Demystifying Magic
High-Level Low-Level Programming

Daniel Frampton
Australian National University

Steve Blackburn
Australian National University

Perry Cheng
IBM Research

Robin Garner
Australian National University

David Grove
IBM Research

Eliot Moss
U. Massachusetts, Amherst

Sergey Salishev
St Petersburg State U., Russia

Abstraction

The Problem

- Systems programmers:
  - strive for reliability, security, maintainability
  - depend on performance and transparent access to low-level primitives
- Abstraction:
  - Enables the former
  - Typically obstructs the latter

The Goal

Abstraction without guilt.

[Ken Kennedy]

Revolution

Why

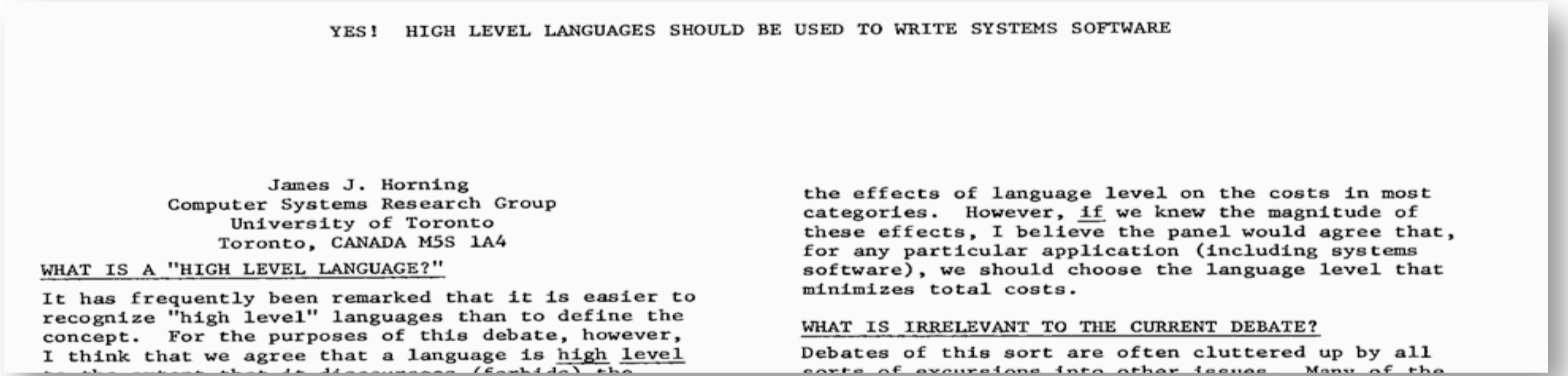
"In order to manage the complexity of software systems, I find it necessary to partition my concerns, and deal with them separately. A major set of concerns is the efficient allocation of a whole variety of resources, but I don't worry about all resources all the time. Efficient allocation of registers is important, but quite separable from memory management, … file management, etc. While I am programming any one of these I would like to take the others for granted."

Why Not

"A person designing and implementing an operating system, … is trying to make a given collection of processors, storage media, and i/o devices work together reliably, efficiently, and conveniently. The operation of these devices is his problem. It is unclear to us why he would ever want to obscure the exact nature of this hardware behind the constructs of a higher-level language."

Yes! High level languages should be used to write systems software

James Horning, ACM Annual Conference/Annual Meeting
Proceedings of the 1975 Annual Conference


YES! HIGH LEVEL LANGUAGES SHOULD BE USED TO WRITE SYSTEMS SOFTWARE

James J. Horning
Computer Systems Research Group
University of Toronto
Toronto, CANADA M5S 1A4

WHAT IS A "HIGH LEVEL LANGUAGE?"

It has frequently been remarked that it is easier to recognize "high level" languages than to define the concept. For the purposes of this debate, however, I think that we agree that a language is high level

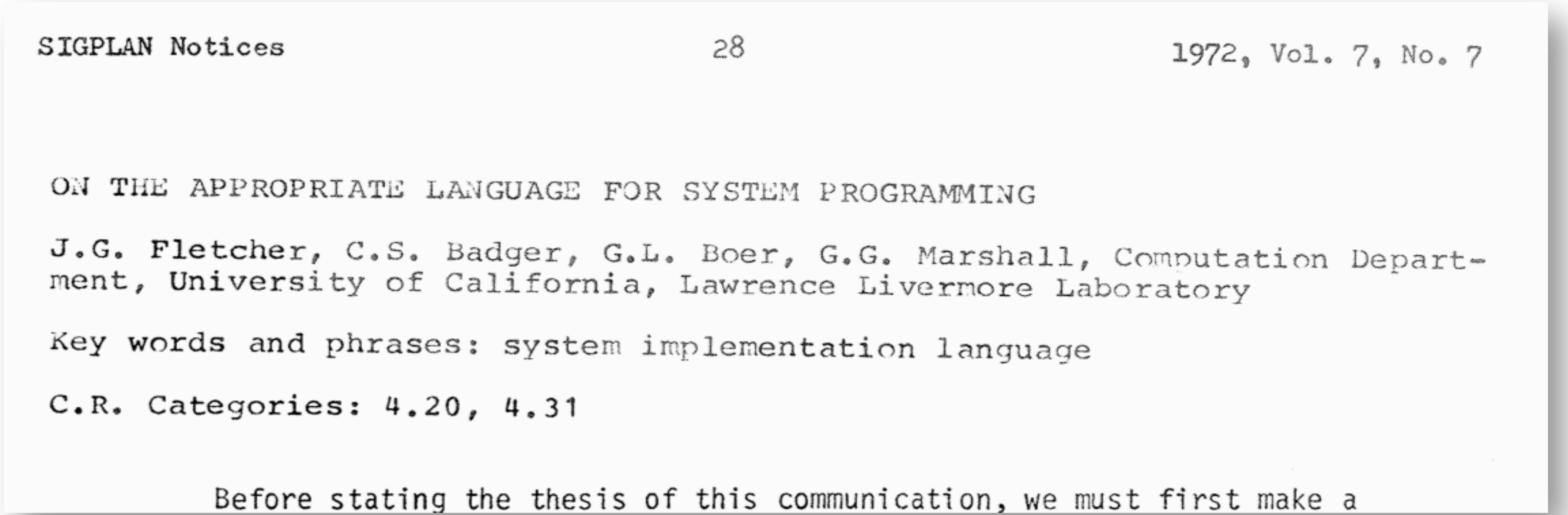
the effects of language level on the costs in most categories. However, if we knew the magnitude of these effects, I believe the panel would agree that, for any particular application (including systems software), we should choose the language level that minimizes total costs.

