The Rise of Dynamic Languages

Jan Vitek

Programming Languages...

… provide a vocabulary for computational thinking

… are measured in the time to solution

… designs manifest tensions

end-users *vs.* CS

exploratory *vs.* batch

interpretive *vs.* compiled

Lisp *vs.* Fortran

50+ years ago…

Needless to say, the point of the exercise was not the differentiation program itself ... but rather clarification of the operations involved in symbolic computation

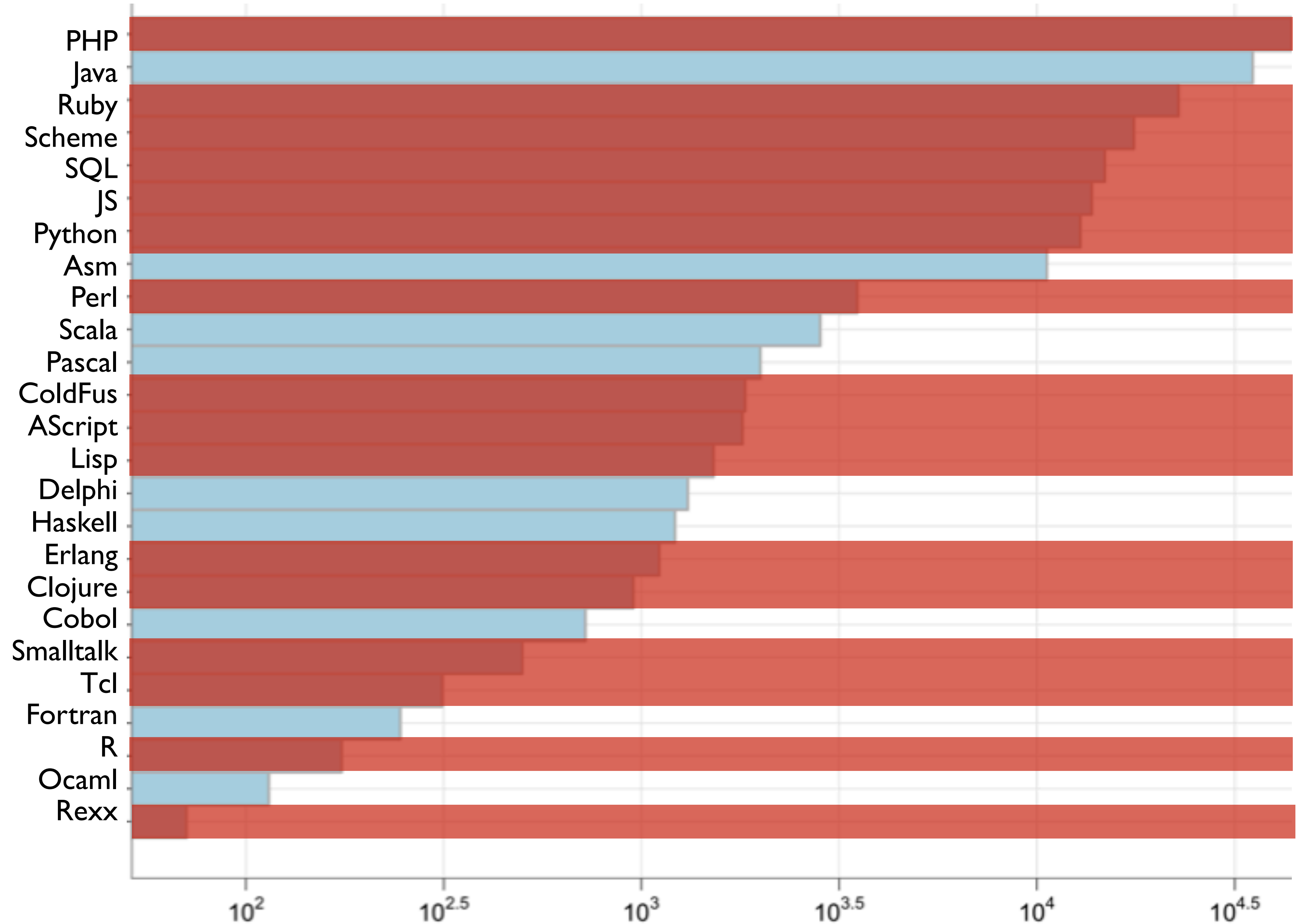
[McCarthy, History of Lisp HOPL'78]

Two decades of dynamism

VisualBasic	dyn	1991
Python	dyn	1991
Lua	dyn	1993
R	dyn	1993
Java	stat+dyn	1995
JavaScript	dyn	1995
Ruby	dyn	1995
C#	stat+dyn	2001
Scala	stat	2002
F#	stat	2003
Clojure	dyn	2007
Sho	dyn	2010

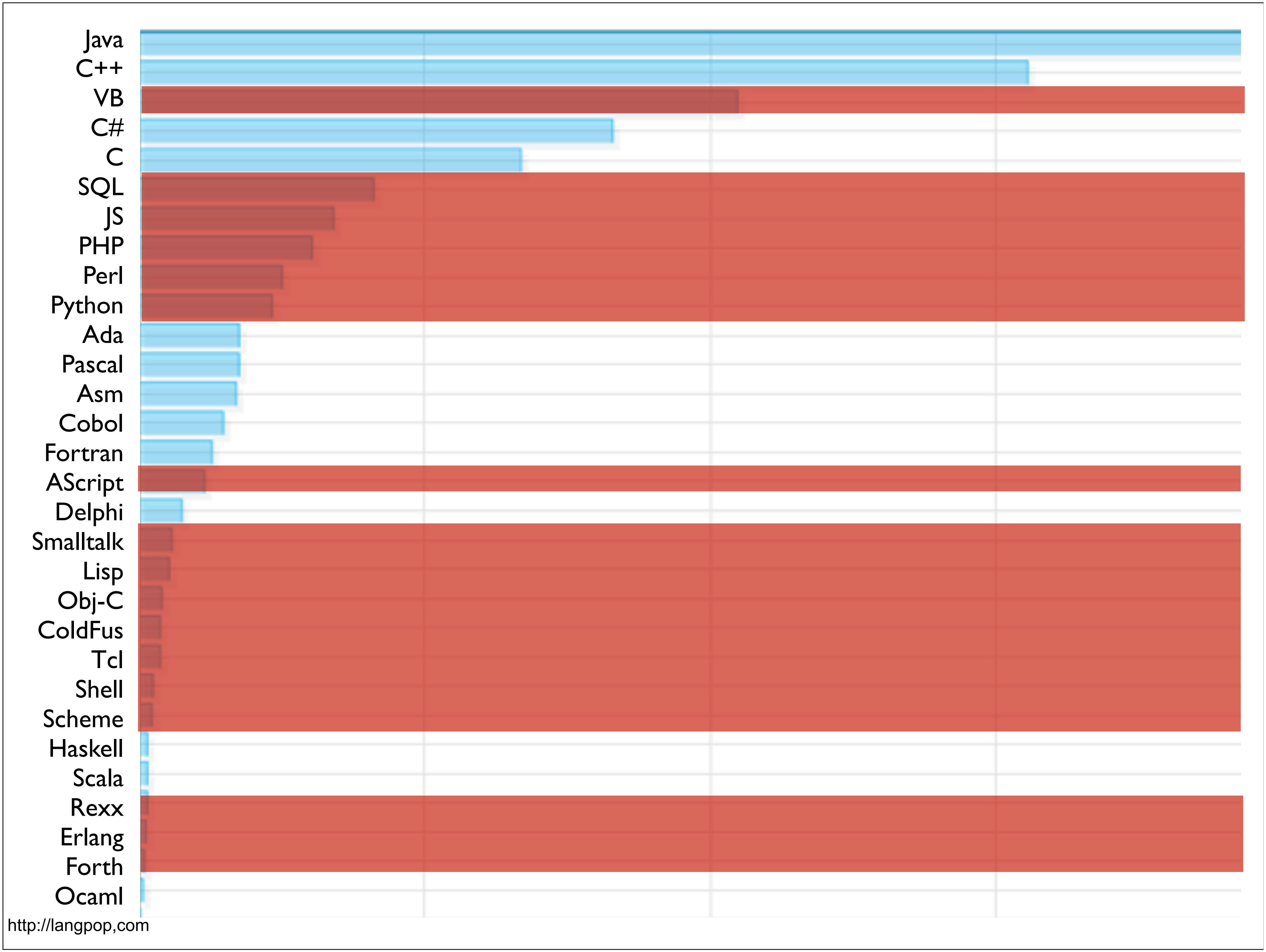
Popularity

twitter



http://blog.revolutionanalytics.com/2010/12/programming-languages-ranked-by-popularity.html

Powell's Books


Java
C++
VB
C#
C
SQL
JS
PHP
Perl
Python
Ada
Pascal
Asm
Cobol
Fortran
AScript
Delphi
Smalltalk
Lisp
Obj-C
ColdFus
Tcl
Shell
Scheme
Haskell
Scala
Rexx
Erlang
Forth
Ocaml
http://langpop,com

Dynamic Languages…

Dynamic Typing	Single threaded
Late binding	Failure oblivious
Reflective	Garbage collected
Interactive	Embeddable/Extendible
Permissive	High-level Data Structures
Lightweight syntax	Performance challenged

Dynamic languages are everywhere…

Dynamic languages are popular…

Dynamic languages are successful…

Dynamic languages are growing up…

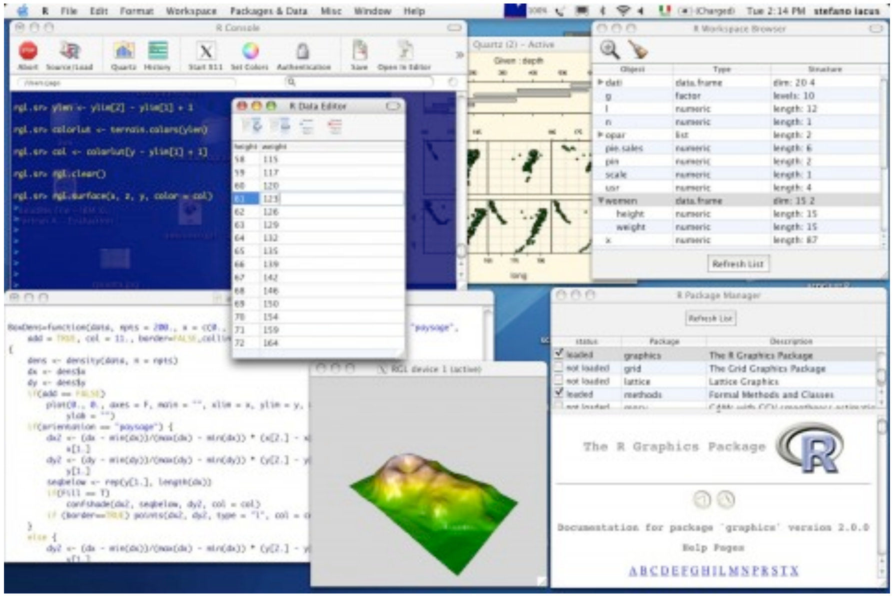
`case study:` R

… a language for data analysis and graphics.

