Uncover, Understand, Own

REGAINING CONTROL OVER YOUR AMD CPU

Uncover

Christian Werling
Security Research Labs

Understand

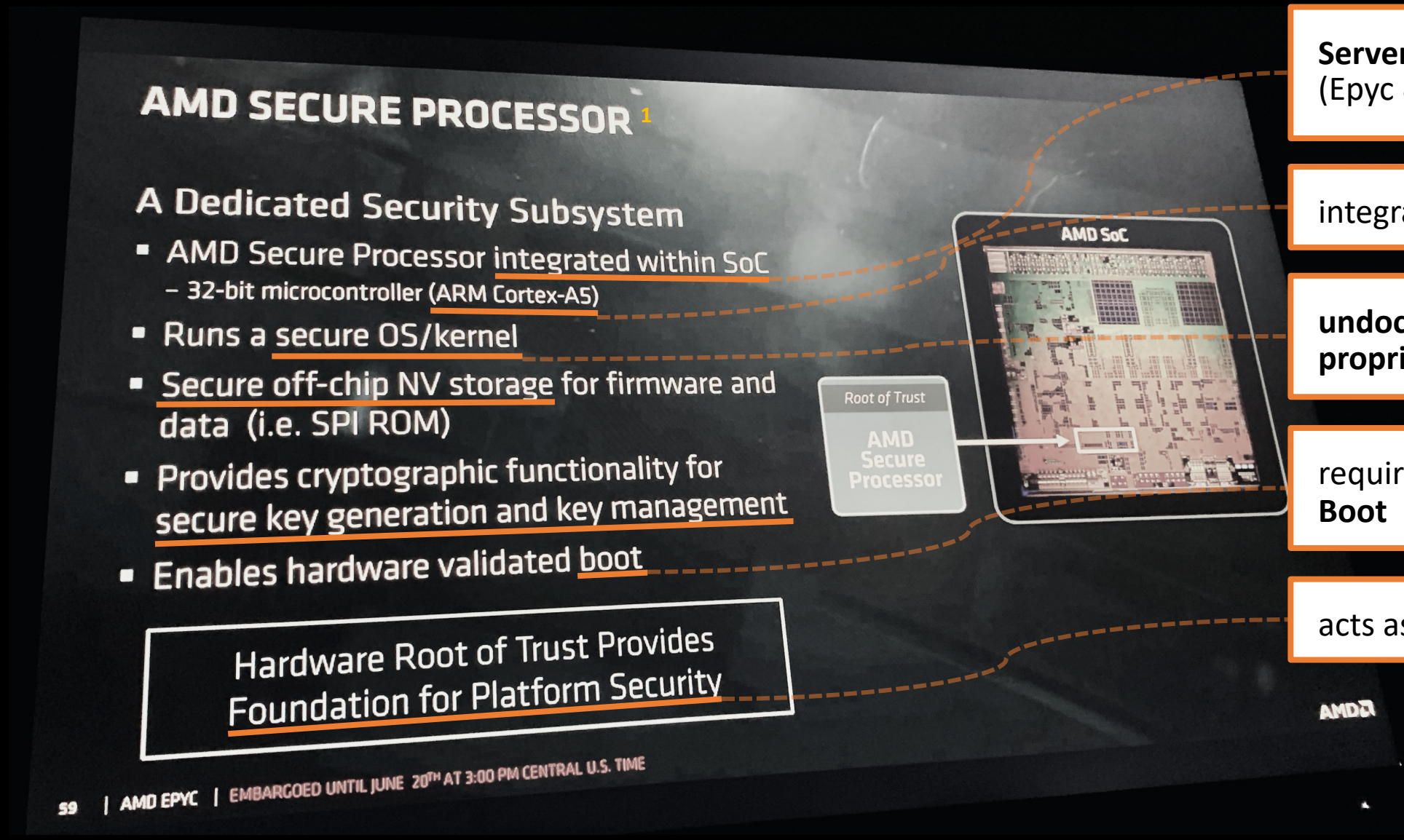
Alexander Eichner
Technische Universität Berlin

Own

Robert Buhren
Technische Universität Berlin

Uncover

REVERSE-ENGINEERING AN UNKNOWN SUBSYSTEM



Server & Desktops
(Epyc & Ryzen)

integrated **since 2013**

undocumented, **proprietary** firmware

required for **Secure Boot**

acts as **trust anchor**

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

Applications

SECURE ENCRYPTED VIRTUALIZATION

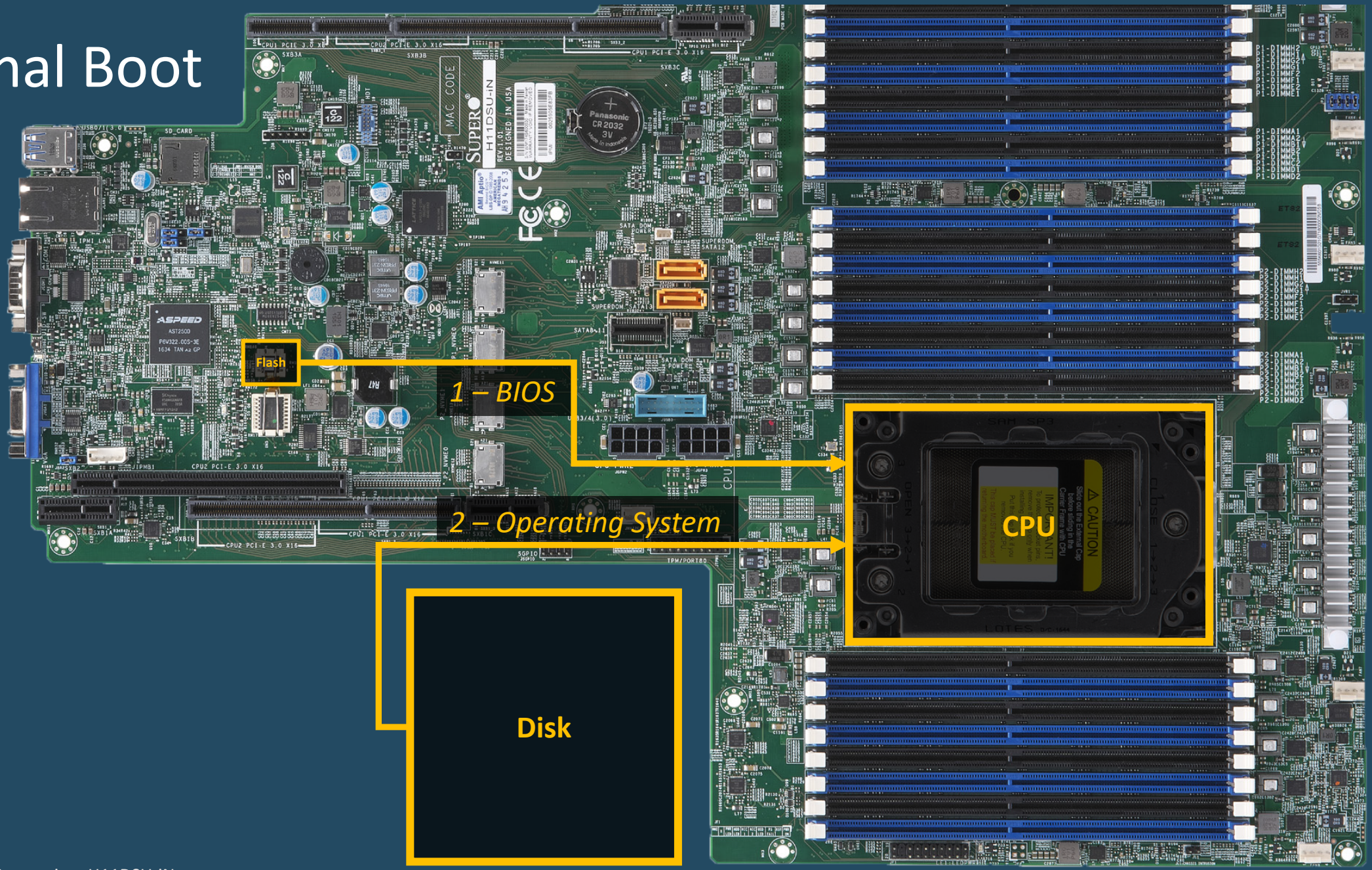
- **SEV p**rotects virtual machines in **untrusted** physical locations (e.g. data centers)
- The **PSP** acts as **remote trusted entity** for the Cloud customer
- PSP promises to **protect VM memory** from the **hypervisor** and even **physical access**

TRUSTED EXECUTION ENVIRONMENT

- Linux to support **PSP TEE API** (kernel patch pending)
- The **PSP** acts as a **black box** inside **your** system that is **trusted by an external entity** (e.g. Netflix)
- This enables DRM on **untrusted** systems like Linux

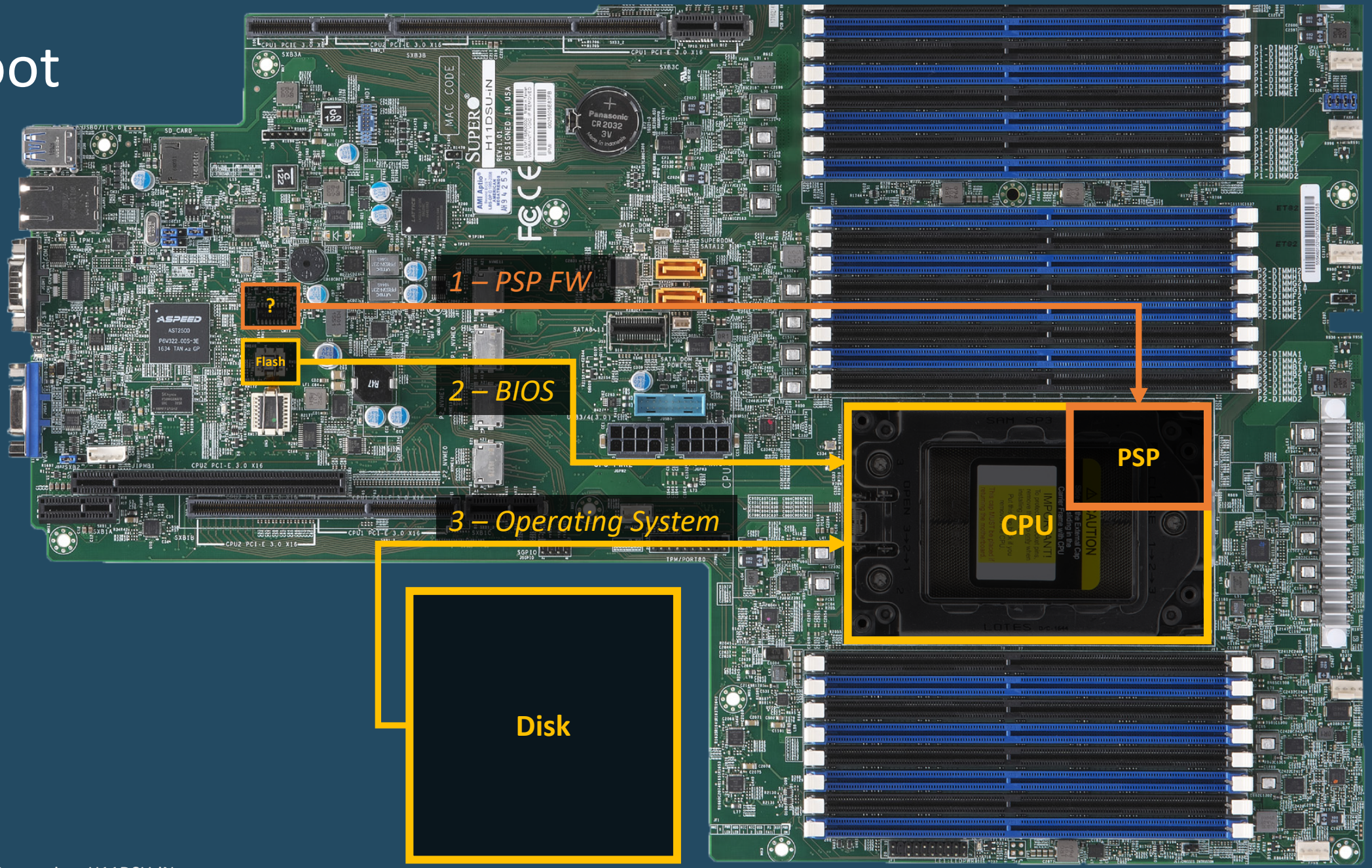
The PSP runs code you
don't know *and* ***don't control.***

Traditional Boot



Source: Motherboard Manual Supermicro H11DSU-iN

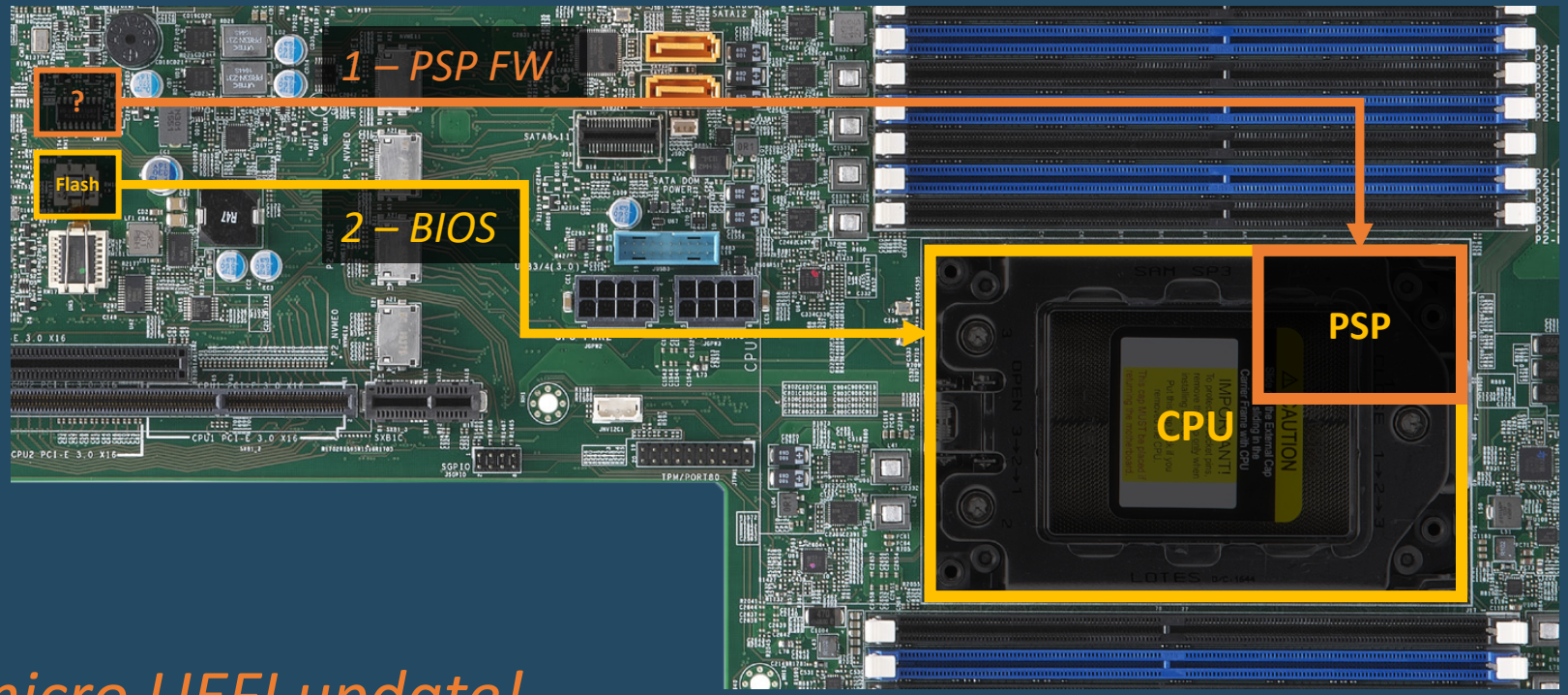
AMD Boot



Source: Motherboard Manual Supermicro H11DSU-iN

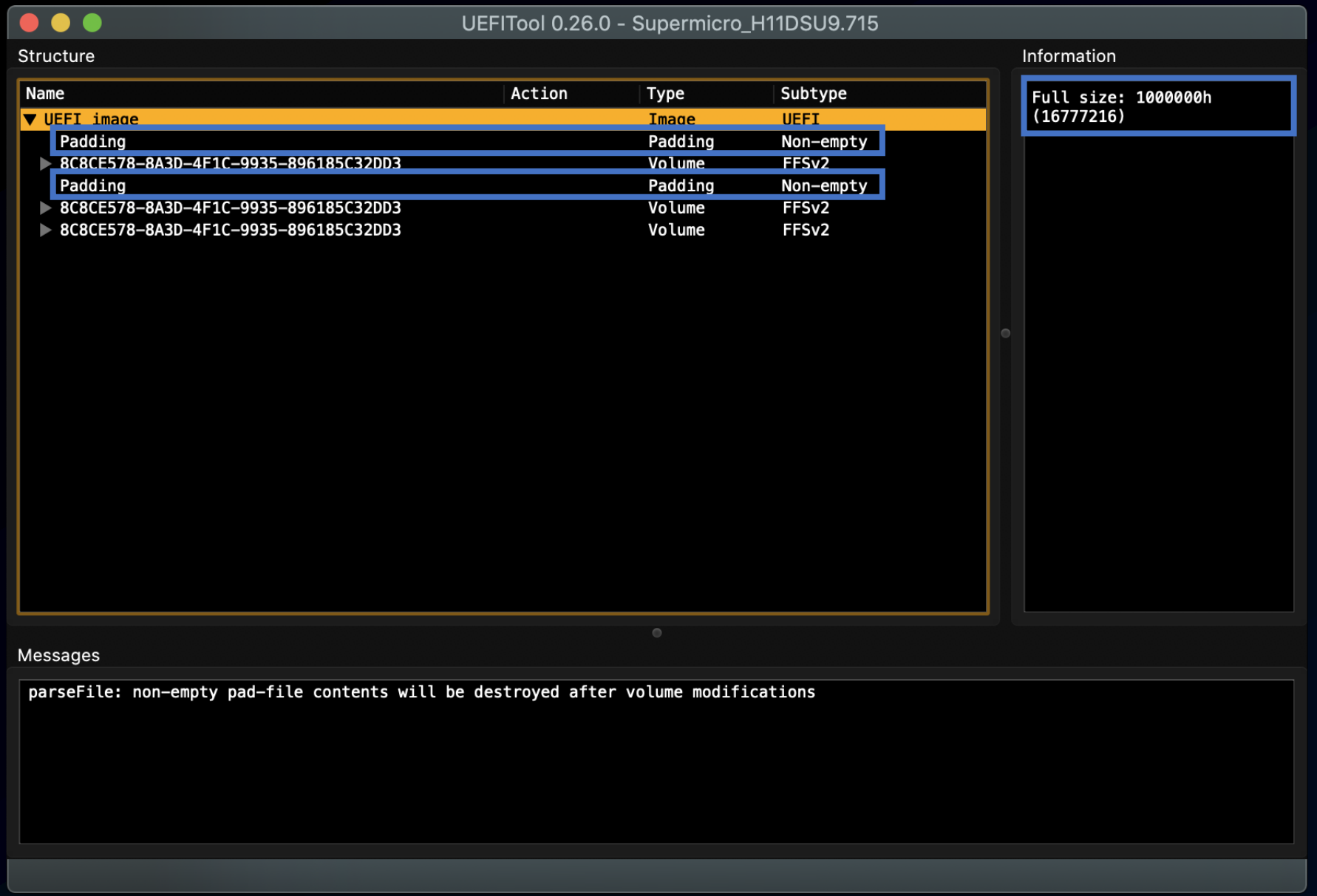
Where is the PSP Firmware loaded from?

