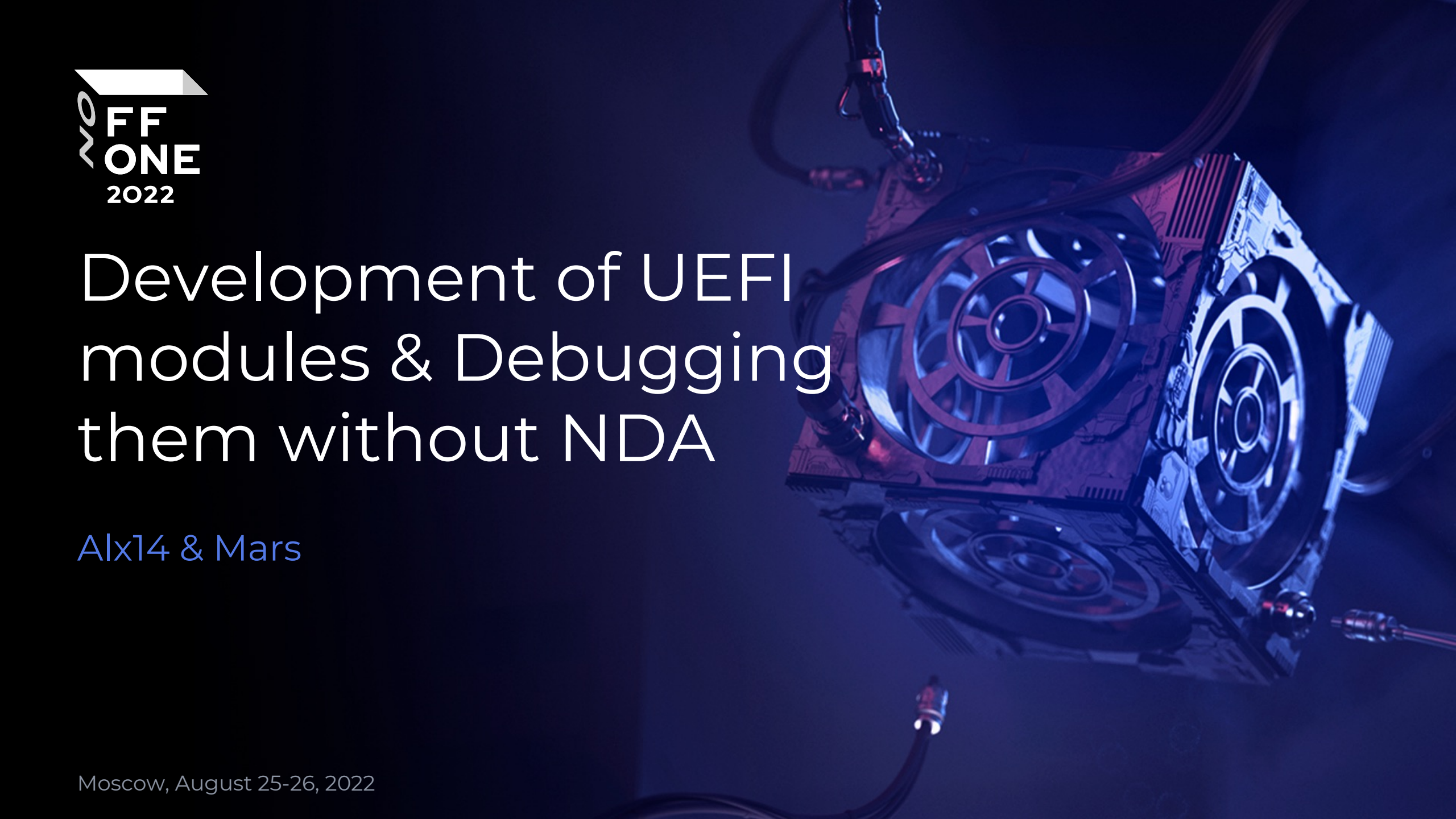


Development of UEFI modules & Debugging them without NDA

Alx14 & Mars

Moscow, August 25-26, 2022

About Us

ALX14

- Independent Researcher
- UEFI, FemtoCells, IOS Researcher
- https://t.me/alx14
- https://twitter.com/alx_pwn

Mars

- Independent Investigator
- Penetration testing of WEB-applications, UEFI Researcher
- https://t.me/lostinffuf

Agenda

- ❑ What is it? UEFI
- ❑ Type of Solutions
- ❑ With Proprietary Blobs
- ❑ Full Opensource
- ❑ Coreboot
- ❑ Libreboot
- ❑ TianoCore EDKII
- ❑ LinuxBoot, NERF, HEADS?
- ❑ JTAG
- ❑ Types of Debugging UEFI
- ❑ OpenOCD
- ❑ Intel System Studio
- ❑ Hardware for debug UEFI (Free)
- ❑ Intel Galileo with OpenOCD
- ❑ Hardware for debug UEFI (NDA)

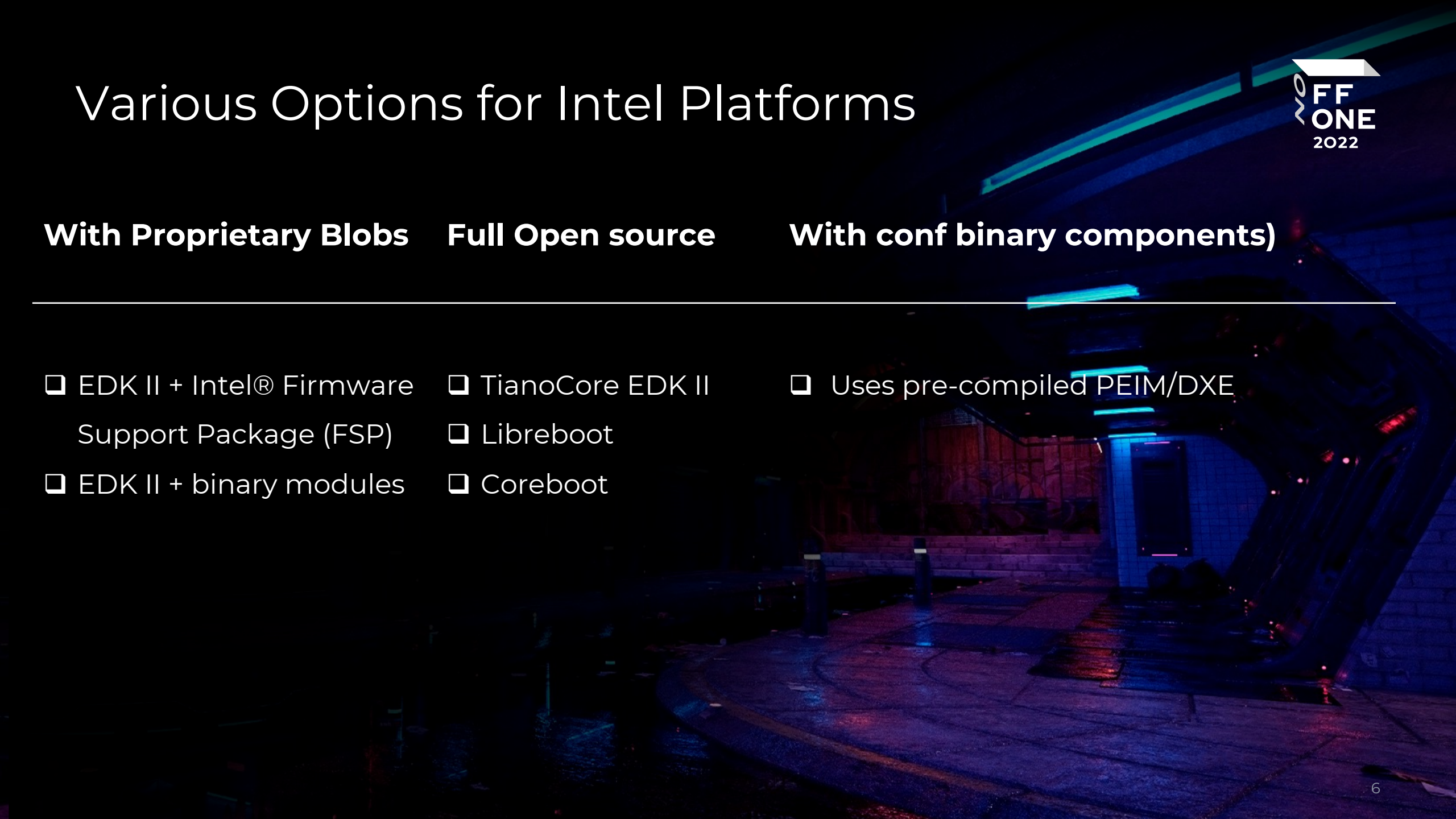
- ❑ Intel JTAGs
- ❑ Intel ITP-XDP
- ❑ Intro to Closed Chassis Debugging
- ❑ Intel® DCI
- ❑ Intel® CCA And USB Debug
- ❑ Variants of debugging
- ❑ What JTAG on x86 can be useful for?
- ❑ Setup environment for EDKII
- ❑ Build EDKII for Quark Platform
- ❑ FLASH Update using CH341a
- ❑ Connect SPI to CH341a
- ❑ Connect UART to Galileo
- ❑ Test UEFI Shell
- ❑ UEFI DEBUGGER OVERVIEW
- ❑ Host & Target Debug Setup
- ❑ Debugging UEFI Firmware using Intel® UDK w/ GDB

What is it? UEFI

What is it

- (Unified) Extensible Firmware Interface
- Modern industrial standard for x86 firmware
- Initially developed by Intel as BIOS replacement for IA64
- Performs HW initialization required to start an OS
- Modular and feature rich, uses well defined and known formats
- Mostly written in C, much easier to develop as legacy BIOS

