SVSM – VM Privilege Level Instantiation and Execution

Tom Lendacky

AMD
together we advance_

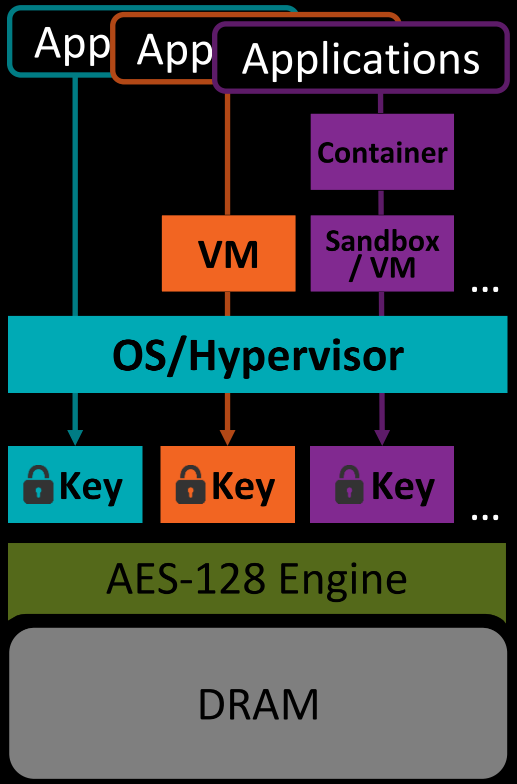
Agenda

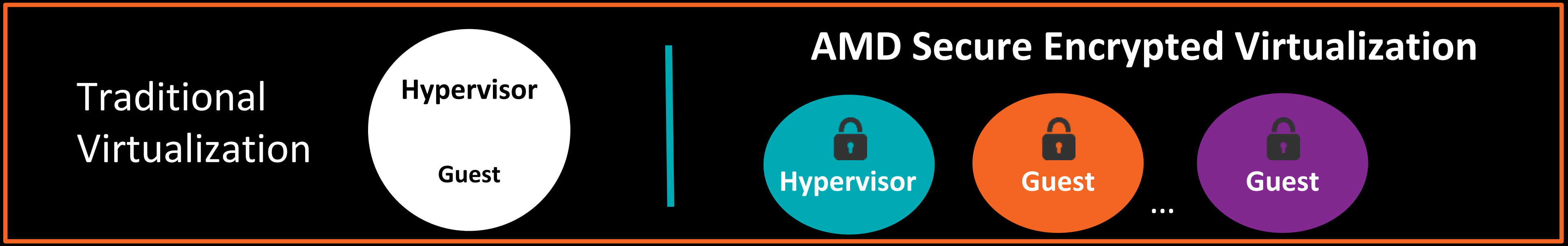
- Review
  - Secure Encrypted Virtualization (SEV)
  - Secure Encrypted Virtualization – Encrypted State (SEV-ES)
  - Secure Encrypted Virtualization - Secure Nested Paging (SEV-SNP)
  - VM Privilege Level (VMPL)
  - Secure VM Service Module (SVSM)

- Instantiation and Execution
  - Explore the options:
    - Creating VMPL levels
    - Switching a vCPU between VMPLs

SEV Review

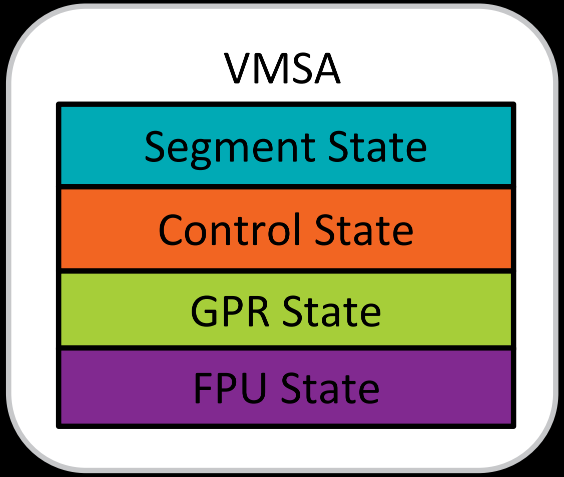
- Protects VMs/Containers from each other, administrator tampering, and untrusted Hypervisor
- One key for Hypervisor and one key per VM or VM/Sandbox with multiple containers
- Cryptographically isolates the hypervisor from the guest VMs
- Integrates with existing AMD-V™ technology
- System can also run unsecure VMs

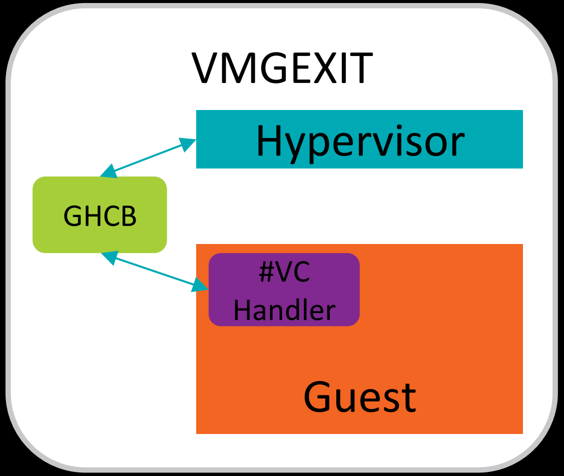



Traditional Virtualization
Hypervisor
Guest
AMD Secure Encrypted Virtualization
Hypervisor
Guest
...
Guest

