llvm-mingw

https://github.com/mstorsjo/llvm-mingw

+

MinGW-w64
A complete runtime environment for GCC & LLVM
for 32 and 64 bit Windows

Martin Storsjö, FOSDEM 2026

What is mingw?

- ”Minimalistic GNU for Windows”
- Commonly thought of as ”GCC on or for Windows”
- A traditional mingw toolchain consists of GCC + binutils + mingw
    - The ”mingw” component provides Windows headers and libraries
        - Standalone reimplementation of the whole Windows SDK
        - Freely redistributable
        - These days: Mainly the mingw-w64 fork
- Mingw is for building totally normal, native Windows executables
    - It is NOT about emulating POSIX (which Cygwin does)

What does LLVM do?

- On Windows, LLVM can emulate both MSVC and GCC
  - `-target <arch>-pc-windows-msvc`
  - `-target <arch>-w64-mingw32` aka `<arch>-w64-windows-gnu`

What is llvm-mingw?

- Most LLVM components can be used as drop-in replacements in an existing toolchain
  - Clang ↔ GCC
  - LLD ↔ GNU ld
  - compiler-rt + libunwind ↔ libgcc
  - libc++ ↔ libstdc++
  - LLDB ↔ GDB
- llvm-mingw is set up standalone from scratch with all components replaced with LLVM counterparts

Why?

- ARM/ARM64
- PDB (Microsoft debugging format)
- Sanitizers
- C++11 threads without involving winpthreads library
- Windows native TLS (instead of emulated TLS)
- Working weak symbols
- One single toolchain targeting cross compilation for 4 architectures
- Built on a modern codebase, easy to work on

GCC catching up

- ARM/ARM64 - ARM64 target in progress
- PDB (Microsoft debugging format) - In progress
- Sanitizers - In progress
- C++11 threads without involving winpthreads library - Available since GCC 13
- Windows native TLS (instead of emulated TLS) - Available since GCC 16
- Working weak symbols - some issues still present
- One single toolchain targeting cross compilation for 4 architectures - No
-

Goal

- Goal: Drop-in compatibility for existing projects that can be built in a mingw configuration
    - Source level compatibility
    - Build tool interface compatibility

Origin story

Origin story

- Background in multimedia; FFmpeg, VLC, etc
    - Porting FFmpeg to odd mobile platforms since 2006
- Goal: VLC on Windows Phone in 2014
    - VLC is very heavily tied to mingw build tools, building >100 third party libraries, mainly with autoconf/automake
        - Primarily cross compiled from a Unix
    - No upstream support for Windows outside of x86 in upstream binutils/GCC
        - Stalled efforts to have a GCC maintainer work on this

VLC on Windows Phone

- Wrap MSVC cl.exe in a shell script
  - Interpret GCC style options, remap them to corresponding MSVC ones
  - Implement enough to get the build going
- VLC for Windows Phone was shipped, built like this
  - Did work, but painful and slow to build

Better ways to get mingw for ARM

- LLVM had some support for Windows on ARM
- LLVM did support targeting mingw, on x86, in existing mingw toolchains
- Missing:
    - Linker
    - Building compiler-rt, libunwind, libc++
    - Building the mingw-w64 libraries
    - Tying it all together
- Initial efforts by Martell Malone, 2015-2017, I got involved in 2016

New grounds: ARM64

- In 2017, there were rumors about Windows coming to ARM64
- WinSDK contained a handful of EXEs for ARM64, and some libraries. (MSVC didn’t include anything targeting ARM64 at this time)
- Wine got initial support for ARM64 by André Zwing
- Tested it out, could successfully run maybe a couple EXEs from WinSDK
- Tried debugging failing ones - crashing in printf
    - Need to implement a different calling convention for printf, in a compiler
    - Familiar with the LLVM codebase, easy to hack up something that seems to work, and got some more executables working