WHAT IS IRRELEVANT TO THE CURRENT DEBATE?

Debates of this sort are often cluttered up by all

On the appropriate language for system programming

ACM SIGPLAN Notices, Volume 7, Number 7, 1972


SIGPLAN Notices 28 1972, Vol. 7, No. 7

ON THE APPROPRIATE LANGUAGE FOR SYSTEM PROGRAMMING

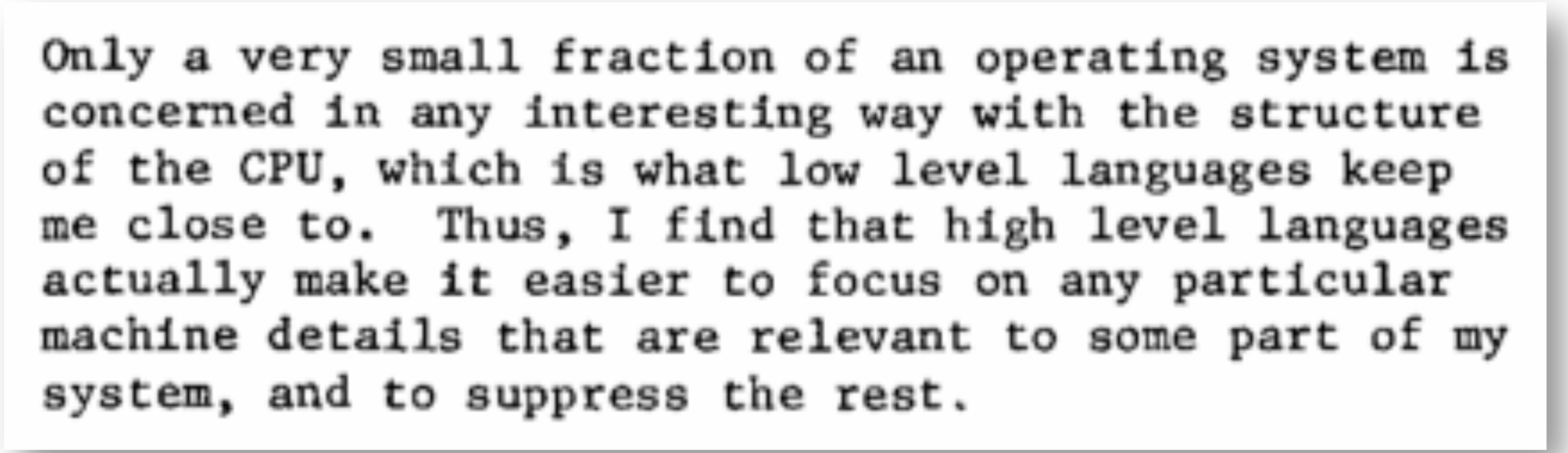
J.G. Fletcher, C.S. Badger, G.L. Boer, G.G. Marshall, Computation Department, University of California, Lawrence Livermore Laboratory

