Cloud Operating Systems

Virtual Memory and Structural Setup

Fabian Rauscher, Daniel Gruss, Andreas Kogler

2023-03-13

Setup

Free pages 994 F9 MemInfo F10 Locks F11 Stacktrace F12 Threads

```
This is on term 0, you should see me now
Kernel end address is 0xffffffff80165000
Now enabling Interrupts...

SWEB-Pseudo-Shell starting...


SWEB: />
```

Setup

- Upstream: https://github.com/IAIK/sweb

Setup

- Upstream: https://github.com/IAIK/sweb
- SWEB Tutorials: https://www.iaik.tugraz.at/teaching/materials/os/tutorials/

Setup

- Upstream: https://github.com/IAIK/sweb
- SWEB Tutorials: https://www.iaik.tugraz.at/teaching/materials/os/tutorials/
- Useful links in the Discord `cloudos-announcements` channel

Setup

- Upstream: https://github.com/IAIK/sweb
- SWEB Tutorials: https://www.iaik.tugraz.at/teaching/materials/os/tutorials/
- Useful links in the Discord `cloudos-announcements` channel
- We recommend you work on Linux

Setup

Building and running SWEB

Building and running SWEB

- mkdir -p /tmp/sweb

Building and running SWEB

- mkdir -p /tmp/sweb
- cd /tmp/sweb

Building and running SWEB

- mkdir -p /tmp/sweb
- cd /tmp/sweb
- cmake /path/to/sourcecode/of/sweb

Building and running SWEB

- mkdir -p /tmp/sweb
- cd /tmp/sweb
- cmake /path/to/sourcecode/of/sweb
- make

Building and running SWEB

- mkdir -p /tmp/sweb
- cd /tmp/sweb
- cmake /path/to/sourcecode/of/sweb
- make
- make **kvm**

- Condition (condition variable)

- Condition (condition variable)
- Mutex

- Condition (condition variable)
- Mutex
- PageManager (allocate and free pages)

- Condition (condition variable)
- Mutex
- PageManager (allocate and free pages)
- ArchMemory (manages virtual address space)

- Condition (condition variable)
- Mutex
- PageManager (allocate and free pages)
- ArchMemory (manages virtual address space)
- main (boot code)

- Condition (condition variable)
- Mutex
- PageManager (allocate and free pages)
- ArchMemory (manages virtual address space)
- main (boot code)
- Syscall (syscall handeling)

- Condition (condition variable)
- Mutex
- PageManager (allocate and free pages)
- ArchMemory (manages virtual address space)
- main (boot code)
- Syscall (syscall handeling)
- uSTL (STL, provides vector, string, map,...)

- Condition (condition variable)
- Mutex
- PageManager (allocate and free pages)
- ArchMemory (manages virtual address space)
- main (boot code)
- Syscall (syscall handeling)
- uSTL (STL, provides vector, string, map,...)
- Loader (user space binary loader)

- Condition (condition variable)
- Mutex
- PageManager (allocate and free pages)
- ArchMemory (manages virtual address space)
- main (boot code)
- Syscall (syscall handeling)
- uSTL (STL, provides vector, string, map,...)
- Loader (user space binary loader)

→ For more info:
https://www.iaik.tugraz.at/teaching/materials/os/tutorials/

VMX

What is VMX?

What is VMX?

What is VMX?

What is VMX?

- Intels Virtual-Machine Extension

What is VMX?

- Intels Virtual-Machine Extension
- Provides hardware support for virtualization

What is VMX?

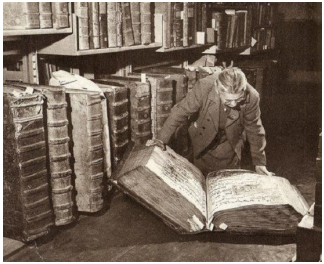
- Intels Virtual-Machine Extension
- Provides hardware support for virtualization
- Two different classes of software:

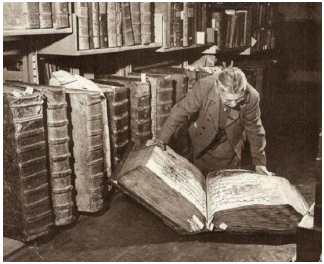
What is VMX?

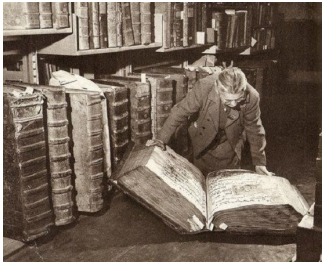
- Intels Virtual-Machine Extension
- Provides hardware support for virtualization
- Two different classes of software:
  - Virtual-machine monitors (VMM)

What is VMX?

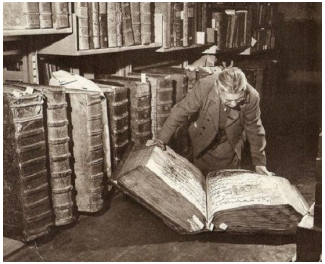
- Intels Virtual-Machine Extension
- Provides hardware support for virtualization
- Two different classes of software:
  - Virtual-machine monitors (VMM)
  - Guest software



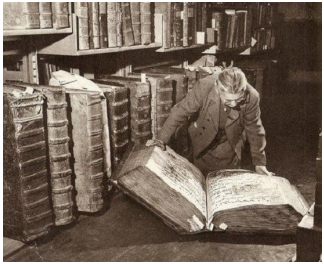




- DO NOT READ IT LIKE A BOOK

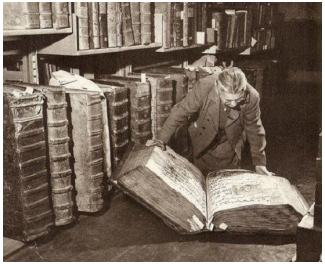


- DO NOT READ IT LIKE A BOOK
- Look things up that you need



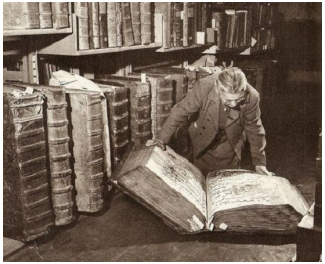
- DO NOT READ IT LIKE A BOOK
- Look things up that you need
- Important volumes for this lecture:

Intel SDM

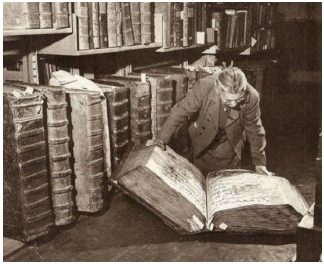


- DO NOT READ IT LIKE A BOOK
- Look things up that you need
- Important volumes for this lecture:
  - Volume 3C: virtual machine extensions (vmx)

Intel SDM



- DO NOT READ IT LIKE A BOOK
- Look things up that you need
- Important volumes for this lecture:
  - Volume 3C: virtual machine extensions (vmx)
  - Volume 3D: appendices with values for constants



- DO NOT READ IT LIKE A BOOK
- Look things up that you need
- Important volumes for this lecture:
  - Volume 3C: virtual machine extensions (vmx)
  - Volume 3D: appendices with values for constants
  - Volume 3A: operating-system support environment

- Gives the guest the illusion of running on real hardware

- Gives the guest the illusion of running on real hardware
- Has full control of the processors recources

- Gives the guest the illusion of running on real hardware
- Has full control of the processors recources
- Manages the ability of the guest to access

- Gives the guest the illusion of running on real hardware
- Has full control of the processors recources
- Manages the ability of the guest to access
  - Processor Recources

- Gives the guest the illusion of running on real hardware
- Has full control of the processors recources
- Manages the ability of the guest to access
  - Processor Recources
  - Physical Memory

- Gives the guest the illusion of running on real hardware
- Has full control of the processors recources
- Manages the ability of the guest to access
  - Processor Recources
  - Physical Memory
  - Interrupts

- Gives the guest the illusion of running on real hardware
- Has full control of the processors recources
- Manages the ability of the guest to access
  - Processor Recources
  - Physical Memory
  - Interrupts
  - I/O

- Gives the guest the illusion of running on real hardware
- Has full control of the processors recources
- Manages the ability of the guest to access
  - Processor Recources
  - Physical Memory
  - Interrupts
  - I/O
  - ...

Getting Started

Enabling VMX

[1] Intel SDM Volume 3, 24.7

[1] Intel SDM Volume 3, 24.7

[1] Intel SDM Volume 3, 24.7

Enabling VMX

- Check for VMX support: CPUID.1:ECX.VMX[bit 5] = 1

[1] Intel SDM Volume 3, 24.7

Enabling VMX

- Check for VMX support: CPUID.1:ECX.VMX[bit 5] = 1
- Enable VMX by setting bit 13 in CR4 [1]

[1] Intel SDM Volume 3, 24.7

VMX Operation

- VMX root operation

VMX Operation

- VMX root operation
  - new instructions (VMX instructions)

VMX Operation

- VMX root operation
  - new instructions (VMX instructions)
  - restrictions on certain control register values

VMX Operation

- VMX root operation
  - new instructions (VMX instructions)
  - restrictions on certain control register values
  - used for the VMM

VMX Operation

- VMX root operation
  - new instructions (VMX instructions)
  - restrictions on certain control register values
  - used for the VMM
- VMX non-root operation

- VMX root operation
  - new instructions (VMX instructions)
  - restrictions on certain control register values
  - used for the VMM
- VMX non-root operation
  - more restricted than normal operation

- VMX root operation
  - new instructions (VMX instructions)
  - restrictions on certain control register values
  - used for the VMM
- VMX non-root operation
  - more restricted than normal operation
  - certain instructions and events cause VM exits

VMX Operation

- VMX root operation
  - new instructions (VMX instructions)
  - restrictions on certain control register values
  - used for the VMM
- VMX non-root operation
  - more restricted than normal operation
  - certain instructions and events cause VM exits
  - used for the guest

- VMX root operation
  - new instructions (VMX instructions)
  - restrictions on certain control register values
  - used for the VMM
- VMX non-root operation
  - more restricted than normal operation
  - certain instructions and events cause VM exits
  - used for the guest
- Transition between the two

VMX Operation

- VMX root operation
  - new instructions (VMX instructions)
  - restrictions on certain control register values
  - used for the VMM
- VMX non-root operation
  - more restricted than normal operation
  - certain instructions and events cause VM exits
  - used for the guest
- Transition between the two
  - VM entries: VMX root operation $\rightarrow$ VMX non-root operation

VMX Operation

- VMX root operation
  - new instructions (VMX instructions)
  - restrictions on certain control register values
  - used for the VMM
- VMX non-root operation
  - more restricted than normal operation
  - certain instructions and events cause VM exits
  - used for the guest
- Transition between the two
  - VM entries: VMX root operation $\rightarrow$ VMX non-root operation
  - VM exits: VMX non-root operation $\rightarrow$ VMX root operation

Entering VMX Operation (Part 1)

[1] Intel SDM Volume 3, 24.8
[2] Intel SDM Volume 3, 24.7

Entering VMX Operation (Part 1)

[1] Intel SDM Volume 3, 24.8
[2] Intel SDM Volume 3, 24.7

Entering VMX Operation (Part 1)

Preparations

[1] Intel SDM Volume 3, 24.8
[2] Intel SDM Volume 3, 24.7

Entering VMX Operation (Part 1)

Preparations

- Setup CR0[1]

[1] Intel SDM Volume 3, 24.8
[2] Intel SDM Volume 3, 24.7

Preparations

- Setup CR0[1]
  - Set bits specified by IA32_VMX_CR0_FIXED0 (0x486)

[1] Intel SDM Volume 3, 24.8
[2] Intel SDM Volume 3, 24.7

Preparations

- Setup CR0[1]
  - Set bits specified by IA32_VMX_CR0_FIXED0 (0x486)
  - Clear bits specified by IA32_VMX_CR0_FIXED1 (0x487)

[1] Intel SDM Volume 3, 24.8
[2] Intel SDM Volume 3, 24.7

Entering VMX Operation (Part 1)

Preparations

- Setup CR0[1]
  - Set bits specified by IA32_VMX_CR0_FIXED0 (0x486)
  - Clear bits specified by IA32_VMX_CR0_FIXED1 (0x487)
- Setup CR4[1]

[1] Intel SDM Volume 3, 24.8
[2] Intel SDM Volume 3, 24.7

Entering VMX Operation (Part 1)

Preparations

- Setup CR0[1]
  - Set bits specified by IA32_VMX_CR0_FIXED0 (0x486)
  - Clear bits specified by IA32_VMX_CR0_FIXED1 (0x487)
- Setup CR4[1]
  - Set bits specified by IA32_VMX_CR4_FIXED0 (0x488)

[1] Intel SDM Volume 3, 24.8
[2] Intel SDM Volume 3, 24.7

Entering VMX Operation (Part 1)

Preparations

- Setup CR0[1]
  - Set bits specified by IA32_VMX_CR0_FIXED0 (0x486)
  - Clear bits specified by IA32_VMX_CR0_FIXED1 (0x487)
- Setup CR4[1]
  - Set bits specified by IA32_VMX_CR4_FIXED0 (0x488)
  - Clear bits specified by IA32_VMX_CR4_FIXED1 (0x489)

[1] Intel SDM Volume 3, 24.8
[2] Intel SDM Volume 3, 24.7

Entering VMX Operation (Part 1)

Preparations

- Setup CR0[1]
  - Set bits specified by IA32_VMX_CR0_FIXED0 (0x486)
  - Clear bits specified by IA32_VMX_CR0_FIXED1 (0x487)
- Setup CR4[1]
  - Set bits specified by IA32_VMX_CR4_FIXED0 (0x488)
  - Clear bits specified by IA32_VMX_CR4_FIXED1 (0x489)
- **Ensure** that bit 2 of IA32_FEATURE_CONTROL (0x3A) is set[2]

[1] Intel SDM Volume 3, 24.8
[2] Intel SDM Volume 3, 24.7

[1] Intel SDM Volume 3, 31

Entering VMX Operation (Part 2)

[1] Intel SDM Volume 3, 31

VMXON region [2]

[1] Intel SDM Volume 3, 31

Entering VMX Operation (Part 2)

VMXON region [2]

- Used by the processor to support VMX operation

[1] Intel SDM Volume 3, 31

VMXON region [2]

- Used by the processor to support VMX operation
- Up to 4KB in size

[1] Intel SDM Volume 3, 31

Entering VMX Operation (Part 2)

VMXON region [2]

- Used by the processor to support VMX operation
- Up to 4KB in size
- VMXON pointer needs to be a 4KB aligned valid physical address

