clojure-scheme

Compiling to Native Code via Scheme

STARRING: GAMBIT SCHEME

AND INTRODUCING:
Nathan
“@takeoutweight”
SORENSON

clojure-scheme

Gambit on iOS

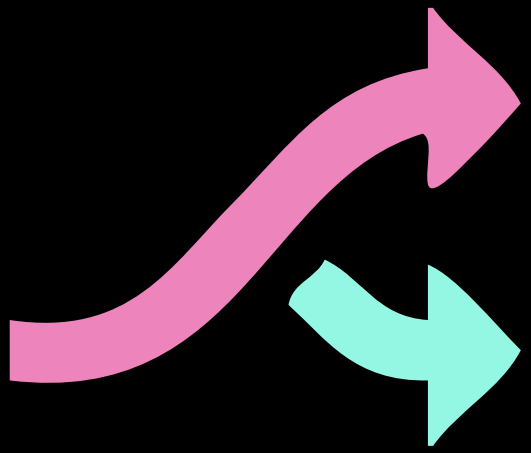
James Long

@jlongster

clojure-scheme

1) Clojure on Scheme
2) Clojure on Gambit Scheme
3) Clojure on iOS

clojure-scheme

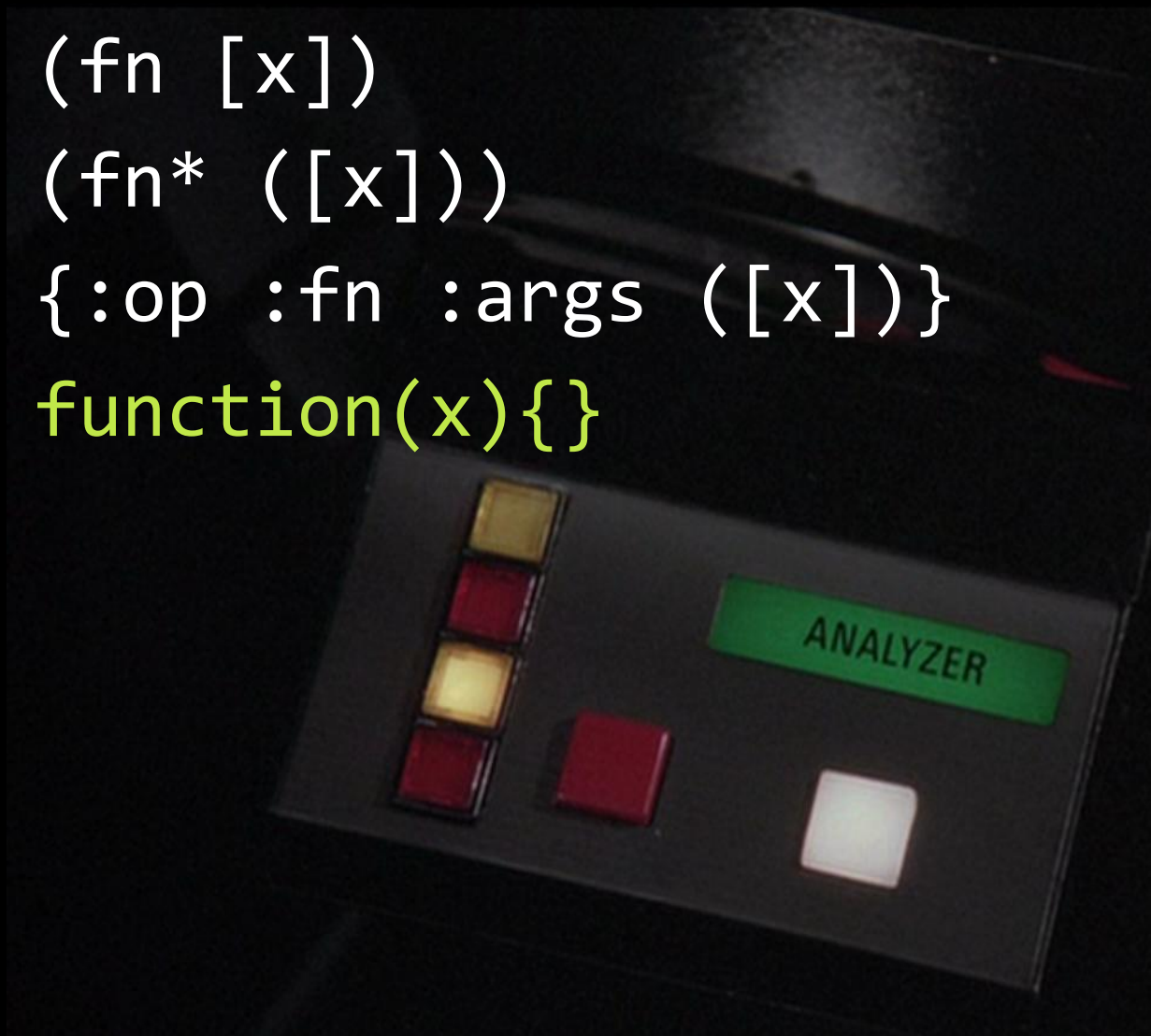


```
function(x) {x.slice(1)}
```

```
(lambda (x) (cdr x))
```

