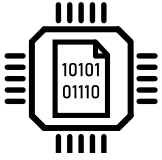
Das Hoch und Runter mit ARM Systemen



Ein Ausflug durch das ARM-Labyrinth

Ahmad Fatoum – a.fatoum@pengutronix.de

Pengutronix.

https://www.pengutronix.de

Unser SoC: NXP i.MX8MM

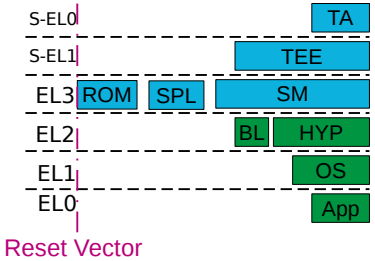
- ARM SoCs gibt es viele, wir nehmen uns einen i.MX8MM als Beispiel
- Mehrere Kerne: 4x ARM Cortex-A53, 1x Cortex-M4
- ARMv8.0-A
- Vergleichsweise offene Dokumentation (>7000 Seite SoC-Doku)

  → Gute Open-Source Software Unterstützung

[MNT Reform]

[Purism Librem 5]

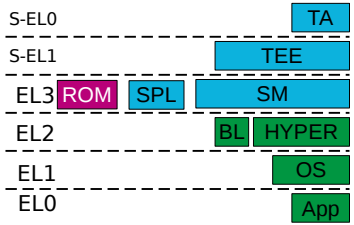
Am Anfang war der Reset-Vector

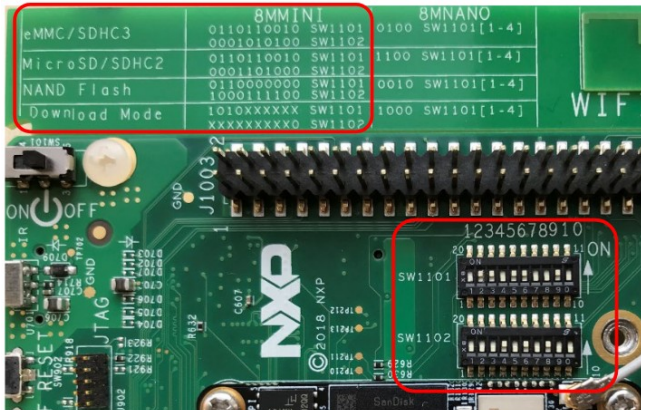


- Startadresse der Programmausführung
- Muss auf direkt ausführbaren Speicher zeigen
  - Eingebauter SRAM → Muss befüllt werden (JTAG, Koprozessor)
  - Memory-Mapped Flash:
    - Aufwand: Muss vorgeflasht sein
    - Kosten: Muss oft als separate Komponente verbaut werden
    - Sicherheit: Wer auch immer es neuprogrammieren kann, hat volle Kontrolle
  - Mask Boot ROM:
    - Aufwand: Kann auch als In-System-Programmer genutzt werden
    - Kosten: Software kann aus vielen verschiedenen Quellen geladen werden
    - Sicherheit: Wenn man es vertraut, kann es auch den Bootloader verifizieren

Boot ROM

- Konfiguriert PLLs, Clocks, Stack, Interrupts
- Basis-System-Initialisierung, z.B. Latenzen des eingebauten SRAMs
- Abfrage der eFuses and Strapping Pins um Bootquelle zu bestimmen
- Bootmedien in bestimmter Reihenfolge abfragen





[EVK]

