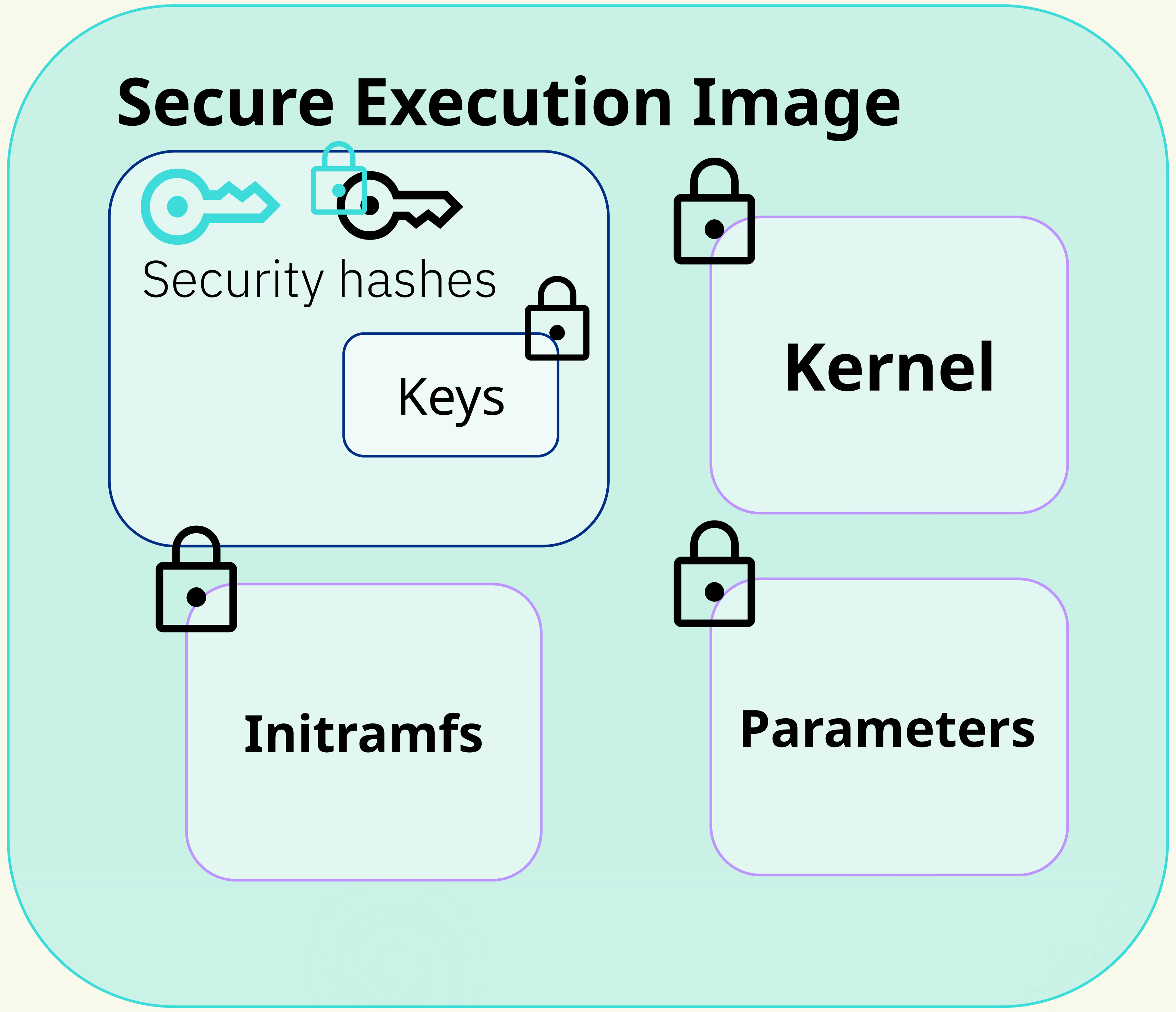
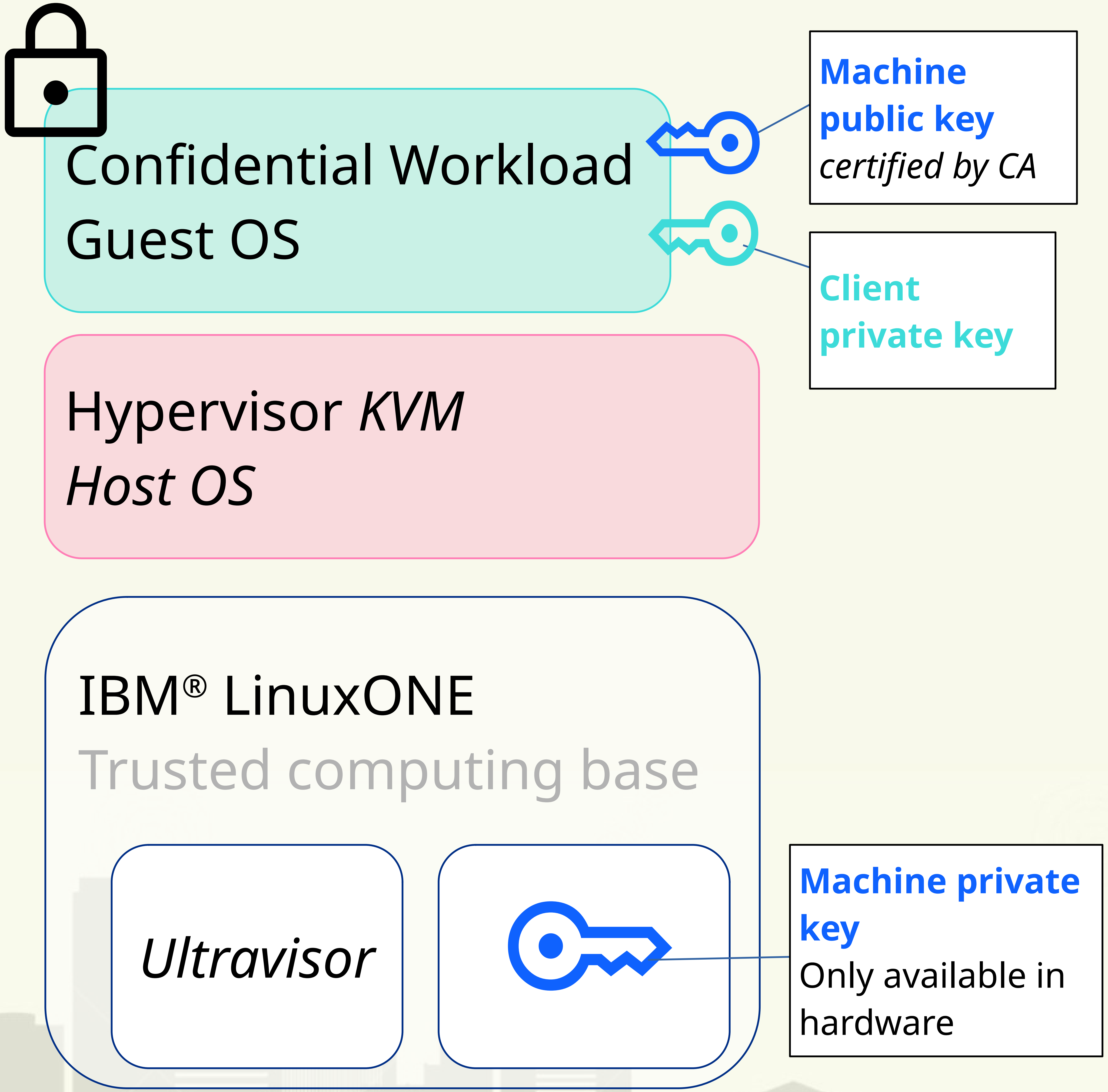
Linux Plumbers Conference | Richmond, VA | Nov. 13-15, 2023

