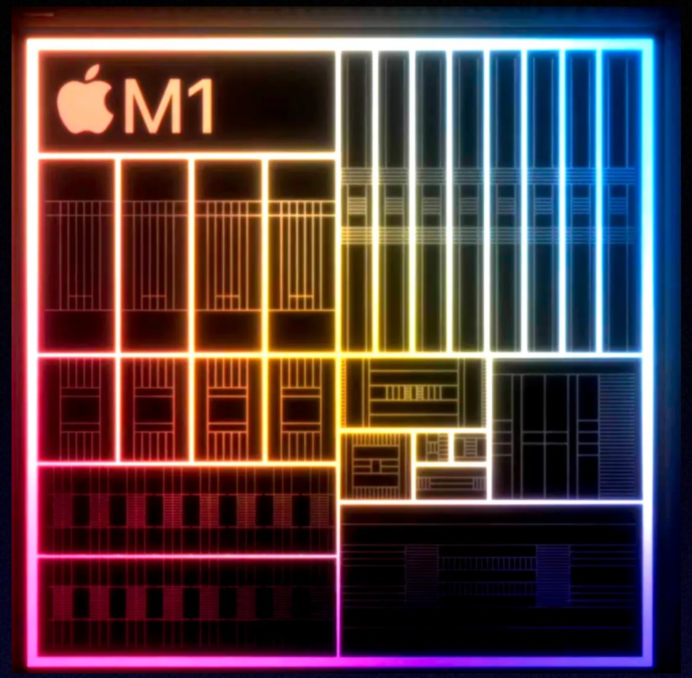
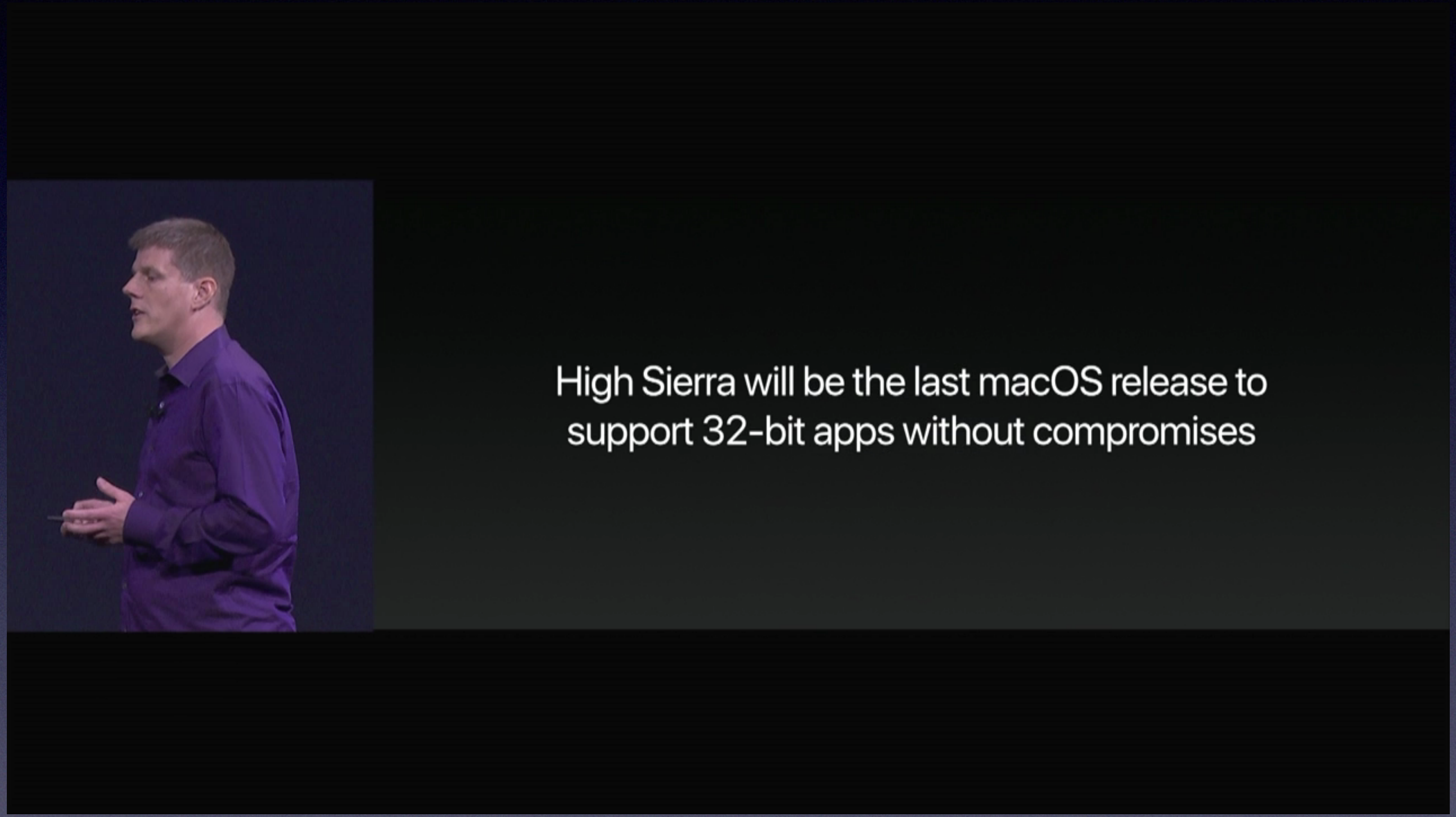


