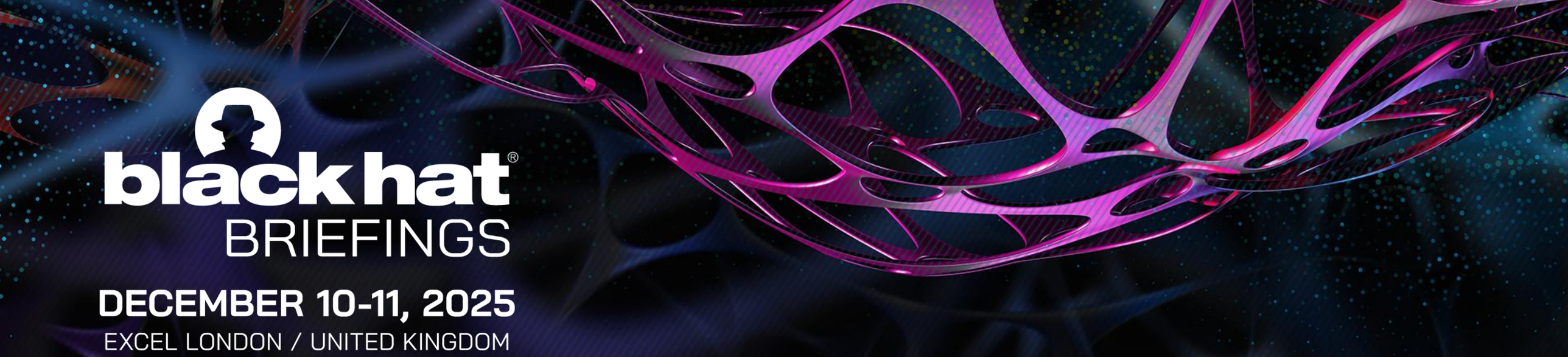

black hat®
BRIEFINGS
DECEMBER 10-11, 2025
EXCEL LONDON / UNITED KINGDOM
#BHEU @BlackHatEvents



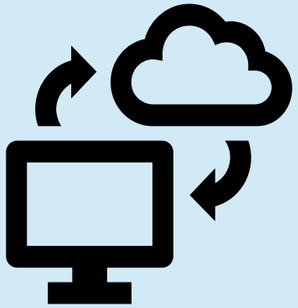
DECEMBER 10-11, 2025
EXCEL LONDON / UNITED KINGDOM

RMPocalypse: A Catch-22 Breaking AMDs Confidential Computing

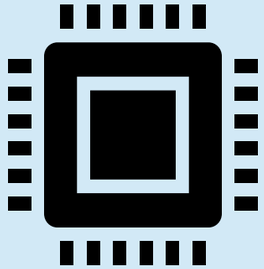
Benedict Schlüter, Shweta Shinde (ETH Zurich)

Attack Surfaces for Confidential Data

At rest

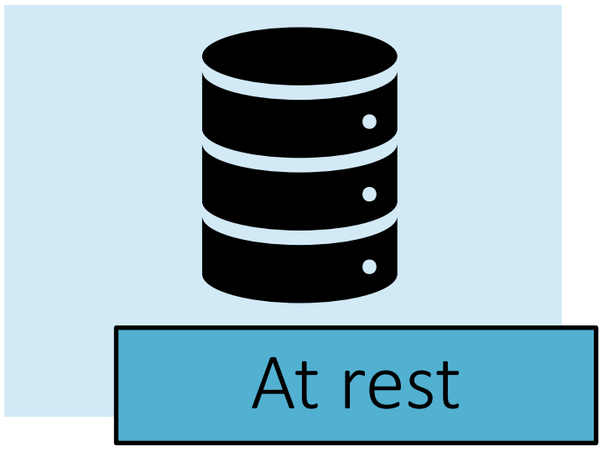


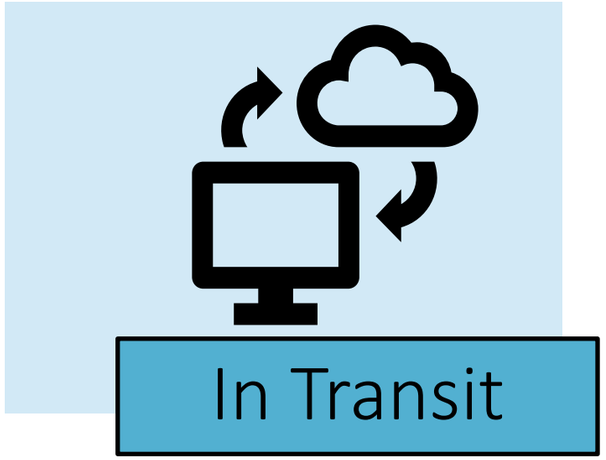
In Transit

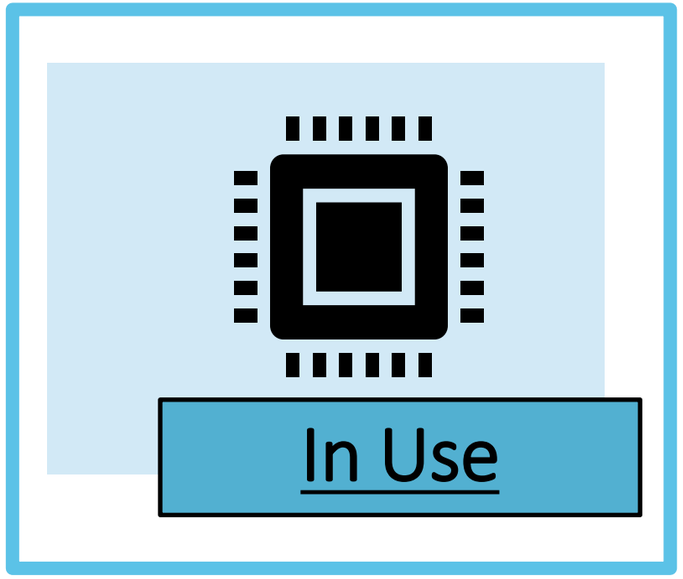


In Use

Attack Surfaces for Confidential Data







Sensitive Apps

Servers (e.g. CPUs and GPUs)

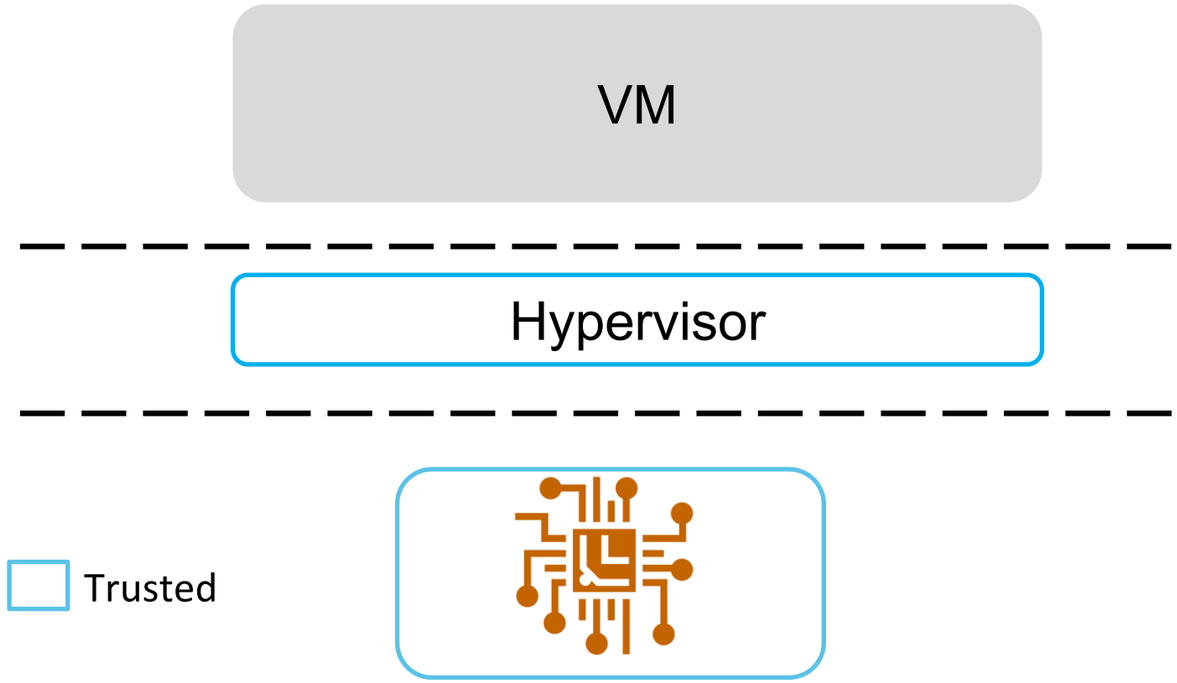
Sensitive Apps

Cloud Providers (e.g. Operating Systems)

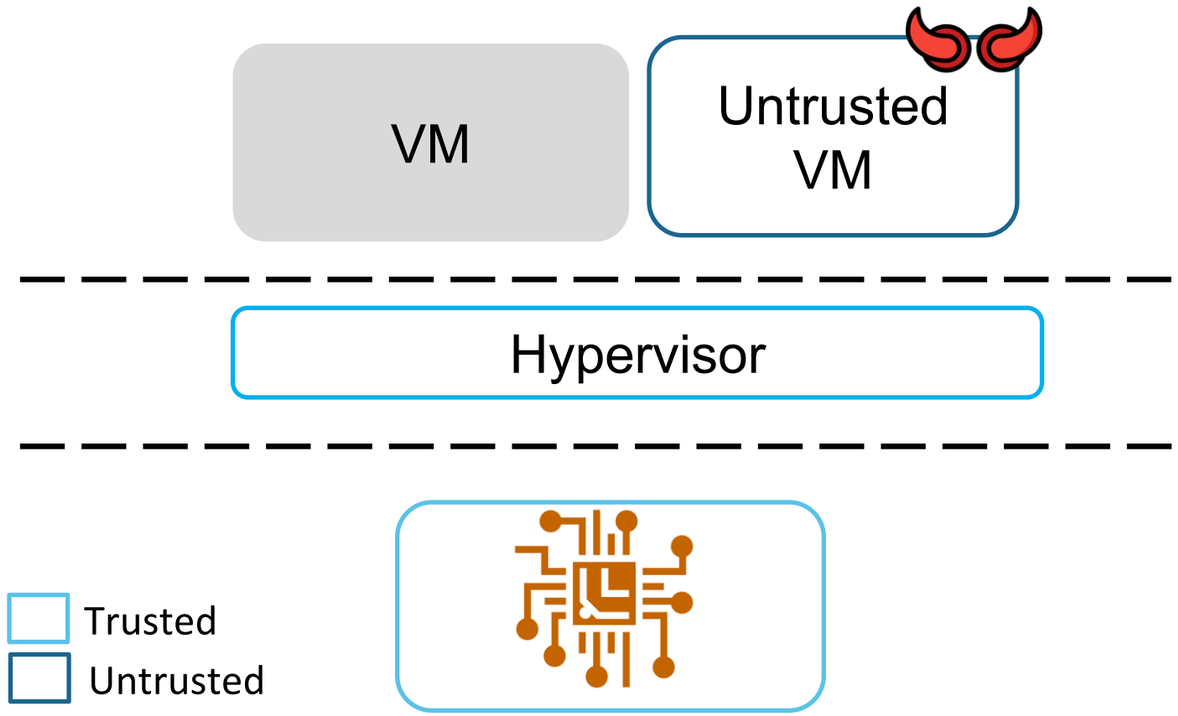
Servers (e.g. CPUs and GPUs)

https://blog.google/technology/ai/google-private-ai-compute/ https://azure.microsoft.com/en-us/solutions/confidential-compute https://aws.amazon.com/confidential-computing/

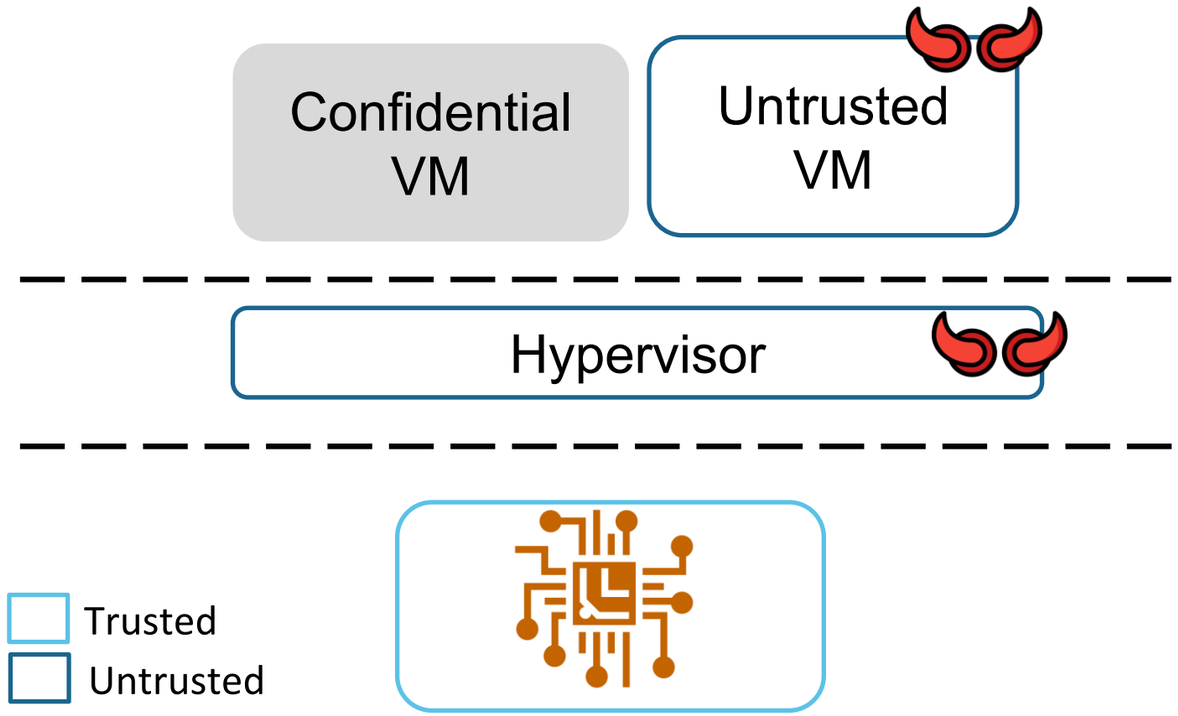
Single tenancy threat model



Multi tenancy threat model (e.g. Cloud)



New Generation CPUs
(e.g., AMD SEV-SNP, Intel TDX, ARM CCA)



Why Look Into AMDs Confidential Computing

Empowering The Industry with Open System Firmware – AMD openSIL

Apr 13, 2023

https://www.amd.com/en/blogs/2023/empowering-the-industry-with-open-system-firmware-.html

