Hello ICON!
Hello RailsConf!
Hi, I’m @tenderlove
Le Git
Le Gîte Hub
Ruby Core Team
I have stickers!
Twitter

Hacker News Sucks!
HackersNews
"Why Slack is inappropriate for open source communications"
"We are, temporarily, in a kind of **dark ages** of end-user open source software."
I like testing

Oops, I died of plague

This is a rat
Electron is flash for the desktop
"slack often idles at 5% CPU usage"
CPU
"Buy a bigger CPU, ya dummy"
Alan Turing
HN on Language Features
"No generics and no sum types? Those are no longer groundbreaking, they're the *bare minimum.*"

*Emphasis not mine*
No generics means your language is a failure
Insurance companies only cover Generics
Sorry, your language has been denied
#supportlocal
Local Coffee!
Local Tea!
Trolling DHH: An Annual Tradition
Phoenix Framework
Faith Based Programmer
"Pleaseworkpleaseworkpleaseworkpleaseworkpleaseworkpleaseworkpleaseworkpleaseworkpleasework"

The Programmer’s Prayer
24 bit audio
10+ Years of Rails
It was the year 2003
What should I do with all this free time?
"We’ve tried to charge your card hundreds of times"
Technical Debt
Developers want to pay off technical debt
Managers do not want to pay off technical debt
(if they can help it)
Tech Debt

Meta Gem

Gem

Gem

Gem

Gem

Gem
Tech Debt

"Bond"

Gem

Gem

Gem

Gem

"Tranche"
Tech Debt

- Rails
  - Gem
  - Gem
  - Gem
  - Webpack
  - Gem
Blood of young people
Concatenating a bunch of strings
Not a Joke
5
NEW SUBWAY EVERY DAY VALUE MENU

$5 EACH FOOTLONGS

CHOOSE FROM 8 GREAT SUBS
5.1
Rails 5.1 is out!
Thanks @rafaelfranca!!!
I don’t have time for this

Things I don’t have time for but would like to do
Non-committal
Humorous use of common phrase
"No Pressure"
ALL OF THE PRESSURE
Searls - Briggs
STRESS
Fear & Self Doubt
5 things I don’t have time to do

but you should do them for me
What if I don’t have time to present 5 things??
What if I want to talk about things I *do* have time for?
Some things I don’t have time to do
but you should do them for me

Maybe like 3 things?
Probably Important

Definitely Not Important
HTTP/2 + Rack

A thing that I do have time for
What is HTTP/2?
1 better than HTTP/1
0.9 better than HTTP/1.1
HTTP/2 (really)

- Binary protocol
- Multiplexes (multiple requests on the same socket)
- Secure (only works over SSL*)
- Push responses
Push Response Flow
Save Time by Pushing "Needed" Resources
H2 Problems
H2 is still *stateless*
HTTP 2

Time

Client

Server

index.html

jquery.js

gorby.css
Double Assets

- Client can cancel (requires round trip)
- Resend assets (extra bandwidth)
Solution: H2O + SALT

Get it, "solution"????
Solution: H2O + Cookies
What is H2O?

https://h2o.example.net
Client

HTTP/2

Proxy (h2o)

HTTP/1.1

Server (Unicorn)
H2 Pushes with HTTP/1.1
Link Header

200 OK
Content-Length: 1024
Content-Type: text/html
Link: </choochoo.css>; rel=preload
Link: </gorby.js>; rel=preload

response body
HTTP/2

Client

Proxy (h2o)

Server (Unicorn)

200 OK
Content-Length: 1024
Content-Type: text/html
Link: </choochoo.css>; rel=preload
Link: </gorby.js>; rel=preload

response body
No state tracking in the app
Link Header Problem

200 OK
Content-Length: 1024
Content-Type: text/html
Link: </choochoo.css>; rel=preload
Link: </gorby.js>; rel=preload

response body
103 Early Hints
103 Early Response

103 Early Hints
Link: </choochoo.css>; rel=preload
Link: </gorby.js>; rel=preload

200 OK
Content-Length: 1024
Content-Type: text/html

response body
Rack Support

Rack 3.0
New ENV hash key! ("yay")

```javascript
env['rack.push'].call(103, [
['Link', '/gorby.css'], []
], [])
```
Backwards Compatibility
Graceful Upgrade
Rack 3.0
This is a talk about code smells
Data I acquired (crap data)
Ruby VM Tricks
How a VM works
VM

