2023-2024学年春季学期

计算机体系结构安全

Computer Architecture Security

授课团队：史岗，陈李维

计算机体系结构安全

Computer Architecture Security

[第8次课] 安全体系结构实践

授课教师：史岗

授课时间：2024.04.15

内容概要

- SP & BP体系结构 (Princeton)
- CHERI体系结构 (Cambridge)
- Security-First 体系结构 (IIE-CAS)
- 总结

内容概要

- SP & BP体系结构 (Princeton)
- CHERI体系结构 (Cambridge)
- Security-First 体系结构 (IIE-CAS)
- 总结

SP & BP Secure Architecture (Princeton)

Secret-Protected (SP) Architecture提出的背景（2005年）

- **Distributed software-based key management**
  - Involves multiple servers
- **Secure coprocessors and crypto tokens (deployed)**
  - Tamper-resistant crypto modules (IBM's 4758) and smartcards
- **Trusted Computing Group (TPM recently available)**
  - Industry: Microsoft NGSCB, Intel LaGrande.
- **Recent secure processor proposals (research)**
  - XOM, AEGIS, VSCoP

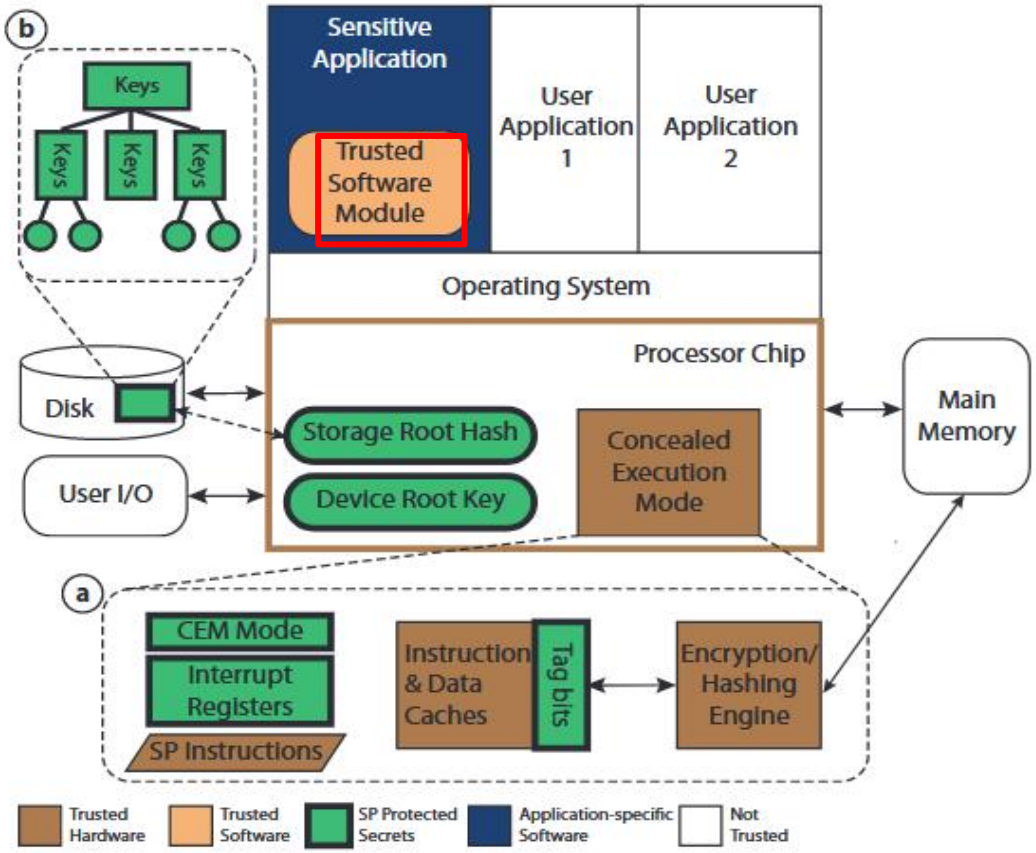
内容引自： "Architecture for Protecting Critical Secrets in Microprocessors"，IEEE/ACM International Symposium on Computer Architecture (ISCA), June 2005.

SP & BP Secure Architecture (Princeton)

- SP威胁模型
  - 软件攻击：
    - 包括恶意软件或可利用的软件漏洞，允许敌手完全控制操作系统以执行软件攻击，访问和修改所有操作系统级抽象，如进程、虚拟内存转换、文件系统、系统调用、内核数据结构、中断行为和I/O。
  - 硬件攻击：
    - 由具有设备物理访问权限的敌手执行，例如直接访问硬盘上的数据、探测物理内存和拦截显示器和I/O总线上的数据。
- SP安全假设
  - 绝大多数软件（包括操作系统）和硬件都是不可信的
  - 通过密码技术构建的执行环境是安全的

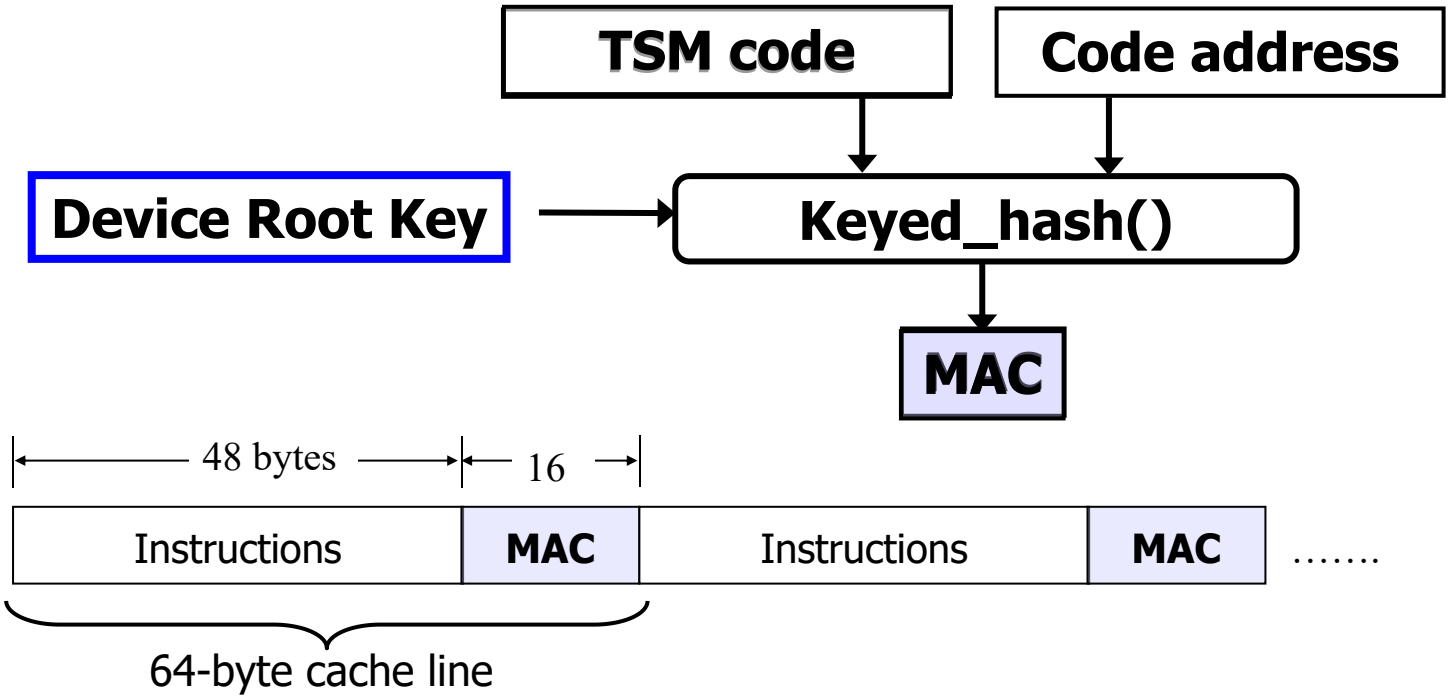
SP & BP Secure Architecture (Princeton)

- 总体架构
  - 隐藏执行模式Concealed Execution Mode (CEM)，硬件提供
  - 可信软件模块Trust Software Module（TSM）执行用户（应用）中的涉及隐私数据的功能，受到CEM的保护



SP & BP Secure Architecture (Princeton)

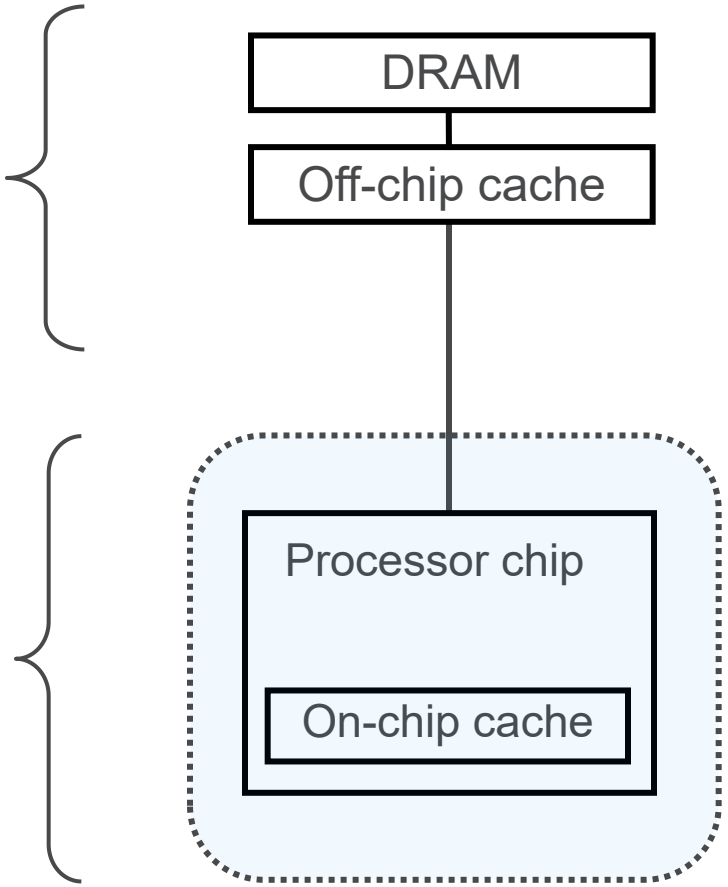
- Concealed execution mode (CEM)
  - 对TSM代码完整性保护



- 用Device Root Key作为密钥
- 加密的粒度为 cache line

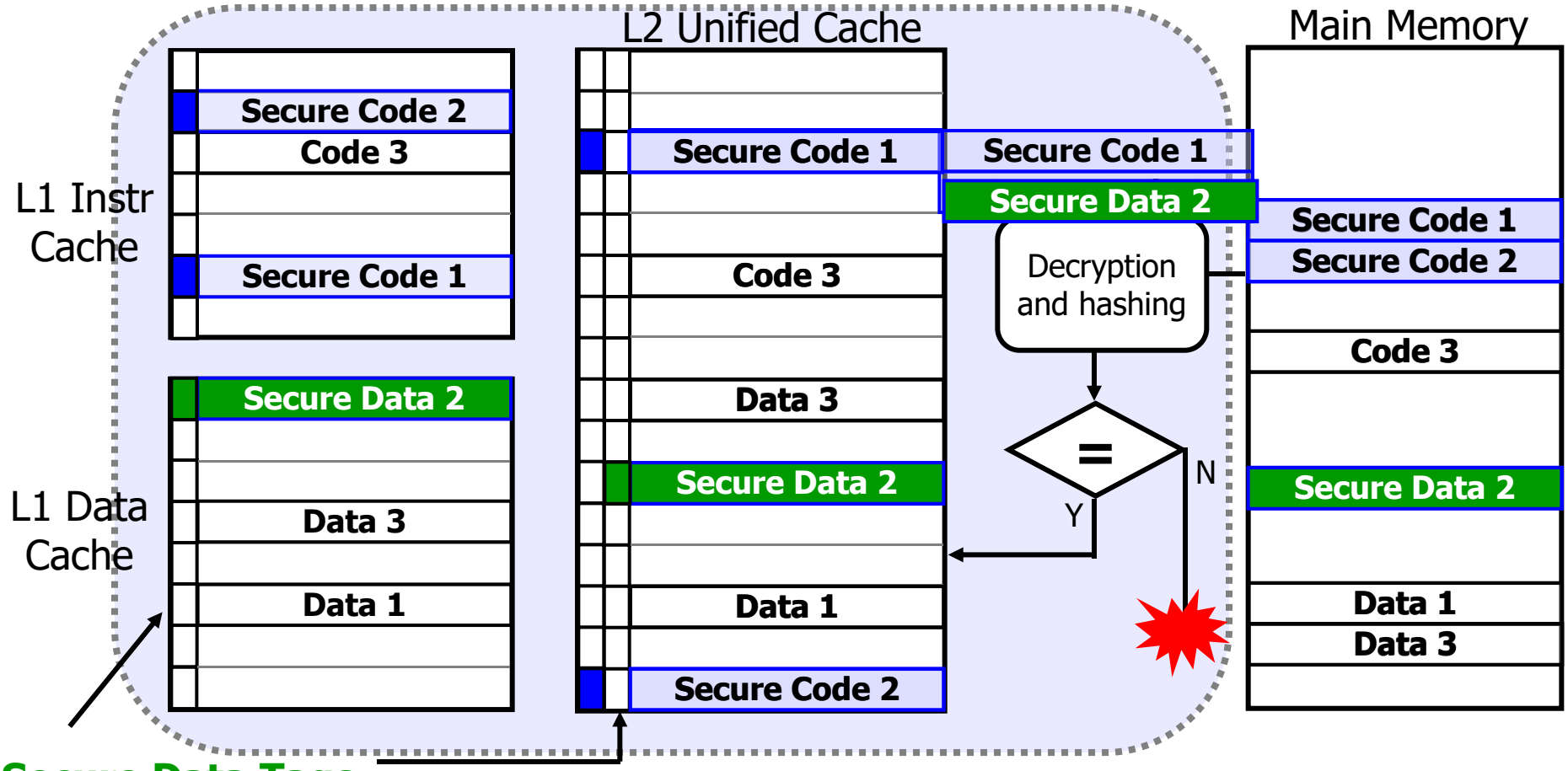
SP & BP Secure Architecture (Princeton)