- The BIOS is stored in **SPI flash memory**
- It contains all **code and data used by the BIOS** during boot up
- Data is arranged according to the UEFI image specification



Let's inspect a Supermicro UEFI update!

Source: Motherboard Manual Supermicro H11DSU-iN

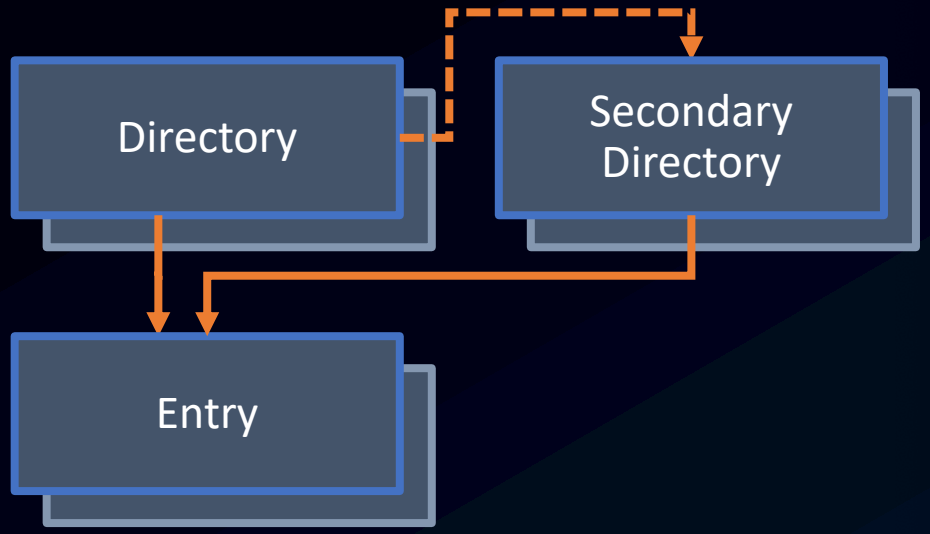


https://github.com/LongSoft/UEFITool

```
$ binwalk -A Supermicro_H11DSU9.715

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
489764        0x77924         ARM instructions, function prologue
489836        0x7796C         ARM instructions, function prologue
489852        0x7797C         ARM instructions, function prologue
489868        0x7798C         ARM instructions, function prologue
489964        0x779EC         ARM instructions, function prologue
489976        0x779F8         ARM instructions, function prologue
[...]
14405063      0xDBCDC7        Intel x86 instructions, nops
14405071      0xDBCDCF        Intel x86 instructions, nops
14405079      0xDBCDD7        Intel x86 instructions, nops
14405087      0xDBCDDF        Intel x86 instructions, nops
14405095      0xDBCDE7        Intel x86 instructions, nops
[...]
```

https://github.com/ReFirmLabs/binwalk

FIRMWARE FILE SYSTEM



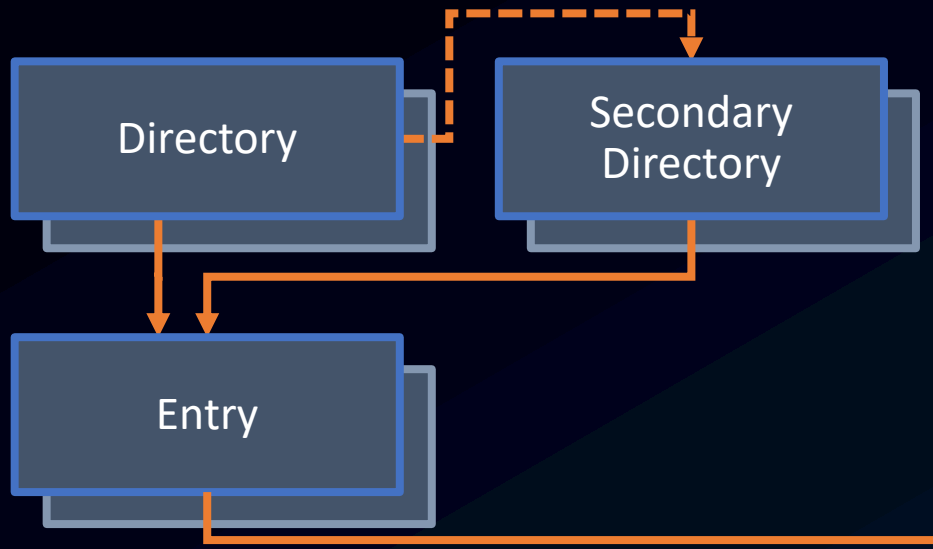
Supermicro_H11DSU9.715

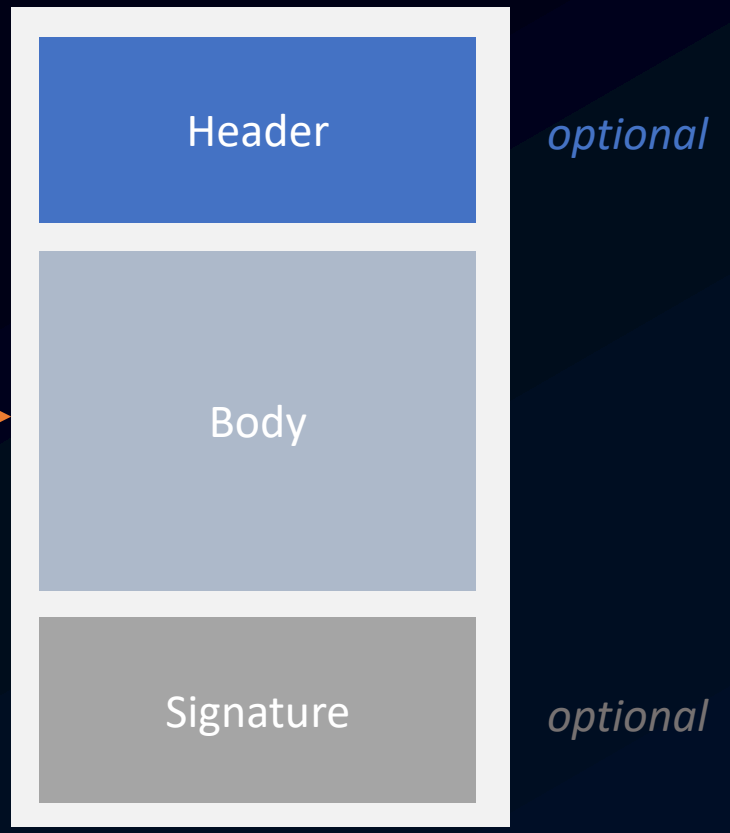
```
0076FE0 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ................
0076FF0 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ................
0077000 Magic    Checksum 10000000 A0040000 $PSPLg..    .
0077010 Type     Size     Address  ?        @    t .
0077020 Type     ...      009414FF 00000000          . .
0077030 ...      80E70000 007707FF 00000000    ..    w .
0077040 08000000 40E10100 005F08FF 00000000    @.    _ .
0077050 0A000000 40030000 00410AFF 00000000    @    A .
0077060 12000000 40560000 00450AFF 00000000    @V   E .
0077070 21000000 10000000 009C0AFF 00000000 !          . .
0077080 24000000 000C0000 009D0AFF 00000000 $          . .
0077090 30000000 200C0000 00A90AFF 00000000 0          . .
00770A0 31000000 20C00000 00B60AFF 00000000 1    .     . .
00770B0 32000000 F0B80000 00770BFF 00000000 2    ..    w .
00770C0 33000000 70DE0000 00300CFF 00000000 3    p.    0 .
00770D0 34000000 A0F10000 000F0DFF 00000000 4    ..        .
00770E0 35000000 A0F00000 00010EFF 00000000 5    ..        .
00770F0 36000000 40C00000 00F20EFF 00000000 6    @.    . .
0077100 Pointer to Secondary Directory      @          . .
0077110 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ................
0077120 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ................
```

0x76FCD out of 0x1000000 bytes

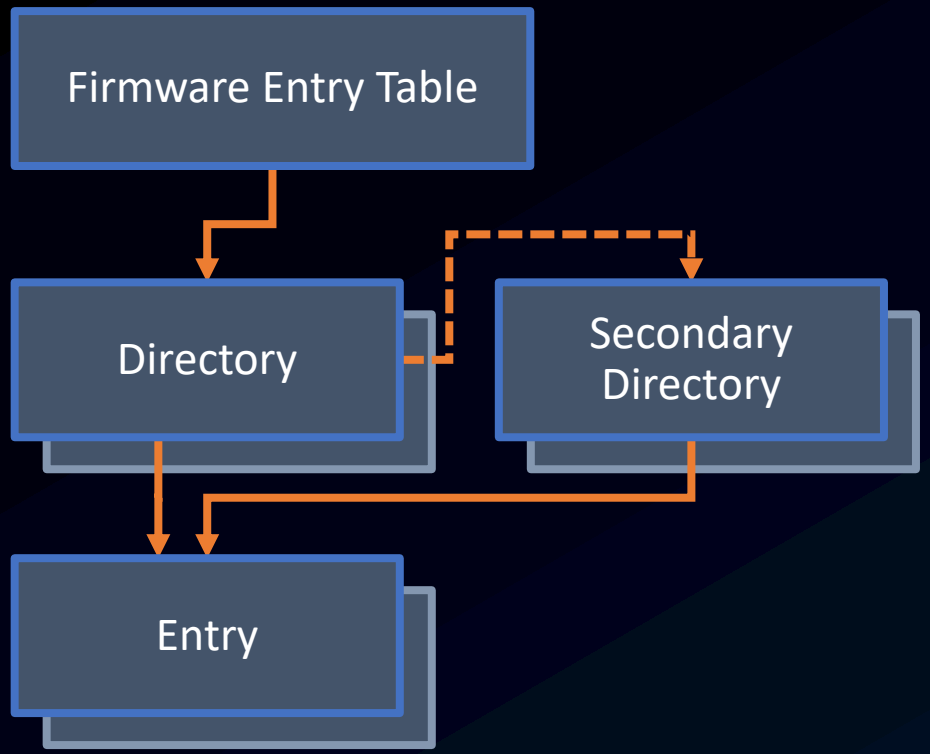
https://github.com/ridiculousfish/HexFiend

FIRMWARE FILE SYSTEM





FIRMWARE FILE SYSTEM



Firmware Entry Table

- FET begins with specific byte sequence (AA55AA55)
- Lists pointers to firmware blobs (e.g. directories) inside the UEFI image
- Earlier versions of the FET are documented in source code of the **Coreboot Project**

https://github.com/coreboot/coreboot/blob/master/util/amdfwtool/amdfwtool.c

```
$ psptool Supermicro_H11DSU9.715
```

Directory	Addr	Type	Magic	Secondary Directory
0	0x77000	PSP_NEW	$PSP	0x149000

	Entry	Address	Size	Type	Magic/ID	Version	Info
	0	0x77400	0x240	AMD_PUBLIC_KEY~0x0	1BB9		
	1	0x149400	0xe780	PSP_FW_BOOT_LOADER~0x1	$PS1	0.7.0.73	signed(1BB9), verified
	2	0x77700	0xe780	PSP_FW_RECOVERY_BOOT_LOADER~0x3	$PS1	FF.7.0.73	signed(1BB9), verified
	3	0x85f00	0x1e140	SMU_OFFCHIP_FW~0x8		4.19.7D.0	compressed, signed(1BB9), verified
	4	0xa4100	0x340	OEM_PSP_FW_PUBLIC_KEY~0xa	2793		
	5	0xa4500	0x5640	SMU_OFF_CHIP_FW_2~0x12		4.19.7D.0	compressed, signed(1BB9), verified
	6	0xa9c00	0x10	WRAPPED_IKEK~0x21			
	7	0xa9d00	0xc00	SEC_GASKET~0x24	$PS1	13.2.0.9	compressed, signed(1BB9), verified
	8	0xaa900	0xc20	ABL0~0x30	0BAR	18.11.12.11	compressed, signed(2793), verified
	9	0xab600	0xc020	ABL1~0x31	AR1B	18.11.12.11	compressed, signed(2793), verified
	10	0xb7700	0xb8f0	ABL2~0x32	AR2B	18.11.12.11	compressed, signed(2793), verified
	11	0xc3000	0xde70	ABL3~0x33	AR3B	18.11.12.11	compressed, signed(2793), verified
	12	0xd0f00	0xf1a0	ABL4~0x34	AR4B	18.11.12.11	compressed, signed(2793), verified
	13	0xe0100	0xf0a0	ABL5~0x35	AR5B	18.11.12.11	compressed, signed(2793), verified
	14	0xef200	0xc040	ABL6~0x36	AR6B	18.11.12.11	compressed, signed(2793), verified
	15	0x149000	0x0	!PL2_SECONDARY_DIRECTORY~0x40			

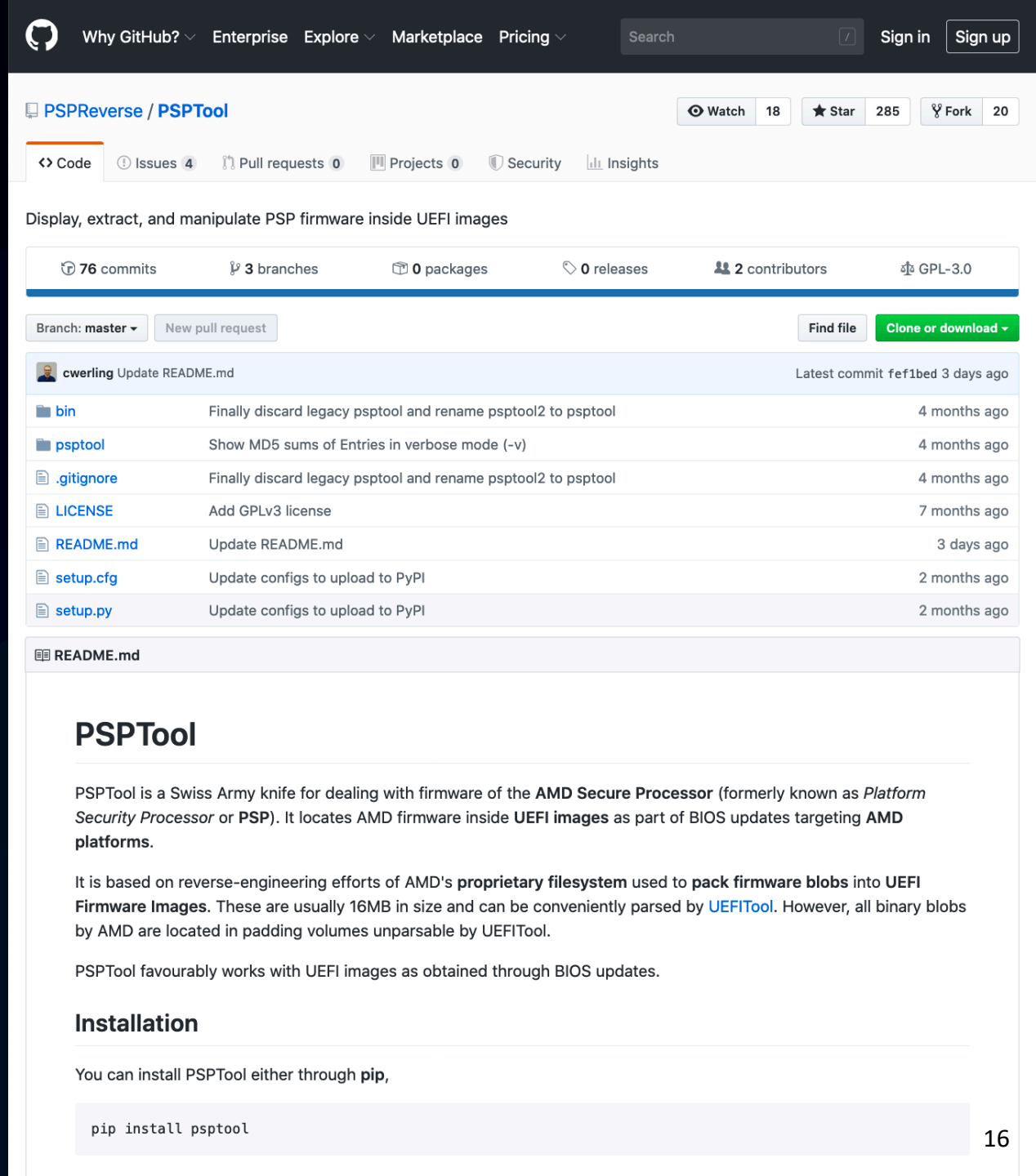
Directory	Addr	Type	Magic	Secondary Directory
1	0x149000	secondary	$PL2	--

	Entry	Address	Size	Type	Magic/ID	Version	Info
	0	0x149400	0xe780	PSP_FW_BOOT_LOADER~0x1	$PS1	0.7.0.73	signed(1BB9), verified
	1	0x159400	0x1e140	SMU_OFFCHIP_FW~0x8		4.19.7D.0	compressed, signed(1BB9), verified

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



Why GitHub? Enterprise Explore Marketplace Pricing Search Sign in Sign up

PSPReverse / PSPTool

Watch 18 Star 285 Fork 20

Code Issues 4 Pull requests 0 Projects 0 Security Insights

Display, extract, and manipulate PSP firmware inside UEFI images

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

Branch: master New pull request Find file Clone or download

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.