Various Options for Intel Platforms



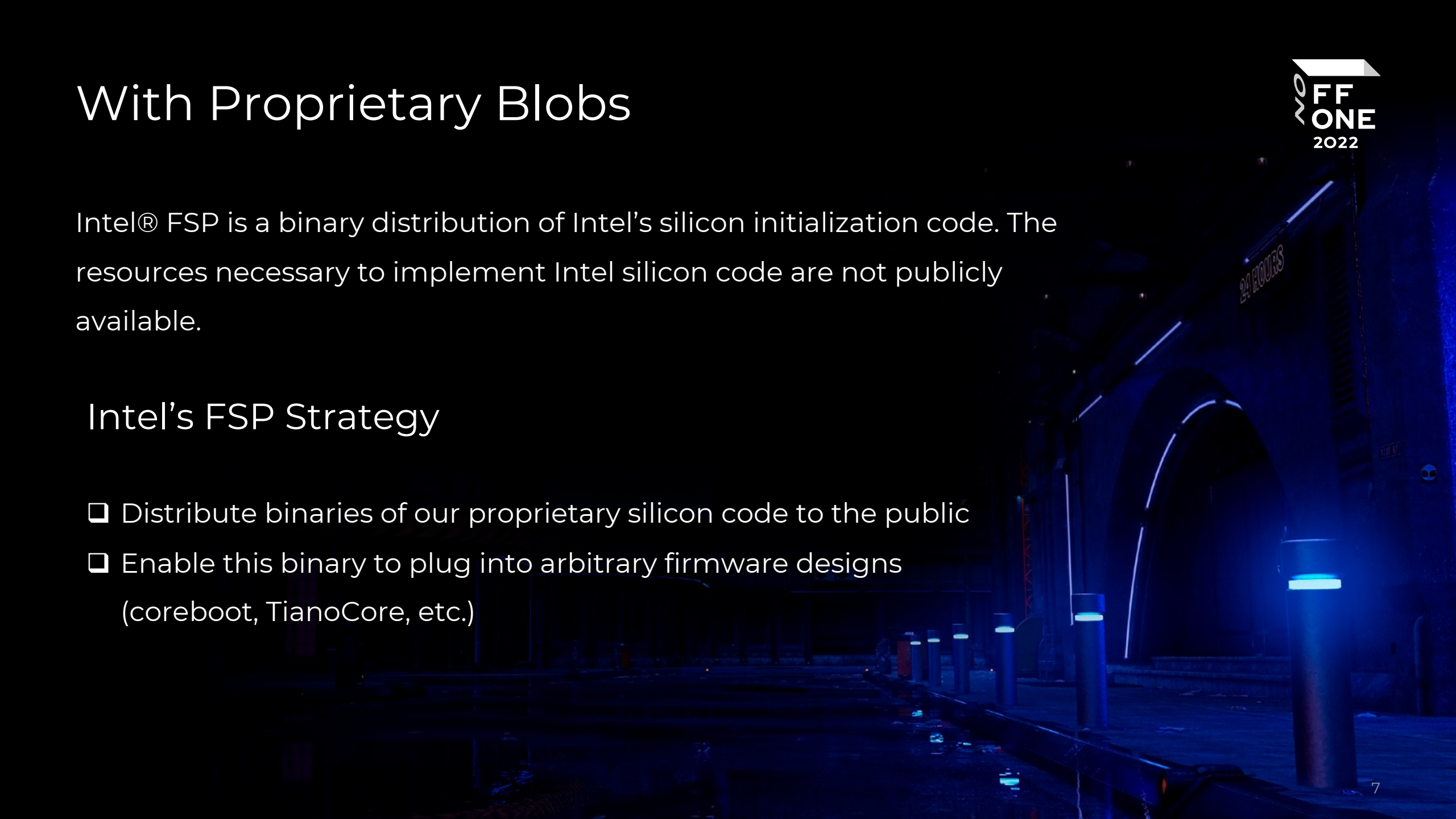
With Proprietary Blobs	Full Open source	With conf binary components)
❑ EDK II + Intel® Firmware Support Package (FSP)	❑ TianoCore EDK II	❑ Uses pre-compiled PEIM/DXE
❑ EDK II + binary modules	❑ Libreboot	
	❑ Coreboot	

With Proprietary Blobs

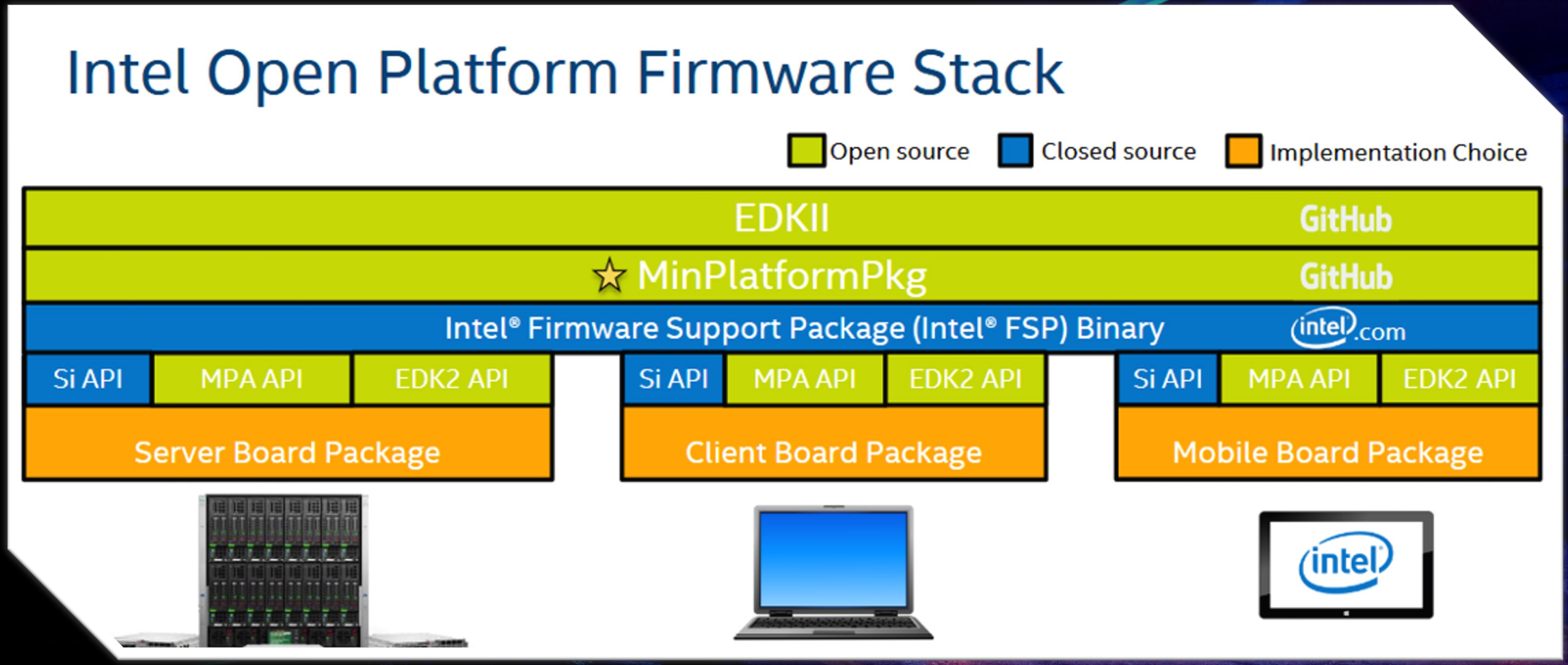
Intel® FSP is a binary distribution of Intel's silicon initialization code. The resources necessary to implement Intel silicon code are not publicly available.

Intel's FSP Strategy

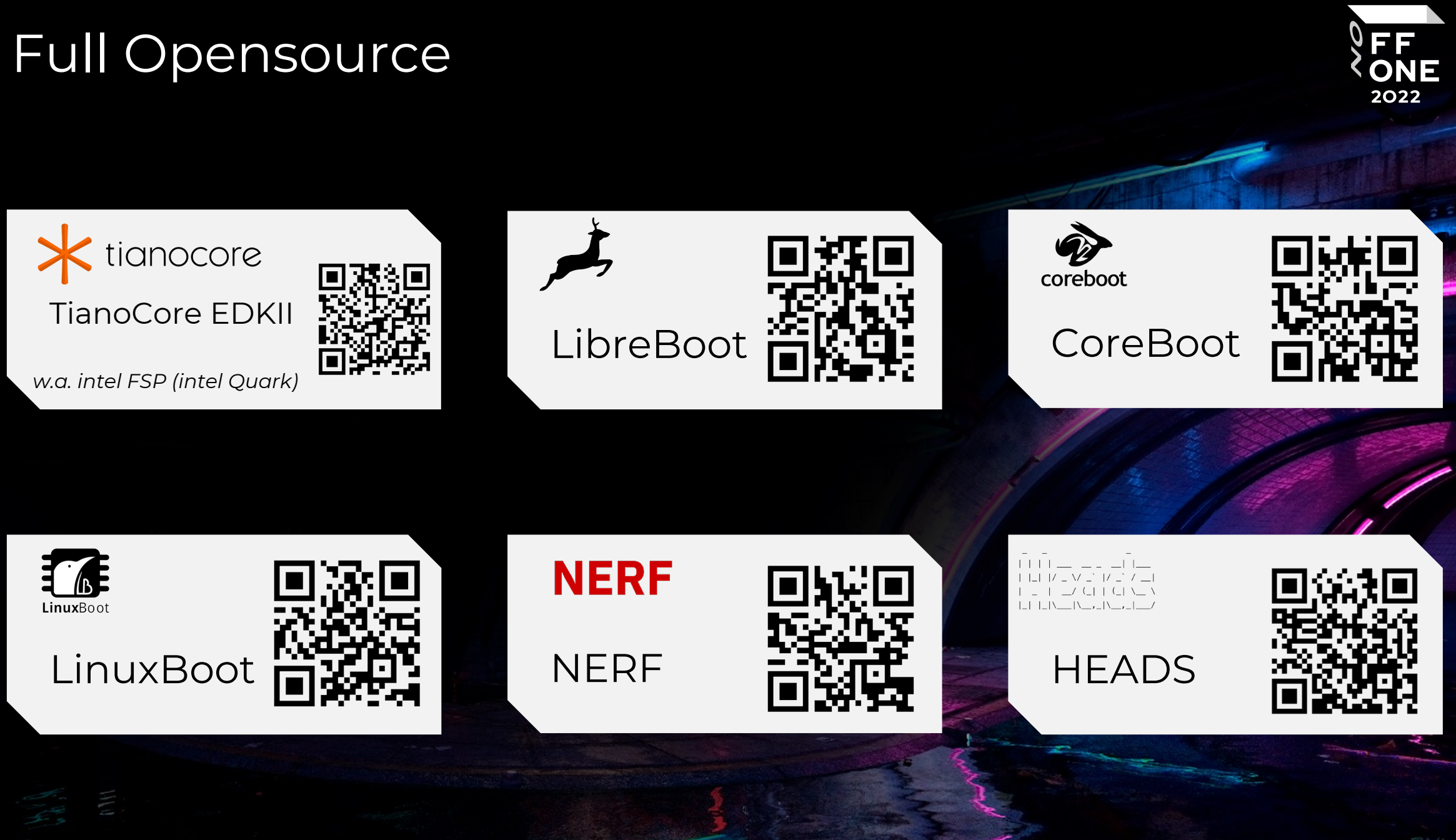
- ❑ Distribute binaries of our proprietary silicon code to the public
- ❑ Enable this binary to plug into arbitrary firmware designs (coreboot, TianoCore, etc.)



With Proprietary Blobs



Full Opensource



Coreboot

- Open source bootable firmware development project for a variety of architectures. The philosophy is to do the bare minimum to keep the equipment running and then hand over control to another program called the payload.
- Intel Management Engine (ME) firmware — we disable and effectively neutralize this blob by removing most of its code and setting flags to disable the ME coprocessor at boot time.
- Completely transfers control of the payload, with no part remaining resident in the system or even available for callback. Uses a very minimal interface to the payload and otherwise imposes no standards on the ecosystem.

Libreboot

- Known briefly as GNU Libreboot - is a free software project which aims to replace the proprietary firmware on most free, lightweight computers and provides documentation aimed at ordinary users.
- It is installed as a coreboot distribution without proprietary binary code.
- It is not a direct fork of coreboot; instead, it is a parallel effort that works closely together and re-base from time to time, created by changing the current code base in a similar way to the Linux-Libre project, which does a cleanup of the Linux kernel of proprietary components.

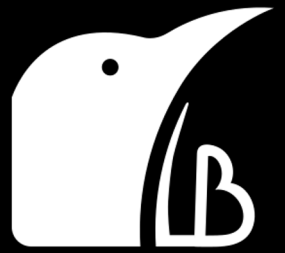
TianoCore EDKII

Tianocore EDK II is the reference implementation of UEFI by Intel. EDK is the abbreviation for EFI Development Kit and is developed by the TianoCore community. TianoCore EDK II is the defacto standard generic UEFI services implementation.

