Tales from Binary Formats

Francesco Zappa Nardelli *Inria*

Based on work done with
Stephen Kell
Computer Lab, Cambridge U.

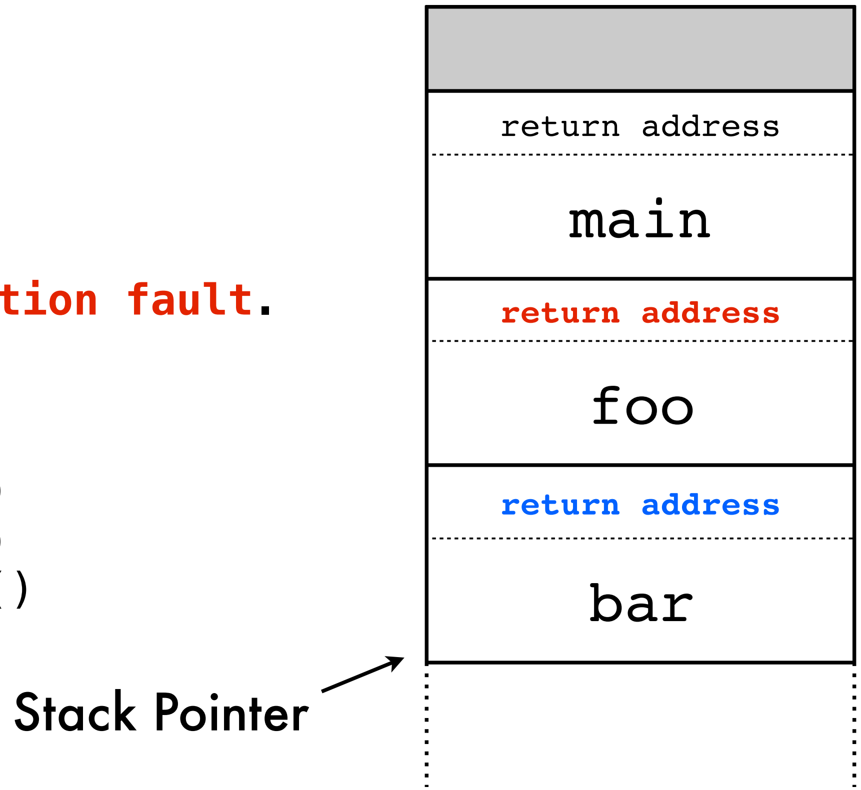
and work done by
Mulligan, Kell, Sewell
Computer Lab, Cambridge U.

```
$ gdb ./a.out
(gdb) run
Starting program: ./a.out

Program received signal Segmentation fault.
0x000000000040049f in bar ()

(gdb)
```

```
$ gdb ./a.out
(gdb) run
Starting program: ./a.out

Program received signal Segmentation fault.
0x000000000040049f in bar ()

(gdb) bt
 #0  0x000000000040049f in bar ()
 #1  0x00000000004004b0 in foo ()
 #2  0x00000000004004bc in main ()
```

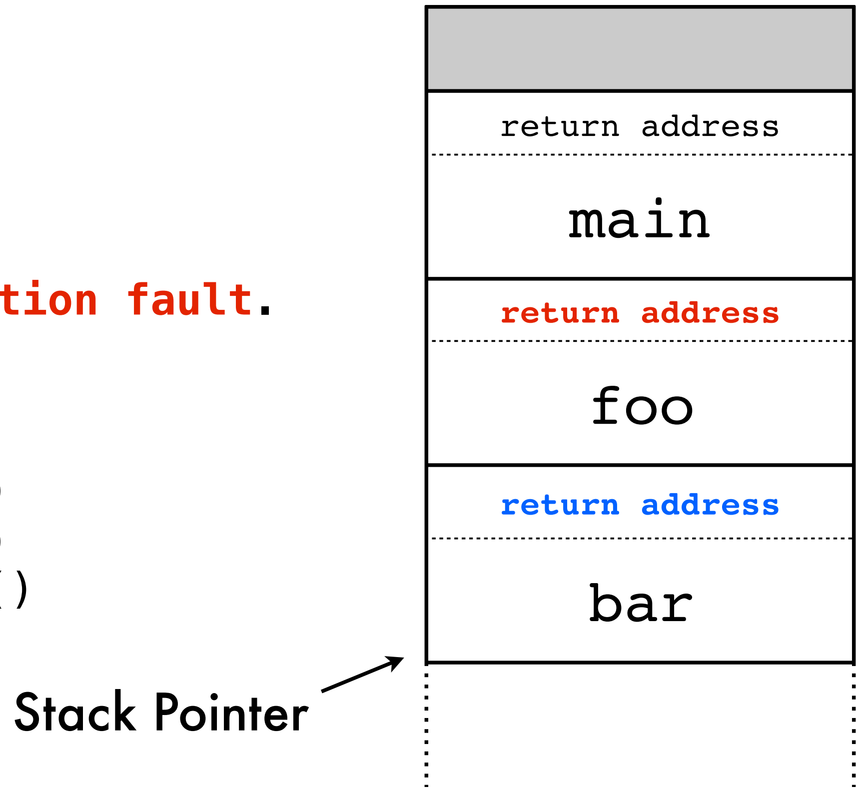
```
$ gdb ./a.out
(gdb) run
Starting program: ./a.out

Program received signal Segmentation fault.
0x000000000040049f in bar ()

(gdb) bt
 #0  0x000000000040049f in bar ()
 #1  0x00000000004004b0 in foo ()
 #2  0x00000000004004bc in main ()
```

Backtrace



Call-stack in memory

```
$ gdb ./a.out
(gdb) run
Starting program: ./a.out

Program received signal Segmentation fault.
0x000000000040049f in bar ()

(gdb) bt
 #0  0x000000000040049f in bar ()
 #1  0x00000000004004b0 in foo ()
 #2  0x00000000004004bc in main ()
```



Backtrace

Call-stack in memory

Given a call-stack
is there a ***reliable*** *and* ***efficient*** way to produce a backtrace?

DWARF unwinding tables

```
$ readelf -Wf a.out


   LOC             CFA      rbx   rbp   r12   r13   r14   r15   ra
00000000004004d0 rsp+8    u     u     u     u     u     u     c-8
00000000004004d2 rsp+16   u     u     u     u     u     c-16  c-8
00000000004004d7 rsp+24   u     u     u     u     c-24  c-16  c-8
00000000004004dc rsp+32   u     u     u     c-32  c-24  c-16  c-8
00000000004004e1 rsp+40   u     u     c-40  c-32  c-24  c-16  c-8
00000000004004e9 rsp+48   u     c-48  c-40  c-32  c-24  c-16  c-8
00000000004004f1 rsp+56   c-56  c-48  c-40  c-32  c-24  c-16  c-8
00000000004004fe rsp+64   c-56  c-48  c-40  c-32  c-24  c-16  c-8
000000000040052a rsp+56   c-56  c-48  c-40  c-32  c-24  c-16  c-8
000000000040052b rsp+48   c-56  c-48  c-40  c-32  c-24  c-16  c-8
000000000040052c rsp+40   c-56  c-48  c-40  c-32  c-24  c-16  c-8
000000000040052e rsp+32   c-56  c-48  c-40  c-32  c-24  c-16  c-8
0000000000400530 rsp+24   c-56  c-48  c-40  c-32  c-24  c-16  c-8
0000000000400532 rsp+16   c-56  c-48  c-40  c-32  c-24  c-16  c-8
0000000000400534 rsp+8    c-56  c-48  c-40  c-32  c-24  c-16  c-8
[...]
```

DWARF unwinding tables

For each instruction...
(identified by the program counter)

LOC	CFA	rbx	rbp	r12	r13	r14	r15	ra
00000000004004d0	rsp+8	u	u	u	u	u	u	c-8
00000000004004d2	rsp+16	u	u	u	u	u	c-16	c-8
00000000004004d7	rsp+24	u	u	u	u	c-24	c-16	c-8
00000000004004dc	rsp+32	u	u	u	c-32	c-24	c-16	c-8
00000000004004e1	rsp+40	u	u	c-40	c-32	c-24	c-16	c-8
00000000004004e[illegible]	rsp+48	u	c-48	c-40	c-32	c-24	c-16	c-8
00000000004004f1	rsp+56	c-56	c-48	c-40	c-32	c-24	c-16	c-8
00000000004004fe	rsp+64	c-56	c-48	c-40	c-32	c-24	c-16	c-8
000000000040052a	rsp+56	c-56	c-48	c-40	c-32	c-24	c-16	c-8
000000000040052b	rsp+48	c-56	c-48	c-40	c-32	c-24	c-16	c-8
000000000040052c	rsp+40	c-56	c-48	c-40	c-32	c-24	c-16	c-8
000000000040052e	rsp+32	c-56	c-48	c-40	c-32	c-24	c-16	c-8
0000000000400530	rsp+24	c-56	c-48	c-40	c-32	c-24	c-16	c-8
0000000000400532	rsp+16	c-56	c-48	c-40	c-32	c-24	c-16	c-8
0000000000400534	rsp+8	c-56	c-48	c-40	c-32	c-24	c-16	c-8

[...]

DWARF unwinding tables

For each instruction…
(identified by the program counter)

…specify how to compute the location on the stack of the return address

```
   LOC             CFA      rbx   rbp   r12   r13   r14   r15   ra
00000000004004d0 rsp+8    u     u     u     u     u     u     c-8
00000000004004d2 rsp+16   u     u     u     u     u     c-16  c-8
00000000004004d7 rsp+24   u     u     u     u     c-24  c-16  c-8
00000000004004dc rsp+32   u     u     u     c-32  c-24  c-16  c-8
00000000004004e1 rsp+40   u     u     c-40  c-32  c-24  c-16  c-8
00000000004004e2 rsp+48   u     c-48  c-40  c-32  c-24  c-16  c-8
00000000004004f1 rsp+56   c-56  c-48  c-40  c-32  c-24  c-16  c-8
00000000004004fe rsp+64   c-56  c-48  c-40  c-32  c-24  c-16  c-8
000000000040052a rsp+56   c-56  c-48  c-40  c-32  c-24  c-16  c-8
000000000040052b rsp+48   c-56  c-48  c-40  c-32  c-24  c-16  c-8
000000000040052c rsp+40   c-56  c-48  c-40  c-32  c-24  c-16  c-8
000000000040052e rsp+32   c-56  c-48  c-40  c-32  c-24  c-16  c-8
0000000000400530 rsp+24   c-56  c-48  c-40  c-32  c-24  c-16  c-8
0000000000400532 rsp+16   c-56  c-48  c-40  c-32  c-24  c-16  c-8
0000000000400534 rsp+8    c-56  c-48  c-40  c-32  c-24  c-16  c-8
[...]
```

DWARF unwinding tables

For each instruction…
(identified by the program counter)

…specify how to compute the location on the stack of the return address

LOC	CFA	rbx	rbp	r12	r13	r14	r15	ra
0000000000400534	rsp+8	c−56	c−48	c−40	c−32	c−24	c−16	c−8
[...]								

Relied upon to implement ***stack unwinding***

By **debuggers**

DWARF unwinding tables

For each instruction…
(identified by the program counter)

…specify how to compute the location on the stack of the return address

```
  LOC              CFA       rbx   rbp   r12   r13   r14   r15   ra
```

Relied upon to implement ***stack unwinding***

By **debuggers** but also by **program analysis tools**

```
0000000000400534 rsp+8     c-56  c-48  c-40  c-32  c-24  c-16  c-8
[...]
```

DWARF unwinding tables

For each instruction...
(identified by the program counter)

...specify how to compute the location on the stack of the return address

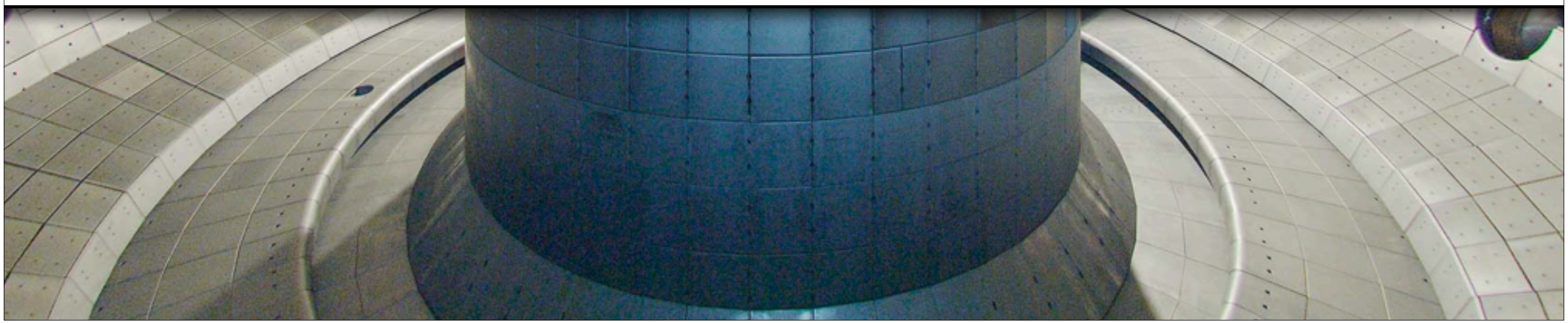
LOC	CFA	rbx	rbp	r12	r13	r14	r15	ra
0000000000400534	rsp+8	c−56	c−48	c−40	c−32	c−24	c−16	c−8
[...]								