Processor Programming Reference (PPR) for AMD Family 1Ah Model 70h, Revision A0 Processors

https://docs.amd.com/v/u/en-US/57930-A0-PUB_3.00

August 30, 2023

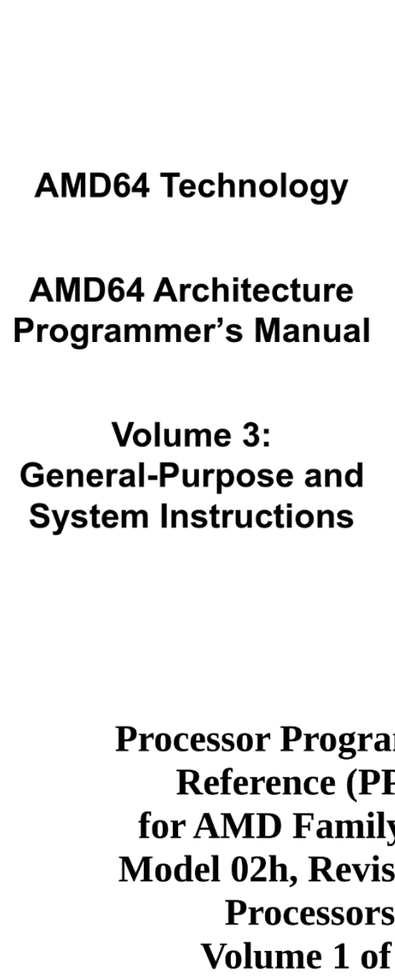
AMD

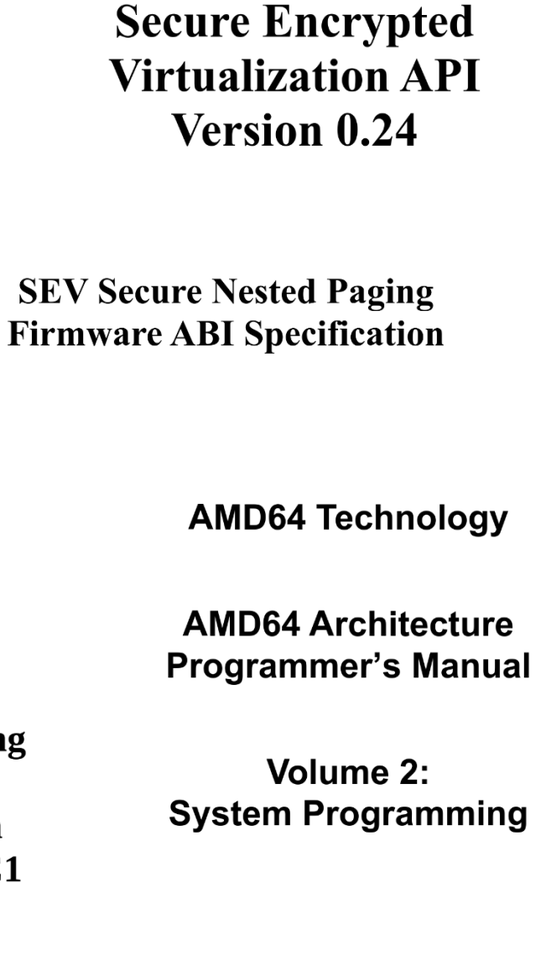
AMD Shares The Technical Details of Technology Powering Innovative Confidential Computing Leadership Cloud Offerings

https://ir.amd.com/news-events/press-releases/detail/1154/amd-shares-the-technical-details-of-technology-powering-innovative-confidential-computing-leadership-cloud-offerings

How It All Started

PPR (ID 57883)
SEV-SNP (ID 70366)
SEV (ID 55766)
BKDG (ID 42301 / 42300)
Programmers Manual (ID 24593)





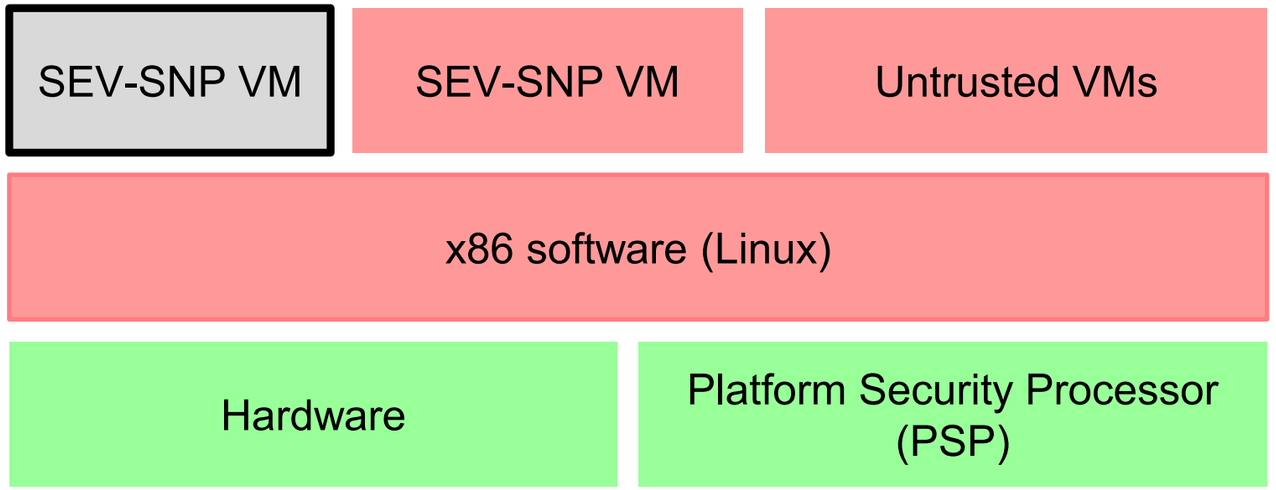
How It All Started

Actions

Before invoking SNP_INIT_EX with INIT_RMP set to 1, software must ensure that no CPUs contain dirty cache lines for the memory containing the RMP.

https://docs.amd.com/v/u/en-US/56860 (Rev 1.57)

SEV-SNP Threat Model

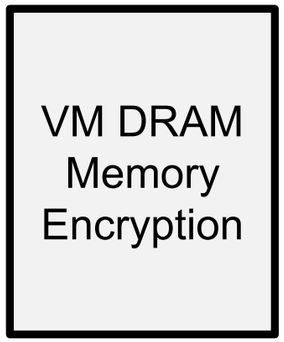


Secure Encrypted Virtual Machine (SEV) History

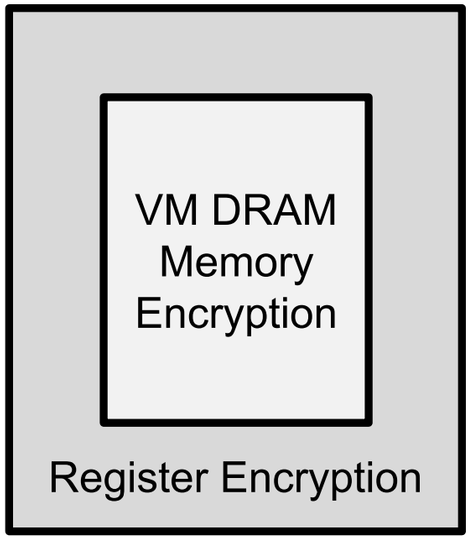
VM DRAM Memory Encryption

SEV

Secure Encrypted Virtual Machine (SEV) History



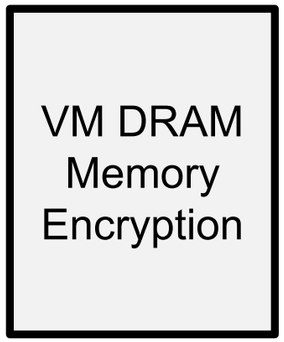
SEV



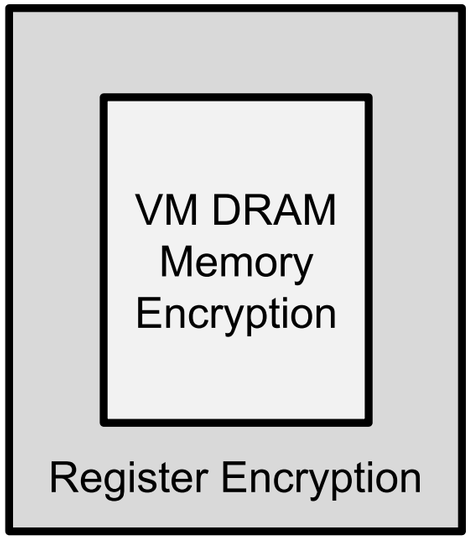
SEV-ES

Encrypted State

Secure Encrypted Virtual Machine (SEV) History

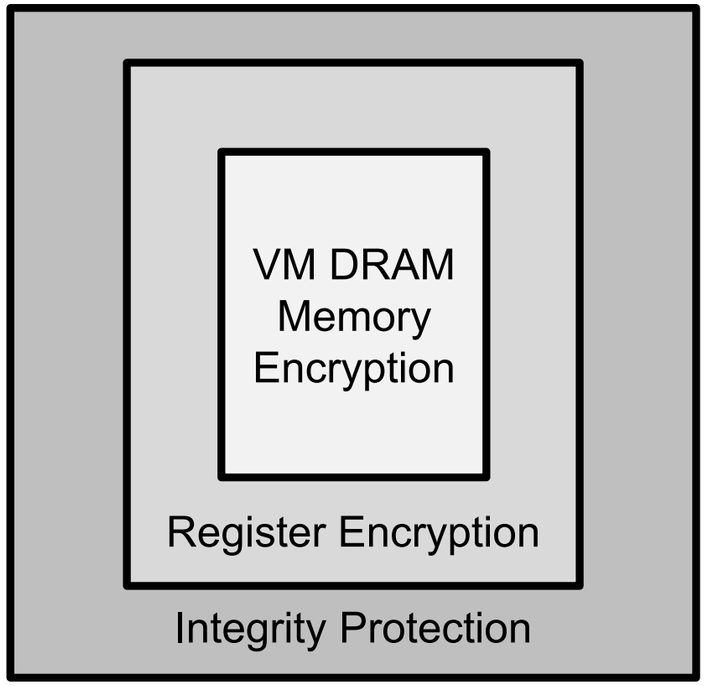


SEV



SEV-ES

Encrypted State



SEV-SNP

Secure Nested Paging

How SEV-SNP Ensures Integrity?

Reverse Map Table (RMP)

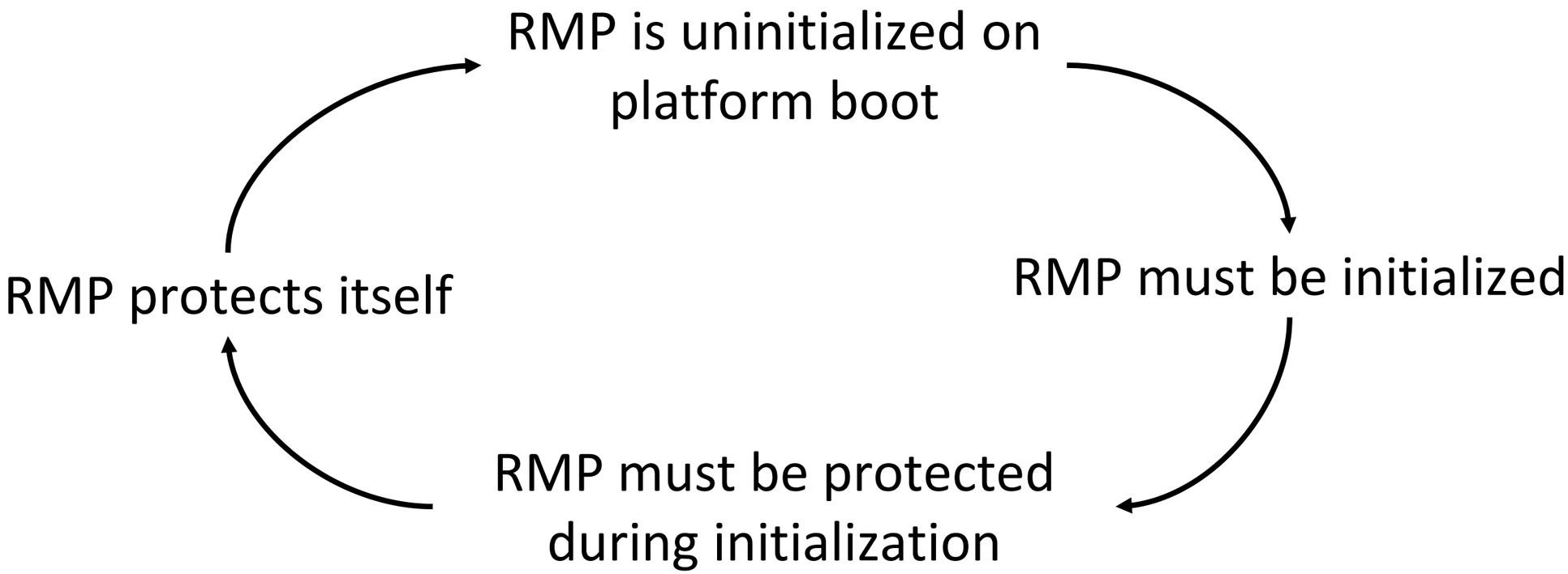
How SEV-SNP Ensures Integrity?

The RMP is a table DRAM holding metadata information for every page in DRAM, including itself.

The RMP is uninitialized on system boot.

Physical Page	Hypervisor Write	Guest ID	Guest Physical Address	Reserved for Firmware
0x0000	False	0x5	0x4000	False
0x1000	True	0x0	0x0	False
0x2000	True	0x0	0x0	False
0x3000	False	0x7	0xFA000	False
…	…	…	…	…
0x80000000	True	0x0	0x0	False

The Catch-22: RMP protects the RMP



SEV-SNP Initialization

x86 CPU running Linux

AMDs Platform Security Processor (PSP)

Prepare system for SEV-SNP init and call PSP

SEV-SNP Initialization

x86 CPU running Linux	AMDs Platform Security Processor (PSP)
Prepare system for SEV-SNP init and call PSP	
	Initialize SEV-SNP and write secure RMP configuration

Exploitation Target

RMP ensures integrity in SEV-SNP

Exploitation Target

RMP ensures integrity in SEV-SNP

Compromise RMP to break SEV-SNP

How It All Started (Reminder)

Actions

Before invoking SNP_INIT_EX with INIT_RMP set to 1, software must ensure that no CPUs contain dirty cache lines for the memory containing the RMP.

https://docs.amd.com/v/u/en-US/56860 (Rev 1.57)

SEV-SNP Initialization

x86 core #0

```
do_sev_snp_init();
pr_info(“RMP[0] %llx\n, RMPTABLE[0]);
```

SEV-SNP Initialization

x86 core #0

```
do_sev_snp_init();
pr_info(“RMP[0] %llx\n, RMPTABLE[0]);
```

```
RMP[0] 0x0
```

First Experiment

x86 core #0

```
WRITE_ONCE(RMPTABLE[0],0xDEADBEEF);
do_sev_snp_init();
pr_info(“RMP[0] %llx\n, RMPTABLE[0]);
```

Let’s violate the statement from the SEV-SNP ABI specification by creating a dirty cacheline.

First Experiment

x86 core #0

```
WRITE_ONCE(RMPTABLE[0],0xDEADBEEF);
do_sev_snp_init();
pr_info(“RMP[0] %llx\n, RMPTABLE[0]);
```

```
RMP[0] 0x0
```

First Experiment

x86 core #0

```
WRITE_ONCE(RMPTABLE[0],0xDEADBEEF);
do_sev_snp_init();
pr_info("RMP[0] %llx\n, RMPTABLE[0]);
```

Failed overwrite

```
RMP[0] 0x0
```

First Experiment

x86 core #0

```
WRITE_ONCE(RMPTABLE[0],0xDEADBEEF);
do_sev_snp_init();
/*
 * ‘huge’ delay between the actual
 * RMP write from the PSP
 */
pr_info(“RMP[0] %llx\n, RMPTABLE[0]);
```

Failed overwrite

```
RMP[0] 0x0
```

Second Experiment

x86 core #0

```
do_sev_snp_init()
pr_info("RMP[0] 0x%llx\n, RMPTABLE[0]);
```

x86 core #1

```
while (!kthread_should_stop())
   WRITE_ONCE(RMPTABLE[0],0xDEADBEEF);
```

Second Experiment

x86 core #0	x86 core #1
`do_sev_snp_init()` `pr_info("RMP[0] 0x%llx\n, RMPTABLE[0]);`	`while (!kthread_should_stop())` `  WRITE_ONCE(RMPTABLE[0],0xDEADBEEF);`

Successful overwrite

```
RMP[0] 0xDEADBEEF
```

Initial Root Cause Analysis

Third Experiment

x86 core #0

```
do_sev_snp_init()
pr_info("RMP[0] 0x%llx\n, RMPTABLE[0]);
```

x86 core #1

```
RMPTABLE = map_as_uncachable(RMPTABLE)
while (!kthread_should_stop())
   WRITE_ONCE(RMPTABLE[0],0xDEADBEEF);
```

Third Experiment

x86 core #0

```
do_sev_snp_init()
pr_info("RMP[0] 0x%llx\n, RMPTABLE[0]);
```

x86 core #1

```
RMPTABLE = map_as_uncachable(RMPTABLE)
while (!kthread_should_stop())
   WRITE_ONCE(RMPTABLE[0],0xDEADBEEF);
```

Failed overwrite

```
RMP[0] 0x0
```

PSP Source Code

```
/* SEV Firmware code */

enable_MP_RMP_protection(…);
enable_DRAM_TMR_protection(…);

/* do RMP/SNP initialization */

enable_SEV-SNP_globally(…);
```

https://github.com/benschlueter/AMD-ASPFW/blob/3ca6650dd35d878b3fcbe5c7f58b145eed042bbf/fw/psp_bl_uapps/sev_uapp/src/sev_mcmd.c#L4012

Reminder

Actions

Before invoking SNP_INIT_EX with INIT_RMP set to 1, software must ensure that no CPUs contain dirty cache lines for the memory containing the RMP.

