

# Processor Architecture Security

## Part 1: Processor Security and Secure Processors



**Jakub Szefer**  
Assistant Professor  
Dept. of Electrical Engineering  
Yale University

**ACACES 2019 – July 14<sup>th</sup> - 20<sup>th</sup>, 2019**

Slides and information available at: <https://caslab.csl.yale.edu/tutorials/acaces2019/>

# Principles of Computer Architecture



Traditional computer architecture has six principles regarding processor design:

- Caching E.g. caching frequently used data in a small but fast memory helps hide data access latencies.
- Pipelining E.g. break processing of an instruction into smaller chunks that can each be executed sequentially reduces critical path of logic and improves performance.
- Predicting E.g. predict control flow direction or data values before they are actually computed allows code to execute speculatively.
- Parallelizing E.g. processing multiple data in parallel allows for more computation to be done concurrently.
- Use of indirection E.g. virtual to physical mapping abstracts away physical details of the system.
- Specialization E.g. custom instructions use dedicated circuits to implement operations that otherwise would be slower using regular processor instructions.

## What are principles for securing processors?

# Processor Security and Secure Processors



Processor security focuses on ensuring **Confidentiality, Integrity, and Availability** from attacks by intelligent adversaries

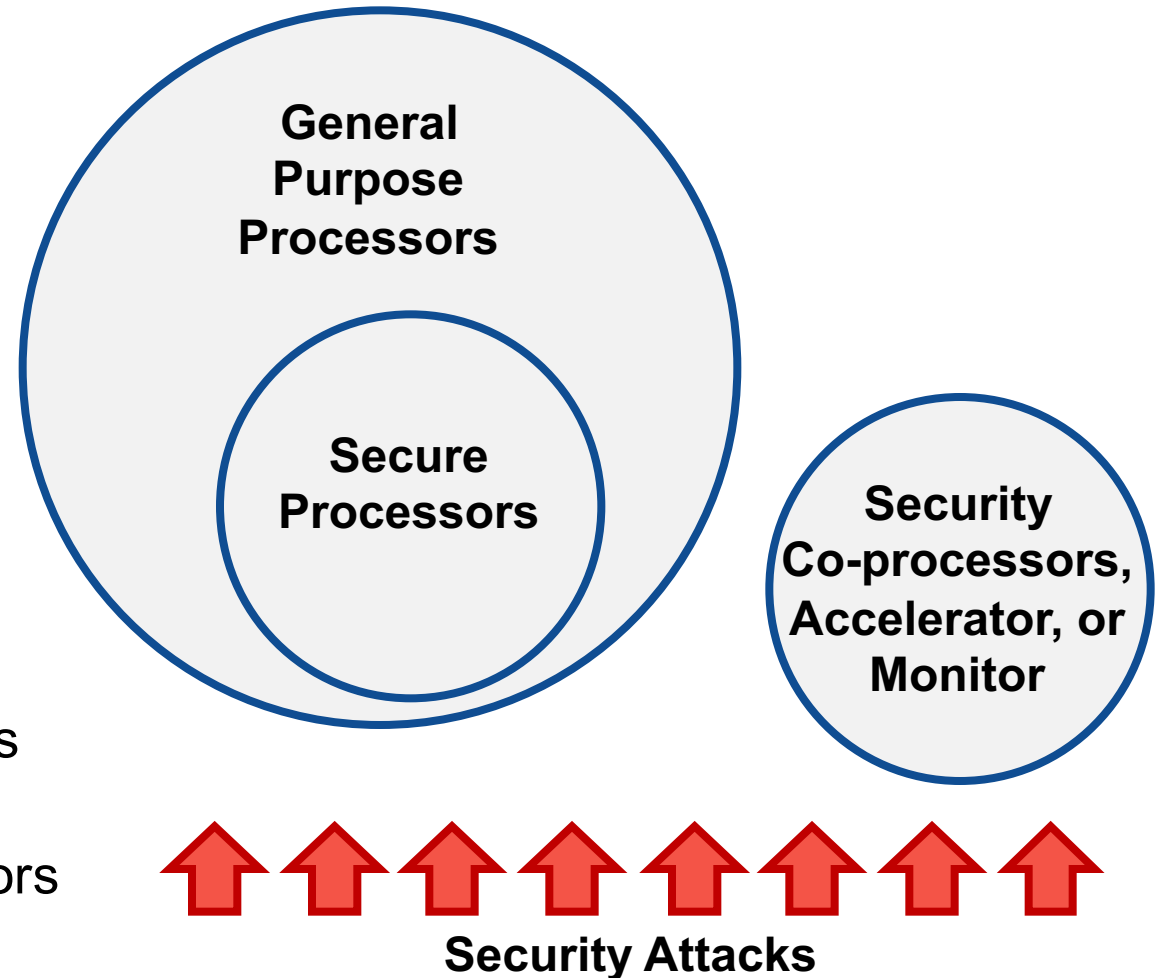
- Reliability is about random errors
- Security is about “smart” attackers

**Course focus: architecture and hardware**

- Many attacks exist on software
- Focus on attacks abusing hardware

**Secure processors:**

- Subset of processors with extra security features
- Provide extra logical isolation for software
- Vulnerable to similar attacks as regular processors





## **Part 1: Processor Security and Secure Processors**

- Present features of secure processors
- Contrast to conventional processors

## **Part 2: Side and Covert Channels**

- Detail side and covert channel attacks on conventional processors and secure processors
- Focus on timing channels

## **Part 3: Securing Caches, Buffers, TLBs, and Directories**

- Present defenses for timing channels in the memory hierarchy
- Solutions for conventional processors and secure processors

## **Part 4: Transient Execution Attacks and Mitigations**

- Discuss attacks leveraging transient execution (and timing channels) and defenses for conventional processors and secure processors

# Secure Processor Architectures



Secure Processor Architectures extend a processor with hardware (and related software) features for protection of software

- Protected pieces of code and data are now commonly called Enclaves
  - But can be also Trusted Software Modules, whole Operating Systems, or Virtual Machines
- Focus on the main processor in the system
  - Others focus on co-processors, cryptographic accelerators, or security monitors
- Add more features to isolate secure software from other, untrusted software
  - Includes untrusted Operating System or Virtual Machines
  - Many also consider physical attacks on memory
- Isolation *should* cover all types of possible ways for information leaks
  - Architectural state
  - Micro-architectural state Most recent threats, i.e. Spectre, etc.
  - Due to spatial or temporal sharing of hardware Side and covert channel threats

# Brief History of Secure Processor Architectures



**Starting with a typical baseline processor, many secure architectures have been proposed**

Starting in late 1990s or early 2000s, academics have shown increased interest in secure processor architectures:

XOM (2000), AEGIS (2003), Secret-Protecting (2005), Bastion (2010),  
NoHype (2010), HyperWall (2012), Phantom (2013), CHERI (2014), Sanctum (2016),  
Keystone (about 2017), Ascend (2017), MI6 (2018)

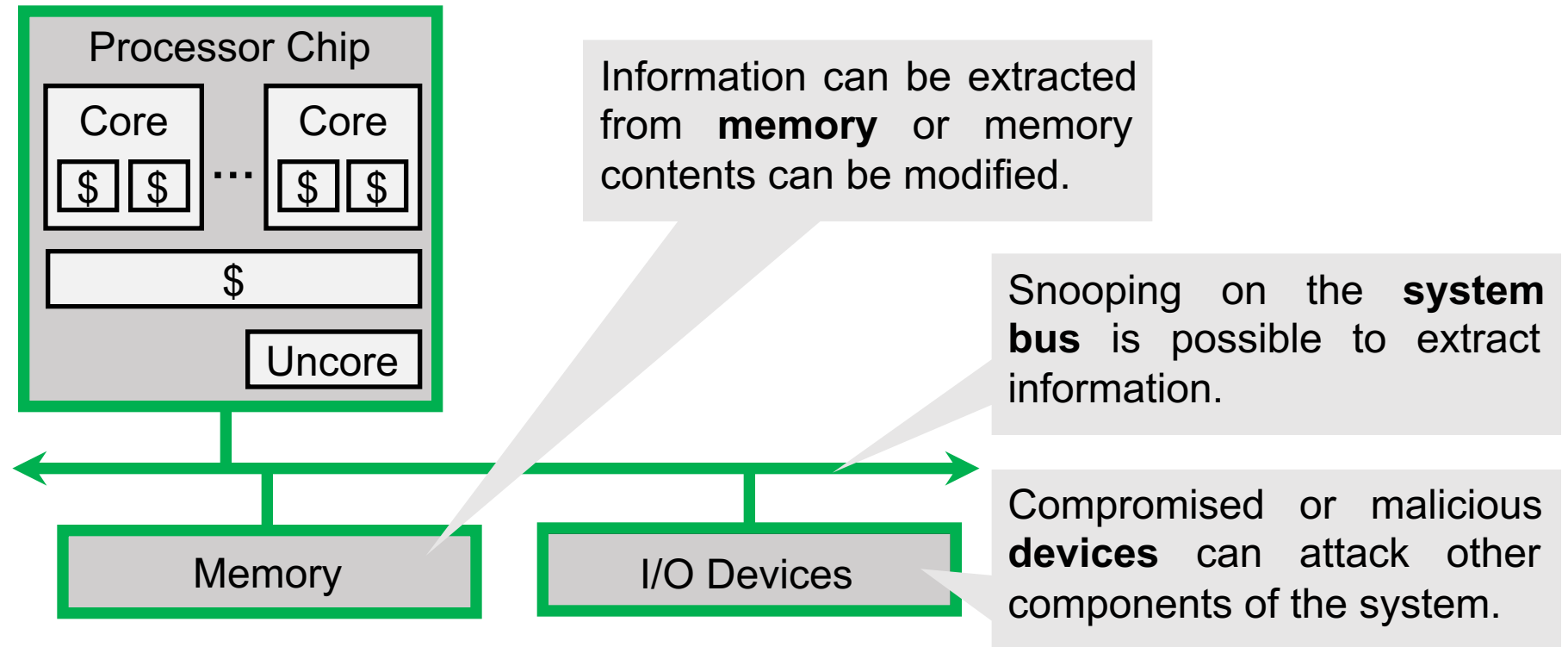
Commercial processor architectures have also included security features:

LPAR in IBM mainframes (1970s), Security Processor Vault in Cell Broadband Engine (2000s),  
ARM TrustZone (2000s), Intel TXT & TPM module (2000s), Intel SGX (mid 2010s),  
AMD SEV (late 2010s)

# Baseline (Unsecure) Processor Hardware



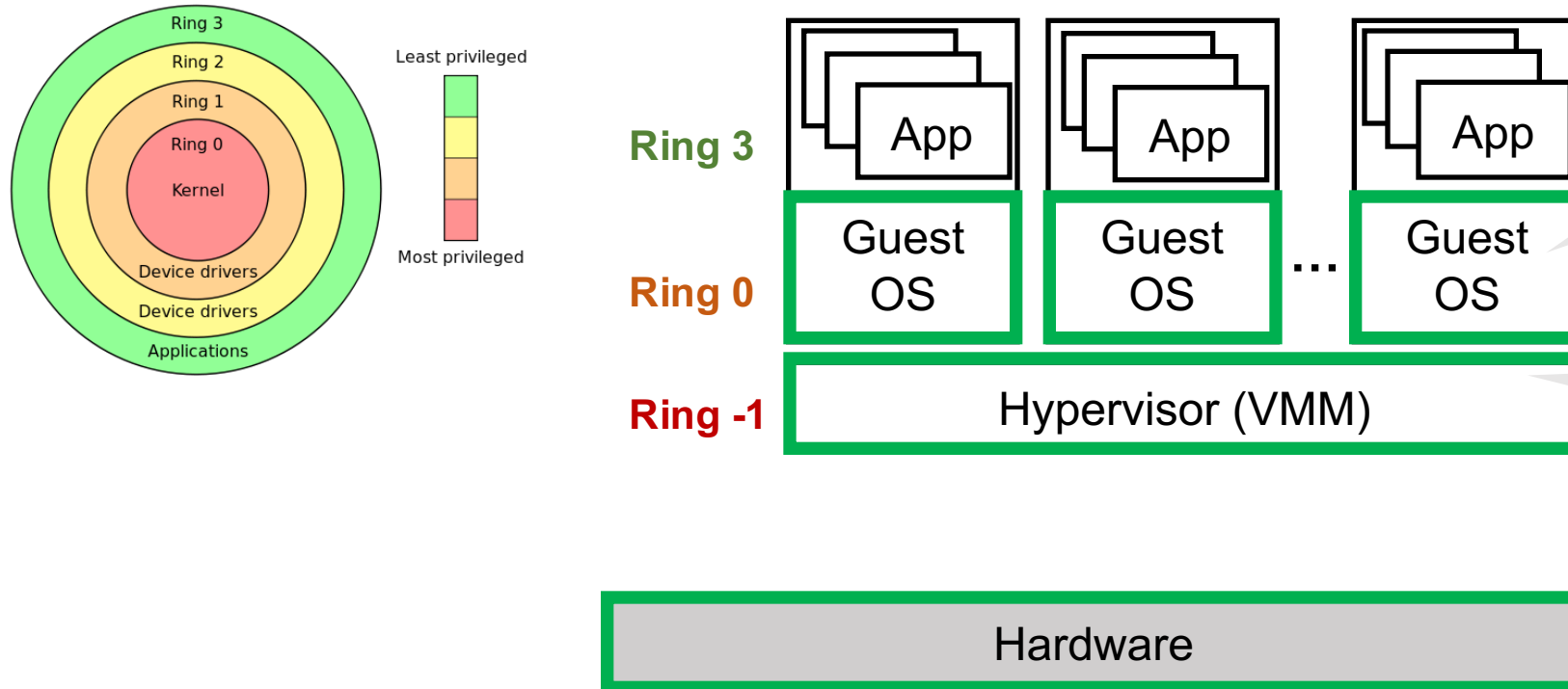
Typical computer system with no secure components nor secure processor architectures considers all the components as trusted:



# Baseline (Unsecure) Processor Software



Typical computer system uses ring-based protection scheme, which gives most privileges (and most trust) to the lowest levels of the system software:



Compromised or malicious **OS** can attack all the applications in the system.

Compromised or malicious **Hypervisor** can attack all the OSes in the system.

Image: [https://commons.wikimedia.org/wiki/File:Priv\\_rings.svg](https://commons.wikimedia.org/wiki/File:Priv_rings.svg)



# Logical Isolation with New Privilege Levels



Modern computer systems define protections in terms of **privilege level** or protection rings, new privilege levels are defined to provide added protections.

