

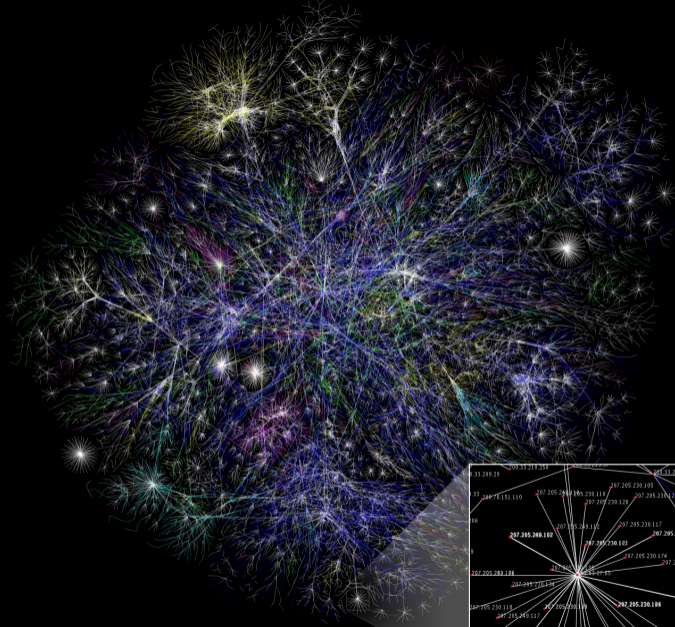
# Cloud Operating Systems

**Daniel Gruss, Andreas Kogler, Fabian Rauscher)**

2022-03-09











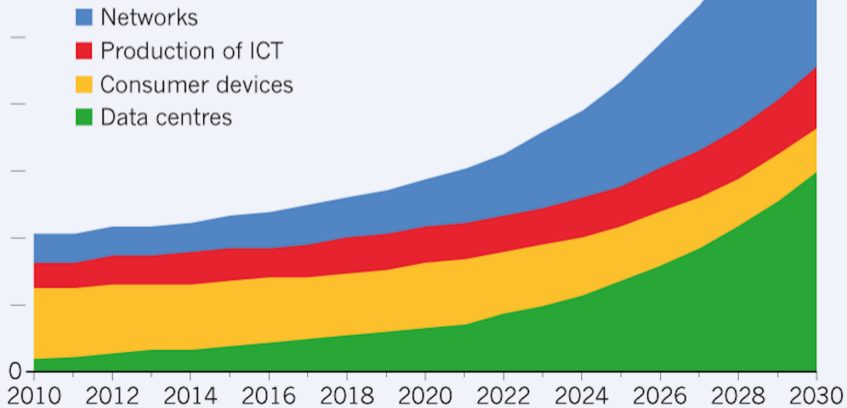


9,000 terawatt hours (TWh)

©nature

## ENERGY FORECAST

20.9% of projected  
electricity demand





from outages blamed on cryptocurrency mining activities.<sup>9</sup>

Despite the fact that, in both examples, policymakers did not decide to take action because of environmental concerns, the examples illustrate how policymakers might have multiple options in putting a halt to cryptocurrency mining. Although Bitcoin might be a decentralized currency, many aspects of the ecosystem surrounding it are not. The competitive Bitcoin market drives miners to take advantage of economies of scale in lowering costs, which also makes it harder for them to operate under the radar. Large-scale miners can easily be targeted with higher electricity rates, moratoria, or, in the most extreme case, confiscation of the equipment used. Moreover, the supply chain of specialized Bitcoin mining de-

## CONCLUSION

As the price of Bitcoin rises, the negative externalities associated with Bitcoin mining increase in kind. This Commentary has shown how a simple economic model might be used to estimate the potential environmental impact of Bitcoin mining for a given Bitcoin price. These estimates reveal that the record-breaking surge in Bitcoin price at the start of 2021 could result in the network consuming as much energy as all data centers globally, with an associated carbon footprint matching London's footprint size. Beyond these environmental impacts, the production of specialized mining devices might exacerbate the global shortage of chips, which could effect the ability to work from home, the economic recovery after the COVID-19 crisis, and

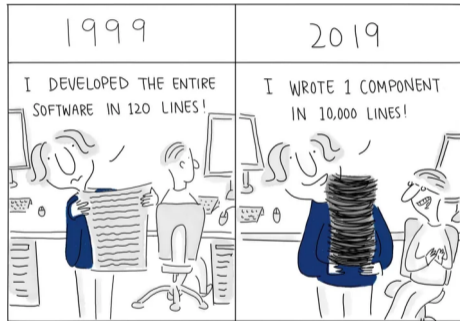
NOT SOLVE BITCOIN'S SUSTAINABILITY PROBLEM. *Joule* 3, 893–898.

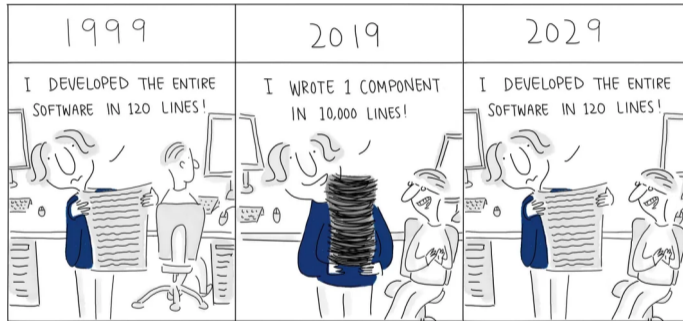
9. Arab News. (2021). Crypto-miners take down Iran electric grids, prompting crackdown. <https://www.arabnews.com/node/1794836/middle-east>.
10. Blandin, A., Pieters, G.C., Wu, Y., Eisermann, T., Dek, A., Taylor, S., and Njoki, D. (2020). 3rd global cryptoasset benchmarking study. <https://www.jbs.cam.ac.uk/faculty-research/centres/alternative-finance/publications/3rd-global-cryptoasset-benchmarking-study/>.
11. Stoll, C., Klaaßen, L., and Gallersdörfer, U. (2019). The Carbon Footprint of Bitcoin. *Joule* 3, 1647–1661.
12. Gallersdörfer, U., Klaaßen, L., and Stoll, C. (2020). Energy Consumption of Cryptocurrencies Beyond Bitcoin. *Joule* 4, 1843–1846.
13. Jin, H., Busvine, D., and Kirton, D. (2020). Analysis: Global chip shortage threatens production of laptops, smartphones and more (Reuters).
14. de Vries, A. (2020). Bitcoin's energy consumption is underestimated: A market dynamics approach. *Energy Res. Soc. Sci.* 70, 101721.

**WE'RE ALL**



**DOOMED**

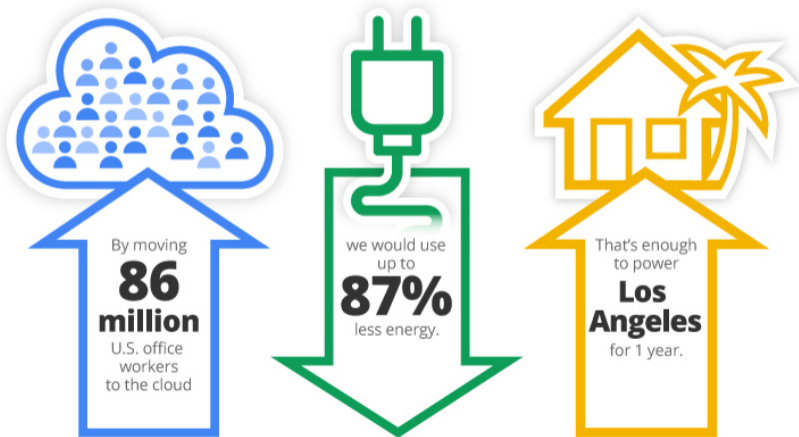




**YESS...**

**MOAR SOFTWARE COMPLEXITY**

# Moving to the cloud can save up to 87% of IT energy



**Cloud means Efficiency**

Apply OS techniques - Example?

- Processes used to have access to all physical memory → that's not efficient!



Apply OS techniques - Example?

- Processes used to have access to all physical memory → that's not efficient!
- Virtualize memory → processes can share resources of one machine and utilize it better

Apply OS techniques - Example?

- Processes used to have access to all physical memory → that's not efficient!
- Virtualize memory → processes can share resources of one machine and utilize it better
- Processes need all the same pages → that's not efficient!

Apply OS techniques - Example?

- Processes used to have access to all physical memory → that's not efficient!
- Virtualize memory → processes can share resources of one machine and utilize it better
- Processes need all the same pages → that's not efficient!
- Let them share memory, using COW, page deduplication, etc.