- Concealed execution mode (CEM)
  - 对TSM数据的保护
    - 数据以密文形式保存在片外
    - 通过加密和Hash进行保护
    - 数据以明文形式保存在片内
    - 对数据加标签用于区分



SP & BP Secure Architecture (Princeton)

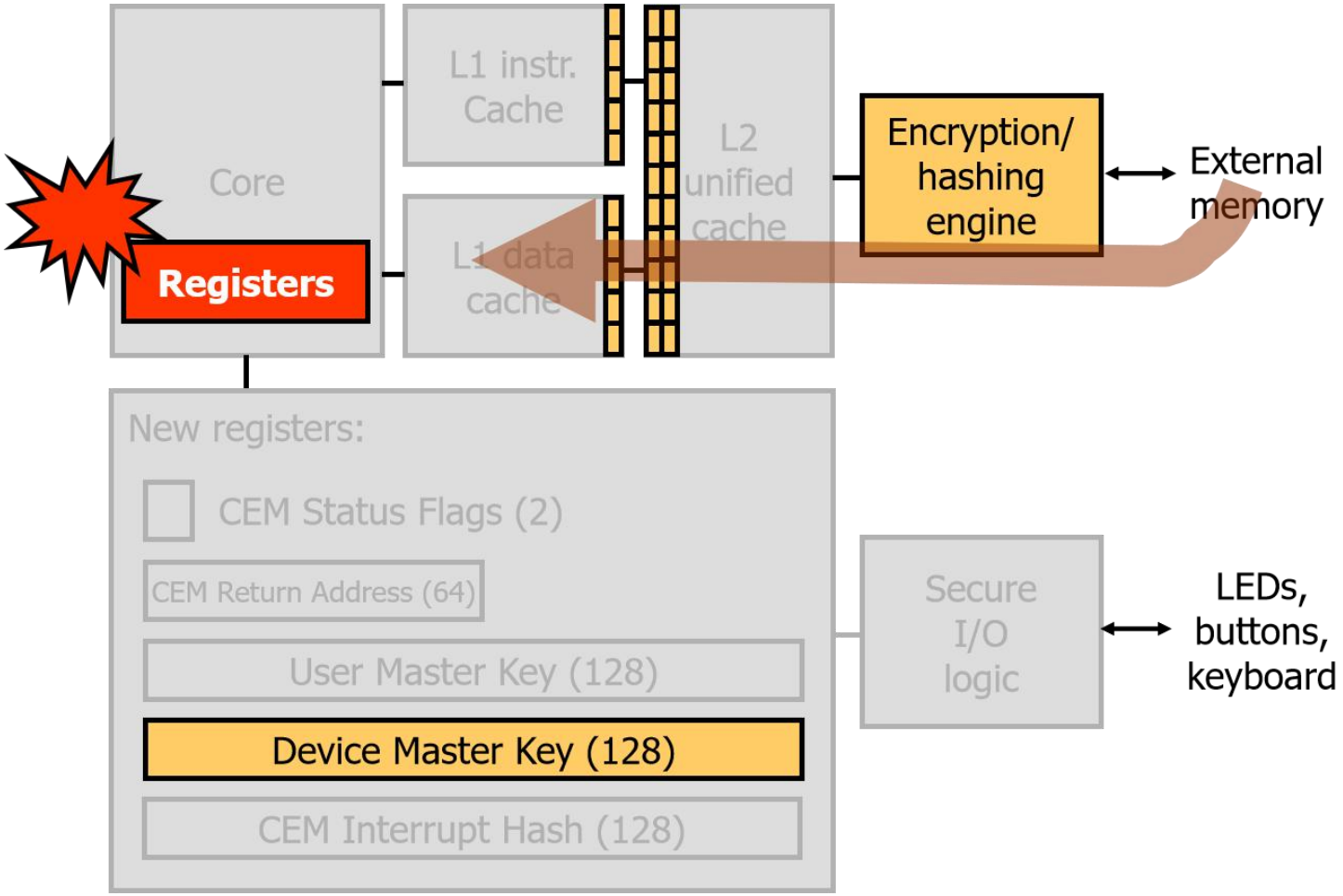
- Concealed execution mode (CEM)
  - 在存储层次中的具体工作原理



内容引自： “Architecture for Protecting Critical Secrets in Microprocessors”, IEEE/ACM International Symposium on Computer Architecture (ISCA), June 2005.

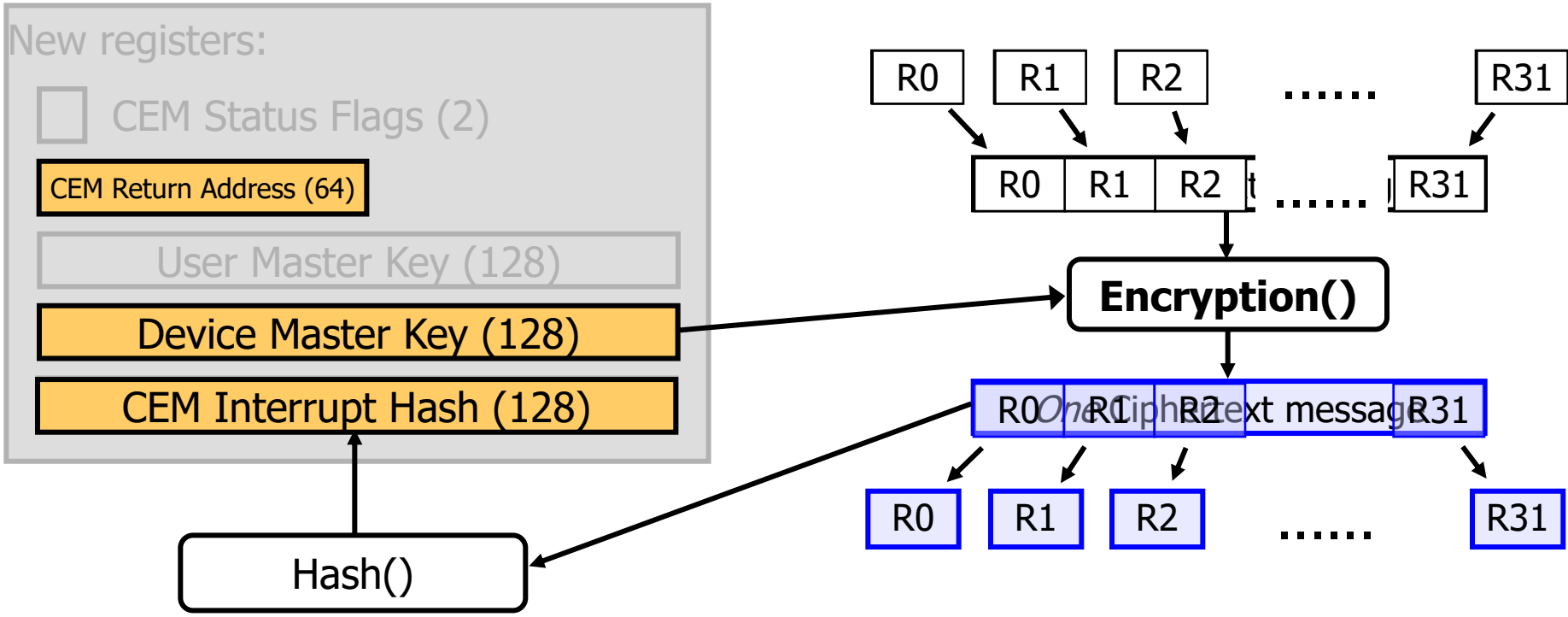
SP & BP Secure Architecture (Princeton)

- Concealed execution mode (CEM)
  - 上述保护所需硬件



SP & BP Secure Architecture (Princeton)

- Concealed execution mode (CEM)
  - 中断发生时的现场保护



- 保护过程无需OS中断处理函数的干预
- Hash值和返回地址都存储在片上

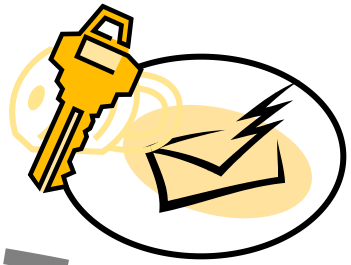
SP & BP Secure Architecture (Princeton)

- SP Architecture 小结

Secure I/O

User Master Key

protects



Operates upon

Trusted software module

Code
Memory
Registers

Execution environment on device

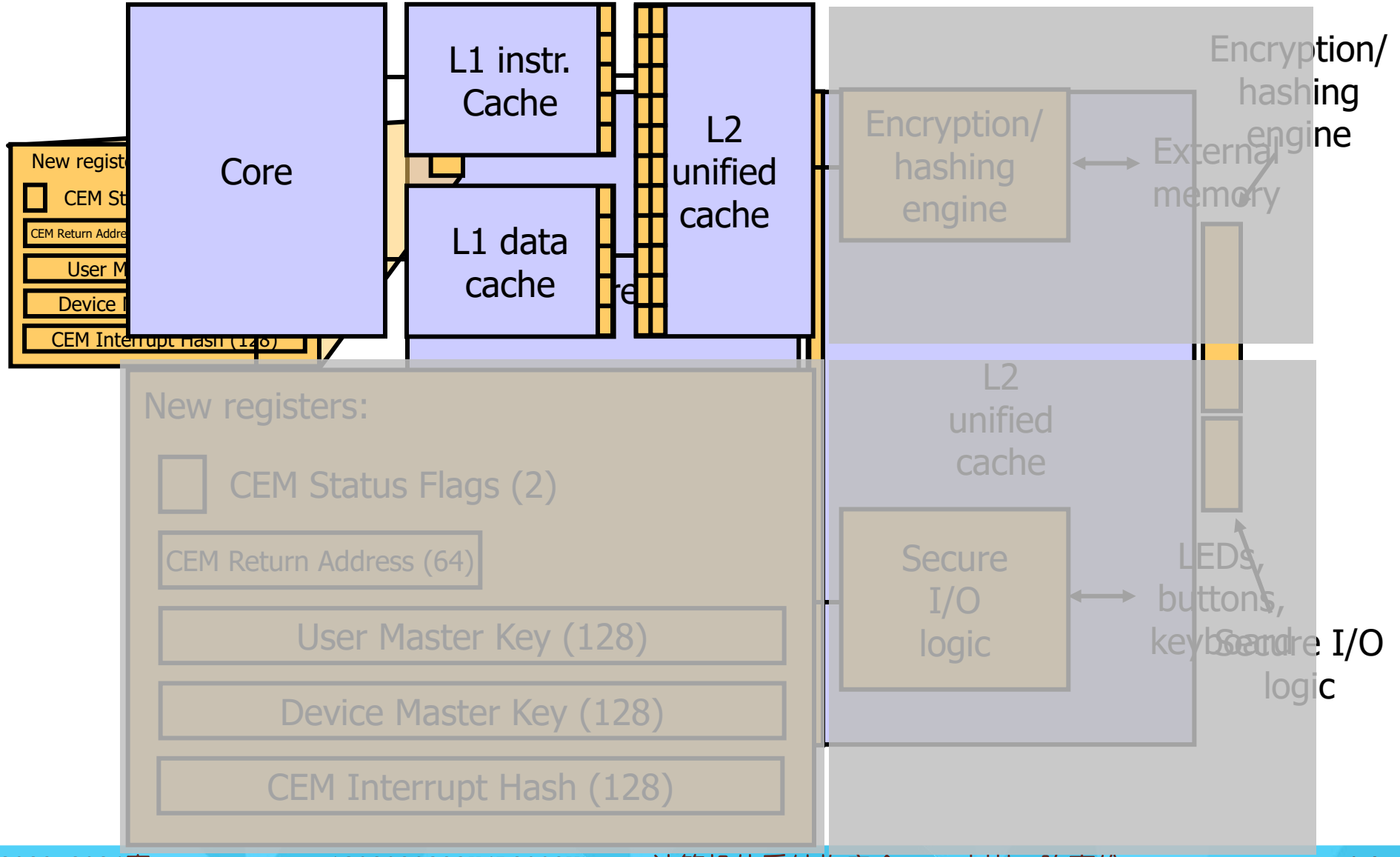
protects

Device initialization

Device Master Key

SP & BP Secure Architecture (Princeton)

