Cutting C++ Exception Time by 93.4%

KHALIL ESTELL

Cutting C++ Exception time by 90%

Part 2

@kammce


SAN JOSÉ STATE UNIVERSITY
1857

Chapters

Chapter 0. 

CppCon 2023 SG 14 ISO Meeting

October 4th

Conclusion

From the data: **β = 190**, **σ = 129**, **Thus:** β > σ ✅

With **150** functions: **783 x** tl::unexpected() calls, which corresponds to if/else checks.

What does this tell us?

- If you only have **4kB**, **8kB**, **16kB**, and even **32kB** flash, then exceptions may not be the right choice for your application because you cannot afford the +8kB flash cost for exceptions.
- With flash sizes **64kB** and above, exceptions seem to be the best option for code size, *IF you can handle the performance cost of throws*.
- Improving unwinding performance, removing the "need" for dynamic memory allocation, and removing TLS requirements, would make exceptions an extremely attractive options for embedded systems.

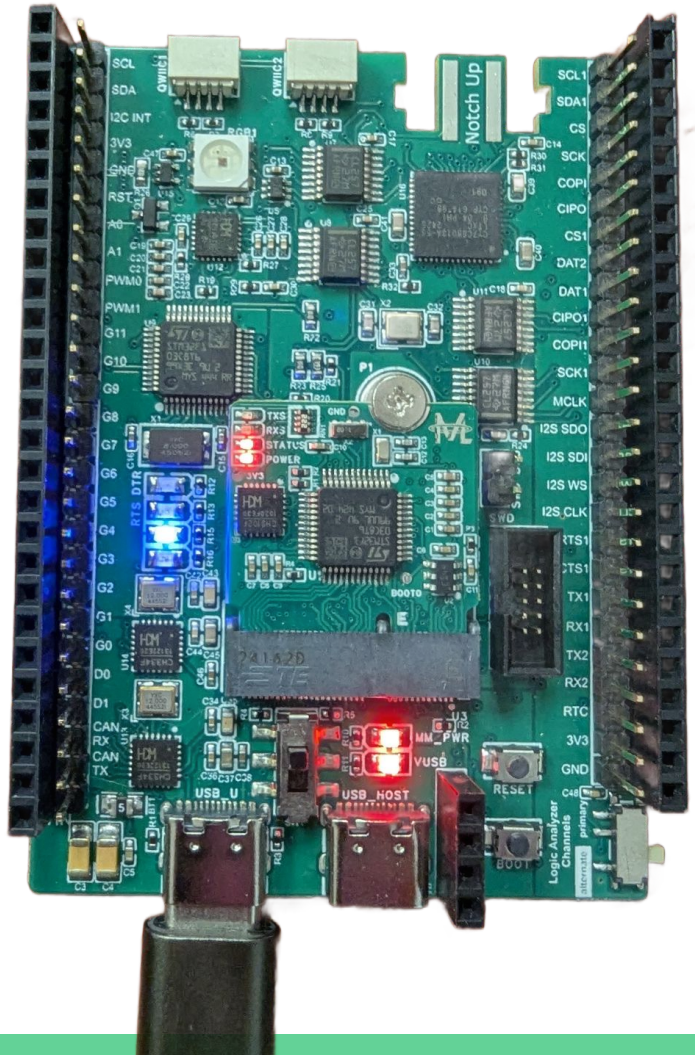
Improving **exceptions** would make them an **extremely attractive** options **for embedded systems**...

Chapter 1. ⏱️

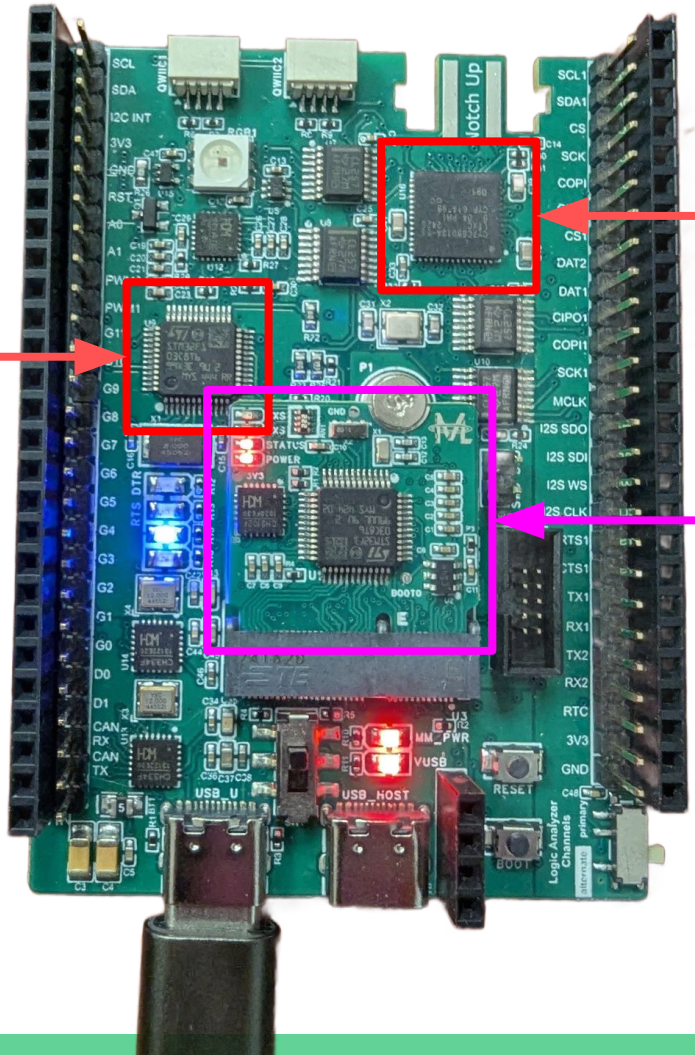
Measuring Performance
Benchmark Design

Test Setup & Hardware

Testing Hardware:
HALbORD P3



Testing Hardware:
HALbORD P3

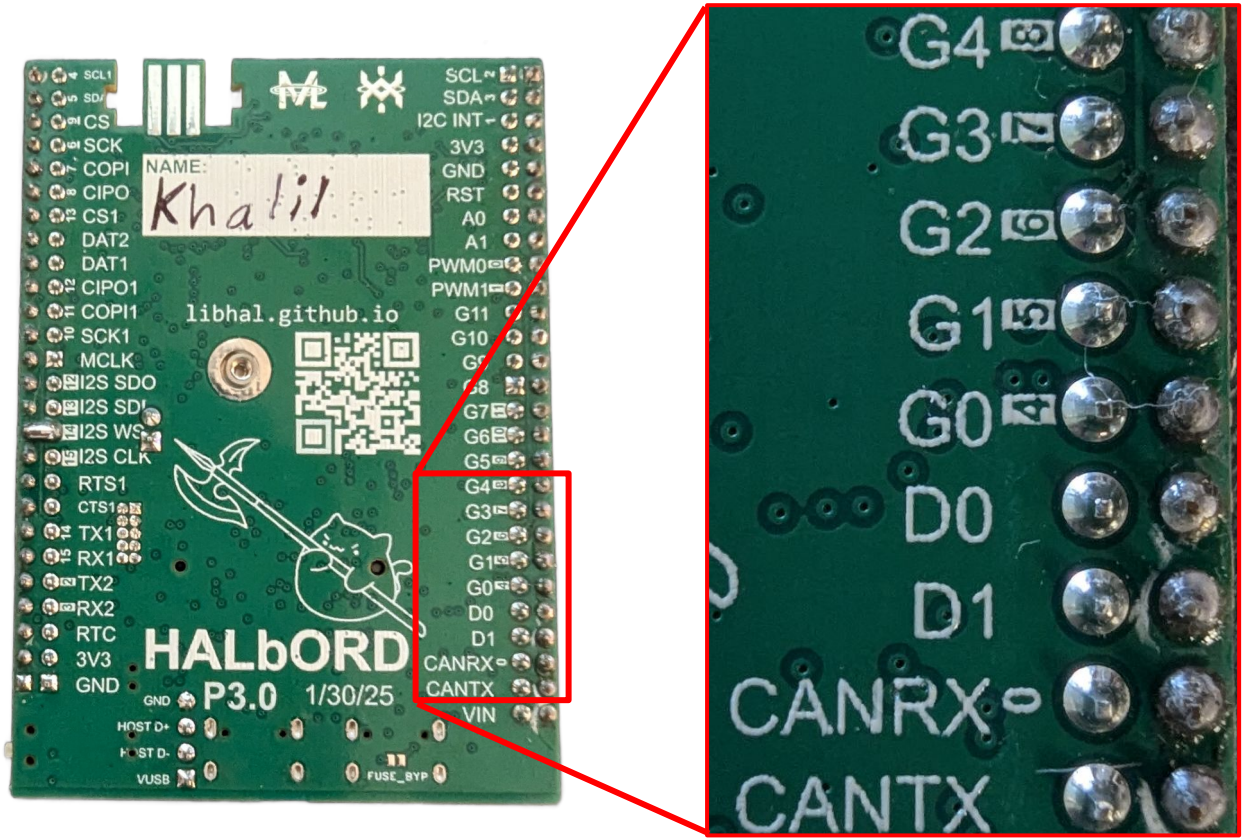


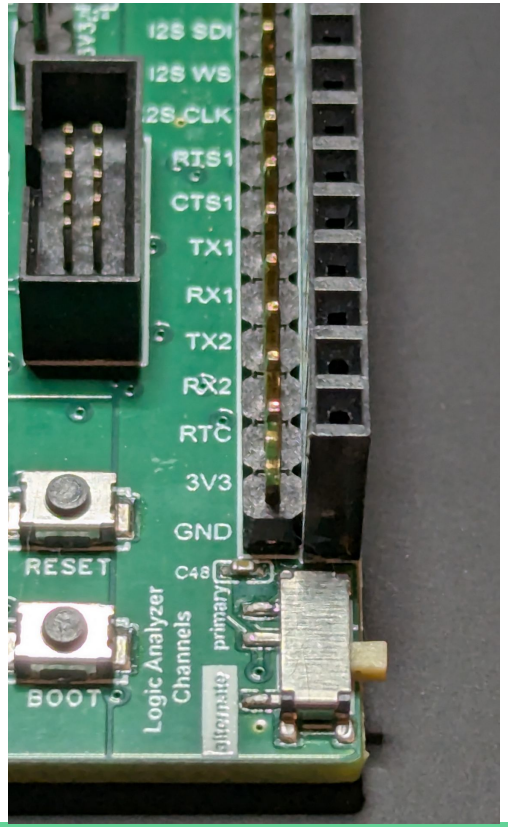
SWD/JTAG Debugger (gdb)

2x16-Channel Logic Analyzer

MCU

Logic Analyzer Channels & Switch

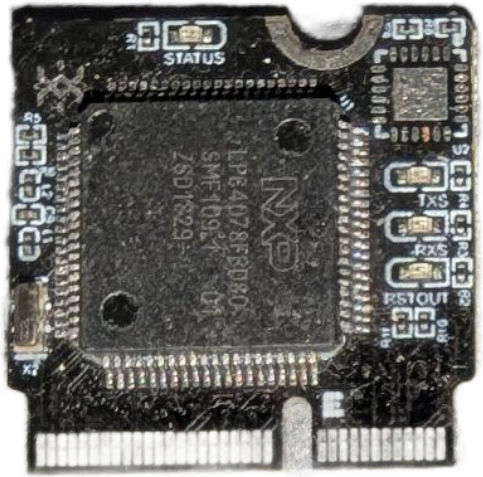




sparkfun ELECTRONICS

MicroMod

512kB of ROM



LPC4078

64kB of ROM
64MHz
Cortex-M3

STM32F103C8

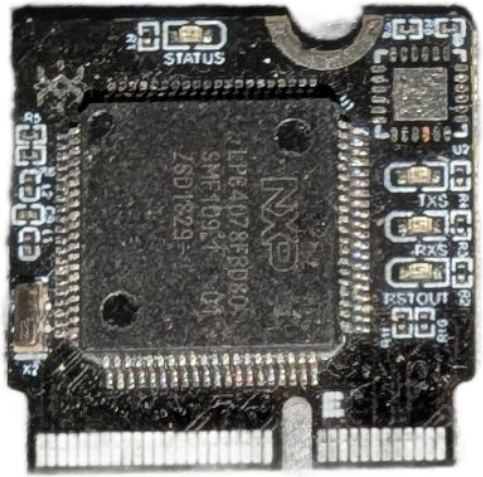
16MB of ROM

RP2040

sparkfun ELECTRONICS

MicroMod

512kB of ROM



LPC4078

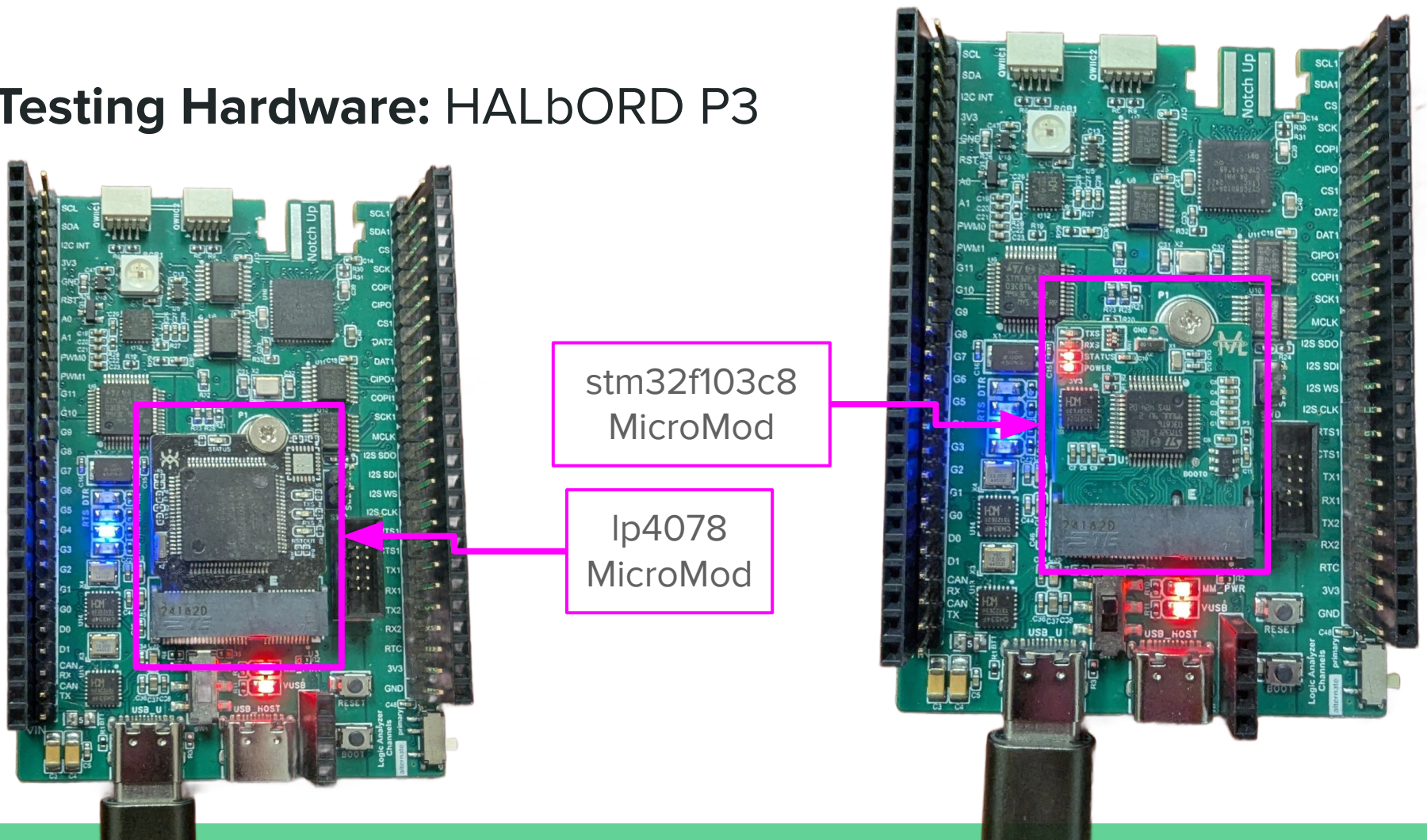
64kB of ROM
64MHz
Cortex-M3

STM32F103C8

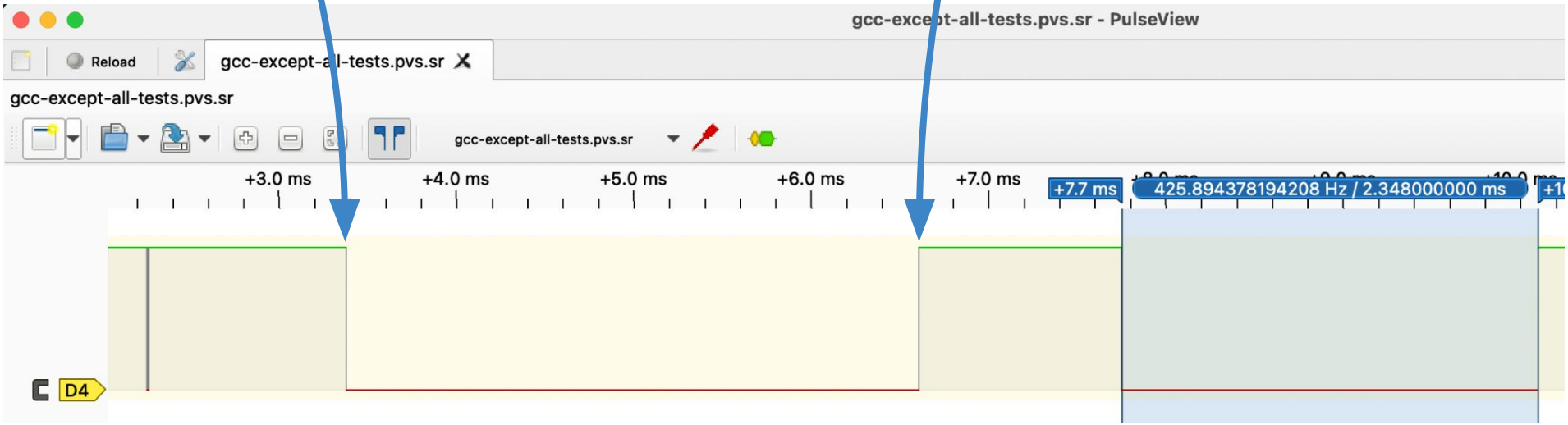
16MB of ROM

RP2040

Testing Hardware: HALbORD P3



Using the Logic Analyzer for Measurement



Microcontroller Benchmark Benefits: No OS

~~fopen()~~

~~networking~~

~~std::print()~~

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

~~std::thread~~

Benchmark Design

My Toolchain

14.2

1.8.10

Benchmark Specification

expected

exceptions

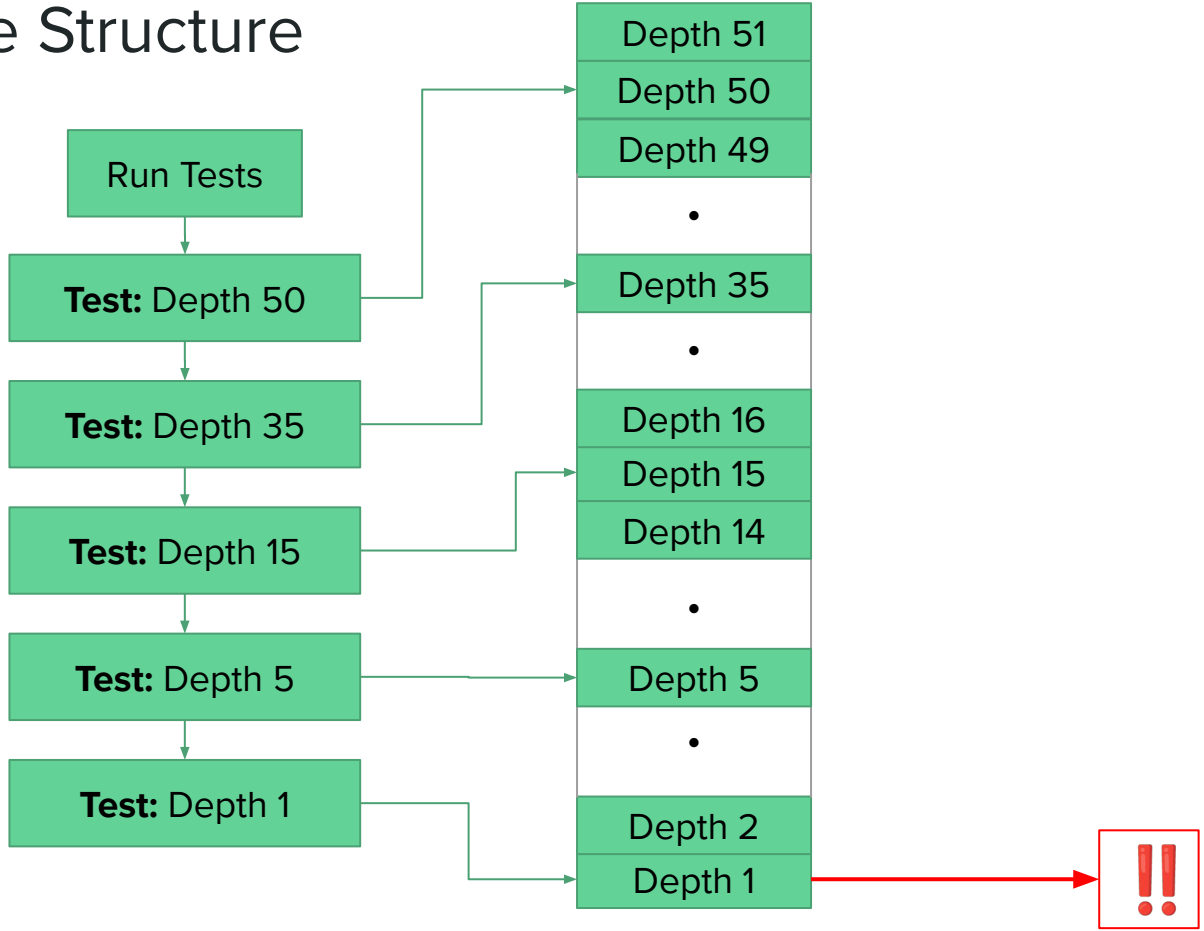
Goal: Measure propagation time up the stack from ***error detection*** to ***handler***.

- Should compare idiomatic `std::expected<T,E>` & `exceptions`.
- Programs and functions must be **isomorphic** to each other
- Test on Release (-O3) mode for highest performance

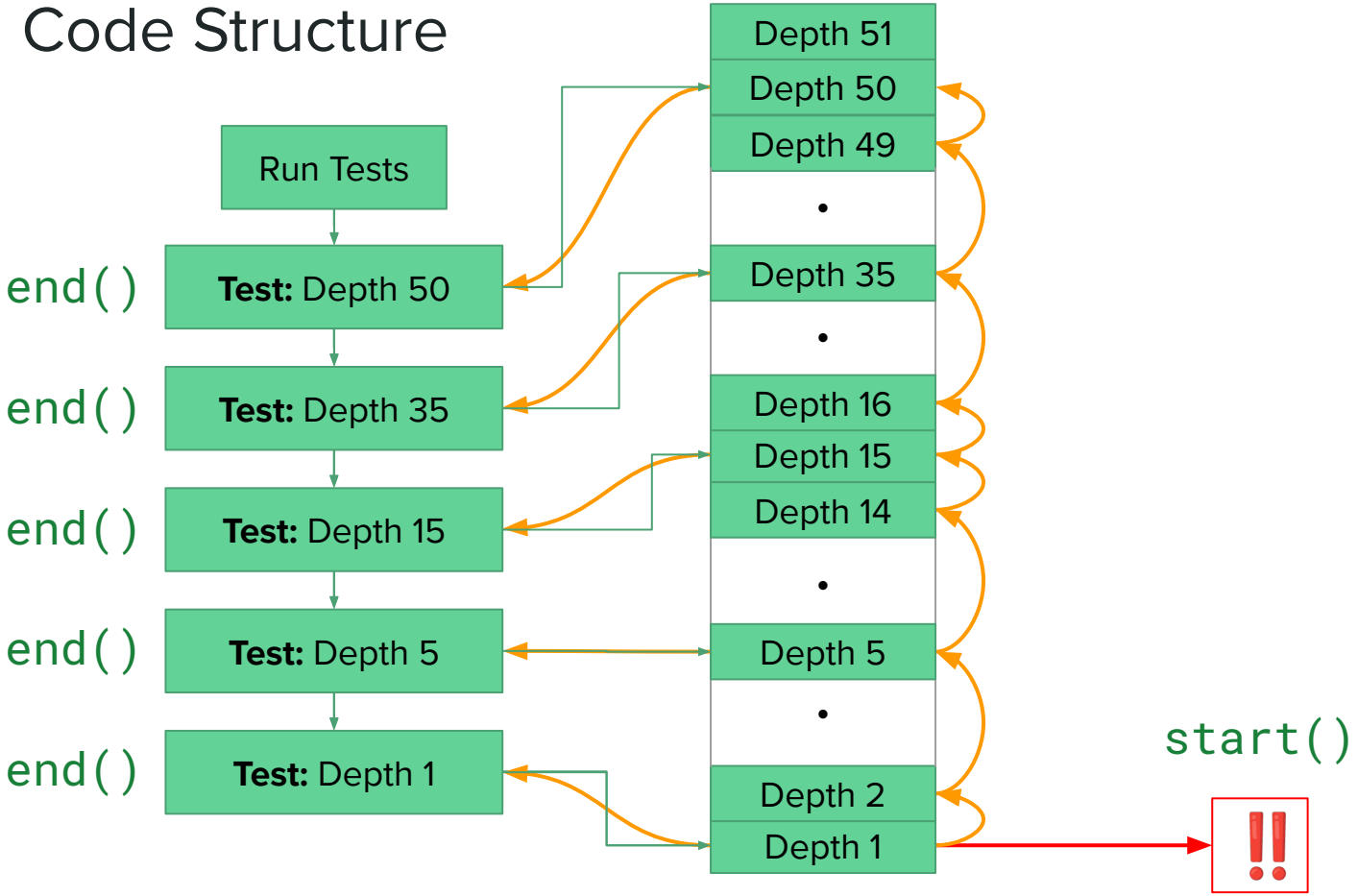
What we are **NOT** Testing

- Allocation Speed

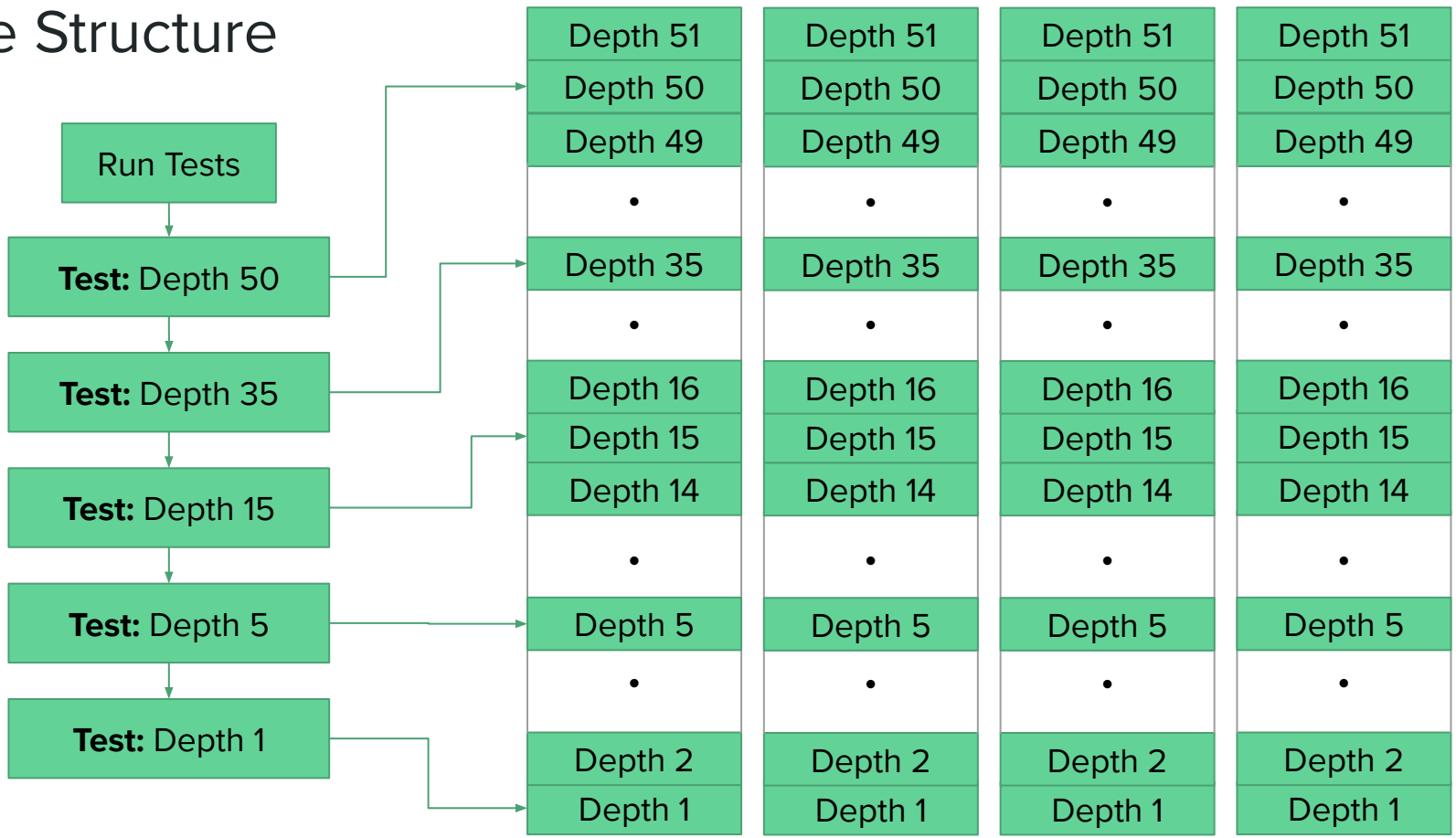
Code Structure



Code Structure



Code Structure



```
// Global side effect
std::int32_t volatile side_effect = 0;
```

```
class simple_object
{
public:
 explicit simple_object(std::int32_t value);
 void do_work();


private:
 std::int32_t m_value;
};
```

```
class destructor_object
{
public:
 explicit destructor_object(std::int32_t value);
 void do_work();

 ~destructor_object()
 {
   side_effect = side_effect + 1;
 }

private:
 std::int32_t m_value;
};
```

```
// Global side effect
std::int32_t volatile side_effect = 0;
```

```
class simple_object
{
public:
 explicit simple_object(std::int32_t value);
 void do_work();

private:
 std::int32_t m_value;
};
```

```
class destructor_object
{
public:
 explicit destructor_object(std::int32_t value);
 void do_work();

 ~destructor_object()
 {
   side_effect = side_effect + 1;
 }

private:
 std::int32_t m_value;
};
```

Benchmark depth function

```
[[gnu::noinline]]
int depth_50_percent_050()
{
 simple_object obj(side_effect >> 8);
 obj.do_work();

 std::array<std::int8_t volatile, 10> preserve_frame = { 8 };

 int const result = depth_49_percent_050();

 // Use the variable after the call
 if (preserve_frame[0] < 0) {
   // Never executed but prevents tail-call optimization
   return -1;
 }
 return result + side_effect;
}
```

Benchmark depth function

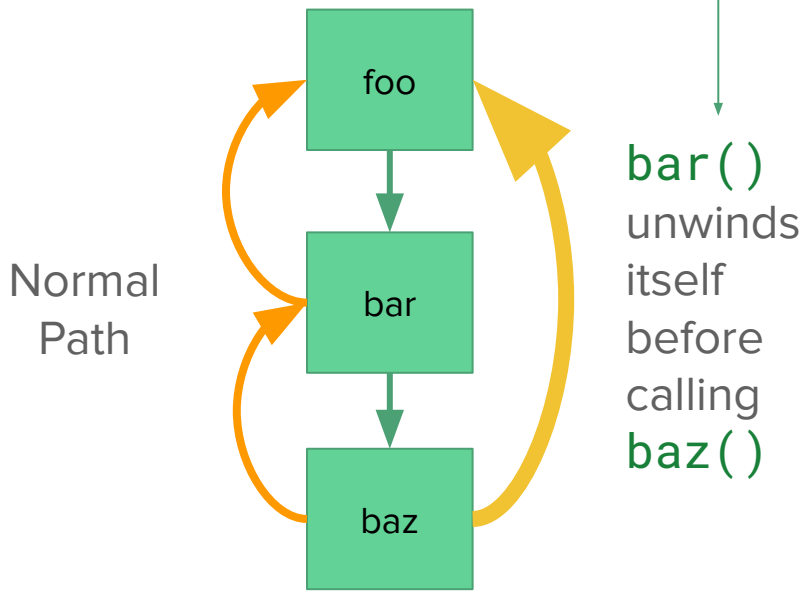
```
[[gnu::noinline]]
int depth_50_percent_050()
{
 simple_object obj(side_effect >> 8);
 obj.do_work();

 std::array<std::int8_t volatile, 10> preserve_frame = { 8 };

 int const result = depth_49_percent_050();

 // Use the variable after the call
 if (preserve_frame[0] < 0) {
   // Never executed but prevents tail-call optimization
   return -1;
 }
 return result + side_effect;
}
```

Benchmark depth function

```
[[gnu::noinline]]
int depth_50_percent_050()
{
 simple_object obj(side_effect >> 8);
 obj.do_work();

 std::array<std::int8_t volatile, 10> preserve_frame = { 8 };

 int const result = depth_49_percent_050();

 // Use the variable after the call
 if (preserve_frame[0] < 0) {
   // Never executed but prevents tail-call optimization
   return -1;
 }
 return result + side_effect;
}
```

Preventing Tail Call Optimizations

```
[[gnu::noinline]]
int depth_50_percent_050()
{
 simple_object obj(side_effect >> 8);
 obj.do_work();

 std::array<std::int8_t volatile, 10> preserve_frame = { 8 };

 int const result = depth_49_percent_050();

 // Use the variable after the call
 if (preserve_frame[0] < 0) {
   // Never executed but prevents tail-call optimization
   return -1;
 }
 return result + side_effect;
}
```



Benchmark depth function

```cpp
[[gnu::noinline]]
int depth_50_percent_050()
{
 simple_object obj(side_effect >> 8);
 obj.do_work();

 std::array<std::int8_t volatile, 10> preserve_frame = { 8 };

 int const result = depth_49_percent_050();

 // Use the variable after the call
 if (preserve_frame[0] < 0) {
   // Never executed but prevents tail-call optimization
   return -1;
 }
 return result + side_effect;
}
```

```cpp
[[gnu::noinline]]
int depth_49_percent_050()
{
 destructor_object obj(side_effect >> 8);
 obj.do_work();

 std::array<std::int8_t volatile, 4> preserve_frame = { 8 };

 int const result = depth_48_percent_050();

 // Use the variable after the call
 if (preserve_frame[0] < 0) {
   // Never executed but prevents tail-call optimization
   return -1;
 }
 return result + side_effect;
}
```

Error types (4B, 16B, 65B)

```
struct test_error_4
{
 std::array<std::uint8_t, 4> data;
};
struct test_error_16
{
 std::array<std::uint8_t, 16> data;
};
struct test_error_65
{
 std::array<std::uint8_t, 65> data;
};
```

- Trivially relocatable (can be **memcpy**)
- Trivially constructible
- Trivially destructible
- Array of 4B has the **same performance** as `int` or `std::errc`

```cpp
[[gnu::noinline]]
std::expected<int, test_error_04>
   depth_50_error_04_percent_050()
{
 simple_object obj(side_effect >> 8);
 obj.do_work();

 std::array<std::int8_t volatile, 10> preserve_frame = { 8 };

 auto const result = depth_49_error_04_percent_050();

 if (!result) {
   return std::unexpected(result.error());
 }

 // Use the variable after the call
 if (preserve_frame[0] < 0) {
   // Never executed but prevents tail-call optimization
   return std::unexpected(result.error());
 }

 return result.value() + side_effect;
}
```

```cpp
[[gnu::noinline]]
int depth_50_percent_050()
{
 simple_object obj(side_effect >> 8);
 obj.do_work();

 std::array<std::int8_t volatile, 10> preserve_frame = { 8 };

 int const result = depth_49_percent_050();

 // Use the variable after the call
 if (preserve_frame[0] < 0) {
   // Never executed but prevents tail-call optimization
   return -1;
 }
 return result + side_effect;
}
```

std::expected

```cpp
[[gnu::noinline]]
std::expected<int, test_error_04>
   depth_50_error_04_percent_050()
{
 simple_object obj(side_effect >> 8);
 obj.do_work();

 std::array<std::int8_t volatile, 10> preserve_frame = { 8 };

 auto const result = depth_49_error_04_percent_050();

 if (!result) {
   return std::unexpected(result.error());
 }

 // Use the variable after the call
 if (preserve_frame[0] < 0) {
   // Never executed but prevents tail-call optimization
   return std::unexpected(result.error());
 }

 return result.value() + side_effect;
}
```

```cpp
[[gnu::noinline]]
int depth_50_percent_050()
{
 simple_object obj(side_effect >> 8);
 obj.do_work();

 std::array<std::int8_t volatile, 10> preserve_frame = { 8 };

 int const result = depth_49_percent_050();

 // Use the variable after the call
 if (preserve_frame[0] < 0) {
   // Never executed but prevents tail-call optimization
   return -1;
 }
 return result + side_effect;
}
```

std::expected

Emitting the Error

```
[[gnu::noinline]]
std::expected<int, test_error_04> depth_01_error_04_percent_000()
{
 simple_object obj(side_effect >> 8);
 obj.do_work();
 std::array<std::int8_t volatile, 14> preserve_frame = { 8 };
 // This is where the error originates
 if (side_effect > 0) {
  start();
   return std::unexpected(test_error_04{
       .data = { 0xDE, 0xAD, 0xBE, 0xEF } });
 }
 // Use the variable after the call
 if (preserve_frame[0] < 0) {
   // Never executed but prevents tail-call optimization
   return -1;
 }
 return side_effect + 1;
}
```

```
[[gnu::noinline]]
int depth_01_percent_000() {
 simple_object obj(side_effect >> 8);
 obj.do_work();
 std::array<std::int8_t volatile, 14> preserve_frame = { 8 };
 // Use the variable after the call
 if (preserve_frame[0] < 0) {
   // Never executed but prevents tail-call optimization
   return -1;
 }
 if (error_size_select == 4) {
  start();
   throw test_error_04{.data = {0xDE, 0xAD, 0xBE, 0xEF}};
 }
 if (error_size_select == 16) {
  start();
   throw test_error_16{.data = {0xDE, 0xAD, 0xBE, 0xEF}};
 }
 if (error_size_select == 65) {
  start();
   throw test_error_65{.data = {0xDE, 0xAD, 0xBE, 0xEF}};
 }
 return side_effect;
}
```

Reaching the Handler

```
[[gnu::noinline]]
void run_test_depth_50_error_04_cleanup_100 ()
{
 // Ensure we will return error
 side_effect = 1;

 auto const result =
depth_50_error_04_percent_100 ();
 if (!result) {
   end();
 }

 pause();
}
```

```
// Test runner
[[gnu::noinline]]
void
run_test_depth_50_error_04_cleanup_000 ()
{
 // Ensure we will throw
 side_effect = 1;
 error_size_select = 4;

 try {
   depth_50_percent_000 ();
 } catch (test_error_04 const& e) {
   end();
 }

 pause();
}
```

How tests are executed

```
// Test runner
void run_test()
{
 run_test_error_04_cleanup_000();
 run_test_error_04_cleanup_025();
 run_test_error_04_cleanup_050();
 run_test_error_04_cleanup_100();
 run_test_error_16_cleanup_000();
 run_test_error_16_cleanup_025();
 run_test_error_16_cleanup_050();
 run_test_error_16_cleanup_100();
 run_test_error_65_cleanup_000();
 run_test_error_65_cleanup_025();
 run_test_error_65_cleanup_050();
 run_test_error_65_cleanup_100();
}
```

```
void run_test_error_04_cleanup_000()
{
 run_test_depth_50_error_04_cleanup_000();
 run_test_depth_35_error_04_cleanup_000();
 run_test_depth_15_error_04_cleanup_000();
 run_test_depth_05_error_04_cleanup_000();
 run_test_depth_01_error_04_cleanup_000();
}
```



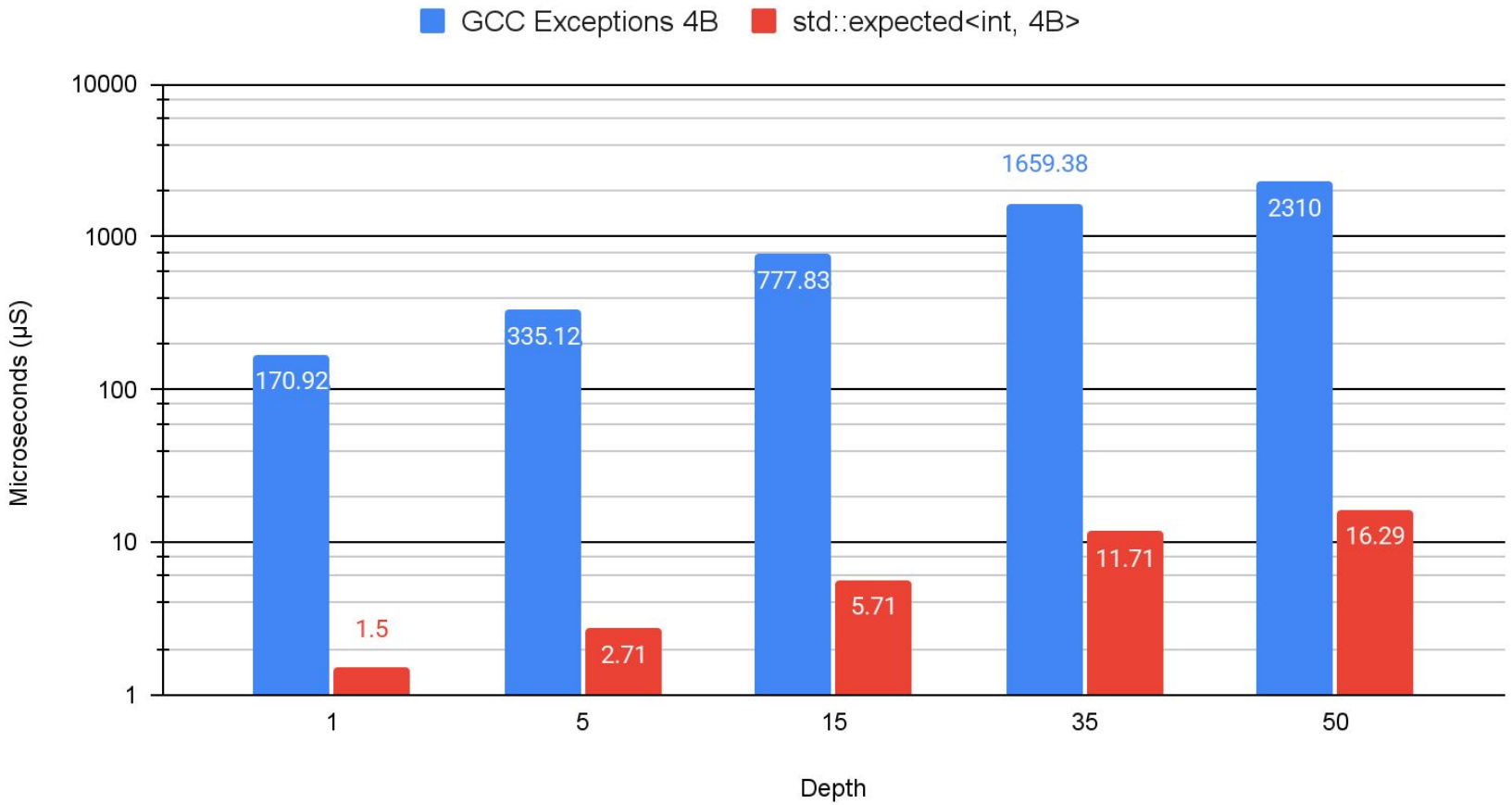
```
void run_test_error_65_cleanup_100()
{
 run_test_depth_50_error_65_cleanup_100();
 run_test_depth_35_error_65_cleanup_100();
 run_test_depth_15_error_65_cleanup_100();
 run_test_depth_05_error_65_cleanup_100();
 run_test_depth_01_error_65_cleanup_100();
}
```

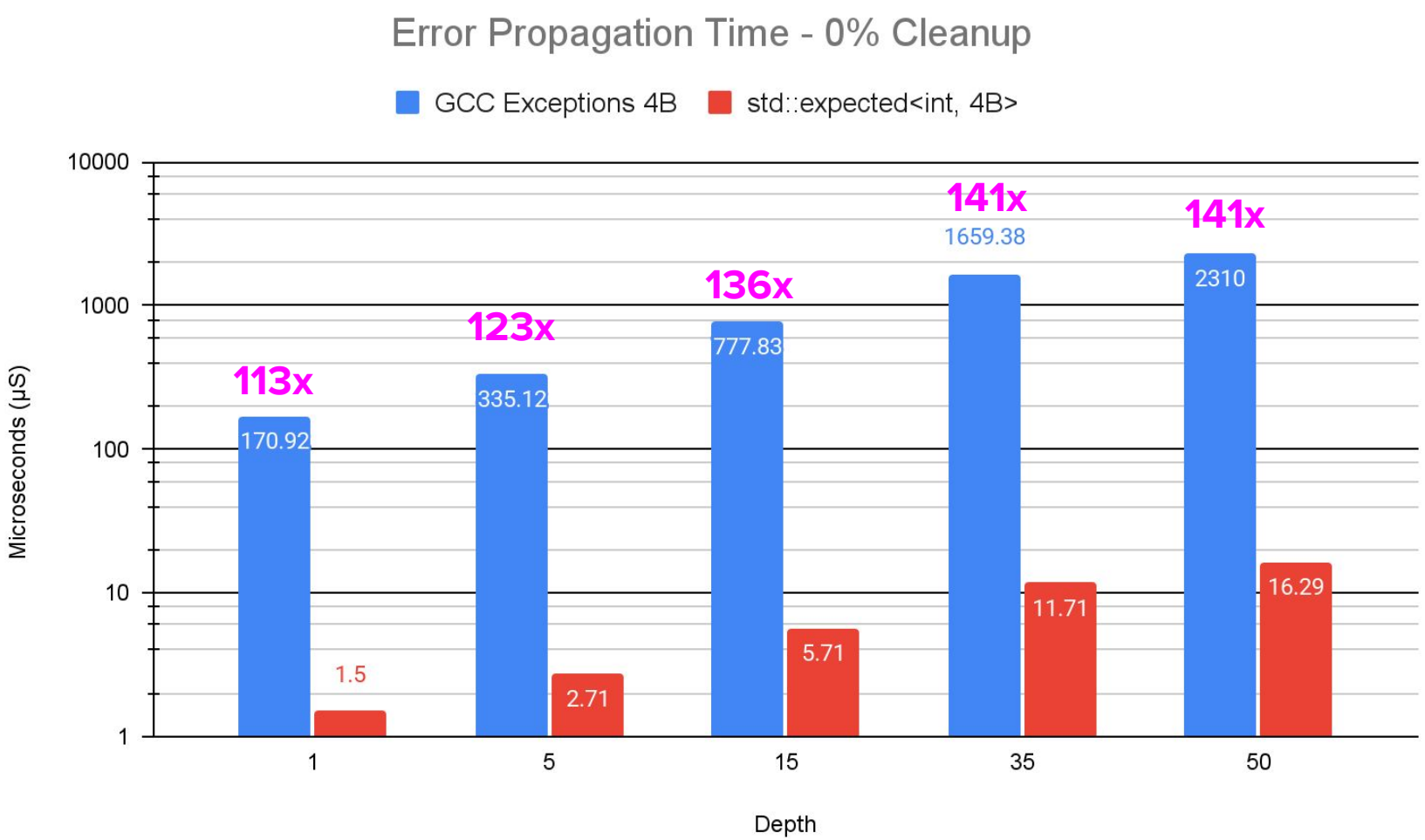
Testing Script Steps

Results!

