Scalable Architectural Support for Trusted Software

David Champagne and Ruby B. Lee
Princeton University

Secure Processor Design
11/02/2017
Dimitrios Skarlatos

Motivation

- Apps handle sensitive/secret information
  - Healthcare -> medical records
  - Finance -> $$
- The common case:
  - Small part of apps require extra protection
  - Security-critical tasks:
    - Trusted Software modules

The Problem

- App-level security can be undermined by OS
  - Full control over applications
  - Too complex to verify
    - Highly vulnerable
- Hardware/Physical attacks
  - Mobile devices can get stollen
  - Probe physical memory

The Problem (of previous work)

- Software approaches are vulnerable
  - Compromised OS
  - Hardware attacks
- Trusted Platform Module (TPM)
  - Only software attacks
- Building security into OS is tough
  - Applications writers
  - Hardware vendors

The Solution so Far

- Virtualization through VMM or Hypervisor
- Create isolated Virtual Machines (VM)
- Current VM solutions are coarse-grain
  - Each VM is one component
  - High VM switch cost

Proposal: Bastion

- A hw-sw arch for protecting security critical tasks
- Provide secure compartments within the VM
  - Avoid VM switches
- Hardware protection for VMM
  - Software and Hardware attacks (off-chip)
  - Secure Launch, Protected virtual memory
  - Secure module storage, Inter module control flow
- FPGA prototype based on OpenSPARC

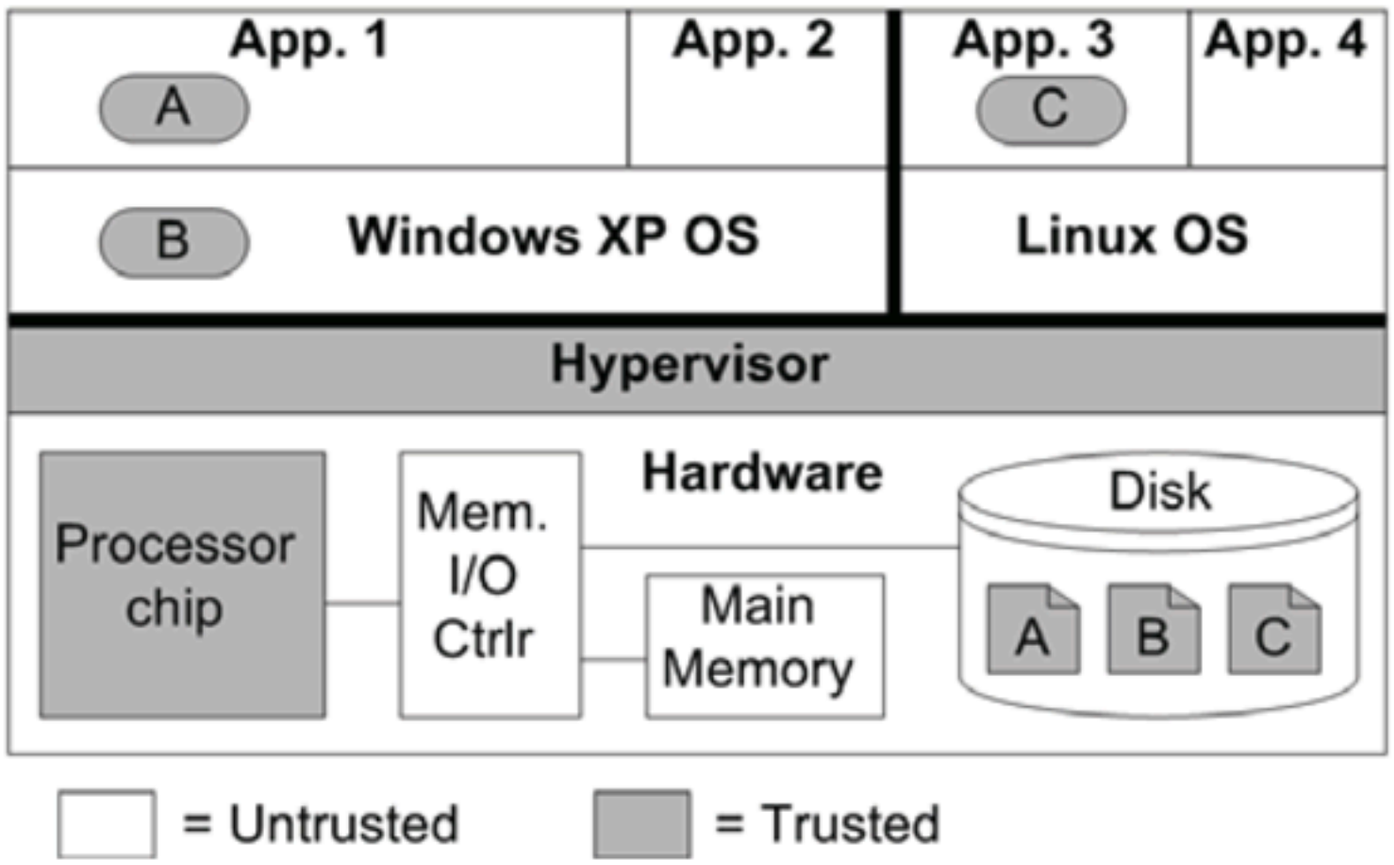
Threat Model

- Hardware:
  - Security perimeter of the chip
  - Memory, buses, disk can get compromised
- Software:
  - OS can be compromised
  - Everything can be snooped (reg, mem, $$)
  - Data integrity can be compromised
- Side-channels and DoS attacks are not covered