** For Intel CPUs need intel FSP*

LinuxBoot, NERF, HEADS? What's What?

LinuxBoot, NERF, HEADS? What's What?



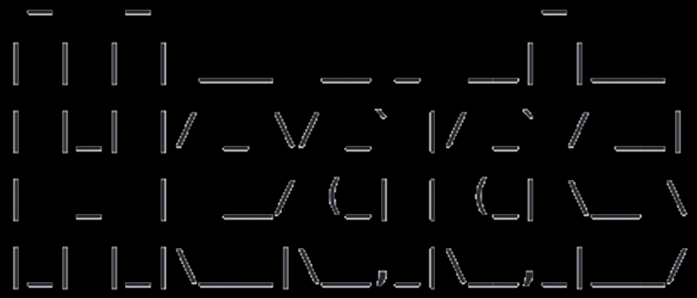
LinuxBoot

is the project that replaces specific firmware functionality with a Linux kernel. LinuxBoot is agnostic to what initramfs is used with the kernel.

NERF

NERF

is LinuxBoot with u-root as the initramfs. u-root contains boot policy tools in Go (e.g. PXE booting, booting via GRUB config) among standard busybox-like utilities rewritten in Go.



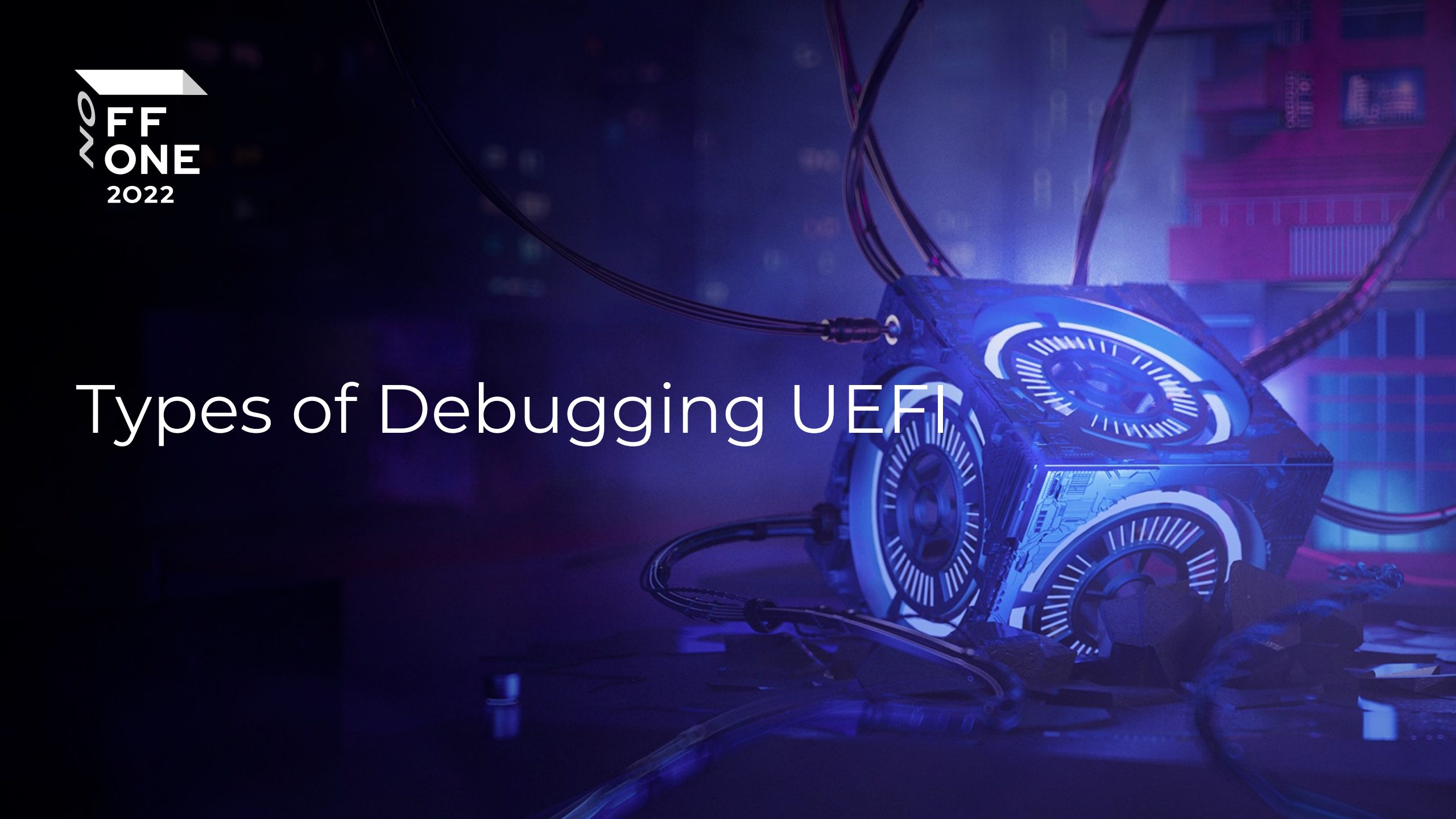
HEADS

is a secure runtime that can be used as the initramfs for LinuxBoot. Take a look at the repository on GitHub. See osresearch.net for more documentation on HEADS.

JTAG

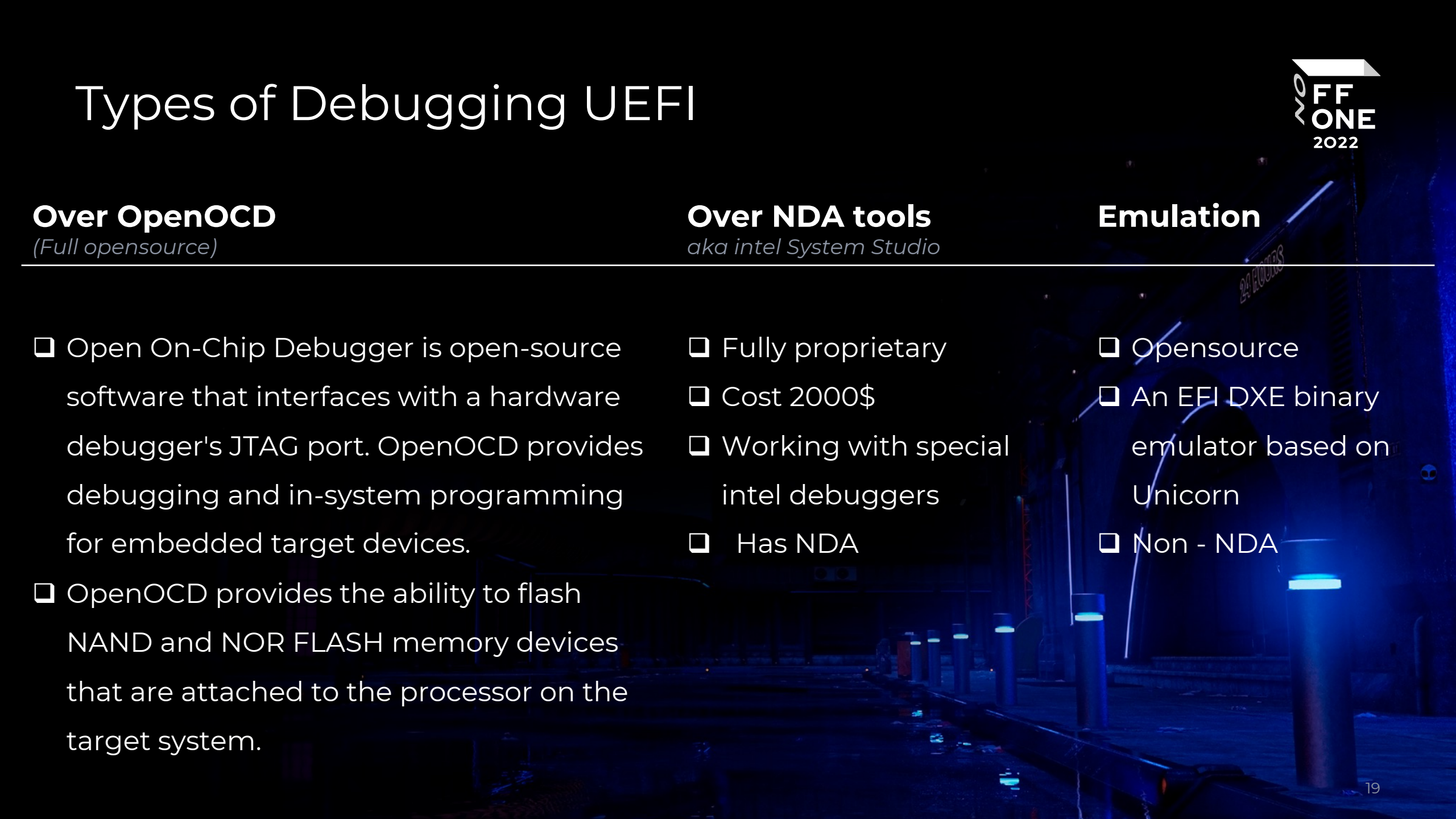
JTAG

- Joint Test Action Group IEEE 1149.1
- Is an industry standard for verifying designs and testing printed circuit boards after manufacture. JTAG implements standards for on-chip instrumentation in electronic design automation (EDA) as a complementary tool to digital simulation.
- IEEE Standard 1149.1
  https://standards.ieee.org/findstds/standard/1149.1-2013.html



Types of Debugging UEFI

Types of Debugging UEFI



Over OpenOCD
(Full opensource)

- Open On-Chip Debugger is open-source software that interfaces with a hardware debugger's JTAG port. OpenOCD provides debugging and in-system programming for embedded target devices.
- OpenOCD provides the ability to flash NAND and NOR FLASH memory devices that are attached to the processor on the target system.