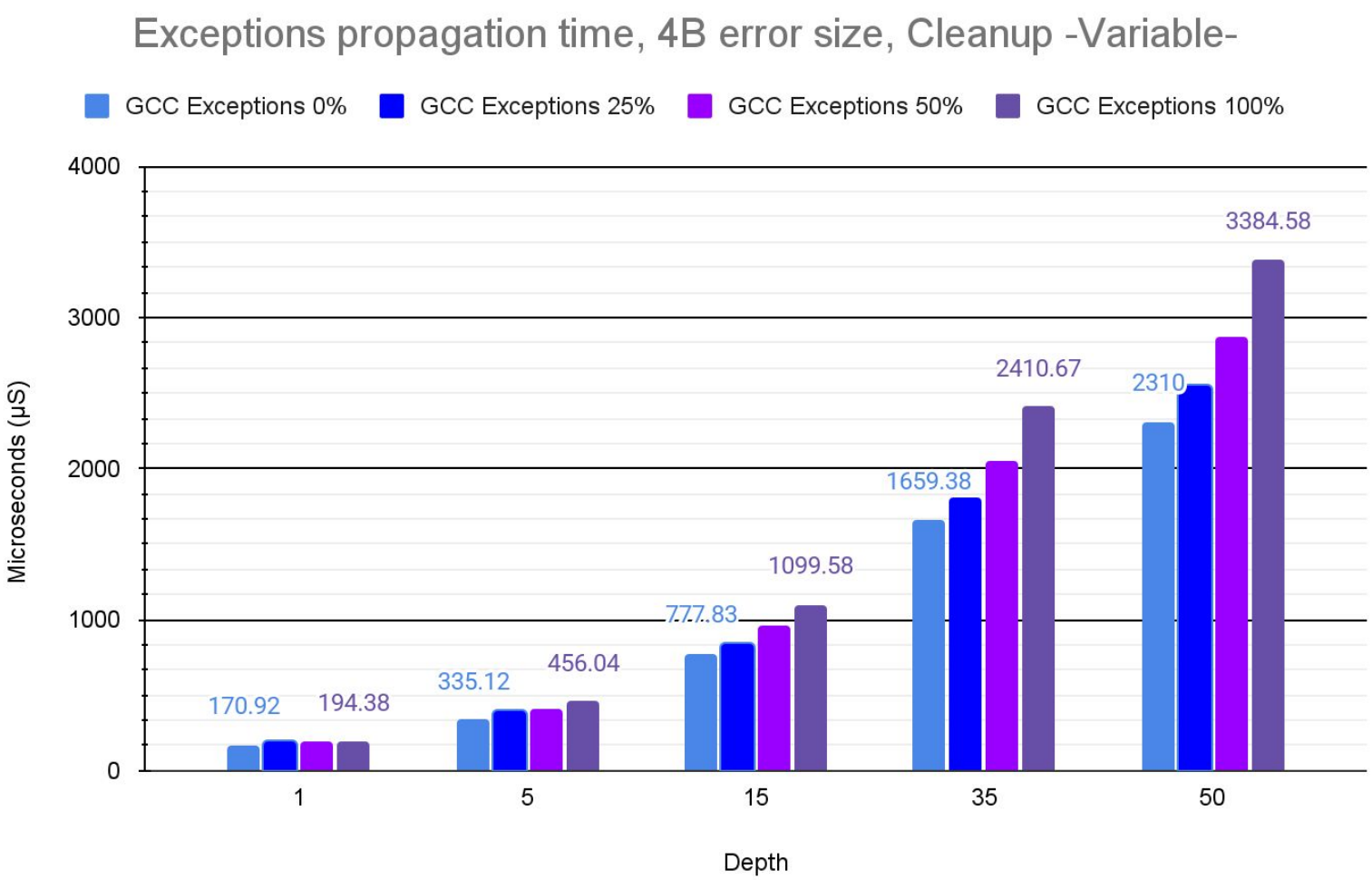
*LOG SCALE

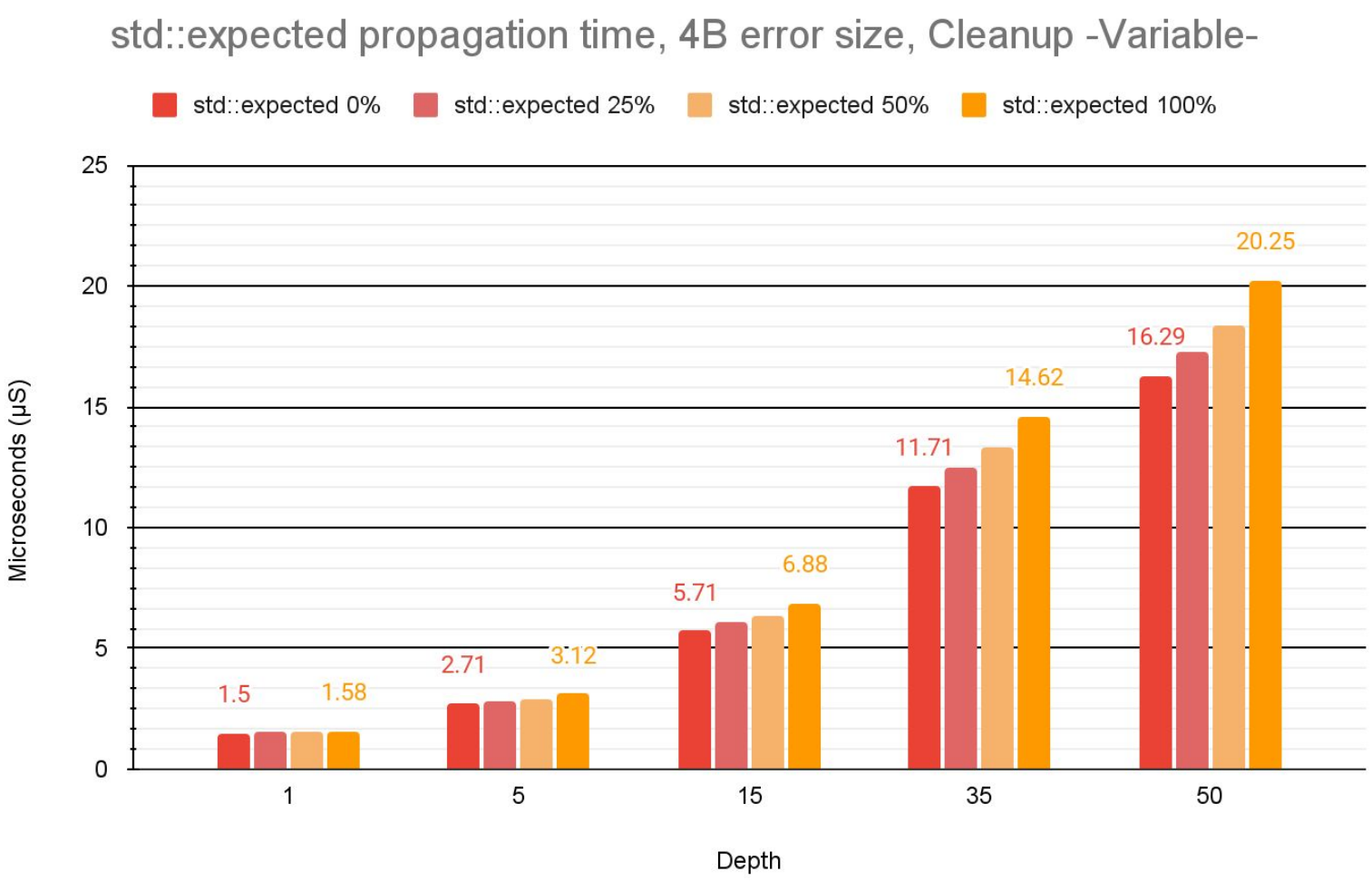
Error Propagation Time - 0% Cleanup



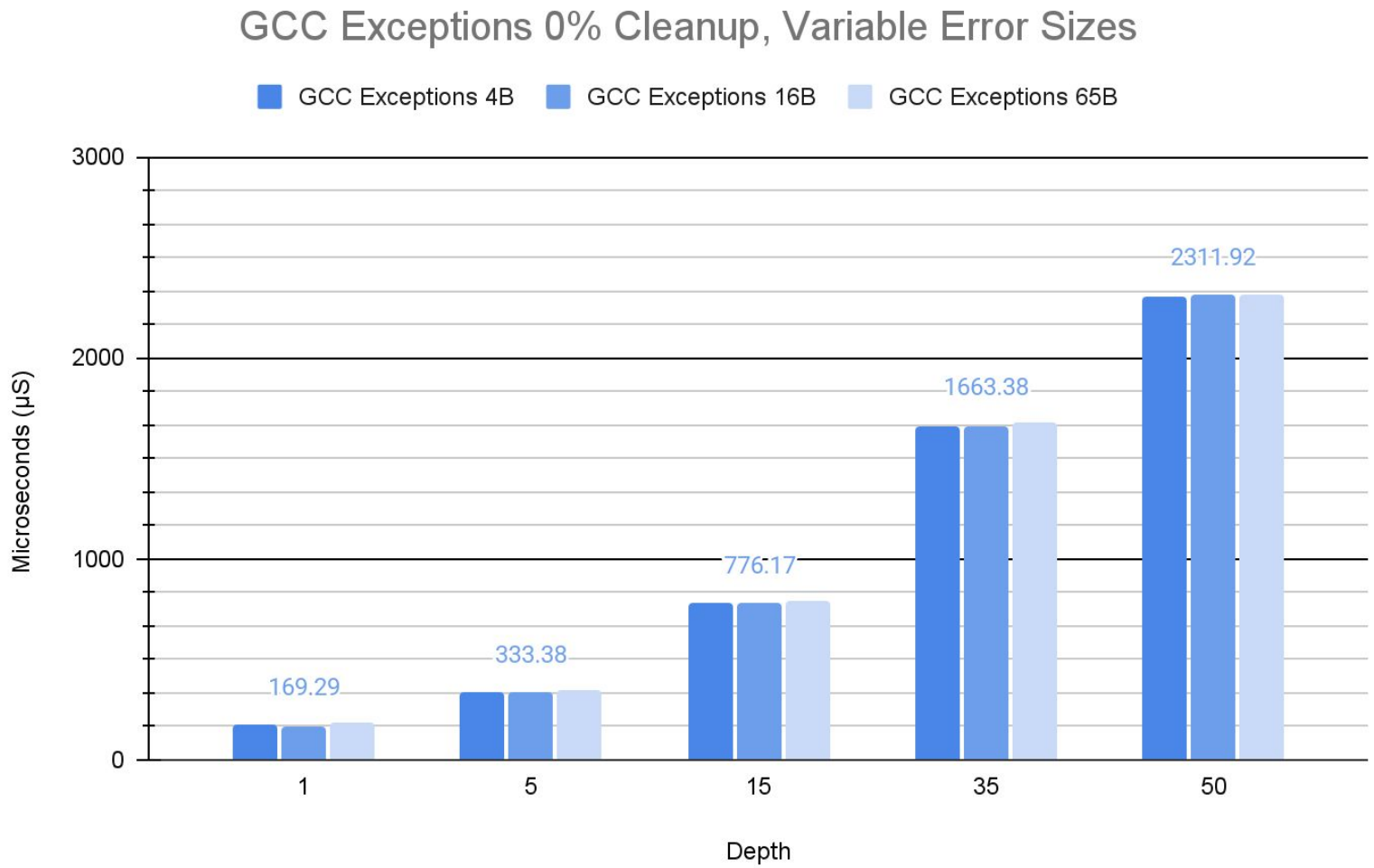

Error Propagation Time - 0% Cleanup
GCC Exceptions 4B
std::expected<int, 4B>
Microseconds (µS)
10000
1000
100
10
1
113x
170.92
1.5
123x
335.12
2.71
136x
777.83
5.71
141x
1659.38
11.71
141x
2310
16.29
1
5
15
35
50
Depth

How does cleanup % effect things?

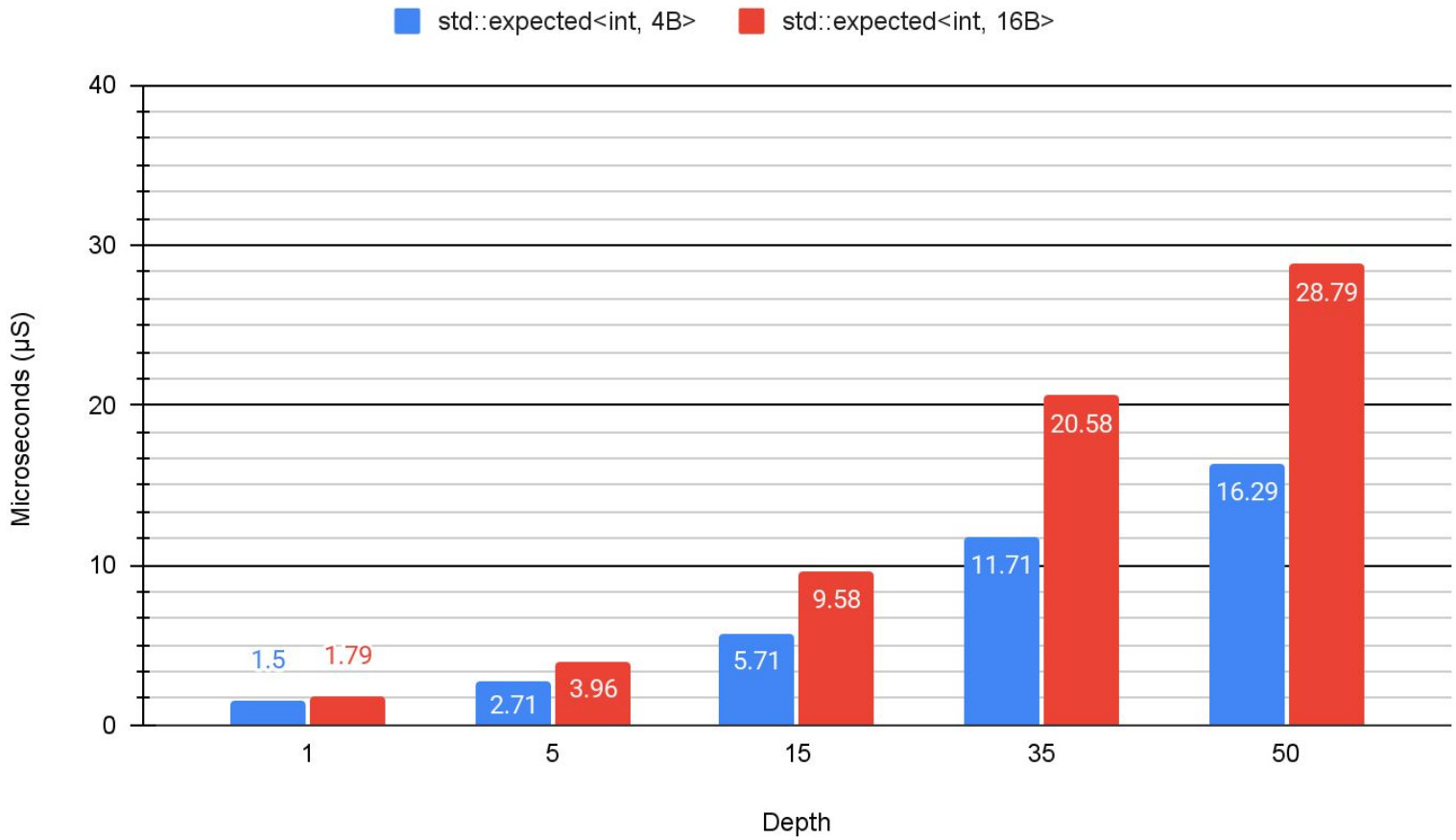

Exceptions propagation time, 4B error size, Cleanup -Variable-
GCC Exceptions 0%
GCC Exceptions 25%
GCC Exceptions 50%
GCC Exceptions 100%
Microseconds (µS)
4000
3000
2000
1000
0
170.92
194.38
335.12
456.04
777.83
1099.58
1659.38
2410.67
2310
3384.58
1
5
15
35
50
Depth

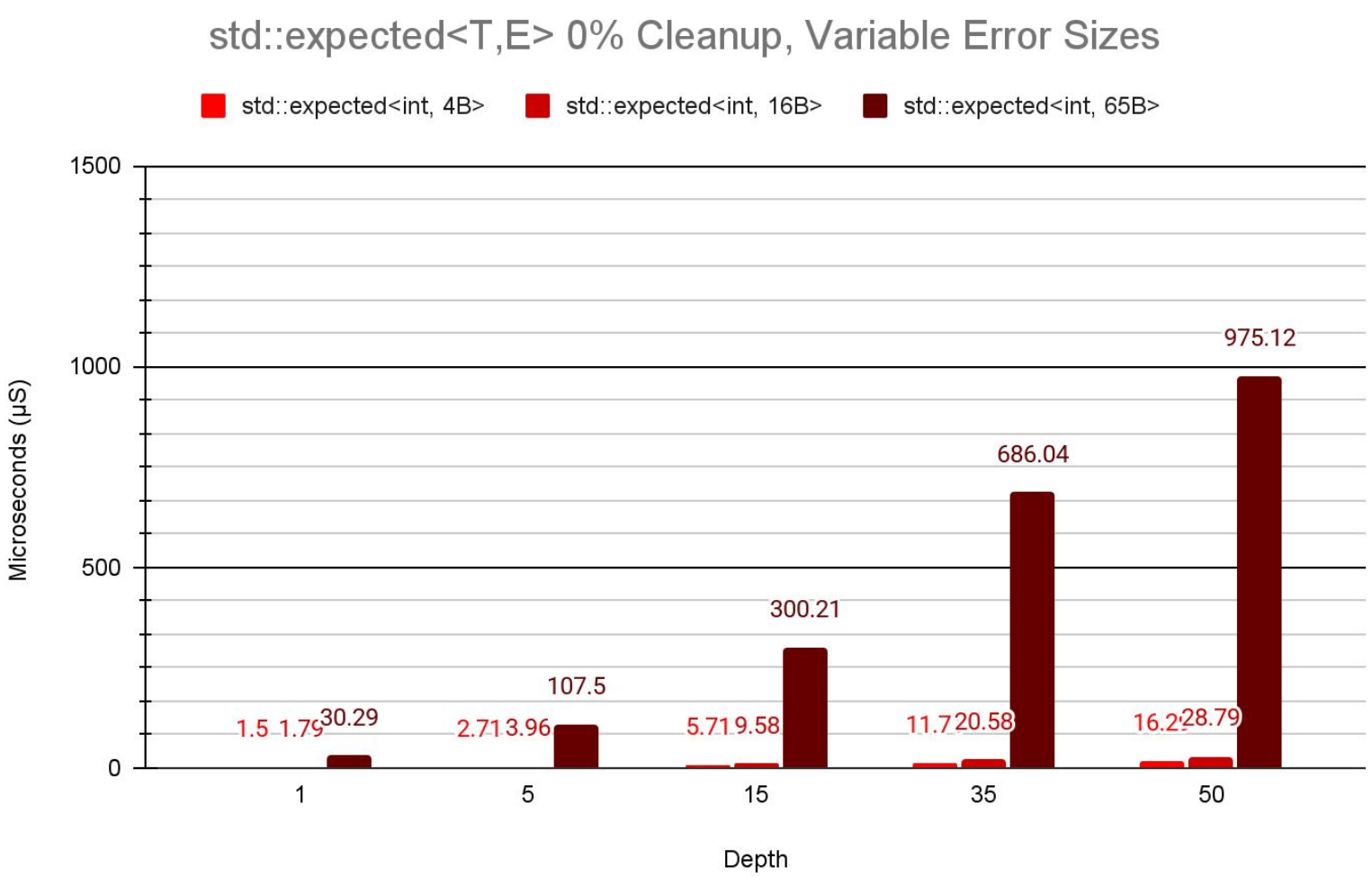

std::expected propagation time, 4B error size, Cleanup -Variable-
std::expected 0%
std::expected 25%
std::expected 50%
std::expected 100%
Microseconds (µS)
25
20
15
10
5
0
1.5
1.58
2.71
3.12
5.71
6.88
11.71
14.62
16.29
20.25
1
5
15
35
50
Depth

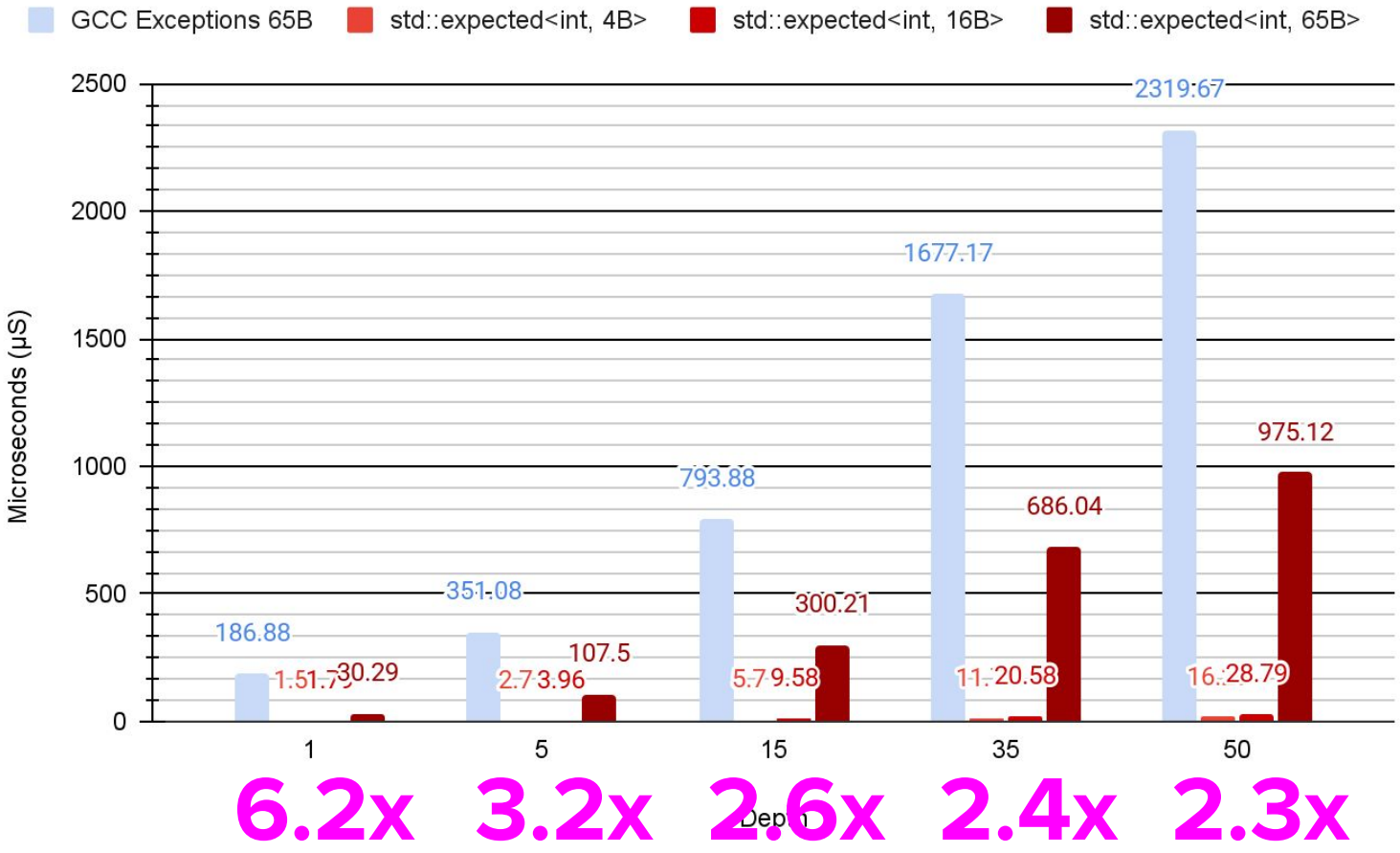
Error object size?


GCC Exceptions 0% Cleanup, Variable Error Sizes
GCC Exceptions 4B
GCC Exceptions 16B
GCC Exceptions 65B
3000
2000
1000
0
Microseconds (µS)
169.29
333.38
776.17
1663.38
2311.92
1
5
15
35
50
Depth

*LINEAR SCALE std::expected<T,E> 0% Cleanup, Variable Error Sizes




std::expected<T,E> 0% Cleanup, Variable Error Sizes
std::expected<int, 4B>
std::expected<int, 16B>
std::expected<int, 65B>
Microseconds (µS)
1500
1000
500
0
1.5
1.79
30.29
2.71
3.96
107.5
5.71
9.58
300.21
11.7
20.58
686.04
16.2
28.79
975.12
1
5
15
35
50
Depth


Propagation Time: 0% Cleanup, errors size [4B, 16B, 65B]
GCC Exceptions 65B
std::expected<int, 4B>
std::expected<int, 16B>
std::expected<int, 65B>
Microseconds (µS)
2500
2000
1500
1000
500
0
186.88
1.51
7.9
30.29
351.08
2.7
3.96
107.5
793.88
5.7
9.58
300.21
1677.17
11.
20.58
686.04
2319.67
16.
28.79
975.12
1
5
15
35
50
6.2x
3.2x
2.6x
2.4x
2.3x

Why the slow down?

```
08000f24 <depth_13_error_65_percent_000()>:
@ Other stuff…
8000f44: f7ff ffa8   bl  8000e98
<depth_12_error_65_percent_000()>
8000f48: f89d 608c   ldrb.w  r6, [sp, #140]  @ 0x8c
8000f4c: b976        cbnz  r6, 8000f6c
<depth_13_error_65_percent_000()+0x48>
8000f4e: 2241        movs  r2, #65 @ 0x41
8000f50: a912        add r1, sp, #72 @ 0x48
8000f52: a801        add r0, sp, #4
8000f54: f00a f99c   bl  800b290 <memcpy>
8000f58: 2241        movs  r2, #65 @ 0x41
8000f5a: 4628        mov r0, r5
8000f5c: a901        add r1, sp, #4
8000f5e: f00a f997   bl  800b290 <memcpy>
```

```
08000e80 <depth_13_error_16_percent_000()>:
@ Other stuff…
8000ea0: f7ff ffae   bl  8000e00
<depth_12_error_16_percent_000()>
8000ea4: f89d 3024   ldrb.w  r3, [sp, #36] @ 0x24
8000ea8: b95b        cbnz  r3, 8000ec2
<depth_13_error_16_percent_000()+0x42>
8000eaa: 7423        strb  r3, [r4, #16]
8000eac: f10d 0c04   add.w ip, sp, #4
8000eb0: ab05        add r3, sp, #20
8000eb2: cb0f        ldmia r3, {r0, r1, r2, r3}
8000eb4: e88c 000f   stmia.w ip, {r0, r1, r2, r3}
8000eb8: e884 000f   stmia.w r4, {r0, r1, r2, r3}
8000ebc: 4620        mov r0, r4
8000ebe: b00b        add sp, #44 @ 0x2c
8000ec0: bd30        pop {r4, r5, pc}
```

What we've learned

- Result types are VERY fast for errors <= 64B
- Cleanup for exceptions is slow
- All runtime is linear for all parameters
- No non-determinism found with exceptions
    - Same numbers ±1us

All benchmarks going forward will use 4B (`sizeof(std::errc)`) as the error size.

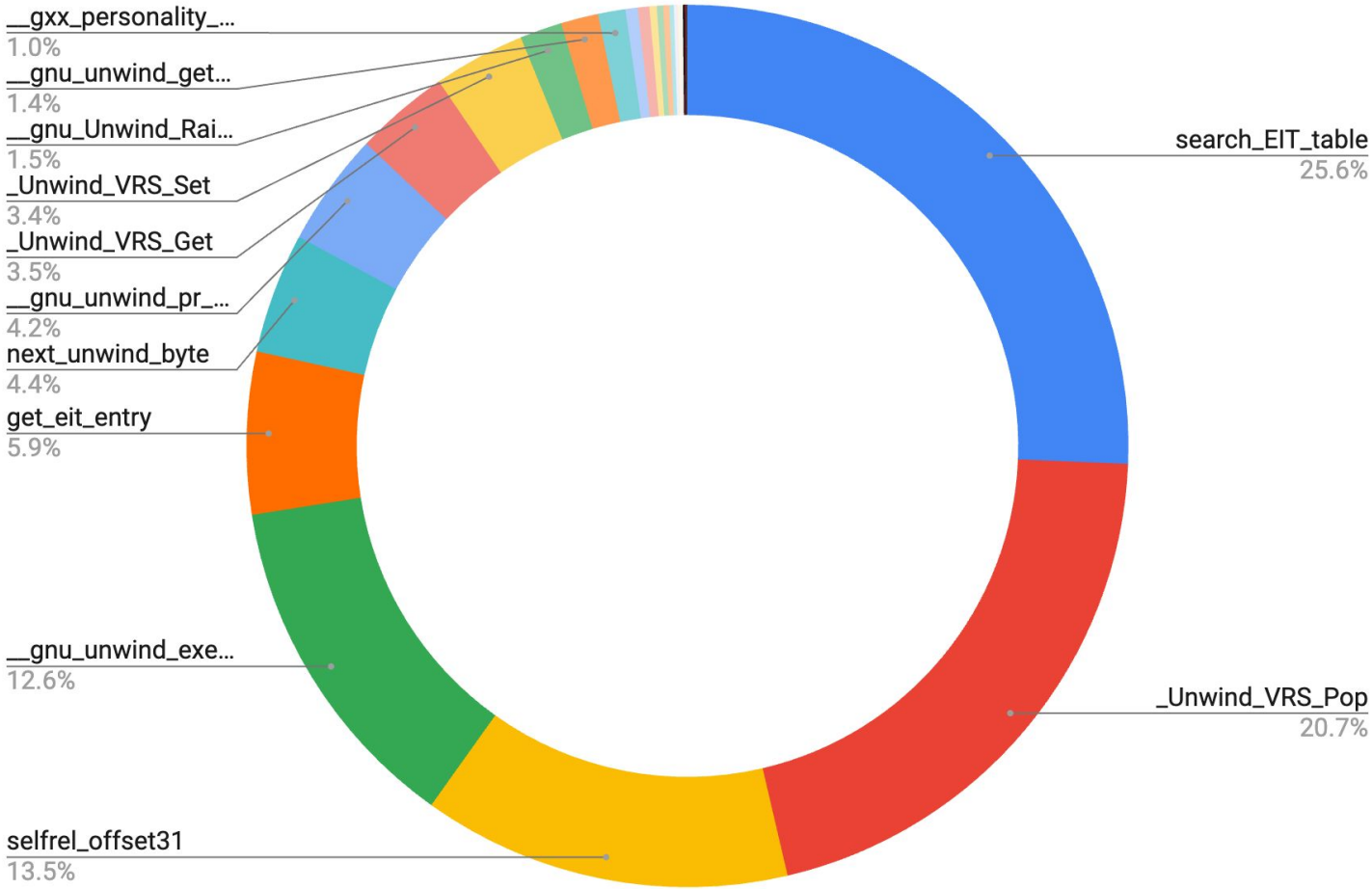
Where do I begin?

What takes up the most time?

Python: control GDB & record instructions per function

```
my_stdout.out("step,function,address")
for step in track(range(instruction_steps),
                  console=my_stderr,
                  description='Processing...'):
   response = gdbmi.write('stepi')
   for message in response:
       payload = message['payload']
       if payload and isinstance(payload, dict) and 'frame' in payload:
           if payload['frame']['func'] == "start":
               # Continue from here in order to reach the next call to throw
               gdbmi.write('continue', read_response=False)
               continue
           my_stdout.out("{},{},{}".format(
               step,
               payload['frame']['func'],
               payload['frame']['addr']
           ))
```

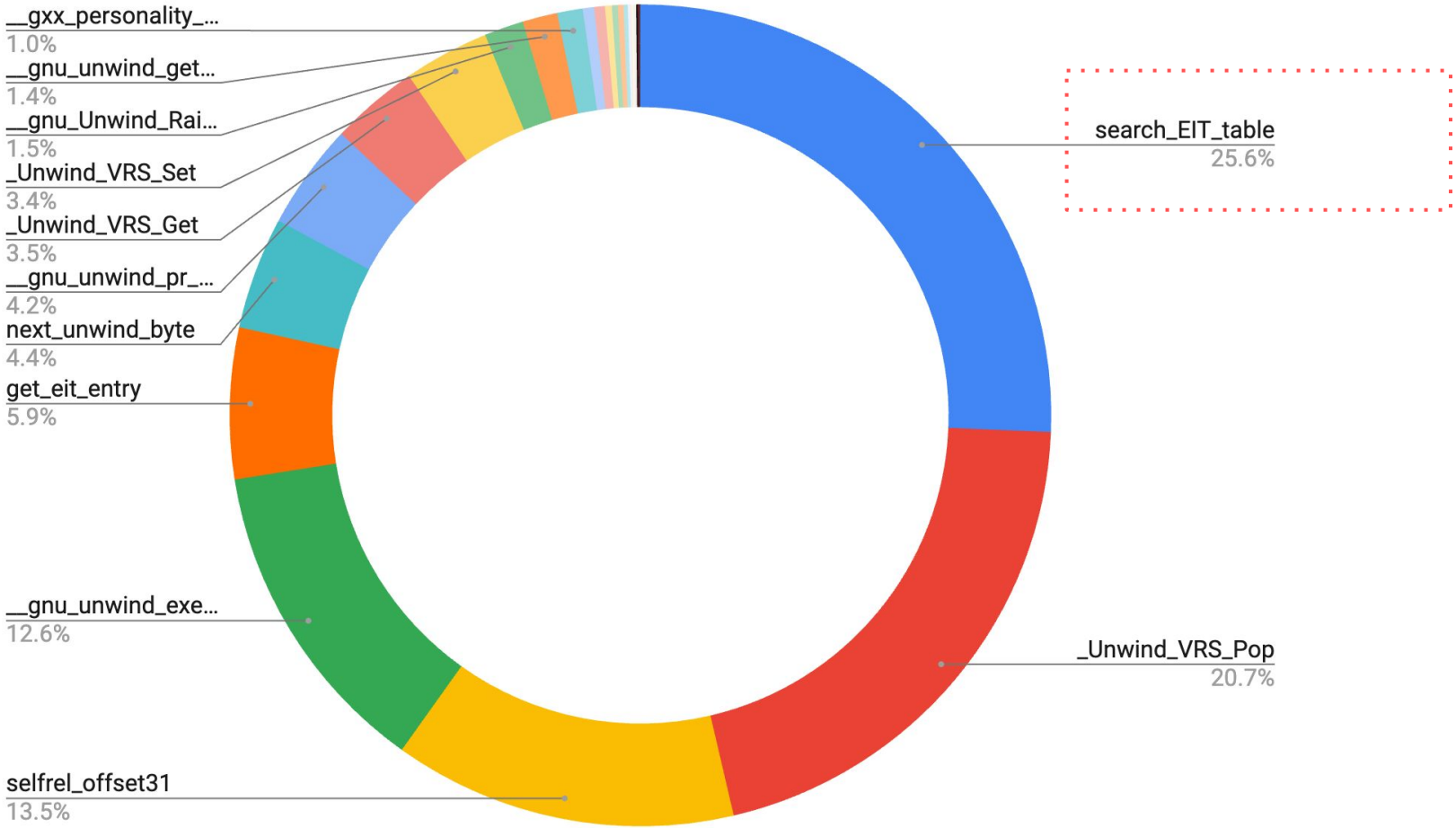
V0


__gxx_personality_...
1.0%
__gnu_unwind_get...
1.4%
__gnu_Unwind_Rai...
1.5%
_Unwind_VRS_Set
3.4%
_Unwind_VRS_Get
3.5%
__gnu_unwind_pr_...
4.2%
next_unwind_byte
4.4%
get_eit_entry
5.9%
__gnu_unwind_exe...
12.6%
selfrel_offset31
13.5%
search_EIT_table
25.6%
_Unwind_VRS_Pop
20.7%

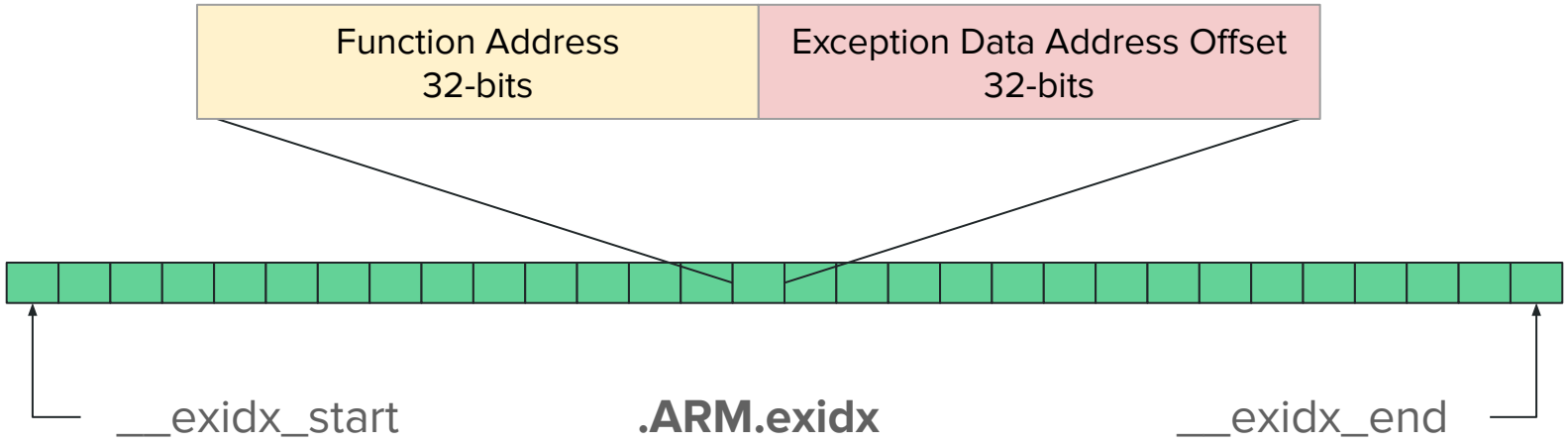
Chapter 2. 🔧

Optimizing GCC exceptions

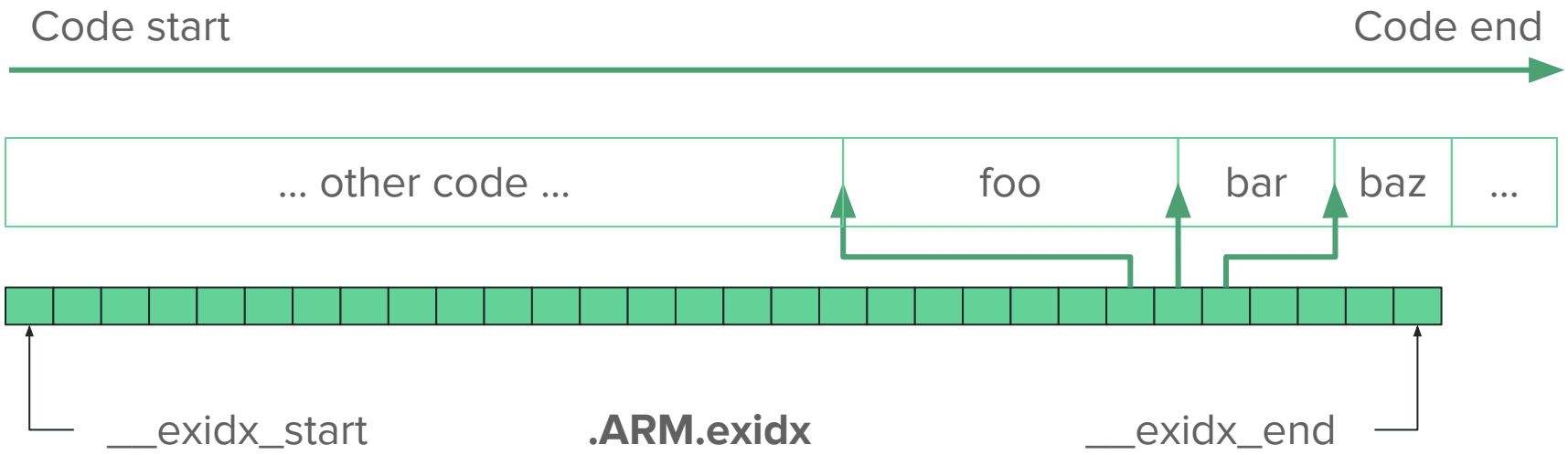
V0


__gxx_personality_...
1.0%
__gnu_unwind_get...
1.4%
__gnu_Unwind_Rai...
1.5%
_Unwind_VRS_Set
3.4%
_Unwind_VRS_Get
3.5%
__gnu_unwind_pr_...
4.2%
next_unwind_byte
4.4%
get_eit_entry
5.9%
__gnu_unwind_exe...
12.6%
selfrel_offset31
13.5%
search_EIT_table
25.6%
_Unwind_VRS_Pop
20.7%

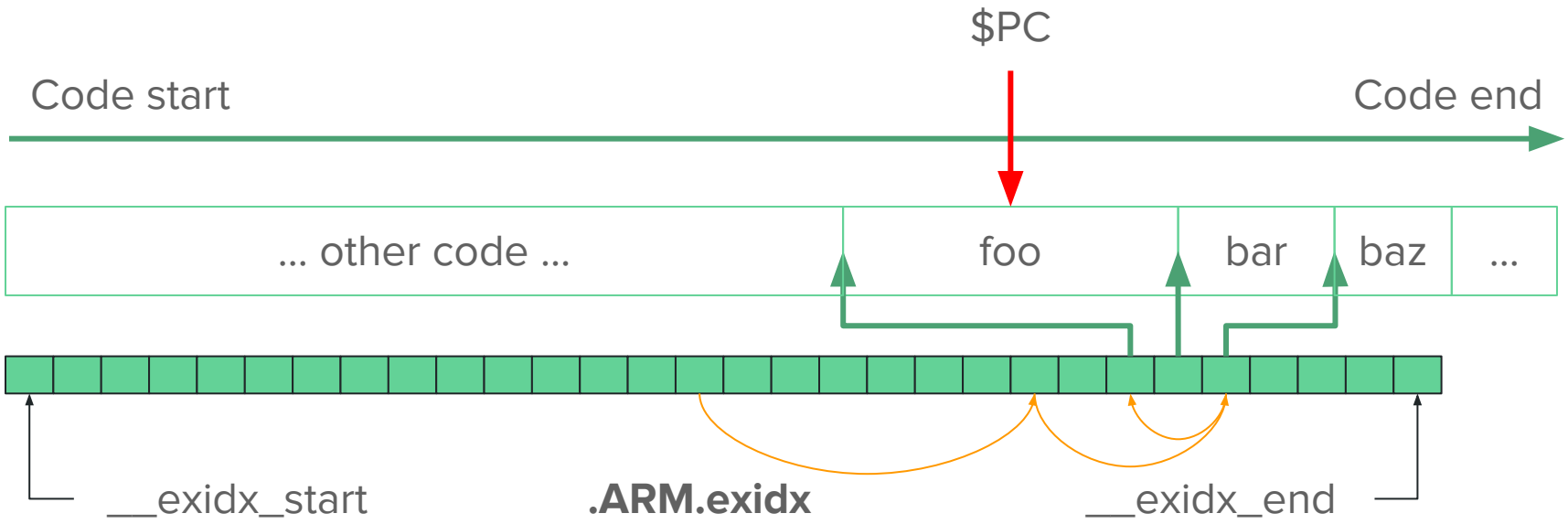
ARM Exception Index: Index Entry



ARM Exception Index: Ordering



ARM Exception Index: Ordering



```
static __EIT_entry const* search_EIT_table(
  __EIT_entry const* table, int nrec, _uw return_address)
{
 _uw next_fn;
 _uw this_fn;
 int n, left, right;
 if (nrec == 0) return (__EIT_entry*)0;
 left = 0;
 right = nrec - 1;
 while (1) {
   n = (left + right) / 2;
   this_fn = selfrel_offset31(&table[n].fnoffset);
   if (n != nrec - 1) next_fn = selfrel_offset31(&table[n + 1].fnoffset) - 1;
   else next_fn = (_uw)0 - 1;
   if (return_address < this_fn) {
     if (n == left) return (__EIT_entry*)0;
     right = n - 1;
   } else if (return_address <= next_fn) return &table[n];
   else left = n + 1;
 }
}
```

Andrei Alexandrescu

Robots Are after Your Job (or at Least the Boring Parts of It): Exploring Generative AI for C++

Commonly Taught `binary_search`

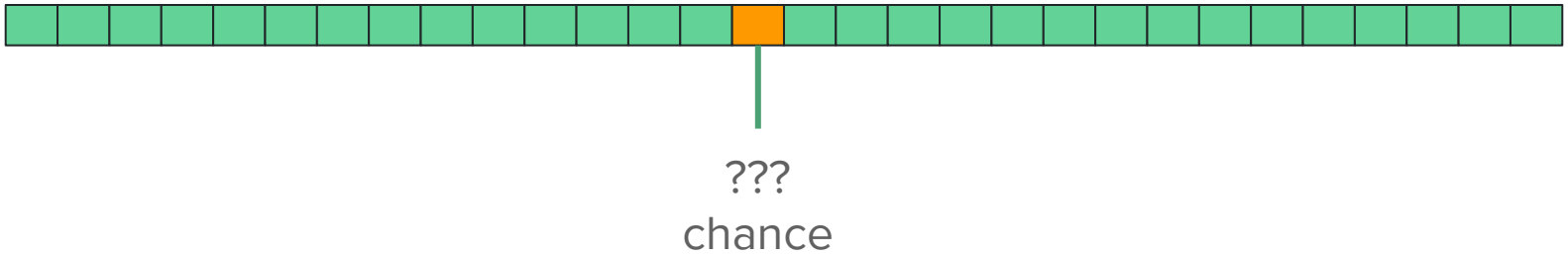
The "violet eyes" version

- First comparison in the middle of the range
- Each pass reduces searched space in half
- 1-2 tests per pass (some consider it 1)
- May return early (as soon as element found)
- Not found decision made after range collapsed

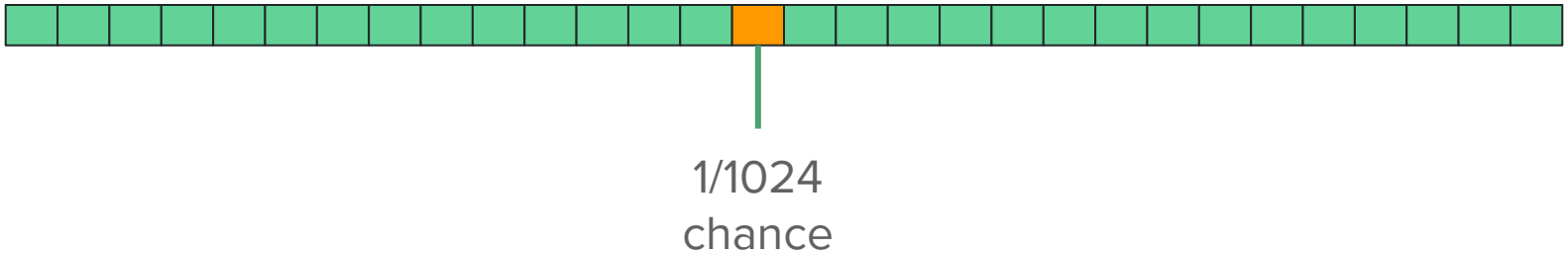
```
template <typename I, typename T>
bool
binary_search(I b, I e, const T& v) {
  while (b < e) {
    auto m = b + (e - b) / 2;
    if (v < *m) {
      e = m;
    } else if (*m < v) {
      b = m + 1;
    } else {
      return true;
    }
  }
  return false;
}
```

NVIDIA 7

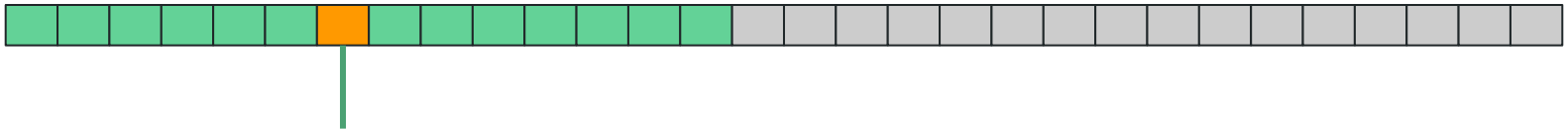
Imagine a list with 1024 entries



Imagine a list with 1024 entries



Imagine a list with 1024 entries

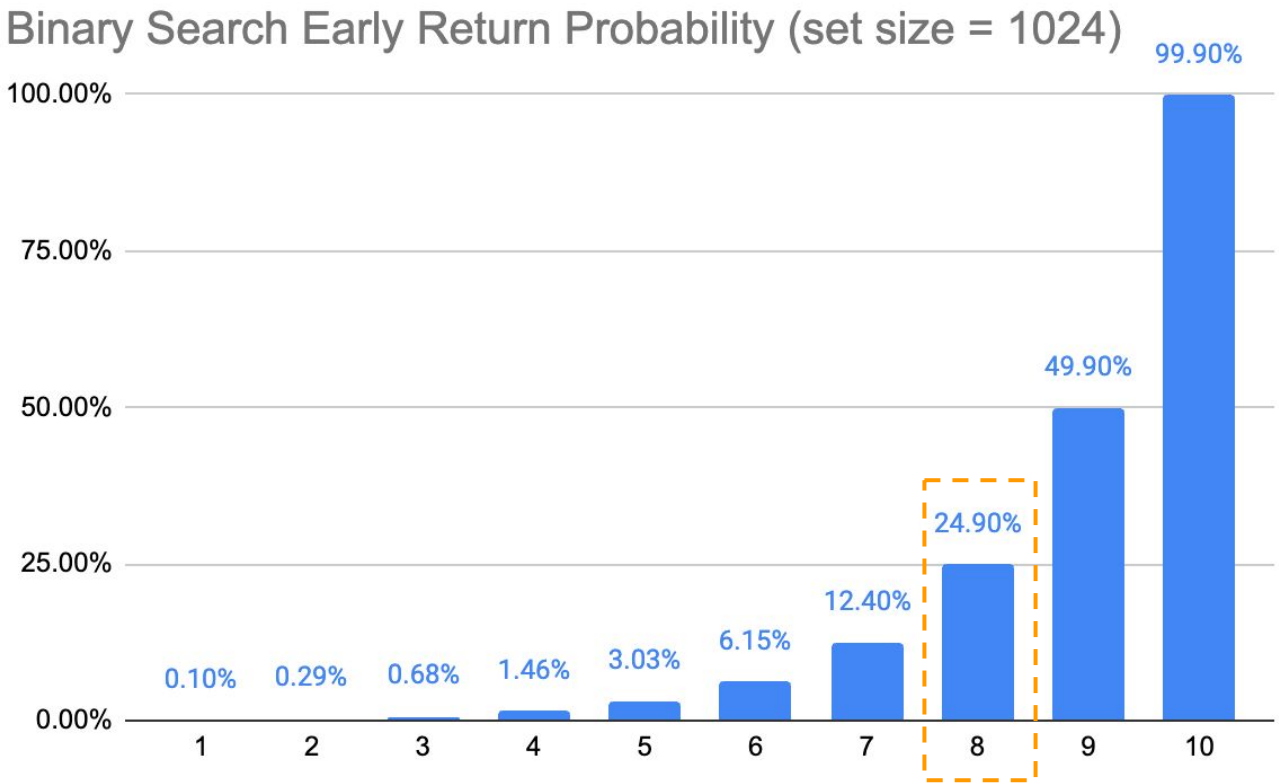


1/512
chance

Imagine a list with 1024 entries

Imagine a list with 1024 entries

24.90% chance this guess early returns



Andrei Alexandrescu

Robots Are after Your Job (or at Least the Boring Parts of It): Exploring Generative AI for C++

