C++ Exceptions and Stack Unwinding

Dave Watson

libunwind Maintainer, Facebook Engineer [cppcon 2017]

Unwinders

- **libunwind**
  https://github.com/libunwind/libunwind
- **llvm-libunwind**
  https://github.com/llvm-mirror/libunwind
- **libgcc**
  https://github.com/gcc-mirror/gcc

Agenda

- Zero cost exceptions
- C++, Itanium, & libunwind Exception API/ABI
- Unwind info – dwarf, elf, and the OS

Itanium?

Not just an architecture

Widely used exception handling ABI:

https://itanium-cxx-abi.github.io/cxx-abi/abi-eh.html

Zero-cost exceptions

Return codes

C++

```
int foo() {
  int res = bar();
  if (res < 0) {
    return res;
  }

  //...

  return 0;
}
```

Return codes

X86_64

```
foo:
        subq    $8, %rsp
        call    bar
        movl    $0, %edx
        testl   %eax, %eax
        cmovg   %edx, %eax
        addq    $8, %rsp
        ret
```

Zero-cost Exceptions

C++

```cpp
void foo() {
  try {
    bar();
  } catch (...) {}
}
```

Zero-cost Exceptions

X86_64

```
foo:
  subq    $8, %rsp
  call    bar
  addq    $8, %rsp
  ret
```

Zero-cost Exceptions

X86_64

```
foo:
  subq    $8, %rsp
  call    bar
  addq    $8, %rsp
  ret
```

Where did the catch block go?

Zero-cost Exceptions

X86_64

Landing pad

```
foo:
  subq    $8, %rsp      movq    %rax, %rdi
  call    bar           call    ___cxa_begin_catch
  addq    $8, %rsp      popq    %rax
  ret                   jmp     ___cxa_end_catch
```

Where did the catch block go?

Zero-cost Exceptions

C++

```cpp
void foo() {
  lock_guard<std::mutex> g(m);
  try {
    bar();
  } catch (Exception1 e) {
  } catch (…) {}
}
```

Zero-cost Exceptions

C++

```cpp
void foo() {
  throw 1;
}
```

Zero-cost Exceptions

X86_64

```
call    ___cxa_allocate_exception
movq    __ZTIi@GOTPCREL(%rip), %rsi
xorl    %edx, %edx
movl    $1, (%rax)
movq    %rax, %rdi
call    ___cxa_throw
```

C++, Itanium, & libunwind API/ABI

C/C++ ABI

Exception stack management: __cxa_begin_catch   __cxa_end_catch

C/C++ ABI

Throwing & allocating: __cxa_allocate_exception __cxa_throw

- Allocation: Unwinding the stack – need to allocate on heap
- Allocation: std::bad_alloc?

- cxa_throw -> landing pad?

C/C++ ABI

Throwing & allocating: __cxa_allocate_exception   __cxa_throw

- Allocation: Unwinding the stack – need to allocate on heap
- Allocation: std::bad_alloc?
    - Emergency memory pools
- cxa_throw -> landing pad?

Itanium ABI

Used on arm, x86 / 64, ppc, mips, ia64, aarch64, others

- Unwind_RaiseException (called from __cxa_throw)

- 'Personality Routine' - __gxx_personality_v0
  - Unwind_Get/SetIP
  - Unwind_Get/SetGP (general purpose register)

libunwind API

libunwind & llvm-libunwind (but not libgcc)

- unw_init_local
- unw_step
- unw_resume
- unw_get_reg
- unw_get_proc_info

Two-Phase unwinding

1: Search Phase

```
context = unw_init_local()
while(true) {
    if (!unw_step(context)) {
      // Cleanup, call terminate()
    }
    personality = unw_get_proc_info(context)
    if (HANDLER_FOUND ==
              personality(context, SEARCH)) {
      break;
    }
  }
```

Two-Phase unwinding

2: Unwind phase

```
context = unw_init_local()

while(true) {
    unw_step(context);
    personality = unw_get_proc_info(context)
    if (INSTALL_CONTEXT ==
                personality(context, CLEANUP)) {
      unw_resume(context);
    }
  }
```

Two-Phase unwinding

What does it buy us?

