Securing secure boot with System Management Mode

Paolo Bonzini
Red Hat, Inc.
KVM Forum 2015

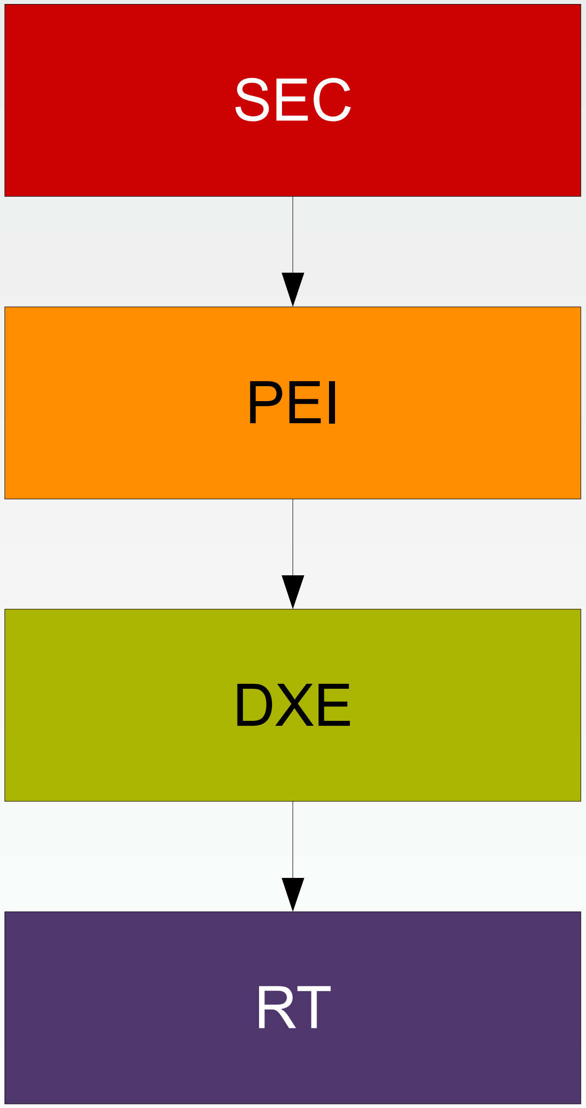
Outline

- UEFI overview
- Secure boot and SMM
- Chipset support: QEMU
- Hypervisor support: KVM

What is UEFI?

- “A replacement for BIOS that's designed to improve software interoperability” – *Microsoft*
  - Modular architecture with reusable components (e.g. SCSI and network stacks)
  - Portable drivers
- “BIOS documentation was bad, but this time we produced a 10,000 page spec” – *me*
- “We missed DOS so much that we burnt it into your ROM” – *Matthew Garrett*

