RingHopper - Hopping from User-space to God Mode

Jonathan Lusky
Benny Zeltser
intel

BlueHat IL 2023

grasshopper photo by Eka P. Amdela on Unsplash

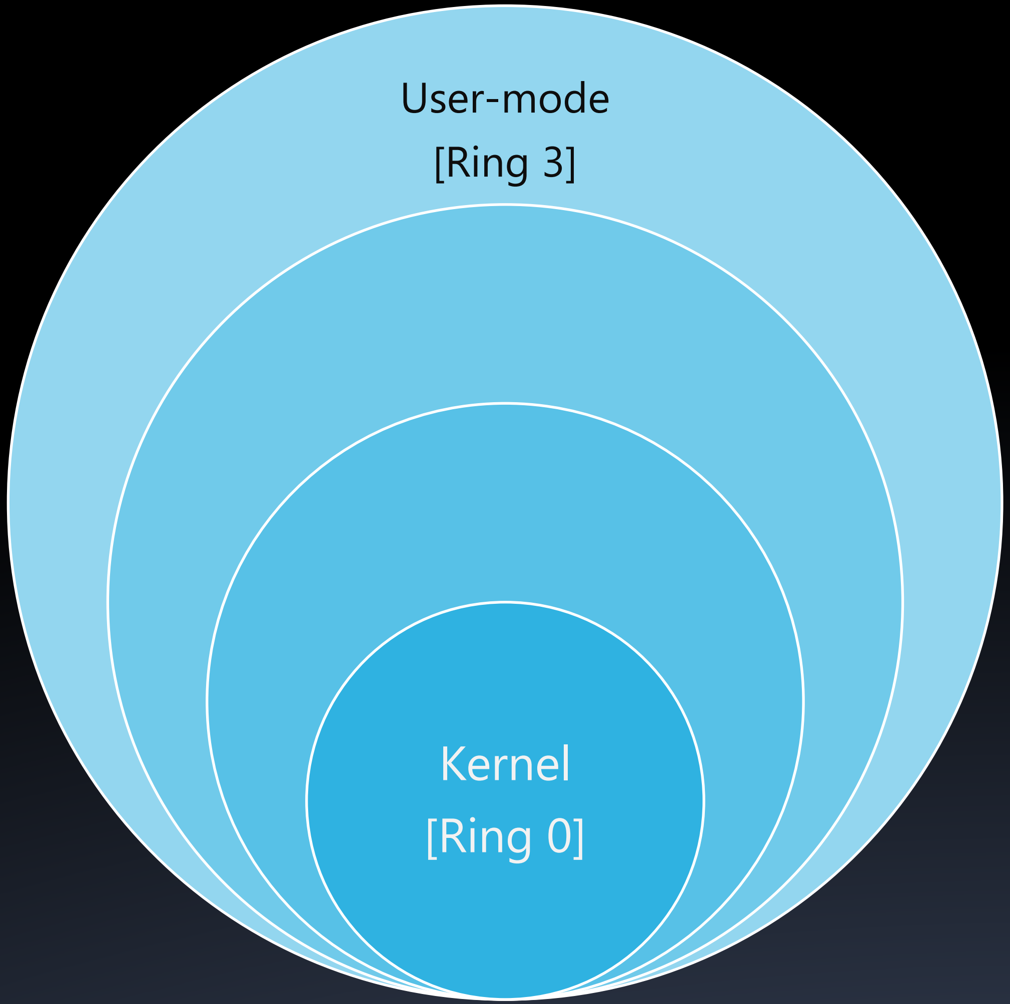
Notices and Disclaimers

- No product can be absolutely secure.
- Intel technologies may require enabled hardware, software or service activation.
- Results have been estimated or simulated.
- Your costs and results may vary.
- © Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

Overview

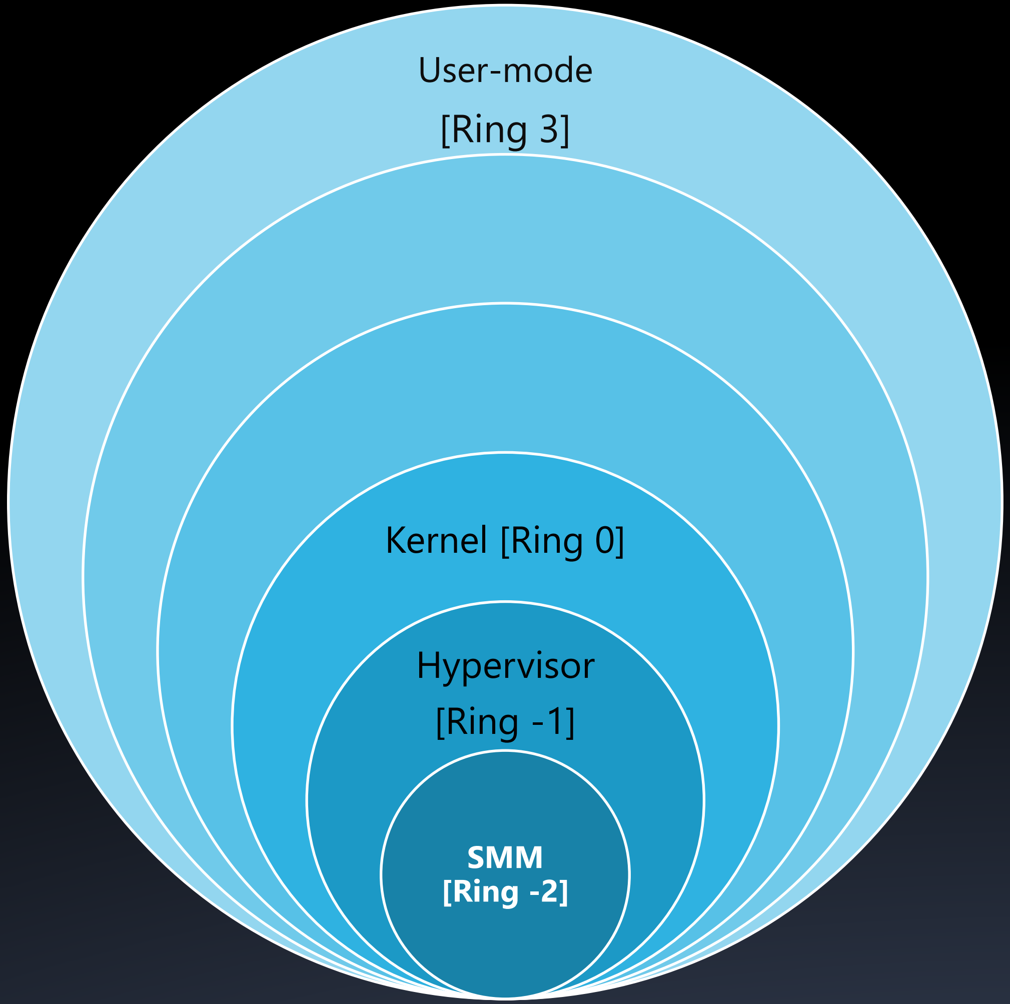
The story of how we obtained write primitives,
hopped into privileged mode,
and acquired total* world domination 😎

Privilege Rings



Privilege Rings

Why so negative?



System Management Mode

How it Started

- Processor operating mode
- Provides low-level system functionality:
  - Power management
  - System hardware control
  - Proprietary OEM designed code
- Transparent to the Hypervisor/OS

System Management Mode

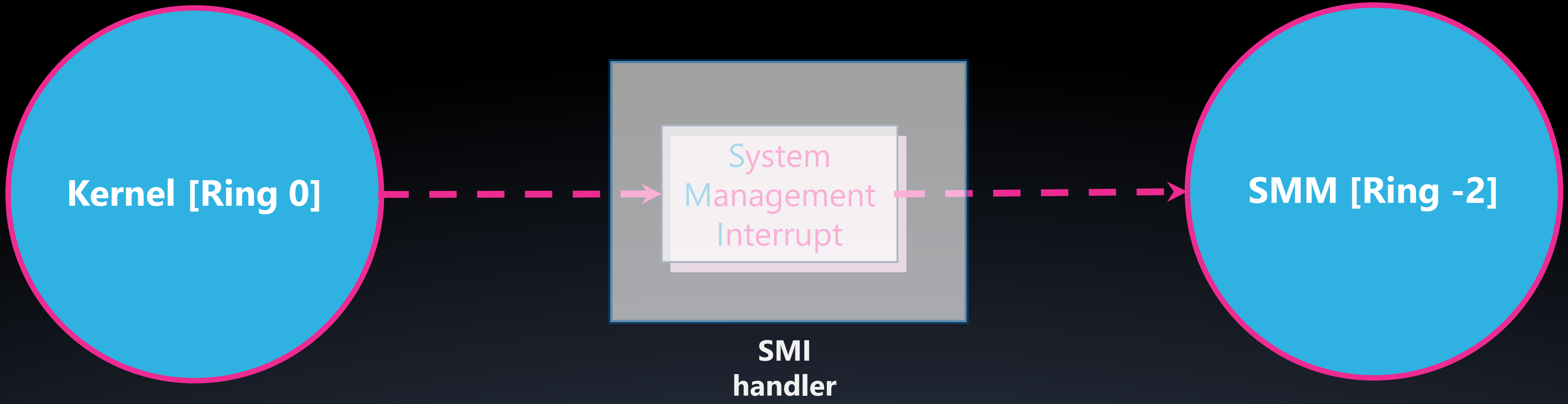
How it's Going

- Wide range of functionalities:
  - Handle USB events at boot time and run time
  - System Management BIOS
  - Many more…

