# Introdução à Virtualização

Gabriel Negreira Barbosa

gabriel.negreira.barbosa [arroba] intel.com

# Importante

- Eu não falo pelo meu empregador
  - Todas as ideias e informações presentes nessa apresentação são de minha inteira responsabilidade

- Não espere o fim da palestra para fazer perguntas!

# Introdução

- Virtualização cada vez mais utilizada:
  - Sistemas operacionais modernos
  - Malware analysis
  - Cloud
  - Etc
- Virtualizar é um desafio!
- Mas como realmente funcionam os sistemas modernos de virtualização?

# Objetivos

- Introduzir o que acontece por trás dos sistemas de virtualização
  - Foco: Intel VMX / VT-x

- Discutir aspectos de segurança em sistemas de virtualização

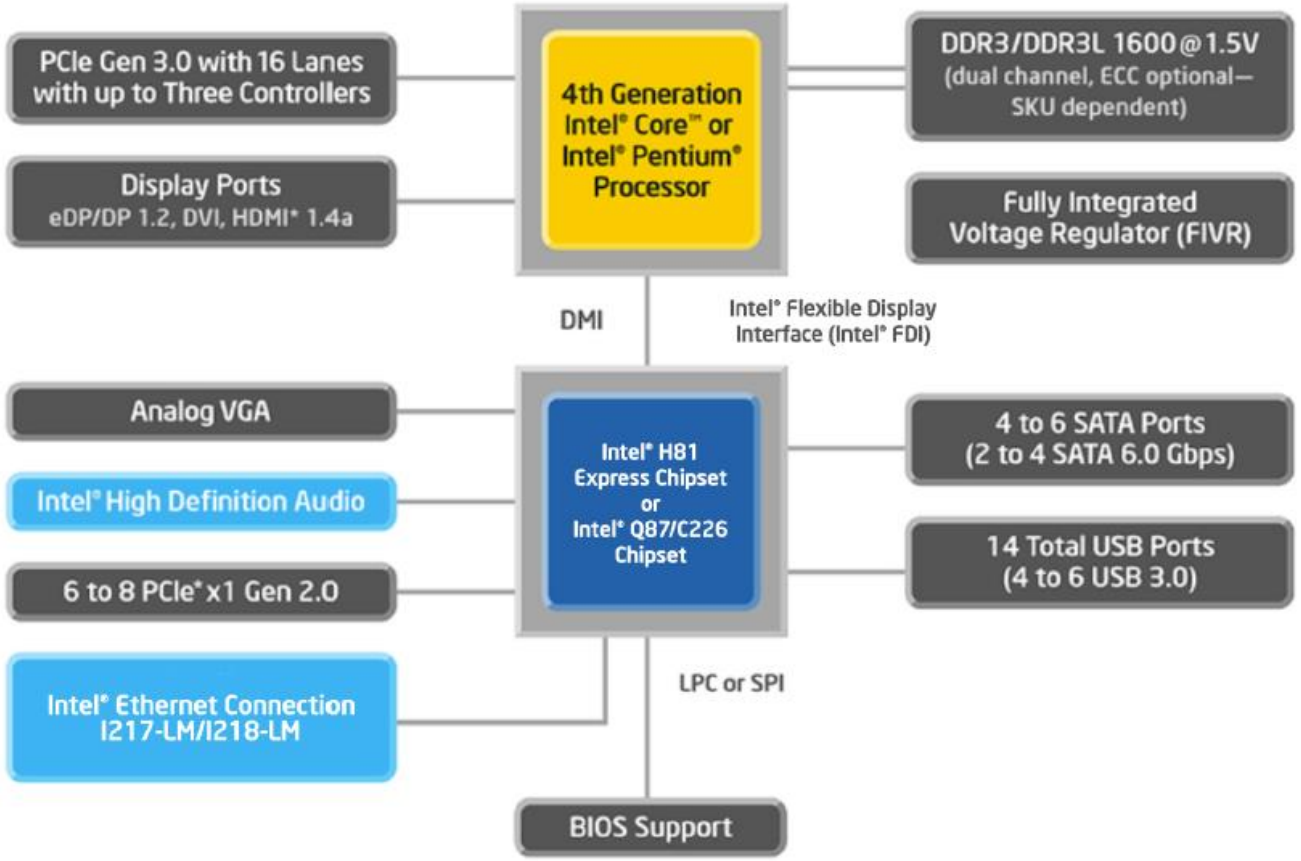
# Agenda

- Discussão de conceitos de virtualização
- Aspectos de segurança
- Conclusão

# Agenda

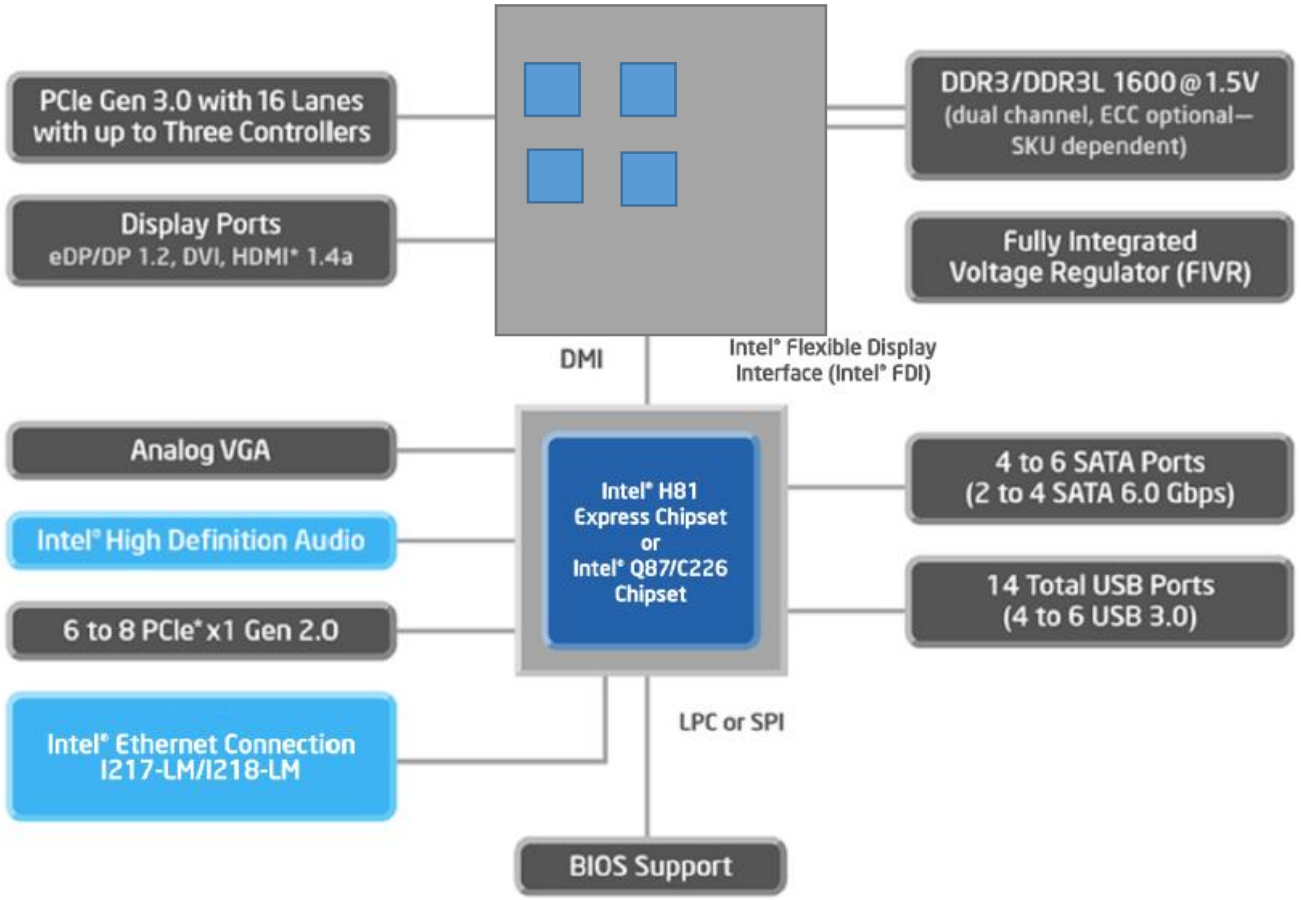
- Discussão de conceitos de virtualização
- Aspectos de segurança
- Conclusão