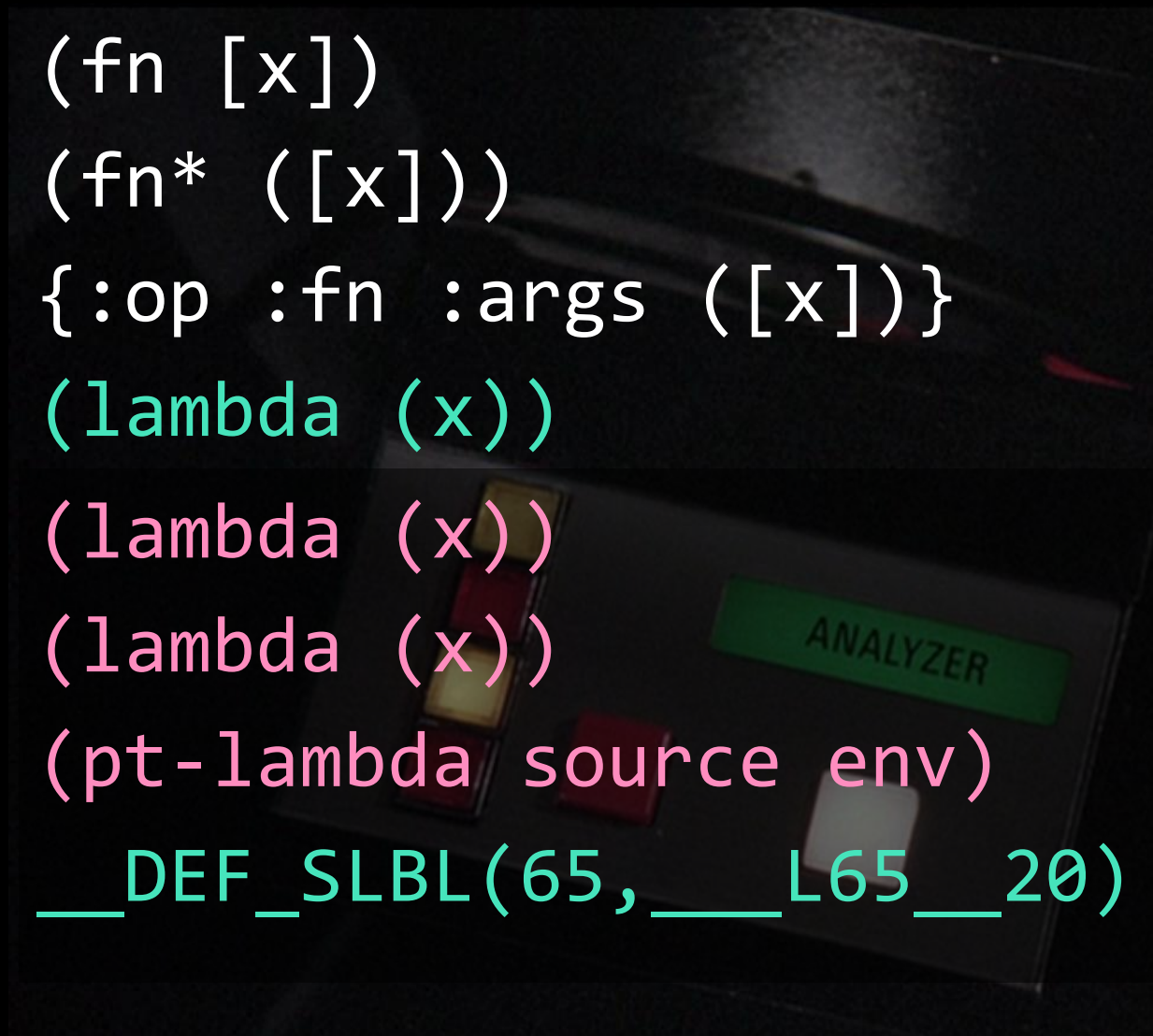
ClojureScript

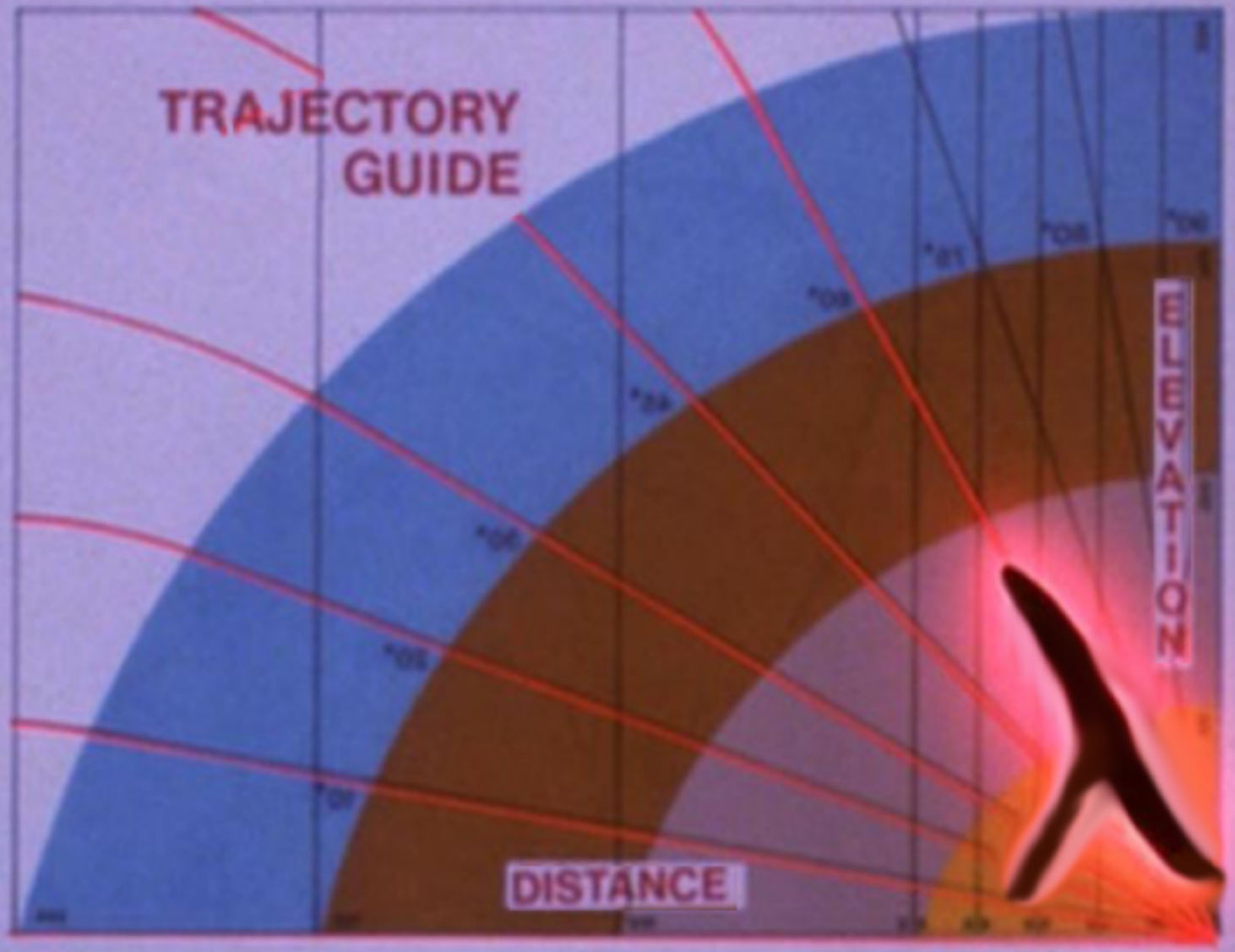
Reading	`(fn [x])`
Macroexpansion	`(fn* ([x]))`
Analysis	`{:op :fn :args ([x])}`
Emission	`function(x){}`

