

Discover UEFI with U-Boot

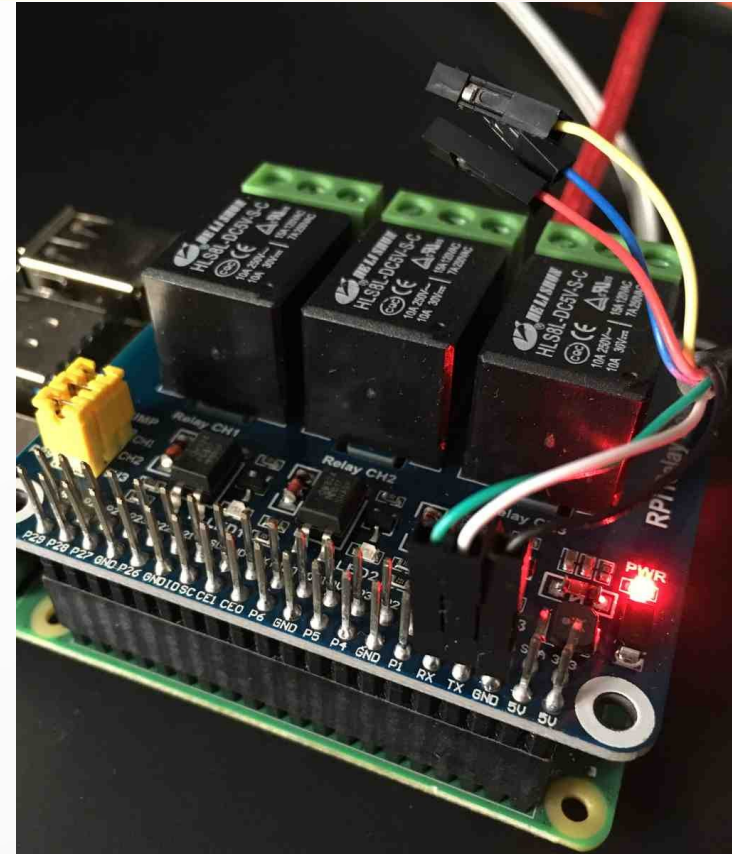
2020-02-01, Heinrich Schuchardt
CC-BY-SA-4.0

About Me

- Software-Consultant ERP, Supply Chain
- Contributor to U-Boot since 2017
- Maintainer of the UEFI sub-system since 02/2019

I Want a Network Drive

- Many single board computers have neither SATA nor PCIe.
- For many boards Ethernet is the fastest connector.
- An SSD drive costs more than most SBCs.



Network Booting in U-Boot

BOOTP

- BOOTP server provides tFTP server address and name of boot script

tFTP

- Boot script loads kernel via tFTP or NFSv3 (UDP)

NFSv3

- **No authentication at all**

iSCSI

- SCSI protocol transported via TCP
- Offers entire data stores (LUNs) to iSCSI client
- Mutual authentication of client and server with CHAP: MD5(ID + secret + challenge)
- Further security via VLAN separation and IPsec

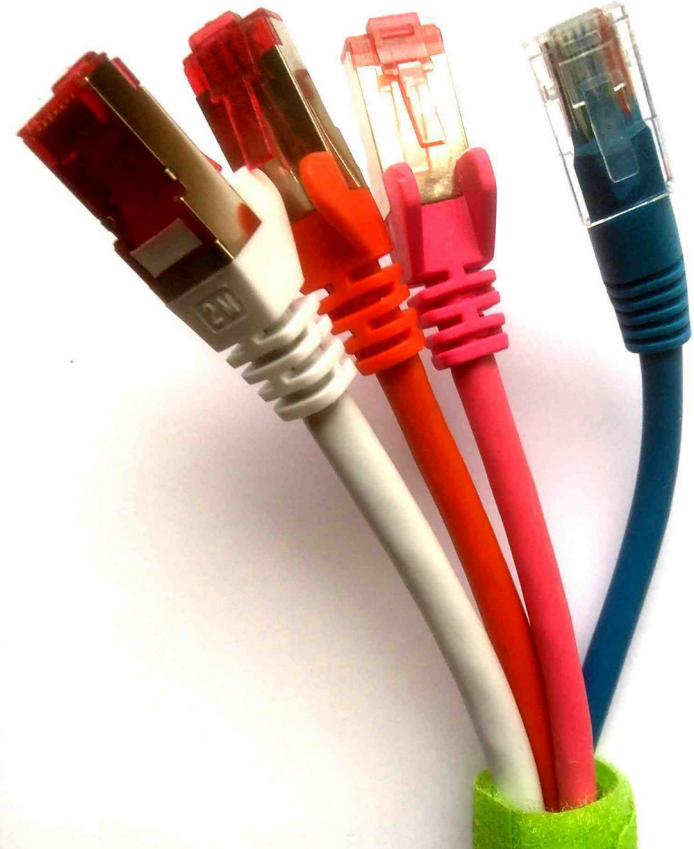
iPXE

Swiss army knife of network booting:

- Boot from HTTP(s) server
- Boot via iSCSI
- Boot via FCoE (Fibre Channel over Ethernet)
- Boot via AoE (ATA over Ethernet)
- Scriptable
- Can be built as **UEFI** payload

See <https://ipxe.org>,

Developer Michael Brown <mcb30@ipxe.org>



UEFI in U-Boot Started 2016



Embedded Linux
Conference



OpenIoT Summit

— FEBRUARY 21-23 — PORTLAND, OR —

Marrying U-Boot, uEFI and grub2

Alexander Graf, *SUSE*

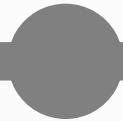
First Try

```
=> load mmc 0:2 $fdt_addr_r dtb
19600 bytes read in 303 ms (62.5 KiB/s)
=> load mmc 0:1 $kernel_addr_r snp.efi
reading snp.efi
149280 bytes read in 31 ms (4.6 MiB/s)
=> bootefi $kernel_addr_r $fdt_addr_r
## Starting EFI application at 42000000 ...
Scanning disks on usb...
Scanning disks on mmc...
MMC Device 1 not found
MMC Device 2 not found
MMC Device 3 not found
Found 5 disks
## Application terminated, r = -2147483639
=> █
```

-2147483639 = 0x80000009 = EFI_OUT_OF_RESOURCES

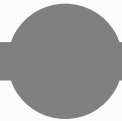
My U-Boot Journey Begins

Bare Minimum
to start GRUB



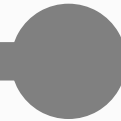
2016/17

iSCSI boot
with iPXE



2018

Run EFI Shell
and EDK II SCT

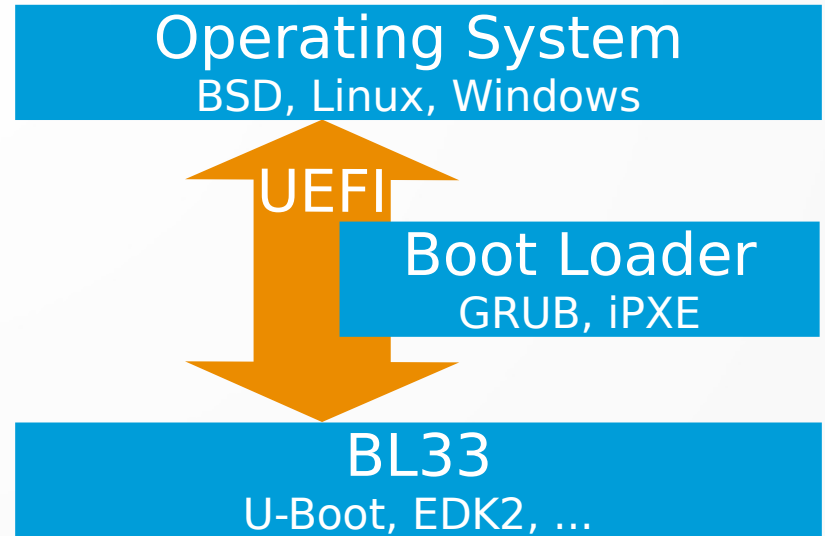


2019

together with 40 UEFI sub-system contributors

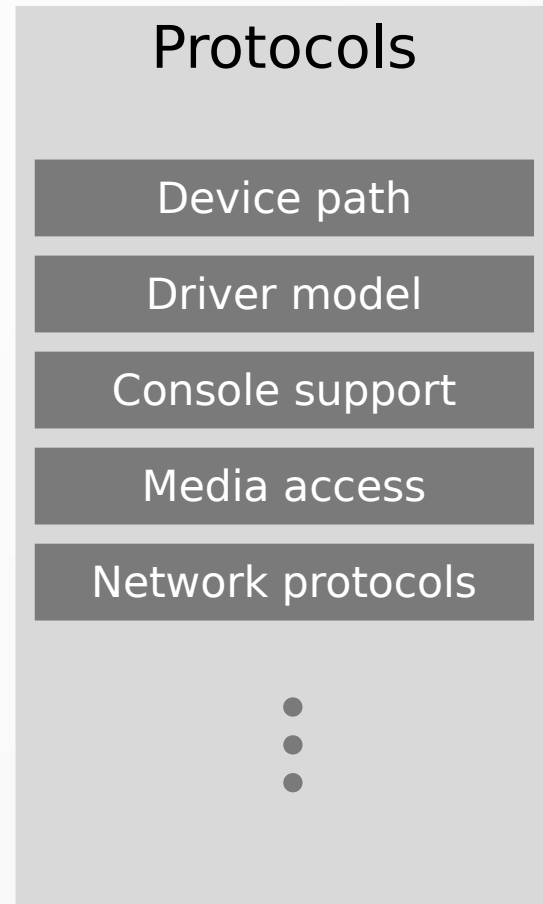
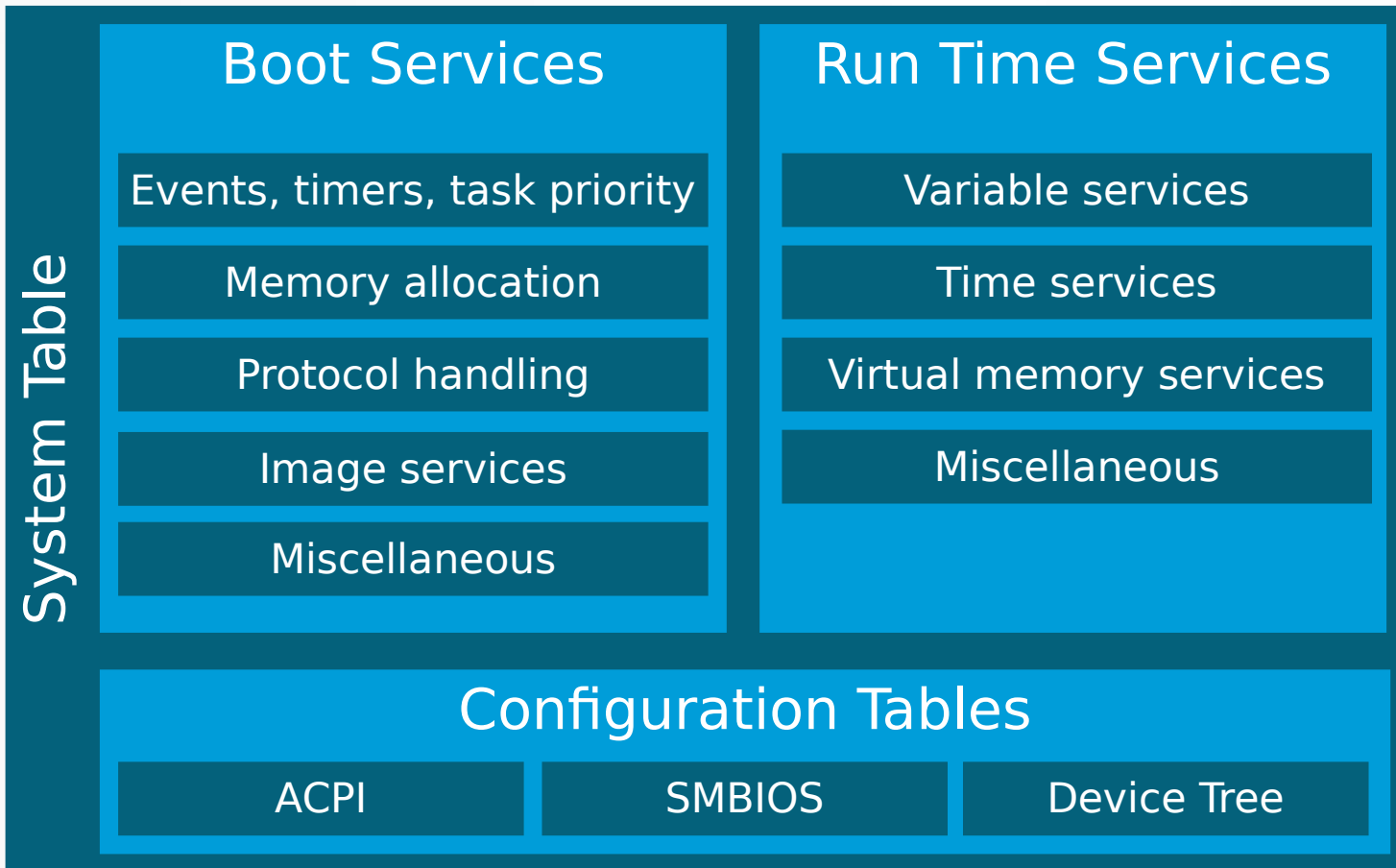
Where Sits UEFI?