Apply OS techniques - Example?

- Processes used to have access to all physical memory → that's not efficient!
- Virtualize memory → processes can share resources of one machine and utilize it better
- Processes need all the same pages → that's not efficient!
- Let them share memory, using COW, page deduplication, etc.
- Processes often cannot do anything but wait → that's not efficient!

Apply OS techniques - Example?

- Processes used to have access to all physical memory → that's not efficient!
- Virtualize memory → processes can share resources of one machine and utilize it better
- Processes need all the same pages → that's not efficient!
- Let them share memory, using COW, page deduplication, etc.
- Processes often cannot do anything but wait → that's not efficient!
- Let other processes run in between



- Efficiency

- Efficiency
- Isolation of tenants (security, reliability, availability)



- Efficiency
- Isolation of tenants (security, reliability, availability)
- Abstraction of hardware

**Virtualization** allows to represent resources in a computer in a way they can be used easily and without the need to know details of their properties

- Decouple operating system from hardware

- Decouple operating system from hardware
  - “computer in computer” - implemented in software

- Decouple operating system from hardware
  - “computer in computer” - implemented in software
  - includes devices (network, keyboard, sound...)

- Decouple operating system from hardware
  - “computer in computer” - implemented in software
  - includes devices (network, keyboard, sound...)
- OS in VM “sees” its hardware, irrespective from the actual hardware in use

- Decouple operating system from hardware
  - “computer in computer” - implemented in software
  - includes devices (network, keyboard, sound...)
- OS in VM “sees” its hardware, irrespective from the actual hardware in use
- OS does not know if HW is concurrently used by other VMS











- Cheaper hardware: one server for one task was common



- Cheaper hardware: one server for one task was common
- most of these servers: idle time 90%



- Cheaper hardware: one server for one task was common
- most of these servers: idle time 90%
- cost issue:



- Cheaper hardware: one server for one task was common
- most of these servers: idle time 90%
- cost issue:
  - support, maintenance



- Cheaper hardware: one server for one task was common
- most of these servers: idle time 90%
- cost issue:
  - support, maintenance
  - power consumption (operation, cooling)



- Cheaper hardware: one server for one task was common
- most of these servers: idle time 90%
- cost issue:
  - support, maintenance
  - power consumption (operation, cooling)
  - space





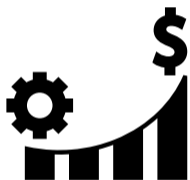
- Cheaper hardware: one server for one task was common
- most of these servers: idle time 90%
- cost issue:
  - support, maintenance
  - power consumption (operation, cooling)
  - space
- Virtualization allows consolidation



- Cheaper hardware: one server for one task was common
- most of these servers: idle time 90%
- cost issue:
  - support, maintenance
  - power consumption (operation, cooling)
  - space
- Virtualization allows consolidation
  - multiple servers on one box









- Better hardware utilization



- Better hardware utilization
- Lower administration cost



- Better hardware utilization
- Lower administration cost
- long-term operations of older applications





- Better hardware utilization
- Lower administration cost
- long-term operations of older applications
- lower down-times



- Better hardware utilization
- Lower administration cost
- long-term operations of older applications
- lower down-times
- simple migration to more powerful hardware









- Performance cost: slower I/O operation



- Performance cost: slower I/O operation
- single point of failure: requires better hardware reliability



- Performance cost: slower I/O operation
- single point of failure: requires better hardware reliability
- security gets more complex



- Virtualization no significant role in internet hosting

- Virtualization no significant role in internet hosting
- often PaaS

- Virtualization no significant role in internet hosting
- often PaaS
- Web hoster (FTP access, HTTP website)

- Virtualization no significant role in internet hosting
- often PaaS
- Web hoster (FTP access, HTTP website)
- Isolation on the OS level (tenants as users)

- Virtualization no significant role in internet hosting
- often PaaS
- Web hoster (FTP access, HTTP website)
- Isolation on the OS level (tenants as users)
- no hardware support → expensive + many problems

- OS-level Virtualization

- OS-level Virtualization
- Para-Virtualization

- OS-level Virtualization
- Para-Virtualization
- Full Virtualization



- OS-level Virtualization
- Para-Virtualization
- Full Virtualization
- Hardware-Assisted Virtualization

- integrated into kernel

- integrated into kernel
- all application software intended to run in a virtual environment get strictly separated virtual runtime environments (container, jail)

- integrated into kernel
- all application software intended to run in a virtual environment get strictly separated virtual runtime environments (container, jail)
- no separate kernels - only process level virtualization

- integrated into kernel
- all application software intended to run in a virtual environment get strictly separated virtual runtime environments (container, jail)
- no separate kernels - only process level virtualization
- can't run other OSes - only for applications

- integrated into kernel
- all application software intended to run in a virtual environment get strictly separated virtual runtime environments (container, jail)
- no separate kernels - only process level virtualization
- can't run other OSes - only for applications
- examples: OpenVZ, Docker, (s)chroot











- Cooperation with OS: OS is aware of virtualization



- Cooperation with OS: OS is aware of virtualization
- needs to modify guest



- Cooperation with OS: OS is aware of virtualization
- needs to modify guest
- not usable for closed source OSes

- OS not aware of being virtualized

- OS not aware of being virtualized
- no need to adapt guest

- OS not aware of being virtualized
- no need to adapt guest
- reduced performance

- OS not aware of being virtualized
- no need to adapt guest
- reduced performance
  - up to 25%



- OS not aware of being virtualized
- no need to adapt guest
- reduced performance
  - up to 25%
- full virtualization of HW required (e.g., emulation via qemu)

- OS not aware of being virtualized
- no need to adapt guest
- reduced performance
  - up to 25%
- full virtualization of HW required (e.g., emulation via qemu)
  - virtual machines not allowed to access physical components

- OS not aware of being virtualized
- no need to adapt guest
- reduced performance
  - up to 25%
- full virtualization of HW required (e.g., emulation via qemu)
  - virtual machines not allowed to access physical components
  - every physical component has to be virtualized and requires drivers in OS

- Guest no longer runs in kernel mode (Ring 0)

- Guest no longer runs in kernel mode (Ring 0)
  - parts that require kernel privileges won't run

- Guest no longer runs in kernel mode (Ring 0)
  - parts that require kernel privileges won't run
- hypervisor (VMM) changes binaries of guest-OS on the fly

- Guest no longer runs in kernel mode (Ring 0)
  - parts that require kernel privileges won't run
- hypervisor (VMM) changes binaries of guest-OS on the fly
- allows supporting any OS

- Guest no longer runs in kernel mode (Ring 0)
  - parts that require kernel privileges won't run
- hypervisor (VMM) changes binaries of guest-OS on the fly
- allows supporting any OS
  - no need to change source



- Guest no longer runs in kernel mode (Ring 0)
  - parts that require kernel privileges won't run
- hypervisor (VMM) changes binaries of guest-OS on the fly
- allows supporting any OS
  - no need to change source
- high performance penalty

- First full x86 virtualization

- First full x86 virtualization
- hypervisor continuously reads program code before it is executed (prescan)

- First full x86 virtualization
- hypervisor continuously reads program code before it is executed (prescan)
- looking for relevant commands

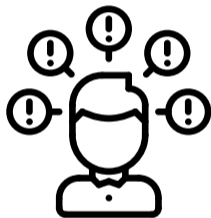
- First full x86 virtualization
- hypervisor continuously reads program code before it is executed (prescan)
- looking for relevant commands
  - change of system state

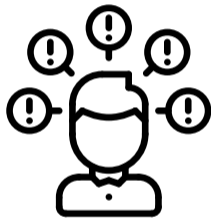
- First full x86 virtualization
- hypervisor continuously reads program code before it is executed (prescan)
- looking for relevant commands
  - change of system state
  - commands depending on CPU state

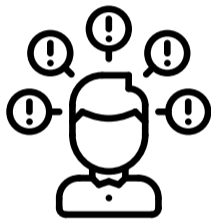
- First full x86 virtualization
- hypervisor continuously reads program code before it is executed (prescan)
- looking for relevant commands
  - change of system state
  - commands depending on CPU state
- sets breakpoint and lets OS run



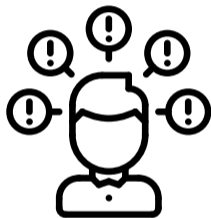




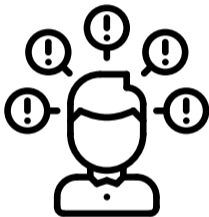




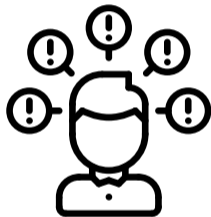
- Diverse problems were to be solved when virtualizing on IA-32:



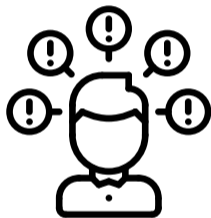
- Diverse problems were to be solved when virtualizing on IA-32:
  - Ring Problems



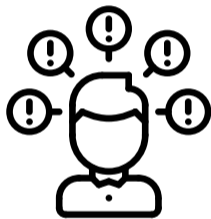
- Diverse problems were to be solved when virtualizing on IA-32:
  - Ring Problems
  - Address Space Compression



- Diverse problems were to be solved when virtualizing on IA-32:
  - Ring Problems
  - Address Space Compression
  - Non-Faulting Access to Priv. State

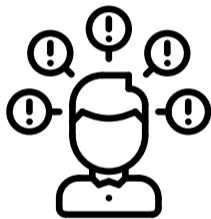


- Diverse problems were to be solved when virtualizing on IA-32:
  - Ring Problems
  - Address Space Compression
  - Non-Faulting Access to Priv. State
  - SYSENTER / SYSEXIT



- Diverse problems were to be solved when virtualizing on IA-32:
  - Ring Problems
  - Address Space Compression
  - Non-Faulting Access to Priv. State
  - SYSENTER / SYSEXIT
  - Interrupt Virtualization





- Diverse problems were to be solved when virtualizing on IA-32:
  - Ring Problems
  - Address Space Compression
  - Non-Faulting Access to Priv. State
  - SYSENTER / SYSEXIT
  - Interrupt Virtualization
  - Hidden States

- usually: application run in ring 3, kernel in ring 0

- usually: application run in ring 3, kernel in ring 0
- guest may not run in ring 0

- usually: application run in ring 3, kernel in ring 0
- guest may not run in ring 0
- ring deprivedging needed: guest must run in ring  $i \neq 0$

- usually: application run in ring 3, kernel in ring 0
- guest may not run in ring 0
- ring deprivedging needed: guest must run in ring  $i \neq 0$ 
  - most often 1 or 3

- guest has to run in a ring it has not been developed for

- guest has to run in a ring it has not been developed for
- certain instructions contain privilege level in result (e.g. `PUSH CS`)

- guest has to run in a ring it has not been developed for
- certain instructions contain privilege level in result (e.g. `PUSH CS`)
- guest OS can find out ring it is running in



- guest has to run in a ring it has not been developed for
- certain instructions contain privilege level in result (e.g. `PUSH CS`)
- guest OS can find out ring it is running in
- may result in diverse problems

- Guest expects to have full address space available

- Guest expects to have full address space available
- hypervisor requires part of address space

- Guest expects to have full address space available
- hypervisor requires part of address space
  - control structures for switching between guest and hypervisor

- Guest expects to have full address space available
- hypervisor requires part of address space
  - control structures for switching between guest and hypervisor
- Access to these areas not allowed for guest. Invokes switch to hypervisor who has to emulate these accesses

- unprivileged software may not access certain elements of the CPU state

- unprivileged software may not access certain elements of the CPU state
- access by guest results in fault: hypervisor can emulate instructions

- unprivileged software may not access certain elements of the CPU state
- access by guest results in fault: hypervisor can emulate instructions
- IA-32 possesses instructions that do not induce a fault:



- unprivileged software may not access certain elements of the CPU state
- access by guest results in fault: hypervisor can emulate instructions
- IA-32 possesses instructions that do not induce a fault:
  - Registers GDTR, IDTR, LDTR and TR are only modifiable in ring 0

- unprivileged software may not access certain elements of the CPU state
- access by guest results in fault: hypervisor can emulate instructions
- IA-32 possesses instructions that do not induce a fault:
  - Registers GDTR, IDTR, LDTR and TR are only modifiable in ring 0
  - can be executed in any ring without fault (without function)

- special commands for fast syscalls

- special commands for fast syscalls
- SYSENTER always switches to ring 0

- special commands for fast syscalls
- SYSENTER always switches to ring 0
- SYSEXIT can only be executed in ring 0

- special commands for fast syscalls
- SYSENTER always switches to ring 0
- SYSEXIT can only be executed in ring 0
- ring 1 thus is problematic

- special commands for fast syscalls
- SYSENTER always switches to ring 0
- SYSEXIT can only be executed in ring 0
- ring 1 thus is problematic
  - SYSENTER switches to hypervisor → has to emulate

- special commands for fast syscalls
- SYSENTER always switches to ring 0
- SYSEXIT can only be executed in ring 0
- ring 1 thus is problematic
  - SYSENTER switches to hypervisor → has to emulate
  - SYSEXIT switches to hypervisor → has to emulate



- interrupts can be masked (so they do not occur if not welcome)

- interrupts can be masked (so they do not occur if not welcome)
- controlled by IF-flag in EFLAGS-Register

- interrupts can be masked (so they do not occur if not welcome)
- controlled by IF-flag in EFLAGS-Register
- Interrupts managed by VM though

- interrupts can be masked (so they do not occur if not welcome)
- controlled by IF-flag in EFLAGS-Register
- Interrupts managed by VM though
- change of IF → fault to hypervisor

- interrupts can be masked (so they do not occur if not welcome)
- controlled by IF-flag in EFLAGS-Register
- Interrupts managed by VM though
- change of IF → fault to hypervisor
- OS do this quite often → performance problem

- interrupts can be masked (so they do not occur if not welcome)
- controlled by IF-flag in EFLAGS-Register
- Interrupts managed by VM though
- change of IF → fault to hypervisor
- OS do this quite often → performance problem
- forwarding of virtual interrupts must consider IF











- Not all state-information accessible via registers



- Not all state-information accessible via registers
- cannot be saved and restored when switching between VMs

- Two new operating modes:

- Two new operating modes:
  - VMX root operation

- Two new operating modes:
  - VMX root operation
    - for hypervisor

- Two new operating modes:
  - VMX root operation
    - for hypervisor
  - VMX non-root operation

- Two new operating modes:
  - VMX root operation
    - for hypervisor
  - VMX non-root operation
    - controlled by hypervisor

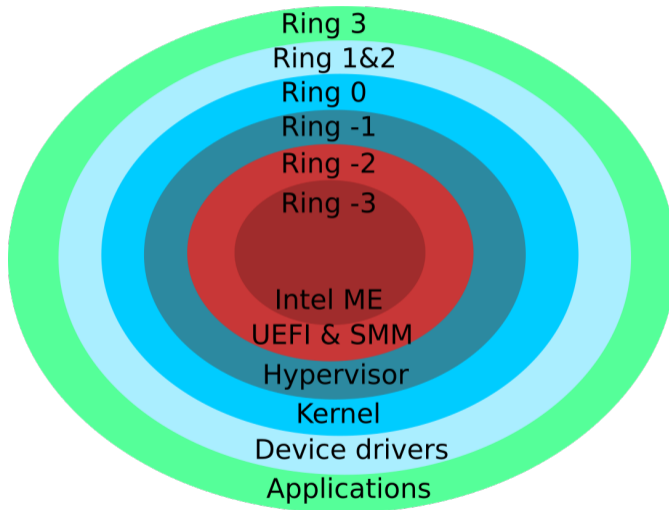


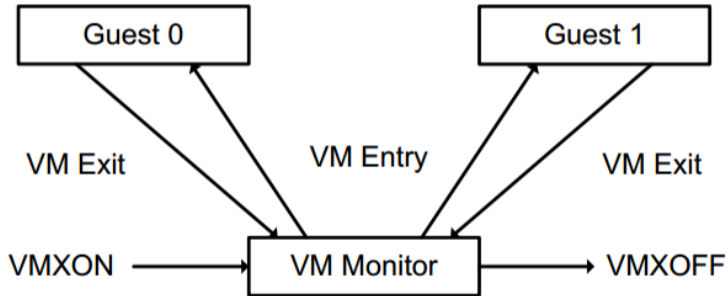
- Two new operating modes:
  - VMX root operation
    - for hypervisor
  - VMX non-root operation
    - controlled by hypervisor
    - supports VMs