[1] Intel SDM Volume 3, 31

Entering VMX Operation (Part 2)

VMXON region [2]

- Used by the processor to support VMX operation
- Up to 4KB in size
- VMXON pointer needs to be a 4KB aligned valid physical address
- Bits 30:0 must contain the VMCS revision identifier (IA32_VMX_BASIC, 0x480)

[1] Intel SDM Volume 3, 31

Entering VMX Operation (Part 2)

VMXON region [2]

- Used by the processor to support VMX operation
- Up to 4KB in size
- VMXON pointer needs to be a 4KB aligned valid physical address
- Bits 30:0 must contain the VMCS revision identifier (IA32_VMX_BASIC, 0x480)
- Can be loaded using the VMXON instruction to enter VMX operation

[1] Intel SDM Volume 3, 31

- VMXON

New Instructions

- VMXON
- VMXOFF

- VMXON
- VMXOFF
- INVEPT

New Instructions

- VMXON
- VMXOFF
- INVEPT
- INVVPID

- VMXON
- VMXOFF
- INVEPT
- INVVPID
- VMCLEAR

- VMXON
- VMXOFF
- INVEPT
- INVVPID
- VMCLEAR
- VMPTRLD

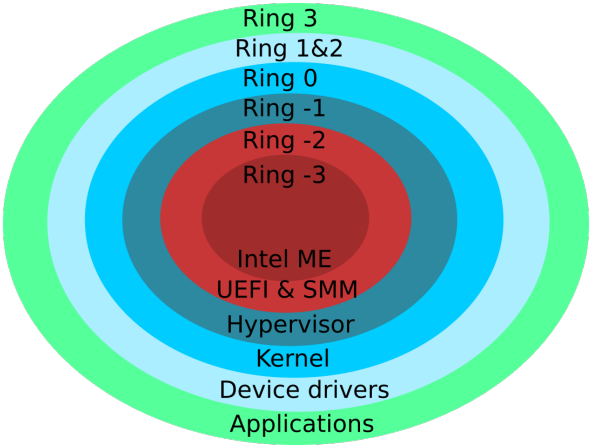
- VMXON
- VMXOFF
- INVEPT
- INVVPID
- VMCLEAR
- VMPTRLD
- VMPTRSTR

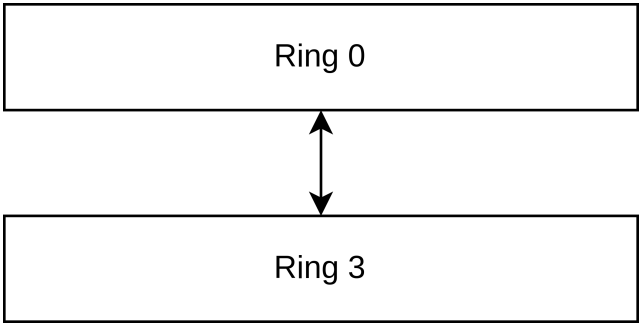
- VMXON
- VMXOFF
- INVEPT
- INVVPID
- VMCLEAR
- VMPTRLD
- VMPTRSTR
- VMLAUNCH/VMRESUME

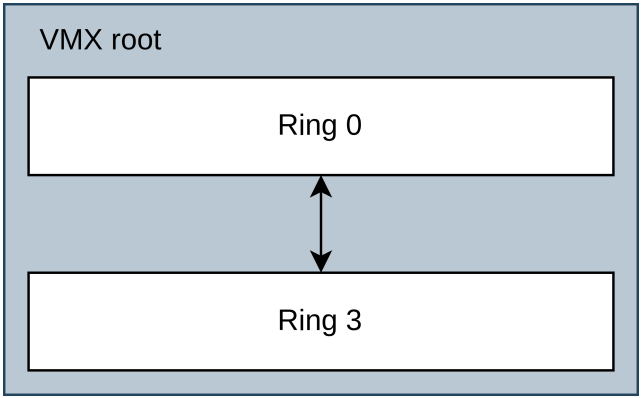
- VMXON
- VMXOFF
- INVEPT
- INVVPID
- VMCLEAR
- VMPTRLD
- VMPTRSTR
- VMLAUNCH/VMRESUME
- VMREAD

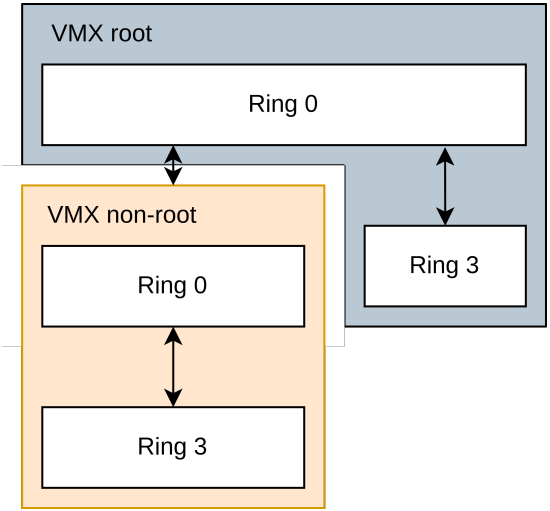
- VMXON
- VMXOFF
- INVEPT
- INVVPID
- VMCLEAR
- VMPTRLD
- VMPTRSTR
- VMLAUNCH/VMRESUME
- VMREAD
- VMWRITE

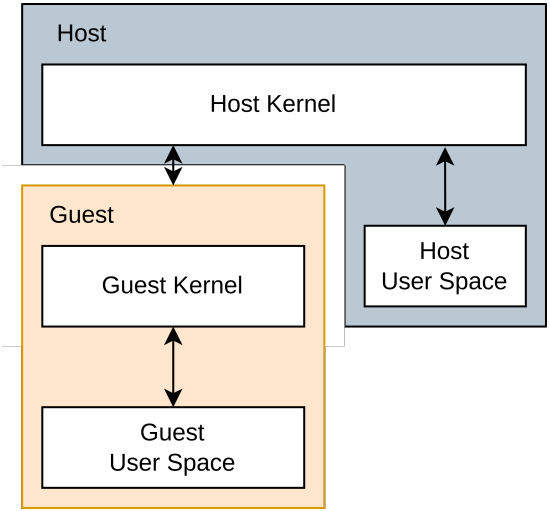
- VMXON
- VMXOFF
- INVEPT
- INVVPID
- VMCLEAR
- VMPTRLD
- VMPTRSTR
- VMLAUNCH/VMRESUME
- VMREAD
- VMWRITE
- VMCALL

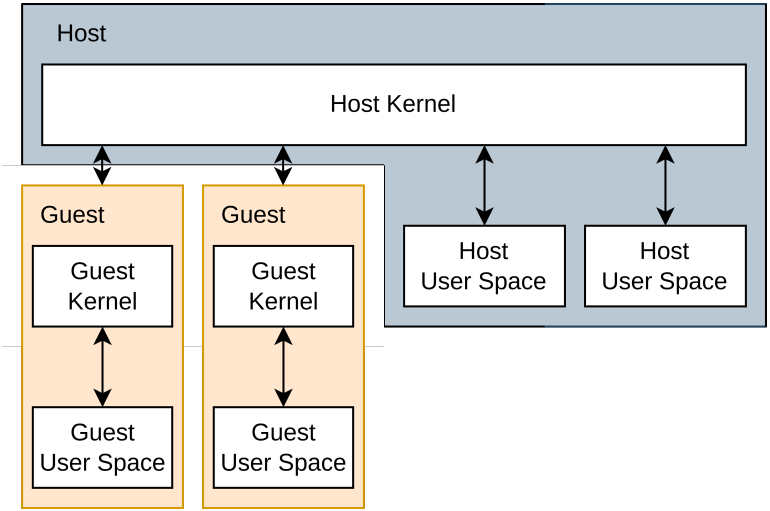

Ring 3
Ring 1&2
Ring 0
Ring -1
Ring -2
Ring -3
Intel ME
UEFI & SMM
Hypervisor
Kernel
Device drivers
Applications

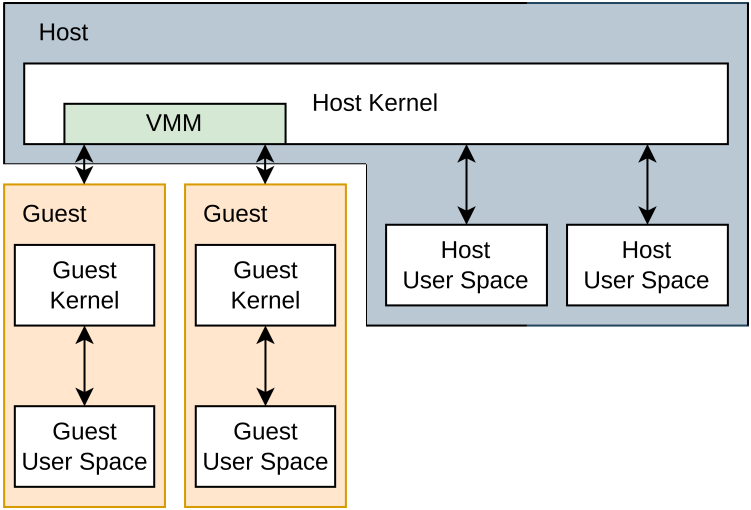

Ring 0
Ring 3

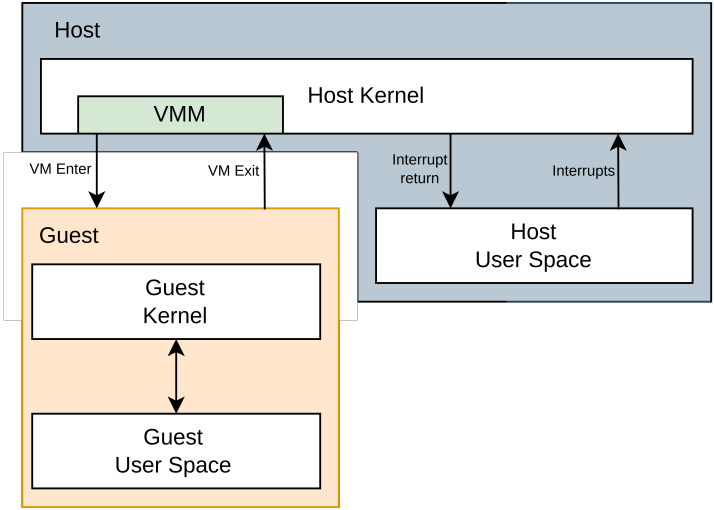

VMX root
Ring 0
Ring 3


VMX root
Ring 0
Ring 3
VMX non-root
Ring 0
Ring 3


Host
Host Kernel
Guest
Guest Kernel
Guest
User Space
Host
User Space


Host
Host Kernel
Guest
Guest Kernel
Guest User Space
Guest
Guest Kernel
Guest User Space
Host User Space
Host User Space


Host
Host Kernel
VMM
Guest
Guest Kernel
Guest User Space
Guest
Guest Kernel
Guest User Space
Host User Space
Host User Space


Host
Host Kernel
VMM
VM Enter
VM Exit
Interrupt return
Interrupts
Host User Space
Guest
Guest Kernel
Guest User Space

VMCS

[2] Intel SDM Volume 3, 25.2

What is a VMCS?

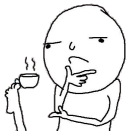
[2] Intel SDM Volume 3, 25.2

What is a VMCS?

Virtual-Machine Control Data Structure (VMCS)

[2] Intel SDM Volume 3, 25.2

What is a VMCS?

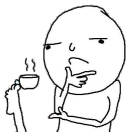


Virtual-Machine Control Data Structure (VMCS)

- Manages VM entries and VM exits

[2] Intel SDM Volume 3, 25.2

What is a VMCS?

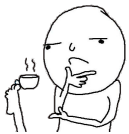


Virtual-Machine Control Data Structure (VMCS)

- Manages VM entries and VM exits
- Used to setup processor behavior in VMX non-root operation

[2] Intel SDM Volume 3, 25.2

What is a VMCS?

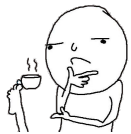


Virtual-Machine Control Data Structure (VMCS)

- Manages VM entries and VM exits
- Used to setup processor behavior in VMX non-root operation
- Can be manipulated using VMCLEAR, VMPTRLD, VMREAD, VMWRITE

[2] Intel SDM Volume 3, 25.2

What is a VMCS?

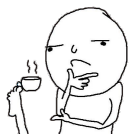


Virtual-Machine Control Data Structure (VMCS)

- Manages VM entries and VM exits
- Used to setup processor behavior in VMX non-root operation
- Can be manipulated using VMCLEAR, VMPTRLD, VMREAD, VMWRITE
- One VMCS per virtual processor

[2] Intel SDM Volume 3, 25.2

What is a VMCS?



Virtual-Machine Control Data Structure (VMCS)

- Manages VM entries and VM exits
- Used to setup processor behavior in VMX non-root operation
- Can be manipulated using VMCLEAR, VMPTRLD, VMREAD, VMWRITE
- One VMCS per virtual processor
- The current VMCS can be accessed using the VMWRITE and VMREAD instructions

[2] Intel SDM Volume 3, 25.2

VMCS Region

[1] Intel SDM Volume 3, 25.2

[1] Intel SDM Volume 3, 25.2

[1] Intel SDM Volume 3, 25.2

- Up to 4KB in size

[1] Intel SDM Volume 3, 25.2

- Up to 4KB in size
- VMCS pointer needs to be a 4KB aligned valid physical address

[1] Intel SDM Volume 3, 25.2

- Up to 4KB in size
- VMCS pointer needs to be a 4KB aligned valid physical address
- Bits 30:0 must contain the VMCS revision identifier (IA32_VMX_BASIC, 0x480)[1]

[1] Intel SDM Volume 3, 25.2

Why?

Why?

Why?

Why?

- Guest configuration

- Guest configuration
- Some registers are not easy to restore

Why?

- Guest configuration
- Some registers are not easy to restore
  - CR4/CR0

Why?

