ClojureScript

Better Semantics at Low Low Prices!


Java™

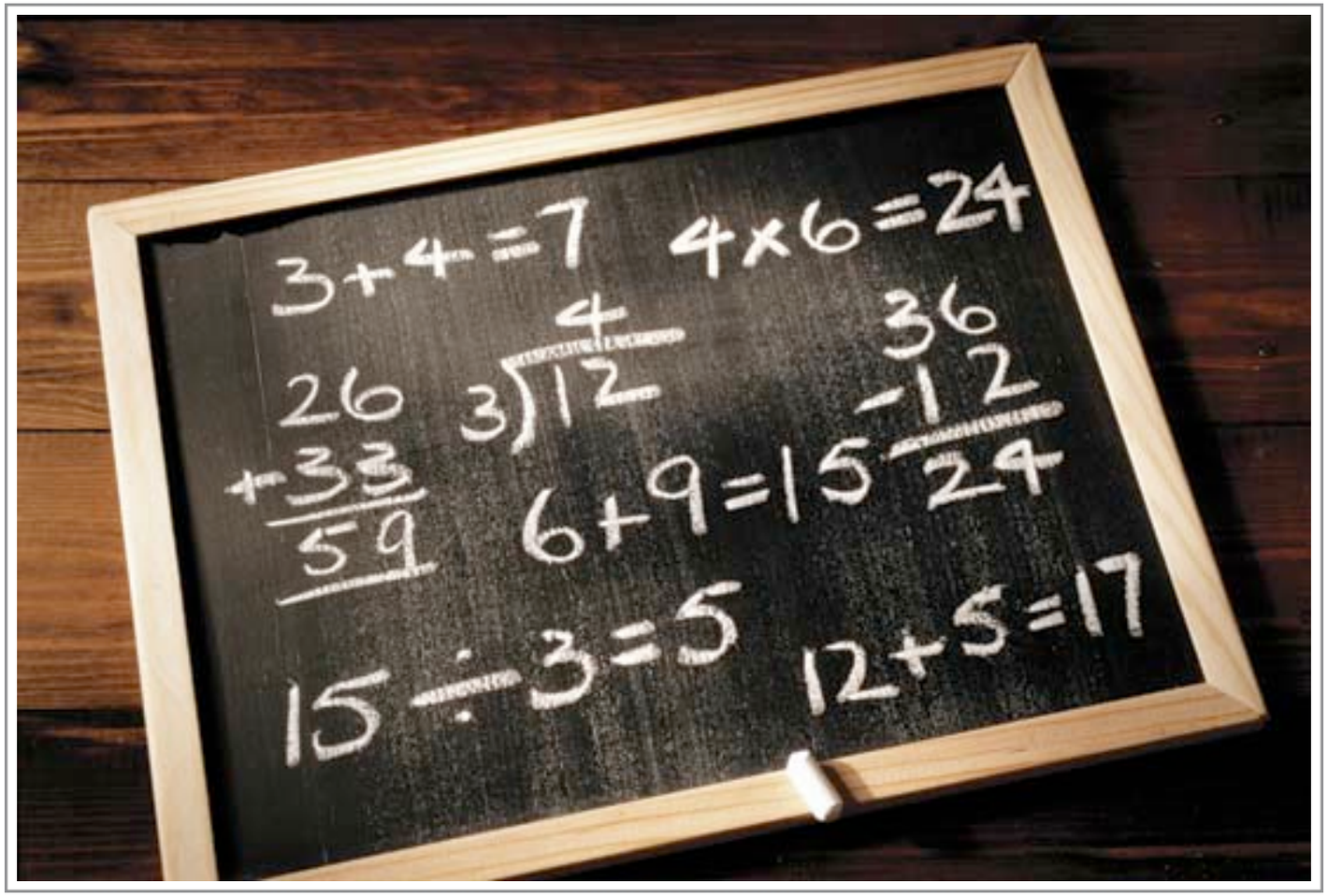

JS


Google

Just JavaScript?

"Lisp programmers know the value of everything and the cost of nothing"

– Alan Perlis


3+4=7
4×6=24
4
26
3)12
36
-12
+33
59
6+9=15
24
15÷3=5
12+5=17

```
var x = 1 + 1;
```

expression oriented

```
(def x (+ 1 1))
```

```
var x = (+).apply(null, [1, 2]);
```

```
(def x (apply + [1 1]))
```

```
var x = foo[0];
```

```
(def x (aget foo 0))
```

```
(def x [+ aget])
```

```clojure
(defmacro aset [a i v]
  (list 'js* "(~{}[~{}] = ~{})" a i v))
```

Compiler macros

Compiler macros

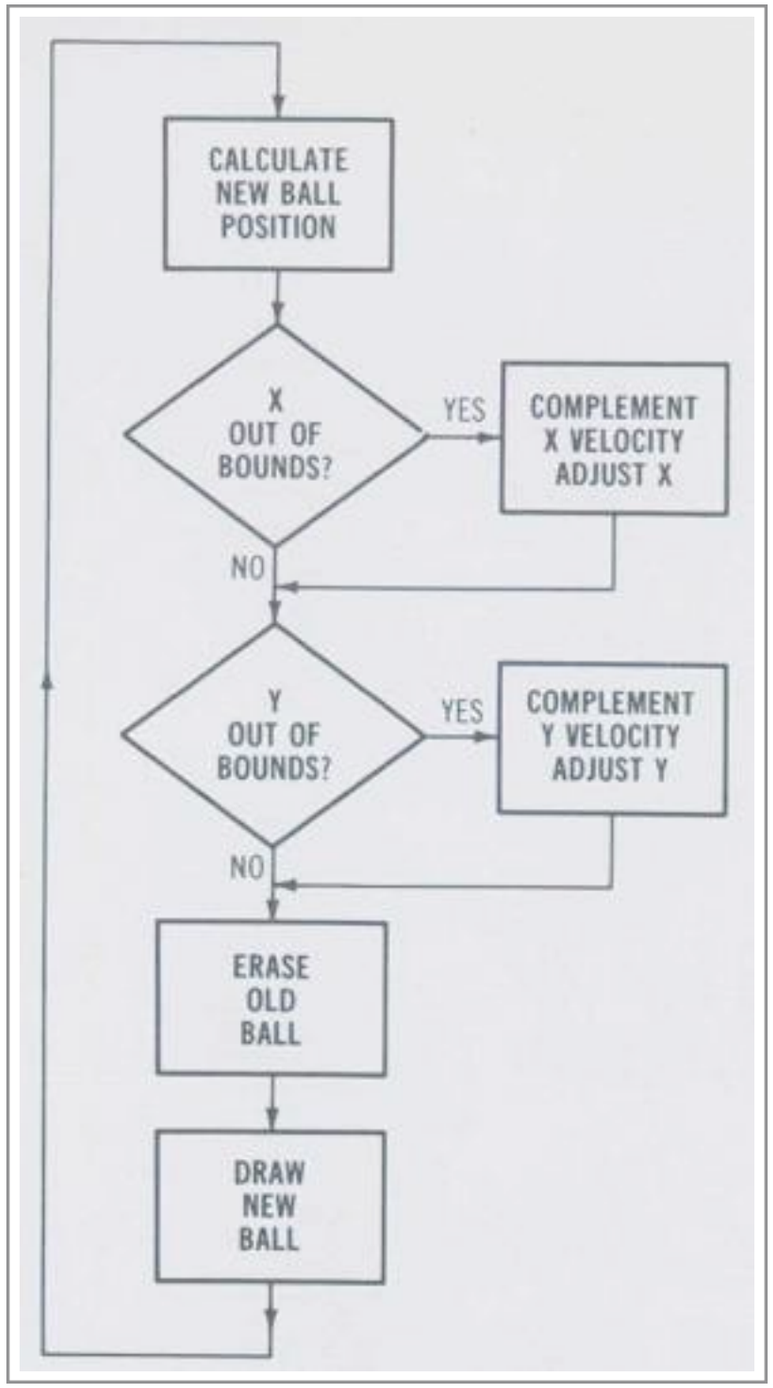
- Optimized behavior for non-higher order usage of *functions* (that is, not using functions as values)

Compiler macros

- Optimized behavior for non-higher order usage of *functions* (that is, not using functions as values)
- In this case we can emit primitive JS operations

Compiler macros

- Optimized behavior for non-higher order usage of *functions* (that is, not using functions as values)
- In this case we can emit primitive JS operations
- Performance and generality!


