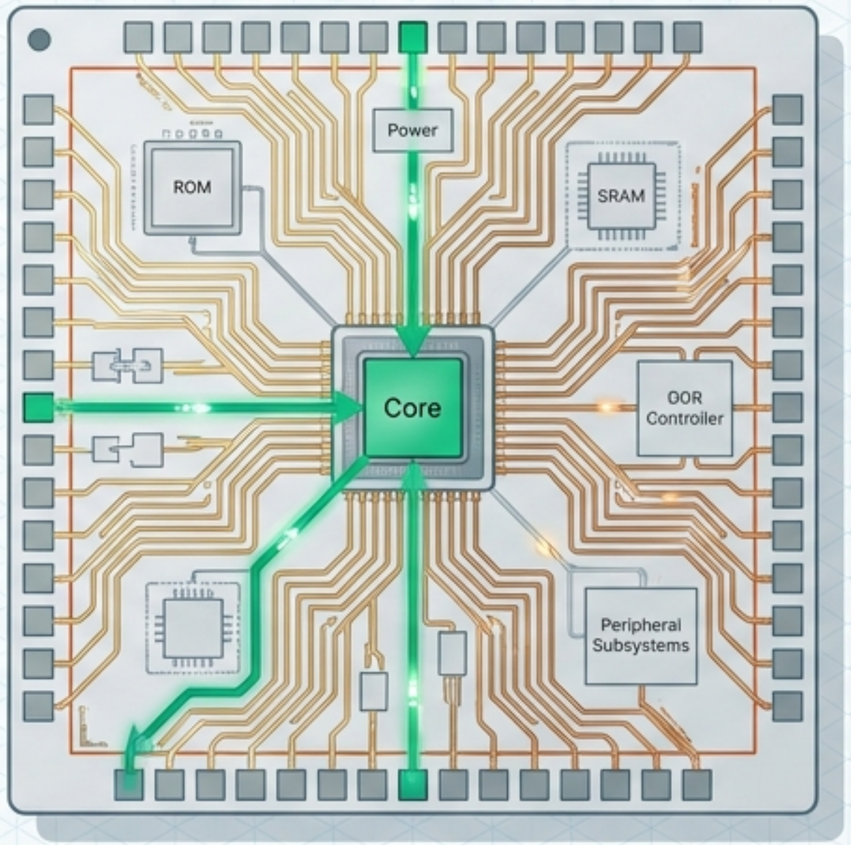
BOOTLOADERS 101: FROM POWER-ON TO LINUX

Demystifying the “Chaos” of the ARM Embedded Boot Flow



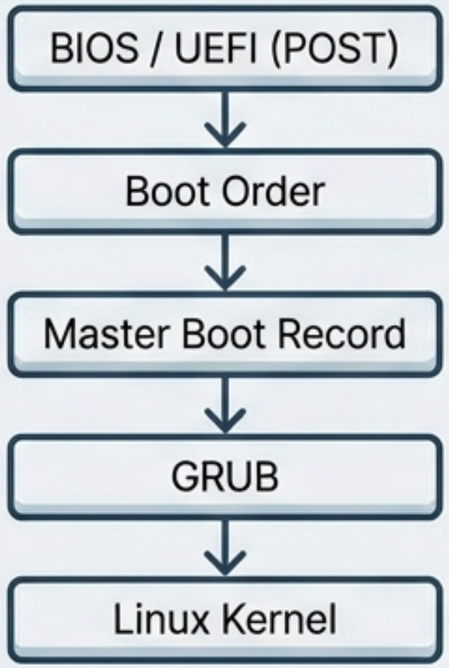
Embedded boot flows often feel like black magic. This deck deconstructs the process, moving from the confusion of high-level ARM flexibility to the concrete electron path of a specific chip (TI K3/AM625).

The Core Mandate: Find, Load, Run

Despite the complexity of modern architecture, a bootloader has one singular purpose. It is a piece of software designed to execute a simple loop, often split into multiple phases to phases to accommodate hardware restrictions.

