动态可信度量

wqdai@hust.edu.cn

代炜琦

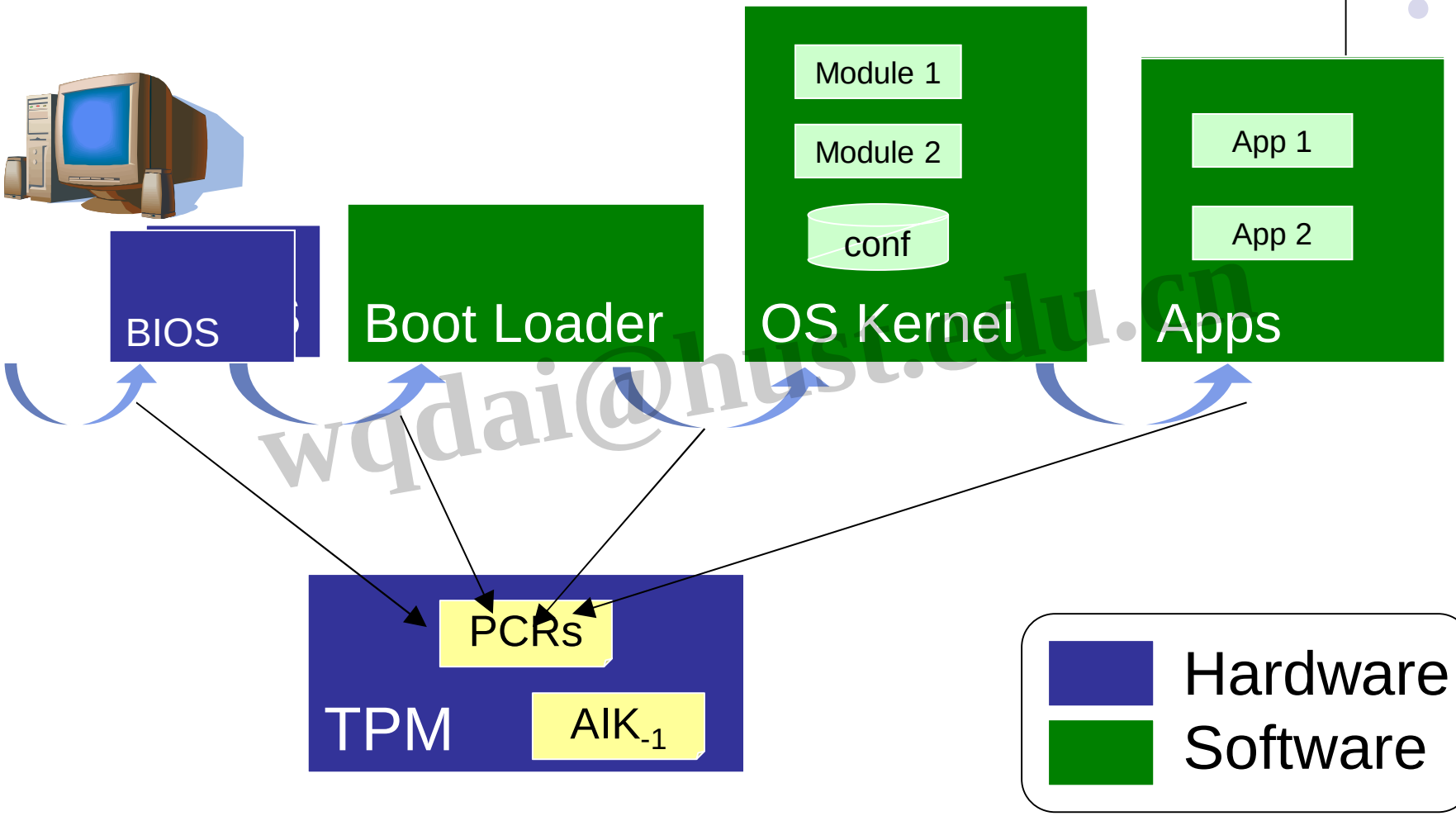
提纲

- 静态可信度量根
- 动态可信度量根
- Flicker: Minimal TCB Code Execution

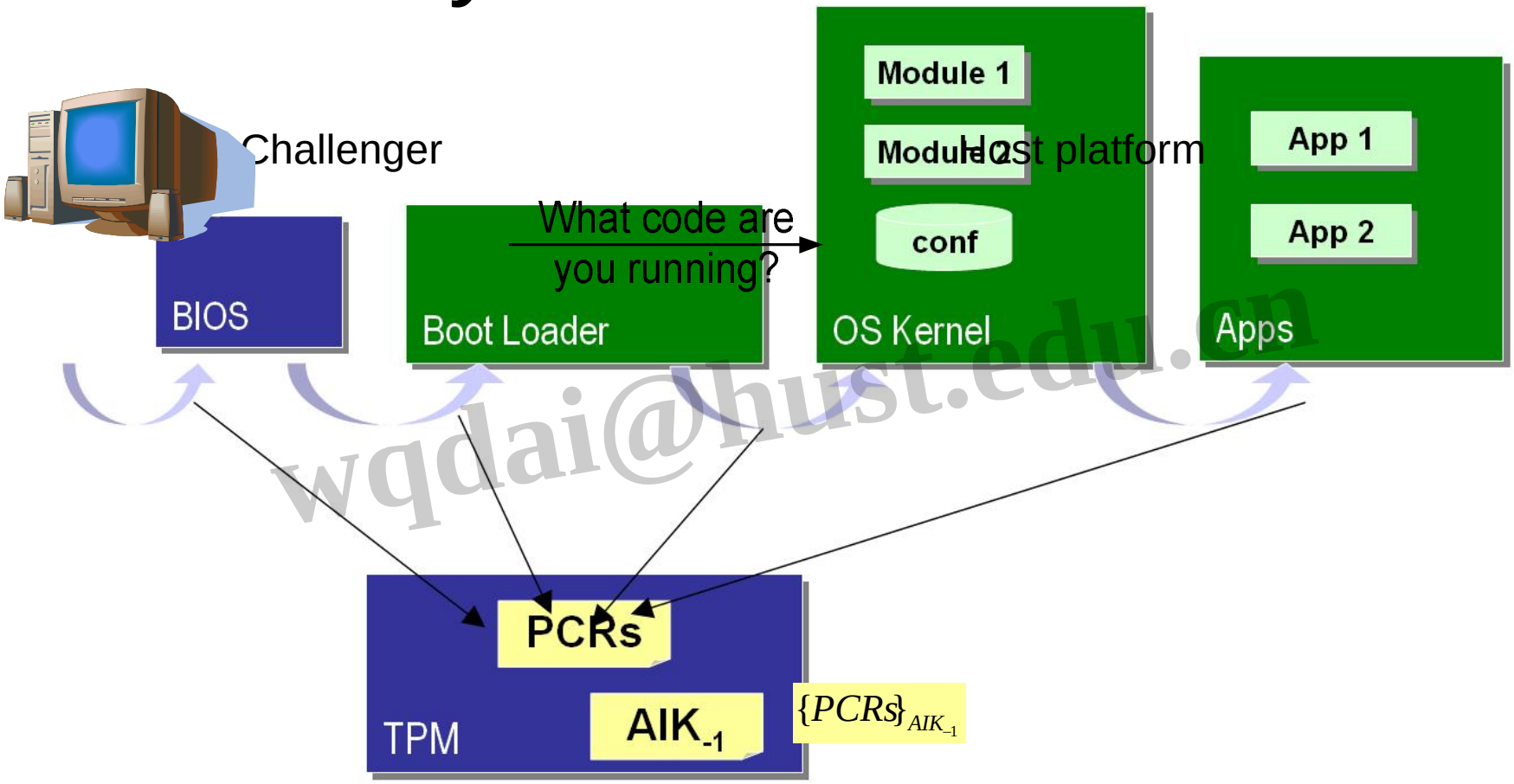
Basic TPM Functions

- PCRs store integrity measurement chain
  - $PCR_{new} = SHA\text{-}1(PCR_{old}||measurement)$
- Remote attestation (PCRs + AIK)
  - Attestation Identity Keys (AIKs) for signing PCRs
  - Attest to value of integrity measurements to remote party
- Sealed storage (PCRs + SRK)
  - Protected storage + unlock state under a particular integrity measurement

TCG-Style Attestation



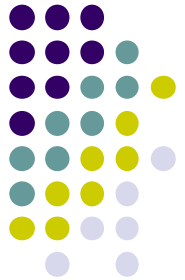
TCG-Style Attestation



TCG-style Attestation的缺点

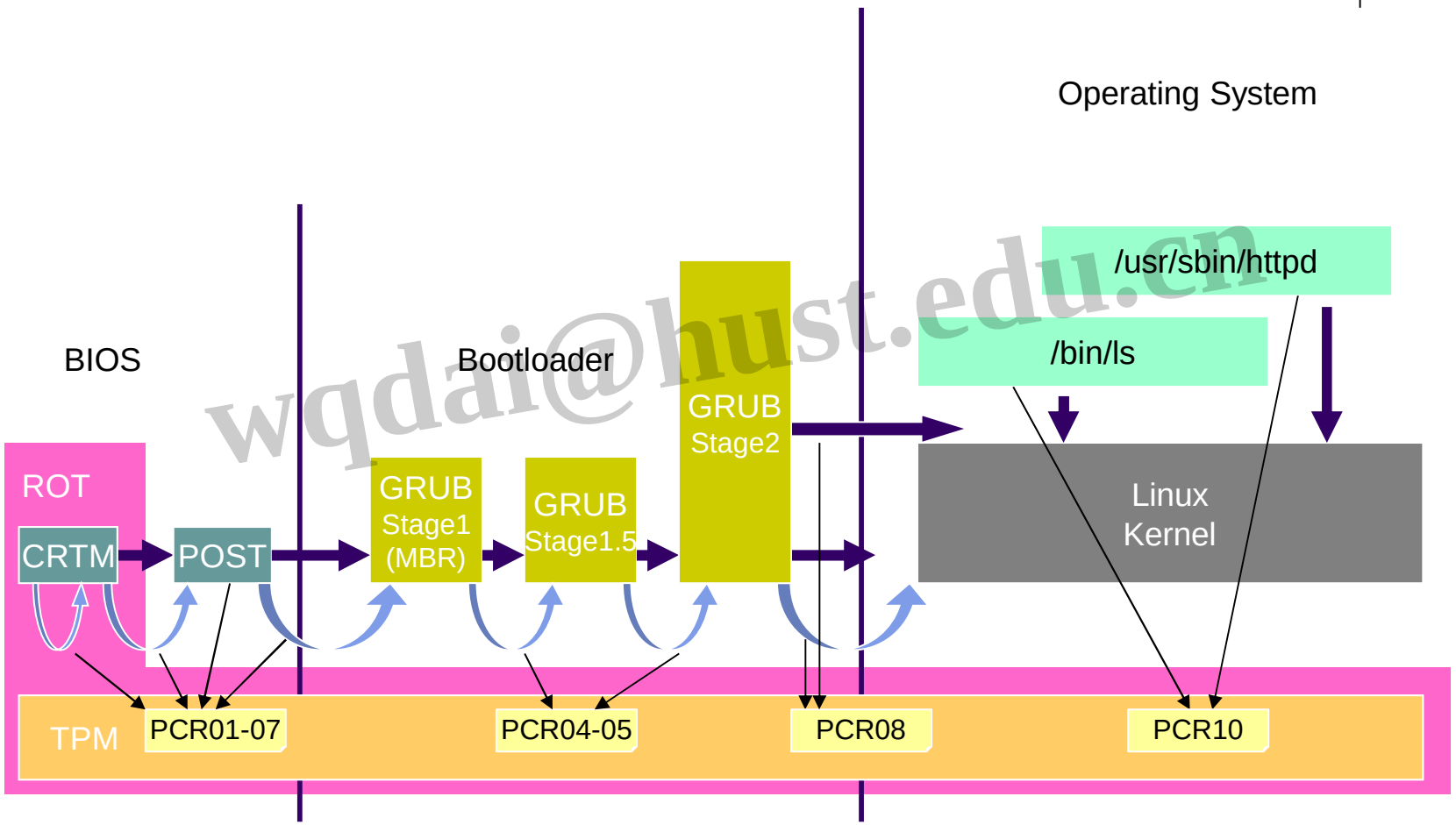
- **Static** root of trust for measurement (**reboot**)
- Measures entire system
  - Requires hundreds of integrity measurements just to boot
  - Every host is different
    - firmware versions, drivers, patches, apps, spyware, …
  - TCB includes entire system!
- Integrity measurements are done at load-time not at run-time
  - Time-of-check-to-time-of-use (TOCTOU) problem
  - Cannot detect any dynamic attacks!
  - No guarantee of execution

Linux Implementation Overview

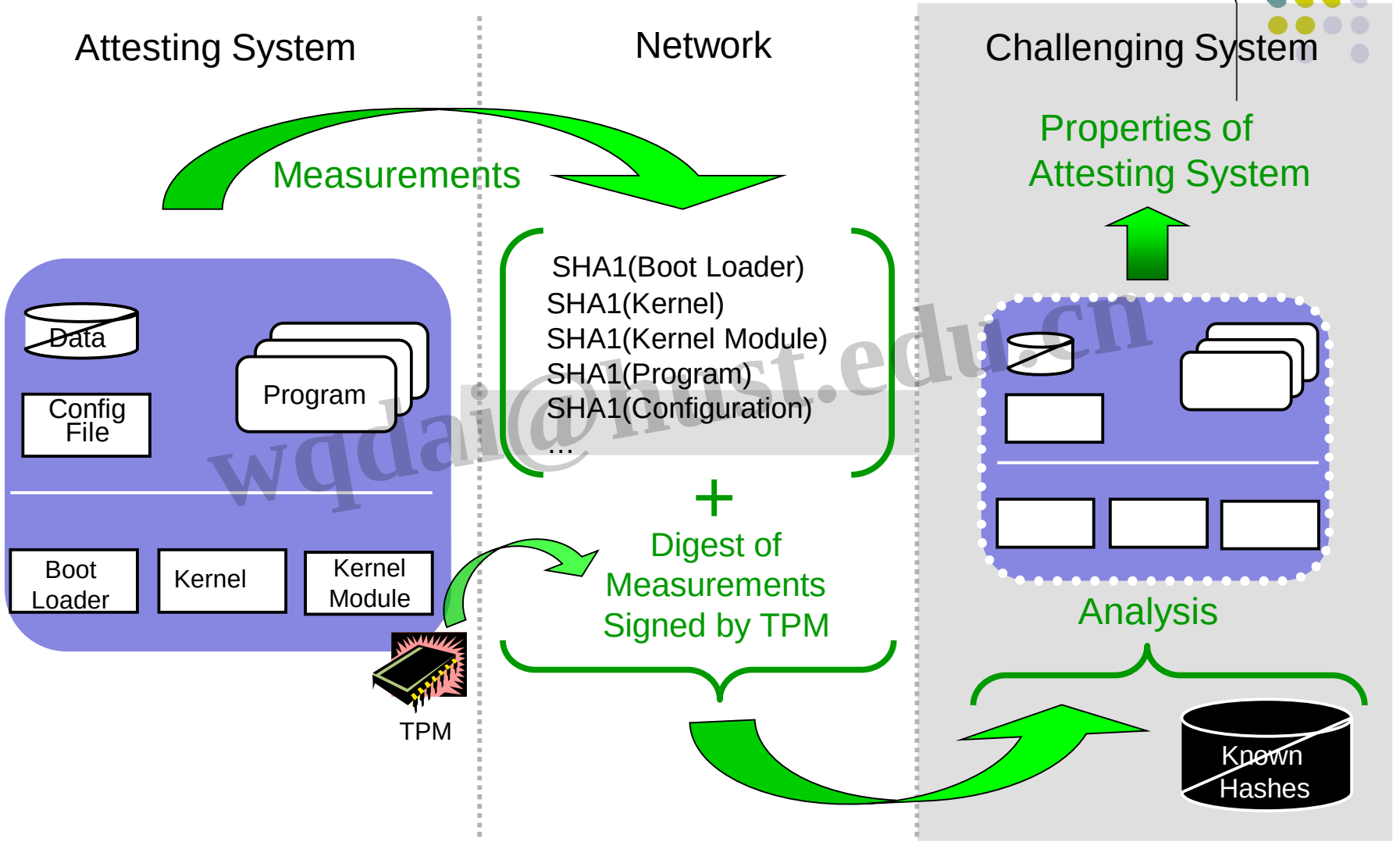


- Use trusted boot to measure BIOS, bootstrap loader, and kernel
- The kernel keeps a list of measurements for each loaded
  - Executable Scripts (shell, Perl, etc)
  - Shared library *Java class files*
  - Kernel module …
- Before a measurement is added to the list, PCR_{10} is extended with that measurement. Integrity of this in-kernel list is guaranteed by PCR_{10}
- Verification: Given initial value of PCR_{10}, extend it with each measurement and match result to current PCR_{10}