SEV-ES Review

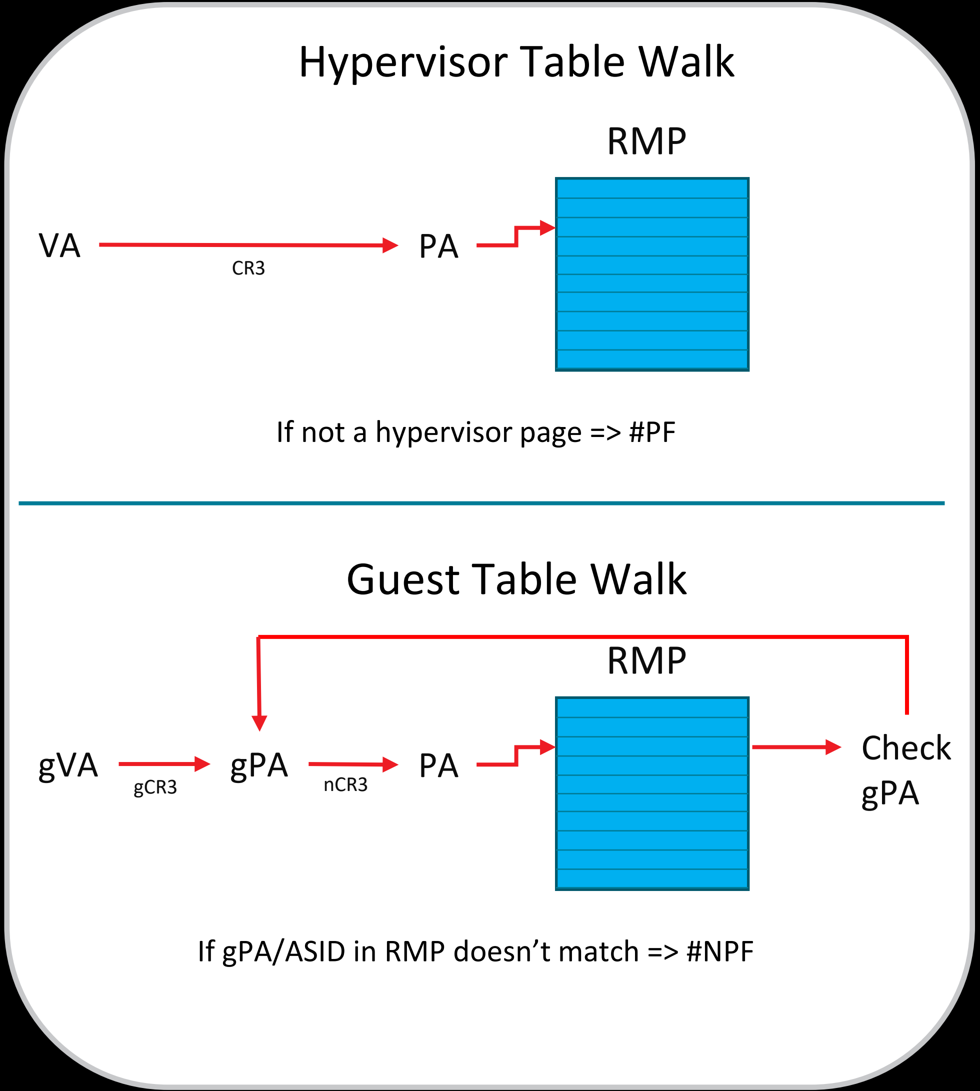
- Guest register state protection
  - Initialized with known state (Initial Processor State)
  - Encrypted and measured as part of the SEV LAUNCH process
  - Integrity check performed on each VMRUN
  - World switches now swap ALL register state
- VMCB under SEV-ES
  - Control Area (VMCB) and Save Area (VMSA) now separated
  - VMCB now points to VMSA
  - VMSA extended to save more state
- Guest-Hypervisor Communication Block (GHCB)
  - Allows guest ←→ hypervisor communication of the state needed to satisfy the guest service request
  - Shared (un-encrypted) page between the hypervisor and the guest
  - GHCB specification
    - Defines the format of the GHCB and how to communicate with the hypervisor





SEV-SNP Review

- Secure Nested Paging
  - Next step in the evolution of SEV
  - SEV/SEV-ES provides Confidentiality
    - SEV – encryption of VM memory
    - SEV-ES – adds encryption of VM registers
  - SEV-SNP builds on SEV-ES and adds Integrity Protection
    - Prevents replay attacks, corruption attacks, remapping attacks
    - Utilizes the Reverse Map Table (RMP) and RMPUPDATE instruction to track:
      - Page Ownership: Hypervisor, Guest/PSP
      - Page Size: 4KB or 2MB
      - Guest Physical Address and ASID
      - VMSA (can be used with a VMRUN instruction)
        - LAUNCH_UPDATE or RMPADJUST instruction
      - Validation
        - PVALIDATE instruction
        - #VC if validation is changed by the hypervisor