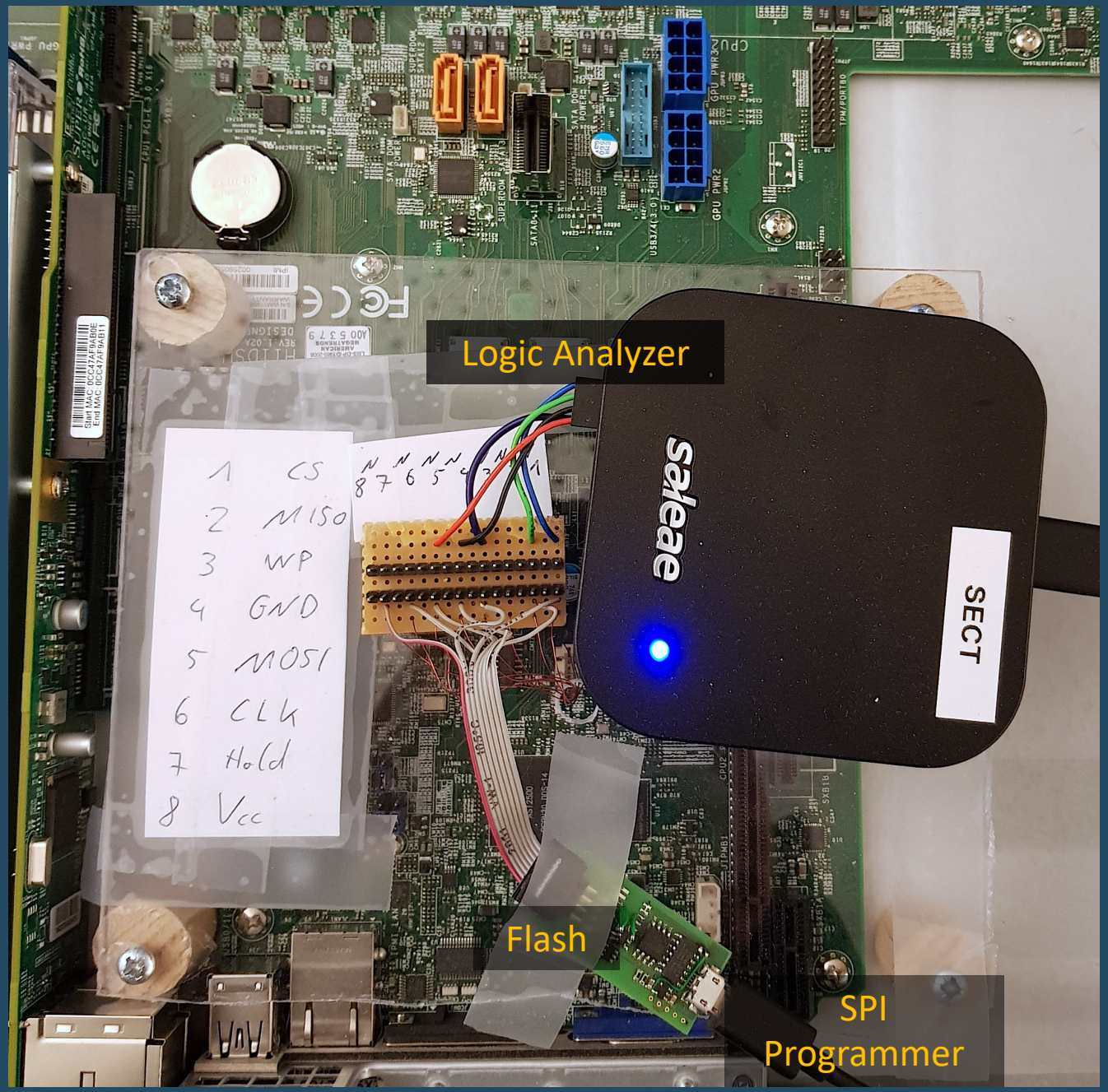
Installation

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

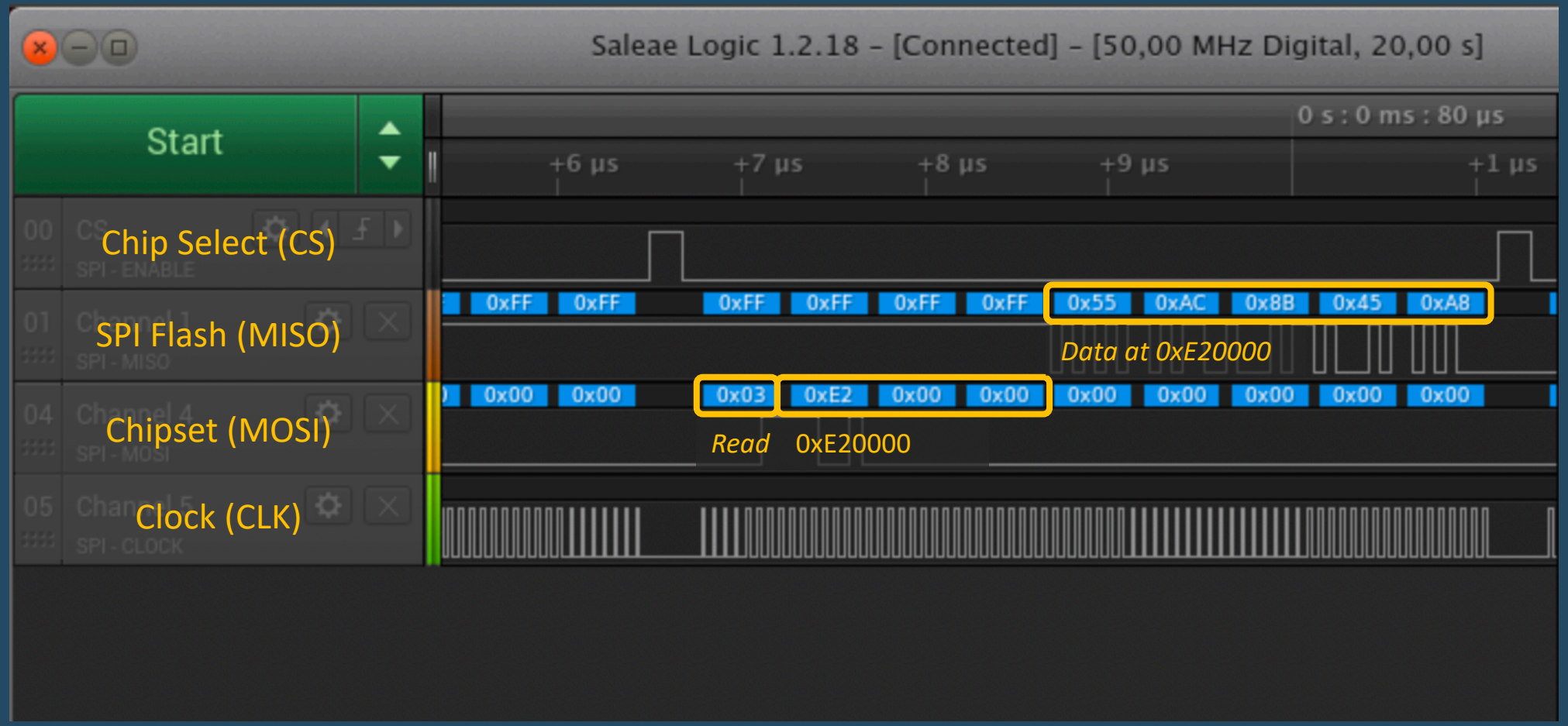
```
pip install psptool
```

The PSP runs code you
***don't know** and **don't control**.*

SPI Programming and Tracing



SPI Programming and Tracing



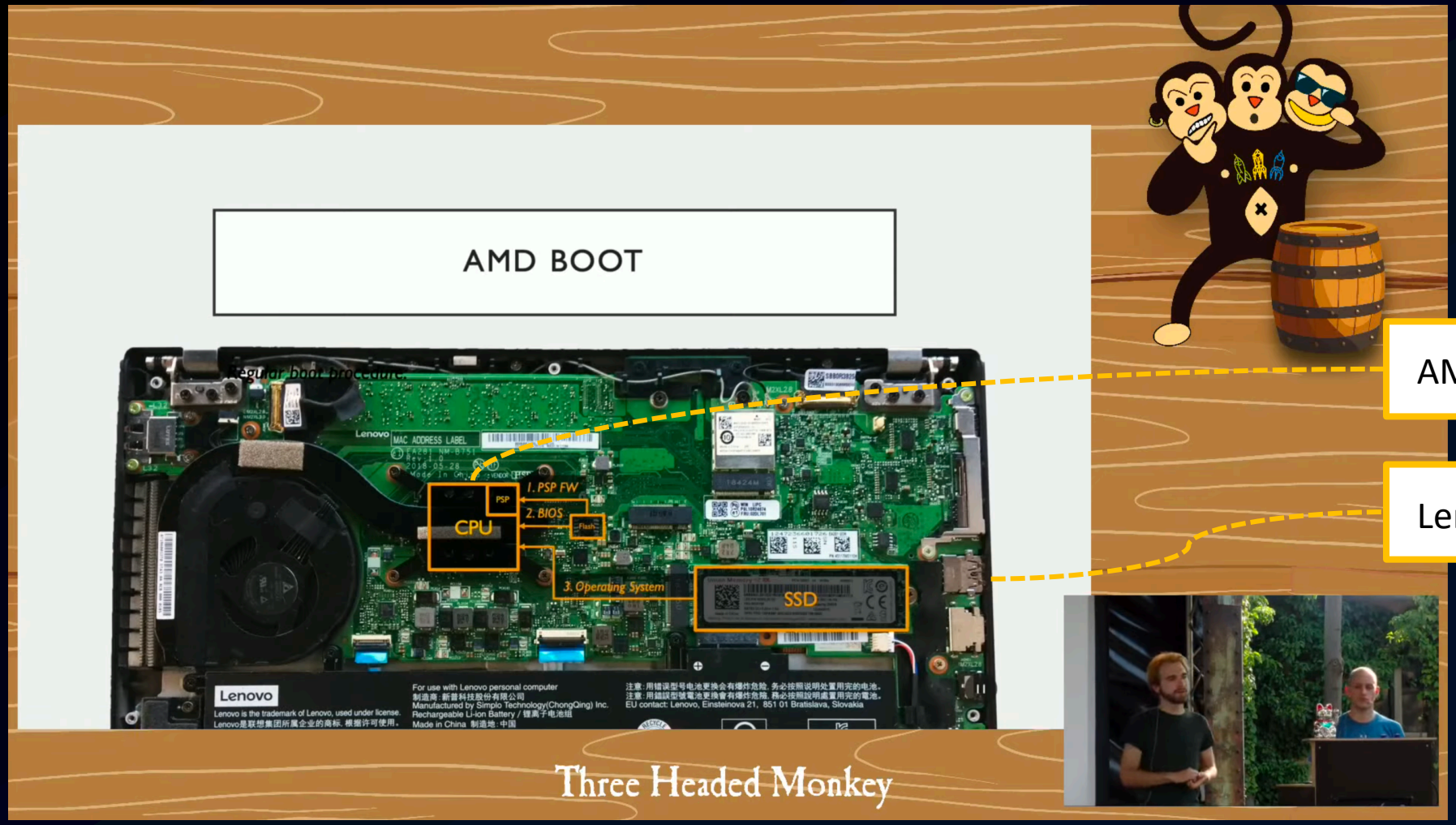
PSPTRACE

- Python-based
- SPI command parsing
- Correlate file system information
- Aggregate duplicate reads
- GPLv3
- Aggregate consecutive reads

```
$ psptrace -o Supermicro_SPI_trace.txt Supermicro_H11DSU9.715

+---------+--------------+----------+-----------------------------+
|   No.   | Lowest access |  Range   |             Type            |
+---------+--------------+----------+-----------------------------+
|    0    |    0xE20000   | 0x000040 |     Firmware Entry Table    |
|   41    |    0x077000   | 0x00012a |         PSP_DIRECTORY       |
|  112    |    0x077400   | 0x000240 |        AMD_PUBLIC_KEY       |
|  181    |    0x149400   | 0x00d780 |      PSP_FW_BOOT_LOADER     |
|         |               |          |                             |
|         |               |          |      ~ 3415 µs delay ~      |
|         |               |          |                             |
|  7083   |    0x149000   | 0x000180 |   PL2_SECONDARY_DIRECTORY   |
|         |               |          |                             |
|         |               |          |       ~ 67 µs delay ~       |
|         |               |          |                             |
|  7094   |    0x117000   | 0x000160 |         BHD_DIRECTORY       |

[...]
```

https://github.com/PSPReverse/PSPTool

More details on our hardware setups: Watch our talk from **CCCamp19**



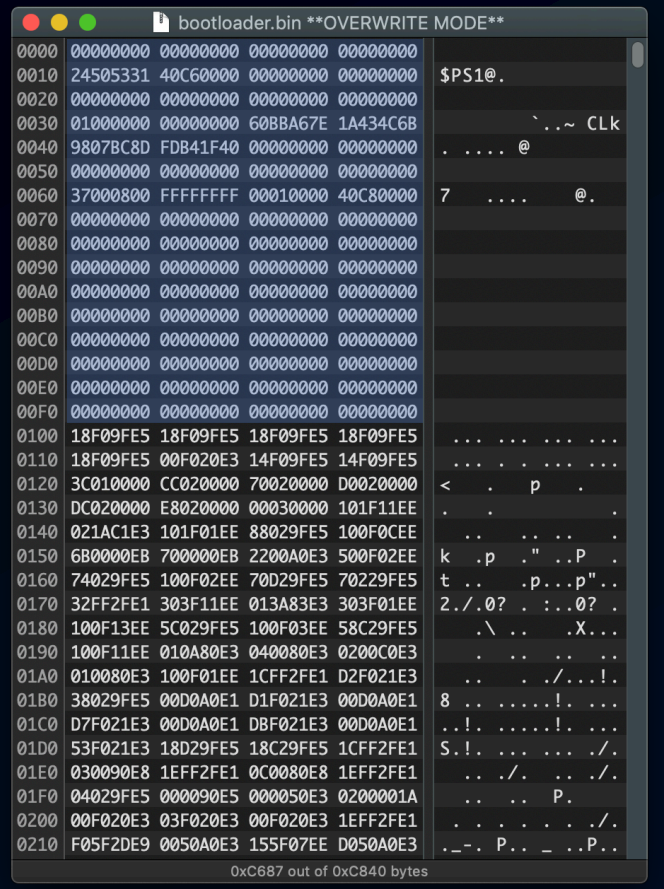
AMD Ryzen 5 Pro 2500U

Lenovo Thinkpad A285

https://media.ccc.de/v/thms-38-dissecting-the-amd-platform-security-processor

Cryptographic protections on files

- Files are **protected** by a **signature**
- Header field determines the according PublicKey[1]
- **AMD Root Public Key** for signature checking is loaded from Flash, but protected by **hash in ROM**



[1] https://developer.amd.com/wp-content/resources/55766.PDF

Early PSP Boot Procedure

On-Chip Bootloader → **Off**-Chip Bootloader (PSP_FW_BOOT_LOADER)

```
$ psptrace -o Supermicro_SPI_trace.txt Supermicro_H11DSU9.715
```