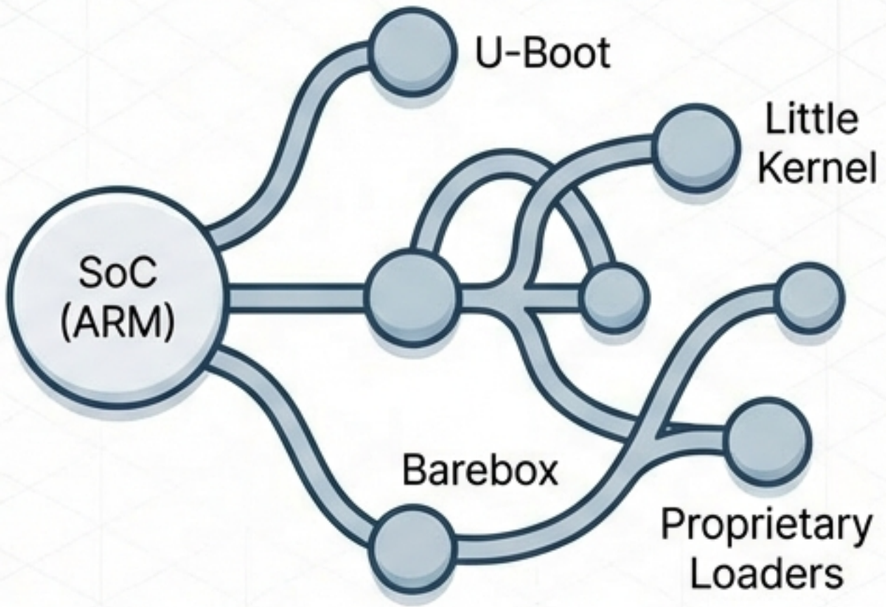
The Comfort of x86 vs. The 'Chaos' of ARM

x86 (The Standard)



Predictable. Standardized handoffs allow one OS image to boot on any PC.

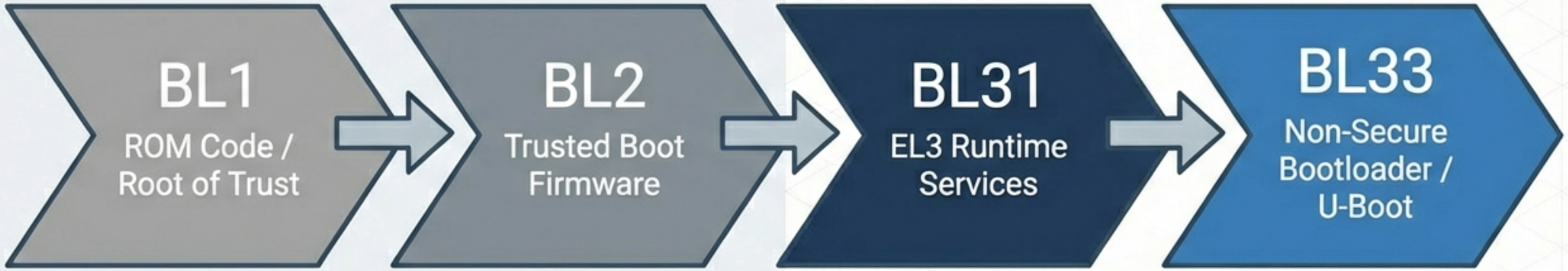
ARM (The Wild West)



Flexible but chaotic. No enforced standard. Manufacturers (TI, NXP, Rockchip) implement unique flows tailored to specific chip constraints.

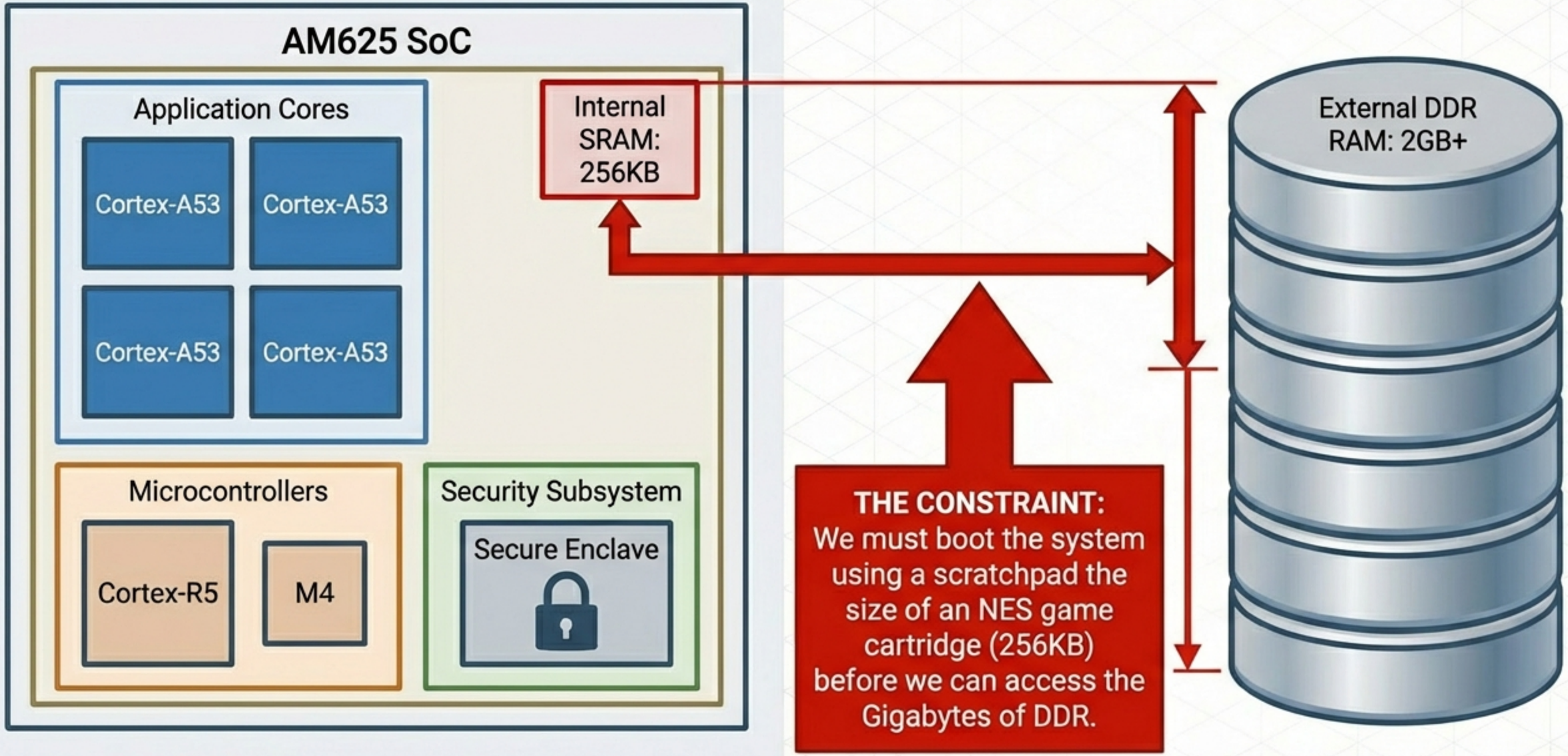
Consequence: An OS image for Raspberry Pi 3 will not boot on a Raspberry Pi 4. Every board requires a tailored distribution.