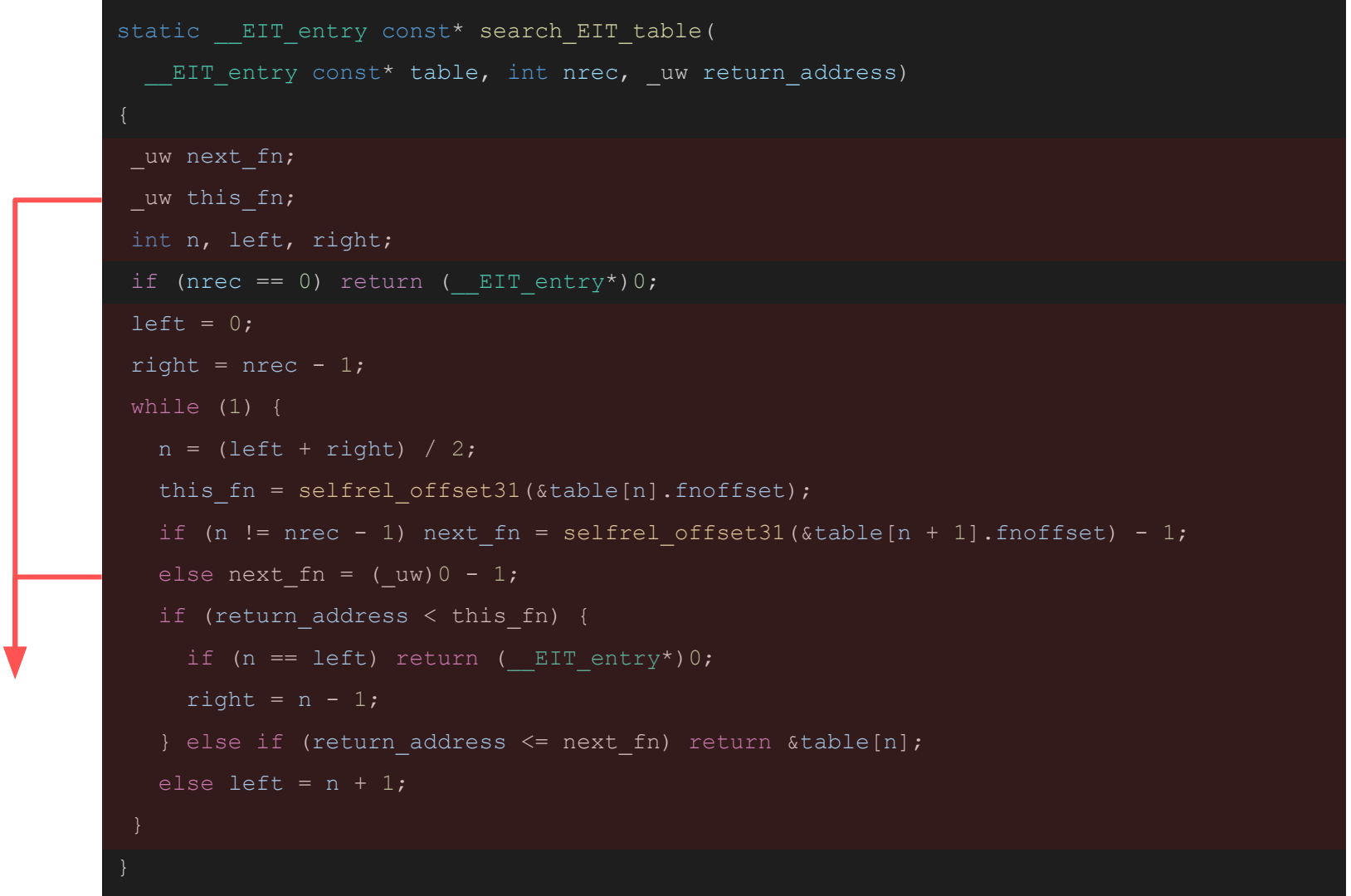
"Industrial" `std::lower_bound`

- Uses indexing
- No early return
- Always performs $log_2(n)$ steps (worst case)
  - However, one call to `less` per pass
  - Less branchy
- Everything *left* of the result is *strictly less*
- More informative/actionable than just a `bool`
- Similar `std::upper_bound` from the other side
- Finally, `std::equal_range` returns both

NVIDIA 9

```
template <class T, class I, class P>
I
lower_bound(I b, I e, const T& v, P p) {
  for (size_t size = e - b; size > 0; ) {
    size_t cut = size / 2;
    if (p(b[cut], v)) {
      b += ++cut;
      size -= cut;
    } else {
      size = cut;
    }
  }
  return b;
}
```

```
static __EIT_entry const* search_EIT_table(
  __EIT_entry const* table, int nrec, _uw return_address)
{
 _uw next_fn;
 _uw this_fn;
 int n, left, right;
 if (nrec == 0) return (__EIT_entry*)0;
 left = 0;
 right = nrec - 1;
 while (1) {
   n = (left + right) / 2;
   this_fn = selfrel_offset31(&table[n].fnoffset);
   if (n != nrec - 1) next_fn = selfrel_offset31(&table[n + 1].fnoffset) - 1;
   else next_fn = (_uw)0 - 1;
   if (return_address < this_fn) {
     if (n == left) return (__EIT_entry*)0;
     right = n - 1;
   } else if (return_address <= next_fn) return &table[n];
   else left = n + 1;
 }
}
```



```
static __EIT_entry const* search_EIT_table(
  __EIT_entry const* table, int nrec, _uw return_address)
{
 _uw next_fn;
 _uw this_fn;
 int n, left, right;
 if (nrec == 0) return (__EIT_entry*)0;
 left = 0;
 right = nrec - 1;
 while (1) {
   n = (left + right) / 2;
   this_fn = selfrel_offset31(&table[n].fnoffset);
   if (n != nrec - 1) next_fn = selfrel_offset31(&table[n + 1].fnoffset) - 1;
   else next_fn = (_uw)0 - 1;
   if (return_address < this_fn) {
     if (n == left) return (__EIT_entry*)0;
     right = n - 1;
   } else if (return_address <= next_fn) return &table[n];
   else left = n + 1;
 }
}
```

```cpp
#include <span>
#include <algorithm>

__EIT_entry const* search_EIT_table(__EIT_entry const* table,
                                    int nrec,
                                    std::uint32_t return_address)
 {
   if (nrec == 0) {
     return nullptr;
   }
   auto const& next_entry = std::ranges::upper_bound(
     std::span(table, nrec), return_address, eit_entry_less_than{});
   auto const& actual_entry = *(next_entry - 1);
   return &actual_entry;
 }
```

```
#include <span>
#include <algorithm>

__EIT_entry const* search_EIT_table(__EIT_entry const* table,
                                    int nrec,
                                    std::uint32_t return_address)
 {
   if (nrec == 0) {
     return nullptr;
   }
   start_cycles = uptime();
   auto const& next_entry = std::ranges::upper_bound(
     std::span(table, nrec), return_address, eit_entry_less_than{});
   auto const& actual_entry = *(next_entry - 1);
   end_cycles = uptime();
   return &actual_entry;
 }
```

RESULT! For 710 functions

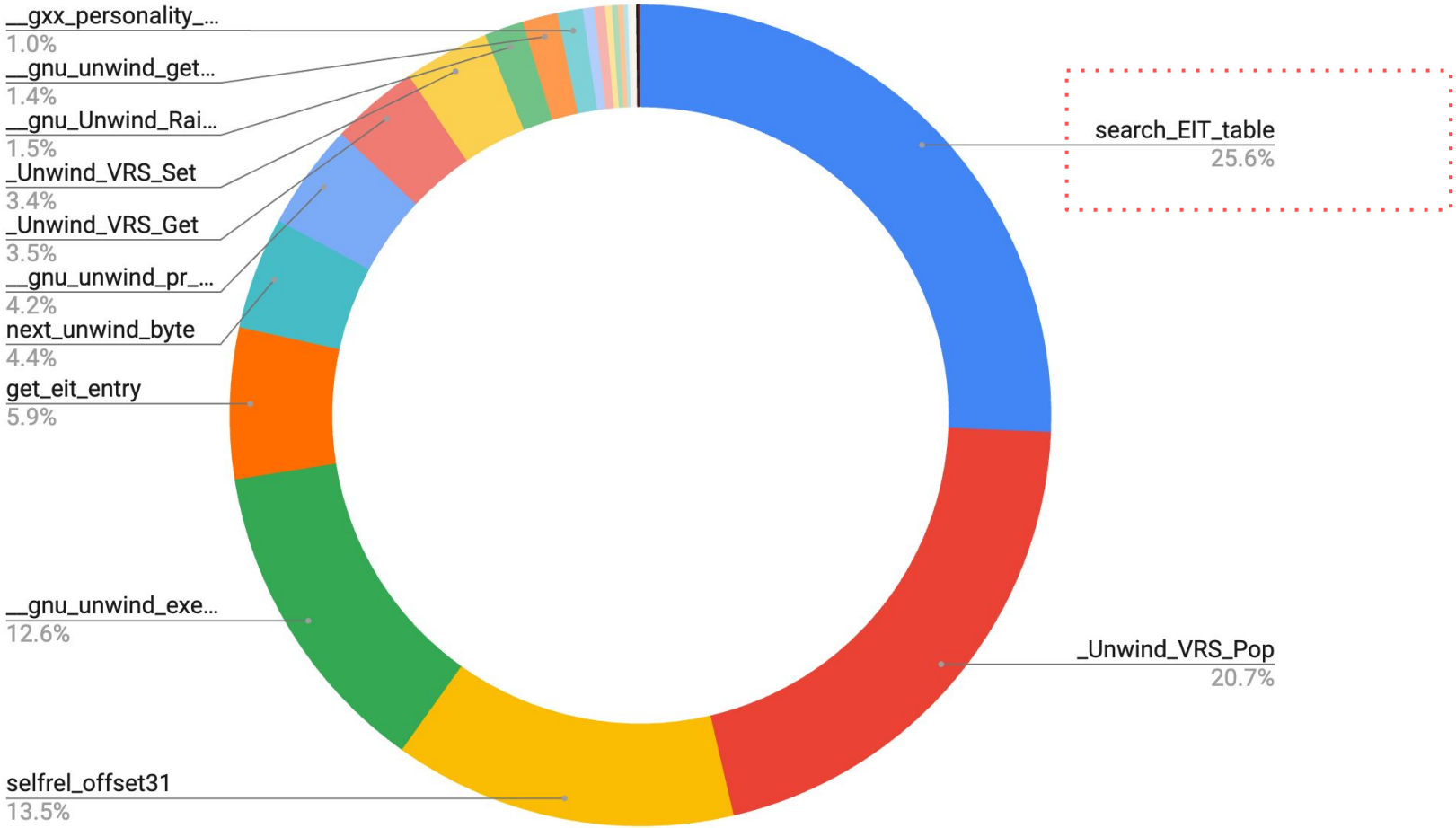
	Cycles		
Implementation	**Average**	**Minimum**	**Maximum**
GCC	379	148	477

RESULT! For 710 functions

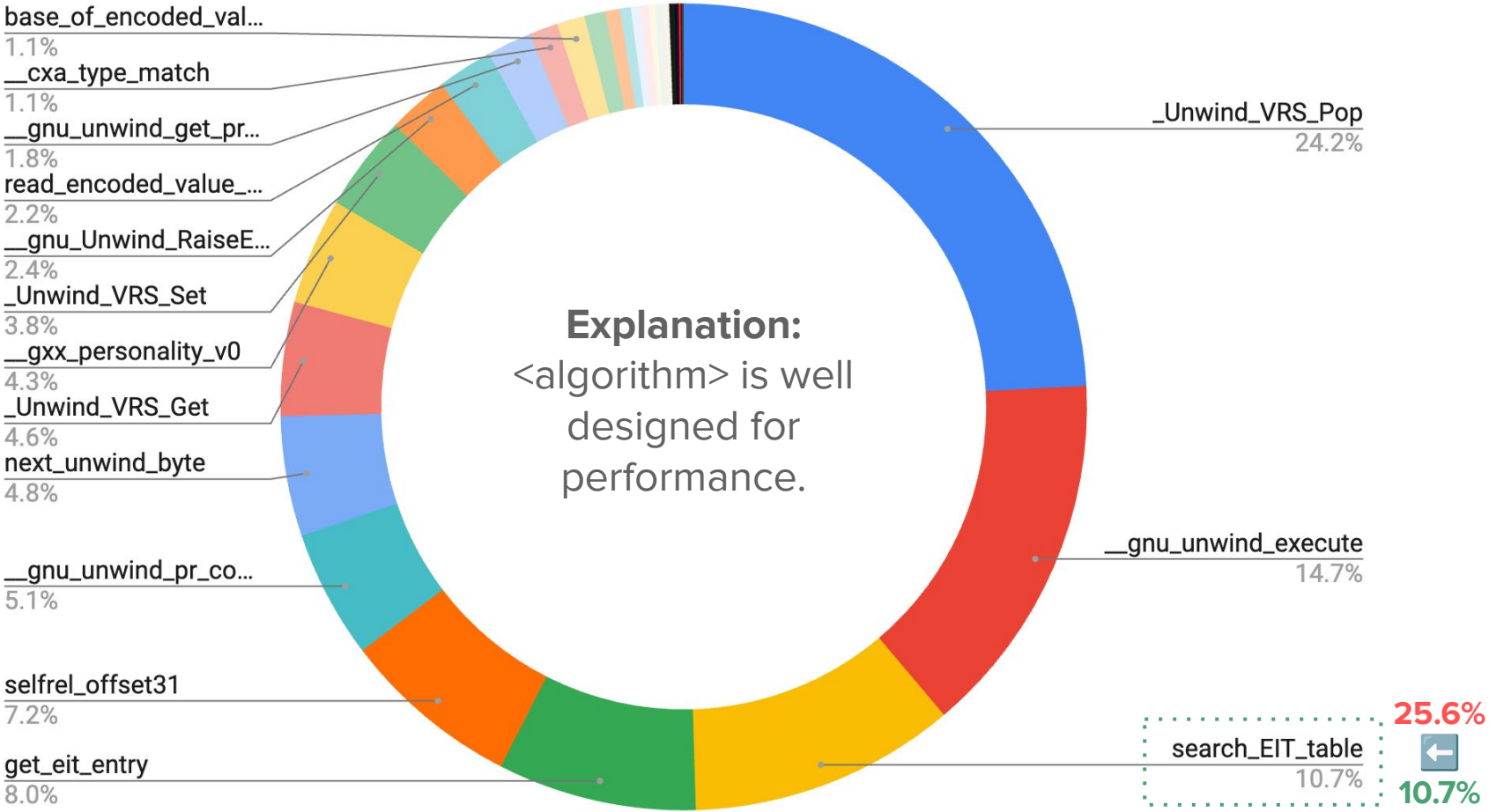
	Cycles		
Implementation	**Average**	**Minimum**	**Maximum**
GCC ❌	379	148	477
std::upper_bound ✅	270	261	288

-28.8% the needed cycles on average

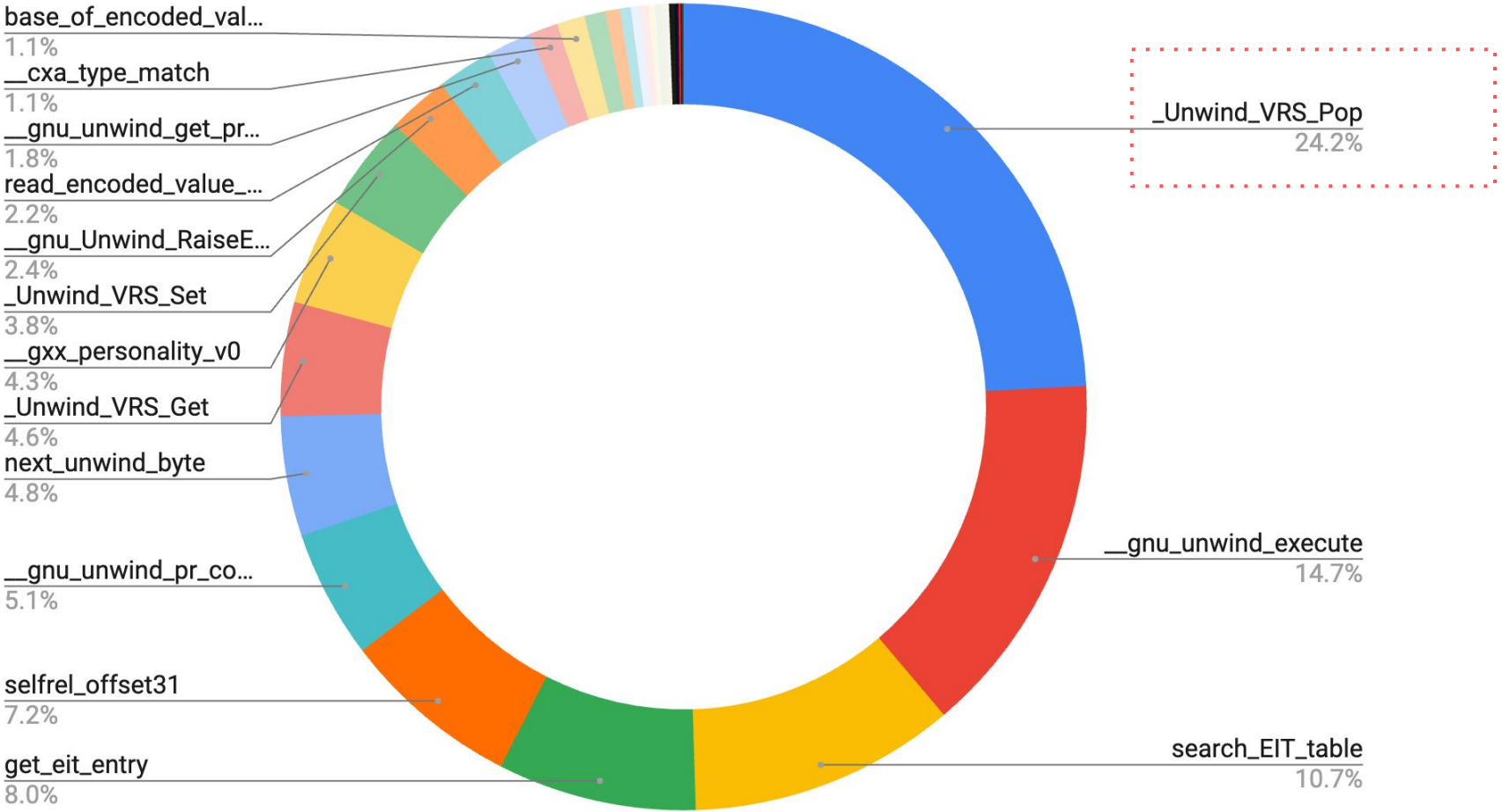
V0


__gxx_personality_...
1.0%
__gnu_unwind_get...
1.4%
__gnu_Unwind_Rai...
1.5%
_Unwind_VRS_Set
3.4%
_Unwind_VRS_Get
3.5%
__gnu_unwind_pr_...
4.2%
next_unwind_byte
4.4%
get_eit_entry
5.9%
__gnu_unwind_exe...
12.6%
selfrel_offset31
13.5%
search_EIT_table
25.6%
_Unwind_VRS_Pop
20.7%

V1


base_of_encoded_val...
1.1%
__cxa_type_match
1.1%
__gnu_unwind_get_pr...
1.8%
read_encoded_value_...
2.2%
__gnu_Unwind_RaiseE...
2.4%
_Unwind_VRS_Set
3.8%
__gxx_personality_v0
4.3%
_Unwind_VRS_Get
4.6%
next_unwind_byte
4.8%
__gnu_unwind_pr_co...
5.1%
selfrel_offset31
7.2%
get_eit_entry
8.0%
_Unwind_VRS_Pop
24.2%
__gnu_unwind_execute
14.7%
search_EIT_table
10.7%
25.6%
10.7%
Explanation:
<algorithm> is well designed for performance.

V1


base_of_encoded_val...
1.1%
__cxa_type_match
1.1%
__gnu_unwind_get_pr...
1.8%
read_encoded_value_...
2.2%
__gnu_Unwind_RaiseE...
2.4%
_Unwind_VRS_Set
3.8%
__gxx_personality_v0
4.3%
_Unwind_VRS_Get
4.6%
next_unwind_byte
4.8%
__gnu_unwind_pr_co...
5.1%
selfrel_offset31
7.2%
get_eit_entry
8.0%
search_EIT_table
10.7%
__gnu_unwind_execute
14.7%
_Unwind_VRS_Pop
24.2%

```
/* ABI defined function to pop registers off the stack.  */
_Unwind_VRS_Result _Unwind_VRS_Pop (_Unwind_Context *context,
           _Unwind_VRS_RegClass regclass,
           _uw discriminator,
           _Unwind_VRS_DataRepresentation representation)
{
 phase1_vrs *vrs = (phase1_vrs *) context;
 switch (regclass) {
   case _UVRSC_CORE: {
     _uw *ptr;
     _uw mask;
     int i;
     if (representation != _UVRSD_UINT32)
       return _UVRSR_FAILED;

     mask = discriminator & 0xffff;
     ptr = (_uw *) vrs->core.r[R_SP];
     /* Pop the requested registers. */
     for (i = 0; i < 16; i++)
     {
       if (mask & (1 << i))
         vrs->core.r[i] = *(ptr++);
     }
     /* The rest ... */
```

```
/* ABI defined function to pop registers off the stack.  */
_Unwind_VRS_Result _Unwind_VRS_Pop (_Unwind_Context *context,
           _Unwind_VRS_RegClass regclass,
           _uw discriminator,
           _Unwind_VRS_DataRepresentation representation)
{
 phase1_vrs *vrs = (phase1_vrs *) context;
 switch (regclass) {
   case _UVRSC_CORE: {
     _uw *ptr;
     _uw mask;
     int i;
     if (representation != _UVRSD_UINT32)
       return _UVRSR_FAILED;

     mask = discriminator & 0xffff;
     ptr = (_uw *) vrs->core.r[R_SP];
     /* Pop the requested registers. */
     for (i = 0; i < 16; i++)
     {
       if (mask & (1 << i))
         vrs->core.r[i] = *(ptr++);
     }
     /* The rest ... */
```

```
mask = discriminator & 0xffff;
ptr = (_uw *) vrs->core.r[R_SP];
/* Pop the requested registers. */
for (i = 0; i < 16; i++)
{
  if (mask & (1 << i))
    vrs->core.r[i] = *(ptr++);
}
```

Cppcon
The C++ Conference

Cppcon.org

Video Sponsorship Provided By

think-cell

push_back with an $\mathcal{O}(n)$ budget

Budget: $ $

```
v.push_back(2);
```

Cost: $

0	0	1	2

13/32

Amortized O(1) Complexity

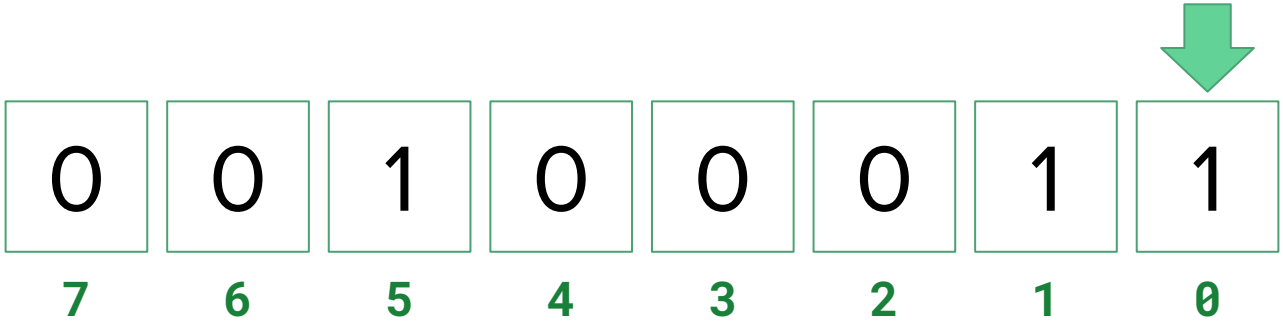
Andreas Weis

1:52 / 51:36

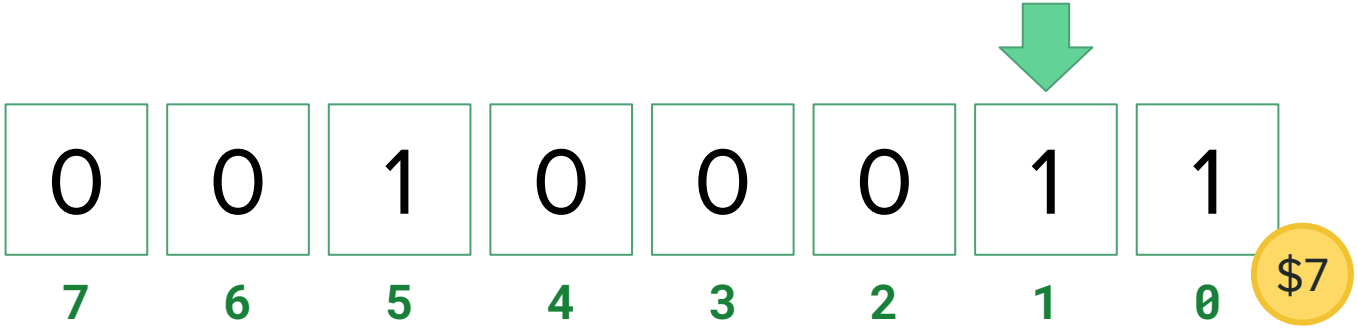
```
/* ABI defined function to pop registers off the stack.  */
_Unwind_VRS_Result _Unwind_VRS_Pop (_Unwind_Context *context,
           _Unwind_VRS_RegClass regclass,
           _uw discriminator,
           _Unwind_VRS_DataRepresentation representation)
{
 phase1_vrs *vrs = (phase1_vrs *) context;
 switch (regclass) {
   case _UVRSC_CORE: {
     _uw *ptr;
     _uw mask;
     int i;
     if (representation != _UVRSD_UINT32)
       return _UVRSR_FAILED;

     mask = discriminator & 0xffff;
     ptr = (_uw *) vrs->core.r[R_SP];
     /* Pop the requested registers. */
     for (i = 0; i < 16; i++)
     {
       if (mask & (1 << i))
         vrs->core.r[i] = *(ptr++);
     }
     /* The rest ... */
```

$ = 1x operation

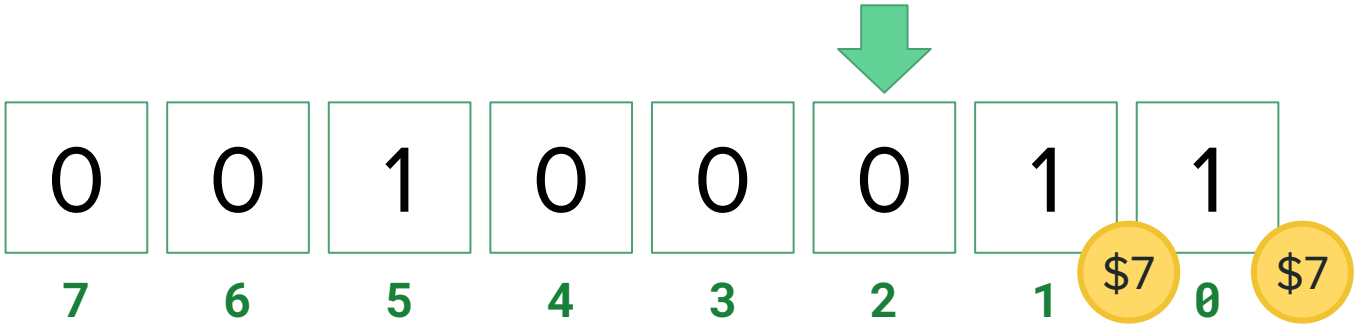
```
mask = discriminator & 0xffff;
ptr = (_uw *) vrs->core.r[R_SP];
/* Pop the requested registers. */
for (i = 0; i < 16; i++)
{
  if (mask & (1 << i))
    vrs->core.r[i] = *(ptr++);
}
```



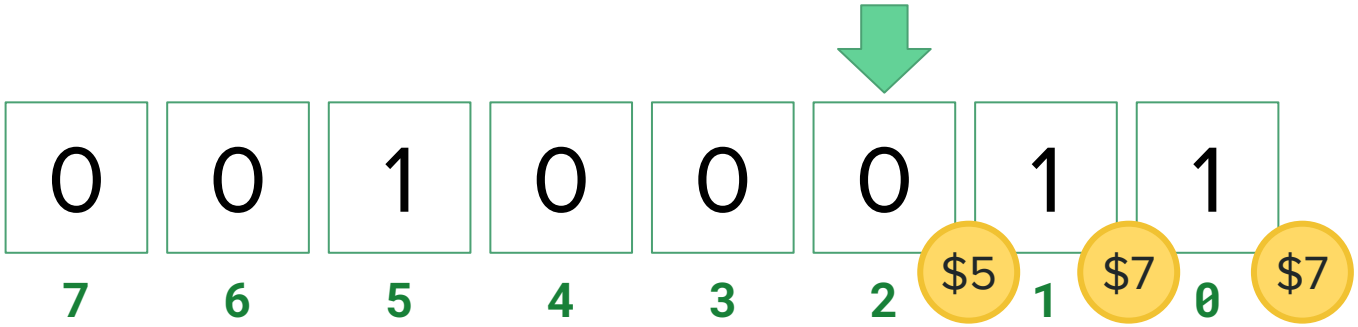

```
mask = discriminator & 0xffff;

ptr = (_uw *) vrs->core.r[R_SP];

/* Pop the requested registers. */

for (i = 0; i < 16; i++)

{

  if (mask & (1 << i))

    vrs->core.r[i] = *(ptr++);

}
```



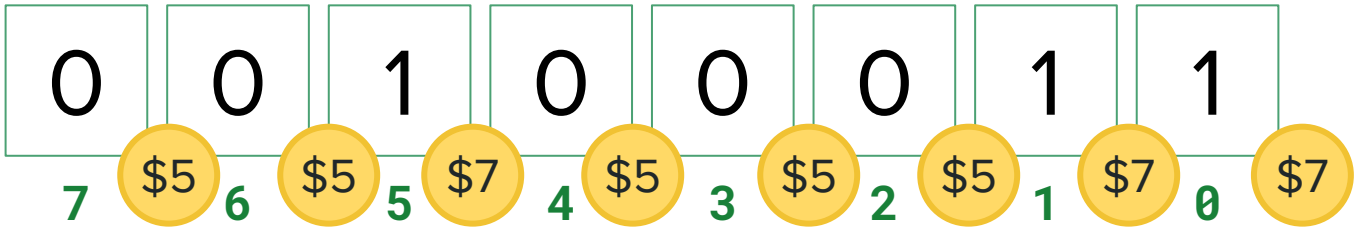

```
mask = discriminator & 0xffff;

ptr = (_uw *) vrs->core.r[R_SP];

/* Pop the requested registers. */

for (i = 0; i < 16; i++)

{

  if (mask & (1 << i))

    vrs->core.r[i] = *(ptr++);

}
```




```
mask = discriminator & 0xffff;

ptr = (_uw *) vrs->core.r[R_SP];

/* Pop the requested registers. */

for (i = 0; i < 16; i++)

{

  if (mask & (1 << i))

    vrs->core.r[i] = *(ptr++);

}
```



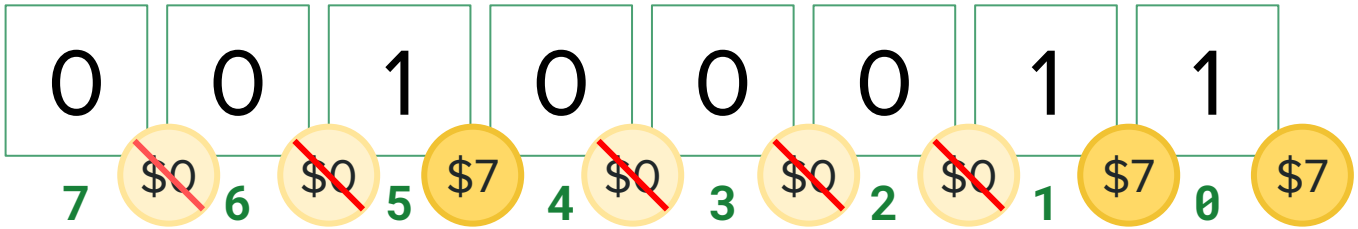

```
mask = discriminator & 0xffff;

ptr = (_uw *) vrs->core.r[R_SP];

/* Pop the requested registers. */

for (i = 0; i < 16; i++)

{

  if (mask & (1 << i))

    vrs->core.r[i] = *(ptr++);

}
```




```
mask = discriminator & 0xffff;

ptr = (_uw *) vrs->core.r[R_SP];

/* Pop the requested registers. */

for (i = 0; i < 16; i++)

{

  if (mask & (1 << i))

    vrs->core.r[i] = *(ptr++);

}
```

TOTAL

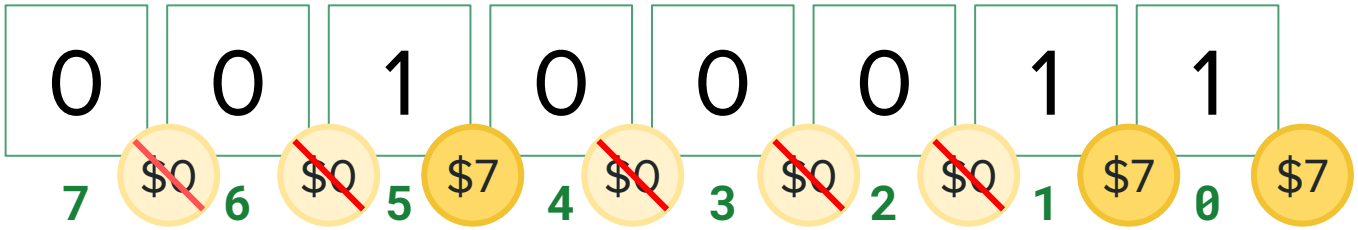
$46




```
mask = discriminator & 0xffff;

ptr = (_uw *) vrs->core.r[R_SP];

/* Pop the requested registers. */

for (i = 0; i < 16; i++)

{

  if (mask & (1 << i))

    vrs->core.r[i] = *(ptr++);

}
```

TOTAL

$??

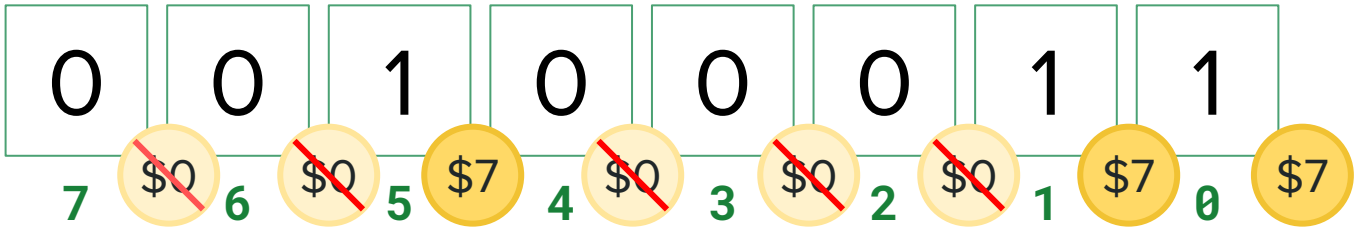



```
mask = discriminator & 0xffff;

ptr = (_uw *) vrs->core.r[R_SP];

/* Pop the requested registers. */

for (i = 0; i < 16; i++)

{

  if (mask & (1 << i))

    vrs->core.r[i] = *(ptr++);

}
```

TOTAL

$21

-54% off



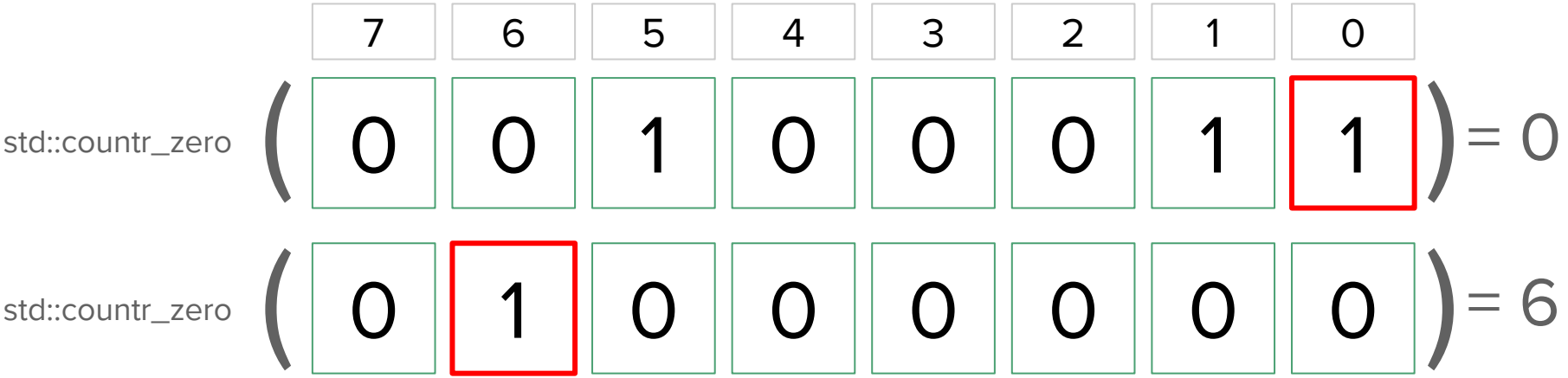

```
mask = discriminator & 0xffff;

ptr = (_uw *) vrs->core.r[R_SP];

/* Pop the requested registers. */

for (i = 0; i < 16; i++)

{

  if (mask & (1 << i))

    vrs->core.r[i] = *(ptr++);

}
```

BARGAIN DEAL!

TOTAL

$21

-54% off



std::countr_zero

Defined in header <bit>

```
template< class T >
constexpr int countr_zero( T x ) noexcept;
```
(since C++20)

Returns the number of consecutive `0` bits in the value of `x`, starting from the least significant bit (“right”).

ARM Cortex M4

```
rbit    r2, r3
clz     r2, r2
```

✅

X86-64

```
bsf edx, ecx
```

✅

RISC-V

```
call    __ctzsi2
```

🟡

https://godbolt.org/z/ezKEGGhGx

std::countr_zero

Defined in header <bit>

```
template< class T >
constexpr int countr_zero( T x ) noexcept;
```

(since C++20)

Returns the number of consecutive `0` bits in the value of `x`, starting from the least significant bit (“right”).

```
while (discriminator) {
 auto const reg_index = std::countr_zero(discriminator);
 vrs->core.r[reg_index] = *(ptr++);
 discriminator &= ~(1 << reg_index);
}
```

cppreference.com

Create account

Page | Discussion | Standard revision: Diff | View | Edit

C++ | Utilities library | Bit manipulation

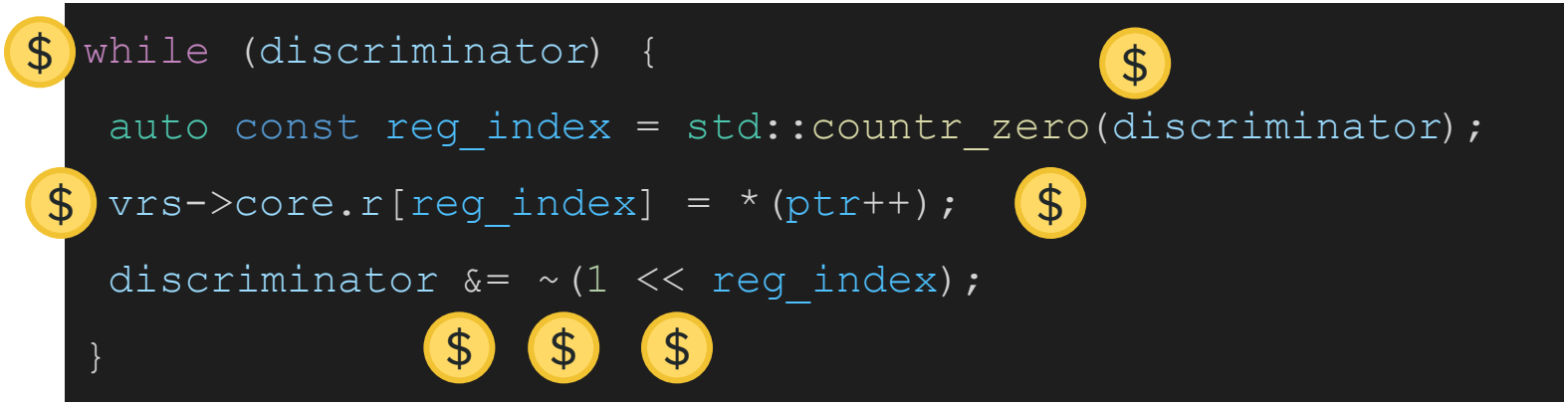
std::countr_zero

Defined in header <bit>

```
template< class T >
constexpr int countr_zero( T x ) noexcept;
```

(since C++20)

Returns the number of consecutive `0` bits in the value of `x`, starting from the least significant bit (“right”).



cppreference.com

Create account

Page | Discussion | Standard revision: Diff | View | Edit

C++ | Utilities library | Bit manipulation

std::countr_zero

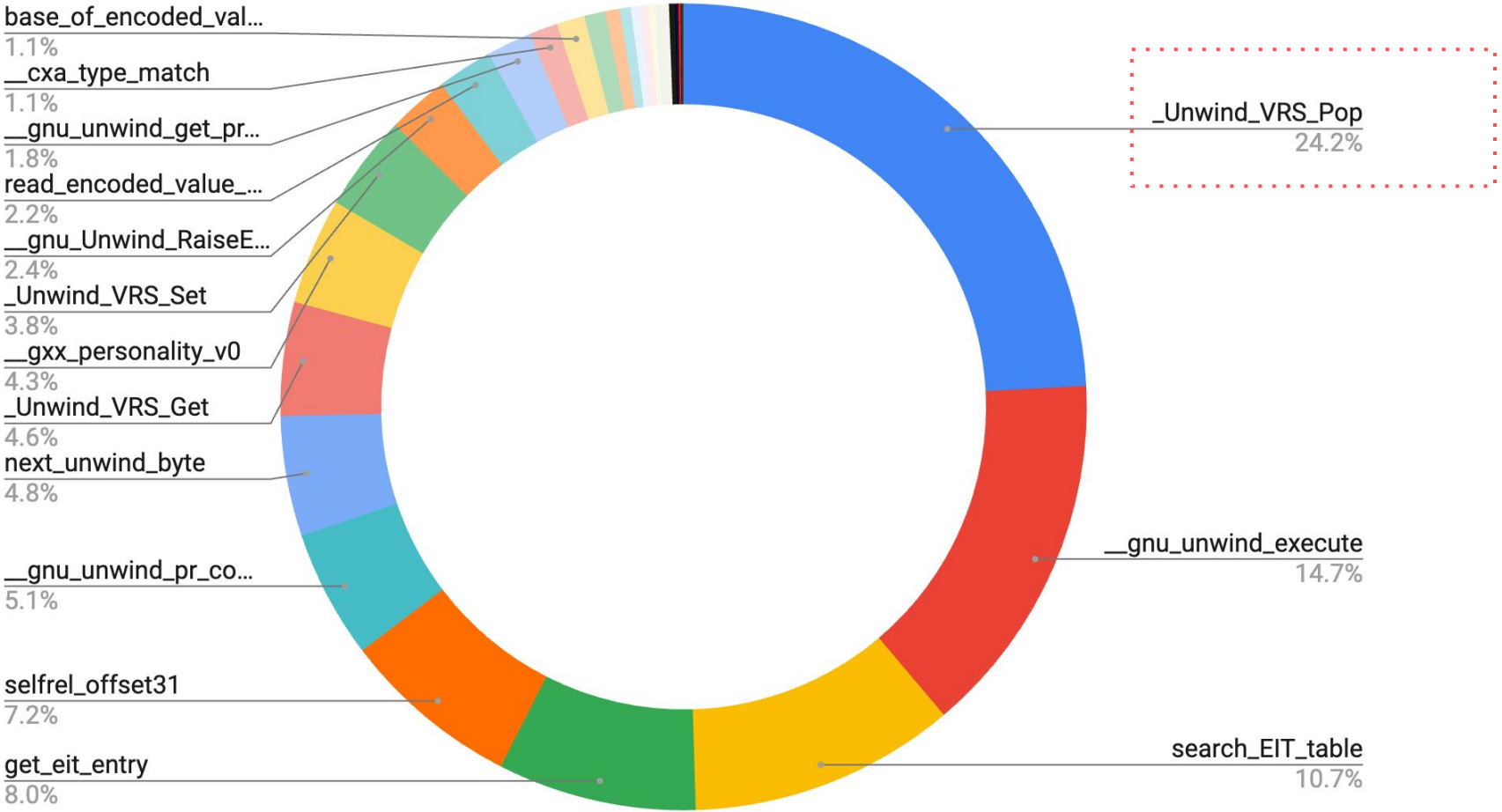
Defined in header <bit>

```
template< class T >
constexpr int countr_zero( T x ) noexcept;
```

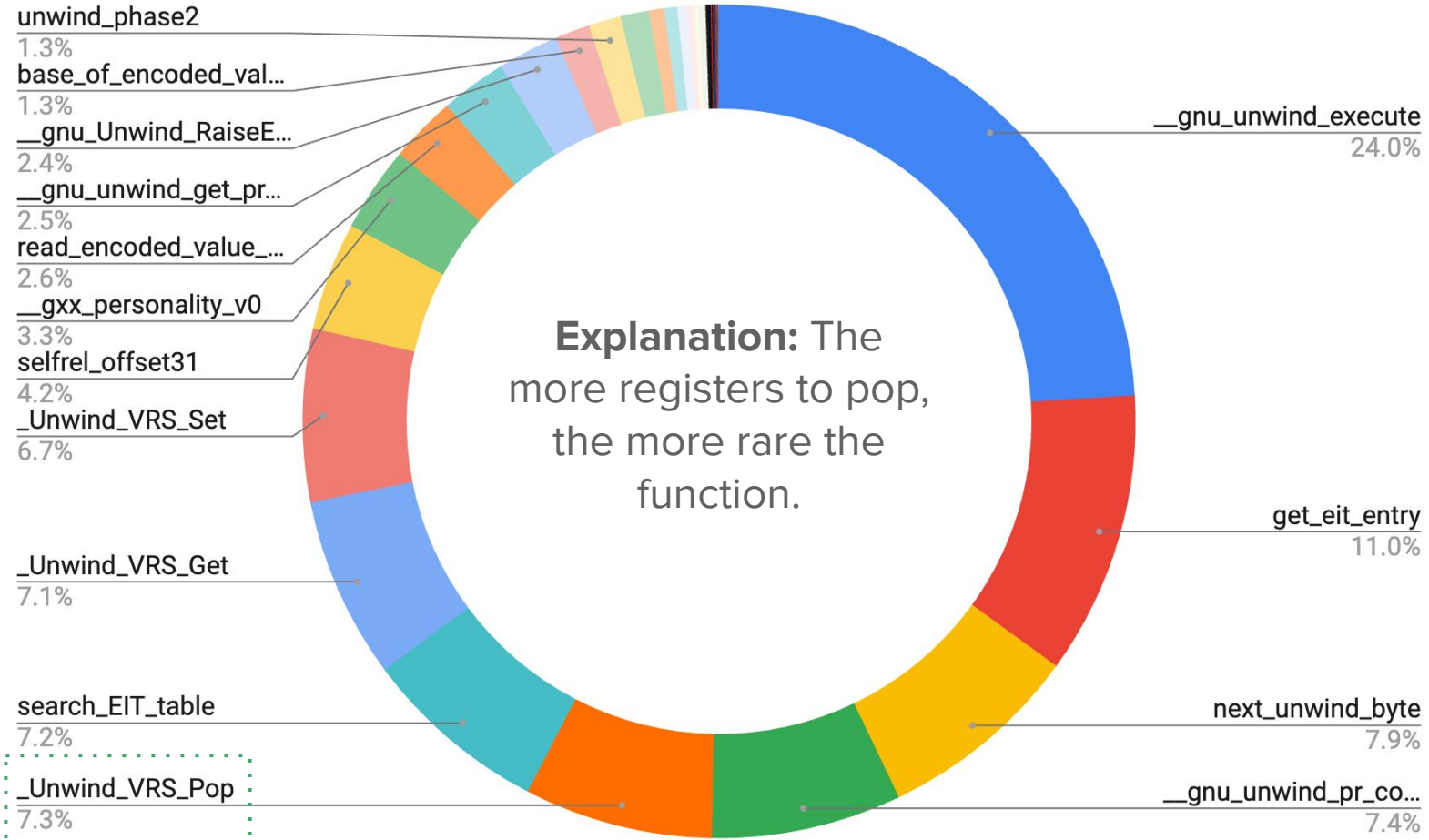
(since C++20)

Returns the number of consecutive `0` bits in the value of `x`, starting from the least significant bit (“right”).

