SEV-SNP: DEVELOPMENT STATUS UPDATE

Ashish Kalra & Michael Roth

KVM FORUM – 2022

SEV-SNP SECURITY FEATURES

- Introduced with “Zen 3”
- Previously had:
  - SEV, “Secure Encrypted Virtualization”
    - Guest data confidentiality via encrypted guest memory
  - SEV-ES, “Secure Encrypted Virtualization – Encrypted State”
    - Additional guest data confidentiality via encrypted vCPU register state
- SEV-SNP, builds on SEV/SEV-ES to also provide:
  - Guest data integrity
    - Secure Nested Paging
  - Control-flow security (optional)
    - CPUID Security
    - Interrupt Security
    - Secure TSC
- More details later

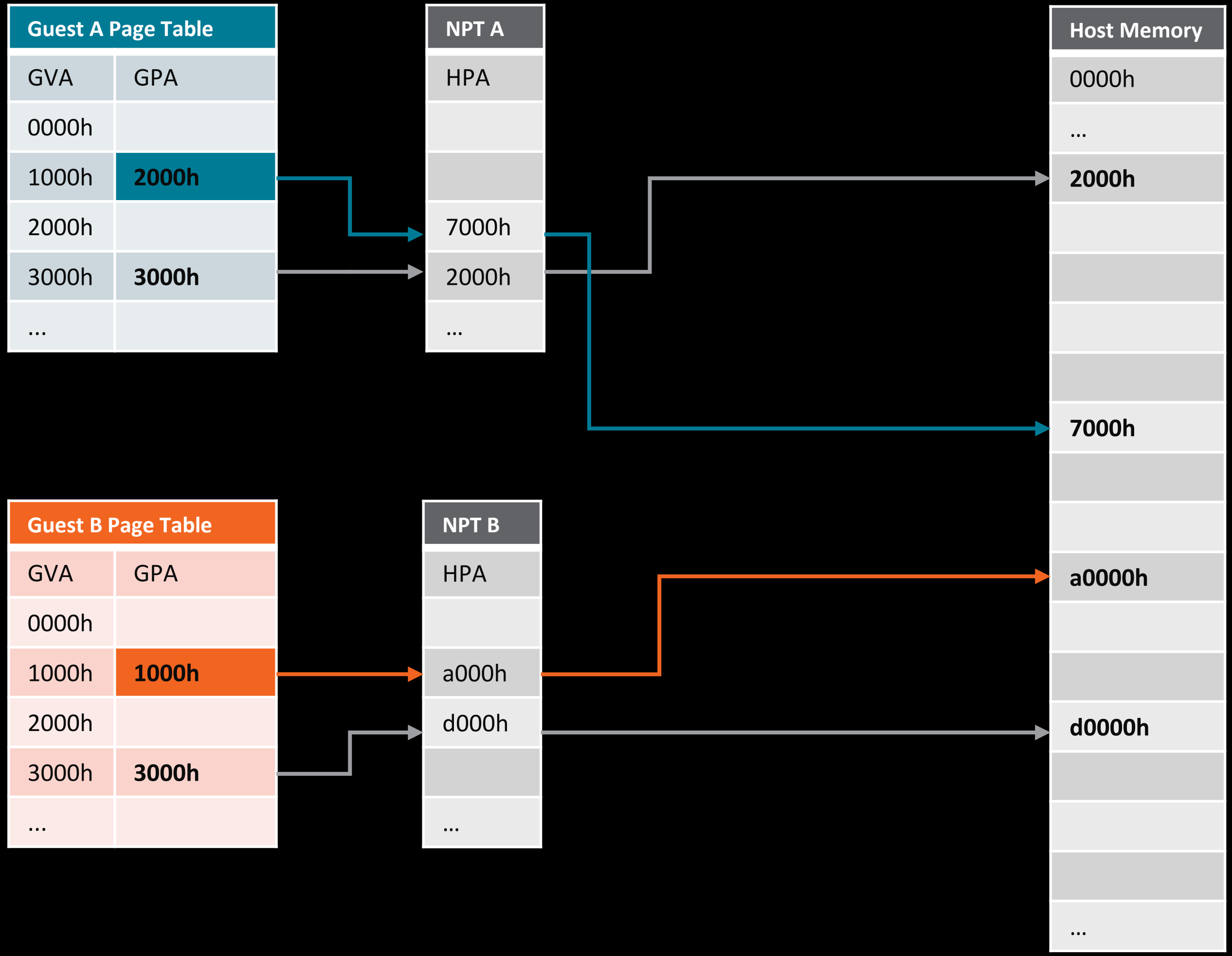
SEV-SNP SECURITY FEATURES: UPSTREAM STATUS

- Guest kernel support upstream
- Guest OVMF support upstream
- Hypervisor support posted (v6)

FEATURE	GUEST SUPPORT	HYPERVISOR SUPPORT
Secure Nested Paging	kernel v5.19 UEFI: edk2-stable202202	v6 posted
CPUID Security (optional)	kernel: v5.19 UEFI: edk2-stable202202	🟩
Interrupt Security (optional)	Future	Future
Secure TSC (optional)	Future	Future

NESTED PAGING

- Guest page table maps GVA -> GPA
- Nested page table maps GPA -> HPA
- For SEV/SEV-ES: C-bit in guest page table determines whether access is encrypted (bit 47 or 51)
- Guest controls C-bit, only ciphertext is stored in memory: provides data confidentiality
- Host controls NPT/memory: things like remap/replay attacks or silently corrupting guest memory still possible



Guest A Page Table	
GVA	GPA
0000h	
1000h	**2000h**
2000h	
3000h	**3000h**
…	

NPT A
HPA
7000h
2000h
…

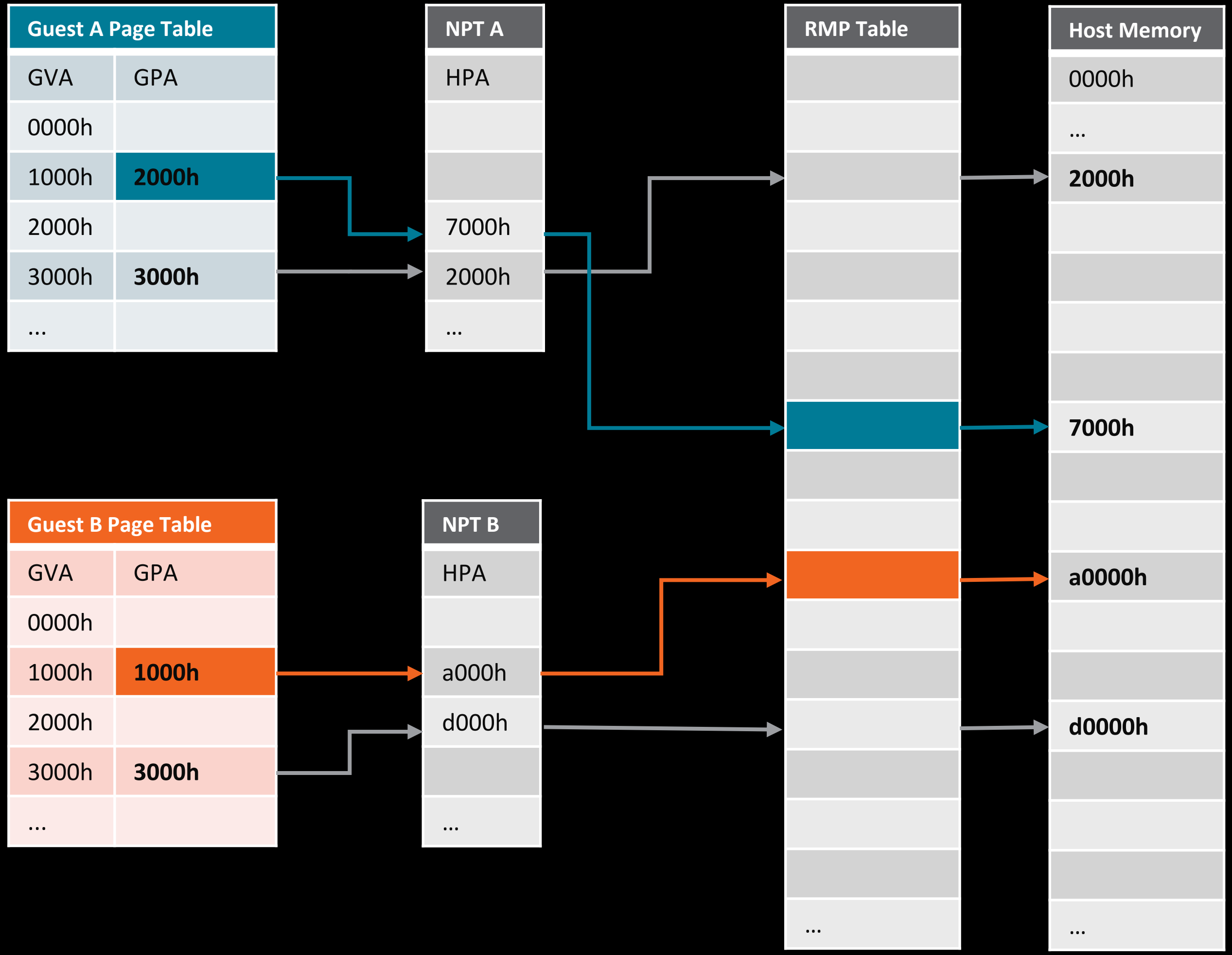
Guest B Page Table	
GVA	GPA
0000h	
1000h	**1000h**
2000h	
3000h	**3000h**
…	