- Stack is still valid when we find catch block. Can resume execution at throw statement, as in common lisp
- C++ - terminate() has full access to the stack - for better error messages

noexcept & terminate()

A practical example

```
void bar() {
  throw 1;
}

int main() {
  auto t = std::thread([]() {
      bar();
    });
  t.join();
}
```

noexcept & terminate()

Without noexcept

```
(gdb) bt
…
#2  in __gnu_cxx::__verbose_terminate_handler() () fro
/lib64/libstdc++.so.6
#3  in ?? () from /lib64/libstdc++.so.6
#4  in std::terminate() () from /lib64/libstdc++.so.6
#5  in ?? () from /lib64/libstdc++.so.6
#6  in start_thread () from /lib64/libpthread.so.0
#7  in clone () from /lib64/libc.so.6
```

noexcept & terminate()

``noexcept will not call std::unexpected and may or may not unwind the stack, which potentially allows the compiler to implement noexcept without the runtime overhead of throw()"

- std::thread has a try/catch around it (up until gcc~8)

noexcept & terminate()

A practical example

```
void bar() {
  throw 1;
}

int main() {
  auto t = std::thread([]() noexcept {
      bar();
    });
  t.join();
}
```

noexcept & terminate()

With noexcept

```
…
#2  in __gnu_cxx::__verbose_terminate_handler() () fro
…
#5  in __gxx_personality_v0 () from /lib64/libstdc++.s
#6  in ?? () from /lib64/libgcc_s.so.1
#7  in _Unwind_RaiseException () from /lib64/libgcc_s.
#8  in __cxa_throw () from /lib64/libstdc++.so.6
#9  in bar () at unwind.cpp:4
…
16 frames
```

Non-exception stack unwinding

- POSIX - **Setjmp & longjmp** – still want to RAII and destroy stack objects, so uses same machinery. Instead of personality, call stop function
- Gnu C - **backtrace**() – don’t need to unwind all registers, only return address and instruction pointer

Unwind info – dwarf, elf, and the OS

Zero-cost Exceptions

Dwarf .eh_frame unwind info

```
readelf --debug-dump=frames binary_name
```

```
DW_CFA_advance_loc: 1 to 4005a4
DW_CFA_def_cfa_offset: 16
DW_CFA_offset: r6 (rbp) at cfa-16
DW_CFA_advance_loc: 3 to 4005a7
DW_CFA_def_cfa_register: r6 (rbp)
DW_CFA_advance_loc: 11 to 4005b2
DW_CFA_def_cfa: r7 (rsp) ofs 8
```