CALCULATE NEW BALL POSITION
X OUT OF BOUNDS?
YES
COMPLEMENT X VELOCITY ADJUST X
NO
Y OUT OF BOUNDS?
YES
COMPLEMENT Y VELOCITY ADJUST Y
NO
ERASE OLD BALL
DRAW NEW BALL

truth

```
if(x) {
  // ...
} else {
  // ...
}
```

```
if(0) {
   // ...
} else {
   // execute
}
```

```
if("") {
    //  ...
} else {
    // execute
}
```


FFFFFFF
FFFFFFF
FFFFFF
FFFUU
UUUU
UUUU
UUUU
UUUU
UUUU

```
(if x
    true
    false)
```

```
(if 0
true
false)
```

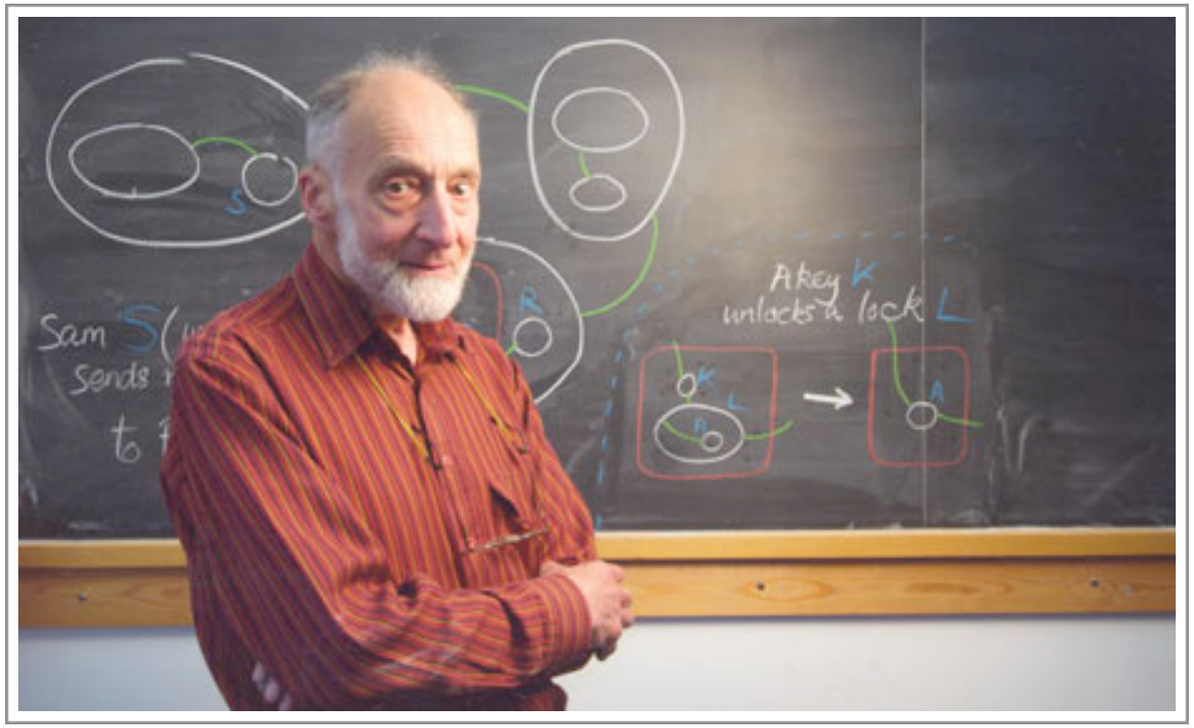
```
(if ""
true
false)
```

```
if(cljs.core.truth_(x)) {
  //  ...
} else {
  //  ...
}
```

```
if(cljs.core.truth_(0)) {
  // execute
} else {
  //  ...
}
```

```
if(cljs.core.truth_("")) {
  // execute
} else {
  //  ...
}
```

```clojure
(defn ^boolean <
  ([x] true)
  ([x y] (cljs.core/< x y))
  ([x y & more]
   (if (cljs.core/< x y)
     (if (next more)
       (recur y (first more) (next more))
       (cljs.core/< y (first more)))
     false)))
```


A key K
unlocks a lock L
K
L
R

type inference

type inference

- ClojureScript compiler has type inference for boolean values in `let` & `if` expressions

type inference

- ClojureScript compiler has type inference for boolean values in `let` & `if` expressions
- `and` / `or` are macros built on `let` & `if`

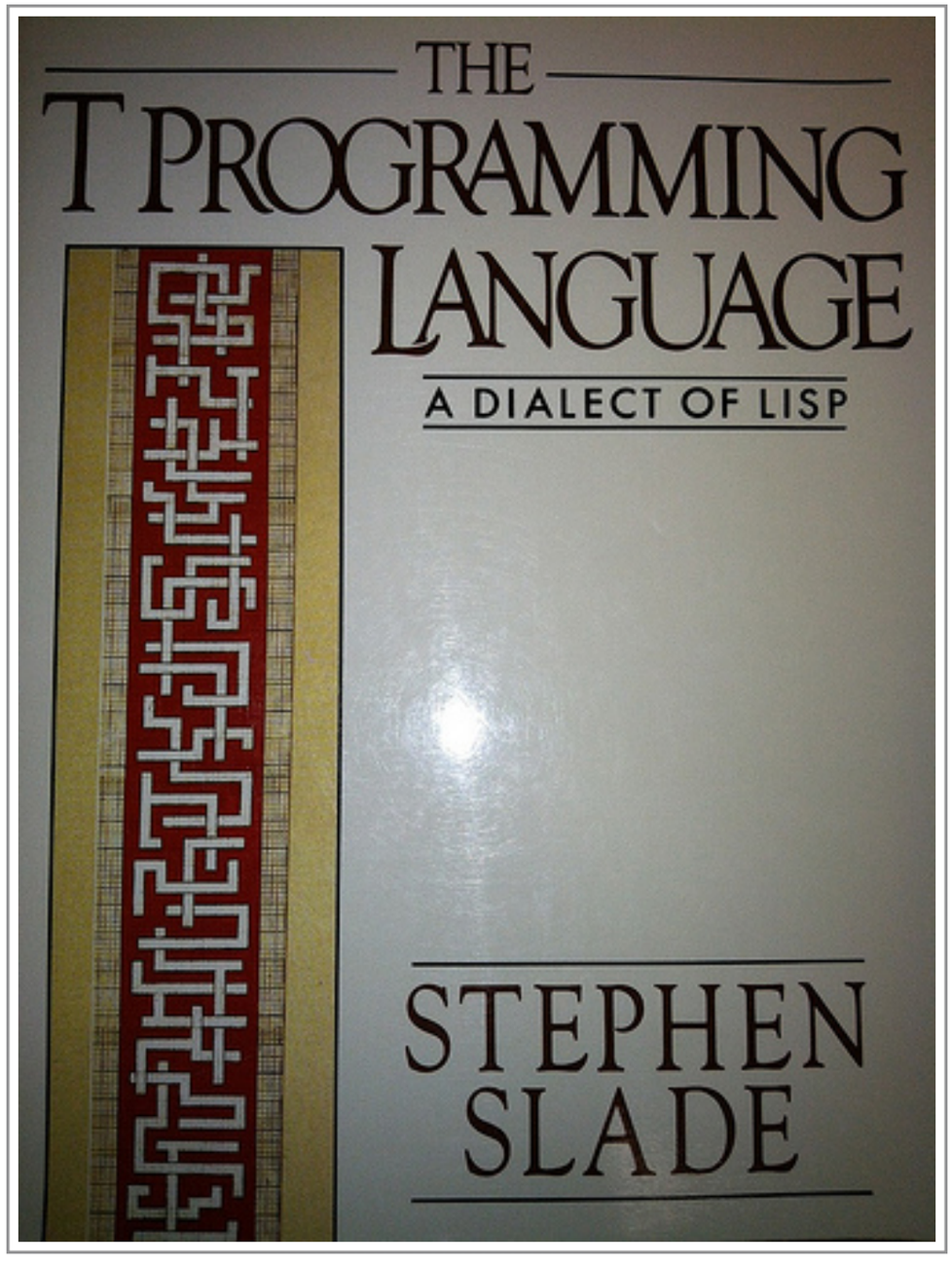
type inference

- ClojureScript compiler has type inference for boolean values in `let` & `if` expressions
- `and` / `or` are macros built on `let` & `if`
- largely eliminates calls to `cljs.core/truth_` in performance critical code (think persistent data structures)

```
(if (< x y)
    true
    false)
```

```
if(cljs.core.truth_(cljs.core._LT_(x, y))) {
  // ...
} else {
  // ...
}
```

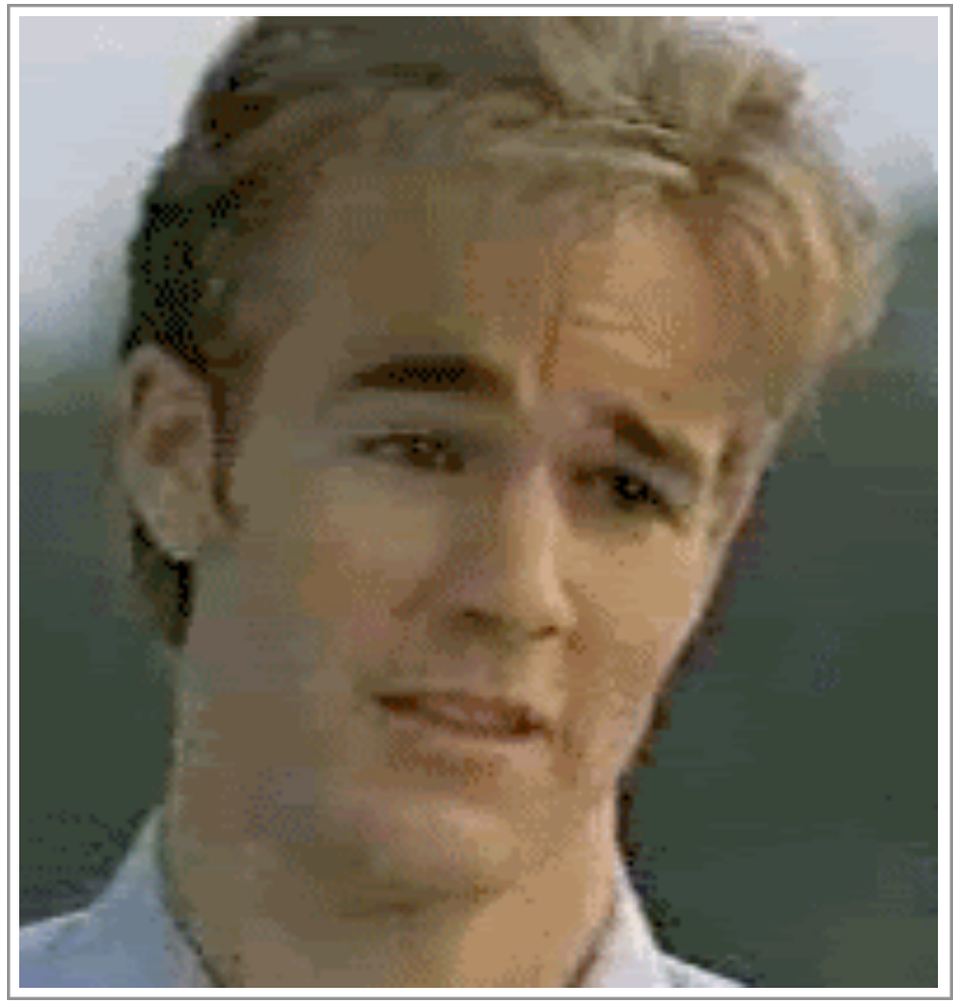
```
if(x < y) {
    //  ...
} else {
    //  ...
}
```