NPT B
HPA
a000h
d000h
…

Host Memory
0000h
…
2000h
7000h
a0000h
d0000h
…

SECURE NESTED PAGING

- Guest page table maps GVA -> GPA
- Nested page table maps GPA -> HPA
- Reverse-map table maps HPA -> GPA
- Also provides additional integrity checks on memory accesses by host/guests
- Provides additional* protection against things like remap/replay/corruption attacks
- How?



Guest A Page Table

GVA	GPA
0000h	
1000h	**2000h**
2000h	
3000h	**3000h**
…	

NPT A

HPA
7000h
2000h
…

Guest B Page Table

GVA	GPA
0000h	
1000h	**1000h**
2000h	
3000h	**3000h**
…	

NPT B

HPA
a000h
d000h
…

RMP Table

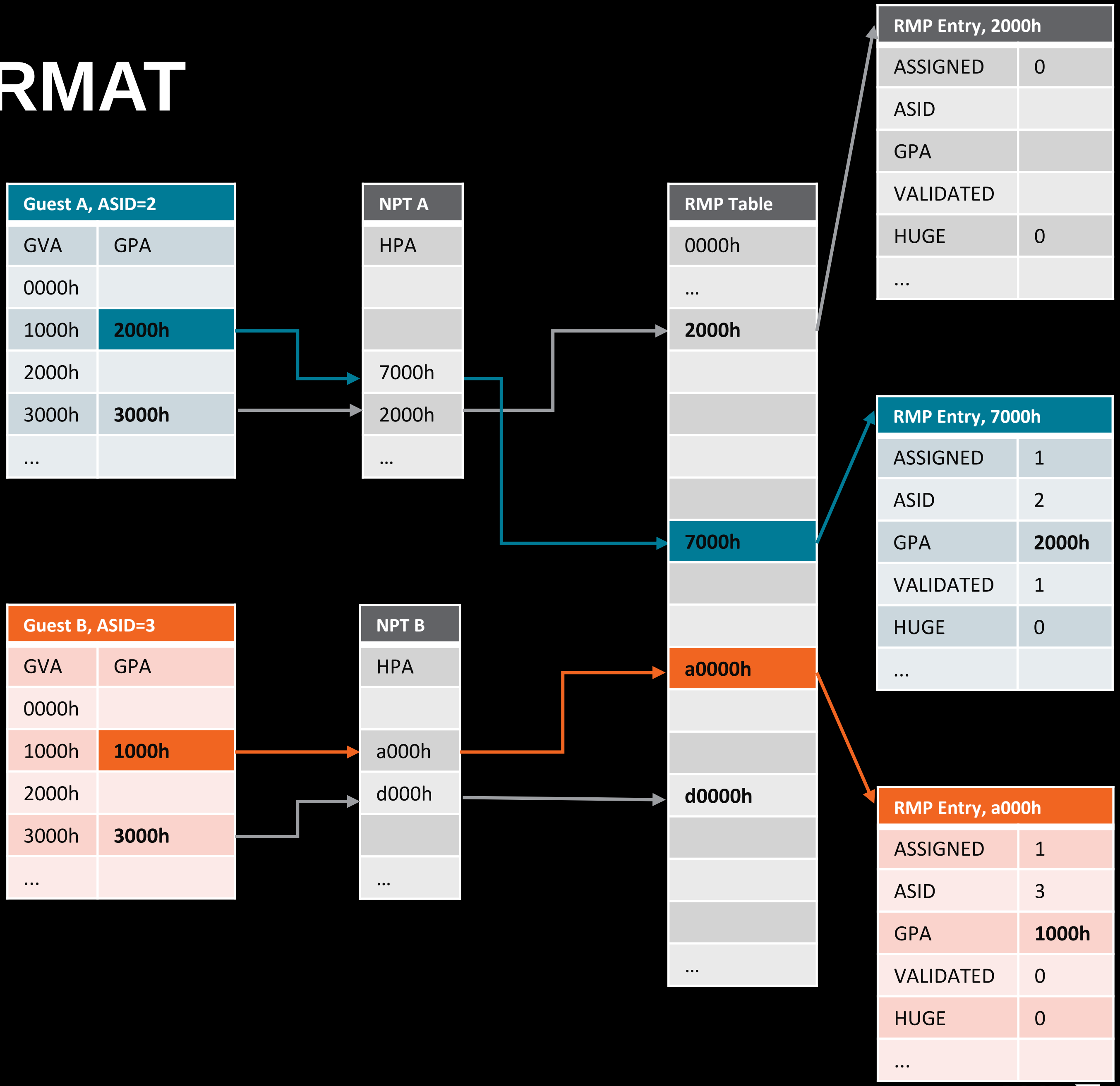
…

Host Memory

0000h
…
2000h
7000h
a0000h
d0000h
…

REVERSE-MAP TABLE FORMAT

- Assigned:
  - 0 -> host-owned, shared
  - 1 -> guest-owned, private (encrypted*)
- ASID: what guest owns it
- GPA: what guest GPA it backs
- Validated: whether guest has PVALIDATED/accepted it yet
  - C-bit=1, but not validated? -> #VC
- Host can modify/set most fields via RMPUPDATE instruction (necessary for guest page-state changes), but only guest can set the validated bit (via PVALIDATE). Important for integrity.



