Confidential Data Inside Encypted VMs

Dario Faggioli

Virtualization SW. Eng. @ SUSE

dfaggioli@opensuse.org

dariof

@DarioFaggioli

@dfaggioli:matrix.org

About Myself

- Living:
  - Empoli (FI)
  - about.me/dario.faggioli
- FLOSS
  - LUG: GOLEM … all the times I can ;-P
  - openSUSE: user & contributor
- Education
  - 2008, Ph.D, ReTiS Lab, Scuola Sant'Anna
    Real-Time System, Linux kernel scheduling
- Work
  - Since 2018, Virtualization Software Engineer @ SUSE
    Xen, Linux kernel, KVM, QEMU, Libvirt
    Working on scheduling, performance evaluation & tuning

Confidential Computing

- What?
  - Keep [your] data confidential ~= secret
- From whom?
  - From everyone!
- How?
  - Encryption
- When?
  - Always

Encrypted Data

When? ⇒ Always!

- when data are stored on a hard drive
  - disk encryption (dm_crypt, …)
  - available since long time
- when data transit over networks
  - secure network comm. protocols (TLS, VPNs, …)
  - available since long time

Encrypted Data

When? ⇒ Always

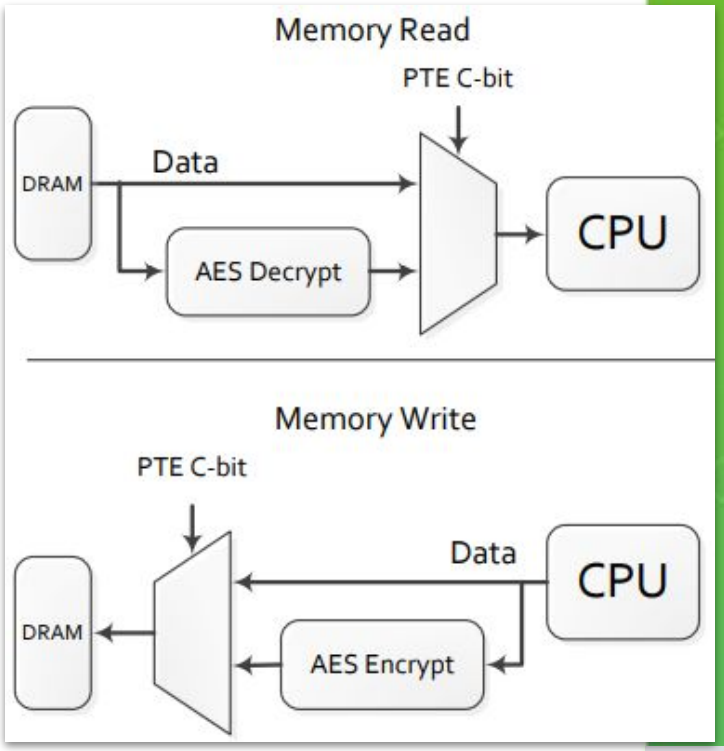
- while sitting in the PC memory (RAM) ?
  - Err… Mmm…
- while being processed on the CPU?
  - Mmm… Err...

New technologies implemented by various HW vendors to address this

Encrypting The Memory Live

In new enough (see later) AMD CPUs

- AMD Secure Processor
  - Additional ARM ® Cortex ® chip
  - Encrypt / Decrypt memory accesses on-the-fly
    - AES, 128 bit keys



- AMD MEMORY ENCRYPTION
- EXTENDING SECURE ENCRYPTED VIRTUALIZATION WITH SEV-ES
- AMD-SEV SNP: Strengthening VM Isolation with Integrity Protection and More

Encrypt Data in Memory / CPU

New technologies implemented by HW vendors

- AMD:
  - SME: Secure Memory Encryption
  - SEV: Secure Encripted Virtualization
  - SNP: Secure Nested Paging
- Intel: SGX, TDX
- IBM
- ARM

Encrypt Data in Memory / CPU

AMD, https://developer.amd.com/sev/

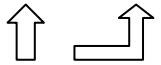
- SME: Secure Memory Encryption
  - Memory can be encrypted (All of it / only part of it)
  - Only one encryption key
    - [Re]Generated at boot by the Secure Processor (SP)
  - protect “only” from cold boot attacks
- SEV: Secure Encripted Virtualization
  - Memory of the VMs can be encrypted
  - Different encryption keys, e.g., 1 per VM, managed by SP
    - VMs isolated from the hypervisor
    - VMs isolated from one another
- SNP: Secure Nested Paging
  - Implements memory (pages) ownership
  - Protect agains (malicious hypervisor) remap and replay attacks

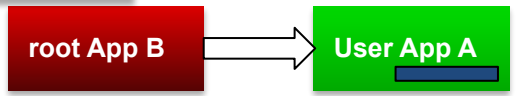
Reading Data in Memory

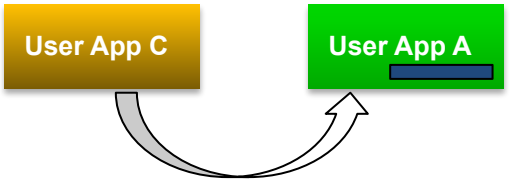
- **Green User**'s App A: 

  - Super Secret Password: 
  - Password in A's memory: 

- Root App B: 

- Legit access to memory: 
  - e.g., A accesses its own mem: 

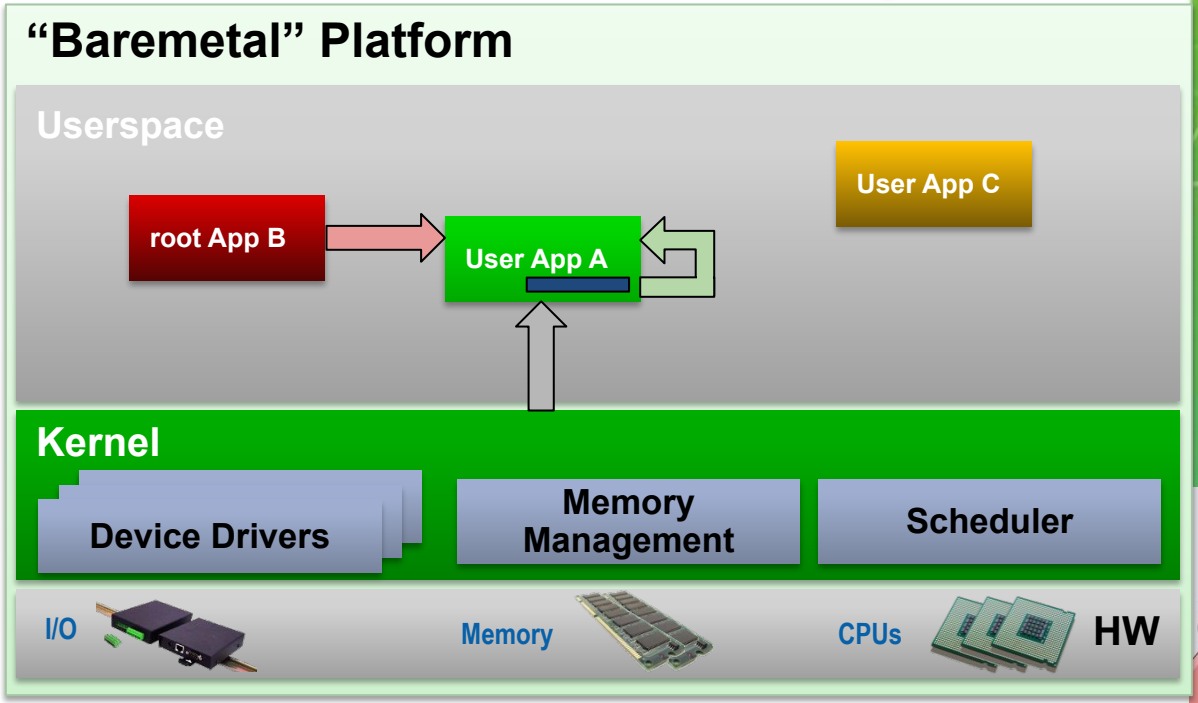
  - e.g., root's B accesses A's mem: 

- "Rogue" access to memory: 
  - e.g., non-root **Yellow User**'s App C reads Green's A's mem: 

Reading Data in Memory

“Baremetal” System
Simple, usual: HW -- OS [kernel] -- Apps

- A can read it’s own memory
- C can’t read A’s memory
- B is `root`, can read A’s memory
- kernel can read A’s memory



Non-Live Demo

```
$ cat pippo
#!/bin/bash
while true ; do
        clear
        echo "PID: $$"
        read -s -p "Input the Super Secret Password: " \
          SUPER_SECRET_PASSWORD; echo
        sleep 30
        echo "Super Secret Password is: $SUPER_SECRET_PASSWORD"
done
```

Print its own PID (for convenience)

Ask for password (and store in plaintext :-O)

After a little while, print the password ad go back to the beginning

Non-Live Demo

- As user (just start `./pippo`, don't type anything else yet!):

```
$ ./pippo
PID: 100894
Input the Super Secret Password:
```

- While `./pippo` wait for me to type a password, as `root`:

```
# gcore -a 100894
# grep -a tumbleweed core.100894
#
```

Non-Live Demo

- Let’s type “tumbleweed” (our super secure password! :-P)

```
$ ./pippo
PID: 100894
Input the Super Secret Password:**********
```

- … And let’s scan again:

```
# gcore 100894
# grep -a tumbleweed core.100894
tumbleweed
tumbleweed
#
```

What Just Happened ?

What just happened:

- We started a program (./pippo), as normal user
- We dumped (with `gcore -a`) & scanned (with `grep -a`) its memory (program `./pippo`, PID 100894) before any password was typed
- We looked for the string "`tumbleweed`" in the dump
- We found nothing
- We type "`tumbleweed`", as the password[*]
- We scan again
- We find the password in the process' memory

All normal!

[*] trivial example, as password is stored in plaintext, but still...

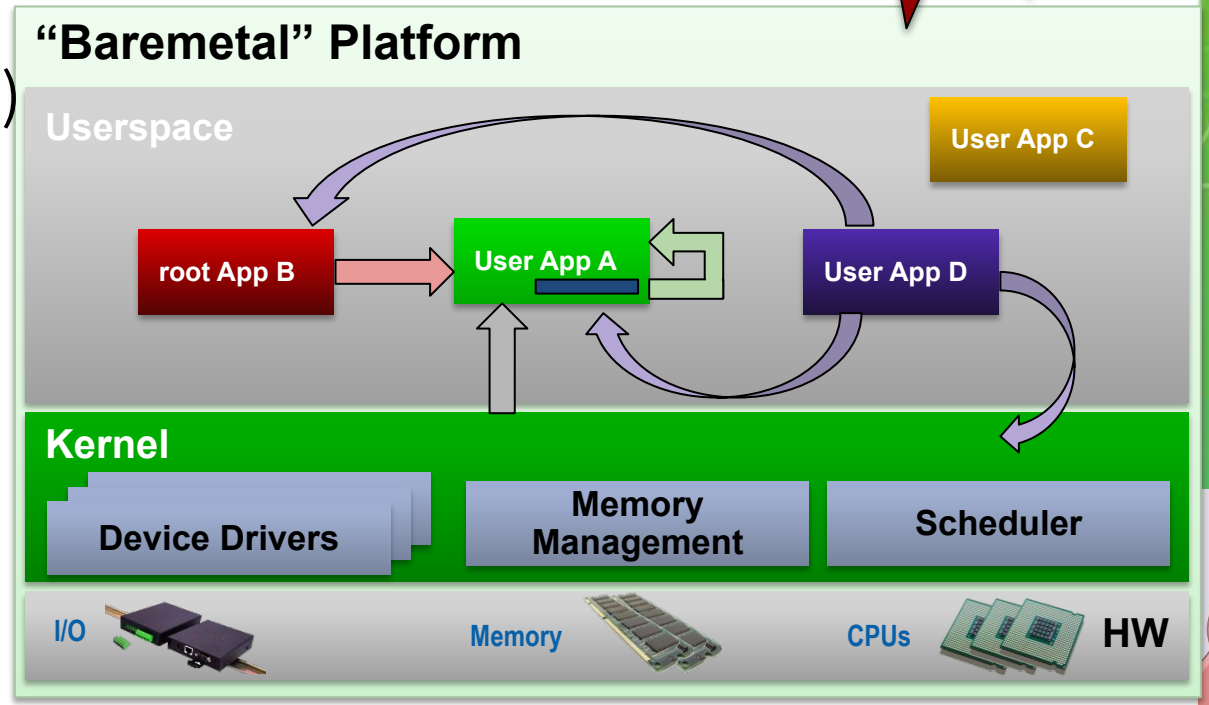
Reading Data in Memory

We're under ATTACK !!!

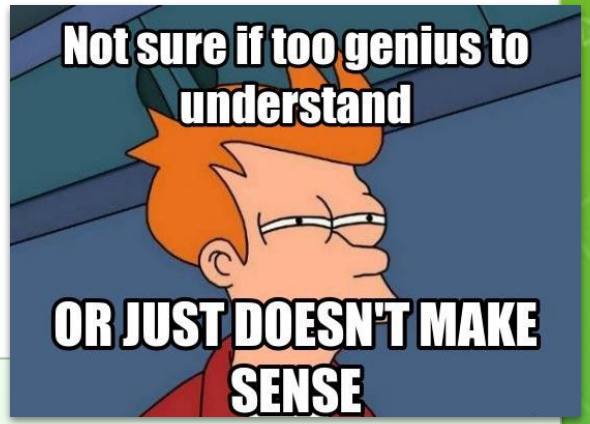
"Baremetal" System
Simple, usual: HW -- OS [kernel] -- Apps

- Arrive D, evil! (compromised?)
  - D attacks A
  - D attacks root
  - D attacks the kernel

That's what security is for! :-)



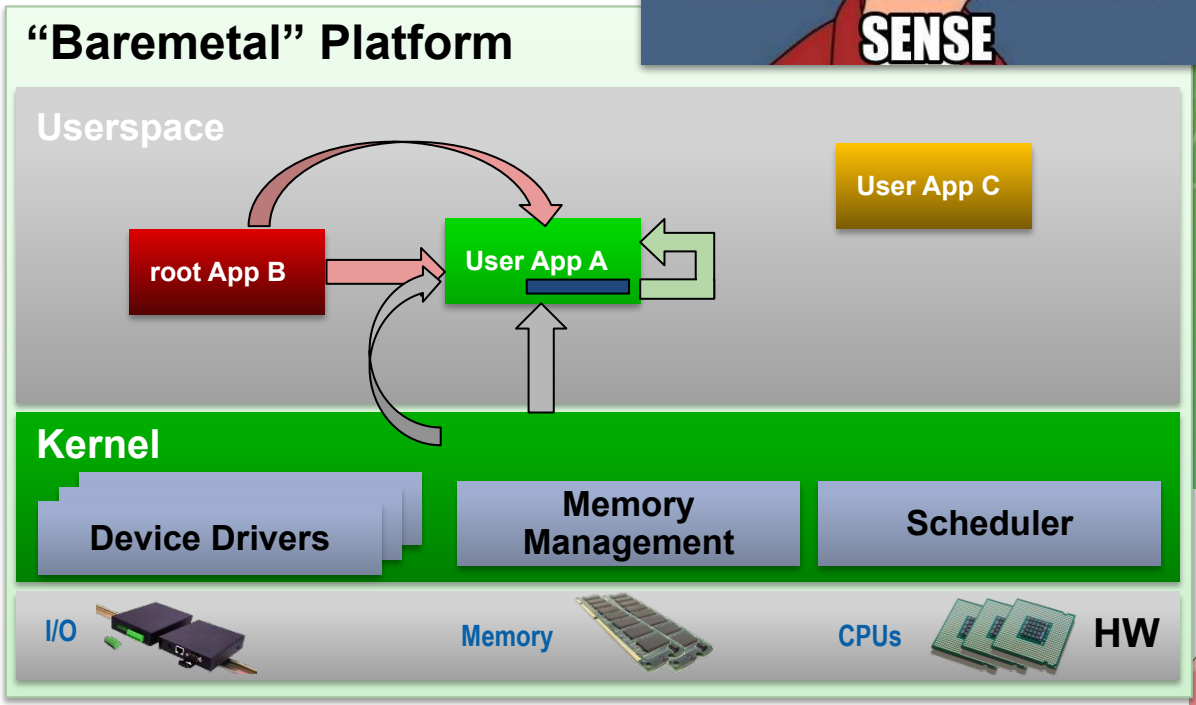
Reading Data in Memory



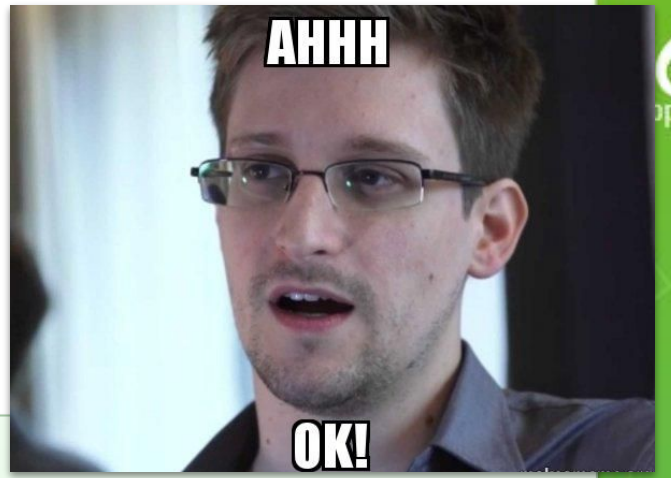
“Baremetal” System
Simple, usual: HW -- OS [kernel] -- Apps

- What if:
  - `root` attacks A
  - the kernel attacks A

How does this even make sense?



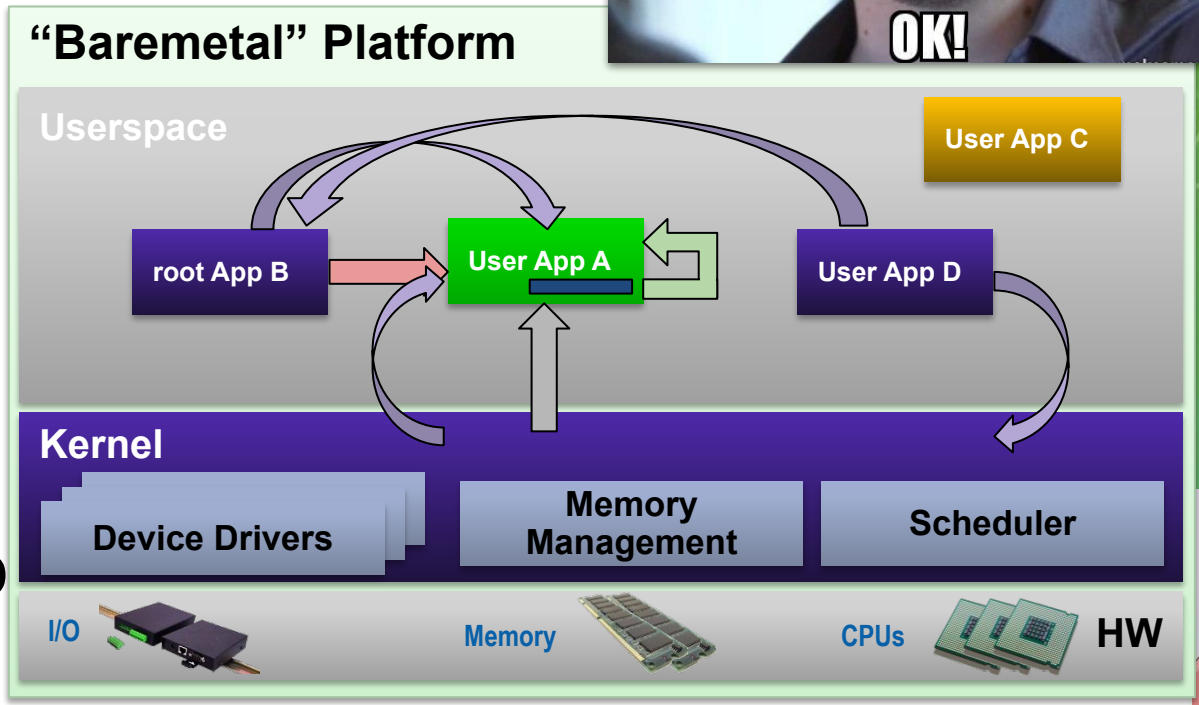
Reading Data in Memory



“Baremetal” System
Simple, usual: HW -- OS [kernel] -- Apps

- What if:
  - `root` attacks A
  - the kernel attacks A

Maybe because compromised by D



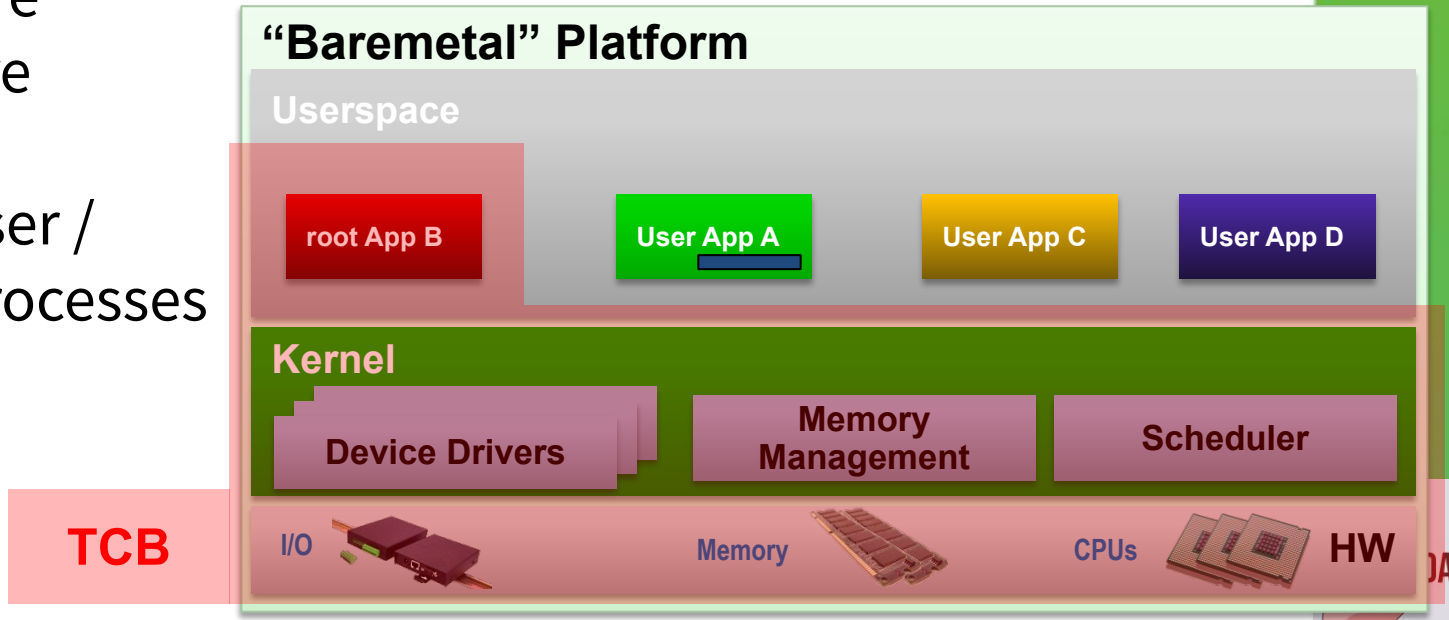
Trusted Computing Base (TCB)

"The trusted computing base (TCB) of a computer system is the set of all hardware, firmware, and/or software components that are critical to its security, in the sense that bugs or vulnerabilities occurring inside the TCB might jeopardize the security properties of the entire system."