Over NDA tools
aka intel System Studio

- Fully proprietary
- Cost 2000$
- Working with special intel debuggers
- Has NDA

Emulation

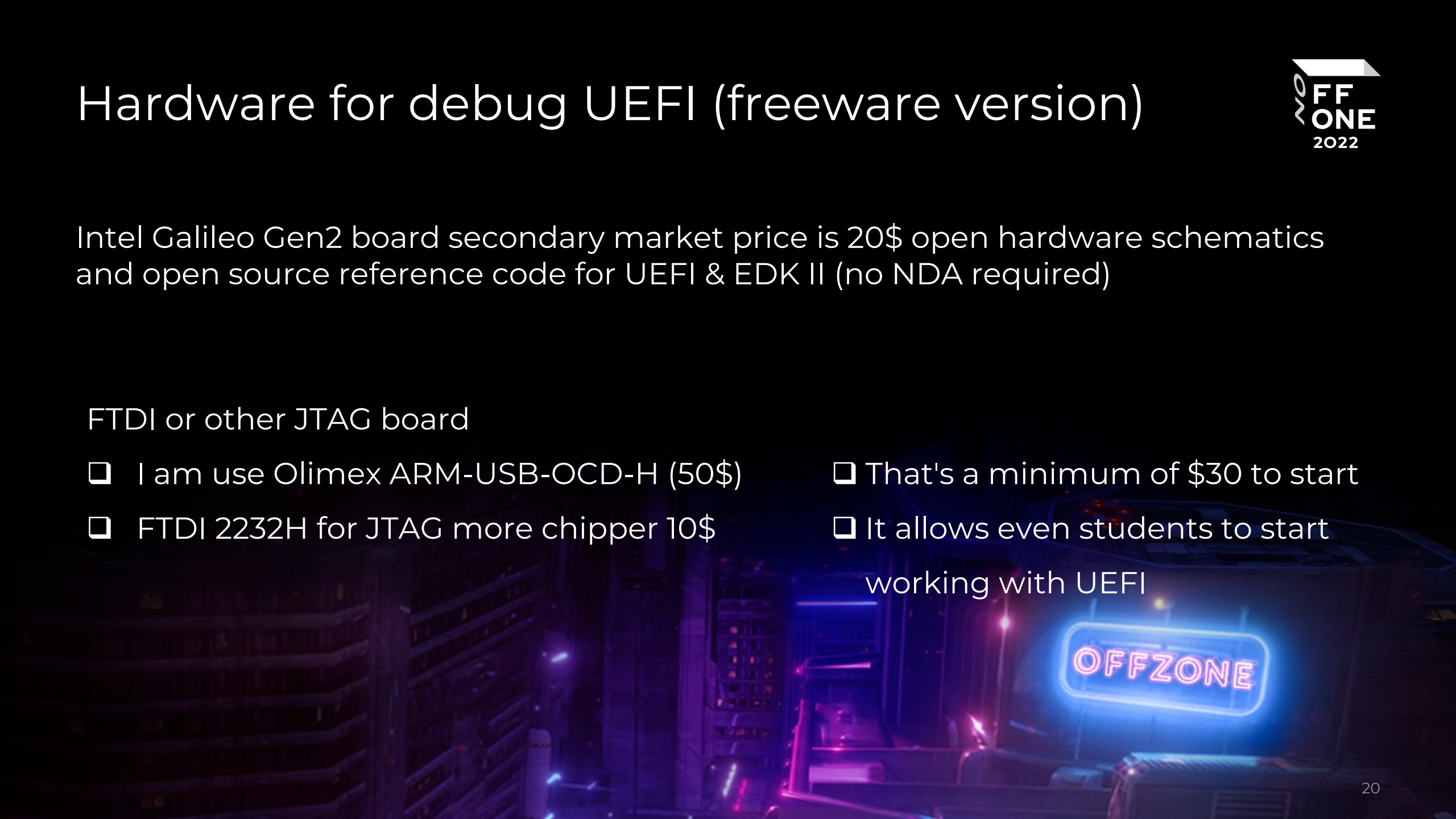
- Opensource
- An EFI DXE binary emulator based on Unicorn
- Non - NDA

Hardware for debug UEFI (freeware version)

Intel Galileo Gen2 board secondary market price is 20$ open hardware schematics and open source reference code for UEFI & EDK II (no NDA required)

FTDI or other JTAG board

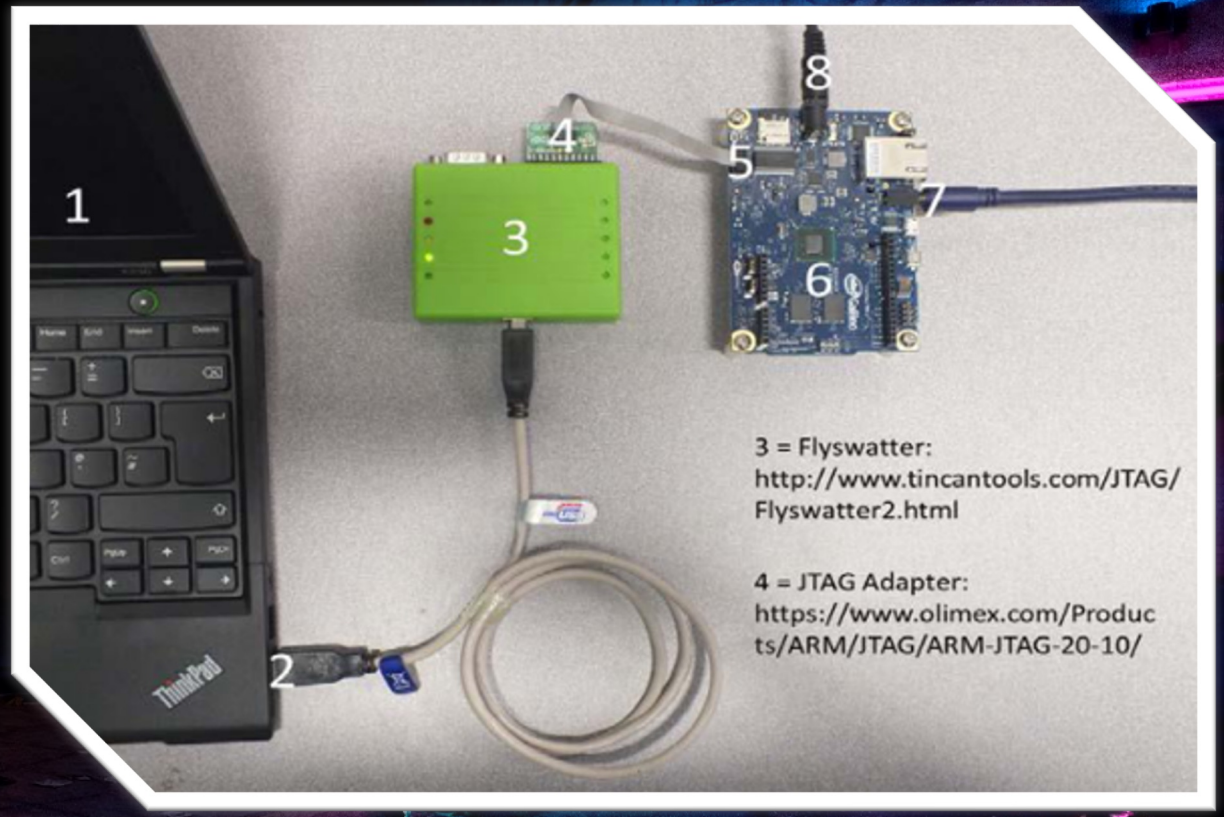
- ❑ I am use Olimex ARM-USB-OCD-H (50$)
- ❑ FTDI 2232H for JTAG more chipper 10$
- ❑ That's a minimum of $30 to start
- ❑ It allows even students to start working with UEFI



Intel Galileo with OpenOCD

Recommended setup for debugging

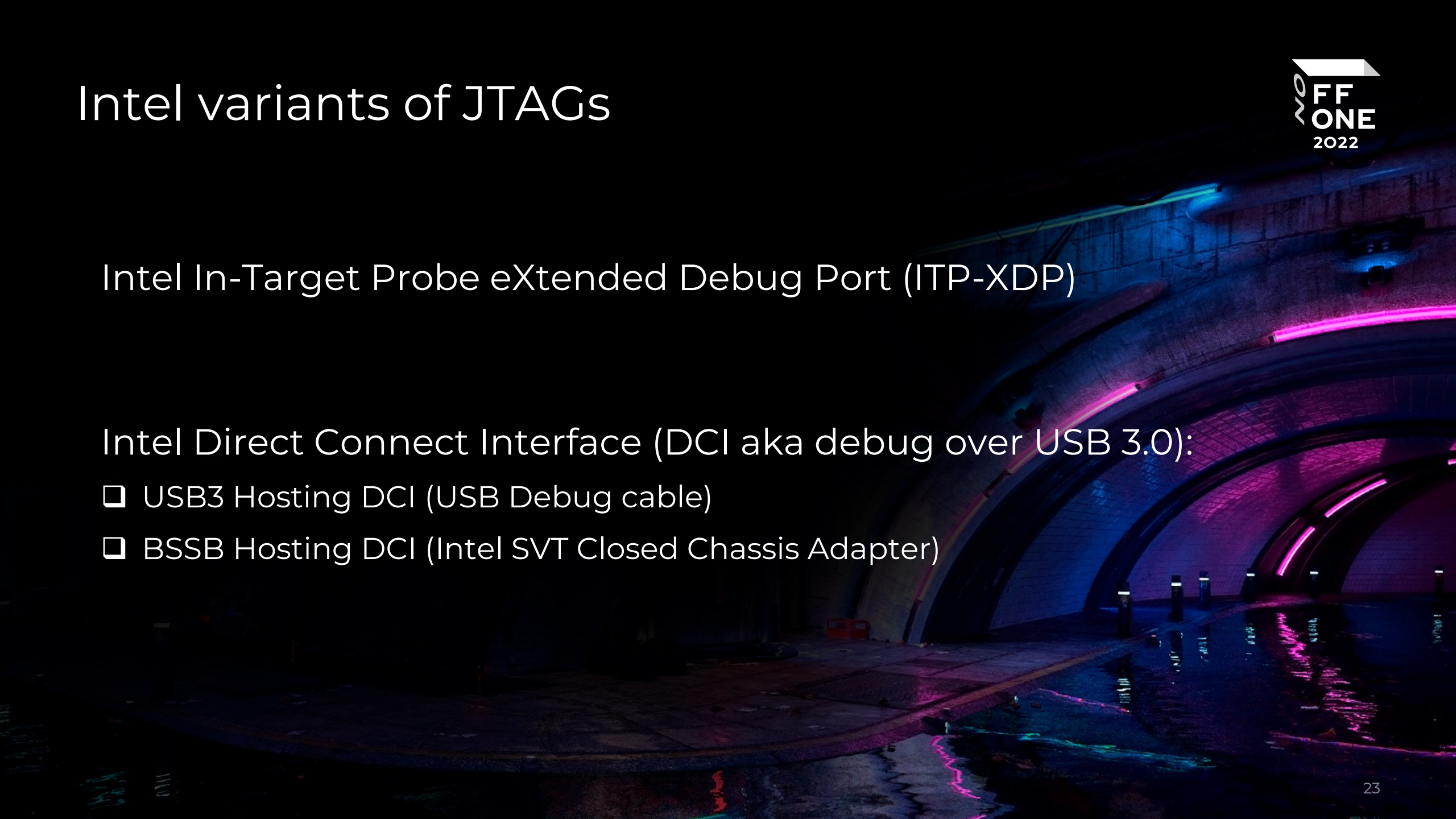
- ❑ Host System
- ❑ USB 2.0 male-male A-B cable
- ❑ JTAG Probe
- ❑ ARM-JTAG-20-10 Adapter
- ❑ JTAG Port
- ❑ Intel® Galileo Board
- ❑ Serial Cable to view boot process
- ❑ Power Supply



Hardware for debug UEFI (NDA version)

- Signing NDA
- Buying Intel System Studio - 2000$ or 30 day trial
- Buying Compatible Board for Debug (minnow board) 400$
- Buying intel Bluebox – 3000$
  - Intel ITP-XDP on ebay 370-1000$
  - Intel SVT not found on public
- That's a minimum of $2,700 to start

