



RISC-V Security Architecture Introduction

6/17/2019



Agenda

- RISC-V Security Philosophy & Mechanisms.
- How to adopt it in a simple Embedded System?
- How to adopt it in a RTOS?
- RISC-V needs an Open-Source Enclave.



Security Philosophy

- Provide small set of hardware primitives that supports multiple security uses
- Less hardware to build, less to get wrong
- Mechanisms to allow code to be pushed out of trusted code base

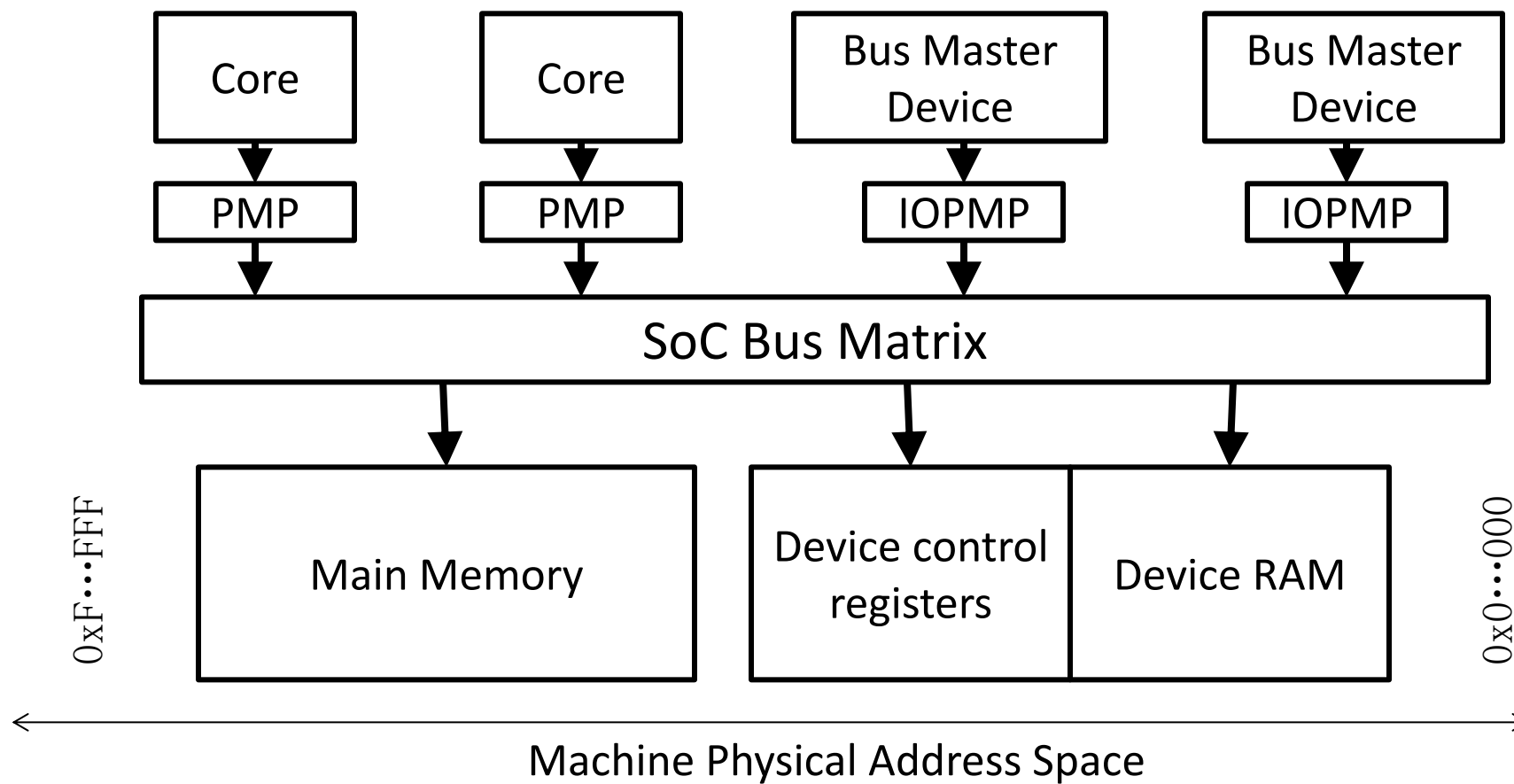


Privilege Modes

- **Machine mode (M-mode)**
 - AKA monitor mode, microcode mode, ...
- **Hypervisor-Extended Supervisor Mode (HS-Mode)**
- **Supervisor Mode (S-mode)**
- **User Mode (U-mode)**
- **Supported combinations of modes:**
 - M (simple embedded systems)
 - M, U (embedded systems with security)
 - M, S, U (systems running Unix-like operating systems)
 - M, S, HS, U (systems running hypervisors)