THE
T PROGRAMMING
LANGUAGE
A DIALECT OF LISP
STEPHEN
SLADE

```
(deftype Foo []
  IFn
  (-invoke [this s]
    (.log js/console s)))

((Foo.) "I'm a function!")
```

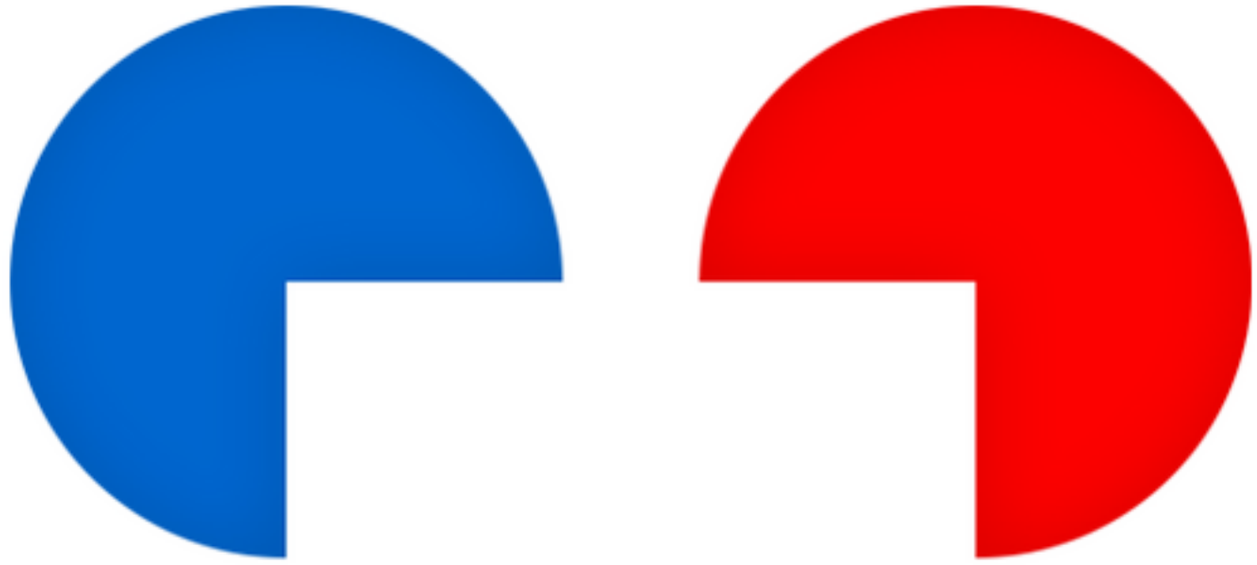
```
fn.call(null, 1, 2);
```

```
obj.call(null, 1, 2);
```



mention expression
oriented

function
unwrapping



Optimizing invokes

Optimizing invokes

- ClojureScript users leverage whole program optimization via Google Closure already

Optimizing invokes

- ClojureScript users leverage whole program optimization via Google Closure already
- We can propagate function information the same way we propagate type hints

Optimizing invokes

- ClojureScript users leverage whole program optimization via Google Closure already
- We can propagate function information the same way we propagate type hints
- We only apply invocation optimizations under whole program optimization

Optimizing invokes

- ClojureScript users leverage whole program optimization via Google Closure already
- We can propagate function information the same way we propagate type hints
- We only apply invocation optimizations under whole program optimization
  - Otherwise would impede interactive development

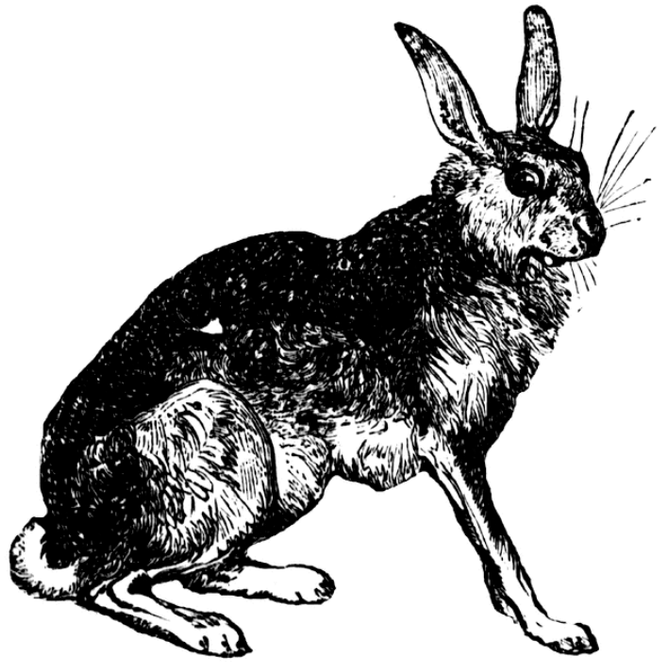
```clojure
(defn foo
  ([] :zero)
  ([x] :one)
  ([x y] :two))
```

```clojure
(foo)     ; => :zero
(foo 1)   ; => :one
(foo 1 2) ; => :two
```

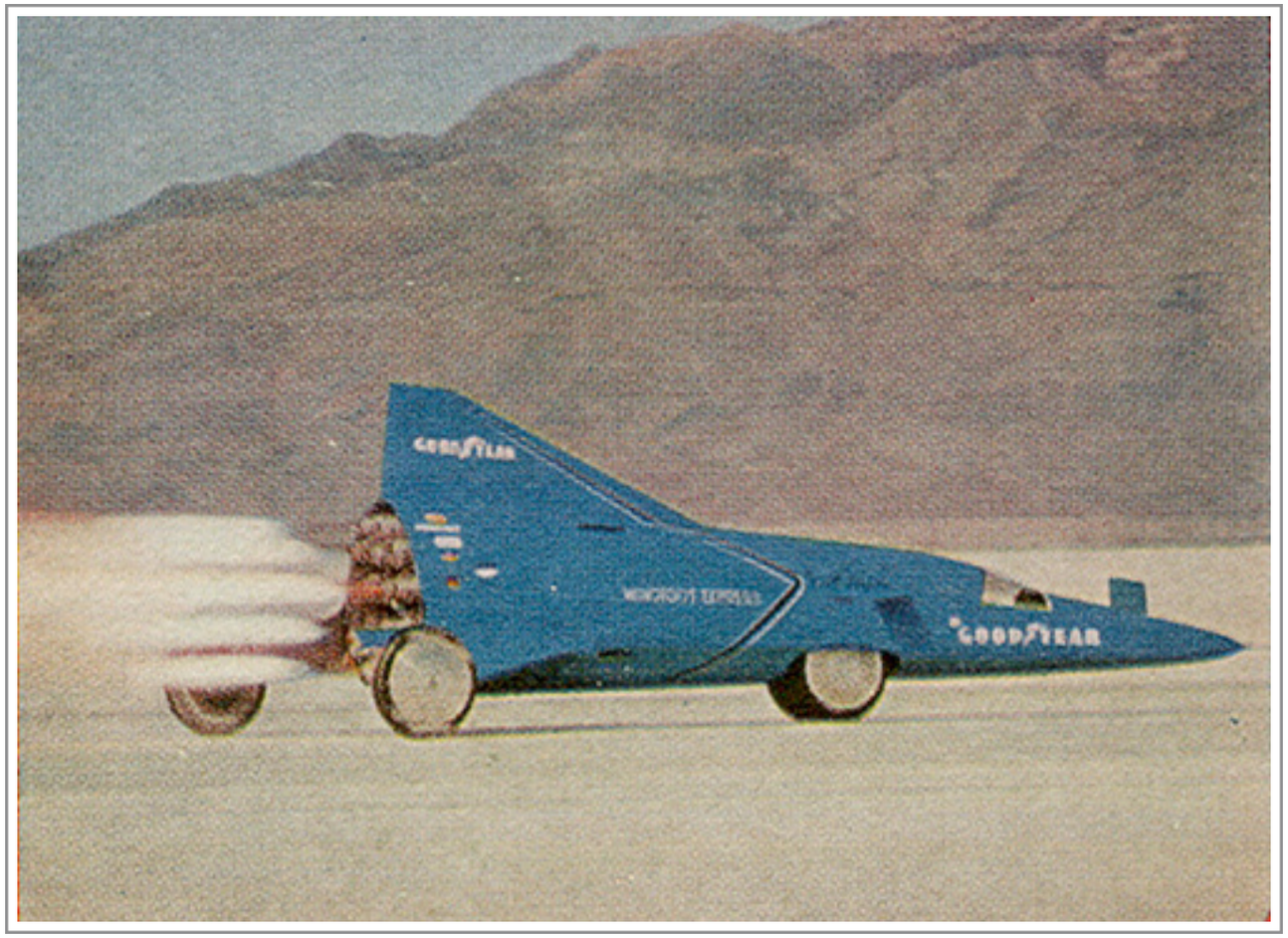
```
function foo() {
   var first = arguments[0];
   var rest = Array.splice(arguments, 1);
   // ...
}
```

```
foo.call(null);
foo.call(null, 1, 2);
```

```
foo.cljs$arity$0();
foo.cljs$arity$1(1);
foo.cljs$arity$2(1, 2);
```