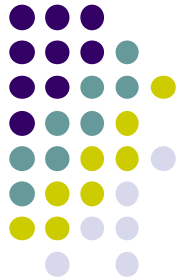
Linux Bootstrap Stages



Integrity Measurement Architecture (IMA)

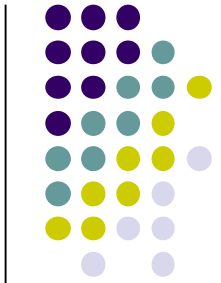


Linux Modifications



- Added support to Linux kernel
  - To measure dynamic linker
  - To measure each executable
- Added support to dynamic linker
  - To measure each shared library
- Added support to kernel module loader
  - To measure kernel modules
- Kernel keeps a measurement cache
  - Files are only measured once!
  - Unless modified (opened for writing)

Linux Application Measurements



cat /proc/tcg/measurements

```
#000: 276249898F406BE176E3D86EDD5A3D20D03EEB11 [remeasure] linuxrc
#001: 9F860256709F1CD35037563DCDF798054F878705 [remeasure] nash
#002: 4CC52A8F7584A750303CB2A41DEA637917DB0310 [clean] insmod
#003: 84ABD2960414CA4A448E0D2C9364B4E1725BDA4F [clean] init
#004: 194D956F288B36FB46E46A124E59D466DE7C73B6 [clean] ld-2.3.2.so
#005: 7DF33561E2A467A87CDD4BB8F68880517D3CAECB [clean] libc-2.3.2.so
#006: 93A0BBC35FD4CA0300AA008F02441B6EAA425643 [clean] rc.sysinit
#007: 66F445E31575CA1ABEA49F0AF0497E3C074AD9CE [clean] bash
#008: F4F6CB0ACC2F1BEE13D60330011DF926D24E5688 [clean] libtermcap.so.2.0.8
#009: 346443AAD8E7089B64B2B67F1A527F7E2CA2D1E5 [clean] libdl-2.3.2.so
#010: 02385033F849A2A4BFB85FD52BCEA27B45497C6C [clean] libnss_files-2.3.2.so
#011: 6CB3437EC500767328F2570C0F1D9AA9C5FEF2F6 [clean] initlog
#012: FD1BCAEF339EAE065C4369798ACAADFF44302C23 [clean] hostname
#013: F6E44B04811CC6F53C58EEBA4EACA3FE9FF91A2E [clean] consoletype
#014: 12A5A9B6657EFEE7FD619A68DA653E02A7D8C661 [clean] grep
#015: 3AF36F2916E574884850373A6E344E4F2C51DD60 [clean] sed
#016: CE516DE1DF0CD230F4A1D34EFC89491CAF3D50E4 [clean] libpcre.so.0.0.1
#017: 5EE8CD72AAD26191879E01221F5E051CE5AAE95F [clean] setsysfont
#018: 8B15F3556E892176B03D775E590F8ADF9DA727C5 [clean] unicode_start
#019: F948CF91C7AF0C2AB6AD650186A80960F5A0DAB1 [clean] kbd_mode
#020: FF02DD8E56F0B2DCFB3D9BF392F2FCE045EFE0BC [clean] dumpkeys
#021: C00804432DFBC924B867FC708CB77F2821B4D320 [clean] loadkeys
#022: DE3AC70601B9BA797774E59BEC164C0DDF11982D [clean] setfont
#023: 7334B75FDF47213FF94708D2862978D0FF36D682 [clean] gzip
#024: AEC13AA4FF01F425ACACF0782F178CDFE3D17282 [clean] minilogd
#025: 09410DDC5FE2D6E7D8A7C3CF5BB4D51ED6C4C817 [clean] sleep
……
```