Physical Memory Protection





M-Mode controls PMPs

- M-mode has access to entire machine after reset
- Configures PMPs and IOPMPs to contain each active context inside a physical partition
- Can even restrict M-mode access to regions until next reset
- M-mode can dynamically swap PMP settings to run different security contexts on a hart



PMP Configuration

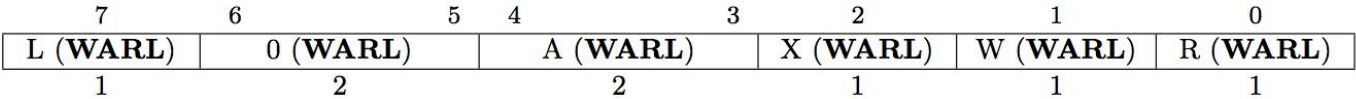


Figure 3.27: PMP configuration register format.

A	Name	Description	pmpaddr	pmpcfg.A	Match type and size
0	OFF	Null region (disabled)	yyyy...yyy	NA4	4-byte NAPOT range
1	TOR	Top of range	yyyy...yyy0	NAPOT	8-byte NAPOT range
2	NA4	Naturally aligned four-byte region	yyyy...yy01	NAPOT	16-byte NAPOT range
3	NAPOT	Naturally aligned power-of-two region, ≥8 bytes	yyyy...y011	NAPOT	32-byte NAPOT range
		
			yy01...1111	NAPOT	2 ^{XLEN} -byte NAPOT range
			y011...1111	NAPOT	2 ^{XLEN+1} -byte NAPOT range
			0111...1111	NAPOT	2 ^{XLEN+2} -byte NAPOT range
			1111...1111	NAPOT	Reserved

Table 3.8: Encoding of A field in PMP configuration registers.

Table 3.9: NAPOT range encoding in PMP address and configuration registers.



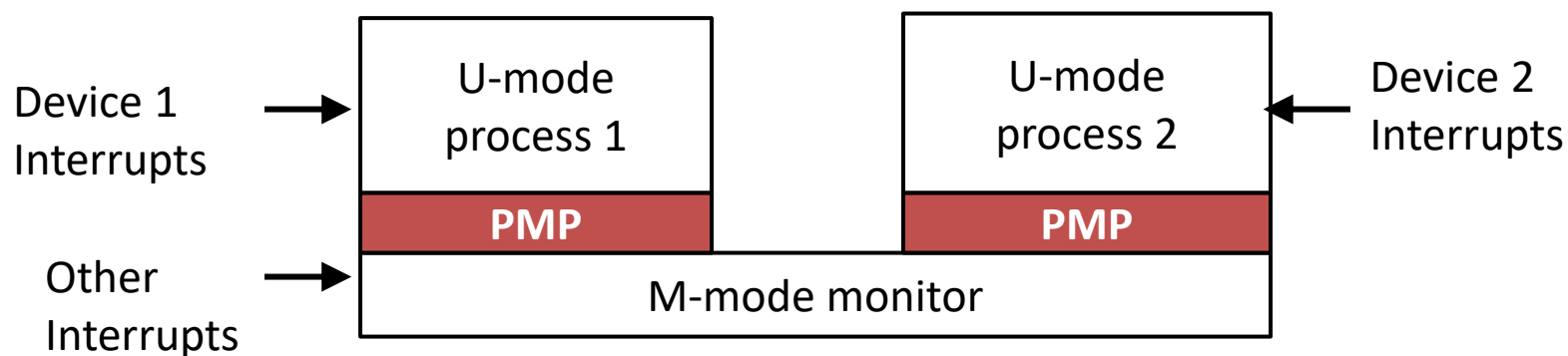
Delegation

- **Exceptions and interrupts can be selectively delegated in hardware out of M-mode to a lower privilege mode**
 - Reduces size of TCB, interrupt handlers can run at full hardware speed but inside a secure partition
 - Some system might require some instructions to be emulated in M-mode
 - Some operations will require M-mode execution
 - e.g., cache flush for software coherence, power down, temporal security fence



Secure Embedded Systems (M, U modes)

- M-mode runs secure boot and runtime monitor
- Embedded code runs in U-mode
- Physical memory protection (PMP) on U-mode accesses
- Interrupt handling can be delegated to U-mode code
 - User-level interrupt support (N-extension)
- Provides arbitrary number of isolated security contexts





Mode-Specific Memory Apertures

- **Problem:** Want device to have different behavior based on accessor's privilege mode
- **Solution:** Provide multiple memory maps, one per privilege mode. Use PMPs (and/or VM system) to constrain security context to access correct aperture.
- **Example:**
0xC000_0000 M-mode access to device reg A
0xC008_0000 U-mode access to device reg A
- **System memory bus does not need to transport metadata with every access, simpler hardware**
- **Higher-privilege mode can easily emulate lower-privilege access**