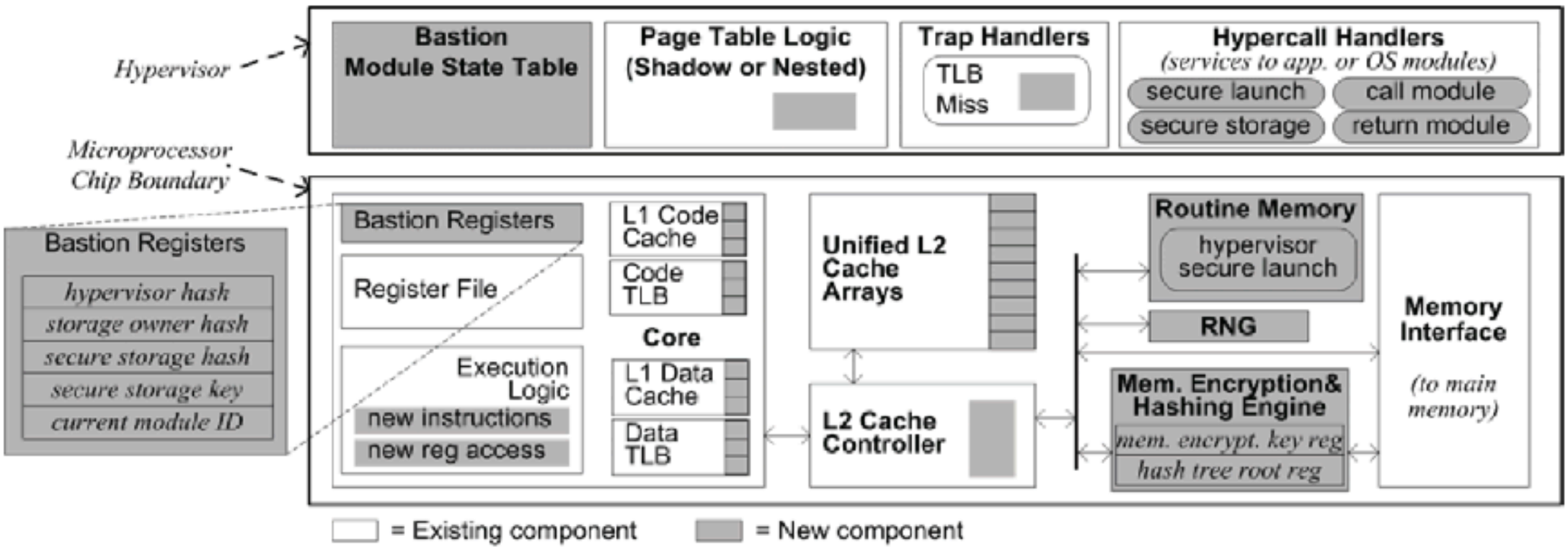
Bastion Overview

- Hypervisor based on HW protection
  - Storage and runtime
  - Supports arbitrary number of trusted SW mods
- Each SW module provided with:
  - Secure execution and storage area
- Baseline HW has VT support:
  - HW ensures that software cannot access VMM code/data