Boot ROMs sind komplex

- Ein Favorit: OMAP tftp-über-usb-ethernet-gadget

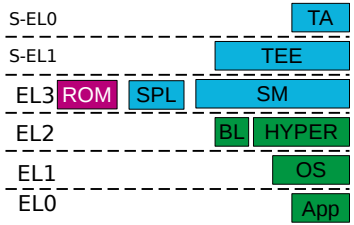
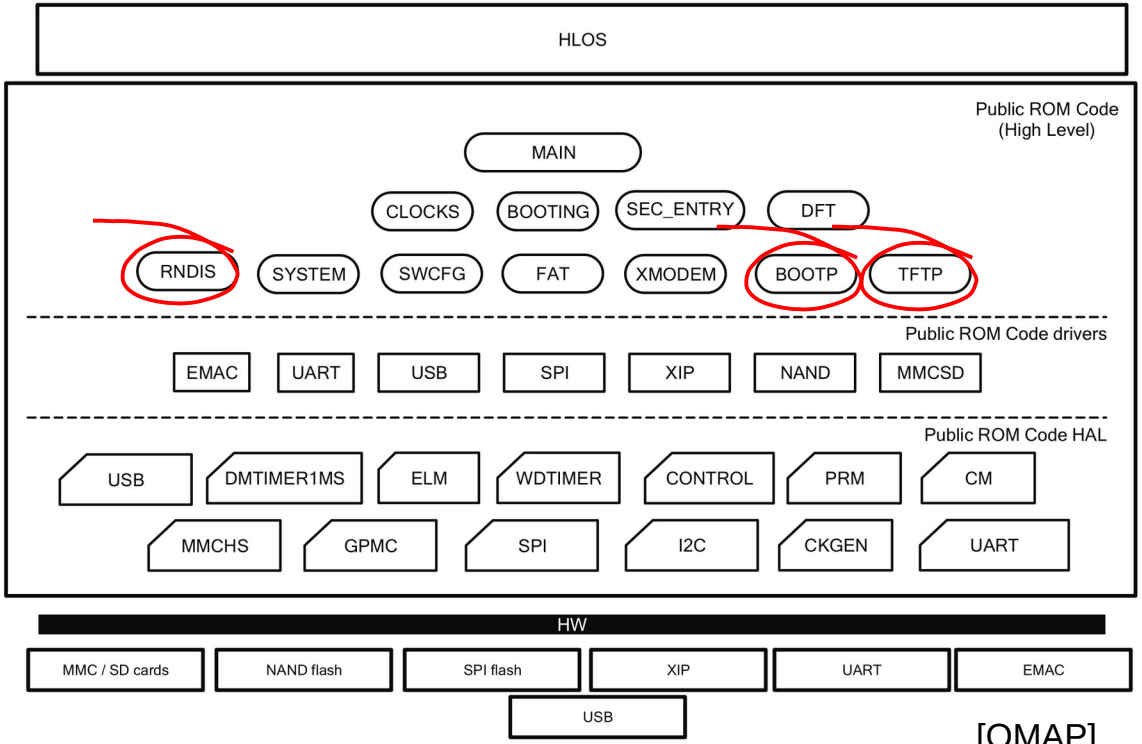


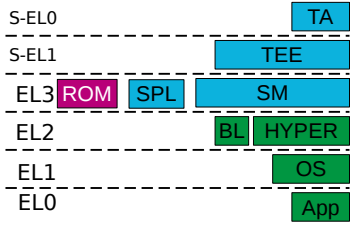
Figure 26-1. Public ROM Code Architecture



[OMAP]

Boot ROMs sind fehleranfällig

- Alles hat Bugs
- Manchmal sind Workarounds möglich:



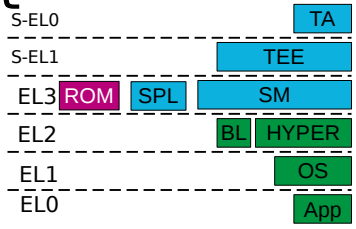

```
ARM: imx8mq: reclock ARM PLL to 800MHz

The BootROM sets up the ARM PLL to run at 1.6GHz and then uses the
divider after the PLL the achieve a CPU clock rate of 800MHz. New Linux
kernels (>= 5.8) switch to a clock path that bypasses the divider, as
the divider should not be used for CPU clock frequencies >1GHz. If the
BootROM setup is left unchanged this causes the CPU clock to jump to
the full 1.6GHz until CPUfreq takes over and reprograms the PLL. This
rate is outside of the chip specification and leads to crashes.

Fix this by reclocking the ARM PLL to 800MHz.
```

- Manchmal nicht:

```
CVE-2017-7932

An improper certificate validation issue was discovered in NXP i.MX [...].
When the device is configured in security enabled configuration, under certain
conditions it is possible to bypass the signature verification by using a
specially crafted certificate leading to the execution of an unsigned image.
```

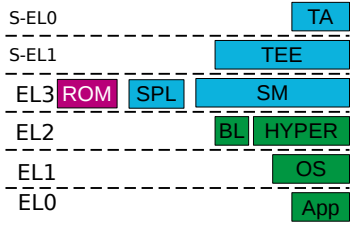
i.MX: Image Vector Table (IVT)

- Auf dem Bootgeräts an einem (typspezifischen) Offset
- Enthält Offsets von
  - Device Configuration Data (DCD)
  - Nächste Bootloader Stage
  - Ladeadresse
  - Command Sequence File (für Secure Boot)
  - NXP-signierte HDMI PHY Firmware
- Beschreibt, ob boot image ein Plugin ist



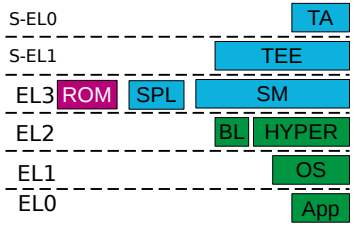
i.MX: Device Configuration Data (DCD)