Secure I/O


Secure Execution Image
Security hashes
Keys
Kernel
Initramfs
Parameters


Confidential Workload
Guest OS
Machine public key
certified by CA
Client private key
Hypervisor KVM
Host OS
IBM® LinuxONE
Trusted computing base
Ultravisor
Machine private key
Only available in hardware

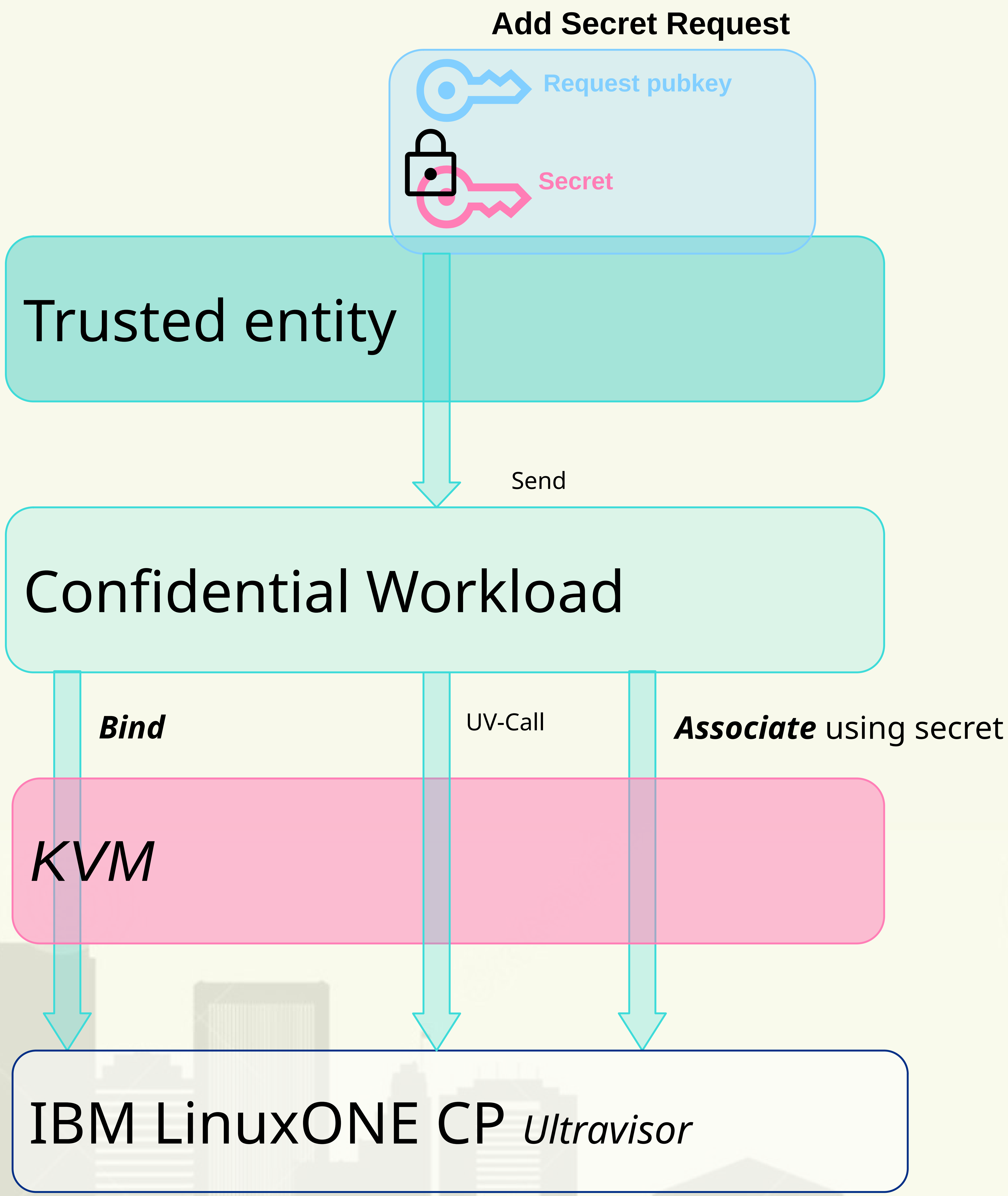
Connecting an EP11 cryptography co-processor to an IBM Secure Execution Guest

Goal
Ensure that KVM/Host cannot get any key material

Bind
- Binds an EP11 exclusively to a SE-guest

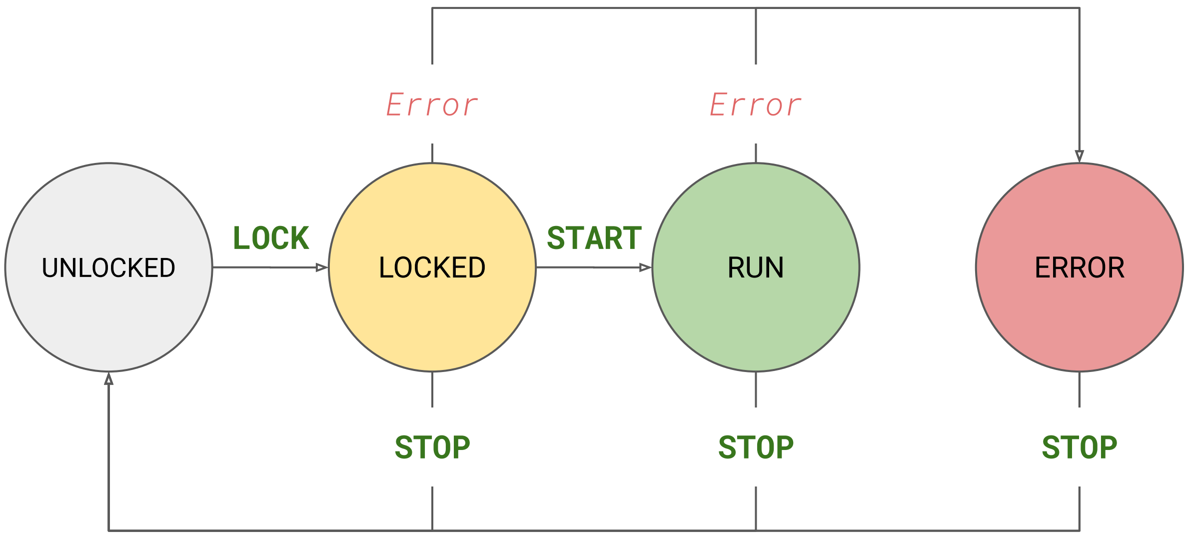
Associate
- Create a session
    - **Secret** defines session
- All key material is session bound
    → Only usable in this session
- Sessions can be continued later