- Ring 3** Application code, least privileged.
- Rings 2 and 1** Device drivers and other semi-privileged code, although rarely used.
- Ring 0** Operating system kernel.
- Ring -1** Hypervisor or virtual machine monitor (VMM), most privileged mode that a typical system administrator has access to.
- Ring -2** System management mode (SMM), typically locked down by processor manufacturer
- Ring -3** Platform management engine, retroactively named “ring -3”, actually runs on a separate management processor.

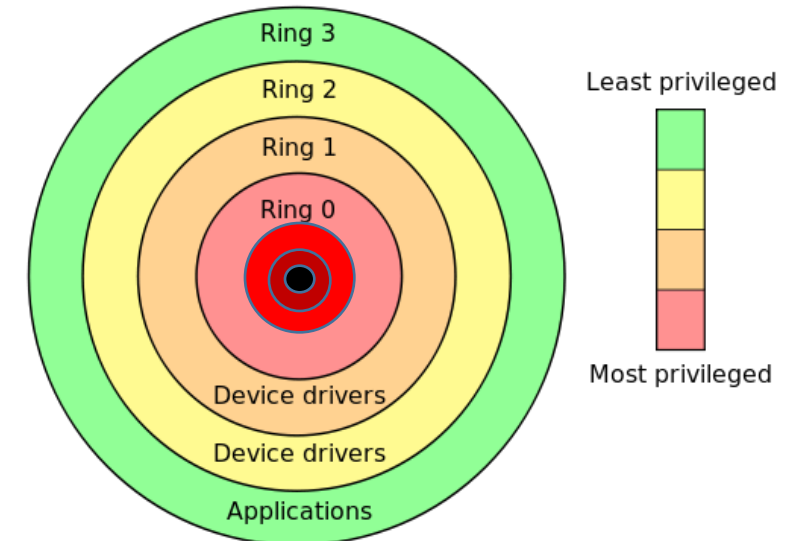


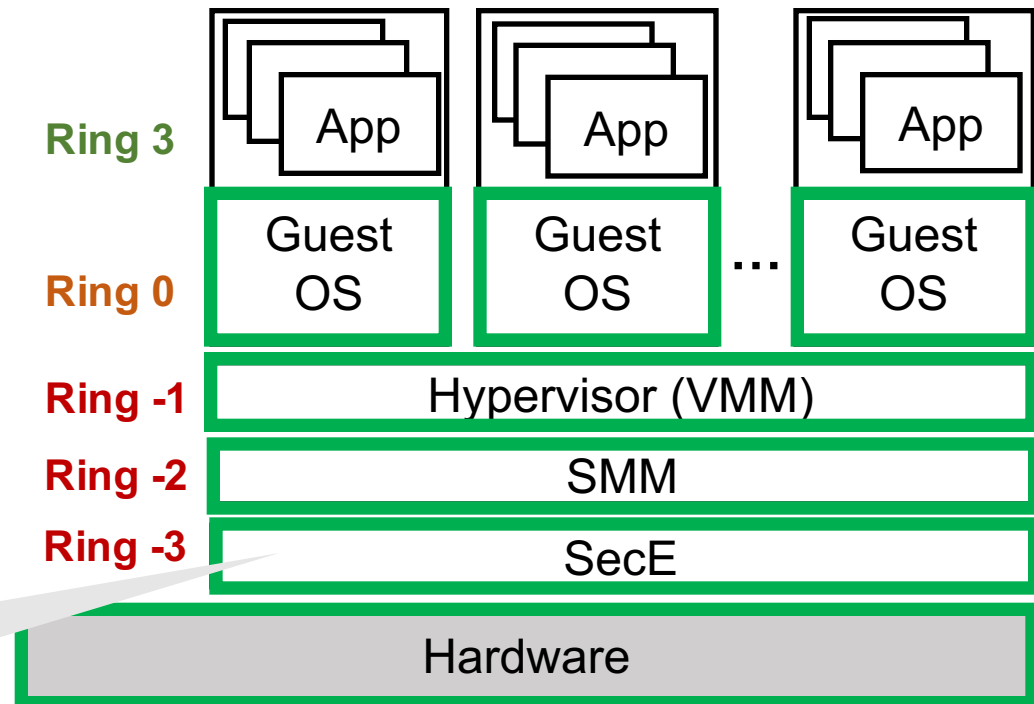
Image:  
[https://commons.wikimedia.org/wiki/File:Priv\\_rings.svg](https://commons.wikimedia.org/wiki/File:Priv_rings.svg)

# Extend Linear Trust to the New Protection Levels



The hardware is most privileged as it is the lowest level in the system.

- There is a linear relationship between protection ring and privilege (lower ring is more privileged)
- Each component **trusts** all the software “below” it



**Security Engine (SecE)** can be something like Intel's ME or AMD's PSP.

# Add Horizontal Privilege Separation



New privileges can be made orthogonal to existing protection rings.

- E.g. ARM's TrustZone's "normal" and "secure" worlds
- Need privilege level (ring number) and normal / secure privilege

Security levels from a lattice:

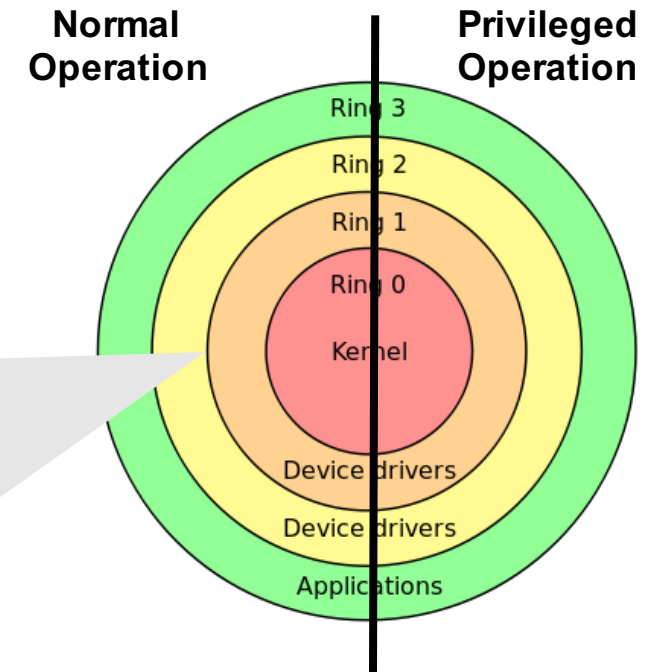
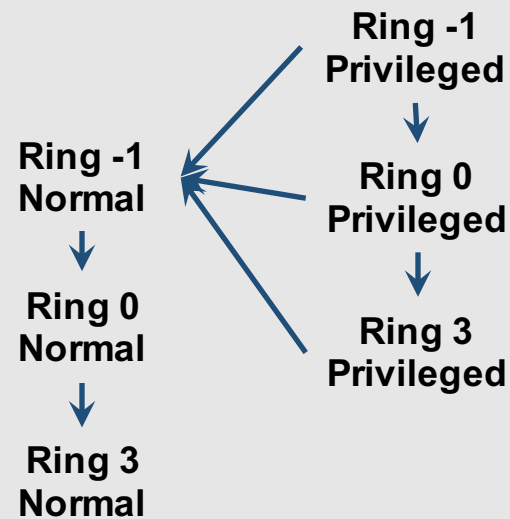
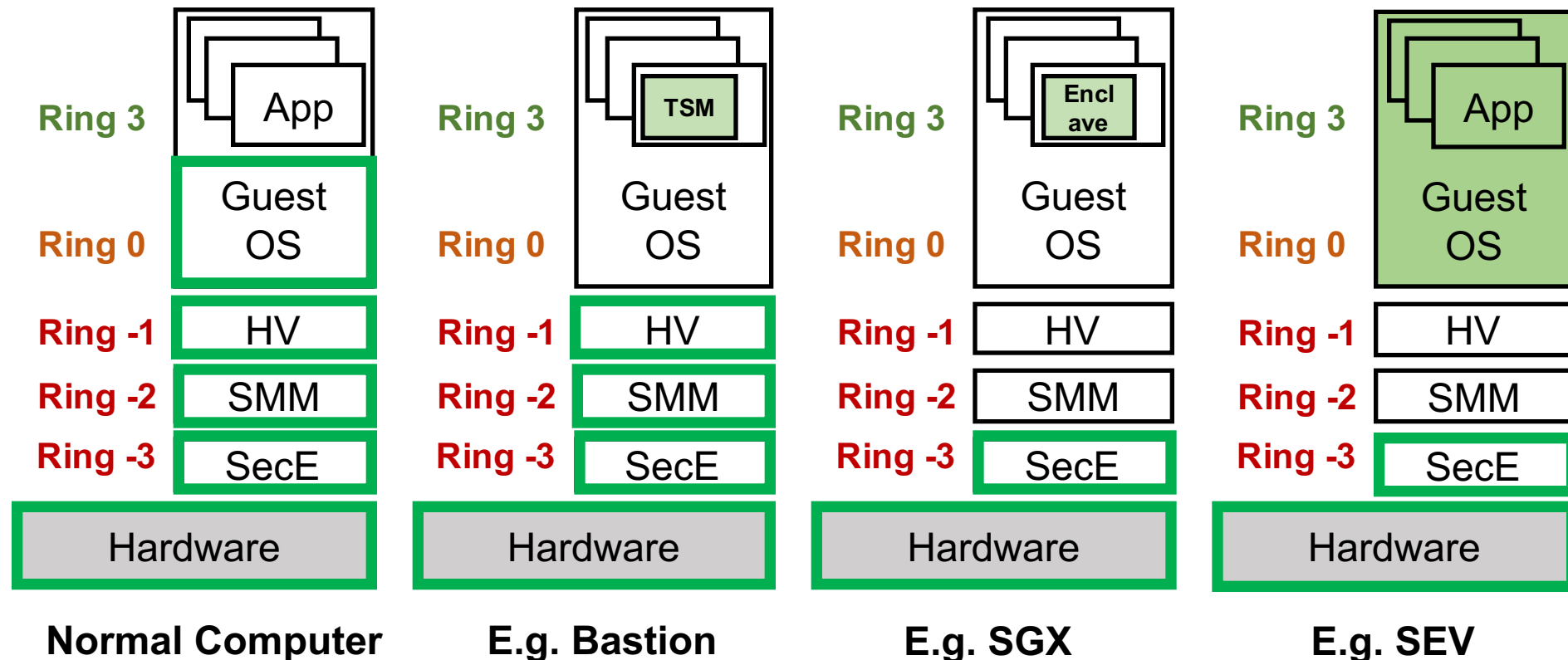


Image:  
[https://commons.wikimedia.org/wiki/File:Priv\\_rings.svg](https://commons.wikimedia.org/wiki/File:Priv_rings.svg)

# Breaking Linear Hierarchy of Protection Rings



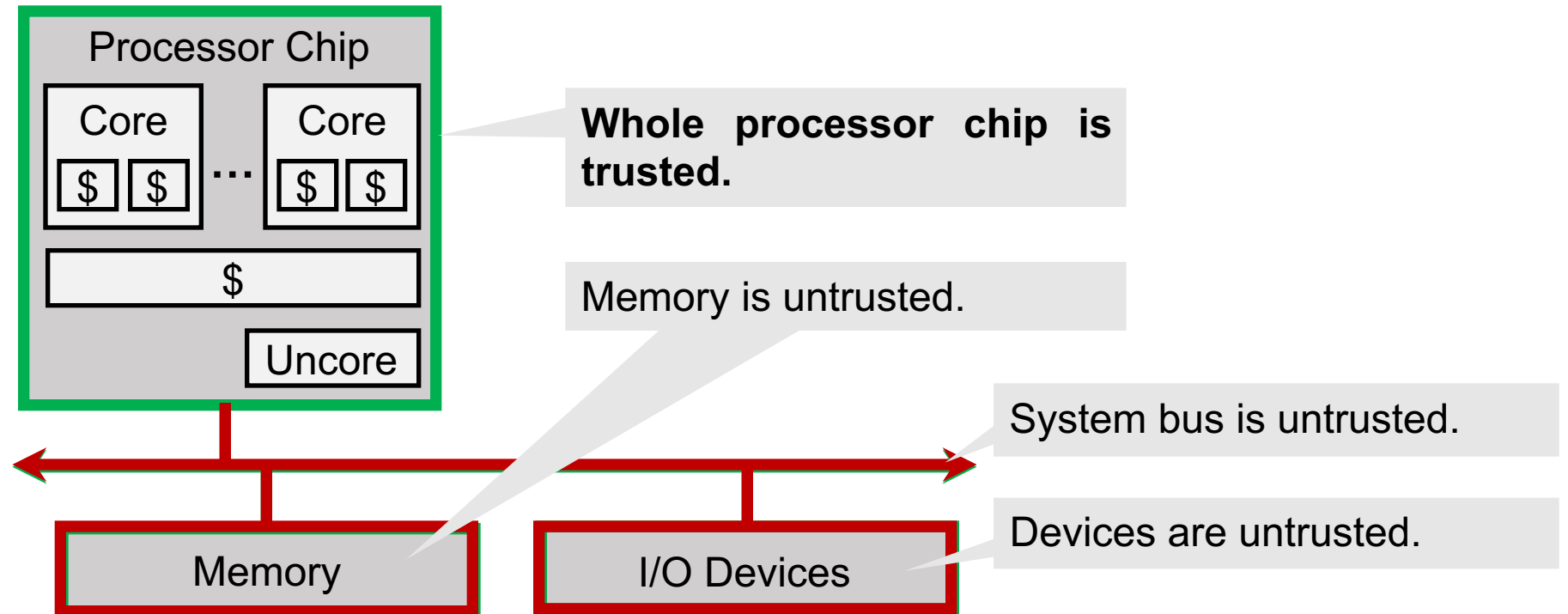
Examples of architectures that do and don't have a linear relationship between privileges and protection ring level:



# Providing Protections with a Trusted Processor Chip



Key to most secure processor architecture designs is the idea of **trusted processor chip** as the security wherein the protections are provided.



# Limitations of Secure Processors



## Threats which are outside the scope of secure processor architectures:

- Bugs or Vulnerabilities in the TCB
- Hardware Trojans and Supply Chain Attacks
- Physical Probing and Invasive Attacks

TCB hardware and software is prone to bugs just like any hardware and software.

Modifications to the processor after the design phase can be sources of attacks.

At runtime hardware can be probed to extract information from the physical realization of the chip.