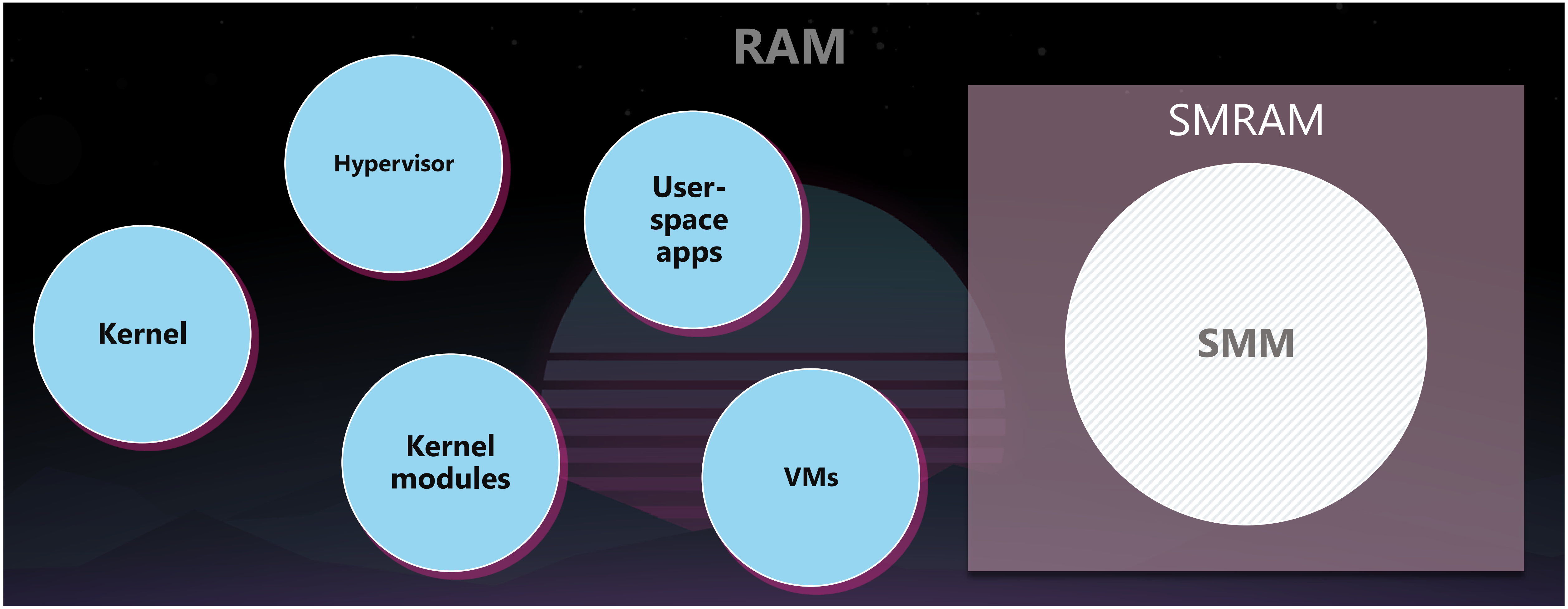
source: http://gunshowcomic.com/648

- Well-guarded

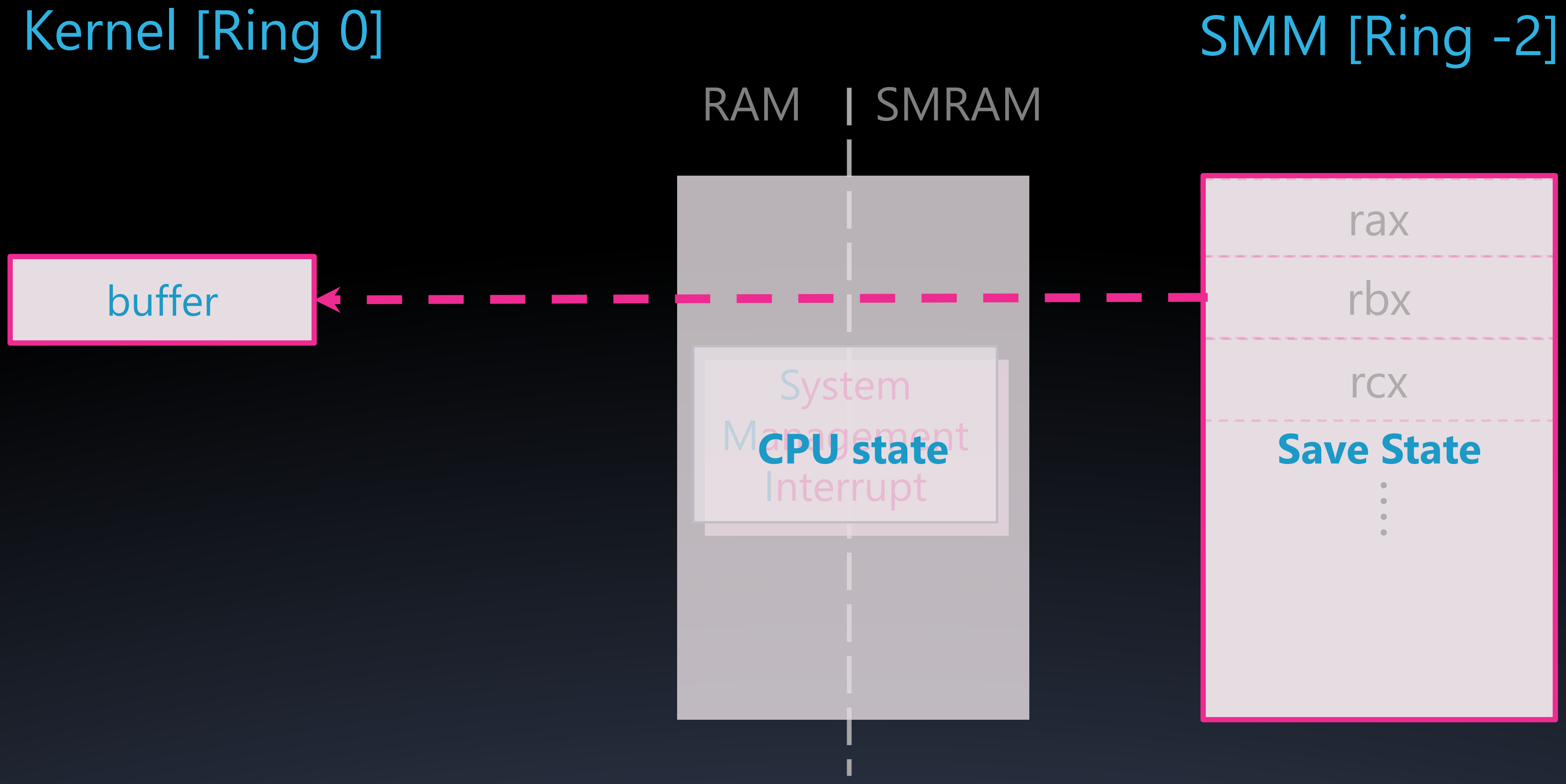
Invoking SMM functions from ring 0



System Management RAM



Communication with SMM



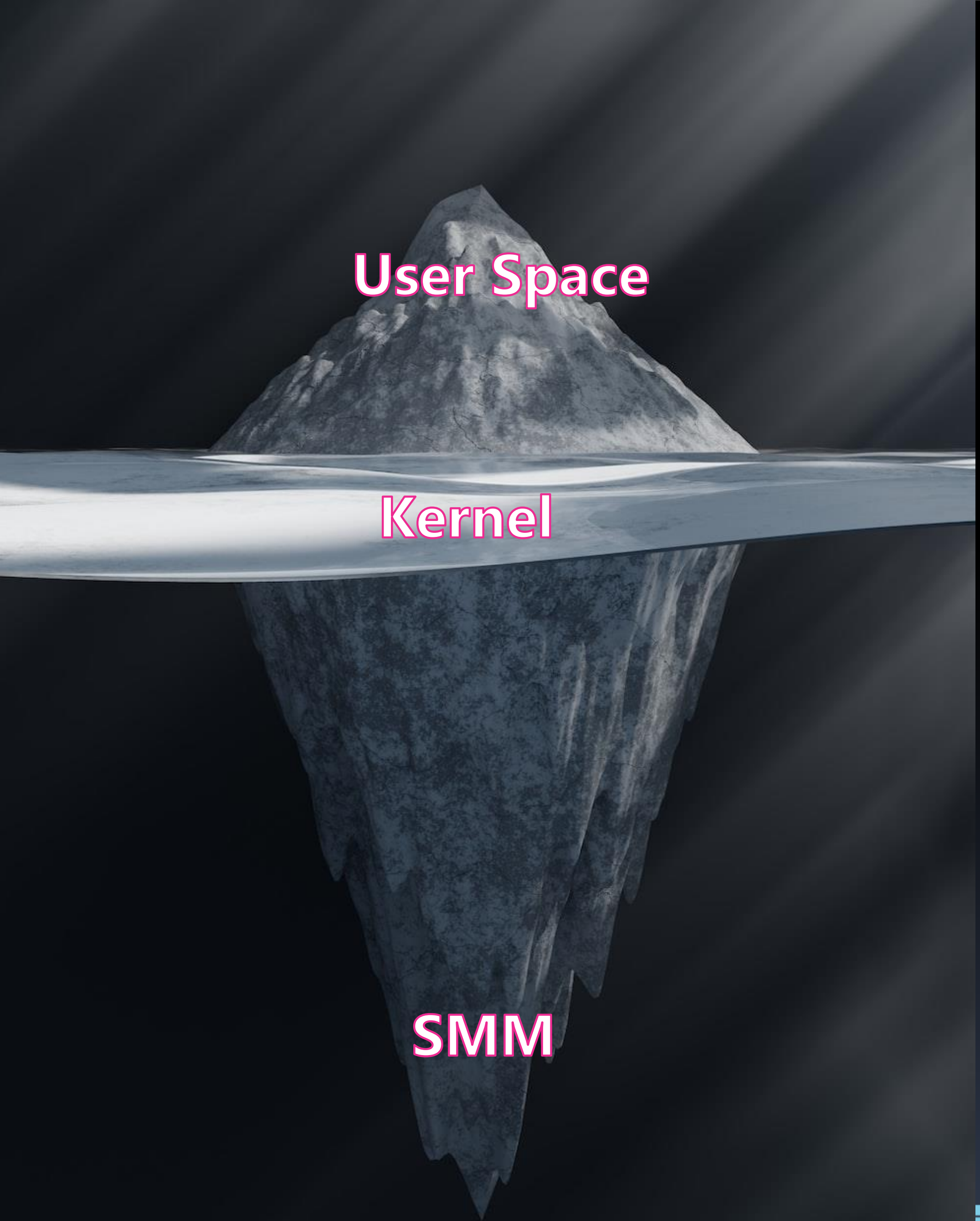
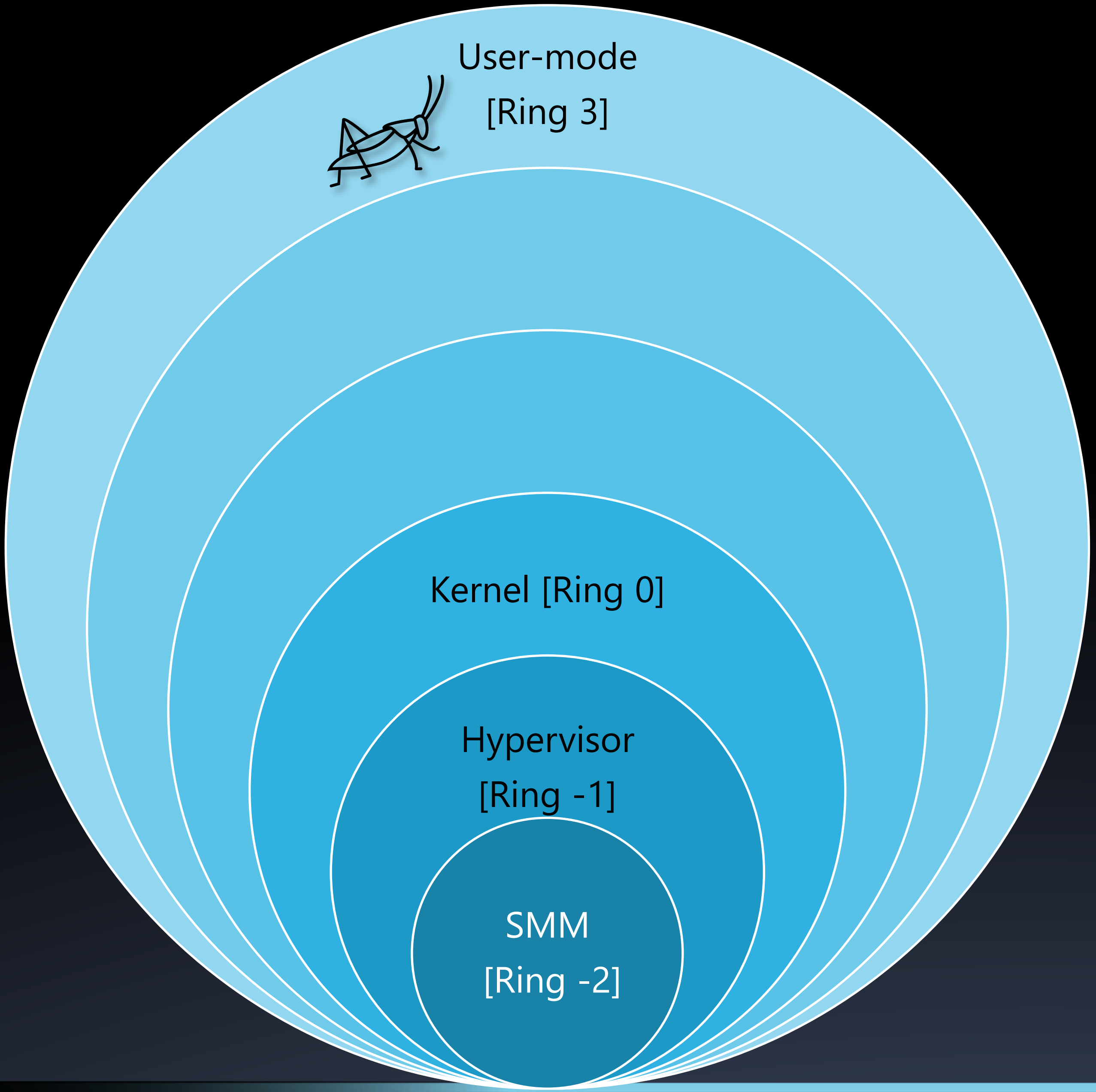


Photo by SIMON LEE on Unsplash

SMM is where you want to be:

- Brick platform
- Steal sensitive information
- Evade different OS security mechanisms
- Install a BootKit
- Disable secure boot
- etc.

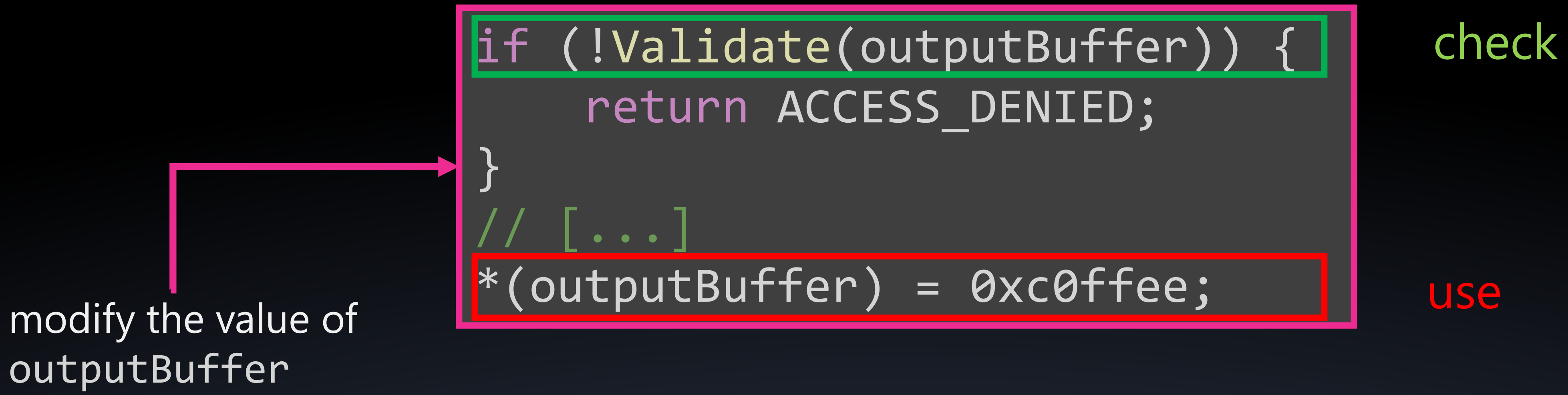


Privilege escalation

Our target

Intel® NUC (Next Unit of Computing)