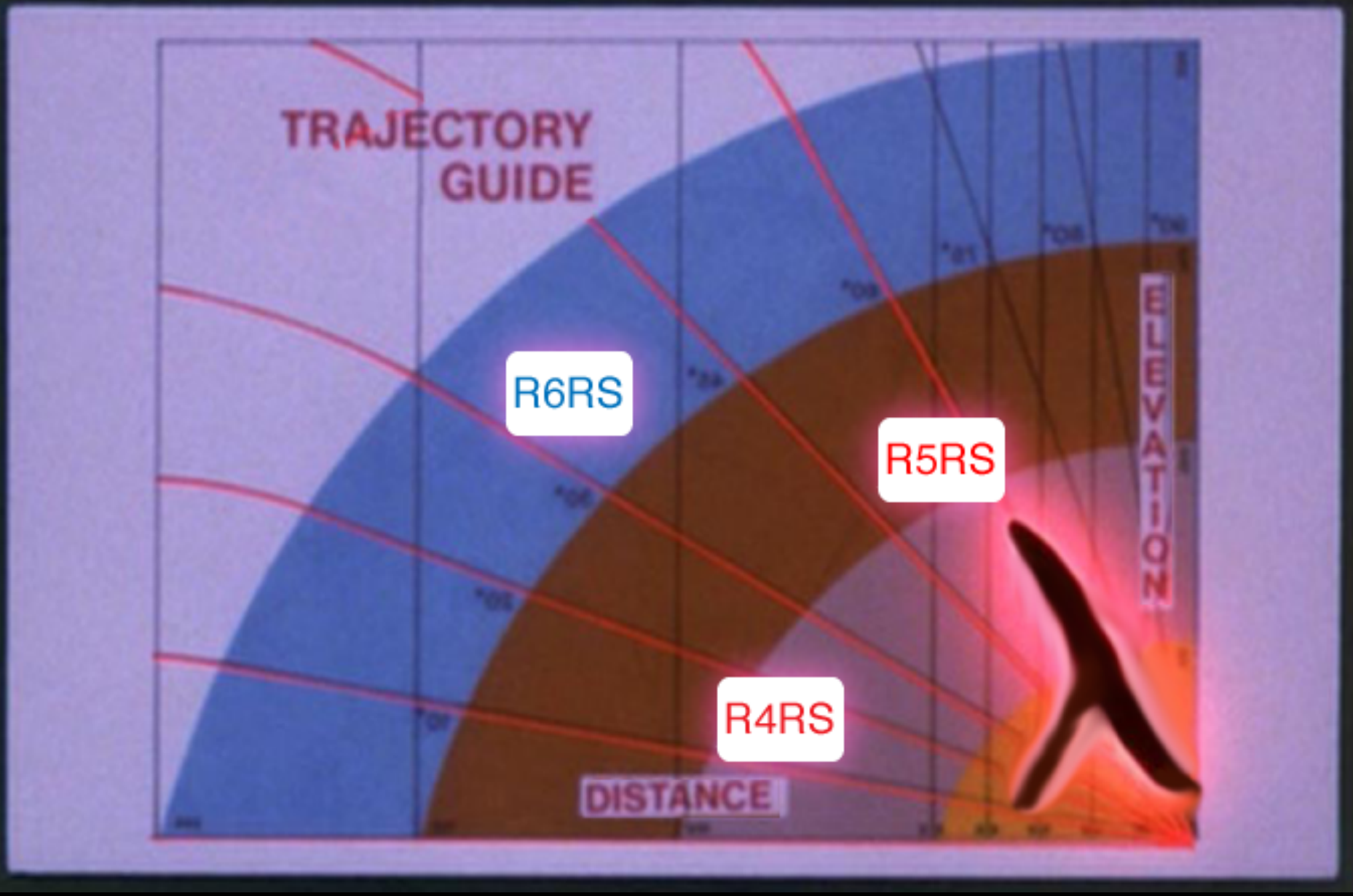


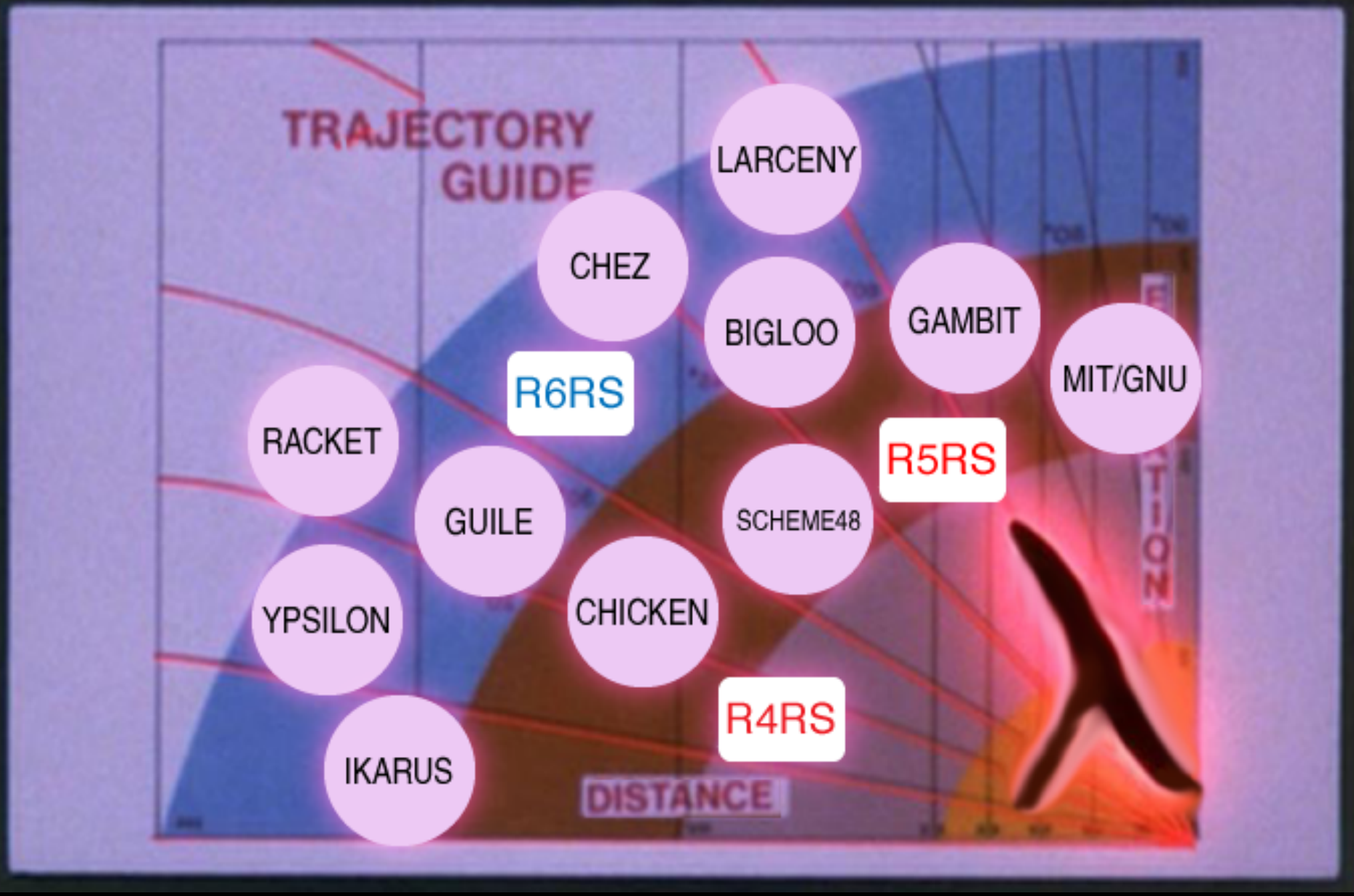
clojure-scheme

Reading	`(fn [x])`
Macroexpansion	`(fn* ([x]))`
Analysis	`{:op :fn :args ([x])}`
clj-scm: Emission	`(lambda (x))`
scm: Reading	`(lambda (x))`
Macroexpansion	`(lambda (x))`
Analysis	`(pt-lambda source env)`
Emission	`__DEF_SLBL(65,___L65__20)`




TRAJECTORY
GUIDE
ELEVATION
DISTANCE


TRAJECTORY
GUIDE
R6RS
R5RS
R4RS
DISTANCE
ELEVATION


TRAJECTORY
GUIDE
LARCENY
CHEZ
BIGLOO
GAMBIT
MIT/GNU
R6RS
RACKET
R5RS
GUILE
SCHEME48
YPSILON
CHICKEN
R4RS
IKARUS
DISTANCE

Clojure and Scheme

- Functional
- Dynamically Typed
- Eagerly Evaluated
- Proper Lexical Scope
- Closures
- Minimal

Macro-Expressible*

`fn`	`lambda`
`(loop [x 1] (recur 2))`	`(let loop ((x 1)) (loop 2))`
`try/catch`	`with-exception-handler`
`apply`	`apply ;(*)`
`if <expr>`	`(let ((e <expr>)) (if (and (!= e #f) (!= e #!void))))`

* “On the Expressive Power of Programming Languages” Matthias Felleisen

Trickier

Persistent Data Structures

Lazy Sequences

Trickier

Garbage Collection

\+ Closures

Persistent Data Structures

Lazy Sequences

Trickier

Garbage Collection

\+ Closures

\+ Polymorphism

Persistent Data Structures

Lazy Sequences

Running on Scheme