🍷 Wine on macOS 💻
State of the Union

Brendan Shanks
WineConf 2022

CODEWEAVERS
SOFTWARE LIBERATORS

Removal of 32-bit support on macOS



2017 - macOS 10.13 High Sierra

2018 - macOS 10.14 Mojave

CODE WEAVERS
SOFTWARE LIBERATORS

- Wine runs 32-bit (x86) Windows EXEs inside a 32-bit Unix process, and 64-bit (x86_64) EXEs inside a 64-bit Unix process
- 32-bit Windows software still extremely common

In 2019, macOS (10.15 Catalina) will no longer run 32-bit processes

🚨This is a problem!🚨

How to run 32-bit code?

- Early experimentation and prototyping done using Hypervisor.framework
- Apple added support to macOS 10.15 for 64-bit processes to create 32-bit code segments
  - Linux has similar support also

Wine's DLLs

- At the time (Wine 4.x), Wine made up of Winelib DLLs: built as ELF/ Mach-O dylibs, implementing Windows APIs, able to call either Windows or Unix APIs
- Windows code uses 32-bit pointers, calling conventions, struct packing, but Unix is 64-bit. Big mismatch!

The “hybrid” compiler

- Fork of Clang 8, implements special 32-on-64 mode:
  - Pointers have an address space, either 32- or 64-bit
  - Variables, functions also have an address space
  - Address space is inferred based on header files
    - anything from system headers is 64-bit
    - Wine headers had pragma added to mark as 32-bit

The “hybrid” compiler

- Processor must be in 64-bit mode
- For every function using newly-added 32-bit-compatible calling conventions, compiler generates thunks (`wine_thunk_function`):
  - far call from 32- to 64-bit mode
  - call the function
  - far return back to 32-bit mode and the original caller

```
#ifdef __i386_on_x86_64__
#pragma clang default_addr_space(push, default)
#pragma clang storage_addr_space(push, default)
#endif

static void *ft_handle = NULL;   // returned from dlopen()

FT_Error (*pFT_Load_Sfnt_Table)( FT_Face face, FT_ULong tag, FT_Long offset, FT_Byte* buffer, FT_ULong* length );

#ifdef __i386_on_x86_64__
#pragma clang default_addr_space(pop)
#pragma clang storage_addr_space(pop)
#endif
```