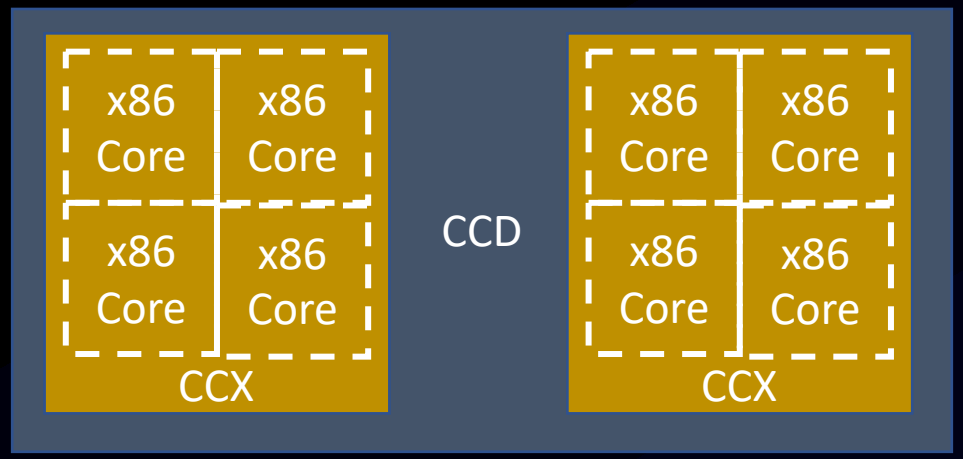
No.	Lowest access	Range	Type
0	0xe20000	0x180007	Firmware Entry Table
41	0x077000	0x00012a	PSP_DIRECTORY
112	0x077400	0x000240	AMD_PUBLIC_KEY
181	0x149400	0x00d780	**PSP_FW_BOOT_LOADER**
			~ 3415 µs delay ~
7083	0x149000	0x000180	PL2_SECONDARY_DIRECTORY
			~ 67 µs delay ~
7094	0x117000	0x000160	BHD_DIRECTORY

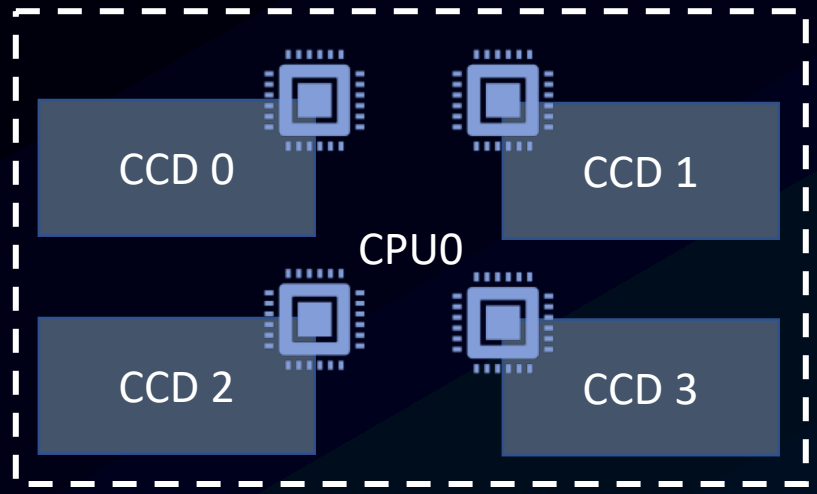
1. Load PSP_DIRECTORY
2. Load AMD_PUBLIC_KEY
3. Verify AMD_PUBLIC_KEY
4. Load PSP_FW_BOOT_LOADER
5. Verify with AMD_PUBLIC_KEY

1. Initialize PSP
2. Load more directories
3. Load and verify applications

Understand

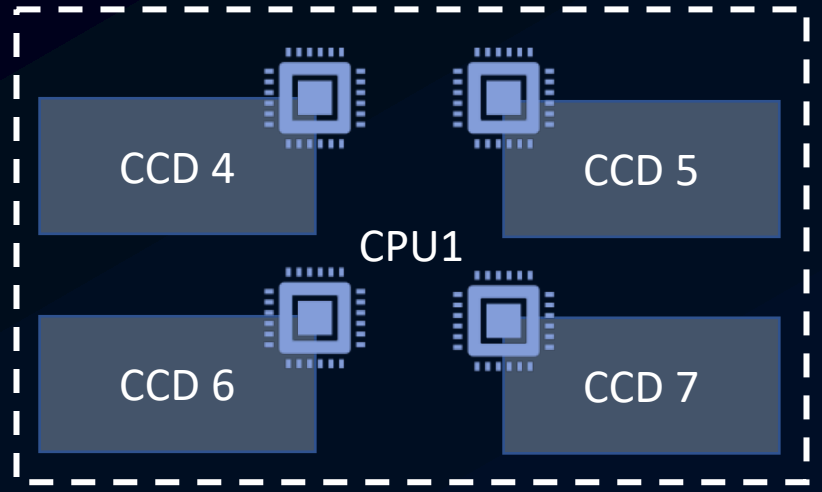
HOW DEEP DOES THE RABBIT HOLE GO?





ONE PSP TO RULE THEM ALL …

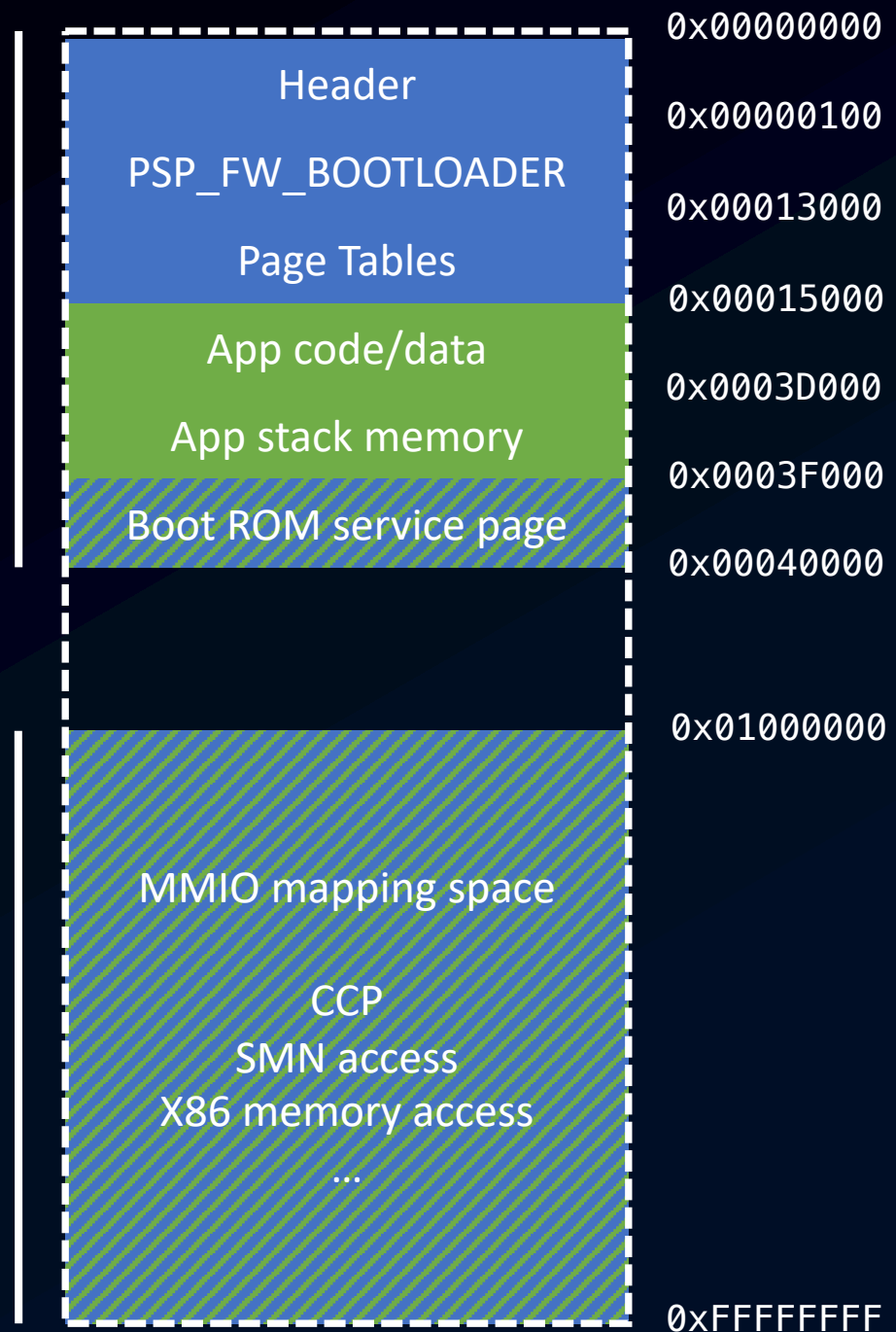
- CCX (Core CompleX): Up to 4 x86 cores (8 threads)
- CCD (Core Complex Die): 2 CCX, Memory controller, etc.
- One PSP per CCD (Naples)
- PSP on CCD 0 is the Master
- Master coordinates initial bringup of platform

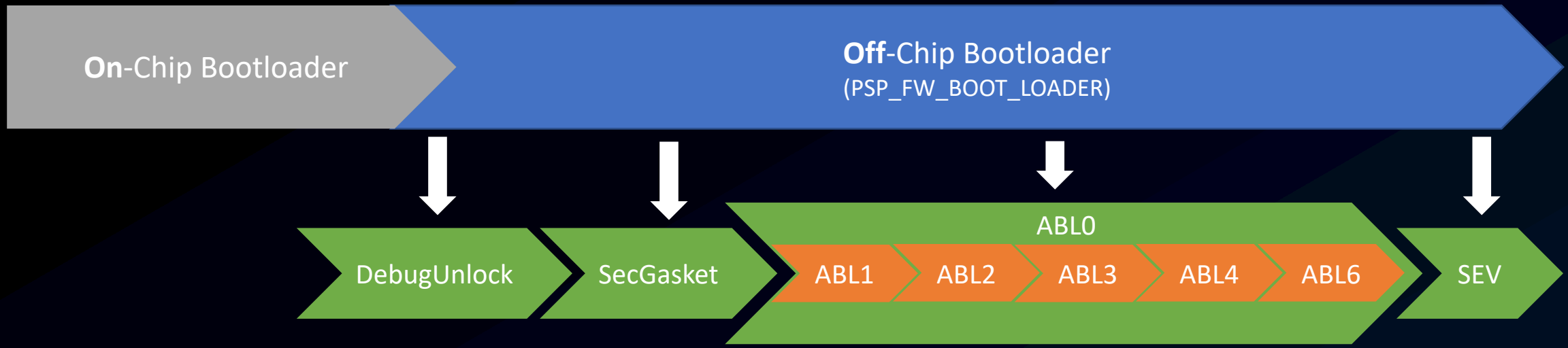


On-Chip Bootloader → **Off**-Chip Bootloader (PSP_FW_BOOT_LOADER) → Applications

MEMORY LAYOUT

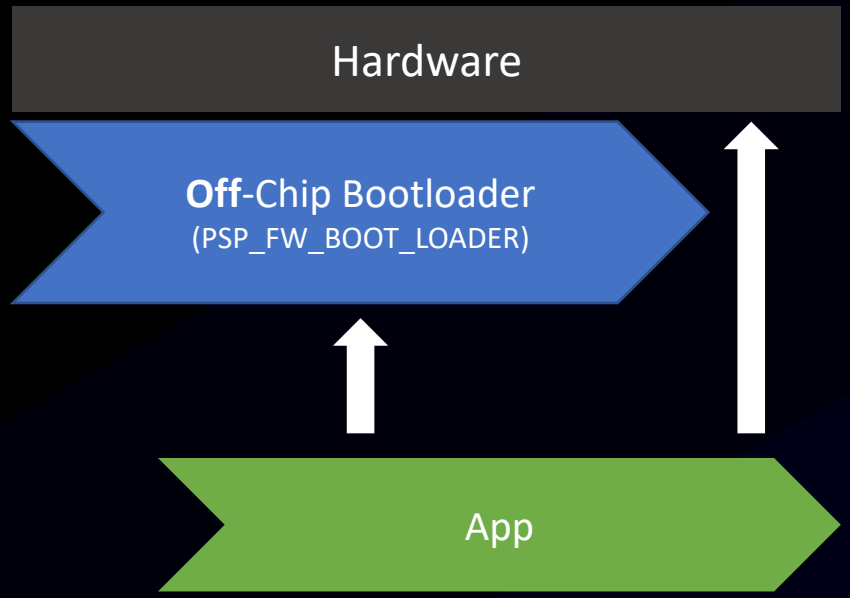
- 256KB on chip SRAM
- Code separated in SVC and USR mode parts
- USR mode parts loaded during boot and later on demand (SEV)





BOOT PROCESS

- **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**
- **SEV app** is loaded during runtime upon the **request of the OS**



THE SYSCALL INTERFACE

76 Syscalls

30 mostly reverse engineered:

- Access SMN
- Access DRAM
- Communicate with PSPs
- Query SMM region
- Busy wait
- Load entries from flash
- Invalidate/Clean PSP memory ranges

28 partly reverse engineered:

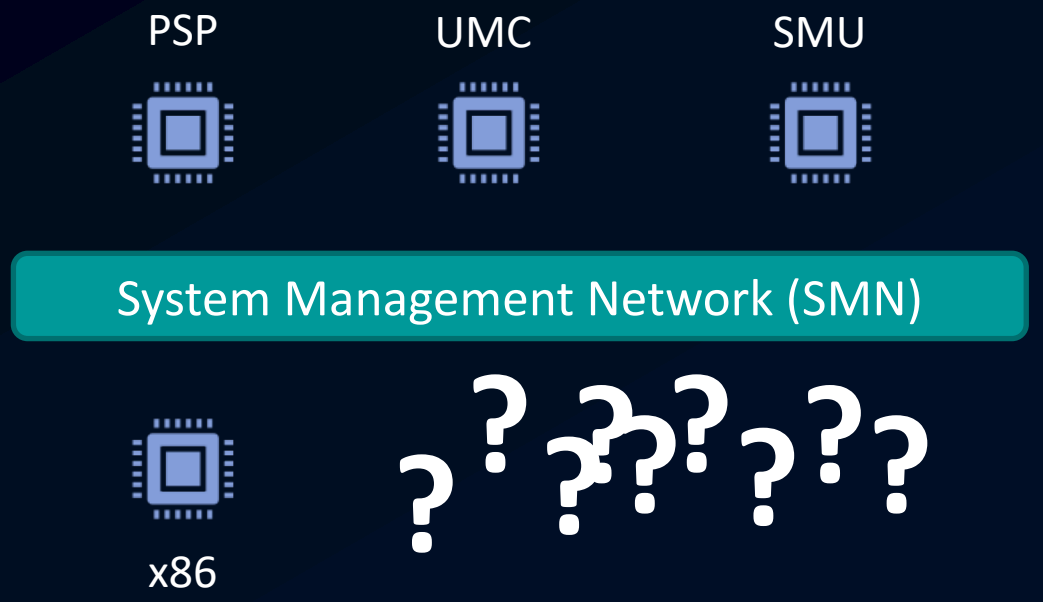
- CCP operations
- More inter-PSP communication

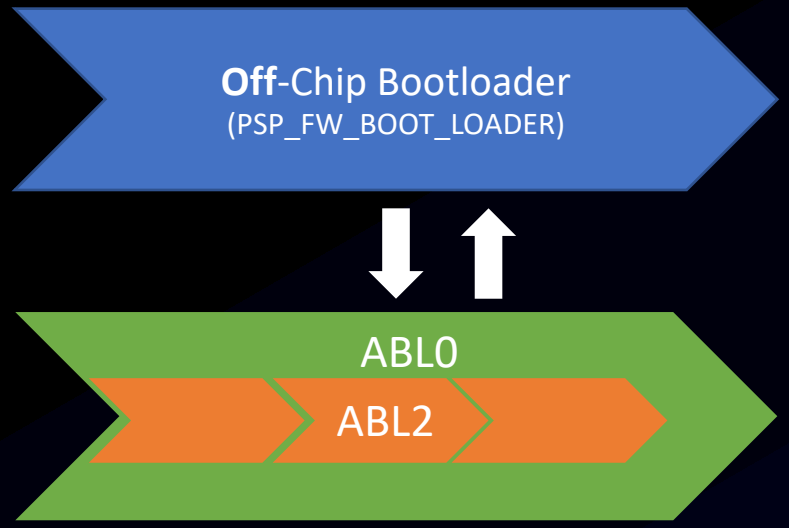
18 completely unknown

Region	Size	WP	MPsp	Offset	RegSz	Description	Register description
0x0001c880	128	+	-			Memory protection slots	
				0x00	32bit	Slot 0: Start address of protected region X86PADDR[47:20] + 4 flags	aaaaaaaaaaaaaaaaaaaaaaaaaaaa????
				0x04	32bit	Slot 0: End address (inclusive) of protected region X86PADDR[47:20] + 4 flags	aaaaaaaaaaaaaaaaaaaaaaaaaaaa????
				0x08	32bit	Slot 0: Control register (seen 0x600000a | 0x6000006)	??????????????????????????????e
				0x0c	32bit	Slot 0: Unused/Reserved (no access observed anywhere)	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
				...	...	Slot 1 - 7	...

