Confidential Computing with OpenBSD — The Next Step

(Sorry, it’s not about NeXTSTEP)

Hans-Jörg Höxer

Confidential Computing with OpenBSD

Agenda

- **Introduction**
- First step: Memory encryption for VMs — SEV
- Next step: vCPU state encryption — SEV-ES
- Conclusion

About

Hans-Jörg Höxer

- Mid-2000s:
  - hshoexer@openbsd.org
- genua GmbH (www.genua.de):
  - hshoexer@genua.de
  - OpenBSD based products
  - Firewalls and VNP-Appliances
  - Confidential Computing

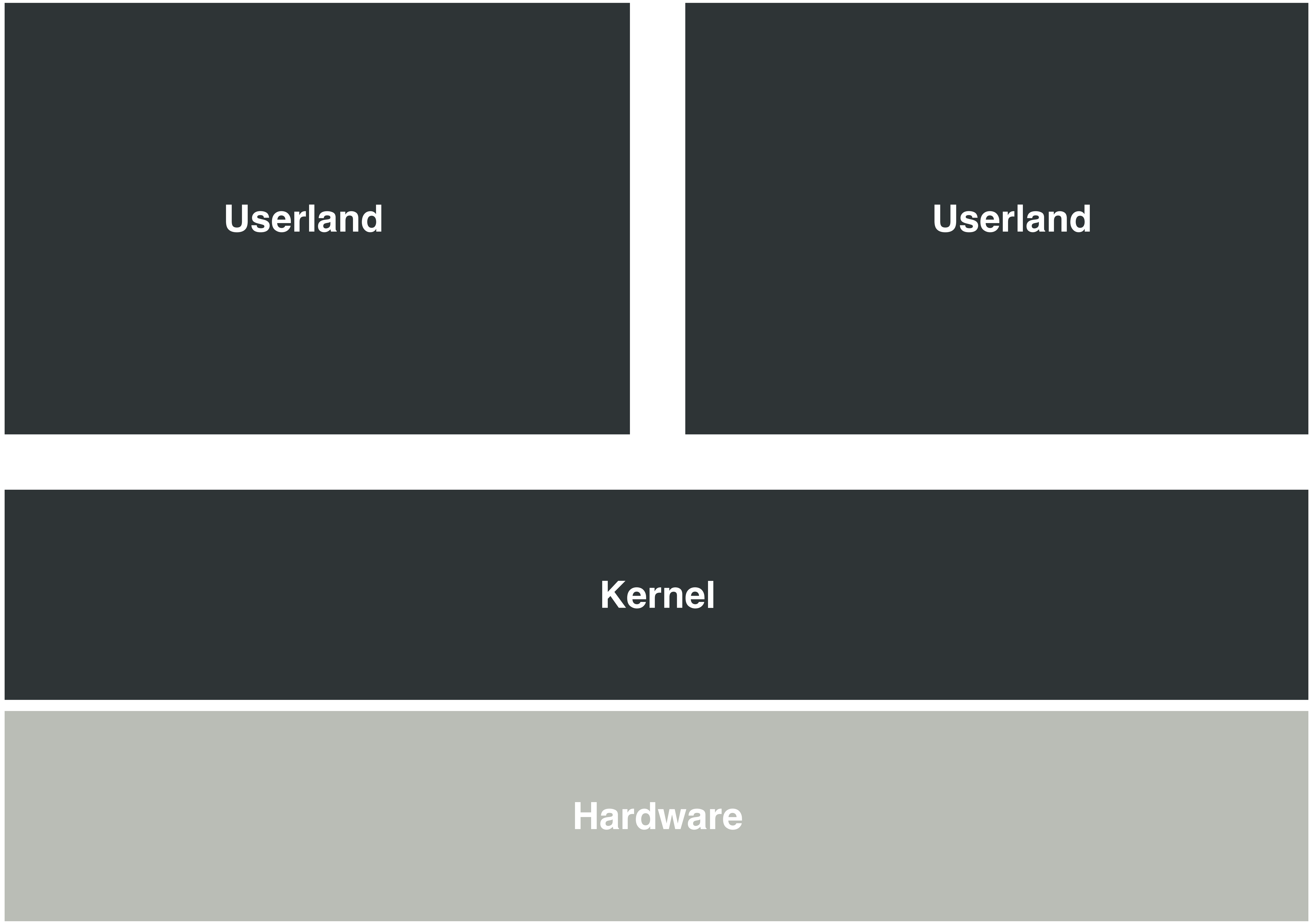
Confidential Computing

What is this all about?

- Problem:
  - Sensitive data in an untrusted environment
  - Context: Virtualisation, VMs, cloud
- Supposed solution:
  - “Turn public cloud into private cloud”
  - Bold claims…

