"the prevalent theme is programming languages,

while this kind of work can sometimes lead somewhere,

projects mostly fail while promising grandiose general results"

-- *anonymous reviewer*

JavaScript Programmers Hate You

orthodoxy

Program have types

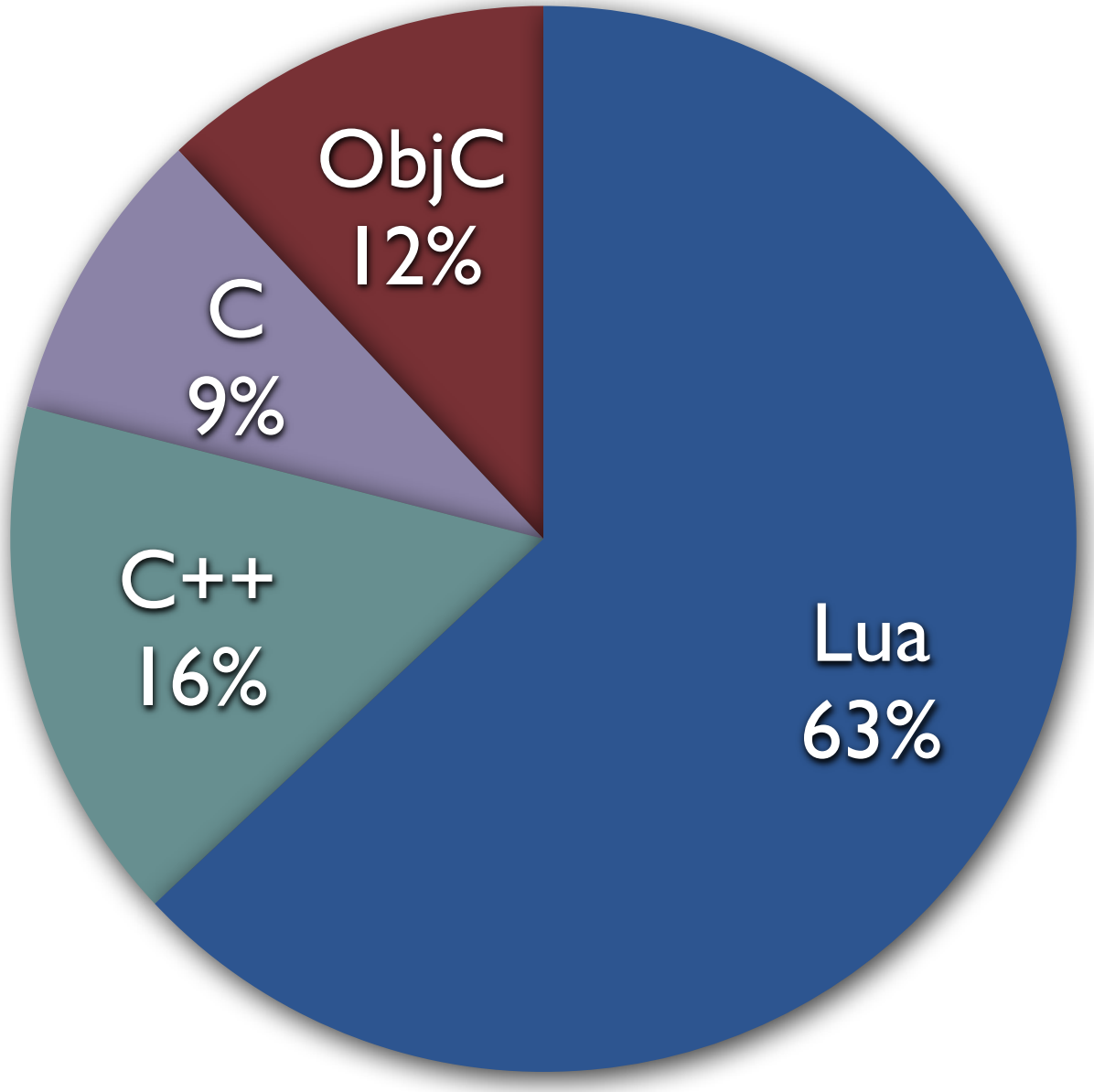
Types increase productivity

Static is better

Correctness is the goal

Decades of dynamism

VB	dyn	1991
Python	dyn	1991
Lua	dyn	1993
R	dyn	1993
Java	stat+dyn	1995
JavaScript	dyn	1995
PHP	dyn	1995
Ruby	dyn	1995
Clojure	dyn	2007



Troy Gaul. **Lightroom Exposed**. `http://www.troygaul.com`

Premiepensionsmyndigheten's Pluto

"This system is behind the Swedish Premium Pension. It automatically invests and manages 220 billion SEK across 5 million accounts."

