SENTER Sandman: Using Intel TXT to Attack BIOSes

Xeno Kovah @xenokovah

Corey Kallenberg @coreykal

John Butterworth @jwbutterworth3

Sam Cornwell @ssc0rnwell

MITRE

Introduction

- **Who we are:**
  - Trusted Computing researchers at The MITRE Corporation
- **What MITRE is:**
  - A not-for-profit company that runs seven US Government "Federally Funded Research & Development Centers" (FFRDCs) dedicated to working in the public interest
  - The first .org, !(.mil | .gov | .com | .edu | .net), on the ARPANET
  - Manager for a number of standards such as *CVE*, CWE, OVAL, CAPEC, STIX, TAXII, etc

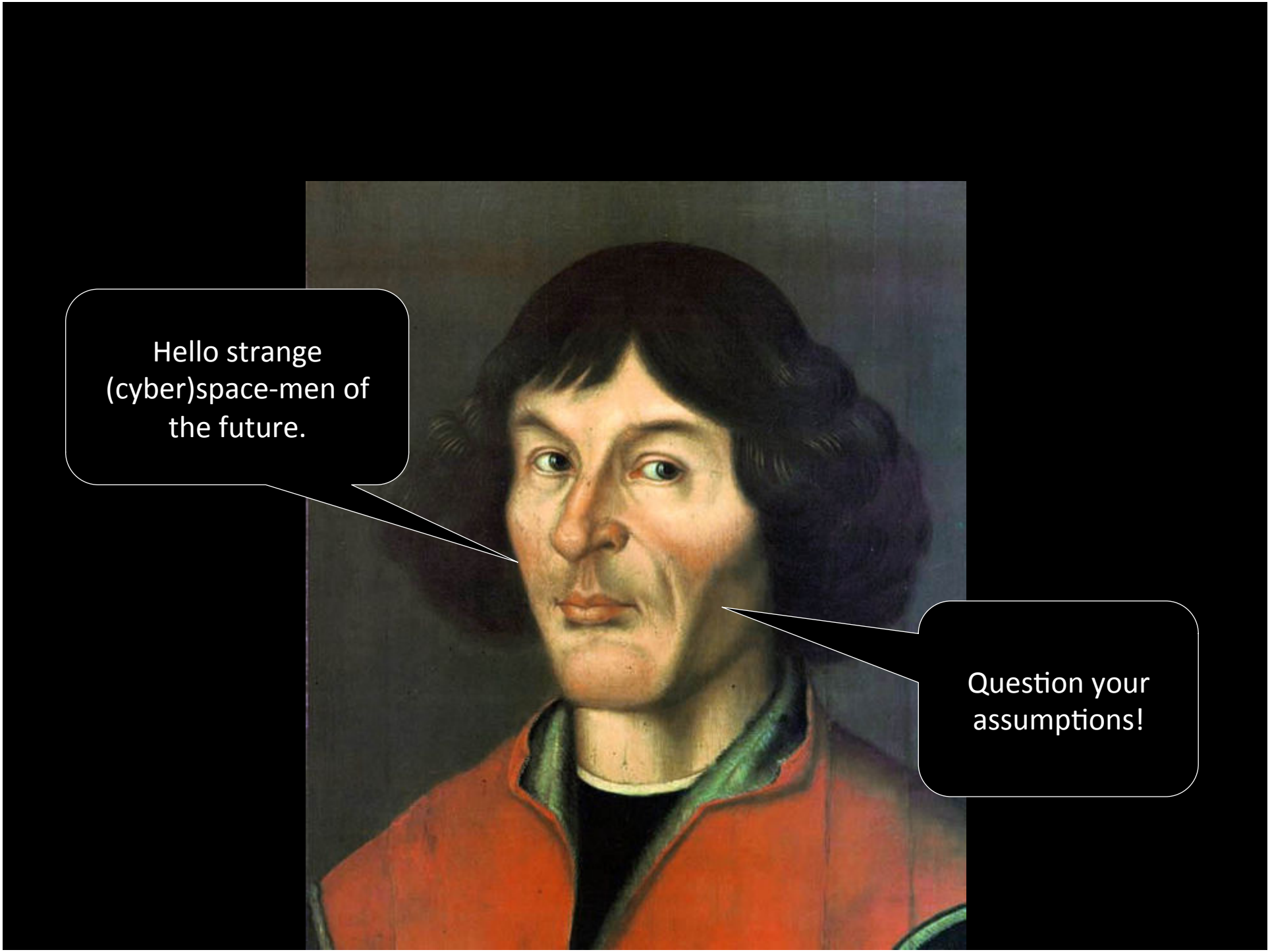
I'd like to tell you a dark fairy tale

The following story is fictional,
and does not depict any actual event

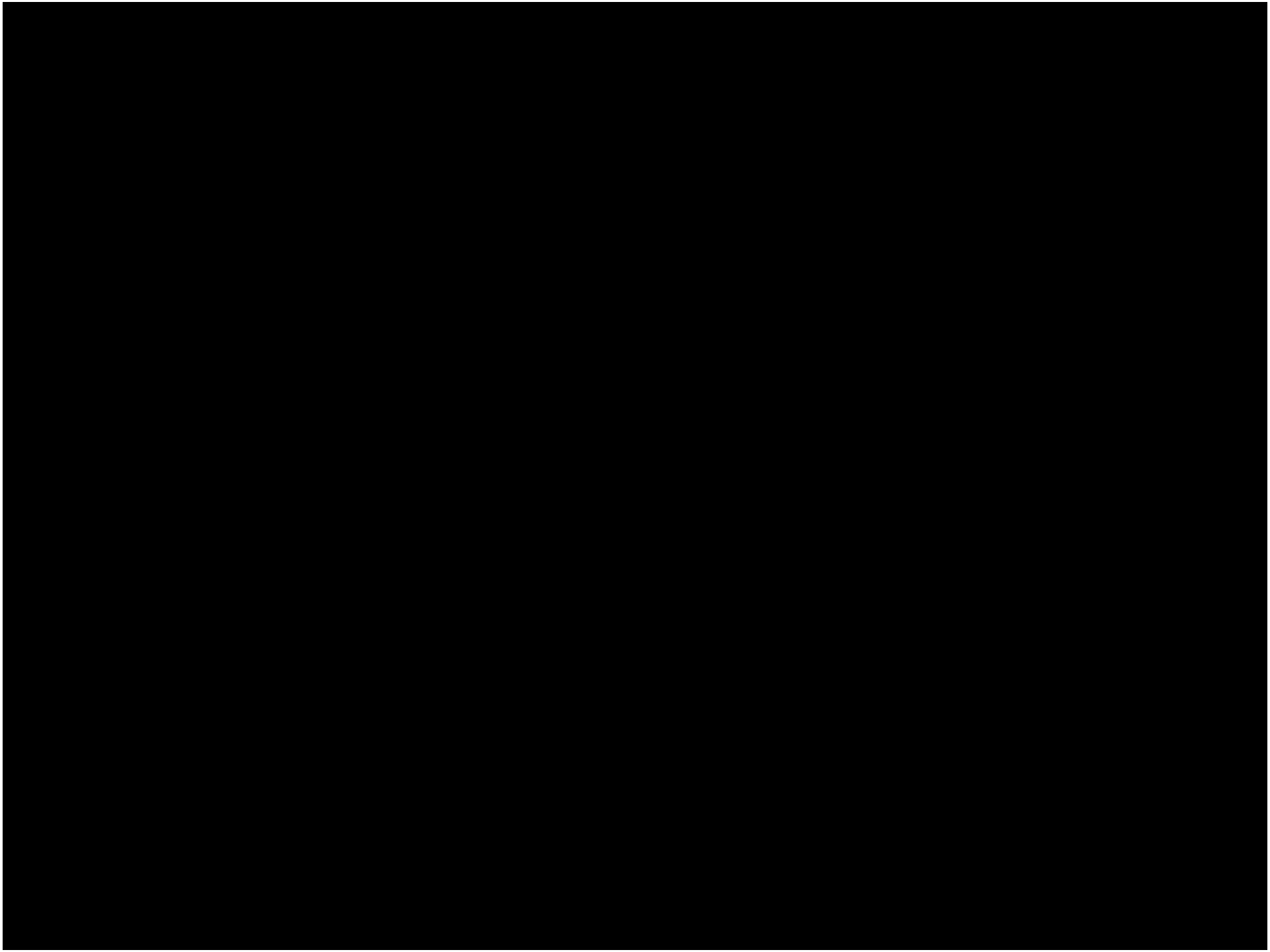
MITRE

Once Upon a Time…

- Off in never never land…


Hello strange (cyber)space-men of the future.
Question your assumptions!


SCIENCE!




THE
ONCE &
FUTURE

Myth meets reality

- **In July 2013 at BlackHat[18], in concert with our talk about BIOS attacks and exploits, we publicly released Copernicus. It was the first free public tool that could check for access control vulnerabilities in a BIOS implementation, and the first that could dump the BIOS contents on most x86-based Windows.**
  - You can download it by googling "MITRE Copernicus" or at [26].
- **In March 2014 at CanSecWest[27] we described an SMM MitM attack ("Smite'em") which can generically defeat many software-based BIOS dump tools like Copernicus, Flashrom, ChipSec and others. But we also released Copernicus 2 which could defeat Smite'em by using Intel Trusted Execution Technology (TXT).**
  - You can download it by googling "MITRE Copernicus 2" or at [28].

Myth meets reality 2

- **In April 2014 at Syscan[29], we publicly disclosed a weakness in some BIOSes where , and we showed a PoC attacker against this weakness called Charizard**
  - So named because Sam Cornwell didn't think of a better name before we dubbed it in honor of his status as youngest team member.
  - The updated Copernicus was posted at the original [26] link.
- **And now here we present for the first time the Sandman, which is like using Copernicus 2 code to perform a Charizard attack.**

There can be only one! Who wins in a smackdown between BIOS malware, and BIOS integrity checkers?

- **The malware. Always.**
- **We built Copernicus to be something "best effort" that could be deployed quickly with minimal requirements, to try and catch firmware malware with their pants down**
  - Where there was darkness, we said "let there be light!" ;)
  - When we live in a world where no one is checking their firmware, any firmware malware need not necessarily fear detection and thus can be vulnerable to a surprise detect
  - Just the act of existing costs attackers development time/money if they hadn't previously provided any self-protection
- **Now let's see what we need to do to make Copernicus actually trustworthy**

It's well understood how to manipulate integrity checking software output

- From within the OS, targeted hooks into Copernicus code
- From within the OS with “DDefy” [20] rootkit style hooks into file writing routines
- From within the HD controller firmware [21][22][23]
- From within the OS with a network packet filter driver
- From within the NIC firmware [24][25]
- Etc. Lots more options

A new generic attack.

- It is possible for SMM to be notified when SPI reads or writes occur
- An attacker who controls the BIOS controls the setup of SMM
- In this way a BIOS-infecting attacker can perform a SMM MitM attack against those who would try to read the BIOS to integrity check it
- We call our SMM MitM “Smite’em, the Stealthy”

Eye of the dragon - FSMIE - hw sequencing

- **This is what allows an attacker in SMM to know when someone is trying to access the flash chip (because a System Management Interrupt (SMI) will fire)**

HSFC—Hardware Sequencing Flash Control Register (SPI Memory Mapped Configuration Registers)

Memory Address:SPIBAR + 06h Attribute: R/W, R/WS
Default Value: 0000h Size: 16 bits

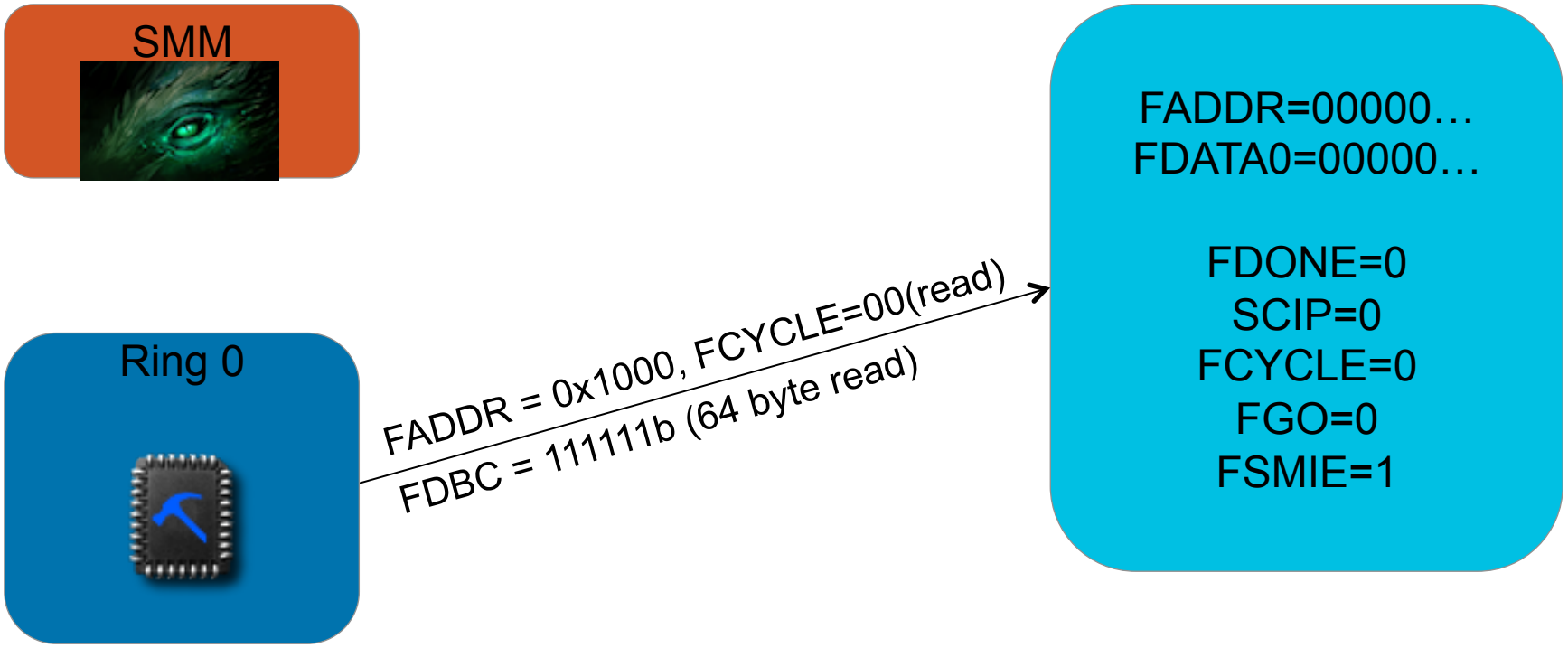
This register is only applicable when SPI device is in descriptor mode.

Bit	Description
15	**Flash SPI SMI# Enable (FSMIE)** — R/W. When set to 1, the SPI asserts an SMI# request whenever the Flash Cycle Done bit is 1.

- **The Flash Cycle Done bit is set to 1 after every read and write**

MITRE

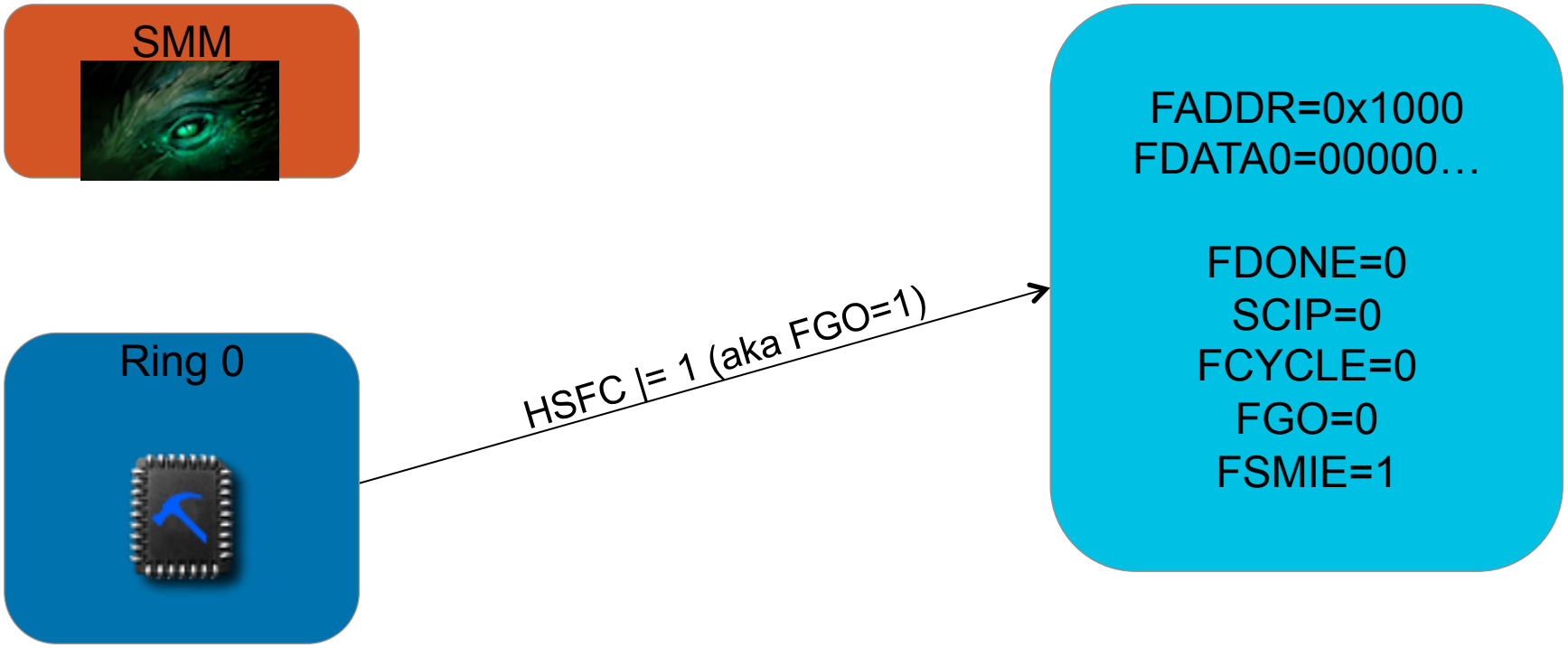
Reading the flash chip in the presence of Smite'em



- **BIOS reading software sets up the location it wants to read (as part of reading the entire chip) and how many bytes to read**

MITRE

Reading the flash chip in the presence of Smite'em



- **BIOS reading software says to start the read**

Reading the flash chip in the presence of Smite'em

SMM

Ring 0

FADDR=0x1000
FDATA0=00000…

FDONE=0
SCIP=1
FCYCLE=0
FGO=1
FSMIE=1

- **Cycle in progress**

MITRE

Reading the flash chip in the presence of Smite'em

SMI

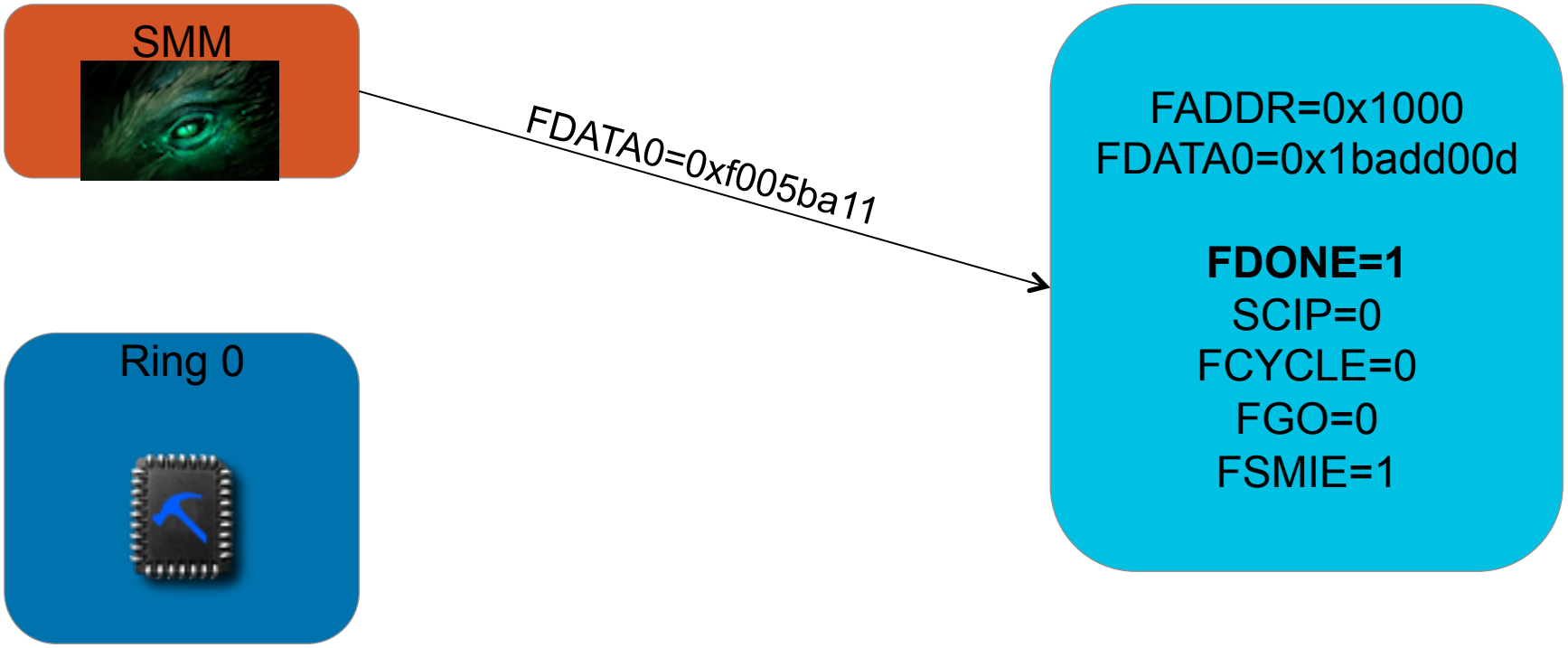
FADDR=0x1000
FDATA0=0x1badd00d

FDONE=1
SCIP=0
FCYCLE=0
FGO=0
FSMIE=1

- **Once the cycle is done, and the data is available for reading, if the FSMIE = 1, an SMI is triggered, giving Smite'em the first look**

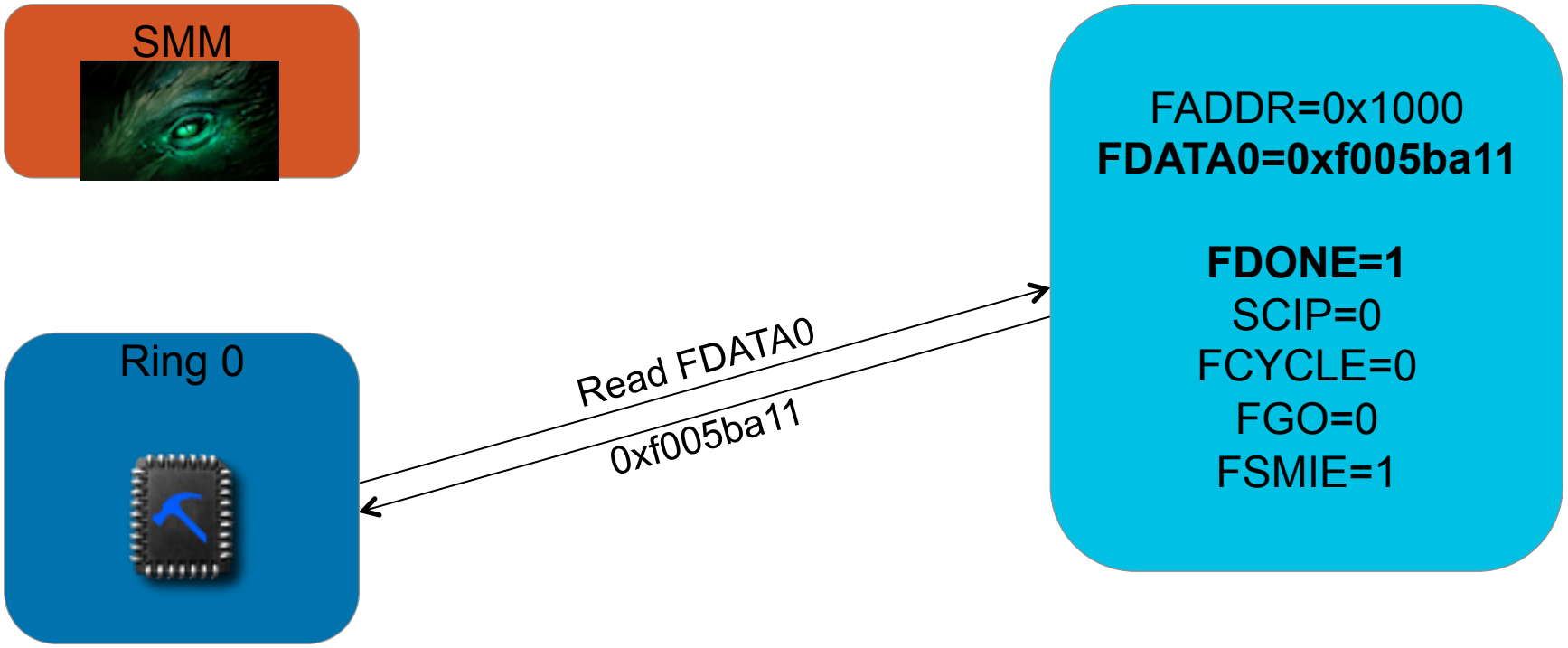
MITRE

Reading the flash chip in the presence of Smite'em



- **Smite'em can change any data that would reveal its presence to the original benign data**

Reading the flash chip in the presence of Smite'em



- **BIOS reading software will be mislead**

Think it can't happen?

