東京 2025

LINUX PLUMBERS CONFERENCE

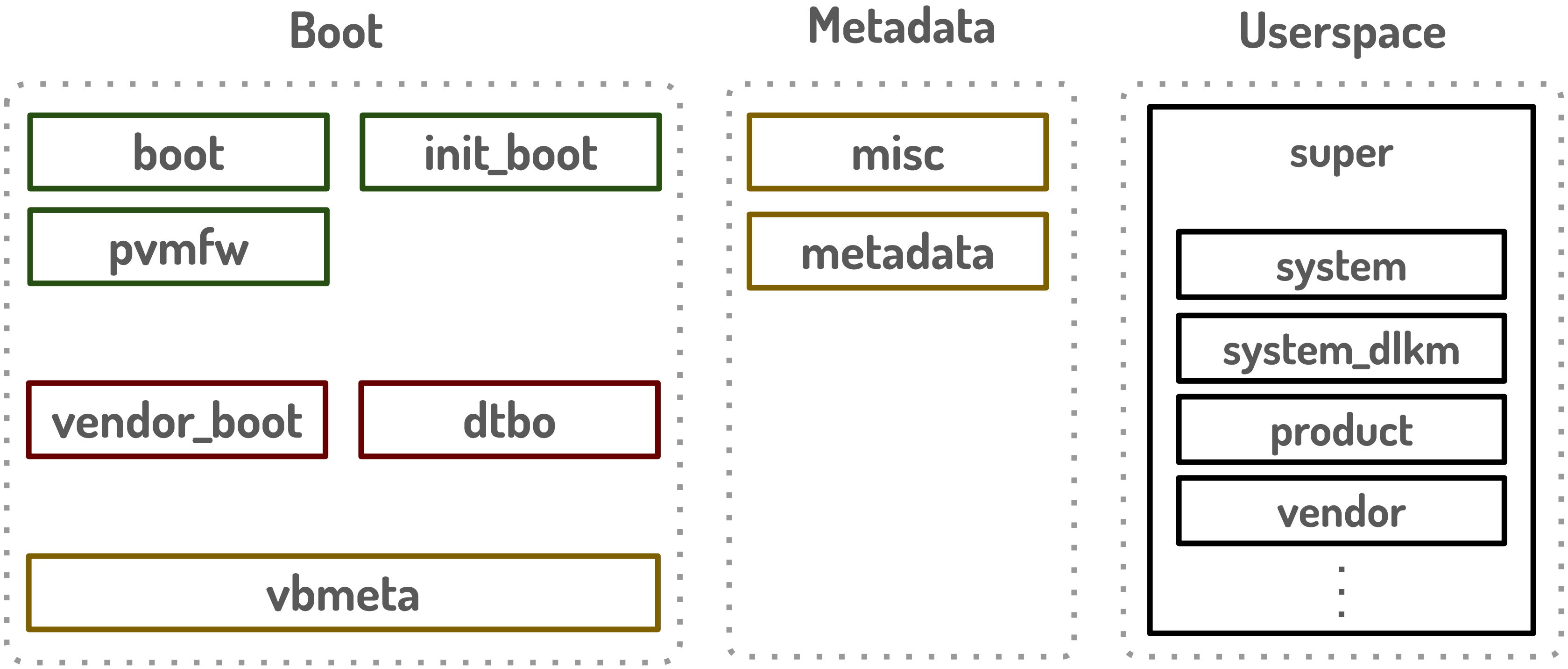
TOKYO, JAPAN / DECEMBER 11-13, 2025

Android Boot, DRTM, UKIs

Dmitrii Merkurev <dimorinny@google.com>
Leif Lindholm <leif.lindholm@oss.qualcomm.com>
Ram Muthiah <rammuthiah@google.com>

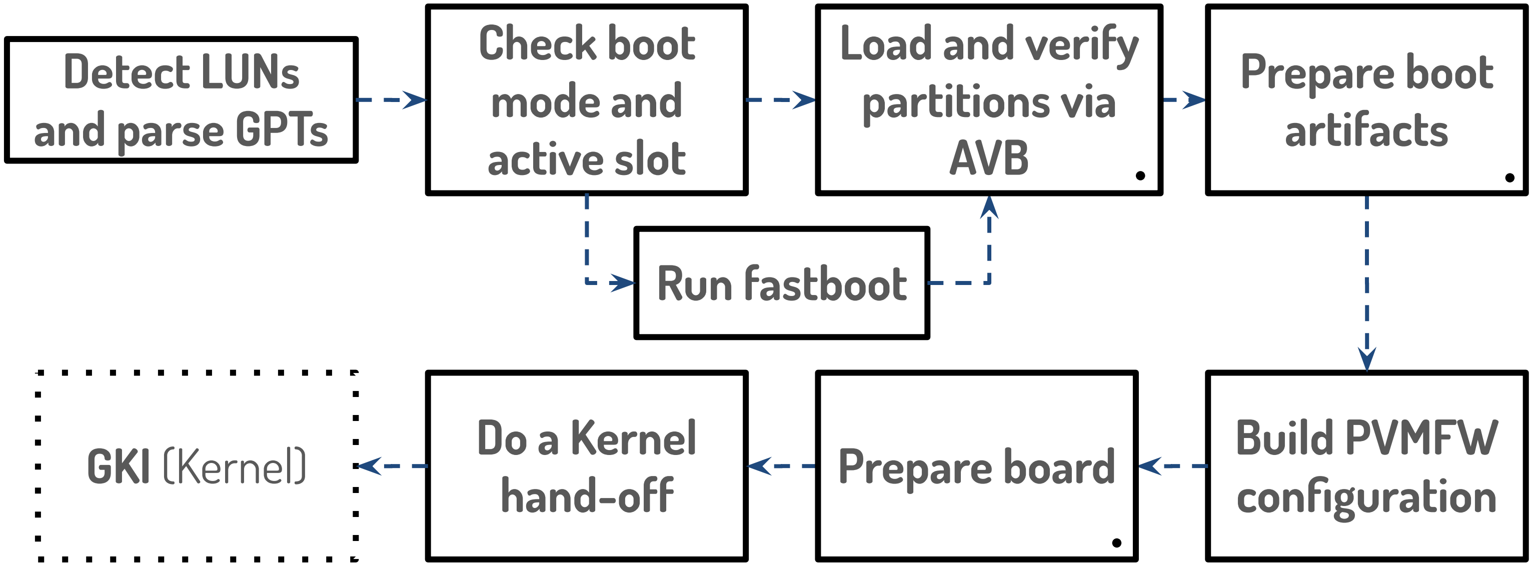
東京 2025
LINUX PLUMBERS CONFERENCE
TOKYO, JAPAN / DEC. 11-13, 2025