- Guest configuration
- Some registers are not easy to restore
  - CR4/CR0
  - Segment bases and access rights

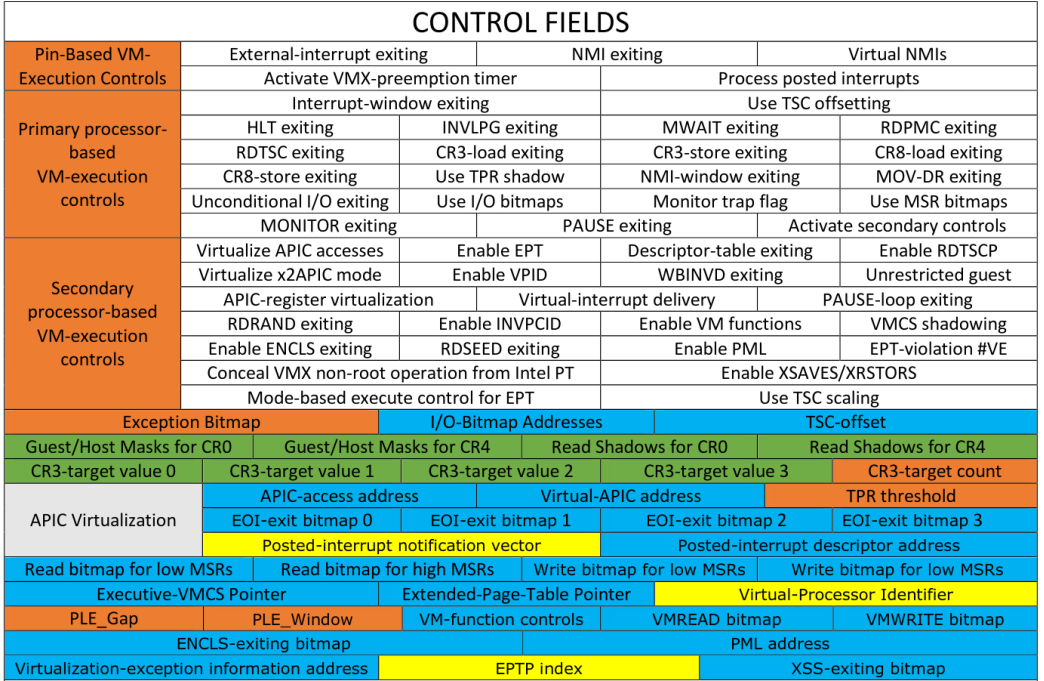
Why?

- Guest configuration
- Some registers are not easy to restore
  - CR4/CR0
  - Segment bases and access rights
  - CR3

Why?

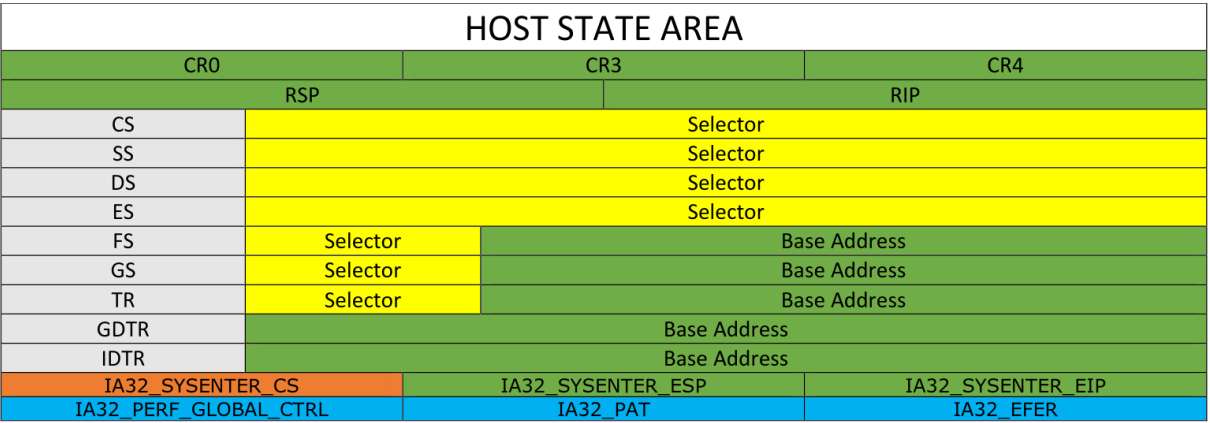
- Guest configuration
- Some registers are not easy to restore
  - CR4/CR0
  - Segment bases and access rights
  - CR3
  - ...

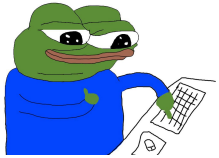
- Natural-Width fields.
- 16-bits fields.
- 32-bits fields.
- 64-bits fields.

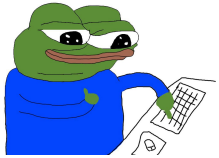

CONTROL FIELDS
Pin-Based VM-Execution Controls
External-interrupt exiting
NMI exiting
Virtual NMIs
Activate VMX-preemption timer
Process posted interrupts
Primary processor-based VM-execution controls
Interrupt-window exiting
Use TSC offsetting
HLT exiting
INVLPG exiting
MWAIT exiting
RDPMC exiting
RDTSC exiting
CR3-load exiting
CR3-store exiting
CR8-load exiting
CR8-store exiting
Use TPR shadow
NMI-window exiting
MOV-DR exiting
Unconditional I/O exiting
Use I/O bitmaps
Monitor trap flag
Use MSR bitmaps
MONITOR exiting
PAUSE exiting
Activate secondary controls
Secondary processor-based VM-execution controls
Virtualize APIC accesses
Enable EPT
Descriptor-table exiting
Enable RDTSCP
Virtualize x2APIC mode
Enable VPID
WBINVD exiting
Unrestricted guest
APIC-register virtualization
Virtual-interrupt delivery
PAUSE-loop exiting
RDRAND exiting
Enable INVPCID
Enable VM functions
VMCS shadowing
Enable ENCLS exiting
RDSEED exiting
Enable PML
EPT-violation #VE
Conceal VMX non-root operation from Intel PT
Enable XSAVES/XRSTORS
Mode-based execute control for EPT
Use TSC scaling
Exception Bitmap
I/O-Bitmap Addresses
TSC-offset
Guest/Host Masks for CR0
Guest/Host Masks for CR4
Read Shadows for CR0
Read Shadows for CR4
CR3-target value 0
CR3-target value 1
CR3-target value 2
CR3-target value 3
CR3-target count
APIC Virtualization
APIC-access address
Virtual-APIC address
TPR threshold
EOI-exit bitmap 0
EOI-exit bitmap 1
EOI-exit bitmap 2
EOI-exit bitmap 3
Posted-interrupt notification vector
Posted-interrupt descriptor address
Read bitmap for low MSRs
Read bitmap for high MSRs
Write bitmap for low MSRs
Write bitmap for low MSRs
Executive-VMCS Pointer
Extended-Page-Table Pointer
Virtual-Processor Identifier
PLE_Gap
PLE_Window
VM-function controls
VMREAD bitmap
VMWRITE bitmap
ENCLS-exiting bitmap
PML address
Virtualization-exception information address
EPTP index
XSS-exiting bitmap

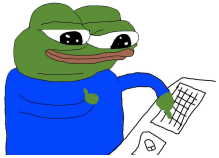
GUEST STATE AREA

CR0	CR3			CR4
DR7				
RSP	RIP			RFLAGS
CS	Selector	Base Address	Segment Limit	Access Right
SS	Selector	Base Address	Segment Limit	Access Right
DS	Selector	Base Address	Segment Limit	Access Right
ES	Selector	Base Address	Segment Limit	Access Right
FS	Selector	Base Address	Segment Limit	Access Right
GS	Selector	Base Address	Segment Limit	Access Right
LDTR	Selector	Base Address	Segment Limit	Access Right
TR	Selector	Base Address	Segment Limit	Access Right
GDTR	[redacted]	Base Address	Limit	[redacted]
IDTR	[redacted]	Base Address	Limit	[redacted]
IA32_DEBUGCTL	IA32_SYSENTER_CS	IA32_SYSENTER_ESP	IA32_SYSENTER_EIP	
IA32_PERF_GLOBAL_CTRL	IA32_PAT	IA32_EFER	IA32_BNDCFGS	
SMBASE				
Activity state	Interruptibility state			
Pending debug exceptions				
VMCS link pointer				
VMX-preemption timer value				
Page-directory-pointer-table entries	PDPTE0	PDPTE1	PDPTE2	PDPTE3
Guest interrupt status				
PML index				

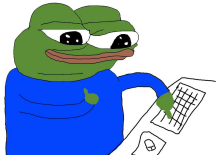

HOST STATE AREA
CR0
CR3
CR4
RSP
RIP
CS
Selector
SS
Selector
DS
Selector
ES
Selector
FS
Selector
Base Address
GS
Selector
Base Address
TR
Selector
Base Address
GDTR
Base Address
IDTR
Base Address
IA32_SYSENTER_CS
IA32_SYSENTER_ESP
IA32_SYSENTER_EIP
IA32_PERF_GLOBAL_CTRL
IA32_PAT
IA32_EFER



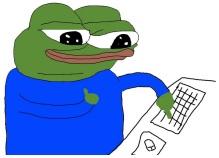




- What about things that are not in the guest state area, like general-purpose registers?



- What about things that are not in the guest state area, like general-purpose registers?
- We have to save and restore them by hand!



- What about things that are not in the guest state area, like general-purpose registers?
- We have to save and restore them by hand!

→ We need a context switch

VMCS Some Important Host Fields

- Host EFER

VMCS Some Important Host Fields

- Host EFER
- Host PAT

VMCS Some Important Host Fields

- Host EFER
- Host PAT
- Host CRx

VMCS Some Important Host Fields

- Host EFER
- Host PAT
- Host CRx
- Host RSP

VMCS Some Important Host Fields

- Host EFER
- Host PAT
- Host CRx
- Host RSP
- Host RIP

VMCS Some Important Host Fields

- Host EFER
- Host PAT
- Host CRx
- Host RSP
- Host RIP
- Host segment selectors

VMCS Some Important Host Fields

- Host EFER
- Host PAT
- Host CRx
- Host RSP
- Host RIP
- Host segment selectors
- TR selector and base address

VMCS Some Important Host Fields

- Host EFER
- Host PAT
- Host CRx
- Host RSP
- Host RIP
- Host segment selectors
- TR selector and base address
- GDTR and IDTR base address

VMCS Some Important Guest Fields

- Guest EFER

VMCS Some Important Guest Fields

- Guest EFER
- Guest PAT

VMCS Some Important Guest Fields

- Guest EFER
- Guest PAT
- Guest CRx

VMCS Some Important Guest Fields

- Guest EFER
- Guest PAT
- Guest CRx
- Guest RSP

VMCS Some Important Guest Fields

- Guest EFER
- Guest PAT
- Guest CRx
- Guest RSP
- Guest RIP

VMCS Some Important Guest Fields

- Guest EFER
- Guest PAT
- Guest CRx
- Guest RSP
- Guest RIP
- Guest RFLAGS

VMCS Some Important Guest Fields

- Guest EFER
- Guest PAT
- Guest CRx
- Guest RSP
- Guest RIP
- Guest RFLAGS
- Guest segment selectors, base addresses, limits, access rights

VMCS Some Important Guest Fields

- Guest EFER
- Guest PAT
- Guest CRx
- Guest RSP
- Guest RIP
- Guest RFLAGS
- Guest segment selectors, base addresses, limits, access rights
- GDTR and IDTR base address, limit

VMCS Some Important Guest Fields

- Guest EFER
- Guest PAT
- Guest CRx
- Guest RSP
- Guest RIP
- Guest RFLAGS
- Guest segment selectors, base addresses, limits, access rights
- GDTR and IDTR base address, limit
- VMCS link pointer (set it to -1)

VMCS Some Important Control Fields

- Pin-Based VM-Execution Controls

VMCS Some Important Control Fields

- Pin-Based VM-Execution Controls
- Primary Processor-Based VM-Execution Controls

VMCS Some Important Control Fields

- Pin-Based VM-Execution Controls
- Primary Processor-Based VM-Execution Controls
- Secondary Processor-Based VM-Execution Controls

VMCS Some Important Control Fields

- Pin-Based VM-Execution Controls
- Primary Processor-Based VM-Execution Controls
- Secondary Processor-Based VM-Execution Controls
- VM entry controls

VMCS Some Important Control Fields

- Pin-Based VM-Execution Controls
- Primary Processor-Based VM-Execution Controls
- Secondary Processor-Based VM-Execution Controls
- VM entry controls
- VM exit controls

VMCS Some Important Control Fields

- Pin-Based VM-Execution Controls
- Primary Processor-Based VM-Execution Controls
- Secondary Processor-Based VM-Execution Controls
- VM entry controls
- VM exit controls
- EPTP

Running VMs

[1] Intel SDM Volume 3, 25.1

[1] Intel SDM Volume 3, 25.1

[1] Intel SDM Volume 3, 25.1

- VMLAUNCH

[1] Intel SDM Volume 3, 25.1

- VMLAUNCH
  - → VMCS not launched yet

[1] Intel SDM Volume 3, 25.1

- VMLAUNCH
  - → VMCS not launched yet
- VMRESUME

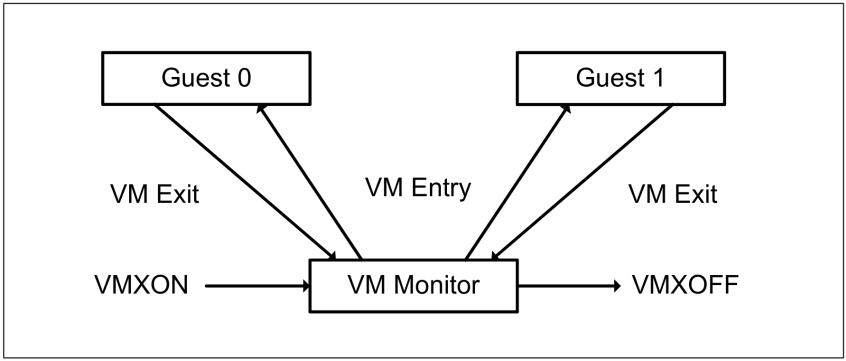
[1] Intel SDM Volume 3, 25.1

- VMLAUNCH
  - → VMCS not launched yet
- VMRESUME
  - → Continue launched VMCS

[1] Intel SDM Volume 3, 25.1

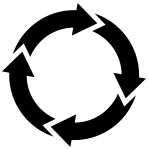
- VMLAUNCH
  - → VMCS not launched yet
- VMRESUME
  - → Continue launched VMCS
- VMCLEAR resets the launch state!

