Draining the Swamp

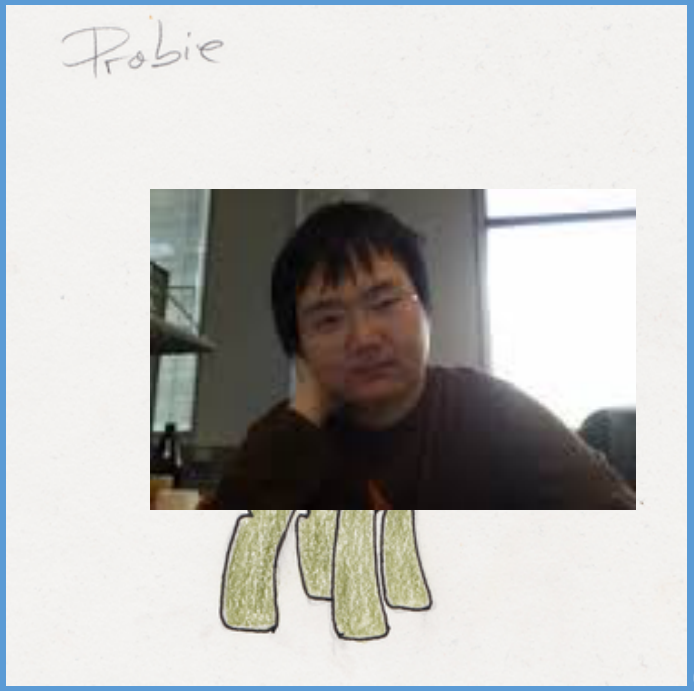
Micro Virtual Machines as Solid Foundation for Language Development


N.Y.C. SEWER
MADE IN CHINA

why languages suck

(and what we are doing about it…)

Tony Hosking
channeling Steve Blackburn
and Kunshan Wang



semantics

(can I make the computer do what I want it to?)


SIEMENS SYSTEM 4004

what could possibly go wrong?

brisbanetimes.com.au

Queensland

Health software brings risk of death

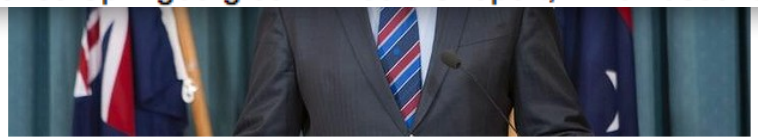
October 27, 2014

Tony Moore
brisbanetimes.com.au senior reporter
View more articles from Tony Moore
Email Tony

A new software program installed to manage medication doses at nine Queensland hospitals is likely to kill a patient within the next month, a Queensland Health risk report says.

Last Friday's report on the Metavision Intensive Care program advised the state government that the likelihood of the program causing preventable loss of life "is assessed as likely and expected to occur within the next month".

Health Minister Lawrence Springborg confirmed the report, which described the likelihood of



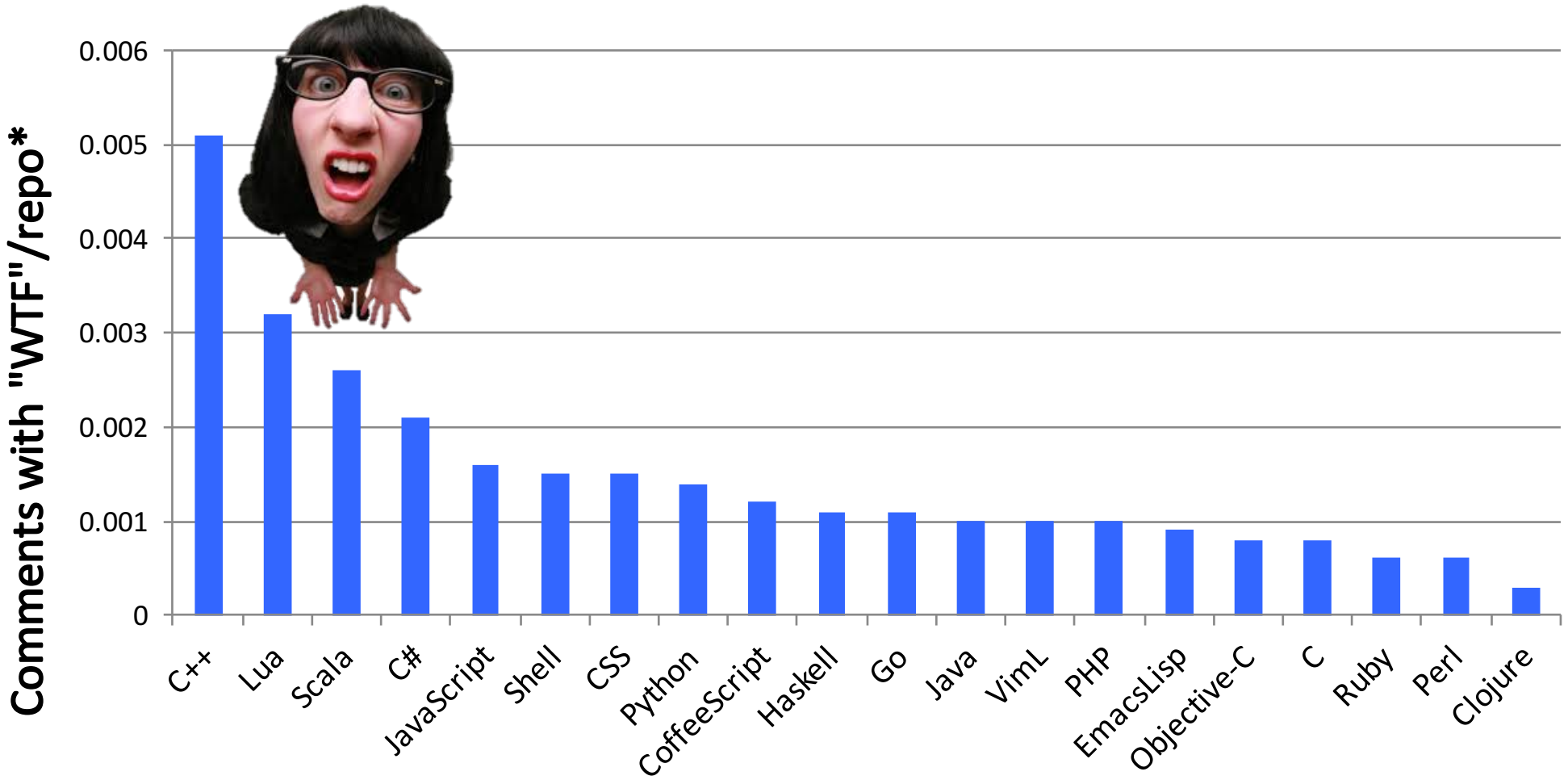
Health Minister Lawrence Springborg says measures have been put in place to prevent harm from a glitch in a program that handles medication doses. *Photo: Glenn Hunt*

A new software program installed to manage medication doses at nine Queensland hospitals is likely to kill a patient within the next month, a Queensland Health risk report says.

Last Friday's report on the Metavision Intensive Care program advised the state government that the likelihood of the program causing preventable loss of life "is assessed as likely and expected to occur within the next month".

Health Minister Lawrence Springborg confirmed the report, which described the likelihood of

but aren’t computer languages precise?