UEFI phases – Boot



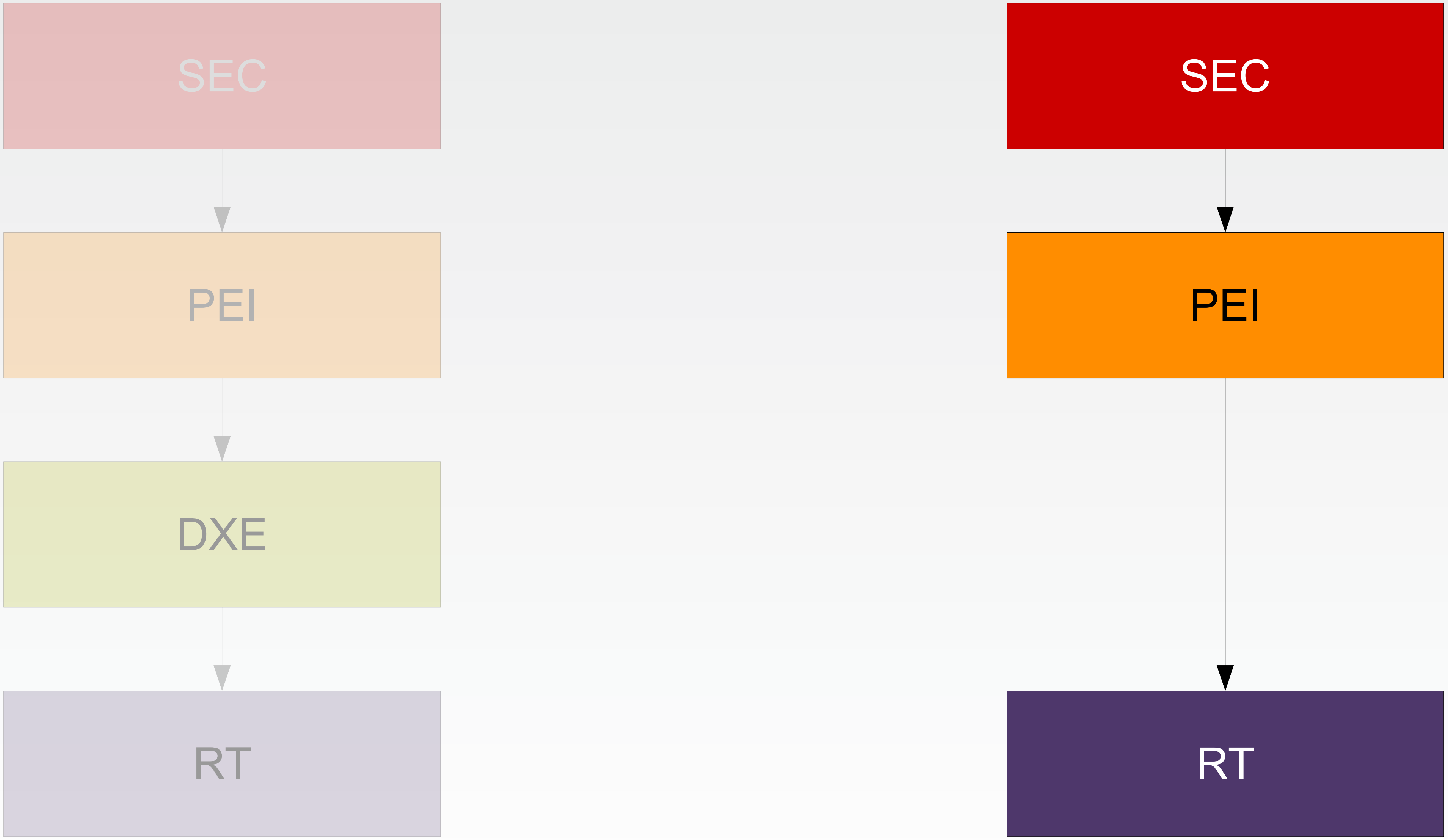
“Secure” phase
Configure memory controller, enable caches
Runs from ROM, decompressing the rest

Pre-EFI Initialization
Initialize chipset
Handle S3 resume

Driver eXecution Environment
Discover hardware
Select boot device (BDS phase)

