# C++ ABI: the only thing that is more important than performance

iCSC 2020

Nis Meinert

German Aerospace Center


Deutsches Zentrum
für Luft- und Raumfahrt

# Reading x86-64 Assembly

…for fun and profit

# Function Prologue & Epilogue

- Few lines of code at the beginning (*prologue*) and end (*epilogue*) of a function, which **prepares** (and eventually restores)
  - the **stack** and
  - **registers**
- Not part of assembly: **convention** (defined & interpreted differently by different OS and compilers)

**Prologue**

```
1 push rbp     ; rbp: frame pointer
2 mov rbp, rsp ; rsp: stack pointer
3 sub rsp, N
```

alternatively

```
1 enter N, 0
```

(reserve N bytes on stack for local use)

**Epilogue**

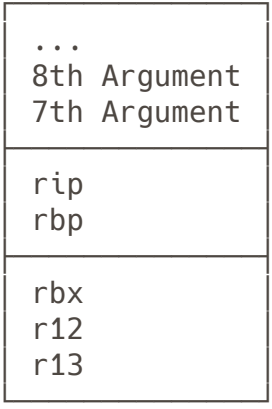
```
1 mov rsp, rbp
2 pop rbp
3 ret
```

alternatively

```
1 leave
2 ret
```

# Stack frame for function call

- CALL = PUSH *address of next instruction* + JMP *target*
- RET pops return address and transfers control there
- pass arguments 1 …6 in registers (`rsi`, `rdx`, …)




(stack frame for function call with 8 arguments and local registers `rbx`, `r12` and `r13`)

# Reading assembly for fun and profit

```
1  int f(int x, int y, int z) {
2      int sum = x + y + z;
3      return sum;
4  }
```

godbolt.org/z/MaWcP9

```
# g92 -O0
  | _Z1fiii:
 1|   push rbp
 1|   mov rbp, rsp
 1|   mov DWORD PTR [rbp-20], edi
 1|   mov DWORD PTR [rbp-24], esi
 1|   mov DWORD PTR [rbp-28], edx
 2|   mov edx, DWORD PTR [rbp-20]
 2|   mov eax, DWORD PTR [rbp-24]
 2|   add edx, eax
 2|   mov eax, DWORD PTR [rbp-28]
 2|   add eax, edx
 2|   mov DWORD PTR [rbp-4], eax
 3|   mov eax, DWORD PTR [rbp-4]
 4|   pop rbp
 4|   ret
```

godbolt.org/z/MaWcP9

# Reading assembly for fun and profit

```
1  int f(int x) {
2      return x + 1;
3  }
4
5  int g(int x) {
6      return f(x + 2);
7  }
```

godbolt.org/z/87GK4q

```
# g92 -O0
  | _Z1fi:
 1|   push rbp
 1|   mov rbp, rsp
 1|   mov DWORD PTR [rbp-4], edi
 2|   mov eax, DWORD PTR [rbp-4]
 2|   add eax, 1
 3|   pop rbp
 3|   ret
  | _Z1gi:
 5|   push rbp
 5|   mov rbp, rsp
 5|   sub rsp, 8
 5|   mov DWORD PTR [rbp-4], edi
 6|   mov eax, DWORD PTR [rbp-4]
 6|   add eax, 2
 6|   mov edi, eax
 6|   call _Z1fi
 7|   leave
 7|   ret
```

godbolt.org/z/87GK4q

# Reading assembly for fun and profit

```
1  void side_effect();
2
3  int f(int x) {
4      auto a = x;
5      side_effect();
6      return a - x;
7  }
```

godbolt.org/z/5xq5n5

```
# g92 -O0
  | _Z1fi:
 3|   push rbp
 3|   mov rbp, rsp
 3|   sub rsp, 32
 3|   mov DWORD PTR [rbp-20], edi
 4|   mov eax, DWORD PTR [rbp-20]
 4|   mov DWORD PTR [rbp-4], eax
 5|   call _Z11side_effectv
 6|   mov eax, DWORD PTR [rbp-4]
 6|   sub eax, DWORD PTR [rbp-20]
 7|   leave
 7|   ret
```

godbolt.org/z/5xq5n5

# Name mangling: C++ vs C

```
1  int f(int x) {
2      return x * x;
3  }
4
5  extern "C" int g(int x) {
6      return x * x;
7  }
```