Pluto = **320 000 lines of Perl**

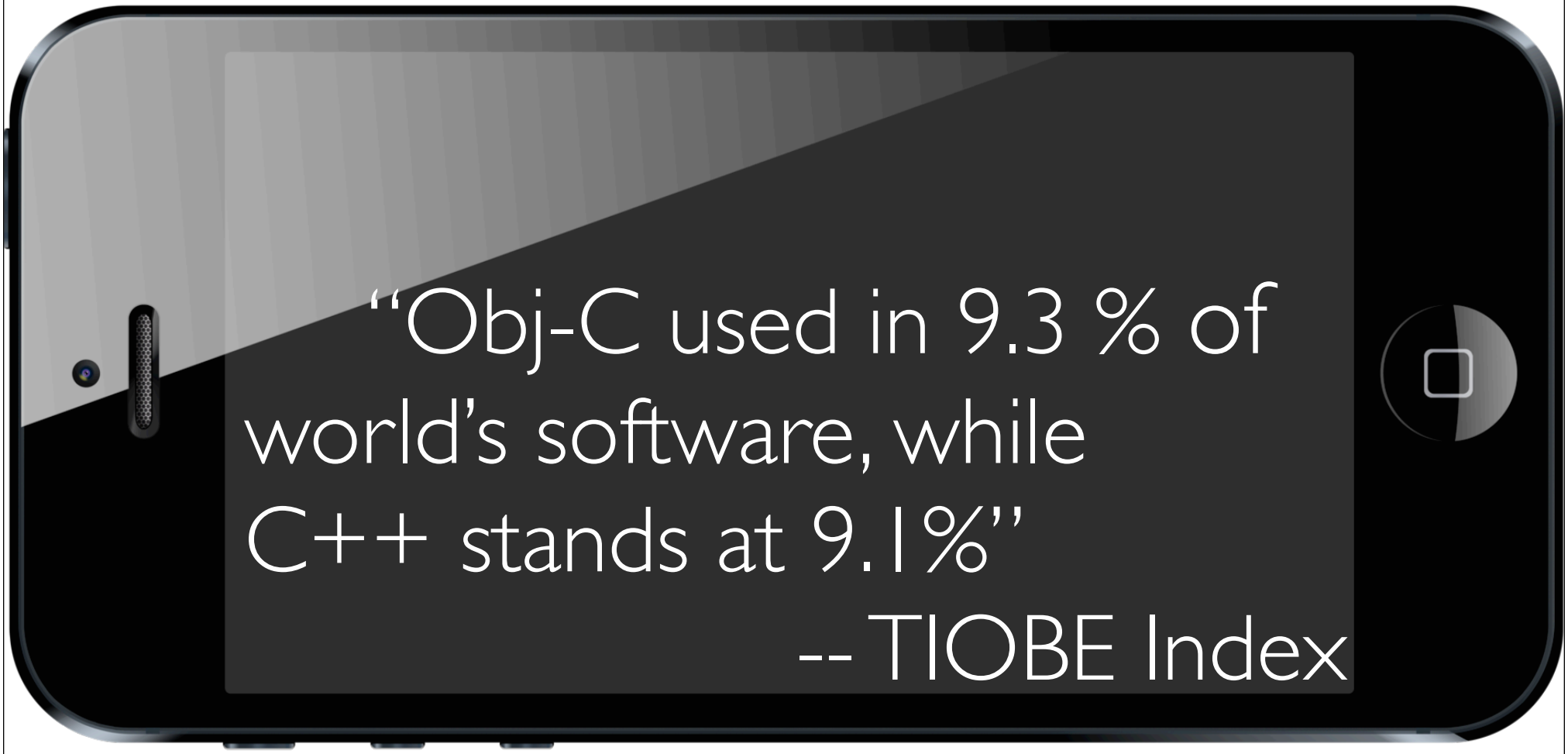
Lundborg, Lemonnier. PPM or how a system written in Perl can juggle with billions. Freenix 2006

Premiepensionsmyndigheten's Pluto

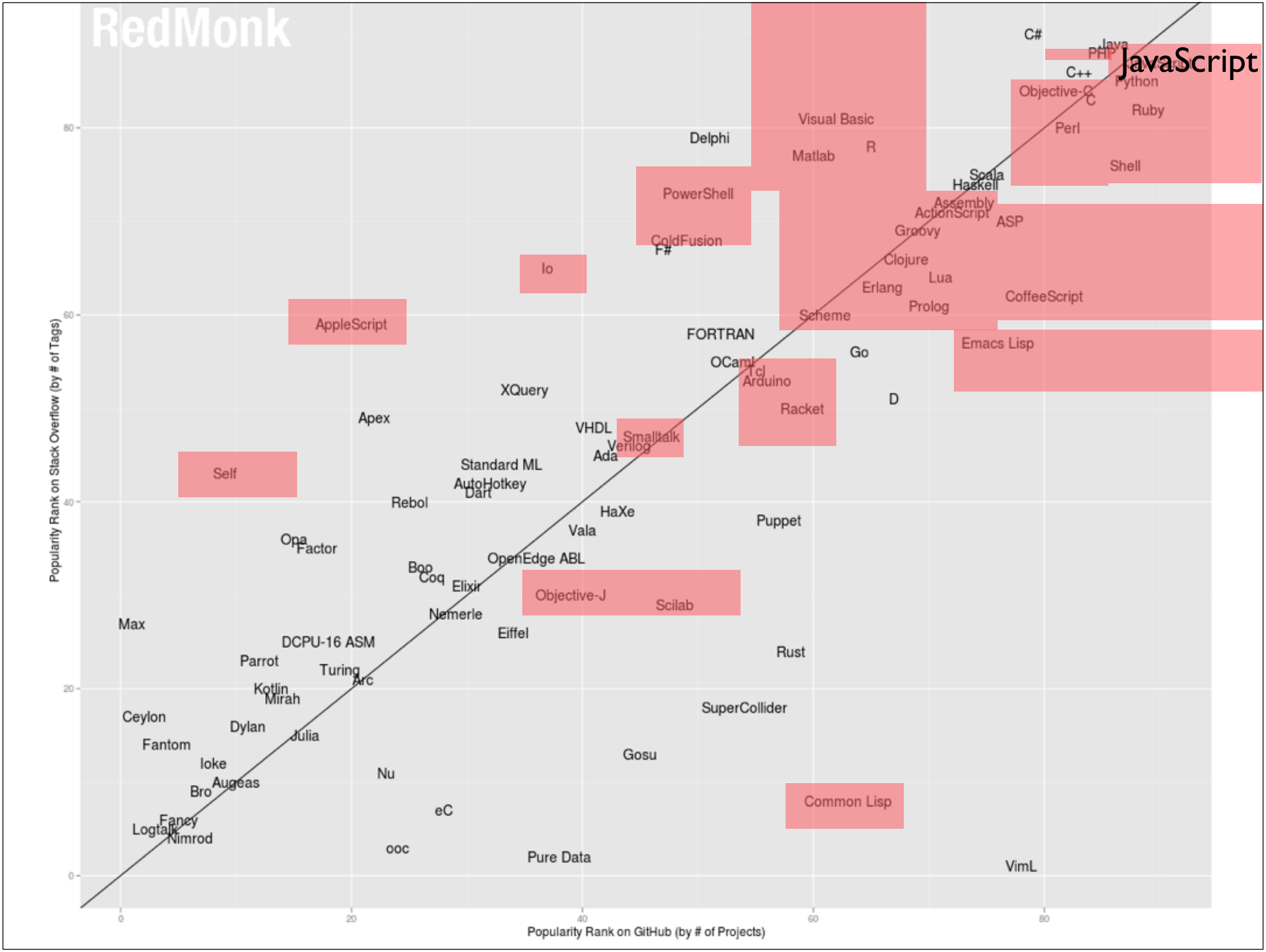
```
contract('do_sell_current_holdings')
  -> in(&is_person, &is_date)
  -> out(&is_state)
  -> enable;

sub do_sell_current_holdings {
   my ($person, $date)   …
   if ($operation eq "BUD_") {…}
   return $state;
}
```