Time Of Check Time Of Use Vulnerability

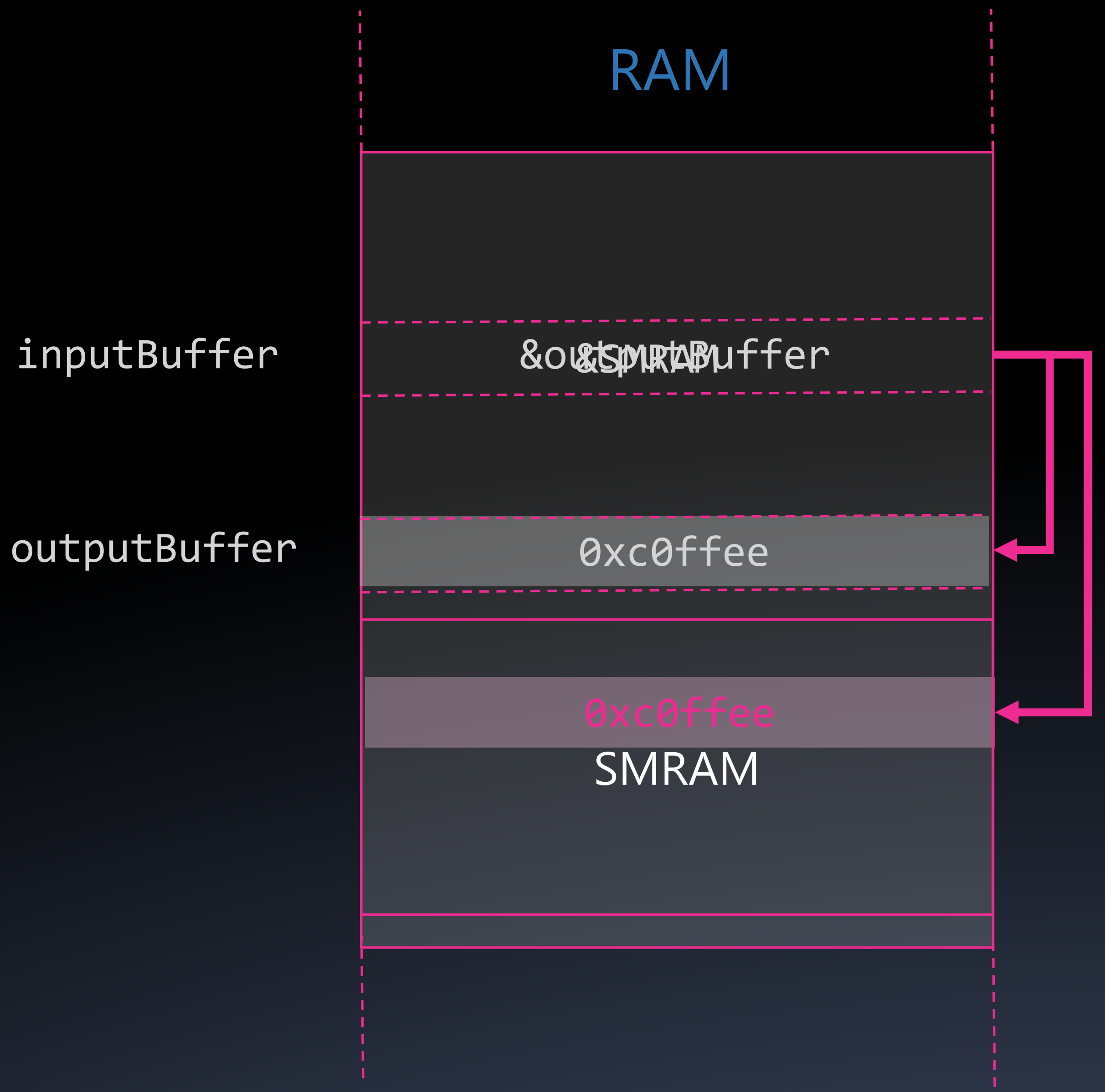


```
if (!Validate(outputBuffer)) {
    return ACCESS_DENIED;
}
// [...]
*(outputBuffer) = 0xc0ffee;
```

check

use

modify the value of outputBuffer



TOCTOU Vulnerability

Toy Example

TOCTOU Vulnerability

Toy Example

```
EFI_STATUS EFIAPI CoffeeSmiHandler(EFI_HANDLE DispatchHandle, CONST VOID
                              *Context, VOID *CommBuffer, UINTN *CommBufferSize) {
    UINT32* inputBuffer = NULL;
    // [...]
    mSmmCpu->ReadSaveState(mSmmCpu, sizeof(UINT32),
                           EFI_SMM_SAVE_STATE_REGISTER_RBX,
                           gSmst->CurrentlyExecutingCpu, inputBuffer);
    // [...]
    if (!SmmIsBufferOutsideSmmValid) {
        DEBUG ((EFI_D_INFO, "Missing validation protocol\n"));
        return EFI_ERROR;
    }
    // [...]
    if (!SmmIsBufferOutsideSmmValid(*inputBuffer, 0x4)){
                return EFI_ACCESS_DENIED;
    }
    **inputBuffer = 0xc0ffee;
    // [...]
}
```

(*inputBuffer == &outputBuffer)