Program

(push, 4)
(push, 3)
(add)

Stack

7
3
Implementation

instructions = [
    [:push, 3],
    [:push, 7],
    [:add],
]

pc = 0
loop do
    instruction = instructions[pc]
    pc += 1
    send function_table[instruction.first]
end
Generate the VM
Loop Elimination

instructions = [[:push, 3], [:push, 7], [:add], ]

def push_op pc, val, instructions
    # code to push on the stack
    # ....
    next_instruction = instructions[pc]
    send(function_table[next_instruction.first]
        pc + 1,
        next_instruction,
        instructions)
end

def add pc, val, instructions
    # code to add
    # ....
    next_instruction = instructions[pc]
    send(function_table[next_instruction.first]
        pc + 1,
        next_instruction,
        instructions)
end

# start of VM
pc = 0
instruction = instructions[pc]
p += 1
send function_table[instruction.first]
Threaded VM
Function call elimination
instructions = [ [':push', 3], [':push', 7], [':add'], ]

function_table = {
    :push => :push_op,
    :add  => :add_op,
}

def push_op pc, val, instructions
    # code to push on the stack
    # ....

    next_instruction = instructions[pc]
    send(function_table[next_instruction.first]
         pc + 1,
         next_instruction,
         instructions)
eend
instructions = [ [:push, 3], [:push, 7], [:add], ]

function_table = {
    :push => ADDRESS_OF(:push_op),
    :add  => ADDRESS_OF(:add_op),
}

label: :add_op
    # code to add
    # ...
    next_instruction = instructions[pc]
    val               = next_instruction.drop(1)
    address           = function_table[next_instruction.first]
    pc               += 1
    GOTO(address)

label: :push_op
    # code to push on the stack
    # ....
    next_instruction = instructions[pc]
    val               = next_instruction.drop(1)
    address           = function_table[next_instruction.first]
    pc               += 1
    GOTO(address)
Lookup Table Elimination
Addresses in Byte Code

instructions = [
    [ADDRESS_OF(:push_op), 3],
    [ADDRESS_OF(:push_op), 7],
    [ADDRESS_OF(:add)],
]

label: :add_op
    # code to add
    # ...
    next_instruction = instructions[pc]
    val              = next_instruction.drop(1)
    address          = next_instruction.first
    pc               += 1
    GOTO(address)

label: :push_op
    # code to push on the stack
    # ....

    next_instruction = instructions[pc]
    val              = next_instruction.drop(1)
    address          = next_instruction.first
    pc               += 1
    GOTO(address)
Direct Threaded VM
Addresses are just integers
MRI’s VM is Direct Threaded
Create custom instructions built from \textit{machine code} at runtime.
Extremely Simple Code

```ruby
require 'lolvm'

class MyRuby
  attr_reader :rb_path2class, :rb_define_method, :mod

  def initialize ctx
    @mod = ctx.create_module 'ruby'
    @ctx = ctx
    init_functions
  end

  private

  def init_functions
    @rb_path2class = make_func('rb_path2class', @ctx.int8.pointer, [@ctx.int8.pointer])
    @rb_define_method = make_func('rb_define_method', @ctx.int8.pointer, [@ctx.int8.pointer, @ctx.int8.pointer, @ctx.int8.pointer, @ctx.int8.pointer, @ctx.int32])
  end

  def make_func name, ret, args
    func_type = LOLVM::Types::Function.new(ret, args, false)
    @mod.add_function name, func_type
  end
end

context = LOLVM::Context.instance
builder = context.create_builder
ruby = MyRuby.new context
mod = context.create_module 'anon mod'
func_type = LOLVM::Types::Function.new(context.double, [], false)

testing = mod.add_function "testing", LOLVM::Types::Function.new(context.int64, [context.int8.pointer], false)
testing.param(0).name = "foo"
bb = context.append_basic_block testing, "entry"
x = mod.add_global context.int64, "rb_CObject"
builder.position_at_end_of bb
builder.ret builder.load x, "obj"

func = mod.add_function "__anon", func_type
bb = context.append_basic_block func, "entry"
builder.position_at_end_of bb

str = builder.global_string "MyRuby", "string"
func_name = builder.global_string "testing", "func_name"

my_ruby = builder.call testing, [str], "rb_str"

builder.ret my_ruby
ee = mod.execution_engine
ee.add_module ruby.mod
p :GOT => ee.run("__anon")
```