Lemonnier. Testing Large Software With Perl. Nordic Perl Workshop 2007

Objective C


"Obj-C used in 9.3 % of world's software, while C++ stands at 9.1%"
-- TIOBE Index

Objective C


RedMonk
JavaScript
C#
Java
PHP
C++
Python
Objective-C
C
Ruby
Perl
Shell
Visual Basic
Delphi
R
Matlab
Scala
Haskell
PowerShell
Assembly
ActionScript
ASP
Groovy
ColdFusion
F#
Clojure
Io
Lua
Erlang
CoffeeScript
Prolog
AppleScript
Scheme
FORTRAN
Emacs Lisp
Go
OCaml
Tcl
Arduino
XQuery
D
Racket
Apex
VHDL
Smalltalk
Verilog
Ada
Self
Standard ML
AutoHotkey
Dart
Rebol
HaXe
Puppet
Vala
Ona
Factor
OpenEdge ABL
Boo
Coq
Elixir
Objective-J
Scilab
Nemerle
Max
Eiffel
DCPU-16 ASM
Rust
Parrot
Turing
Arc
Kotlin
Mirah
SuperCollider
Ceylon
Dylan
Julia
Fantom
Gosu
Ioke
Nu
Augeas
Bro
Common Lisp
eC
Fancy
Logtalk
Nimrod
ooc
Pure Data
VimL
Popularity Rank on Stack Overflow (by # of Tags)
Popularity Rank on GitHub (by # of Projects)
0
20
40
60
80

JavaScript Programmers Hate You

A Clash of World Views

Computer Science

Fixed programs, transient data

there will always be another input

Data Science

Fixed data, transient programs

there will always be another query

Programming for the masses

ML, Haskell, Scala, C++ are all domain specific languages

The "domain" is programming in the large by experts

Programming for the masses

Programming languages should be gateway drugs to computational thinking

Instead enforce a rigid programming discipline

Dynamic Typing

If static typing has benefits:

- preventing some errors ahead of time
- simplifying generation of efficient code
- providing machine-checked documentation

Why is it a bad idea?

Dynamic Typing

Static typing only catches trivial errors

most systems can't even catch NPEs, or off-by-one errors

Static typing ossifies code and hinders evolution

make the type checker *globally* happy before testing a *local* change

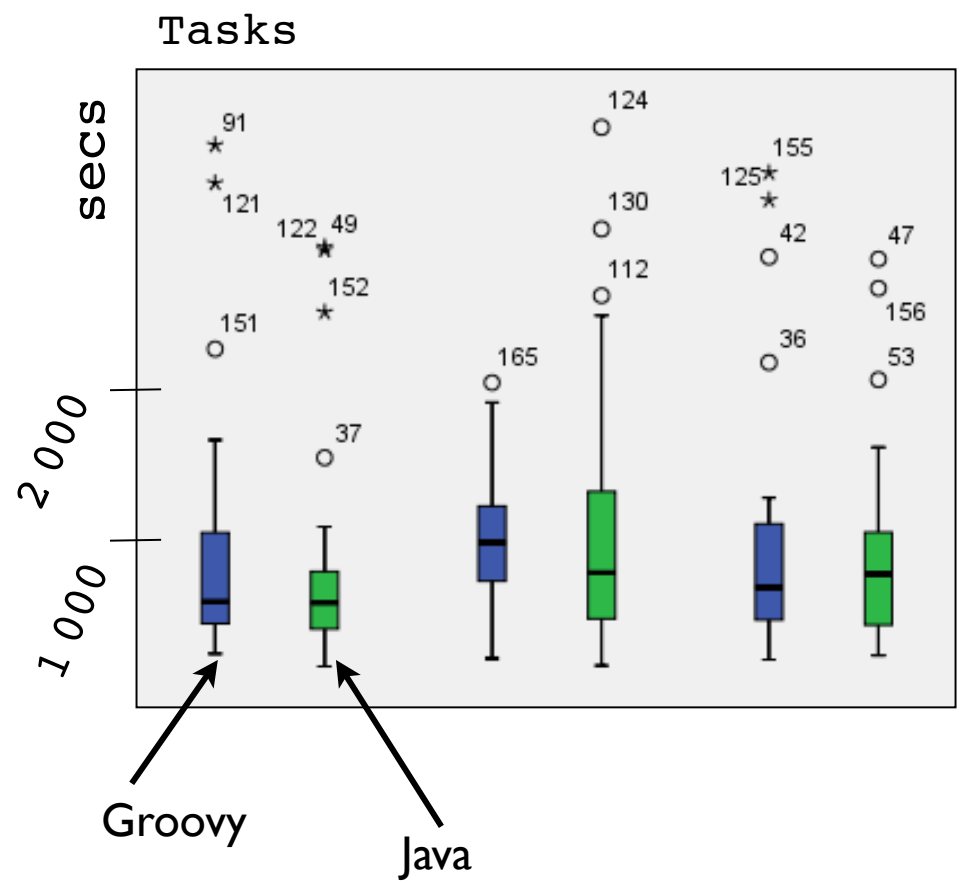
Static typing slows down the rate of development

pessimistic typing, in case of doubt just say *no*

Dynamic Typing

Hypothesis:

No difference in time solving semantic bugs with a dynamically or statically typed language



Steinberg. **What is the impact of static type systems on maintenance tasks?** *MSc Thesis U.Duisburg-Essen*