check

use

Reading value of RBX from the Save State

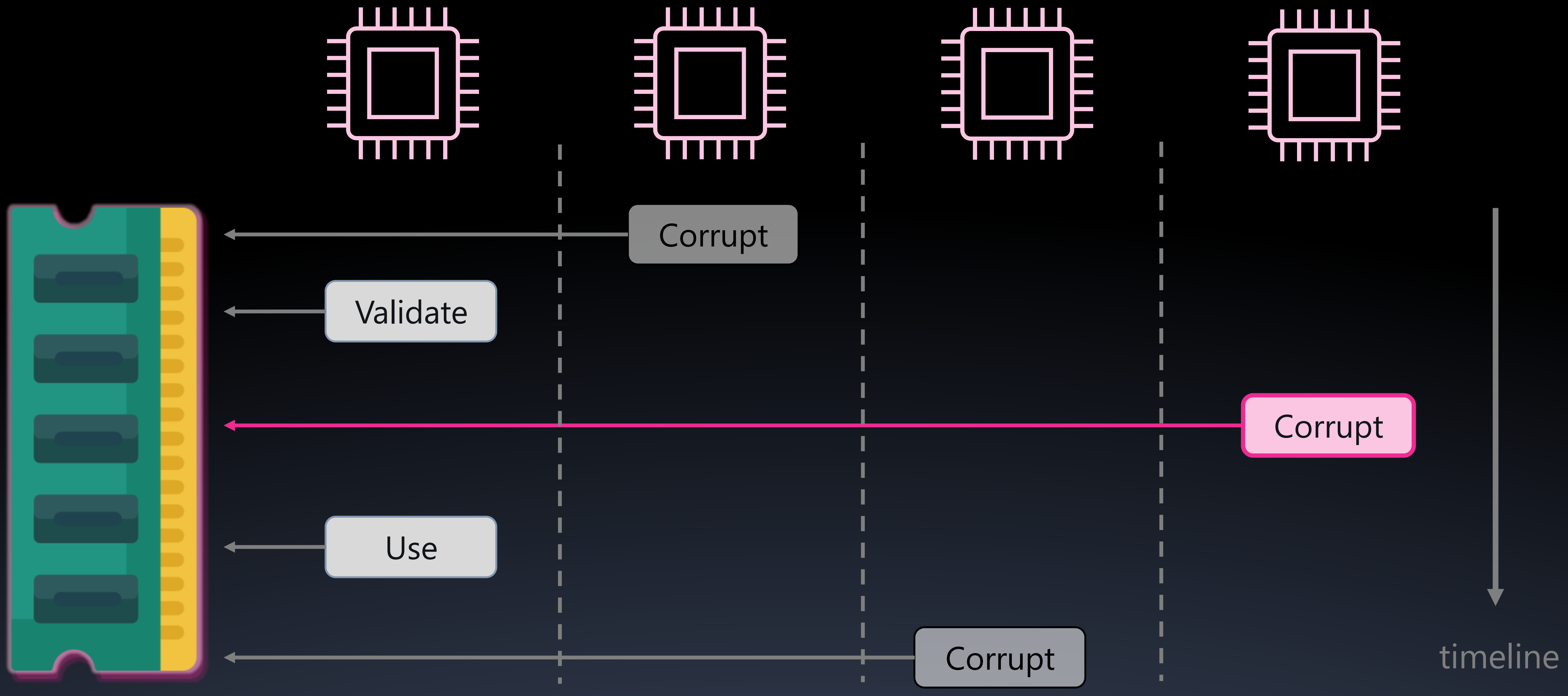
Checking the buffer is not in SMRAM

Modifying *inputBuffer between the check and use

Assigning a value to the buffer

BlueHat IL

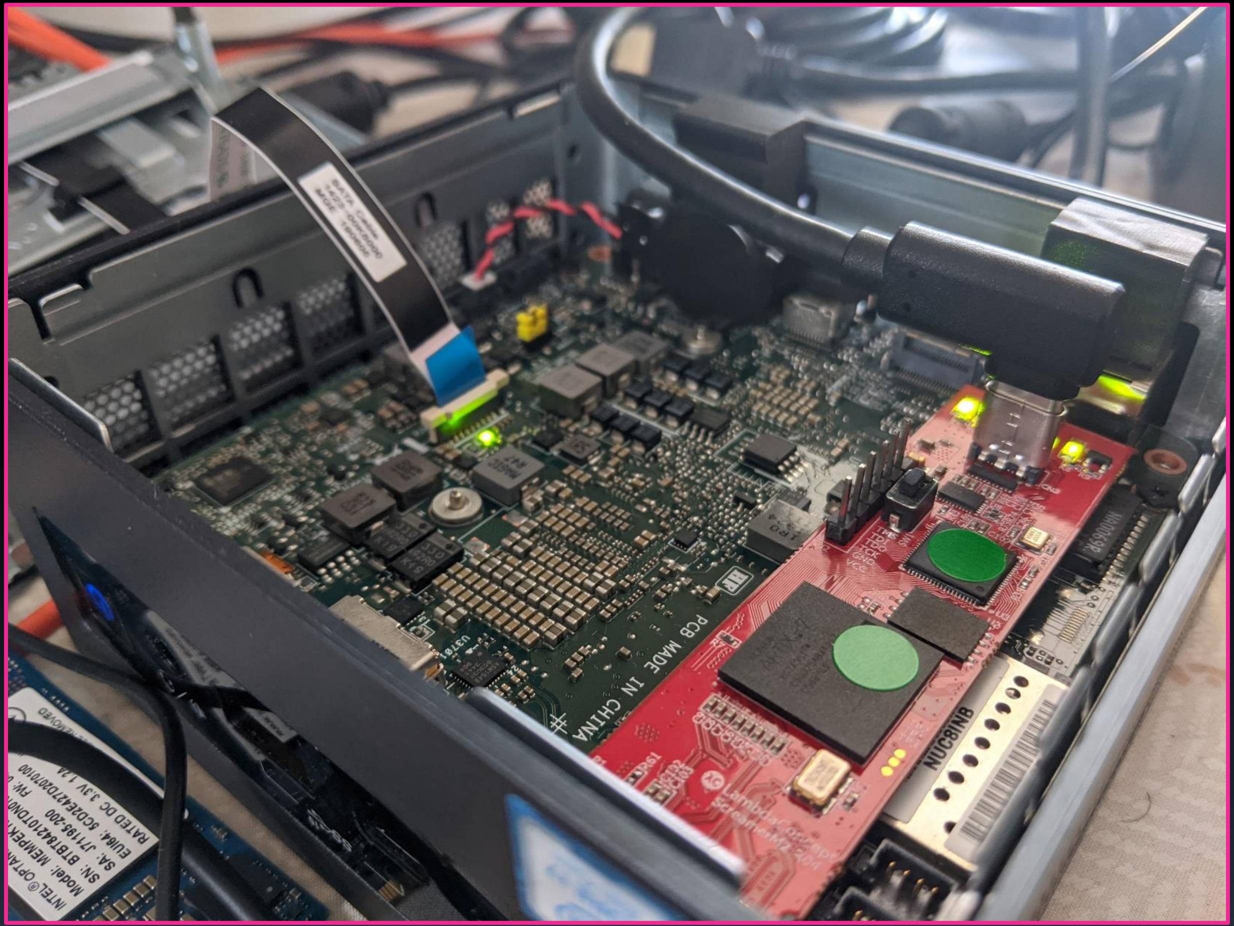
TOCTOU Classic Exploitation



DMA

DMA is the way of peripheral devices to access RAM directly, without the CPU

DMA via PCILeech



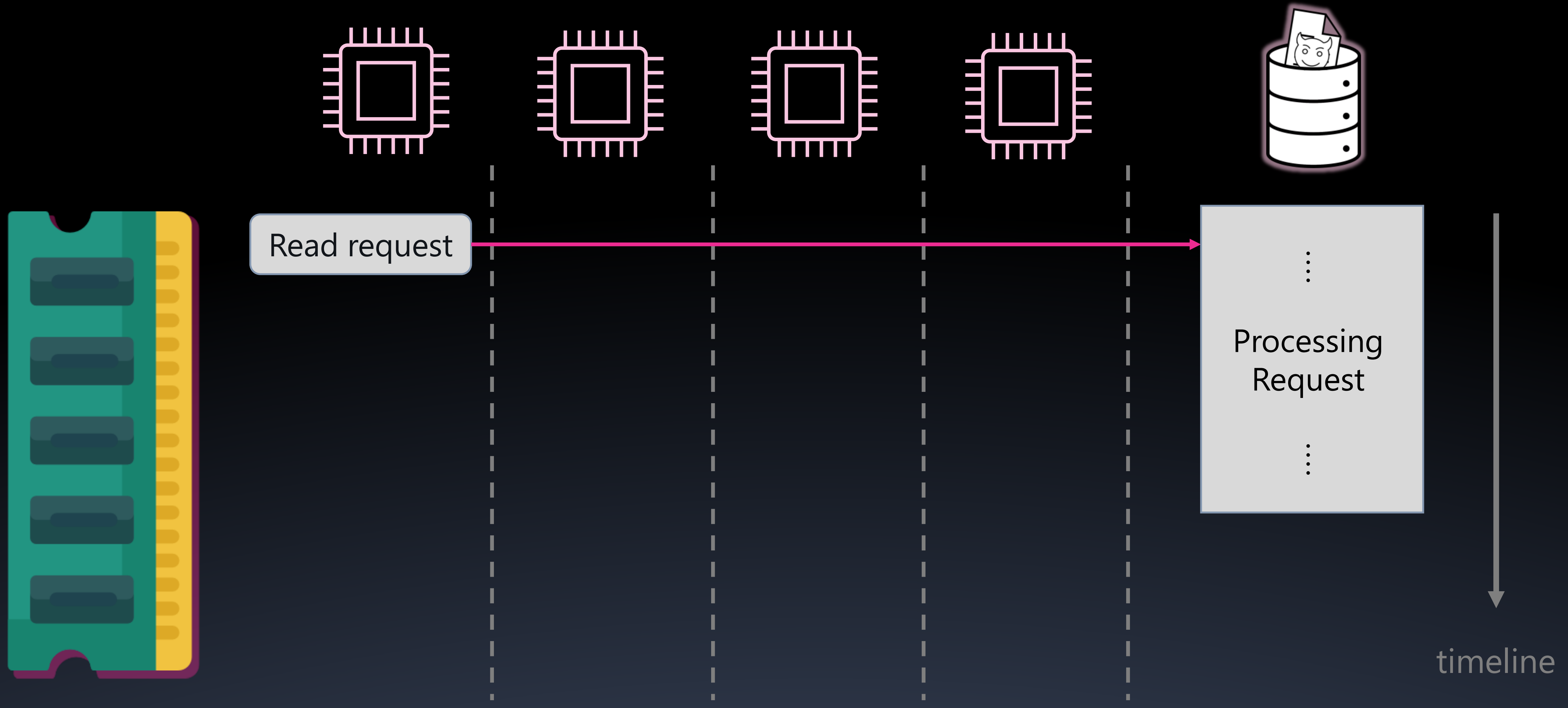
awesome tool by Ulf Frisk - https://github.com/ufrisk/pcileech

Photo by Frank R on Unsplash

Physical to remote

- Utilized the HDD to perform DMA
- Generated DMA transactions based on work by Rafal Wojtczuk in *Subverting the Xen Hypervisor*

TOCTOU SMM Exploitation



Read request

Processing Request

TOCTOU SMM Exploitation

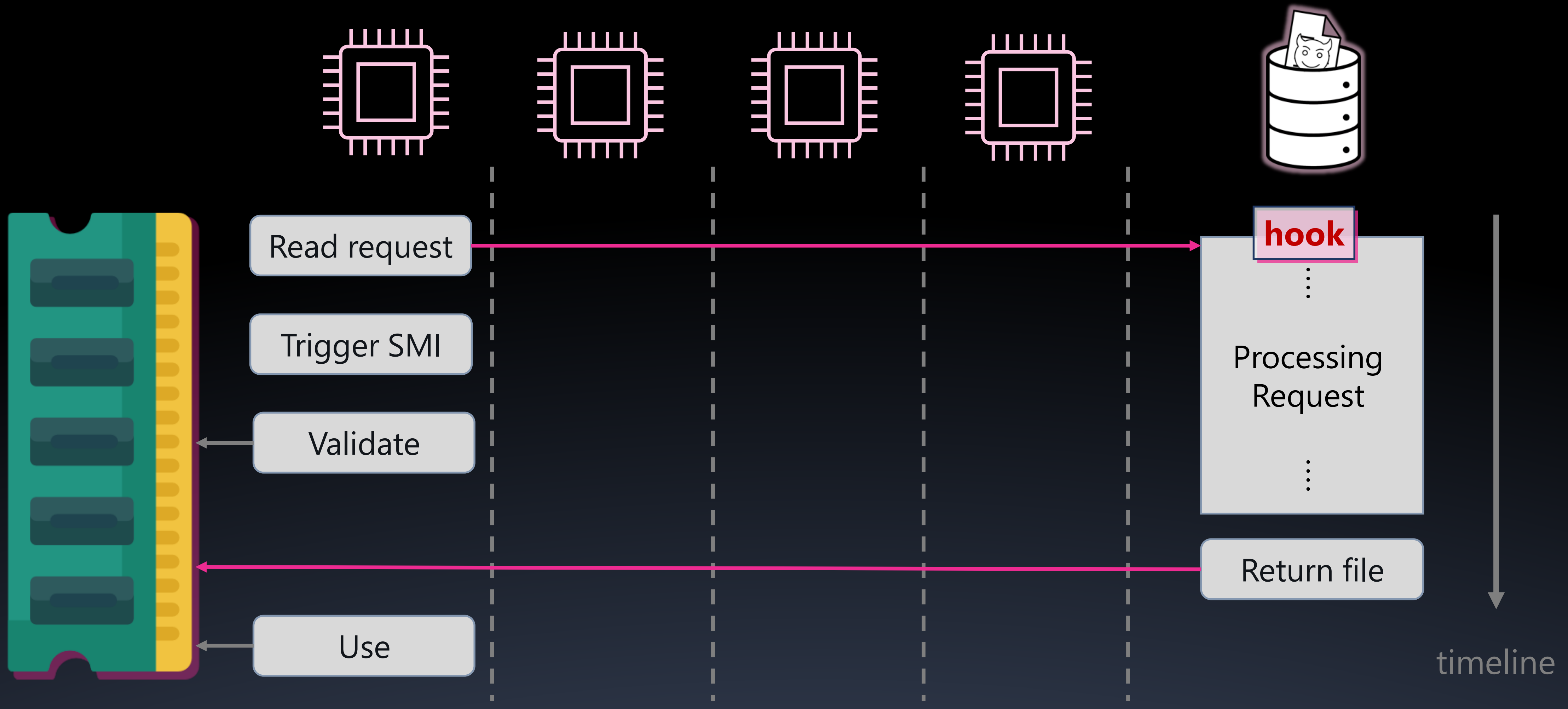


Photo by Victor Serban on Unsplash

Recap

- ☑ What is SMM and how to work with it
- ☑ Turning TOCTOU issues into write primitive to the SMRAM
- ☑ Manipulating DMA transactions
- ☑ Executing code in SMM

Recap

☑ What is SMM and how to work with it

☑ Turning TOCTOU issues into write primitive to the SMRAM

☑ Manipulating DMA transactions

☑ ~~Executing code in SMM~~

Code Execution

Initial capabilities

SmbiosDmiEdit DXE driver

Code Execution

Initial capabilities

Write-primitives from the *SmbiosDmiEdit* DXE driver

```
**(input_buffer + 2) = 0x28;
**(input_buffer + 6) = sub_2428(qword_6D58, v3);
**(input_buffer + 0xa) = sub_248C(qword_6D58);
**(input_buffer + 0xe) = qword_6C08 ? qword_6C08 : qword_6D58;
**(input_buffer + 0x12) = word_6D68;
```

Code Execution

Classic Approach

Find an executable memory region

Forge arbitrary payload

Get unrestricted memory access

Code is RO, data is NX

Weak write primitives

Static + RO page table

Code Execution

Challenges

A classic approach might not work

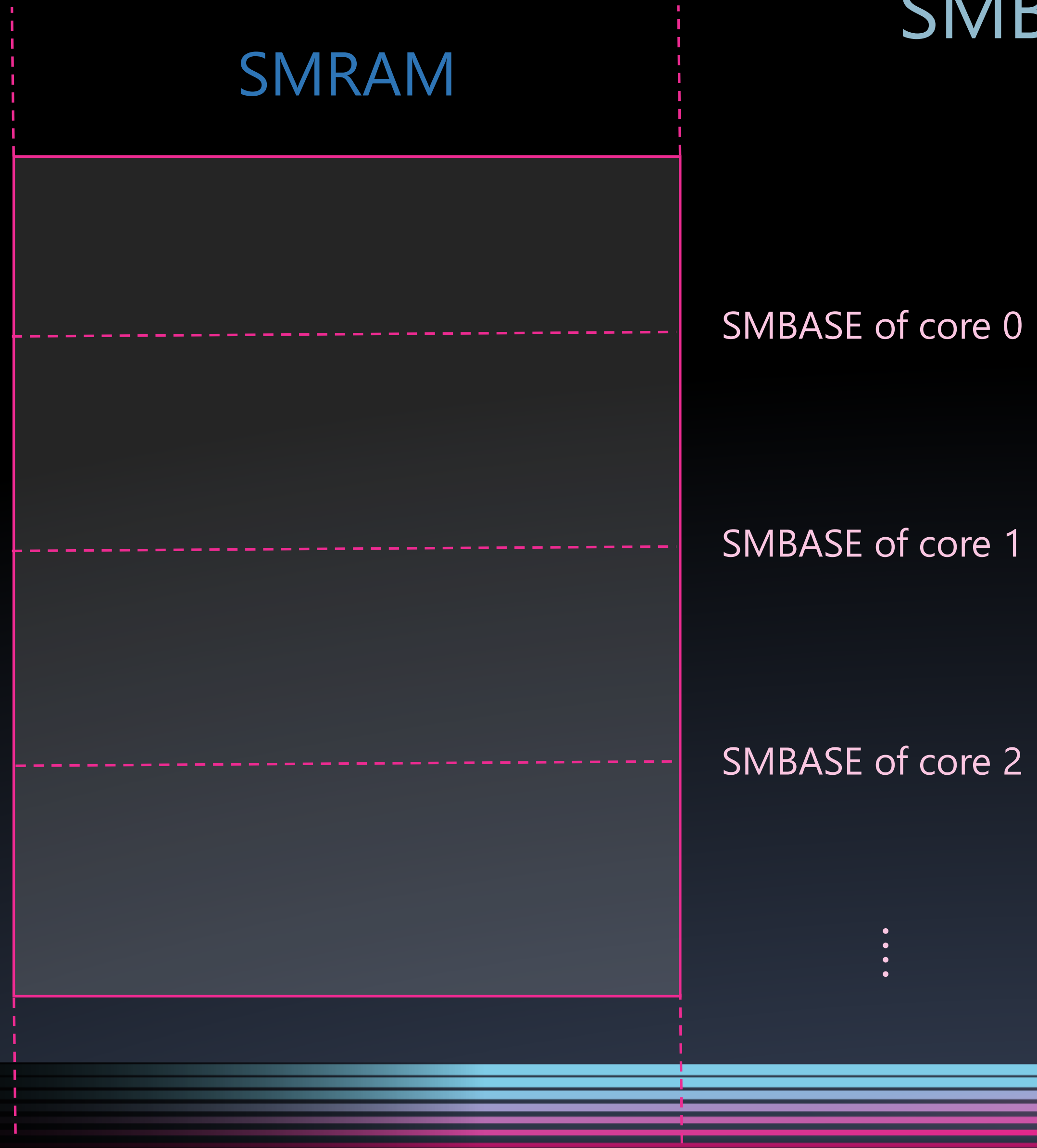
source: https://www.mememaker.net/meme/such-challenge-very-hard