Guest A, ASID=2	
GVA	GPA
0000h	
1000h	2000h
2000h	
3000h	3000h
…	

NPT A
HPA
7000h
2000h
…

Guest B, ASID=3	
GVA	GPA
0000h	
1000h	1000h
2000h	
3000h	3000h
…	

NPT B
HPA
a000h
d000h
…

RMP Table
0000h
…
2000h
7000h
a0000h
d0000h
…

RMP Entry, 2000h	
ASSIGNED	0
ASID	
GPA	
VALIDATED	
HUGE	0
…	

RMP Entry, 7000h	
ASSIGNED	1
ASID	2
GPA	2000h
VALIDATED	1
HUGE	0
…	

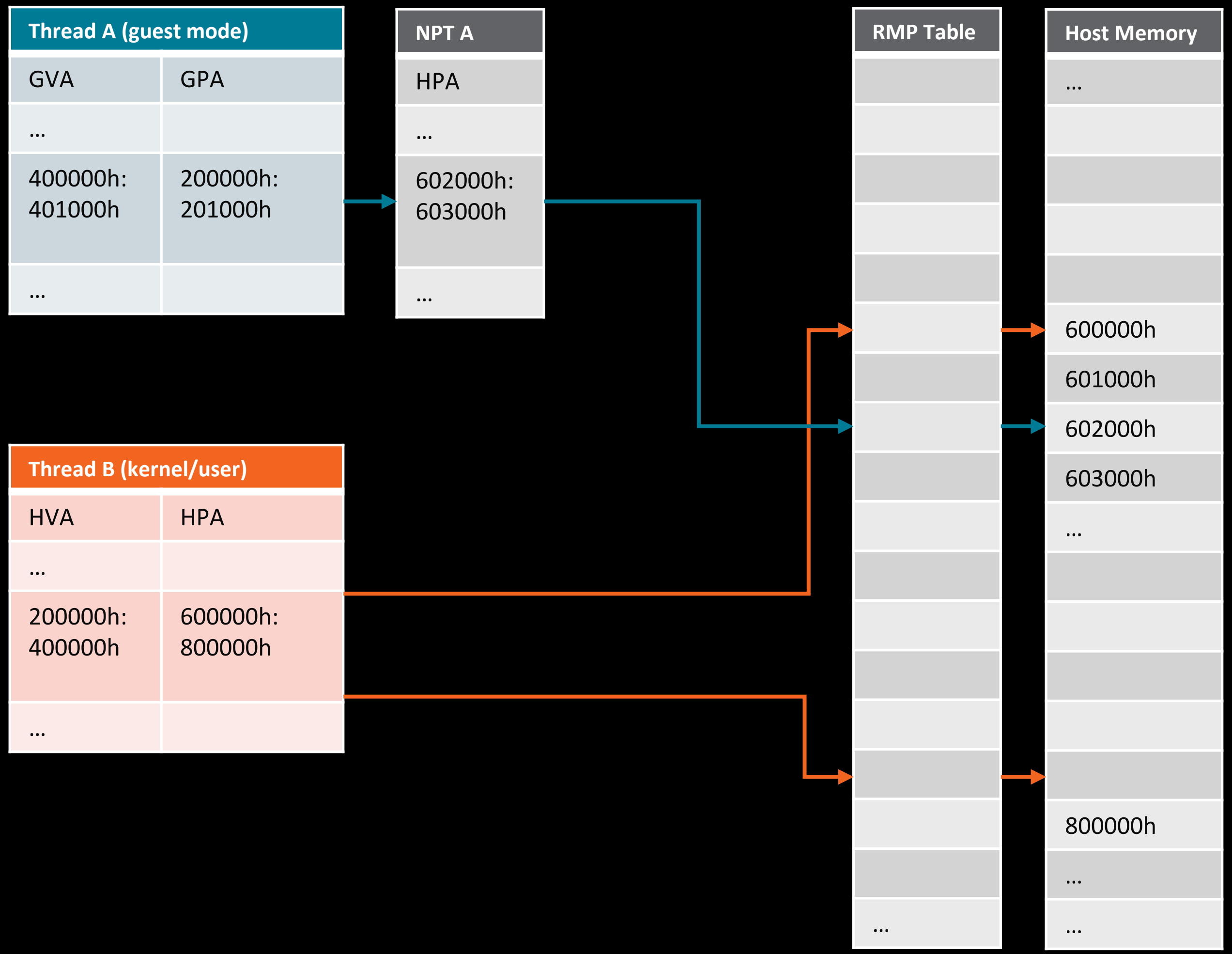
RMP Entry, a000h	
ASSIGNED	1
ASID	3
GPA	1000h
VALIDATED	0
HUGE	0
…	

KVM SUPPORT: SECURE NESTED PAGING

- Host setup/initialization of RMP table
- Guest instance setup/initialization
  - pinning pages (KVM_MEMORY_ENCRYPT_REG_REGION)
  - update RMP entries for guest boot (KVM ioctl) / runtime (GHCB request)
- RMP Fault-handling
  - Host #PF (kernel/userspace)
  - Guest #NPF →

RMP FAULT-HANDLING (HOST, #PF)

- RMP check violations result in #PF for host threads (error bit 31)
- Page overlap checks
  - 2M host mapping cannot overlap with a private page when writing →
  - True for kernel/userspace mappings
  - Kernel direct mappings need handling too (2M and 4K)
- R/W permission checks
  - Cannot write to private pages
  - userspace write -> SIGBUS to kill process
  - kernel write -> crash (buggy/malicious kernel)
  - reads allowed (ciphertext)



Thread A (guest mode)	
GVA	GPA
…	
400000h: 401000h	200000h: 201000h
…	

NPT A
HPA
…
602000h: 603000h
…

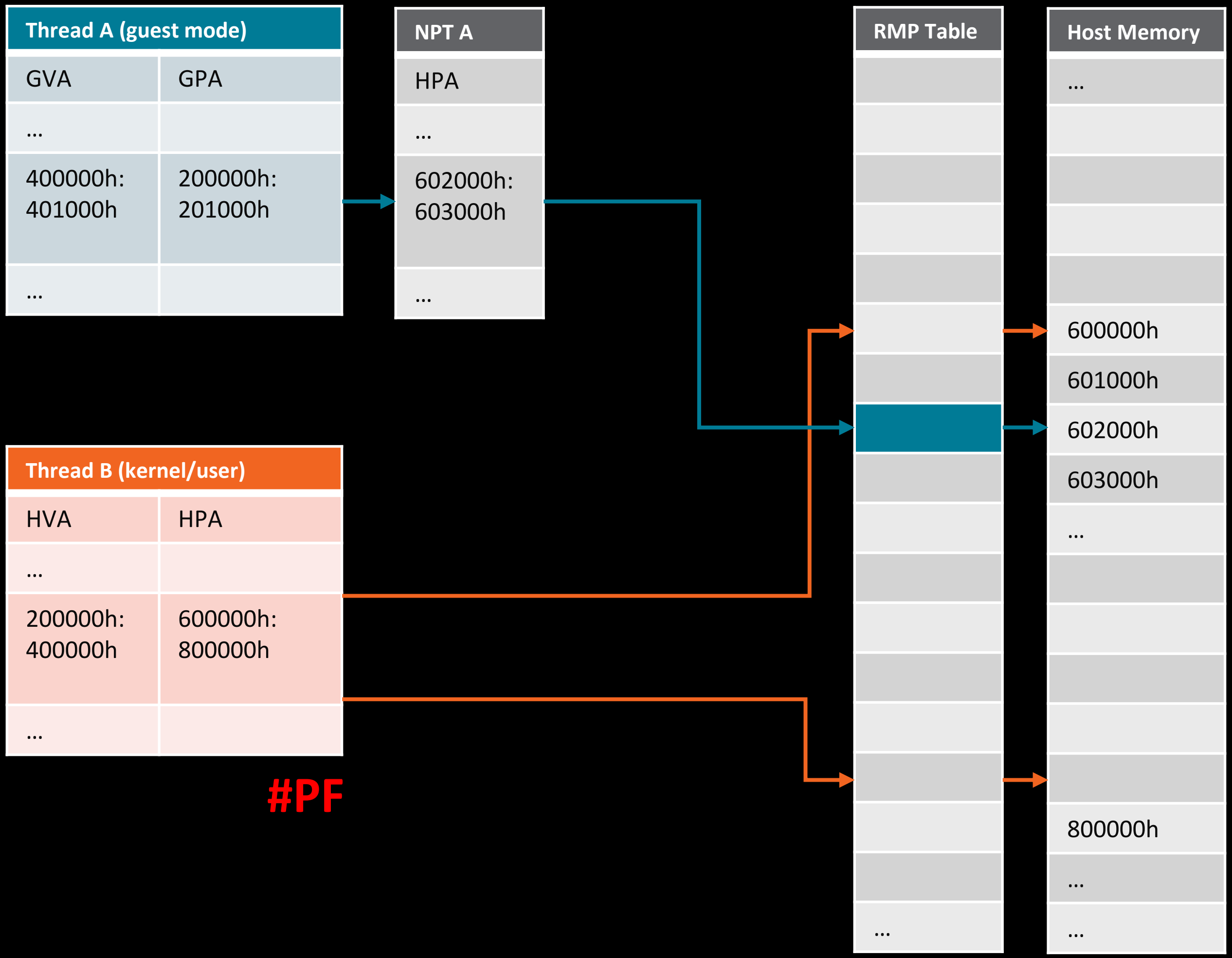
Thread B (kernel/user)	
HVA	HPA
…	
200000h: 400000h	600000h: 800000h
…	

