Repo

Bootable applications

An introduction

github.com/michaelo/bootable-applications

Michael Odden / NDC Oslo 2025
me@michaelodden.com

Vibe check

Practical info

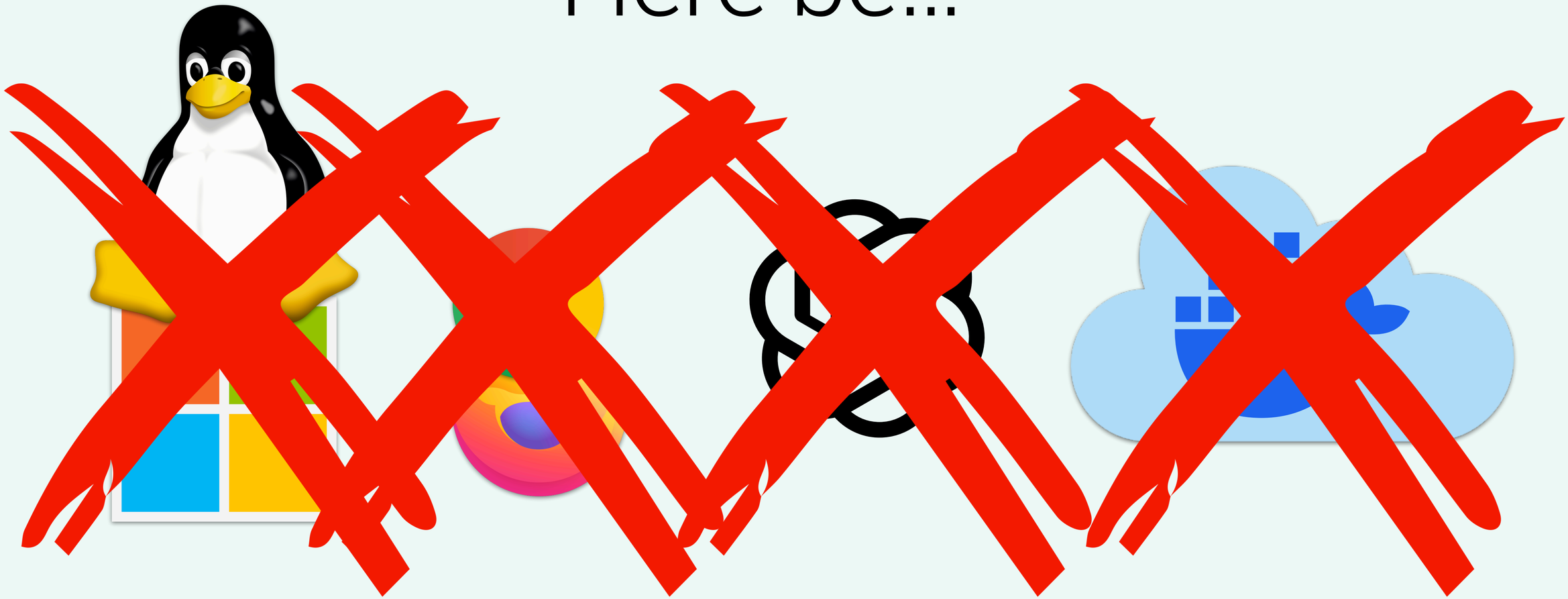
Repo →

github.com/michaelo/bootable-applications

Scope

But... why!?

Here be...