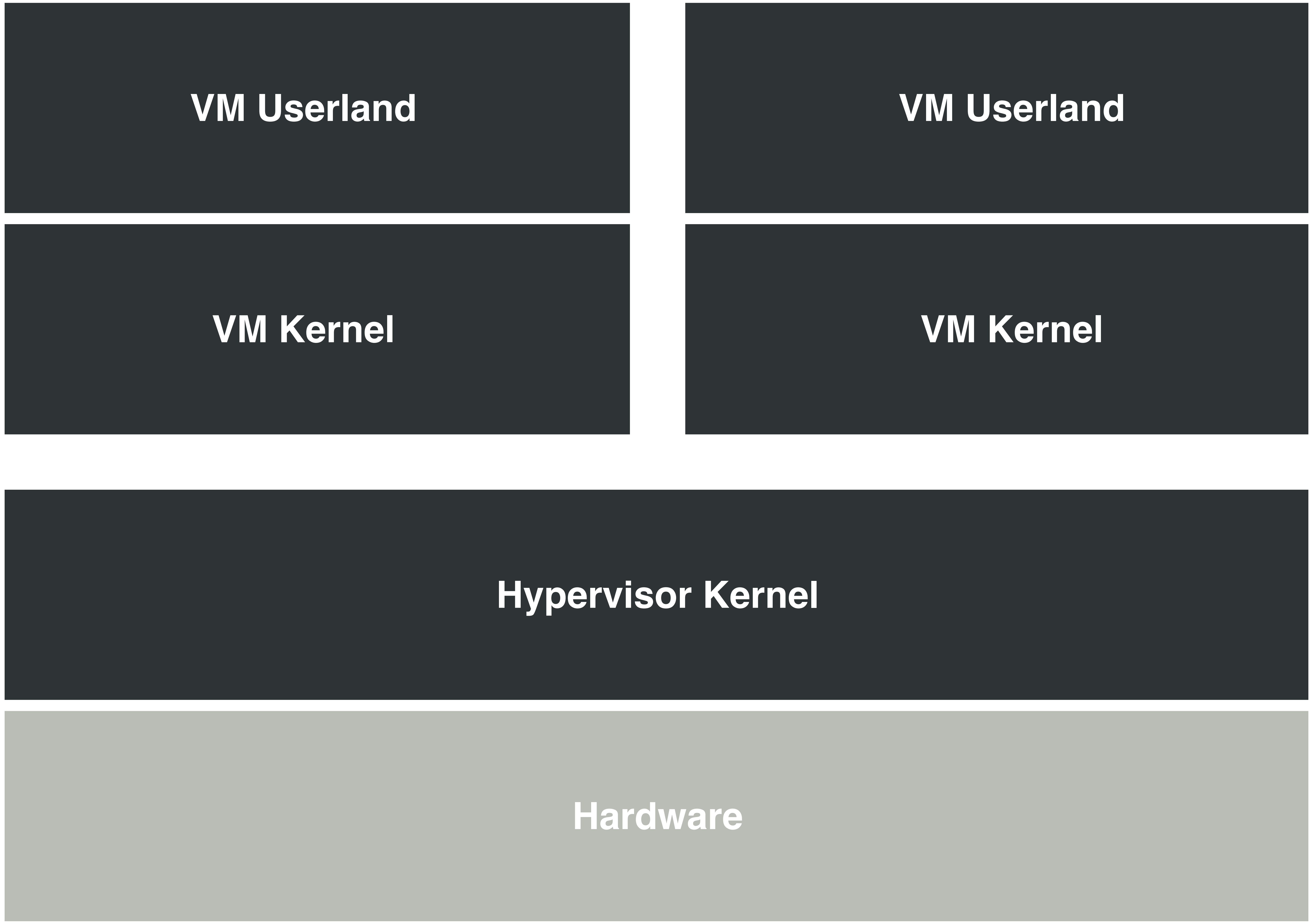
➡Learn by implementing for OpenBSD

Untrusted Environments



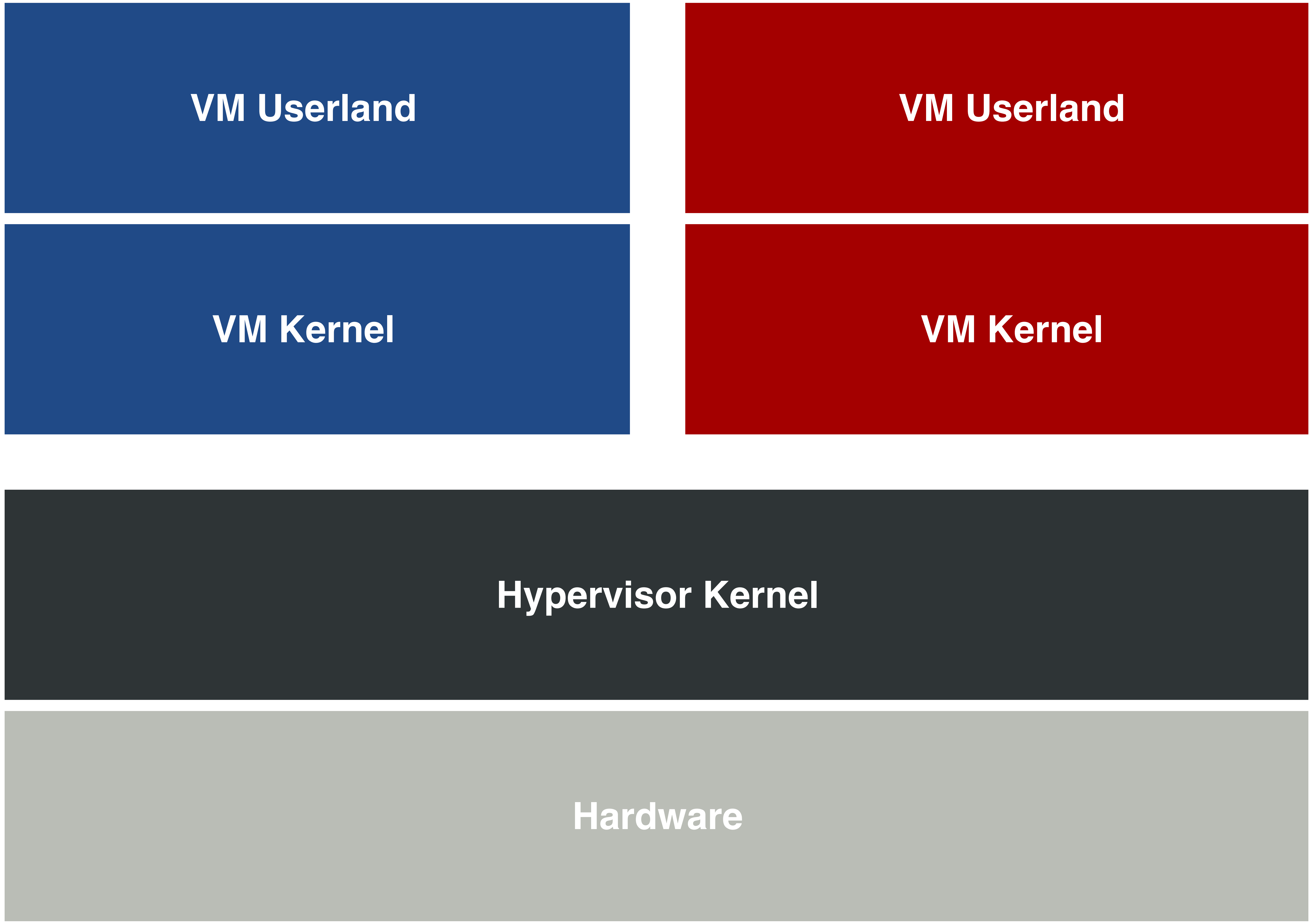
Generic OS

Untrusted Environments



Virtualisation

Untrusted Environments



Confidential VM

Confidential Computing

Claims

- Techniques to protect computing workload from its untrusted environment
  - Data confidentiality
  - Data integrity
  - Code integrity
- Isolation levels
  - Function or library isolation
  - Application isolation

  ⭐Virtual machine isolation

Confidential Computing

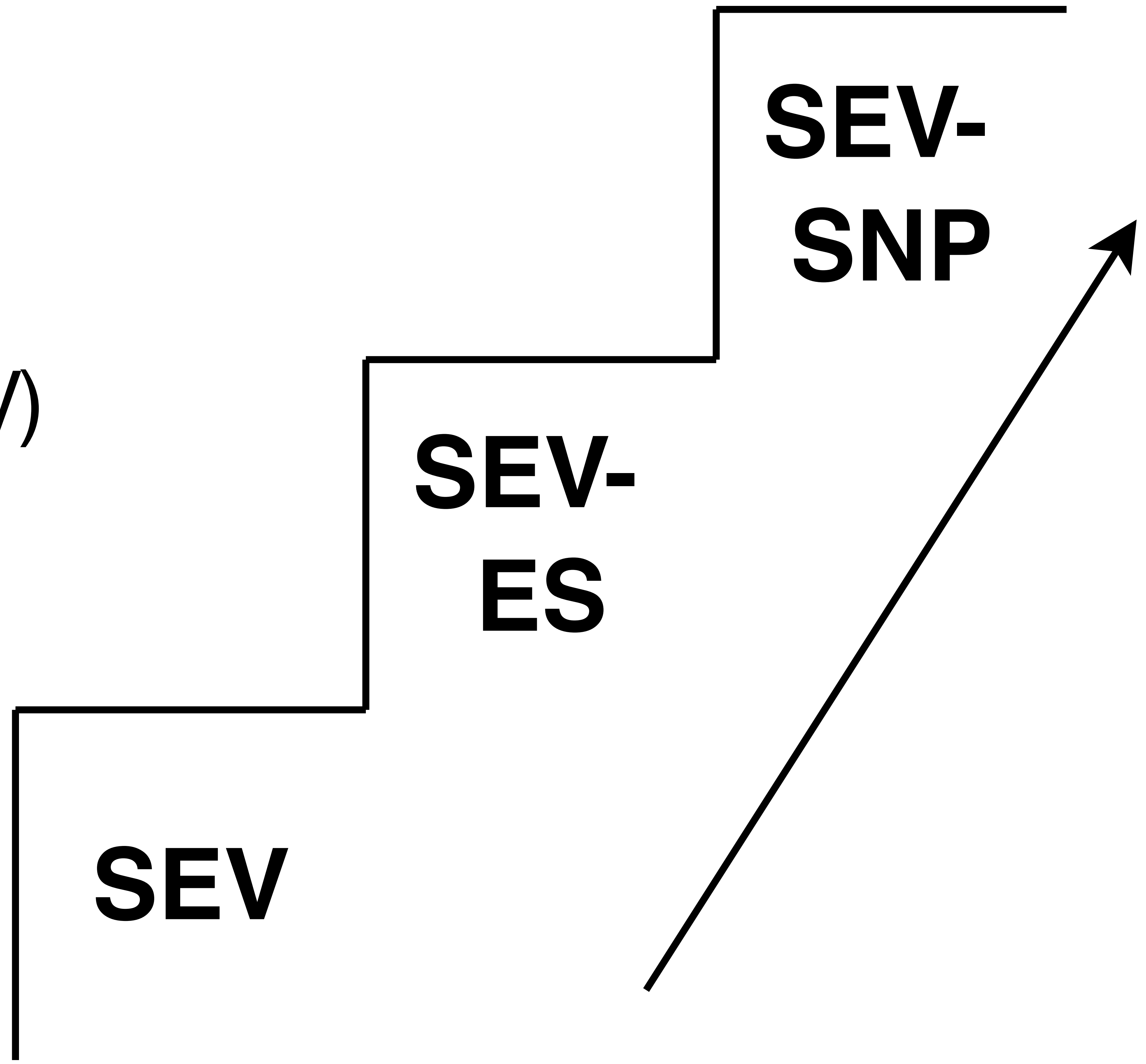
Hardware Support

- Hardware support:
  - ⭐Runtime encryption
  - Attestation
  - Strong isolation
- Examples:
  - **AMD SEV, SEV-ES, SEV-SNP**
  - Intel TDX, Arm CCA