*these numbers are actually pretty meaningless, but the graph makes a point

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

let’s do a little programming…

a little js…

```
$ jsc
> [] + []

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

Source: Gary Bernhardt CodeMash 2012

a little more js...

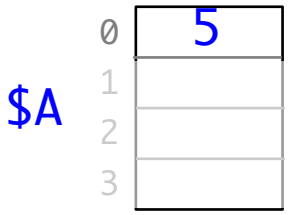
```
> Array(14)
,,,,,,,
> Array(14).join("foo")
foofoofoofoofoofoofoofoofoofoofoofoo
> Array(14).join("foo" + 1)
foo1foo1foo1foo1foo1foo1foo1foo1foo1foo1foo1foo1foo1foo1foo1
> Array(14).join("foo" - 1) + " Batman!"
NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaN Batman!
```

Source: Gary Bernhardt CodeMash 2012

some php...

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

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



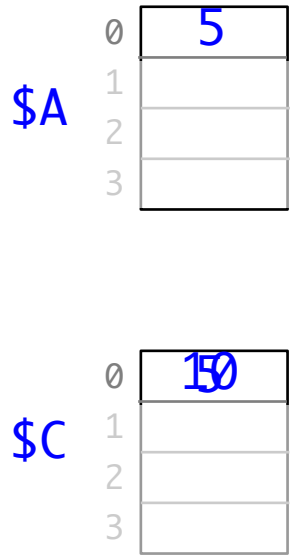

```
A[0]: 5
```

some php…

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

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

```
A[0]: 5
```



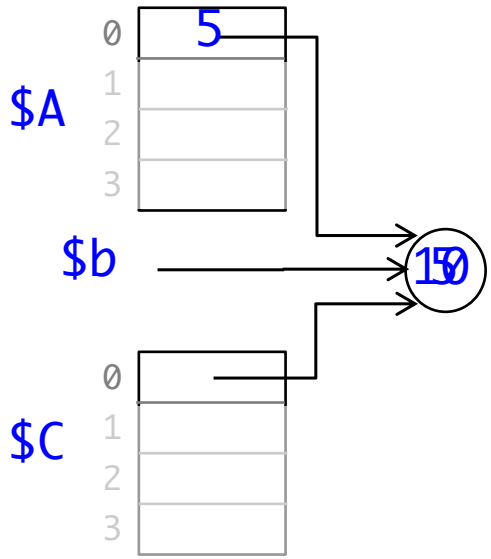
In PHP, variables are always assigned by value. That is to say, when you assign an expression to a variable, the entire value of the original expression is copied into the destination variable. This means, for instance, that after assigning one variable's value to another, changing one of those variables 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
```

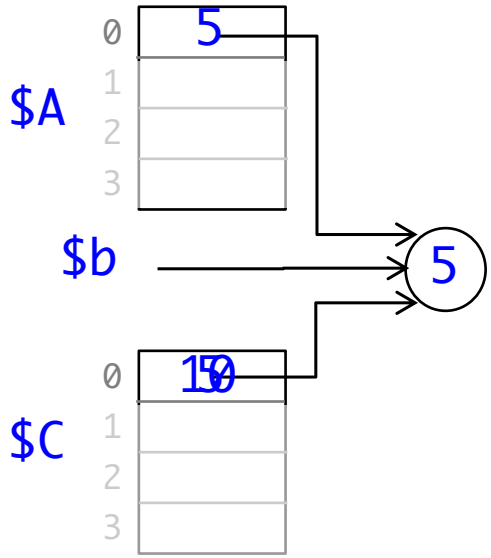


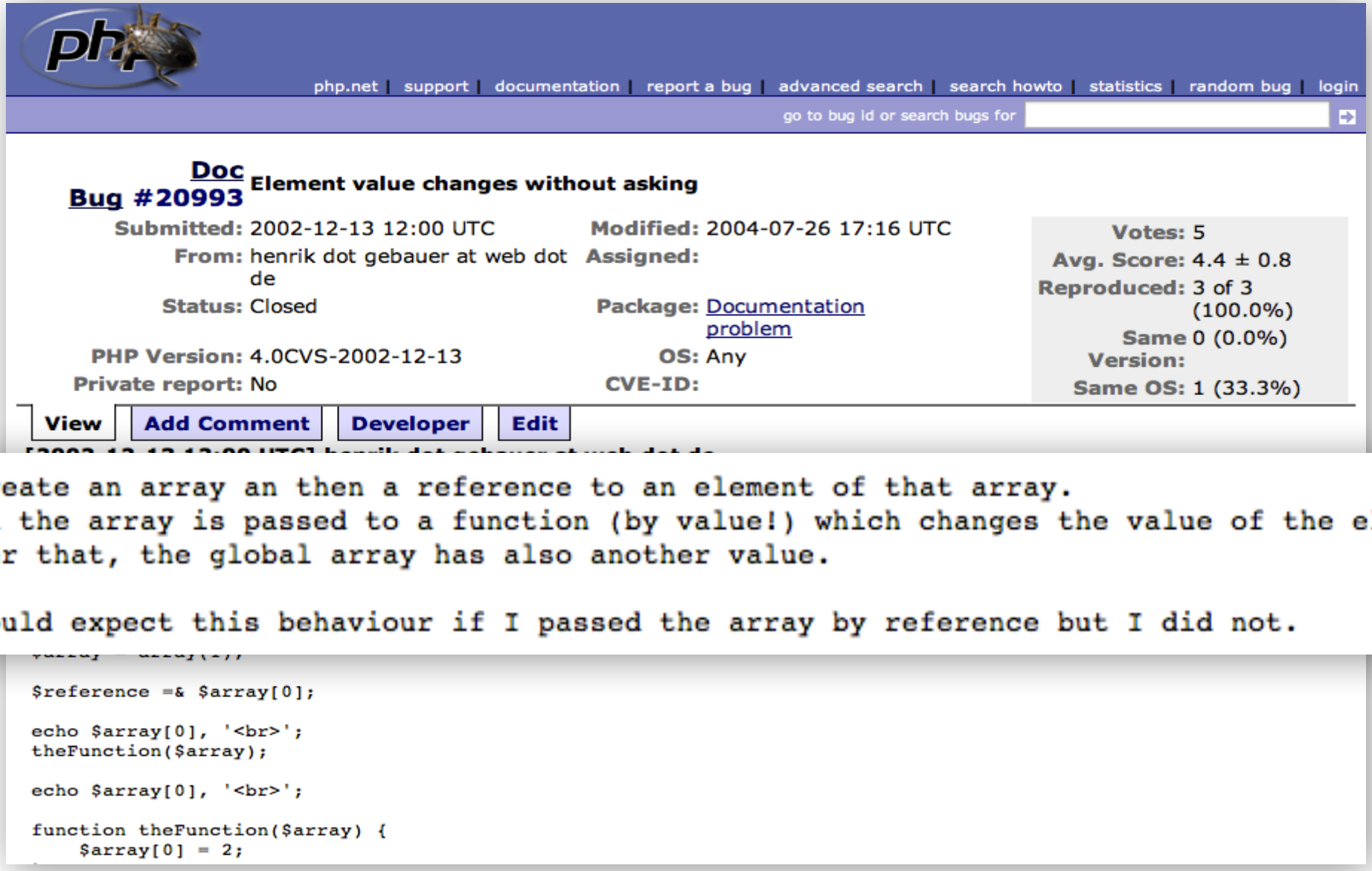
some php…

```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
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.
$reference =& $array[0];
echo $array[0], '
';
theFunction($array);
echo $array[0], '
';
function theFunction($array) {
$array[0] = 2;

Pull Requests

Add a Pull Request

History

All | Comments | Changes | Git/SVN commits | Related reports

[2002-12-13 12:42 UTC] msopacua@php.net

Verified and added testcase to CVS

[2002-12-13 12:50 UTC] moriyoshi@php.net

Verified with 4.2.3

[2002-12-13 14:51 UTC] moriyoshi@php.net

[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.