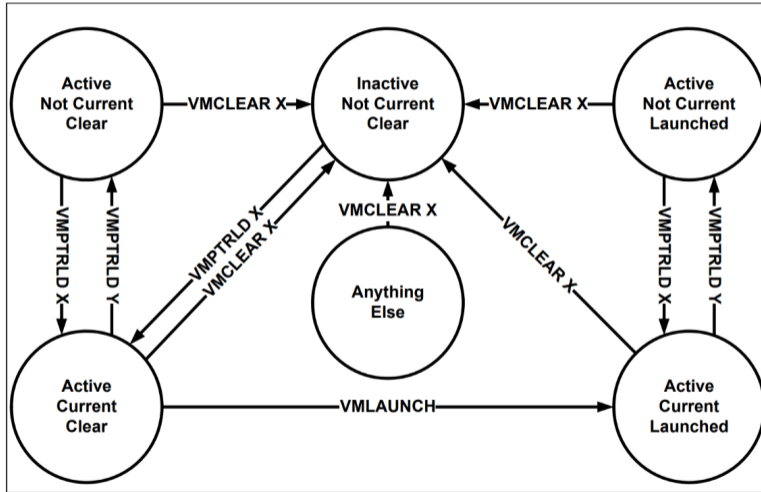
- Two new operating modes:
  - VMX root operation
    - for hypervisor
  - VMX non-root operation
    - controlled by hypervisor
    - supports VMs
- Both modes have ring 0-3

- Two new operating modes:
  - VMX root operation
    - for hypervisor
  - VMX non-root operation
    - controlled by hypervisor
    - supports VMs
- Both modes have ring 0-3
- guest can run in ring 0

- Two new operating modes:
  - VMX root operation
    - for hypervisor
  - VMX non-root operation
    - controlled by hypervisor
    - supports VMs
- Both modes have ring 0-3
- guest can run in ring 0
- hypervisor said to be running in “ring -1”







- VM entry: root operation  $\rightarrow$  non-root operation



- VM entry: root operation  $\rightarrow$  non-root operation
- VM exit: non-root operation  $\rightarrow$  root operation

- VM entry: root operation  $\rightarrow$  non-root operation
- VM exit: non-root operation  $\rightarrow$  root operation
- VMCS: Virtual Machine Control Structure

- VM entry: root operation  $\rightarrow$  non-root operation
- VM exit: non-root operation  $\rightarrow$  root operation
- VMCS: Virtual Machine Control Structure
  - Guest-state-area

- VM entry: root operation  $\rightarrow$  non-root operation
- VM exit: non-root operation  $\rightarrow$  root operation
- VMCS: Virtual Machine Control Structure
  - Guest-state-area
  - Host-state-area

- VM entry: root operation  $\rightarrow$  non-root operation
- VM exit: non-root operation  $\rightarrow$  root operation
- VMCS: Virtual Machine Control Structure
  - Guest-state-area
  - Host-state-area
- Entry/Exit loads/saves information using the proper area

- Contains elements comprising the state of the virtual CPU of a VMCS

- Contains elements comprising the state of the virtual CPU of a VMCS
- VM-exit requires loading certain registers (like segment registers, CR3, IRTR...)

- Contains elements comprising the state of the virtual CPU of a VMCS
- VM-exit requires loading certain registers (like segment registers, CR3, IRTR...)
- GSA contains fields for these registers



- Contains elements comprising the state of the virtual CPU of a VMCS
- VM-exit requires loading certain registers (like segment registers, CR3, IRTR...)
- GSA contains fields for these registers
- GSA contains fields for other information not readable via registers

- Contains elements comprising the state of the virtual CPU of a VMCS
- VM-exit requires loading certain registers (like segment registers, CR3, IRTR...)
- GSA contains fields for these registers
- GSA contains fields for other information not readable via registers
  - e.g. interruptability state

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

Copyright 2017, [@Noteworthy](#) (Intel Manual of July 2017)

# 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	Selector	Base Address	Segment Limit	Access Right	
IDTR	Selector	Base Address	Segment Limit	Access Right	
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

CRO		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	

- Addressed using physical addresses

- Addressed using physical addresses
- not part of guest address space

- Addressed using physical addresses
- not part of guest address space
- hypervisor may run in different address space as guest (CR3 part of state)



- Addressed using physical addresses
- not part of guest address space
- hypervisor may run in different address space as guest (CR3 part of state)
- VM-exits leave detailed information on reason for exit in VMCS

- Addressed using physical addresses
- not part of guest address space
- hypervisor may run in different address space as guest (CR3 part of state)
- VM-exits leave detailed information on reason for exit in VMCS
  - exit reason

- Addressed using physical addresses
- not part of guest address space
- hypervisor may run in different address space as guest (CR3 part of state)
- VM-exits leave detailed information on reason for exit in VMCS
  - exit reason
  - exit qualification

## VM-EXIT CONTROL FIELDS

VM-Exit Controls	Save debug controls		Host address space size		Load IA32_PERF_GLOBAL_CTRL	
	Acknowledge interrupt on exit	Save IA32_PAT	Load IA32_PAT	Save IA32_EFER	Load IA32_EFER	
	Save VMX preemption timer value		Clear IA32_BNDCFGS		Conceal VM exits from Intel PT	
VM-Exit Controls for MSRs	VM-exit MSR-store count	VM-exit MSR-store address				
	VM-exit MSR-load count	VM-exit MSR-load address				

## VM-EXIT INFORMATION FIELDS

Basic VM-Exit Information	Exit reason		Exit qualification			
	Guest-linear address		Guest-physical address			
VM Exits Due to Vectored Events	VM-exit interruption information		VM-exit interruption error code			
VM Exits That Occur During Event Delivery	IDT-vectoring information		IDT-vectoring error code			
VM Exits Due to Instruction Execution	VM-exit instruction length		VM-exit instruction information			
	I/O RCX	I/O RSI	I/O RDI	I/O RIP		
VM-instruction error field						

- Example: MOV CR

- Example: MOV CR
- Exit reason: “control register access”

- Example: MOV CR
- Exit reason: “control register access”
- Exit qualification:

- Example: MOV CR
- Exit reason: “control register access”
- Exit qualification:
  - which CR



- Example: MOV CR
- Exit reason: “control register access”
- Exit qualification:
  - which CR
  - direction ( $R_x \rightarrow CR$  or  $CR \rightarrow R_x$ )

- Example: MOV CR
- Exit reason: “control register access”
- Exit qualification:
  - which CR
  - direction ( $R_x \rightarrow CR$  or  $CR \rightarrow R_x$ )
  - register used

# 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	
	RDTSC exiting		CR3-load exiting		CR3-store exiting	
	CR8-store exiting		Use TPR shadow		NMI-window exiting	
	Unconditional I/O exiting		Use I/O bitmaps		Monitor trap flag	
	MONITOR exiting		PAUSE exiting		Activate secondary controls	
Secondary processor-based VM-execution controls	Virtualize APIC accesses		Enable EPT		Descriptor-table exiting	
	Virtualize x2APIC mode		Enable VPID		WBINVD exiting	
	APIC-register virtualization		Virtual-interrupt delivery		PAUSE-loop exiting	
	RDRAND exiting		Enable INVPCID		Enable VM functions	
	Enable ENCLS exiting		RDSEED exiting		Enable PML	
	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 high 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		

The next step ( $\approx$  2005):

- Virtualization Hardware Extensions for Intel and AMD

The next step ( $\approx$  2005):

- Virtualization Hardware Extensions for Intel and AMD
- substantially lower overheads for VMs

The next step ( $\approx$  2005):

- Virtualization Hardware Extensions for Intel and AMD
  - substantially lower overheads for VMs
  - better isolation

The next step ( $\approx$  2005):

- Virtualization Hardware Extensions for Intel and AMD
  - substantially lower overheads for VMs
  - better isolation
  - IaaS VMs become widely used

- Support for interrupt-virtualization



- Support for interrupt-virtualization
  - VM-exit with every external interrupt (cannot be masked by guest)

- Support for interrupt-virtualization
  - VM-exit with every external interrupt (cannot be masked by guest)
  - VM-exit when guest-OS ready to accept interrupts (EFLAGS.IF==1)

- Support for interrupt-virtualization
  - VM-exit with every external interrupt (cannot be masked by guest)
  - VM-exit when guest-OS ready to accept interrupts (EFLAGS.IF==1)
- Support for CR0 and CR4-virtualization

- Support for interrupt-virtualization
  - VM-exit with every external interrupt (cannot be masked by guest)
  - VM-exit when guest-OS ready to accept interrupts (EFLAGS.IF==1)
- Support for CR0 and CR4-virtualization
  - VM-exit with any change of these registers

- Support for interrupt-virtualization
  - VM-exit with every external interrupt (cannot be masked by guest)
  - VM-exit when guest-OS ready to accept interrupts (EFLAGS.IF==1)
- Support for CR0 and CR4-virtualization
  - VM-exit with any change of these registers
  - can be set on which bits this shall happen

- Address Space Compression

- Address Space Compression
  - change of address space with any switch guest/hypervisor

- Address Space Compression
  - change of address space with any switch guest/hypervisor
  - guest owns full virtual address space



