WHAT HAPPENS BEFORE APP_MAIN

- 1st stage bootloader (ROM) loads 2nd stage bootloader to RAM (IRAM & DRAM) from flash offset 0x1000 (4096 bytes offset).
- 2nd stage bootloader loads partition table and main app image from flash. main app incorporates both RAM & read-only segments mapped via flash cache.
- main app image executes. at this point the 2nd CPU and RTOS scheduler can be started.

PRO CPU + APP CPU

reset vector code (ISR?)
0x4000 0400 mask ROM
} cannot be modified

1ST STAGE

① RESET FROM DEEP SLEEP

RTC_CNTL_STORE6_REG - ret address
RTC_CNTL_STORE7_REG - CRC
} valid? jump

② POWER ON RESET / SOC / WATCHDOG SOC

check GPIO_STRAP_REG if UART / SDIO download mode requested.

③ SOFTWARE CPU RESET / WATCHDOG CPU

based on EFUSE, load code from flash or start BASIC (UART) interpreter.

1st 4KB sector = secure boot IV + signature of app image
0x1000 = app binary image.

0X1000 = 2nd stage bootloader (components / bootloader) ESP-IDF

↓

adds flexibility to flash layout (partition tables).
support flash encryption
secure boot
over the air (OTA) updates.

0X8000 = partition table

factory, OTA partitions
OTA info partion → which one to boot

copies data & code sections to IRAM, DRAM.
for load addresses in IROM and DROM regions,
flash MMU is configured

bootloader can be changed by copying it to build directory. it will be used insted of one in ESP-IDF.

APPLICATION STARTUP

PRO CPU

entry point → call_start_cpu0() :components/esp32/cpu_start.c

-enable heap allocator

-make APP CPU jump to its entry point
↓
call_start_cpu1()
↓
__weak start_cpu1()

jumps to
↓

__weak start_cpu0()

scheduler is started in both CPUs.

app_main () → main task

stack size, priority : menuconfig

IRAM

instruction RAM

ESP-IDF allocates part of internal SRAM0 for IRAM.

1st 64 KB used for PRO and APP CPU caches

0x40080000 → 0x400A0000

used to store parts of application which need to run from RAM.

few components of ESP-IDF, parts WiFi stack

```
void IRAM_ATTR gpio_isr_handler (void *arg) {
    ...     // placed into IRAM
}
```

interrupt handlers must be placed into IRAM if ESP_INTR_FLAG_IRAM is used when registering interrupt handler. such ISR may only all functions in IRAM or ROM. all constant data must be placed into DRAM with DRAM_ATTR.

timing critical code may be placed into IRAM.

ESP32 reads code and data from 32 KB cache.

NOTE: all FreeRTOS APIs are currently placed into IRAM.

IROM (flash)

0x400D0000 - 0x40400000 region
transparently cached by flash MMU using 2 32KB blocks
0x40070000 - 0x40080000

DRAM

non-constant static data and zero initialized data
0x3FFB0000 - 0x3FFF0000 (256 KB)

constant data may also be placed in DRAM if ISR is using it.
DRAM_ATTR const char [] format_string = "%p %x";

use ESP_EARLY_LOGx macro when logging from ISR

__NOINIT_ATTR :
value will not be initialized. maintained after s/w reset too.

DROM (flash)

0x3F400000 - 0x3F800000 (4MB)
used to access external flash memory via flash MMU.

RTC slow memory

variables used in deep-sleep code stored here.

RTC_NOINIT_ATTR → variables Keep their value after waking too.

RTC_NOINIT_ATTR uint32_t rtc_noinit_data;

DMA CAPABLE REQUIREMENT

DMAS require data in DRAM, word-aligned (use DMA_ATTR).

DMA_ATTR uint8t buffer[] = "I want to send something";

DMA_ATTR static uint8_t buffer[] = "I want to send something";

use macro WORD_ALIGNED_ATTR in function variables.

uint8_t stuff;

WORD_ALIGNED_ATTR uint8_t buffer[] = "...";