Intel variants of JTAGs

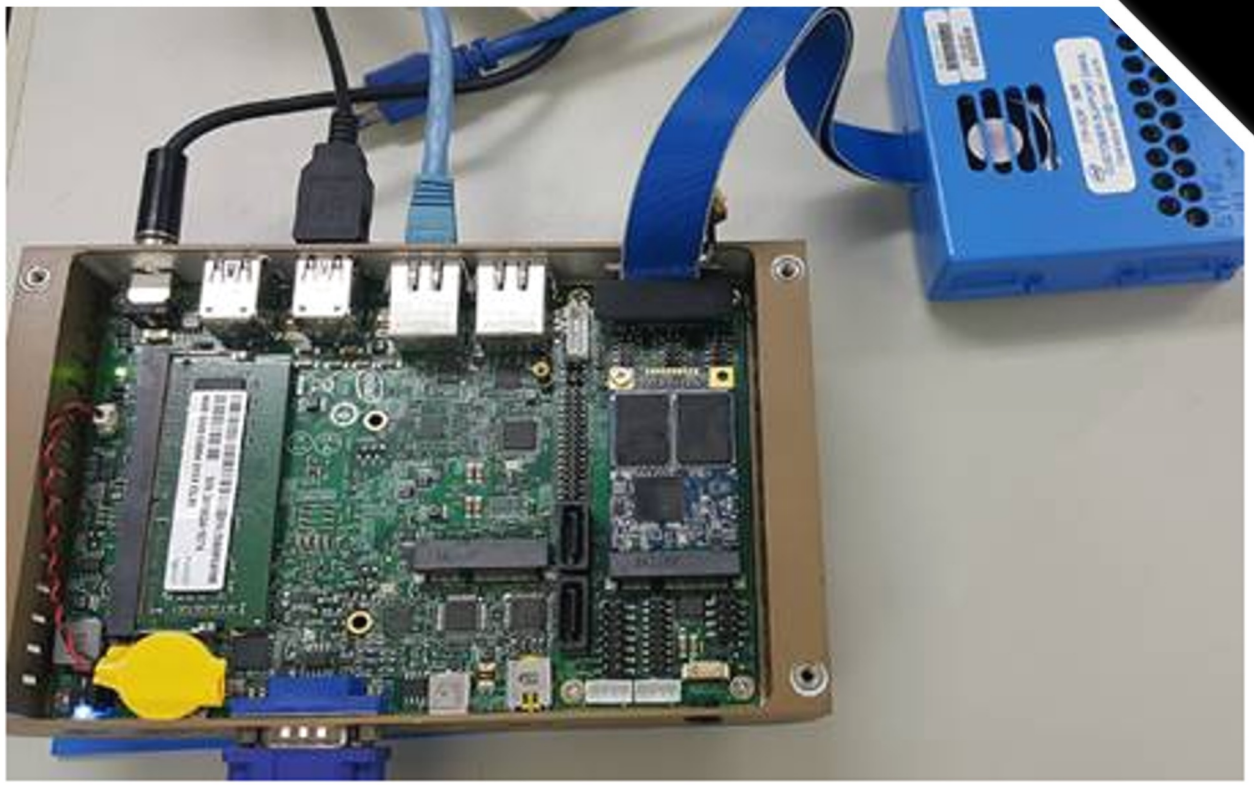


Intel In-Target Probe eXtended Debug Port (ITP-XDP)

Intel Direct Connect Interface (DCI aka debug over USB 3.0):

- USB3 Hosting DCI (USB Debug cable)
- BSSB Hosting DCI (Intel SVT Closed Chassis Adapter)

Intel ITP-XDP

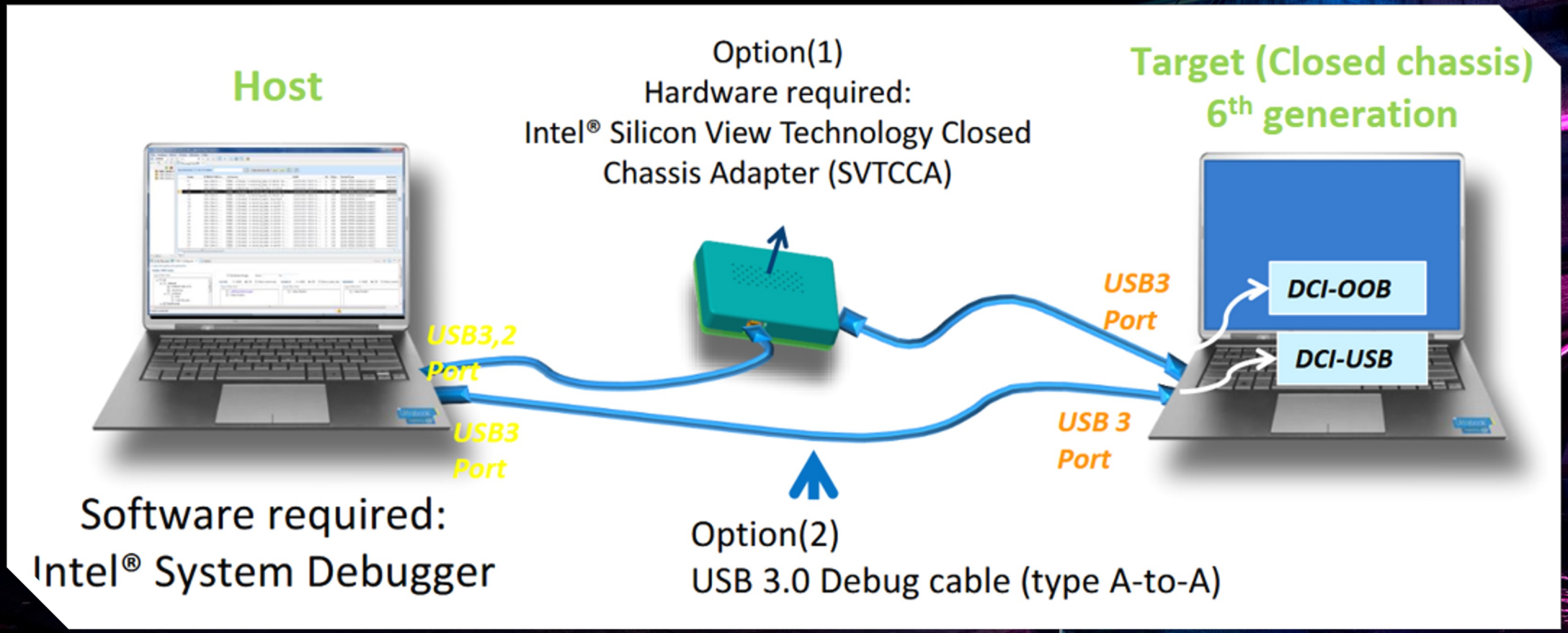


Figure

1: Cable connections between

ITP-XDP3 probe and the Sasby Island target

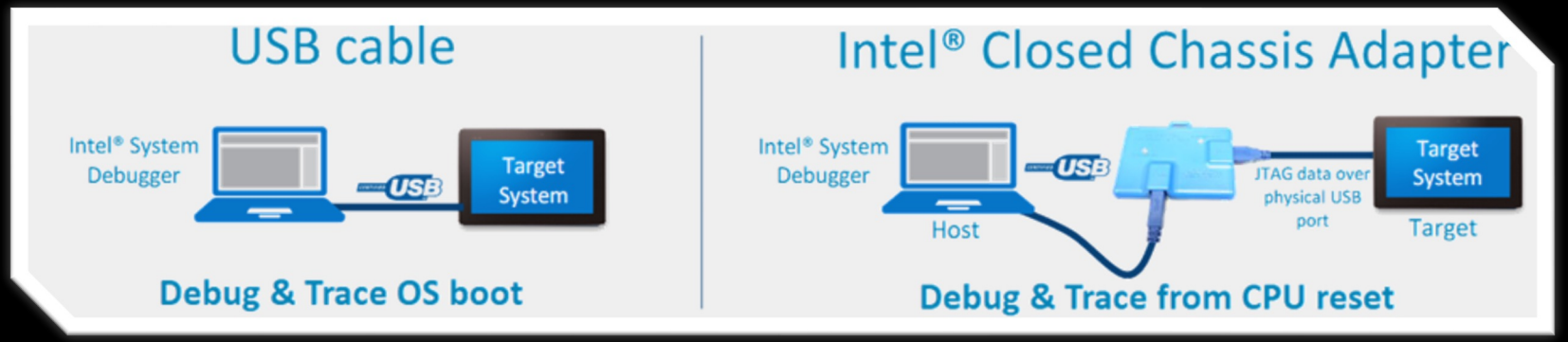
Intel Direct Connect Interface Overview



Intro to Closed Chassis Debugging

JTAG*-based system debug and system trace over low-cost USB* connections Intro to Closed Chassis Debugging

- Works on production hardware instead of expensive engineering samples
- Debug over a standard USB* connection instead of expensive JTAG probes
- Supports debugging Unified Extensible Firmware Interface (UEFI) platform firmware
- Developers no longer need to wait for a turn at a shared JTAG debugging station
- Design flexibility alleviates the requirement for an accessible hardware JTAG port