Attack fails if we map pages as **uncachable** on Linux

PSP source code indicates RMP protection during SEV-SNP initialization

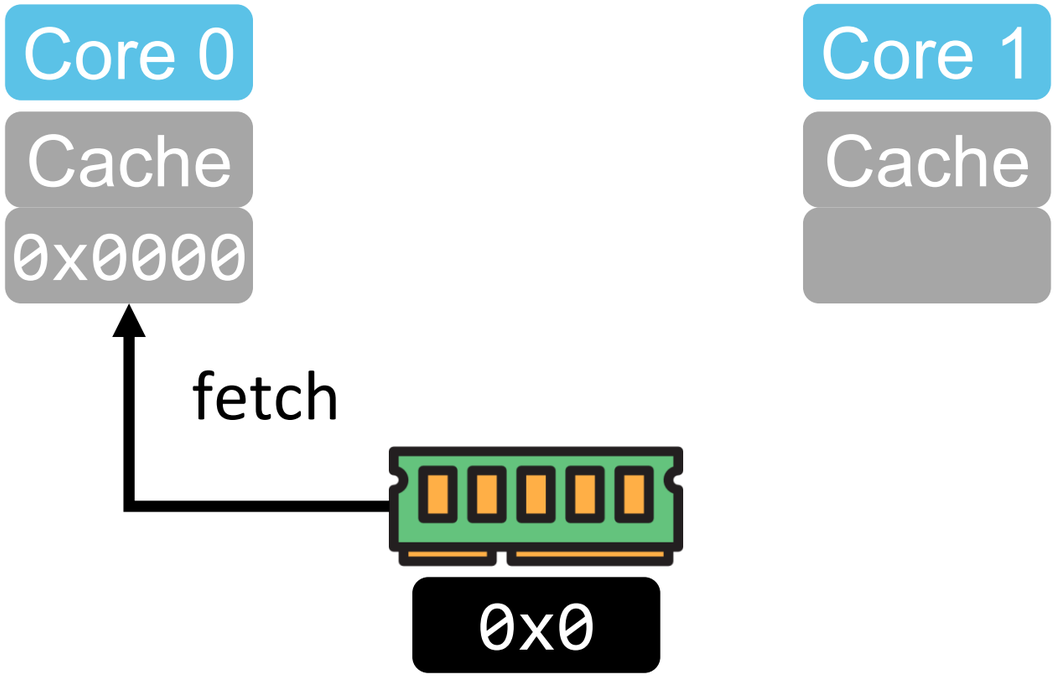
https://docs.amd.com/v/u/en-US/56860 (Rev 1.57)

Initial Hypothesis

It seems like a cache coherency vulnerability!

Cache Coherency

Core 0 fetches memory into cache

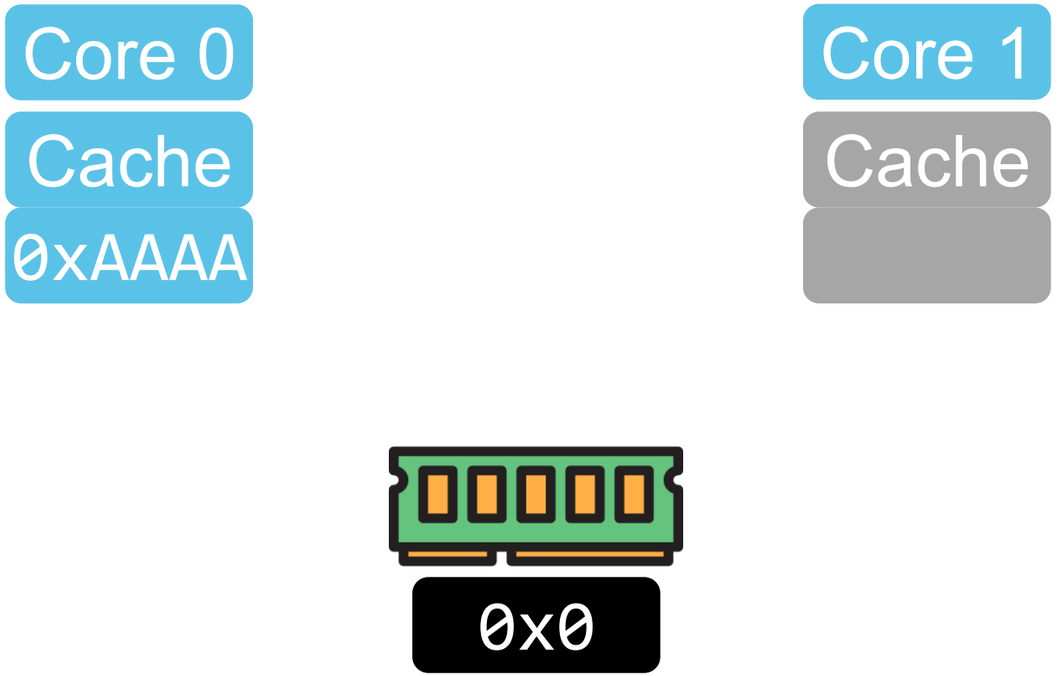


clean cacheline

dirty cacheline

Cache Coherency

Core 0 writes to memory (cache)

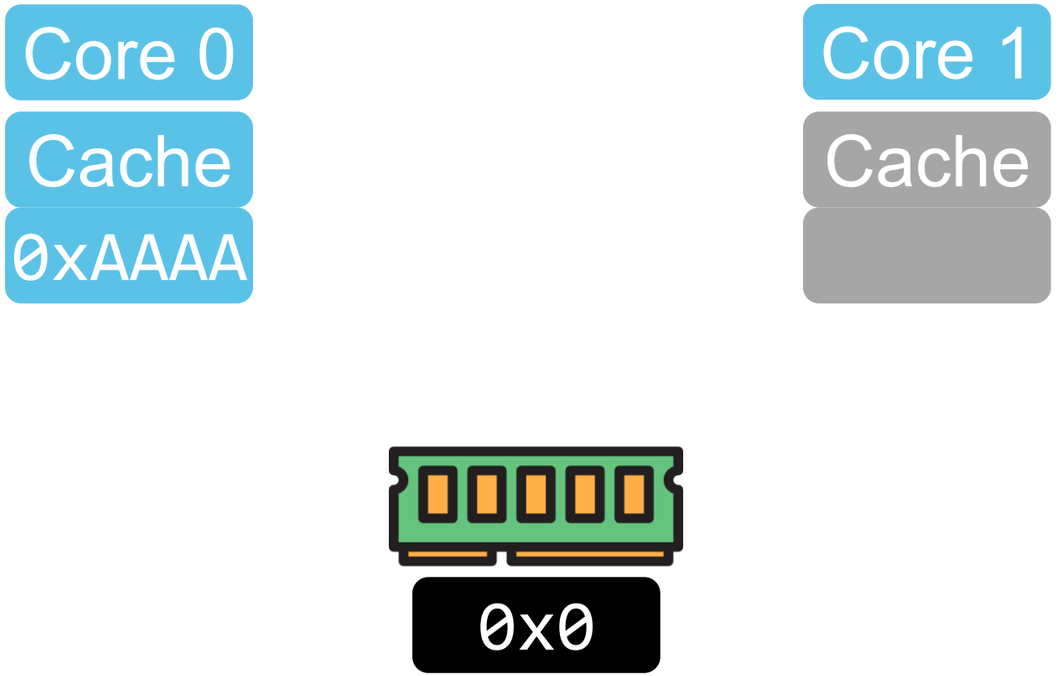


clean cacheline

dirty cacheline

Cache Coherency

Core 1 wants to write to the same memory

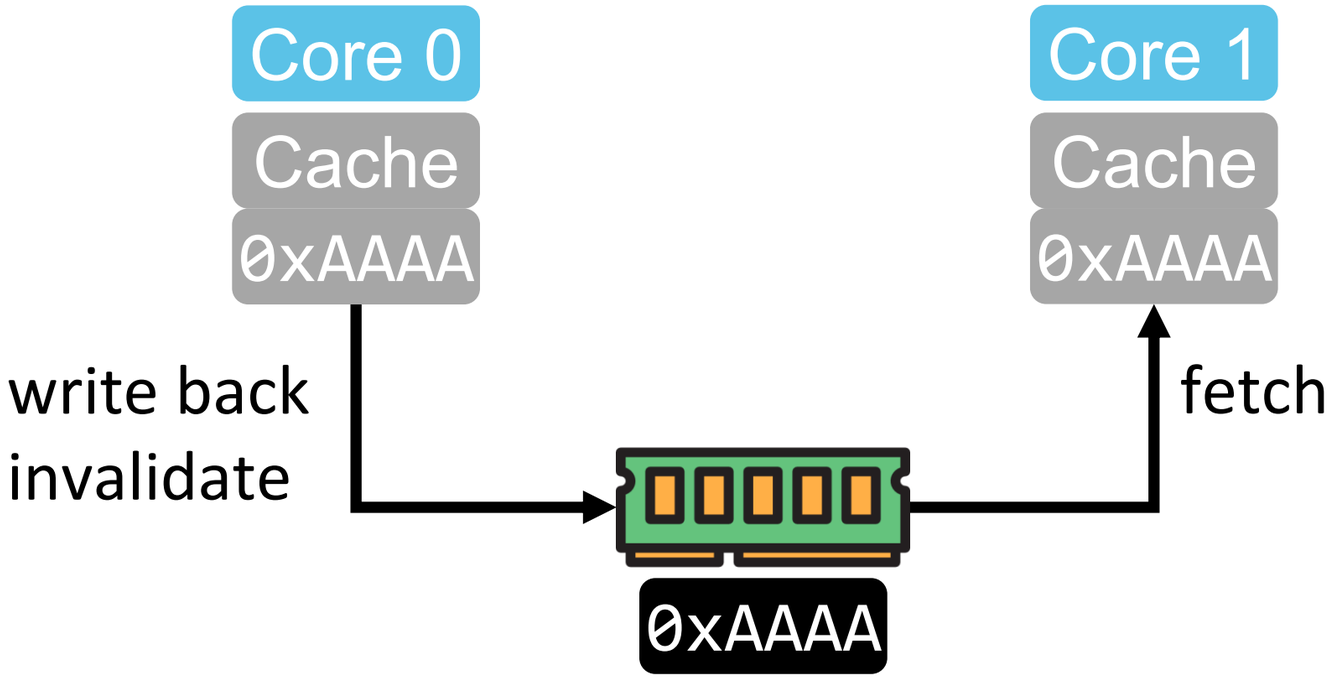


clean cacheline

dirty cacheline

Cache Coherency

Core 1 invalidates dirty line in Core 0 caches and fetches memory to its cache

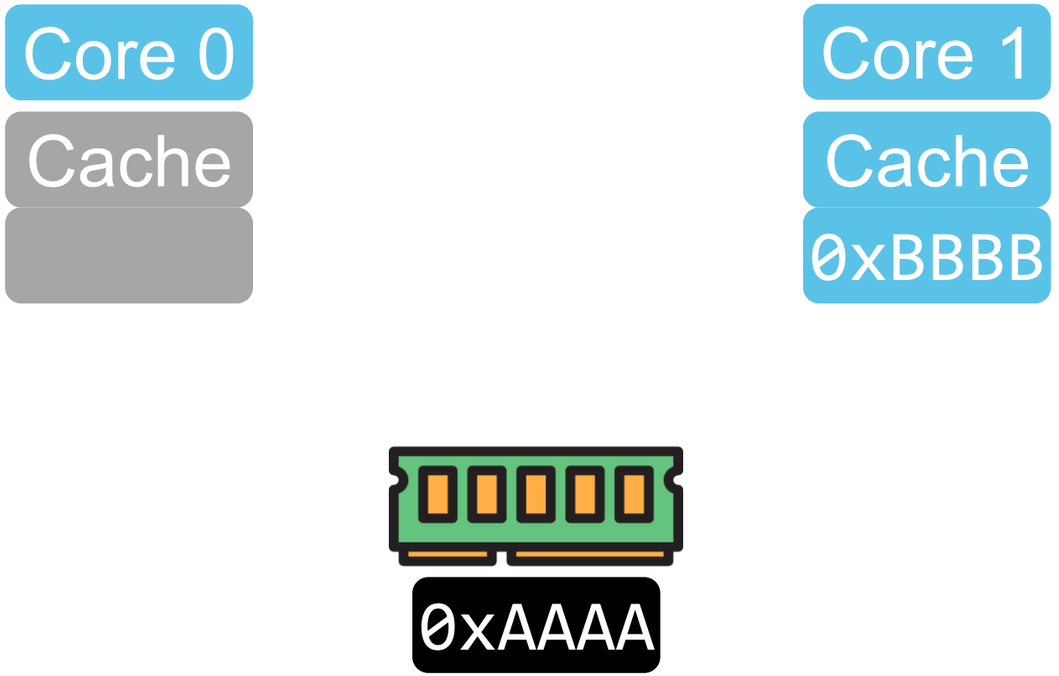


clean cacheline

dirty cacheline

Cache Coherency

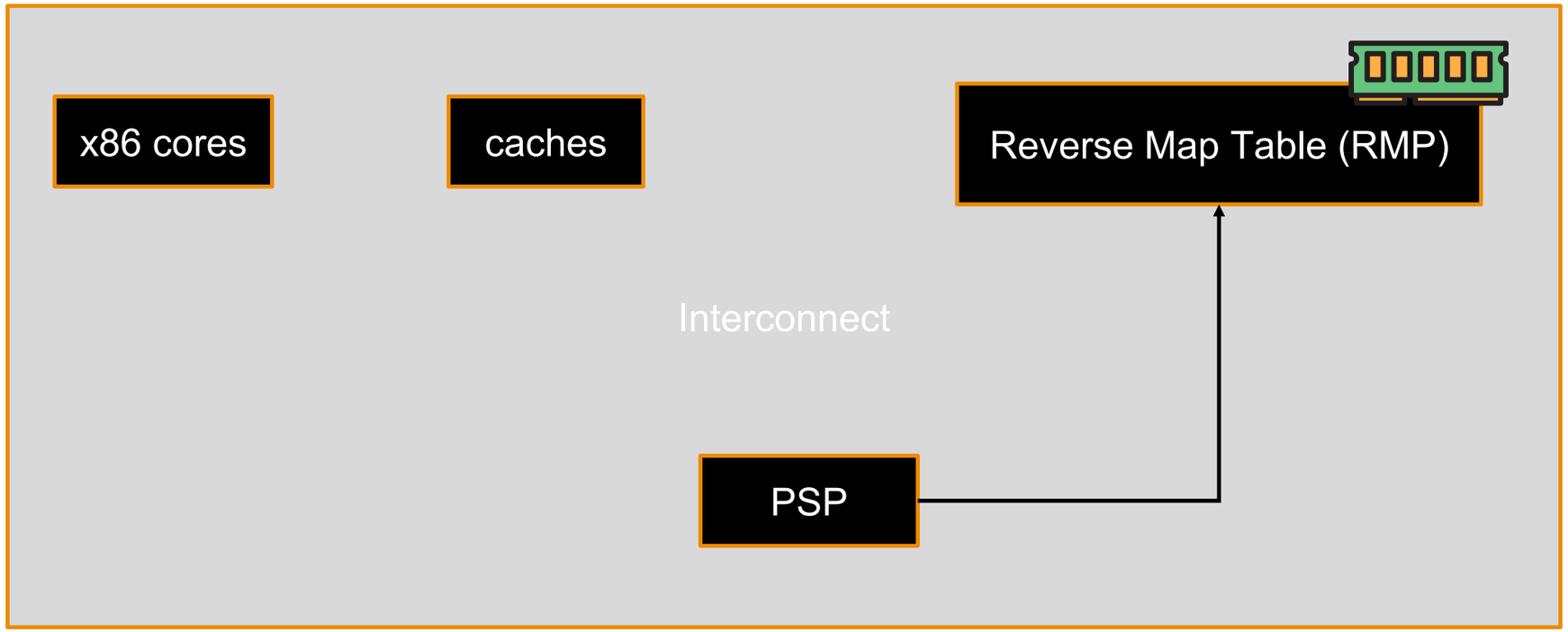
Core 1 writes to the memory (cache)