Mode-Specific Memory Apertures

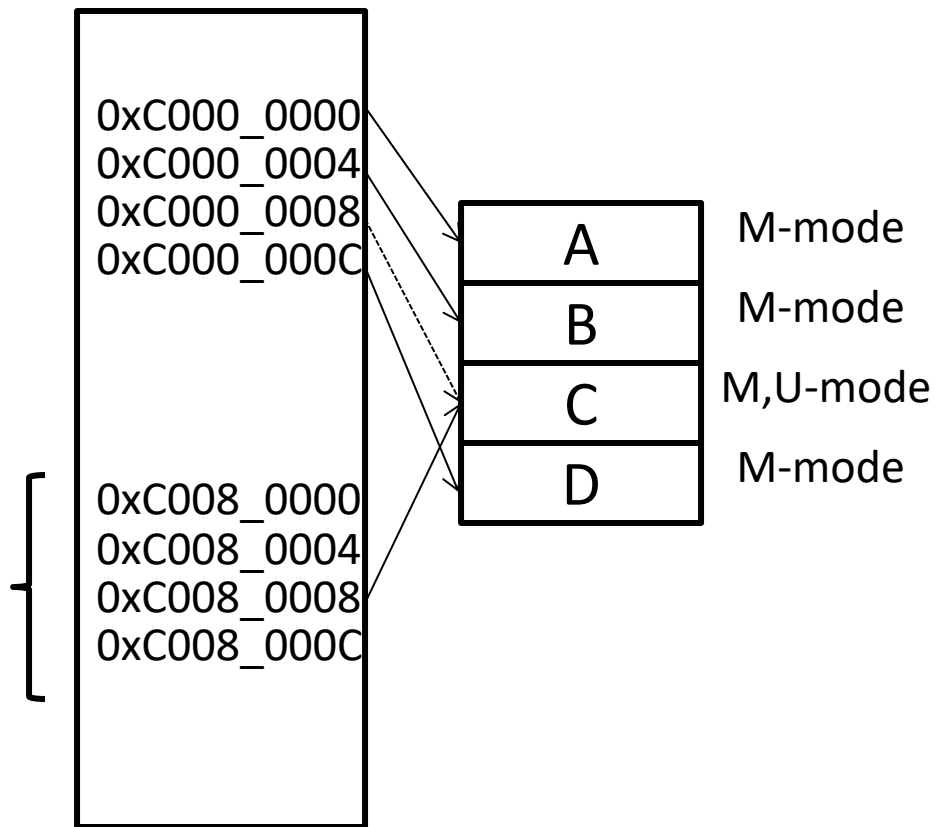
- **Machine-mode only**

- 0xC000_0000
 - Register A 0xC000_0000
 - Register B 0xC000_0004
 - Register D 0xC000_000C