Run-time services
Available while OS runs

UEFI phases – S3 resume



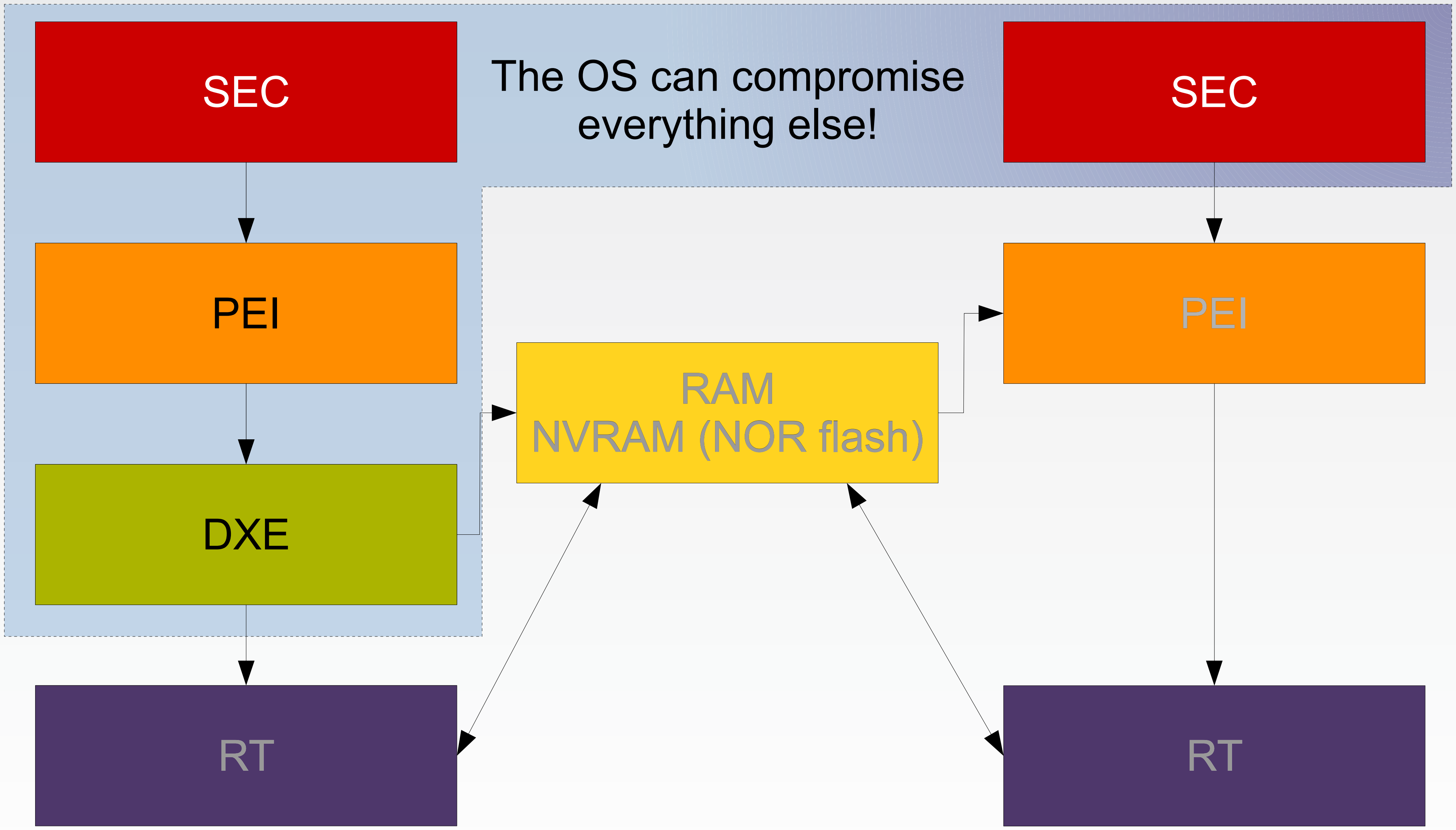
What's secure boot?

- Firmware-verified chain of trust until OS loads
- OS handles the chain of trust after boot
  - Enforce signing for all code running in the kernel (e.g. kernel modules)
  - Enforce signing for device firmware (especially if firmware is not checked by the device)
  - Non-privileged code should not be able to run arbitrary unsigned ring-0 code (e.g. via /dev/mem)
- OS should not be able to inject arbitrary code into the firmware!

How can the OS attack the firmware?

- PEI S3 resume uses data from DXE
  - CPU data (MSRs, control registers, …)
  - S3 bootscript to initialize other devices
- Run-time services must access flash to provide persistent variable storage
  - Variable storage includes the keys used to enforce secure boot!
  - Changes to “trusted” variables must be signed by a higher-level key

