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). The CSME begins executing from a built-in ROM that is integrally part of the silicon die. This hardwired ROM does not need to be loaded; it uses a hardware-default SPI controller state and Direct Memory Access (DMA) to pull the signed CSME firmware from flash into SRAM. The CSME then acts as the platform's early bring-up controller, initializing critical PCH components including clocks, GPIO, and applying mandatory firmware patches to the Power Management Controller (PMC). Without a valid PMC patch, platform power sequencing and boot cannot proceed. Once early platform readiness is established, CSME reads Field Programmable Fuses (FPF) to determine Boot Guard configuration policies.

Importantly, CSME itself does not perform the authoritative cryptographic verification of the Key Manifest (KM) and Boot Policy Manifest (BPM). Its role is to establish platform security state, enforce initial policy gating, and release the main CPU from reset once conditions are met.

During CPU Reset Vector Execution and ACM Launch: After CSME releases the CPU from reset, the processor begins execution at the architectural reset vector located at physical address 0xFFFFFFF0. CPU microcode consults the Firmware Interface Table (FIT) located in flash to identify microcode updates and the location of the Authenticated Code Module (ACM). The CPU performs hardware-based cryptographic signature verification of the ACM using an Intel public key hash permanently fused into the processor silicon.

Upon successful verification, the ACM is loaded into Authenticated Code RAM (ACRAM), an isolated execution environment typically implemented within protected cache structures that prevent external DMA access or snooping. The ACM does not trust the pre-boot environment and independently reads the Field Programmable Fuses (FPF) from the PCH to retrieve the OEM Root Key Hash.

Using this fuse-rooted trust anchor, the ACM performs a complete chain of authoritative validation: 1. Verifies the Key Manifest (KM) against the fused OEM Root Key Hash 2. Verifies the KM signature integrity 3. Verifies the Boot Policy Manifest (BPM) using the validated KM 4. Verifies the Initial Boot Block (IBB), corresponding to Stage1A firmware hash defined in the BPM

Only if all validation steps succeed does the ACM transfer execution to Stage1A of Slim Bootloader.

Stage1A Initialization and Cache-as-RAM Setup: Stage1A is the first Slim Bootloader software component to execute after ACM validation. It begins in

16-bit real mode and quickly transitions the CPU into 32-bit protected mode by loading a minimal Global Descriptor Table (GDT) defining flat memory segments and setting the Protection Enable (PE) bit in the CR0 control register.

Early debug facilities may be initialized here, commonly by configuring a UART interface according to board configuration headers. Stage1A constructs the FSP-T (Firmware Support Package - Temporary RAM) User Product Data (UPD) structure with platform-specific parameters including cache size, temporary stack region, and temporary RAM boundaries.

Stage1A invokes the FspTempRamInit() API from Intel FSP-T, which configures a portion of CPU cache (L2/L3) as Cache-as-RAM (CAR). This provides a temporary working memory environment before DRAM initialization. FSP-T returns a Hand-Off Block (HOB) describing the temporary RAM region.

Stage1A then allocates and initializes the LdrGlobal structure, which serves as Slim Bootloader's internal global data repository containing firmware state, memory boundaries, debug configuration, and internal pointers. Because interrupts are disabled at this stage, the pointer to LdrGlobal is stored in the IDTR register for consistent access across early modules.

Depending on Boot Guard policy, Stage1A may perform verification of Stage1B firmware before packaging required parameters into the Stage1A_PARAM structure and transferring control to Stage1B.

Stage1B Execution and DDR Initialization (FSP-M): Stage1B transitions the system from temporary cache-backed memory to full DRAM operation. It parses board-specific configuration data typically generated from YAML descriptions that define memory topology, DIMM population, supported speeds, voltage profiles, and SPD override data.

This configuration populates the UPD structure for FSP-M (Firmware Support Package - Memory). Stage1B performs prerequisite platform initialization such as early GPIO setup, clock configuration, and power rail sequencing necessary for memory training.

Stage1B calls FspMemoryInit(), which executes Intel's Memory Reference Code (MRC) to train DRAM timing, detect installed memory modules, and initialize the memory controller. FSP-M builds a comprehensive HOB list describing actual system memory layout, including usable RAM regions, reserved memory, MMIO ranges, and capability flags.

