C++ Exceptions for Smaller Firmware

By: Khalil Estell (@kammce)

@kammce
Graduated 2017 from SJSU
Ex-Googler (~6y)
Volunteer @ SJSU
ISO C++ Committee Member

SAN JOSÉ STATE UNIVERSITY
1857

SJSU ROBOTICS

Motivation

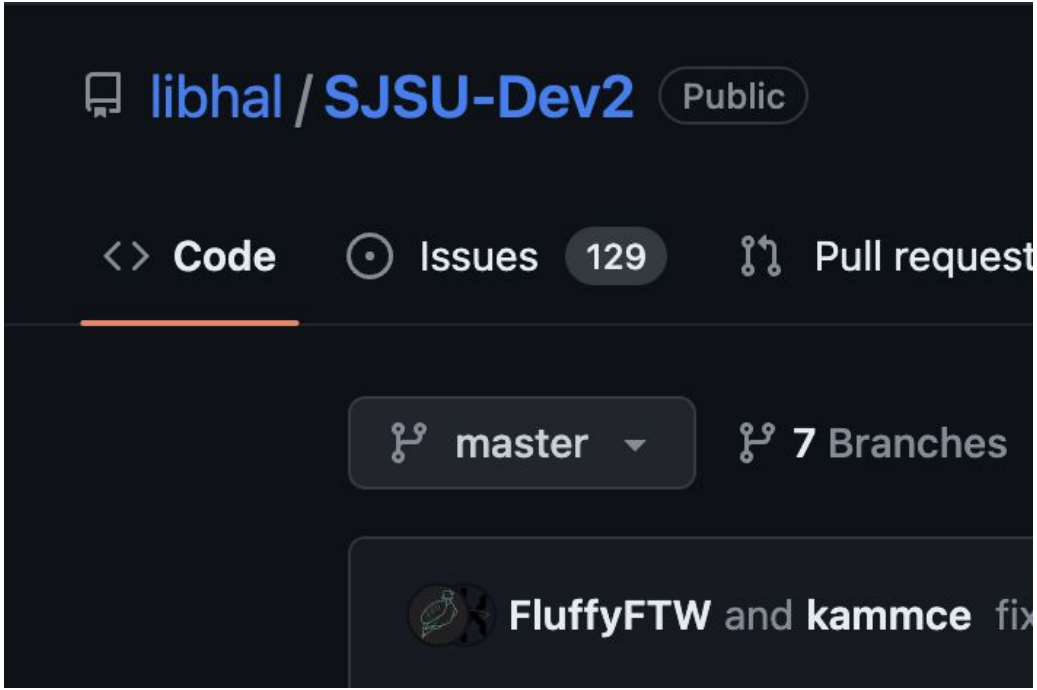

I²C Master 1
SDA
SCL
I²C Slave 1
1:32:10
SJSU CMPE 146 I2C Lecture 10-03-17
283 views • 6 years ago

C++ Core Guidelines

February 15, 2024

Editors:

- Bjarne Stroustrup
- Herb Sutter



Microcontroller Constraints: No Operating System

~~fopen()~~

~~networking~~

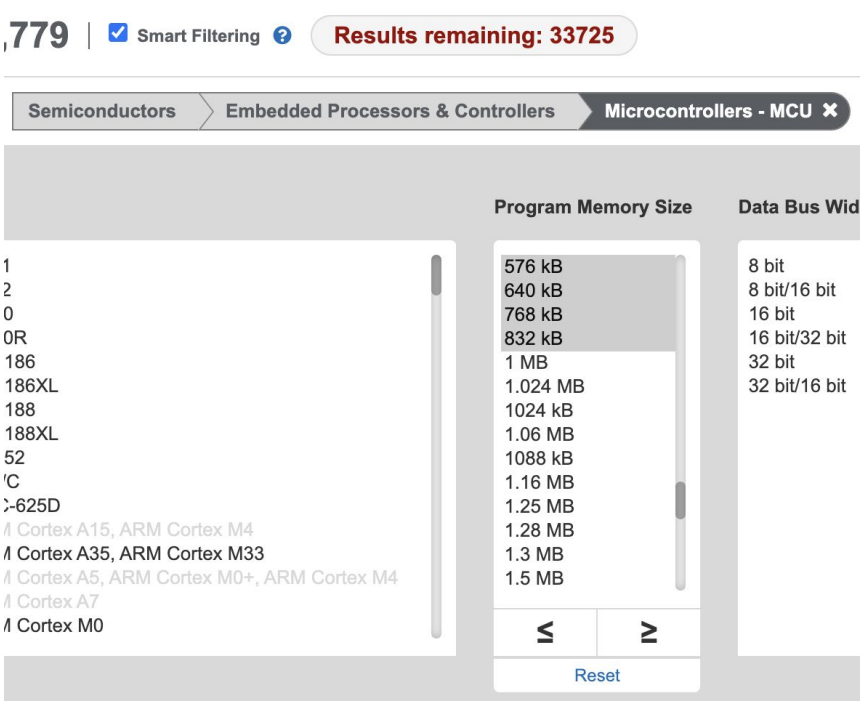
~~std::print()~~

~~std::chrono::steady_clock::now();~~

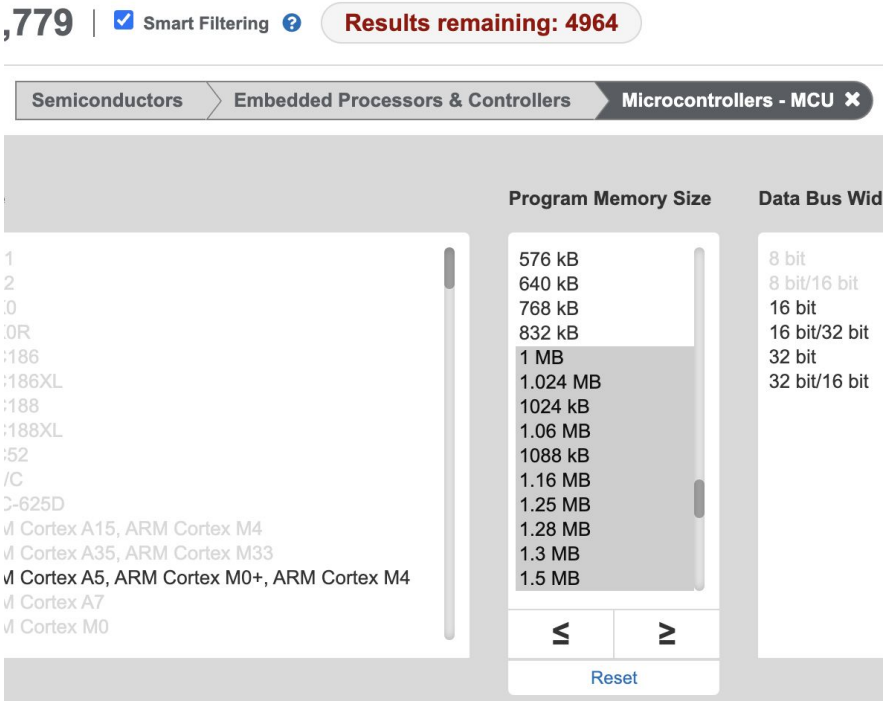
~~std::thread~~

Microcontroller Constraints: Limited Storage & Memory

< 1 MB = 33,725 devices

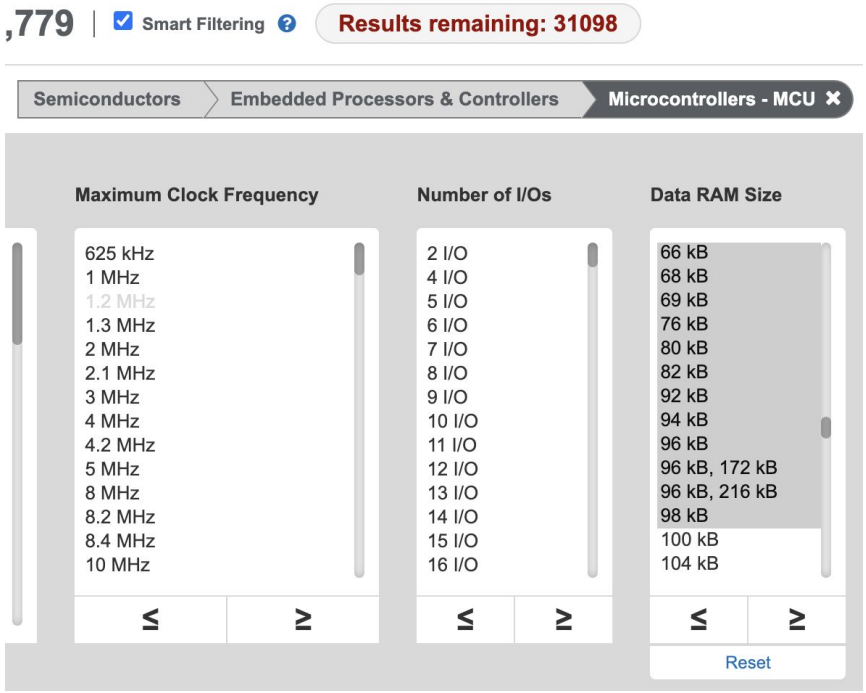


>= 1 MB = 4,964 devices

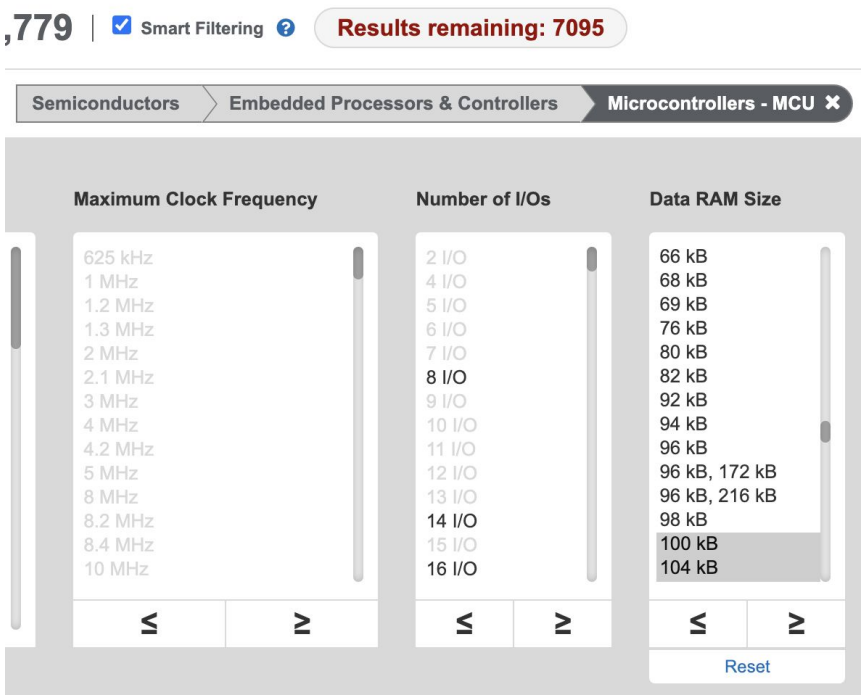


Microcontroller Constraints: Limited Storage & Memory

< 100kB = 31,098 devices



>= 100kB = 7095



With so little memory how do you prevent
running out of memory
or
memory fragmentation?

Typical Best Practice: Do not/avoid dynamic allocations

~~new~~

~~calloc~~

~~free~~

~~malloc~~

~~delete~~

~~realloc~~

Typical Best Practice: Do not/avoid dynamic allocations

~~new~~

~~std::string~~

~~calloc~~

~~std::unique_ptr~~

~~free~~

~~std::vector~~

~~malloc~~

~~delete~~

~~std::unordered_map~~

~~realloc~~

~~std::shared_ptr~~

Designed and maintained by John Wellbelove

Embedded Template Library

A C++ template library for embedded applications

MIT licensed

Slack

Github

Arduino

Is the ETL free?

Home | Overview | Documentation | Tutorials | Links | About | Contact

release date december 2023 | c++ 98/03/11/14/17 | license MIT | contributors 122

msvc passing | gcc passing | clang passing | build passing | code quality A

Arctic Code Vault

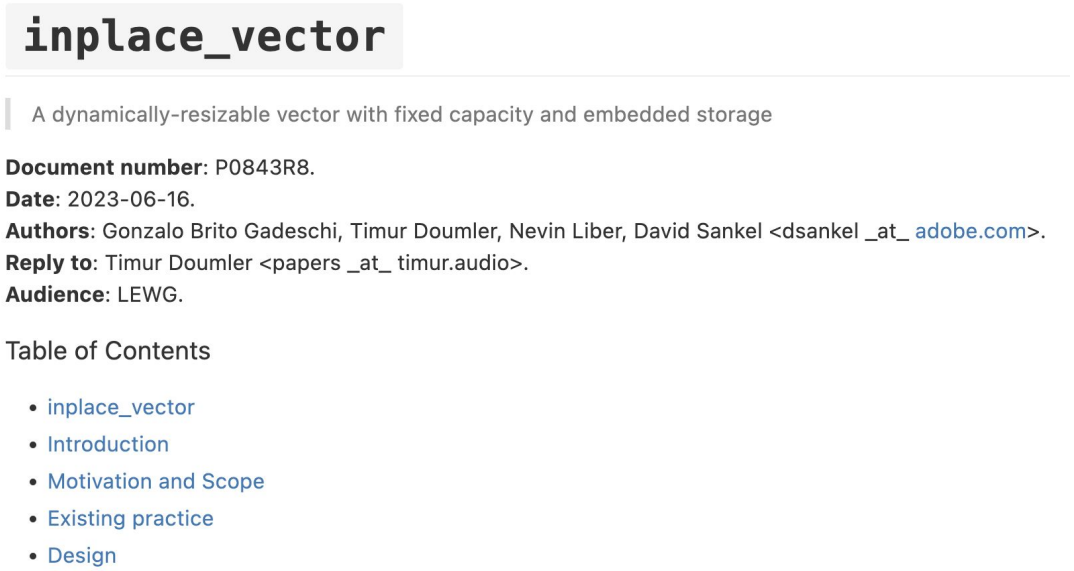
Motivation

C++ is a great language to use for embedded applications and templates are a powerful aspect of it. The standard library can offer a great deal of well tested functionality, but there are some parts that do not fit well with deterministic behaviour and limited resource requirements. These limitations usually preclude the use of dynamically allocated memory which means that the STL containers are unusable.

What is needed is a template library where the user can declare the size, or maximum size of any object upfront. As most embedded compilers do not currently support the standard beyond C++ 03, it would also be nice to have access to some of the features introduced in the later library.

Lets not forget about

- `std::inplace_vector`: p0843r8 voted into C++26 @ St Louis 2024 ISO C++ meeting



inplace_vector

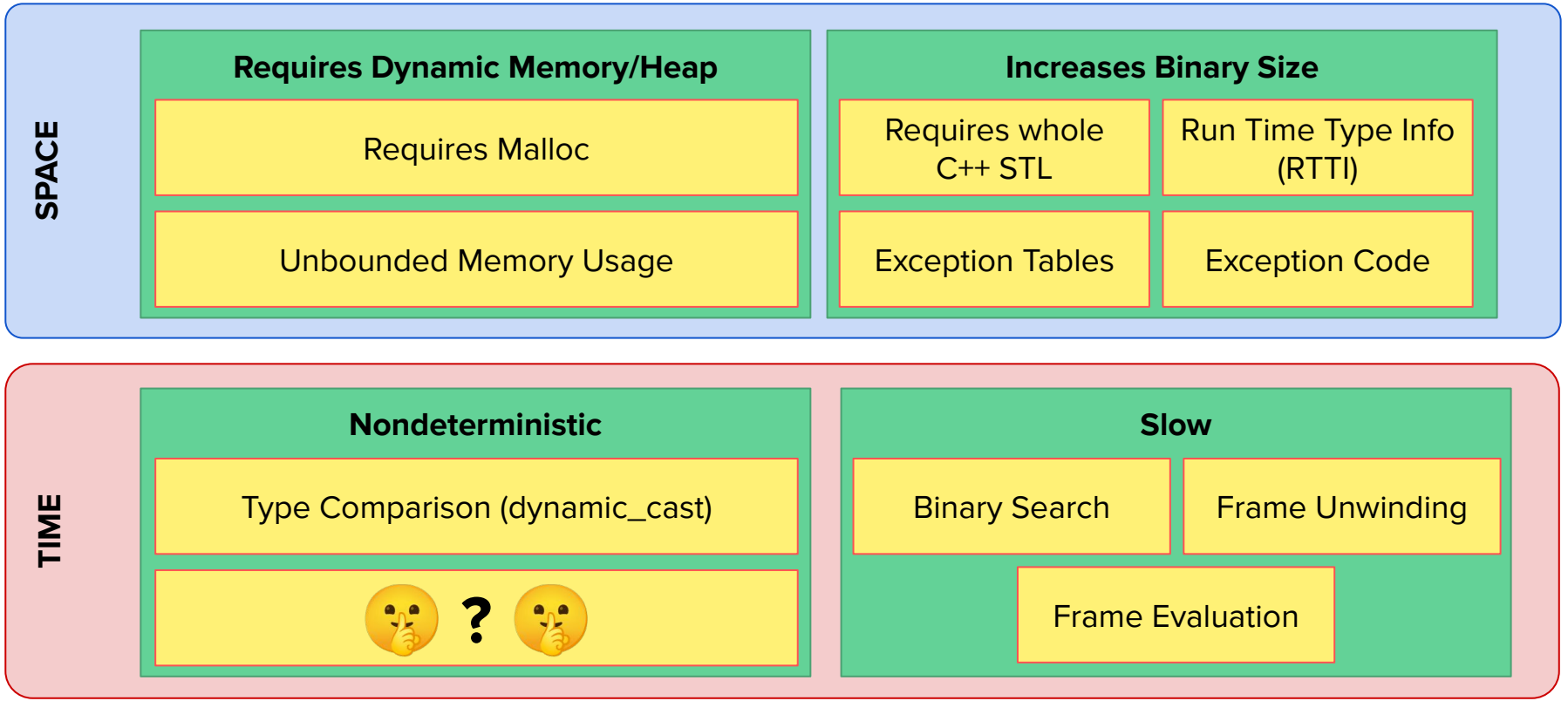
> A dynamically-resizable vector with fixed capacity and embedded storage

Document number: P0843R8.
Date: 2023-06-16.
Authors: Gonzalo Brito Gadeschi, Timur Doumler, Nevin Liber, David Sankel <dsankel _at_ adobe.com>.
Reply to: Timur Doumler <papers _at_ timur.audio>.
Audience: LEWG.

Table of Contents

- inplace_vector
- Introduction
- Motivation and Scope
- Existing practice
- Design

So why do firmware developers avoid C++ exceptions?



Back to Class!

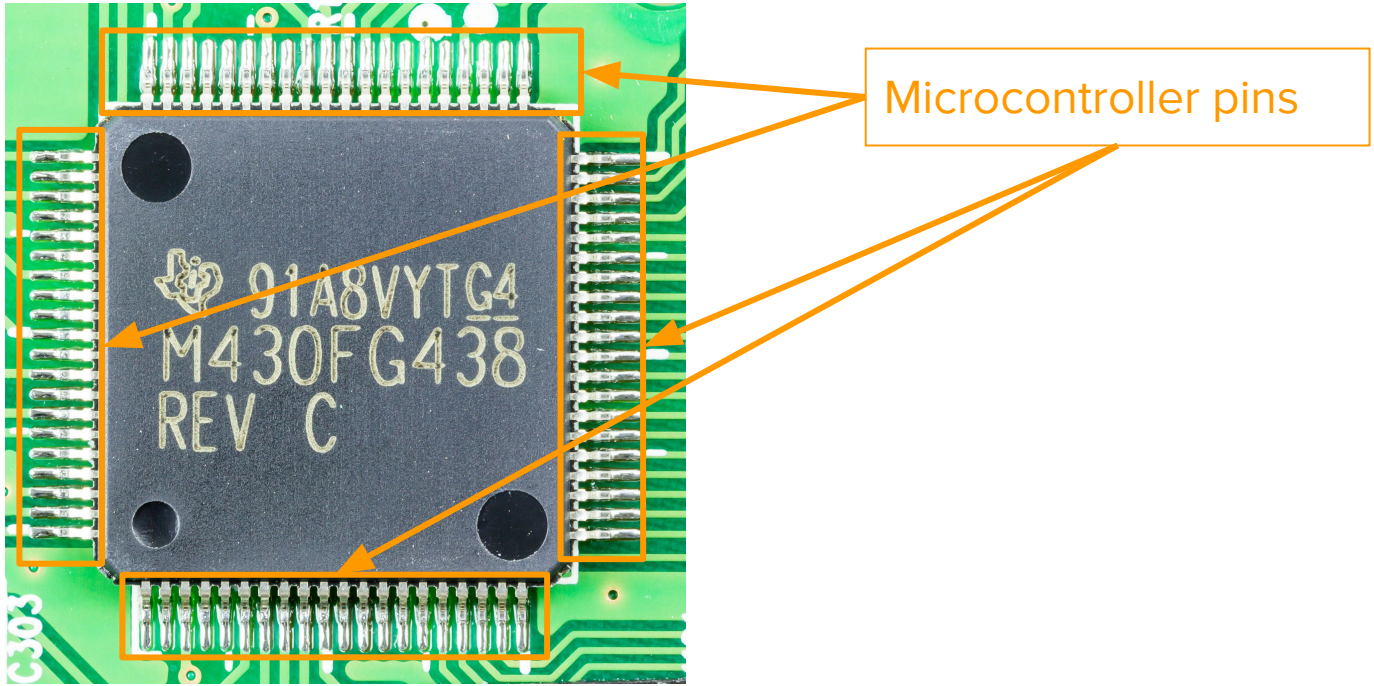
← Files | master | SJSU-Dev2 / library / peripherals / lpc40xx / | ↑ Top

gpio.hpp	Reorganize SJSU-Dev2 Library structure	3 years ago
i2c.hpp	fixed lpx40xx i2c pin functions and pin	last year
pin.hpp	Reorganize SJSU-Dev2 Library structure	3 years ago
pulse_capture.hpp	Reorganize SJSU-Dev2 Library structure	3 years ago
pwm.hpp	Reorganize SJSU-Dev2 Library structure	3 years ago
spi.hpp	Reorganize SJSU-Dev2 Library structure	3 years ago
system_controller.hpp	Fix Issue with LPC40xx UART Baud Rat...	3 years ago
timer.hpp	Reorganize SJSU-Dev2 Library structure	3 years ago
uart.hpp	Add IsConnected implementation to ES...	2 years ago

Consider the output pin

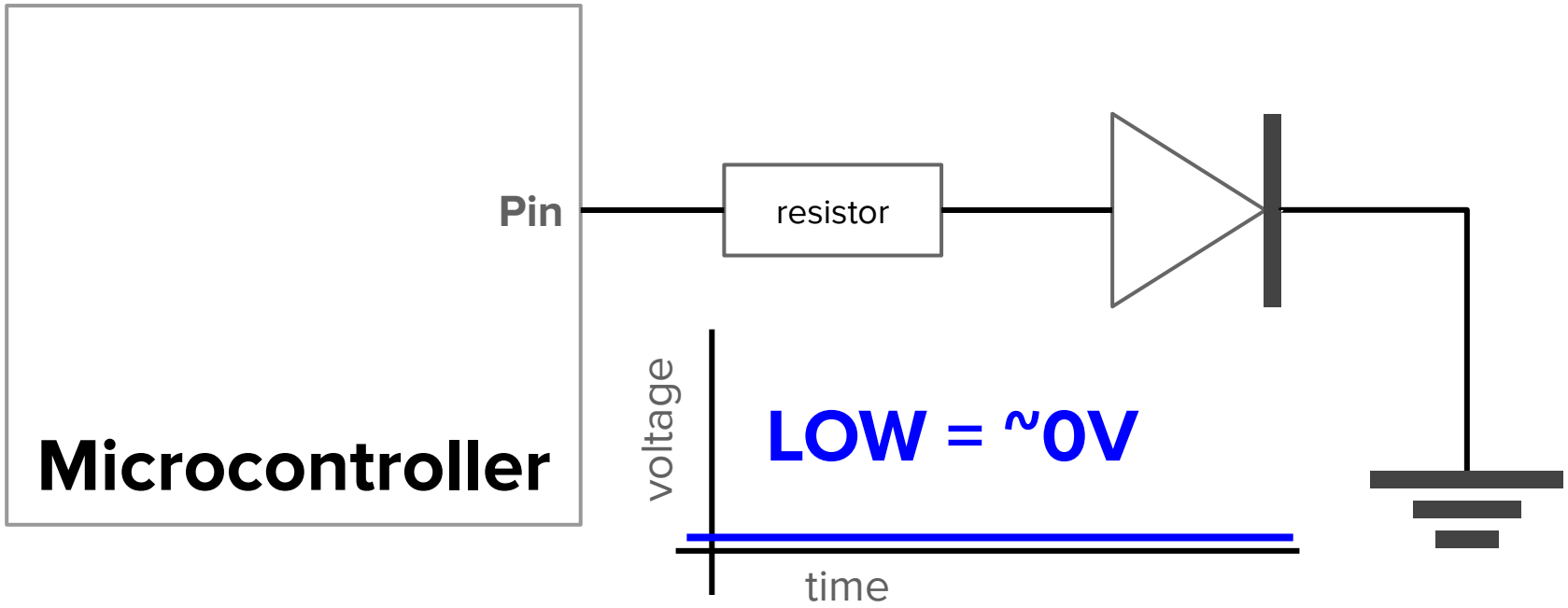
Microcontroller

Consider the output pin

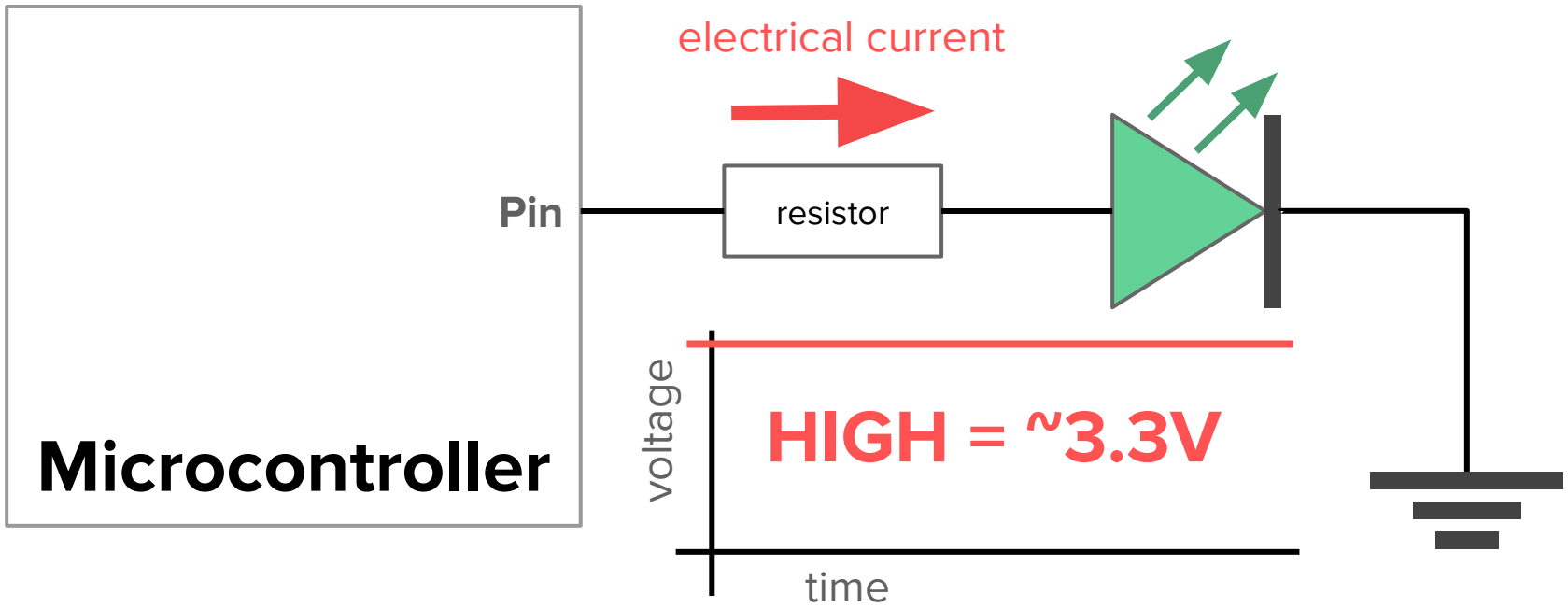


Microcontroller

Consider the output pin

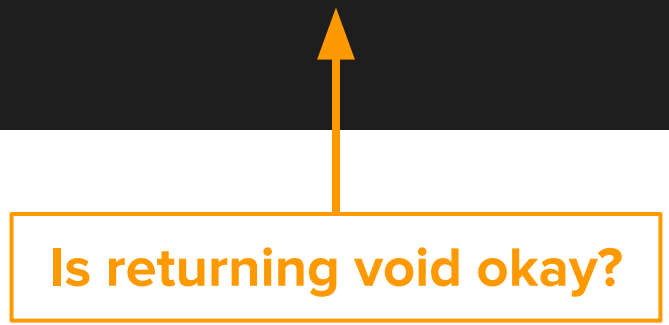


Consider the output pin

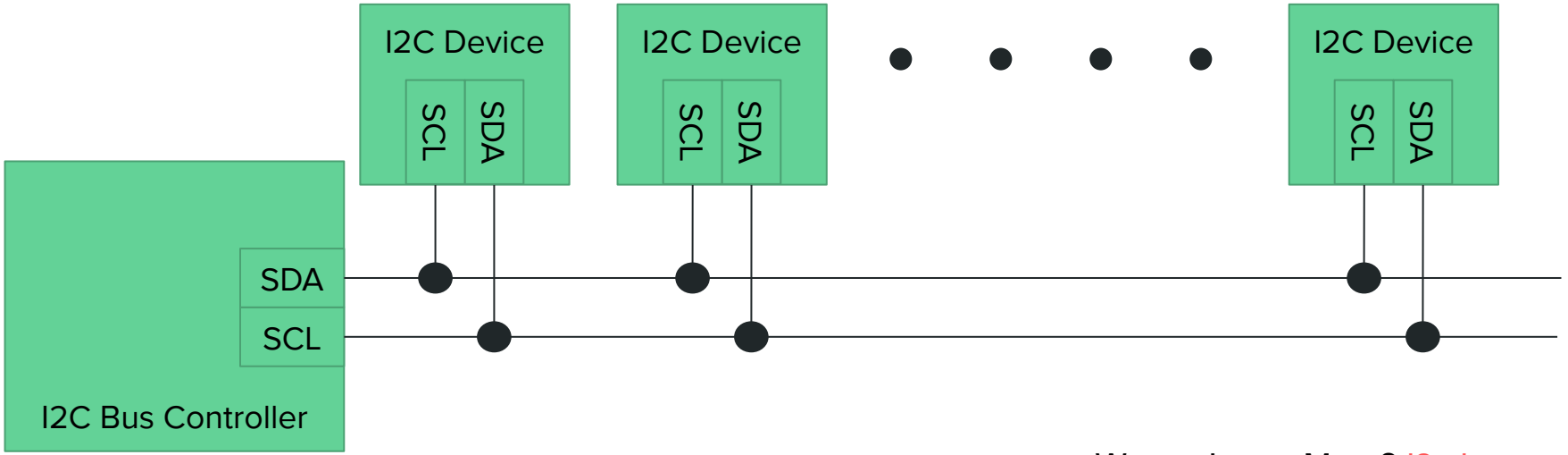


```
class output_pin
{
public:
 virtual ~output_pin() = default;
 virtual void level(bool p_high) = 0;
};
```

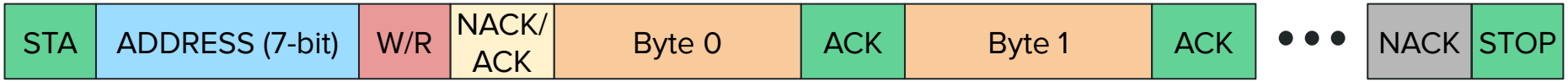
```
class output_pin
{
public:
 virtual ~output_pin() = default;
 virtual void level(bool p_high) = 0;
};
```