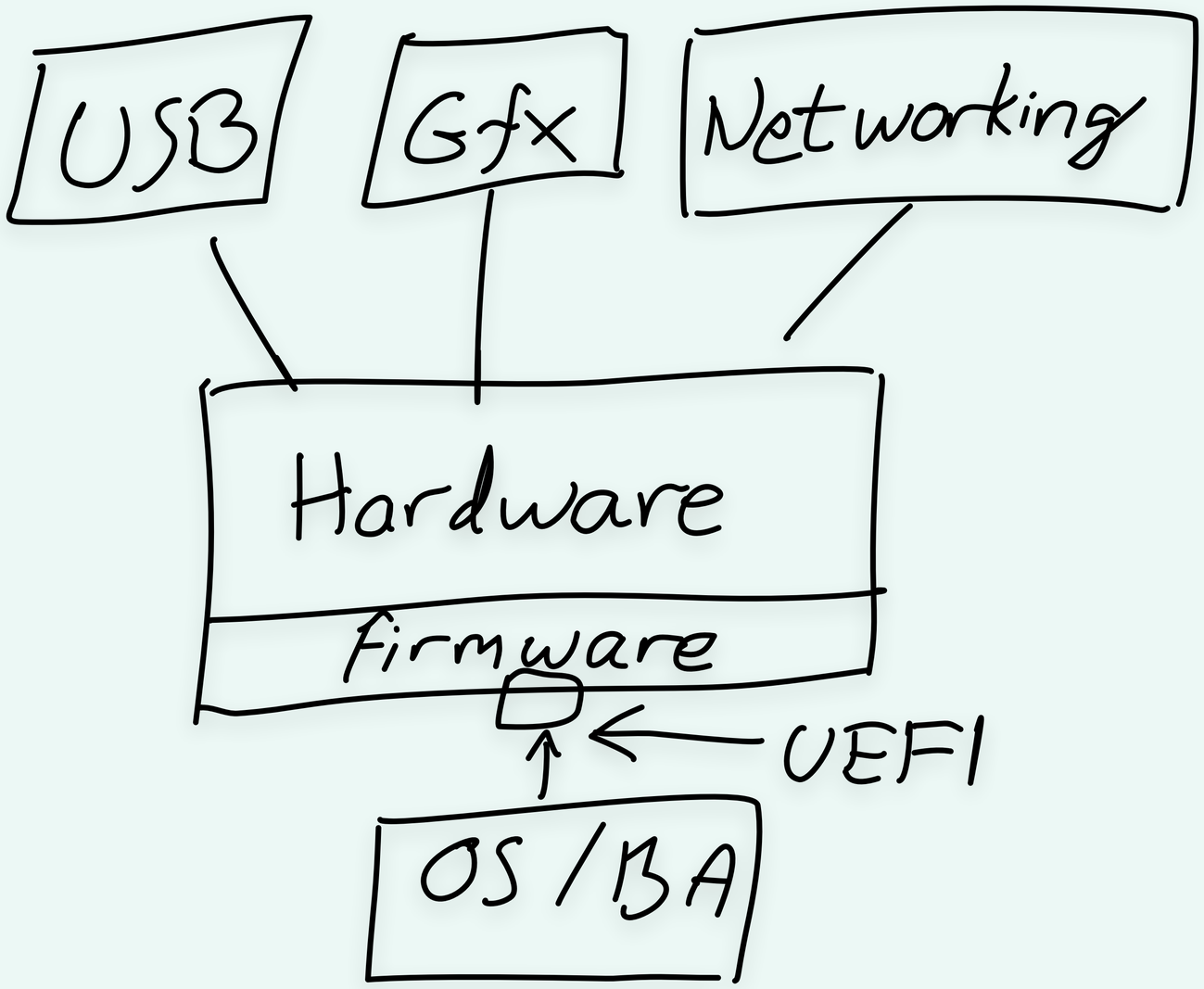
… but there might be dragons
in the code

UEFI?