PCR_{10}

Linux Application Measurements

cat /proc/tpm/pcrs

```
PCR-00: 0A 2A B1 F6 56 EA ED 4C 53 F0 C7 9D 5E 05 61 37 51 B7 1C E5
PCR-01: 5F DB 12 AD B3 34 7D D6 90 63 46 72 D8 DE 02 1C F3 3C 00 F7
PCR-02: EB B3 BA AE E7 57 4B B6 37 AA AB 67 0F 9A C1 BC EB 6F 80 F3
PCR-03: 04 FD EC DD 50 1D AF 0F 62 4C 1F 99 60 12 CF 30 44 FF 46 10
PCR-04: 28 E3 E8 F0 CA 34 ED DD 58 AA 7E 71 F6 FC AE 08 C3 88 EB 05
PCR-05: E7 23 99 CD A3 1D 37 E4 35 61 B7 1A 85 68 3B 66 7F 51 B6 B4
PCR-06: 04 FD EC DD 50 1D AF 0F 62 4C 1F 99 60 12 CF 30 44 FF 46 10
PCR-07: 04 FD EC DD 50 1D AF 0F 62 4C 1F 99 60 12 CF 30 44 FF 46 10
PCR-08: DC 0E 38 C4 F4 46 F7 BC DF C8 83 CA CC 86 E2 69 50 C5 0E 66
PCR-09: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PCR-10: 50 48 FF 78 06 63 CB BF A5 F6 43 0B DA 41 1A 15 74 C3 1A 92
PCR-11: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PCR-12: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PCR-13: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PCR-14: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PCR-15: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

静态可信度量的问题

- 2006年，德国Dresden大学的研究者发现现有的静态可信度量根存在着如下缺陷：
  - 引导装载程序（Bootloader）中存在bug
    - Dartmouth的Bear project的可信引导程序装载不完整
    - IBM IMA把加载内核镜像和度量并扩展PCR分开
    - Trusted GRUB在硬盘损坏或光盘启动时不扩展PCR
  - 无法抵御BIOS攻击
    - BIOS可刷写，机器启动时不对BIOS进行度量
  - TPM的重启攻击
    - 对LPC总线攻击在平台没有重启情况下重启TPM

静态可信度量根在实际应用中的不足

- 可拓展性：一个庞大的信任链不易拓展。为了让接收者判定系统的可信性，需要知道链中所有的可执行部件的可信性。如机器修正ROM BIOS，或者对操作系统打补丁，那么信任链就被破坏了。
- 度量的时机问题：系统安全的启动并被判定为可信之后，可以发生很多事情。比如攻击者可以在一个可信的系统中利用缓冲区溢出来替换可执行文件。这样看来，静态可信根只能提供装载时的保证（load-time guarantee），而不能提供运行时保障（run-time guarantee）。

静态可信度量根的问题

- 包容性：要使一个静态可信根对于一个第三方来说是有说服力的，必须度量TCB的所有组件。但很多系统，如Win和Linux，在开发的时候TCB定义得有问题，使得系统中所有的可执行部件（可执行程序、库、shell脚本、perl脚本等）都要被度量。因为这些部件都与安全性相关。

- 静态可信度量根
- 动态可信度量根
- Flicker: Minimal TCB Code Execution

动态可信度量根

- 作为TPM 1.2规范的一部分，TCG定义了一种新机制来度量系统的启动过程：动态可信度量根。

动态可信度量根

- 动态可信度量根由CPU发出一条新增安全指令出发，告诉TPM芯片开始要创建一个受控的和可信的执行环境
- 信任链由该条新增指令开始重置动态平台寄存器，TPM 1.2规范中规定的8个新增PCR 16~23平台寄存器，以此为基础开始构建信任链而且不需要重启整个平台
- 不像静态信任根，动态可信度量根使得可以在任意时刻开始构建信任链并且可以按照需要多次创建可信执行环境而不用重启整个平台

动态可信度量根

- 对比SRTM把BIOS核心不可修改的部分作为静态核心度量根，简称S-CRTM，那么这里新增的CPU指令就是动态核心可信度量根，简称D-CRTM。
- 目前Intel的TXT技术，以及AMD的SVM技术都是采用动态可信度量根作为基础的安全技术
  - 当然这两项技术不仅仅只是DRTM，还包括DMA保护等等一些其他安全特性，但是DRTM无疑是这些系统最核心的信任基础、信任根。

动态可信度量根

- 动态可信度量的信任链移除了BIOS及其配置，可选ROM及其配置以及前面提到的由于bugs带来漏洞的Boot loaders
  - 不会受到上述提到的缺陷1和缺陷2所带来的威胁。
- 而且动态可信度量可以开始于任意时刻且不用重新启动整个平台
  - 不存在度量的时机问题这个不足了，也不会影响到用户体验，对不可间断服务的影响也大大的降低。

动态可信度量根

- DRTM需要对处理器和芯片进行更改
- 使用DRTM技术的有：
  - Intel’s Lagrande Technology (LT) -->Trusted Execution Technology (TXT)
  - AMD’s Presidio Secure Platform Technology -->AMD’s Secure Virtual Machine (SVM)

动态可信度量根

- 引入了一条新的CPU指令用来创建一个*受控和可证明的执行环境*
  - 在Intel的技术中这条新指令被称为SENTER。
  - 对应的，AMD的新指令称为SKINIT。
  - 从功能上来说SENTER和SKINIT是一致的。

Locality

- 动态可信度量根的主要目的是引导一个可信的操作系统，并且该操作系统能支持创建多个相互隔离的安全域，在安全域中再运行客户操作系统或者是应用程序。
- 然而当可信操作系统、客户操作系统和应用程序都使用同一个TPM时，就出现了问题：
  - TPM作为一个被动的接受指令的硬件，它如何能区分度量值、命令是源自可信操作系统、客户操作系统还是应用程序呢？

Locality

- TCG 1.2规范中引入了locality的概念。
- Locality的作用就是鉴别一个TPM请求的发起者
  - 每一Locality级别用于代表一个TPM请求的发起者身份
- 规范中定义了5个locality：
  - Locality 4 可信硬件：也表示动态可信度量根；
  - Locality 3辅助组件：该Locality的使用是可选的，目前Intel使用到了，而AMD没有使用；
  - Locality 2：该locality是可信操作系统运行时所使用的；
  - Locality 1：该locality是可信操作系统为应用程序提供的执行环境所使用的；
  - Locality 0 ：用于对TPM 1.1向下兼容。

Locality

- 在TPM 1.1规范中TPM拥有16个平台配置寄存器PCR，在TPM 1.2规范中增加了8个共计24个平台配置寄存器PCR。
- 静态可信度量为了保证PCR度量值的可信，在TPM设计的时候要求PCR寄存器仅仅在平台重启的时候才可以被重置到初始值
- 而动态可信度量为了能够不影响用户体验或者是服务的连续性可以在**任意时刻重置动态PCR寄存器**（PCR 16~23）到初始值
  - 为了使TPM区分TPM指令来源于可信硬件、可信操作系统、可信操作系统中的应用程序还是普通环境（由S-CRTM及其信任链构建的传统环境），TPM 1.2规范限制了可以重置PCR 16~23的Locality。

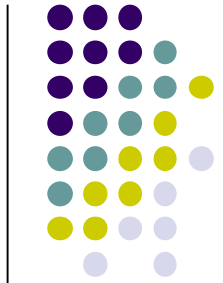
Locality

PCR索引	Alias	pcrReset	PCR是否可以被Locality 4, 3, 2, 1, 0所重置	PCR是否可以被Locality 4, 3, 2, 1, 0扩展
0-15	静态可信度量根	0	0,0,0,0,0	1,1,1,1,1
16	Debug	1	1,1,1,1,1	1,1,1,1,1
17	Locality 4	1	1,0,0,0,0	1,1,1,0,0
18	Locality 3	1	1,0,0,0,0	1,1,1,0,0
19	Locality 2	1	1,0,0,0,0	0,1,1,0,0
20	Locality 1	1	1,0,1,0,0	0,1,1,1,0
21	可信操作系统控制	1	0,0,1,0,0	0,0,1,0,0
22	可信操作系统控制	1	0,0,1,0,0	0,0,1,0,0
23	程序指定	1	1,1,1,1,1	1,1,1,1,1

Locality

- 由于引入了Locality概念以及多个Locality级别，DRTM的保护可以做得更加安全。
  - 比如说用户可以设置某个密钥只可以在Locality 1下被unseal出来并seal到可信操作系统的PCR上，那么在普通非可信操作系统中，恶意软件无法偷取到这个密钥；
  - 而当恶意软件启动DRTM进入Locality时，因为DRTM信任链更加安全，恶意软件很难进入可信操作系统，或者是让不可信系统得到与可信系统相同的度量值。各个Locality的空间相互隔离，用户不需要去担心敏感信息被不可信的操作系统所窃取。

Late Launch Background

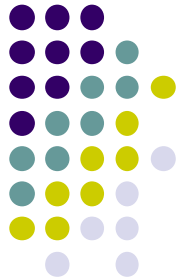


- Supported by new commodity CPUs
  - SVM for AMD
  - TXT (formerly LaGrande) for Intel
- Designed to launch a VMM without a reboot
  - Hardware-based protections ensure launch integrity
- New CPU instruction (SKINIT/SENTER) accepts a memory region as input and atomically:
  - Resets dynamic PCRs
  - Disables interrupts
  - Extends a measurement of the region into PCR 17
  - Begins executing at the start of the memory region

AMD

intel

Dynamic Root of Trust for Measurement aka: Late Launch

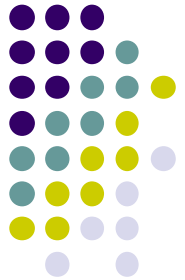


- Involves both CPU and TPM v1.2
- Security properties similar to reboot
  - Without a reboot!
  - Removes many things from TCB