- **Flashrom 0.9.7 source**

```
ichspi.c:                hsfc = REGREAD16(ICH9_REG_HSFC);
ichspi.c:                hsfc &= ~HSFC_FCYCLE; /* set read operation */
ichspi.c:                hsfc &= ~HSFC_FDBC; /* clear byte count */
ichspi.c:                hsfc |= (((block_len - 1) << HSFC_FDBC_OFF) & HSFC_FDBC);
ichspi.c:                hsfc |= HSFC_FGO; /* start */
```

- **If you don't account for hw/sw sequencing's FSMIE bit (as no previous software did), you will just lose and provide false assurances of a lack of BIOS compromise**

Smite'em vs. Copernicus 2

How can we defeat Smite'em?

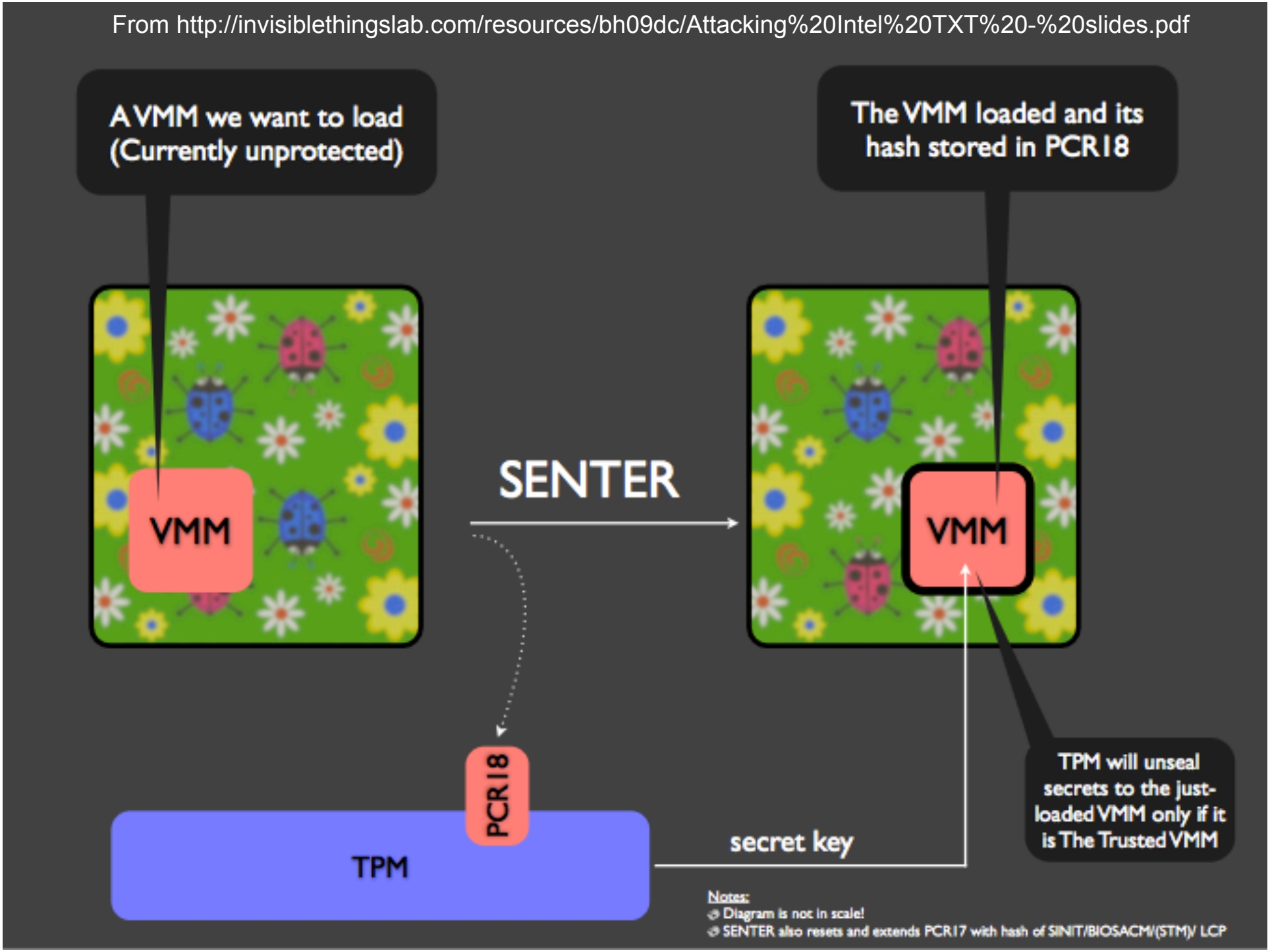
- **Smite'em lives in SMM, let's disable SMIs**
- **But its not sufficient to just disable them from an OS driver, because an attacker could just nop out our code to do so**

- **A side effect of Intel TXT is that it disables SMIs**
  - "The ILP must re-enable SMIs which were disabled as part of the SENTER process"
- **So lets learn about Intel Trusted Execution Technology (TXT)**
  - Called "Safer Mode Extensions" (SMX) in the Intel manuals

Intel Trusted Execution Technology (TXT)

- **Dynamic Root of Trust for Measurement**
- **A means to provide "late launch" trust**
  - You had a presumed-compromised system, you start TXT, and you're left in a state you setup and that you can trust

From http://invisiblethingslab.com/resources/bh09dc/Attacking%20Intel%20TXT%20-%20slides.pdf



How does it work?

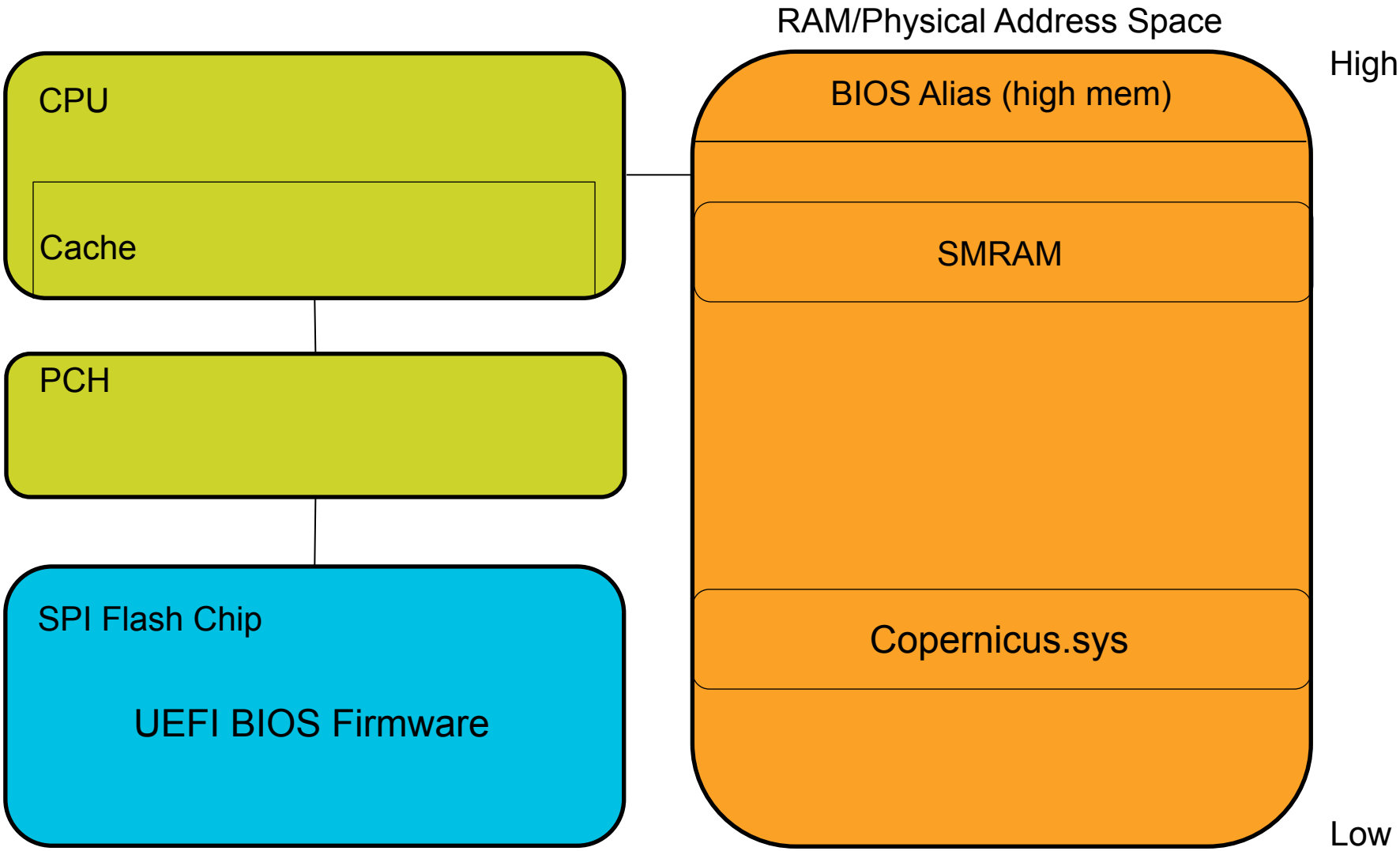
- New Intel instruction “GETSEC”
- It’s sort of like CPUID in that it’s a single instruction that does different things based on what the value is in the EAX register at the time that it’s called
- EAX = 0; GETSEC[CAPABILITIES] = report the capabilities
- EAX = 1; GETSEC[ENTERACCS] = run authenticated code (AC)
- EAX = 2; GETSEC[EXITAC] = stop running AC
- EAX = 3; GETSEC[*SENTER*] = Start a Measured Launch Environment (MLE) – this is the main one we care about, and the source of the title of this talk
- EAX = 4; GETSEC[SEXIT] = exit MLE
- EAX = 5; GETSEC[PARAMETERS] = reports supported AC info
- EAX = 6; GETSEC[SMCTRL] = *turn on SMIs*
- EAX = 7; GETSEC[WAKEUP] = wake up sleeping processors

We're only interested in a subset

- **We have to use GETSEC[CAPABILITIES] and GETSEC[PARAMETERS] just for sanity checking purposes**
- **We mainly care about SENTER and SEXIT to start and stop our MLE**
- **We're *NOT* going to use SMCTRL or WAKEUP**
  - The whole point here is to freeze SMM code in place

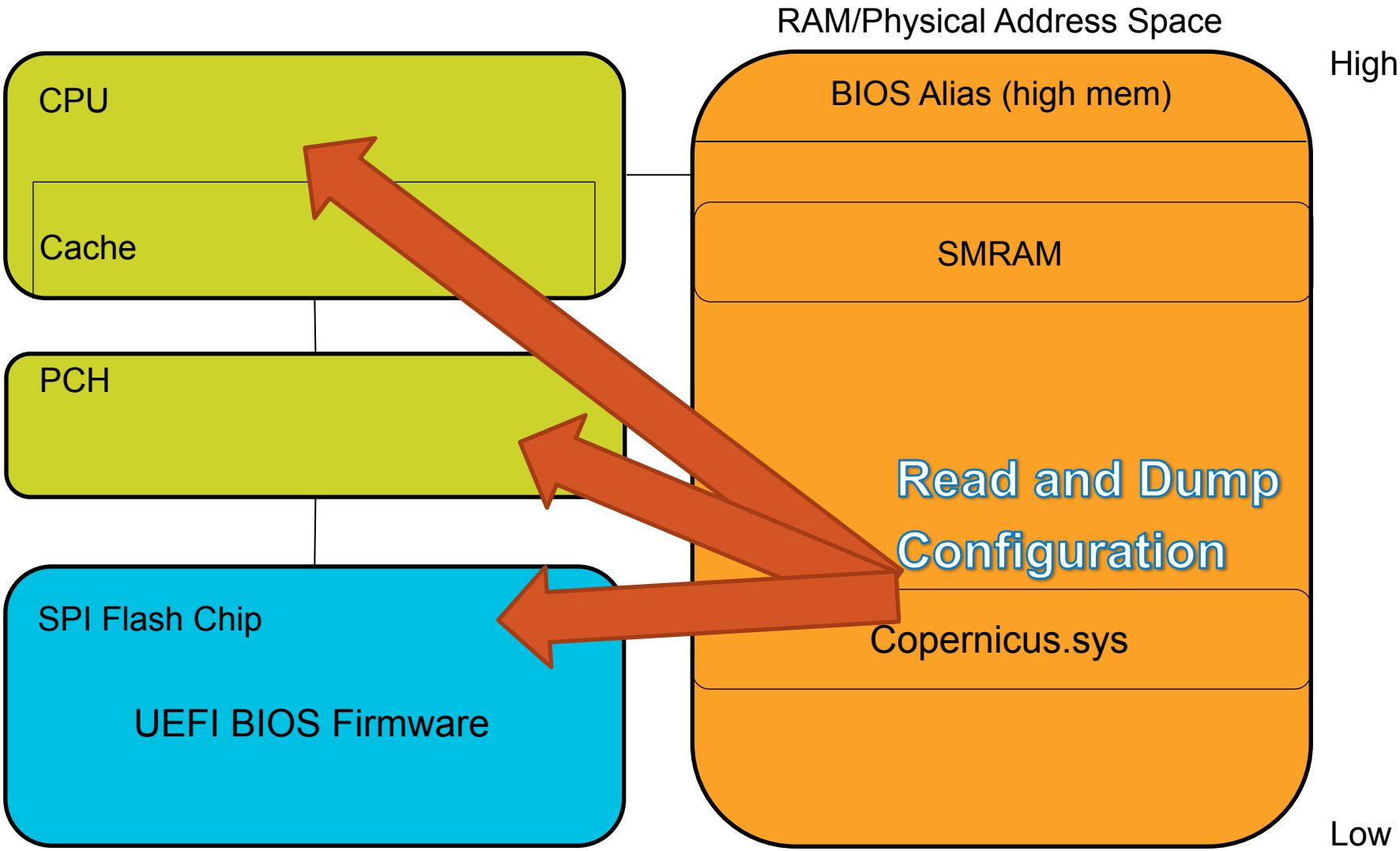
SENTER THE DRAGON!

Copernicus 1 Architecture



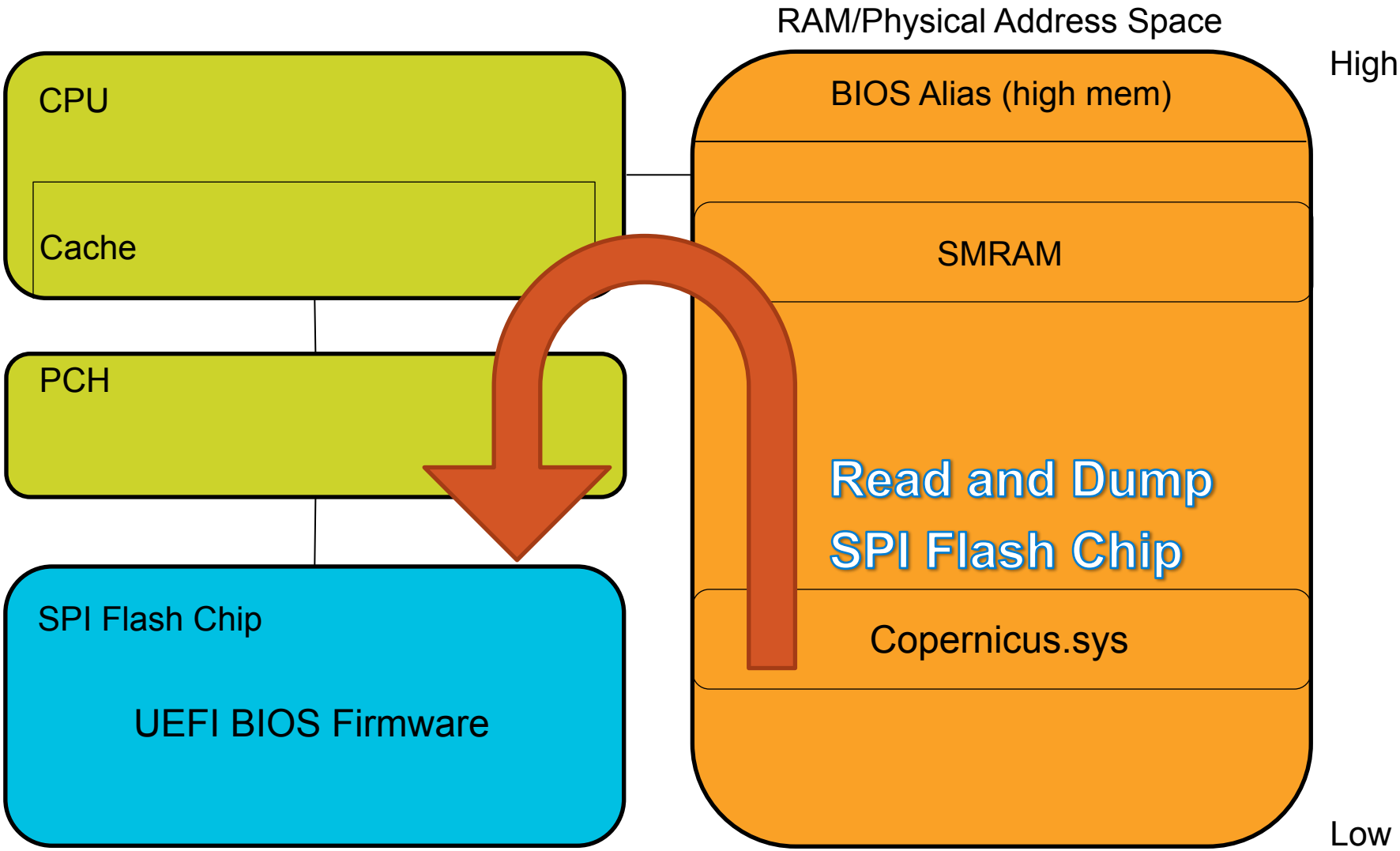
MITRE

Copernicus 1 Architecture



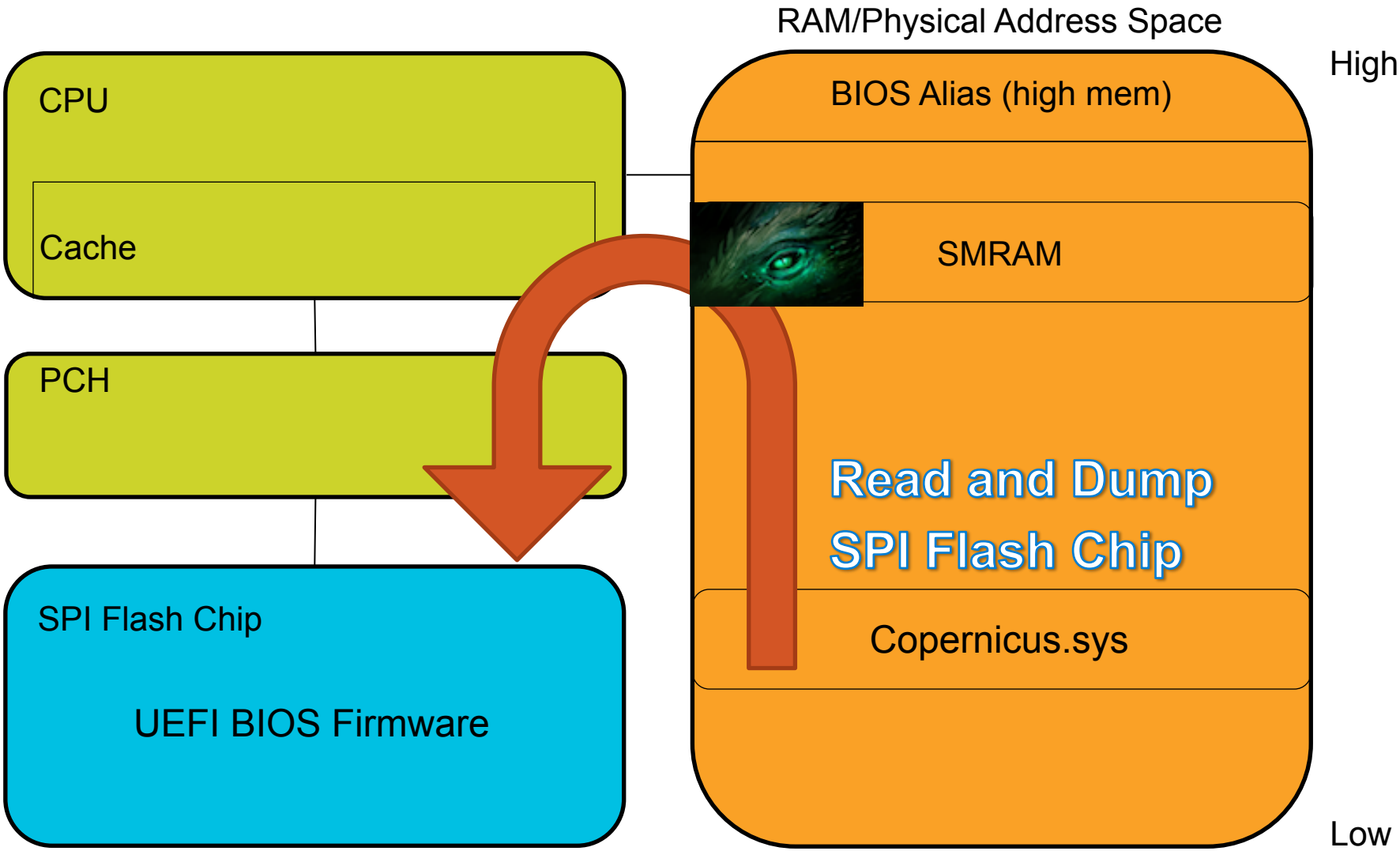
MITRE

Copernicus 1 Architecture



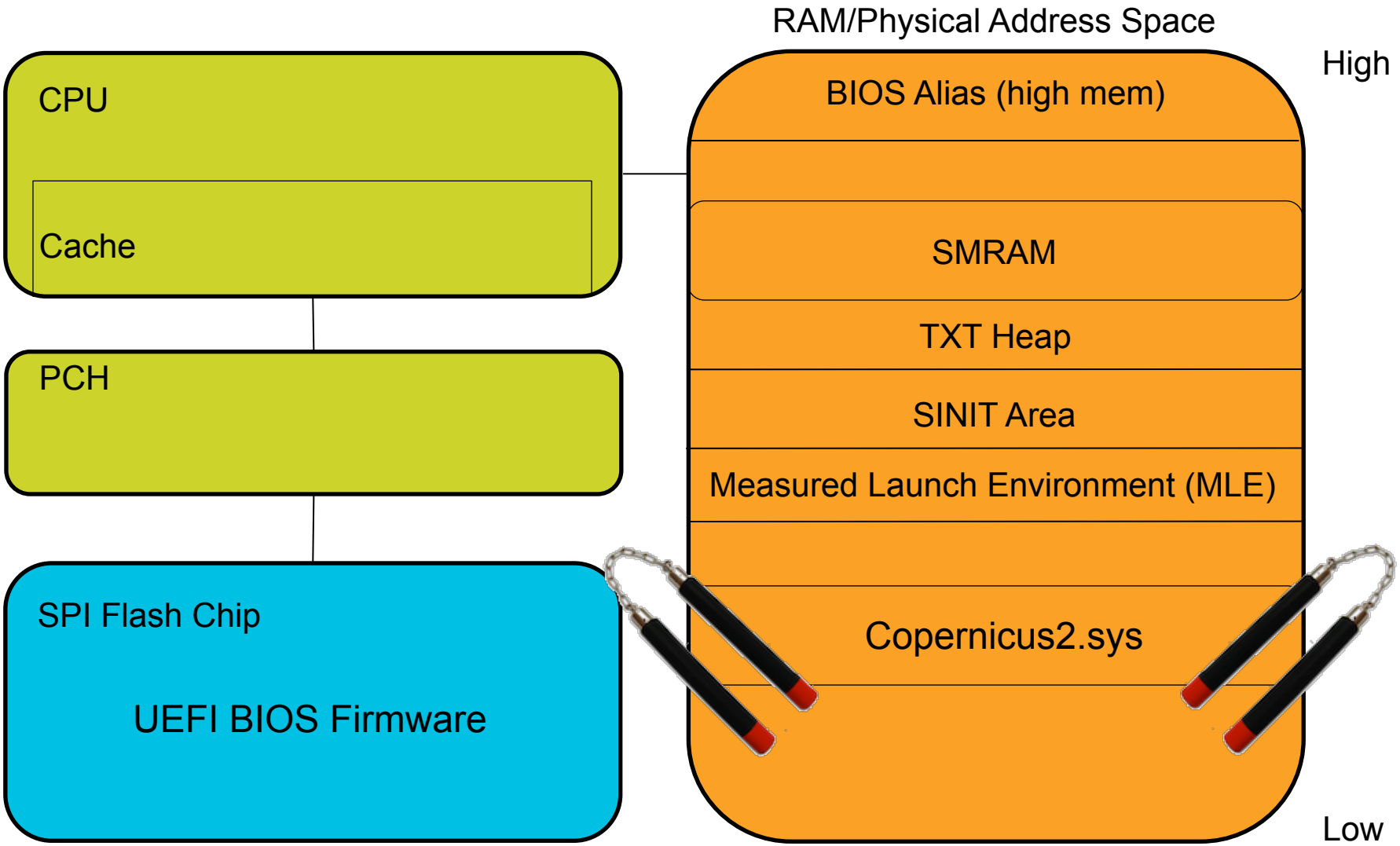
MITRE

Smite'em Attacks!