A simplified Introduction to I2C



Wanna Learn More? i2c-bus.org



Possible I2C Errors

ADDR NACK

Device Not Present

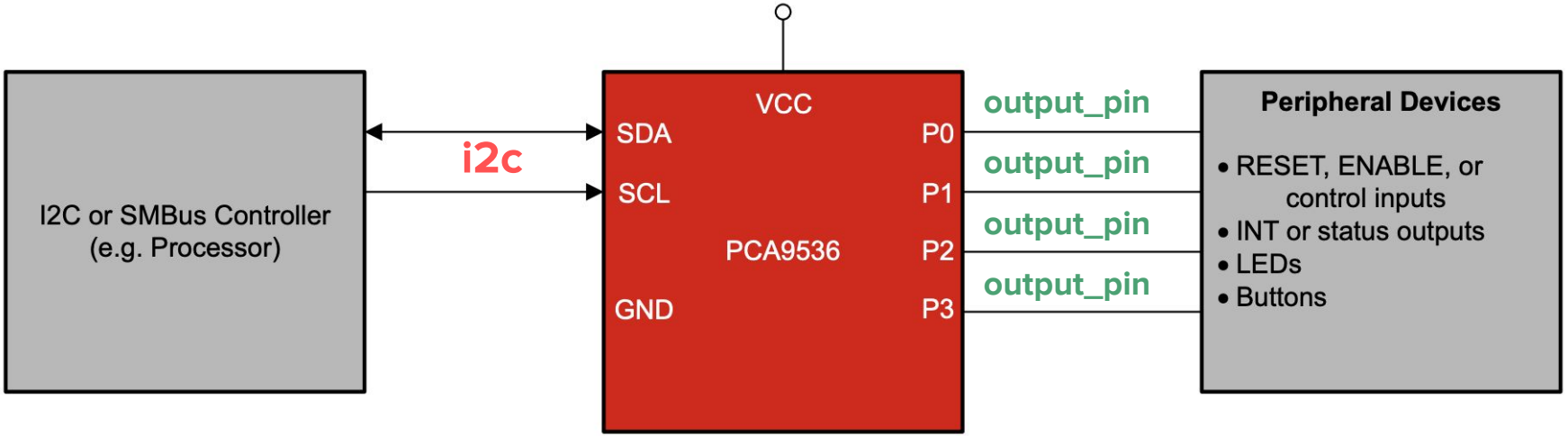
IO ERROR

Unexpected Bus State

STA	ADDRESS (7-bit)	W/R	NACK/ ACK	Byte 0	ACK	Byte 1	ACK	• • •	NACK	STOP

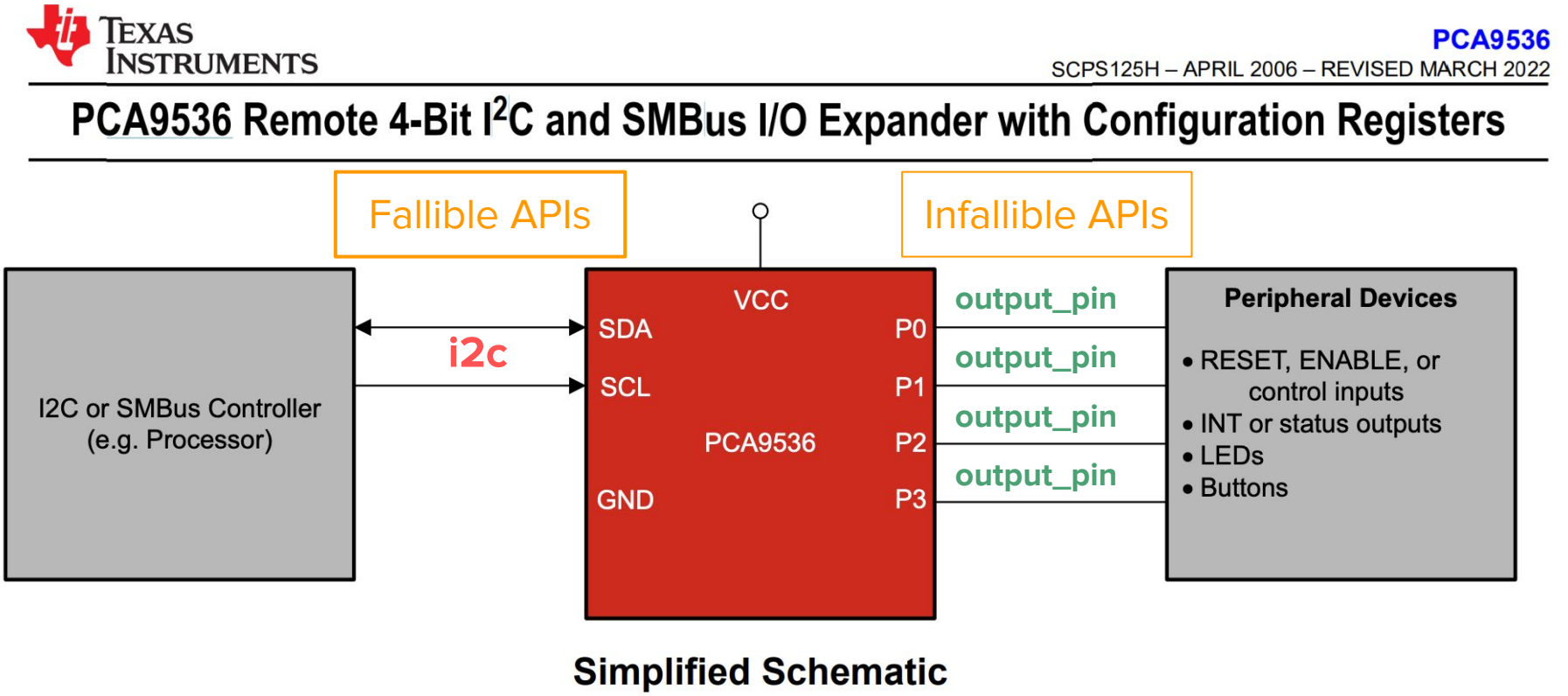
Let's consider an I2C I/O expander

PCA9536 Remote 4-Bit I²C and SMBus I/O Expander with Configuration Registers



Simplified Schematic

Let's consider an I2C I/O expander



Illustrating the problem: Return mismatch

```
/// @returns false if this failed!
bool io_expander_level(std::uint8_t p_pin_number, bool p_high_voltage) {
  // Implementation details...
}
void some_output_pin::level(bool p_high_voltage) {
  auto success = io_expander_level(m_pin, p_high_voltage);
  if (not success) {
    // What do you do here ???
  }
}
```

Consider how output pin may be used?

```
class led {
   led(output_pin& p_pin);
};
```

🟡 not severe

```
class heating_element {
   heating_element(output_pin& p_control);
};
```

‼️ severe 🚨

```
class emergency_relay_cut_off {
   emergency_relay_cut_off(output_pin&
p_control);
};
```

```
class output_pin
{
public:
 virtual ~output_pin() = default;
 virtual status level(bool p_high) = 0;
 virtual result<bool> level() = 0;
};
```

std::expected<T, E>

```
#include <expected>
enum class my_error {
  precondition_violated, invalid_arguments, // etc ...
};
std::expected<int, my_error> get_version() {
  if (device_disconnected) {
      return std::unexpected(my_error::precondition_violated);
  }
  return 15;
}
```

The code if we used **exceptions**...

```
void toggle_led(output_pin& p_pin,
                milliseconds p_transition_time) {
   p_pin.level(true);
   delay(p_transition_time / 2);
   p_pin.level(false);
   delay(p_transition_time / 2);
}
```

The code if we used **std::expected**

Notice the return type change?

```
status toggle_led(
    output_pin& p_pin,
    milliseconds p_transition_time) {
  if (auto status = p_pin.level(true);
      !status) {
      return std::unexpected(status.error());
  }
  delay(p_transition_time / 2);
  if (auto status = p_pin.level(false);
      !status) {
      return std::unexpected(status.error());
  }
  delay(p_transition_time / 2);
}
```

The code if we used **Error Propagator P2561R1 + std::expected**...

```
status toggle_led(
    output_pin& p_pin,
    milliseconds p_transition_time) {
  p_pin.level(true)??;
  delay(p_transition_time / 2);
  p_pin.level(false)??
  delay(p_transition_time / 2);
}
```

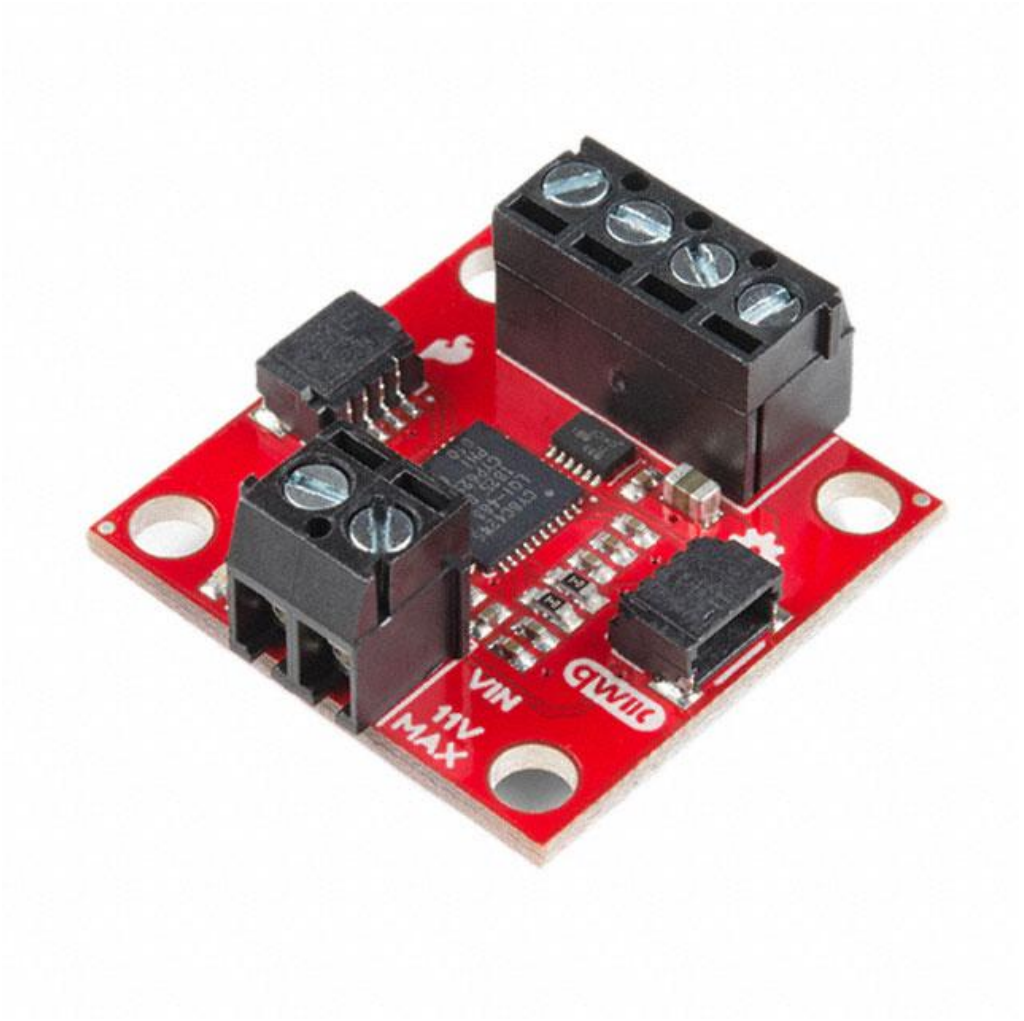
Return type is still `status`

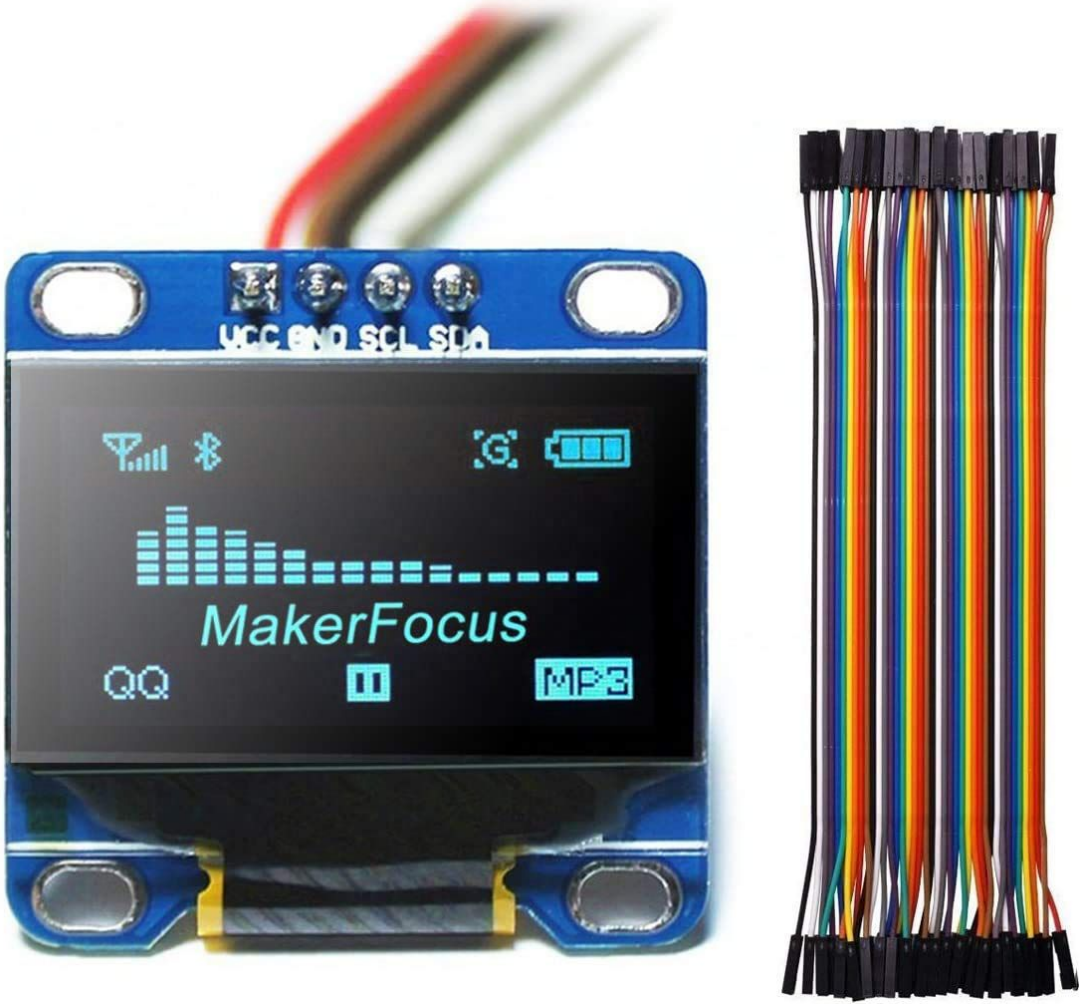
Settled on **std::expected** + Error Propagator Macro

```
status toggle_led(output_pin& p_pin,
                  milliseconds p_delay) {
   SJ_CHECK(p_pin.level(true));
   delay(p_delay / 2);
   SJ_CHECK(p_pin.level(false));
   delay(p_delay / 2);
   return success();
}
```

Which APIs need to return errors?

Almost all of them


VIN
11V
MAX


VCC GND SCL SDA
MakerFocus
QQ
MP3

So I moved **almost** all APIs to use **std::expected**...

After **2.5 years**,
what did the students think?

IF/ELSE everywhere! Just (void) cast away the problems!

```
void blink_led(
    output_pin& p_pin,
    milliseconds p_transition_time) {
    (void)p_pin.level(true);
    delay(p_transition_time / 2);
    (void)p_pin.level(false);
    delay(p_transition_time / 2);
}
```

Annoyed by the virality of error return types

```
class motion_controller {
public:
 motion_controller create(/* ... */);
 status rotate(degrees_t p_degrees);
 status move_to(coord_t p_x, coord_t p_y, coord_t p_z);
 status configure_pid(std::uint16_t p_p,
                      std::uint16_t p_i,
                      std::uint16_t p_d);
 private:
 status write_command(std::uint8_t p_register_address,
                      std::uint32_t p_payload);
 result<status_t> read_status(std::uint8_t
p_register_address);
 status wait_for_eeprom_to_finish_loading ();
  /* ... */
}
```

Everything returns status

MACROs were OK, but noisy

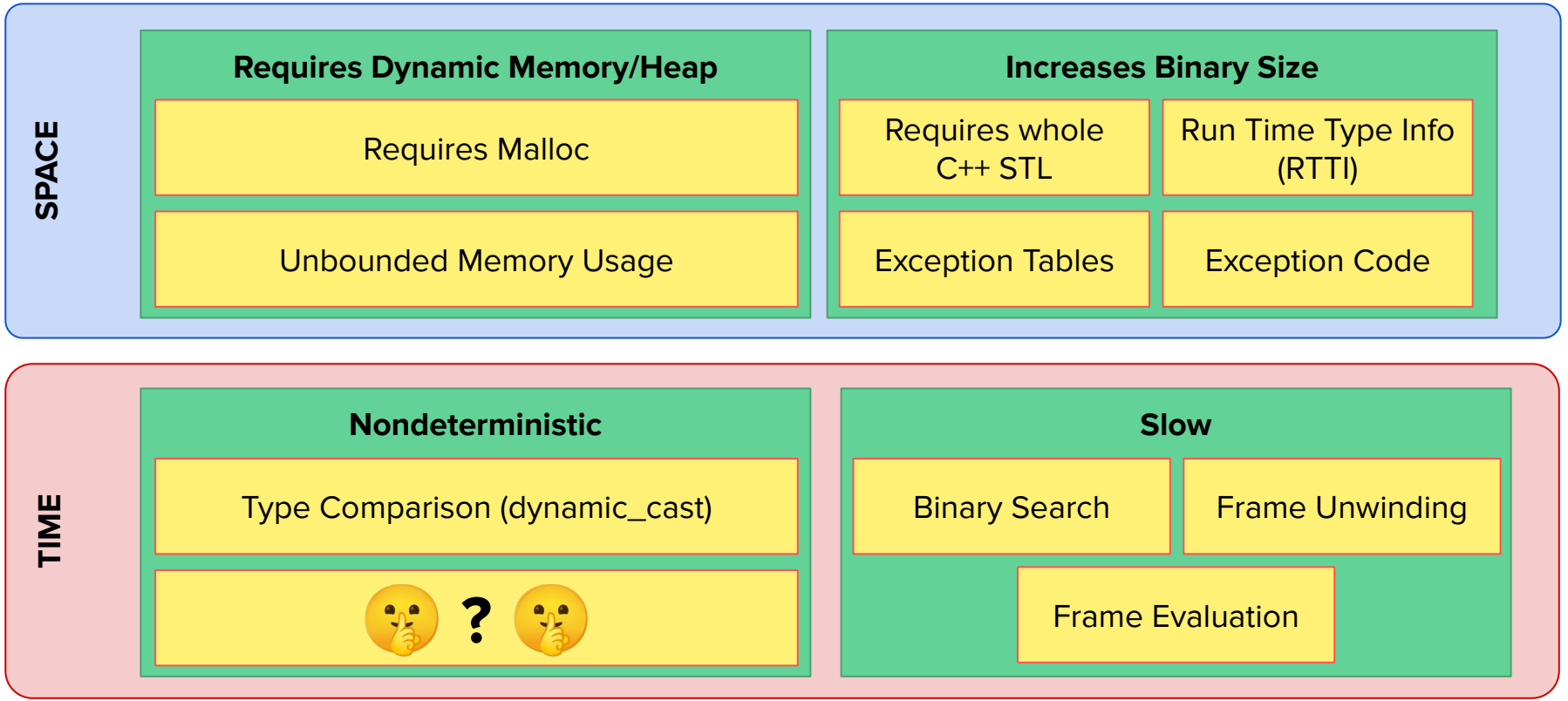
```
std::uint8_t const device_select =
  SJ_CHECK(pin1.level()) << 1 | SJ_CHECK(pin1.level());
auto const voltage = SJ_CHECK(v_sense[device_select].read());
```

MACROs were OK, but noisy

```
std::uint8_t const device_select =
  SJ_CHECK(pin1.level()) << 1 | SJ_CHECK(pin1.level());
auto const voltage = SJ_CHECK(v_sense[device_select].read());
```

So my students asked me,
why not use
C++ exceptions?

I verbally explained them this diagram...



Most accepted it...
Some asked why?

Why is it non-deterministic?

Why is it slow?

Why does it bloat the code?

Why does it need heap?

I would give an answer...
They'd accept...

Let's Make exceptions work on ARM!

Let's start simple

Arm GNU Toolchain GCC 11.3

```
arm-none-eabi-g++
-o except.elf
except.cpp
-std=c++20
-Os -g
-fexceptions
-frtti
-mcpu=cortex-m4
-mfloat-abi=hard -mthumb
--specs=nosys.specs
--specs=nano.specs
--Wl,-T linker.ld
```

```
[[gnu::noinline]] int start()
{
   throw 5;
}

int main()
{
 volatile int return_code = 0;
 try {
   return_code = start();
 } catch (...) {
   return_code = -1;
 }
 return return_code;
}
```

Barrier #1

Exceptions **disabled!**

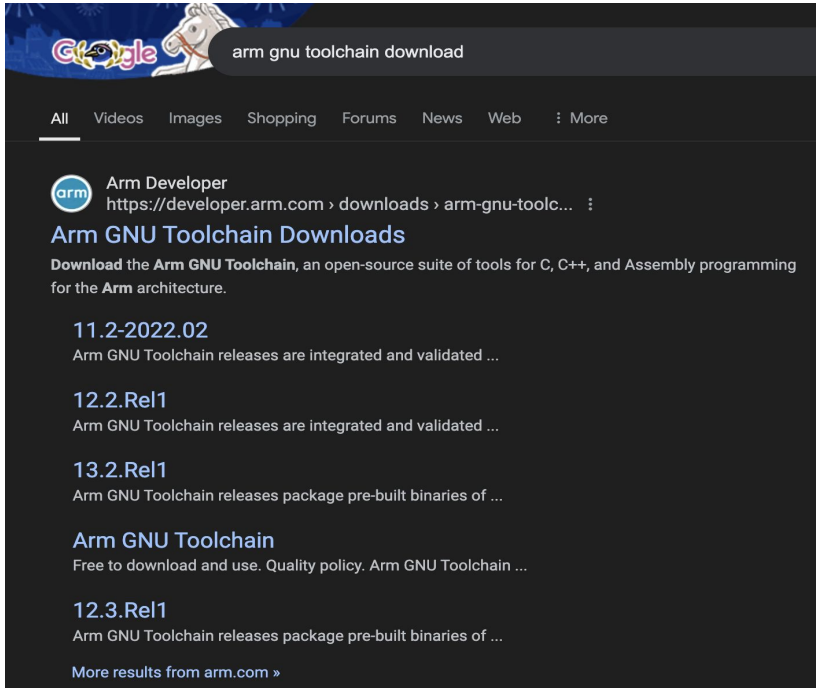
☠️ TERMINATED 💀

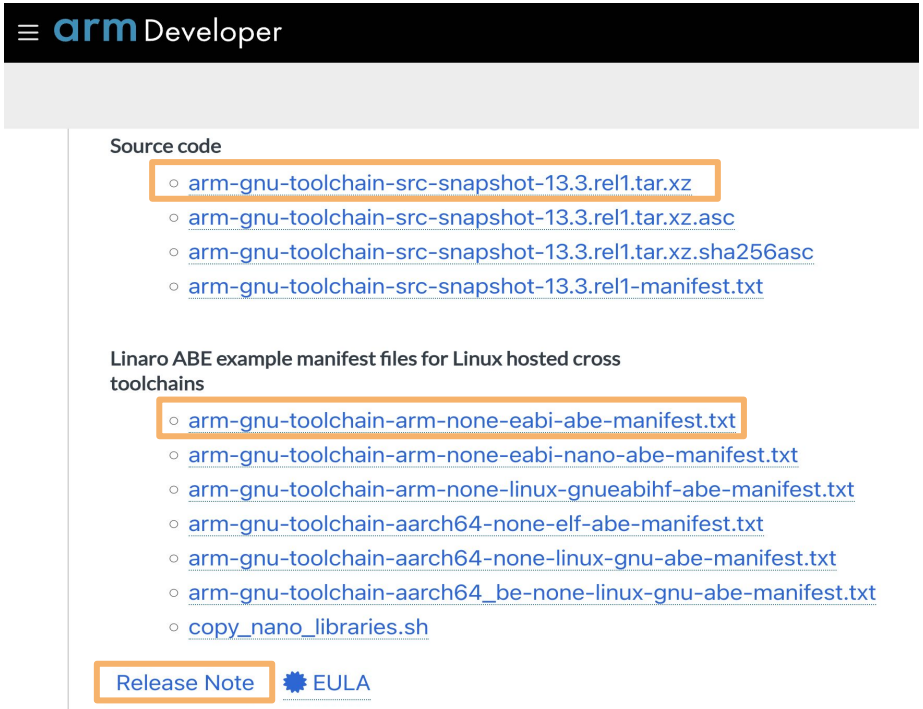
(any time the code throws)

Breaking Barrier #1

Enabling Exceptions in ARM GCC

Step 1: Download & Build "ARM GNU Toolchain download"





Step 2. Find and modify `-fno-exceptions` → `-fexceptions`

Step 3: Define low level C API: `_sbrk`

```
extern "C"
{
 std::array<std::byte, 1024> heap_memory{};
 std::span<std::byte> available_memory = heap_memory;
 void * _sbrk(int p_amount)
 {
   if (p_amount > available_memory.size())
   {
     return nullptr;
   }
   auto result      = available_memory.subspan(0, p_amount);
   available_memory = available_memory.subspan(p_amount);
   return result.data();
 }
}
```

🎉 IT WORKS! 🥳

but ...

Barrier #2

Large Binary Sizes

Size Comparison

‼️😱‼️

+56x bigger

Uses ~29% of 512kB Flash!!

Using Exceptions:

text	data	bss	dec	hex	filename
150008	2016	1328	89352	15d08	except.elf

Using a `std::expected<int, int>` type:

text	data	bss	dec	hex	filename
2680	1372	840	7516	1d5c	expected.elf

Baseline size type:

text	data	bss	dec	hex	filename
2656	1372	840	7516	1d5c	baseline.elf

Breaking Barrier #2

Eliminating the bloat

Breaking Barrier #2: Searching the assembly

```
14496    0000a344 <d_growable_string_callback_adapter>:
14497        a344:   b5f0        push    {r4, r5, r6, r7, lr}
14498        a346:   4614        mov r4, r2
14499        a348:   6852        ldr r2, [r2, #4]
14500        a34a:   68a5        ldr r5, [r4, #8]
14501        a34c:   1c4b        adds    r3, r1, #1
14502        a34e:   4413        add r3, r2
14503        a350:   42ab        cmp r3, r5
14504        a352:   b083        sub sp, #12
14505        a354:   460e        mov r6, r1
14506        a356:   4607        mov r7, r0
14507        a358:   d811        bhi.n   a37e <d_growable_string_callback_adapter+0x3a>
14508        a35a:   68e5        ldr r5, [r4, #12]
14509        a35c:   b96d        cbnz    r5, a37a <d_growable_string_callback_adapter
            +0x36>
14510        a35e:   6863        ldr r3, [r4, #4]
14511        a360:   6820        ldr r0, [r4, #0]
14512        a362:   4632        mov r2, r6
14513        a364:   4418        add r0, r3
14514        a366:   4639        mov r1, r7
14515        a368:   f008 f9a2   bl  126b0 <__aeabi_memcpy>
14516        a36c:   e9d4 3200   ldrd    r3, r2, [r4]
14517        a370:   4433        add r3, r6
14518        a372:   549d        strb    r5, [r3, r2]
14519        a374:   6863        ldr r3, [r4, #4]
14520        a376:   4433        add r3, r6
```

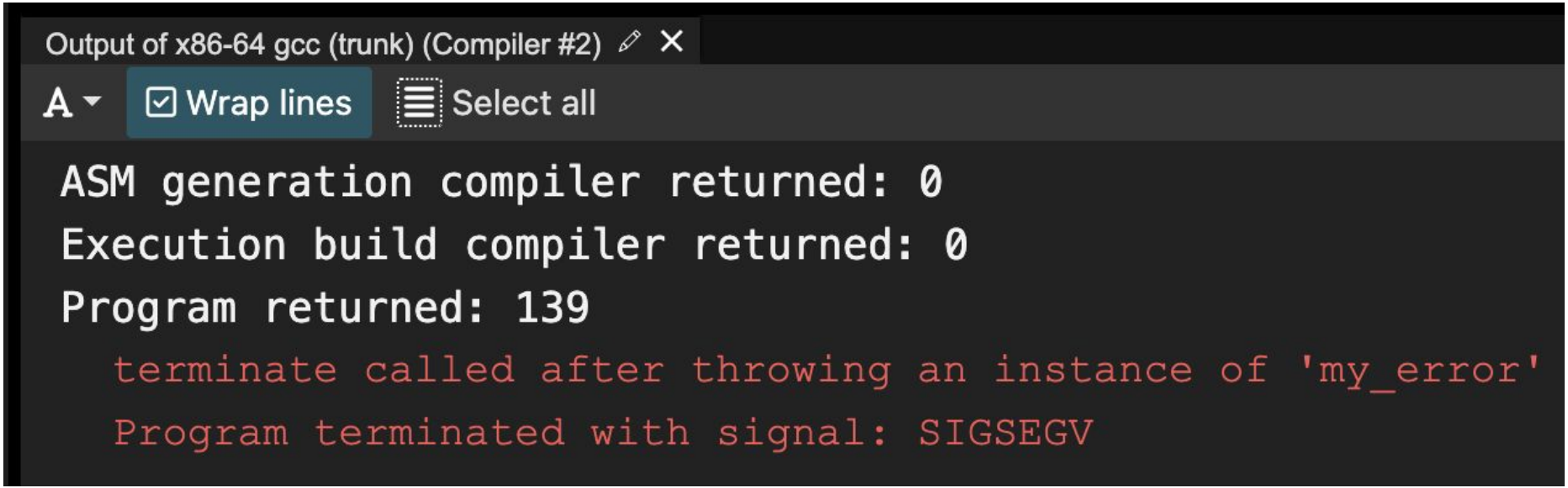
Breaking Barrier #2: Searching the assembly

```
14778   0000a5c4 <d_append_num>:
14779       a5c4:   e92d 41f0   stmdb   sp!, {r4, r5, r6, r7, r8, lr}
14780       a5c8:   b088        sub sp, #32
14781       a5ca:   460a        mov r2, r1
14782       a5cc:   4604        mov r4, r0
14783       a5ce:   491a        ldr r1, [pc, #104]  ; (a638 <d_append_num+0x74>)
14784       a5d0:   a801        add r0, sp, #4
14785       a5d2:   f006 fd91   bl  110f8 <sprintf>
14786       a5d6:   a801        add r0, sp, #4
14787       a5d8:   f008 f880   bl  126dc <strlen>
14788       a5dc:   b340        cbz r0, a630 <d_append_num+0x6c>
14789       a5de:   ad01        add r5, sp, #4
14790       a5e0:   f8d4 1100   ldr.w   r1, [r4, #256]  ; 0x100
14791       a5e4:   182f        adds    r7, r5, r0
14792       a5e6:   f04f 0800   mov.w   r8, #0
14793       a5ea:   e009        b.n a600 <d_append_num+0x3c>
14794       a5ec:   460b        mov r3, r1
14795       a5ee:   42bd        cmp r5, r7
```

Breaking Barrier #2: Find & eliminate the bloat...

```
13923  00009e14 <__gnu_cxx::__verbose_terminate_handler()>:
13924      9e14:   b570        push    {r4, r5, r6, lr}
13925      9e16:   4b3b        ldr r3, [pc, #236]  ; (9f04
           <__gnu_cxx::__verbose_terminate_handler()+0xf0>)
13926      9e18:   781a        ldrb    r2, [r3, #0]
13927      9e1a:   b082        sub sp, #8
13928      9e1c:   2a00        cmp r2, #0
13929      9e1e:   d141        bne.n   9ea4 <__gnu_cxx::__verbose_terminate_handler()
           +0x90>
13930      9e20:   2401        movs    r4, #1
13931      9e22:   701c        strb    r4, [r3, #0]
13932      9e24:   f006 f8f0   bl  10008 <__cxa_current_exception_type>
13933      9e28:   2800        cmp r0, #0
13934      9e2a:   d031        beq.n   9e90 <__gnu_cxx::__verbose_terminate_handler()
           +0x7c>
13935      9e2c:   6844        ldr r4, [r0, #4]
13936      9e2e:   4d36        ldr r5, [pc, #216]  ; (9f08
           <__gnu_cxx::__verbose_terminate_handler()+0xf4>)
13937      9e30:   7823        ldrb    r3, [r4, #0]
13938      9e32:   2b2a        cmp r3, #42 ; 0x2a
13939      9e34:   bf08        it  eq
13940      9e36:   3401        addeq   r4, #1
13941      9e38:   2200        movs    r2, #0
13942      9e3a:   f04f 30ff   mov.w   r0, #4294967295 ; 0xffffffff
13943      9e3e:   4611        mov r1, r2
13944      9e40:   ab01        add r3, sp, #4
13945      9e42:   9001        str r0, [sp, #4]
13946      9e44:   4620        mov r0, r4
13947      9e46:   f006 f849   bl  fedc <__cxa_demangle>
```

Breaking Barrier #2: Within GCC



So what can we do?

Breaking Barrier #2: with 5 lines of code

```
namespace __cxxabiv1 {
std::terminate_handler __terminate_handler = +[]() {
 while (true) { continue; }
};
}
```

Overrides GCC's weak symbol

Code Size Reduced!

text	data	bss	dec	hex	filename
13632	120	660	14412	384c	except.elf

-91% size reduction

Barrier #3

Uses **dynamic** allocation

Barrier #3: What is a throw expression in assembly?

```
0000800c <start()>:
   800c:   b508        push    {r3, lr}
   800e:   2004        movs    r0, #4
   8010:   f000 f93c   bl  828c <__cxa_allocate_exception>
   8014:   2305        movs    r3, #5
   8016:   4902        ldr r1, [pc, #8]
   8018:   6003        str r3, [r0, #0]
   801a:   2200        movs    r2, #0
   801c:   f000 fd0a   bl  8a34 <__cxa_throw>
   8020:   0000af48    .word   0x0000af48
```

Breaking Barrier #3

Overriding the allocator

Breaking Barrier #3: Override the allocator

```
extern "C"
{
std::array<std::uint8_t, 256> storage{};
void* __cxa_allocate_exception(unsigned int p_size) noexcept
{
  static constexpr size_t offset = 128;
  return storage.data() + offset;
}
void __cxa_free_exception(void*) noexcept
{
  // Do nothing here.
}
}  // extern "C"
```

Barrier #4

The Spectre of RTTI

Breaking Barrier #4

Disabling RTTI

Breaking Barrier #4: Disabling RTTI

Action: Replace `-frtti` with `-fno-rtti`

What happens if you disable it? ***Note: C++ Exceptions REQUIRE RTTI to work.***

1. 🟡 Compiler tells you that you cannot use C++ exceptions with RTTI disabled
2. 🟡 Compiler strips all RTTI information not involved in exception handling?
3. 🟡 Compiler is silent and removes RTTI, using C++ exceptions becomes UB

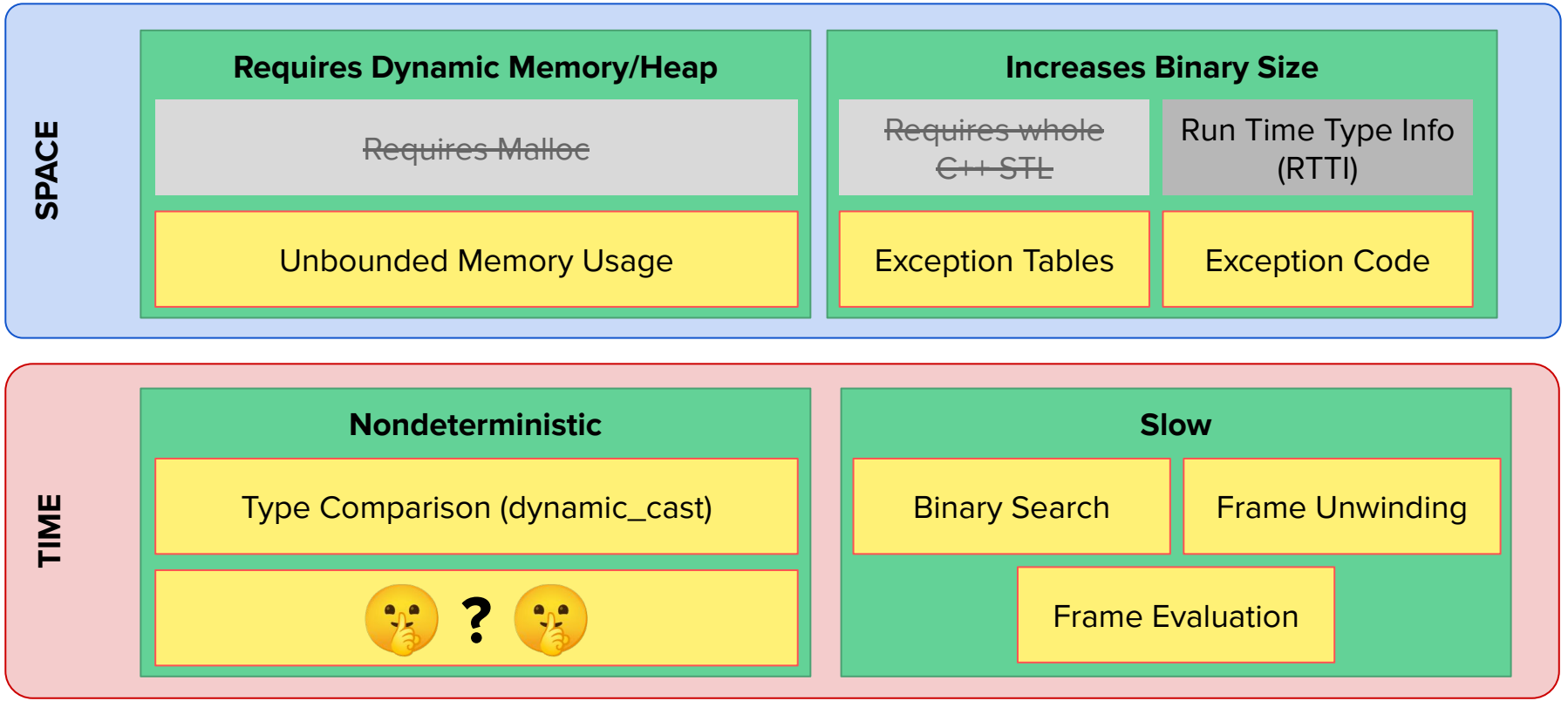
Breaking Barrier #4: Disabling RTTI

Action: Replace `-frtti` with `-fno-rtti`

What happens if you disable it?