The ARM Reference Flow

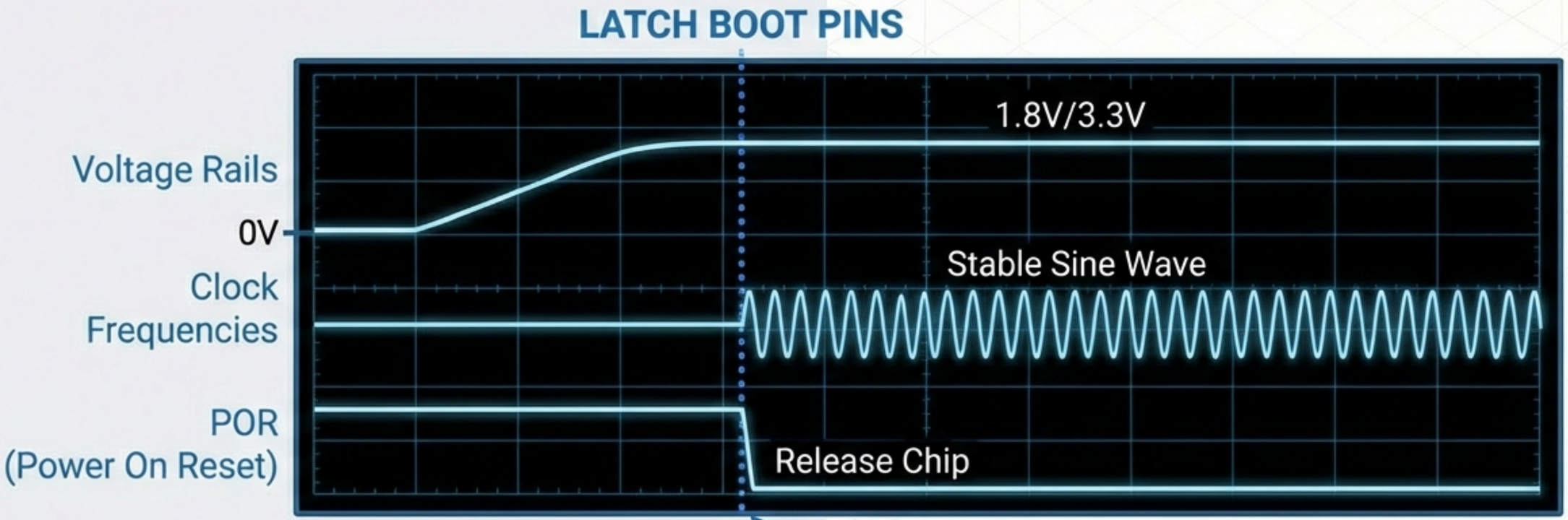


The Manufacturer's Twist: ARM provides this "Reference Boot Flow" (Trusted Firmware-A) as an ideal. However, manufacturers like Texas Instruments often "remix" this flow to balance cost, physical die size, and security features. The map is not the territory.