## Threats which are underestimated when designing secure processor architectures:

- Side Channel Attacks

Information can leak through timing, power, or electromagnetic emanations from the implementation



The **Trusted Computing Base (TCB)** is the set of hardware and software that is responsible for realizing the TEE:

- TEE is created by a set of all the components in the TCB
- TCB is trusted to correctly implement the protections
- Vulnerability or successful attack on TCB nullifies TEE protections
- TCB is trusted
- TCB may not be trustworthy, if is not verified or is not bug free

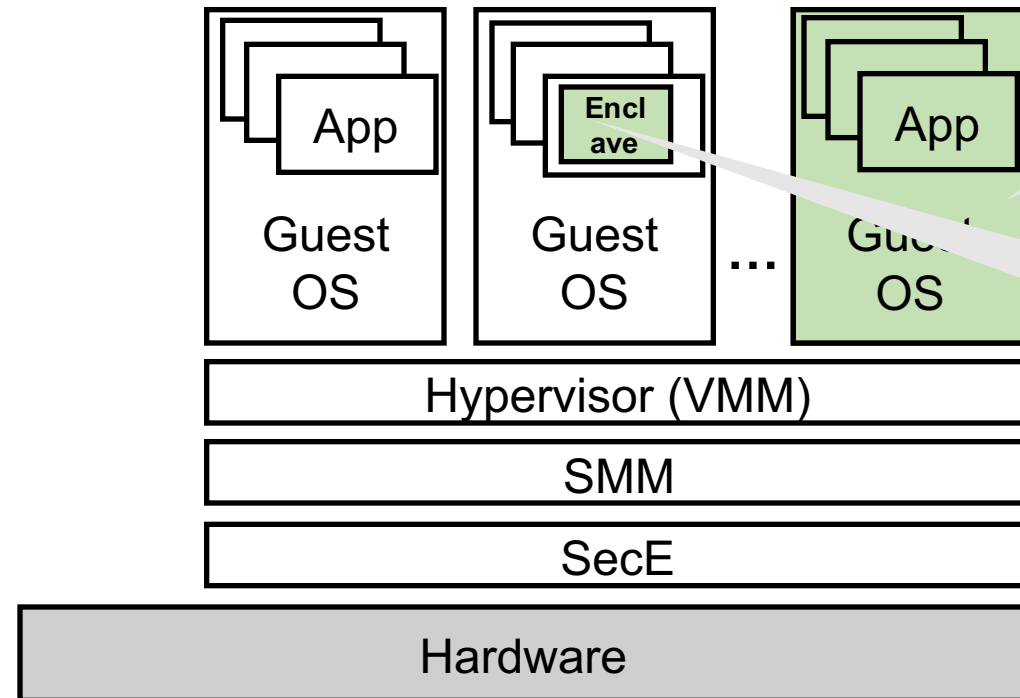
The goal of **Trusted Execution Environments (TEEs)** is to provide protections for a piece of code and data from a range of software and hardware attacks.

- Multiple mutually-untrusting pieces of protected code can run on a system at the same time

# TEEs and Software They Protect



Different architectures mainly focus on **protecting Trusted Software Modules** (a.k.a. enclaves) or **whole Virtual Machines**.



Some TEEs have support for protecting whole virtual machines.

Other TEEs support Trusted Software Modules, a.k.a. enclaves



# Protections Offered by Secure Processor Architectures



Security properties for the TEEs that secure processor architectures aim to provide:

- Confidentiality

Confidentiality is the prevention of the disclosure of secret or sensitive information to unauthorized users or entities.

- Integrity

Integrity is the prevention of unauthorized modification of protected information without detection.

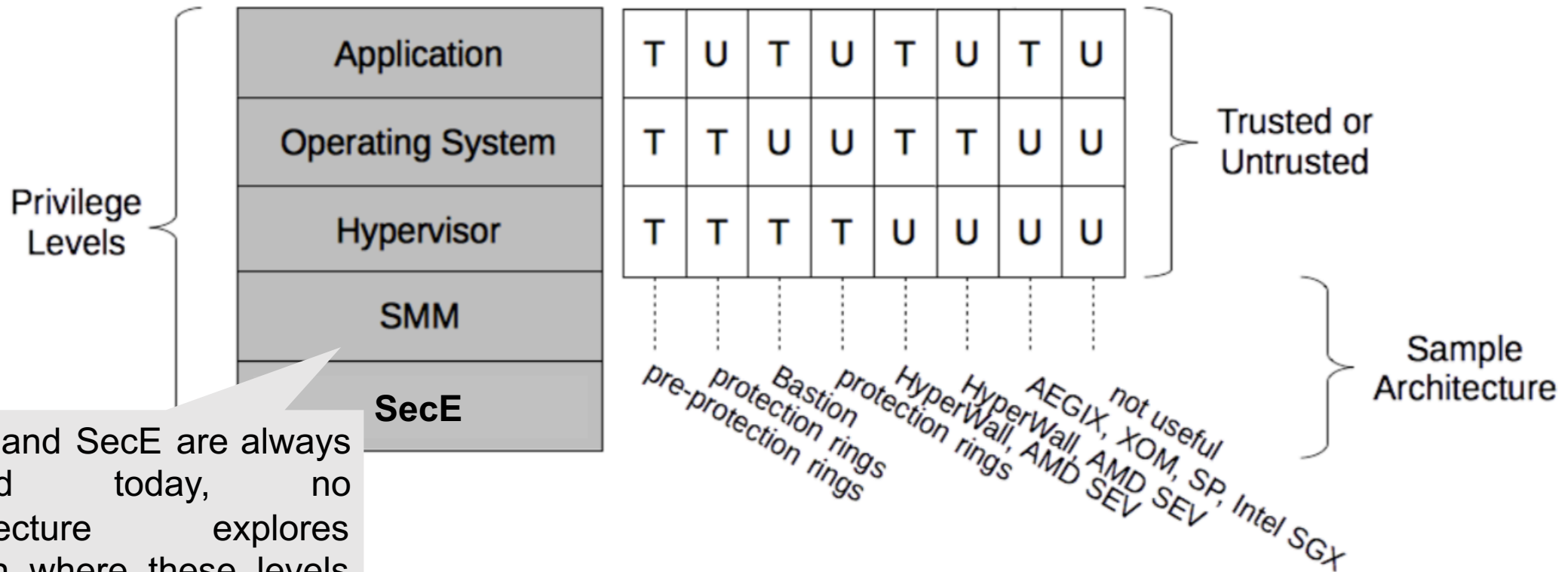
- Availability is usually not provided usually

Confidentiality and integrity protections are from attacks by other components (and hardware) not in the TCB. **There is typically no protection from malicious TCB.**

# Protections Categorized by Architecture



Secure processor architectures break the linear relationship (where lower level protection ring is more trusted):



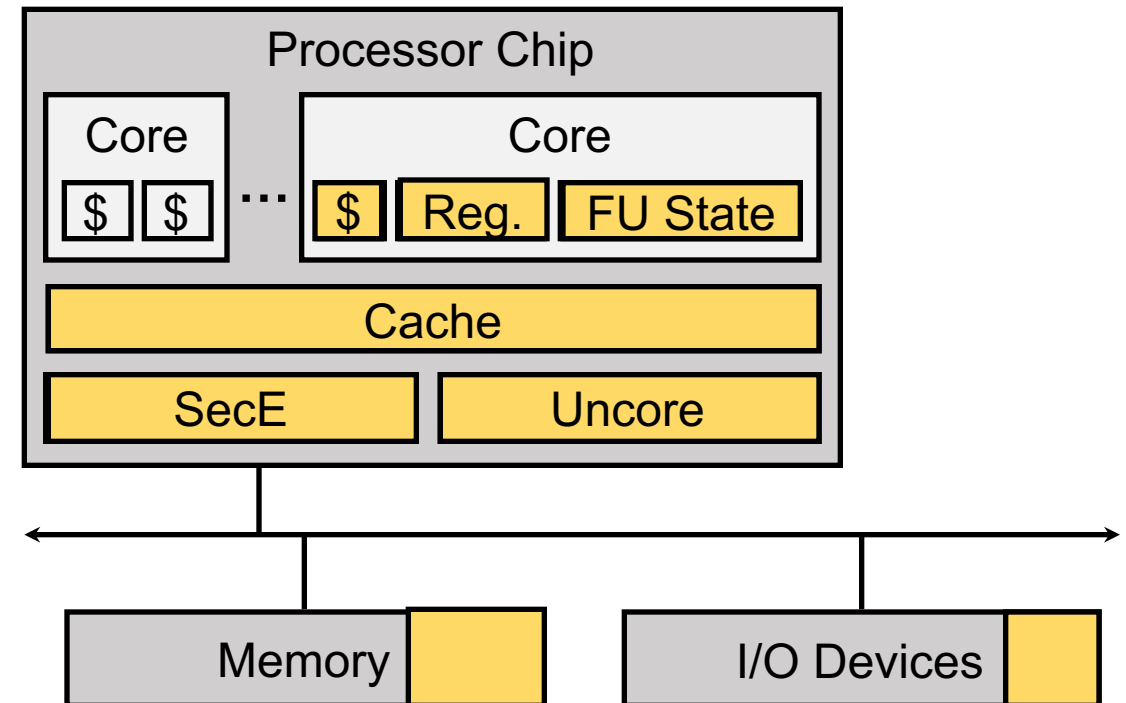
SMM and SecE are always trusted today, no architecture explores design where these levels are untrusted.

# Protecting State of the Protected Software



Protected software's **state** is distributed throughout the processor. All of it needs to be protected from the untrusted components and other (untrusted) protected software.

- Protect memory through encryption and hashing with integrity trees
- Flush state, or isolate state, of functional units in side processor cores
- Isolate state in uncore and any security modules
- Isolate state in I/O and other subsystems

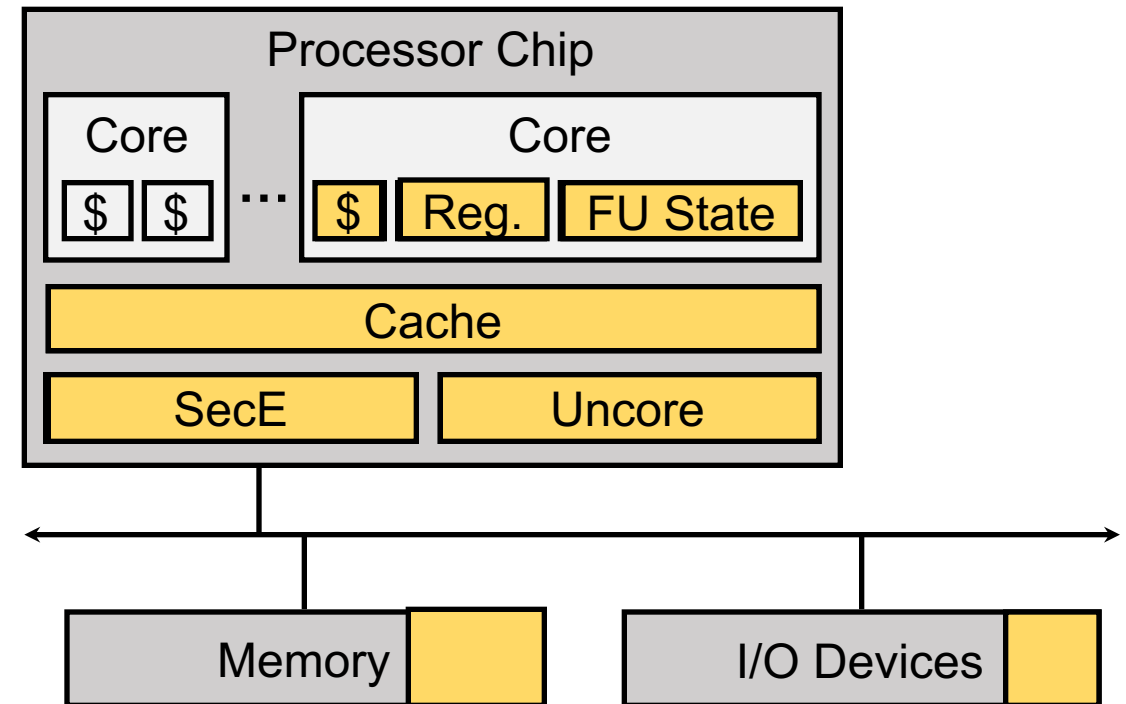


# Ideal No Side-Effects Execution



Secure processor architectures ideally have **no side-effects which are visible to the untrusted components** whenever protected software is executing.

1. System is in some state before protected software runs
2. Protected software runs modifying system state
3. When protected software is interrupted or terminates the state modifications are erased



# No Protections from Protected Software



The software (code and data) executing within TEE protections is assumed to be benign and not malicious:

- Goal of Secure Processor Architectures is to create minimal TCB that realizes a TEE within which the protected software resides and executes
- Secure Processor Architectures can not protect software if it is buggy or has vulnerabilities

**Code bloat** endangers invalidating assumptions about benign protected software.

**Attacks from within protected software** should be defended.

# Hardware TCB as Circuits or Processors



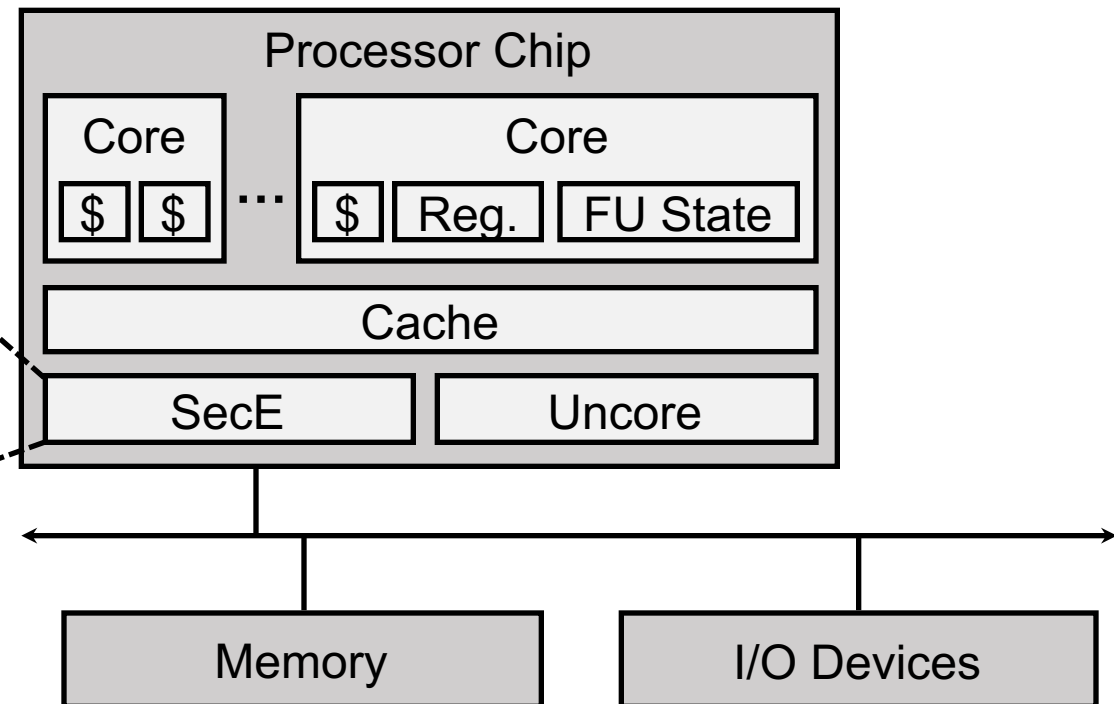
Key parts of the hardware TCB can be implemented as dedicated circuits or as firmware or other code running on dedicated processor

- **Custom logic or hardware state machine:**

- Most academic proposals

- **Code running on dedicated processor:**

- Intel ME = ARC processor or Intel Quark processor
- AMD PSP = ARM processor



**Vulnerabilities in TCB “hardware” can lead to attacks that nullify the security protections offered by the system.**

# Ensuring Trustworthy TCB Execution



Trustworthiness of the TCB depends on the ability to monitor the TCB code (hardware and software) execution as the system runs.