Confidential Computing

Building Blocks

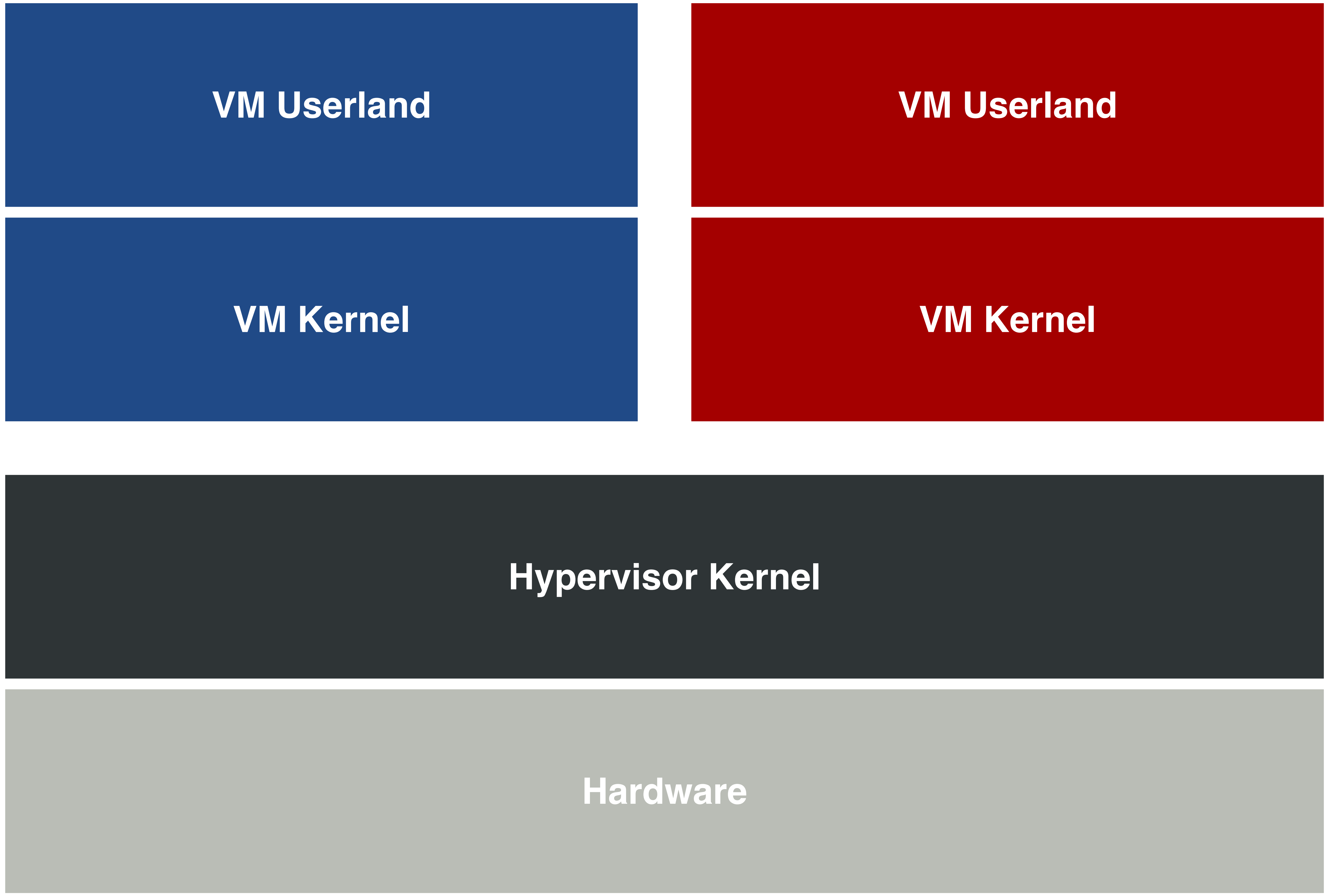
- AMD SEV
    - Secure Encrypted Virtualisation (SEV)
        - Runtime Encryption
    - SEV Encrypted State
        - vCPU State Encryption
    - SEV Secure Nested Paging
        - Integrity Protection



AMD SEV-* Building Blocks

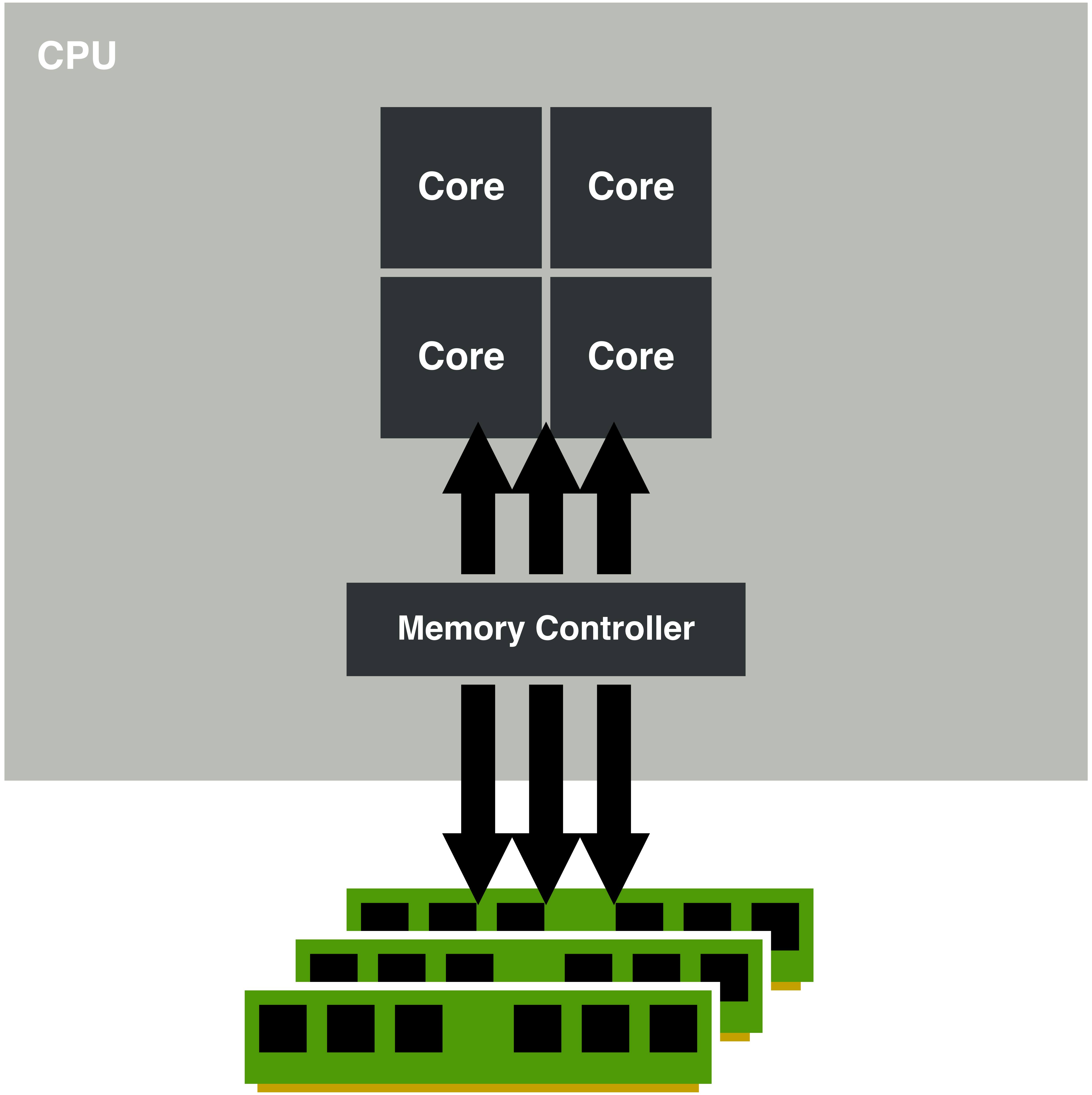
AMD Secure Encrypted Virtualisation

Confidential VM



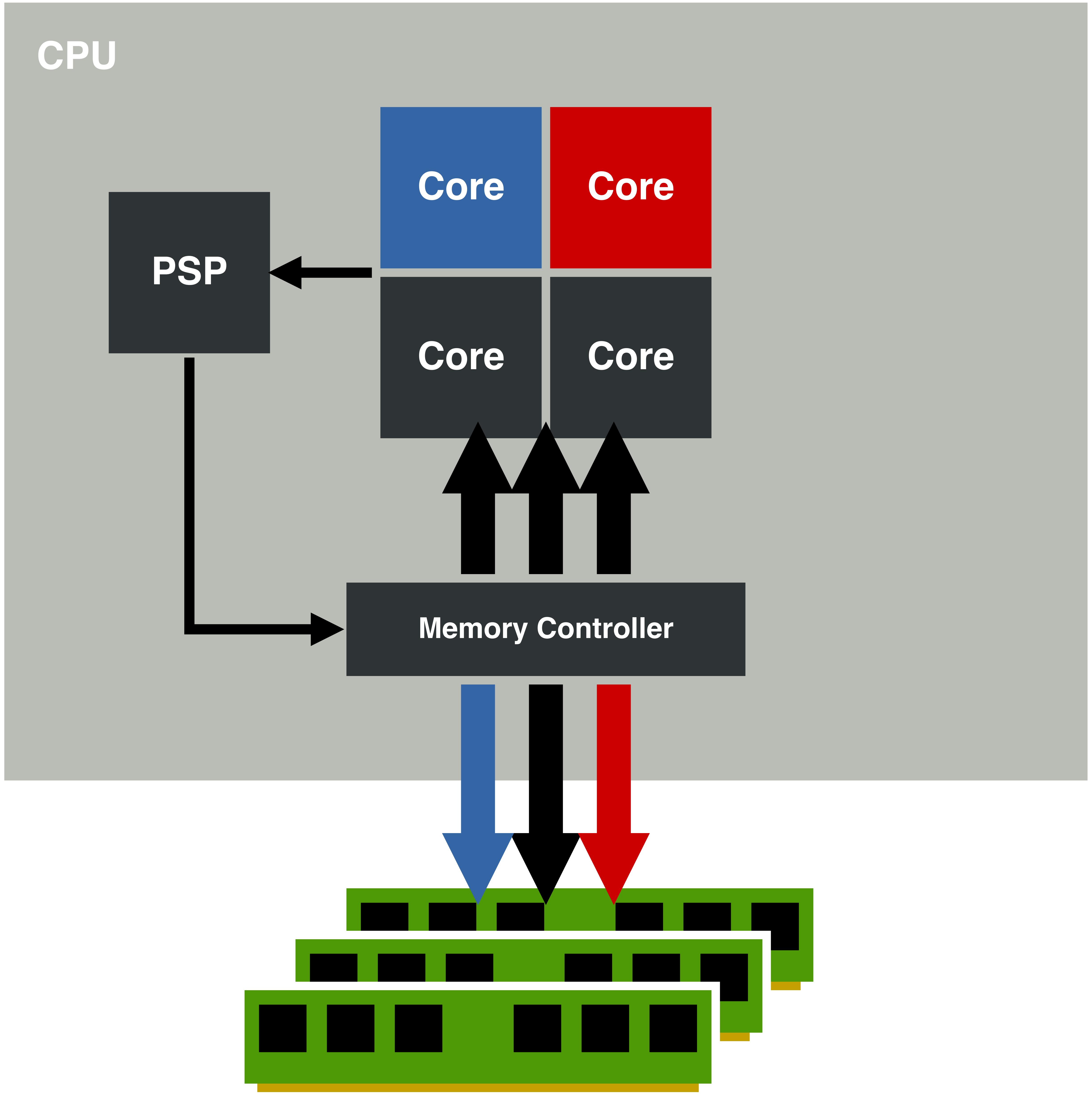
Confidential VM

AMD SEV
Architecture



AMD SEV

Architecture



Confidential Computing with OpenBSD

Agenda

- Introduction
- **First step: Memory encryption for VMs — SEV**
- Next step: vCPU state encryption — SEV-ES
- Conclusion