Case Study: The TI K3 Architecture (AM625)

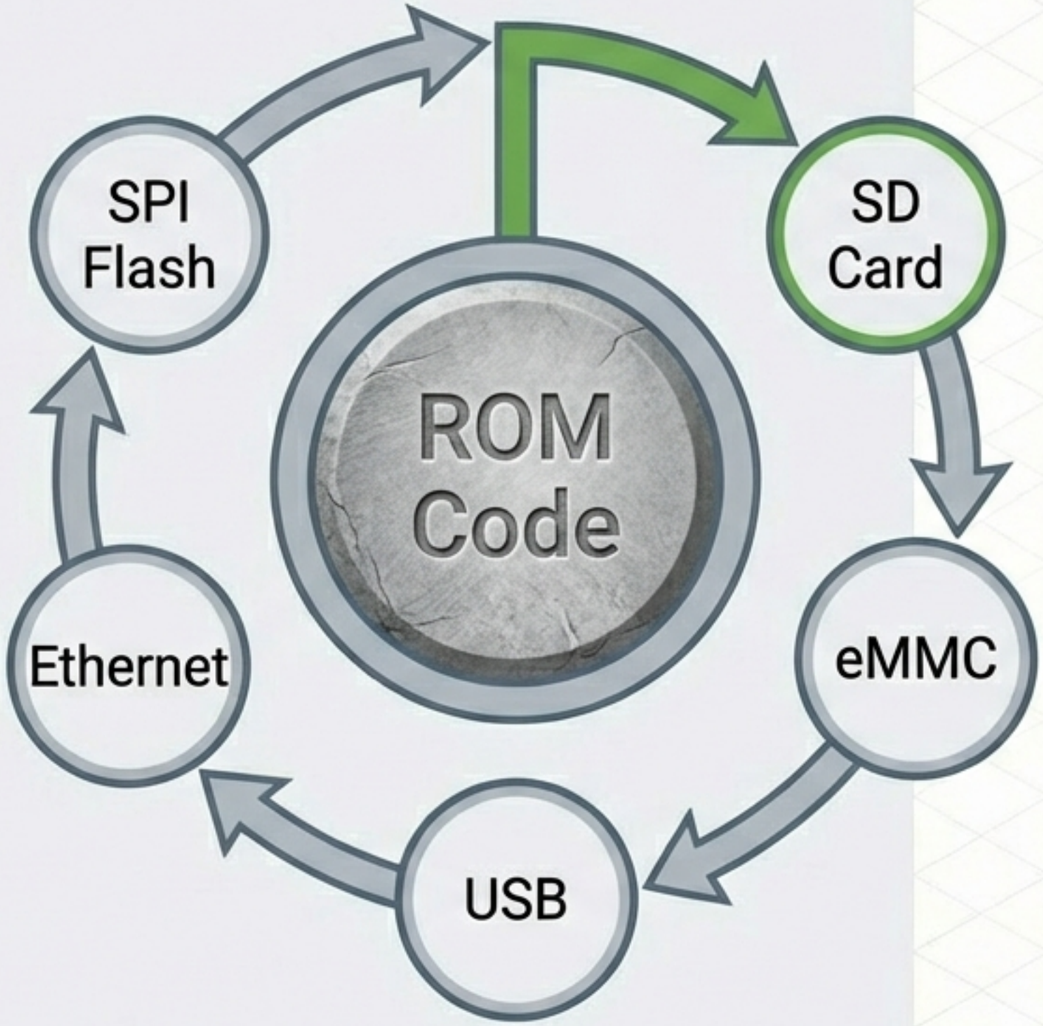


Step 0: Power On and Physical Stability



The Hardware Handshake:
Before code runs, physics happens. The chip latches physical pins to determine its identity.
- Where is the code? (USB? SD Card? SPI?)
- What is the crystal frequency?

Step 1: The ROM Loader (The Immutable Code)



