

Part 1: Annotated and Corrected Version of Original Boot Flow (Arrow Lake + Slim Bootloader)

During Power-On and CSME Bring-Up: When the system receives power, the Platform Controller Hub (PCH) powers up first and internally activates the Converged Security and Management Engine (CSME), which is a dedicated microcontroller embedded in the PCH silicon. The CSME begins executing from a built-in ROM that contains immutable boot code. This ROM performs internal validation and then authenticates the signed CSME firmware stored in a dedicated region of the SPI flash (inside the IFWI image). Once authenticated, the CSME firmware runs and applies hardware fuses that were permanently set at manufacturing time—these fuses define the Boot Guard v3 configuration (e.g., whether boot verification or measurement is required, whether OEM signing is allowed). It then locates and validates the Key Manifest (KM) and Boot Policy Manifest (BPM), which are signed structures in the BIOS region of SPI flash. The KM contains a public key for validating the ACM, and the BPM identifies the specific ACM module and firmware stages to verify. Only after all policies are validated and Boot Guard mode is established does the CSME release the main CPU from reset. The CPU cannot proceed until this entire trust policy has been enforced.

During CPU Reset Vector Execution and ACM Launch: After CSME releases the CPU, it exits reset and starts executing from the reset vector at physical address 0xFFFFFFF0—this is part of the top of SPI flash. The reset vector points to a JMP instruction that transfers control to the Authenticated Code Module (ACM), whose location is defined in the FIT (Firmware Interface Table). The FIT is a structure in SPI flash near the top of the BIOS region and includes entries for ACMs, microcode, and firmware volumes. The CPU loads the ACM and performs a cryptographic signature verification using a hardcoded Intel public key hash fused into the CPU hardware at the factory. This is the beginning of the Boot Guard root of trust from the CPU's perspective. The ACM executes in a protected memory region carved out from the L3 cache (Authenticated Code RAM). While CSME does not participate in signature verification at this stage, its earlier validation of the Boot Policy Manifest (BPM) ensures that only a policy-approved ACM can be loaded. The ACM reads and interprets the BPM itself to determine whether to verify or measure the next firmware stage (Stage1A). If the hash verification of Stage1A succeeds, ACM concludes its role and transfers control to Stage1A.

Stage1A Initialization and Cache-as-RAM Setup: Stage1A, the first Slim Bootloader component to run after ACM validation, begins in 16-bit real mode and immediately transitions the CPU into 32-bit protected mode. This is done by loading a minimal Global Descriptor Table (GDT) defining flat memory segments and setting the PE bit in CR0. Early debugging capability is optionally initialized here, typically by configuring a UART controller as specified in board-level configuration headers. With the CPU in protected mode, Stage1A constructs a configuration structure known as the FSP-T UPD (User Product Data) with platform-specific details like cache size, stack base, and temporary RAM bounds. This structure is passed to the `FspTempRamInit()` API from Intel's FSP-T (Firmware Support Package for Temporary RAM). FSP-T then sets up a reserved portion of the L2/L3 CPU cache as temporary RAM (CAR), effectively simulating usable memory in the absence of DRAM. In return, FSP-T provides a Hand-Off Block (HOB) describing the size and base of this temporary RAM region. Stage1A then allocates a central Slim Bootloader structure called `LdrGlobal`, which holds global firmware state, memory information, debug settings, and pointers to critical regions. Since interrupts are disabled at this stage, the pointer to `LdrGlobal` is saved in the IDTR register—normally used for the interrupt descriptor table—so that it can be accessed consistently by other modules. Stage1A may verify the Stage1B firmware block (using a hash or signature) depending on the

Boot Guard policy, and packages all key values into a structure called `Stage1A_PARAM` before jumping to the Stage1B entry point.