Confidential Computing for OpenBSD

Goals

- Implement support for AMD SEV-*:
  - psp(4), vmd(8), vmm(4), GENERIC
  - Both host and guest
- Step by step:
  - [x] SEV — OpenBSD 7.6 (October 2024)
  - [x] SEV-ES — OpenBSD 7.8-beta (upcoming release)
  - [ ] SEV-SNP — work in progress
- Compatibility:
  - Linux/KVM host

AMD SEV

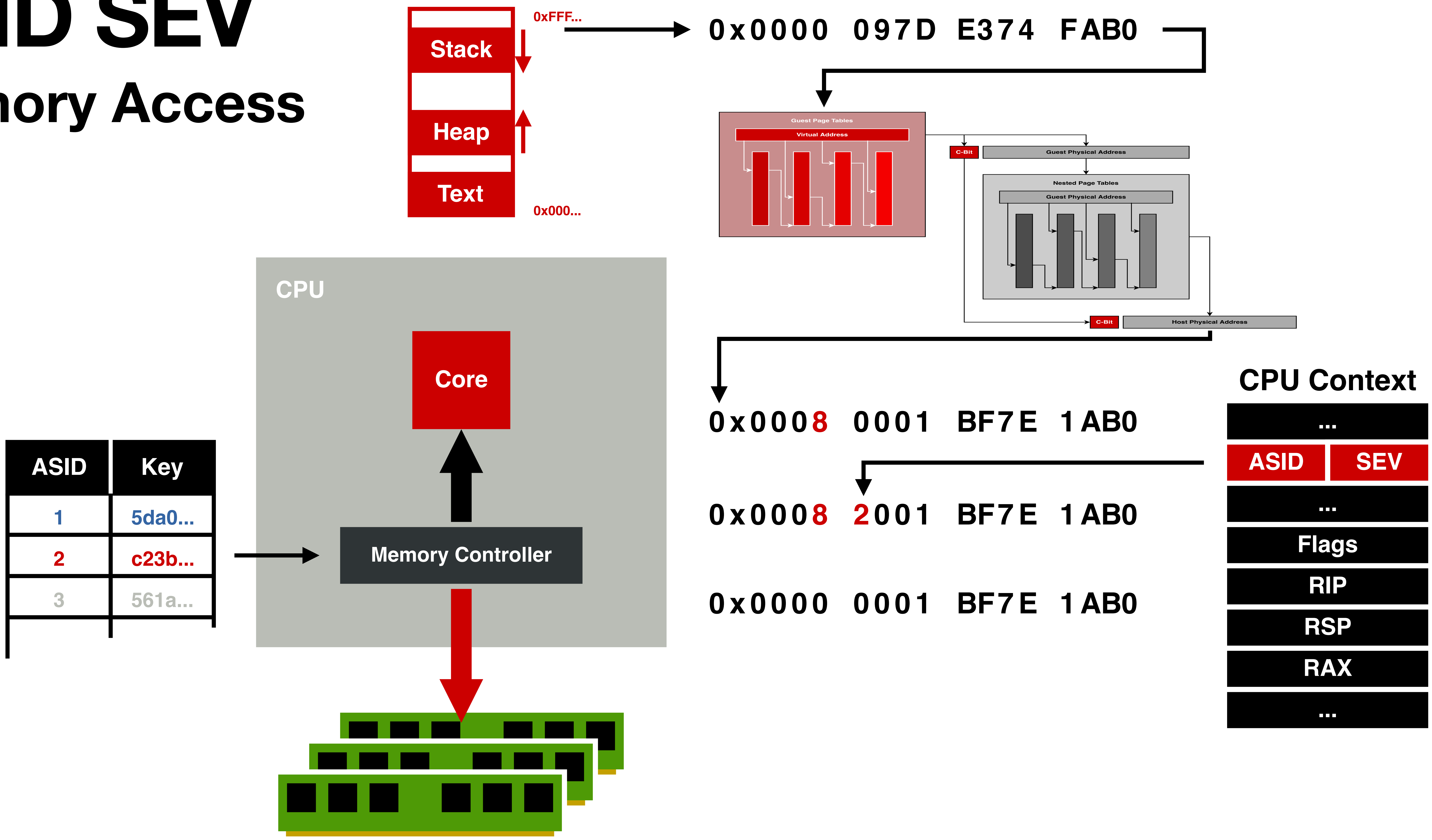
Secure Encrypted Virtualisation

- Guest VM controls encryption!
  - Page tables:
    - “Crypt bit” (C-bit)
    - Private data
    - Public data — shareable:
      - DMA bounce buffers used by virtio(4)
      - Implemented in bus_dma(9)
- Guest and host support in OpenBSD 7.6

AMD SEV

Memory Access

*



AMD SEV

Limitations

- Problem:
  - vCPU state visible to (untrusted) hypervisor
  - Including extended FPU state (AES-NI)
- Solution:
  - SEV-ES
  - Encrypting vCPU state

Confidential Computing with OpenBSD

Agenda

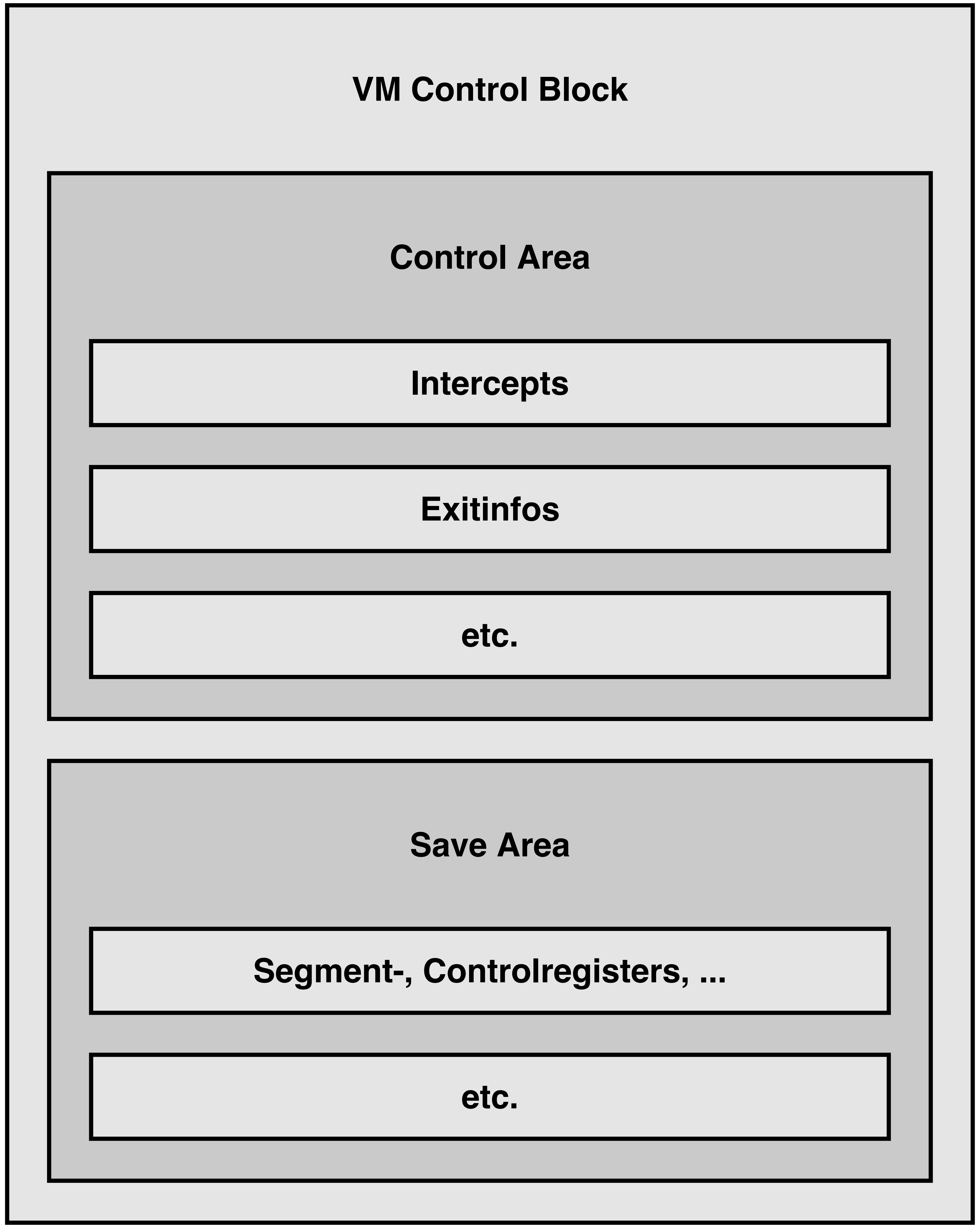
- Introduction
- First step: Memory encryption for VMs — SEV
- **Next step: vCPU state encryption — SEV-ES**
- Conclusion