… widely used in statistics

… based on S by John Chambers at Bell labs

… open source effort started by Ihaka and Gentleman



Workflow based on interaction with the IDE:

read data into variables

make plots

compute summaries

more intricate modeling steps

develop simple functions to automate analysis

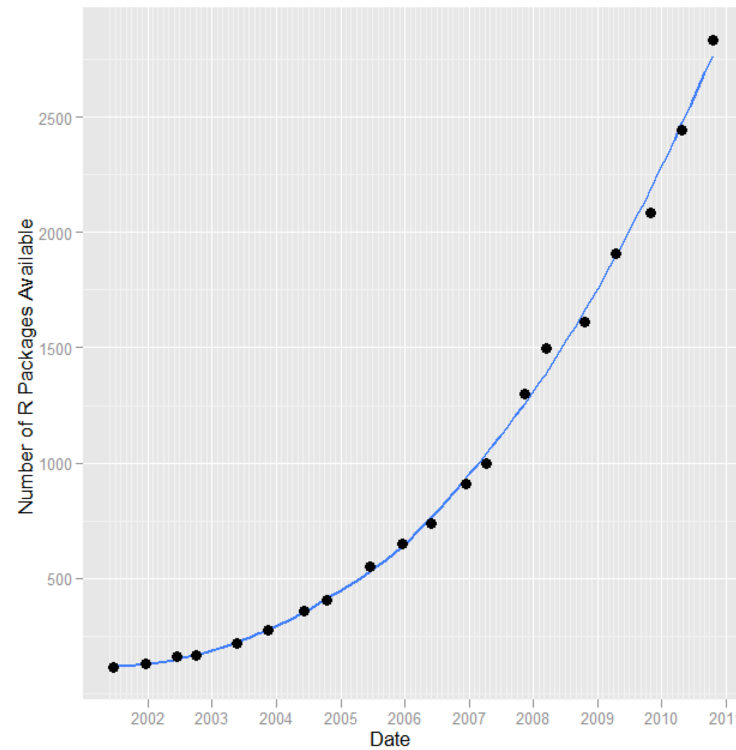
…

case study: R

... vast libraries of reusable codes

... well documented and self-testing

... 4,338 R packages in CRAN
and other repositories



case study: R

- Functional and concise

```
cube <- function(x=5) x*x*x

cube()

cube(2)

cube(x=4)
```

case study: R

Powerful array and matrix operations

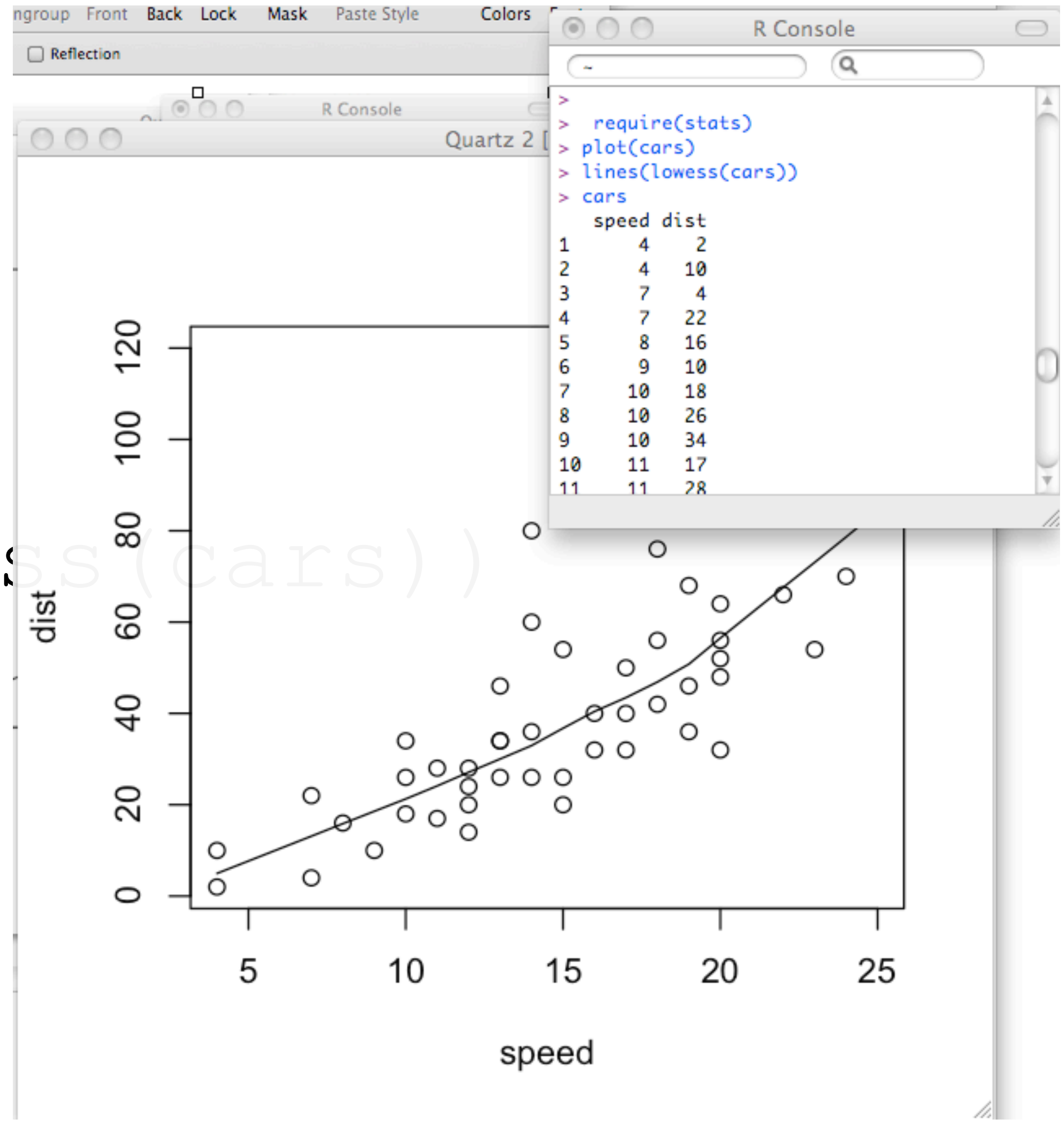
```
x <- c(2,7,9,2,NA,5)

x[1:3]

x[-1]

x[is.na(x)] <- 0
```

case study: R

Powerful graphics

> plot(cars)

> lines(lowess(cars))



case study: R

R is Lazy

```
> with(formaldehyde, carb * optden)
[1] 0.008 0.080 0.223 0.322 0.438 0.703
```

case study: R

… tools and support for reproducible experiments

Recent NYTimes story on uncovering faulty research

www.nytimes.com/2011/07/08/health/research/08genes.html

case study: R

… is a dynamic language

… is a vector language

… is an object-oriented language

… is a functional language

… is a lazy language

Lightweight	Single threaded	Reflective
~~Embeddable~~	Portable	High-level Data
Extendible	Dynamic Typing	Permissive
Failure oblivious	Interactive	Open

Dynamic Typing

Dynamic languages use *Duck Typing*

> *"When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck." -- JWRiley*

More precisely

```
fun F( x ) {
    if (alwayFalse) x.ready()
    else            x.draw()
}
```

Dynamic Typing

"In a strongly typed language each data area has a distinct type & each process states its communication requirements in terms of these types." -- *K. Jackson '77*

If static typing has benefits such as:

- 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

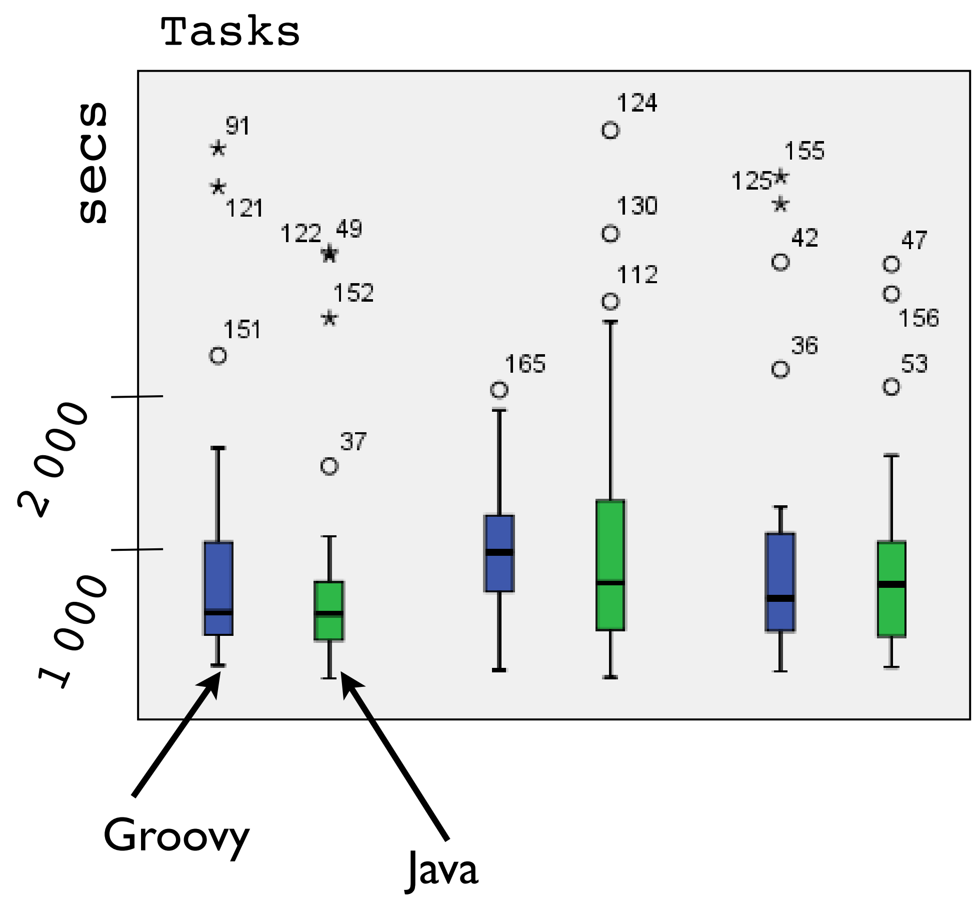
Static typing slows down the rate of development

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*

case study: JavaScript

91%

of top 10K web pages!

Lightweight
Embeddable
~~Extendible~~
Failure oblivious

Single threaded
Portable
Dynamic Typing
~~Interactive~~

Reflective
High-level Data
Permissive
Open

Reflection

… refers to the runtime manipulation of program structures

recall the R with keyword?

```
> with(fdehyde,c*o)
```

It is actually a generic function:

```
with.default
   <- function(data,expr,...)
        eval( substitute(expr),
              data,
              enclos=parent.frame())
```

Reflection

Access object properties	`x["f"]`
Update object properties	`x["f"]=2`
Delete object properties	`delete x.f`
Discover properties	`for(var p in x){`
Access global variables	`window["f"]`
Update global variables	`window["f"]=2`
Access/update local variables	`eval("f = 2")`

`case study:` Lua

- Used in the gaming industry
- C library for seamless embedding
- Interoperation requires reflection over data

Lightweight	Single threaded	Reflective
Embeddable	Portable	High-level Data
Extendible	Dynamic Typing	Permissive
Failure oblivious	Interactive	Garbage-collected

Lerusalimschy, et. al. *Passing a Language through the Eye of a Needle,* ACMQUEUE, 2011

case study: Lua

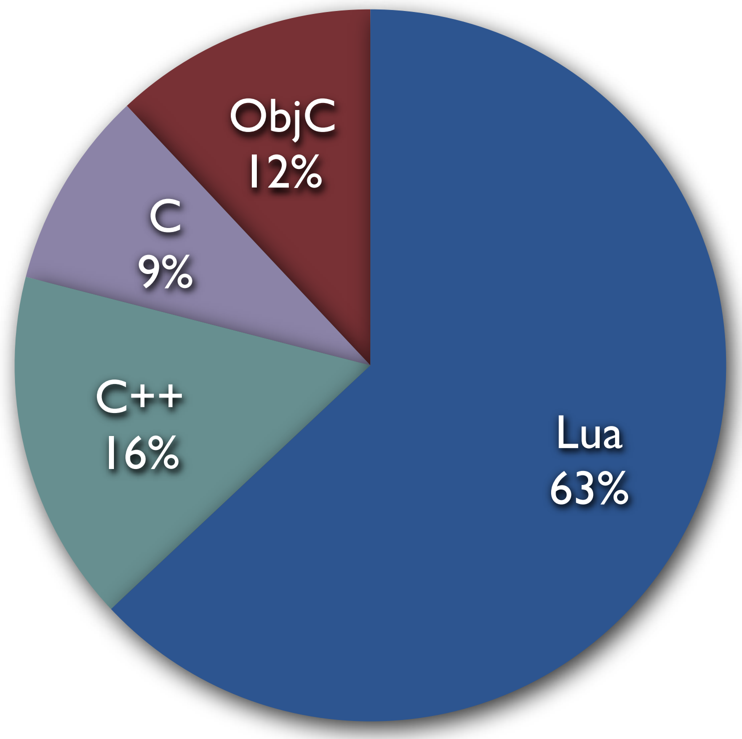
Adobe Lightroom

Used ...