- Address Space Compression
  - change of address space with any switch guest/hypervisor
  - guest owns full virtual address space
- Ring Problems, SYSENTER/SYSEXIT

- Address Space Compression
  - change of address space with any switch guest/hypervisor
  - guest owns full virtual address space
- Ring Problems, SYSENTER/SYSEXIT
  - Guest can now run in ring 0

- Nonfaulting Access to Privileged State

- Nonfaulting Access to Privileged State
  - access raise fault into hypervisor

- Nonfaulting Access to Privileged State
  - access raise fault into hypervisor
- Hidden State

- Nonfaulting Access to Privileged State
  - access raise fault into hypervisor
- Hidden State
  - Saved into VMCS

- Hypervisor uses virtual memory

- Hypervisor uses virtual memory
- guest OS uses virtual memory

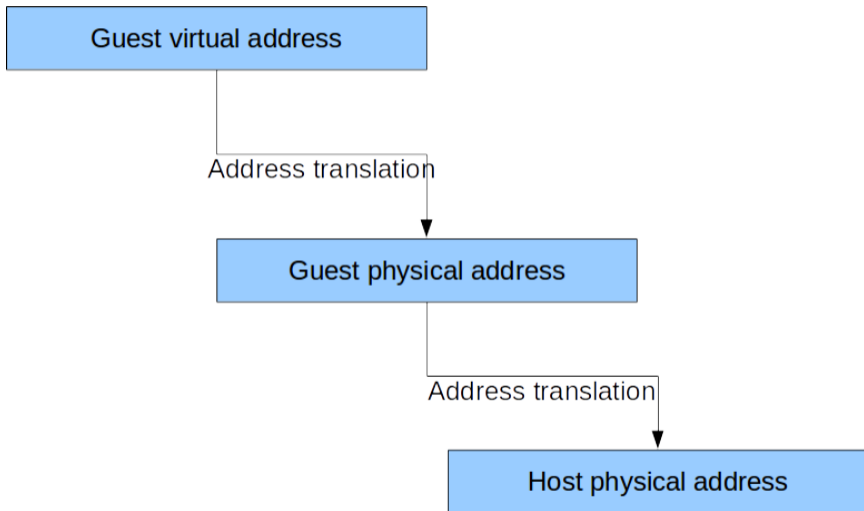


- Hypervisor uses virtual memory
- guest OS uses virtual memory
- hardware supports pagetables

- Hypervisor uses virtual memory
- guest OS uses virtual memory
- hardware supports pagetables
- how does this work?

- Hypervisor uses virtual memory
- guest OS uses virtual memory
- hardware supports pagetables
- how does this work?
  - shadow page tables

- Hypervisor uses virtual memory
- guest OS uses virtual memory
- hardware supports pagetables
- how does this work?
  - shadow page tables
  - hardware support

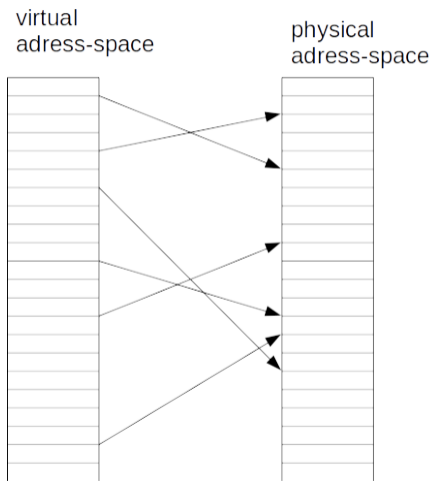


All problems in computer science can be solved by another level of indirection.

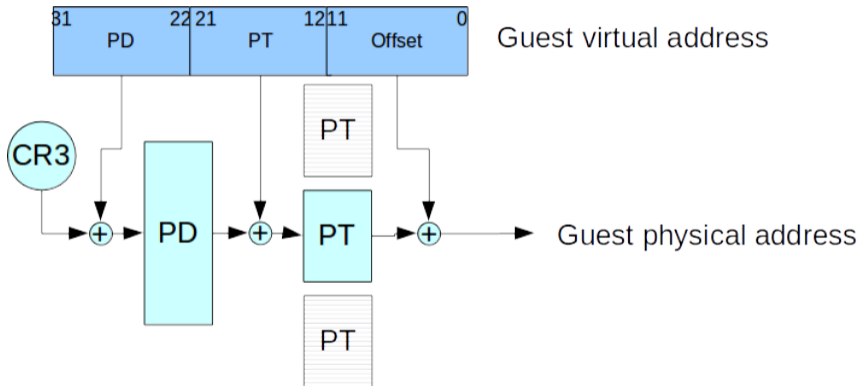
All problems in computer science can be solved by another level of indirection.

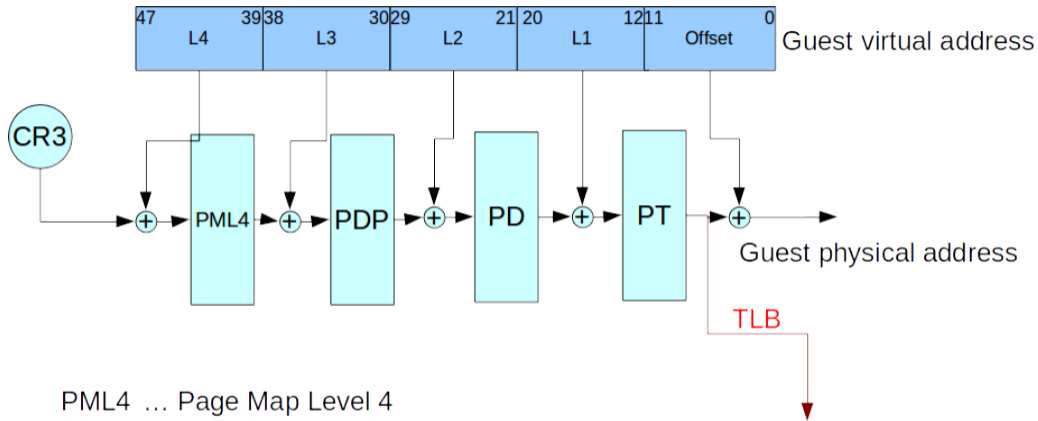
But that usually will create another problem.