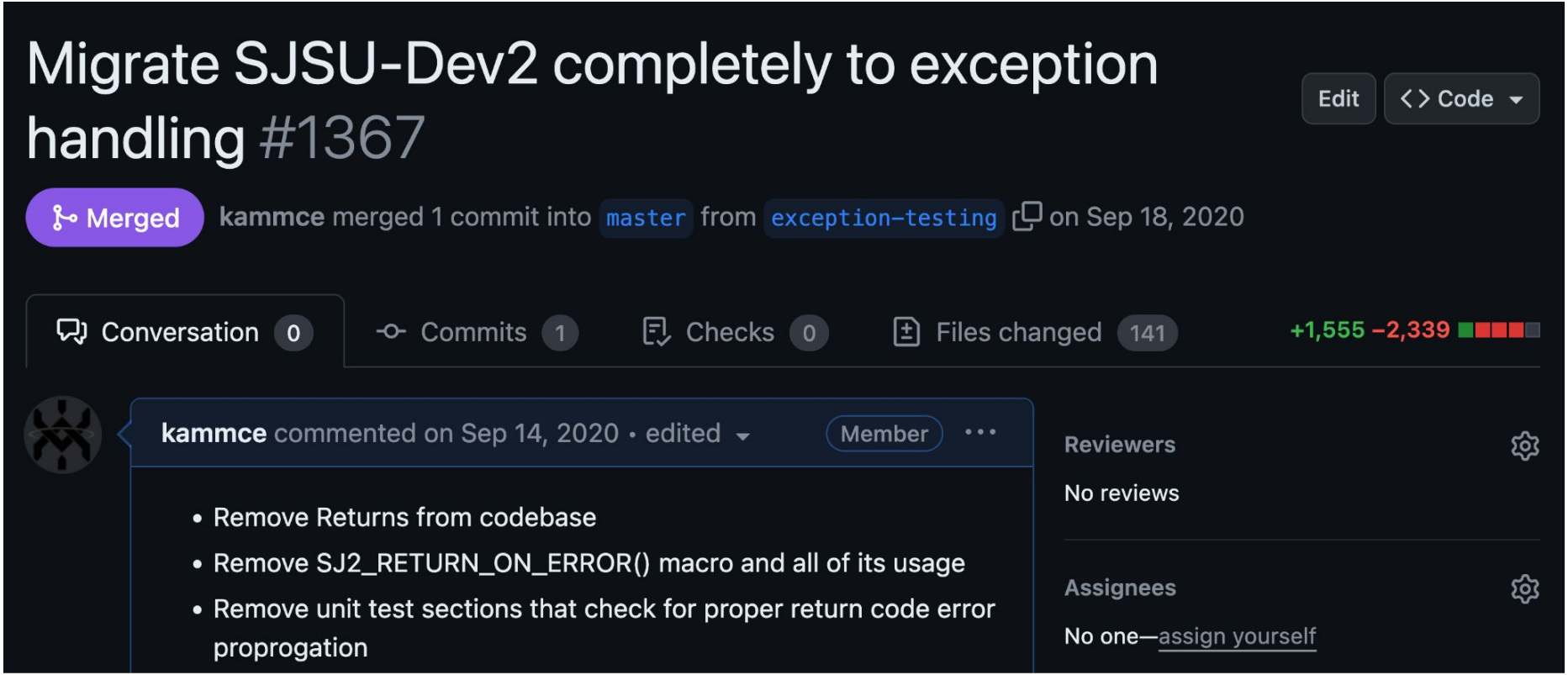
1. ❌ Compiler tells you that you cannot use C++ exceptions with RTTI disabled
2. ✅ Compiler strips all RTTI information not involved in exception handling?
3. ❌ Compiler is silent and removes RTTI, using C++ exceptions becomes UB

So why do firmware developers avoid C++ exceptions?

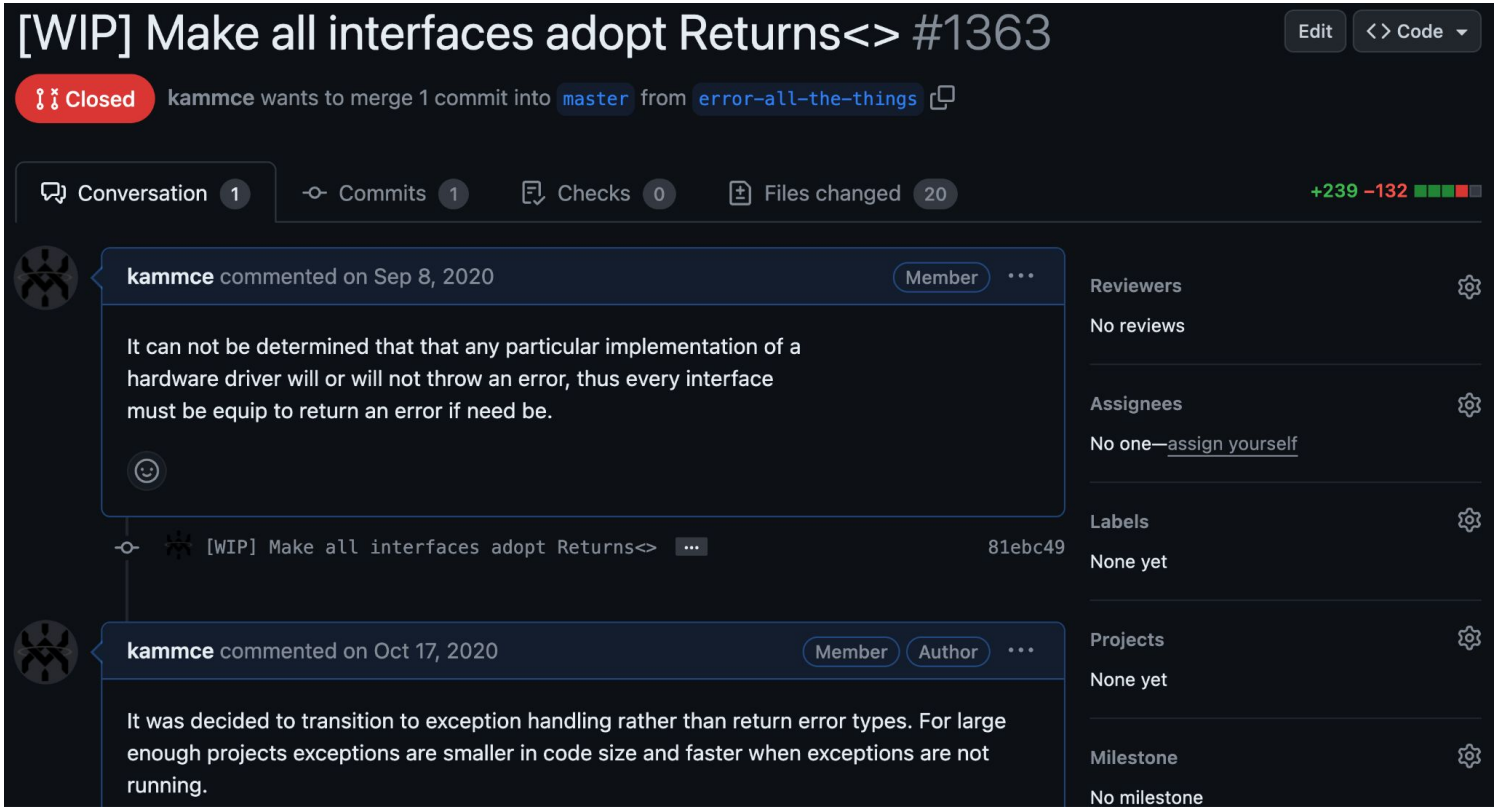


But at this point,
I had seen enough!

Refactoring my project to use exceptions



Result<T> PR closed



What I had observed in terms of Binary Size

Small Projects/Demos

Large Projects

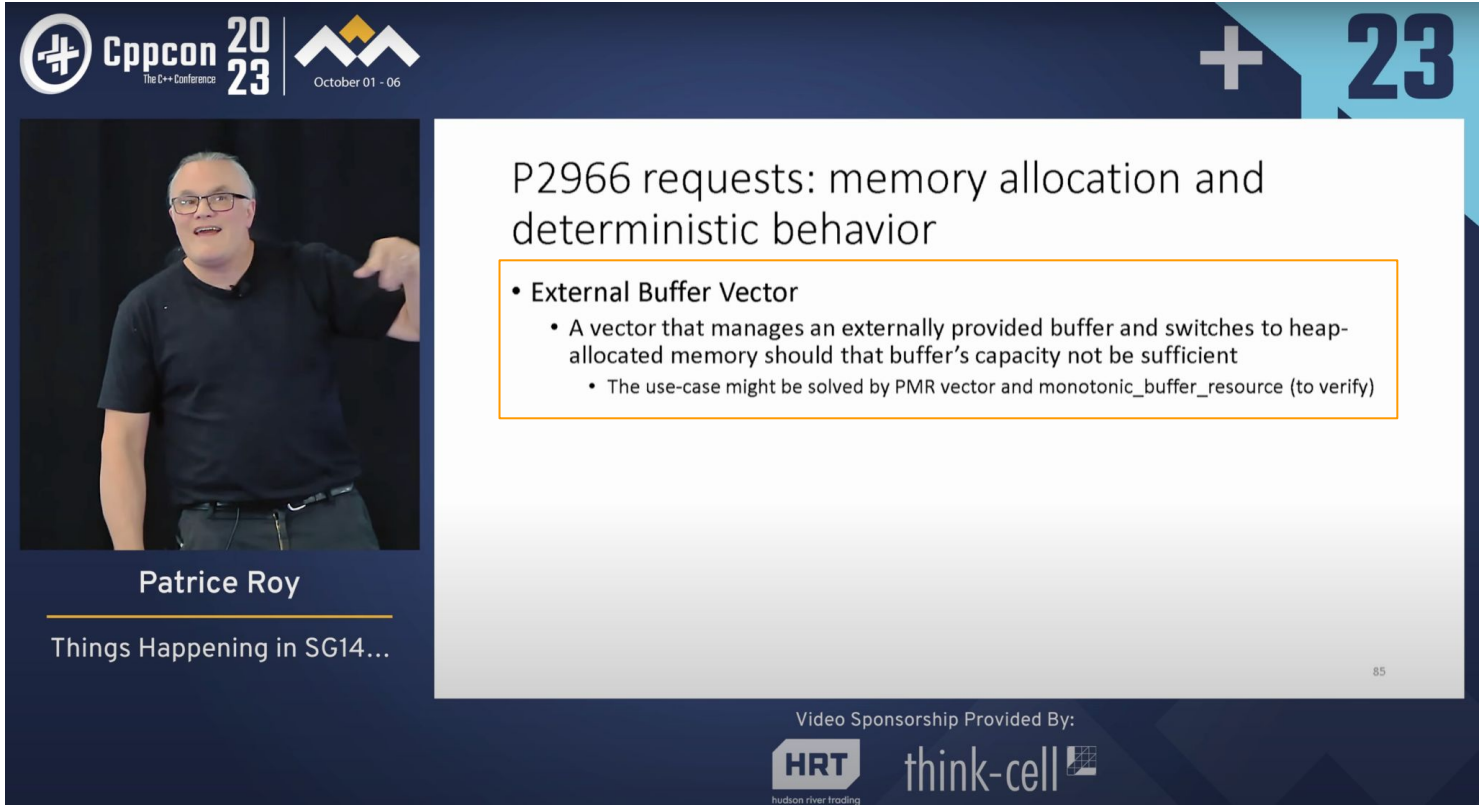
What I had observed in terms of Binary Size

Small Projects/Demos

Large Projects

ISO C++ Meeting 2023 @ CppCon

Things Happening in SG14... - Patrice Roy - CppCon 2023



Concerns with std::pmr::vector

std::vector<T,Allocator>::push_back

`void push_back( const T& value );`	(1)	(constexpr since C++20)
`void push_back( T&& value );`	(2)	(since C++11) (constexpr since C++20)

How do you know it worked without exceptions?

"Its a shame embedded devs don't use exceptions...

It would make their binaries smaller."

My Hypothesis

- Cost of using result types: `S(N) = aN`
  - `a` = average cost of a check per call
  - `N` = number of checked function calls

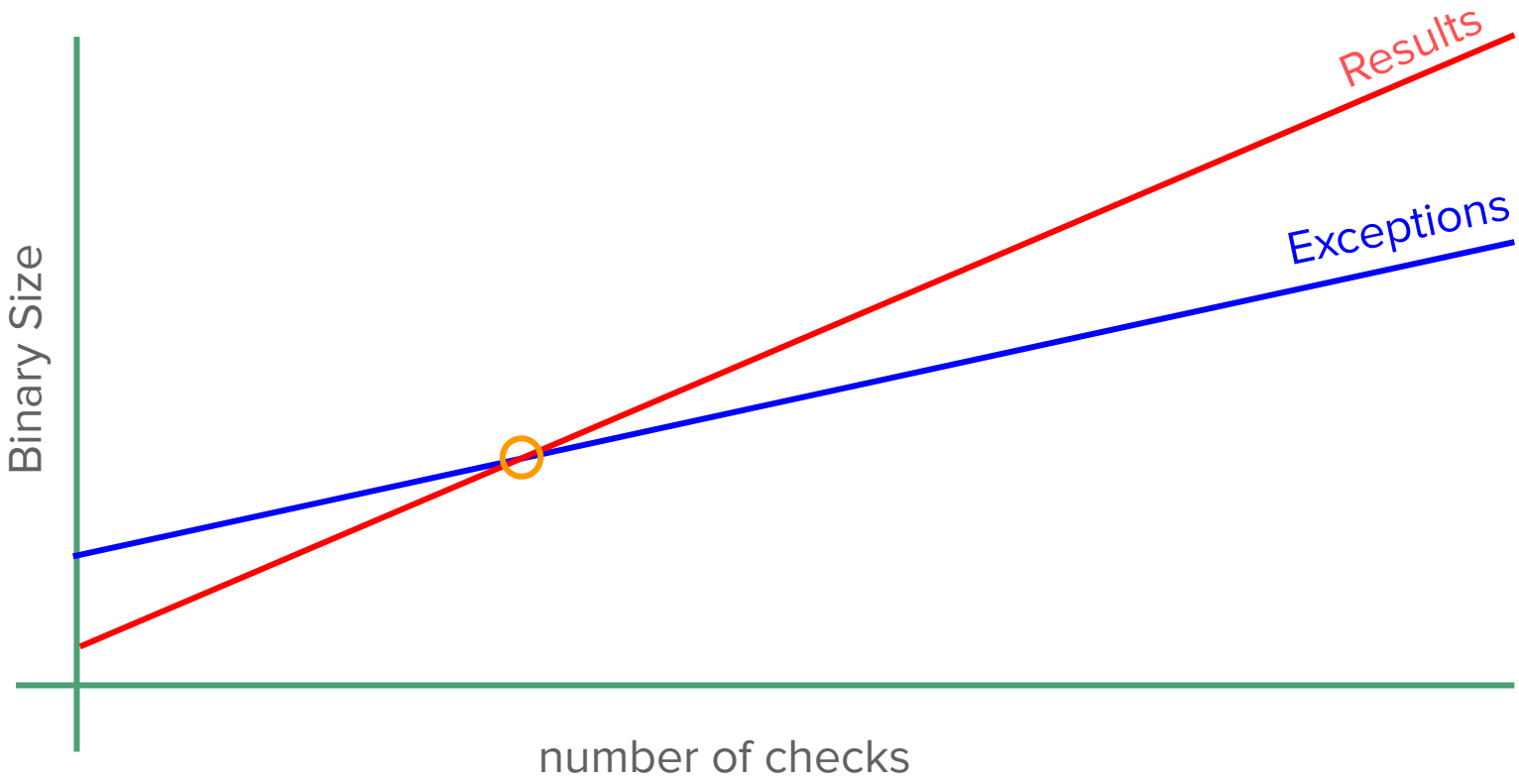
My Hypothesis

- Cost of using result types: `S(N) = aN`
  - `a` = average cost of a check per call
  - `N` = number of checked function calls
- Cost of using exception: `S(M) = bM + E`
  - `b` = average cost of exception metadata
  - `M` = number of non-leaf functions
  - `E` = cost of exception machinery (libunwind, etc)
  - **NOTE: this is grossly simplified**

My Hypothesis

- Cost of using result types: `S(N) = aN`
  - `a` = average cost of a check per call
  - `N` = number of checked function calls
- Cost of using exception: `S(M) = bM + E`
  - `b` = average cost of exception metadata
  - `M` = number of non-leaf functions
  - `E` = cost of exception machinery (libunwind, etc)
  - **NOTE: this is grossly simplified**
- I assert that: aN > bM
  - Let's assume we have an application of 100 error reporting functions
  - **N (# of functions called) >= M (total number of functions)**

My Hypothesis



The Experiment: Generating C++ code

Due to time constraints, I cannot go over this program in full.

- Defines classes with and without trivial destructors
- Create sequences of functions that call other functions
- Those functions would also create class objects and call their class functions

I wrote a program to compare **std::expected** to **exception handling**.

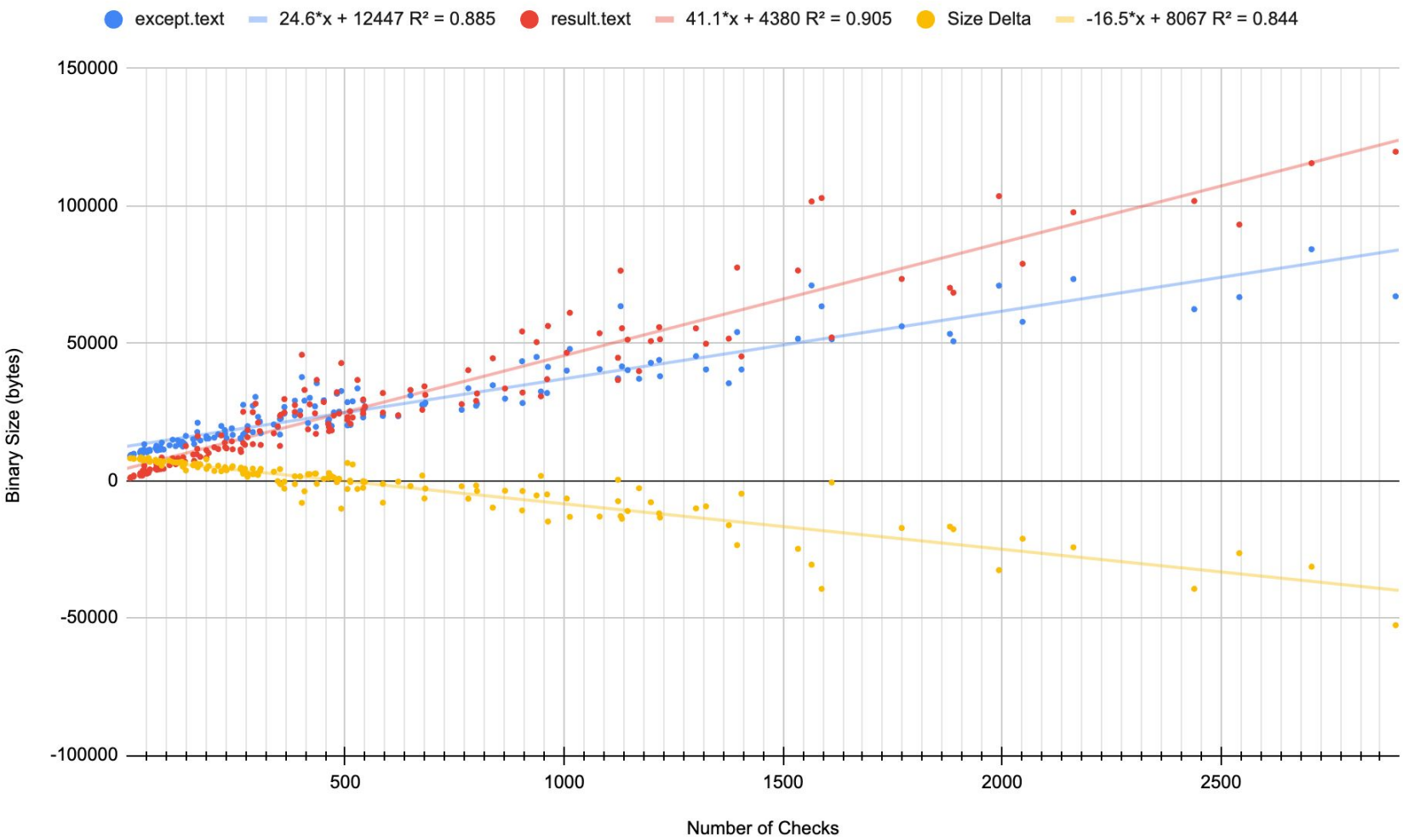
My requirements were:

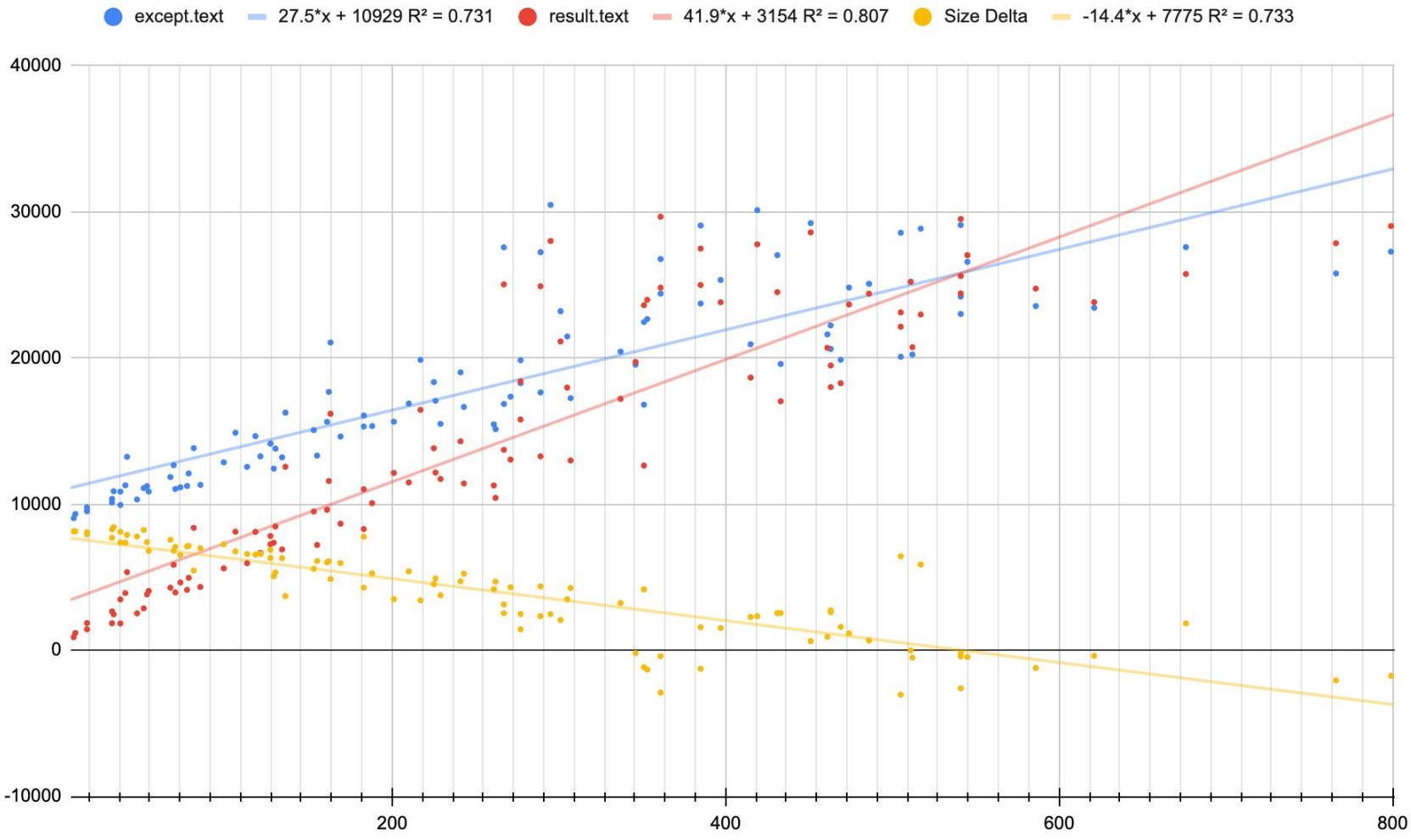
1. Generate two C++ programs (`except.cpp` & `result.cpp`) that are isomorphic to each other.
2. Focus on propagation vs handling.

Sample Generated C++ code

```cpp
int funct_group11_8()
{
 volatile static std::uint32_t inner_side_effect = 0;
 inner_side_effect = inner_side_effect + 1;
 class_0 instance_0(side_effect);
 instance_0.trigger();
 instance_0.trigger();
 instance_0.trigger();
 instance_0.trigger();

 class_1 instance_1(side_effect);
 instance_1.trigger();

 class_2 instance_2(side_effect);
 instance_2.trigger();
 instance_2.trigger();
 side_effect = side_effect - funct_group11_9();
 return side_effect;
}
```

```cpp
std::expected<int, my_error_t> funct_group11_8()
{
 volatile static std::uint32_t inner_side_effect = 0;
 inner_side_effect = inner_side_effect + 1;

 auto instance_0 = class_0::make(side_effect);
 if (!instance_0) {
   return std::unexpected(instance_0.error());
 }
 {
   auto scoped_result = instance_0.value().trigger();
   if (!scoped_result) {
     return std::unexpected(scoped_result.error());
   }
 }
 {
   auto scoped_result = instance_0.value().trigger();
   if (!scoped_result) {
     return std::unexpected(scoped_result.error());
   }
 }
 {
   auto scoped_result = instance_0.value().trigger();
   if (!scoped_result) {
     return std::unexpected(scoped_result.error());
   }
 }
 // etc...
 if (auto result = funct_group11_9(); !result) {
   return std::unexpected(result.error());
 } else {
   side_effect = side_effect - result.value();
 }
 return side_effect;
}
```


except.text
24.6*x + 12447 R² = 0.885
result.text
41.1*x + 4380 R² = 0.905
Size Delta
-16.5*x + 8067 R² = 0.844
150000
100000
50000
0
-50000
-100000
Binary Size (bytes)
500
1000
1500
2000
2500
Number of Checks


except.text
27.5*x + 10929 R² = 0.731
result.text
41.9*x + 3154 R² = 0.807
Size Delta
-14.4*x + 7775 R² = 0.733
40000
30000
20000
10000
0
-10000
200
400
600
800

But that wasn't to convince
you

That was to show you that
something is here.

C++ Exceptions from throw to catch on GCC ARM

Things that will NOT be covered here

- Nested exceptions
- Anything other than table based exceptions

Consider the following

```
struct error {};

void foo() {
 try {
   bar();
 } catch (error const& p_error) {
   // end up here...
 }
}

void bar() {
 destructible_t obj;
 baz();
} // ~destructible_t(&my_object)

void baz() { throw error{}; }
```

The 3 types of functions

Leaf Functions

Transparent-Trivial Functions

Reentrant Functions

The 3 types of functions

Leaf Functions

❌ Calls other functions
❌ Contains destructors
❌ try/catch

Transparent-Trivial Functions

Reentrant Functions

The 3 types of functions

Leaf Functions

❌ Calls other functions
❌ Contains destructors
❌ try/catch

Transparent-Trivial Functions

✅ Calls other functions
❌ Contains destructors
❌ try/catch

Reentrant Functions

The 3 types of functions

Leaf Functions

❌ Calls other functions
❌ Contains destructors
❌ try/catch

Transparent-Trivial Functions

✅ Calls other functions
❌ Contains destructors
❌ try/catch

Reentrant Functions

✅ Calls other functions
🟡 Contains destructors
🟡 try/catch

Classifying the functions

Foo

Bar

Baz

```
struct error {};

void foo() {
 try {
   bar();
 } catch (error const& p_error) {
   // end up here...
 }
}

void bar() {
 destructible_t obj;
 baz();
} // ~destructible_t(&my_object)

void baz() { throw error{}; }
```

ARM Cortex-M Registers & Calling Convention

Registers	Info
R0	**Scratch Registers:**
R1	Used to pass parameters & return values
R2	Additional parameters will be passed on the stack if the inputs exceed what can be passed in the registers
R3	
R4	**Preserved Registers:**
R5	Must be restored before a subroutine returns.
R6	On function entry: Needed registers are pushed onto the stack.
R7	On function exit: Used registers are popped off the stack back into the registers.
R8	
R9	
R10	
R11	
R12	**Scratch Register**
R13 / SP	Stack Pointer
R14 / LR	Link Register (Updated with the return address of a subroutine call) Pushed on the stack if the function calls other functions
R15 / PC	Program Counter (current location in the program)

ARM Cortex-M Registers & Calling Convention

```
00003720 <foo()>:
   3720:   b510        push    {r4, lr} ; save R4 & LR
   ;
   ; Do stuff...
   ; Call a function at some point...
   ;
   372c:   4620        mov r0, r4 ; Set return value
   ; Restore R4 back to R4 & LR into PC return to the caller
   372e:   bd10        pop {r4, pc}
```

Virtual ARM CPU		SRAM	
Reg	*Value*	*Address*	*Value*
R0	XXXX	0x4000	0x????????
R1	XXXX	0x3FFC	0x????????
R2	XXXX	0x3FF8	0x????????
R3	XXXX	0x3FF4	0x????????
R4	?	0x3FF0	0x????????
R5	?	0x3FEC	caller of foo
R6	–	0x3FE8	caller r4
R7	–	0x3FE4	caller r5
R8	–	0x3FE0	foo()+10
R9	–	0x3FDC	[foo] r5
R10	–	0x3FD8	[foo] r4
R11	–	0x3FD4	obj[1/3]
R12	–	0x3FD0	obj[2/3]
R13/SP	?	0x3FCC	obj[3/3]
R14/LR	?	0x3FC8	bar()+14
R15/PC	?	0x3FC4	[bar] r4

```
struct error {};

void foo() {
 try {
   bar();
 } catch (error const& p_error) {
   // end up here...
 }
}

void bar() {
 destructible_t obj;
 baz();
} // ~destructible_t(&my_object)

void baz() { throw error{}; }
```

Virtual ARM CPU		SRAM	
Reg	*Value*	*Address*	*Value*
R0	XXXX	0x4000	0x????????
R1	XXXX	0x3FFC	0x????????
R2	XXXX	0x3FF8	0x????????
R3	XXXX	0x3FF4	0x????????
R4	?	0x3FF0	0x????????
R5	?	0x3FEC	caller of foo
R6	–	0x3FE8	caller r4
R7	–	0x3FE4	caller r5
R8	–	0x3FE0	foo()+10
R9	–	0x3FDC	[foo] r5
R10	–	0x3FD8	[foo] r4
R11	–	0x3FD4	obj[1/3]
R12	–	0x3FD0	obj[2/3]
R13/SP	?	0x3FCC	obj[3/3]
R14/LR	?	0x3FC8	bar()+14
R15/PC	?	0x3FC4	[bar] r4

```
struct error {};

void foo() {
 try {
   bar();
 } catch (error const& p_error) {
   // end up here...
 }
}

void bar() {
 destructible_t obj;
 baz();
} // ~destructible_t(&my_object)

void baz() { throw error{}; }
```

We are here

Itanium C++ ABI: What are throws?

```
struct my_error {};

void throws_my_error()
{
  throw my_error{};
}
```

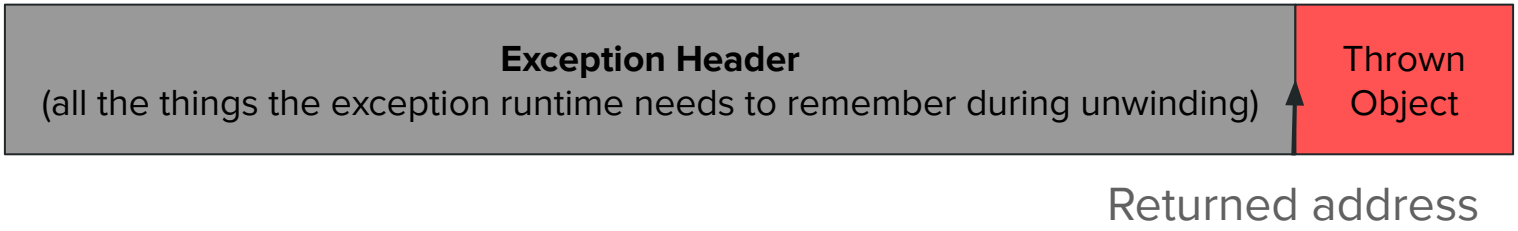
```
throws_my_error():
        push    {r7, lr}
        add     r7, sp, #0
        movs    r0, #1
        bl      __cxa_allocate_exception
        movs    r3, r0
        movs    r0, r3
        ldr     r3, .L18
        movs    r2, #0
        movs    r1, r3
        bl      __cxa_throw
.L18:
        .word   typeinfo for my_error
```

Itanium C++ ABI: Throw guidelines

```
void __cxa_throw(
        void* thrown_exception,
        std::type_info* type_info,
        void (*dtor)(void*));
```

Itanium C++ ABI: How `__cxa_allocate_exception` works?

```
std::array<std::uint8_t, 1024> exception_storage;
void* __cxa_allocate_exception(size_t p_thrown_size)
{
 if (exception_storage.size() < p_thrown_size + sizeof(exception_header)) {
   std::terminate();
 }
 return exception_storage.data() + sizeof(exception_header);
}
```



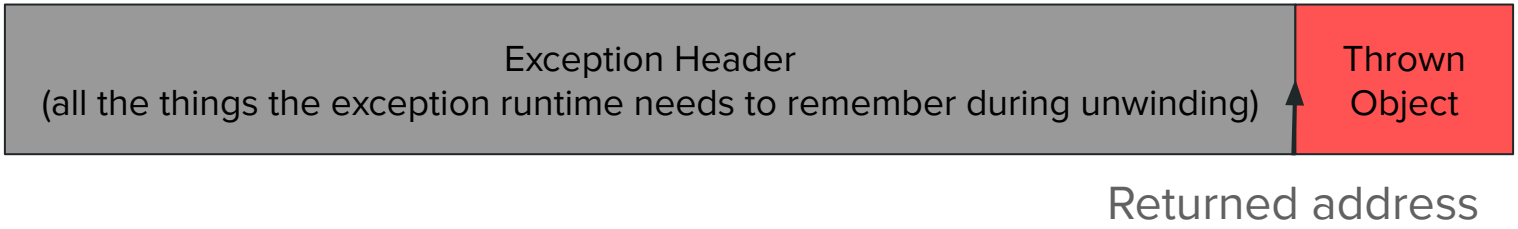
Itanium C++ ABI: What is the Exception Header?

```
struct exception_header
{
 cortex_m_cpu cpu{};
 std::type_info* type_info;
 destructor_t* destructor;
 cache_t cache{};
};
```

```
enum class runtime_state : std::uint8_t
{
 get_next_frame = 0,
 enter_function = 1,
 unwind_frame = 2,
 handled = 3,
};

struct cache_t
{
 std::uint32_t state_and_rel_address;
 const index_entry_t* entry_ptr;
}
```

```
struct cortex_m_cpu
{
 register_t r0; // Remove?
 register_t r1; // Remove?
 register_t r2; // Remove?
 register_t r3; // Remove?
 register_t r4;
 register_t r5;
 register_t r6;
 register_t r7;
 register_t r8;
 register_t r9;
 register_t r10;
 register_t r11;
 register_t ip;
 register_t sp;
 register_t lr;
 register_t pc;
};
```

Why not call `malloc`?