Assembles machine code at run time using LLVM
Idea 1: Custom Instructions at Runtime
Data I acquired (crap data)
Build a Police Scanner
Software Defined Radio
FM Band
Narrow FM

Digital Trunking signal

Audio
Digital Signal Processing
Continuous / Discrete

Very Discrete
Discrete signals are just a list of points
Discrete Signal (crap data)
Convolution
Signal Convolution
Formal Definition

\[ y[i] = \sum_{j=0}^{M-1} h[j] x[i - j] \]
def convolve x, h
    Array.new(x.length + h.length - 1) do |i|
        h.map.with_index { |h_j, j|
            if i - j < 0 || i - j > x.length - 1
                0
            else
                h_j * x[i - j]
            end
        }.inject(:+)
    end
end
Convolved Signal Length:
\[ x.\text{length} + h.\text{length} - 1 \]
Low Pass Filters
Identity Signal

- Graphs showing the relationship between X, Y, and H.
- Axes and values labeled accordingly.
Shift
Radar
Radar

Transmission

Reception
Does one signal contain the other?
received \ast \text{ transmitted}
Gaussian Distribution
signal = 100.times.map { rand }
3.times do
  signal = convolve signal, signal
end
Self Convolution

Gaussian Distribution

...
def convolve x, h
    Array.new(x.length + h.length - 1) do |i|
        h.map.with_index { |h_j, j|
            if i - j < 0 || i - j > x.length - 1
                0
            else
                h_j * x[i - j]
            end
        }.inject(:+)
    end
end
R, the hard way
Arrays are 1 based

```r
> data <- c(4, 3, 2, 1)
> data[0]
numeric(0)
> data[0] + 1
numeric(0)
> 
```
There must be an easier way!
Data I acquired (crap data)
Implementing cons, car, and cdr in many languages

(using only lambdas)
Why?
I like learning new languages
But I don’t have much time
Bare minimum to get endorsed on LinkedIn

Do people still use Klout?
What are cons, car, and cdr?
cons forms a cell

L R
car returns the left side
cdr returns the right side

L  R
Only Lambdas?
```
R

```tenderlove.cons <- function(x, y) {
  function(m) { m(x, y) }
}

```Node

```var cons = function(x, y) {
  return function(m) { return m(x, y); }
}

```Ruby

```cons = ->(x, y) {
  ->(m) { m.(x, y) }
}
tenderlove.car <- function(z) {
  z(function(p, q) { p })
}

var car = function(z) {
  return z(function(p, q) { return p; });
}

car = ->(z) {
  z.->(p, q) { p }
}
Usage

> tenderlove.cons(1, 2)
function(m) { m(x, y) }
<environment: 0x7f8c55961f80>
> tenderlove.car
function(z)    { z(function(p, q) { p }) }  
> tenderlove.car(tenderlove.cons(1, 2))
[1] 1
>
tenderlove.cdr <- function(z) {
  z(function(p, q) { q })
}

var cdr = function(z) {
  return z(function(p, q) { return q; });
}

cdr = ->(z) {
  z.-(->(p, q) { q })
}
each

tenderlove.each <- function(func, lst) {
  if (!is.null(lst)) {
    element <- tenderlove.car(lst)
    func(element)
    tenderlove.each(func, tenderlove.cdr(lst))
  }
}

var each = function(func, list) {
  if (list) {
    element = car(list);
    func(element);
    each(func, cdr(list));
  }
}

each = ->(func, list) {
  if list
    element = car(list)
    func(element)
    each(func, cdr(list))
  end
}
Usage

```javascript
var each = function(func, list) {
    if(list) {
        element = car(list);
        func(element);
        each(func, cdr(list));
    }
}

each(console.log, cons(1, (cons(2, cons(3, null)))));
```

```
[aaron@TC dsp (master)]$ node dsp.js
1
2
3
```


```bash
[aaron@TC dsp (master)]$
```
Don’t need loops
Don’t need arrays
Don’t need hashes
Numbers?
Numbers