New grounds: ARM64

- Have a mostly working test environment in Wine
- LLVM/Clang got some initial commits for writing COFF-ARM64 in mid 2017, by people from Qualcomm
- Had WinSDK libraries for ARM64, but not the MSVC parts/libraries
- No MSVC compiler
- Improved code generation, reverse engineering how relocations should work, implementing linker: able to compile small C executables
  - Preexisting MS `link.exe` turned out to be capable of linking ARM64, allowing figuring out relocations
- Folded the ARM64 effort into the overall LLVM+mingw effort, added initial support for ARM64 in mingw-w64
  - Notable side quest: UCRT support in mingw-w64

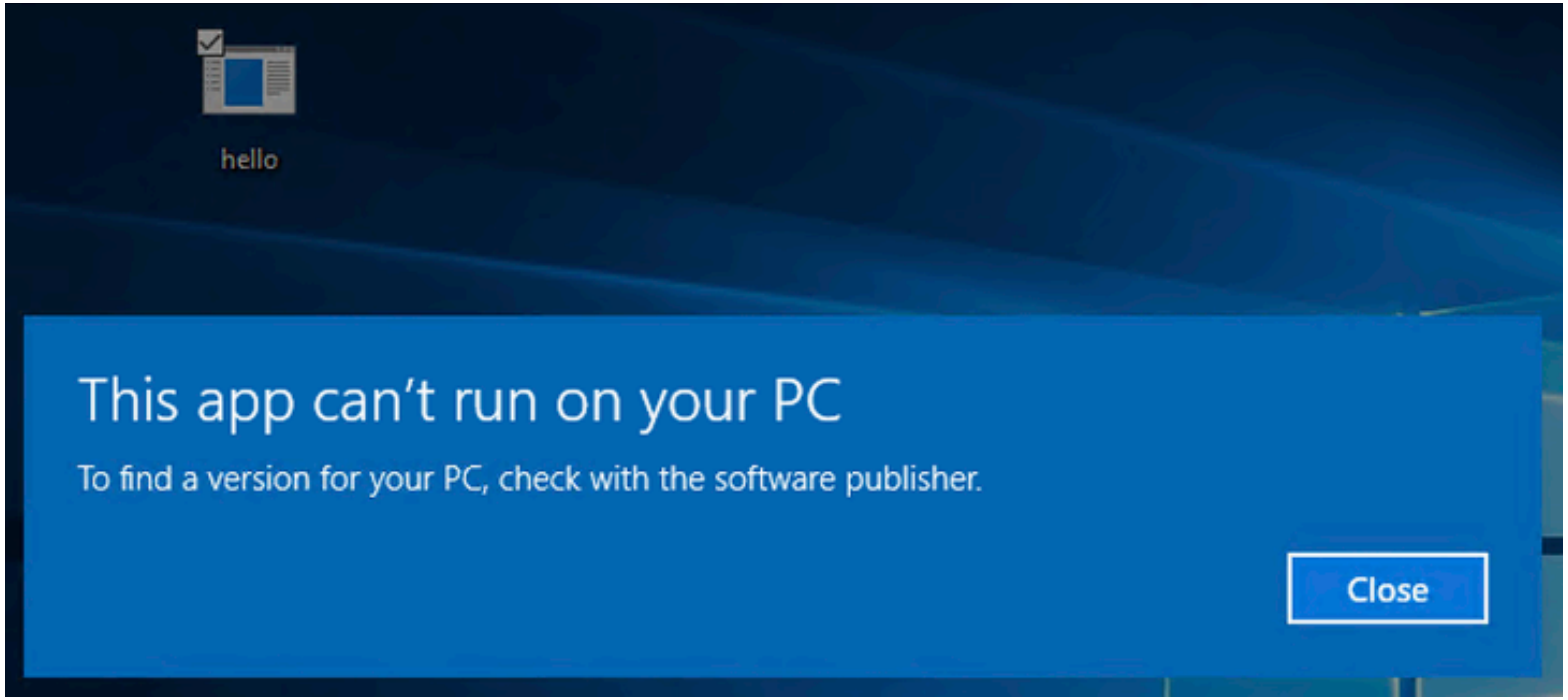
Founding llvm-mingw

- By the end of 2017, had a seemingly working toolchain
  - Had managed to find a working configuration of libunwind, libcxxabi, libcxx - working C++ target
  - Targeting ARM64 worked mostly as well as other architectures
- The mingw linker interface for LLD, by Martell Malone, was finally merged
  - A somewhat working LLVM+mingw environment could be bootstrapped, without any non-upstream patches
- No intent of maintaining a toolchain distribution in itself, but building this setup was kinda complicated, with lots of hacks, so I had to share a reference example of it
  - The project graduated from a branch on VideoLAN's ”docker-buildbots” to a separate repo ” llvm-mingw” on GitHub - by the end of 2017