- **User + machine mode**

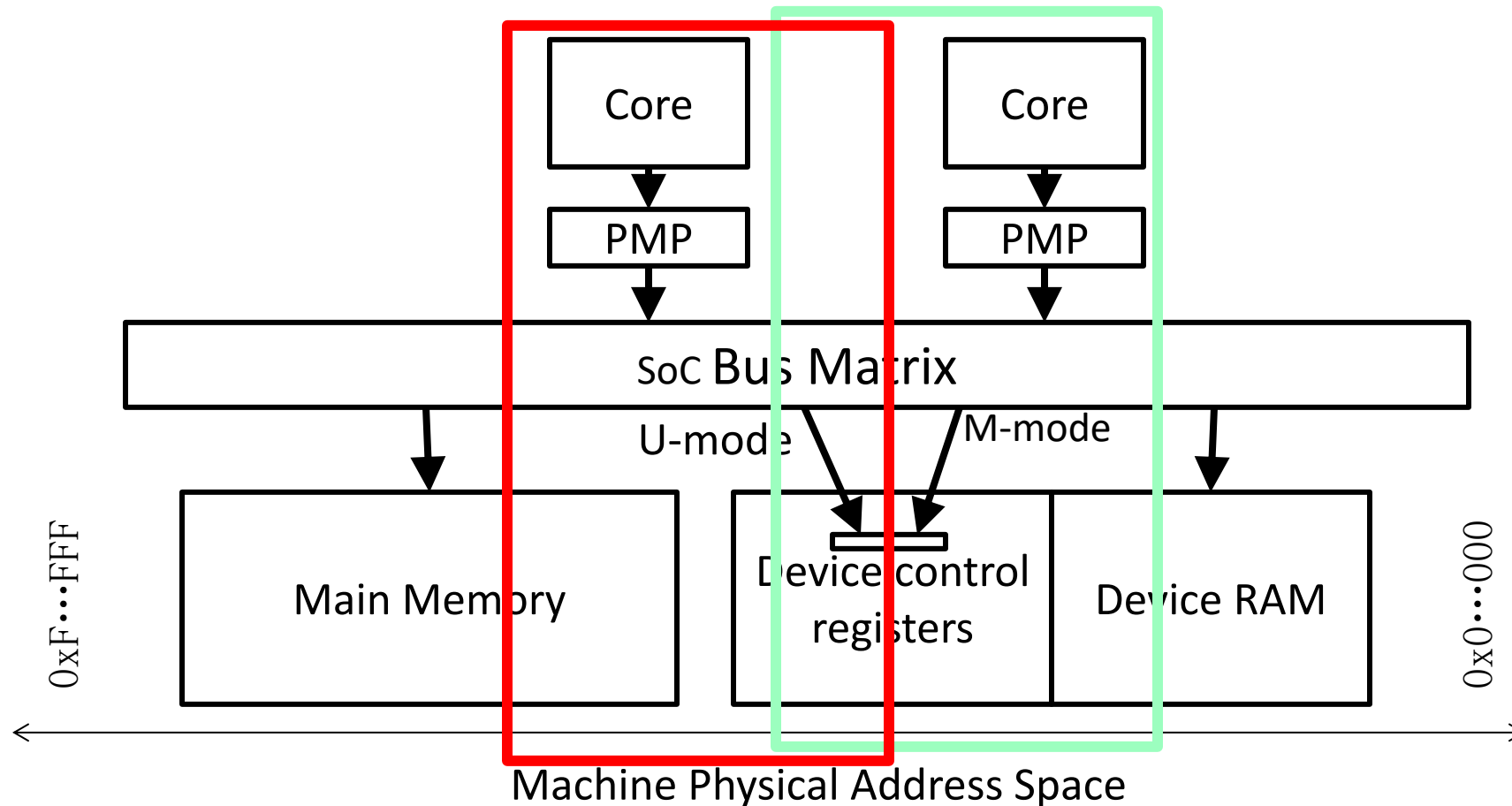
- 0xC008_0000
 - Register C 0xC008_0008

PMP grants user permission



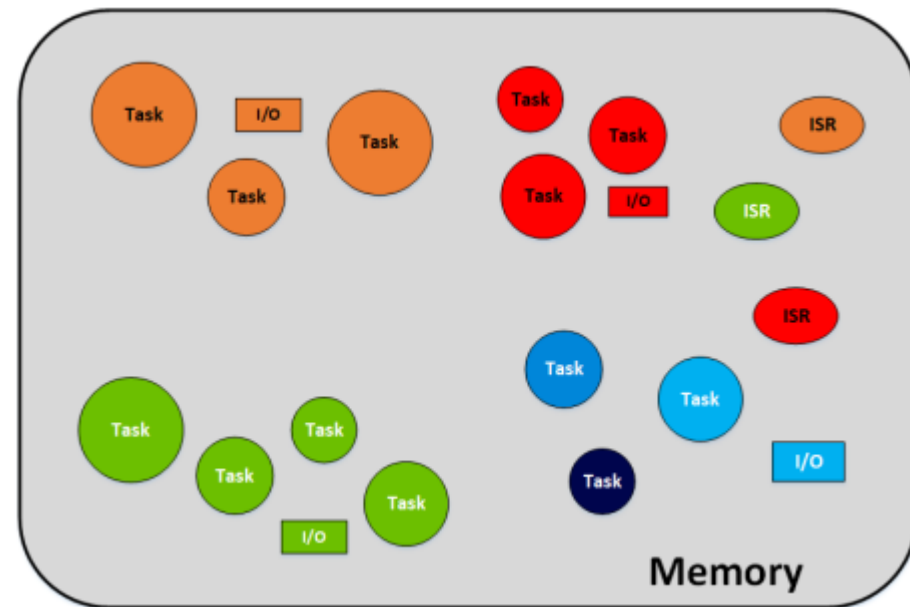


Mode-Specific Memory Apertures



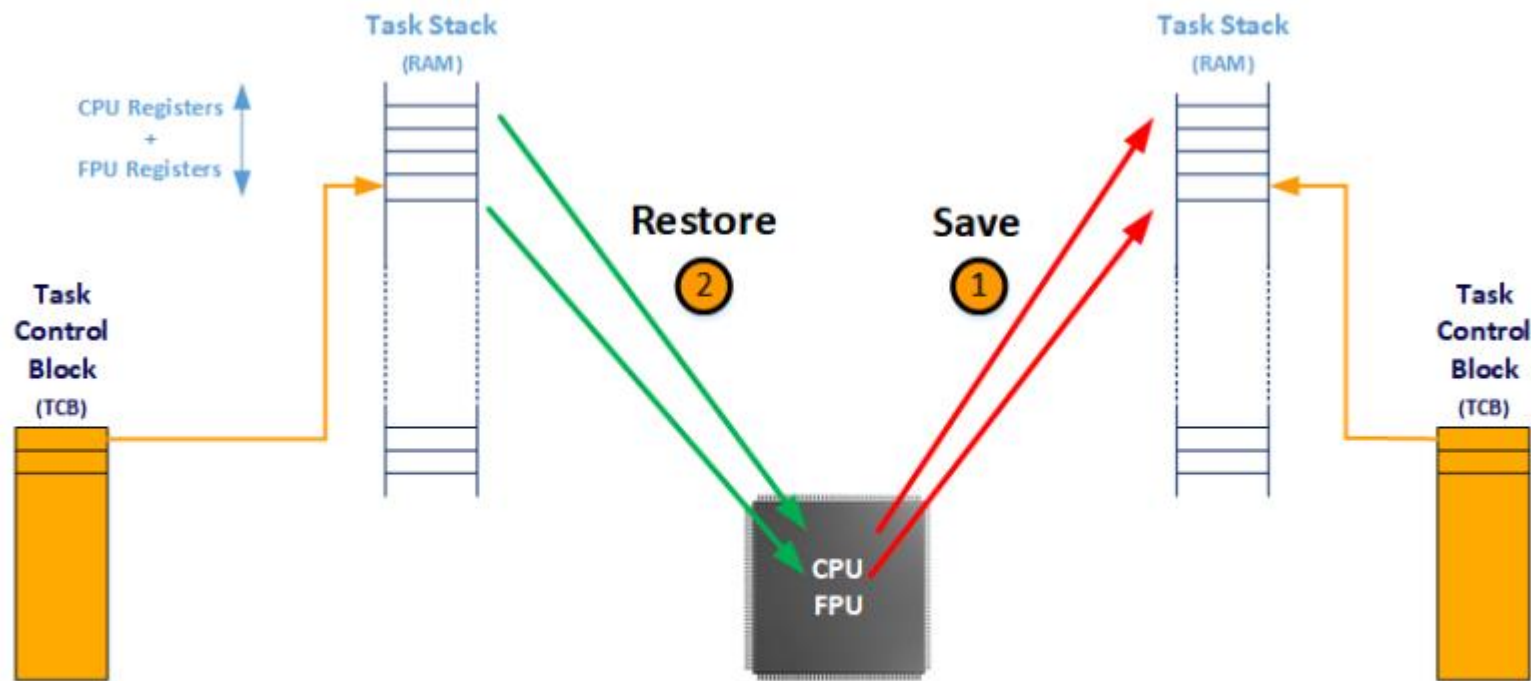
Typical RTOS without Physical Memory Protection

- **Without a PMP, RTOS tasks run in MACHINE-mode**
 - Access to all resources
 - Done for performance reasons
- **Drawbacks**
 - Reliability of the system is in the hands of the application code
 - ISRs and tasks have full access to the memory address space
 - Tasks can disable interrupts
 - Task stacks can overflow without detection
 - Code can execute out of RAM
 - Susceptible to code injection attacks
 - A misbehaved task can take the whole system down
 - Expensive to get safety certification for the whole product