# Arquitetura Geral




Source: http://www.intel.com/content/www/us/en/intelligent-systems/embedded-systems-training/ia-introduction-basics-paper.html

# Arquitetura Geral – VMX

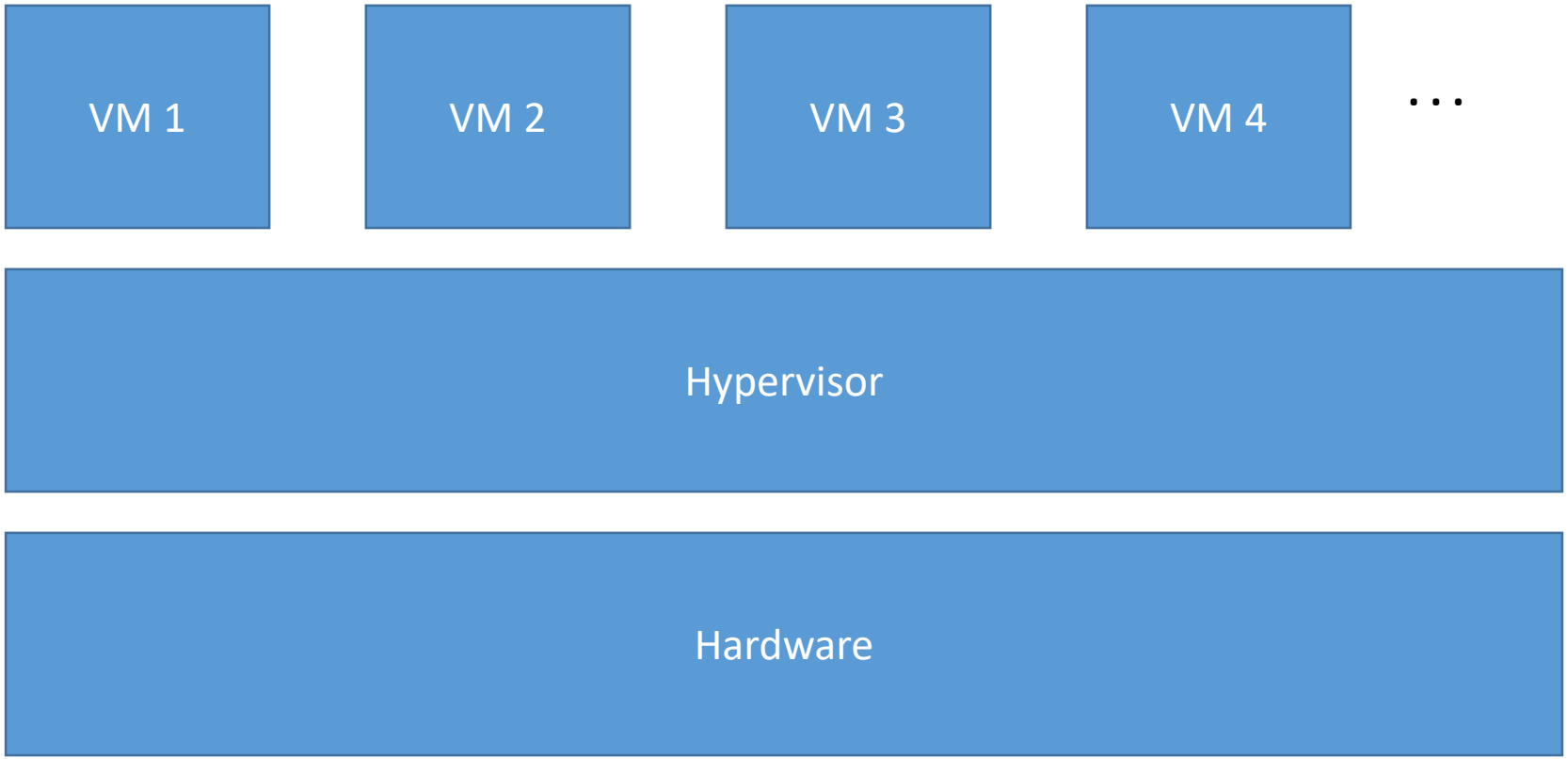



Source: http://www.intel.com/content/www/us/en/intelligent-systems/embedded-systems-training/ia-introduction-basics-paper.html