MITRE

Copernicus 2 Architecture

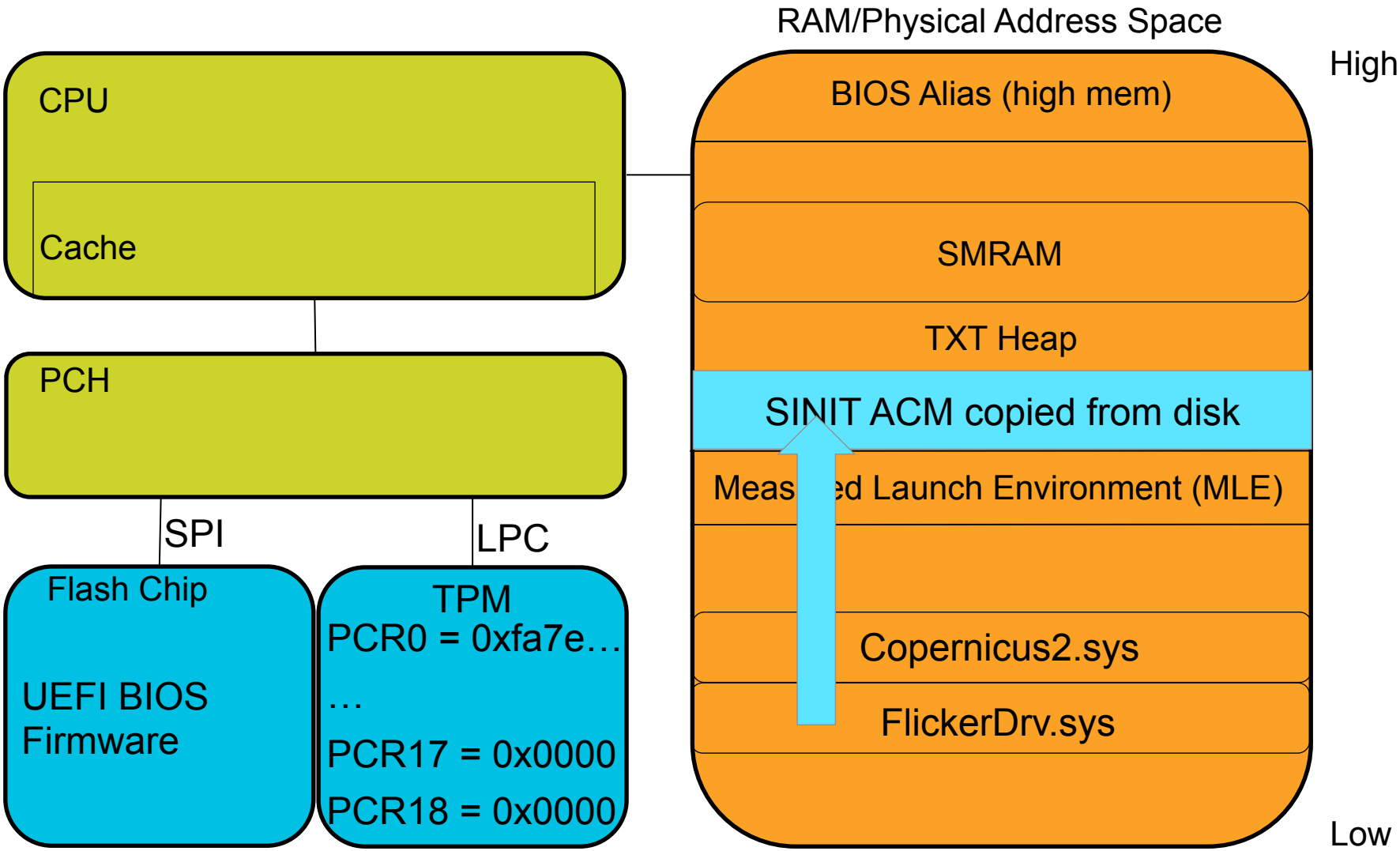


MITRE

Overall behavior 1

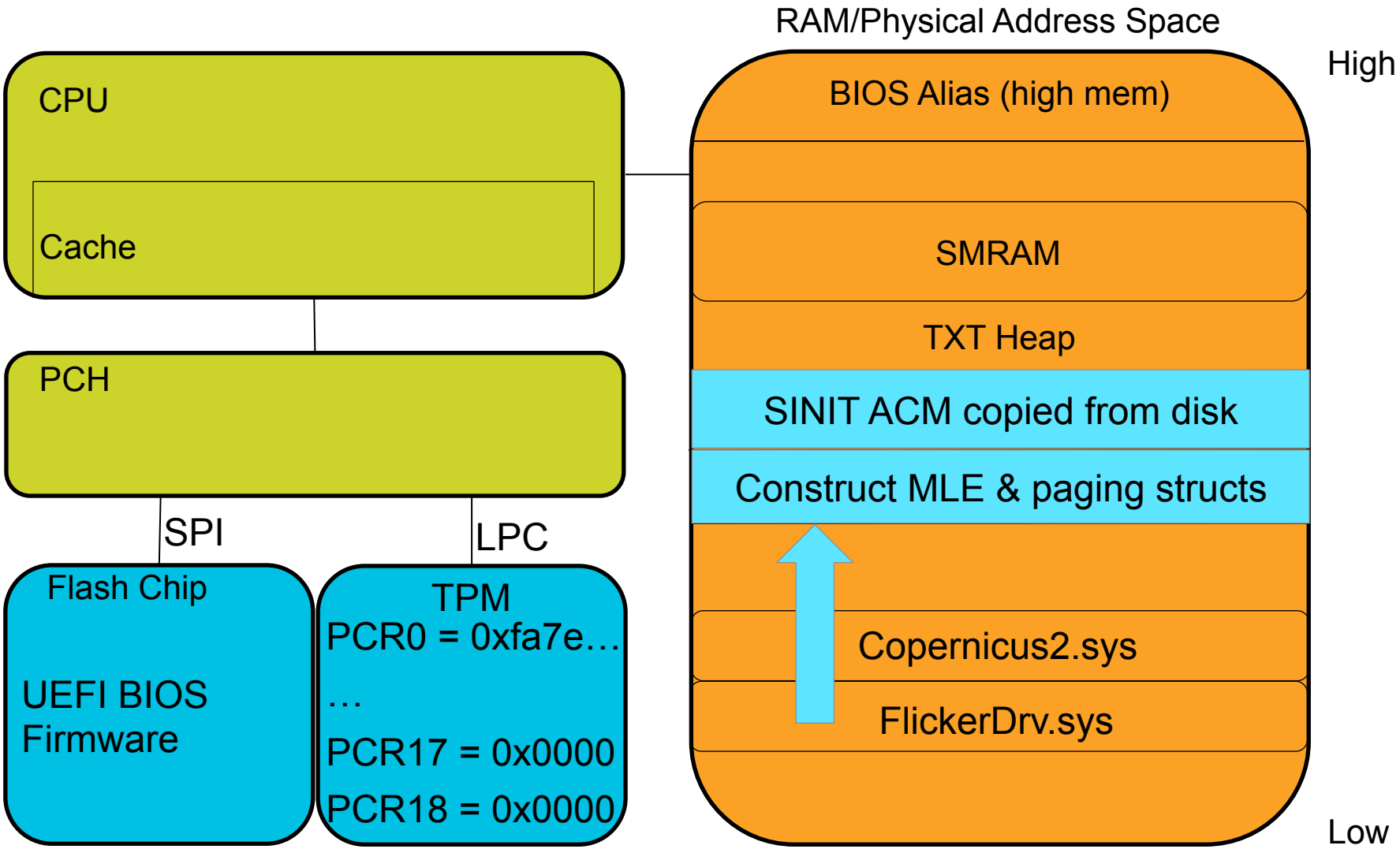
- **Initial actions:**
  - Turn on TPM & TXT
  - Provision TPM key for later signature verification
  - Load Flicker driver, pass MLE code to Flicker, tell it to start