Android partitions

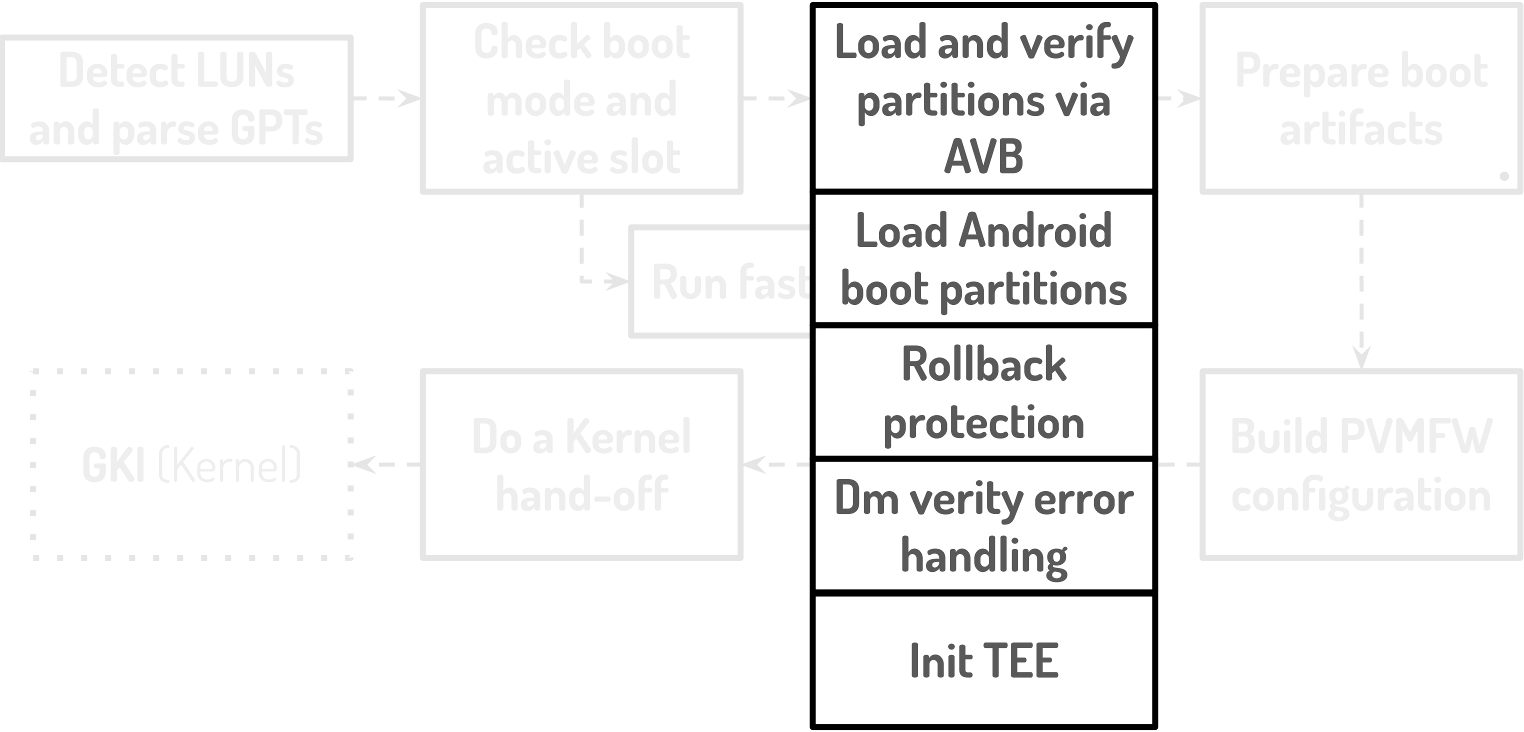


🟩 - Google provided. 🟫 - Mixed ownership. 🟥 - Vendor specific.

