Reliable and Fast DWARF-based Stack Unwinding

Théophile Bastian
Stephen Kell
Francesco Zappa Nardelli

ENS Paris, University of Kent, Inria

Webpage (incl. slides)
https://huit.re/frdwarf

Funding
ONR VerticA
Google Research Fellowship
$ ./a.out
Segmentation fault.
$ ./a.out
Segmentation fault.

(gdb) backtrace
#0 0x54625 in fct_b
#1 0x54663 in fct_a
#2 0x54674 in main
$ ./a.out
Segmentation fault.

(gdb) backtrace
#0  0x54625 in fct_b
#1  0x54663 in fct_a
#2  0x54674 in main

How does it work?
$ ./a.out
Segmentation fault.

(gdb) backtrace
#0 0x54625 in fct_b
#1 0x54663 in fct_a
#2 0x54674 in main

How does it work?
How do we get the return address?

![Diagram showing stack layout with return address and stack pointer.]

- main
- fct_a: 0x54674
- fct_b: 0x54663
- Stack pointer (%rsp)
How do we get the return address?

What if we only have %rsp?
## DWARF unwinding data

<table>
<thead>
<tr>
<th>PC</th>
<th>CFA</th>
<th>rbx</th>
<th>rbp</th>
<th>r12</th>
<th>r13</th>
<th>r14</th>
<th>r15</th>
<th>ra</th>
</tr>
</thead>
<tbody>
<tr>
<td>0084950</td>
<td>rsp+8</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>c-8</td>
</tr>
<tr>
<td>0084952</td>
<td>rsp+16</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>0084954</td>
<td>rsp+24</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>0084956</td>
<td>rsp+32</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>0084958</td>
<td>rsp+40</td>
<td>u</td>
<td>u</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>0084959</td>
<td>rsp+48</td>
<td>u</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>008495a</td>
<td>rsp+56</td>
<td>c-56</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>0084962</td>
<td>rsp+64</td>
<td>c-56</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>0084a19</td>
<td>rsp+56</td>
<td>c-56</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>0084a1d</td>
<td>rsp+48</td>
<td>c-56</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>0084a1e</td>
<td>rsp+40</td>
<td>c-56</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
</tbody>
</table>
DWARF unwinding data

<table>
<thead>
<tr>
<th>PC</th>
<th>CFA</th>
<th>rbx</th>
<th>rbp</th>
<th>r12</th>
<th>r13</th>
<th>r14</th>
<th>r15</th>
<th>ra</th>
</tr>
</thead>
<tbody>
<tr>
<td>0084950</td>
<td>rsp+8</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>c-8</td>
</tr>
<tr>
<td>0084952</td>
<td>rsp+16</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>c-16 c-8</td>
</tr>
<tr>
<td>0084954</td>
<td>rsp+24</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>c-24 c-16 c-8</td>
<td></td>
</tr>
<tr>
<td>0084956</td>
<td>rsp+32</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>c-32 c-24 c-16 c-8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0084958</td>
<td>rsp+40</td>
<td>u</td>
<td>u</td>
<td>c-40 c-32 c-24 c-16 c-8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0084959</td>
<td>rsp+48</td>
<td>u</td>
<td>c-48 c-40 c-32 c-24 c-16 c-8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>008495a</td>
<td>rsp+56</td>
<td>c-56 c-48 c-40 c-32 c-24 c-16 c-8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0084962</td>
<td>rsp+64</td>
<td>c-56 c-48 c-40 c-32 c-24 c-16 c-8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0084a19</td>
<td>rsp+56</td>
<td>c-56 c-48 c-40 c-32 c-24 c-16 c-8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0084a1d</td>
<td>rsp+48</td>
<td>c-56 c-48 c-40 c-32 c-24 c-16 c-8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0084a1e</td>
<td>rsp+40</td>
<td>c-56 c-48 c-40 c-32 c-24 c-16 c-8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For each instruction...  
(identified by its program counter)
DWARF unwinding data