godbolt.org/z/cj7bqx

```
# g92 -O0
  | _Z1fi:
 1|   push rbp
 1|   mov rbp, rsp
 1|   mov DWORD PTR [rbp-4], edi
 2|   mov eax, DWORD PTR [rbp-4]
 2|   imul eax, eax
 3|   pop rbp
 3|   ret
  | g:
 5|   push rbp
 5|   mov rbp, rsp
 5|   mov DWORD PTR [rbp-4], edi
 6|   mov eax, DWORD PTR [rbp-4]
 6|   imul eax, eax
 7|   pop rbp
 7|   ret
```

godbolt.org/z/cj7bqx

# Name mangling: C++ vs C

```
1  int f(int x) {
2      return x * x;
3  }
4
5  extern "C" int g(int x) {
6      return x * x;
7  }
```

godbolt.org/z/cj7bqx

**Why?**

- → overloading
- → namespaces
- → templating

(Name of function doesn't suffice to resolve JMP location)

```
# g92 -O0
  | _Z1fi:
 1|   push rbp
 1|   mov rbp, rsp
 1|   mov DWORD PTR [rbp-4], edi
 2|   mov eax, DWORD PTR [rbp-4]
 2|   imul eax, eax
 3|   pop rbp
 3|   ret
  | g:
 5|   push rbp
 5|   mov rbp, rsp
 5|   mov DWORD PTR [rbp-4], edi
 6|   mov eax, DWORD PTR [rbp-4]
 6|   imul eax, eax
 7|   pop rbp
 7|   ret
```

godbolt.org/z/cj7bqx

# Name mangling in C++

```
1  void f(int) {}
2
3  void f(double) {}
4
5  namespace my_fancy_namespace {
6  void f(int) {}
7  } // my_fancy_namespace
```

godbolt.org/z/jWY14x

```
# g92 -O2
  | _Z1fi:
 1|   ret
  | _Z1fd:
 3|   ret
  | _ZN18my_fancy_namespace1fEi:
  |   ret
```

godbolt.org/z/jWY14x

- → C++ does not standardize name mangling
- → *Annotated C++ Reference Manual* even actively discourages usage of common mangling schemes. (Prevent linking when other aspects of ABI are incompatible.)

# What is ABI?

# What is ABI (*Application Binary Interface*)?

**Specifies interaction of functions and types across TUs[†] (translation units)**

- → Platform-specific (*e.g.*, Linux on x86-64 CPU)
- → Vendor-specified (*e.g.*, gcc)
- → not controlled by WG21

(Titus Winters: *Similar to a binary network protocol*)

Photo by Spencerian at `en.wikipedia.org` (2005)

[†] *TU*: ultimate input to the compiler from which an object file is generated (*i.e.*, typically the `.cpp` file)

# What is ABI (*Application Binary Interface*)?

**Specifies interaction of functions and types across TUs[†] (translation units)** covering:

- → Name mangling of functions
- → Name mangling of types
- → `sizeof` and alignment of objects
- → Bytes semantics of the binary representation of objects
- → Calling convention

(Titus Winters: *Similar to a binary network protocol*)

Photo by Spencerian at `en.wikipedia.org` (2005)

[†] *TU*: ultimate input to the compiler from which an object file is generated (*i.e.*, typically the `.cpp` file)

# Why should I care?

…do you depend on any pre-compiled shared library?

# Why should I care?

**Why should I care?**

- → **Linking** different TUs requires usage of same ABI
- → Typically a problem at API boundaries when combining TUs (*e.g.*, shared libraries) that were compiled at different **time**s
- → Similar to binary network protocols: ABI tells you how to interpret bytes

Why should I care? $\Leftrightarrow$ Why do network protocols have versions?

(Problem: not all ABIs encode version number)

# ABI: the problem

**ABI does not encode version number**

- **Q**: How to check if a given TU uses a compatible ABI?
- **A**: You can't!
- What happens if ABI is incompatible?
  (a) Linking fails during compile time (good)
  (b) Program spectacularly dies during run time (bad)
- Why isn't this a common problem?
  - Itanium ABI is mostly stable since C++11


History

# ABI breakage of `std::string`

- → Before C++11: libstdc++ relied on copy-on-write (COW)
- → C++11 disallows COW
  - → fewer indirections
  - → short string optimization (SSO)
- → Problem: passing COW string to impl that expects SSO **may link** (same mangled name)!
  - → one word passed
  - → three words read
- → *Solution*[†]: gcc changed mangled name

```
1  // until C++11
2  struct string {
3      struct control_block {
4          /* ... */
5      };
6      control_block *data;
7  };
8
9  // since C++11
10 struct string {
11     char *data;
12     std::size_t size;
13     std::size_t capacity;
14 }
```

godbolt.org/z/KM5Tvq

↪ Take-away for compiler vendors: ABI break was a huge disaster

[†] RHEL 7 still uses old `std::string` ABI to provide compatibility for older `.so`