- It's "the good guys"
- If even 1 piece of the TCB is:
  - malicious
  - compromised
- No point fighting any longer, **we lost** !!

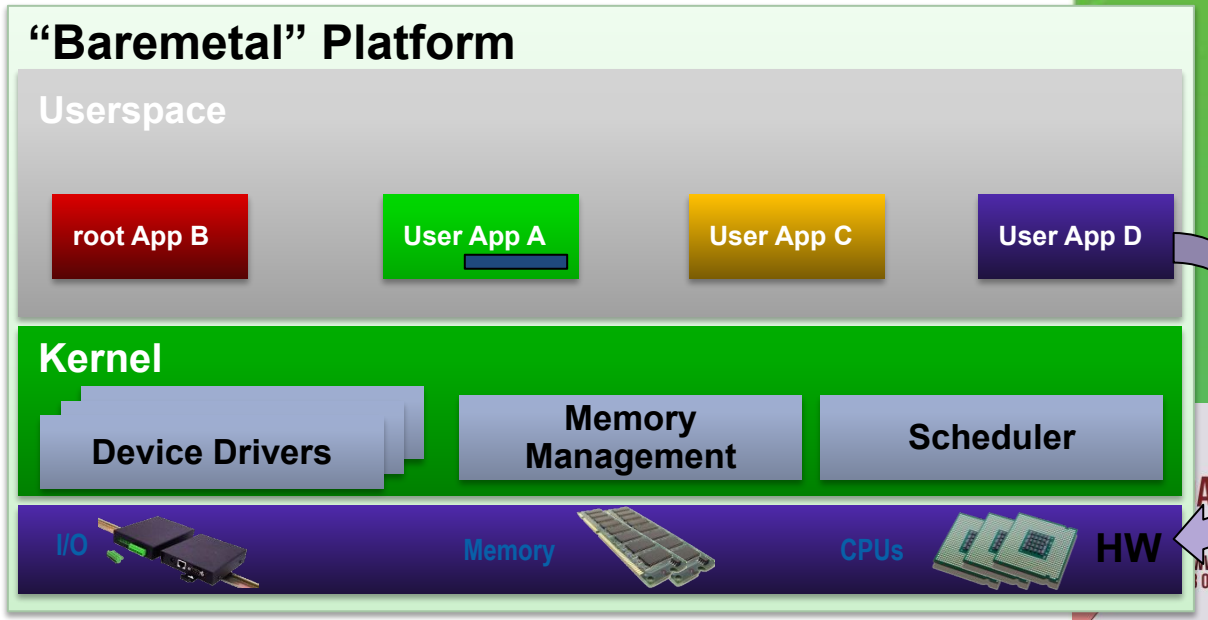
Trusted Computing Base (TCB)

- The smaller, the better
- On a baremetal system:
  - Hardware
  - Firmware
  - Kernel
  - `root` user / `root` processes



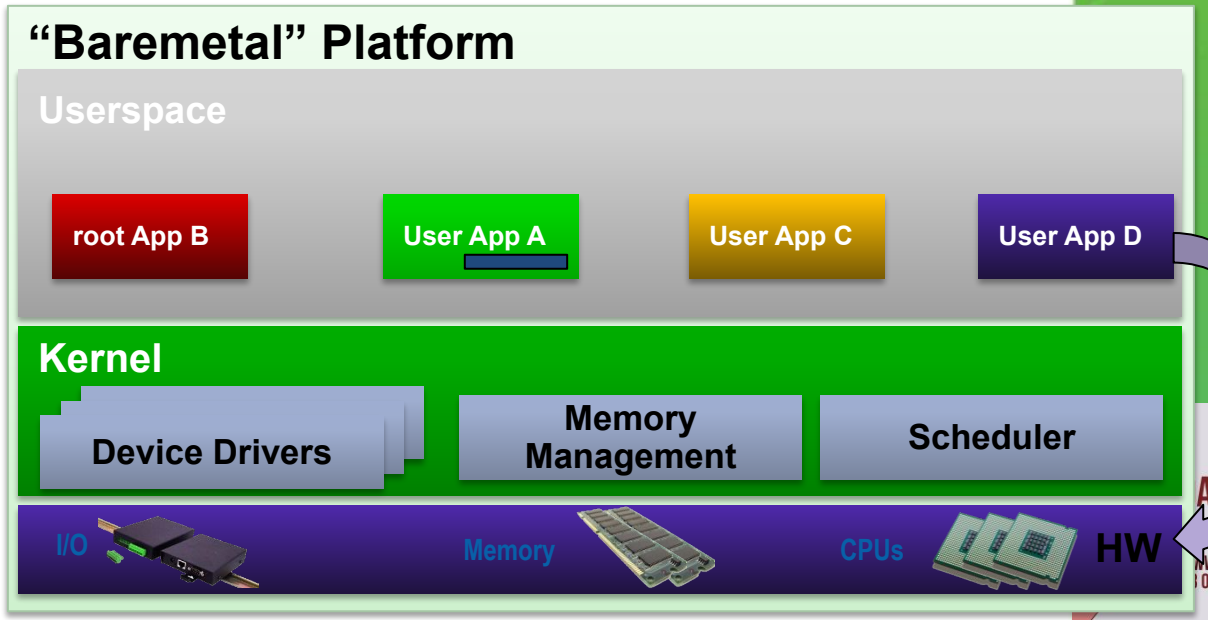
Trusted Computing Base (TCB)

- On a baremetal system:
  - Hardware
    - D wins hardware (e.g., bugs that can lead to HW crashes)



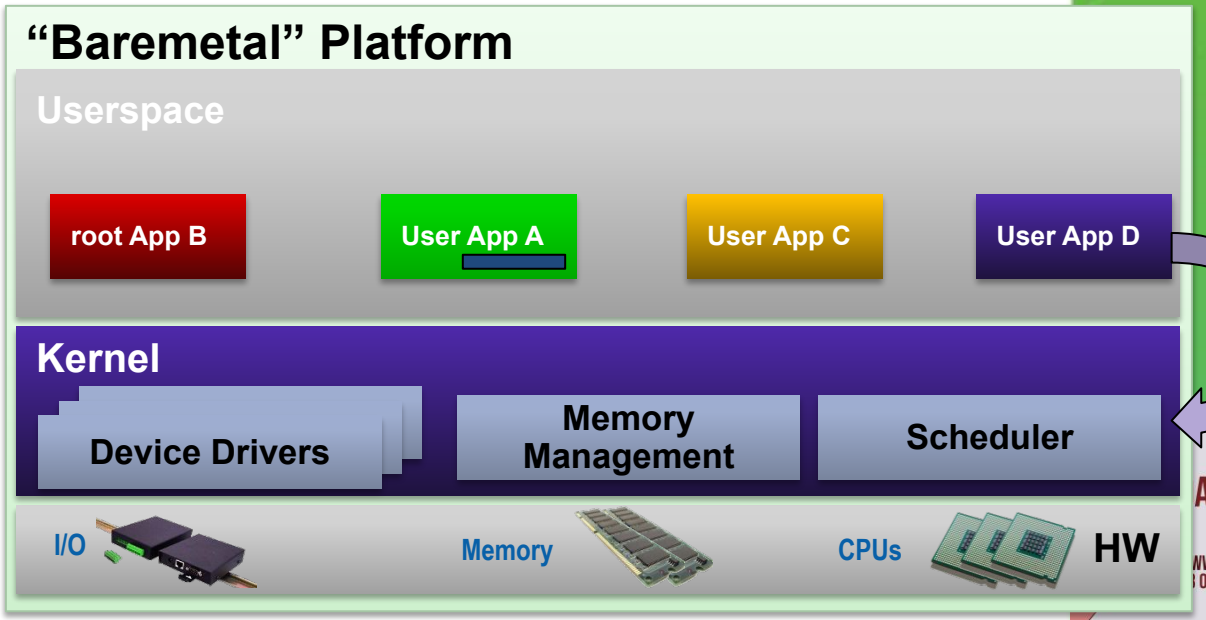
Trusted Computing Base (TCB)

- On a baremetal system:
  - Firmware
    - D win's firmware (e.g., Firmware backdoors)



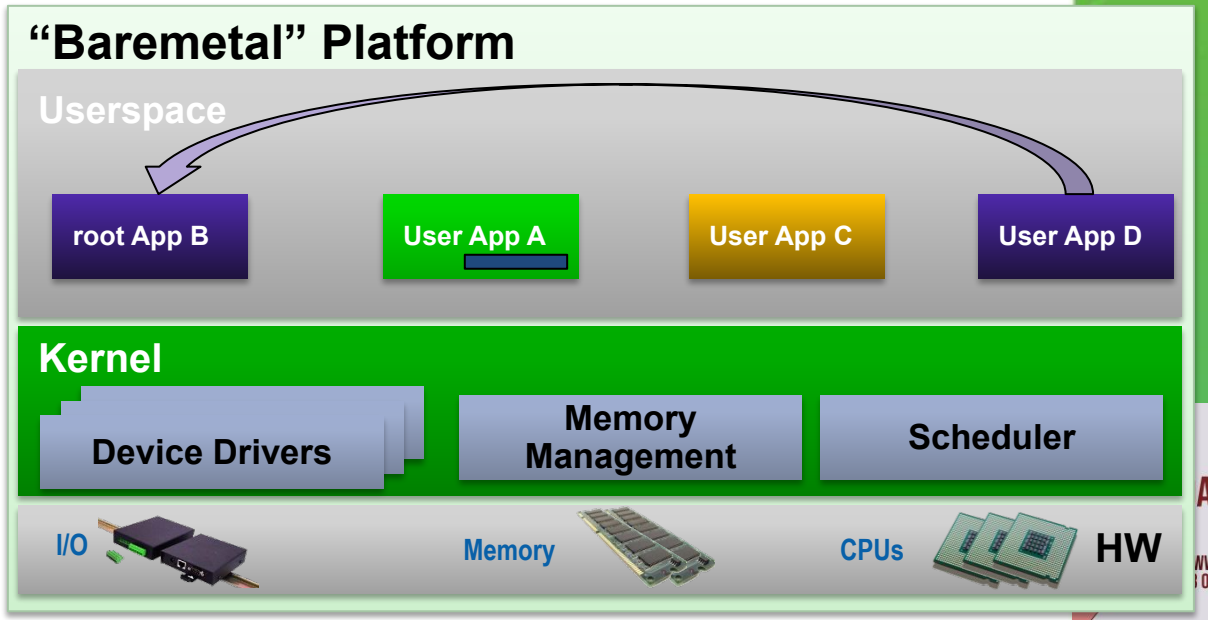
Trusted Computing Base (TCB)

- On a baremetal system:
  - Kernel
    - D win's the Kernel (e.g., Kernel/driver bugs)



Trusted Computing Base (TCB)

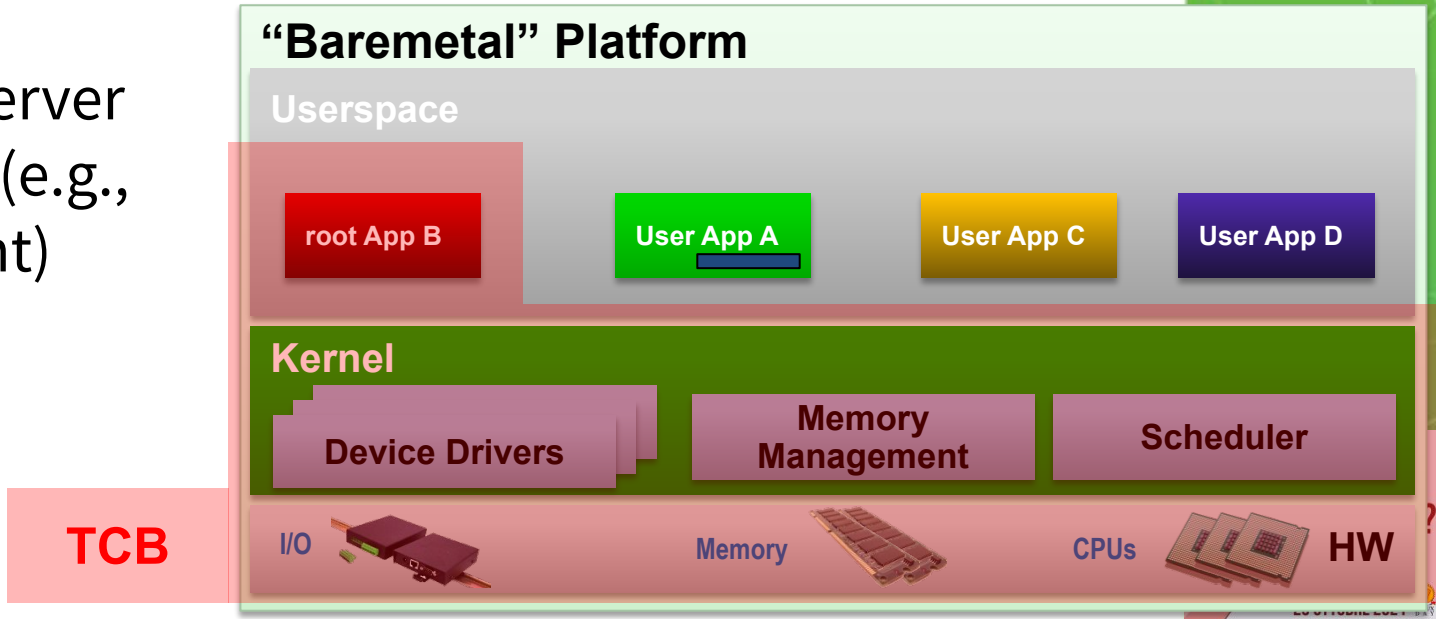
- On a baremetal system:
  - `root` user / root processes
    - D win's becomes root (e.g., priv. escalation in system daemons)



Who Owns The TCB ?

Who is in charge of the components of the TCB

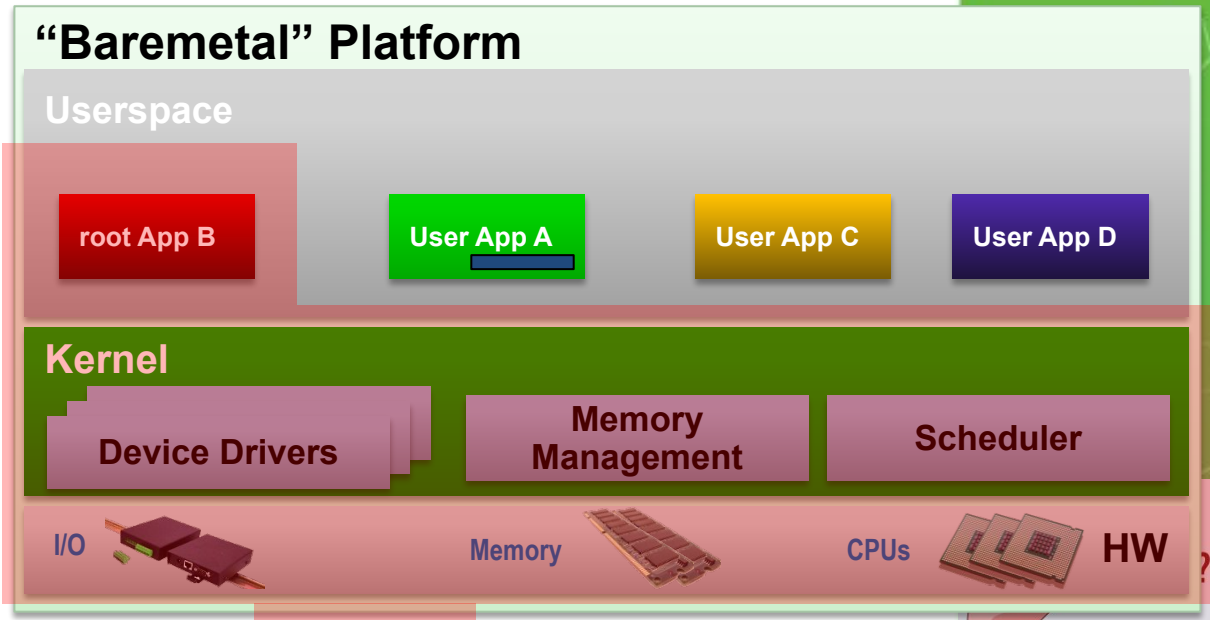
- Personal baremetal box/server
  - Me
- My company's baremetal box/server
  - My company (e.g., IT department)



Whose Data Are Stolen ?

In case of breach in the TCB, what can the attacker steal?

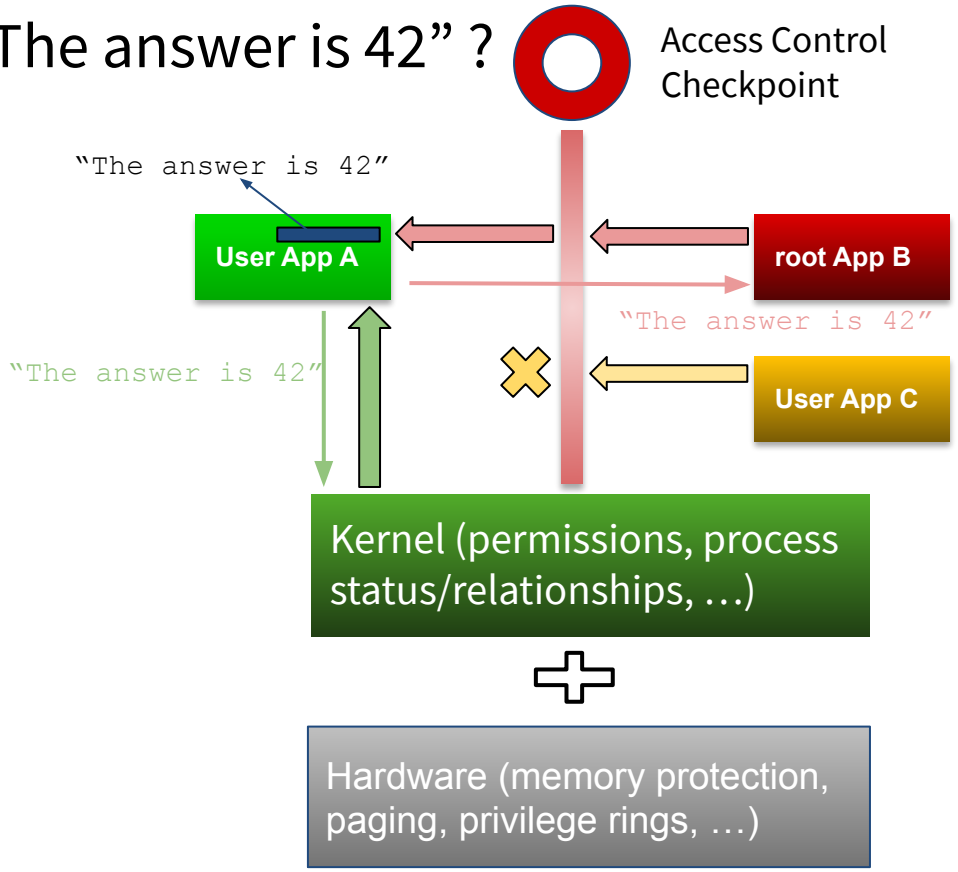
- Personal baremetal box/server
  - **My data**
  - E.g.,passwords, credit cards, health, digital ID, ...
- My company's baremetal box/server
  - **My company**'s data
  - E.g., financials, industrial secrets, ...



Enforcing Memory Access Control: Baremetal

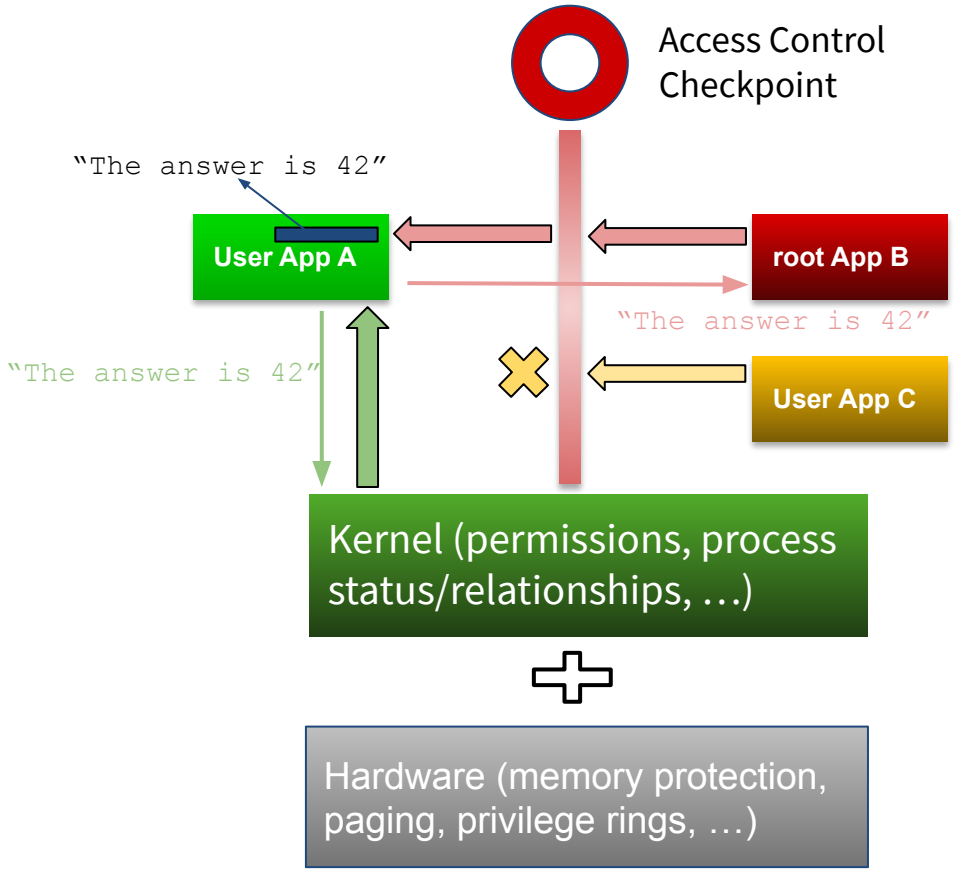
Who can read the secret “The answer is 42” ?