Saving and restoring state
VM Exit and Entry

- Regular SVM or SEV enabled VM:
  - Minimal state saved in VMCB
  - vmm(4) saves all remaining state in vCPU data structure
    - See exception/interrupt handling and stack frame
- SEV-ES enabled VM:
  - Full vCPU state saved automatically to encrypted VMSA
  - vCPU state invisible (encrypted) for vmm(4)
- Host state saved to Host Save Area

AMD SEV-ES
VMCB

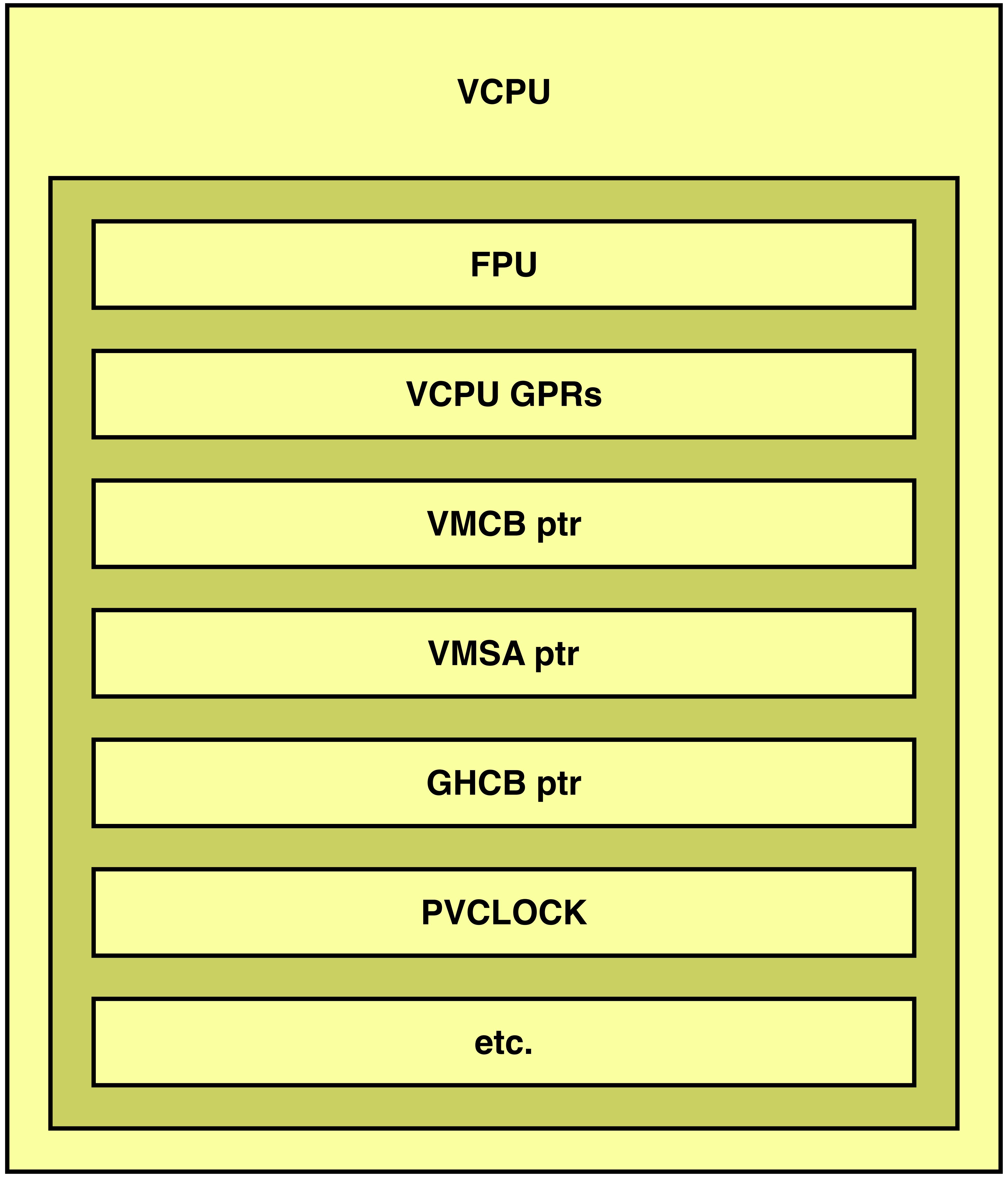
- Virtual Machine Control Block (VMCB)
  - Control Area
  - Save Area
    - Minimal vCPU state
  - VMSAVE and VMLOAD



VMCB

AMD SEV-ES
vCPU

- vCPU data structure
  - Maintained by vmm(4)
  - vCPU state
  - Auxiliary data

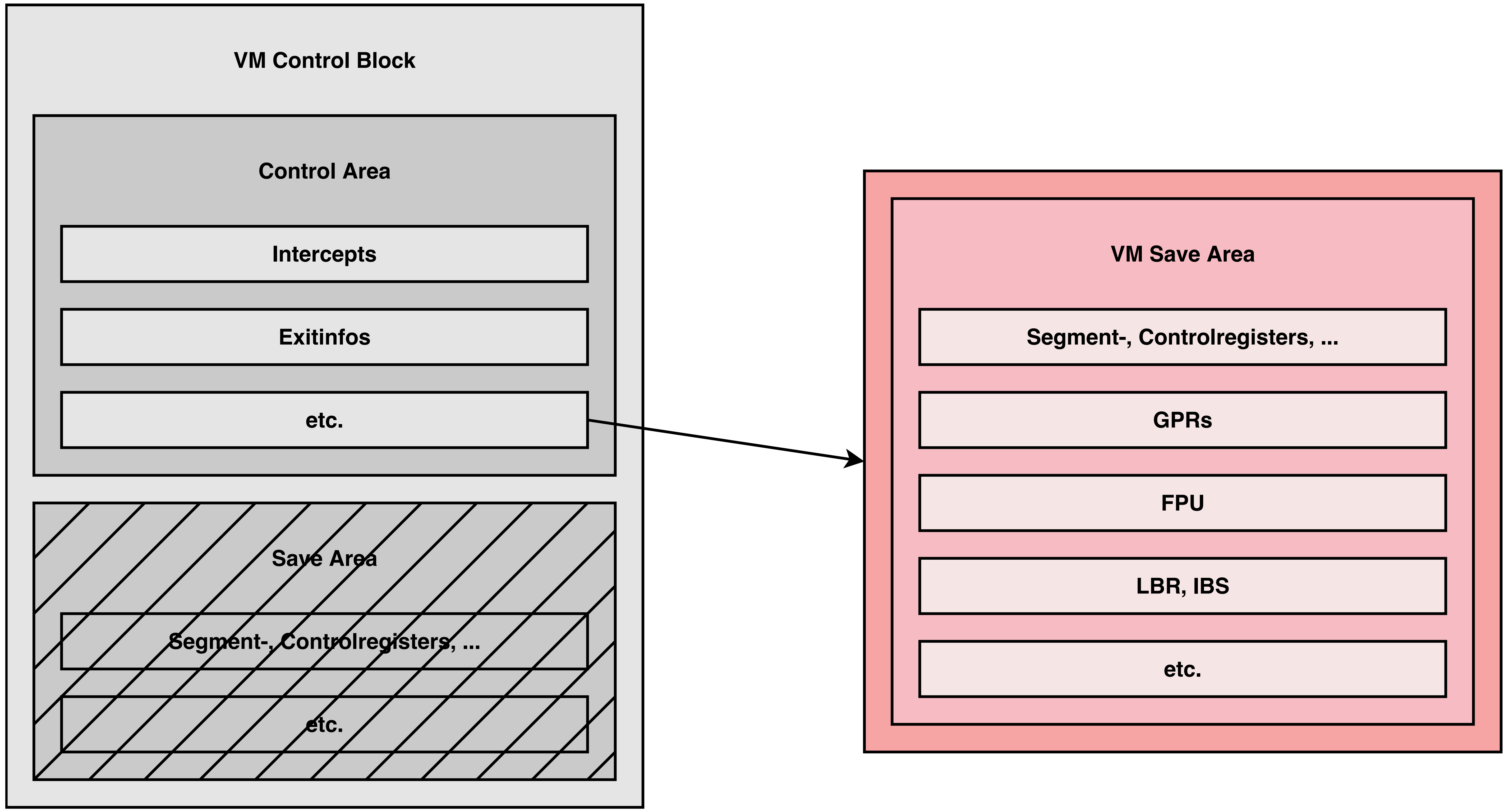


vCPU

AMD SEV-ES

VMSA

- Virtual Machine Save Area
  - Maintained by CPU
  - Full vCPU state
  - Encrypted
- “Swapped” with host state



