All you ever wanted to know about the AMD Platform Security Processor and were afraid to emulate

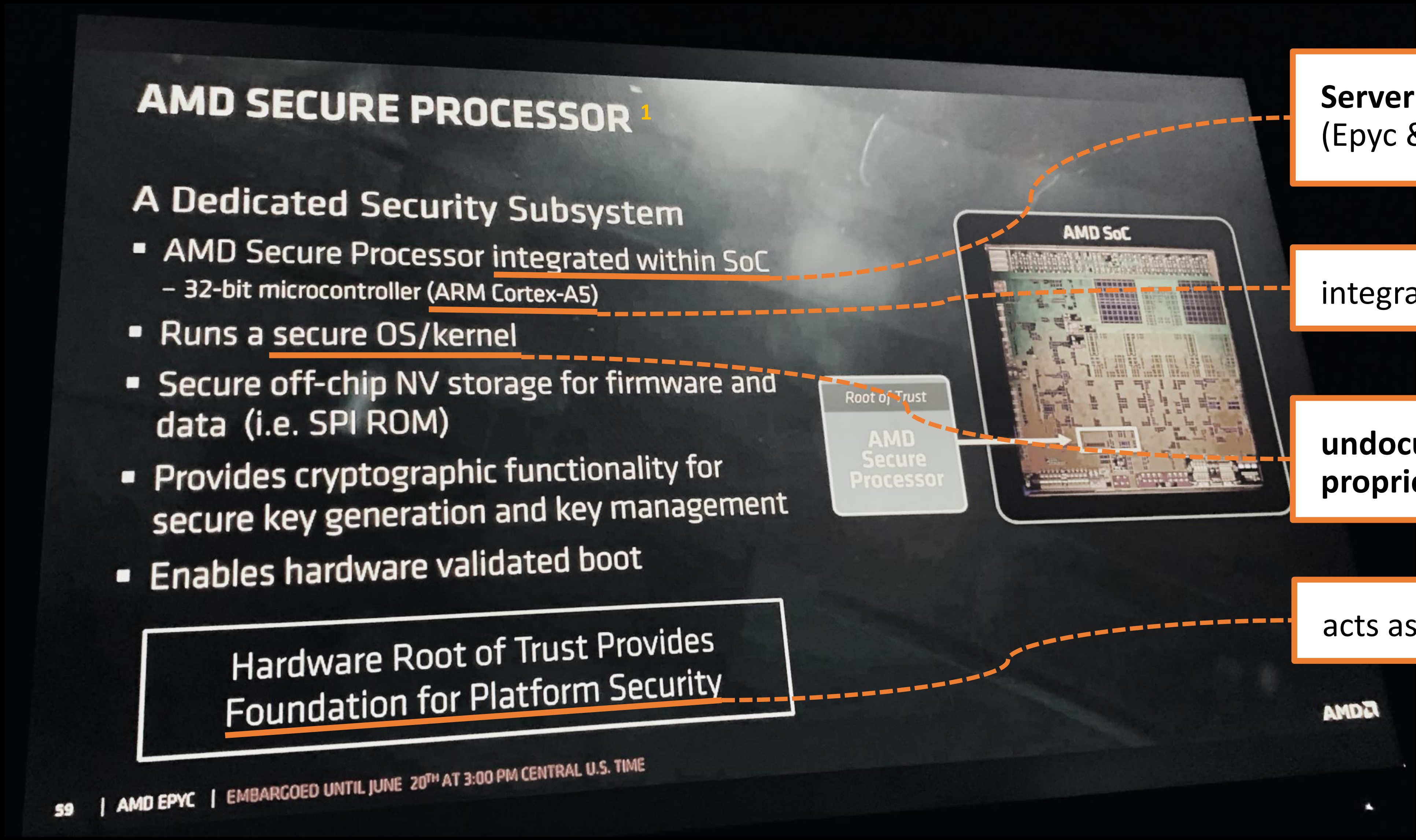
INSIDE A DEEPLY EMBEDDED SECURITY PROCESSOR.

Alexander Eichner
Technische Universität Berlin

Robert Buhren
Technische Universität Berlin

Outline

- What is the Platform Security Processor (PSP)?
  - Why emulate it?
- How to emulate the PSP
- What can we do with the emulator?



AMD SECURE PROCESSOR [1]

A Dedicated Security Subsystem

- AMD Secure Processor integrated within SoC
  - 32-bit microcontroller (ARM Cortex-A5)
- Runs a secure OS/kernel
- Secure off-chip NV storage for firmware and data (i.e. SPI ROM)
- Provides cryptographic functionality for secure key generation and key management
- Enables hardware validated boot

Hardware Root of Trust Provides Foundation for Platform Security

59 | AMD EPYC | EMBARGOED UNTIL JUNE 20TH AT 3:00 PM CENTRAL U.S. TIME

Server & Desktops
(Epyc & Ryzen)

integrated **since 2013**

undocumented, proprietary firmware

acts as **trust anchor**

[1] Formerly known as *Platform Security Processor (i.e.* ***PSP****)*

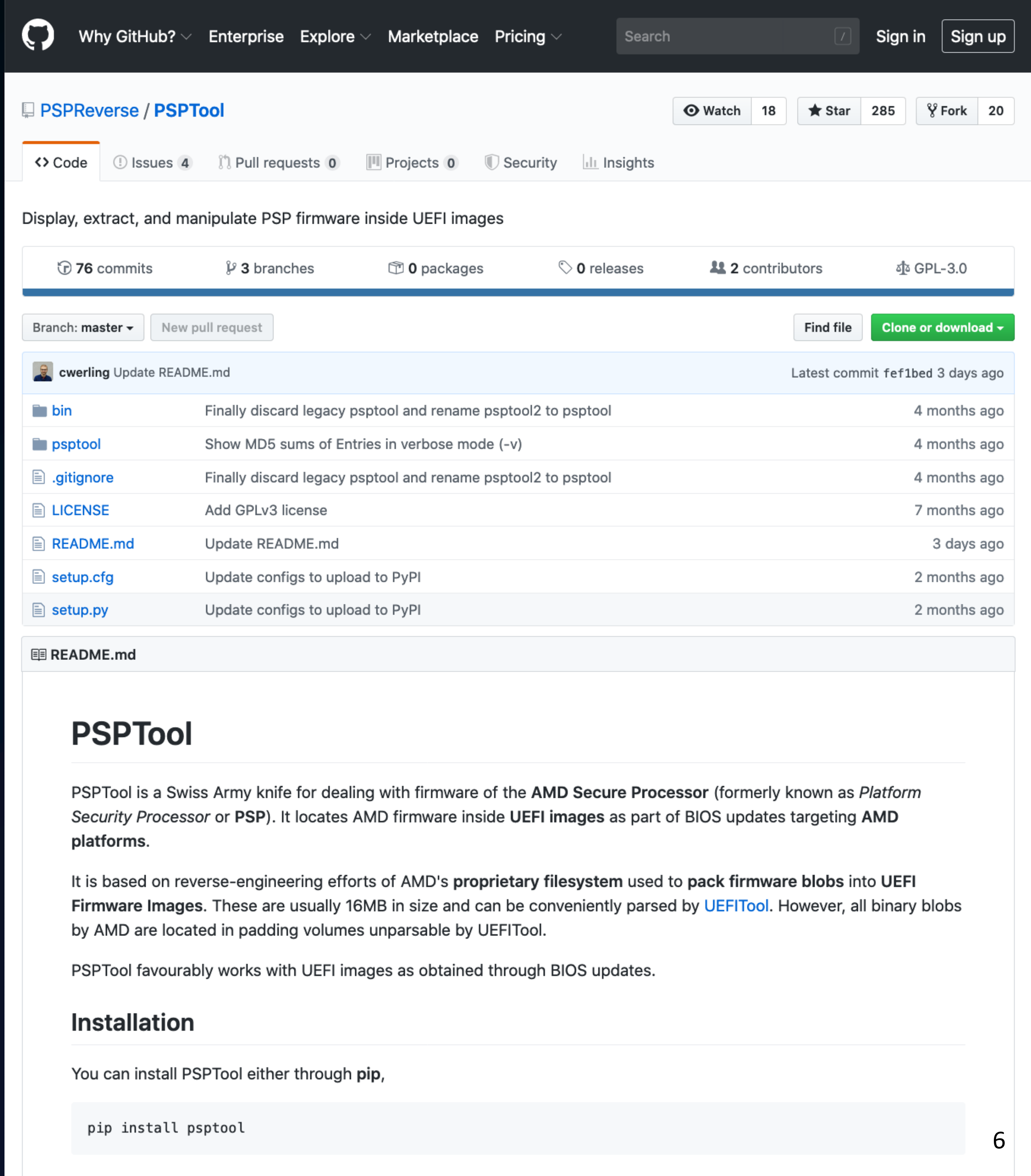
Why Emulate?

- Proprietary software at the highest privilege level
- Static analysis is possible but time consuming (boring ☺)
    - Only good for a single firmware version
- Emulation (if done right) enable easy analysis of future firmware versions

PSPTOOL

- Python-based
- Command-line interface
- Parsing
- Extraction
- Manipulation
- Decompression
- Signature verification
- PEM export of keys
- Duplicate detection
- Signature update
- Python API
- GPLv3

https://github.com/PSPReverse/PSPTool



PSPReverse / PSPTool

Display, extract, and manipulate PSP firmware inside UEFI images

76 commits · 3 branches · 0 packages · 0 releases · 2 contributors · GPL-3.0

cwerling Update README.md — Latest commit fef1bed 3 days ago

bin	Finally discard legacy psptool and rename psptool2 to psptool	4 months ago
psptool	Show MD5 sums of Entries in verbose mode (-v)	4 months ago
.gitignore	Finally discard legacy psptool and rename psptool2 to psptool	4 months ago
LICENSE	Add GPLv3 license	7 months ago
README.md	Update README.md	3 days ago
setup.cfg	Update configs to upload to PyPI	2 months ago
setup.py	Update configs to upload to PyPI	2 months ago

README.md

PSPTool

PSPTool is a Swiss Army knife for dealing with firmware of the **AMD Secure Processor** (formerly known as *Platform Security Processor* or **PSP**). It locates AMD firmware inside **UEFI images** as part of BIOS updates targeting **AMD platforms.**

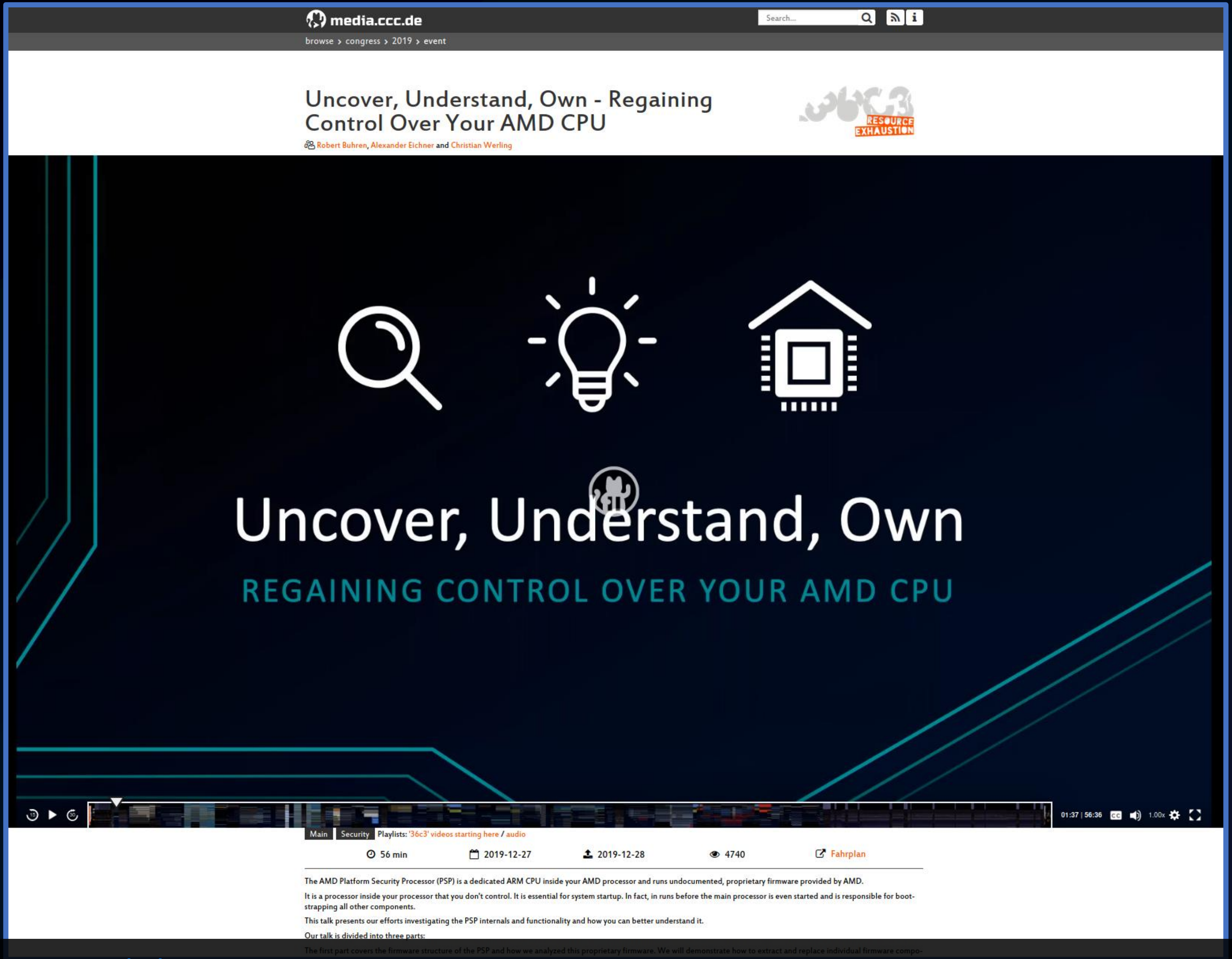
It is based on reverse-engineering efforts of AMD's **proprietary filesystem** used to **pack firmware blobs** into **UEFI Firmware Images**. These are usually 16MB in size and can be conveniently parsed by UEFITool. However, all binary blobs by AMD are located in padding volumes unparsable by UEFITool.