Relied upon to implement ***stack unwinding***

By **debuggers** but also by **program analysis tools**

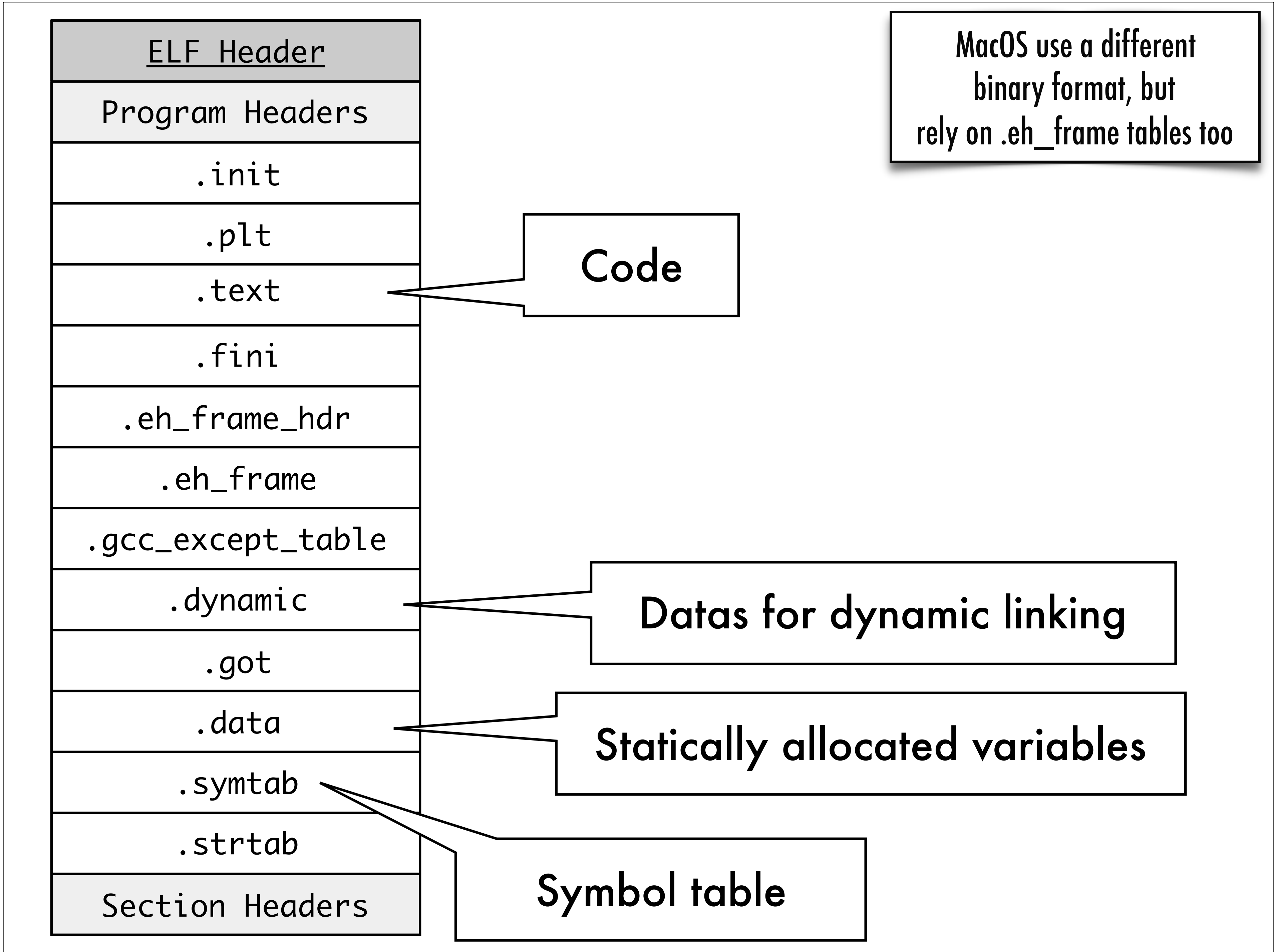
And by the **C++ runtime** to implement **exceptions!!!**

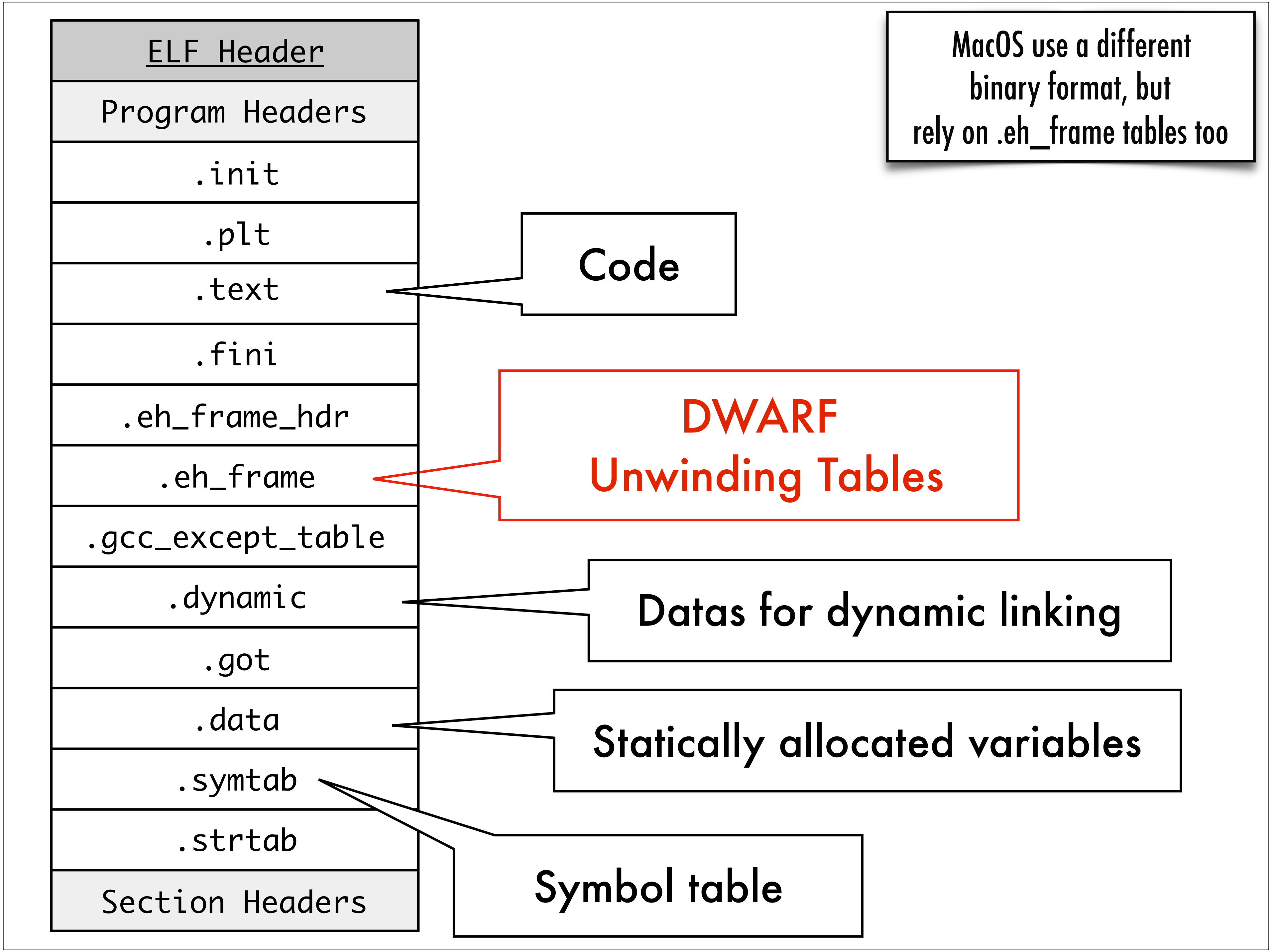
Debug tables are not only for debugging



ELF Header
Program Headers
.init
.plt
.text
.fini
.eh_frame_hdr
.eh_frame
.gcc_except_table
.dynamic
.got
.data
.symtab
.strtab
Section Headers

MacOS use a different binary format, but rely on .eh_frame tables too


ELF Header
Program Headers
.init
.plt
.text
.fini
.eh_frame_hdr
.eh_frame
.gcc_except_table
.dynamic
.got
.data
.symtab
.strtab
Section Headers
MacOS use a different binary format, but rely on .eh_frame tables too
Code
Datas for dynamic linking
Statically allocated variables
Symbol table

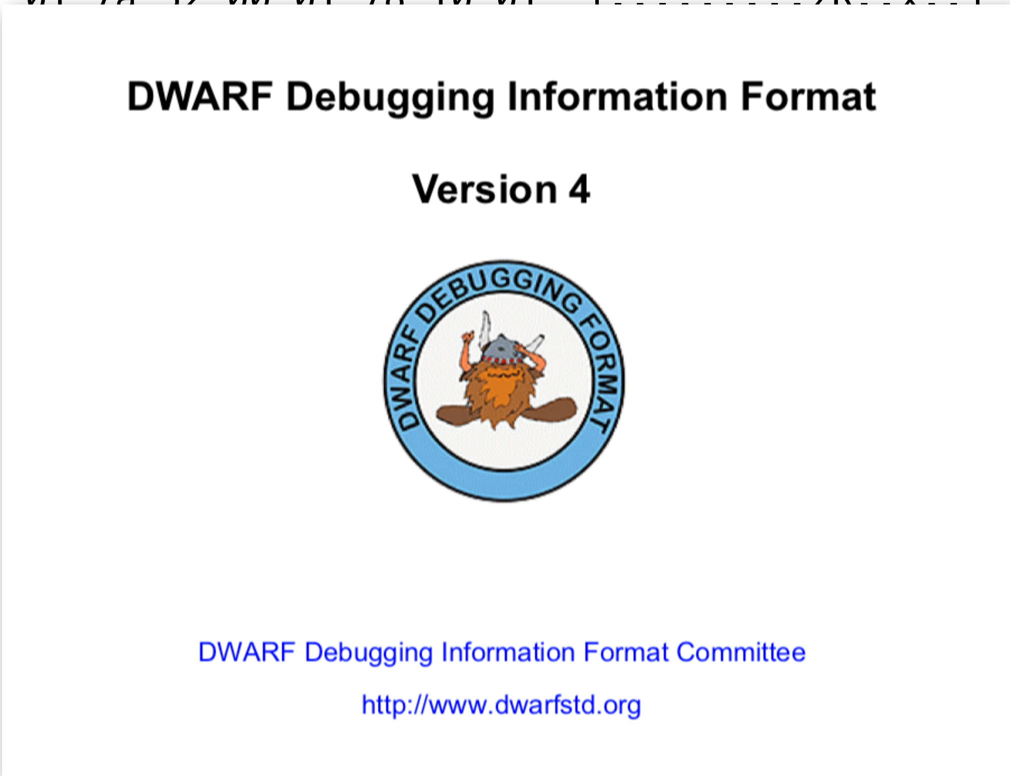

ELF Header
Program Headers
.init
.plt
.text
.fini
.eh_frame_hdr
.eh_frame
.gcc_except_table
.dynamic
.got
.data
.symtab
.strtab
Section Headers
MacOS use a different binary format, but rely on .eh_frame tables too
Code
DWARF Unwinding Tables
Datas for dynamic linking
Statically allocated variables
Symbol table

Unwinding Tables on disk

```
0000f670  14 00 00 00 00 00 00 00  01 7a 52 00 01 78 10 01  |.........zR..x..|
0000f680  1b 0c 07 08 90 01 07 10  14 00 00 00 1c 00 00 00  |................|
0000f690  90 0d ff ff 2a 00 00 00  00 00 00 00 00 00 00 00  |....*...........|
0000f6a0  14 00 00 00 00 00 00 00  01 7a 52 00 01 78 10 01  |.........zR..x..|
0000f6b0  1b 0c 07 08 90 01 00 00  24 00 00 00 1c 00 00 00  |........$.......|
0000f6c0  10 0d ff ff 40 00 00 00  00 0e 10 46 0e 18 4a 0f  |....@......F..J.|
0000f6d0  0b 77 08 80 00 3f 1a 3b  2a 33 24 22 00 00 00 00  |.w...?.;*3$"....|
0000f6e0  1c 00 00 00 44 00 00 00  2e 0e ff ff 07 00 00 00  |....D...........|
0000f6f0  00 41 0e 10 86 02 43 0d  06 42 0c 07 08 00 00 00  |.A....C..B......|
0000f700  1c 00 00 00 64 00 00 00  15 0e ff ff 25 00 00 00  |....d.......%...|
0000f710  00 41 0e 10 86 02 43 0d  06 60 0c 07 08 00 00 00  |.A....C..`......|
0000f720  1c 00 00 00 84 00 00 00  1a 0e ff ff 11 00 00 00  |................|
0000f730  00 41 0e 10 86 02 43 0d  06 4c 0c 07 08 00 00 00  |.A....C..L......|
0000f740  1c 00 00 00 a4 00 00 00  0b 0e ff ff 1a 00 00 00  |................|
0000f750  00 41 0e 10 86 02 43 0d  06 55 0c 07 08 00 00 00  |.A....C..U......|
0000f760  1c 00 00 00 c4 00 00 00  05 0e ff ff 1c 00 00 00  |................|
```

Unwinding Tables on disk

```
0000f670  14 00 00 00 00 00 00 00  01 7a 52 00 01 78 10 01  |.........zR..x..|
0000f680  1b 0c 07 08 90 01 07 10  14 00 00 00 1c 00 00 00  |................|
0000f690  90 0d ff ff 2a 00 00 00  00 00 00 00 00 00 00 00  |....*...........|
0000f6a0  14 00 00 00 00 00 00 00  01 7a 52 00 01 78 10 01  |.........zR..x..|
0000f6b0  1b 0c 07 08 90 01 00 00
0000f6c0  10 0d ff ff 40 00 00 00
0000f6d0  0b 77 08 80 00 3f 1a 3b
0000f6e0  1c 00 00 00 44 00 00 00
0000f6f0  00 41 0e 10 86 02 43 0d
0000f700  1c 00 00 00 64 00 00 00
0000f710  00 41 0e 10 86 02 43 0d
0000f720  1c 00 00 00 84 00 00 00
0000f730  00 41 0e 10 86 02 43 0d
0000f740  1c 00 00 00 a4 00 00 00
0000f750  00 41 0e 10 86 02 43 0d
0000f760  1c 00 00 00 c4 00 00 00
```