Because of **ABI Compatibility & Flexibility.**

- Allows the size of the exception header to grow or shrink without an ABI break
- Reduces the allocations to 1

Itanium C++ ABI: How `__cxa_throw` works

Virtual ARM CPU		SRAM	
Reg	*Value*	*Address*	*Value*
R0	XXXX	0x4000	0x????????
R1	XXXX	0x3FFC	0x????????
R2	XXXX	0x3FF8	0x????????
R3	XXXX	0x3FF4	0x????????
R4	?	0x3FF0	0x????????
R5	?	0x3FEC	caller of foo
R6	–	0x3FE8	caller r4
R7	–	0x3FE4	caller r5
R8	–	0x3FE0	foo()+10
R9	–	0x3FDC	[foo] r5
R10	–	0x3FD8	[foo] r4
R11	–	0x3FD4	obj[1/3]
R12	–	0x3FD0	obj[2/3]
R13/SP	?	0x3FCC	obj[3/3]
R14/LR	?	0x3FC8	bar()+14
R15/PC	?	0x3FC4	[bar] r4

```
void __cxa_throw(exception_ptr p_thrown_exception,
                 std::type_info* p_type_info,
                 destructor_t p_destructor)
{
 // TLS variable for std::current_exception
 active_exception = p_thrown_exception;
 auto& exception_header = extract_exception_header(
                            p_thrown_exception);
 exception_header.type_info = p_type_info;
 exception_header.destructor = p_destructor;

capture_cpu_core_and_unwind1(exception_header.cpu);
 raise_exception(exception_header);
 std::terminate();
}
```

Itanium C++ ABI: How `__cxa_throw` works

Virtual ARM CPU		SRAM	
Reg	*Value*	*Address*	*Value*
R0	0x2000'C000	0x4000	0x????????
R1	type_info	0x3FFC	0x????????
R2	0x0	0x3FF8	0x????????
R3	0x????????	0x3FF4	0x????????
R4	0x????????	0x3FF0	0x????????
R5	0x????????	0x3FEC	caller of foo
R6	–	0x3FE8	caller r4
R7	–	0x3FE4	caller r5
R8	–	0x3FE0	foo()+10
R9	–	0x3FDC	[foo] r5
R10	–	0x3FD8	[foo] r4
R11	–	0x3FD4	obj[1/3]
R12	–	0x3FD0	obj[2/3]
R13/SP	0x3FC4	0x3FCC	obj[3/3]
R14/LR	baz()+4	0x3FC8	bar()+14
R15/PC	__cxa_throw	0x3FC4	[bar] r4

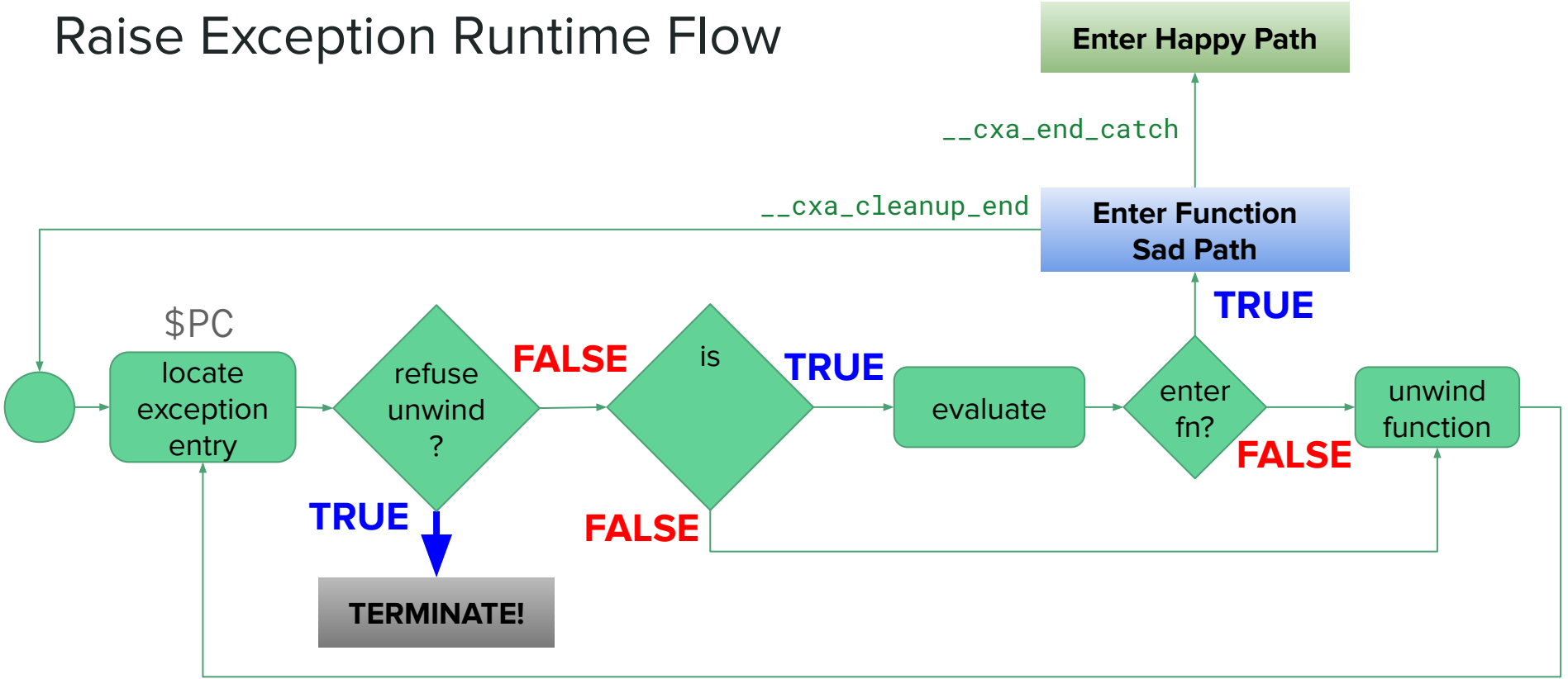
```
void __cxa_throw(exception_ptr p_thrown_exception,
                 std::type_info* p_type_info,
                 destructor_t p_destructor)
{
 // TLS variable for std::current_exception
 active_exception = p_thrown_exception;
 auto& exception_header = extract_exception_header(
                            p_thrown_exception);
 exception_header.type_info = p_type_info;
 exception_header.destructor = p_destructor;

capture_cpu_core_and_unwind1(exception_header.cpu);
 raise_exception(exception_header);
 std::terminate();
}
```

Itanium C++ ABI: How __cxa_throw works

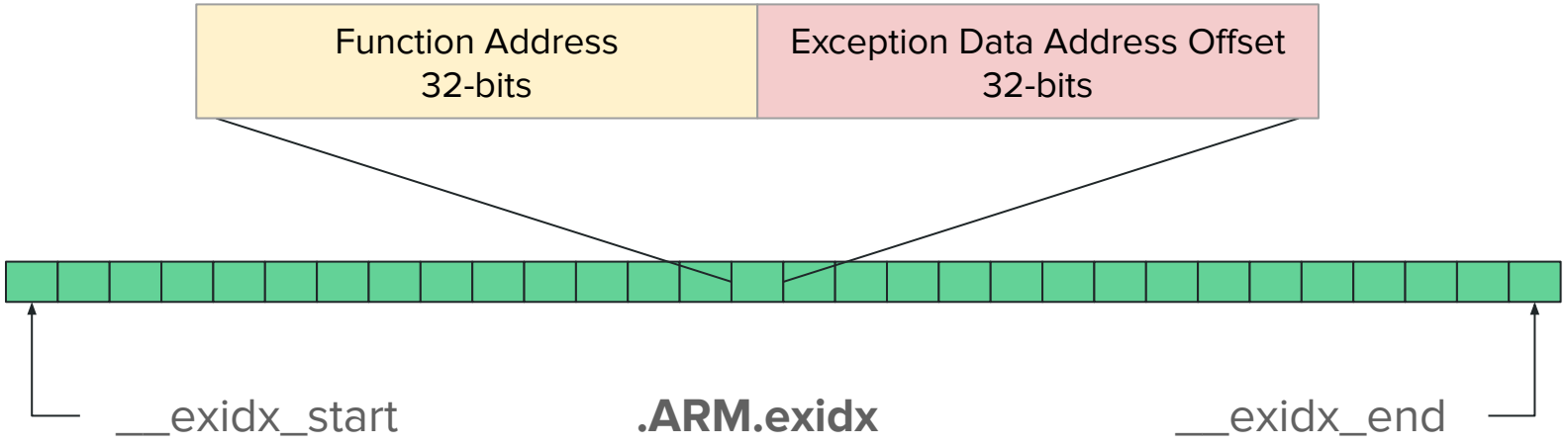
Virtual ARM CPU		SRAM	
Reg	*Value*	*Address*	*Value*
R0	0x2000'C000	0x4000	0x????????
R1	type_info	0x3FFC	0x????????
R2	0x0	0x3FF8	0x????????
R3	0x????????	0x3FF4	0x????????
R4	0x????????	0x3FF0	0x????????
R5	0x????????	0x3FEC	caller of foo
R6	–	0x3FE8	caller r4
R7	–	0x3FE4	caller r5
R8	–	0x3FE0	foo()+10
R9	–	0x3FDC	[foo] r5
R10	–	0x3FD8	[foo] r4
R11	–	0x3FD4	obj[1/3]
R12	–	0x3FD0	obj[2/3]
R13/SP	0x3FC4	0x3FCC	obj[3/3]
R14/LR	baz()+4	0x3FC8	bar()+14
R15/PC	baz()+4	0x3FC4	[bar] r4

```
void __cxa_throw(exception_ptr p_thrown_exception,
                 std::type_info* p_type_info,
                 destructor_t p_destructor)
{
 // TLS variable for std::current_exception
 active_exception = p_thrown_exception;
 auto& exception_header = extract_exception_header(
                            p_thrown_exception);
 exception_header.type_info = p_type_info;
 exception_header.destructor = p_destructor;

capture_cpu_core_and_unwind1(exception_header.cpu);
 raise_exception(exception_header);
 std::terminate();
}
```

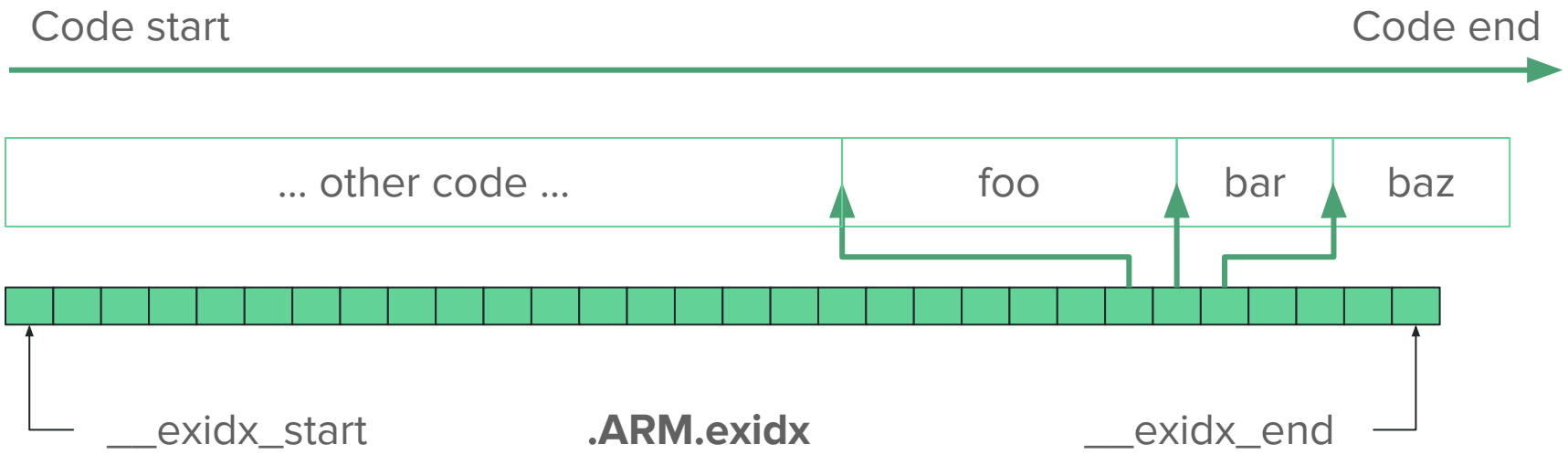
Raise Exception Runtime Flow



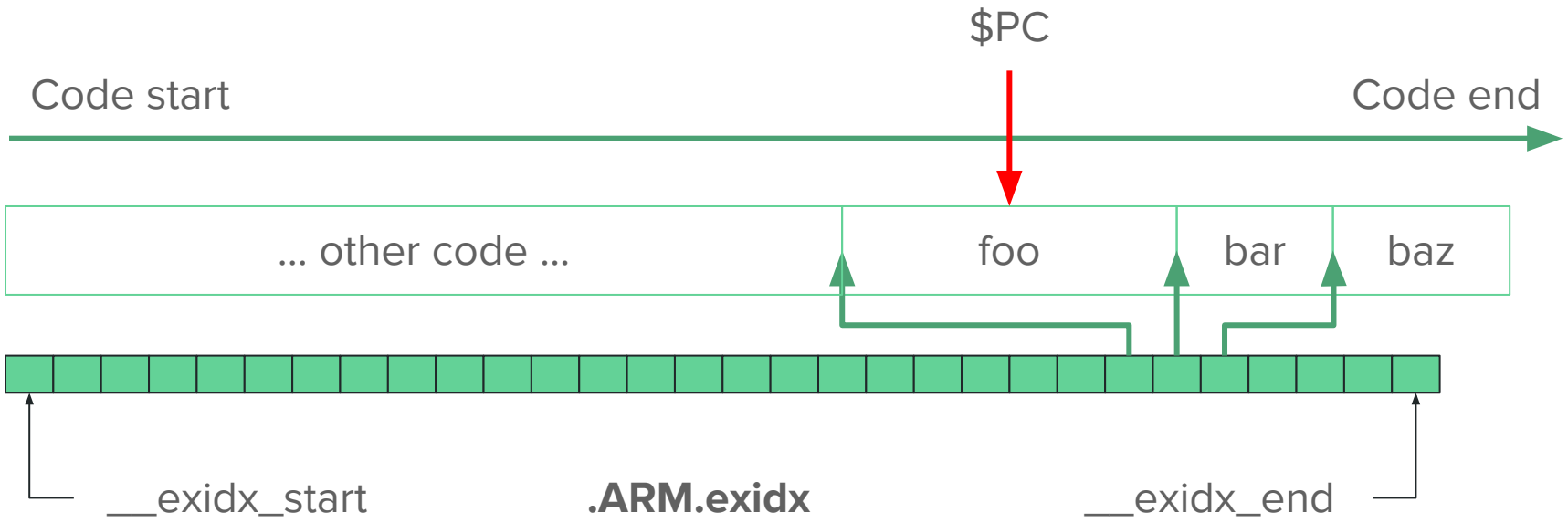
ARM Exception Index: Index Entry



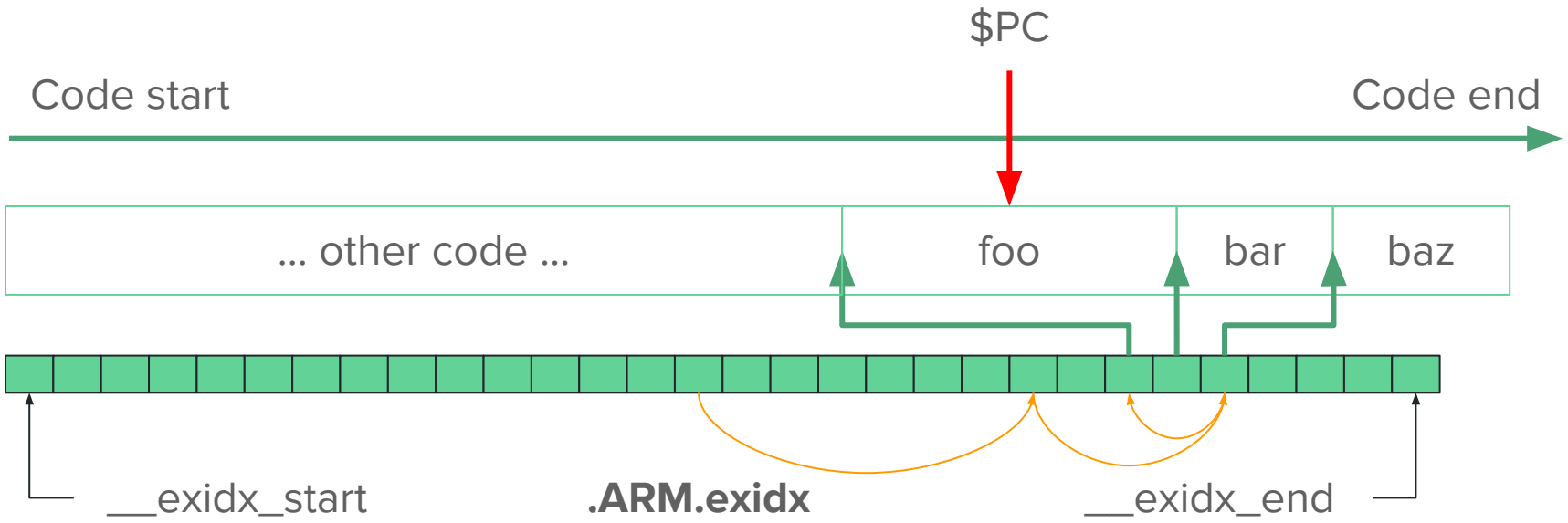
ARM Exception Index: Ordering



ARM Exception Index: Ordering



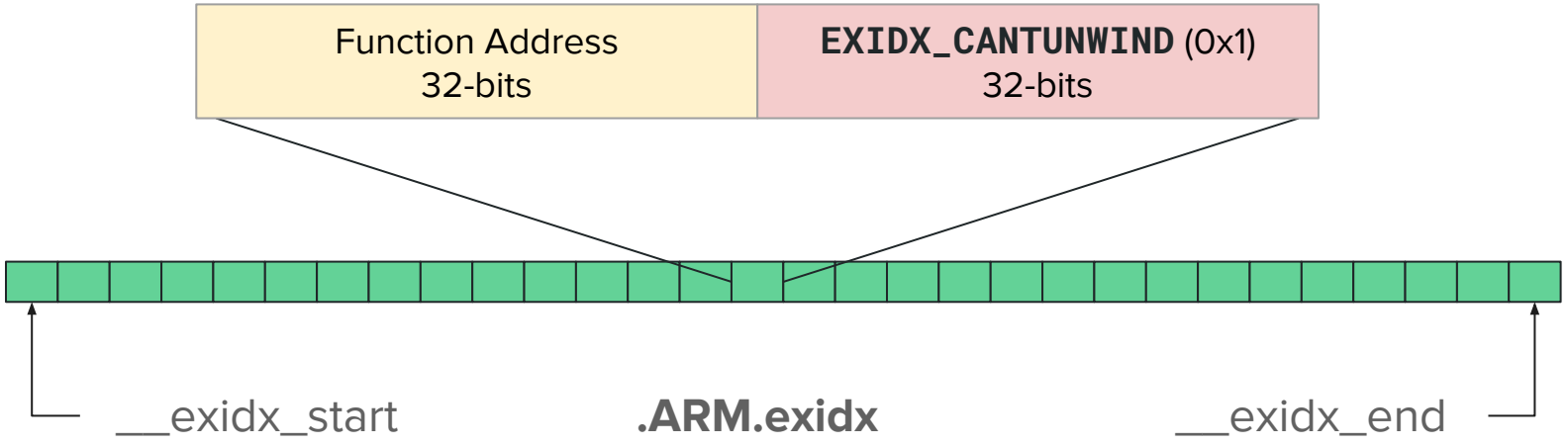
ARM Exception Index: Ordering



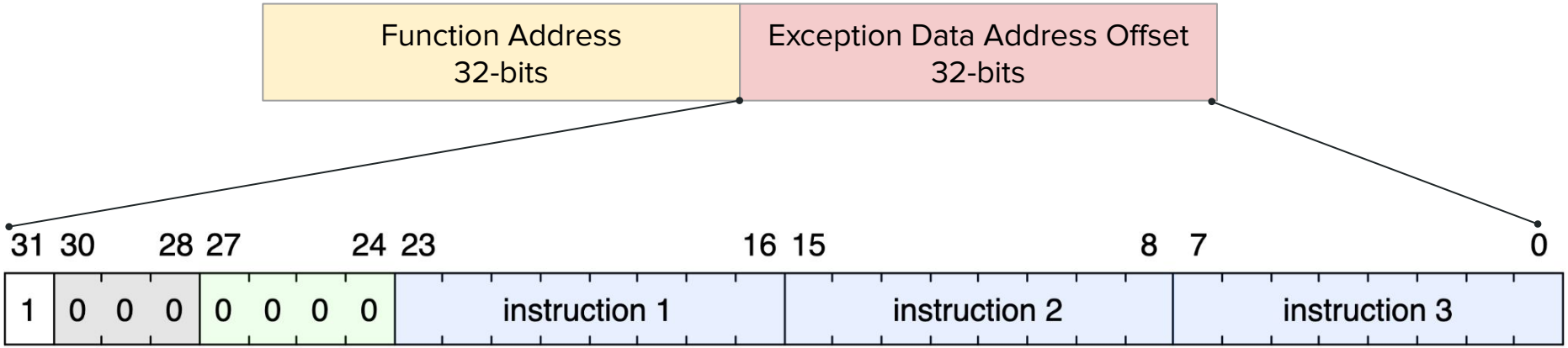
ARM Exception Index: Index Entry

```
struct arm_exception_index_t
{
 std::uint32_t prel31_address;
 union content
 {
   std::uint32_t prel31_metadata;
   std::uint32_t inlined_metadata;
 };
};
```

ARM Exception Index: 😐 "noexcept" Index Entry



ARM Exception Index: Inlined Personality Data



- For transparent 😐 functions needing only 3 unwind instructions.
- Saves memory

ARM unwind instructions

Instruction	Explanation
00xxxxxx	vsp = vsp + (xxxxxx << 2) + 4. Covers range 0x04-0x100 inclusive
01xxxxxx	vsp = vsp – (xxxxxx << 2) - 4. Covers range 0x04-0x100 inclusive
10100nnn	Pop r4-r[4+nnn]
10101nnn	Pop r4-r[4+nnn], r14 (lr)
10110010 uleb128	vsp = vsp + 0x204 + (uleb128 << 2) (for vsp increments of 0x104-0x200, use 00xxxxxx twice)
1001nnnn (nnnn != 13,15)	Set vsp = r[nnnn]
10110000	Finish
10000000 00000000	refuse to unwind / call std::terminate (for example, out of a cleanup)

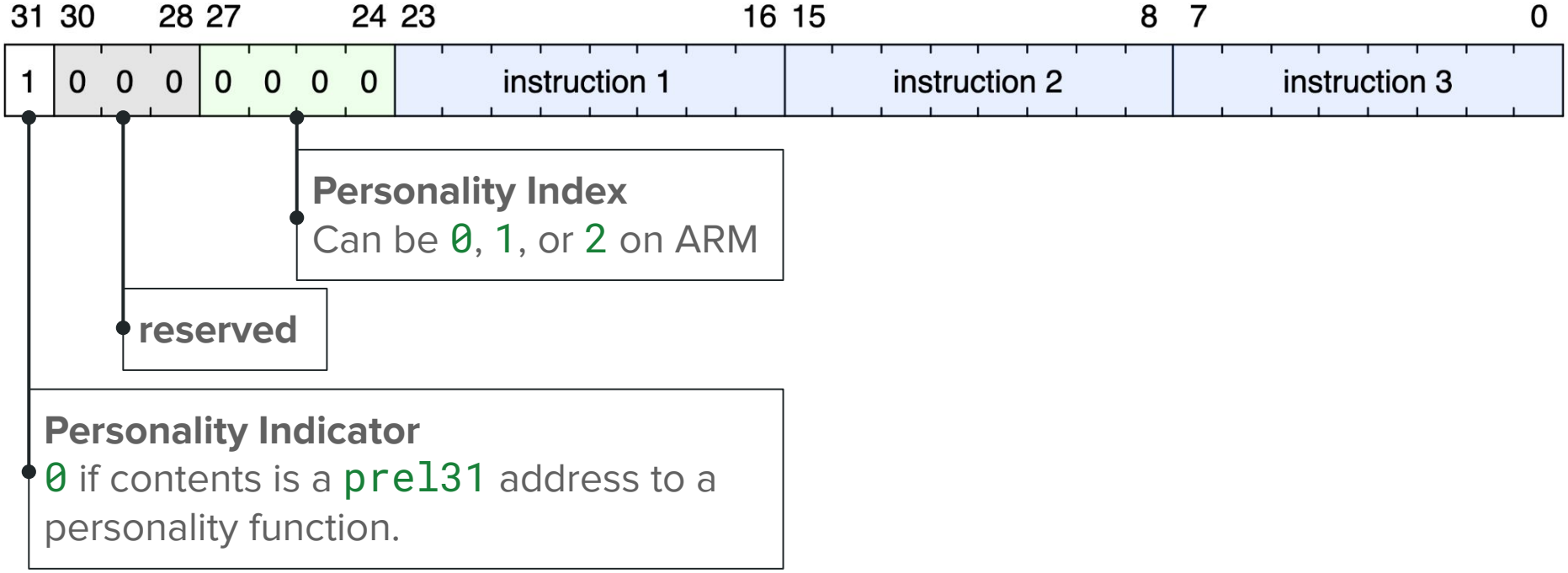
Note: Instructions are in 8-bit segments

3 bytes typically

Limit is 7 bytes

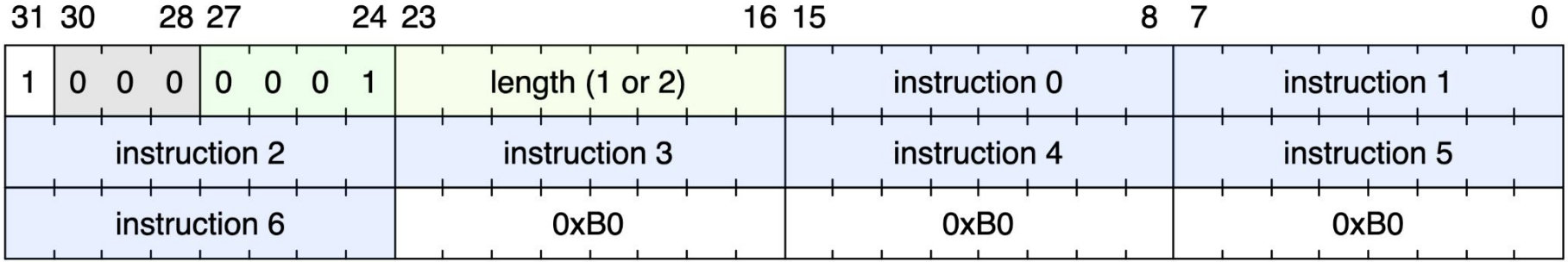
ARM Personality Data: Unwind Instructions

SU16 = Short Personality with 16-bit scopes (always 4-bytes)

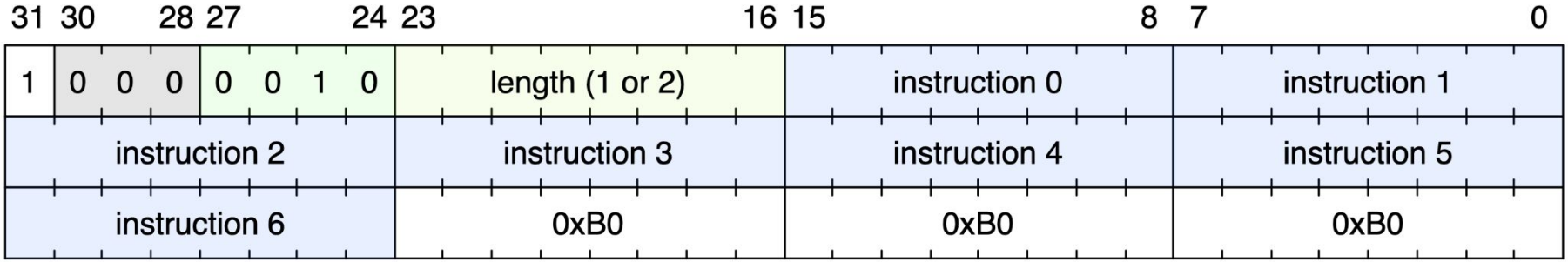


ARM Personality Data: Unwind Instructions

LU16 = Long Personality with 16-bit scopes (always 8-bytes to 12-bytes)

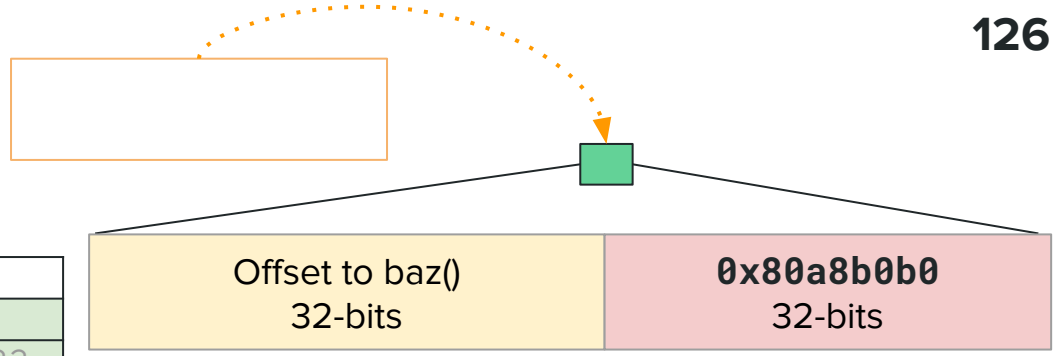


LU32 = Long Personality with 32-bit scopes (always 8-bytes to 12-bytes)

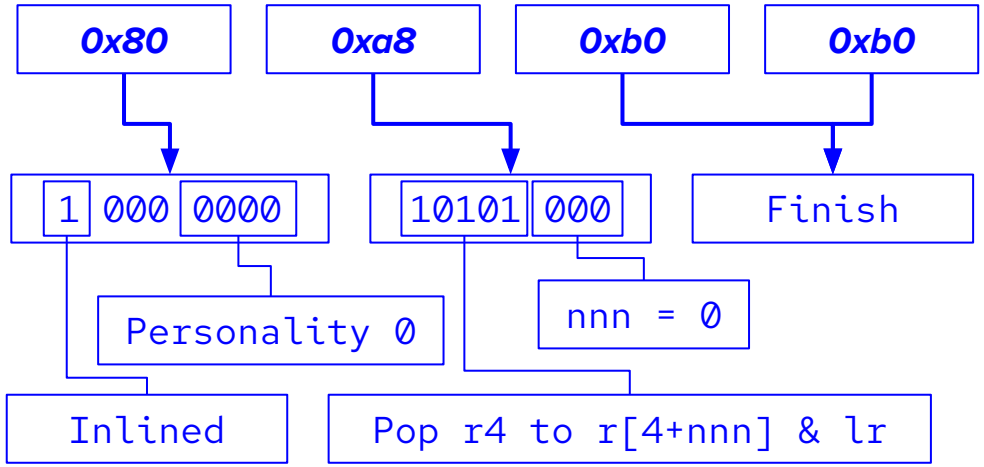


Unwinding baz()

Virtual ARM CPU		SRAM	
Reg	*Value*	*Address*	*Value*
R0	0x2000'C000	0x4000	0x????????
R1	type_info	0x3FFC	0x????????
R2	0x0	0x3FF8	0x????????
R3	0x????????	0x3FF4	0x????????
R4	0x????????	0x3FF0	0x????????
R5	0x????????	0x3FEC	caller of foo
R6	–	0x3FE8	caller r4
R7	–	0x3FE4	caller r5
R8	–	0x3FE0	foo()+10
R9	–	0x3FDC	[foo] r5
R10	–	0x3FD8	[foo] r4
R11	–	0x3FD4	obj[1/3]
R12	–	0x3FD0	obj[2/3]
R13/SP	0x3FC4	0x3FCC	obj[3/3]
R14/LR	baz()+4	0x3FC8	bar()+14
R15/PC	baz()+4	0x3FC4	[bar] r4

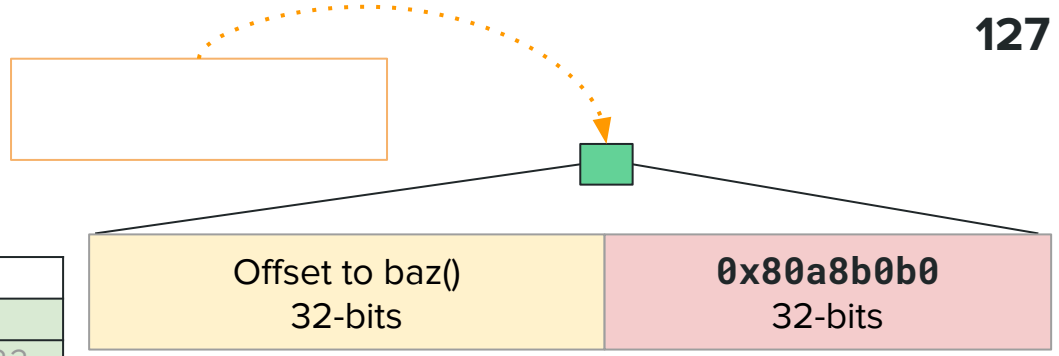


Instruction	Explanation
10101nnn	Pop r4-r[4+nnn], r14 (lr)



Unwinding baz()

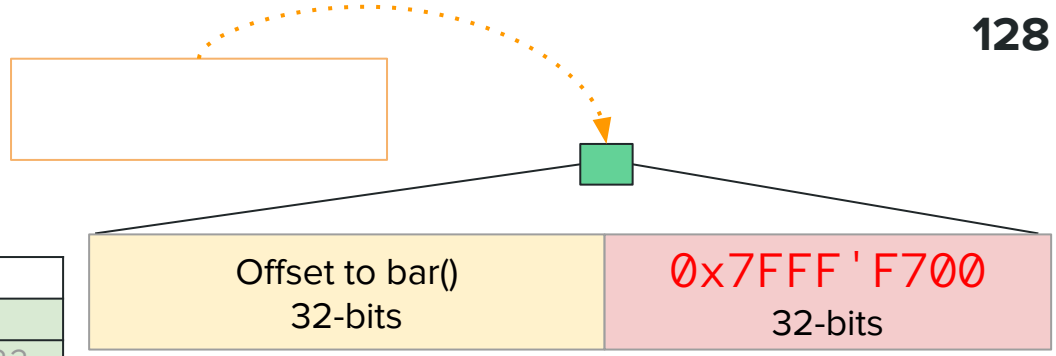
Virtual ARM CPU		SRAM	
Reg	*Value*	*Address*	*Value*
R0	0x2000'C000	0x4000	0x????????
R1	type_info	0x3FFC	0x????????
R2	0x0	0x3FF8	0x????????
R3	0x????????	0x3FF4	0x????????
R4	[bar] r4	0x3FF0	0x????????
R5	0x????????	0x3FEC	caller of foo
R6	–	0x3FE8	caller r4
R7	–	0x3FE4	caller r5
R8	–	0x3FE0	foo()+10
R9	–	0x3FDC	[foo] r5
R10	–	0x3FD8	[foo] r4
R11	–	0x3FD4	obj[1/3]
R12	–	0x3FD0	obj[2/3]
R13/SP	0x3FCC	0x3FCC	obj[3/3]
R14/LR	bar()+14	0x3FC8	bar()+14
R15/PC	bar()+14	0x3FC4	[bar] r4



Instruction	Explanation
10101nnn	Pop r4-r[4+nnn], r14 (lr)

bar entry

Unwinding bar()



Virtual ARM CPU		SRAM	
Reg	*Value*	*Address*	*Value*
R0	0x2000'C000	0x4000	0x????????
R1	type_info	0x3FFC	0x????????
R2	0x0	0x3FF8	0x????????
R3	0x????????	0x3FF4	0x????????
R4	[bar] r4	0x3FF0	0x????????
R5	0x????????	0x3FEC	caller of foo
R6	–	0x3FE8	caller r4
R7	–	0x3FE4	caller r5
R8	–	0x3FE0	foo()+10
R9	–	0x3FDC	[foo] r5
R10	–	0x3FD8	[foo] r4
R11	–	0x3FD4	obj[1/3]
R12	–	0x3FD0	obj[2/3]
R13/SP	0x3FCC	0x3FCC	obj[3/3]
R14/LR	bar()+14	0x3FC8	bar()+14
R15/PC	bar()+14	0x3FC4	[bar] r4

Note: Data isn't inlined because Bar is

Calling bar()'s Destructors...