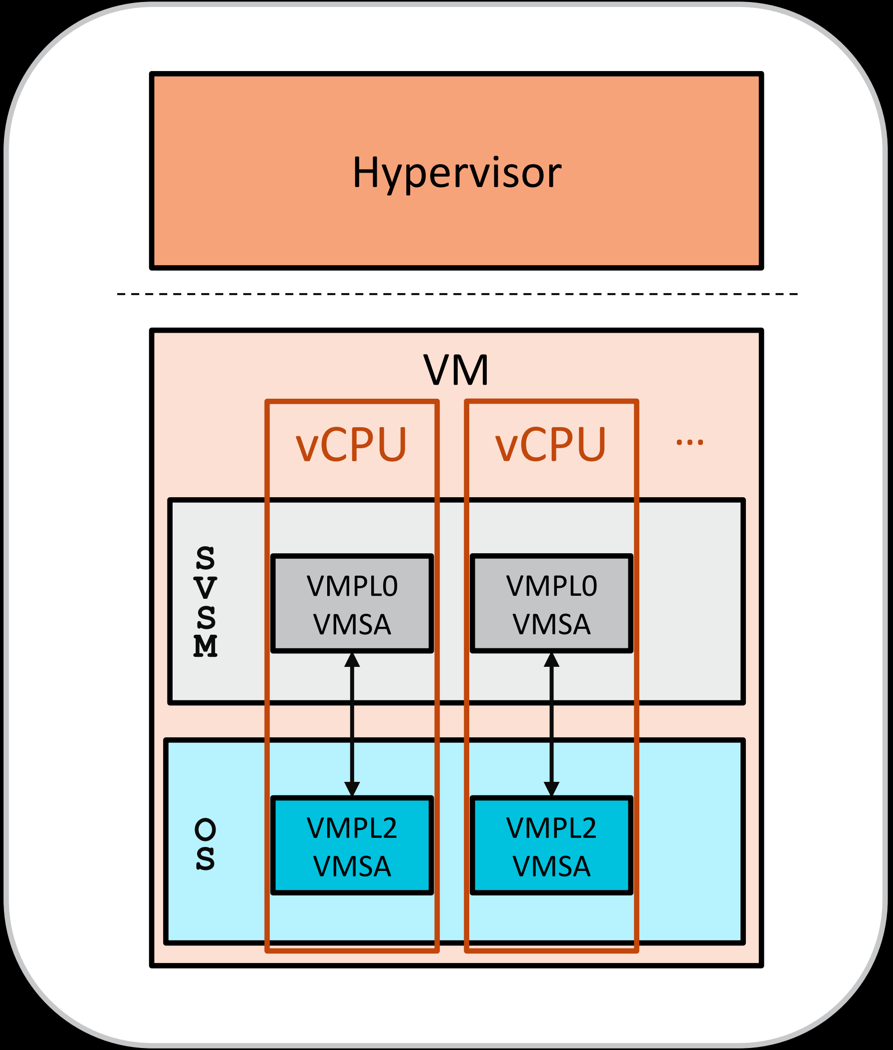


VMPL Review

- Virtual Machine Privilege Level (VMPL)
  - Allows a SEV-SNP guest to divide its address space in up to 4 levels
  - VMPL0 - VMPL3 (VMPL0 being most privileged)
    - Higher privileged VMPL can provide secure services for lesser privileged VMPL
      - e.g., VMPL0 can provide secure services for VMPL2
    - VMPL level is set in the VM Save Area (VMSA) page
    - KVM/Linux SEV-SNP guests run at VMPL0 today
  - Each RMP entry has page permissions for each VMPL level
    - Read, Write, Execute (User/Supervisor), Supervisor Shadow Stack
    - Guest can set permissions for a lesser privileged VMPL using RMPADJUST
      - Only VMPL0 can set the VMSA attribute for use in running a vCPU
    - Allows a higher privileged VMPL to protect itself from a lesser privileged VMPL