RMP Table
…

Host Memory
…
600000h
601000h
602000h
603000h
…
800000h
…
…

RMP FAULT-HANDLING (HOST, #PF)

- RMP check violations result in #PF for host threads (error bit 31)
- Page overlap checks
  - 2M host mapping cannot overlap with a private page when writing
  - True for kernel/userspace mappings
  - Kernel direct mappings need handling too (2M)
- R/W permission checks
  - Cannot write to private pages →
  - userspace write -> SIGBUS to kill process
  - kernel write -> crash (buggy/malicious kernel)
  - reads allowed (ciphertext)



Thread A (guest mode)	
GVA	GPA
…	
400000h: 401000h	200000h: 201000h
…	

NPT A
HPA
…
602000h: 603000h
…

Thread B (kernel/user)	
HVA	HPA
…	
200000h: 400000h	600000h: 800000h
…	

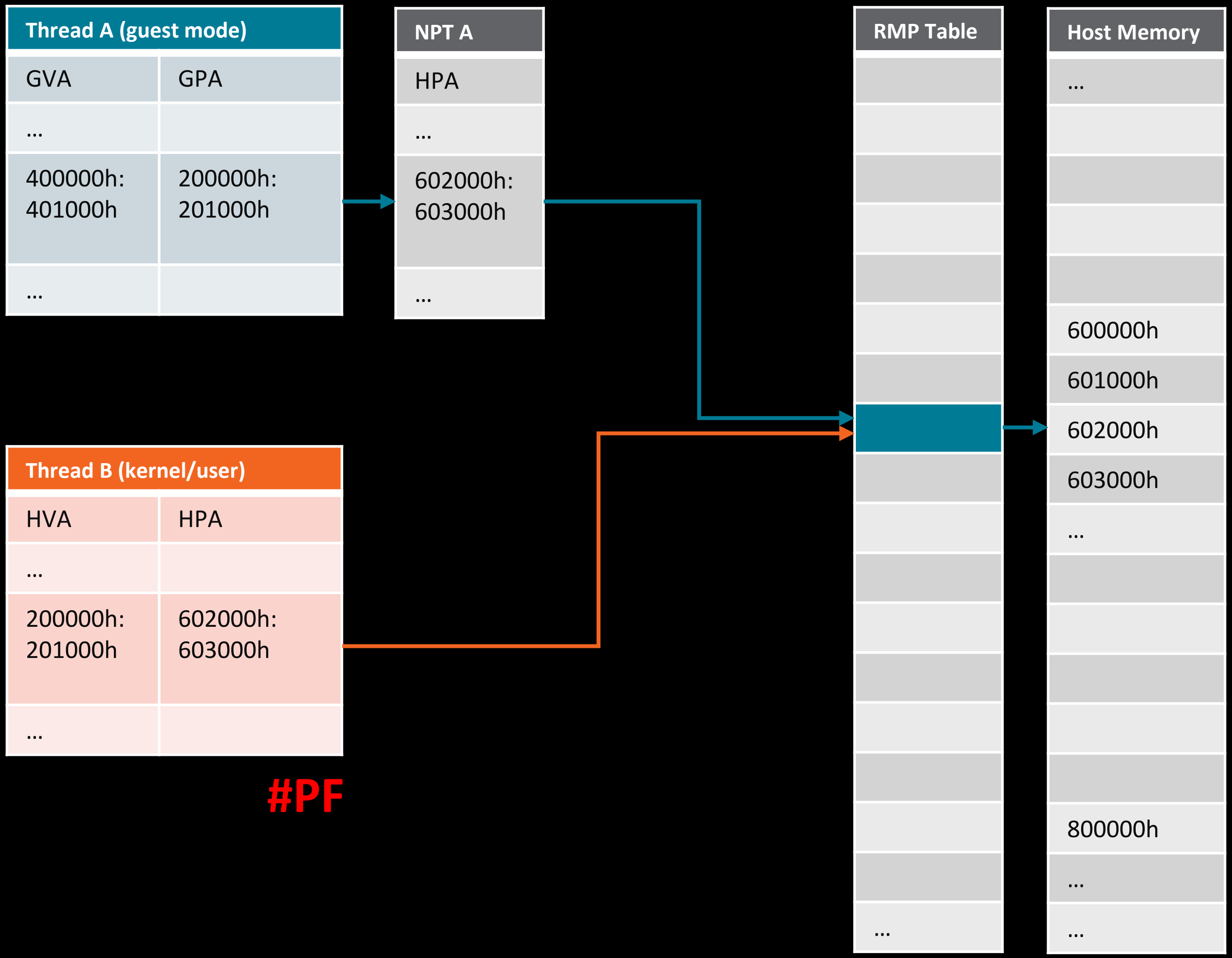
#PF

RMP Table
…

Host Memory
…
600000h
601000h
602000h
603000h
…
800000h
…
…

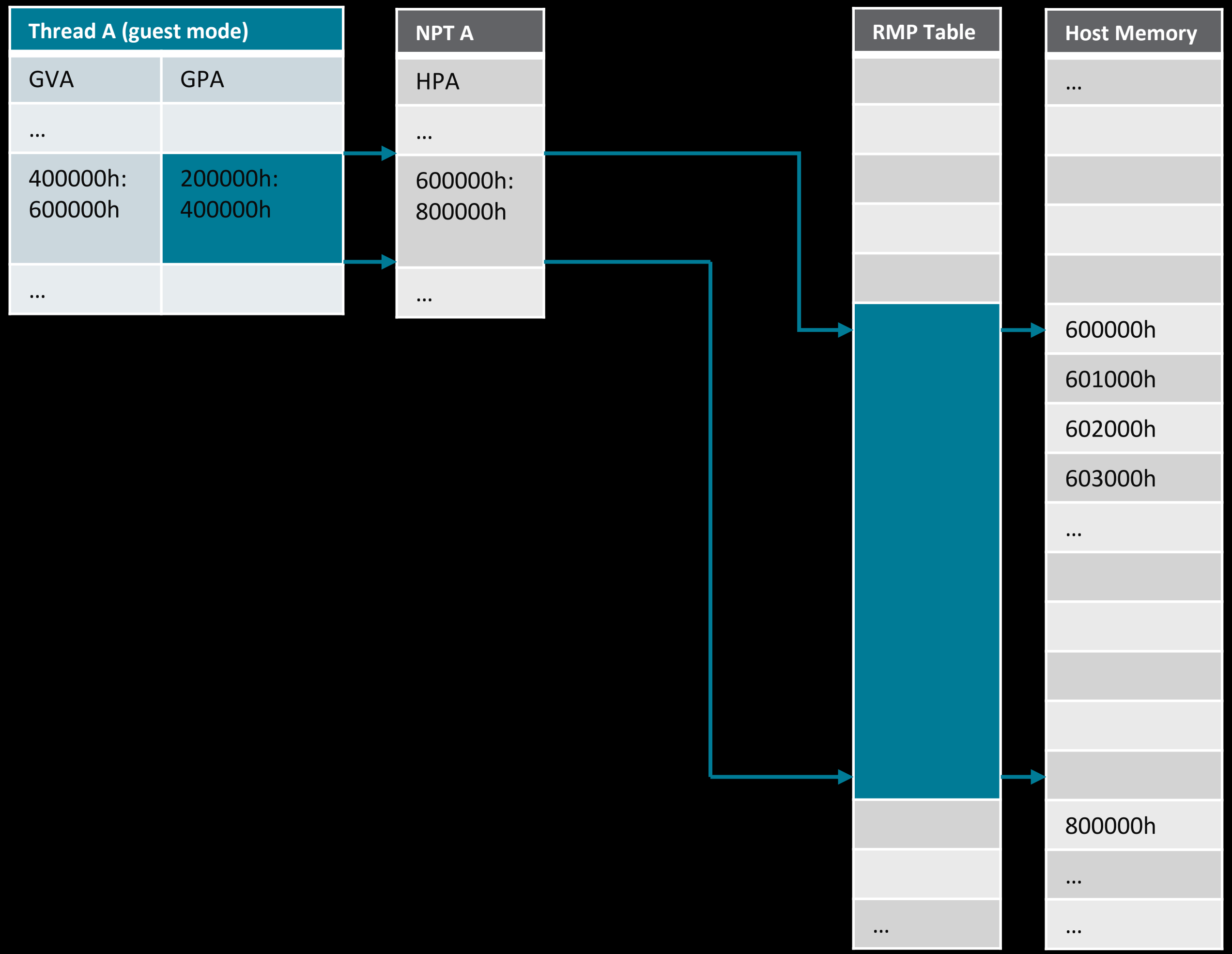
RMP FAULT-HANDLING (HOST, #PF)