- 以最小的硬件开销保护了用户的隐私数据



Bastion Security Architecture（2010年）

威胁模型

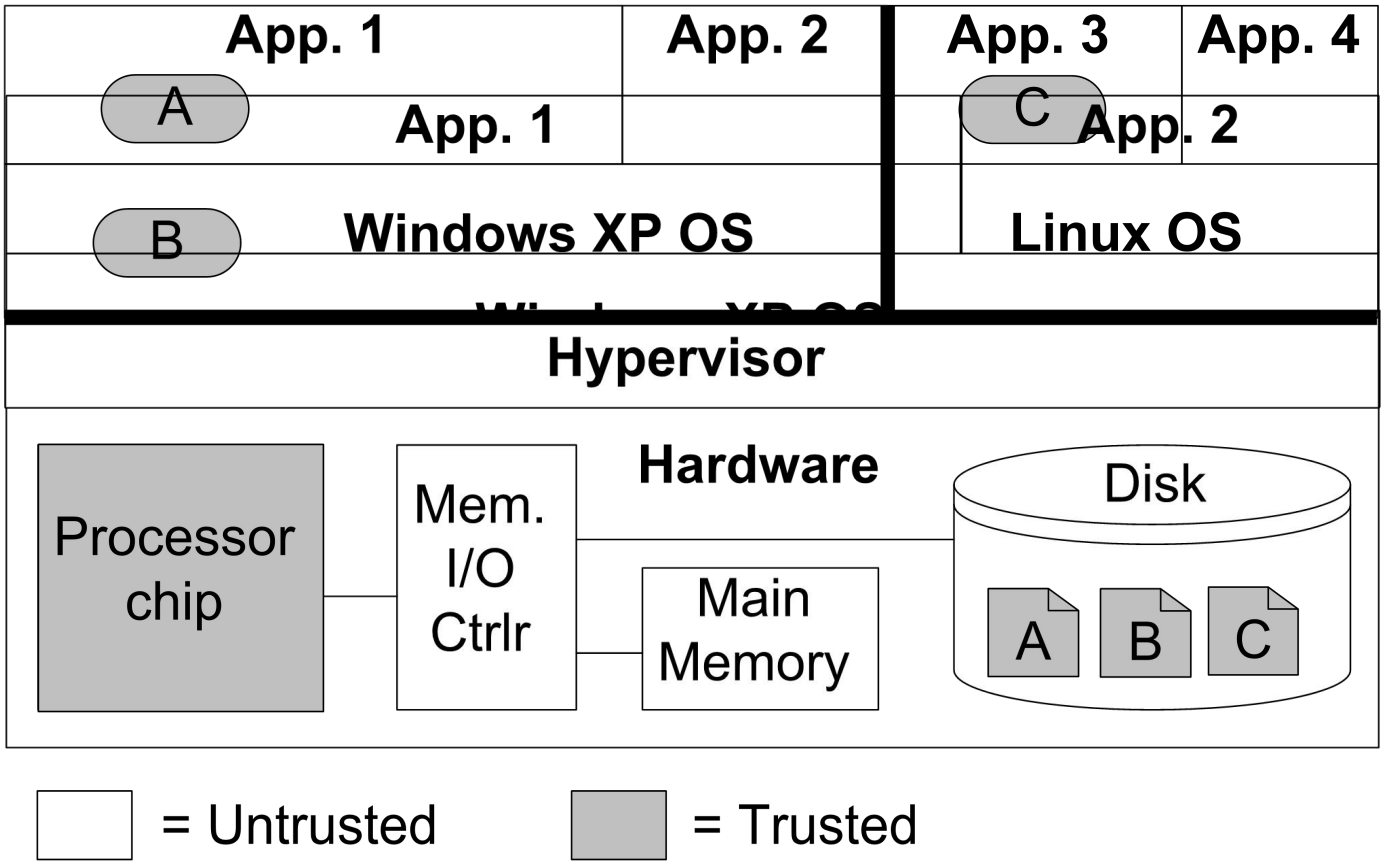
- 软件攻击：
  - 内部攻击：由软件实体自身执行的，导致目标软件实体的状态被破坏或泄漏。是由目标实体代码的漏洞导致的，而不是由外部实体直接读取或覆盖目标实体的状态。
  - 外部攻击：是由本地软件执行的，直接读取或覆盖目标软件实体的状态，例如恶意操作系统窥视应用程序的数据空间。
- 硬件攻击：
  - 由构成正常计算平台的硬件组件实施，如内存芯片、磁盘、I/O控制器等；也可以由专门为实施攻击而添加到平台上的新组件实施，如总线探测器等。攻击旨在直接窥视或破坏安全关键软件的状态，还可能试图拦截可信I/O端点之间的安全I/O通信。

安全假设

- CPU芯片是可信的，所有其他硬件（存储芯片、外部总线、硬盘驱动器等）都不可信。
- 虚拟机管理程序和小型可信软件模块是可信的，所有其他软件（操作系统和应用程序）都是不可信的。

SP & BP Secure Architecture (Princeton)

总体架构



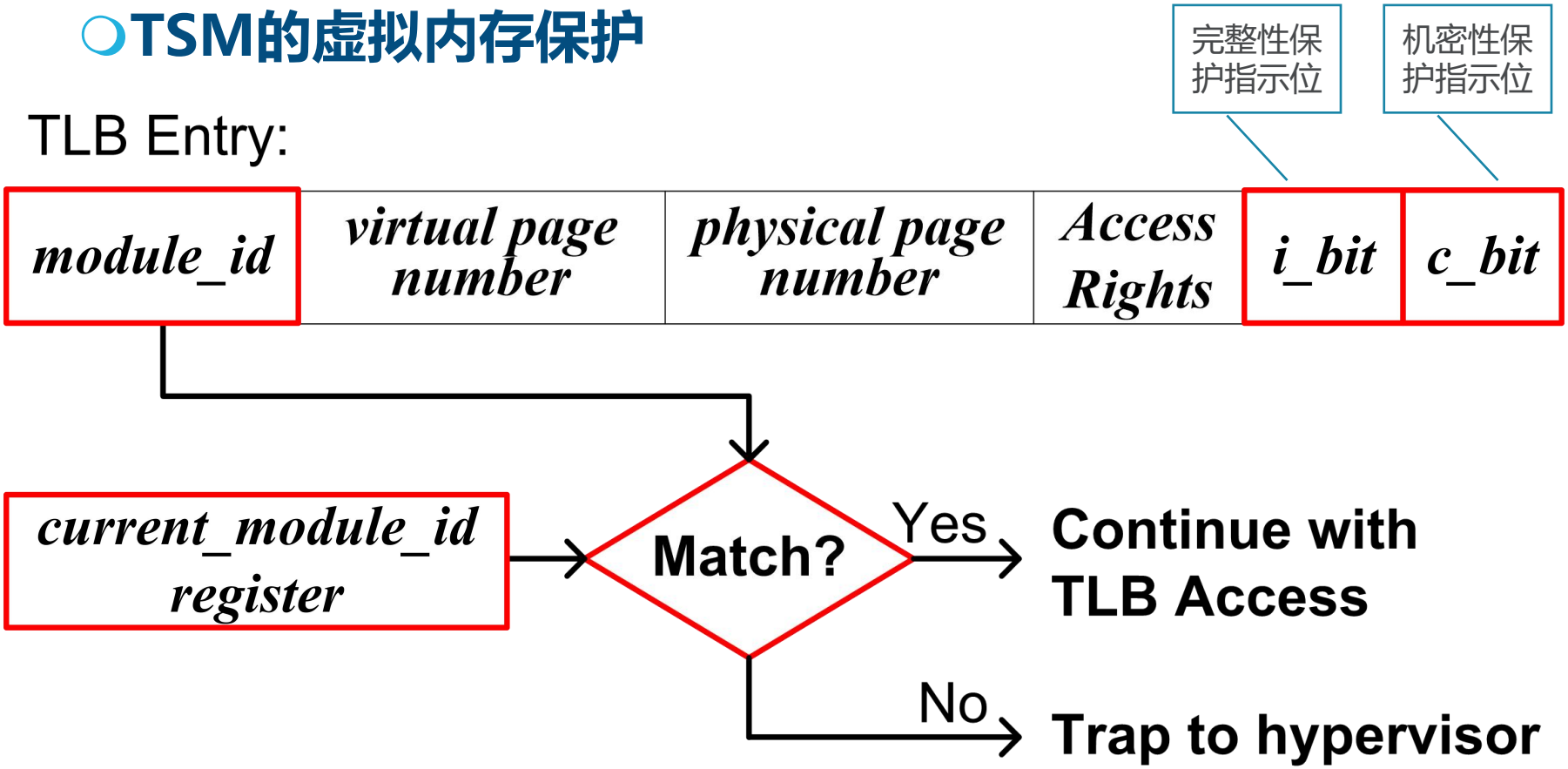
引自：D. Champagne, R.B. Lee, "Scalable Architectural Support for Trusted Software", IEEE Intl. Symp. on High-Performance Computer Architecture (HPCA), Jan. 2010

SP & BP Secure Architecture (Princeton)

- 总体架构
  - 安全启动（Secure Launch）
    - Bastion处理器的复位向量指向不受信任的BIOS代码，就像传统处理器一样。该代码建立一个基本的执行环境，然后将CPU控制权移交给一个不受信任的Hypervisor加载程序。
    - 加载程序从持久存储（例如磁盘）中获取Hypervisor二进制映像，将其加载到内存中，然后跳转到Hypervisor的初始化例程。
    - 安全Hypervisor必须调用一个**新的secure_launch指令**，以启动Bastion处理器对Hypervisor内存进行运行时保护，并激活hypervisor的安全存储功能
    - secure_launch指令调用处理程序（on-chip），它会计算Hypervisor的哈希值，如果不一致，其安全存储区就锁定；还会生成一个新的密钥，用于数据在从芯片缓存存回主存储器时进行加密和哈希

总体架构

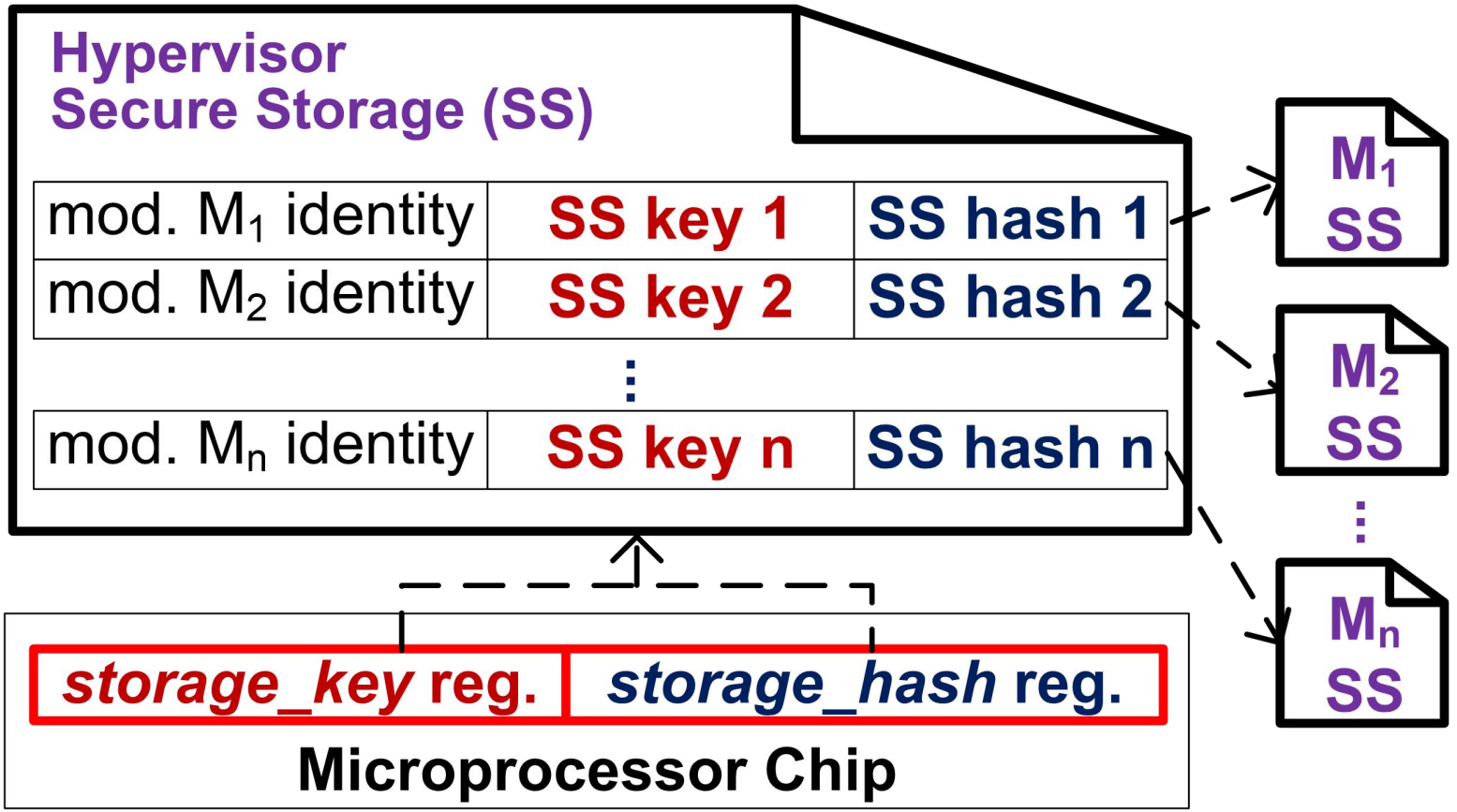
TSM的虚拟内存保护



- 映射关系在SECURE_LAUNCH hypercall调用时由Hypervisor建立
- Hypervisor处理TLB miss
- module_id zero 代表不可信软件

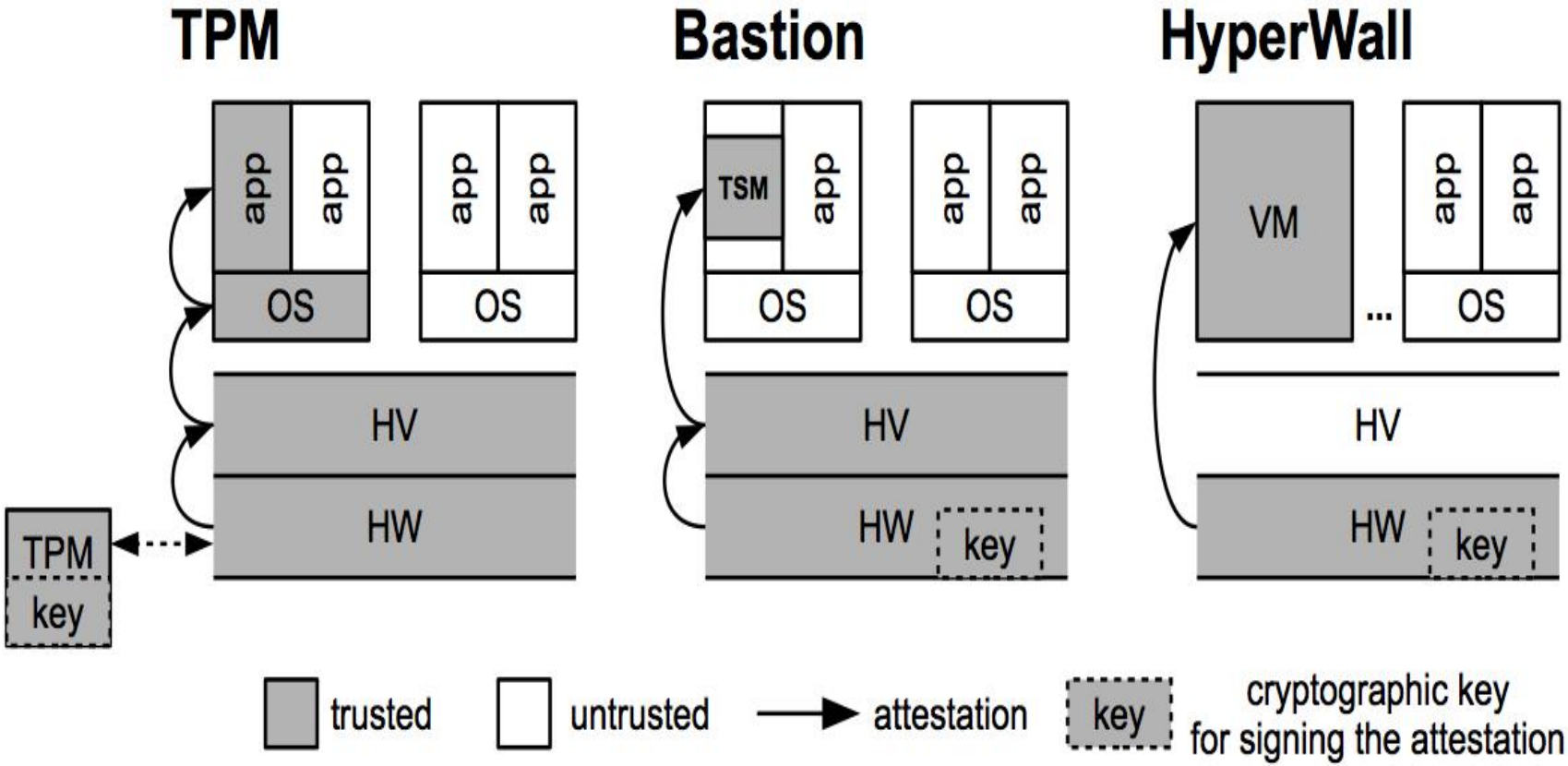
SP & BP Secure Architecture (Princeton)