- Bytecode, den der i.MX BootROM interpretiert
- Ausgeführt vor dem Laden des boot programms ins den Speicher
- Darf auf bestimme Speicherregionen zugreifen
  - write 8/16/32 bit literal
  - poll until `(*addr & mask) == mask`
  - set/clear bits
- Ist mächtig genug um einen DDR3 SDRAM Controller aufzusetzen
  - Aber nicht flexibel genug um DDR4 komplett aufzusetzen
  - Oder einen PMIC per I2C zu konfigurieren



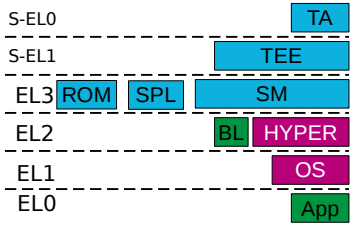
i.MX: Plugin Image

- ARM-Code, für den Fall, dass die DCD den SDRAM nicht aufsetzen kann
- springt zurück in den BootROM
- Pluspunkt: BootROM übernimmt frühe Authentifizierung
- Aber: Lesen vom Speichermedium muss man selbst implementieren
  - Neuere i.MX8M SoCs unterstützen ROM API um BootROM-Treiber zu benutzen
  - Die neusten i.MX95 SoC aber nicht mehr, weil anderer CPU initial bootet :(
- Wenig benutzt: Von barebox gar nicht



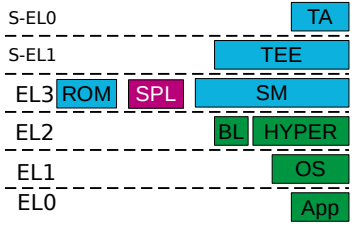
Was nun?

- Wir wissen wie wir eigenen Code ausführen
- Was fehlt damit ein Linux läuft?
  - DRAM aufsetzen
  - Secure monitor installieren für CPU PM
  - Device Tree übergeben
  - Kernel Image anspringen



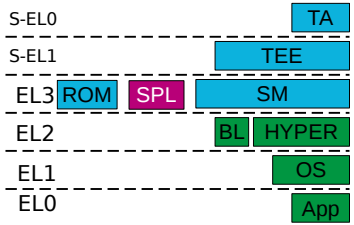
DRAM aufsetzen

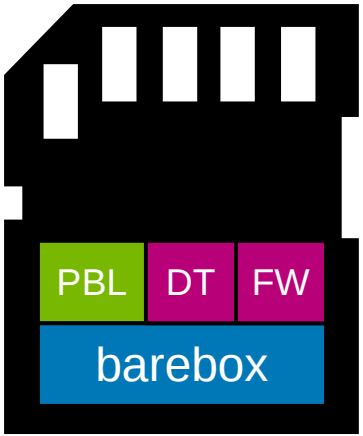
- Spannungsregler einstellen
  - Werkseinstellungen nicht immer passend
  - Laufzeiteinstellung über I^2C
- DDR DRAM Controller aufsetzen
  - Hersteller liefert Tool und Spreadsheet, das C-Code generiert
  - DDR PHY Mikrocontroller Firmware Blob fürs Link Training
- Code muss außerhalb des DRAMs ausgeführt werden um es zu initialisieren:
  - DCD Kommandos: wie beim i.MX6
  - ARM-Code: Notwendig um mit DDR4 PHY µC zu kommunizieren



Secondary Program Loader (SPL)

- Auch first-stage bootloader genannt (BootROM is 0th Stage)
- Startet aus dem SRAM oder eXecute-In-Place (XIP) vom Flash und setzt DRAM auf
- Bei barebox, übernimmt das der PreBootloader (PBL):
  - Im einfachsten Fall, barebox direkt an den Ende des SDRAM extrahieren
    - Ermöglicht komprimierte multiplattform bootloader images
  - Plattformabhängig, kann es auch:
    - Hardware-Beschreibung durch reichen an nachgeladenen barebox proper
    - Low-level-Initialisierung und SDRAM aufsetzen
    - Frühes Laden von Boot-Firmware