Key words and phrases: system implementation language

C.R. Categories: 4.20, 4.31

Before stating the thesis of this communication, we must first make a

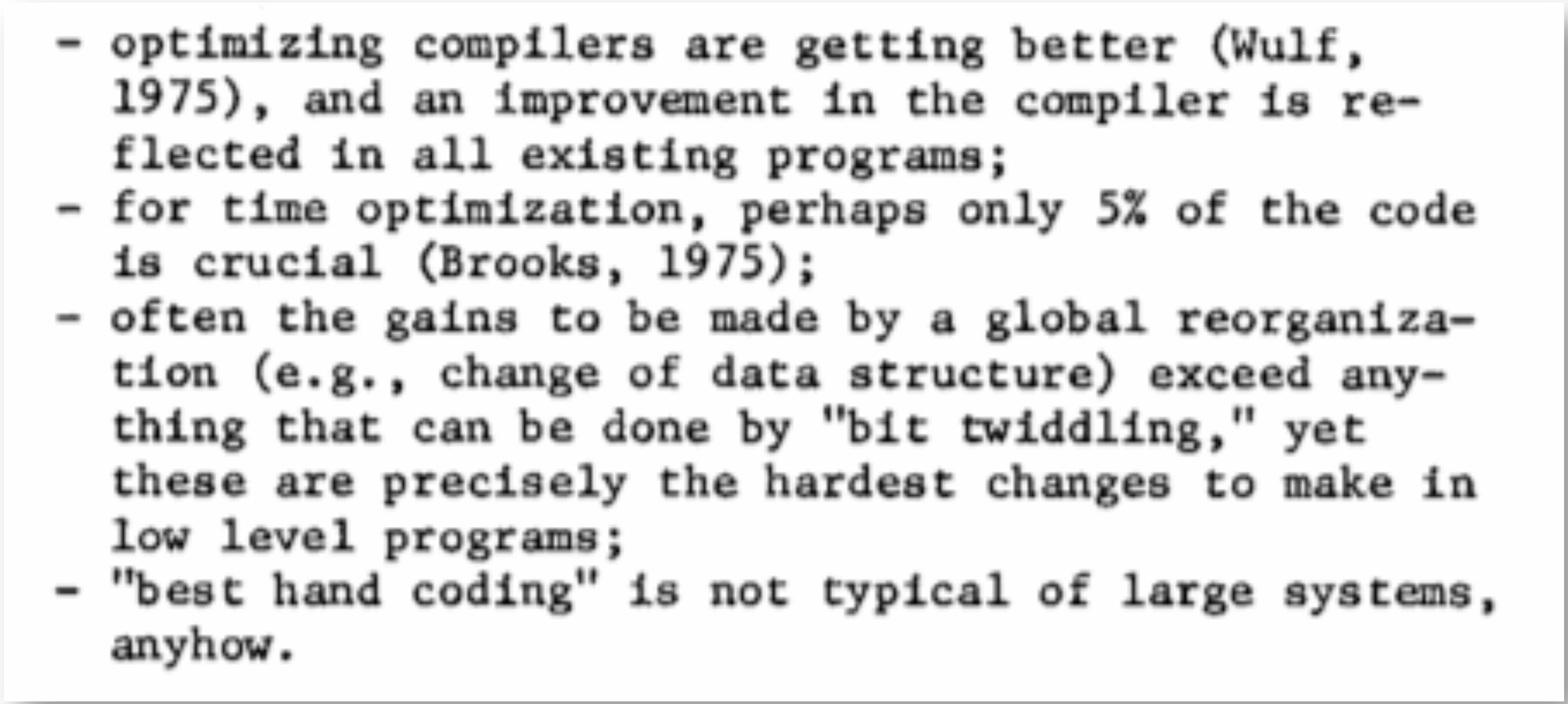
Why (1975)



Only a very small fraction of an operating system is concerned in any interesting way with the structure of the CPU, which is what low level languages keep me close to. Thus, I find that high level languages actually make it easier to focus on any particular machine details that are relevant to some part of my system, and to suppress the rest.

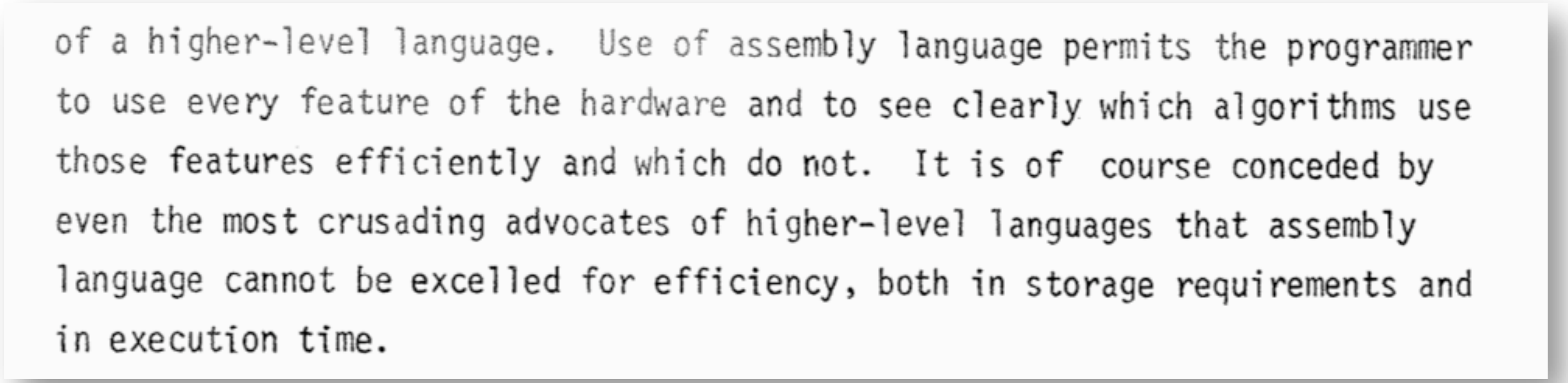
James J. Horning, Yes! High level languages should be used to write systems software. **ACM Annual Conference/Annual Meeting Proceedings of the 1975 Annual Conference.**

Why (1975)


- optimizing compilers are getting better (Wulf, 1975), and an improvement in the compiler is reflected in all existing programs;
- for time optimization, perhaps only 5% of the code is crucial (Brooks, 1975);
- often the gains to be made by a global reorganization (e.g., change of data structure) exceed anything that can be done by "bit twiddling," yet these are precisely the hardest changes to make in low level programs;
- "best hand coding" is not typical of large systems, anyhow.