# Quiz Time

# Quiz: Will it break ABI?

**Proposal:** make `std::vector<T>::push_back` return a reference to the element in its new location

```
void push_back(const T&);
```

```
T& push_back(const T&);
```

# Quiz: Will it break ABI?

```
1  template <typename T>
2  struct vector {
3      void push_back_1(const T&);
4      T& push_back_2(const T&);
5  };
6
7  void f(vector<int> v) {
8      v.push_back_1(42);
9      v.push_back_2(42);
10 }
```

godbolt.org/z/9def7a

```
# g92 -O0
  | _Z1f6vectorIiE:
 7|   push rbp
 7|   mov rbp, rsp
 7|   sub rsp, 16
 8|   mov DWORD PTR [rbp-8], 42
 8|   lea rax, [rbp-8]
 8|   mov rsi, rax
 8|   lea rdi, [rbp+16]
 8|   call _ZN6vectorIiE11push_back_1ERKi
 9|   mov DWORD PTR [rbp-4], 42
 9|   lea rax, [rbp-4]
 9|   mov rsi, rax
 9|   lea rdi, [rbp+16]
 9|   call _ZN6vectorIiE11push_back_2ERKi
10|   nop
10|   leave
10|   ret
```

godbolt.org/z/9def7a

# Quiz: Will it break ABI?

Both, `void push_back` and `T& push_back` have the same mangled name (Itanium ABI)

- → **Two** definitions in the old and the new TU
- → ODR violation
- → Linker will pick only **one** definition (by **overwriting** the other)
- → ABI break: reading return value from `eax` when there is none

# Quiz: Will it break ABI?

**Proposal:** make `std::vector<T>::emplace_back` return a reference to the element in its new location

```
template<class... Args> void emplace_back(Args&&...);
```

```
template<class... Args> T& emplace_back(Args&&...);
```

# Quiz: Will it break ABI?

```
1  template <typename T>
2  struct vector {
3      template<class... Args>
4      void emplace_back_1(Args&&...);
5
6      template<class... Args>
7      T& emplace_back_2(Args&&...);
8  };
9
10 void f(vector<int> v) {
11     v.emplace_back_1(42);
12     v.emplace_back_2(42);
13 }
```

godbolt.org/z/dYMsza

# Quiz: Will it break ABI?

```
1  template <typename T>
2  struct vector {
3      template<class... Args>
4      void emplace_back_1(Args&&...);
5
6      template<class... Args>
7      T& emplace_back_2(Args&&...);
8  };
9
10 void f(vector<int> v) {
11     v.emplace_back_1(42);
12     v.emplace_back_2(42);
13 }
```

godbolt.org/z/dYMsza

**Mangled names:** (Itanium ABI)

1. `_ZN6vectorIiE14emplace_back_1IJiEEEvDpOT_`
2. `_ZN6vectorIiE14emplace_back_2IJiEEERiDpOT_`

# Quiz: Will it break ABI?

`void emplace_back` and `T& emplace_back` have different mangled names (Itanium ABI)

- → Two definitions in the old and the new TU
- → but no ODR violation
- → No ABI break: old code calls the old one, new code calls the new one

# Quiz: Will it break ABI?

**Proposal:** extend `std::lock_guard<T>` to allow for a variadic set of heterogeneous mutexes

```
template<class Mutex> class lock_guard;
```

```
template<class... Mutexes> class lock_guard;
```

# Quiz: Will it break ABI?

```
1  template <typename>
2  struct lock_guard_1 {
3      lock_guard_1() {}
4  };
5
6  template <typename...>
7  struct lock_guard_2 {
8      lock_guard_2() {}
9  };
10
11 void f() {
12     lock_guard_1<int> l1{};
13     lock_guard_2<int> l2{};
14 }
```

godbolt.org/z/MKPq35

```
# g92 -O0
   | _Z1fv:
11|   push rbp
11|   mov rbp, rsp
11|   sub rsp, 16
12|   lea rax, [rbp-1]
12|   mov rdi, rax
12|   call _ZN12lock_guard_1IiEC1Ev
13|   lea rax, [rbp-2]
13|   mov rdi, rax
13|   call _ZN12lock_guard_2IJiEEC1Ev
14|   nop
14|   leave
14|   ret
   | _ZN12lock_guard_1IiEC2Ev:
 3|   push rbp
 3|   mov rbp, rsp
 3|   mov QWORD PTR [rbp-8], rdi
 3|   nop
[...]
```

godbolt.org/z/MKPq35

# Quiz: Will it break ABI?

`<class T> class` and `<class... T> class` have different mangled names (Itanium ABI)

- → ABI break: for example in `auto f(std::lock_guard<M>& lk);`
  - → *cf.* `godbolt.org/z/Pex6Gx`
- → User compiles `f` using old `lock_guard`
- → User then tries to call it from a TU using new `lock_guard`
- → Mangled names don't match: linker error!