clean cacheline

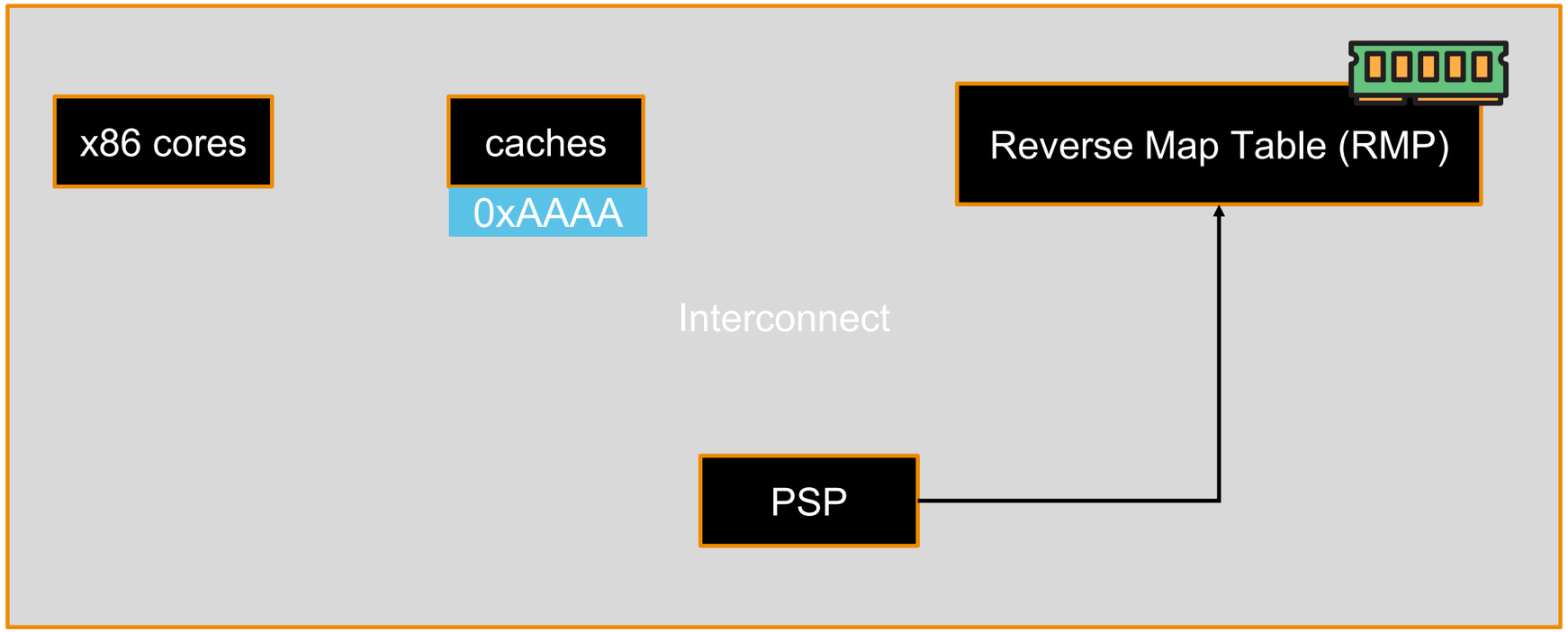
dirty cacheline

Initial Hypothesis



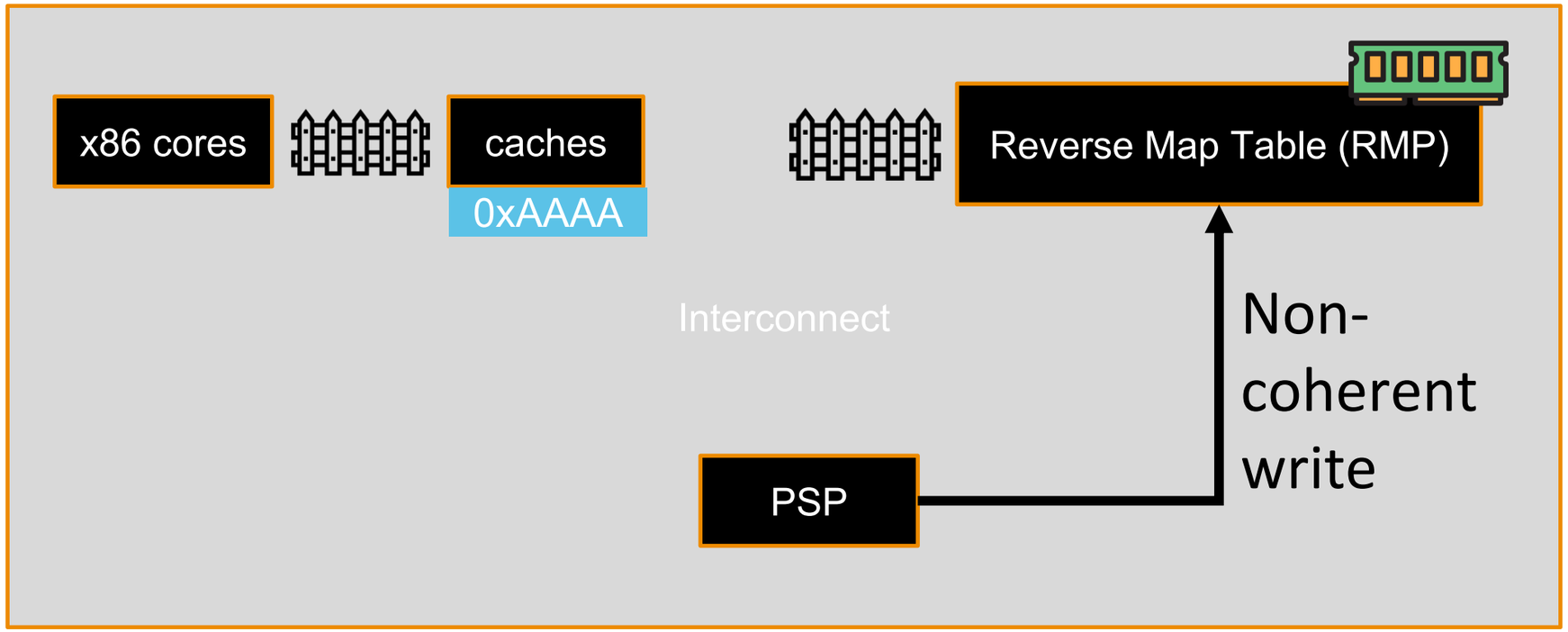
Initial Hypothesis

x86 creates dirty cache entry pointing to RMP DRAM



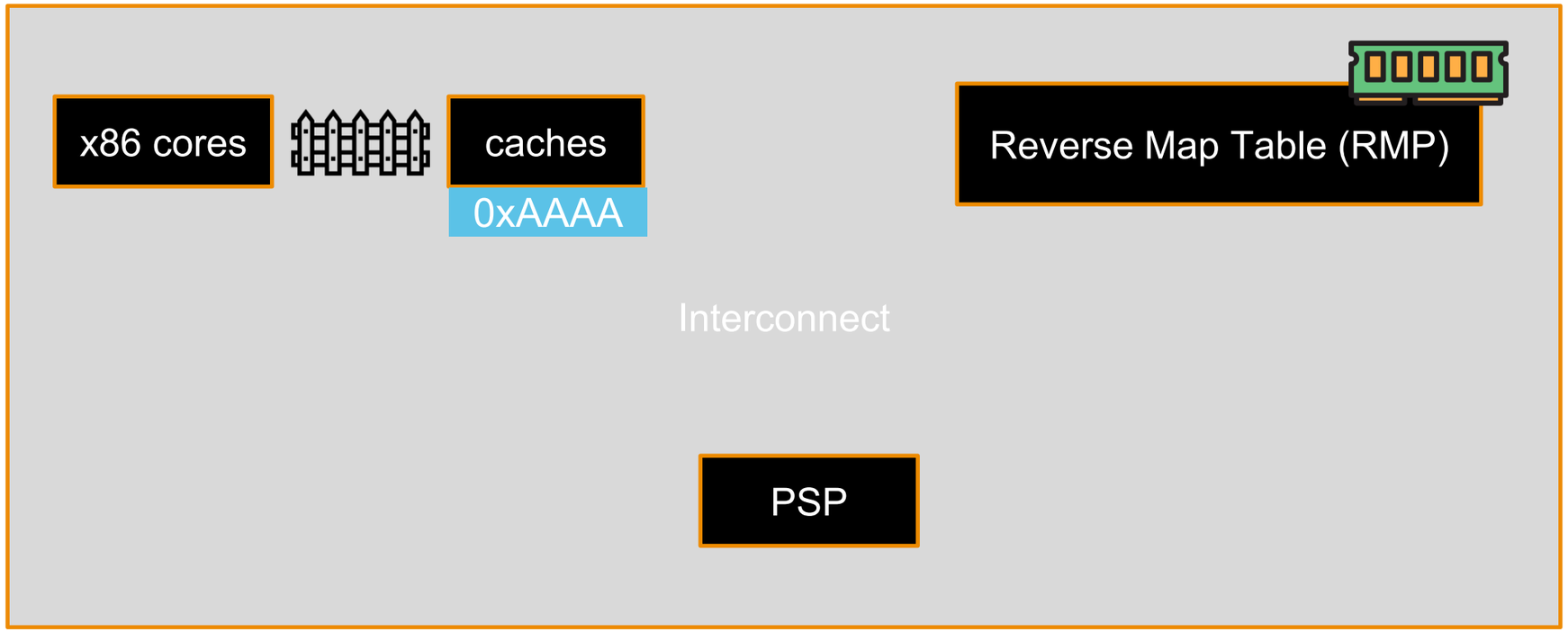
Initial Hypothesis

PSP installs barriers and writes to RMP non-coherent



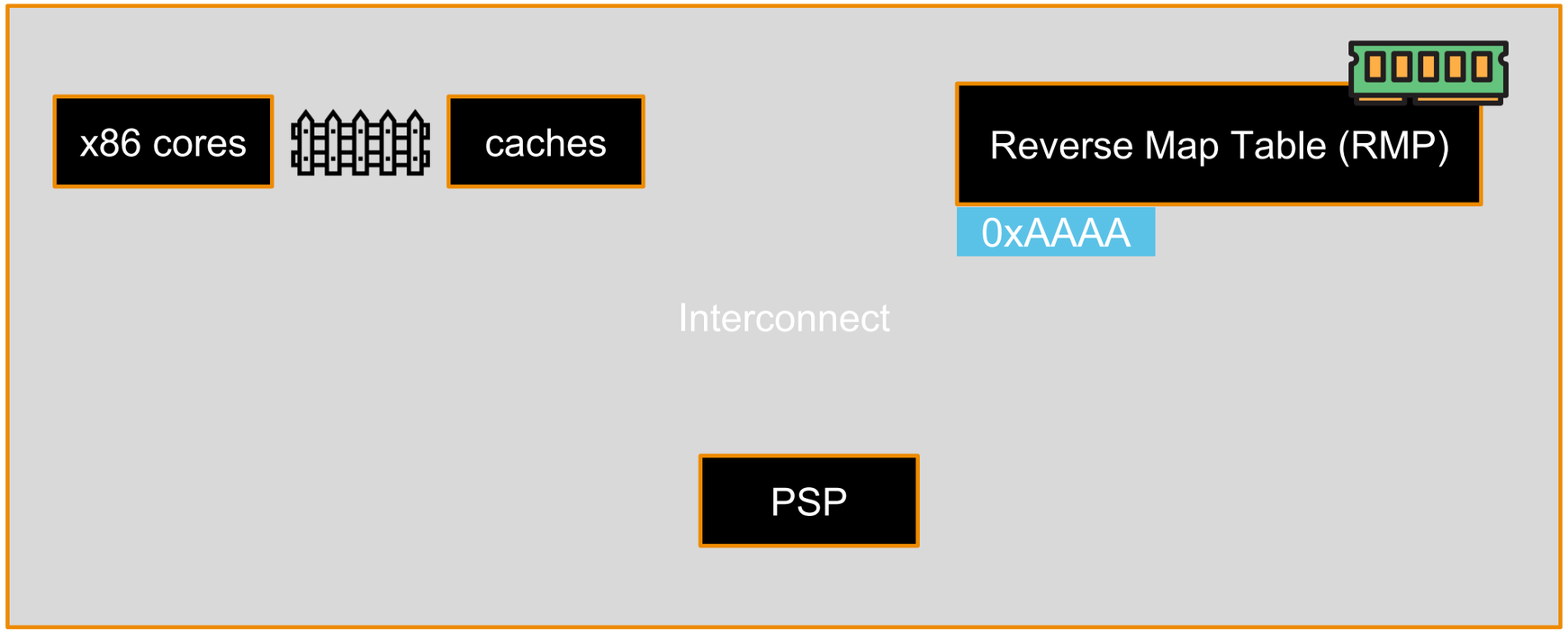
Initial Hypothesis

Overwrite RMP with dirty cachelines after RMP initialization



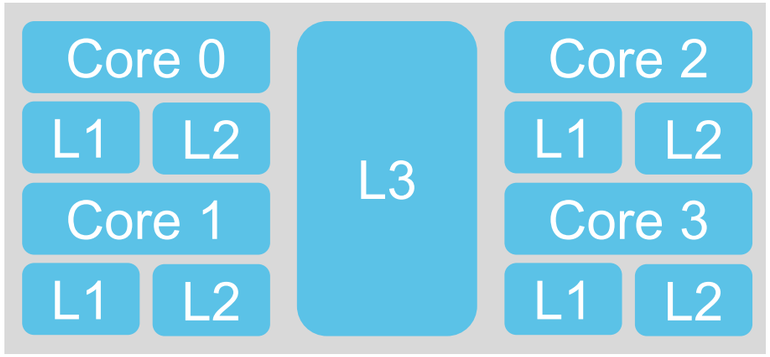
Initial Hypothesis

Overwrite RMP with dirty cachelines after RMP initialization



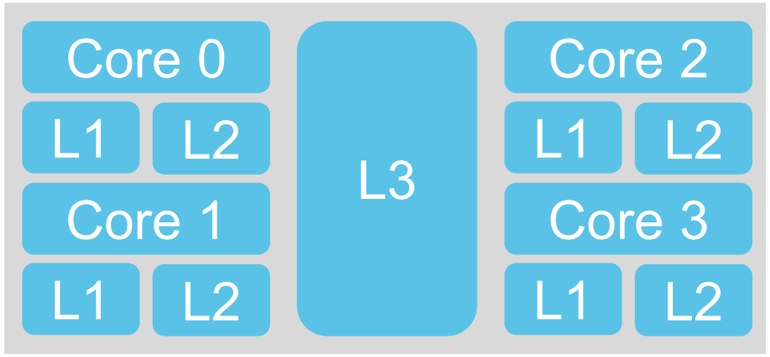
Root Cause Analysis Continued

More Background



CCX: Core Compute Complex

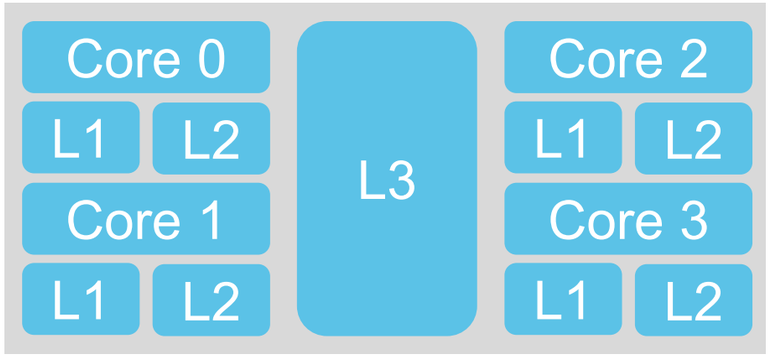
More Background



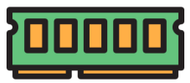
Data Fabric connects different components