V1

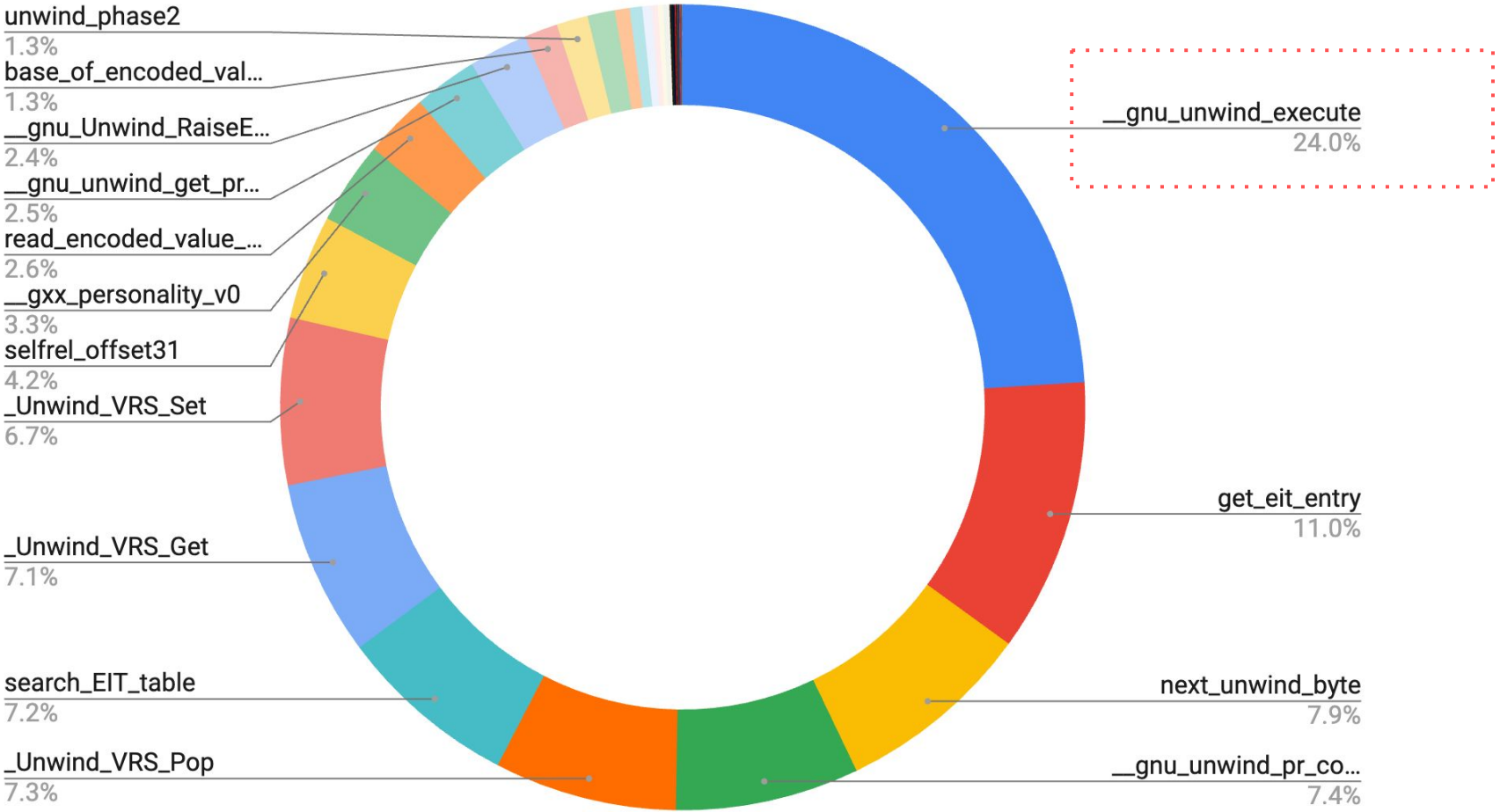

base_of_encoded_val...
1.1%
__cxa_type_match
1.1%
__gnu_unwind_get_pr...
1.8%
read_encoded_value_...
2.2%
__gnu_Unwind_RaiseE...
2.4%
_Unwind_VRS_Set
3.8%
__gxx_personality_v0
4.3%
_Unwind_VRS_Get
4.6%
next_unwind_byte
4.8%
__gnu_unwind_pr_co...
5.1%
selfrel_offset31
7.2%
get_eit_entry
8.0%
search_EIT_table
10.7%
__gnu_unwind_execute
14.7%
_Unwind_VRS_Pop
24.2%

V2

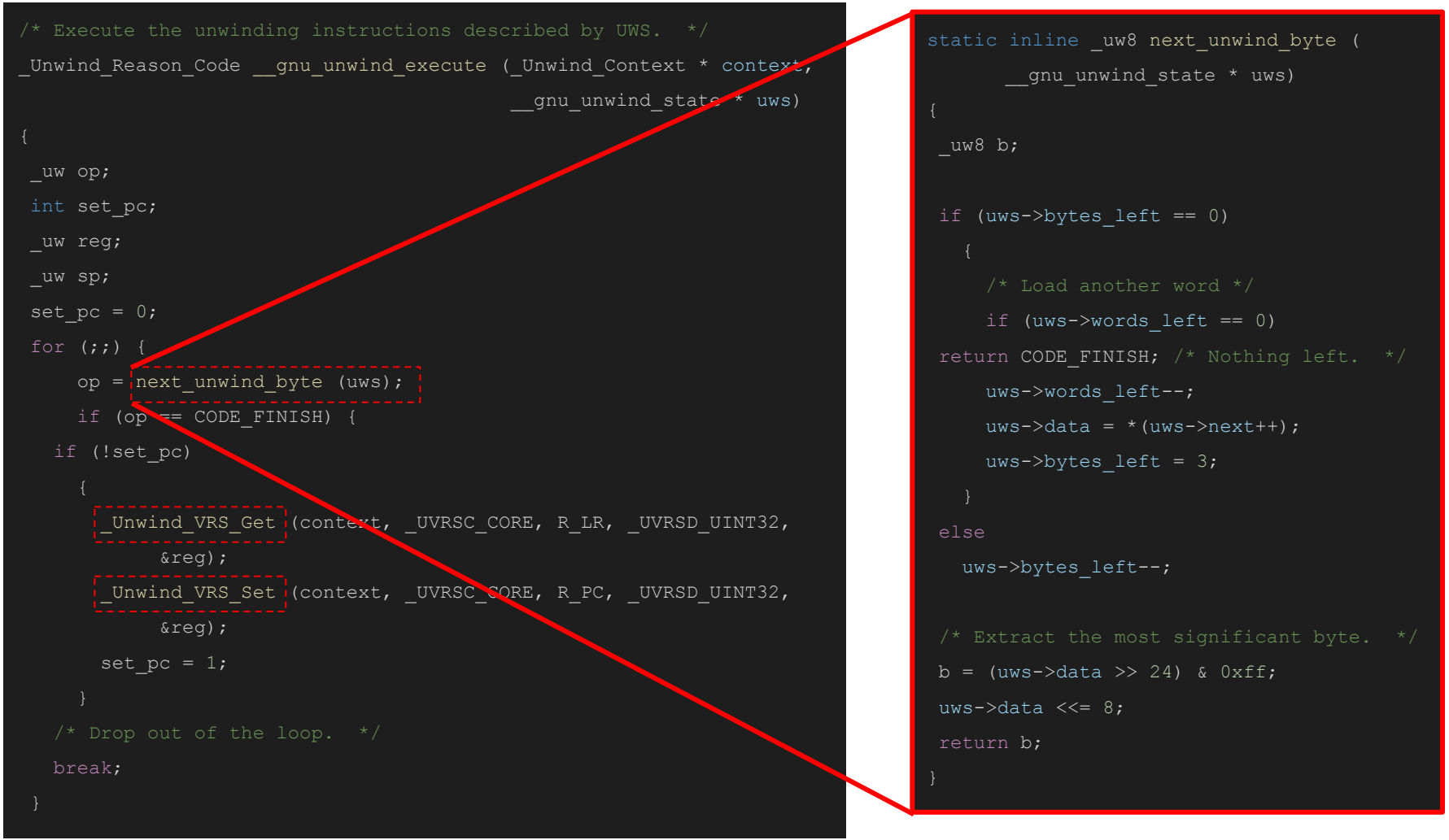

unwind_phase2
1.3%
base_of_encoded_val...
1.3%
__gnu_Unwind_RaiseE...
2.4%
__gnu_unwind_get_pr...
2.5%
read_encoded_value_...
2.6%
__gxx_personality_v0
3.3%
selfrel_offset31
4.2%
_Unwind_VRS_Set
6.7%
_Unwind_VRS_Get
7.1%
search_EIT_table
7.2%
_Unwind_VRS_Pop
7.3%
__gnu_unwind_execute
24.0%
get_eit_entry
11.0%
next_unwind_byte
7.9%
__gnu_unwind_pr_co...
7.4%
Explanation: The more registers to pop, the more rare the function.

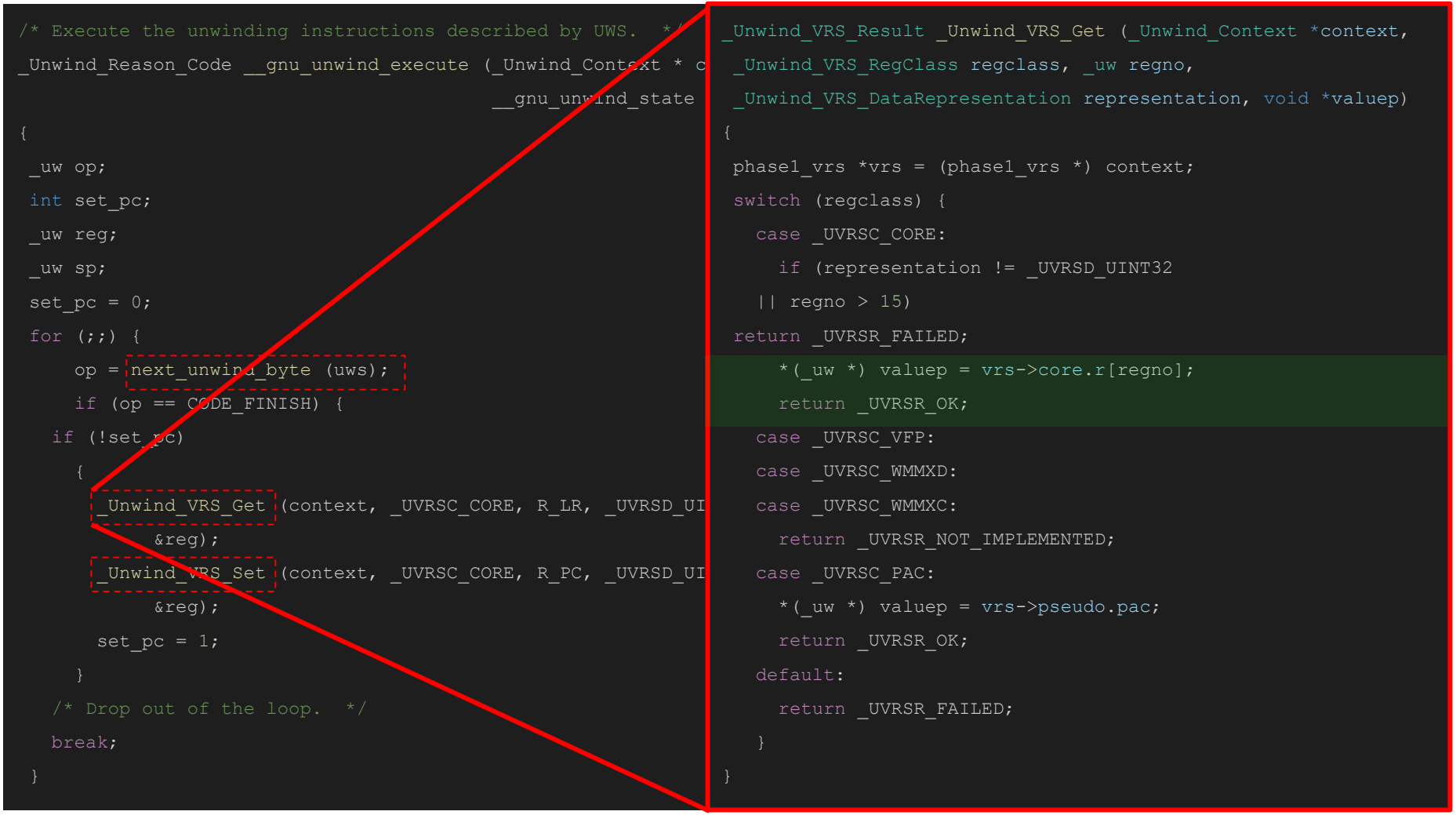
24%
➡️
7.3%

V2


unwind_phase2
1.3%
base_of_encoded_val...
1.3%
__gnu_Unwind_RaiseE...
2.4%
__gnu_unwind_get_pr...
2.5%
read_encoded_value_...
2.6%
__gxx_personality_v0
3.3%
selfrel_offset31
4.2%
_Unwind_VRS_Set
6.7%
_Unwind_VRS_Get
7.1%
search_EIT_table
7.2%
_Unwind_VRS_Pop
7.3%
__gnu_unwind_execute
24.0%
get_eit_entry
11.0%
next_unwind_byte
7.9%
__gnu_unwind_pr_co...
7.4%

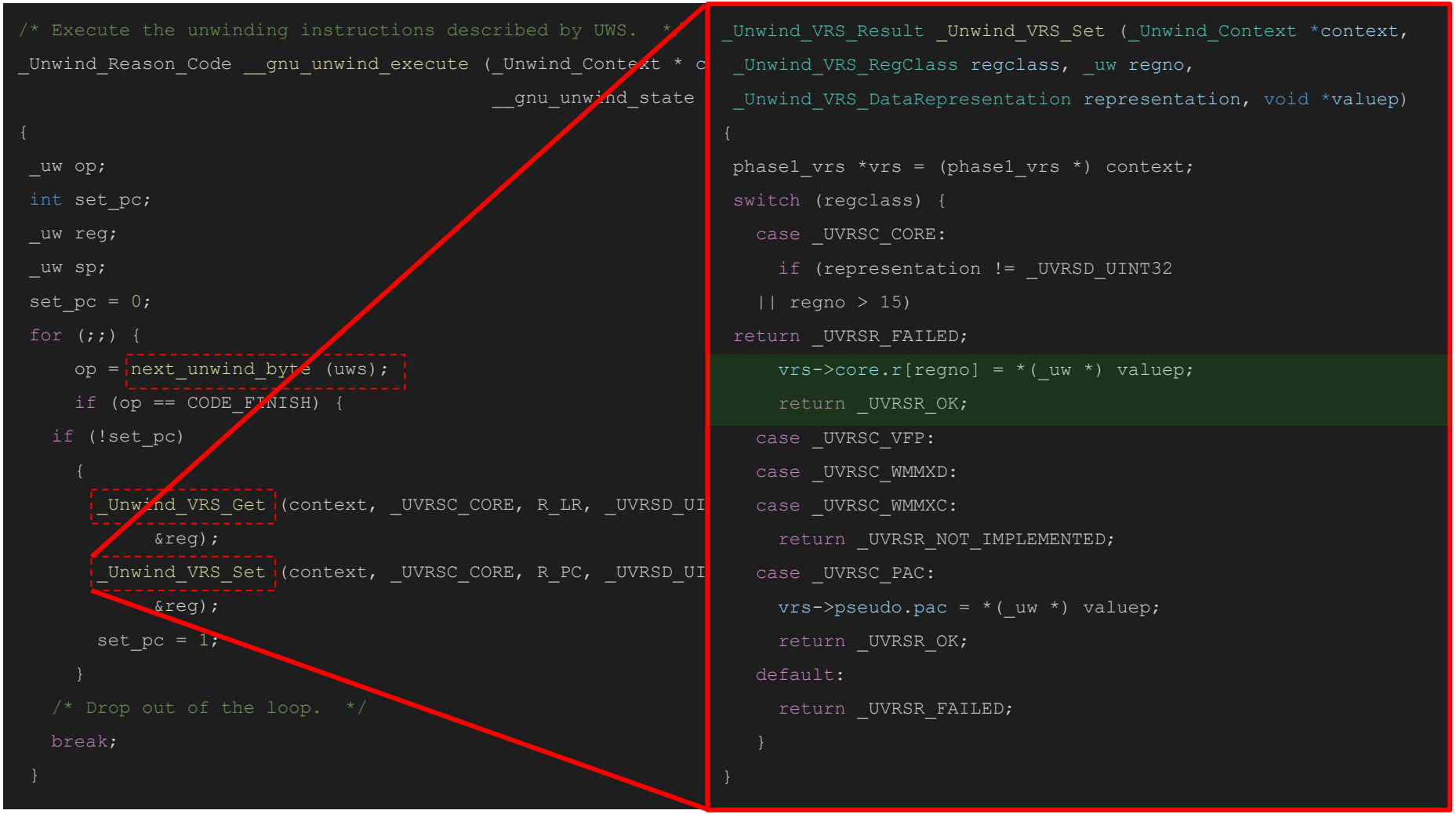
```
/* Execute the unwinding instructions described by UWS.  */
_Unwind_Reason_Code __gnu_unwind_execute (_Unwind_Context * context,
                                          __gnu_unwind_state * uws)
{
 _uw op;
 int set_pc;
 _uw reg;
 _uw sp;
 set_pc = 0;
 for (;;) {
     op = next_unwind_byte (uws);
     if (op == CODE_FINISH) {
   if (!set_pc)
     {
       _Unwind_VRS_Get (context, _UVRSC_CORE, R_LR, _UVRSD_UINT32,
            &reg);
       _Unwind_VRS_Set (context, _UVRSC_CORE, R_PC, _UVRSD_UINT32,
            &reg);
       set_pc = 1;
     }
   /* Drop out of the loop.  */
   break;
 }
```


/* Execute the unwinding instructions described by UWS. */
_Unwind_Reason_Code __gnu_unwind_execute (_Unwind_Context * context,
__gnu_unwind_state * uws)
{
_uw op;
int set_pc;
_uw reg;
_uw sp;
set_pc = 0;
for (;;) {
op = next_unwind_byte (uws);
if (op == CODE_FINISH) {
if (!set_pc)
{
_Unwind_VRS_Get (context, _UVRSC_CORE, R_LR, _UVRSD_UINT32,
&reg);
_Unwind_VRS_Set (context, _UVRSC_CORE, R_PC, _UVRSD_UINT32,
&reg);
set_pc = 1;
}
/* Drop out of the loop. */
break;
}
static inline _uw8 next_unwind_byte (
__gnu_unwind_state * uws)
{
_uw8 b;
if (uws->bytes_left == 0)
{
/* Load another word */
if (uws->words_left == 0)
return CODE_FINISH; /* Nothing left. */
uws->words_left--;
uws->data = *(uws->next++);
uws->bytes_left = 3;
}
else
uws->bytes_left--;
/* Extract the most significant byte. */
b = (uws->data >> 24) & 0xff;
uws->data <<= 8;
return b;
}



```
/* Execute the unwinding instructions described by UWS.  */
_Unwind_Reason_Code __gnu_unwind_execute (_Unwind_Context * c
                                          __gnu_unwind_state
{
 _uw op;
 int set_pc;
 _uw reg;
 _uw sp;
 set_pc = 0;
 for (;;) {
     op = next_unwind_byte (uws);
     if (op == CODE_FINISH) {
   if (!set_pc)
     {
       _Unwind_VRS_Get (context, _UVRSC_CORE, R_LR, _UVRSD_UI
             &reg);
       _Unwind_VRS_Set (context, _UVRSC_CORE, R_PC, _UVRSD_UI
             &reg);
       set_pc = 1;
     }
   /* Drop out of the loop.  */
   break;
 }
```

```
_Unwind_VRS_Result _Unwind_VRS_Get (_Unwind_Context *context,
 _Unwind_VRS_RegClass regclass, _uw regno,
 _Unwind_VRS_DataRepresentation representation, void *valuep)
{
 phase1_vrs *vrs = (phase1_vrs *) context;
 switch (regclass) {
   case _UVRSC_CORE:
     if (representation != _UVRSD_UINT32
   || regno > 15)
 return _UVRSR_FAILED;
     *(_uw *) valuep = vrs->core.r[regno];
     return _UVRSR_OK;
   case _UVRSC_VFP:
   case _UVRSC_WMMXD:
   case _UVRSC_WMMXC:
     return _UVRSR_NOT_IMPLEMENTED;
   case _UVRSC_PAC:
     *(_uw *) valuep = vrs->pseudo.pac;
     return _UVRSR_OK;
   default:
     return _UVRSR_FAILED;
   }
}
```



```
/* Execute the unwinding instructions described by UWS.  */
_Unwind_Reason_Code __gnu_unwind_execute (_Unwind_Context * c
                                          __gnu_unwind_state
{
 _uw op;
 int set_pc;
 _uw reg;
 _uw sp;
 set_pc = 0;
 for (;;) {
     op = next_unwind_byte (uws);
     if (op == CODE_FINISH) {
   if (!set_pc)
     {
       _Unwind_VRS_Get (context, _UVRSC_CORE, R_LR, _UVRSD_UI
                        &reg);
       _Unwind_VRS_Set (context, _UVRSC_CORE, R_PC, _UVRSD_UI
                        &reg);
       set_pc = 1;
     }
   /* Drop out of the loop.  */
   break;
 }
```

```
_Unwind_VRS_Result _Unwind_VRS_Set (_Unwind_Context *context,
 _Unwind_VRS_RegClass regclass, _uw regno,
 _Unwind_VRS_DataRepresentation representation, void *valuep)
{
 phase1_vrs *vrs = (phase1_vrs *) context;
 switch (regclass) {
   case _UVRSC_CORE:
     if (representation != _UVRSD_UINT32
   || regno > 15)
 return _UVRSR_FAILED;
     vrs->core.r[regno] = *(_uw *) valuep;
     return _UVRSR_OK;
   case _UVRSC_VFP:
   case _UVRSC_WMMXD:
   case _UVRSC_WMMXC:
     return _UVRSR_NOT_IMPLEMENTED;
   case _UVRSC_PAC:
     vrs->pseudo.pac = *(_uw *) valuep;
     return _UVRSR_OK;
   default:
     return _UVRSR_FAILED;
   }
}
```

```
_Unwind_VRS_Result _Unwind_VRS_Set (_Unwind_Context *context,
 _Unwind_VRS_RegClass regclass, _uw regno,
 _Unwind_VRS_DataRepresentation representation, void *valuep)
{
 phase1_vrs *vrs = (phase1_vrs *) context;
 switch (regclass) {
   case _UVRSC_CORE:
     if (representation != _UVRSD_UINT32
   || regno > 15)
 return _UVRSR_FAILED;
     vrs->core.r[regno] = *(_uw *) valuep;
     return _UVRSR_OK;
   case _UVRSC_VFP:
   case _UVRSC_WMMXD:
   case _UVRSC_WMMXC:
     return _UVRSR_NOT_IMPLEMENTED;
   case _UVRSC_PAC:
     vrs->pseudo.pac = *(_uw *) valuep;
     return _UVRSR_OK;
   default:
     return _UVRSR_FAILED;
   }
}
```

```
[[gnu::always_inline]] _Unwind_VRS_Result
_My_Unwind_VRS_Set(
   _Unwind_Context* context,
   _Unwind_VRS_RegClass regclass,
   _uw regno,
   _Unwind_VRS_DataRepresentation representation,
   void* valuep)
 {
   auto* vrs = reinterpret_cast<phase1_vrs*>(context);
   vrs->core.r[regno] = *(_uw*)valuep;
   return _UVRSR_OK;
 }
```

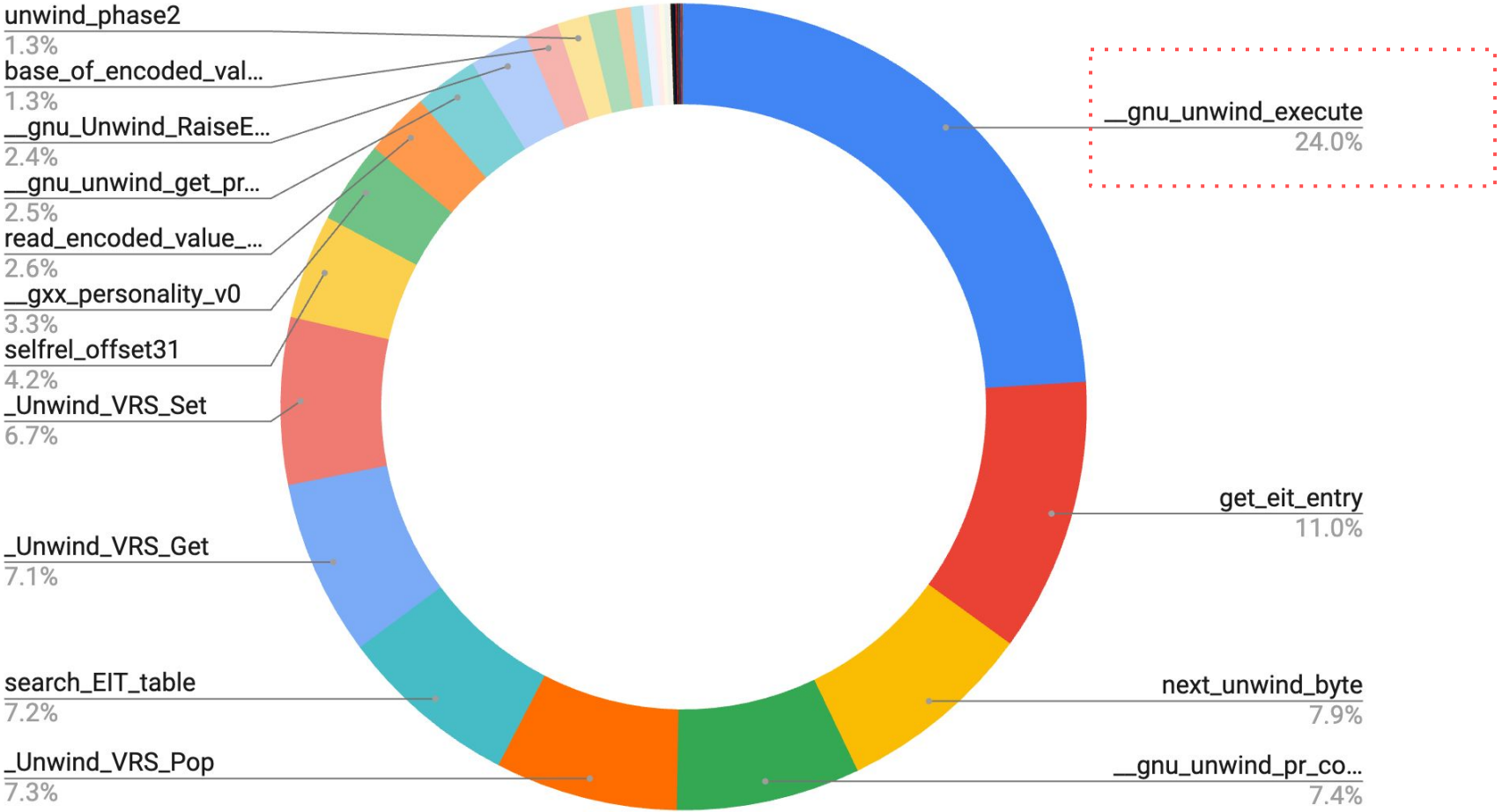
```
_Unwind_VRS_Result _Unwind_VRS_Get (_Unwind_Context *context,
 _Unwind_VRS_RegClass regclass, _uw regno,
 _Unwind_VRS_DataRepresentation representation, void *valuep)
{
 phase1_vrs *vrs = (phase1_vrs *) context;
 switch (regclass) {
   case _UVRSC_CORE:
     if (representation != _UVRSD_UINT32
   || regno > 15)
 return _UVRSR_FAILED;
     *(_uw *) valuep = vrs->core.r[regno];
     return _UVRSR_OK;
   case _UVRSC_VFP:
   case _UVRSC_WMMXD:
   case _UVRSC_WMMXC:
     return _UVRSR_NOT_IMPLEMENTED;
   case _UVRSC_PAC:
     *(_uw *) valuep = vrs->pseudo.pac;
     return _UVRSR_OK;
   default:
     return _UVRSR_FAILED;
   }
}
```

```
[[gnu::always_inline]] _Unwind_VRS_Result
_My_Unwind_VRS_Get(
   _Unwind_Context* context,
   _Unwind_VRS_RegClass regclass,
   _uw regno,
   _Unwind_VRS_DataRepresentation representation,
   void* valuep)
 {
   auto* vrs = reinterpret_cast<phase1_vrs*>(context);
   *(_uw*)valuep = vrs->core.r[regno];
   return _UVRSR_OK;
 }
```

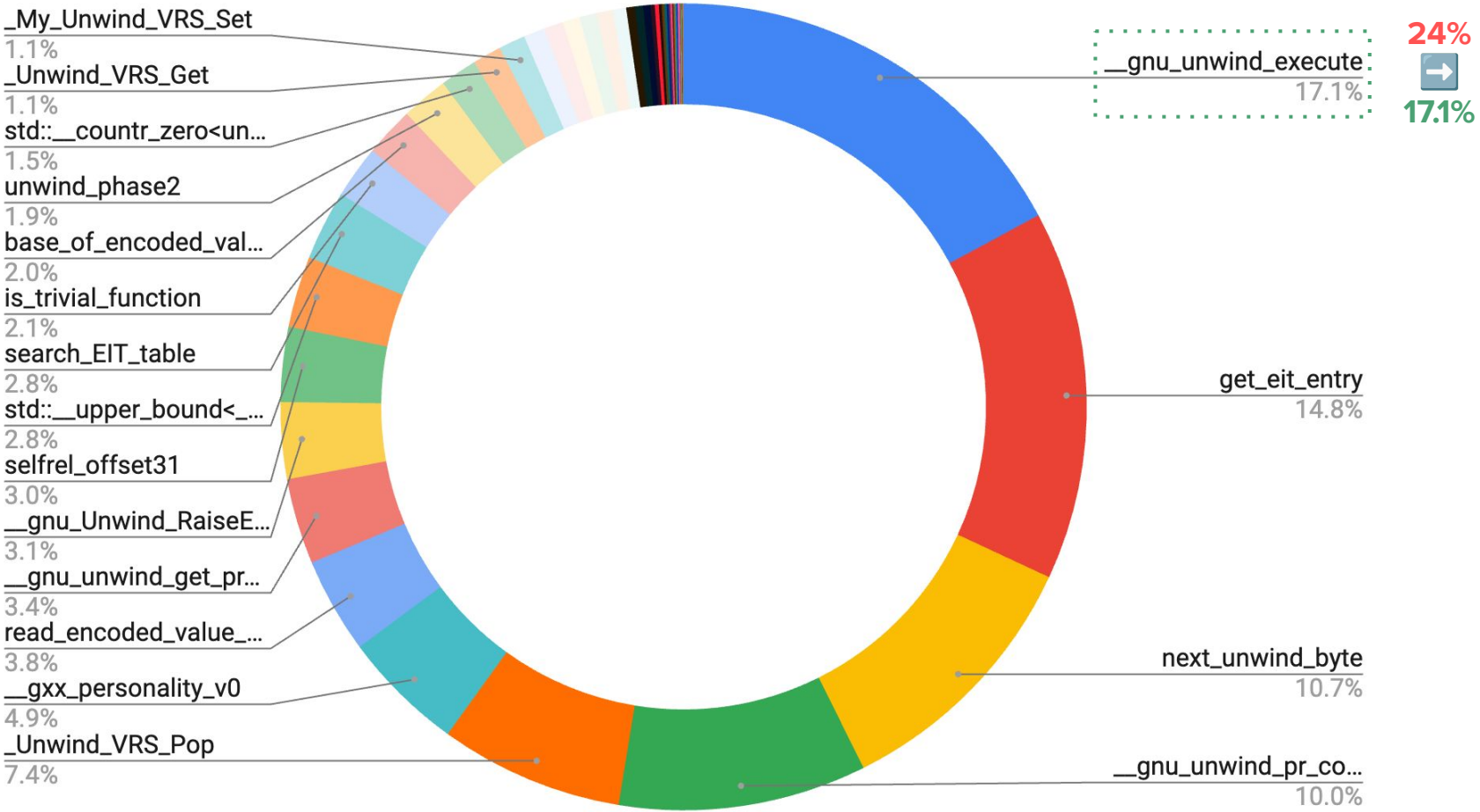
```
_uw8 next_unwind_byte(__gnu_unwind_state* uws)
 {
   _uw8 b;
   if (uws->bytes_left == 0) {
     /* Load another word */
     if (uws->words_left == 0)
       return 0xB0; /* Nothing left.  */
     uws->words_left--;
     uws->data = *(uws->next++);
     uws->bytes_left = 3;
   } else
     uws->bytes_left--;

   /* Extract the most significant byte.  */
   b = (uws->data >> 24) & 0xff;
   uws->data <<= 8;
   return b;
 }
```

```
[[gnu::always_inline]] _uw8
next_unwind_byte(__gnu_unwind_state* uws)
 {
   _uw8 b;
   if (uws->bytes_left == 0) {
     /* Load another word */
     if (uws->words_left == 0)
       return 0xB0; /* Nothing left.  */
     uws->words_left--;
     uws->data = *(uws->next++);
     uws->bytes_left = 3;
   } else
     uws->bytes_left--;

   /* Extract the most significant byte.  */
   b = (uws->data >> 24) & 0xff;
   uws->data <<= 8;
   return b;
 }
```

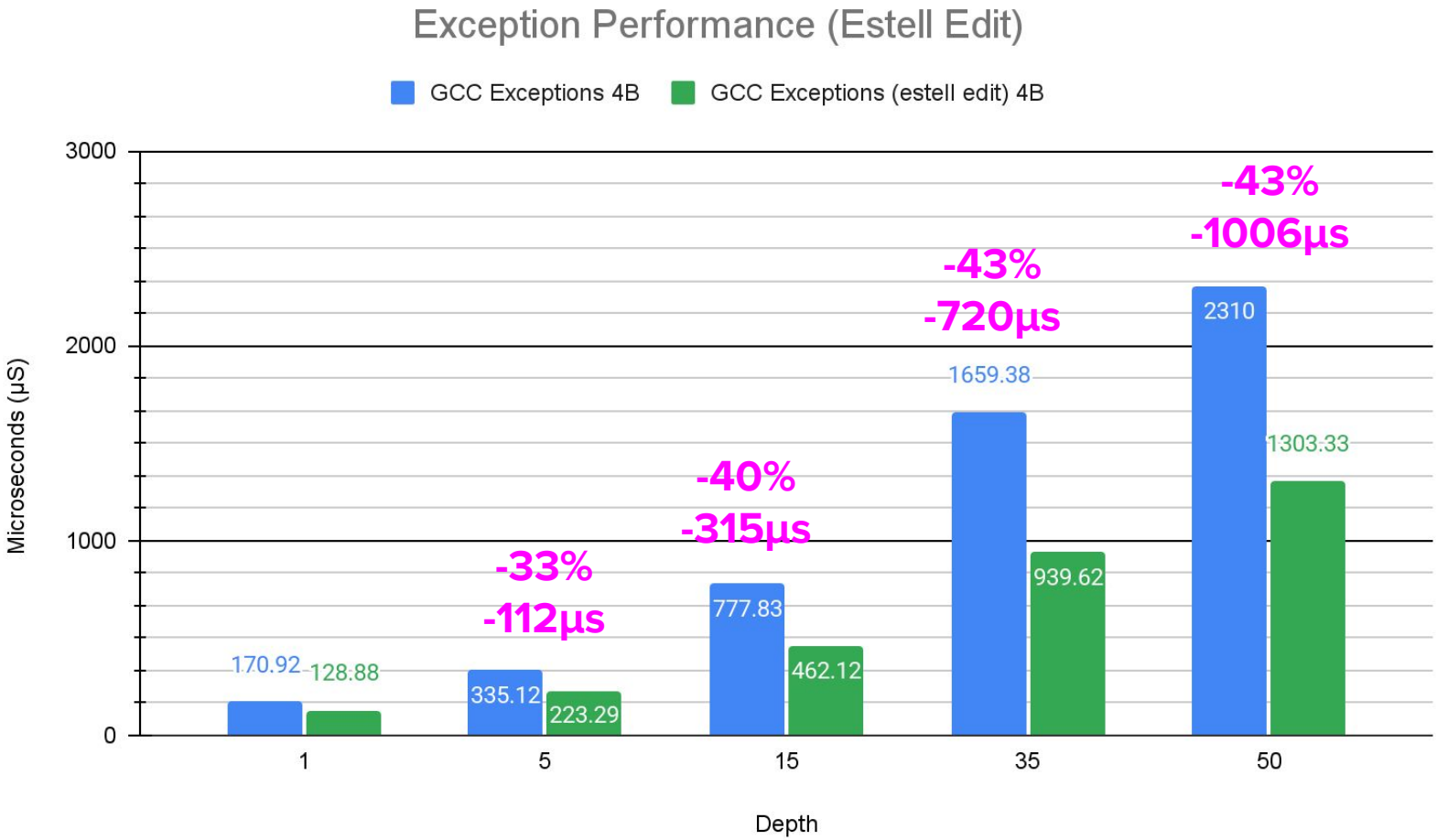
V2


unwind_phase2
1.3%
base_of_encoded_val...
1.3%
__gnu_Unwind_RaiseE...
2.4%
__gnu_unwind_get_pr...
2.5%
read_encoded_value_...
2.6%
__gxx_personality_v0
3.3%
selfrel_offset31
4.2%
_Unwind_VRS_Set
6.7%
_Unwind_VRS_Get
7.1%
search_EIT_table
7.2%
_Unwind_VRS_Pop
7.3%
__gnu_unwind_execute
24.0%
get_eit_entry
11.0%
next_unwind_byte
7.9%
__gnu_unwind_pr_co...
7.4%

V3


_My_Unwind_VRS_Set
1.1%
_Unwind_VRS_Get
1.1%
std::__countr_zero<un...
1.5%
unwind_phase2
1.9%
base_of_encoded_val...
2.0%
is_trivial_function
2.1%
search_EIT_table
2.8%
std::__upper_bound<_...
2.8%
selfrel_offset31
3.0%
__gnu_Unwind_RaiseE...
3.1%
__gnu_unwind_get_pr...
3.4%
read_encoded_value_...
3.8%
__gxx_personality_v0
4.9%
_Unwind_VRS_Pop
7.4%
__gnu_unwind_execute
17.1%
24%
➡️
17.1%
get_eit_entry
14.8%
next_unwind_byte
10.7%
__gnu_unwind_pr_co...
10.0%

Results!! 


Exception Performance (Estell Edit)
GCC Exceptions 4B
GCC Exceptions (estell edit) 4B
Microseconds (µS)
3000
2000
1000
0
1
5
15
35
50
Depth
170.92
128.88
335.12
223.29
777.83
462.12
1659.38
939.62
2310
1303.33
-33%
-112µs
-40%
-315µs
-43%
-720µs
-43%
-1006µs

🎉 Telling people about it! 🗣️

October 2023

Jason Turner

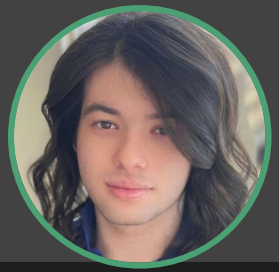
So after that change the cycle count dropped down to ~270 cycles!

That's fascinating... I wonder if you could reduce the space you need to check by doing some sort of linear interpolation.

Ah, kinda like assuming "if the program counter is the middle of the binary then maybe the function's entry is also near the middle?

Kinda of. I just think there may be some structure here you can take advantage of. But maybe not.

Nearpoint Search Algorithm



Madeline Schneider

... well exceptions could never be faster than result types.

Oh yes, I don't expect to beat result types... But now that you say that I'm going to try!

How is that even going to be possible?

* Bad data from 2023 due to misconfigured flash accelerator.

"Consider this madeline..."

Depth	**Cycles**
	std::expected<T,E>
6	556 cycles*
1	92 cycles*

Did some napkin math and got ~65 cycles

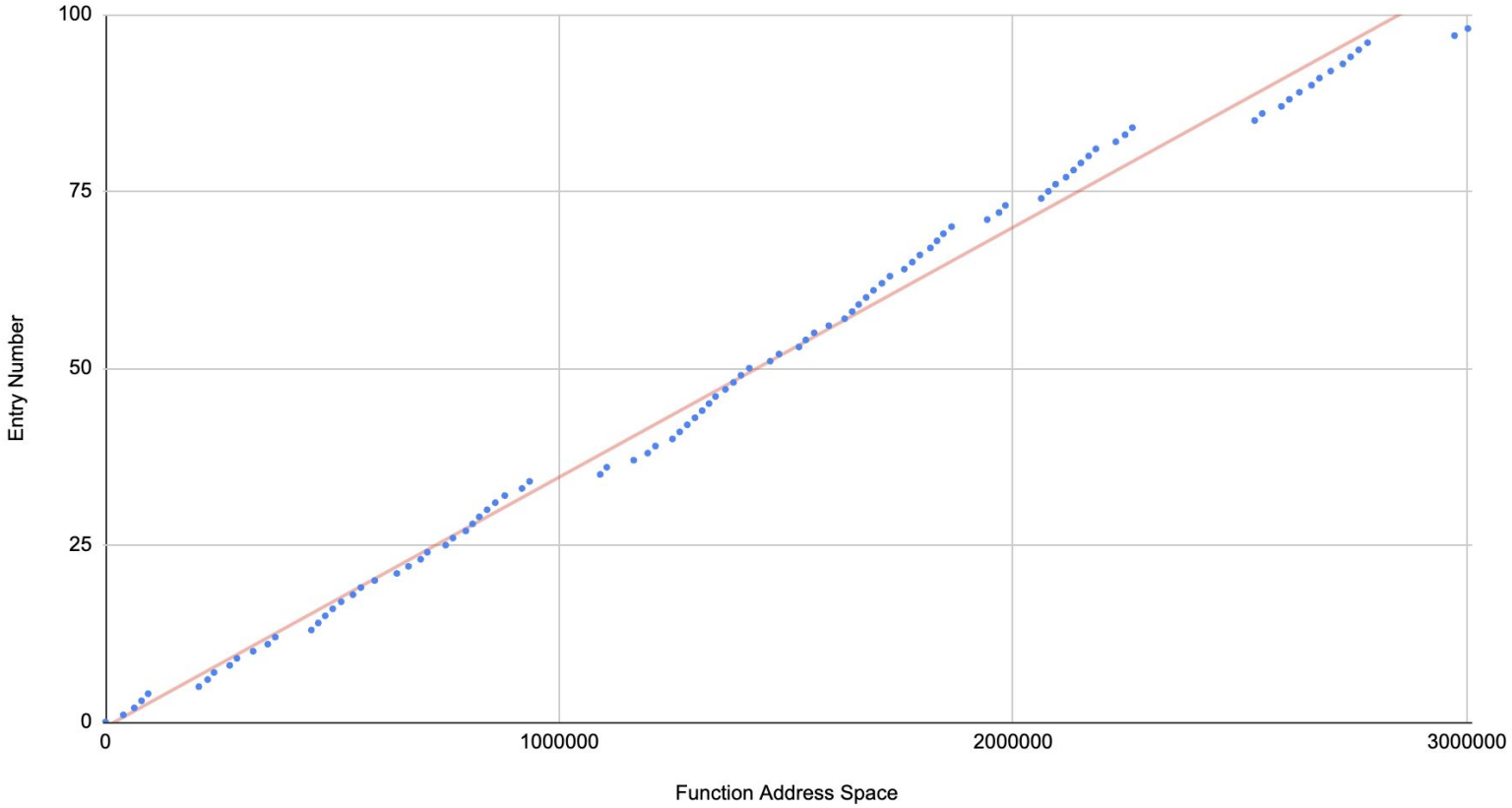
Search time: 270 cycles ➡️ 27 cycles

Nearpoint Search Algorithm Requirements

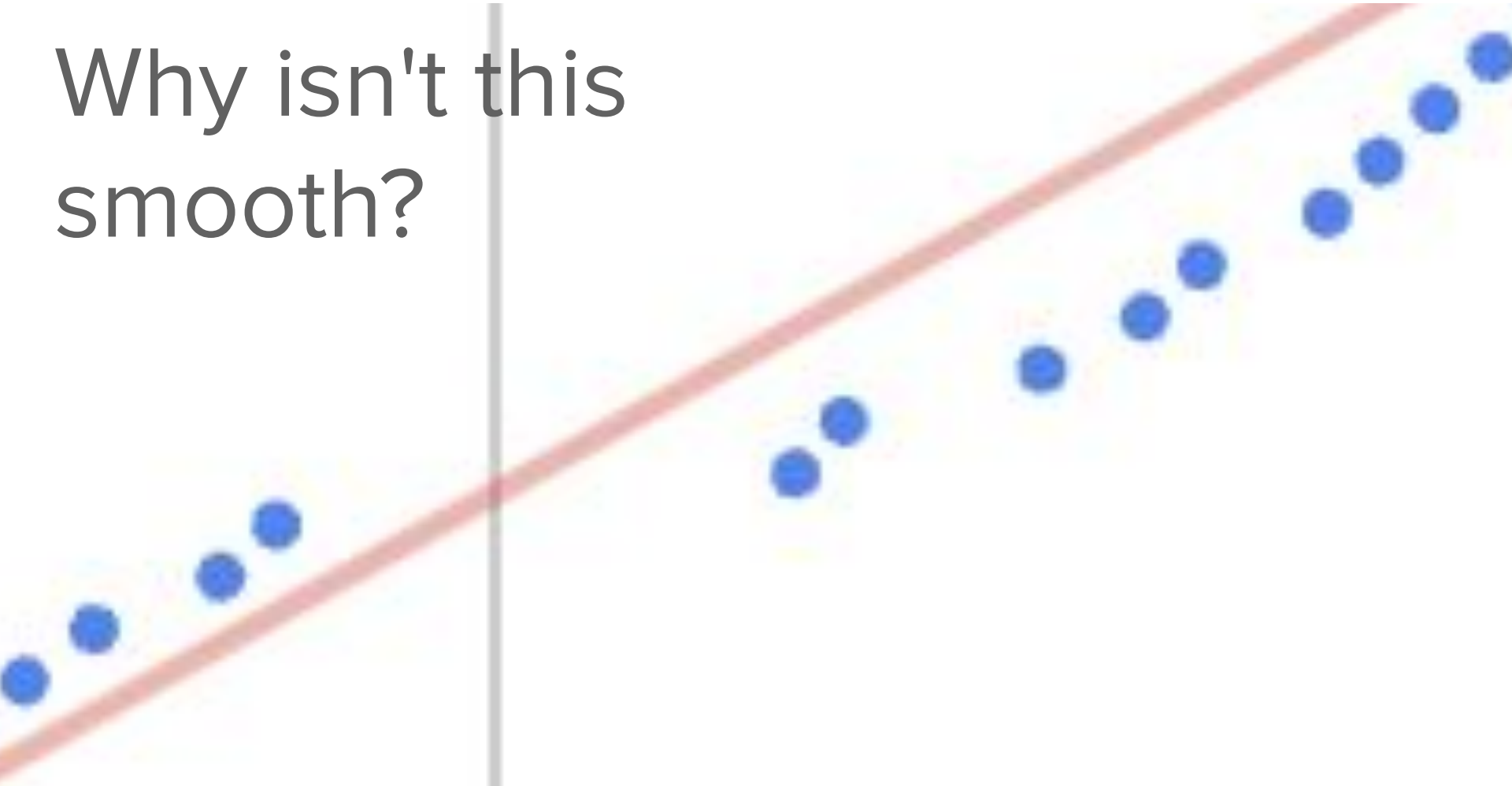
✅ Order of magnitude faster than binary search
✅ Try and preserve exception space saving

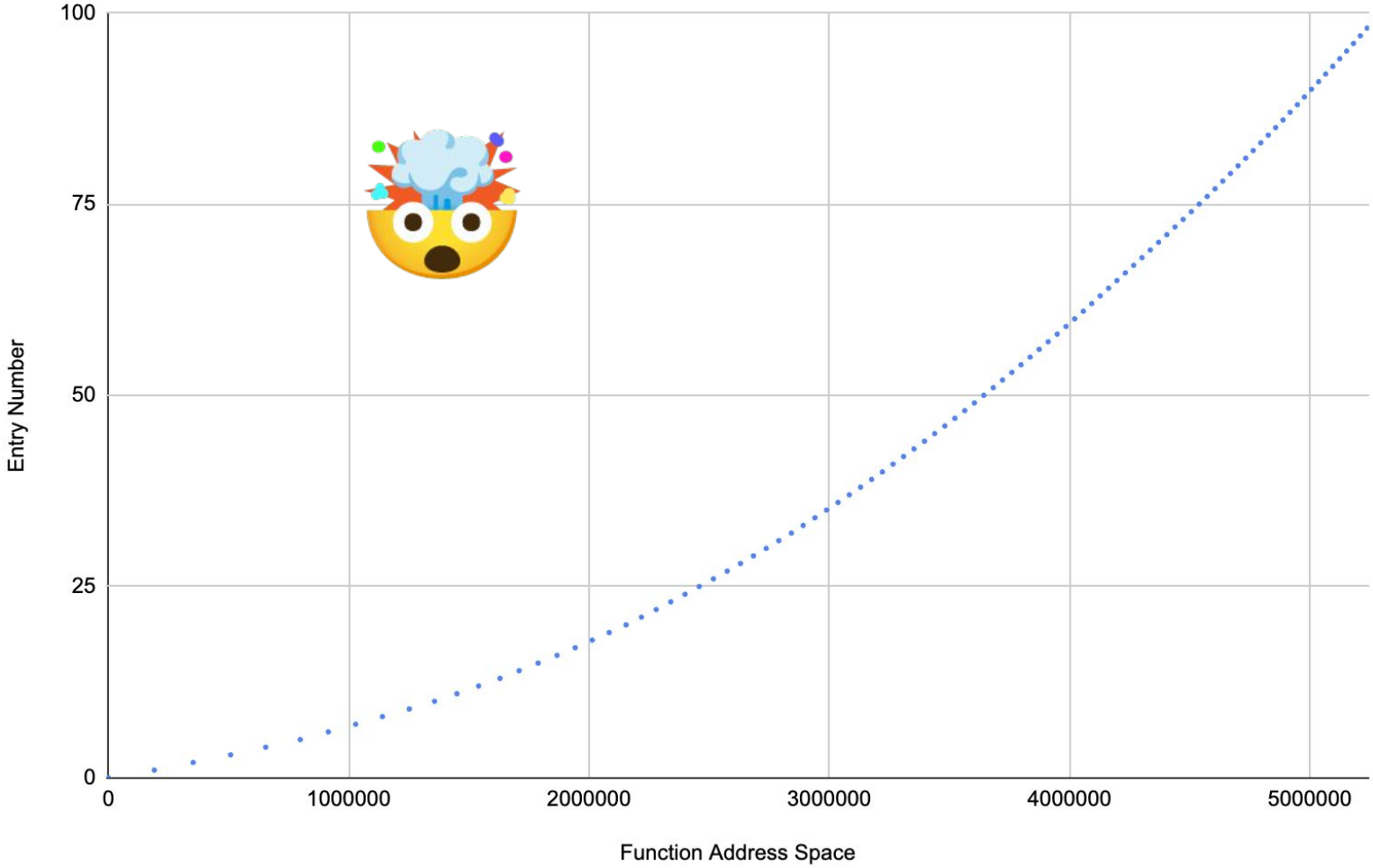
December 2023

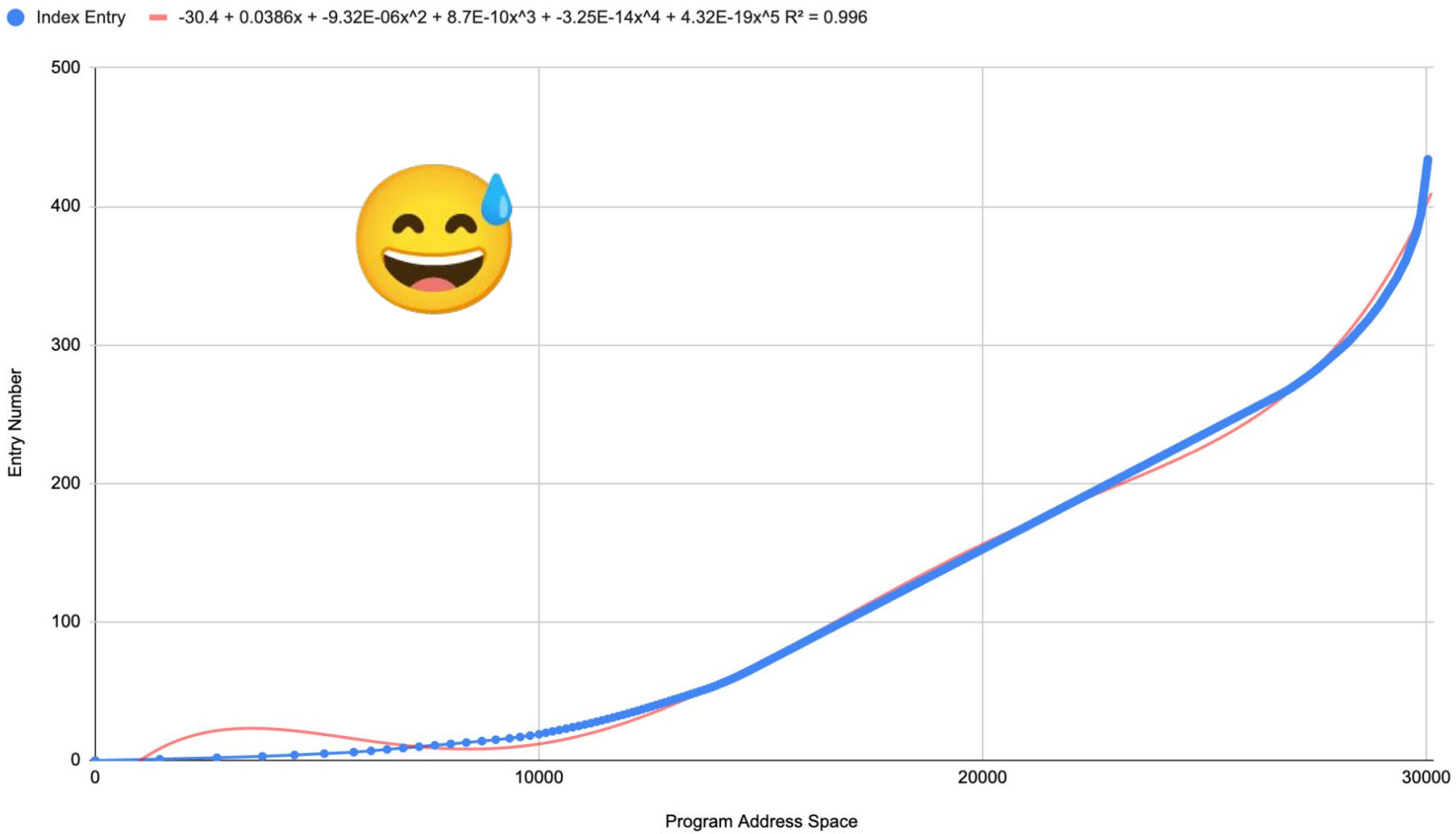
On a flight home from a business trip...