- `call/cc` & friends
  - Resumable exceptions
  - Backjumping search
  - Ambiguous operator
  - Cooperative concurrency
- Proper Tail Calls

An Old Wrong Put Right

- [plt-scheme] Android; compiling to Java byte code; Clojure, *Geoffrey S. Knauth*
  - [plt-scheme] Android; compiling to Java byte code; Clojure, *Benjamin L. Russell*
    - [plt-scheme] Android; compiling to Java byte code; Clojure, *Noel Welsh*
      - [plt-scheme] Android; compiling to Java byte code; Clojure, *Benjamin L. Russell*
    - [plt-scheme] Android; compiling to Java byte code; Clojure, *Shriram Krishnamurthi*
      - [plt-scheme] Android; compiling to Java byte code; Clojure, *Benjamin L. Russell*
      - [plt-scheme] Android; compiling to Java byte code; Clojure, *Shriram Krishnamurthi*
      - [plt-scheme] Android; compiling to Java byte code; Clojure, *Benjamin L. Russell*
      - [plt-scheme] Android; compiling to Java byte code; Clojure, *Robby Findler*
      - [plt-scheme] Android; compiling to Java byte code; Clojure, *Shriram Krishnamurthi*
      - [plt-scheme] Android; compiling to Java byte code; Clojure, *Benjamin L. Russell*
      - [plt-scheme] Android; compiling to Java byte code; Clojure, *Henk Boom*