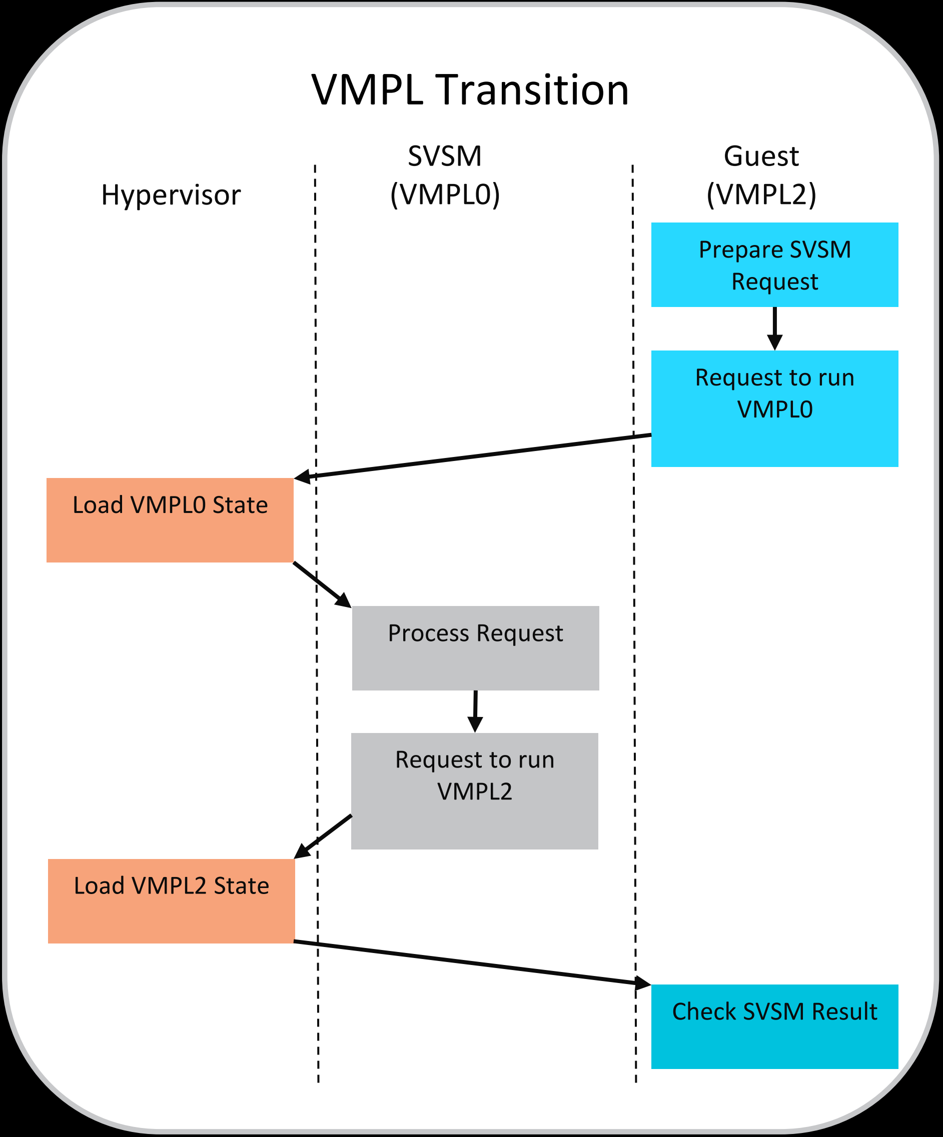
SVSM Review

- Secure VM Service Module (SVSM)
  - Runs at VMPL0
  - Creates VMPL0 VMSA pages for all APs
  - Creates VMPL2 VMSA for the BSP
    - Up to the VMPL2 to create VMSA's at VMPL2 for APs
  - Initiates execution of the VMPL2 target
  - Provides an API to allow VMPL2 to request services from VMPL0
  - Uses:
    - Live Migration
    - vTPM
    - … and more



VMPL Transition

- Create a vCPU at a VMPL level
  - GHCB AP Create hypercall
    - Issued by current VMPL to create the same or new VMPL
- Request to run a VMPL level
  - GHCB Run VMPL hypercall
    - Issued by current VMPL to request running a new VMPL
      - KVM always runs the current VMPL until requested to change
- VMM/Hypervisor is responsible for creating/running the requested VMPL
  - Possible approaches:
    - One VM* and each vCPU is multiple VMPLs
    - One VM* and each vCPU is a dedicated VMPL
    - One VM* per VMPL



* a VM is a KVM object structure created using the KVM_CREATE_VM ioctl

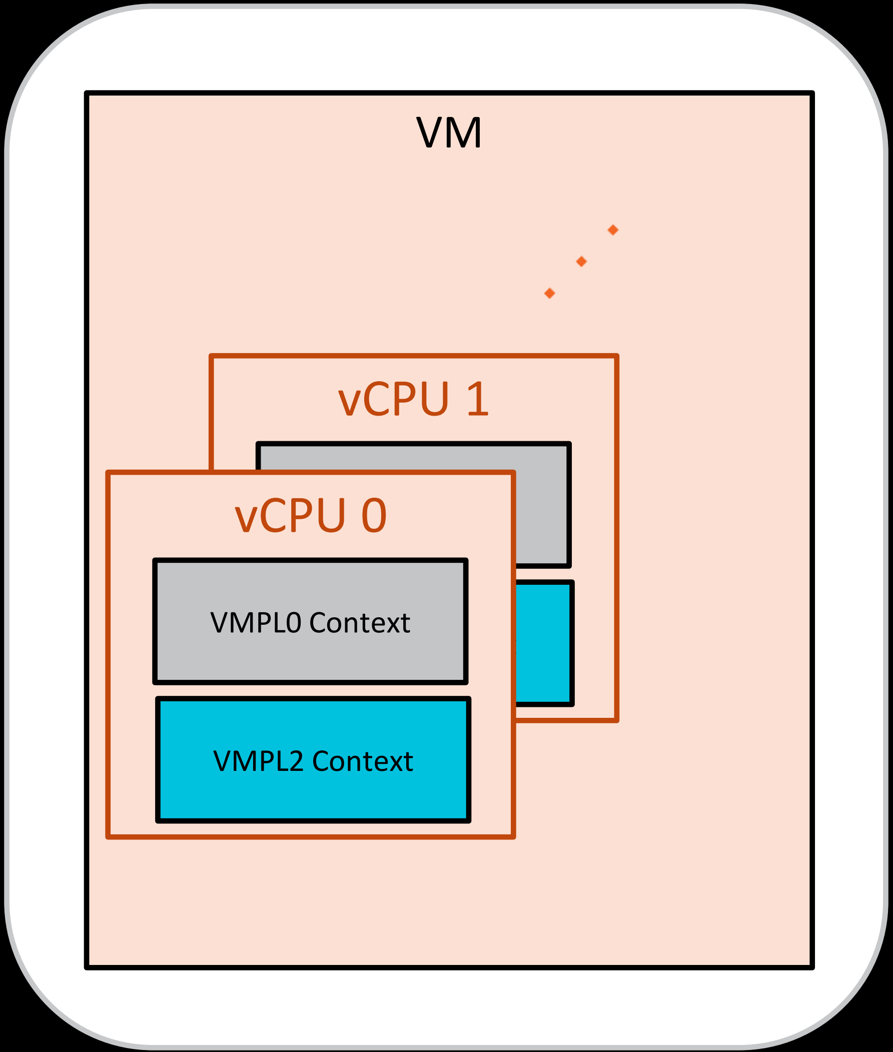
VMPL Instantiation & Execution Approaches