    - BIOS, boot loader, DMA-enabled devices, …
- When combined with virtualization
  - VMM can be measured (MVMM)
  - Integrity of loaded code can be attested
  - Untrusted legacy OS can coexist with trusted software
- Allows introduction of new, higher-assurance software without breaking existing systems

AMD Secure Virtual Machine

- Late launch with support for attestation
  - New instruction: SKINIT (Secure Kernel Init)
  - Requires appropriate platform support (e.g., TPM 1.2)
  - Allows verifiable startup of trusted software
    - Such as a VMM
    - Based on hash comparison

SKINIT (Secure Kernel Init)



- Accepts address of Secure Loader Block (SLB)
  - Memory region up to 64 KB
- SKINIT executes atomically
  - Sets CPU state similar to INIT (soft reset)
  - Disables interrupts
  - Enables DMA protection for entire 64 KB SLB
  - Causes TPM to *reset dynamic PCRs* to 0
  - Sends SLB contents to TPM
  - **TPM hashes SLB contents and extends PCR 17**
  - Begins executing SLB

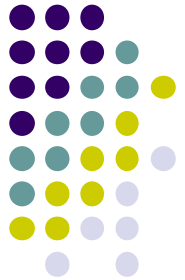
AMD's Secure Virtual Machine

- SKINIT指令对处理器进行重新初始化并建立一个安全的执行环境来运行secure loader。
  - 安全执行环境：中断被屏蔽掉，虚拟内存关闭，禁止DMA操作，除了一个处理器执行SKINIT指令，其他的处理器都处于休眠状态。
- 虚拟机管理器（VMM）或者可信操作系统的secure loader被封装到一个安全的引导块
  - 在CPU跳转到secure loader执行之前，安全引导块要被度量，其度量值被放进TPM中的PCR17当中
- 这个PCR寄存器只能通过特殊的LPC总线周期来写，所以不能够被软件来模拟。这一点就保证只可能是CPU来写PCR17
- PCR17中的值就形成了DRTM。为了防止任何针对SKINIT过程的篡改，前面描述的所有步骤和条件都是由机器自动执行。

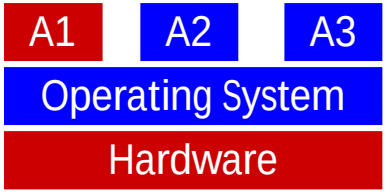
SKINIT指令

- 定义扩展扩展特征寄存器MSR_EFER的物理地址
- 将扩展特征寄存器的内容读取到EAX和EDX
- 通过EAX的值判断CPU是否支持SKINIT
- 将待执行的安全装载模块SLB的地址放到EAX
- 执行SKINIT命令

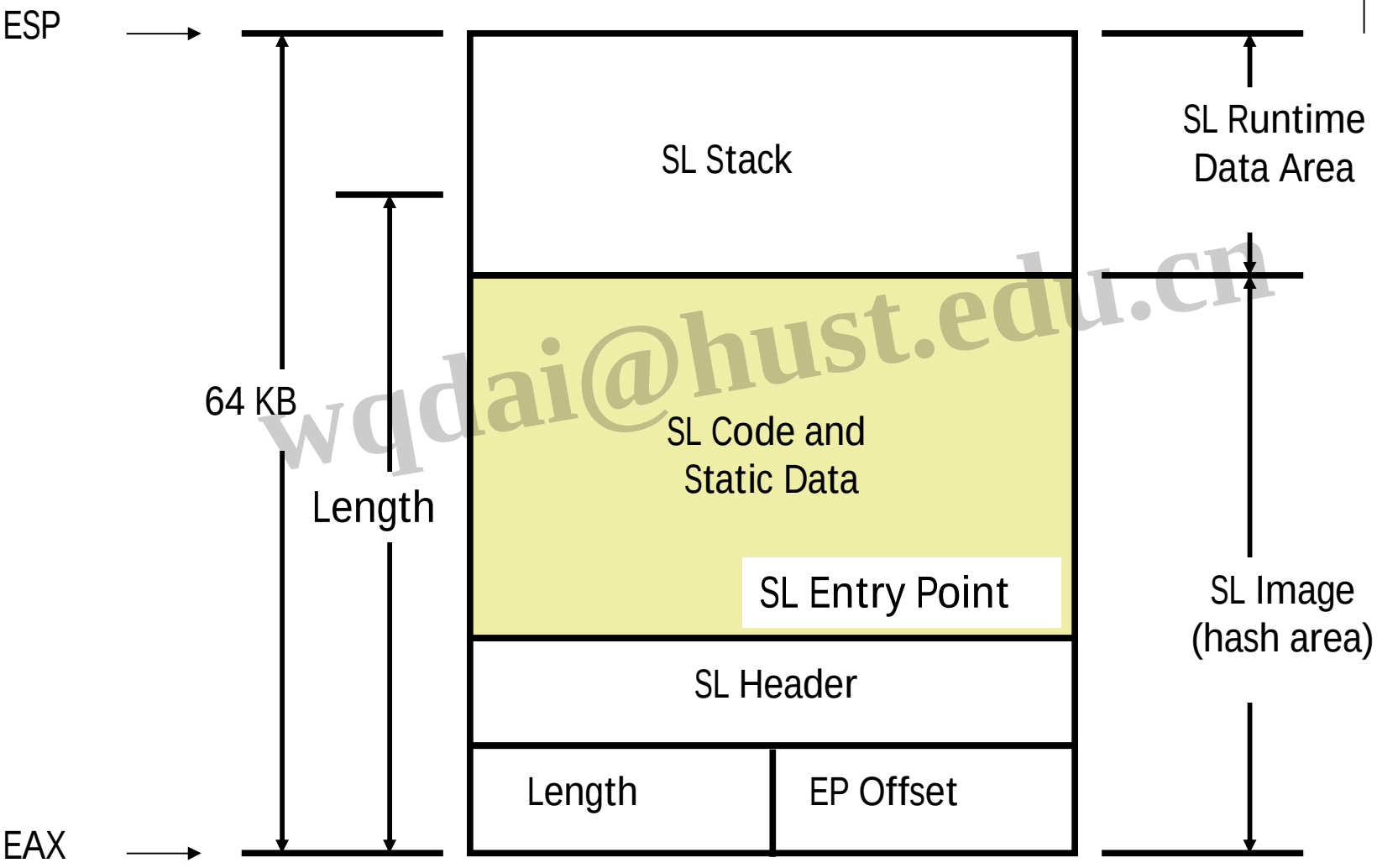
AMD SVM Security Discussion



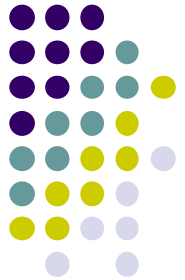
- Property: Verifiable untampered code execution
- SKINIT + TCG 1.2 provide very strong security properties
- Minimal TCB: Only hardware and application need to be trusted
  - Hardware = CPU (SKINT) + TPM



Secure Loader Block Layout



SKINIT TPM Operations



- TPM v1.2 includes notion called locality
  - Similar to software privilege level
  - 4 is highest, 0 is lowest
  - Certain PCRs associated with localities
  - PCR 17 is associated with locality 4
  - SKINIT is the only locality 4 operation
- SKINIT sends contents of SLB to TPM
  - TPM hashes SLB to create a measurement
  - TPM resets PCR17, sets PCR17 = 0
    - Distinct from boot-time value of PCR17= -1
    - Allows verifier to know that SKINIT was executed
  - TPM performs *PCR_Extend*(17, hash(SLB))

AMD’s Secure Virtual Machine

- SKINIT函数的入参struct slb *slb是安全引导块的物理地址。
- 通过读取MSR_EFER寄存器的第12比特可以判断处理器是否支持SVM拓展。
- 判断处理器支持SVM之后，将安全引导块的地址装入EAX寄存器。
- 接着执行SKINIT指令。
- 除非SKINIT指令执行失败，否则不会返回到调用SKINIT函数处。

AMD’s Secure Virtual Machine

- 安全引导块(SLB)组成了一个头结构，指出了secure loader的入口点和引导块的字节数。SLB必须被放置在一块连续的线性物理内存当中。SLB的大小被限制在64K以内。
- CS和SS指向一个32比特的4G大小的段。

AMD's Secure Virtual Machine

- Secure loader的代码紧跟在SLB头结构之后。
- 会保证屏蔽中断，禁用虚拟内存，禁止DMA，仅有一个处理器指令(SKINIT)执行。
- Secure loader要做的第一件事就是用32bit的描述符重新初始化段寄存器，使得secure loader能够读取所有的4G地址空间。
- Loader完成了初始化之后，它就将系统准备好来执行一个可信操作系统（VMM）。

TXT

- 类似于AMD的SKINIT，Intel在06年提出了Lagrande Technology (LT)，后来叫做TXT
- 首先是TXT增加了一个认证的代码模块authenticated code module (AC module，又称为SINIT ACM)，在指令GETSEC[SENTER]执行后这个代码模块首先被认证，因为该模块是经过签名的并包含一个公钥，在Intel的CPU中含有这个公钥的hash值，所以首先CPU将认证这个公钥，然后再用这个公钥去对AC module进行认证。
- 在认证完了之后才会执行AC module，而之后AC module将加载经过度量后的启动环境MLE (Measured Launched Environment)， MLE类似于AMD的SLB。

TXT

- 此外TXT还具有启动控制策略架构LCP（Launch Control Policy）来验证启动过程，该架构由三个部分构成：LCP策略引擎（LCP Policy Engine），LCP策略和LCP策略数据对象。
- LCP 策略引擎是SINIT ACM中的一部分，所以在Intel的CPU认证SINIT ACM通过后，在SINIT ACM中的这部分策略引擎将会强制读取策略，从而保证只有可信的环境才能被DRTM所启动

TXT

- LCP策略则是保存在TPM中，由SINIT ACM强制读取。这些策略是保存在TPM的非易失性存储部分，所以这些策略本身是非常安全的，并且不会因为TPM或者平台的重置而导致策略丢失。

TXT

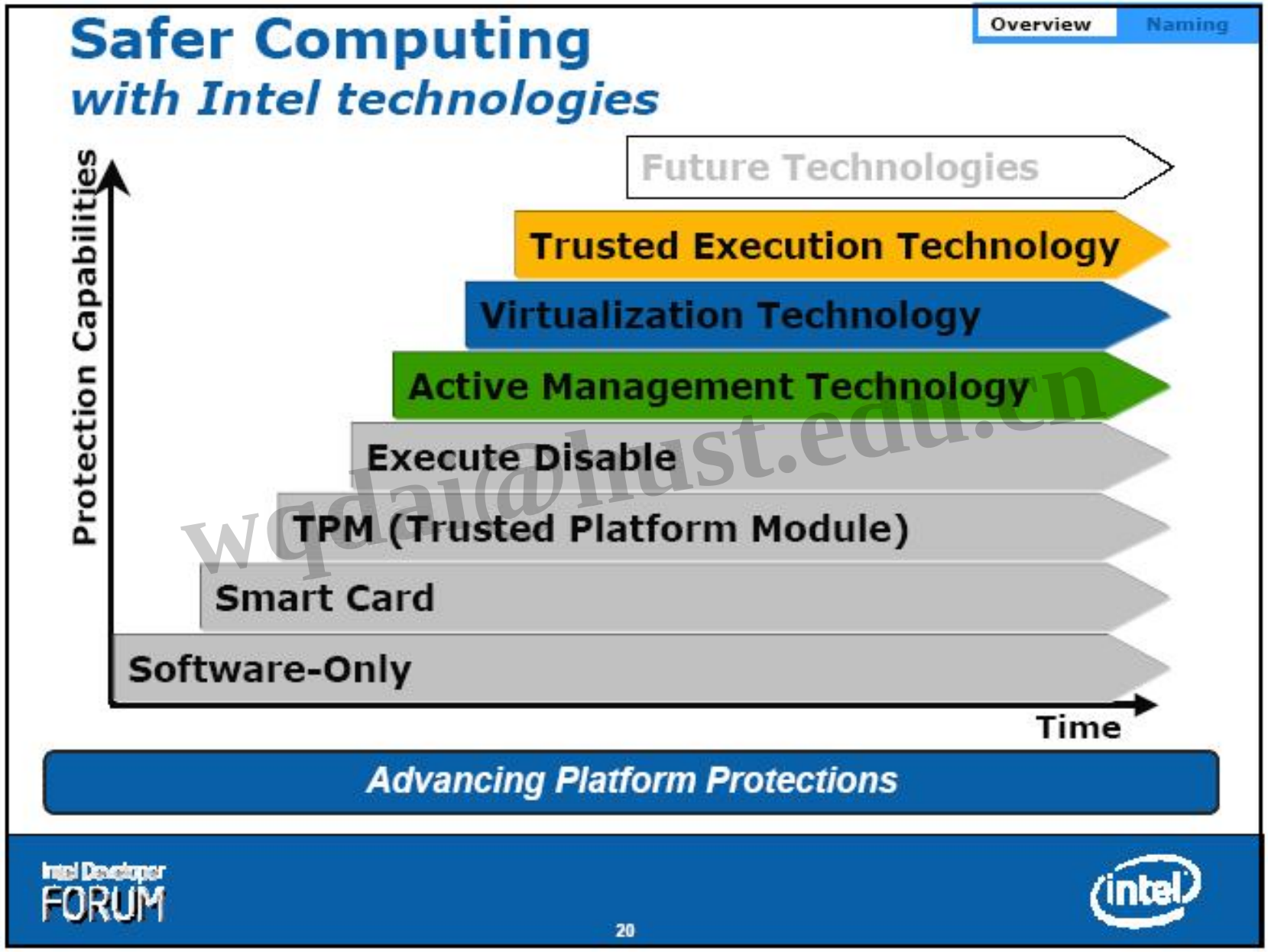
- LCP策略数据对象是由LCP策略所指向的MLE的清单或者是平台的配置清单
- 因为TPM的非易失性存储空间有限，所以一般不会把各种MLE的标准度量值直接存放，而是在非易失性存储空间存放这些清单的度量值
  - LCP策略数据对象对安全要求较低，可以存放在硬盘上。

TXT

- TXT的执行过程主要有以下4部分组成：
  - 经过度量的启动
  - MLE的初始化
  - MLE运转
  - MLE销毁

Intel Trusted eXecution Technology

- Safer Mode Extensions (SMX) and Virtual Machine Extensions (VMX)
  - Intel TXT processor instructions
- SMX introduces ***Authenticated Code*** capabilities
  - AC module loaded into CPU-internal RAM
  - AC module contains a digital signature
  - Processor calculates hash and verifies signature
  - Execution isolated from external memory and bus



Safer Mode Extensions

- Detecting and Enabling SMX.软件可以使用CPUID instruction来检测CPU对SMX操作的支持。软件将1放入EAX寄存器，执行CPUID，返回ECX的值的第6位指明了对SMX操作是否支持(即GETSEC指令是否可用)。

Table 1. CPUID Extended Feature Information in ECX

Bit #	Mnemonic	Description
0	SSE3	Streaming SIMD Extensions 3. A value of 1 indicates the processor supports Streaming SIMD Extensions 3.
3	MONITOR	MONITOR/MWAIT. A value of 1 indicates the processor supports this feature.
4	DS-CPL	CPL Qualified Debug Store. A value of 1 indicates the processor supports the extensions to the Debug Store feature to allow for branch message storage qualified by CPL.
5	VMX	Virtual Machine Extensions. A value of 1 indicates the processor supports VMX.
6	SMX	Safer Mode Extensions. A value of 1 indicates the processor supports SMX.