```
    $array = unserialize(serialize($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.

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

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`.

performance

(speed, correctness, reliability)

```c
int NUM = 111181111;

int is_prime(int n) {
   int i;
   for(i = 2; i < n; i++) {
     if (n % i == 0) {
       return 0;
     }
   }
   return 1;
 }
```

```python
NUM = 111181111

def is_prime(n):
    i = 2
    while i < n:
        if n % i == 0:
            return False
        i += 1

    return True
```

0.624s

15.609s

25X difference!

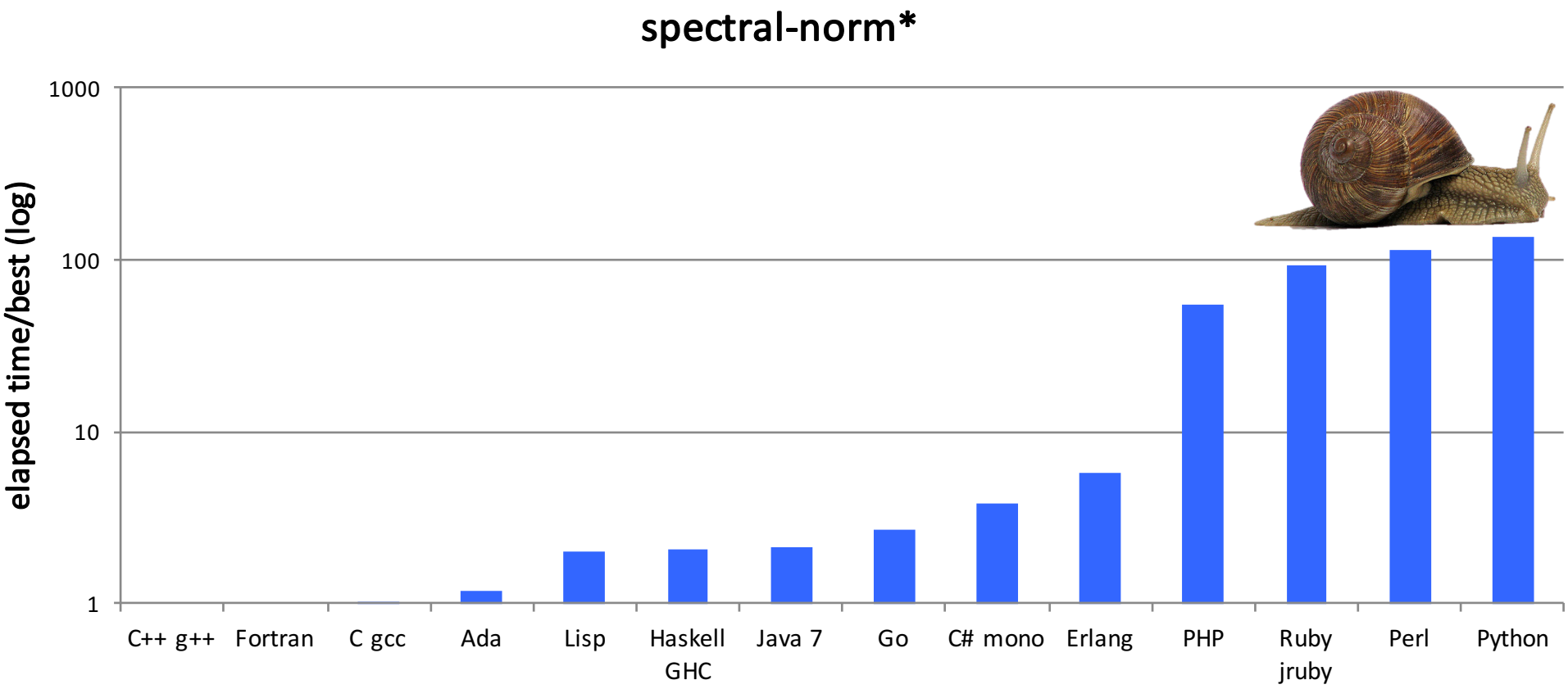
```
<?

Can be very useful for big projects, when you create a lot of objects that should stay in
memory. So GC can't clean them up and just wasting CPU time.

?>
```