James J. Horning, Yes! High level languages should be used to write systems software. **ACM Annual Conference/Annual Meeting Proceedings of the 1975 Annual Conference.**

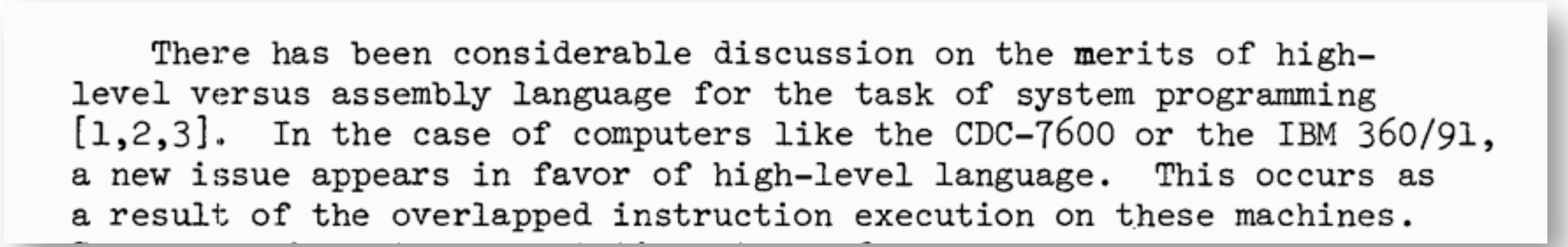
Why Not (1972)


of a higher-level language. Use of assembly language permits the programmer to use every feature of the hardware and to see clearly which algorithms use those features efficiently and which do not. It is of course conceded by even the most crusading advocates of higher-level languages that assembly language cannot be excelled for efficiency, both in storage requirements and in execution time.

J.G. Fletcher, On the appropriate language for system programming.
ACM SIGPLAN Notices, Volume 7, Number 7, 1972

Why The 70s Shift to C?

- Better language design
- Better language implementation
- More complex software
- More complex hardware
- More hardware (portability)



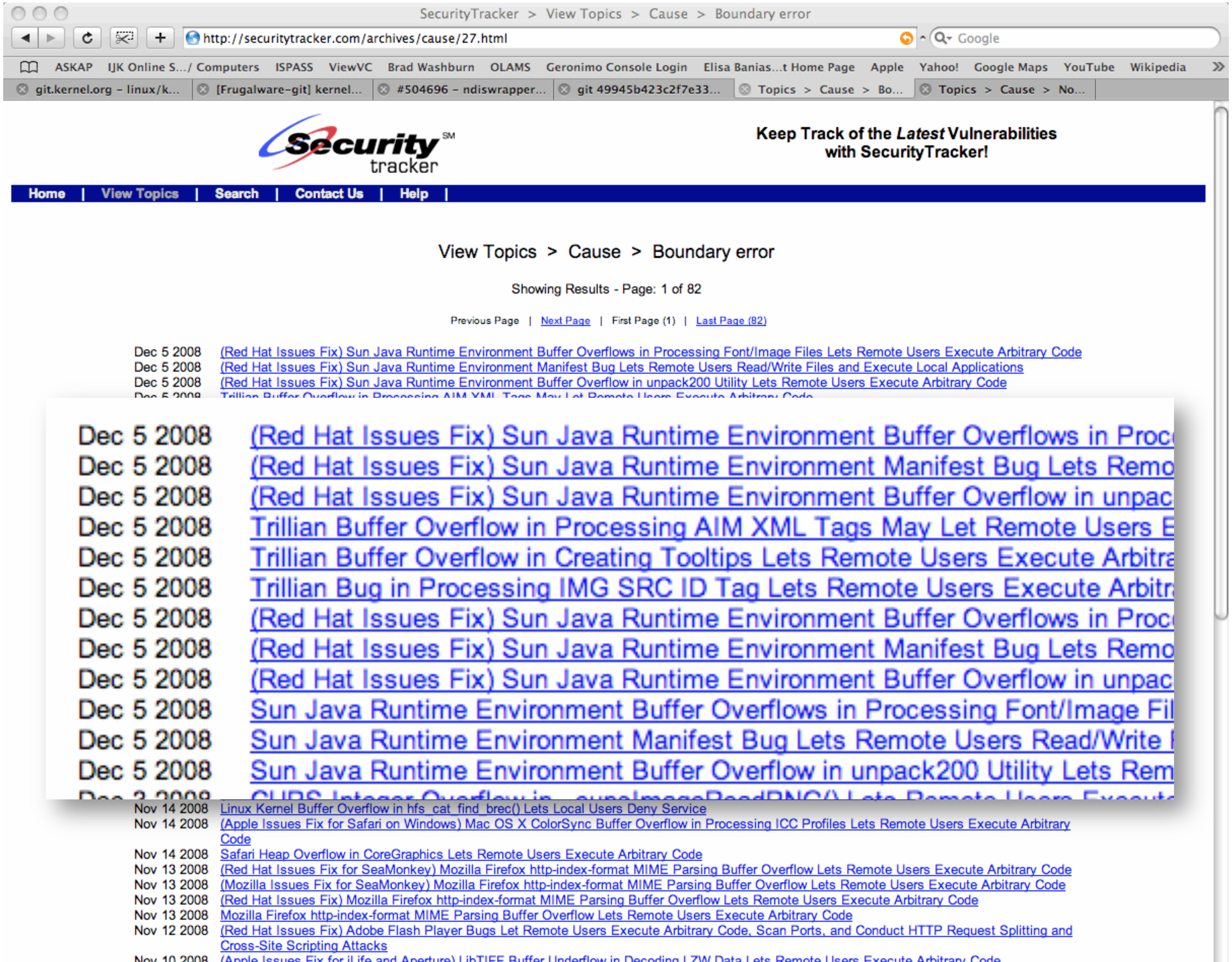
There has been considerable discussion on the merits of high-level versus assembly language for the task of system programming [1,2,3]. In the case of computers like the CDC-7600 or the IBM 360/91, a new issue appears in favor of high-level language. This occurs as a result of the overlapped instruction execution on these machines.