7	EST	Enhanced Intel SpeedStep Technology. A value of 1 indicates that the processor supports this technology.
8	TM2	Thermal Monitor 2. A value of 1 indicates whether the processor supports this technology.
10	CNXT-ID	L1 Context ID. A value of 1 indicates the L1 data cache mode can be set to either adaptive mode or shared mode. A value of 0 indicates this feature is not supported. See definition of the IA32_MISC_ENABLE MSR Bit 24 (L1 Data Cache Context Mode) for details.

1 SMX Functionality

- 在处理器中通过GETSEC指令来提供对SMX的功能的支持。这条指令支持多个子功能。在GETSEC指令执行时由EAX中的值来决定执行哪个子功能（这些子功能共享同一个操作码，0F 37）。

Index (EAX)	Leaf function	Description
0	CAPABILITIES	Return the available leaf functions of the GETSEC instruction
1	Undefined	Reserved
2	ENTERACCS	Enter authenticated code execution mode
3	EXITAC	Exit authenticated code execution mode
4	SENTER	Launch a measured environment
5	SEXIT	Exit the measured environment
6	PARAMETERS	Return SMX related parameter information
7	SMCTRL	SMX mode control
8	WAKEUP	Wake up processors from SENTER sleep state
9 - (4G-1)	Undefined	Reserved

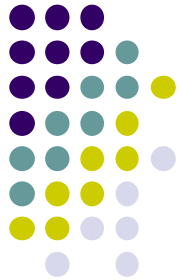
3 SMX Instruction Summary

- 系统软件首先通过执行GETSEC[CAPABILITIES]指令来查询可用的GETSEC子功能。

Table 4. Capabilities Result Encoding (EBX=0)

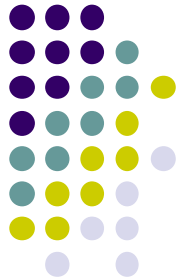
Field	Bit position	Description
Chipset present	0	Intel® TXT-capable chipset is present
Undefined	1	Reserved
ENTERACCS	2	GETSEC[ENTERACCS] is available
EXITAC	3	GETSEC[EXITAC] is available
SENTER	4	GETSEC[SENTER] is available
SEXIT	5	GETSEC[SEXIT] is available
PARAMETERS	6	GETSEC[PARAMETERS] is available
SMCTRL	7	GETSEC[SMCTRL] is available
WAKEUP	8	GETSEC[WAKEUP] is available
Undefined	30:9	Reserved
ExtendedLeafs	31	Reserved for Extended Information Reporting

Measured Launched Environment



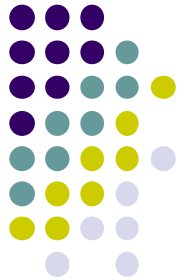
- Main objective: Protected Execution, i.e. provide applications with an execution environment where they can be executed **without being observed or compromised by untrusted applications**
- This environment is called *Measured Launched Environment*
- TXT protects the launch and the execution of this MLE
- MLE can be launch at anytime, including long after the boot

Launch of the MLE — Process



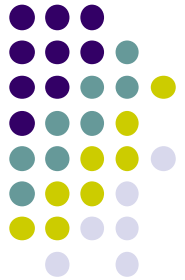
- The launching environment loads the MLE and Authenticated Code (AC) in memory
- The launching environment calls the GETSEC[SENTER] instruction
- The processor loads, authenticates (digital signature) and executes the AC
- The AC checks the configuration of the chipset and the processors
- The AC measures the MLE, sends the measurements to the TPM and launches the MLE

Protection of the MLE — DMA



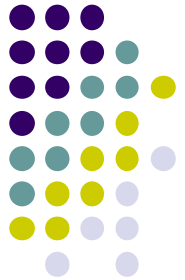
- Need to protect the MLE against unauthorized modifications
- DMA: protection using the Intel VT-d technology (requires chipset modifications) to prevent unauthorized DMA transfers to/from a memory area belonging to the MLE

Protection of the MLE — Misc.



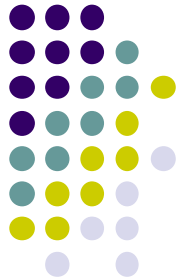
- Protected Input/Output: data encryption between the driver in the MLE and the I/O device (e.g. mouse, keyboard...)
- Protected Graphics
  - Data encryption between the driver in the MLE and the graphic card
  - Proof to the user that what is displayed in a part of the screen really comes from the MLE

Intel TXT vs. AMD SVM



- **Both Removes BIOS/bootloader/OS/etc**. from trust chain
  - Creates dynamic root of trust (DRTM)
- AMD SVM does not support *authenticated code*
  - Whether this will be significant is not yet known
  - AC needs code signed such that chipset can verify it
  - Chipset needs public key, crypto capabilities
    - Additional complexity
    - Code update issues

What is Trusted Boot (tboot)?

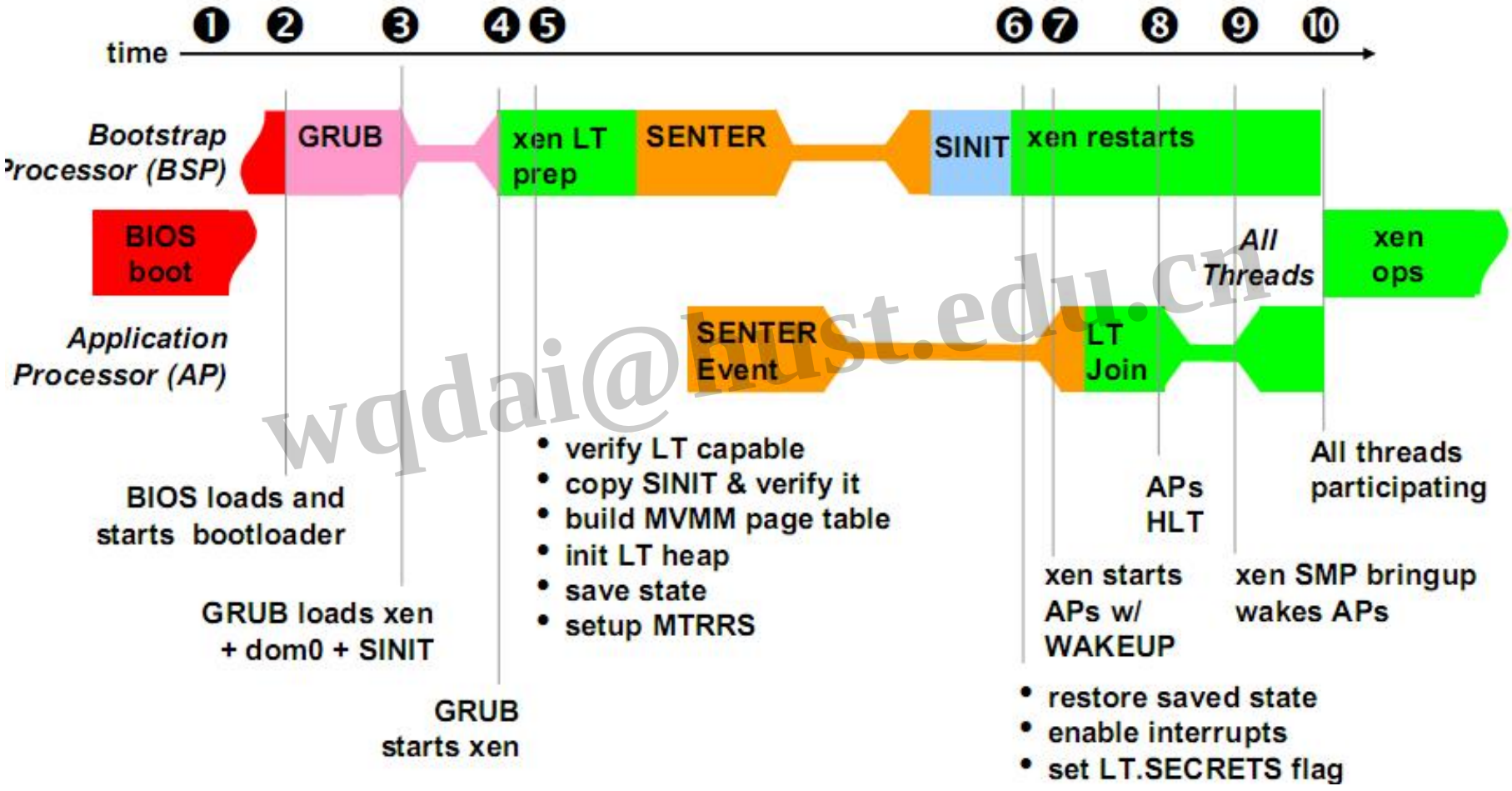


- Open source, pre-kernel/VMM module
- Uses Intel TXT to perform verified launch of OS kernel/VMM
  - Today only supports Xen
- Project also contains tools for policy creation and Provisioning
  - Intel TXT Launch Control Policy (LCP)
  - Tboot Verified Launch policy

Trusted Boot provides a foundation for a Trusted Xen

- Root of trust is in hardware: Intel TXT dynamic launch
- Tboot is verified by Intel TXT Launch Control Policy (LCP)
  - Part of measured launch
- Tboot Verified Launch verifies Xen and Dom0 (+ initrd)

Tboot for Xen



Flicker: Minimal TCB Code Execution

Jonathan M. McCune
Carnegie Mellon University

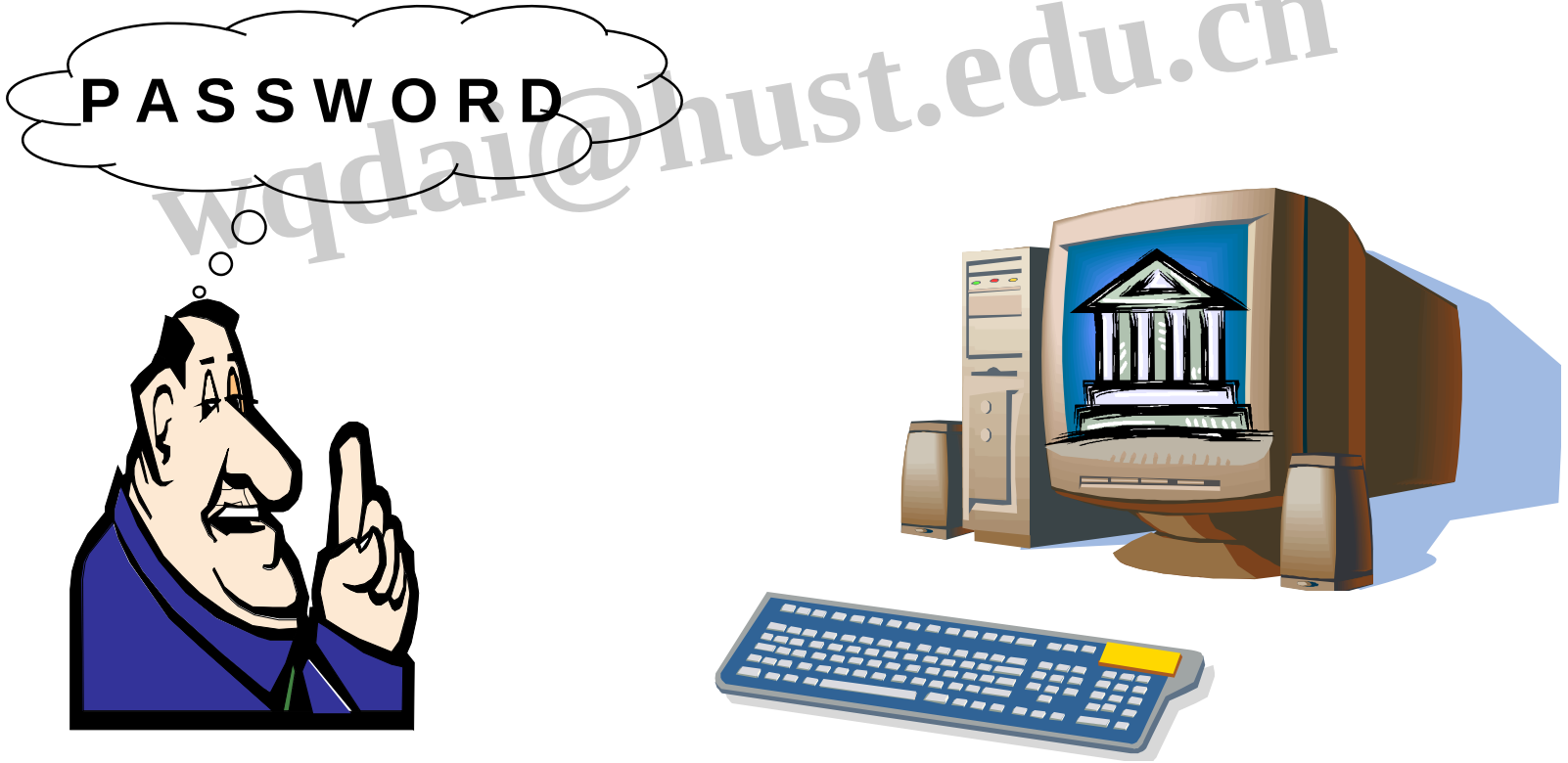
March 27, 2008

Bryan Parno, Arvind Seshadri
Adrian Perrig, Michael Reiter

Carnegie Mellon CyLab
CONFIDENCE FOR A NETWORKED WORLD

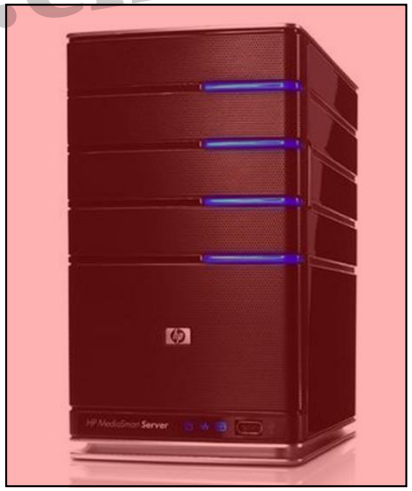
Password Reuse

- People often use 1 password for 2+ websites
- Banking, social networking, file sharing, …



Password Exposure

- Password provided to compromised web server



www.myhobby.com is compromised!

Password Verification

- What if…
  - A compromised OS cannot learn the password
  - Only essential code can access password
    - Decrypt SSL traffic
    - Salt and hash password
    - Compare with stored hash
    - Output MATCH or FAILURE
  - Can remotely verify this is so
- Requires strong system security

- What about zero knowledge protocols?