Android flow

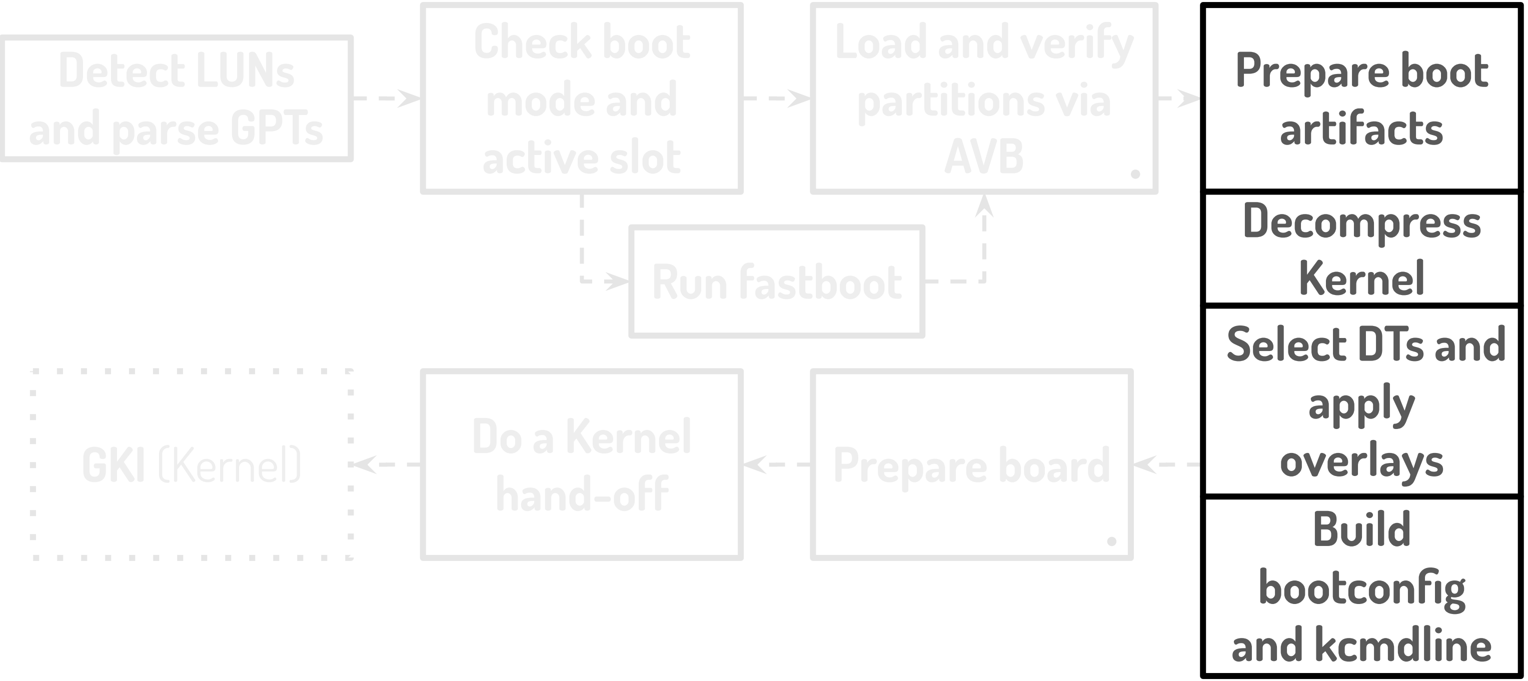

Detect LUNs and parse GPTs
Check boot mode and active slot
Load and verify partitions via AVB
Prepare boot artifacts
Run fastboot
Build PVMFW configuration
Prepare board
Do a Kernel hand-off
GKI (Kernel)

Android flow

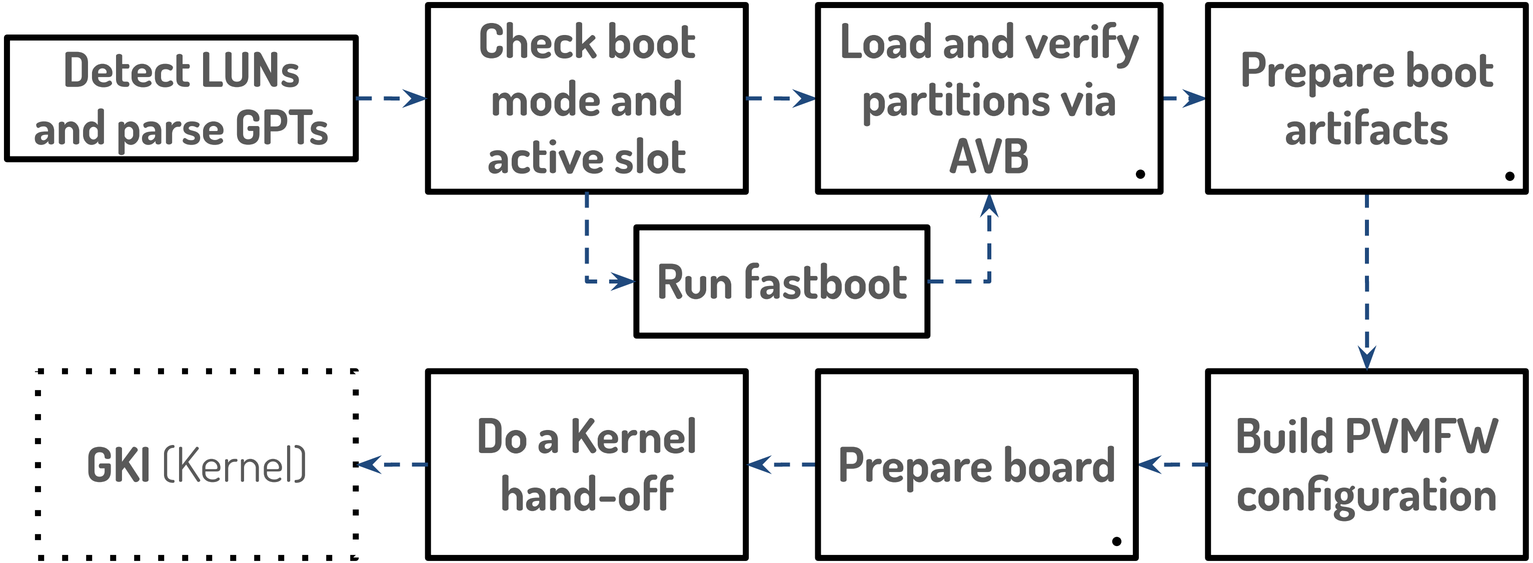

Detect LUNs and parse GPTs
Check boot mode and active slot
Run fast
Load and verify partitions via AVB
Load Android boot partitions
Rollback protection
Dm verity error handling
Init TEE
Prepare boot artifacts
Build PVMFW configuration
Do a Kernel hand-off
GKI (Kernel)