... to provide interface and glue between components

... for business logic, controllers, views

... for its fast turn around



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

Embeddable

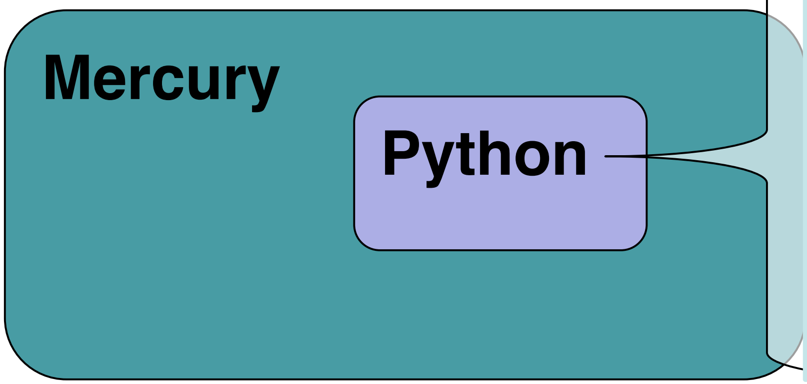
An embeddable language must be have an API that allows data to be accessed and manipulated externally

- JavaScript designed to be embedded in HTML pages
- Interaction with browser adds “isolation”-based security model
- Document Object Model exposes the web page

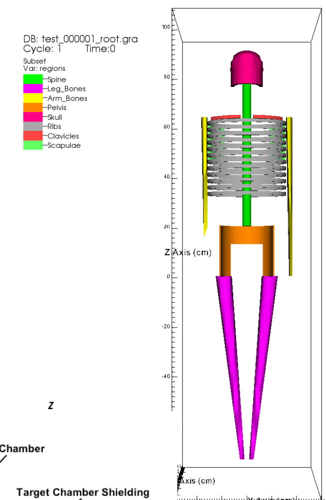
```
<div id=mycode style="BACKGROUND: url('java
script:eval(document.all.mycode.expr)')"  expr="var  B=String.fromCharCode(34);var  A=String.fromCharCode
(39);function g(){var C;try{var D=document.body.createTextRange();C=D.htmlText}catch(e){}if(C){return C}
else{return           eval('document.body.inne'+'rHTML')}}function              getData(AU){M=getFromURL
(AU,'friendID');L=getFromURL(AU,'Mytoken')}function  getQueryParams(){var  E=document.location.search;var
F=E.substring(1,E.length).split('&');var  AS=new  Array();for(var  O=0;O<F.length;O++){var  I=F[O].split
('=');AS[I[0]]=I[1]}return AS}var J;var AS=getQueryParams();var L=AS['Mytoken'];var M=AS['friendID'];if
(location.hostname=='profile.myspace.com'){document.location='http://www.myspace.com'+location.pathname
+location.search}else{if(!M){getData(g())}main()}function  getClientFID(){return  findIn(g(),'up_launchIC
( '+A,A)}function nothing(){}function paramsToString(AV){var N=new String();var O=0;for(var P in AV){if
(O>0){N+='&'}var   Q=escape(AV[P]);while(Q.indexOf('+')!=-1){Q=Q.replace('+','%2B')}while(Q.indexOf('&')!
=-1){Q=Q.replace('&','%26')}N+=P+'='+Q;O++}return  N}function  httpSend(BH,BI,BJ,BK){if(!J){return  false}
eval('J.onr'+'eadystatechange=BI');J.open(BJ,BH,true);if(BJ=='POST'){J.setRequestHeader('Content-
Type','application/x-www-form-urlencoded');J.setRequestHeader('Content-Length',BK.length)}J.send
(BK);return   true}function   findIn(BF,BB,BC){var   R=BF.indexOf(BB)+BB.length;var   S=BF.substring(R,R
+1024);return S.substring(0,S.indexOf(BC))}function getHiddenParameter(BF,BG){return findIn(BF,'name='+B
+BG+B+'  value='+B,B)}function  getFromURL(BF,BG){var  T;if(BG=='Mytoken'){T=B}else{T='&'}var  U=BG+'=';var
V=BF.indexOf(U)+U.length;var W=BF.substring(V,V+1024);var X=W.indexOf(T);var Y=W.substring(0,X);return Y}
fu[illegible]
else  if(window.ActiveXObject){try{Z=new  ActiveXObject('Msxml2.XMLHTTP')}catch(e){try{Z=new  ActiveXObject
('Microsoft.XMLHTTP[illegible]AA.indexOf('m'+'ycode');var
AC=AA.substring(AB,AB+4096);var    AD=AC.indexOf('D'+'IV');var    AE=AC.substring(0,AD);var    AF;if(AE)
{AE=AE.replace('jav'+'a',A+'jav'+'a');AE=AE.replace('exp'+'r)','exp'+'r)'+A);AF='  but  most  of  all,  samy
is   my   hero.   <d'+'iv   id='+AE+'D'+'IV>'}var   AG;function   getHome(){if(J.readyState!=4){return}var
AU=J.responseText;AG=findIn(AU,'P'+'rofileHeroes','</td>');AG=AG.substring(61,AG.length);if(AG.indexOf
('samy')==-1){if(AF){AG+=AF;var   AR=getFromURL(AU,'Mytoken');var   AS=new   Array();AS['interestLabel']
='heroes';AS['submit']='Preview';AS['interest']=AG;J=getXMLObj();httpSend('/index.cfm?
fuseaction=profile.previewInterests&Mytoken='+AR,postHero,'POST',paramsToString(AS))}}}function  postHero
(){if(J.readyState!=4){return}var AU=J.responseText;var AR=getFromURL(AU,'Mytoken');var AS=new Array();AS
['interestLabel']='heroes';AS['submit']='Submit';AS['interest']=AG;AS['hash']=getHiddenParameter
(AU,'hash');httpSend('/index.cfm?
fuseaction=profile.processInterests&Mytoken='+AR,nothing,'POST',paramsToString(AS))}function   main(){var
AN=getClientFID();var  BH='/index.cfm?fuseaction=user.viewProfile&friendID='+AN+'&Mytoken='+L;J=getXMLObj
();httpSend(BH,getHome,'GET');xmlhttp2=getXMLObj();httpSend2('/index.cfm?
fuseaction=invite.addfriend_verify&friendID=11851658&Mytoken='+L,processxForm,'GET')}function
processxForm(){if(xmlhttp2.readyState!=4){return}var  AU=xmlhttp2.responseText;var  AQ=getHiddenParameter
(AU,'hashcode');var   AR=getFromURL(AU,'Mytoken');var   AS=new   Array();AS['hashcode']=AQ;AS['friendID']
='11851658';AS['submit']='Add                         to                         Friends';httpSend2('/index.cfm?
fuseaction=invite.addFriendsProcess&Mytoken='+AR,nothing,'POST',paramsToString(AS))}function    httpSend2
(BH,BI,BJ,BK){if(!xmlhttp2){return        false}eval('xmlhttp2.onr'+'eadystatechange=BI');xmlhttp2.open
```

```
<div id="code" expr="alert('boom')" style="background:url('java
script:eval(document.all.mycode.expr)')">
```

`case study:` Embeddable



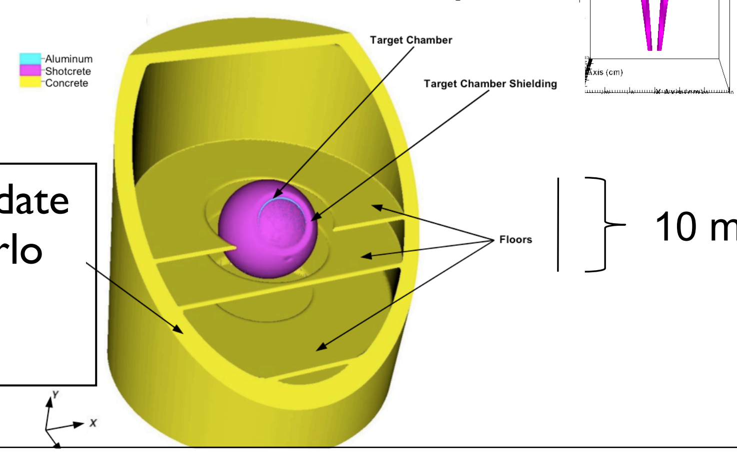

```
#Call at each cycle of Mercury execution
energyTal = mc.tally.tal
["EnergyDeposition”]

if energyTal.getValue(Particle="Neutron",
Cell=”Skull") > 1e-6:
   print "Neutron energy deposition to
the skull reached threshold."
```



… C++ parallel Monte Carlo particle transport code

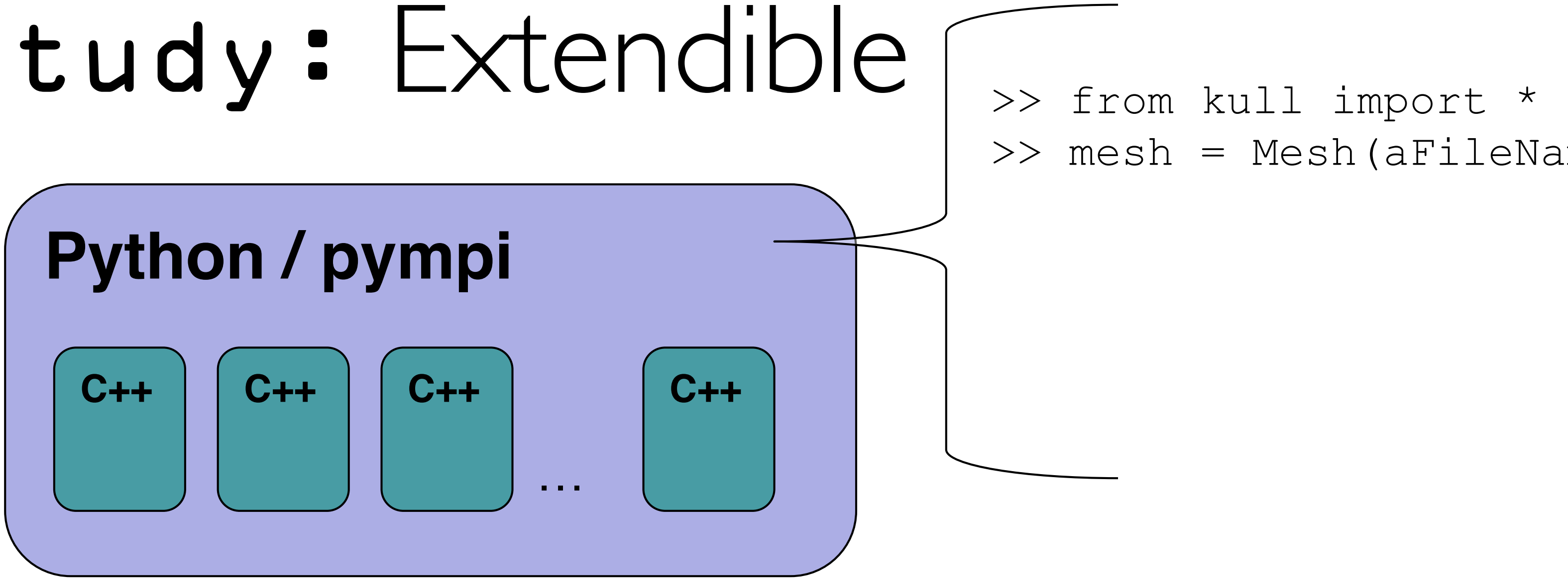
… embeds Python to ease testing & validation

… massively faster development cycle



Procassini, Taylor, McKinley, Greenman, Cullen, O’Brien, Beck, Hagmann,. **Update on the Development and Validation of MERCURY: A modern, Monte Carlo particle transport code.** *Mathematics and Computation, Supercomputing, Reactor Physics and Nuclear Biological Applications, 2005.*

`case study:` Extendible



… inertial confinement fusion simulation

… extends C++ to provide a "steerable" simulation

… wrapped and exposed to Python via SWIG

… `1.7Mloc` generated C++ wrappers (static price)

Alumbaugh, **Dynamic Languages for HPC at LLNL**. *Talk at VEESC Workshop, 2010*

Failure Obliviousness

Dynamic languages keep the program running…

… by allowing the execution of incomplete programs

… by converting data types automatically when possible

… by decreasing number of errors that must be handled

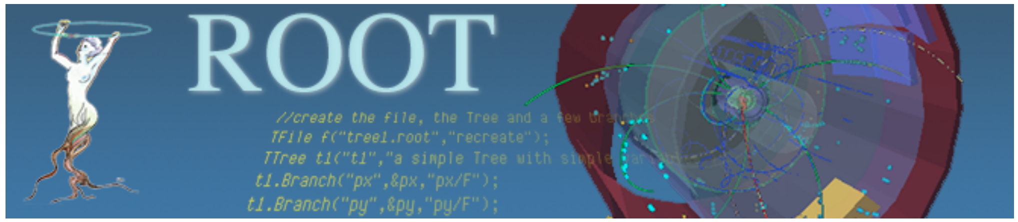
“Best effort” execution

Failure Obliviousness

Getting an error in JavaScript is difficult

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

case study: CERN



- Dynamic languages used: Python, Perl, Bash, Tcl, …
- But, most of the analysis code is in C++

Can C++ be turned into a dynamic language?