An Old Wrong Put Right

From: **Rich Hickey** (rich at richhickey.com)
To: **[plt-scheme]**
Date: *Wed Nov 28 09:23:57 EST 2007*

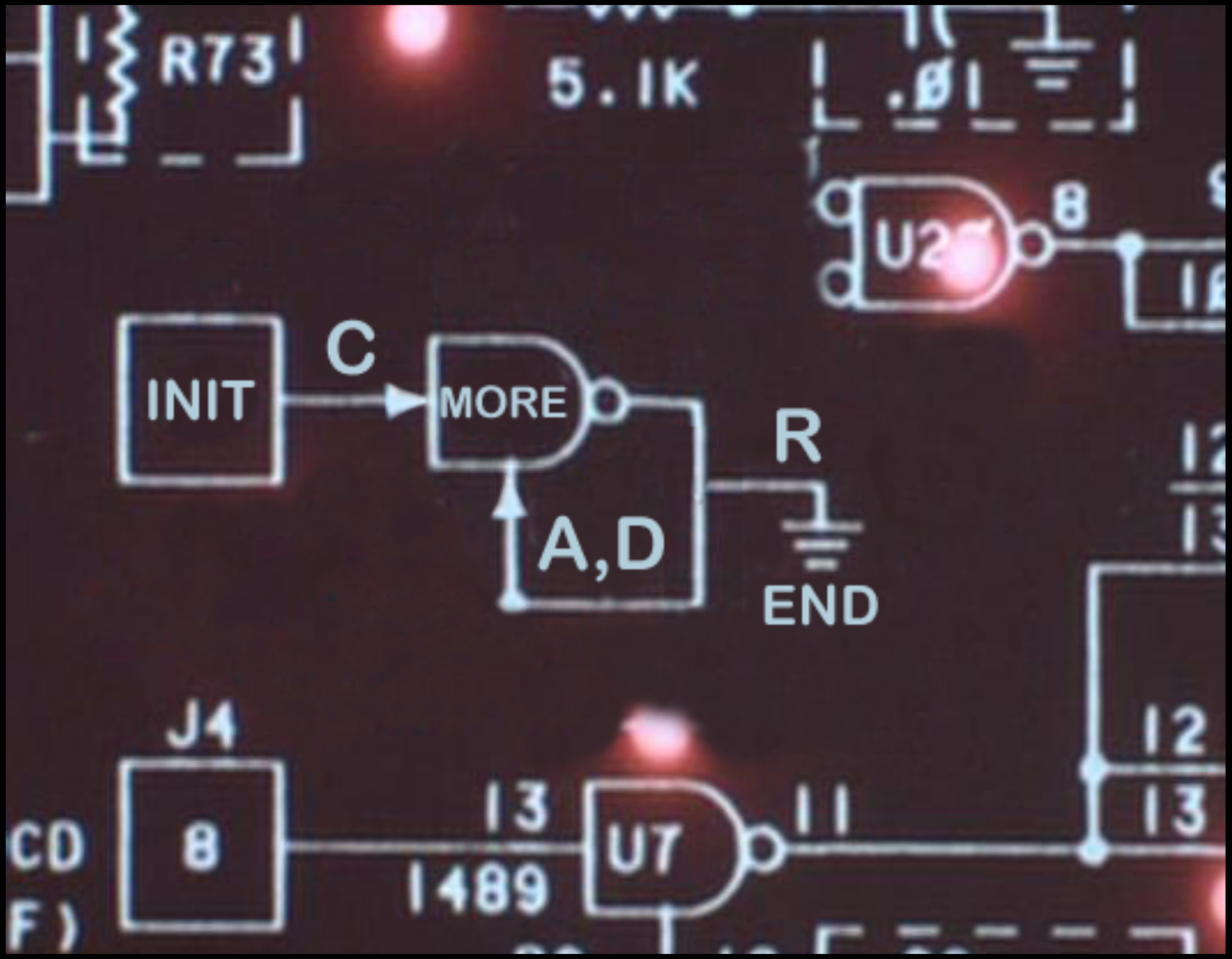
Hi, I'm the author of Clojure. Here's how I would write it in Clojure:

```
(defn machine [stream]
  (let [step {[:init 'c] :more
              [:more 'a] :more ...
```

Regards, Rich Hickey

caaddr Langauge

```
(car ‘(1 2 3)) =>  1
(cdr ‘(1 2 3)) => (2 3)
(cadr ‘(1 2 3)) => 2
(caddar ‘((1 2 3) 4)) => 3
```


R73
5.1K
.01
U2
8
INIT
C
MORE
R
A,D
END
J4
CD
8
13
1489
U7
11
12
13
F)

caaddr Langauge

```
(defn machine [stream]
  (let [step {[:init 'c] :more
              [:more 'a] :more
              [:more 'd] :more
              [:more 'r] :end
              [:end nil] :t}]
    (loop [state :init
           stream stream]
      (let [next (step [state (first stream)])]
        (when next
          (if (= next :t)
              :t
              (recur next (rest stream)))))))))
```

“The Swine Before Perl” Shriram Krishnamurthi

```
(define (init stream)
    (case (car stream)
      ((c) (more (cdr stream)))))

(define (more stream)
    (case (car stream)
      ((a) (more (cdr stream)))
      ((d) (more (cdr stream)))
      ((r) (end  (cdr stream)))))

(define (end stream) (null? stream))
```

caaddr Langauge

2 million character string

jvm loop/recur: **630ms**

gambit tail calls: **23ms** (27x)