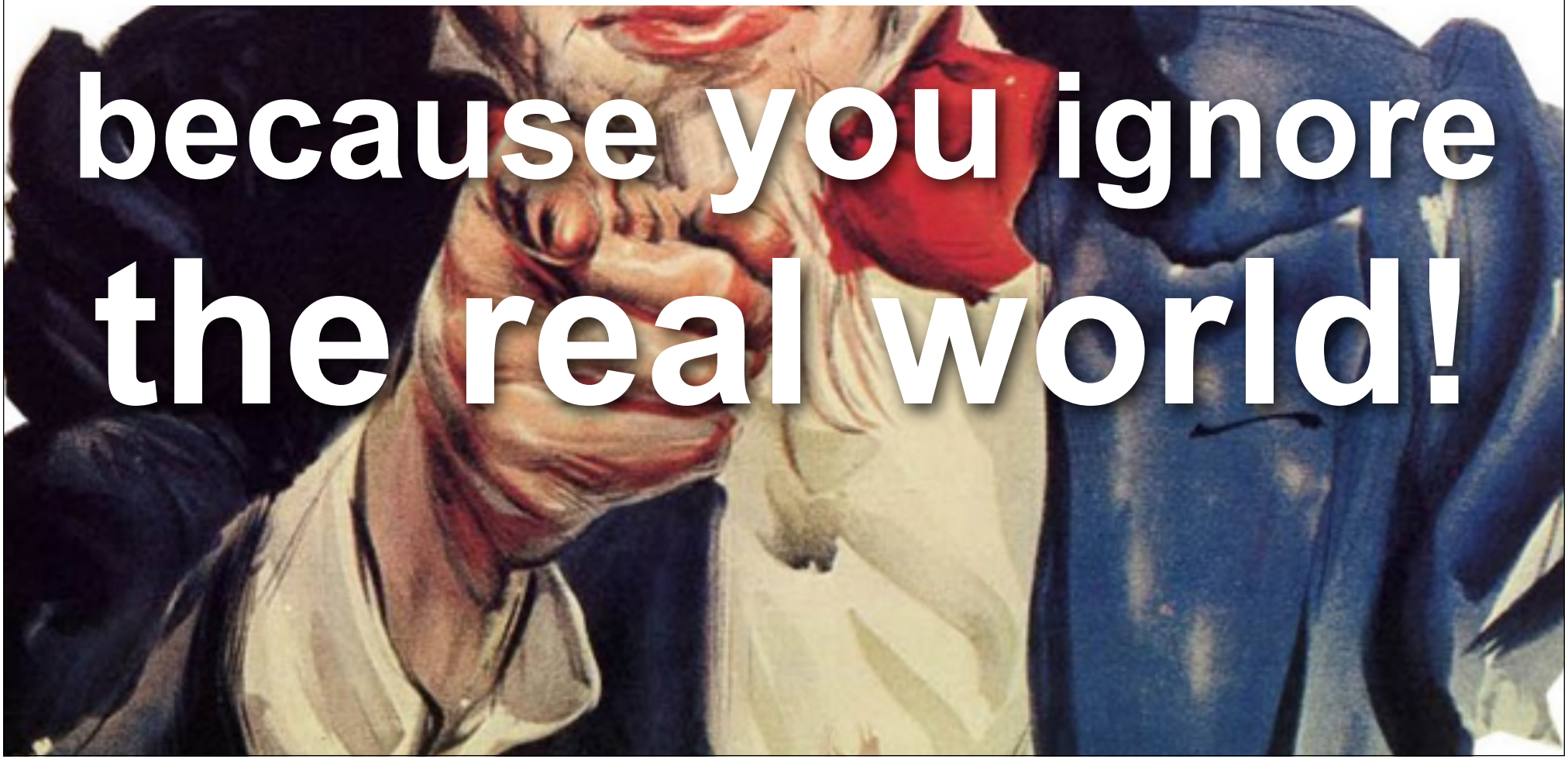
Programming for the masses

R learned in one lecture

Most PHP programmers never read a language manual

Design languages for all

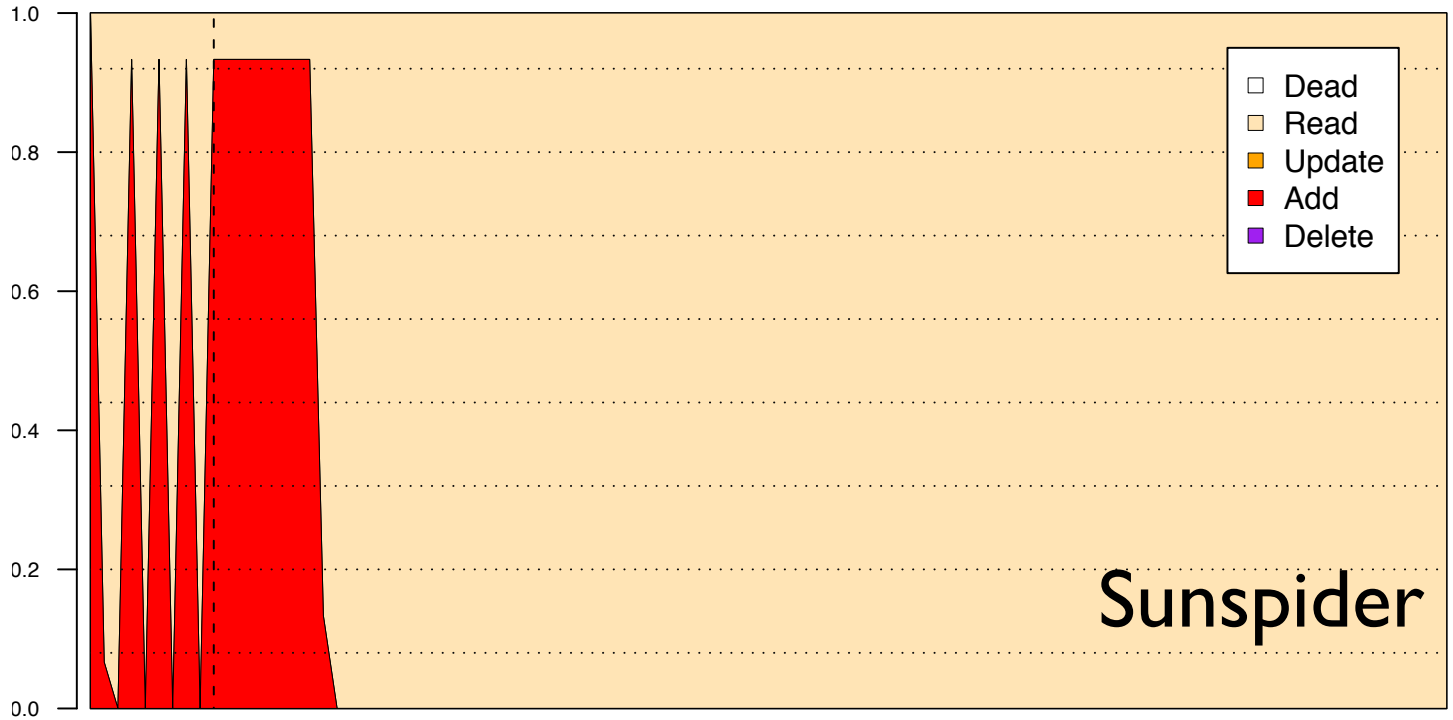