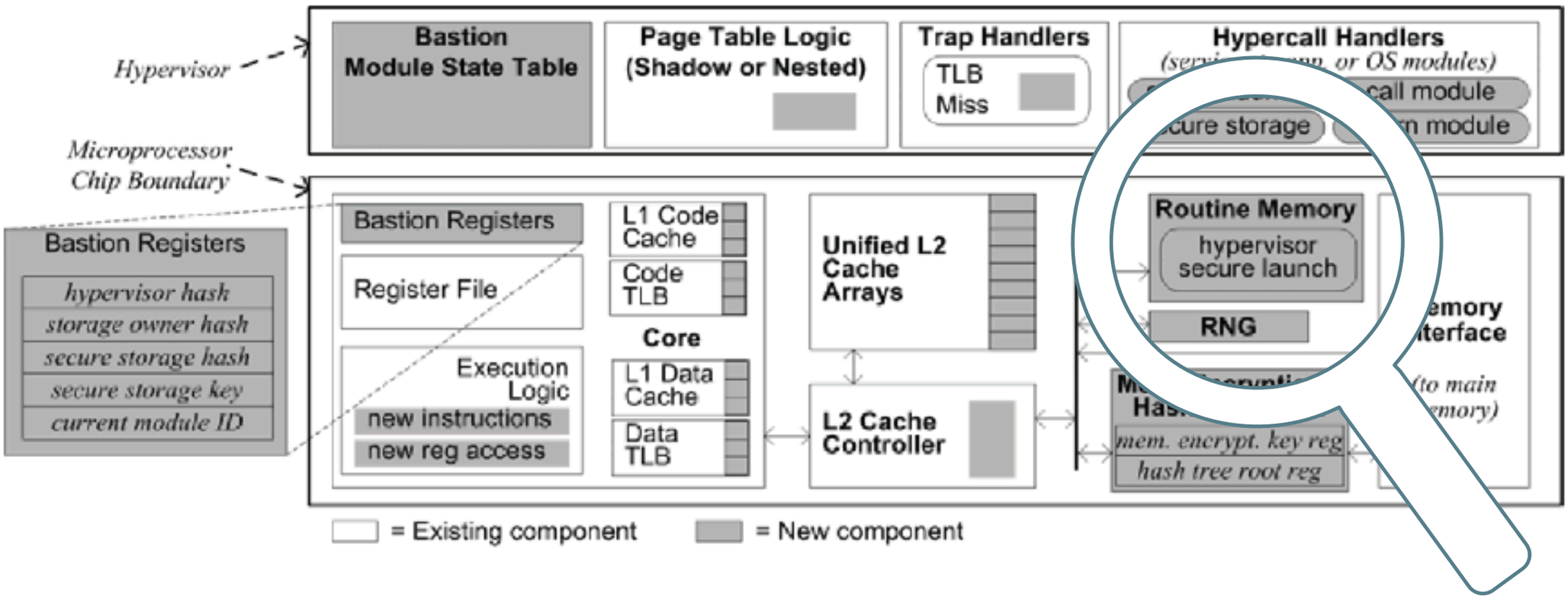
Bastion for Trusted Apps A, B, C



Bastion Architecture



Bastion Architecture - Bootup



Secure Launch

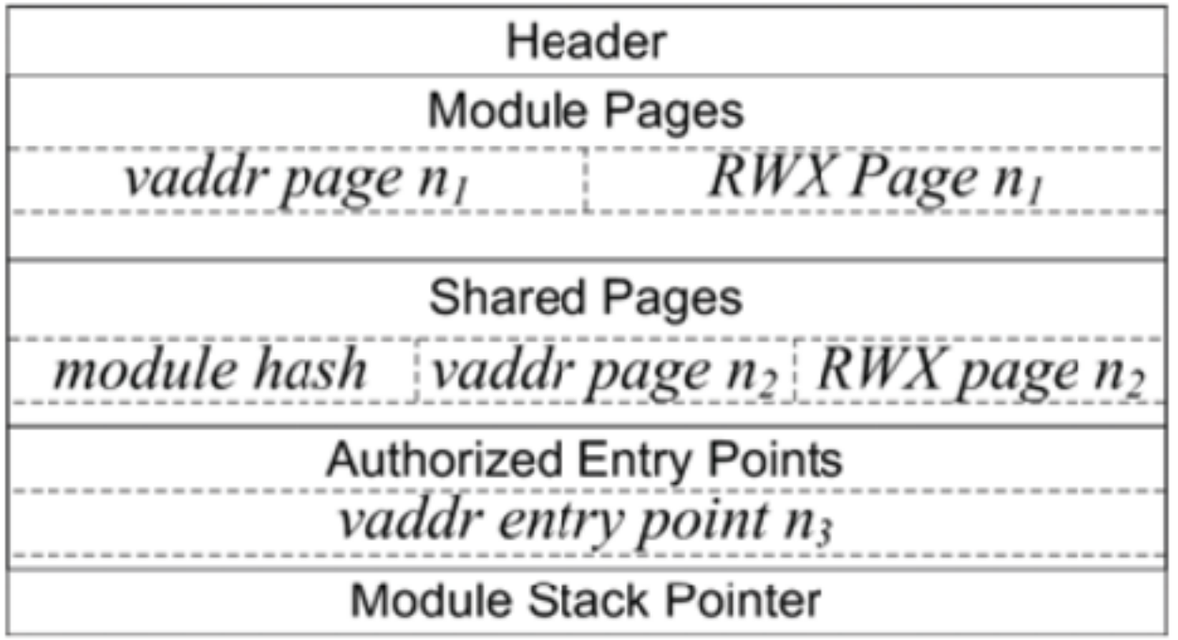
- Basic BIOS boot process
- Load hypervisor binary into memory
- secure_launch gets invoked:
  - crypto hash over the state of VMM
  - store hash into hypervisor_hash reg
- If secure_launch is skipped or corrupted
  - Secure storage remains locked -> tied to hash

Secure Launch

- secure_launch generates new key for each power-on
  - Used to protect VMM memory at runtime
- VMM data:
  - automatically encrypted
  - on-chip crypto engine
  - cache lines are hashed -> fingerprinting

Trusted Software Modules

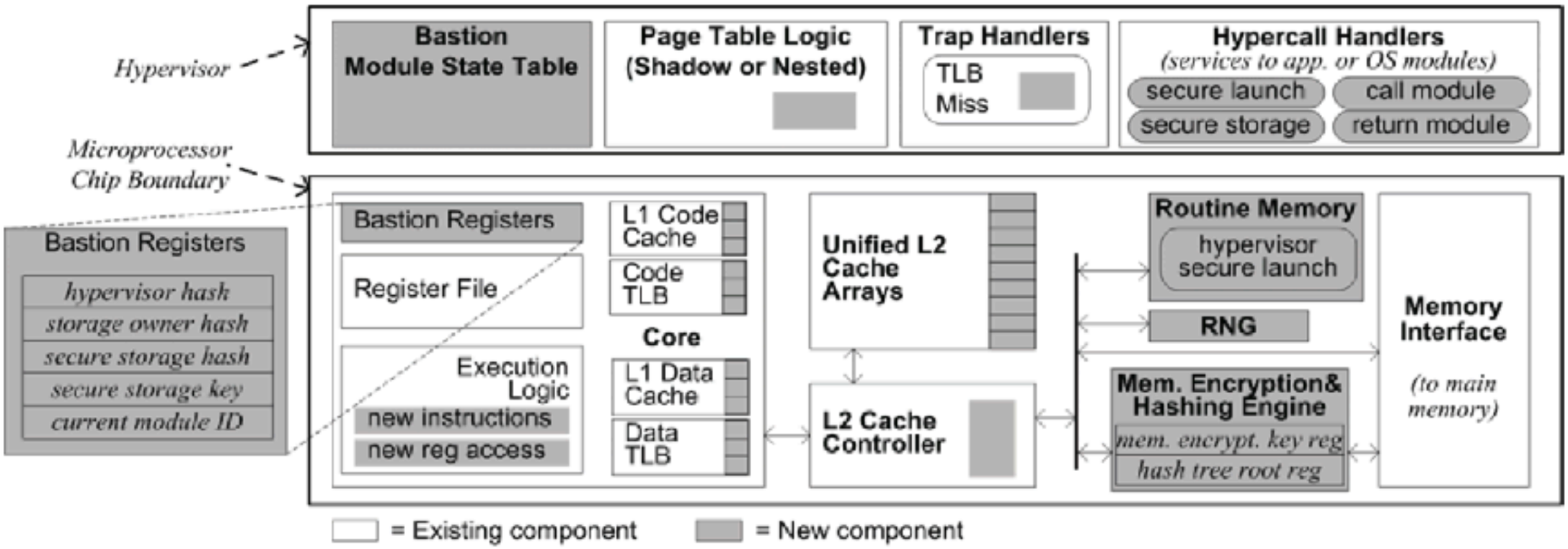
- VMs follow the normal setup procedure
  - Bootup sequence, memory allocation
- secure_launch hyper call
  - Ptr to the security segment
  - private code and data pages
  - Access rules to those pages
  - Authorized entry points
  - Computes the module_identity hash