Why OO Languages Need Tail Calls

Guy Steele: Tail Call Optimization is Pretty Cool.

http://www.eighty-twenty.org/index.cgi/tech/oo-tail-calls-20111001.html

Gambit Scheme

- Largely self-hosted
- Targets “GVM” bytecode
- Interpreted or compiled
- Flexible FFI constructs
- Embeddable runtime
- Green Threads (OS Threads coming soon)
- Long compile times: 7kloc ~ 14mins $O(n^2)$

Installing Gambit

```
$ port install gcc-mp-4.8
$ git clone git://github.com/feeley/gambit.git
$ ./configure --enable-single-host --prefix=/usr/local/Gambit-C/v4.6.7.gcc4.8 --enable-multiple-versions CC=gcc-mp-4.8 CXX=g++-mp-4.8 CPP=cpp-mp-4.8
$ make install
```

Installing clojure-scheme

$ git clone https://github.com/takeoutweight/clojure-scheme.git

```
user> (require 'cljscm.compiler)
user> (cljsm/compile-file “myfile.cljscm”)
  => myfile.scm
```

$ gsc

```
 > (load “cljscm_core”)
 > (load “myfile.scm”)
 > (myns/myplus 3 4) => 7
```

$ gsc myfile.scm => myfile.o1

$ gsc -exe myfile.scm => ./myfile

Gratuitous Microbenchmarks

Gratuitous Microbenchmarks

```
(fib 36)
```

Clojure: **1130**ms

ClojureScript on V8: **780**ms (1.4x)

clojure-scheme: **600**ms (1.9x)

Gratuitous Microbenchmarks

(fib 36)

Clojure: 1130ms

ClojureScript on V8: 780ms (1.4x)

clojure-scheme: 600ms (1.9x)

MRI 1.8: 26,200ms

Gratuitous Microbenchmarks

```
(reduce + (take 1000000 (range)))
```

ClojureScript on V8: **1630ms**

clojure-scheme: **860ms** (1.9x)

Clojure: **350ms** (4.6x)

clojure-scheme:
(no polymorpishm) **110ms** (14.6x)

Gratuitous Microbenchmarks

```
(reduce conj {}
  (take 10000 (map (fn[x][x x])(range))))
```

clojure-scheme: **280ms**

ClojureScript on V8: **180ms** (1.5x)

Clojure: **15ms** (18x)

Gratuitous Microbenchmarks

Compiling 3 kloc file

Clojure: **0.9**s (205x)

ClojureScript: **2.6**s (71x)

clojure-scheme: **185.0**s (3 minutes)