Data Fabric

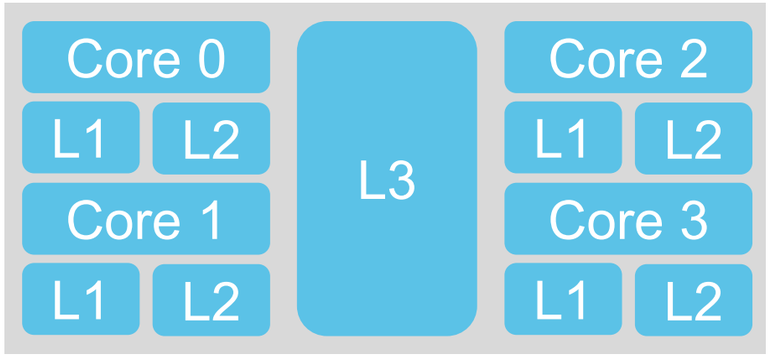
More Background



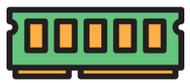
CCM: Core Coherent Master



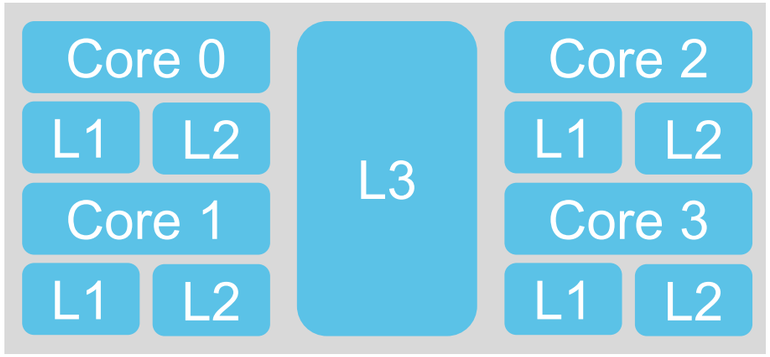
More Background



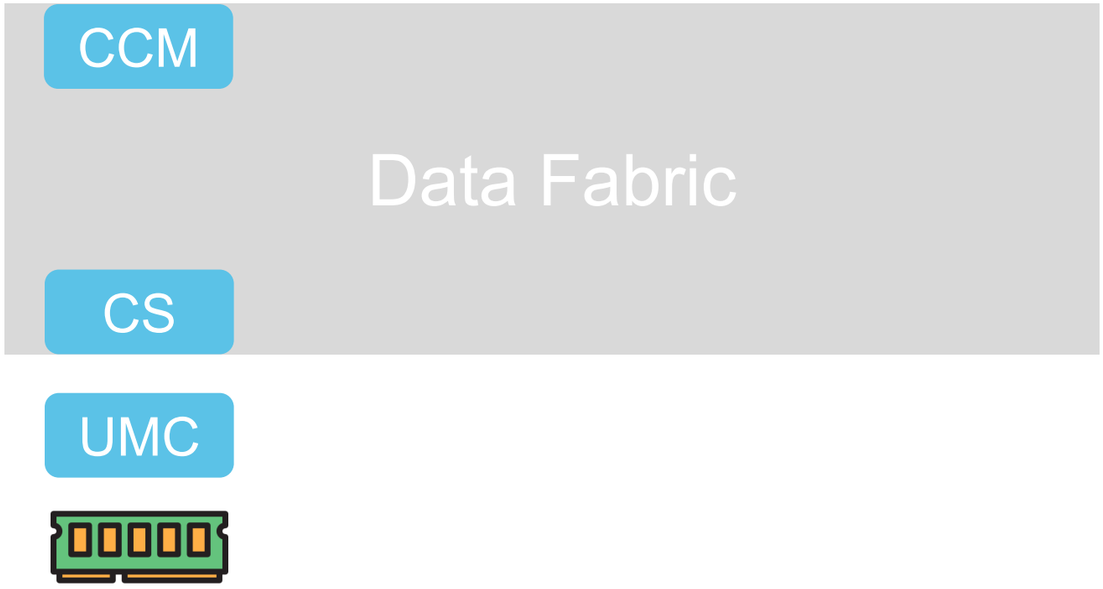
CS: Coherent Slave



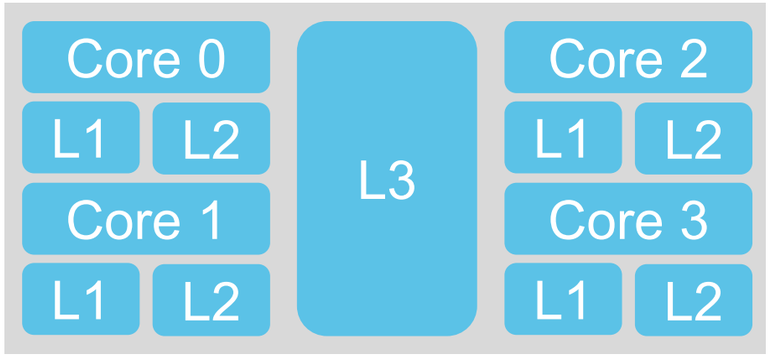
More Background



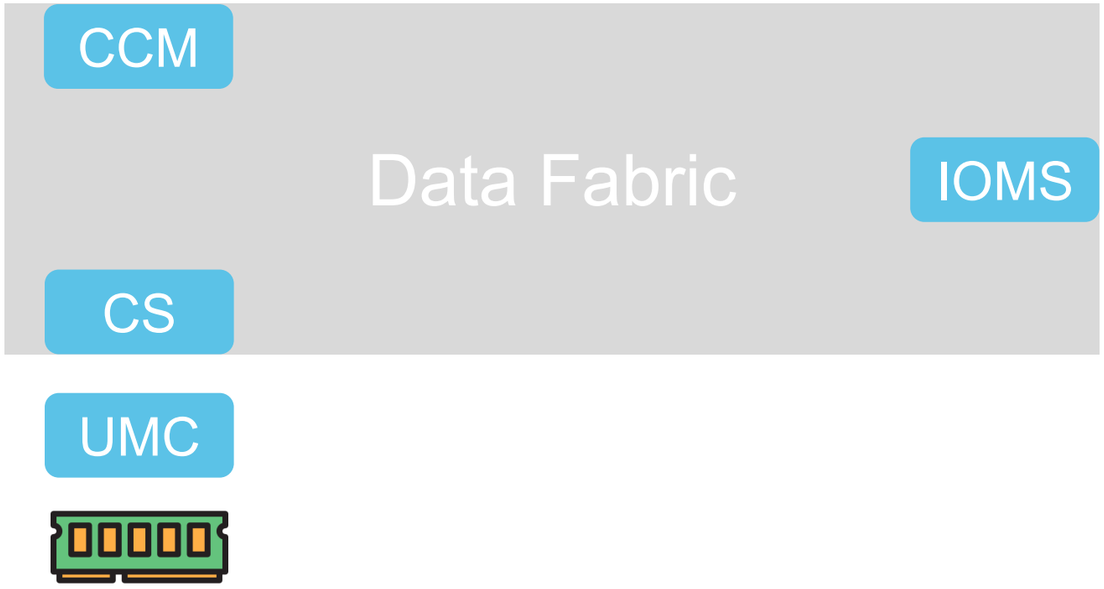
UMC: Unified Memory Controller



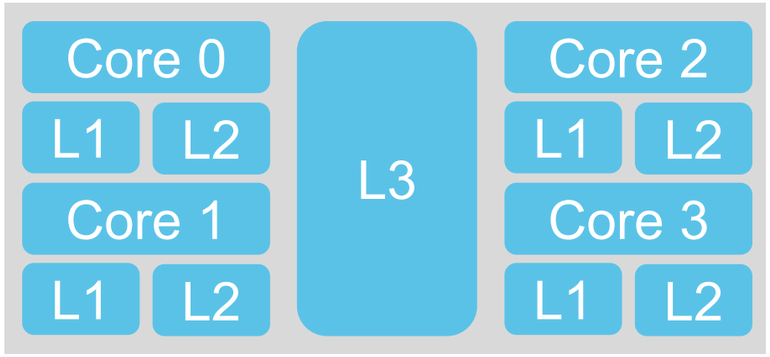
More Background



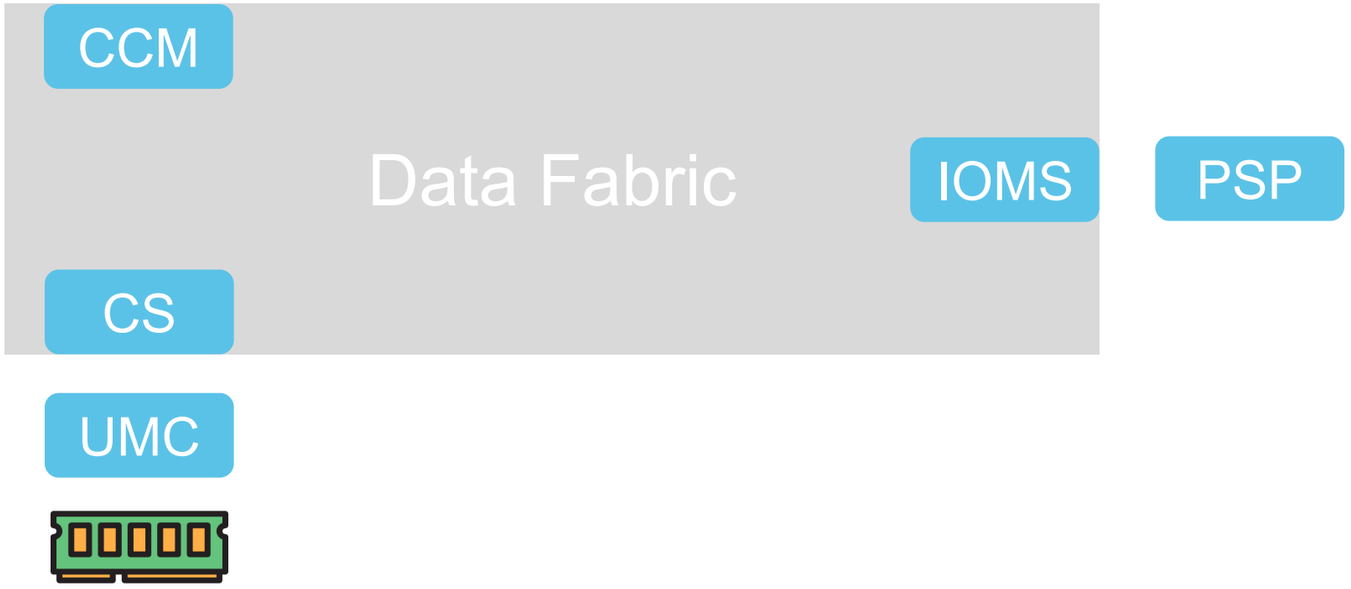
IOMS: I/O Master Slave



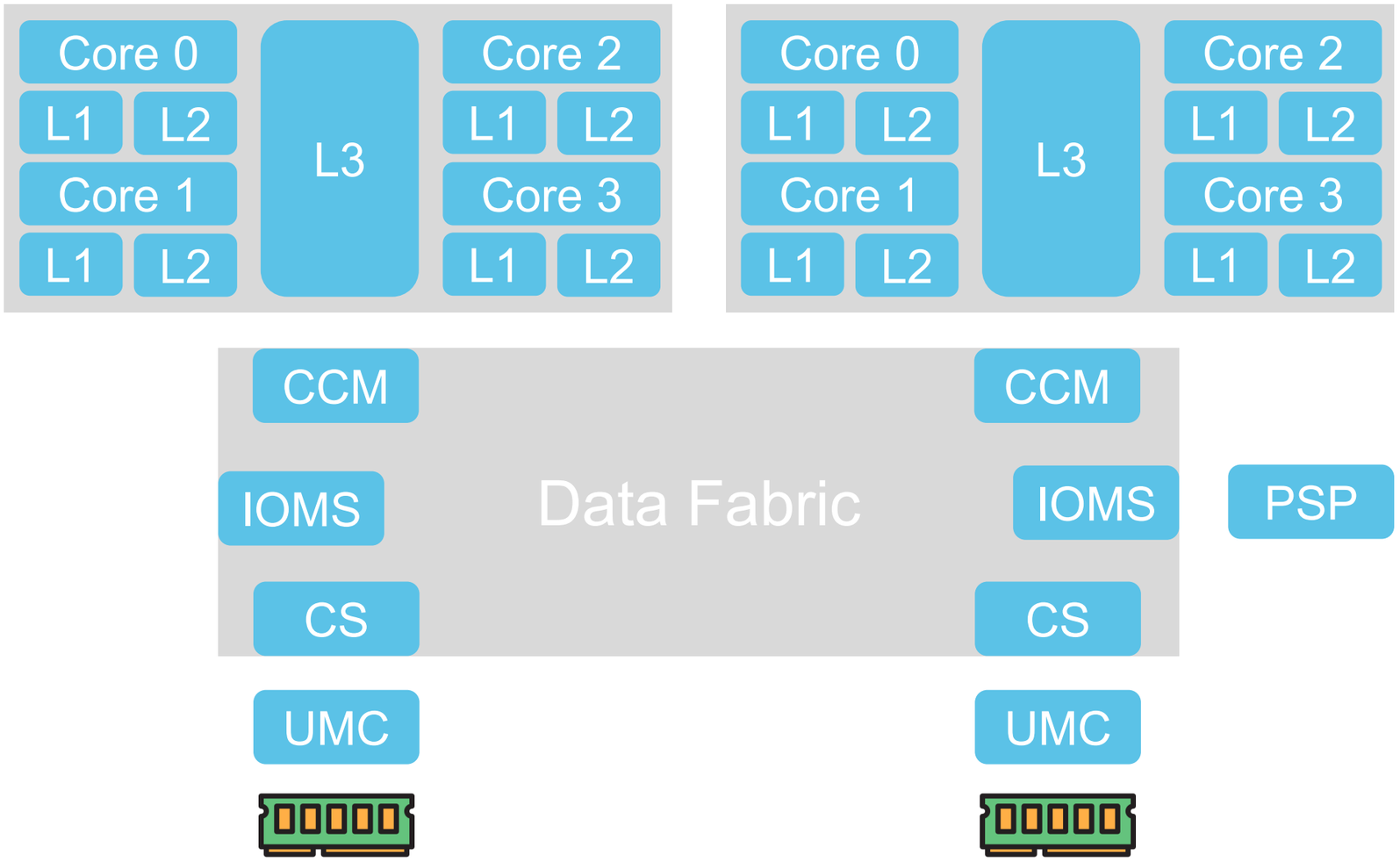
More Background



PSP: Platform Security Processor



More Background



RMP Protections During Initialization

Trusted Memory Region (s)

TMRs block accesses between the CS and UMC based on their origin.

```
/**
 * PSP TMR Allocation can be found here:
 * http://confluence.amd.com/display/ASST/PSP+TMR+Allocation
 *
 * ES_TMR should be dynamically allocated
 */
#define SEV_ES_TMR_POOL     (1)
#define SEV_ES_TMR_CRC      (2)
#define SEV_ES_TMR_SIZE     (0x100000)
#define SEV_ES_TMR_SIZE_SNP (0x200000) /* If SNP Enabled, 2MB TMR. CSF-945 */
#define SNP_RMP_CPU_TMR     (3)
#define SNP_RMP_IOMMU_TMR   (4)
#define SEV_CPU_TMR         (5) /* S3 image */
#define SEV_CPU_TMR2        (6) /* SSCB access TMR */
#define SNP_RMP_TMPM_TMR    (7) /* Allow RMP writes by TMPM */
```

https://github.com/benschlueter/AMD-ASPFW/blob/3ca6650dd35d878b3fcbe5c7f58b145eed042bbf/fw/psp_bl_uapps/sev_uapp/src/df_regs.h#L21

Trusted Memory Region (s)

```
/* D18F7xA08...D18F7xAB0 [Trusted Memory Region Control 7:0] (DF::TmrCtl) */
#define TMR_CTRL_SEC_VALID_SHIFT    (16)                                /* Enable TMR security level checks. Not used for SNP */
#define TMR_CTRL_SEC_VALID_FLAG     (1ul << TMR_CTRL_SEC_VALID_SHIFT)
#define TMR_CTRL_SAE_IOM_SHIFT      (13)
#define TMR_CTRL_SAE_IOM_FLAG       (1ul << TMR_CTRL_SAE_IOM_SHIFT)   /* Source access == IOM (IOMMU) */
#define TMR_CTRL_SAE_NCM_SHIFT      (12)
#define TMR_CTRL_SAE_NCM_FLAG       (1ul << TMR_CTRL_SAE_NCM_SHIFT)   /* Source access == Non-Coherent Master (MPDMA/etc) */
#define TMR_CTRL_SAE_CCM_SHIFT      (10)
#define TMR_CTRL_SAE_CCM_FLAG       (1ul << (TMR_CTRL_SAE_CCM_SHIFT)) /* Source access == CCM (x86, uCode) */
#define TMR_CTRL_SAE_PIE_SHIFT      (9)
#define TMR_CTRL_SAE_PIE_FLAG       (1ul << (TMR_CTRL_SAE_PIE_SHIFT)) /* Source access == PIE (DF: Power, Interrupts, Etcetera)
*/
#define TMR_CTRL_SAE_SMU_SHIFT      (8)
#define TMR_CTRL_SAE_SMU_FLAG       (1ul << (TMR_CTRL_SAE_SMU_SHIFT)) /* Source access == SMU */
```