- Kernel (i.e., software) enforces the access control
- With help from paging (i.e., hardware)



Enforcing Memory Access Control: Baremetal

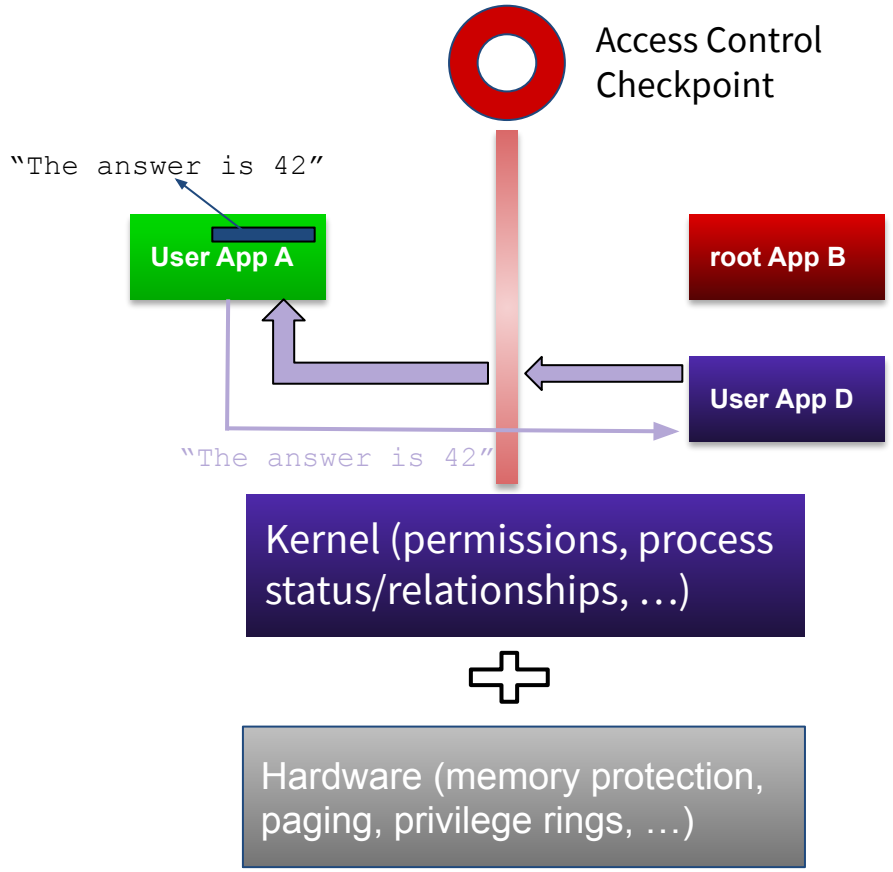
If either one fails...



Enforcing Memory Access Control: Baremetal

If either one fails...

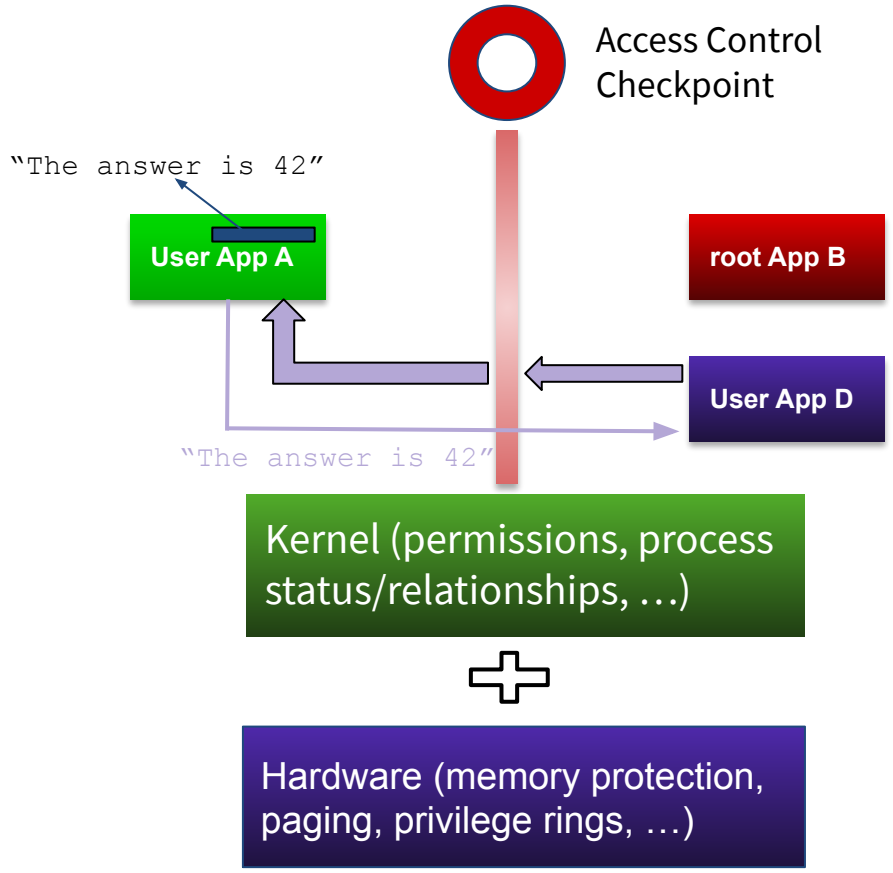
- e.g., exploited Kernel bug



Enforcing Memory Access Control: Baremetal

If either one fails...

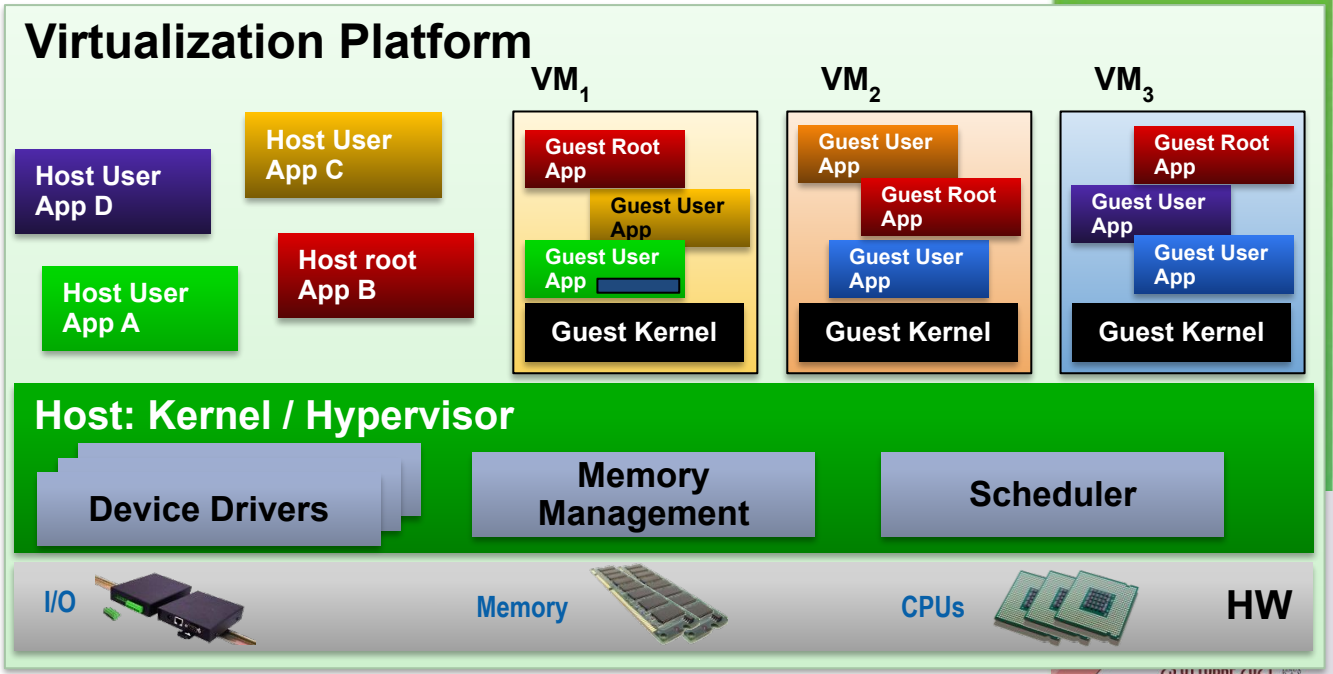
- e.g., exploited HW bug
  - see Spectre, Meltdown & Friends!



Enters Virtualization

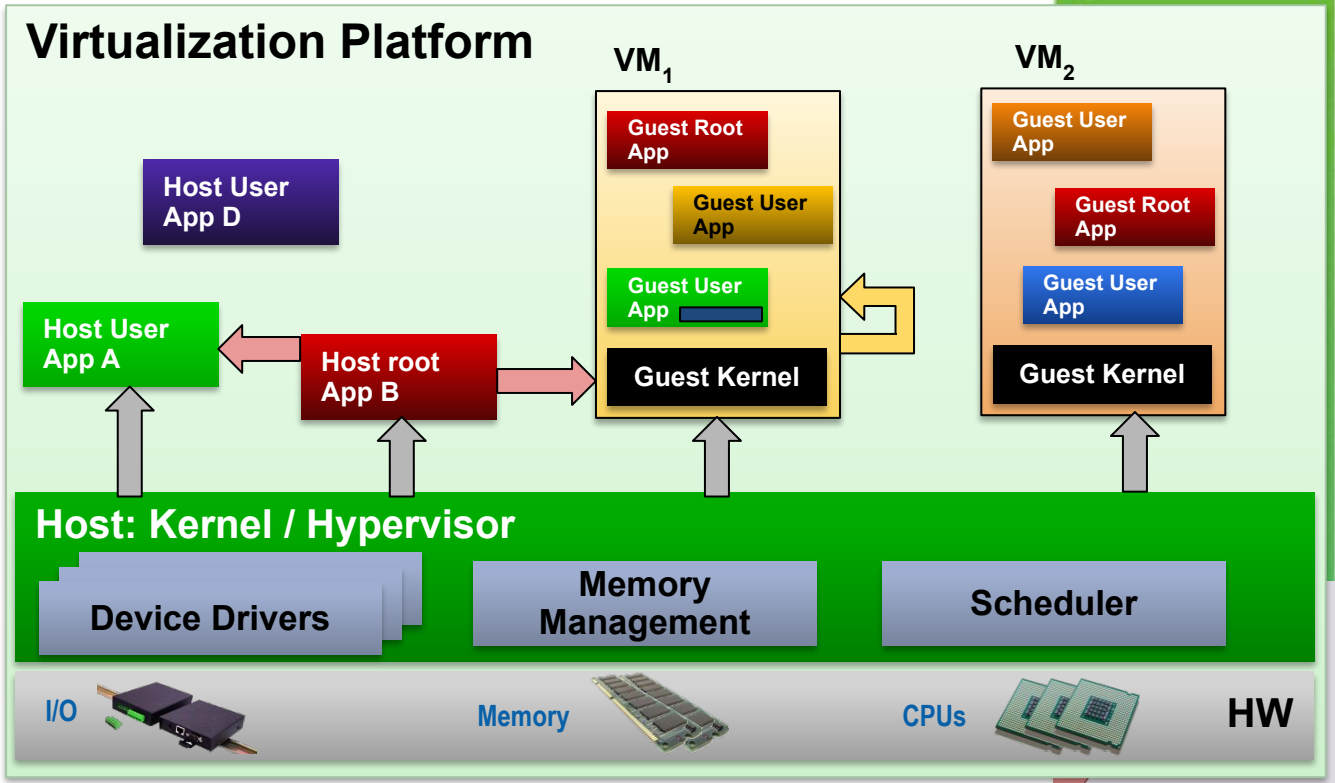
- What was baremetal ⇒ Host
  - Host Apps (root & user) are still there

- Kernel += Hypervisor
  - VMs (= Guests)
    - VM hardware (virtual / emulated)
    - VM Kernel
    - VM user / `root` Apps



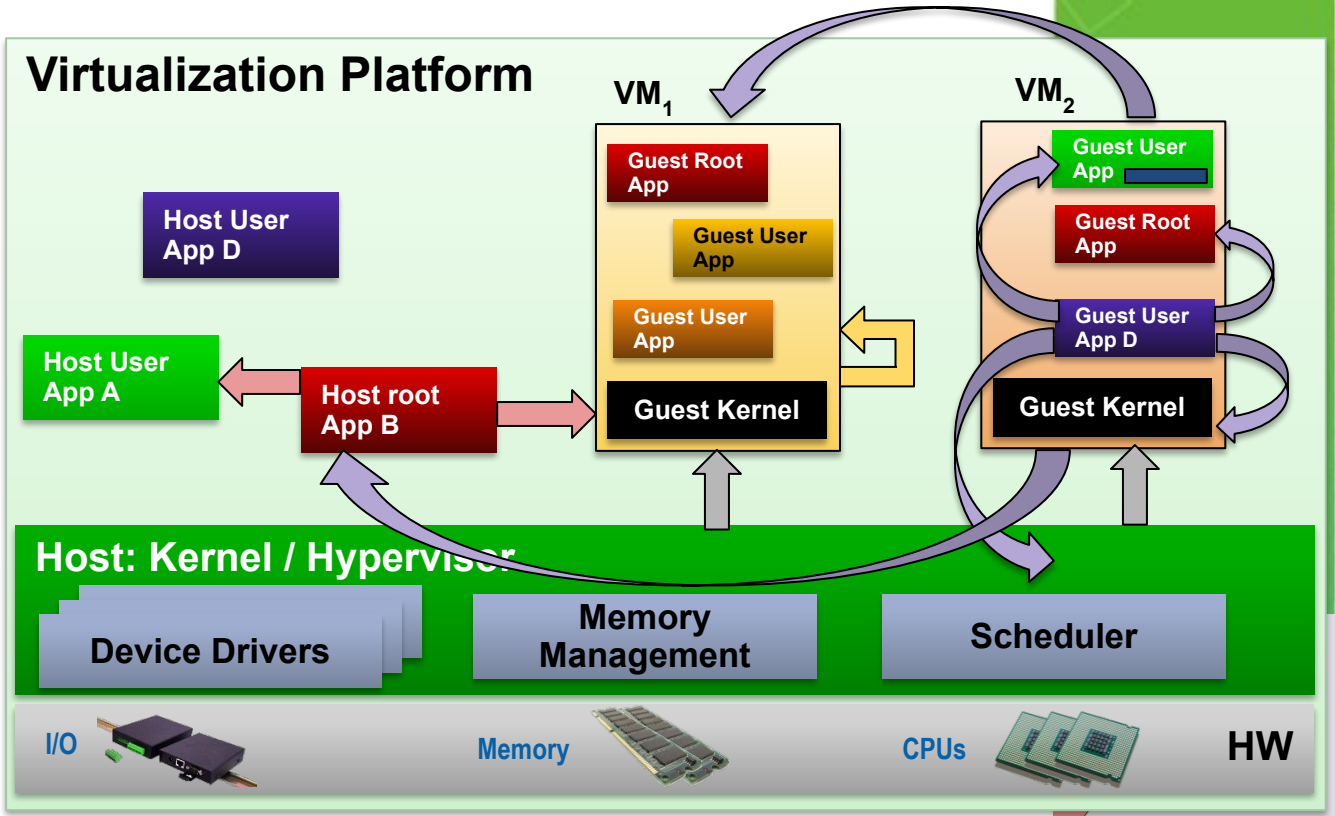
Virtualization: Legit Operations

- Inside a VM:
  - same as on baremetal
- Host ⇔ VMs
  - Kernel/Hypervisor reads all Host Apps' and VMs' memory
  - Host root Apps reads all Host Apps' and VMs' memory

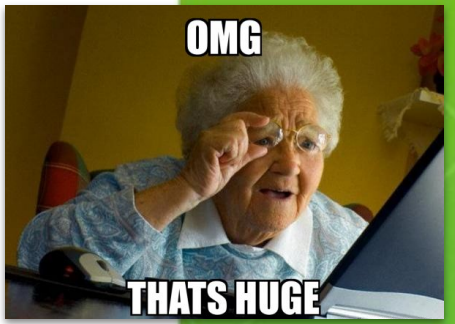


Virtualization: Attacks

- A Lot !!!
- VM2 Guest (evil) User App D can attack:
  - VM2 Guest Apps (`root` and user)
  - VM2 Guest Kernel
  - Host Kernel
  - Other VMs
  - Host Apps (`root` and user)

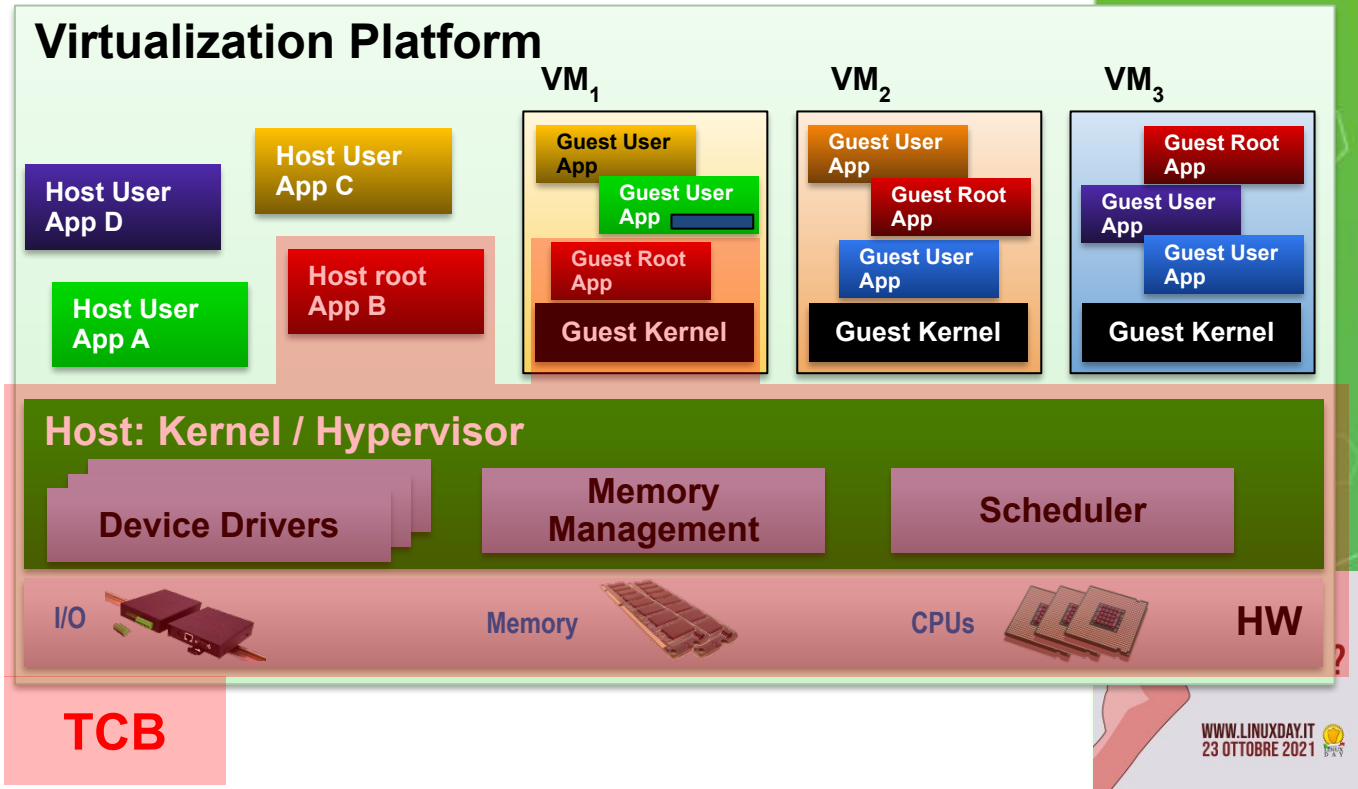


Virtualization: TCB



From the point of view of Guest User App A, in VM1 (where the sensitive data are)

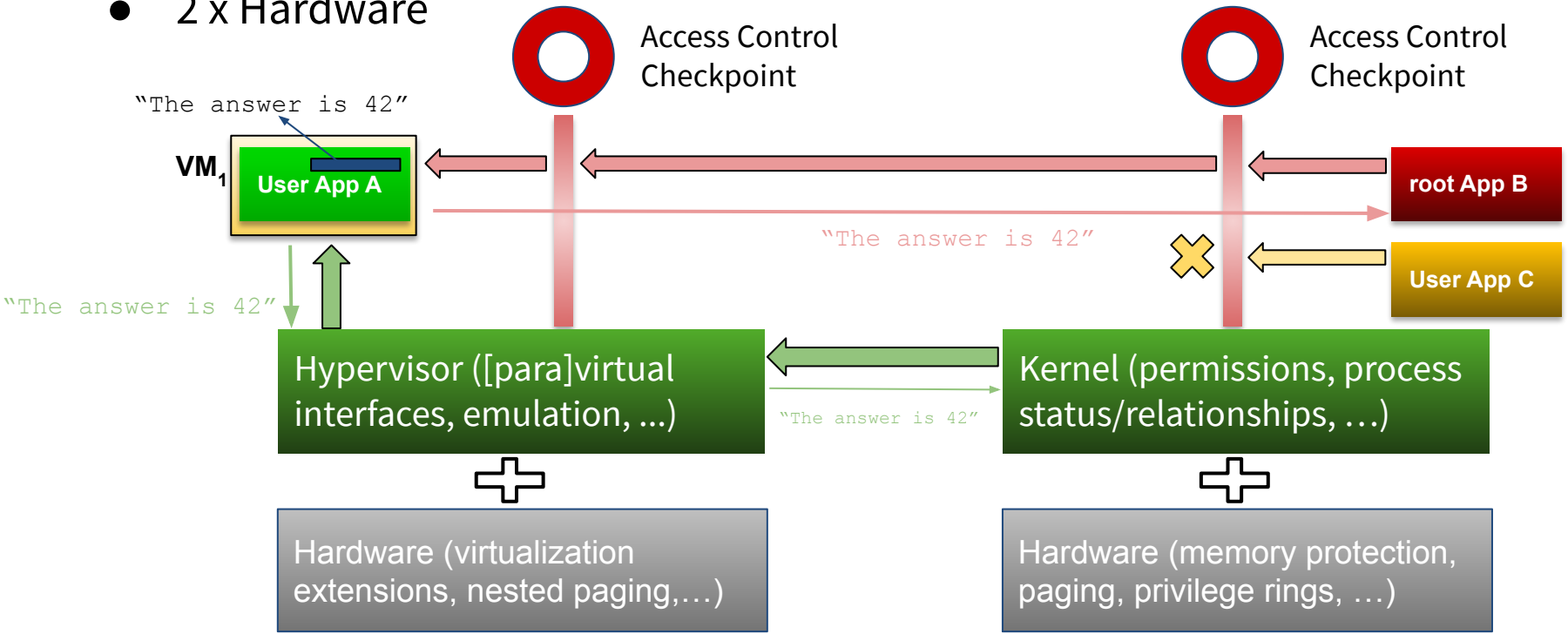
- Firmware
- Hardware
- Host Kernel / Hypervisor
- Host's `root` user & Apps
- VM1 Guest Kernel
- VM1 `root` user & Apps



Enforcing Memory Access Control: Virtualization

Who, from host, can read the secret “The answer is 42” ?