<table>
<thead>
<tr>
<th>PC</th>
<th>CFA</th>
<th>rbx</th>
<th>rbp</th>
<th>r12</th>
<th>r13</th>
<th>r14</th>
<th>r15</th>
<th>ra</th>
</tr>
</thead>
<tbody>
<tr>
<td>0084950</td>
<td>rsp+8</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>c-8</td>
</tr>
<tr>
<td>0084952</td>
<td>rsp+16</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>0084954</td>
<td>rsp+24</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>0084956</td>
<td>rsp+32</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>0084958</td>
<td>rsp+40</td>
<td>u</td>
<td>u</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>0084959</td>
<td>rsp+48</td>
<td>u</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>008495a</td>
<td>rsp+56</td>
<td>c-56</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>0084962</td>
<td>rsp+64</td>
<td>c-56</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>0084a19</td>
<td>rsp+56</td>
<td>c-56</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>0084a1d</td>
<td>rsp+48</td>
<td>c-56</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>0084a1e</td>
<td>rsp+40</td>
<td>c-56</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
</tbody>
</table>

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

... an expression to compute its return address location on the stack
The real Dwar
t
code for a Turing-complete stack machine
tto reconstruct the table
[...]

30 24 34 FDE pc=004020..004040
DW_CFA_def_cfa_offset: 16
DW_CFA_advance_loc: 6 to 0000000000004026
DW_CFA_def_cfa_offset: 24
DW_CFA_advance_loc: 10 to 0000000000004030
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)
The real DWARF

30 24 34 FDE pc=004020..004040
DW_CFA_def_cfa_offset: 16
DW_CFA_advance_loc: 6 to 0000000000004026
DW_CFA_def_cfa_offset: 24
DW_CFA_advance_loc: 10 to 0000000000004030
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)
[...]

→ bytecode for a Turing-complete stack machine
→ which is interpreted on demand at runtime
to reconstruct the table
What does this imply?

Your compiler generates code for **two machines**: your processor and the DWARF VM.

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

```
main:
.cfi_startproc
pushq  %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq  %rsp, %rbp
.cfi_def_cfa_register 6
subq  $32, %rsp
movl  %edi, -20(%rbp)
movq  %rsi, -32(%rbp)
```

```
.cfi_*: inline DWARF!
```
What does this imply?

Your compiler generates code for two machines: your processor and the DWARF VM.

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

`main:
.cfi_startproc
pushq  %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq  %rsp, %rbp
.cfi_def_cfa_register 6
subq  $32, %rsp
movl  %edi, -20(%rbp)
movq  %rsi, -32(%rbp)

.cfi_*: inline DWARF!`

⇒ Cumbersome to generate for the compiler
   ➞ might do it wrong
   ➞ might not do it at all

⇒ If you write inline asm, you must write inline DWARF!
.section .eh_frame,"a",@progbits
5: .long 7f-6f # Length of Common Information Entry
6: .long 0x0 # CIE Identifier Tag
   .byte 0x1 # CIE Version
   .ascii "zR\0" # CIE Augmentation
   .uleb128 0x1 # CIE Code Alignment Factor
   .sleb128 -4 # CIE RA Column
   .byte 0x8 # Augmentation size
   .uleb128 0x1 # FDE Encoding (pcrel sdata4)
   .byte 0x1b # DW_CFA_def_cfa
   .byte 0xc
   .uleb128 0x4
   .uleb128 0x0
   .align 4
7: .long 17f-8f # FDE Length
8: .long 8b-5b # FDE CIE offset
   .long 1b- # FDE initial location
   .long 4b-1b # FDE address range
   .uleb128 0x0 # Augmentation size
   .byte 0x16 # DW_CFA_val_expression
   .uleb128 0x8
   .uleb128 10f-9f
9: .byte 0x78 # DW_OP_breg8
    .sleb128 3b-1b
In glibc, lowlevellock.h:
off by one error in unwinding data.