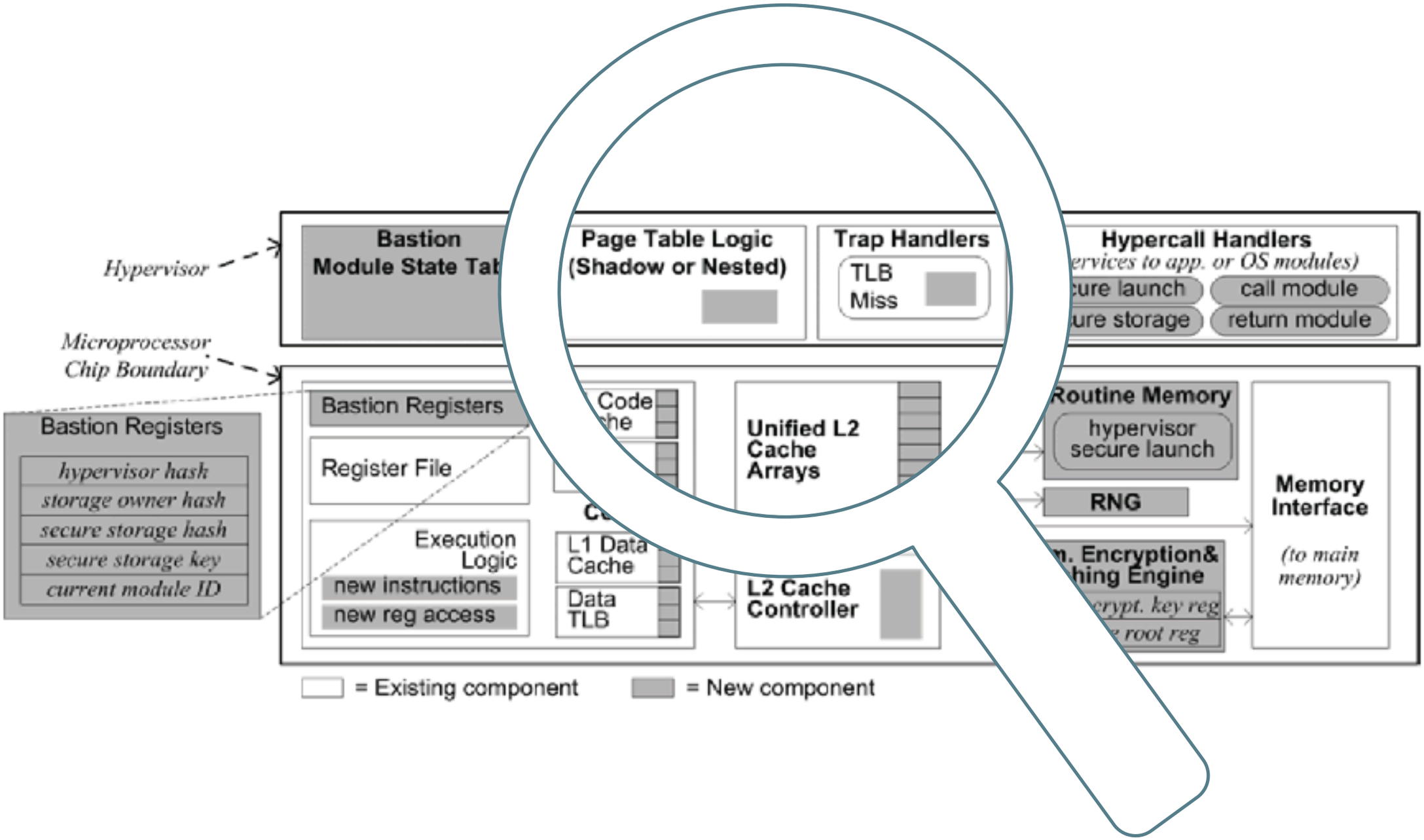
Shadow Access Control

- Hypervisor utilizes the security segment
  - No aliasing between modules' private pages
  - Modules sharing memory must agree
- Module State Table (in the VMM)
  - Mapping each machine page to a set of modules
  - Identify modules through module_id (8-20 bits)

Bastion Architecture - Virtual Memory



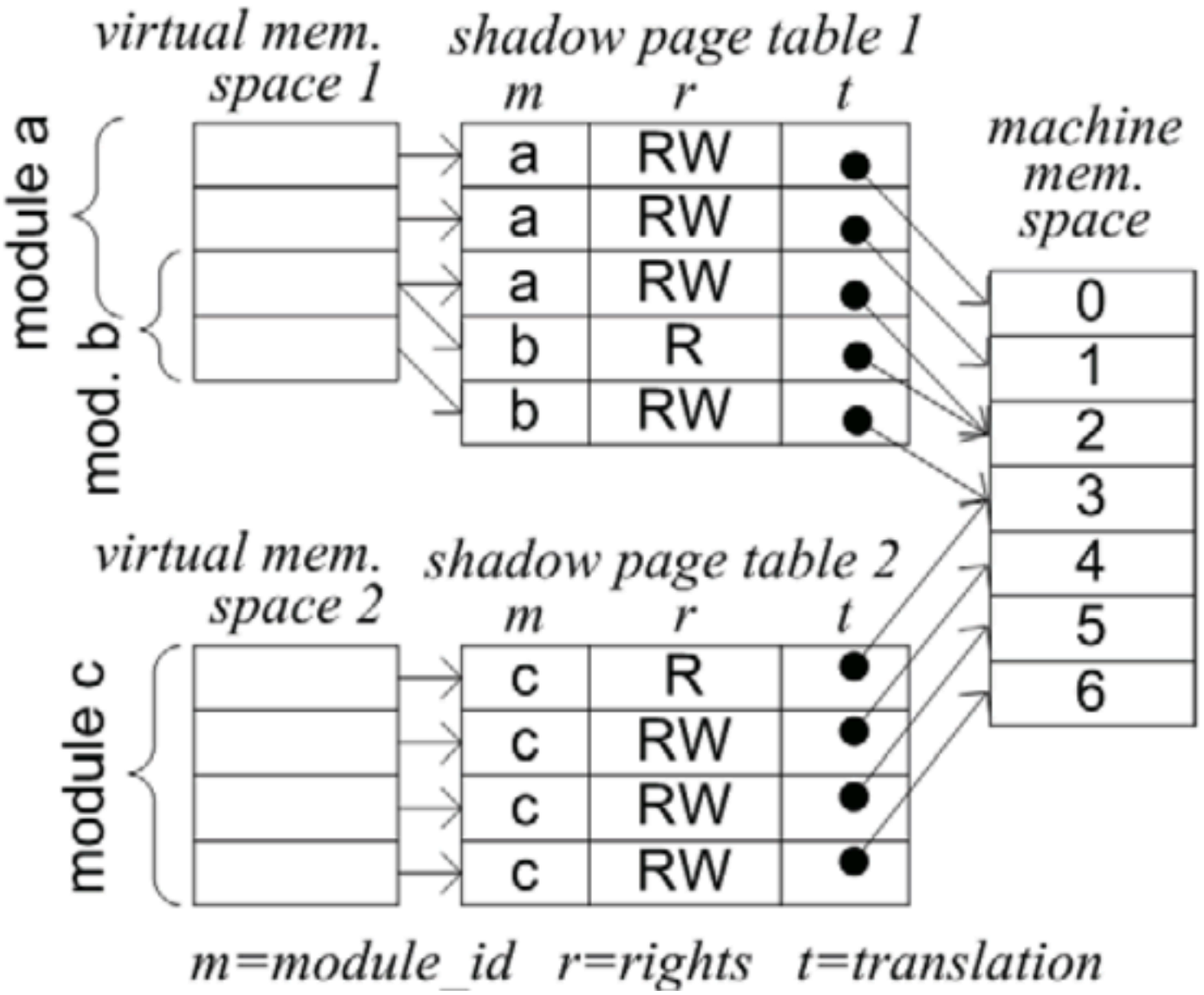
Bastion Architecture - Virtual Memory



Virtual Memory Enforcement

- Hardware enforces Shadow Access Control Rules
- Rules are embedded in I,D-TLBs
- TLB entries are extended with module_id
- Infringement triggers TLB handler, invokes VMM
- Hypervisor manages module_id placement
  - Shadow page tables get extended
  - Replicate entries for shared pages :(