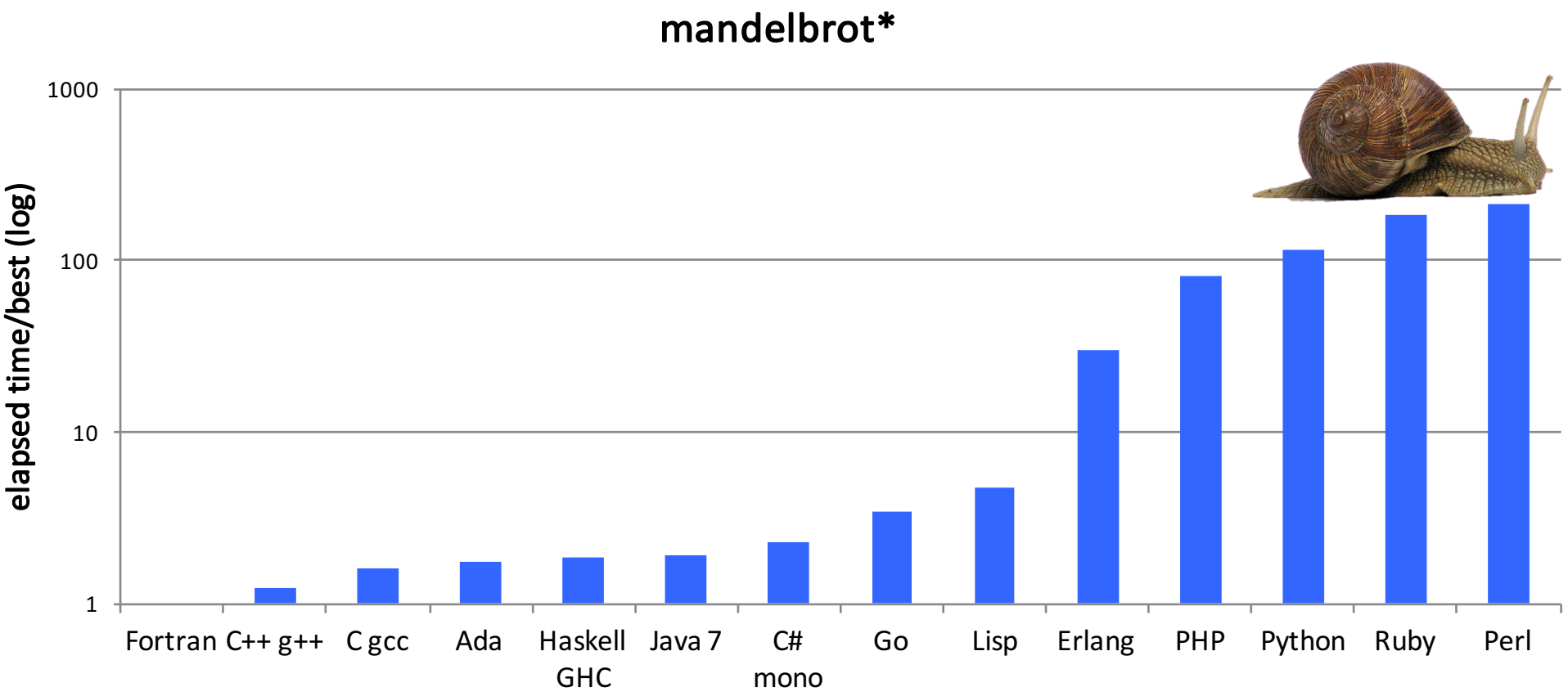
100% performance
improvement

http://php.net/manual/en/function.gc-disable.php



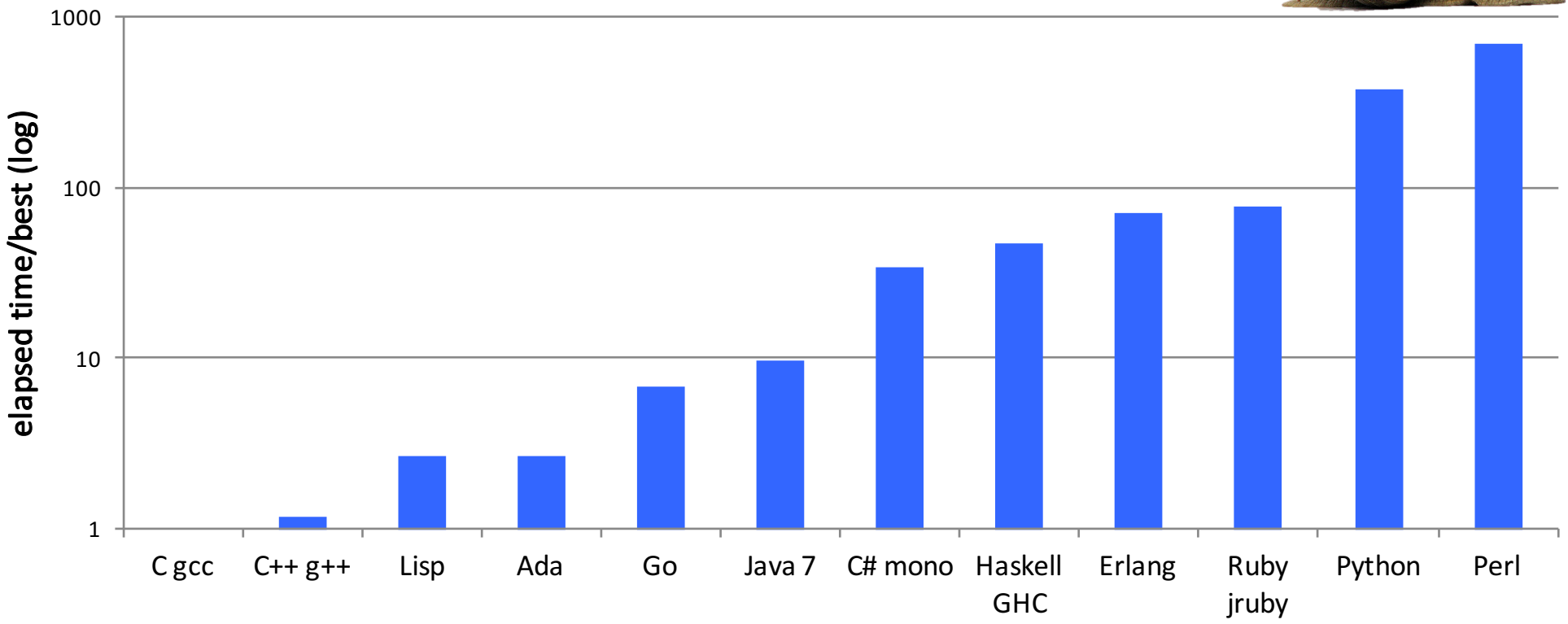
*these numbers are actually pretty meaningless, but the graph makes a point

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



*these numbers are actually pretty meaningless, but the graph makes a point

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



*these numbers are actually pretty meaningless, but the graph makes a point

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

Forbes | EYVoice: Realizing The Potential Of Latin American Women Entrepreneurs

Jay McGregor
Contributor

FOLLOW

I cover all aspects of technology and enterprise.
full bio →

Opinions expressed by Forbes Contributors are their own.

Share

Comment Now

CONFERENCES AND MORE

TECH 6/19/2014 @ 6:14AM | 9,619 views

Facebook Goes Down In Worldwide Outage

+ Comment Now + Follow Comments

Facebook suffered a worldwide outage this morning that stopped users across the globe from accessing the social network.

The website and smartphone apps were unable to reach Facebook from about 9am GMT, and resumed normal service at 9:24am GMT.

I contacted Facebook for information, but received no response. However it did release a statement to the Guardian: "Earlier this morning, we experienced an issue that prevented people from posting to Facebook for a brief period of time. We resolved the issue quickly, and we are now back to 100%. We're sorry for any inconvenience this may have caused".

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

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

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

Malware Spam Ask Learn WordPress

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 /?-***

the cost is formidable

maintenance cost
- *debugging wtf is costly*

performance cost of 10-100x
- *sluggish apps*
- *inefficient servers*

energy cost
- *energy bills for server farms*
- *battery performance on mobile devices*

why?

too hard

- difficult concepts
  - *concurrency* *(so punt and go for the GIL)*
  - *garbage collection* *(so punt and go for naïve RC)*
- difficult to engineer
  - *copy-on-write* *(so punt and call odd semantics a feature)*

BAKED IN

insidious orders of ignorance

when you don't know what you don't know

reference counting seems easy

cool, we can use it to implement copy-on-write!

its performance seems OK *(in our v00.1 VM)*

developers can deal with cycles *(programs will be small)*

swamp of naïve implementation

when you don't know how bad you are

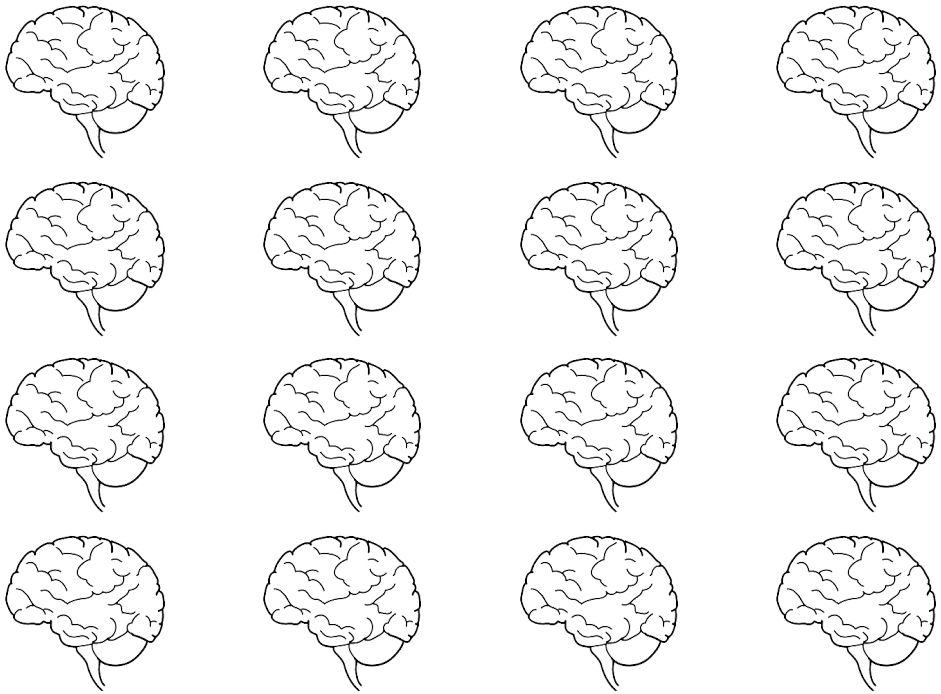
- *gc is not a problem for us* *(we measured it)*
- *a 24-byte object header is OK* *(we measured it)*
- *rc incs and decs don't cost much* *(we measured it)*

what's so hard?