Zero-cost Exceptions

Dwarf .eh_frame unwind info

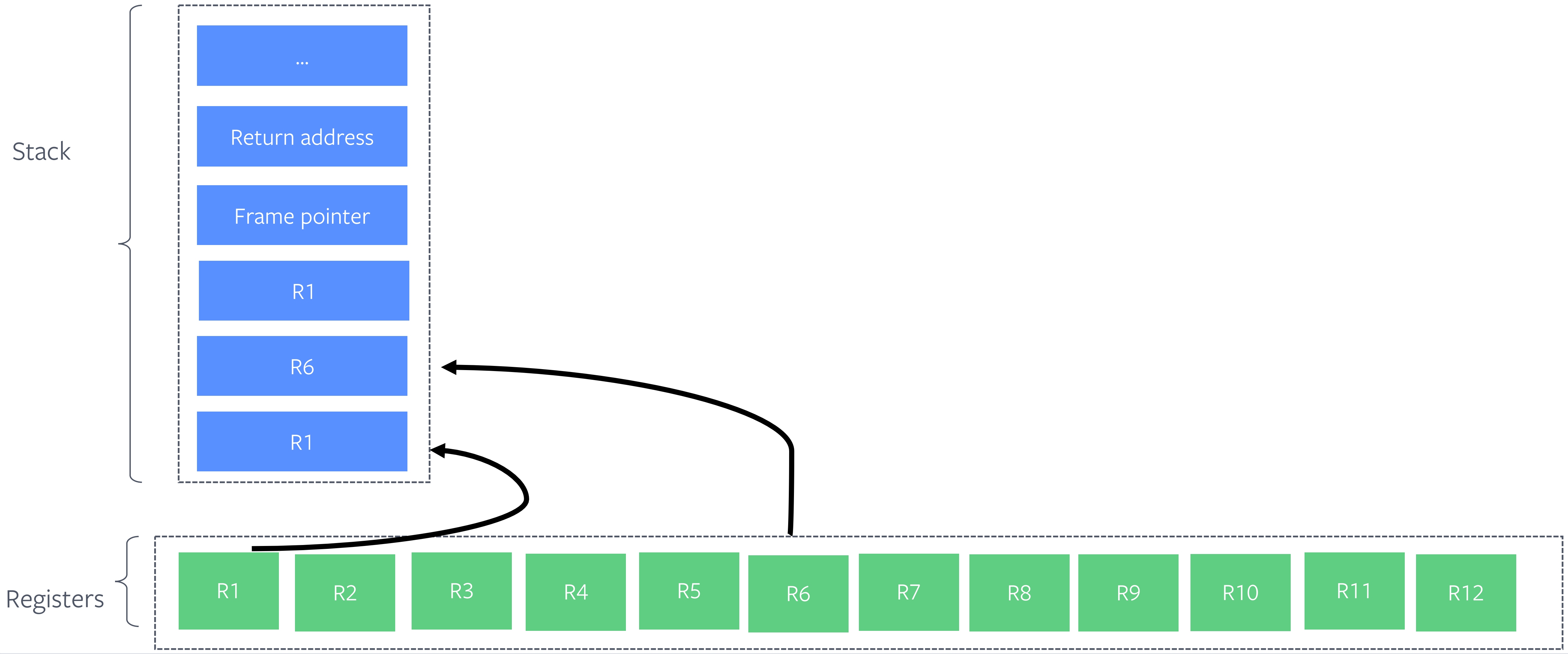
```
readelf --debug-dump=frames binary_name
```

```
DW_CFA_advance_loc: 1 to 4005a4
DW_CFA_def_cfa_offset: 16
DW_CFA_offset: r6 (rbp) at cfa-16
DW_CFA_advance_loc: 3 to 4005a7
DW_CFA_def_cfa_register: r6 (rbp)
DW_CFA_advance_loc: 11 to 4005b2
DW_CFA_def_cfa: r7 (rsp) ofs 8
```

```
foo:
    push   %rbp
    mov    %rsp, %rbp
    callq  _bar
    mov    $0x0, %eax
    pop    %rbp
    retq
```

Unwinding the stack with Dwarf

Finding where registers are saved



Which registers need restoring?

- Caller-saved
- Callee-saved
- Frame-related (Instruction, Frame, Stack Pointers)

Which registers need restoring?

- ~~Caller-saved~~
- **Callee-saved**
- **Frame-related** (Instruction, Frame, Stack Pointers)

Chasing bugs

Dwarf .eh_frame unwind info

```
void b() {
  char foo[1] __attribute((aligned(32)));
}
```

Chasing bugs

Dwarf .eh_frame unwind info

```
void b() {
  char foo[1] __attribute((aligned(32)));
}
```

```
_b:
  lea    0x8(%rsp), %r10
  and    $0xffffffffffffffe0, %rsp
  pushq  -0x8(%r10)
  push   %rbp
  mov    %rsp, %rbp
  push   %r10
  pop    %r10
  pop    %rbp
  lea    -0x8(%r10), %rsp
  retq
```

Chasing bugs

Dwarf .eh_frame unwind info

```
  DW_CFA_advance_loc: 5 to 5
  DW_CFA_def_cfa: r10 (r10) ofs 0
  DW_CFA_advance_loc: 9 to e
  DW_CFA_expression: r6 (rbp)
(DW_OP_breg6 (rbp): 0)
  DW_CFA_advance_loc: 5 to 13
  DW_CFA_def_cfa_expression
(DW_OP_breg6 (rbp): -8; DW_OP_deref)
  DW_CFA_advance_loc: 2 to 15
  DW_CFA_def_cfa: r10 (r10) ofs 0
  DW_CFA_advance_loc: 5 to 1a
  DW_CFA_def_cfa: r7 (rsp) ofs 8
```

```
_b:
  lea    0x8(%rsp), %r10
  and    $0xffffffffffffffe0, %rsp
  pushq  -0x8(%r10)
  push   %rbp
  mov    %rsp, %rbp
  push   %r10
  pop    %r10
  pop    %rbp
  lea    -0x8(%r10), %rsp
  retq
```