Stage1B Execution and DDR Initialization (FSP-M): Stage1B is responsible for initializing DRAM and transitioning the system from temporary cache-backed memory to full main memory execution. It begins by parsing board-specific configuration data, typically generated from YAML files, which describe the platform's memory topology, number of DIMM slots, supported frequencies, voltage profiles, and SPD override data if applicable. This data is used to populate the UPD (User Product Data) structure for the Memory FSP (FSP-M). Stage1B performs minimal platform initialization such as early GPIO setup, clock muxing, and voltage rail preparation needed before DDR training can proceed. It then calls `FspMemoryInit()` from FSP-M, passing the populated UPD. FSP-M performs silicon-validated DDR training procedures (MRC - Memory Reference Code), detects which memory modules are actually populated (e.g., YAML may specify two SoDIMMs, but only one may be installed), and builds a HOB (Hand-Off Block) list representing the **actual usable memory** discovered and initialized. This includes detailed descriptors for each memory region, its type, and capabilities. Once DRAM is up, Stage1B migrates the `LdrGlobal` structure and runtime stack from temporary CAR into system RAM. It also updates the memory boundaries in `LdrGlobal`, including `TOLUM` (Top of Low Usable Memory), `TOUUM` (Top of Upper Usable Memory), and reserved memory regions for later boot stages. After memory migration, Stage1B calls the `TempRamExit()` API provided by FSP-M to cleanly disable CAR and return the CPU cache to its normal coherent state. Once temporary resources are shut down and DRAM is active, Stage1B jumps to the Stage2 entry point defined via `_ModuleEntryPoint`, passing in a `STAGE2_PARAM` structure that includes pointers to the HOB list and `LdrGlobal`.

Stage2 Execution, Silicon Init, and Payload Handoff: Stage2 executes entirely from DRAM and handles the remainder of board and silicon initialization before the payload (OS loader) takes over. It begins by unshadowing itself—relocating the firmware image from SPI flash to DRAM—to improve execution performance. Stage2 then restores MRC (Memory Reference Code) training parameters saved in HOBs during Stage1B. These are critical for S3 resume (suspend-to-RAM) scenarios, where a full retraining of DDR can be skipped.

Next, Stage2 prepares UPDs for FSP-S (Silicon Init), populating them with platform-specific settings for the PCH and CPU such as PCIe configuration, power gating policies, thermal throttling limits, and GPIO table definitions. It invokes `FspSiliconInit()` to perform the remaining low-level hardware initialization tasks. FSP-S configures subsystems such as USB controllers, UARTs, SPI interfaces, PCH power management controller (PMC), and interrupt routing.

Following FSP-S execution, Stage2 proceeds with platform bring-up tasks: - **Multi-processor initialization:** Brings additional logical CPU cores (APs) online. - **PCIe Enumeration:** Walks the PCI bus, assigns device numbers and base address registers (BARs), and maps memory/io resources for each device. - **ACPI Table Generation:** Uses internal platform libraries and board config to dynamically build tables like RSDP (Root System Descriptor Pointer), FADT (Fixed ACPI Description Table), DSDT (Differentiated System Description Table), MADT (Multiprocessor Table), and SSDT (Secondary Tables).

ACPI tables depend on earlier memory HOBs (especially for memory map and MMIO layout), and are published for the OS or payload to consume. Stage2 finalizes the firmware state by notifying FSP of `ReadyToBoot` and `EndOfFirmware` phases, allowing final cleanups (e.g., TPM finalization, clearing sensitive cache).

Stage2 then locates the configured payload (e.g., OsLoader, UEFI binary, or Linux kernel), validates it, and loads it into memory. It provides the payload with a pointer to the HOB list via `STAGE2_PARAM`, which includes everything needed for runtime boot: hardware map, memory topology, ACPI tables, and firmware resource descriptors. At this point, control fully transitions to the payload.

Payload Execution and Kernel Handoff (Linux and Windows Examples): Once Stage2 transfers control to the payload, the final system boot stage begins. The payload—typically OsLoader in Slim Bootloader—uses the HOB list and optionally `LdrGlobal` to access board-specific hardware data and ACPI tables. If the payload is configured for Linux: - It loads the Linux `bzImage` (compressed kernel image) into DRAM at a specified entry point. - It prepares a `boot_params` structure with references to the initial RAM disk (initrd), command-line arguments, and optionally a Device Tree Blob (DTB) if DT-based boot is used. - The kernel receives either ACPI tables (via RSDP) or DTB depending on build config and command-line override (e.g., `acpi=off`).

If booting Windows: - The OsLoader typically launches a Windows Boot Manager image (e.g., `bootmgfw.efi`) via UEFI payload or directly. - The Windows kernel expects a full ACPI table set to enumerate hardware and load drivers. - The root of trust (Boot Guard measurements, TPM logs) may be consumed by Windows Defender or BitLocker for attestation.

Finally, once the OS kernel is loaded and its memory and hardware tables are mapped (via ACPI or DT), it takes over full system control, begins device enumeration, mounts the root file system, and starts user-space services or logins.