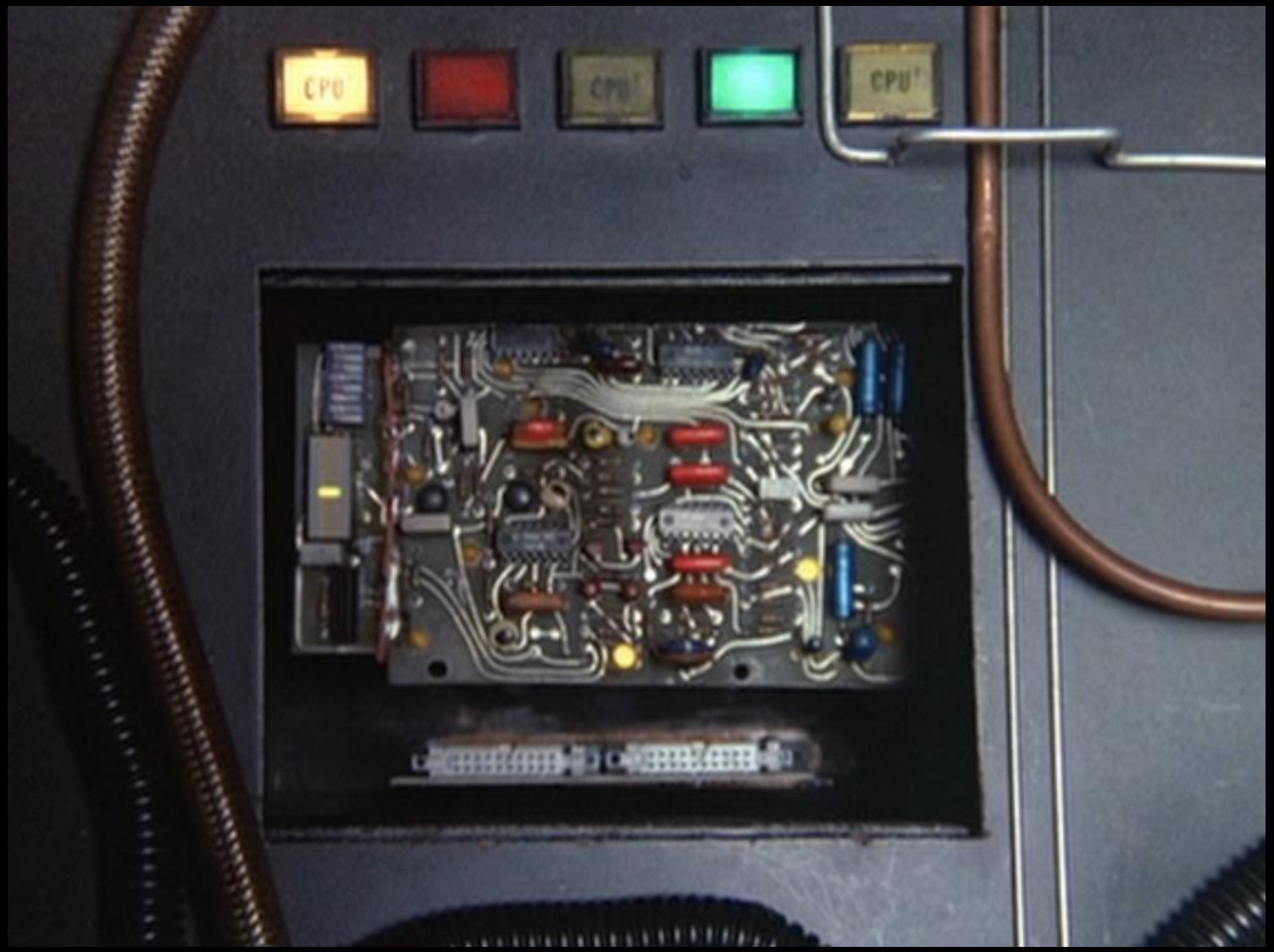

TURBO
BOOST
VOICE
ANALYZER

Turbo Boosters

- `block`
- `fixnum`
- `unsafe`*
- `c-lambda`**

(*) *“Caution, the Risk Factor is extremely high!”*

FFI



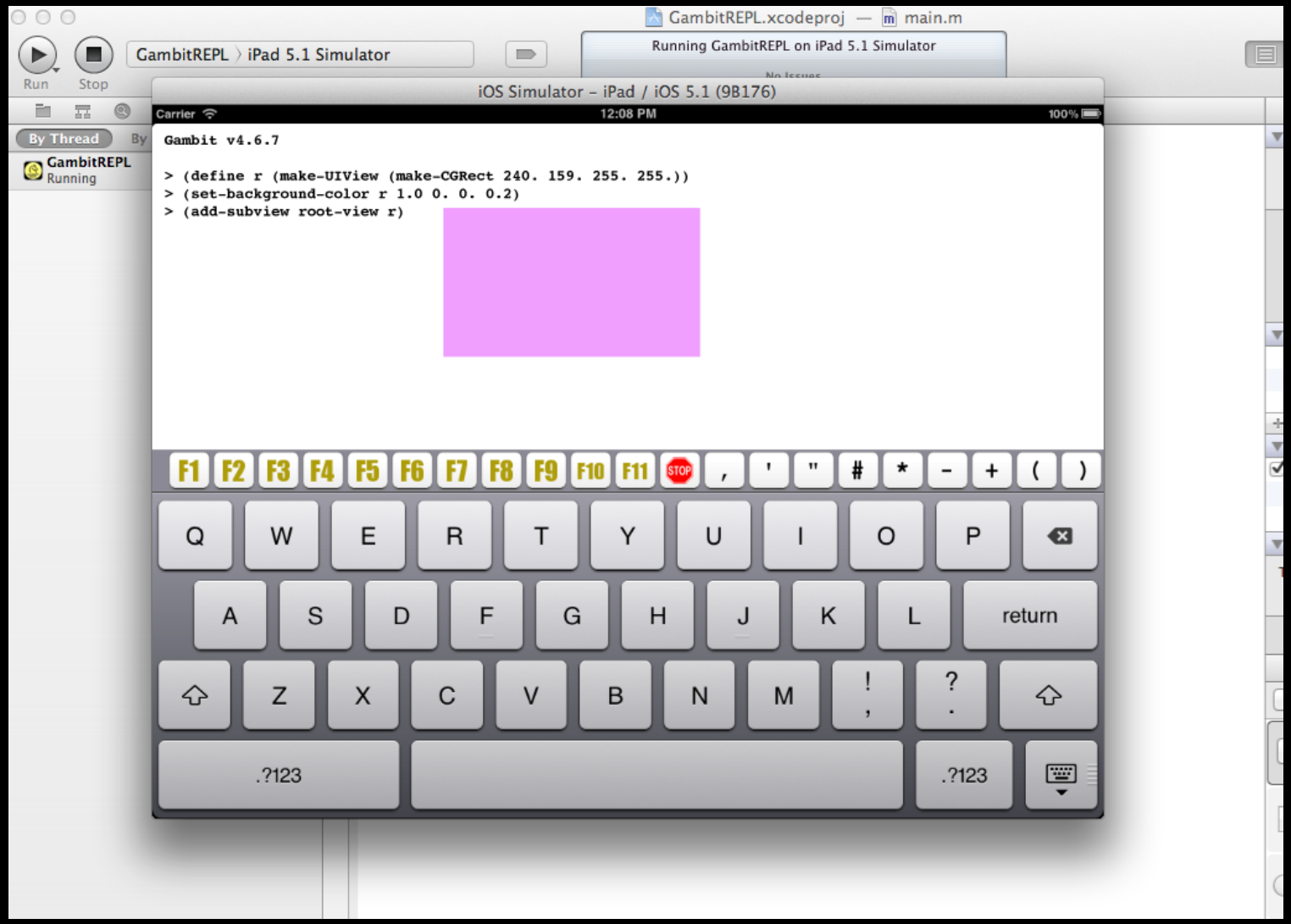
FFI

`c-define-type` *type conversion*

`c-lambda` *scheme->C, inline C*

`c-define` *C->scheme, C callable stubs*

Hello Rectangle



Building the iOS Cross-Compiler

- Need Xcode command line tools & iPhone SDK
- edit misc/build-gambit-iOS
  - ios_version="5.1”
  - platforms_dir="/Applications/Xcode.app/Contents/Developer/Platforms”
  - choose armv7 as native target
- download an official source distribution:
  - gambc-v4_6_7.tgz (or tweak configure script)

$ sh misc/build-gambit-iOS

Building the iOS Example

```
$ cd gambit/contrib/GambitREPL
$ ln -s ../../misc/gambit-iOS gambit-iOS
$ make prepare-for-xcode
$ make program_.m
```

and disable Xcode's "show live issues"

Hello Rectangle

```
CGRect viewRect = CGRectMake(10, 500, 100, 100);

UIView *rct =
[[UIView alloc] initWithFrame:viewRect];

rct.backgroundColor =
[UIColor colorWithRed:0.5 green:0.5 blue:1.0 alpha:0.9];

[myRootView addSubview: rct];
```