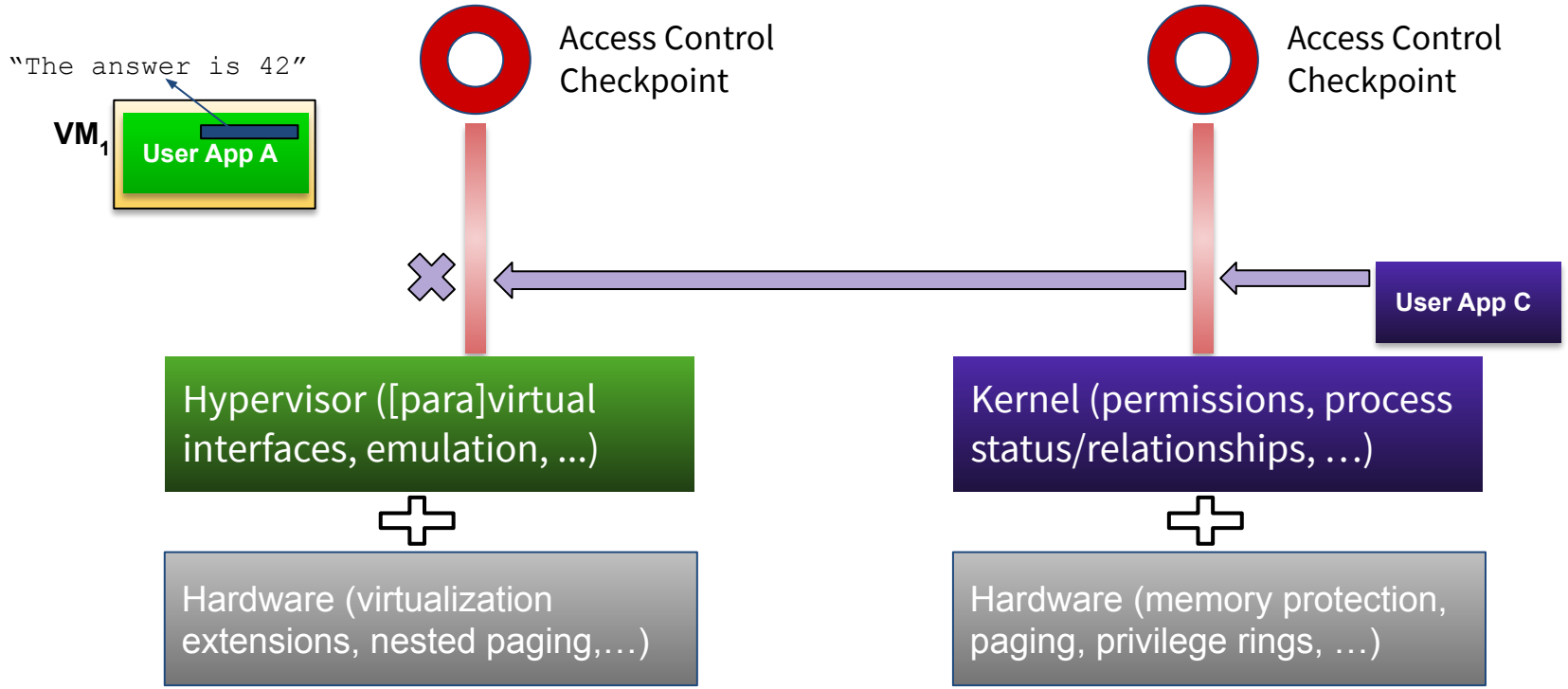
- 2 x Software (Host Kernel, Hypervisor)
- 2 x Hardware



Enforcing Memory Access Control: Virtualization

if *one* fails...

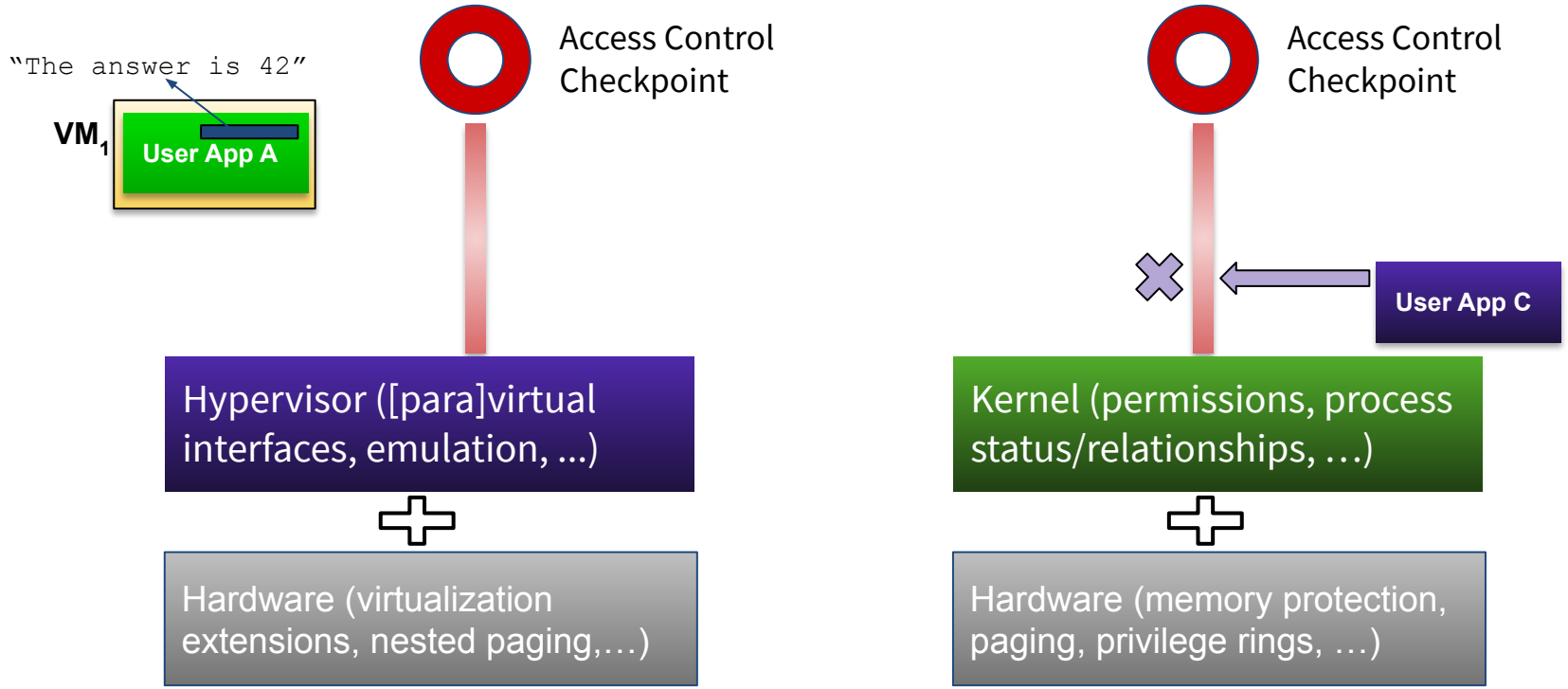
- e.g., Exploited kernel bug



Enforcing Memory Access Control: Virtualization

if *another one* fails...

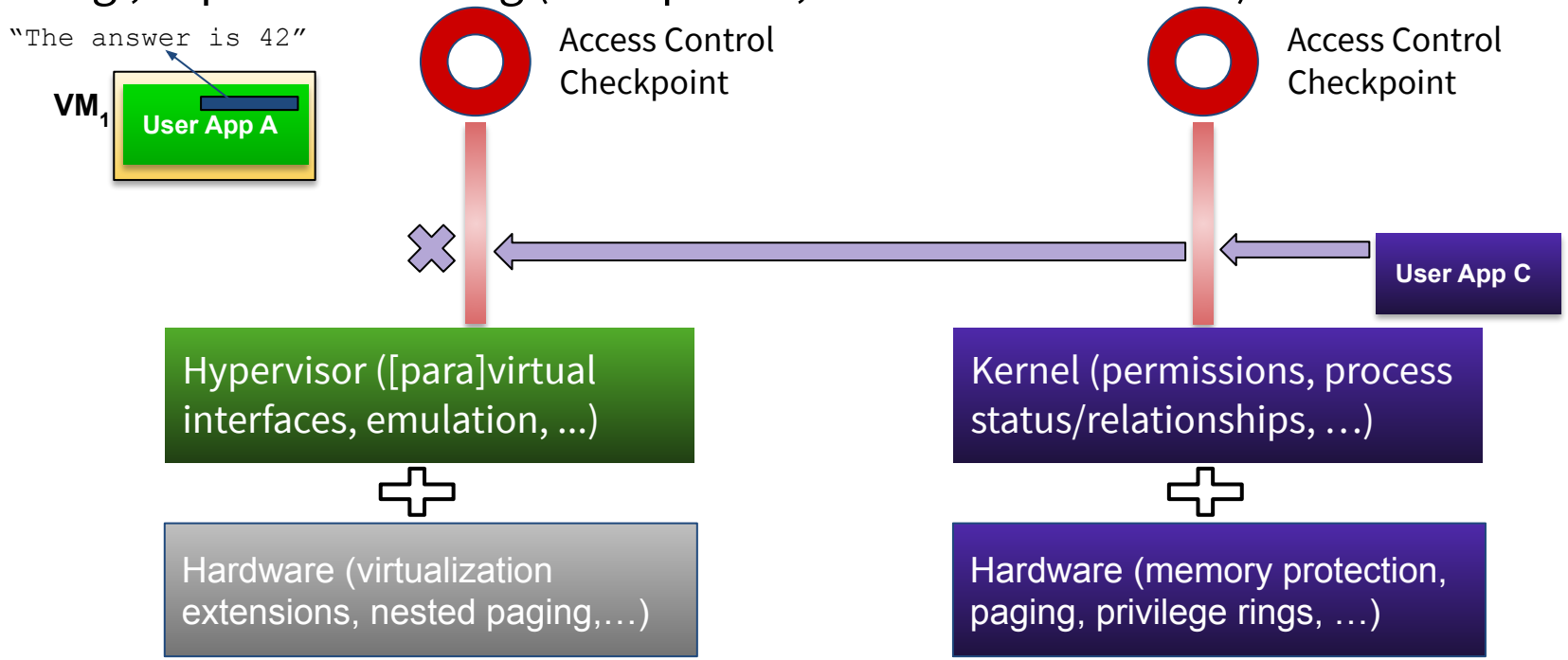
- e.g., Exploited hypervisor bug



Enforcing Memory Access Control: Virtualization

Even if *two* fail...

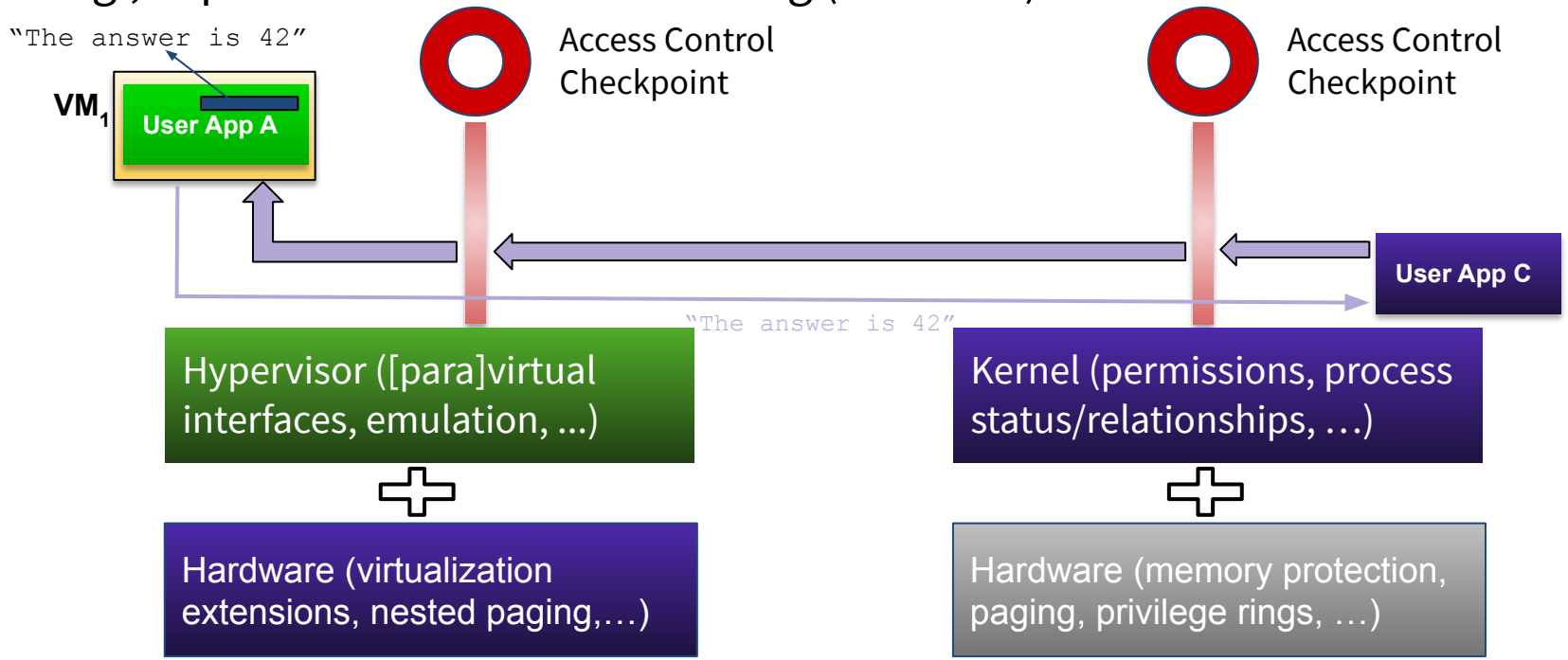
- e.g., Exploited kernel bug
- e.g., Exploited HW bug (like Spectre, Metldown & Friends)



Enforcing Memory Access Control: Virtualization

It must be at least two, and the “proper” twos, that fail...

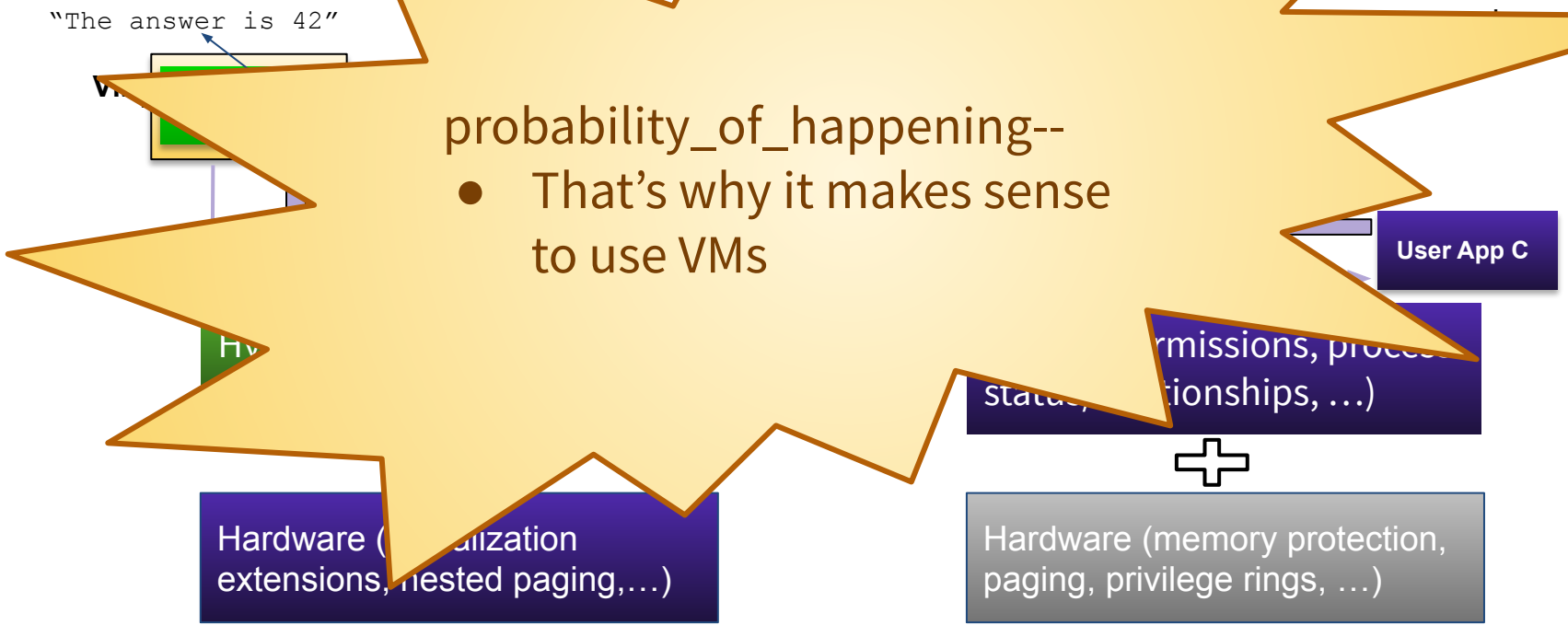
- e.g., Exploited kernel bug
- e.g., Exploited virtualization HW bug (like L1TF)



Enforcing Memory Access Control: Virtualization

It must be at least two, and the “proper” twos, that

- e.g., Exploited kernel bug
- e.g., Exploited



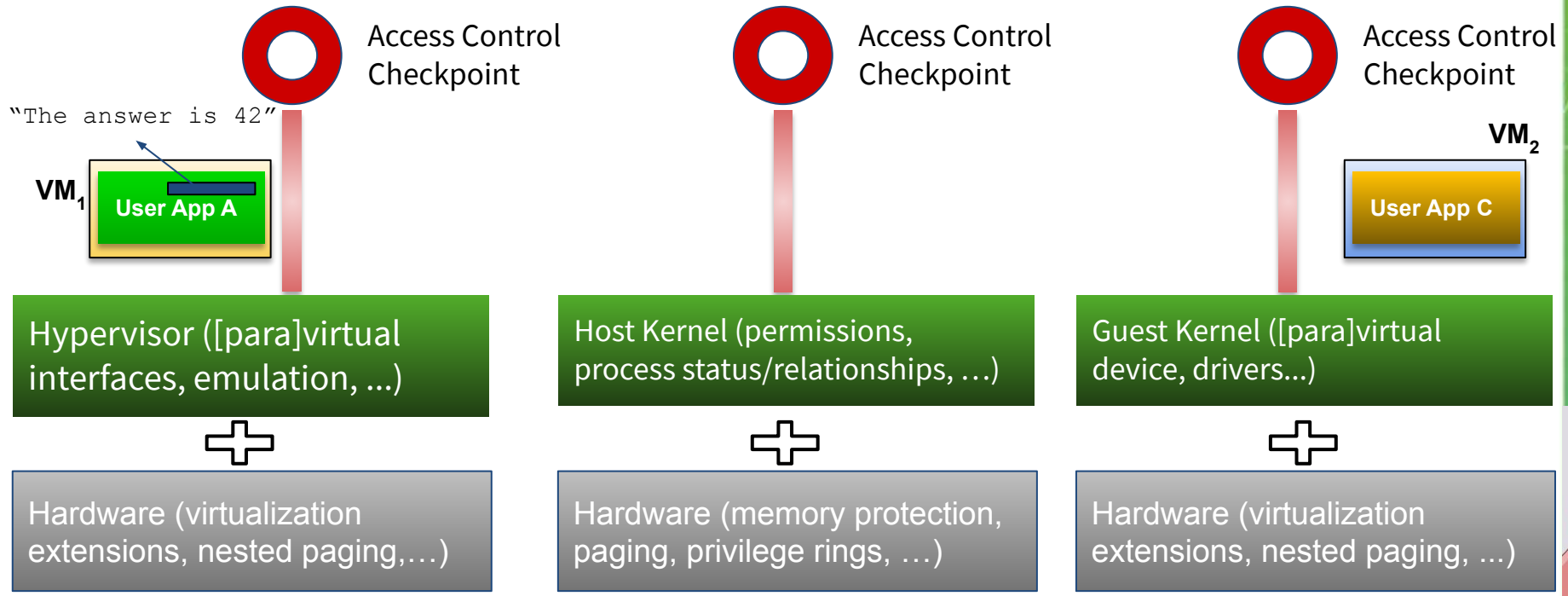
probability_of_happening--

- That’s why it makes sense to use VMs

Enforcing Memory Access Control: Virtualization

Who, from another VM, can read the secret “The answer is 42” ?