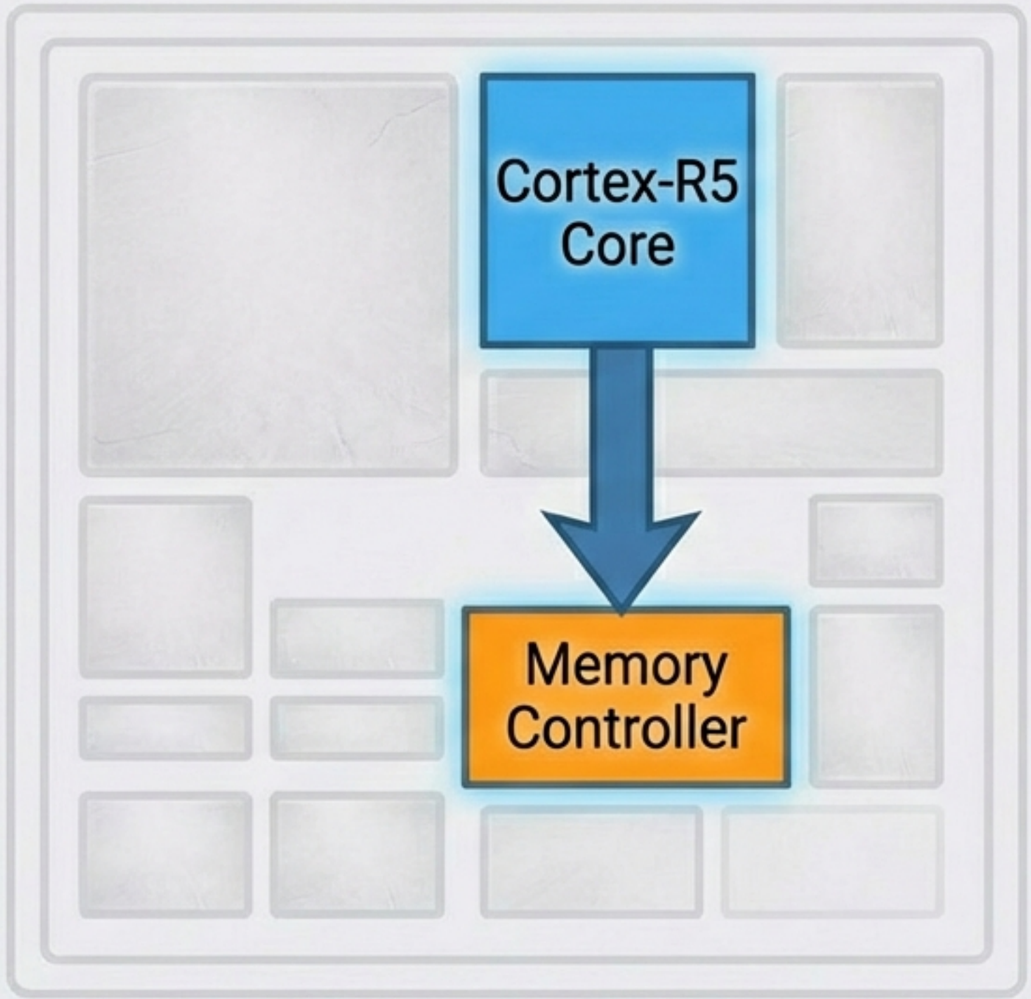
Mission: Find “tiboot3.bin”

- The ROM code is etched into the silicon at the factory. It cannot be patched.
- It is “dumb” but persistent. It cycles through the boot interfaces defined in Step 0 until it finds a specific file named “tiboot3.bin”.
- **Risk:** A bug here is permanent. It forces a complete chip re-spin.

Step 2: The Security Subsystem (Root of Trust)

Paranoia as a Feature: We assume ROM might be compromised. The Security Subsystem validates the loaded binary against keys physically etched into the chip.
Firewalls: If the code attempts to touch forbidden memory, the bus is killed immediately.