JavaScript Programmers Hate You

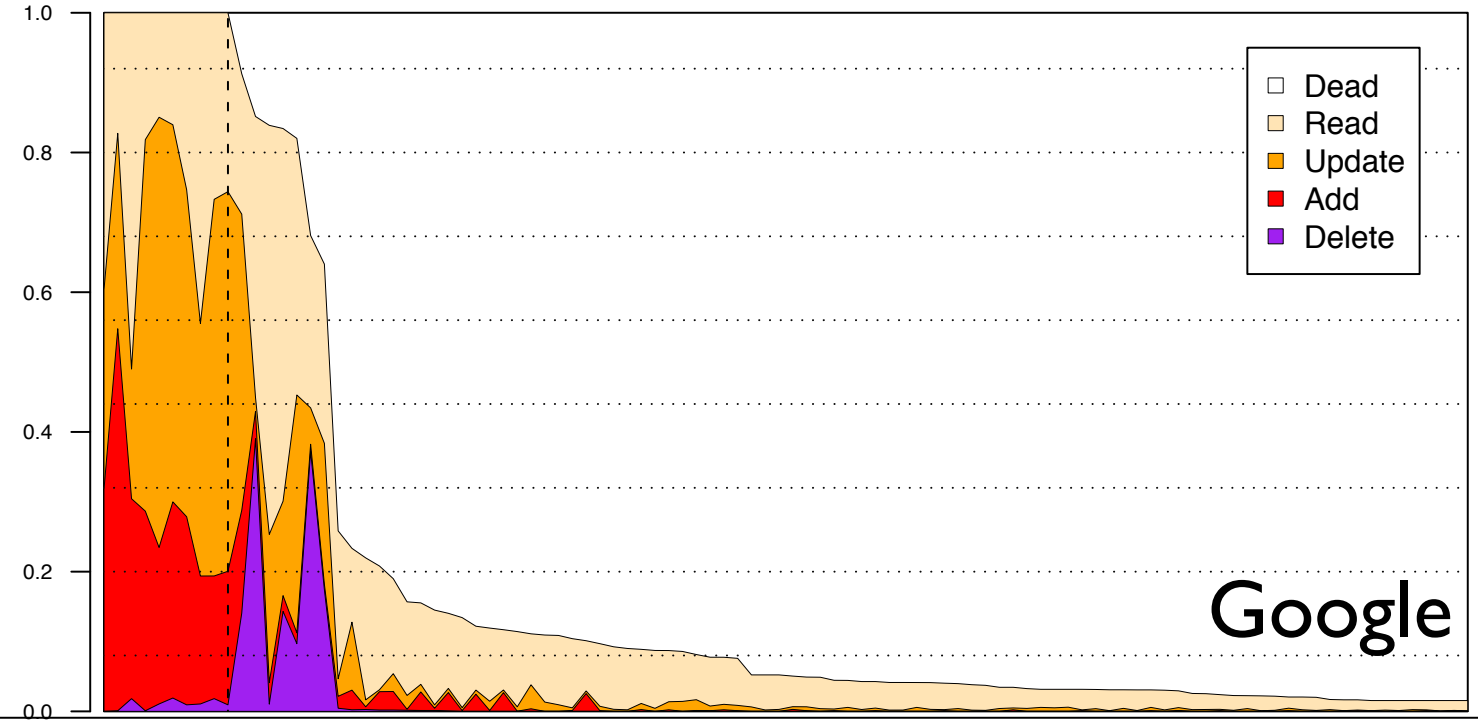


Programs as Data

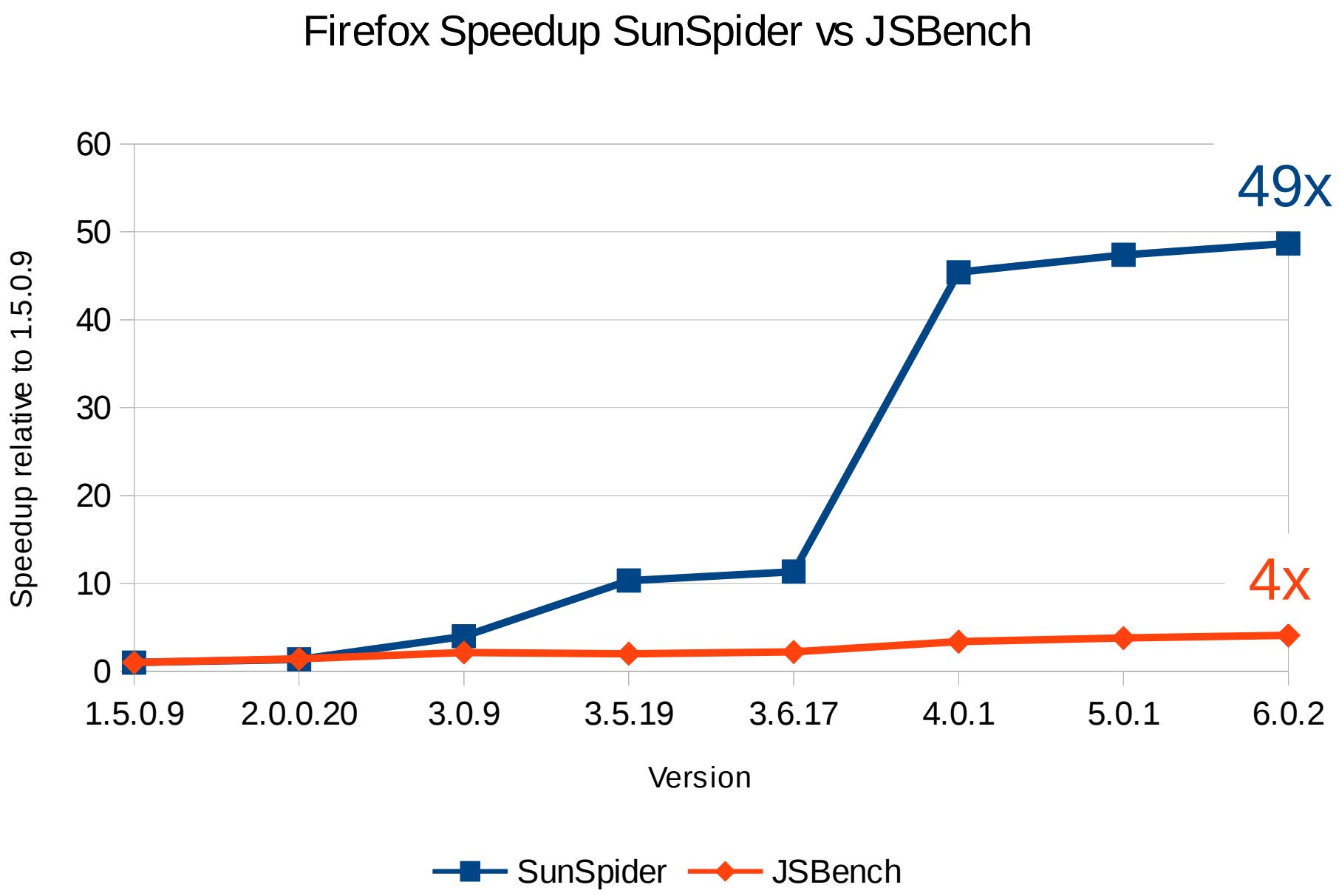
Programming language design should be informed by empirical studies of actual use