  - A viable alternative for passwords
  - Our techniques are more general
    - Password verification is just an example

Outline

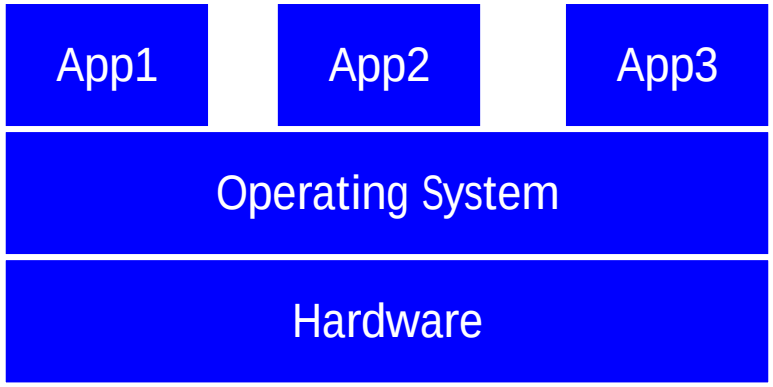
1. **Existing approaches to system security**
2. Remote attestation and verification
3. Static root of trust for measurement
4. Dynamic root of trust for measurement
5. Flicker: Minimal TCB Code Execution

- Optional
  - Example: IBM Integrity Measurement Arch.
  - Specifics of AMD SVM / Intel TXT

Some Current Approaches

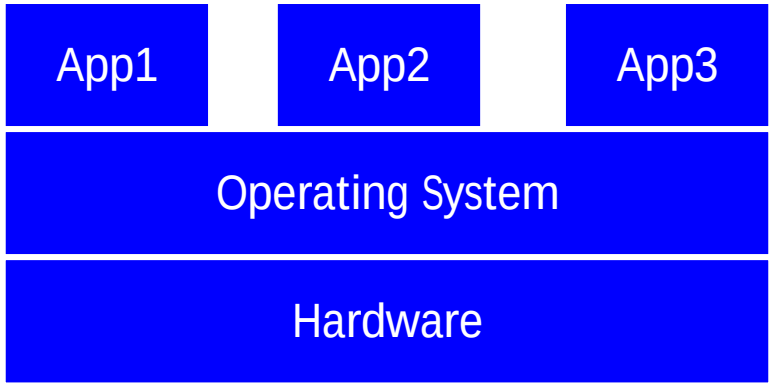
- Program code in ROM
- Secure boot
- Virtual-machine-based isolation

- Evaluation metric: size of Trusted Computing Base (TCB)



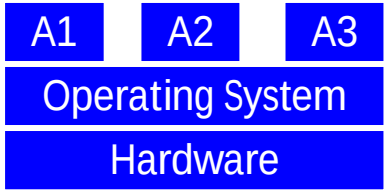
Security Properties to Consider

- How can we trust operations that our devices perform?
- How can we trust App1?
- What if App2 has a security vulnerability?
- What if Operating System has a security vulnerability?



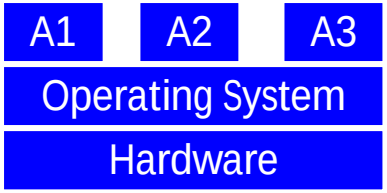
Program Code in ROM

- Advantages
  - Simplicity
  - Adversary cannot inject any additional software
- Disadvantages
  - Cannot update software (without exchanging ROM)
  - Adversary can still use control-flow attack
  - Entire system is in TCB, no isolation
- Verdict
  - Impractical for current systems
  - Code updates are critical
    - Bug fixes
    - New features



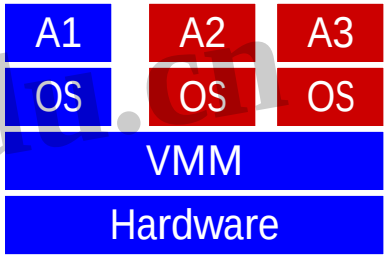
Secure Boot

- Boot process uses signature chain
  - BIOS verifies signature on boot loader
  - Boot loader verifies signature on OS, ...
- Advantages
  - Only approved software can be loaded
    - Assuming no vulnerabilities
- Disadvantages
  - Adversary only needs to compromise singe component
  - Entire system is in TCB, no isolation
  - Not all software is commercial
- Verdict
  - Entire system is still part of TCB
  - Relatively weak security guarantee



Virtual-machine-based Isolation

- Approach: Isolate applications by executing them inside different Virtual Machines
- Advantages
  - Smaller TCB
  - Isolation between applications
- Disadvantages
  - VMM is still large and part of TCB
  - Relatively complex, not suitable for average user
- Verdict: Smaller TCB, step in right direction



Flicker: An Execution Infrastructure for TCB Minimization

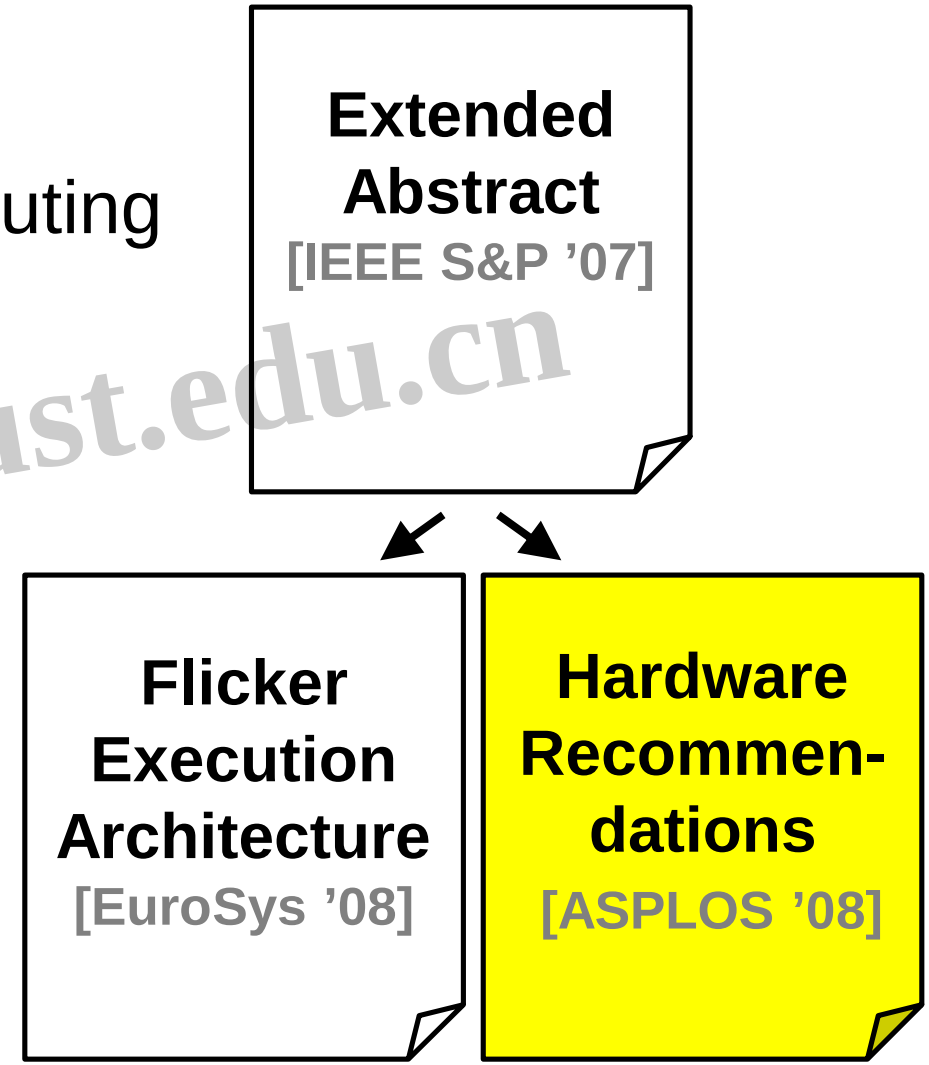
Jonathan McCune[1], Bryan Parno[1], Adrian Perrig[1], Michael Reiter[2], and Hiroshi Isozaki[1,3]

[1] *Carnegie Mellon University*
[2] *University of North Carolina*
[3] *Toshiba Corporation*

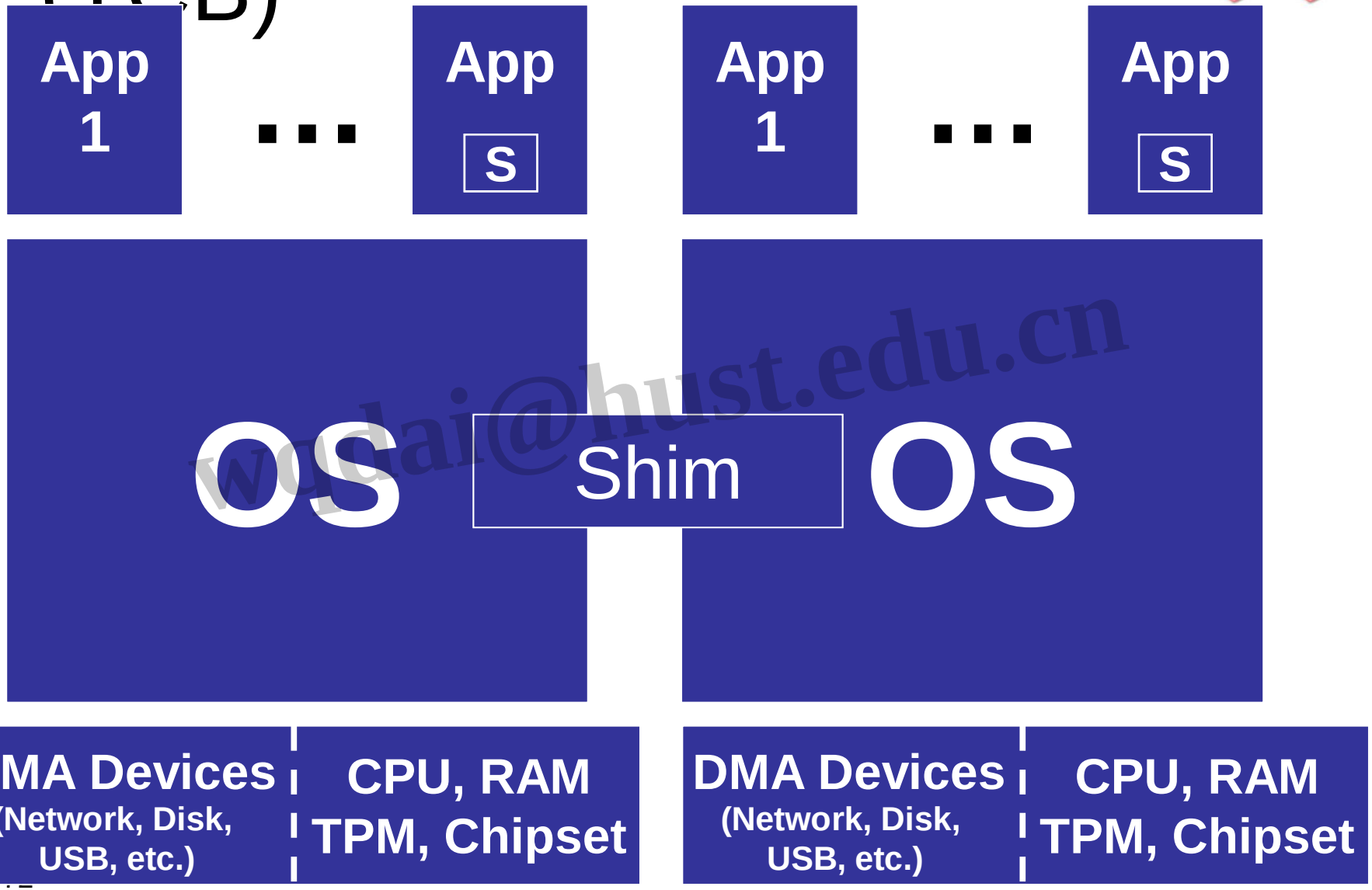
April 4, 2008

The Flicker Project

- Goals
  - Minimize Trusted Computing Base
  - Do not trust OS
  - Commodity hardware
- This talk
  - Performance today
  - Recommendations for tomorrow



Trusted Computing Base (TCB)



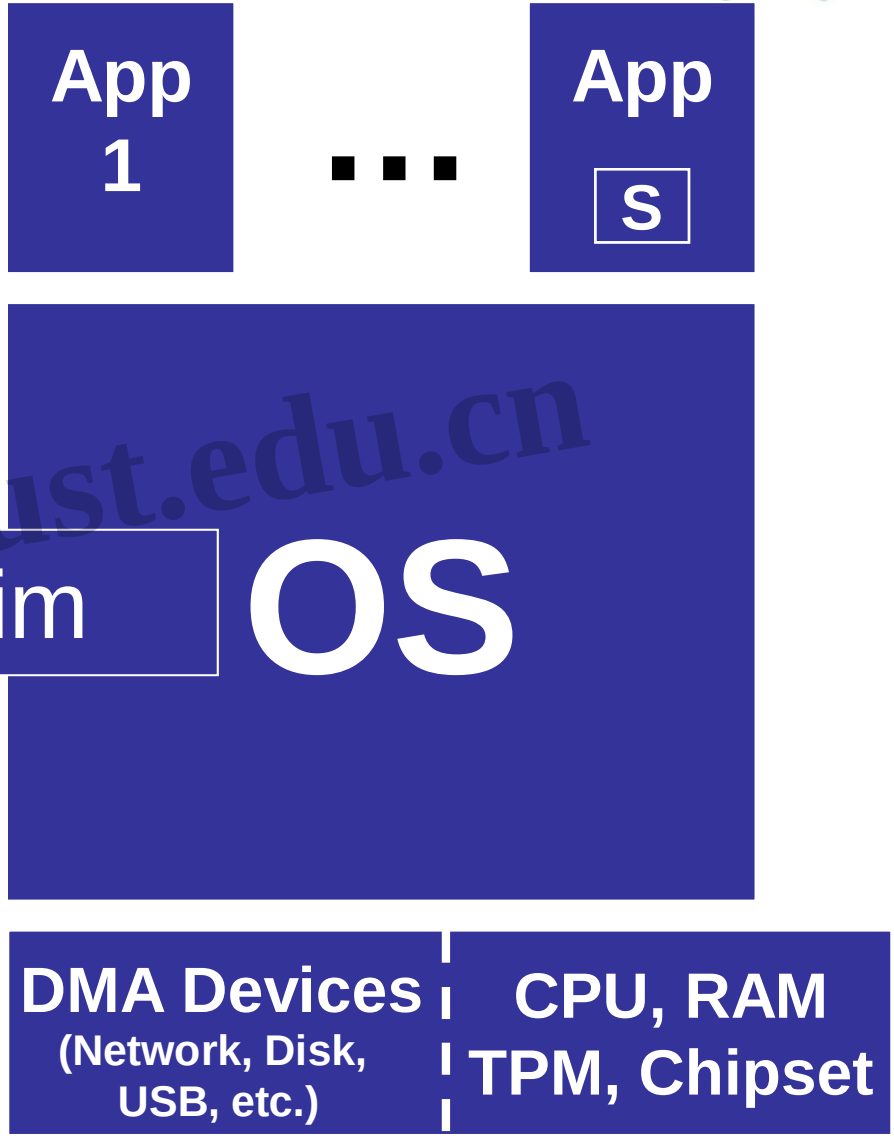
TCB Reduction with Flicker

Today, TCB for sensitive code S:

- Includes App
- Includes OS
- Includes other Apps
- Includes hardwa

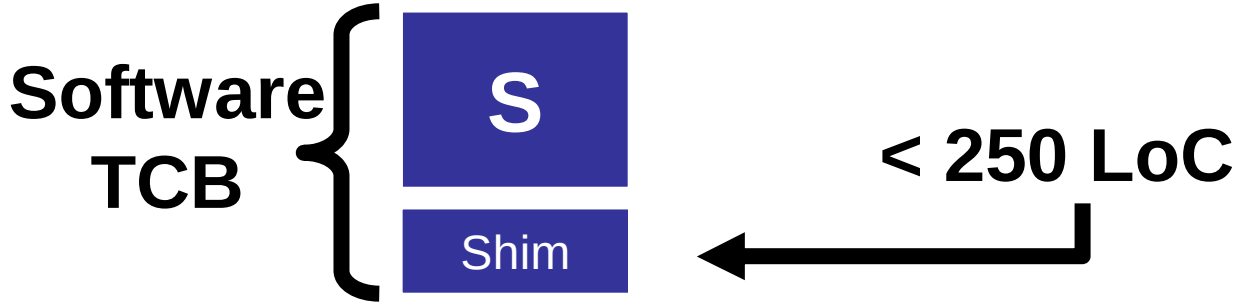
With Flicker, S's TCB:

- Includes Shim
- Includes some hardware



Flicker's Properties

- Isolate security-sensitive code execution from all other code and devices
- Attest to security-sensitive code and its arguments and nothing else
- Convince a remote party that security-sensitive code was protected
- Add < 250 LoC to the software TCB



Adversary Capabilities

- Run arbitrary code with maximum privileges
- Subvert any DMA-enabled device
  - E.g., network cards, USB devices, hard drives
- Perform limited hardware attacks
  - E.g., power cycle the machine
  - Excludes physically monitoring/modifying CPU-to-RAM communication

Basic Flicker Architecture

1. Pause current execution environment (legacy OS)
2. Execute security-sensitive code using *late launch*
3. Preserve session-state with TPM sealed storage
4. Resume previous environment

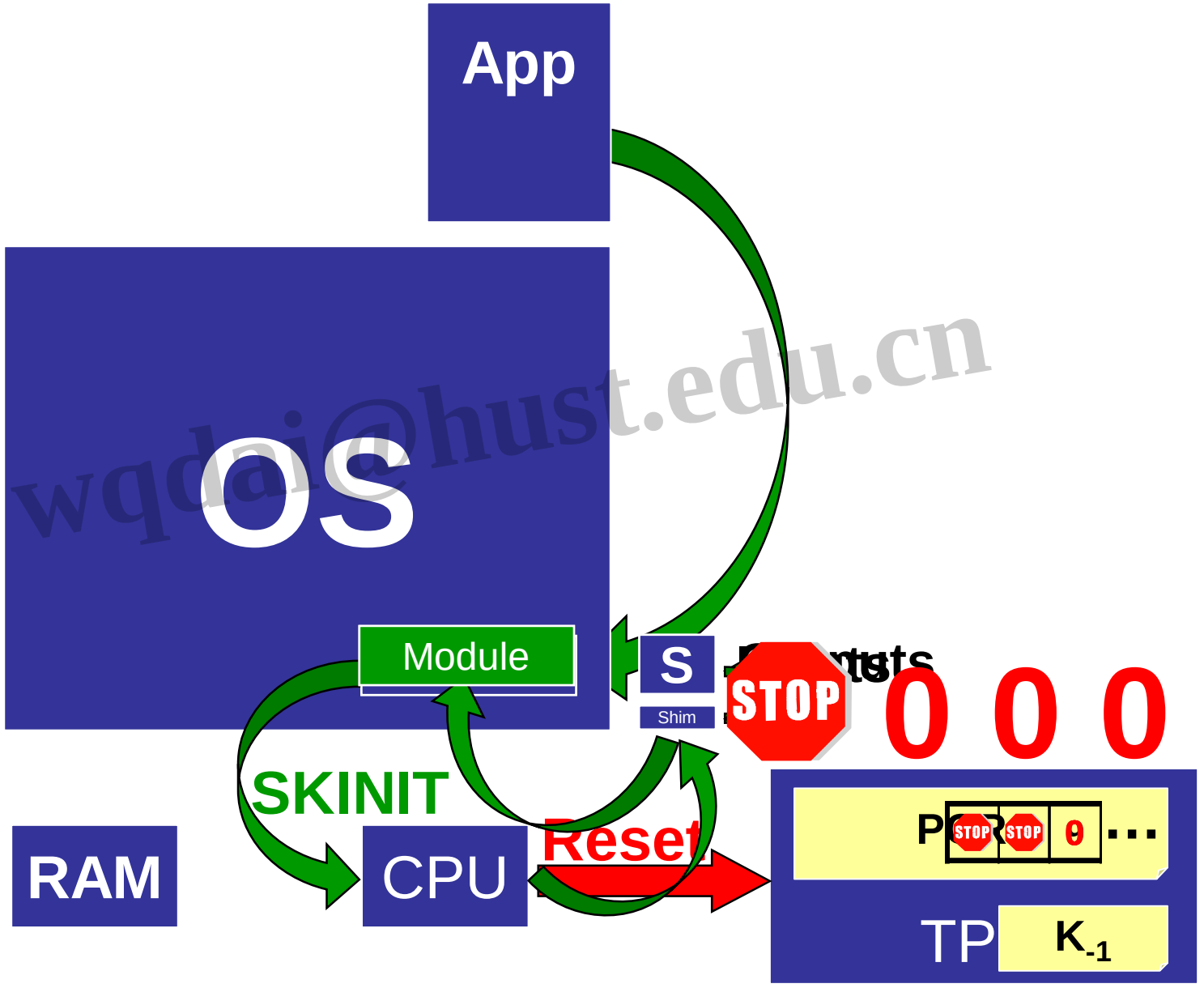
Not the intended use of late launch, sealed storage

- Intended use is an infrequent, disruptive event
  - Use to replace lowest-level system software
  - All but one CPU must be halted for late launch
- Our use resembles a context switch
  - Setup protected execution environment for sensitive app
  - Late launch and TPM sealed storage on the critical path

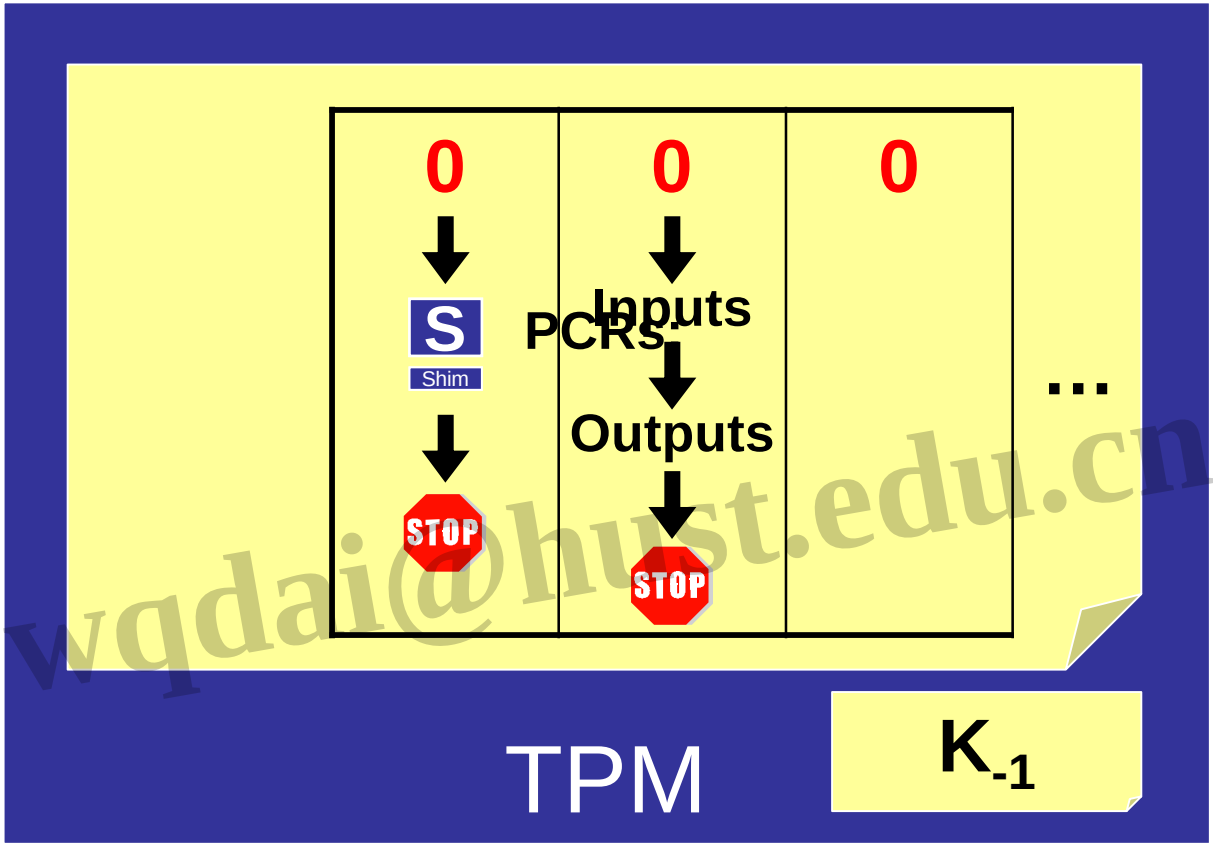
Architecture Overview

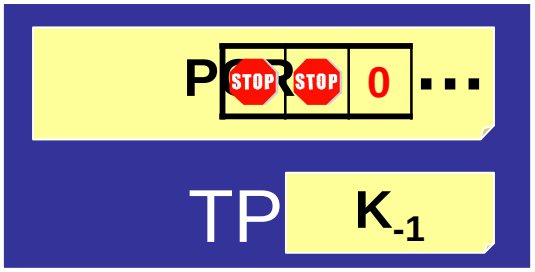
- Core technique
  - Pause current execution environment (untrusted OS)
  - Execute security-sensitive code with hardware-enforced isolation
  - Resume previous execution
- Extensions
  - Attest **only** to code execution and protection
  - Preserve state securely across invocations
  - Establish secure communication with remote parties

Execution Flow

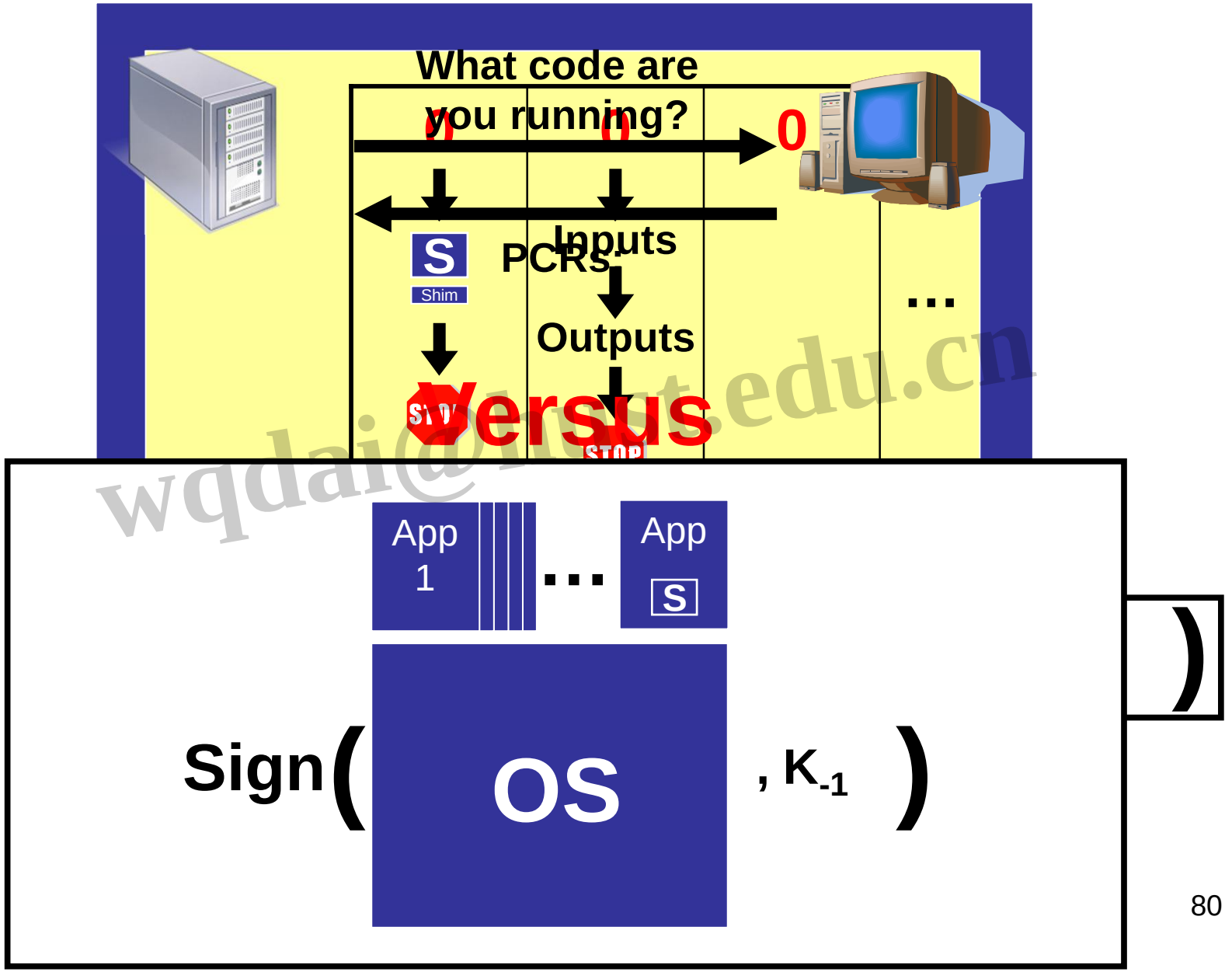


Attestation



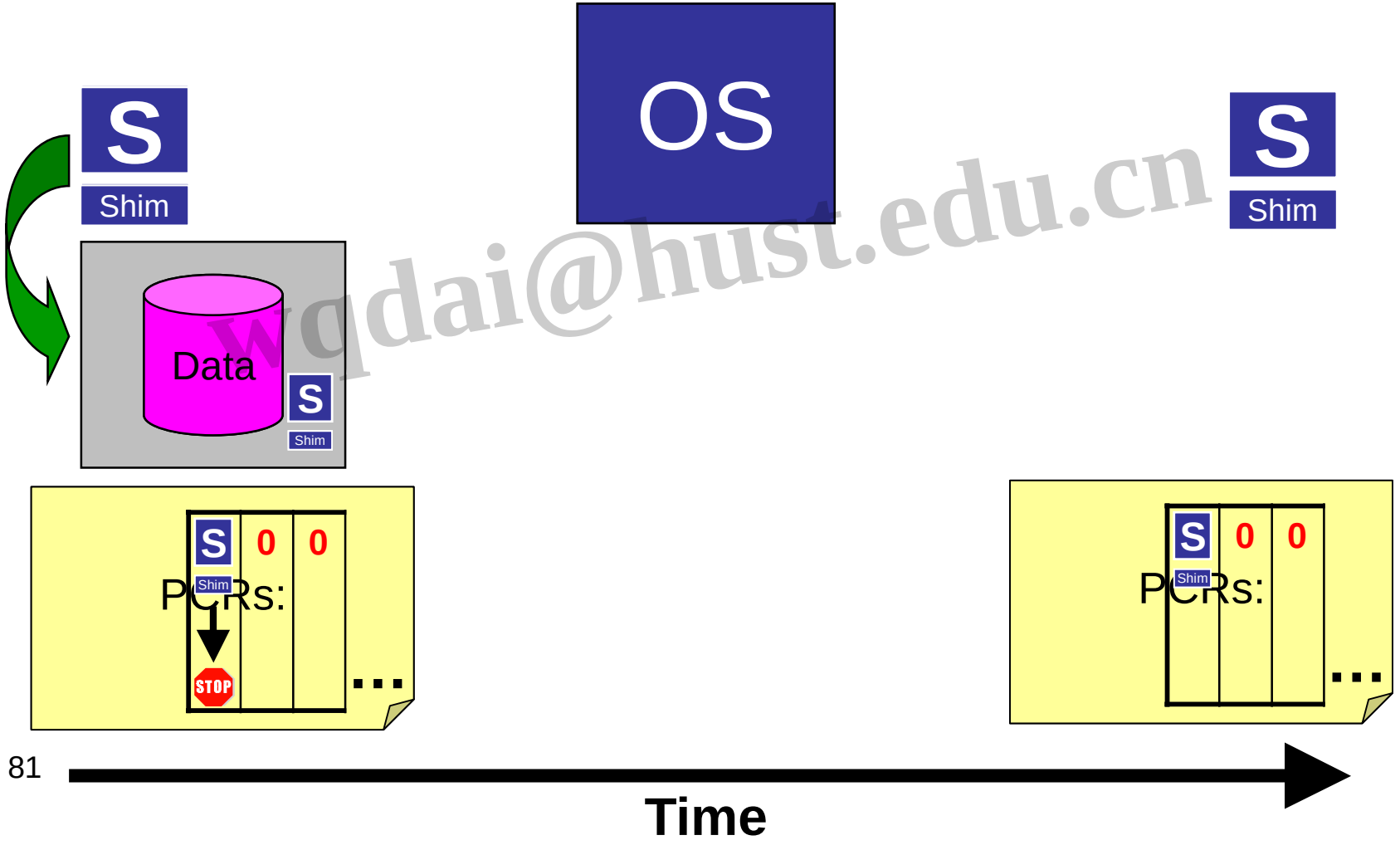


Attestation



Context Switch with Sealed Storage

- **Seal** data under combination of code, inputs, outputs
  - Data unavailable to other code



Developing With Flicker

- Sensitive code linked against the Flicker library
- Customized linker script lays out binary
- Application interacts with Flicker via a Flicker kernel module