Booting with ATF on ARMv8



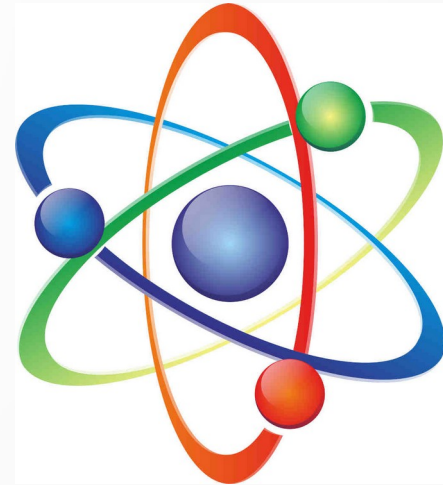
ATF - ARM Trusted Firmware
OP-TEE - Open Portable Trusted Execution Environment
PSCI - Power State Coordination Interface
UEFI - Unified Extensible Firmware Interface

UEFI



“Atoms” of UEFI

- Handles
 - void* pointer
 - Protocols are installed on handles
- Events
 - Triggered by timer or service call
 - Callback function



Lifetime of a Handle

Creation by
installing first protocol

Deletion by
removing last protocol



InstallProtocolInterface
InstallMultipleProtocolInterfaces

UninstallProtocolInterface
UninstallMultipleProtocolInterfaces

Driver

- Handle with `EFI_DRIVER_BINDING_PROTOCOL`
 - GUID
 - {0x18A031AB, 0xB443, 0x4D1A, {0xA5, 0xC0, 0x0C, 0x09, 0x26, 0x1E, 0x9F, 0x71}}
 - Protocol Interface Structure
 - Supported()
 - Start()
 - Stop()
 - Version
 - ImageHandle
 - DriverBindingHandle

Device (aka Controller)

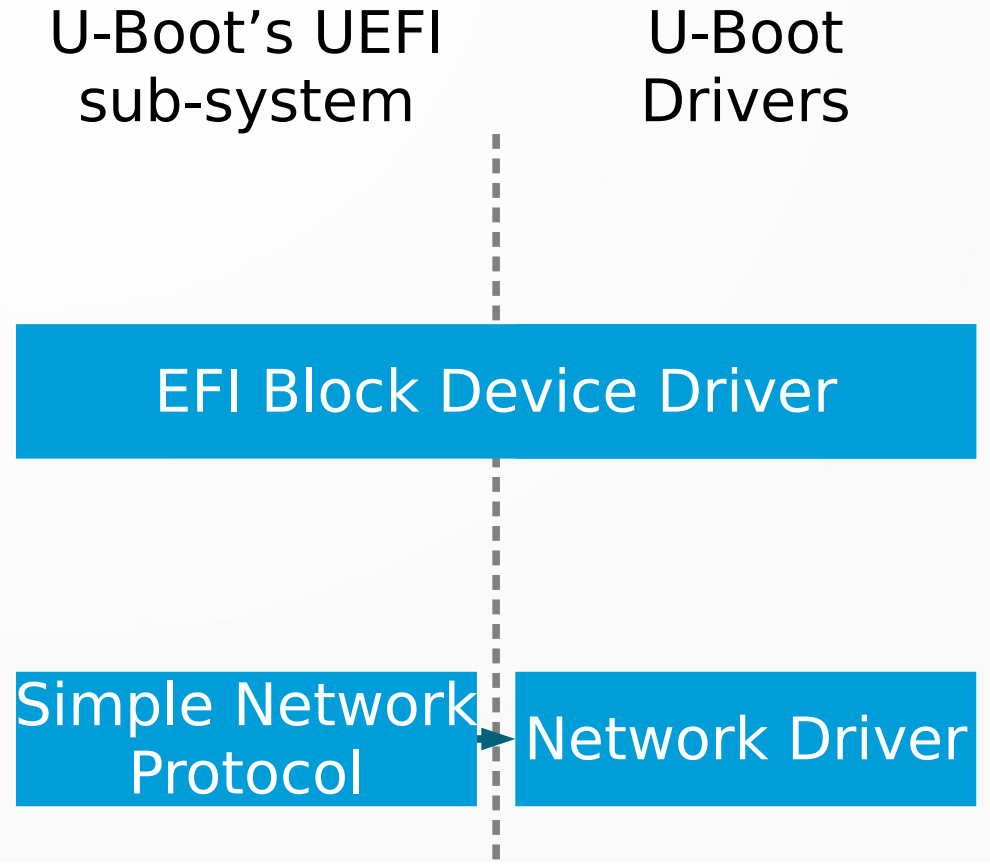
- Handle with the `EFI_DEVICE_PATH_PROTOCOL`
 - Sequence of device path nodes
 - Arranges devices in a tree

- ▶ `PciRoot(0x0)`
- ▶ `PciRoot(0x0)/Pci(0x1,0x1)`
- ▶ `PciRoot(0x0)/Pci(0x1,0x1)/Pci(0x0,0x0)/NVMe(0x1,AD-A9-B1-73-55-38-24-00)`
- ▶ `PciRoot(0x0)/Pci(0x1,0x1)/Pci(0x0,0x0)/NVMe(0x1,AD-A9-B1-73-55-38-24-00)
/HD(1,GPT,F24494A4-585B-4E34-A367-4DC70CFFC93D,0x800,0x1DC800)`
- ▶ `PciRoot(0x0)/Pci(0x8,0x2)`
- ▶ `PciRoot(0x0)/Pci(0x8,0x2)/Pci(0x0,0x0)/Sata(0x0,0x0,0x0)
/HD(1,GPT,11C3D446-F6E4-4C67-937E-992AFC6F454F,0x800,0x108800)`