Charles Landau, On high-level languages for system programming ACM SIGPLAN Notices, Volume 11, Issue 1,1976

Revolution II ?

A New Revolution?

- Change continues
  - heterogeneous multi-core?
  - more complex software
- Higher-level languages
  - type safe
  - memory safe
  - strong abstractions over hardware


SecurityTracker > View Topics > Cause > Boundary error
http://securitytracker.com/archives/cause/27.html
Google
ASKAP
IJK Online S.../ Computers
ISPASS
ViewVC
Brad Washburn
OLAMS
Geronimo Console Login
Elisa Banias...t Home Page
Apple
Yahoo!
Google Maps
YouTube
Wikipedia
git.kernel.org - linux/k...
[Frugalware-git] kernel...
#504696 - ndiswrapper...
git 49945b423c2f7e33...
Topics > Cause > Bo...
Topics > Cause > No...
Security tracker SM
Keep Track of the *Latest* Vulnerabilities
with SecurityTracker!
Home | View Topics | Search | Contact Us | Help |

View Topics > Cause > Boundary error

Showing Results - Page: 1 of 82

Previous Page | Next Page | First Page (1) | Last Page (82)

Dec 5 2008 (Red Hat Issues Fix) Sun Java Runtime Environment Buffer Overflows in Processing Font/Image Files Lets Remote Users Execute Arbitrary Code
Dec 5 2008 (Red Hat Issues Fix) Sun Java Runtime Environment Manifest Bug Lets Remote Users Read/Write Files and Execute Local Applications
Dec 5 2008 (Red Hat Issues Fix) Sun Java Runtime Environment Buffer Overflow in unpack200 Utility Lets Remote Users Execute Arbitrary Code
Dec 5 2008 Trillian Buffer Overflow in Processing AIM XML Tags May Let Remote Users Execute Arbitrary Code

Dec 5 2008 (Red Hat Issues Fix) Sun Java Runtime Environment Buffer Overflows in Proc
Dec 5 2008 (Red Hat Issues Fix) Sun Java Runtime Environment Manifest Bug Lets Remo
Dec 5 2008 (Red Hat Issues Fix) Sun Java Runtime Environment Buffer Overflow in unpac
Dec 5 2008 Trillian Buffer Overflow in Processing AIM XML Tags May Let Remote Users E
Dec 5 2008 Trillian Buffer Overflow in Creating Tooltips Lets Remote Users Execute Arbitra
Dec 5 2008 Trillian Bug in Processing IMG SRC ID Tag Lets Remote Users Execute Arbitr
Dec 5 2008 (Red Hat Issues Fix) Sun Java Runtime Environment Buffer Overflows in Proc
Dec 5 2008 (Red Hat Issues Fix) Sun Java Runtime Environment Manifest Bug Lets Remo
Dec 5 2008 (Red Hat Issues Fix) Sun Java Runtime Environment Buffer Overflow in unpac
Dec 5 2008 Sun Java Runtime Environment Buffer Overflows in Processing Font/Image Fil
Dec 5 2008 Sun Java Runtime Environment Manifest Bug Lets Remote Users Read/Write
Dec 5 2008 Sun Java Runtime Environment Buffer Overflow in unpack200 Utility Lets Rem
Dec 3 2008 CUPS Integer Overflow in _cupsImageReadPNG() Lets Remote Users Execute

Nov 14 2008 Linux Kernel Buffer Overflow in hfs_cat_find_brec() Lets Local Users Deny Service
Nov 14 2008 (Apple Issues Fix for Safari on Windows) Mac OS X ColorSync Buffer Overflow in Processing ICC Profiles Lets Remote Users Execute Arbitrary Code
Nov 14 2008 Safari Heap Overflow in CoreGraphics Lets Remote Users Execute Arbitrary Code
Nov 13 2008 (Red Hat Issues Fix for SeaMonkey) Mozilla Firefox http-index-format MIME Parsing Buffer Overflow Lets Remote Users Execute Arbitrary Code
Nov 13 2008 (Mozilla Issues Fix for SeaMonkey) Mozilla Firefox http-index-format MIME Parsing Buffer Overflow Lets Remote Users Execute Arbitrary Code
Nov 13 2008 (Red Hat Issues Fix) Mozilla Firefox http-index-format MIME Parsing Buffer Overflow Lets Remote Users Execute Arbitrary Code
Nov 13 2008 Mozilla Firefox http-index-format MIME Parsing Buffer Overflow Lets Remote Users Execute Arbitrary Code
Nov 12 2008 (Red Hat Issues Fix) Adobe Flash Player Bugs Let Remote Users Execute Arbitrary Code, Scan Ports, and Conduct HTTP Request Splitting and Cross-Site Scripting Attacks
Nov 10 2008 (Apple Issues Fix for iLife and Aperture) LibTIFF Buffer Underflow in Decoding LZW Data Lets Remote Users Execute Arbitrary Code

Spot The Bug…

```
logmessage("DEBUG: fd: %d select(): fd %d is ready for read\n", sockfd,
sockfd);

         /* read as much data as we can */
    rres = w_read(sockfd,buf[sockfd]);
    switch (rres)
    {
      case -1:
```

```
logmessage("DEBUG: fd: %d select(): fd %d is ready for read\n", sockfd,
sockfd);

         /* read as much data as we can */
    rres = w_read(sockfd,buf[sockfd],MAXLINE);
    switch (rres)
    {
      case -1:
```

Solutions?