PSPTool favourably works with UEFI images as obtained through BIOS updates.

Installation

You can install PSPTool either through **pip**,

```
pip install psptool
```



media.ccc.de

Search...

browse › congress › 2019 › event

Uncover, Understand, Own - Regaining Control Over Your AMD CPU

Robert Buhren, Alexander Eichner and Christian Werling

Main | Security | Playlists: '36c3' videos starting here / audio

56 min | 2019-12-27 | 2019-12-28 | 4740 | Fahrplan

The AMD Platform Security Processor (PSP) is a dedicated ARM CPU inside your AMD processor and runs undocumented, proprietary firmware provided by AMD.

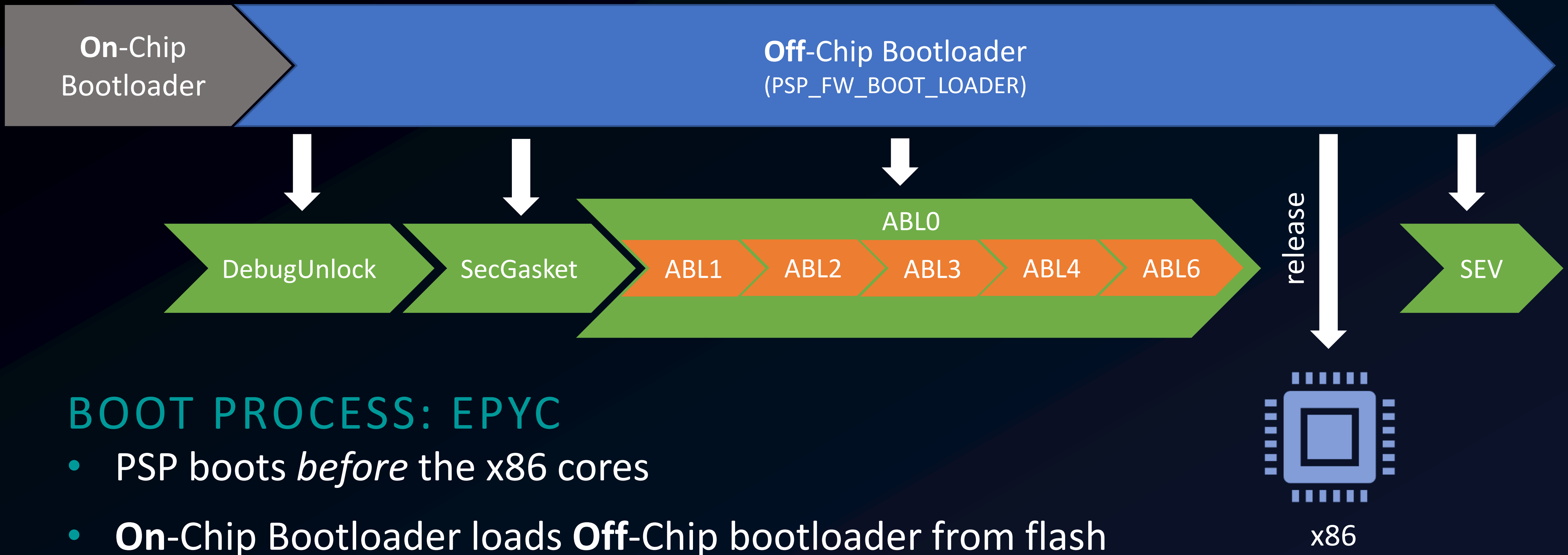
It is a processor inside your processor that you don't control. It is essential for system startup. In fact, in runs before the main processor is even started and is responsible for bootstrapping all other components.

This talk presents our efforts investigating the PSP internals and functionality and how you can better understand it.

Our talk is divided into three parts:

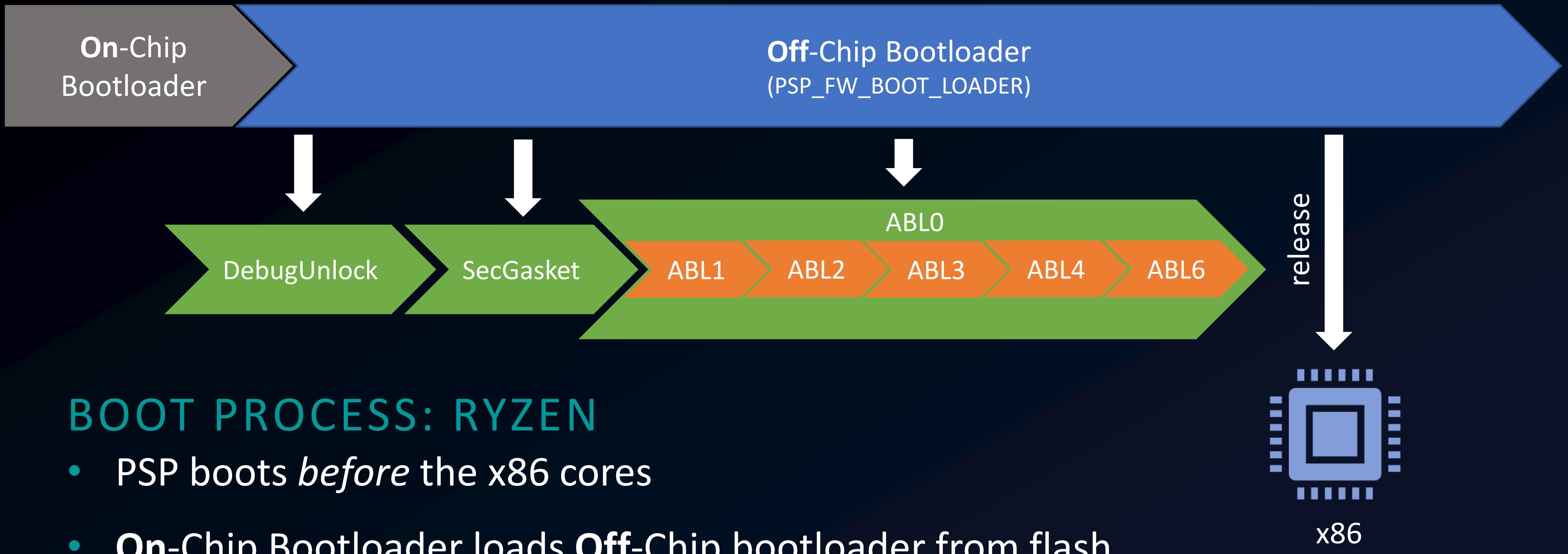
The first part covers the firmware structure of the PSP and how we analyzed this proprietary firmware. We will demonstrate how to extract and replace individual firmware compo-

https://media.ccc.de/v/36c3-10942-uncover_understand_own_-_regaining_control_over_your_amd_cpu



BOOT PROCESS: EPYC

- PSP boots *before* the x86 cores
- **On**-Chip Bootloader loads **Off**-Chip bootloader from flash
- **Off**-Chip Bootloader loads and executes apps in specific order
- System is initialized by different **ABL stages**
- Load UEFI image and release x86 cores from reset
- **SEV app** is loaded during runtime upon the **request of the OS**



BOOT PROCESS: RYZEN

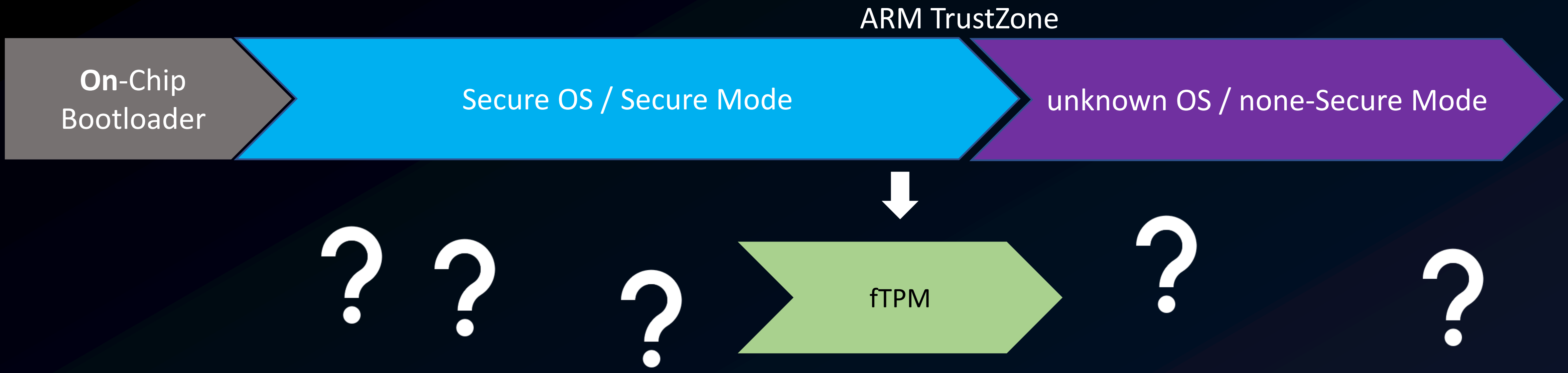
- PSP boots *before* the x86 cores
- **On**-Chip Bootloader loads **Off**-Chip bootloader from flash
- **Off**-Chip Bootloader loads and executes apps in specific order
- System is initialized by different **ABL stages**
- Load UEFI image and release **x86** cores from reset

On-Chip Bootloader

Secure OS

BOOT PROCESS: RYZEN

- PSP boots *before* the x86 cores
- **On**-Chip Bootloader loads **Off**-Chip bootloader from flash
- **Off**-Chip Bootloader loads and executes apps in specific order
- System is initialized by different **ABL stages**
- Load UEFI image and release **x86** cores from reset
- **SRAM is overwritten with Secure OS (Kinibi TEE)**



BOOT PROCESS: RYZEN

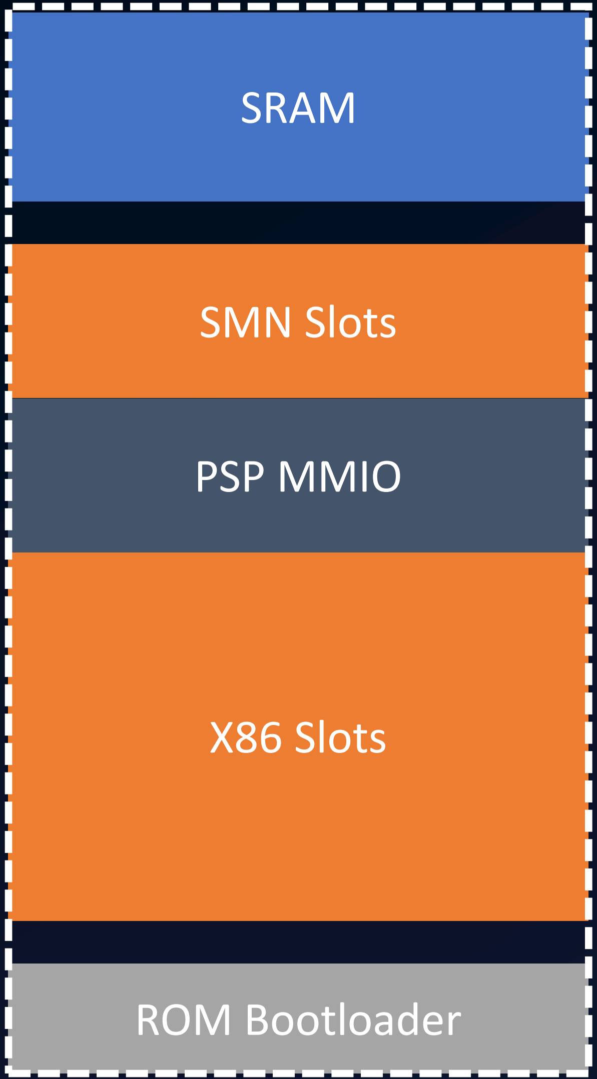
- PSP boots *before* the x86 cores
- **On**-Chip Bootloader loads **Off**-Chip bootloader from flash
- **Off**-Chip Bootloader loads and executes apps in specific order
- System is initialized by different **ABL stages**
- Load UEFI image and release **x86** cores from reset
- **SRAM is overwritten with Secure OS (Kinibi TEE)**
  - Firmware TPM is *one* application of this OS

