OS Implications for Confidential Computing

Jörg Rödel <jroedel@suse.com>

28. September 2023

SUSE

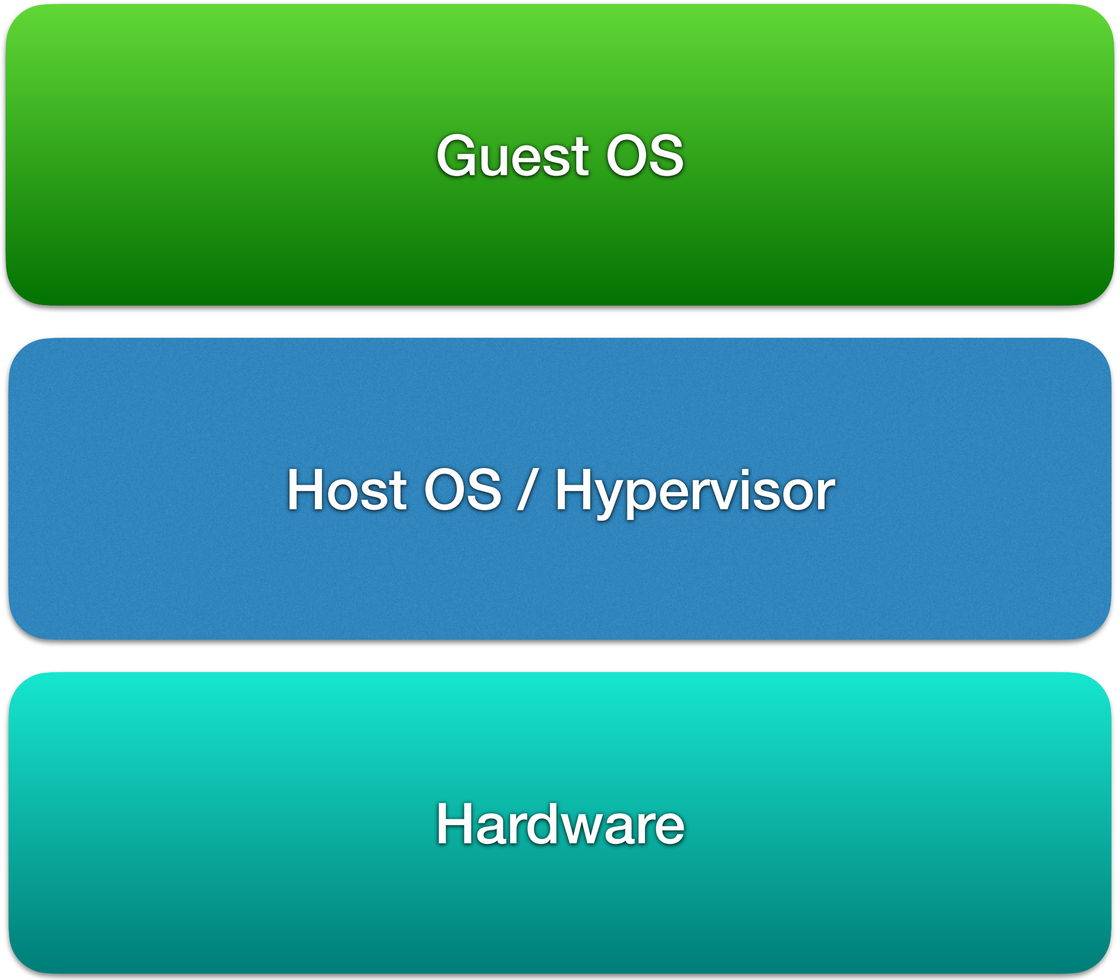
Confidential Computing Definition

Confidential Computing Definition

- Depending on type of Hypervisor (HV) isolation
- Different definitions for Confidential Computing (CoCo):
  - Software Isolation: HV is isolated from guest by software means
  - Hardware Isolation: HV uses of hardware extensions to isolate guest
- This talk will focus on Confidential Computing using Hardware Isolation
- Newer Hardware supports Trusted Execution Environments (TEEs)
- Attestation is integral part of CoCo