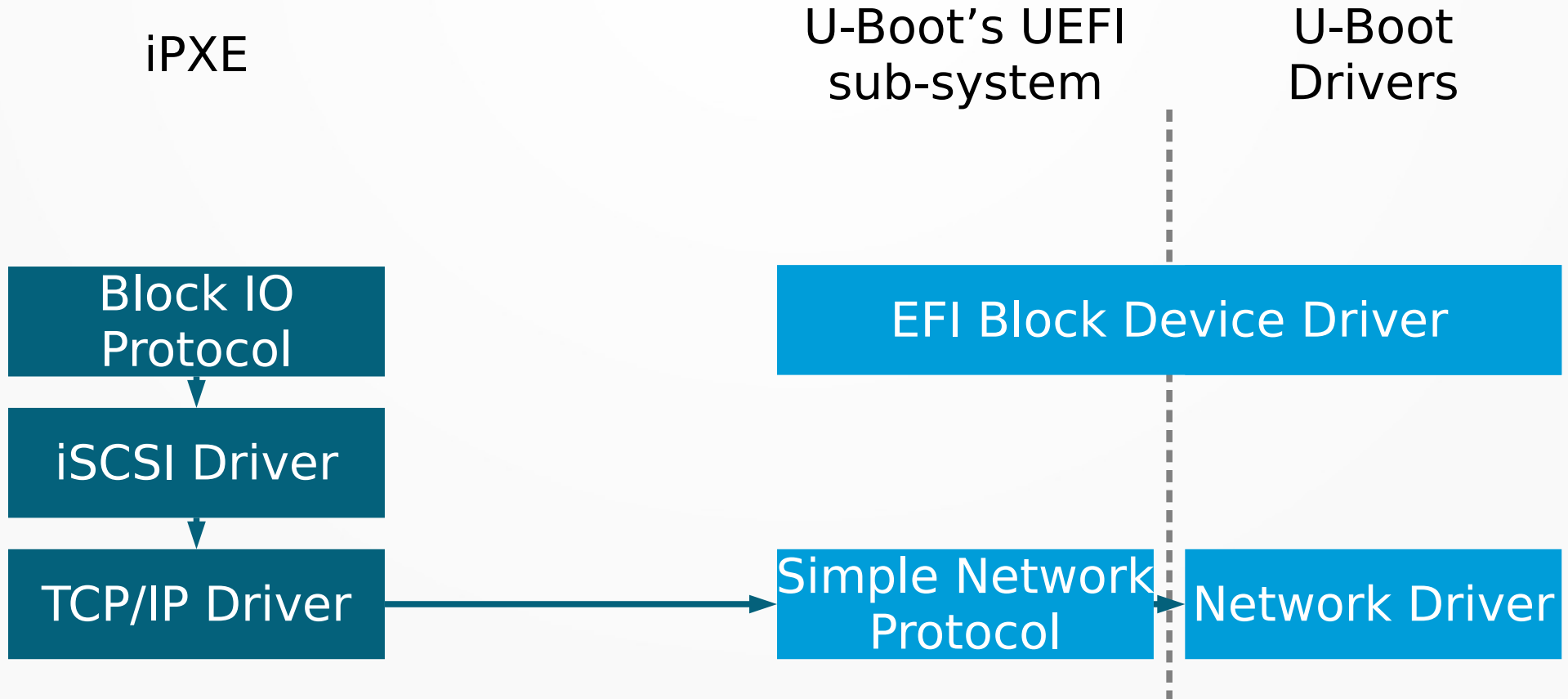
Attaching Drivers

- ConnectController() boot service
 - calls Supported() methods of all drivers to find matches for controller
 - calls Start() method of the matching drivers
- Driver
 - installs protocols on controller
 - may create child controllers