DWARF Debugging Information Format

Version 4

DWARF Debugging Information Format Committee

http://www.dwarfstd.org

DWARF Debug Unwinding Table

```
DW_CFA_def_cfa_offset: 16
DW_CFA_advance_loc: 6 to 0000000000400376
DW_CFA_def_cfa_offset: 24
DW_CFA_advance_loc: 10 to 0000000000400380
DW_CFA_def_cfa_expression (DW_OP_breg7 (rsp): 8;
    DW_OP_breg16 (rip): 0; DW_OP_lit15;
    DW_OP_and; DW_OP_lit11; DW_OP_ge; DW_OP_lit3;
    DW_OP_shl; DW_OP_plus)
[...]
```

DWARF Debug Unwi

Bytecode instructions

```
DW_CFA_def_cfa_offset: 16
DW_CFA_advance_loc: 6 to 0000000000400376
DW_CFA_def_cfa_offset: 24
DW_CFA_advance_loc: 10 to 0000000000400380
DW_CFA_def_cfa_expression (DW_OP_breg7 (rsp): 8;
    DW_OP_breg16 (rip): 0; DW_OP_lit15;
    DW_OP_and; DW_OP_lit11; DW_OP_ge; DW_OP_lit3;
    DW_OP_shl; DW_OP_plus)
[...]
```

Arbitrary expressions accessing registers and memory locations

DWARF Debug Unwi

Bytecode instructions

```
DW_CFA_def_cfa_offset: 16
DW_CFA_advance_loc: 6 to 0000000000400376
DW_CFA_def_cfa_offset: 24
DW_CFA_advance_loc: 10 to 0000000000400380
DW_CFA_def_cfa_expression (DW_OP_breg7 (rsp): 8;
    DW_OP_breg16 (rip): 0; DW_OP_lit15;
    DW_OP_and; DW_OP_lit11; DW_OP_ge; DW_OP_lit3;
    DW_OP_shl; DW_OP_plus)
```

Interpreted on demand to inspect the call-stack

A Turing-complete stack-based machine

can dereference memory and access values in machine registers

— in a place where few expect it

Badly specified

Two subtly incompatible formats: *debug_frame* and *eh_frame*

Badly specified

Two subtly incompatible formats: *debug_frame* and *eh_frame*

Airs – Ian Lance Taylor

.eh_frame

January 10, 2011 at 11:12 pm · Filed under Programming

When gcc generates code that handles exceptions, it produces tables that describe how to unwind the stack. These tables are found in the .eh_frame section. The format of the .eh_frame section is very similar to the format of a DWARF .debug_frame section. Unfortunately, it is not precisely identical. I don't know of any documentation which describes this format. The following should be read in conjunction with the relevant section of the DWARF standard, available from http://dwarfstd.org.


Badly specified
Two subtly incompatible formats: debug_frame and eh_frame

Badly specified

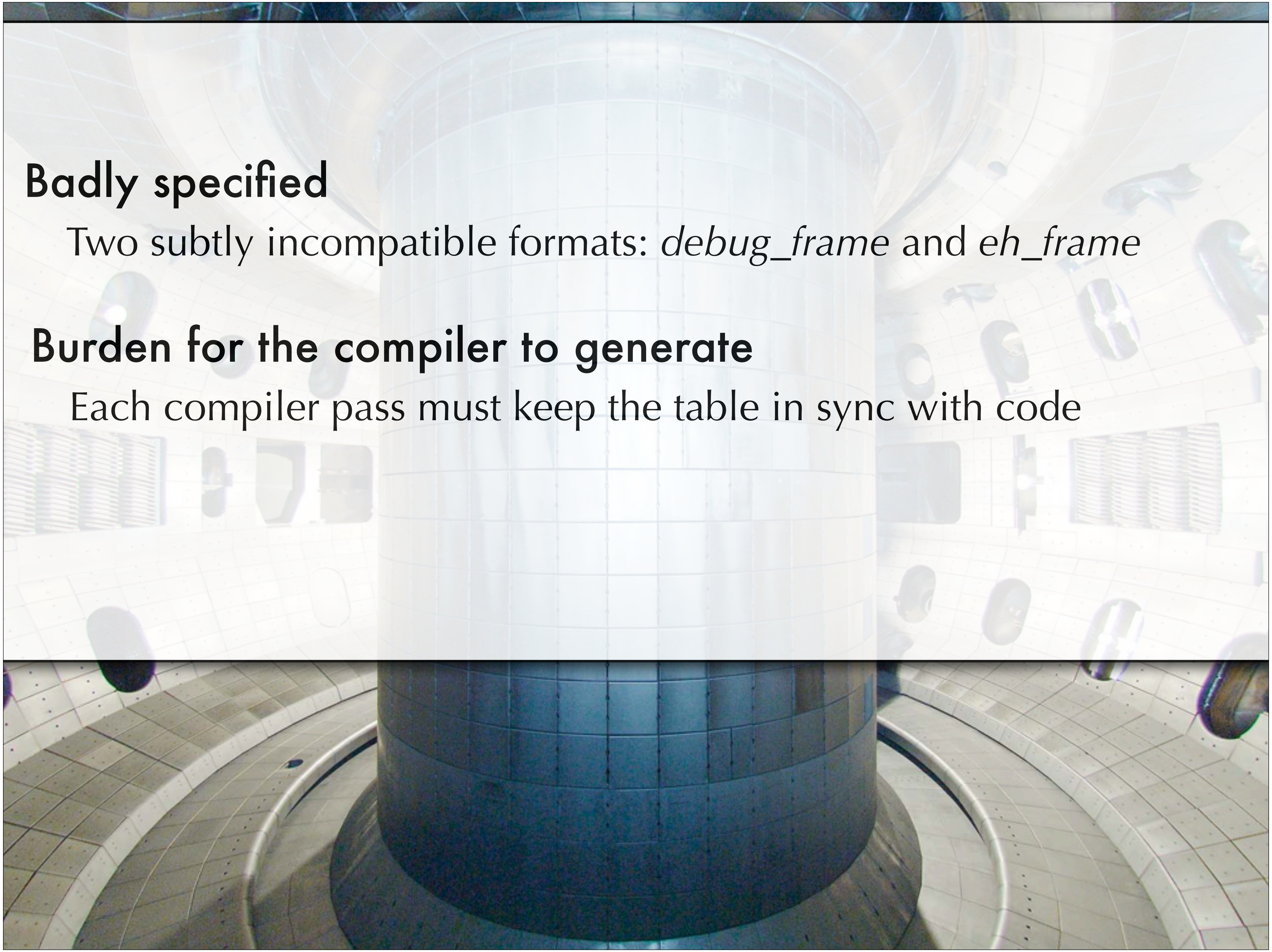
Two subtly incompatible formats: *debug_frame* and *eh_frame*

Burden for the compiler to generate

Each compiler pass must keep the table in sync with code

```
foo:
    .cfi_startproc
   pushq   %rbp
   .cfi_def_cfa_offset 16
   .cfi_offset 6, -16
   movq    %rsp, %rbp
   .cfi_def_cfa_register 6
   movl    %edi, -4(%rbp)
   movl    %esi, -8(%rbp)
   movl    -4(%rbp), %edx
   movl    -8(%rbp), %eax
   addl    %edx, %eax
   popq    %rbp
   .cfi_def_cfa 7, 8
   ret
   .cfi_endproc
```

```
$ gcc -S foo.c
```


Badly specified
Two subtly incompatible formats: debug_frame and eh_frame
Burden for the compiler to generate
Each compiler pass must keep the table in sync with code

Badly specified

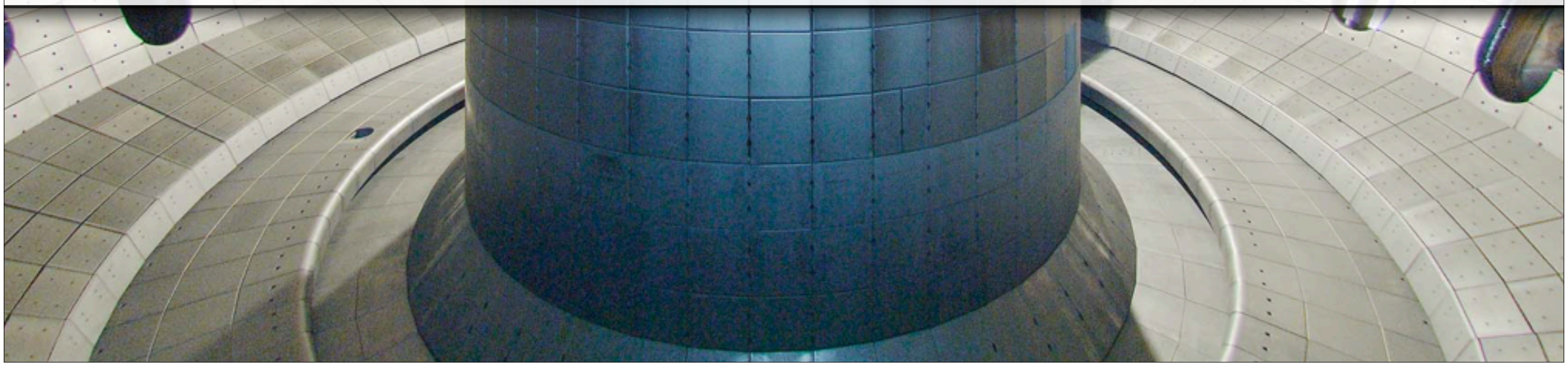
Two subtly incompatible formats: *debug_frame* and *eh_frame*

Burden for the compiler to generate

Each compiler pass must keep the table in sync with code

Potential vector for arbitrary code execution

Proof-of-concept attack built six years ago



Exploiting the hard-working DWARF: Trojan and Exploit Techniques With No Native Executable Code

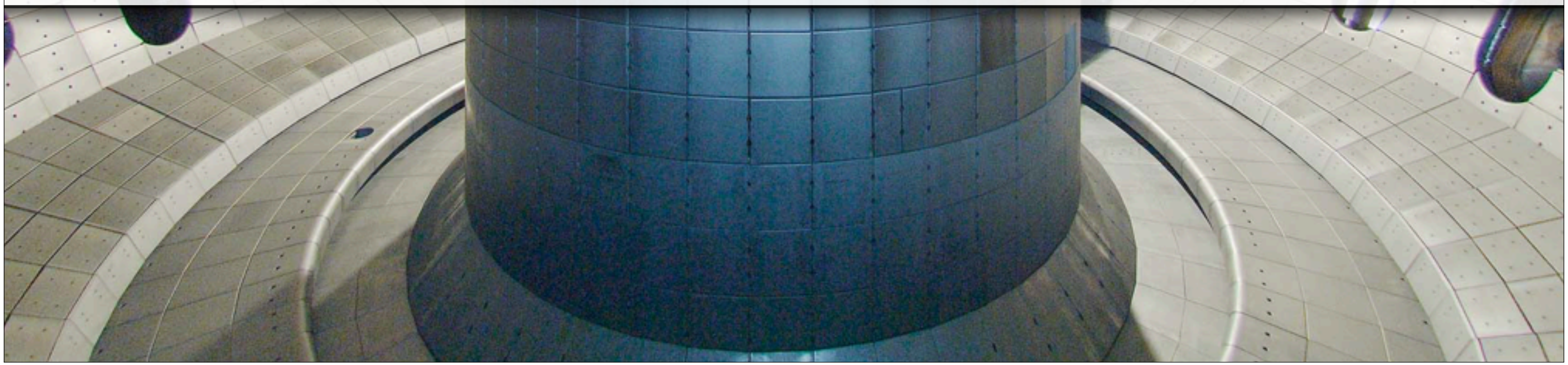
James Oakley, Sergey Bratus
Computer Science Dept.
Dartmouth College
Hanover, New Hampshire
james.oakley@alum.dartmouth.org, sergey@cs.dartmouth.edu