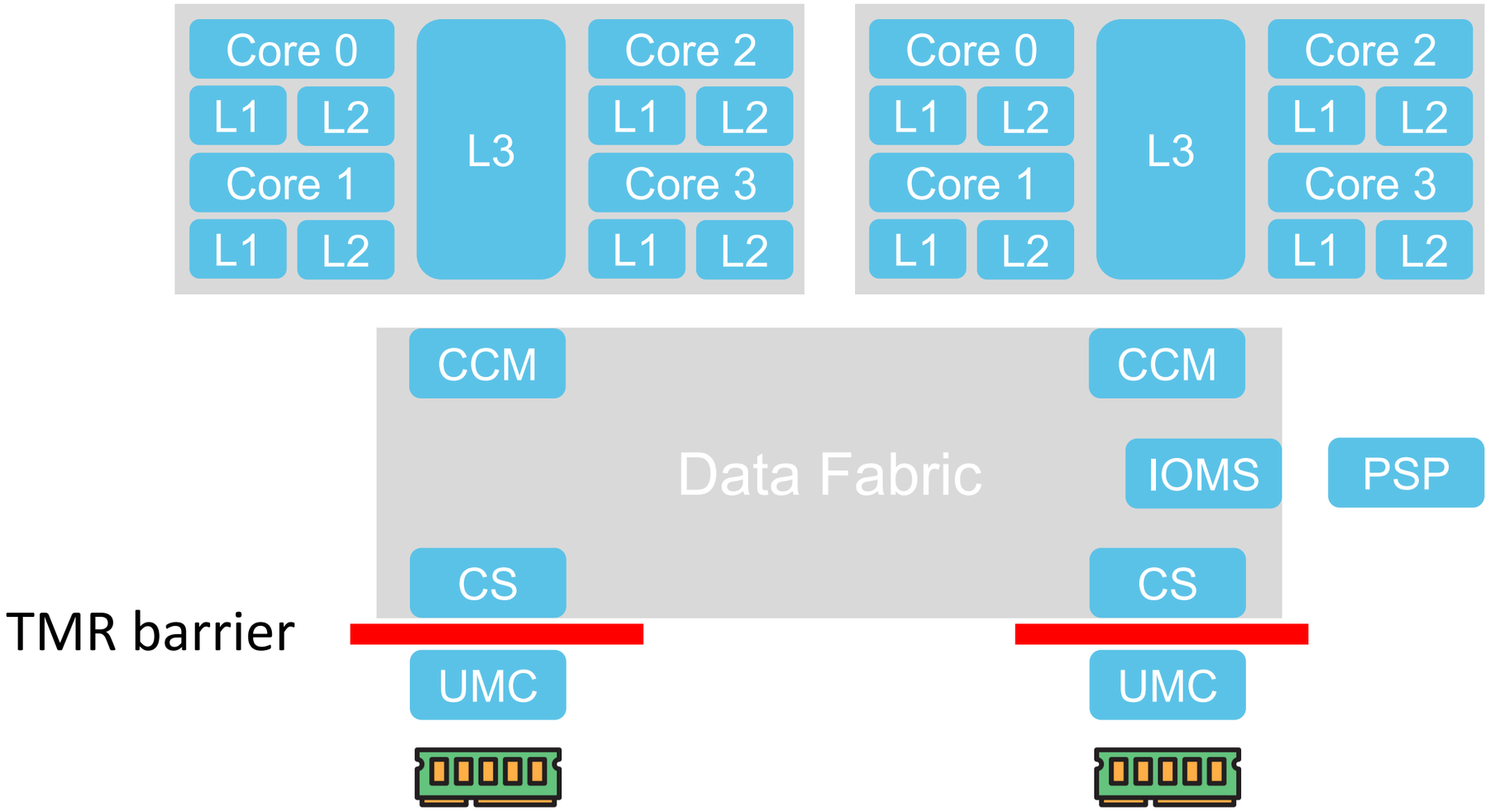
https://github.com/benschlueter/AMD-ASPFW/blob/3ca6650dd35d878b3fcbe5c7f58b145eed042bbf/fw/psp_bl_uapps/sev_uapp/src/df_regs.h#L39

Trusted Memory Region (s)

PSP removes TMR after RMP initialization such that x86 microcode (e.g., RMPUPDATE) can modify the RMP.

https://github.com/benschlueter/AMD-ASPFW/blob/3ca6650dd35d878b3fcbe5c7f58b145eed042bbf/fw/psp_bl_uapps/sev_uapp/src/df_regs.h#L72C1-L79C4

Trusted Memory Region (s)



TMR barrier

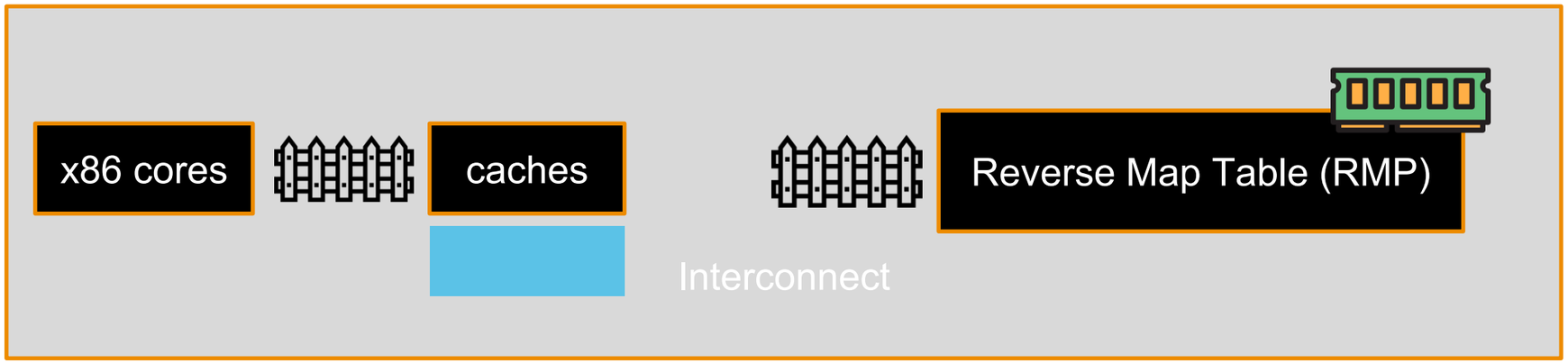
Barrier on the x86 cores

```
/*
    * If we are going to initialize the RMP,
    * tell other MPs to protect the RMP from writes.
    * This operation waits for the other MPs to 'ack' that they will not write
    * over the RMP which would cause a system crash after we install the TMR.
*/
```

https://github.com/benschlueter/AMD-ASPFW/blob/3ca6650dd35d878b3fcbe5c7f58b145eed042bbf/fw/psp_bl_uapps/sev_uapp/src/sev_mcmd.c#L4250

Barrier on the x86 cores

Initially we assumed there must be an x86 barrier



Experiments

x86 core #0

```
do_sev_snp_init()
pr_info(“RMP[0] 0x%llx\n,
RMPTABLE[0]);
```

x86 core #1

```
WRITE_ONCE(RMPTABLE[0],0xABCD);
while (!kthread_should_stop())
  if (READ_ONCE(RMPTABLE[0])!= 0xABCD)
    WRITE_ONCE(RMPTABLE[0],0xAAAACCCC);
```

Experiments

x86 core #0

```
do_sev_snp_init()
pr_info(“RMP[0] 0x%llx\n,
RMPTABLE[0]);
```

x86 core #1

```
WRITE_ONCE(RMPTABLE[0],0xABCD);
while (!kthread_should_stop())
  if (READ_ONCE(RMPTABLE[0])!= 0xABCD)
    WRITE_ONCE(RMPTABLE[0],0xAAAACCCC);
```

```
RMP[0] 0xAAAACCCC
```

Experiments

x86 core #0

```
do_sev_snp_init()
pr_info("RMP[0] 0x%llx\n,
RMPTABLE[0]);
```

x86 core #1

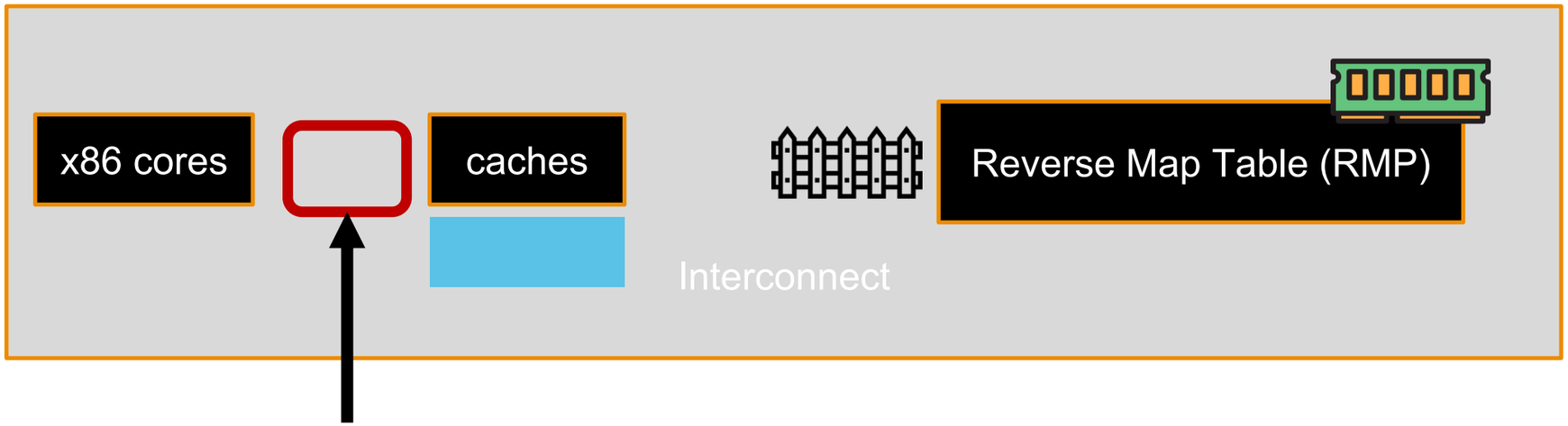
```
WRITE_ONCE(RMPTABLE[0],0xABCD);
while (!kthread_should_stop())
  if (READ_ONCE(RMPTABLE[0])!= 0xABCD)
    WRITE_ONCE(RMPTABLE[0],0xAAAACCCC);
```

Sucessful overwrite **during** RMP init

```
RMP[0] 0xAAAACCCC
```

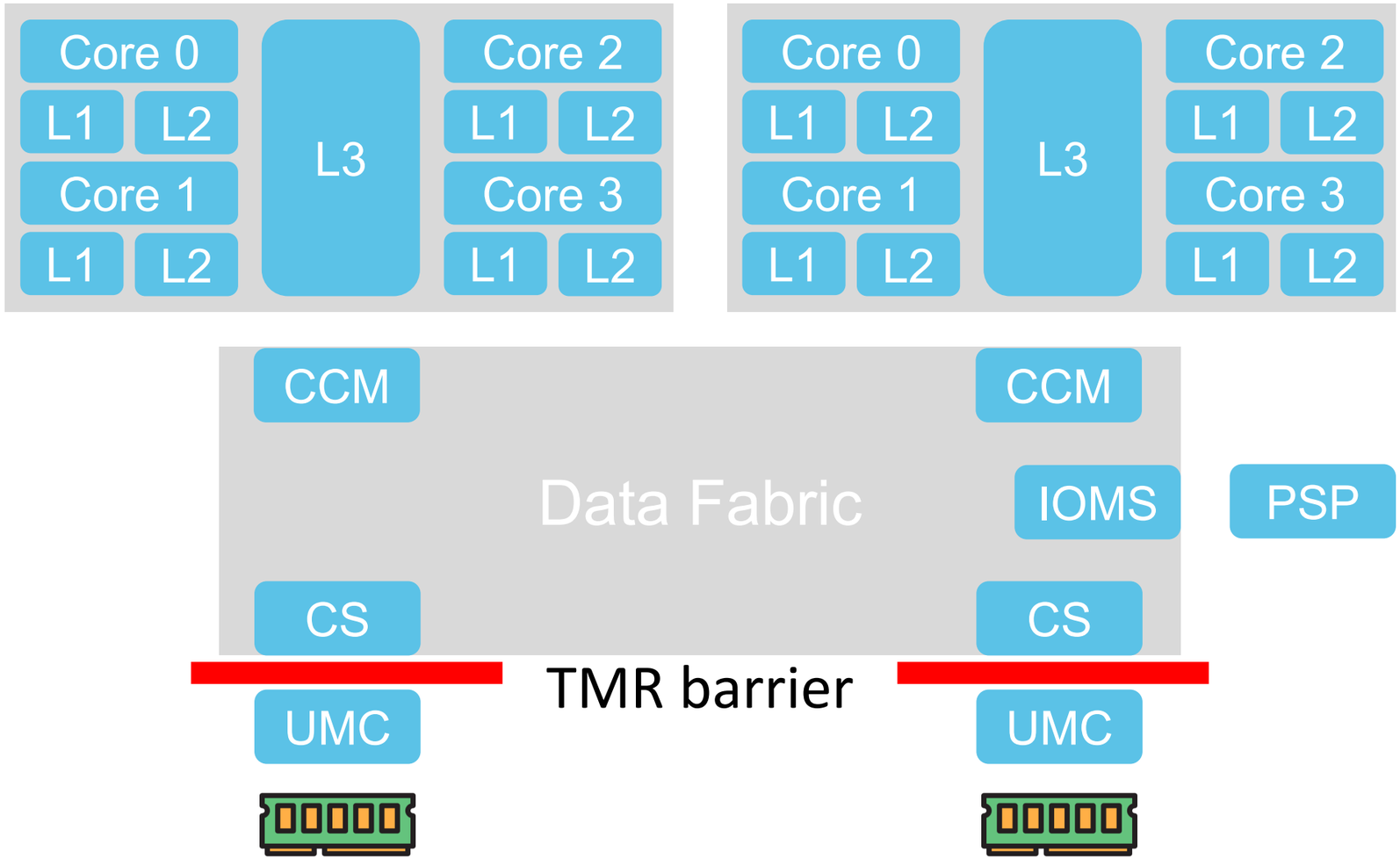
Barrier on the x86 cores

Our experiments show that there is no barrier between the x86 cores and the caches during RMP init