Testing on the real OS

- Had a mostly working toolchain, and a mostly working ARM64 target that had been developed and tested with Wine
- VideoLAN managed to get a prototype ARM64 device from Microsoft
- Could try out my existing toolchain with the real OS now
    - Did it work?

Testing on the real OS



This app can’t run on your PC

To find a version for your PC, check with the software publisher.

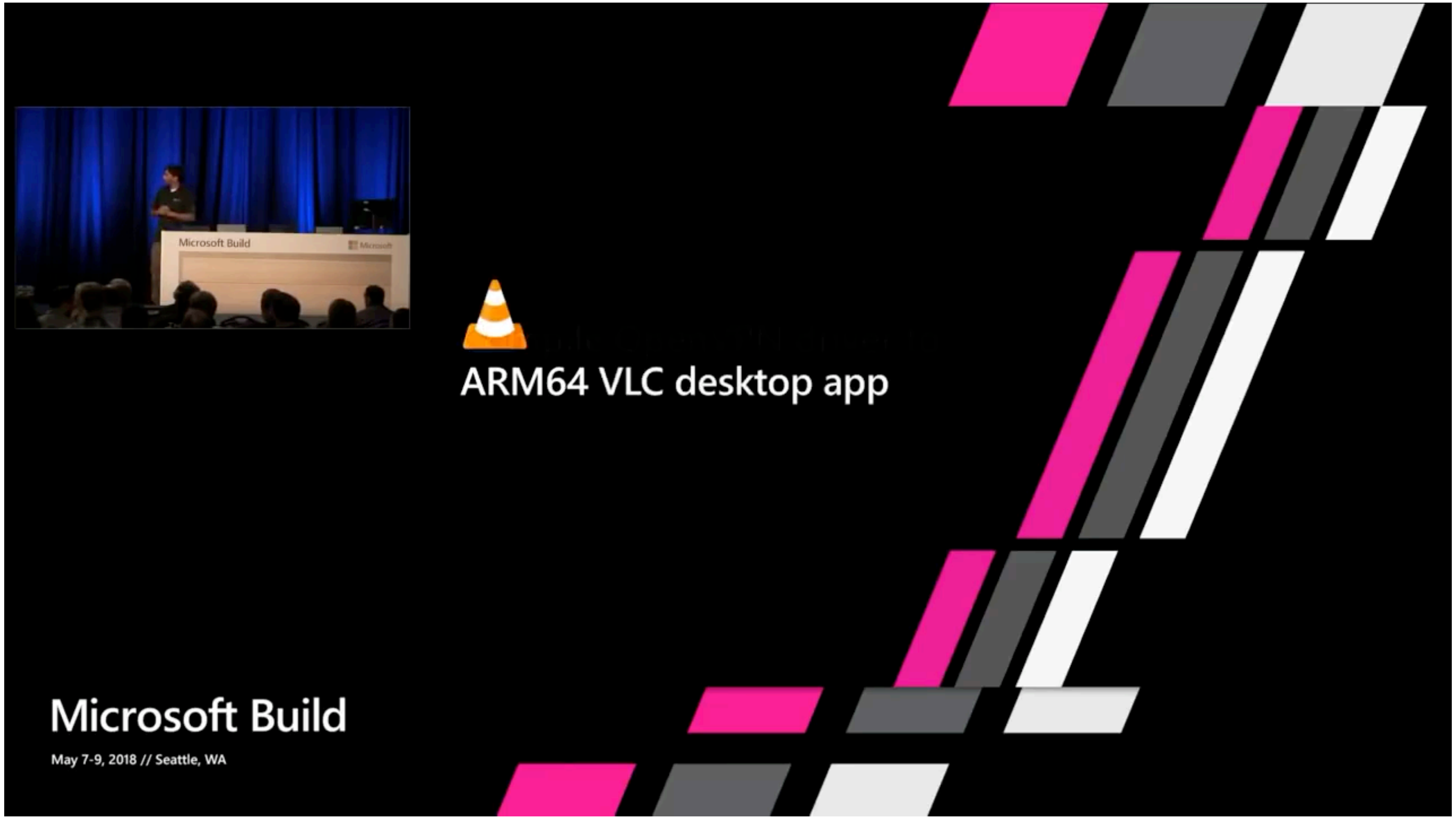
Testing on the real OS

- Did it work? Almost - three issues remained:
  - Windows on ARM(64) refuses to load binaries without the dynamicbase flag set
  - Missing stack probing; Wine on Linux doesn't need stack probing. Simple executables worked, ones that had functions with a large stack crashed
  - setjmp was broken - the UCRT (and msvcrt) setjmp functions require SEH unwind info
    - Qt/FreeType does a few setjmp/longjmp
    - Near trivial to implement custom setjmp/longjmp functions
- Manage to make a working build of VLC (with a bunch of hacks) in 2 weeks

MS Build Conference 2018

- May 2018, Microsoft unveiling Windows for ARM64 for developers
- Showcasing how to port apps to it, by recompiling with the latest Visual Studio

MS Build Conference 2018



ARM64 VLC desktop app

Microsoft Build

May 7-9, 2018 // Seattle, WA

MS Build Conference 2018

- Youtube, “Windows 10 on ARM for developers : Build 2018” https://youtu.be/vdYIaUeZnqc?t=1337 at 22:17
- "So we shipped him a couple of devices, we loaded them up with Windows, we gave them preview tools and in shockingly short order of time they had this up and running.”
- "When I asked Jean-Baptiste how much code did he have to change, his answer: Zero. He changed zero lines of code. His biggest problem was, again, VLC has its own homebrew way of building things, so ingesting the latest Visual Studio preview compiler, into his build system, that was his biggest challenge. Changing code, zero. He didn't have to change a single line of code, everything just worked.”