Association Secret
- Encrypted secret, created on another trusted machine
- Secrets are managed by UV per SE guest instance



TDISP State Machine

- One state machine per TDI
- State transitions
  - TDISP command from the host
  - Device or function reset
  - Error condition



State	Device Config Changes?	DMA/MMIO	Device hold confidential data?	Usage
UNLOCKED	Yes	Yes - Not Confidential	No	Legacy
LOCKED	No	No	No	Verification by TVM
RUN	No	Yes - Confidential	**Yes**	TDI in use by guest
ERROR	No	No	**Yes**	Fatal Error - Confidential data wiped

CoVE-IO

RISC-V Secure I/O

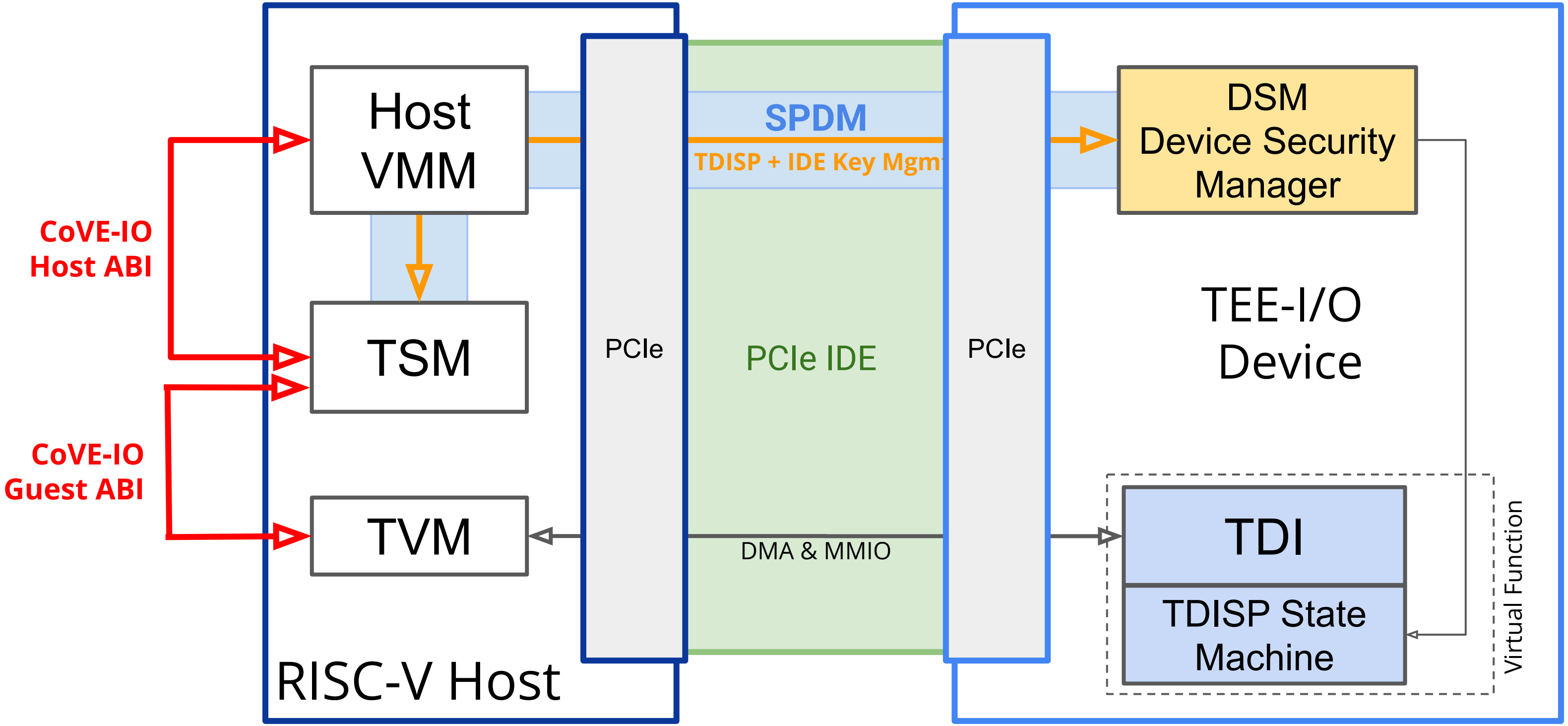
Jiewen Yao (Intel) / Samuel Ortiz (Rivos)

[RISC-V AP-TEE-IO TG]

Rivos

CoVE-IO

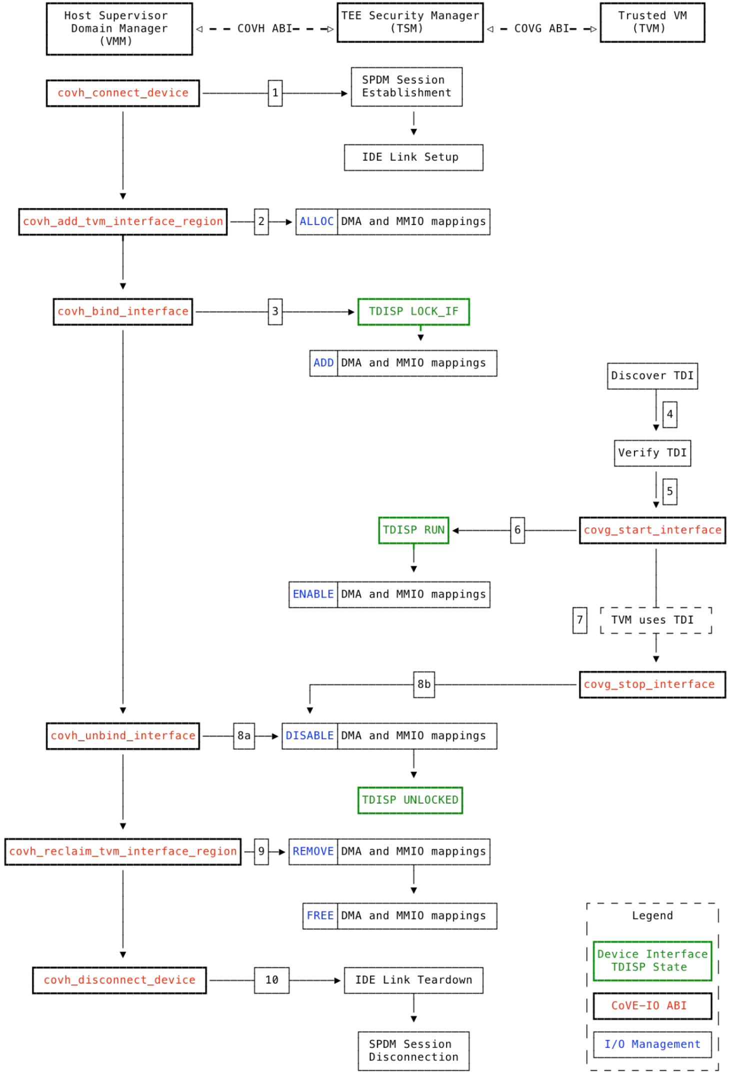
- Confidential Virtual machine Extensions for I/O
- Extension of the [RISC-V Confidential Computing ISA]() (a.k.a. CoVE)