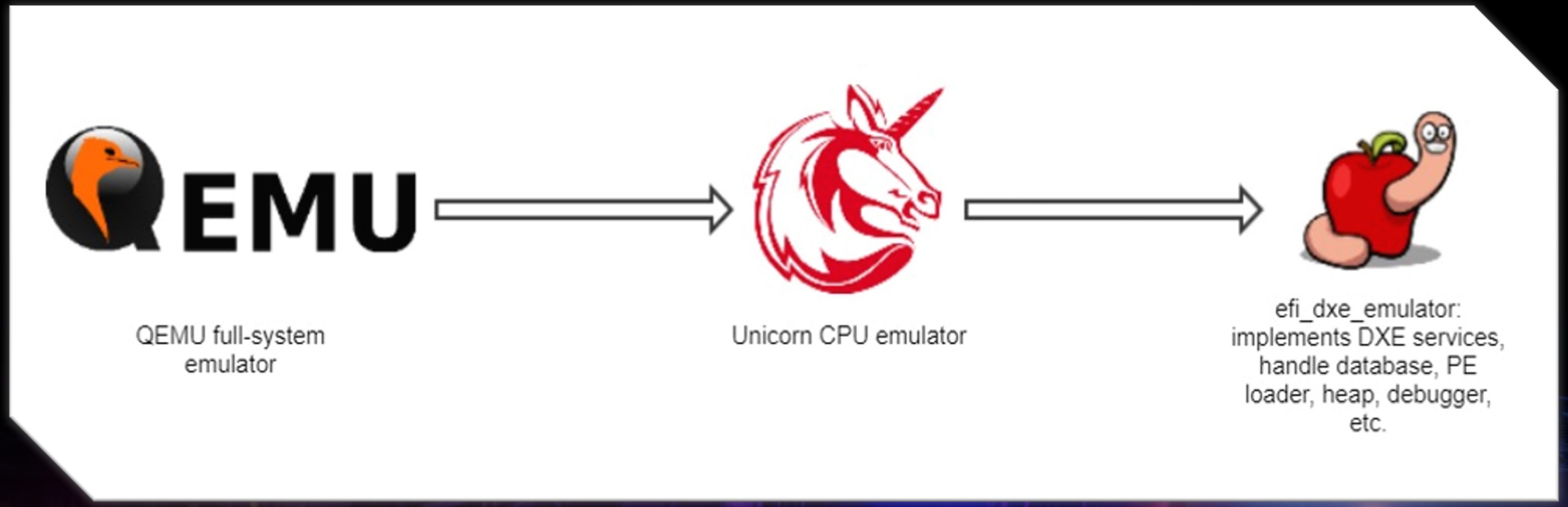


Intel® CCA And USB3.0 Debug Cable

Intel SVTCCA can be purchased through the Intel® Design-In Tools Store

Intel® DCI DbC2/3 A-to-A Debug Cable

Debugging and emulation of UEFI

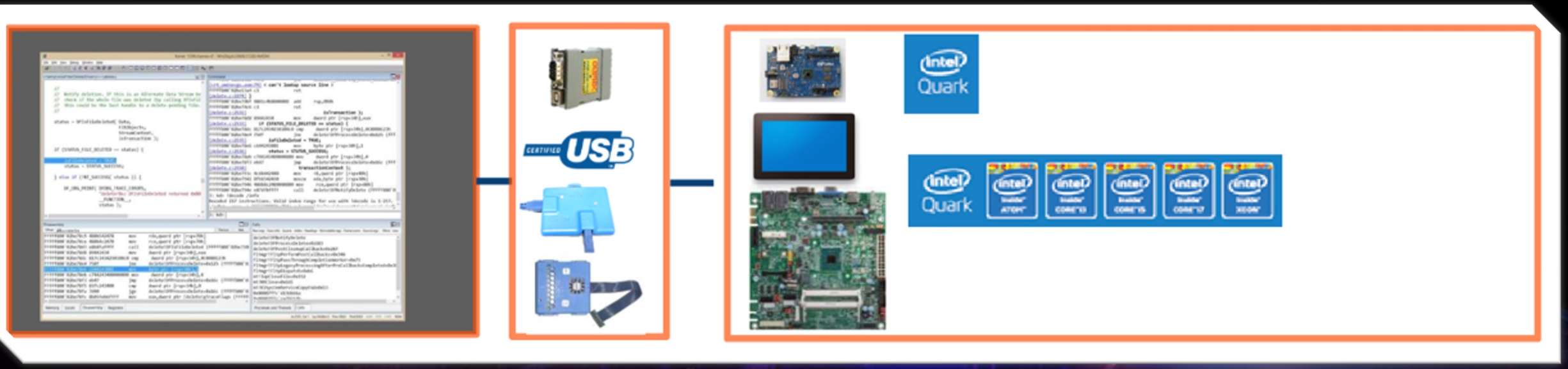


efi_dxe_emulator components

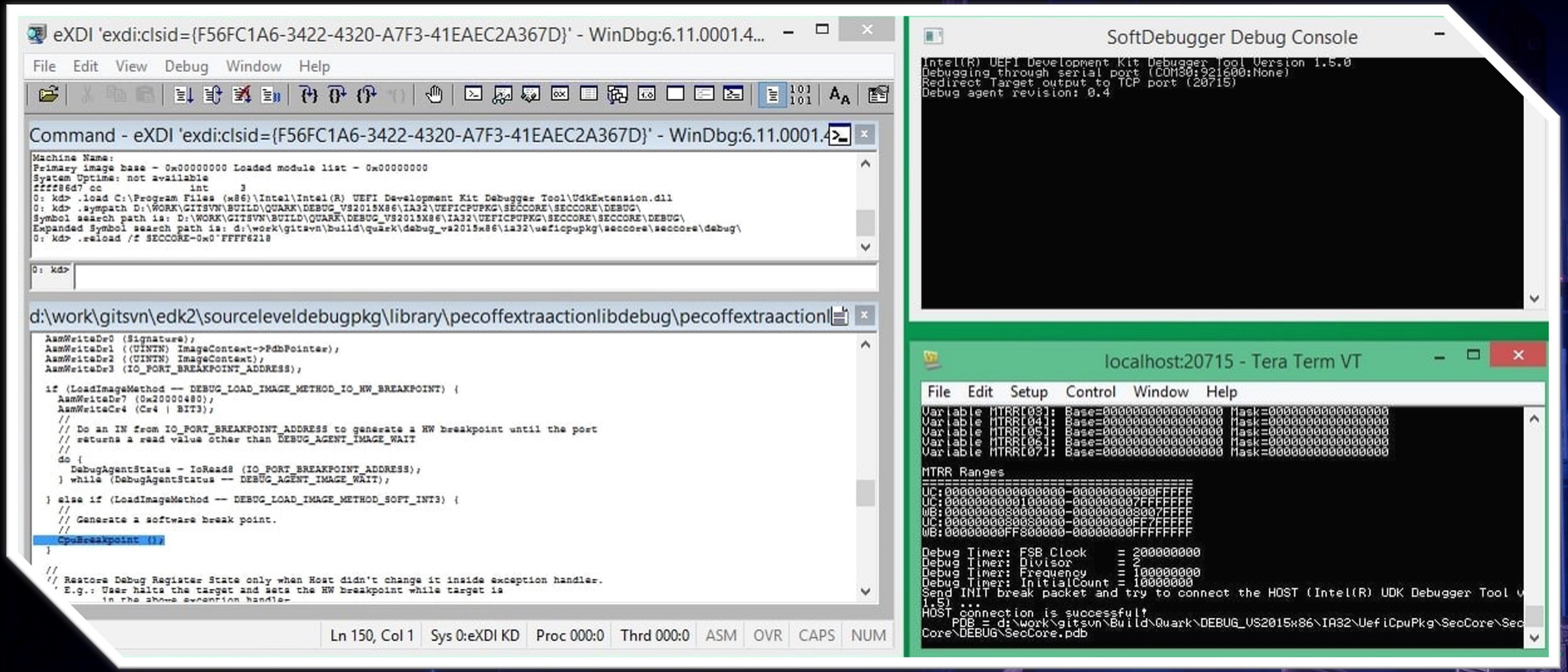
efi_dxe_emulator components

- efi_dxe_emulator - based on the famous Unicorn emulation engine, which is a fork of QEMU.
- IQEMU is just a tool, Unicorn engine is a real framework, and as such it provides a rich set of APIs that can be used by a large number of programming languages through dedicated bindings.
- Unfortunately, the Unicorn engine itself is not sufficient to perform dynamic analysis of UEFI modules. Since Unicorn focuses only on emulation of the processor, it has no a priori knowledge of UEFI-related concepts such as boot services, runtime services, protocols or even the PE format. To address these shortcomings, the efi_dxe_emulator independently implements some of the most commonly used UEFI services.

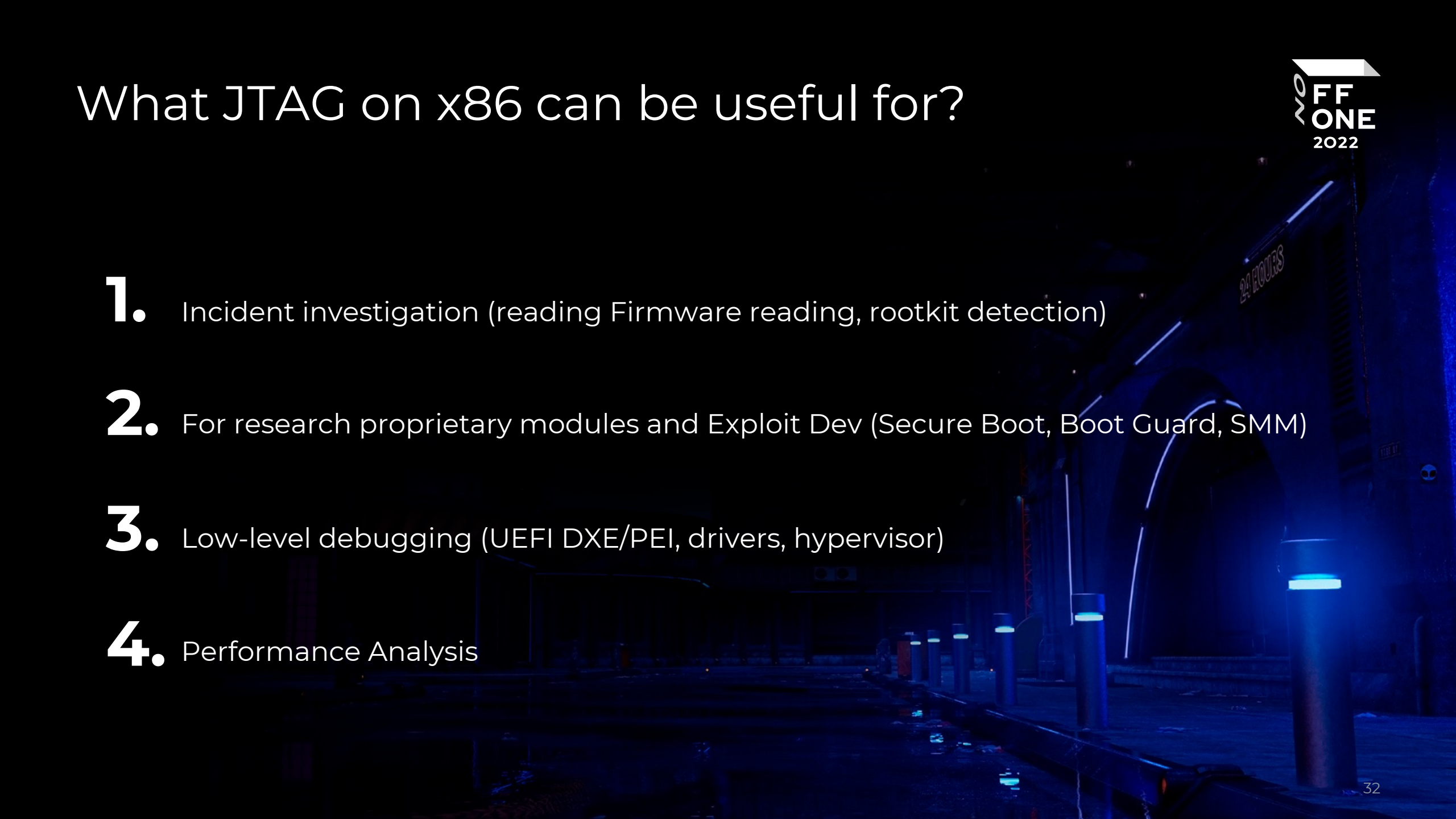
Variants of debugging



Variants of debugging with intel UDK



What JTAG on x86 can be useful for?