PSP Hardware

PSP MEMORY LAYOUT

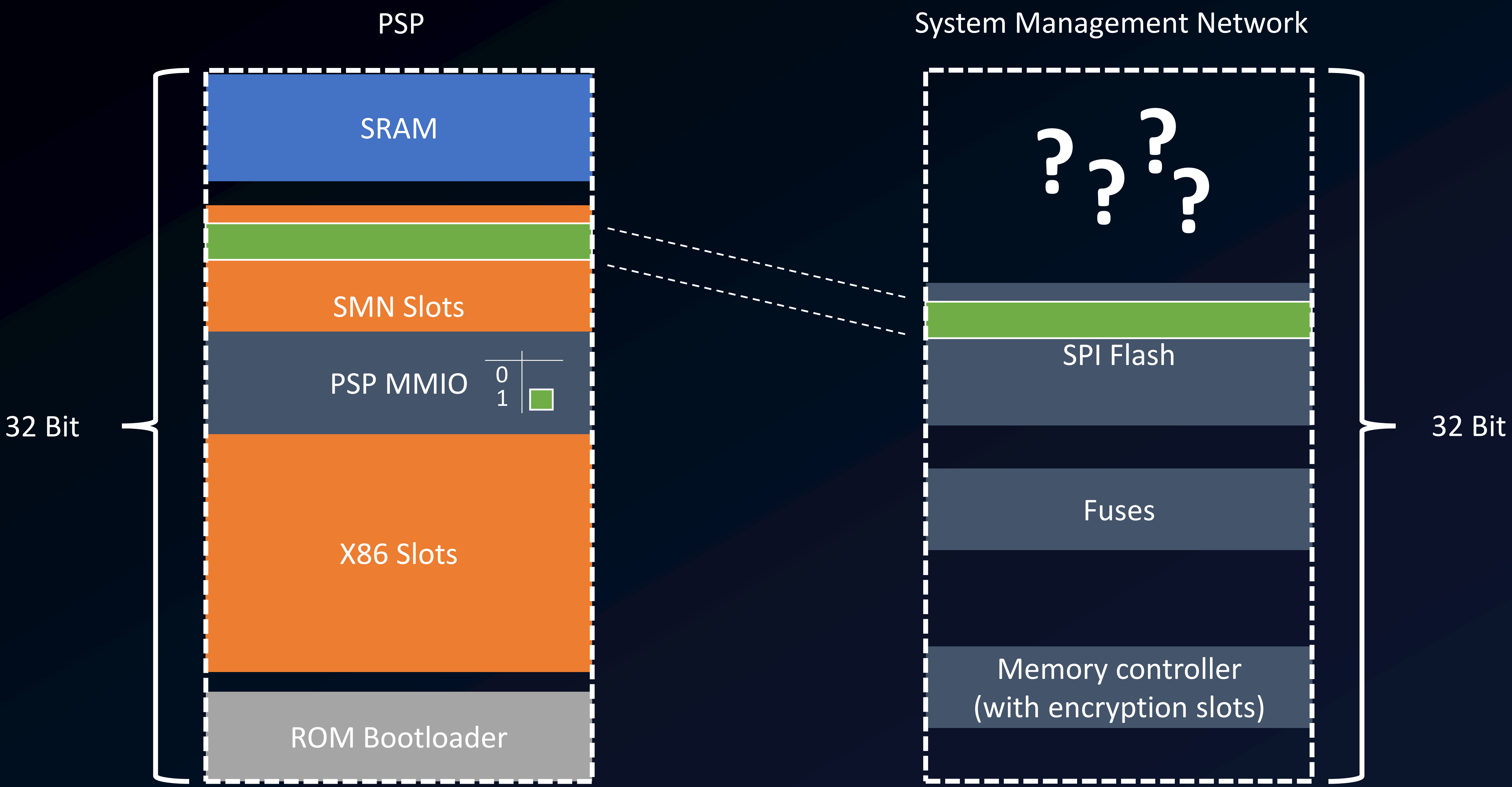
- 256 KB (Zen1) or 384 KB (Zen2) SRAM
  - Off-Chip BL and Applications
- On-Chip BL (ROM) at ARM high vectors (0xFFFF0000)
- MMIO: IRQ controller (custom), timer, crypto accelerator (CCP), X86 and SMN slot controller
- System Management Network Slots
- X86 address space slots

A slot is a “view” into another address space

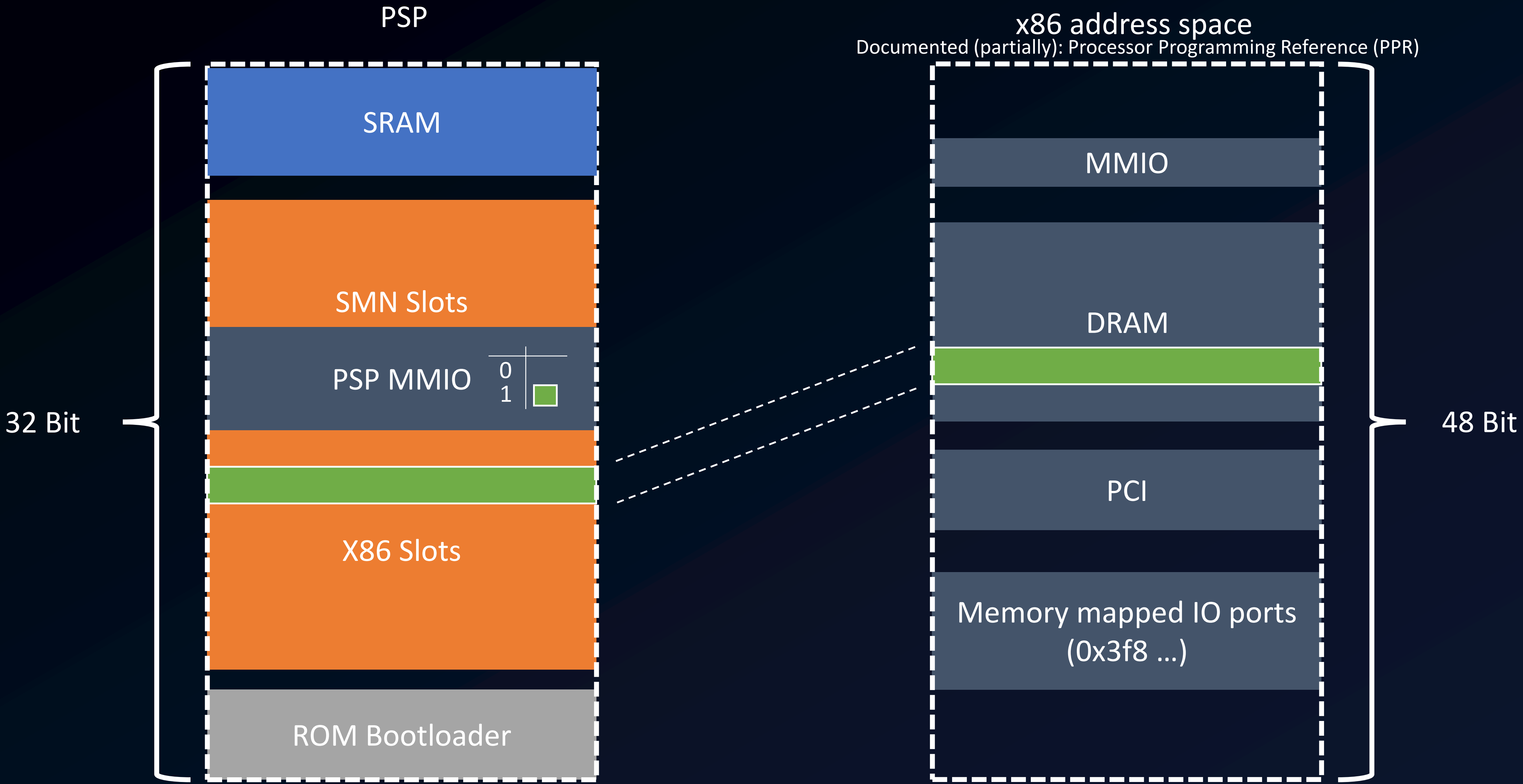


32 Bit / 4 GB

PSP ADDRESS SPACES



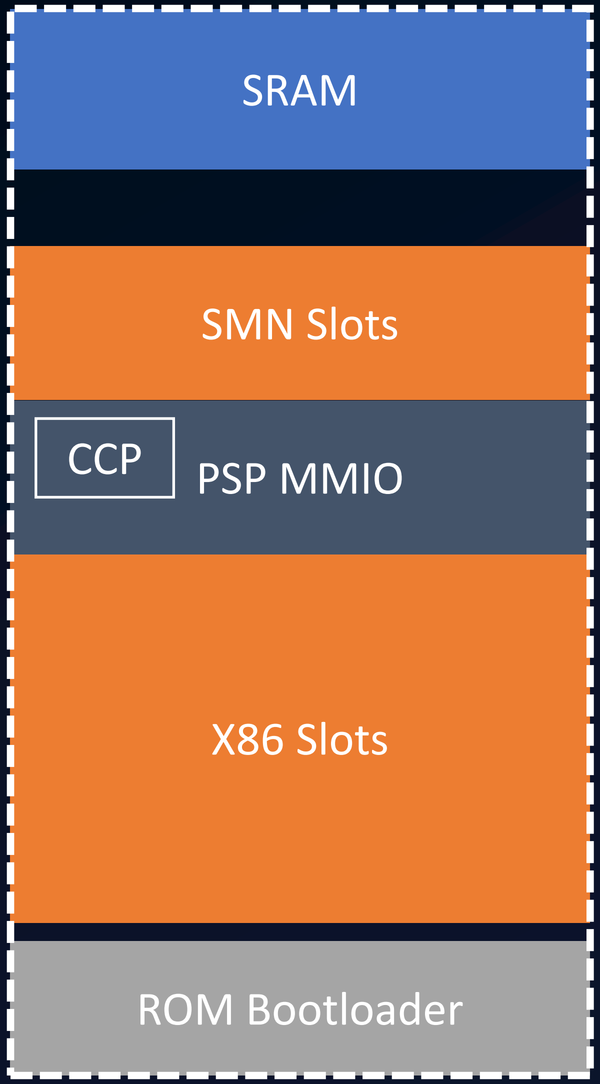
PSP ADDRESS SPACES



PSP CRYPTO ACCELERATOR (CCP)