- RMP check violations result in #PF for host threads (error bit 31)
- Page overlap checks
  - 2M host mapping cannot overlap with a private page when writing
  - True for kernel/userspace mappings
  - Kernel direct mappings need handling too (2M)
- R/W permission checks
  - Cannot write to private pages
  - userspace write -> SIGBUS to kill process
  - kernel write -> crash (buggy/malicious kernel)
  - reads allowed (ciphertext)



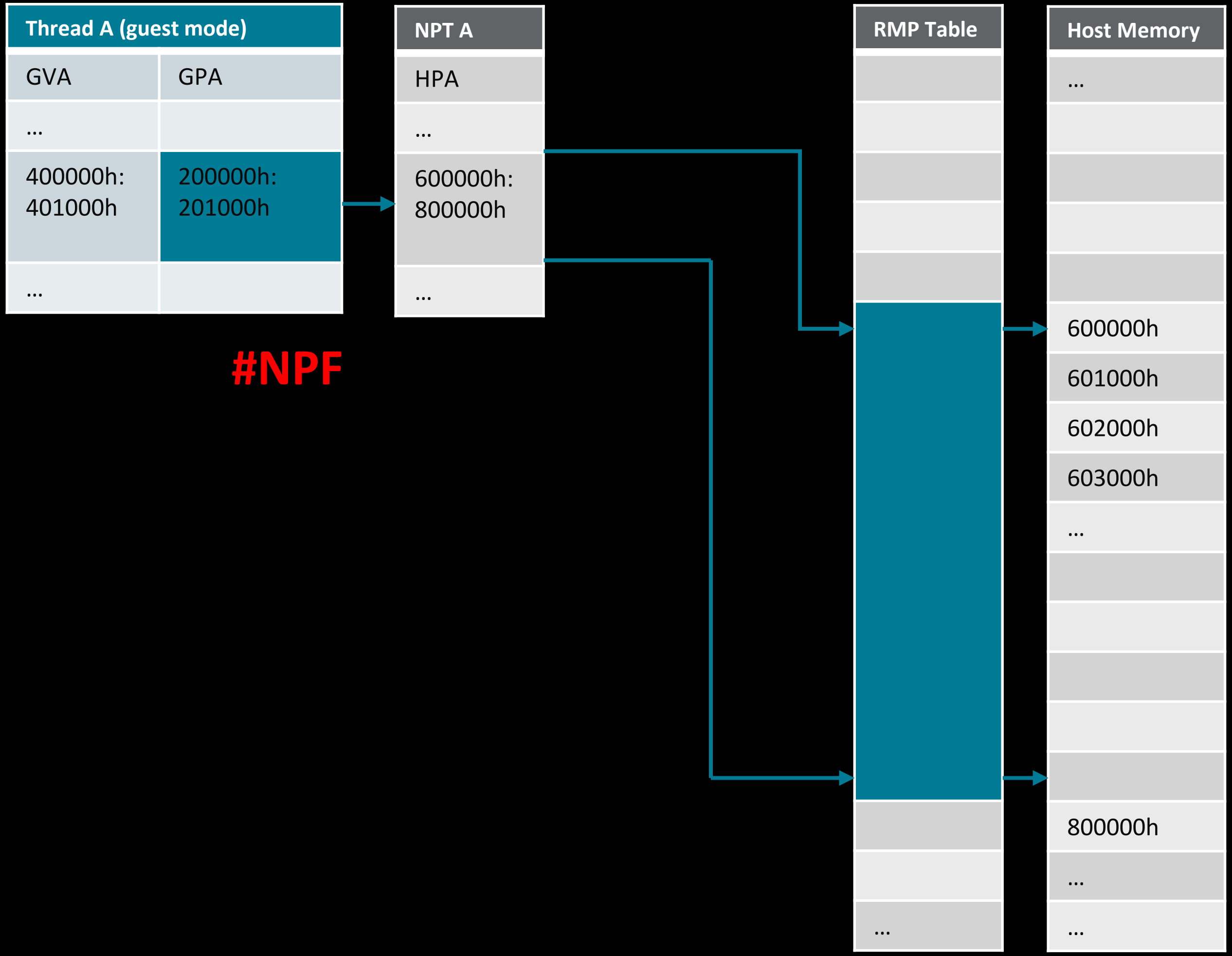
RMP FAULT-HANDLING (KVM MMU, #NPF)

- RMP check violations result in #NPF for guest vcpu threads (error bit 31)
- Page size mismatch checks
  - KVM may optimistically map a 2M private range using huge page/RMP entry
  - Guest can optimistically PVALIDATE 2M ranges to match this →
  - If 4K pvalidate: #NPF with RMP/SIZEM bits set
    - split NPT mapping
    - PSMASH 2M RMP entry
- C-bit mismatch checks
  - if C=1: RMP entry should be private
  - if C=0: RMP entry should be shared
  - Otherwise: #NPF with RMP/ENC bits set
    - Implicit page state change, update RMP entry to match