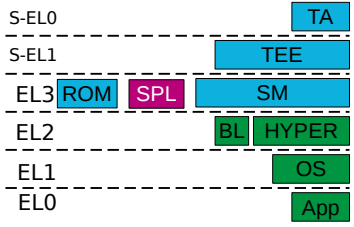


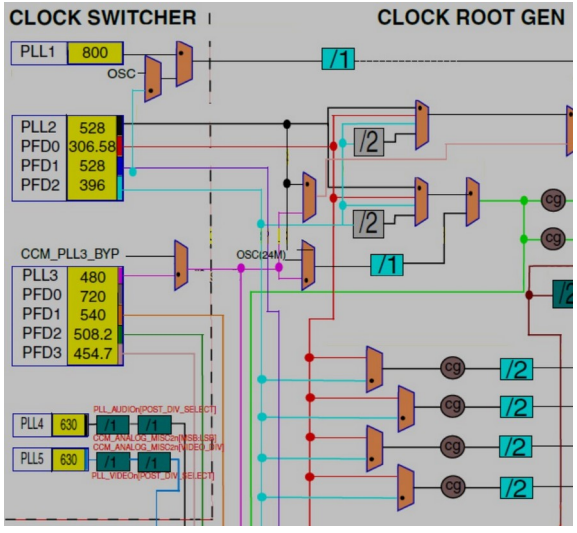


PBL: SoC initialisieren

```
imx8mm_cpu_lowlevel_init();
imx8mm_early_clock_init();
```

- Non-secure EL0/1 vorbereiten
- Spätere Level als AArch64 konfigurieren
- Ausschalten von Traps, MMU, caches … usw.
  - Register können invalide vom BootROM zurückgelassen worden sein
- Clock tree initialisieren für PBL-Nutzung

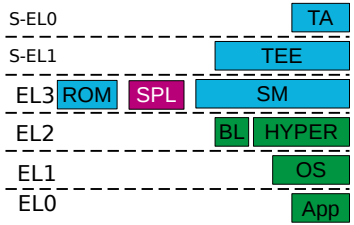




[MX6-CLK]

PBL: C und die erste Peripherie

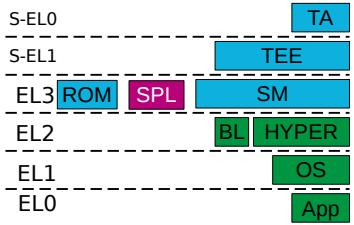
```
relocate_to_current_addr();
setup_c();
setup_uart();
```

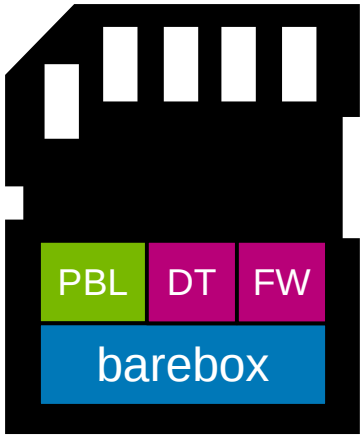


- C Umgebung aufsetzen
- Frühe serielle Ausgabe konfigurieren
  - imx8m_early_setup_uart_clock
  - imx8mm_setup_pad: Pin muxing
  - imx8m_uart_setup: z.B. standard Baudrate
  - pbl_set_putc: globalen Funktionspointer für Ausgabe hinterlegen

PBL: nachladen

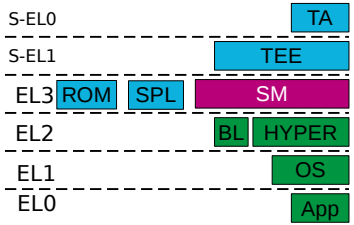
- PMIC konfigurieren
- SDRAM-Controller konfigurieren
- BootROM hat selbes Bootmedium ja schon konfiguriert
  → weiter benutzen statt komplette von neuem zu konfiguration
- BootROM-loaded PBL lädt vollen barebox nach von der selben Bootquelle
  - Serial → USB Gadget aufsetzen um weiter kommunizieren zu können
  - eMMC → volles barebox-Image nachladen
- Wie bootet man aber die weiteren CPU Kerne?





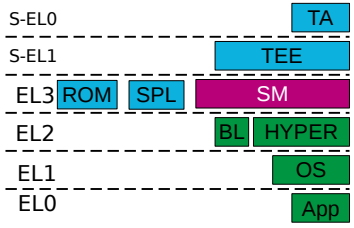
ARM PSCI

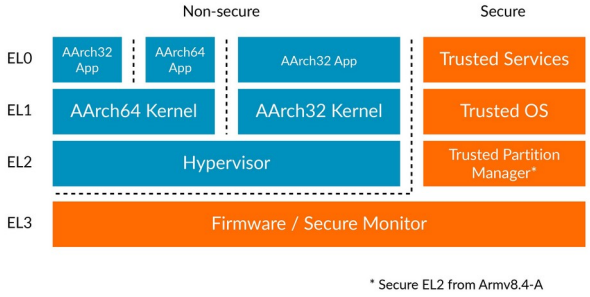
- Steuerung der weiteren Kerne sehr SoC-spezifisch
- Native und virtualisierte Kernel brauchen unterschiedliche Power Management Treiber
- Lösung: Generische Schnittstelle
  - Wenn vom Hypervisor aufgerufen (EL2), fällt man in den secure monitor (EL3)
  - Wenn vom Kernel aufgerufen (EL1), fällt man in den Hypervisor (EL2)
- Secure Monitor ist per Hardware-Mechanismen isoliert