We are here

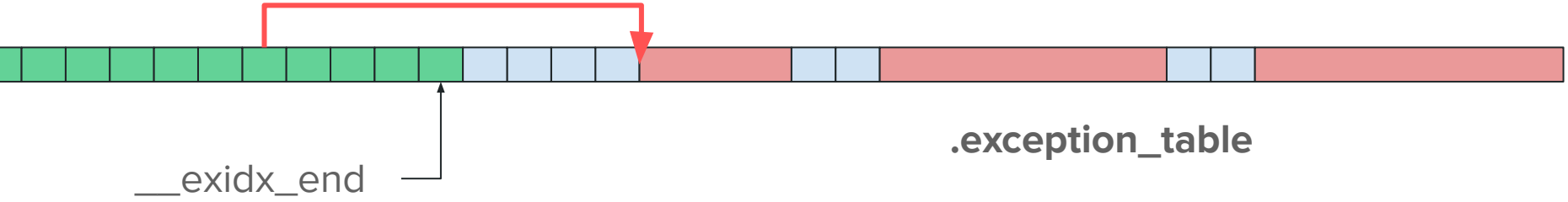
```
struct error {};

void foo() {
 try {
   bar();
 } catch (error const& p_error) {
   // end up here...
 }
}

void bar() {
 destructible_t obj;
 baz();
} // ~destructible_t(&my_object)

void baz() { throw error{}; }
```

ARM Exception Table

Note: Exception table is 32-bit aligned

Function Address 32-bits	Exception Data Address Offset 0x7FFF'F700

=-2304

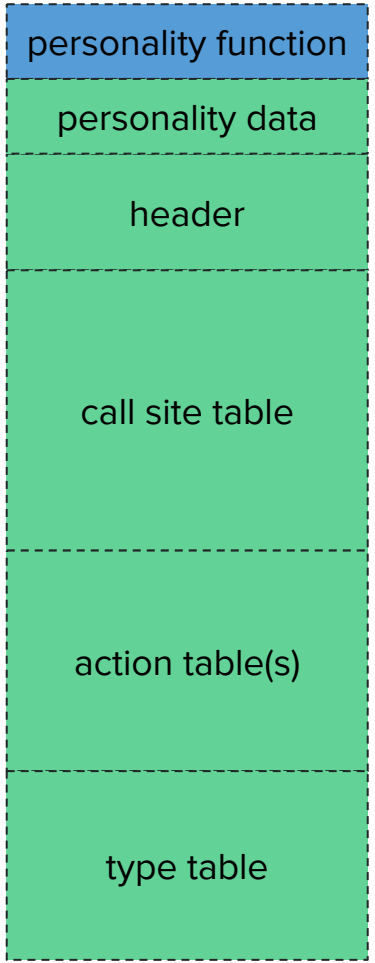


LU16/32 personality

GCC LSDA

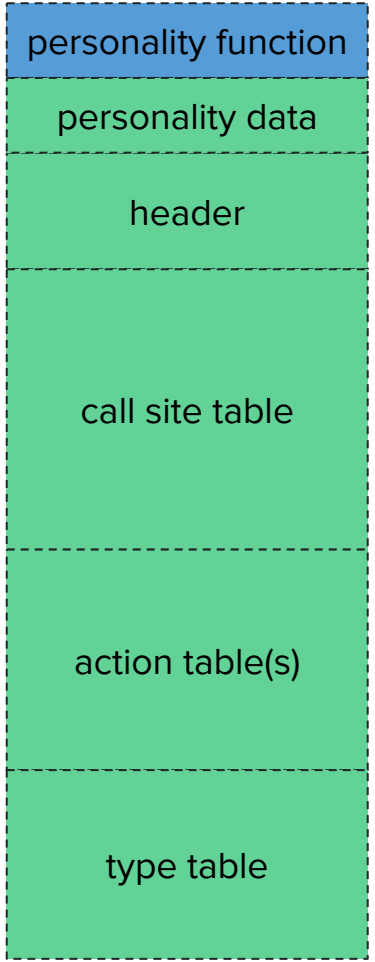
What is a LSDA?

- L.S.D.A. = **L**anguage **S**pecific **D**ata **A**rea
- LSDA is included in the Itanium exception ABIs in order to support other languages such as **Java**, **C#**, etc.
- Provides a means to allows exceptions to propagate from language to language.



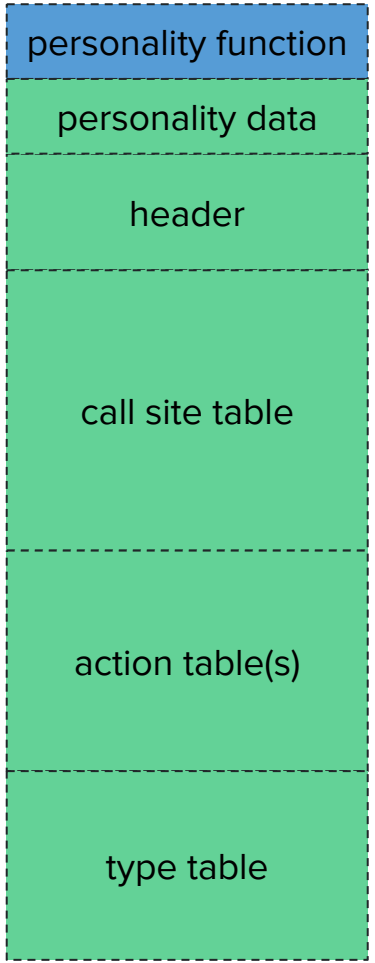
What is a LSDA?

- L.S.D.A. = **L**anguage **S**pecific **D**ata **A**rea
- LSDA is included in the Itanium exception ABIs in order to support other languages such as **Java**, **C#**, etc.
- Provides a means to allows exceptions to propagate from language to language.
- GCC leverages this to make platform agnostic exception data



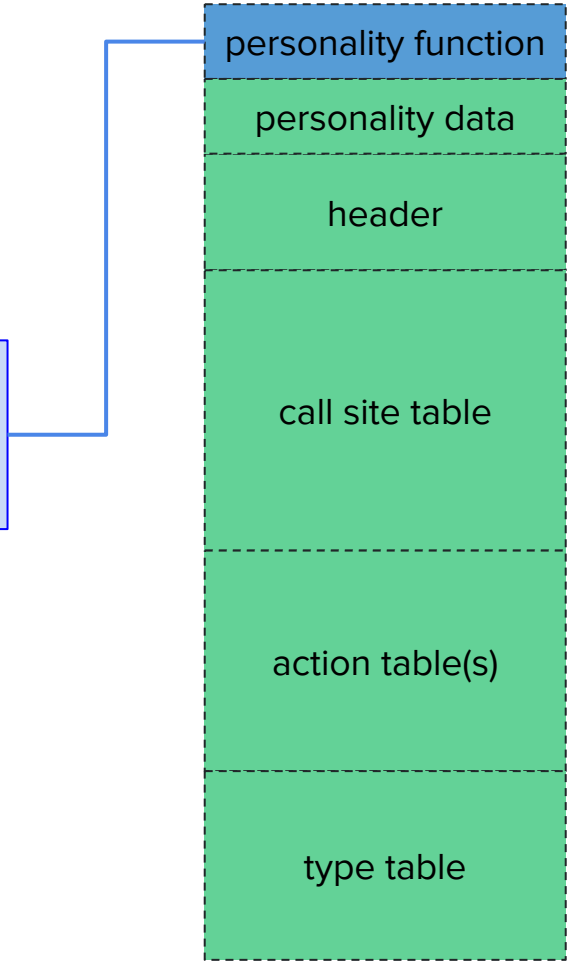
GCC LSDA: Sections

- **Personality Function:** Function that can handle this data
- **Personality Data:** Unwind information
- **Header**: Describes the encoding of the tables below
- **Call Site Table:** Contains important frame scope areas
- **Action Table:** Provides catch order
- **Type table:** Holds the caught types in the function



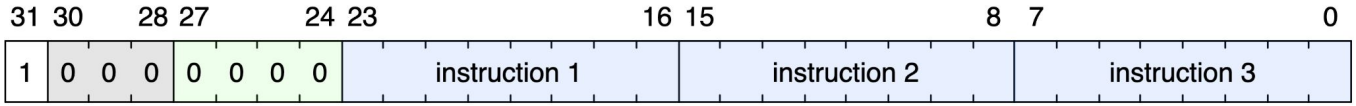
GCC LSDA: Personality Function

31-bit offset to **`gcc_personality_v0`**
`0x7FFF'F320`



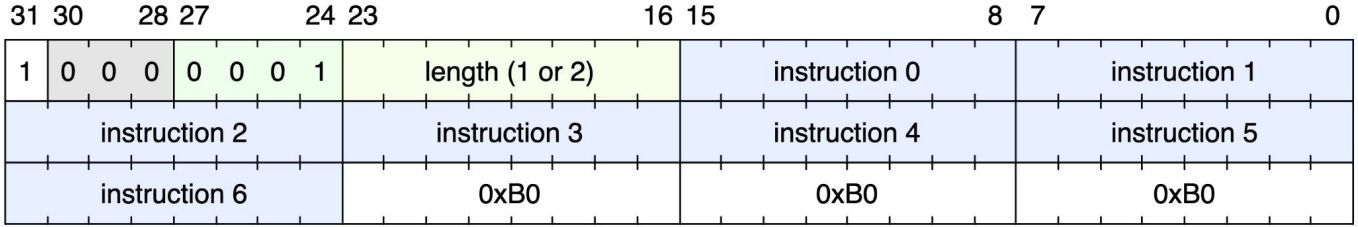
GCC LSDA: Personality Data

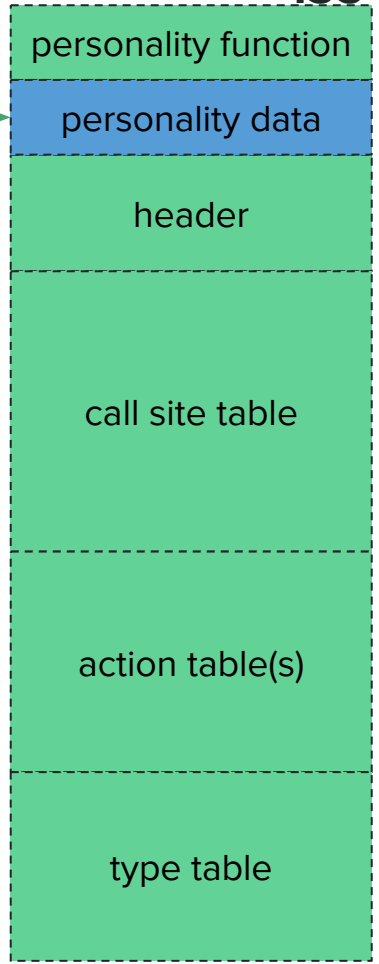
SU16 = Short Personality



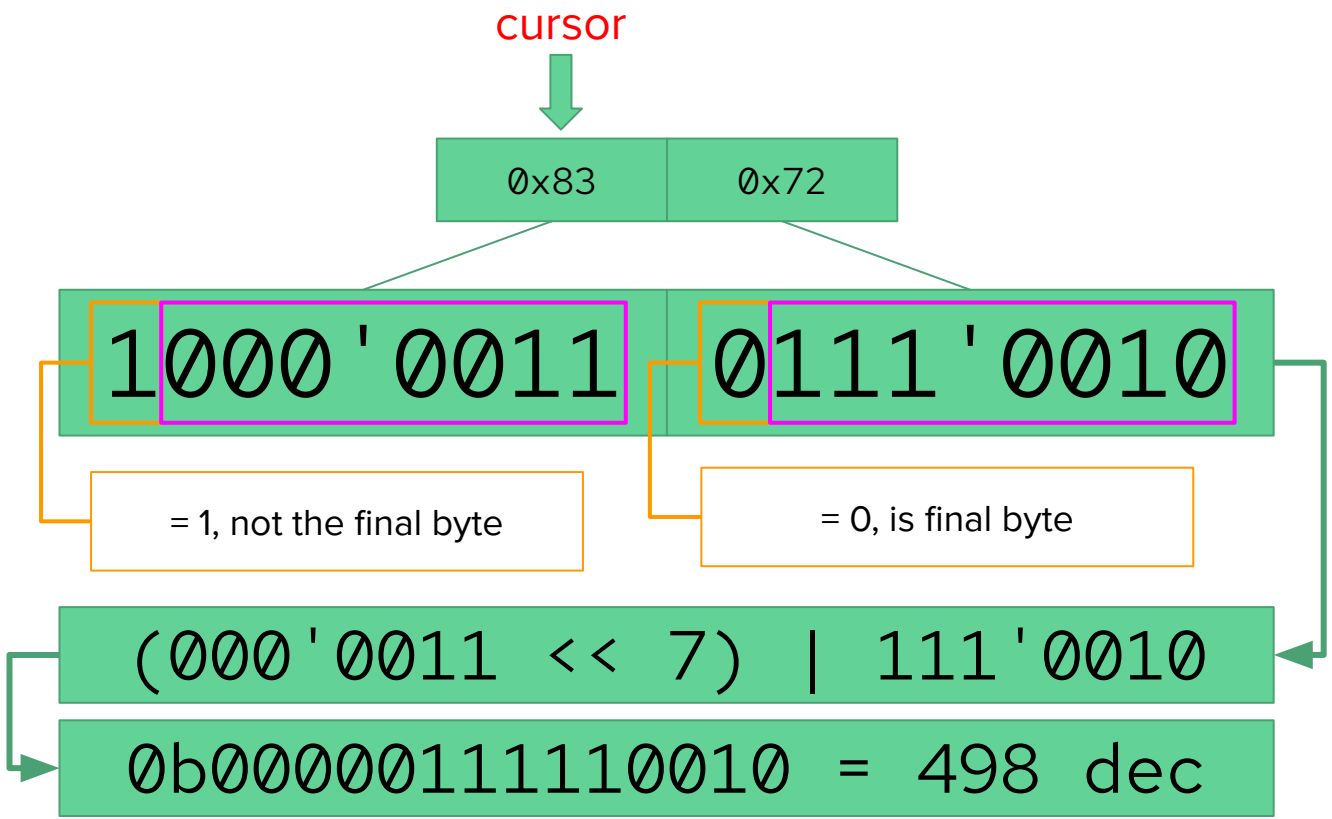
OR

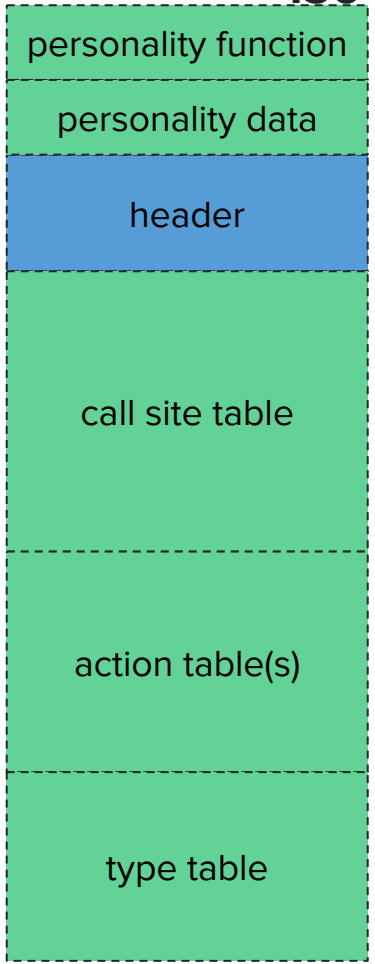
LU16/32 = Long Personality



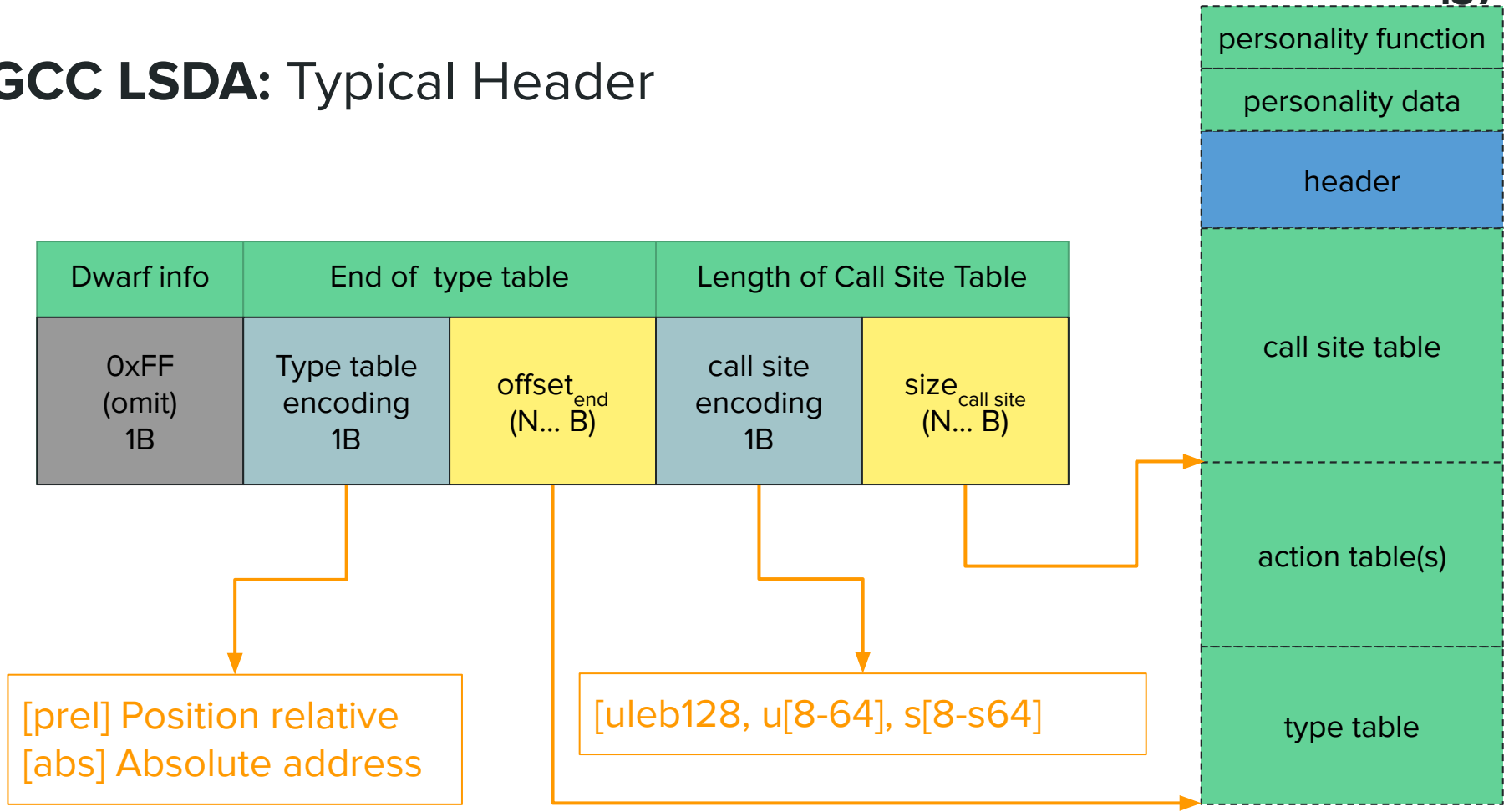


GCC LSDA: LEB128 variable length encoding

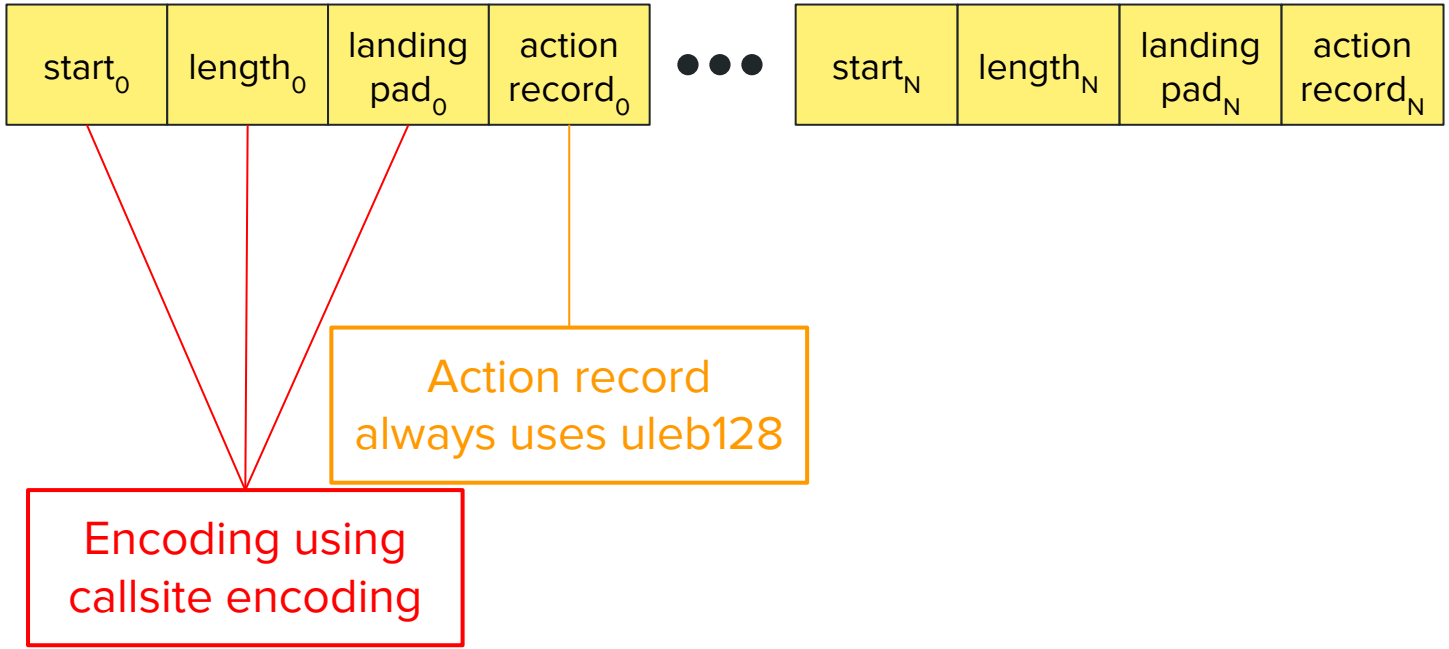


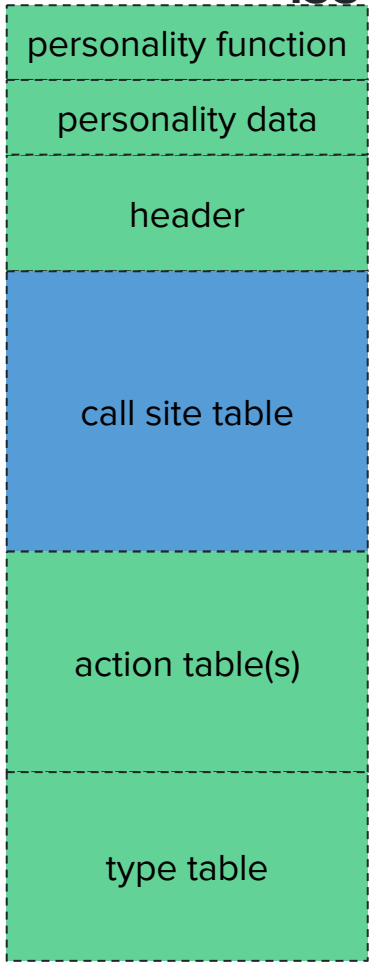


GCC LSDA: Typical Header



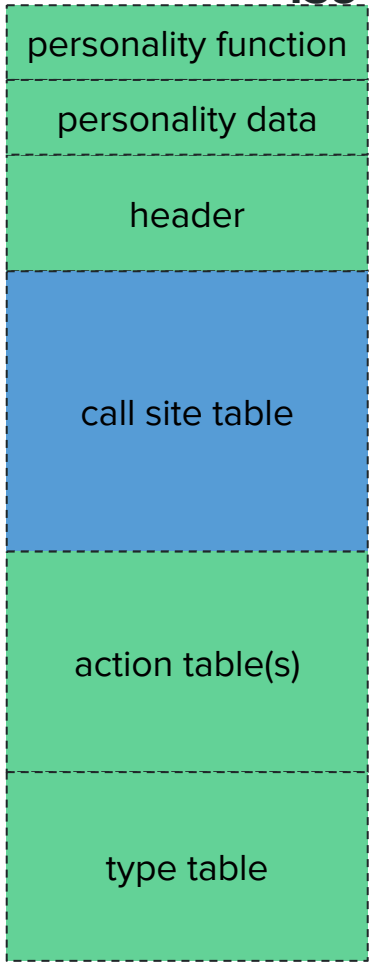
GCC LSDA: Call Site Table





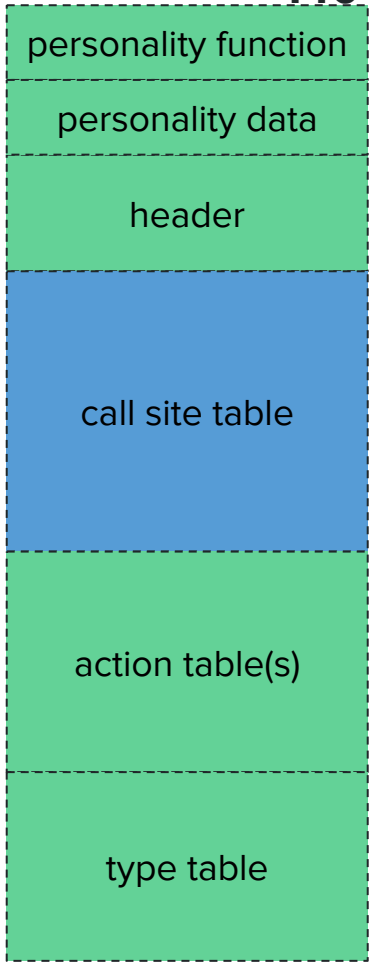
GCC LSDA: Decoding the Call Site Table

```
do {
 auto start = read_encoded_data(&lsda_cursor, call_site_format);
 auto length = read_encoded_data(&lsda_cursor, call_site_format);
 landing_pad = read_encoded_data(&lsda_cursor, call_site_format);
 action = read_uleb128(&lsda_cursor);

 if (start <= rel_pc && rel_pc < start + length) {
   if (landing_pad == 0) {
     p_exception_object.cache.state(runtime_state::unwind_frame);
     return;
   }
   break;
 }
} while (lsda_cursor < call_site_end);
```



GCC LSDA: Executing Destructors

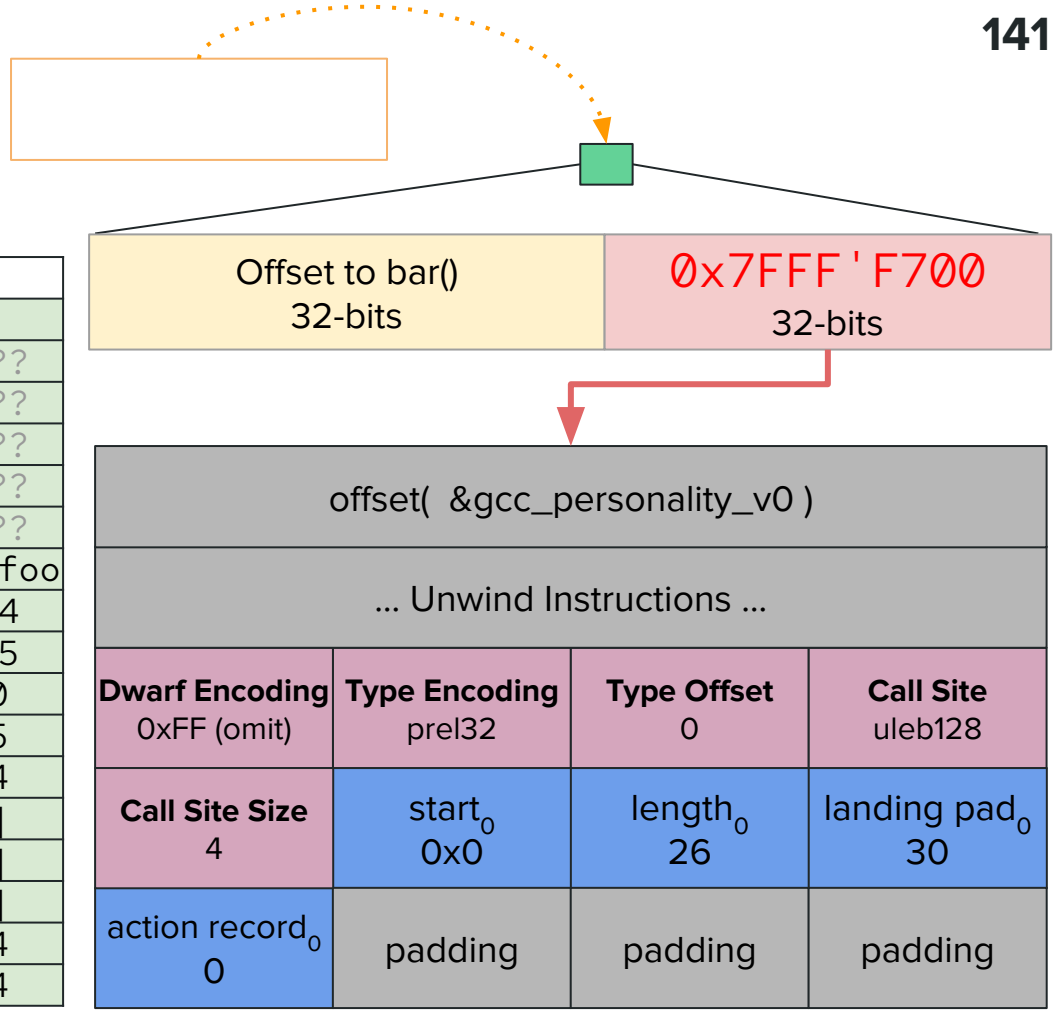
```
if (type_table_end < call_site_end ||
    action == 0) {
  // LSB must be set to 1 to jump to an address
  auto const destination = (entry.function() + landing_pad) | 0b1;
  // Set PC to the cleanup destination
  virtual_cpu.pc = destination;
  // Install CPU state
  restore_cpu_core(virtual_cpu);
}
```



bar entry

Entering bar()