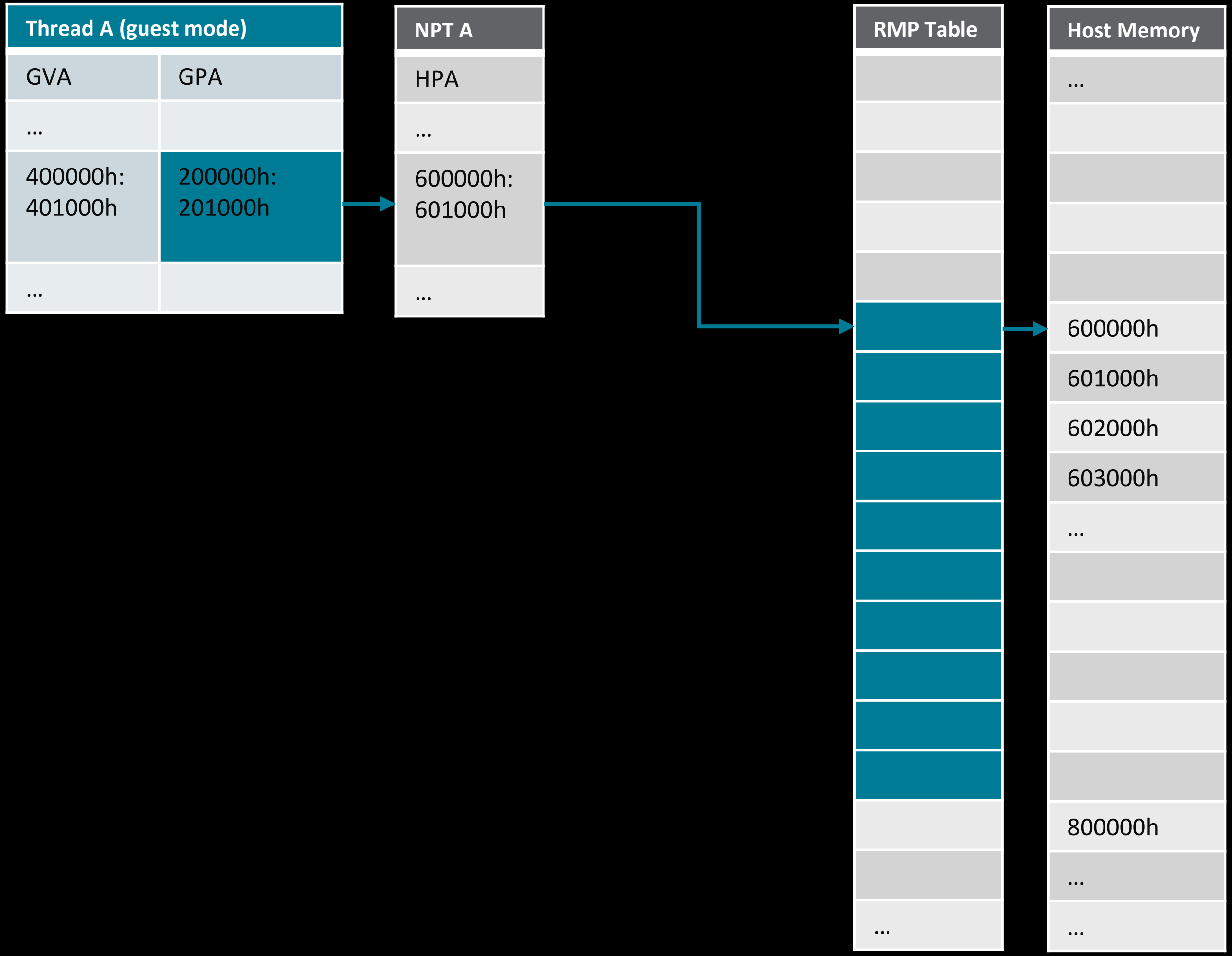
RMP FAULT-HANDLING (KVM MMU, #NPF)

- RMP check violations result in #NPF for guest vcpu threads (error bit 31)
- Page size mismatch checks
  - KVM may optimistically map a 2M private range using huge page/RMP entry
  - Guest can optimistically PVALIDATE 2M ranges to match this
  - If 4K PVALIDATE: #NPF with RMP/SIZEM bits set
    - split NPT mapping
    - PSMASH 2M RMP entry→
- C-bit mismatch checks
  - if C=1: RMP entry should be private
  - if C=0: RMP entry should be shared
  - Otherwise: #NPF with RMP/ENC bits set
    - Implicit page state change, update RMP entry to match



RMP FAULT-HANDLING (KVM MMU, #NPF)

- RMP check violations result in #NPF for guest vcpu threads (error bit 31)
- Page size mismatch checks
  - KVM may optimistically map a 2M private range using huge page/RMP entry
  - Guest can optimistically PVALIDATE 2M ranges to match this
  - If 4K PVALIDATE: #NPF with RMP/SIZEM bits set
    - split NPT mapping
    - PSMASH 2M RMP entry
- C-bit mismatch checks
  - if C=1: RMP entry should be private
  - if C=0: RMP entry should be shared
  - Otherwise: #NPF with RMP/ENC bits set
    - Implicit page state change, update RMP entry to match



KVM SUPPORT: SECURE NESTED PAGING

- Host setup/initialization of RMP table
- Guest instance setup/initialization
  - pinning pages (KVM_MEMORY_ENCRYPT_REG_REGION)
  - update RMP entries for guest boot (KVM ioctl) / runtime (GHCB request)
- RMP Fault-handling
  - Host #PF (kernel/userspace)
  - Guest #NPF
- Fairly minor changes since v5, however:
- New proposal: UPM (Unmapped Private Memory), private FD-backed memory

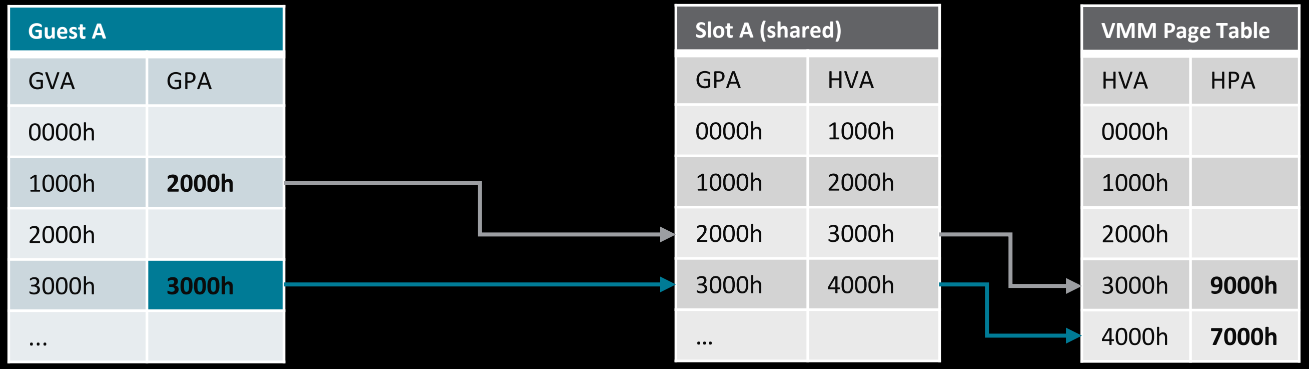
UNMAPPED PRIVATE MEMORY

- Proposed kernel infrastructure to back confidential guests with pages that are not mappable/accessible by userspace
- Generally synonymous with Chao Peng’s private memslot patchset:
  - “KVM: mm: fd-based approach for supporting KVM guest private memory”
- Proposed by a number of a developers for various reasons, but the most prevalent driver is TDX support, where writes to private guest memory by userspace result in #MC
- Also being evaluated for use with SEV-SNP, pKVM, and possibly others

UPM - PRIVATE MEMSLOTS

- Currently both shared/private memory are backed by normal memslots
  - private memory can be mapped into userspace just like normal memory
  - malloc() / mmap() →
- Adds new private memslot struct
  - Provides both shared/private memory
  - private memory allocated separately via memfd
  - memfd uses MFD_INACCESSIBLE
    - Not readable/writable
    - Can’t be mmap()’d into userspace
- KVM MMU uses an xarray to determine whether to map guest memory from shared/private pool

#NPF: GPA->HPA lookup (normal memslot)



Guest A

GVA	GPA
0000h	
1000h	**2000h**
2000h	
3000h	**3000h**
…	

Slot A (shared)