UEFI phases – Communication



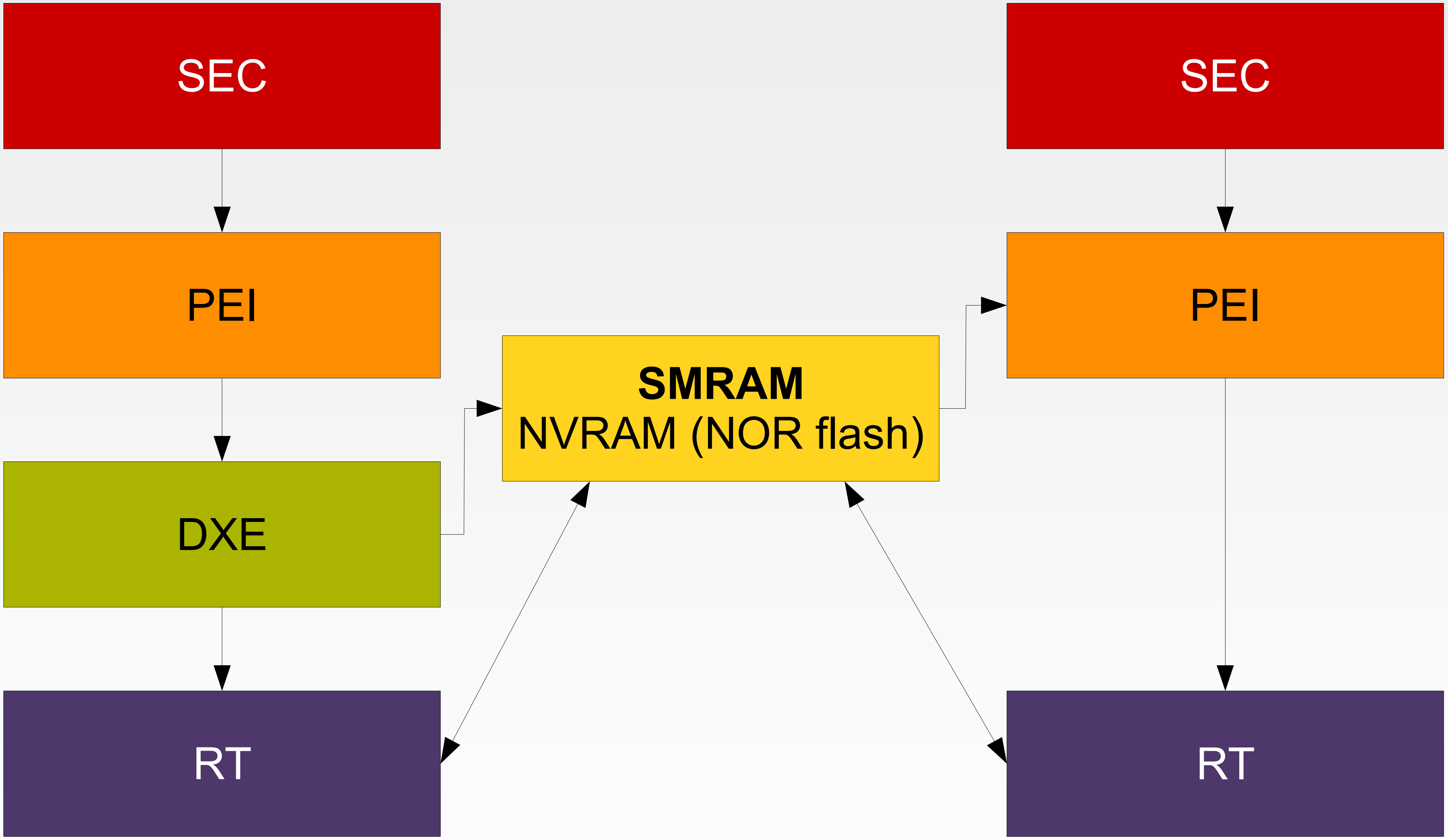
System Management Mode

- First introduced in 1990 (80386SL)
- Lets the chipset interrupt the running program with arbitrary code by asserting SMI#
  - OUT to port 0B2h usually triggers an SMI
- Processor state is stored in RAM, execution starts at a known address (SMBASE+8000h)
  - SMBASE=30000h on startup
  - SMBASE can be changed by the SMM handler
- The RSM instruction resumes normal execution

System Management Mode: SMRAM

- The chipset can keep some RAM hidden to processors not in SMM
  - Originally the 128K at A0000h were used
  - Usually shadowed by video memory if not in SMM
  - On modern chipsets, up to 8MB of memory below 4GB (“TSEG”) can be reserved for SMM
- SMRAM and TSEG configuration can be locked
  - No-tampering guarantee!

Securing secure boot!



Attacking secure boot: SMI handler

- VU#127284: accessing RAM from SMM (2009)

```
movq   0x407d(%rip),%rax   ;; TSEG
callq *0x18(%rax)          ;; RAM
```

- VU#976132: “All of the available systems we evaluated stored boot script in unprotected ACPI NVS” (2014)

  “The only system we identified that used the SMM lockbox to protect the boot script was a UEFI development motherboard […] It dispatched functions in unprotected ACPI NVS”

Attacking secure boot: insecure hardware!