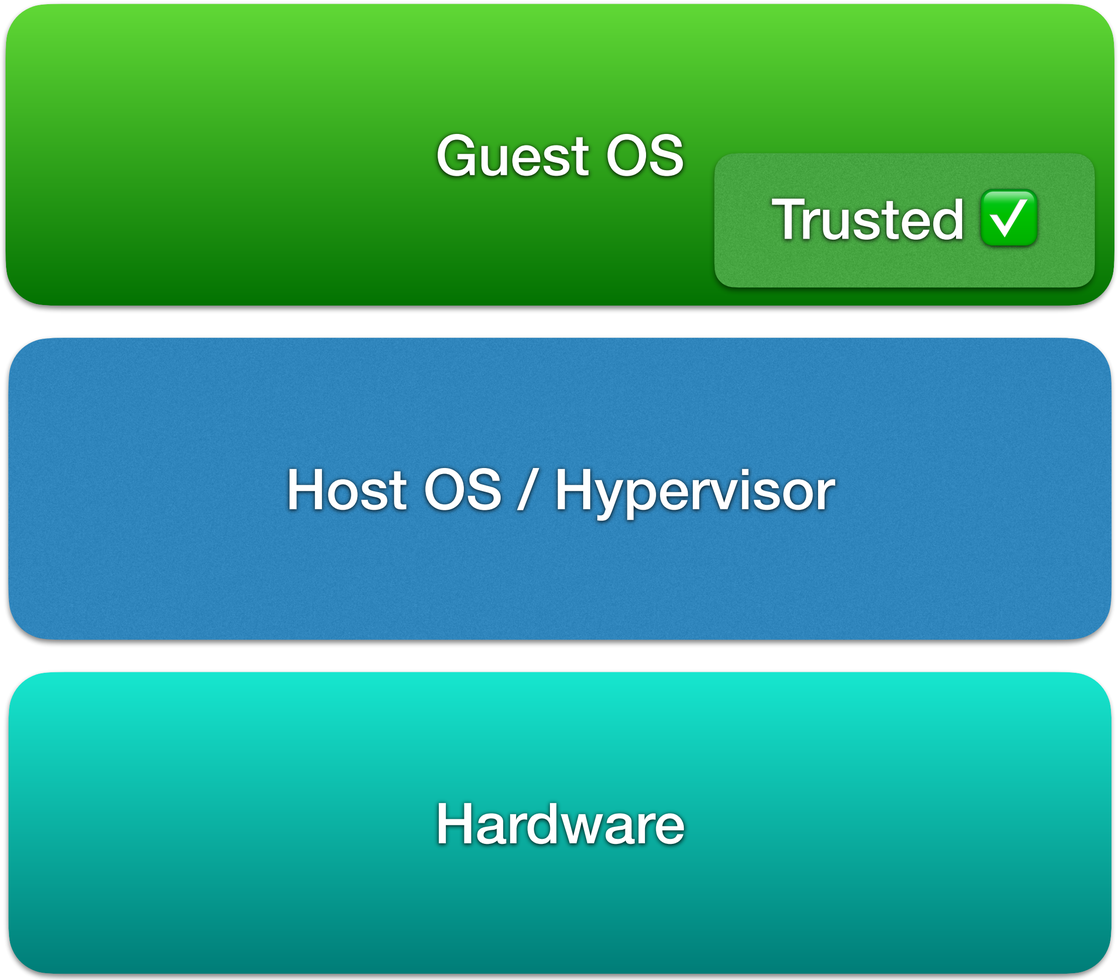
SUSE

Trusted Execution Base



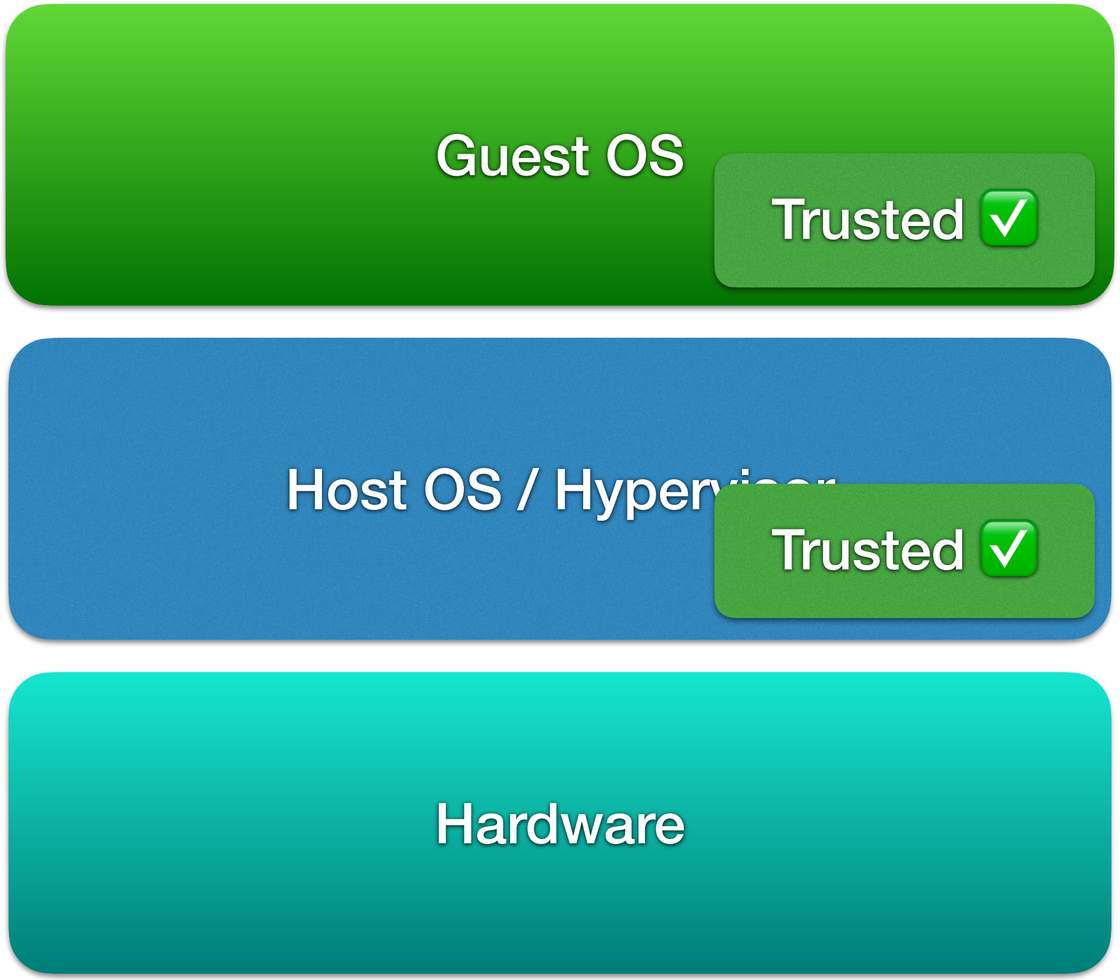
SUSE

Trusted Execution Base



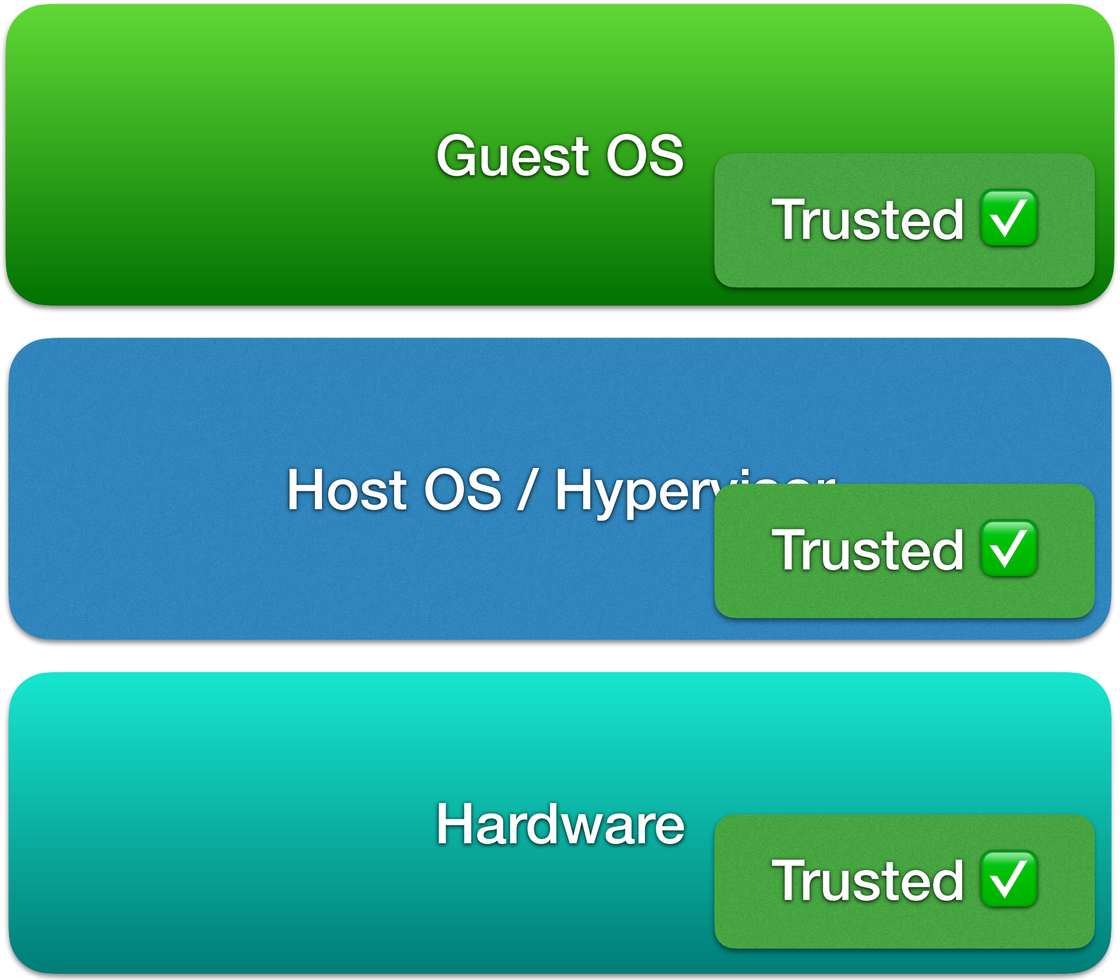
SUSE

Trusted Execution Base



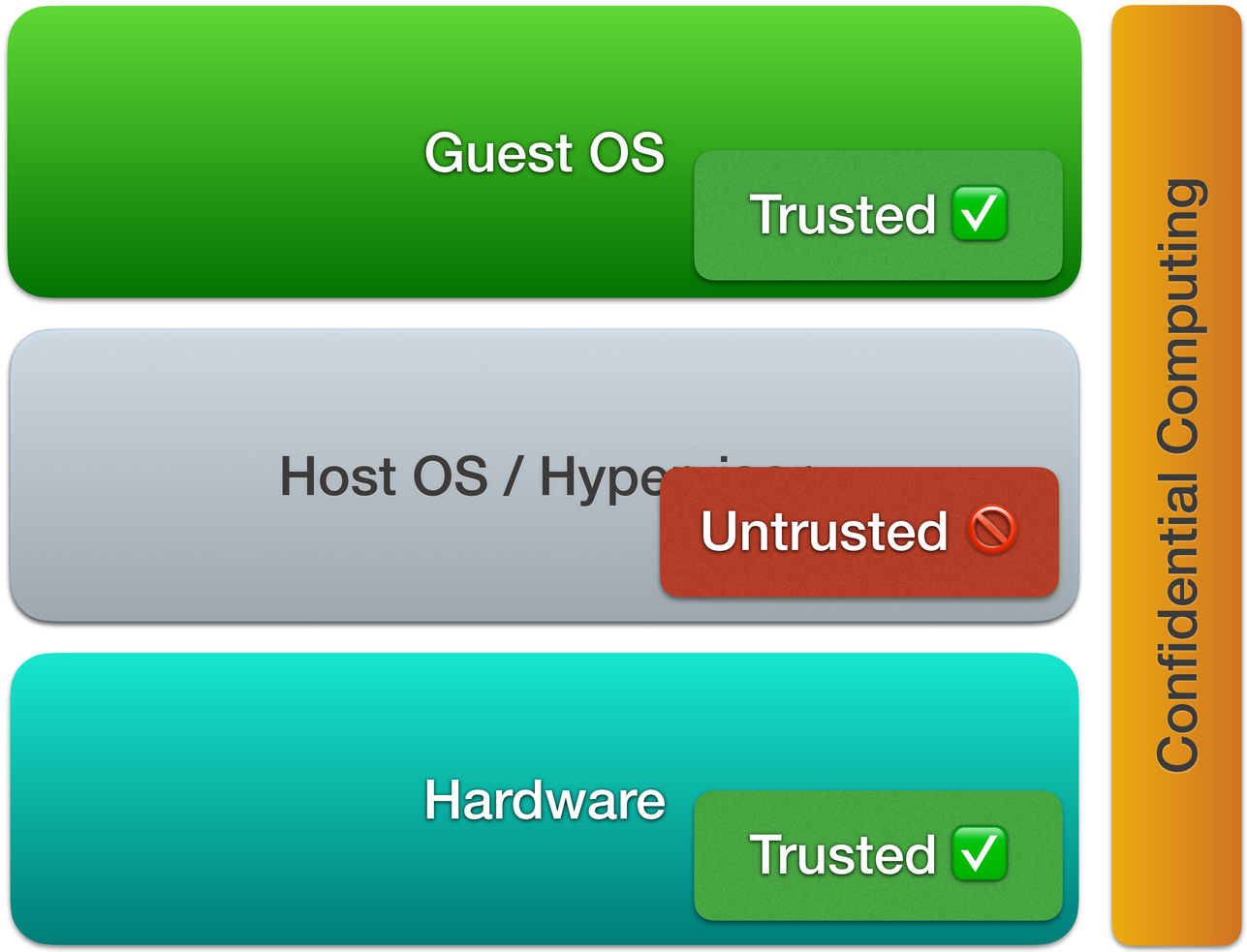
SUSE

Trusted Execution Base



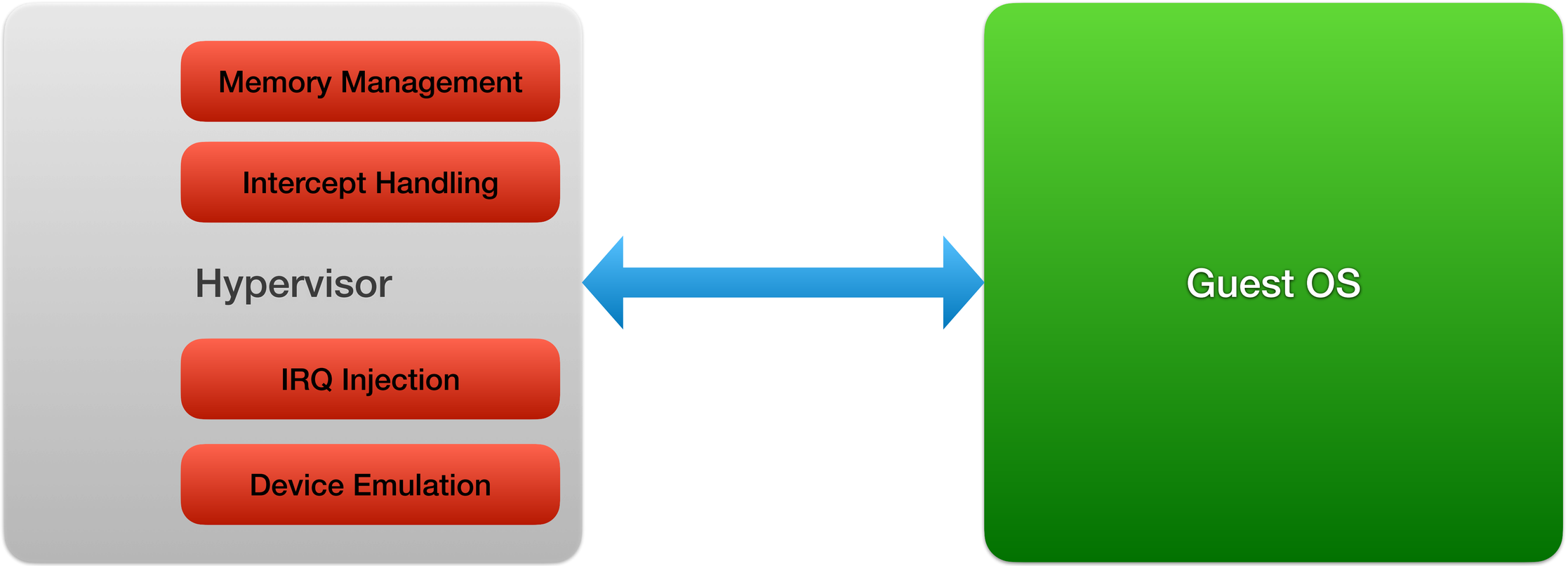
SUSE

Trusted Execution Base



SUSE

Hypervisor Guest Interface



Memory Management

Memory Management

- HV allocates memory for guest usage
- Can read/write arbitrary guest memory at any time
  - Some HV map all guest memory all the time
  - Others use on-demand mappings
- Malicious HV could use that to steal data and/or modify code