=> Let's try to leverage SMM internal mechanisms to our advantage

Code Execution

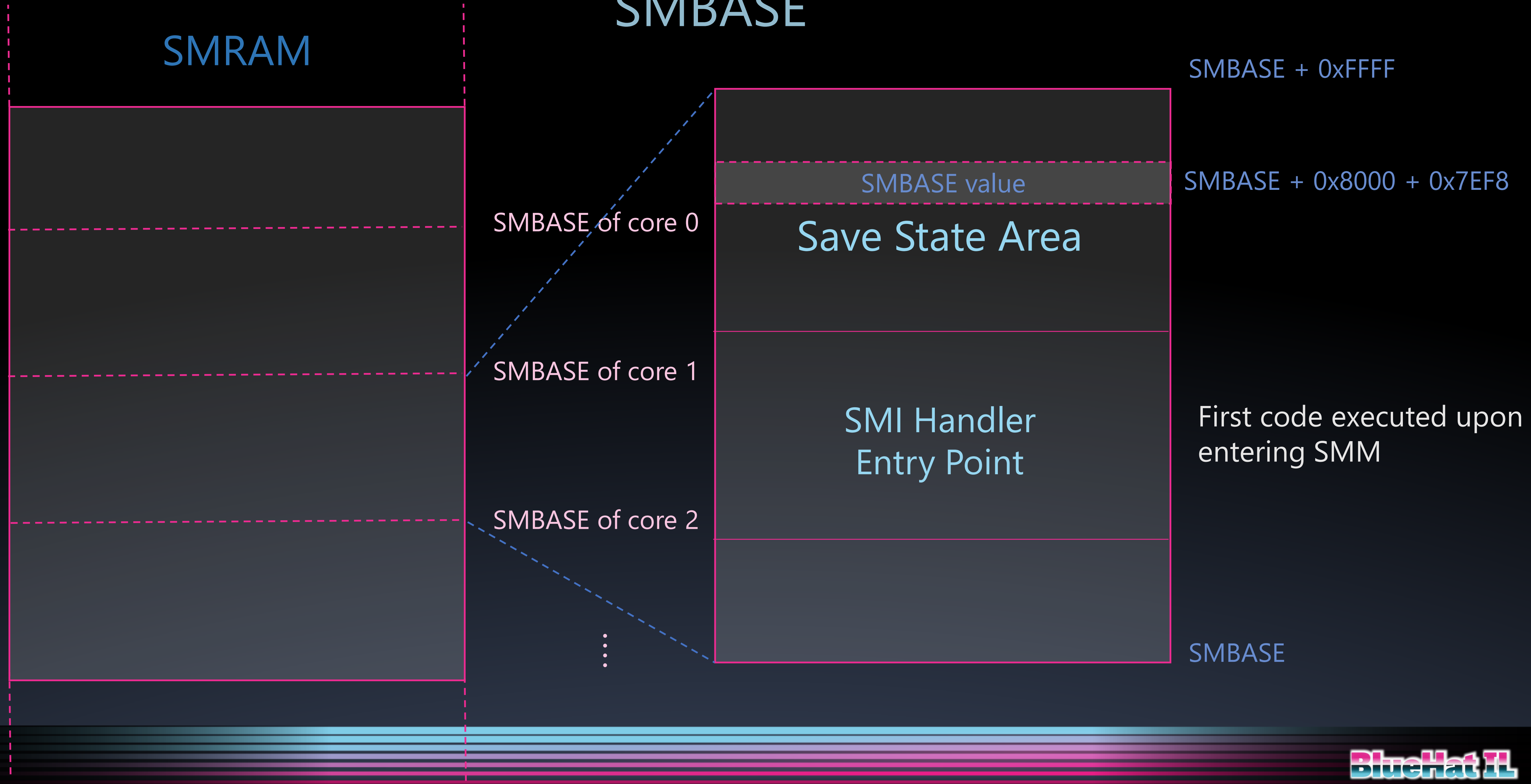
Code Execution

SMBASE



Code Execution

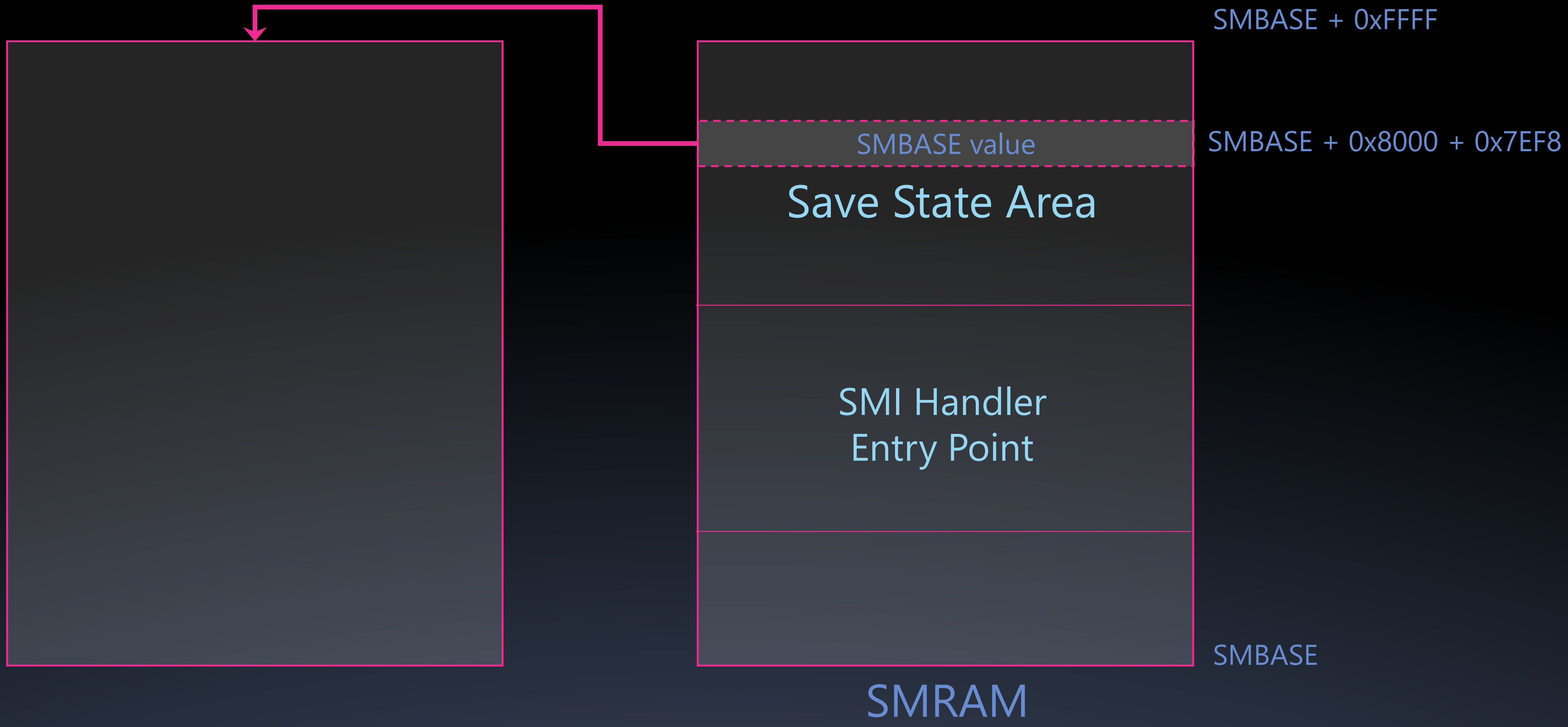
SMBASE



First code executed upon entering SMM

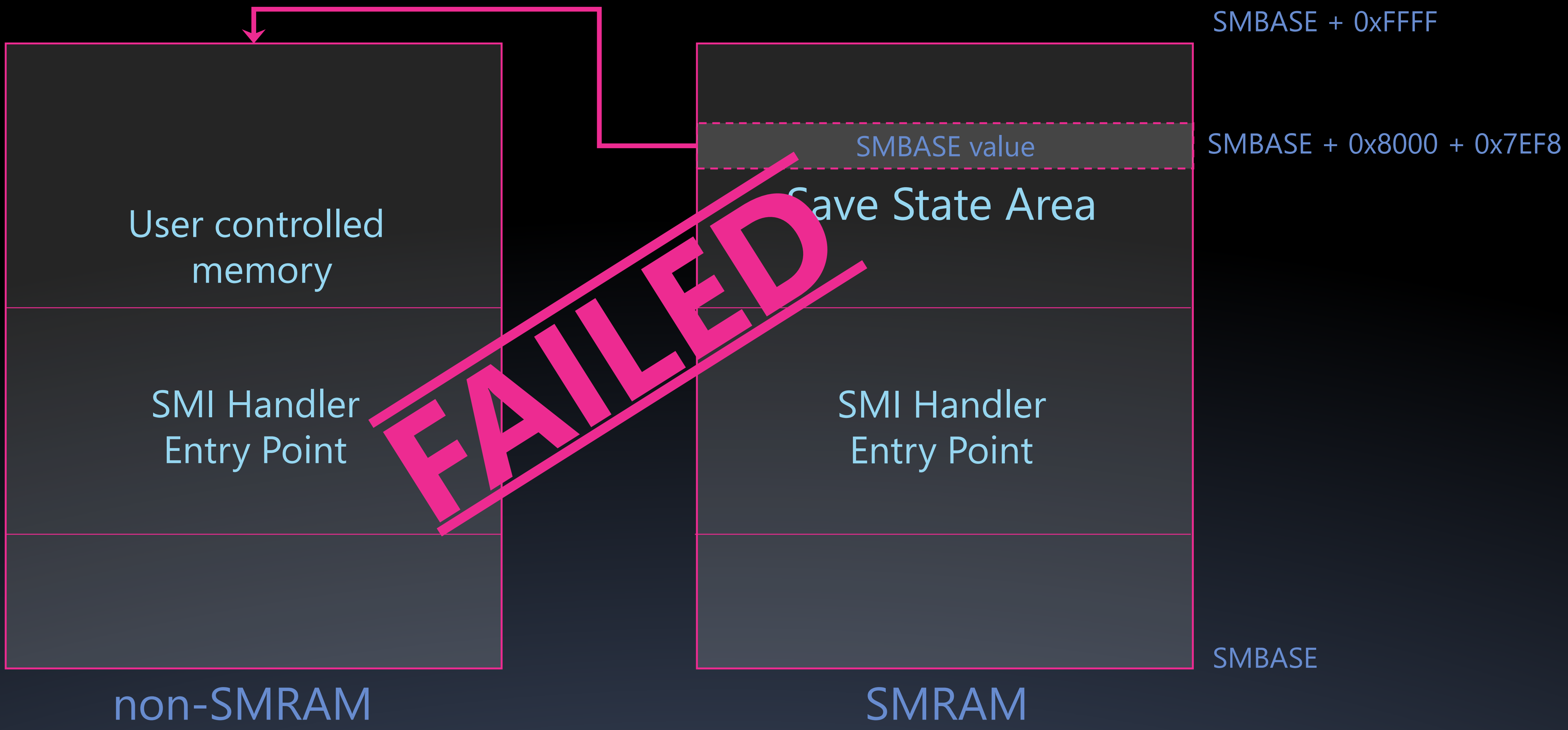
Code Execution

SMBASE Relocation



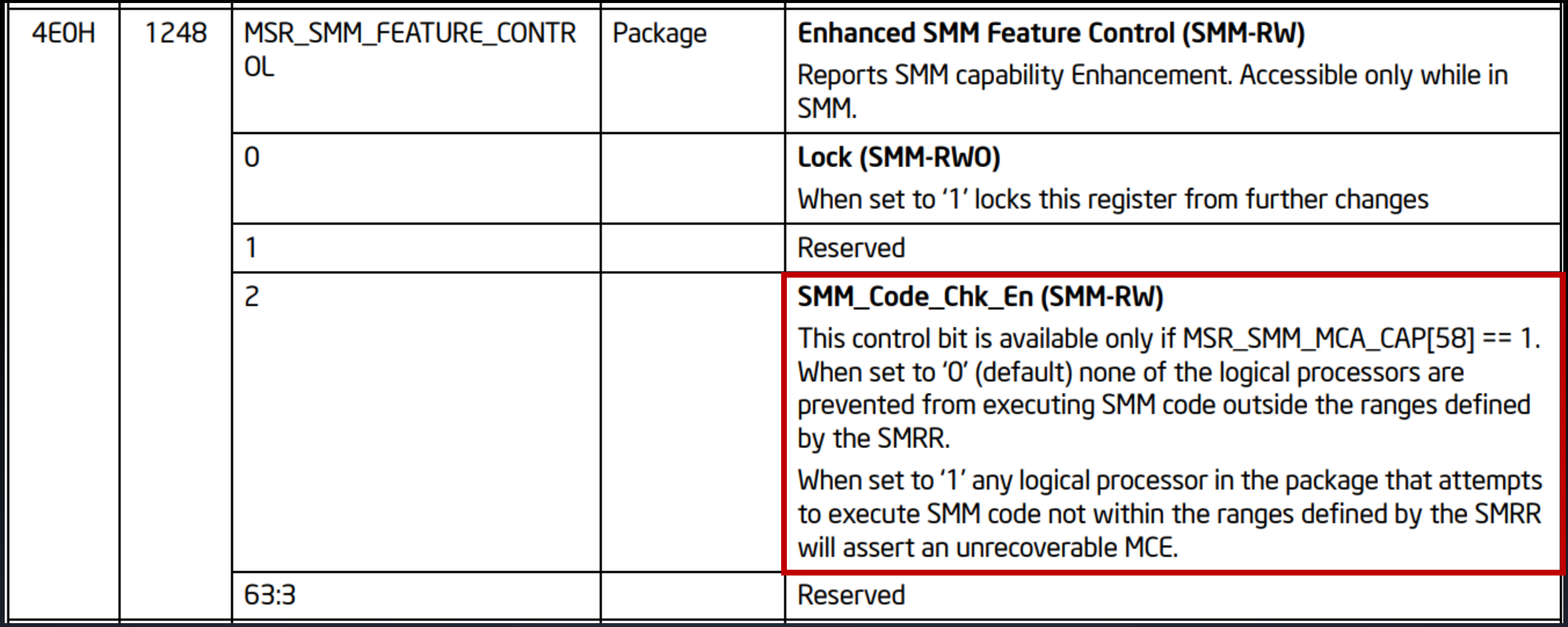
Code Execution

SMBASE Relocation Attack



Code Execution

SMM "SMEP"