USENIX WOOT'11

Each compiler pass must keep the table in sync with code

Potential vector for arbitrary code execution

Proof-of-concept attack built six years ago



Badly specified

Two subtly incompatible formats: *debug_frame* and *eh_frame*

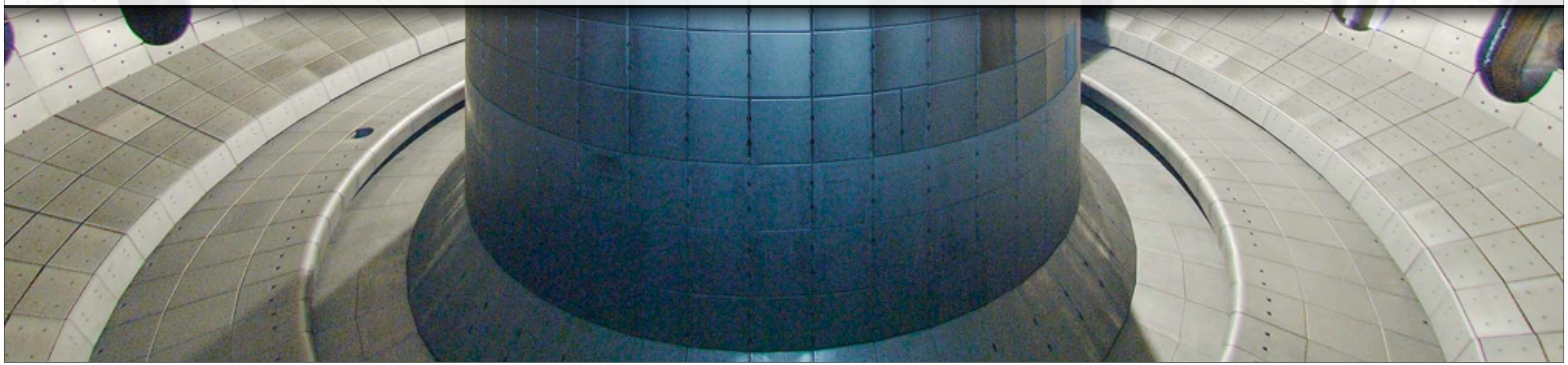
Burden for the compiler to generate

Each compiler pass must keep the table in sync with code

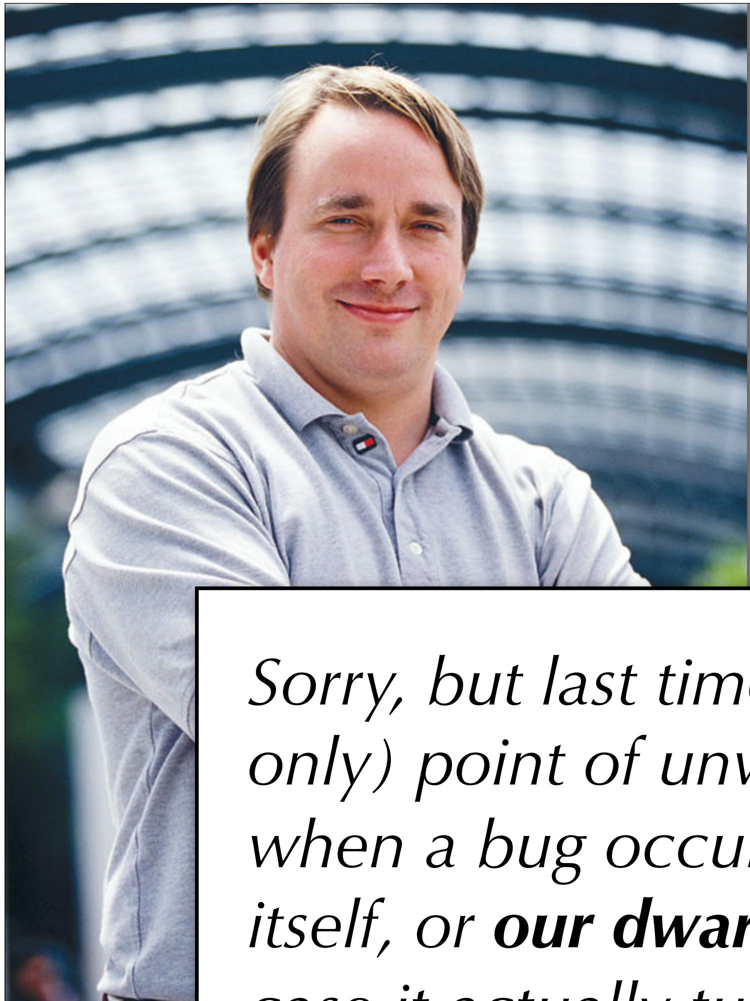
Potential vector for arbitrary code execution

Proof-of-concept attack built five years ago

Complex, Buggy, Untested



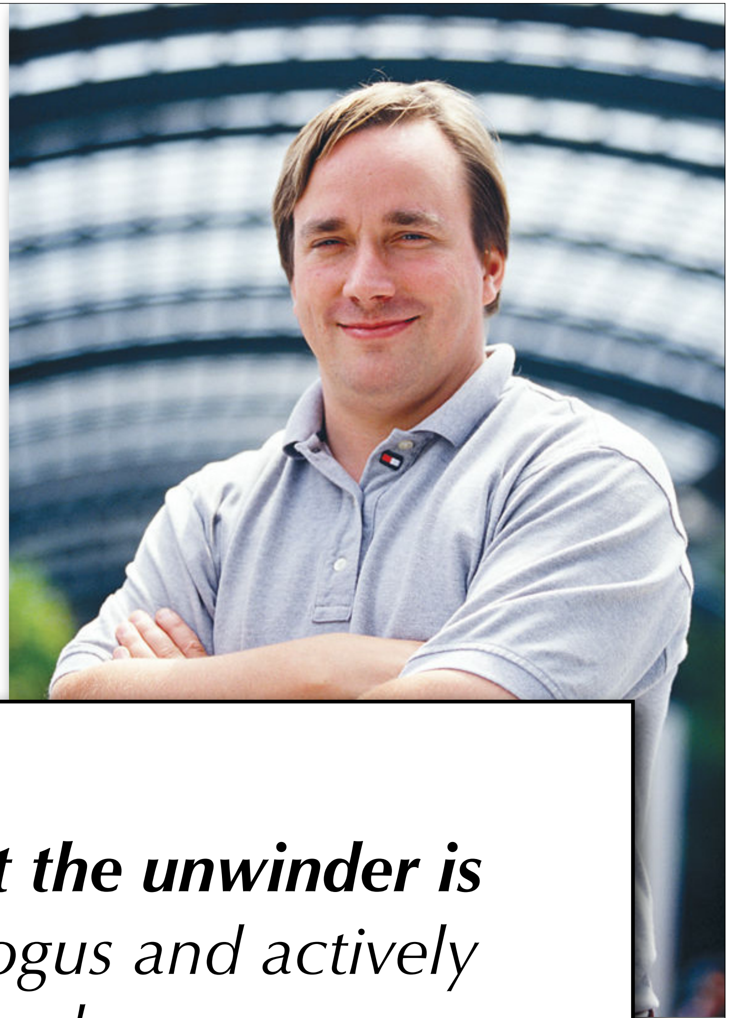
Why doesn't the Linux Kernel rely on DWARF tables?



Why doesn't the Linux Kernel rely on DWARF tables?

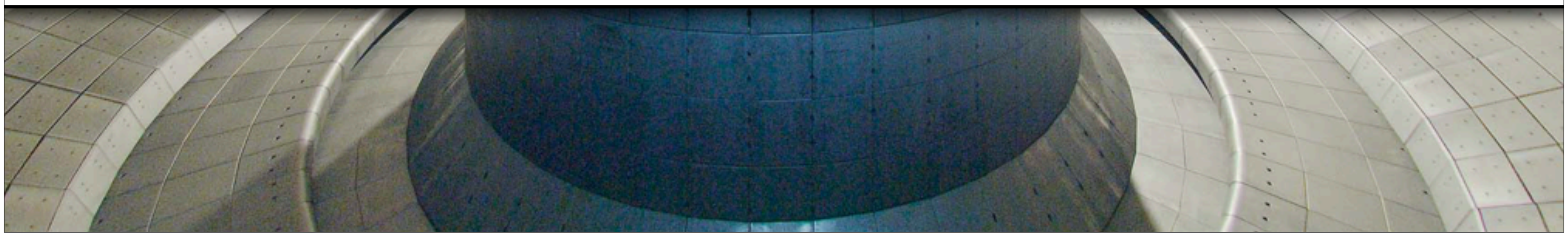
Sorry, but last time was too painful. The whole (and only) point of unwinders is to make debugging easy when a bug occurs. But ***the dwarf unwinder had bugs*** *itself, or* ***our dwarf information had bugs****, and in either case it actually turned several trivial bugs into a* ***total undebuggable hell****.*

Linus Torvalds, Kernel mailing list, 2012



*If you can **mathematically prove that the unwinder is correct** — even in the presence of bogus and actively incorrect unwinding information — and never ever follows a bad pointer, **I'll reconsider**.*

Validation of unwinding tables



```
<foo>:
  push   %r15
  push   %r14
  mov    $0x3,%eax
  push   %r13
  push   %r12
  push   %rbp
  push   %rbx
  sub    $0x68,%rsp
  cmp    $0x1,%edi
  movl   $0x0,0x14(%rsp)
  je     49e4a0
  add    $0x68,%rsp
  pop    %rbx
  pop    %rbp
```

```
<foo>:
  push   %r15
  push   %r14
  mov    $0x3,%eax
```

Assume:

- `eh_frame` table has been generated by the compiler
- we can interpret the `eh_frame` bytecode to generate the unwinding tables

```
  movl   $0x0,0x14(%rsp)
  je     49e4a0
  add    $0x68,%rsp
  pop    %rbx
  pop    %rbp
```

		CFA	ra
<foo>:			
push	%r15	rsp+8	c-8
push	%r14	rsp+16	c-8
mov	$0x3,%eax	rsp+24	c-8
movl	$0x0,0x14(%rsp)	rsp+160	c-8
je	49e4a0	rsp+160	c-8
add	$0x68,%rsp	rsp+160	c-8
pop	%rbx	rsp+56	c-8
pop	%rbp	rsp+48	c-8

Assume:

- `eh_frame` table has been generated by the compiler
- we can interpret the `eh_frame` bytecode to generate the unwinding tables

<foo>:		CFA	ra
push	%r15	rsp+8	c-8
push	%r14	rsp+16	c-8
mov	$0x3,%eax	rsp+24	c-8
push	%r13	rsp+24	c-8
push	%r12	rsp+32	c-8
push	%rbp	rsp+40	c-8
push	%rbx	rsp+48	c-8
sub	$0x68,%rsp	rsp+56	c-8
cmp	$0x1,%edi	rsp+160	c-8
movl	$0x0,0x14(%rsp)	rsp+160	c-8
je	49e4a0	rsp+160	c-8
add	$0x68,%rsp	rsp+160	c-8
pop	%rbx	rsp+56	c-8
pop	%rbp	rsp+48	c-8

<foo>:		CFA	ra
push	%r15	rsp+8	c-8
push	%r14	rsp+16	c-8
mov	$0x3,%eax	rsp+24	c-8
push	%r13	rsp+24	c-8
push	%r12	rsp+32	c-8
push	%rbp	rsp+40	c-8
push	%rbx	rsp+48	c-8
sub	$0x68,%rsp	rsp+56	c-8
pop	%rbx	rsp+56	c-8
pop	%rbp	rsp+48	c-8

When `foo` is called, *before executing its first instruction:*

return address is stored at `*rsp`

<foo>:		CFA	ra
push	%r15	rsp+8	c-8
push	%r14	rsp+16	c-8
mov	$0x3,%eax	rsp+24	c-8
push	%r13	rsp+24	c-8
push	%r12	rsp+32	c-8
push	%rbp	rsp+40	c-8
push	%rbx	rsp+48	c-8
sub	$0x68,%rsp	rsp+56	c-8
pop	%rbx	rsp+56	c-8
pop	%rbp	rsp+48	c-8

After `push %r15`, `rsp` has been decreased by 8:

return address is stored at `*(rsp+8)`

<foo>:		CFA	ra
push	%r15	rsp+8	c-8
push	%r14	rsp+16	c-8
mov	$0x3,%eax	rsp+24	c-8
push	%r13	rsp+24	c-8
push	%r12	rsp+32	c-8
push	%rbp	rsp+40	c-8
push	%rbx	rsp+48	c-8
sub	$0x68,%rsp	rsp+56	c-8

After `push %r14`, `rsp` has been decreased by 8:

return address is stored at `*(rsp+16)`

pop	%rbx	rsp+56	c-8
pop	%rbp	rsp+48	c-8

<foo>:		CFA	ra
push	%r15	rsp+8	c-8
push	%r14	rsp+16	c-8
mov	$0x3,%eax	rsp+24	c-8
push	%r13	rsp+24	c-8
push	%r12	rsp+32	c-8
push	%rbp	rsp+40	c-8
push	%rbx	rsp+48	c-8
sub	$0x68,%rsp	rsp+56	c-8

After `mov $0x3,%eax`:

return address is still stored at `*(rsp+16)`

pop	%rbx	rsp+56	c-8
pop	%rbp	rsp+48	c-8

<foo>:		CFA	ra
push	%r15	rsp+8	c-8
push	%r14	rsp+16	c-8
mov	$0x3,%eax	rsp+24	c-8
push	%r13	rsp+24	c-8
push	%r12	rsp+32	c-8
push	%rbp	rsp+40	c-8
pop	%rbx	rsp+56	c-8
pop	%rbp	rsp+48	c-8

The unwinding table is redundant wrt the binary code

It captures *some abstract execution* of the code

```
<foo>:                                  CFA        ra
  push   %r15                           rsp+8      c-8
  push   %r14                           rsp+16     c-8
  mov    $0x3,%eax                      rsp+24     c-8
  push   %r13                           rsp+24     c-8
  push   %r12                           rsp+32     c-8
  push   %rbp                           rsp+40     c-8
  push   %rbx                           rsp+48     c-8
  sub    $0x68,%rsp                     rsp+56     c-8
  cmp    $0x1,%edi                      rsp+160    c-8
  movl   $0x0,0x14(%rsp)                rsp+160    c-8
  je     49e4a0                         rsp+160    c-8
  add    $0x68,%rsp                     rsp+160    c-8
  pop    %rbx                           rsp+56     c-8
  pop    %rbp                           rsp+48     c-8
```

```
<foo>:                                  CFA        ra
  push   %r15                           rsp+8      c-8
  push   %r14                           rsp+16     c-8
  mov    $0x3,%eax                      rsp+24     c-8
  push   %r13                           rsp+24     c-8
  push   %r12                           rsp+32     c-8
  push   %rbp                           rsp+40     c-8
  push   %rbx                           rsp+48     c-8
  sub    $0x68,%rsp                     rsp+56     c-8
  cmp    $0x1,%edi                      rsp+160    c-8
  movl   $0x0,0x14(%rsp)                rsp+160    c-8
  je     49e4a0                         rsp+160    c-8
  add    $0x68,%rsp                     rsp+160    c-8
  pop    %rbx                           rsp+56     c-8
  pop    %rbp                           rsp+48     c-8
```

<foo>:		CFA	ra
push	%r15	rsp+8	c-8

Suppose that we know that **in an execution**:

- the return address is stored at `0xFFFF1158`

We read `%rsp` and it stores `0xFFFF1000`

push	%rbp	rsp+40	c-8
push	%rbx	rsp+48	c-8
sub	$0x68,%rsp	rsp+56	c-8
cmp	$0x1,%edi	rsp+160	c-8
movl	$0x0,0x14(%rsp)	rsp+160	c-8
je	49e4a0	rsp+160	c-8
add	$0x68,%rsp	rsp+160	c-8
pop	%rbx	rsp+56	c-8
pop	%rbp	rsp+48	c-8

```
<foo>:                       CFA       ra
  push   %r15                rsp+8     c-8
```

Suppose that we know that **in an execution**:

- the return address is stored at `0xFFFF1158`

We read `%rsp` and it stores `0xFFFF1000`

```
  push   %rbp                rsp+40    c-8
  push   %rbx                rsp+48    c-8
  sub    $0x68,%rsp          rsp+56    c-8
  cmp    $0x1,%edi           rsp+160   c-8
  movl   $0x0,0x14(%rsp)     rsp+160   c-8
```

Evaluating the **entry** on the **current register values**,

should compute the **concrete address** of the return address.