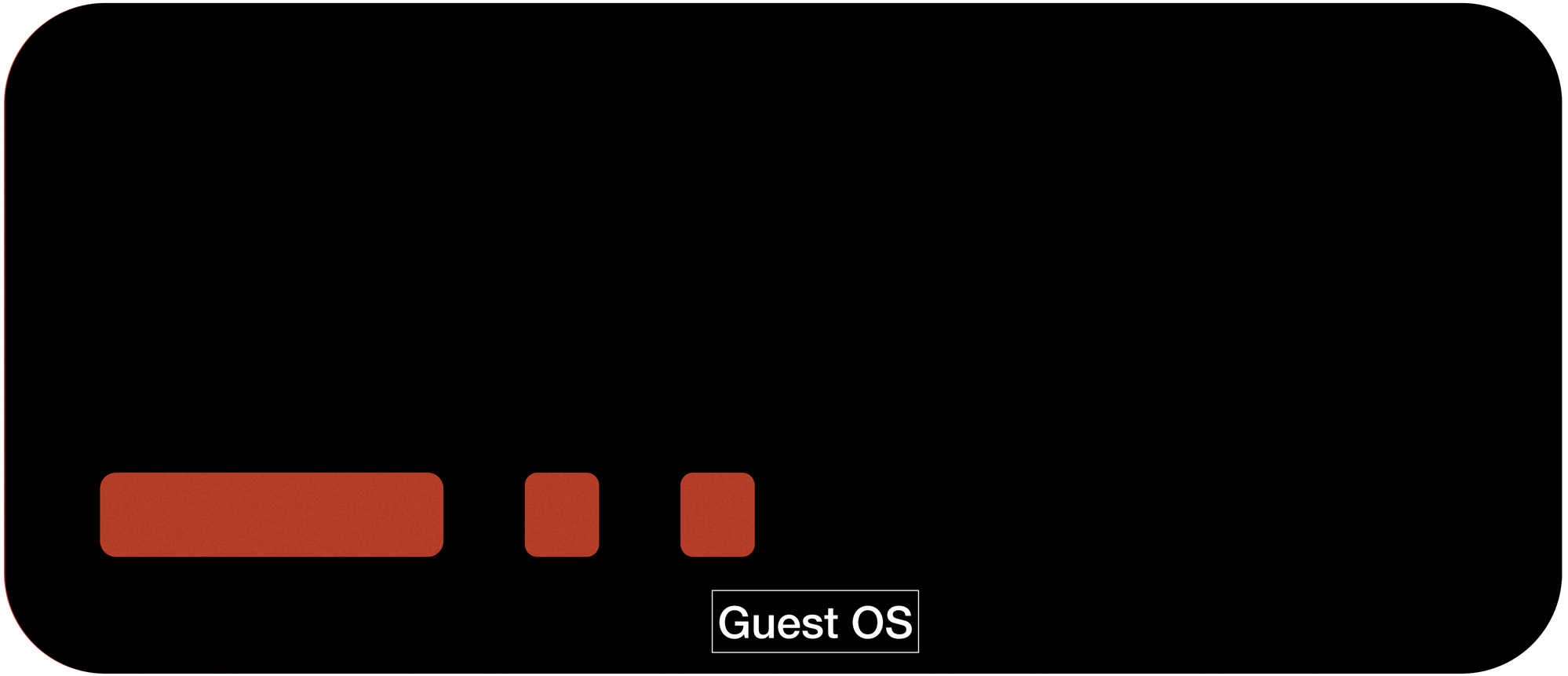
SUSE

Case Study: AMD Secure Encrypted Virtualization

- Confidential Computing in stages - building on each other
  - **Secure Encrypted Virtualization (SEV) - Guest memory encryption**

Guest Memory Encryption

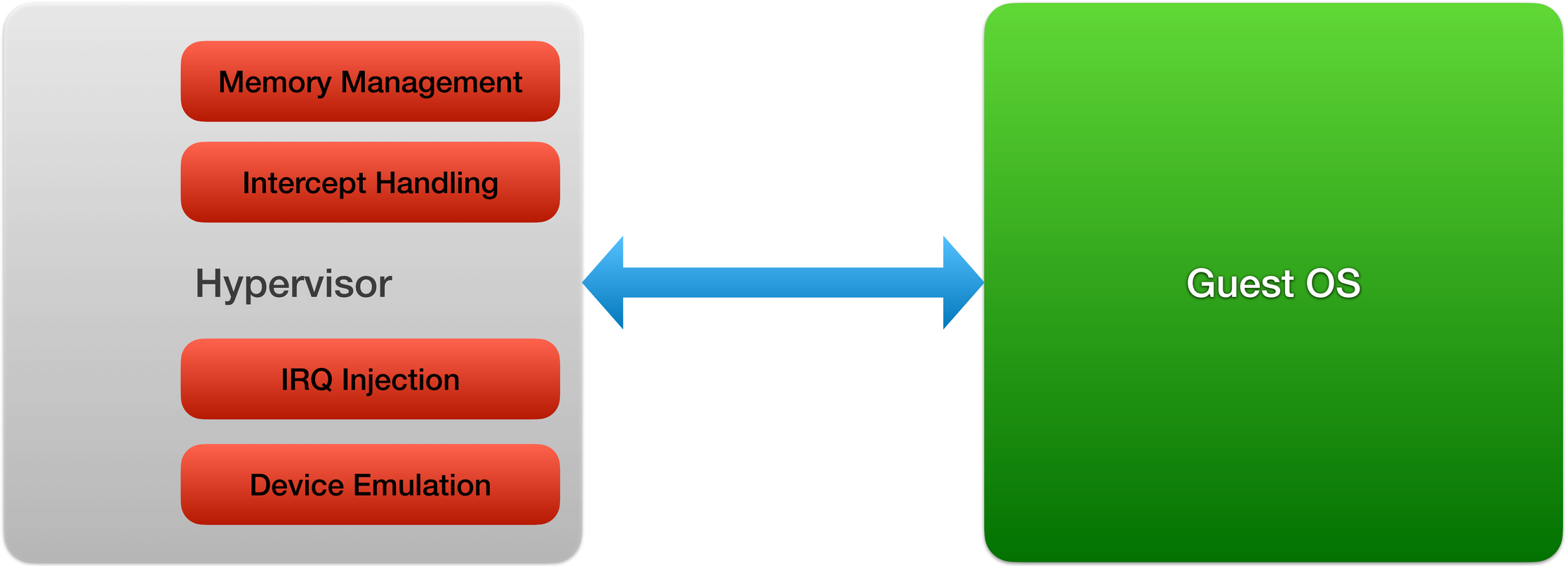
Guest Memory Encryption



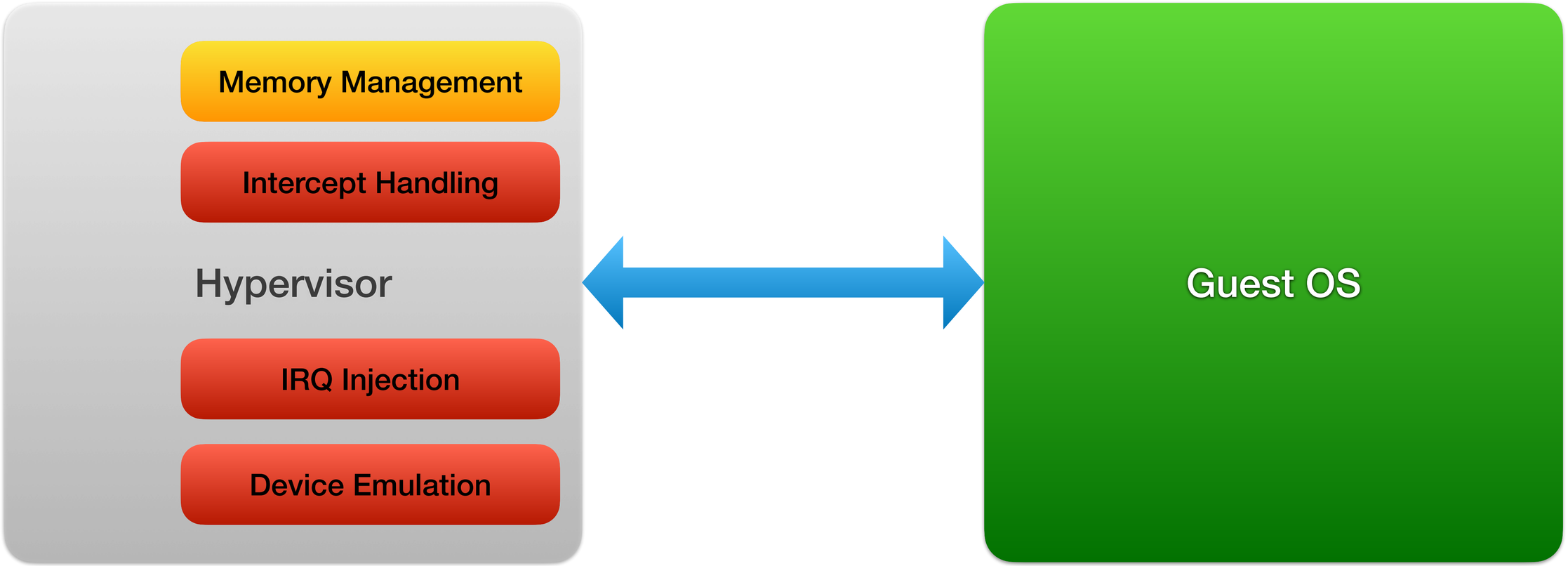
OS Implications

- OS needs to be aware of shared and private (encrypted) memory
- Can decide which memory is shared with the HV
- Sharing implementations:
  - Page-table based: Shared/Private is a flag in the PTE
  - GPA space partitioning: Shared/Private parts of the GPA space
- OS needs changes in Page-table, MMIO, and DMA memory allocation code
- HV can still do memory replay and remapping attacks

Hypervisor Guest Interface



Hypervisor Guest Interface



Intercept Handling

Intercept Handling

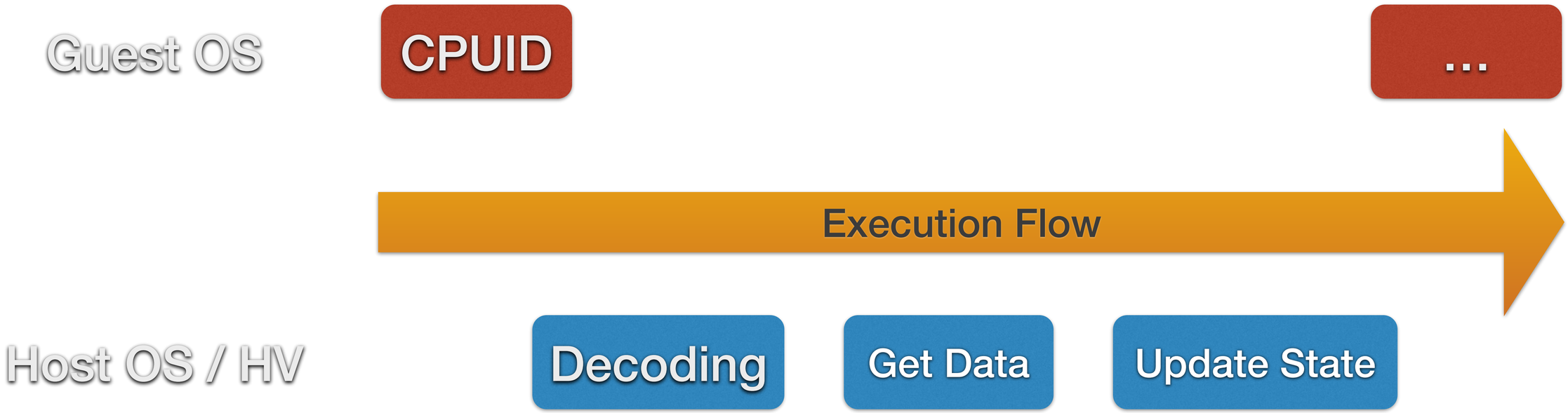
- For intercept handling HV needs access to guest registers
- Registers can be implicit or explicit instruction parameters
- For instruction emulation the HV needs to update guest registers
- Malicous HV could attack the guest:
  - Changing control flow by changing IP or SP
  - Steal secret data, e.g. AES keys stored in FPU registers