- 总体架构
  - 数据安全存储



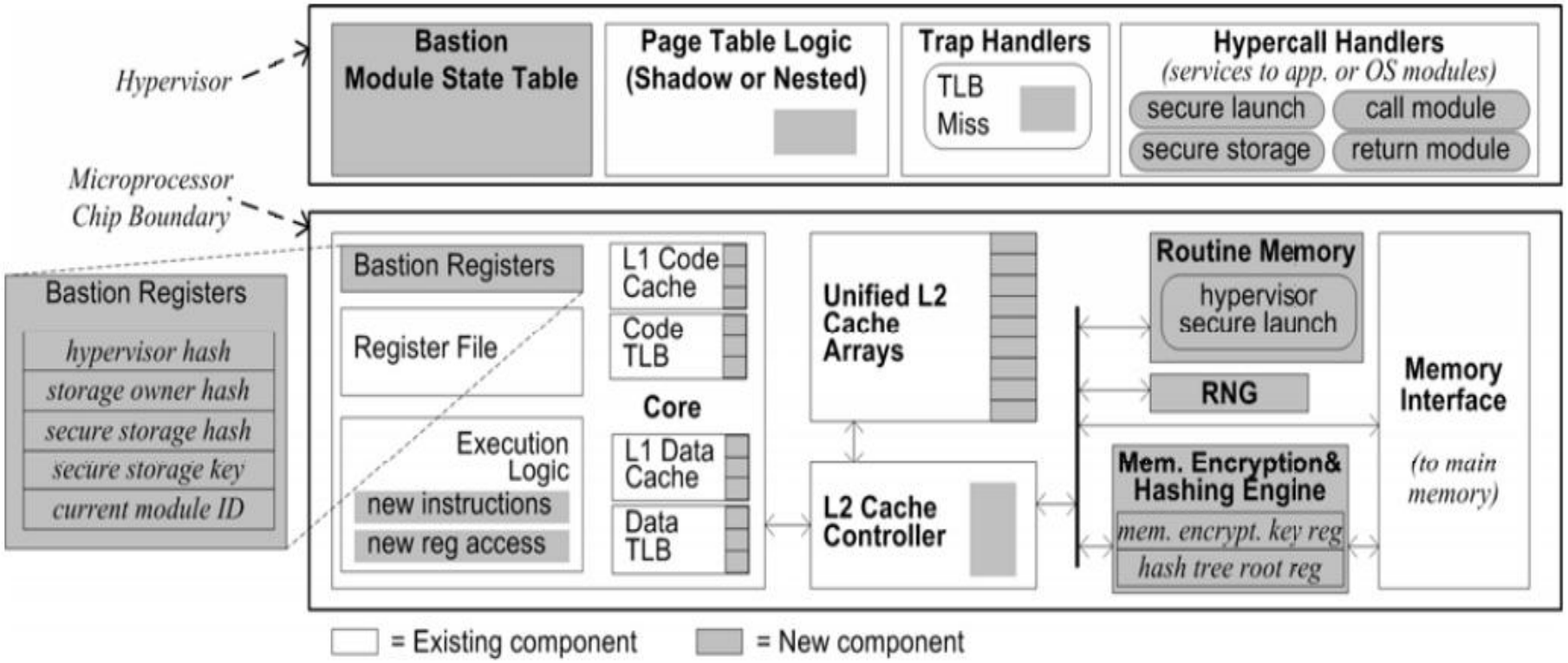
SP & BP Secure Architecture (Princeton)

- 总体架构
  - 跨层完整性度量



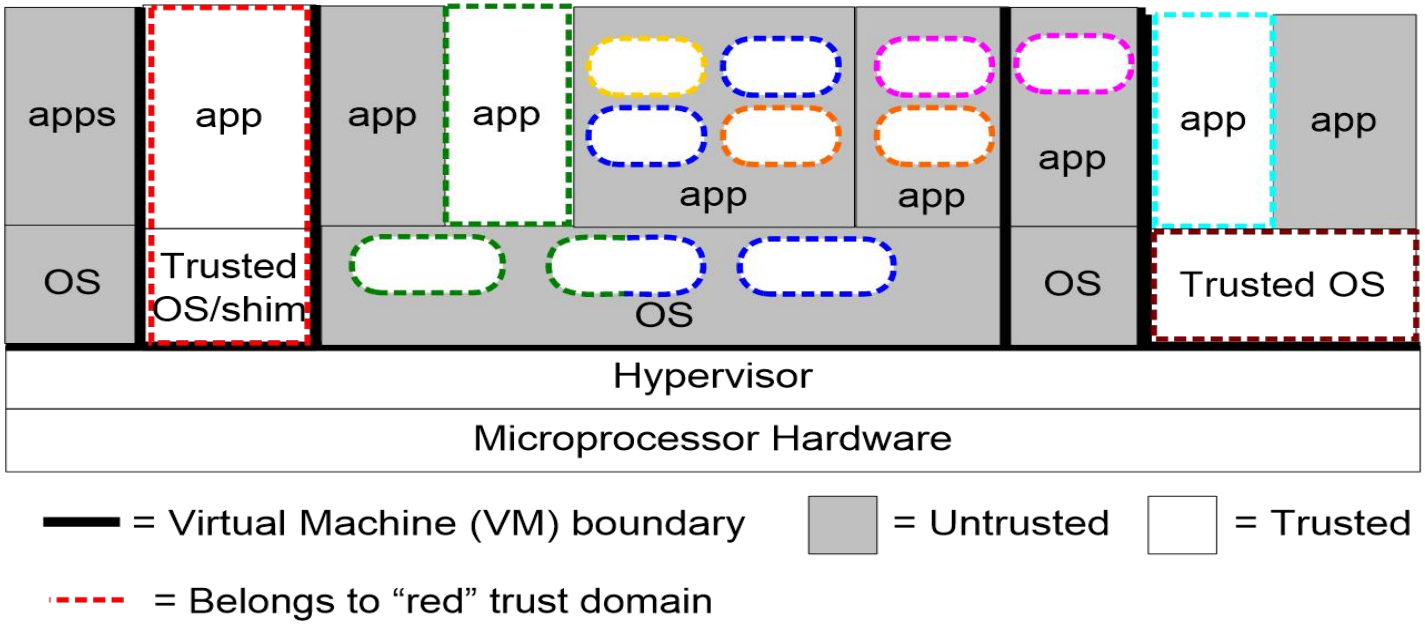
SP & BP Secure Architecture (Princeton)

- 对现有软硬件的修改



SP & BP Secure Architecture (Princeton)

- BP相比与SP的进步
  - 通过对Hypervisor的保护，从而可以同时支持多个TSM的运行
  - 尝试将TPM的功能放到CPU里面，是否合适由历史评判
  - 实现了对安全关键任务的灵活执行



内容概要

- SP & BP体系结构 (Princeton)
- CHERI体系结构 (Cambridge)
- Security-First 体系结构 (IIE-CAS)
- 总结

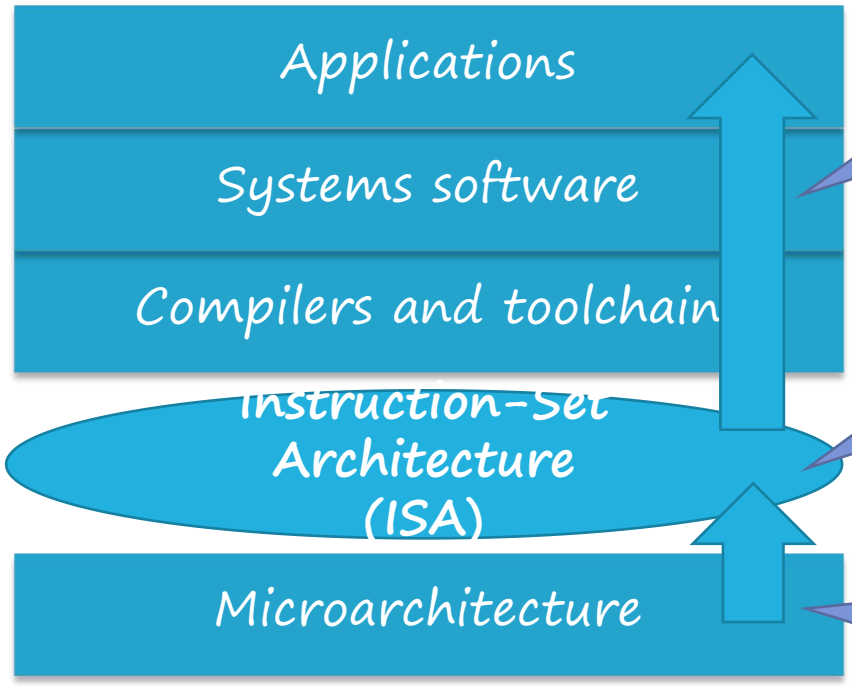
CHERI Secure Architecture （Cambridge）

- Capability Hardware Enhanced RISC Instructions (CHERI) (2014年)
  - CHERI（能力硬件增强 RISC 指令）是剑桥大学推出的旨在**重新审视硬件和软件的基本设计选择**，以显著提高系统安全性的项目。
  - CHERI **扩展了传统的硬件指令集架构 (ISA)**，以实现细粒度的内存保护。 CHERI 内存保护功能允许对内存不安全的编程语言（例如 C 和 C++）进行调整，以针对当前广泛的漏洞，提供强大高效的保护。
  - CHERI 可扩展的分区功能支持对操作系统 (OS) 和应用程序代码进行**细粒度分解**，限制安全漏洞的影响。

CHERI Secure Architecture （Cambridge）

- 主要特点
  - 是处理器级的体系结构保护模型
    - 用硬件和软件协同建立能力（Capability）系统
    - 向指令集 (ISA) 添加新的安全原语
    - 通过CPU 和SoC 的微架构扩展来实现
    - 能让软件具有新的安全行为
  - CHERI 缓解 C/C++中的漏洞
    - 包括：Hypervisors、操作系统、语言运行时、浏览器等软件
    - 细粒度内存保护可阻止任意代码执行攻击，阻止常见的漏洞利用工具
    - 可扩展的分区缓解许多类型漏洞利用，甚至是未知类型漏洞

为软件安全提供体系结构的原语



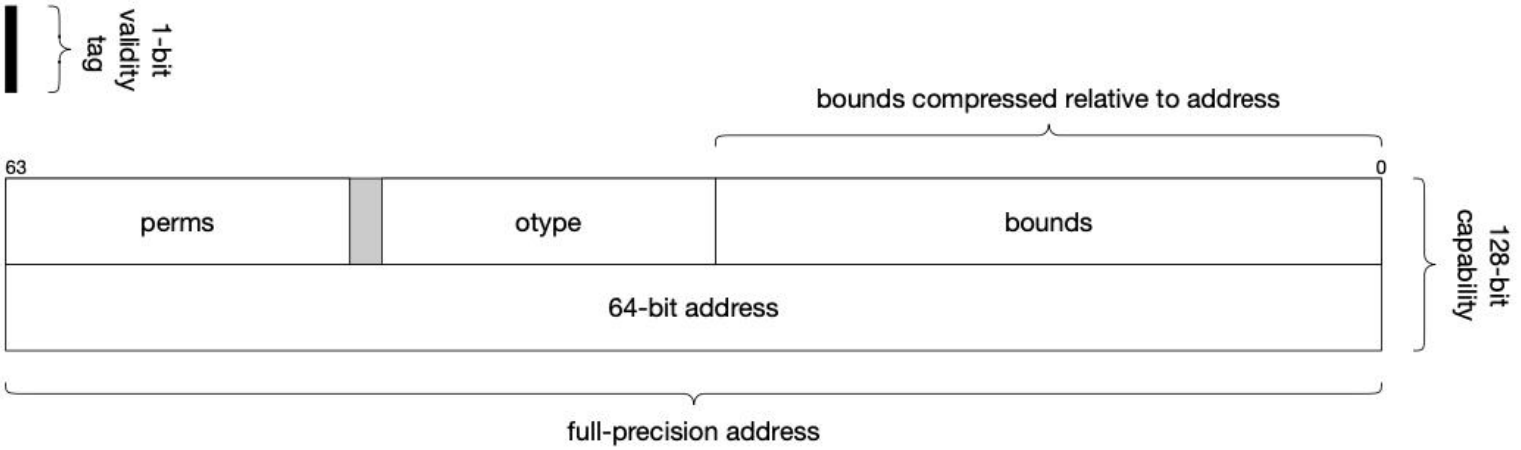
软件基于能力来强化安全属性，包括空间性和时间性内存安全，也可以构造隔离分区

编译器、系统软件和应用程序使用它来约束程序的执行

微架构实现了能力数据类型和标记内存，确保能力在操作和使用上被强制执行，例如能力界限、单调性和来源有效性

能力（Capability）是什么?