東京 2025
LINUX PLUMBERS CONFERENCE
TOKYO, JAPAN / DEC. 11-13, 2025

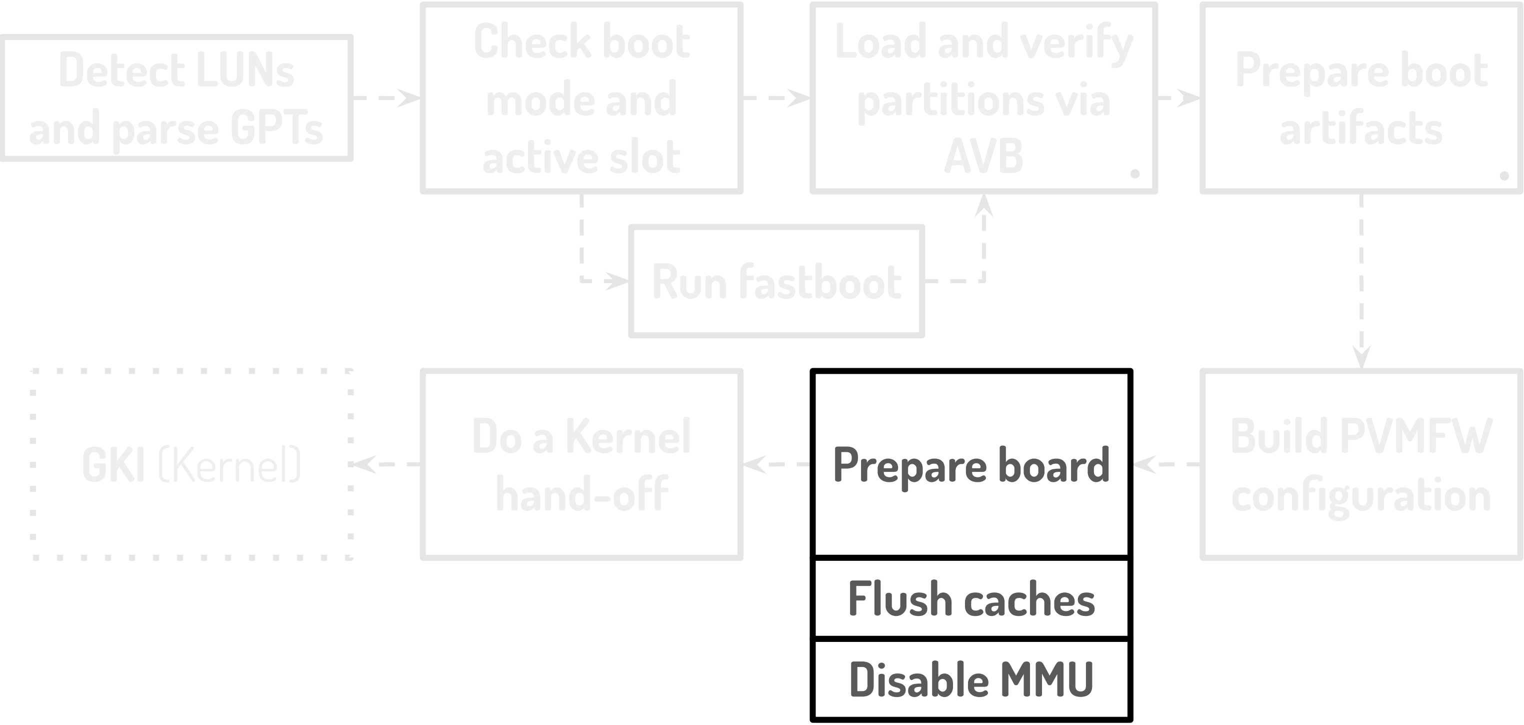
Android flow


Detect LUNs and parse GPTs
Check boot mode and active slot
Load and verify partitions via AVB
Run fastboot
Prepare boot artifacts
Decompress Kernel
Select DTs and apply overlays
Build bootconfig and kcmdline
Prepare board
Do a Kernel hand-off
GKI (Kernel)