Rivos

CoVE-IO ABI and Flows

- **connect() → bind() → start()**
- **TSM owns the SPDM session**
  - And manages TDISP and the IDE keys
- **TSM manages DMA & MMIO mappings**
- **Host VMM owns the physical device**
  - Host initiates the SPDM session establishment
  - VMM →TSM: **`covh_connect_device()`**
- **Host VMM binds the device interface**
  - VMM →TSM: **`covh_bind_interface()`**
- **Guest verifies the bound interface**
- **Guest starts the interface**
  - Guest →TSM: `covh_start_interface()`



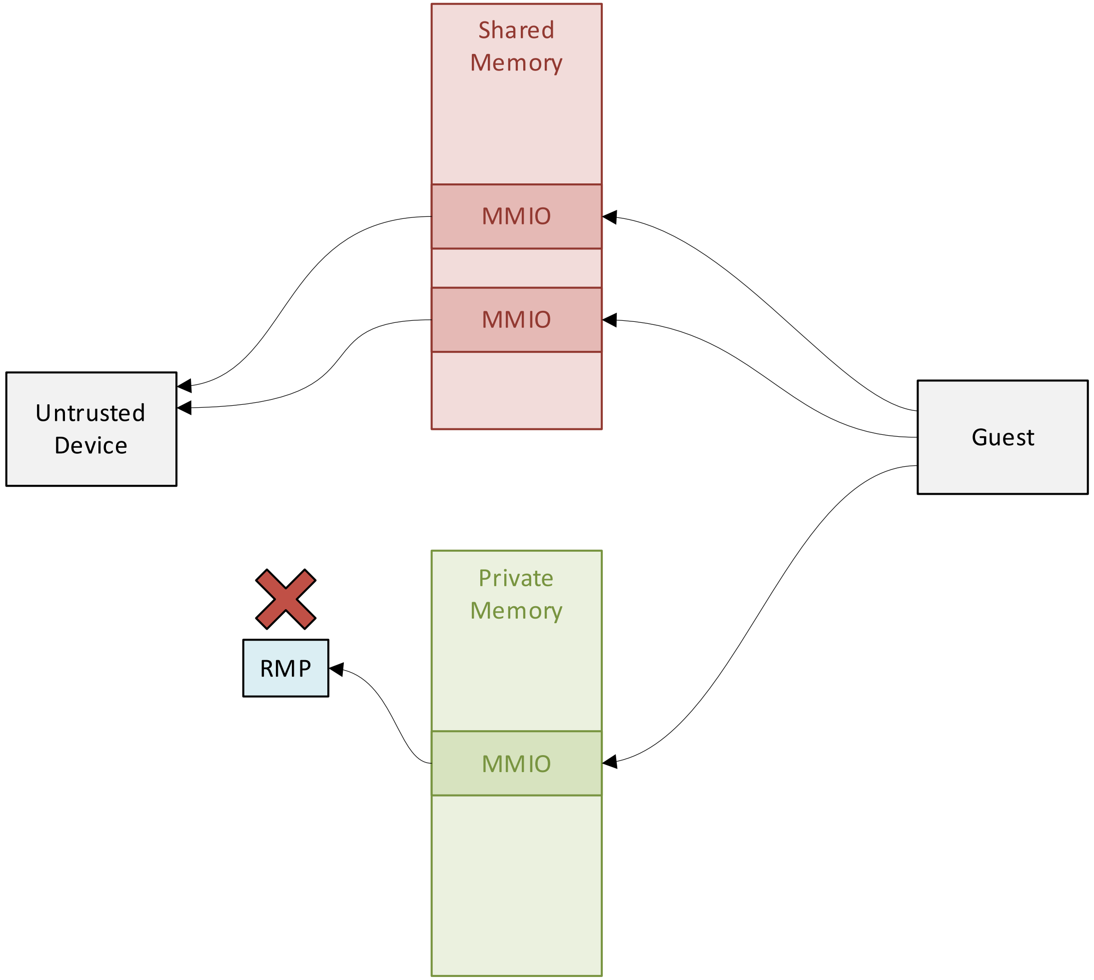
Rivos

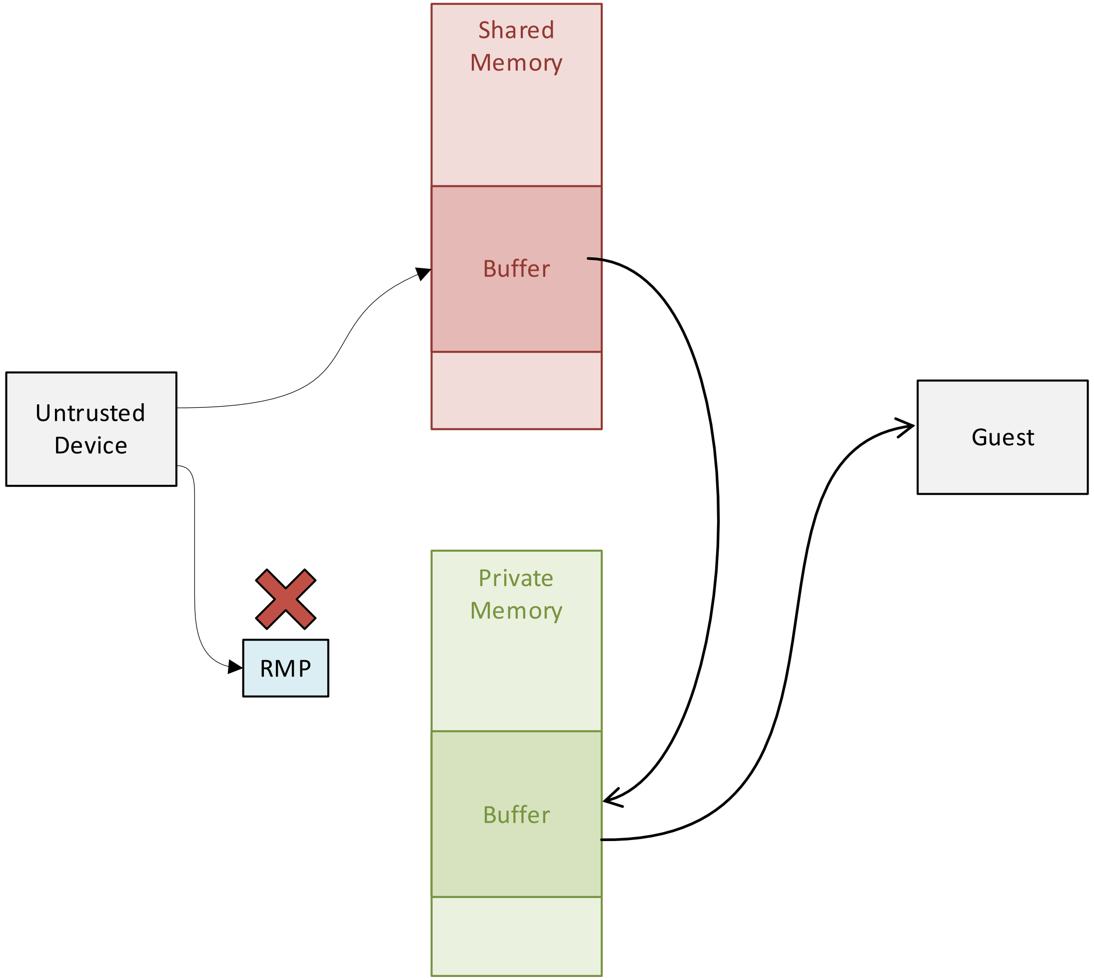
SEV-TIO: Trusted I/O for SEV-SNP Guests

Jeremy Powell & Tom Lendacky

together we advance_

SEV-SNP – Untrusted Devices

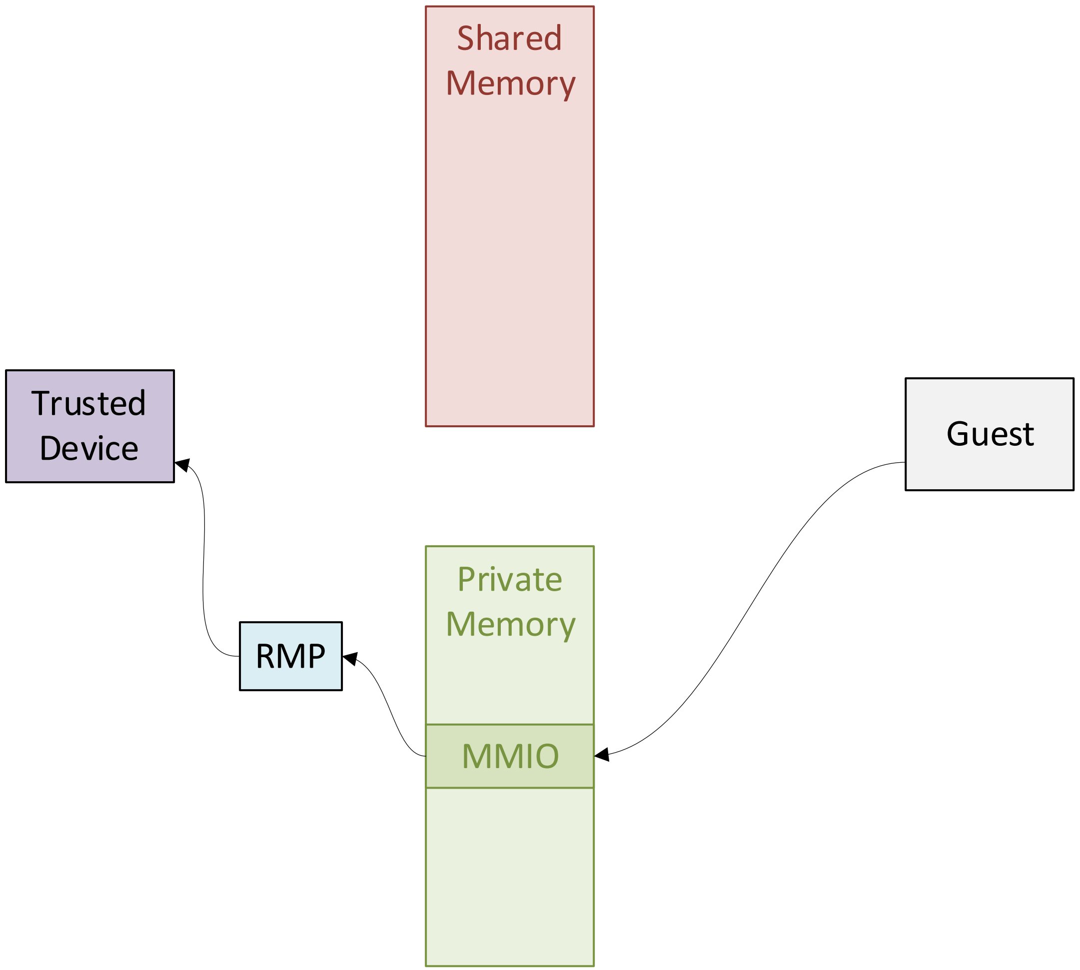