Dynamic Validation of Unwinding Tables

Abstract state

stack of concrete **addresses where return address are stored**

Abstract instruction semantics

`call`: **push** the **content of** `%rsp` on top of the abstract state
`ret`: **pop** one value from the abstract state

Validation at each instruction

evaluate the return address eh_frame entry for ip
compare with the value in abstract state

Dynamic Validation of Unwinding T

Abstract state

stack of concrete **addresses**

Abstract instruction sem

`call`: **push** th … top of the abstract state

`ret`: **pop** … bstract state

Vali … ction

… urn address eh_frame entry for ip

… ith the value in abstract state

callee-saved registers require some care

The eh_frame_check tool

Goal: validate eh_frame entries along one execution path

gdb:

- step-by-step binary execution, access to processor state

Python:

- parsing ELF and DWARF binary code *(building on pyelftool)*
- evaluating DWARF expressions
- scripting gdb to implement the dynamic analysis

The eh_frame_check tool

Goal: validate eh_frame entries along one execution path

gdb:

- step-by-step binary execution, access to processor state

Python:

- parsing ELF and DWARF binary code *(building on pyelftool)*
- evaluating DWARF expressions
- scripting gdb to implement the dynamic analysis

Can process *a few k asm instructions/sec*: good for now

```
short a,b,g;
long c;
char d;
int e, f;

void main() {
  for (; f; f++)
    for (; e; e++)
      for (; c; c++) {
        g = a % b;
        for (; d <= 1; d++)
          ;
      }
}
```

CSmith + Creduce + eh_frame_check.py

```
<main>:                              CFA       ra
  4004e0:   push  %rbx               rsp+8     c-8
                                     rsp+16    c-8
  [..]
                                     [..]

  40061d:  pop  %rbx                 rsp+16    c-8
  40061e:  retq                      rsp+16    c-8
```

Abstract state [0xFFFF1000]

<main>:	CFA	ra
4004e0: push %rbx	rsp+8	c-8
	rsp+16	c-8
[..]	[..]	
40061d: pop %rbx	rsp+16	c-8
40061e: retq	rsp+16	c-8

Abstract state [0xFFFF1000]

%rsp 0xFFFF1000

<main>:	CFA	ra
4004e0: push %rbx	rsp+8	c-8
	rsp+16	c-8
[..]	[..]	
40061d: pop %rbx	rsp+16	c-8
40061e: retq	rsp+16	c-8

Abstract state [0xFFFF1000]

%rsp 0xFFFF1000

<main>:	CFA	ra
4004e0: push %rbx	rsp+8	c-8
	rsp+16	c-8
[..]	[..]	
40061d: pop %rbx	rsp+16	c-8
40061e: retq	rsp+16	c-8

Abstract state [0xFFFF1000]

%rsp 0xFFFF0FF8

```
<main>:                                  CFA      ra
  4004e0:  push  %rbx                    rsp+8    c-8
                                         rsp+16   c-8
  [..]
                                         [..]

  40061d:  pop  %rbx                     rsp+16   c-8
  40061e:  retq                          rsp+16   c-8
```

Abstract state [0xFFFF1000]

```
%rsp    0xFFFF0FF8
```

```
<main>:                           CFA       ra
  4004e0:  push  %rbx             rsp+8     c-8
                                  rsp+16    c-8
  [..]
                                  [..]

  40061d:  pop   %rbx             rsp+16    c-8
  40061e:  retq                   rsp+16    c-8
```

Abstract state [0xFFFF1000]

%rsp 0xFFFF0FF8

```
<main>:                            CFA       ra
  4004e0:  push  %rbx              rsp+8     c-8
                                   rsp+16    c-8
  [..]
                                   [..]

  40061d:  pop  %rbx               rsp+16    c-8
  40061e:  retq                    rsp+16    c-8
```

Abstract state [0xFFFF1000]

%rsp 0xFFFF0FF8

	CFA	ra
<main>:		
4004e0: push %rbx	rsp+8	c-8
	rsp+16	c-8
[..]	[..]	
40061d: pop %rbx	rsp+16	c-8
40061e: retq	rsp+16	c-8

Abstract state [0xFFFF1000]

%rsp 0xFFFF1000

```
<main>:                              CFA       ra
  4004e0:  push  %rbx                rsp+8     c-8
                                     rsp+16    c-8
  [..]
                                     [..]

  40061d:  pop  %rbx                 rsp+16    c-8
  40061e:  retq                      rsp+16    c-8
```

Abstract state [0xFFFF1000]

```
%rsp    0xFFFF1000
```

```
<main>:                            CFA       ra
  4004e0:  push  %rbx              rsp+8     c-8
                                   rsp+16    c-8
  [..]
                                   [..]

  40061d:  pop  %rbx               rsp+16    c-8
  40061e:  retq                    rsp+16    c-8
```

```
0xFFFF1000+16-8   !=   0xFFFF1000
```

At -O0, -O2, or -Os. *Not at* -O1.

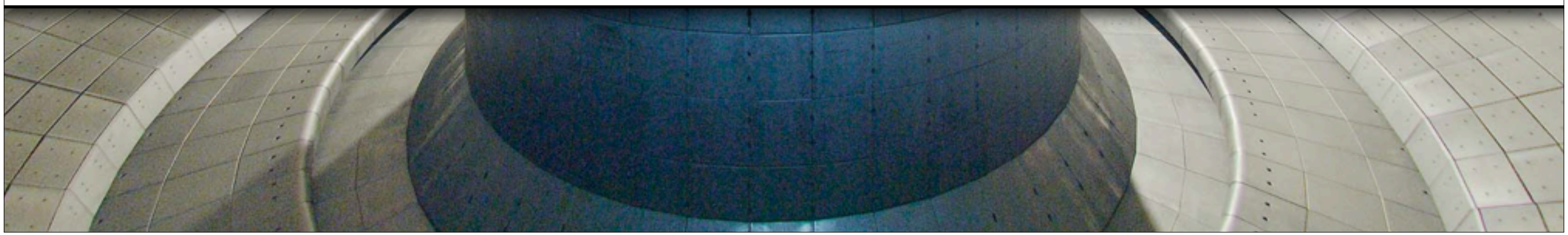
```
<main>:                              CFA       ra
  4004e0:   push   %rbx              rsp+8     c-8
                                     rsp+16    c-8
  [..]
                                     [..]

  40061d:  pop   %rbx                rsp+16    c-8
  40061e:  retq                      rsp+16    c-8
```

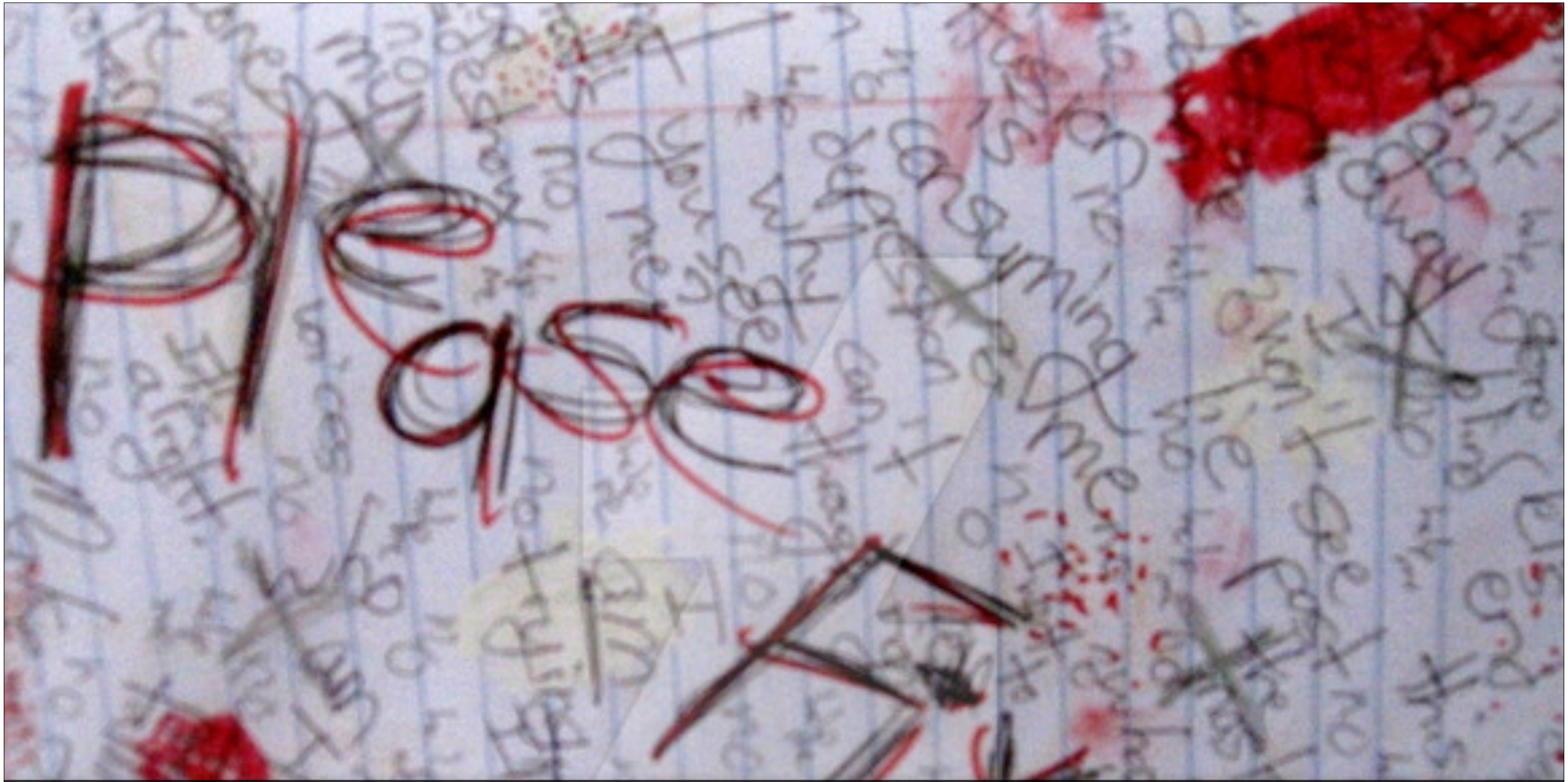
At -O0, -O2, or -Os. *Not at* -O1.

<main>:		CFA	ra
4004e0:	push %rbx	rsp+8	c-8
		rsp+16	c-8
[..]		[..]	
40061d:	pop %rbx	rsp+16	c-8
40061e:	retq	rsp+16	c-8

Just a coincidence: at -O1 `rbx` is not saved.

Validation of unwinding tables is effective





https://llvm.org/bugs/show_bug.cgi?id=13161


Please
https://llvm.org/bugs/
show_bug.cgi?id=13161

The sad truths

Generating `eh_frame` is a *burden* for the compiler

Some compilers do not generate `eh_frame` at all

The sad truths

Generating `eh_frame` is a *burden* for the compiler

Some compilers do not generate `eh_frame` at all

OCaml `eh_frame` code was contributed recently by JaneStreet

The sad truths

Generating `eh_frame` is a *burden* for the compiler

Some compilers do not generate `eh_frame` at all

OCaml `eh_frame` code was contributed recently by JaneStreet

To enable binary profilers!!!

The sad truths

Generating `eh_frame` is a *burden* for the compiler

Some compilers do not generate `eh_frame` at all

OCaml `eh_frame` code was contributed recently by JaneStreet

Did you ever attempt **debugging**
jit-generated assembly gone wrong?

The sad truths

Generating `eh_frame` is a *burden* for the compiler

Some compilers do not generate `eh_frame` at all

OCaml `eh_frame` code was contributed recently by JaneStreet

Did you ever attempt **debugging**
jit-generated assembly gone wrong?