SVM and SEV

VM Exit and Entry

- VM Exit
  - Minimal state and hidden state saved to VMCB with VMSAVE
  - vCPU state saved by vmm(4)
- VM Entry
  - vCPU state restored by vmm(4)
  - State in VMCB restored with VMLOAD

SVM and SEV

VM Exit and Entry

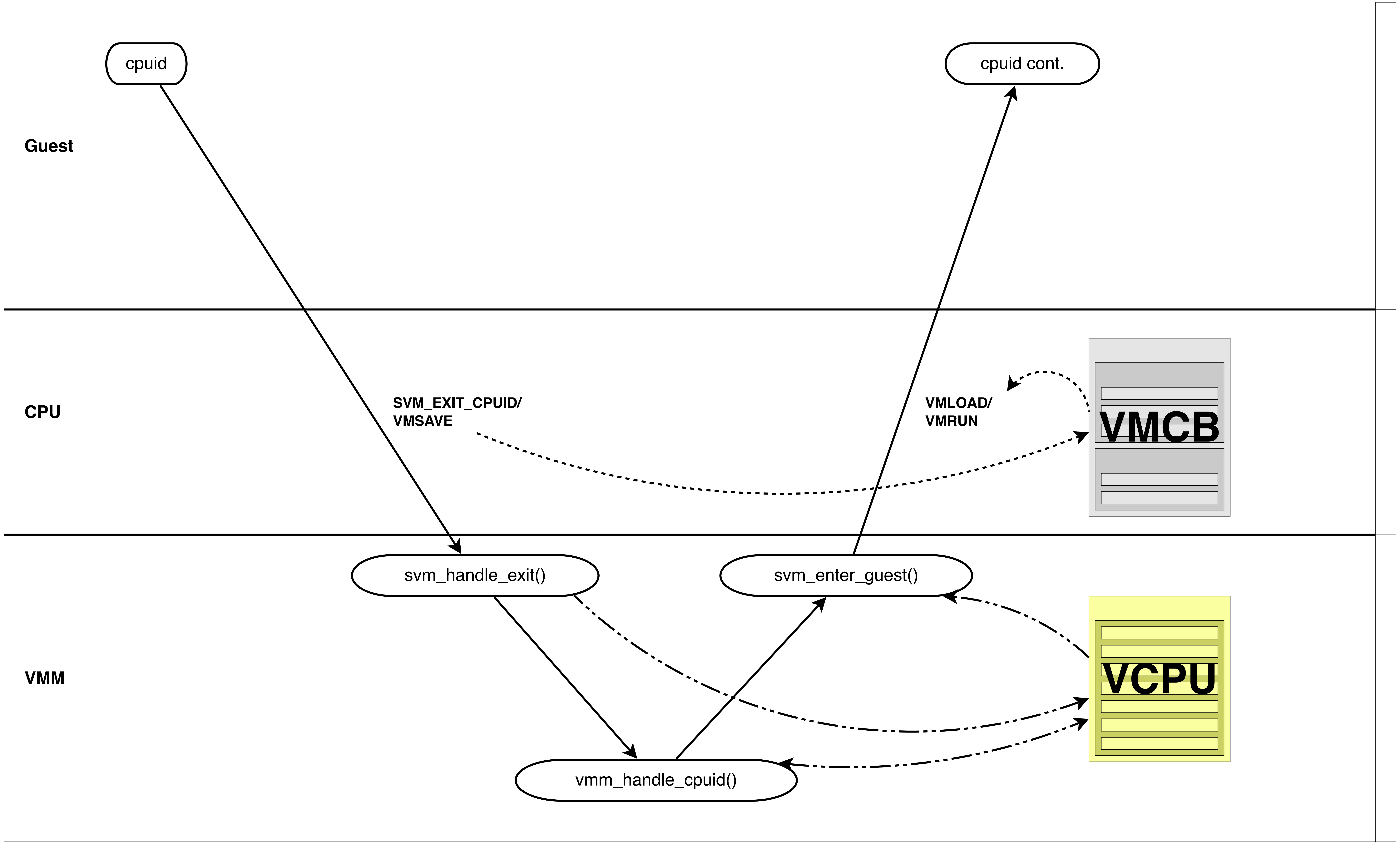
sys/arch/amd64/include/specialreg.h:

```
#define CPUID(code, eax, ebx, ecx, edx)                          \
        __asm volatile("cpuid"                                   \
            : "=a" (eax), "=b" (ebx), "=c" (ecx), "=d" (edx)    \
            : "a" (code))
```

Code (aka function) 0:

- eax: Largest standard function
- ebx, ecx, edx: “AuthenticAMD”, “GenuineIntel”, …

VM Exit



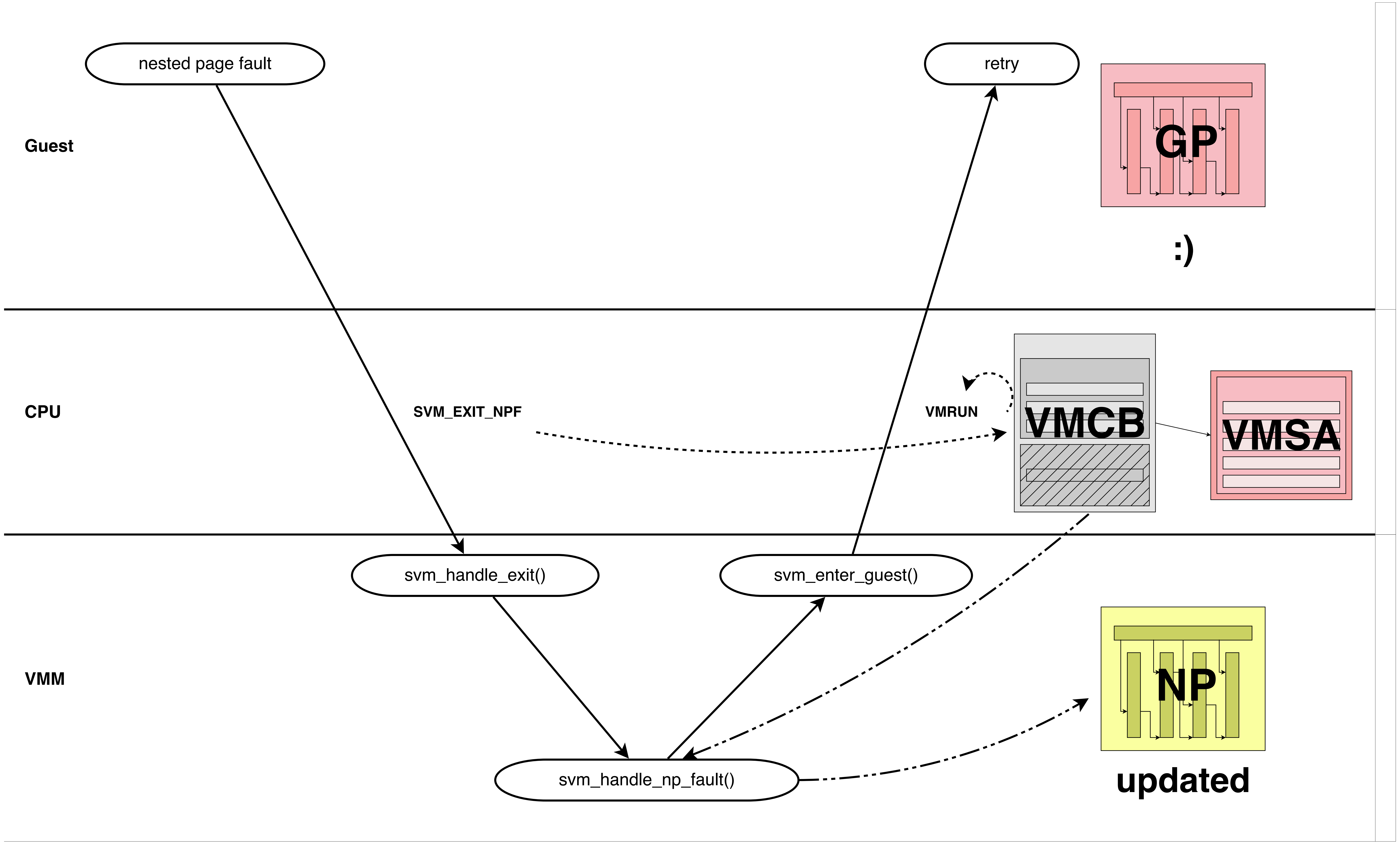
cpuid
cpuid cont.
Guest
CPU
VMM
SVM_EXIT_CPUID/
VMSAVE
VMLOAD/
VMRUN
VMCB
svm_handle_exit()
svm_enter_guest()
VCPU
vmm_handle_cpuid()