GPA	HVA
0000h	1000h
1000h	2000h
2000h	3000h
3000h	4000h
…	

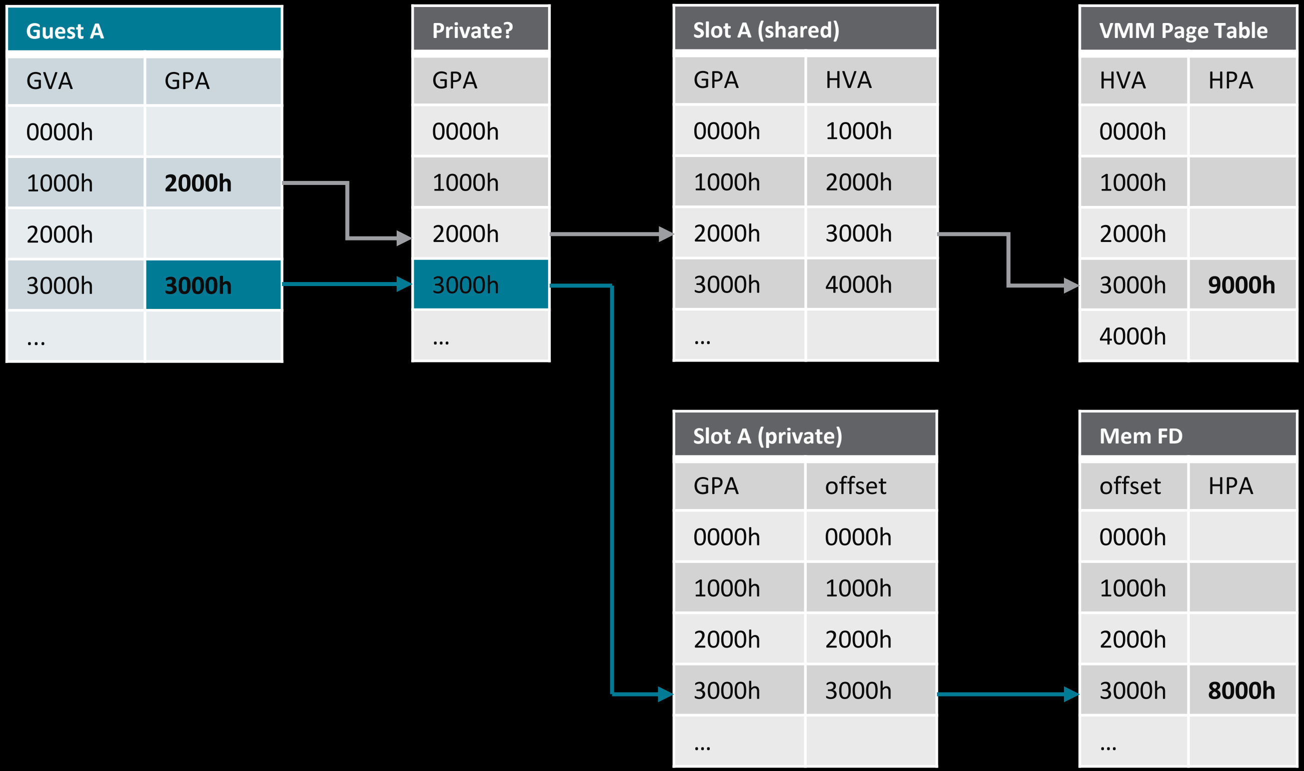
VMM Page Table

HVA	HPA
0000h	
1000h	
2000h	
3000h	**9000h**
4000h	**7000h**

UPM – PRIVATE MEMSLOTS

- Currently both shared/private memory are backed by normal memslots
  - private memory can be mapped into userspace just like normal memory
  - malloc() / mmap()
- Adds new private memslot struct
  - Provides both shared/private memory
  - private memory allocated separately via memfd
  - memfd uses MFD_INACCESSIBLE
    - Not readable/writable
    - Can't be mmap()'d into userspace
- KVM MMU uses an xarray to determine whether to map guest memory from shared/private pool

#NPF: GPA->HPA lookup (private memslot)



Guest A

GVA	GPA
0000h	
1000h	**2000h**
2000h	
3000h	**3000h**
…	

Private?

GPA
0000h
1000h
2000h
3000h
…

Slot A (shared)

GPA	HVA
0000h	1000h
1000h	2000h
2000h	3000h
3000h	4000h
…	

VMM Page Table

HVA	HPA
0000h	
1000h	
2000h	
3000h	**9000h**
4000h	

Slot A (private)

GPA	offset
0000h	0000h
1000h	1000h
2000h	2000h
3000h	3000h
…	

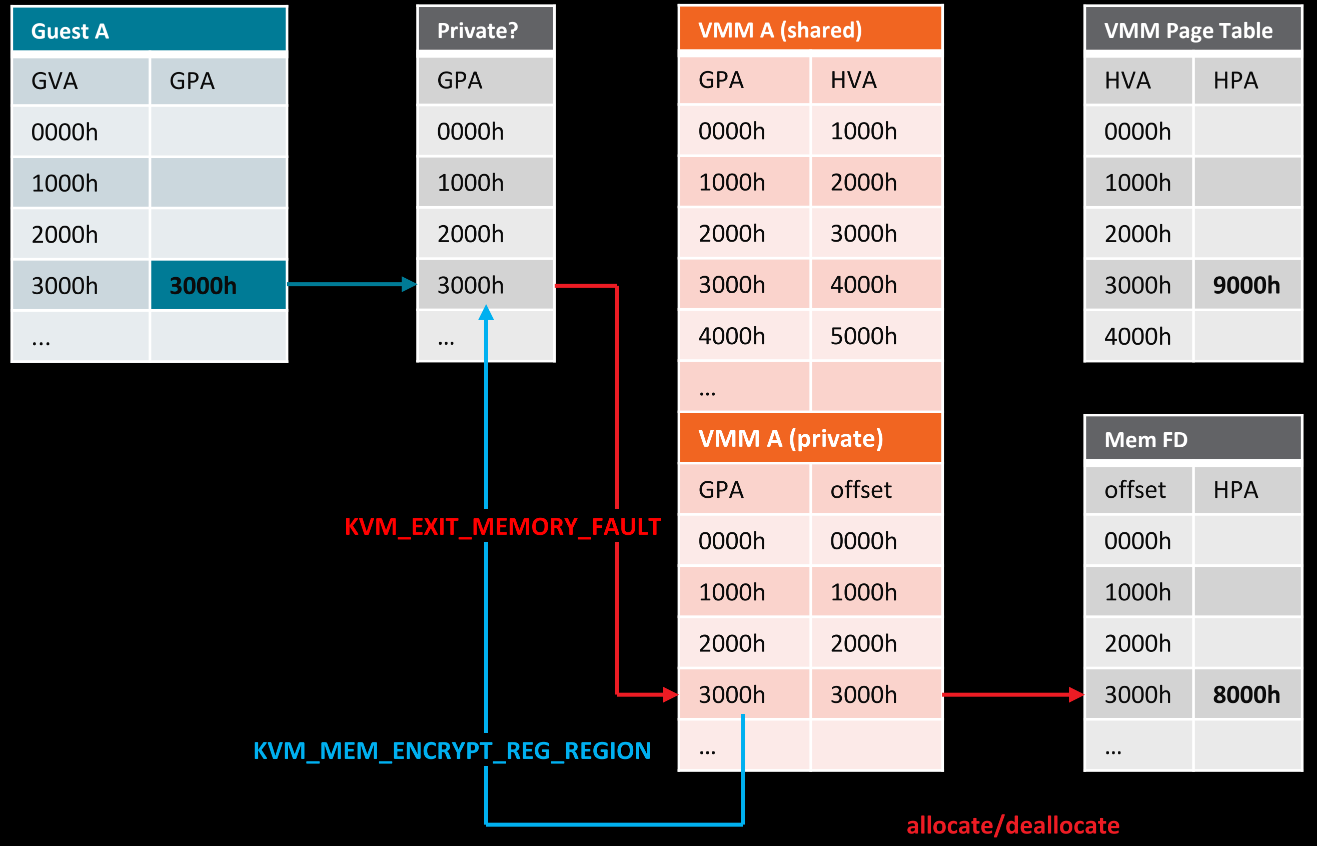
Mem FD

offset	HPA
0000h	
1000h	
2000h	
3000h	**8000h**
…	

UPM – IMPLICIT CONVERSIONS

- KVM MMU uses an xarray to determine whether to map guest memory from shared/private pool
  - xarray controlled purely by userspace
    - KVM_MEM_ENCRYPT_REG_REGION
    - KVM_MEM_ENCRYPT_UNREG_REGION
- Implicit conversion
  - if C-bit does not match xarray state:
    - KVM_EXIT_MEMORY_FAULT
    - alloc/dealloc private/shared memory
    - VMM converts using REG/UNREG ioctl
- Explicit conversion
  - GHCB page-state change request forwarded to userspace
    - KVM_EXIT_VMGEXIT
    - alloc/dealloc private/shared memory
    - VMM converts using REG/UNREG ioctl

#NPF: GPA->HPA lookup/conversion (private memslot)



Guest A

GVA	GPA
0000h	
1000h	
2000h	
3000h	**3000h**
…	

Private?