1
2
3
4
5



JVM vs. JS vs. Dart vs. CLJS

```
function A(i,j) {
  return 1/((i+j)*(i+j+1)/2+i+1);
}

function Au(u,v) {
  for (var i=0; i<u.length; ++i) {
    var t = 0;
    for (var j=0; j<u.length; ++j)
      t += A(i,j) * u[j];
    v[i] = t;
  }
}

function Atu(u,v) {
  for (var i=0; i<u.length; ++i) {
    var t = 0;
    for (var j=0; j<u.length; ++j)
      t += A(j,i) * u[j];
    v[i] = t;
  }
}

function AtAu(u,v,w) {
  Au(u,w);
  Atu(w,v);
}
```

```
function spectralnorm(n) {
  var i,
      u=new Float64Array(n),
      v=new Float64Array(n),
      w=new Float64Array(n),
      vv=0,
      vBv=0;
  for (i=0; i<n; ++i) {
    u[i] = 1; v[i] = w[i] = 0;
  }
  for (i=0; i<10; ++i) {
    AtAu(u,v,w);
    AtAu(v,u,w);
  }
  for (i=0; i<n; ++i) {
    vBv += u[i]*v[i];
    vv  += v[i]*v[i];
  }
  return Math.sqrt(vBv/vv);
}

print(
  spectralnorm(+arguments[0]).toFixed(9)
);
```

```clojure
(ns spectral-norm.core)

(defn a [i j]
  (/ 1 (+ (/ (* (+ i j)
                (+ i j 1)) 2) i 1)))

(defn au [u v]
  (let [len (alength u)]
    (dotimes [i len]
      (loop [t 0 j 0]
        (if (< j len)
          (recur (+ t (* (a i j)
                         (aget u j)))
            (inc j))
          (aset v i t))))))

(defn atu [u v]
  (let [len (alength u)]
    (dotimes [i len]
      (loop [t 0 j 0]
        (if (< j len)
          (recur (+ t (* (a j i)
                         (aget u j)))
            (inc j))
          (aset v i t))))))

(defn atau [u v w]
  (au u w)
  (atu w v))
```

```clojure
(defn approximate [n]
  (let [u (array) v (array) w (array)]
    (dotimes [i n]
      (aset u i 1)
      (aset v i 0)
      (aset w i 0))
    (dotimes [i 10]
      (atau u v w)
      (atau v u w))
    (loop [i 0 vbv 0 vv 0]
      (if (< i n)
        (recur (inc i)
               (+ vbv (* (aget u i)
                         (aget v i)))
               (+ vv (* (aget v i)
                        (aget v i))))
        (.sqrt js/Math (/ vbv vv))))))

(js/print (approximate 5500))
```

```dart
#import('dart:math');

num A(i, j) {
  return 1/((i+j)*(i+j+1)/2+i+1);
}

void Au(u, v) {
  for (var i=0; i<u.length; i++) {
    var t = 0;
    for (var j=0; j<u.length; j++)
      t += A(i,j) * u[j];
    v[i] = t;
  }
}

void Atu(u, v) {
  for (var i=0; i<u.length; i++) {
    var t = 0;
    for (var j=0; j<u.length; j++)
      t += A(j,i) * u[j];
    v[i] = t;
  }
}

void AtAu(u, v, w) {
  Au(u,w);
  Atu(w,v);
}
```

```dart
num spectralnorm(n) {
  var i,
      u= new List(n),
      v= new List(n),
      w= new List(n),
      vv=0, vBv=0;
  for (i=0; i<n; i++) {
    u[i] = 1; v[i] = w[i] = 0;
  }
  for (i=0; i<10; i++) {
    AtAu(u,v,w);
    AtAu(v,u,w);
  }
  for (i=0; i<n; i++) {
    vBv += u[i]*v[i];
    vv  += v[i]*v[i];
  }
  return sqrt(vBv/vv);
}

void main() {
  print(spectralnorm(5500));
}
```

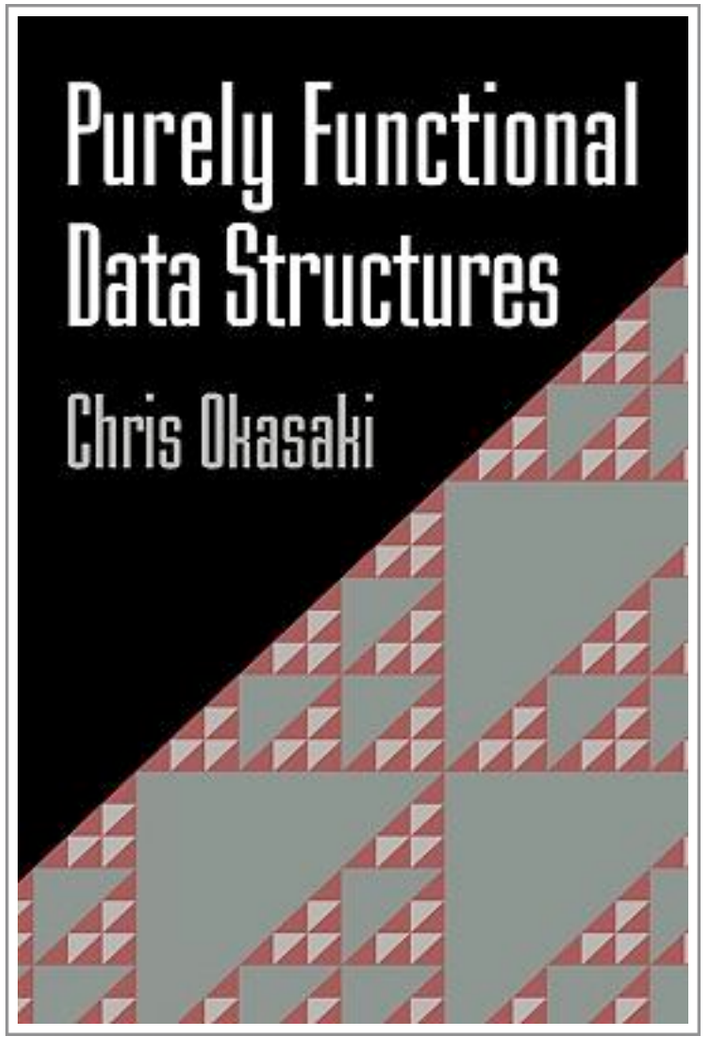
demo

```
'(1 2 3)
```

```
[1 2 3]
```

```
{:foo 1 :bar 2 :baz 3}
```