Design languages with the same attention to detail Apple pays the iPhone?





Richards, Lesbrene, Burg, Vitek. **An Analysis of the Dynamic Behavior of JavaScript Programs.** PLDI'10



Richards, Gal, Eich, Vitek. **JSBench: Automating the Construction of JavaScript Benchmarks.** OOPSLA'11

Programs to Data

The Ostrich posture does not make Reflection go away

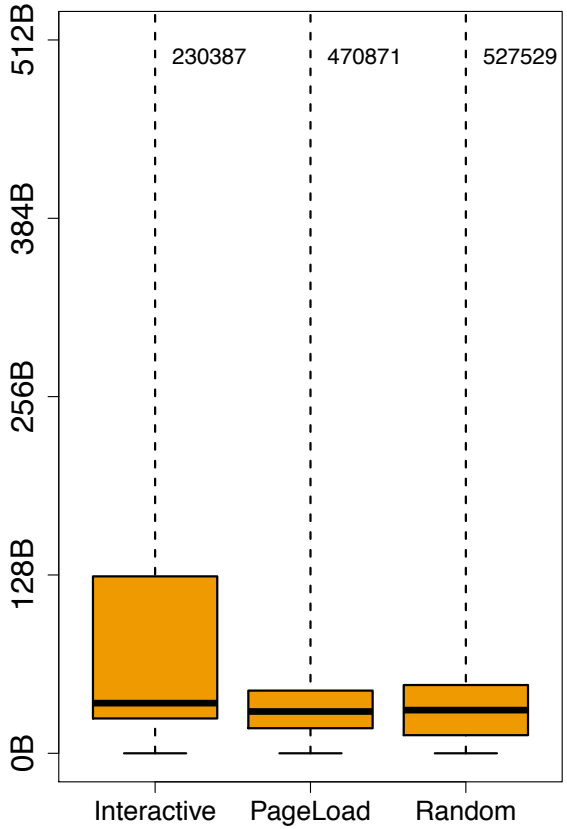
Accept that reflection is here to stay and deal with it

Eval Usage

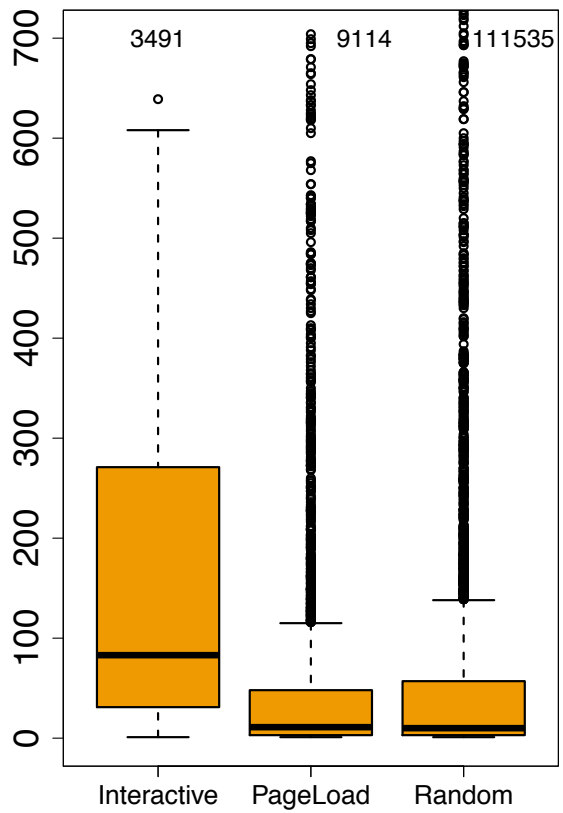
100% of top 100 sites use JavaScript

82% use eval!