GPA
0000h
1000h
2000h
3000h
…

VMM A (shared)

GPA	HVA
0000h	1000h
1000h	2000h
2000h	3000h
3000h	4000h
4000h	5000h
…	

VMM A (private)

GPA	offset
0000h	0000h
1000h	1000h
2000h	2000h
3000h	3000h
…	

VMM Page Table

HVA	HPA
0000h	
1000h	
2000h	
3000h	**9000h**
4000h	

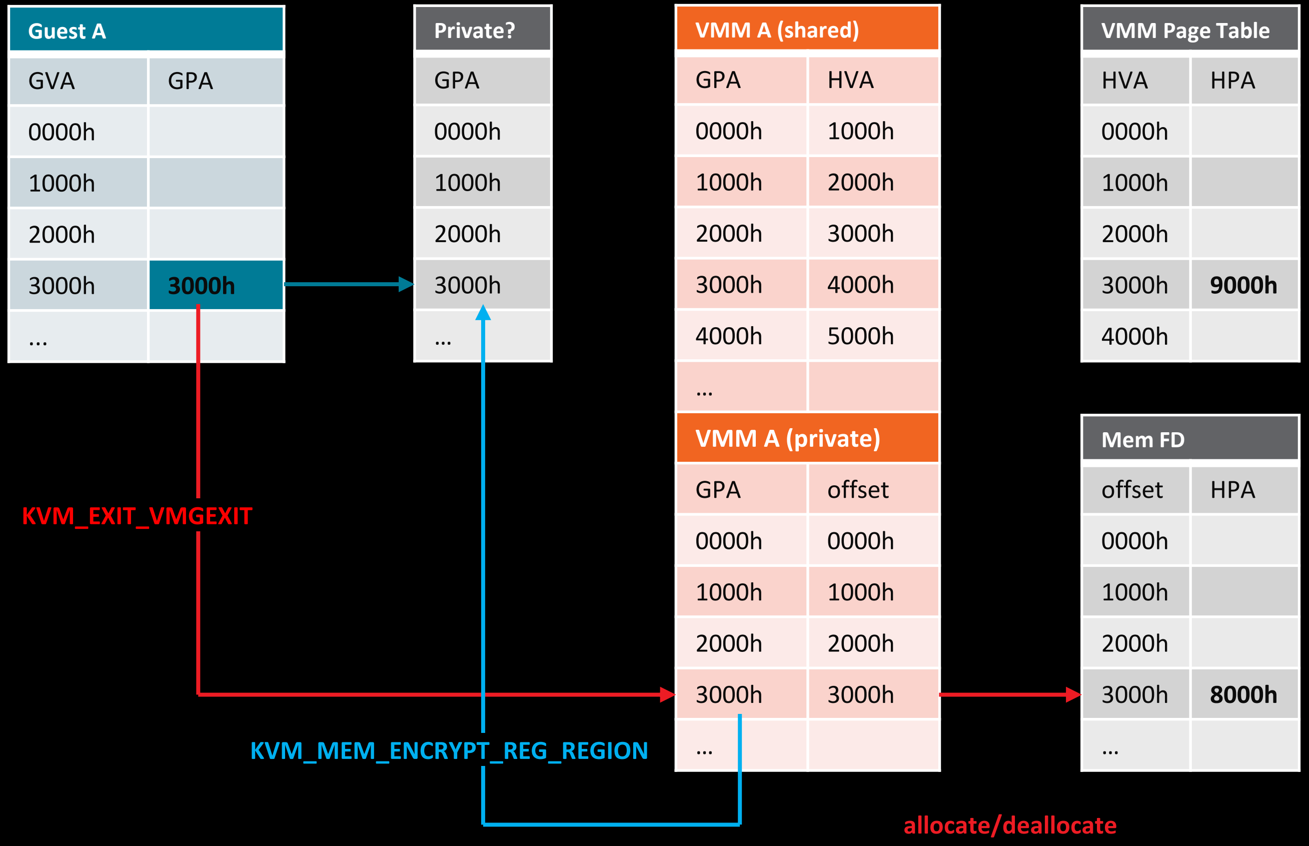
Mem FD

offset	HPA
0000h	
1000h	
2000h	
3000h	**8000h**
…	

UPM – EXPLICIT CONVERSIONS

- KVM MMU uses an xarray to determine whether to map guest memory from shared/private pool
  - xarray controlled purely by userspace
    - KVM_MEM_ENCRYPT_REG_REGION
    - KVM_MEM_ENCRYPT_UNREG_REGION
- Implicit conversion
  - if C-bit does not match xarray state:
    - KVM_EXIT_MEMORY_FAULT
    - alloc/dealloc private/shared memory
    - VMM converts using REG/UNREG ioctl
- Explicit conversion
  - GHCB page-state change request forwarded to userspace
    - KVM_EXIT_VMGEXIT
    - alloc/dealloc private/shared memory
    - VMM converts using REG/UNREG ioctl

#NPF: GPA->HPA lookup/conversion (private memslot)



Guest A

GVA	GPA
0000h	
1000h	
2000h	
3000h	3000h
…	

Private?

GPA
0000h
1000h
2000h
3000h
…

VMM A (shared)

GPA	HVA
0000h	1000h
1000h	2000h
2000h	3000h
3000h	4000h
4000h	5000h
…	

VMM A (private)

GPA	offset
0000h	0000h
1000h	1000h
2000h	2000h
3000h	3000h
…	

VMM Page Table

HVA	HPA
0000h	
1000h	
2000h	
3000h	9000h
4000h	

Mem FD

offset	HPA
0000h	
1000h	
2000h	
3000h	8000h
…	

UPM: PROS/CONS

- Pros:
  - Shared infrastructure for managing private guest pages
    - Cross-platform: SNP / TDX, potentially cross-architecture
  - Less chance of guest disruption/exploitation from accessing private memory in userspace
  - Lazy-pinning support
- Cons:
  - More management complexity in VMMs:
    - Allocating/de-allocating private memory
      - Potential for 2X memory usage
        - Lazily-deallocate for performance?
        - Immediately deallocate to reduce memory usage?
    - Handling of new private memslot structure
    - Memory pinning/affinity considerations
  - Performance
    - More exits to userspace, more context switches

KVM SUPPORT: SNP + UPM

- v6 SNP hypervisor patchset uses non-UPM implementation (likely v7 as well)
- Separate tree adds UPM support on top of v6:
  - VMGEXITS for GHCB page-state changes forwarded to userspace
  - Uses UPM to manage memory pinning instead of existing SEV approach
  - KVM_CAP_UPM flag to switch between modes
- Will maintain separate trees until UPM stable/upstream
- Continue to work with community to upstream either solution

FUTURE DEVELOPMENT WORK

- Interrupt Security
- Secure VM Service Modules (SVSM)
  - VMPL0 OS implementation
  - Interrupt security
  - Live migration acceleration
  - vTPM
  - Upcoming talk by Tom Lendacky
- Secure TSC support
- SEV-SNP lazy-pinning support (free with UPM)
- Lazy-PVALIDATE support for SNP guests (patches posted)

FUTURE DEVELOPMENT WORK

- Interrupt Security
- Secure VM Service Modules (SVSM)
  - VMPL0 OS implementation
  - Interrupt security
  - Live migration acceleration
  - vTPM
  - Upcoming talk by Tom Lendacky
- Secure TSC support
- SEV-SNP lazy-pinning support (free with UPM)
- Lazy-PVALIDATE support for SNP guests (patches posted)

- **Questions?**

Copyright and 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 releases, for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or 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

AMD