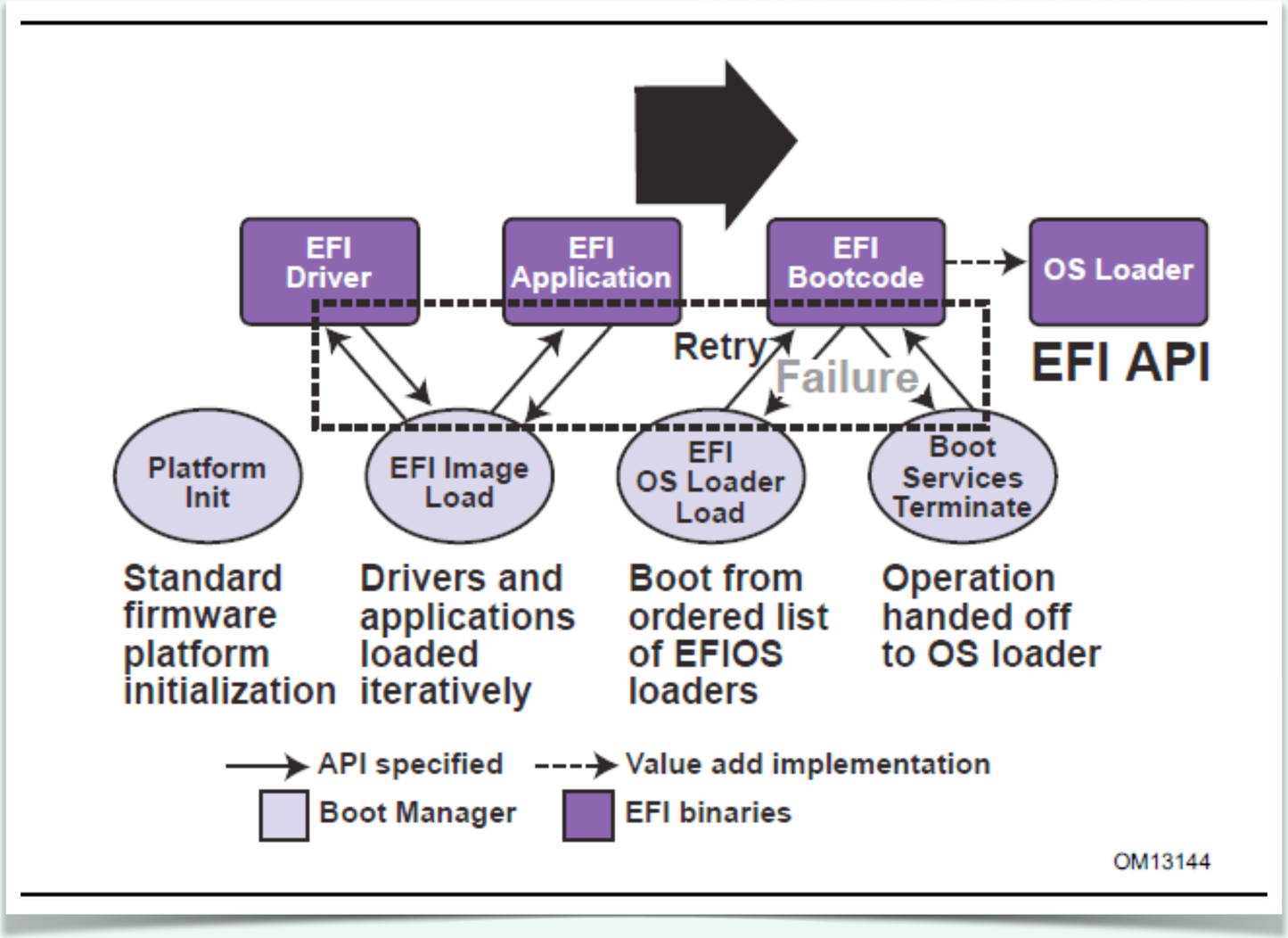
Unified Extensible Firmware Interface

··· Intel - Microsoft - ARM - AMD - Apple ···

··· IBM - nvidia - Cisco - Broadcom ···

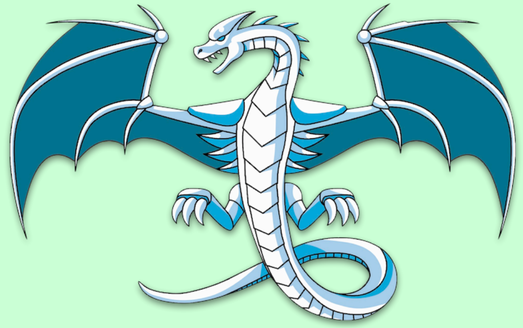

USB
Gfx
Networking
Hardware
Firmware
UEFI
OS / BA

Well defined specification FTW



https://uefi.org/specs/UEFI/2.10/

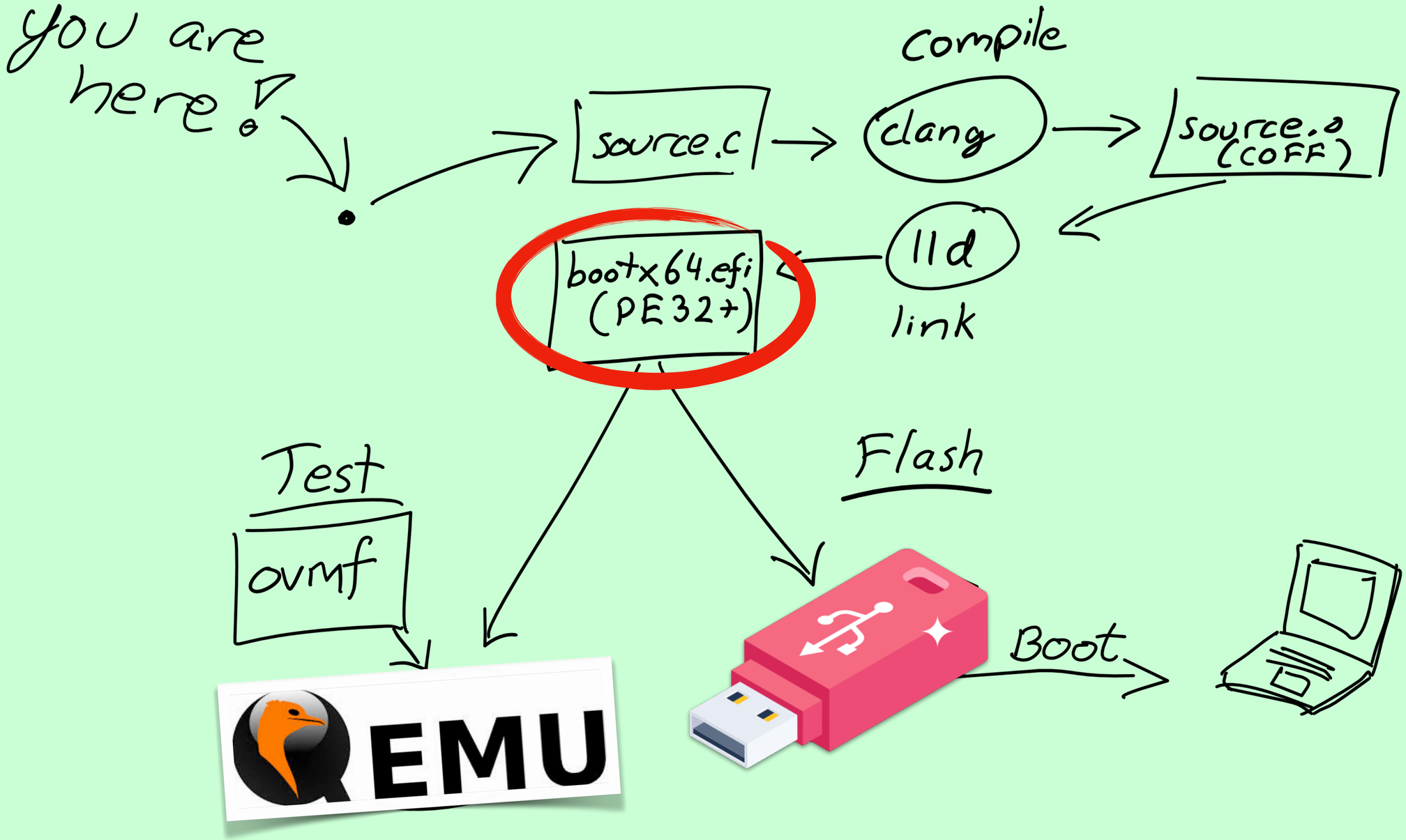
The tools



LLVM

lil_uefi

The process 



Icons: Flaticon

Anatomy of a UEFI-application

- Main function
  - **system_table* - entry point to everything UEFI
    - Console input/output
    - Boot services
      - Query/address protocols, Memory allocation, Event handling, …
    - Runtime services
      - Query runtime information from hardware devices, e.g. system time

examples > hello > C anatomy.c > ...

```c
// Example anatomy of a UEFI-application
#include "lil_uefi/lil_uefi.h"
#include <stddef.h> // for NULL

// Entry point:
// https://uefi.org/specs/UEFI/2.10/04_EFI_System_Table.html#efi-image-entry-
EFI_UINTN EfiMain(EFI_HANDLE handle, EFI_SYSTEM_TABLE *system_table)
{
    EFI_STATUS status = 0;

    // Prevent automatic 5min abort
    system_table->BootServices->SetWatchdogTimer(0, 0, 0, NULL);

    // Locate protocol(s) of choice - e.g graphics output device
    EFI_GUID gfx_out_guid = EFI_GRAPHICS_OUTPUT_PROTOCOL_GUID;
    struct EFI_GRAPHICS_OUTPUT_PROTOCOL *gfx_out_prot;
    status = system_table->BootServices->LocateProtocol(
        &gfx_out_guid,
        0, // The first graphics device
        (void **)&gfx_out_prot);

    if (!status)
    {
        return status;
    }

    // Access the console output
    EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL *out = system_table->ConOut;
    out->ClearScreen(out);

    out->SetCursorPosition(out, 0, 0);
    out->OutputString(out, L"Hello, world!");

    return status;
}
```