Copernicus 2 Architecture



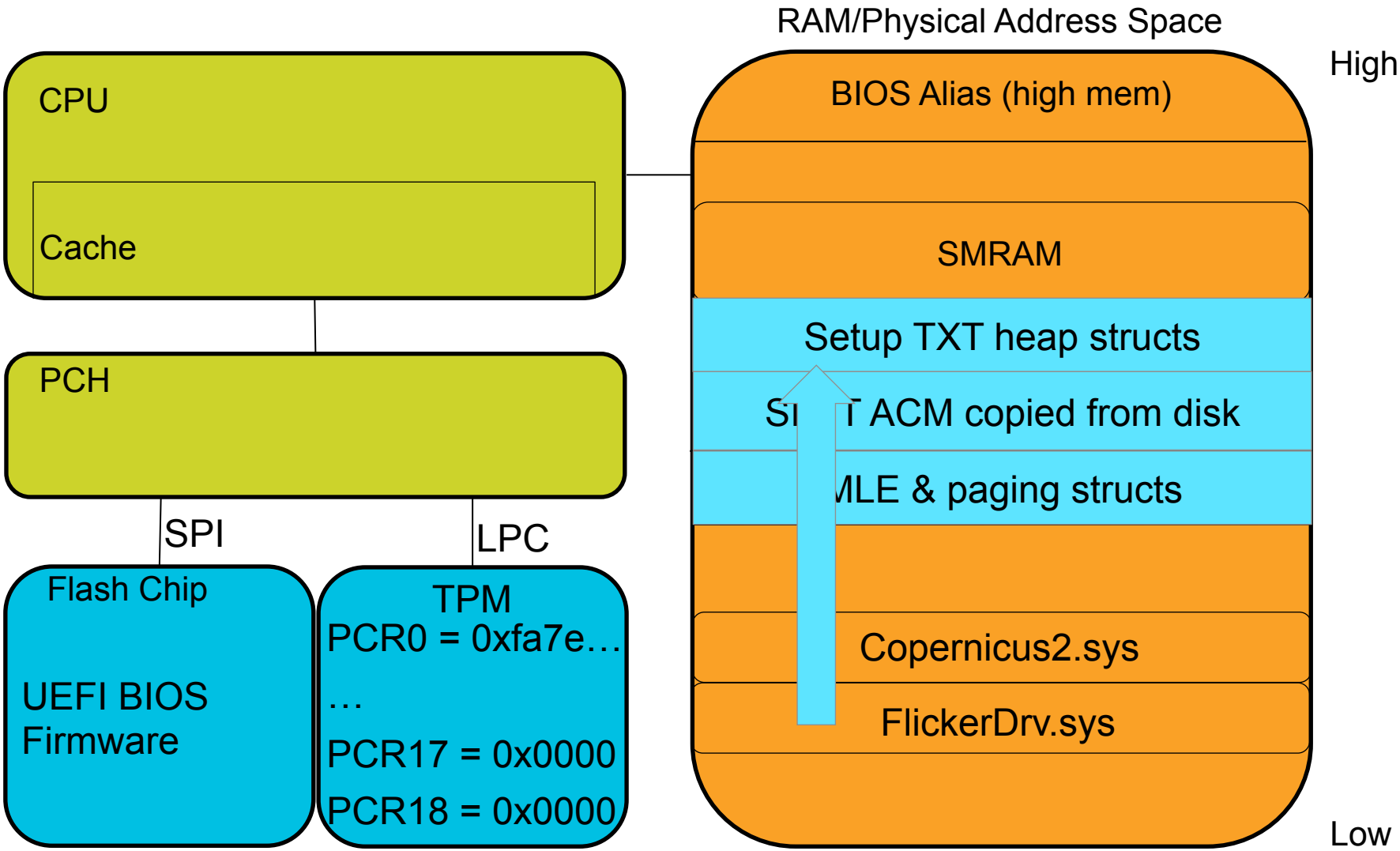
MITRE

Copernicus 2 Architecture



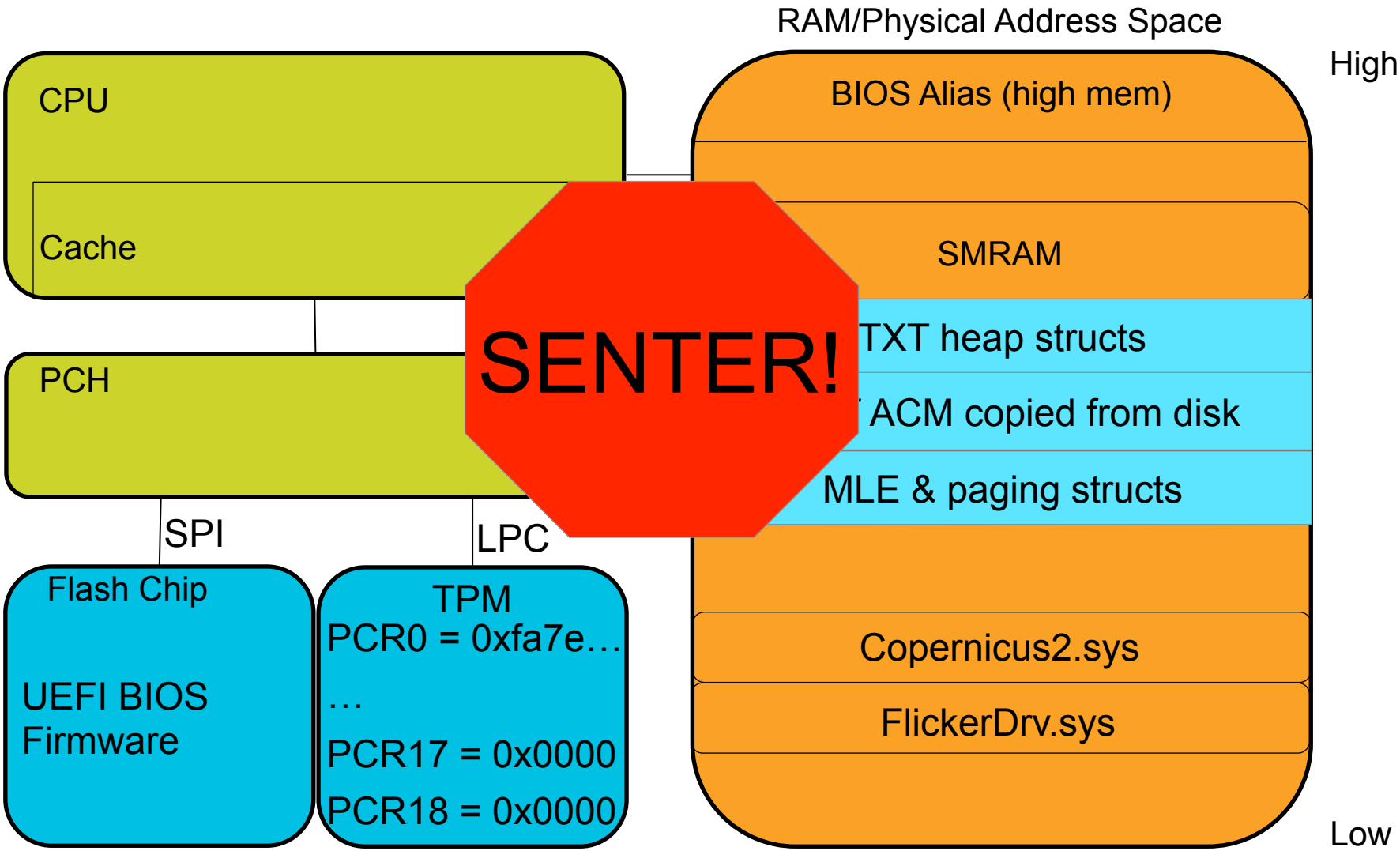
MITRE

Copernicus 2 Architecture



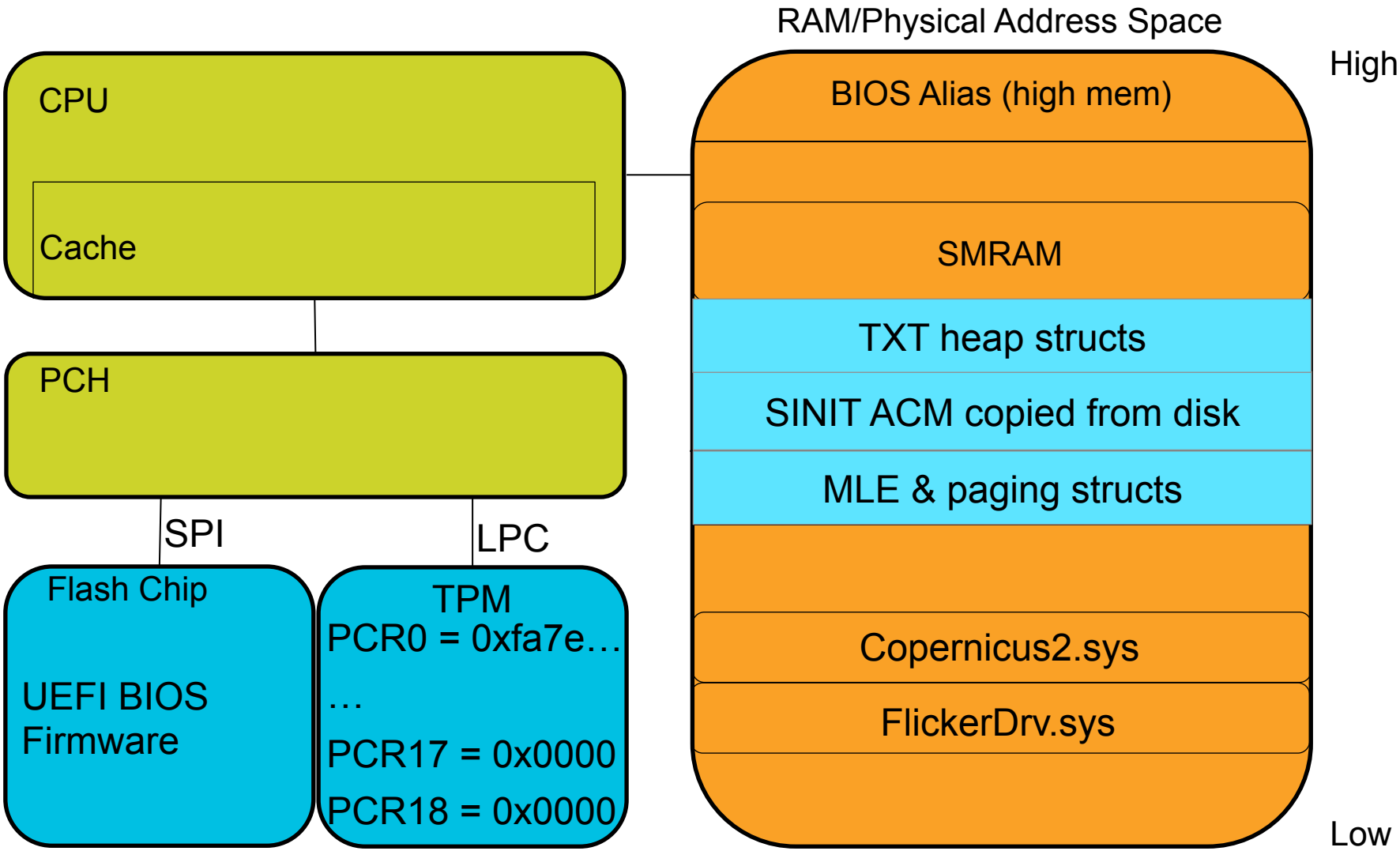
MITRE

Copernicus 2 Architecture



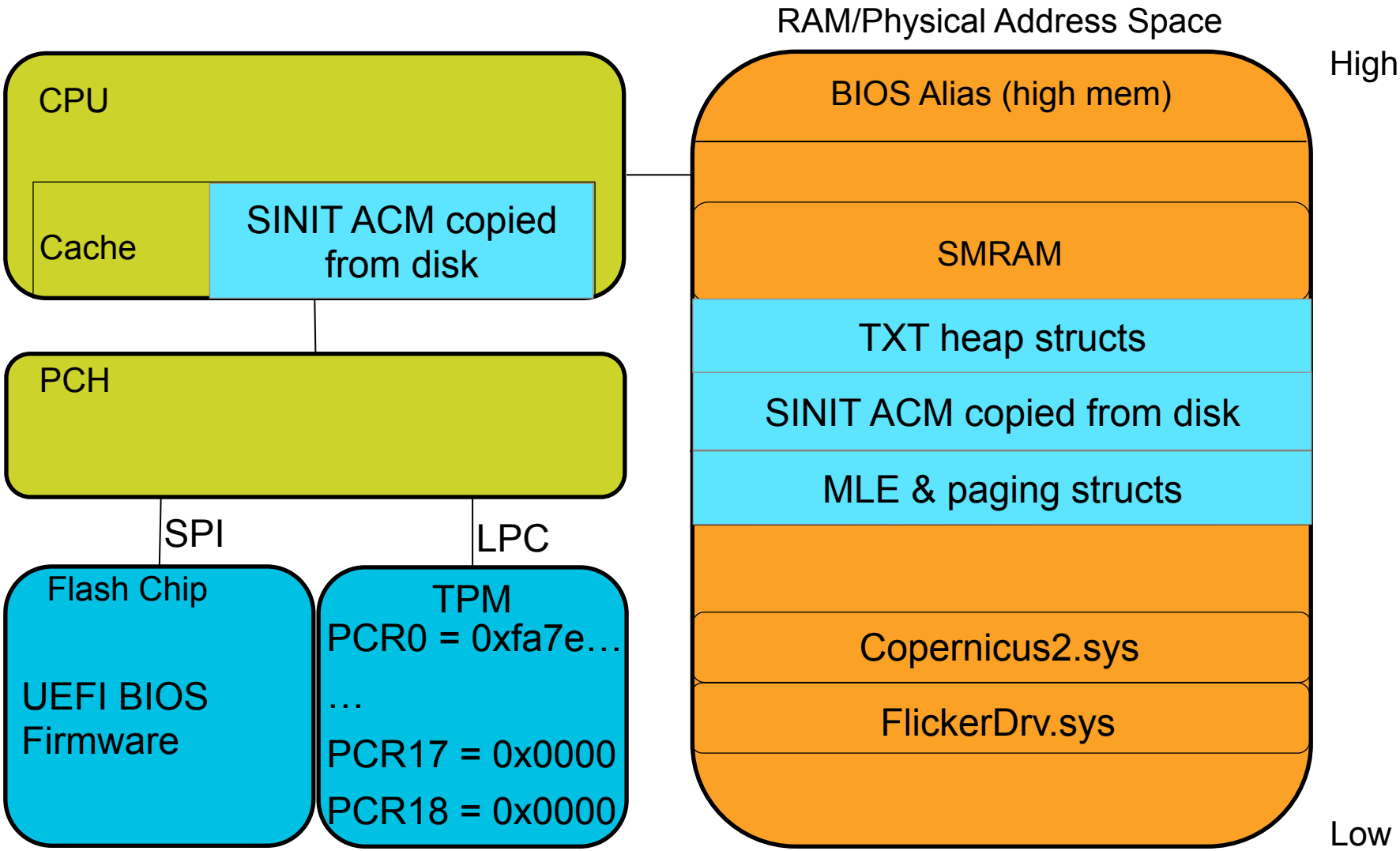
MITRE

Copernicus 2 Architecture



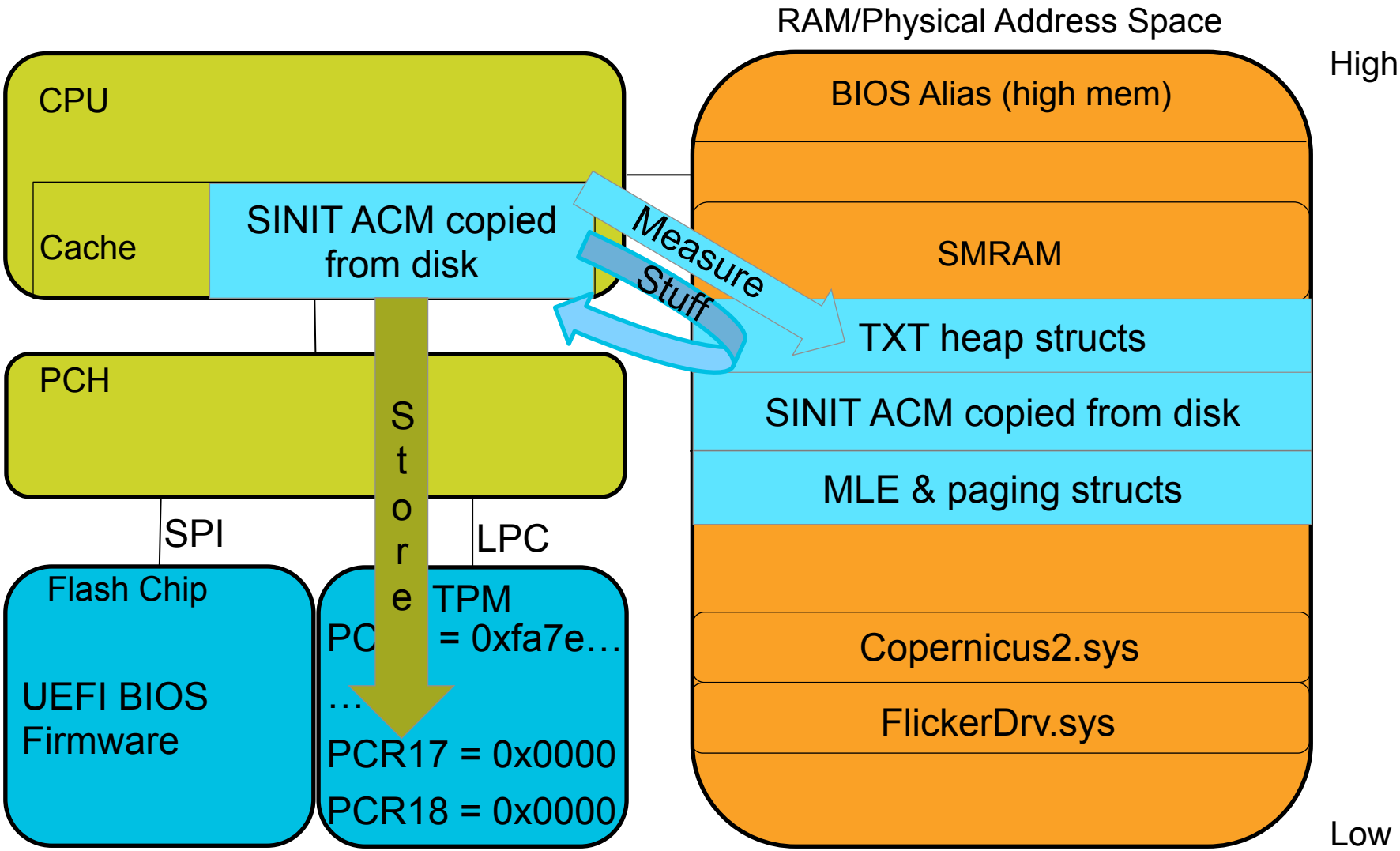
MITRE

Copernicus 2 Architecture



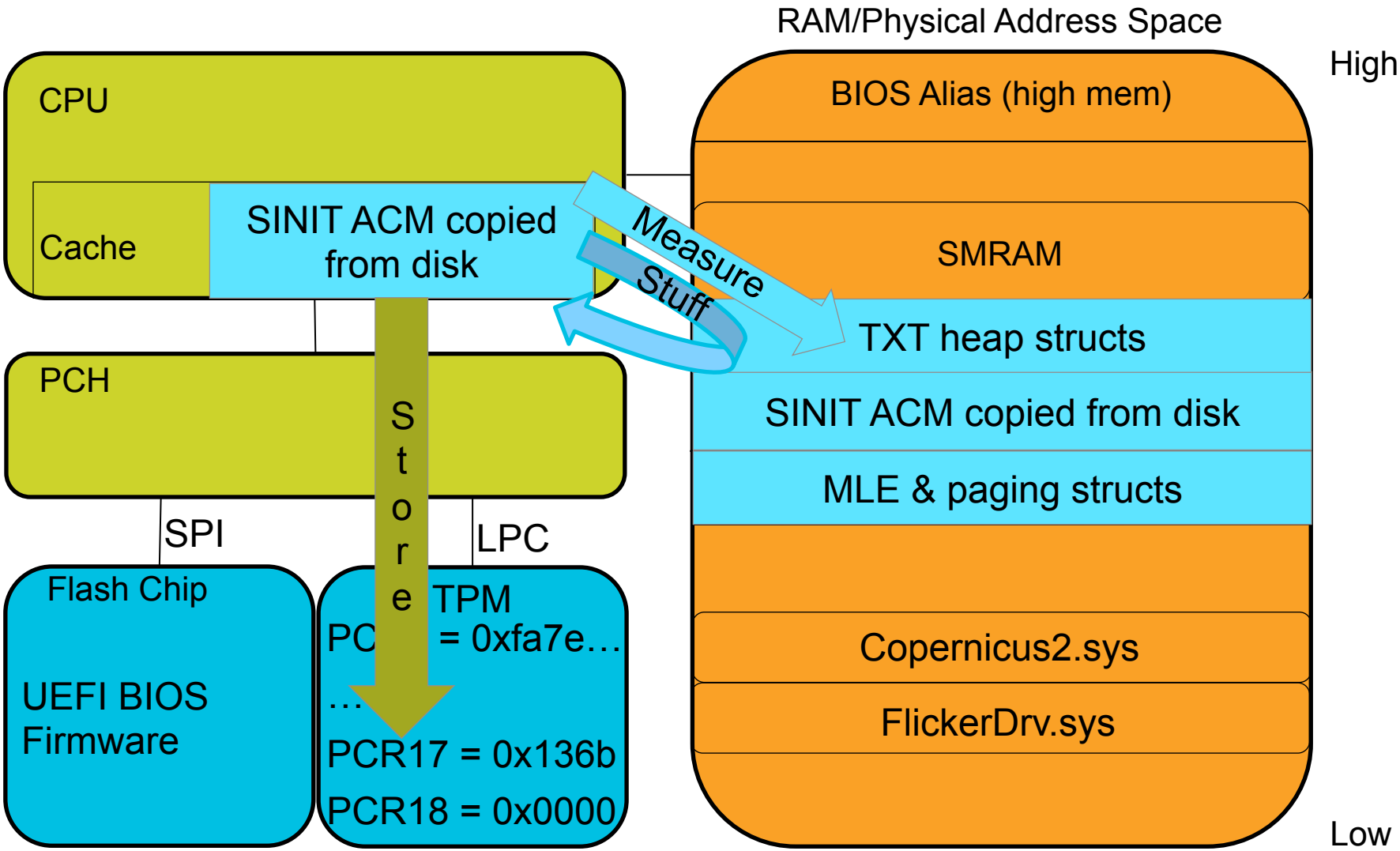
MITRE

Copernicus 2 Architecture



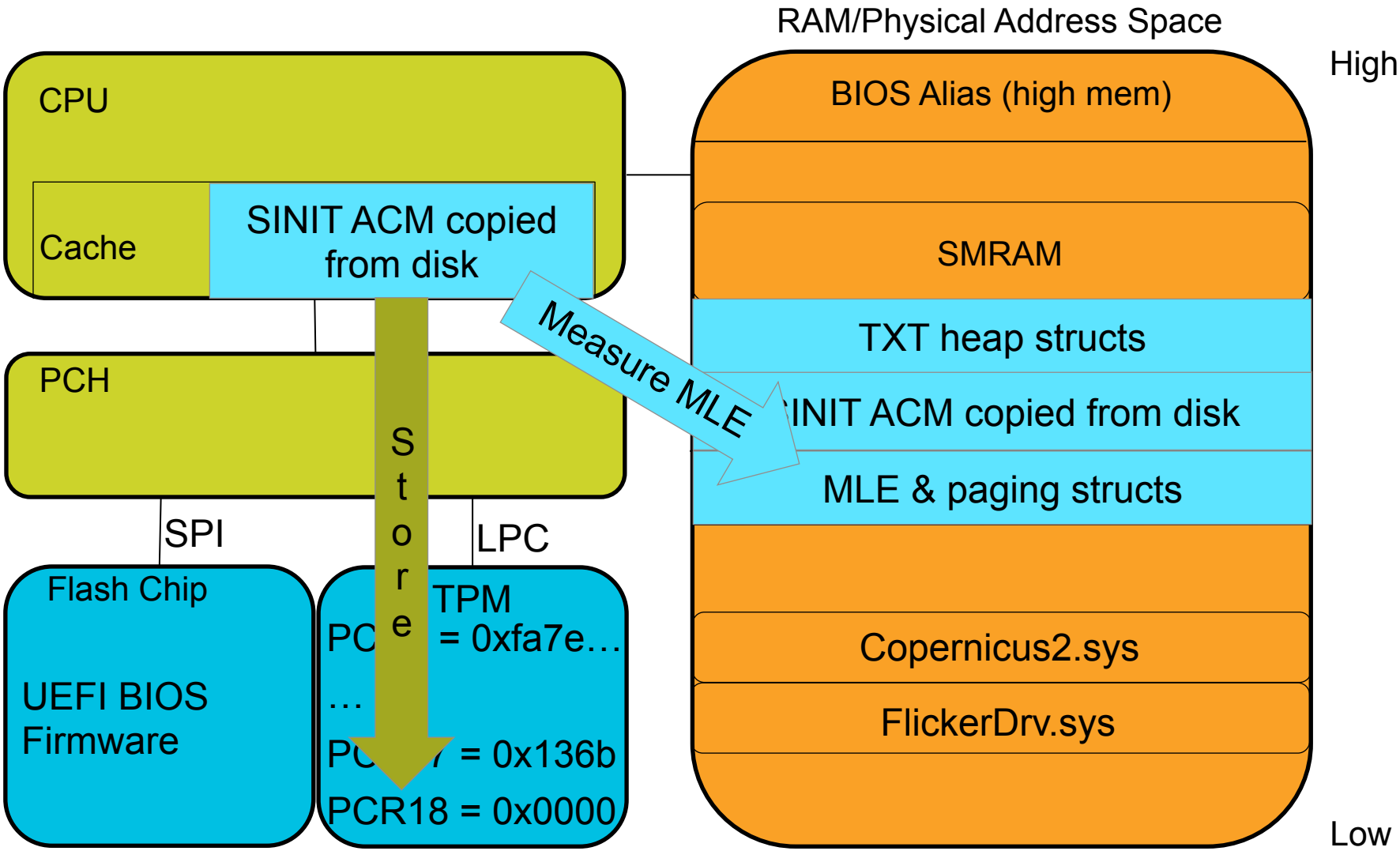
MITRE

Copernicus 2 Architecture



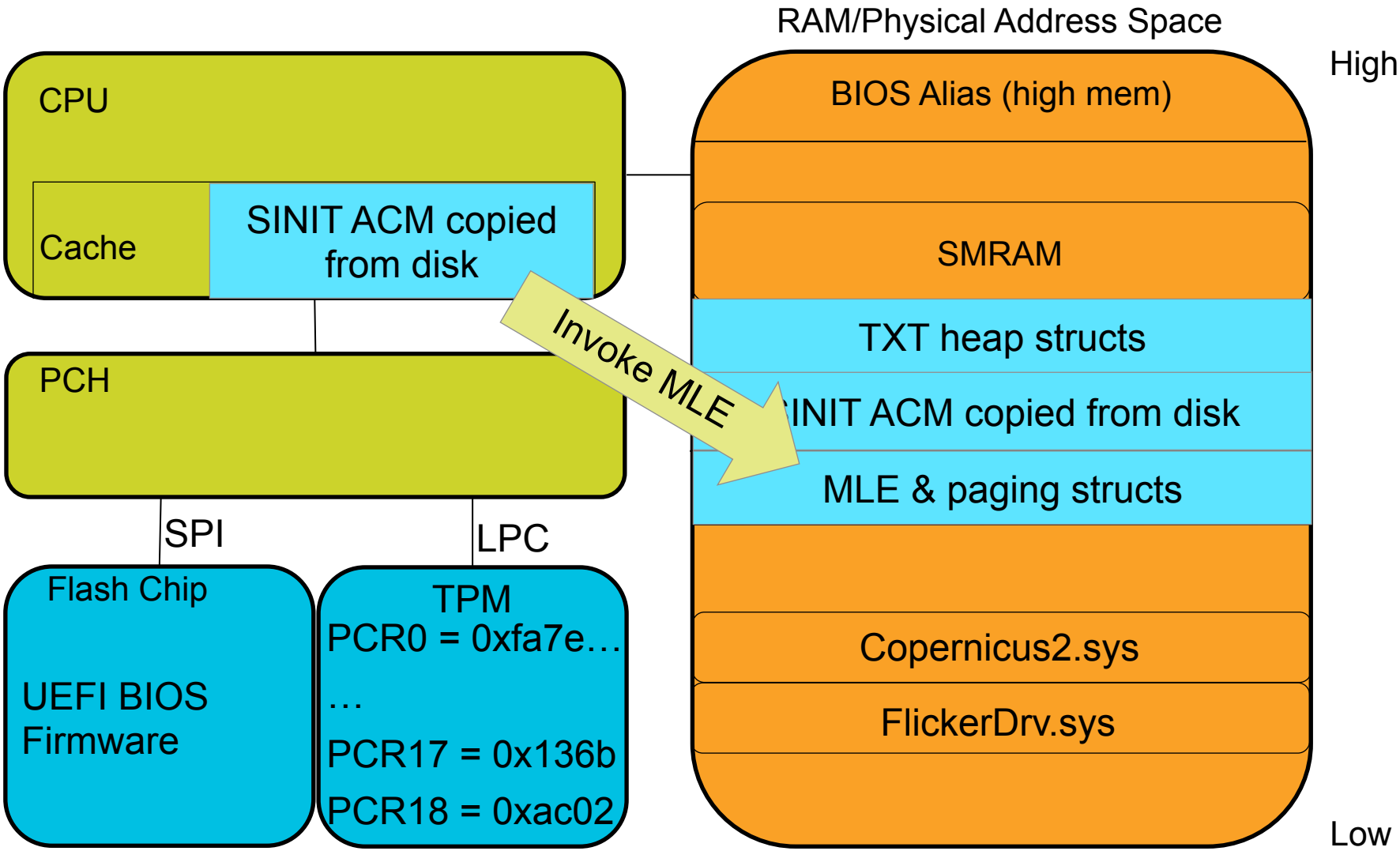
MITRE

Copernicus 2 Architecture



MITRE

Copernicus 2 Architecture

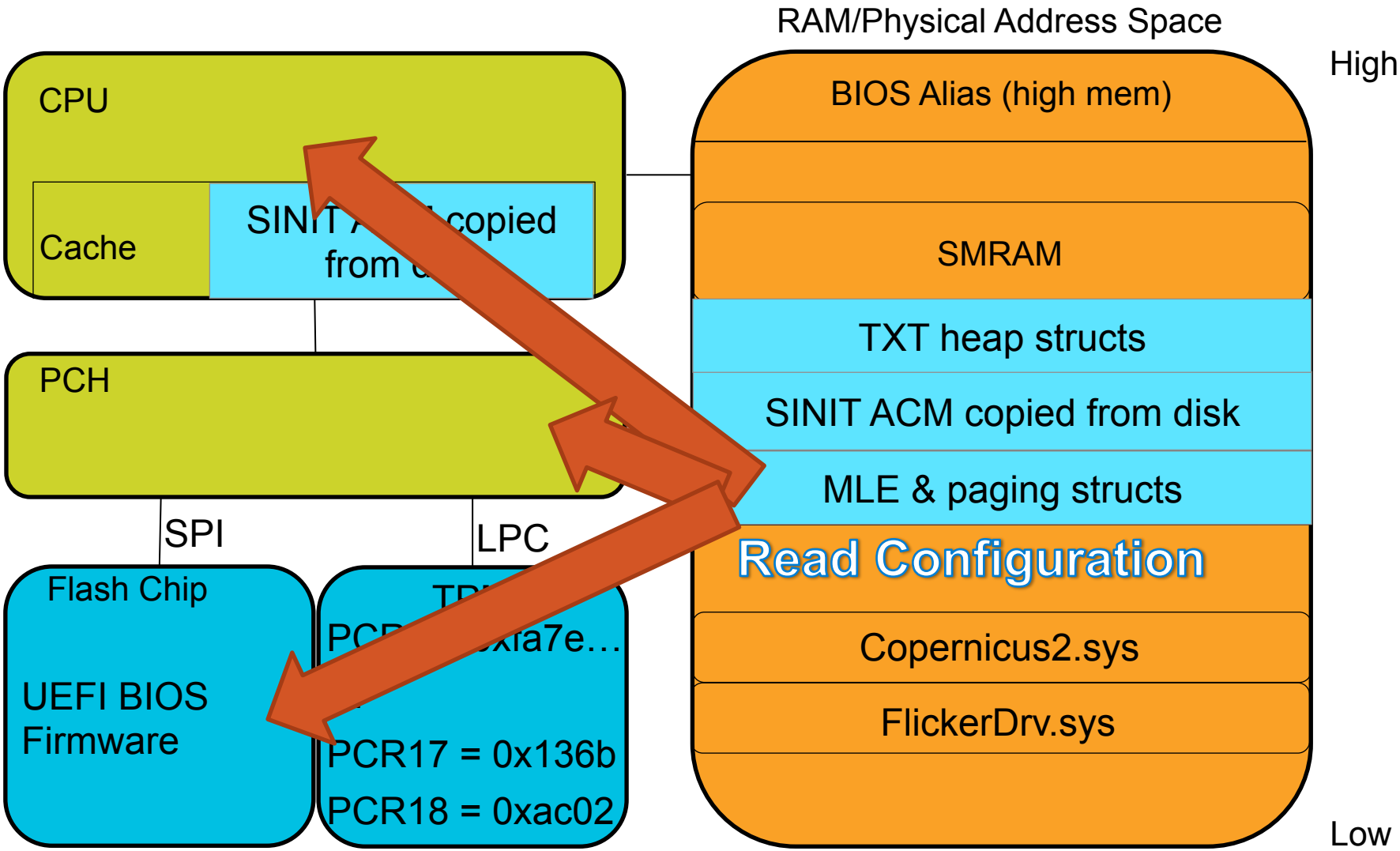


MITRE

Overall behavior 2

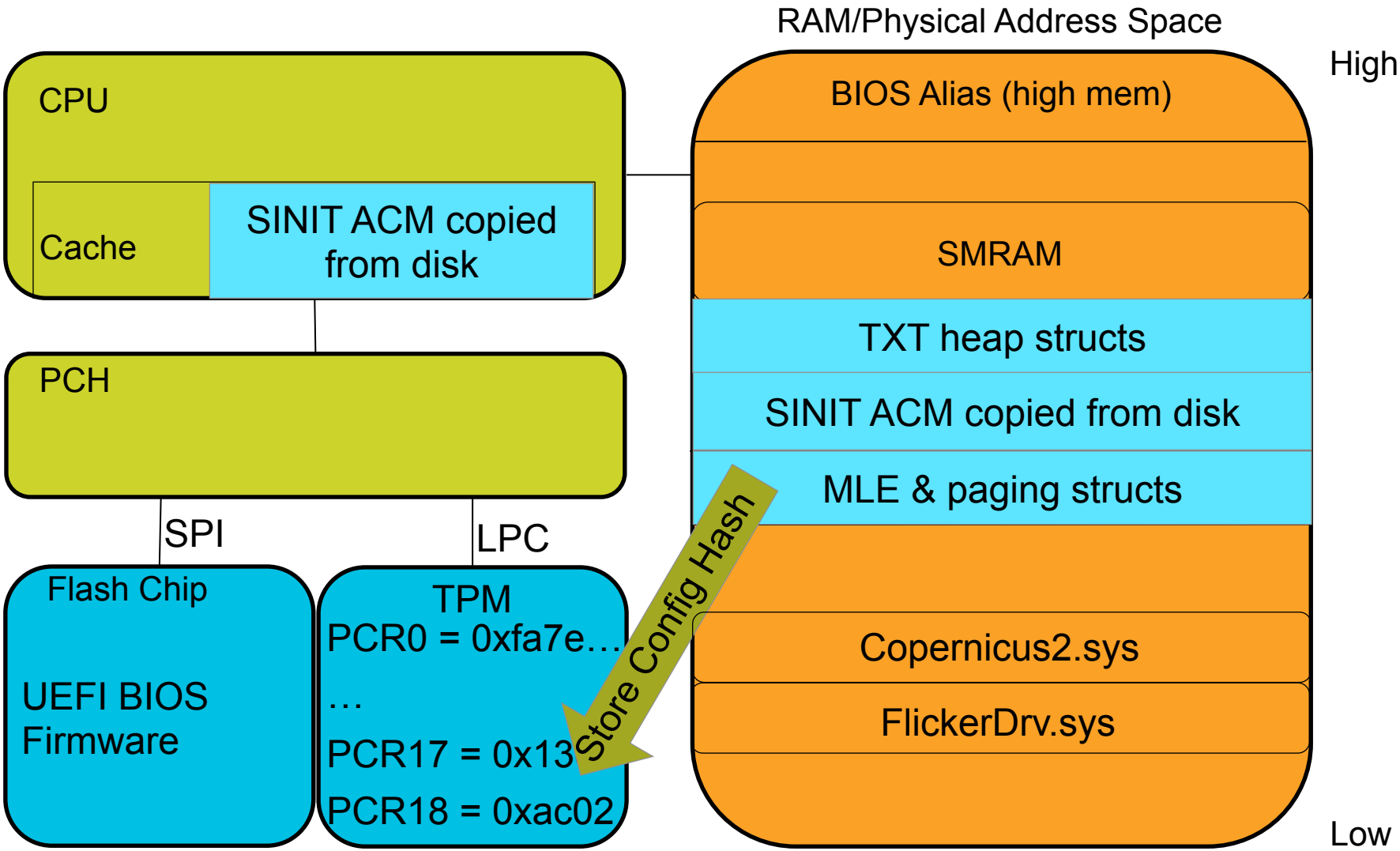
- **MLE actions:**
  - Read config info, place text in buffer, SHA1 hash it, extend buffer into TPM Platform Configuration Register (PCR) 18
  - Read BIOS 0x10000 at a time, SHA1 hash it, extend buffer into PCR 18
  - SEXIT