**TCB should be monitored to ensure it is trustworthy.**

Monitoring of TCB requires mechanisms to:

- Fingerprint and authenticate TCB code
- Monitor TCB execution
- Protect TCB code (on embedded security processor)
  - Virtual Memory, ASLR, ...

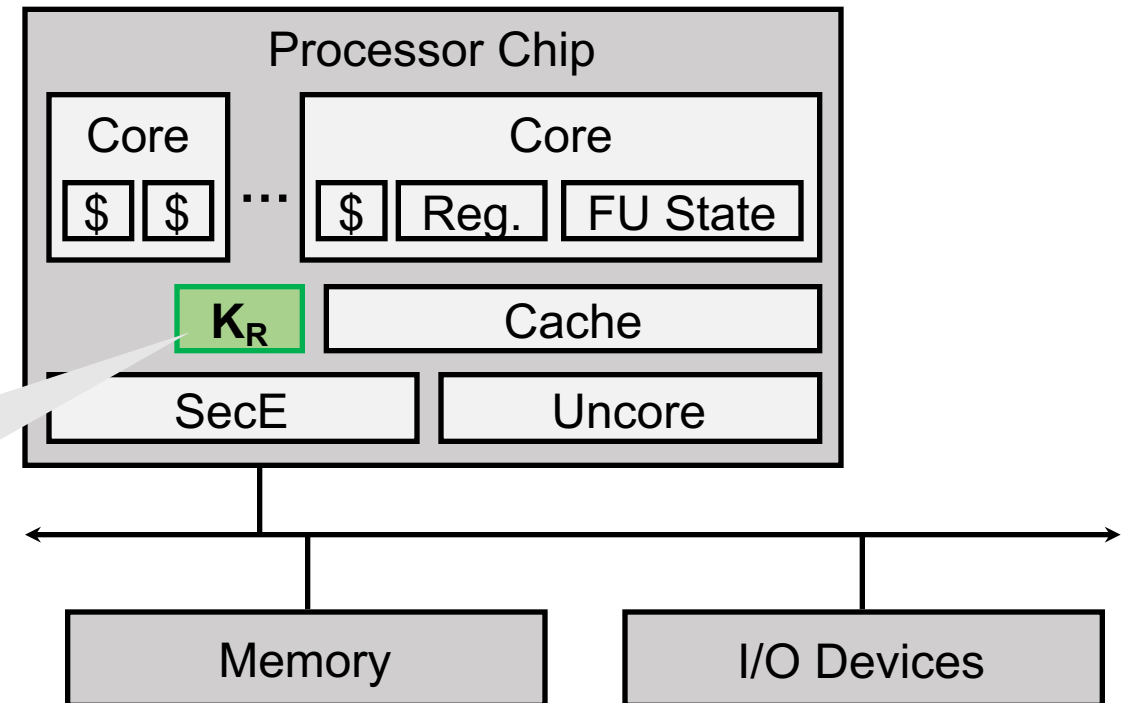
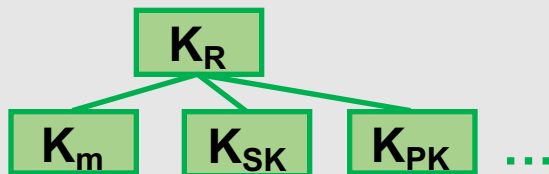
# Root of Trust for TCB



Security of the system is derived from a **root of trust**.

- A secret (cryptographic key) only accessible to TCB components
- Derive encryption and signing keys from the root of trust

Hierarchy of keys can be derived from the root of trust



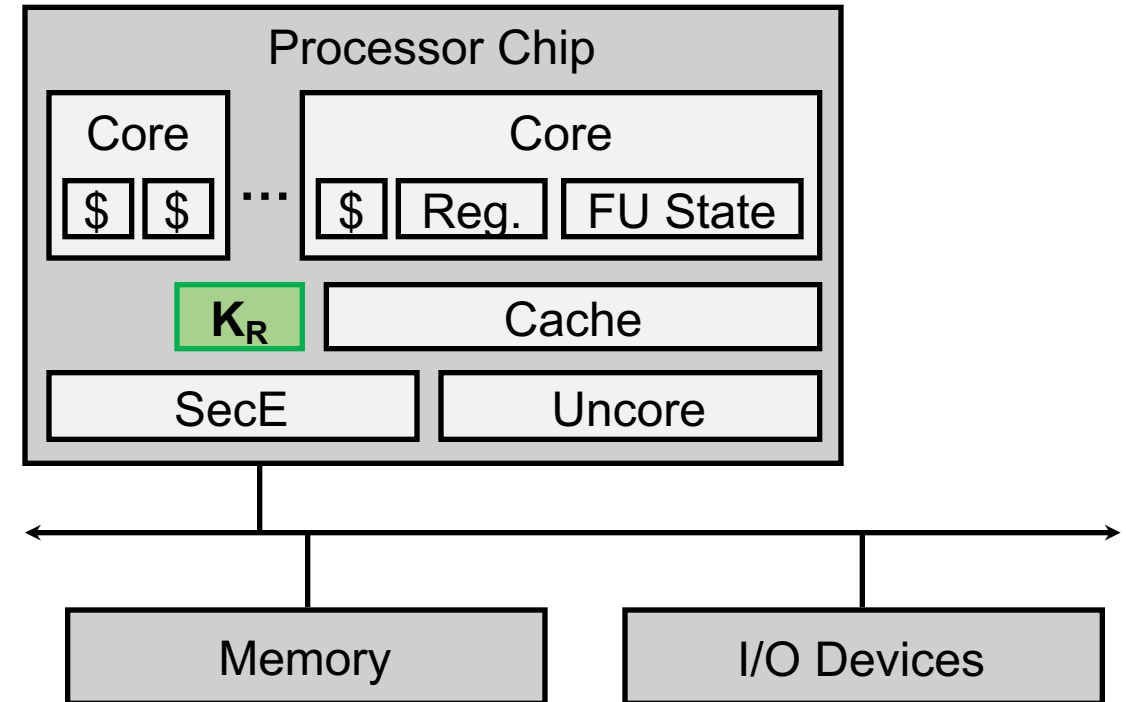


# Root of Trust and Processor Key



Each processor requires a unique secret.

- **Burn in at the factory** by the manufacturer (but implies trust issues with manufacturer and the supply chain)
  - E.g. One-Time Programmable (OTP) fuses
- Use **Physically Uncloneable Functions** (but requires reliability)
  - Extra hardware to derive keys from PUF
  - Mechanisms to generate and distribute certificates for the key

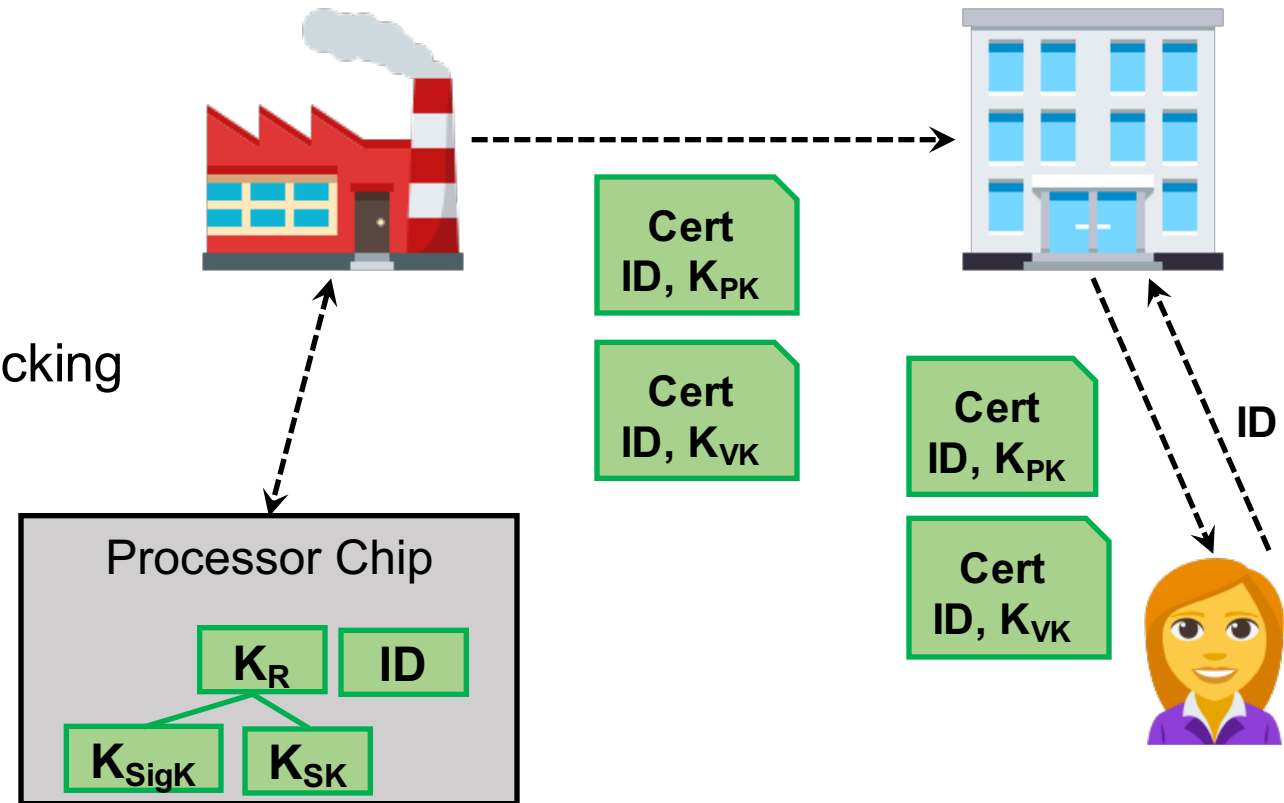


# Derived Keys and Key Distribution



Derived from the root of trust are signing and verification keys.

- Public key,  $K_{PK}$ , for encrypting data to be sent to the processor
  - Data handled by the TCB
- Signature verification key,  $K_{VK}$ , for checking data signed by the processor
  - TCB can sign user keys
- Key distribution for PUF based designs will be different
- Need infrastructure!

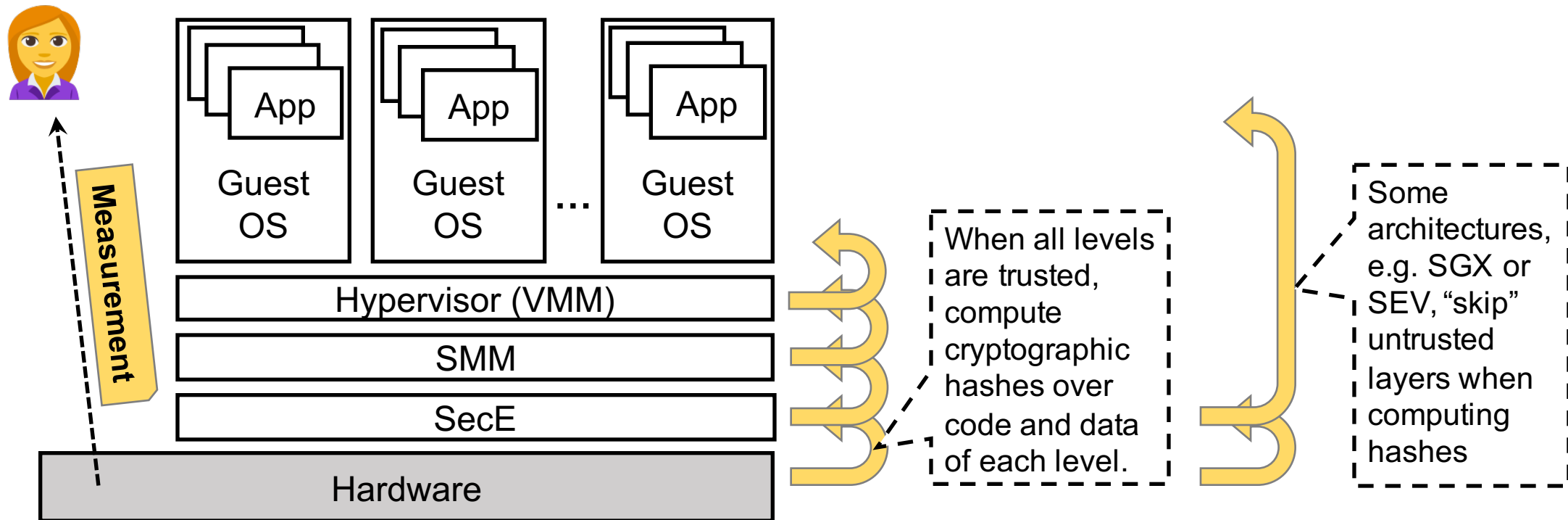


Emoji Image:  
<https://www.emojione.com/emoji/1f3ed>  
<https://www.emojione.com/emoji/1f469-1f4bc>  
<https://www.emojione.com/emoji/1f3e2>

# Software Measurement



With an embedded signing key, the software running in the TEE can be “measured” to attest to external users what code is running on the system.



Emoji Image:  
<https://www.emojione.com/emoji/1f469-1f4bc>

# Using Software Measurement



## **Trusted / Secure / Authenticated Boot:**

- Abort boot when wrong measurement is obtained
- Or, continue booting but do not decrypt secrets
- Legitimate software updates will change measurements, may prevent correct boot up

## **Remote attestation:**

- Measure and digitally sign measurements that are sent to remote user

## **Data sealing (local or remote):**

- Only unseal data if correct measurements are obtained

## **TOC-TOU attacks and measurements:**

- Time-of-Check to Time-of-Use (TOC-TOU) attacks leverage the delay between when a measurement is taken, and when the component is used
- Cannot easily use hashes to prevent TOC-TOU attacks

# Need for Continuous Monitoring of Protected Software



**Continuous monitoring** is potential solution to TOC-TOU:

- Constantly measure the system, e.g. performance counters, and look for anomalies
- Requires knowing correct and expected behavior of system
- Can be used for continuous authentication

Attacker can “hide in the noise” if they change the execution of the software slightly and do not affect performance counters significantly.