Once DRAM is operational, Stage1B migrates the LdrGlobal structure and execution stack from CAR into main memory. It updates memory boundary information such as TOLUM (Top of Low Usable Memory), TOUUM (Top of Upper Usable Memory), and reserved firmware regions.

Stage1B then invokes TempRamExit() to disable Cache-as-RAM and restore CPU cache to normal coherent operation before transferring control to Stage2 via the Stage2 entry point, passing the HOB list pointer and LdrGlobal within STAGE2_PARAM.

Stage2 Execution, Silicon Initialization, and Payload Handoff: Stage2 runs fully from DRAM and performs remaining platform initialization. It unshadows the firmware image by relocating code from SPI flash to DRAM for faster execution. It restores memory training data stored in HOBs to support S3 resume flows, allowing DRAM initialization to be skipped during suspend-to-RAM resume.

Stage2 prepares UPDs for FSP-S (Silicon Init) with platform-specific configuration for CPU and PCH subsystems including PCIe topology, power management policies, thermal limits, interrupt routing, and GPIO configuration. It invokes FspSiliconInit() to initialize USB controllers, SPI buses, UARTs, power management controllers, and other chipset components.

After FSP-S completion, Stage2 performs additional platform tasks such as bringing up secondary processor cores, enumerating PCIe devices, assigning BARs, and mapping system resources.

Stage2 dynamically constructs ACPI tables including RSDP, XSDT/RSDT, FADT, MADT, DSDT, and SSDTs using memory HOB information and platform configuration data. These tables describe hardware topology, interrupt routing, power management, and device configuration expected by modern operating systems.

Once platform initialization is finalized, Stage2 signals ReadyToBoot and EndOfFirmware phases to FSP for cleanup operations such as TPM finalization and cache sanitation.

Stage2 then validates, loads, and transfers control to the configured payload (such as OsLoader or a UEFI-style binary), passing the HOB list pointer that contains memory map, ACPI table references, SMBIOS information, and firmware resource descriptors.

Payload Execution and Kernel Handoff (Linux and Windows Examples): After Stage2 hands control to the payload, the OS loader becomes responsible for loading the operating system kernel.

For Linux payloads, the loader places the compressed bzImage kernel into DRAM at the configured entry address and constructs the boot_params structure containing pointers to the initial RAM disk (initrd), kernel command line arguments, and firmware-provided platform data. Depending on configuration, the kernel receives either ACPI information through the RSDP pointer or a Device Tree Blob (DTB) if device-tree-based boot is selected or ACPI is disabled.

For Windows payloads, OsLoader typically launches a Windows Boot Manager image which relies on a complete ACPI table set to enumerate hardware devices, configure interrupt routing, and load appropriate drivers. Platform trust measurements stored in TPM and Boot Guard logs may be consumed by Windows security services for secure boot attestation.

Once the kernel has mapped memory, parsed ACPI or DT structures, and initialized core subsystems, it assumes full control of the system and transitions into user-space initialization.

NOTE – ACPI Table Handoff in Slim Bootloader: In Slim Bootloader, the operating system does not search memory to locate ACPI tables. Instead, SBL explicitly passes the ACPI Root System Description Pointer (RSDP) to the payload through a GUID-typed Hand-Off Block (HOB) as defined by the Universal Payload Interface.

During Stage2, SBL builds a HOB list that includes system memory descriptors, SMBIOS data, optional device tree information, and an ACPI Table HOB. This ACPI Table HOB contains an EFI_PHYSICAL_ADDRESS field pointing directly to the RSDP structure in memory. The payload parses the HOB list at entry, retrieves the RSDP address, and uses it to locate the XSDT/RSDT and all subsequent ACPI tables for the operating system.

This mechanism ensures a clean, secure, and standardized handoff of firmware-generated platform tables without exposing internal Slim Bootloader data structures such as LdrGlobal.

References: -

Official Slim Bootloader

Boot Flow - Memory Map

Boot Guard/Root of trust

Converged security and management engine