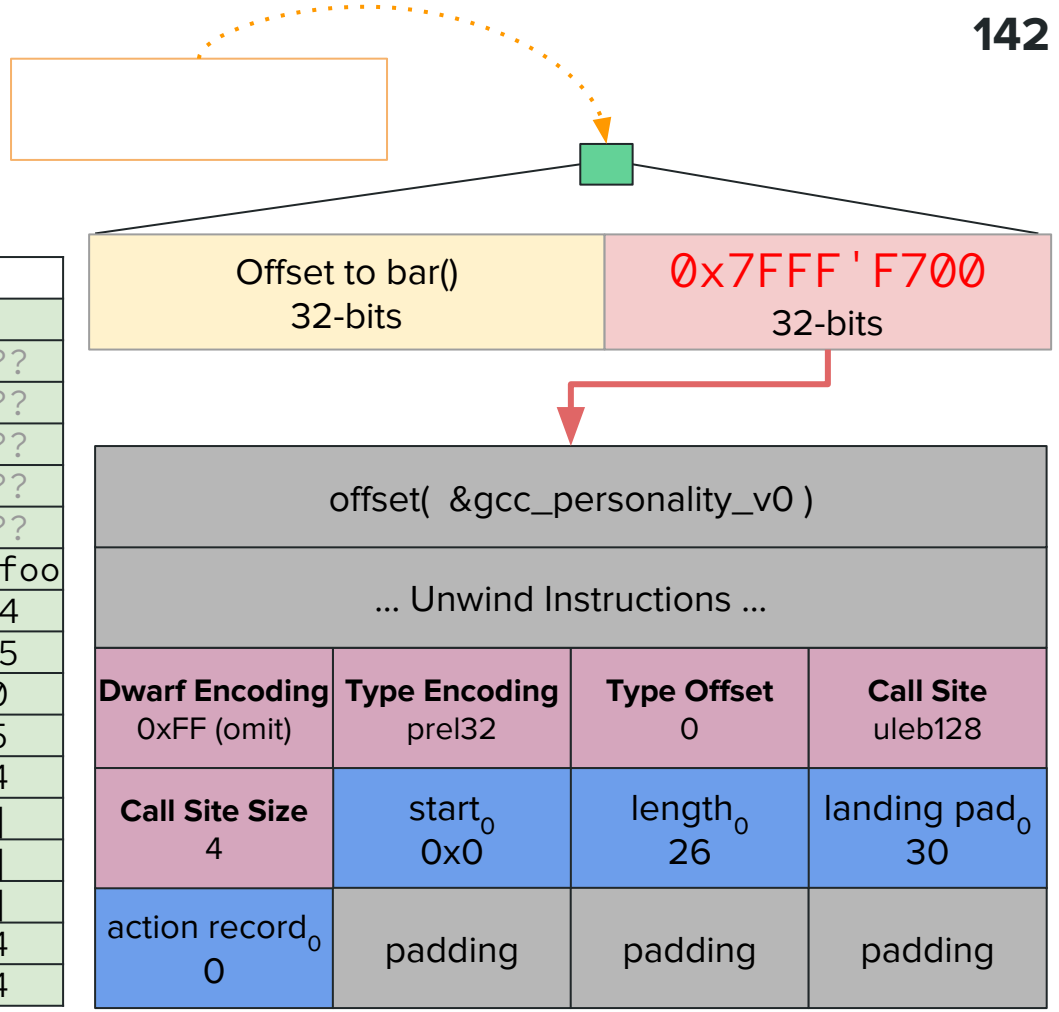
Virtual ARM CPU		SRAM	
Reg	*Value*	*Address*	*Value*
R0	0x2000'C000	0x4000	0x????????
R1	type_info	0x3FFC	0x????????
R2	0x0	0x3FF8	0x????????
R3	0x????????	0x3FF4	0x????????
R4	[bar] r4	0x3FF0	0x????????
R5	0x????????	0x3FEC	caller of foo
R6	–	0x3FE8	caller r4
R7	–	0x3FE4	caller r5
R8	–	0x3FE0	foo()+10
R9	–	0x3FDC	[foo] r5
R10	–	0x3FD8	[foo] r4
R11	–	0x3FD4	obj[1/3]
R12	–	0x3FD0	obj[2/3]
R13/SP	0x3FCC	0x3FCC	obj[3/3]
R14/LR	bar()+14	0x3FC8	bar()+14
R15/PC	bar()+14	0x3FC4	[bar] r4



bar entry

Entering bar()

Virtual ARM CPU		SRAM	
Reg	*Value*	*Address*	*Value*
R0	0x2000'C000	0x4000	0x????????
R1	type_info	0x3FFC	0x????????
R2	0x0	0x3FF8	0x????????
R3	0x????????	0x3FF4	0x????????
R4	[bar] r4	0x3FF0	0x????????
R5	0x????????	0x3FEC	caller of foo
R6	–	0x3FE8	caller r4
R7	–	0x3FE4	caller r5
R8	–	0x3FE0	foo()+10
R9	–	0x3FDC	[foo] r5
R10	–	0x3FD8	[foo] r4
R11	–	0x3FD4	obj[1/3]
R12	–	0x3FD0	obj[2/3]
R13/SP	0x3FCC	0x3FCC	obj[3/3]
R14/LR	bar()+30	0x3FC8	bar()+14
R15/PC	bar()+30	0x3FC4	[bar] r4



Executing bar()'s Destructor

Virtual ARM CPU		SRAM	
Reg	*Value*	*Address*	*Value*
R0	0x2000'C000	0x4000	0x????????
R1	type_info	0x3FFC	0x????????
R2	0x0	0x3FF8	0x????????
R3	0x????????	0x3FF4	0x????????
R4	[bar] r4	0x3FF0	0x????????
R5	0x????????	0x3FEC	caller of foo
R6	–	0x3FE8	caller r4
R7	–	0x3FE4	caller r5
R8	–	0x3FE0	foo()+10
R9	–	0x3FDC	[foo] r5
R10	–	0x3FD8	[foo] r4
R11	–	0x3FD4	obj[1/3]
R12	–	0x3FD0	obj[2/3]
R13/SP	0x3FCC	0x3FCC	obj[3/3]
R14/LR	bar()+30	0x3FC8	bar()+14
R15/PC	bar()+30	0x3FC4	[bar] r4

```
000088b4 <bar()>:
    88b4: b500          push {lr}
    88b6: 4b09          ldr  r3, [pc, #36]  ; (88dc <bar()+0x28>)
    88b8: b083          sub  sp, #12
    88ba: 791b          ldrb r3, [r3, #4]
    88bc: 220f          movs r2, #15
    88be: 9201          str  r2, [sp, #4]
    88c0: b92b          cbnz r3, 88ce <bar()+0x1a>
    88c2: a801          add  r0, sp, #4
    88c4: f000 f832     bl   892c <destructible_t::~destructible_t()>
    88c8: b003          add  sp, #12
    88ca: f85d fb04     ldr.w     pc, [sp], #4
    88ce: f7f7 fc09     bl   e4 <baz() [clone .part.0]>
    88d2: a801          add  r0, sp, #4
    88d4: f000 f82a     bl   892c <destructible_t::~destructible_t()>
    88d8: f000 fcc6     bl   9268 <__cxa_end_cleanup>
    88dc: 1000065c  .word     0x1000065c
```

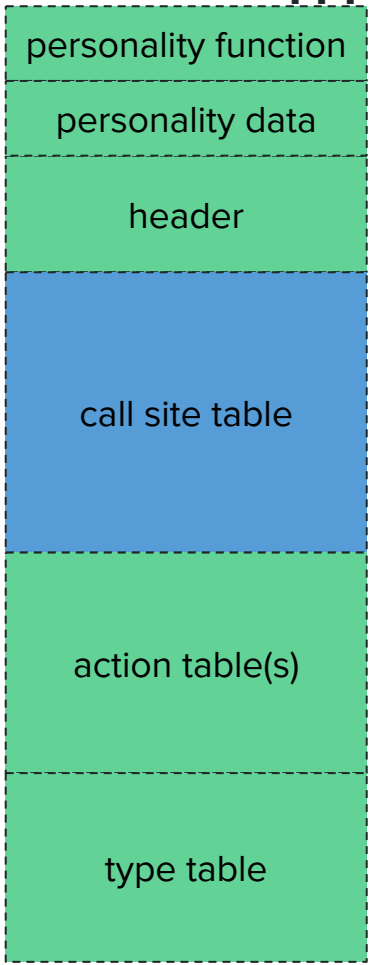
GCC LSDA: Executing Destructors

Load address of object on stack into R0

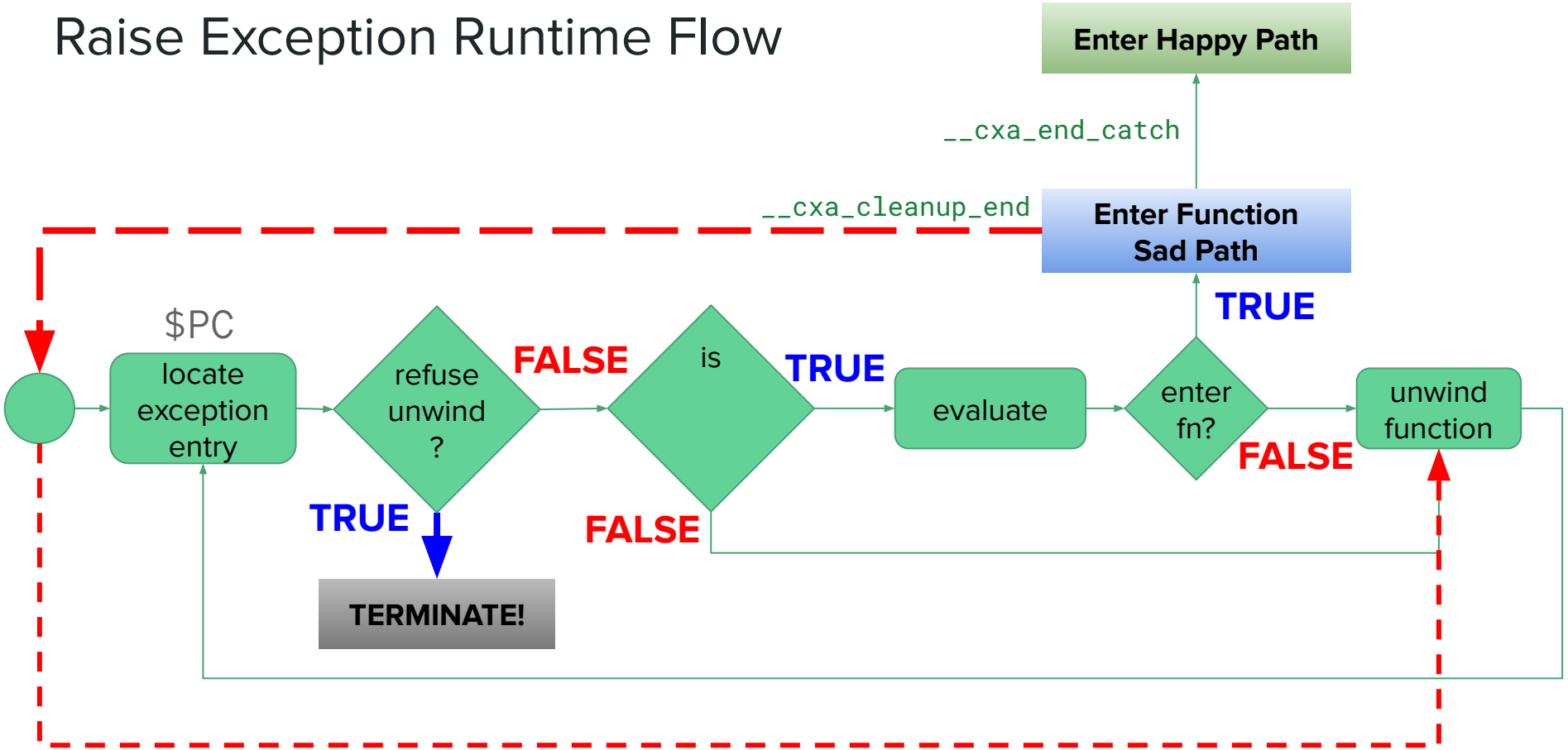
```
000088b4 <bar()>:
   ; happy path instructions
   88d2: a801        add   r0, sp, #4
   88d4: f000 f82a   bl    892c <destructible_t::~destructible_t()>
   88d8: f000 fcc6   bl    9268 <__cxa_end_cleanup>
```

Call Destructor

1. Recover exception object
2. Set state to "unwind frame"
3. raise_exception()

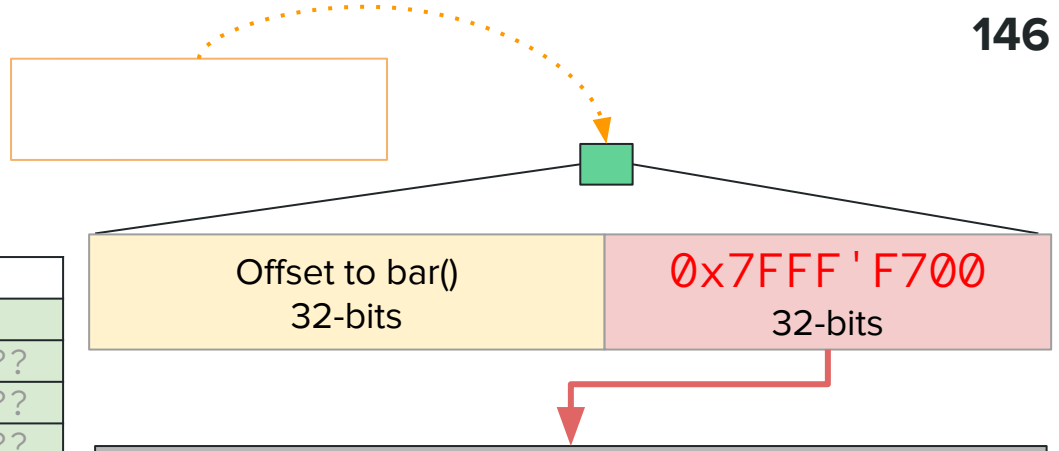


Raise Exception Runtime Flow



bar entry

After Bar's Destructors



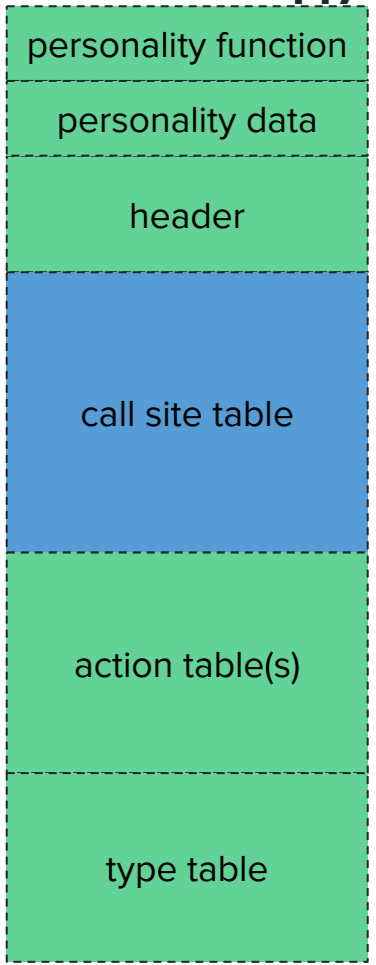
Virtual ARM CPU		SRAM	
Reg	*Value*	*Address*	*Value*
R0	0x2000'C000	0x4000	0x????????
R1	type_info	0x3FFC	0x????????
R2	0x0	0x3FF8	0x????????
R3	0x????????	0x3FF4	0x????????
R4	[bar] r4	0x3FF0	0x????????
R5	0x????????	0x3FEC	caller of foo
R6	–	0x3FE8	caller r4
R7	–	0x3FE4	caller r5
R8	–	0x3FE0	foo()+10
R9	–	0x3FDC	[foo] r5
R10	–	0x3FD8	[foo] r4
R11	–	0x3FD4	obj[1/3]
R12	–	0x3FD0	obj[2/3]
R13/SP	0x3FCC	0x3FCC	obj[3/3]
R14/LR	bar()+30	0x3FC8	0x????????
R15/PC	bar()+30	0x3FC4	0x????????

GCC LSDA: Executing Destructors

Call Site 0 Landing Pad

```
000088b4 <bar()>:
   ; happy path instructions
   88da: a801         add   r0, sp, #4
   88dc: f000 f834    bl    8948 <destructible_t::~destructible_t()>
   88e0: 4668         mov   r0, sp
   88e2: f000 f831    bl    8948 <destructible_t::~destructible_t()>
   88e6: f000 fccd    bl    9284 <__cxa_end_cleanup>
```

Call Site 1 Landing Pad



Unwinding bar()

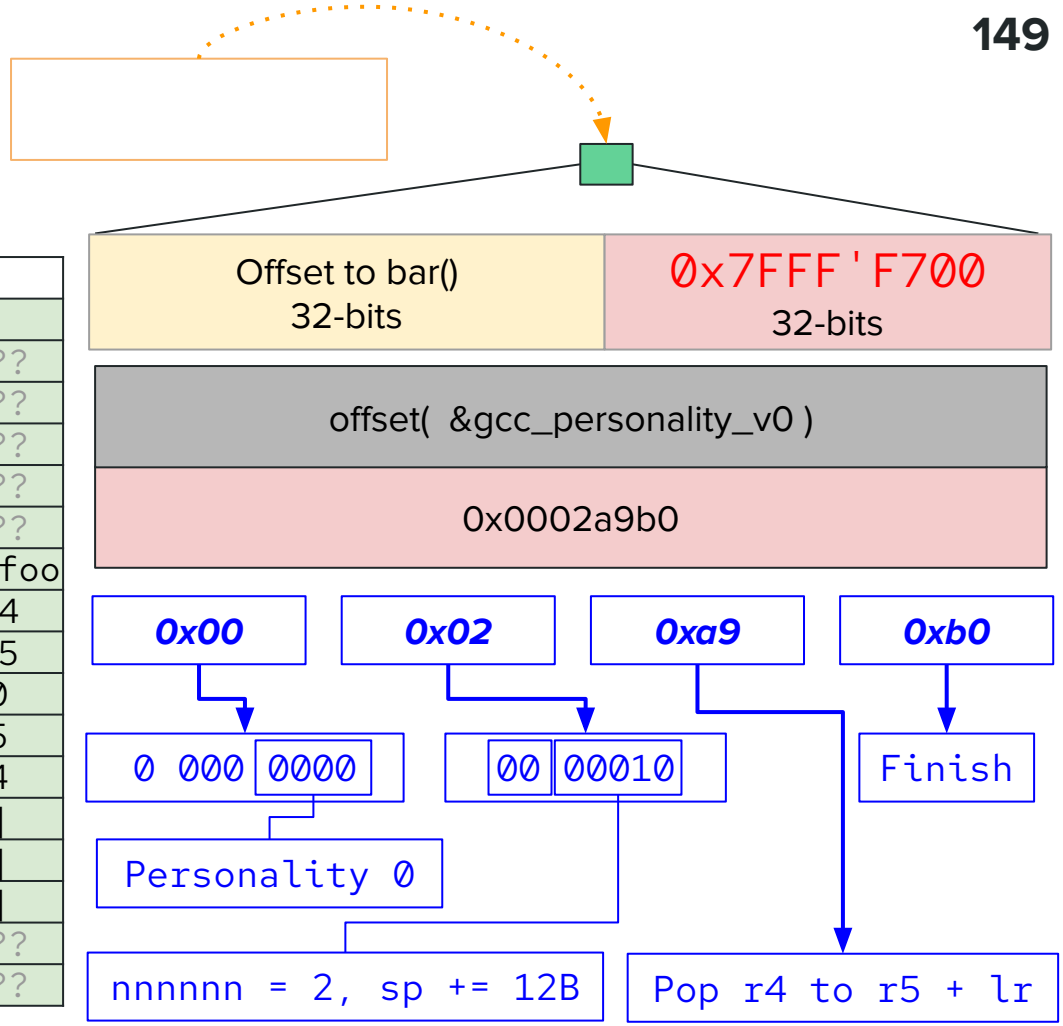
We are here

```
struct error {};

void foo() {
 try {
   bar();
 } catch (error const& p_error) {
   // end up here...
 }
}

void bar() {
 destructible_t obj;
 baz();
} // ~destructible_t(&my_object)

void baz() { throw error{}; }
```

bar entry

Unwinding bar()

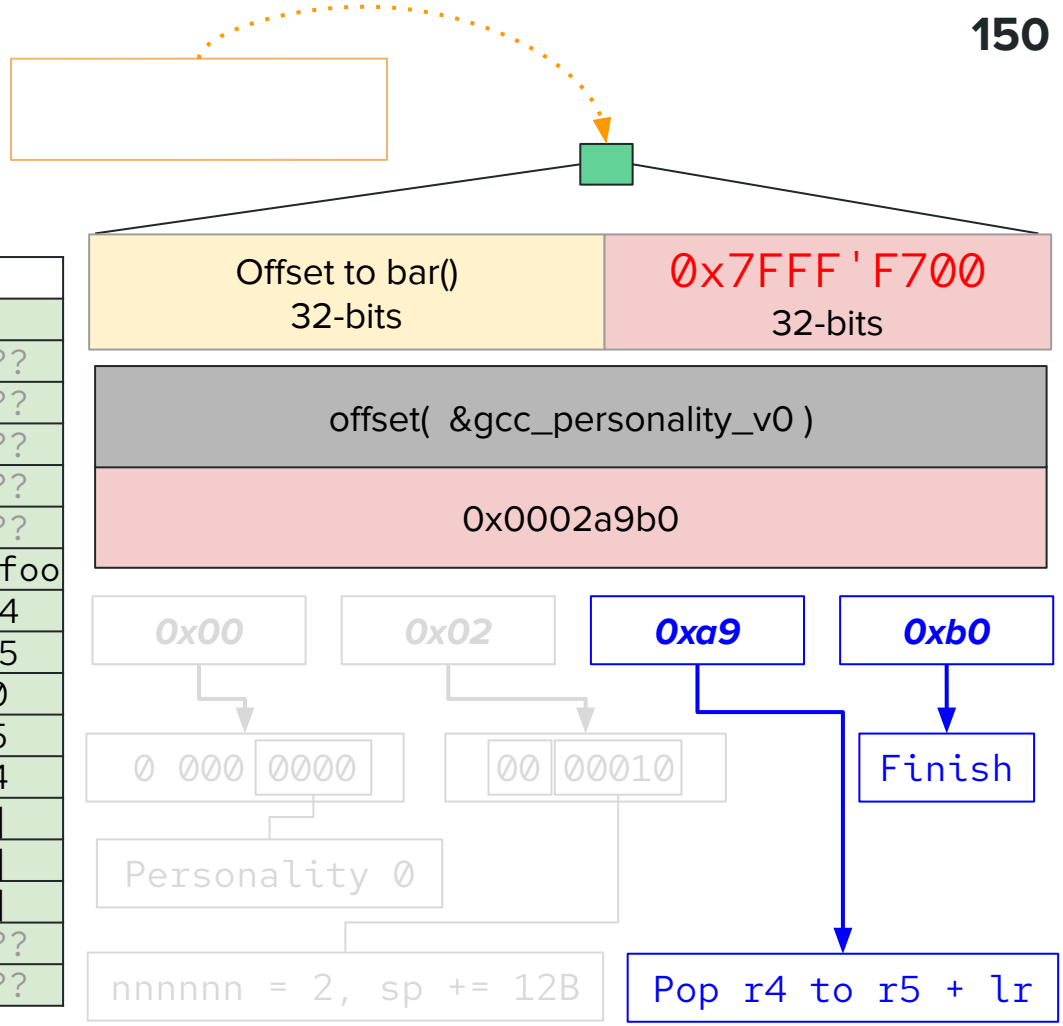
Virtual ARM CPU		SRAM	
Reg	*Value*	*Address*	*Value*
R0	0x2000'C000	0x4000	0x????????
R1	type_info	0x3FFC	0x????????
R2	0x0	0x3FF8	0x????????
R3	0x????????	0x3FF4	0x????????
R4	[bar] r4	0x3FF0	0x????????
R5	0x????????	0x3FEC	caller of foo
R6	–	0x3FE8	caller r4
R7	–	0x3FE4	caller r5
R8	–	0x3FE0	foo()+10
R9	–	0x3FDC	[foo] r5
R10	–	0x3FD8	[foo] r4
R11	–	0x3FD4	obj[1/3]
R12	–	0x3FD0	obj[2/3]
R13/SP	0x3FCC	0x3FCC	obj[3/3]
R14/LR	bar()+14	0x3FC8	0x????????
R15/PC	bar()+14	0x3FC4	0x????????



bar entry

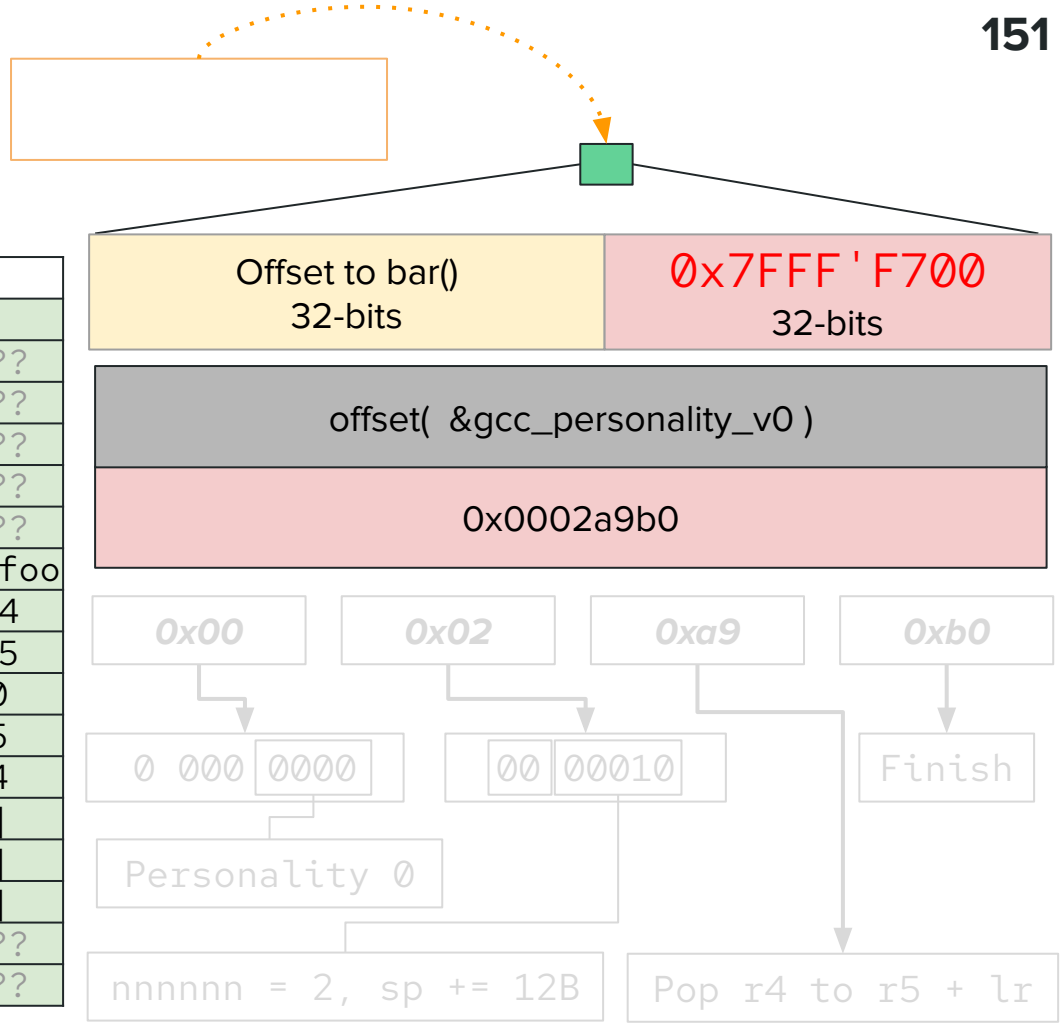
Unwinding bar()

Virtual ARM CPU		SRAM	
Reg	*Value*	*Address*	*Value*
R0	0x2000'C000	0x4000	0x????????
R1	type_info	0x3FFC	0x????????
R2	0x0	0x3FF8	0x????????
R3	0x????????	0x3FF4	0x????????
R4	[bar] r4	0x3FF0	0x????????
R5	0x????????	0x3FEC	caller of foo
R6	–	0x3FE8	caller r4
R7	–	0x3FE4	caller r5
R8	–	0x3FE0	foo()+10
R9	–	0x3FDC	[foo] r5
R10	–	0x3FD8	[foo] r4
R11	–	0x3FD4	obj[1/3]
R12	–	0x3FD0	obj[2/3]
R13/SP	0x3FD8	0x3FCC	obj[3/3]
R14/LR	bar()+14	0x3FC8	0x????????
R15/PC	bar()+14	0x3FC4	0x????????



Unwinding bar()

Virtual ARM CPU		SRAM	
Reg	*Value*	*Address*	*Value*
R0	0x2000'C000	0x4000	0x????????
R1	type_info	0x3FFC	0x????????
R2	0x0	0x3FF8	0x????????
R3	0x????????	0x3FF4	0x????????
R4	[foo] r4	0x3FF0	0x????????
R5	[foo] r5	0x3FEC	caller of foo
R6	–	0x3FE8	caller r4
R7	–	0x3FE4	caller r5
R8	–	0x3FE0	foo()+10
R9	–	0x3FDC	[foo] r5
R10	–	0x3FD8	[foo] r4
R11	–	0x3FD4	obj[1/3]
R12	–	0x3FD0	obj[2/3]
R13/SP	0x3FE0	0x3FCC	obj[3/3]
R14/LR	foo()+10	0x3FC8	0x????????
R15/PC	foo()+10	0x3FC4	0x????????

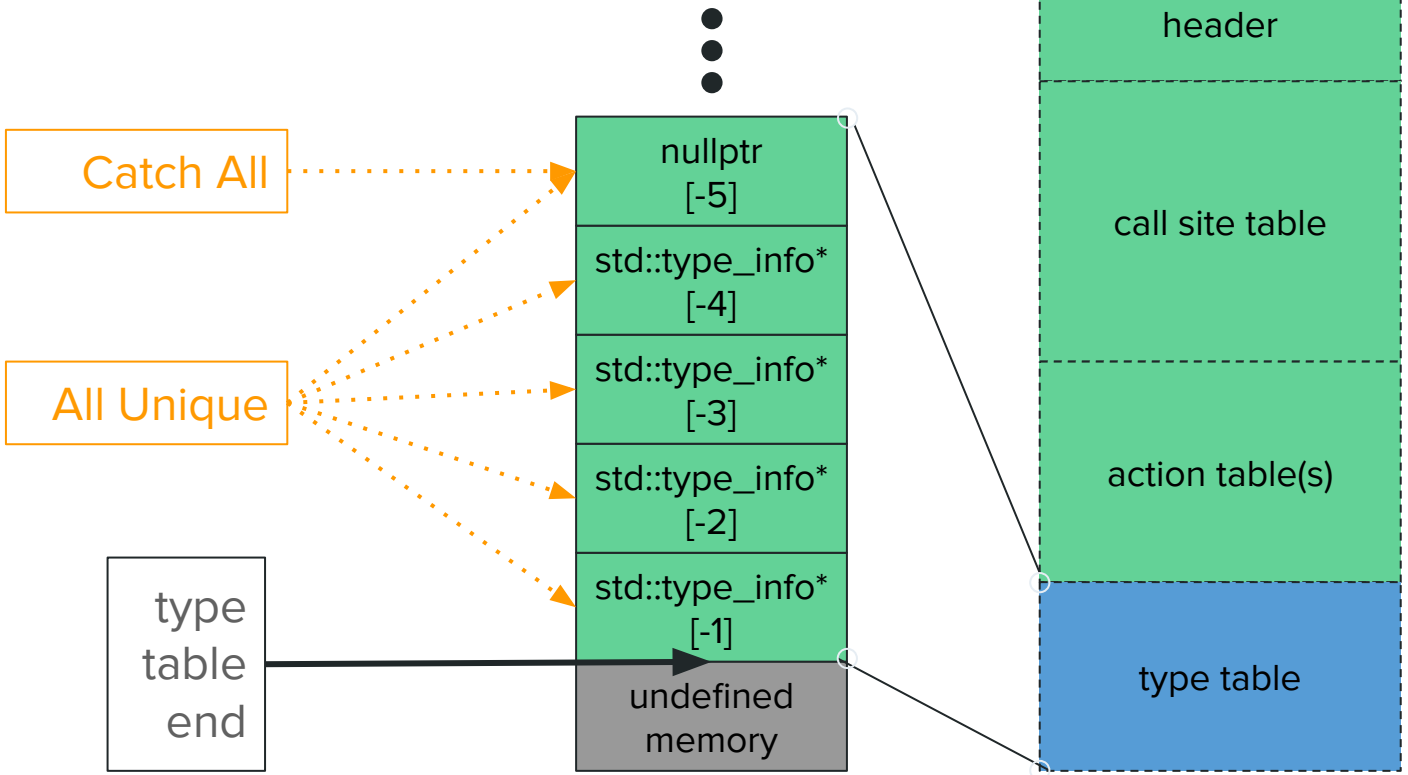


Entering foo()'s Catch block...