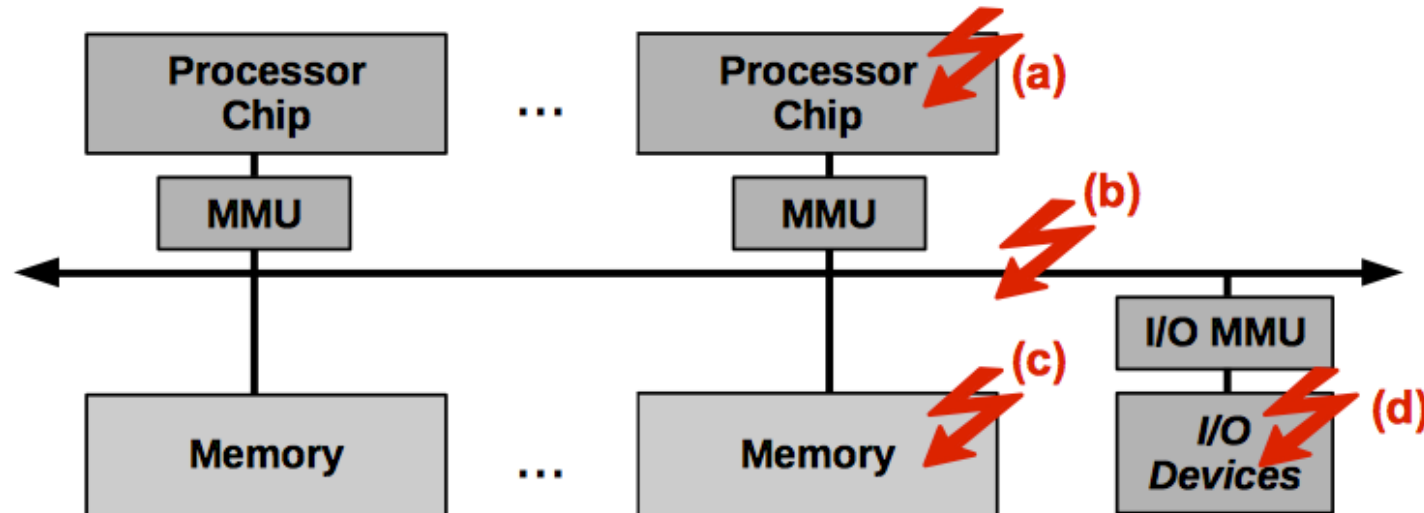
# Memory Protections in Secure Processors

# Sources of Attacks on Memory



Memory is vulnerable to different types of attacks:

- Untrusted software running on the processor
- Physical attacks on the memory bus, other devices snooping on the bus, man-in-the-middle attacks with malicious device
- Physical attacks on the memory (Coldboot, ...)
- Malicious devices using DMA or other attacks



# Types of Attacks on Memory



Different types of attacks exist (very similar to attacks in network settings):

- Snooping 

Passive attack, try to read data contents.
- Spoofing 

Active attack, inject new memory commands to try to read or modify data.
- Splicing 

Active attack, combine portions of legitimate memory commands into new memory commands (to read or modify data).
- Replay 

Active attack, re-send old memory command (to read or modify data).
- Disturbance 

Active attack, DoS on memory bus, repeated memory accesses to age circuits, repeated access to make Rowhammer, etc.



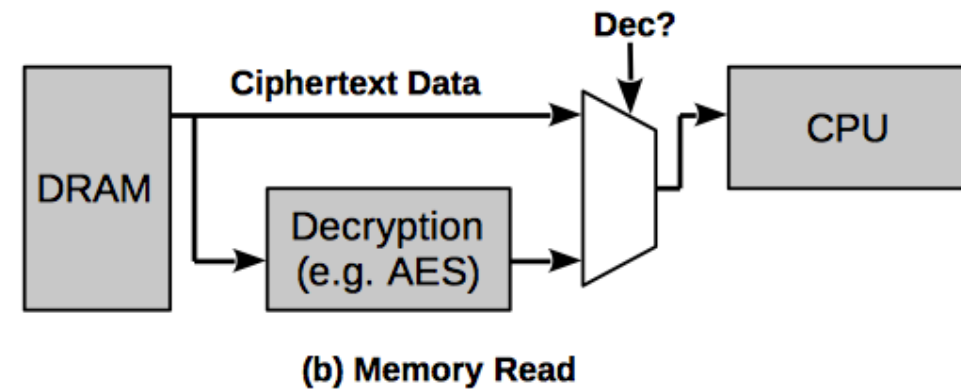
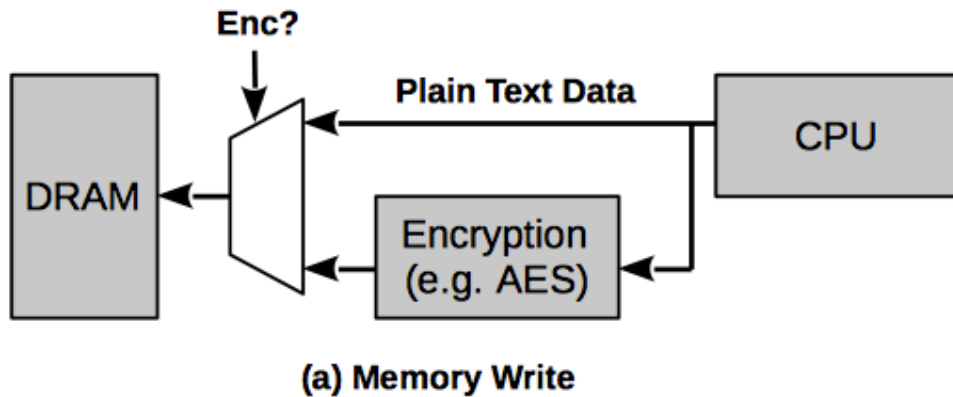
# Confidentiality Protection with Encryption



Contents of the memory can be protected with encryption. Data going out of the CPU is encrypted, data coming from memory is decrypted before being used by CPU.

- a) Encryption engine (usually AES in CTR mode) encrypts data going out of processor chip
- b) Decryption engine decrypts incoming data

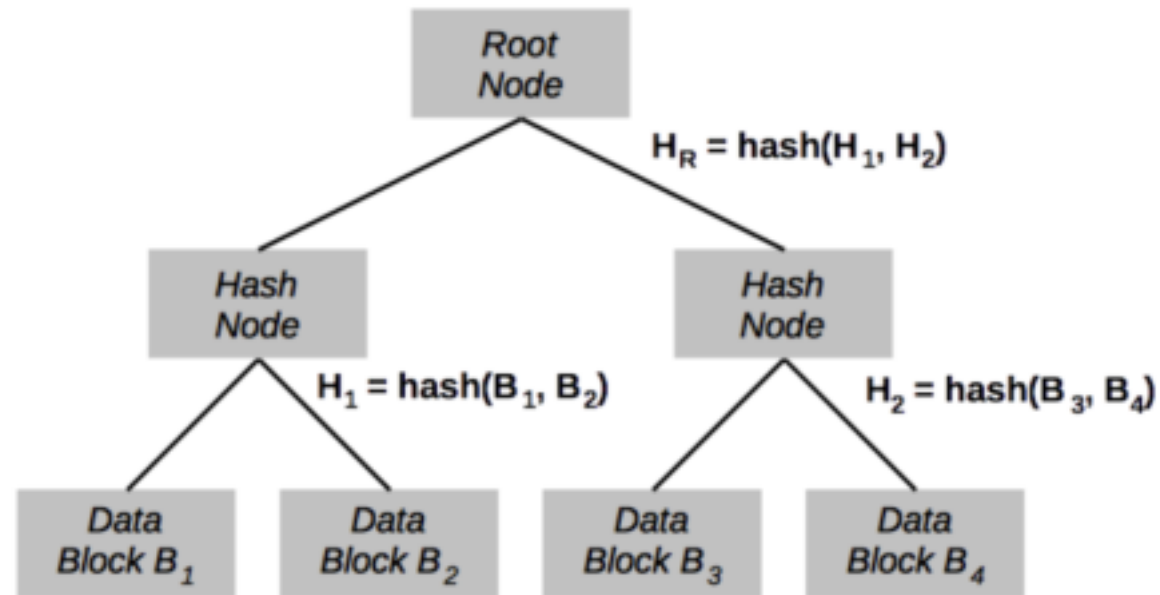
Pre-compute encryption pads, then only need to do XOR; speed depends on how well counters are fetched / predicted.



# Integrity Protection with Hash Trees



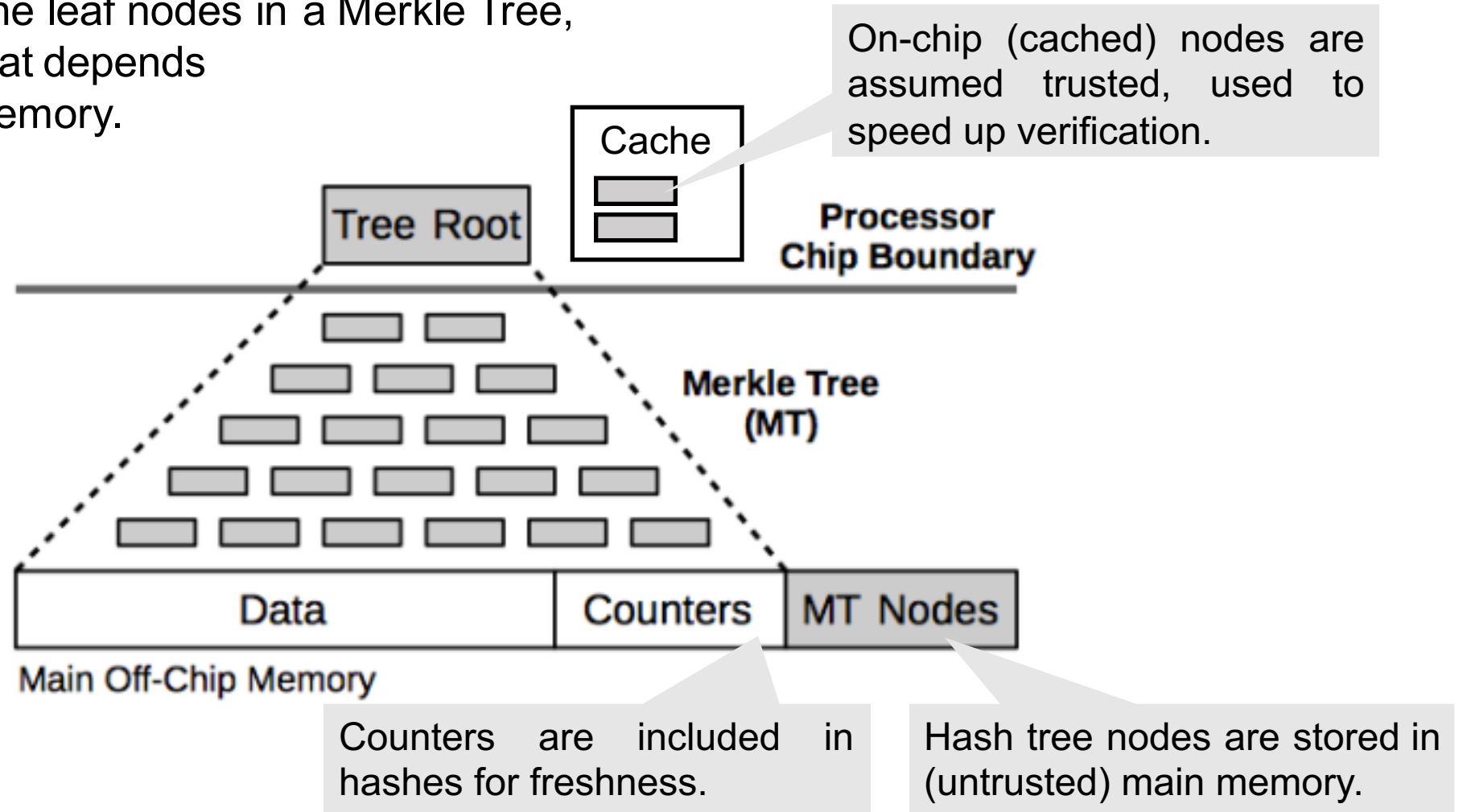
**Hash tree** (also called **Merkle Tree**) is a logical tree structure, typically a binary tree, where two child nodes are hashed together to create parent node; the root node is a hash that depends on value of all the leaf nodes.



# Integrity Protection with Hash Trees



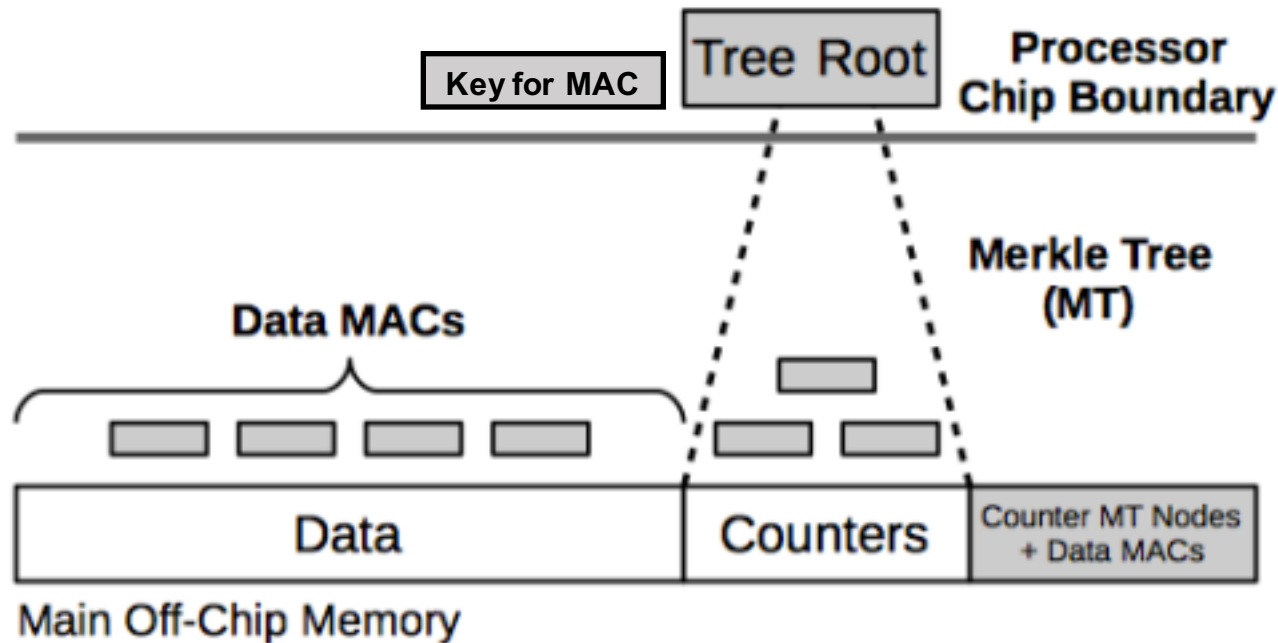
Memory blocks can be the leaf nodes in a Merkle Tree, the tree root is a hash that depends on the contents of the memory.