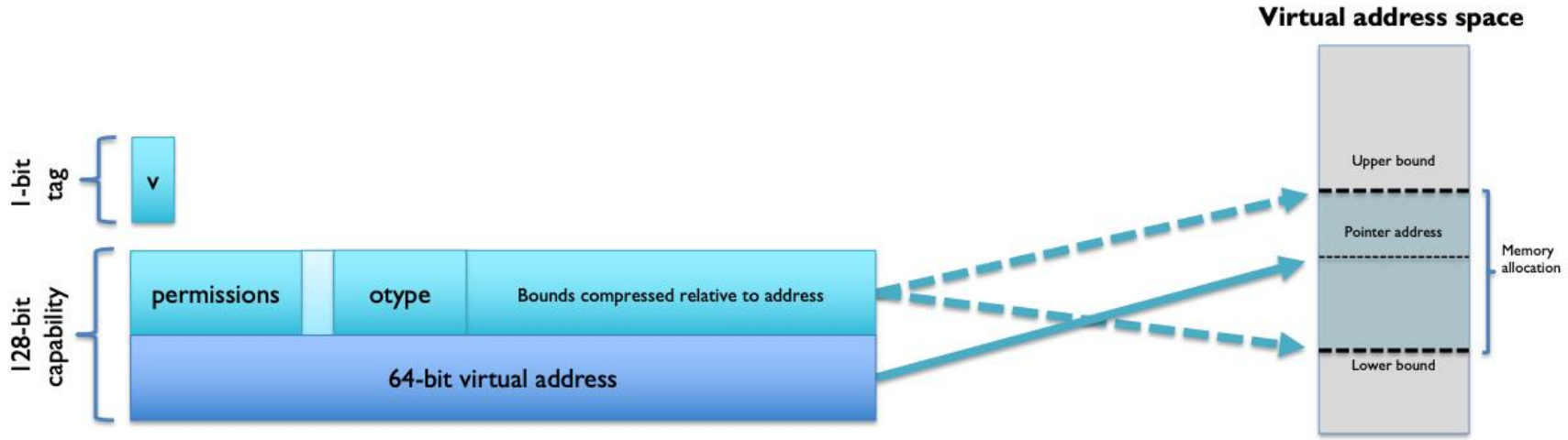
- 是一种新的体系结构数据类型，就像整数、浮点数、向量一样。它的宽度是整数指针类型的两倍：64-128、32- 64
- 用途是保护指向代码或数据的指针数据（虚拟地址），所有内存访问，包括Load/Store、取指等，都必须经过能力授权（检查）
- 或者称为：权能数据、扩展指针数据、胖指针.....



能力数据类型的组成

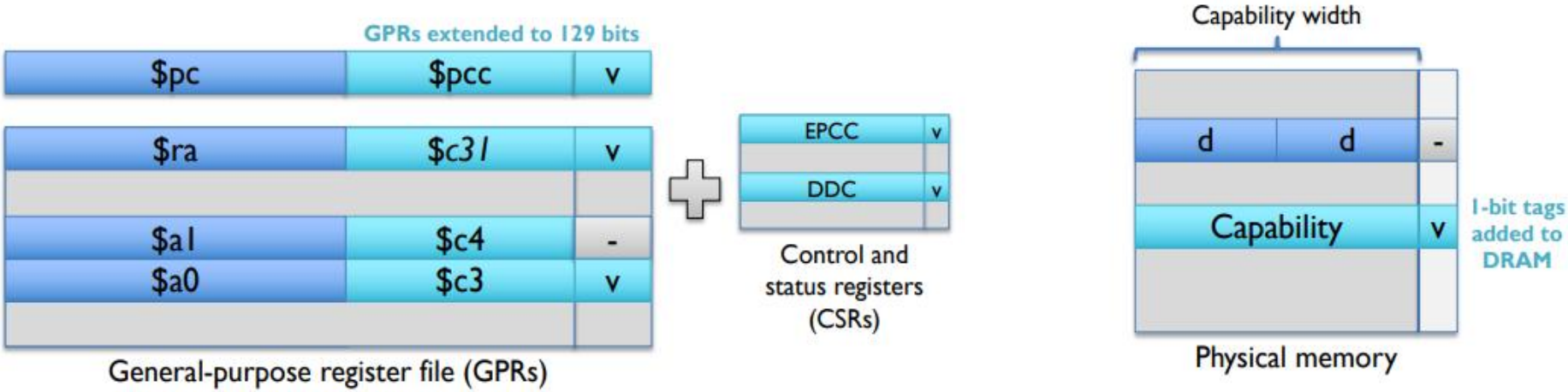
元数据

- 标签：用于跟踪能力的有效性。如果无效，则该能力不能用于Load、Store、取指等操作，并发生一个CPU异常。Tag只能清除，不能再次设置。
- 边界： 下界和上界描述了能力授权/检查的地址空间范围，超出范围会发生一个CPU异常。范围可以缩小，但不能增大。
- 权限： 权限掩码控制着能力的使用方式，对Load、Store和取指进行授权。最初设置后，之后权限只能清除，不能再打开。
- 对象类型： 如果该值不等于-1，则该能力被 "密封"。密封能力可用于实现不透明指针类型（只能通过固定的函数或接口调用该指针）。



CHERI Secure Architecture (Cambridge)

能力寄存器



- **64-bit general-purpose registers (GPRs)** are extended with **64 bits of metadata** and a **1-bit validity tag**
- **Program counter (PC)** is extended to be the **program-counter capability ($PCC)**
- **Default data capability ($DDC)** constrains legacy integer-relative ISA load and store instructions
- **Tagged memory** protects capability-sized and -aligned words in DRAM by adding a **1-bit validity tag**
- **Various system mechanisms** are extended (e.g., capability-instruction enable control register, new TLB/PTE permission bits, exception code extensions, saved exception stack pointers and vectors become capabilities, etc.)

引自： "CHERI Capability Hardware Enhanced RISC Instructions",University of Cambridge and SRI International Web Slide Deck – 16 January 2024

CHERI Secure Architecture (Cambridge)

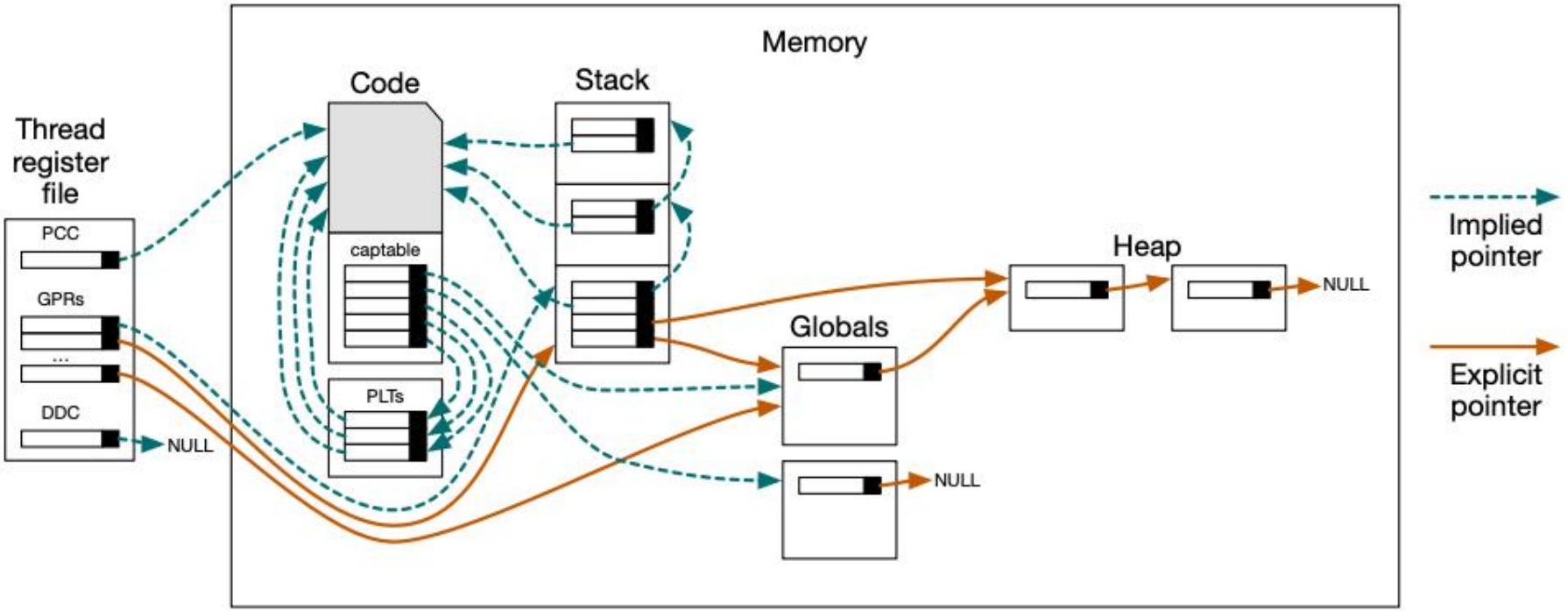
- 能力指令集扩展
  - 检索能力字段：检索各种能力字段的值，包括Tag、地址、权限和对象类型。
  - 操作能力字段：根据单调性设置或修改各种字段，包括地址、权限和对象类型。
  - 通过能力加载或存储：通过适当授权的能力，加载数据、能力或其他值。
  - 控制流：执行跳转或跳转并链接寄存器到能力目标。
  - 特殊功能寄存器：读取和设置特殊功能寄存器的值，例如在异常处理中异常程序计数器功能（EPCC）的值。
  - 分区：相关指令支持快速保护域转换。

Instruction	Description	Priv.	Soft.
CGetBase	Get capability base		
CGetOffset	Get capability offset		
CGetLen	Get capability length		
CGetTag	Get capability tag		
CGetPerm	Get capability permissions		
CToPtr	Convert capability to pointer		
CPtrCmp	Compare two capabilities		
CIncBase	Increment capability base		
CSetLen	Set capability length		
CClearTag	Clear capability tag		
CSetOffset	Set capability offset		
CFromPtr	Convert pointer to capability		
CSC	Store capability via capability		
CLC	Load capability via capability		
CL[BHWD][U]	Load data via capability		
CS[BHWD]	Store data via capability		
CLL[WD]	Load linked data via capability		
CSC[WD]	Store conditional data via capability		
CGetPCC	Get program-counter capability		
CBTU	Branch if capability tag unset		
CBTS	Branch if capability tag set		
CJR	Capability jump		
CJALR	Capability jump and link		
CGetCause	Get capability cause register	P	
CSetCause	Set capability cause register	P	
CGetSealed	**Get capability sealed bit**		
CGetType	**Get capability type**		
CSeal	**Seal capability**		
CUnseal	**Unseal capability**		
CCheckPerm	**Check capability permissions**		
CCheckType	**Check capability type**		
CCall	**Invoke object capability**		S
CReturn	**Return from object capability**		S

P: Privileged instruction available only to the supervisor.
S: Implemented in part or fully via an exception to the supervisor.

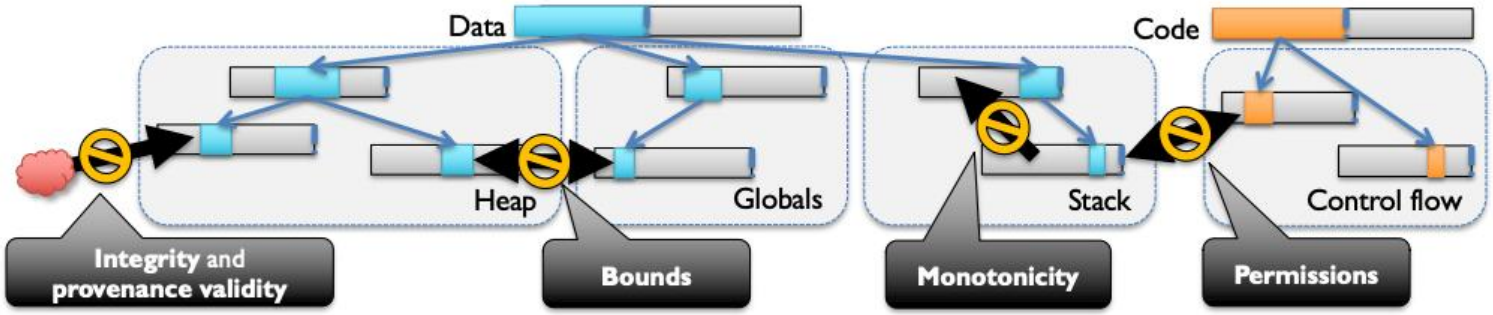
CHERI 细粒度内存保护

- 在整个地址空间中，用能力代替整数地址
- 通过软件（包括内核、运行时链接器、内存分配器和编译器生成的代码）最小化边界和权限
- 硬件只允许通过授予的功能进行取指、加载和存储
- 标签确保所有指针的完整性和出处有效性