[1] Intel SDM Volume 3, 25.1


Guest 0
Guest 1
VM Exit
VM Entry
VM Exit
VMXON
VM Monitor
VMXOFF

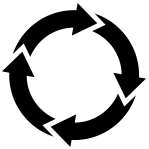
[1] Intel SDM Volume 3, 25.1

VMCS States



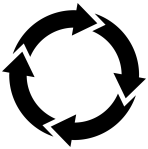
[1] Intel SDM Volume 3, 25.1

VMCS States

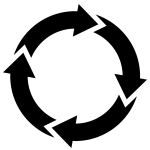


[1] Intel SDM Volume 3, 25.1

- Current

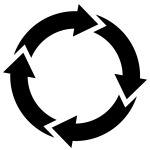


[1] Intel SDM Volume 3, 25.1



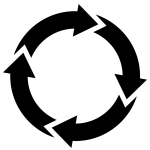
- Current
  - only one VMCS can is current at a time

[1] Intel SDM Volume 3, 25.1



- Current
  - only one VMCS can is current at a time
  - VMWRITE, VMREAD, VMLAUNCH, VMRESUME operate on the current VMCS

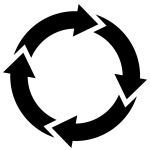
[1] Intel SDM Volume 3, 25.1



- Current
  - only one VMCS can is current at a time
  - VMWRITE, VMREAD, VMLAUNCH, VMRESUME operate on the current VMCS
- Active

[1] Intel SDM Volume 3, 25.1

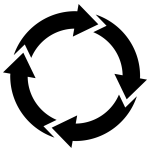
VMCS States



- Current
  - only one VMCS can is current at a time
  - VMWRITE, VMREAD, VMLAUNCH, VMRESUME opperate on the current VMCS
- Active
  - VMCS is managed by the CPU

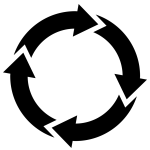
[1] Intel SDM Volume 3, 25.1

VMCS States



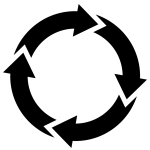
- Current
  - only one VMCS can is current at a time
  - VMWRITE, VMREAD, VMLAUNCH, VMRESUME opperate on the current VMCS
- Active
  - VMCS is managed by the CPU
  - multiple VMCS can be active at a time

[1] Intel SDM Volume 3, 25.1



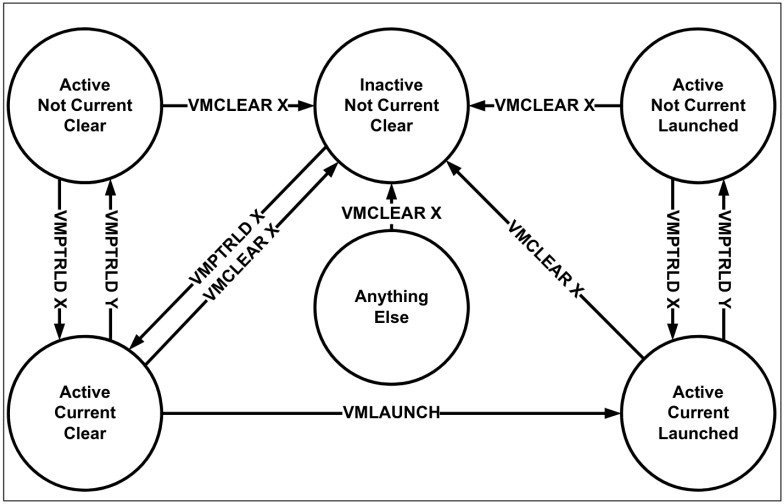
- Current
  - only one VMCS can is current at a time
  - VMWRITE, VMREAD, VMLAUNCH, VMRESUME opperate on the current VMCS
- Active
  - VMCS is managed by the CPU
  - multiple VMCS can be active at a time
- Launched

[1] Intel SDM Volume 3, 25.1



- Current
  - only one VMCS can is current at a time
  - VMWRITE, VMREAD, VMLAUNCH, VMRESUME opperate on the current VMCS
- Active
  - VMCS is managed by the CPU
  - multiple VMCS can be active at a time
- Launched
  - VMCS is launched and can only be continued with VMRESUME

[1] Intel SDM Volume 3, 25.1


Active
Not Current
Clear
VMCLEAR X
Inactive
Not Current
Clear
VMCLEAR X
Active
Not Current
Launched
VMPTRLD X
VMPTRLD Y
VMPTRLD X
VMCLEAR X
VMCLEAR X
Anything
Else
VMCLEAR X
VMPTRLD X
VMPTRLD Y
Active
Current
Clear
VMLAUNCH
Active
Current
Launched

[1] Intel SDM Volume 3, 25.1

[1] Intel SDM Volume 3, 25.1

[1] Intel SDM Volume 3, 25.1

- VMCLEAR

[1] Intel SDM Volume 3, 25.1

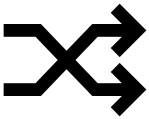
Changing VMCS States

- VMCLEAR
  - initializes a new VMCS

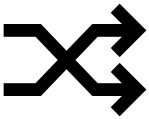
[1] Intel SDM Volume 3, 25.1

Changing VMCS States

- VMCLEAR
  - initializes a new VMCS
  - sets active VMCS to inactive, not current, and not launched

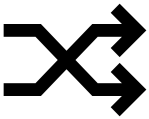


[1] Intel SDM Volume 3, 25.1



- VMCLEAR
  - initializes a new VMCS
  - sets active VMCS to inactive, not current, and not launched
- VMPTRLD

[1] Intel SDM Volume 3, 25.1



- VMCLEAR
  - initializes a new VMCS
  - sets active VMCS to inactive, not current, and not launched
- VMPTRLD
  - sets VMCS to active and current

[1] Intel SDM Volume 3, 25.1

- VMCLEAR
  - initializes a new VMCS
  - sets active VMCS to inactive, not current, and not launched
- VMPTRLD
  - sets VMCS to active and current
  - sets previous current VMCS to not current

[1] Intel SDM Volume 3, 25.1

- VMCLEAR
  - initializes a new VMCS
  - sets active VMCS to inactive, not current, and not launched
- VMPTRLD
  - sets VMCS to active and current
  - sets previous current VMCS to not current
- VMLAUNCH

[1] Intel SDM Volume 3, 25.1

- VMCLEAR
  - initializes a new VMCS
  - sets active VMCS to inactive, not current, and not launched
- VMPTRLD
  - sets VMCS to active and current
  - sets previous current VMCS to not current
- VMLAUNCH
  - sets VMCS to launched

[1] Intel SDM Volume 3, 25.1

Changing VMCS States

- VMCLEAR
  - initializes a new VMCS
  - sets active VMCS to inactive, not current, and not launched
- VMPTRLD
  - sets VMCS to active and current
  - sets previous current VMCS to not current
- VMLAUNCH
  - sets VMCS to launched
- VMRESMUE

[1] Intel SDM Volume 3, 25.1

Changing VMCS States

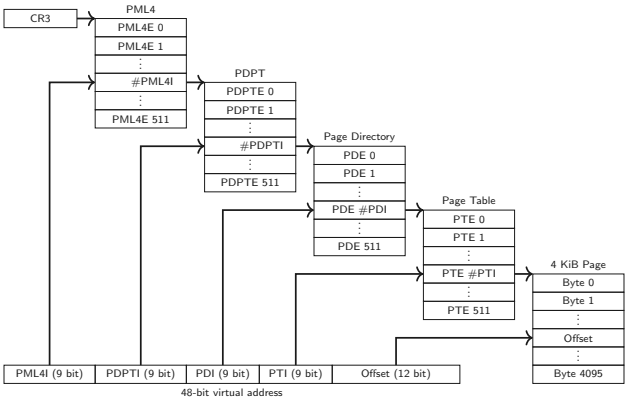
- VMCLEAR
  - initializes a new VMCS
  - sets active VMCS to inactive, not current, and not launched
- VMPTRLD
  - sets VMCS to active and current
  - sets previous current VMCS to not current
- VMLAUNCH
  - sets VMCS to launched
- VMRESMUE
  - can only be used on launched VMCS

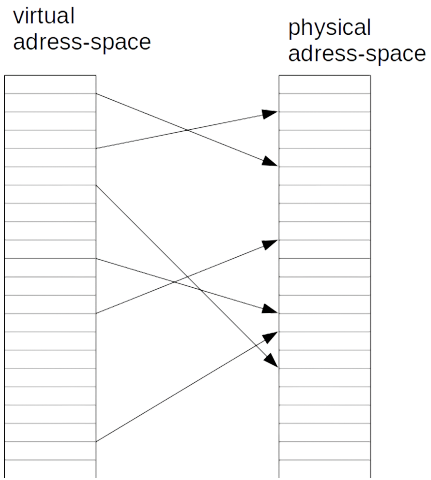
[1] Intel SDM Volume 3, 25.1

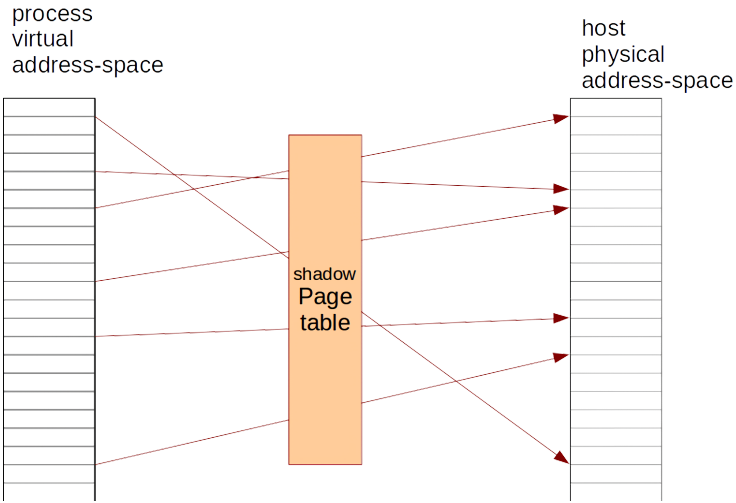
EPT

How can we give a guest access to a physical address space without giving it access to the hosts physical address space?

Virtual Address Translation




virtual
adress-space
physical
adress-space


process
virtual
address-space
shadow
Page
table
host
physical
address-space

- guest runs on a normal page table

Shadow Page Table

- guest runs on a normal page table
- guest can not modify their real page table

Shadow Page Table

- guest runs on a normal page table
- guest can not modify their real page table
- instead the guest has a copy of the real page table

Shadow Page Table

- guest runs on a normal page table
- guest can not modify their real page table
- instead the guest has a copy of the real page table
- when the guest changes page table copy

Shadow Page Table

- guest runs on a normal page table
- guest can not modify their real page table
- instead the guest has a copy of the real page table
- when the guest changes page table copy
  - Hypervisor has to catch access

Shadow Page Table

- guest runs on a normal page table
- guest can not modify their real page table
- instead the guest has a copy of the real page table
- when the guest changes page table copy
  - Hypervisor has to catch access
  - update shadow (real) page table

- guest runs on a normal page table
- guest can not modify their real page table
- instead the guest has a copy of the real page table
- when the guest changes page table copy
  - Hypervisor has to catch access
  - update shadow (real) page table

→ guest can manage virtual memory without access to arbitrary physical memory or real PPNs

Shadow Page Table

Shadow Page Table

Shadow Page Table

Advantages:

Advantages:

+ address translations are fast

Advantages:

+ address translations are fast
+ requires only minimal hardware support

Advantages:

+ address translations are fast
+ requires only minimal hardware support

Disadvantages:

Shadow Page Table

Advantages:

+ address translations are fast
+ requires only minimal hardware support

Disadvantages:

- page table modifications are slow

Shadow Page Table

Advantages:

+ address translations are fast
+ requires only minimal hardware support

Disadvantages:

- page table modifications are slow
- only works with paging enabled

Shadow Page Table

Advantages:

+ address translations are fast
+ requires only minimal hardware support

Disadvantages:

- page table modifications are slow
- only works with paging enabled
- x2 memory overhead for page tables in the guest

Shadow Page Table

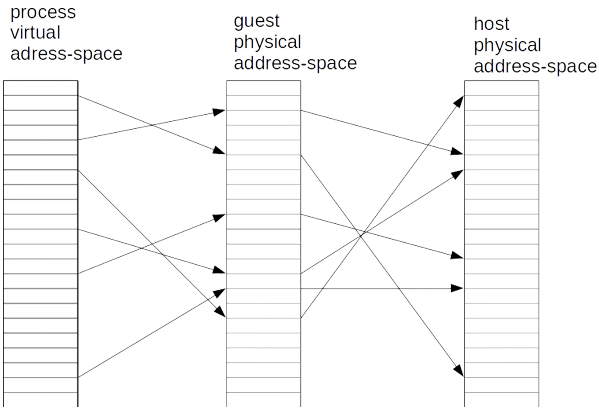
Advantages:

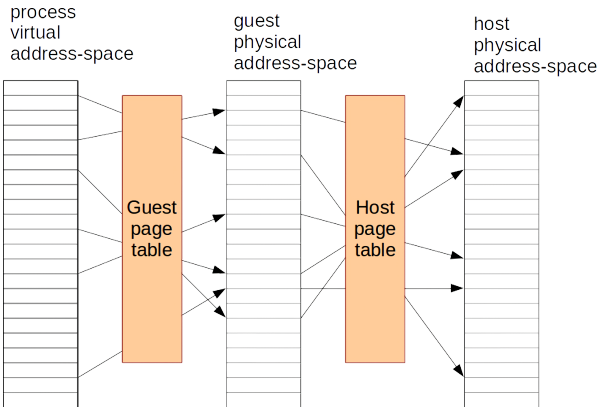
+ address translations are fast
+ requires only minimal hardware support

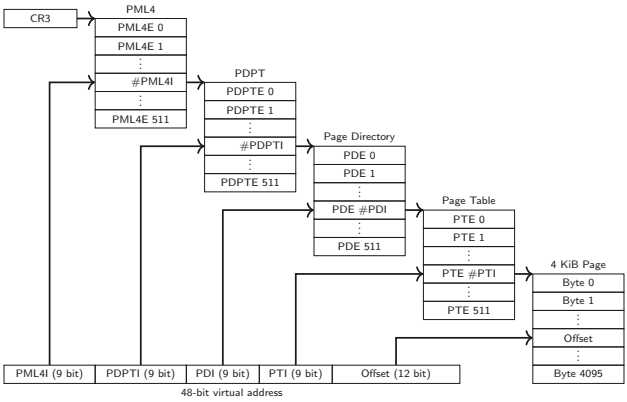
Disadvantages:

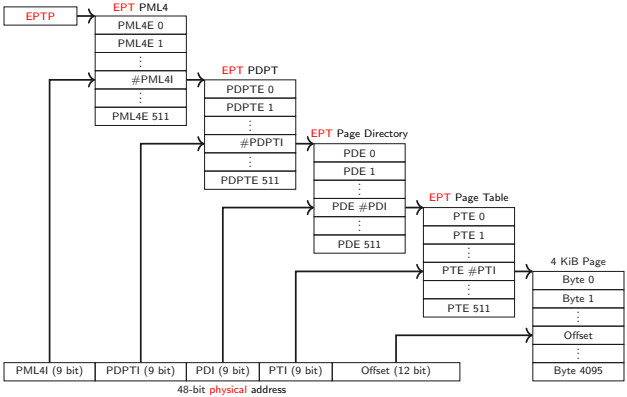
- page table modifications are slow
- only works with paging enabled
- x2 memory overhead for page tables in the guest
- complicated to implement


WE NEED TO GO DEEPER

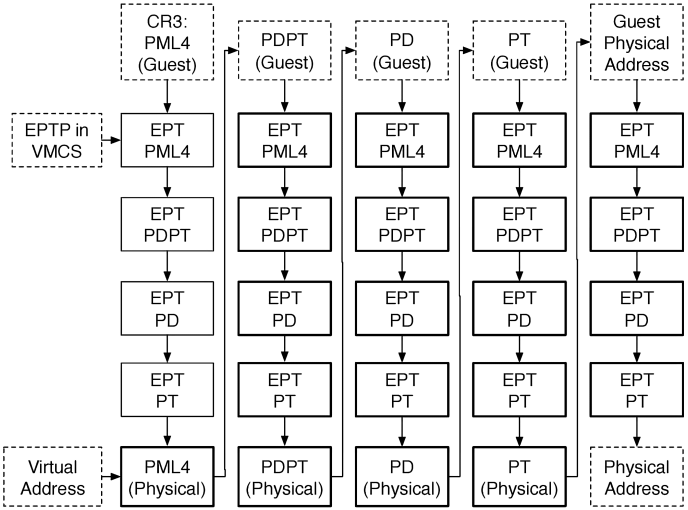

process
virtual
adress-space
guest
physical
address-space
host
physical
address-space


process
virtual
address-space
guest
physical
address-space
host
physical
address-space
Guest
page
table
Host
page
table

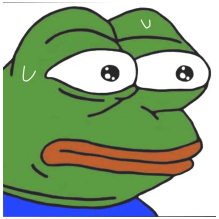

CR3
PML4
PML4E 0
PML4E 1
#PML4I
PML4E 511
PDPT
PDPTE 0
PDPTE 1
#PDPTI
PDPTE 511
Page Directory
PDE 0
PDE 1
PDE #PDI
PDE 511
Page Table
PTE 0
PTE 1
PTE #PTI
PTE 511
4 KiB Page
Byte 0
Byte 1
Offset
Byte 4095
PML4I (9 bit)
PDPTI (9 bit)
PDI (9 bit)
PTI (9 bit)
Offset (12 bit)
48-bit virtual address


EPTP
EPT PML4
PML4E 0
PML4E 1
#PML4I
PML4E 511
EPT PDPT
PDPTE 0
PDPTE 1
#PDPTI
PDPTE 511
EPT Page Directory
PDE 0
PDE 1
PDE #PDI
PDE 511
EPT Page Table
PTE 0
PTE 1
PTE #PTI
PTE 511
4 KiB Page
Byte 0
Byte 1
Offset
Byte 4095
PML4I (9 bit)
PDPTI (9 bit)
PDI (9 bit)
PTI (9 bit)
Offset (12 bit)
48-bit physical address

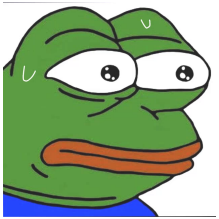
Isn't this slow?



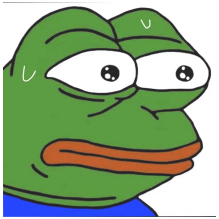
Caching saves us yet again



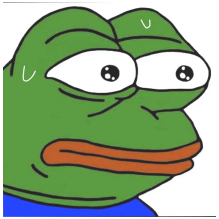
Caching saves us yet again



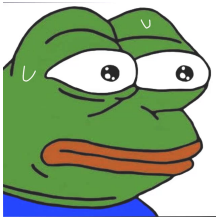
Caching saves us yet again



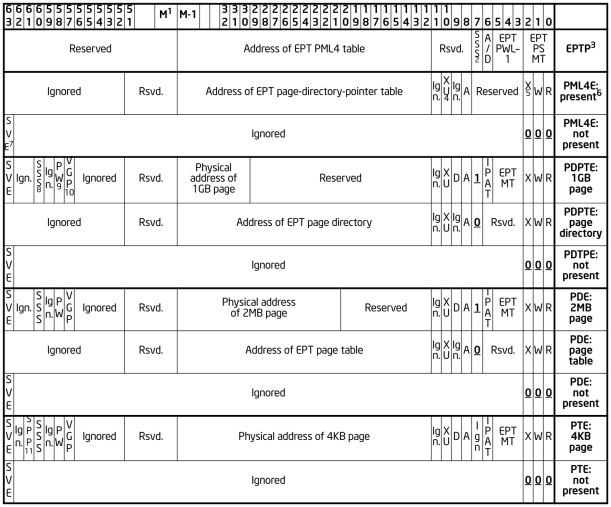
- EPT translations are cached in the TLB



- EPT translations are cached in the TLB
- Translations are tagged with the EPTP



- EPT translations are cached in the TLB
- Translations are tagged with the EPTP
- We can invalidate all TLB entries for a given EPTP or all EPT TLB entries using the INVEPT instruction


63 62 61 60 59 58 57 56 55 54 53 52 51 M[1] M-1 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Reserved
Address of EPT PML4 table
Rsvd.
SSS2
A/D
EPT PWL−1
EPT PS MT
EPTP[3]
Ignored
Rsvd.
Address of EPT page-directory-pointer table
Ign.
XU
Ign.
A
Reserved
XS
W
R
PML4E: present[6]
SVE[7]
Ignored
0 0 0
PML4E: not present
SVE
Ign.
SSS
Ign.
PWL
VGP
Ignored
Rsvd.
Physical address of 1GB page
Reserved
Ign.
XU
D
A
1
PAT
EPT MT
X W R
PDPTE: 1GB page
Ignored
Rsvd.
Address of EPT page directory
Ign.
XU
Ign.
A
0
Rsvd.
X W R
PDPTE: page directory
SVE
Ignored
0 0 0
PDTPE: not present
SVE
Ign.
SSS
Ign.
PWL
VGP
Ignored
Rsvd.
Physical address of 2MB page
Reserved
Ign.
XU
D
A
1
PAT
EPT MT
X W R
PDE: 2MB page
Ignored
Rsvd.
Address of EPT page table
Ign.
XU
Ign.
A
0
Rsvd.
X W R
PDE: page table
SVE
Ignored
0 0 0
PDE: not present
SVE
Ign.
SPP[11]
SSS
Ign.
PWL
VGP
Ignored
Rsvd.
Physical address of 4KB page
Ign.
XU
D
A
Ign
PAT
EPT MT
X W R
PTE: 4KB page
SVE
Ignored
0 0 0
PTE: not present

VM Exit

[1] Intel SDM Volume 3, 28.1

[1] Intel SDM Volume 3, 28.1

[1] Intel SDM Volume 3, 28.1

- An event that causes a VM exit directly **does not** update the architectural state

[1] Intel SDM Volume 3, 28.1

- An event that causes a VM exit directly **does not** update the architectural state
- An event that causes a VM exit indirectly **does** update the architectural state (example: PF causes a write to CR2 and CR2 exiting is enabled)

[1] Intel SDM Volume 3, 28.1

- An event that causes a VM exit directly **does not** update the architectural state
- An event that causes a VM exit indirectly **does** update the architectural state (example: PF causes a write to CR2 and CR2 exiting is enabled)
- If a VM exit is triggered by executing specified instructions we can gain further information

[1] Intel SDM Volume 3, 28.1

- An event that causes a VM exit directly **does not** update the architectural state
- An event that causes a VM exit indirectly **does** update the architectural state (example: PF causes a write to CR2 and CR2 exiting is enabled)
- If a VM exit is triggered by executing specified instructions we can gain further information
  - VM exit qualification → VM-exit reason specific information

[1] Intel SDM Volume 3, 28.1

- An event that causes a VM exit directly **does not** update the architectural state
- An event that causes a VM exit indirectly **does** update the architectural state (example: PF causes a write to CR2 and CR2 exiting is enabled)
- If a VM exit is triggered by executing specified instructions we can gain further information
  - VM exit qualification → VM-exit reason specific information
  - VM exit instruction length → length of the instruction that caused the VM exit

[1] Intel SDM Volume 3, 28.1

- An event that causes a VM exit directly **does not** update the architectural state
- An event that causes a VM exit indirectly **does** update the architectural state (example: PF causes a write to CR2 and CR2 exiting is enabled)
- If a VM exit is triggered by executing specified instructions we can gain further information
  - VM exit qualification → VM-exit reason specific information
  - VM exit instruction length → length of the instruction that caused the VM exit
  - VM exit instruction information → detailed information on the instruction that caused the VM exit

[1] Intel SDM Volume 3, 28.1

VM Exit

- An event that causes a VM exit directly **does not** update the architectural state
- An event that causes a VM exit indirectly **does** update the architectural state (example: PF causes a write to CR2 and CR2 exiting is enabled)
- If a VM exit is triggered by executing specified instructions we can gain further information
  - VM exit qualification → VM-exit reason specific information
  - VM exit instruction length → length of the instruction that caused the VM exit
  - VM exit instruction information → detailed information on the instruction that caused the VM exit
- Host RFLAGS is cleared (except bit 1)

[1] Intel SDM Volume 3, 28.1

Table 27-5. Exit Qualification for I/O Instructions

Bit Position(s)	Contents
2:0	Size of access: 0 = 1-byte 1 = 2-byte 3 = 4-byte Other values not used
3	Direction of the attempted access (0 = OUT, 1 = IN)
4	String instruction (0 = not string; 1 = string)
5	REP prefixed (0 = not REP; 1 = REP)
6	Operand encoding (0 = DX, 1 = immediate)
15:7	Not currently defined
31:16	Port number (as specified in DX or in an immediate operand)
63:32	Not currently defined. These bits exist only on processors that support Intel 64 architecture.

- Instructions that cause VM exits unconditionaly

- Instructions that cause VM exits unconditionaly
  - CPUID

- Instructions that cause VM exits unconditionaly
  - CPUID
  - GETSEC

- Instructions that cause VM exits unconditionaly
  - CPUID
  - GETSEC
  - INVD

- Instructions that cause VM exits unconditionaly
  - CPUID
  - GETSEC
  - INVD
  - XSETBV

VM Exit Reasons

- Instructions that cause VM exits unconditionaly
  - CPUID
  - GETSEC
  - INVD
  - XSETBV
  - VMX instructions (excluding VMFUNC)

- Instructions that cause VM exits unconditionaly
  - CPUID
  - GETSEC
  - INVD
  - XSETBV
  - VMX instructions (excluding VMFUNC)
- Instructions that cause VM exits conditionaly

VM Exit Reasons

- Instructions that cause VM exits unconditionaly
  - CPUID
  - GETSEC
  - INVD
  - XSETBV
  - VMX instructions (excluding VMFUNC)
- Instructions that cause VM exits conditionaly
  - RDMSR/WRMSR

VM Exit Reasons

- Instructions that cause VM exits unconditionaly
  - CPUID
  - GETSEC
  - INVD
  - XSETBV
  - VMX instructions (excluding VMFUNC)
- Instructions that cause VM exits conditionaly
  - RDMSR/WRMSR
  - IN/OUT

VM Exit Reasons

- Instructions that cause VM exits unconditionaly
  - CPUID
  - GETSEC
  - INVD
  - XSETBV
  - VMX instructions (excluding VMFUNC)
- Instructions that cause VM exits conditionaly
  - RDMSR/WRMSR
  - IN/OUT
  - MOV to/from CRx

VM Exit Reasons

- Instructions that cause VM exits unconditionaly
  - CPUID
  - GETSEC
  - INVD
  - XSETBV
  - VMX instructions (excluding VMFUNC)
- Instructions that cause VM exits conditionaly
  - RDMSR/WRMSR
  - IN/OUT
  - MOV to/from CRx
  - PAUSE

VM Exit Reasons

- Instructions that cause VM exits unconditionaly
  - CPUID
  - GETSEC
  - INVD
  - XSETBV
  - VMX instructions (excluding VMFUNC)
- Instructions that cause VM exits conditionaly
  - RDMSR/WRMSR
  - IN/OUT
  - MOV to/from CRx
  - PAUSE
  - ...

VM Exit Reasons

- Instructions that cause VM exits unconditionaly
  - CPUID
  - GETSEC
  - INVD
  - XSETBV
  - VMX instructions (excluding VMFUNC)
- Instructions that cause VM exits conditionaly
  - RDMSR/WRMSR
  - IN/OUT
  - MOV to/from CRx
  - PAUSE
  - ...
- Exceptions

VM Exit Reasons

- Instructions that cause VM exits unconditionaly
  - CPUID
  - GETSEC
  - INVD
  - XSETBV
  - VMX instructions (excluding VMFUNC)
- Instructions that cause VM exits conditionaly
  - RDMSR/WRMSR
  - IN/OUT
  - MOV to/from CRx
  - PAUSE
  - ...
- Exceptions
- VMX-preemption timer

VM Exit Reasons

- Instructions that cause VM exits unconditionaly
  - CPUID
  - GETSEC
  - INVD
  - XSETBV
  - VMX instructions (excluding VMFUNC)
- Instructions that cause VM exits conditionaly
  - RDMSR/WRMSR
  - IN/OUT
  - MOV to/from CRx
  - PAUSE
  - ...
- Exceptions
- VMX-preemption timer
- NMIs

VM Exit Reasons

- Instructions that cause VM exits unconditionaly
  - CPUID
  - GETSEC
  - INVD
  - XSETBV
  - VMX instructions (excluding VMFUNC)
- Instructions that cause VM exits conditionaly
  - RDMSR/WRMSR
  - IN/OUT
  - MOV to/from CRx
  - PAUSE
  - ...
- Exceptions
- VMX-preemption timer
- NMIs
- ...