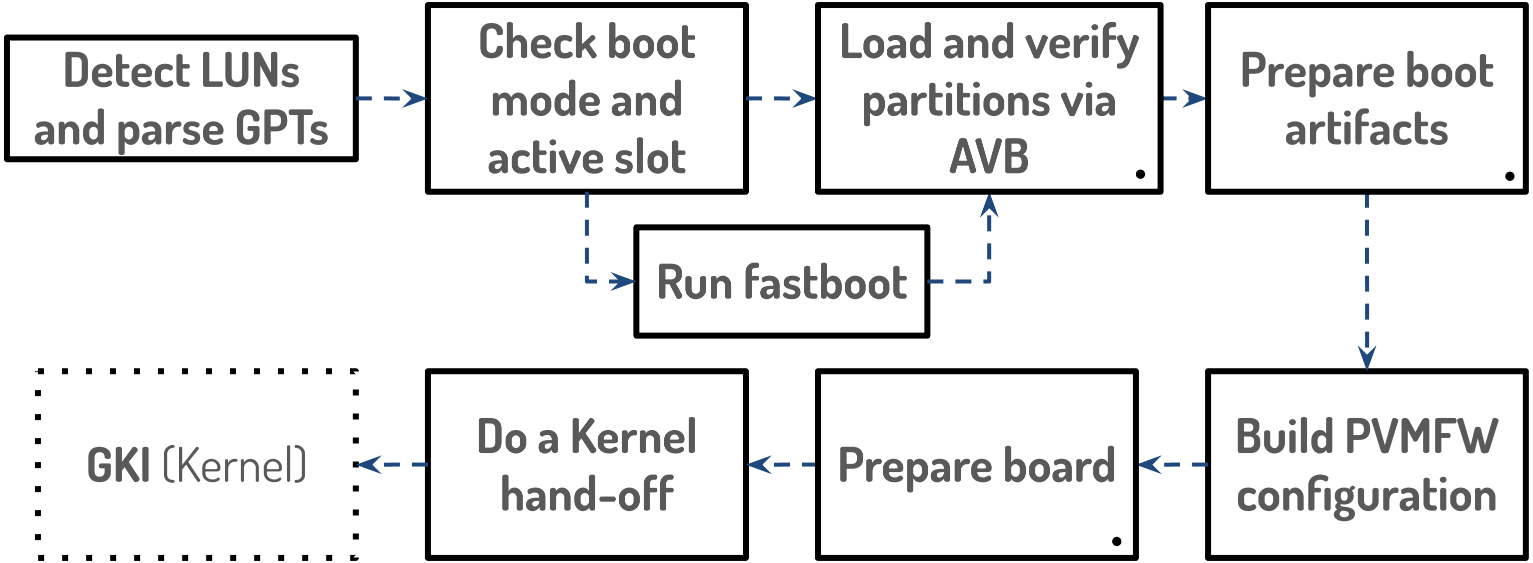
Android flow


Detect LUNs and parse GPTs
Check boot mode and active slot
Run fastboot
Load and verify partitions via AVB
Prepare boot artifacts
Build PVMFW configuration
Prepare board
Do a Kernel hand-off
GKI (Kernel)

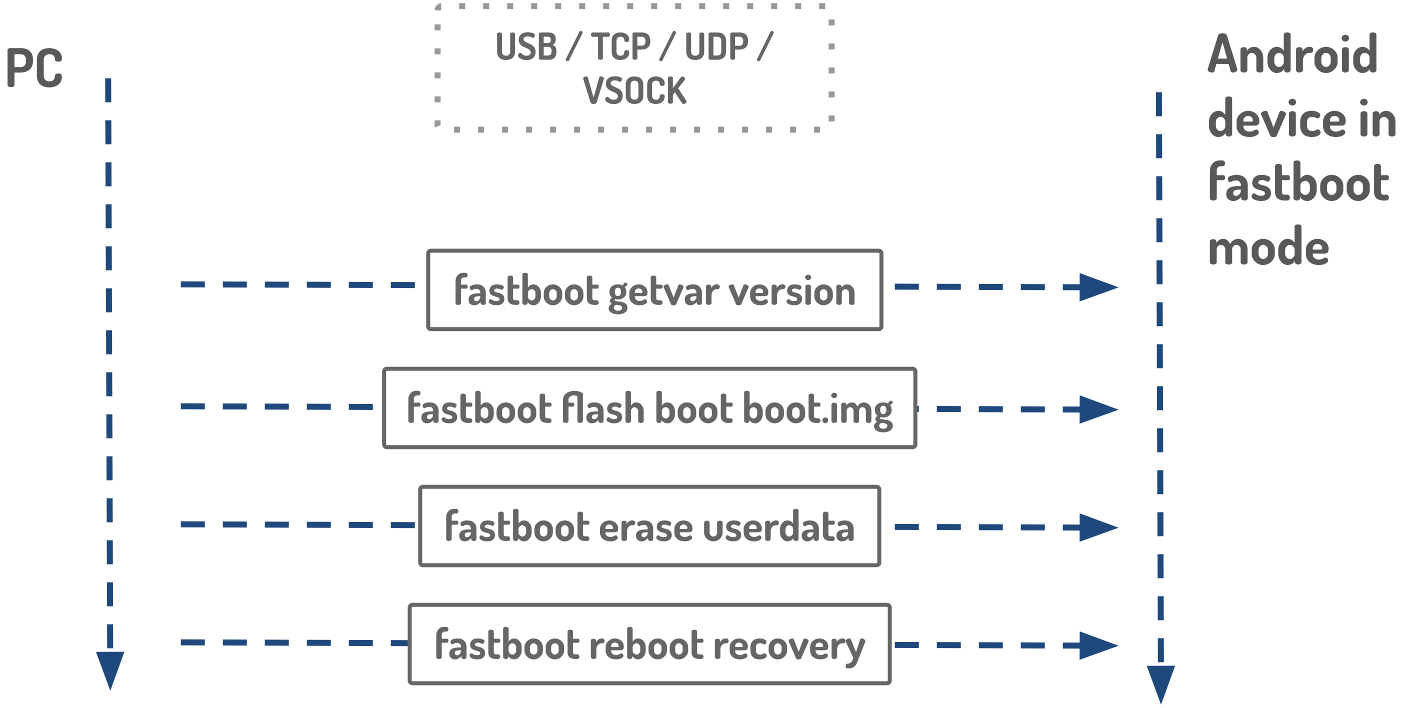
Android flow


Detect LUNs and parse GPTs
Check boot mode and active slot
Load and verify partitions via AVB
Prepare boot artifacts
Run fastboot
Build PVMFW configuration
Prepare board
Flush caches
Disable MMU
Do a Kernel hand-off
GKI (Kernel)

Android flow


Detect LUNs and parse GPTs
Check boot mode and active slot
Load and verify partitions via AVB
Prepare boot artifacts
Run fastboot
GKI (Kernel)
Do a Kernel hand-off
Prepare board
Build PVMFW configuration

Fastboot



```
fastboot getvar version
fastboot flash boot boot.img
fastboot erase userdata
fastboot reboot recovery
```

Introduce **GBL**

Generic bootloader (GBL) is Android boot flow UEFI application provided by Google.