RTOS Context Switch without PMP

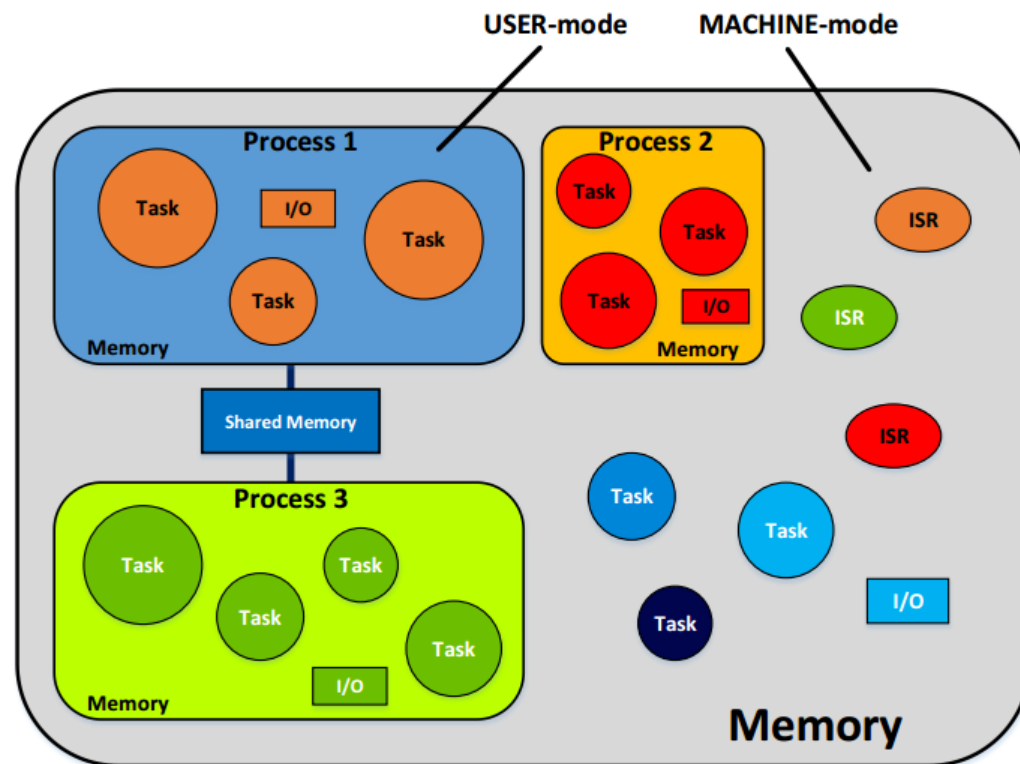


**Context
Switch**



Typical RTOS with Physical Memory Protection

- **Tasks are grouped by processes**
 - Can have multiple tasks per process
- **ISRs have full access to memory**
 - Would be very complex otherwise
- **Benefits:**
 - Memory of one process is not accessible to other processes
 - Unless they share a common memory space
 - Some processes might not need to be safety certified
 - Less expensive and faster time-to-market
 - User tasks can't disable interrupts
 - Task stack overflows can be detected by the PMP



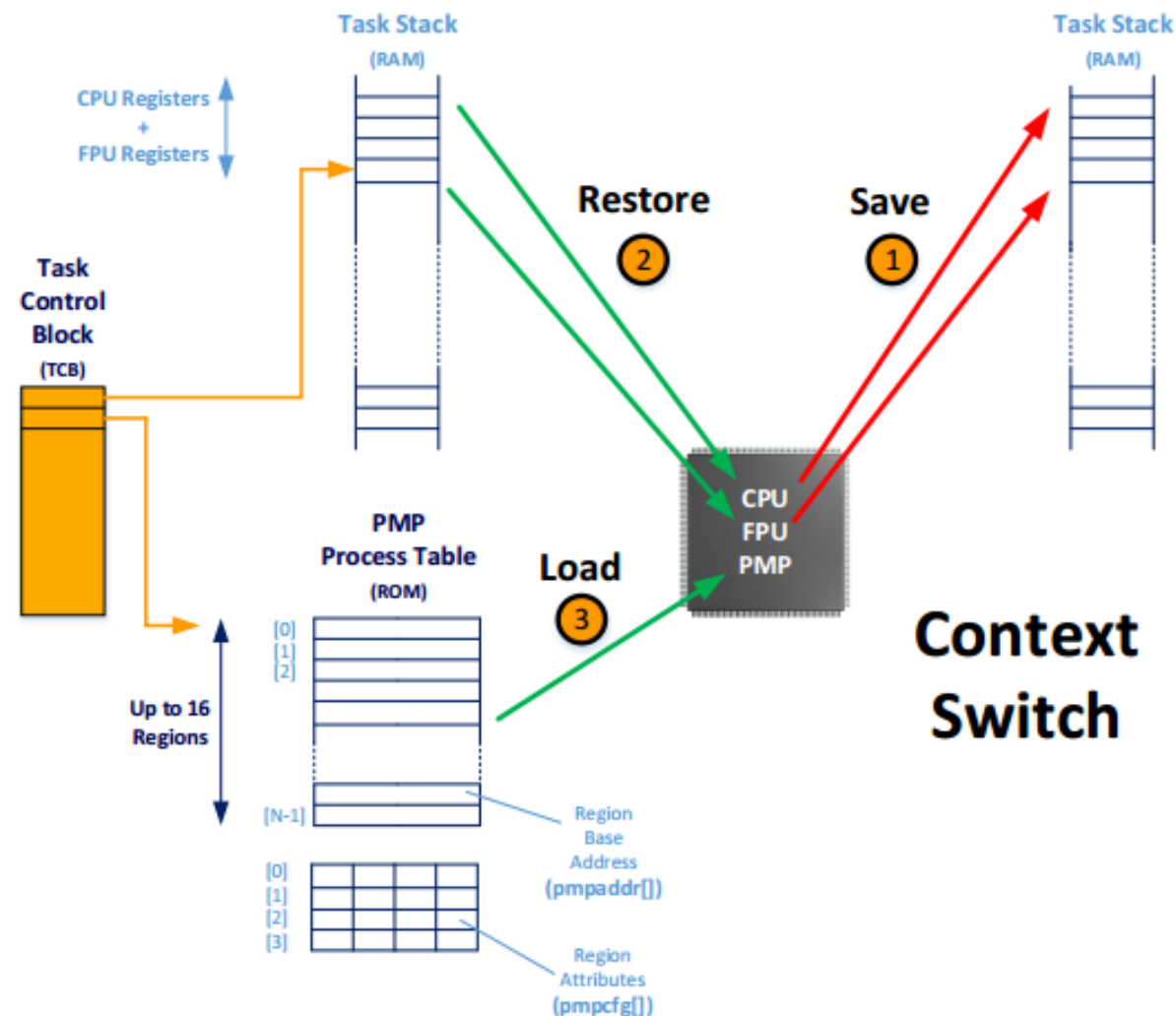


CONFIDENTIAL – COPYRIGHT 2018 SIFIVE. ALL RIGHTS RESERVED.