jit

concurrency

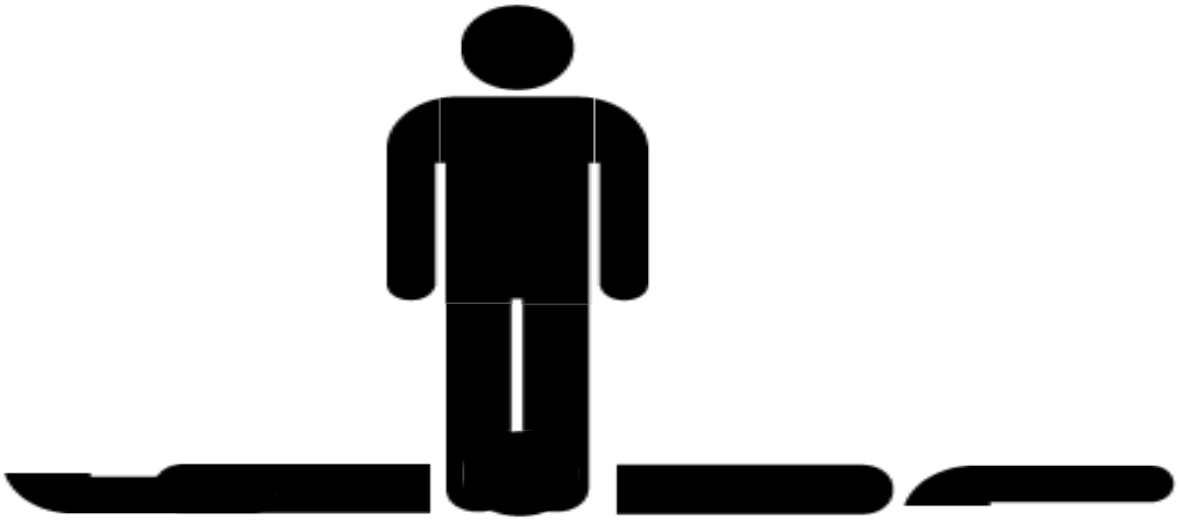
gc



jit + concurrency + gc

language/impl	jit?	concurrency	gc
CPython	interpreted	GIL	naive RC
PyPy	tracing jit	GIL	MMTk-like
Unladen Swallow	template jit	same as CPython	same as CPython
Jython	jvm byte-code	jvm	jvm
PHP	interpreted	?	naive RC
PHP (HHVM)	tracing jit	?	naive RC
Ruby (MRI)	interpreted	GIL	mark-sweep
Perl	interpreted	?	naive RC
Lua	interpreted	no threads	mark-sweep
LuaJIT	tracing JIT	same as Lua	same as Lua

the result...



the result...

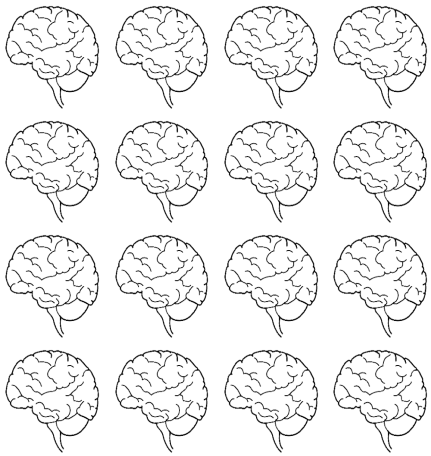
I don’t know how to stop it, there was never any intent to write a programming language [...] I have absolutely no idea how to write a programming language, I just kept adding the next logical step on the way.

Rasmus Lerdorf, creator of PHP

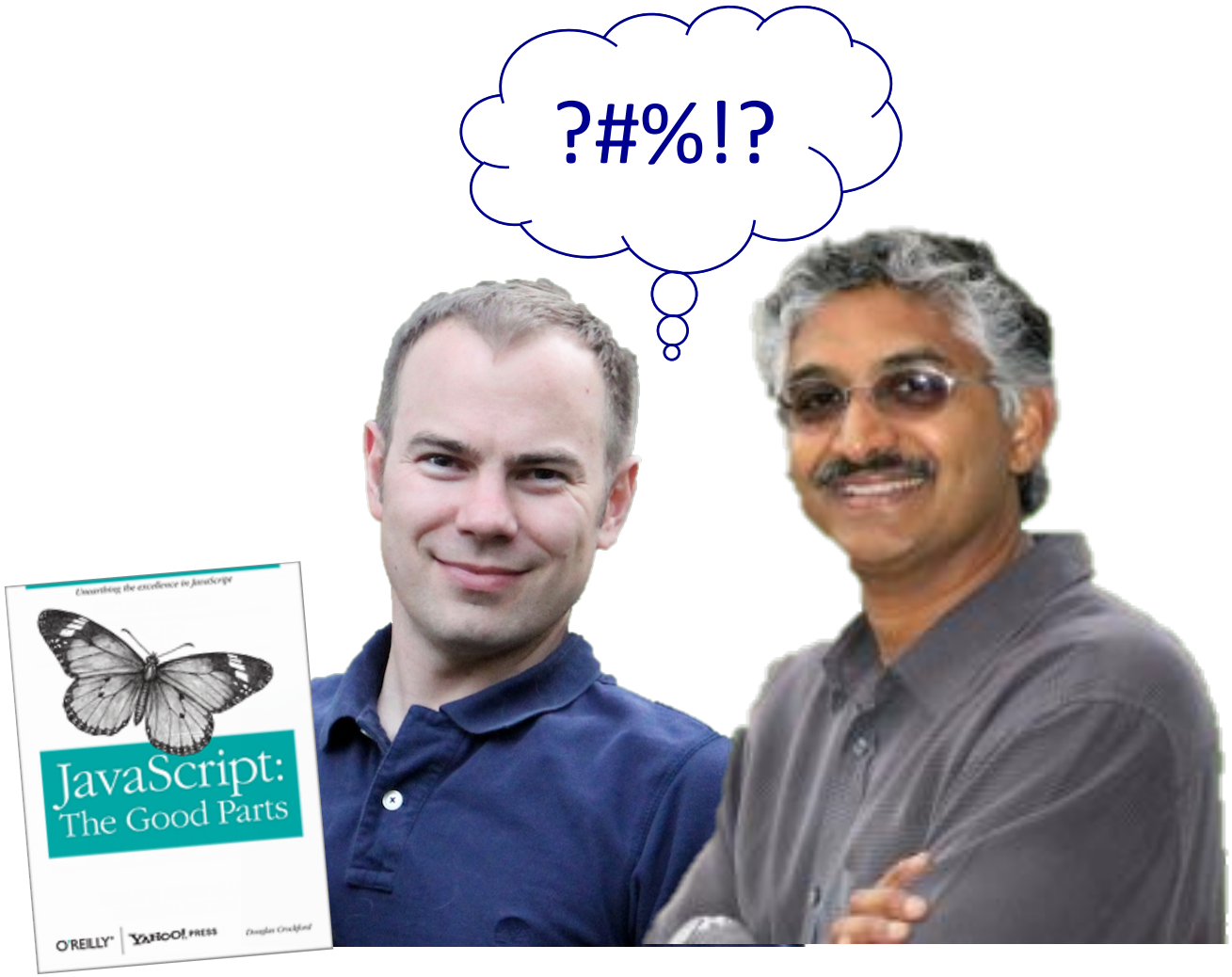
existing approaches

LuaJIT

HHVM




JRuby


?#%!?
JavaScript:
The Good Parts
O'REILLY
YAHOO! PRESS

how did we get here?