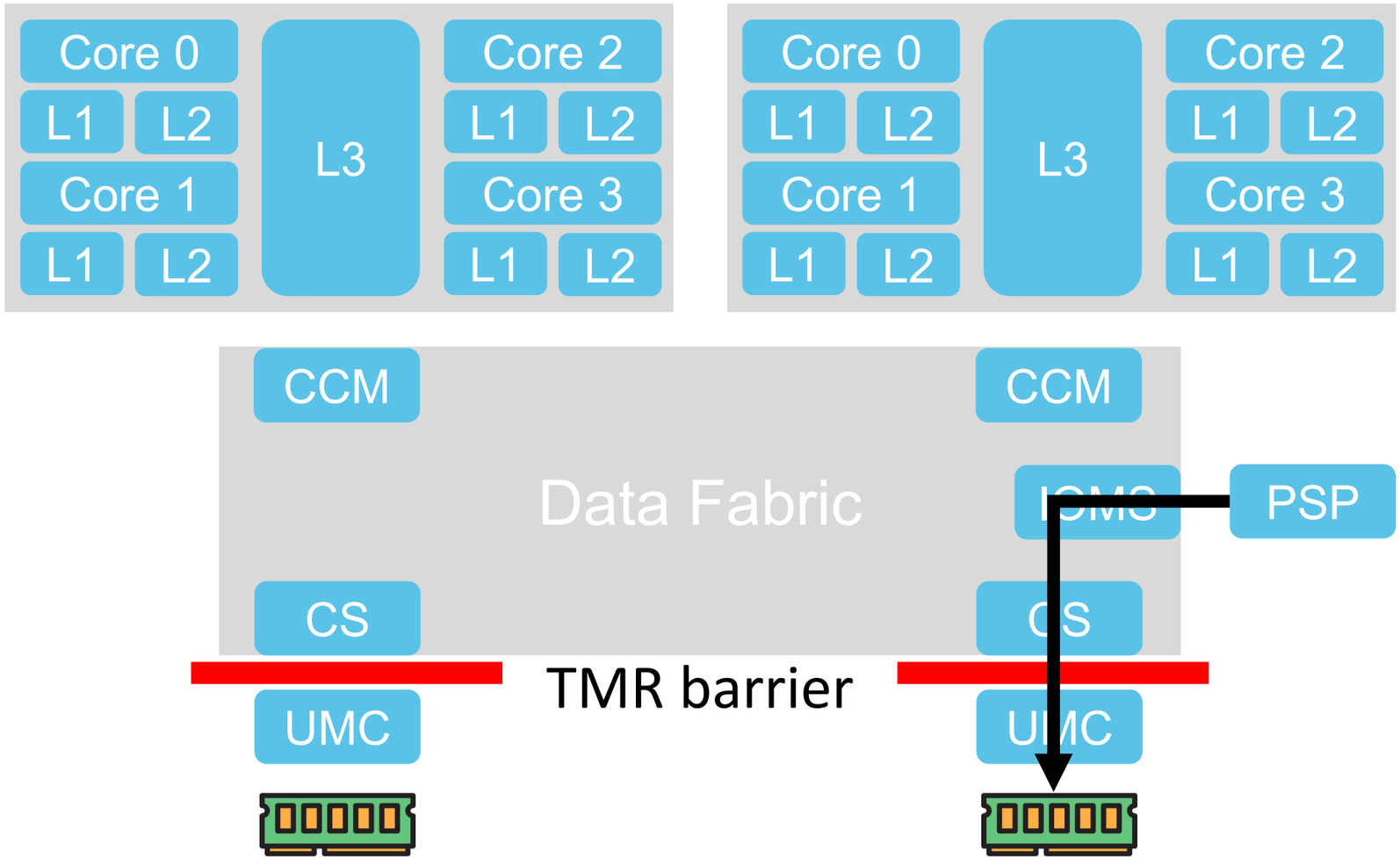
SEV-SNP Initialization Flow

PSP initializes TMR barrier to block x86 writes



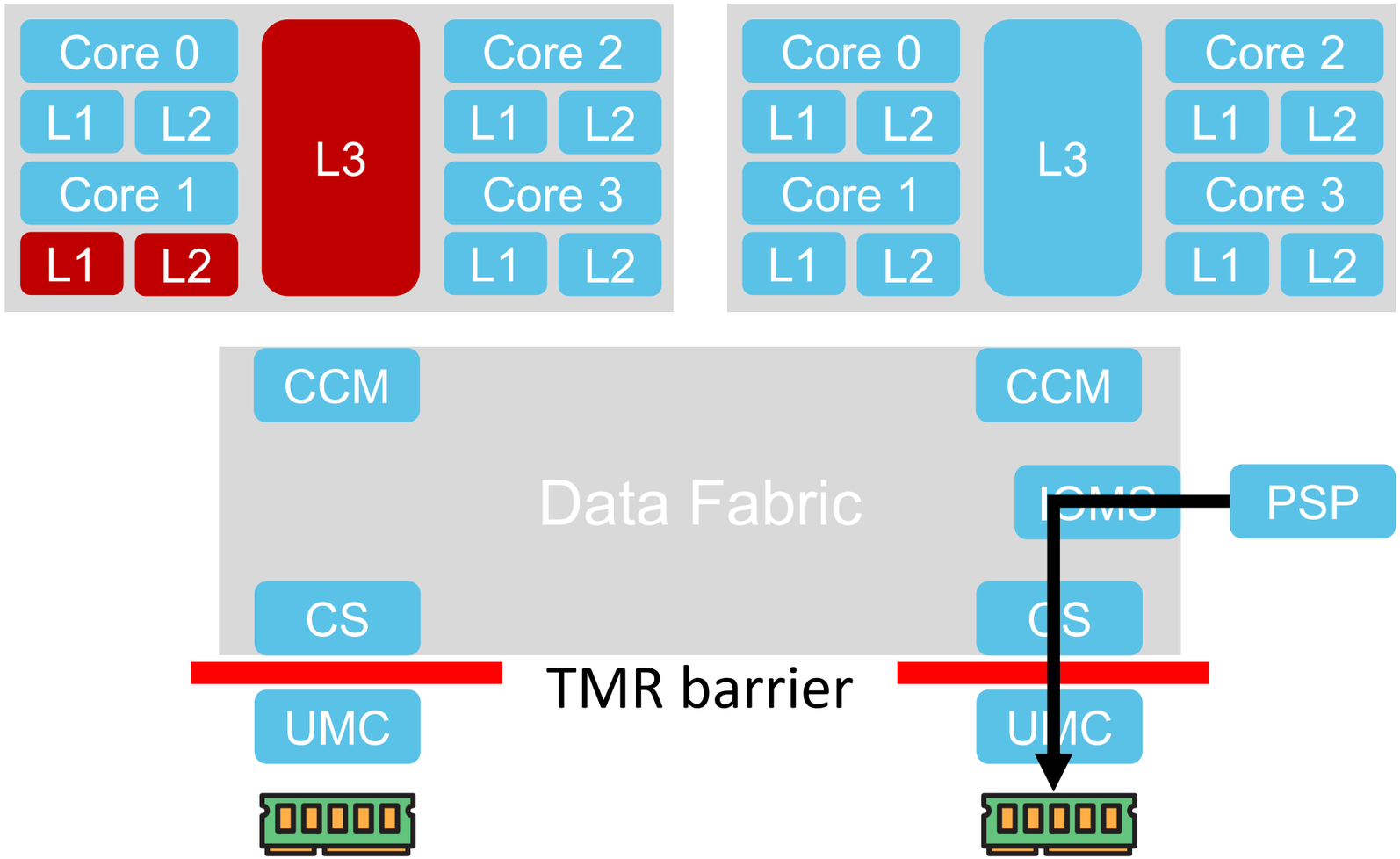
SEV-SNP Initialization Flow

PSP initializes the RMP table by writing the secure start configuration to DRAM



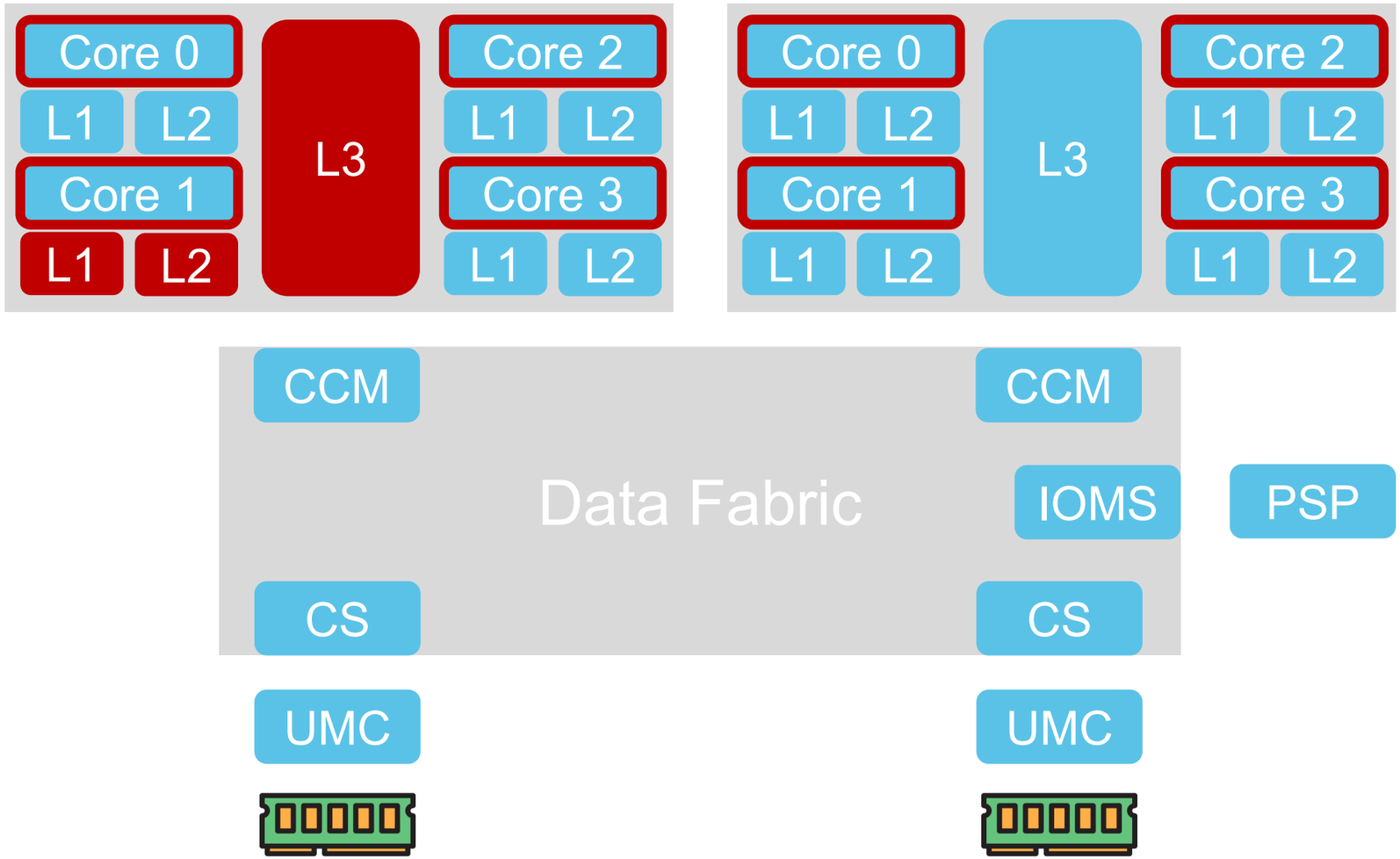
SEV-SNP Initialization Flow

Cores can still create dirty cachelines pointing to RMP DRAM



SEV-SNP Initialization Flow

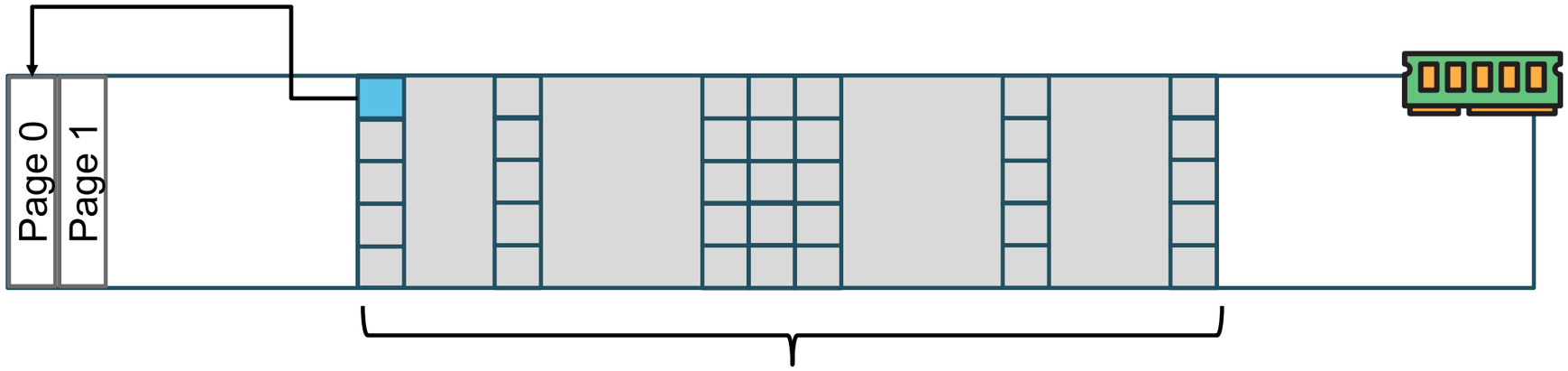
PSP enables RMP checks at the core level and removes TMR barrier



Exploitation

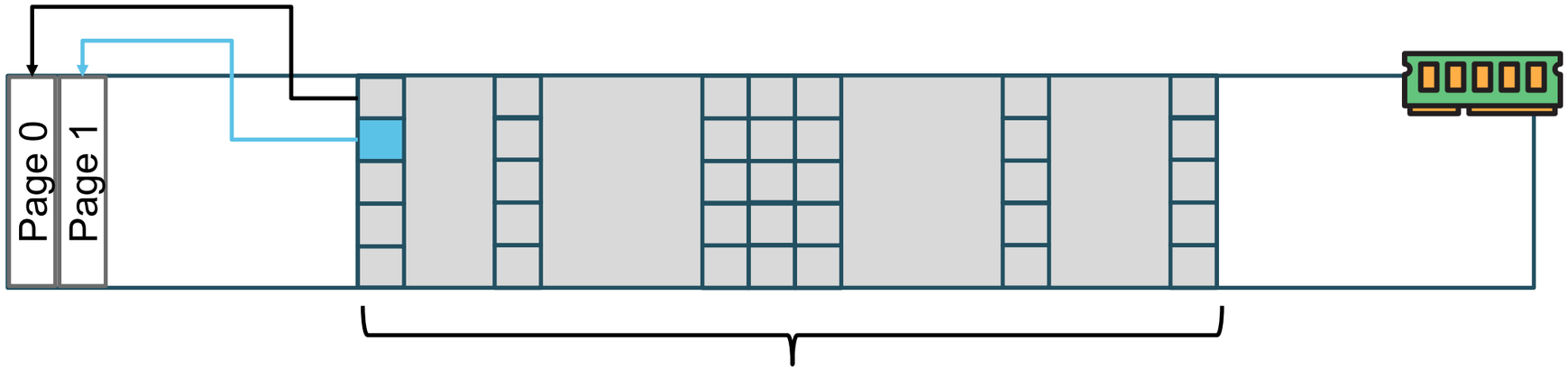
RMP in DRAM

RMP entry referencing the first Page [0x0000]



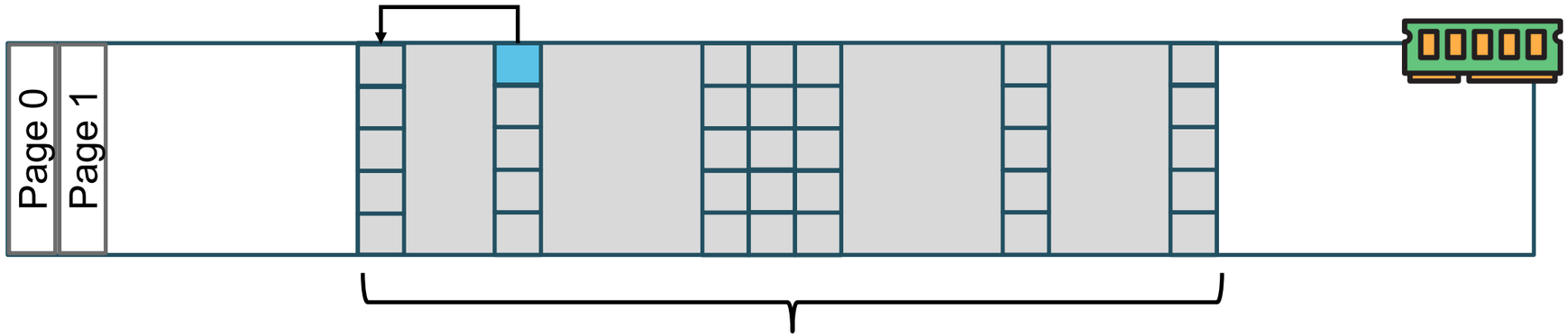
RMP in DRAM

RMP entry referencing the second Page [0x1000]