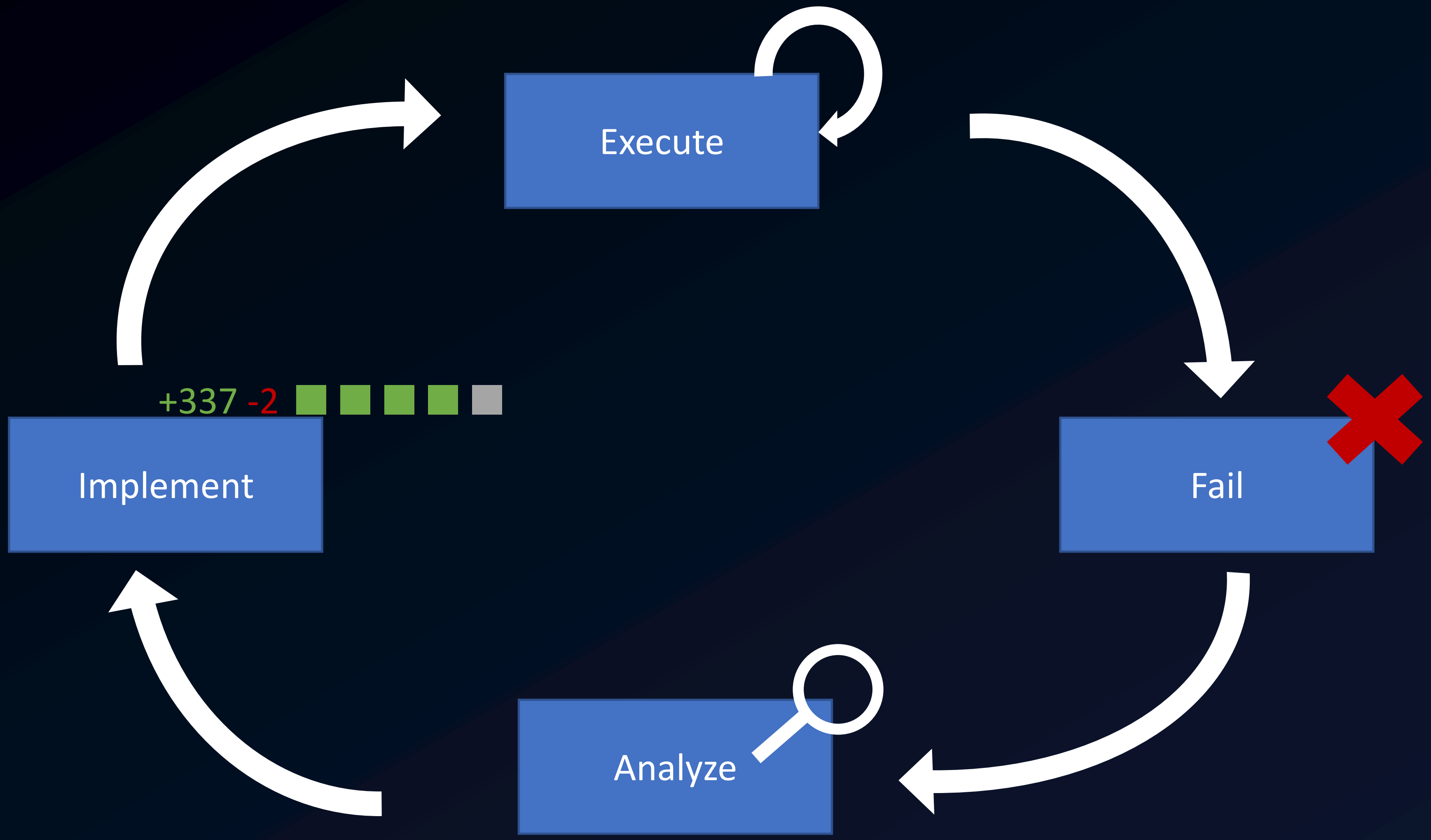
- PSP contains a Cryptographic Coprocessor V5 (CCP)
- Support for: SHA, RSA, AES, ECC, ZLIB, TRNG
- Used to verify signatures, decompress firmware files and as a DMA copy engine
- No “official” documentation available, but….

There is a Linux kernel driver: drivers/crypto/ccp



32 Bit / 4 GB

WHILE (!SUCCEEDED) {



Execute

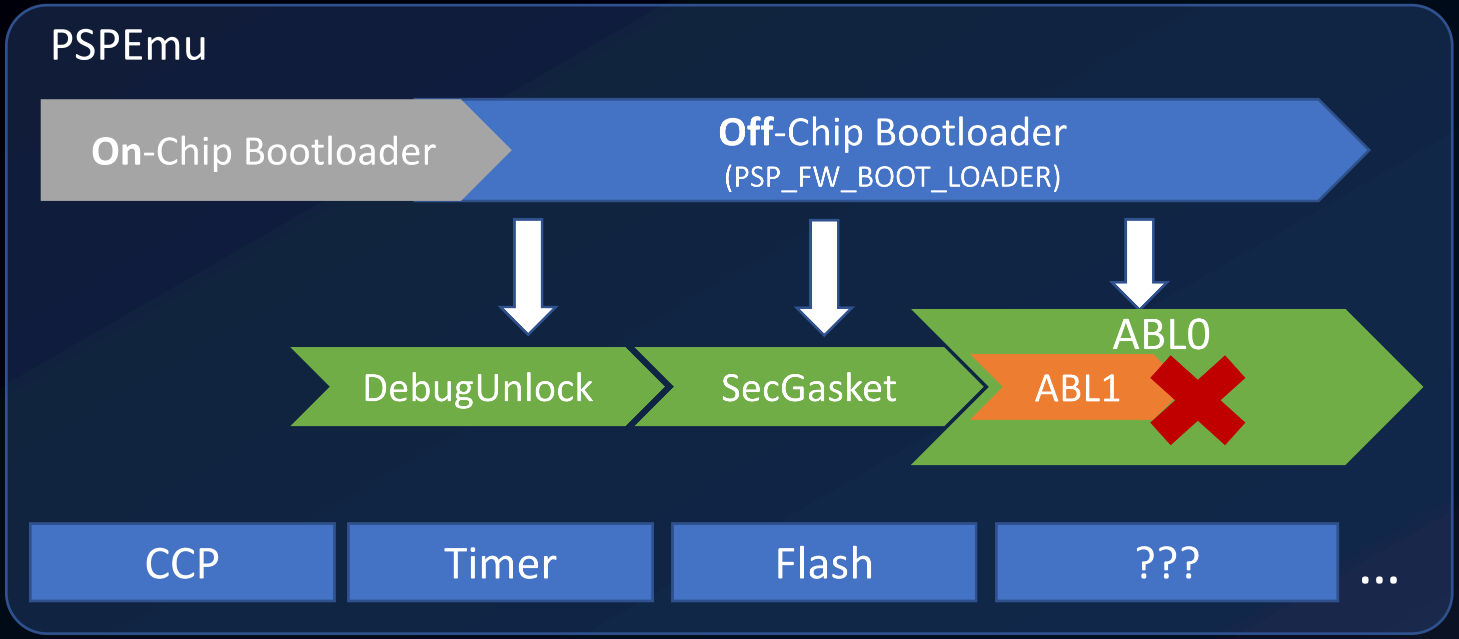
Fail

Analyze

Implement

+337 -2 🟩🟩🟩🟩⬜

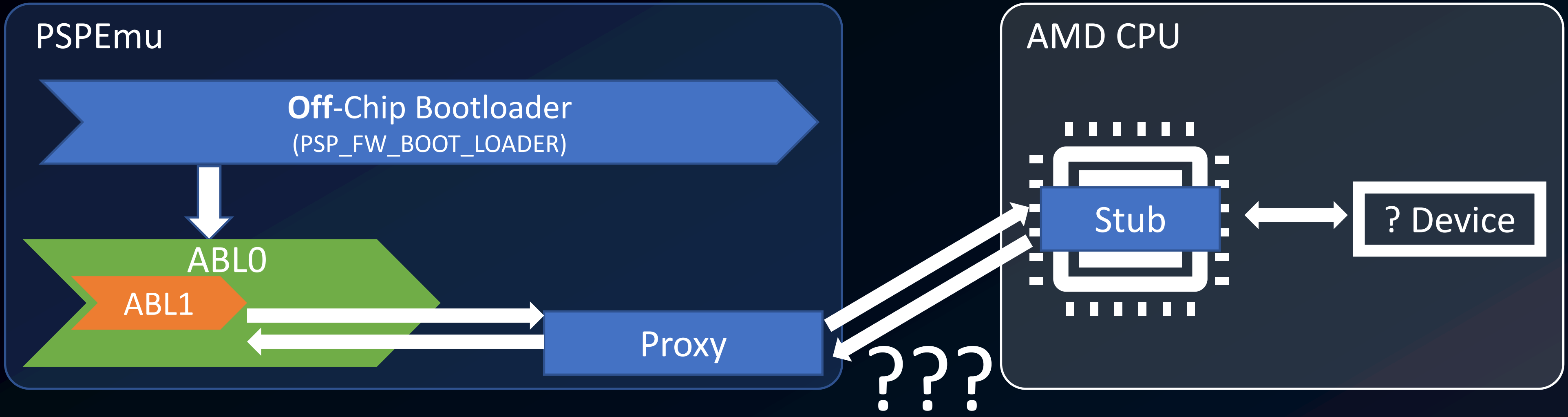
}



SUCCESS! (KIND OF)

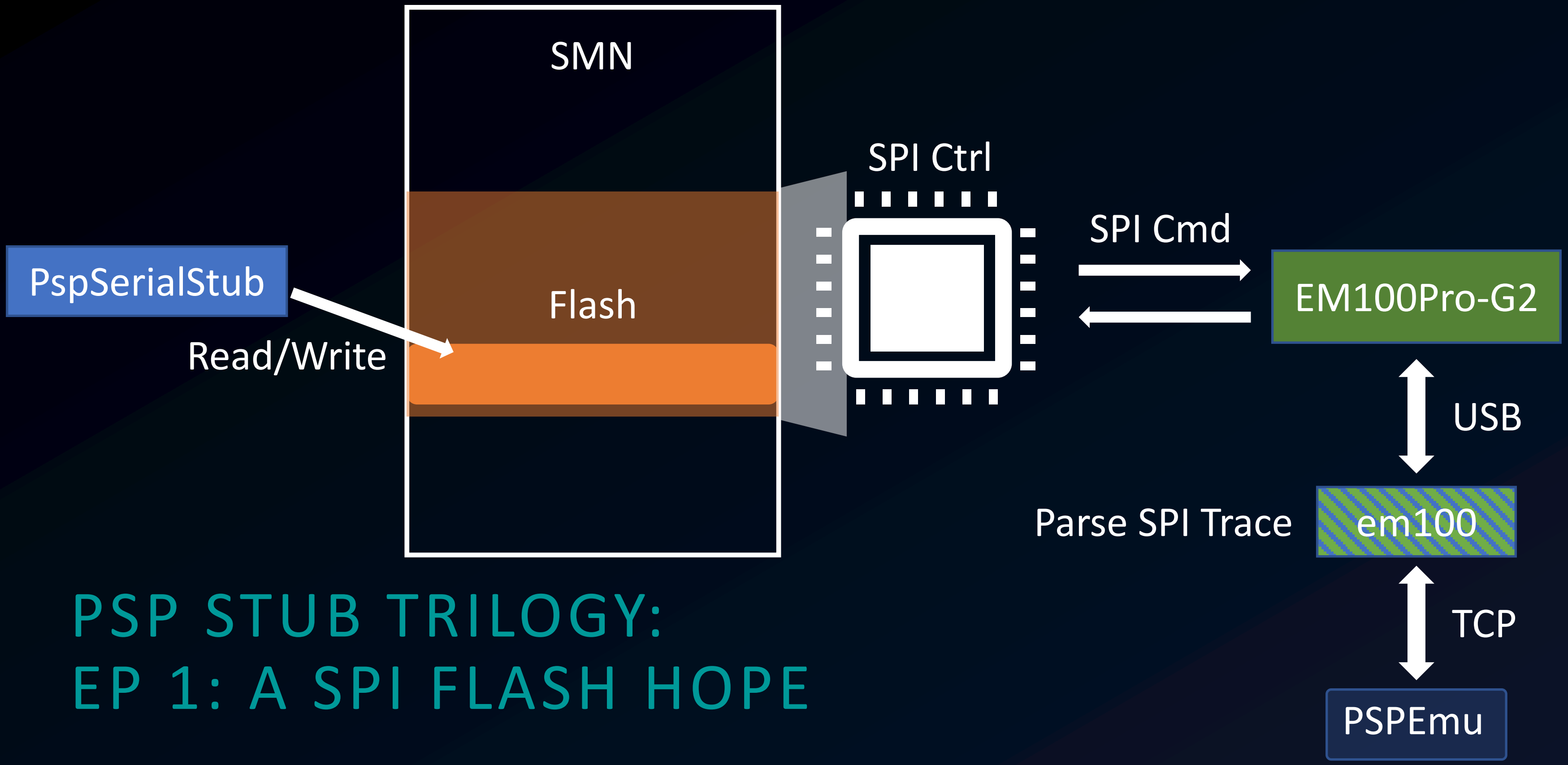
- On Chip BL completes
- Off Chip BL starts and executes first two apps
- Off Chip BL executes first ABL stage but what next?
- Emulating all devices not feasible