tenderlove.zero <- function(f) {
  function(x) { x }
}

tenderlove.one <- function(f) {
  function(x) { f(x) }
}

tenderlove.two <- function(f) {
  function(x) { f(f(x)) }
}

tenderlove.three <- function(f) {
  function(x) { f(f(f(x))) }
}

print(tenderlove.three(function(x) { x + 1 })(0)) => 3
Addition

tenderlove.one <- function(f) {
  function(x) { f(x) }
}

tenderlove.add <- function(a, b) {
  function(f) { function(x) { a(f)(b(f)(x)) } }
}

tenderlove.two <- tenderlove.add(tenderlove.one, tenderlove.one)
tenderlove.three <- tenderlove.add(tenderlove.one, tenderlove.two)

print(tenderlove.three(function(x) { x + 1 })(0)) # => 3
Church Encoding
Don’t need Numbers
Don’t need Mathematics
Logic?
true / false / if

True = ->(x, y) { x }
False = ->(x, y) { y }
If = ->(cond, t, f) { cond.(t, f) }

If.(True, ->() { :true }, ->() { :false }).call # => :true
If.(False, ->() { :true }, ->() { :false }).call # => :false
Don’t need booleans
Don’t need conditionals
tenderlove.cons <- function(x, y) { function(m) { m(x, y) } }
tenderlove.car <- function(z) { z(function(p, q) { p }) }
tenderlove.cdr <- function(z) { z(function(p, q) { q }) }

tenderlove.each <- function(func, lst) {
  if (!is.null(lst)) {
    element <- tenderlove.car(lst)
    func(element)
    tenderlove.each(func, tenderlove.cdr(lst))
  }
}

tenderlove.each_with_index <- function(func, lst) {
  idx <- 0
  tenderlove.each(function(element) {
    func(idx, element)
    idx = idx + 1
  }, lst)
}
array indexing

tenderlove.vec_at <- function(idx, lst) {
  if(idx == 0) {
    tenderlove.car(lst)
  } else {
    tenderlove.vec_at(idx - 1, tenderlove.cdr(lst))
  }
}
Converting Vectors to cons

tenderlove.c2cons <- function(cs) {
  foldl <- function(proc, init, lst) {
    if(length(lst) == 0) {
      init
    } else {
      foldl(proc, proc(head(lst, 1), init), tail(lst, -1))
    }
  }
  foldl(tenderlove.cons, NULL, rev(cs))
}
Convolution in R

tenderlove.conv <- function(x, h) {
  x_list <- tenderlove.c2cons(x)
  h_list <- tenderlove.c2cons(h)

  indexes <- 0:(length(x) + length(h) - 1)
  sapply(indexes, function(i) {
    sum(tenderlove.cons2c(tenderlove.map_with_index(function(j, h_j) {
      if (i - j < 0 || i - j > length(x) - 1) {
        0
      } else {
        h_j * tenderlove.vec_at(i - j, x_list)
      }
    }, h_list)))
  })
}
Feed in test data

> data <- c(0, -1, -1.2, 2, 1.5, 1.4, 0.5, 0, -0.5)
> plot(data)
> filter <- c(2, 0, 1)
> plot(filter)
> plot(tenderlove.conv(data, filter))
tenderlove

Nice.
Input Real Data

> plot(tenderlove.conv(unlist(dat$V2), c(1)))

Error: evaluation nested too deeply: infinite recursion in function (expressions=)?

I f#$@*ing hate R.
Things I don’t have time for:
Figuring out R.
My Signal (crap data)
Something I built
Bathroom Scale
msp430
Raspberry Pi + Motion Sensor
I made this

Weight (g)

Cat Enters

Cat Leaves

Time

2017-01-23 15:53:51
2017-01-23 15:53:56
2017-01-23 15:54:01
2017-01-23 15:54:06
2017-01-23 15:54:11
2017-01-23 15:54:16
2017-01-23 15:54:21
2017-01-23 15:54:26
2017-01-23 15:54:31
2017-01-23 15:54:37
2017-01-23 15:54:42
2017-01-23 15:54:47
2017-01-23 15:54:52
2017-01-23 15:54:57
2017-01-23 15:55:02
2017-01-23 15:55:07
2017-01-23 15:55:12
2017-01-23 15:55:17
2017-01-23 15:55:22
2017-01-23 15:55:27
2017-01-23 15:55:32
Wrap-Up
It’s hard to make shit scale

I told you those graphs were crap