Expressions

Dwarf .eh_frame unwind info – Turing Complete

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)
```

How do we find dwarf unwind info?

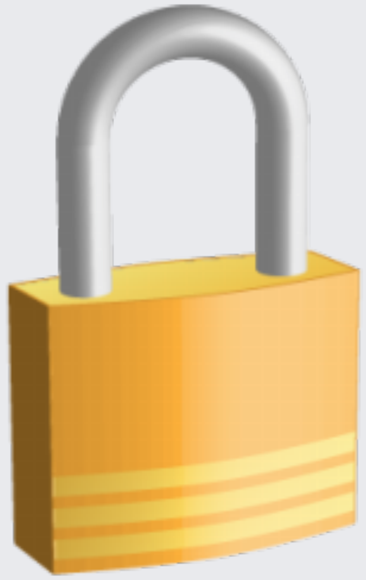
Elf format

- Elf format: .eh_frame section
- frame sections could be split across multiple libraries

How do we find dwarf unwind info?

DLOPEN & DLCLOSE

- glibc - dl_iterate_phdr - iterate all loaded libraries using runtime loader



- What happens if we try to read dwarf info from file that is unmapped in a different thread?

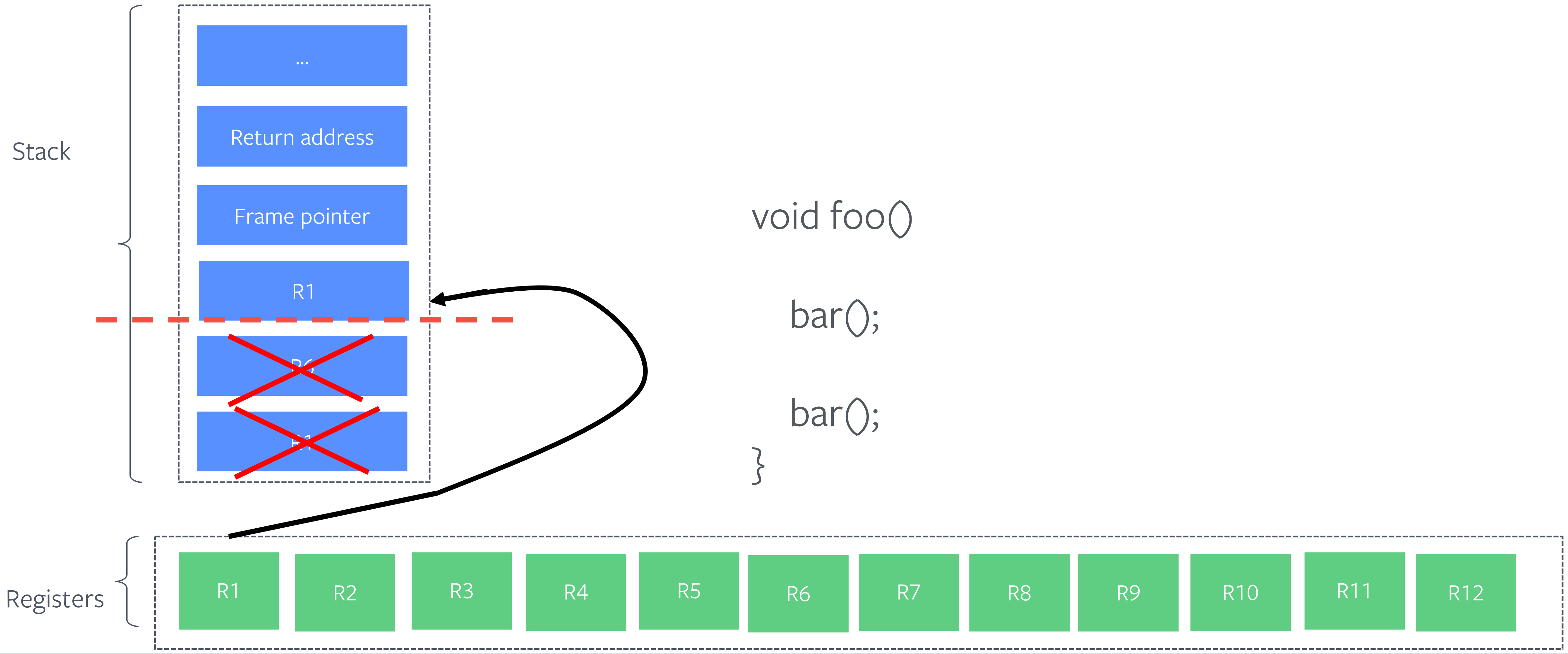
Caching results of dl_iterate_phdr

dlopen, dlclose, dl_iterate_phdr all take lock

- libgcc - takes lock briefly and checks version counter. Caches 8 object file headers
- llvm-libunwind - caches unlimited frames using rw_lock
- libunwind – resume takes lock, unwinding optimistically uses cache

Unwinding the stack with Dwarf

Unwinding to a specific Instruction with Caching