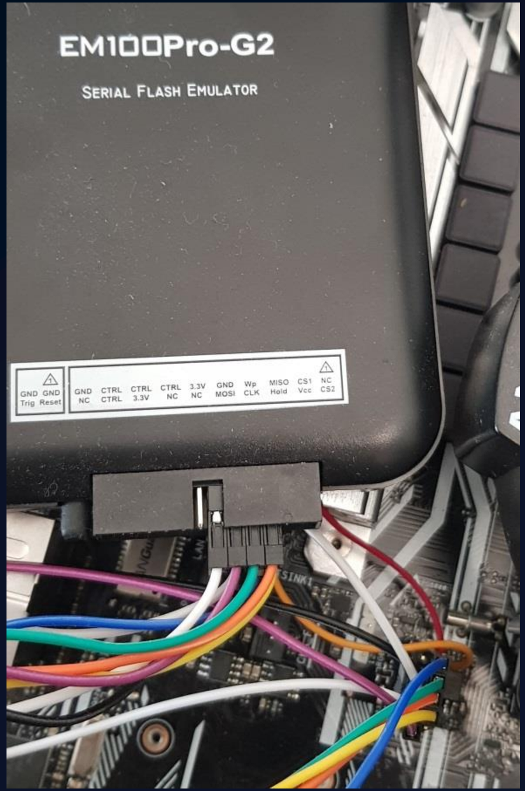
```
[…]
INFO    STS 0x000057c6[0x00001165][  SVC, S, M, I,NF,0x00014000] STRING "POST CODE (PSP): PSPSTATUS_C2P_MASTER_INITIALIZED_SLAVE_WAITED_FOR_MASTER"
INFO    STS 0x000057c6[0x0000122f][  SVC, S, M, I,NF,0x00014000] STRING "POST CODE (PSP): PSPSTATUS_MASTER_GOT_BOOT_MODE_AND_SENT_TO_ALL_SLAVES“
INFO    STS 0x000057c6[0x000057bf][  SVC, S, M, I,NF,0x00014000] STRING "POST CODE (PSP): PSPSTATUS_BOOTLOADER_SUCCESSFULLY_ENTERED_C_MAIN“
INFO    STS 0x000057c6[0x000057bf][  SVC, S, M, I,NF,0x00014000] STRING "POST CODE (PSP): PSPSTATUS_HMAC_KEY_DERIVED_SUCCESSFULLY“
INFO    STS 0x000057c6[0x000057bf][  SVC, S, M, I,NF,0x00014000] STRING "POST CODE (PSP): PSPSTATUS_SPIROM_INITIALIZED_SUCCESSFULLY“
INFO    STS 0x000057c6[0x000057bf][  SVC, S, M, I,NF,0x00014000] STRING "POST CODE (PSP): PSPSTATUS_BIOS_DIRECTORY_READ_FROM_SPI_TO_SRAM“
INFO    STS 0x000057c6[0x000057bf][  SVC, S, M, I,NF,0x00014000] STRING "POST CODE (PSP): PSPSTATUS_EARLY_UNLOCK_CHECK“
INFO    STS 0x000057c6[0x000057bf][  SVC, S, M, I,NF,0x00014000] STRING "POST CODE (PSP): PSPSTATUS_BOOTLOADER_PROGRAMMED_MBAT_TABLE_SUCCESSFULLY“
INFO    STS 0x000057c6[0x000057bf][  SVC, S, M, I,NF,0x00014000] STRING "POST CODE (PSP): PSPSTATUS_SECURITY_GASKET_BINARY_VALIDATED_AND_EXECUTED“
INFO    STS 0x000057c6[0x000057bf][  SVC, S, M, I,NF,0x00014000] STRING "POST CODE (PSP): PSPSTATUS_BOOTLOADER_LOADED_SMU_FW_SUCCESSFULLY“
INFO    STS 0x000057c6[0x000057bf][  SVC, S, M, I,NF,0x00014000] STRING "POST CODE (PSP): PSPSTATUS_MP1_TAKEN_OUT_OF_RESET“
INFO    STS 0x000057c6[0x000057bf][  SVC, S, M, I,NF,0x00014000] STRING "POST CODE (PSP): PSPSTATUS_PSP_AND_SMU_CONFIGURED_WAFFLE“
INFO    STS 0x000057c6[0x000057bf][  SVC, S, M, I,NF,0x00014000] STRING "POST CODE (PSP): PSPSTATUS_FW_VALIDATION_COMPLETED“
INFO    STS 0x000057c6[0x000057bf][  SVC, S, M, I,NF,0x00014000] STRING "POST CODE (PSP): PSPSTATUS_BOOTLOADER_LOADED_AGESA0_FROM_SPIROM_SUCCESSFULLY“
[…]
```



PROXY MODE

- Passthrough hardware accesses to real hardware
- Stub running on real PSP
- Reads/Writes to devices get captured by generic proxy component and forwarded to the real hardware
- Which communication channel to use?

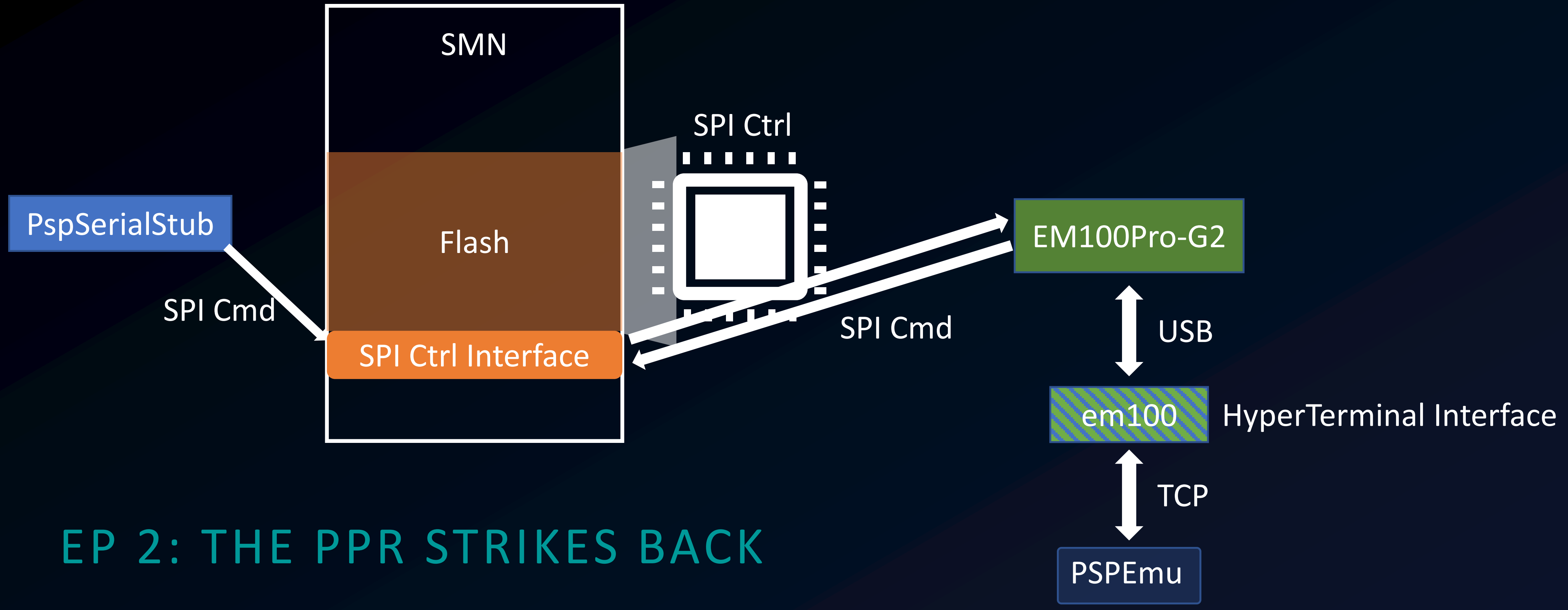




PSP STUB TRILOGY:
EP 1: A SPI FLASH HOPE

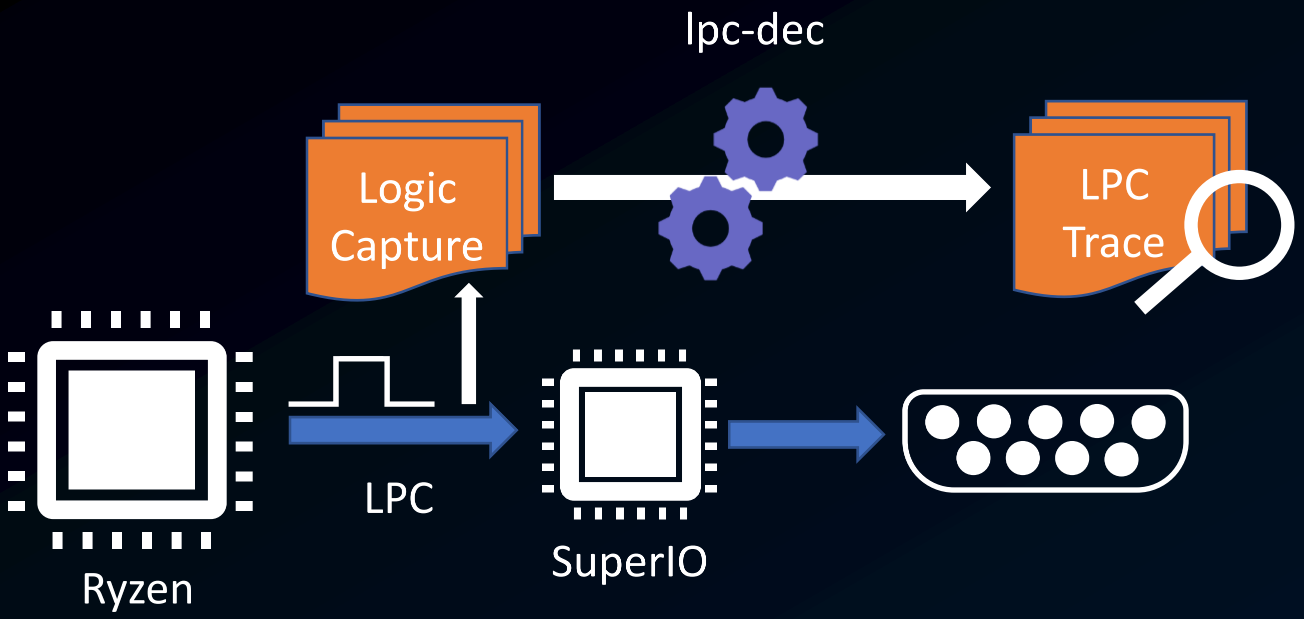
- Use SPI Flash interface and emulator
- Exchange data using SPI Flash Read and Page Program requests
- Works reliable but slow (2-3 accesses per second)
- Requires an expensive flash emulator