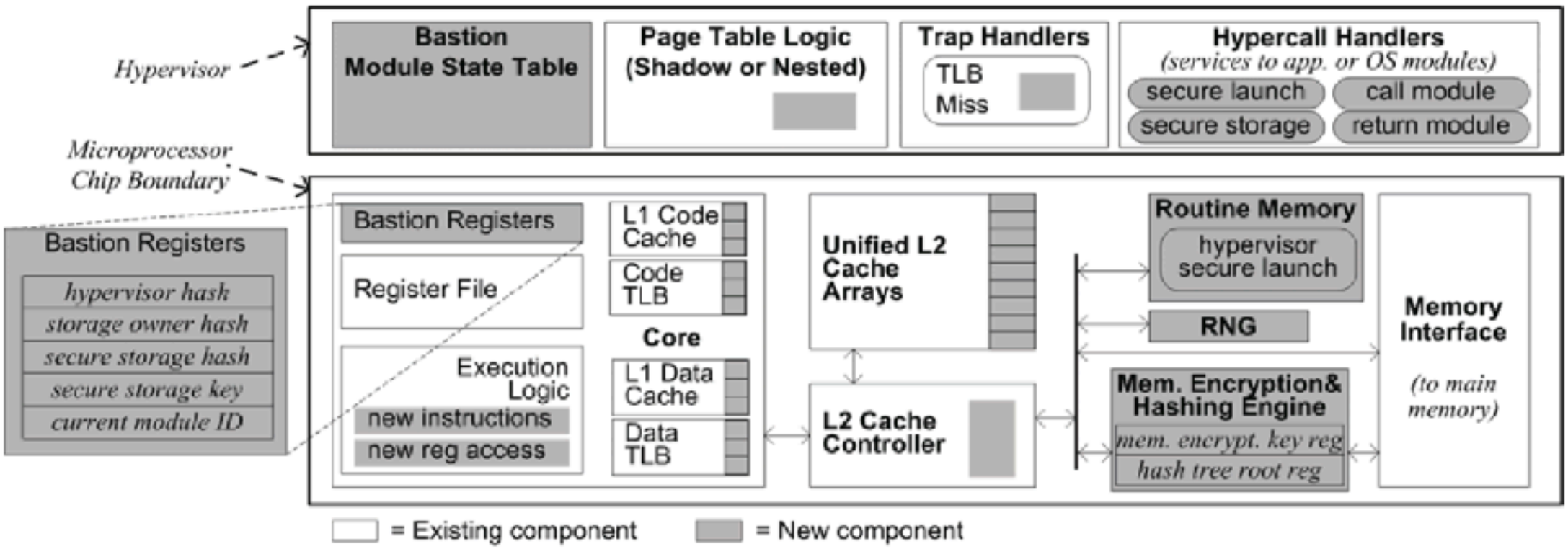
Shadow Access Control + Shadow Page Tables



