Break ABI to Save C++

Notes

- This will be high level
- I'll definitely gloss over some details
- When others discuss this topic they tend to focus on the "Leave no room for a lower level language" guiding principle of C++
- *I* am going to focus on Best Practices and why ABI stability is probably making your code worse right now
- You really want to stick through the whole thing because I also cover a way to move forward at the end

What is ABI?

- Application
- Binary
- Interface

https://en.wikipedia.org/wiki/Application_binary_interface

What is ABI?

It is how your objects and types look to the compiler. Some things that are included in the discussion of ABI:

- Size, number and order of data members in a type
- Size, number and order of virtual functions
- Number and order of function parameters
- Function return types

What happens when we change the return type?

Changing the return type

- https://godbolt.org/z/5e8K8h9GM
- Return type is not part of the name mangling, so if the header doesn't match the binary, things can go very wrong

What happens when we change the order of virtual functions?

Changing The Order of Virtual Functions

- https://godbolt.org/z/PbGfn8df4
- If you change the order of virtual functions, the compiler will call the wrong one!

What happens when we change the order of member variables?

Changing The Order of Member Variables

- https://godbolt.org/z/dW8YocG1Y
- Just like changing virtual function order, the compiler would generate the wrong offset

OK, so why would you ever actually do this?

Old Library A and New Executable B are speaking a different language, AND THEY DON'T KNOW IT!

They are using the same words, but they have different meaning!

ABI Stability - What We Gain

To prevent ABI breakage related bugs, our compiler and standard library vendors have been giving us “ABI Stability” for some time now.

- This is good, in some ways
- We can build a new C++ executable on our OS with a new version of the compiler and know that our system provided C++ libraries, compiled with an older compiler will work just fine.

ABI Stability - What is Stuck

- Can never change the return type of a function
- Can never add/remove/reorder virtual functions
- Can never add/remove/reorder member variables
- Can never add/remove/reorder function parameters (but we can add new overloads, which mostly solves that issue)

ABI Stability - What is Potentially Lost

- Stdlib have a bug that would require changing layout? Sorry, we're stuck with that bug!
- Would adding/removing a member variable to std::vector increase performance or save bytes? Can't do it!
- Is there a bug in the design of the stdlib that needs to be fixed? Sorry, that's probably an ABI break!

ABI Stability - What Have We Actually Lost?

- https://cor3ntin.github.io/posts/abi/
- Rumors of committee members quitting
- https://docs.microsoft.com/en-us/cpp/overview/visual-cpp-language-conformance?view=msvc-160

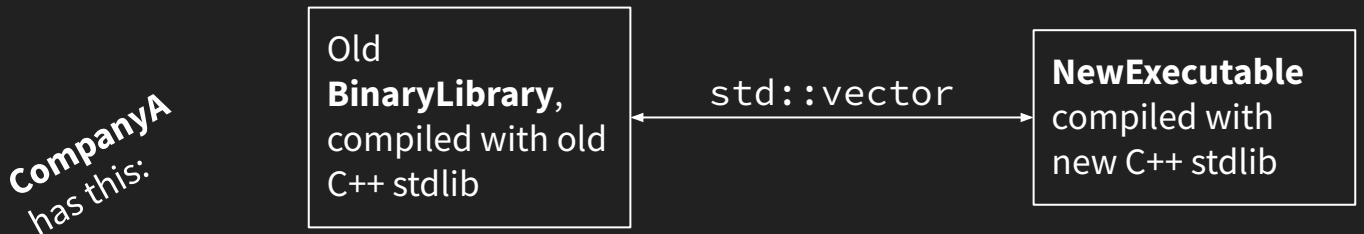
ABI Stability - Why Do We Have It?

If so much is lost by maintaining ABI stability, why do we have it?

My understanding is: users demand it.

Users Demand ABI Stability!

They say “I rely on this old binary-only library from a vendor (or from my own company), and the sources are lost or unavailable and I MUST have ABI stability if I’m going to use the next version of your compiler!”



With this model, `std::vector` *can never change!*

Fine, but will std::vector really need to change?

Maybe not, but what about regex, or function or map (associative containers in general)?

Is **CompanyA** in a Good or Bad Situation?

CompanyA:

- Has an old **BinaryLibrary** of unknown quality
- They cannot fix discovered bugs (well technically they might be able to patch the binary?)
- Probably carrying UB and not-yet-discovered security issues they cannot fix

CompanyA:

- Thinks everything is fine, their product works!
- But the issue is more like:

How Can I Possibly Know Their Code is Broken?!