```
void foo()

    bar();

    bar();
}
```

How do we find dwarf unwind info?

Caching dwarf info

- libunwind - caches dwarf *results* directly. Each unw_step is a hash table lookup, and apply.
- Sensitive to hashtable size, each IP needs an entry.

Fast stack traces

Backtrace(), as used by jemalloc, tcmalloc, etc

- need less info – most registers not needed
- libunwind supports by packing unwind info to single 64bit
- Some arches (arm) pack all unwind data in the same way

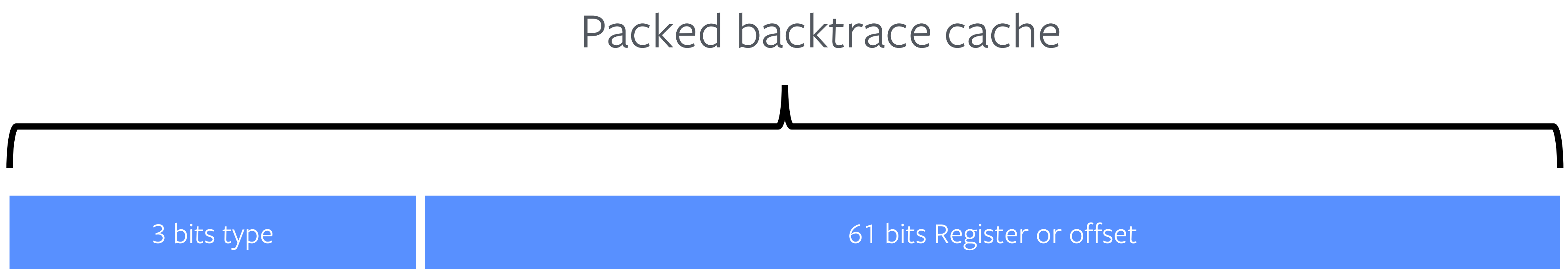
If fails, falls back to normal unw_step()

Fast stack traces

Frame types:

- Frame pointer – follow rbp