__attribute__((stdcall32))

32-bit pointer

```
BOOL WINAPI GetFontFileData( DWORD instance_id, DWORD unknown, UINT64 offset, void *buff, DWORD buff_size )
{
…
    pFT_Load_Sfnt_Table(ft_face, table, offset, buff, &len);
…
}
```

cast to 64-bit pointer

If pointer sizes didn't match, compiler throws an error:

```
wine/dlls/gdi32/freetype.c:1929:19: error: assigning 'void *' to '__storage32 void
                  *__storage32' changes address space of pointer
                    ft_handle = dlopen(SONAME_LIBFREETYPE, RTLD_NOW);
                              ^ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

The “hybrid” compiler

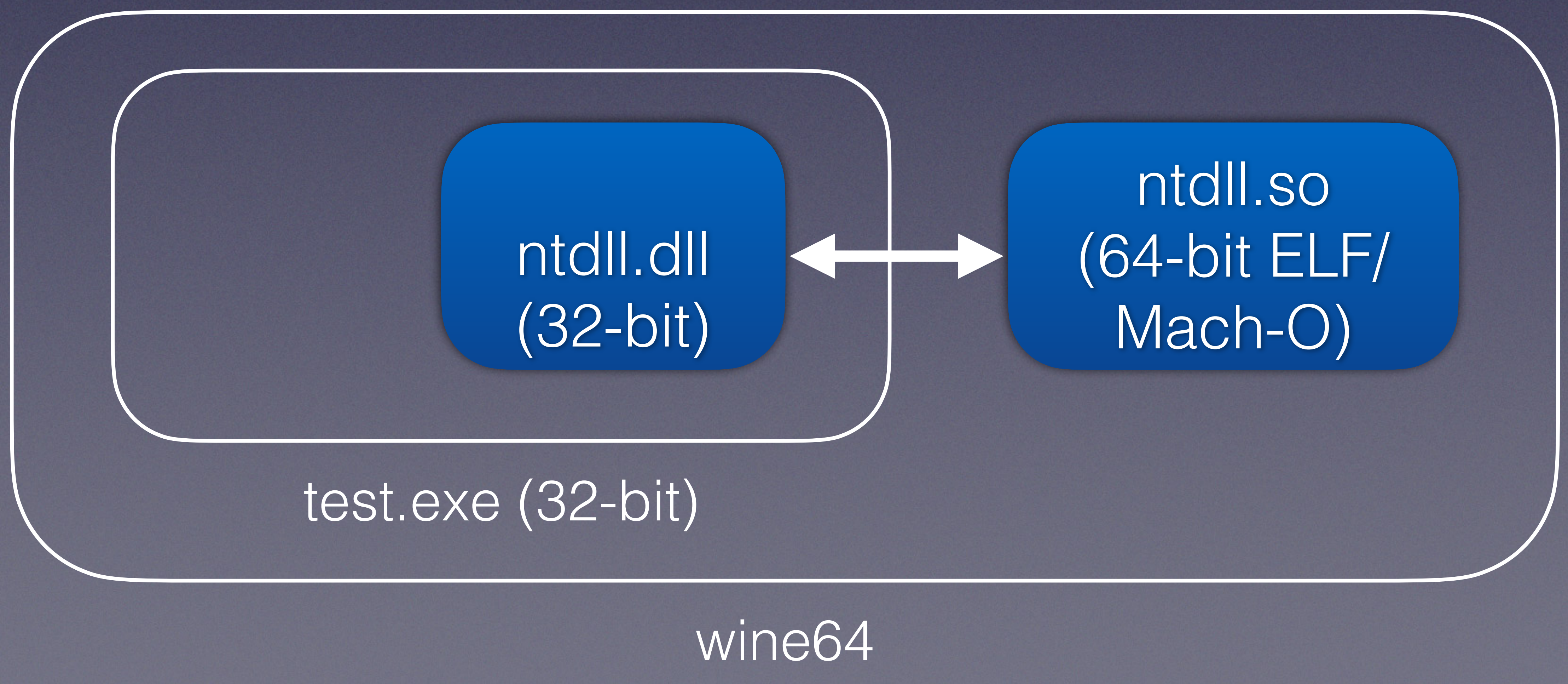
- In practice, worked very well, but still many special cases requiring changes in Wine:
  - Want to pass data coming from Unix library to a Windows function: had to make temporary copy
  - `glMapBuffer`: used `mach_vm_remap` to remap to below 4GB
  - XAudio passes complex structs straight to Unix FAudio: had to marshal structs

The "hybrid" compiler

- Shipped CrossOver 19 (Wine 4.12) in December 2019 🎉
- Continued to use same compiler with minimal changes for CrossOver 20 (Wine 5.0), 21 (6.0), 22 (Wine 7.7)
- Wine changes resulted in massive diff vs. upstream, not merged upstream