https://github.com/PSPReverse/em100/tree/network-mode-v1



EP 2: THE PPR STRIKES BACK

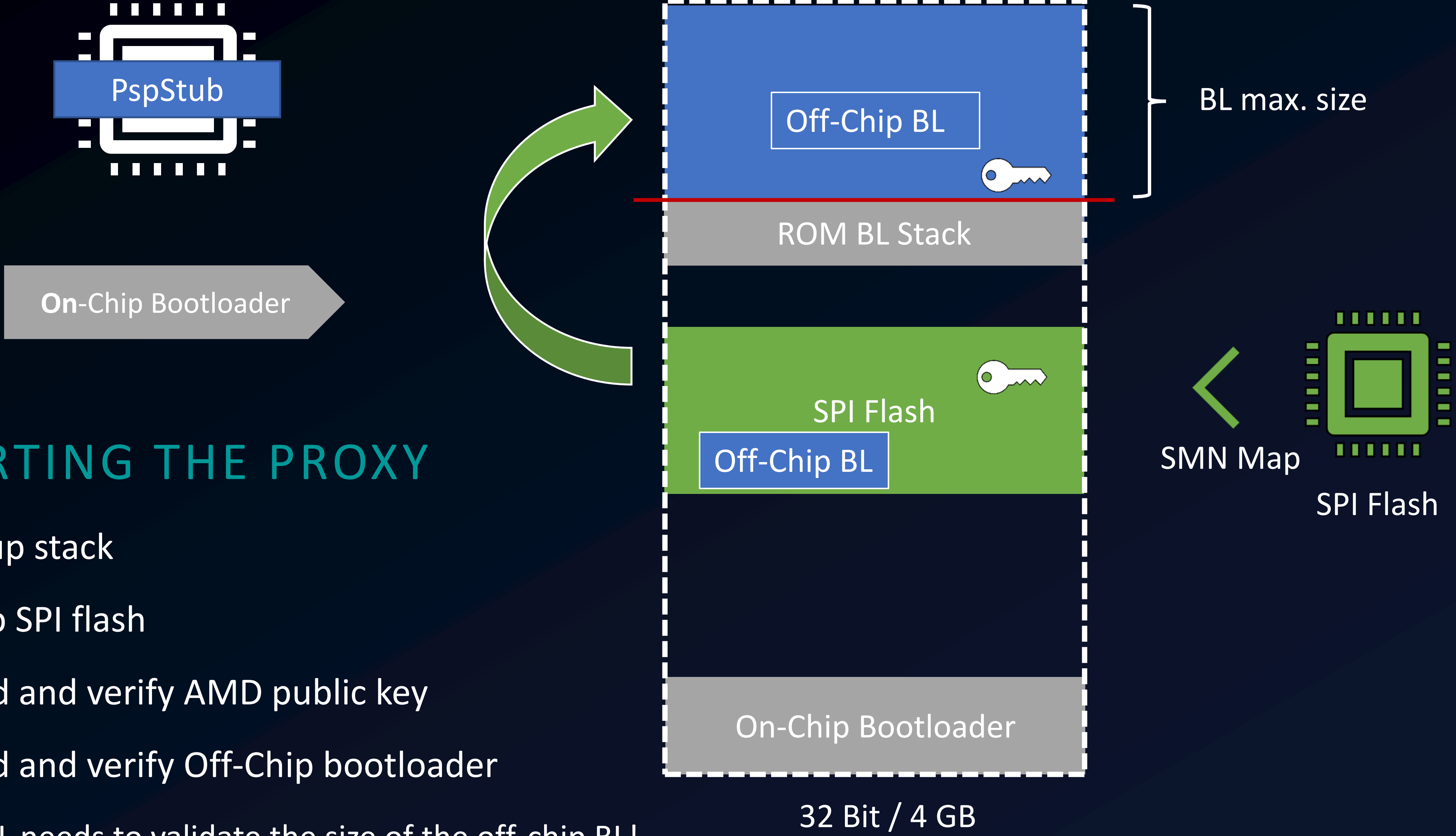
- Found AMD PPR
- Explains Low Level SPI register interface
- We can execute arbitrary commands now!
- Enables use of DediProg EM100 Hyper Terminal
- Blazingly fast (don‘t forget to disable Nagle for TCP!)
- Still requires an expensive flash emulator ☹



EP 3: RETURN OF THE UART

- Explore use of the legacy UART for a low cost solution
- SuperIO chip attached to the SoC via LPC
- Need correct sequence to enable UART
- Analyze SuperIO accesses over the LPC bus from logic capture (lpc-dec)
- Much slower than SPI but very cheap

https://github.com/AlexanderEichner/lpc-dec

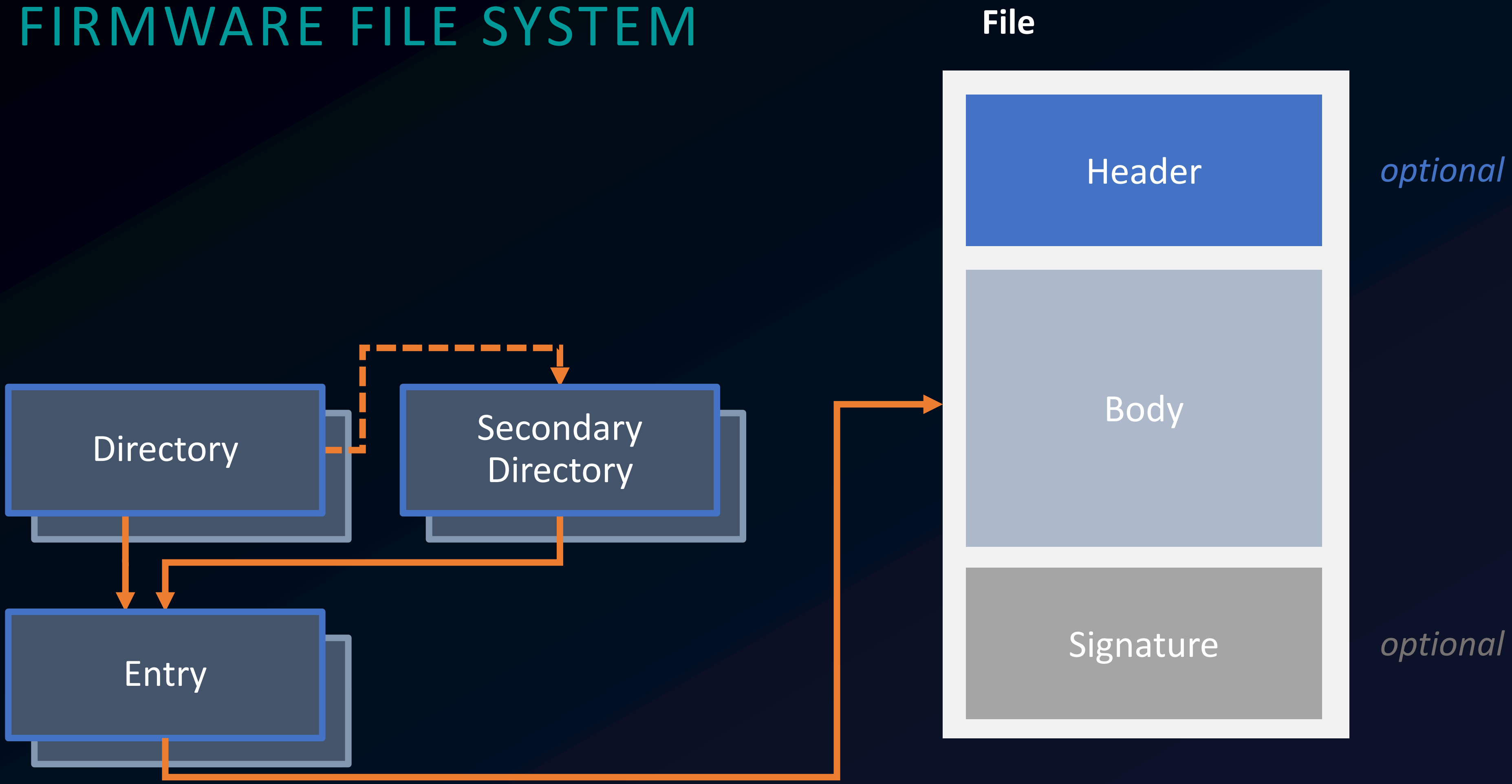


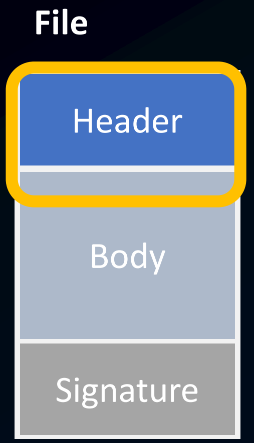
INSERTING THE PROXY

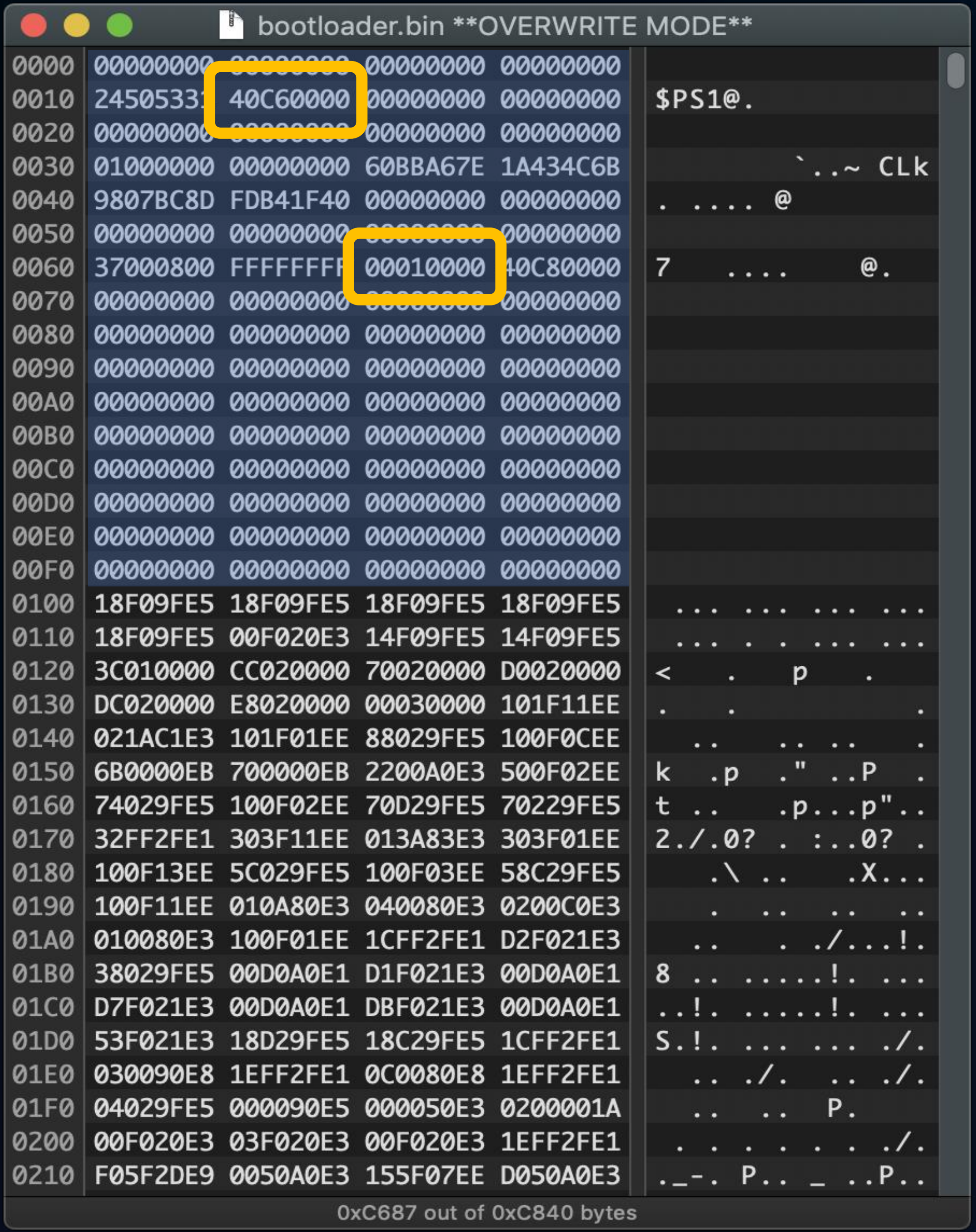
1. Setup stack
2. Map SPI flash
3. Load and verify AMD public key
4. Load and verify Off-Chip bootloader

The On-Chip BL needs to validate the size of the off-chip BL!

FIRMWARE FILE SYSTEM



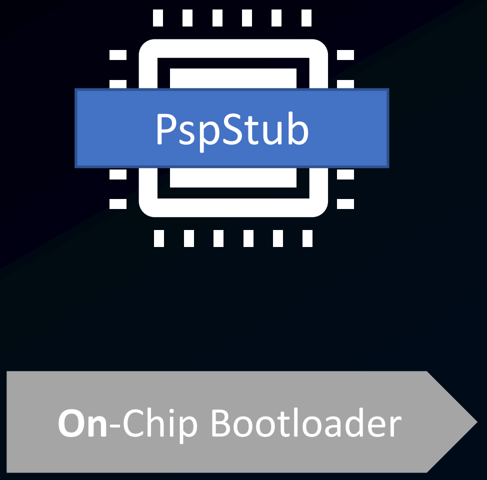



bootloader.bin **OVERWRITE MODE**

```
0000 00000000 00000000 00000000 00000000
0010 24505331 40C60000 00000000 00000000   $PS1@.
0020 00000000 00000000 00000000 00000000
0030 01000000 00000000 60BBA67E 1A434C6B            `..~ CLk
0040 9807BC8D FDB41F40 00000000 00000000   . .... @
0050 00000000 00000000 00000000 00000000
0060 37000800 FFFFFFFF 00010000 40C80000   7   ....     @.
0070 00000000 00000000 00000000 00000000
0080 00000000 00000000 00000000 00000000
0090 00000000 00000000 00000000 00000000
00A0 00000000 00000000 00000000 00000000
00B0 00000000 00000000 00000000 00000000
00C0 00000000 00000000 00000000 00000000
00D0 00000000 00000000 00000000 00000000
00E0 00000000 00000000 00000000 00000000
00F0 00000000 00000000 00000000 00000000
0100 18F09FE5 18F09FE5 18F09FE5 18F09FE5    ... ... ... ...
0110 18F09FE5 00F020E3 14F09FE5 14F09FE5    ... . . ... ...
0120 3C010000 CC020000 70020000 D0020000   <   .   p   .
0130 DC020000 E8020000 00030000 101F11EE   .   .           .
0140 021AC1E3 101F01EE 88029FE5 100F0CEE     ..    .. ..   .
0150 6B0000EB 700000EB 2200A0E3 500F02EE   k  .p  ." ..P  .
0160 74029FE5 100F02EE 70D29FE5 70229FE5   t ..    .p...p"..
0170 32FF2FE1 303F11EE 013A83E3 303F01EE   2./.0? . :..0? .
0180 100F13EE 5C029FE5 100F03EE 58C29FE5       .\ ..    .X...
0190 100F11EE 010A80E3 040080E3 0200C0E3      .   ..   ..   ..
01A0 010080E3 100F01EE 1CFF2FE1 D2F021E3     ..     . ./...!.
01B0 38029FE5 00D0A0E1 D1F021E3 00D0A0E1   8 .. .....!. ...
01C0 D7F021E3 00D0A0E1 DBF021E3 00D0A0E1   ..!. .....!. ...
01D0 53F021E3 18D29FE5 18C29FE5 1CFF2FE1   S.!. ... ... ./.
01E0 030090E8 1EFF2FE1 0C0080E8 1EFF2FE1     .. ./.   .. ./.
01F0 04029FE5 000090E5 000050E3 0200001A     ..   ..   P.
0200 00F020E3 03F020E3 00F020E3 1EFF2FE1    . . . . . . ./.
0210 F05F2DE9 0050A0E3 155F07EE D050A0E3   ._-. P.. _ ..P..
```