Yes. It’s pointers and structs all the way down

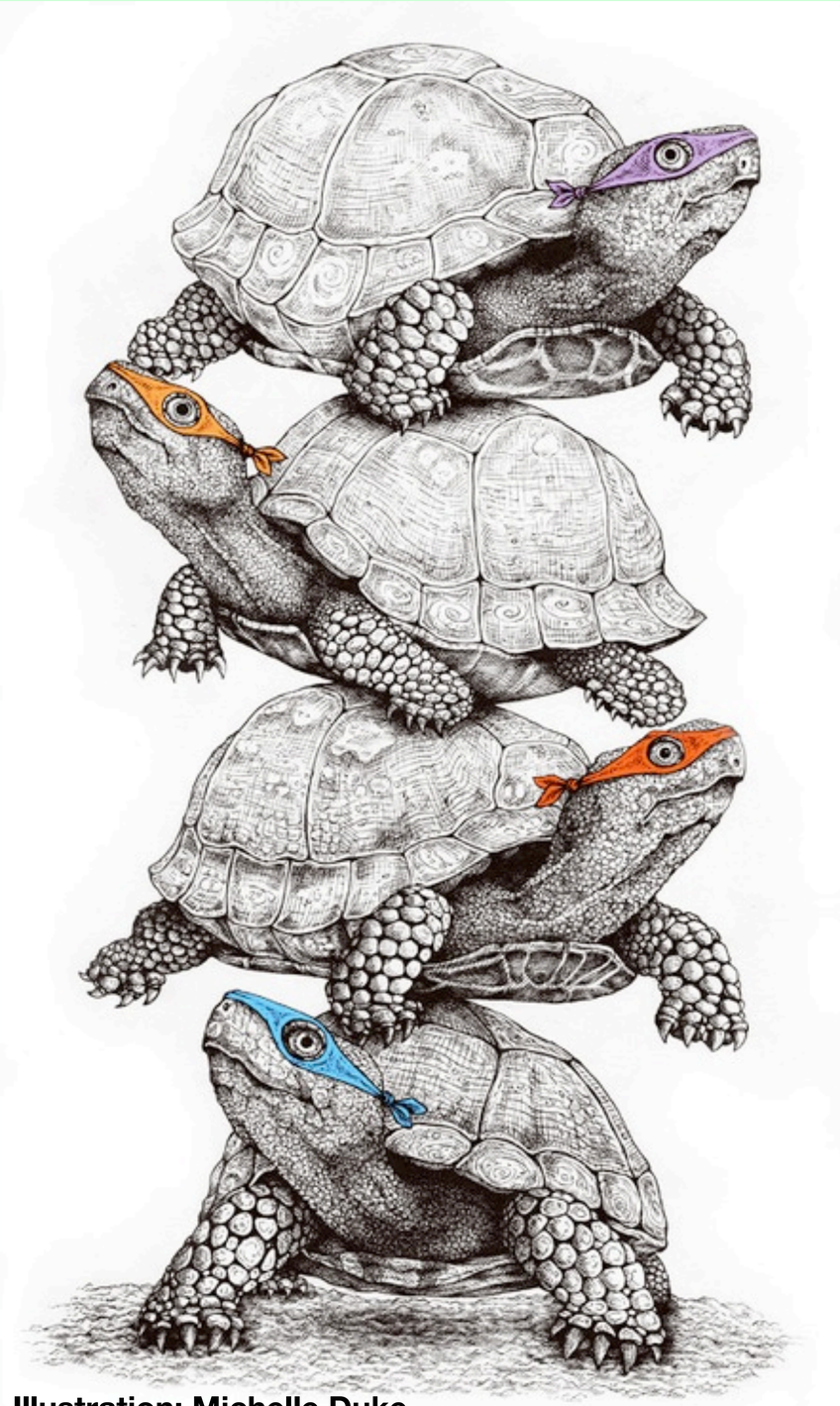


Illustration: Michelle Duke

It’s demo-time!

Let’s dive in…

Couple of tips / nice to knows

- PC-users: Disable secure boot
- There’s no “standard library” by default
- For long-running apps: disable automatic reboot watchdog (see hello.c)
- Not all systems are born equal:
  - The exact behaviour and capabilities available might differ from system to system. E.g. qemu and some PCs will render console output to the screen, whereas MacBook Pros will not.
  - qemu w/ OVMF has no pointer device drivers loaded at boot time, whereas all PCs and MBPs I’ve tested do.

Further reading

- **UEFI specification**
  https://uefi.org/specifications
- **lil_uefi by Allen Webster and Ryan Fleury**
  https://handmade.network/p/308/lil-uefi/
- **osdev.org**
  http://wiki.osdev.org/UEFI
- **TianoCore / EDK II - “EFI Development Kit” - Reference-impl by Intel**
  https://www.tianocore.org/ / https://github.com/tianocore/edk2
  OVMF is borrowed from this
- **uefi-simple**
  https://github.com/pbatard/uefi-simple

Questions?

Now go do something ~~good~~ fun!