1. Fortify low-level languages (C/C++)
   - Memory safety (e.g., cons. GC, `talloc`)
   - Idioms (e.g., restrictive use of types)
   - Tools (e.g., Valgrind’s `memcheck`)
2. Use a Systems PL
   - BLISS, Modula-3, Cyclone
3. Use two languages
   - FFI’s such as JNI & PInvoke
4. Extend a high-level language
   - Jikes RVM extends Java

Extending: Long History

- Modula-3
  - SPIN
- Java
  - JikesRVM, OVM, Moxie, DRLVM (C/C++ VM)
- C#
  - Bartok, Singularity

- *usually in the context of runtime research*
  - *access to the language/runtime*
  - *complex systems programming task*

Our (Small) Battle for the Revolution

Our “Battleground”

- Jikes RVM
  - Java-in-Java Virtual Machine
  - Combined proven and unproven methods
- Higher-level abstractions
  - Type safety
  - Memory safety
- But what about the cost...
  - Abstraction without guilt?

Our Approach

- **Key Principle: Containment**
  - Minimize exposure to low-level coding
- Extensibility
  - Requirements change quickly
  - Languages change slowly
- Encapsulation
  - Contained low-level semantics
- Fine-grained lowering of semantics
  - Minimize impedance
  - Separation of concerns

Our Framework

- Extend semantics
  - Intrinsic methods
- Controlling semantics
  - Scoped semantic changes
- Extend types
  - box/unbox, ref/value, arch. sizes, etc
- *Prefer* to retain syntax (pragmatism)
  - Existing front-end tools useable

A Concrete Example

```
void prefetchObjects(OOP *buffer, int size) {
  for(int i=0; i < size; i++) {
    OOP current = buffer[i];

    asm volatile("prefetchnta (%0)" ::
                 "r" (current));
  }
}
```

A Concrete Example

```
void prefetchObjects(OOP *buffer, int size) {
  for(int i=0; i < size; i++) {
    OOP current = buffer[i];

    asm volatile("prefetchnta (%0)" ::
                 "r" (current));
  }
}
```

A Concrete Example

```
void prefetchObjects(OOP *buffer, int size) {
  for(int i=0; i < size; i++) {
    OOP current = buffer[i];

    asm volatile("prefetchnta (%0)" ::
                "r" (current));
  }
}
```

Java Version

```
void prefetchObjects(
 ?!? buffer) {

  for(int i=0;i<buffer.length;i++)
  {

    ?!? current = buffer[i];

    ?!?


  }
}
```

```
void prefetchObjects(
  OOP *buffer,
  int size) {

  for(int i=0;i < size;i++){

    OOP o = buffer[i];


    asm volatile(
      "prefetchnta (%0)" ::
      "r" (o));
  }
}
```

“Magic” in JikesRVM

- Raw access to memory?
- Use **int** and “magic” (peek & poke)

```
int ref;
int value = VM_Magic.loadIntAtOffset(ref, offset);
```

“Magic” in JikesRVM

- Raw access to memory?
- Use **int** and “magic” (peek & poke)
- Use **ADDRESS** macro (as **int** or **long**)

```
ADDRESS ref;
int value = VM_Magic.loadIntAtOffset(ref, offset);
```

“Magic” in JikesRVM

- Raw access to memory?
- Use **int** and “magic” (peek & poke)
- Use **ADDRESS** macro (as **int** or **long**)
- Use magical **Address** type (~= **void***)
  - Typed; magic on “instance”

```
Address ref;
int value = ref.loadInt(offset);
```

“Magic” in JikesRVM

- Raw access to memory?
- Use **`int`** and “magic” (peek & poke)
- Use **`ADDRESS`** macro (as **`int`** or **`long`**)
- Use magical **`Address`** type (~= **`void*`**)
- Use **`ObjectReference`** magic type
  - More strongly typed
  - Abstracts over mechanism (handle/pointer)

Java Version

```
void prefetchObjects(
  ObjectReference[] buffer) {

  for(int i=0;i<buffer.length;i++) {

    ObjectReference current
      = buffer[i];

    ?!?



  }
}
```

```
void prefetchObjects(
  OOP *buffer,
  int size) {

  for(int i=0;i < size;i++){

    OOP o = buffer[i];


    asm volatile(
      "prefetchnta (%0)" ::
      "r" (o));
  }
}
```

Intrinsics

- Contract between user and compiler
  - Implement semantics beyond language
  - Requires co-operation of compiler writer
- Canonical intrinsics

```
class ObjectReference {
  ...
  @Intrinsic{Prefetch}
  void prefetch() {} // empty
  ...
}
```

Intrinsics
Prefetch
LoadInt
…

- Express intrinsic abstractly
  - e.g., Intermediate language

Java Version

```java
void prefetchObjects(
  ObjectReference[] buffer) {

  for(int i=0;i<buffer.length;i++) {

    ObjectReference current
      = buffer[i];

    current.prefetch();

  }
}
```

```c
void prefetchObjects(
  OOP *buffer,
  int size) {

  for(int i=0;i < size;i++){

    OOP o = buffer[i];

    asm volatile(
      "prefetchnta (%0)" ::
      "r" (o));
  }
}
```

Other performance overheads?