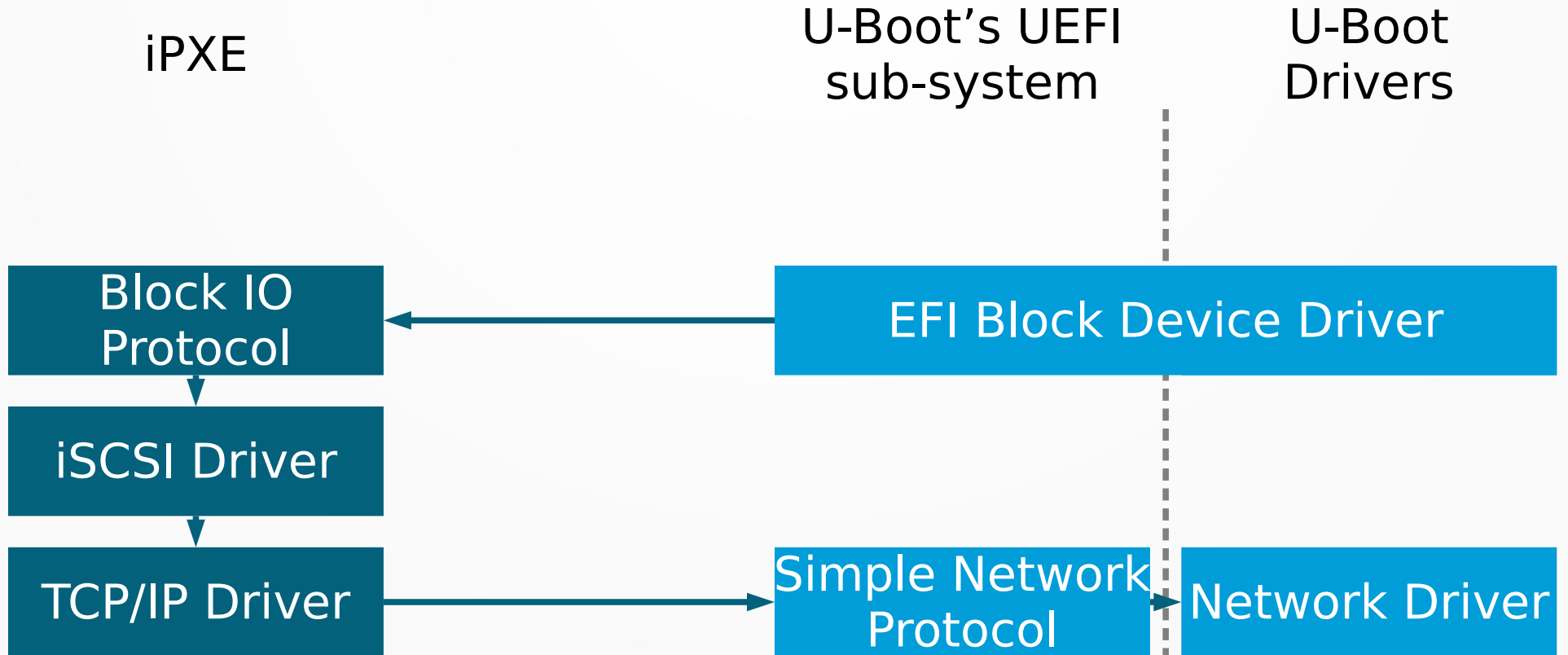
U-Boot Exposes Ethernet



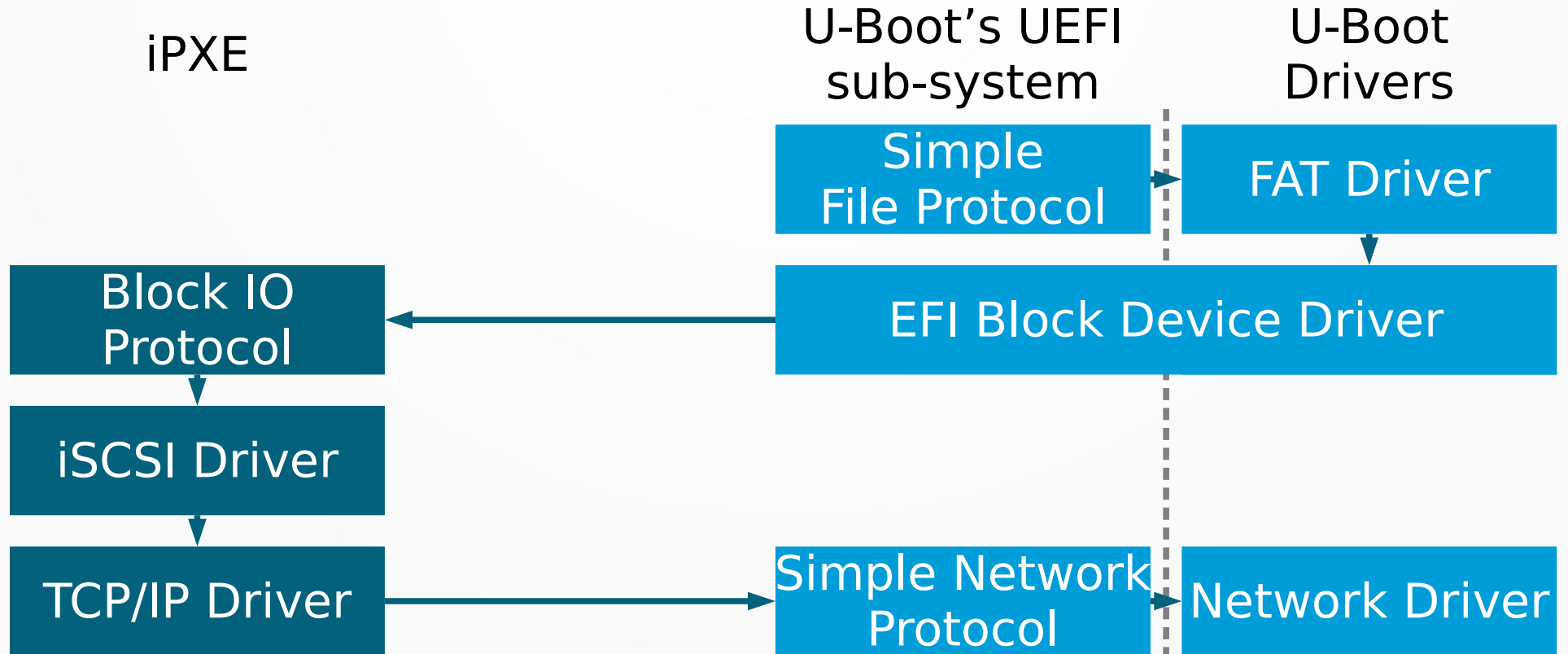
iPXE Exposes Block IO Protocol



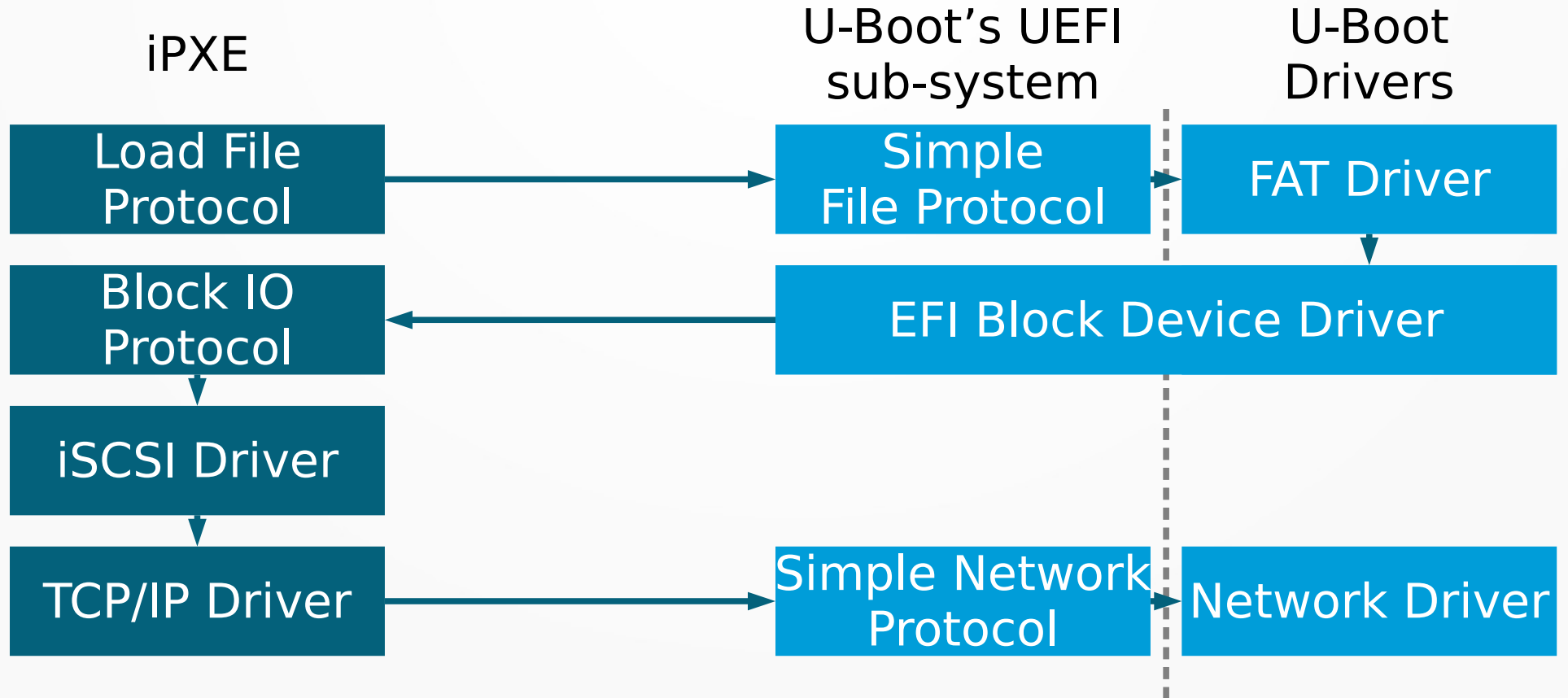
iPXE Connects Controller



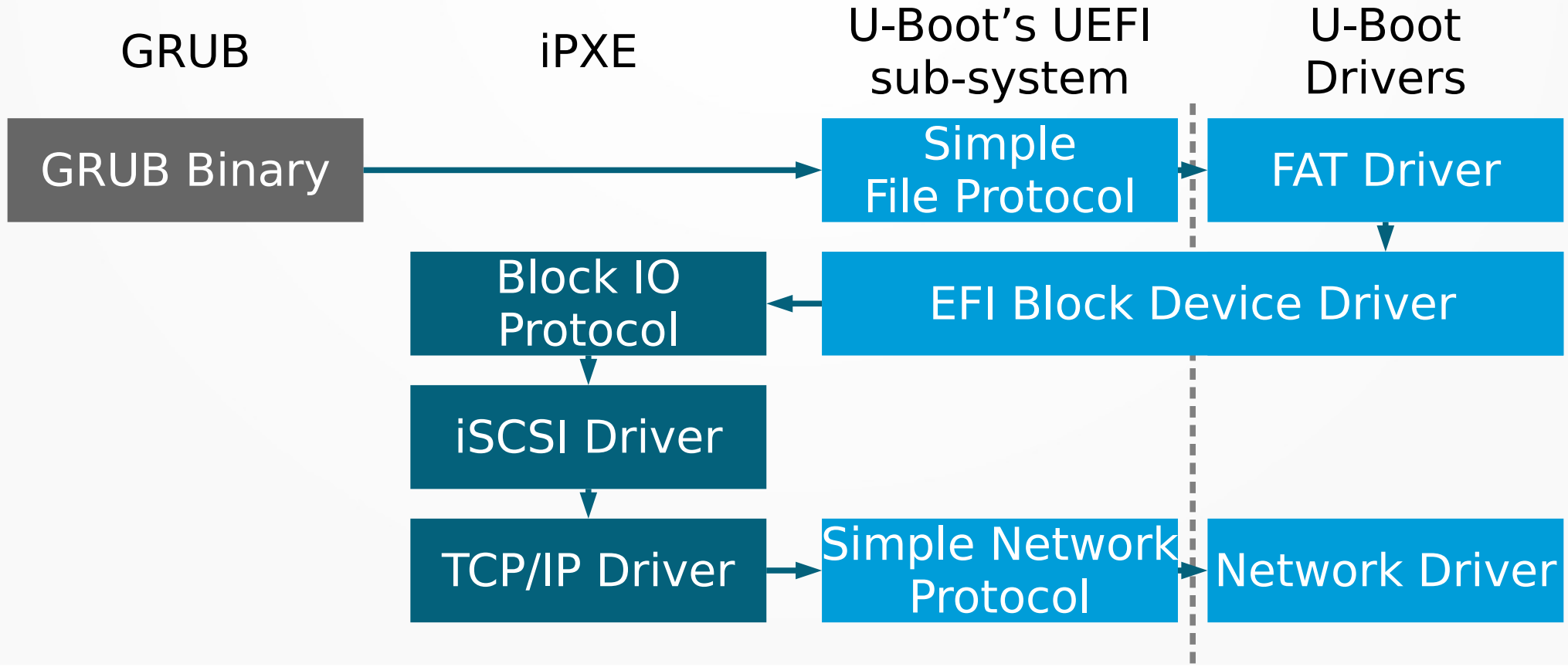
U-Boot Discovers Partitions



iPXE and U-Boot Loading File



GRUB Loading Kernel



Take Away

- Providing UEFI in U-Boot as a standardized API allows for easy integration with other software



Implementation Events

- U-Boot is single threaded
- No interrupts supporting networking, timers
- Call event handling routines in
 - console routines
 - network routines
 - CheckEvent(), WaitForEvent(), RestoreTPL(), Stall()

Integration of UEFI sub-system

- U-Boot is in the middle of moving from legacy drivers to a device tree based driver model
- UEFI sub-system sits on top of U-Boot rather than being integrated into U-Boot driver model

Development Targets

- Support subset of UEFI specification
 - Embedded Base Boot Requirements (EBBR)
 - Boot services
 - Run time services
 - Required elements according to UEFI 2.8, chapter 2.6
- Stay small
 - 31000 lines, ca. 70 kiB in U-Boot binary

Achievements in 2019

- Missing boot services added
- Major improvements in UEFI standard compliance
<https://github.com/U-Boot-EFI/u-boot-sct-results>
- U-Boot runs EFI shell on ARM, x86, x86_64
- U-Boot runs EDK II SCT on ARM, x86

Work in Progress

- Verified UEFI Boot via FIT images
Cristian Ciocâltea
- UEFI Secure Boot
Takahiro Akashi (Linaro)
- EFI_RNG_PROTOCOL based on hardware RNG
Sugosh Gani (Linaro)