We are here

```
struct error {};

void foo() {
 try {
   bar();
 } catch (error const& p_error) {
   // end up here...
 }
}

void bar() {
 destructible_t obj;
 baz();
} // ~destructible_t(&my_object)

void baz() { throw error{}; }
```

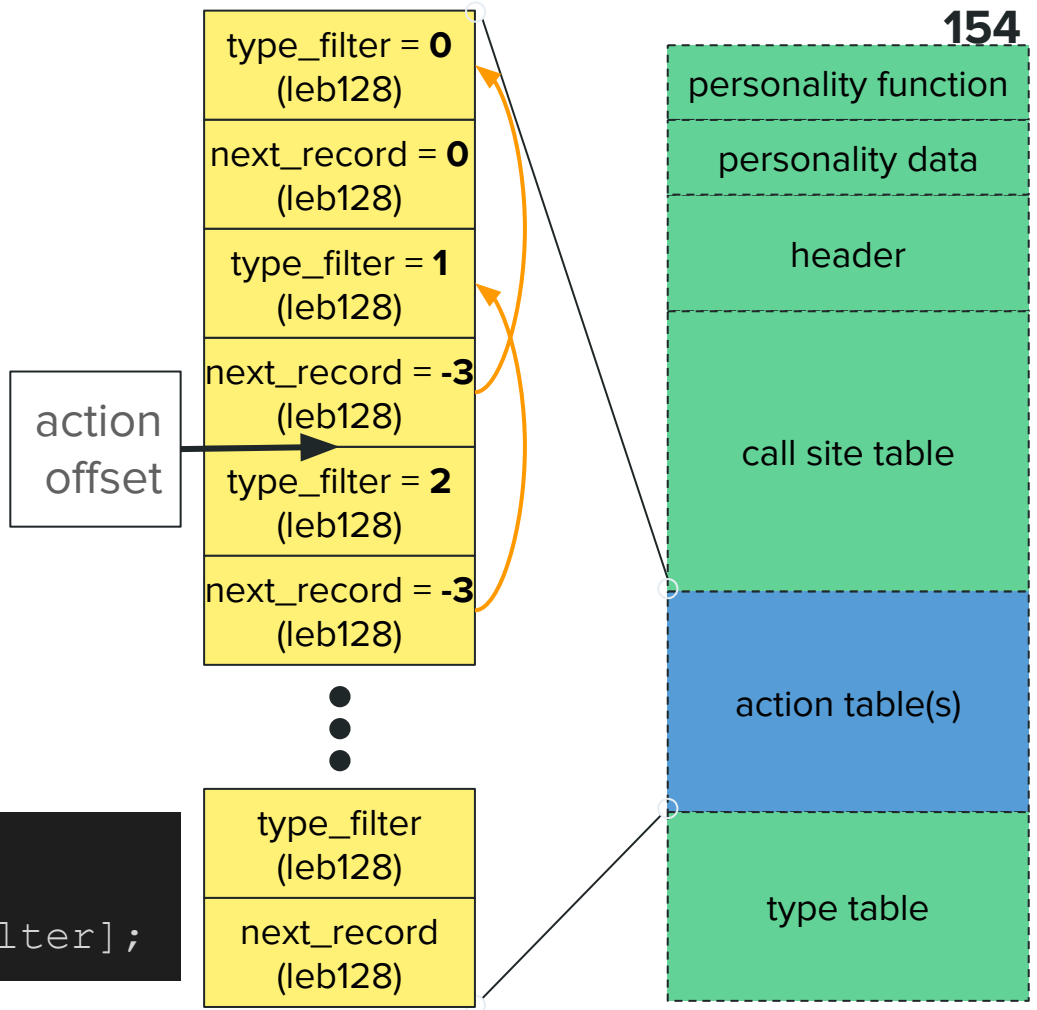
GCC LSDA: Type Table



GCC LSDA: Action Table

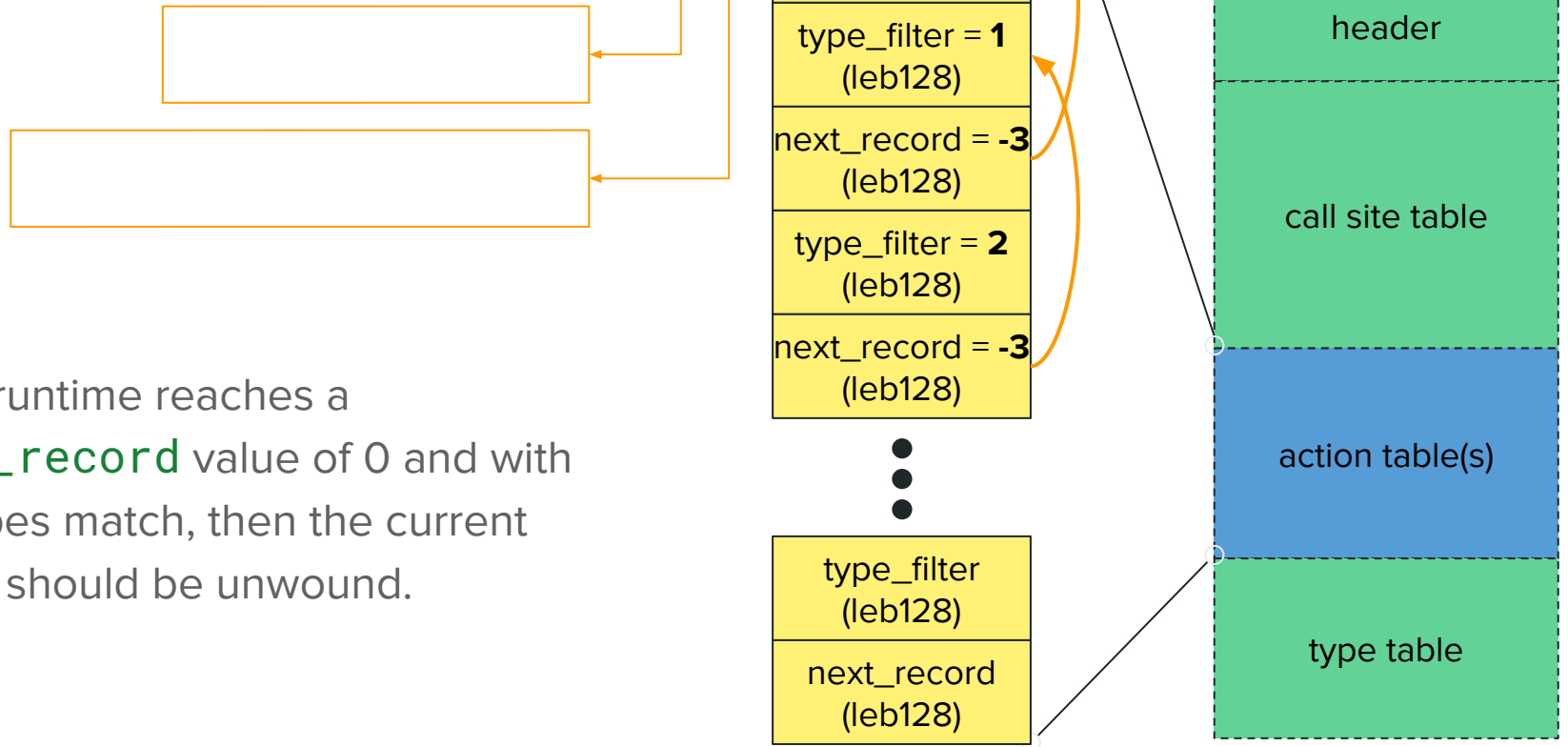
- The action table represents
  - The sequence of catch blocks attached to a try scope
  - provides the mapping between catch block, switch case value, and the `std::type_info` of the caught object.
- The type filter is the negative **index** into the **type table**.

```
std::type_info* current_type =
        type_table_end[-type_filter];
```



0 == end of records

GCC LSDA: Action Table

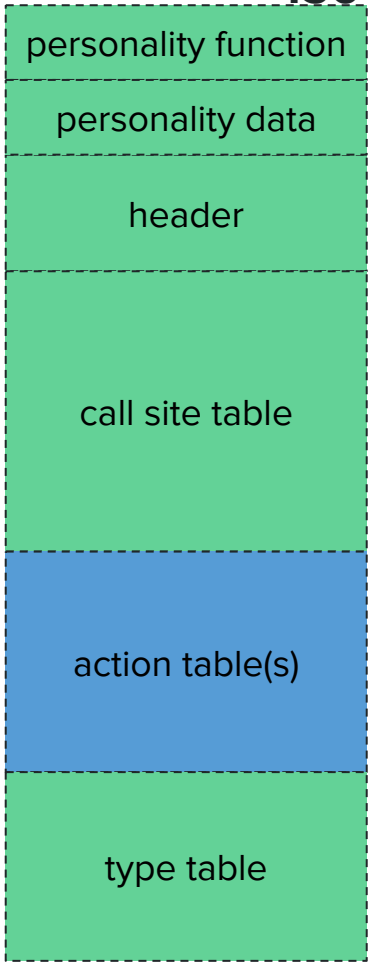


If the runtime reaches a `next_record` value of 0 and with no types match, then the current frame should be unwound.

GCC LSDA: Entering the Catch Block

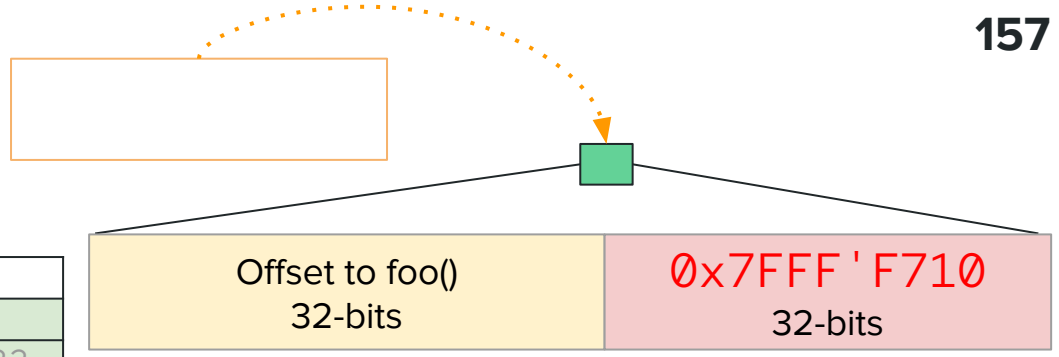
If a match between the type that was thrown and a catch type perform the following steps:

1. Set **R0** ← exception object's address
2. Set **R1** ← type filter number
3. Set **PC** ← `function_address + landing_pad | 1`
4. Set **actual CPU** to **virtual CPU**.



foo entry

Entering foo()



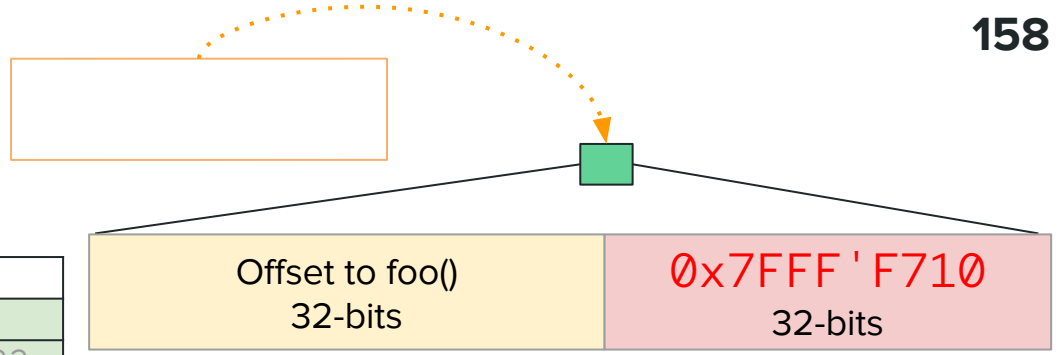
Virtual ARM CPU		SRAM	
Reg	*Value*	*Address*	*Value*
R0	0x2000'C000	0x4000	0x????????
R1	type_info	0x3FFC	0x????????
R2	0x0	0x3FF8	0x????????
R3	0x????????	0x3FF4	0x????????
R4	[foo] r4	0x3FF0	0x????????
R5	[foo] r5	0x3FEC	caller of foo
R6	–	0x3FE8	caller r4
R7	–	0x3FE4	caller r5
R8	–	0x3FE0	foo()+10
R9	–	0x3FDC	[foo] r5
R10	–	0x3FD8	[foo] r4
R11	–	0x3FD4	obj[1/3]
R12	–	0x3FD0	obj[2/3]
R13/SP	0x3FE0	0x3FCC	obj[3/3]
R14/LR	foo()+10	0x3FC8	0x????????
R15/PC	foo()+10	0x3FC4	0x????????

Dwarf Encoding 0xFF (omit)	**Type Encoding** prel32	**End of Type** 13	**Call Site** uleb128
Call Site Size 4	$start_0$ 0x0	$length_0$ 12	$landing\ pad_0$ 16
$action\ record_0$ 1	$filter_0$ 1	next offset 0	padding
std::type_info* for error			

```
throw_type_info == caught_type_info
```

foo entry

Entering foo()

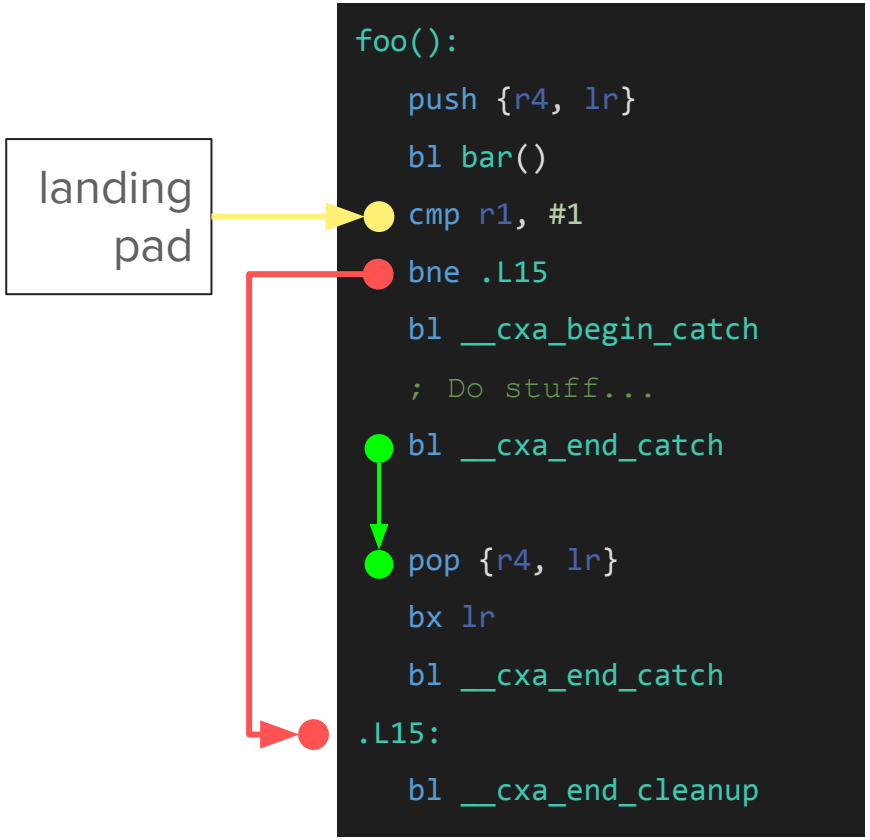


Virtual ARM CPU		SRAM	
Reg	*Value*	*Address*	*Value*
R0	0x2000'A100	0x4000	0x????????
R1	1	0x3FFC	0x????????
R2	0x0	0x3FF8	0x????????
R3	0x????????	0x3FF4	0x????????
R4	[foo] r4	0x3FF0	0x????????
R5	[foo] r5	0x3FEC	caller of foo
R6	–	0x3FE8	caller r4
R7	–	0x3FE4	caller r5
R8	–	0x3FE0	foo()+10
R9	–	0x3FDC	[foo] r5
R10	–	0x3FD8	[foo] r4
R11	–	0x3FD4	obj[1/3]
R12	–	0x3FD0	obj[2/3]
R13/SP	0x3FE0	0x3FCC	obj[3/3]
R14/LR	foo()+16	0x3FC8	0x????????
R15/PC	foo()+16	0x3FC4	0x????????

Dwarf Encoding 0xFF (omit)	**Type Encoding** prel32	**End of Type** 13	**Call Site** uleb128
Call Site Size 4	$start_0$ 0x0	$length_0$ 12	$landing\ pad_0$ 16
$action\ record_0$ 1	$filter_0$ 1	next offset 0	padding
std::type_info* for error			

```
throw_type_info == caught_type_info
```

Within the Catch Block



- **`__cxa_begin_catch`**
  - R0 contains the exception object which will be the first parameter to this function.
  - Set exception state to **handled**
  - Takes the exception object and returns the adjusted pointer to the exception object.
- **`__cxa_end_catch`**
  - Call destructor of thrown object
  - Frees thrown object memory

And that's

🎉 **ARM GCC Exceptions!** 🎉

Space Cost of Exception Handling

We are going to size up the following

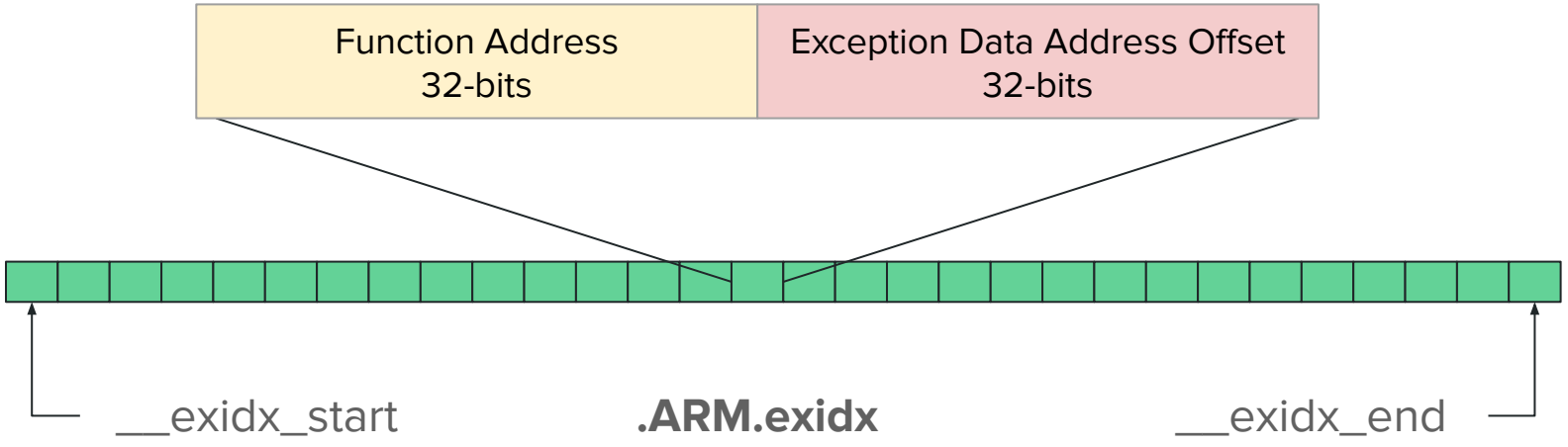
Structure	Size	Notes
Exception Index Entry	???	
SU16 Personality Data	???	
LU16/32 Personality Data	???	
LSDA header	???	
LSDA Call Site Table	???	
Cleanup Landing Area	???	
LSDA Action Table	???	
LSDA Type Table	???	
Catch Block	???	

Assumptions

- Assume the code is being compiled to minimize code size
- Assume that functions are generally at or below **16 kB** in size each
  - Firefox Debug Symbols Spreadsheet
- Assume that no function has more than a total of **63 catch blocks** and **destructor scopes** in it.
- Assume everything uses LEB128 & ULEB128 encodings.

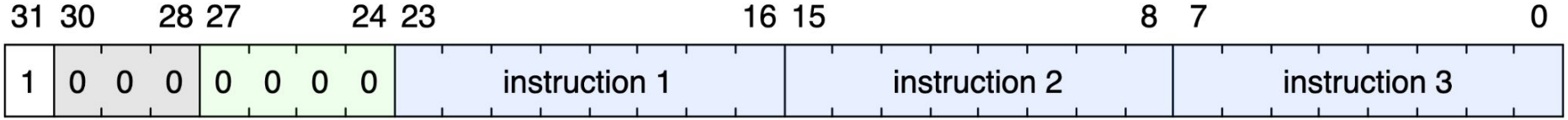
Firefox Function Distribution	
242	functions above (16kB)
325108	functions below (16kB)
157946	functions above (127B)
167404	functions below (127B)

sizeof(Exception Index Entry) = **8 bytes**



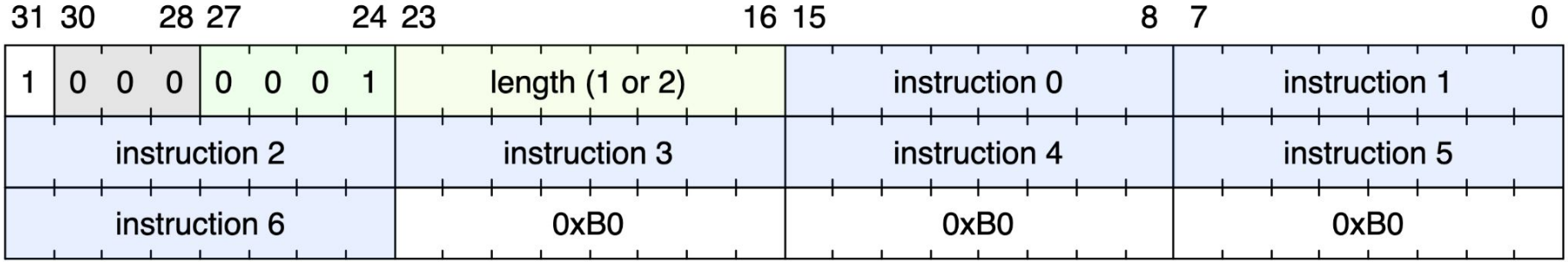
As defined by the ABI.

sizeof(SU16 Personality Data) = **4 bytes**



As defined by the ABI.

sizeof(LU16/32 Personality Data) = **12 bytes**



Worst case size as defined in the ABI.

sizeof(Catch Block) = **12 bytes**

- sizeof(cmp) = 2 bytes
- sizeof(beq) = 2 bytes
- sizeof(bl __cxa_begin_catch) = 4 bytes
- sizeof(bl __cxa_end_catch) = 4 bytes

```
00000000 <catch_my_errors()>:
  0:   b508        push    {r3, lr}
  2:   f7ff fffe   bl <may_throw_exception()>
  6:   e00f        b.n 1e <catch_my_errors()+0x28>
  8:   2901        cmp r1, #1
  a:   d001        beq.n   10 <catch_my_errors()+0x10>
  c:   f7ff fffe   bl <__cxa_end_cleanup>
 10:   f7ff fffe   bl <__cxa_begin_catch>
 14:   4805        ldr r0, [pc, #12]
 16:   f7ff fffe   bl <printf>
 1a:   f7ff fffe   bl <__cxa_end_catch>
 1e:   bd08        pop {r3, pc}
 26:   00000000    .word   0x00000000
```

sizeof(Cleanup Landing Area) = **6D + 4**

```
000088b4 <bar()>:
   ; happy path instructions
   88da: a801         add  r0, sp, #4
   88dc: f000 f834    bl   8948 <destructible_t::~destructible_t()>
   88e0: 4668         mov  r0, sp
   88e2: f000 f831    bl   8948 <destructible_t::~destructible_t()>
   88e6: f000 fccd    bl   9284 <__cxa_end_cleanup>
```

D = Each objects within a frame requiring cleanup

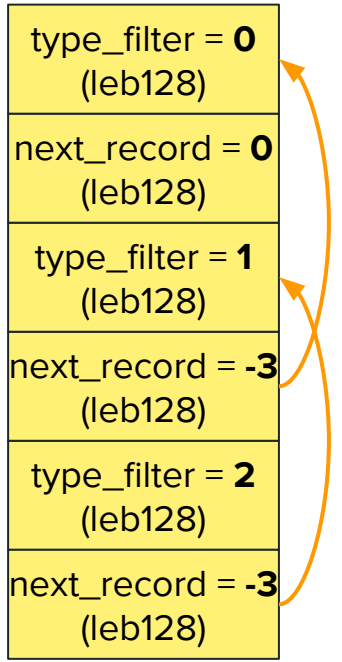
- sizeof(mov/add/sub) = 2 bytes
- sizeof(bl to destructor) = 4 bytes
- sizeof(bl __cxa_end_cleanup) = 4 bytes

6(2) + 4 = 16B

sizeof(LSDA Action Table) = **2A**

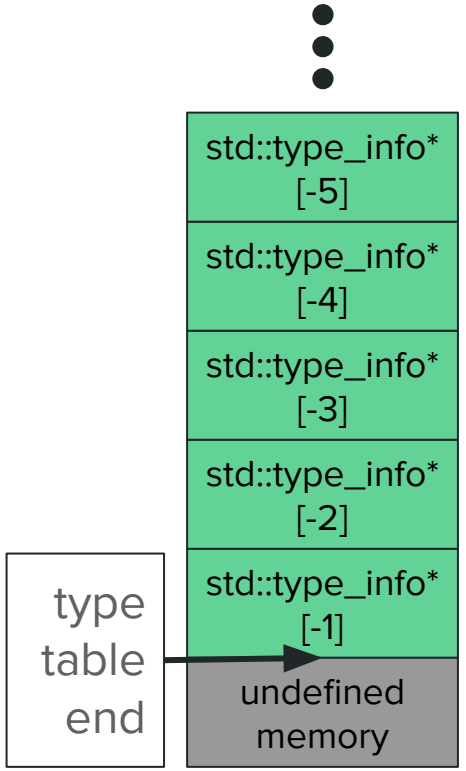
A = catch block count

- There is an entry for each
  - catch block
  - cleanup scope
- The range of a signed byte of LEB128 integer is -64 to 63
- Due to our constraints:
  - "Type filter" will not exceed 63 selectable types
  - "Next record" should not exceed 63 because entries are packed next to each other



sizeof(LSDA Type Table) = **4T**

T = number of unique types caught in the function

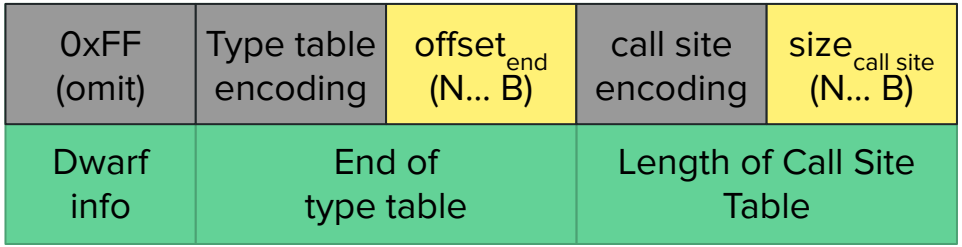


sizeof(LSDA Call Site) = **4S** ↔ **7S**

S = number of try scopes + scopes w/ cleanup

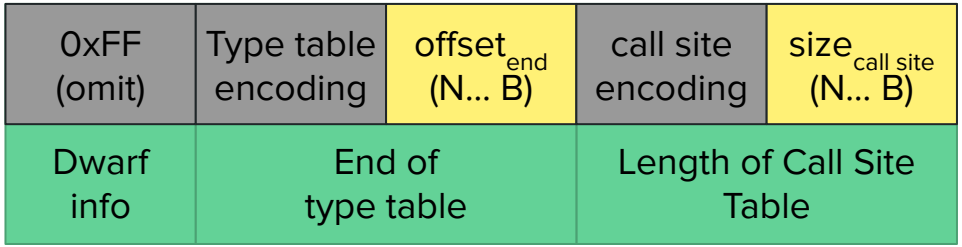
- 16kB function size constraint constraints **Start**, **length** & **landing pad** to 16kB or 2-bytes of ULEB128.
- The action table shouldn't exceed a size of 2*A (or 2*63) which is 126 bytes, thus the **Action record offset** should stay as a single ULEB128 byte

sizeof(LSDA header) = 5 ↔ 7



- Minimum size of 5 bytes
- What could grow in it?
  - Type table size?
  - Call site length?

sizeof(LSDA header) = 5 ↔ 7



- With our constraints
  - **Type table** cannot exceed 63 types so the max size is 4*63 = **252 bytes**
  - **Action table** cannot exceed 63 catch blocks + cleanup regions so the max is 2*(63) = **126 Bytes**
  - Total size without the call site of **372 bytes**, leaving 16,004 bytes for the call site.
  - Assuming the worst case of 7S, this would require a function to have ~**2,286** call site scopes, which seems impractical for most code.

I assert that with our constraints the size range is 5 bytes to 7 bytes.

We are going to size up the following

Structure	Size (bytes)	Notes
Exception Index Entry	8	
SU16 Personality Data	4	
LU16/32 Personality Data	12	
LSDA header	5 ↔ 7	
LSDA Call Site Table	4S ↔ 7S	where S = count of try & cleanup scopes
Cleanup Landing Area	6D + 4	where D = count of objects that need cleanup
LSDA Action Table	2A	where A = the number of catch blocks & cleanup scopes
LSDA Type Table	4T	where T = number of unique types caught in the function
Catch Block	12	

Examples! Trivially Unwindable Function

```
void blink_n_times(
  output_pin& p_led,
  uint8_t p_count)
{
 for (uint8_t i = 0; i < p_count; i++) {
   led.level(false);
   delay(500ms);
   led.level(true);
   delay(500ms);
 }
}
```

- **Metadata Components:**
  - Index Entry: 8-bytes
- **Total: 8 bytes**
- The size will not increase unless:
  - Stack size exceeds 512 (+8 bytes)
  - Object with destructor added
  - try/catch added

Examples! Function with single destructor scope

```
void foo(i2c& p_i2c,
         std::span<std::byte> p_data)
{
 std::lock_guard<my_mutex_t> lock(my_mutex);
 write(i2c, 0x44, p_data);
}
```

- **Metadata Components**
  - Index Entry: 8 bytes
  - GCC LSDA:
    - Header: 5 bytes
    - Personality: 4 bytes (SU16)
    - Call site table: 4 bytes
    - Padding: 3 bytes
  - Cleanup Landing Area: 10 bytes
- **Total: 34 bytes**
- The size will not increase unless:
  - Stack size exceeds 512 (+8 bytes)
  - Object with destructor added (+6 each)
  - try/catch added (Uses LSDA)

Examples! Function with try/catch

```
[[nodiscard]] inline bool probe(i2c& p_i2c,
  hal::byte p_address) {
 std::array<std::byte, 1> data_in{};
 bool device_acknowledged = true;
 try {
   p_i2c.transaction(p_address, {}, data_in);
 } catch (const hal::no_such_device&) {
   device_acknowledged = false;
 }
 return device_acknowledged;
}
```

- **Metadata Components**
  - Index Entry: 8 bytes
  - GCC LSDA:
    - Header: 5 bytes (min sized)
    - Personality: 4 bytes (SU16)
    - Try/Destructor Scopes: 1
    - Catch Count: 1
    - Unique Types Caught: 1
  - Catch Blocks: 1

$8 + (5 + 4 + 4S + 2A + 4T) + 12C \rightarrow 8 + (5 + 4 + 4(1) + 2(1) + 4(1)) + 12(1)$

Total: 47 bytes

Space Cost of Return Type Error Handling

Basis Function: A function calling other functions

Consider the following with no error handling/using exception

```
namespace {
uint32_t func1();
uint32_t func2();
uint32_t func3();
}  // namespace

uint32_t none()
{
  return func1() + func2() +
func3();
}
```

```
000004c0 <none()>:
    4c0:   b508        push    {r3, lr}
    4c2:   f7ff fff7   bl  4b4 <func1()>
    4c6:   4603        mov r3, r0
    4c8:   f7ff fff6   bl  4b8 <func2()>
    4cc:   4403        add r3, r0
    4ce:   f7ff fff5   bl  4bc <func3()>
    4d2:   4418        add r0, r3
    4d4:   bd08        pop {r3, pc}
    4d6:   bf00        nop
```

GCC ARM 12.3.0
-O3
-std=c++23
-fexceptions
-frtti
-mthumb
-mcpu=cortex-m4
-mfloat-abi=hard

Returning **std::expected**

```
enum class my_error_t : std::uint8_t
{
 error1,
 error2,
 error3,
 error4
};
```

```
namespace {
std::expected<uint32_t, my_error_t> func1();
// the rest . . .
}  // namespace

std::expected<uint32_t, my_error_t> returning_expected()
{
 uint32_t sum = 0;
 auto result1 = func1();
 [[unlikely]] if (!result1) {
   return std::unexpected(result1.error());
 }
 sum += result1.value();
 auto result2 = func2();
 [[unlikely]] if (!result2) {
   return std::unexpected(result2.error());
 }
 sum += result2.value();
 auto result3 = func3();
 [[unlikely]] if (!result3) {
   return std::unexpected(result3.error());
 }
 sum += result3.value();
 return sum;
}
```

Returning **std::expected: Disassembled!**

```
00000468 <returning_expected()>:
    468:   b510        push    {r4, lr}
    46a:   b084        sub sp, #16
    46c:   4603        mov r3, r0
    46e:   4668        mov r0, sp
    470:   f7ff ffe8   bl  444 <func1()>
    474:   f89d 2004   ldrb.w  r2, [sp, #4]
    478:   b1e2        cbz r2, 4b4 <using_expected()+0x4c>
    47a:   a802        add r0, sp, #8
    47c:   9c00        ldr r4, [sp, #0]
    47e:   f7ff ffe7   bl  450 <func2()>
    482:   f89d 200c   ldrb.w  r2, [sp, #12]
    486:   b172        cbz r2, 4a6 <using_expected()+0x3e>
    488:   9a02        ldr r2, [sp, #8]
    48a:   4414        add r4, r2
    48c:   f7ff ffe6   bl  45c <func3()>
    490:   f89d 200c   ldrb.w  r2, [sp, #12]
    494:   b13a        cbz r2, 4a6 <using_expected()+0x3e>
    496:   9902        ldr r1, [sp, #8]
    498:   2201        movs    r2, #1
    49a:   440c        add r4, r1
    49c:   4618        mov r0, r3
    49e:   601c        str r4, [r3, #0]
    4a0:   711a        strb    r2, [r3, #4]
    4a2:   b004        add sp, #16
    4a4:   bd10        pop {r4, pc}
    4a6:   f89d 1008   ldrb.w  r1, [sp, #8]
    4aa:   7019        strb    r1, [r3, #0]
    4ac:   4618        mov r0, r3
    4ae:   711a        strb    r2, [r3, #4]
    4b0:   b004        add sp, #16
    4b2:   bd10        pop {r4, pc}
    4b4:   f89d 1000   ldrb.w  r1, [sp]
    4b8:   7019        strb    r1, [r3, #0]
    4ba:   4618        mov r0, r3
    4bc:   711a        strb    r2, [r3, #4]
    4be:   b004        add sp, #16
    4c0:   bd10        pop {r4, pc}
    4c2:   bf00        nop
```

Cost of **6 bytes** per function call

Returning **bool**

```
namespace {
bool func1(std::uint32_t* p_value);
// The rest...
}  // namespace
```

```
bool returns_bool(std::uint32_t* p_value)
{
 uint32_t sum = 0;
 uint32_t temporary = 0;

 auto success = func1(&temporary);
 [[unlikely]] if (not success) {
   return false;
 }
 sum += temporary;

 success = func2(&temporary);
 [[unlikely]] if (not success) {
   return false;
 }
 sum += temporary;

 success = func3(&temporary);
 [[unlikely]] if (not success) {
   return false;
 }
 sum += temporary;

 *p_value = sum;

 return true;
}
```