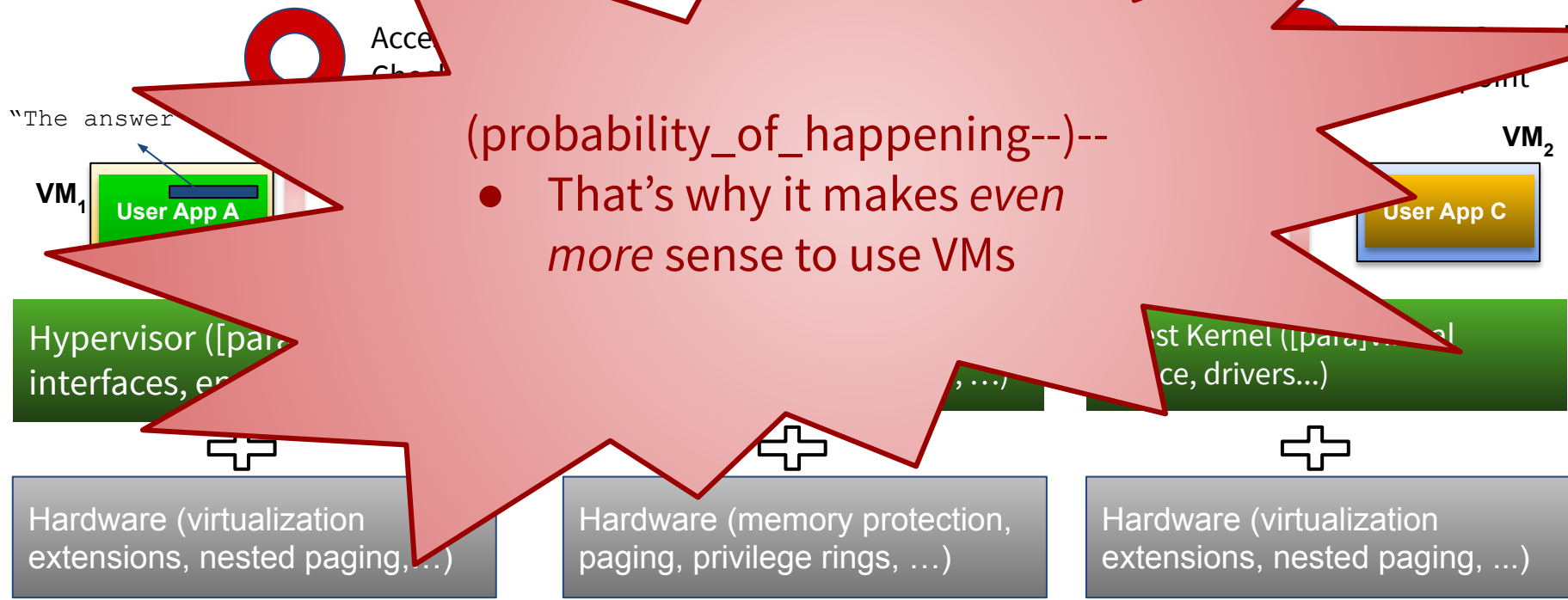
- 3 x Software (Host Kernel, Hypervisor, Guest kernel)
- 3 x Hardware



Enforcing Memory Access Control: Virtualization

Who, from another VM, can read the secret “The a is 42” ?

- 3 x Softwar (Host Kernel, Hyp Gu)
- 3 x Hardware



(probability_of_happening--)--

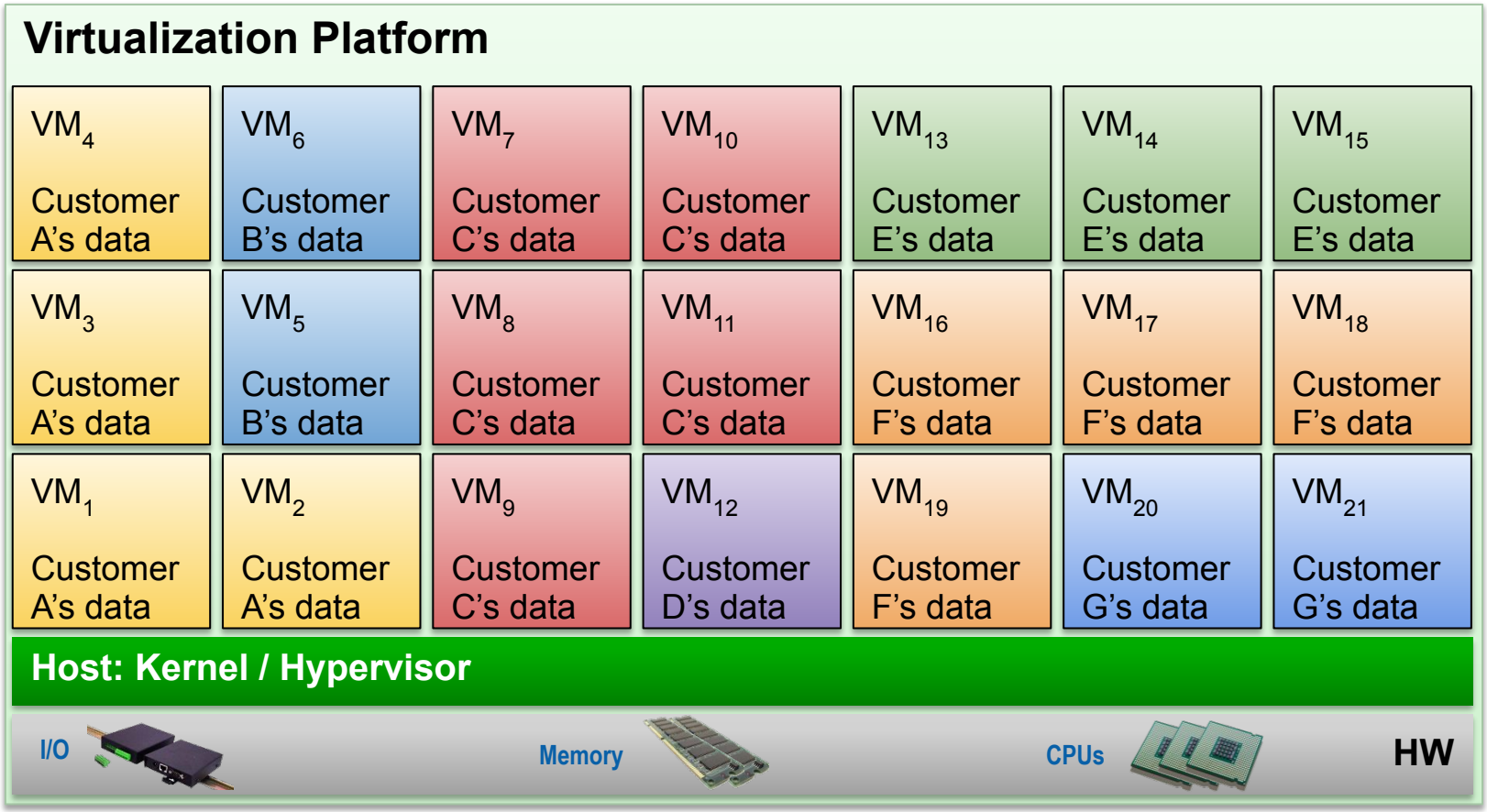
- That’s why it makes *even more* sense to use VMs

Who Owns The TCB, in Virtualization ?

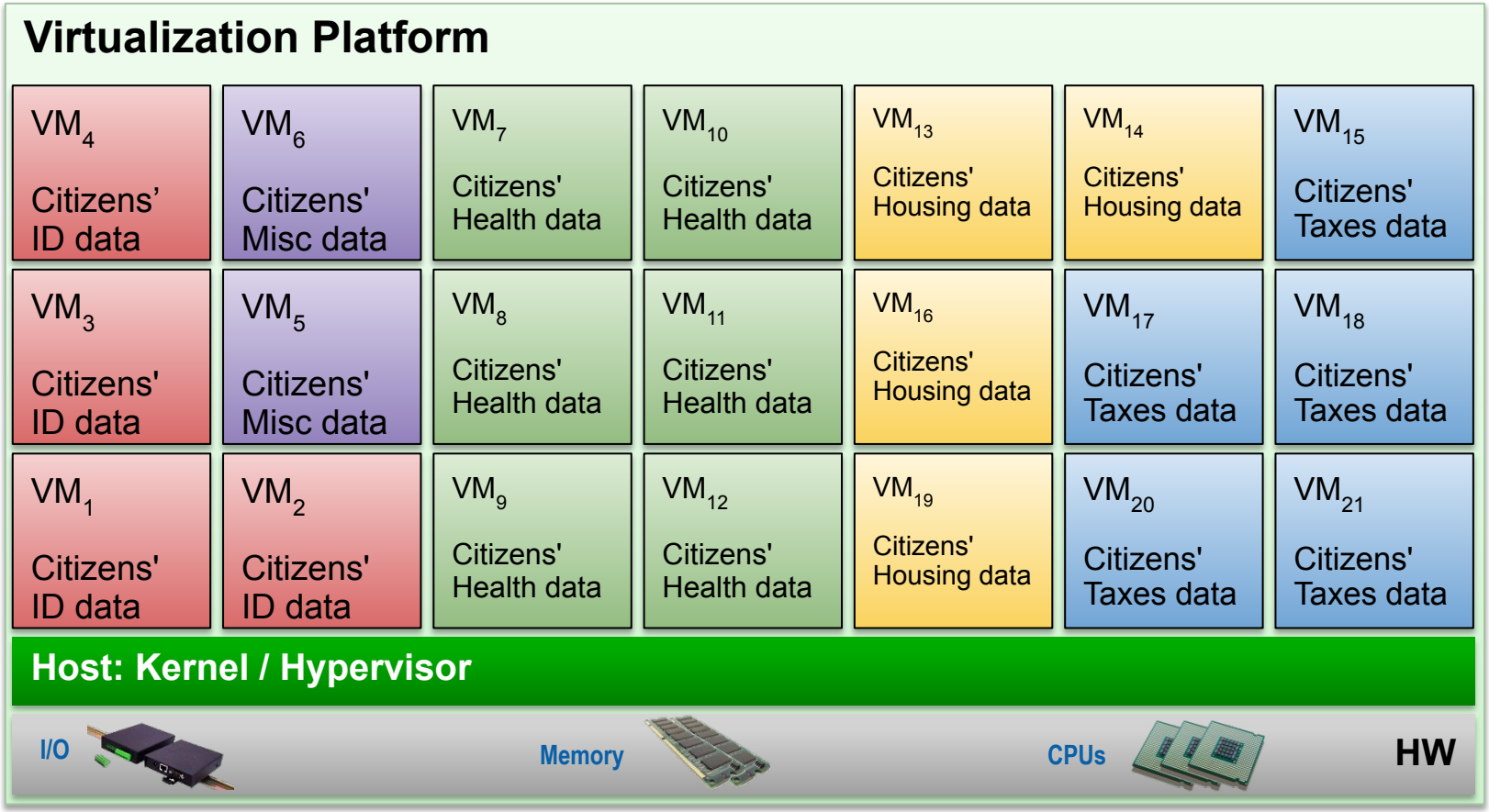
Who is in charge of the components of the TCB

- Personal server(s) & VMs box/server
  - Me
- My company's server(s) / private cloud & VMs
  - My company (e.g., IT department)
- Public Cloud (AWS, GCP, Azure)
  - The Cloud provider
- *Sensitive* servers / private cloud & VMs
  E.g., *Public Administrations*' server(s) / private cloud & VMs
  - PAs' (e.g., their IT departments)

Public Cloud (AWS, GCP, Azure, ...)



PA's Private / Hybrid Cloud



Whose Data Are Stolen ?

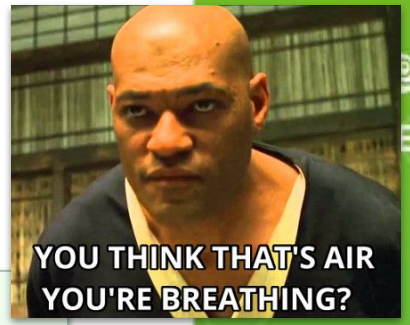
In case of breach in the TCB, what can the attacker steal?

- Personal server(s) & VMs box/server
  - **My data**
- My company's server(s) / private cloud & VMs
  - **My company**'s data
- Public Cloud (AWS, GCP, Azure)
  - ***All the** Cloud Provider's **customers and users***
- *Sensitive* servers / private cloud & VMs
  E.g., **Public Administrations**' server(s) / private cloud & VMs
  - ***All citizens***' data

DATI A CHI?

WWW.LINUXDAY.IT
23 OTTOBRE 2021

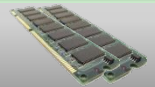
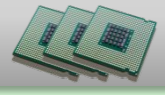
Public Cloud (AWS, GCP, Azure, ...)



Virtualization Platform

VM_4 Citizens' A's data	VM_6 Citizens' B's data	VM_7 Citizens' C's data	VM_{10} Citizens' C's data	VM_{13} Citizens' E's data	VM_{14} Citizens' E's data	VM_{15} Citizens' E's data
VM_3 Citizens' A's data	VM_5 Citizens' B's data	VM_8 Citizens' C's data	VM_{11} Citizens' C's data	VM_{16} Citizens' F's data	VM_{17} Citizens' F's data	VM_{18} Citizens' F's data
VM_1 Citizens' A's data	VM_2 Citizens' A's data	VM_9 Citizens' C's data	VM_{12} Citizens' D's data	VM_{19} Citizens' F's data	VM_{20} Citizens' G's data	VM_{21} Citizens' G's data

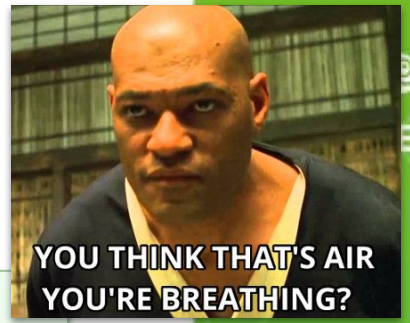
Host: Kernel / Hypervisor

I/O  Memory  CPUs  HW

Hypervisor gets compromised, e.g., by successful attack

Hypervisor might be malicious <<Do you really trust Google, Amazon, Microsoft? Eh? Eh? Eh?>>

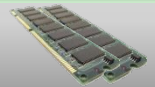
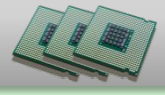
Public Cloud (AWS, GCP, Azure, ...)



Virtualization Platform

VM_4 Citizens' A's data	VM_6 Citizens' B's data	VM_7 Citizens' C's data	VM_{10} Citizens' C's data	VM_{13} Citizens' E's data	VM_{14} Citizens' E's data	VM_{15} Citizens' E's data
VM_3 Citizens' A's data	VM_5 Citizens' B's data	VM_8 Citizens' C's data	VM_{11} Citizens' C's data	VM_{16} Citizens' F's data	VM_{17} Citizens' F's data	VM_{18} Citizens' F's data
VM_1 Citizens' A's data	VM_2 Citizens' A's data	VM_9 Citizens' C's data	VM_{12} Citizens' D's data	VM_{19} Citizens' F's data	VM_{20} Citizens' G's data	VM_{21} Citizens' G's data

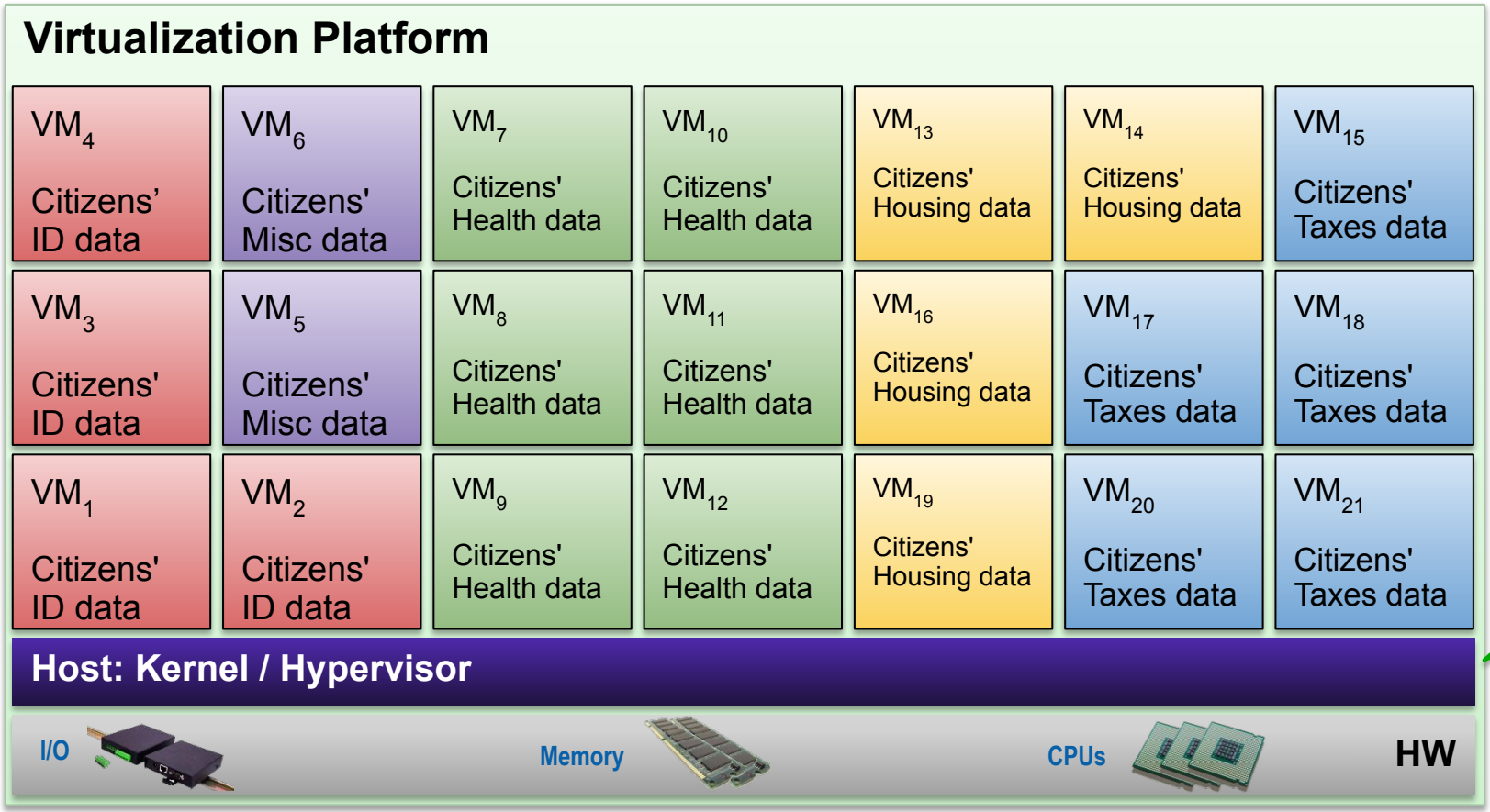
Host: Kernel / Hypervisor

I/O  Memory  CPUs  HW

Hypervisor gets compromised, e.g., by successful attack

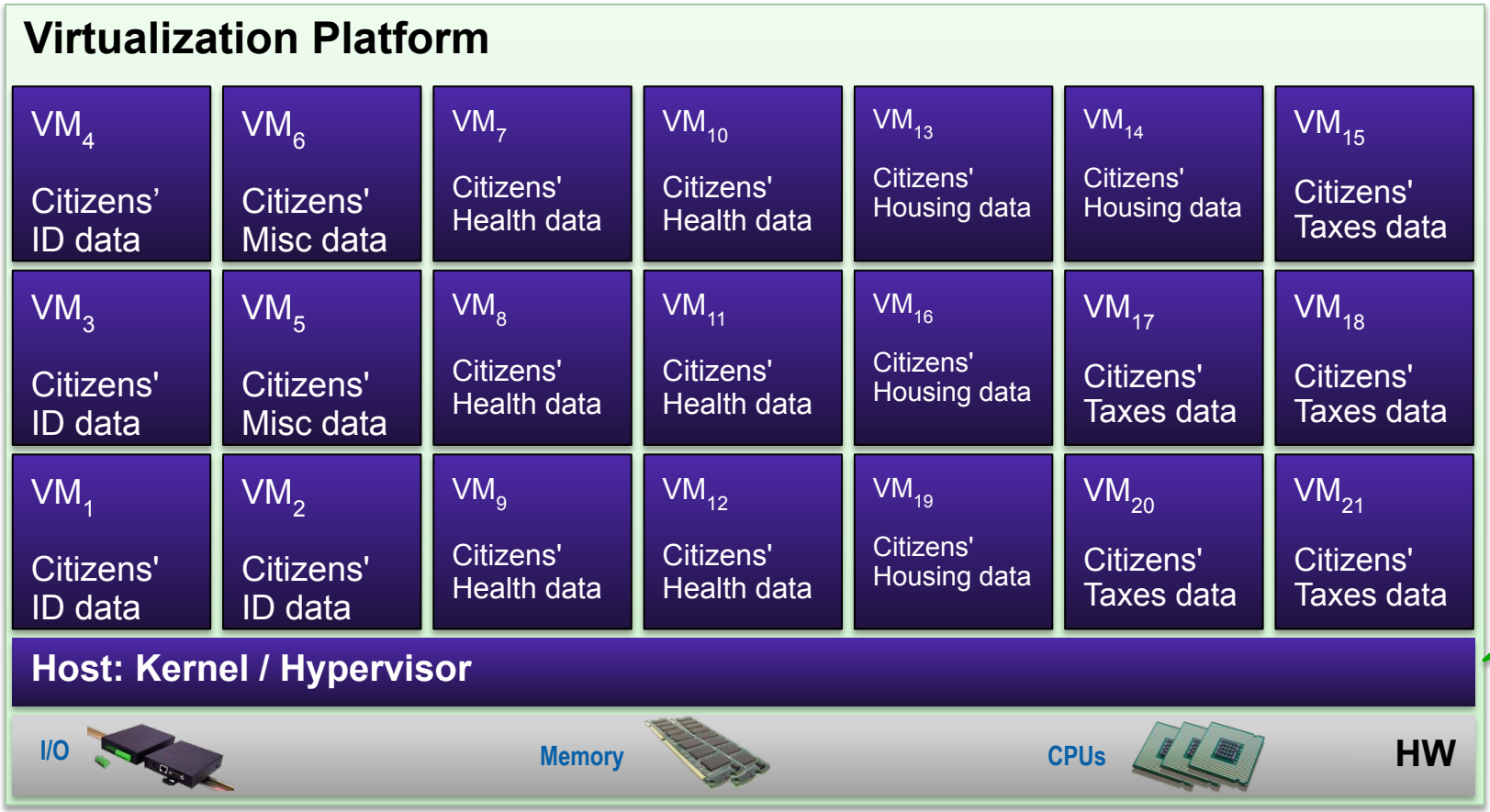
Hypervisor might be malicious <<Do you really trust Google, Amazon, Microsoft? Eh? Eh? Eh?>>