VMPL Instantiation & Execution

- One VM and each vCPU is multiple VMPLs
  - All performed within KVM

  - Instantiation – AP Create
    - Saves VMSA GPA in target vCPU struct
    - Kicks target vCPU
    - Target vCPU validates/translates VMSA GPA
    - Target vCPU creates/resets APIC instance

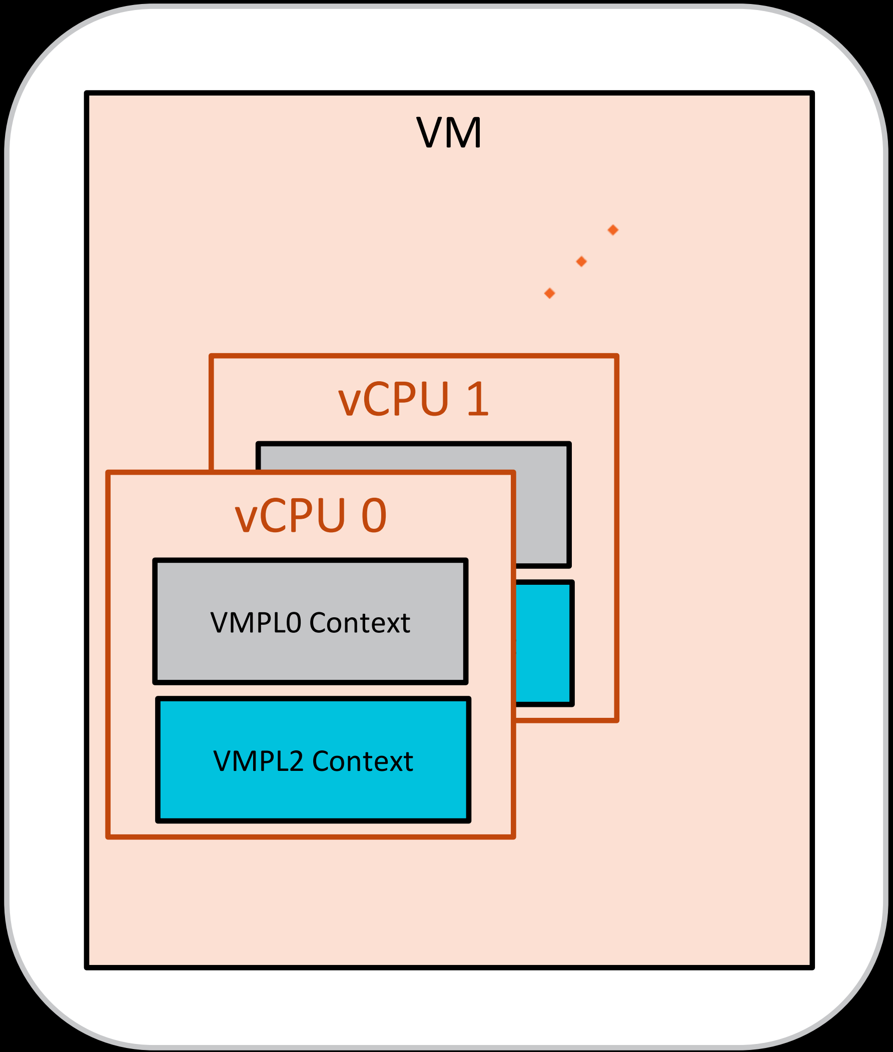
  - Execution – Run VMPL
    - Fields within VM Control Block (VMCB) swapped
      - VMSA SPA
      - GHCB GPA
      - Tracking registers (EFER, CR0, etc.)
    - Next VMRUN now runs a different VMPL