David Wheeler

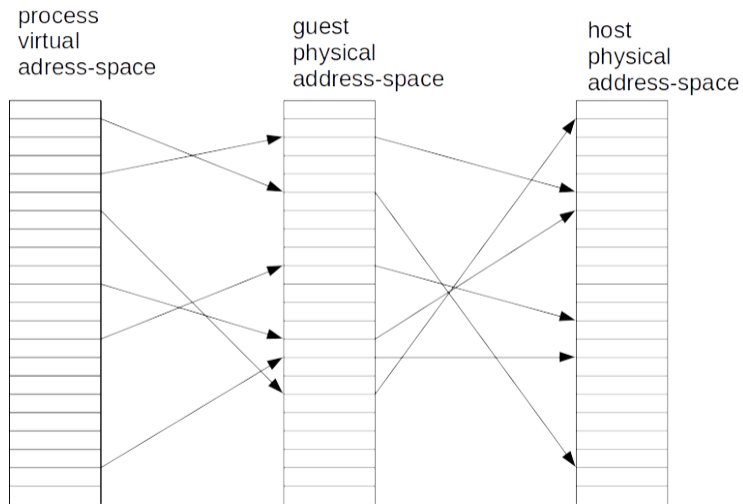


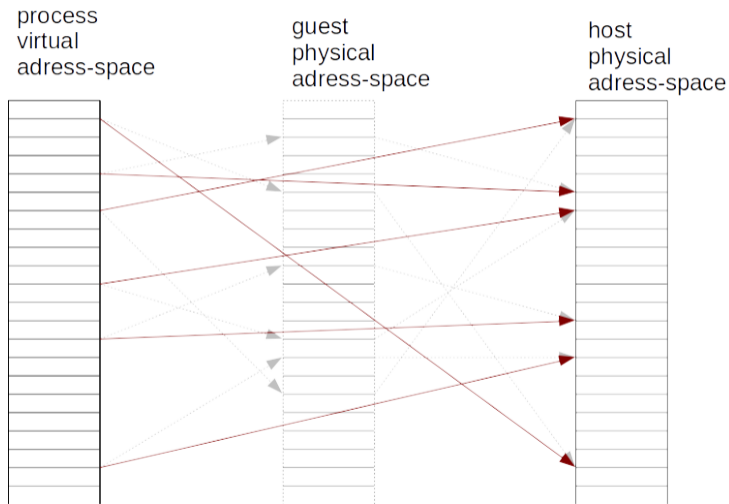


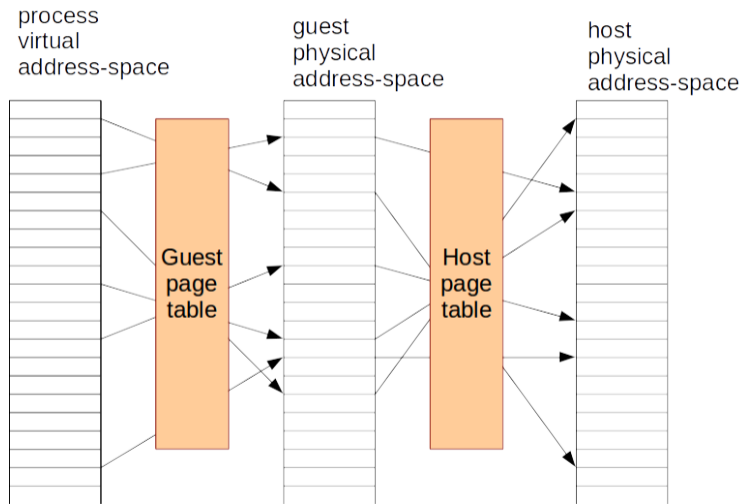


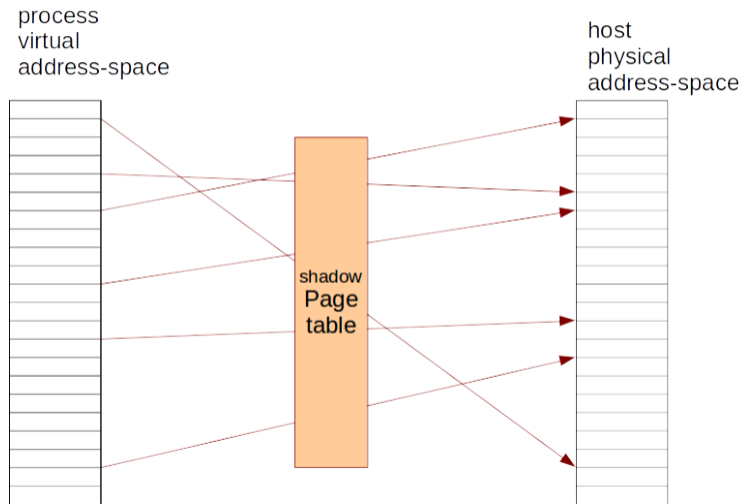


PML4 ... Page Map Level 4  
 PDP ... Page Directory Pointer  
 PD ... Page Directory  
 PT ... Page Table









- **merges** both page tables into one that the HW uses

- **merges** both page tables into one that the HW uses
- when guest changes own page table



- **merges** both page tables into one that the HW uses
- when guest changes own page table
  - Hypervisor has to catch access

- **merges** both page tables into one that the HW uses
- when guest changes own page table
  - Hypervisor has to catch access
  - update shadow page table

- when HW changes shadow page table

- when HW changes shadow page table
- update guest PT

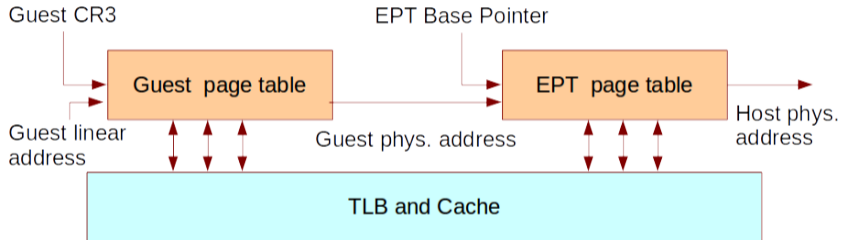
- when HW changes shadow page table
- update guest PT
  - expensive!

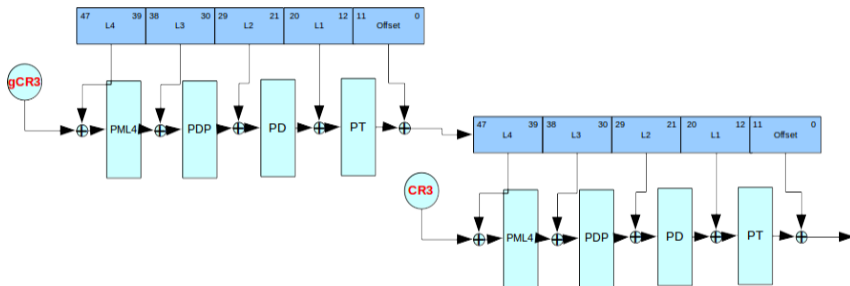
- when HW changes shadow page table
- update guest PT
  - expensive!
  - page faults caught by hypervisor

- when HW changes shadow page table
- update guest PT
  - expensive!
  - page faults caught by hypervisor
  - must run through guest PTs

- when HW changes shadow page table
- update guest PT
  - expensive!
  - page faults caught by hypervisor
  - must run through guest PTs
  - must emulate accessed and modified bits for guest



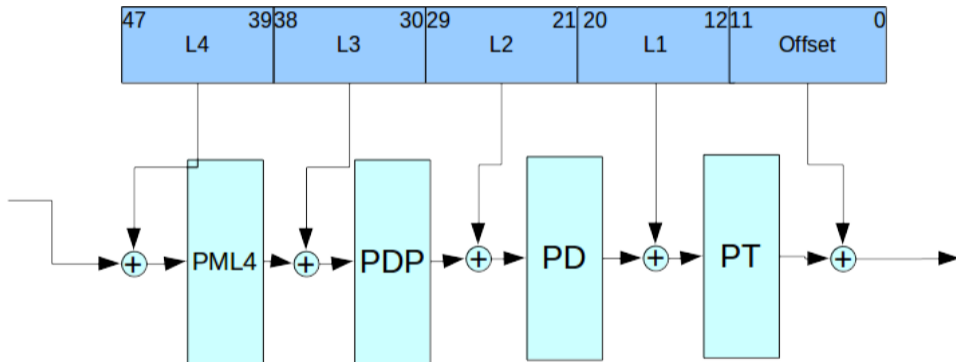


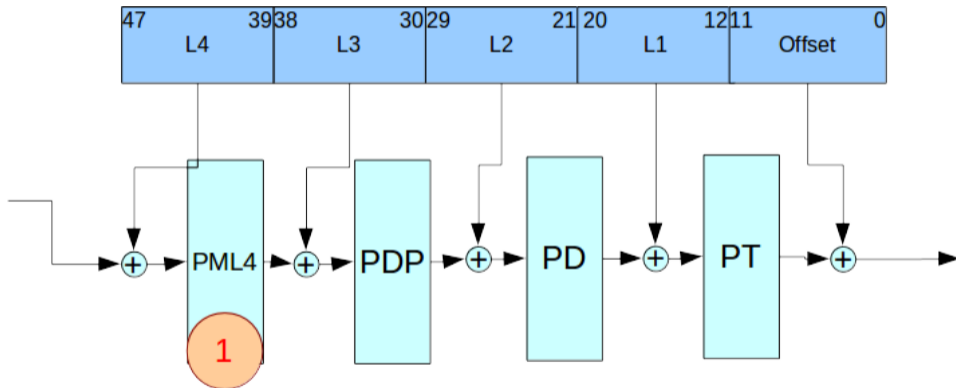


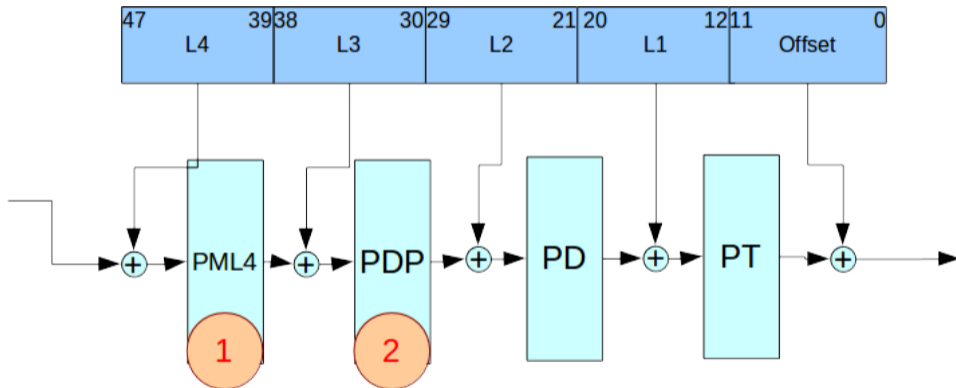
“guest page walk”

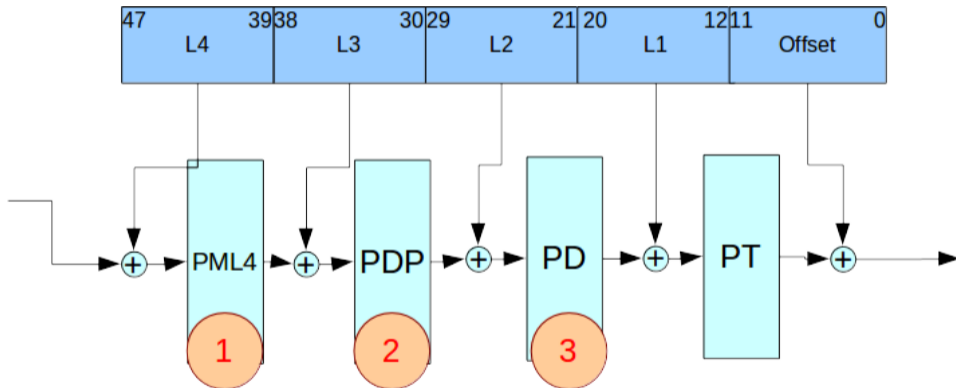
- lots of memory accesses....