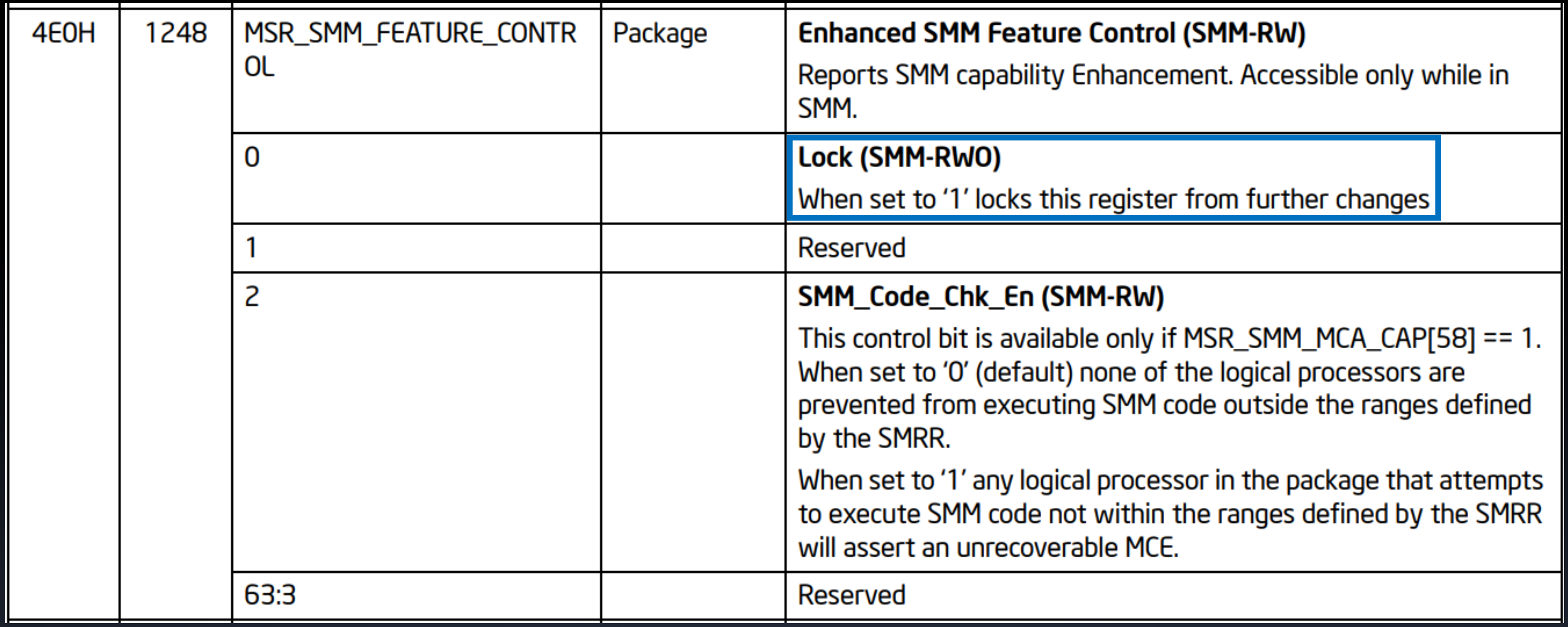


4E0H	1248	MSR_SMM_FEATURE_CONTROL	Package	**Enhanced SMM Feature Control (SMM-RW)** Reports SMM capability Enhancement. Accessible only while in SMM.
		0		**Lock (SMM-RWO)** When set to '1' locks this register from further changes
		1		Reserved
		2		**SMM_Code_Chk_En (SMM-RW)** This control bit is available only if MSR_SMM_MCA_CAP[58] == 1. When set to '0' (default) none of the logical processors are prevented from executing SMM code outside the ranges defined by the SMRR. When set to '1' any logical processor in the package that attempts to execute SMM code not within the ranges defined by the SMRR will assert an unrecoverable MCE.
		63:3		Reserved

Code Execution

SMM “SMEP”

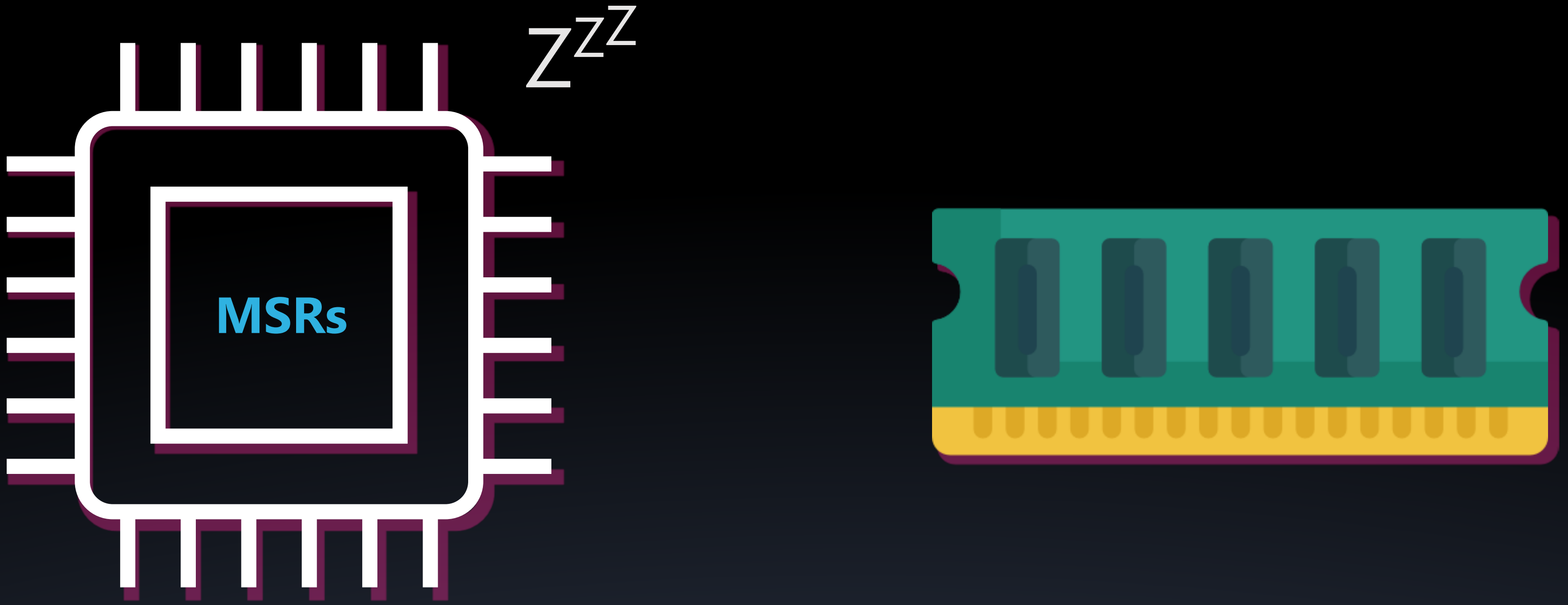
SMM_FEATURE_CONTROL cannot be modified until reboot…



4E0H	1248	MSR_SMM_FEATURE_CONTROL	Package	**Enhanced SMM Feature Control (SMM-RW)** Reports SMM capability Enhancement. Accessible only while in SMM.
		0		**Lock (SMM-RWO)** When set to ‘1’ locks this register from further changes
		1		Reserved
		2		**SMM_Code_Chk_En (SMM-RW)** This control bit is available only if MSR_SMM_MCA_CAP[58] == 1. When set to ‘0’ (default) none of the logical processors are prevented from executing SMM code outside the ranges defined by the SMRR. When set to ‘1’ any logical processor in the package that attempts to execute SMM code not within the ranges defined by the SMRR will assert an unrecoverable MCE.
		63:3		Reserved

… what if we cut the power to the CPU?

S3 sleep state



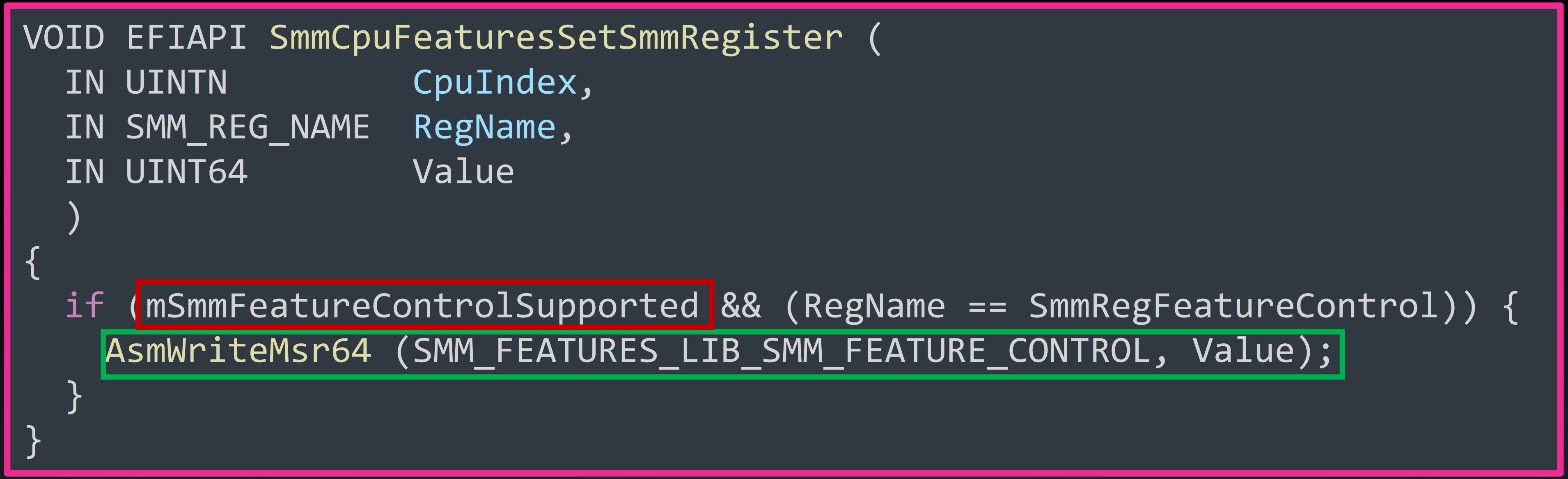
Code Execution

S3 sleep state

	`SMM_Code_Chk_En` state
Normal execution	`set`
Going into S3	`set`
Back from S3	`clear`
Initialization code	`set`

Code Execution

SMM “SMEP” + S3



```
VOID EFIAPI SmmCpuFeaturesSetSmmRegister (
  IN UINTN          CpuIndex,
  IN SMM_REG_NAME   RegName,
  IN UINT64         Value
  )
{
  if (mSmmFeatureControlSupported && (RegName == SmmRegFeatureControl)) {
    AsmWriteMsr64 (SMM_FEATURES_LIB_SMM_FEATURE_CONTROL, Value);
  }
}
```

Code Execution in SMM – full recipe

1. Set the value o
2. Go into S3 slee
3. Return from S3
4. Create a fake S
5. Modify the SM t memory
6. Trigger an SMI

Defeating RO pages

- SMI Handler Entry Point:
    - Starts running in real mode
    - Initializes the page table (setting cr3)

- We execute our own SMI Handler Entry Point

=> We’re accessible to all DRAM w/o page-table restrictions

Code Execution in SMM

Mitigations

RO Memory

https://edk2-docs.gitbook.io/a-tour-beyond-bios-mitigate-buffer-overflow-in-ue/summary/policy_control

Code Execution in SMM

Mitigations

NX/RO Memory

Heap Guard

SMM Static Paging

https://edk2-docs.gitbook.io/a-tour-beyond-bios-mitigate-buffer-overflow-in-ue/summary/policy_control

Code Execution in SMM

Mitigations

We don’t mind these mitigations:

Stack Guard
NULL pointer detection
Heap Guard
Memory Profile
NX Stack

NX/RO Memory
Image Protection
SMM Static Paging
Read-only Page Table

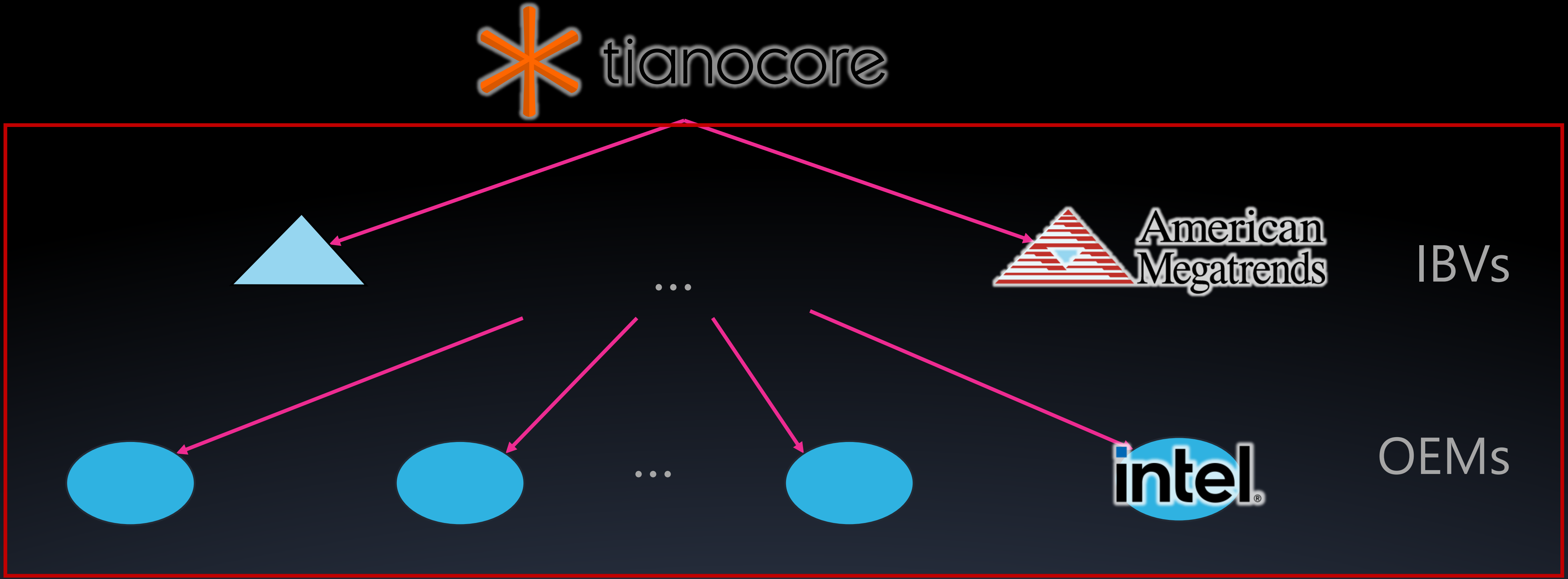
https://edk2-docs.gitbook.io/a-tour-beyond-bios-mitigate-buffer-overflow-in-ue/summary/policy_control