- Sigreturn – get registers from ucontext_t
- Aligned – Dereference a register

Otherwise fall back to dwarf cache / processing

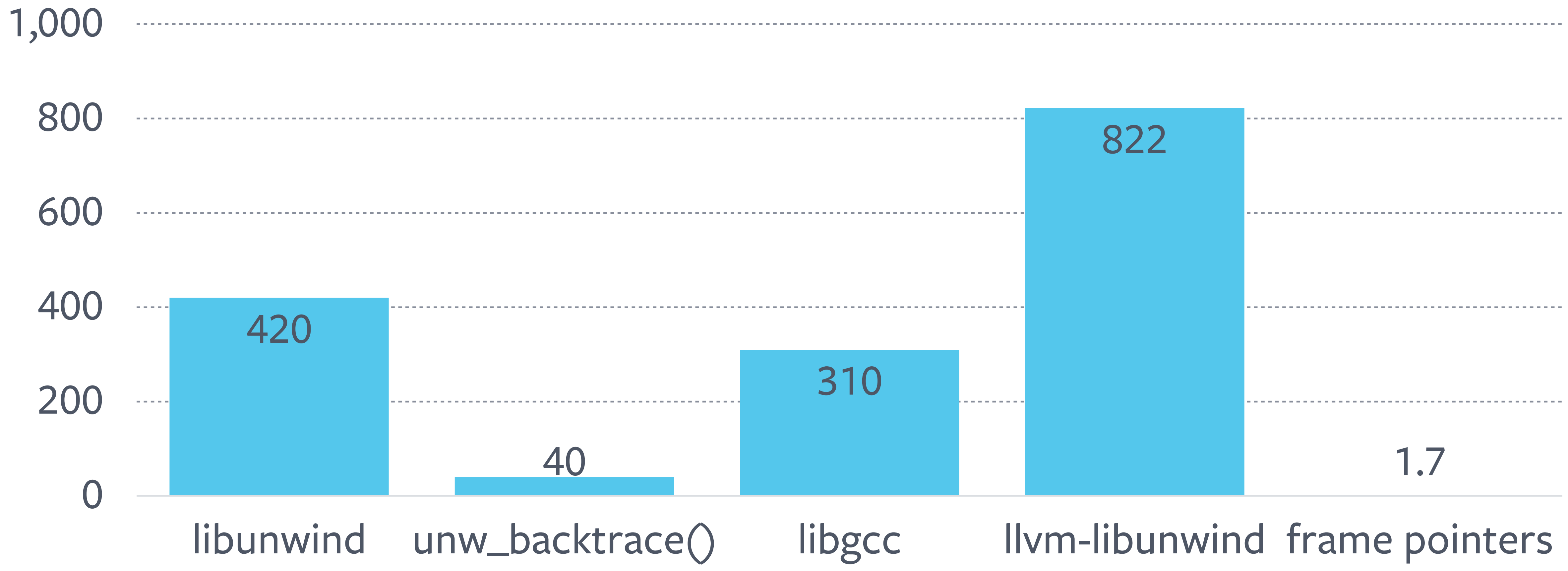


Fixing Performance

1) unw_backtrace -> learn about dwarf expressions
2) Dwarf cache too small
3) Global lock contention in dl_iterate_phdr

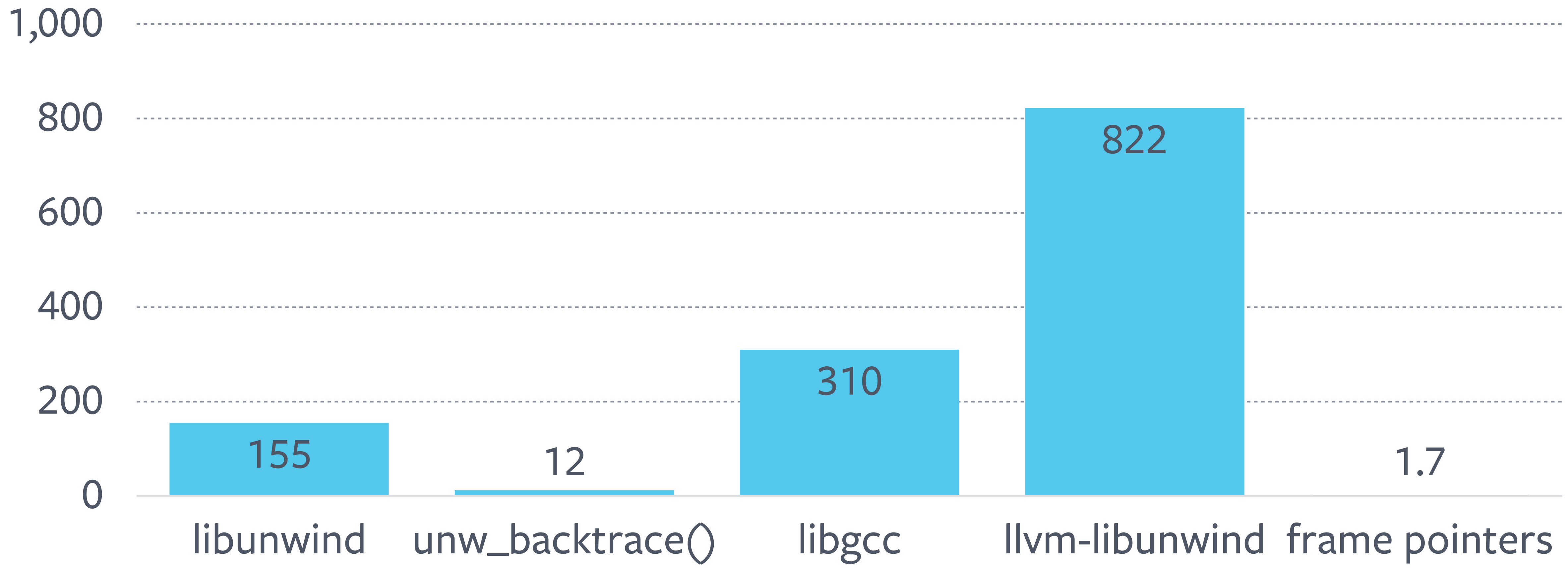
Fast stack traces

backtrace() time in ms



Fast stack traces

backtrace() time in ms, using ./configure –disable-block-signals



When to use exceptions?

Exceptional case needs to be

- Several orders of magnitude less frequent if in unwind cache
- Even more if not

facebook

Questions