- Every major compiler release fixes 100’s of bugs
- **BinaryLibrary** was compiled with an older compiler
- Therefore, **BinaryLibrary** was compiled with a buggy compiler
- Q.E.D.

OK, That’s a Bit of Hyperbole, But Seriously

- **BinaryLibrary** didn’t have the advantage of:
  - State of the art static analysis
  - Modern compiler warnings
  - Runtime analysis with sanitizers

By refusing to break ABI we are encouraging and enabling companies to use old, broken, insecure, and unfixable code that was compiled with a buggy compiler!

This only helps enforce the idea that C++ is an insecure language!

To Save C++, We Must Break ABI in the Standard Library

- We can fix stdlib bugs
- We can improve stdlib performance
- We discourage the use of libraries of unknown quality

Why Haven’t We Done This Yet?

- We don’t want to splinter the C++ community because of large companies refusing to update compilers and deal with ABI breakage
- A silent ABI break is very dangerous, so this must be detectable at compile, link, or runtime
- What about the people who have truly no other option, and must use an old **BinaryLibrary**?

1. We don’t want to splinter the C++ community because of large companies refusing to update compilers and deal with ABI breakage

Newsflash: Large companies already cling to old compilers because large companies move slowly!

2. A silent ABI break is very dangerous, so this must be detectable at compile, link, or runtime

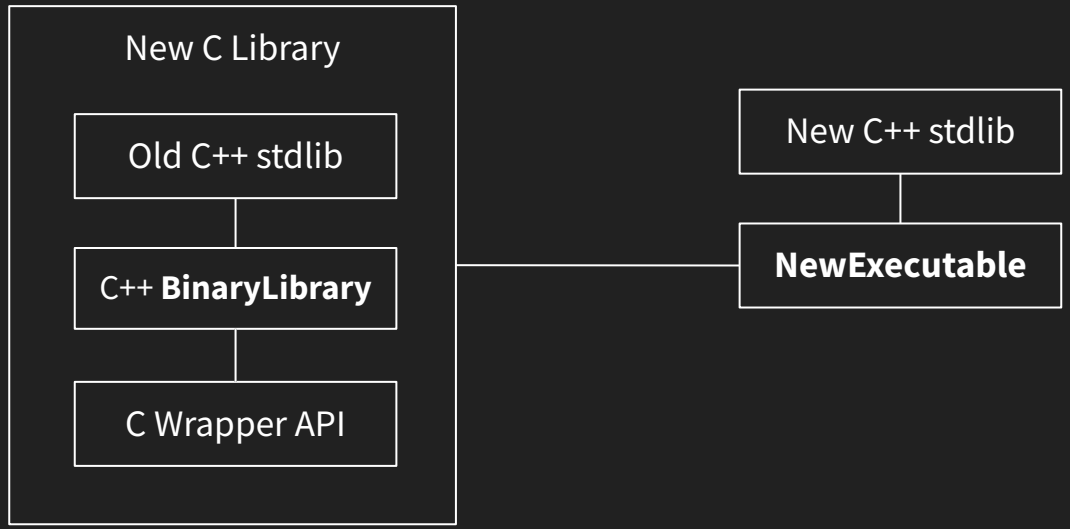
DLL versioning is a solved problem. Every toolchain has a way to deal with different versions of the same library. Just increment the stdlib version number.

3. What about the people who have truly no other option, and must use an old **BinaryLibrary**?

Option A: Reimplement the **BinaryLibrary**. This is the best option, but might be a lot of work.

Option B: Provide a C Wrapper

Providing a C Wrapper for Your **BinaryLibrary**



Note: This is kind of how Windows system DLLs are written

Providing a C Wrapper for Your **BinaryLibrary**

- This allows you to insulate yourself from the different stdlib versions
- Windows system DLLs are written in a way similar to this, with C++ internally, but only C facing functions
- It is not ideal, but with some creativity you can move past an ABI break
- For bonus points, re-wrap the C code in a header-only C++ library
- This will make fully aware of which functions you actually need from your **BinaryLibrary**
- Now you know which functions you need to reimplement when you replace your **BinaryLibrary** with something you can maintain in the future
- Example: https://godbolt.org/z/YexPsd4hM

Break ABI to Save C++

- This can be done, so we can move forward
- There should be a schedule for when it is allowed
- I have conducted informal polls with all recent CppCast guests, and they all think it is important to break ABI
- It can be done safely
- There are options for people stuck with a **BinaryLibrary**