Copernicus 2 Architecture



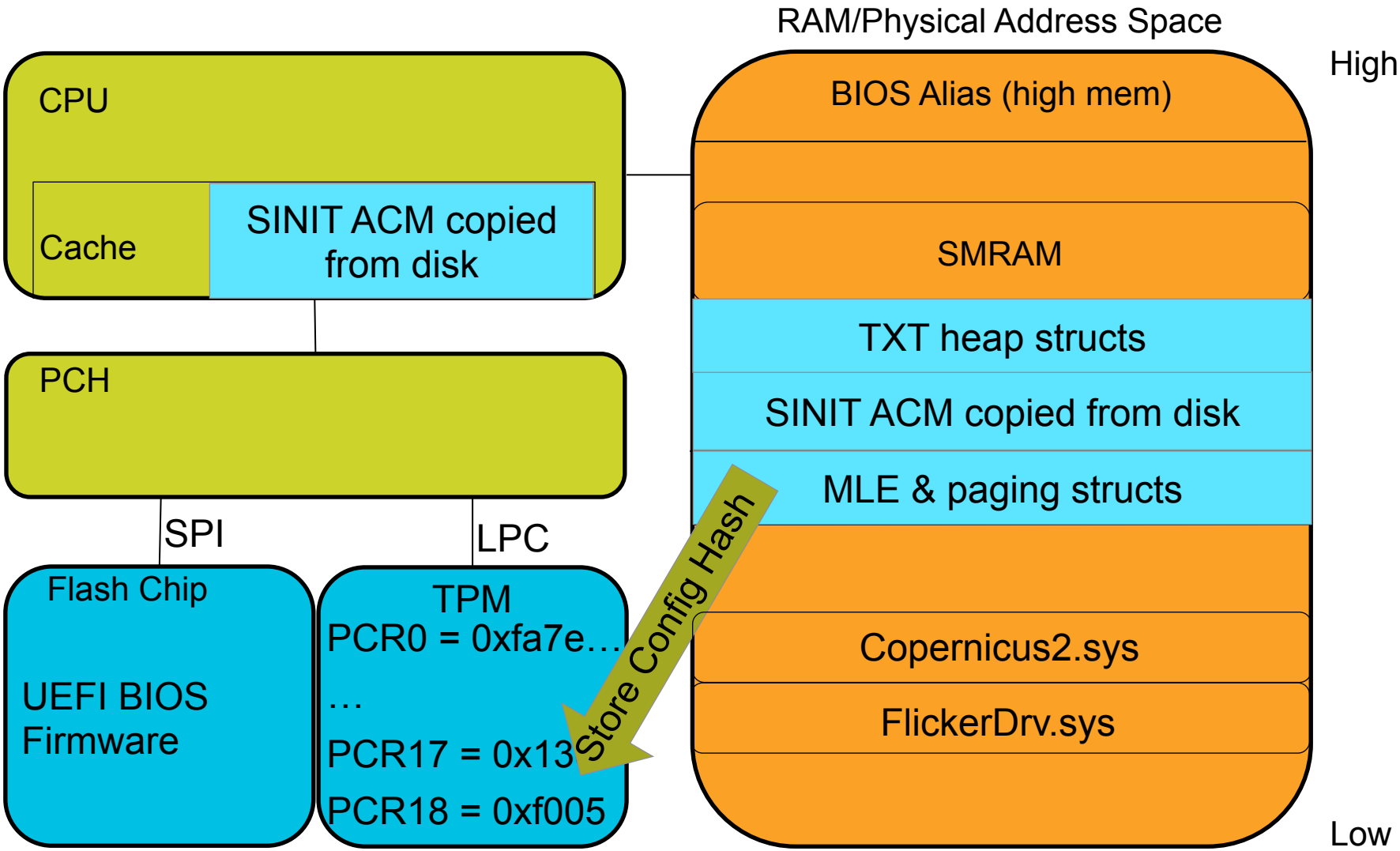
MITRE

Copernicus 2 Architecture



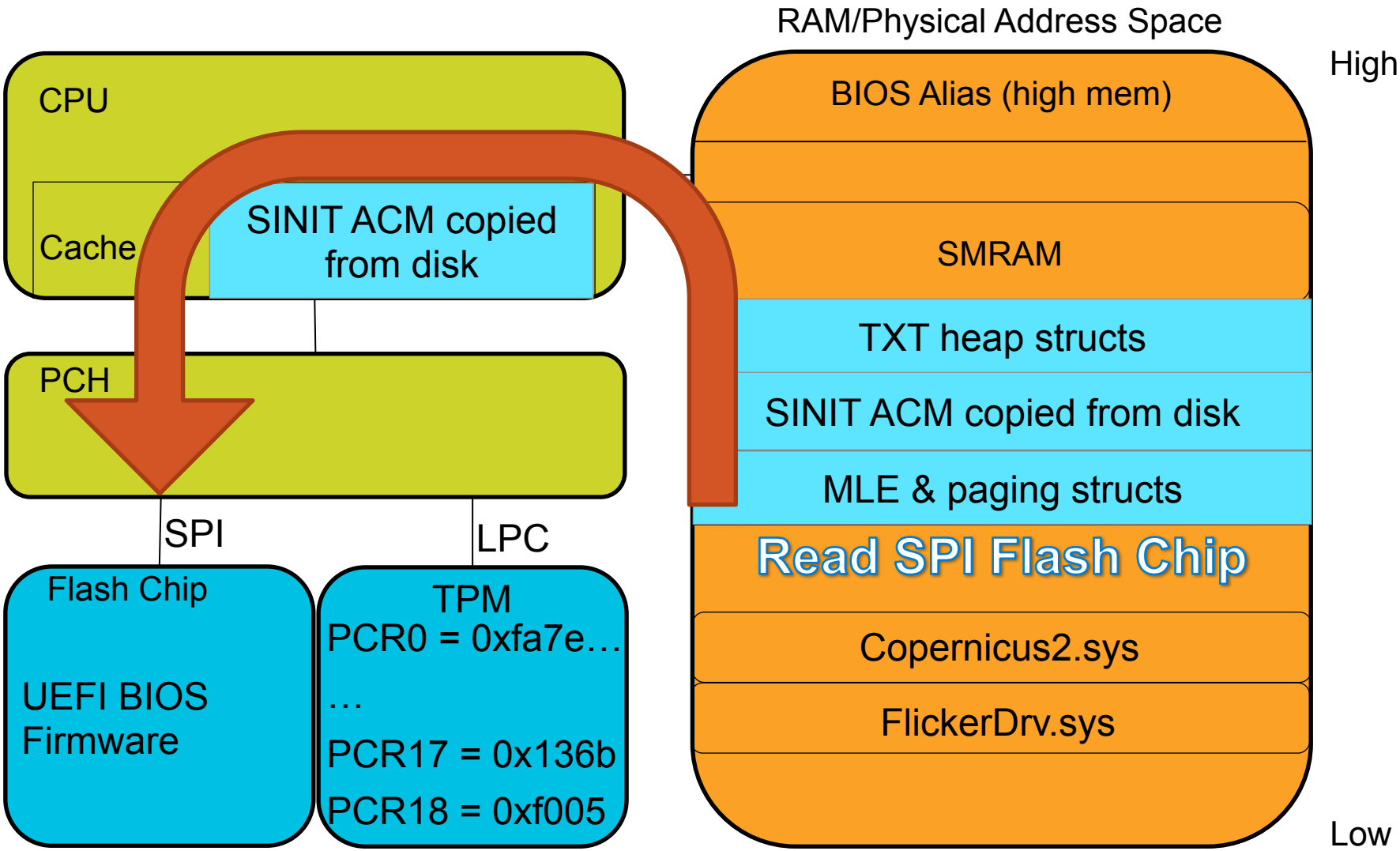
MITRE

Copernicus 2 Architecture



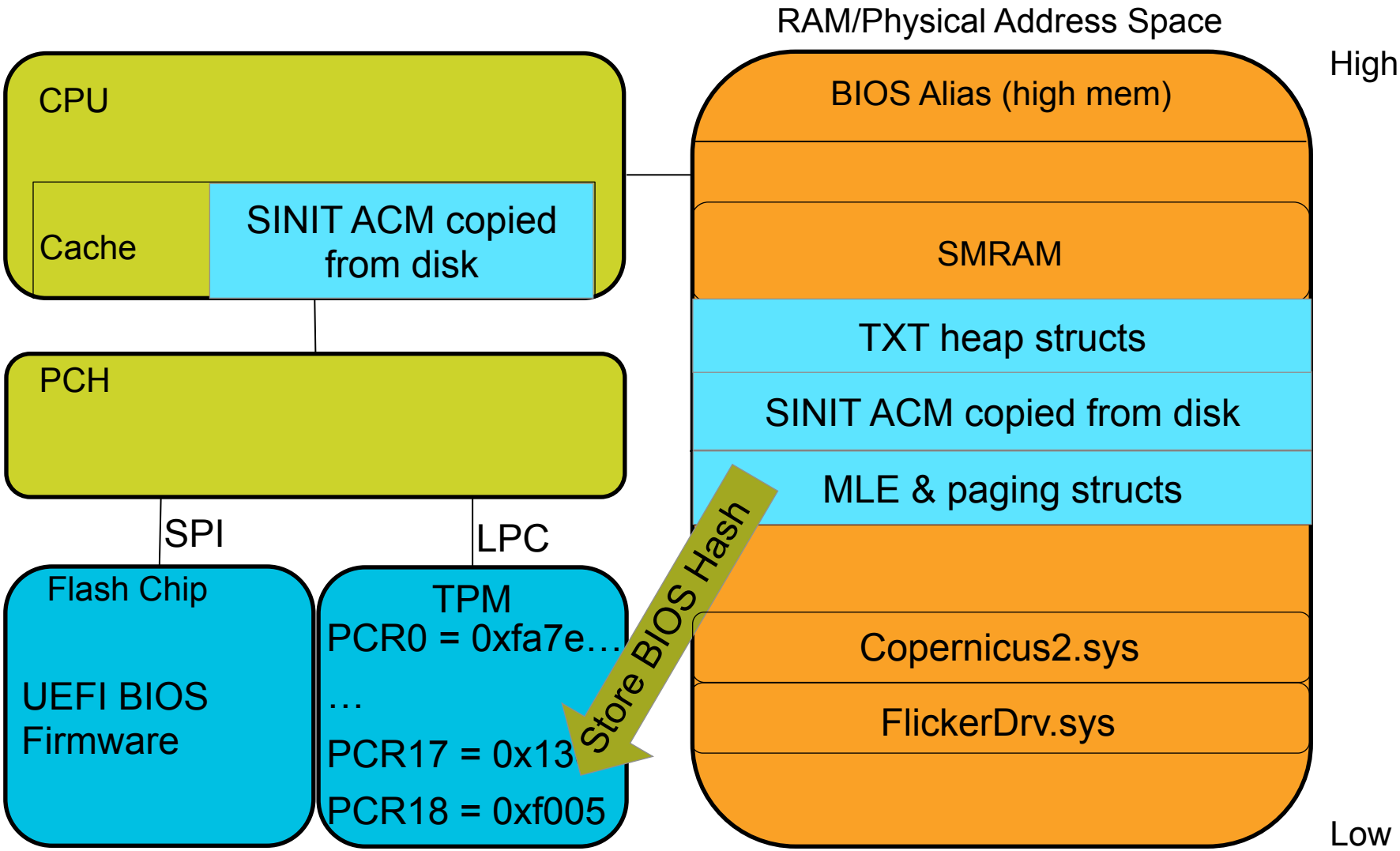
MITRE

Copernicus 2 Architecture



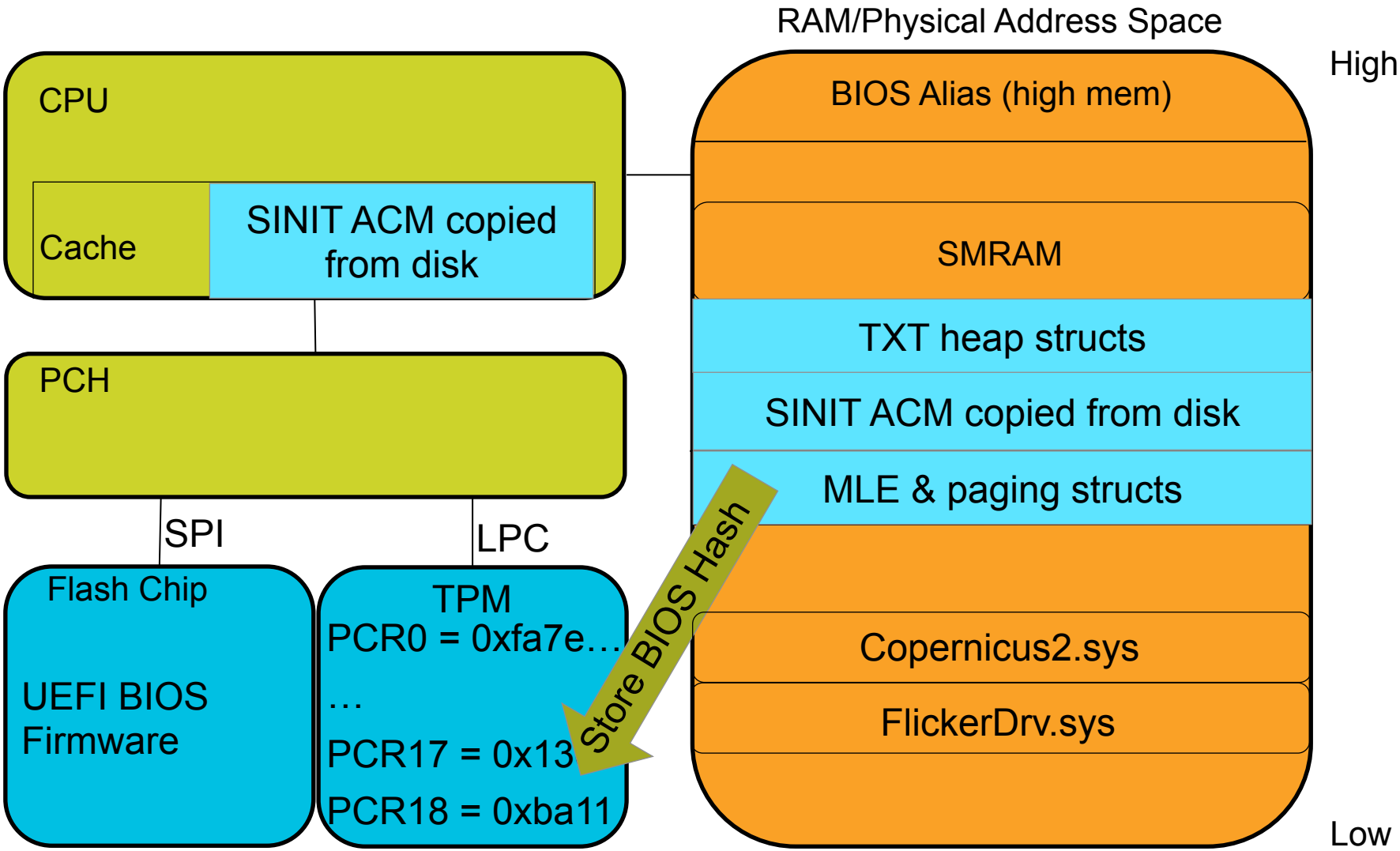
MITRE

Copernicus 2 Architecture



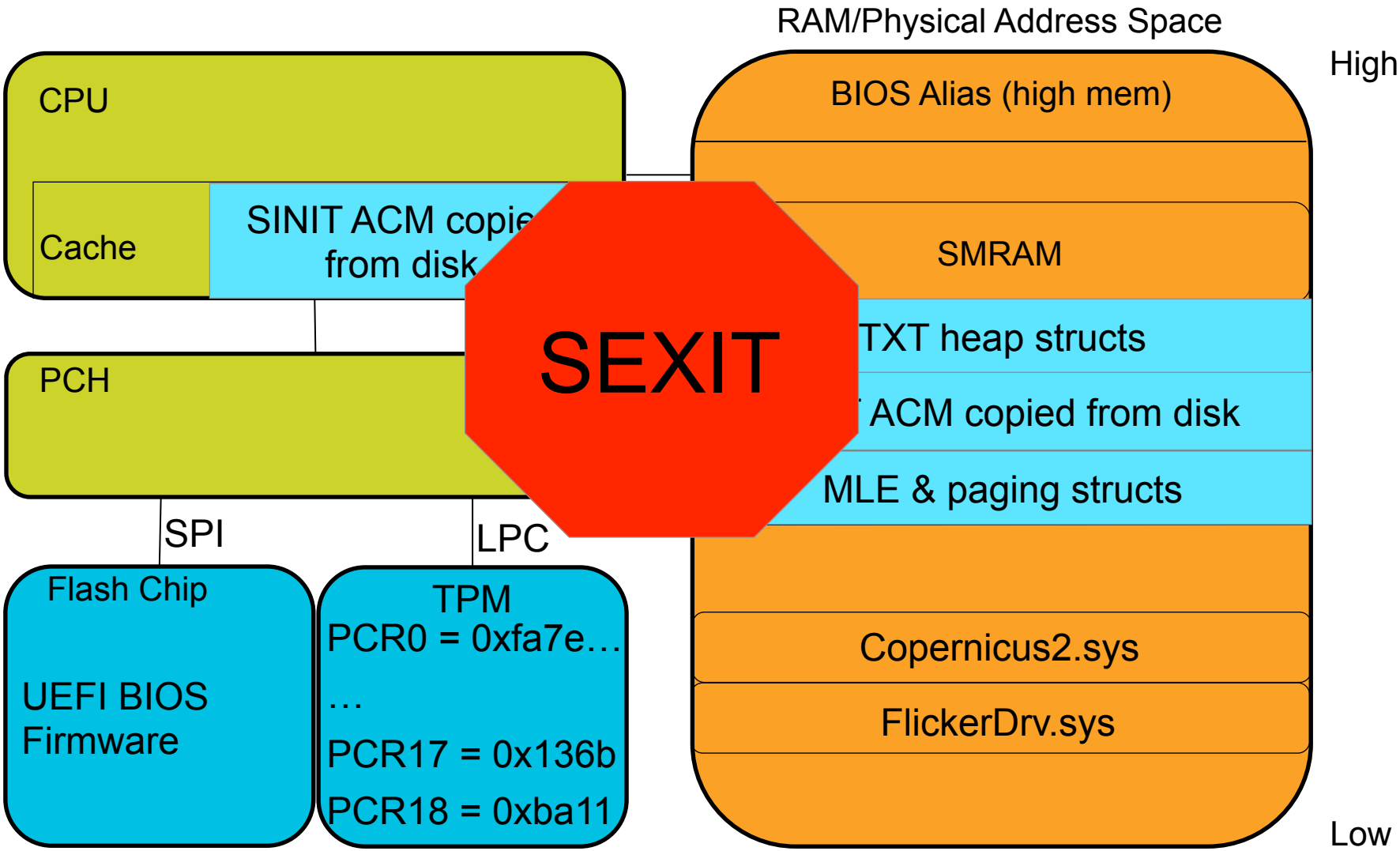
MITRE

Copernicus 2 Architecture



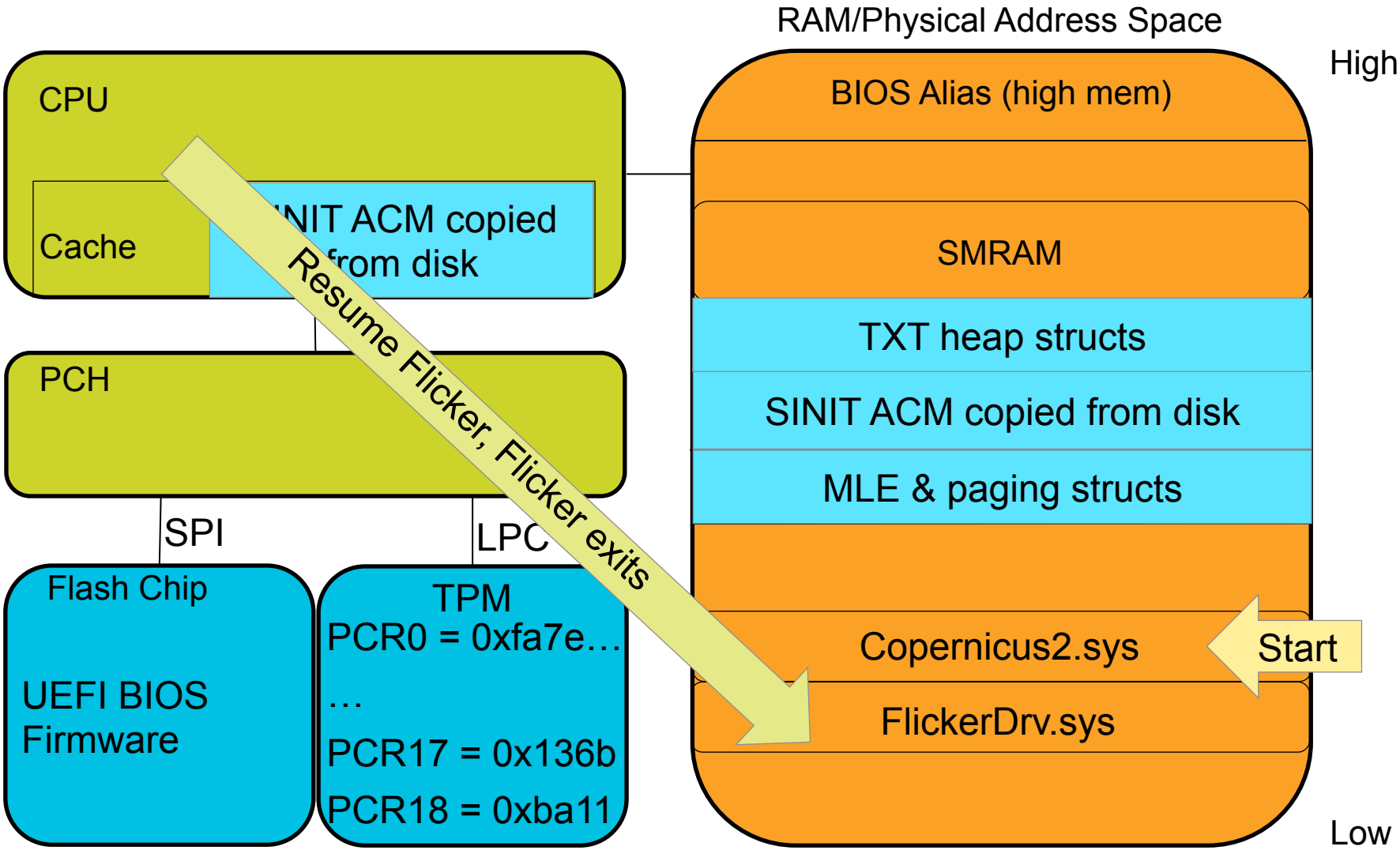
MITRE

Copernicus 2 Architecture



MITRE

Copernicus 2 Architecture



MITRE

Charizard

MITRE

Intel SPI Flash Protection Mechanisms

- **Intel provides a number of protection mechanisms that can "lock down" the flash chip.**
- **It's up to OEMs/IBVs to use these Intel provided mechanisms in a coherent way to implement things like:**
  - UEFI variable protection
  - Signed firmware update requirement

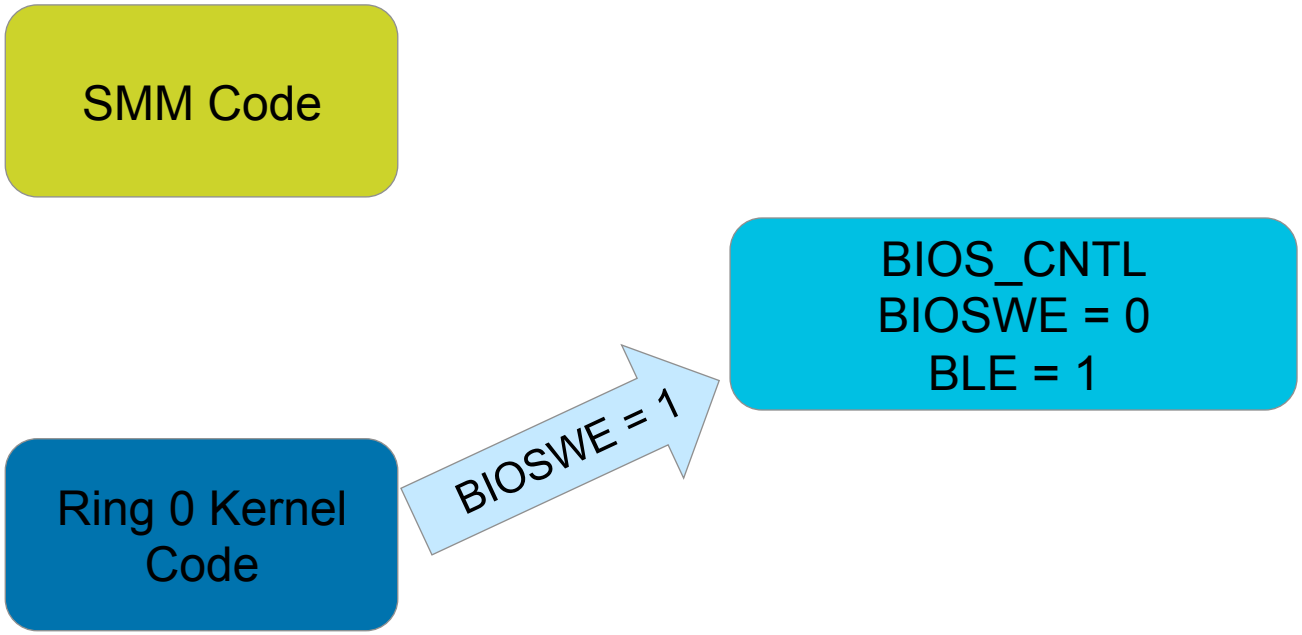
BIOS_CNTL

1	**BIOS Lock Enable (BLE)** — R/WLO. 0 = Setting the BIOSWE will not cause SMIs. 1 = Enables setting the BIOSWE bit to cause SMIs. Once set, this bit can only be cleared by a PLTRST#
0	**BIOS Write Enable (BIOSWE)** — R/W. 0 = Only read cycles result in Firmware Hub I/F cycles. 1 = Access to the BIOS space is enabled for both read and write cycles. When this bit is written from a 0 to a 1 and BIOS Lock Enable (BLE) is also set, an SMI# is generated. This ensures that only SMI code can update BIOS.