100
75
50
25
0
Entry Number
0
1000000
2000000
3000000
Function Address Space

Why isn't this smooth?




100
75
50
25
0
Entry Number
0
1000000
2000000
3000000
4000000
5000000
Function Address Space


Index Entry and Decimal Addr
Index Entry
-30.4 + 0.0386x + -9.32E-06x^2 + 8.7E-10x^3 + -3.25E-14x^4 + 4.32E-19x^5 R² = 0.996
500
400
300
200
100
0
Entry Number
0
10000
20000
30000
Program Address Space

ISO C++ Meeting Japan 🇯🇵

Host: Woven By Toyota

March 2024

Ben Craig

Oh come on, floating point arithmetic isn't that bad. It performs quite well actually!

The problem is the space budget for embedded devices **without** an **FPU**. The floating point operations would **exceed our ROM**.

Hmmm, alright, then maybe desktop/servers environments could opt for this?

There is a problem there too. In the kernel, using floating point operations **corrupts user mode state**.

Nearpoint Search Algorithm Requirements

- ✅ Order of magnitude faster than binary search
- ✅ Try and preserve exception space saving
- ✅ Integer math ONLY ✨ NEW ✨

Robin Rowe

Yammers on about exceptions on embedded...

Robin Rowe

Yammers on about exceptions on embedded...

Huh... You know. I agree with your premise, but I don't know if I agree with your conclusion.

Robin Rowe

... 5 hours of discussion ...

Robin Rowe

... 5 hours of discussion ...

TrapC: Memory Safe C Programming with No UB

Pr. Robin Rowe <Robin.Rowe@trasec.com>
Number: n3423 - TrapC
Date: 2025-01-07
For: ISO WG14 C Committee meeting at Graz University of Technology, Graz, Austria, February 24 to February 28, 2025

t_str
MEMORY-SAFE C STRING

accu conference 2025

Keynote
C++ Exceptions are Code Compression

Months go by...

ISO C++ Meeting

St. Louis 🇺🇸

June 2024

Herb Sutter

... Shows Herb my current runtime results ...

Very interesting. Now, how do you handle catching exceptions?

My runtime doesn't support polymorphic catch yet. I can only catch types **directly**. I plan to write my own "**dynamic-cast-like**" algorithm for this.

Make sure your dynamic-cast-like implementation has deterministic performance. It should have an **upper bound** on **space** and **time** cost.

Chapter 3. ⚡

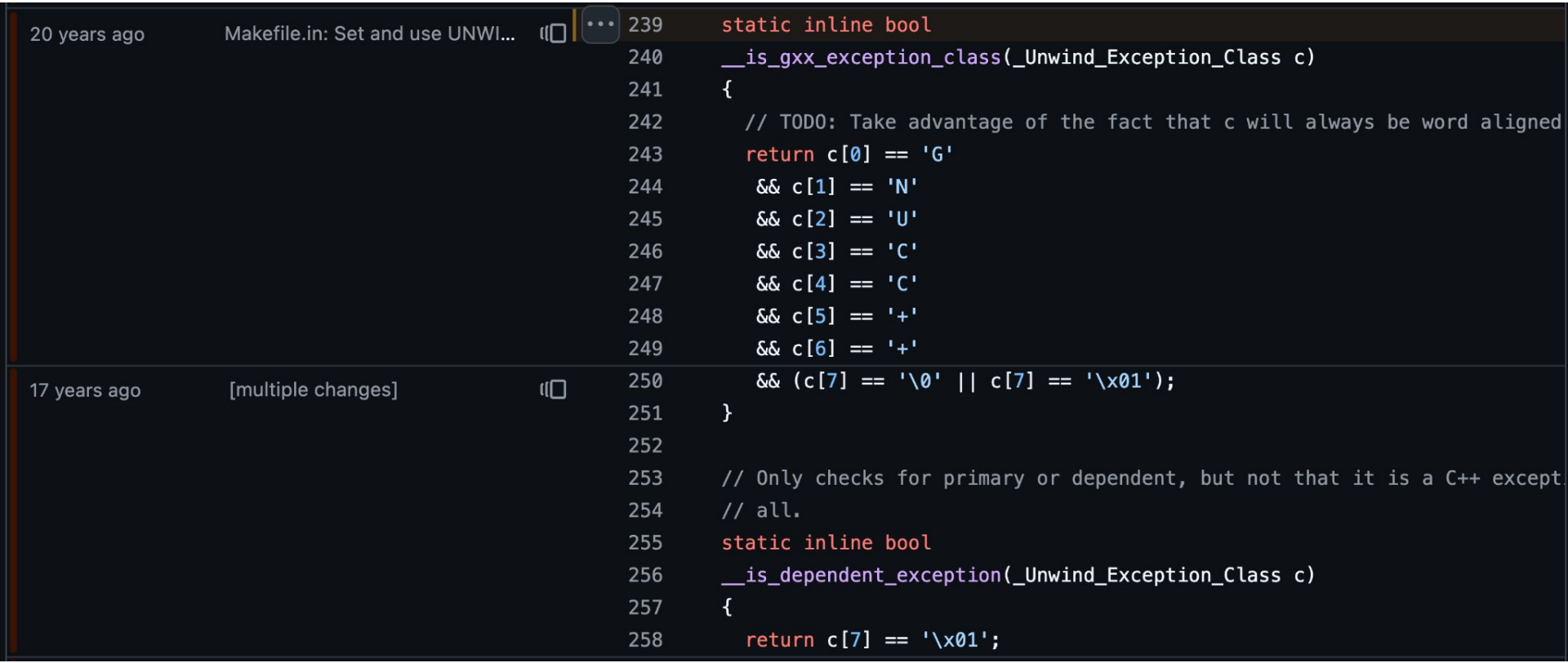
Estell Exception Runtime

Inefficiencies spotted

- Searches index **2x** per frame
- dynamic_cast tree traversal
- Multilang support
- TODOs in source

```
// FILE: gcc/libstdc++-v3/libsupc++/unwind-cxx.h
bool __is_gxx_exception_class(_Unwind_Exception_Class c)
{
 // TODO: Take advantage of the fact that C will always be
 //       word aligned.
 return c[0] == 'G'
  && c[1] == 'N'
  && c[2] == 'U'
  && c[3] == 'C'
  && c[4] == 'C'
  && c[5] == '+'
  && c[6] == '+'
  && (c[7] == '\0' || c[7] == '\x01');
}


// FILE: gcc/libstdc++-v3/libsupc++/eh_arm.cc
__cxa_type_match_result __cxa_type_match() {
 // Check if exception came from GNU C++
 bool foreign_exception
   = !__is_gxx_exception_class(ue_header->exception_class);
}
```

A **TODO** almost old enough to drink 🍺



Optimization #0

Rewrote from scratch in C++

Optimization #1
Single Phase Exceptions

Single Phase Exceptions

Two Phase Exceptions:

1. **Phase 1:** Determine if a suitable catch handler exists. If non exist call `std::terminate`
2. **Phase 2:** Otherwise, unwind stack, perform cleanup, and enter catch handler.

Single Phase

Single Phase Exceptions

Two Phase Exceptions:

1. **Phase 1:** Determine if a suitable catch handler exists. If non exist call `std::terminate`
2. **Phase 2:** Otherwise, unwind stack, perform cleanup, and enter catch handler.

Single Phase

One Phase Exceptions:

1. **Phase 1:** Unwind stack and perform cleanup until you reach a suitable catch handler or a point of termination (noexcept, main, thread)

Single Phase Exceptions

Consequences?

1. With two phases, if a suitable catch block is not found, your terminate handler can do a **core dump**.
2. Applications may depend on destructors only being called if a handler exists.

Positives?

Single Phase

Single Phase Exceptions

Consequences?

1. With two phases, if a suitable catch block is not found, your terminate handler can do a **core dump**.
2. Applications may depend on destructors only being called if a handler exists.

Single Phase

Positives?

1. Faster, without using additional storage
2. **Same behaviour** as using `std::expected`

Optimization #2

TLS Control Block

thread local storage


```
struct exception_control_block
{
 cortex_m_cpu cpu{};
 flattened_hierarchy<4> type_info{};
 std::size_t catch_type_offset = 0;
 void* thrown_object;
 destructor_t* destructor = nullptr;
 exception_cache cache{};
};

thread_local exception_control_block
control_block{};
```

Returned address

Control Block Size:
128B **per thread**
assuming max hierarchy size of 4

vs

128B **per exception**

Optimization #3

Use PMR Allocation

Polymorphic memory resource

```cpp
#include <memory_resource>
/**
* @brief Set the global exception allocator function
*
* @param p_allocator - the global exception memory resource. The lifetime of
* the memory resource should be static.
*/
void set_exception_allocator(std::pmr::memory_resource* p_allocator) noexcept;

/**
* @brief Get the global exception allocator function
*
* @returns the global exception memory allocator
*/
std::pmr::memory_resource* get_exception_allocator() noexcept;
```

Exception Header

```
struct exception_header
{
 std::pmr::memory_resource* allocator = nullptr;
 std::size_t size = 0uz;
 alignas(std::max_align_t) std::byte thrown_object;
};
```

```
void* __wrap___cxa_allocate_exception(std::size_t p_thrown_size)
{
 auto const total_memory = sizeof(exception_header) + p_thrown_size;

 auto* const resource = get_exception_allocator();
 auto* allocated_memory = resource->allocate(total_memory);
 auto* header = static_cast<exception_header*>(allocated_memory);

 header->size = total_memory;
 header->allocator = resource;

 return & header->data;
}
```

Optimization #4

Skipping the first search

```
void __wrap___cxa_throw(void* p_thrown_exception,
                        std::type_info* p_type_info,
                        ke::destructor_t p_destructor) noexcept(false)
{
  auto& control_block = ke::control_block; // Load TLS reference
  control_block.thrown_object = p_thrown_exception;
  control_block.destructor = p_destructor;
  ke::capture_cpu_core(control_block.cpu);
  ke::flatten_rtti(p_thrown_exception, control_block.type_info, p_type_info);
  // Use the expanded unwind instructions to quickly unwind cxa_throw.
  ke::unwind_frame(ke::cxa_throw_unwind_instructions, control_block.cpu);
  // Raise exception returns when an error or call to terminate has been found
  ke::raise_exception(control_block);
  // If raise exception returns it means it failed and we should terminate
  control_block.cache.state(ke::runtime_state::handled_state);
  std::terminate();
}
```

```
void __wrap___cxa_throw(void* p_thrown_exception,
                        std::type_info* p_type_info,
                        ke::destructor_t p_destructor) noexcept(false)
{
  auto& control_block = ke::control_block; // Load TLS reference
  control_block.thrown_object = p_thrown_exception;
  control_block.destructor = p_destructor;
  ke::capture_cpu_core(control_block.cpu);
  ke::flatten_rtti(p_thrown_exception, control_block.type_info, p_type_info);
  // Use the expanded unwind instructions to quickly unwind cxa_throw.
  ke::unwind_frame(ke::cxa_throw_unwind_instructions, control_block.cpu);
  // Raise exception returns when an error or call to terminate has been found
  ke::raise_exception(control_block);
  // If raise exception returns it means it failed and we should terminate
  control_block.cache.state(ke::runtime_state::handled_state);
  std::terminate();
}
```

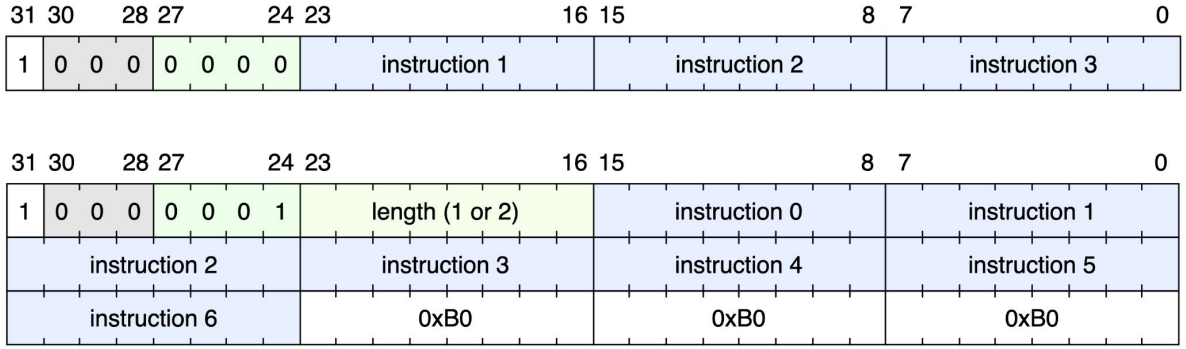
```
template<typename F>
instructions_t cache(F* p_function_to_be_cached)
{
 auto const& entry = ke::get_index_entry(function_address);
 if (entry.has_inlined_personality()) {
   return personality_to_unwind_instructions(&entry.personality_offset);
 } else {
   return spare_instruction();
 }
}

auto const cxa_throw_unwind_instructions = cache(&__wrap___cxa_throw);
auto const cxa_rethrow_unwind_instructions = cache(&__wrap___cxa_rethrow);
```

Computed at **static initialization** time.

Optimization #5

Prearrange unwind instructions

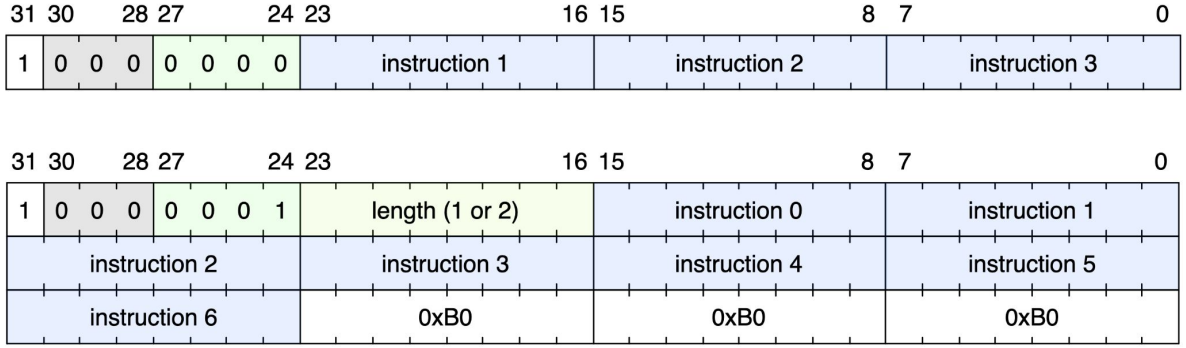
ARM short & long unwind formats




```
      case runtime_state::unwind_frame: {
        auto const instructions =
          create_instructions_from_entry(p_exception_object);
        unwind_frame(instructions, p_exception_object.cpu);
        p_exception_object.cache.state(runtime_state::get_next_frame);
      }
}}}
```

```
[[gnu::always_inline]] _uw8
next_unwind_byte(__gnu_unwind_state* uws)
 {
   _uw8 b;
   if (uws->bytes_left == 0) {
     /* Load another word */
     if (uws->words_left == 0)
       return 0xB0; /* Nothing left.  */
     uws->words_left--;
     uws->data = *(uws->next++);
     uws->bytes_left = 3;
   } else
     uws->bytes_left--;
   /* Extract the most significant byte.  */
   b = (uws->data >> 24) & 0xff;
   uws->data <<= 8;
   return b;
 }
```

ARM short & long unwind formats




```
    case runtime_state::unwind_frame: {
      auto const instructions =
        create_instructions_from_entry(p_exception_object);
      unwind_frame(instructions, p_exception_object.cpu);
      p_exception_object.cache.state(runtime_state::get_next_frame);
    }
}}}
```

```
struct instructions_t
{
 std::array<std::uint8_t, 8> data{
   arm_ehabi::finish, arm_ehabi::finish,
   arm_ehabi::finish, arm_ehabi::finish,
   arm_ehabi::finish, arm_ehabi::finish,
   arm_ehabi::finish, arm_ehabi::finish,
 };
};
```

Optimization #6

JIT unwinding

just in time

```
void unwind_frame(instructions_t const& p_instructions,
                  cortex_m_cpu& p_cpu)
{
 static constexpr std::array<void*, 256> jump_table{
    &&vsp_add_0,    // [0]
    &&vsp_add_1,    // [1]
    &&vsp_add_2,    // [2]
    &&vsp_add_3,    // [3]
    &&vsp_add_4,    // [4]
    &&vsp_add_5,    // [5]
    &&vsp_add_6,    // [6]
    &&vsp_add_7,    // [7]
    &&vsp_add_8,    // [8]
    &&vsp_add_9,    // [9]
    &&vsp_add_10,   // [10]
    &&vsp_add_11,   // [11]
```

ARM Unwind instructions are 8-bits in size.

Cost = 256 * 4B = **1024 B**

GCC label address extension

```
&&vsp_add_11, // [11]
```

```
auto instruction_ptr = p_instructions.data.begin();
auto const* sp_ptr = *p_cpu.sp;

while (true) {
  auto const* instruction_handler = jump_table[*instruction_ptr++];
  goto *instruction_handler;

finish_unwind:
  p_cpu.sp = sp_ptr;
  [[likely]] if (move_lr_to_pc) {
    p_cpu.pc = p_cpu.lr;
  }
  break;
reserved_or_spare_thus_terminate:
  std::terminate();
  break;
```

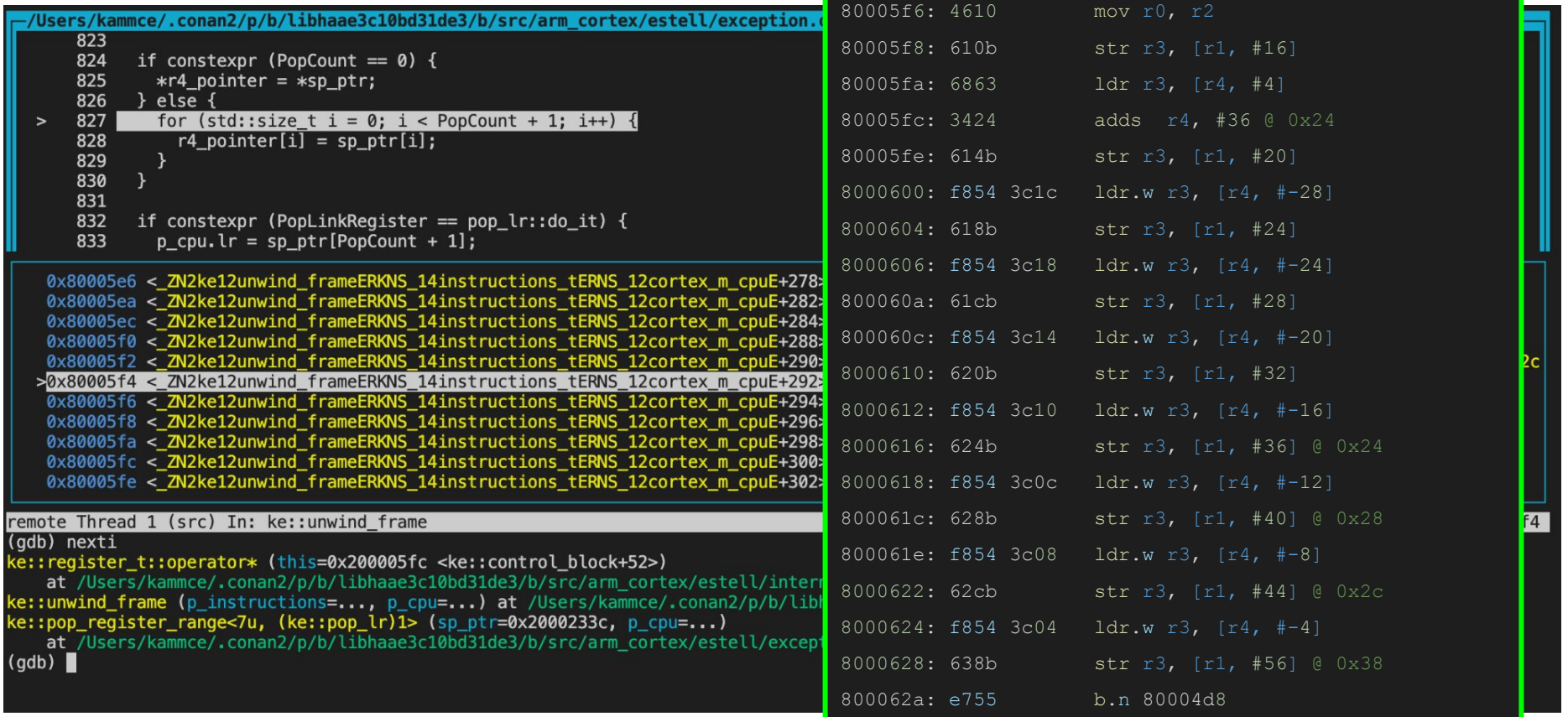
```
 //
+=============================================================================+
 // |                       Sequentially Pop Registers + LR
|
 //
+=============================================================================+
 pop_off_stack_r4_to_r11_and_lr :
   sp_ptr = pop_register_range<7, pop_lr::do_it>(sp_ptr, p_cpu);
   continue;
 pop_off_stack_r4_to_r10_and_lr :
   sp_ptr = pop_register_range<6, pop_lr::do_it>(sp_ptr, p_cpu);
   continue;
 pop_off_stack_r4_to_r9_and_lr :
   sp_ptr = pop_register_range<5, pop_lr::do_it>(sp_ptr, p_cpu);
   continue;
 pop_off_stack_r4_to_r8_and_lr :
   sp_ptr = pop_register_range<4, pop_lr::do_it>(sp_ptr, p_cpu);
   continue;
```

```
template<size_t PopCount, pop_lr PopLinkRegister = pop_lr::skip>
[[nodiscard("The result of this function MUST be assigned to reg SP")]]
std::uint32_t const* pop_register_range(
      std::uint32_t const* sp_ptr,
      cortex_m_cpu& p_cpu) {
 auto* r4_pointer = &p_cpu.r4.data;
 static_assert(PopCount <= 7, "Pop Count cannot be above 7");
 if constexpr (PopCount == 0) {
   *r4_pointer = *sp_ptr;
 } else {
   for (std::size_t i = 0; i < PopCount + 1; i++) {
     r4_pointer[i] = sp_ptr[i];
   }
 }
 if constexpr (PopLinkRegister == pop_lr::do_it) {
   p_cpu.lr = sp_ptr[PopCount + 1];
 }
 return sp_ptr + PopCount + 1 +
            unsigned{ PopLinkRegister == pop_lr::do_it };
}
```

The only runtime conditional

```
template<size_t PopCount, pop_lr PopLinkRegister = pop_lr::skip>
[[nodiscard("The result of this function MUST be assigned to reg SP")]]
std::uint32_t const* pop_register_range(
      std::uint32_t const* sp_ptr,
      cortex_m_cpu& p_cpu) {
 auto* r4_pointer = &p_cpu.r4.data;
 static_assert(PopCount <= 7, "Pop Count cannot be above 7");
 if constexpr (PopCount == 0) {
   *r4_pointer = *sp_ptr;
 } else {
   for (std::size_t i = 0; i < PopCount + 1; i++) {
     r4_pointer[i] = sp_ptr[i];
   }
 }
 if constexpr (PopLinkRegister == pop_lr::do_it) {
   p_cpu.lr = sp_ptr[PopCount + 1];
 }
 return sp_ptr + PopCount + 1 +
            unsigned{ PopLinkRegister == pop_lr::do_it };
}
```

The only runtime conditional


/Users/kammce/.conan2/p/b/libhaae3c10bd31de3/b/src/arm_cortex/estell/exception.
823
824 if constexpr (PopCount == 0) {
825 *r4_pointer = *sp_ptr;
826 } else {
827 for (std::size_t i = 0; i < PopCount + 1; i++) {
828 r4_pointer[i] = sp_ptr[i];
829 }
830 }
831
832 if constexpr (PopLinkRegister == pop_lr::do_it) {
833 p_cpu.lr = sp_ptr[PopCount + 1];
0x80005e6 <_ZN2ke12unwind_frameERKNS_14instructions_tERNS_12cortex_m_cpuE+278>
0x80005ea <_ZN2ke12unwind_frameERKNS_14instructions_tERNS_12cortex_m_cpuE+282>
0x80005ec <_ZN2ke12unwind_frameERKNS_14instructions_tERNS_12cortex_m_cpuE+284>
0x80005f0 <_ZN2ke12unwind_frameERKNS_14instructions_tERNS_12cortex_m_cpuE+288>
0x80005f2 <_ZN2ke12unwind_frameERKNS_14instructions_tERNS_12cortex_m_cpuE+290>
>0x80005f4 <_ZN2ke12unwind_frameERKNS_14instructions_tERNS_12cortex_m_cpuE+292>
0x80005f6 <_ZN2ke12unwind_frameERKNS_14instructions_tERNS_12cortex_m_cpuE+294>
0x80005f8 <_ZN2ke12unwind_frameERKNS_14instructions_tERNS_12cortex_m_cpuE+296>
0x80005fa <_ZN2ke12unwind_frameERKNS_14instructions_tERNS_12cortex_m_cpuE+298>
0x80005fc <_ZN2ke12unwind_frameERKNS_14instructions_tERNS_12cortex_m_cpuE+300>
0x80005fe <_ZN2ke12unwind_frameERKNS_14instructions_tERNS_12cortex_m_cpuE+302>
remote Thread 1 (src) In: ke::unwind_frame
(gdb) nexti
ke::register_t::operator* (this=0x200005fc <ke::control_block+52>)
at /Users/kammce/.conan2/p/b/libhaae3c10bd31de3/b/src/arm_cortex/estell/interr
ke::unwind_frame (p_instructions=..., p_cpu=...) at /Users/kammce/.conan2/p/b/libh
ke::pop_register_range<7u, (ke::pop_lr)1> (sp_ptr=0x2000233c, p_cpu=...)
at /Users/kammce/.conan2/p/b/libhaae3c10bd31de3/b/src/arm_cortex/estell/except
(gdb)
80005f4: 6823 ldr r3, [r4, #0]
80005f6: 4610 mov r0, r2
80005f8: 610b str r3, [r1, #16]
80005fa: 6863 ldr r3, [r4, #4]
80005fc: 3424 adds r4, #36 @ 0x24
80005fe: 614b str r3, [r1, #20]
8000600: f854 3c1c ldr.w r3, [r4, #-28]
8000604: 618b str r3, [r1, #24]
8000606: f854 3c18 ldr.w r3, [r4, #-24]
800060a: 61cb str r3, [r1, #28]
800060c: f854 3c14 ldr.w r3, [r4, #-20]
8000610: 620b str r3, [r1, #32]
8000612: f854 3c10 ldr.w r3, [r4, #-16]
8000616: 624b str r3, [r1, #36] @ 0x24
8000618: f854 3c0c ldr.w r3, [r4, #-12]
800061c: 628b str r3, [r1, #40] @ 0x28
800061e: f854 3c08 ldr.w r3, [r4, #-8]
8000622: 62cb str r3, [r1, #44] @ 0x2c
8000624: f854 3c04 ldr.w r3, [r4, #-4]
8000628: 638b str r3, [r1, #56] @ 0x38
800062a: e755 b.n 80004d8
<ke::unwind_frame(ke::instructions_t const&,
ke::cortex_m_cpu&)+0x8>

Optimization #7

Flatten RTTI hierarchy

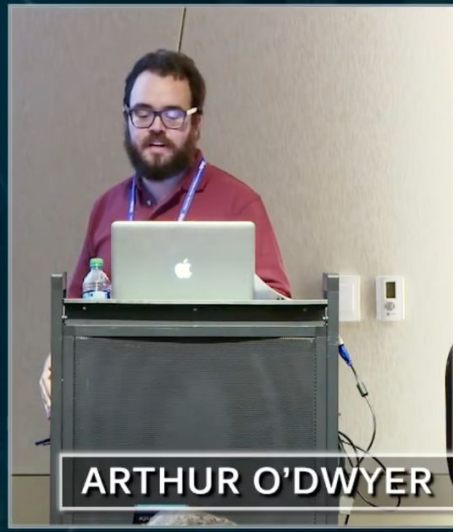
run time type information

Possible implementation

```
template<class P, class From, class To = remove_pointer_t<P>>
To *dynamicast(From *p) {
    if constexpr (is_same_v<From, To>) {
        return p;
    } else if constexpr (is_base_of_v<To, From>) {
        return (To *)(p);
    } else if constexpr (is_void_v<To>) {
        return truly_dynamic_to_mdo(p);
    } else if constexpr (is_base_of_v<From, To>) {
        return truly_dynamic_from_base_to_derived<To>(p);
    } else {
        return truly_dynamic_between_unrelated_classes<To>(p);
    }
}
```

Technically we should also verify that To is an *accessible* base of From in the current lexical scope...

59



dynamic_cast
From Scratch

RTTI for classes hierarchies

Class type with no inheritance

```
class std_exception_rtti :
public __class_type_info {
 const char* name;
};
```

8 B

Class type with one parent

```
class std_logic_error_rtti :
public __si_class_type_info {
 const char* name;
 std::type_info* parent;
};
```

12 B

Class type with multiple parents

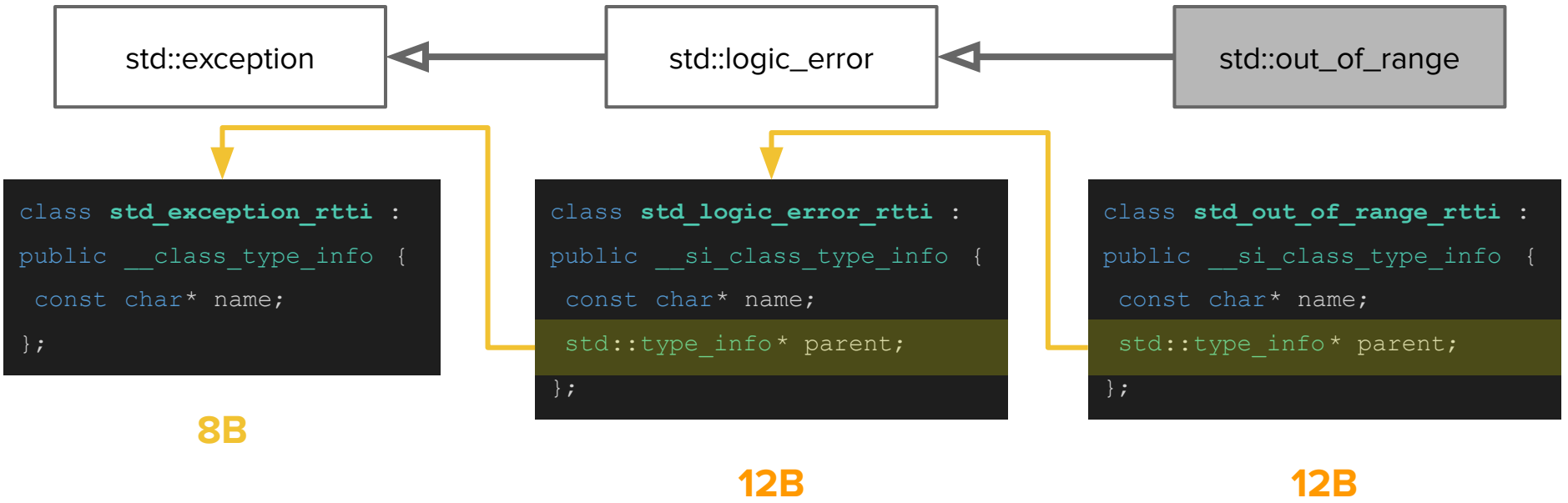
```
class multi_inherit_error_rtti :
public __vmi_class_type_info {
 const char* name;
 long flags;
 size_t length;
 type_info_and_offset parents[];
};
```

20 B + (length * 8B)

RTTI for classes hierarchies

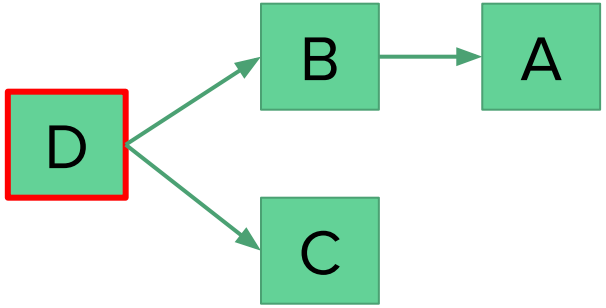
Cost = 8B + 12B + 12B = **32 B**

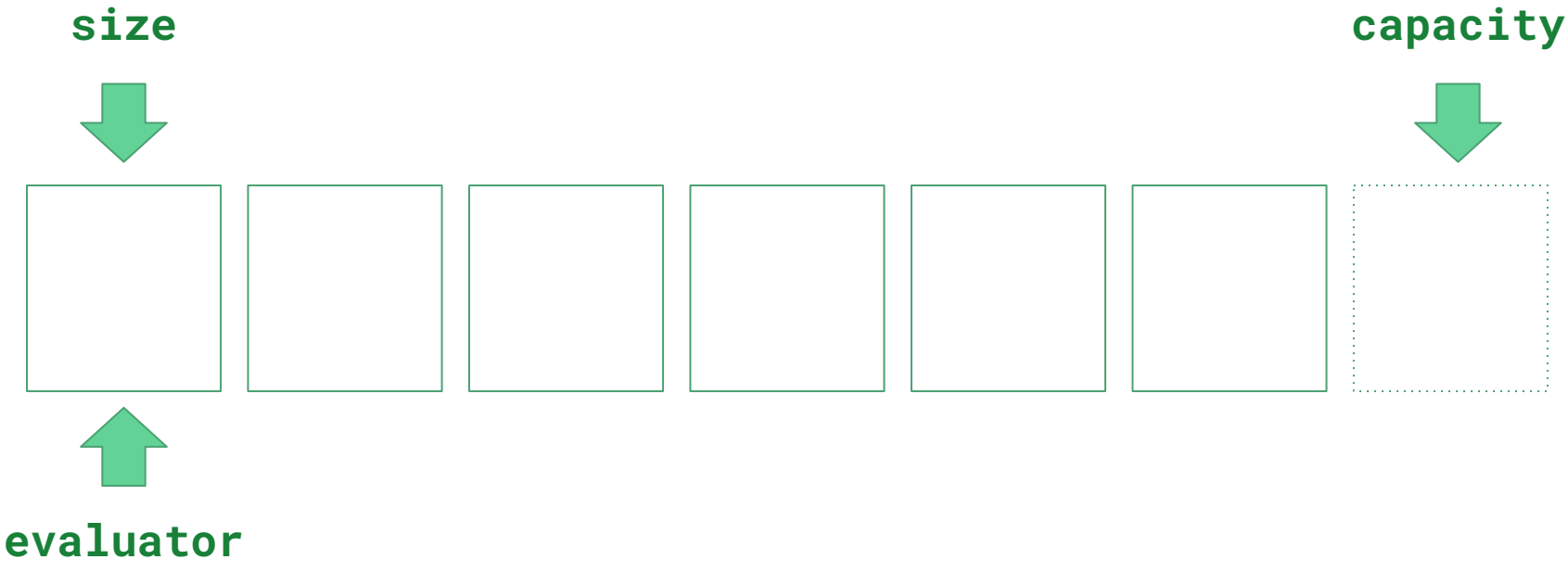
🤫 *Ignoring the size of strings*
🤫 *Ignoring the size of base classes*



`std::fixed_vector<void*,7>`

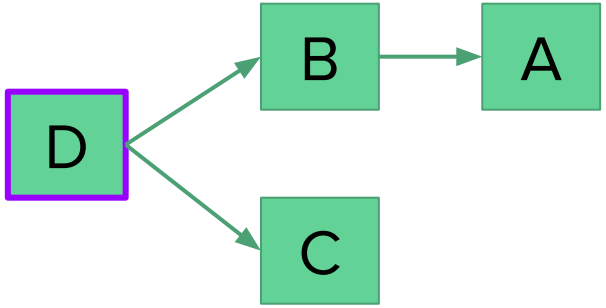
Flattening Algorithm





std::fixed_vector<void*,7>

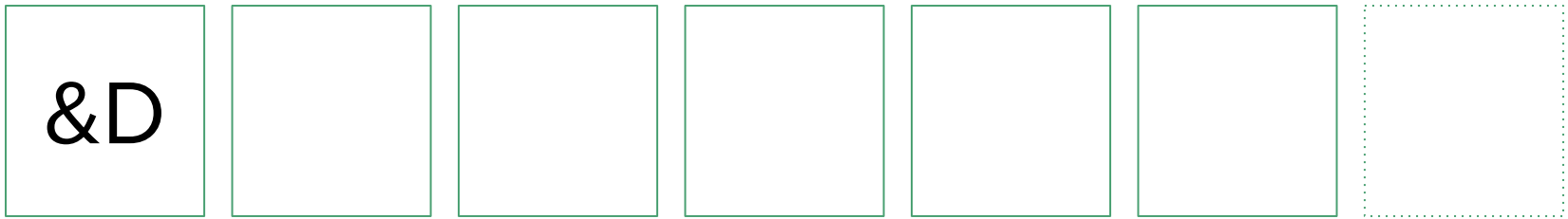
Flattening Algorithm




```
class multi_inherit_error_rtti :
public __vmi_class_type_info {
 const char* name;
 long flags;
 size_t length;
 type_info_and_offset parents[];
};
```

size

capacity

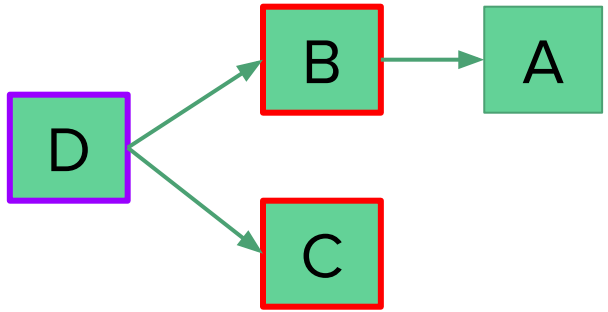


evaluator

```
for(parent : parents) { types.push_back(parent);}
```

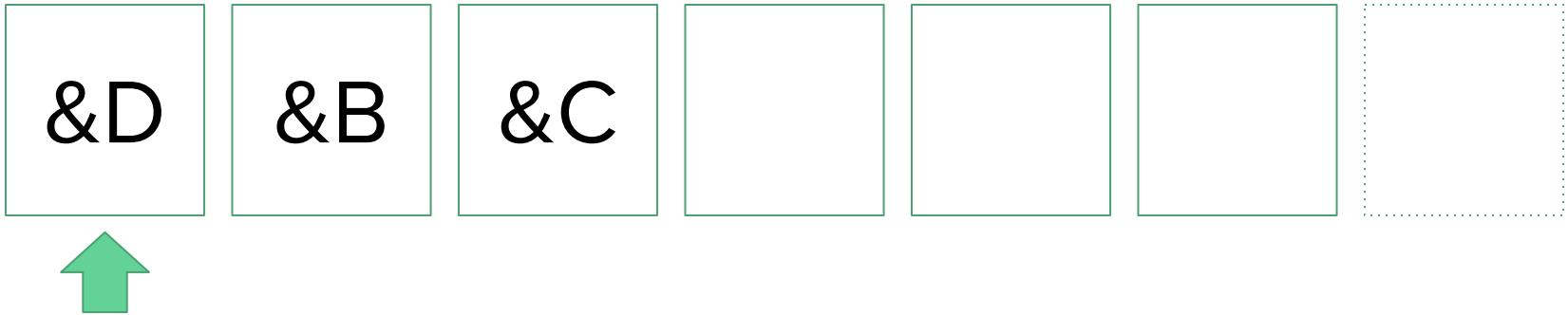
std::fixed_vector<void*,7>

Flattening Algorithm



size

capacity



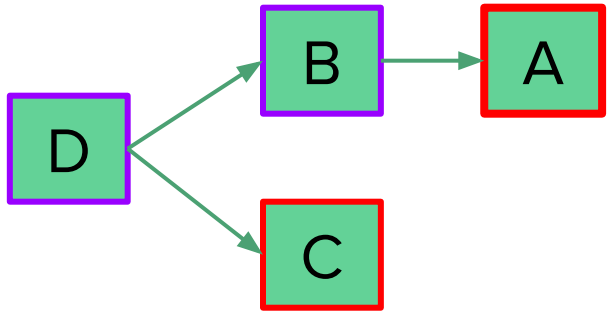
evaluator

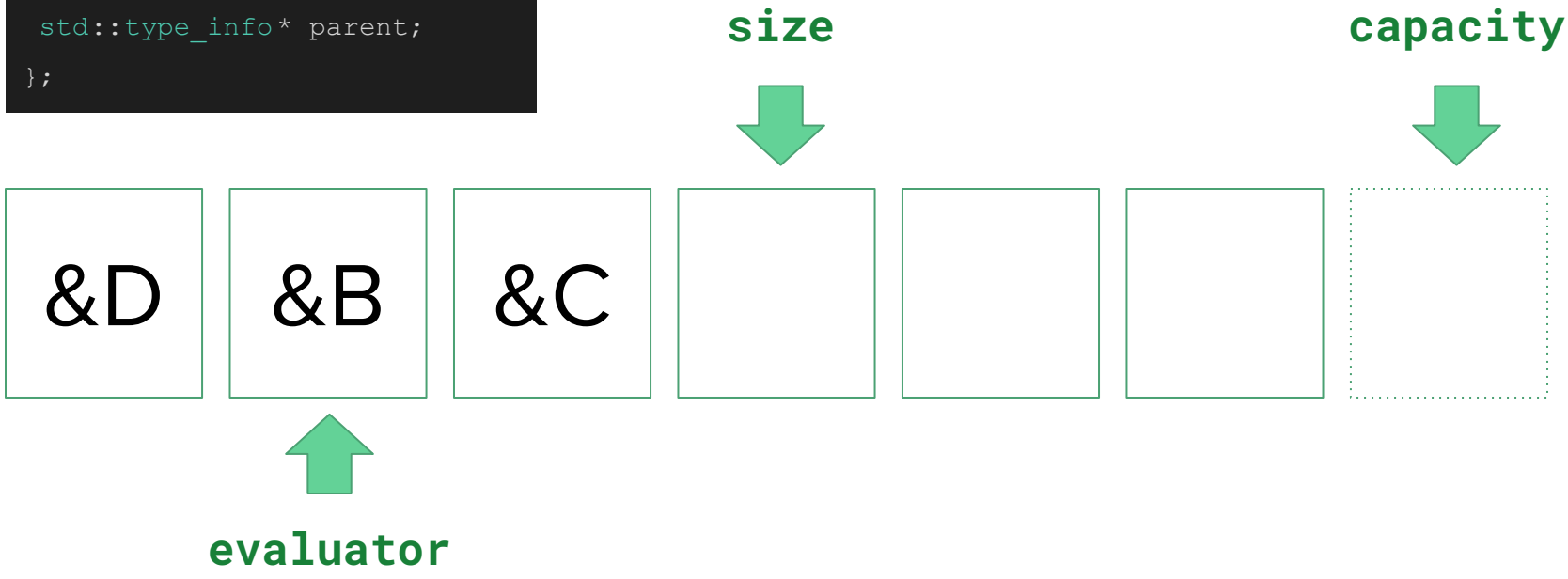
```
for(parent : parents) { types.push_back(parent);}
```

std::fixed_vector<void*,7>

Flattening Algorithm

```
class std_logic_error_rtti :
public __si_class_type_info {
 const char* name;
 std::type_info* parent;
};
```