# Quiz: Will it break ABI?

**Proposal:** change hashing by `std::hash` to improve performance of `std::unordered_map` by 3-4x (*cf.* `absl::node_hash_map`)

# Quiz: Will it break ABI?

**Proposal:** change hashing by `std::hash` to improve performance of `std::unordered_map` by 3-4x (*cf.* `absl::node_hash_map`)

**ABI break:**

- → Hash value for an object is computed in old TU and stored in map
- → (Different) hash value is computed in new TU and used to lookup value in map
- → **Semantic meaning of binary representation has changed!**

# Other Examples

More examples:

- `std::regex` currently is 10-100x slower than equivalents in Rust or Go (*cf.* any talk of Hana Dusíková)
- Make `std::unique_ptr` zero-overhead (*cf.* Chandler Carruth, *There Are No Zero-cost Abstractions*)
- Add `std::int128_t` which already is supported on more and more platforms
- Make `std::bitset` trivially destructible
- Exceptions (would be an entire talk on its own)
- …

(read `P2028` and `P1863` by Titus Winters for more information)

# Future Prospects

# Future Prospects

**Break ABI with future releases:** move run-time failures to compile-time (when possible) by changing mangled name

- → new namespaces: *e.g.*, `std2::` or `std::abi42::`
  - → developers now have to choose between `std::optional` and `std2::optional` or duplicate code with overloading
  - → when will `std3::` arrive?
- → change entire mangling scheme: *e.g.*, _Z ↦ _Y on Itanium
  - → Compiler vendors could ship both forms
- → Introduce version number (ABI break)
  - → MSVC does already include version numbers in DLLs (MSVCs users are used to recompile with each new version of Visual Studio …)

# State-of-the-Art

- No (major?) ABI breaks since C++11 (**12 years in 2023!**)
- Hyrum's Law (or `xkcd.com/1172`): passing ABI-unstable types across ABI boundaries happened to work for more than a decade. People rely on ABI stability, even though it was never explicitly promised.
- Expose C-APIs whenever possible

# State-of-the-Art

**Standard Meeting, Prague 2020**

- → WG21 is not in favor of an ABI break in C++23 or C++26
- → WG21 is in favor of an ABI break in a future version of C++
- → WG21 will take time to consider proposals requiring an ABI break
- → WG21 will not promise stability forever
- → WG21 wants to keep prioritizing performance over stability

**Bryce | BlackLivesMatter** @blelba... · 16 Feb
Performance
ABI Stability
Ability to Change

You can pick two, choose wisely.

#cppprg

25 12 63

Twitter: @blebach (2020)

(Corentin Jabot: *There was no applause. But I'm not sure we fully understood what we did and the consequences it could have.*)

# State-of-the-Art

**Jonathan Müller** @foonathan · 03 Feb
What's the point of standard library containers if they can't give the best performance?

2 11

**Titus Winters**
@TitusWinters

Replying to @foonathan

They're common and readily available? (Which does have some value.)

Committing to ABI is like admitting that the standard library is aiming to be McDonald's - It's everywhere, it's consistent, and it technically solves the problem.

18:02 · 03 Feb 20 · Twitter Web App

**2** Retweets **1** Quote Tweet **17** Likes

Twitter: `@TitusWinters` (2020)

# Literature

- Titus Winters, P2028: *What is ABI, and What Should WG21 Do About It?*
- Titus Winters, P1863: *ABI - Now or Never*
- Roger Orr, P1654: *ABI breakage - summary of initial comments*
- Titus Winters on CppCast #224: *The C++ ABI*
- John Lakos on CppCast #233: *Large Scale C++*
- Corentin Jabot on `cor3ntin.github.io/posts/abi/`: *The Day The Standard Library Died*
- JeanHeyd Meneide on `thephd.github.io/freestanding-noexcept-allocators-vector-memory-hole`
- Danila Kutenin on `youtu.be/GRuX31P4Ric`: *C++ STL best and worst performance features and how to learn from them*