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 cpuid
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	<rax>
rbx	<rbx>
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830478
rip	main+0

```
main:
  push   rbx
  xor    eax, eax
  sub    rsp, 0x10
  cpuid
  mov    rdi, rsp
  mov    DWORD PTR [rsp], ebx
  mov    DWORD PTR [rsp+0x4], ecx
  mov    DWORD PTR [rsp+0x8], edx
  mov    DWORD PTR [rsp+0x10], 0x0
  call   401030 <puts@plt>
  add    rsp, 0x10
  xor    eax, eax
  pop    rbx
  ret
```

Registers

rax	<rax>
rbx	<rbx>
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830478
rip	main+0

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 cpuid
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	<rax>
rbx	<rbx>
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830470
rip	main+1

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 cpuid
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0x0
rbx	<rbx>
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830470
rip	main+3

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 cpuid
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0x0
rbx	<rbx>
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+7

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 cpuid
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0xd
rbx	0x756e6547
rcx	0x6c65746e
rdx	0x49656e69
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+9

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 cpuid
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0xd
rbx	"Genu"
rcx	"ntel"
rdx	"inel"
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+9

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 cpuid
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0xd
rbx	"Genu"
rcx	"ntel"
rdx	"inel"
rdi	0x7ffff7830460
rsp	0x7ffff7830460
rip	main+12

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 cpuid
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0xd
rbx	"Genu"
rcx	"ntel"
rdx	"inel"
rdi	0x7ffff7830460
rsp	0x7ffff7830460
rip	main+16

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 cpuid
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0xd
rbx	"Genu"
rcx	"ntel"
rdx	"inel"
rdi	0x7ffff7830460
rsp	0x7ffff7830460
rip	main+21

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 cpuid
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0xd
rbx	"Genu"
rcx	"ntel"
rdx	"inel"
rdi	0x7ffff7830460
rsp	0x7ffff7830460
rip	main+26

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 cpuid
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0xd
rbx	"Genu"
rcx	"ntel"
rdx	"inel"
rdi	0x7ffff7830460
rsp	0x7ffff7830460
rip	main+34

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 cpuid
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0xc
rbx	"Genu"
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+39

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 cpuid
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0xc
rbx	"Genu"
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830470
rip	main+43

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 cpuid
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0x0
rbx	"Genu"
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830470
rip	main+45

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 cpuid
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0x0
rbx	<rbx>
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+46

CPUID Hypervisor

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 cpuid
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	<rax>
rbx	<rbx>
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830478
rip	main+0

CPUID Hypervisor

```
main:
  push   rbx
  xor    eax, eax
  sub    rsp, 0x10
  cpuid
  mov    rdi, rsp
  mov    DWORD PTR [rsp], ebx
  mov    DWORD PTR [rsp+0x4], ecx
  mov    DWORD PTR [rsp+0x8], edx
  mov    DWORD PTR [rsp+0x10], 0x0
  call   401030 <puts@plt>
  add    rsp, 0x10
  xor    eax, eax
  pop    rbx
  ret
```

Registers

rax	<rax>
rbx	<rbx>
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830478
rip	main+0

CPUID Hypervisor

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 cpuid
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	<rax>
rbx	<rbx>
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830470
rip	main+1

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 cpuid
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0x0
rbx	<rbx>
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830470
rip	main+3

CPUID Hypervisor

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 cpuid
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0x0
rbx	<rbx>
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+7

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 cpuid
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0x0
rbx	<rbx>
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+7

1. VM Exit → VMM takes control

Registers

rax	0x0
rbx	<rbx>
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+7

1. VM Exit → VMM takes control
2. Check VM Exit reason → CPUID

Registers

rax	0x0
rbx	<rbx>
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+7

1. VM Exit → VMM takes control
2. Check VM Exit reason → CPUID
3. Emulate CPUID

Registers

rax	0x0
rbx	<rbx>
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+7

1. VM Exit → VMM takes control
2. Check VM Exit reason → CPUID
3. Emulate CPUID
   3.1 Check rax → vendor string (rax = 0)

Registers

rax	0x0
rbx	<rbx>
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+7

1. VM Exit → VMM takes control
2. Check VM Exit reason → CPUID
3. Emulate CPUID
   - 3.1 Check rax → vendor string (rax = 0)
   - 3.2 Set rbx, rcx, rdx to the vendor string

Registers

rax	0x0
rbx	<rbx>
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+7

1. VM Exit → VMM takes control
2. Check VM Exit reason → CPUID
3. Emulate CPUID
   - 3.1 Check rax → vendor string (rax = 0)
   - 3.2 Set rbx, rcx, rdx to the vendor string

Registers

rax	0x0
rbx	"SWEB"
rcx	"SWEB"
rdx	"SWEB"
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+7

1. VM Exit → VMM takes control
2. Check VM Exit reason → CPUID
3. Emulate CPUID
   - 3.1 Check rax → vendor string (rax = 0)
   - 3.2 Set rbx, rcx, rdx to the vendor string
   - 3.3 Set rax to the maximum input value

Registers

rax	0x0
rbx	"SWEB"
rcx	"SWEB"
rdx	"SWEB"
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+7

1. VM Exit → VMM takes control
2. Check VM Exit reason → CPUID
3. Emulate CPUID
   - 3.1 Check rax → vendor string (rax = 0)
   - 3.2 Set rbx, rcx, rdx to the vendor string
   - 3.3 Set rax to the maximum input value

Registers

rax	0xd
rbx	"SWEB"
rcx	"SWEB"
rdx	"SWEB"
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+7

1. VM Exit → VMM takes control
2. Check VM Exit reason → CPUID
3. Emulate CPUID
   - 3.1 Check rax → vendor string (rax = 0)
   - 3.2 Set rbx, rcx, rdx to the vendor string
   - 3.3 Set rax to the maximum input value
4. VM Resume

Registers

rax	0xd
rbx	"SWEB"
rcx	"SWEB"
rdx	"SWEB"
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+7

CPUID Hypervisor

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 cpuid
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0xd
rbx	"SWEB"
rcx	"SWEB"
rdx	"SWEB"
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+7

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 cpuid
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0xd
rbx	"SWEB"
rcx	"SWEB"
rdx	"SWEB"
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+7

1. VM Exit → VMM takes control
2. Check VM Exit reason → CPUID
3. Emulate CPUID
   - 3.1 Check rax → vendor string (rax = 0)
   - 3.2 Set rbx, rcx, rdx to the vendor string
   - 3.3 Set rax to the maximum input value
   - 3.4 Advance rip by the instruction length (field VM-exit instruction length)
4. VM Resume

Registers

rax	0xd
rbx	"SWEB"
rcx	"SWEB"
rdx	"SWEB"
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+7

1. VM Exit → VMM takes control
2. Check VM Exit reason → CPUID
3. Emulate CPUID
   - 3.1 Check rax → vendor string (rax = 0)
   - 3.2 Set rbx, rcx, rdx to the vendor string
   - 3.3 Set rax to the maximum input value
   - 3.4 Advance rip by the instruction length (field VM-exit instruction length)
4. VM Resume

Registers

rax	0xd
rbx	"SWEB"
rcx	"SWEB"
rdx	"SWEB"
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+9

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 cpuid
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0xd
rbx	"SWEB"
rcx	"SWEB"
rdx	"SWEB"
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+9

CPUID Hypervisor

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 cpuid
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0xd
rbx	"SWEB"
rcx	"SWEB"
rdx	"SWEB"
rdi	0x7ffff7830460
rsp	0x7ffff7830460
rip	main+12

CPUID Hypervisor

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 cpuid
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0xd
rbx	"SWEB"
rcx	"SWEB"
rdx	"SWEB"
rdi	0x7ffff7830460
rsp	0x7ffff7830460
rip	main+16

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 cpuid
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0xd
rbx	"SWEB"
rcx	"SWEB"
rdx	"SWEB"
rdi	0x7ffff7830460
rsp	0x7ffff7830460
rip	main+21

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 cpuid
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0xd
rbx	"SWEB"
rcx	"SWEB"
rdx	"SWEB"
rdi	0x7ffff7830460
rsp	0x7ffff7830460
rip	main+26

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 cpuid
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0xd
rbx	"SWEB"
rcx	"SWEB"
rdx	"SWEB"
rdi	0x7ffff7830460
rsp	0x7ffff7830460
rip	main+34

CPUID Hypervisor

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 cpuid
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0xc
rbx	"SWEB"
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+39

CPUID Hypervisor

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 cpuid
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0xc
rbx	"SWEB"
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830470
rip	main+43

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 cpuid
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0x0
rbx	"SWEB"
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830470
rip	main+45

CPUID Hypervisor

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 cpuid
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0x0
rbx	<rbx>
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+46


[redacted]
VMM
GUEST
I REQUEST YOUR ASSISTANCE


[redacted]
VMM
GUEST
I REQUEST YOUR ASSISTANCE

- The only purpose of this instruction is to trigger a VM exit

- The only purpose of this instruction is to trigger a VM exit
- Can be used by the guest to request hypervisor assistance

- The only purpose of this instruction is to trigger a VM exit
- Can be used by the guest to request hypervisor assistance
- Calling convention is hypervisor specific

Emulate CPUID with a Hypercall

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 vmcall
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	<rax>
rbx	<rbx>
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830478
rip	main+0

Emulate CPUID with a Hypercall

```
main:
  push   rbx
  xor    eax, eax
  sub    rsp, 0x10
  vmcall
  mov    rdi, rsp
  mov    DWORD PTR [rsp], ebx
  mov    DWORD PTR [rsp+0x4], ecx
  mov    DWORD PTR [rsp+0x8], edx
  mov    DWORD PTR [rsp+0x10], 0x0
  call   401030 <puts@plt>
  add    rsp, 0x10
  xor    eax, eax
  pop    rbx
  ret
```

Registers

rax	<rax>
rbx	<rbx>
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830478
rip	main+0

Emulate CPUID with a Hypercall

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 vmcall
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	<rax>
rbx	<rbx>
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830470
rip	main+1

Emulate CPUID with a Hypercall

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 vmcall
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0x0
rbx	<rbx>
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830470
rip	main+3

Emulate CPUID with a Hypercall

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 vmcall
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0x0
rbx	<rbx>
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+7

Emulate CPUID with a Hypercall

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 vmcall
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0x0
rbx	<rbx>
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+7

1. VM Exit → VMM takes control

Registers

rax	0x0
rbx	<rbx>
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+7

Emulate CPUID with a Hypercall

1. VM Exit → VMM takes control
2. Check VM Exit reason → VM call

Registers

rax	0x0
rbx	<rbx>
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+7

Emulate CPUID with a Hypercall

1. VM Exit → VMM takes control
2. Check VM Exit reason → VM call
3. Handle VM call

Registers

rax	0x0
rbx	<rbx>
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+7

Emulate CPUID with a Hypercall

1. VM Exit → VMM takes control
2. Check VM Exit reason → VM call
3. Handle VM call
   3.1 Check rax → vendor string (rax = 0)

Registers

rax	0x0
rbx	<rbx>
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+7

Emulate CPUID with a Hypercall

1. VM Exit → VMM takes control
2. Check VM Exit reason → VM call
3. Handle VM call
   - 3.1 Check rax → vendor string (rax = 0)
   - 3.2 Set rbx, rcx, rdx to the vendor string

Registers

rax	0x0
rbx	<rbx>
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+7

Emulate CPUID with a Hypercall

1. VM Exit → VMM takes control
2. Check VM Exit reason → VM call
3. Handle VM call
   - 3.1 Check rax → vendor string (rax = 0)
   - 3.2 Set rbx, rcx, rdx to the vendor string

Registers

rax	0x0
rbx	"SWEB"
rcx	"SWEB"
rdx	"SWEB"
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+7

Emulate CPUID with a Hypercall

1. VM Exit → VMM takes control
2. Check VM Exit reason → VM call
3. Handle VM call
   - 3.1 Check rax → vendor string (rax = 0)
   - 3.2 Set rbx, rcx, rdx to the vendor string
   - 3.3 Set rax to the maximum input value

Registers

rax	0x0
rbx	"SWEB"
rcx	"SWEB"
rdx	"SWEB"
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+7

Emulate CPUID with a Hypercall

1. VM Exit → VMM takes control
2. Check VM Exit reason → VM call
3. Handle VM call
   - 3.1 Check rax → vendor string (rax = 0)
   - 3.2 Set rbx, rcx, rdx to the vendor string
   - 3.3 Set rax to the maximum input value

Registers

rax	0xd
rbx	"SWEB"
rcx	"SWEB"
rdx	"SWEB"
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+7

Emulate CPUID with a Hypercall

1. VM Exit → VMM takes control
2. Check VM Exit reason → VM call
3. Handle VM call
   - 3.1 Check rax → vendor string (rax = 0)
   - 3.2 Set rbx, rcx, rdx to the vendor string
   - 3.3 Set rax to the maximum input value
   - 3.4 Advance rip by the instruction length (field VM-exit instruction length)

Registers

rax	0xd
rbx	"SWEB"
rcx	"SWEB"
rdx	"SWEB"
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+7

Emulate CPUID with a Hypercall

1. VM Exit → VMM takes control
2. Check VM Exit reason → VM call
3. Handle VM call
   - 3.1 Check rax → vendor string (rax = 0)
   - 3.2 Set rbx, rcx, rdx to the vendor string
   - 3.3 Set rax to the maximum input value
   - 3.4 Advance rip by the instruction length (field VM-exit instruction length)

Registers

rax	0xd
rbx	"SWEB"
rcx	"SWEB"
rdx	"SWEB"
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+10

Emulate CPUID with a Hypercall

1. VM Exit → VMM takes control
2. Check VM Exit reason → VM call
3. Handle VM call
   - 3.1 Check rax → vendor string (rax = 0)
   - 3.2 Set rbx, rcx, rdx to the vendor string
   - 3.3 Set rax to the maximum input value
   - 3.4 Advance rip by the instruction length (field VM-exit instruction length)
4. VM Resume

Registers

rax	0xd
rbx	"SWEB"
rcx	"SWEB"
rdx	"SWEB"
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+10

Emulate CPUID with a Hypercall

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 vmcall
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0xd
rbx	"SWEB"
rcx	"SWEB"
rdx	"SWEB"
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+10

Emulate CPUID with a Hypercall

```
main:
 push    rbx
 xor     eax, eax
 sub     rsp, 0x10
 vmcall
 mov     rdi, rsp
 mov     DWORD PTR [rsp], ebx
 mov     DWORD PTR [rsp+0x4], ecx
 mov     DWORD PTR [rsp+0x8], edx
 mov     DWORD PTR [rsp+0x10], 0x0
 call    401030 <puts@plt>
 add     rsp, 0x10
 xor     eax, eax
 pop     rbx
 ret
```

Registers

rax	0xd
rbx	"SWEB"
rcx	"SWEB"
rdx	"SWEB"
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+10

Emulate CPUID with a Hypercall

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 vmcall
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0xd
rbx	"SWEB"
rcx	"SWEB"
rdx	"SWEB"
rdi	0x7ffff7830460
rsp	0x7ffff7830460
rip	main+13