(gdb) backtrace
#0 0x406c2c in _L_lock_19
#1 0x406c2c in _L_lock_19
#2 0x4069c6 in abort
#3 0x401017 in main
Complex & slow

.udata 

.section .eh_frame,"a",@progbits

.long 7f -6f # Length of Common Information Entry
.long 0x0 # CIE Identifier Tag
.byte 0x1 # CIE Version
.ascii "zR \0 " # CIE Augmentation
.uleb128 0x1 # CIE Code Alignment Factor
.sleb128 -4 # CIE RA Column
.byte 0x8 # Augmentation size
.uleb128 0x1 # FDE Encoding (pcrel sdata4)
.byte 0x1b # DW_CFA_def_cfa
.byte 0xc
.uleb128 0x4
.uleb128 0x0
.align 4

.long 17f -8f # FDE Length
.long 8b -5b # FDE CIE offset
.long 1b -. # FDE initial location
.long 4b -1b # FDE address range
.uleb128 0x0 # Augmentation size
.byte 0x16 # DW_CFA_val_expression
.uleb128 0x8
.uleb128 10f -9f
.byte 0x78 # DW_OP_breg8
.sleb128 3b-1b
Complex & slow

Pervasive:
relied upon by profilers, debuggers, aaand...
.section .eh_frame,"a",@progbits

Complex & slow

.uleb128 0x1  # CIE Code Alignment Factor
.sleb128 -4   # CIE RA Column

Pervasive:
relied upon by profilers,
d debuggers, aaand...
C++ exceptions.

⇝ not only for debuggers!
“Sorry, but last time was too f...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, 2012
“Sorry, but last time was too f...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, 2012

This is where we still are!
“Sorry, but last time was too f... 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.”

“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.”

— Linus Torvalds, 2012
Correctness by construction: synthesis of unwinding tables
<foo>:
  push  %r15
  push  %r14
  mov   $0x3,%eax
  push  %r13
  push  %r12
  push  %rbp
  push  %rbx
  sub   $0x68,%rsp
  add   $0x68,%rsp
  pop   %rbx
<foo>:
    push   %r15
    push   %r14
    mov    $0x3,%eax
    push   %r13
    push   %r12
    push   %rbp
    push   %rbx
    sub    $0x68,%rsp
    add    $0x68,%rsp
    pop    %rbx

    CFA     ra
    rsp+8   c-8
    rsp+16  c-8
    rsp+24  c-8
    rsp+24  c-8
    rsp+32  c-8
    rsp+40  c-8
    rsp+48  c-8
    rsp+56  c-8
    rsp+160 c-8
    rsp+56  c-8
<foo>:

<table>
<thead>
<tr>
<th></th>
<th>CFA</th>
<th>ra</th>
</tr>
</thead>
<tbody>
<tr>
<td>push</td>
<td>%r15</td>
<td>rsp+8</td>
</tr>
<tr>
<td>push</td>
<td>%r14</td>
<td>rsp+16</td>
</tr>
<tr>
<td>mov</td>
<td>$0x3,%eax</td>
<td>rsp+24</td>
</tr>
<tr>
<td>push</td>
<td>%r13</td>
<td>rsp+24</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sub</td>
<td>$0x68</td>
<td>rsp+56</td>
</tr>
<tr>
<td>add</td>
<td>$0x68</td>
<td>rsp+108</td>
</tr>
<tr>
<td>pop</td>
<td>%rbx</td>
<td></td>
</tr>
</tbody>
</table>

**Assumptions**

- the compiler generated the unwinding data
- we have a reliable DWARF interpreter
Upon function call, \( ra = *(\%rsp) \)
<foo>:

push %r15
push %r14
mov $0x3,%eax
push %r13
push %r12
push %rbp
push %rbx
sub $0x68,%rsp
add $0x68,%rsp
pop %rbx

push decreases %rsp by 8: ra = *(%rsp + 8)
<table>
<thead>
<tr>
<th>Instruction</th>
<th>CFA</th>
<th>ra</th>
</tr>
</thead>
<tbody>
<tr>
<td>push %r15</td>
<td>rsp+8</td>
<td>c-8</td>
</tr>
<tr>
<td>push %r14</td>
<td>rsp+16</td>
<td>c-8</td>
</tr>
<tr>
<td>mov $0x3,%eax</td>
<td>rsp+24</td>
<td>c-8</td>
</tr>
<tr>
<td>push %r13</td>
<td>rsp+24</td>
<td>c-8</td>
</tr>
<tr>
<td>push %r12</td>
<td>rsp+32</td>
<td>c-8</td>
</tr>
<tr>
<td>push %rbp</td>
<td>rsp+40</td>
<td>c-8</td>
</tr>
<tr>
<td>push %rbx</td>
<td>rsp+48</td>
<td>c-8</td>
</tr>
<tr>
<td>sub $0x68,%rsp</td>
<td>rsp+56</td>
<td>c-8</td>
</tr>
<tr>
<td>add $0x68,%rsp</td>
<td>rsp+160</td>
<td>c-8</td>
</tr>
<tr>
<td>pop %rbx</td>
<td>rsp+56</td>
<td>c-8</td>
</tr>
</tbody>
</table>