  - The preliminary stack of patches was somewhere around 60-70 patches
- Alternative conclusion: It’s more attractive to build a new compiler backend and assemble a new toolchain from scratch, than to integrate autotools based projects with the MSVC compiler.

Project maturing

Old hacks

- In 2018, LLVM didn’t have an objcopy/strip tool for PE-COFF
    - Built objcopy/strip from GNU binutils - which doesn’t recognize ARM/ARM64 executables
    - Had a small wrapper, which changed the architecture of the objects/executables to x86, ran the GNU tool, and changed it back to the original
    - Added support for PE-COFF in llvm-objcopy in early 2019
- LLVM’s resource compiler didn’t have integrated preprocessing or a GNU windres compatible interface
    - Had a shellscript wrapper, doing option handling and coordinating preprocessing
    - Integrated preprocessing in llvm-rc and added a windres interface in 2021

Improving mingw compatibility

- Support for ”auto-import” and runtime pseudo relocations
  - Allows referencing data in other DLLs without explicit `dllimport` attributes
  - Probably the main mingw specific feature
    - Was required for linking a C++ standard library from a DLL, with the Itanium C++ ABI

Maturing

- The llvm-mingw project has matured a lot since then
  - Most hacks in the build process removed now
    - (but ARM64EC adds new ones)
  - Providing not only cross compilers, but also toolchains that run on Windows, even on ARM and ARM64, since late 2018
  - The MSYS2 package manager now provides ”clang64” environments, which similarly use Clang, LLD, libc++ etc as default toolchain

Releases

- These days - all releases built on free Github Actions
- Nightly builds with latest LLVM, latest mingw-w64
- One release of llvm-mingw for each LLVM release, every 2 weeks

Thank you!

https://github.com/mstorsjo/llvm-mingw