- Clang changes also not upstream

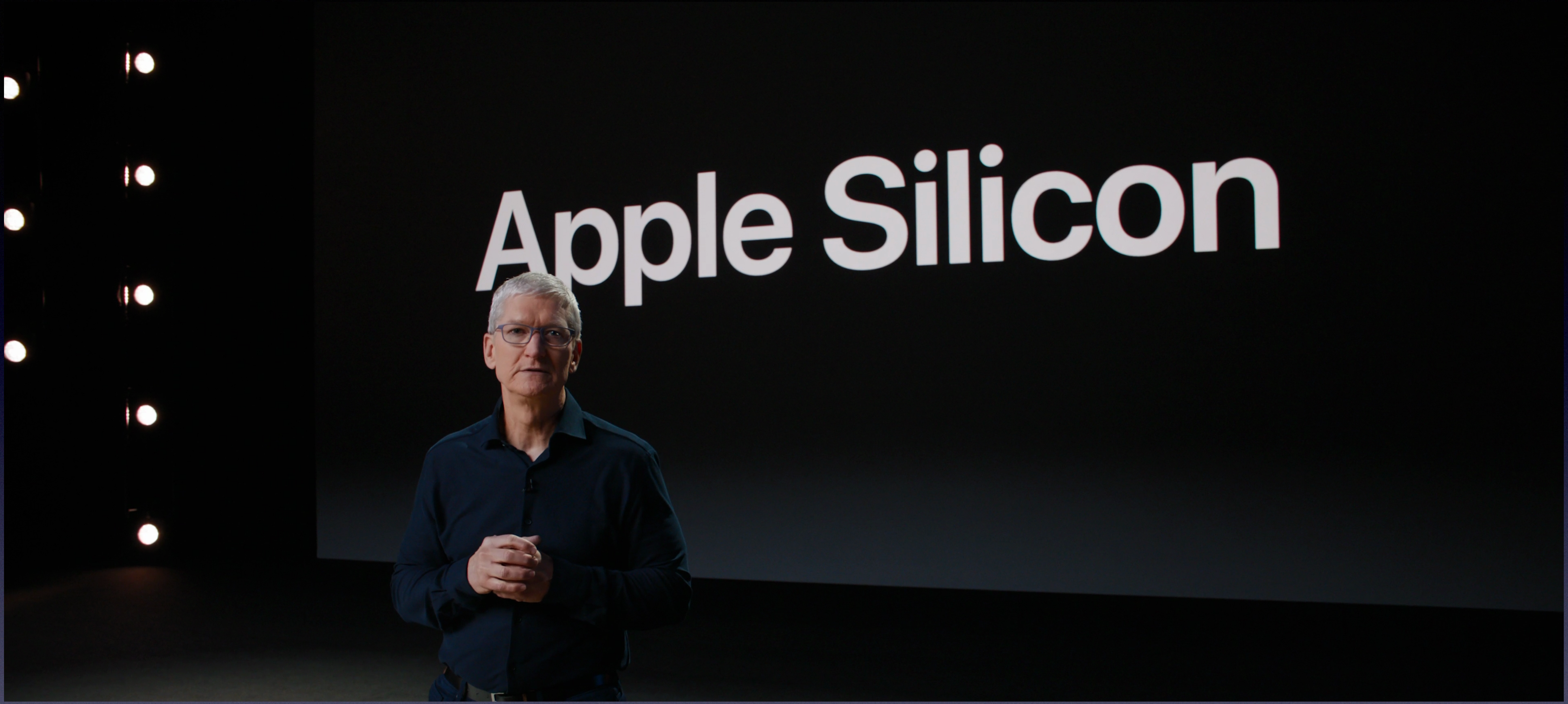
The upstream solution: PE Separation

- Build as much of Wine as possible as PE DLLs, without direct access to Unix APIs
- For Unix API access, use a separate Unix lib without direct access to (non-ntdll) Windows APIs
- Creates a hard boundary between PE DLLs and Unix .so/dylibs
- Allows for PE DLL to be 32-bit and Unix dylib to be 64-bit, with thunks in-between to marshal parameters and structs



The upstream solution: Wow64

- With PE separation, becomes possible to use approach similar to Windows-on-Windows-64 (Wow64) architecture used on Windows for running 32-bit EXEs
- Exists in upstream Wine, partially functional today!
- CrossOver 22 (Wine 7.7) uses both Wow64 and hybrid compiler
  - hybrid compiler only used for 3 DLLs



Apple announces the Mac's transition to ARM64/
"Apple Silicon"

June 2020

Rosetta 2

- macOS included “Rosetta 2”, a translator/emulator for running existing (64-bit) Intel Mac applications
    - Rosetta 2 also supports the 32-bit code segments!
- Also works at command-line, and macOS includes all binaries as “fat” arm64 and x86_64 binaries
    - Building Wine currently must be done from an emulated command-line

pip — zsh — 80×8