SUSE

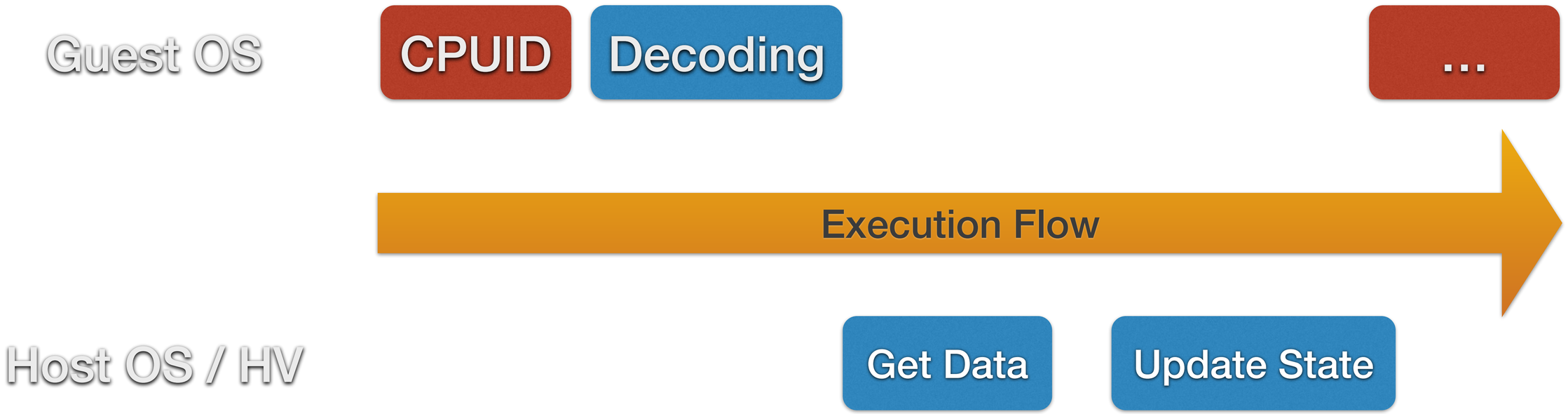
Case Study: AMD Secure Encrypted Virtualization

- Confidential Computing in stages - building on each other
  - Secure Encrypted Virtualization (SEV) - Guest memory encryption
  - **SEV Encrypted State (SEV-ES) - Guest register encryption**

