUser(tiga, pcalcado);
// Error: tiga trying to activate pcalcado
```

first we extract common protocol

```
function makeAuthorisationCheckingFunction(functionToExecute){
  return function(currentUser, userToModify) {
    if(currentUser == userToModify|| currentUser.admin) {
      functionToExecute(userToModify);
    } else {
      throw new Error(currentUser.login + " trying to "+ functionToExecute.name  + u
    }
  }
}
```

then we clean up our functions

```
function deleteUser(userToDelete){
    deleteRecord(userToDelete);
}

function activateUser(userToActivate){
    activate(userToActivate);
}

function makeAuthorisationCheckingFunction(functionToExecute){
  return function(currentUser, userToModify) {
    if(currentUser == userToModify|| currentUser.admin) {
      functionToExecute(userToModify);
    } else {
      throw new Error(currentUser.login + " trying to "+ functionToExecute.name  + u
    }
  }
}
```

then we use them.

```
function deleteUser(userToDelete){
    deleteRecord(userToDelete);
}

function activateUser(userToActivate){
    activate(userToActivate);
}

function makeAuthorisationCheckingFunction(functionToExecute){
  return function(currentUser, userToModify) {
    if(currentUser == userToModify|| currentUser.admin) {
      functionToExecute(userToModify);
    } else {
      throw new Error(currentUser.login + " trying to "+ functionToExecute.name  + u
    }
  }
}
var safeActivateUser = makeAuthorisationCheckingFunction(activateUser);
var safeDeleteUser = makeAuthorisationCheckingFunction(deleteUser);

safeDeleteUser(admin, pcalcado);
safeDeleteUser(pcalcado, pcalcado);
safeDeleteUser(tiga, pcalcado);
// tiga trying to execute deleteUser on pcalcado
```

then we use them.

redundant?

```
function deleteUser(userToDelete){
    deleteRecord(userToDelete);
}

function activateUser(userToActivate){
    activate(userToActivate);
}

function makeAuthorisationCheckingFunction(functionToExecute){
  return function(currentUser, userToModify) {
    if(currentUser == userToModify|| currentUser.admin) {
      functionToExecute(userToModify);
    } else {
      throw new Error(currentUser.login + " trying to "+ functionToExecute.name  + u
    }
  }
}
var safeActivateUser = makeAuthorisationCheckingFunction(activateUser);
var safeDeleteUser = makeAuthorisationCheckingFunction(deleteUser);

safeDeleteUser(admin, pcalcado);
safeDeleteUser(pcalcado, pcalcado);
safeDeleteUser(tiga, pcalcado);
// tiga trying to execute deleteUser on pcalcado
```

IT'S FUN TO USE LEARNING FOR EVIL!

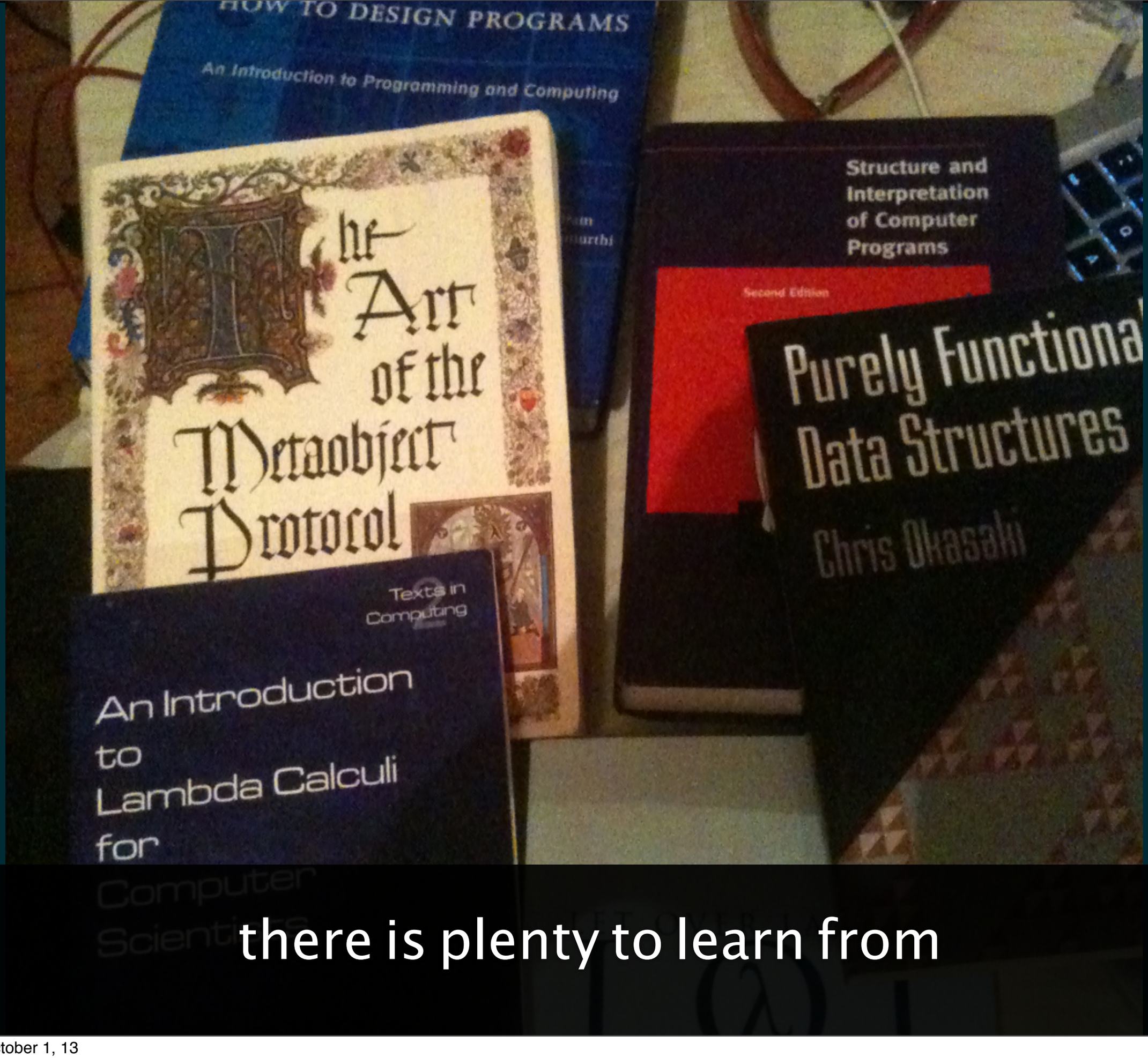
http://bit.ly/JiETnd

our “framework”:

1 – Extract protocol in “combinators”

2 – Keep mutable state in closures

no scary monads required



there is plenty to learn from

Introducing Functional Programming with Underscore.js

Functional JavaScript

Michael Fogus
Forewords by Steve Vinoski & Jeremy Ashkenas

O'REILLY®

in the javascript community too

phil calçado

http://philcalcado.com
@pcalcado

www.soundcloud.com

How to Design Programs - http://bit.ly/K0BfrL

Structure and Interpretation of Computer Programs - http://bit.ly/K0BjYm

The Art of the Metaobject Protocol
http://amzn.to/K0BqU1

Purely Functional Data Structures
http://amzn.to/JFn4KG

Let Over Lambda - http://amzn.to/IMMkNO

An Introduction to Lambda Calculi for Computer Scientists - http://amzn.to/IX8d1B

All drawings are available as t-shirts from the awesome Diesel Sweeties - http://dieselsweeties.com/