Photo by Andrey Tikhonovskiy on Unsplash

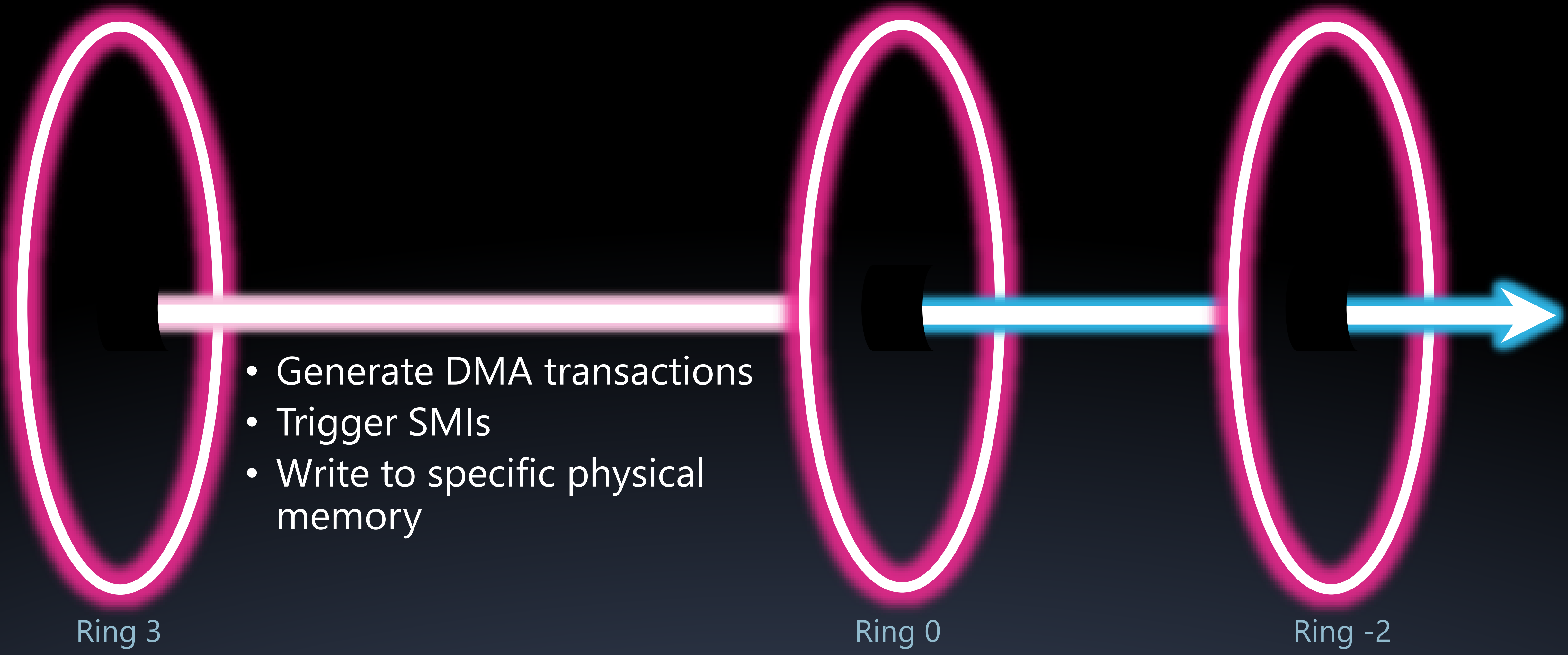
photo by Sivani Bandaru on Unsplash

The demo Gods have forsaken us

The FW Ecosystem



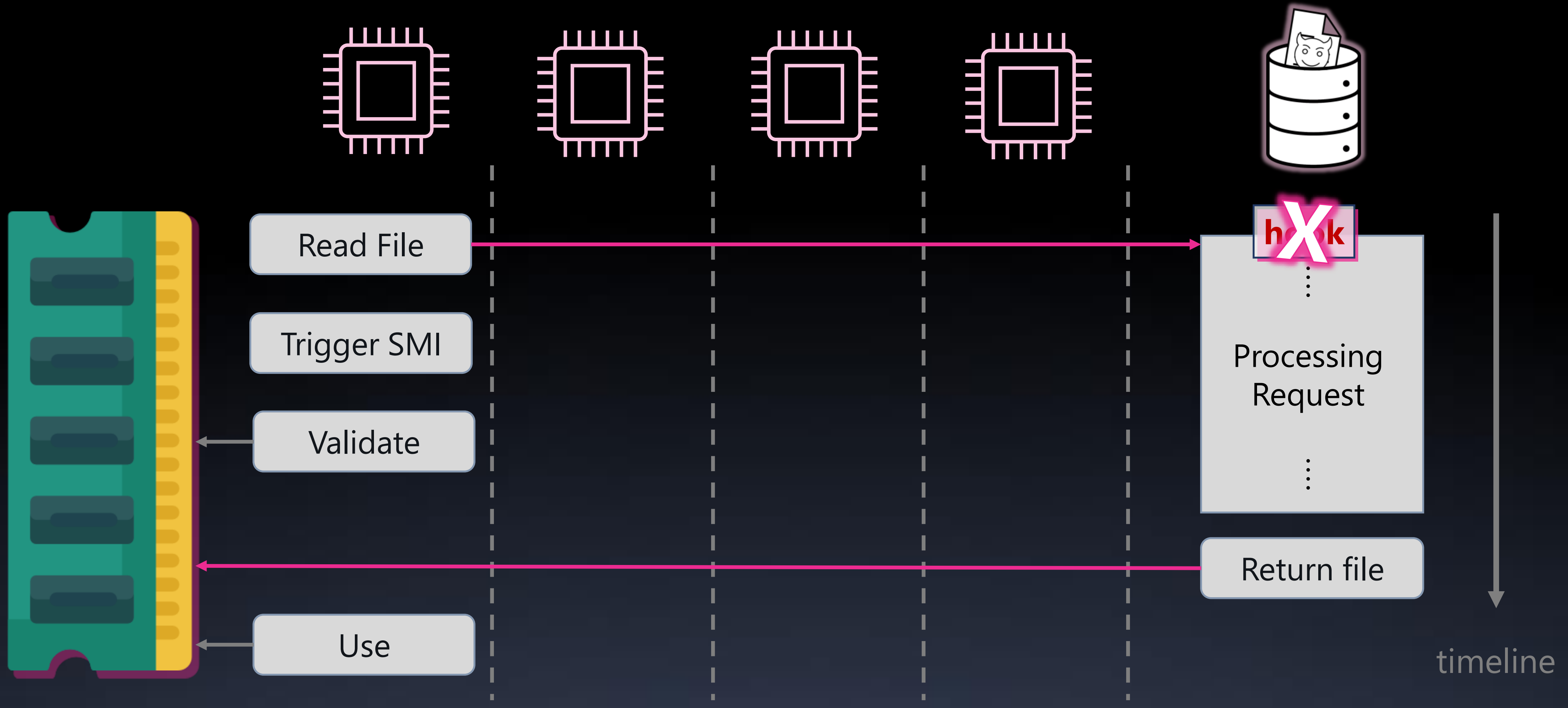
> 200 million devices manufactured in 2020 only

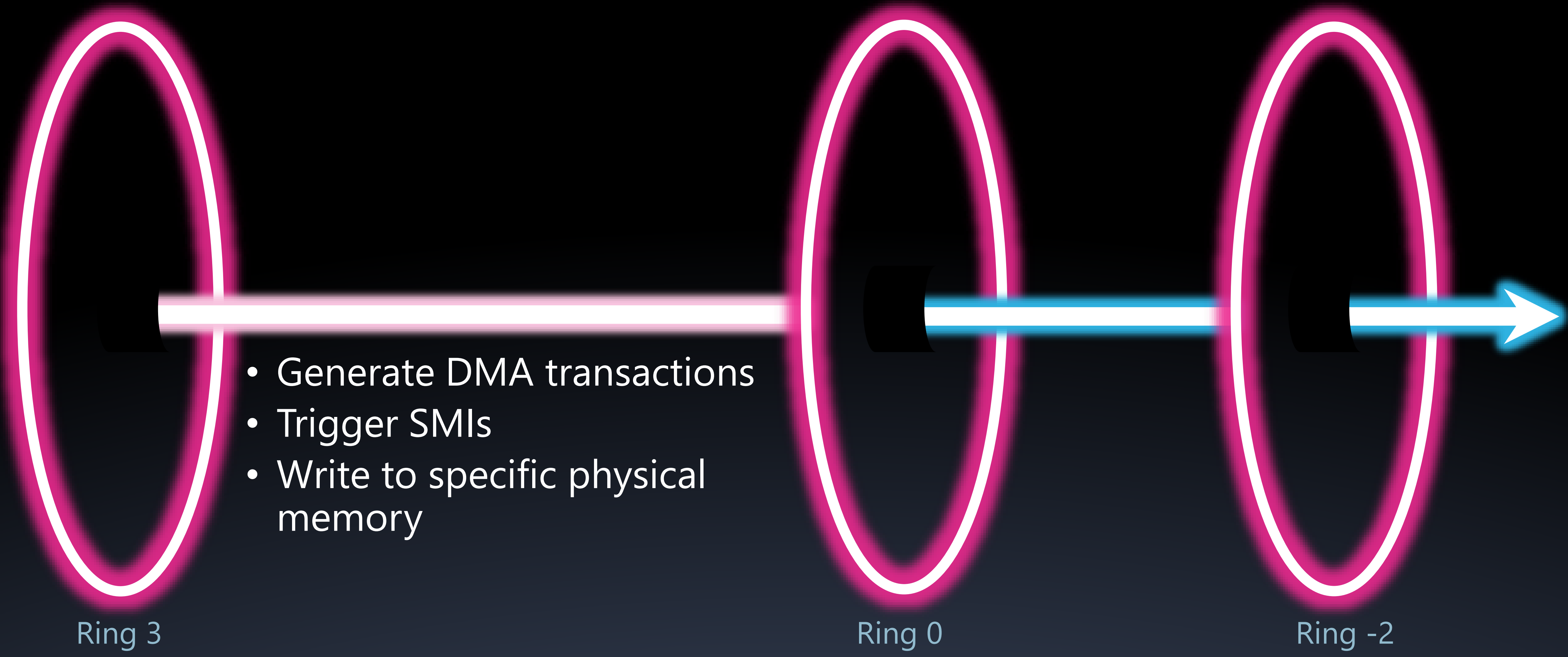


- Generate DMA transactions
- Trigger SMIs
- Write to specific physical memory

Exploitation from ring 3

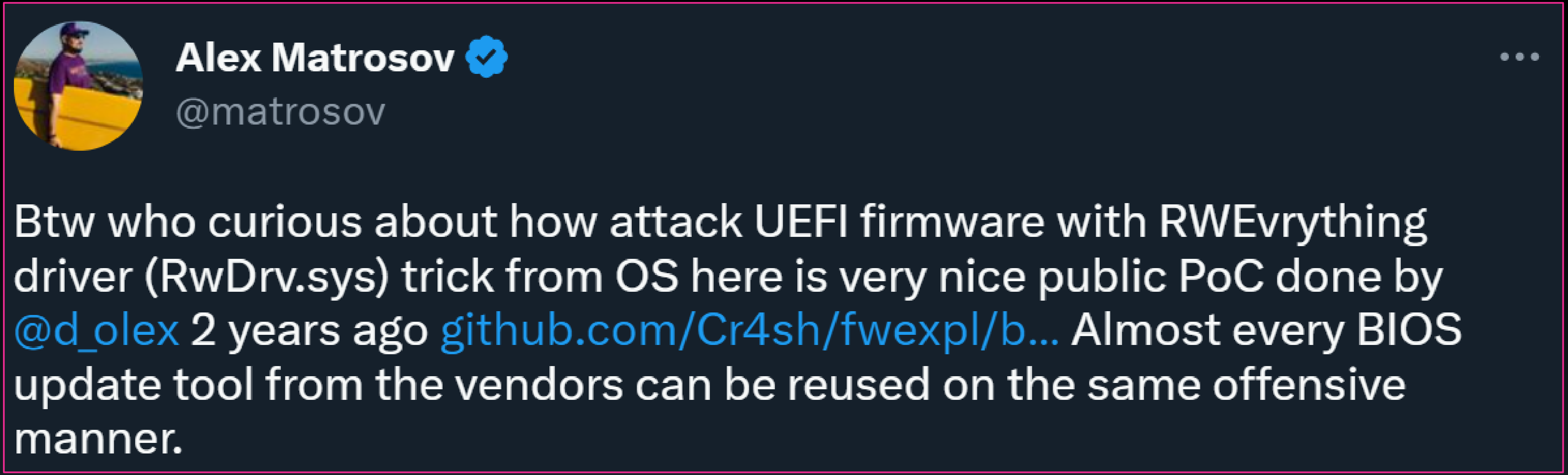
Generating DMA transactions





- Generate DMA transactions
- Trigger SMIs
- Write to specific physical memory

Exploitation from ring 3



Alex Matrosov

@matrosov

Btw who curious about how attack UEFI firmware with RWEvrything driver (RwDrv.sys) trick from OS here is very nice public PoC done by @d_olex 2 years ago github.com/Cr4sh/fwexpl/b... Almost every BIOS update tool from the vendors can be reused on the same offensive manner.