```
#include “flick
const char* msg = “He
void flicker_main(void *    puts) {
        for(int i=0;i  13;i++)
            OUTPUT[i] = msg[i];
        }
```

Made available at:
/proc/flicker/output

Default Functionality

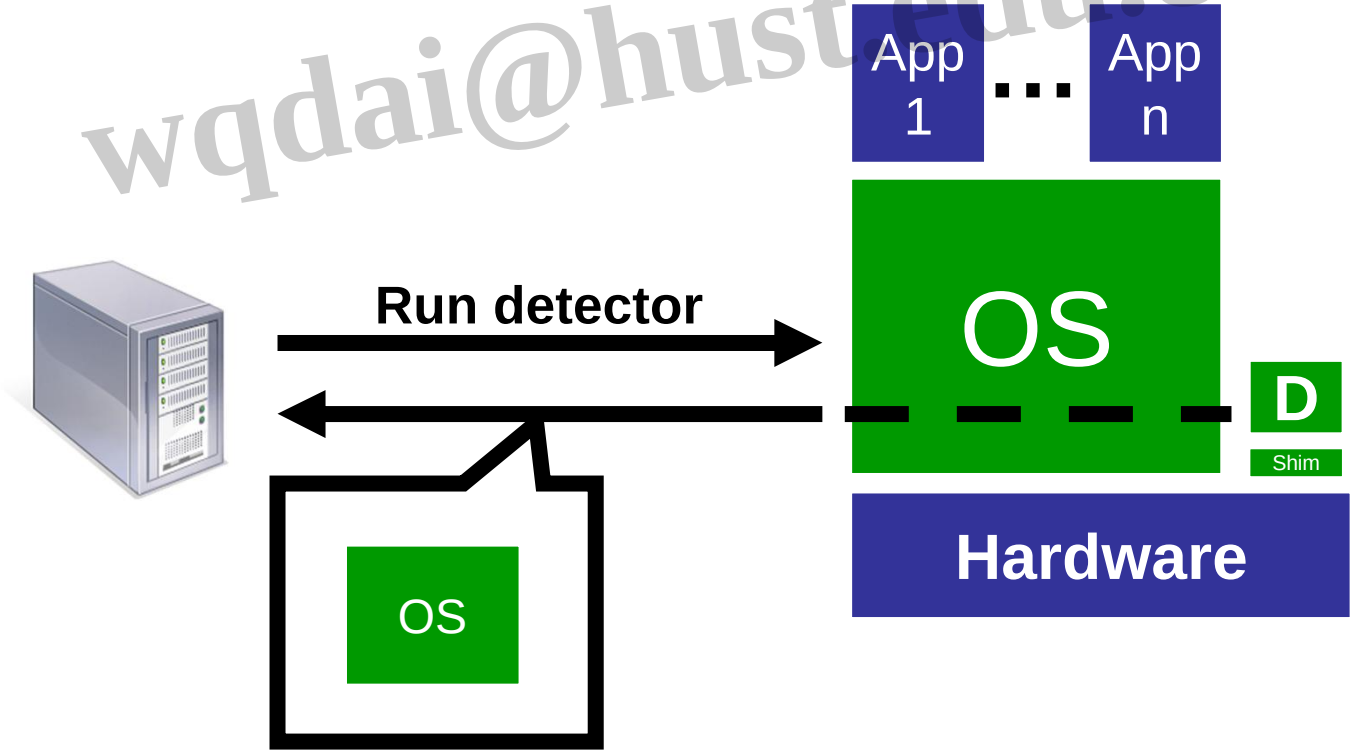
- Shim can execute arbitrary x86 code but provides very limited functionality
- Fortunately, many security-sensitive functions do not require much
  - E.g., key generation, encryption/decryption, FFT
- Functionality can be added to support a particular security-sensitive operation
- We have partially automated the extraction of support code for security-sensitive code

Existing Flicker Modules

- **OS Protection** Memory protection, ring 3 execution
- **Crypto** Crypto ops (RSA, SHA-1, etc.)
- **Memory Alloc.** Malloc/free/realloc
- **Secure Channel** Secure remote communication
- **TPM Driver** Communicate with TPM
- **TPM Utilities** Perform TPM ops

Application: Rootkit Detector

- Administrator can check the integrity of remote hosts
  - E.g., only allow uncompromised laptops to connect to the corporate VPN



Application: SSH Passwords

nonce

Start

S
Shim

Gen {K, K^{-1}}

K^{-1} S Shim

S
Shim
K

K

$Encrypt_K(passwd)$

K^{-1} S Shim

$Encrypt_K(passwd)$

OK!

Other Applications Implemented

- Enhanced Certificate Authority (CA)
  - Private signing key isolated from entire system
- Verifiable distributed computing
  - Verifiably perform a computational task on a remote computer
  - Ex: SETI@Home, Folding@Home, distcc

Generic Context-Switch Overhead

- Each Flicker context switch requires:
  - SKINIT
  - TPM-based protection of application state

Results

SKINIT	14 ms
Unseal application state	905 ms
Reseal application state	20 ms
Total	**939 ms**

Rootkit Detection Performance

SKINIT	14 ms
Hash of Kernel	22 ms
PCR Extend Result	1 ms
TPM Quote	973 ms
Total	**1023 m**

37 ms Disruption

Non-Disruptive

Running detector every 30 seconds has negligible impact on system throughput

SSH Performance

- Setup time (217 ms) dominated by key generation (185 ms)
- Password verification (937 ms) dominated by TPM Unseal (905 ms)

Adds < 2 seconds of delay to client login

Optimizing Flicker's Performance

- Non-volatile storage
  - Access control based on PCRs
  - Read in 20ms, Write in 200 ms
  - Store a symmetric key for "sealing" and "unsealing" state

Reduces context-switch overhead by an order of magnitude

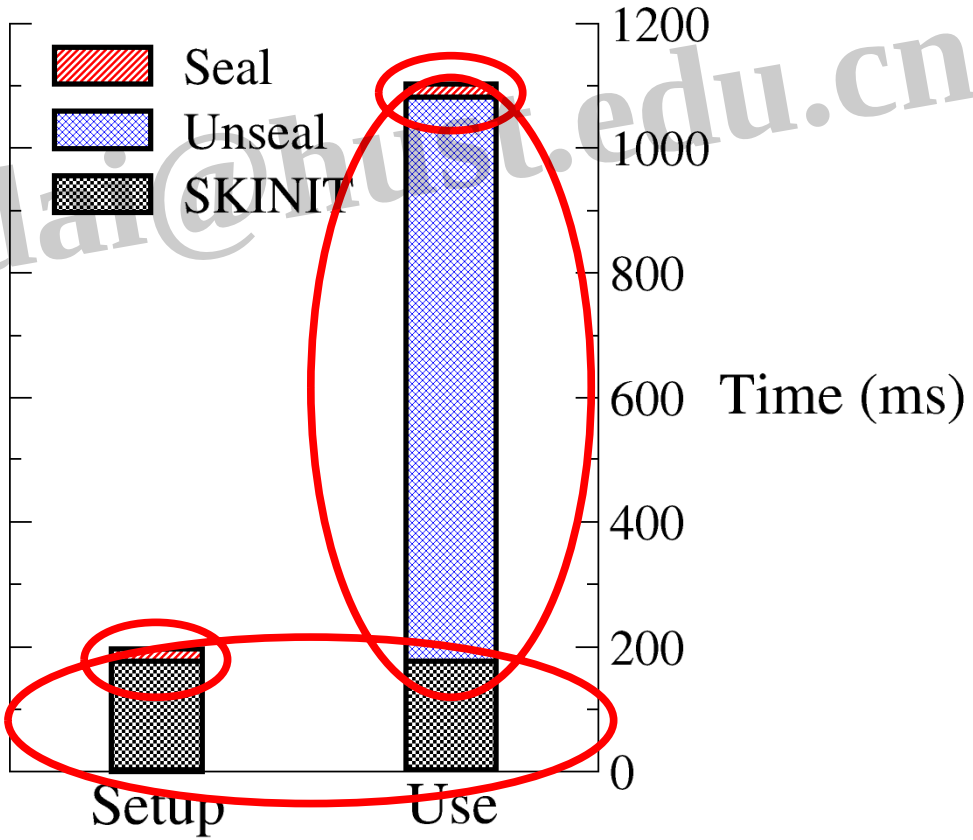
Hardware Performance Improvements

[ASPLOS 2008]

- Late launch cost only incurred when Flicker session launches
- TPM (Un)Seal only used for long-term storage
- Multicore systems remain interactive
- Context switch overheads (common case) resemble VM switches today (~0.5 µs)

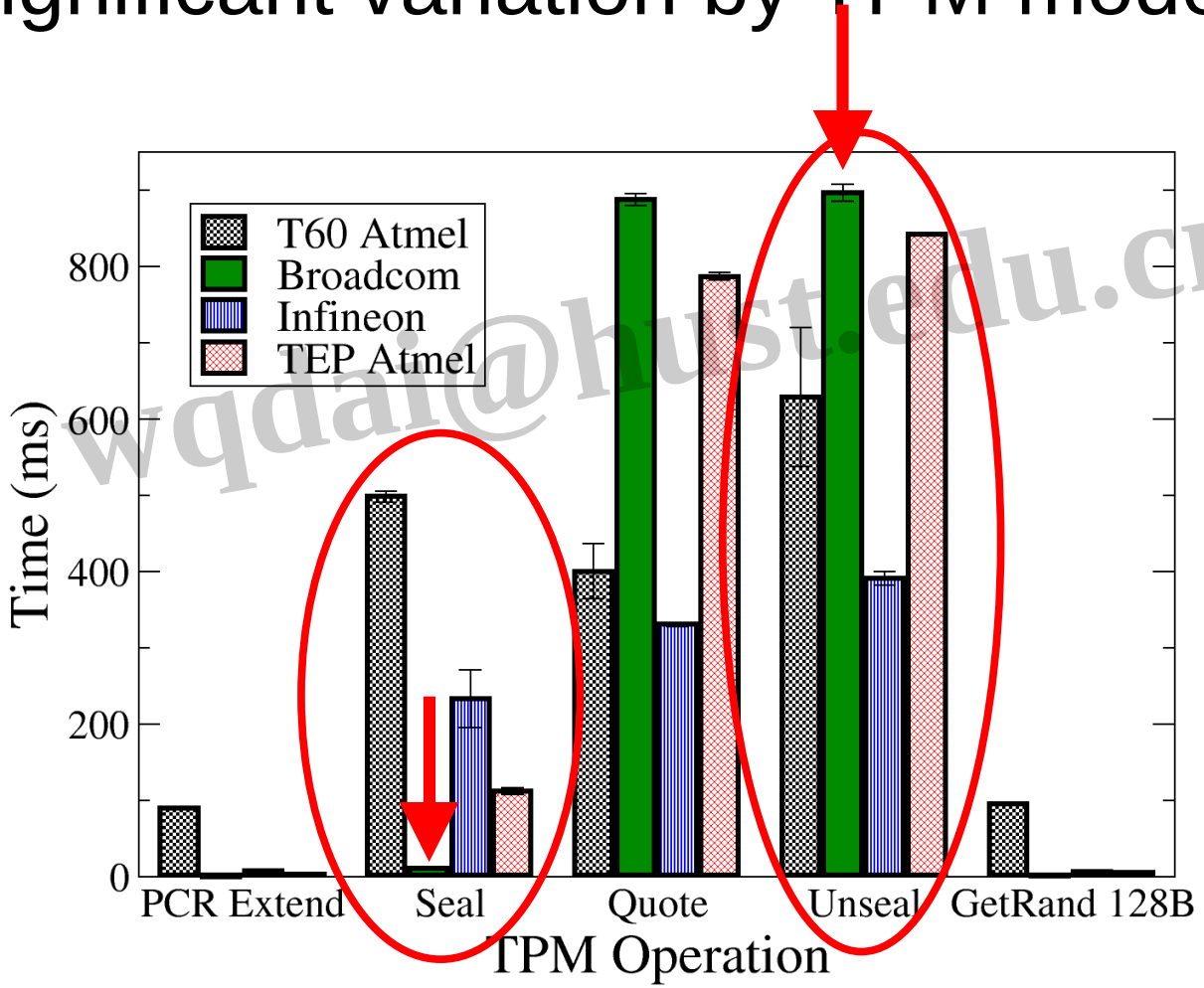
TPM-related Performance