Moore's law

'Transistor density will double approximately every two years.'

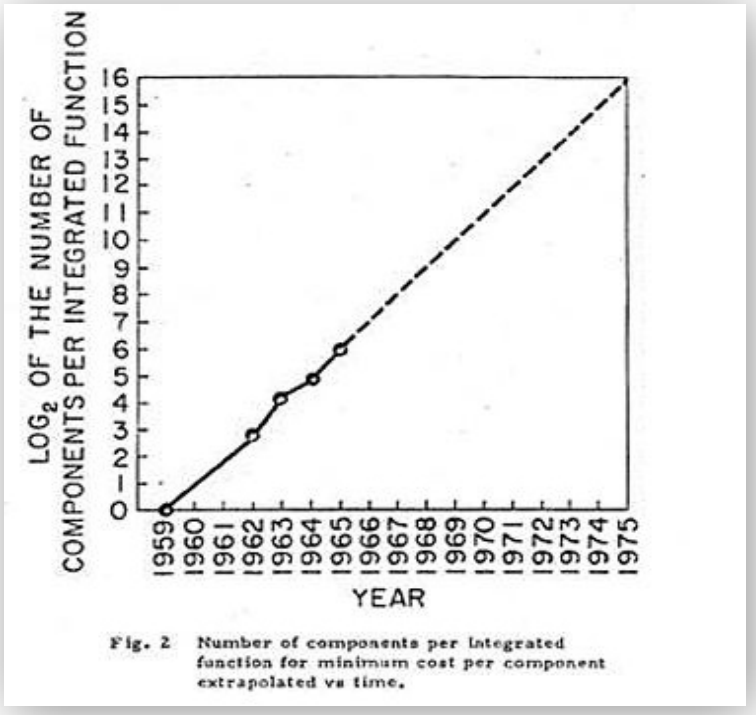


Fig. 2 Number of components per integrated function for minimum cost per component extrapolated vs time.

Dennard scaling

'As MOSFET features shrink, switching time *and* power consumption will fall proportionately.'

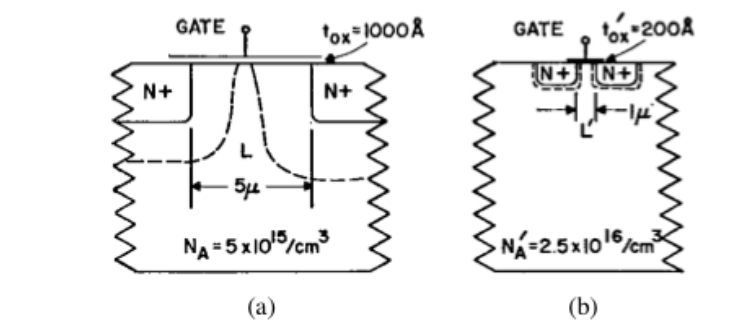


Fig. 1. Illustration of device scaling principles with $\kappa = 5$. (a) Conventional commercially available device structure. (b) Scaled-down device structure.

Dennard, Gaensslen, Yu, Rideout, Bassous and Leblanc, IEEE SSC, 1974

…however…

Moore's Law

'Transistor density will double approximat… two years.'

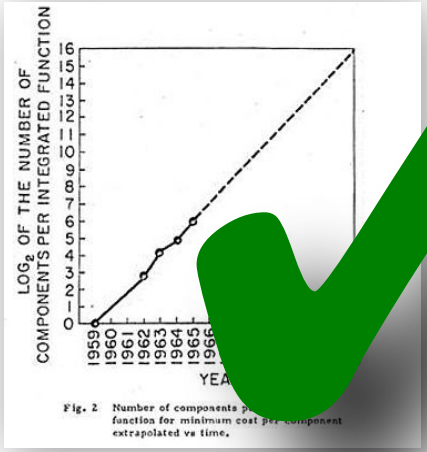
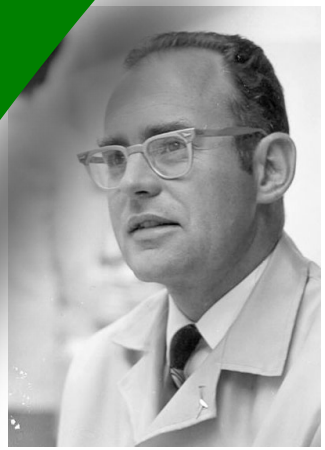


Fig. 2 Number of components p… function for minimum cost p… extrapolated vs time.



Dennard scaling

'As MOSFET features shrink, switchi… *and* power … ption will fall proportionately.'