- **The above bits are part of the BIOS_CNTL register on the ICH.**
- **BIOS_CNTL.BIOSWE bit enables write access to the flash chip.**
- **BIOS_CNTL.BLE bit provides an opportunity for the OEM to implement an SMM routine to protect the BIOSWE bit.**

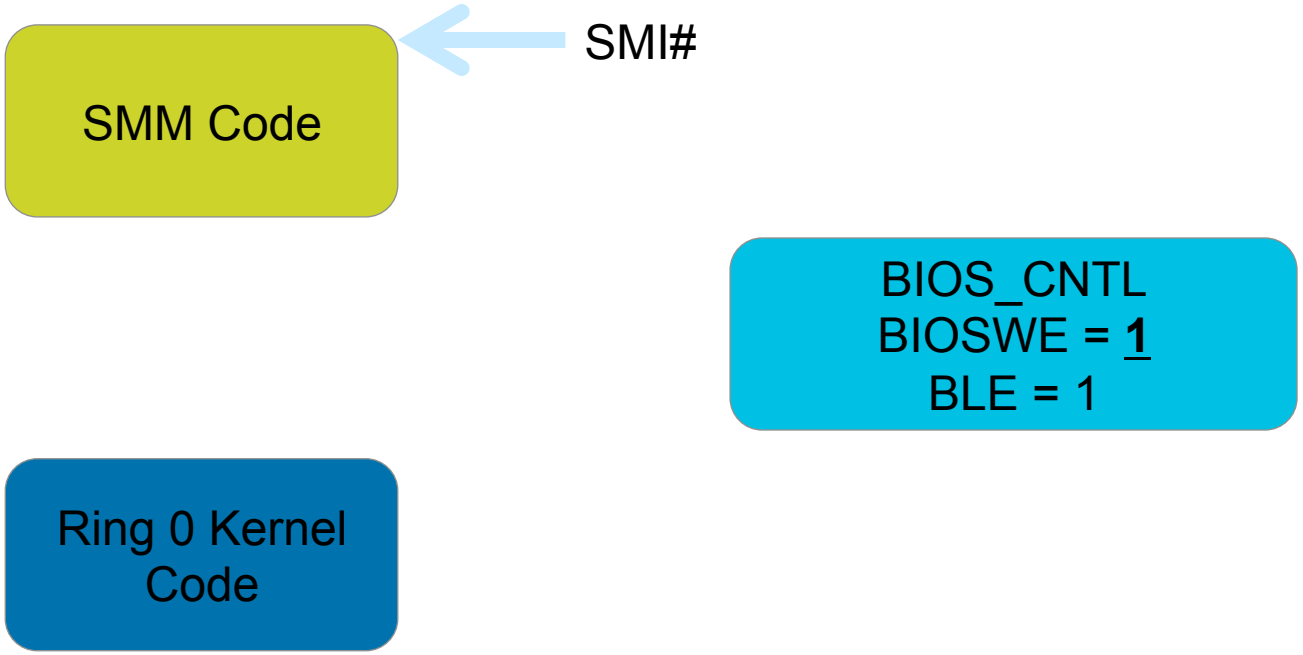
from: http://www.intel.com/content/www/us/en/chipsets/6-chipset-c200-chipset-datasheet.html

SMM BIOSWE protection (1 of 2)



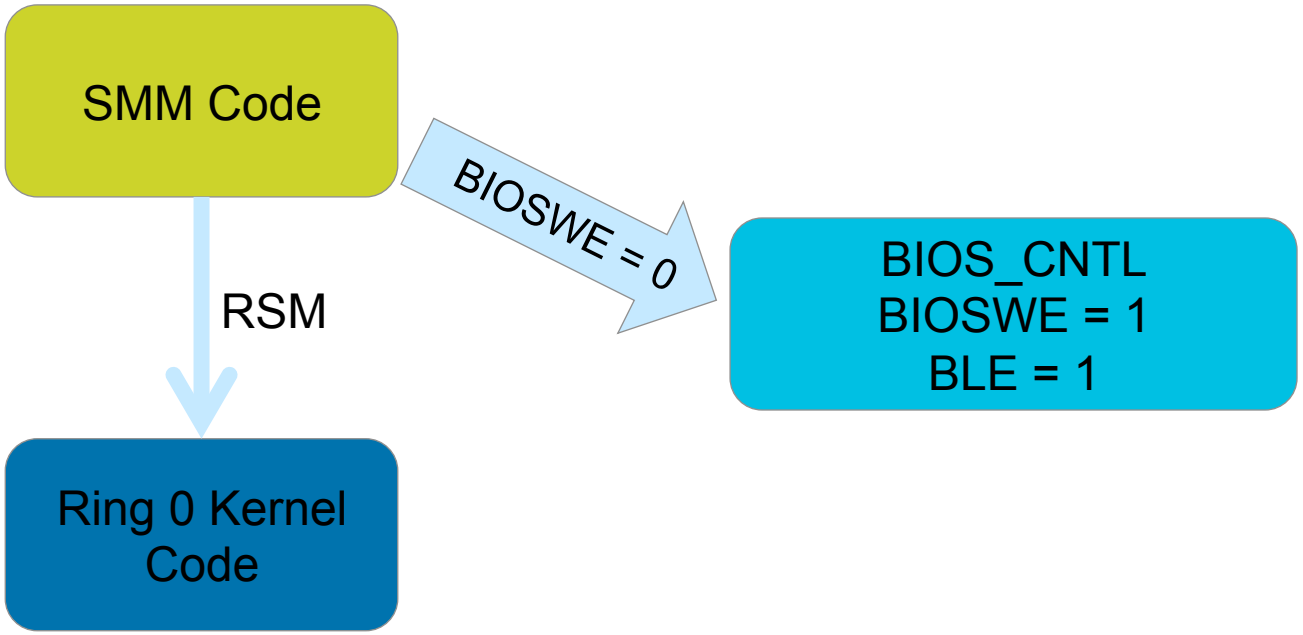
- **Here the attacker tries to set the BIOS Write Enable bit to 1 to allow him to overwrite the BIOS chip.**

SMM BIOSWE protection (2 of 2)



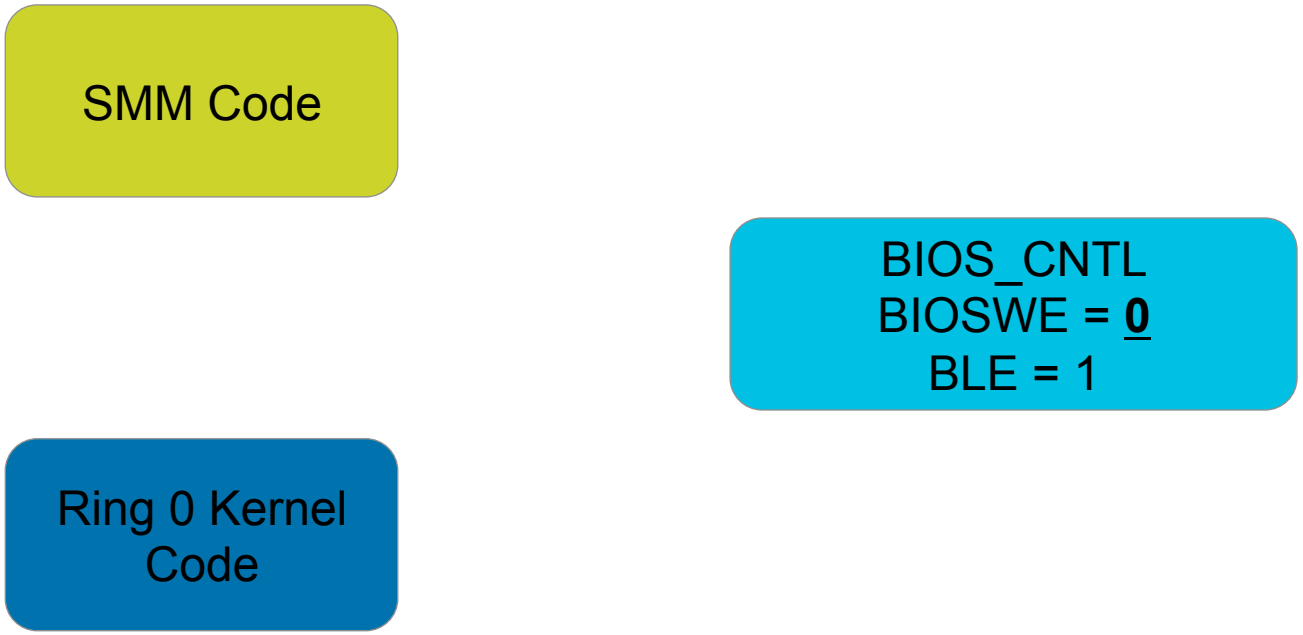
- **The write to the BIOSWE bit while BLE is 1 causes the CPU to generate a System Management Interrupt (SMI#).**

SMM BIOSWE protection (2 of 2)



- **The SMM code immediately writes 0 back to the BIOSWE bit before resuming the kernel code**

SMM BIOSWE protection (2 of 2)



- **The end result is that BIOSWE is always 0 when non-SMM code is running.**

Protected Range SPI Flash Protections

21.1.13 PR0—Protected Range 0 Register (SPI Memory Mapped Configuration Registers)

Memory Address: SPIBAR + 74h Attribute: R/W
Default Value: 00000000h Size: 32 bits

Note: This register can not be written when the FLOCKDN bit is set to 1.

Bit	Description
31	**Write Protection Enable** — R/W. When set, this bit indicates that the Base and Limit fields in this register are valid and that writes and erases directed to addresses between them (inclusive) must be blocked by hardware. The base and limit fields are ignored when this bit is cleared.
30:29	Reserved
28:16	**Protected Range Limit** — R/W. This field corresponds to FLA address bits 24:12 and specifies the upper limit of the protected range. Address bits 11:0 are assumed to be FFFh for the limit comparison. Any address greater than the value programmed in this field is unaffected by this protected range.
15	**Read Protection Enable** — R/W. When set, this bit indicates that the Base and Limit fields in this register are valid and that read directed to addresses between them (inclusive) must be blocked by hardware. The base and limit fields are ignored when this bit is cleared.
14:13	Reserved
12:0	**Protected Range Base** — R/W. This field corresponds to FLA address bits 24:12 and specifies the lower base of the protected range. Address bits 11:0 are assumed to be 000h for the base comparison. Any address less than the value programmed in this field is unaffected by this protected range.

- **Protected Range registers can also provide write protection to the flash chip.**

MITRE

HSFS.FLOCKDN

HSFS—Hardware Sequencing Flash Status Register (SPI Memory Mapped Configuration Registers)

Memory Address: SPIBAR + 04h
Default Value: 0000h
Attribute: RO, R/WC, R/W
Size: 16 bits

Bit	Description
15	**Flash Configuration Lock-Down (FLOCKDN)** — R/W/L. When set to 1, those Flash Program Registers that are locked down by this FLOCKDN bit cannot be written. Once set to 1, this bit can only be cleared by a hardware reset due to a global reset or host partition reset in an Intel® ME enabled system.

- **HSFS.FLOCKDN bit prevents changes to the Protected Range registers once set.**

SMM_BWP

13.1.32 BIOS_CNTL—BIOS Control Register (LPC I/F—D31:F0)

Offset Address:	DCh	Attribute:	R/WLO, R/W, RO
Default Value:	20h	Size:	8 bit
Lockable:	No	Power Well:	Core

Bit	Description
7:6	Reserved
5	**SMM BIOS Write Protect Disable (SMM_BWP)**— R/WLO. This bit set defines when the BIOS region can be written by the host. 0 = BIOS region SMM protection is disabled. The BIOS Region is writable regardless if processors are in SMM or not. (Set this field to 0 for legacy behavior) 1 = BIOS region SMM protection is enabled. The BIOS Region is not writable unless all processors are in SMM.

- **SMM_BWP offers a way to prevent malicious ring 0 writes to the SPI Flash even if SMI's are suppressed.**
- ~~**Of 8005 systems we surveyed, only 6**~~ **actually set SMM_BWP = 1**
  - Thanks to some forced patching, it's up to 1605/~10k! Only 84% fail! ;)

Source: Intel 8-series-chipset-pch-datasheet.pdf

MITRE

Intel Protections Summary

- **The Protected Range Registers, BIOS_CNTL, and SMM_BWP provide overlapping protection of the SPI flash chip that contains the platform firmware.**
  - Protected Range registers can be configured to block all write access to ranges of the SPI Flash.
  - BIOS_CNTL protection puts SMM in a position to decide who can write to the SPI Flash. (weaker)
  - SMM_BWP says "only a CPU in SMM is allowed to write to the flash chip" (stronger)

Charizard

- **BIOS_CNTL protection of the SPI Flash can be defeated on a large number of systems by temporarily suppressing SMM.**
  - Attack does not require arbitrary code execution in SMM.
- **But how can we suppress SMM?**

12.8.1.1 GEN_PMCON_1—General PM Configuration 1 Register (PM—D31:F0)

Offset Address:	A0–A1h	Attribute:	R/W, RO, R/WLO
Default Value:	0000h	Size:	16 bits
Lockable:	No	Usage:	ACPI, Legacy
		Power Well:	Core

4	**SMI_LOCK**—R/WLO. When this bit is set, writes to the GLB_SMI_EN bit (PMBASE + 30h, bit 0) will have no effect. Once the SMI_LOCK bit is set, writes of 0 to SMI_LOCK bit will have no effect (that is, once set, this bit can only be cleared by PLTRST#).

- **SMI_LOCK controls access to the next bit we care about in GBL_SMI_EN (it's a typo in the manual above)**
- **3216 of 8005 (~40%) systems surveyed did not have SMI_LOCK set.**
  - A greater number could probably be made vulnerable by downgrading the BIOS to a vulnerable revision, which is usually allowed.

Source: Intel 8-series-chipset-pch-datasheet.pdf

13.8.3.7 SMI_EN—SMI Control and Enable Register

I/O Address:	PMBASE + 30h	Attribute:	R/W, R/WO, WO, R/WL
Default Value:	00000002h	Size:	32 bit
Lockable:	No	Usage:	ACPI or Legacy
Power Well:	Core		

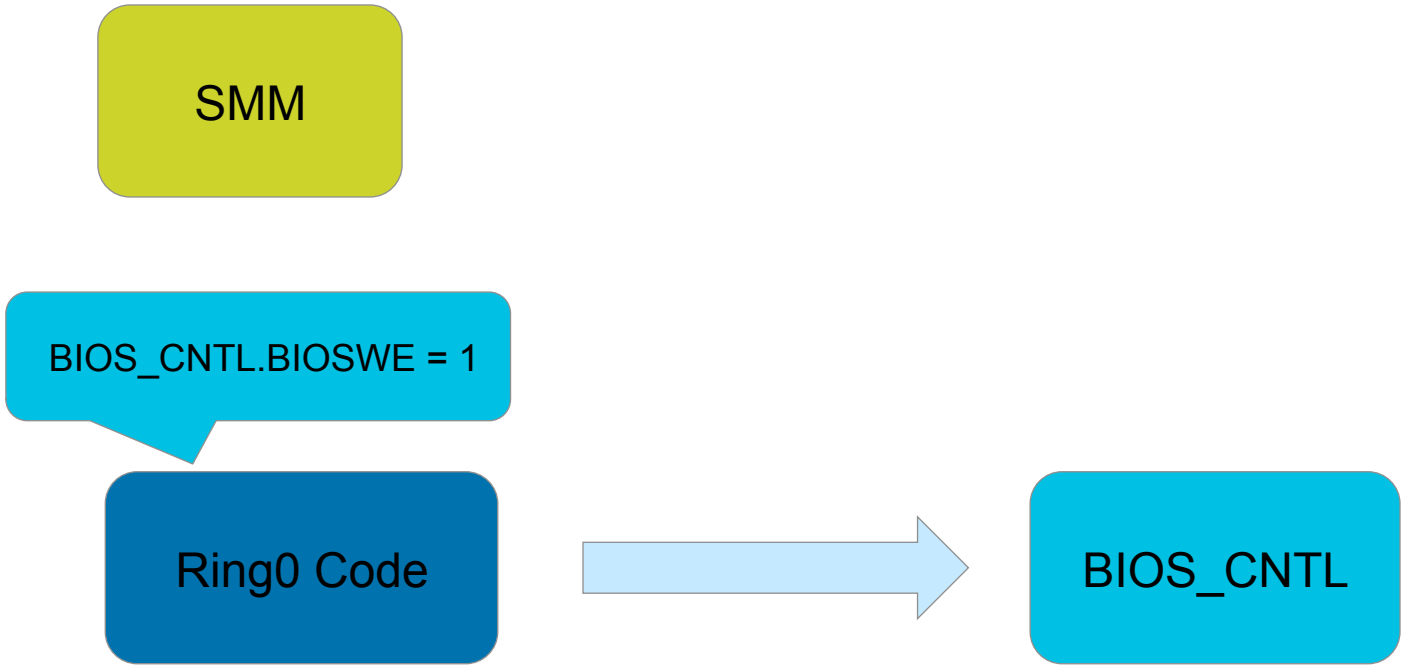
0	**GBL_SMI_EN** — R/WL. 0 = No SMI# will be generated by PCH. This bit is reset by a PCI reset event. 1 = Enables the generation of SMI# in the system upon any enabled SMI event. **NOTE:** When the SMI_LOCK bit is set, this bit cannot be changed.

- **If SMI_LOCK == 0, we can set GBL_SMI_EN to 0 to temporarily disable SMIs and write to flash regions relying on BIOS_CNTL protection, like the EFI variable region.**

Source: Intel 8-series-chipset-pch-datasheet.pdf

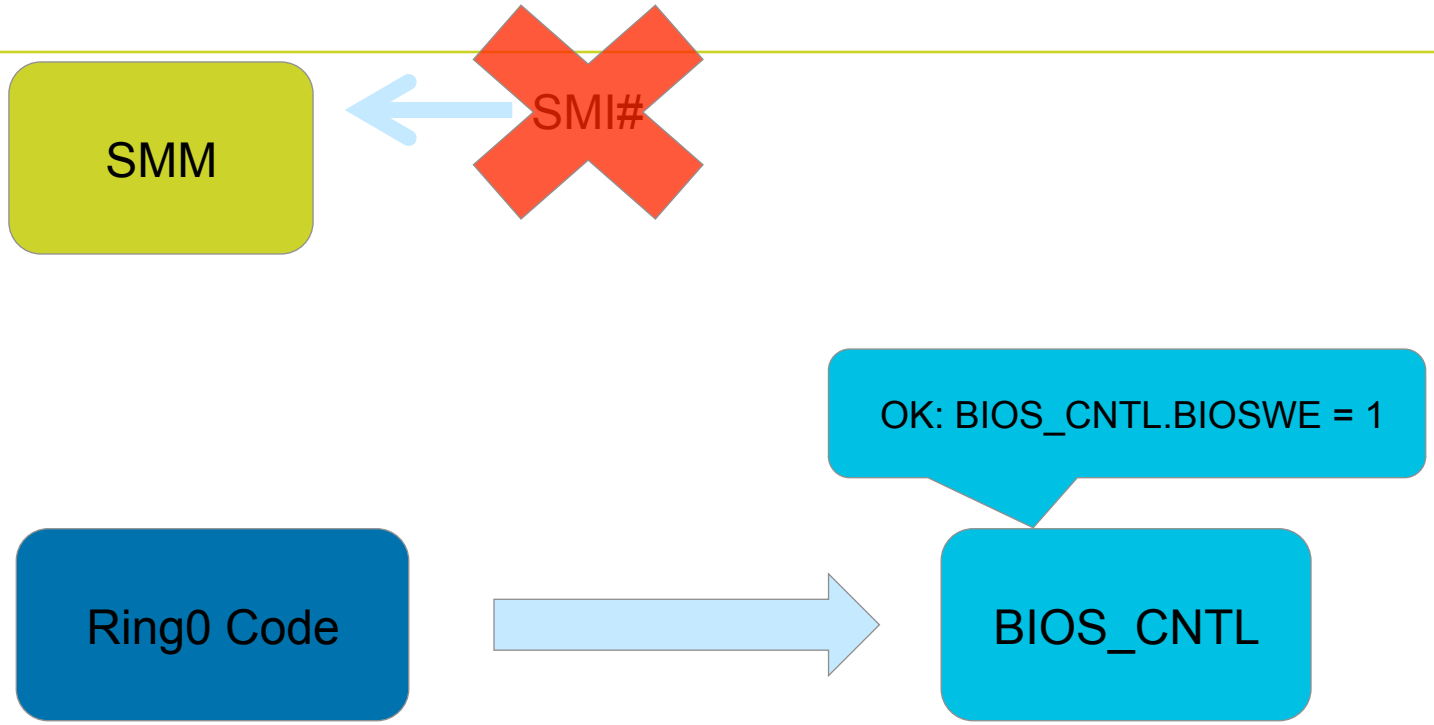
MITRE

Disabled BIOSWE protection (1 of 2)