SYSTEM MANAGEMENT NETWORK (SMN)

- Hidden control network
- Dedicated address space
- PSP maps regions into own address space to access device registers

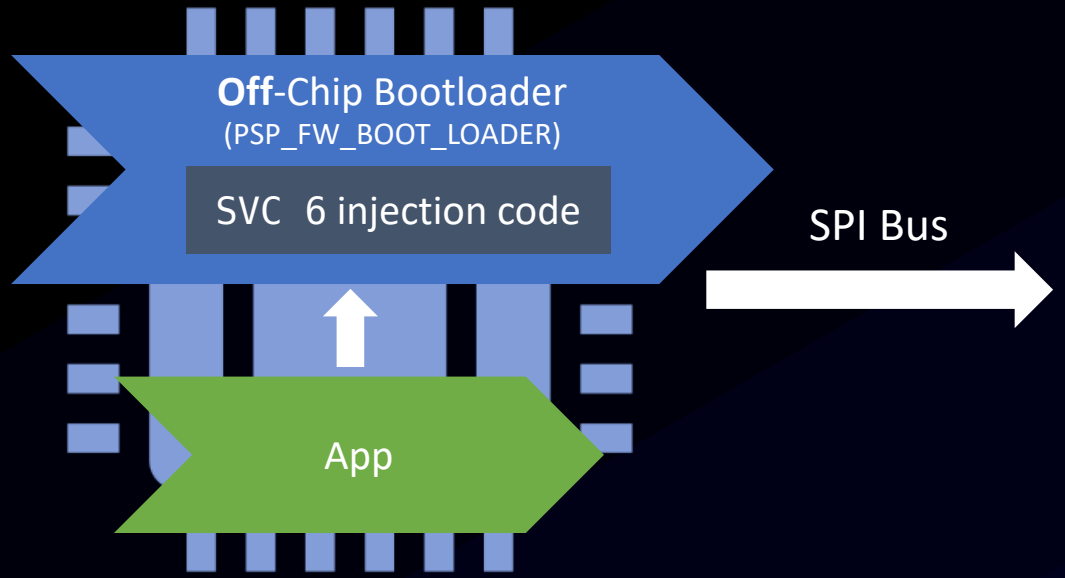




ENABLE DEBUG OUTPUT

- Lots of interesting debug strings
- SVC `0x6` uses string address as the first argument
- *Not* implemented in release firmware ☹

```
$ strings AR2B.bin

[…]

!!!ATTENTION: Simnow r30138 or later is required for the following
polling loop.

        Send following data to slaves:

                mixedWithNvdimmInSystem = %x

                mixedWithNvdimmInSocket = %04x

                mixedWithNvdimmInDie = %08x %08x

----------------------------------------------------------------------
----------

Sync Speed Disabled - Gathering Speed Data for single die only

        Master: Retrieve debug data from the slaves at debug sync
point %04x

[…]

$ arm-none-eabi-objdump -b binary --adjust-vma 0x16000 -D AR2B.bin -m
armv5 -Mforce-thumb |grep -B 5 "svc\t6"

[…]

   2191c:       a0be            add     r0, pc, #760    ; (adr r0,
0x21c18)

   2191e:       df06            svc     6
```



SPI Bus

EXFILTRATING DEBUG OUTPUT

- Problem: No x86 memory available at this time
- Only known/accessible device is SPI flash
- Dump it on the SPI bus without altering flash
- Filter the SPI trace

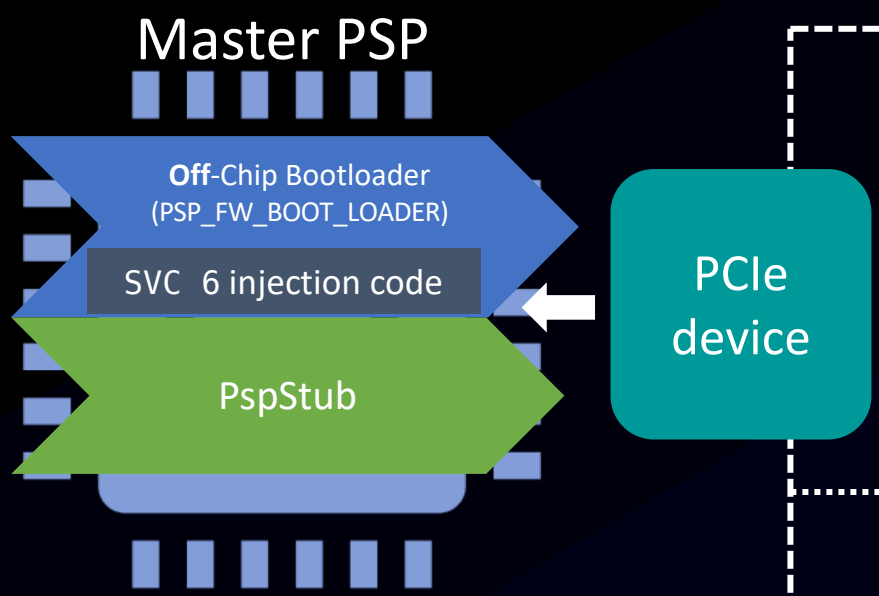
`SPI Trace.csv`

`psp-log-dump.py`

`PSP.log`

SUCCESS!

```
[…]
MEM PARAMS:
        AGESA BL Heap Size : 7800
        BottomIo : 0080
        MemHoleRemap : 1
        LimitBelow1TB : 1
        UserTimingMode : 0
        MemClockValue : 1200
        MemRestoreCtl : 0
        SaveMemContextCtl : 1
        ExternalVrefCtl : 0
        ForceTrainMode : 2
        AMP : 0

         0x00800F12 (32b)
         0x00006031 (32b)
         0x00800F12 (32b)
         0x00006031 (32b)
ZP DDR4 DRAM Initialization - Phase 2

Mem Phase 2 Start
Start PState Sync

DDR Phy Initialization
Start DDR Training using PMU

Begin PMU Based DRAM Init and Training
PspBootRomServices:SystemSocketCount: 2
PspBootRomServices:SystemDieCount: 8
PspBootRomServices:DiesPerSocket DieNum: 4
PspBootRomServices:SocketId: 0
PspBootRomServices:PhysDieId: 0

                No 'UMCF' singature at FCH BiosRam offset 0
                Sending Agesa memory test UMC MCA failure result to slave

[…]
```



PCIe device

X86 Core

Ring 0

ccp.ko/psp-sev.c

Ring 3

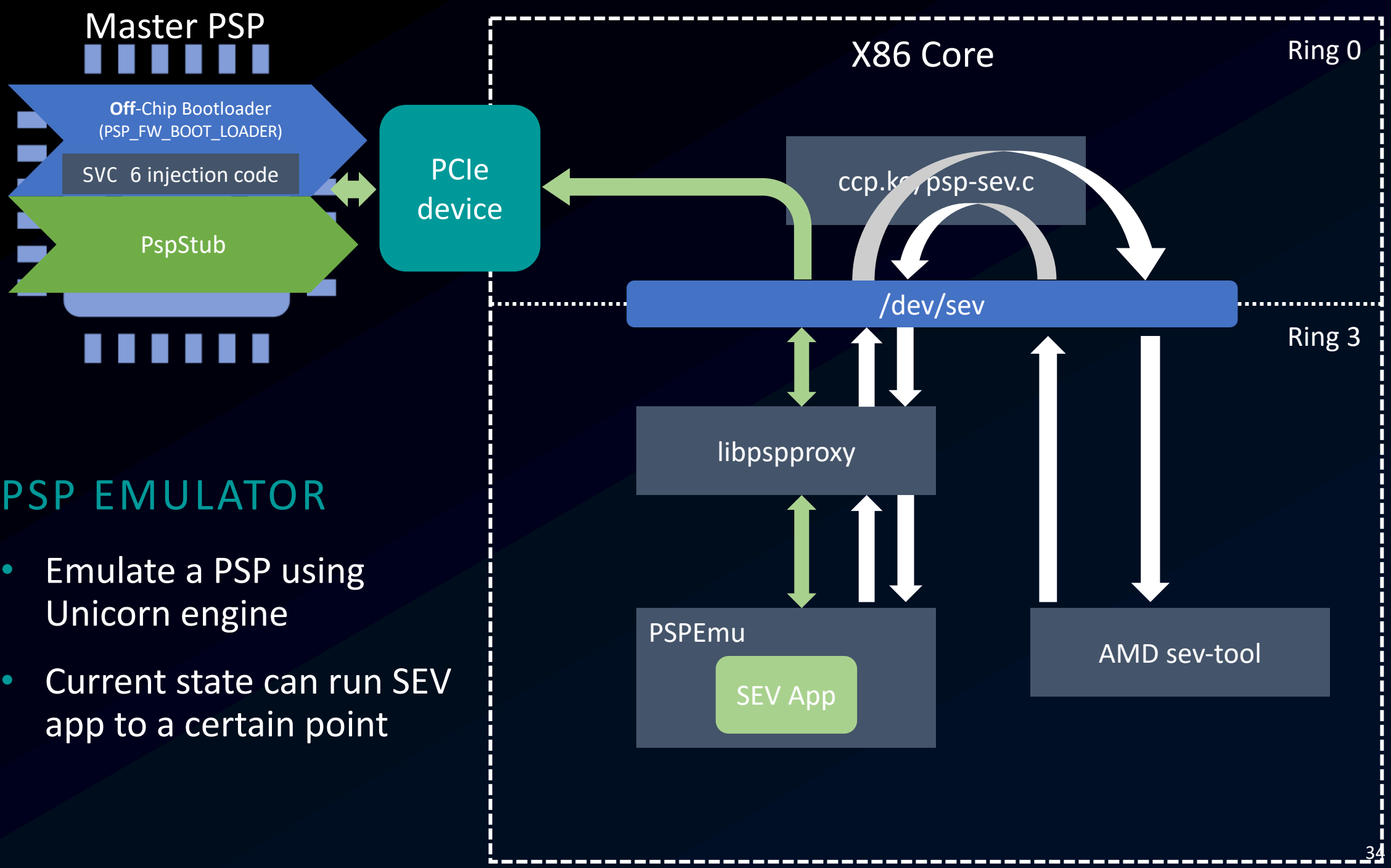
pypsppproxy

libpsppproxy

EXPLORING THE SMN DEVICES

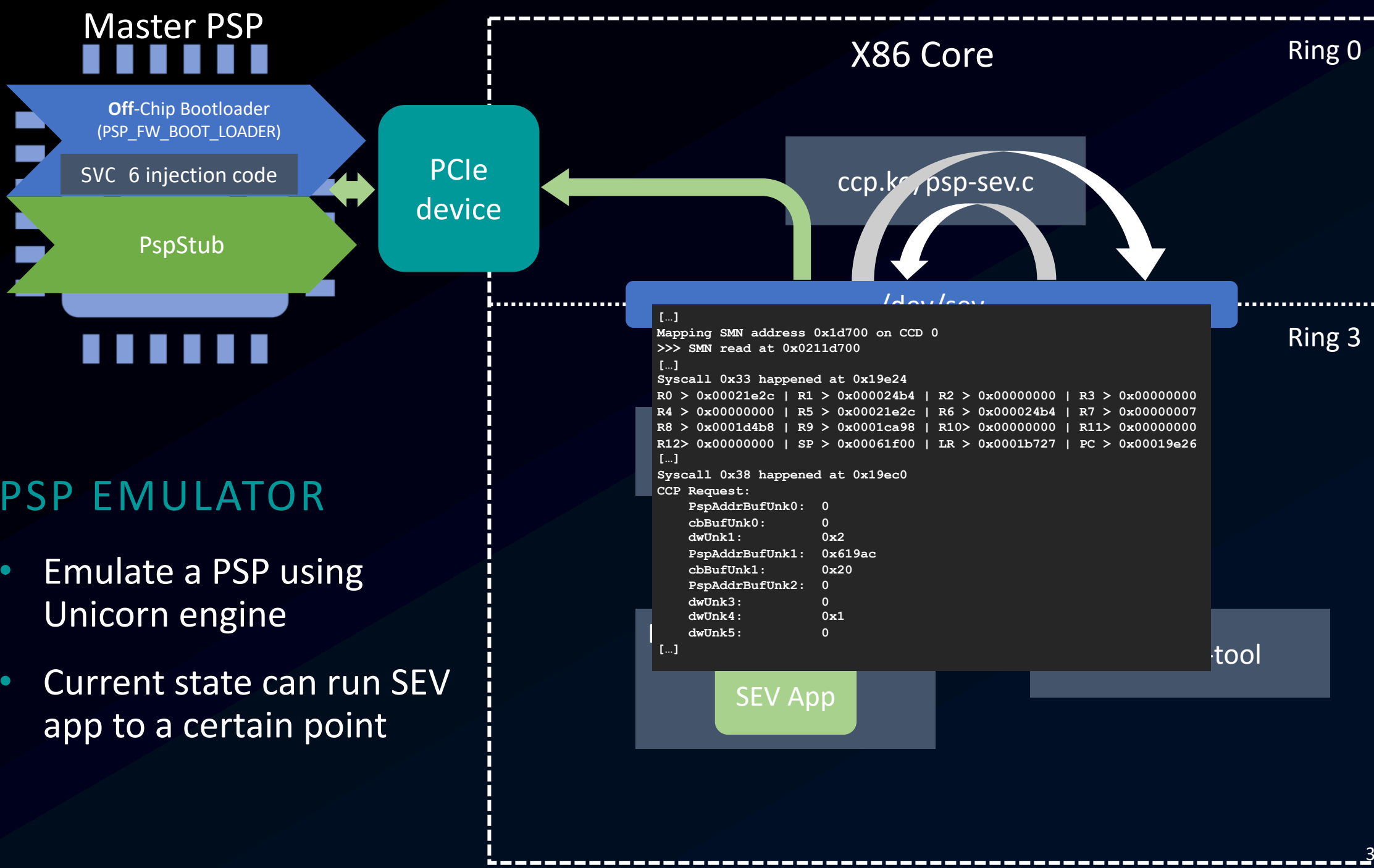
- Replace SEV app with a stub
- Executes requests on a target PSP:
  - Read/Write SMN address
  - Execute syscall
  - Read/Write PSP memory

```
import pypsppproxy;
[…]
proxy = pypsppproxy.PSPProxy("/dev/sev");
if proxy.getLastRc() == 0:
    rc, virtAddr, physAddr = proxy.allocX86Mem(2 * 1024 * 1024);
    if rc == 0:
[…]
        for idCcd in xrange(8):
            proxy.setCcd(idCcd);
            _, uR0 = proxy.callSvc(0x28, 0x14, 0x1, 0x0, 0x0);
            proxy.writeSmn(idCcd, 0x1c890, 4, (physAddr >> 20) |
5);
            proxy.writeSmn(idCcd, 0x1c894, 4, physAddr >> 20);
            proxy.writeSmn(idCcd, 0x1c898, 4, 0x600000a);
            _, uR0 = proxy.callSvc(0x28, 0x14, 0x0, 0x0, 0x0);
```



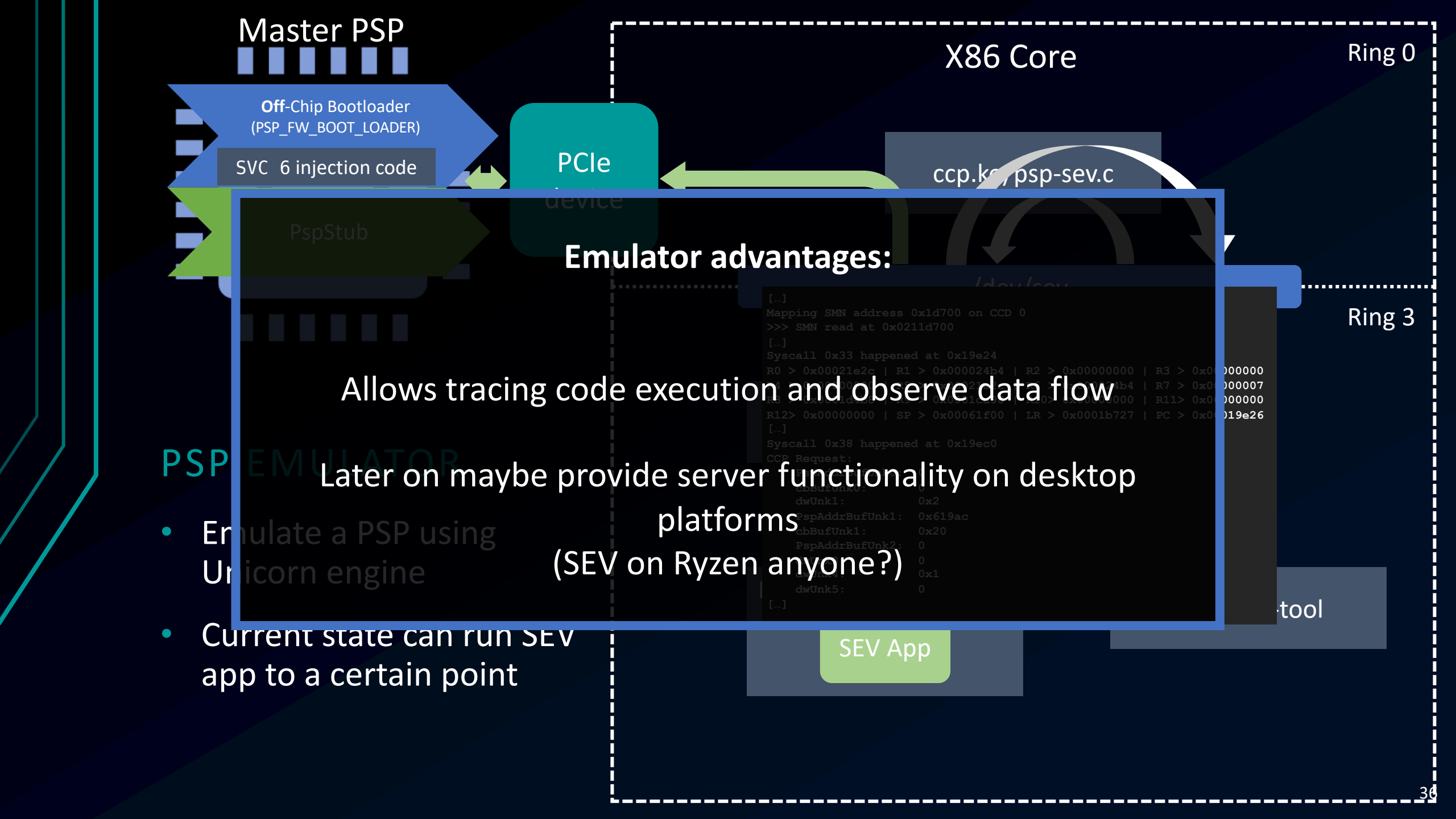
PSP EMULATOR

- Emulate a PSP using Unicorn engine
- Current state can run SEV app to a certain point