TrustZone

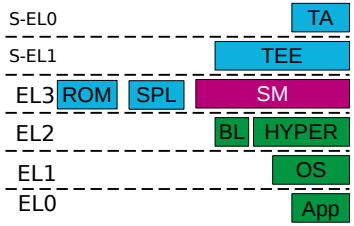
- Unterteilt System in secure and non-secure world
  - Linux and normale Applikationen laufen in der non-secure world
  - Non-secure world kann per Secure Monitor Call Instruktion geschützte Ressourcen anfragen
    - Power Management
    - Clock und Reset Treiber (SCMI)
    - Hacks, die der Hersteller nichts ins mainline Linux mogeln konnte



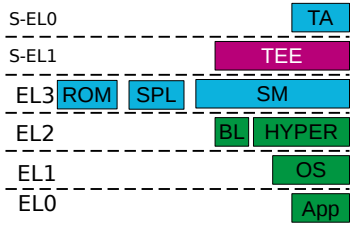


Trusted Firmware - A

- Referenz-Implementierung einer secure world firmware
- Was das genau bedeutet ist SoC-spezifisch. Was aber gemeinsam ist: Ausführung in der secure (trusted) world. Auf dem i.MX8M:
  - Stellt den Secure Monitor bereit
  - Übernimmt CPU Power Management über PSCI
  - Konfiguriert den Interrupt Controller
  - Lädt, und leitet Secure Monitor Calls weiter an, das Trusted Execution Environment (OS der secure world)



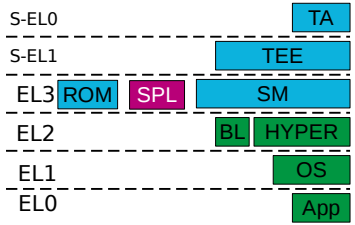
OP-TEE

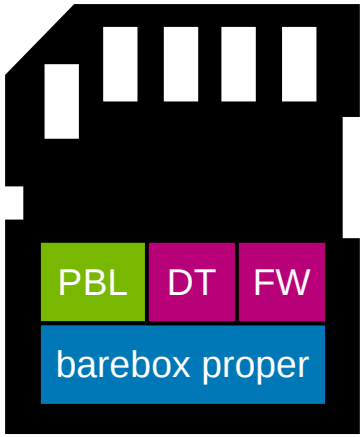


- Zunehmender Bedarf an manipulationssicheren trusted applications (TAs)
  - Handhabung sensitiver Daten, z.B. TLS-Schlüssel, Plattenverschlüsslung
  - Streamen kopiergeschützter Inhalte
  - firmware Trusted Platform Module (fTPM)
  - Verified/Secure Boot unterstützen
- Lösung
  - „trustlets“ in einem secure EL0 ausführen mit standardisierter API zur Kommunikation mit dem Trusted OS
  - Non-secure Userspace greift über Secure Monitor Calls auf Dienste zu
- OP-TEE ist eine open-source Implementierung der GP TEE Spezifikation

PBL: Umgang mit der TF-A

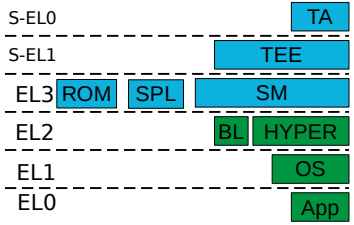
- TF-A Binary wird in barebox PBL reingelinkt
- PBL kopiert vollen barebox an die Rücksprungadresse der TF-A im SDRAM
- TF-A setzt secure monitor in EL3 auf und übergibt Kontrolle an barebox im non-secure EL2
- Dann läuft voller barebox und da es jetzt EL2 statt EL3 ist:
  - Initialisierung der MMU und Caches
  - Verifikation des barebox proper Hashes bei Verified Boot
  - barebox proper an den Ende des SDRAM extrahieren
  - barebox proper anspringen mit dem Device Tree