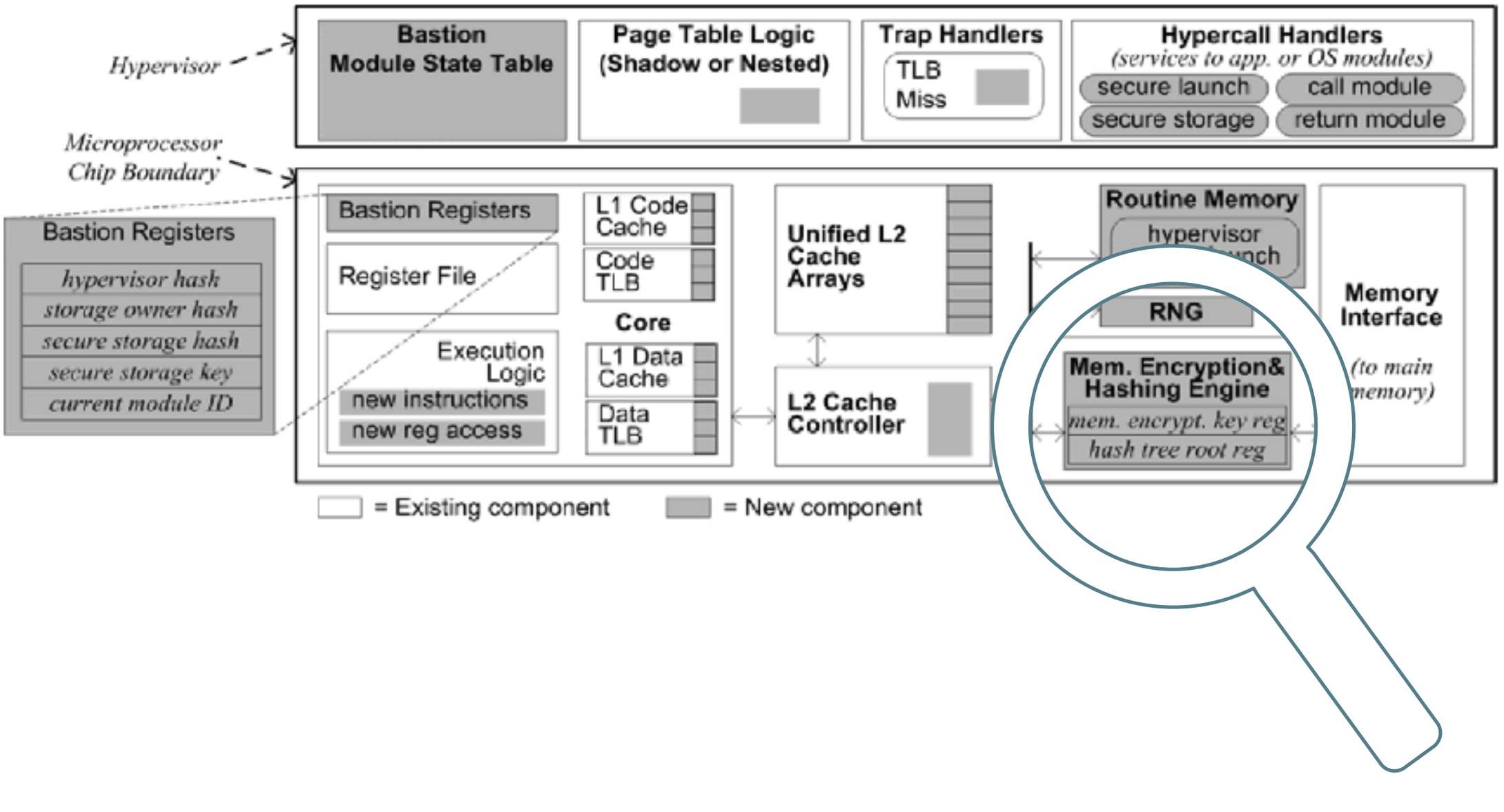
Nested Page Tables

- Only track guest-physical-to-machine mappings
  - Instead of copies for all guest-virtual-to-machine
- On a miss:
  - Construct it on-the-fly
  - Invoke VMM to add module_id!! :(
- Restrict or deny a request to access data
  - Guest OS gets “restricted” access to perform mapping?