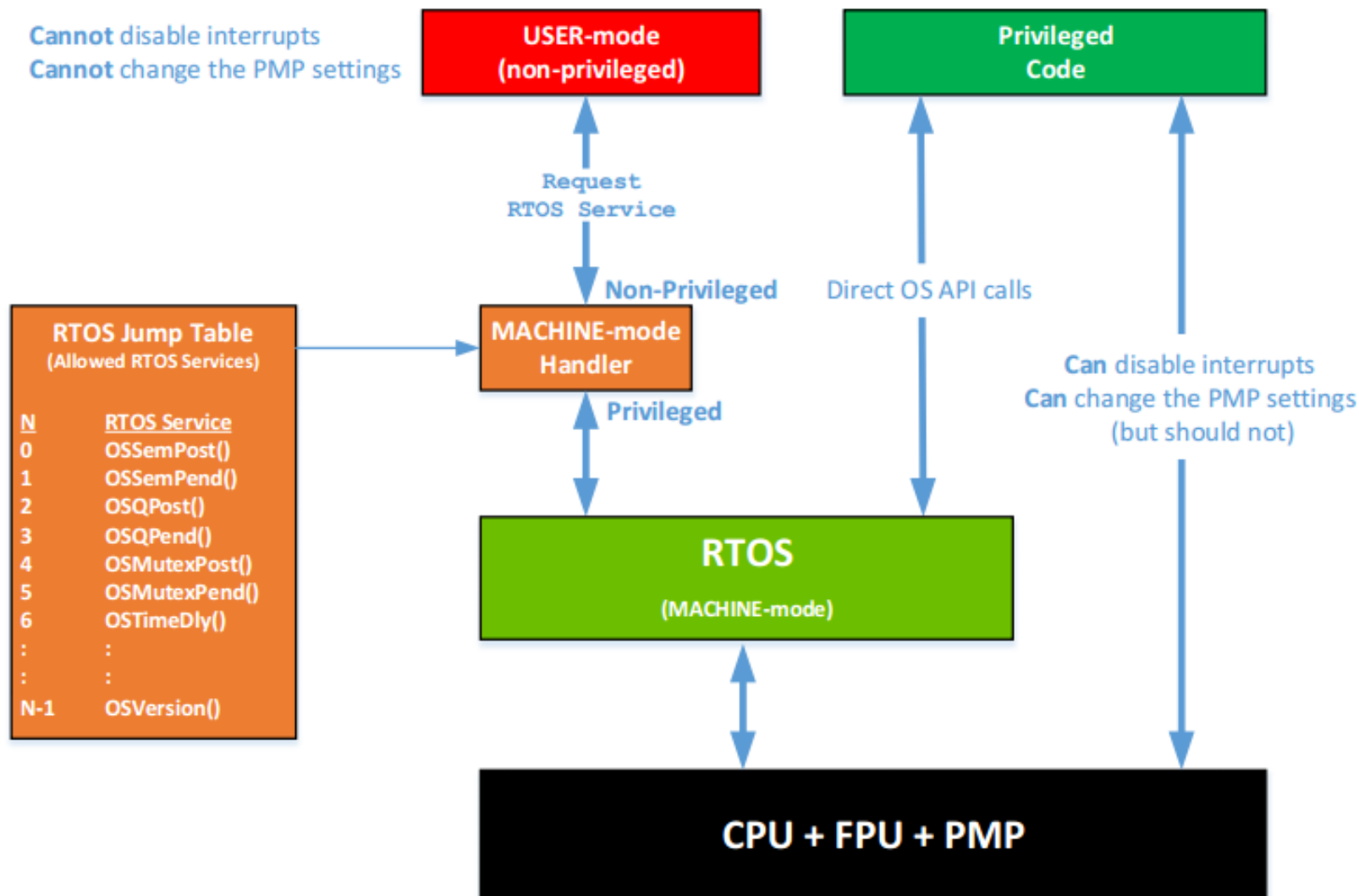


RTOS with PMP – OS updates PMP when Context Switch





RTOS with PMP—User tasks run in USER-mode





Secure Enclave as a Cornerstone Security Primitive

- **Strong security capabilities**
 - Authenticate itself (device)
 - Authenticate software
 - Guarantee the integrity and privacy of remote execution
- **A cornerstone for building new security applications**
 - Confidential computing in the cloud (e.g., machine learning)
 - Secure IoT sensor network