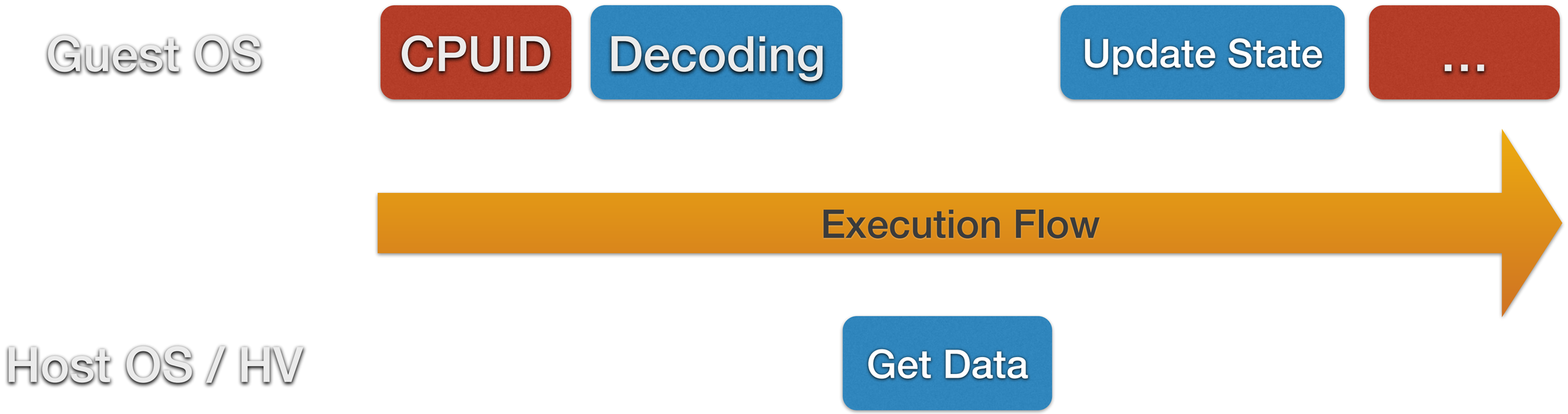
SEV-ES Guest-Host Flow



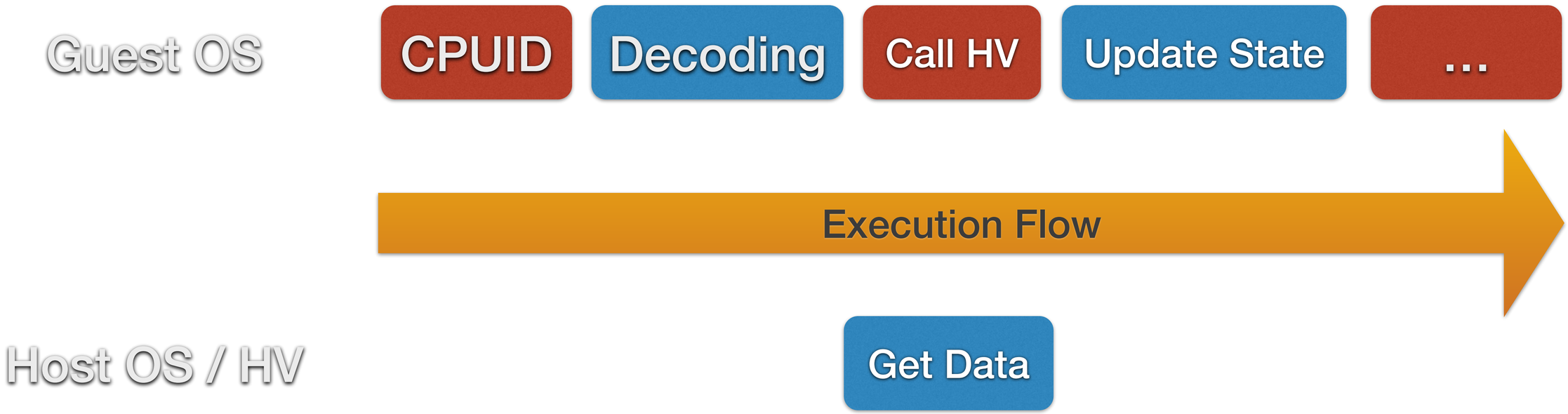
SEV-ES Guest-Host Flow



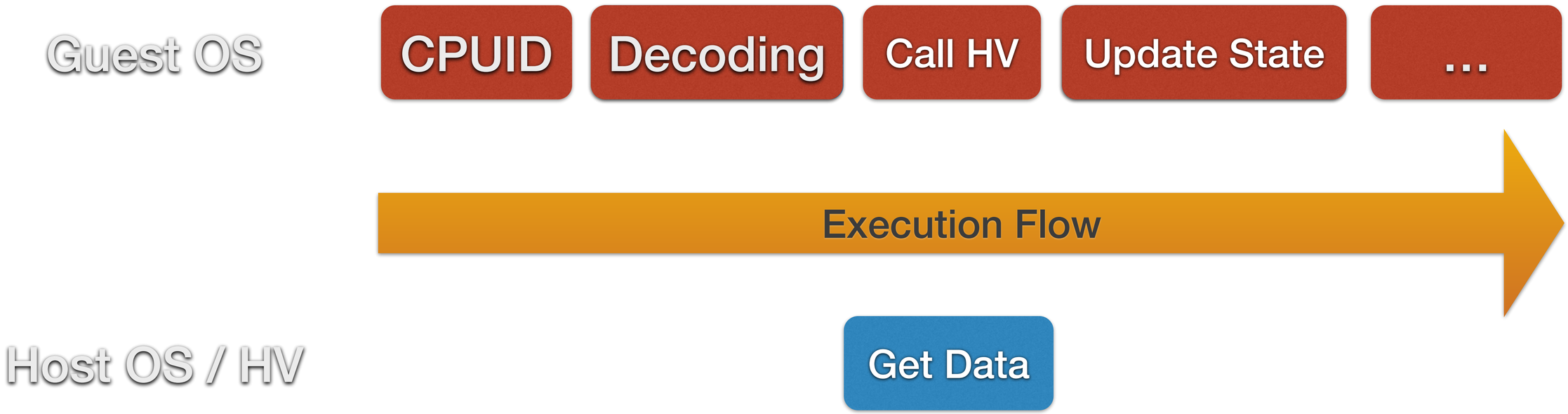
SEV-ES Guest-Host Flow



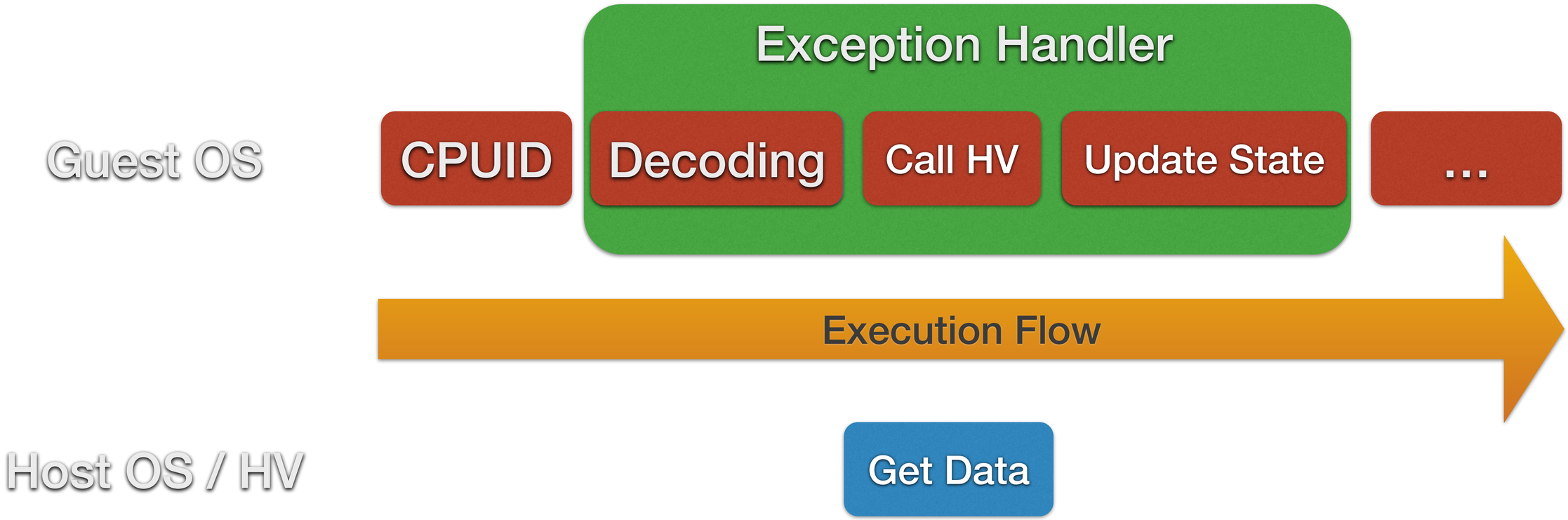
SEV-ES Guest-Host Flow



SEV-ES Guest-Host Flow



SEV-ES Guest-Host Flow

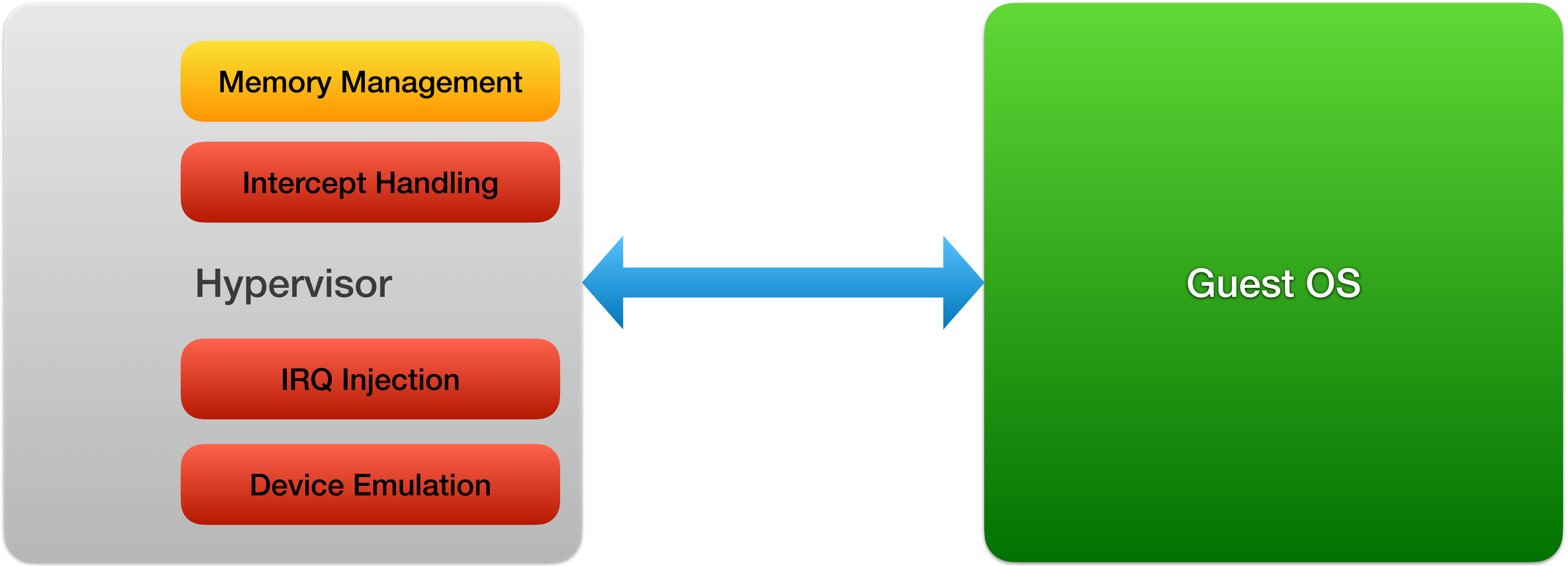


OS Implications

- OS needs to implement new exception handler
  - Will handle intercepts in trusted guest context
  - Uses hyper-calls to exchange data with HV
- Exception handler needed very early in OS boot
  - Before first intercepted instruction is executed
- Exception handler usually includes a full instruction decoder

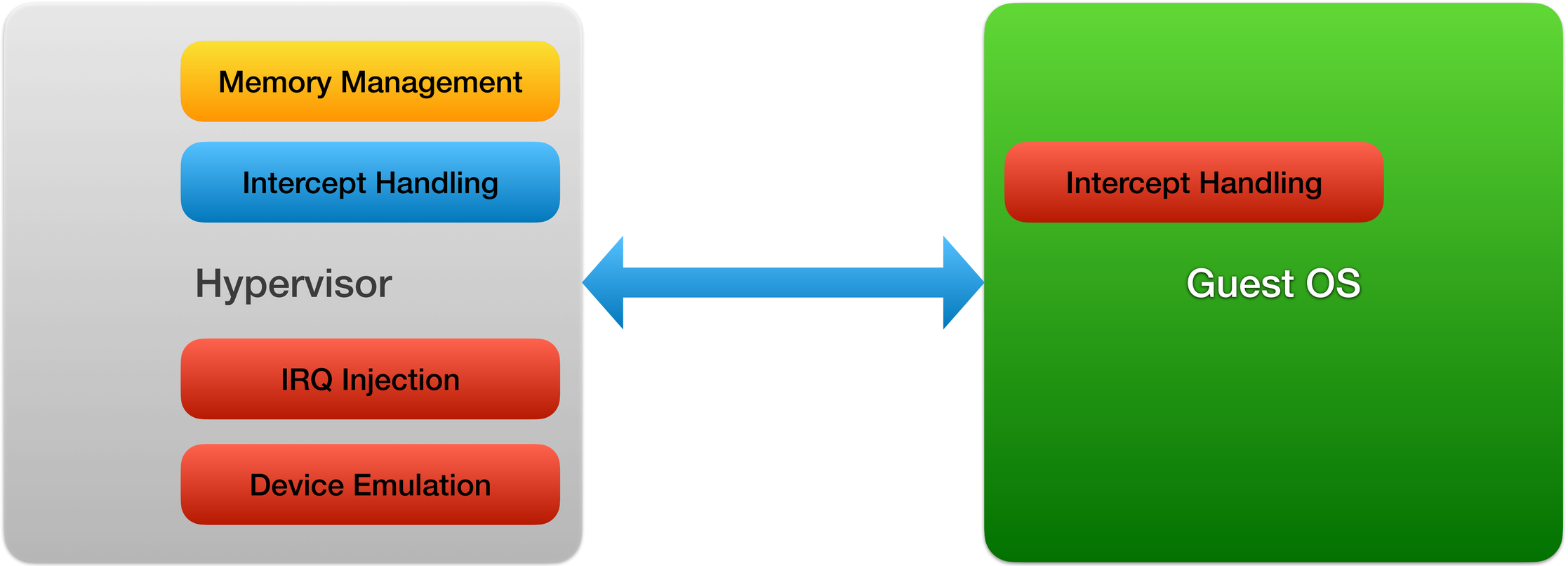
SUSE

Hypervisor Guest Interface



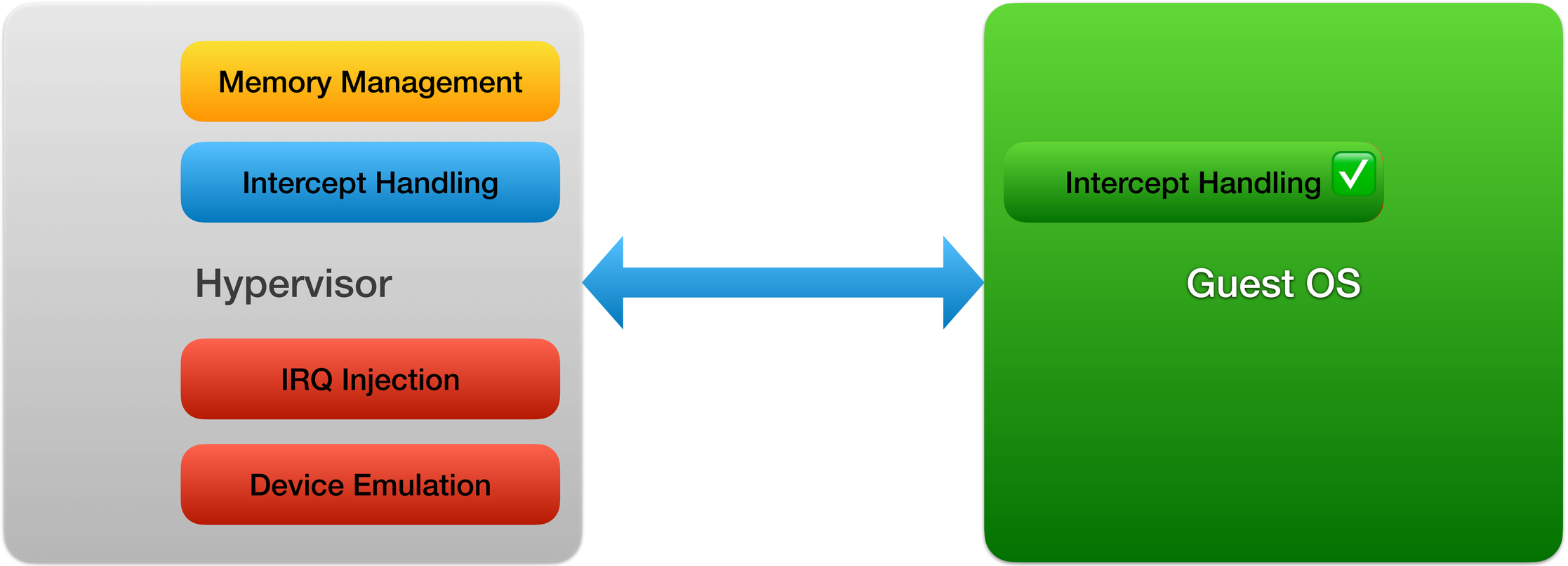
SUSE

Hypervisor Guest Interface



SUSE

Hypervisor Guest Interface



SUSE

Memory Management Revisited

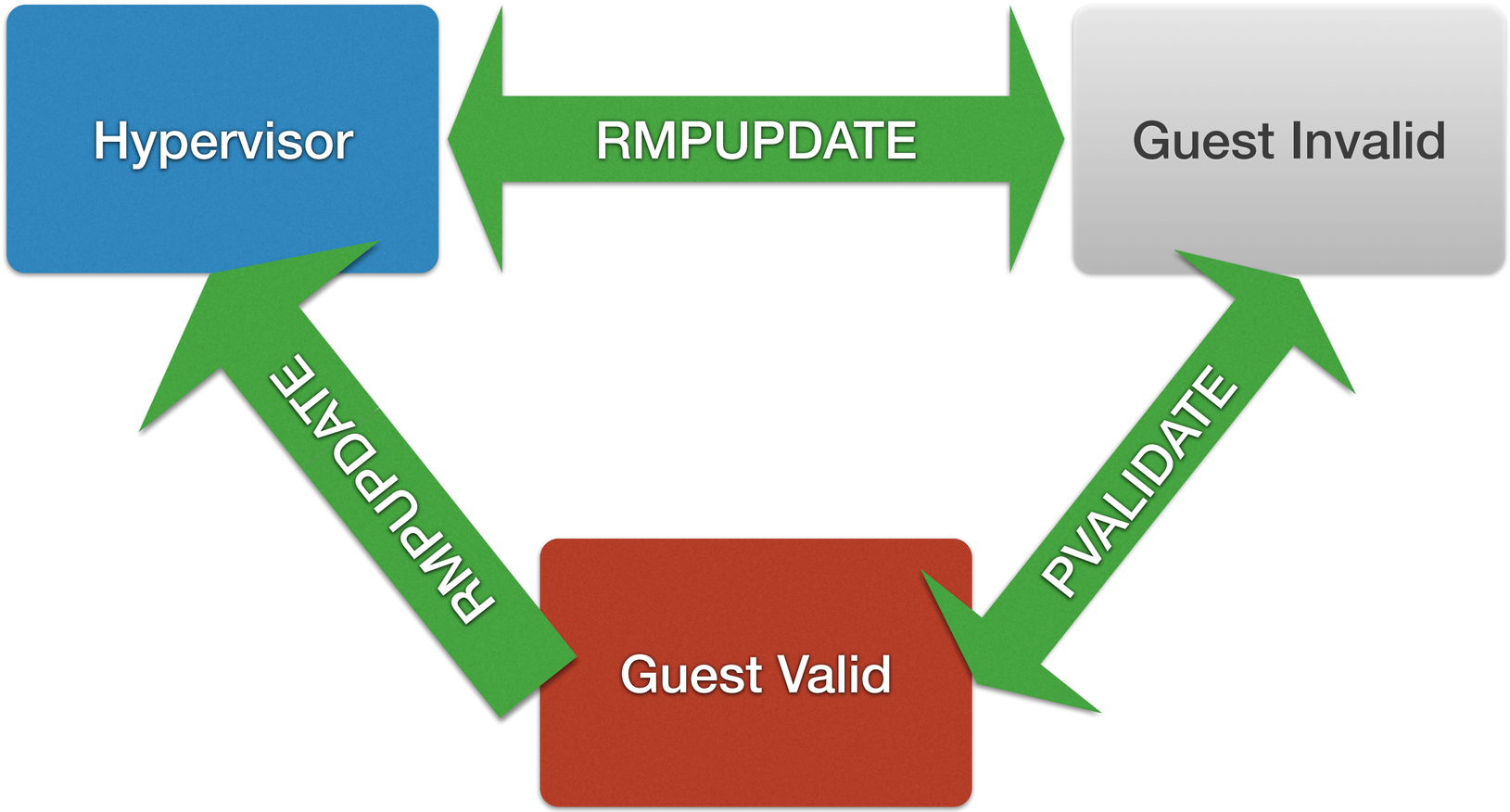
Memory Management Revisited

- Using only memory encryption leaves attack vectors open
  - Memory replay: HV replays an old version of an encrypted page
  - Memory remapping: HV maps encrypted page at different GPA
- Not possible to migitate
- But become detectable via hardware extension