- Caching attacks
  - Mark SMRAM as writeback-cached, pollute cache, generate SMI
  - Newer processors have SMRR (SMM memory range register)
  - Does not apply under virtualization
- Chipset attacks
  - On Q35 TSEG size can be locked, but TSEG base cannot!
  - Not vulnerable due to incomplete chipset emulation

Attacking secure boot: insecure hardware!

- Unprotected NOR flash
  - Flash access should only be allowed from SMM… except if firmware forgot to set that bit to 1
  - This matters for virtual machines too
- Interrupt descriptor table
  - Somehow force SMI handler to take an exception
  - Recent processors reset IDT limit to 0
  - Undocumented, but KVM has to do the same!

http://www.ssi.gouv.fr/uploads/IMG/pdf/IT_Defense_2010_final.pdf

Securing OVMF

- Assume firmware has no bugs :-)
- KVM must:
  - Perform SMM world switch (SMI, RSM)
  - Hide SMRAM to processors not in SMM
- QEMU must:
  - Implement required chipset registers
  - Protect flash from processors not in SMM
  - Support KVM extensions for SMM (and TCG)
- Target: Q35 (440FX SMRAM too small)

QEMU: What is missing?

- Q35-specific SMRAM features
  - TSEG
  - SMRAM locking
- Per-CPU visibility of SMRAM
  - Avoid races on SMP guests
- Flash protection

QEMU: What is there already?

- Basic support for SMRAM at 0xA0000
- All registers reside in PCI configuration space: migration format won't change
- TCG support for per-CPU address spaces