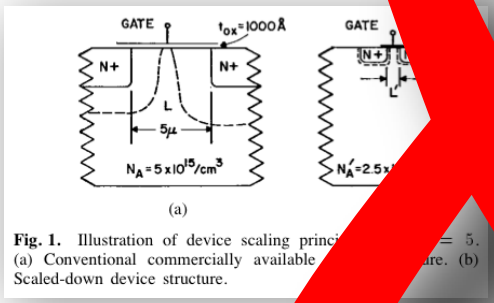
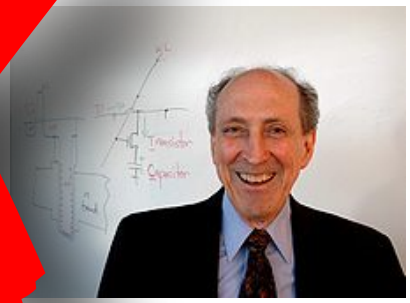
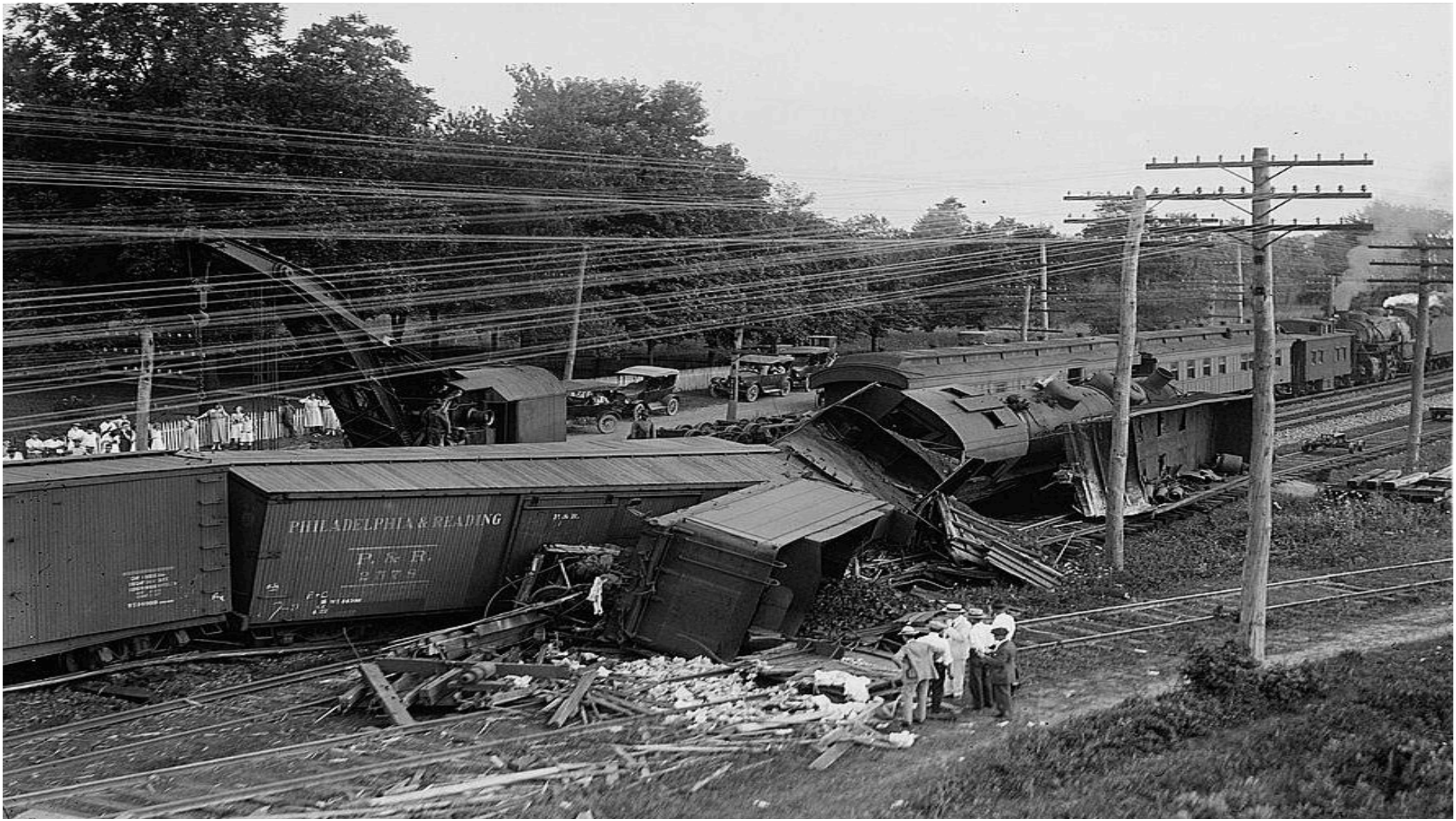


Fig. 1. Illustration of device scaling princi… = 5. (a) Conventional commercially available … ure. (b) Scaled-down device structure.



Dennard, Gaenssle… out, Bassous and Leblanc, IEEE SSC, 1974

how are things looking now?


PHILADELPHIA & READING
P. & R.

recap

recap

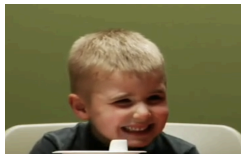
languages suck

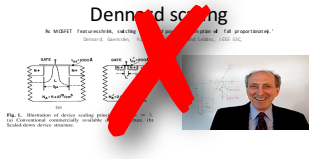
concurrency + gc + jit = headache

instant gratification + orders of ignorance

free lunch gone

heterogeneity is here





what to do?