The main value:

For partners, ecosystem:

- Reduce the vendor’s integration burden
- Provide production ready open source Android boot flow reference implementation

For Google:

- Faster uptake of Android Boot changes by partners

Introduce **GBL**

- no_std Rust UEFI app (dynamic allocations use UEFI)
- Built by **BAZEL**
- Support arm64 / x86_64 / riscv64 architectures
- Available as a part of **AOSP**, so fully open source
- Statically compiled against trusted components (ATF, libavb, boringssl, libfdt, libufdt)
- Shipped as a part of dedicated **efisp** _a/_b partition
- Already can be used to boot Cuttlefish and major vendors dev boards

Why **UEFI**?

- Already adopted by major partners on production
- Supported by major firmwares (**EDK2, LK, U-Boot**)
- **UEFI protocols** is a flexible way to implement vendor-specific logic
- Offers a variety of existing standardized interfaces for use such as block devices, network, etc
- The UEFI runtime spec is stable. Version 2.10 in use for more than 10 years.
- Advocated by ARM's **SystemReady** initiative

Standard **UEFI protocols** used by GBL

- **Block IO (sync, async)**
  - EFI_BLOCK_IO_PROTOCOL
  - EFI_BLOCK_IO2_PROTOCOL
  - EFI_BLOCK_ERASE_PROTOCOL
- **Network (fastboot)**
  - EFI_SIMPLE_NETWORK_PROTOCOL
- **RNG (rng-seed, kaslr)**
  - EFI_RNG_PROTOCOL
- **Crypto (override boringssl)**
  - EFI_HASH2_PROTOCOL
- **Logging**
  - EFI_LOADED_IMAGE_PROTOCOL
  - EFI_DEVICE_PATH_PROTOCOL
  - EFI_SIMPLE_TEXT_INPUT_PROTOCO
  - EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL
- UEFI **memory allocation** service API

UEFI protocols introduced by GBL

- GblEfiBootControlProtocol
  - **A/B slots**, **boot modes, kernel handoff** implementations.
- GblEfiBootMemoryProtocol
  - Control GBL's dynamic **memory allocations** and pre-define vendor-specific offsets for boot artifacts.
- GblOsConfigurationProtocol
  - **Select DTs**, propagate **bootconfig** with vendor-specific details.
- GblEfiAvbProtocol
  - Handles vendor-specific aspects of **Android Verified Boot** (PK verification, rollback protection, TEE initialization).
- GblEfiAvfProtocol
  - Prepare **AVF configuration** based on vendor-specific data (**DICE chain**, **Secret Keeper PK**).
- GblEfiDebugProtocol
  - Handle GBL errors in a vendor-specific way.

UEFI protocols introduced by GBL

- GblEfiFastbootProtocol
  - Device locking, vendor-specific properties, commands.
- GblEfiFastbootTransportProtocol
  - Vendor-specific transports (**USB**, local transport for **fastboot UI**).

- EfiDtFixupProtocol*
  - Inspect final **DT** and modify it in a vendor-specific way.

Links

- GBL
  - Build artifacts
  - Development
  - Deploy updates
- LK UEFI implementation
- EDK2 GBL protocols headers (WIP)
- EDK2 DT_FIXUP protocol header (WIP)

DRTM

- Protect against early stage firmware compromises by resetting the chain of trust to a trusted element after the boot firmware completes and then performing a measured boot within a reduced TCB state (by disabling all other cores and DMAs)
- Why consider DRTM now?
  - Boot firmware is now in EL2 as Protected Virtualization becomes a requirement for Android
  - Mitigates existing vulnerabilities in boot firmware projects

DRTM

- How could Android adopt the ARM DRTM spec?
  - ACPI is used, DeviceTree isn't.
  - The many implementation of Android mean a singular DLME isn't feasible.
  - GBL (UEFI) does measurements through Android Verified Boot
    - Could these be repeated in the Dynamic Measurement Environment?
  - One Constraint - this implementation is ideally architecture agnostic

ACPI

Or we could switch Android to booting with ACPI

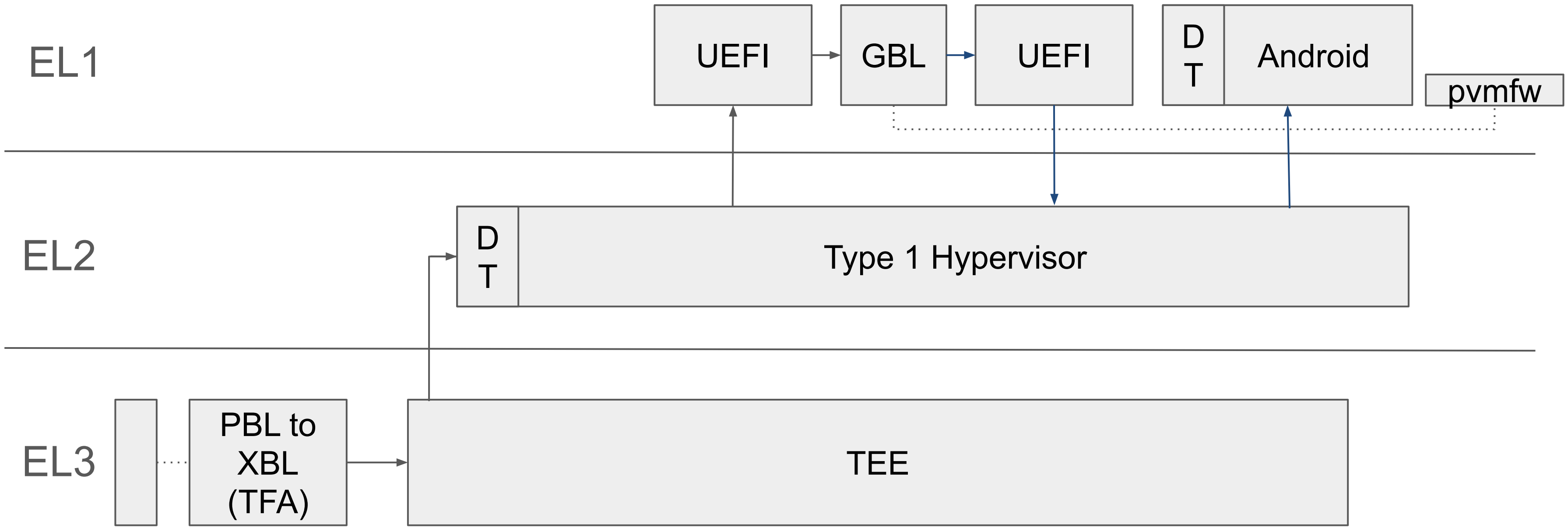
This brings with it a couple of problems:

- Lack of the very granular power management required for good battery life*
- Would violate the Peace of Westphalia

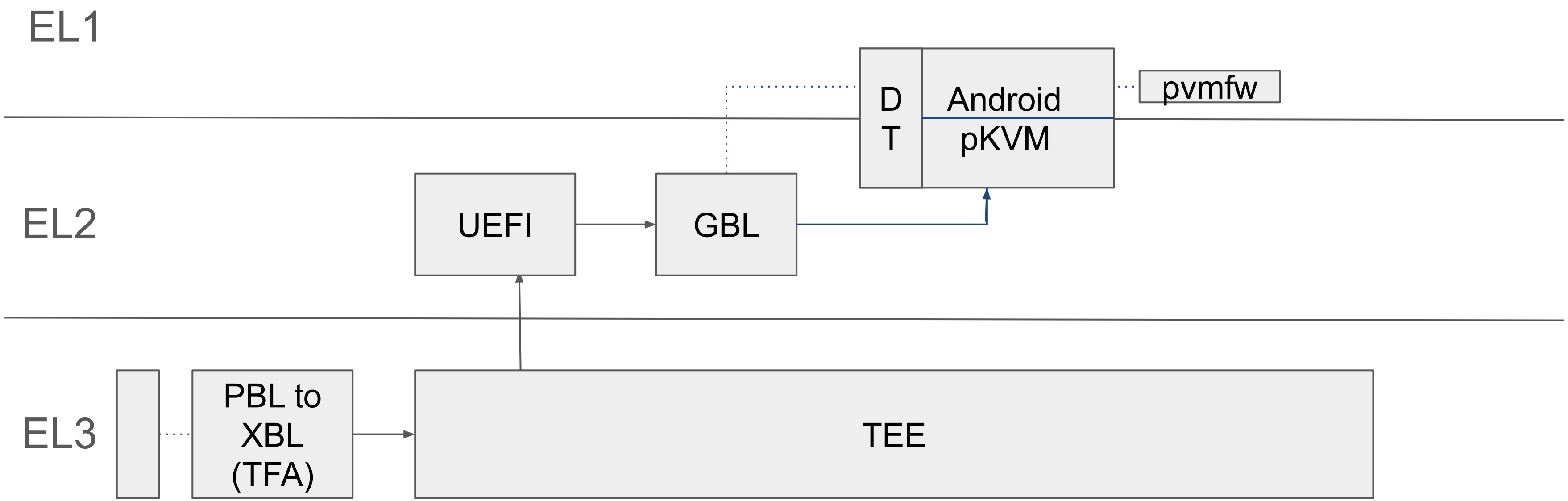
*But that is a problem that needs resolving anyway.

Already in use on UEFI systems for Windows (System Guard Secure Launch), and .

Android Boot w/ a Hypervisor



Android Boot w/ pKVM



Device Tree

- ACPI can’t co-exist
- Options
  - Don’t Measure the Final Device Tree
  - Diff the Final Device Tree against the pre-patched one and check a blocklist
    - watchdog, memory regions, etc
  - Acquire all patched values during the measured boot and have a fixed DT
- Adopt a similar scheme for bootconfig and kcmdline patching