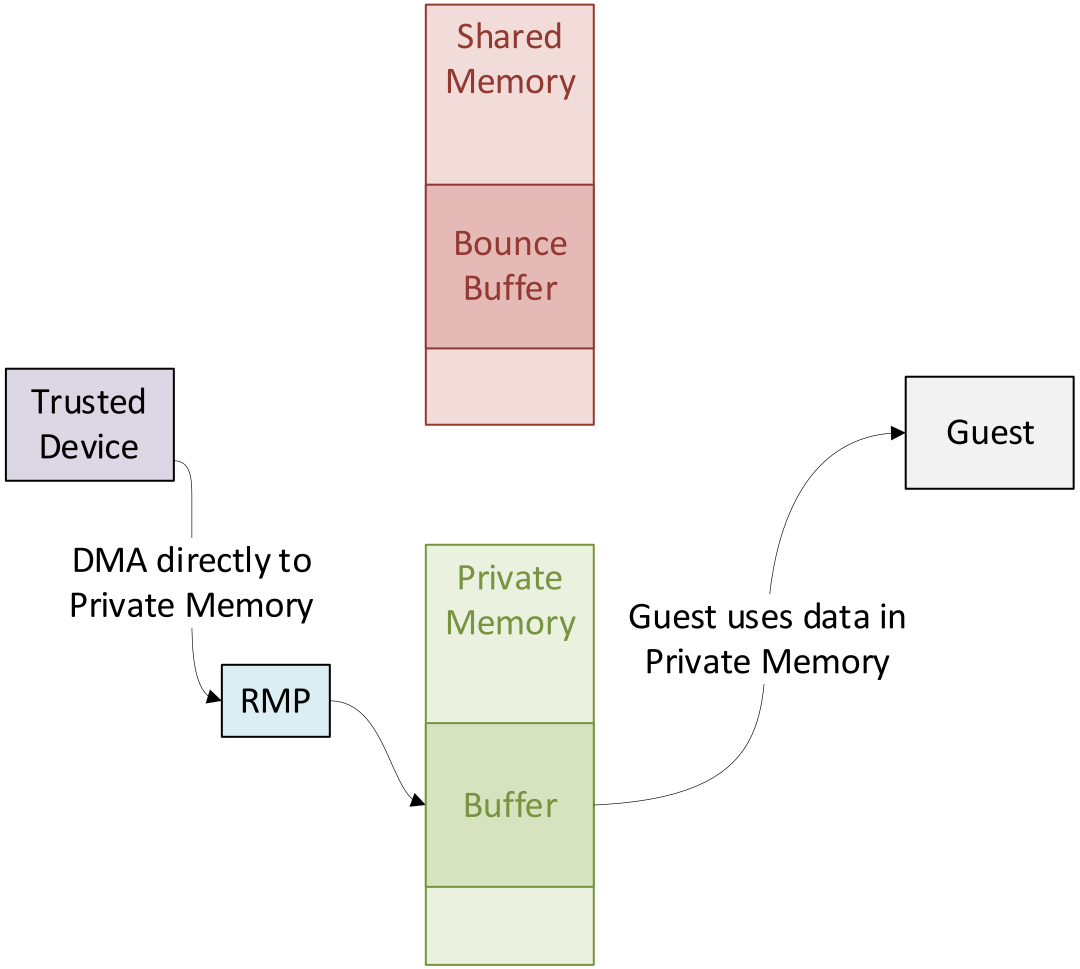
MMIO Cannot Map to Private Memory

DMA to Private Memory Blocked

SEV-TIO – Trusted Devices



MMIO Can Map to Private Memory



DMA to Private Memory Allowed

Guest & TDI Lifecycle

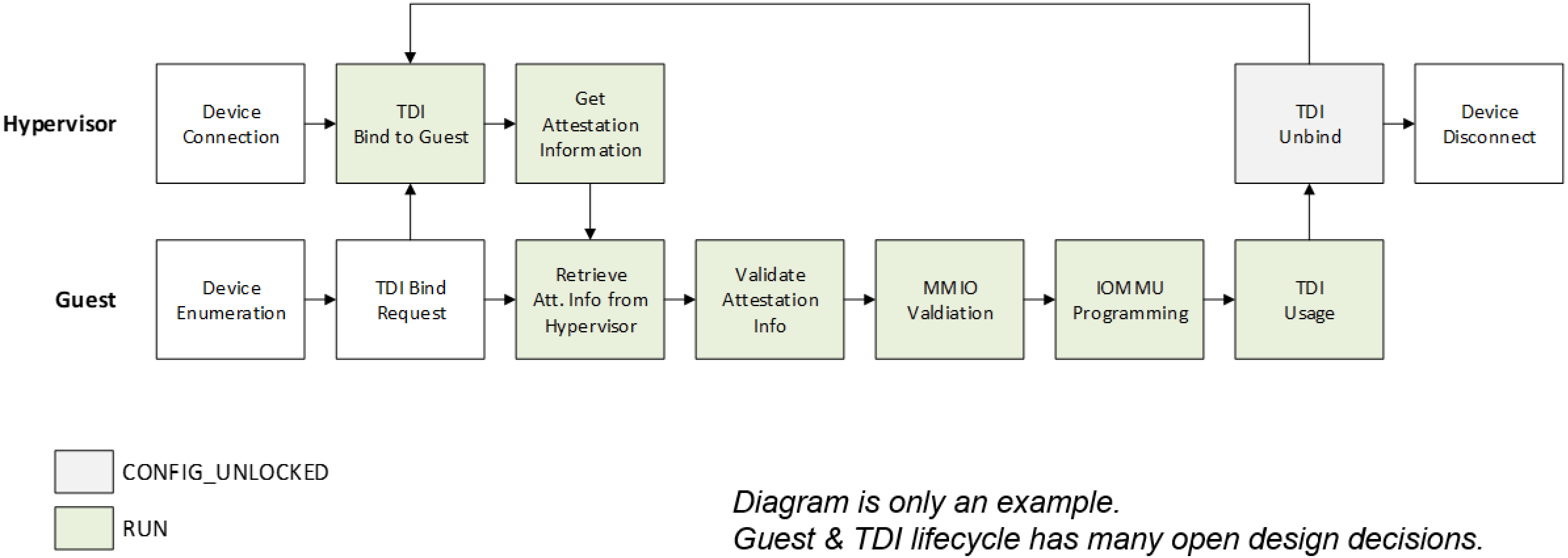
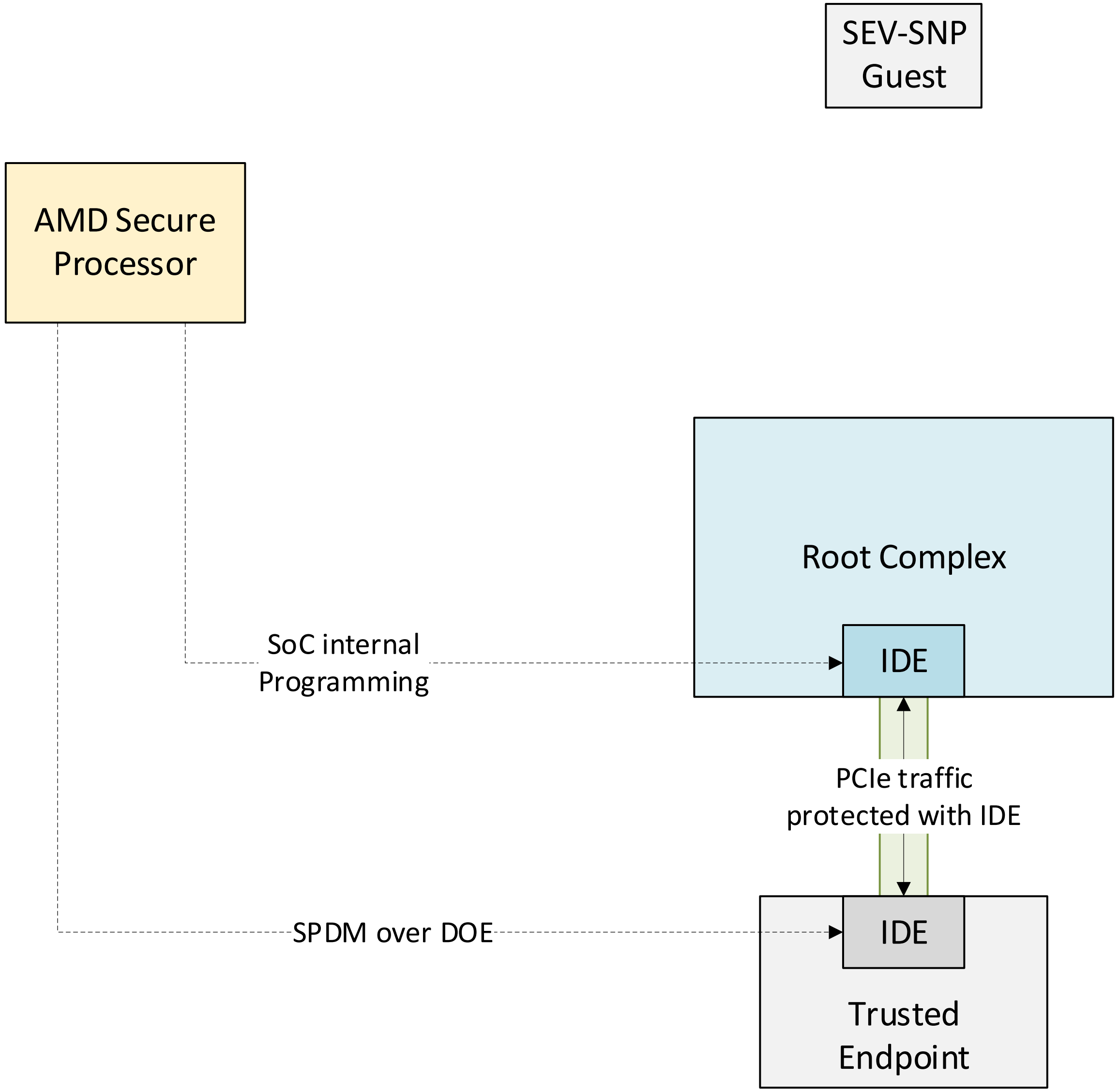


Diagram is only an example.
Guest & TDI lifecycle has many open design decisions.