Device-Tree

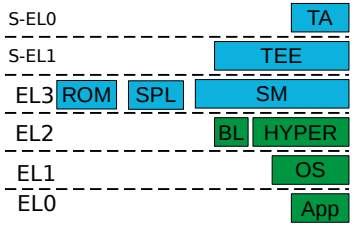
- Nicht alle Hardware ist dynamisch erkennbar
- Hardware-Beschreibung in C Code skaliert nicht
- Original Open Firmware Spezifikation beschreibt wie Hardware eigene Gerätebeschreibung und Treiber mitliefern kann
- Linux setzt auf eine Erweiterung der Hardware-Beschreibung um statische board Beschreibung im C code zu ersetzten, insbesonders auf ARM
  - Spezifikation wird jetzt separat gepflegt
  - Bindings und Device Trees sind weiterhin in erster Linie im Linux Kernel Repository gepflegt
    - wird aber auch benutzt von FreeBSD, barebox, U-Boot, TF-A, OP-TEE, Zephyr, …
    - nicht ARM-spezifisch: Auch verwendet auf RISC-V, PowerPC, MIPS, sogar paar x86, …
- Obligatorisch für alle modernen ARM Linux Plattformen



Device-Tree: Beispiel

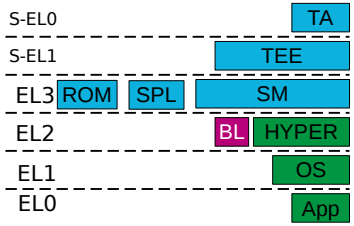
```
/* SoC device tree */
flexspi: spi@30bb0000 {
        compatible = "nxp,imx8mm-fspi";
        reg = <0x30bb0000 0x10000>, <0x8000000 0x10000000>;
        reg-names = "fspi_base", "fspi_mmap";
        interrupts = <GIC_SPI 107 IRQ_TYPE_LEVEL_HIGH>;
        clocks = <&clk IMX8MM_CLK_QSPI_ROOT>,
                 <&clk IMX8MM_CLK_QSPI_ROOT>;
        clock-names = "fspi", "fspi_en";
        #address-cells = <1>;
        #size-cells = <1>;
};
```

```
/* Board device tree */
&flexspi {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_flexspi>;

        flash@0 {
                reg = <0>;

                compatible = "jedec,spi-nor";
                spi-max-frequency = <80000000>;
                spi-tx-bus-width = <4>;
                spi-rx-bus-width = <4>;
        };
};
```



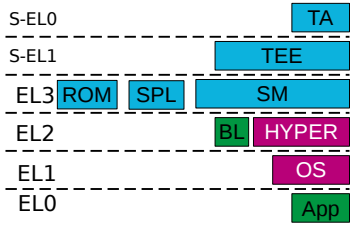
barebox proper

- Fast schon ein kleiner Betriebssystemkern
  - Meiste Treiber aus Linux portiert, kann DT parsen
  - Keine Interrupts, nur Kooperatives Multitasking
  - *nix Abstraktionen: virtual file system, block layer, character devices, file descriptors und Shell
- Aber mit viel Funktionalität um Boot und Entwicklung zu vereeinfachen
  - Redundante Watchdog-überwachte Bootsequenzen
  - Geteilter Bootstate mit OS für Variablenaustausch
  - Hardware Manipulation direkt von der Shell
  - Dynamisches Anpassen des Device Trees
  - Verified Boot
  - Alles in einer einzelnen Binary



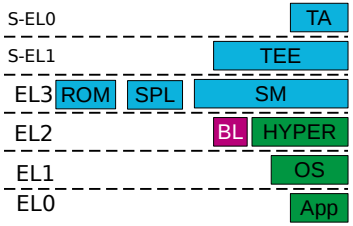
Linux: kernel image aufrufen

- Bootloader muss:
  - Interrupts maskieren
  - Den standard ARM timer initialisieren
  - Kernel laden an den im Header beschriebenen Offset
  - MMU ausschalten sowie Data Caches
  - CPU Register Initialisieren entweder für EL2 oder EL1
    - z.B. x0 für den Device Tree Blob
- Sprunginstruktion