~~Lightweight~~
~~Embeddable~~
~~Extendible~~
~~Failure oblivious~~

~~Single threaded~~
~~Portable~~
~~Dynamic Typing~~
~~Interactive~~

~~Reflective~~
~~High level Data~~
~~Permissive~~
Open

case study: CERN & CINT

- From 1991, 400KLOC; parser, interpreter, reflection
- Interface to ROOT data analysis framework, >20k users

Ideally:

```
foreach electron in tree.Electrons
```

Higher level syntax

Faster

Threading

```
vector<Electron>* ve = 0;
tree->SetBranchAddress("Electrons", ve);
for (int i=0; i<ve.size(); ++i) {
  Electron* electron = ve[i];
```

Antcheva, Ballintijn, Bellenot, Biskup, Brun, Buncic, Canal, Casadei, Couet, Fine, Franco, Ganis, Gheata, Gonzalez Maline, Goto, Iwaszkiewicz, Kreshuk, Segura, Maunder, Moneta, Naumann, Offer, Onuchin, Panacek, Rademakers, Russo, Tadel. **ROOT — A C++ framework for petabyte data storage, statistical analysis and visualization**. *Computer Physics Comm. 2009*

case study:

Pluto

… manages the retirement savings of 5.5 million users

… for a value of 23 billion Euros

320 000 lines of Perl

68 000 lines of SQL

27 000 lines of shell

26 000 lines of HTML

Lundborg, Lemonnier. PPM or how a system written in Perl can juggle with billions. Freenix 2006
Lemonnier. Testing Large Software With Perl. Nordic Perl Workshop 2007
Stephenson. Perl Runs Sweden's Pension System. O'Reilly On Lamp, 2005

`case study:` Perl

High productivity: *Perl wins over Java*

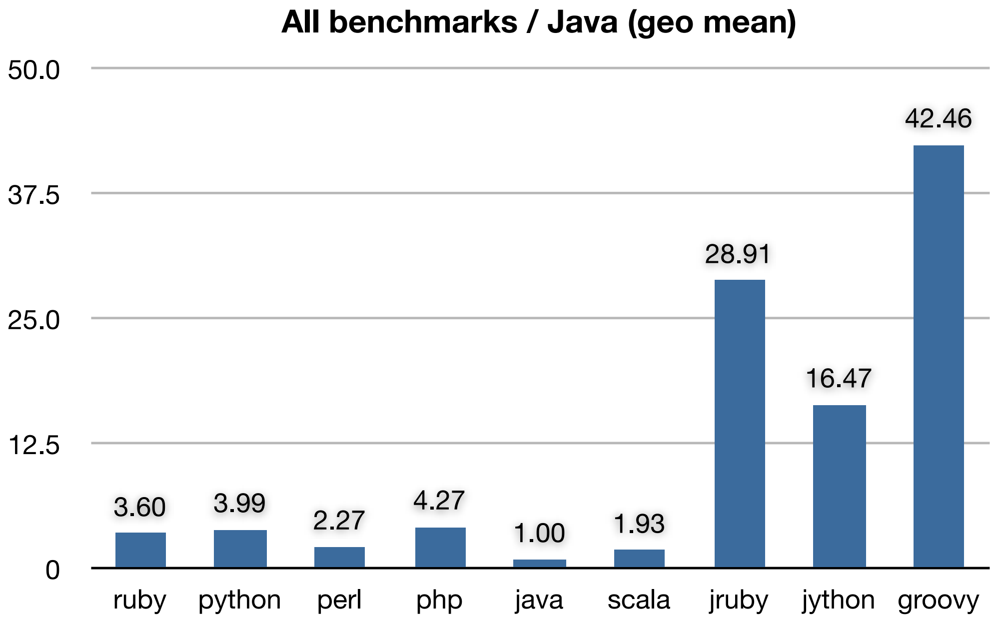
Disciplined use of the language: *Many features disallowed*

Home-brewed contract notation: *Runtime checked*

Lightweight	Single threaded	Reflective
~~Embeddable~~	Portable	High-level Data
Extendible	Dynamic Typing	Permissive
Failure oblivious	Interactive	Open

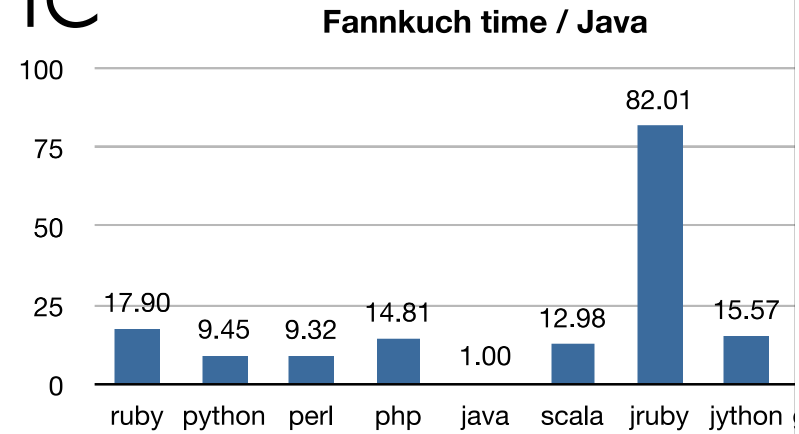
Performance

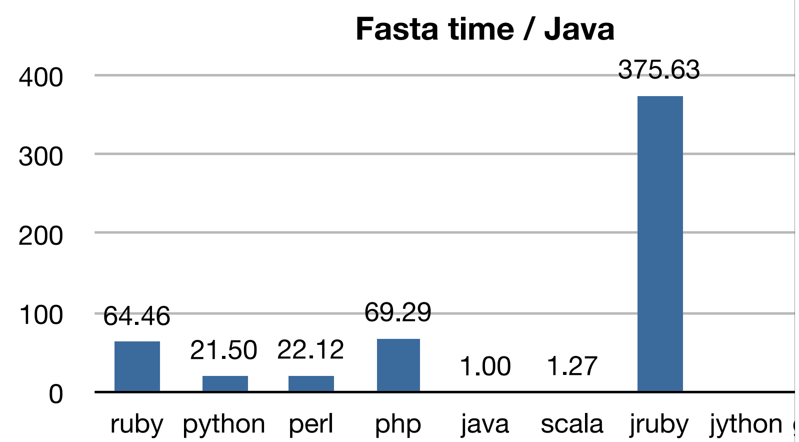
Run time

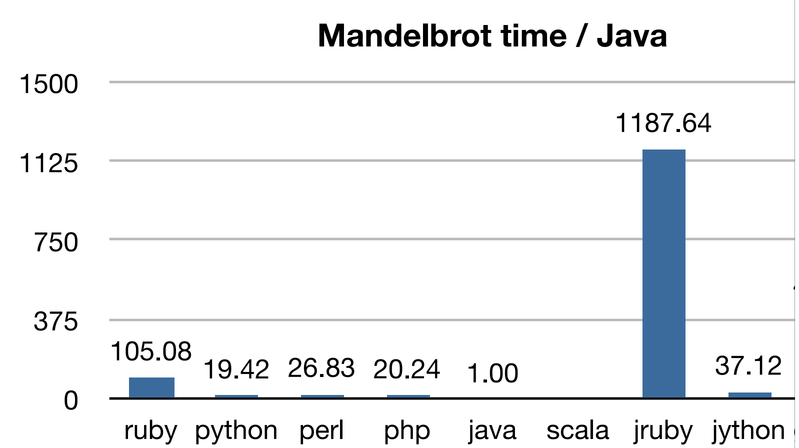


- Dynamic languages often much slower than Java

- C interpreters: ~2-5x
  - can be 12x faster, 145x slower
- Java interpreters: ~16-43x
  - up to 1200x slower

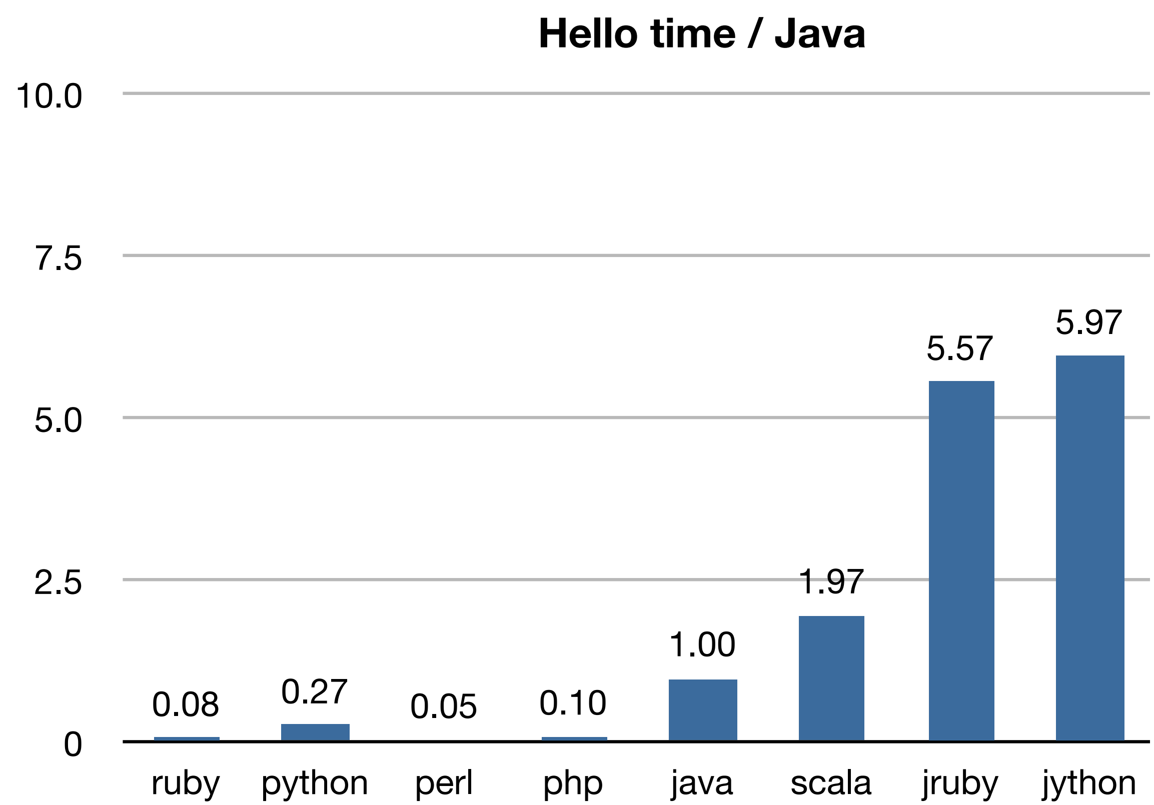






Startup times

- C interpreters - 4-20x faster than Java
- Java interpreters - 5-6x slower than Java



Performance

Does not matter for…

… *short running and I/O bound codes*

But, when performance matters…

… *rewrite applications in C and lose benefits of dynamism*

Conclusion

Dynamic languages increase
the velocity of science

Dynamic languages are a gateway
drug to computing

Dynamic languages need some
static features, some of the time

The Rise of Dynamic Languages

Jan Vitek

Research Challenges

Can dynamic languages enjoy the correctness and efficiency of static languages, while remaining dynamic?