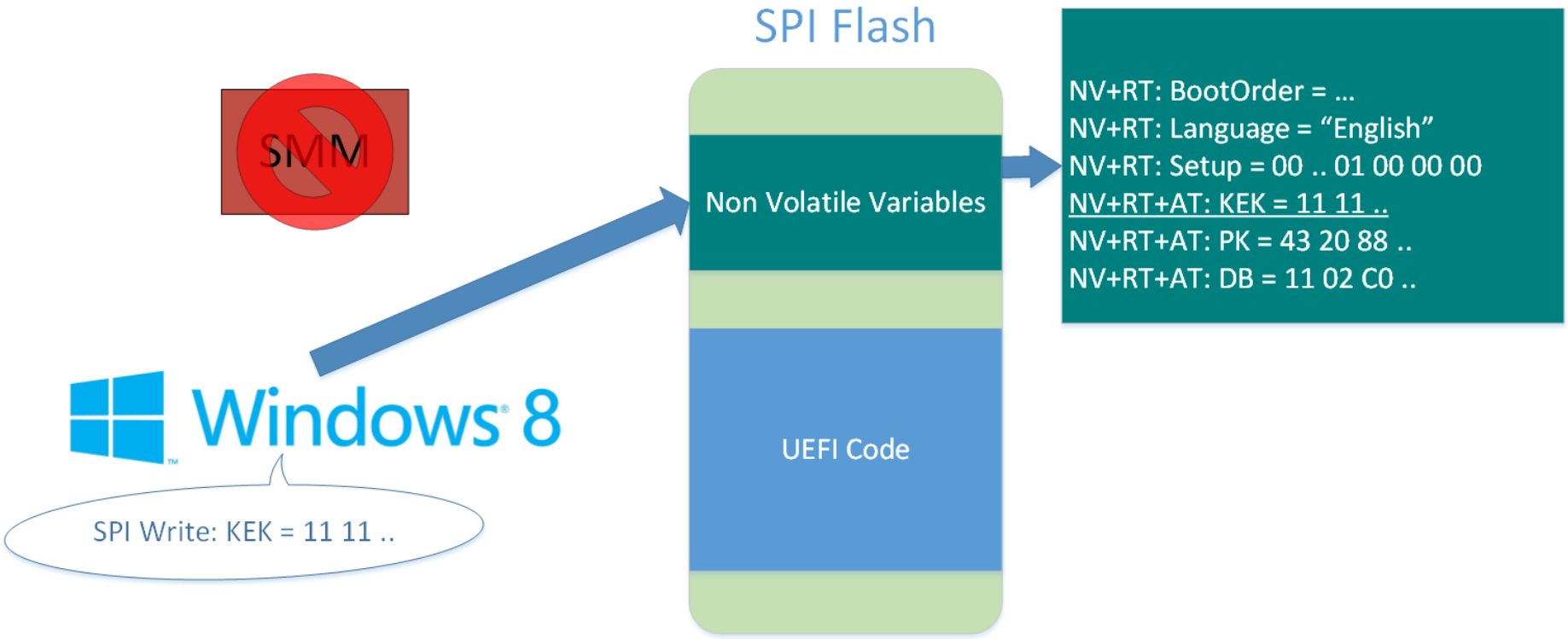


- **Again the attacker tries to set the BIOS Write Enable bit to 1 to allow him to overwrite the BIOS chip.**

Disabled BIOSWE protection (2 of 2)



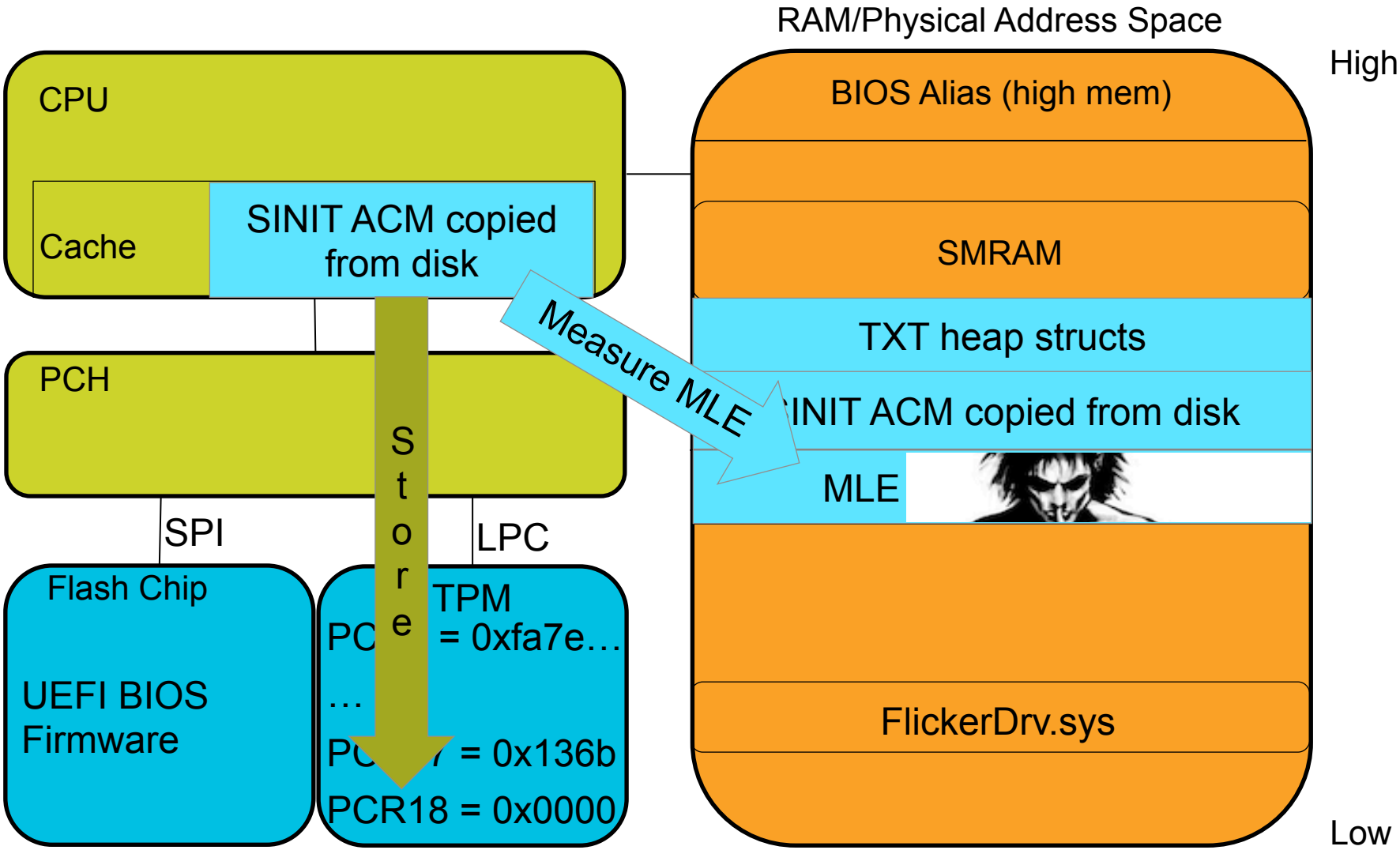
- **This time the SMI that protects BIOSWE fails to fire.**



- **Ring 0 can now modify authenticated EFI Variables, which allows trivial bypassing of Secure Boot.**

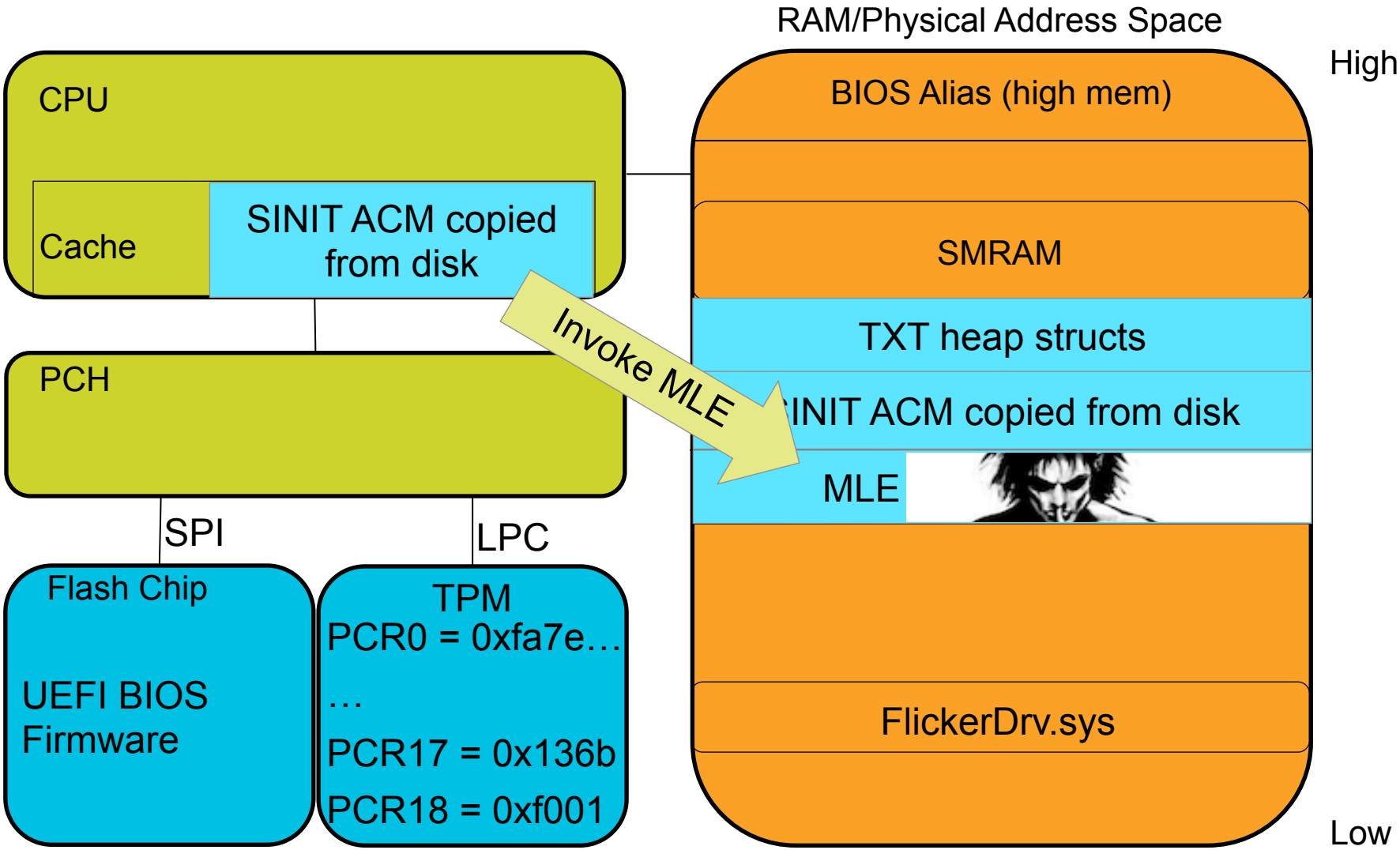

ENTER SANDMAN

Copernicus 2 Architecture



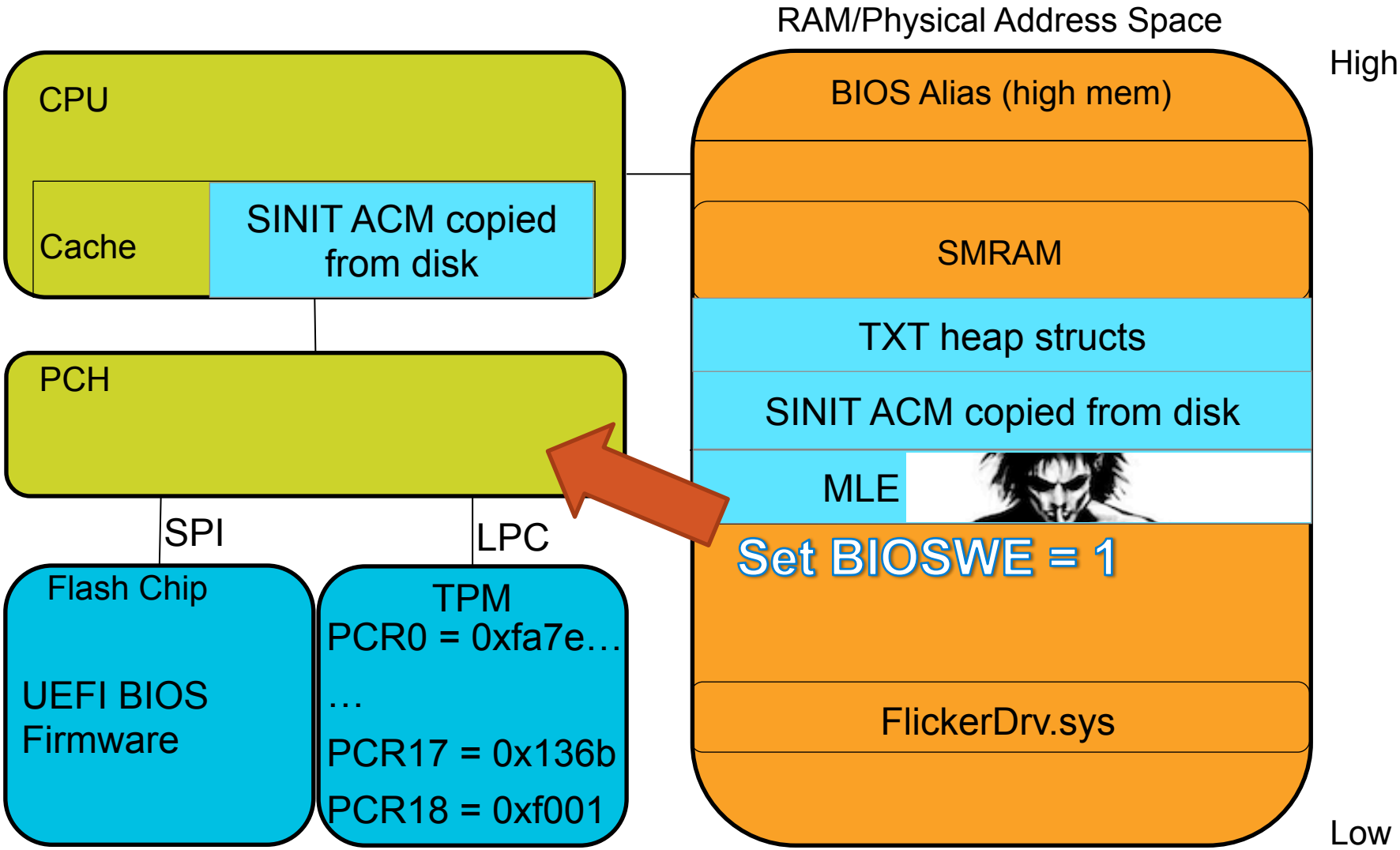
MITRE

Copernicus 2 Architecture



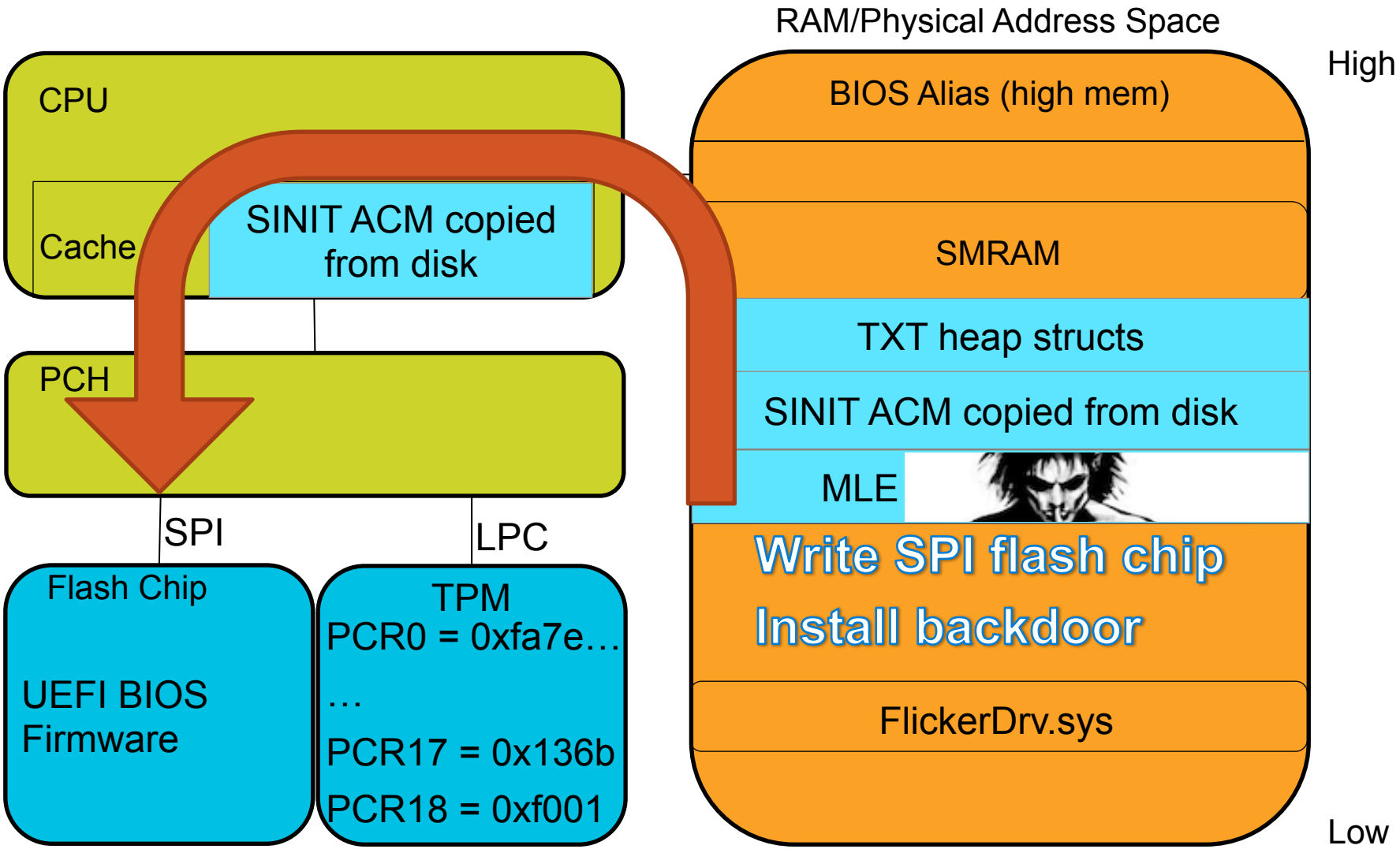
MITRE

Copernicus 2 Architecture



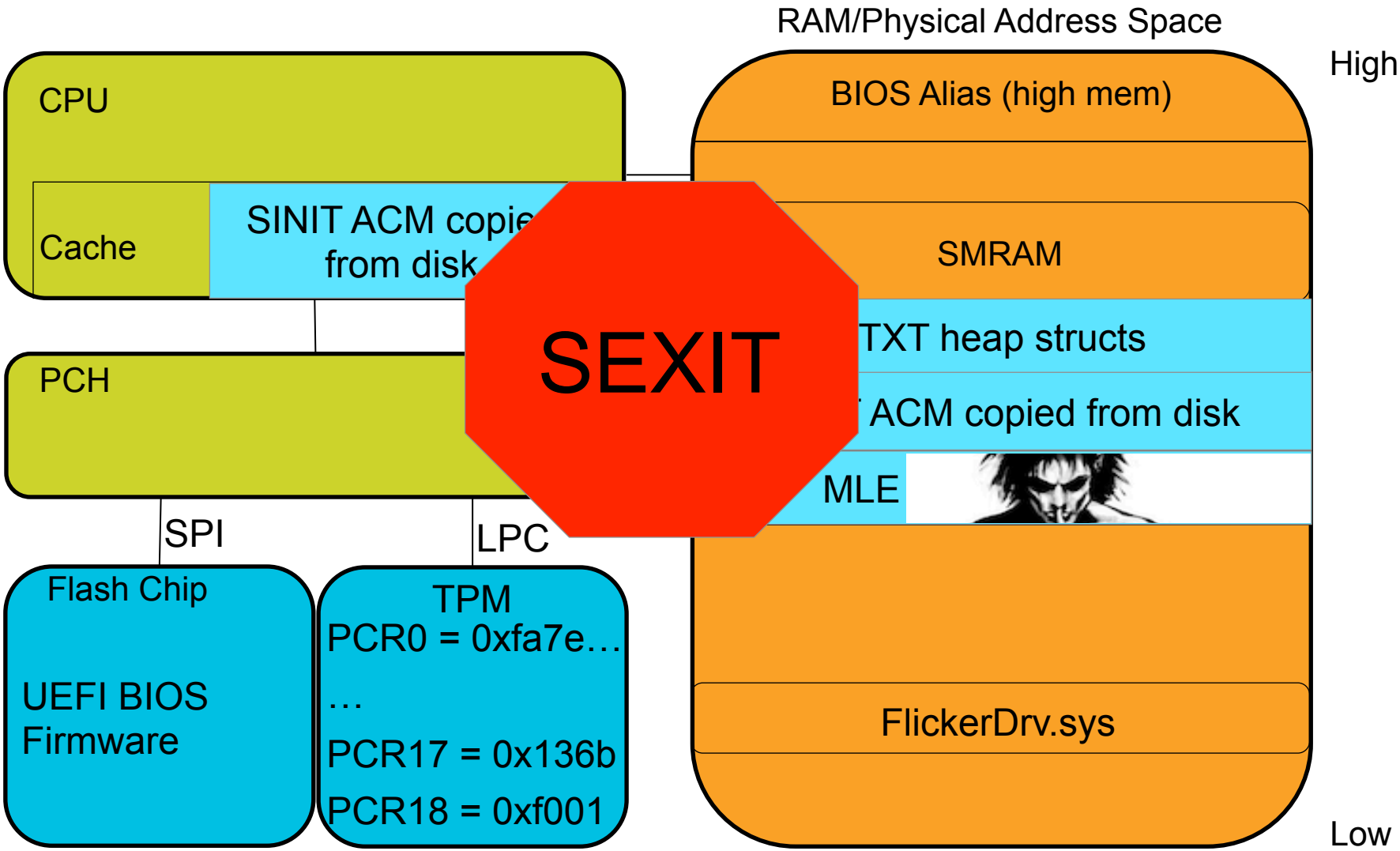
MITRE

Copernicus 2 Architecture



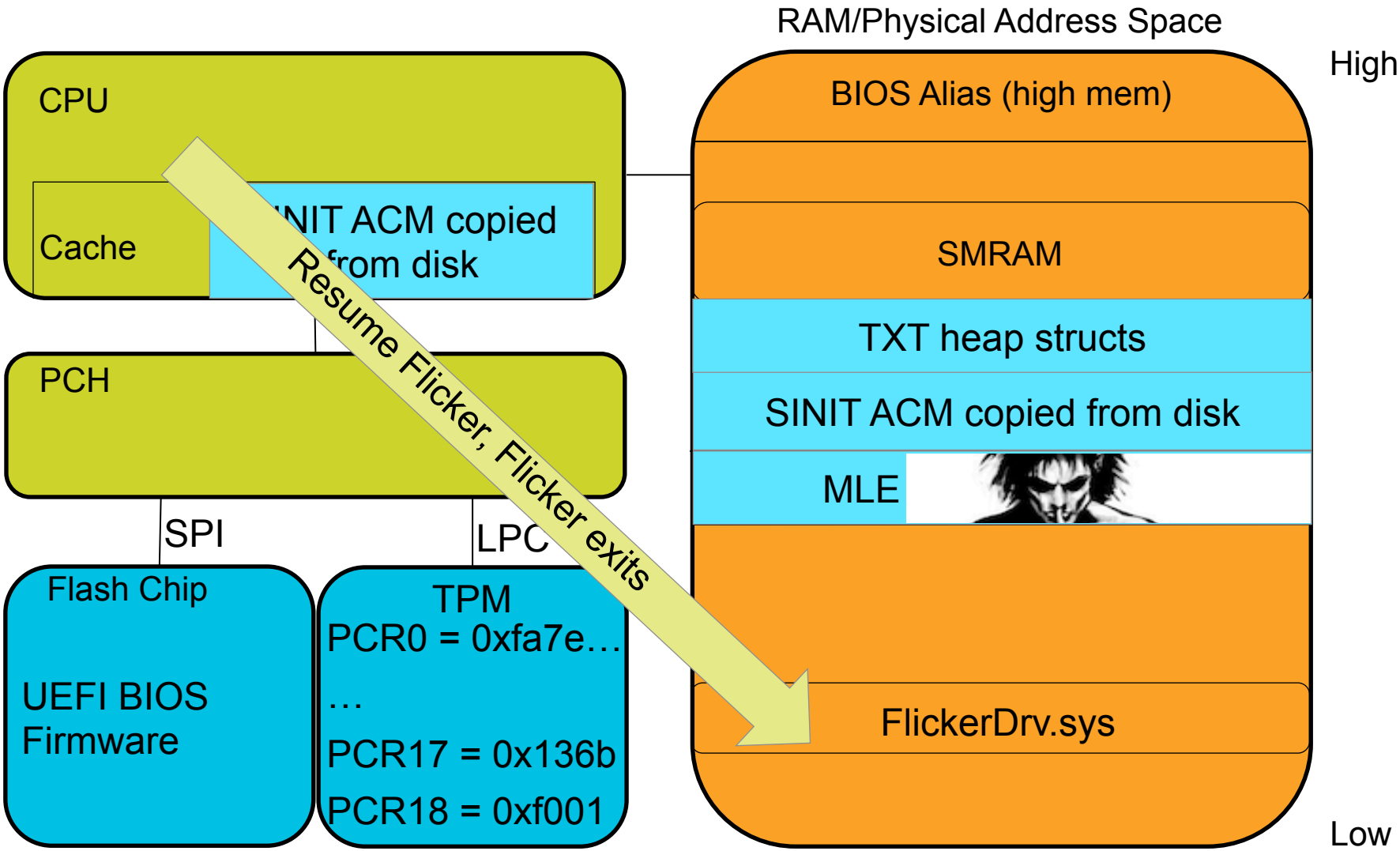
MITRE

Copernicus 2 Architecture



MITRE

Copernicus 2 Architecture



MITRE

Caveat Dormientes

- Xeno pursued TXT for SMI suppression in Copernicus 2 & Sandman because he read this in the Intel TXT Developer's Guide:
- "The ILP must re-enable SMIs which were disabled as part of the SENTER process; most systems will not function properly if SMIs are disabled for any length of time. …"
- OK, good so far…but wait…Xeno didn't read to the end of the paragraph…
- "Newer CPUs may automatically enable SMIs on entry to the MLE;"
- Oh… Well that could be a problem. So how do we know which "Newer CPUs" behave which way, and under which conditions they "may" enable SMIs?