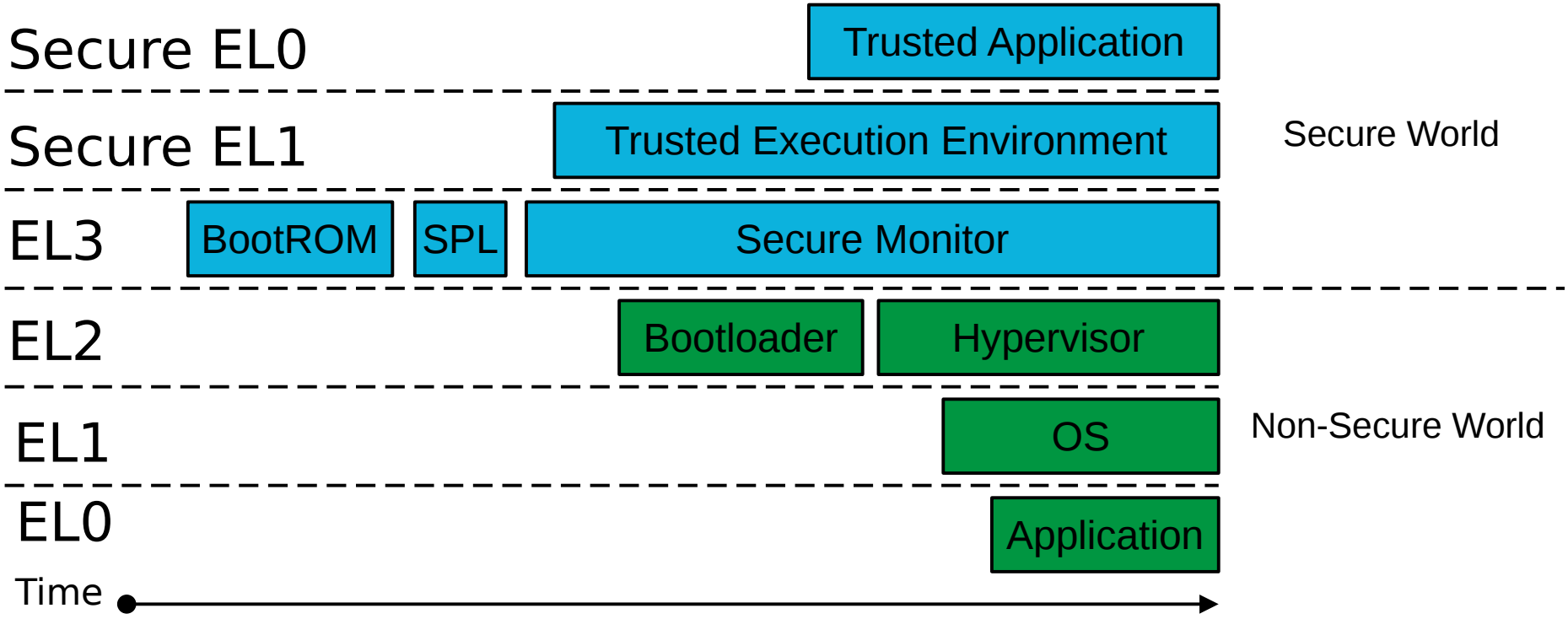


Was fehlt noch? UEFI auf ARM

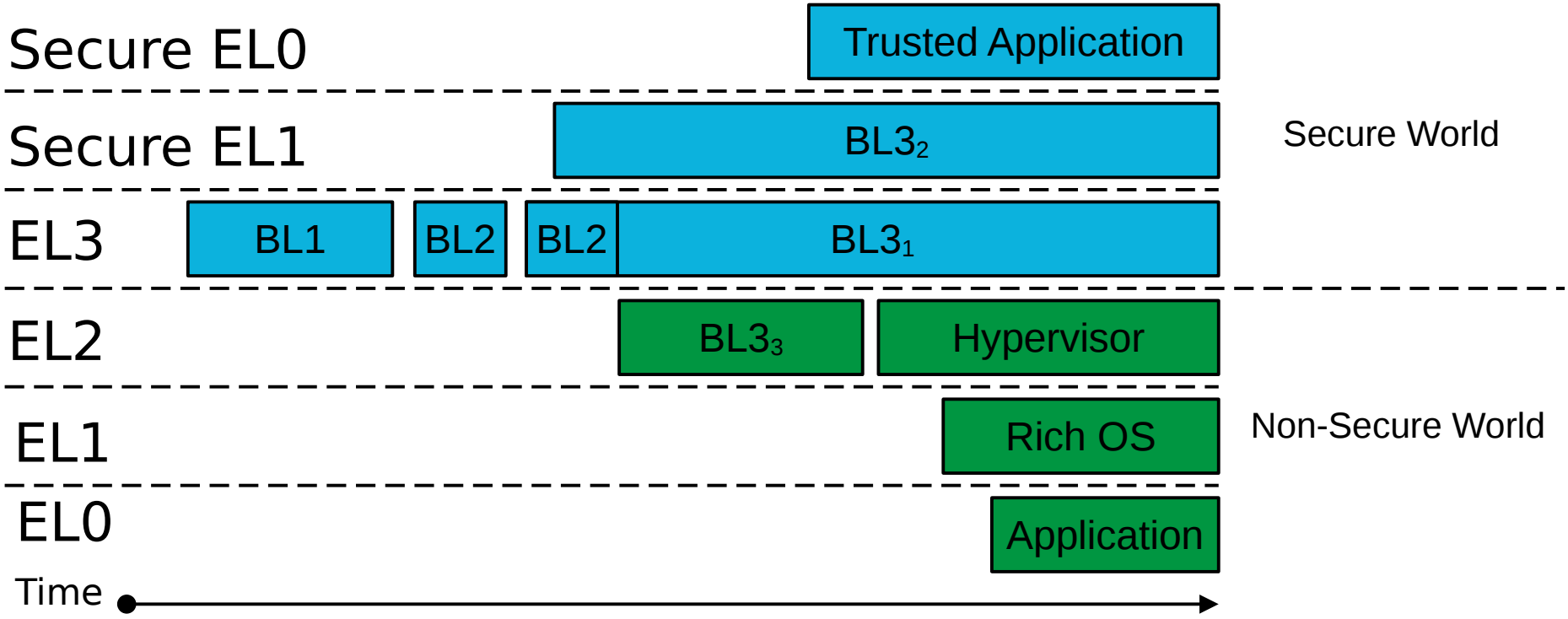
- Distros mögen ihren GRUB
  → Bootloader (U-Boot, Tianocore) können ein (minimales) UEFI bereitstellen
  → UEFI payload dann einfach für ARM kompilieren
- UEFI ist fast ein eigenes operating system
  - Embedded/Server Base Boot Requirements (EBBR/SBBR) definiert eine Teilmenge
- Für Hersteller eingebetter Hardware:
  - Löst einige schon gelöste Probleme anders
  - Man ist immer noch für den ganzen Stack verantwortlich → Jede extra Komponente ist extra Wartungsaufwand
- Für kommerzielle SBCs:
  - Nützlich um general purpose distros direkt zu booten
  - Kann aber noch nicht die harten Probleme lösen ohne dedizierten Flash für Firmware
    - Keine generische Laufzeitablage von Variablen
    - Wie führt man ein fail-safe GRUB Update durch?
- Für ARM Server:
  - Noch ein Schritt weiter: ACPI erweitert um device tree properties
  - Endziel: Booten von alten long term stable Distros auf neuer Server Hardware