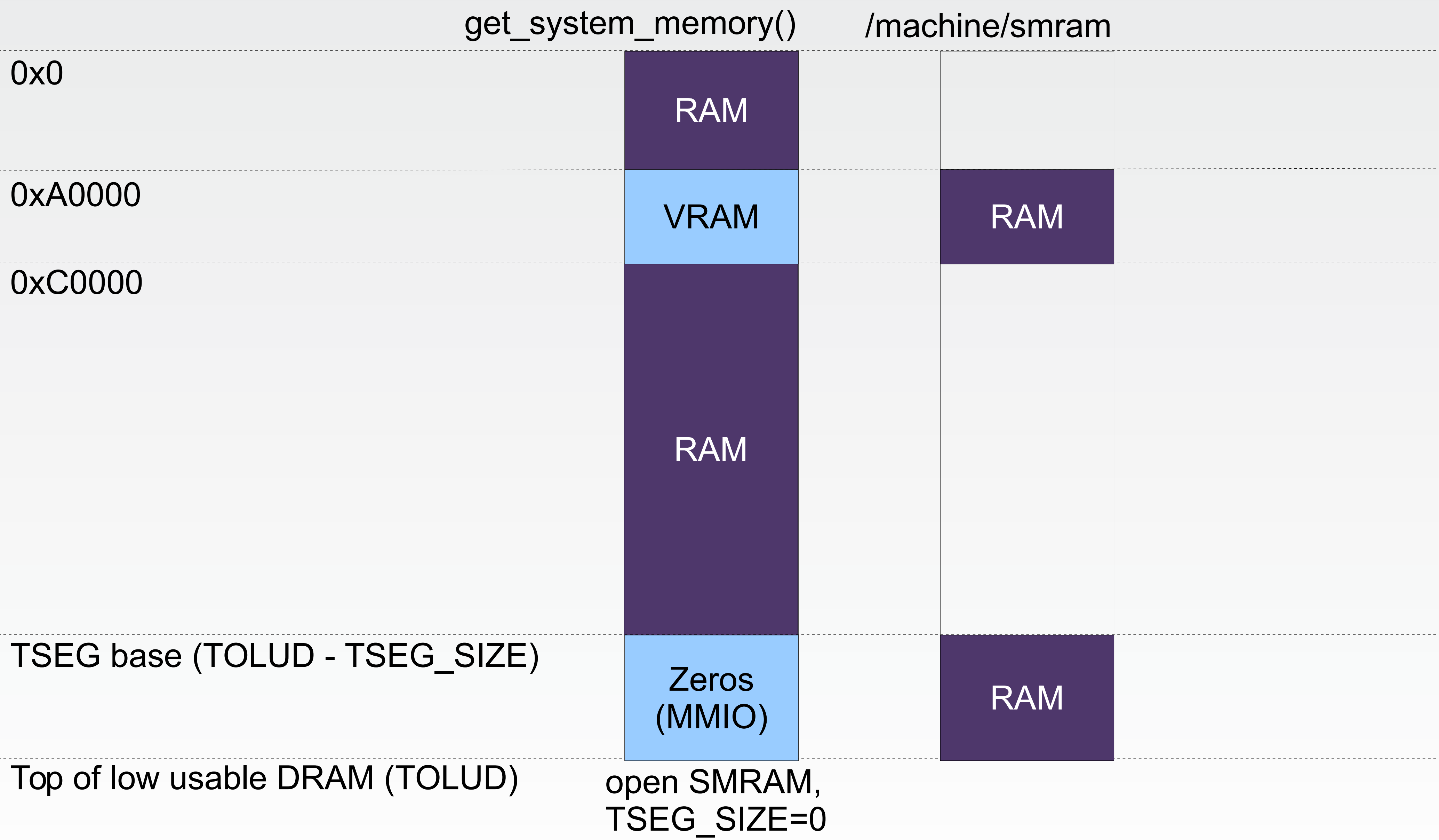
QEMU: Flash access

- Q35 uses memory-mapped NOR flash
  - Writes to flash put it in “device mode”
  - On real hardware, writes also trigger an SMI; the SMI handler puts the flash back in ROM mode
  - Complicated and prone to races
  - Newer chipsets work around the races with even more complicated protocols
- Can we do anything simpler?