Case Study: AMD Secure Encrypted Virtualization

- Confidential Computing in stages - building on each other
  - Secure Encrypted Virtualization (SEV) - Guest memory encryption
  - SEV Encrypted State (SEV-ES) - Guest register encryption
  - **SEV Secure Nested Paging (SEV-SNP)— Introducing page states**

Page States in AMD SEV-SNP



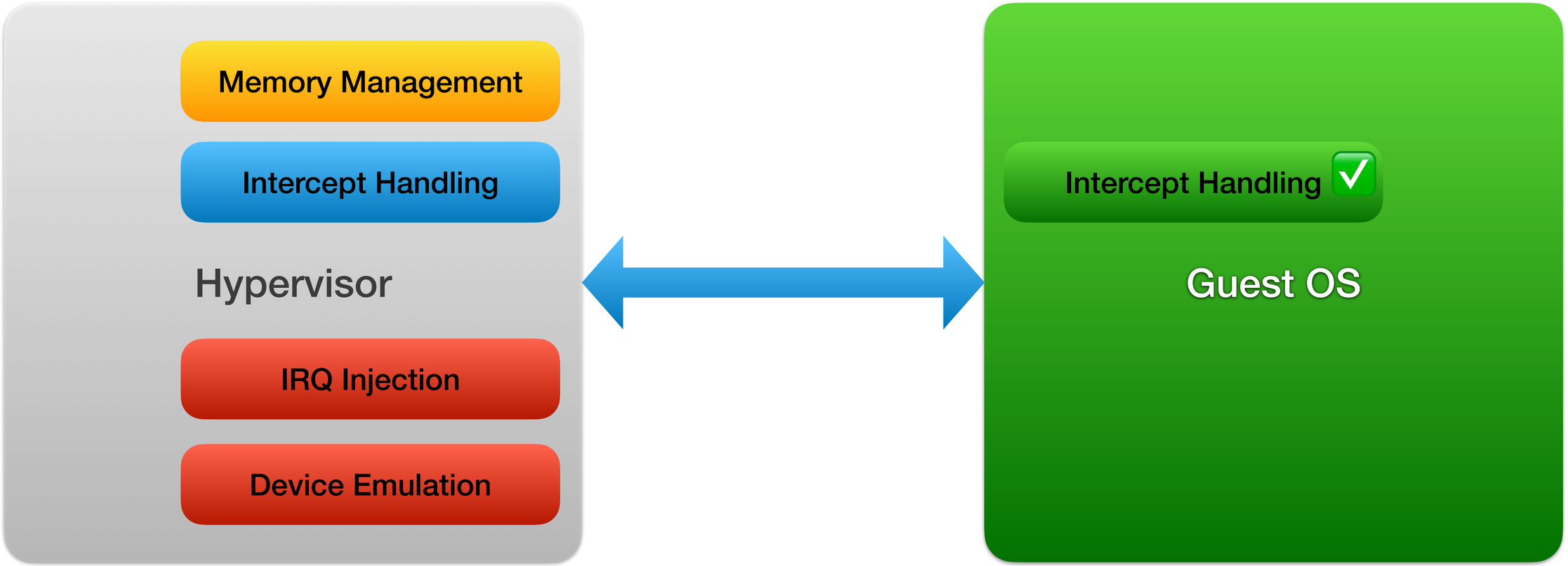
SUSE

OS Implications

- OS needs to keep track of page states
  - Hypervisor vs. Guest-invalid vs. Guest-valid
- Keeping track allows to reliably detect malicious HV behavior
- OS also needs a new paravirtual interface to HV
  - Some page-state changes need to be coordinated with HV
  - OS needs to tell HV when pages switch between HV and Guest-Invalid

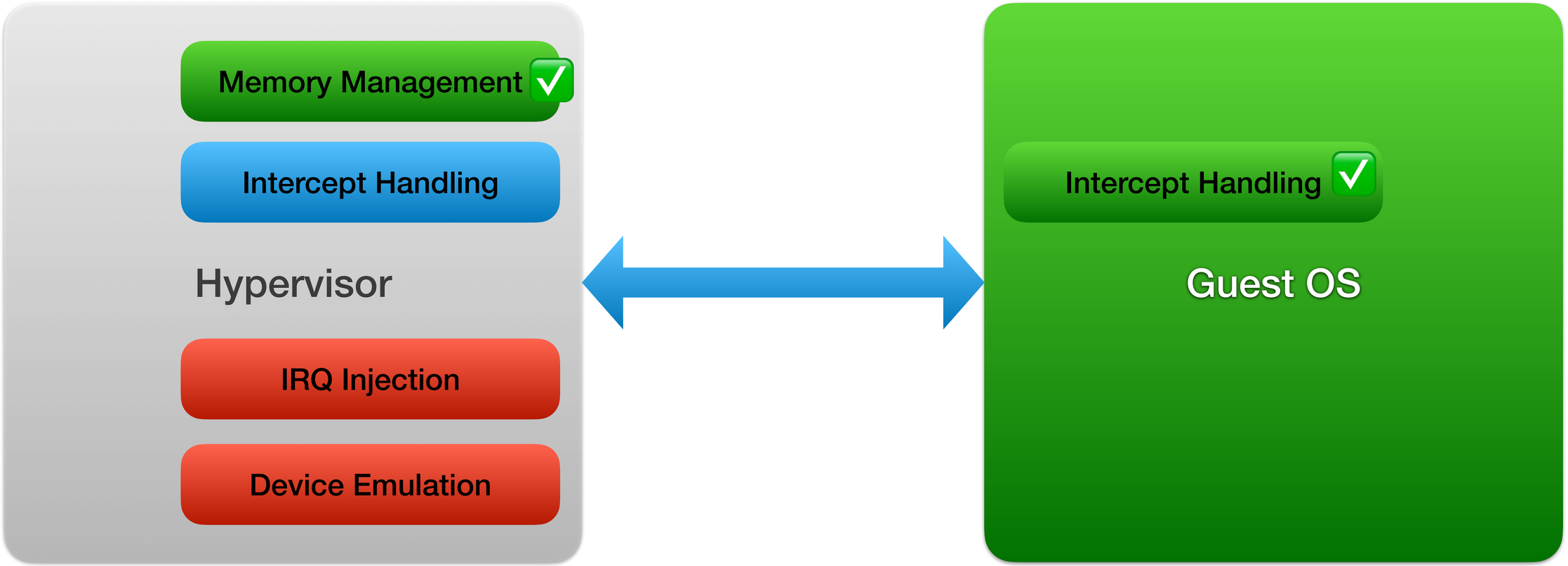
SUSE

Hypervisor Guest Interface



SUSE

Hypervisor Guest Interface



SUSE

Interrupt Injection

SUSE

Interrupt Injection

- HV can inject interrupts at any time
- Usually the HV tracks when the guest is ready for IRQs
  - IRQs enabled
  - No Interrupt shadow
- OSes disable IRQs when not able to handle them
- Malicious HV could inject IRQs while the guest is in a critical state
  - Could be used to change control flow in guest and reveal secrets

Case Study: AMD Secure Encrypted Virtualization

- Confidential Computing in stages - building on each other
  - Secure Encrypted Virtualization (SEV) - Guest memory encryption
  - SEV Encrypted State (SEV-ES) - Guest register encryption
  - SEV Secure Nested Paging (SEV-SNP) - Introducing page states
  - **SEV-SNP Secure Interrupt Injection - Guest is responsible for delivering IRQs**

SEV-SNP Secure Interrupt Injection

- When enabled HV can only inject one event: #HV
- Guest OS needs to be prepared to receive this event at any time
  - Needs to be an IST vector
- PV protocol used to communicate which event is delivered
  - Also involves software blocking of new #HV events

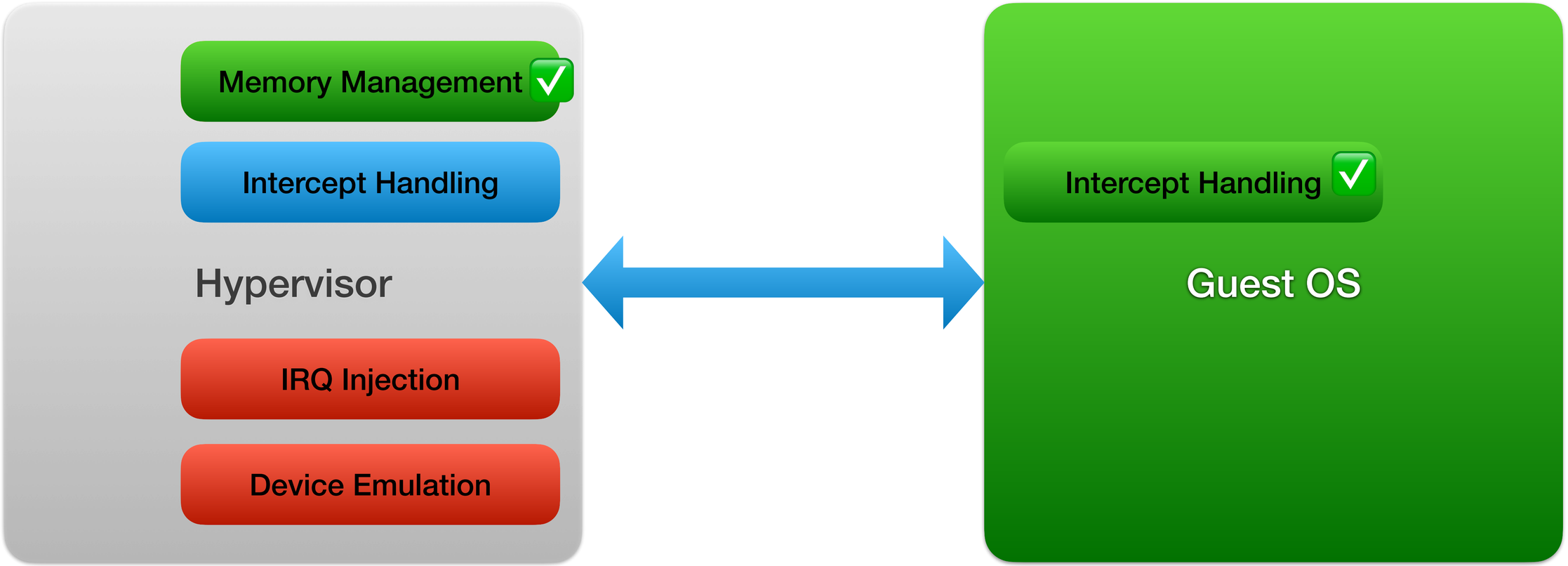
SUSE

OS Implications

- OS needs to implement handler for #HV and harden it
  - Must be IST
  - Must support nesting and detection for malicious injection
- OS needs to track when IRQ handlers can run and deliver events itself
  - Instrumentation of IRQ enable/disable events
  - Manual IRQ handler launching with stack switching