and again: \( ra = \ast(\%rsp + 16) \)
<foo>:

push %r15  
push %r14  
mov $0x3,%eax  
push %r13  
push %r12  
push %rbp  
push %rbx  
sub $0x68,%rsp
add $0x68,%rsp
pop %rbx

<table>
<thead>
<tr>
<th>CFA</th>
<th>ra</th>
</tr>
</thead>
<tbody>
<tr>
<td>rsp+8</td>
<td>c-8</td>
</tr>
<tr>
<td>rsp+16</td>
<td>c-8</td>
</tr>
<tr>
<td>rsp+24</td>
<td>c-8</td>
</tr>
<tr>
<td>rsp+24</td>
<td>c-8</td>
</tr>
<tr>
<td>rsp+32</td>
<td>c-8</td>
</tr>
<tr>
<td>rsp+40</td>
<td>c-8</td>
</tr>
<tr>
<td>rsp+48</td>
<td>c-8</td>
</tr>
<tr>
<td>rsp+56</td>
<td>c-8</td>
</tr>
<tr>
<td>rsp+56</td>
<td>c-8</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This mov leaves %rsp untouched:
ra = *(%rsp + 16)
```plaintext
<foo>:
    push %r15
    push %r14
    mov $0x3,%eax
    push %r13
    push %r12
    push %rbp
    push %rbx
    sub $0x68,%rsp
    add $0x68,%rsp
    pop %rbx
```

<table>
<thead>
<tr>
<th>CFA</th>
<th>ra</th>
</tr>
</thead>
<tbody>
<tr>
<td>rsp+8</td>
<td>c-8</td>
</tr>
<tr>
<td>rsp+16</td>
<td>c-8</td>
</tr>
<tr>
<td>rsp+24</td>
<td>c-8</td>
</tr>
<tr>
<td>rsp+24</td>
<td>c-8</td>
</tr>
<tr>
<td>rsp+32</td>
<td>c-8</td>
</tr>
<tr>
<td>rsp+40</td>
<td>c-8</td>
</tr>
<tr>
<td>rsp+48</td>
<td>c-8</td>
</tr>
<tr>
<td>rsp+56</td>
<td>c-8</td>
</tr>
<tr>
<td>rsp+160</td>
<td>c-8</td>
</tr>
<tr>
<td>rsp+56</td>
<td>c-8</td>
</tr>
</tbody>
</table>

The unwinding table captures an abstract execution of the code...
<foo>:
  push  %r15
  push  %r14
  mov   $0x3,%eax
  push  %r13
  push  %r12
  push  %rbp
  push  %rbx
  sub   $0x68,%rsp
  add   $0x68,%rsp
  pop   %rbx

  CFA       ra
  rsp+8     c-8
  rsp+16    c-8
  rsp+24    c-8
  rsp+24    c-8
  rsp+32    c-8
  rsp+40    c-8
  rsp+48    c-8
  rsp+56    c-8
  rsp+160   c-8
  rsp+56    c-8

...and thus is redundant with the binary.
Upon entering a function, we know

\[ \text{CFA} = %rsp - 8 \quad \text{RA} = \text{CFA} + 8 \]

The semantics of each instruction specifies how it changes the CFA.

- Heuristic to decide whether we index with %rbp or %rsp

With a symbolic execution with an abstract semantics, we can synthesize the unwinding table line by line.

Control flow: forward data-flow analysis

The fixpoints are immediate, cf article

Implemented on top of CMU’s BAP
Demo time!
Unwinding data is slow.