Or manually code **unwinding instructions** in **inline asm**?

```
#define LLL_STUB_UNWIND_INFO_START
    ".section  .eh_frame,\"a\",@progbits\n"
"5:\t" ".long 7f-6f              # Length of Common Information Entry\n”
"6:\t" ".long 0x0                # CIE Identifier Tag\n\t"
    ".byte 0x1                   # CIE Version\n\t"
    ".ascii \"zR\\0\"            # CIE Augmentation\n\t"
    ".uleb128 0x1                # CIE Code Alignment Factor\n\t”
    ".sleb128 -4                 # CIE RA Column\n\t”
    ".byte 0x8                   # Augmentation size\n\t"
    ".uleb128 0x1                # FDE Encoding (pcrel sdata4)\n\t”
    ".byte 0x1b                  # DW_CFA_def_cfa\n\t"
    ".byte 0xc
    ".uleb128 0x4\n\t"
    ".uleb128 0x0\n\t"
    ".align 4\n"
"7:\t" ".long 17f-8f             # FDE Length\n"
"8:\t" ".long 8b-5b              # FDE CIE offset\n\t”
    ".long 1b-.                  # FDE initial location\n\t"
    ".long 4b-1b                 # FDE address range\n\t"
    ".uleb128 0x0                # Augmentation size\n\t”
    ".byte 0x16                  # DW_CFA_val_expression\n\t”
    ".uleb128 0x8\n\t"
    ".uleb128 10f-9f\n"
"9:\t" ".byte 0x78               # DW_OP_breg8\n\t"
    ".sleb128 3b-1b\n"
```

glibc: lowlevellock.h

```
#define LLL_STUB_UNWIND_INFO_START
    ".section  .eh_frame,\"a\",@progbits\n"
"5:\t" ".long 7f-6f            # Length of Common Information Entry\n”
"6:\t" ".long 0x0              # CIE Identifier Tag\n\t"
    ".byte 0x1                 # CIE Version\n\t"
```

Despite great care, off by 1 offset error…

```
    ".byte 0x1b
    ".byte 0xc                 # DW_CFA_def_cfa\n\t"
    ".uleb128 0x4\n\t"
    ".uleb128 0x0\n\t"
    ".align 4\n"
"7:\t" ".long 17f-8f           # FDE Length\n"
"8:\t" ".long 8b-5b            # FDE CIE offset\n\t”
    ".long 1b-.                # FDE initial location\n\t"
    ".long 4b-1b               # FDE address range\n\t"
    ".uleb128 0x0              # Augmentation size\n\t”
    ".byte 0x16                # DW_CFA_val_expression\n\t”
    ".uleb128 0x8\n\t"
    ".uleb128 10f-9f\n"
"9:\t" ".byte 0x78             # DW_OP_breg8\n\t"
    ".sleb128 3b-1b\n"
```

glibc: lowlevellock.h

```
#define LLL_STUB_UNWIND_INFO_START
    ".section  .eh_frame,\"a\",@progbits\n"
"5:\t" ".long 7f-6f             # Length of Common Information Entry\n"
"6:\t" ".long 0x0               # CIE Identifier Tag\n\t"
    ".byte 0x1                  # CIE Version\n\t"
```

Despite great care, off by 1 offset error...

```
Breakpoint 2, 0x0000000000406c2c in _L_lock_19 ()
(gdb) disass
Dump of assembler code for function _L_lock_19:
=> 0x0000000000406c2c <+0>:     lea    0x2ba13d(%rip),%rdi        # 0x6c0d70 <lock>
   0x0000000000406c33 <+7>:     sub    $0x80,%rsp
   0x0000000000406c3a <+14>:    callq  0x436d10 <__lll_lock_wait_private>
   0x0000000000406c3f <+19>:    add    $0x80,%rsp
   0x0000000000406c46 <+26>:    jmpq   0x4069c6 <abort+70>
End of assembler dump.
(gdb) bt
#0  0x0000000000406c2c in _L_lock_19 ()
#1  0x0000000000406c3f in _L_lock_19 ()
#2  0x00000000004069c6 in abort ()
#3  0x0000000000401017 in main () at foo1.c:4
(gdb)
```

```
"9:\t" ".byte 0x78              # DW_OP_breg8\n\t"
    ".sleb128 3b-1b\n"
```

glibc: lowlevellock.h

Synthesis of unwinding tables

A Simple `rsp`/`rbp`-based Synthesis Strategy

Upon entering a function:

$$CFA \;=\; \texttt{rsp} - 8 \qquad\qquad RA \;=\; CFA + 8$$

Check the semantics of each instruction in each basic block:

— *Is* `rbp` *used as a base pointer?*

— *Is* `rsp` *modified?*

— *update the computation of CFA accordingly*

Chain across basic blocks following the control-flow graph

fixpoint immediate!

Some extra complexity to trace the addresses of callee-saved registers

A Simple `rsp`/`rbp`-based Synthesis Strategy

Proof of concept

Major obstacle: tools (or lack thereof) to analyse binaries

— *Is* `rbp` *used as a base pointer?*

— *Is* `rsp` *modified?*

— *update the computation of CFA accordingly*

Chain across basic blocks following the control-flow graph

fixpoint immediate!

Some extra complexity to trace the addresses of callee-saved registers

Fixpoint fallback

Compiler relies on base pointer
even if
-fomit-frame-pointer is specified

```
int main() {

 read_integer(z_max);

 for(int z=0; z < z_max; z++) {
   int x[z];
   …do something with x...
 }
}
```

LOC	CFA	rbx	rbp	r12	r13	r14	r15	ra
0000000000400526	rsp+8	u	u	u	u	u	u	c-8
0000000000400527	rsp+16	u	c-16	u	u	u	u	c-8
000000000040052a	rbp+16	u	c-16	u	u	u	u	c-8
0000000000400537	rbp+16	c-56	c-16	c-48	c-40	c-32	c-24	c-8
00000000004005cf	rsp+8	c-56	c-16	c-48	c-40	c-32	c-24	c-8

Fuzzy tests DWARF interpreters

Validate unwinding tables against the binary code

Synthesize unwinding tables from the binary/asm code

Identify bugs in generation of tables
Prevent code injection attacks
Make unwinding tables available to other systems
(inline assembly / JIT compilers)

Wider horizons

.debug_type: A Model of Data Types

```
 Compilation Unit @ offset 0x0:
  Length:        0x7e (32-bit)
  Version:       4
  Abbrev Offset: 0x0
  Pointer Size:  8
<0><b>: Abbrev Number: 1 (DW_TAG_compile_unit)
   <c>   DW_AT_producer    : Apple LLVM v9.0.0
   <10>   DW_AT_language    : 12  (ANSI C99)
   <12>   DW_AT_name        : hand.c
   <16>   DW_AT_stmt_list   : 0x0
   <1a>   DW_AT_comp_dir    : /Users/zappa/tmp
   <1e>   DW_AT_low_pc      : 0x0
   <26>   DW_AT_high_pc     : 0x1b
<1><2a>: Abbrev Number: 2 (DW_TAG_subprogram)
   <2b>   DW_AT_low_pc      : 0x0
   <33>   DW_AT_high_pc     : 0x1b
   <37>   DW_AT_frame_base  : (DW_OP_reg6 (rbp))
   <39>   DW_AT_name        : foo
   <3d>   DW_AT_decl_file   : 1
   <3e>   DW_AT_decl_line   : 1
   <3f>   DW_AT_prototyped  : 1
   <3f>   DW_AT_type        : <0x6e>
   <43>   DW_AT_external    : 1
<2><43>: Abbrev Number: 3 (DW_TAG_formal_parameter)
   <44>   DW_AT_location    :(DW_OP_fbreg: -4)
   <47>   DW_AT_name        : x
   <4b>   DW_AT_decl_file   : 1
   <4c>   DW_AT_decl_line   : 1
   <4d>   DW_AT_type        : <0x6e>
<2><51>: Abbrev Number: 3 (DW_TAG_formal_parameter)
   <52>   DW_AT_location    : (DW_OP_fbreg: -6)
   <55>   DW_AT_name        : y
   <59>   DW_AT_decl_file   : 1
   <5a>   DW_AT_decl_line   : 1
   <5b>   DW_AT_type        : <0x75>
<2><5f>: Abbrev Number: 3 (DW_TAG_formal_parameter)
   <60>   DW_AT_location    : (DW_OP_fbreg: -16)
   <63>   DW_AT_name        : z
   <67>   DW_AT_decl_file   : 1
   <68>   DW_AT_decl_line   : 1
   <69>   DW_AT_type        : <0x7c>
<2><6d>: Abbrev Number: 0
<1><6e>: Abbrev Number: 4 (DW_TAG_base_type)
   <6f>   DW_AT_name        : int
   <73>   DW_AT_encoding    : 5   (signed)
   <74>   DW_AT_byte_size   : 4
<1><75>: Abbrev Number: 4 (DW_TAG_base_type)
   <76>   DW_AT_name        : short
   <7a>   DW_AT_encoding    : 5   (signed)
   <7b>   DW_AT_byte_size   : 2
<1><7c>: Abbrev Number: 5 (DW_TAG_pointer_type)
   <7d>   DW_AT_type        : <0x6e>
<1><81>: Abbrev Number: 0
```

```
<2><5f>: Abbrev Number: 3 (DW_TAG_formal_parameter)
    <60>   DW_AT_location    : (DW_OP_fbreg: -16)
    <63>   DW_AT_name        : z
    <67>   DW_AT_decl_file   : 1
    <68>   DW_AT_decl_line   : 1
    <69>   DW_AT_type        : <0x7c>
    <2><6d>: Abbrev Number: 0

<1><6e>: Abbrev Number: 4 (DW_TAG_base_type)
    <6f>   DW_AT_name        : int
    <73>   DW_AT_encoding    : 5(signed)
    <74>   DW_AT_byte_size   : 4

<1><7c>: Abbrev Number: 5 (DW_TAG_pointer_type)
 <7d>   DW_AT_type        : <0x6e>
```

```
<44>   DW_AT_location    :(DW_OP_fbreg: -4)
<47>   DW_AT_name        : x
<4b>   DW_AT_decl_file   : 1
<4c>   DW_AT_decl_line   : 1
<4d>   DW_AT_type        : <0x6e>
```

```
   <7a>   DW_AT_encoding    : 5   (signed)
   <7b>   DW_AT_byte_size   : 2
<1><7c>: Abbrev Number: 5 (DW_TAG_pointer_type)
   <7d>   DW_AT_type        : <0x6e>
<1><81>: Abbrev Number: 0
```

```
<2><5f>: Abbrev Number: 3 (DW_TAG_formal_parameter)
    <60>   DW_AT_location    : (DW_OP_fbreg: -16)
    <63>   DW_AT_name        : z
    <67>   DW_AT_decl_file   : 1
    <68>   DW_AT_decl_line   : 1
    <69>   DW_AT_type        : <0x7c>
    <2><6d>: Abbrev Number: 0

<1><6e>: Abbrev Number: 4 (DW_TAG_base_type)
    <6f>   DW_AT_name        : int
    <73>   DW_AT_encoding    : 5(signed)
    <74>   DW_AT_byte_size   : 4

<1><7c>: Abbrev Number: 5 (DW_TAG_pointer_type)
 <7d>   DW_AT_type        : <0x6e>
```

```
<44>   DW_AT_location    :(DW_OP_fbreg: -4)
<47>   DW_AT_name        : x
<4b>   DW_AT_decl_file   : 1
<4c>   DW_AT_decl_line   : 1
<4d>   DW_AT_type        : <0x6e>
```

```
   <7a>   DW_AT_encoding    : 5   (signed)
   <7b>   DW_AT_byte_size   : 2
<1><7c>: Abbrev Number: 5 (DW_TAG_pointer_type)
   <7d>   DW_AT_type        : <0x6e>
<1><81>: Abbrev Number: 0
```

```
<2><5f>: Abbrev Number: 3 (DW_TAG_formal_parameter)
    <60>   DW_AT_location     : (DW_OP_fbreg: -16)
    <63>   DW_AT_name         : z
    <67>   DW_AT_decl_file    : 1
    <68>   DW_AT_decl_line    : 1
    <69>   DW_AT_type         : <0x7c>
    <2><6d>: Abbrev Number: 0

<1><6e>: Abbrev Number: 4 (DW_TAG_base_type)
    <6f>   DW_AT_name         : int
    <73>   DW_AT_encoding     : 5(signed)
    <74>   DW_AT_byte_size    : 4

<1><7c>: Abbrev Number: 5 (DW_TAG_pointer_type)
 <7d>   DW_AT_type         : <0x6e>
```

```
<44>   DW_AT_location     :(DW_OP_fbreg: -4)
<47>   DW_AT_name         : x
<4b>   DW_AT_decl_file    : 1
<4c>   DW_AT_decl_line    : 1
<4d>   DW_AT_type         : <0x6e>
```

```
   <7a>   DW_AT_encoding     : 5   (signed)
   <7b>   DW_AT_byte_size    : 2
<1><7c>: Abbrev Number: 5 (DW_TAG_pointer_type)
   <7d>   DW_AT_type         : <0x6e>
<1><81>: Abbrev Number: 0
```

```
<2><5f>: Abbrev Number: 3 (DW_TAG_formal_parameter)
    <60>   DW_AT_location      : (DW_OP_fbreg: -16)
    <63>   DW_AT_name          : z
    <67>   DW_AT_decl_file     : 1
    <68>   DW_AT_decl_line     : 1
    <69>   DW_AT_type          : <0x7c>
    <2><6d>: Abbrev Number: 0

<1><6e>: Abbrev Number: 4 (DW_TAG_base_type)
    <6f>   DW_AT_name          : int
    <73>   DW_AT_encoding      : 5(signed)
    <74>   DW_AT_byte_size     : 4

<1><7c>: Abbrev Number: 5 (DW_TAG_pointer_type)
 <7d>   DW_AT_type          : <0x6e>
```

```
<44>    DW_AT_location      :(DW_OP_fbreg: -4)
<47>    DW_AT_name          : x
<4b>    DW_AT_decl_file     : 1
<4c>    DW_AT_decl_line     : 1
<4d>    DW_AT_type          : <0x6e>
```

```
    <7a>   DW_AT_encoding      : 5    (signed)
    <7b>   DW_AT_byte_size     : 2
<1><7c>: Abbrev Number: 5 (DW_TAG_pointer_type)
    <7d>   DW_AT_type          : <0x6e>
<1><81>: Abbrev Number: 0
```

.debug_line: synchronising source and object

```
CU: /Users/zappa/tmp/hand.c:
File name                Line number    Starting address
hand.c                       5           0x8048604
hand.c                       6           0x804860a
hand.c                       9           0x8048613
hand.c                      10           0x804861c
hand.c                       9           0x8048630
hand.c                      11           0x804863c
hand.c                      15           0x804863e
hand.c                      16           0x8048647
hand.c                      17           0x8048653
hand.c                      18           0x8048658
```

Column numbers are also supported

If DWARF tables are correct

by combining informations in different tables

1. ***we have metadata about computation at runtime***

Example: type information for all stack/register allocated variables

Outcome: identify roots for a ***precise*** **garbage collector** for C

runtime type-checking for C

2. ***we can relate source and machine code***

Outcome: ***precise provenance informations***

translation validation for existing C compilers

If DWARF tables are correct

by combining informations in different tables

1. *we hav*

Example: t

Outcome:

2. *we can*

Outcome:

```
if (obj->type == OBJ_COMMIT) {
  if (process_commit(walker, (struct commit *)iobj))
    return -1;
  return 0;
  }
}
```