Return **bool: Disassembled!**

```
00000510 <returns_bool(unsigned long*)>:
    510:   b510        push    {r4, lr}
    512:   b082        sub sp, #8
    514:   4602        mov r2, r0
    516:   a801        add r0, sp, #4
    518:   f7ff ffee   bl  4f8 <func1(unsigned long*)>
    51c:   b910        cbnz    r0, 524 <returns_bool+0x14>
    51e:   2000        movs    r0, #0
    520:   b002        add sp, #8
    522:   bd10        pop {r4, pc}
    524:   a801        add r0, sp, #4
    526:   9901        ldr r1, [sp, #4]
    528:   f7ff ffea   bl  500 <func2(unsigned long*)>
    52c:   2800        cmp r0, #0
    52e:   d0f6        beq.n   51e <returns_bool+0xe>
    530:   a801        add r0, sp, #4
    532:   9c01        ldr r4, [sp, #4]
    534:   f7ff ffe8   bl  508 <func3(unsigned long*)>
    538:   2800        cmp r0, #0
    53a:   d0f0        beq.n   51e <returns_bool+0xe>
    53c:   9b01        ldr r3, [sp, #4]
    53e:   4421        add r1, r4
    540:   440b        add r3, r1
    542:   6013        str r3, [r2, #0]
    544:   e7ec        b.n 520 <returns_bool+0x10>
    546:   bf00        nop
```

Cost of **4 bytes** per function call

Using Rust

```
rustc
--target thumbv7em-none-eabi
-C opt-level="s"
--emit=obj
rust.rs
```

```rust
pub fn combined() -> Result<i32, u32> {
 let mut num: i32 = 0;
 num += func1()?;
 num += func2()?;
 num += func3()?;
 return Ok(num);
}

#[inline(never)]
pub fn func1() -> Result<i32, u32> { return Ok(5); }
#[inline(never)]
pub fn func2() -> Result<i32, u32> { return Err(88);}
#[inline(never)]
pub fn func3() -> Result<i32, u32> { return Ok(13); }
```

Return Rust: Disassembled!

```
00000000 <rust::combined::h7a7b0cc556a62e18>:
   0: b5f0          push  {r4, r5, r6, r7, lr}
   2: af03          add r7, sp, #12
   4: f84d bd04     str r11, [sp, #-4]!
   8: b082          sub sp, #8
   a: 4604          mov r4, r0
   c: 4668          mov r0, sp
   e: f7ff fffe     bl  0xe
  12: e9dd 0500     ldrd r0, r5, [sp]
  16: b108          cbz  r0, 0x1c
  18: 2001          movs r0, #1
  1a: e014          b  0x46
  1c: 4668          mov  r0, sp
  1e: f7ff fffe     bl 0x1e
  22: e9dd 0600     ldrd r0, r6, [sp]
  26: b110          cbz  r0, 0x2e
  28: 2001          movs r0, #1
  2a: 4635          mov  r5, r6
  2c: e00b          b  0x46
  2e: 4668          mov  r0, sp
  30: f7ff fffe     bl 0x30
  34: e9dd 0100     ldrd r0, r1, [sp]
  38: b110          cbz  r0, 0x40
  3a: 2001          movs r0, #1
  3c: 460d          mov  r5, r1
  3e: e002          b  0x46
  40: 1970          adds r0, r6, r5
  42: 1845          adds r5, r0, r1
  44: 2000          movs r0, #0
  46: e9c4 0500     strd r0, r5, [r4]
  4a: b002          add  sp, #8
  4c: f85d bb04     ldr  r11, [sp], #4
  50: bdf0          pop  {r4, r5, r6, r7, pc}
```

```
objdump --disassemble --demangle
rust.o > rust.S
```

Cost of **6 bytes** per function call

Using Rust (u64)

```
rustc
--target thumbv7em-none-eabi
-C opt-level="s"
--emit=obj
rust.rs
```

```rust
#[inline(never)]
pub fn combined() -> Result<i32, u64> {
    let mut num: i32 = 0;
    num += func1()?;
    num += func2()?;
    num += func3()?;
    Ok(num)
}

#[inline(never)]
pub fn func1() -> Result<i32, u64> { Ok(5) }
#[inline(never)]
pub fn func2() -> Result<i32, u64> { Err(8800) }
#[inline(never)]
pub fn func3() -> Result<i32, u64> { Ok(13) }
```

Return Rust (u64): Disassembled

```
00000000 <rust::combined::hece36e1d2889c9fd>:
       0: b5f0          push  {r4, r5, r6, r7, lr}
       2: af03          add   r7, sp, #12
       4: f84d bd04     str   r11, [sp, #-4]!
       8: b084          sub   sp, #16
       a: 4604          mov   r4, r0
       c: 4668          mov   r0, sp
       e: f7ff fffe     bl    0xe
      12: 9800          ldr   r0, [sp]
      14: b958          cbnz  r0, 0x2e
      16: 4668          mov   r0, sp
      18: 9d01          ldr   r5, [sp, #4]
      1a: f7ff fffe     bl    0x1a
      1e: 9800          ldr   r0, [sp]
      20: b928          cbnz  r0, 0x2e
      22: 4668          mov   r0, sp
      24: 9e01          ldr   r6, [sp, #4]
      26: f7ff fffe     bl    0x26
      2a: 9800          ldr   r0, [sp]
      2c: b148          cbz   r0, 0x42
      2e: 9903          ldr   r1, [sp, #12]
      30: 9802          ldr   r0, [sp, #8]
      32: e9c4 0102     strd  r0, r1, [r4, #8]
      36: 2001          movs  r0, #1
      38: 6020          str   r0, [r4]
      3a: b004          add   sp, #16
      3c: f85d bb04     ldr   r11, [sp], #4
      40: bdf0          pop   {r4, r5, r6, r7, pc}
      42: 9901          ldr   r1, [sp, #4]
      44: 1970          adds  r0, r6, r5
      46: 4408          add   r0, r1
      48: 6060          str   r0, [r4, #4]
      4a: 2000          movs  r0, #0
      4c: e7f4          b     0x38
```

```
objdump --disassemble --demangle
rust.o > rust.S
```

Cost of **4 bytes** per function call

Distributed vs Centralized Error Handling

Distributed Error Handling

- A form of error handling where the work & code for propagating error information up the stack to a handler is distributed across all functions that participate in error handling.
- If a function wants to call a function that can emit an error, it is the calling functions job to either:
  - Handle the error

    OR

  - Propagate (return) the error
- Using a return type or an out parameter as your error handling is an example of this.

Centralized Error Handling

- A form of error handling where a singular mechanism along with some metadata about the system is capable of propagating error information from the detection site to the appropriate handler.
- Calling code does not have to perform any work during normal operation in order to participate in error handling.
- Normal code & error handling code are in separate domains.
- Exception handling is a form of centralized error handling

My Claim:
Centralized Error handling schemes have an advantage in terms of code size compared to distributed error handling.

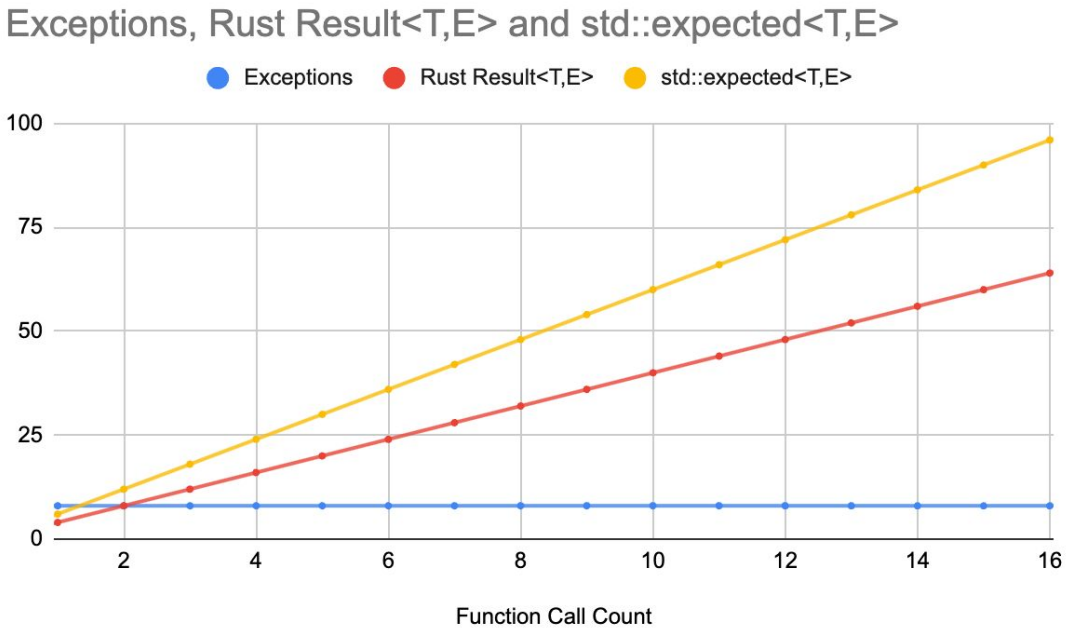
Point 1:

Branching isn't FREE!

Point 1: Branching isn't free!

Consider the following function using any distributed type error mechanism…

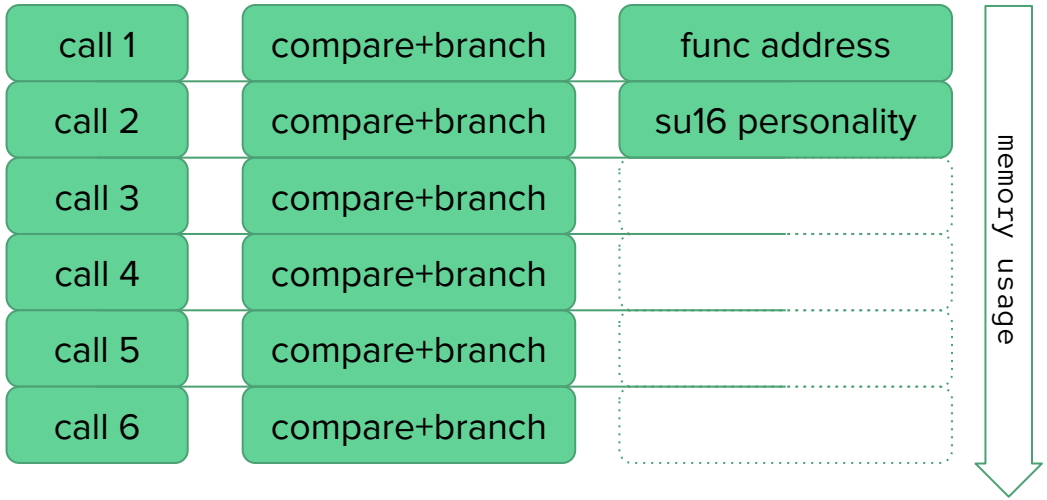
```
uint32_t func1();
uint32_t func2();
uint32_t func3();
uint32_t func4();
uint32_t func5();

uint32_t combined_function() {
    return func1() + func2() +
           func3() + func4() +
           func5();
}
```



Point 1: Branching isn't free!

Consider the following function using any return type error mechanism…

```
uint32_t func1();
uint32_t func2();
uint32_t func3();
uint32_t func4();
uint32_t func5();
uint32_t func6();

uint32_t combined_function() {
   return func1() + func2() + func3() +
          func4() + func5() + func6();
}
```



Point 1: Branching isn't free!

```
[[nodiscard]] inline bool probe(i2c& p_i2c,
  hal::byte p_address)
{
 std::array<std::byte, 1> data_in{};
 bool device_acknowledged = true;

 try {
   p_i2c.transaction(p_address, {}, data_in);
 } catch (const hal::no_such_device&) {
   device_acknowledged = false;
 }
 return device_acknowledged;
}
```

Consider this function from before.
Adds size is **47 bytes**

How many checked functions would cost the same?

About **11** to **12 checked** function calls, ignoring the additional return paths.

Point 2:

Most functions are trivially unwindable

Point 2: Most functions are trivially unwindable

1. What types of functions do you write?
2. What kinds of functions do you normally see/read?
3. How many functions in your code base **handle errors** vs **propagating it up**?
4. How much of your functions create objects with **non-trivial destructors**?

Point 2: Most functions are trivially unwindable

Consider **Audacity**:

- **1054** C++ files **only 39 files** (or 3.7% of files) have a catch block.
- Total of **60** try scopes and **69** catch blocks were found in the source code.
- Out of **177433** functions **50** functions (0.028%) had a try/catch blocks
- Exception sections add **+8%** memory relative to the **.text** section (15,415 kB)

Point 2: Most functions are trivially unwindable

Let's consider code with this distribution of functions:

1. **~5%** of functions catch errors
2. **~25%** allocate objects with non-trivial destructors
3. **~55%** of functions are trivially unwindable
4. **~10%** of functions is leaf functions

Point 2: Most functions are trivially unwindable

NOTE:
***C** code is always leaf/trivial*

So if we had some code with 1,000 functions:

1. 50 try/catch functions require +47B ➔ lets use **64 bytes**
2. 250 cleanup functions require +34B ➔ lets use **46 bytes**
3. 550 trivial functions require 8 to 24 ➔ lets use **8 bytes** (LU should be rare)
4. 100 leaf functions use **0 bytes** of exception info

Point 2: Most functions are trivially unwindable

NOTE:
***C** code is always leaf/trivial*

So if we had some code with 1,000 functions:

1. 50 try/catch functions require +47B ➜ lets use **64 bytes**
2. 250 cleanup functions require +34B ➜ lets use **46 bytes**
3. 550 trivial functions require 8 to 24 ➜ lets use **8 bytes** (LU should be rare)
4. 100 leaf functions use **0 bytes** of exception info

8000 + 50*64 + 250*46 + 550*8 + 100*0 = **27,100 bytes**

Point 2: Most functions are trivially unwindable

So if we had some code with 1,000 functions:

1. 50 try/catch functions require +47B ➜ lets use **64 bytes**
2. 250 cleanup functions require +34B ➜ lets use **46 bytes**
3. 550 trivial functions require 8 to 24 ➜ lets use **8 bytes** (LU should be rare)
4. 100 leaf functions use **0 bytes** of exception info

8000 + 50*64 + 250*46 + 550*8 + 100*0 = **27,100 bytes**

27,100 bytes * (1 check / 4 bytes) = **6,775 checked function calls**

Point 2: Most functions are trivially unwindable

So if we had some code with 1,000 functions:

1. 50 try/catch functions require +47B ➔ lets use **64 bytes**
2. 250 cleanup functions require +34B ➔ lets use **46 bytes**
3. 550 trivial functions require 8 to 24 ➔ lets use **8 bytes** (LU should be rare)
4. 100 leaf functions use **0 bytes** of exception info

8000 + 50*64 + 250*46 + 550*8 + 100*0 = **27,100 bytes**

27,100 bytes * (1 check / 4 bytes) = **6,775 checked function calls**

6,775 checked / 900 functions = **~7.5 checked function calls per function**

NOTE: Should be less, because we aren't considering additional epilogs

Point 2: Most functions are trivially unwindable

So if we had some code with 1,000 functions:

1. 50 try/catch functions require +47B ➔ lets use **64 bytes**
2. 250 cleanup functions require +34B ➔ lets use **46 bytes**
3. 550 trivial functions require 8 to 24 ➔ lets use **8 bytes** (LU should be rare)
4. 100 leaf functions use **0 bytes** of exception info

8000 + 50*64 + 250*46 + 550*8 + 100*0 = **27,100 bytes**

27,100 bytes * (1 check / 6 bytes) = **4,516 checked function calls**

4,516 checked / 900 functions = **~5.0 checked function calls per function**

NOTE: Should be less, because we aren't considering additional epilogs

Point 2: Most functions are trivially unwindable

So if we had some code with 1,000 functions:

1. 50 try/catch functions require +47B ➔ lets use **64 bytes**
2. 250 cleanup functions require +34B ➔ lets use **46 bytes**
3. 550 trivial functions require 8 to 24 ➔ lets use **8 bytes** (LU should be rare)
4. 100 leaf functions use **0 bytes** of exception info

8000 + 50*64 + 250*46 + 550*8 + 100*0 = **27,100 bytes**

27,100 bytes * (1 check / 8 bytes) = **3387.5 checked function calls**

3387.5 checked / 900 functions = **~3.76 checked function calls per function**

NOTE: Assuming the average if 8-bytes per check w/ return path

How does C++ exceptions enable smaller binaries?

- Reduced codegen when calling fallible functions.
- The unwind information **scales** with the **number of functions you have** **NOT** the **number of functions you call**.
- Eliminates additional return paths in functions.
- The unwind information is a compressed representation of normal instructions.
- You only pay for the exception mechanism code once.

THIS ISN'T EVEN ITS **FINAL FORM!**

Exceptions can be smaller!!

Why not do this?

__gcc_personality_v0

Dwarf info	End of type table		Length of Call Site Table	
0xFF (omit) 1B	Type table encoding 1B	$offset_{end}$ (N... B)	call site encoding 1B	$size_{call\ site}$ (N... B)

__new_personality

End of type table	Length of Call Site Table
$offset_{end}$ (N... B)	$size_{call\ site}$ (N... B)

Type table encoding = prel
Call site encoding = uleb128

THIS IS BACKWARDS COMPATIBLE!

Got old code? Why not use a tool to convert?

__gcc_personality_v0

Dwarf info	End of type table		Length of Call Site Table	
0xFF (omit) 1B	Type table encoding 1B	$offset_{end}$ (N... B)	call site encoding 1B	$size_{call\ site}$ (N... B)

__new_personality?

End of type table	Length of Call Site Table
$offset_{end}$ (N... B)	$size_{call\ site}$ (N... B)

Saves 3 bytes for each reentrant function

Why not do this?

```
catch_chain():
        push    {lr}
        bl      foo()
        pop     {pc}
        cmp     r1, #1
        bne     .L3
        bl      __cxa_begin_catch
        ldr     r0, .L13
        bl      printf
        bl      __cxa_end_catch
.L3:
        bl      __cxa_begin_catch
        ldr     r0, .L13+4
        bl      printf
.L4:
        bl      __cxa_end_catch
        pop     {lr}
```

Saves 2B for each attached catch.

```
catch_chain():
        push    {lr}
        bl      foo()
        pop     {pc}
        cmp     r1, #1
        bne     .L3
        bl      __cxa_begin_catch
        ldr     r0, .L13
        bl      printf
        b       .L4
.L3:
        bl      __cxa_begin_catch
        ldr     r0, .L13+4
        bl      printf
.L4:
        bl      __cxa_end_catch
        pop     {lr}
```

Exceptions can be so much more than they are now

What are C++ exceptions really?

What are C++ exceptions really?

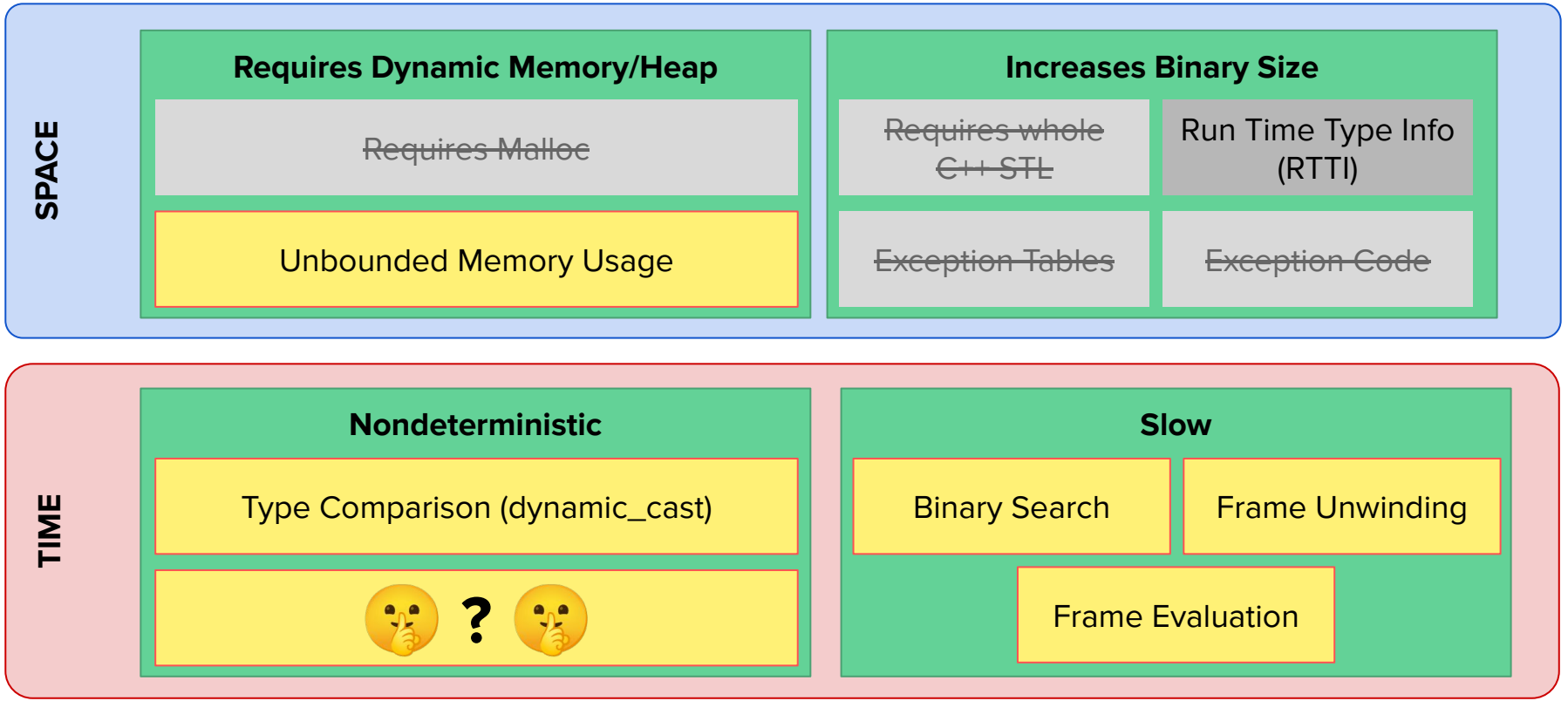
They are a **code** compression

```
00000000 <rust::combined::hece36e1d2889c9fd>:
      0: b5f0        push  {r4, r5, r6, r7, lr}
      2: af03        add   r7, sp, #12
      4: f84d bd04   str   r11, [sp, #-4]!
      8: b084        sub   sp, #16
      a: 4604        mov   r4, r0
      c: 4668        mov   r0, sp
      e: f7ff fffe   bl    0xe
     12: 9800        ldr   r0, [sp]
     14: b958        cbnz  r0, 0x2e
     16: 4668        mov   r0, sp
     18: 9d01        ldr   r5, [sp, #4]
     1a: f7ff fffe   bl    0x1a
     1e: 9800        ldr   r0, [sp]
     20: b928        cbnz  r0, 0x2e
     22: 4668        mov   r0, sp
     24: 9e01        ldr   r6, [sp, #4]
     26: f7ff fffe   bl    0x26
     2a: 9800        ldr   r0, [sp]
     2c: b148        cbz   r0, 0x42
     2e: 9903        ldr   r1, [sp, #12]
     30: 9802        ldr   r0, [sp, #8]
     32: e9c4 0102   strd  r0, r1, [r4, #8]
     36: 2001        movs  r0, #1
     38: 6020        str   r0, [r4]
     3a: b004        add   sp, #16
     3c: f85d bb04   ldr   r11, [sp], #4
     40: bdf0        pop   {r4, r5, r6, r7, pc}
     42: 9901        ldr   r1, [sp, #4]
     44: 1970        adds  r0, r6, r5
     46: 4408        add   r0, r1
     48: 6060        str   r0, [r4, #4]
     4a: 2000        movs  r0, #0
     4c: e7f4        b     0x38
```

+8 bytes

```
000004c0 <none()>:
    4c0:   b508          push    {r3, lr}
    4c2:   f7ff fff7     bl   4b4 <func1()>
    4c6:   4603          mov r3, r0
    4c8:   f7ff fff6     bl   4b8 <func2()>
    4cc:   4403          add r3, r0
    4ce:   f7ff fff5     bl   4bc <func3()>
    4d2:   4418          add r0, r3
    4d4:   bd08          pop {r3, pc}
    4d6:   bf00          nop
```

So why do firmware developers avoid C++ exceptions?



wHaT aBoUt tHe SaD pAtH pErFoRmAnCe!?

Cutting C++ Exception time by ~~+76%~~ +88% ✨

By: Khalil Estell

Some performance numbers

Function Depth	std::expected <u32, u32>	GCC & libunwind	My implementation
6	556	15344	2652
Multiplier for depth 6	1x	27x slower	4.77x slower
96	8566	184454	22102
Multiplier for depth 96	1x	21.53x slower	2.58x slower

But let's be honest with ourselves...

EXCEPTIONS

F****ing

SUCK!

What does `foo` throw???

```
void foo();
```

What does `foo` throw???

```
void foo();
```

Docs say, "throws `some_error`"

But does it actually ONLY throw that?

What does `foo` throw???

```
void foo();
```

Docs say, "throws `some_error`"

But does it actually ONLY throw that?

For version 5.1.0 this is true.

What does `foo` throw???

```
void foo();
```

Docs say, "throws `some_error`"

But does it actually ONLY throw that?

For version 5.1.0 this is true.

```
self.requires("foo-lib/[^5.1.0]")
self.requires("foo-lib/[^5.1.1]")
```

YOU DON'T KNOW!

THE **ULTIMATE** PROBLEM WITH EXCEPTIONS!

Luckily it's not impossible to know

Exception Insights Tool

By: Khalil Estell

Special Thanks

De-fragmenting C++: Making Exceptions and RTTI More Affordable and Usable - Herb Sutter CppCon 2019

87K views • 4 years ago

CppCon

A fundamental reason why C++ is successful and loved is its adherence to Stroustrup's zero-overhead principle: You don't pay for ...

9 moments C++'s evolution priorities | Part 1: Exception Handling (EH) | Code review... | Pathology 101 | Core...

Doc. No. P1947
Date: 2019-11-18
Audience: EWG and LEWG
Reply to: Bjarne Stroustrup (bs@ms.com)

C++ exceptions and alternatives

Bjarne Stroustrup

> *There has been little serious research on the topics of performance of exceptions and reliability of the resulting code in C++.*

✨ Supporting ✨

this work

Github sponsorship page

For inquiry: estell.exceptions@gmail.com

Questions?

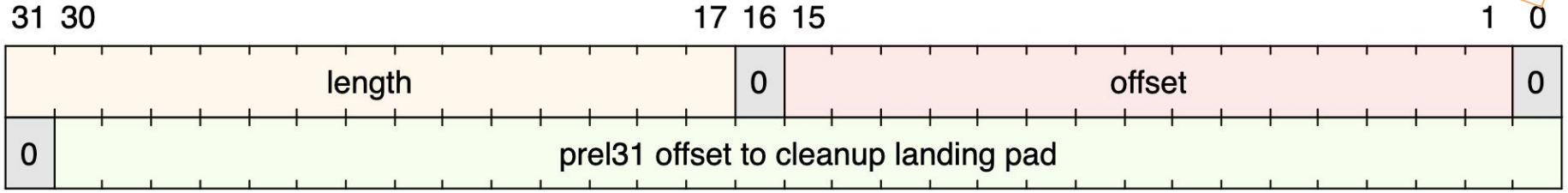
Github sponsorship page

For inquiry: estell.exceptions@gmail.com

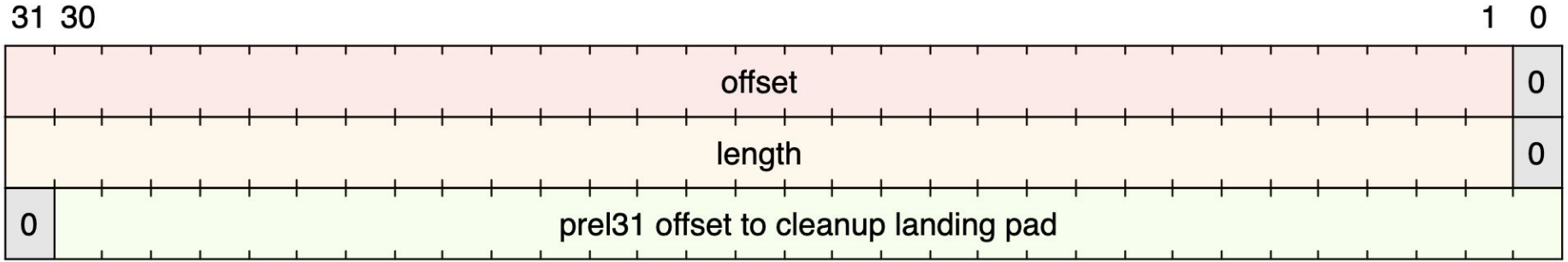
ARM Personality Data: cleanup descriptors

NOT IMPORTANT!

16-bit cleanup descriptor



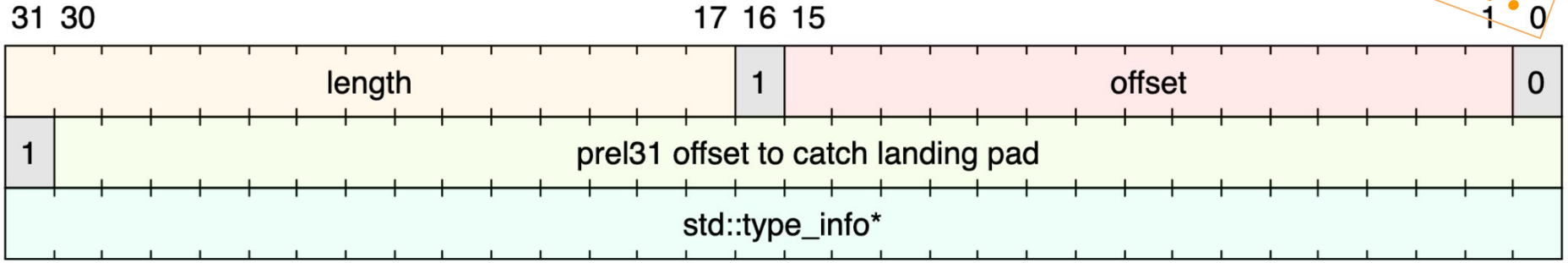
32-bit cleanup descriptor



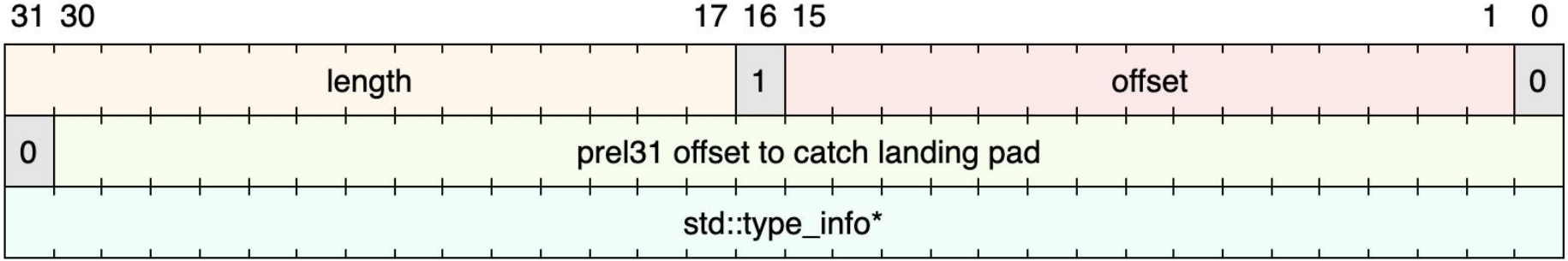
ARM Personality Data: catch descriptors

NOT IMPORTANT!

Catch by reference descriptor



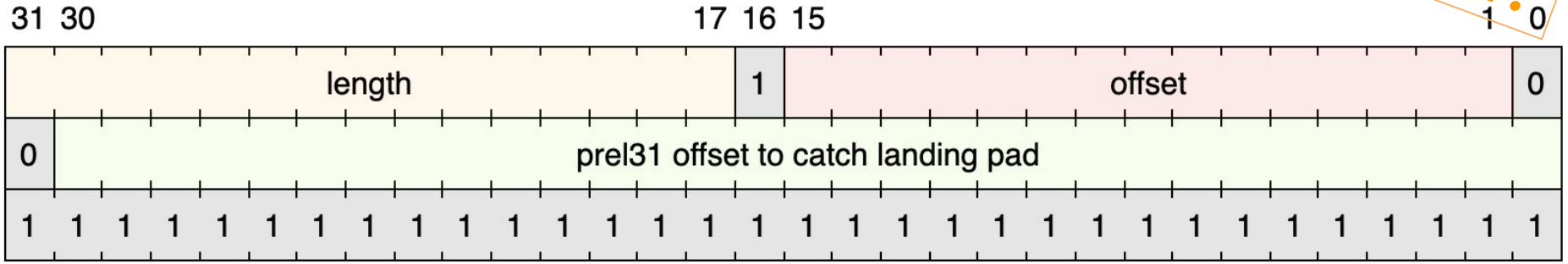
Catch by "anything else"



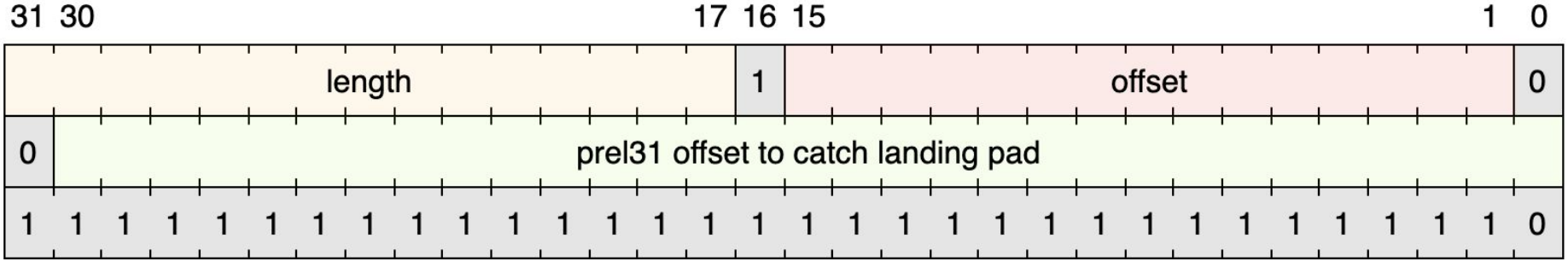
ARM Personality Data: catch descriptors

NOT IMPORTANT!

Catch All



Catch All & terminate



ARM Personality Data: Example Complete data

NOT IMPORTANT!