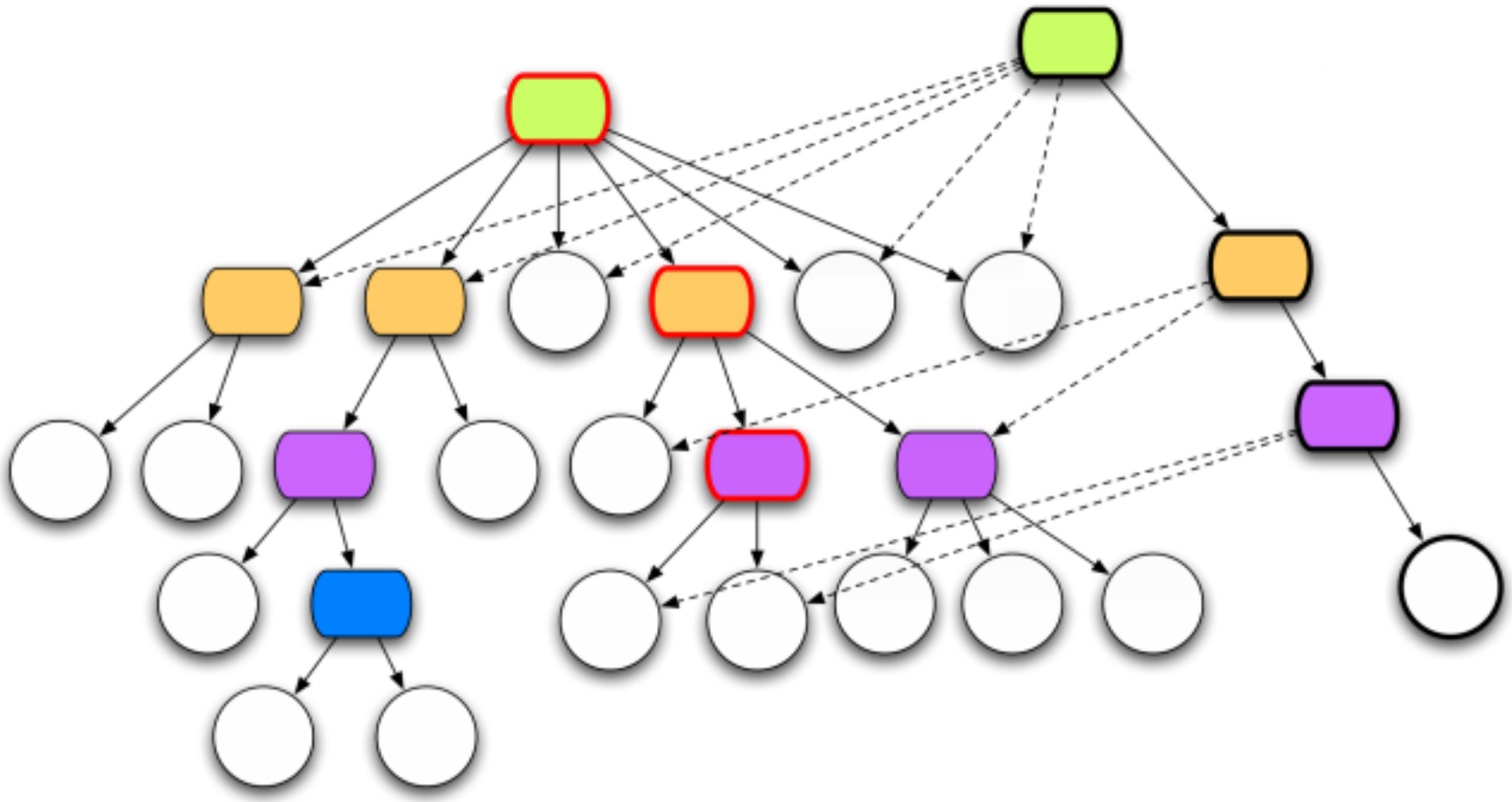
```
#{1 2 3}
```

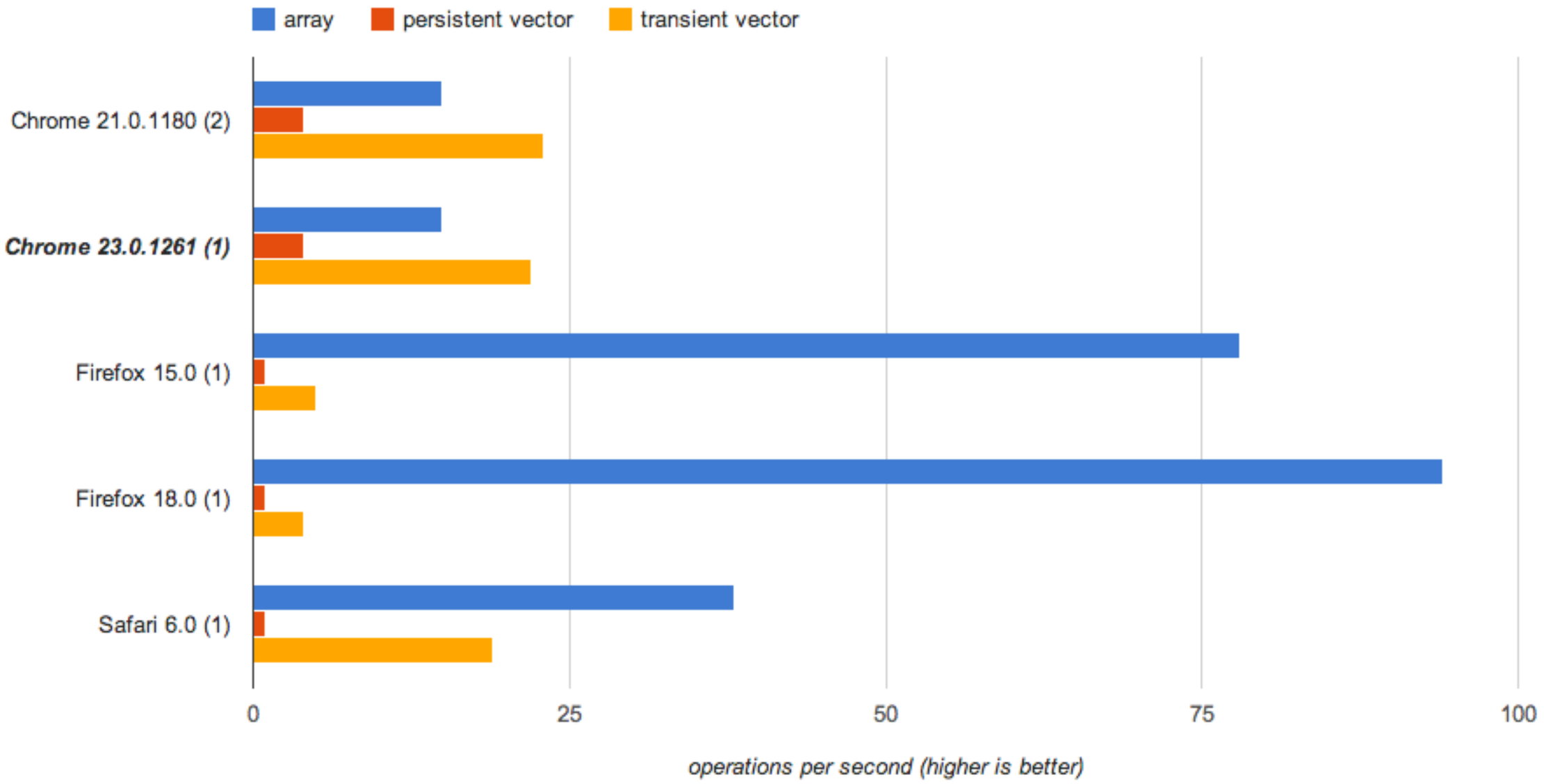

Purely Functional
Data Structures
Chris Okasaki

```
var x = 1 + 1;
```

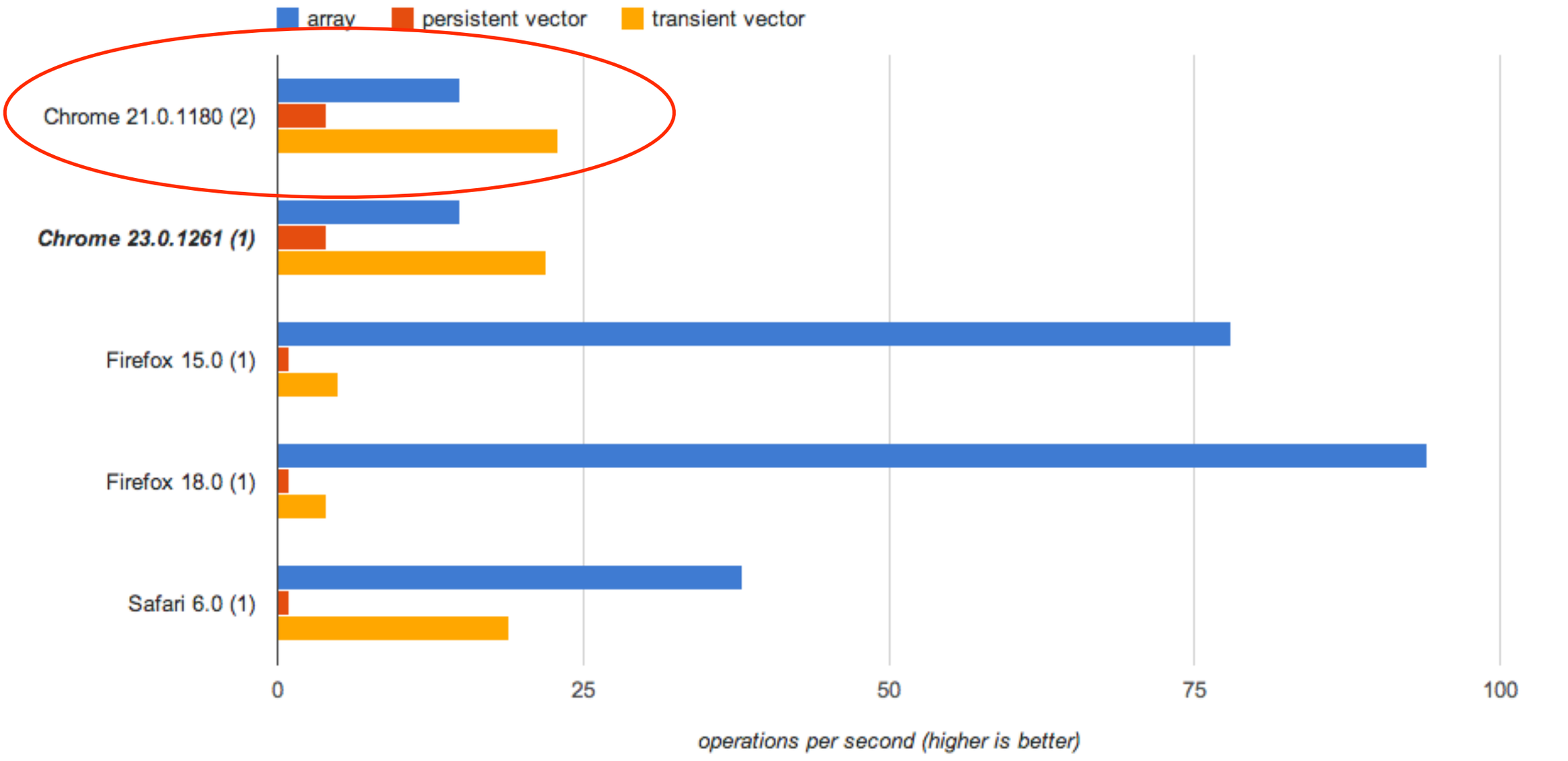
```
var x = [];
var y = x.push(1);
// ...
```

```
(let [x []
      y (conj x 1)]
  ; ...
)
```