Dynamically Diagnosing Type Errors in Unsafe Code

Stephen Kell

OOPSLA'16

1. *we hav*

Example: t

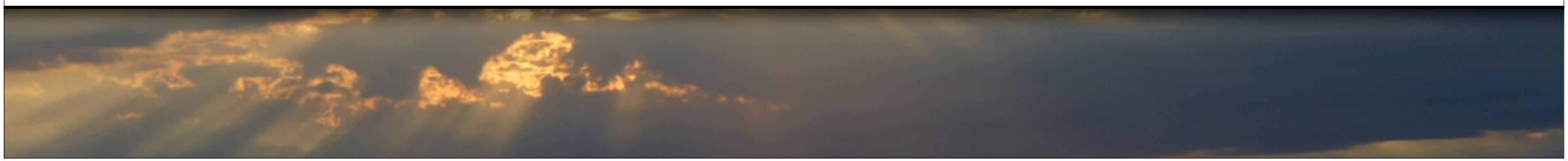
Outcome:

check this at run-time

```
if (obj->type == OBJ_COMMIT) {
  if (process_commit(walker, (struct commit *)iobj))
    return -1;
  return 0;
  }
}
```

2. *we can*

Outcome:



Dynamically Diagnosing Type Errors in Unsafe Code

Stephen Kell

OOPSLA'16

1. we hav

Example: t

Outcome:

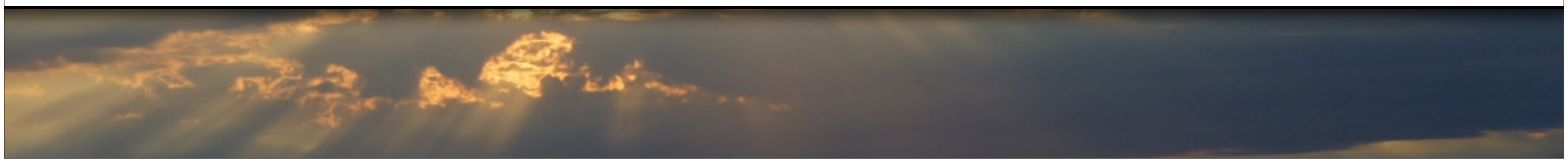
2. we can

Outcome:

check this at run-time

```
if (obj->type == OBJ_COMMIT) {
  if (process_commit(walker, (struct commit *)iobj))
    return -1;
  return 0;
  }
}
```

binary compatible
source compatible
reasonable performance
not C-specific



Entropy Workshop

Shopping List

1. specify the architecture
2. specify the C source language
3. verify the compiler
4. specify & verify the hardware
5. specify & verify functional properties

A new kernel is born

```
ld -m elf_x86_64 --build-id -o vmlinux \
 -T arch/x86/kernel/vmlinux.lds \
 arch/x86/kernel/head{_64,64,}.o \
 arch/x86/kernel/init_task.o init/built-in.o \
 --start-group \
 {usr,arch/x86,kernel,mm,fs}/built-in.o \
 {ipc,security,crypto,block}/built-in.o \
 lib/lib.a arch/x86/lib/lib.a \
 lib/built-in.o arch/x86/lib/built-in.o \
 {drivers,sound,firmware}/built-in.o \
 {arch/x86/{pci,power,video},net}/built-in.o \
 --end-group \
.tmp_kallsyms2.o
```

A new kernel is born

```
ld -m elf_x86_64 --build-id -o vmlinux \
 -T arch/x86/kernel/vmlinux.lds \
 arch/x86/kernel/head{_64,64,}.o \
 arch/x86/kernel/init_task.o init/built-in.o \
 --start-group \
 {usr,arch/x86,kernel,mm,fs}/built-in.o \
```

What is happening in this link command?

```
 {arch/x86/{pci,power,video},net}/built-in.o \
 --end-group \
.tmp_kallsyms2.o
```

Of POPLs past

In this paper we provide a context where linking can be studied, and separate compilability can be formally stated and checked. We propose a framework where each module is separately compiled to a self-contained entity called a *linkset*; we show that separately compiled, compatible modules can be safely linked together.

Luca Cardelli
"Program Fragments, Linking, and Modularisation"
POPL '97

Of POPLs past

In this paper we provide a context where linking can be studied, and separate compilability can be formally stated and checked. We propose a framework where each module is separately compiled to a self-contained entity called a *linkset*; we show that separately compiled, compatible modules can be safely linked together.

Is separate compilation the substance of linking?

POPL '97

A new kernel is born

```
ld -m elf_x86_64 --build-id -o vmlinux \
 -T arch/x86/kernel/vmlinux.lds \
 arch/x86/kernel/head{_64,64,}.o \
 arch/x86/kernel/init_task.o init/built-in.o \
 --start-group \
 {usr,arch/x86,kernel,mm,fs}/built-in.o \
 {ipc,security,crypto,block}/built-in.o \
 lib/lib.a arch/x86/lib/lib.a \
 lib/built-in.o arch/x86/lib/built-in.o \
 {drivers,sound,firmware}/built-in.o \
 {arch/x86/{pci,power,video},net}/built-in.o \
 --end-group \
.tmp_kallsyms2.o
```

A new kernel is born

```
ld -m elf_x86_64 --build-id -o vmlinux \
 -T arch/x86/kernel/vmlinux.lds \
 arch/x86/kernel/head{_64,64,}.o \
 arch/x86/kernel/init_task.o init/built-in.o \
 --start-group \
 {usr,arch/x86,kernel
 {ipc,security,crypto
 lib/lib.a arch/x86/l
 lib/built-in.o arch/x86/lib/built-in.o \
 {drivers,sound,firmware}/built-in.o \
 {arch/x86/{pci,power,video},net}/built-in.o \
 --end-group \
.tmp_kallsyms2.o
```

Linker script

specifies what the linker should do

System software is written in C, right? With a bit of assembly?

C structure layout cannot be used, because the definitions must be addressable symbolically from assembly as well as from C.

System software is written in C, right? With a bit of assembly?

```
/* NOTE: gcc doesn't actually guarantee that global
 * objects will be laid out in memory in the order
 * of declaration, so put these in different sections
 * and use the linker
```

C structure layout cannot be used, because the definitions must be addressable symbolically from assembly as well as from C.

```
r them. */

pmd_t pmd0[PTRS_PER_P
__ ((
  __section__ (".data..vm0.pmd"), aligned(PAGE SIZE)));
pgd_t swapper_pg_dir[PTRS_PER_PGD] __attribute__ ((
  __section__ (".data..vm0.pgd"), aligned(PAGE SIZE)));
pte_t pg0[PT_INITIAL * PTRS_PER_PTE] __attribute__ ((
  __section__ (".data..vm0.pte"), aligned(PAGE SIZE)));
```

It’s this whole other language

```
/* Put page table entries (swapper_pg_dir) as the first
 * thing in .bss. This ensures that it has bss
 * alignment (PAGE SIZE).
 */

. = ALIGN(bss_align);

.bss : AT(ADDR(.bss) - LOAD_OFFSET) {
  *(.data..vm0.pmd) *(.data..vm0.pgd) *(.data..vm0.pte)
  *(.bss..page aligned)
  *(.dynbss) *(.bss)
  *(COMMON)
}
```

It's this whole other language

```
/* Put page table entries (swapper_pg_dir) as the first
 * thing in .bss. This ensures that it has bss
 * alignment (PAGE SIZE).
 */

. = ALIGN(bss_align);

.bss : AT(ADDR(.bss) - LOAD_OFFSET) {
  *(.data..vm0.pmd) *(.data..vm0.pgd) *(.data..vm0.pte)
  *(.bss..page aligned)
  *(.dynbss) *(.bss)
  *(COMMON)
}
```

vmlinuz.lds is **408 lines** long

linker-speak super powers

- Memory layout
- Memory placement
- Inter-module encapsulation
- Inter-module binding
- Inter-module versioning
- Link-time deduplication
- Build-time flexibility & configuration
- Extensibility
- Instrumentation
- Introspection

```c
/* Write formatted output to STREAM from the
 * format string FORMAT.
 */
int __fprintf (FILE *stream, const char *format, ...) {
  va_list arg; int done;
  va_start (arg, format);
  done = vfprintf (stream, format, arg); va end (arg);
  return done;
}

ldbl_hidden_def ( __fprintf , fprintf )
ldbl_strong_alias ( __fprintf , fprintf )
/* We define the function with the real name here.
 * But deep down in libio the original function IO
 * fprintf is also needed. So make an alias.
 */
ldbl_weak_alias ( __fprintf , _IO_fprintf )
```

Doesn't this matter only for obscure systems code?

```
void *malloc(size_t sz)
{ /* my own malloc */ }

int main(void)
{ // ...
  int *is = malloc(42 * sizeof (int));
}
```

Will it call my `malloc()` or the "other" one?

- statically or dynamically linked?
- what linker options?
- what compiler options?
- where does the other `malloc()` come from?

Doesn't this matter only for obscure systems code?

```
if (&_IO_stdin_used != NULL)
  /* do something ... */
else
  /* do something else ... */
```

Is the **else** branch ever taken?

Doesn’t this matter only for obscure systems code?

```
if (&_IO_stdin_used != NULL)
  /* do something ... */
else
  /* do something else ... */
```

Is the **else** branch ever taken?

- Expect that the address of any global variable is non-null?
- The C11 spec (6.5.3.2 pt 3) suggests that the "else" cannot run
- Not true if you have weak symbols
- In practice, on most implementations of C, it can

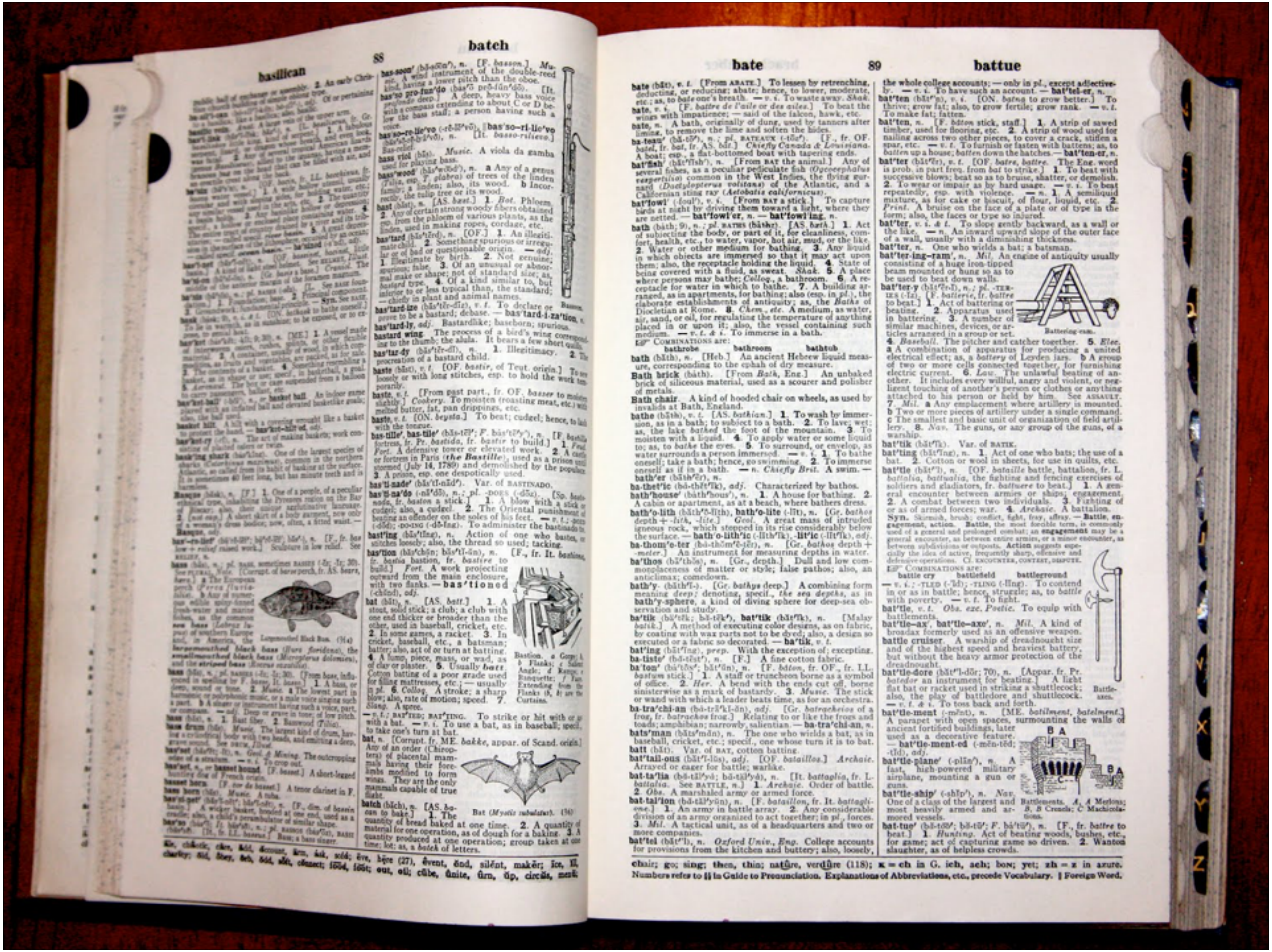
The linker can change the semantics of the programming language

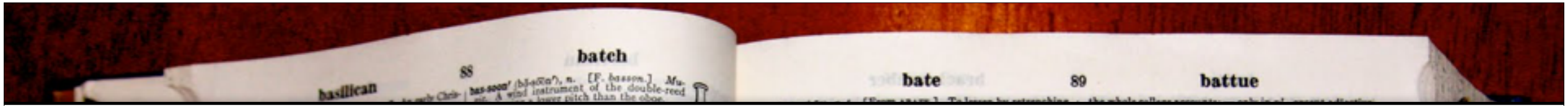

We surely need a new language!