PSP EMULATOR

- Emulate a PSP using Unicorn engine
- Current state can run SEV app to a certain point


Master PSP
Off-Chip Bootloader
(PSP_FW_BOOT_LOADER)
SVC 6 injection code
PspStub
PCIe
X86 Core
Ring 0
ccp.ko / psp-sev.c
Ring 3
SEV App
Emulator advantages:
Allows tracing code execution and observe data flow
Later on maybe provide server functionality on desktop platforms
(SEV on Ryzen anyone?)
PSP EMULATOR
Emulate a PSP using Unicorn engine
Current state can run SEV app to a certain point

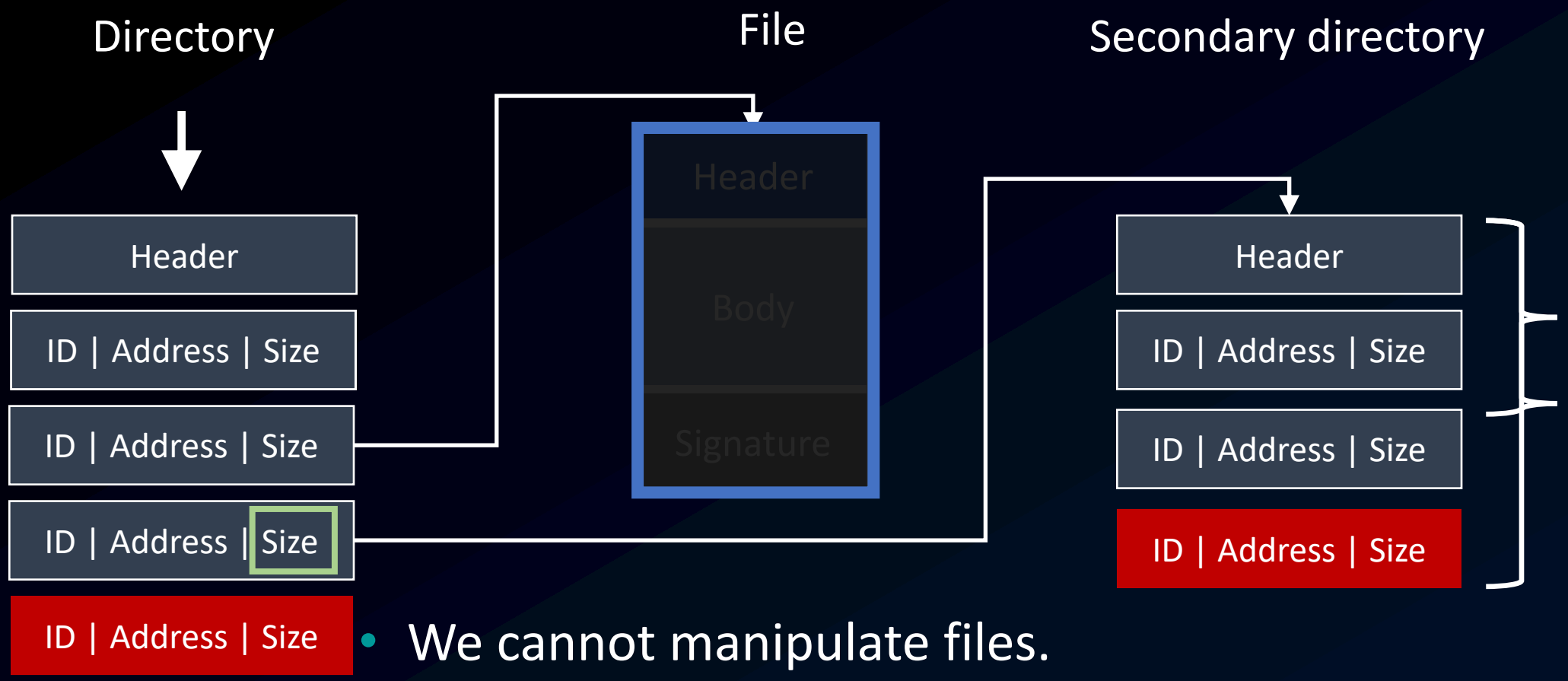
INTERESTED? HERE IS THE CODE

- Code will be available on https://github.com/PSPReverse
- Repositories
  - PSPTool — Display, extract, and manipulate firmware images
  - psp-docs — Documentation about hardware interfaces, syscalls
  - psp-includes — Shared interface headers
  - psp-apps — Build your own apps running on the PSP
  - linux — Linux kernel with our modifications
  - libpspproxy — Userspace PSP proxy library for the stub
  - PSPEmu — Unicorn-based PSP emulator
  - sev-tool — AMDs sev-tool with our modifications

Own

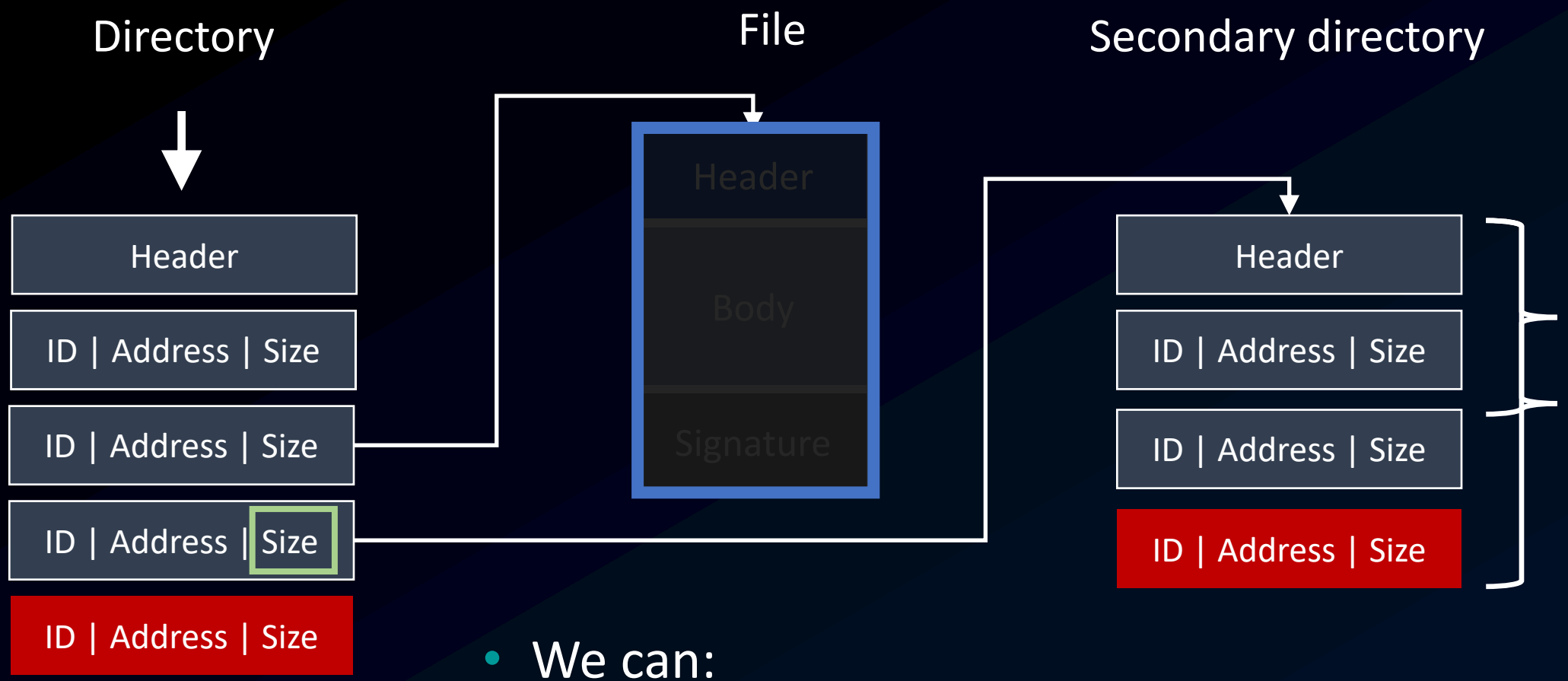
PART 1:
BOUNDS CHECKING IS HARD

Attacker Capabilities

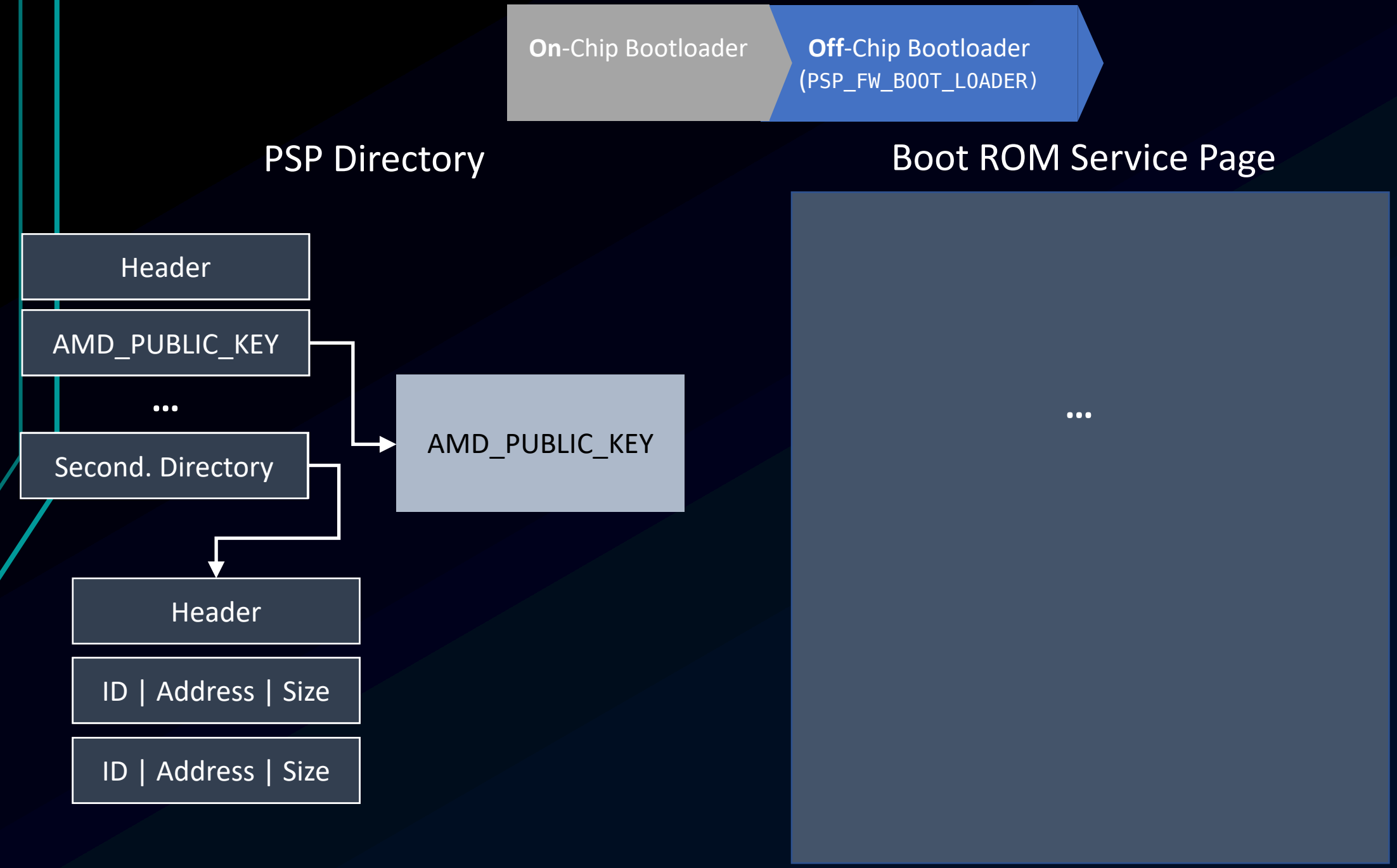


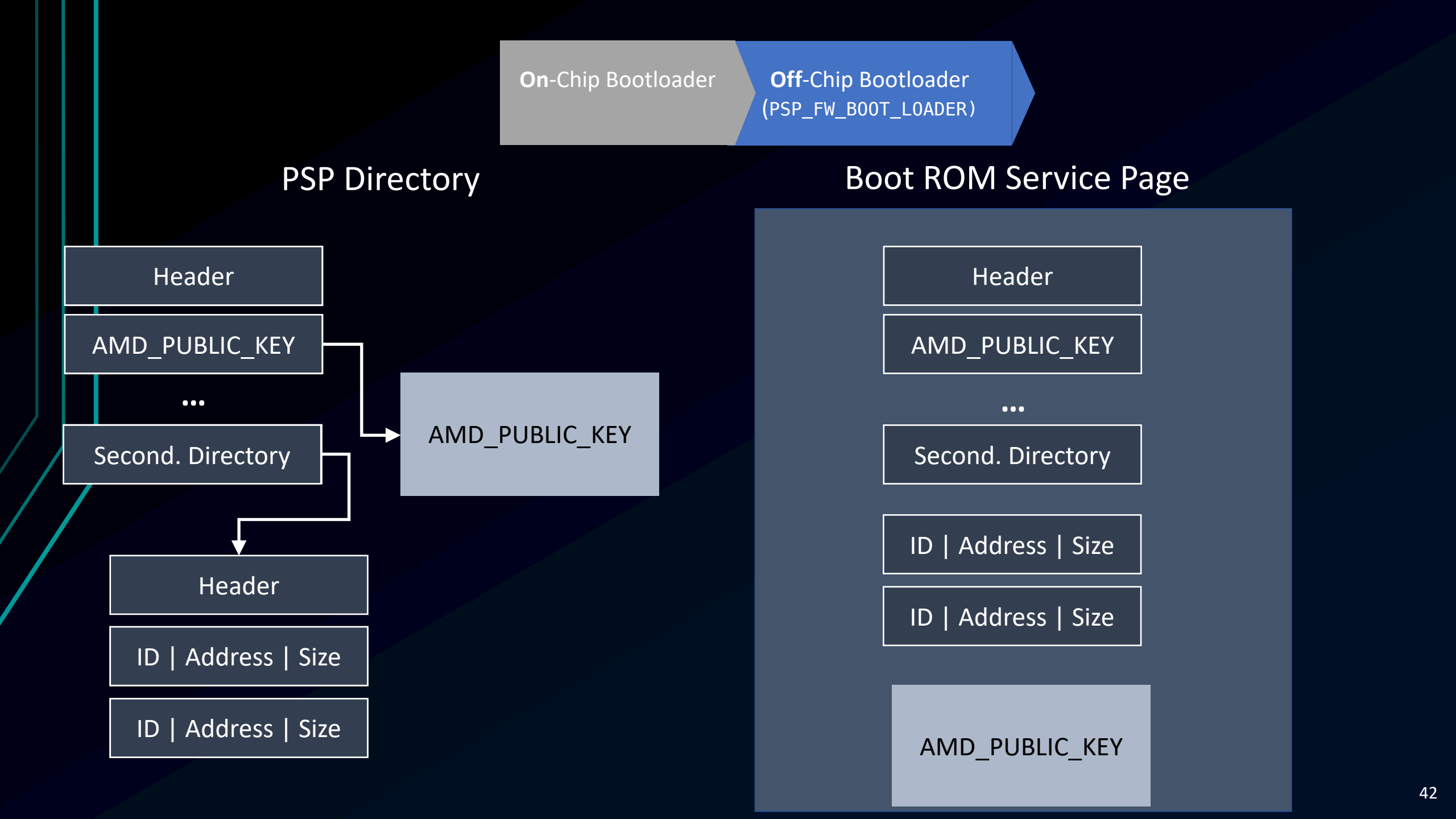
- We cannot manipulate files.
- We ***can*** manipulate the directories!