PA's Private / Hybrid Cloud



Hypervisor gets compromised, e.g., by successful attack

PA's Private / Hybrid Cloud



Hypervisor gets compromised, e.g., by successful attack

Reading Data from VMs' Memory

- Inside a VM (called TW)

```
$ ./pippo
PID: 1820
Input the Super Secret Password:
```

- On the host (as `root`)

```
# ps aux | grep qemu | grep TW | awk '{print $2}'
102198
# gcore -a 102198
# grep -a linuxday2021 core.102198
#
```

Reading Data from VMs' Memory

- This time, password is "linuxday2021"

```
$ ./pippo
PID: 1820
Input the Super Secret Password: ************
```

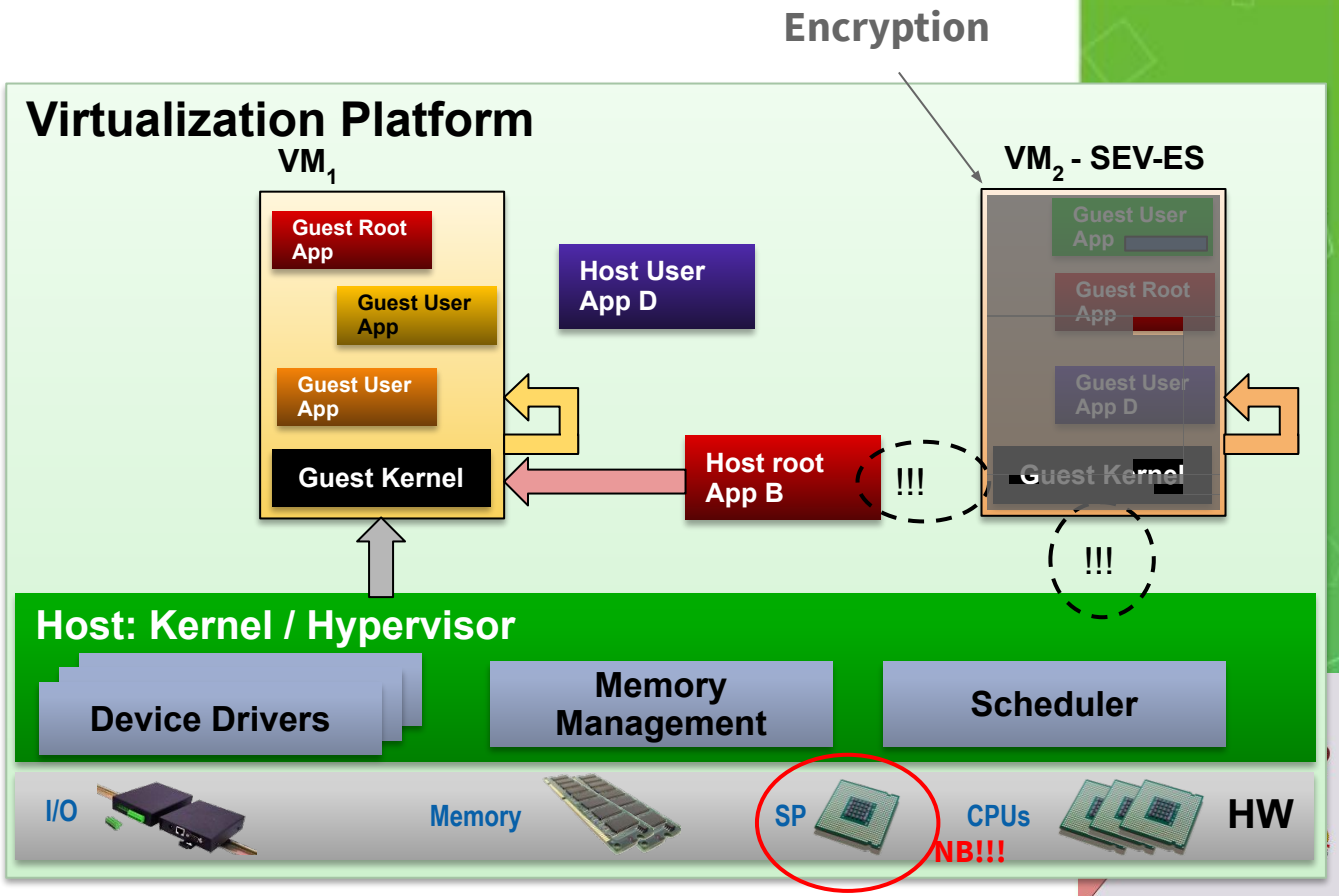
- On the host (as `root`)

```
# ps aux | grep qemu | grep TW | awk '{print $2}'
102198
# gcore -a 102198
# grep -a linuxday2021 core.102198
linuxday2021
!Q#_[Secret Password: ecret Password: !Input the Super Secret Password:!
30EU0RD!SUPER_SECRET_PASSWORD!SUPER_SECRET_PASSWORD105BU@Ug!C'EU[SWORD1!
/usr/bin/sleep!sleep 30SWORD1A?U9BUs: $A  EU[sword is:
$SUPER_SECRET_PASSWORD"A#_[word is: $SUPER_SECRET_PASSWORD1y11111:
"1!linuxday2021!190USWORD!
```

Encrypted Virtualization: Legit Ops

VM_2 is a SEV-ES VM

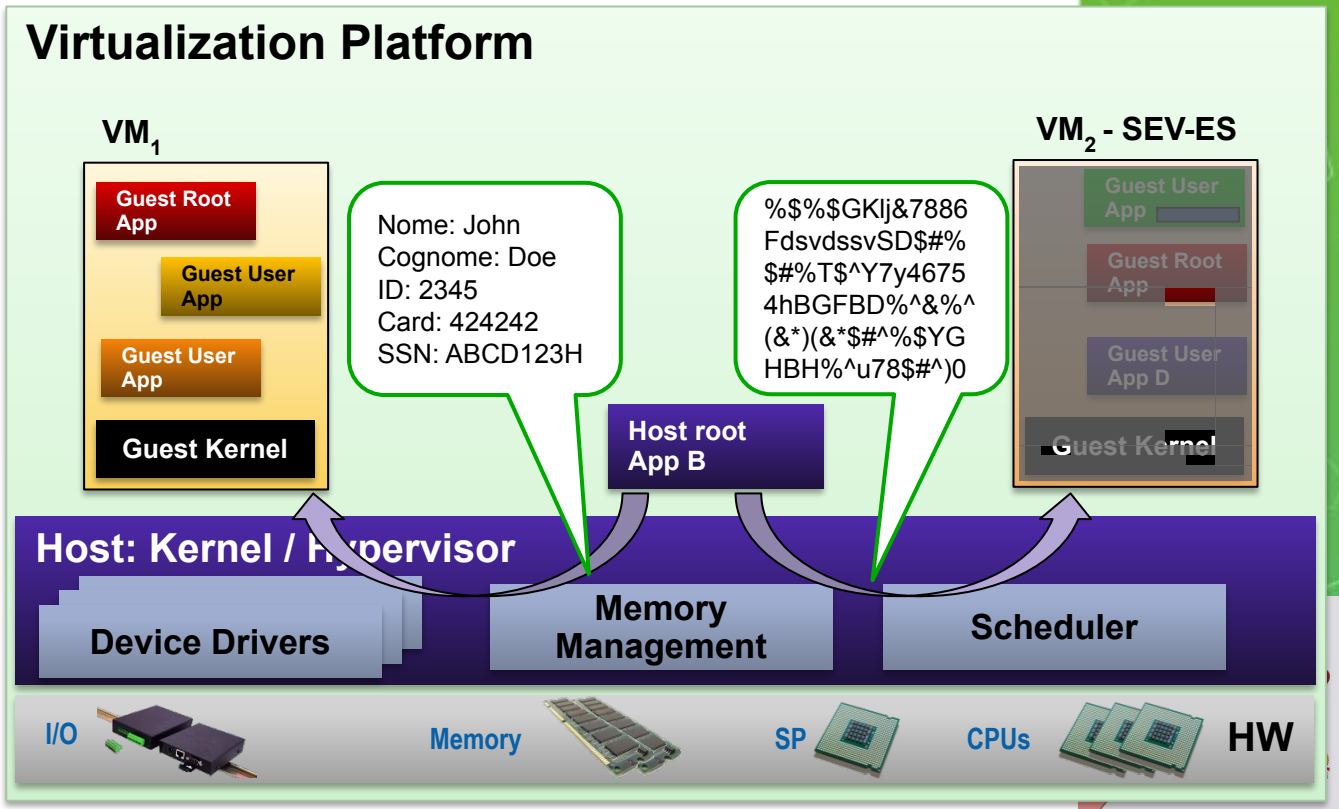
- Can read its own memory
- Kernel / Hypervisor can't read its memory
- Host `root` can't read its memory



Encrypted Virtualization: Attacks

VM_2 is a SEV-ES VM

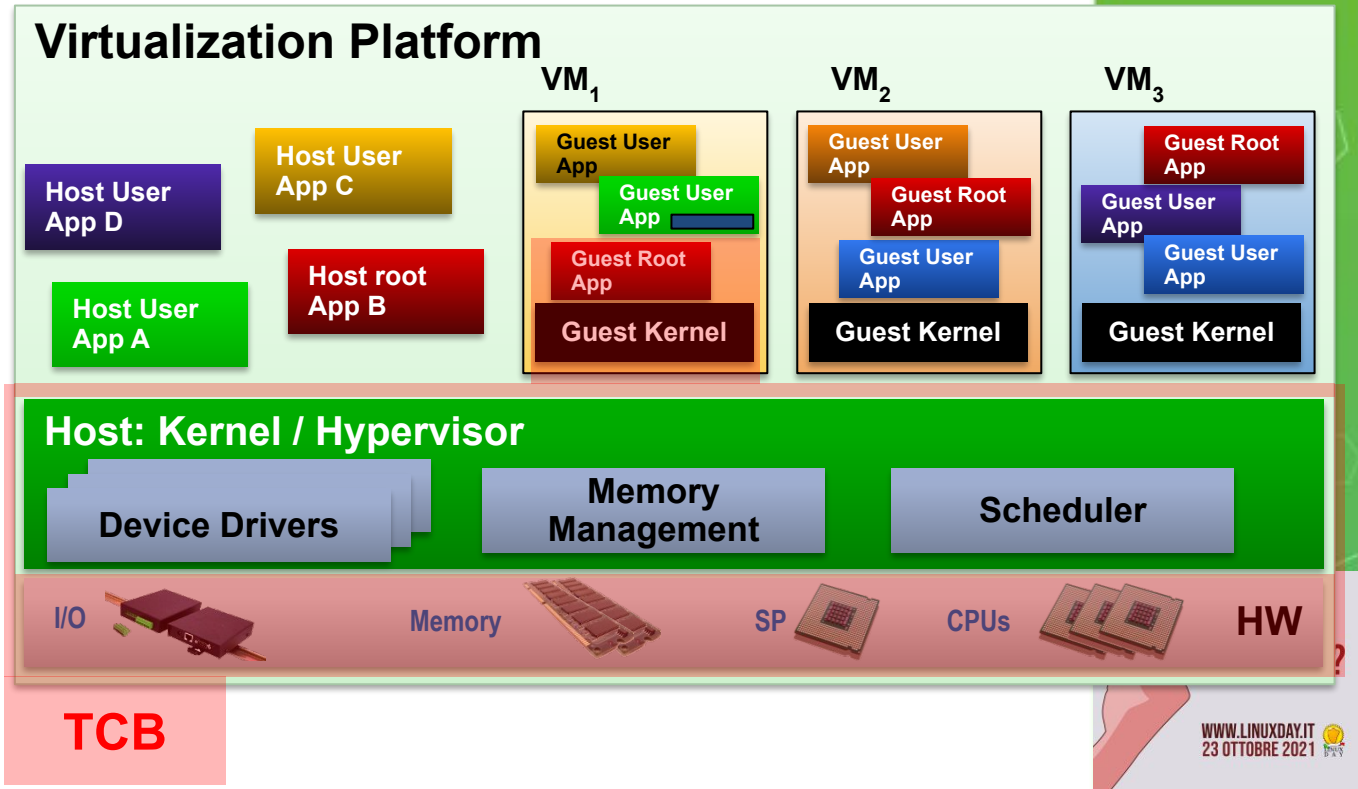
- Can read its own memory
- Kernel / Hypervisor can't read its memory
- Host root cant read its memory



Encrypted Virtualization: TCB

From the point of view of Guest User App A, in VM1 (where the sensitive data are)

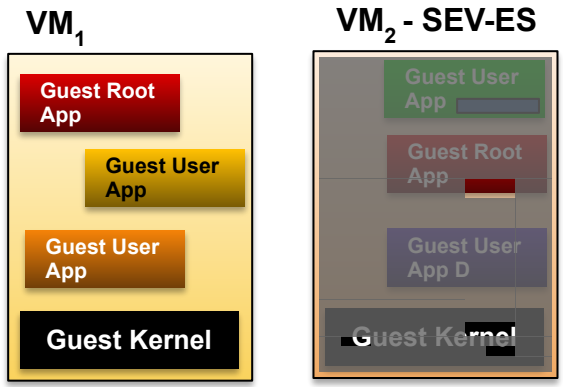
- Firmware
- Hardware
- ~~Host Kernel / Hypervisor~~
- ~~Host's `root` user & Apps~~
- VM1 Guest Kernel
- VM1 `root` user & Apps



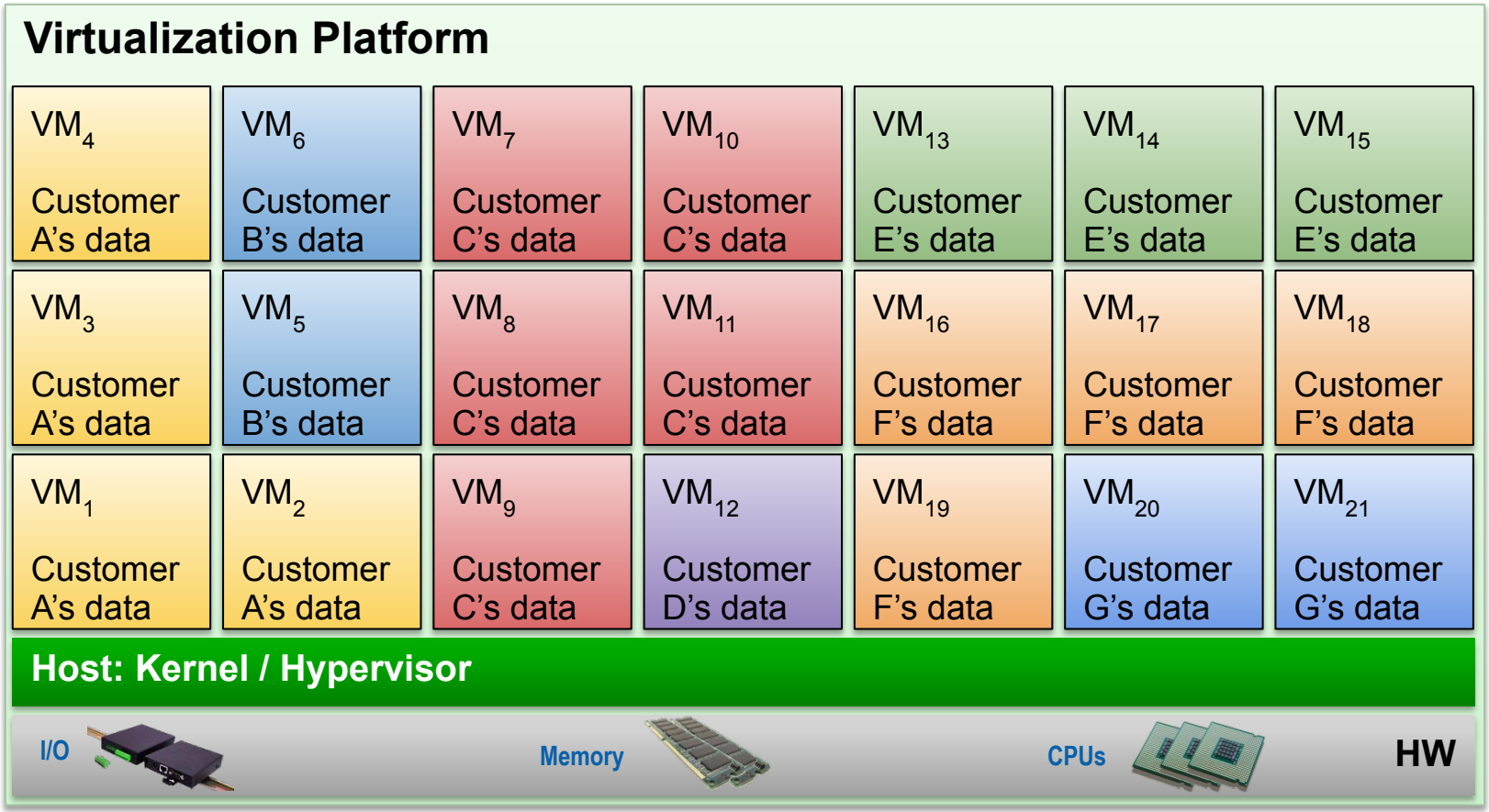
Encrypted Virtualization with SEV-ES

Note that:

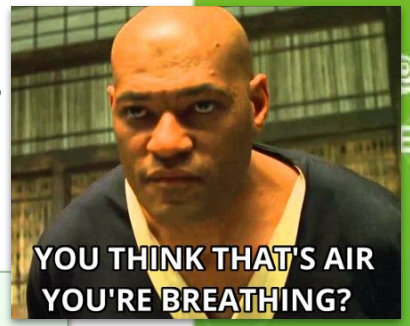
- Not all VM_2 memory is encrypted
- Some small pieces are not encrypted
  - Necessary for communication between VM and hypervisor (for “implementing” virtualization)
  - The VM is in control and decides what is encrypted and what is not
- VM_1 is not encrypted
  - Encrypted and non-encrypted VMs can coexist on the same host



If Public Cloud Offers Encrypted Virt.



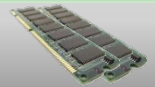
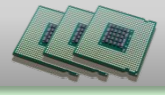
If Public Cloud Offers Encrypted Vi



Virtualization Platform

VM_4 Customer A's data	VM_6 Customer B's data	VM_7 Customer C's data	VM_{10} Customer C's data	VM_{13} Customer E's data	VM_{14} Customer E's data	VM_{15} Customer E's data
VM_3 Customer A's data	VM_5 Customer B's data	VM_8 Customer C's data	VM_{11} Customer C's data	VM_{16} Customer F's data	VM_{17} Customer F's data	VM_{18} Customer F's data
VM_1 Customer A's data	VM_2 Customer A's data	VM_9 Customer C's data	VM_{12} Customer D's data	VM_{19} Customer F's data	VM_{20} Customer G's data	VM_{21} Customer G's data

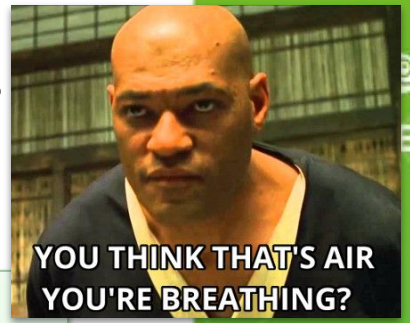
Host: Kernel / Hypervisor

I/O  Memory  CPUs  HW

Hypervisor gets compromised, e.g., by successful attack

<<Hypervisor is controlled by Cloud Provider… Who knows what they're up to ?!?!>>

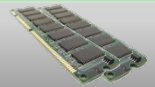
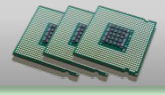
If Public Cloud Offers Encrypted Vi



Virtualization Platform

VM4 Citizens' A's data	VM6 Citizens' B's data	VM7 Citizens' C's data	VM10 Citizens' C's data	VM13 Citizens' E's data	VM14 Citizens' E's data	VM15 Citizens' E's data
VM3 Citizens' A's data	VM5 Citizens' B's data	VM8 Citizens' C's data	VM11 Citizens' C's data	VM16 Citizens' F's data	VM17 Citizens' F's data	VM18 Citizens' F's data
VM1 Citizens' A's data	VM2 Citizens' A's data	VM9 Citizens' C's data	VM12 Citizens' D's data	VM19 Citizens' F's data	VM20 Citizens' G's data	VM21 Citizens' G's data

Host: Kernel / Hypervisor

I/O  Memory  CPUs  HW

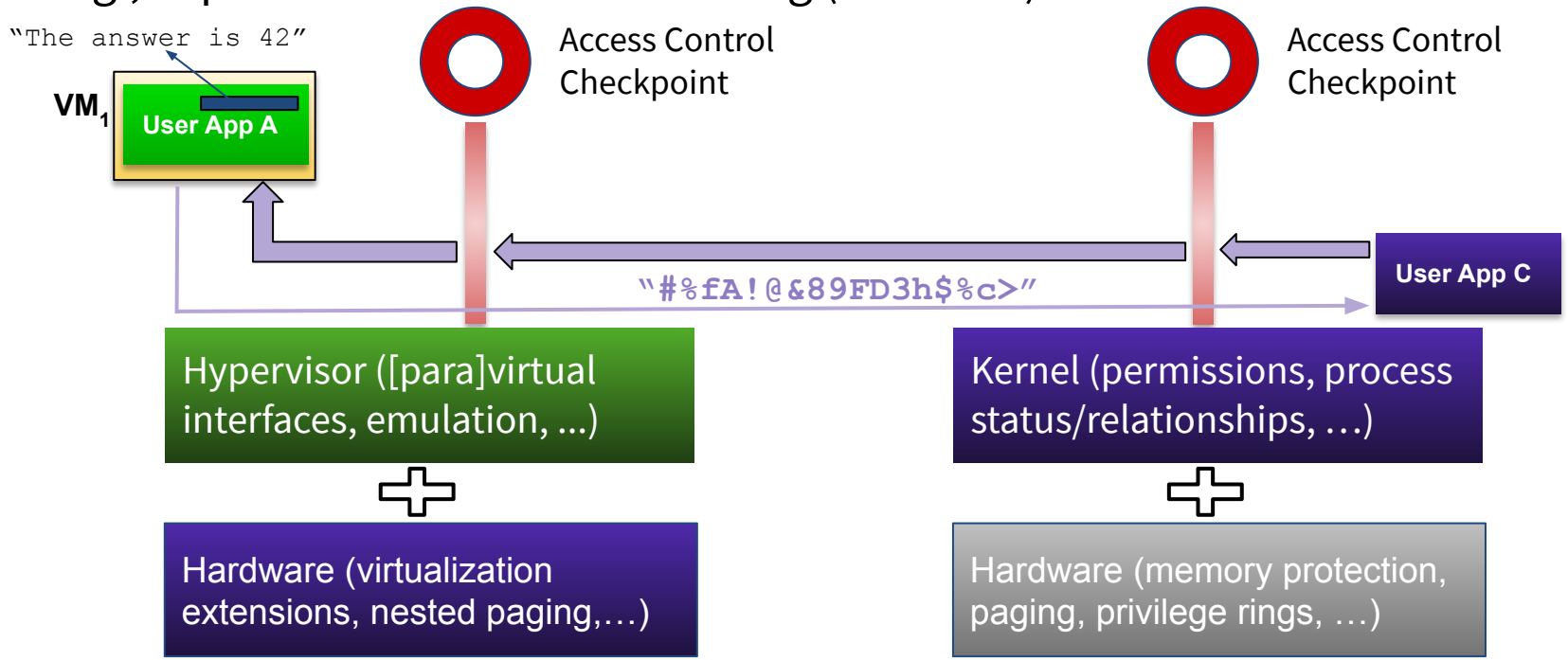
Hypervisor gets compromised, e.g., by successful attack

<<Hypervisor is controlled by Cloud Provider… Who knows what they're up to ?!?!>>

Enforcing Memory Access Control: Encrypted Virtualization

Even if it is two, and the “proper” twos, that fail...