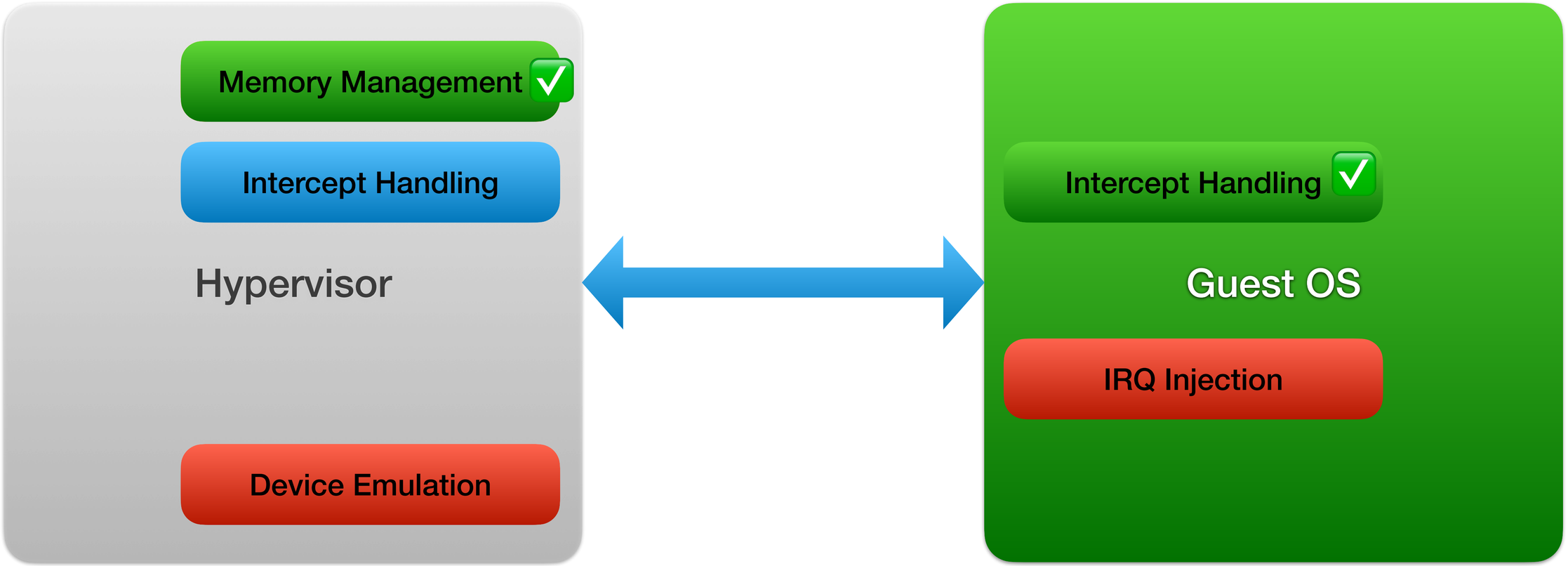
SUSE

Hypervisor Guest Interface



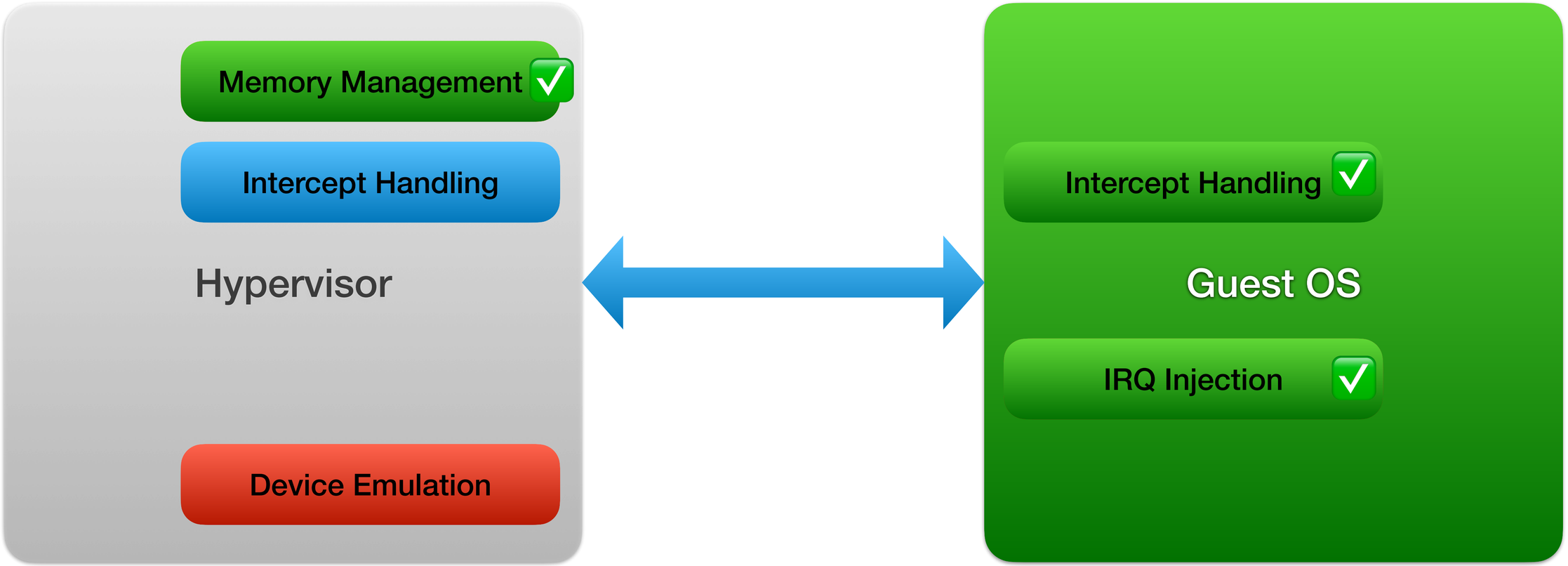
SUSE

Hypervisor Guest Interface



SUSE

Hypervisor Guest Interface



SUSE

Device Emulation

Device Emulation

- All HV-emulated are treated as insecure
- Some devices carry security sensitive state: TPM
- Two approaches:
  - Device driver hardening
  - In-guest Paravisor

SUSE

Device Driver Hardening

- Often done via device driver fuzzing
- Implemented using HV-side fuzzers
  - Had some success in finding bugs in device drivers
  - Patches are sent to the Linux kernel for device driver hardening
- But, overall, a difficult approach which is never finished

In-Guest Paravisor

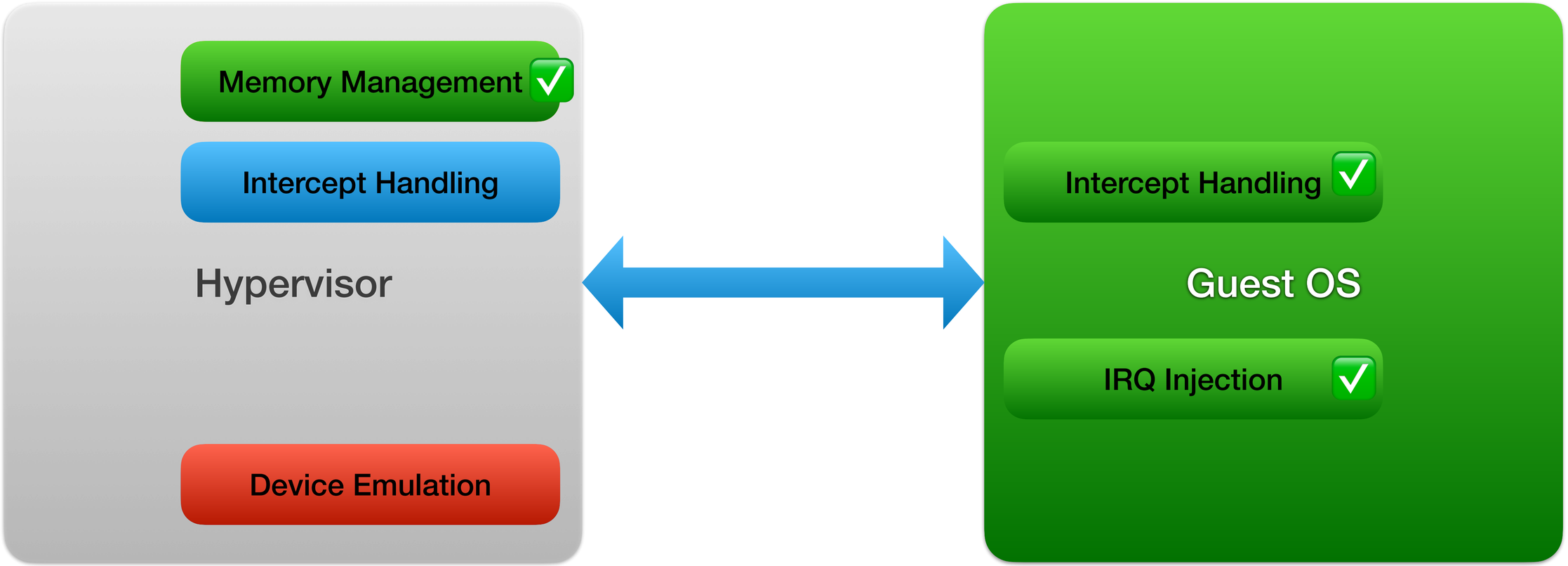
- Uses hardware capabilities for isolation within TEE
  - AMD SEV-SNP VM Privilege Levels (VMPLs)
- VMPLs allow memory separation within an SEV-SNP guest
- 4 Levels - each with its own CPU state
- Use cases:
  - Make guest memory inaccessible to OS
  - Securely move HV functionality into TEE

SUSE

OS Implications

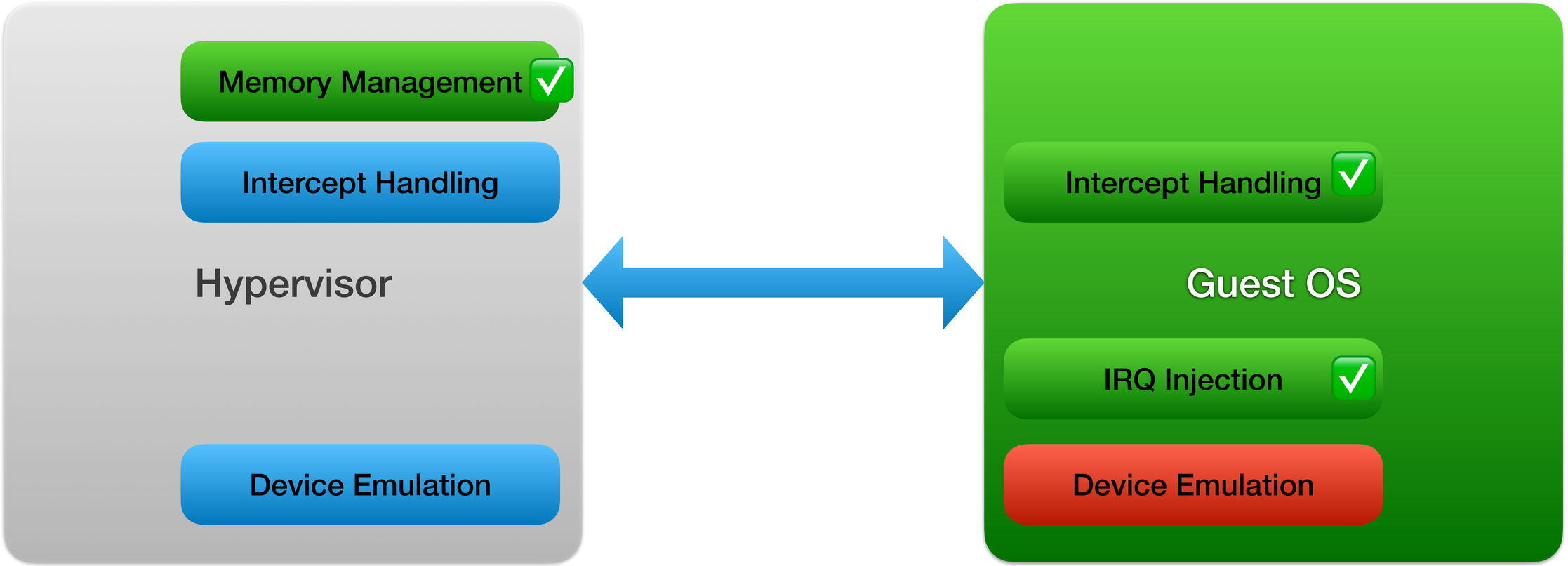
- OS needs to harden device drivers
  - Involves continuous fuzzing and code review
- For paravisor support OS needs implement PV calls to paravisor
  - Specific protocol using shared memory
  - Additional protocols for emulated devices