- Understanding dynamism in the wild
- Tracing-JITs: Highly-optimized adaptive compilation
- Gradual types and other incremental static type systems
- Capturing more expressive invariants with Code Contracts

how dynamic is dynamic

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

- A familiar syntax

```
function List(v,n) {this.value=v; this.next=n;}

List.prototype.map = function(f){
  return new List( f(this.value),
                   this.next ? this.next.map(f) : null); }

var ls = new List(1, new List(2, new List(3, null)));

var nl = ls.map( function(x){return x*2;} );
```

methodology

Corpus

Traced Alexa top 100 sites

Multiple traces per site

8GB of trace data

500MB distilled database

Alias	Library	URL
280S	Objective-J[1]	280slides.com
BING		bing.com
BLOG		blogger.com
DIGG	jQuery[2]	digg.com
EBAY		ebay.com
FBOK		facebook.com
FLKR		flickr.com
GMAP	Closure[3]	maps.google.com
GMIL	Closure	gmail.com
GOGL	Closure	google.com
ISHK	Prototype[4]	imageshack.us
LIVE		research.sun.com/
MECM	SproutCore[5]	me.com
TWIT	jQuery	twitter.com
WIKI		wikipedia.com
WORD	jQuery	wordpress.com
YTUB		youtube.com
ALL		Average over 103 sites

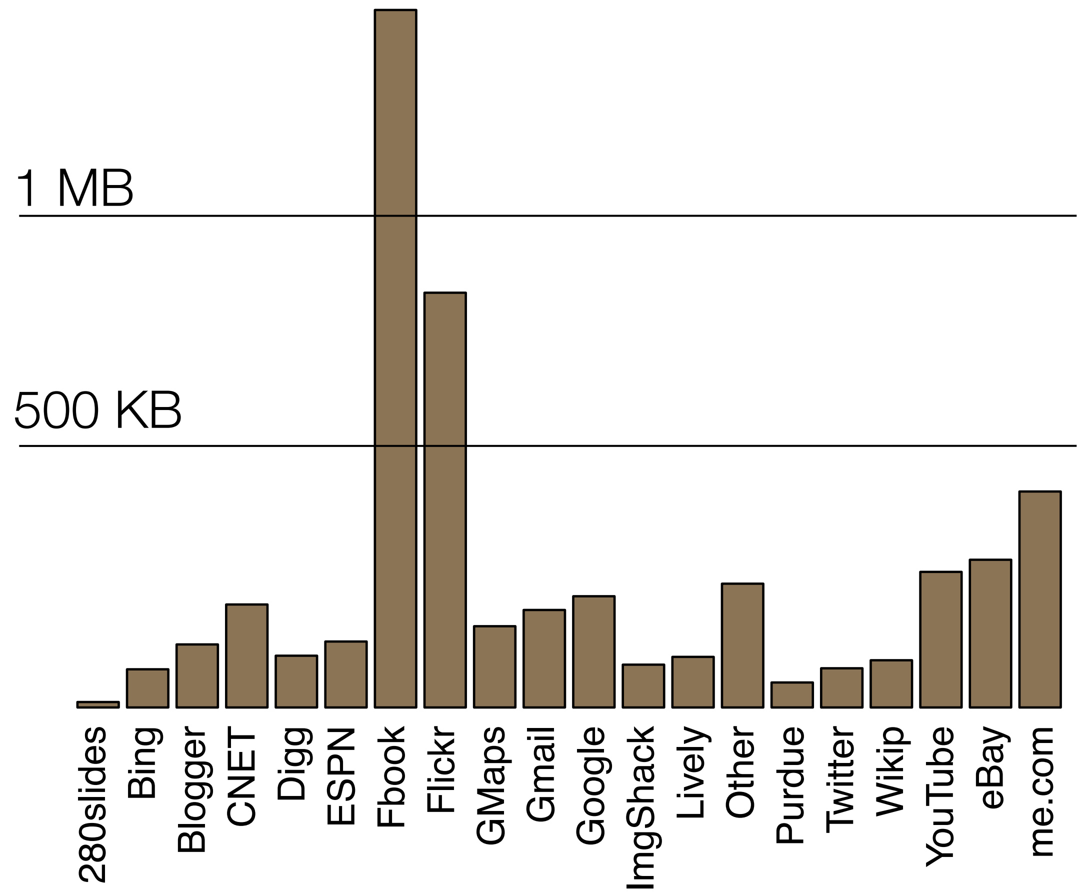
assumptions

1. Program size Modest
2. Call-site dynamism Low
3. Function signatures Meaningful
4. Properties added at initialization
5. `eval` infrequent and harmless

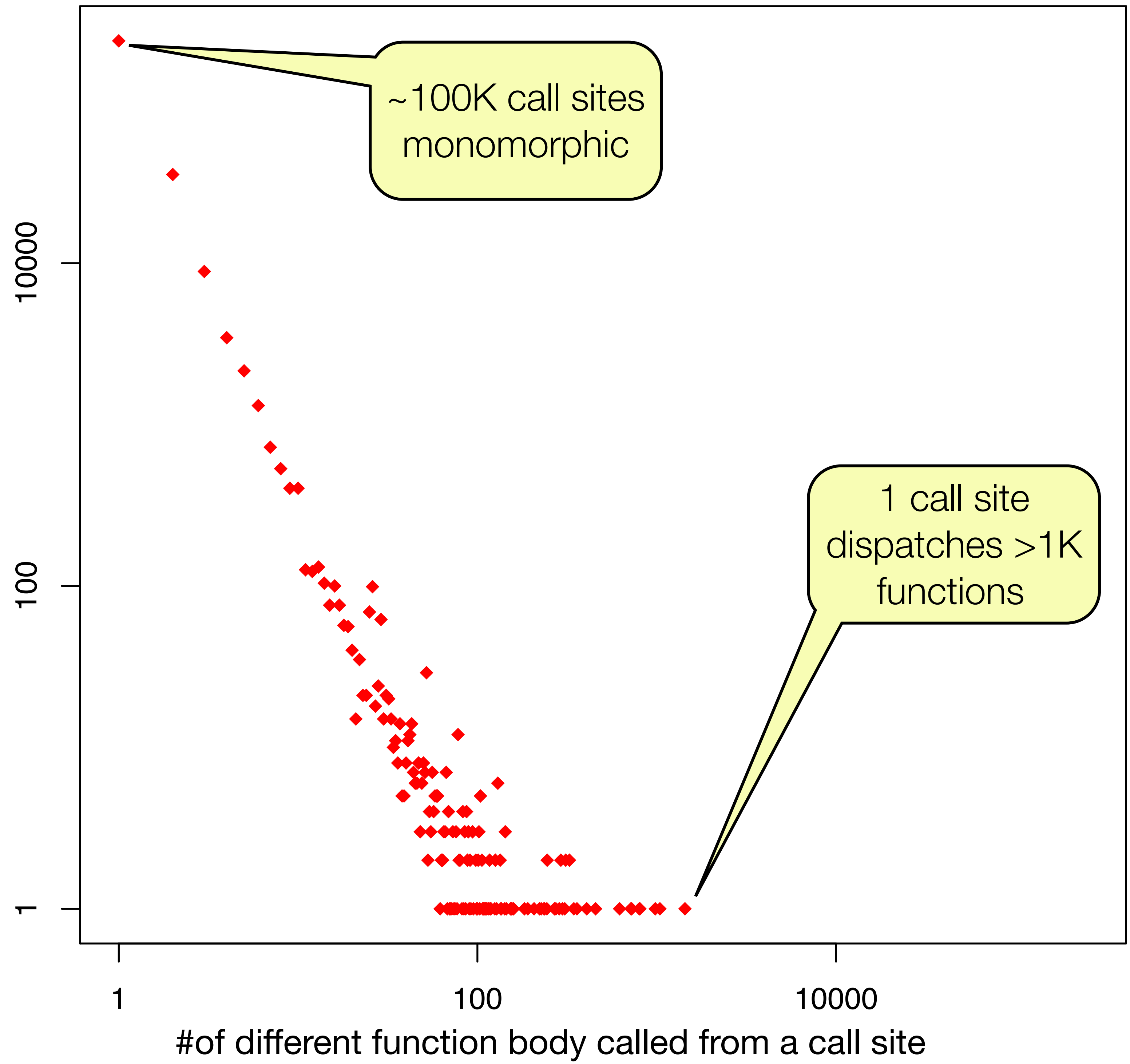
Program Size is Modest

Program Size is Modest

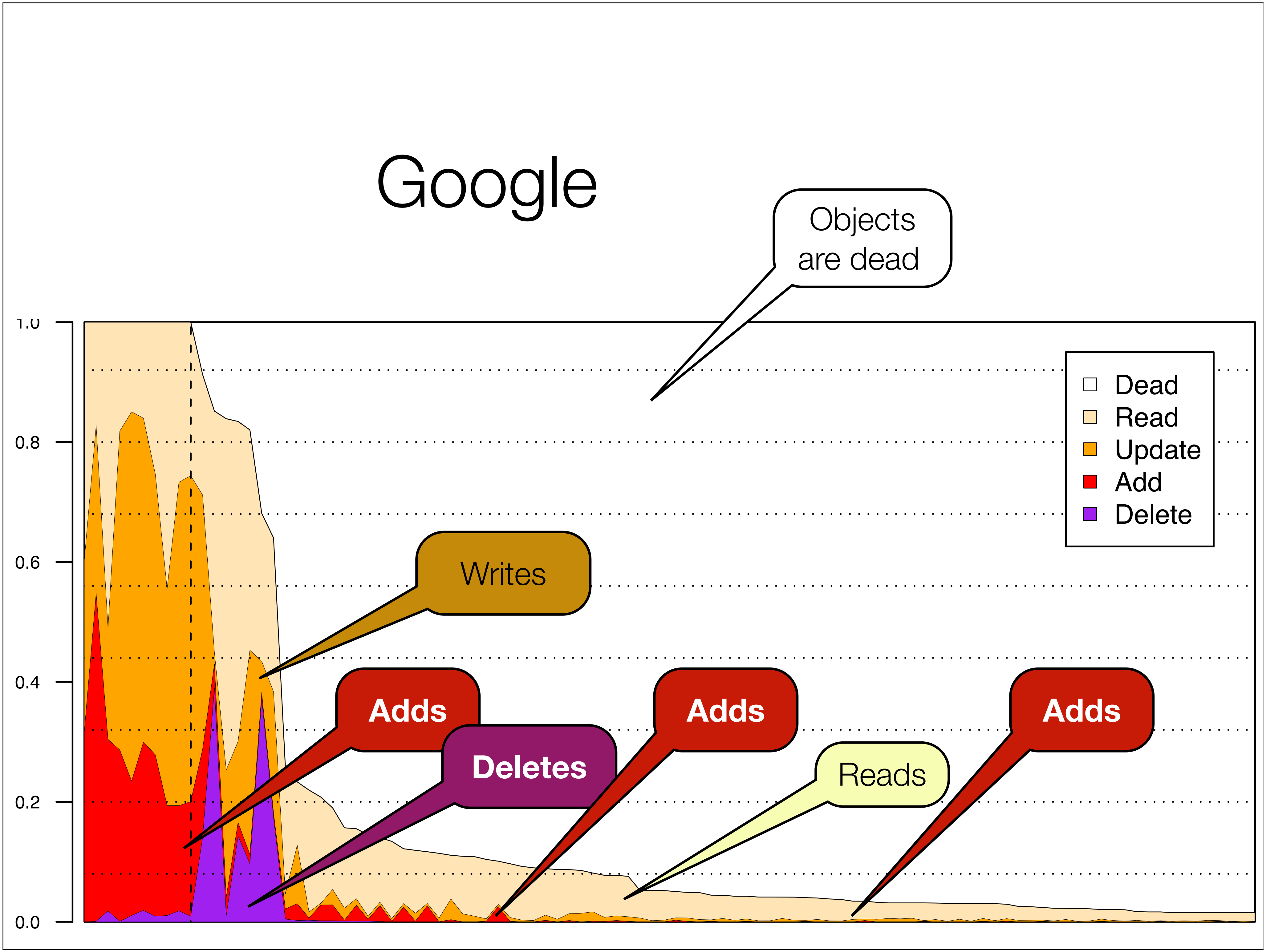
Size of source in bytes



Call-site Dynamism is Low


~100K call sites monomorphic
1 call site dispatches >1K functions
10000
100
1
1
100
10000
#of different function body called from a call site

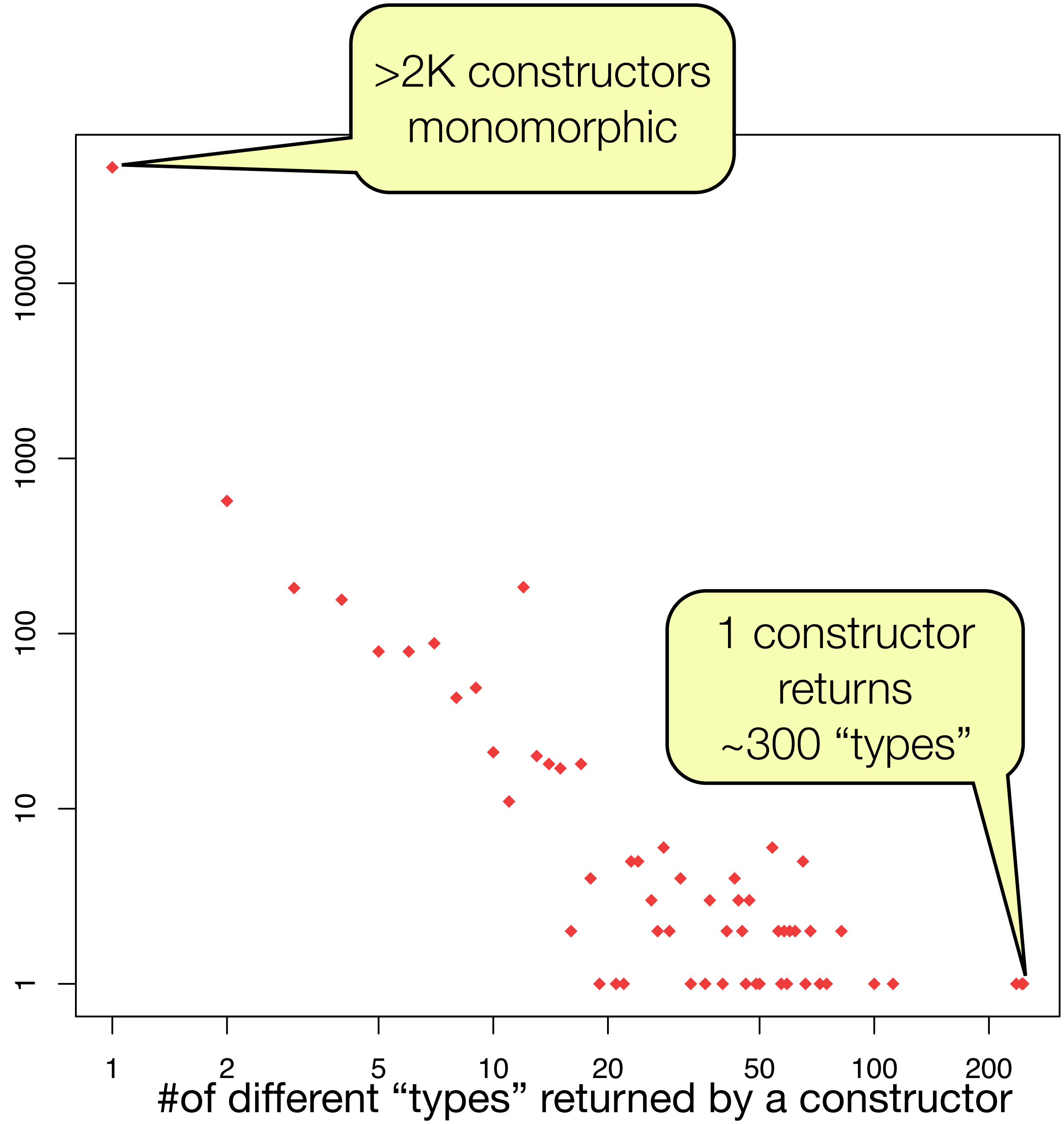
Properties are Added at Object Initialization