Step 3: R5 SPL – The First Software

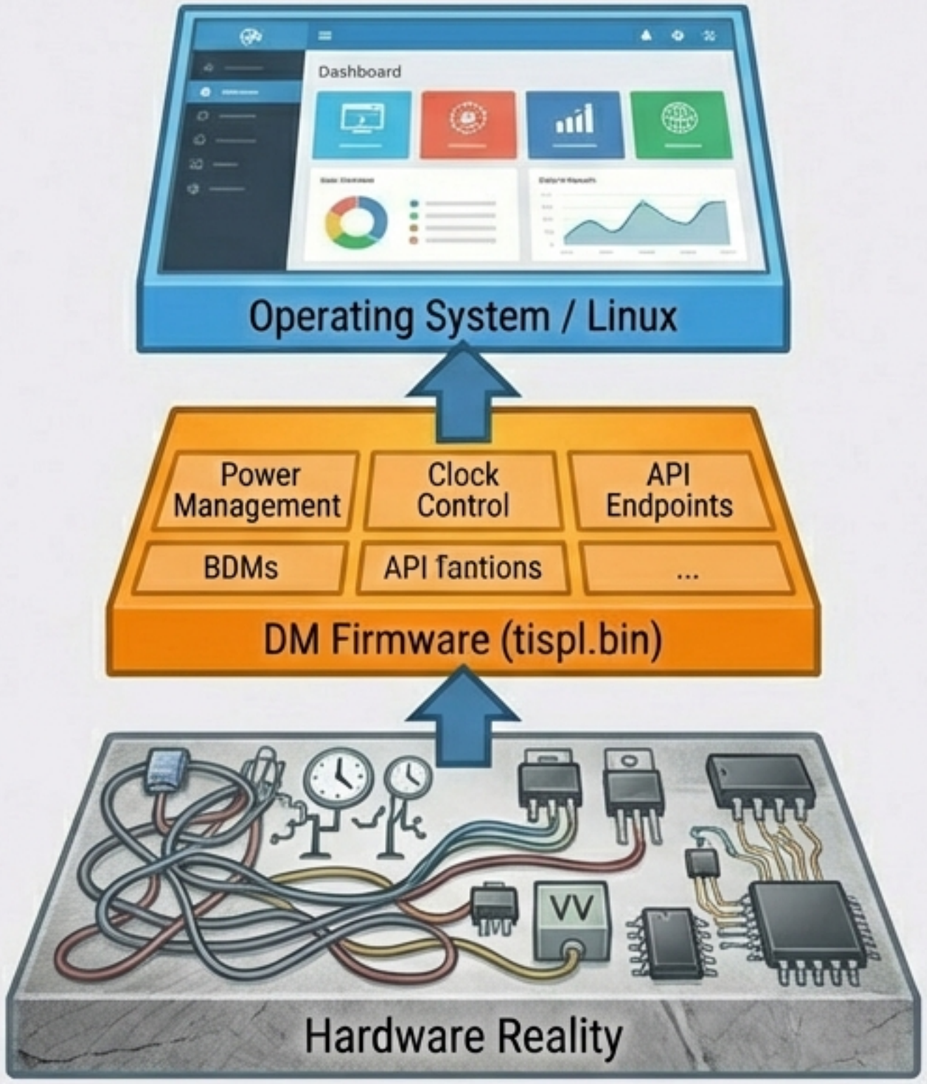


Micro-to-Macro:

- We are NOT on the main CPU yet. We are running on a Cortex-R5 microcontroller because of the SRAM size limit.
- **The Goal:** Initialize DDR.

  The R5 SPL (Secondary Program Loader) configures the memory controller to “unlock” the external RAM.
  This is the moment the system upgrades from a 256KB scratchpad to Gigabytes of workspace.

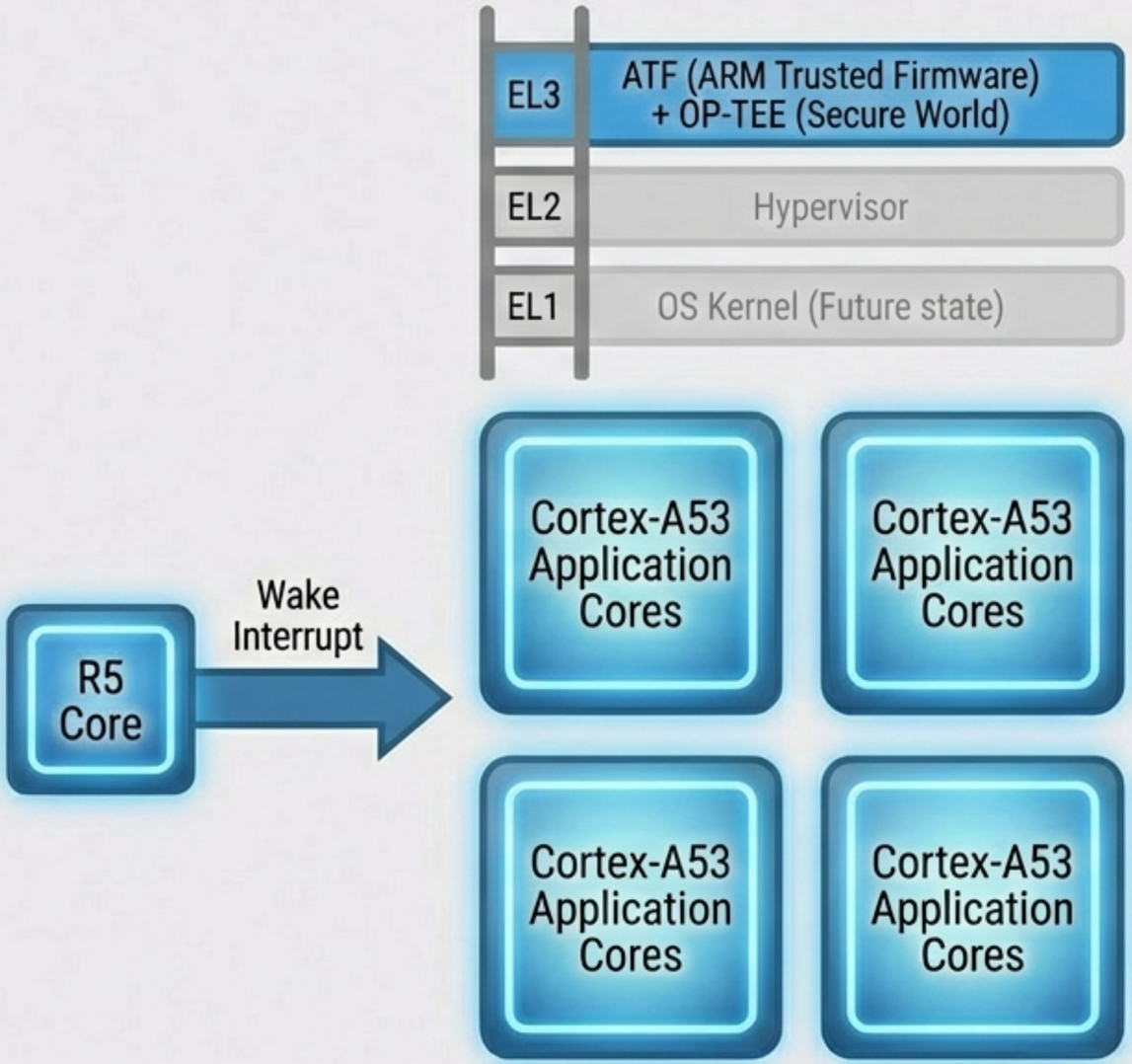
Step 4: System Configuration (TISPL)