```
Last login: Wed Sep 28 16:18:18 on ttys146
pip@Brendans-MacBook-Pro-M1 ~ % uname -m
arm64
pip@Brendans-MacBook-Pro-M1 ~ % arch -x86_64 zsh
pip@Brendans-MacBook-Pro-M1 ~ % uname -m
x86_64
pip@Brendans-MacBook-Pro-M1 ~ %
```

🍷 Wine on Rosetta 2

- Wine needed minimal changes for Rosetta 2:
  - GPU detection in the Mac driver assumed all GPUs were PCI devices
  - Started building Wine with `-mfpmath=sse` to avoid x87 FPU
  - Some preloader changes needed to shift Rosetta's memory allocations
  - SMBIOS table needed to be generated for `GetSystemFirmwareTable()`

CODE WEAVERS
SOFTWARE LIBERATORS

Rosetta 2 Limitations

- x87 floating point performance currently quite slow, exceptions not implemented
- No AVX support
- Not able to detect cross-process code modification through `mach_vm_write`
- Cannot retrieve x86 register state cross-process through Mach calls
- x86 debug registers not really implemented
- Translation is opaque: no logging/debugging for Rosetta itself

🐜🕷 Rosetta 2 Bugs 🪲🦟

- Rosetta team has been very responsive, many bugs fixed in last 2 years
- Finding and identifying bugs can be a challenge!
- `movw` from segment selector to memory would write 32 bits instead of 16, possibly overwriting data
- Race conditions between `SIGUSR1` delivery and modifying segment selectors (`popl %ds`, `ljmpq`)

- Overall, Rosetta works very well
- Excellent performance
- Even with translated CPU, game performance often better on Apple Silicon than on Intel Macs w/integrated graphics