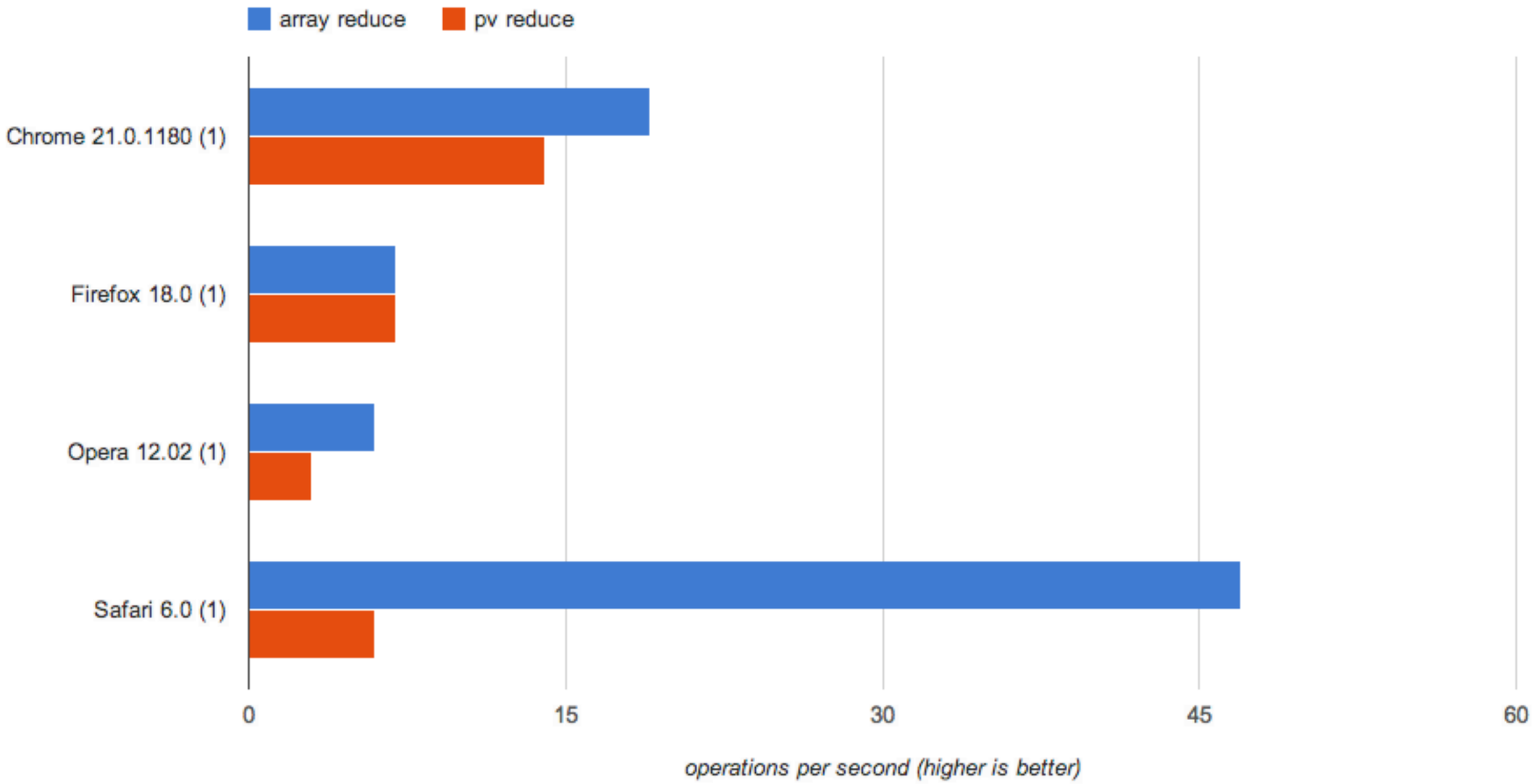




Browserscope thinks you are using **Chrome 23.0.1261** No?



Browserscope thinks you are using **Chrome 23.0.1261** No?



Browserscope thinks you are using **Chrome 21.0.1180** No?

Testing in Chrome 21.0.1180.89 on Mac OS X 10.7.4

Test		Ops/sec

array push	`var a = [];`
	`for(var i = 0; i < 1000; i++) {`
	`  a.push(i);`
	`}`
		140,531
		±6.25%
		fastest

local storage	`for(var i = 0; i < 1000; i++) {`
	`  localStorage.setItem("foo"+i, i);`
	`}`
		44.41
		±2.35%
		100% slower

jQuery	`for(var i = 0; i < 1000; i++) {`
	`  var divs = jQuery("#runner");`
	`}`
		856
		±1.79%
		99% slower

164X slower

Testing in Chrome 21.0.1180.89 on Mac OS X 10.7.4		
Test		**Ops/sec**
array	`pd_bench.core.array_build()`	15.71 ±5.14% 34% slower
persistent vector	`pd_bench.core.pv_build()`	4.62 ±3.81% 80% slower
transient vector	`pd_bench.core.tpv_build()`	23.70 ±4.94% fastest

getElementById	`for(var i = 0; i < 1000000; i++) {`
	`  var el = document.getElementById("runner");`
	`}`
		9.03
		±1.99%
		61% slower

```clojure
(first '(1 2 3))                    ; => 1

(first [1 2 3])                     ; => 1

(first {:foo 1 :bar 2 :baz 3})      ; => [:bar 2]

(first #{1 2 3})                    ; => 1

(first "foo")                       ; => "f"
```

```
function Foo() {
  // ...
}
Foo.prototype.bar = function() {
  // ...
}

function Baz() {
  // ...
}
Baz.prototype.bar = function() {
  // ...
}

function woz(x) {
  x.bar();
}
```

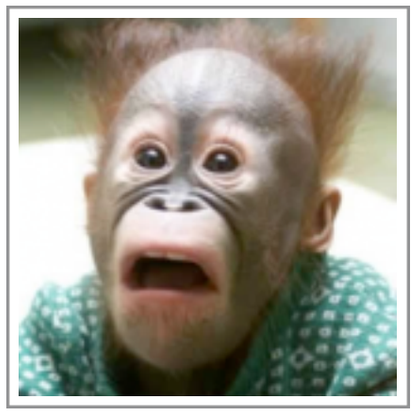
javascript polymorphism

this process is less informal

```
(defprotocol ISeq
  (-first [coll])
  (-rest [coll]))
```

Expression problem

great feature
but the cost is high



```
String.prototype.myCoolExtension = function() {
   ...
}
```

```
(extend-type string
  IMyCoolExtension
  (-awesome [this]
    ; ...
    ))
```

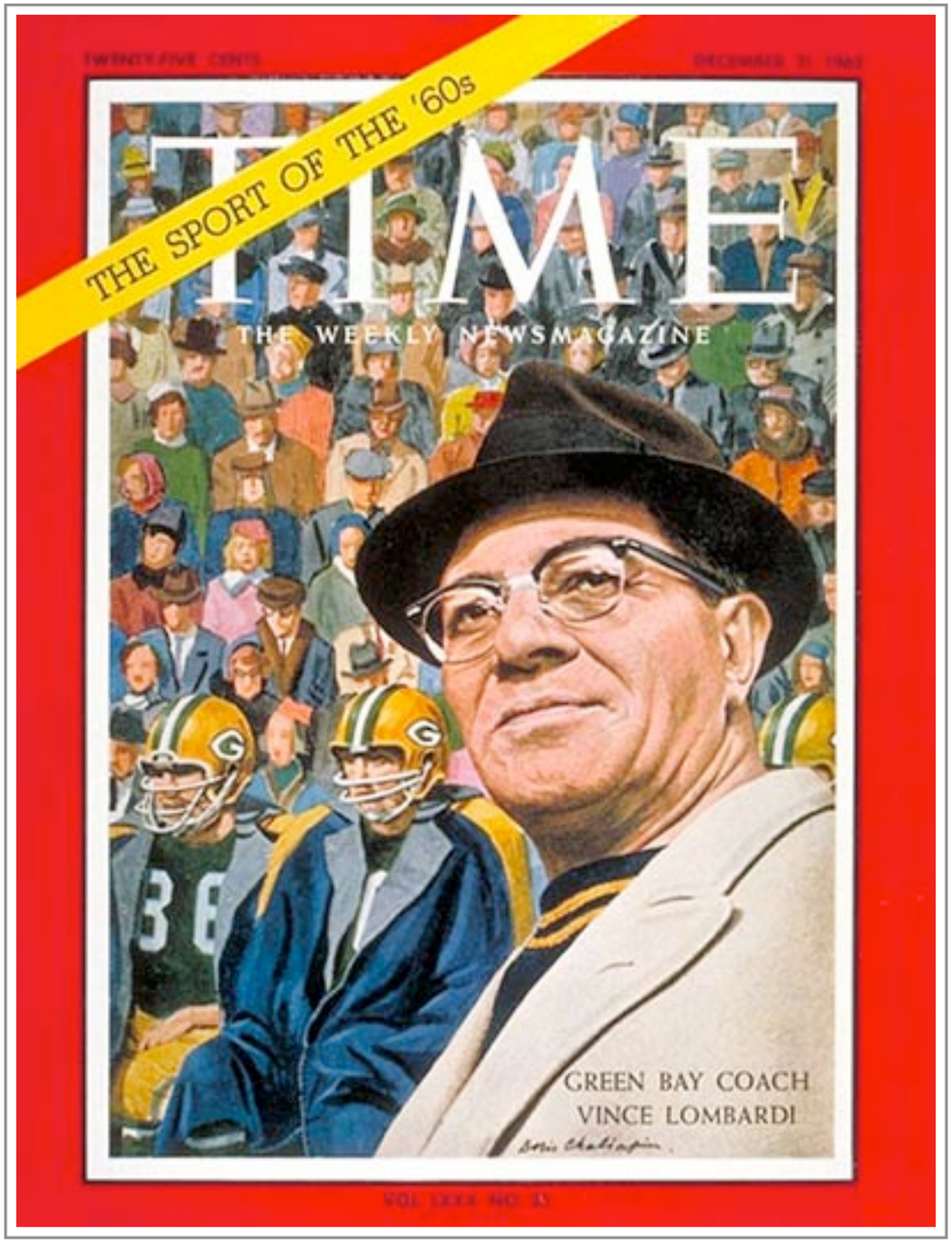
```
(extend-type default
  IMyCoolExtension
  (-awesome [this]
    ; ...
    ))
```

```
(extend-type default
  IHash
  (-hash [this]
    ; ...
    ))
```

Browser REPL

demo

MACROS



December 17, 1962

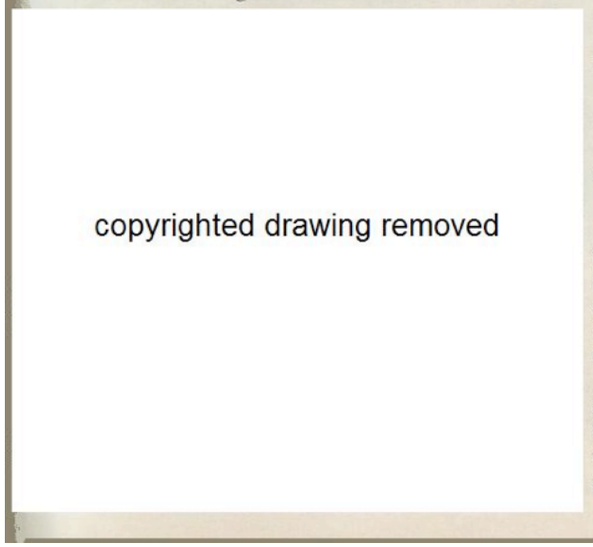
A PROBLEM FOR THE LOGICAL

Who Owns the Zebra?