We surely need a new language!
Maybe in fact we need
semantics for linker-speak

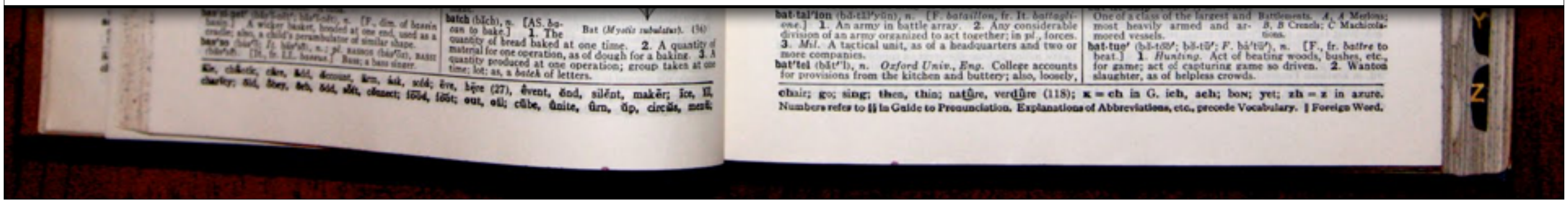
Updated Shopping List

1. specify the object file formats
2. specify the linker's own format
3. verify the linker
4. go back to the other shopping list


basilican
88
batch
bate
89
battue



System V Unix included a detailed description of the linker.



System V Unix included a detailed description of the linker.

Removed when ELF was introduced, replaced by a man page.

System V Unix included a detailed description of the linker.

Removed when ELF was introduced, replaced by a man page.

The Posix standard omits the man page.

One good linker deserves another

- 1972: AT&T Unix linker
- 1977: BSD linker
- c. 1983: original GNU linker
- 1988: System V r4 linker (introduces ELF)
- c. 1990: GNU BFD linker
- 2008: GNU gold linker
- c. 2012: LLVM lld linker

A common ambition: be *mostly like that other linker*

Validation: do "they seem to work"?

```
$ ld -o OUTPUT /lib/crt0.o hello.o -lc
```

GNU LD

The linker will search an archive only once, at the location where it is specified on the command line. If the archive defines a symbol which was undefined in some object which appeared before the archive on the command line, the linker will include the appropriate file(s) from the archive. However, an undefined symbol in an object appearing later on the command line will not cause the linker to search the archive again.

See the '-(' option for a way to force the linker to search archives multiple times.

```
$ ld -o OUTPUT /lib/crt0.o hello.o -lc
```

GNU LD

The linker will search an archive only once, at the location where it is specified on the command line. If the archive defines a symbol which was undefined in some object which appeared before the archive on the command line, the linker will include the appropriate file(s) from the archive. However, an undefined symbol in an object appearing later on the command line will not cause the linker to search the archive again.

See the '-(' option for a way to force the linker to search archives multiple times.

But other linkers implement something slightly different...

1. specify the object file formats
2. specify the linker's own format
3. verify the linker
4. go back to the other shopping list

1. specify the object file form

2. specify the linke

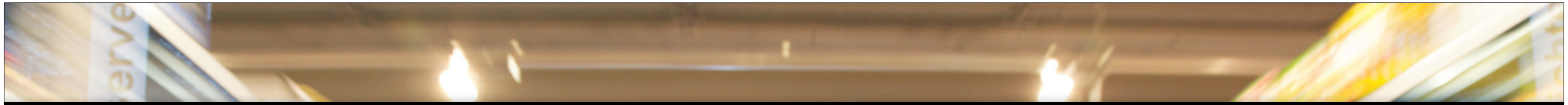
3. verify th

4. er shopping list

The Missing Link: Explaining ELF Static Linking, Semantically

Stephen Kell Dominic P. Mulligan Peter Sewell

OOPSLA'16



Executable Lem spec of ELF static linking

ELF file format

Executable, actually working linker!

Archs: x86-64 and AArch64, PPC64, IA32

Readable!

About 2 person-years of effort so far…

Key steps

- read command line
- gather input files (incl. archives, scripts)
- resolve symbols
- discard unneeded inputs
- *size support structures (GOT, PLT, . . .)*
- *interpret linker script…*
- ... one pass to define & size output
- … another pass to place output
- *complete support structures*
- *apply relocations*
- write output file

Status

- **ELF Spec well-validated**
- Linking spec not quite complete
  - some looseness (e.g. in link order) not captured yet
  - ABI-specific optimisations not modelled
- **More than a reference implementation**:
  - capture space of permitted links
  - **usable in proof**
- **Not yet usable as test oracle but not far off**…

Did you say proofs?

- Written in Lem http://www.cl.cam.ac.uk/~pes20/lem/

  Extracted to Isabelle/HOL (33,150 lines)

- Proved termination of linker on all inputs (~1.5k lines)

- Proved a correctness theorem about a (very simple) relocation

  For AMD64

  Around 4,500 lines (mostly re-usable lemmas)

Export DWARF
metadata at run-time

Formalisation of binary formats
including compiler specific extensions

Translation validation
via DWARF

Validation and synthesis of DWARF tables

C language design and analysis
to integrate linker semantics

Precise provenance
relations

Executable specification of linker speak

Policy-based linker speak

We introduce our design and implementation of Cobbler, a proof-of-concept toolkit capable of compiling a Turing-complete language into well-formed ELF executable metadata that get "executed" by the runtime loader (RTLD). Our proof-of-concept toolkit highlights how important it is that defenders expand their focus beyond the code and data sections of untrusted binaries, both in static analysis and in the dynamic analysis of the early runtime setup stages as well as any time the RTLD is invoked.

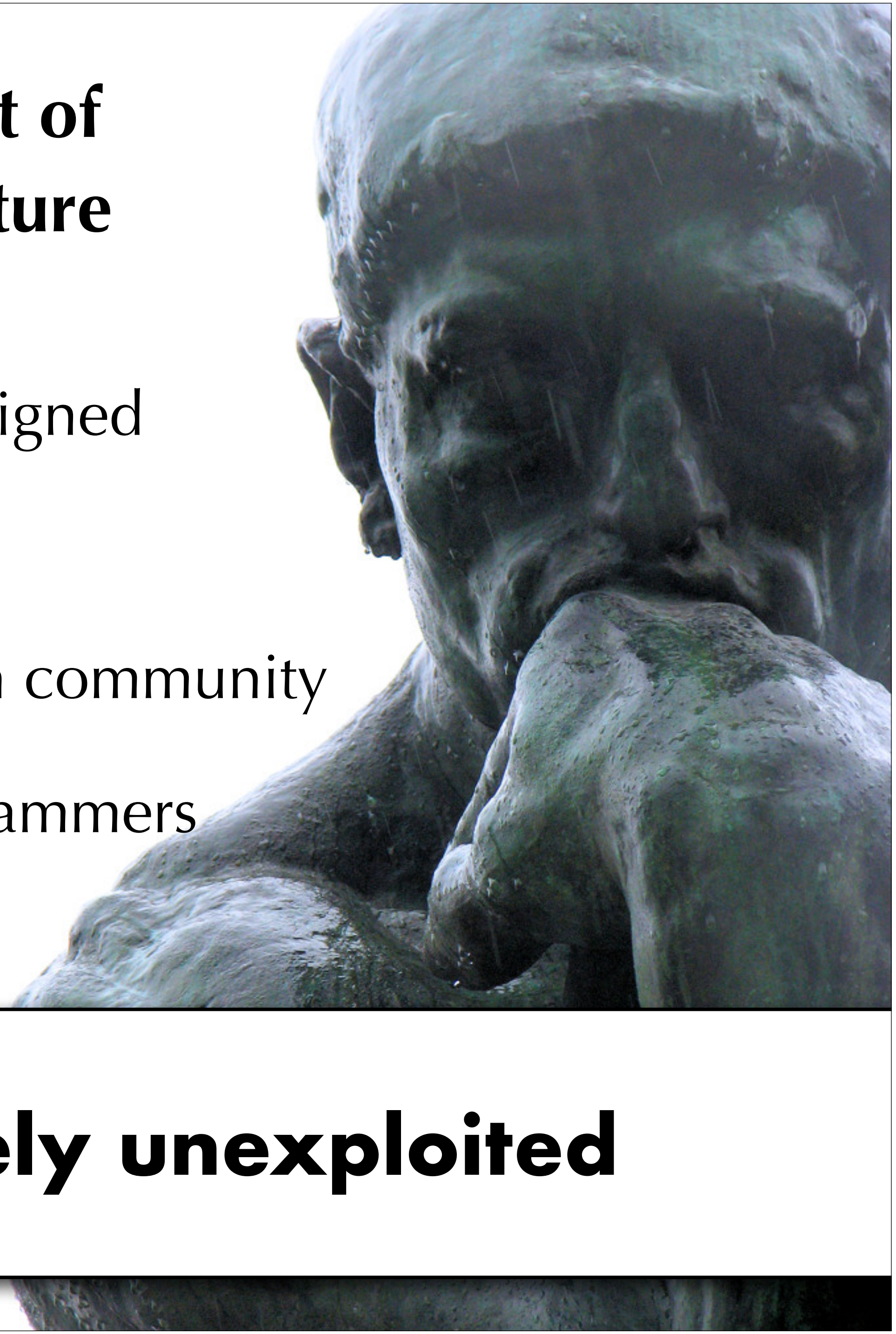
"Weird Machines" in ELF: A Spotlight on the Underappreciated Metadata

Rebecca Shapiro
Sergey Bratus
Sean W. Smith

WOOT'13

We introduce our design a…
tation of Cobbler, a proof-of-conc…
compiling a Turing-complet… run-
ELF executable meta…
time loader (RT… high-
lights h… …ers expand their fo-
…sections of untrusted bina-
…ysis and in the dynamic analysis
…ne setup stages as well as any time the
…nvoked.

Binary formats: the heart of our computing infrastructure



Poorly specified and badly designed

Pervaded by subtle bugs

Mostly ignored by the research community

Not understood by most programmers

Potential largely unexploited

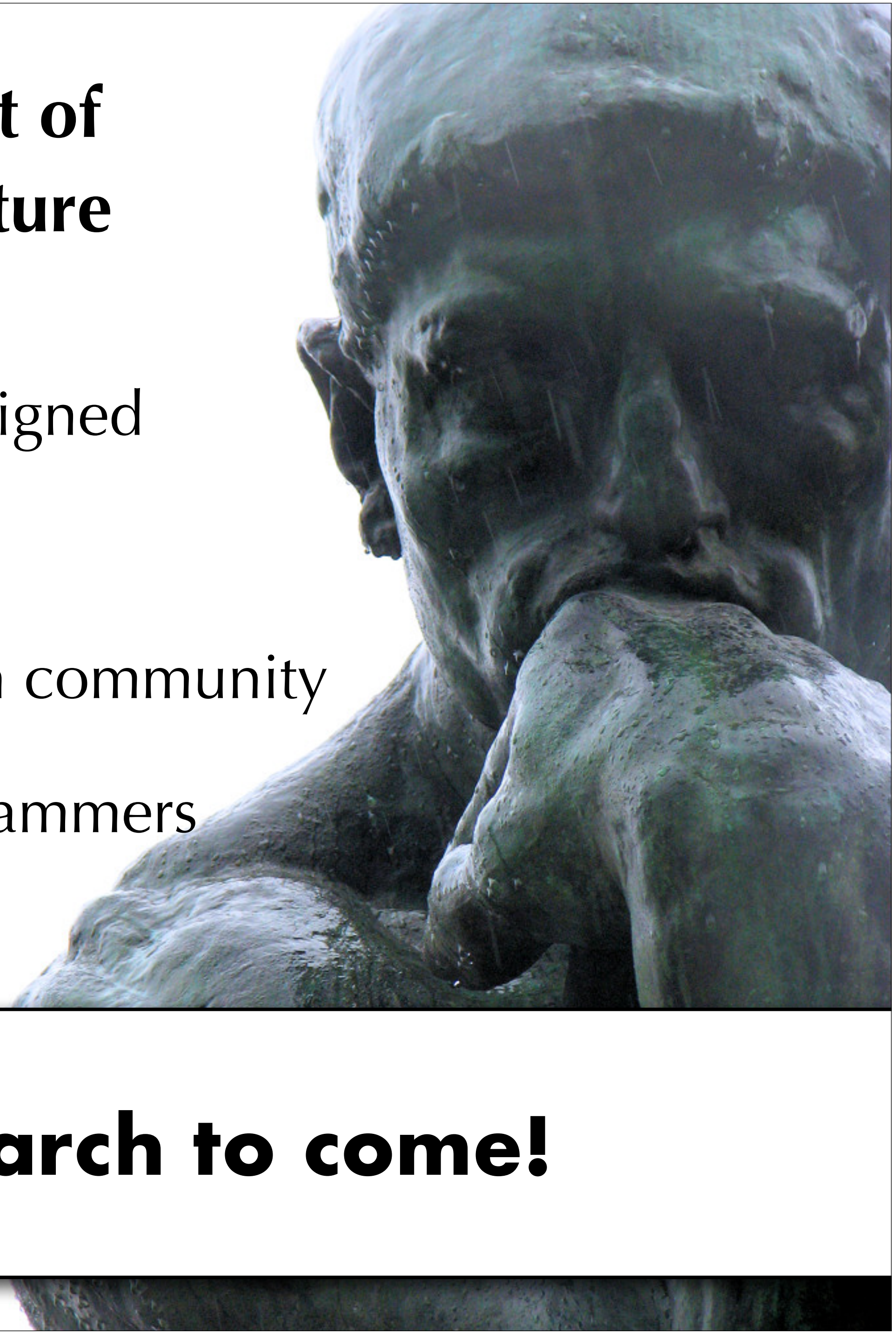
Binary formats: the heart of our computing infrastructure

Poorly specified and badly designed

Pervaded by subtle bugs

Mostly ignored by the research community

Not understood by most programmers



Exciting research to come!