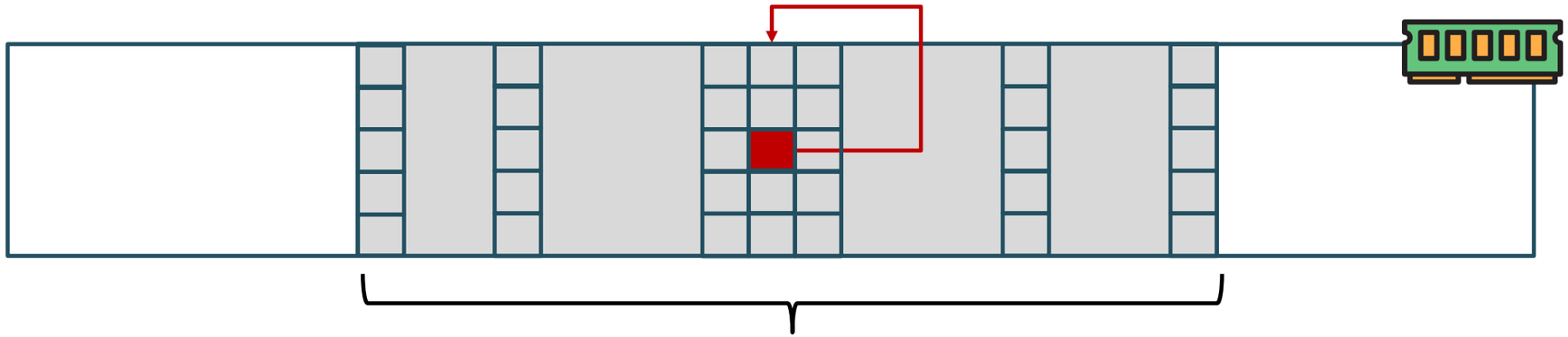
RMP in DRAM

RMP entry referencing the first RMP table page



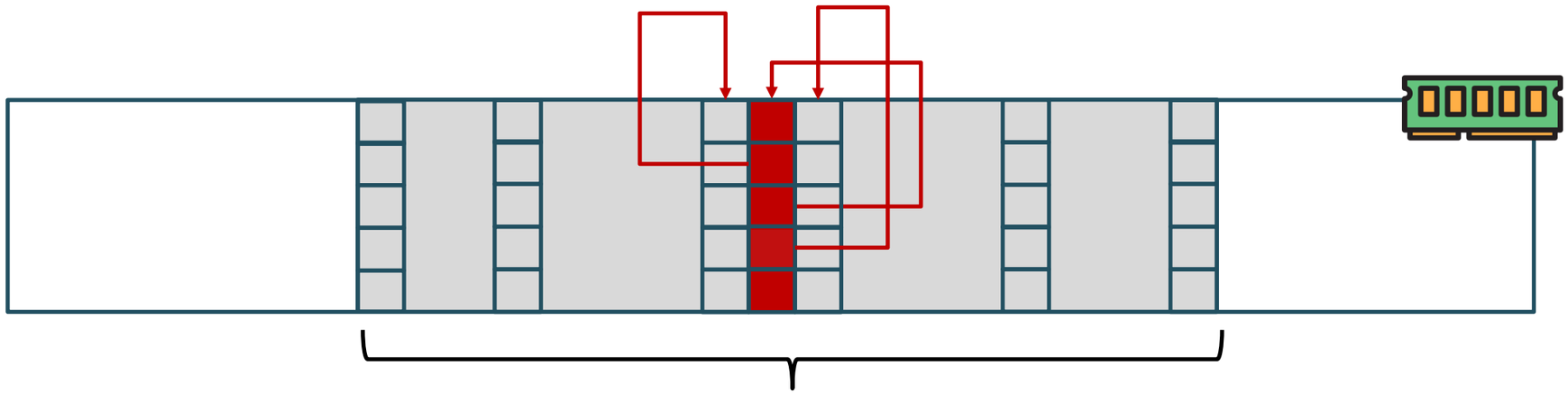
RMP in DRAM

RMP entry referencing itself

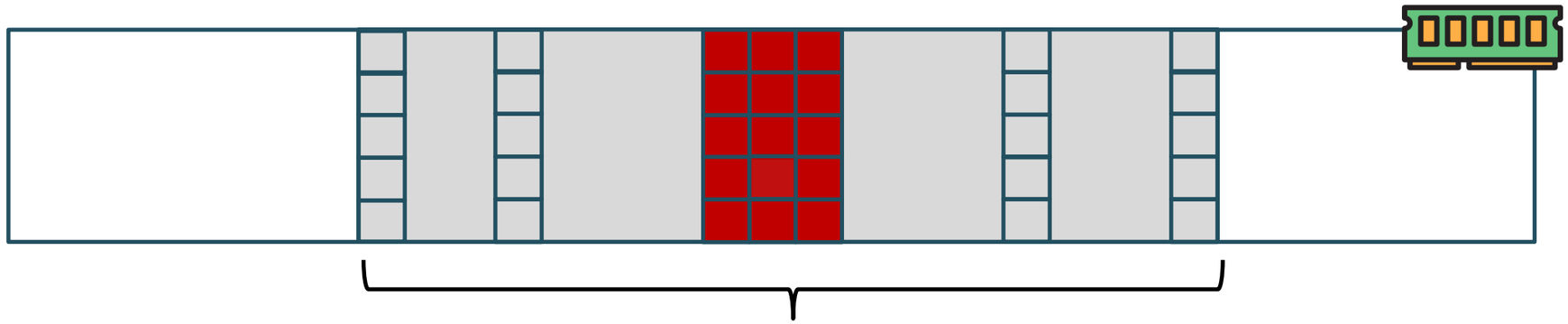


RMP Table in DRAM

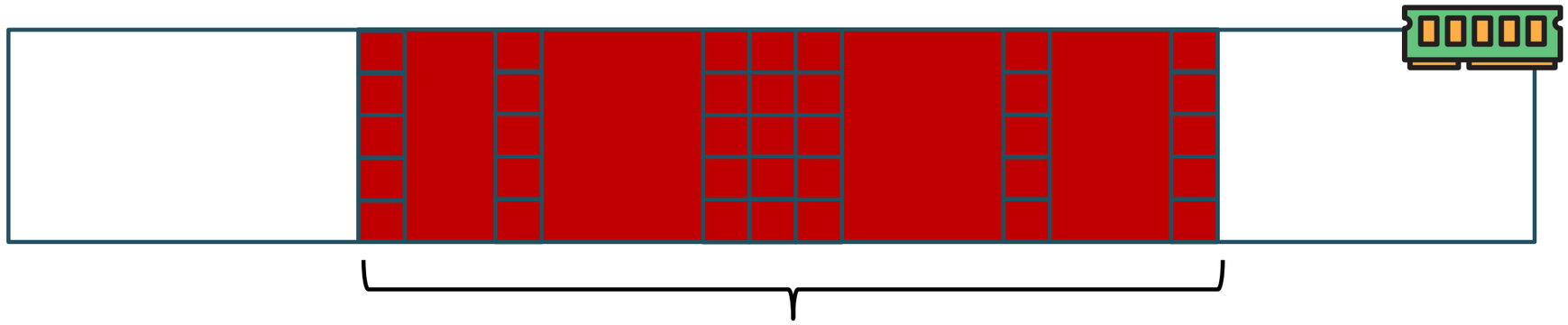
RMP Overwrite



RMP Overwrite

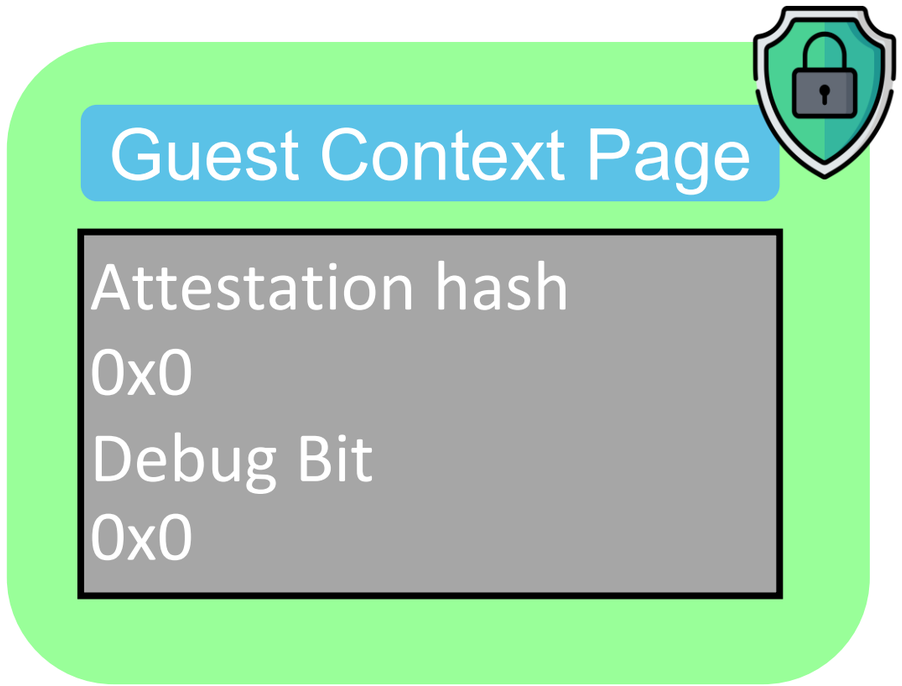


RMP Overwrite



Impact of a compromised RMP

RMP prevents hypervisor writes to sensitive SEV-SNP VM metadata



Impact of a compromised RMP

RMP prevents hypervisor writes to sensitive SEV-SNP VM metadata

Guest Context Page

Attestation hash
0x0
Debug Bit
0x0

Compromised RMP allows the hypervisor to overwrite “Guest Context Page”

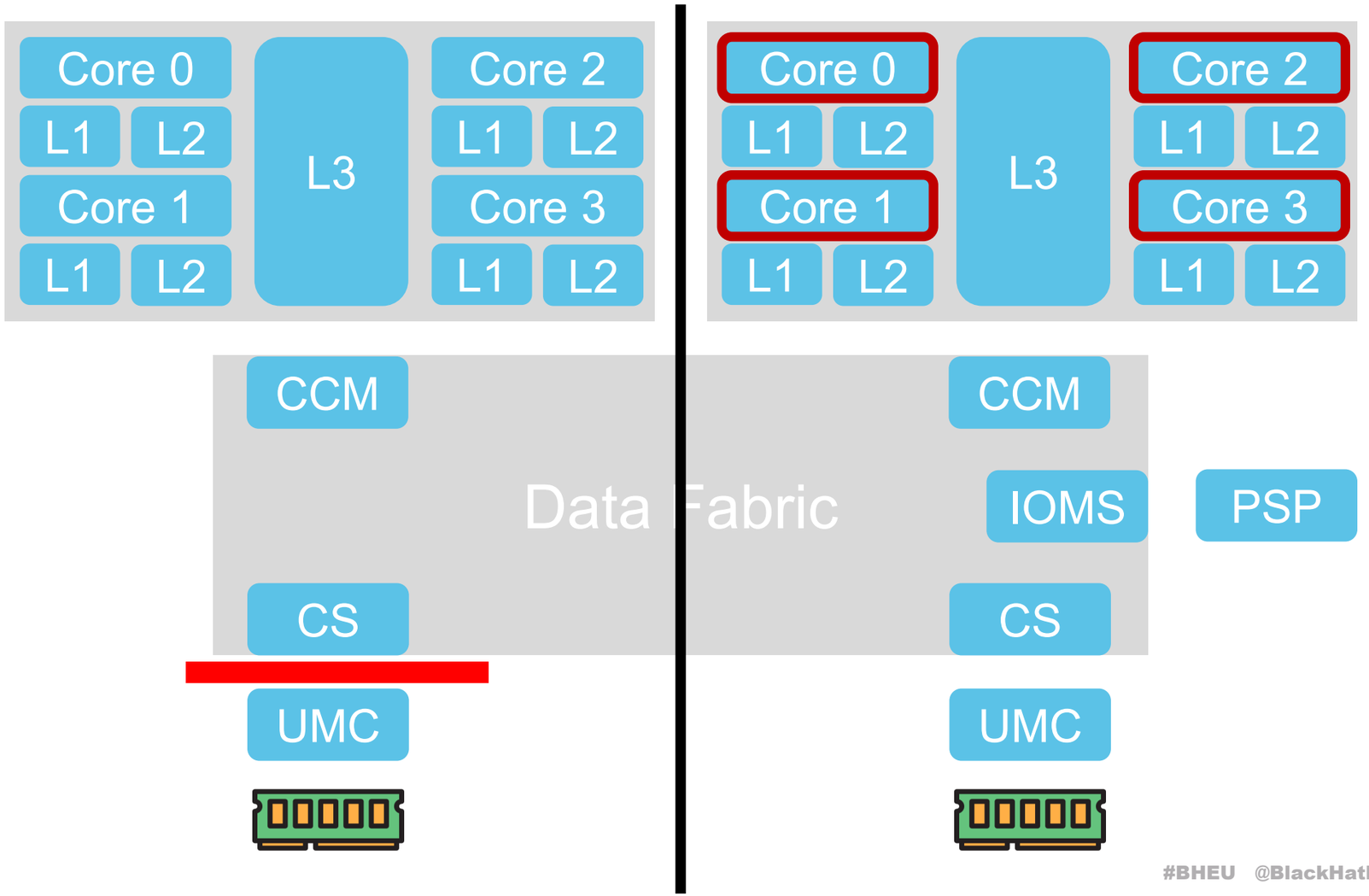
Guest Context Page

Attestation hash
0xAAAAAAAA
Debug Bit
0x1

Root Cause

Root Cause Analysis

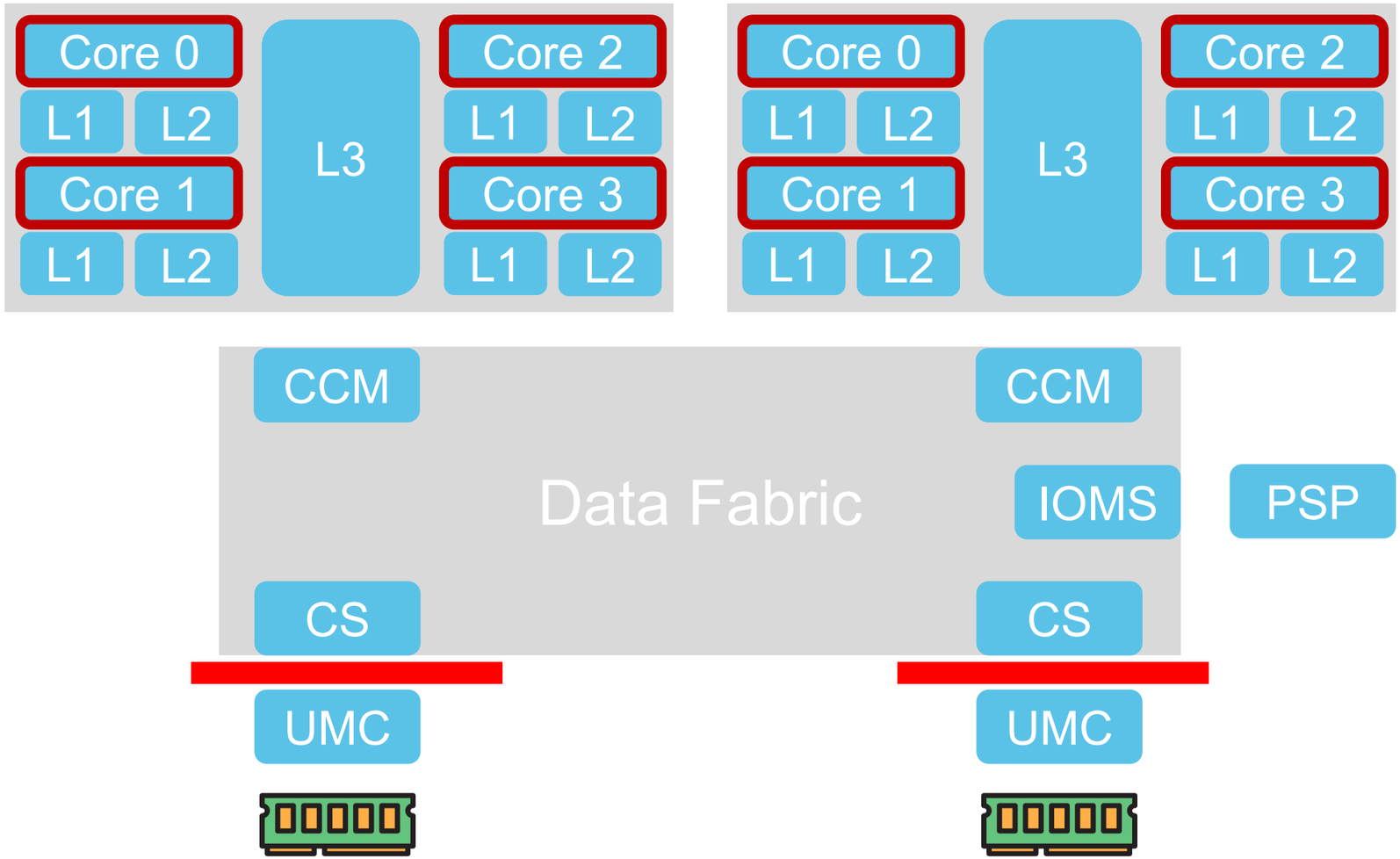
Misalignment between two different barriers allow for unchecked writes to RMP memory



Mitigation

Proposed Mitigation

Barriers between the cores and caches during RMP initialization



Summary

RMPocalypse is a firmware/microcode vulnerability on

- Zen 3 (first SEV-SNP processor, 2021)
- Zen 4 (2022)
- Zen 5 (newest generation, 2024)

https://rmpocalypse.github.io/

CVE 2025-0033

Mitigation:

- AMD supplied firmware updates (PSP SEV firmware + Microcode)

Key Takeaways

- A single unchecked 8 byte write compromises all security guarantees of SEV-SNP
- By open sourcing proprietary platform code AMD allows external security researchers to audit their system and secure it
- Modern systems consist of more components than jus the CPU and DRAM that offer attack surfaces