microvm

microvm.org

ANU

Kunshan Wang

Yi Lin

Steve Blackburn

Tony Hosking

NICTA

Michael Norrish

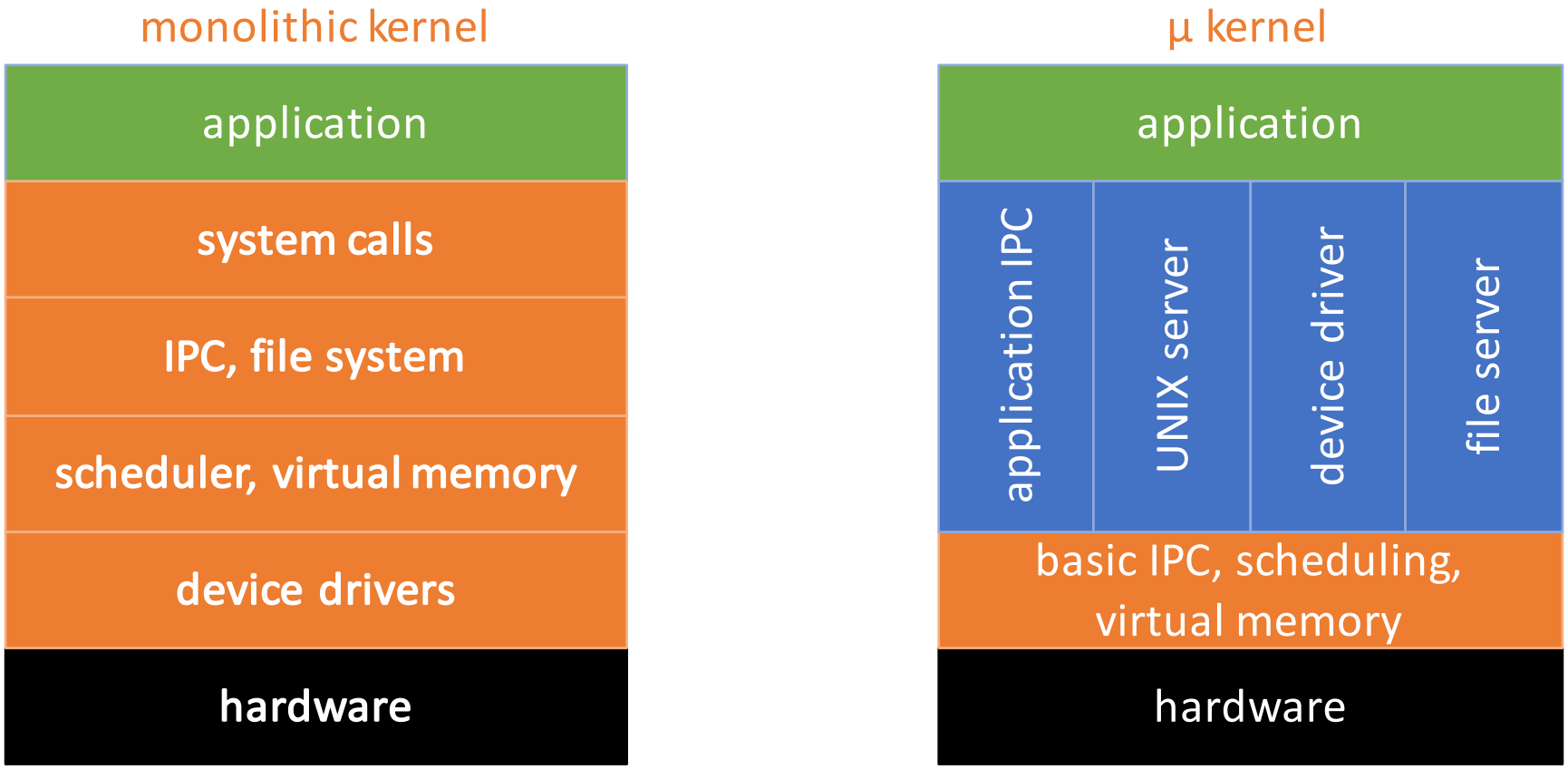
UMass

Eliot Moss

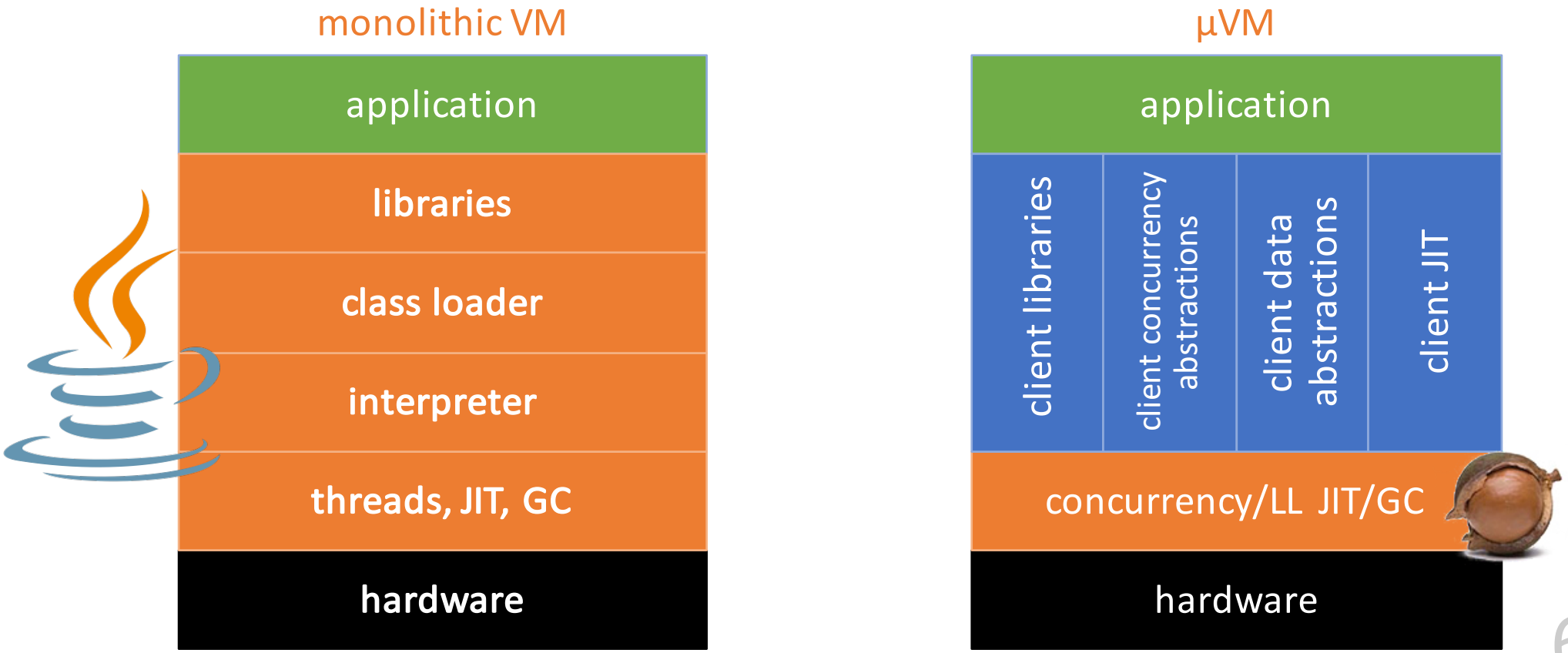
Tim Richards

Adam Nelson

analogous to microkernels



micro virtual machines



a microvm

very small

low-level

substrate for language implementation

goal of a formally verified implementation

just three abstractions

- memory
- concurrency
- architecture

only implement what is *essential*; client does the rest

compared to…

llvm

- *very small (no heavyweight opts)*
- *targets managed languages (dynamic, gc'd)*
- *concurrency and threading model built in*

jvm

- *very small (no heavyweight opts)*
- *much lower-level of abstraction*
- *ssa*

challenges

getting the abstraction right

- keeping it simple (yet rich & performant)
- support for speculative opt & osr
- right concurrency abstractions

getting portability right

- what to expose (endianness, word width)
- what to support (simd, htm)

The Mu Micro Virtual Machine

A Solid Foundation for Language Development

Useful Links

We coined the concept of *micro virtual machines* for the developement of managed languages.

"Mu" is the name of our specific micro virtual machine. This project has a specification and a reference implementation which are publicly available:

- Mu Specification (v2): canonical information
- Mu Reference Implementation (v2): an implementation of the Mu spec in Scala
- Mu Tutorial (current): a step-by-step guide
- The Micro Virtual Machine Project on GitHub: this includes many sub-projects
- High-level Issue Tracker: general discussions and questions not specific to any sub-project

Overview

A large fraction of today's software is written in *managed languages*. These languages increase

README.rest

Mu Specification

This document aims to provide a detailed description of Mu, a micro virtual machine, including its architecture, instruction set and type system.

This branch uses the goto-with-values form. The previous branch using SSA form with PHI nodes is in the phi branch.

Main specification:

formal verification (a la seL4)

a central goal of the project

has influenced design

reinforced simplicity, clarity

status (June 2016)

full spec v 0.02
- formal spec underway

full reference implementation v 0.02

performance implementation v 0.02 underway
- implemented in Rust (see ISMM'16)

client language bindings
- Haskell -> bf (working), GHC (underway)
- RPython -> python (underway), running SOM interpreter

details

types, threads & stacks, native interface

```
.typedef @i64 = int<64>
.const @I64_1 <@i64> = 1
.funcsig @i_i = @i64 (@i64)
.funcdef @fac VERSION @fac_v1 <@i_i> {
    %entry(<@i64> %n):
        %zero = EQ <@i64> %n @I64_0
        BRANCH2 %zero %iszero() %notzero(%n)
    %iszero():
        RET @I64_1
    %notzero(<@i64> %n):
        %nm1 = SUB <@i64> %n @I64_1
        %rec = CALL <@i_i> @factorial_rec (%nm1)
        %result = MUL <@i64> %n %rec
        RET %result
}
```

types

C

int float double

void* void (*)()

__m128

struct {…} struct {…, T[]} T[n]

NO REF TYPES

μ

int<n> float double

uptr<T> ufuncptr<Sig>

vector<T n>

struct<T …> hybrid<F… V> array<T n>

ref<T> iref<T>

funcref<Sig> stackref

threadref

threads and stacks

distinct abstractions

threads can swap stacks:

```
%result = SWAPSTACK %stack
                    RET_WITH <T>
                    PASS_VALUES <U> (%val)
```

continuations, coroutines, lightweight threads

threads and stacks

threads can trap:

%trap2 = TRAP <T> %exc() KEEPALIVE(%a %b)

threads trap at **enabled** watchpoints

WATCHPOINT 42 <> %dis() %ena() %exc()
KEEPALIVE(%a)

introspection API for stacks

```
new_stack(Function f)
top_frame(Stack s)
next_frame(Frame frm)
enable_watchpoint(int i)
```

```
cur_func(Frame frm)
cur_inst(Frame frm)
pop_frames_to(Frame frm)
push_frame(Function f)
```

threads and stacks

swapstack + traps + introspection:

- OSR
- guards
- dynamic optimization
- profiling

atomics and synchronization

C11-style atomics

similar to LLVM

also futex

basis for client synchronization abstractions

unsafe native interface

(raw, untraced) pointers

- `uptr<T>`
- `LOAD/STORE` via pointers

function pointers (to C functions)

- `ufuncptr<Sig>`
- `CCALL`: call native function directly

other work

transactional memory primitives
- abstract logging and rollback
- exploits best-effort HTM where possible

formalisation effort in HOL4
- verification of Mu implementations
- well-defined client semantics

thank you


questions?