Bootablauf


Secure EL0
Trusted Application
Secure EL1
Trusted Execution Environment
Secure World
EL3
BootROM
SPL
Secure Monitor
EL2
Bootloader
Hypervisor
EL1
OS
Non-Secure World
EL0
Application
Time

Bootablauf (in ARM-Nomenklatur)



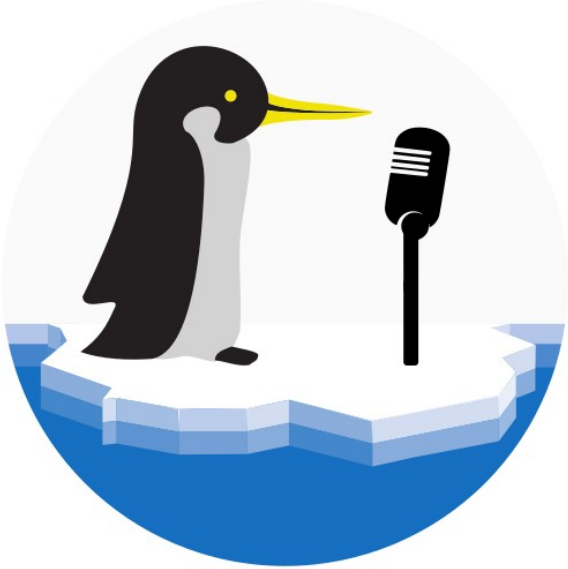
Neugierig nach mehr?

- DOOM portieren für Einsteiger - Heavy Metal auf Bare Metal (Ahmad Fatoum)
- HOWTO build a product with OP-TEE (Rouven Czerwinski)
- Runtime Services: Share System Resources on Multi-Processor System (Lionel Debieve)
- Wenn Geräte an Bäumen wachsen: Linux-Device-Tree-Portierung (Ahmad Fatoum)
- EBBR: Standard Boot for Embedded Platforms (Alexander Graf, Grant Likely)

Neugierig nach mehr?

Kernel Panic: Board on Fire

Unser Podcast mit low-level Lagerfeuergeschichten



Bildquellen

- [MNT Reform]: https://www.crowdsupply.com/mnt/reform
- [Purism Librem 5]: https://puri.sm/products/librem-5/
- [EVK]: https://static6.arrow.com/aropdfconversion/3eb66c6ff328259ad060ed3a1cb5b2e34dcaa693/imx8mmevkhug.pdf
- [OMAP]: https://www.yumpu.com/en/document/view/18631588/chapter-26-initializationpdf
- [MX6-CLK]: https://www.shuzhiduo.com/A/mo5kwRnn5w/
- [ARM-EL]: https://developer.arm.com/architectures/learn-the-architecture/exception-model/execution-and-security-states

Danke!

Fragen?

(Gerne auch beim Stand vorbei kommen!)