Follow the clues

- One clue in the EXITAC instruction leaf which is what the SINIT module would run as it's exiting and going to hand off to MLE

GETSEC[EXITAC]—Exit Authenticated Code Execution Mode

<Pseudocode snip>

```
IF (SENTERFLAG=0)
    THEN Unmask SMI, INIT, NMI, and A20M pin event;
ELSEIF (IA32_SMM_MONITOR_CTL[0] = 0)
    THEN Unmask SMI pin event;
ACMODEFLAG← 0;
EIP← tempEIP;
END;
```

- This translates to "If we are done with an SENTER, and we're about to transfer to the MLE, and if IA32_SMM_MONITOR_CTL[0] = 1, then SMIs will be suppressed when we transfer to the MLE"
- So we want to be able to set IA32_SMM_MONITOR_CTL[0] = 1
- How do we do that?

9BH	155	IA32_SMM_MONITOR_CTL	SMM Monitor Configuration (R/W)	If CPUID.01H: ECX[bit 5 or bit 6] = 1
		0	Valid (R/W)	
		1	Reserved	
		2	Controls SMI unblocking by VMXOFF (see Section 34.14.4)	If IA32_VMX_MISC[bit 28])
		11:3	Reserved	
		31:12	MSEG Base (R/W)	
		63:32	Reserved	

- **OK, bit 0 is the valid bit.**
- **This architectural MSR is only valid if you call CPUID with EAX set to 1, and if the returned ECX has bit 5 or 6 = 1**
- **So what are those bits?**

5	VMX	**Virtual Machine Extensions.** A value of 1 indicates that the processor supports this technology
6	SMX	**Safer Mode Extensions.** A value of 1 indicates that the processor supports this technology. See Chapter 5, "Safer Mode Extensions Reference".

- **Well that's not helpful, because that implies anything that supports SMX will support IA32_SMM_MONITOR_CTL**

MITRE

Follow the clues

- **Hmm…here's some other interesting facts that turned up in the search through the manuals**

- The IA32_SMM_MONITOR_CTL MSR is supported only on processors that support the dual-monitor treatment.[1] On other processors, accesses to the MSR using RDMSR or WRMSR generate a general-protection fault (#GP(0)).
- A write to the IA32_SMM_MONITOR_CTL MSR using WRMSR generates a general-protection fault (#GP(0)) if executed outside of SMM or if an attempt is made to set any reserved bit. An attempt to write to the

1. Software should consult the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1) to determine whether the dual-monitor treatment is supported.

IA32_SMM_MONITOR_CTL is an MSR that can be written only in SMM.

- **OK, so we can't force the valid bit in the MSR. We'll come back to that.**
- **OK now we need to consult IA32_VMX_BASIC to know if a processor supports dual-monitor treatment to know if it supports IA32_SMM_MONITOR_CTL…**

480H	1152	IA32_VMX_BASIC	**Reporting Register of Basic VMX Capabilities (R/O)** See Appendix A.1, "Basic VMX Information."	If CPUID.01H:ECX.[bit 5] = 1

If bit 49 is read as 1, the logical processor supports the dual-monitor treatment of system-management interrupts and system-management mode. See Section 34.15 for details of this treatment.

MITRE

Double Your Pleasure, Double Your Fun

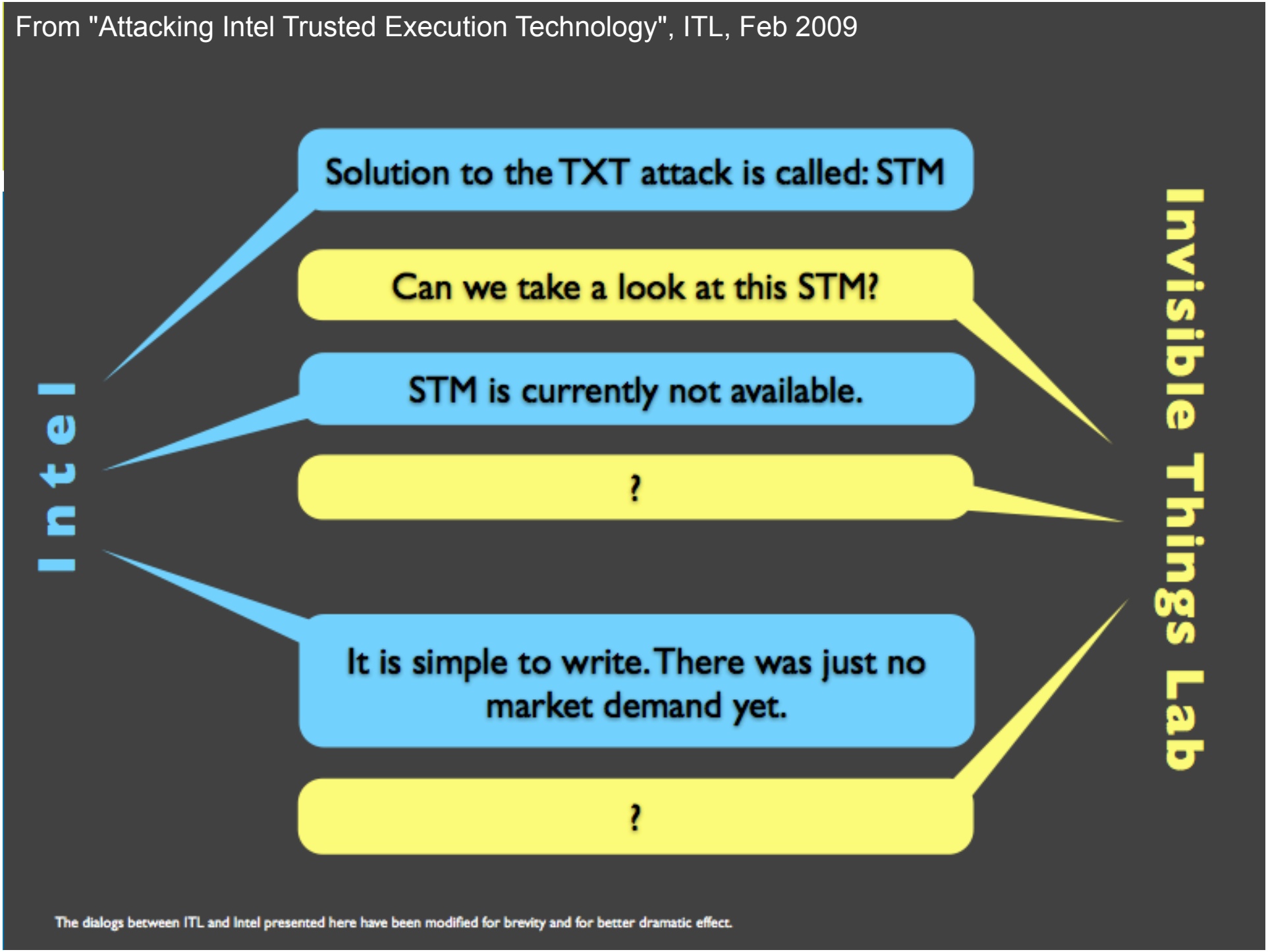
- **What is this "dual-monitor treatment"?**

34.15.1 Dual-Monitor Treatment Overview

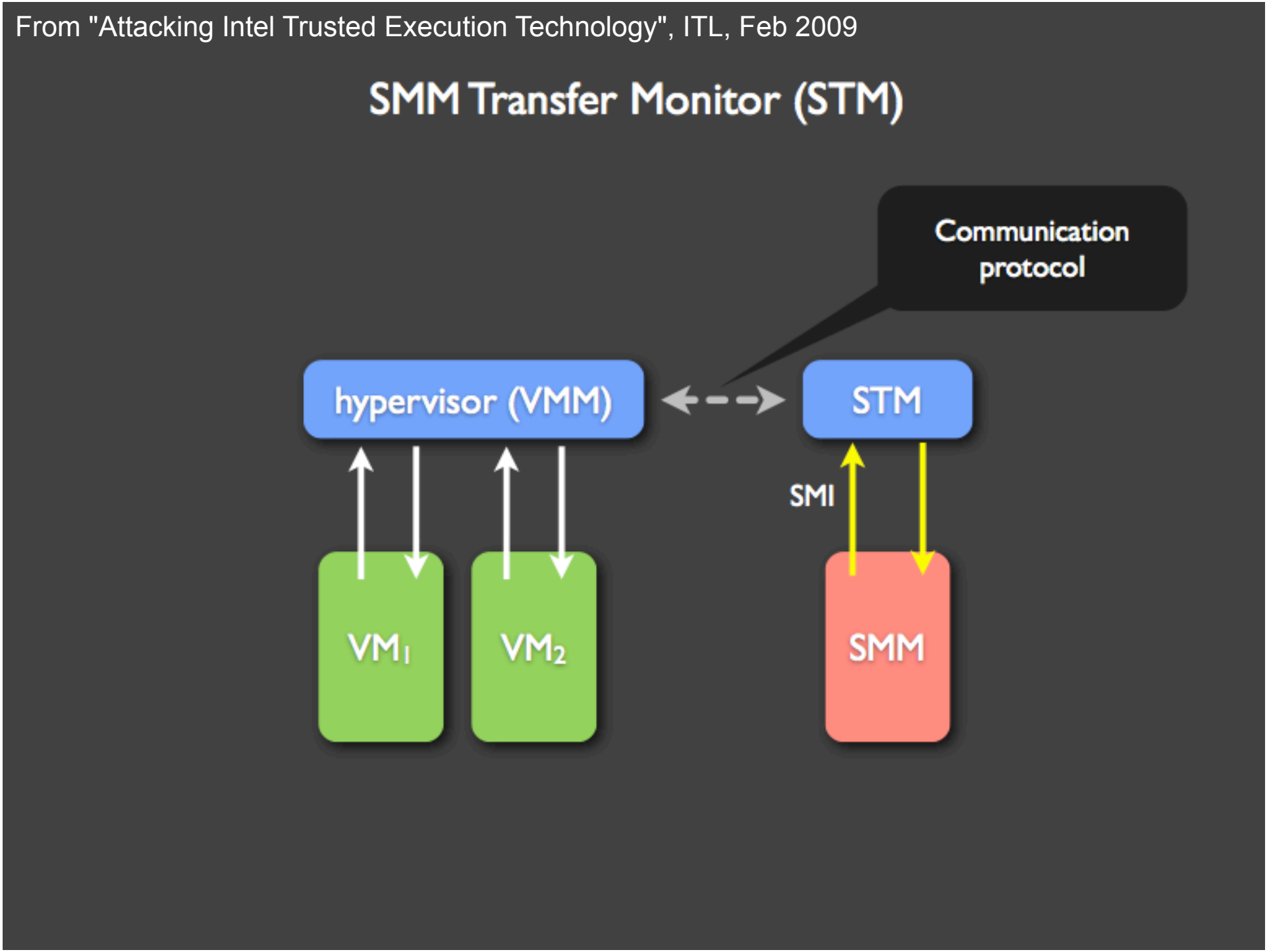
The dual-monitor treatment uses an executive monitor and an SMM-transfer monitor (STM).

- **Oh! It's the use of the STM! You remember STMs right? No? They were the solution to Intel TXT's problem with SMM that ITL found back in 2009**

From "Attacking Intel Trusted Execution Technology", ITL, Feb 2009



From "Attacking Intel Trusted Execution Technology", ITL, Feb 2009



From "Attacking Intel Trusted Execution Technology", ITL, Feb 2009

Intel told us they do have STM specification that answers some of our concerns (e.g. that STM is difficult to write), and the spec is available under NDA.

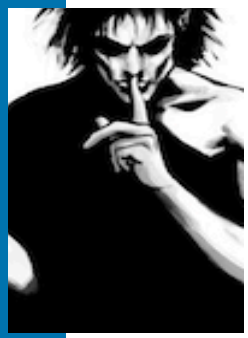
Intel offered us a chance to read the STM spec…
...but required signing an NDA.
…
We refused.

(We'd rather not tie our hands with signing an NDA — we prefer to wait for some STM to be available and see if we can break it :)

- **While our MITRE team also likes to take the black box approach to looking at things, we eventually decided that entering into some NDAs with Intel and other BIOS vendors would accelerate our discovery of problems and help move BIOS security forward faster, which is our ultimate goal.**
- **So we checked out the Intel STM document (still not released 5 years later…on version .99 v8 ;)), and came across this handy flow chart… which we *didn't* receive permission from Intel to even partially reproduce :(**

MITRE

MITRE




This way for win

And in the end, just gave in and asked Intel…

- **Intel says that the "newer CPUs" are some Nehalem and newer**
- **But I also accidentally found out that Xeon CPU's don't support STM, and rely on the Static Root of Trust for Measurement (SRTM)**
  - :O That's not good! We showed in our "BIOS Chronomancy"[18] talk last year how SRTMs without truly immutable core roots of trust are fundamentally flawed and vulnerable
    - Oh, but they support using TXT ACMs as the entry point for the reset vector, so functionally the SRTM…so…hmm…As long as SMM can never ever get compromised directly while the system is running *cough*futurecoreytalk*cough* then I guess you're OK…

Result

- **Sandman (and Copernicus 2) work on older machines that unconditionally suppress SMIs, and on newer machines that include an STM**
  - "The once and future king"…
- **The BIOS vendor needs to include an STM in their shipping BIOS, because the STM lives in SMM**
  - SMM should be locked so that people can't put stuff there.
    - If anyone can just add an STM on demand that would be an attack.
- **We're not aware of any vendors shipping or planning to ship STMs (but we haven't conducted a survey)**
- **Where Copernicus 2 has fallen, Smite'em holds court!**
- **Intel could fix this by just re-enabling SMI suppression for TXT. We asked them to, but we don’t think they will.**

MITRE

Must I be vulnerable to Sandman to run Copernicus 2?

- No. TXT has an access control mechanism where you can set a Launch Control Policy (LCP)
- The LCP can specify that you only want to allow specific MLEs to run on your systems
- LCPs are stored in the TPM's non-volatile RAM
- So basically when you're provisioning the TPM so that you can trust your measurements from some MLE like Copernicus 2, you can also lock it in so that only Copernicus 2 can run in the future

Conclusions

- **We have now completed the "BLE-SMI Suppression Trilogy"**
  - Via CPU cache poisoning at EkoParty [13]
  - Via GBI_SMI_EN at Syscan[29]
  - Via TXT at SummerCon [30]
  - ("BlessMe Suppression" doesn't sound as cool as "Xen 0wning" but if ITL has a trilogy, we need one too ;))
- **Are there other ways? Probably, but it doesn't matter**
- **We've been telling people to stop relying on BIOSWE/BLE protection and starting using Protected Range Registers and SMM_BWP for a while now.**
- **We recently found what we think is a *fundamental* architectural flaw in BIOSWE/BLE which will render it completely useless**
- **Stay tuned ;)**

A new caveat has entered the ring!

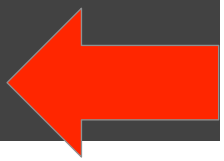
From "Ring -3 Rootkits", ITL, July 2009

http://invisiblethingslab.com/resources/bh09usa/Ring%20-3%20Rootkits.pdf

We *do* like many of the new Intel technologies (VT-x, VT-d, TXT), ...

But AMT is different in that it can potentially be greatly abused by the attacker

(VT-d or TXT can potentially be bypassed, but they cannot *help* the attacker!)



- We also like security technologies like TXT
- Wojtczuk & Rutkowska caveated this when they showed an SINIT buffer overflow (subsequently patched) which yielded SMM privs
- We've now showed another (more architectural) way that TXT can be a double-edged sword
  - But BIOS-vendors can mitigate this attack by not relying on BIOS_CNTL.BIOSWE/BLE as a primary access control

Closing thoughts

- Sleep with one eye open!
- Gripping your pillow tight!
- EXXXXIIIITTTT LIGHT!
- EEEENNNNNTTTTEERRR NIIIIGHT!

Thanks, Contacts, Questions?

- **Thanks to FOO from Intel who saw our Cop 2 slides and pointed out the differing ways SMIs behave, and for John Loucaides and Bruce Monroe from Intel PSIRT for working with us on our various disclosures and getting us in touch with FOO.**

- **Thanks to Jon McCune and the team from CMU for making Flicker for people to build on.**
  - Our changes we contributed back to make it work with default Win 7 32 with PAE enabled are here:
  - http://sourceforge.net/p/flickertcb/code/ci/experimental-pae-support

- **xkovah, callenberg, jbutterworth, scornwell @ mitre.org**

- **@xenokovah, @coreykal, @jwbutterworth3, @ssc0rnwell**

MITRE

Backup

MITRE

"But I can detect that Sandman ran because of unexpected PCR values!"

- The attacker could reboot the system ASAP after the BIOS has been written to in order to try and gain their SMM or whatever other privileges ASAP.
- The attacker could run an expected/clean MLE immediately afterwards to have the PCR values overwritten with expected/ non-suspicious values
- ? The attacker could launch an MLE in a way that they knew would fail (but wouldn't reboot the system) so as to allow it to proceed past PCR reset but not until PCR extend.
- ? The attacker could manually reset the PCR from within the MLE at locality 3?

References

- [1] Attacking Intel BIOS – Alexander Tereshkin & Rafal Wojtczuk – Jul. 2009 http://invisiblethingslab.com/resources/bh09usa/Attacking%20Intel%20BIOS.pdf
- [2] TPM PC Client Specification - Feb. 2013 http://www.trustedcomputinggroup.org/developers/pc_client/specifications/
- [3] Evil Maid Just Got Angrier: Why Full-Disk Encryption With TPM is Insecure on Many Systems – Yuriy Bulygin – Mar. 2013 http://cansecwest.com/slides/2013/Evil%20Maid%20Just%20Got%20Angrier.pdf
- [4] A Tale of One Software Bypass of Windows 8 Secure Boot – Yuriy Bulygin – Jul. 2013 http://blackhat.com/us-13/briefings.html#Bulygin
- [5] Attacking Intel Trusted Execution Technology - Rafal Wojtczuk and Joanna Rutkowska – Feb. 2009 http://invisiblethingslab.com/resources/bh09dc/Attacking%20Intel%20TXT%20-%20paper.pdf
- [6] Another Way to Circumvent Intel® Trusted Execution Technology - Rafal Wojtczuk, Joanna Rutkowska, and Alexander Tereshkin – Dec. 2009 http://invisiblethingslab.com/resources/misc09/Another%20TXT%20Attack.pdf
- [7] Exploring new lands on Intel CPUs (SINIT code execution hijacking) - Rafal Wojtczuk and Joanna Rutkowska – Dec. 2011 http://www.invisiblethingslab.com/resources/2011/Attacking_Intel_TXT_via_SINIT_hijacking.pdf
- [7] Meet 'Rakshasa,' The Malware Infection Designed To Be Undetectable And Incurable - http://www.forbes.com/sites/andygreenberg/2012/07/26/meet-rakshasa-the-malware-infection-designed-to-be-undetectable-and-incurable/

References 2

- [8] Implementing and Detecting an ACPI BIOS Rootkit – Heasman, Feb. 2006 http://www.blackhat.com/presentations/bh-europe-06/bh-eu-06-Heasman.pdf
- [9] Implementing and Detecting a PCI Rookit – Heasman, Feb. 2007 http://www.blackhat.com/presentations/bh-dc-07/Heasman/Paper/bh-dc-07-Heasman-WP.pdf
- [10] Using CPU System Management Mode to Circumvent Operating System Security Functions - Duflot et al., Mar. 2006 http://www.ssi.gouv.fr/archive/fr/sciences/fichiers/lti/cansecwest2006-duflot-paper.pdf
- [11] Getting into the SMRAM:SMM Reloaded – Duflot et. Al, Mar. 2009 http://cansecwest.com/csw09/csw09-duflot.pdf
- [12] Attacking SMM Memory via Intel® CPU Cache Poisoning – Wojtczuk & Rutkowska, Mar. 2009 http://invisiblethingslab.com/resources/misc09/smm_cache_fun.pdf
- [13] Defeating Signed BIOS Enforcement – Kallenberg et al., Sept. 2013 – URL not yet available, email us for slides
- [14] Mebromi: The first BIOS rootkit in the wild – Giuliani, Sept. 2011 http://www.webroot.com/blog/2011/09/13/mebromi-the-first-bios-rootkit-in-the-wild/

References 3

- [15] Persistent BIOS Infection – Sacco & Ortega, Mar. 2009 http://cansecwest.com/csw09/csw09-sacco-ortega.pdf
- [16] Deactivate the Rootkit – Ortega & Sacco, Jul. 2009 http://www.blackhat.com/presentations/bh-usa-09/ORTEGA/BHUSA09-Ortega-DeactivateRootkit-PAPER.pdf
- [17] Sticky Fingers & KBC Custom Shop – Gazet, Jun. 2011 http://esec-lab.sogeti.com/dotclear/public/publications/11-recon-stickyfingers_slides.pdf
- [18] BIOS Chronomancy: Fixing the Core Root of Trust for Measurement – Butterworth et al., May 2013 http://www.nosuchcon.org/talks/D2_01_Butterworth_BIOS_Chronomancy.pdf
- [19] New Results for Timing-based Attestation – Kovah et al., May 2012 http://www.ieee-security.org/TC/SP2012/papers/4681a239.pdf

References 4

- [20] Low Down and Dirty: Anti-forensic Rootkits - Darren Bilby, Oct. 2006
  http://www.blackhat.com/presentations/bh-jp-06/BH-JP-06-Bilby-up.pdf
- [21] Implementation and Implications of a Stealth Hard-Drive Backdoor – Zaddach et al., Dec. 2013
  https://www.ibr.cs.tu-bs.de/users/kurmus/papers/acsac13.pdf
- [22] Hard Disk Hacking – Sprite, Jul. 2013
  http://spritesmods.com/?art=hddhack
- [23] Embedded Devices Security and Firmware Reverse Engineering - Zaddach & Costin, Jul. 2013
  https://media.blackhat.com/us-13/US-13-Zaddach-Workshop-on-Embedded-Devices-Security-and-Firmware-Reverse-Engineering-WP.pdf
- [24] Can You Still Trust Your Network Card – Duflot et al., Mar. 2010
  http://www.ssi.gouv.fr/IMG/pdf/csw-trustnetworkcard.pdf
- [25] Project Maux Mk.II, Arrigo Triulzi, Mar. 2008
  http://www.alchemistowl.org/arrigo/Papers/Arrigo-Triulzi-PACSEC08-Project-Maux-II.pdf

References 5

- [26] Copernicus: Question your assumptions about BIOS Security – Butterworth, July 2013 http://www.mitre.org/capabilities/cybersecurity/overview/cybersecurity-blog/copernicus-question-your-assumptions-about
- [27] Copernicus 2: SENTER the Dragon – Kovah et al., Mar 2014 https://cansecwest.com/slides/2014/Copernicus2-SENTER_the-Dragon-CSW.pptx
- [28] Playing Hide and Seek with BIOS Implants – Kovah, Mar 2014 http://www.mitre.org/capabilities/cybersecurity/overview/cybersecurity-blog/playing-hide-and-seek-with-bios-implants
- [29] Setup for Failure: Defeating UEFI – Kallenberg et al., Apr 2014 http://syscan.org/index.php/download/get/6e597f6067493dd581eed737146f3afb/SyScan2014_CoreyKallenberg_SetupforFailureDefeatingSecureBoot.zip
- [30] SENTER Sandman: Using Intel TXT to Attack BIOSes – Kovah et al., June 2014