1. Incident investigation (reading Firmware reading, rootkit detection)
2. For research proprietary modules and Exploit Dev (Secure Boot, Boot Guard, SMM)
3. Low-level debugging (UEFI DXE/PEI, drivers, hypervisor)
4. Performance Analysis

Setup

Setup EDKII on Linux

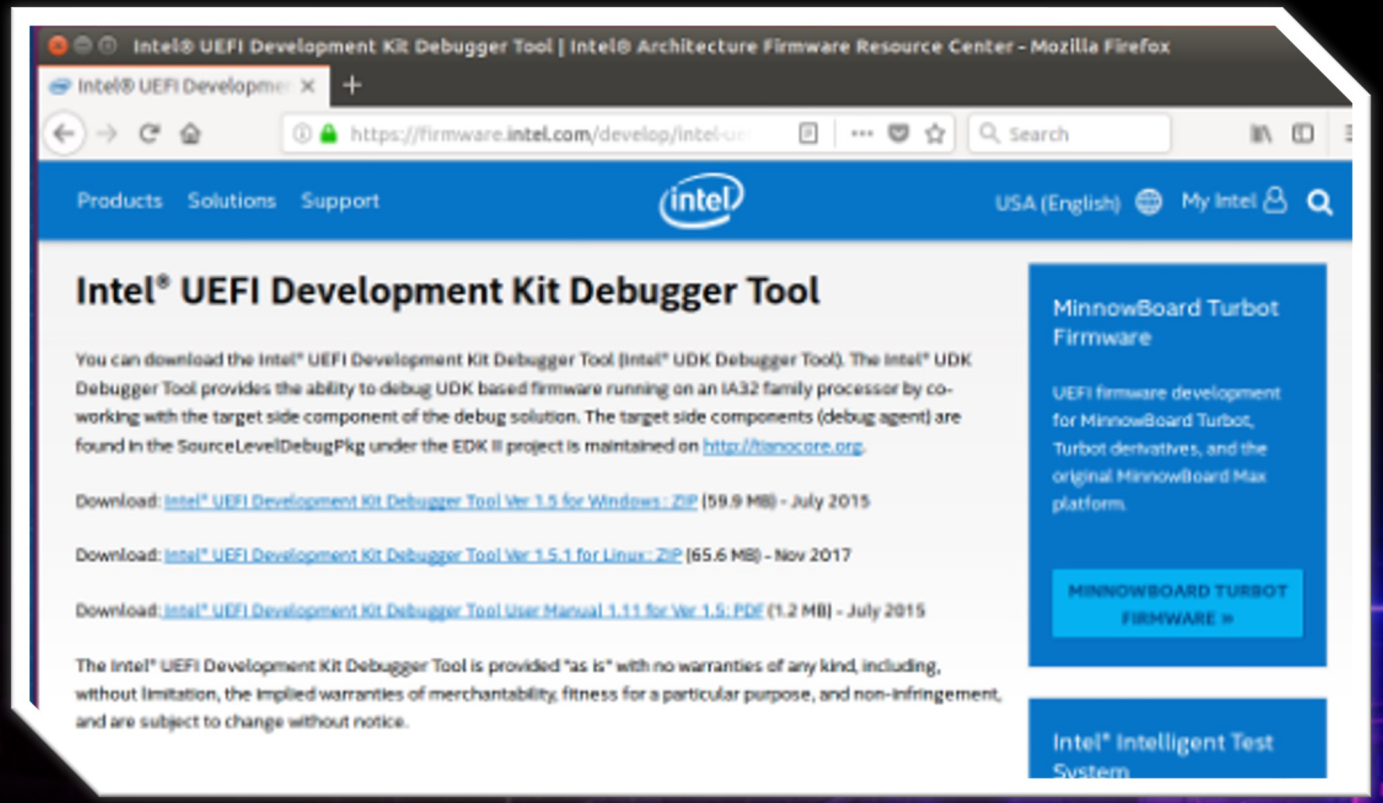
- Ubuntu 16.04
- GIT client
- GCC 4.9 compiler
- ASL compiler
- Python 2.7
- NASM
- GNU Debugger (GDB)

Build EDKII for Quark Platform

```
apt install -y git nasm iasl build-essential uuid-dev gdb gcc-4.9 g++-4.9
git clone https://github.com/tianocore/edk2.git -b vUDK2018
git clone https://github.com/tianocore/edk2-non-osi.git

export WORKSPACE=$PWD
export PACKAGES_PATH=$WORKSPACE/edk2:$WORKSPACE/edk2-non-osi/Silicon/Intel
export EDK_TOOLS_PATH=$WORKSPACE/edk2/BaseTools
cd $WORKSPACE/edk2

git submodule update --init

make -C BaseTools

. edksetup.sh BaseTools

build -a IA32 -t GCC49 -p QuarkPlatformPkg/Quark.dsc -D SOURCE_DEBUG_ENABLE
build -a IA32 -t GCC49 -p QuarkPlatformPkg/QuarkMin.dsc -D SOURCE_DEBUG_ENABLE
```

Host & Target Debug Setup

Download application: https:/firmware.intel.com-Develop-Tools



Target source: SourceLevelDebugPkg at TianoCore.org

Nuances associated with developing

- incorrectly structured information in the documentation
- Don't forget git submodules update -unit
- Compilers to build for Quark platform should be 4.9
- Don't forget to install all the packages you need to build the firmware
- We did the entire compilation in the distribution Ubuntu 16.04, with Ubuntu 18.04 there are their own quirks and drawbacks, we started everything with the docker container
- Tip: It is better not to hurry, think about every command you entered
- You can use qemu with ovmf to test it, it is faster and easier (better in some cases)
- If you do not succeed in the build step, look carefully at the errors and try to figure out what went wrong. Maybe python is the wrong version, maybe nasm is not installed, maybe the compilers are the wrong version or make; maybe you don't have openssl missing or basically an error because of dependencies

FLASH Update using CH341a

1

Be sure to dump your firmware from Intel Galileo

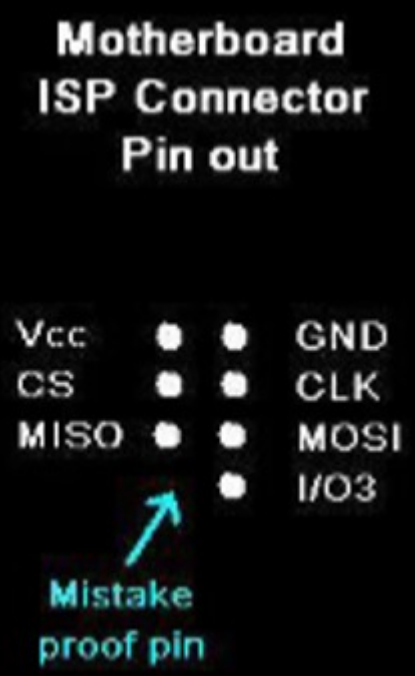
```
flashrom --programmer ch341a_spi -r
galileo-orig.bin
```

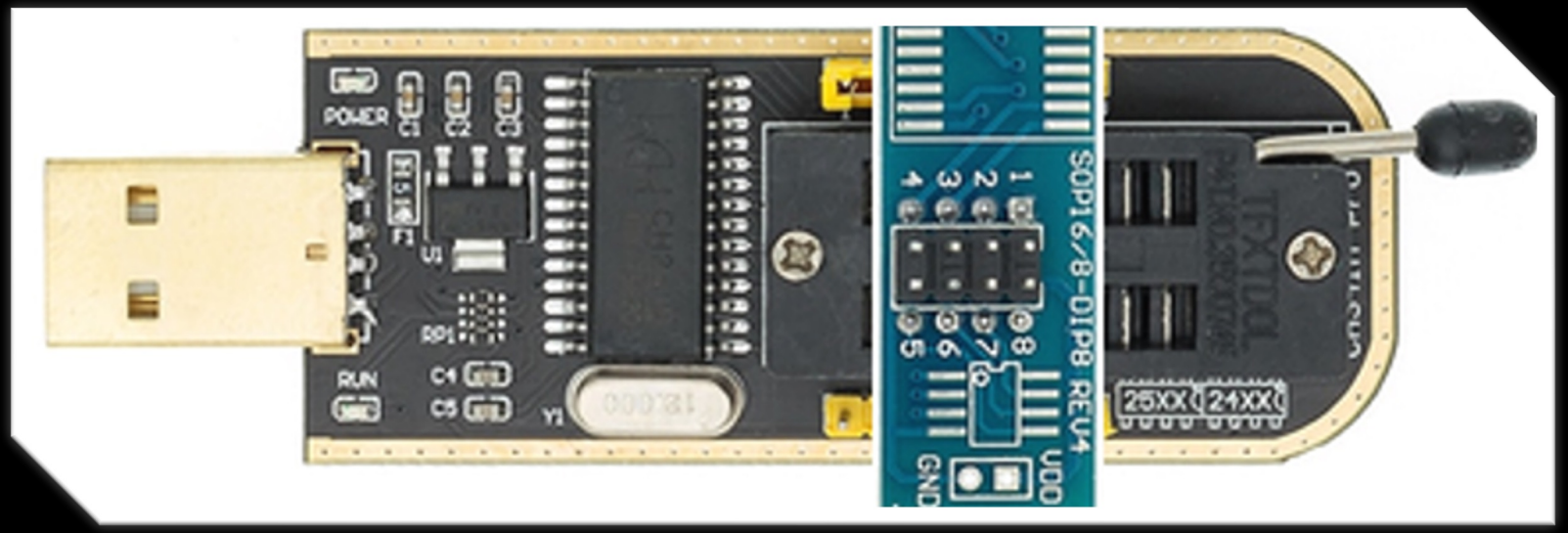
2

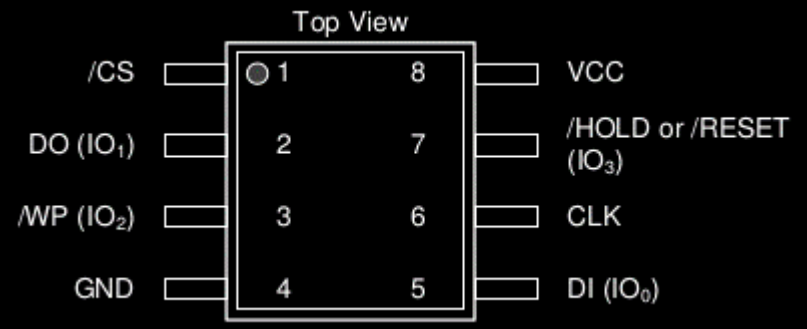
Flashing new fresh UEFI

```
flashrom --programmer ch341a_spi –w QUARK.fd
~/YouWorkDir/Build/Quark/DEBUG_GCC49/FV/QUARK.f
```