The Future

PE Separation

- 32-bit Windows apps will be able to use Vulkan (especially important for wined3d/DXVK to use MoltenVK)
- Ability to swap %gs register when entering/leaving Windows code

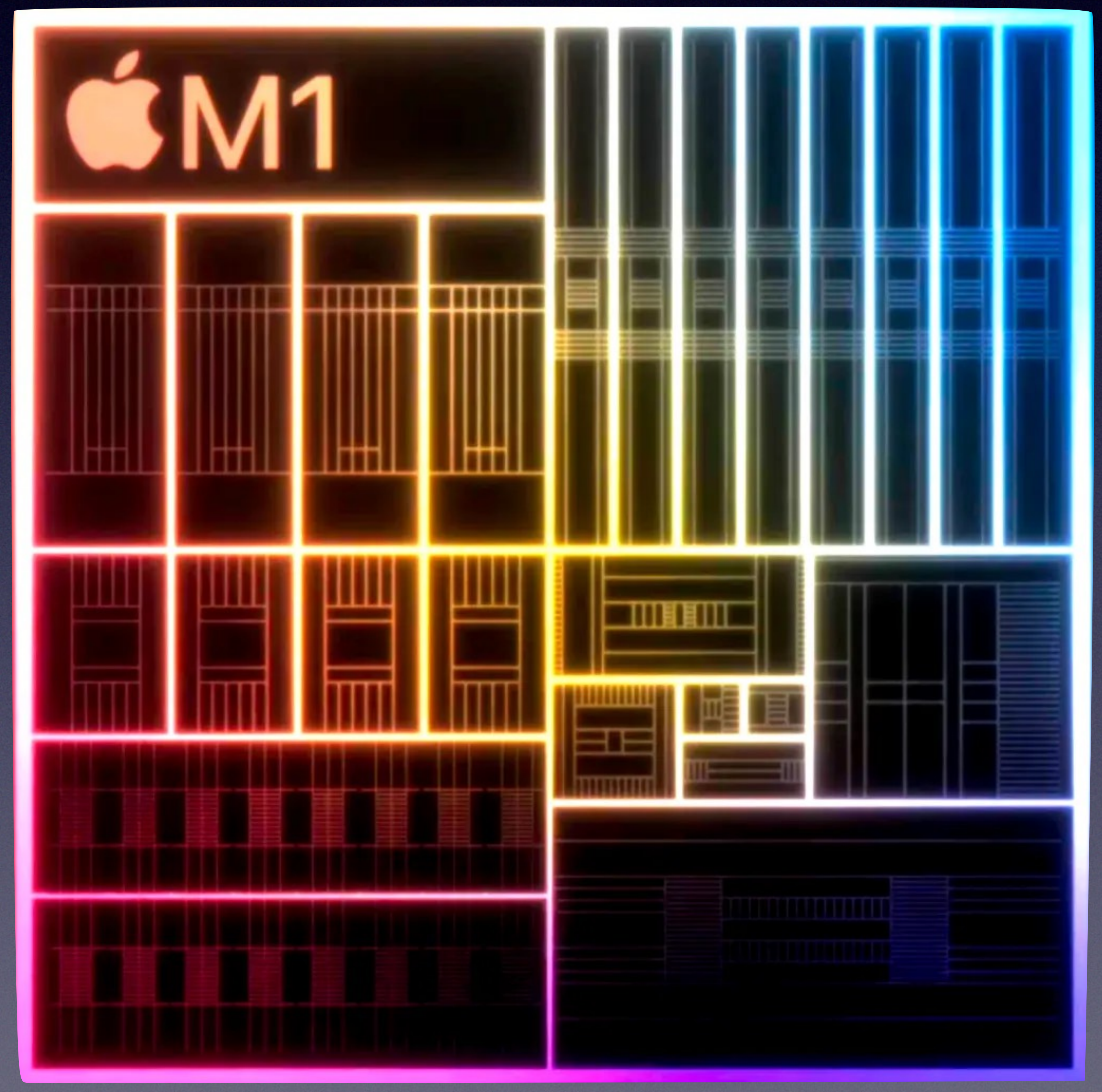
%gs conflict

- Both x86_64 macOS and Windows point %gs to important thread-specific data (macOS TSD, Windows TEB)
- Without a hard boundary between Unix and Windows code, they had to share same %gs (the macOS one)
- Apple reserved %gs:30h (Self) and %gs:58h (ThreadLocalStorage), allows most Windows apps to work

- %gs:60h (ProcessEnvironmentBlock)
  - accessed by apps linked against most versions of the Win10 SDK. Fixed in W11 SDK
  - CrossOver has a hack to make this work
- %gs:8h (StackBase)
  - accessed by Chromium before v87 (when I landed a fix)
  - CrossOver has some load-time binary patches for CEF (Rockstar Launcher, beamNG)
- %gs:20h (FiberData)
  - some games access this, no solution currently
- Hoping that once PE separation is complete, %gs can be swapped when entering/leaving Windows code

ARM64 Wine

- Some basic work done on this in 2020, but not much attention since
- Apple enforces PAGEZERO >= 4GB, prevents USER_SHARED_DATA from being mapped at natural place (0x7FFE0000)
- 16KB page size
- x18 register (used for TEB) is reserved
- Likely useful more for Wow64 than for native ARM64 software



CODEWEAVERS
SOFTWARE LIBERATORS

Questions?