# VMX (Virtual-Machine Extensions)

- VMX Root
  - Comportamento muito parecido como não-VMX
    - Exemplo de diferença: instruções VMX
  - Em geral, VMM (Virtual Machine Monitor) / **hypervisor** roda como VMX root
- VMX Non-Root
  - Ambiente com restrições e modificações para facilitar a virtualização
    - Exemplos ao decorrer da apresentação
  - Em geral, **máquinas virtuais** rodam como VMX non-root

# Arquitetura em Alto Nível



# Transições VMX

- VM Entry
  - Transições para VMX Non-root
  - Ocorre pelas instruções VMLAUNCH e VMRESUME
- VM Exit
  - Transições para VMX root
  - Ocorre por certas instruções (por exemplo, VMCALL) e eventos quando em VMX non-root
    - Mais exemplos ao decorrer da apresentação

# Ciclo de Vida do VMM

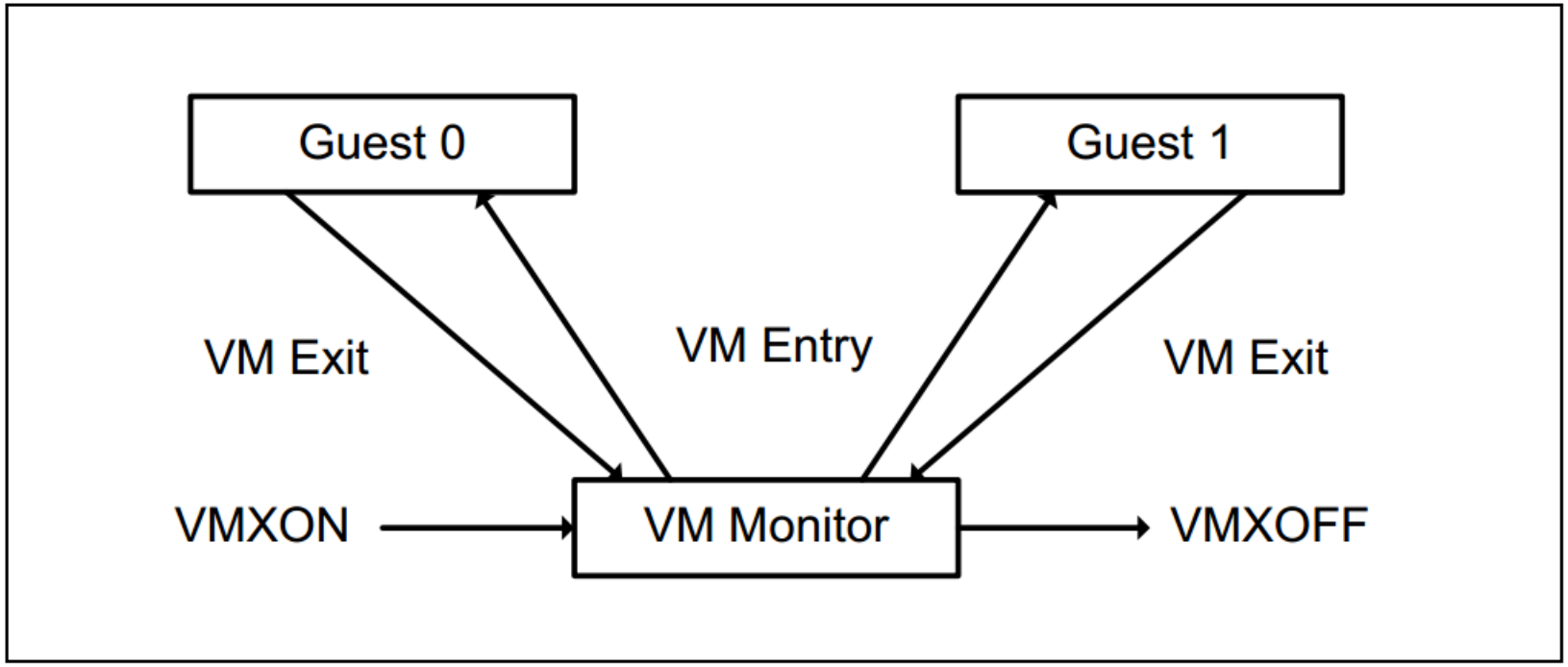
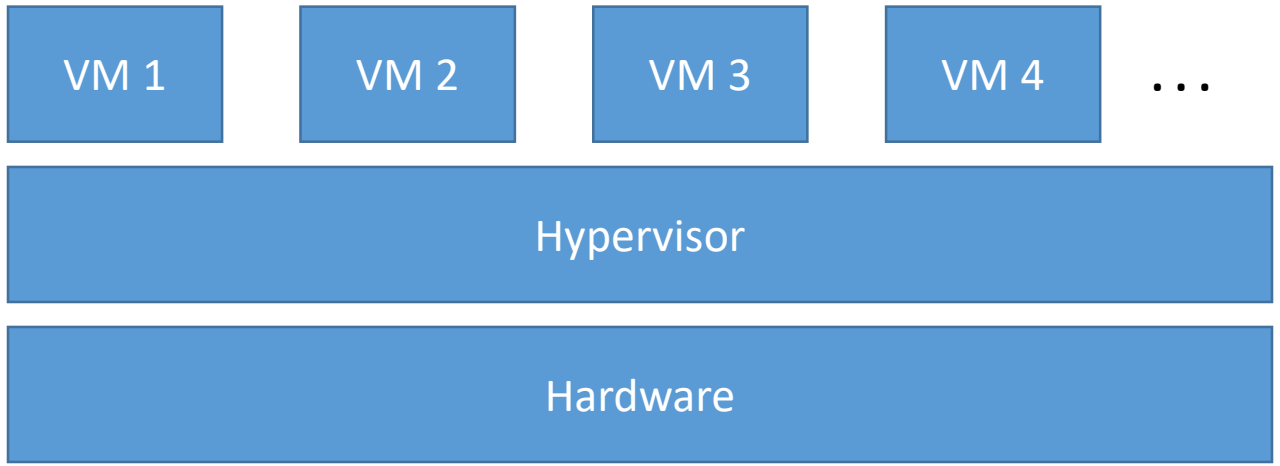