# Integrity Protection with Bonsai Hash Trees



Message Authentication Codes (MACs) can be used instead of hashes, and a smaller “Bonsai” tree can be constructed.

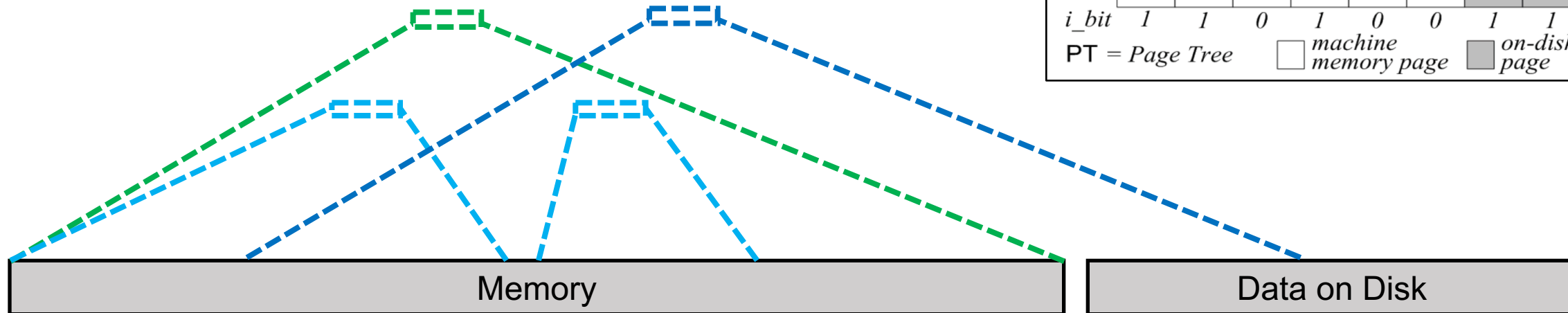
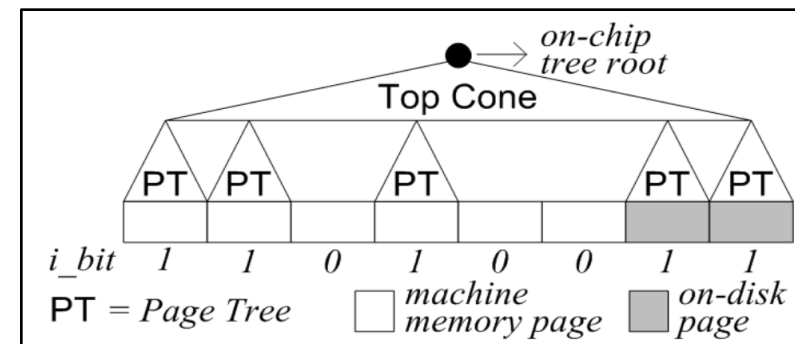


# Integrity Protection of Selected Memory Regions



- For encryption, type of encryption does not typically depend on memory configuration
- For integrity, the integrity tree needs to consider:
  - Protect whole memory
  - Protect parts of memory (e.g. per application, per VM, etc.)
  - Protect external storage (e.g. data swapped to disk)

E.g., Bastion's memory integrity tree (Champagne, et al., HPCA '10)

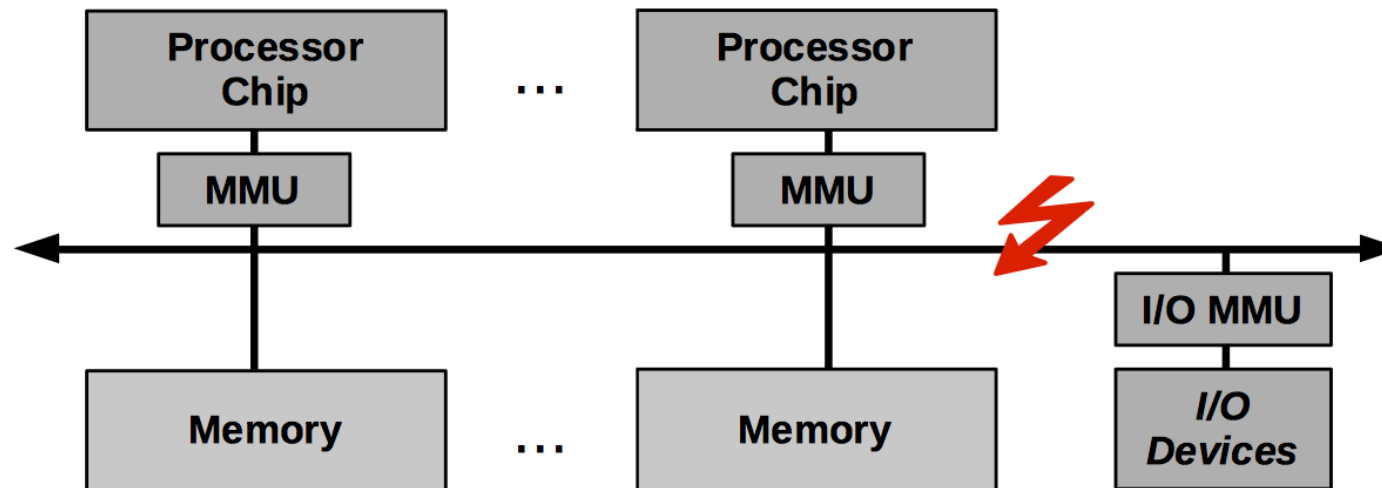


# Memory Access Pattern Protection



Snooping attacks can target extracting data (protected with encryption) or **extracting access patterns** to learn what a program is doing.

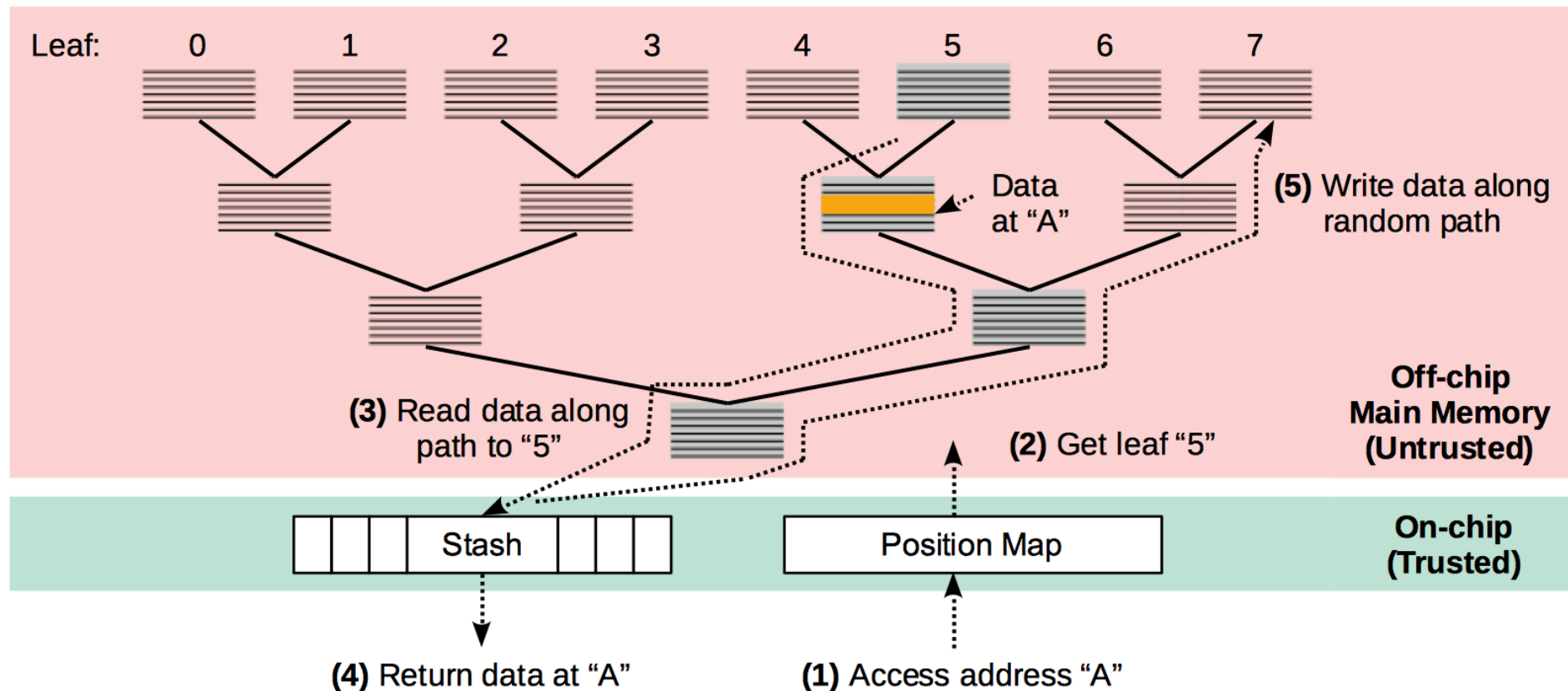
- Easier in Symmetric multiprocessing (SMP) due to shared bus
- Possible in other configuration if there are untrusted components



# Memory Access Pattern Protection



Access patterns (traffic analysis) attacks can be protected with use Oblivious RAM, such as Path ORAM. This is on top of encryption and integrity checking.

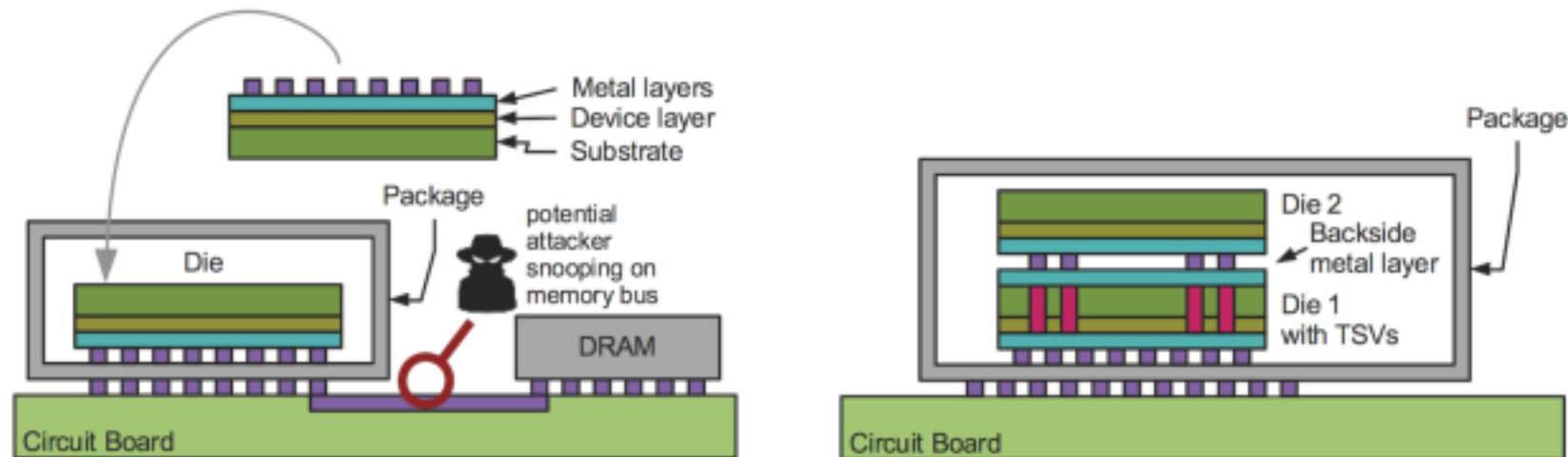


# Leveraging 2.5D and 3D Integration



With 2.5D and 3D integration, the memory is brought into the same package as the main processor chip. Further, with embedded DRAM (eDRAM) the memory is on the same chip.

- Potentially probing attacks are more difficult
- Still limited memory (eDRAM around 128MB in 2017)





# Security of Non-Volatile Memories and NVRAMs



- Non-volatile memories (NVMs) can store data even when there is no power
- Non-volatile random-access memory (NVRAM) is a specific type of NVM that is suitable to serve as a computer system's main memory, and replace or augment DRAM
- Many types of NVRAMs:
  - ReRAM – based on memristors, stores data in resistance of a dielectric material
  - FeRAM – uses ferroelectric material instead of a dielectric material
  - MRAM – uses ferromagnetic materials and stores data in resistance of a storage cell
  - PCM – typically uses chalcogenide glass where different glass phases have different resistances

## Security considerations

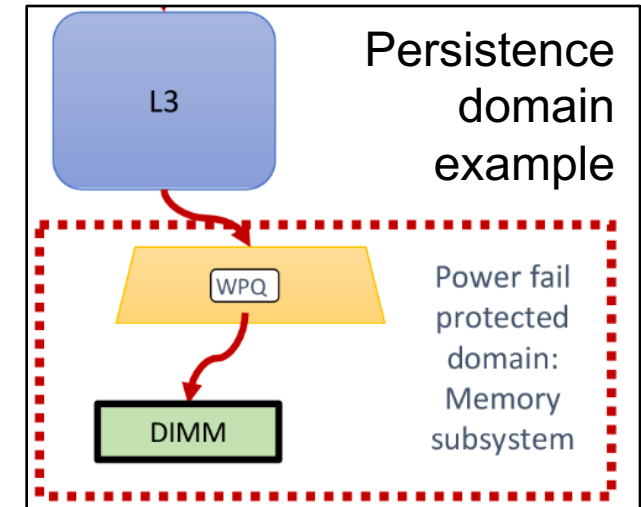
- Data remanence makes passive attacks easier (e.g. data extraction)
- Data is maintained after reboot or crash (security state also needs to be correctly restored after reboot or crash)

# Features of Systems using NVRAMs



## Persistence:

- Data persists across reboots and crashes, possibly with errors
- Need atomicity for data larger than one memory word (either all data or no data is “persisted”)
  - E.g. Write Pending Queue (WPQ) – memory controller has non-volatile storage or enough stored charge to write pending data back to the NV-DIMM or NVRAM