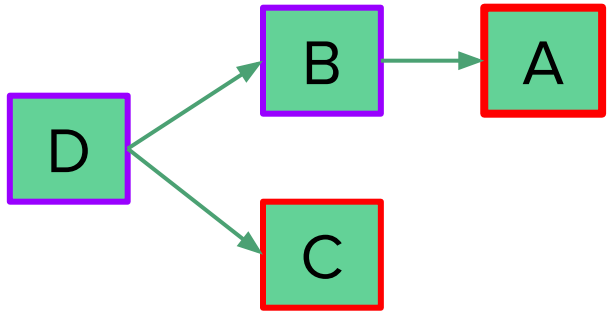


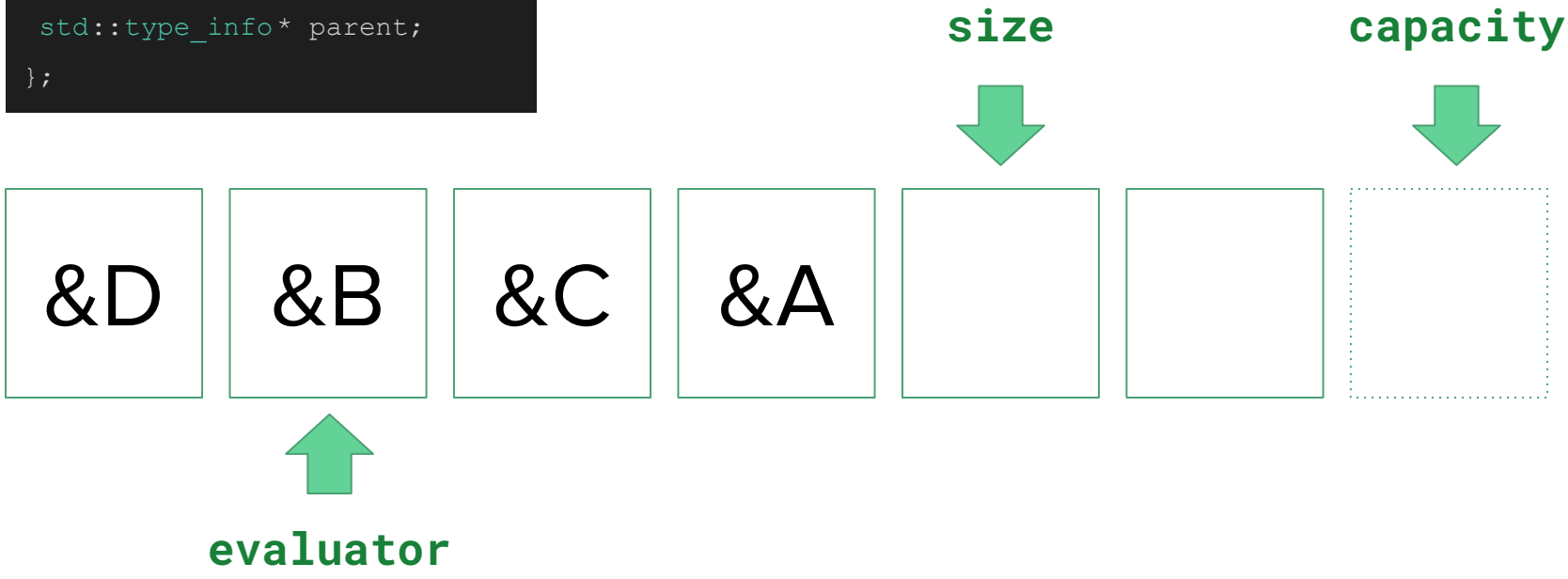


std::fixed_vector<void*,7>

Flattening Algorithm

```
class std_logic_error_rtti :
public __si_class_type_info {
 const char* name;
 std::type_info* parent;
};
```

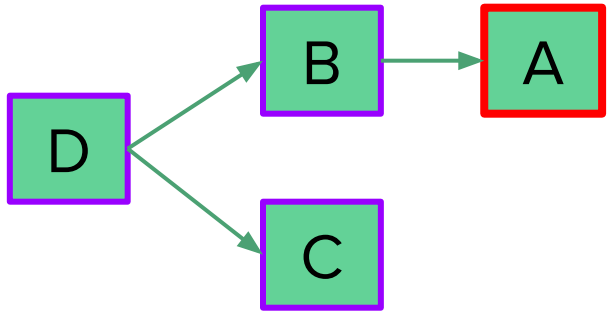


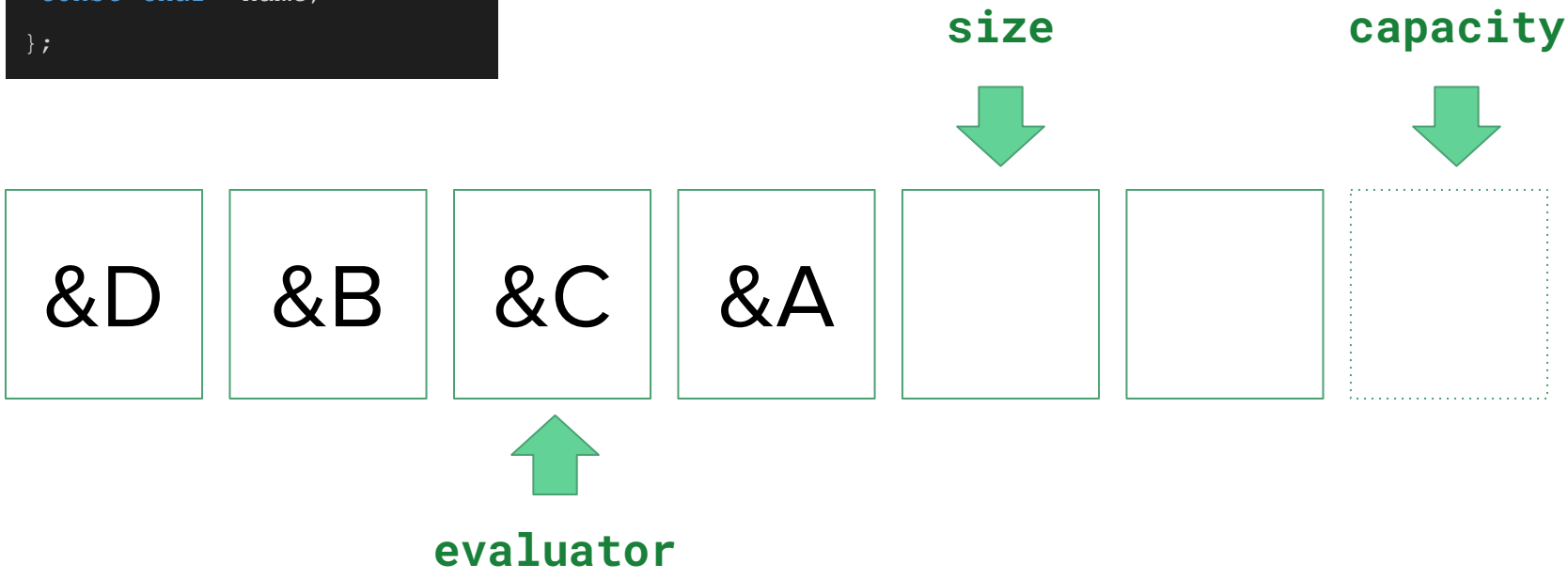


std::fixed_vector<void*,7>

Flattening Algorithm

```
class std_exception_rtti :
public __class_type_info {
 const char* name;
};
```

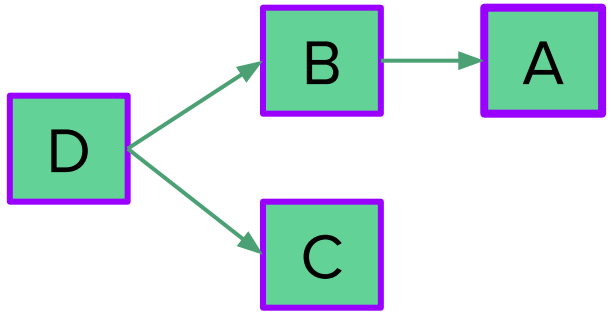


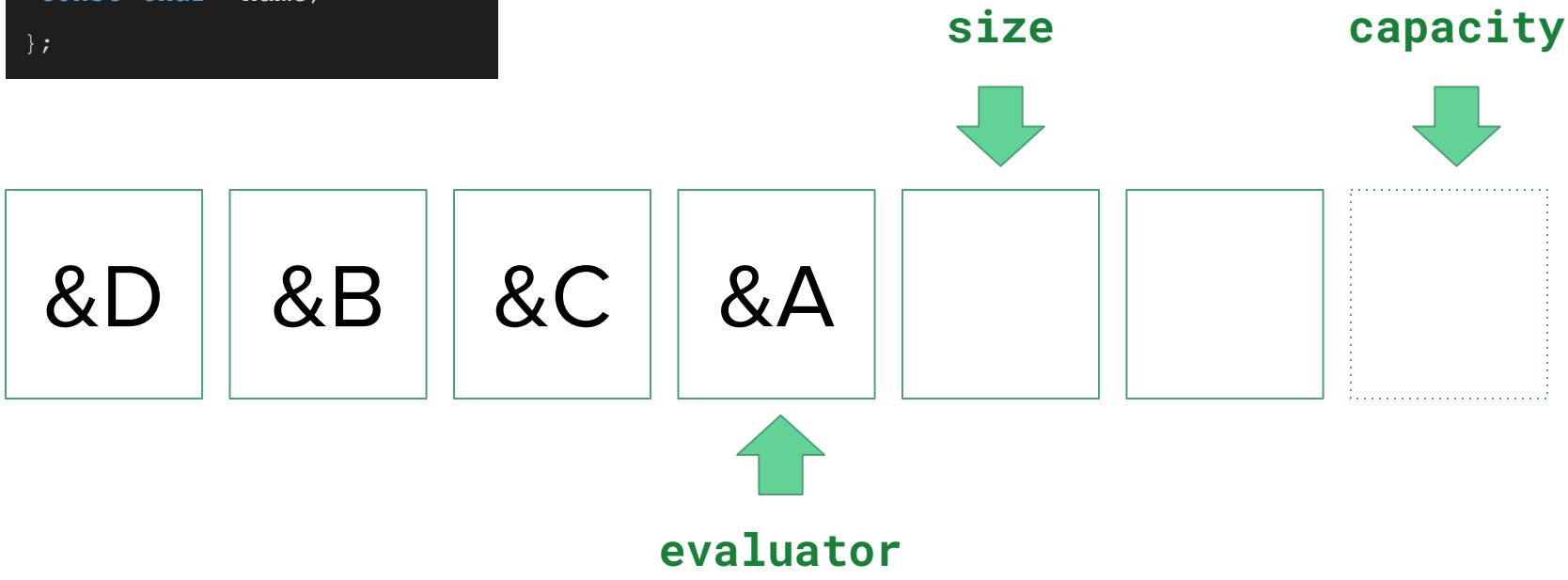


std::fixed_vector<void*,7>

Flattening Algorithm

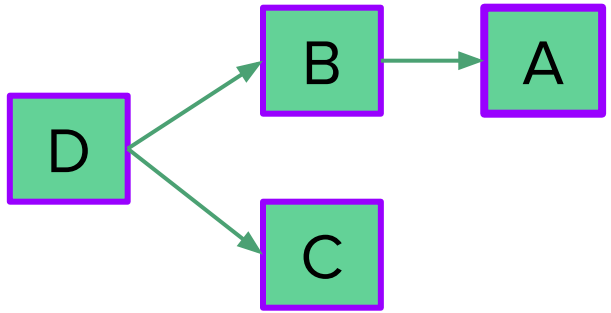
```
class std_exception_rtti :
public __class_type_info {
 const char* name;
};
```

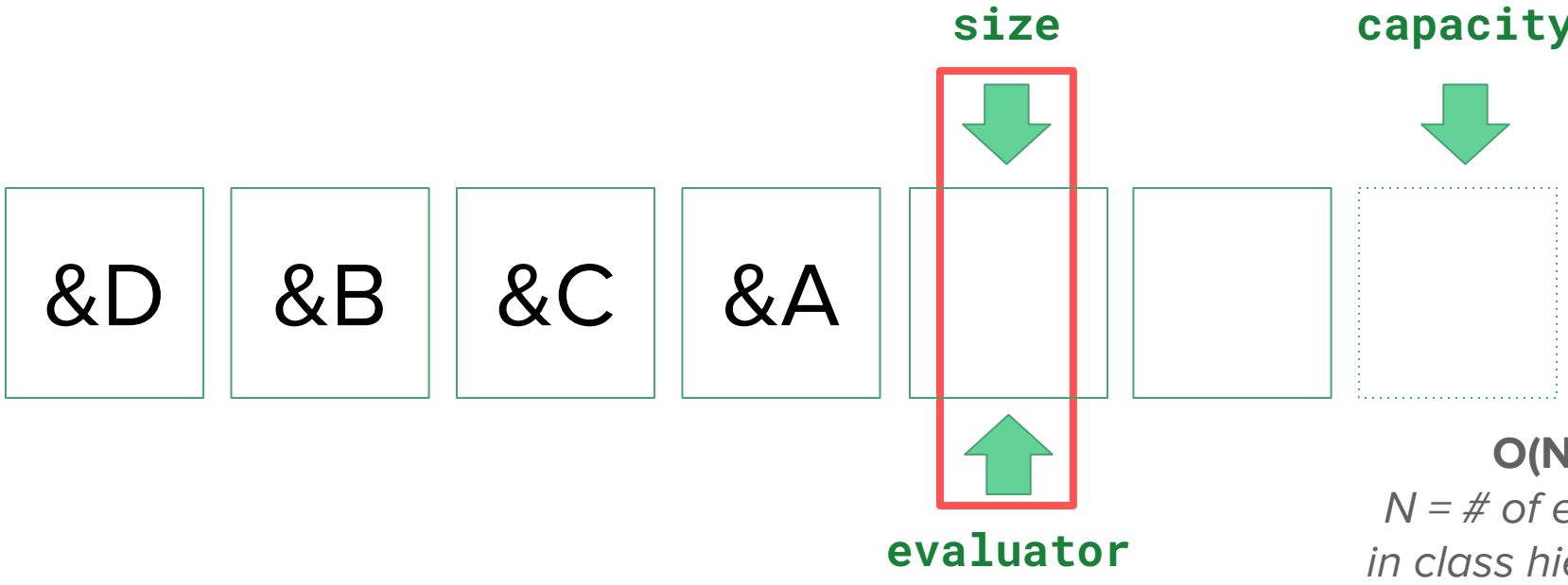




`std::fixed_vector<void*,7>`

Flattening Algorithm



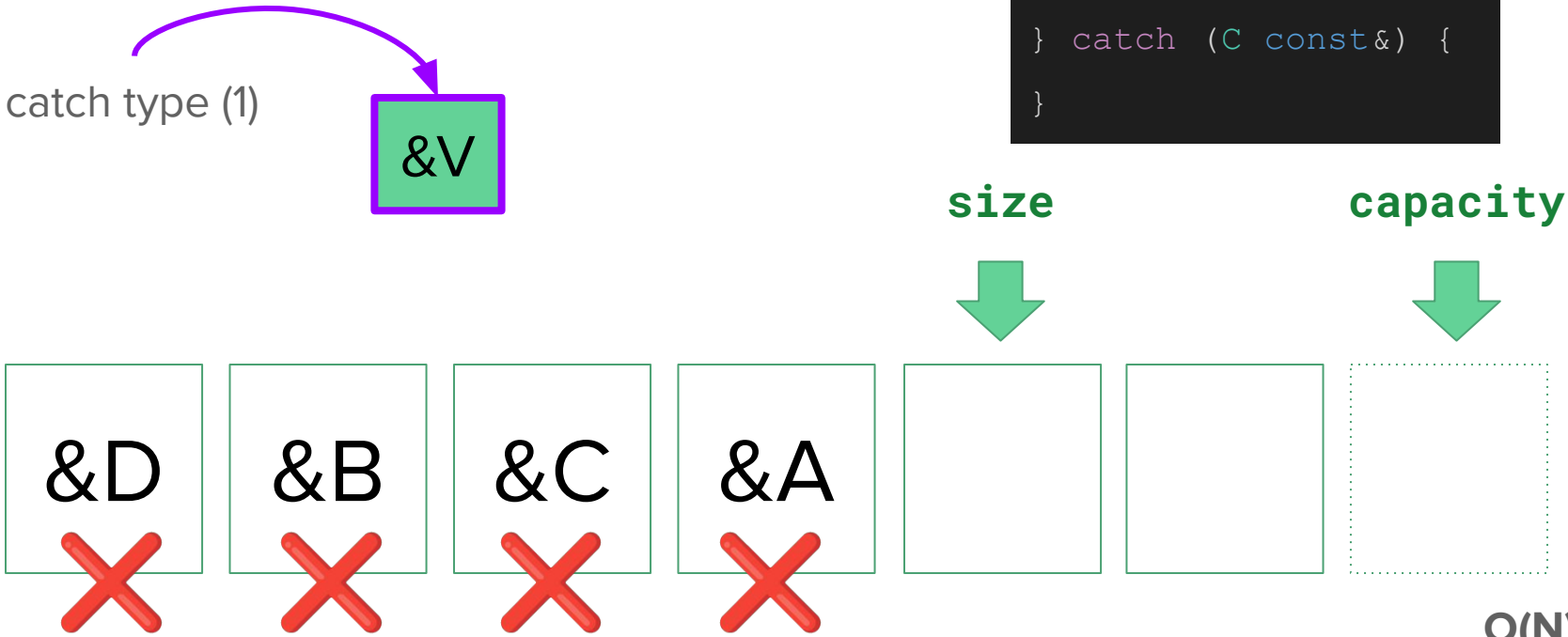


O(N)

N = # of entities in class hierarchy

Dynamic_cast Algorithm

```
try {
 // stuff...
} catch (V const&) {
} catch (C const&) {
}
```

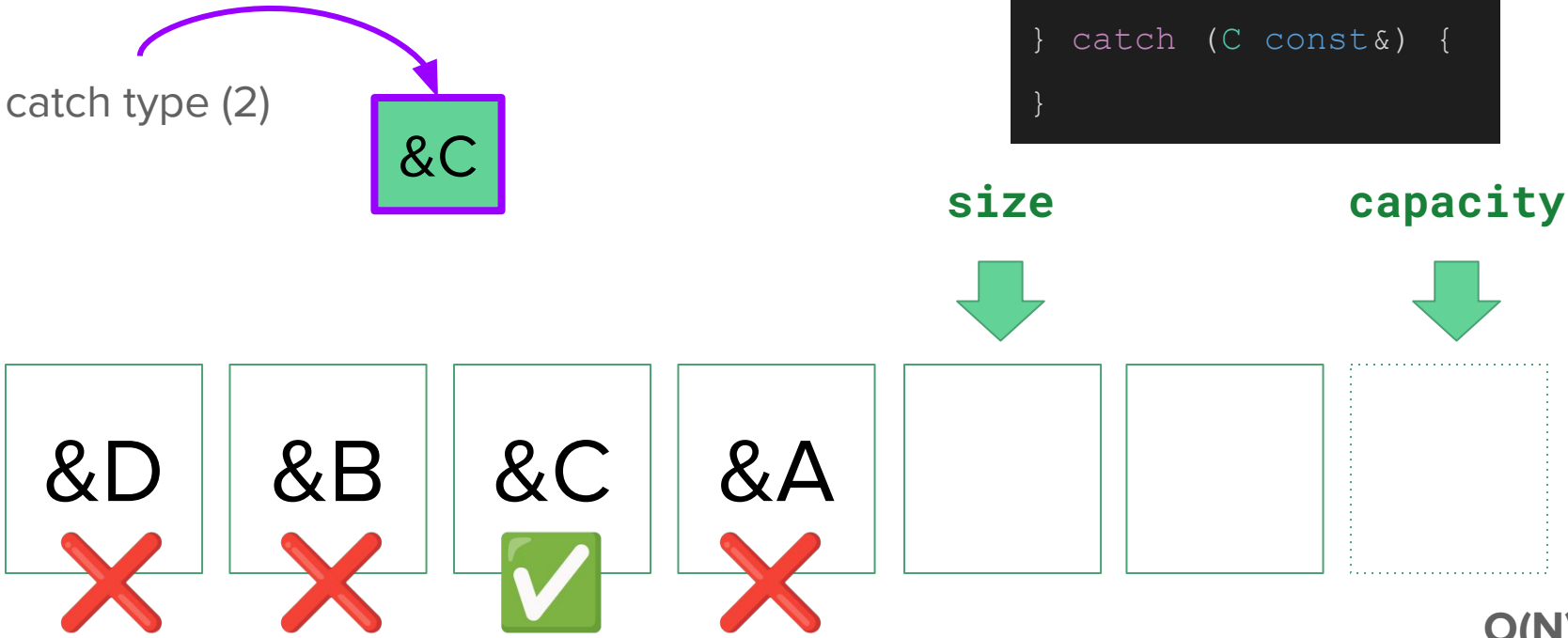


O(N)

N = # of entities in class hierarchy

Dynamic_cast Algorithm

```
try {
 // stuff...
} catch (V const&) {
} catch (C const&) {
}
```



O(N)

N = # of entities in class hierarchy

"dynamic_cast" Algorithm

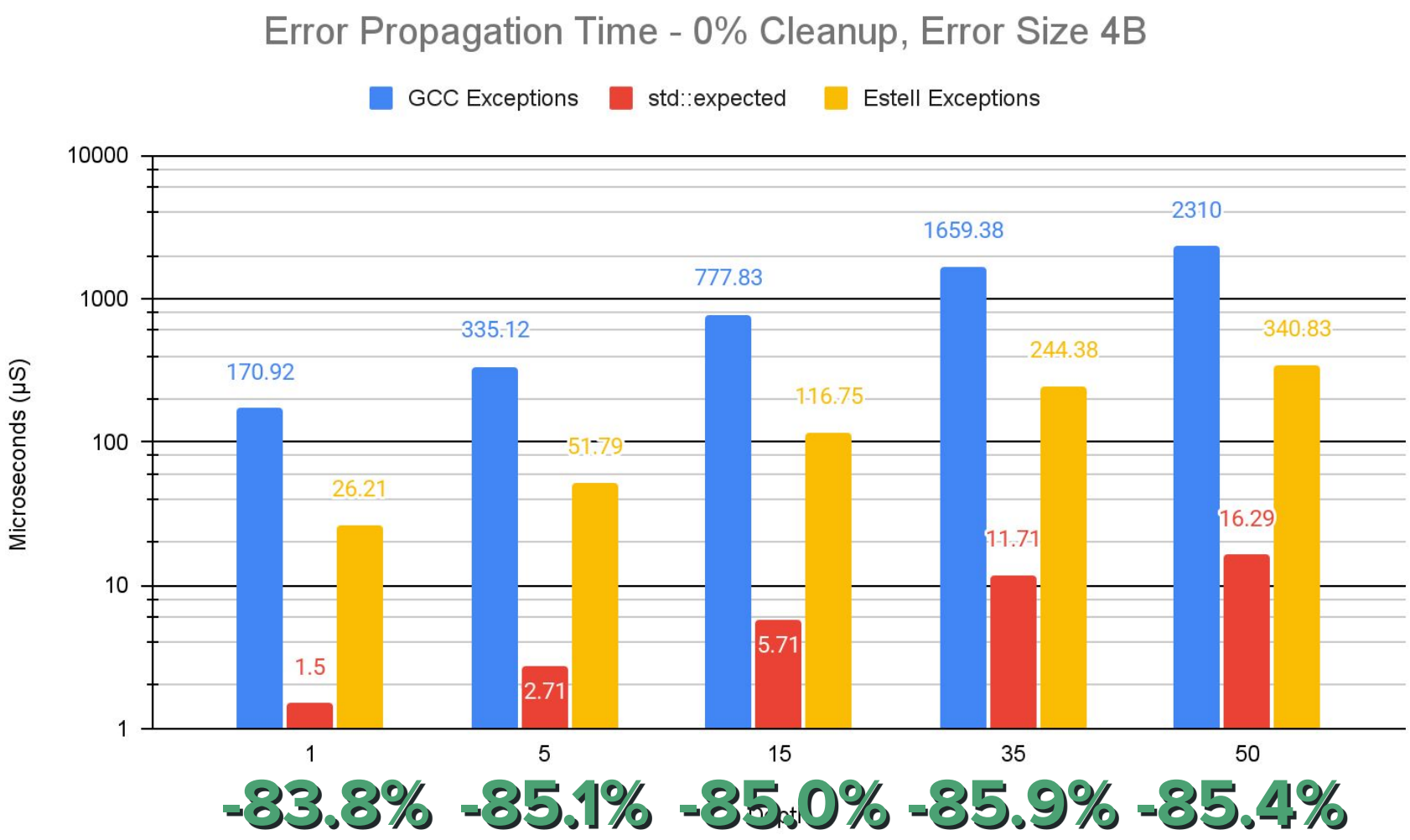
O(N*M)

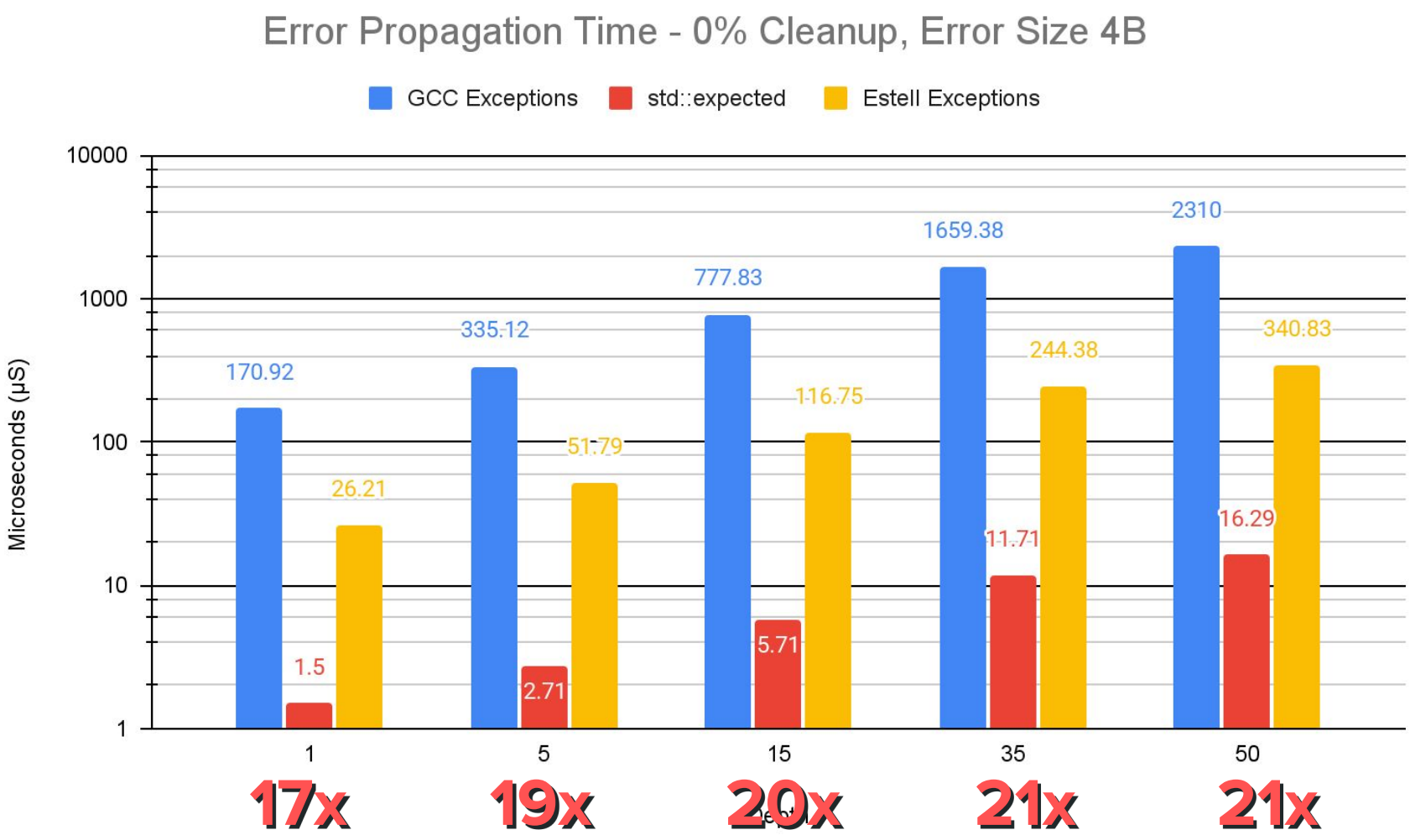
N = # of entities in class hierarchy

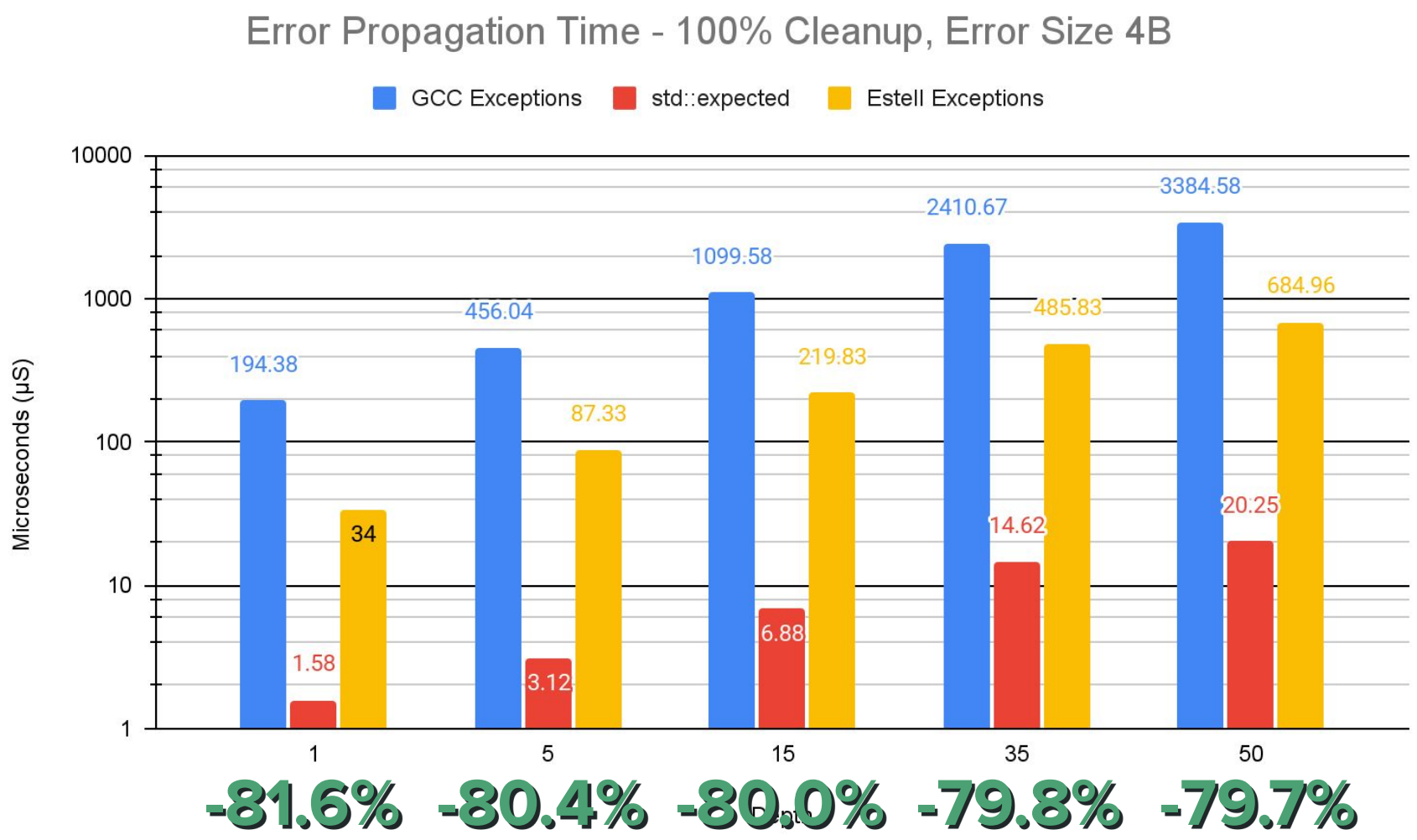
M = # of catches until handler found

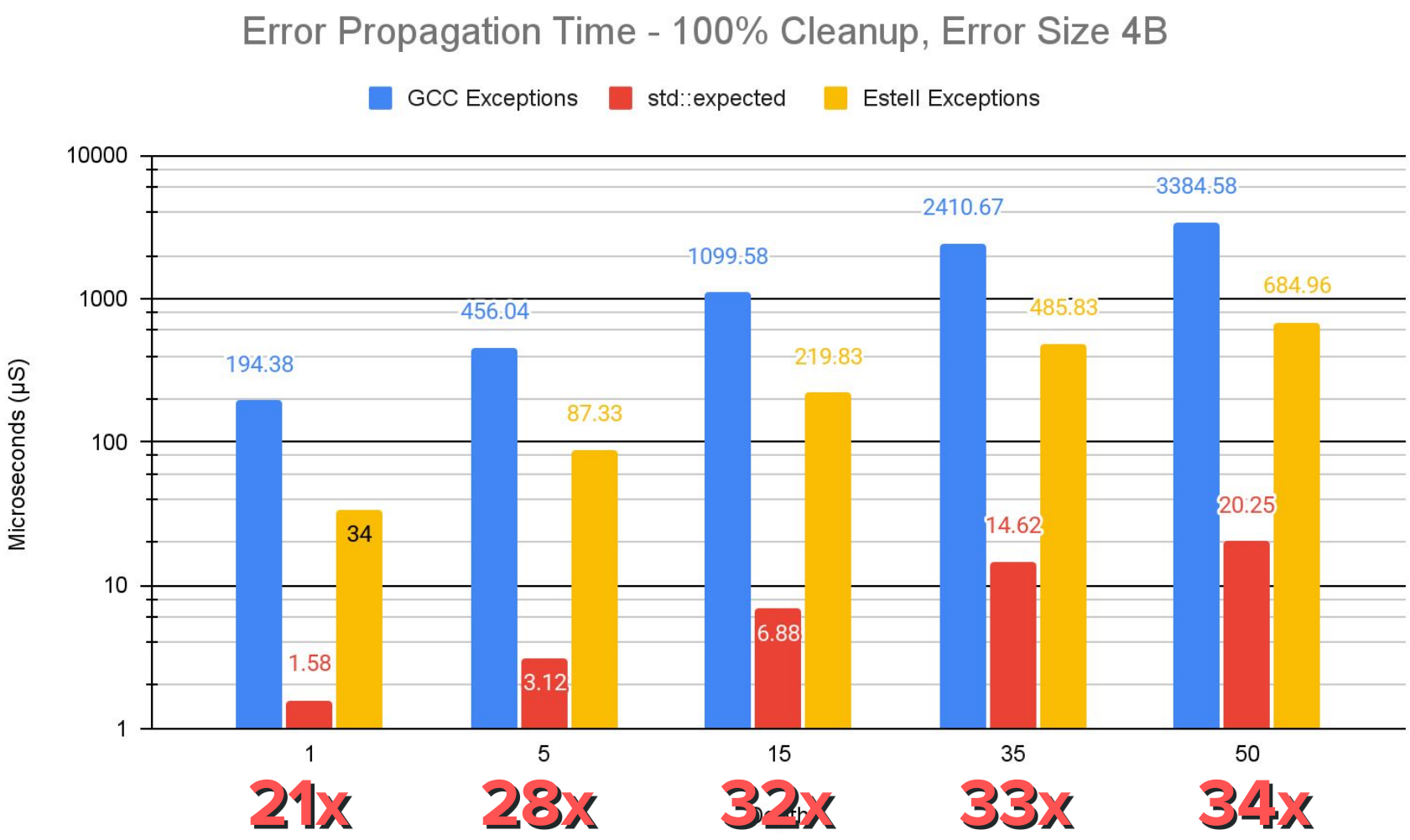
```
for (auto const& type_entry : p_exception_object.type_info) {
  if (type_entry.type_info == catch_type) {
    p_exception_object.choosen_type_offset = type_entry.offset;
    // Prepare for context installation...
    restore_cpu_core(cpu);
  }
}
```

RESULTS!!


Error Propagation Time - 0% Cleanup, Error Size 4B
GCC Exceptions
std::expected
Estell Exceptions
Microseconds (µS)
10000
1000
100
10
1
170.92
1.5
26.21
335.12
2.71
51.79
777.83
5.71
116.75
1659.38
11.71
244.38
2310
16.29
340.83
1
5
15
35
50
-83.8%
-85.1%
-85.0%
-85.9%
-85.4%


Error Propagation Time - 0% Cleanup, Error Size 4B
GCC Exceptions
std::expected
Estell Exceptions
Microseconds (µS)
10000
1000
100
10
1
170.92
1.5
26.21
335.12
2.71
51.79
777.83
5.71
116.75
1659.38
11.71
244.38
2310
16.29
340.83
1
5
15
35
50
17x
19x
20x
21x
21x


Error Propagation Time - 100% Cleanup, Error Size 4B
GCC Exceptions
std::expected
Estell Exceptions
Microseconds (µS)
10000
1000
100
10
1
194.38
1.58
34
456.04
3.12
87.33
1099.58
6.88
219.83
2410.67
14.62
485.83
3384.58
20.25
684.96
1
5
15
35
50
-81.6%
-80.4%
-80.0%
-79.8%
-79.7%


Error Propagation Time - 100% Cleanup, Error Size 4B
GCC Exceptions
std::expected
Estell Exceptions
Microseconds (µS)
10000
1000
100
10
1
194.38
1.58
34
456.04
3.12
87.33
1099.58
6.88
219.83
2410.67
14.62
485.83
3384.58
20.25
684.96
1
5
15
35
50
21x
28x
32x
33x
34x

Wait... where is the

non-determinism???

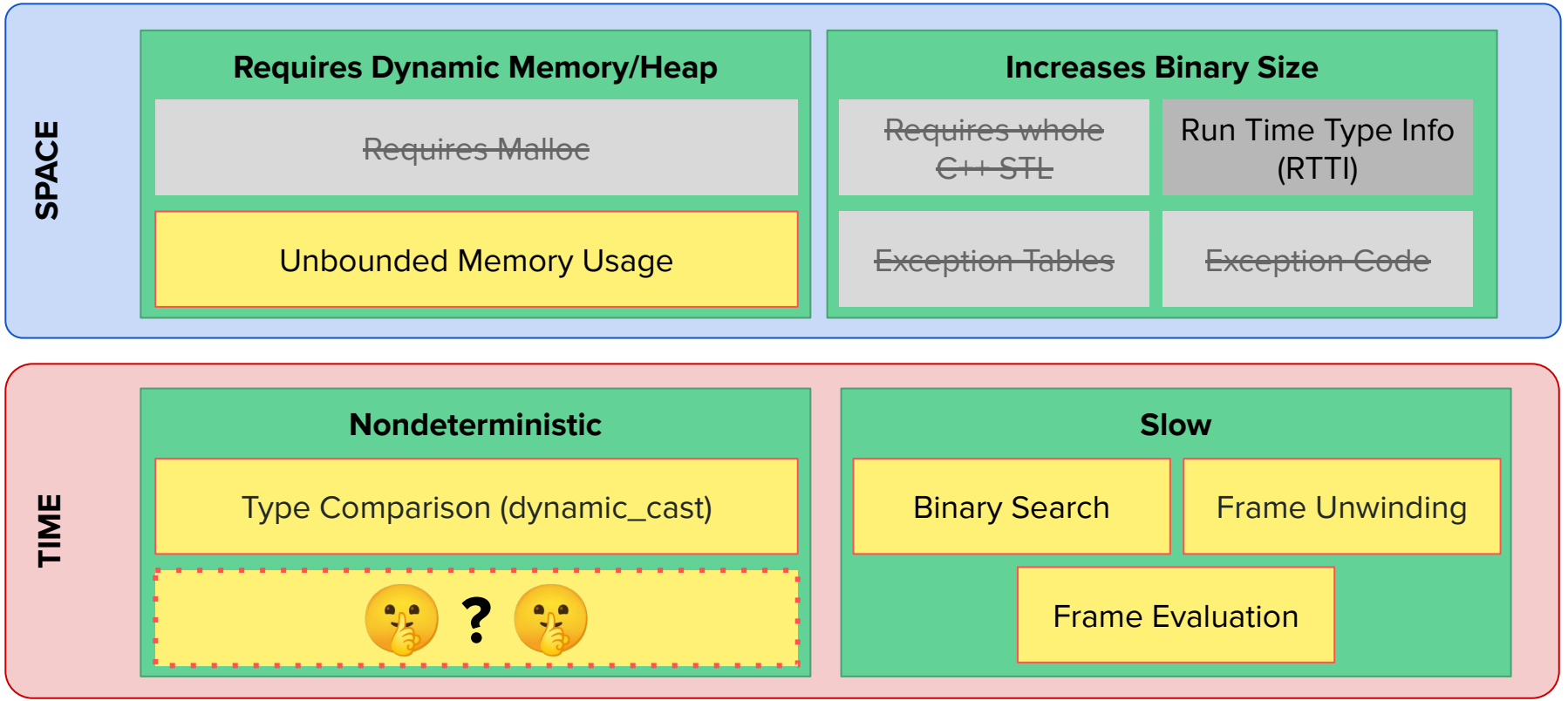
I couldn't find any

¯_(ツ)_/¯

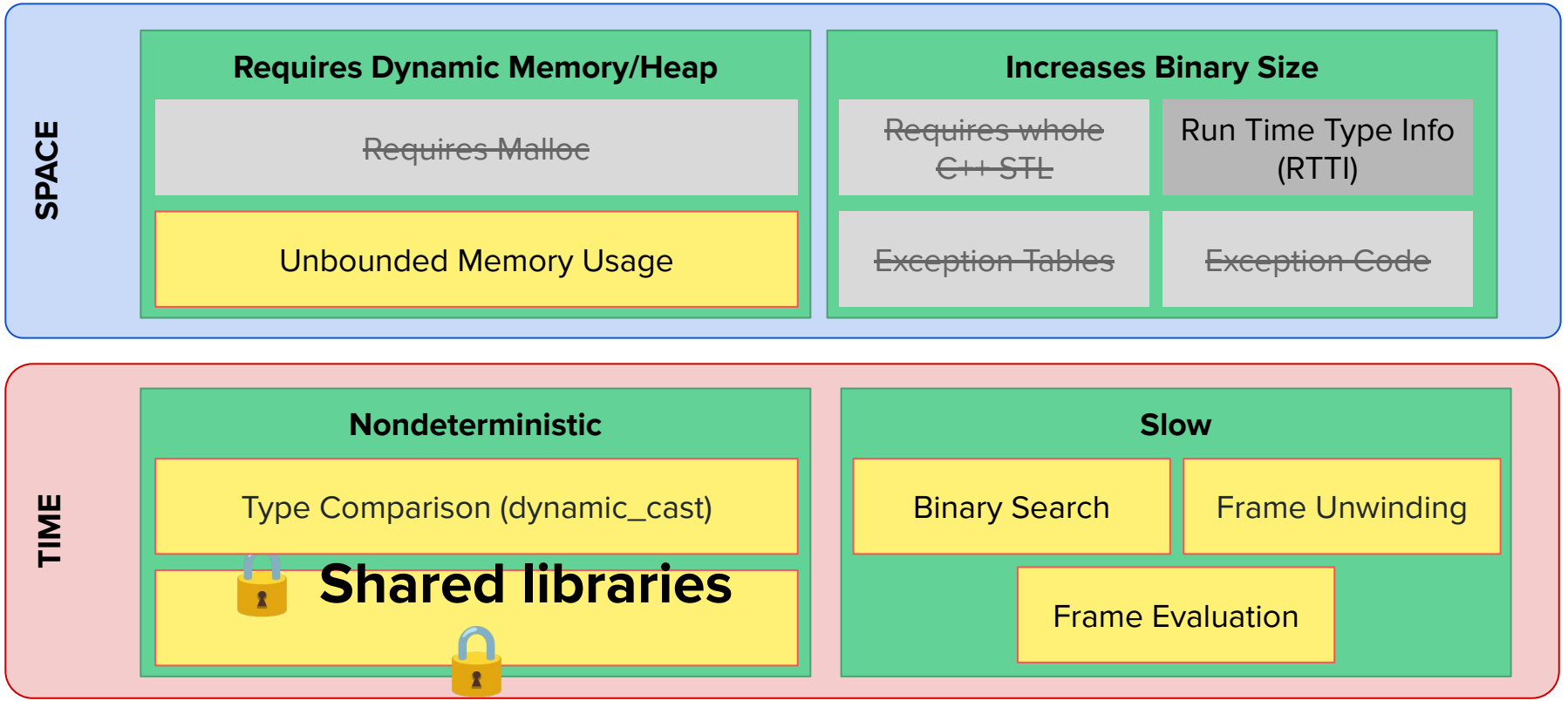
On my microcontroller...

Use shared libraries

So why do firmware developers avoid C++ exceptions?



So why do firmware developers avoid C++ exceptions?



Bug 71744 - Concurrently throwing exceptions is not scalable

Status:	RESOLVED FIXED	**Reported:**	2016-07-03 22:11 UTC by Nadav Har'El
Alias:	None	**Modified:**	2023-01-09 07:30 UTC (History)
		CC List:	5 users (show)
Product:	gcc	**See Also:**	
Component:	libgcc (show other bugs)	**Host:**	
Version:	5.3.1	**Target:**	
		Build:	
Importance:	P3 normal	**Known to work:**	
Target Milestone:	---	**Known to fail:**	
Assignee:	Florian Weimer	**Last reconfirmed:**	2018-11-19 00:00:00
URL:			
Keywords:			
Depends on:			
Blocks:			

Bug 71744 - Concurrently throwing exceptions is not scalable

Status: RESOLVED FIXED

Alias: None

Product: gcc
Component: libgcc (show other bugs)
Version: 5.3.1

Importance: P3 normal
Target Milestone: ---
Assignee: Florian Weimer

URL:
Keywords:

Depends on:
Blocks:

Florian Weimer 2023-01-09 07:30:59 UTC Comment 32

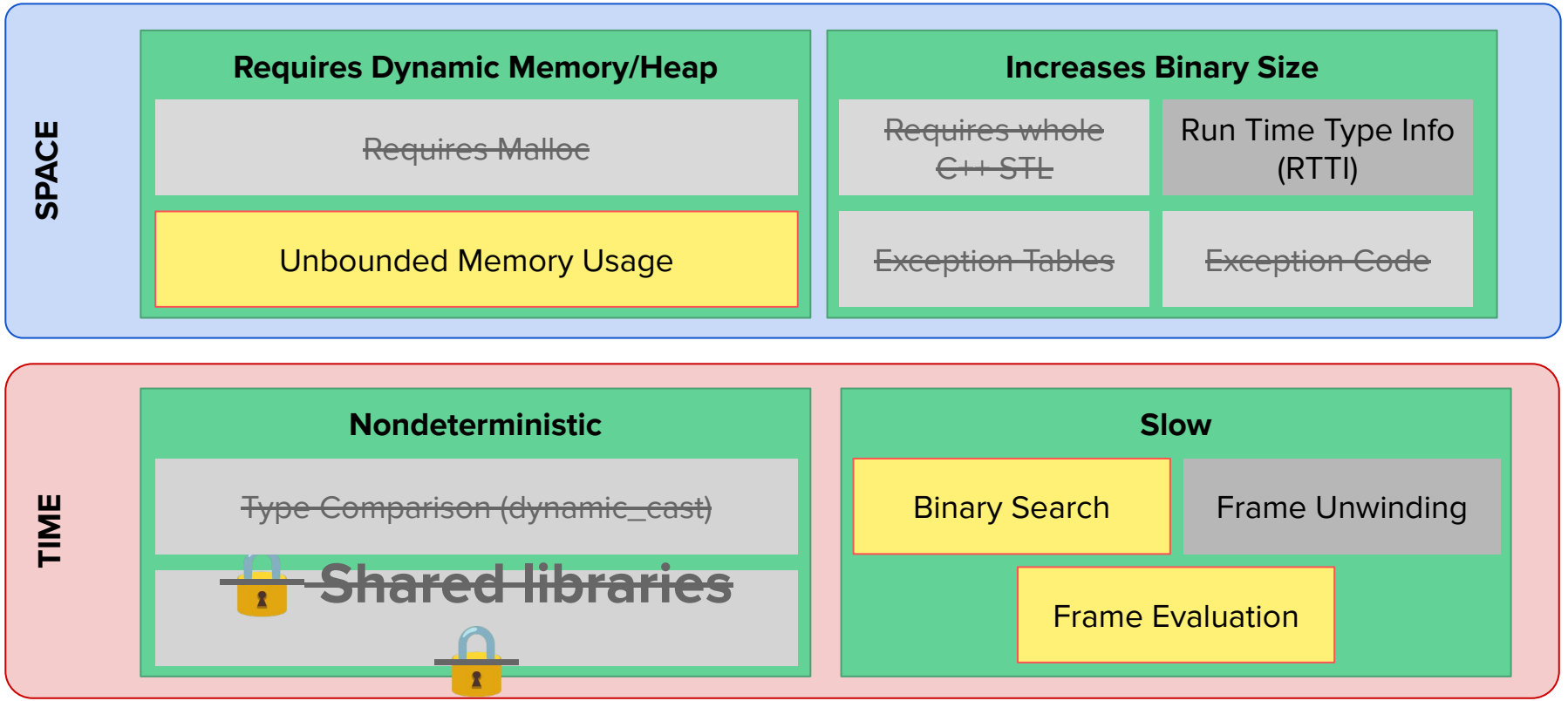
Fixed via r12-6210-g790854ea7670f1 for dynamically linked code:

```
commit 790854ea7670f11c14d431c102a49181d2915965
Author: Florian Weimer <fweimer@redhat.com>
Date:   Tue Jan 4 15:47:30 2022 +0100

    libgcc: Use _dl_find_object in _Unwind_Find_FDE

    libgcc/ChangeLog:

            * unwind-dw2-fde-dip.c (_Unwind_Find_FDE): Call _dl_find_object
            if available.
```

And r13-2706-g6e80a1d164d1f9 for the run-time code registration interface:

```
commit 6e80a1d164d1f996ad08a512c000025a7c2ca893
Author: Thomas Neumann <tneumann@users.sourceforge.net>
Date:   Tue Mar 1 21:57:35 2022 +0100

    eliminate mutex in fast path of __register_frame

    The __register_frame/__deregister_frame functions are used to register
    unwinding frames from JITed code in a sorted list. That list itself
    is protected by object_mutex, which leads to terrible performance
    in multi-threaded code and is somewhat expensive even if single-threaded.
    There was already a fast-path that avoided taking the mutex if no
    frame was registered at all.

    This commit eliminates both the mutex and the sorted list from
    the atomic fast path, and replaces it with a btree that uses
    optimistic lock coupling during lookup. This allows for fully parallel
    unwinding and is essential to scale exception handling to large
    core counts.

    libgcc/ChangeLog:

            * unwind-dw2-fde.c (release_registered_frames): Cleanup at shutdown.
            (__register_frame_info_table_bases): Use btree in atomic fast path.
            (__deregister_frame_info_bases): Likewise.
            (_Unwind_Find_FDE): Likewise.
            (base_from_object): Make parameter const.
            (classify_object_over_fdes): Add query-only mode.
            (get_pc_range): Compute PC range for lookup.
            * unwind-dw2-fde.h (last_fde): Make parameter const.
            * unwind-dw2-btree.h: New file.
```

So why do firmware developers avoid C++ exceptions?