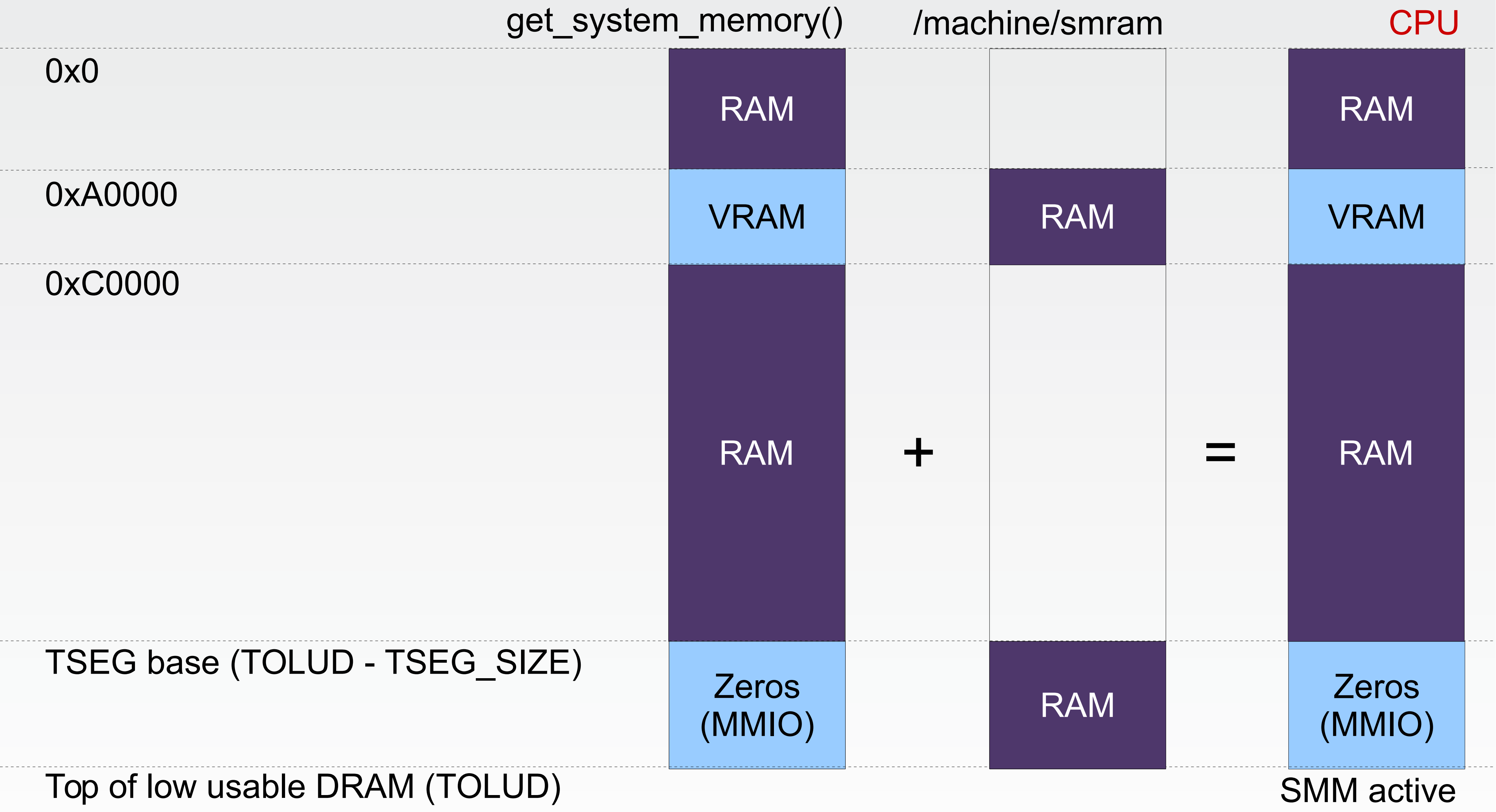
QEMU: Flash access

- Discard writes to flash outside SMM mode
  - Flash remains in ROM mode
  - OVMF does not need a special SMI handler
- Easily implemented with QEMU “memory transaction attributes”
  - TCG: `tlb_set_page_with_attrs`
  - KVM: pass “in SMM?” flag via `struct kvm_run`, read it on KVM_EXIT_MMIO/KVM_EXIT_IO
- Avoids non-standard extensions to Q35 registers

Modeling SMRAM (board)



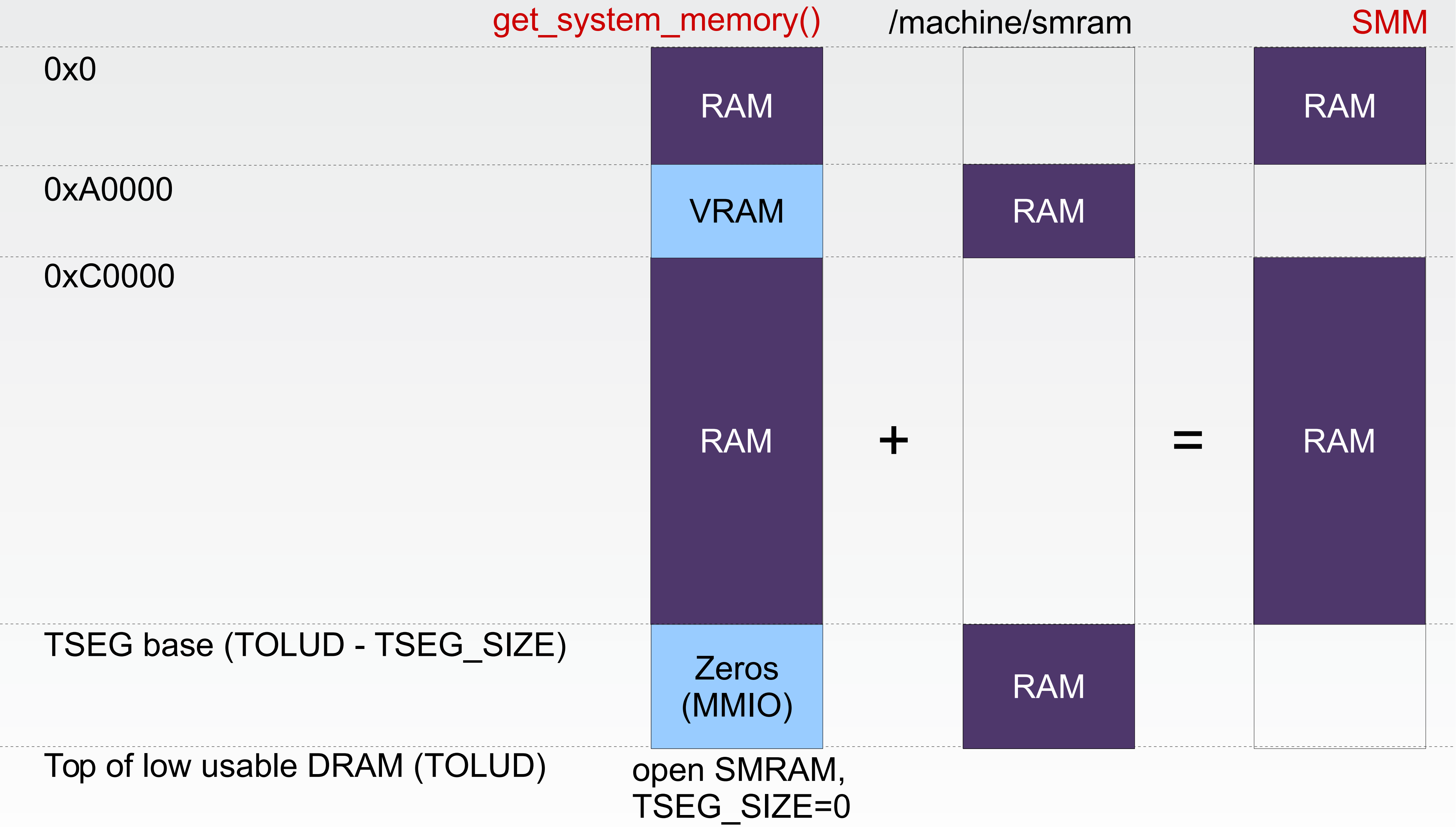
Modeling SMRAM (TCG)