- e.g., Exploited kernel bug
- e.g., Exploited virtualization HW bug (like L1TF)



Reading Data from SEV-ES VMs' Memory

- Inside a VM (called TW-SEV-ES):

```
$ ./pippo
PID: 1820
Input the Super Secret Password:
```

- On the host (as `root`)

```
# ps aux | grep qemu | grep TW-SEV-ES | awk '{print $2}'
102198
# gcore -a 102198
# grep -a GOLEM core.102198
#
```

Reading Data from SEV-ES VMs' Memory

- This time, as password, we type "GOLEM" [1]

```
$ ./pippo
PID: 1820
Input the Super Secret Password: *****
```

- On the host (as `root`)

```
# ps aux | grep qemu | grep TW-SEV-ES | awk '{print $2}'
102198
# gcore -a 102198
# grep -a GOLEM core.102198
#
```

Hardware Prerequisites

Different CPU Generation, different Features:

- SEV
  - 1st Gen. AMD EPYC Processors (Naples)
  - en.wikichip.org/wiki/amd/microarchitectures/zen
- SEV-ES
  - 2nd Gen. AMD EPYC Processors (Rome)
  - en.wikichip.org/wiki/amd/microarchitectures/zen_2
- SEV-SNP
  - 3rd Gen. AMD EPYC Processors (Milan)
  - en.wikichip.org/wiki/amd/microarchitectures/zen_3
    - From earlier *this* year

Software Prerequisites

Support for SEV & SEV-ES present in:

- Upstream projects:
  - QEMU: since **v6.0**
  - OVMF: since **Nov 2020**
  - Libvirt: since v7.5.0 (I think)
  - Linux Kernel, guest support (inside VM): since **v5.10**
  - Linux Kernel, KVM support(as host): since v5.11

- openSUSE Tumbleweed
  - full (host/guest) support since mid-May
- openSUSE Leap 15.3 - SUSE Linux Enterprise Server 15 SP3
  - SEV-ES guest support

SEV[-ES] Encrypted VMs on openSUSE Tumbleweed

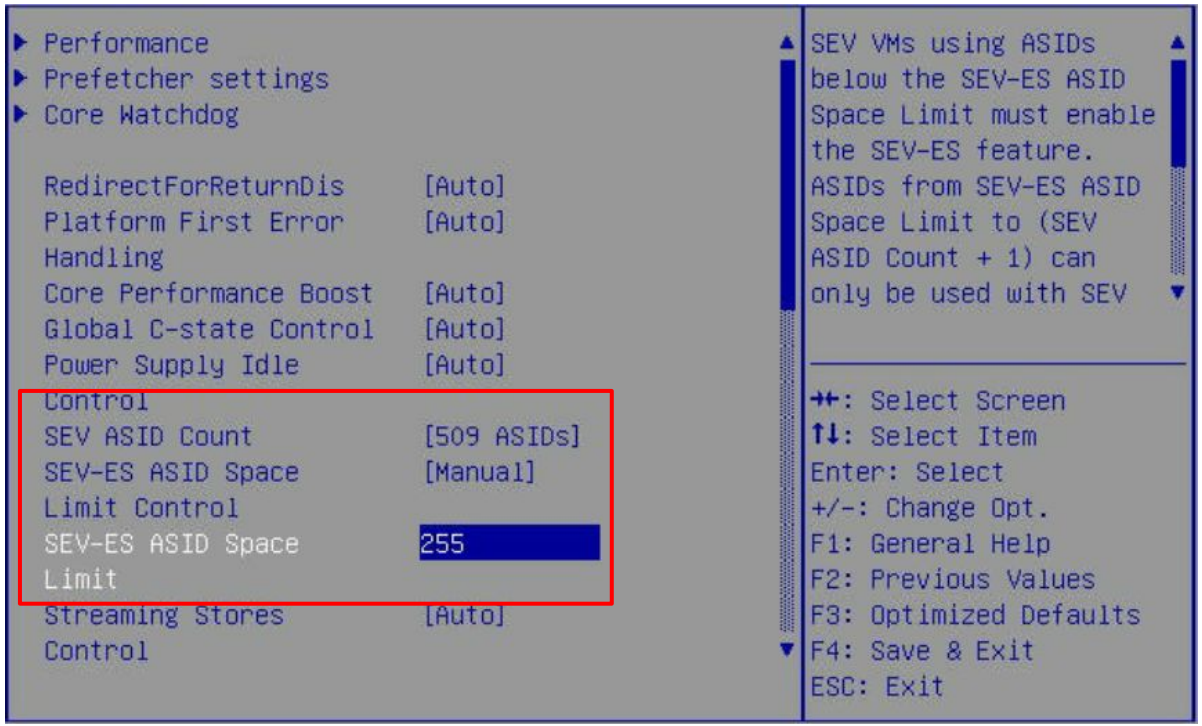
- All that is necessary is there, in openSUSE Tumbleweed[*]
  - Host kernel support
  - Guest kernel support
  - OVMF support
  - QEMU support
  - Libvirt support

Other distros? Check the relevant docs for requirements (see previous slide)

- GUIs are still a bit lacking
  - E.g., Virtual Machine Manager
  - We still some manual tweaking from CLI / config files

Preparation: BIOS

- SEV & SEV-ES needs being enabled
- Number of ASIDs for SEV & SEV-ES must be set (> 1 !)



Preparation: BIOS

ASIDs = Address Space IDentifiers

- Used by hardware for identify processes or VMs, for performance reasons (e.g., TLB tagging)
- In SEV, used to select the right encryption key, when accessing encrypted memory.

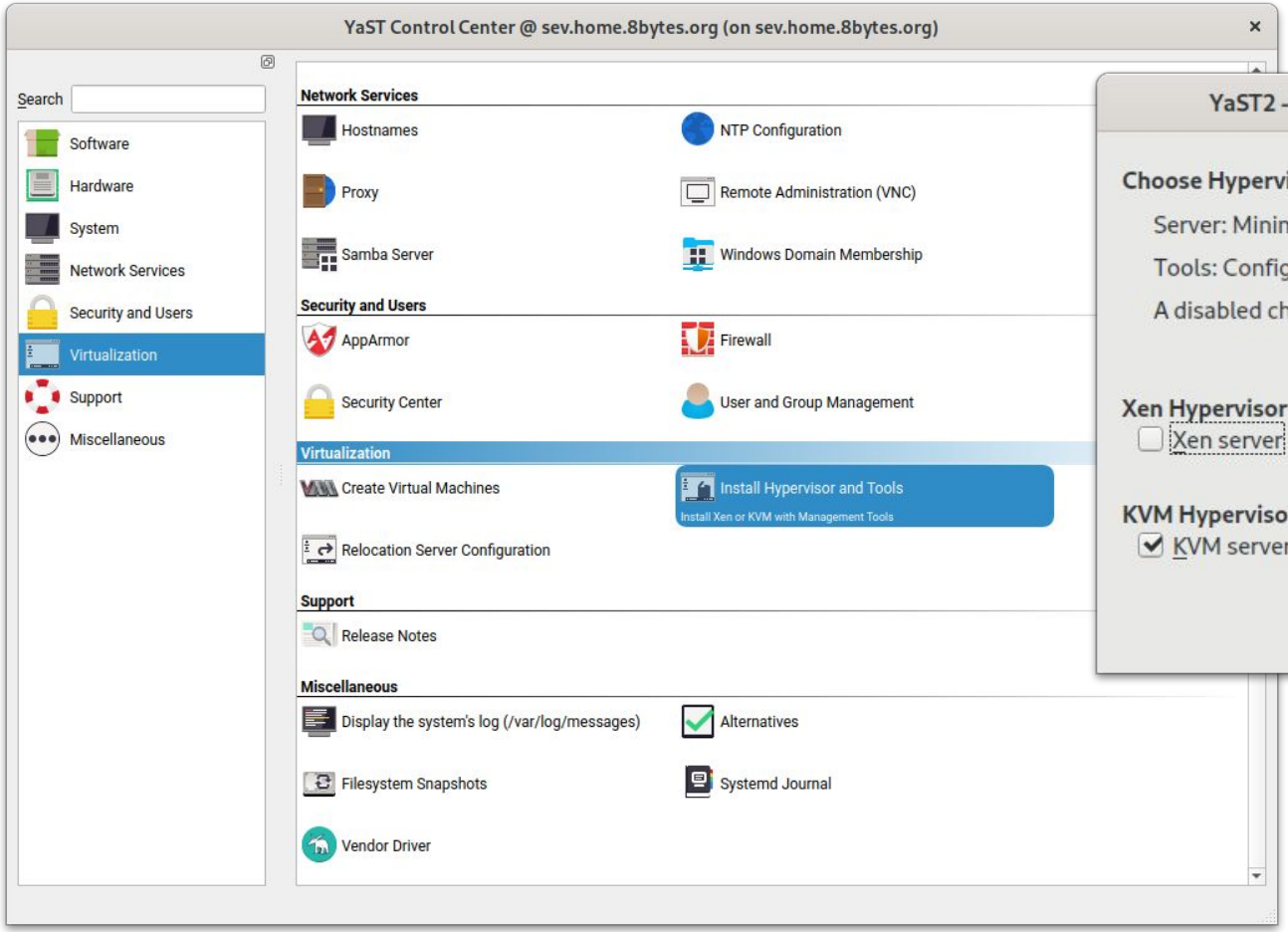
Preparation: Software

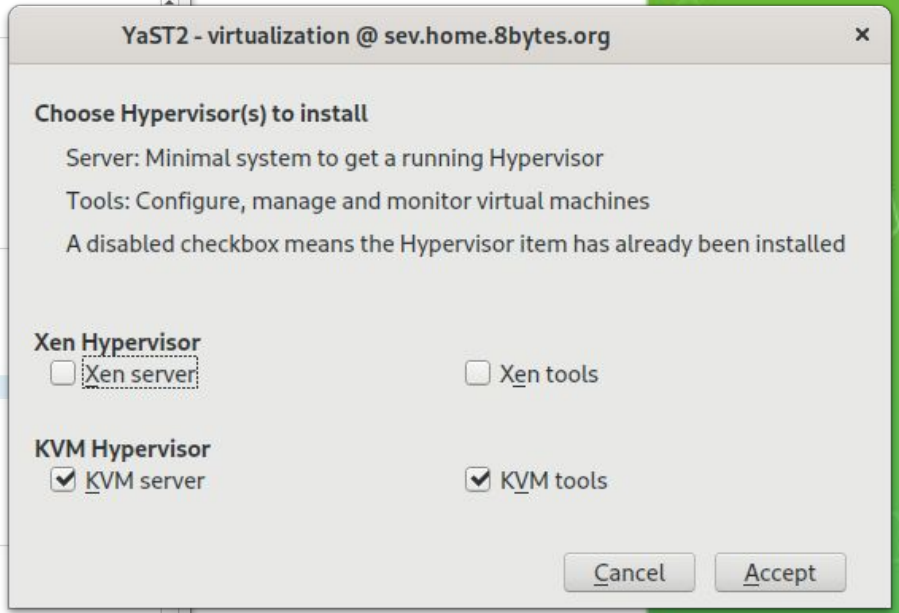
Install the KVM stack:

- in a terminal
  - `zypper in -t pattern kvm_server kvm_tools`
    `zypper in virt-viewer`
  - create a network bridge (not required, though)

- via YaST
  - installs all the software and make the bridge

Preparation: Software, via YaST





Preparation: Software

- Check:

```
myepic:~ # dmesg |grep SEV
[   13.890438] ccp 0000:23:00.1: SEV firmware update successful
[   13.986479] ccp 0000:23:00.1: SEV API:1.42 build:42
[   17.753095] SEV supported: 255 ASIDs
[   17.757527] SEV-ES supported: 254 ASIDs
```

- If no seeing "`SEV supported: 255 ASIDs`" try:

```
myepic:~ # rmmod kvm_amd
myepic:~ # modprobe kvm-amd sev=1 sev-es=1
myepic:~ # dmesg | grep SEV
[   13.890438] ccp 0000:23:00.1: SEV firmware update successful
[   13.986479] ccp 0000:23:00.1: SEV API:1.42 build:42
[   17.753095] SEV supported: 255 ASIDs
[   17.757527] SEV-ES supported: 254 ASIDs
```

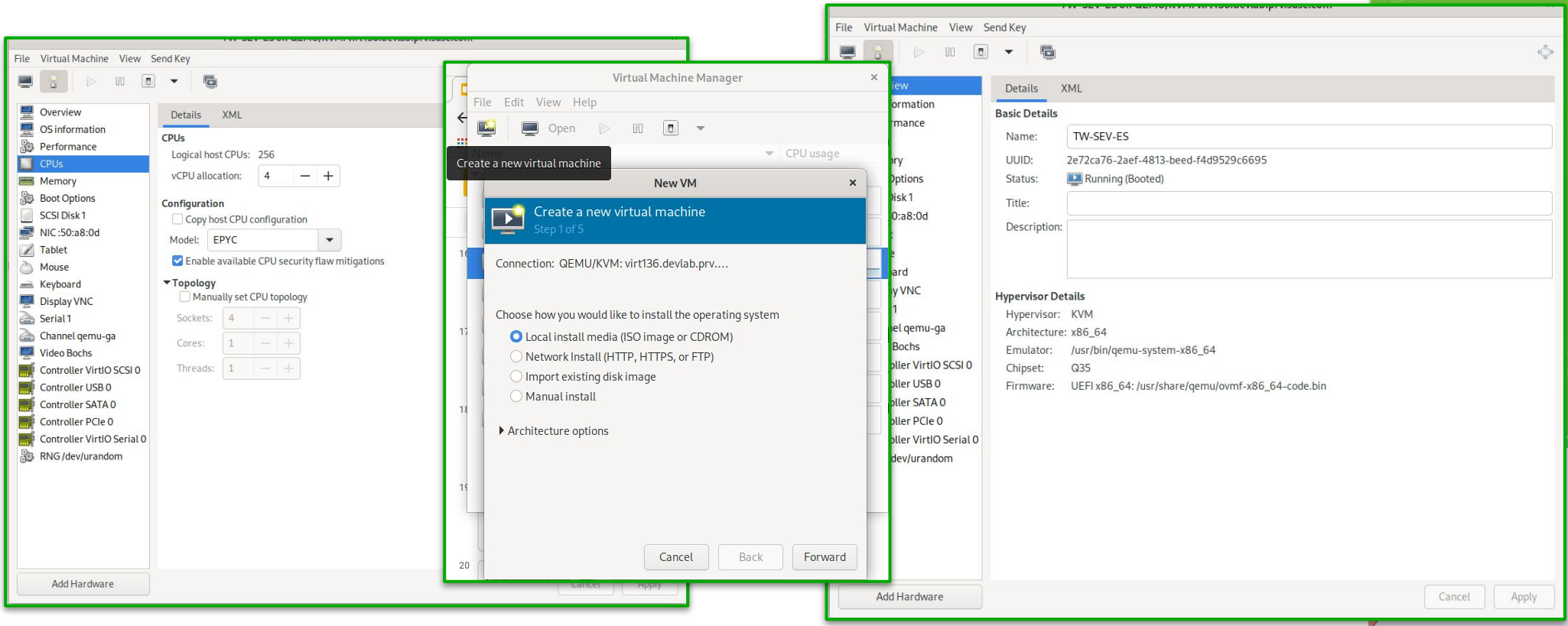
Creating Your SEV VM

- From terminal:
  - the following command will start a VM installation
  - see: Running Encrypted VMs on openSUSE Tumbleweed

```
# virt-install \
     --arch x86_64 \
     --name "TW-SEV" \
     --vcpus 4 \
     --cpu EPYC \
     --memory 4096 \
     --machine q35 \
     --memtune hard_limit=4563402 \
     --disk size=32,target.bus=scsi \
     --controller type=scsi,model=virtio-scsi,driver.iommu=on \
     --network network=default,model=virtio,driver.iommu=on \
     --launchSecurity sev,policy=0x3 \
     --boot \
loader=/usr/share/qemu/ovmf-x86_64-code.bin,loader.readonly=yes,loader.type=pflash,
nvram.template=/usr/share/qemu/ovmf-x86_64-vars.bin,loader_secure=no \
     --install os=opensusetumbleweed
```

How About GUI ?

Yes, but not possible to configure SEV[-ES] details...



Generic Options

```
# virt-install \
     --name "TW-SEV" \
     --arch x86_64 \
     --cpu EPYC \
     --machine q35 \
     --vcpus 4 \
     --memory 4096 \
     --memtune hard_limit=4563402 \
     --disk size=32,target.bus=scsi \
     --controller type=scsi,model=virtio-scsi,driver.iommu=on \
     --network network=default,model=virtio,driver.iommu=on \
     --launchSecurity sev,policy=0x3 \
     --boot \
loader=/usr/share/qemu/ovmf-x86_64-code.bin,loader.readonly=yes,loader.type=pflash,nv
ram.template=/usr/share/qemu/ovmf-x86_64-vars.bin,loader_secure=no \
     --install os=opensusetumbleweed
```

VM's name

VM's virtual CPUs model

nr. of vCPUs and amount of RAM

VM's virtual disk (32 GBs)

openSUSE Tumbleweed as a VM's OS (too)

VM's virtual BIOS is OVMF

SEV Specific Options

```
# virt-install \
     --name "TW-SEV" \
     --arch x86_64 \
     --cpu EPYC \
     --machine q35 \
     --vcpus 4 \
     --memory 4096 \
     --memtune hard_limit=4563402 \
     --disk size=32,target.bus=scsi \
     --controller type=scsi,model=virtio-scsi,driver.iommu=on \
     --network network=default,model=virtio,driver.iommu=on \
     --launchSecurity sev,policy=0x3 \
     --boot \
loader=/usr/share/qemu/ovmf-x86_64-code.bin,loader.readonly=yes,loader.type=pflash,nv
ram.template=/usr/share/qemu/ovmf-x86_64-vars.bin,loader_secure=no \
     --install os=opensusetumbleweed
```

VM will see an “emulated” EPYC vCPU

4563402 KB = 4 GB + 360 MB
“just a bit more”™ ® than the VM’s RAM

iommu=on required for DMA to work
(via unencrypted bounce-buffers)

Where we say we want **SEV**

the proper firmware (i.e, one with SEV[-ES] support)
binary needs being specified explicitly

SEV Specific Options

```
--memtune hard_limit=4563402
```

- How much memory can be locked (by QEMU) in RAM, for the VM
  - Locked == never swapped to disk
  - All memory of SEV[-ES] guests must be locked
- Set this a little higher than guest RAM
  - QEMU may need to allocate more (e.g., MMIO regions, etc)

Some More Details

`driver.iommu=on`

For devices (e.g., disk, network, …)

- Otherwise, devices won't work inside the guest
  - VirtIO model: guest VirtIO drivers assume that the hypervisor can write to all of guest memory
    - But it can't! KVM and QEMU (on the host) cannot write to encrypted guest memory directly!
  - `iommu=on` enables using Linux kernels's DMA-API
- DMA data in SEV[-ES] can now go through unencrypted bounce buffers

Some More Details

```
--launchSecurity sev,polocy=0x3
```

- Enables SEV for the guest
  - Policy: bit-field, for enabling/disabling features
  - `0x03` SEV
  - `0x07` SEV-ES
- Choose always `0x3` (SEV) and change it to (SEV-ES) later
  - Currently, no support for reboot an SEV-ES guest
  - The installer may want to reboot!