AMD
together we advance_

VMPL Instantiation & Execution…

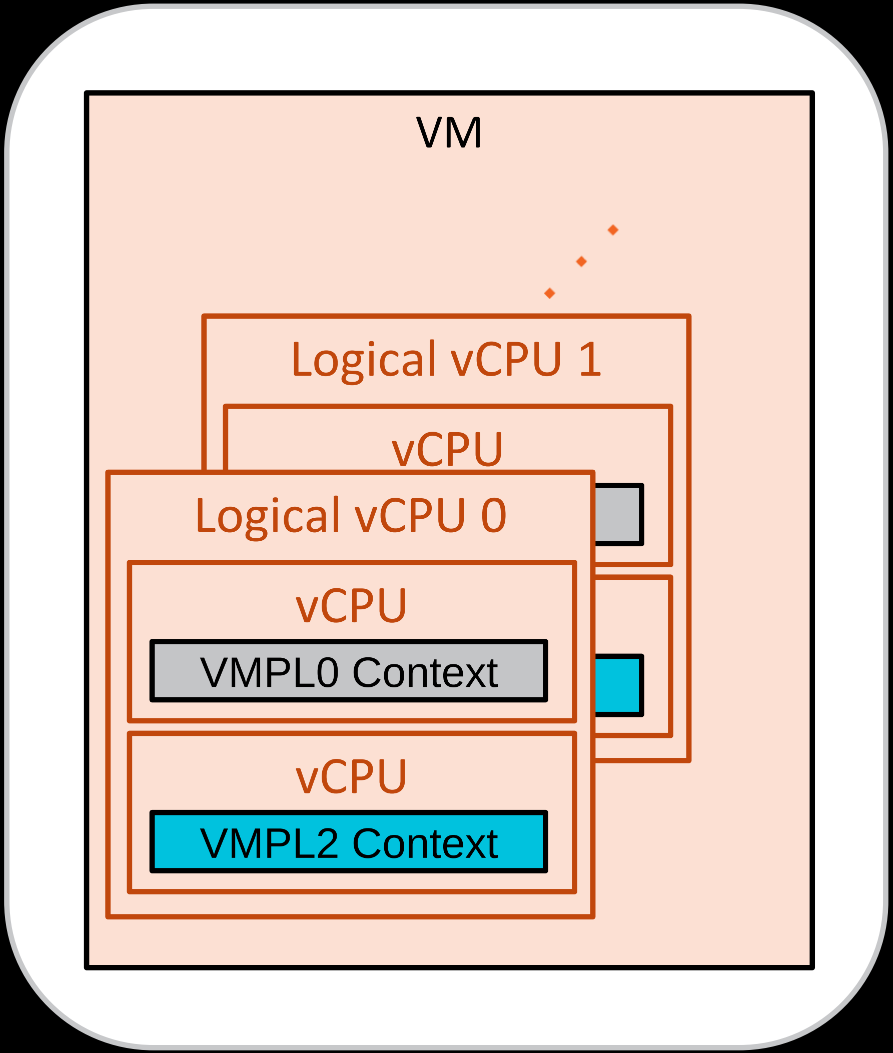
- One VM and each vCPU is multiple VMPLs
  - All performed within KVM
  - Pros
    - Fast
      - AP Create kicks target vCPU
      - Run VMPL stays in the VMRUN loop
    - VMM Agnostic
      - KVM localized changes
      - No VMM API changes
  - Cons
    - Need to save/restore defined set of data
    - APIC Support
      - Multiple APIC instances per vCPU



AMD
together we advance_

VMPL Instantiation & Execution…

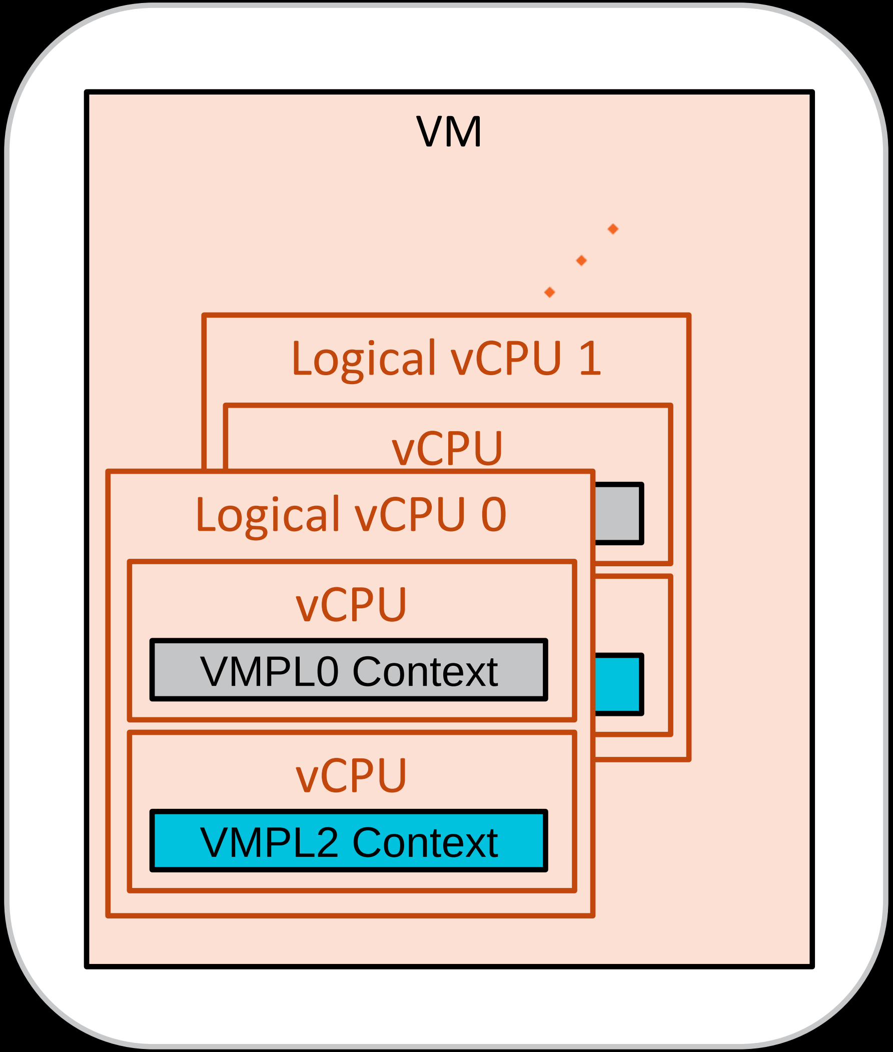
- One VM and each vCPU is a dedicated VMPL
  - Instantiation – AP Create
    - Request sent to VMM
      - VMM creates new vCPU
      - New vCPU validates/translates/sets VMSA GPA
  - Execution – Run VMPL
    - Request sent to VMM
      - VMM pauses the current VMPL vCPU
      - VMM resumes the target VMPL vCPU