- lots of memory accesses....
- but how many exactly?

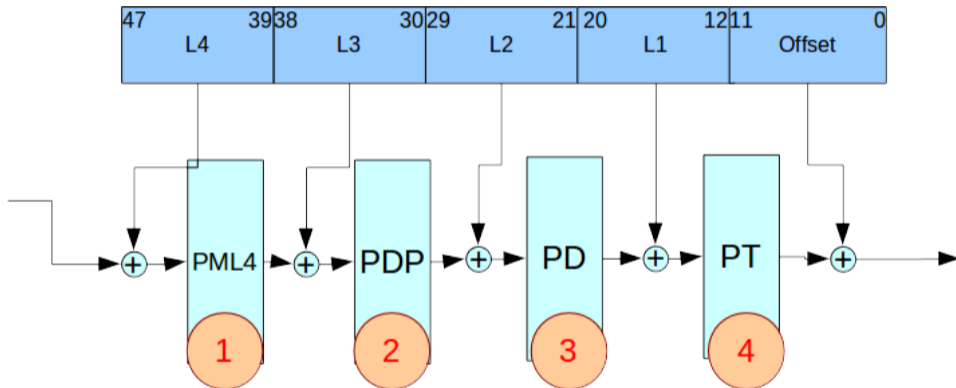


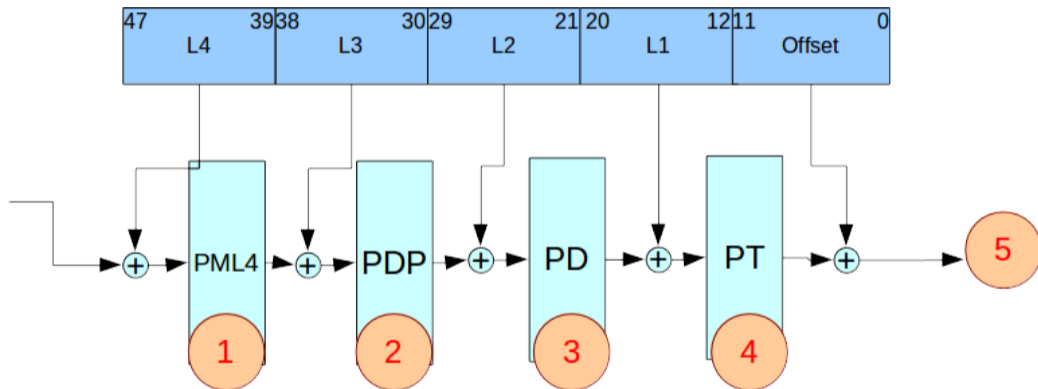


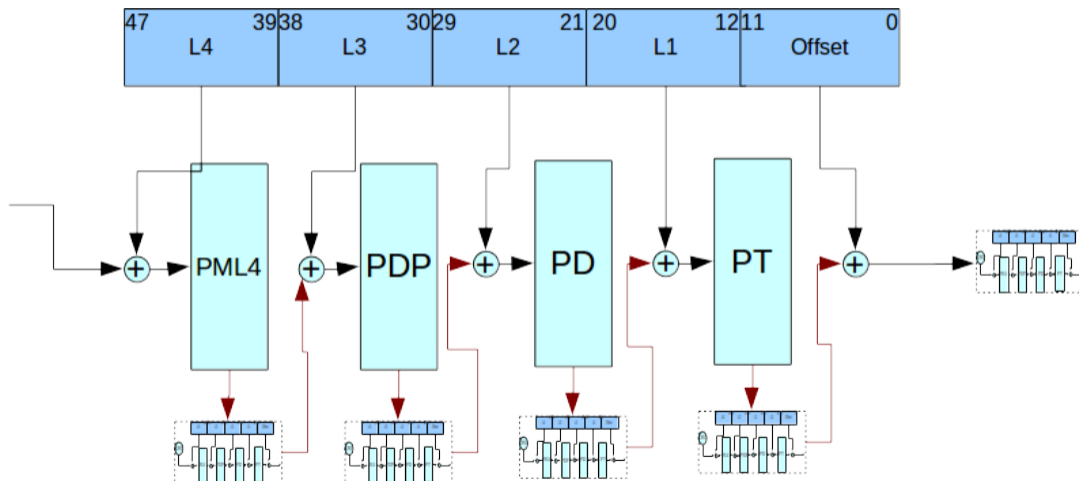












max. number of memory accesses per address translation

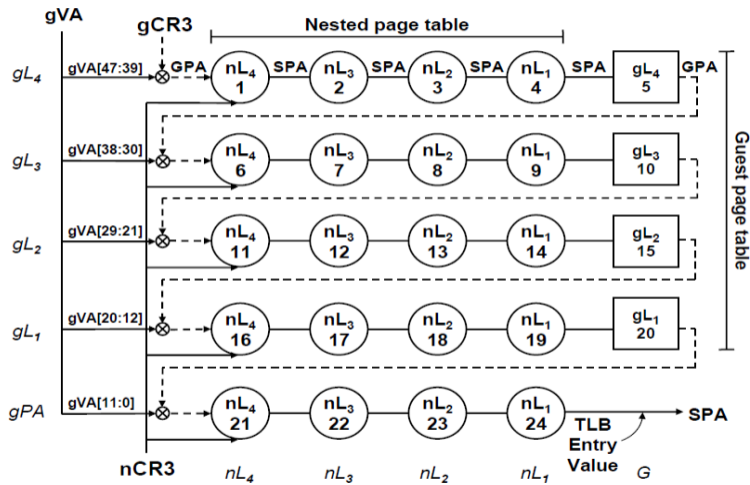
- 5 on guest level

max. number of memory accesses per address translation

- 5 on guest level
- each induces 5 on host level

max. number of memory accesses per address translation

- 5 on guest level
- each induces 5 on host level
- makes 25!



- depending on application: 3.9-4.6 times slower



- depending on application: 3.9-4.6 times slower
- but: TLB

- EPT only used if VM active

- EPT only used if VM active
- Translations tagged in TLB with EPT-basepointer

- EPT only used if VM active
- Translations tagged in TLB with EPT-basepointer
  - differentiate TLB-entries of different VMs

- EPT only used if VM active
- Translations tagged in TLB with EPT-basepointer
  - differentiate TLB-entries of different VMs
  - TLB-flush per guest possible

- EPT only used if VM active
- Translations tagged in TLB with EPT-basepointer
  - differentiate TLB-entries of different VMs
  - TLB-flush per guest possible
- VPID: virtual processor ID

- EPT only used if VM active
- Translations tagged in TLB with EPT-basepointer
  - differentiate TLB-entries of different VMs
  - TLB-flush per guest possible
- VPID: virtual processor ID
  - unique value for each VM

- EPT only used if VM active
- Translations tagged in TLB with EPT-basepointer
  - differentiate TLB-entries of different VMs
  - TLB-flush per guest possible
- VPID: virtual processor ID
  - unique value for each VM
  - translations tagged in TLB using VPID





1. Enable VMX via CR4

1. Enable VMX via CR4
2. Allocate a VMXON region and use the VMXON instruction

1. Enable VMX via CR4
2. Allocate a VMXON region and use the VMXON instruction
3. Allocate an MSR Bitmap region (we don't want a trap for all MSRs)

1. Enable VMX via CR4
2. Allocate a VMXON region and use the VMXON instruction
3. Allocate an MSR Bitmap region (we don't want a trap for all MSRs)
4. Use VMCLEAR instruction

1. Enable VMX via CR4
2. Allocate a VMXON region and use the VMXON instruction
3. Allocate an MSR Bitmap region (we don't want a trap for all MSRs)
4. Use VMCLEAR instruction
5. Execute VMPTRLD to make a VMCS the “current VMCS”

1. Enable VMX via CR4
2. Allocate a VMXON region and use the VMXON instruction
3. Allocate an MSR Bitmap region (we don't want a trap for all MSRs)
4. Use VMCLEAR instruction
5. Execute VMPTRLD to make a VMCS the “current VMCS”
6. Allocate a VMCS region and set up the VMCS (using VMWRITES)

1. Enable VMX via CR4
2. Allocate a VMXON region and use the VMXON instruction
3. Allocate an MSR Bitmap region (we don't want a trap for all MSRs)
4. Use VMCLEAR instruction
5. Execute VMPTRLD to make a VMCS the “current VMCS”
6. Allocate a VMCS region and set up the VMCS (using VMWRITES)
7. Use the VMLAUNCH



Similar problem as with Userspace-Kernelspace Isolation:

1. user needs help for some operations (e.g., HW interaction)

Similar problem as with Userspace-Kernelspace Isolation:

1. user needs help for some operations (e.g., HW interaction)

→ can use a syscall!