Some More Details

`--boot ... ... ...`

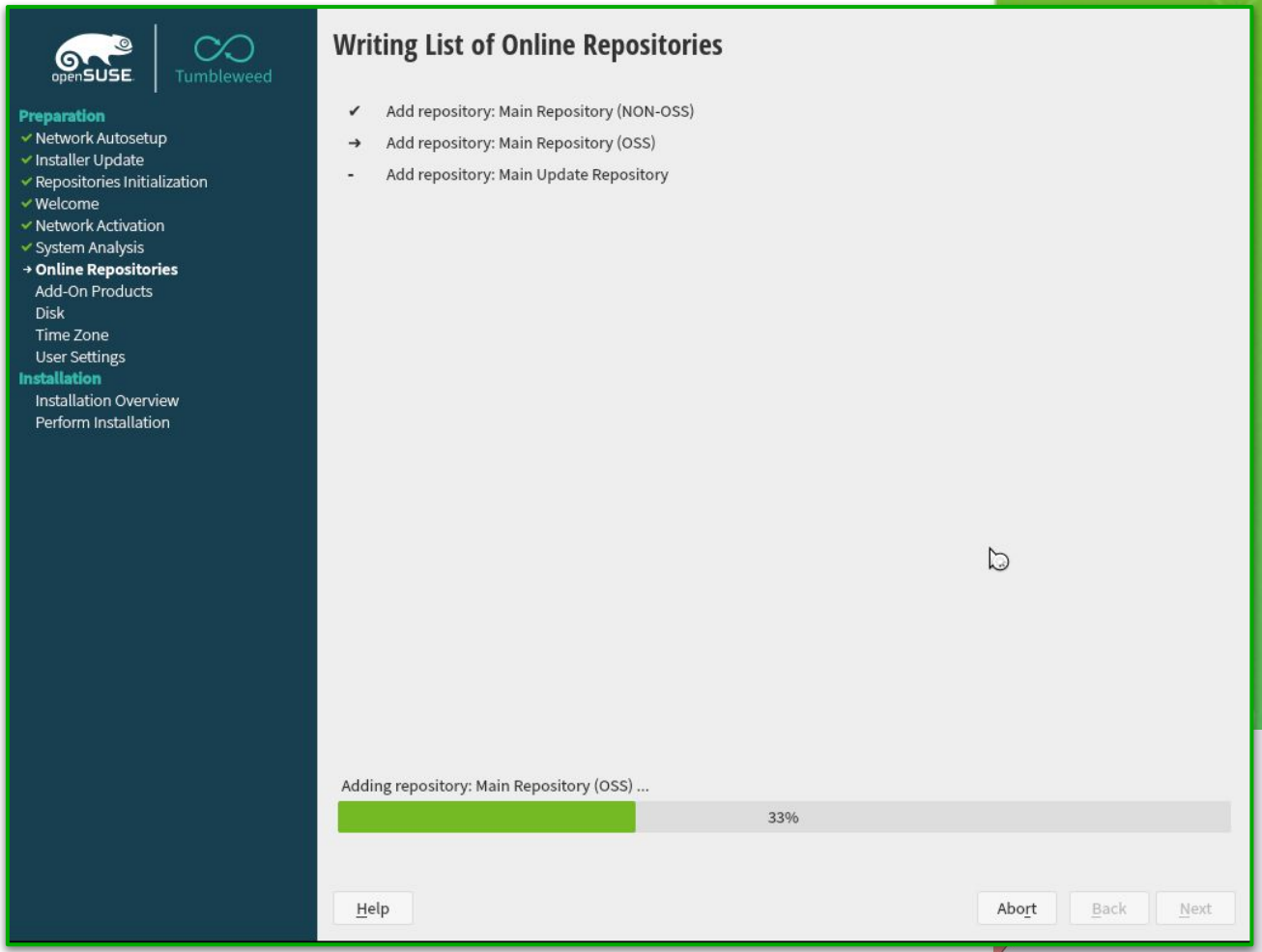
- Specify the firmware image
  - An OVMF UEFI BIOS, with SEV-ES support, is required.
  - Currently we need to specify it manually

Installation

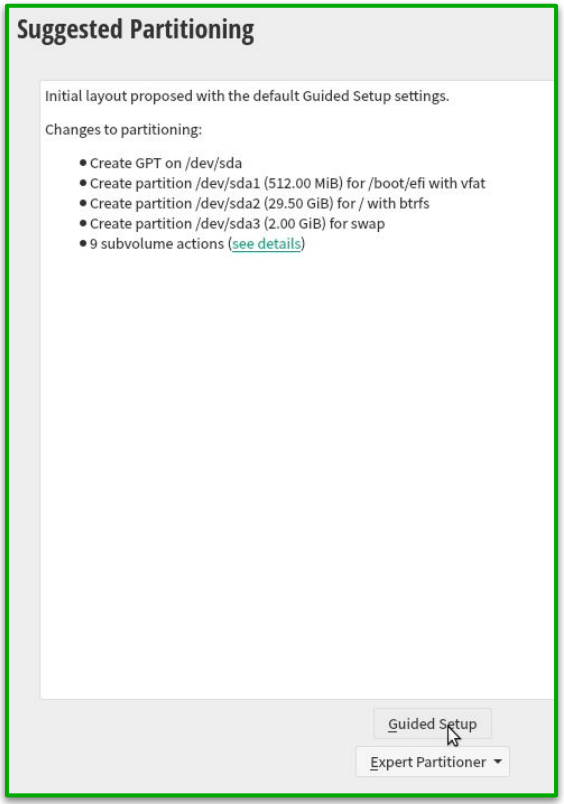
Just install openSUSE Tumbleweed normally

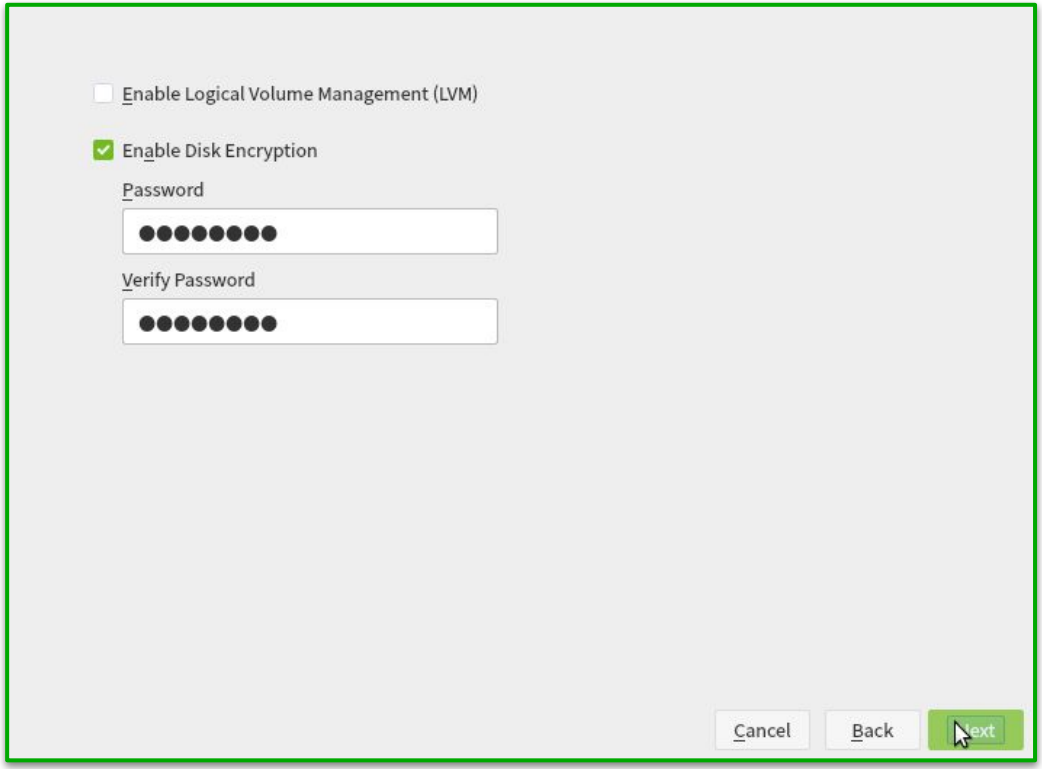
Remember to enable hard disk encryption

Not strictly required…
… But it does not make sense to have memory encrypted and not disk!



Installation: Hard Disk Encryption



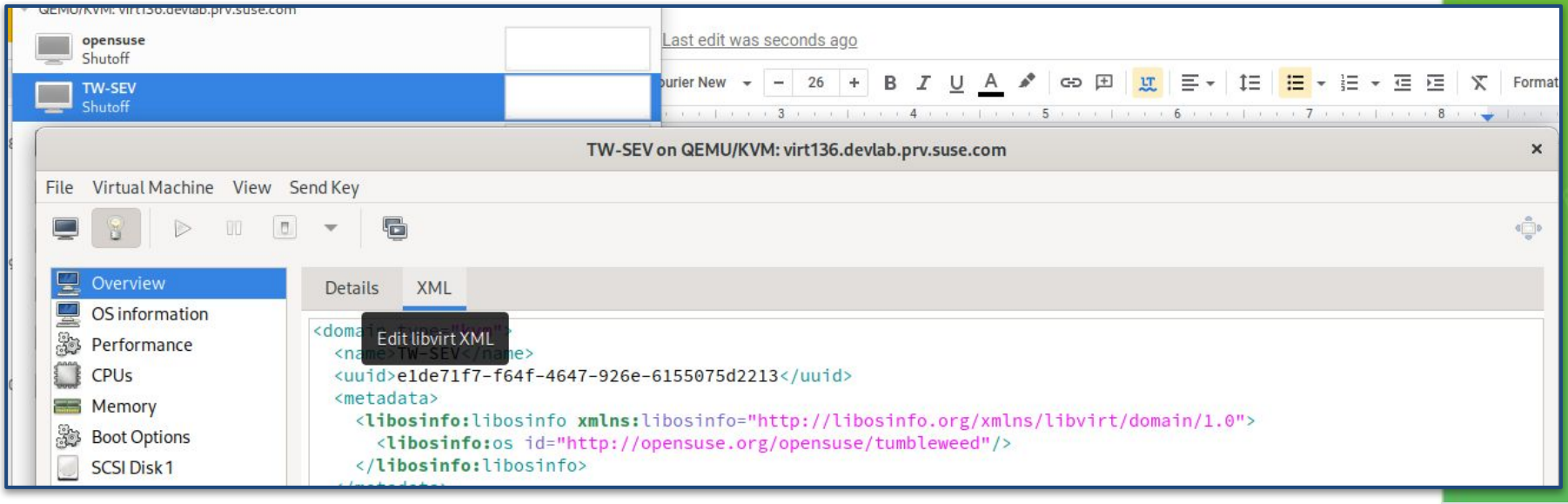


Installation: Hard Disk Encryption

For only typing the disk decryption password once, follow (post installation) either of these:

- Setting up Full Disk Encryption on openSUSE Tumbleweed
- SDB:Encrypted root file system

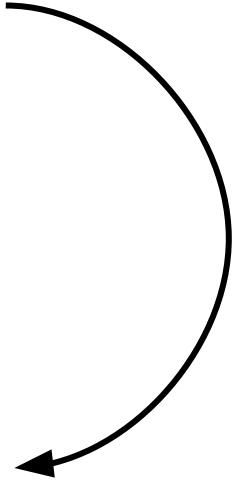
From SEV to SEV-ES

- `virsh edit TW-SEV`
- 

From SEV to SEV-ES

- Change this:

```
<launchSecurity type="sev">
  <cbitpos>51</cbitpos>
  <reducedPhysBits>1</reducedPhysBits>
  <policy>0x0003</policy>
</launchSecurity>
```



- Into this:

```
<launchSecurity type="sev">
  <cbitpos>51</cbitpos>
  <reducedPhysBits>1</reducedPhysBits>
  <policy>0x0007</policy>
</launchSecurity>
```

Did It Work ?

- SEV VM:

```
TW-SEV:~ # virt-what
kvm
TW-SEV:~ #
TW-SEV:~ # cat /etc/os-release | grep NAME
NAME="openSUSE Tumbleweed"
PRETTY_NAME="openSUSE Tumbleweed"
CPE_NAME="cpe:/o:opensuse:tumbleweed:20211019"
TW-SEV:~ #
TW-SEV:~ # dmesg | grep SEV
[    0.065250] AMD Memory Encryption Features active: SEV
```

Did It Work ?

- SEV-ES VM:

```
TW-SEV-ES:~ # virt-what
kvm
TW-SEV-ES:~ #
TW-SEV-ES:~ # cat /etc/os-release | grep NAME
NAME="openSUSE Tumbleweed"
PRETTY_NAME="openSUSE Tumbleweed"
CPE_NAME="cpe:/o:opensuse:tumbleweed:20211019"
TW-SEV-ES:~ #
TW-SEV-ES:~ # dmesg |grep SEV-ES
[    0.067175] AMD Memory Encryption Features active: SEV SEV-ES
```

In Summary

- With SEV[-ES] and (in a bit) SNP, VMs' memory can be encrypted
- Host / Hypervisor / Cloud Provider can't "spy" VMs
- We can stop having to trust the Host / the Hypervisor / the Cloud Provider
- Encryption happens in hardware
- We must continue to trust hardware
  - Shall we?

Some Links & References

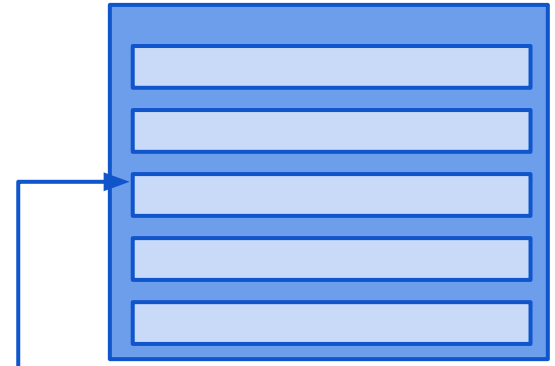
- Confidential Virtual Machines with AMD SEV-ES and openSUSE Tumbleweed - Joerg Rodel
- Confidential Virtual Machines with SEV and SNP - Joerg Roedel
- Running Encrypted VMs on openSUSE Tumbleweed
- Setting up Full Disk Encryption on openSUSE Tumbleweed


openSUSE

Some More Technical Details

Virtualization 101

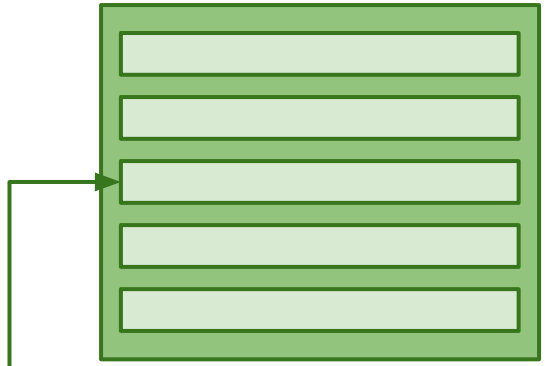
VM



Virtual Registers

- State of the physical CPU's register while the VM is running

Physical CPU

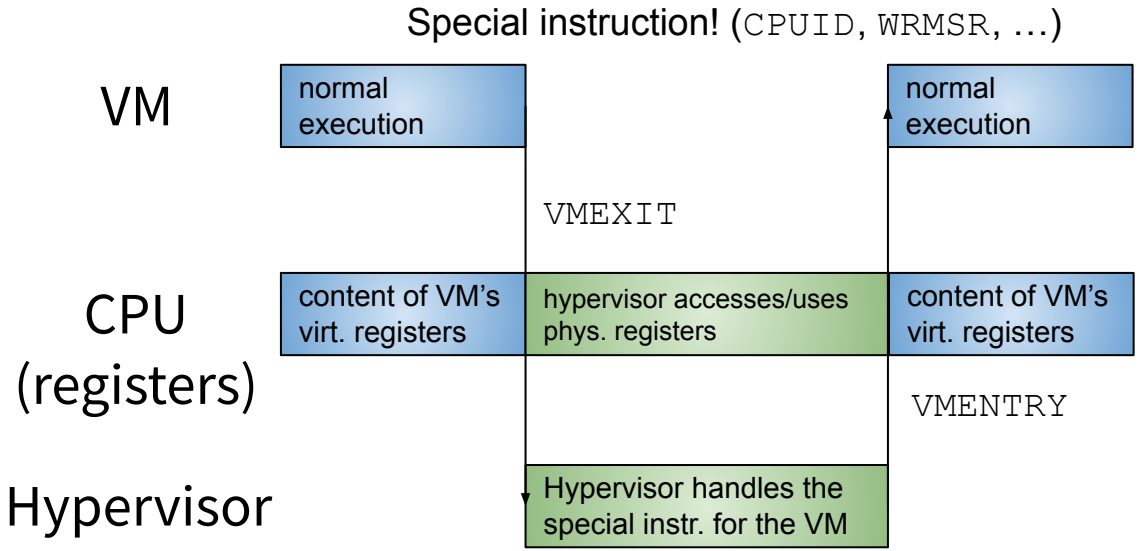


Registers

- Host data that the CPU elaborates

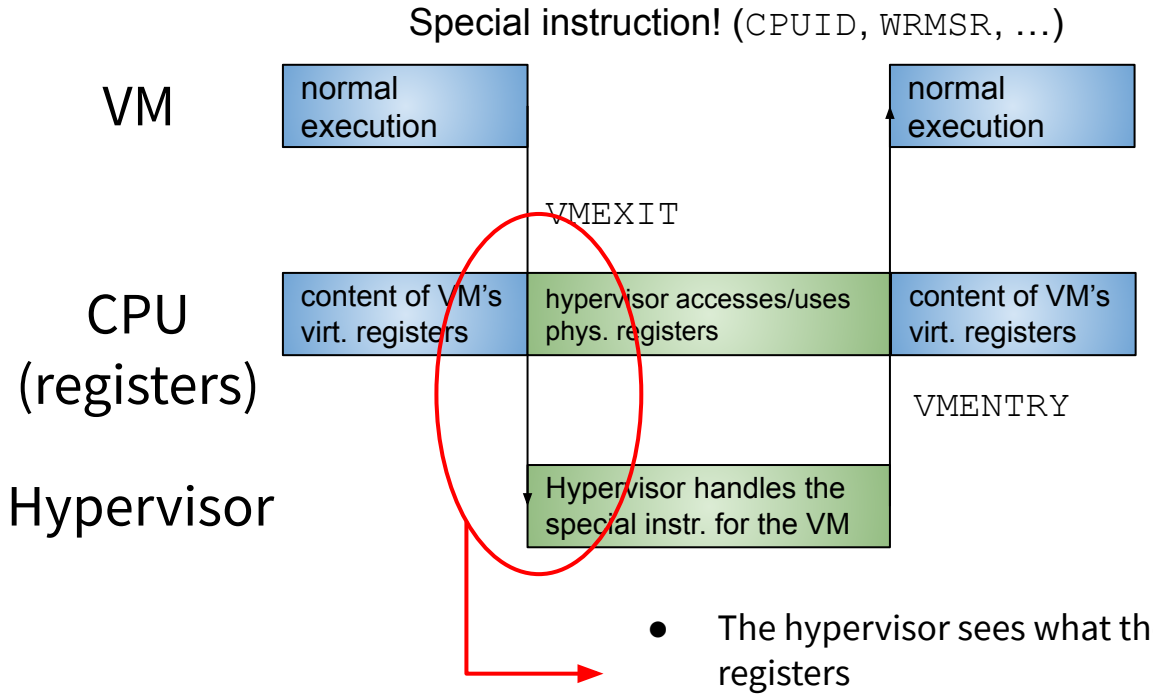
Some More Technical Details

Virtualization 101



Some More Technical Details

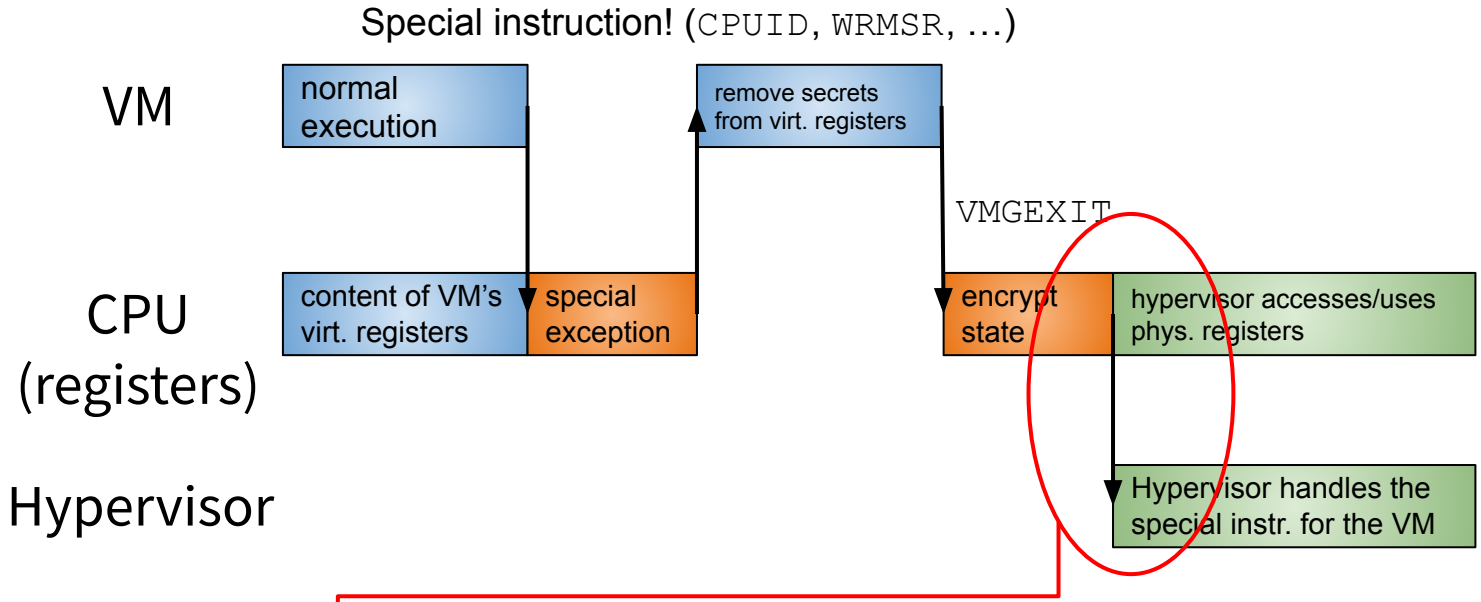
Virtualization 101



- The hypervisor sees what the VM, while it was running put in registers
- What if there are secrets?

Some More Technical Details

Virtualization 101



- Hypervisor only sees what the VM left there
- Only what it needs for handling the special instruction/event
- hopefully, no secrets there!