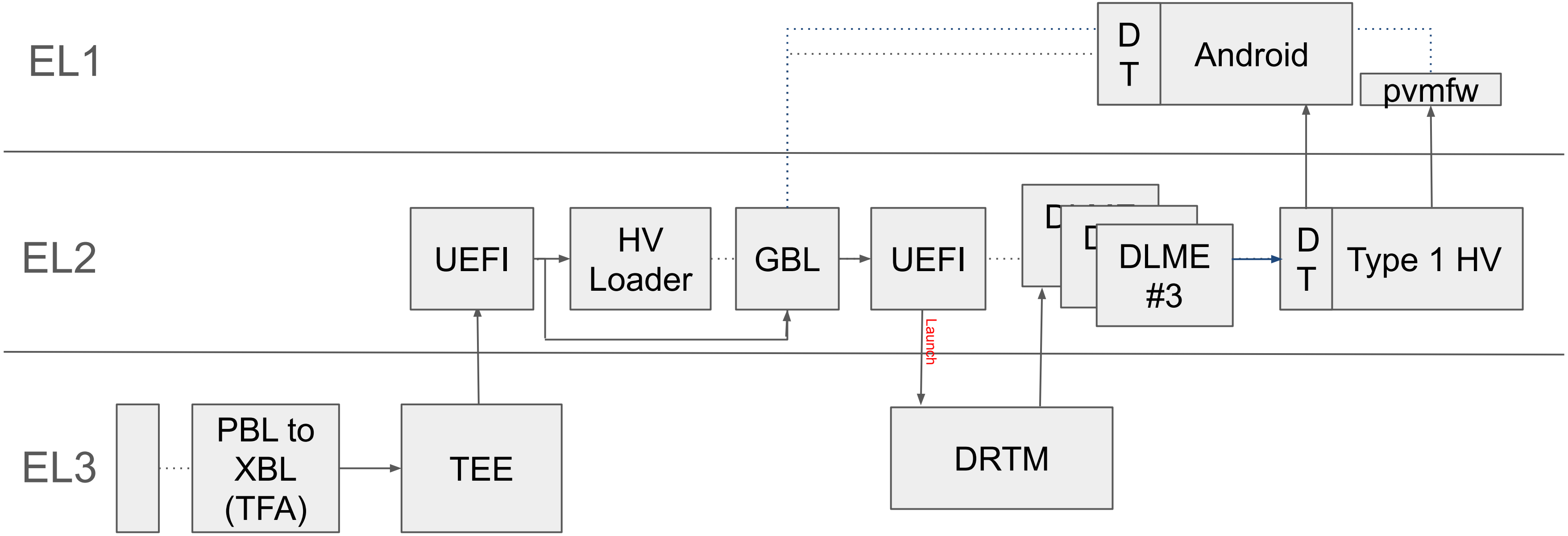
DLME

- Minimal new software architecture or device specifics - SMC calls
  - Let's ignore AMD and Intel (for now)
- Capability to measure hypervisors and other custom payloads in memory
- Hardware acceleration for measurements (offboard specifically)
- Options
  - multi-stage DLMEs
  - single DLME that's distributed by Android and forked per platform

Multi-Stage DLMEs

- Android devices have many flavors ..... even if shipping the same platform
- LTS is **critical** for Android
- Split DLMEs enable Android Platform boot logic to be updated independently of Device Specific boot logic

Multi-Stage DLMEs



Open Problems

- If adopting DT - the calling convention for ARM's DLME spec has to be updated to include DT.
- Android specifics have to be acquired during the measurement phase
  - RNG seeds, AVB public keys + Rollback indices, Boot Modes + Reasons, RD Selection, DT Selections + Patches, MTE settings, DICE handovers + DTs and patches for the pvms at least
- Strict rules about handoff between DLME stages
  - Last one will have to initialize the boot CPU, enable the caches and MMU, and set up virtual memory mappings
- What payloads should be measured? And how will attestation occur?
  - Kernel, Pre-Final Device Tree + Ramdisks, HV, vm Kernel, vm DT

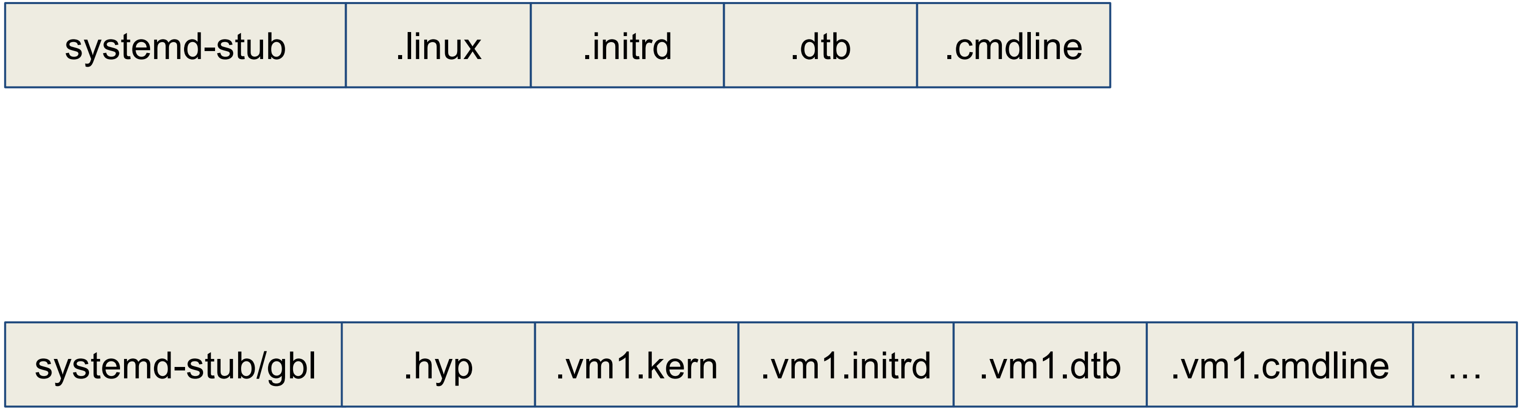
UKIs

From Qualcomm Technologies' side, we have been working on moving the Gunyah™ Type-1 hypervisor out of firmware and towards standards-compliant boot. Also required for Protected Virtualization

- One of the opens is that there's no real standard for booting Linux (and hence Android) systems with Type-1 hypervisors.
- Red Hat's nmbl project make use of systemd-stub and associated tools to generate a UKI
  - What if we extend that with a hypervisor type?
- Oh hey, GBL could slot right into that systemd-stub slot in the diagram, couldn't it?

Gunyah is a product of Qualcomm Innovation Center, Inc.

UKI image layout



systemd-stub	.linux	.initrd	.dtb	.cmdline

systemd-stub/gbl	.hyp	.vm1.kern	.vm1.initrd	.vm1.dtb	.vm1.cmdline	…

Thank you! Questions?

- UEFI in Android
- DeviceTree for ARM DRTM
- multi-stage DLME
- GBL as an EFI Stub