Attacker Capabilities

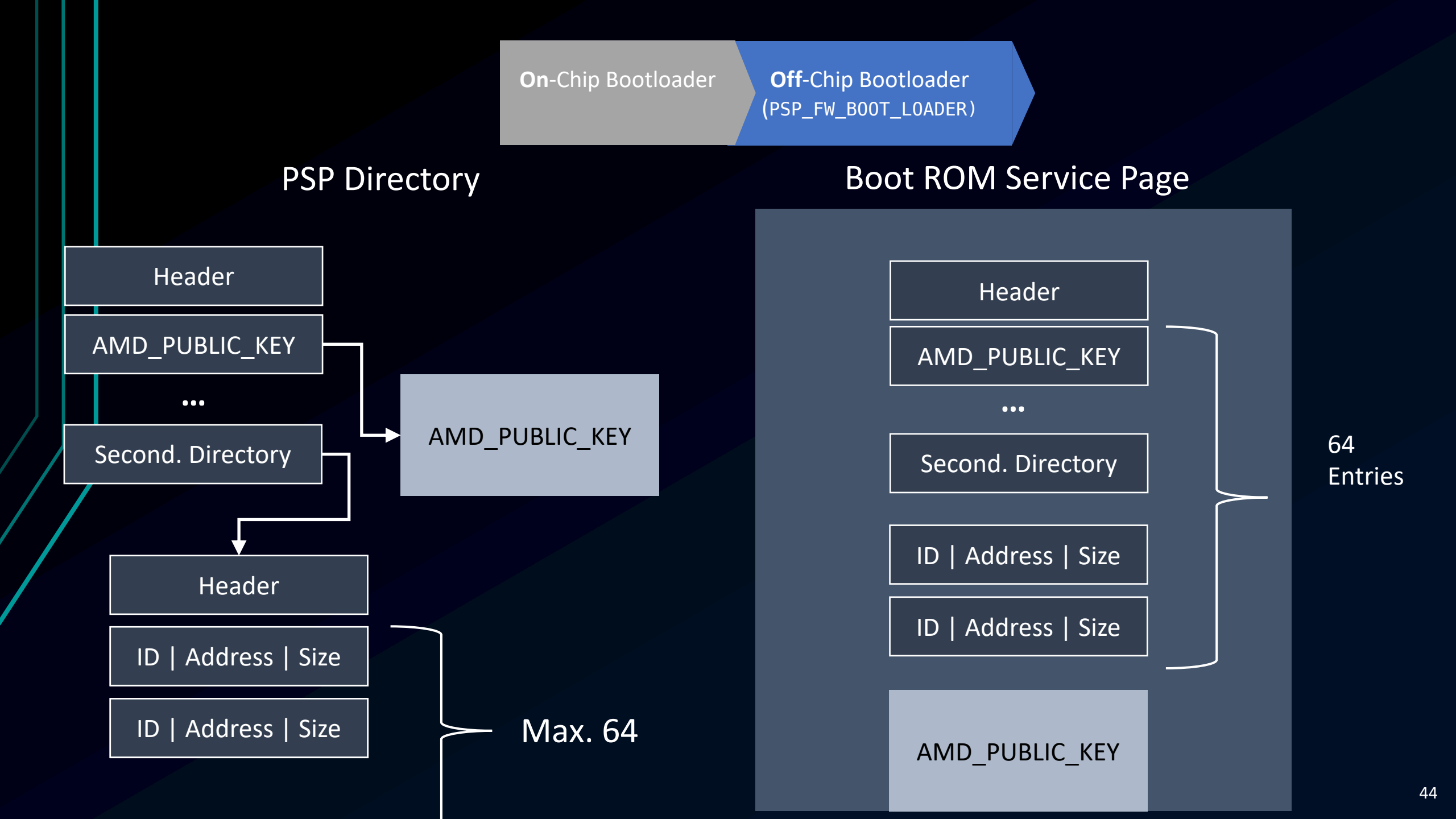


- We can:
  - Add Entries
  - Remove Entries
  - Change Entries


On-Chip Bootloader
Off-Chip Bootloader
(PSP_FW_BOOT_LOADER)
PSP Directory
Boot ROM Service Page
Header
AMD_PUBLIC_KEY
...
Second. Directory
AMD_PUBLIC_KEY
Header
ID | Address | Size
ID | Address | Size
...


On-Chip Bootloader
Off-Chip Bootloader
(PSP_FW_BOOT_LOADER)
PSP Directory
Boot ROM Service Page
Header
AMD_PUBLIC_KEY
...
Second. Directory
AMD_PUBLIC_KEY
Header
ID | Address | Size
ID | Address | Size
Header
AMD_PUBLIC_KEY
...
Second. Directory
ID | Address | Size
ID | Address | Size
AMD_PUBLIC_KEY


What could possibly
go wrong?


On-Chip Bootloader
Off-Chip Bootloader
(PSP_FW_BOOT_LOADER)
PSP Directory
Boot ROM Service Page
Header
AMD_PUBLIC_KEY
...
Second. Directory
AMD_PUBLIC_KEY
Header
ID | Address | Size
ID | Address | Size
Max. 64
Header
AMD_PUBLIC_KEY
...
Second. Directory
ID | Address | Size
ID | Address | Size
AMD_PUBLIC_KEY
64
Entries

On-Chip Bootloader → **Off**-Chip Bootloader (PSP_FW_BOOT_LOADER)

PSP Directory

Header

AMD_PUBLIC_KEY

...

Second. Directory

Header

ID | Address | Size

ID | Address | Size

PUBLIC KEY

Max. 64

```
int append_second(void) {
  ...
  if (nr_entries > 64u)
    return -1;
  ...
  return 0;
}
```

Boot ROM Service Page

Header

IC_KEY

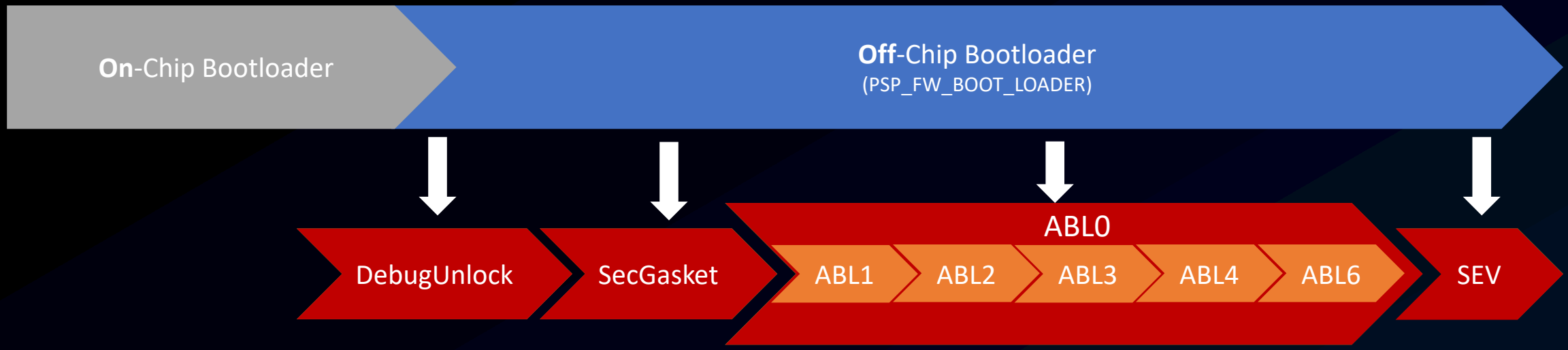
ectory

ID | Address | Size

ID | Address | Size

PUBLIC KEY

64 Entries



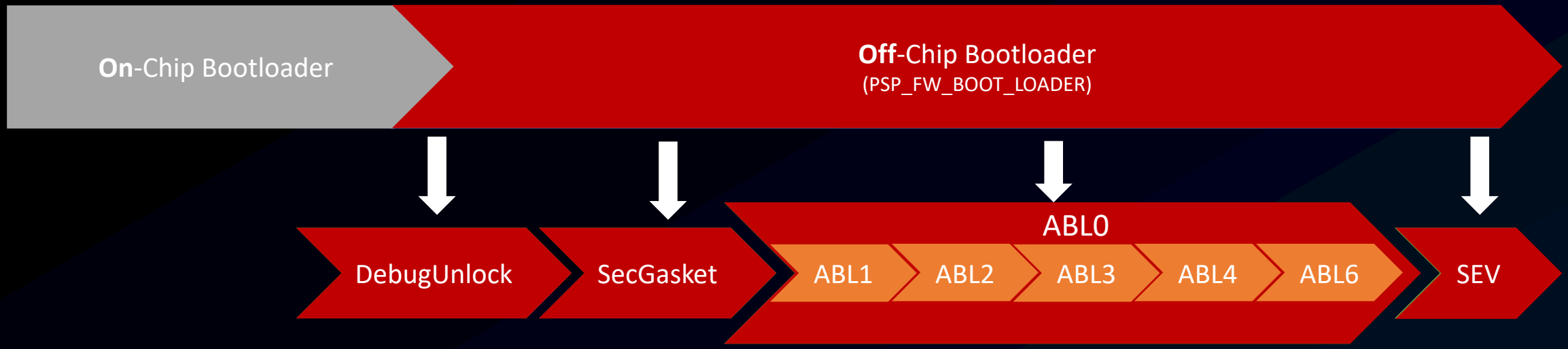
BOOT PROCESS

- Directory parsing takes place before loading any application.

-> We control the user mode beginning from the first application.

Own

PART 2:
INPUT VALIDATION IS HARD



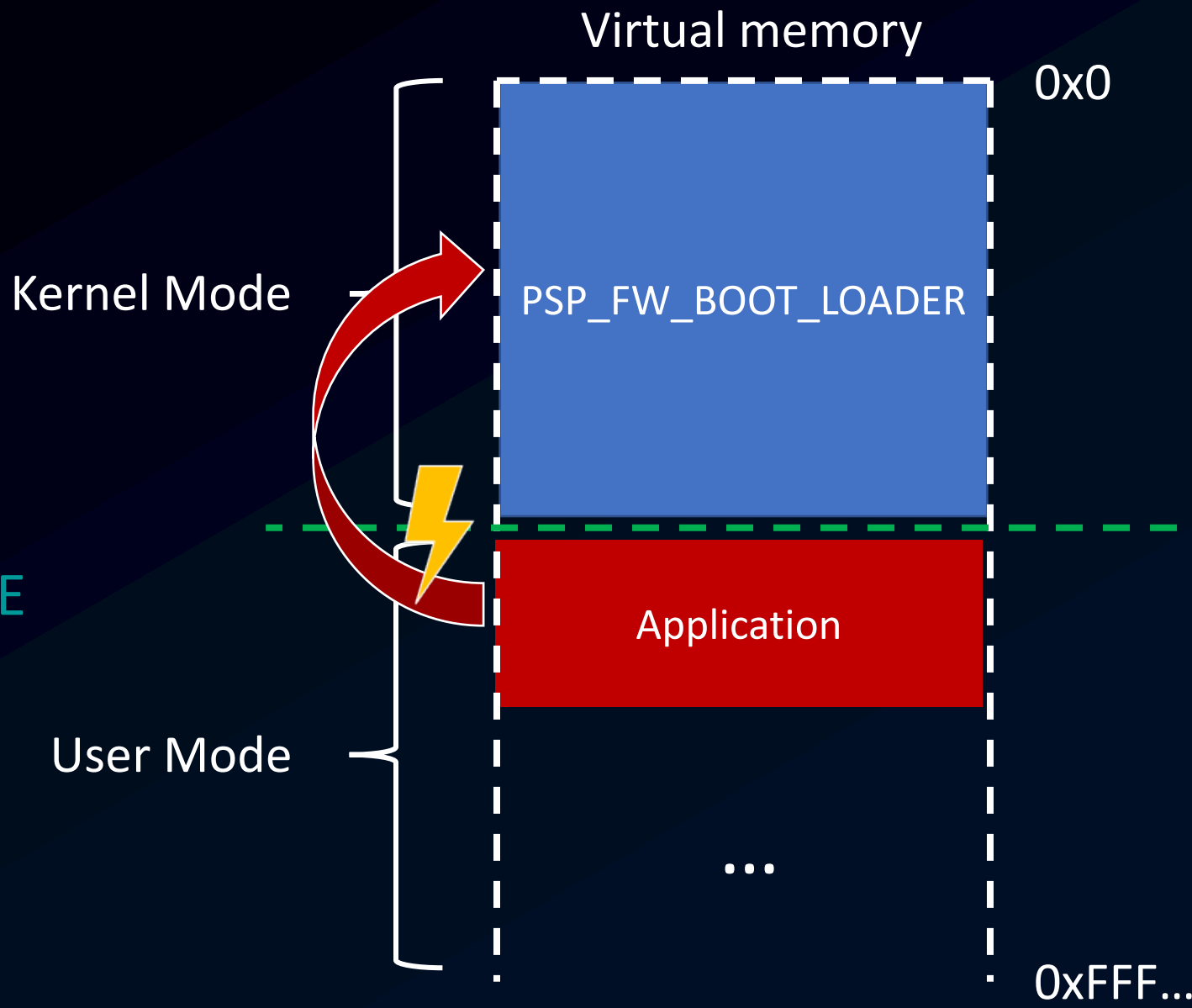
BOOT PROCESS

- Directory parsing takes place before loading any application.

We control the user mode beginning from the first application.

How can we take over the kernel mode?

Off-Chip Bootloader
(PSP_FW_BOOT_LOADER)



VIRTUAL ADDRESS SPACE

User space applications can't access kernel space memory.

The "split" is enforced by the Memory Management Unit

```
int copy_from_flash(void* dst, void* src, int size);
```

Flash

BIOS Directory

Header

ID | Address | Size

...

ID | Address | Size

Header

ID | Address | Size

ID | Address | Size

PAGE TABLES

Virtual memory

CODE

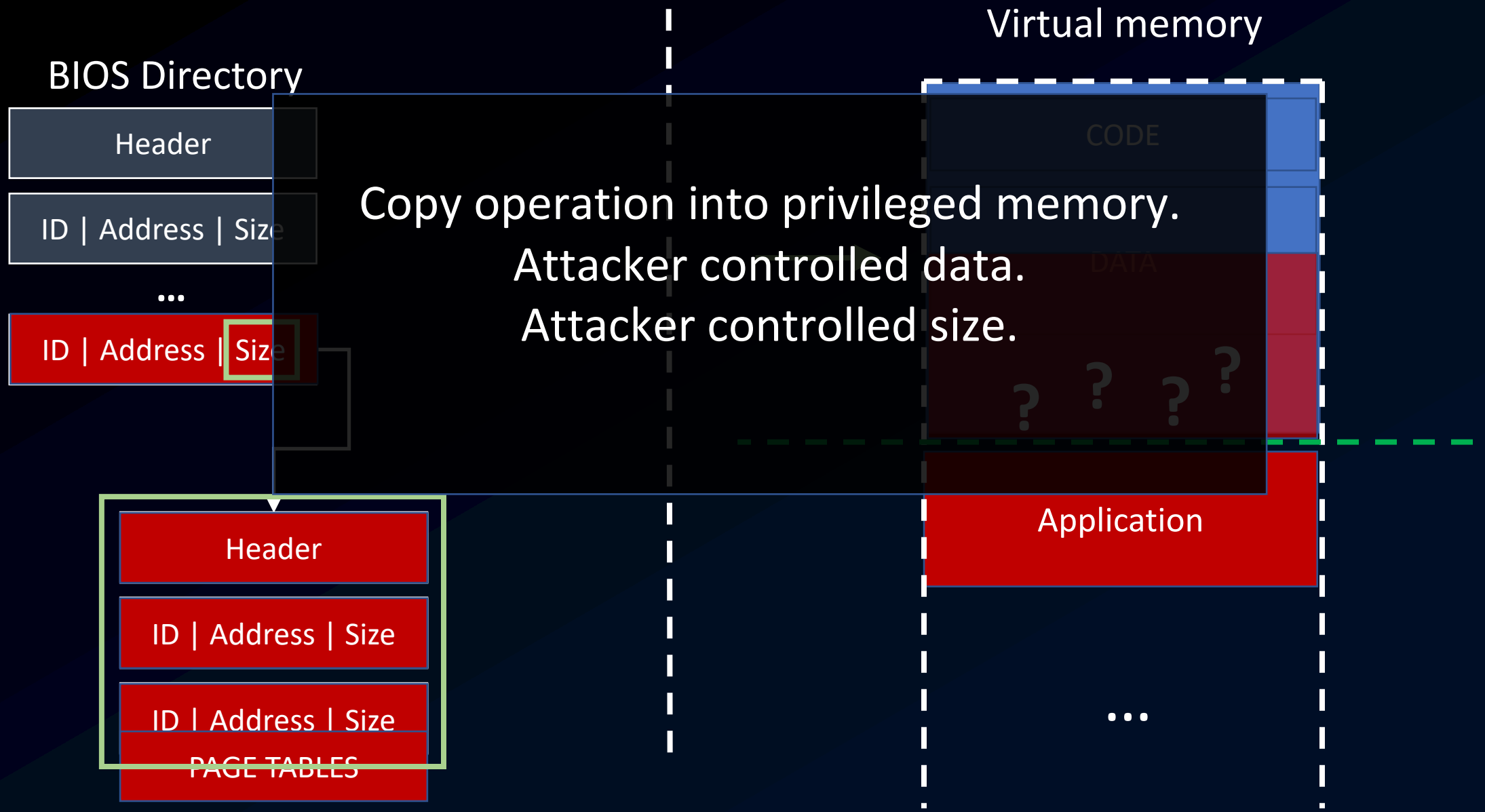
DATA

? ? ? ?