KVM design

- Per-CPU address space too expensive
  - O(#vcpus) syscalls on every memory map change
  - O(#vcpus) higher cost of retrieving dirty bitmap
- Define 2 memory maps (“address spaces”) shared by all VCPUs
  - Appropriate address space for VCPU chosen according to current mode
  - Pass address space id to KVM_GET_DIRTY_LOG and KVM_SET_MEMORY_REGION
  - QEMU: one MemoryListener per address space

Modeling SMRAM (KVM)



KVM implementation: generic code

- New functions:

```
__kvm_memslots(struct kvm *, int as_id)
```

```
kvm_vcpu_gfn_to_memslot
kvm_vcpu_gfn_to_hva
kvm_vcpu_gfn_to_pfn
kvm_vcpu_gfn_to_page
kvm_vcpu_read_guest_page
kvm_vcpu_read_guest
...
```

- New architecture-specific hook:

```
kvm_vcpu_memslots(struct kvm_vcpu *vcpu);
```

KVM implementation: x86 MMU

- Use `kvm_vcpu_*` functions where appropriate
- SMM added to shadow page “role”
  - The role acts as the hash key
  - GPA→HVA mapping for arbitrary SPTEs
- Special memory regions must be added to all address spaces
  - Identity page table
  - APIC access page
  - Real-mode TSS

KVM implementation: world switch

- New ioctl: KVM_SMI
- 64-bit version different between Intel and AMD
  - QEMU uses AMD format
  - Tiano Core expects Intel format
- Intel doesn't document part of the state!
  - Segment descriptor caches
  - TR base/limit
- Therefore, KVM uses AMD format
- Nested VMX/SVM state not saved

State

- QEMU: released in 2.4
- KVM: released in Linux 4.2
- OVMF: patches under review
  - 97 files changed, 17631 insertions, 587 deletions
  - SMM core from Intel Quark SDK
    - 32-bit only
    - Uniprocessor guest only
  - TCG: qemu-system-i386
  - KVM: **-cpu** *model***,-lm,-nx**

Acknowledgements

- Laszlo Ersek (OVMF)
- Gerd Hoffmann (Q35)
- Radim Krčmář, Xiao Guangrong (KVM review)
- Michael S. Tsirkin (QEMU review)

Q&A