In New York's posh Madison Avenue bars, stranger accosts stranger with a mimeographed sheet of paper and the question "Have you seen this?" In university dormitories, the problem is tacked to the doors. In suburban households in Westchester, Long Island and Connecticut, the ring of the telephone is likely to herald a voice that asks: "Is it the Norwegian?" The cause of the excitement is the brain-teaser reproduced on this page, with illustration provided by Steve Cook. The facts essential to solving the problem—which can indeed be solved by combining deduction, analysis and sheer persistence—are as follows:

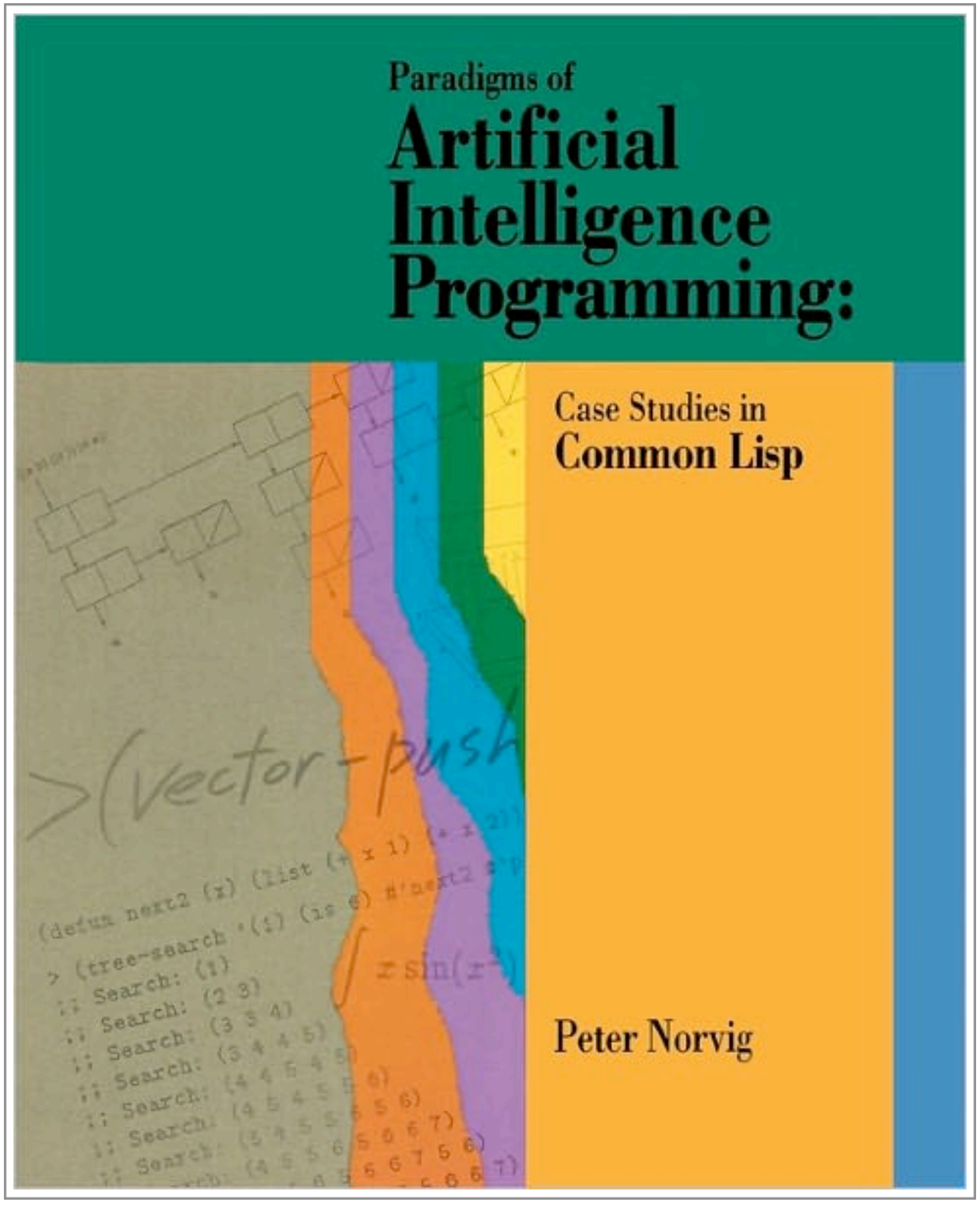
1. There are five houses.
2. The Englishman lives in the red house.
3. The Spaniard owns the dog.
4. Coffee is drunk in the green house.
5. The Ukrainian drinks tea.
6. The green house is immediately to the right of the ivory house.
7. The Old Gold smoker owns snails.
8. Kools are smoked in the yellow house.
9. Milk is drunk in the middle house.
10. The Norwegian lives in the first house.
11. The man who smokes Chesterfields lives in the house next to the man with the fox.
12. Kools are smoked in the house next to the house where the horse is kept.
13. The Lucky Strike smoker drinks orange juice.
14. The Japanese smokes Parliaments.
15. The Norwegian lives next to the blue house.

Now, who drinks water? Who owns the zebra?



In the interest of clarity, it must be added that each of the five houses is painted a different color, and their inhabitants are of different national extractions, own different pets, drink different beverages and smoke different brands of American cigarets. One other thing: In Statement 6, *right* means *your* right.

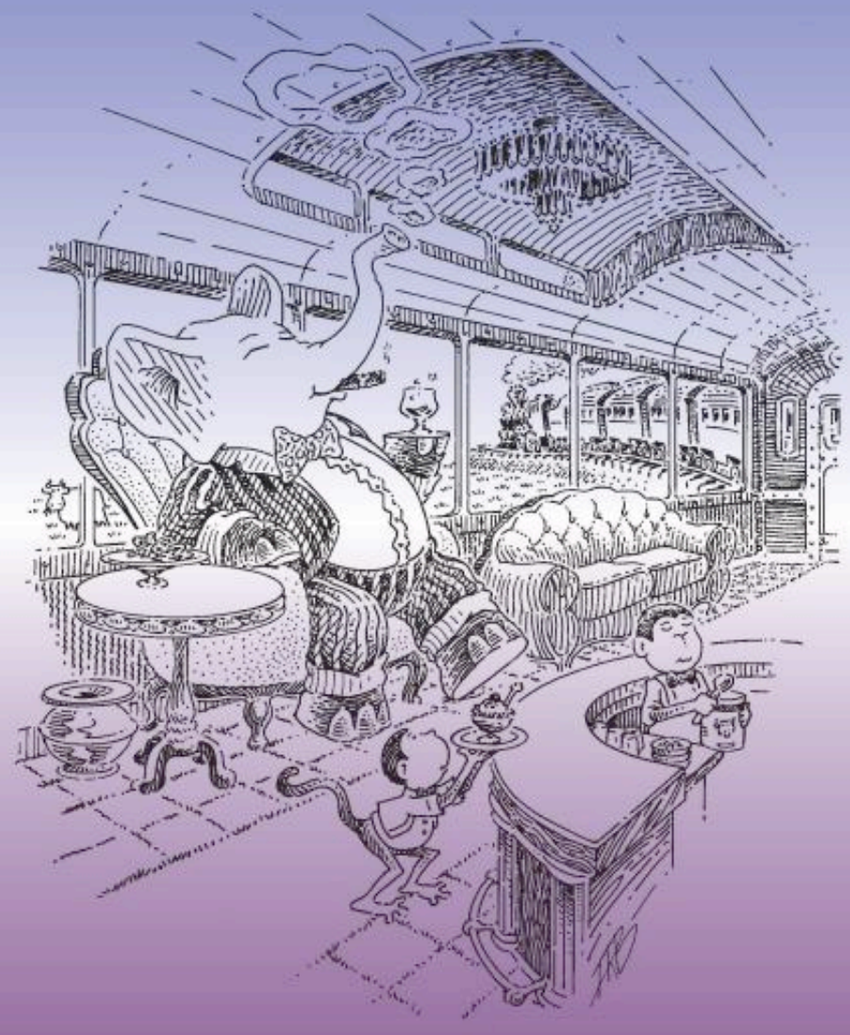
LIFE International will be glad to receive answers from its readers and will publish one or more of those which best combine, in the editors' judgment, the proper solution with brevity and clarity in expounding the logic by which the solution was reached. No intuitive answers, please.