Emulate CPUID with a Hypercall

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 vmcall
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0xd
rbx	"SWEB"
rcx	"SWEB"
rdx	"SWEB"
rdi	0x7ffff7830460
rsp	0x7ffff7830460
rip	main+17

Emulate CPUID with a Hypercall

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 vmcall
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0xd
rbx	"SWEB"
rcx	"SWEB"
rdx	"SWEB"
rdi	0x7ffff7830460
rsp	0x7ffff7830460
rip	main+22

Emulate CPUID with a Hypercall

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 vmcall
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0xd
rbx	"SWEB"
rcx	"SWEB"
rdx	"SWEB"
rdi	0x7ffff7830460
rsp	0x7ffff7830460
rip	main+27

Emulate CPUID with a Hypercall

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 vmcall
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0xd
rbx	"SWEB"
rcx	"SWEB"
rdx	"SWEB"
rdi	0x7ffff7830460
rsp	0x7ffff7830460
rip	main+35

Emulate CPUID with a Hypercall

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 vmcall
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0xc
rbx	"SWEB"
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+40

Emulate CPUID with a Hypercall

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 vmcall
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0xc
rbx	"SWEB"
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830470
rip	main+44

Emulate CPUID with a Hypercall

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 vmcall
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0x0
rbx	"SWEB"
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830470
rip	main+46

Emulate CPUID with a Hypercall

```
main:
 push   rbx
 xor    eax, eax
 sub    rsp, 0x10
 vmcall
 mov    rdi, rsp
 mov    DWORD PTR [rsp], ebx
 mov    DWORD PTR [rsp+0x4], ecx
 mov    DWORD PTR [rsp+0x8], edx
 mov    DWORD PTR [rsp+0x10], 0x0
 call   401030 <puts@plt>
 add    rsp, 0x10
 xor    eax, eax
 pop    rbx
 ret
```

Registers

rax	0x0
rbx	<rbx>
rcx	<rcx>
rdx	<rdx>
rdi	<rdi>
rsp	0x7ffff7830460
rip	main+47

- We can control I/O and MSR accesses

- We can control I/O and MSR accesses
- Bitmaps define which I/O ports and MSRs cause VM exits

- We can control I/O and MSR accesses
- Bitmaps define which I/O ports and MSRs cause VM exits
- Giving the guest direct access to external hardware can be dangerous

- We can control I/O and MSR accesses
- Bitmaps define which I/O ports and MSRs cause VM exits
- Giving the guest direct access to external hardware can be dangerous
  - Guests could mess up the host state

- We can control I/O and MSR accesses
- Bitmaps define which I/O ports and MSRs cause VM exits
- Giving the guest direct access to external hardware can be dangerous
  - Guests could mess up the host state
  - DMAs don't care about our EPT

VM Exit Reasons - IN/OUT and RDMSR/WRMSR

- We can control I/O and MSR accesses
- Bitmaps define which I/O ports and MSRs cause VM exits
- Giving the guest direct access to external hardware can be dangerous
  - Guests could mess up the host state
  - DMAs don't care about our EPT

$\rightarrow$ Disable access or use bitmaps to limit the ports/MSRS


VIOLATION


VIOLATION

- Same as page fault just for EPT translations

- Same as page fault just for EPT translations
- Guest memory accesses that violate the permissions set in the EPT cause VM exits

- Same as page fault just for EPT translations
- Guest memory accesses that violate the permissions set in the EPT cause VM exits
- The VMM can handle EPT violations accordingly

- Same as page fault just for EPT translations
- Guest memory accesses that violate the permissions set in the EPT cause VM exits
- The VMM can handle EPT violations accordingly
- This gives us a lot of room to play around with:

- Same as page fault just for EPT translations
- Guest memory accesses that violate the permissions set in the EPT cause VM exits
- The VMM can handle EPT violations accordingly
- This gives us a lot of room to play around with:
  - On-demand paging

- Same as page fault just for EPT translations
- Guest memory accesses that violate the permissions set in the EPT cause VM exits
- The VMM can handle EPT violations accordingly
- This gives us a lot of room to play around with:
  - On-demand paging
  - Copy-on-Write

- Same as page fault just for EPT translations
- Guest memory accesses that violate the permissions set in the EPT cause VM exits
- The VMM can handle EPT violations accordingly
- This gives us a lot of room to play around with:
  - On-demand paging
  - Copy-on-Write
  - Deduplication

- Same as page fault just for EPT translations
- Guest memory accesses that violate the permissions set in the EPT cause VM exits
- The VMM can handle EPT violations accordingly
- This gives us a lot of room to play around with:
  - On-demand paging
  - Copy-on-Write
  - Deduplication
  - EPT Hooking

- Same as page fault just for EPT translations
- Guest memory accesses that violate the permissions set in the EPT cause VM exits
- The VMM can handle EPT violations accordingly
- This gives us a lot of room to play around with:
  - On-demand paging
  - Copy-on-Write
  - Deduplication
  - EPT Hooking
  - ...

VM Exit Reasons - External Interrupt

VM Exit Reasons - External Interrupt

- Raised by external devices (PIT, disk, network card, ...)

VM Exit Reasons - External Interrupt

- Raised by external devices (PIT, disk, network card, ...)
- External devices continue operating while a guest is scheduled

- Raised by external devices (PIT, disk, network card, ...)
- External devices continue operating while a guest is scheduled

→ Interrupts for the host might occur while the guest is scheduled

VM Exit Reasons - External Interrupt

- Raised by external devices (PIT, disk, network card, ...)
- External devices continue operating while a guest is scheduled

→ Interrupts for the host might occur while the guest is scheduled

- VMCS can be configured for a VM exit on external interrupts

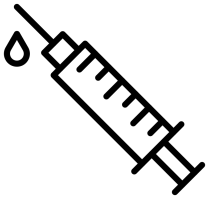
- Raised by external devices (PIT, disk, network card, ...)
- External devices continue operating while a guest is scheduled

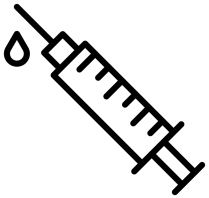
→ Interrupts for the host might occur while the guest is scheduled

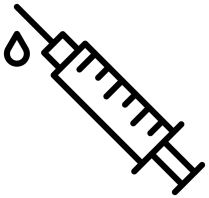
- VMCS can be configured for a VM exit on external interrupts
- Once the host sets IF=1 the external interrupt is delivered normally

- We can force a VM exit as soon as the guest is able to recieve interrupts (Guest RFLAGS.IF $= 1$)

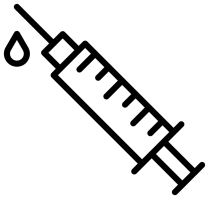
- We can force a VM exit as soon as the guest is able to recieve interrupts (Guest RFLAGS.IF $= 1$)
- Very useful for injecting interrupts



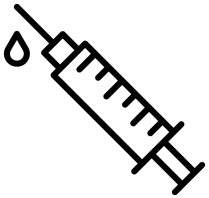




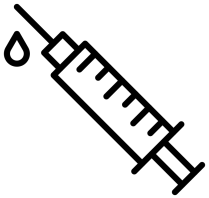
- The host can inject events into the guest



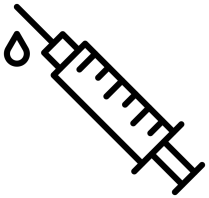
- The host can inject events into the guest
  - External interrupts



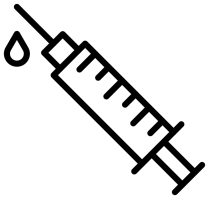
- The host can inject events into the guest
  - External interrupts
  - Non-maskable interrupts



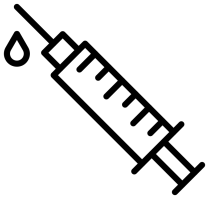
- The host can inject events into the guest
  - External interrupts
  - Non-maskable interrupts
  - Exceptions



- The host can inject events into the guest
  - External interrupts
  - Non-maskable interrupts
  - Exceptions
  - Software interrupts



- The host can inject events into the guest
  - External interrupts
  - Non-maskable interrupts
  - Exceptions
  - Software interrupts
  - ...



- The host can inject events into the guest
  - External interrupts
  - Non-maskable interrupts
  - Exceptions
  - Software interrupts
  - ...
- Very useful for device emulation

Table 25-16. Format of the VM-Entry Interruption-Information Field

Bit Position(s)	Content
7:0	Vector of interrupt or exception
10:8	Interruption type: 0: External interrupt 1: Reserved 2: Non-maskable interrupt (NMI) 3: Hardware exception (e.g,. #PF) 4: Software interrupt (INT *n*) 5: Privileged software exception (INT1) 6: Software exception (INT3 or INTO) 7: Other event
11	Deliver error code (0 = do not deliver; 1 = deliver)
30:12	Reserved
31	Valid

- Set interrupt information in VM-entry interruption-information

- Set interrupt information in VM-entry interruption-information
- Optional: set error code in VM-entry exception error code

- Set interrupt information in VM-entry interruption-information
- Optional: set error code in VM-entry exception error code
- For software interrupts: set instruction length in VM-entry instruction length

- Set interrupt information in VM-entry interruption-information
- Optional: set error code in VM-entry exception error code
- For software interrupts: set instruction length in VM-entry instruction length
- Injection of external interrupts is only allowed if guest RFLAGS.IF $= 1$

- Set interrupt information in VM-entry interruption-information
- Optional: set error code in VM-entry exception error code
- For software interrupts: set instruction length in VM-entry instruction length
- Injection of external interrupts is only allowed if guest RFLAGS.IF $= 1$
  $\rightarrow$ Interrupt-window exiting

1. Check pending interrupts
   → Timer, Keyboard

Int. Window Exiting	0
Int. Info.	0
rflags.IF	0
pending IRQs	Timer, Keyboard

Event Injection - Example

1. Check pending interrupts
   → Timer, Keyboard
2. Check guest IF flag → 0

Int. Window Exiting	0
Int. Info.	0
rflags.IF	0
pending IRQs	Timer, Keyboard

Event Injection - Example

1. Check pending interrupts
   → Timer, Keyboard
2. Check guest IF flag → 0
3. Set interrupt-window exiting

Int. Window Exiting	1
Int. Info.	0
rflags.IF	0
pending IRQs	Timer, Keyboard

Event Injection - Example

1. Check pending interrupts
   → Timer, Keyboard
2. Check guest IF flag → 0
3. Set interrupt-window exiting
4. Continue guest

Int. Window Exiting	1
Int. Info.	0
rflags.IF	0
pending IRQs	Timer, Keyboard

Event Injection - Example

```
push    rbp
mov     rbp, rsp
sti
nop
pop     rbp
ret
```

Int. Window Exiting	1
Int. Info.	0
rflags.IF	0
pending IRQs	Timer, Keyboard

Event Injection - Example

```
push    rbp
mov     rbp, rsp
sti
nop
pop     rbp
ret
```

Int. Window Exiting	1
Int. Info.	0
rflags.IF	0
pending IRQs	Timer, Keyboard

Event Injection - Example

```
push    rbp
mov     rbp, rsp
sti
nop
pop     rbp
ret
```

Int. Window Exiting	1
Int. Info.	0
rflags.IF	0
pending IRQs	Timer, Keyboard

Event Injection - Example

```
push    rbp
mov     rbp, rsp
sti
nop
pop     rbp
ret
```

Int. Window Exiting	1
Int. Info.	0
rflags.IF	1
pending IRQs	Timer, Keyboard

Event Injection - Example

```
push    rbp
mov     rbp, rsp
sti
nop
pop     rbp
ret
```

Int. Window Exiting	1
Int. Info.	0
rflags.IF	1
pending IRQs	Timer, Keyboard

Event Injection - Example

1. VM Exit → VMM takes control

Int. Window Exiting	1
Int. Info.	0
rflags.IF	1
pending IRQs	Timer, Keyboard

Event Injection - Example

1. VM Exit → VMM takes control
2. Check VM Exit reason
   → interrupt window

Int. Window Exiting	1
Int. Info.	0
rflags.IF	1
pending IRQs	Timer, Keyboard

Event Injection - Example

1. VM Exit → VMM takes control
2. Check VM Exit reason
   → interrupt window
   2.1 Check pending interrupts
   → Timer, Keyboard

Int. Window Exiting	1
Int. Info.	0
rflags.IF	1
pending IRQs	Timer, Keyboard

Event Injection - Example

1. VM Exit → VMM takes control
2. Check VM Exit reason
   → interrupt window
   - 2.1 Check pending interrupts
     → Timer, Keyboard
   - 2.2 Set interruption information

Int. Window Exiting	1
Int. Info.	0x80000020
rflags.IF	1
pending IRQs	Timer, Keyboard

Event Injection - Example

1. VM Exit → VMM takes control
2. Check VM Exit reason
   → interrupt window
   - 2.1 Check pending interrupts
     → Timer, Keyboard
   - 2.2 Set interruption information
   - 2.3 Continue guest

Int. Window Exiting	1
Int. Info.	0x80000020
rflags.IF	1
pending IRQs	Keyboard

Event Injection - Example

```
push    rbp
mov     rbp, rsp
sti
nop
pop     rbp
ret
```

Int. Window Exiting	1
Int. Info.	0x80000020
rflags.IF	1
pending IRQs	Keyboard

```
push    rbp
mov     rbp, rsp
sti
nop
pop     rbp
ret
```

Int. Window Exiting	1
Int. Info.	0x80000020
rflags.IF	1
pending IRQs	Keyboard

Event Injection - Example

```
<irq handler>
iret
```

Int. Window Exiting	1
Int. Info.	0x80000020
rflags.IF	0
pending IRQs	Keyboard

Event Injection - Example

```
<irq handler>
iret
```

Int. Window Exiting	1
Int. Info.	0x80000020
rflags.IF	0
pending IRQs	Keyboard

Event Injection - Example

```
push    rbp
mov     rbp, rsp
sti
nop
pop     rbp
ret
```

Int. Window Exiting	1
Int. Info.	0x80000020
rflags.IF	1
pending IRQs	Keyboard

Event Injection - Example

```
push    rbp
mov     rbp, rsp
sti
nop
pop     rbp
ret
```

Int. Window Exiting	1
Int. Info.	0x80000020
rflags.IF	1
pending IRQs	Keyboard

1. VM Exit → VMM takes control

Int. Window Exiting	1
Int. Info.	0x20
rflags.IF	1
pending IRQs	Keyboard

Event Injection - Example

1. VM Exit → VMM takes control
2. Check VM Exit reason
   → interrupt window

Int. Window Exiting	1
Int. Info.	0x20
rflags.IF	1
pending IRQs	Keyboard

Event Injection - Example

1. VM Exit → VMM takes control
2. Check VM Exit reason
   → interrupt window
   2.1 Check pending interrupts
   → Keyboard