AMD
together we advance_

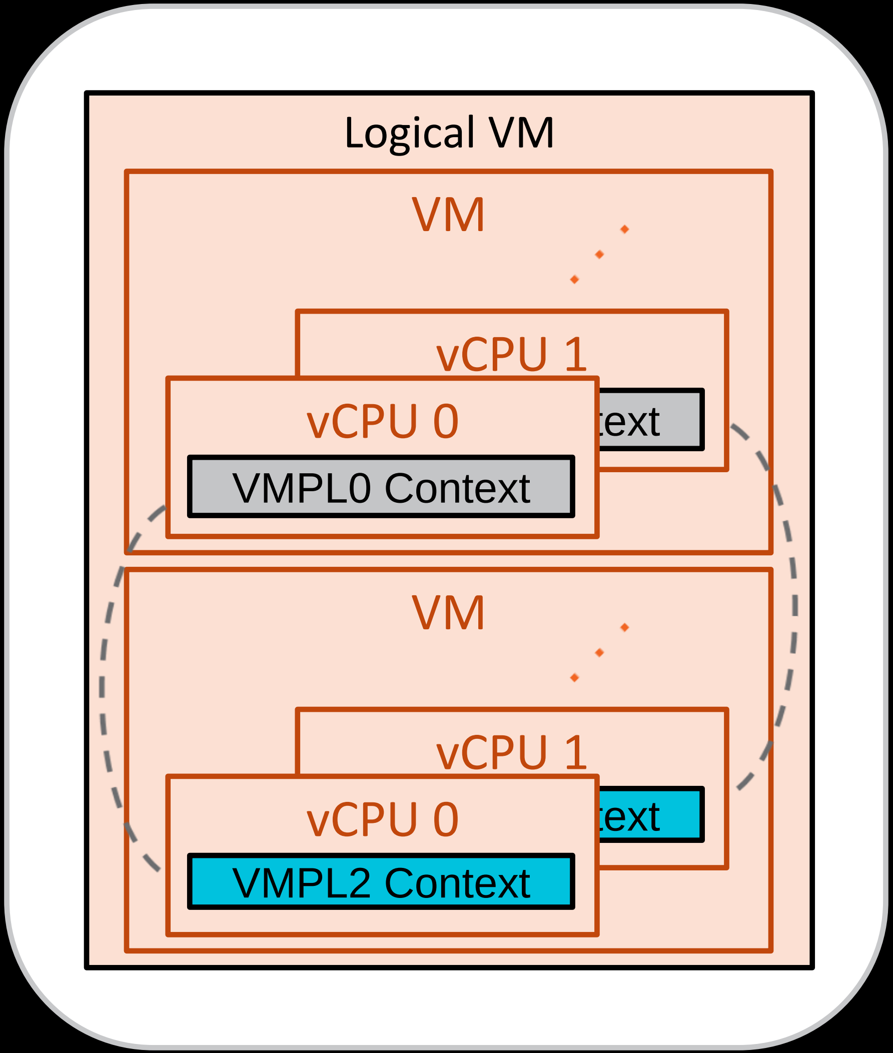
VMPL Instantiation & Execution…

- One VM and each vCPU is a single VMPL
  - Pros
    - Separate vCPU object
      - Maintains separate VMSA, GHCB, APIC, etc. contexts
  - Cons
    - Need to transition to userspace (VMM)
    - Multiple vCPU threads
      - vCPU pinning becomes more difficult
    - Qemu requires vCPU namespace support
      - Affects interrupt routing
    - KVM requires vCPU namespace support
      - Index is APIC ID based
      - Affects interrupt routing