Abstraction Layers:

- With DDR initialized, we load Device Management Firmware.
- This firmware abstracts the complex physics of power and clock trees.
- Linux doesn’t need to know *how* to turn on a clock; it just asks the Firmware to do it.

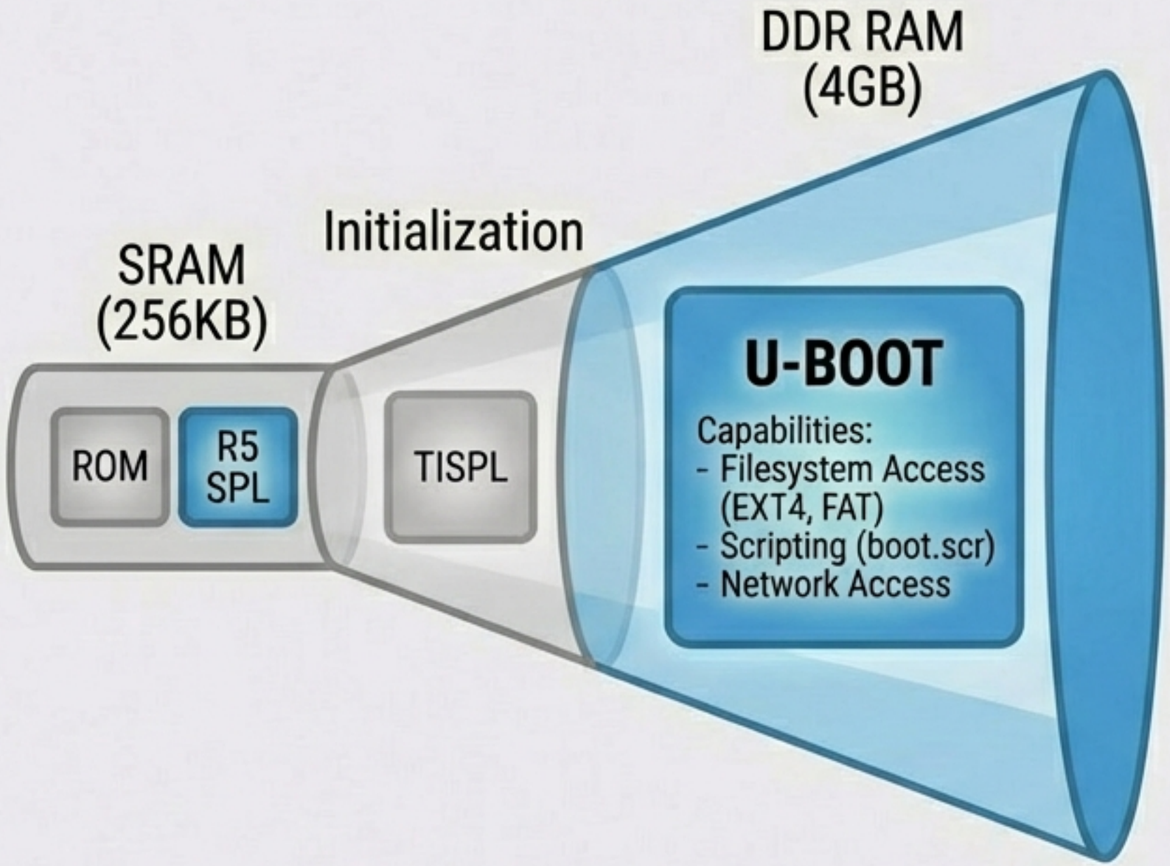
Step 5: Waking the Giants (A53 & EL3)



The Baton Pass:

- The microcontroller wakes the main CPUs. They start at Exception Level 3 (EL3)—the highest privilege.
- **Action**: Initialize Runtime Services (ATF) and the Trusted Execution Environment (OP-TEE/Software TPM) to run parallel to the OS.