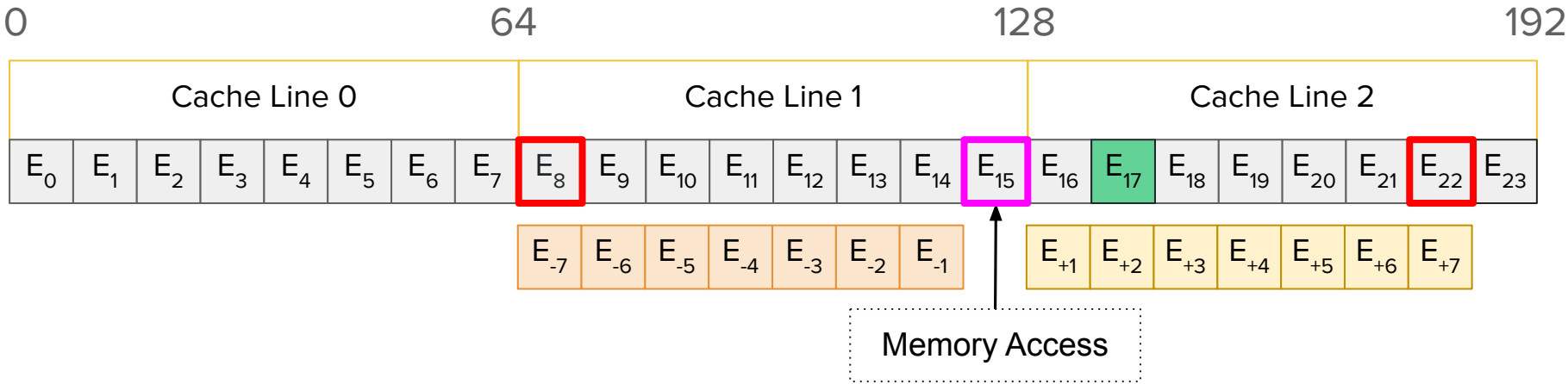
Chapter 4. 🎯

Making Nearpoint Search Work

Nearpoint Search Algorithm Requirements

- ✅ Order of magnitude faster than binary search
- ✅ Try and preserve exception space saving
- ✅ Integer math ONLY
- ✅ Max guess error < 8 ✨ NEW ✨

Why limit error to less than 8? (Max 2 cache misses)



64 bytes can hold
8 exception entries

🔀 Entry Coalescence 🔀

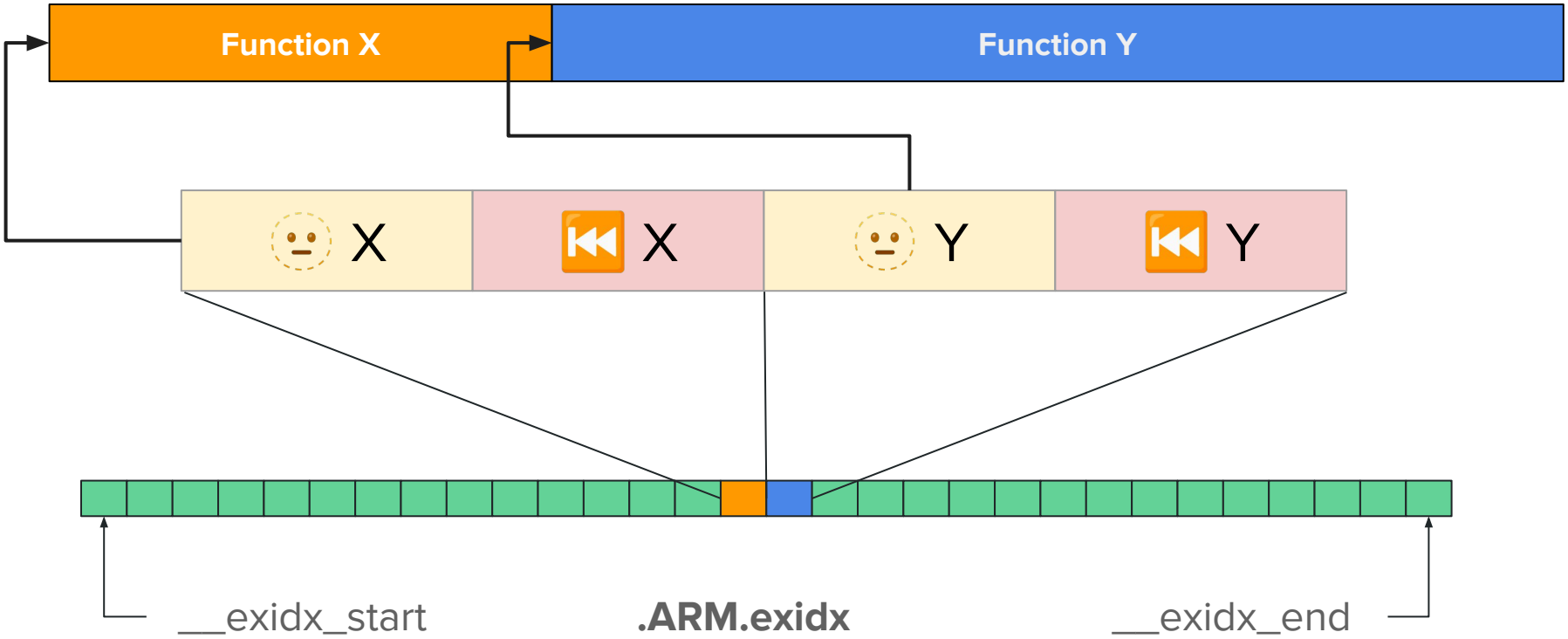
P3313R0
Impacts of noexept on ARM table based exception metadata

Draft Technical Report, 2024-05-22

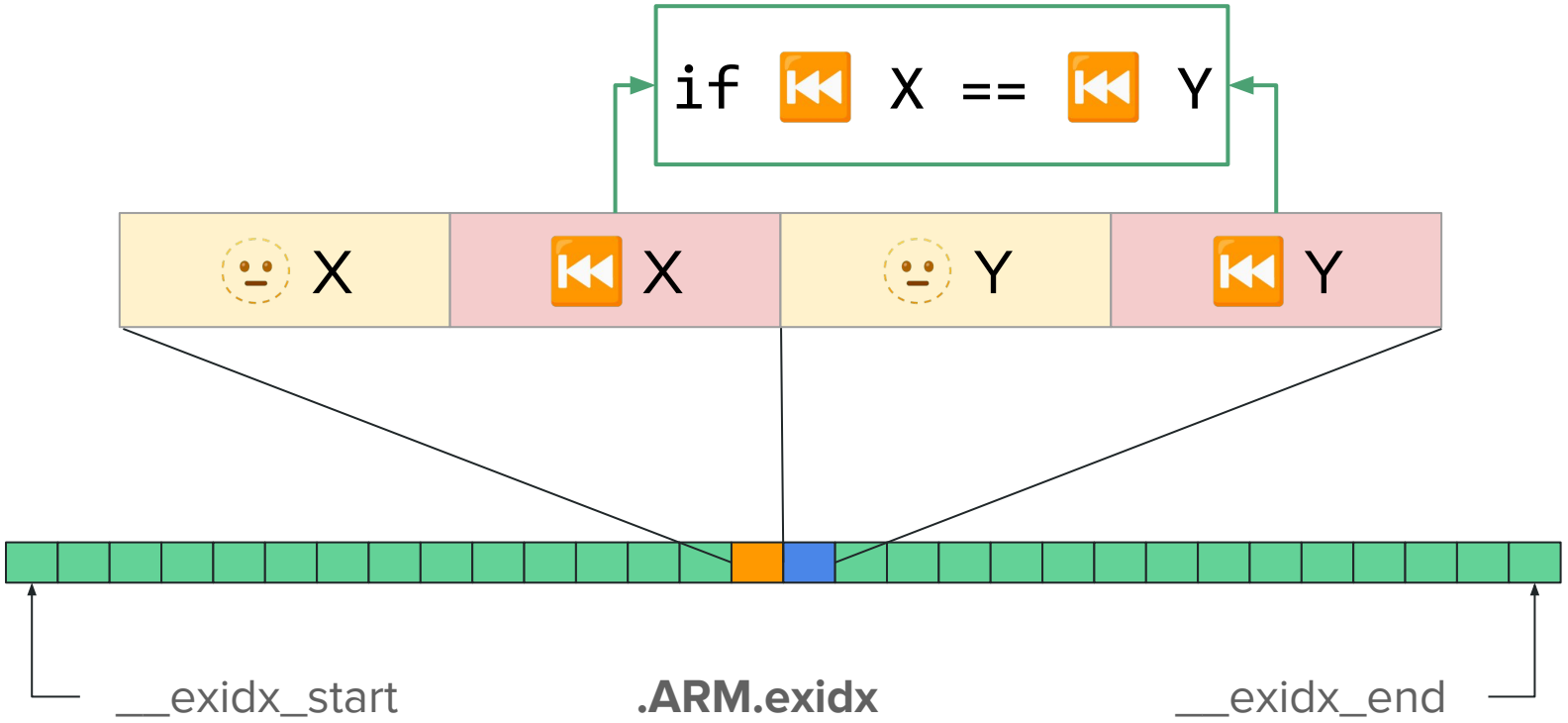
This version:
https://wg21.link/P3313R0

Author:
Khalil Estell

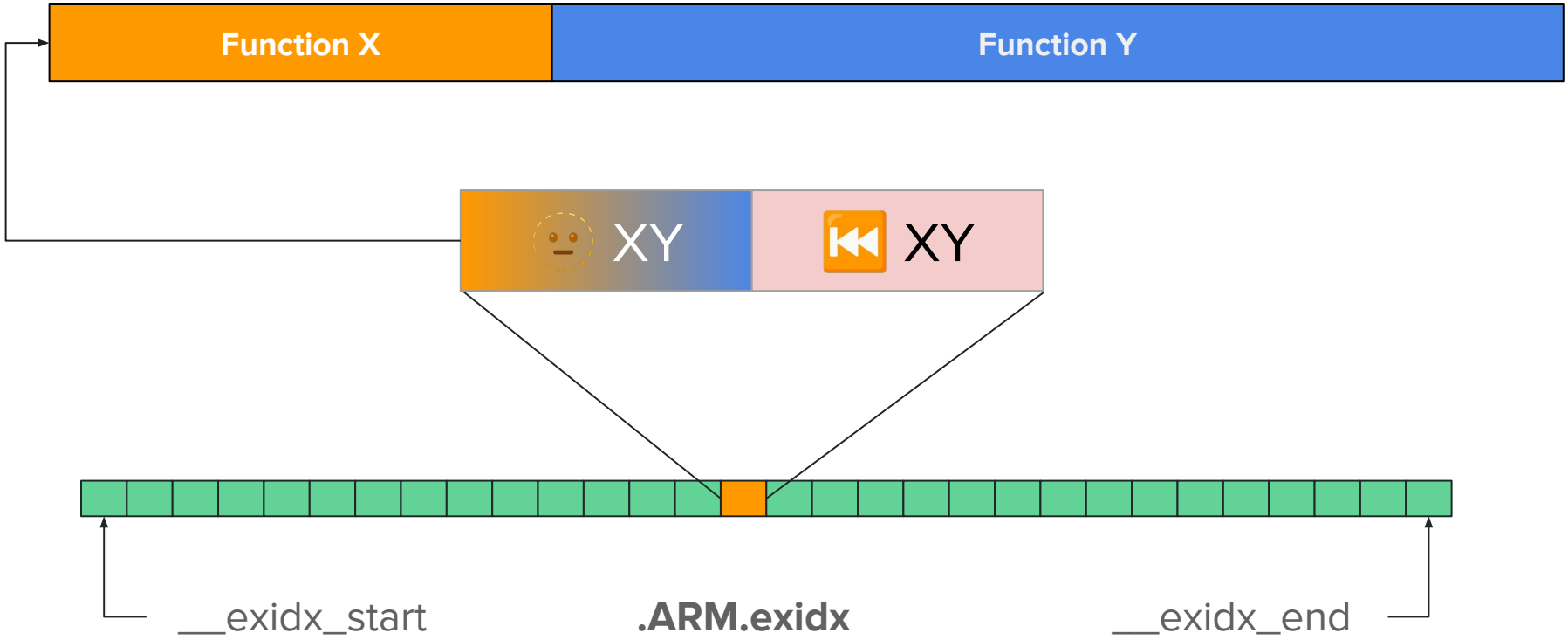
GCC's 🔀 Entry Coalescence



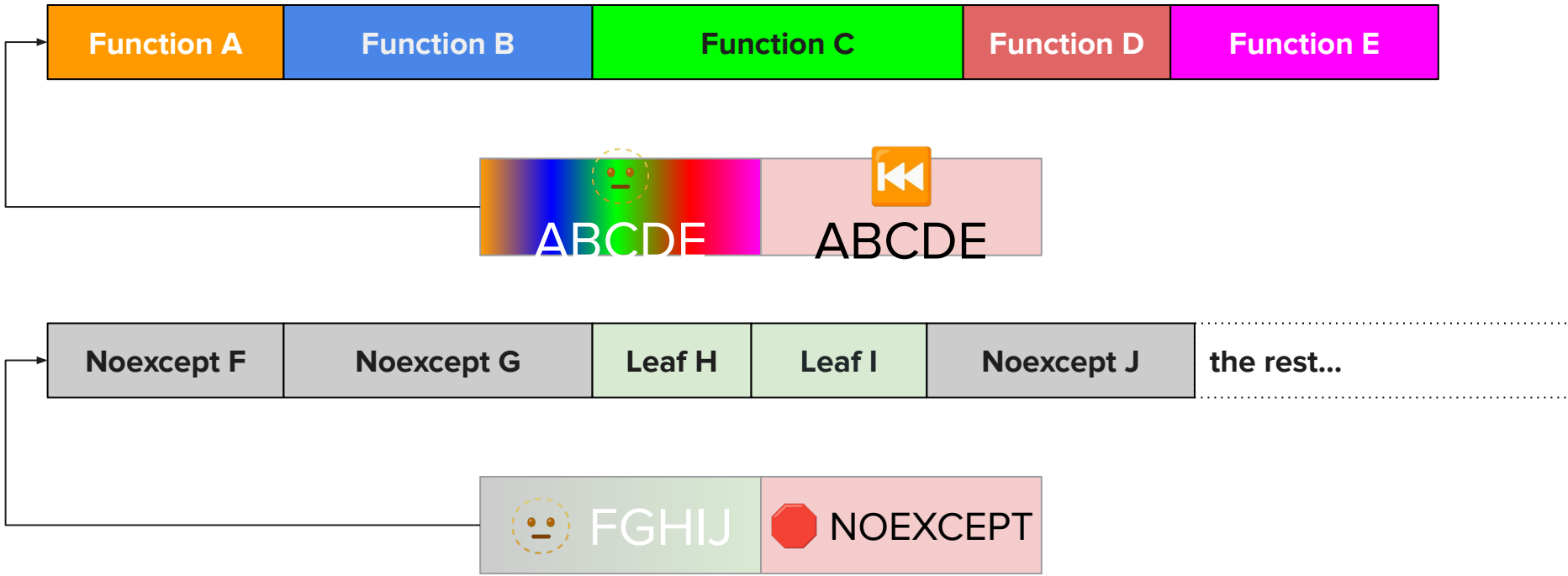
GCC's 🔀 Entry Coalescence



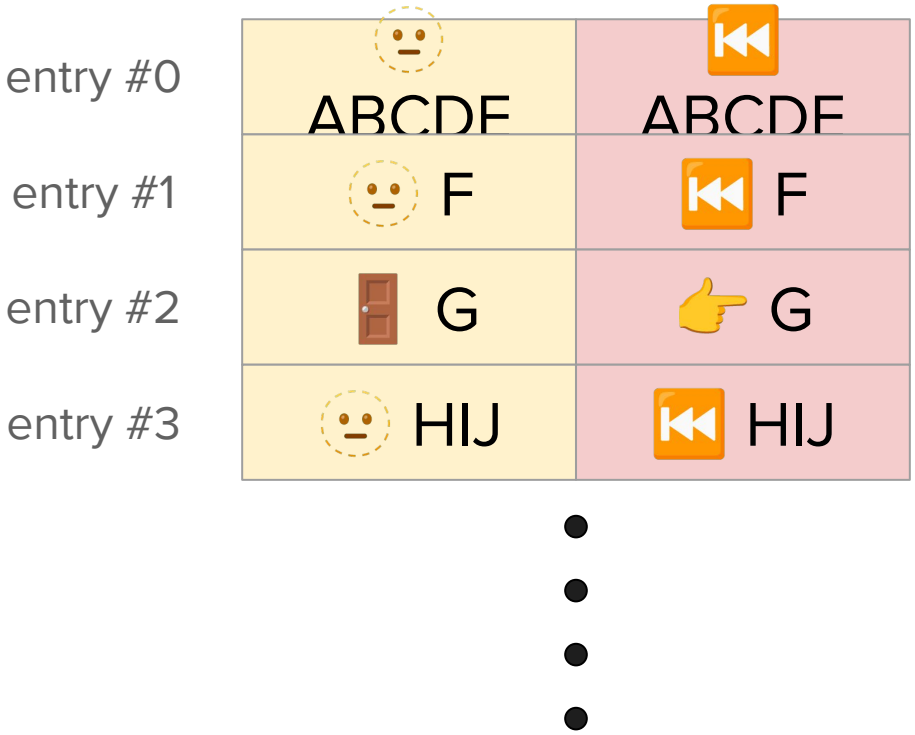
GCC's 🔀 Entry Coalescence



Function Grouping

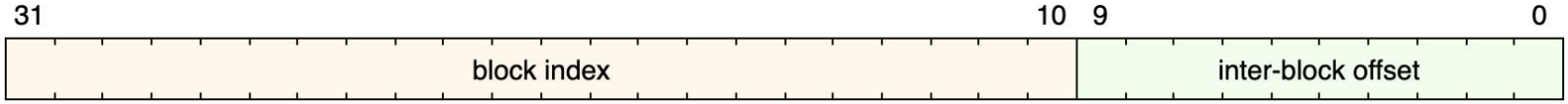


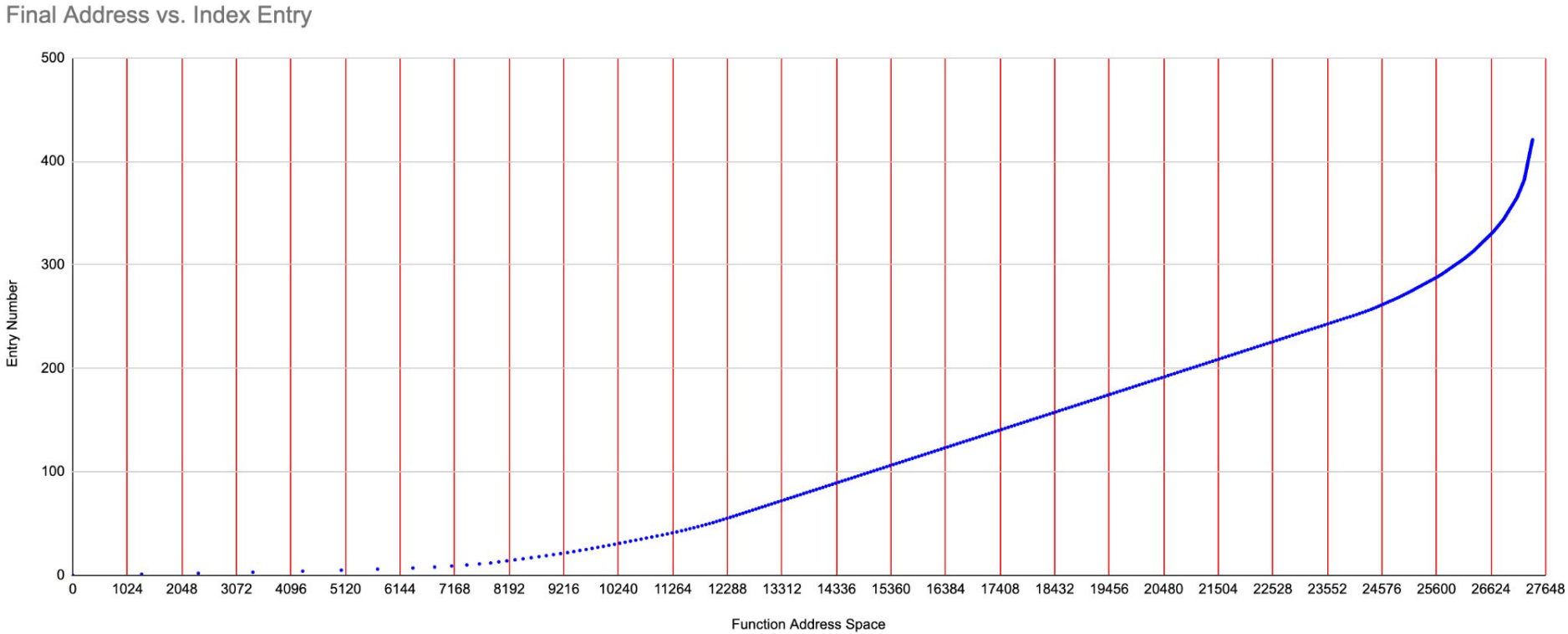
Nearpoint Function Group Sort by Size 🧲

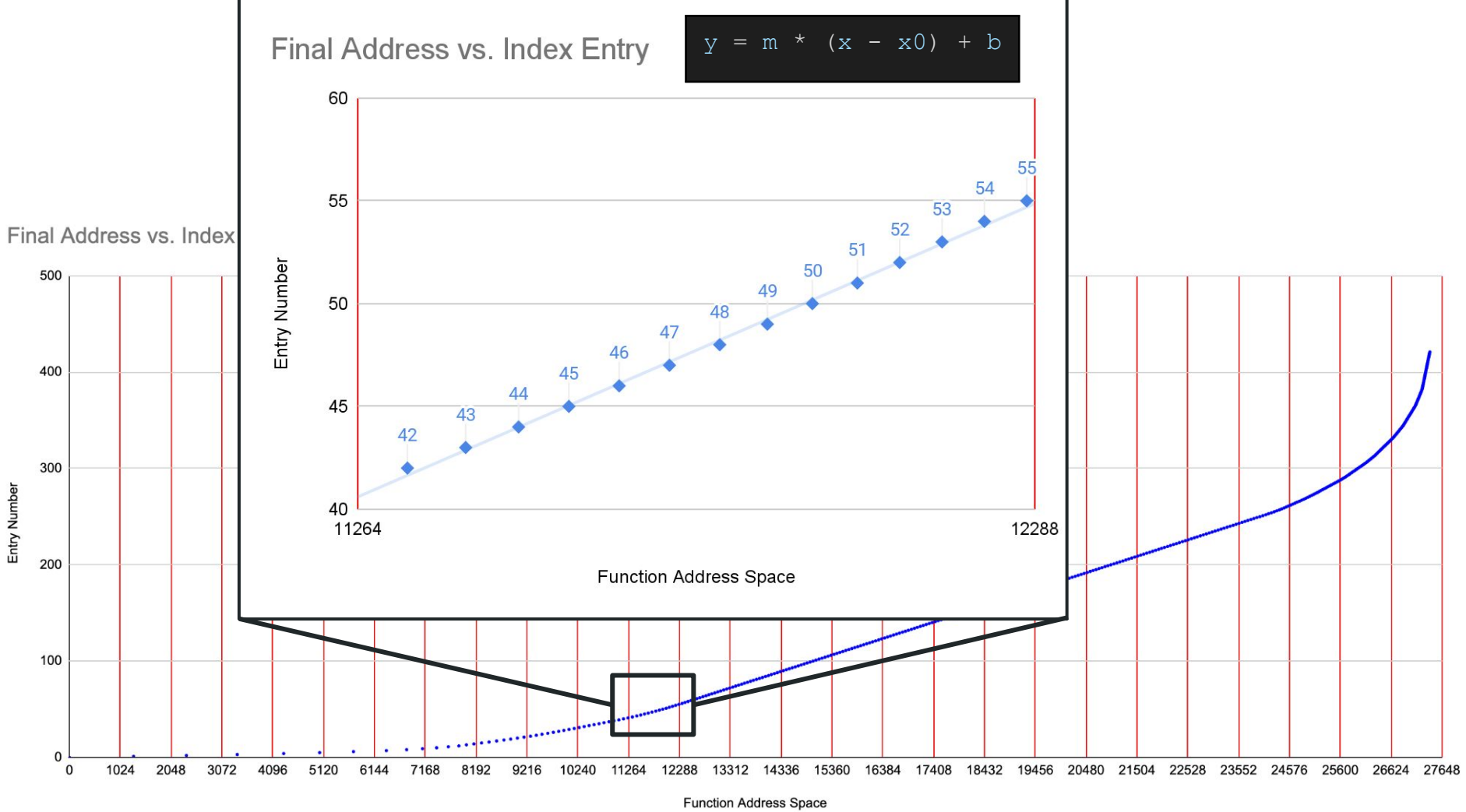


Block size must be a power of 2 (1024 in this example)

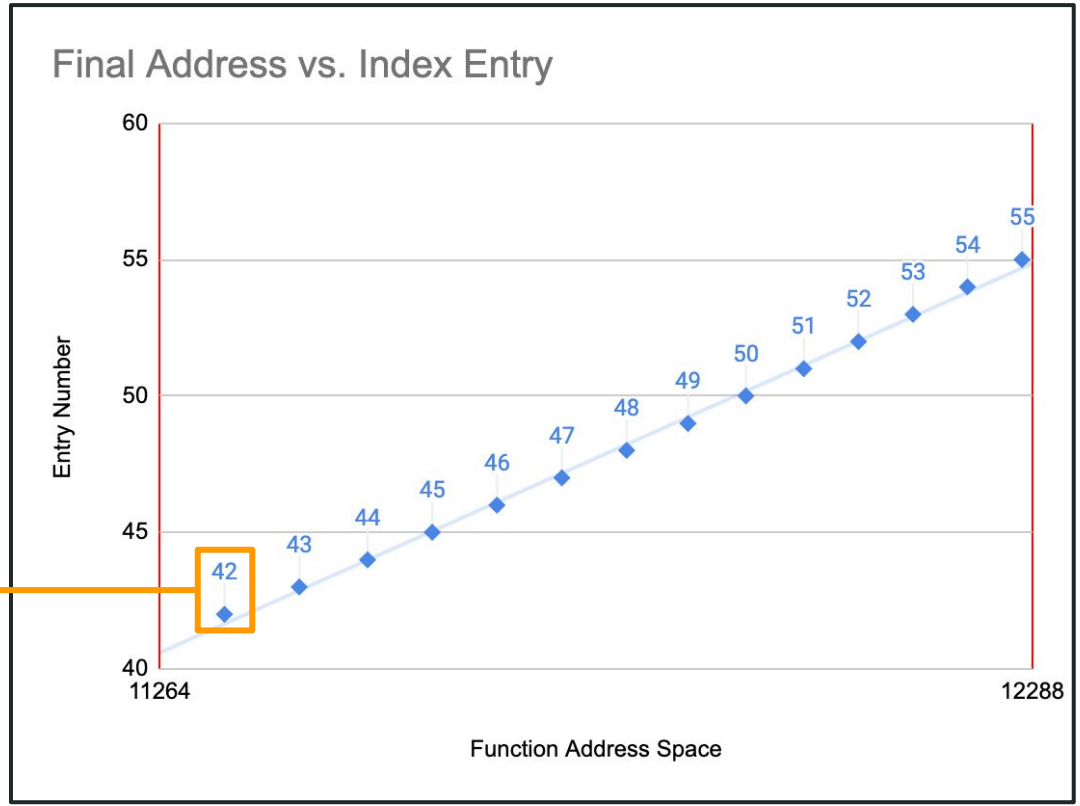
PC



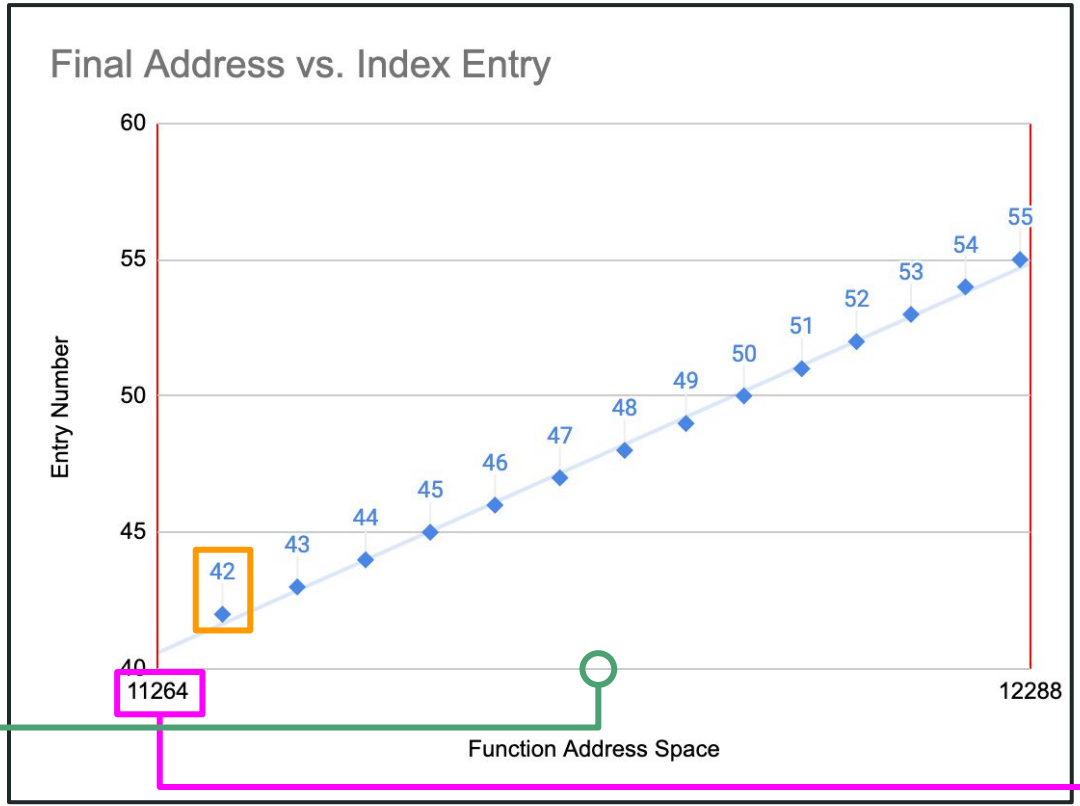



Final Address vs. Index Entry
y = m * (x - x0) + b
Entry Number
Function Address Space
60
55
50
45
40
11264
12288
42
43
44
45
46
47
48
49
50
51
52
53
54
55
Final Address vs. Index
Entry Number
Function Address Space
500
400
300
200
100
0
0
1024
2048
3072
4096
5120
6144
7168
8192
9216
10240
11264
12288
13312
14336
15360
16384
17408
18432
19456
20480
21504
22528
23552
24576
25600
26624
27648

```
y = m * (x - x0) + b
```


Final Address vs. Index Entry
60
55
50
45
40
Entry Number
42
43
44
45
46
47
48
49
50
51
52
53
54
55
11264
12288
Function Address Space

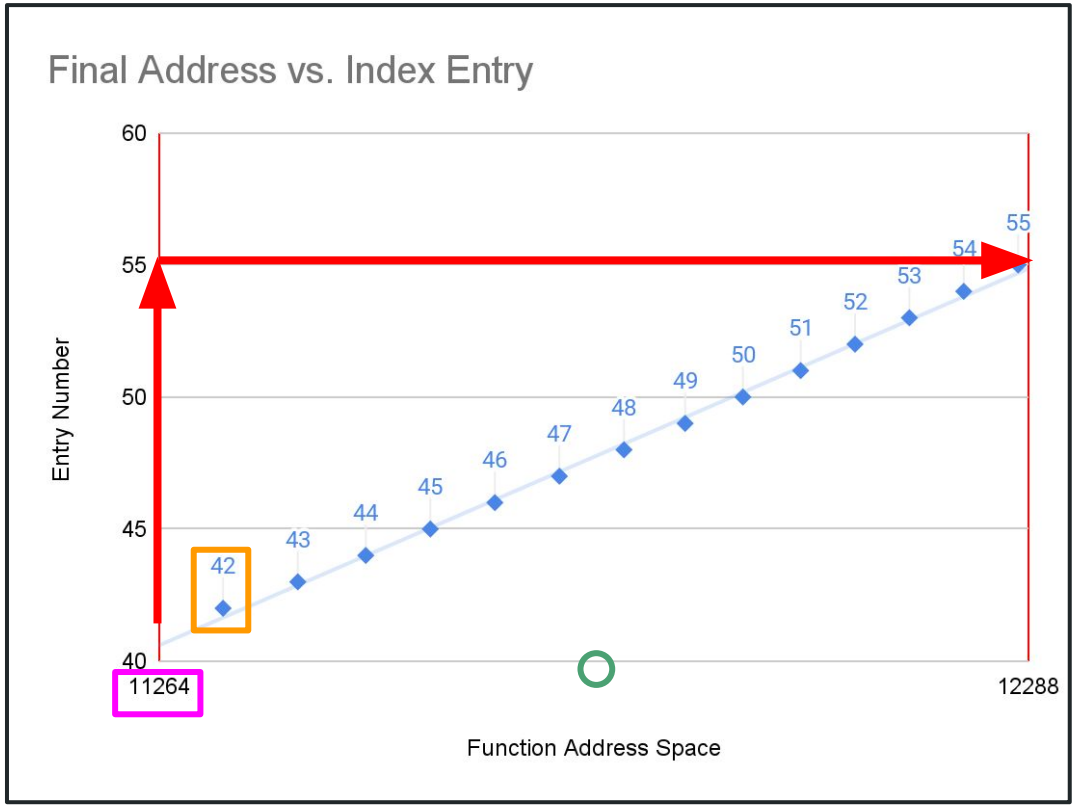
```
y = m * (x - x0) + b
```

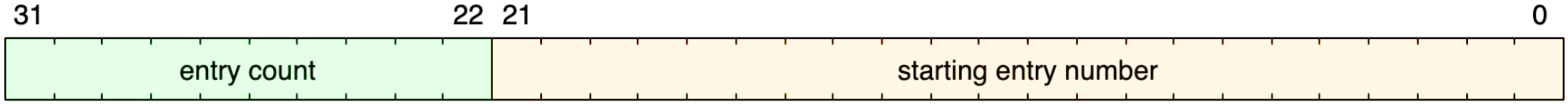

Final Address vs. Index Entry
60
55
50
45
40
Entry Number
42
43
44
45
46
47
48
49
50
51
52
53
54
55
11264
12288
Function Address Space

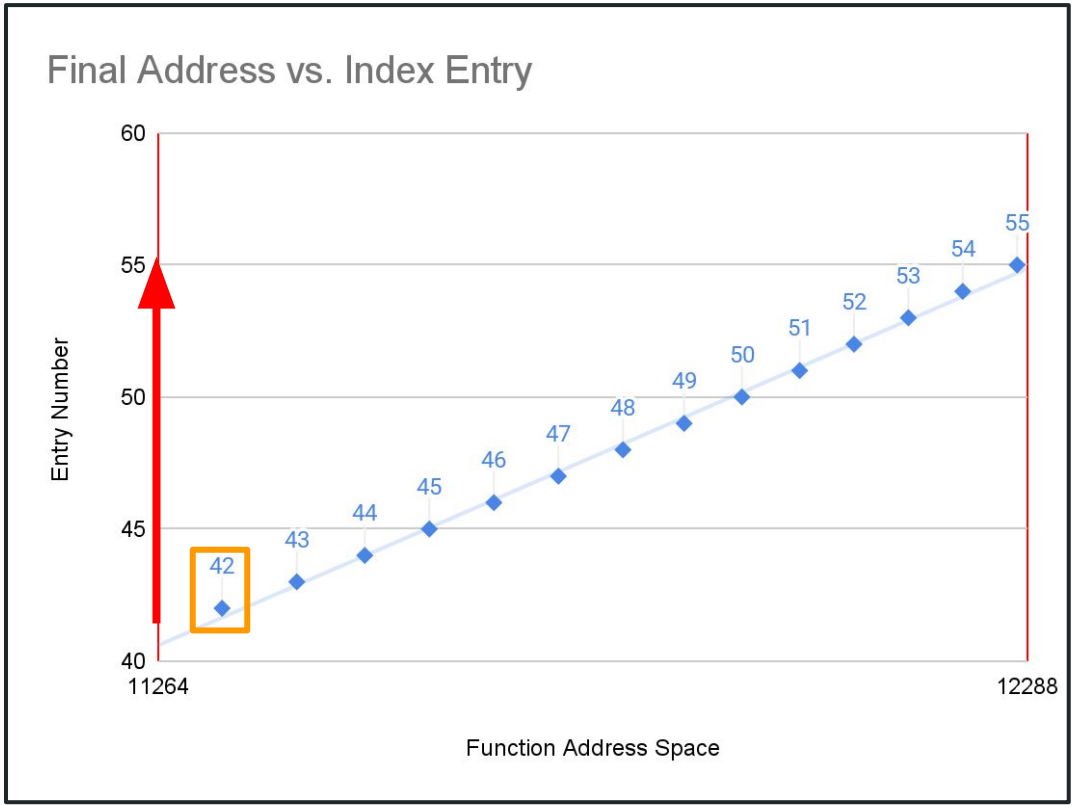
$$y = m * (x - x0) + b$$

Rise
13

Run
1024
(same for all blocks)




31
22
21
0
entry count
starting entry number


Final Address vs. Index Entry
60
55
50
45
40
Entry Number
42
43
44
45
46
47
48
49
50
51
52
53
54
55
11264
12288
Function Address Space

```
y = m * (x - x0) + b
```

```
struct nearpoint_descriptor
{
 std::uint32_t block_power = 0;
 std::uint32_t text_address = 0;
};
```

```
std::array<std::uint32_t, 2> const
_near_point_descriptor_data  = {
 0x0000000b,
 0x08000044,
};
```

```
std::array<std::uint32_t, 18> const _normal_table_data = {
 (0 << 11) | 1, // Block(start=0, count=1)
 (0 << 11) | 1, // Block(start=0, count=1)
 (0 << 11) | 1, // Block(start=0, count=1)
 (0 << 11) | 1, // Block(start=0, count=1)
 (0 << 11) | 1, // Block(start=0, count=1)
 (0 << 11) | 1, // Block(start=0, count=1)
 (1 << 11) | 1, // Block(start=1, count=1)
 (1 << 11) | 1, // Block(start=1, count=1)
 (2 << 11) | 2, // Block(start=2, count=2)
 (3 << 11) | 3, // Block(start=3, count=3)
 (5 << 11) | 3, // Block(start=5, count=3)
 (7 << 11) | 6, // Block(start=7, count=6)
 (12 << 11) | 14, // Block(start=12, count=14)
 (25 << 11) | 20, // Block(start=25, count=20)
 (44 << 11) | 27, // Block(start=44, count=27)
 (70 << 11) | 28, // Block(start=70, count=28)
 (97 << 11) | 32, // Block(start=97, count=32)
 (128 << 11) | 26, // Block(start=128, count=26)
};
```

```
std::uintptr_t near_point_guess_index(std::uintptr_t p_program_counter)
{
 auto const block_power = __except_abi::near_point_descriptor.block_power;
 auto const program_offset = __except_abi::near_point_descriptor.text_address;
 auto const inter_block_mask = (1U << block_power) - 1U;

 auto const pc = p_program_counter - program_offset;
 auto const inter_block_location = pc & inter_block_mask;
 auto const block_index = pc >> block_power;
 auto const block_info = __except_abi::normal_table[block_index];

 auto const entry_start = block_info >> block_power;
 auto const entry_count = block_info & inter_block_mask;

 auto const guess_offset = (inter_block_location * entry_count) >> block_power;
 auto const guess_location = entry_start + guess_offset
 return guess_location;
}
```

Pull info from descriptor

Break apart **PC**

Lookup **block_info**

Break up **block_info**

Calculate linear equation

```
index_entry_t const& get_index_entry_near_point (std::uint32_t p_program_counter)
{
 auto const index_table = get_arm_exception_index ();
 auto const initial_guess = near_point_guess_index (p_program_counter);
 auto it = index_table.begin() + initial_guess;

 if (p_program_counter < it->function()) {
   do {
     --it;
   } while (it->function() > p_program_counter);
 } else {
   do {
     ++it;
   } while (it->function() <= p_program_counter);
   --it;
 }

 return *it;
}
```