Similar problem as with Userspace-Kernelspace Isolation:

1. user needs help for some operations (e.g., HW interaction)

→ can use a syscall!

2. What about VMs?

Similar problem as with Userspace-Kernelspace Isolation:

1. user needs help for some operations (e.g., HW interaction)

→ can use a syscall!

2. What about VMs?

3. Same concept different level:

Similar problem as with Userspace-Kernelspace Isolation:

1. user needs help for some operations (e.g., HW interaction)

→ can use a syscall!

2. What about VMs?

3. Same concept different level:

→ Hypercalls!

Similar problem as with Userspace-Kernelspace Isolation:

1. user needs help for some operations (e.g., HW interaction)

→ can use a syscall!

2. What about VMs?

3. Same concept different level:

→ Hypercalls!

Similar problem as with Userspace-Kernelspace Isolation:

1. user needs help for some operations (e.g., HW interaction)

→ can use a syscall!

2. What about VMs?

3. Same concept different level:

→ Hypercalls! via the `vmcall` instruction

## Optimization

- Full virtualization often not needed



## Optimization

- Full virtualization often not needed
- Serverless / Edge Computing (it's still a form of cloud computing)

## Optimization

- Full virtualization often not needed
- Serverless / Edge Computing (it's still a form of cloud computing)
- Virtualization is not for free → why not skip it and just use OS level isolation?

## Optimization

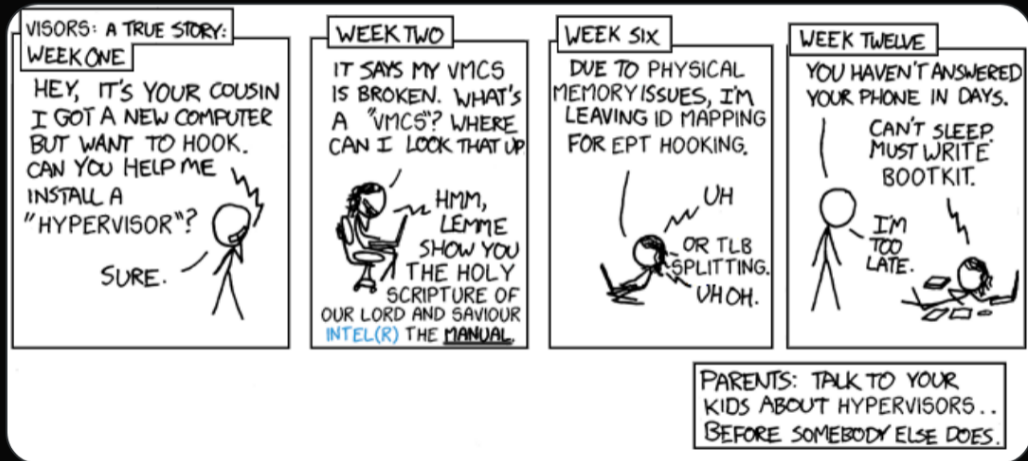
- Full virtualization often not needed
- Serverless / Edge Computing (it's still a form of cloud computing)
- Virtualization is not for free → why not skip it and just use OS level isolation?
- Context switches between processes are expensive → why not skip process isolation and just use language-level isolation?

Cloud Operating Systems → Hardware-assisted virtualization



cts @gf\_256 · 5. Apr. 2020

Talk to your kids about hypervisors...before someone else does



↻ 19

♡ 60



- Seminar-style

- Seminar-style
- You code

- Seminar-style
- You code
- You plan



- Seminar-style
- You code
- You plan
- You present

Daniel Gruss, Andreas Kogler, Fabian Rauscher

- 90P. = 100%

- 90P. = 100%
- 80P. → 1

- 90P. = 100%
- 80P. → 1
- 70P. → 2

- 90P. = 100%
- 80P. → 1
- 70P. → 2
- 60P. → 3

- 90P. = 100%
- 80P. → 1
- 70P. → 2
- 60P. → 3
- 50P. → 4

- 10 participants → 2-3 teams with each 4 participants (default)



- 10 participants → 2-3 teams with each 4 participants (default)
- 5 ECTS = 500h with 125h per team member

- 10 participants → 2-3 teams with each 4 participants (default)
- 5 ECTS = 500h with 125h per team member
- Team of 3? Same effort but +5 points

- 10 participants → 2-3 teams with each 4 participants (default)
- 5 ECTS = 500h with 125h per team member
- Team of 3? Same effort but +5 points
- Team of 2? Same effort but +10 points

- 10 participants → 2-3 teams with each 4 participants (default)
- 5 ECTS = 500h with 125h per team member
- Team of 3? Same effort but +5 points
- Team of 2? Same effort but +10 points

→ send us your registration until Friday March 11

- 24.3. Structure Setup  
Estimated Team Effort: 75h, Points: 2P.

After each deadline there is a Debriefing + Outlook meeting

- 24.3. Structure Setup  
Estimated Team Effort: 75h, Points: 2P.
- 7.4. Executing Guest Code  
Estimated Team Effort: 100h, Points: 4P.

After each deadline there is a Debriefing + Outlook meeting

- 24.3. Structure Setup  
Estimated Team Effort: 75h, Points: 2P.
- 7.4. Executing Guest Code  
Estimated Team Effort: 100h, Points: 4P.
- 5.5. Interrupt + Emulate PIC + Video Output + Feature Bidding 1  
Estimated Team Effort: 100h, Points: 4P.

After each deadline there is a Debriefing + Outlook meeting

- 24.3. Structure Setup  
Estimated Team Effort: 75h, Points: 2P.
- 7.4. Executing Guest Code  
Estimated Team Effort: 100h, Points: 4P.
- 5.5. Interrupt + Emulate PIC + Video Output + Feature Bidding 1  
Estimated Team Effort: 100h, Points: 4P.
- 26.5. Boot Guest SWEB Shell + Virtualize Disk + Feature Bidding 2  
Estimated Team Effort: 75h, Points: 35P.

After each deadline there is a Debriefing + Outlook meeting



- 24.3. Structure Setup  
Estimated Team Effort: 75h, Points: 2P.
- 7.4. Executing Guest Code  
Estimated Team Effort: 100h, Points: 4P.
- 5.5. Interrupt + Emulate PIC + Video Output + Feature Bidding 1  
Estimated Team Effort: 100h, Points: 4P.
- 26.5. Boot Guest SWEB Shell + Virtualize Disk + Feature Bidding 2  
Estimated Team Effort: 75h, Points: 35P.
- 9.6. Feature PoC in Booted Guest SWEB  
Estimated Team Effort: 75h, Points: 10P.

After each deadline there is a Debriefing + Outlook meeting

- 24.3. Structure Setup  
Estimated Team Effort: 75h, Points: 2P.
- 7.4. Executing Guest Code  
Estimated Team Effort: 100h, Points: 4P.
- 5.5. Interrupt + Emulate PIC + Video Output + Feature Bidding 1  
Estimated Team Effort: 100h, Points: 4P.
- 26.5. Boot Guest SWEB Shell + Virtualize Disk + Feature Bidding 2  
Estimated Team Effort: 75h, Points: 35P.
- 9.6. Feature PoC in Booted Guest SWEB  
Estimated Team Effort: 75h, Points: 10P.
- 23.6. Feature Implementation Done + Final Presentation and Demo in Booted Guest SWEB  
Estimated Team Effort: 75h, Points: 35P.

After each deadline there is a Debriefing + Outlook meeting