Bastion Architecture - Main Memory



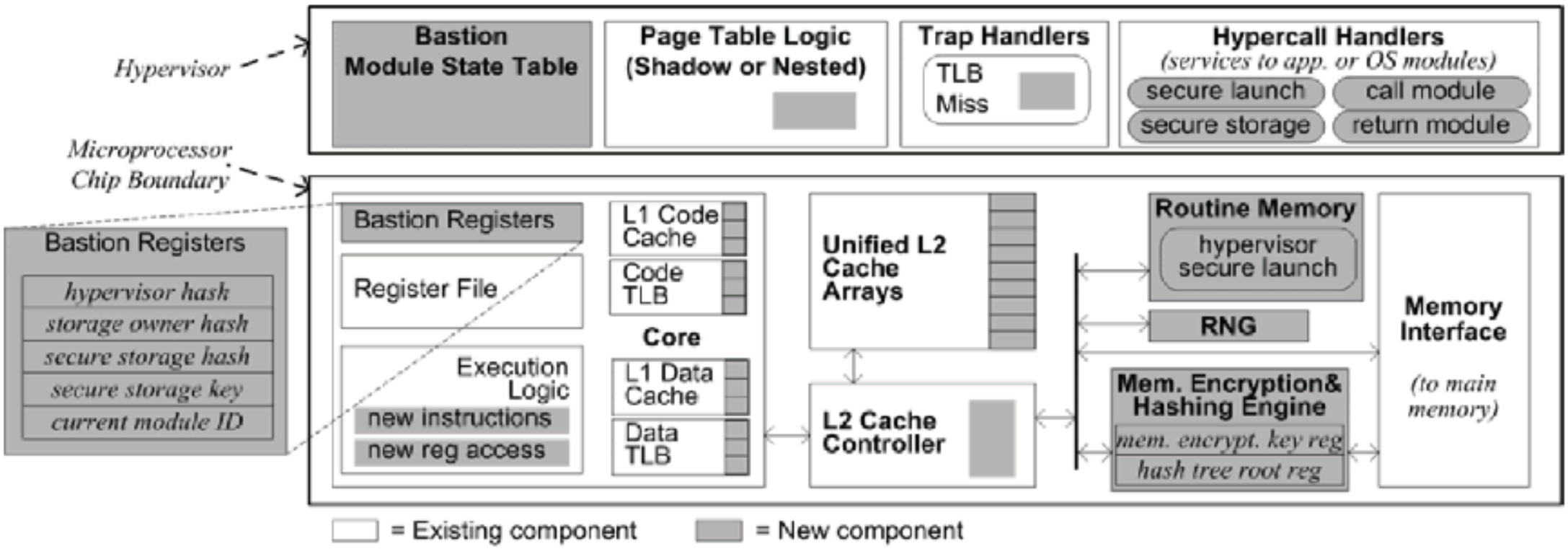
Bastion Architecture - Main Memory



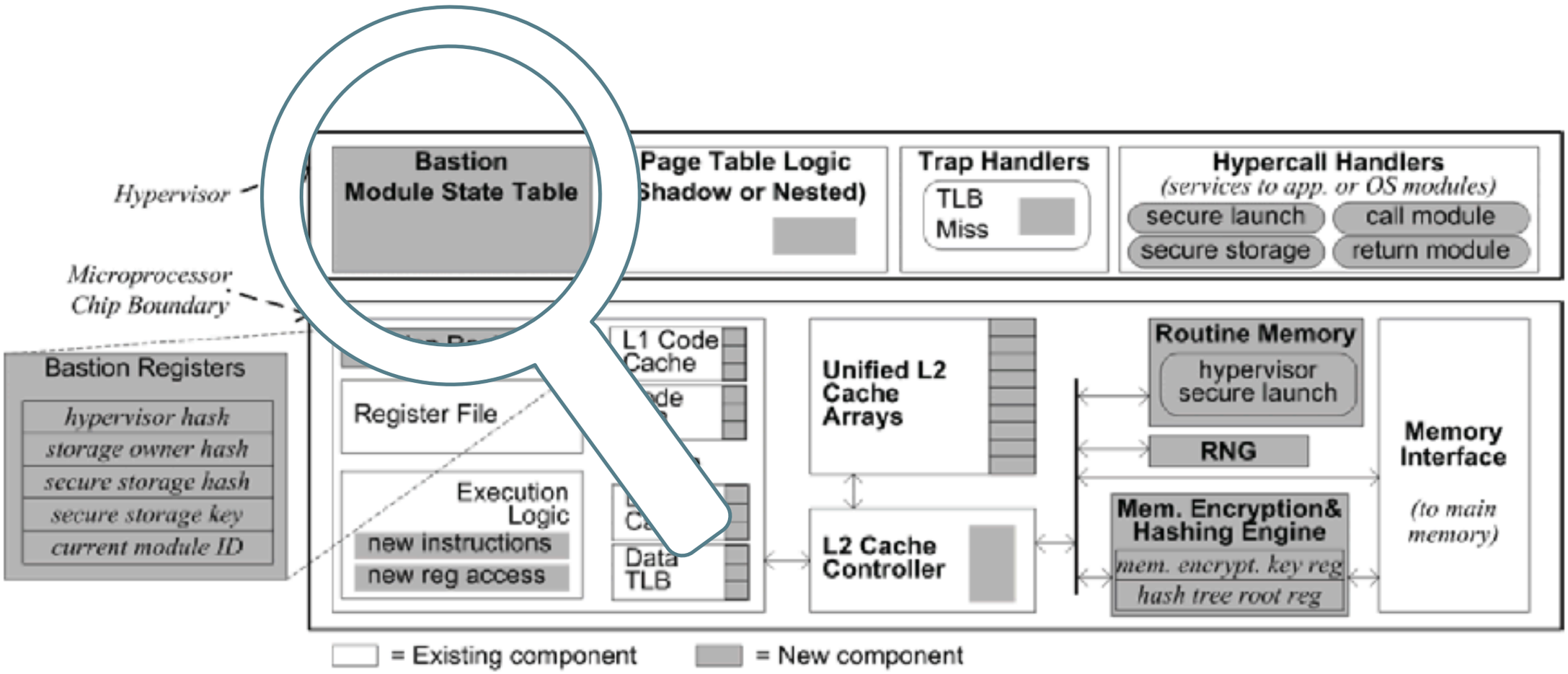
Secure Physical Memory

- Hardware engines
  - Encrypts and decrypts memory traffic
  - Verifies integrity through hash tree
- Hash tree is based on Merkle tree (only VMM, Trusted Mod)
  - TLB entries get extended with i_bit and c_bit
  - L2 cache lines replicate the bits
  - Trigger encryption and hashing

Bastion Architecture - Control Flow



Bastion Architecture - Control Flow



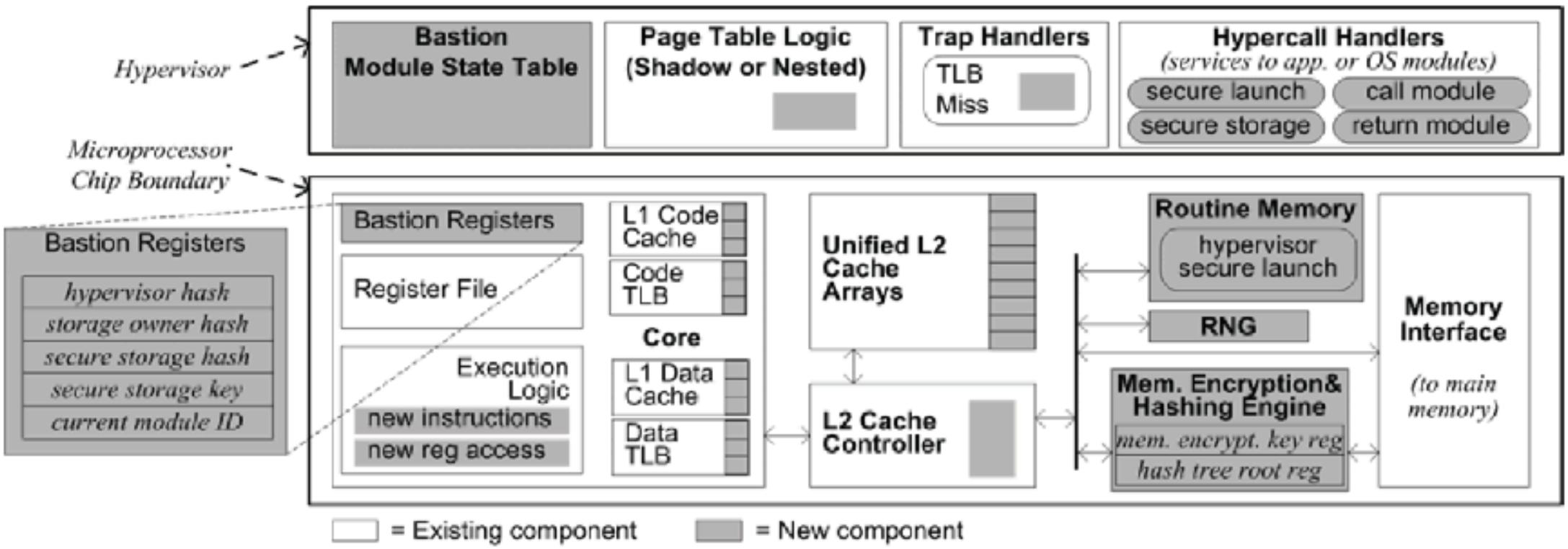
Inter-Module Control Flow

- All VMs begin with module_id set to 0
- Call_module hypercall
  - Input: module_id VA of entry point
- Entry point get verified against list of authorized
- Module State Table gets update
- current_module_id changes on TLB miss by VMM
- Same for return: VMM intervenes