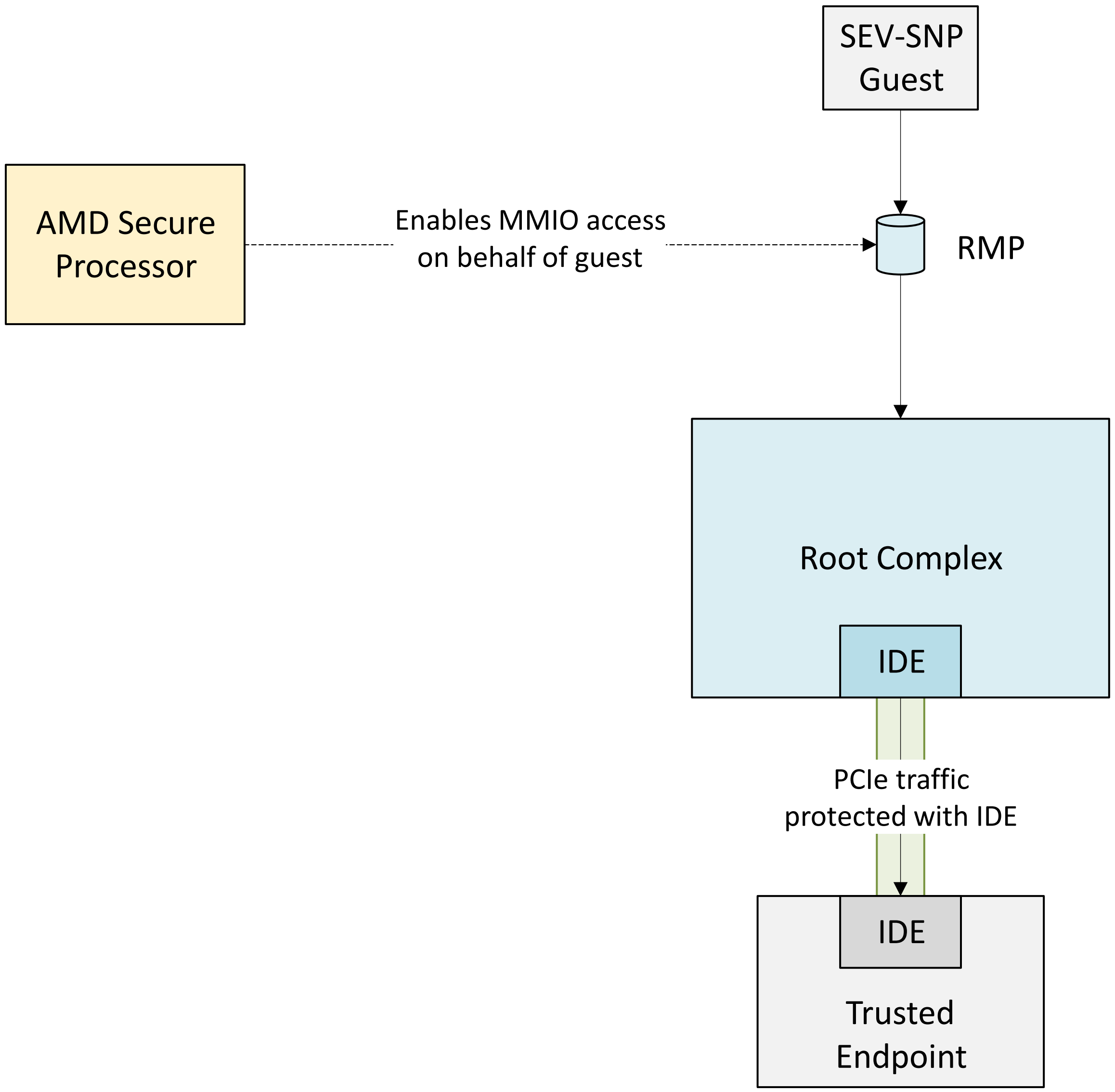
Protecting PCIe Traffic

- Secure Protocol and Data Model (SPDM)
  - Protocol connecting device to ASP
  - Encrypted and integrity protected
  - Control path for configuring trusted device
- PCIe Integrity and Data Encryption (IDE)
  - Confidentiality and authenticity between PCIe ports
  - Keyed by ASP using SPDM
  - PCIe traffic protected at transaction layer