0xC687 out of 0xC840 bytes

- Offset 0x14: Bodysize (0xc640)
- Offset 0x64: Load address (0x100)

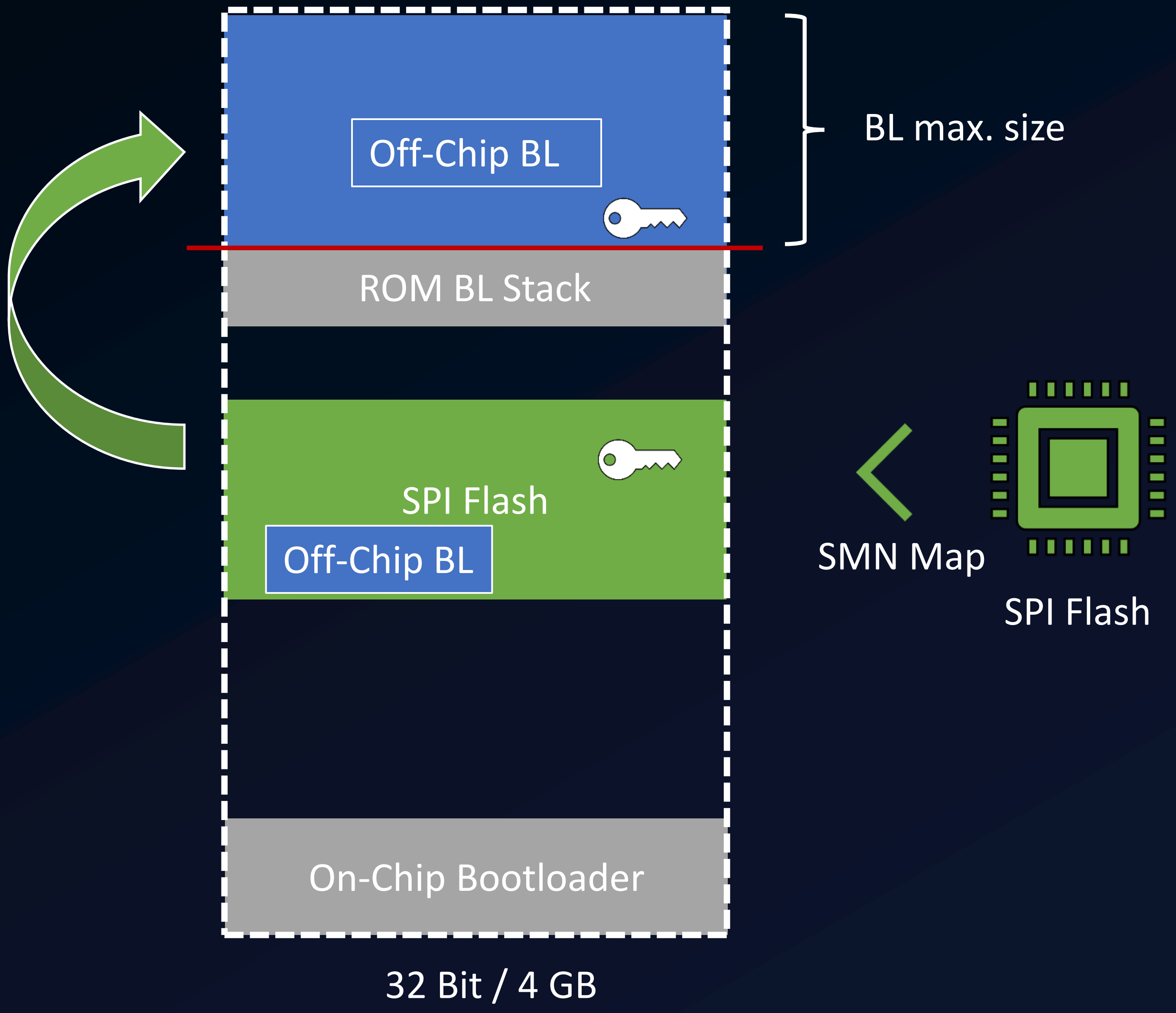


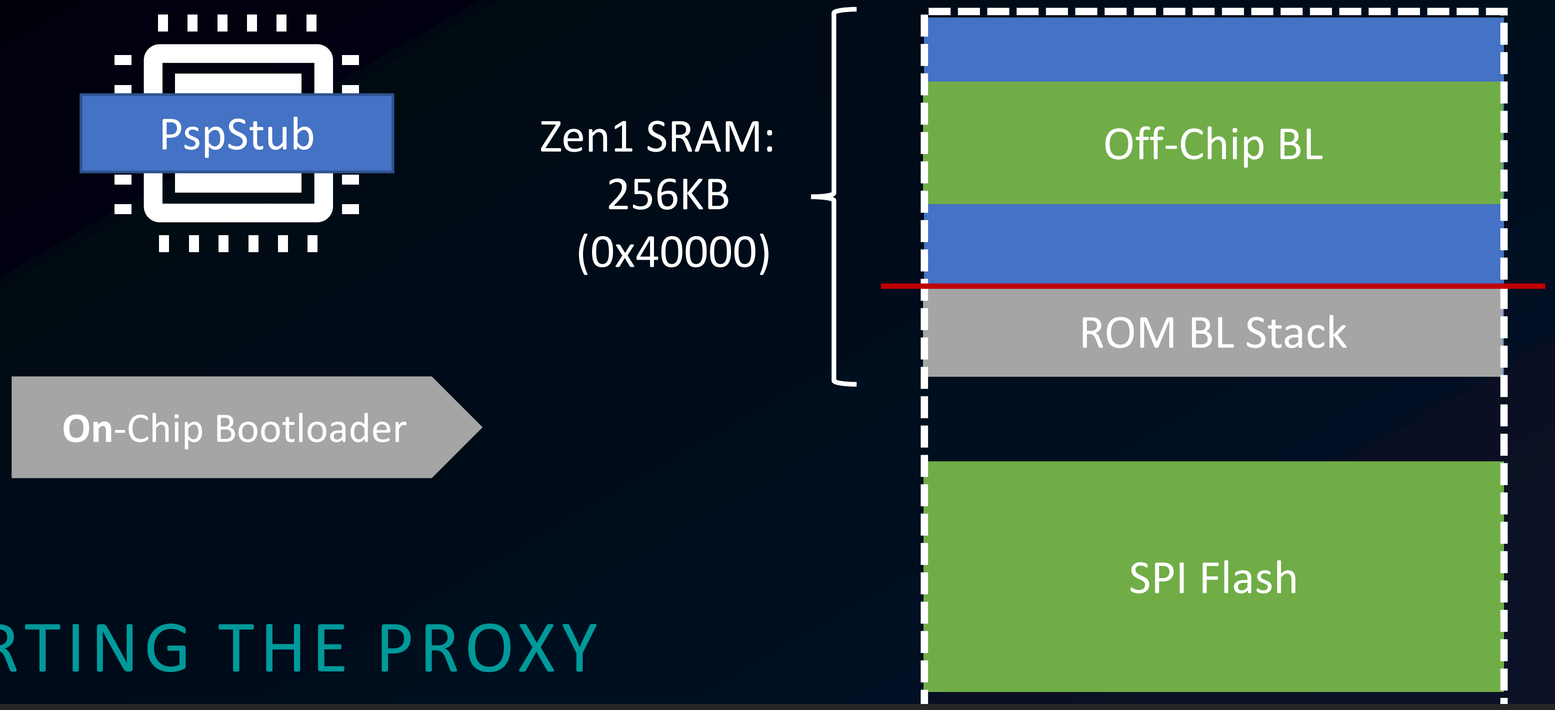
INSERTING THE PROXY

Off-Chip BL is copied into SRAM and *then* verified (to avoid TOCTOU).

The header is processed *before* the signature is checked.

-> Input validation is required.





abs(0x100) +
abs(0xc640)

= 0xc740

abs(0x100) +
abs(0x8000c640)

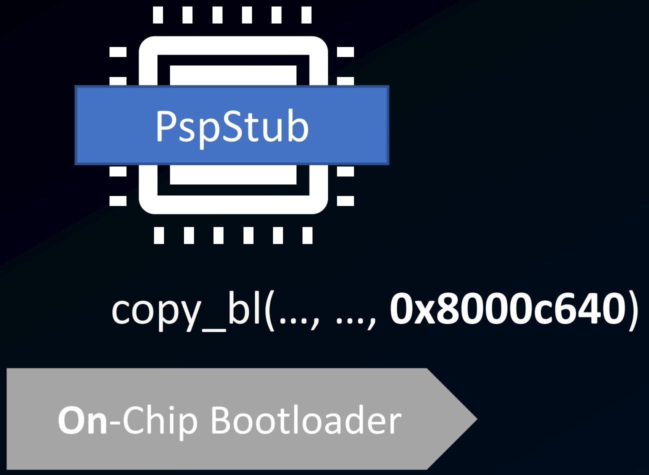
= 0xc740

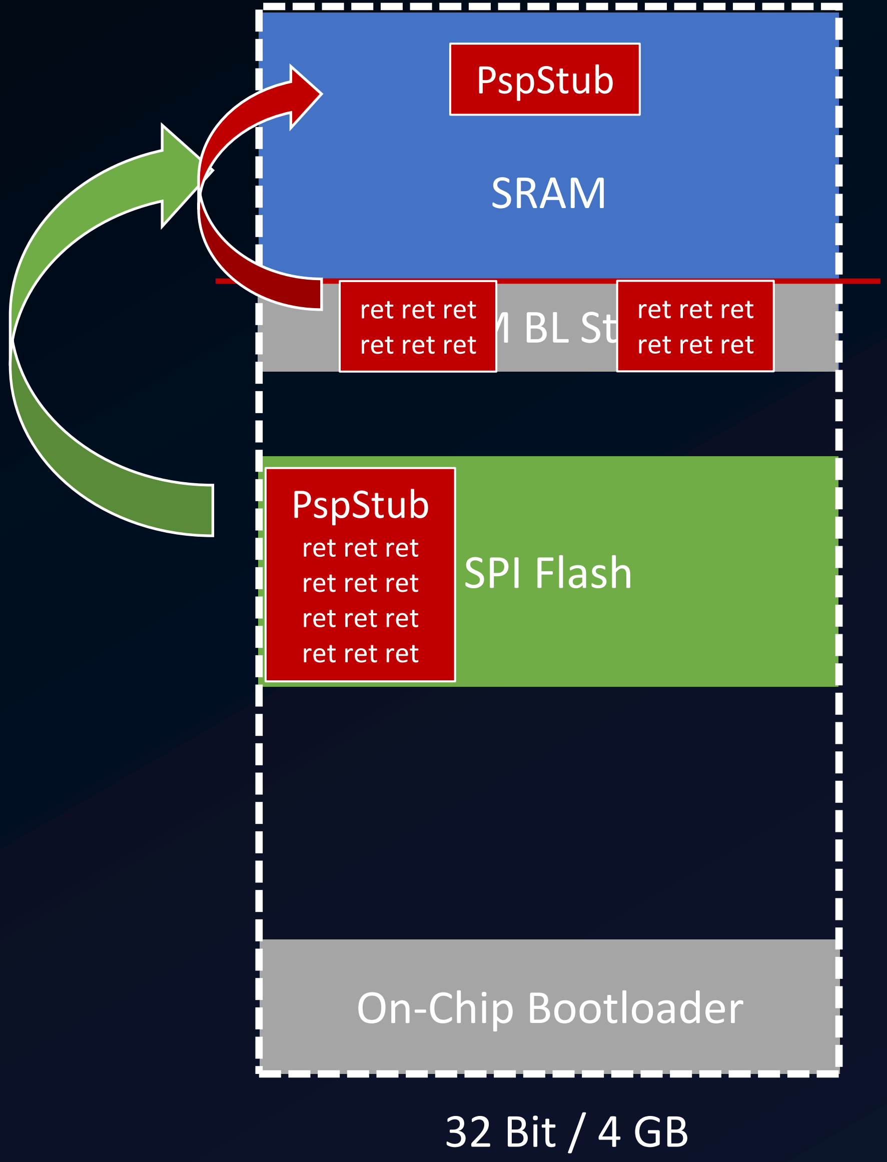
INSERTING THE PROXY

```
void load_off_chip_bl (void) {
  ...
  if (abs(load_address) + abs(body_size) > ROM_BL_STACK)
    return -1;
  ...
  copy_bl(load_address, spi_src, body_size)
  return 0;
}
```