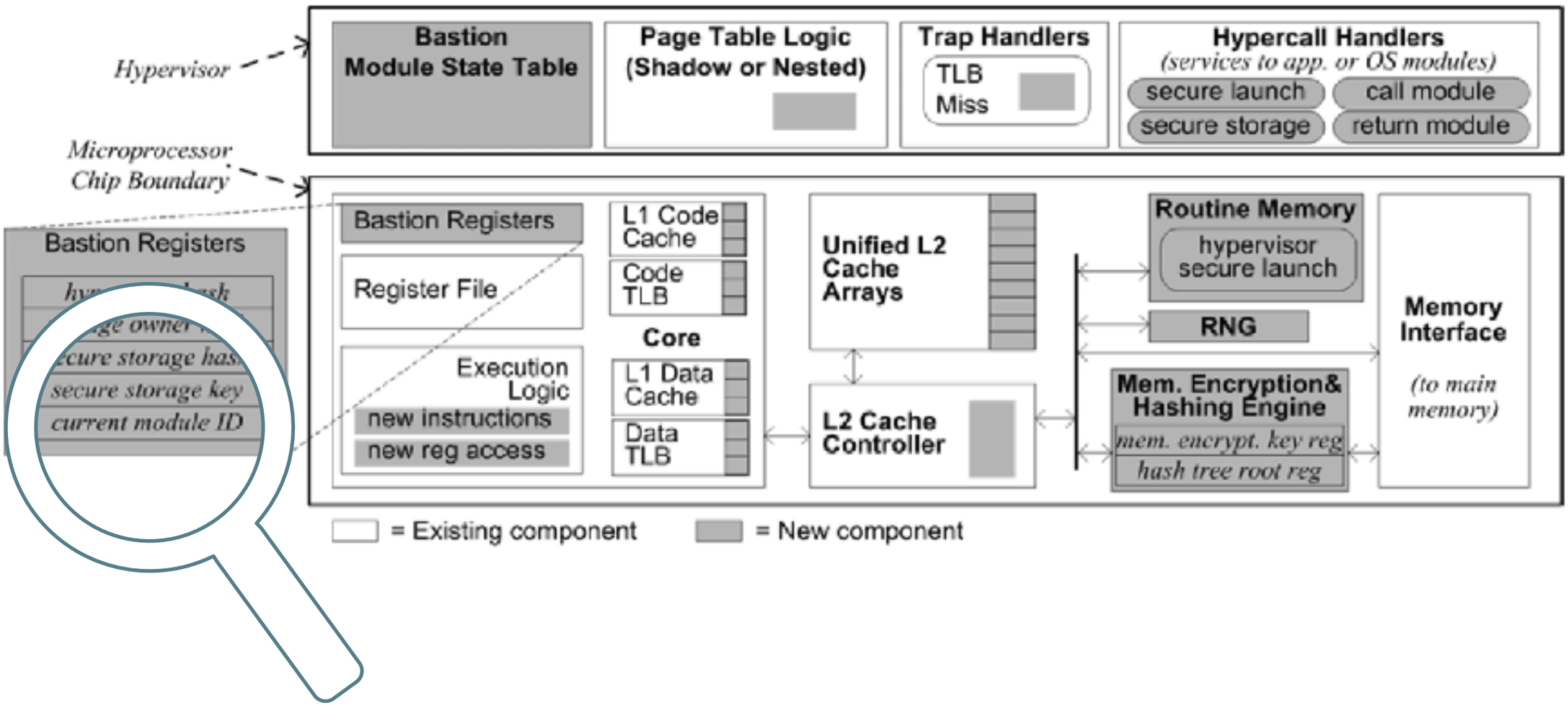
Module Preemption

- When a protected module gets preempted
  - VMM intervenes and saves register state
  - Malicious OS cannot observe or modify regs
- VMM intervention is guaranteed
  - TLB miss on module_id mismatch

Bastion Architecture - Storage



Bastion Architecture - Storage



Secure Storage

- secure_storage_key and secure_storage_hash
  - Key for encryption
  - Hash for fingerprinting
- The key, hash pair is stored on VMM secure storage
  - Protected by 2 HW registers on Bastion

Secure Storage

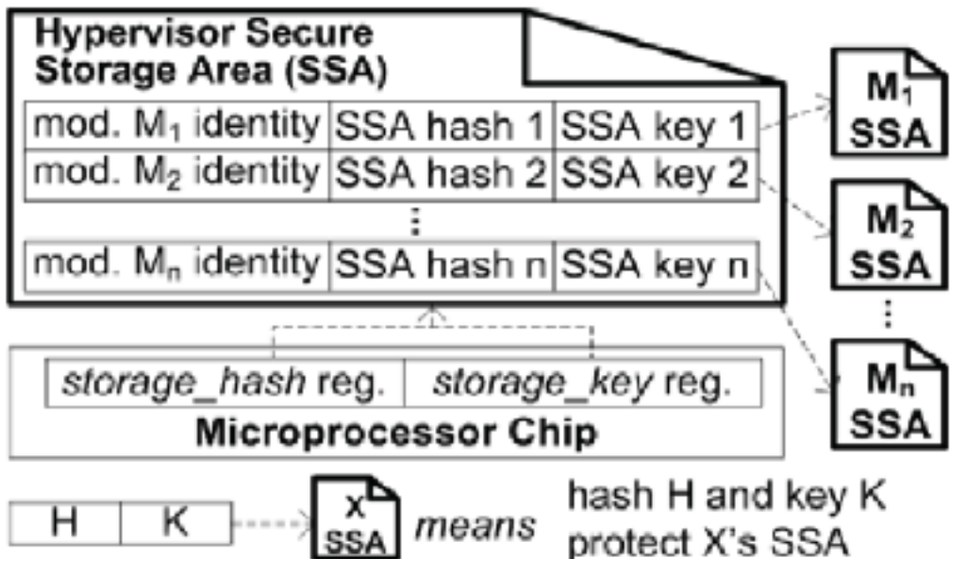
- A protected module:

  1. Generages Key

  2. Encrypts Data

  3. Computes hash

  4. Stores Data



Implementation

- Single-thread UltraSparc
- Bastion hardware
- Modified hypervisor
- New set of registers (2048 bit of reg storage)
- Extended TLBs with 5-bits for module_id
- L2 does not exist, emulated by Microblaze
- AES engine through FSL

Hardware Complexity

	Original System	Bastion Additions *absolute (change)*
SPARC CPU Core	*T1 core*	*new regs, TLB tags, new instructions*
Slice Registers	19.6K	1.6K (+8.2%)
Slice LUTs	30.9K	1.1K (+3.6%)
BRAMs	98	0 (+0%)
Non-core HW	*crossbar, L2 ctrlr,etc*	*AES crypto core, µBlaze interface*
Slice Registers	28.7K	5.9K (+20.6%)
Slice LUTs	39.5K	6.5K (+16.5%)
BRAMs	114	15 (+13.2%)

Software Complexity

	Original System	Bastion Additions *absolute (change)*
OpenSolaris OS	9.06M	0 (+0%)
Hypervisor	38.1K	2.9K (+7.6%)
Cache Firmware	12.1K	0.8K (+6.6%)
CPU Routine	0	1.5K (N/A)

Pros & Cons

- Complete work that includes SW and HW
- Implementation Effort

- Uniprocessor
- Arbitrary Trusted Software modules -> module_id
- Most of the key concepts are not new
- FPGA prototype seems incomplete