VMPL Instantiation & Execution…

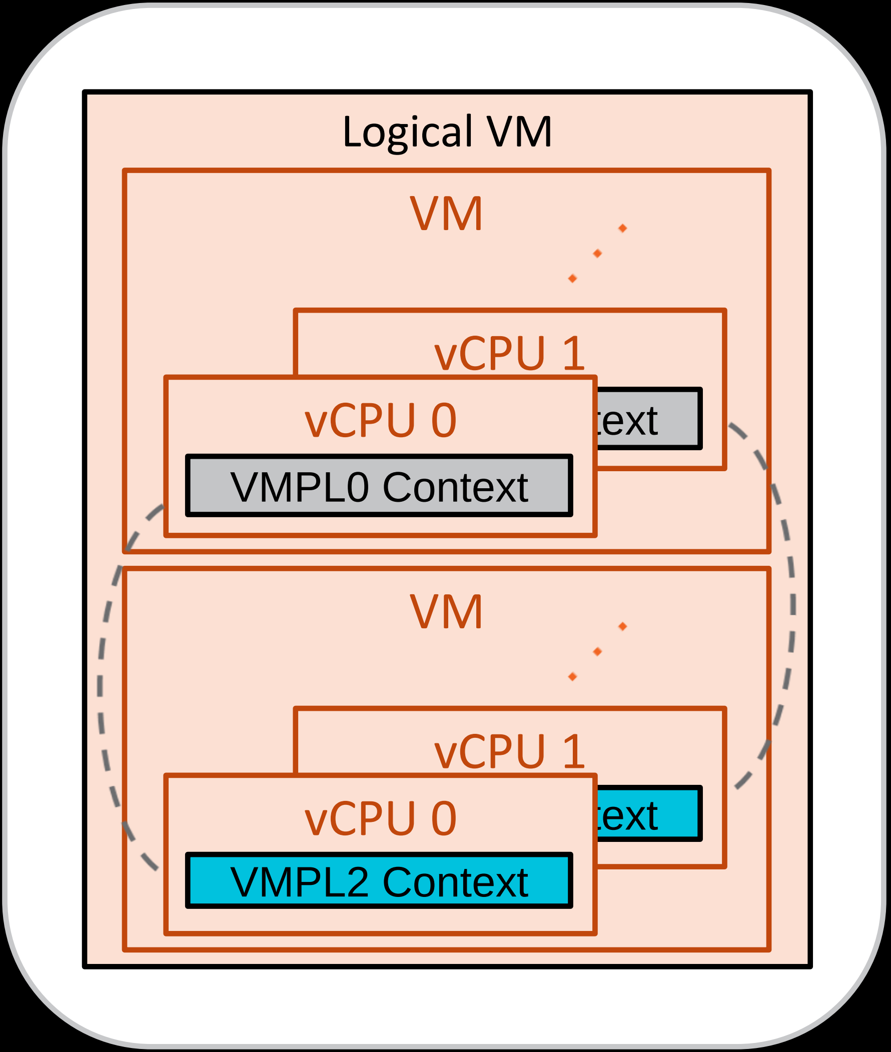
- One VM per VMPL
  - Instantiation – AP Create
    - Request sent to VMM
      - VMM creates new VM (KVM Object)
      - VMM creates new vCPUs within new VM
        - New vCPU validates/translates/sets VMSA GPA
  - Execution – Run VMPL
    - Request sent to VMM
      - VMM pauses the current VM (VMPL) vCPU
      - VMM resumes the target VM (VMPL) vCPU



AMD
together we advance_

VMPL Instantiation & Execution…

- One VM per VMPL
  - Pros
    - Separate VM results in a separate vCPU object
      - Maintains separate VMSA, GHCB, APIC, etc. contexts
  - Cons
    - Need to transition to userspace (VMM)
    - Multiple vCPU threads
      - vCPU pinning becomes more difficult
    - Qemu requires VM-to-device association
      - Affects interrupt routing
    - Gmem memory can’t be shared across VMs (currently) [1]



[1] – https://lore.kernel.org/lkml/cover.1691446946.git.ackerleytng@google.com/

Summary

- Which approach is best?
  - RFC of the first approach submitted to get the conversation started
    - TODO: Multi-VMPL Interrupt Routing/APIC Support
      - Only two VMPL levels
      - No injection into VMPL0

  - Transitions to userspace will be expensive
    - VMPL switch needs to be fast
      - Memory acceptance, Alternate Injection, vTPM
    - Rules out the approaches that require exit to the VMM, unless:
      - Can it be avoided with KVM changes/optimizations?
      - Create some association between vCPUs with same APIC ID
        - Kick vCPU across VMs/KVM Objects?

REFERENCES

- Links to the following reference material can be found at https://developer.amd.com/sev
  - White Papers & Specifications
    - AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More
    - Guest Hypervisor Communication Block Specification
    - Secure VM Service Module Specification
    - AMD64 Architecture Programmer’s Manual Volume 2: System Programming

- Code/Patches (https://github.com/coconut-svsm)
  - COCONUT SVSM: https://github.com/coconut-svsm/svsm
  - Hypervisor Patches:
    - Linux Kernel: https://github.com/coconut-svsm/linux
    - Qemu: https://github.com/coconut-svsm/qemu
  - Guest Patches: Upstream in EDKII/OVMF and Linux

DISCLAIMER

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED 'AS IS." AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

AMD