RISC-V needs an Open-Source Enclave

- **Existing enclave systems are proprietary and difficult to experiment with**
 - Closed-source commercial hardware (e.g., Intel SGX, ARM TrustZone)
 - Lack of good research infrastructure
- **A Lot of Challenges for Enclaves**
 - Hardware vulnerabilities: Intel SGX - ForeShadow (USENIX'18), AMD SEV – SEVered (EuroSec'18)
 - Side channel attacks and physical attacks
 - Important questions: do patches really fix the problem? Are there any other issues?
- **Open Source Design**
 - Provides transparency & enables high assurance
 - Builds a community to help people work on the same problems



Keystone: Open Framework for Secure Enclaves

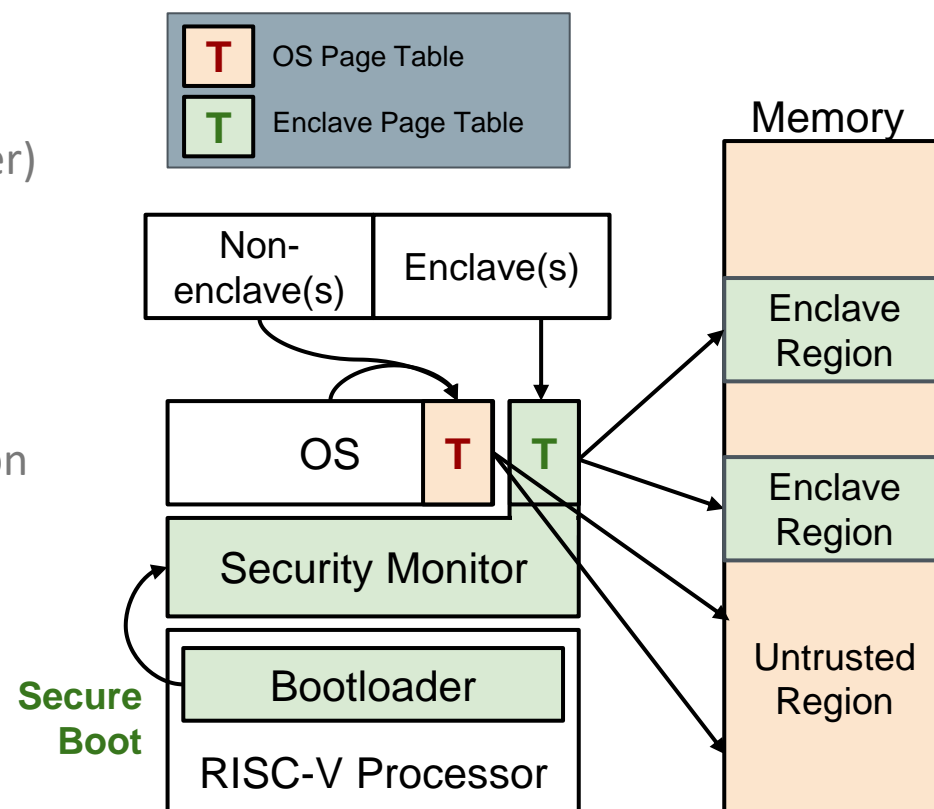
- **The First Full-Stack Open-Source Enclave for Minimal Requirements**
 - Root of trust, security monitor, device driver, SDK, ...
 - Memory isolation, secure bootstrapping, remote attestation, ...
- **Memory Isolation only with Standard RISC-V Primitives**
 - RISC-V Privileged ISA (U-, S-, and M-mode support)
 - Physical Memory Protection (PMP)
 - Demonstrate in unmodified processors
- **Open Framework: Built Modular & Portable for Easy Extension**
 - Platform-agnostic isolated execution environment
 - Platform-specific threat models (cross-core side channels, untrusted external memory, etc)
 - Use various entropy sources/roots of trust in different platforms

github.com/keystone-enclave



Keystone Enclave Architecture

- **Consists of Two Privileged Software**
 - **Bootloader** (read-only, baked in CPU's boot ROM)
 - **Security monitor** (verified by the bootloader)
- **Bootloader**
 - Measures the security monitor
- **Security Monitor (SM)**
 - Manages enclaves
 - Enclave measurement for remote attestation
 - Securely manages memory resources
 - Manages enclave page tables
 - Handles interrupts
- **Keystone Driver (OS module)**
 - Provides Keystone API
 - Coordinates the OS and the SM





Non-ISA-Specific Security Mechanisms

- BIST + Trusted boot from on-chip ROM
- Secure key storage and attack-resistant crypto
- PUFs
- Tamper-detect circuits
- True Random-Number Generators (TRNG)
- Memory encryption and integrity checks