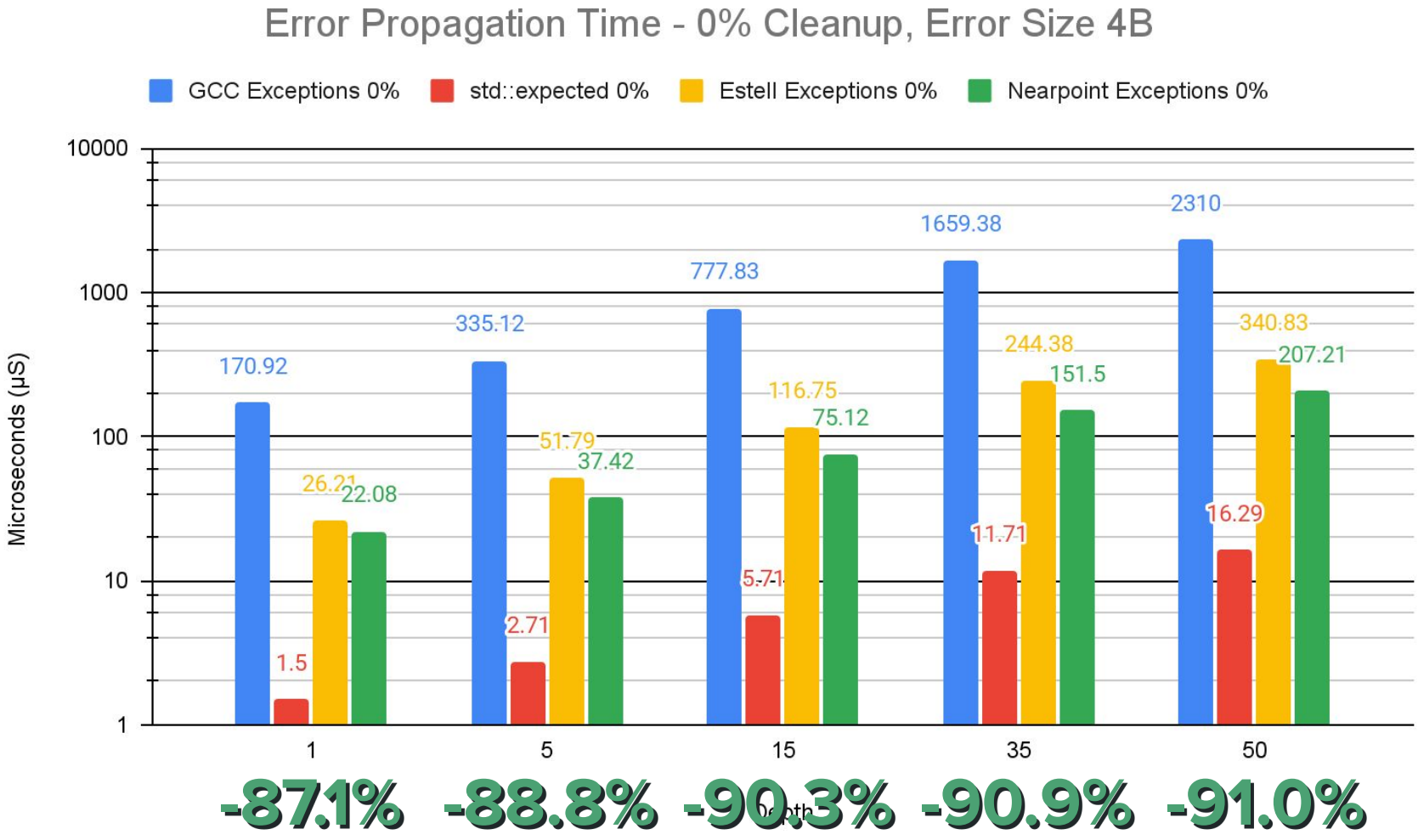
Guess

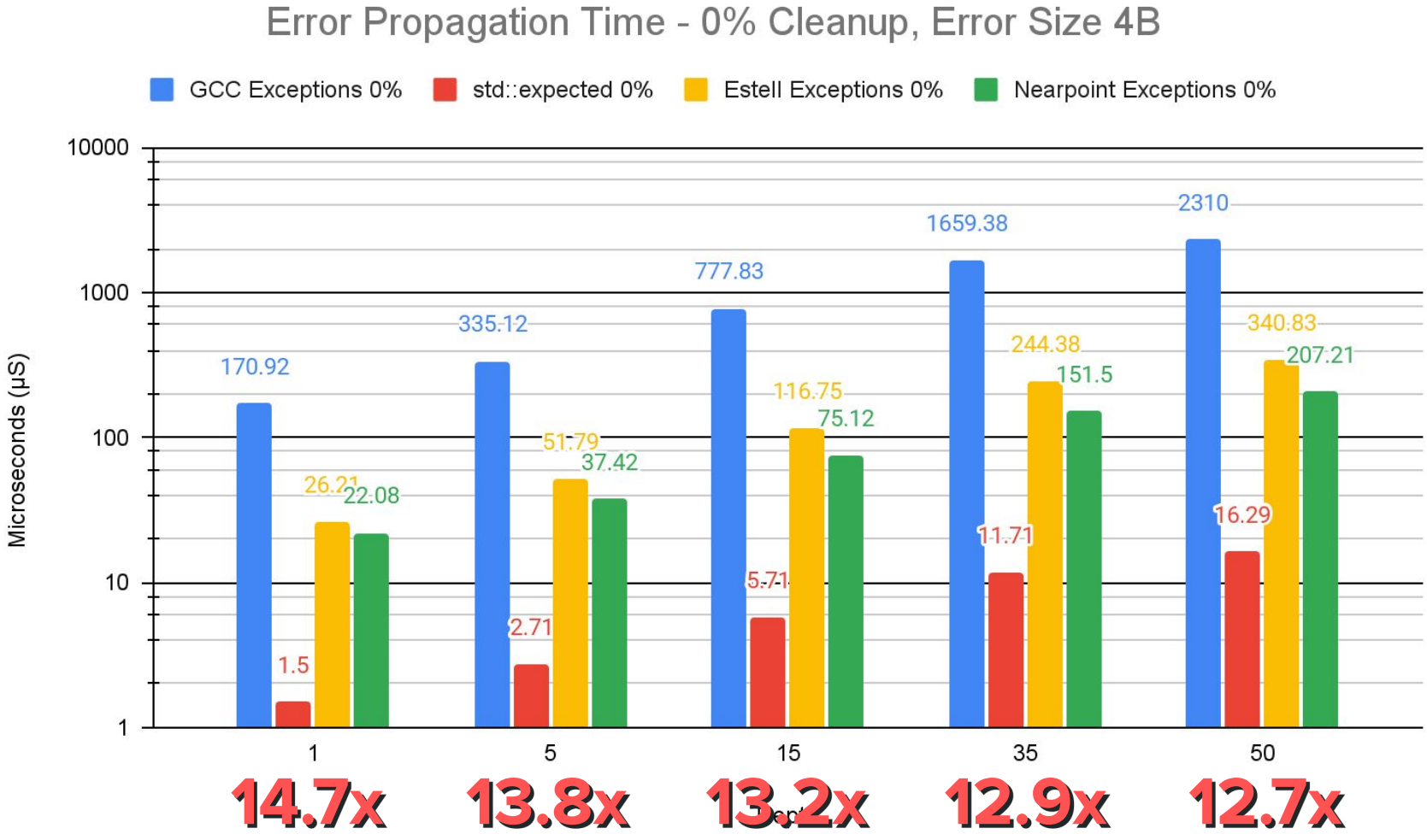
Scan Left

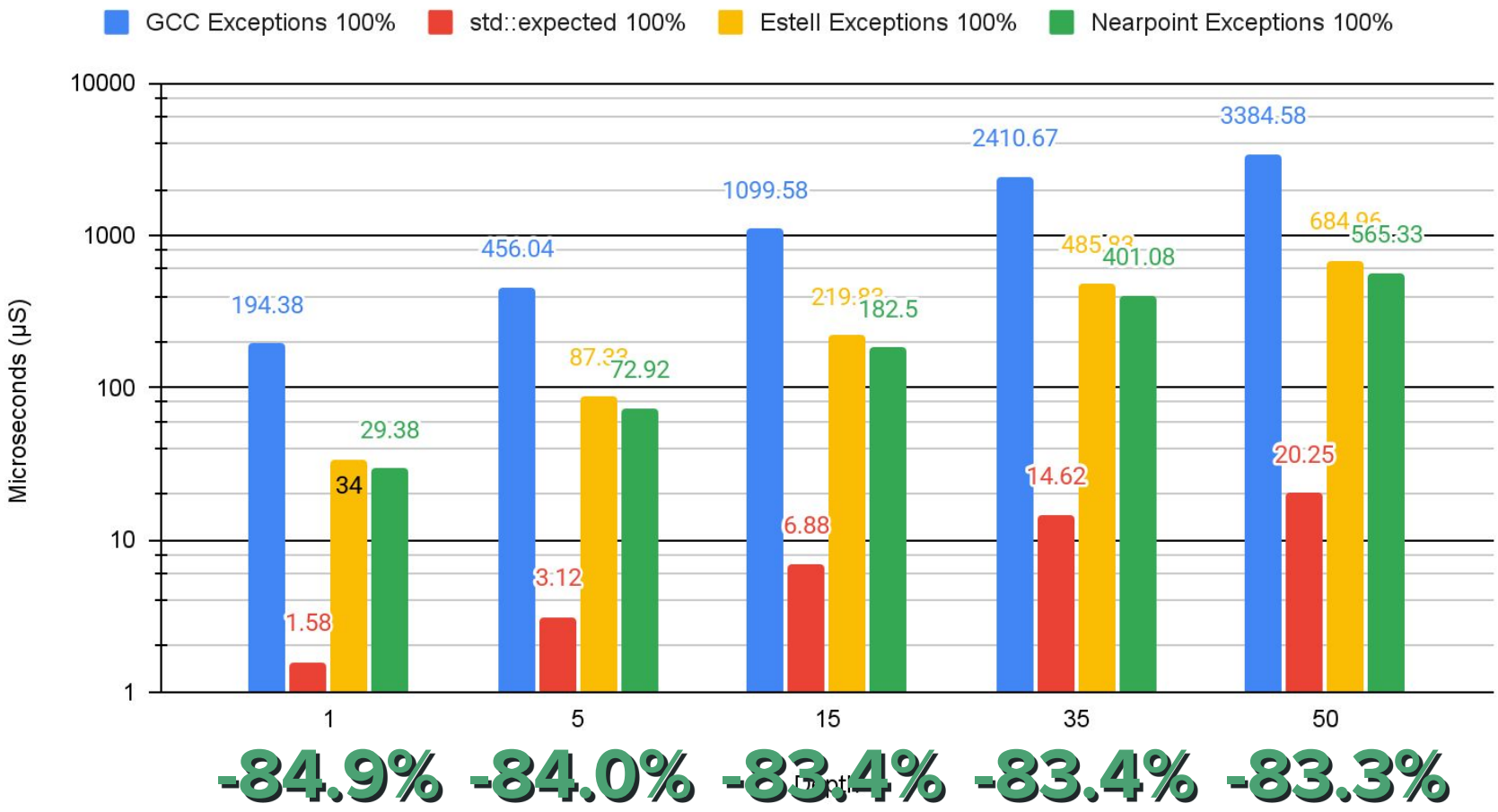
Scan Right

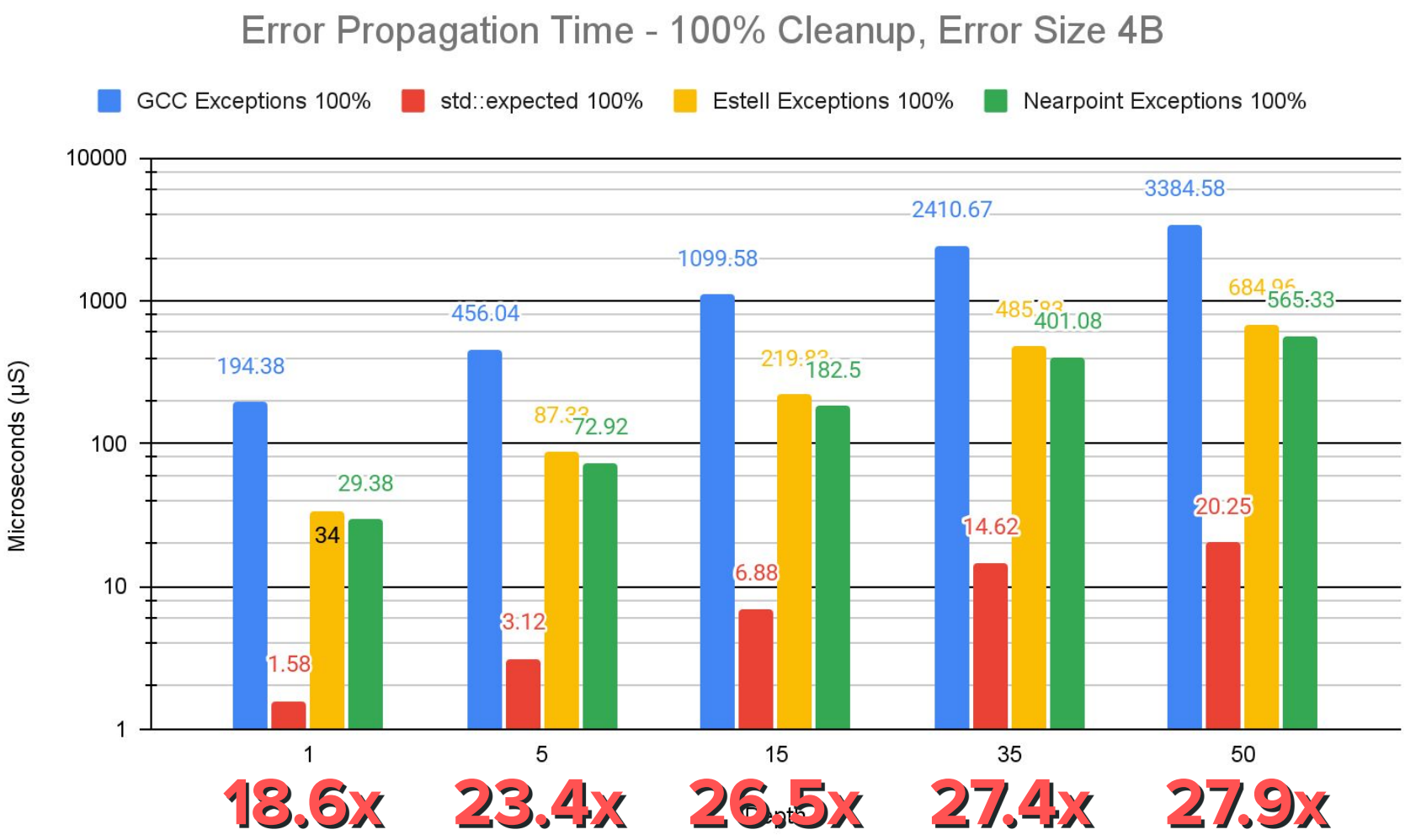
RESULTS!!

>90% @ >=11 frames


Error Propagation Time - 0% Cleanup, Error Size 4B
GCC Exceptions 0%
std::expected 0%
Estell Exceptions 0%
Nearpoint Exceptions 0%
Microseconds (µS)
10000
1000
100
10
1
170.92
1.5
26.21
22.08
335.12
2.71
51.79
37.42
777.83
5.71
116.75
75.12
1659.38
11.71
244.38
151.5
2310
16.29
340.83
207.21
1
5
15
35
50
-87.1%
-88.8%
-90.3%
-90.9%
-91.0%


Error Propagation Time - 0% Cleanup, Error Size 4B
GCC Exceptions 0%
std::expected 0%
Estell Exceptions 0%
Nearpoint Exceptions 0%
Microseconds (µS)
10000
1000
100
10
1
170.92
1.5
26.21
22.08
335.12
2.71
51.79
37.42
777.83
5.71
116.75
75.12
1659.38
11.71
244.38
151.5
2310
16.29
340.83
207.21
1
5
15
35
50
14.7x
13.8x
13.2x
12.9x
12.7x


Error Propagation Time - 100% Cleanup, Error Size 4B
GCC Exceptions 100%
std::expected 100%
Estell Exceptions 100%
Nearpoint Exceptions 100%
Microseconds (µS)
10000
1000
100
10
1
194.38
1.58
34
29.38
456.04
3.12
87.33
72.92
1099.58
6.88
219.92
182.5
2410.67
14.62
485.83
401.08
3384.58
20.25
684.96
565.33
1
5
15
35
50
-84.9%
-84.0%
-83.4%
-83.4%
-83.3%


Error Propagation Time - 100% Cleanup, Error Size 4B
GCC Exceptions 100%
std::expected 100%
Estell Exceptions 100%
Nearpoint Exceptions 100%
Microseconds (µS)
10000
1000
100
10
1
194.38
1.58
34
29.38
456.04
3.12
87.33
72.92
1099.58
6.88
219.92
182.5
2410.67
14.62
485.83
401.08
3384.58
20.25
684.96
565.33
1
5
15
35
50
18.6x
23.4x
26.5x
27.4x
27.9x

Binary Size Difference

```
  text    data     bss     dec     hex filename
 47575     240    2648   50463    c51f stm32f103c8/estell/Release/except.cpp.elf
```

Binary Size Difference

```
 text    data     bss     dec     hex filename
47575     240    2648   50463    c51f stm32f103c8/estell/Release/except.cpp.elf

 text    data     bss     dec     hex filename
46987     268    2648   49791    c27f stm32f103c8/estell/Release/nearpoint.cpp.elf
```

⬇️ **-588 bytes**

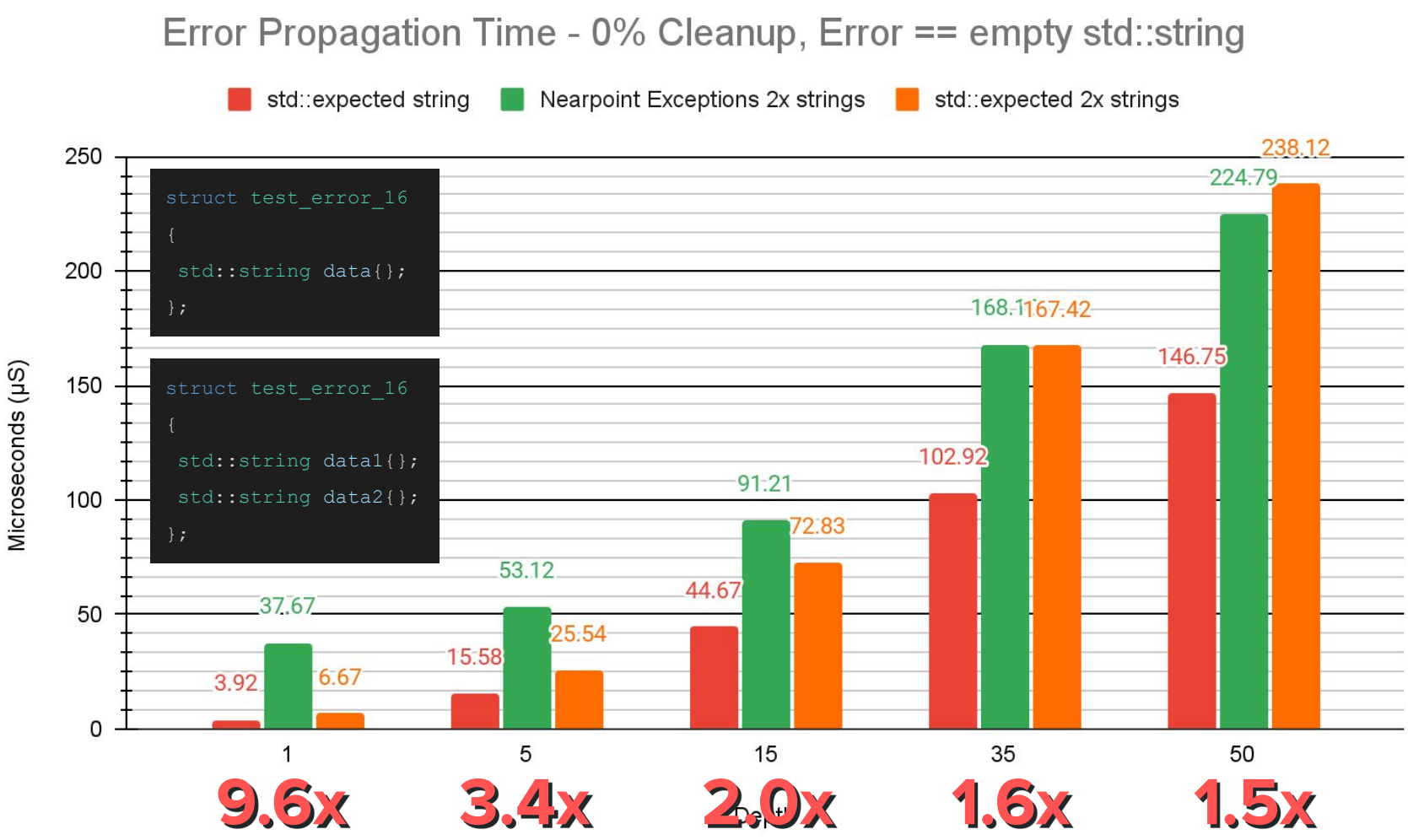
**Block power = 11*

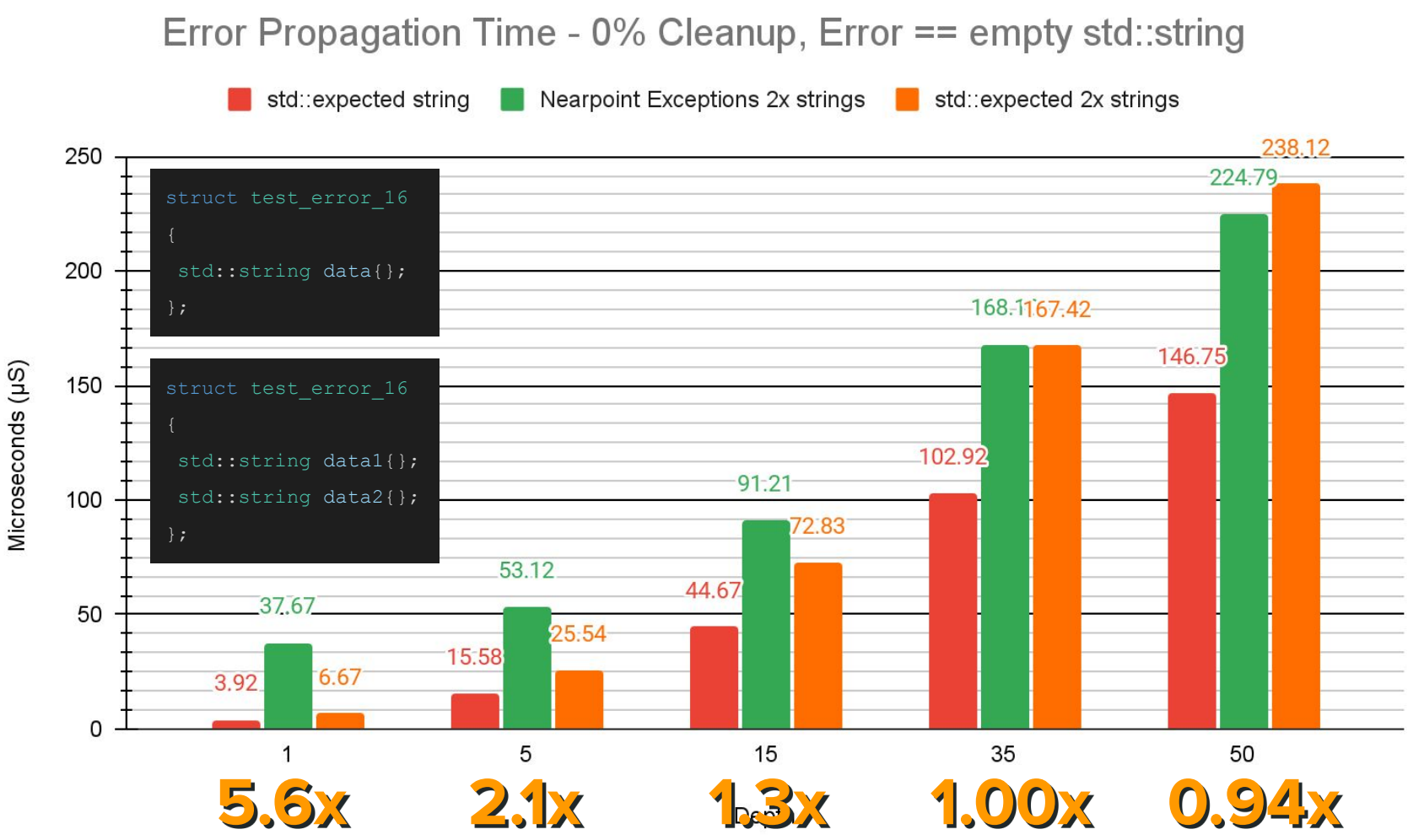
🤔 Let's compare to an `std::string{}; // empty`

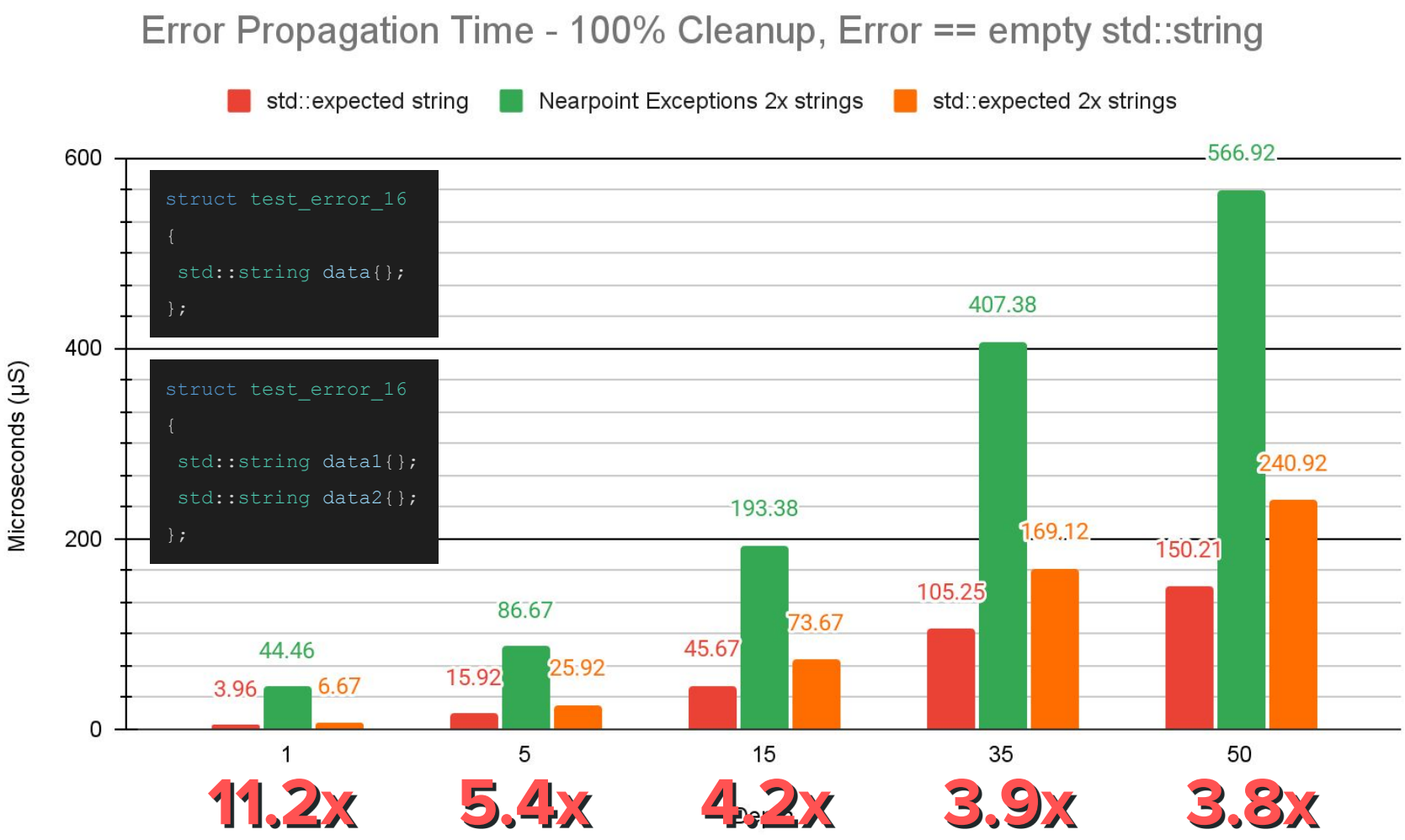
- Default initialized to an empty string
  - No dynamic memory allocation
- Has a nontrivial move constructor
- A stand-in for a **complex error object**

```
struct test_error_16
{
 std::string data{};
};
```

```
struct test_error_16
{
 std::string data1{};
 std::string data2{};
};
```


Error Propagation Time - 0% Cleanup, Error == empty std::string
std::expected string
Nearpoint Exceptions 2x strings
std::expected 2x strings
Microseconds (µS)
250
200
150
100
50
0
struct test_error_16
{
std::string data{};
};
struct test_error_16
{
std::string data1{};
std::string data2{};
};
3.92
37.67
6.67
15.58
53.12
25.54
44.67
91.21
72.83
102.92
168.11
167.42
146.75
224.79
238.12
1
5
15
35
50
9.6x
3.4x
2.0x
1.6x
1.5x


Error Propagation Time - 0% Cleanup, Error == empty std::string
std::expected string
Nearpoint Exceptions 2x strings
std::expected 2x strings
Microseconds (µS)
250
200
150
100
50
0
struct test_error_16
{
std::string data{};
};
struct test_error_16
{
std::string data1{};
std::string data2{};
};
3.92
37.67
6.67
15.58
53.12
25.54
44.67
91.21
72.83
102.92
168.11
167.42
146.75
224.79
238.12
1
5
15
35
50
5.6x
2.1x
1.3x
1.00x
0.94x


Error Propagation Time - 100% Cleanup, Error == empty std::string
std::expected string
Nearpoint Exceptions 2x strings
std::expected 2x strings
Microseconds (µS)
600
400
200
0
struct test_error_16
{
std::string data{};
};
struct test_error_16
{
std::string data1{};
std::string data2{};
};
3.96
44.46
6.67
15.92
86.67
25.92
45.67
193.38
73.67
105.25
407.38
169.12
150.21
566.92
240.92
1
5
15
35
50
11.2x
5.4x
4.2x
3.9x
3.8x

Why is `std::expected` impacted so badly?

Result type propagation must:

1. O(M): Determine a suitable handler
2. O(N): Unwind frame
3. **O(N):** Move/copy error up a frame

Exception propagation must:

1. O(M): Determine a suitable handler
2. O(N): Unwind frames
3. **O(1):** Allocate & construct error

RECAP: How did I reduce C++ exceptions via GCC on an ARM Cortex-M by 90%?

- Used C++ standard libraries when appropriate
- Moved as much of the code to compile time as possible
- Applied data driven design to exception index & unwind instructions
- Cached & pre-formatted data when appropriate.
- Rewrote runtime in C++
- **I thought it was possible**

Chapter 5. 

Future of Exceptions

(as Khalil sees it)

Exceptional Software Design & Exception Insights Tool

Part 3

Unique Exception Format

(De-duplicate Exception Data Table)

First step to exceptions in freestanding C++

```
void set_exception_allocator(std::pmr::memory_resource*) noexcept;
  std::pmr::memory_resource* get_exception_allocator() noexcept;
```

Async (coro) Exceptions Unwinding
Async Destructors

Vendoring this Technology

```
-Wl,--plugin=estell_exceptions.so
-Wl,--plugin=exception_insight.so
```

❌ No need to update the standard
❌ No need to update the compiler
❌ No need to break the ABI
❌ No need for recompile your binaries*
🟡 Might need to update the linker
✅ Exceptions become modular

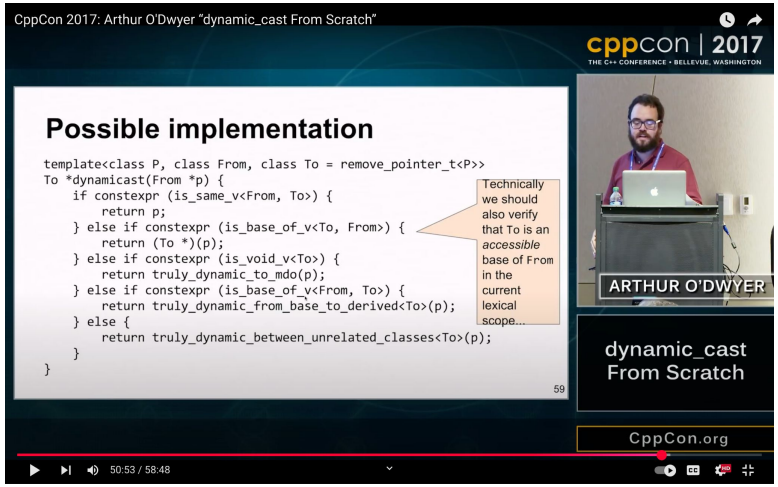
** -ffunction-sections (or similar) is needed for nearpoint ordering*

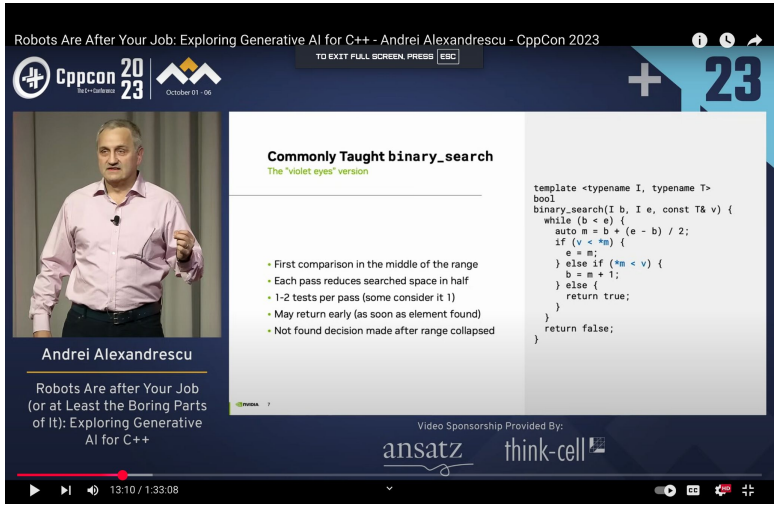
Special Thanks

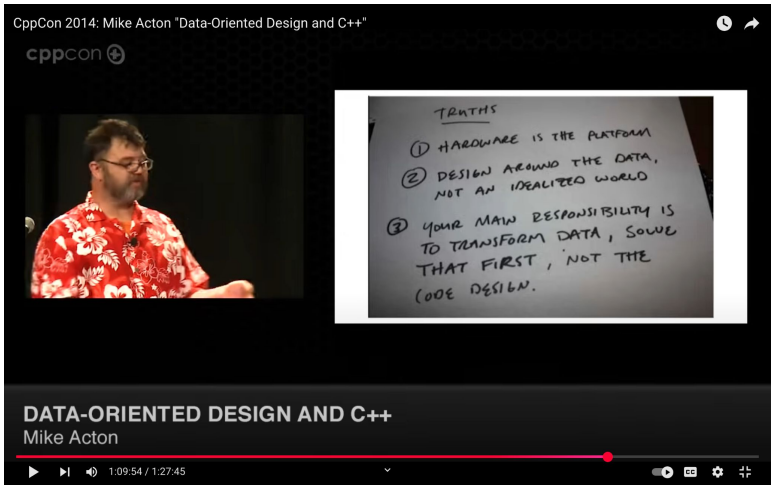
- Jason Turner
- Patrice Roy
- Guy Davidson
- Inbal Levi
- Ben Craig
- Lewis Baker
- Herb Sutter
- Bjarne Stroustrup
- Robin Rowe

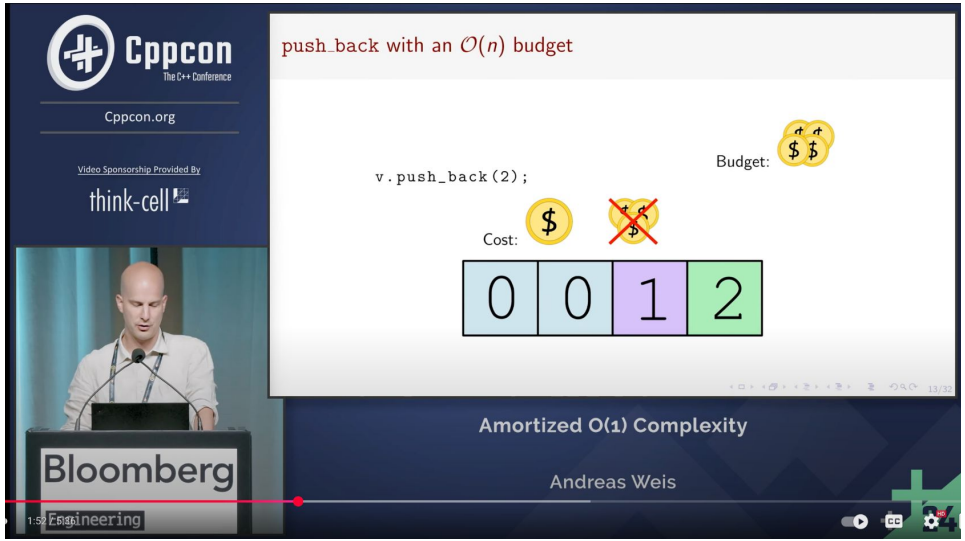
Special Thanks

- Jason Turner
- Patrice Roy
- **Guy Davidson**
- Inbal Levi
- Ben Craig
- Lewis Baker
- Herb Sutter
- Bjarne Stroustrup
- Robin Rowe


CppCon 2017: Arthur O'Dwyer "dynamic_cast From Scratch"
cppcon | 2017
THE C++ CONFERENCE • BELLEVUE, WASHINGTON
Possible implementation
template<class P, class From, class To = remove_pointer_t<P>>
To *dynamicast(From *p) {
if constexpr (is_same_v<From, To>) {
return p;
} else if constexpr (is_base_of_v<To, From>) {
return (To *)(p);
} else if constexpr (is_void_v<To>) {
return truly_dynamic_to_mdo(p);
} else if constexpr (is_base_of_v<From, To>) {
return truly_dynamic_from_base_to_derived<To>(p);
} else {
return truly_dynamic_between_unrelated_classes<To>(p);
}
}
Technically we should also verify that To is an accessible base of From in the current lexical scope...
59
ARTHUR O'DWYER
dynamic_cast From Scratch
CppCon.org
50:53 / 58:48


Robots Are After Your Job: Exploring Generative AI for C++ - Andrei Alexandrescu - CppCon 2023
TO EXIT FULL SCREEN, PRESS ESC
Cppcon 2023
October 01 - 06
Commonly Taught binary_search
The "violet eyes" version
First comparison in the middle of the range
Each pass reduces searched space in half
1-2 tests per pass (some consider it 1)
May return early (as soon as element found)
Not found decision made after range collapsed
template <typename I, typename T>
bool
binary_search(I b, I e, const T& v) {
while (b < e) {
auto m = b + (e - b) / 2;
if (v < *m) {
e = m;
} else if (*m < v) {
b = m + 1;
} else {
return true;
}
}
return false;
}
Andrei Alexandrescu
Robots Are after Your Job (or at Least the Boring Parts of It): Exploring Generative AI for C++
Video Sponsorship Provided By:
ansatz
think-cell
13:10 / 1:33:08


CppCon 2014: Mike Acton "Data-Oriented Design and C++"
cppcon
TRUTHS
1 HARDWARE IS THE PLATFORM
2 DESIGN AROUND THE DATA, NOT AN IDEALIZED WORLD
3 YOUR MAIN RESPONSIBILITY IS TO TRANSFORM DATA, SOLVE THAT FIRST, NOT THE CODE DESIGN.
DATA-ORIENTED DESIGN AND C++
Mike Acton
1:09:54 / 1:27:45


Cppcon
The C++ Conference
Cppcon.org
Video Sponsorship Provided By
think-cell
push_back with an $\mathcal{O}(n)$ budget
Budget:
v.push_back(2);
Cost:
0 0 1 2
Amortized O(1) Complexity
Andreas Weis
Bloomberg
Engineering

Questions?

How to support this research?

Consider making a 501(c)(3) eligible donation to my research fund!

San Jose State University Research Fund Name:
Exceptional Software Research Fund

Objective
Research & improve exception performance, size efficiency and tooling.

SJSU Tower Account #034-1300-1248

Contact Info

University Email:
khalil.estell@sjsu.edu
Inquiries:
estell.exceptions@gmail.com

Email me for benchmark source

Want To learn More?

Technical Research Briefings Available!

Interactive sessions where I share my current findings, methodologies, and future roadmap while gathering feedback on your team's specific needs.

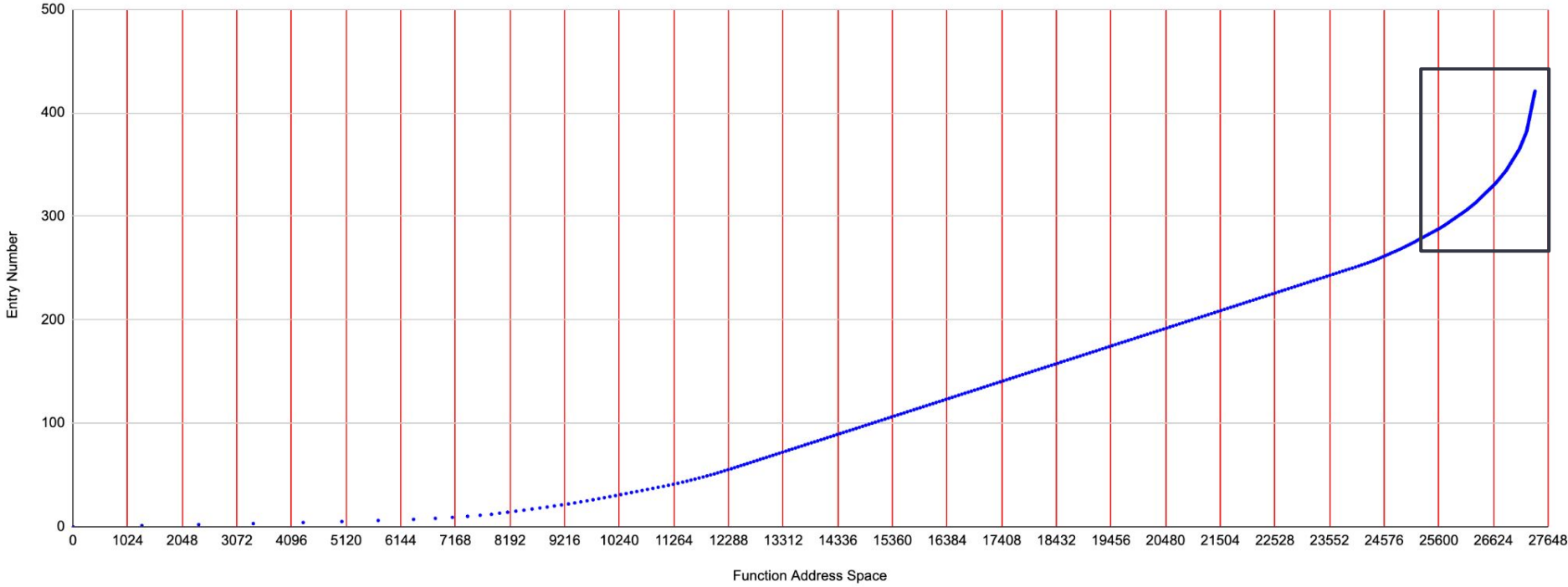
What you'll get

- Deep dive into exception performance breakthroughs
- Early insights into the Exception Insights tool development
- Understanding of how your codebase could benefit
- Opportunity to influence the research direction

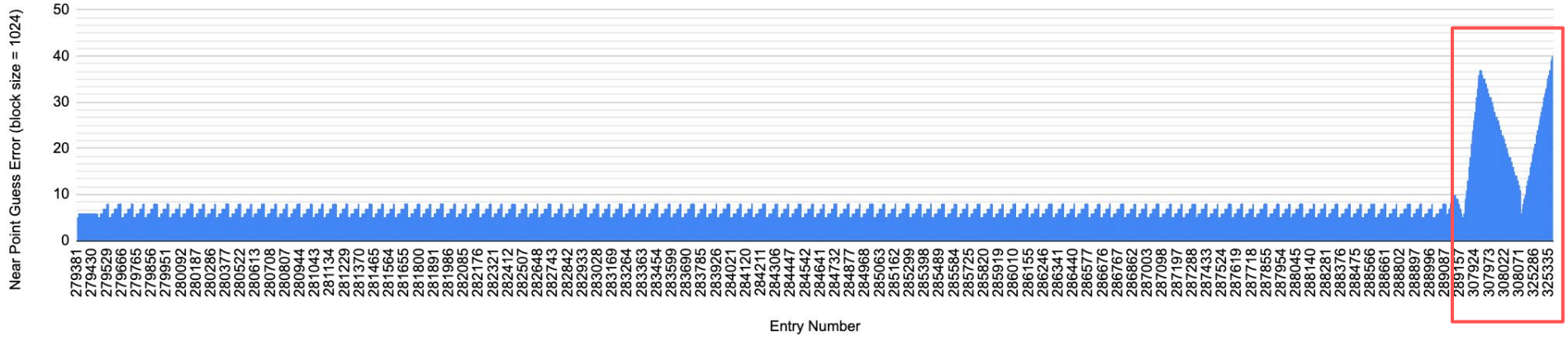
Interested in HALbORD?

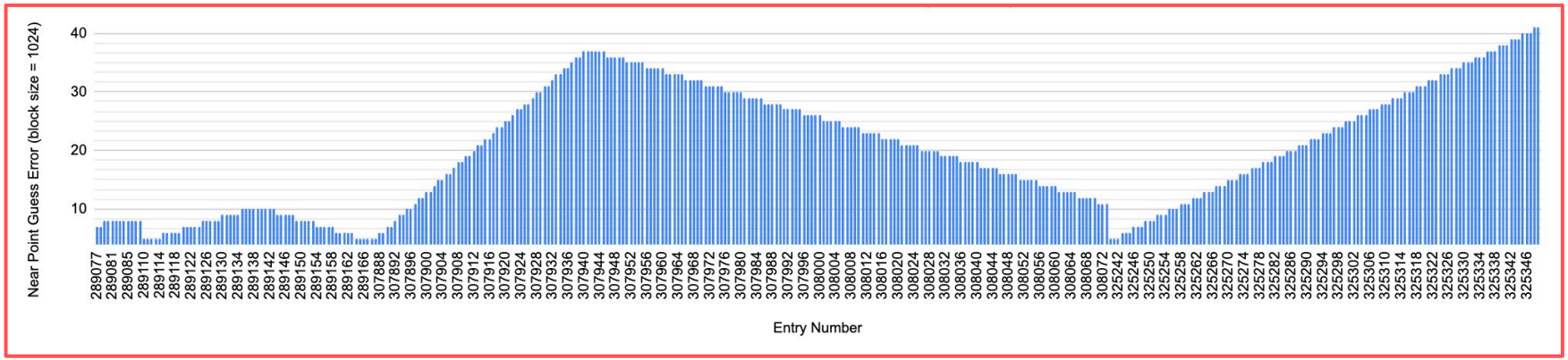
Scan the QR code to learn more!

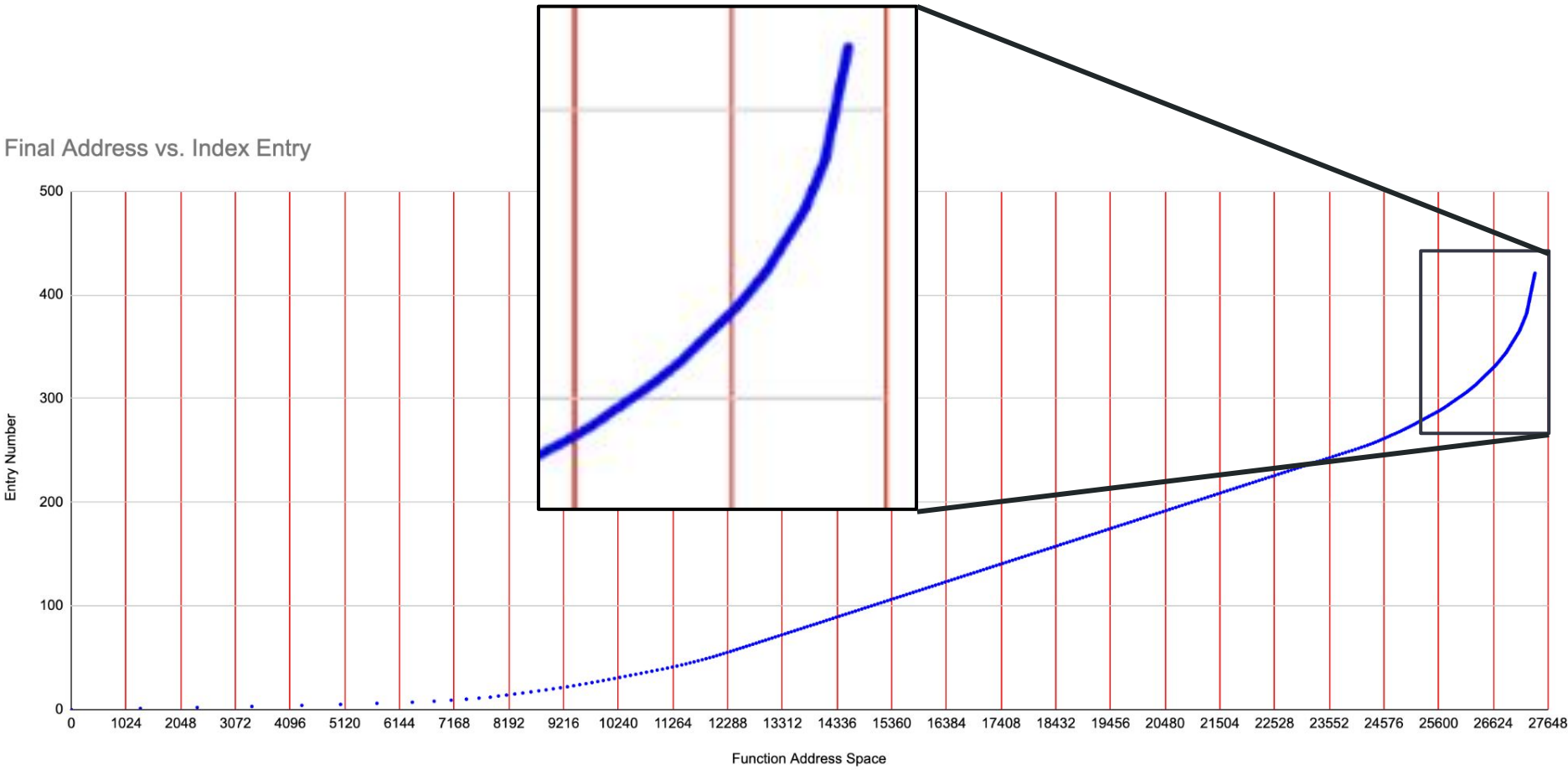
Final Address vs. Index Entry



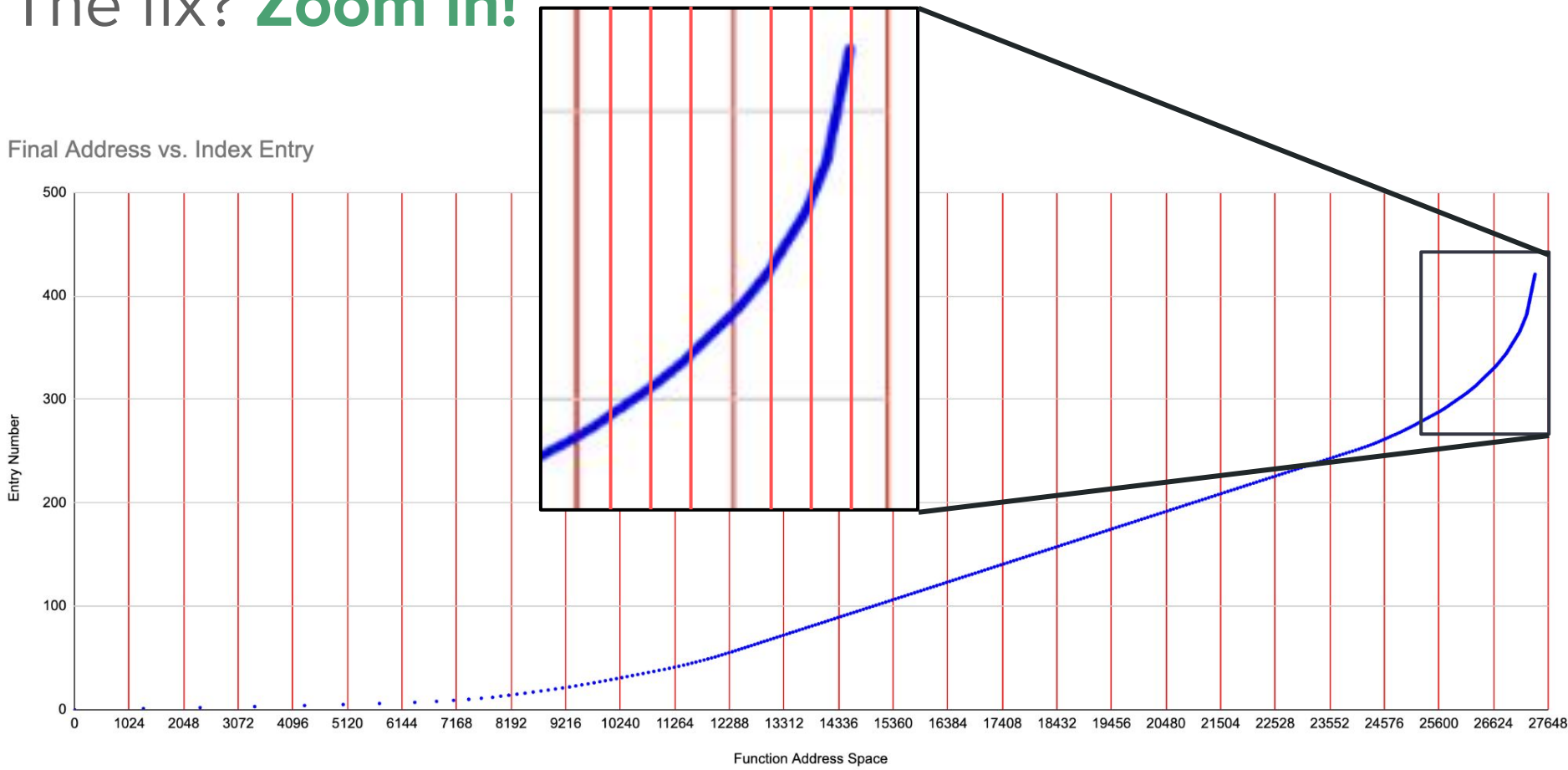
Nearpoint Simulated Error Testing



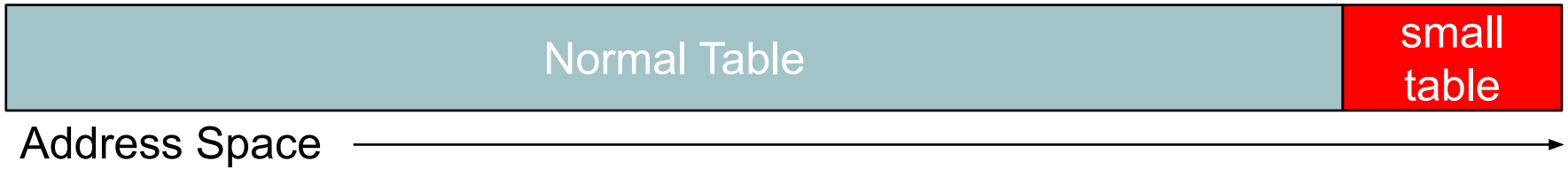



Final Address vs. Index Entry
Entry Number
Function Address Space
0
100
200
300
400
500
0
1024
2048
3072
4096
5120
6144
7168
8192
9216
10240
11264
12288
13312
14336
15360
16384
17408
18432
19456
20480
21504
22528
23552
24576
25600
26624
27648

The fix? **Zoom in!**



Near Point Linear Approx. Table




```
struct nearpoint_descriptor
{
 std::uint32_t normal_block_size = 0;
 std::uint32_t text_starting_address = 0;
 std::uint32_t small_block_size = 0;
 std::uint32_t small_starting_address = 0;
};
```