Hello Rectangle

```
(c-define-type CGRect (struct "CGRect"))

(define make-CGRect
  (c-lambda (float float float float) CGRect
"struct CGRect *r = malloc(sizeof *r);
 *r = CGRectMake(___arg1, ___arg2, ___arg3, ___arg4);
 ___result_voidstar = r;"))
```

Hello Rectangle

```
(c-define-type id
  (pointer (struct "objc_object")
  (id Class)
  "release_id"))

(define make-UIView
  (c-lambda (CGRect) id
"__result = retain_id(
     [[UIView alloc] initWithFrame:__arg1]);"))
```

Hello Rectangle

```
(define set-background-color
  (c-lambda (id float float float float) void
 "___CAST(UIView*,___arg1).backgroundColor =
[UIColor colorWithRed:___arg2
   green:___arg3 blue:___arg4 alpha:___arg5];"))

(define add-subview
 (c-lambda (id id) void
    "[___arg1 addSubview: ___arg2];"))
```

Hello Rectangle

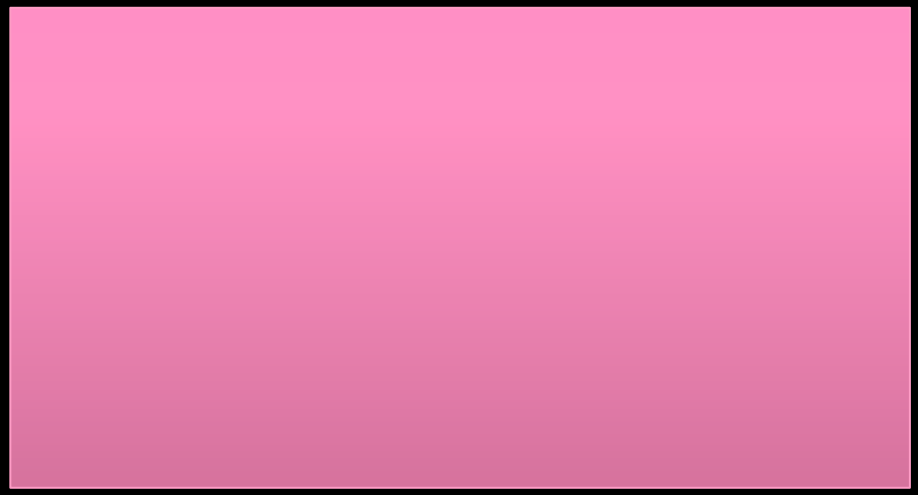
```
(define root-view nil)
(c-define (set-root-view! root) (id) void
  “set_root_view” “extern”
  (set! root-view root))
```

ViewController.m:

```
- (void) viewDidLoad {
    set_root_view(self);
}
```

Hello Rectangle

```
> (define r
    (make-UIView (make-CGRect) 50. 50. 100. 50.)
> (set-background-color r 240. 159. 255. 255.)
> (add-subview root-view r)
```



Dynamic Objective-C

```
objc_msgSend(id theReceiver, SEL
theSelector, ...)
```

```
object_getInstanceVariable(id obj, const
char *name, void **outValue)
```

```
class_addMethod(Class  cls,  SEL  name,  IMP
imp, const char *types)
```

Jason Felice's reflective bridge:

https://github.com/maitria/gambit-objc

Objective-C Proxies

```
(-lookup [o k] (
 {:x (gsc/c-lambda (CGRect) float
       "___result = ___arg1.x;")
  :y …} k))
```

Coming Soon

Ogre 3D Engine

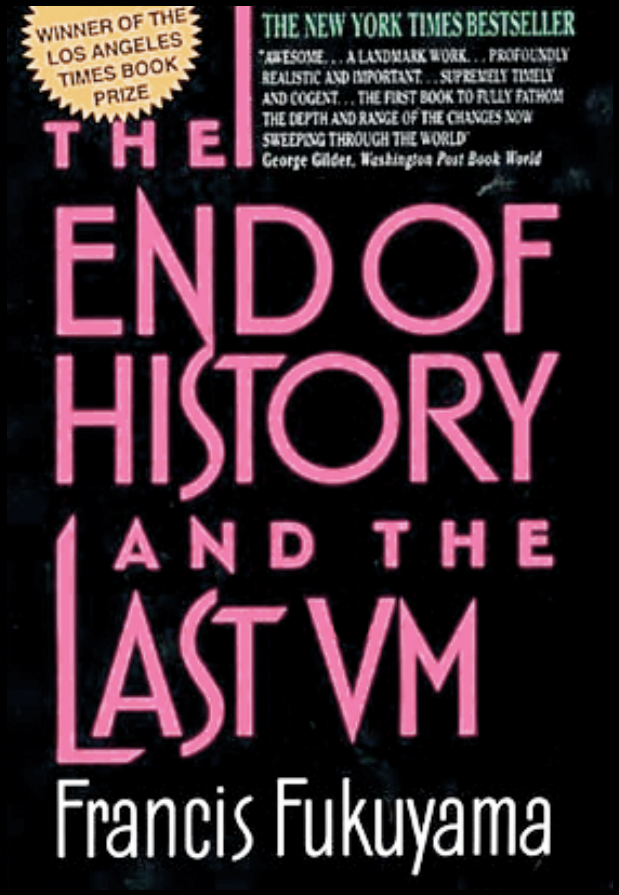
RPythonic generated C++ bindings

Coming Soon

- RPythonic auto-generating C++ bindings
- CMake build scripts
- cljsbuild-like Leiningen plugin
- Clojure repl into iPhone simulator

Existential Angst

Is JavaScript the inevitable, inescapable compiler target?



clojure-scheme

Compiling to Native Code via Scheme

Nathan Sorenson

@takeoutweight

github.com/takeoutweight

gambitscheme.org