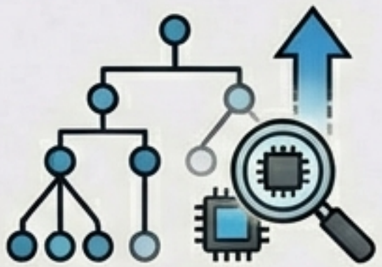
Step 6: U-Boot & The Final Handoff



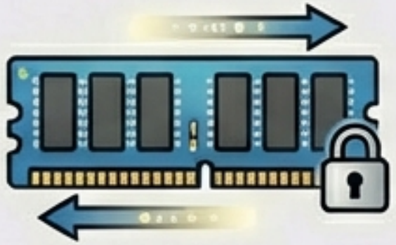
The Last Mile:

- U-Boot runs in the massive DDR space. It has the luxury of size. It reads “uEnv.txt”, loads the Kernel Image and Device Tree, and prepares the final arguments for Linux.

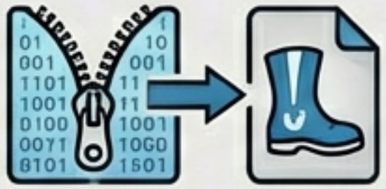
The Destination: Linux Requirements



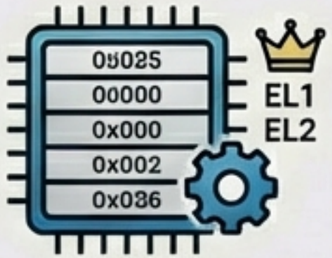
- [ ] **Hardware Map**
  Device Tree (.dtb). A file describing every UART, MMC, and Clock address.



- [ ] **Stable Memory**
  Initialized DDR. The playground must be open.

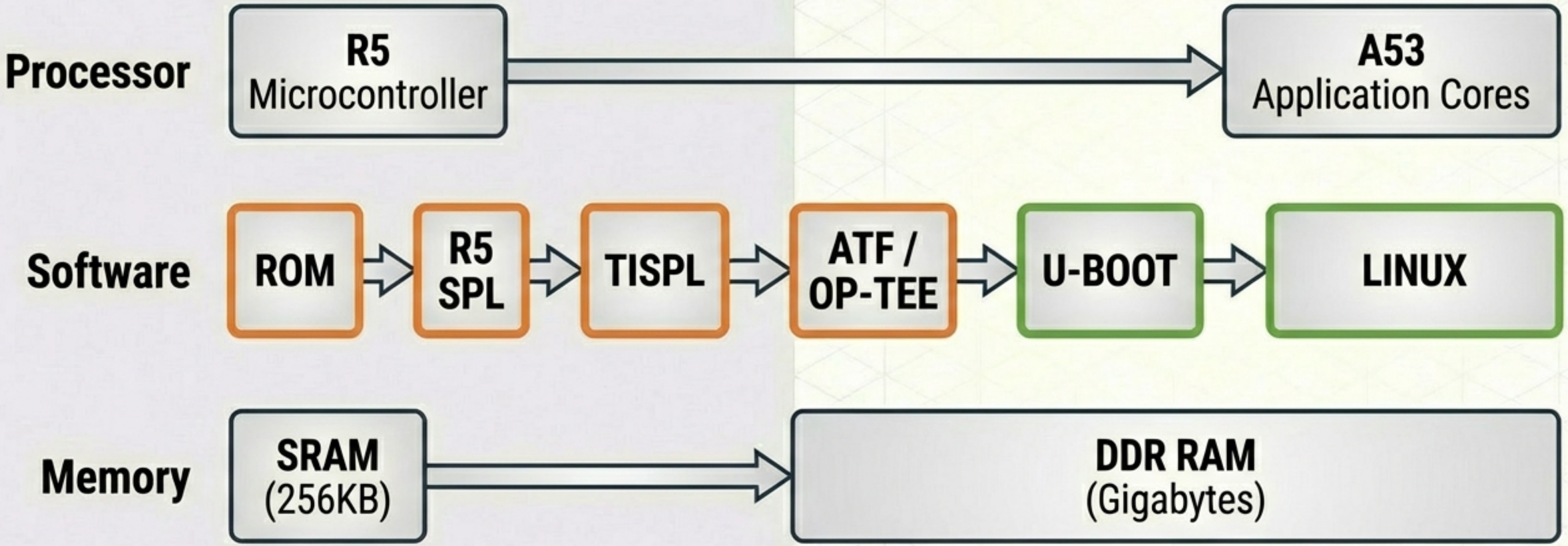


- [ ] **Uncompressed Binary**
  Image. The ARM64 kernel cannot decompress itself. U-Boot must unzip it into memory.



- [ ] **Register State**
  Clean handoff state. CPU must be in EL1 or EL2.

The “Relay Race” Summary



Embracing the Complexity

Flexibility = Chaos

The lack of standardization is a feature, not a bug. It allows for highly optimized, tailored silicon.

Tailored Software

Bootloaders are not generic. They are tightly coupled to the electrical reality of the specific board.

Trust is Paramount

From etched ROM to x509 certificates, modern booting is as much about security chains as it is about loading code.

Resources: TI K3 Technical Reference Manual | Linux Kernel Booting Documentation