Hypervisor Guest Interface

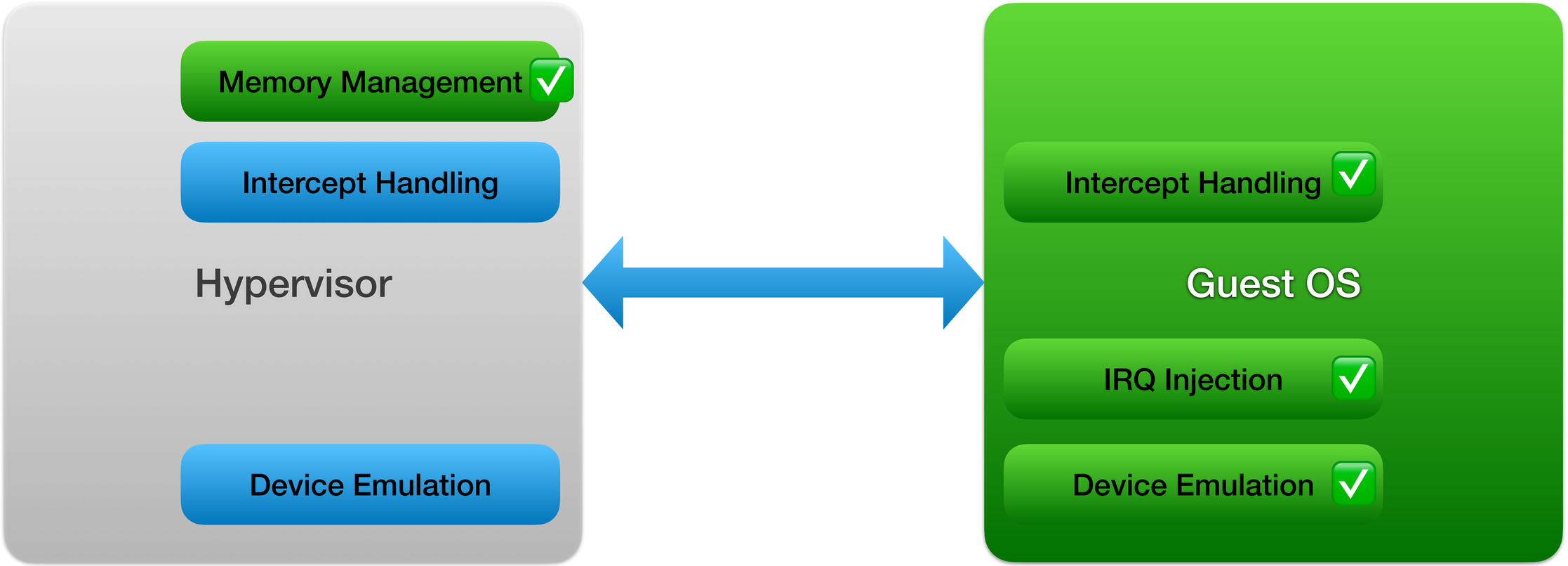


SUSE

Hypervisor Guest Interface



Hypervisor Guest Interface



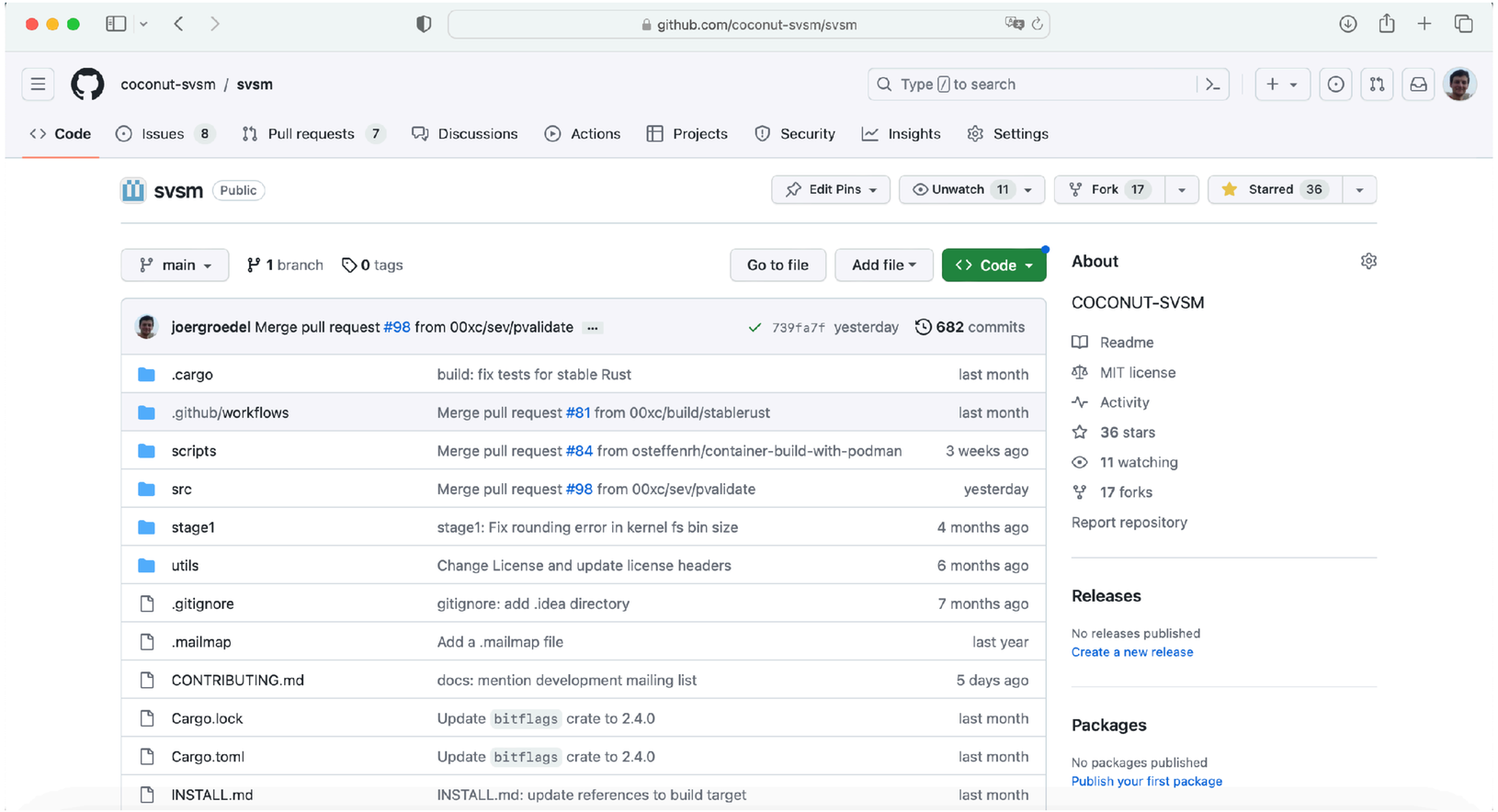
SUSE

COCONUT Secure VM Service Module

- COCONUT Secure VM Service Module (SVSM) currently under development
- OS-level project written in stable Rust
- Will support unprivileged exection mode (CPL3)
- First use-case: Secure TPM 2.0 emulation for CoCo guests
  - Needed for attestation
- Can emulate more devices in the future
- Allows to move some CoCo specifics from OS into Paravisor

SUSE

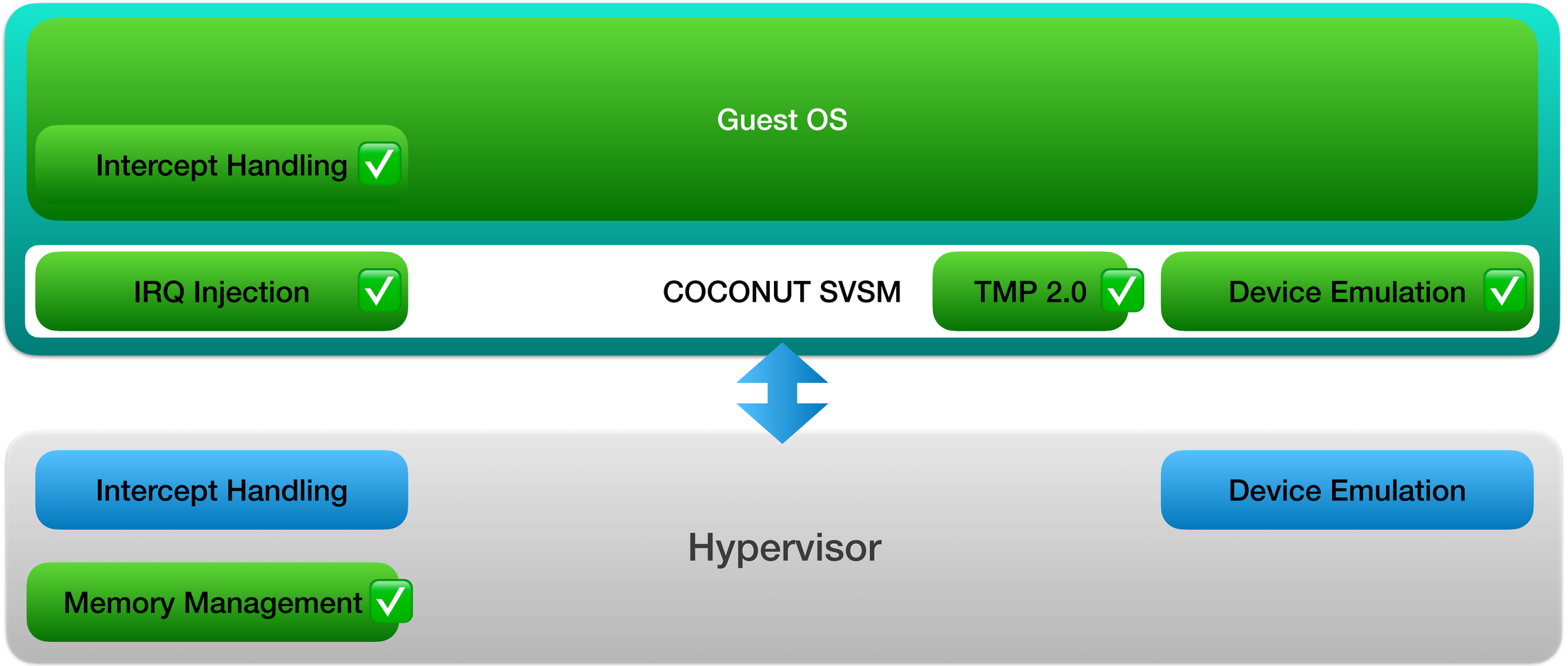
COCONUT Secure VM Service Module



https://github.com/coconut-svsm/svsm/

SUSE

CoCo Virtualization Stack Vision



SUSE

Thank You! Questions?