## Granularity of persistence:

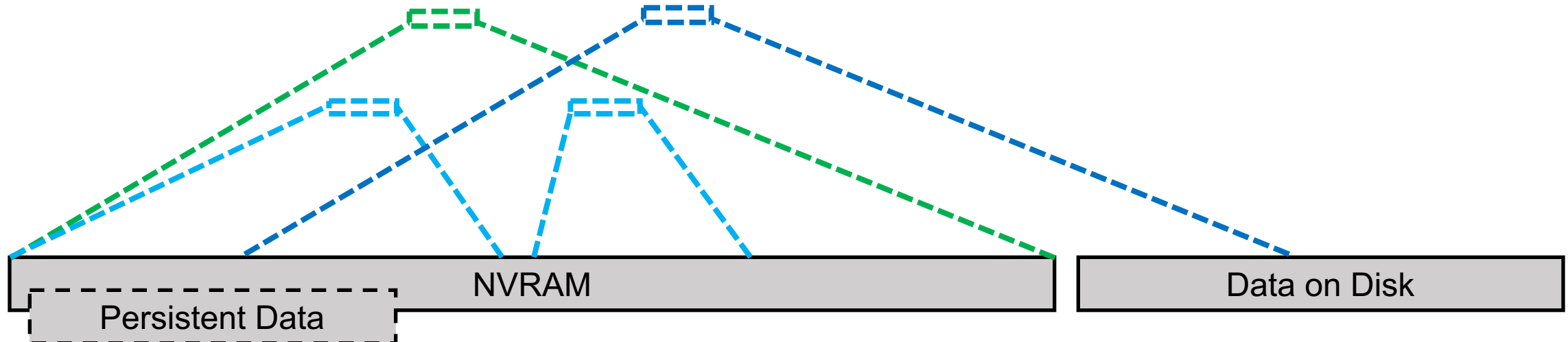
- Hide non-volatility from the system: simply use memory as DRAM replacement
- Expose non-volatility to the system: allow users to select which data is non-volatile
  - Linux support through Direct Access (DAX) since about 2014
  - Developed for NV-DIMMs (e.g., battery backed DRAM, but works for NVRAMs)

Image: [https://www.usenix.org/system/files/login/articles/login\\_summer17\\_07\\_rudoff.pdf](https://www.usenix.org/system/files/login/articles/login_summer17_07_rudoff.pdf)

# Integrity Protection of NVRAMs



- For integrity, the integrity tree needs to additionally consider:
  - Atomicity of memory updates for data and related security state (so it is correct after reboot or a crash)
  - Which data in NVRAM is to be persisted (i.e. granularity)



# Encrypted, Hashed, Oblivious Access Memory Assumption



Off-chip memory is untrusted and the contents is assumed to be protected from the snooping, spoofing, splicing, replay, and disturbance attacks:

- **Encryption** – snooping and spoofing protection
- **Hashing** – spoofing, splicing, replay (counters must be used), and disturbance protection
- **Oblivious Access** – snooping protection



# Multiprocessor and Many-core Secure Processors

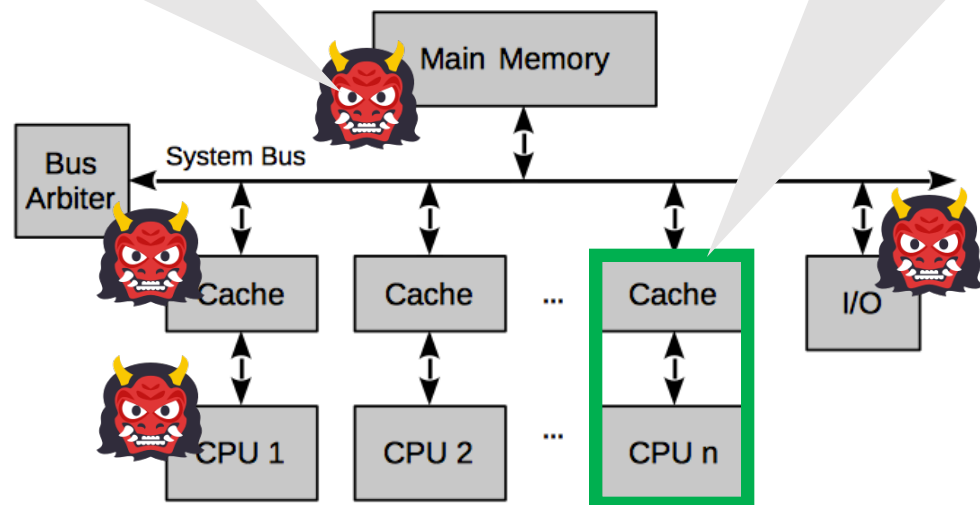
# Multiprocessor Architectures



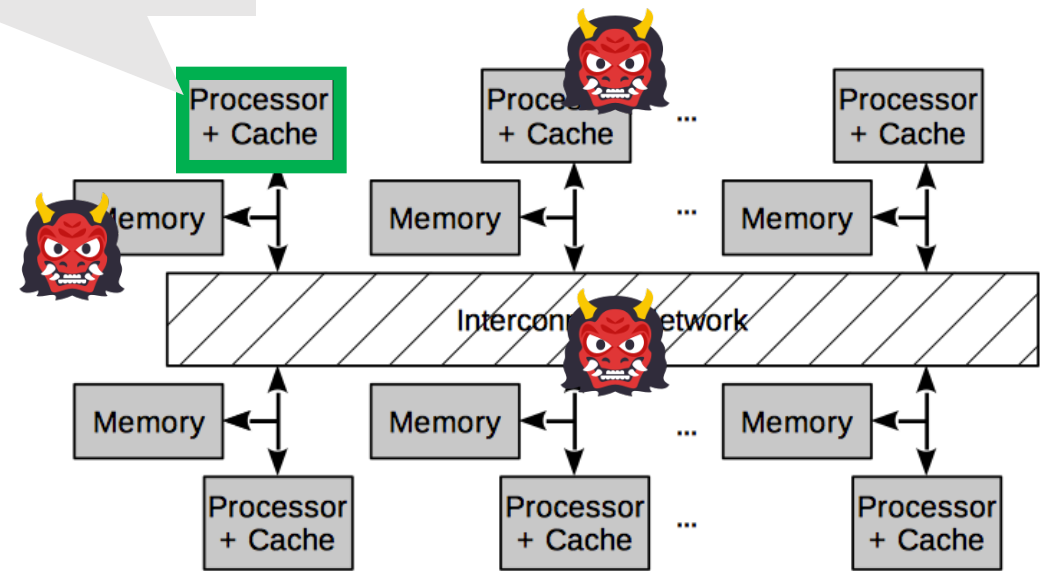
Symmetric Multi Processing (SMP) and Distributed Share Memory (DSM) also referred to as Non-Uniform Memory Access (NUMA) offer two ways of connecting many CPUs together.

Other components on the same system are untrusted

Individual processors are still trusted



SMP



DSM / NUMA

Emoji Image:  
<https://www.emojione.com/emoji/1f479>

# SMP Protections



Encrypt traffic on the bus between processors

- Each source-destination pair can share a hard-coded key
- Or use distribute keys using public key infrastructure (within a computer)

Use MACs for integrity of messages

- Again, each source-destination pair can share a key

Use Merkle trees for memory protection

- Can snoop on the shared memory bus to update the tree root node as other processors are doing memory accesses
- Or per-processor tree

# DSM / NUMA Protections



Encrypt traffic on the bus between processors

- Again need a shared key

Use MACs for integrity of messages

- Again, each source-destination pair can share a key

Use Merkle trees for memory protection

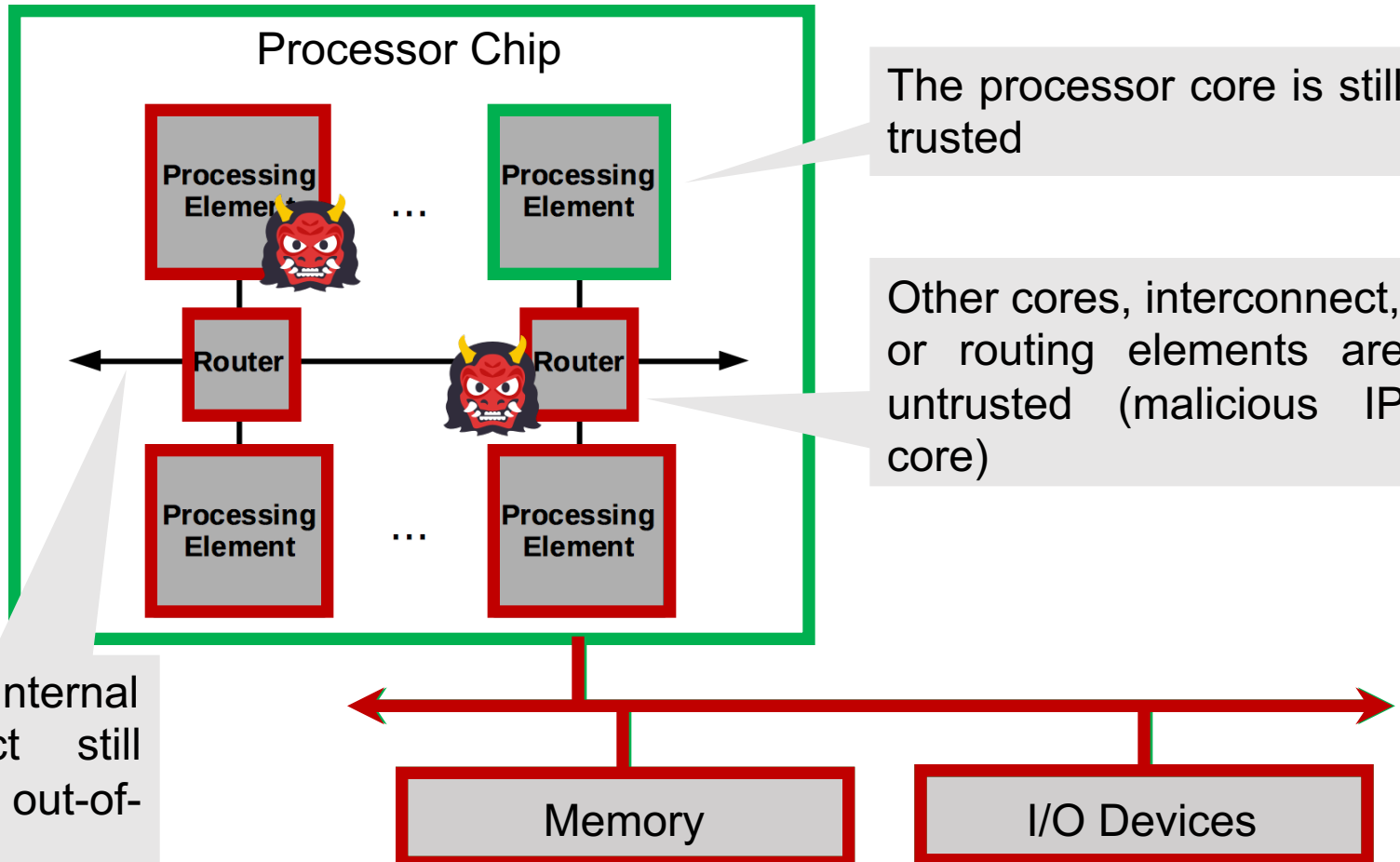
- No-longer can snoop on the traffic (DSM is point to point usually)



# Many-core Trust Boundary



Trusted processor chip boundary is reduced in most research focusing on many-core security



The processor core is still trusted

Other cores, interconnect, or routing elements are untrusted (malicious IP core)

Probing of internal interconnect still assumed out-of-scope.

Emoji Image:  
<https://www.emojione.com/emoji/1f479>

# Architecture and Hardware Security Intersection



With many-core chips, the threats architects worry about start to overlap with hardware security researchers' work

- Untrusted 3<sup>rd</sup> party intellectual property (IP) cores
- Malicious foundry
- Untrusted supply chain

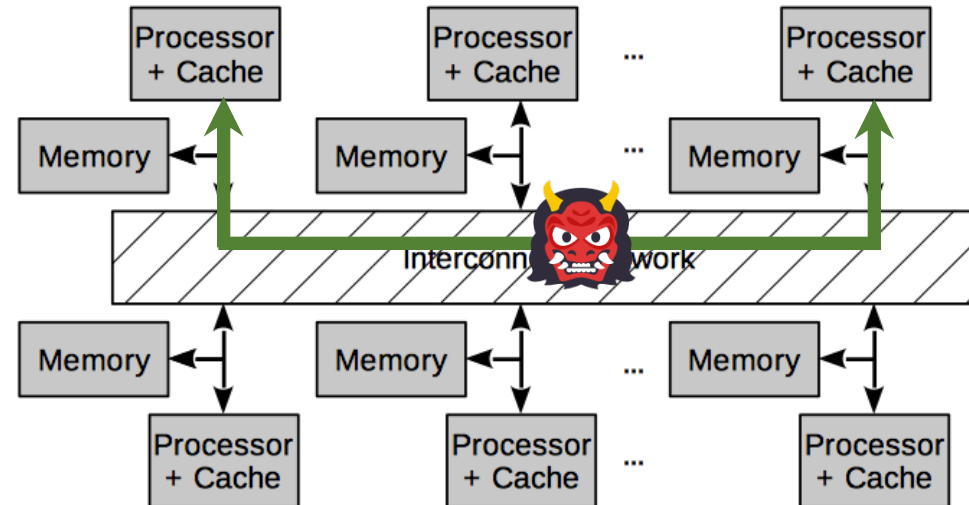
Architecture solutions (add encryption, add hashing, etc.) complement defenses developed by hardware security experts (split manufacturing, etc.).