AMD SEV-ES
VM Exit Types

- Automatic Exits
  - Asynchronous
  - No vCPU state needed by vmm(4)
- Non-Automatic Exits
  - All other exits
  - Guest decides on what vCPU state to expose

AE VM Exit

*

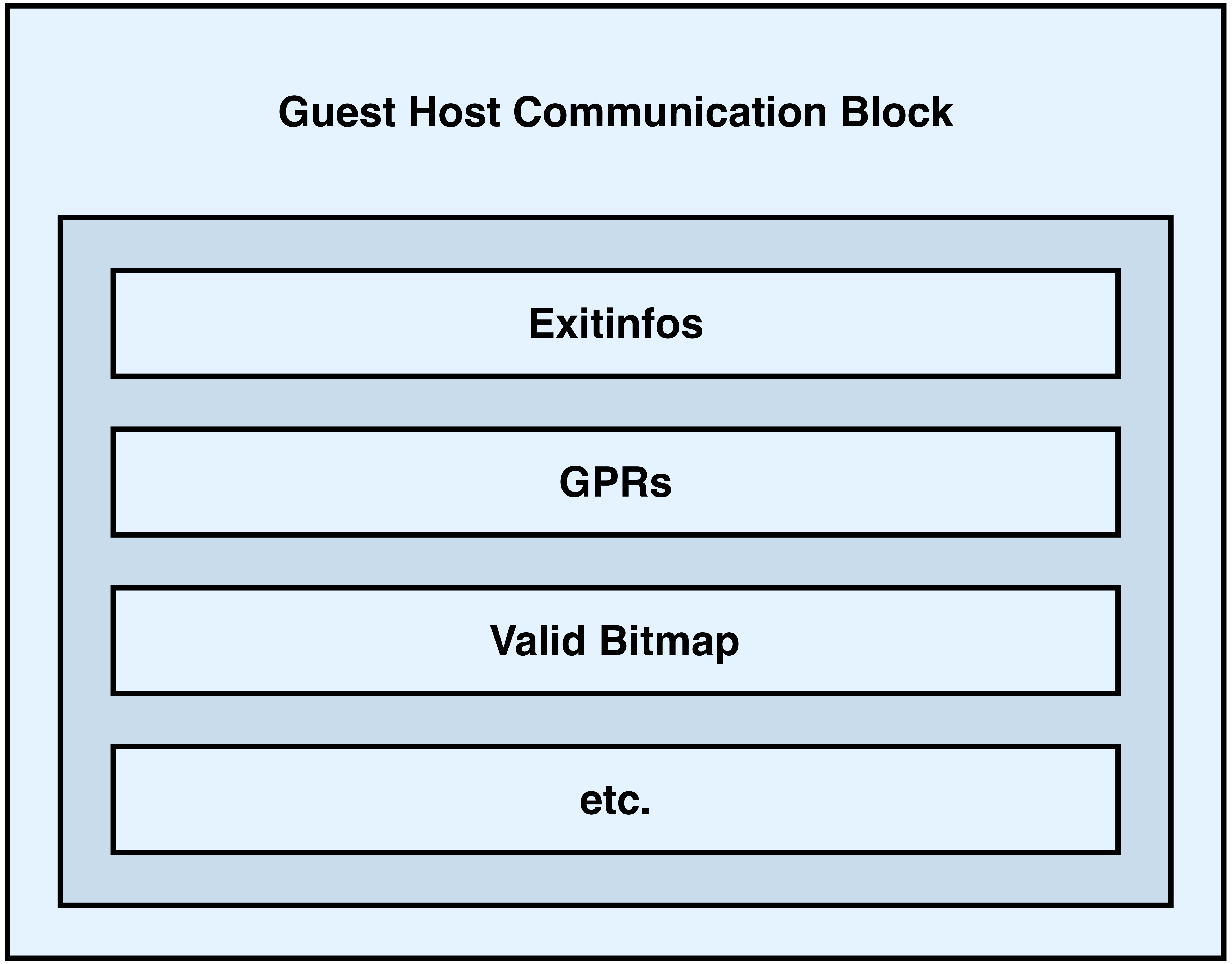


nested page fault
retry
Guest
GP
:)
CPU
SVM_EXIT_NPF
VMRUN
VMCB
VMSA
svm_handle_exit()
svm_enter_guest()
VMM
NP
svm_handle_np_fault()
updated

#VC Trap

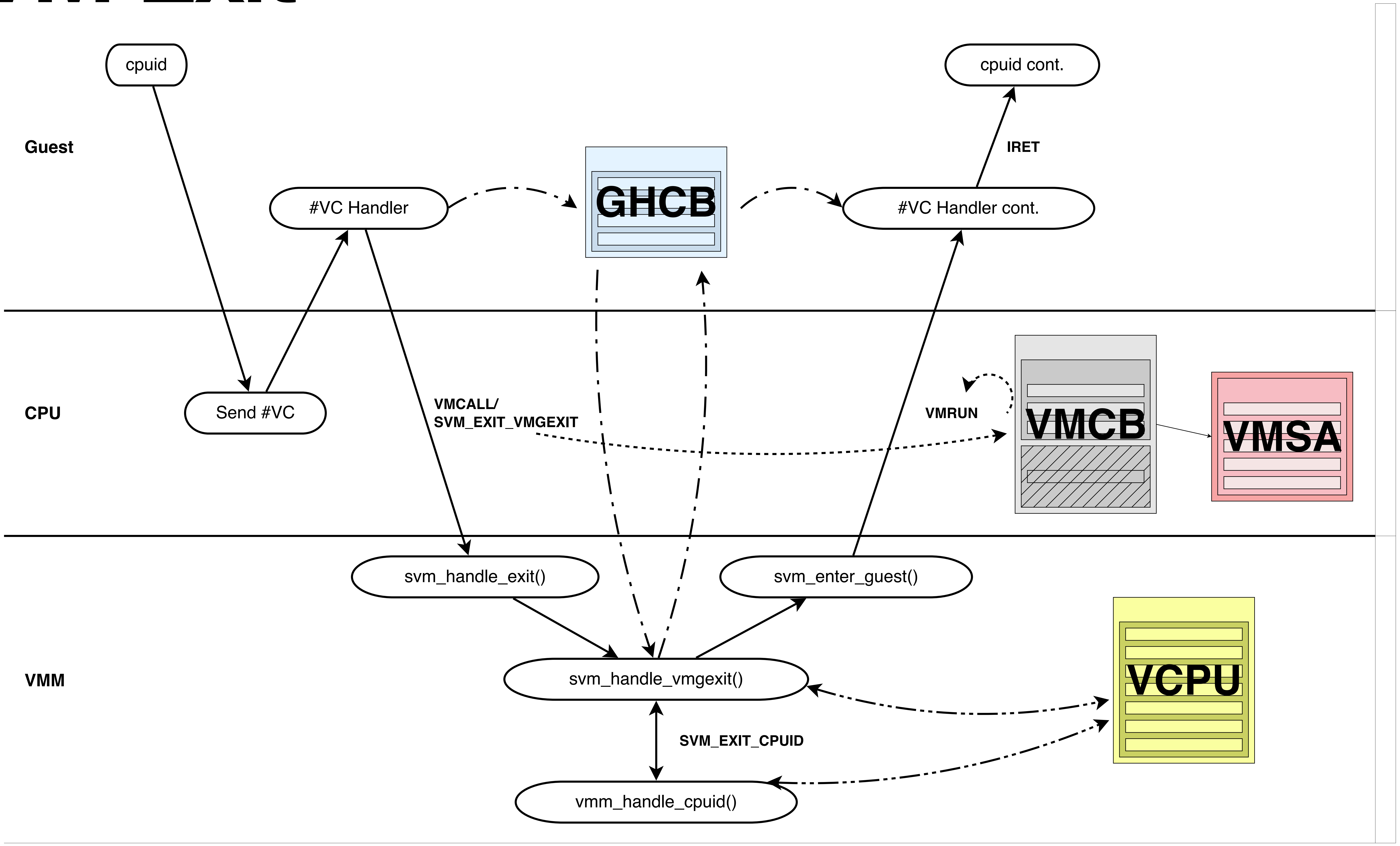
Non-Automatic VM Exits

- VM Exit redirected to #VC trap handler
- Guest decides on what vCPU state to be shared with vmm(4)
- Software defined Guest Host Communication Block (GHCB)
- Unencrypted memory shared by guest with vmm(4)
- GHCB MSR points to GPA of GHCB
- GHCB MSR visible in VMCB @0xA0



GHCB

NAE VM Exit



Guest
CPU
VMM
cpuid
cpuid cont.
IRET
#VC Handler
GHCB
#VC Handler cont.
Send #VC
VMCALL/
SVM_EXIT_VMGEXIT
VMRUN
VMCB
VMSA
svm_handle_exit()
svm_enter_guest()
svm_handle_vmgexit()
VCPU
SVM_EXIT_CPUID
vmm_handle_cpuid()

SEV-ES Bootstrap

*

#VC in locore0

- Challenge:
  - CPUID might raise #VC
  - Guest is not “enlightened” yet
  - Plain GENERIC kernel
- Tentative #VC handler:
  - No SEV-ES, nothing happens, all fine :)
  - SEV-ES enabled guest:
    - Handle #VC trap