Java Version

```
@NoBoundsCheck
void prefetchObjects(
  ObjectReference[] buffer) {

  for(int i=0;i<buffer.length;i++) {

    ObjectReference current
      = buffer[i];

    current.prefetch();


  }
}
```

```
void prefetchObjects(
  OOP *buffer,
  int size) {

  for(int i=0;i < size;i++){

    OOP o = buffer[i];


    asm volatile(
      "prefetchnta (%0)" ::
      "r" (o));
  }
}
```

Semantic Regimes

- Scoped semantic change
  - Additive
    - `UncheckedCast` (allow certain intrinsics)
  - Subtractive
    - `NoNew` (allocation via `new()` not permitted)
  - Other
    - `SpillAllRegisters`
    - `NoGCYield`
    - `NoBoundsChecks`

`ObjectReference` Overhead?

- Classes are heap allocated and passed-by-reference

```
@Unboxed
class ObjectReference {
  ...
  @Intrinsic{Prefetch}
  void prefetch() { ... }
  ...
}
```

- No dynamic information (vtable, etc)
  - essentially a C **`struct`**

Type System Extension

- `@Unboxed` types
  - Remove per-instance typing (`struct`)
- Explicit value/reference types
  - Typed pointers (drivers, external data)
- Control field layout
  - Externally defined interfaces?
  - Alternative to marshalling, etc.?
- What about new primitives?
  - **`@RawStorage`**
  - Native width backing data
  - Only accessed with intrinsics

Both Versions

```
@NoBoundsCheck
void prefetchObjects(
  ObjectReference[] buffer) {


  for(int i=0;i<buffer.length;i++) {

    ObjectReference current
      = buffer[i];

    current.prefetch();



  }
}
```

```
void prefetchObjects(
  OOP *buffer,
  int size) {

  for(int i=0;i < size;i++){

    OOP o = buffer[i];


    asm volatile(
      "prefetchnta (%0)" ::
      "r" (o));
  }
}
```

`ObjectReference` Abstraction

- Insulates from implementation
- Handles?

```
class ObjectReference {
  int handle;
  void prefetch() {
    getPayload().prefetch();
  }
  Address getPayload() { ... }
}
```

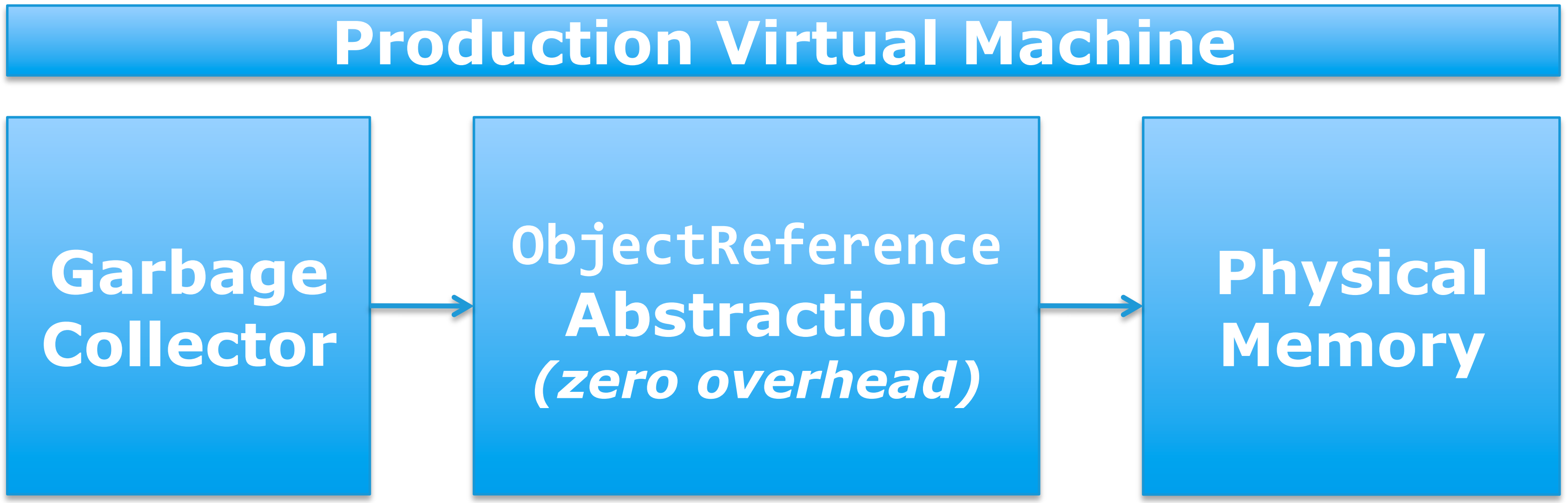
`ObjectReference` Abstraction

- Insulates from implementation
- Handles?

```
class ObjectReference {
  int handle;
  void prefetch() {
    getPayload().prefetch();
  }
  Address getPayload() { ... }
}
```