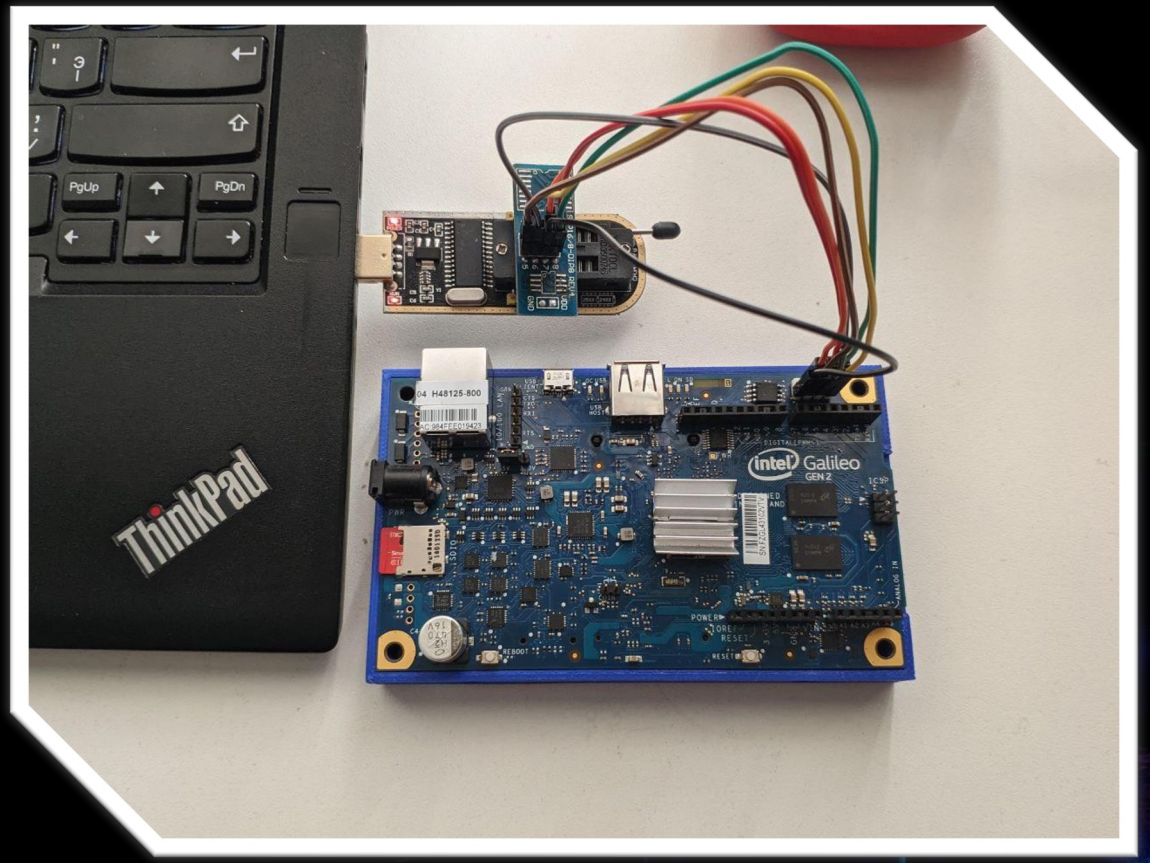
Connect SPI to CH341a



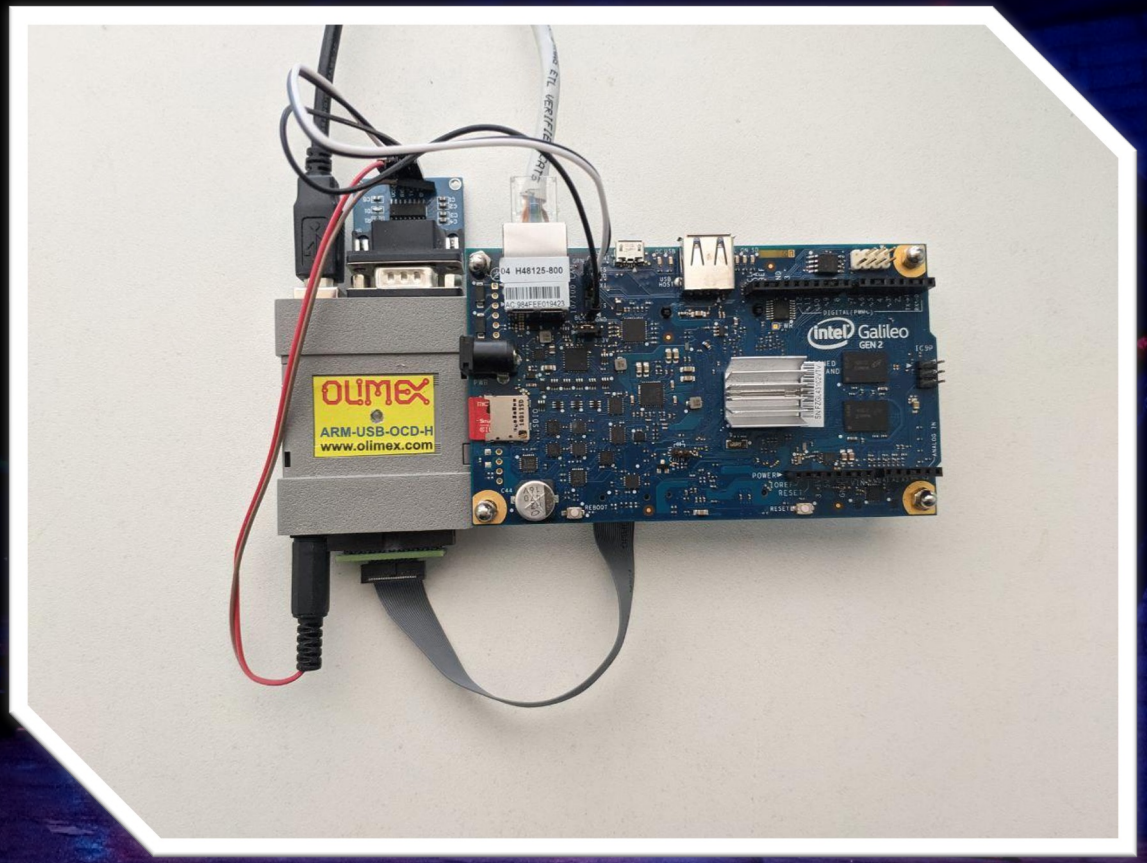




Connect

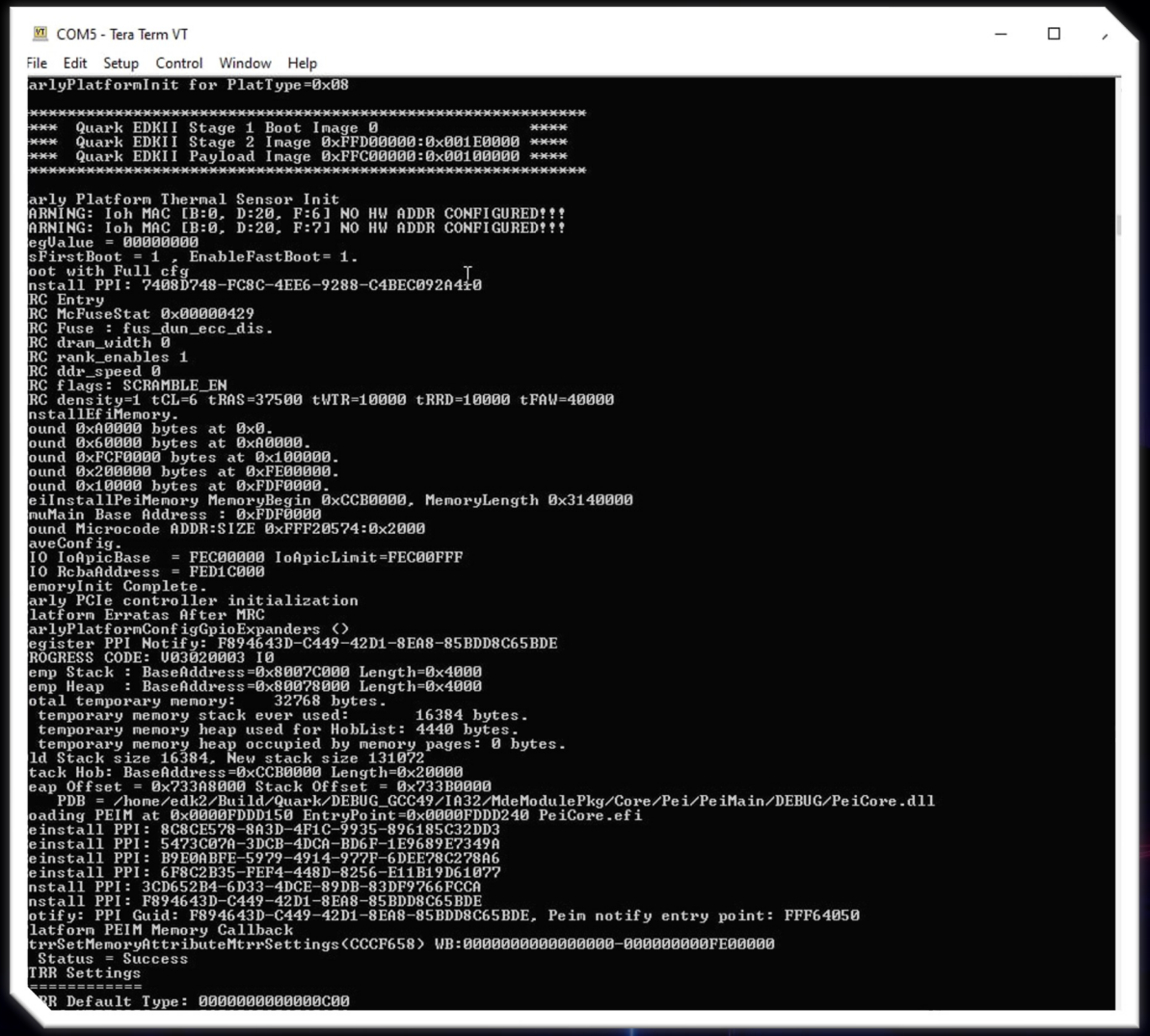


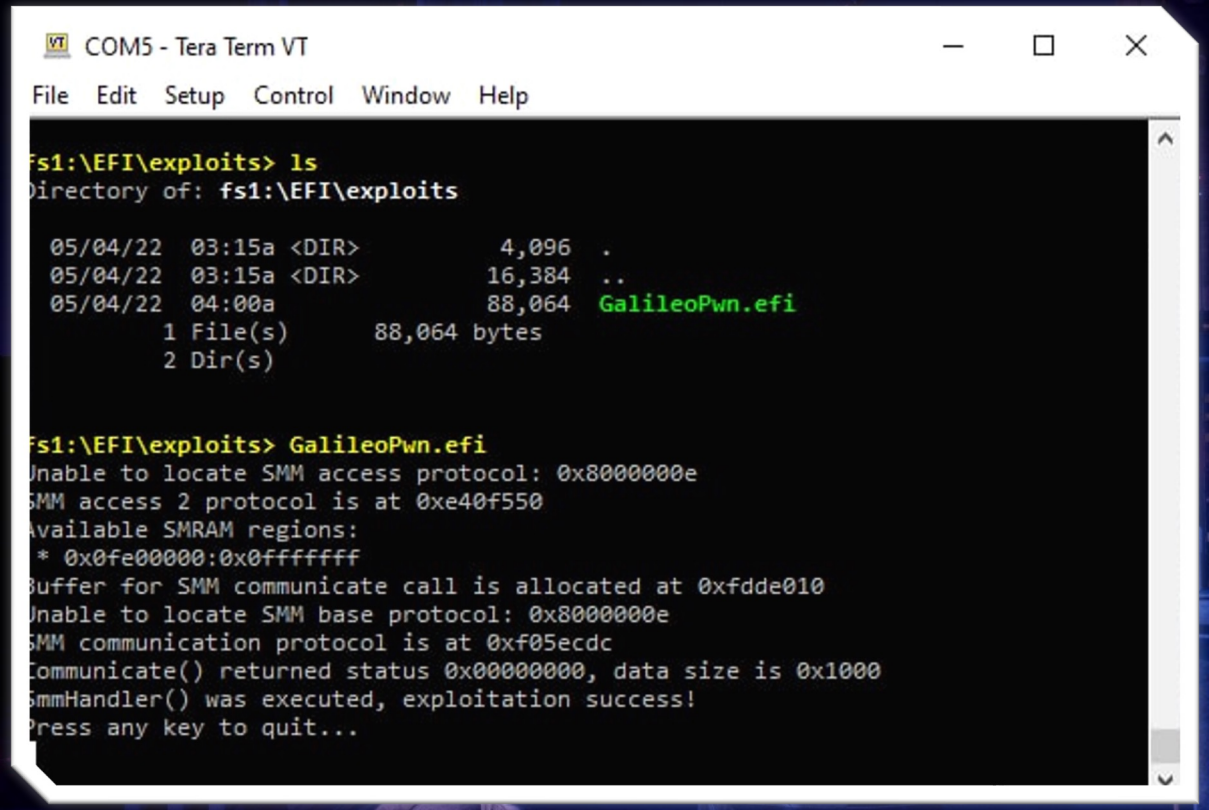
SPI to CH341a



UART and JTAG to Galileo

Test UEFI Shell






OFF
ONE
2022

ON FF ONE 2022

DEMO