Validated size: 0xc640

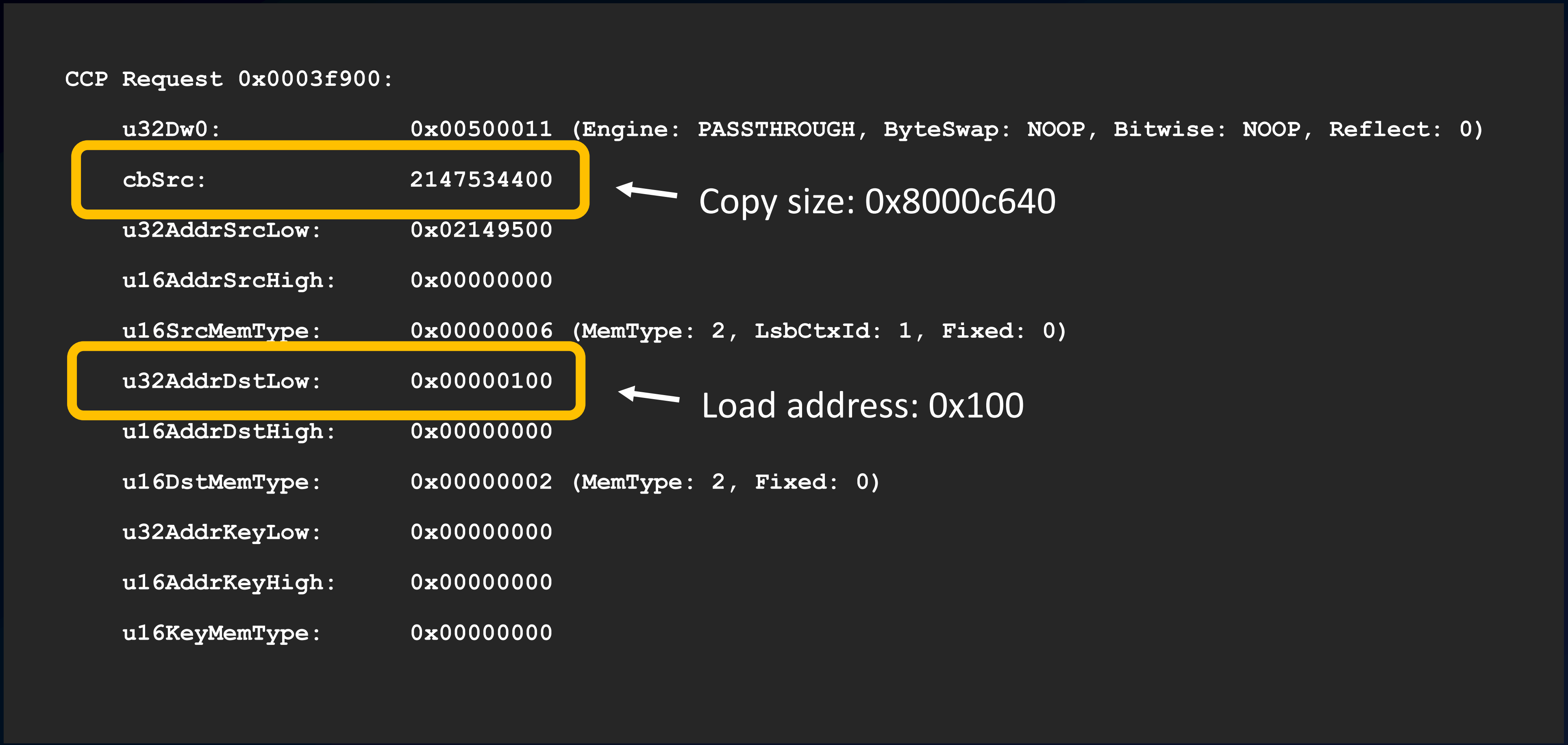
Used size: 0x8000c640





INSERTING THE PROXY

1. Place PspStub in SPI flash
   - Appended with return addresses
2. Flip sign-bit of body size
3. Success!



```
CCP Request 0x0003f900:
    u32Dw0:                 0x00500011 (Engine: PASSTHROUGH, ByteSwap: NOOP, Bitwise: NOOP, Reflect: 0)
    cbSrc:                  2147534400
    u32AddrSrcLow:          0x02149500
    u16AddrSrcHigh:         0x00000000
    u16SrcMemType:          0x00000006 (MemType: 2, LsbCtxId: 1, Fixed: 0)
    u32AddrDstLow:          0x00000100
    u16AddrDstHigh:         0x00000000
    u16DstMemType:          0x00000002 (MemType: 2, Fixed: 0)
    u32AddrKeyLow:          0x00000000
    u16AddrKeyHigh:         0x00000000
    u16KeyMemType:          0x00000000
```

Copy size: 0x8000c640

Load address: 0x100

~~Issue~~ Feature summary

~~AFFECTED~~ SUPPORTED SYSTEMS

- Zen and Zen+ CPUs (probably)
  - Confirmed:
    - Zen: Ryzen 1700X, Epyc 7281
    - Zen+: Ryzen PRO 3500U, Ryzen 5 2600
- Zen2 is NOT affected

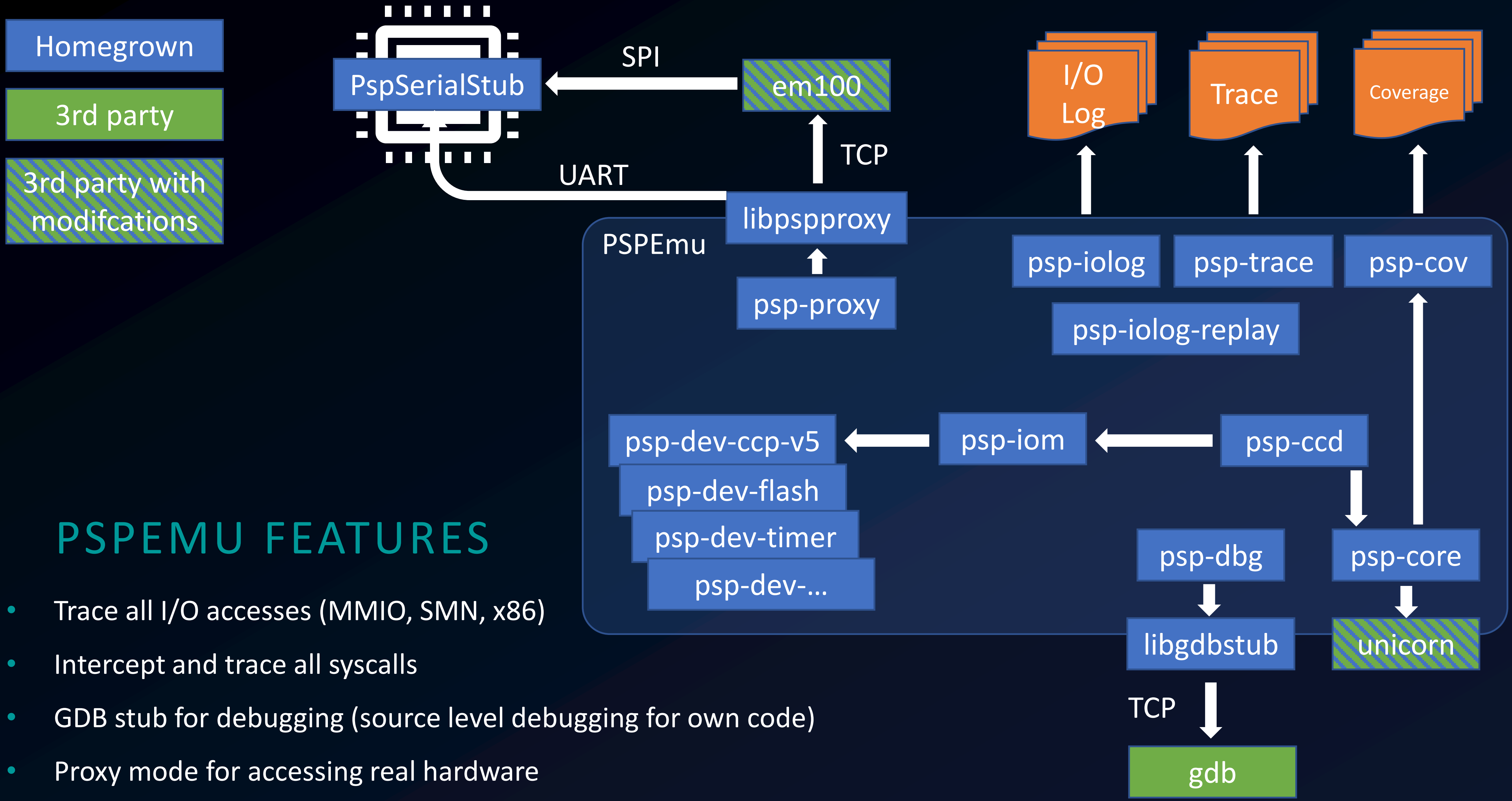
DISCLOSURE TIMELINE

- Reported to AMD 26th February 2020
- …
- Response: 11th May 2020!
- Known bug
  - “AMD has developed mitigations in various products where appropriate.”

PSPEMU BASICS

```
./PSPEmu

    --emulation-mode on-chip-bl

    --flash-rom uefi.ROM

    --on-chip-bl on-chip-bl.bin

    --trace-log /tmp/log

    --trace-svcs

    --dbg <port>
```

- Sets starting point in boot process
- Flash image to use for emulated SPI flash
- Sets on chip BL binary
- Trace log destination
- Configures syscall tracing
- GDB stub



PSPEMU FEATURES

- Trace all I/O accesses (MMIO, SMN, x86)
- Intercept and trace all syscalls
- GDB stub for debugging (source level debugging for own code)
- Proxy mode for accessing real hardware
- Create coverage traces for later analysis
- I/O record and replay

CURRENT STATE

```
WARNING! BIOS Recovery mode has been detected,
Please put the file "X370PRO.CAP" into HDD or a removable USB media device,
 And then reset your computer.
You can also insert ASUS Support CD to your CD-ROM and reset your computer,
If you have done these, Please wait a moment and don't shutdown system!!!
System will be automatically rebooted
```

Working:

- Bootstrap platform when in proxy mode
  - DRAM works!
  - Ryzen 1700X (Zen)
- Stable communication channel with PSP
  - Fast but expensive
  - Slow but very cheap
- Toolchain for writing and debugging your own code
- I/O log record and replay (no access to real hardware required for first steps)
- Basic micropython port for the PSP ☺

Todo:

- Full platform boot (with UEFI)
- Emulate multiple CCDs/PSPs
- Support multiple CCDs/PSPs in the stub
- Investigate SecureOS on Ryzen
- Test Zen+/Zen2 support
- Zen3?

MAY THE CODE BE WITH YOU

- https://github.com/PSPReverse/PSPEmu - Main emulator
- https://github.com/PSPReverse/libpspproxy - PSP proxy base library
- https://github.com/PSPReverse/unicorn - Patched unicorn
- https://github.com/PSPReverse/psp-apps - Contains the PSP stub
- https://github.com/PSPReverse/em100 - For the flash emulator transport channel
- https://github.com/PSPReverse/PSPTool - Analyze UEFI images
- https://github.com/AlexanderEichner/libgdbstub - Generic portable GDB stub library
- https://github.com/AlexanderEichner/micropython - Port of micropython to the PSP