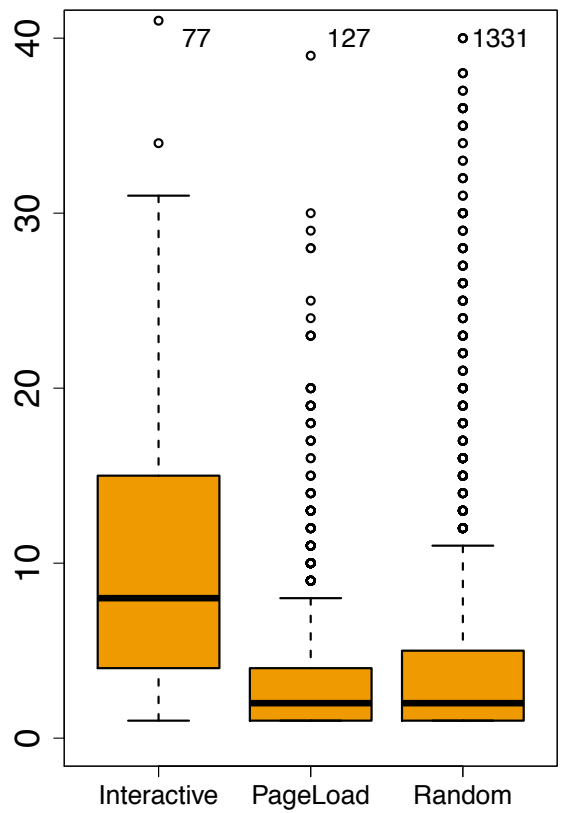
String Size



Calls



Call Sites

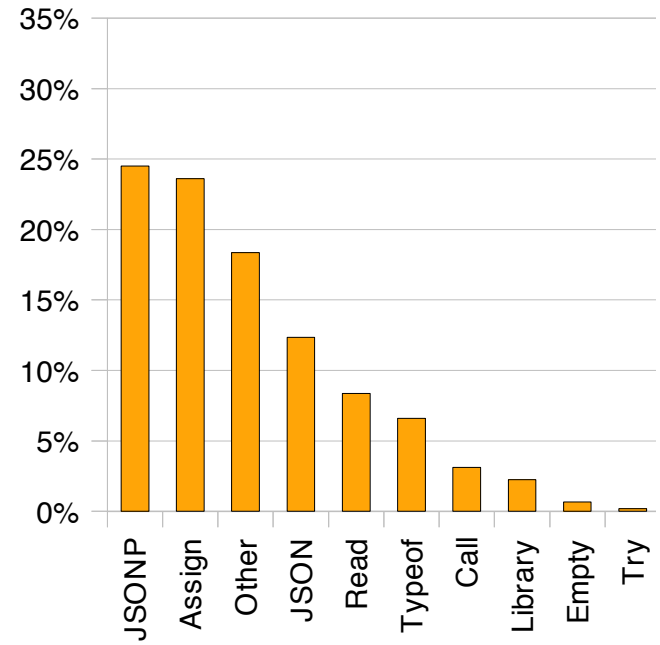


Richards, Hammer, Burg, Vitek. **The Eval that Men Do.** ECOOP 2011

The Shape of Eval

Identified common patterns:

JSON	eval('{"x": 2}')
JSONP	eval("f({x: 2})")
Read	eval("obj . f")
Assign	eval("id = x")
Typeof	eval('typeof('+x+')!="undefined"')
Try	eval('try{throw v=14}catch(e){}')
Call	eval('get("menu")')
Empty	



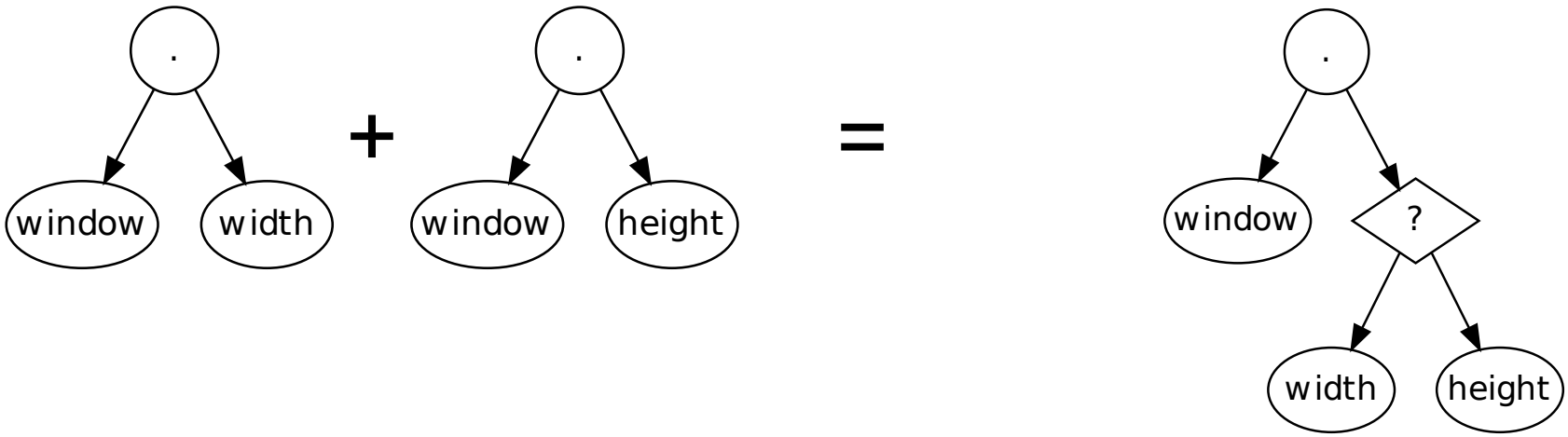
(a) INTERACTIVE

Patterns	1	2	3	4	5
Callsites	27553	303	92	3	1

Eval begone!

```
window.width = 10;
window.height = 20;

function getDimension(x){
  d = eval("window." + x);
}

getDimension("width");
getDimension("height");
```

```
d = (x == "width"
    ? window.width
    : window.height);
```



Meawad, Richards, Morandat, Vitek. **Eval Begone! : Semi-Automated Removal of Eval from JavaScript Programs** OOPSLA '12

JavaScript Programmers Hate You

Avoid non-Problems

Decades of research on Alias analysis for Java, Ownership types, and Information flow…

Noble, Potter, Vitek. **Flexible Alias Protection**, ECOOP 1998.

Programmer Productivity First

The metric that matters is time-to-solution

Late answers are wrong answers

Failure Obliviousness

Dynamic languages keep the program running…

… by execution of incomplete programs

… by converting data types automatically

… by swallowing errors

"Best effort", optimistic, execution

Failure Obliviousness

Getting an error in JavaScript is difficult

```
x = {};        // object
x.b = 42;      // field add
y = x["f"];    // undefined
z = y.f;       // error
```

Failure Obliviousness

- New JS VM that aborts untrusted JavaScript code on policy violation
- From program's point of view these are random failures
- Most programs are resilient and keep working

Policy	Functional	AdBlock	Partial	Broken
`AddOnly`	36	8	5	1
`SendAfterRead`	42	7	1	0

Hammer, Richards, Vitek. **Flexible Access Control Policies w. Delimited Histories & Revocation**, OOPSLA13