- During every Flicker context switch
  - Application state protection while OS runs



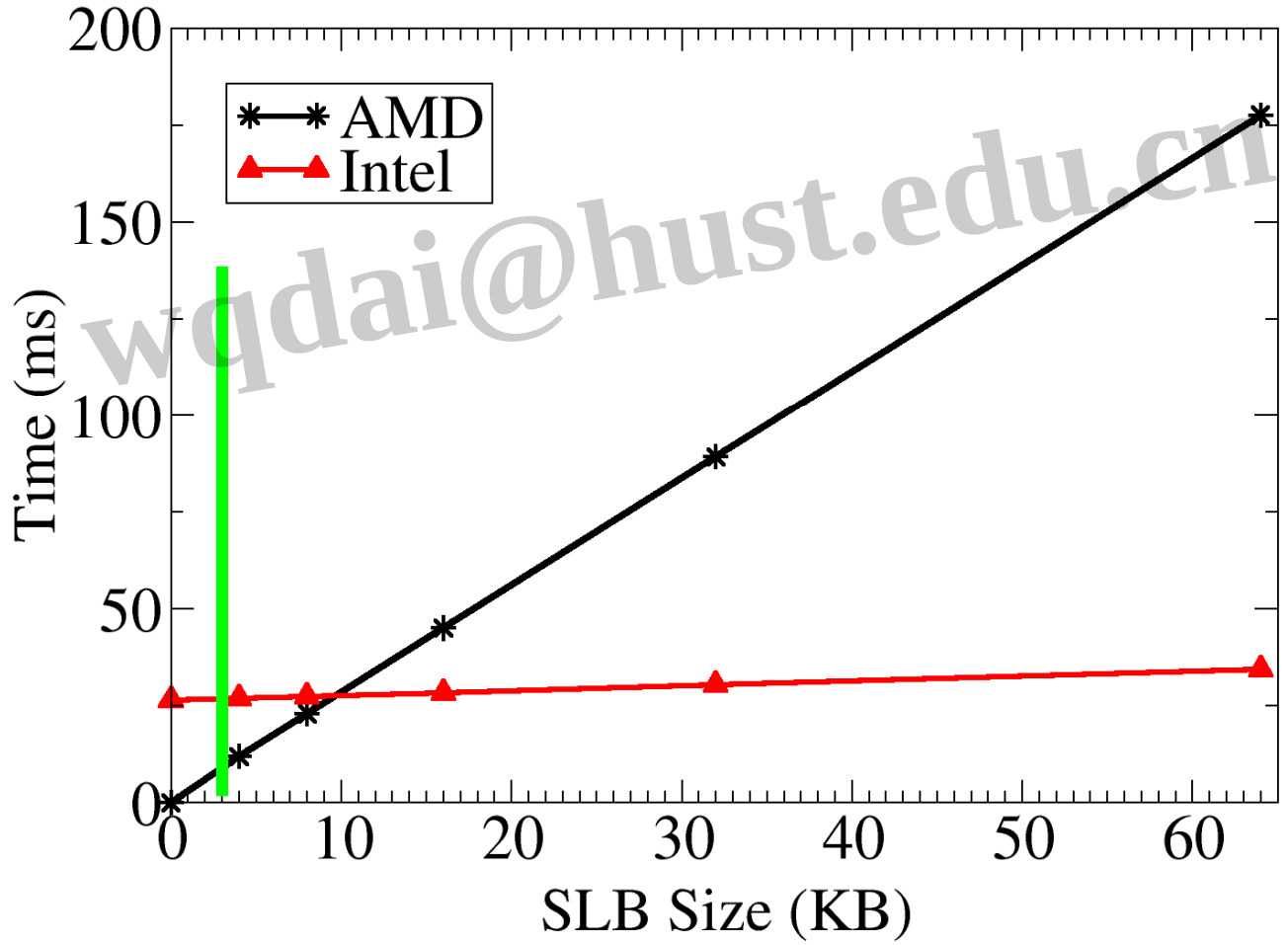
TPM Microbenchmarks

- Significant variation by TPM model



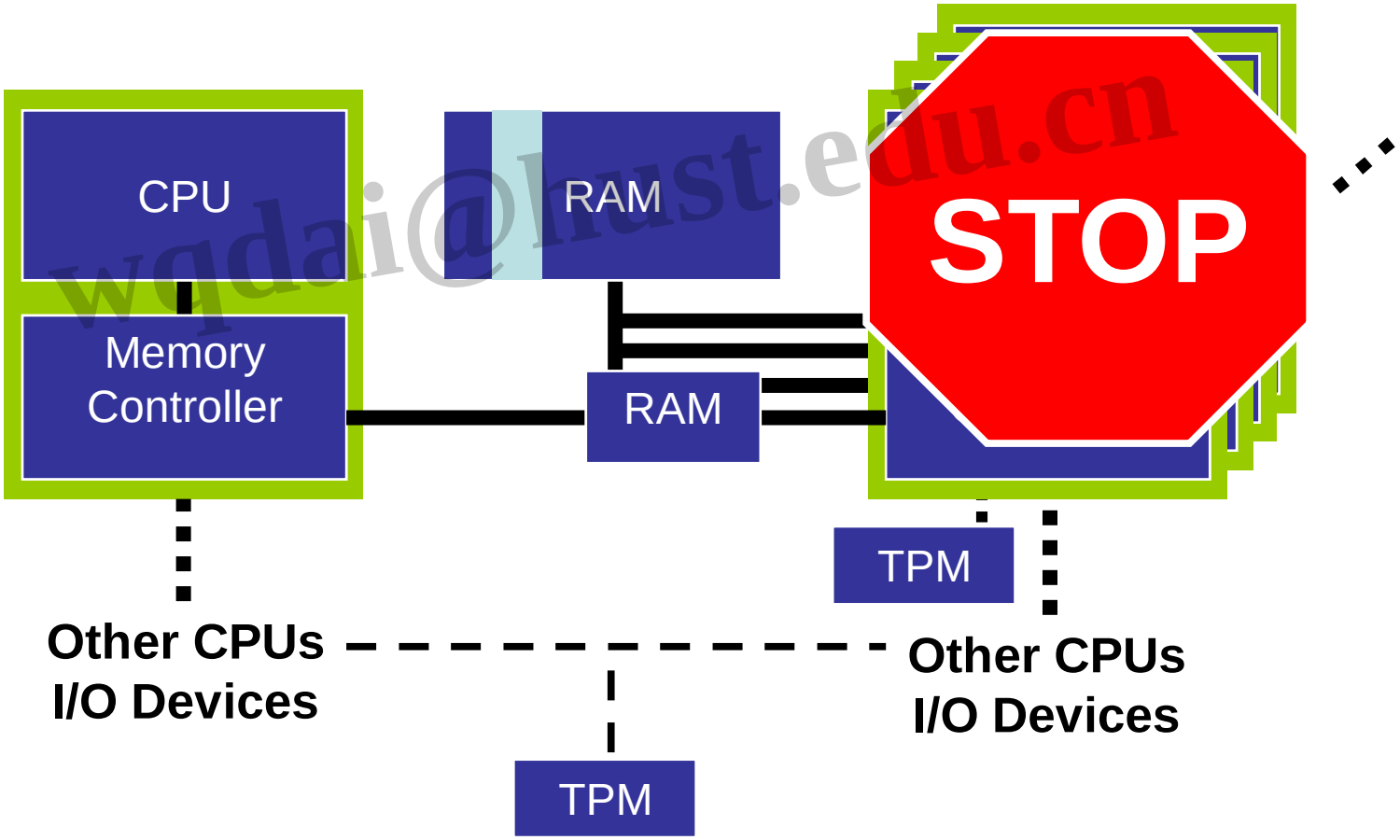
Breakdown of Late Launch Overhead

- After ~4KB, code can measure itself



Late Launch Performance

- Late launch requires APs to stop execution
  - More cores = more expensive



Overheads and Recommendations

O1 Late launch and TPM sealed storage overhead on every context switch

R1 Secure context switch between sensitive applications without TPM

O2 Other cores must halt to enable late launch

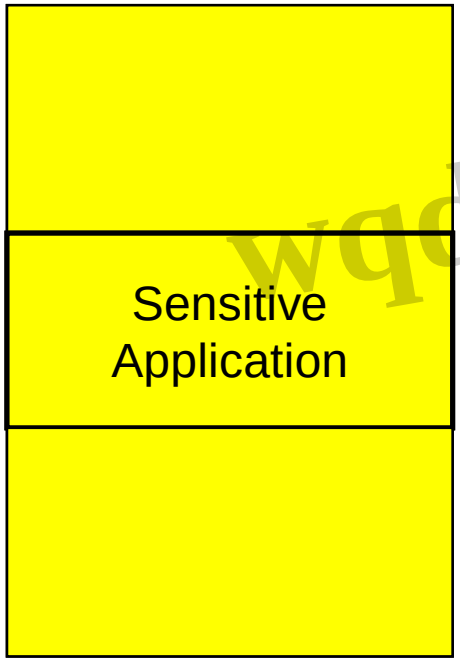
R2 Secure execution on multiple CPUs concurrently

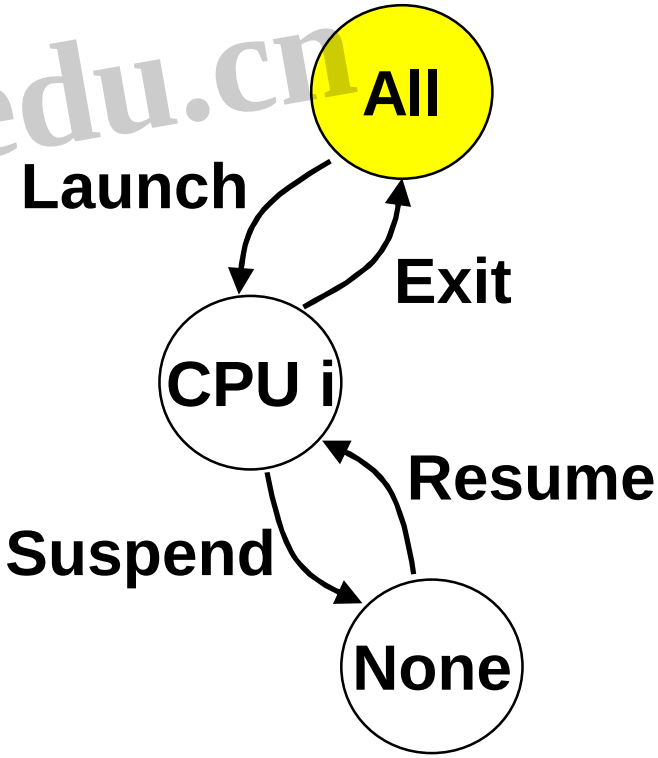
O3 TPM equipped to store measurements from only one late launch

R3 TPM support for concurrent Flicker sessions

New States for Memory Pages

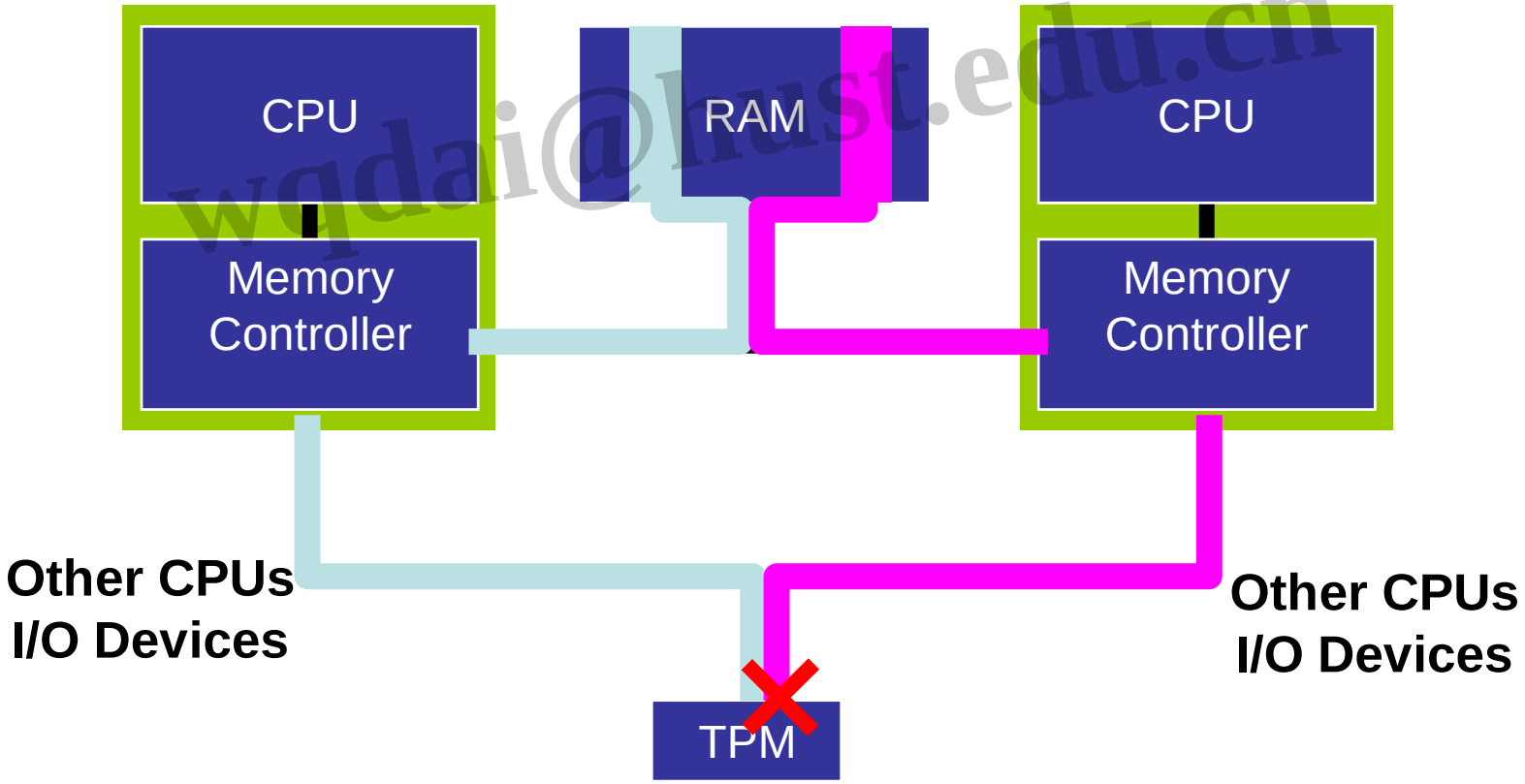
- Avoid TPM for short-term data protection
- Memory Controller already supports DMA protection vector





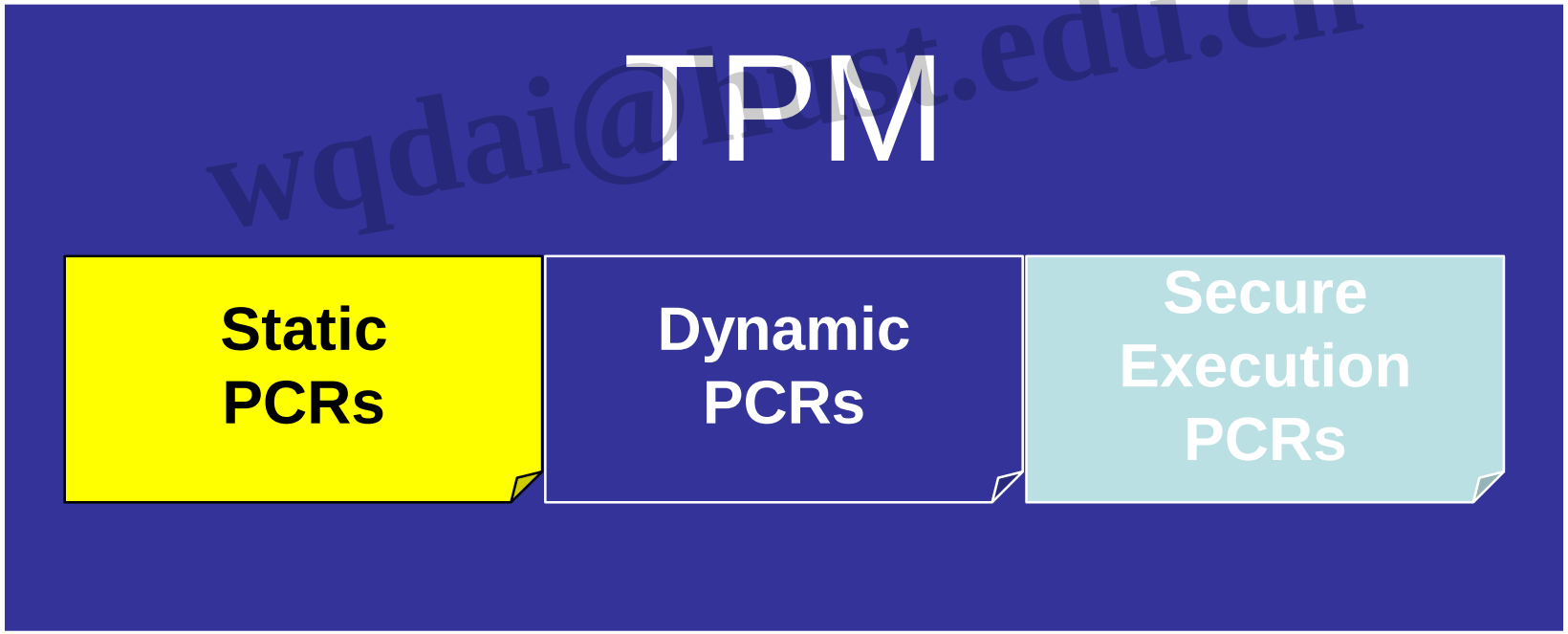
Concurrent Flicker Sessions

- Other CPUs continue to perform useful work
- Memory Controllers isolate secure state



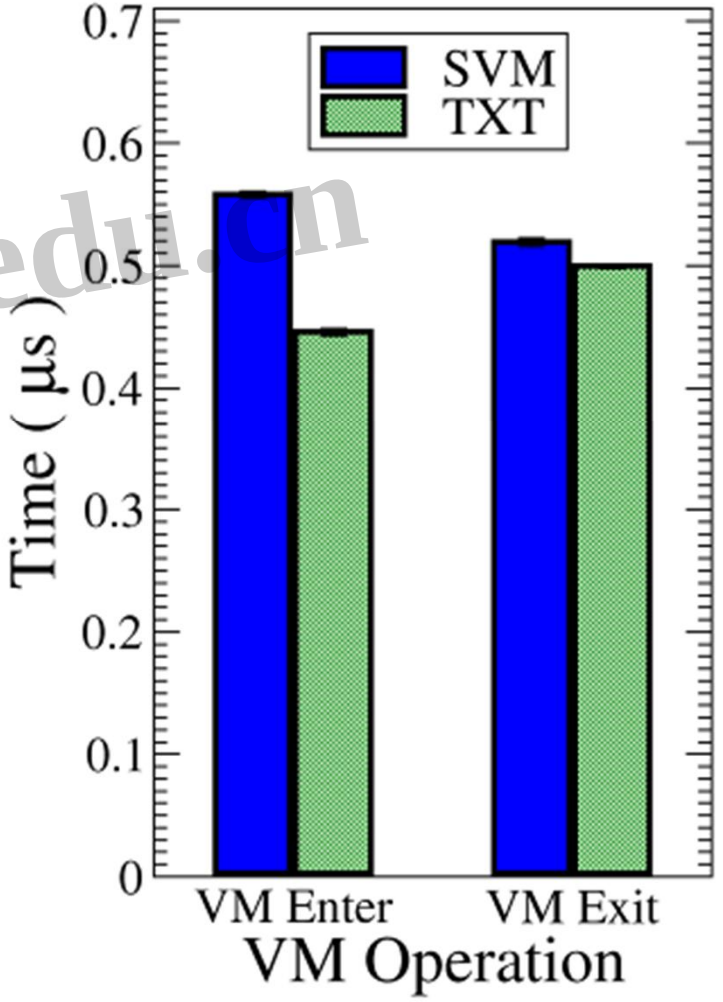
Add *Secure Execution* PCRs to TPM

- Each SE PCR holds state for one Flicker session
- Bounds number of concurrent Flicker sessions



Anticipated Improvement

- Late launch cost only incurred when Flicker session launches
- TPM (Un)Seal only used for long-term storage
- Multicore systems remain interactive
- Context switch overheads (common case) resemble VM switches today



Ongoing Work

- Containing malicious or malfunctioning security-sensitive code
- Creating a trusted path to the user
- Porting implementation to Intel
- Improving automatic privilege separation

Related Work

- Secure coprocessors
  - Dyad [Yee 1994], IBM 4758 [JiSmiMi 2001]
- System-wide attestation
  - Secure Boot [ArFaSm 1997], IMA [SaZhJaDo 2004], Enforcer [MaSmWiStBa 2004]
- VMM-based isolation
  - BIND [Shi, Perrig, van Doorn 2005]
  - AppCores [Singaravelu et al. 2006]
  - Trustworthy Kiosks [Garriss et al. 2006]
  - Proxos [Ta-Min, Litty, Lie 2006]
  - Overshadow [Chen et al. 2008]
- “Traditional” uses of late launch
  - Trustworthy Kiosks [GaCáBeSaDoZh 2006], OSLO [Kauer 2007],

Conclusions

- Explore how far an application's TCB can be minimized
- Isolate security-sensitive code execution
- Provide fine-grained attestations
- Allow application writers to focus on the security of their own code
- TCB for applications can be greatly reduced
- Recommendations to enhance performance
  - State protection during context switch
  - Flicker sessions on multiple CPUs
  - TPM support for concurrent Flicker sessions