https://twitter.com/matrosov/status/1045922881677352961

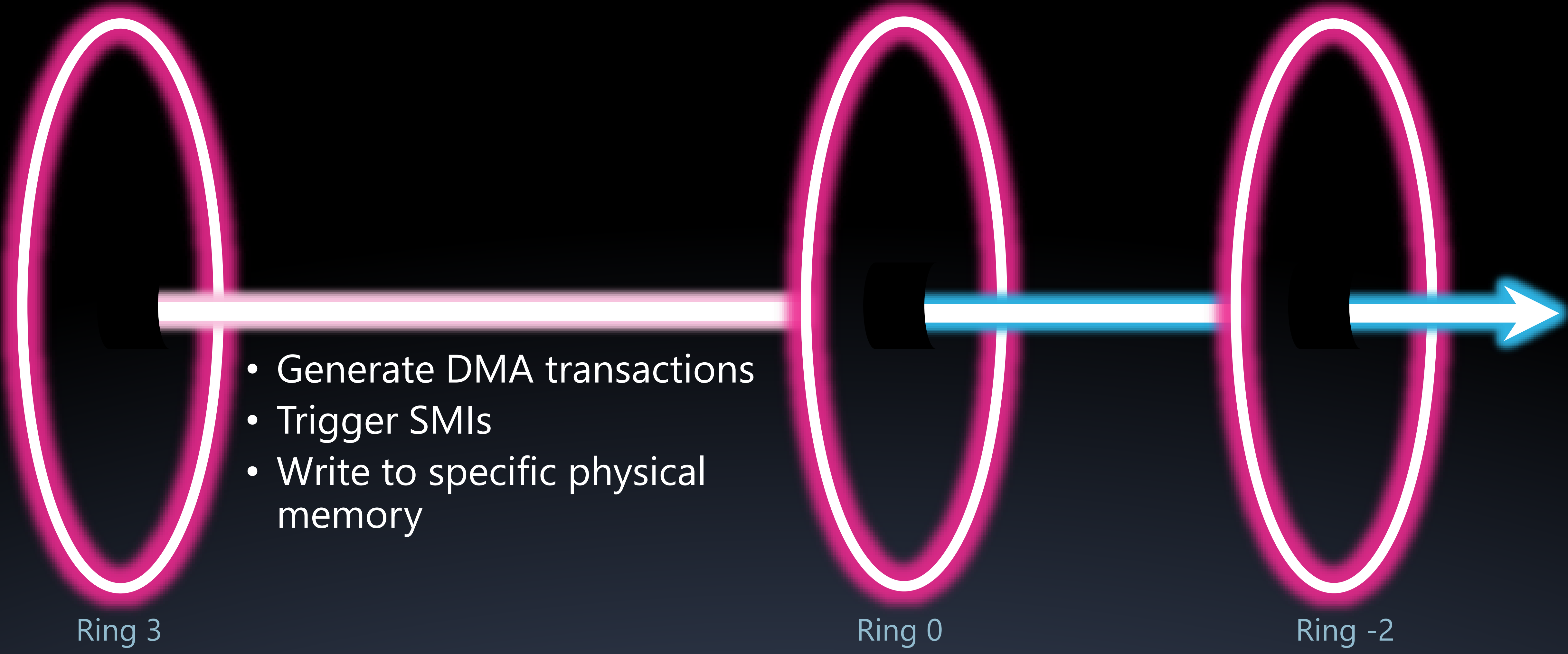
Exploitation from ring 3

Triggering SMI

AMI provides:

- A Linux driver (amifldrv_mod)
- A signed Windows driver (amifldrv64.sys)

Both drivers expose APIs for triggering any SMI



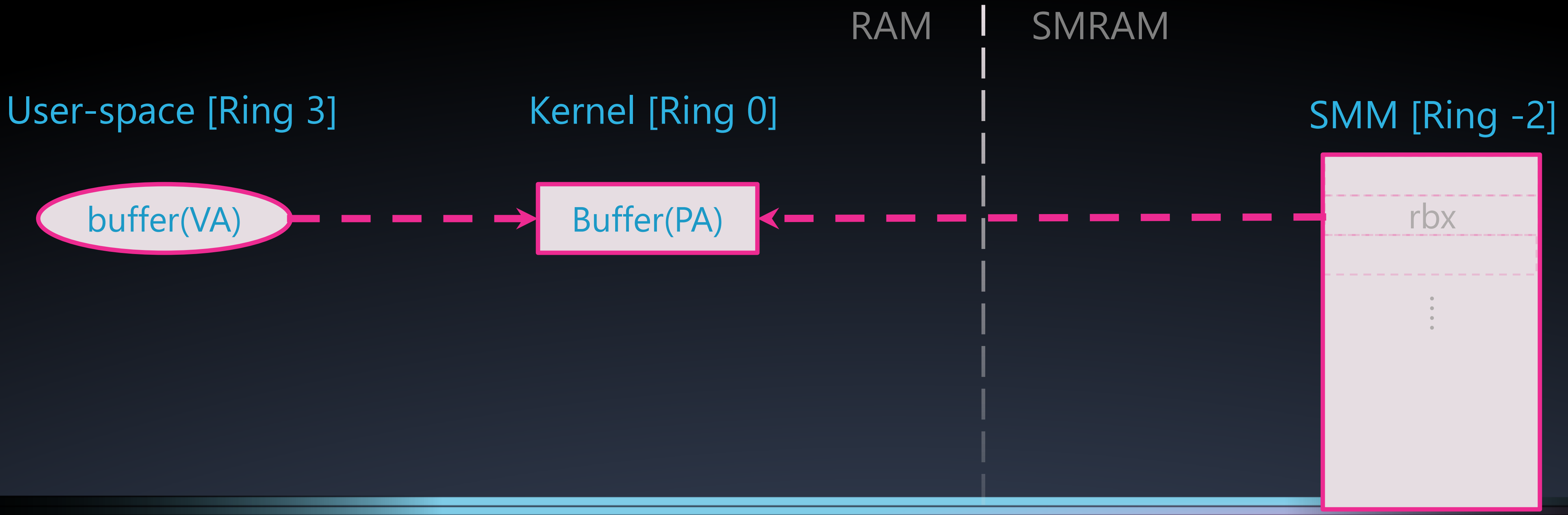
- Generate DMA transactions
- Trigger SMIs
- Write to specific physical memory

Exploitation from ring 3

Writing to physical memory

Communication with SMM done via special buffer in non-SMRAM memory

The drivers create a physical ⇔ virtual mapping of this buffer



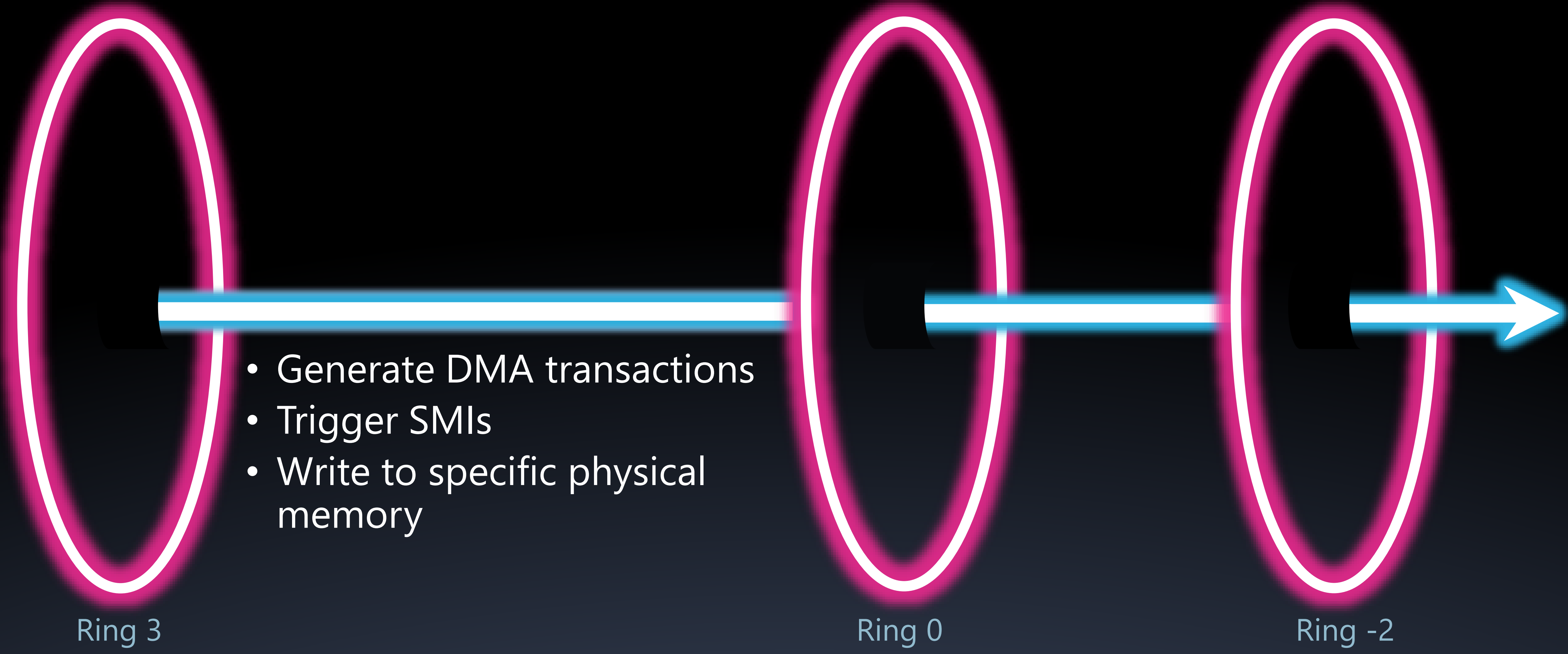
Exploitation from ring 3

Code execution

1. Map a non-SMRAM buffer to a user-space address
2. Perform simultaneously in a loop:

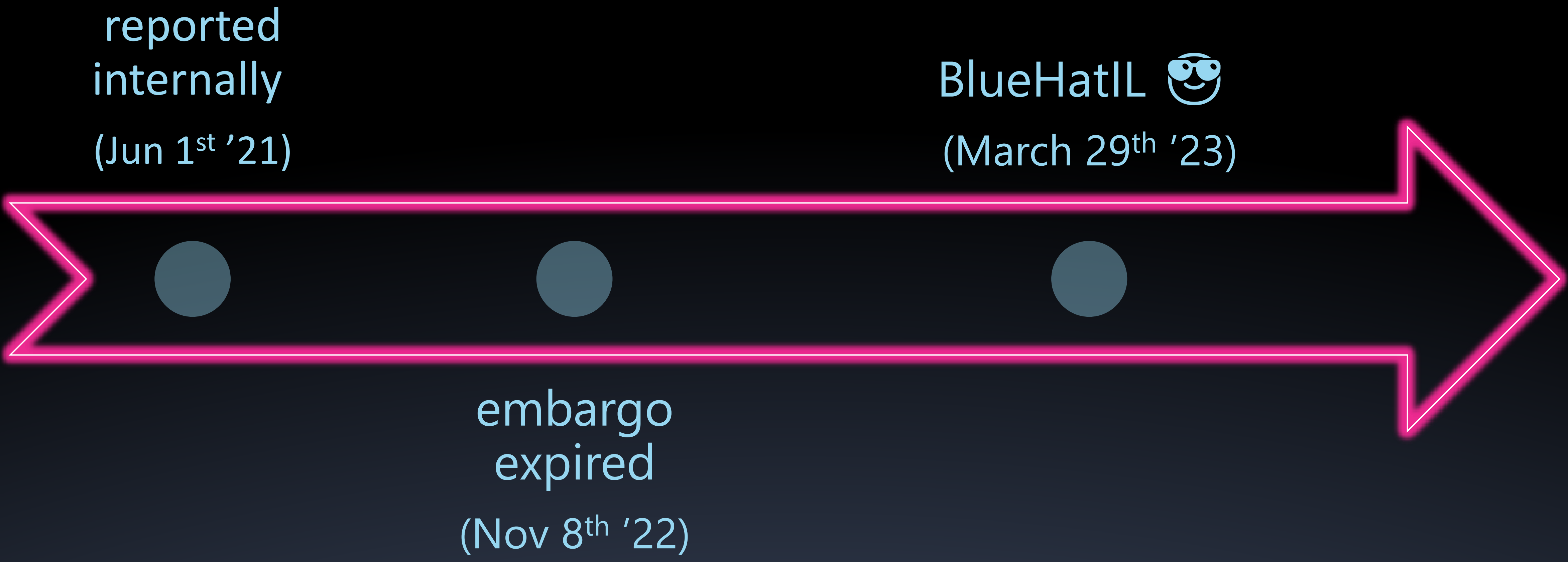
Trigger SMI
with provided buffer
as input

Read “malicious” file
into buffer



- Generate DMA transactions
- Trigger SMIs
- Write to specific physical memory

Timeline



reported internally

(Jun 1st ’21)

embargo expired

(Nov 8th ’22)

BlueHatIL 😎

(March 29th ’23)

Key Takeaways

- UEFI threats are real
- SMI handlers compose a fruitful attack surface
- UEFI research has an interesting future

Key Takeaways

- UEFI threats are real
- SMI handlers compose a fruitful attack surface
- UEFI research has an interesting future – stay tuned

Thank you

grasshopper photo by Eka P. Amdela on Unsplash