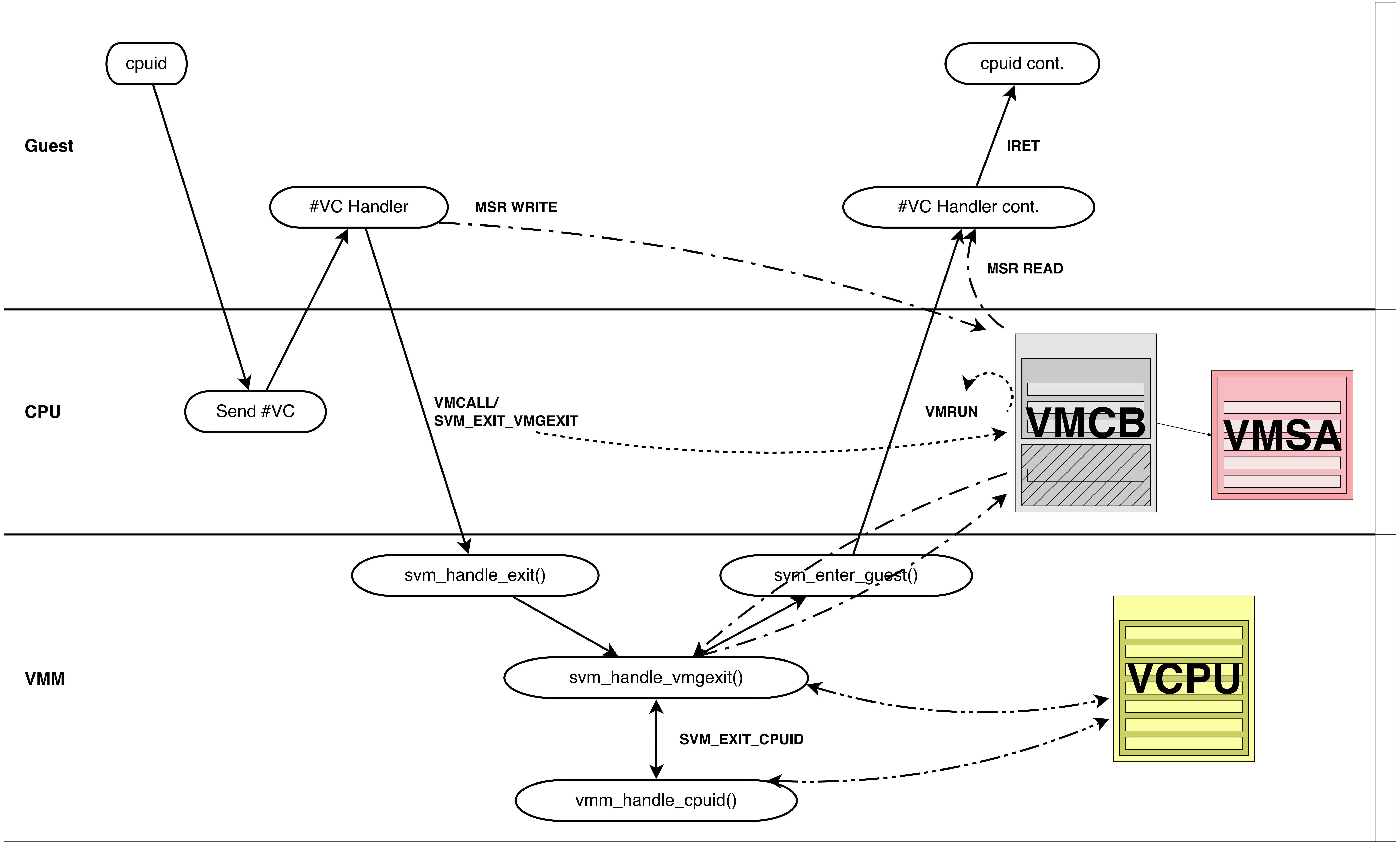
SEV-ES Bootstrap

*

#VC in locore0

- #VC handler:
  - Paging not enabled yet
  - No GHCB shared with vmm(4)
- GHCB MSR protocol:
  - Use low 12 bits to encode requests
  - WRMSR followed by VMCALL/VMGEXIT
  - vmm(4) encodes response as GHCB GPA in VMCB (@0xA0)
  - RDMSR by guest

GHCB MSR Protocol



Paravirtualisation *

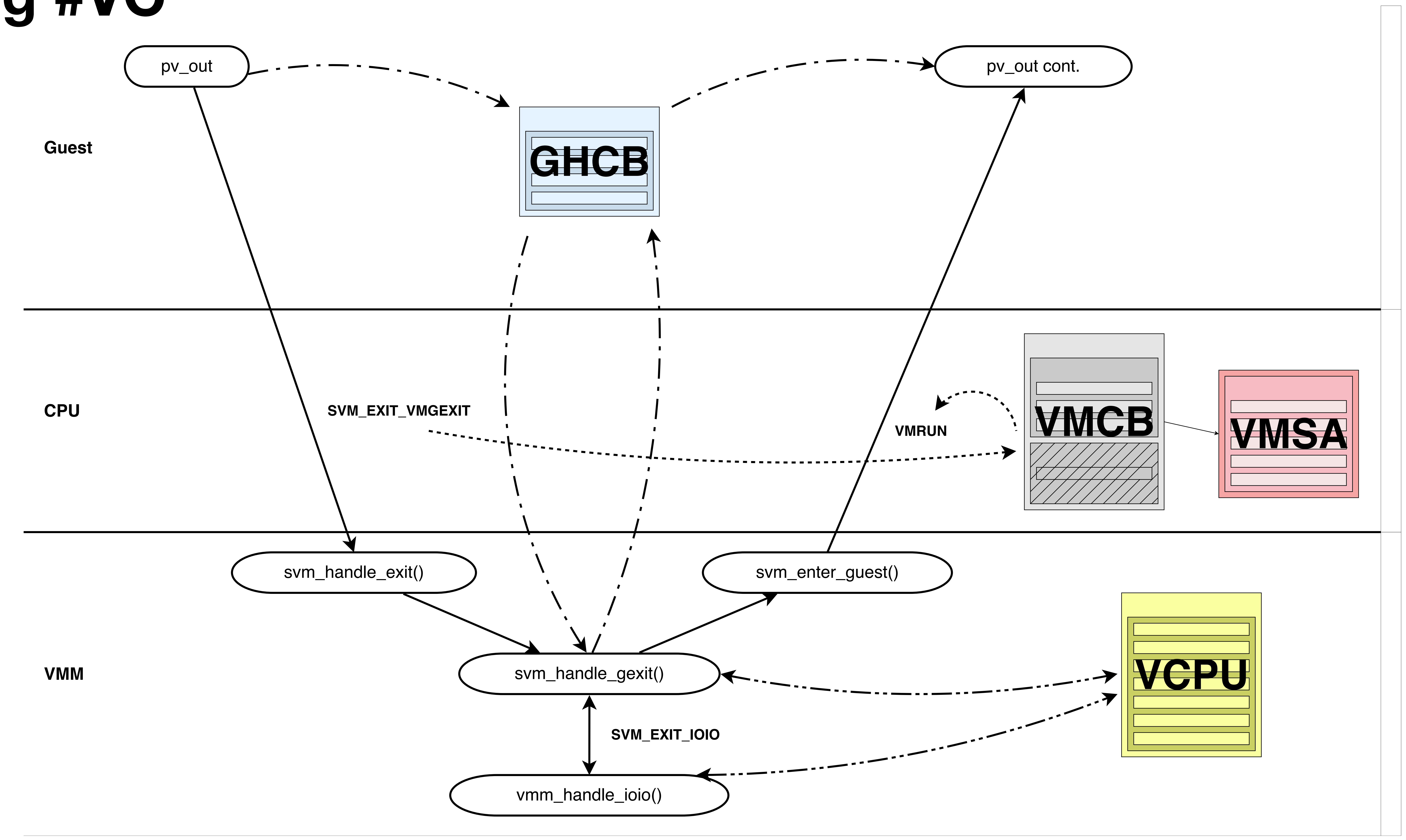
Avoiding #VC

- vmm(4) emulates PIC i8259
  - OUT in IRQ handler
  - Each IRQ raises several #VC
- Paravirtualising IN/OUT
  - When SEV-ES is enabled
  - Codepatch IN/OUT with paravirtualised version
  - Completely avoids #VC after bootstrapping the kernel

Paravirtualization

*

Avoiding #VC



Confidential Computing with OpenBSD

Agenda

- Introduction
- First step: Memory encryption for VMs — SEV
- Next step: vCPU state encryption — SEV-ES
- **Conclusion**

Conclusion

Next step and beyond

- SEV-ES works with OpenBSD 7.8-beta:
    - GENERIC supports both host and guest
    - psp(4), vmm(4) and vmd(8) support implemented/updated
    - Integrated into source tree; to be released
- Next goals
    - IOMMU support for SEV-SNP
    - vmm(4)/vmd(8) support for SEV-SNP
    - OpenBSD SEV-SNP guest already running on Linux/KVM host
    - BSDBoot/Kernel Exec?
    - Performance?
    - Attestation?
    - SMP?
    - ...

Thank you!

Questions?

Don’t forget to remember!