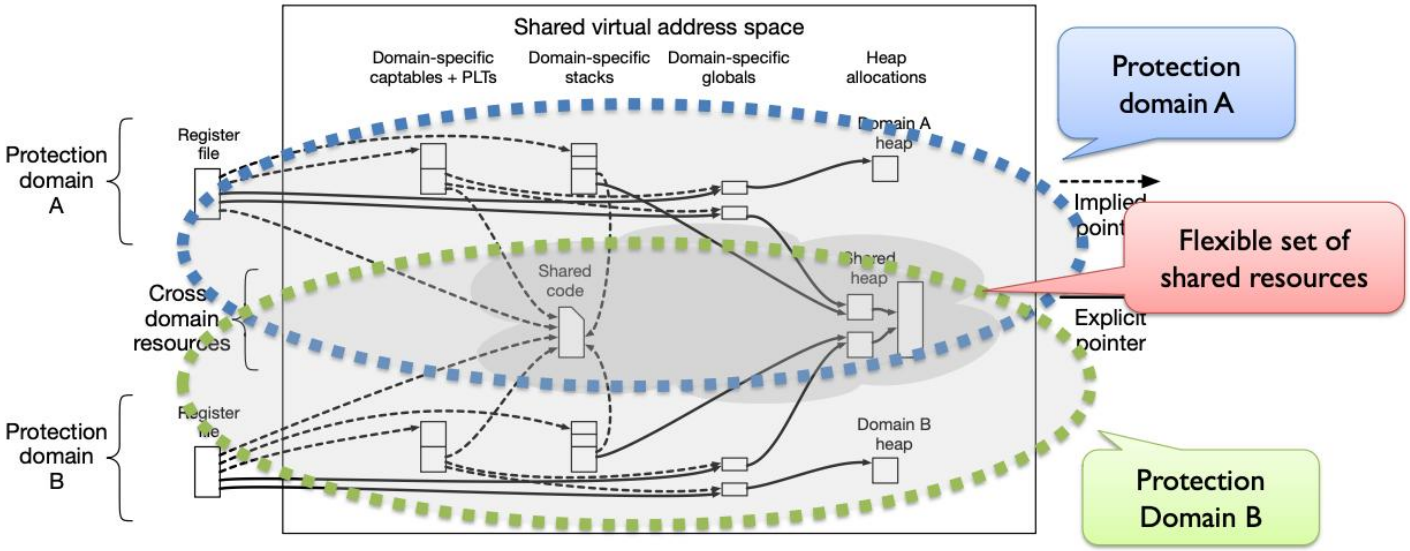
CHERI 细粒度内存保护（指针举例）

- 完整性和出处有效性确保有效指针通过有效转换从其他有效指针衍生出来，无效指针不能使用
- 边界可防止指针被用于访问错误对象
- 单调性防止指针权限升级--例如，扩大边界
- 权限限制指针的意外使用--例如，指针的 W^X



基于CHERI 构造可扩展的软件分区

- 传统的基于 MMU 的软件分区技术将大型软件应用程序分解为多个组件，这些组件在独立进程中运行，仅通过使用进程间通信（IPC）实现的受控通信进行连接。然而，基于 MMU 的分区设计由于利用了多个地址空间和页面粒度共享，因此在可扩展性方面受到很大限制：随着分区数量及其通信的增加，性能、安全性会受到显著影响。
- CHERI 能力允许在地址空间内构建隔离和受控共享，从而提供更高的分区可扩展性。正在执行的分区不能访问其他分区的资源，也不能访问更广泛的系统内存。边界和权限检查可确保分配给分区的能力只允许访问内部资源；单调性可确保这些权限不会被修改以包括其他资源。



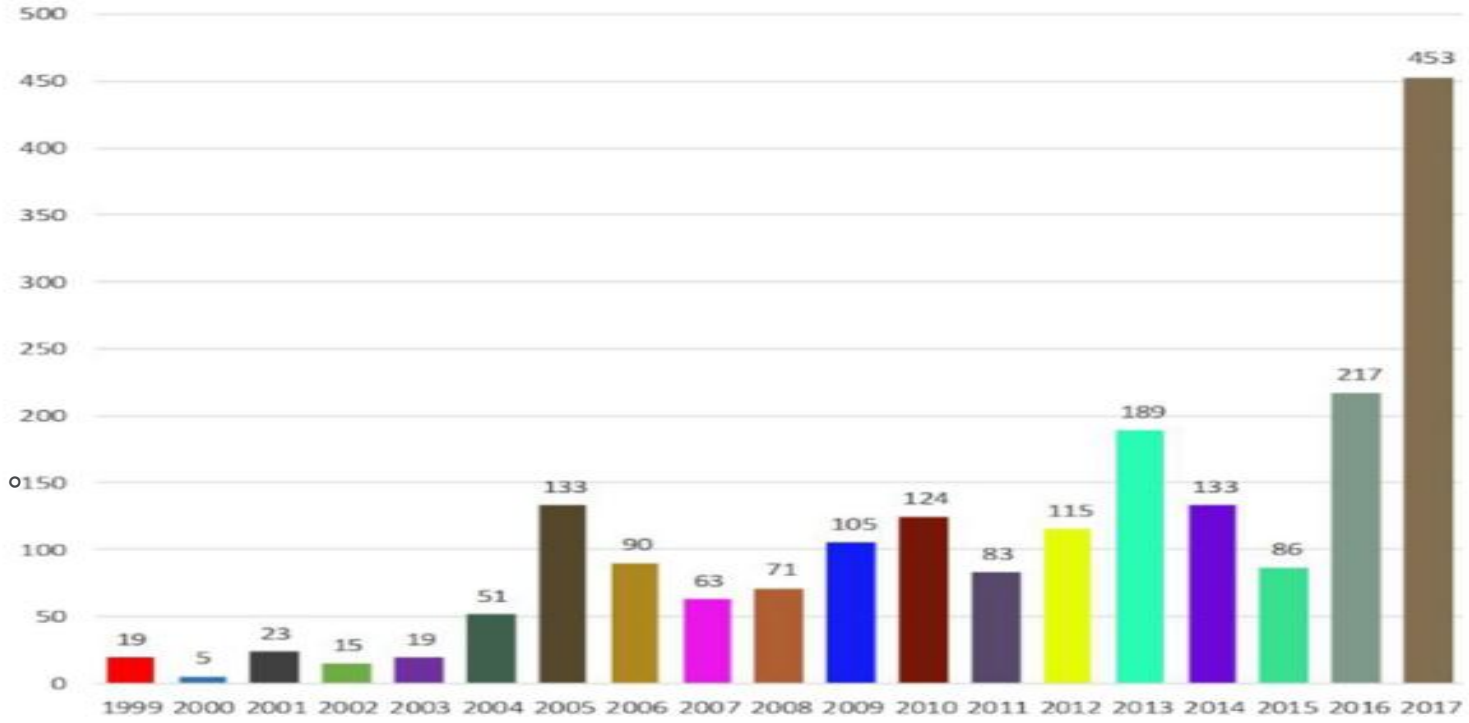
CHERI Secure Architecture （Cambridge）

- CHERI扩展了 Clang、LLVM 和 LLD 编译器和工具链，针对混合能力和纯能力编译模式，进行了一系列更改：
  - 在 Clang 代码生成中使用 LLVM 中间表示法 (IR)，更好地区分整数和指针类型。
  - 调整编译器，使其不再对指针做出各种假设，如确定指针大小和寻址范围。
  - 指示编译器具备处理各种类型指针的能力。这包括堆栈指针和返回地址，也包括 C++ 特有的构造，如指向成员的指针、指向 vtables 的指针以及这些 vtables 中的条目。
  - 在编译过程中的不同位置引入绑定设置，尤其是与栈分配和子对象相关的位置。
  - 添加新限定符形式的次要 C 语言扩展，以及允许查询和修改能力属性（例如，限制边界和权限）的新内置程序。
  - 在后端引入新的代码生成，以实现能力类型和新的 ABI。
  - 更新优化传递以了解能力--尤其是在后期阶段，现有的 DAG 模式识别可能会被打乱，或者优化可能会导致标签被不适当地保留或丢失。
  - 增加编译器诊断功能，以检测与混合能力或纯能力 C/C++ 不兼容的习语。

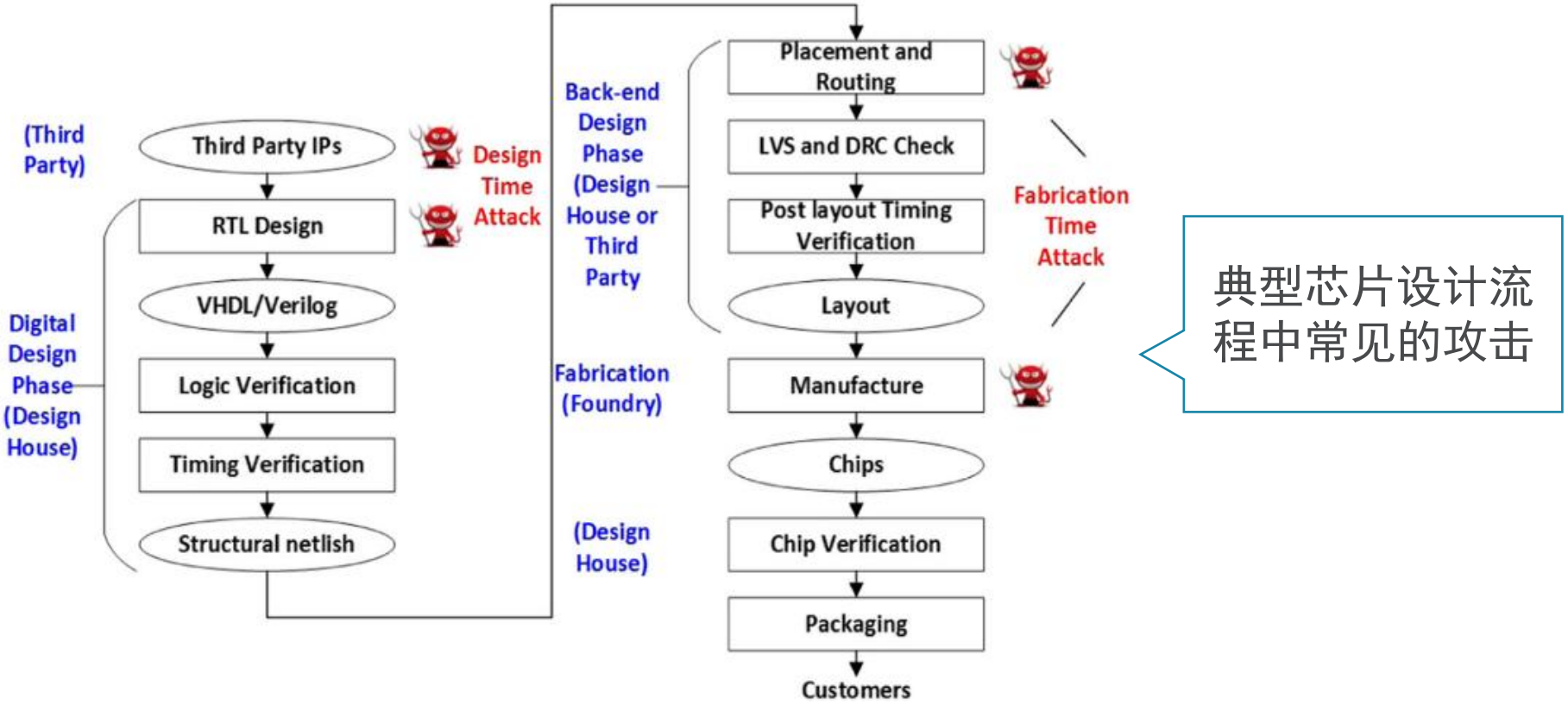
内容概要

- SP & BP体系结构 (Princeton)
- CHERI体系结构 (Cambridge)
- **Security-First 体系结构 (IIE-CAS)**
- 总结

- 软件漏洞不可避免
  - 软件的复杂性已经超出了现有验证方法的能力
    - 例子：火狐浏览器包含6313名贡献者、3600万行代码



- 硬件组件不安全
  - 性能为中心的架构设计导致安全风险存在
    - 例如：分支预测、高速缓存、指令预取等优化与安全伴生
  - 现代处理器复杂冗长的设计流程导致安全无法保证



- 攻击越来越复杂
  - 攻击范围的宽度
    - 攻击可以从任何地方发起，从应用软件到物理硬件所有层次
  - 攻击手段的多样性
    - 蠕虫和僵尸网络：自我复制和传播
    - 测信道攻击：利用硬件漏洞将目标从软件转移到硬件

核心理念

- 安全需要在架构设计阶段就融入，并综合权衡性能、安全和成本。
- 安全所需的控制应具有主动性，并具有最高的权限。
- 物理隔离和硬件隔离比逻辑隔离安全得多，执行环境与安全控制环境应采用强隔离方式。

Security-First体系结构

- 结构上具有计算处理器和**主动安全处理器（ASP），它们有各自的资源，包括内存、磁盘、网络和I/O设备，单向的信息流关系保证了计算与安全的强隔离。**

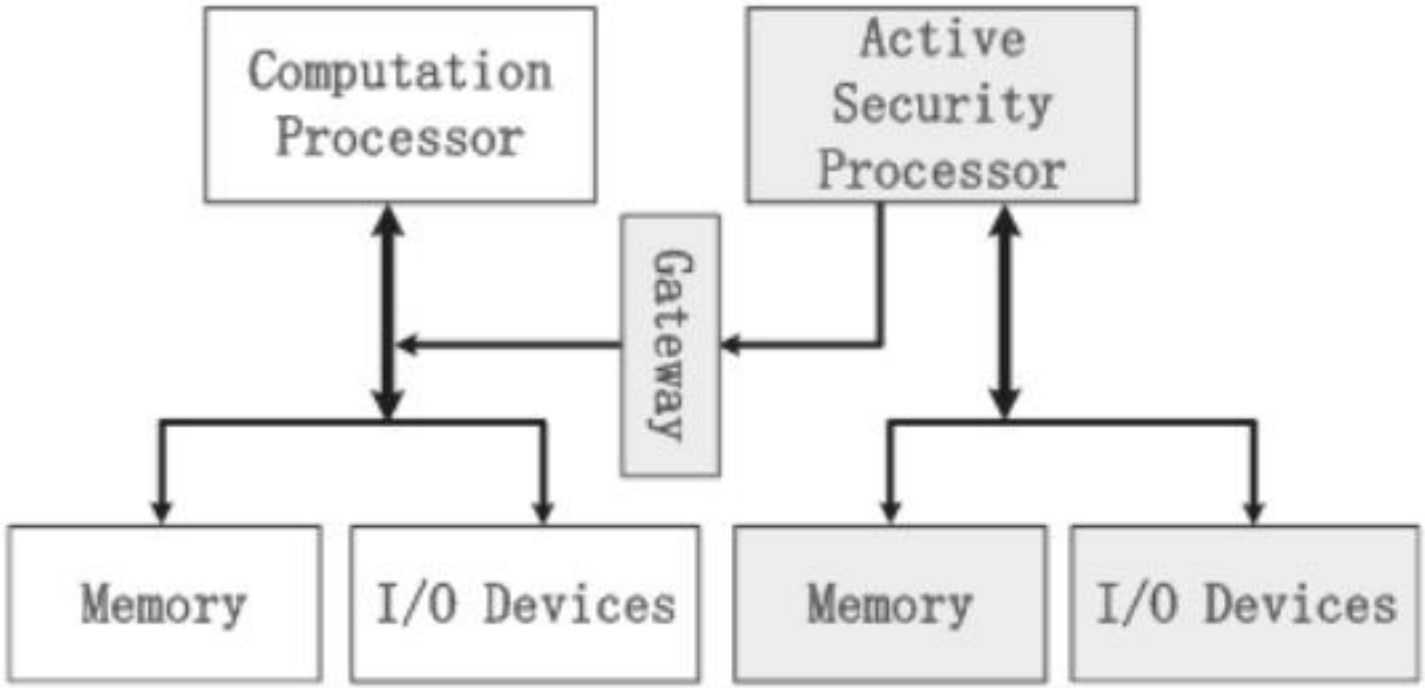


Fig. 1. Overview of the security-first architecture.