Paradigms of
Artificial Intelligence Programming:
Case Studies in Common Lisp
Peter Norvig

As Peter Norvig points out,

24 billion candidate solutions

The Reasoned Schemer



Daniel P. Friedman, William E. Byrd,
and Oleg Kiselyov

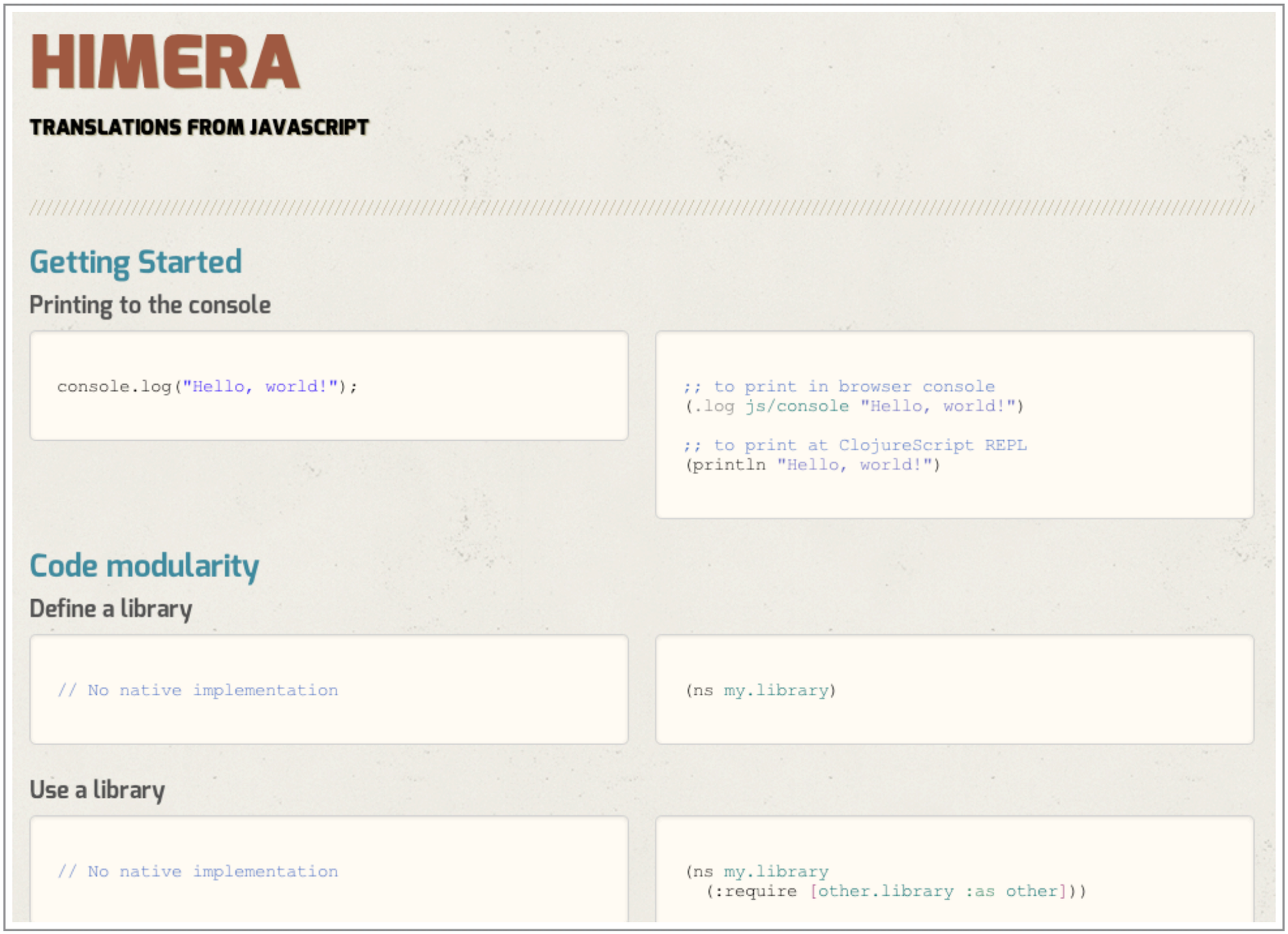
```
(defn zebrao [hs]
  (mu/symbol-macrolet [_ (lvar)]
   (m/all
    (m/== [_ _ [_ _ 'milk _ _] _ _] hs)
    (firsto hs ['norwegian _ _ _ _])
    (nexto ['norwegian _ _ _ _] [_ _ _ _ 'blue] hs)
    (righto [_ _ _ _ 'ivory] [_ _ _ _ 'green] hs)
    (membero ['englishman _ _ _ 'red] hs)
    (membero [_ 'kools _ _ 'yellow] hs)
    (membero ['spaniard _ _ 'dog _] hs)
    (membero [_ _ 'coffee _ 'green] hs)
    (membero ['ukrainian _ 'tea _ _] hs)
    (membero [_ 'lucky-strikes 'oj _ _] hs)
    (membero ['japanese 'parliaments _ _ _] hs)
    (membero [_ 'oldgolds _ 'snails _] hs)
    (nexto [_ _ _ 'horse _] [_ 'kools _ _ _] hs)
    (nexto [_ _ _ 'fox _] [_ 'chesterfields _ _ _] hs))))
```

demo

```
(ns cljs-stl.zebra.demo
  (:require-macros [cljs.core.logic.macros :as m]
                   [clojure.tools.macro :as mu])
  (:use [cljs.core.logic :only
         [firsto membero lvar *occurs-check*]]))

(defn nexto [x y l]
  (m/conde
    [(righto x y l)]
    [(righto y x l)]))

(defn zebrao [hs]
  (mu/symbol-macrolet [_ (lvar)]
   (m/all
    (m/== [_ _ [_ _ 'milk _ _] _ _] hs)
    (firsto hs ['norwegian _ _ _ _])
    (nexto ['norwegian _ _ _ _] [_ _ _ _ 'blue] hs)
    (righto [_ _ _ _ 'ivory] [_ _ _ _ 'green] hs)
    (membero ['englishman _ _ _ 'red] hs)
    (membero [_ 'kools _ _ 'yellow] hs)
    (membero ['spaniard _ _ 'dog _] hs)
    (membero [_ _ 'coffee _ 'green] hs)
    (membero ['ukrainian _ 'tea _ _] hs)
    (membero [_ 'lucky-strikes 'oj _ _] hs)
    (membero ['japanese 'parliaments _ _ _] hs)
    (membero [_ 'oldgolds _ 'snails _] hs)
    (nexto [_ _ _ 'horse _] [_ 'kools _ _ _] hs)
    (nexto [_ _ _ 'fox _] [_ 'chesterfields _ _ _] hs))))

(.log js/console
  (binding [*occurs-check* false]
    (m/run 1 [q] (zebrao q))))
```

```
function who_owns_the_fish() {
    var NATIONALITIES   = [ "British", "Swedish", "Danish", "Norwegian", "German" ];
    var BEVERAGES       = [ "tea", "milk", "coffee", "beer", "water" ];
    var TOBACCO_BRANDS  = [ "pallmall", "dunhill", "marlboro", "winfield", "rothmans" ];
    var PETS            = [ "dogs", "cats", "horses", "birds", "fish" ];
    var COLORS          = [ "red", "green", "white", "yellow", "blue" ];

    function assert(cond) { if (!cond) amb.fail() }
    function iff(c1, c2) { if (c1 != c2) amb.fail() }
    function amb_all(values, program){
        amb(values, program);
        amb.fail();
    }
    function find_house(houses, type, val) {
        for (var i = houses.length; --i >= 0;)
            if (houses[i][type] == val)
                return i;
    }
    function neighbors(houses, t1, v1, t2, v2) {
        var h1 = find_house(houses, t1, v1), h2 = find_house(houses, t2, v2);
        assert(Math.abs(h1 - h2) == 1);
    }

    function add_house(houses){
        var this_house = houses.length;
        function other(values, type) {
            var a = [];
            for (var i = 0; i < values.length; ++i)
                if (find_house(houses, type, values[i]) == null)
                    a.push(values[i]);
            return a;
        }
        amb(other(NATIONALITIES, "nat"), function(nat){
            amb_all(other(COLORS, "col"), function(col){
                amb_all(other(PETS, "pet"), function(pet){
                    amb_all(other(BEVERAGES, "bev"), function(bev){
                        amb_all(other(TOBACCO_BRANDS, "tob"), function(tob){
                            iff( nat == "British", col == "red" );
                            iff( nat == "Swedish", pet == "dogs" );
                            iff( nat == "Danish", bev == "tea" );
                            iff( col == "white",
                                 this_house > 0
                                 && houses[this_house - 1].col == "green" );
                            iff( col == "green", bev == "coffee" );
                            iff( tob == "pallmall", pet == "birds" );
                            iff( col == "yellow", tob == "dunhill" );
                            iff( this_house == 2, bev == "milk" );
                            iff( this_house == 0, nat == "Norwegian" );
                            iff( tob == "winfield", bev == "beer" );
                            iff( nat == "German", tob == "rothmans" );

                            var h = { nat: nat, bev: bev, tob: tob, pet: pet, col: col };
                            var a = houses.concat([ h ]);

                            if (a.length == 5) {
                                neighbors(a, "tob", "marlboro", "pet", "cats");
                                neighbors(a, "pet", "horses", "tob", "dunhill");
                                neighbors(a, "nat", "Norwegian", "col", "blue");
                                neighbors(a, "tob", "marlboro", "bev", "water");

                                // if we get thus far, we have a solution!
                                console.log(a);
                            } else {
                                add_house(a);
                            }
                        })})})})}); // now I dare you, tell me that lisp syntax is ugly.
    };
    add_house([]);
}
```

http://himera.herokuapp.com/synonym.html



HIMERA

TRANSLATIONS FROM JAVASCRIPT

Getting Started

Printing to the console

```
console.log("Hello, world!");
```

```
;; to print in browser console
(.log js/console "Hello, world!")

;; to print at ClojureScript REPL
(println "Hello, world!")
```

Code modularity

Define a library

```
// No native implementation
```

```
(ns my.library)
```

Use a library

```
// No native implementation
```

```
(ns my.library
  (:require [other.library :as other]))
```

Costs!

- JS arithmetic underneath, not Clojure numerics

- JS arithmetic underneath, not Clojure numerics
- Debugging

- JS arithmetic underneath, not Clojure numerics
- Debugging
- Not bootstrapped - requires Clojure

- JS arithmetic underneath, not Clojure numerics
- Debugging
- Not bootstrapped - requires Clojure
- Community (not corporate) development

- performance enhancements are not evenly distributed (keywords, multimethods)

- performance enhancements are not evenly distributed (keywords, multimethods)
- a lot of generated code for trivial programs

- performance enhancements are not evenly distributed (keywords, multimethods)
- a lot of generated code for trivial programs
- small details present in Clojure are missing in ClojureScript - they add up

```
http://cloc.sourceforge.net v 1.55  T=1.0 s (25.0 files/s, 14367.0 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
ClojureScript                   14           1455            217           7677
Clojure                         10            570            197           4249
Javascript                       1              0              0              2
-------------------------------------------------------------------------------
SUM:                            25           2025            414          11928
-------------------------------------------------------------------------------
~/development/clojure/clojurescript(master) $
```

Thank you!

Questions?