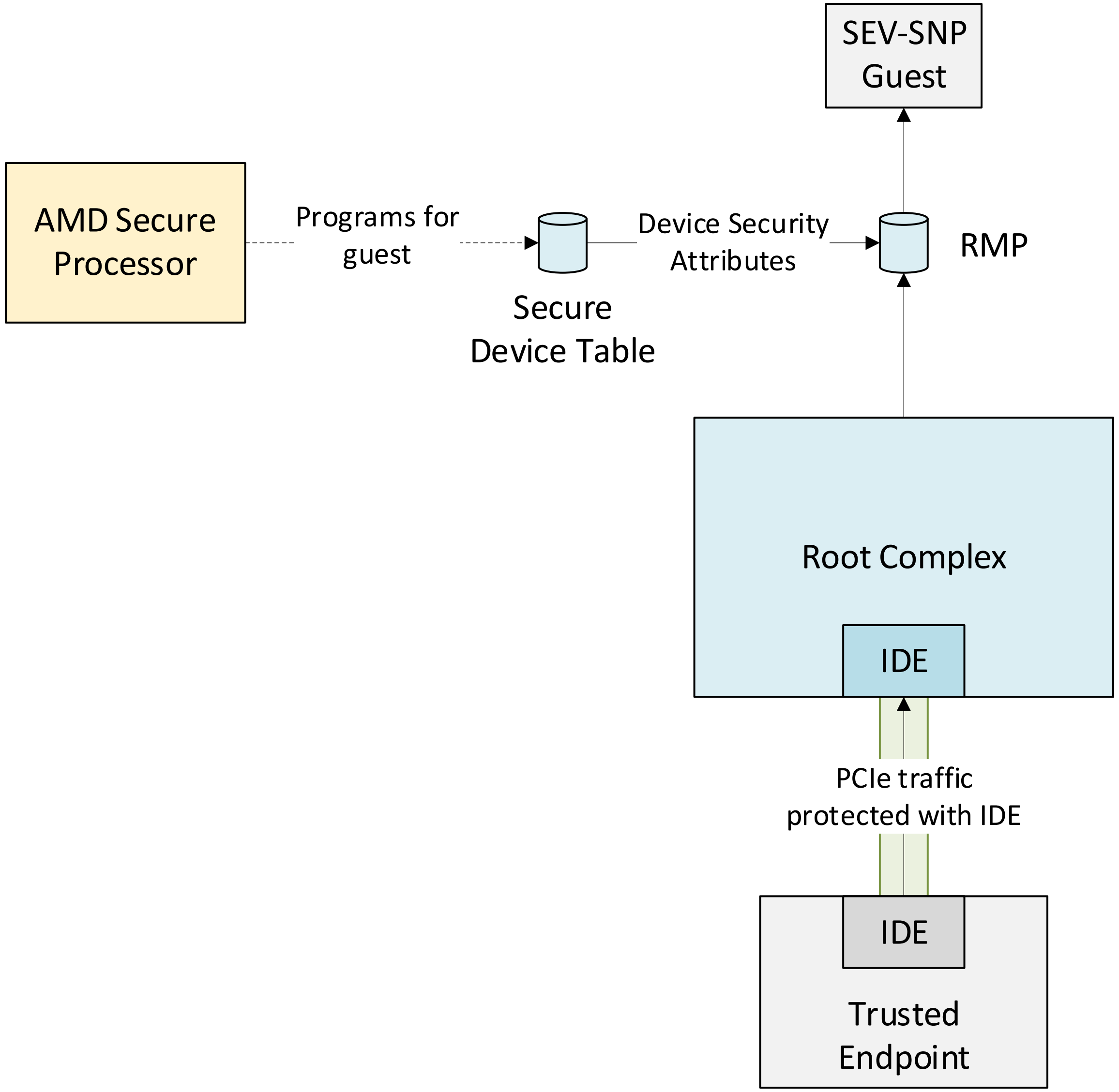
Enabling Secured MMIO

- Private MMIO ranges
  - Located in private guest memory
  - ASP verifies configuration (system physical ⇔ device mapping)
  - Configures RMP for secure MMIO
- Guest MMIO access
  - Existing RMP checks prevent remapping attacks
  - MMIO access is routed to appropriate device through the device's IDE stream



Enabling Secured DMA

- Secure Device Table (SDT)
  - Additional RID-indexed table
  - Associates the RID with the guest
  - ASP programs on behalf of guest
- Access control on DMA
  - Hardware routes and tags traffic per SDT entries
  - IOMMU performs RMP checks like CPU MMU
  - Guest manages DMA targets like any other private guest memory (i.e., using PVALIDATE instruction)



AMD
together we advance_

Disclaimer

The information contained herein is for informational purposes only, and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale.

AMD's products are not designed, intended, authorized or warranted for use as components in systems intended for surgical implant into the body, or in other applications intended to support or sustain life, or in any other application in which the failure of AMD's product could create a situation where personal injury, death, or severe property or environmental damage may occur. AMD reserves the right to discontinue or make changes to its products at any time without notice.

AMD does not provide a license/sublicense to any intellectual property rights relating to any standards, including but not limited to any audio and/or video codec technologies such as AVC/H.264/MPEG-4, AVC, VC-1, MPEG-2, and DivX/xVid.

AMD

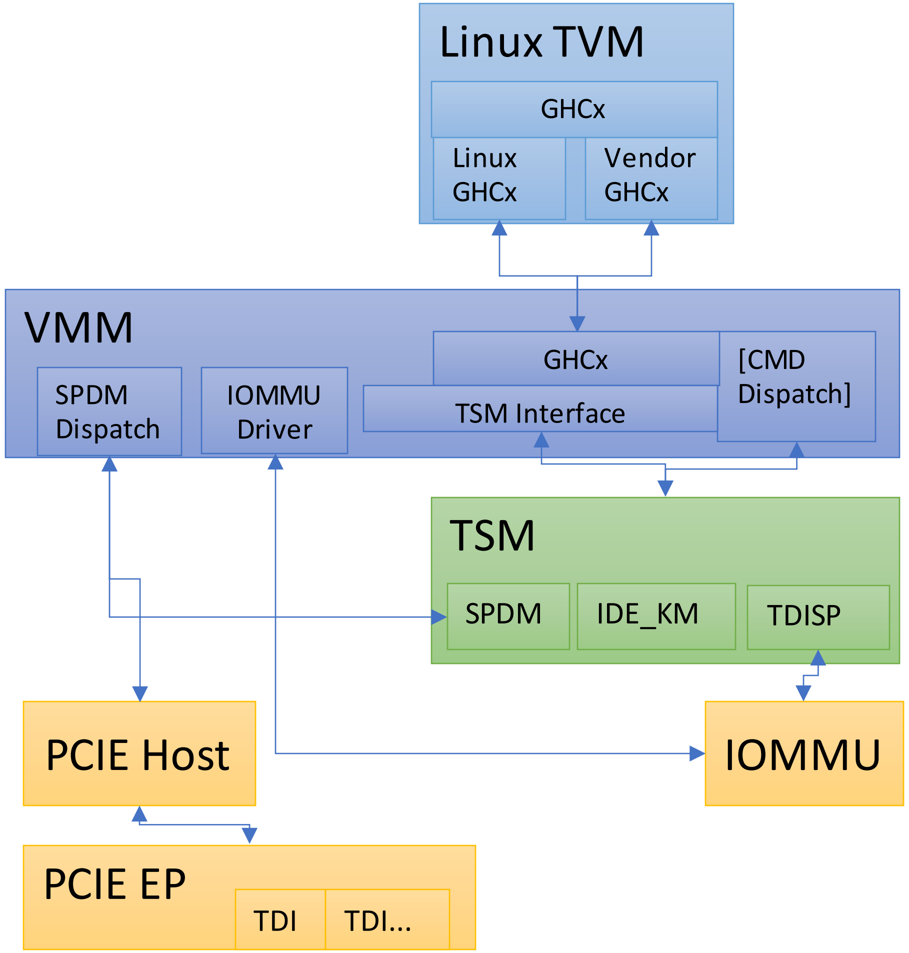
Framing Questions

- What if there was a standard Platform Secure I/O interface definition?
  - How much deviation would upstream tolerate from that standard?
- Given there is no standard and multiple vendor interface proposals in flight what is Linux's role in mitigating differentiation?
- Given the fundamental complexity of Secure I/O where do we start as a community?

Towards a Linux Secure I/O Interface Definition

- Guiding principles:
  - Find the coarsest grained (externalizes the least amount of complexity to Linux) and minimal set of verbs to transition a device-instance into and out of Secure I/O operation. Advocate for vendors to move their interfaces to that standard.
  - Start with the most ruthlessly simple implementation, but no simpler, and incrementally evolve to address more use cases.

Apply the Principles



- SPDM, IDE, and TDISP protocol abstracted behind TSM
- Common TSM verbs: Connect, Bind, Unbind, Disconnect, Info (Certs, Measurements, Report)
- Linux GHCx for common guest to host operations
- Dispatcher(s) to help limited TSM execution environments offload protocol handling complexity