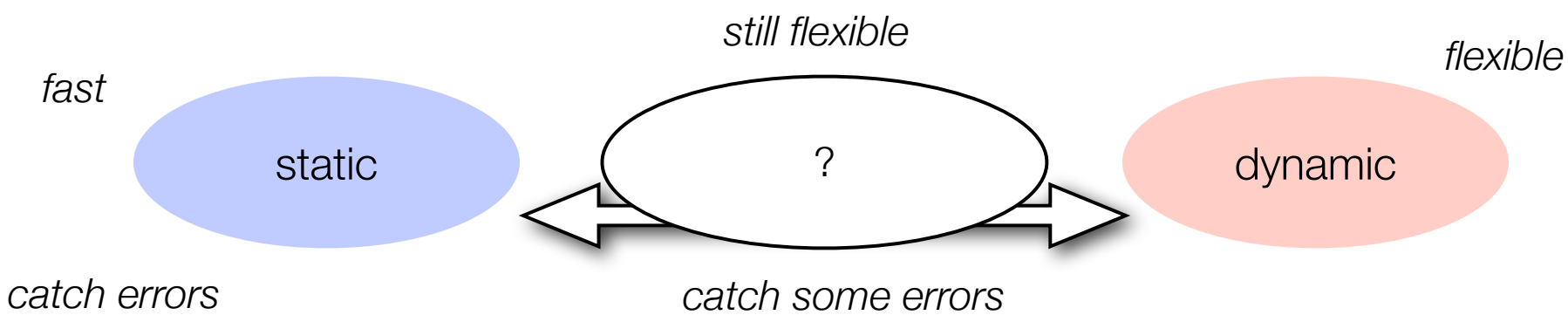
Programmer Productivity First

Don’t prove theorems because you know how but because you need to

What are the properties that really improve time to solution

Gradual Typing

Introduce a novel type construct that mediates between static and dynamic.



Wrigstad, Nardelli, Lebresne, Ostlund, Vitek. **Integrating Typed and Untyped Code in a Scripting Language** POPL10

Gradual Typing

```
def id(x              ) = x;


x = id( [“toobad”] )

x[0] + “!”
```

Wrigstad, Nardelli, Lebresne, Ostlund, Vitek. **Integrating Typed and Untyped Code in a Scripting Language** POPL10

Gradual Typing

```
def id(x : [Int] ) = x;

…  id( [42,24] )  …
```

Wrigstad, Nardelli, Lebresne, Ostlund, Vitek. **Integrating Typed and Untyped Code in a Scripting Language** POPL10

Gradual Typing

```
def id(x :like[Int]) = x;


x = id( [“toobad”] )

x[0] + “!”
```

Wrigstad, Nardelli, Lebresne, Ostlund, Vitek. **Integrating Typed and Untyped Code in a Scripting Language** POPL10

Gradual Typing

- In Thorn, a simple contract:
  - adding type annotations will never slow down the program
  - adding type annotations will not break a running program
  - type system is "always on"

Bloom, Field, Nystrom, Ostlund, Richards, Strnisa, Vitek, Wrigstad.
Thorn-Robust, Concurrent, Extensible Scripting on the JVM. OOPSLA'12

Programmer Productivity First

Programming languages matter much less then we like to admit

Millions of lines of PHP at Facebook, Sweden's pluto is in Perl

Variable lookup in R

$$\frac{\mathsf{getfun}(H, \Gamma, \mathtt{x}) = \delta}{\mathbb{C}[\mathtt{x}(\overline{\mathtt{a}})]\ \Gamma * S; H \implies \delta\ \Gamma * \mathbb{C}[\mathtt{x}(\overline{\mathtt{a}})]\ \Gamma * S; H} \quad [\textsc{ForceF}]$$

$$\frac{\mathsf{getfun}(H, \Gamma, \mathtt{x}) = \nu}{\mathbb{C}[\mathtt{x}(\overline{\mathtt{a}})]\ \Gamma * S; H \implies \mathbb{C}[\nu(\overline{\mathtt{a}})]\ \Gamma * S; H} \quad [\textsc{GetF}]$$

$$\frac{H(\nu) = \lambda\overline{\mathtt{f}}.\mathtt{e}, \Gamma' \qquad \mathsf{args}(\overline{\mathtt{f}}, \overline{\mathtt{a}}, \Gamma, \Gamma', H) = F, \Gamma'', H'}{\mathbb{C}[\nu(\overline{\mathtt{a}})]\ \Gamma * S; H \implies \mathtt{e}\ \Gamma'' * \mathbb{C}[\nu(\overline{\mathtt{a}})]\ \Gamma * S; H'} \quad [\textsc{InvF}]$$

$$\frac{\Gamma = \iota * \Gamma' \qquad \iota(H, \mathtt{x}) = \nu \qquad H(\nu) = \lambda\overline{\mathtt{f}}.\mathtt{e}, \Gamma''}{\mathsf{getfun}(H, \Gamma, \mathtt{x}) = \nu} \quad [\textsc{GetF1}]$$

$$\frac{\Gamma = \iota * \Gamma' \qquad \iota(H, \mathtt{x}) = \nu \qquad H(\nu) \neq \lambda\overline{\mathtt{f}}.\mathtt{e}, \Gamma''}{\mathsf{getfun}(H, \Gamma, \mathtt{x}) = \mathsf{getfun}(H, \Gamma', \mathtt{x})} \quad [\textsc{GetF2}]$$

$$\frac{\Gamma = \iota * \Gamma' \qquad \iota(H, \mathtt{x}) = \delta \qquad H(\delta) = \nu \qquad H(\nu) = \lambda\overline{\mathtt{f}}.\mathtt{e}, \Gamma''}{\mathsf{getfun}(H, \Gamma, \mathtt{x}) = \nu} \quad [\textsc{GetF3}]$$

$$\frac{\Gamma = \iota * \Gamma' \qquad \iota(H, \mathtt{x}) = \delta \qquad H(\delta) = \mathtt{e}\ \Gamma''}{\mathsf{getfun}(H, \Gamma, \mathtt{x}) = \delta} \quad [\textsc{GetF4}]$$

$$\frac{\Gamma = \iota * \Gamma' \qquad \iota(H, \mathtt{x}) = \delta \qquad H(\delta) = \nu \qquad H(\nu) \neq \lambda\overline{\mathtt{f}}.\mathtt{e}, \Gamma''}{\mathsf{getfun}(H, \Gamma, \mathtt{x}) = \mathsf{getfun}(H, \Gamma', \mathtt{x})} \quad [\textsc{GetF5}]$$

Variable lookup in R

`c()` ≠ **`d←c; d()`**

< 0.05% context sensitive lookups

symbols that affected are **`c`** and **`file`**

Morandat, Hill, Osvald, Vitek. **Evaluating the Design of the R Language**. ECOOP'12


I HATE YOU THIS MUCH!
-darks.-04-

- Programming language techniques **are** needed **in practice**
- **But** we must change our value system to reward and measure benefits to users
- Language research must be **informed** and motivated by **real-world usage**
- We must **embrace dynamism** and push static techniques into the runtime