Int. Window Exiting	1
Int. Info.	0x20
rflags.IF	1
pending IRQs	Keyboard

Event Injection - Example

1. VM Exit → VMM takes control
2. Check VM Exit reason
   → interrupt window
   2.1 Check pending interrupts
   → Keyboard
   2.2 Set interruption information

Int. Window Exiting	1
Int. Info.	0x80000021
rflags.IF	1
pending IRQs	Keyboard

Event Injection - Example

1. VM Exit → VMM takes control
2. Check VM Exit reason
   → interrupt window
   2.1 Check pending interrupts
   → Keyboard
   2.2 Set interruption information
   2.3 No other pending interrupts
   → clear interrupt-window exiting

Int. Window Exiting	0
Int. Info.	0x80000021
rflags.IF	1
pending IRQs	

1. VM Exit → VMM takes control
2. Check VM Exit reason
   → interrupt window
   2.1 Check pending interrupts
   → Keyboard
   2.2 Set interruption information
   2.3 No other pending interrupts
   → clear interrupt-window exiting
   2.4 Continue guest

Int. Window Exiting	0
Int. Info.	0x80000021
rflags.IF	1
pending IRQs	

Event Injection - Example

```
push    rbp
mov     rbp, rsp
sti
nop
pop     rbp
ret
```

Int. Window Exiting	0
Int. Info.	0x80000021
rflags.IF	1
pending IRQs	

```
push    rbp
mov     rbp, rsp
sti
nop
pop     rbp
ret
```

Int. Window Exiting	0
Int. Info.	0x80000021
rflags.IF	1
pending IRQs	

Event Injection - Example

```
<irq handler>
iret
```

Int. Window Exiting	0
Int. Info.	0x80000021
rflags.IF	0
pending IRQs	

Event Injection - Example

```
<irq handler>
iret
```

Int. Window Exiting	0
Int. Info.	0x80000021
rflags.IF	0
pending IRQs	

Event Injection - Example

```
push    rbp
mov     rbp, rsp
sti
nop
pop     rbp
ret
```

Int. Window Exiting	0
Int. Info.	0x80000021
rflags.IF	1
pending IRQs	

Event Injection - Example

```
push    rbp
mov     rbp, rsp
sti
nop
pop     rbp
ret
```

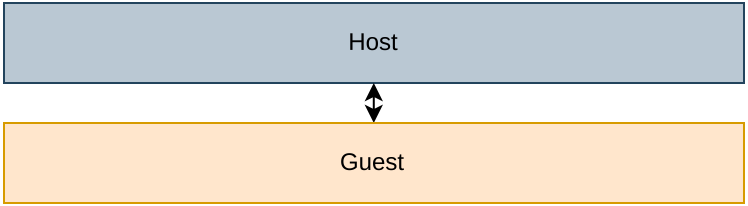
Int. Window Exiting	0
Int. Info.	0x80000021
rflags.IF	1
pending IRQs	

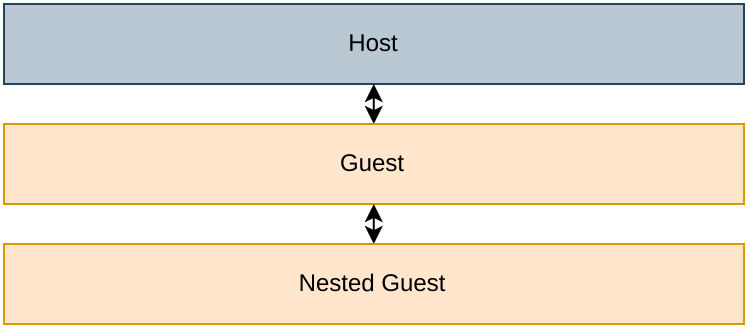
Event Injection - Example

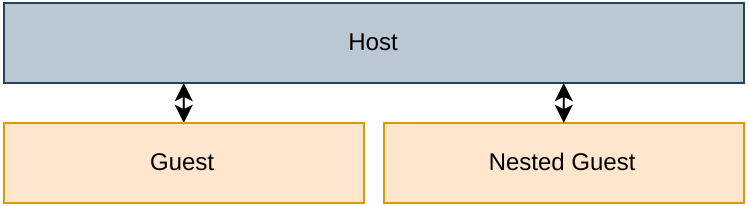
```
push    rbp
mov     rbp, rsp
sti
nop
pop     rbp
ret
```

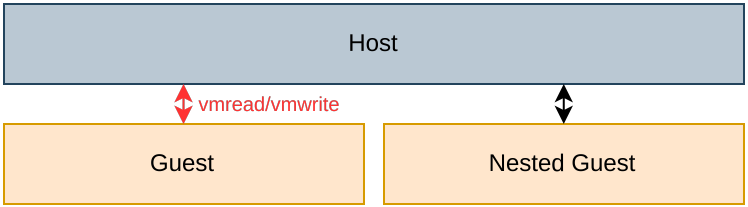
Int. Window Exiting	0
Int. Info.	0x80000021
rflags.IF	1
pending IRQs	

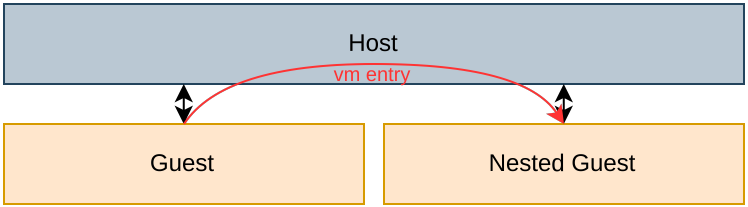
Nested Virtualization

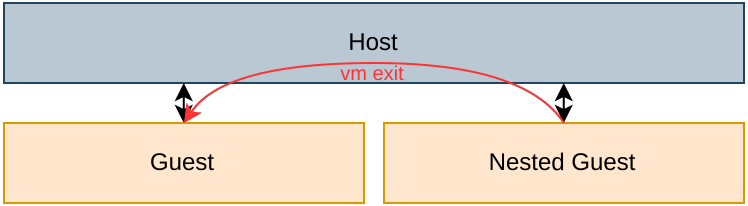

Host
Guest

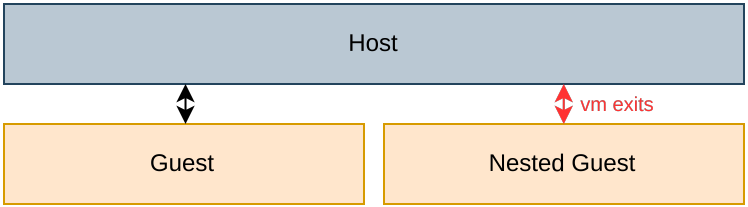

Host
Guest
Nested Guest


Host
Guest
Nested Guest


Host
vmread/vmwrite
Guest
Nested Guest


Host
vm entry
Guest
Nested Guest


Host
vm exit
Guest
Nested Guest


Host
vm exits
Guest
Nested Guest

Nested Virtualization

Nested Virtualization

- "Real" nested VMX would be too expensive

- "Real" nested VMX would be too expensive

→ Run the nested guest as a normal guest and emulate VMX

- "Real" nested VMX would be too expensive

→ Run the nested guest as a normal guest and emulate VMX

- vmread and vmwrite cause a VM exit

- "Real" nested VMX would be too expensive

→ Run the nested guest as a normal guest and emulate VMX

- vmread and vmwrite cause a VM exit
- Host maintains a real VMCS and configurations for the nested guests

- "Real" nested VMX would be too expensive

→ Run the nested guest as a normal guest and emulate VMX

- vmread and vmwrite cause a VM exit
- Host maintains a real VMCS and configurations for the nested guests
- VM entry causes a VM exit to the host

- "Real" nested VMX would be too expensive

→ Run the nested guest as a normal guest and emulate VMX

- vmread and vmwrite cause a VM exit
- Host maintains a real VMCS and configurations for the nested guests
- VM entry causes a VM exit to the host

→ Host schedules nested guest

- "Real" nested VMX would be too expensive

→ Run the nested guest as a normal guest and emulate VMX

- vmread and vmwrite cause a VM exit
- Host maintains a real VMCS and configurations for the nested guests
- VM entry causes a VM exit to the host

→ Host schedules nested guest

→ Nested guest VM exits can be forwarded to the guest or handled by the host

- Can be enabled in the VMCS

- Can be enabled in the VMCS
- VMCS link pointer contains the address of a shadow VMCS

- Can be enabled in the VMCS
- VMCS link pointer contains the address of a shadow VMCS
- Guest vmwrite and vmread modify the shadow VMCS

- Can be enabled in the VMCS
- VMCS link pointer contains the address of a shadow VMCS
- Guest vmwrite and vmread modify the shadow VMCS

$\rightarrow$ less VM exits

- Can be enabled in the VMCS
- VMCS link pointer contains the address of a shadow VMCS
- Guest vmwrite and vmread modify the shadow VMCS

$\rightarrow$ less VM exits

- Host can use vmread and vmwrite on the shadow VMCS

- Can be enabled in the VMCS
- VMCS link pointer contains the address of a shadow VMCS
- Guest vmwrite and vmread modify the shadow VMCS

→ less VM exits

- Host can use vmread and vmwrite on the shadow VMCS
- the shadow VMCS can not be launched

- Can be enabled in the VMCS
- VMCS link pointer contains the address of a shadow VMCS
- Guest vmwrite and vmread modify the shadow VMCS

→ less VM exits

- Host can use vmread and vmwrite on the shadow VMCS
- the shadow VMCS can not be launched
- Host still has to maintain a normal VMCS for the nested

- Can be enabled in the VMCS
- VMCS link pointer contains the address of a shadow VMCS
- Guest vmwrite and vmread modify the shadow VMCS

→ less VM exits

- Host can use vmread and vmwrite on the shadow VMCS
- the shadow VMCS can not be launched
- Host still has to maintain a normal VMCS for the nested
- Bitmaps can be used to restrict the guest from accessing VMCS fields

- Can be enabled in the VMCS
- VMCS link pointer contains the address of a shadow VMCS
- Guest vmwrite and vmread modify the shadow VMCS

→ less VM exits

- Host can use vmread and vmwrite on the shadow VMCS
- the shadow VMCS can not be launched
- Host still has to maintain a normal VMCS for the nested
- Bitmaps can be used to restrict the guest from accessing VMCS fields

→ Host does not always need to copy the whole shadow VMCS

- Up to 25 memory accesses per address dereference with EPT

Nested EPT

- Up to 25 memory accesses per address dereference with EPT
- 125 if we add a third layer

Nested EPT

- Up to 25 memory accesses per address dereference with EPT
- 125 if we add a third layer

$\rightarrow$ use normal EPT for nested guests

Nested EPT

- Up to 25 memory accesses per address dereference with EPT
- 125 if we add a third layer

→ use normal EPT for nested guests

- Guest maintains a copy of the real EPT

Nested EPT

- Up to 25 memory accesses per address dereference with EPT
- 125 if we add a third layer

→ use normal EPT for nested guests

- Guest maintains a copy of the real EPT
- Host maintains the real nested guest EPT

- Host has to...

- Host has to...
  - Update the real EPT if the shadow changes

- Host has to...
  - Update the real EPT if the shadow changes
  - Copy accessed and dirty bits from the real EPT to the shadow

- Host has to...
  - Update the real EPT if the shadow changes
  - Copy accessed and dirty bits from the real EPT to the shadow
- updating real EPT can be done on demand:

- Host has to...
  - Update the real EPT if the shadow changes
  - Copy accessed and dirty bits from the real EPT to the shadow
- updating real EPT can be done on demand:
  1. schedule nested guest with empty EPT

- Host has to...
  - Update the real EPT if the shadow changes
  - Copy accessed and dirty bits from the real EPT to the shadow
- updating real EPT can be done on demand:
  1. schedule nested guest with empty EPT
  2. on EPT violation check the shadow EPT

- Host has to...
  - Update the real EPT if the shadow changes
  - Copy accessed and dirty bits from the real EPT to the shadow
- updating real EPT can be done on demand:
  1. schedule nested guest with empty EPT
  2. on EPT violation check the shadow EPT
  3. map page if it exists, forward VM exit to guest if it does not

- Host has to...
  - Update the real EPT if the shadow changes
  - Copy accessed and dirty bits from the real EPT to the shadow
- updating real EPT can be done on demand:
  1. schedule nested guest with empty EPT
  2. on EPT violation check the shadow EPT
  3. map page if it exists, forward VM exit to guest if it does not
- set shadow EPT pages to read only to detect updates to existing mappings

Exercise

Exercise Overview

Exercise Overview

- Goal: write a hypervisor that runs SWEB inside of SWEB

- Goal: write a hypervisor that runs SWEB inside of SWEB
- Base requirements:

- Goal: write a hypervisor that runs SWEB inside of SWEB
- Base requirements:
  - The guest has to eventually get to the SWEB entry and boot normally <Boot Code>

- Goal: write a hypervisor that runs SWEB inside of SWEB
- Base requirements:
  - The guest has to eventually get to the SWEB entry and boot normally <Boot Code>
  - Emulate video output, PIC, PIT and the keyboard

- Goal: write a hypervisor that runs SWEB inside of SWEB
- Base requirements:
  - The guest has to eventually get to the SWEB entry and boot normally <Boot Code>
  - Emulate video output, PIC, PIT and the keyboard
  - Forward the host VFSSyscalls to the guest via hypercalls

- Goal: write a hypervisor that runs SWEB inside of SWEB
- Base requirements:
  - The guest has to eventually get to the SWEB entry and boot normally <Boot Code>
  - Emulate video output, PIC, PIT and the keyboard
  - Forward the host VFSSyscalls to the guest via hypercalls
  - The guest should run normally and receive regular timer interrupts

- Goal: write a hypervisor that runs SWEB inside of SWEB
- Base requirements:
  - The guest has to eventually get to the SWEB entry and boot normally <Boot Code>
  - Emulate video output, PIC, PIT and the keyboard
  - Forward the host VFSSyscalls to the guest via hypercalls
  - The guest should run normally and receive regular timer interrupts
  - Running a guest should not block the host from running other processes

- Start the guest in protected mode

- Start the guest in protected mode
- Copy the currently running SWEB

- Start the guest in protected mode
- Copy the currently running SWEB
  - reset multiboot remainder and fd_num_

- Start the guest in protected mode
- Copy the currently running SWEB
  - reset multiboot remainder and fd_num_
- Checkout parseMultibootHeader for creating a new multiboot header

Questions?