So much that performance cannot unwind online! It must copy to disk the whole call stack every few instants and analyze it later at report time!
Unwinding data is slow.
Unwinding data is slooo
Unwinding data is slooooo
Unwinding data is slooooon
Unwinding data is slooooooo
Unwinding data is slooooooo
Unwinding data is sloooooooooo
Unwinding data is sloooooooow.
Unwinding data is sloooooooow.

So much that perf cannot unwind online! It must copy to disk the whole call stack every few instants and analyze it later at report time!
Unwinding data compilation
30 24 34 FDE pc=004020..004040
  DW_CFA_def_cfa_offset: 16
  DW_CFA_advance_loc: 6 to 0000000000004026
  DW_CFA_def_cfa_offset: 24
  DW_CFA_advance_loc: 10 to 0000000000004030
  DW_CFA_def_cfa_expression (DW_OP_breg7 (rsp): 8;
   DW_OP_breg16 (rip): 0; ...)

unwind_context_t _eh_elf (
  unwind_context_t ctx , uint ptr_t pc )
{
  unwind_context_t out_ctx ;
  switch ( pc ) {
    . . .
    case 0x615 . . . 0x618 :
      out_ctx . rsp = ctx . rsp + 8 ;
      out_ctx . rip = ∗ ( ( uint ptr_t ) ( out_ctx . rsp − 8) ) ;
      out_ctx . flags = 3u ;
      return out_ctx ;
    . . .
  }
}

ELF file: "eh_elf"
30 24 34 FDE pc=004020..004040
DW_CFA_def_cfa_offset: 16
DW_CFA_advance_loc: 6 to 0000000000004026
DW_CFA_def_cfa_offset: 24
DW_CFA_advance_loc: 10 to 0000000000004030
DW_CFA_def_cfa_expression (DW_OP_breg7 (rsp): 8;
   DW_OP_breg16 (rip): 0; ...)

<table>
<thead>
<tr>
<th>PC</th>
<th>CFA</th>
<th>rbx</th>
<th>rbp</th>
<th>ra</th>
</tr>
</thead>
<tbody>
<tr>
<td>0084950</td>
<td>rsp+8</td>
<td>u</td>
<td>u</td>
<td>c-8</td>
</tr>
<tr>
<td>0084952</td>
<td>rsp+16</td>
<td>u</td>
<td>u</td>
<td>c-8</td>
</tr>
<tr>
<td>0084954</td>
<td>rsp+24</td>
<td>u</td>
<td>u</td>
<td>c-8</td>
</tr>
<tr>
<td>0084956</td>
<td>rsp+32</td>
<td>u</td>
<td>u</td>
<td>c-8</td>
</tr>
</tbody>
</table>
unwind_context_t _eh_elf(unwind_context_t ctx, uintptr_t pc)
{
    unwind_context_t out_ctx;
    switch(pc) {
        ...
        case 0x615 ... 0x618:
            out_ctx.rsp = ctx.rsp + 8;
            out_ctx.rip = *(((uintptr_t*)(out_ctx.rsp - 8));
            out_ctx.flags = 3u;
            return out_ctx;
    }
    ...
}
- **libunwind**: most common library for unwinding

- **libunwind-eh ELF**: modified version to support eh_elfs

  ▶️ Same API, almost “relink-and-play” for existing projects!
Performances

Unwinding speedup vs. libunwind:
- x15 on perf gzip
- x25 on perf hackbench

Space overhead vs. DWARF:
- x2.6 – x3
What’s next?
Synthesis + compare = verification of unwinding data!

Integrate synthesis into compilers & debuggers
  → support for inline assembly, fallback method, ...

Integrate into perf for online unwinding

Probably many more cool projects!

Come and chat if interested! :}

18/18
Fixpoint upon control flow merge

If eg.

\[ CFA(A) = c-48 \quad CFA(B) = c-52 \]

no possible unwinding data for C, even for the compiler!

Also, no possible clean function postlude!

\[ \implies CFA(A) = CFA(B) \text{ and merge is immediate} \]
Variable stack frame size!

We cannot hope for a simple invariant,...
but the compiler cannot either.

⇒ the compiler will fallback to %rbp
even with --fomit-frame-pointer