# Protected Inter-processor Communication



In addition to the existing assumption about protected memory communication, designs with multiple processors or cores assume the inter-processor communication will be protected:

- Confidentiality
- Integrity
- Communication pattern protection



Emoji Image:  
<https://www.emojione.com/emoji/1f479>

# Performance Challenges



Interconnects between processors are very fast:

- E.g. HyperTransport specifies speeds in excess of 50 GB/s
  - AES block size is 128 bits
  - Encryption would need 3 billion (giga) AES block encryptions or decryptions per second
- Tricks such as counter mode encryption can help
  - Only XOR data with a pad
  - But need to have or predict counters and generate the pads in time



# Designing Secure Processors

# Principles of Secure Processor Architecture Design



Four principles for secure processor architecture design based on existing designs and also on ideas about what ideal design should look like are:

- 1. Protect Off-chip Communication and Memory**
- 2. Isolate Processor State among TEE Execution and other Software**
- 3. Allow TCB Introspection**
- 4. Authenticate and Continuously Monitor TEE and TCB**

- Architectural state
- Micro-architectural state
- Due to spatial or temporal sharing of hardware

Additional design suggestions:

- Avoid code bloat
- Minimize TCB
- Ensure hardware security (Trojan prevention, supply chain issues, etc.)
- Use formal verification

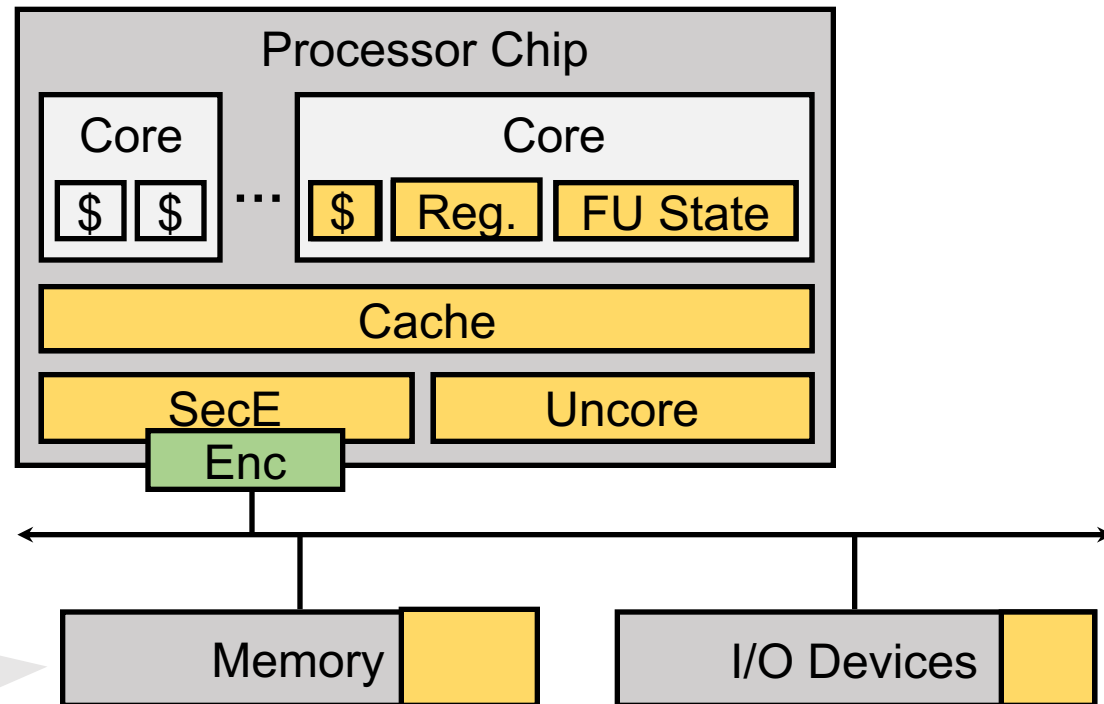
# Protect Off-chip Communication and Memory



Off-chip components and communication are untrusted, need protection with **encryption**, **hashing**, **access pattern protection**.

Open research challenges:

- Performance
- Key distribution



E.g. encryption defends Cold boot style attacks on main memory.

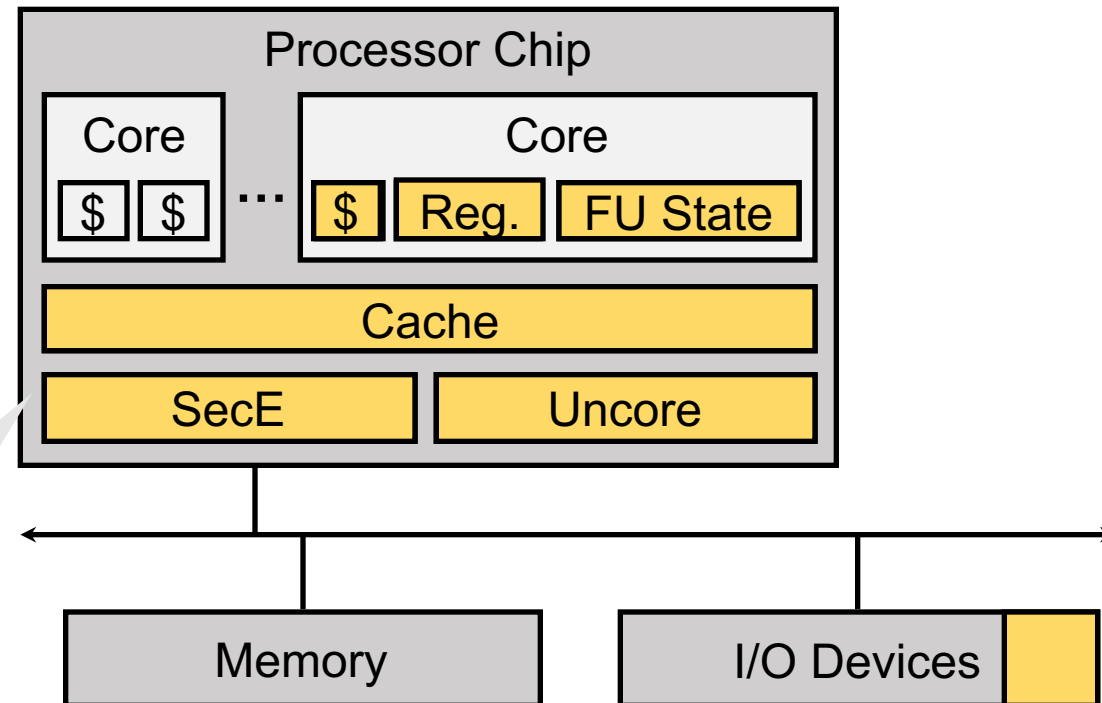
# Isolate Processor State among TEE Execution



When switching among protected software and other software or other protected software, need to flush the state, or save and restore it, to prevent one software influencing another.

Open research challenges:

- Performance
- Finding all the state to flush or clean
- Isolate state during concurrent execution
- ISA interface to allow state flushing



E.g. flushing state helps defend Spectre and Meltdown type attacks.



# Allow TCB Introspection

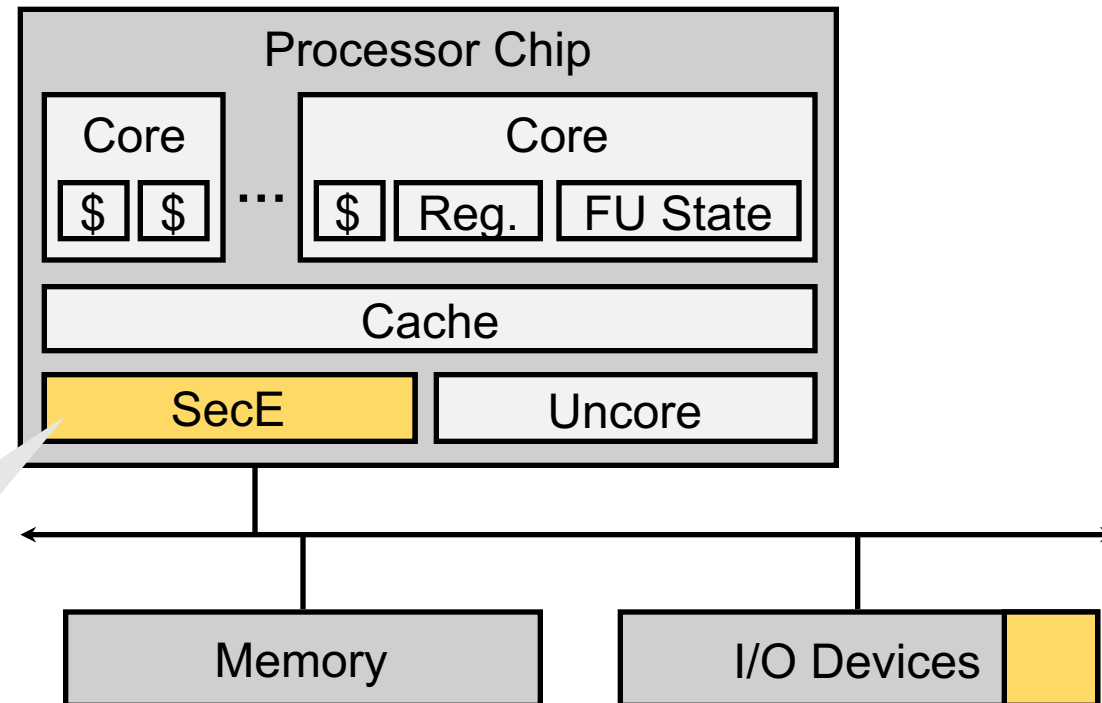


Need to ensure correct execution of TCB, through **open access to TCB design, monitoring, fingerprinting, and authentication.**

Open research challenges:

- ISA interface to introspect TCB
- Area, energy, performance costs due extra features for introspection
- Leaking information about TCB or TEE

E.g. open TCB design can minimize attacks on ME or PSP security engines



# Authenticate and Continuously Monitor TEE and TCB



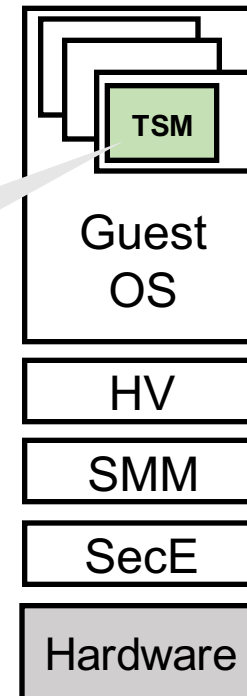
Monitoring of software running inside TEE, e.g. TSMs or Enclaves, gives assurances about the state of the protected software.

Likewise monitoring TCB ensures protections are still in place.

Open research challenges:

- Interface design for monitoring
- Leaking information about TEE

E.g. continuous monitoring of a TEE can help prevent TOC-TOU attacks.



# Pitfalls and Fallacies



- Pitfall: Security by Obscurity 

E.g. recent attacks on industry processors.
- Fallacy: Hardware Is Immutable 

Most actually realized architectures use a security processor (e.g. ME or PSP).
- Pitfall: Wrong Threat Model 

E.g. original SGX did not claim side channel protection, but researchers attacked it.
- Pitfall: Fixed Threat Model 

Most designs are one-size-fits all solutions.
- Pitfall: Use of Outdated or Custom Crypto 

E.g. today's devices will be in the field for many years, but do not use post-quantum crypto.
- Pitfall: Not Addressing Side Channels 

Most architectures underestimate side channels.
- Pitfall: Requiring Zero-Overhead Security 

Performance-, area-, or energy-only focused designs ignore security.
- Pitfall: Code Bloat 

E.g. rather than partition a problem, large code pieces are ran instead TEEs; also TCB gets bigger and bigger leading to bugs.
- Pitfall: Incorrect Abstraction 

Abstraction (e.g. ISA assumptions) does not match how device or hardware really behaves.

# Pitfalls and Fallacies



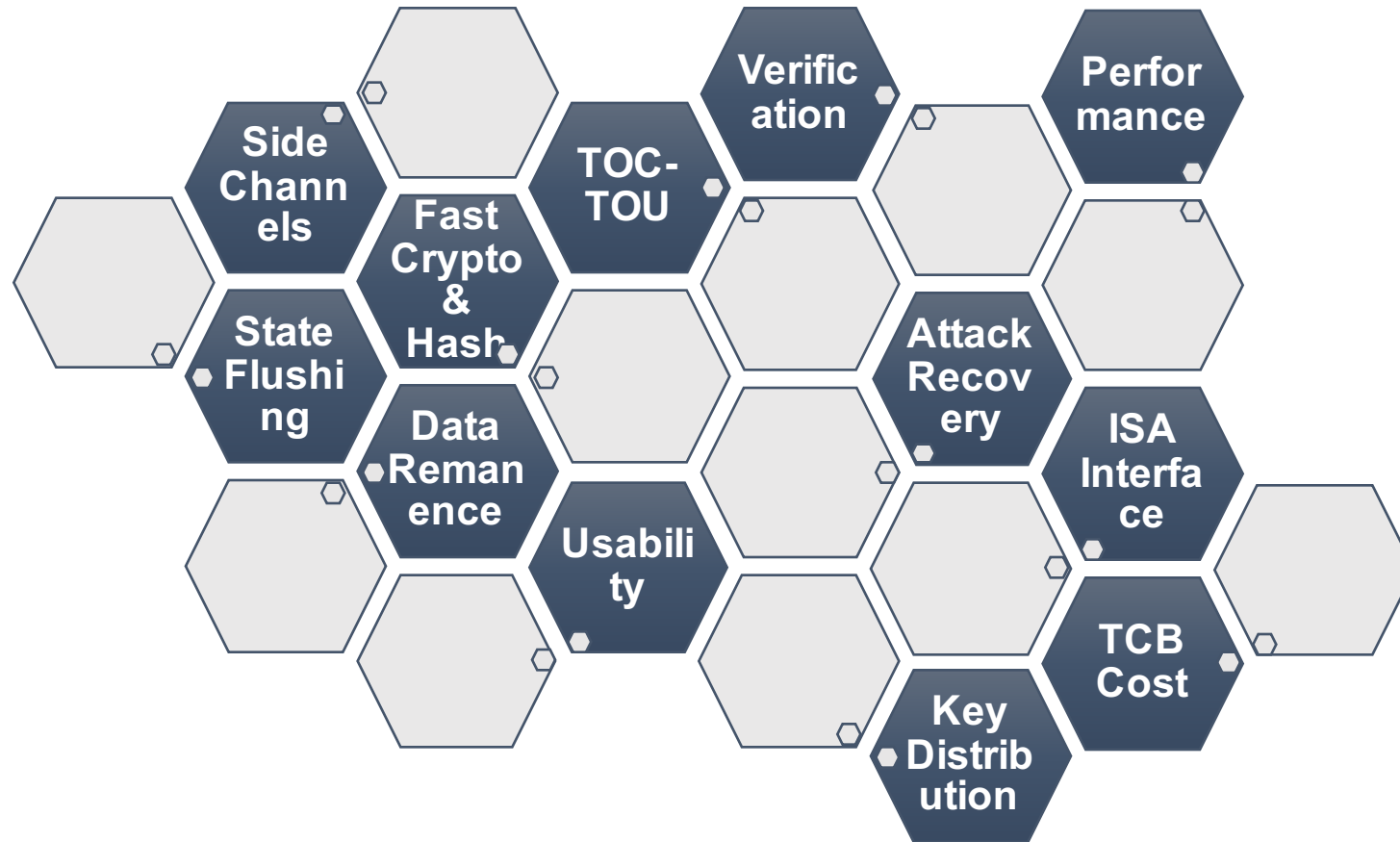
- Pitfall: Focus Only on Speculative Attacks
- ...

Defending only speculative attacks does not ensure classical attacks are also protected

# Challenges in Secure Processor Design



A number of challenges remain in research on secure processor designs:



# Thank You!



## Related reading...

*Jakub Szefer, "Principles of Secure Processor Architecture Design," in Synthesis Lectures on Computer Architecture, Morgan & Claypool Publishers, October 2018.*

<https://caslab.csl.yale.edu/books/>