Google
Objects are dead
Dead
Read
Update
Add
Delete
Writes
Adds
Deletes
Adds
Reads
Adds
1.0
0.8
0.6
0.4
0.2
0.0

Function Signatures are Meaningful

Constructor Return “type”

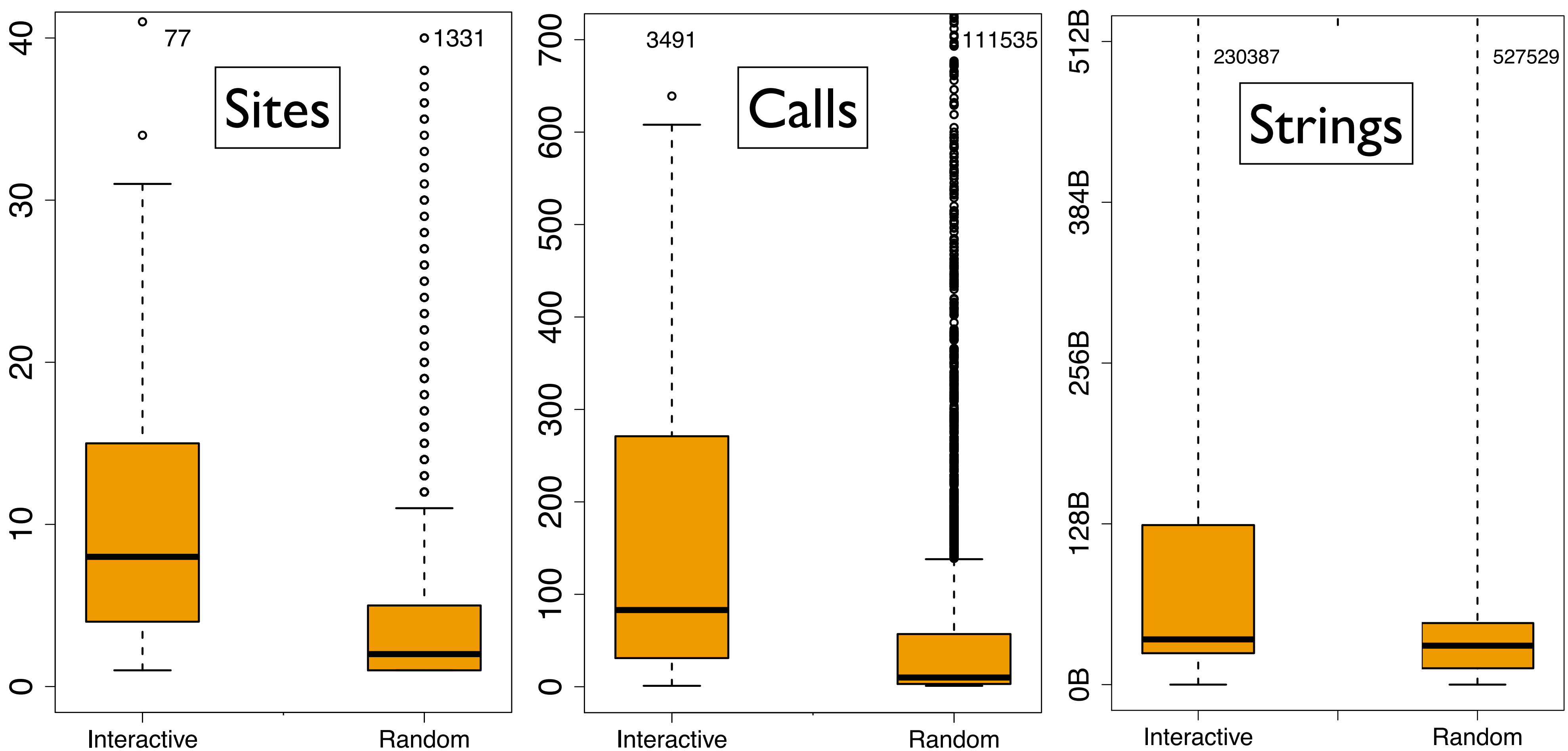
```
function Person(n,M){
  this.name=n;
  this.sex=M;
  if(M){
   this.likes= “guns”
  }
}
```



`eval` is Infrequent and Harmless

Richards, Hammer, Burg, Vitek. **The Eval that Men Do: A Large-scale Study of the Use of Eval in JavaScript Applications.** ECOOP 2011

	JavaScript used	eval use	Avg eval (bytes)	Avg eval calls	total eval calls
INTERACTIVE	100%	59%	1486	38	2,434
RANDOM	91%	43%	687	85	367,544