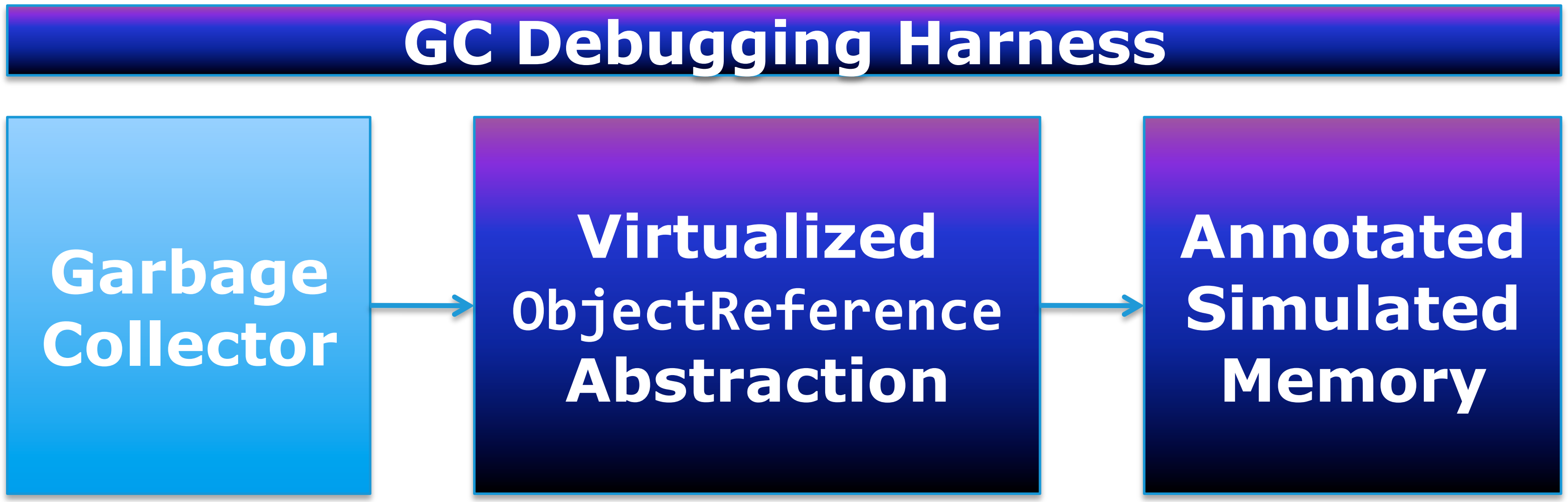
Power of Abstraction

- Allows alternate implementations
  - Instrumentation
  - Virtualization


Production Virtual Machine
Garbage Collector
ObjectReference Abstraction (zero overhead)
Physical Memory

Power of Abstraction

- Allows alternate implementations
  - Instrumentation
  - Virtualization


GC Debugging Harness
Garbage Collector
Virtualized ObjectReference Abstraction
Annotated Simulated Memory

Roadmap

- 10 years of experience
  - mostly in runtime development
  - broader applications?
- Simple unboxed types
  - `Address`, `ObjectReference`, `Word`, etc.
- Semantic regimes
  - `NoBoundsChecks`, etc
- Excellent performance
- Still working on:
  - value/ref types, general unboxing
  - richer semantic regimes
  - more powerful intrinsic mechanism

Summary

- Higher-languages for low-level coding
- We propose a framework
  - Extensible semantics (don’t bake in)
  - Containment of semantic change
  - Extend types
- Much implemented
- Interested in feedback, thoughts
  - **`sudo`** instead of **`root`**?

Questions?

Abstraction without guilt.

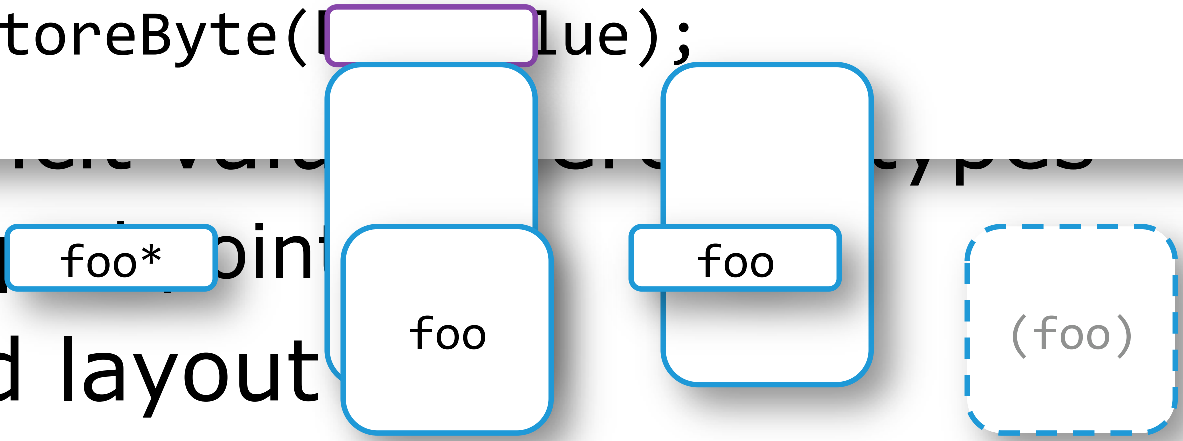
[Ken Kennedy]

Type System Extension

- Raw storage
  - User-defined backing data

```
@RawStorage(lengthInWords=true, length=1)
class Address {
   …
byte loadByte();
void storeByte(b     lue);
}
```

- Explicit valu er   types
  - Ty oint
- Field layout



Considerations

- Low-level coding is the *exception*
  - *not the rule*
- Huge range of low-level details
  - not an all-or-nothing requirement
- Lift the level of abstraction
- Maintain flexibility
  - Languages change slowly
  - Requirements change quickly
- Pragmatic

Defining Terms

- High-level language
  - type safe
  - memory safe
  - strong abstractions over hardware
- Low-level programming
  - requires transparent, efficient access to underlying hardware and/or OS, unimpeded by abstractions.