Application

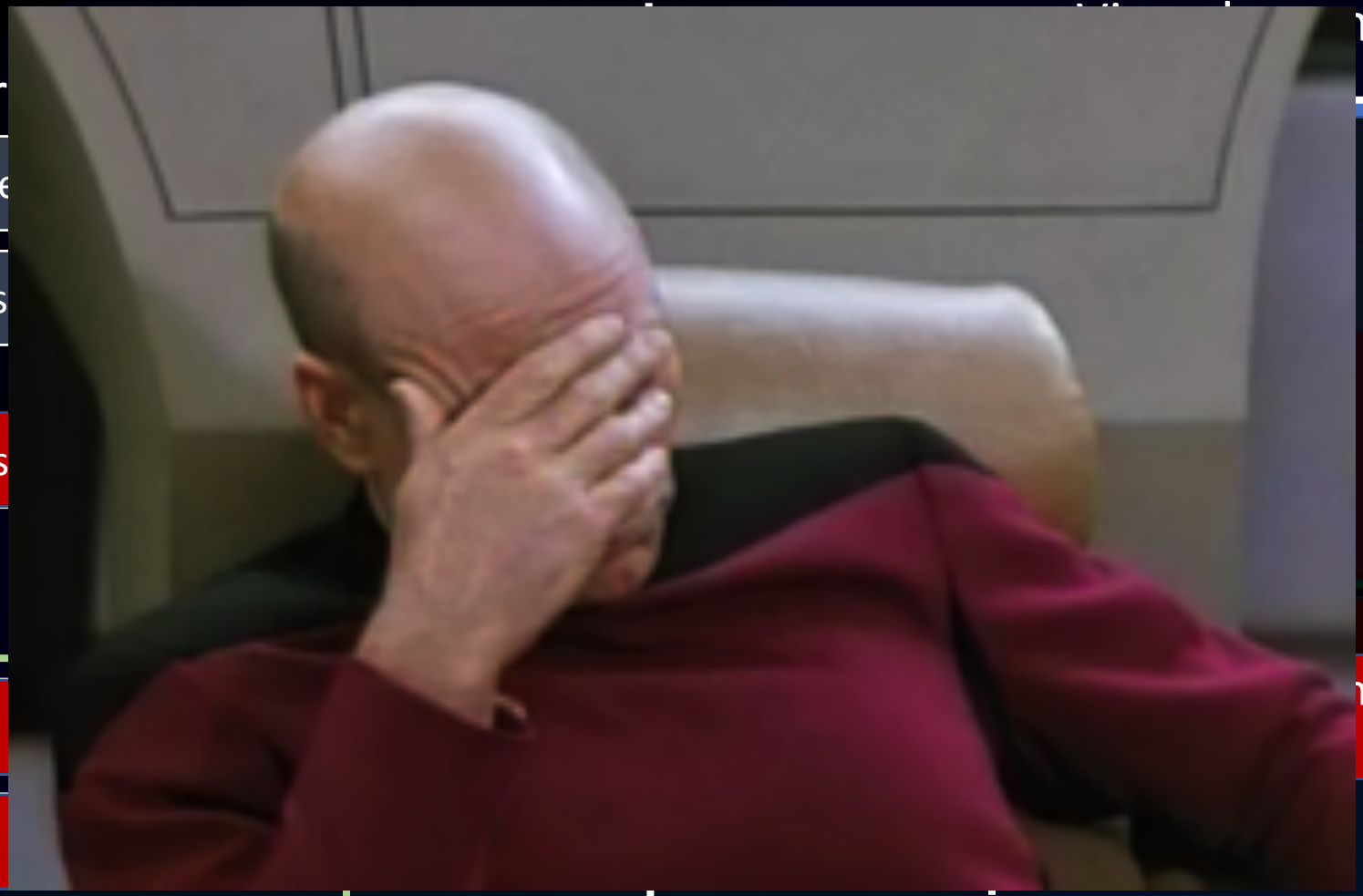
...

```
int copy_from_flash(void* dst, void* src, int size);
```



```
int copy_from_flash(void* dst, void* src, int size);
```

Flash



BIOS Dir

Heade

ID | Addres

...

ID | Addres

ID

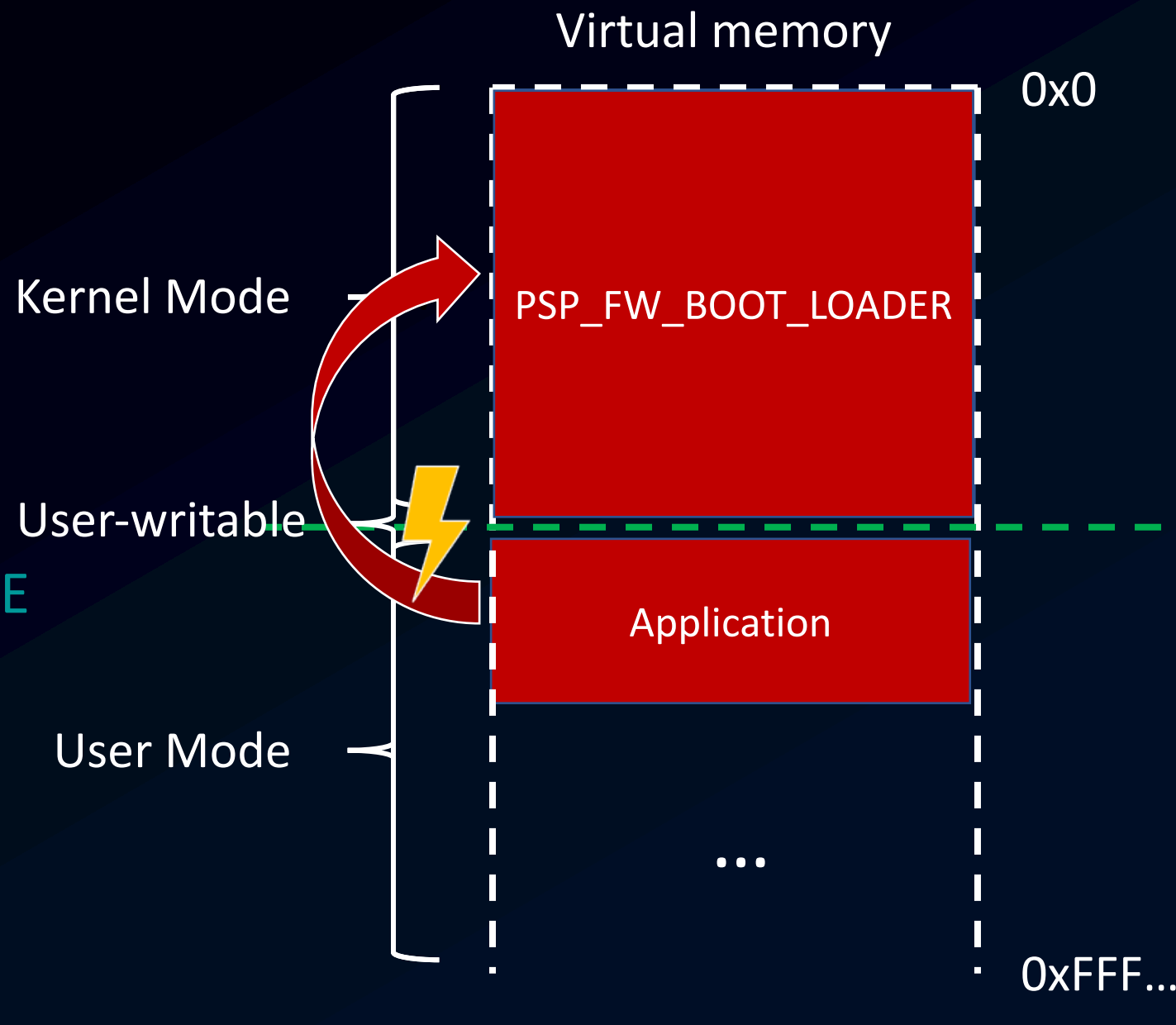
ID | Address | Size

PAGE TABLES

...

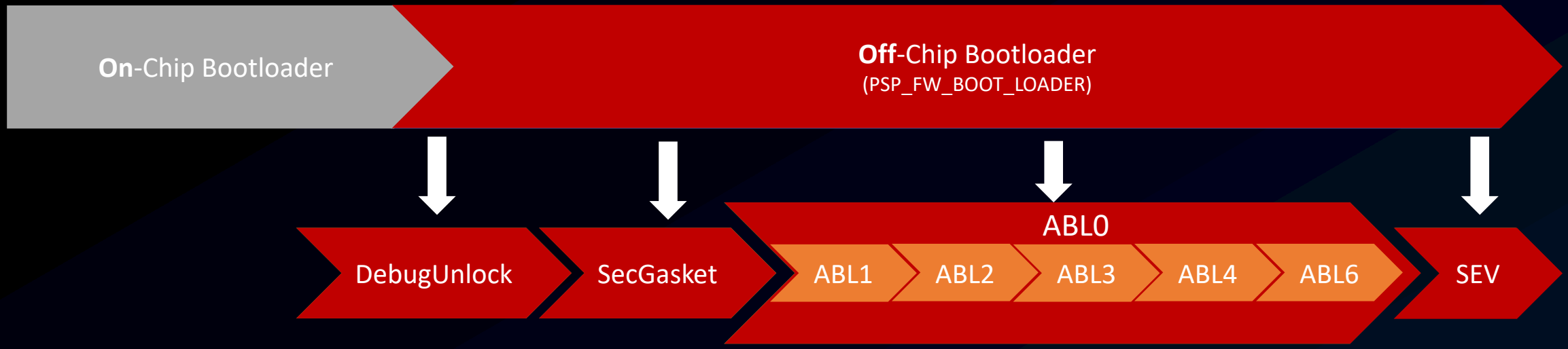
?

Off-Chip Bootloader
(PSP_FW_BOOT_LOADER)



VIRTUAL ADDRESS SPACE

Overwriting the page tables allows us to declare all memory as user-writable.



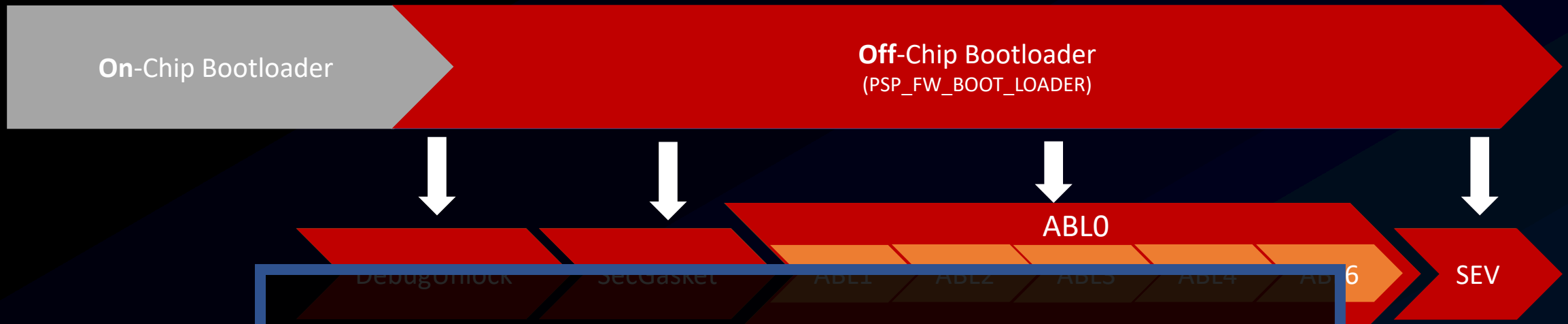
BOOT PROCESS

- Directory parsing takes place before loading any application.

-> We control the user mode beginning from the first application.

-> We control the kernel mode beginning from the first application.

AMD has fixed these issues!



The PSP does ***not*** implement roll-back prevention.

We can always re-flash a vulnerable firmware.

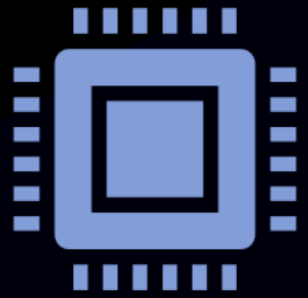
BOOT PROCESS

- Directory parsing takes place before loading any application.

-> We control the user mode beginning from the first application.

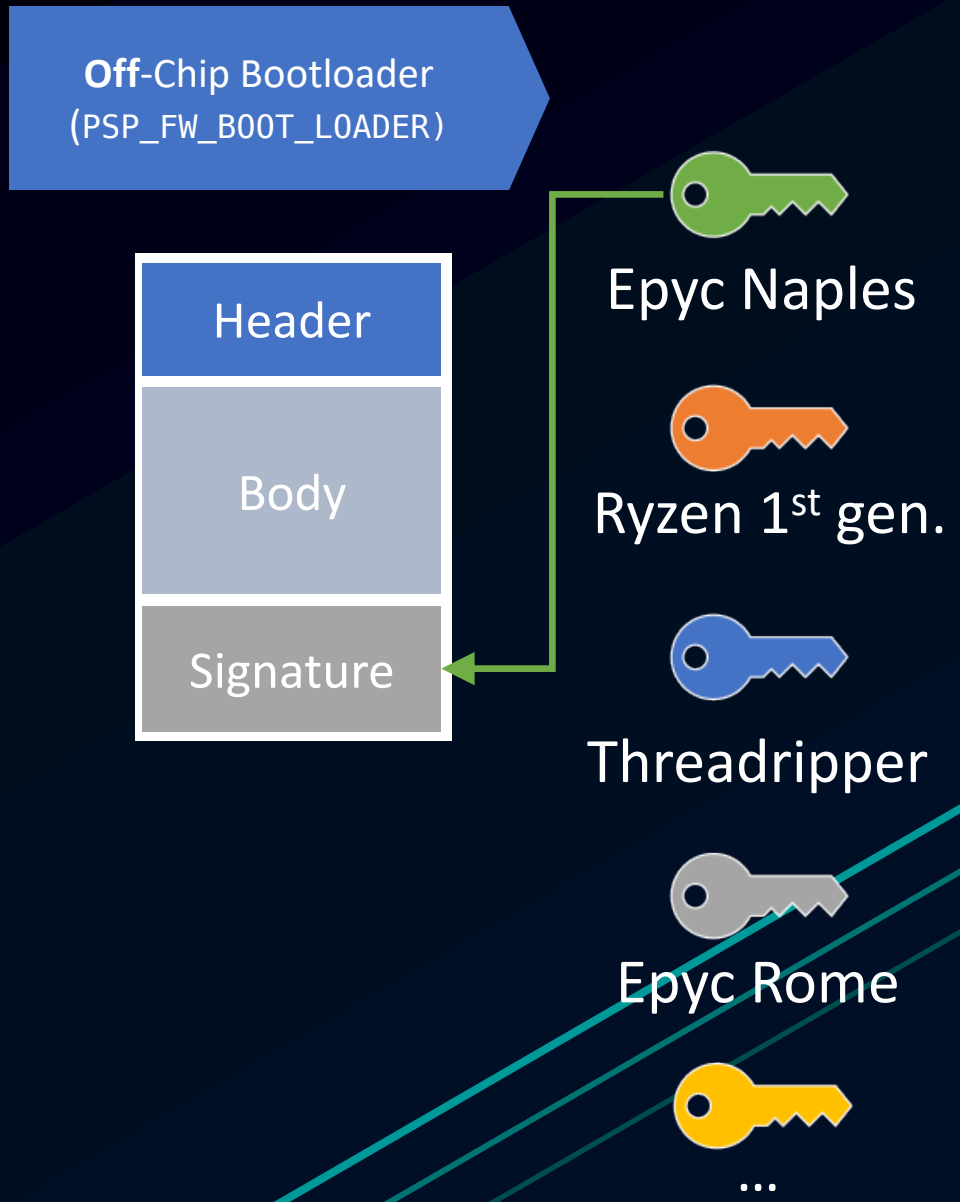
-> We control the kernel mode beginning from the first application.

AMD has fixed these issues!



Affected Systems

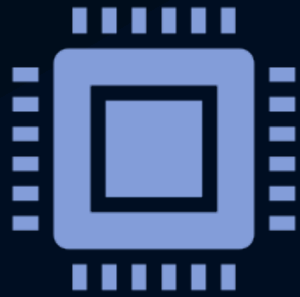
- Epyc Naples (Zen1)
  - Proven with our setup
- Ryzen 1st gen.
  - *probably*
- The rest
  - ???



Is this an (security) issue?

Depends …

- Physical access is required (UEFI flashing)



Issue for:

- Secure boot.
- Trusted Execution Environment.
- Secure Encrypted Virtualization (SEV)
  - Paper: Insecure Until Proven Updated

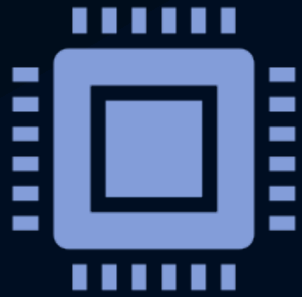
Buhren, Robert, Christian Werling, and Jean-Pierre Seifert. "Insecure Until Proven Updated: Analyzing AMD SEV's Remote Attestation." Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2019.

This is an opportunity!

Gain more insight into the PSP!

Allows further research on other subsystems

- PSP loads **SMU firmware**
- PSP allows access to **SMM code**
- PSP loads **UEFI code**



Idea
By Adrien Coquet, FR

magnifier
By Desainer Kanan, ID

UNCOVER, UNDERSTAND, OWN

Regaining Control Over Your AMD CPU

THANK YOU

Christian Werling
Security Research Labs

Alexander Eichner
Technische Universität Berlin

Robert Buhren
Technische Universität Berlin

Security Research Labs

Technische Universität Berlin

Security in Telecommunications

Further details

- **Github repository** https://github.com/PSPReverse
- **Reverse engineering** Talk at Camp'19

  Dissecting the AMD Platform Security Processor

  https://media.ccc.de/v/thms-38-dissecting-the-amd-platform-security-processor
- **Cloud security** Paper at CCS'19

  Insecure Until Proven Updated: Analyzing AMD SEV's Remote Attestation

  https://arxiv.org/abs/1908.11680
- Linux TEE kernel patches: https://lkml.org/lkml/2019/10/23/449