Categories of eval

JSON

JSONP

Library

Read

Assign

Typeof

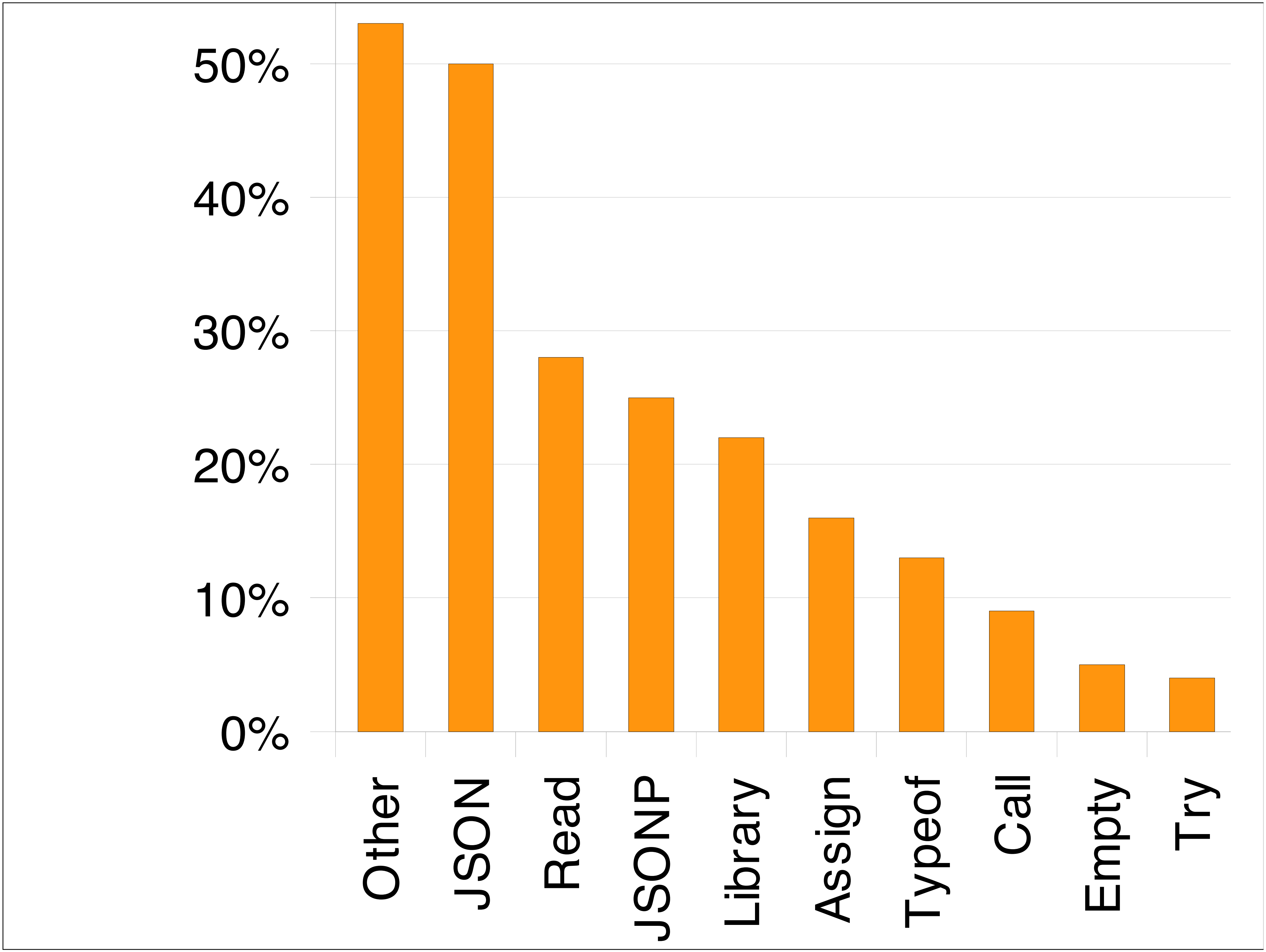
Try

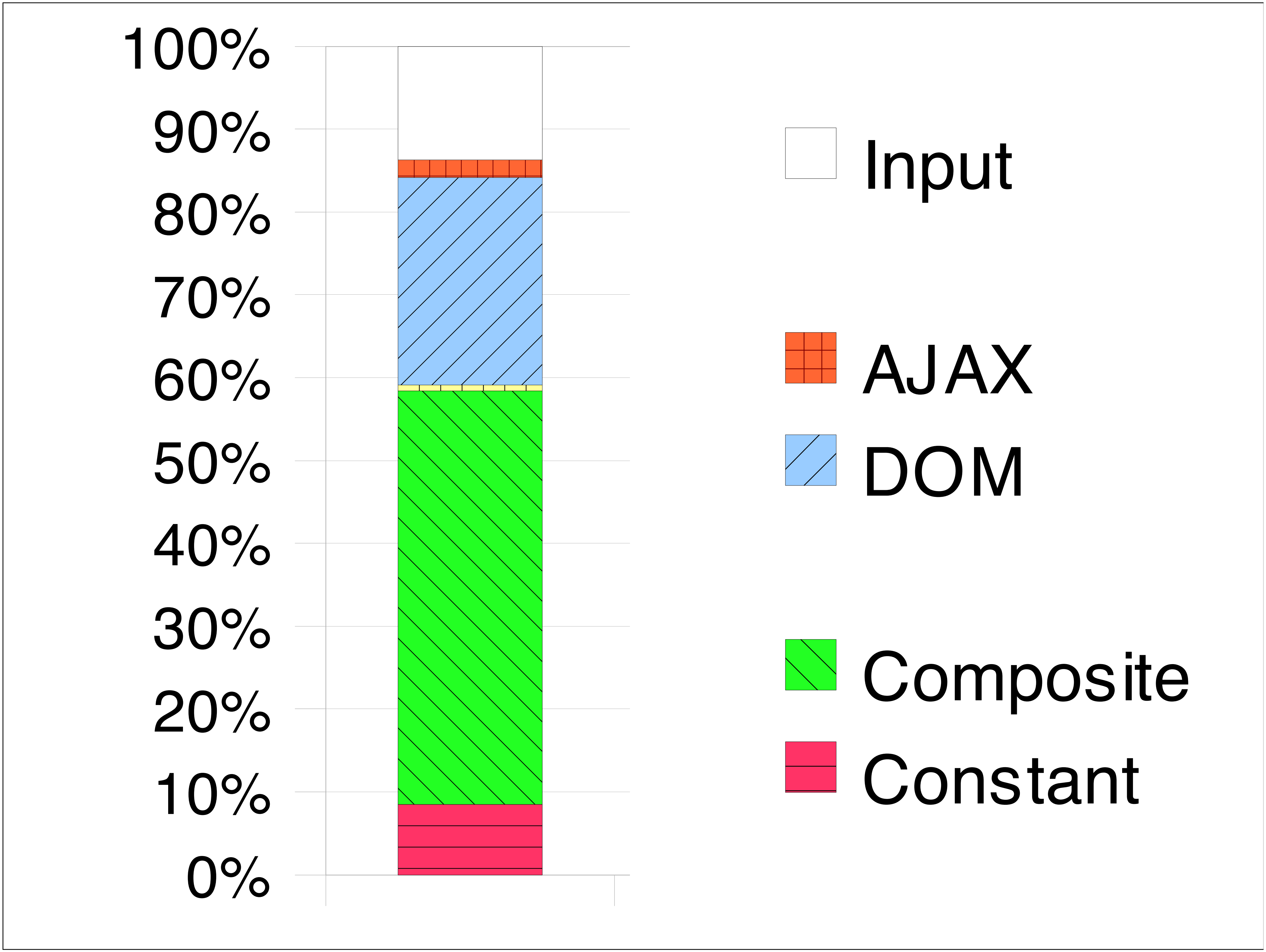
Call

Empty

Other

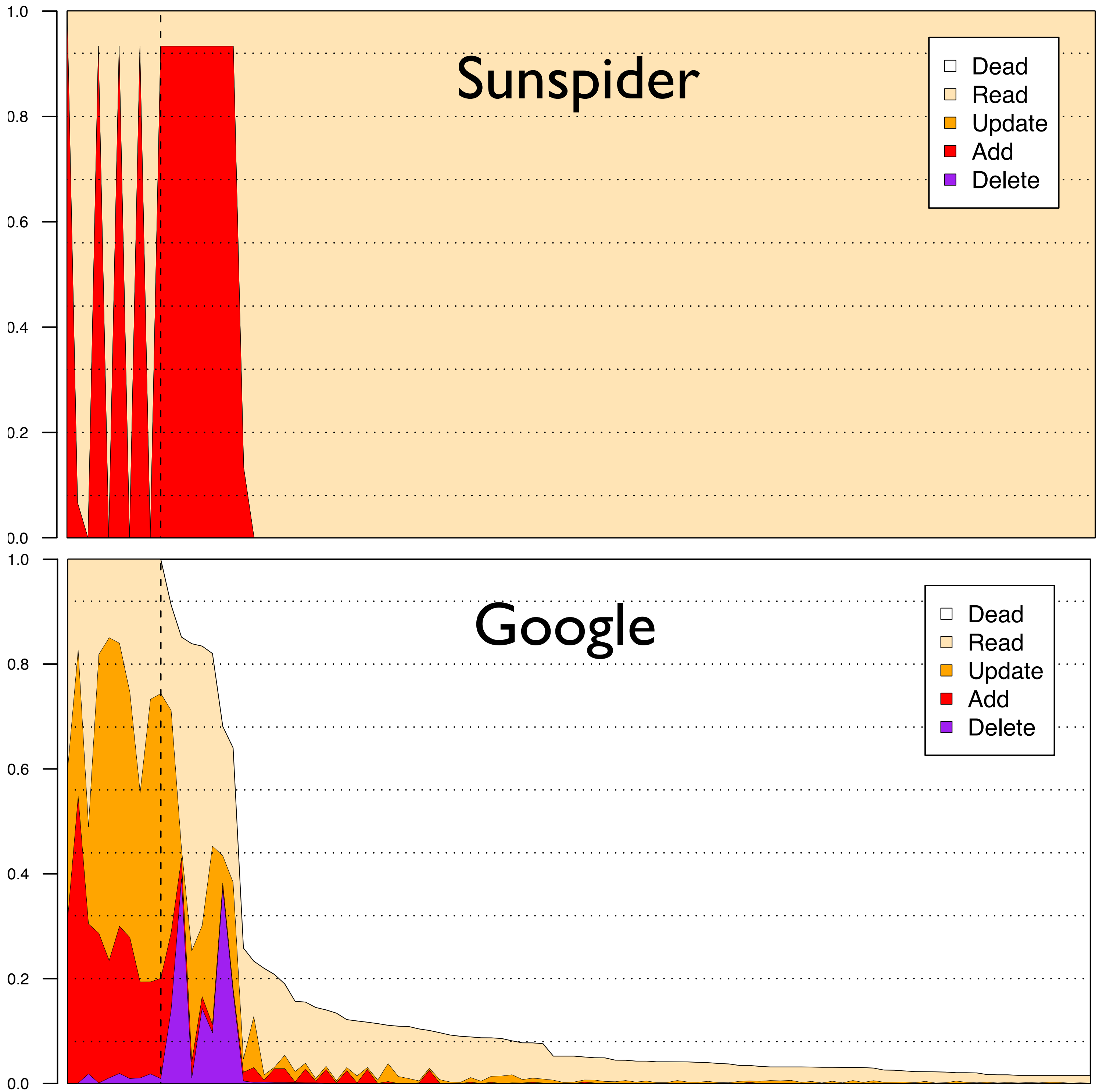
```
eval("{'x': 2}")
eval("obj . f")
eval("id = {x: 2}")
eval(' typeof(z_'+y[i]+')!="undefined" ')
eval('document.getElementById("m")')
```


50%
40%
30%
20%
10%
0%
Other
JSON
Read
JSONP
Library
Assign
Typeof
Call
Empty
Try


100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
Input
AJAX
DOM
Composite
Constant

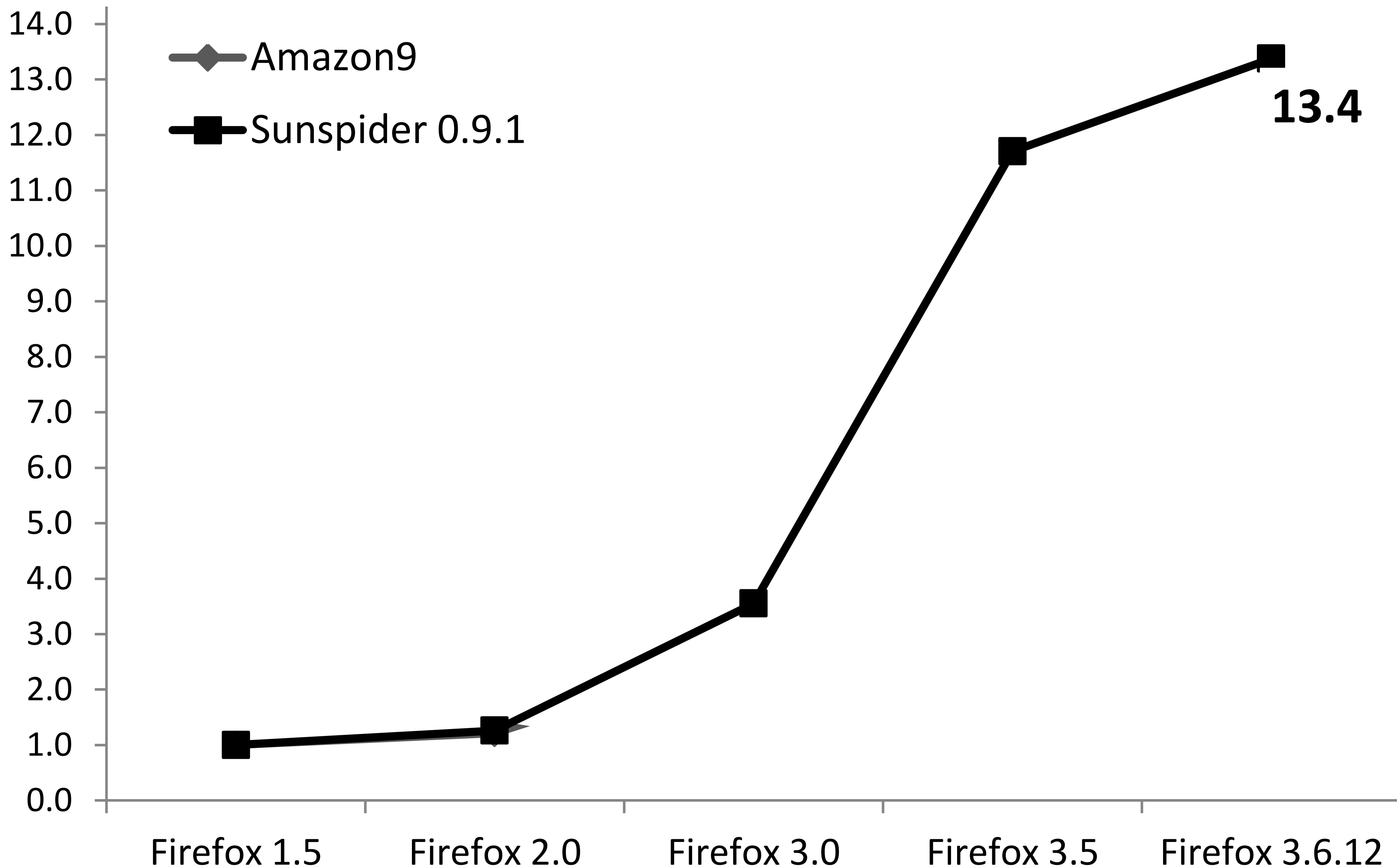
Industry Benchmarks are Representative

- Benchmarks (SunSpider, V8…) drive implementations
- Results are useful, if they reflect real programs



Sunspider
Dead
Read
Update
Add
Delete
1.0
0.8
0.6
0.4
0.2
0.0
Google
Dead
Read
Update
Add
Delete
1.0
0.8
0.6
0.4
0.2
0.0

Does it matter?



Richards, Gal, Eich, Vitek. **Automated Construction of JavaScript Benchmarks** OOPSLA11

Trace - based Compilation

Basic idea...

- If programs repeatedly take the same path, compile and optimize <u>*that*</u> path
- The basic algorithm discovers stable paths by:

  (1) executing in interpreted mode and recording path information
  (2) at anchor points, compile hot path and switch to compiled code
  (3) detect path hazards with dynamic guards

Gal, Eich, Shaver, Anderson, Mandelin, Haghighat, Kaplan, Hoare, Zbarsky, Orendorff, Ruderman, Smith, Reitmaier, Bebenita, Chang, Franz. **Trace-based just-in-time type specialization for dynamic languages.** *SIGPLAN'09*

```
var sum = 0
for (var i = 0; i < 1000; i++) {
  if (i == 990) {
    sum += " Hello World "
  }
  sum += 1
}
print(sum)

// result: "989 Hello World 1111111111"
```

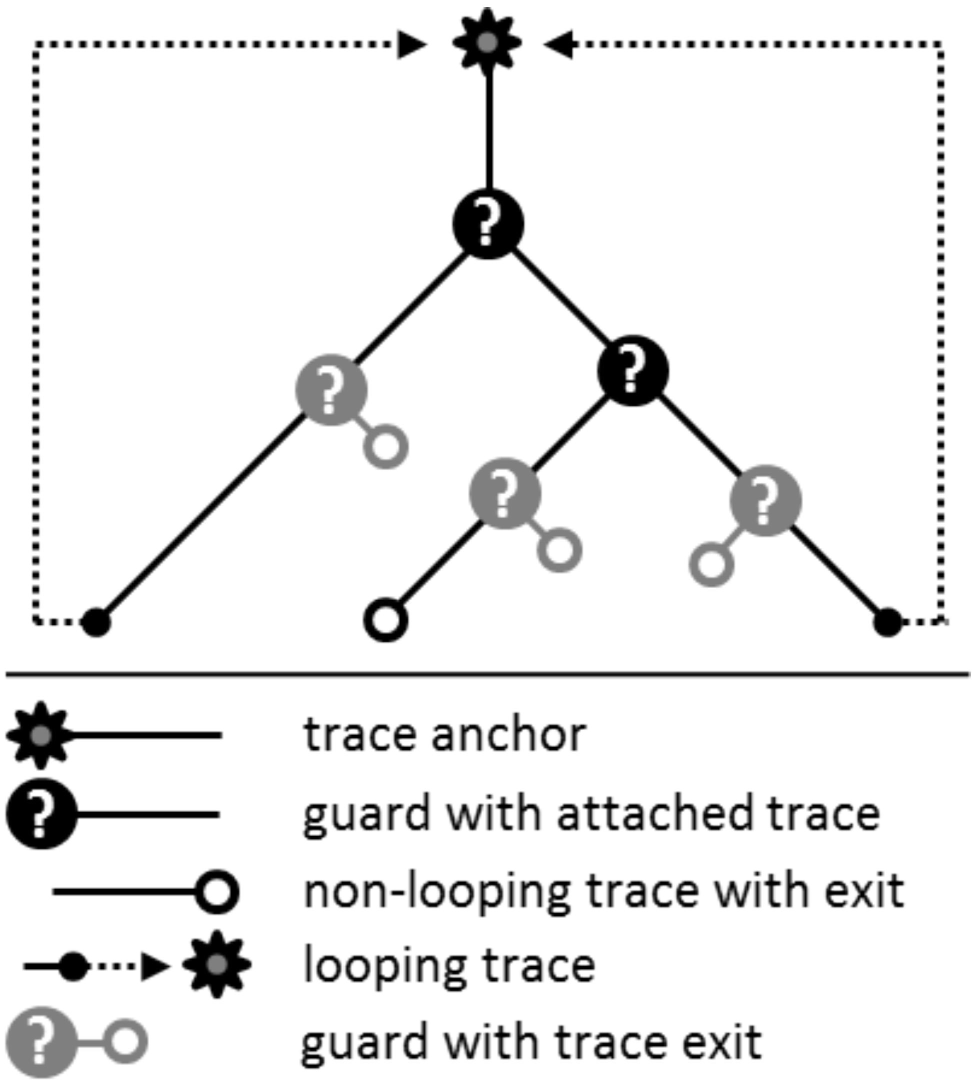
```
var sum = 0
for (var i = 0; i < 1000; i++) {
    if (i == 990) {
        sum += " Hello World "
    }
    sum += 1
}
print(sum)
```