- 主动安全处理器具有最高访问权限（SuperRoot）
  - ASP具有访问计算处理器侧所有资源的权限

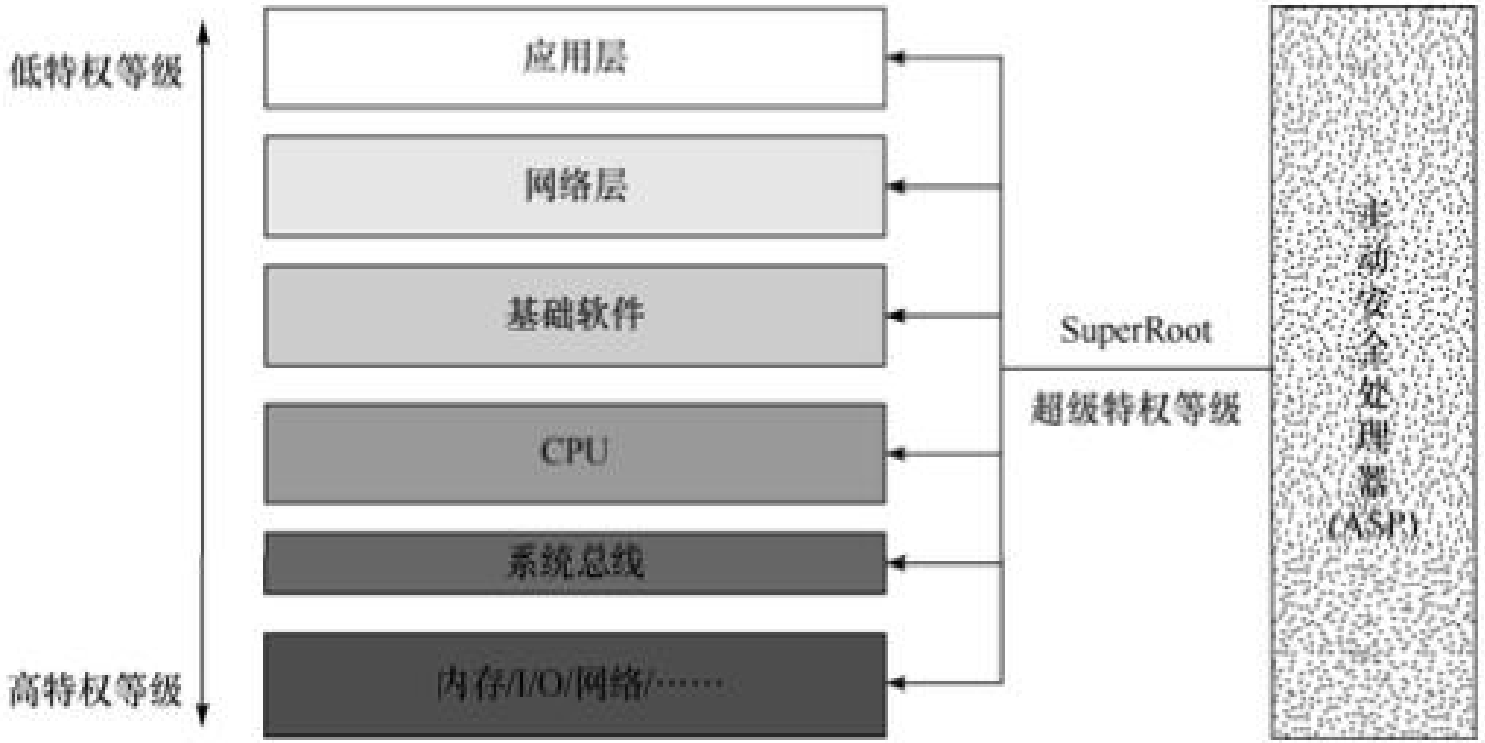
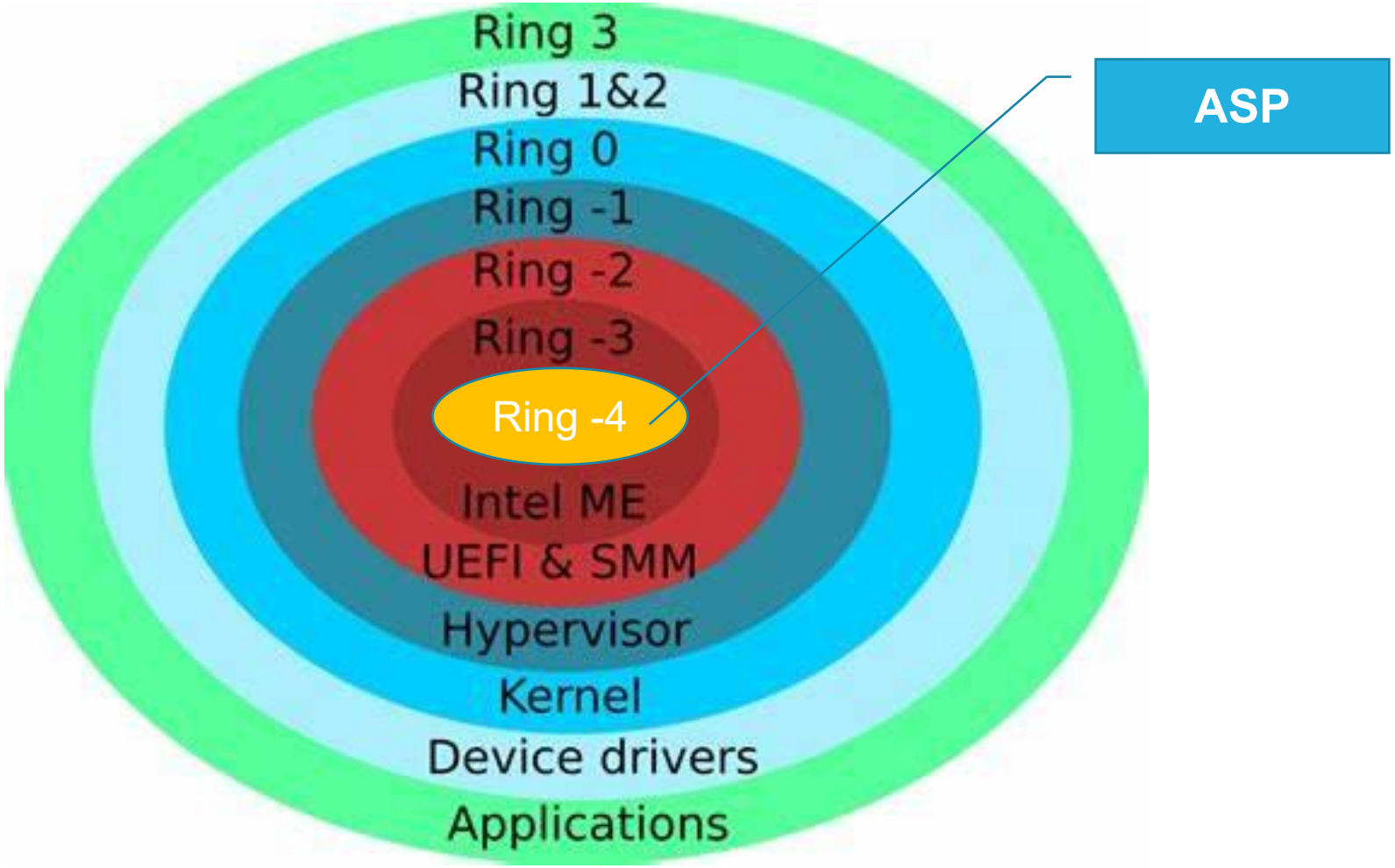


图 4.3　主动安全处理器超级访问权限

X86架构重的安全优先体系结构权限环



- 安全任务与计算任务分离
  - 安全任务自主运行在独立环境中，安全措施主动实施，两者既不是调用-被调用的关系，也不是并行的关系，是监测与被监测、控制与被控制的关系。

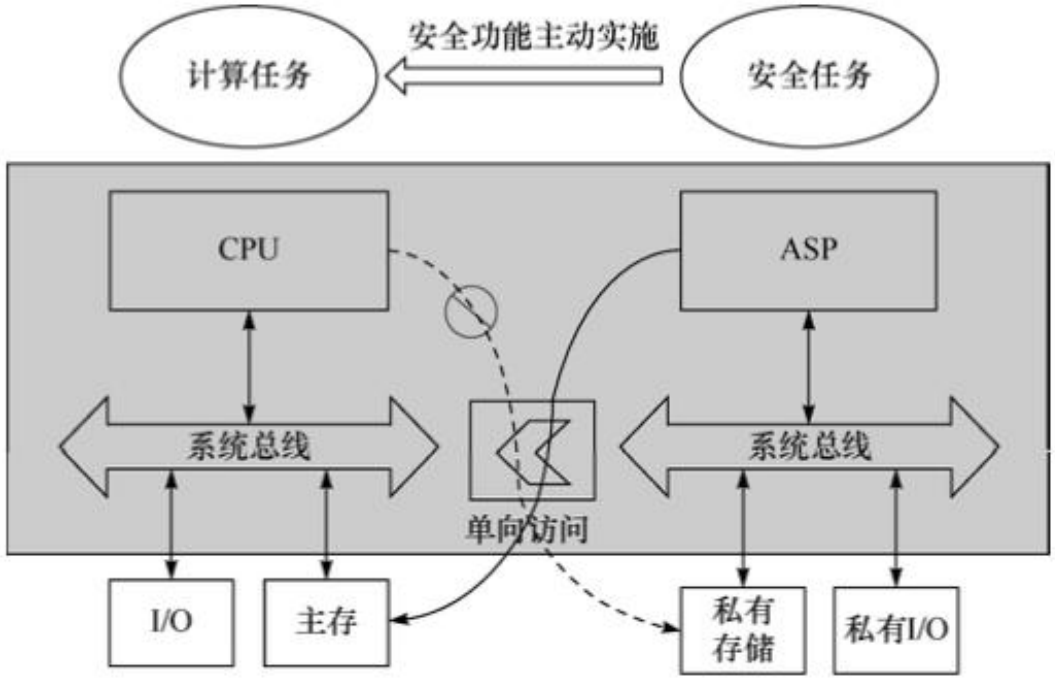


图 4.2　安全与计算分离、单向物理隔离

- 安全机制
  - 恶意行为主动在线检测和安全检查
    - 通过处理器中的性能计数器获得微架构/架构事件：Cache模式、Call-Return模式、系统调用等
    - 监测重要数据结构：抵御rootkit 攻击
    - 监测安全策略是否合规实施：I/O 操作、文件操作、日志记录等

- 安全机制
  - 举例：Rootkit攻击主动检测
    - 其关键思想是定期监测计算子系统的内存内容（关键内核数据结构的内存内容），以检测rootkit攻击的活动。

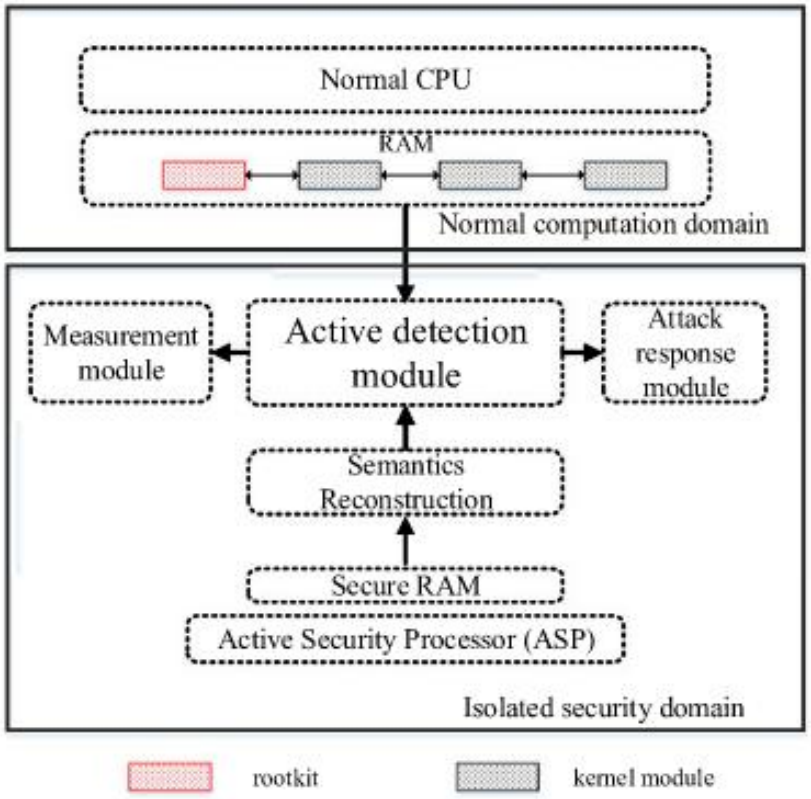


Fig. 2. Overview of active detection on Rootkit attack.