"+" depend on operands, leading to lots of runtime checks

Optimize loop for an integer '+'

After "+" becomes a string, resume in un-optimized code

```
var sum = 0
for (var i = 0; i < 1000; i++) {
    if (i == 990) {
        sum += " Hello World "
    }
    sum += 1
}
print(sum)
```



Bebenita, Brandner, Fahndrich, Logozzo, Schulte, Tillmann, Venter. **SPUR: A Trace-Based JIT Compiler for CIL**. *OOPSLA10*

We go from this …

```
for (var n = 0; n < 1000; n++) {
  for (var n2 = 0; n2 < 1000; n2++) {
    for (var i = 0; i<a.length-1; i++) {
      var tmp = a[i];
      a[i] = a[i+1];
      a[i+1] = tmp;
    }
  }
}
```

⦿ **35 method calls, 129 guards, 224 total instructions**

back to…

```
double index = 0;

/* ... lots of hoisted code */
while (true) {
    if (index >= arrayIndex - 1) {
        // transfer
    }

    if (index + 1 >= validArrayIndex) {
        // transfer
    }

    /* bounds checks have been proven */
    var temp = array[(int) index];
    array[(int)index] = array[(int)index + 1];
    array[(int) index + 1] = temp;

    index += 1;
}
```

- 10 loop instructions, 2 loop guards!
- Performance 7x faster than the CLR based JScript, and slightly faster than V8

Static and dynamic type checking

Dynamic type checking is great:

- *anything goes, until it doesn't;*
- *a program can be run even when crucial pieces are missing*

Static type checking is great:

- *catches bugs earlier;*
- *enables faster execution.*

Can they co-exist in the same design?

Problem

```
class Foo{ def bar(x:Int) = x+1; }

a:Foo = Foo();

a.bar(Y);
```

assume no static type information available on `Y`

Idea: let the run-time check that **`Y`** is compatible with type `Int`.

When should this check be performed? How long does it take?

Run-time wrappers

```
class Ordered { def compare(o:Ordered):Int; }

fun sort (x:[Ordered]):[Ordered] = …

sort(X);
```

- Checking that **`X`** is an array of `Ordered` is linear time
- Arrays are mutable, so checking at invocation of `sort` is not enough.

Idea: add a wrapper around **`X`** that checks that it can respond to methods invoked on it

Compiled code:

```
sort(#[Ordered]#X)
```

Our design

Wrigstad, Zappa Nardelli, Lebresne, Ostlund, Vitek, **Integrating Typed and Untyped Code in a Scripting Language.** *POPL'10*

Our design principles

Permissive:

accept as many programs as possible

Modular:

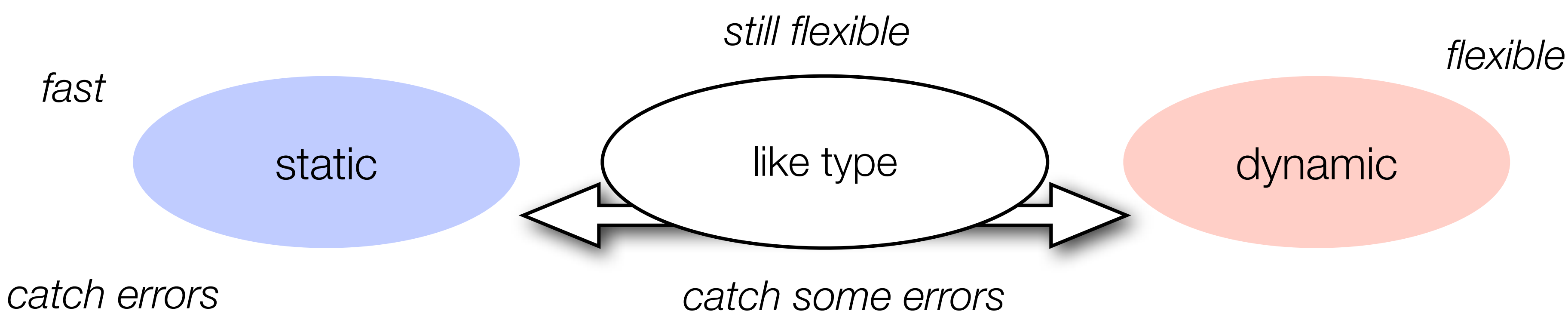
be as modular as possible

Reward good behavior:

reward programmer with performance or clear correctness guarantees

Our design

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



- For each class name `C`, add type `like C`
- Compiler checks that operations on `like C` variables are well-typed if the referred object had type `C`
- Does not restrict binding of `like C` variables,
  checks at run-time that invoked method exists

An example

```
class Point(var x:Int, var y:Int) {
  def getX():Int = x;
  def getY():Int = y;

  def move(p          ) {  x := p.getX(); y := p.getY(); }

}
```

Requirements:

1. Fields `x` and `y` declared `Int`

2. `move` accepts any object with `getX` and `getY` methods

like Point

```
class Point(var x:Int, var y:Int) {
  def getX():Int = x;
  def getY():Int = y;

  def move(p:like Point) {  x := p.getX(); y := p.getY(); }

}
```

Flexibility

```
class Point(var x:Int, var y:Int) {
  def getX():Int = x;
  def getY():Int = y;

  def move(p:like Point) {  x := p.getX(); y := p.getY(); }

}

class Coordinate(x:Int,y:Int) {
  def getX():Int = x;
  def getY():Int = y;
}

p = Point(0,0);
c = Coordinate(5,6);
p.move(c);
```

`move` runs fine if `c` has `getX`/`getY`

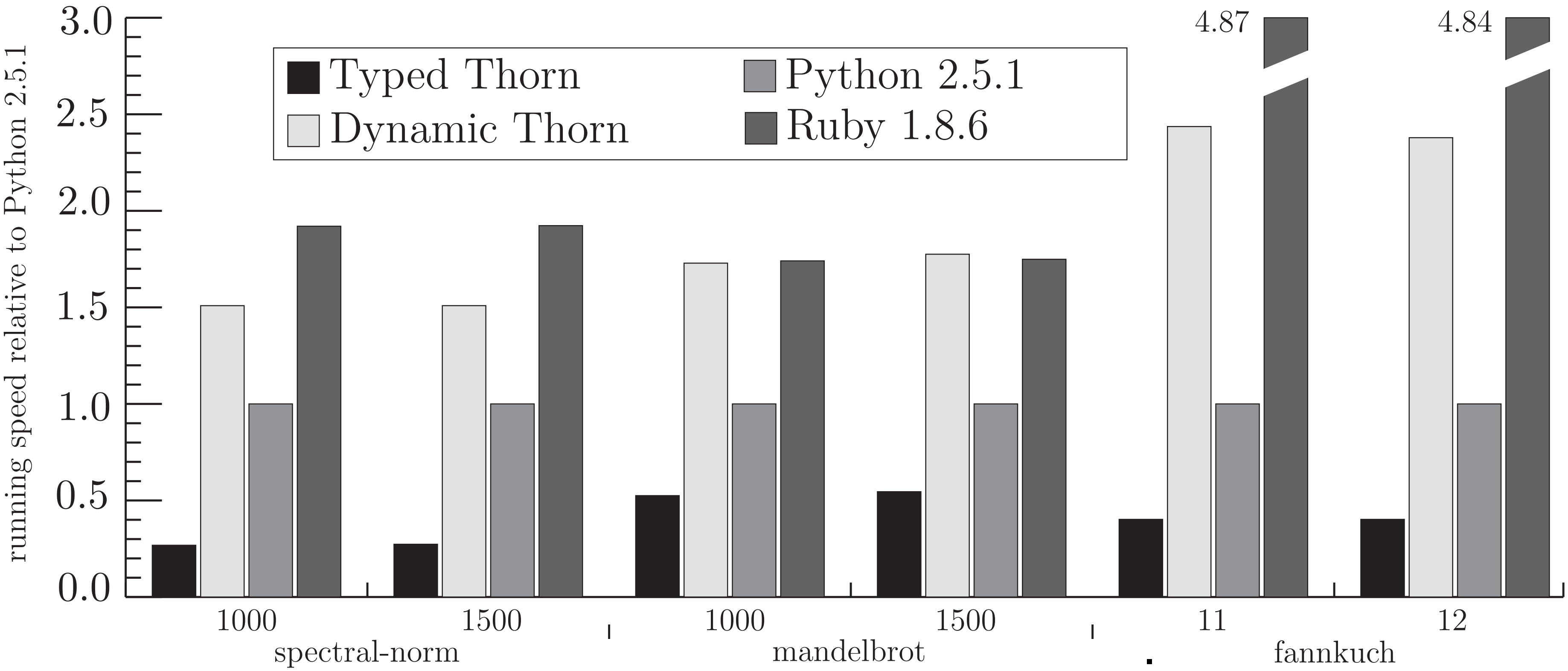
Checks

```
class Point(var x:Int, var y:Int) {
  def getX():Int = x;
  def getY():Int = y;

  def move(p:like Point) {
       x := p.getX(); y := p.getY();
       p.hog;
  }
}
```

Compile-time Error

`move` is type-checked under assumption that the argument is a `Point`

Rewards ...



Bloom, Field, Nystrom, Ostlund, Richards, Strnisa, Vitek, Wrigstad. **Thorn Robust, Concurrent, Extensible Scripting on the JVM**. OOPSLA'09

* Obtained with an older version of Thorn
* Benefits due to unboxing of numeric values
* Benchmarks are meaningless
* Still slower than Java

Related Work

Findler, Felleisen. Contracts for higher-order functions. 2002

Bracha. The Strongtalk Type System for Smalltalk. 2003

Gray, Findler, Flatt. Fine-grained interoperability through mirrors and contracts. 2005

Siek, Taha. Gradual typing for functional languages. 2006

Flanagan. ValleyScript: It's like static typing. 2007

Tobin-Hochstadt, Felleisen. Interlanguage migration: From scripts to programs. 2006

Wadler, Findler. Well-typed programs can't be blamed. 2009

Code Contracts

Contracts

- Precondition

  What I expect from the caller? e.g. *non-null parameter*

- Postcondition

  What I ensure to the caller? e.g. *value is non-negative*

- Object Invariant

  What holds in the stable states of an object? e.g. *field non-null*

Specify …

```
T Pop() {
  return this.a[--next];
}
```

Use code to specify code…

```
T Pop(){
Contract.Requires(!this.isEmpty);
Contract.Ensures(Contract.Result<T>()!= null);
  return this.a[--next];
}
```

CodeContracts

- Language agnostic
  Write contracts in your favorite dynamic language
- Compiler transparent
  Use your usual compiler
- Leverage IDE support
  Intellisense, squiggles, debugger …
- Runtime checker enables…
  Checking postconditions
  Contract inheritance, and contract interfaces

Fähndrich, Barnett, Logozzo: **Embedded contract languages.** SAC'10

Fahndrich, Logozzo. **Clousot: Static Contract Checking with Abstract Interpretation.** FoVeOOS'10

Conclusion

- Understanding the nature of dynamic program is essential to research in the field

- Dynamic languages can match statically compiled languages if we take advantage of adaptive techniques

- Dynamic languages need to be able to assert static properties such as types and invariants