- 安全机制
  - 敏感计算任务负载分担
    - ASP 可以为合法应用程序提供安全的执行环境。选定的代码和数据可以预先离线加密，加密后的代码和数据可以密封在带有签名的ASP侧安全区域。
    - 安全区域通过专用的应用程序接口读入到 ASP，内部代码和数据就会被解密，并通过检查签名来验证其完整性。在必要的环境初始化后，代码开始运行。
    - 由于 ASP 与 CP 物理隔离，CP 上运行的任何应用程序或操作系统都看不到代码和数据。安全区域类似于Intel SGX 的飞地，不同之处在于安全区域运行在物理隔离的 ASP 上。

- 安全机制
  - 主动可信计算
    - 传统的可信计算有两个局限性：
      - 可信平台模块（TPM） **不能主动启动可信测量**，它需要由 CPU 等其他计算单元调用。因此，一旦计算机被入侵，可信测量过程就有可能被规避。
      - 虽然从 BIOS 到各级操作系统的信任链传递在理论上是合理的，但基于 TPM在实际环境中却很脆弱，因为它**缺乏运行时变化检测方法**。
    - 基于ASP的主动可行计算：
      - 静态可信度量扩展到操作系统内核、模块、库和所有应用程序都必须在**执行前进行测量**。
      - 动态可信度量是在计算机运行时进行的，它测量**运行时的代码段和静态数据**等许多因素。

- 安全机制
  - 主动噪声干扰
    - 在安全优先的架构中，ASP可以主动添加噪声，使确定性的CPU模式增加“非确定性变化”，作为计算机的信任根，ASP负责通过在CPU缓存内部动态配置随机表和相关的刷新逻辑来随机化CPU缓存布局。
    - ASP决定:
      - 是否启用/禁用缓存布局随机化;
      - 何时随机化当前缓存的布局
      - 如何通过调整密钥来更新随机表的哈希键

优势1：保障防御体系自身安全

- 安全控制环境与执行环境单向分离，**结构隔离强度高**
- 为防御体系核心提供**可信的执行环境**

优势2：提升防御体系运行效率

- 为防御体系核心提供**高效的处理引擎**
- 为密码运算、安全AI推理提供**高效的硬件加速**

优势3：兼顾计算效率与安全

- 防御体系核心运行在ASP上，**不占用系统计算资源**
- CPU上安全负载减少，**有效减少安全与计算之间任务切换开销**

- 四种典型的系统级实施方案：
  - 在芯片组内部实现

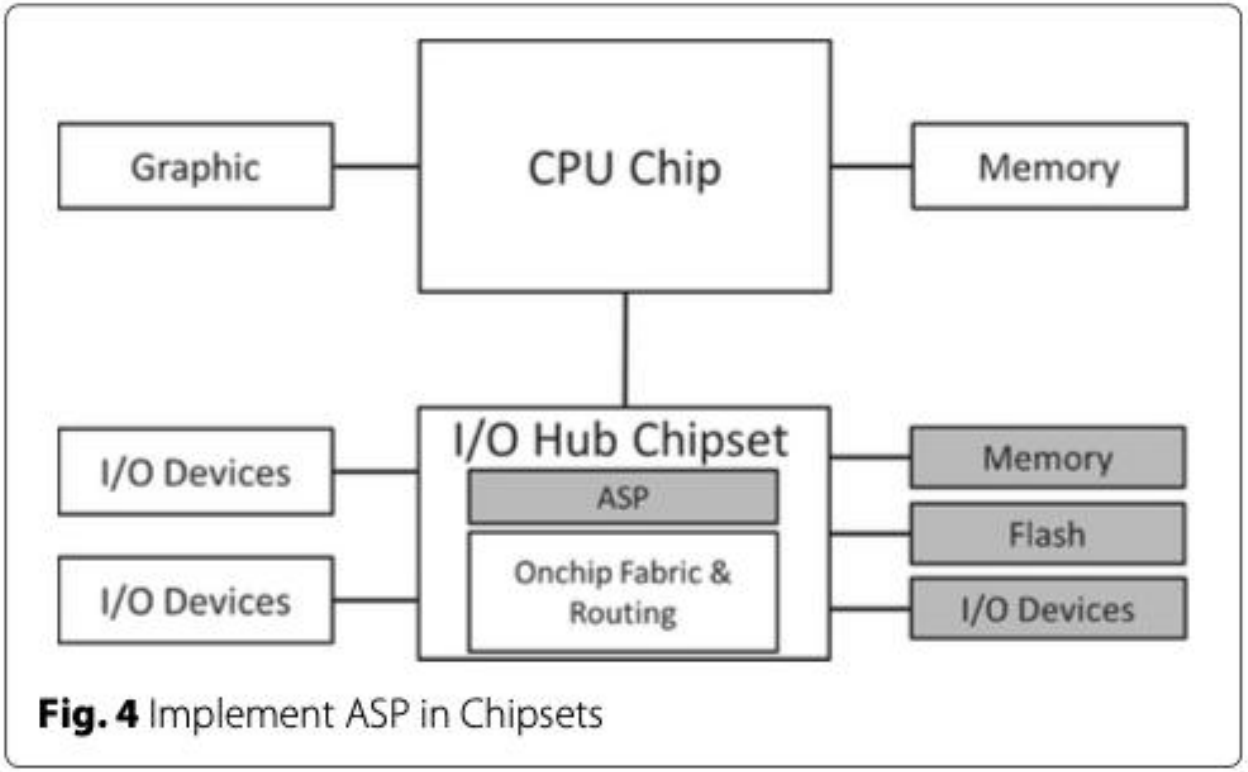


Fig. 4 Implement ASP in Chipsets

- 四种典型的系统级实施方案：
  - 在处理器芯片内实现

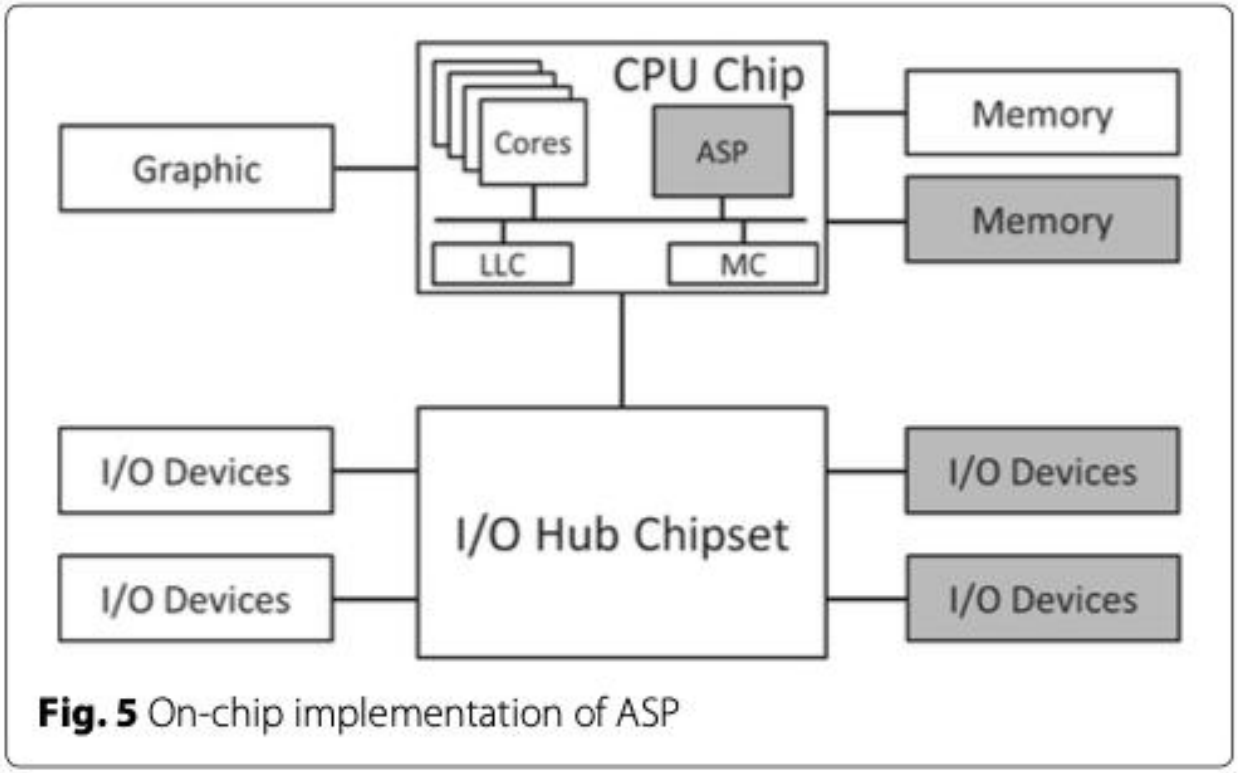


Fig. 5 On-chip implementation of ASP

- 四种典型的系统级实施方案：
  - 在独立芯片上实现

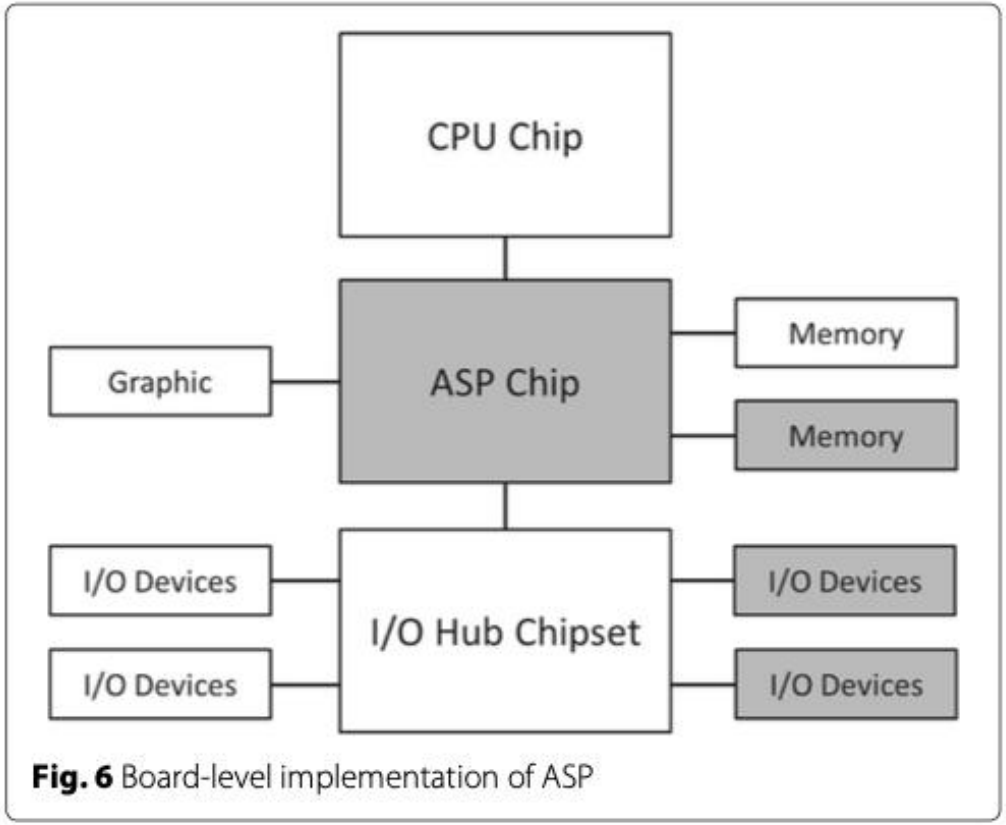
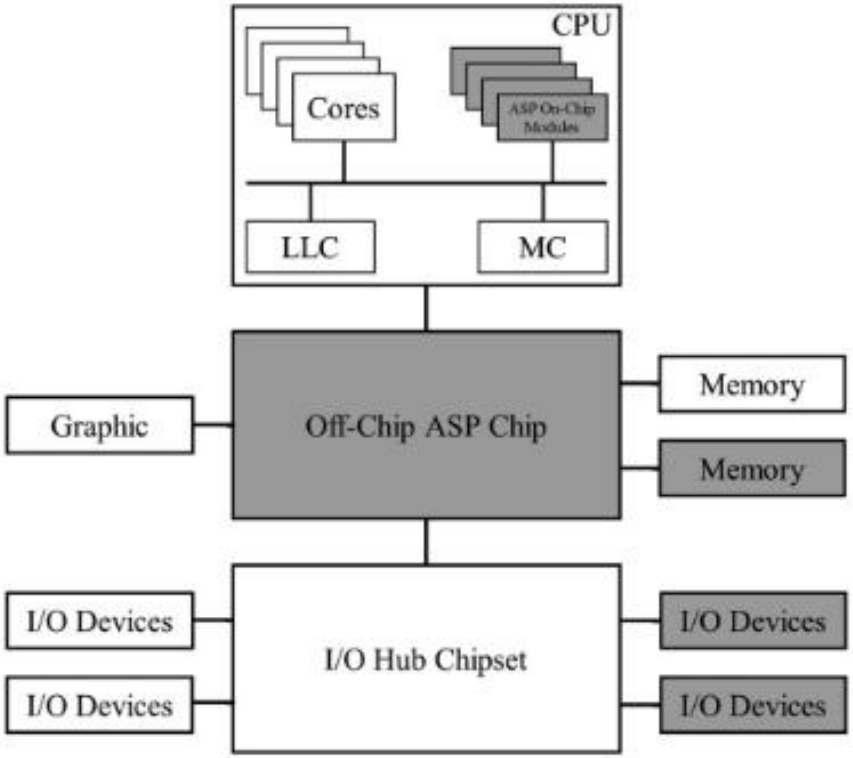


Fig. 6 Board-level implementation of ASP

- 四种典型的系统级实施方案：
  - 片上和片外组合实现



内容概要

- SP & BP体系结构 (Princeton)
- Cheri体系结构 (Cambridge)
- Security-First 体系结构 (IIE-CAS)
- 总结

总结

- 介绍了从2005年到目前，3个有代表性的安全体系结构的项目实践。
  - SP：通过加密手段，保护用户重要隐私数据安全性为目标
  - BP：通过对Hypervisor的保护，支持多个保护环境
  - CHERI：通过对指针数据的保护，解决内存安全问题
  - Security-First：通过将安全提高到最高权限，对其它环境进行监测和调控

总结

- 体会到安全在体系结构中的地位从“**附属**”到“**重视**”到“**最高**”的转变。
- 展望未来，可以肯定，安全将越来越多的**融入到体系结构设计**之中。


Q&A