Figure 23-1. Interaction of a Virtual-Machine Monitor and Guests

# Arquitetura em Alto Nível




- Como o hypervisor pode controlar o que causará VM Exit?

# Intel VMCS (Virtual Machine Control Structure)

- Gerenciamento de transições VMX
  - VM Entry e VM Exit
- Gerenciamento do comportamento do processador em VMX non-root
- Diferentes VMCS podem ser associados à mesma VM
- VMCS region: área de memória alocada para um dado VMCS
- VMPTRLD, VMPTRST, VMWRITE, VMREAD, VMCLEAR

Table 24-1. Format of the VMCS Region

| Byte Offset | Contents |
|---|---|
| 0 | Bits 30:0: VMCS revision identifier<br>Bit 31: shadow-VMCS indicator (see Section 24.10) |
| 4 | VMX-abort indicator |
| 8 | VMCS data (implementation-specific format) |

# Intel VMCS

- VMCS data está organizado em 6 grupos:
  - Guest-state area
    - Estado do processador é salvo aqui em VM Exit e restaurado daqui em VM Entry
  - Host-state area
    - Estado do processador é carregado daqui quando ocorre VM Exit
  - VM-execution control fields
    - Controla o comportamento do processador quando em VMX non-root e algumas causas de VM Exit
  - VM-exit control fields
    - Controla VM Exits
  - VM-entry control fields
    - Controla VM Entries
  - VM-exit information fields
    - Informações sobre a causa e natureza do VM Exit

# Intel VMCS – Guest-State Area – Exemplos

- Control registers CR0, CR3 e CR4
- RSP, RIP, RFLAGS
- Alguns campos de CS e SS
- Alguns MSRs

# Intel VMCS – Host-State Area – Exemplos

- Control registers CR0, CR3 e CR4
- RSP, RIP
- Alguns campos de CS e SS
- Alguns MSRs

# Intel VMCS – VM-Execution Control Fields – Exemplos

- I/O bitmap
- MSR bitmap
- Definições de certos eventos e instruções que geram VM Exit

# Intel VMCS – VM-Exit Control Fields – Exemplos

- VM-exit MSR-load count
- VM-exit MSR-load address

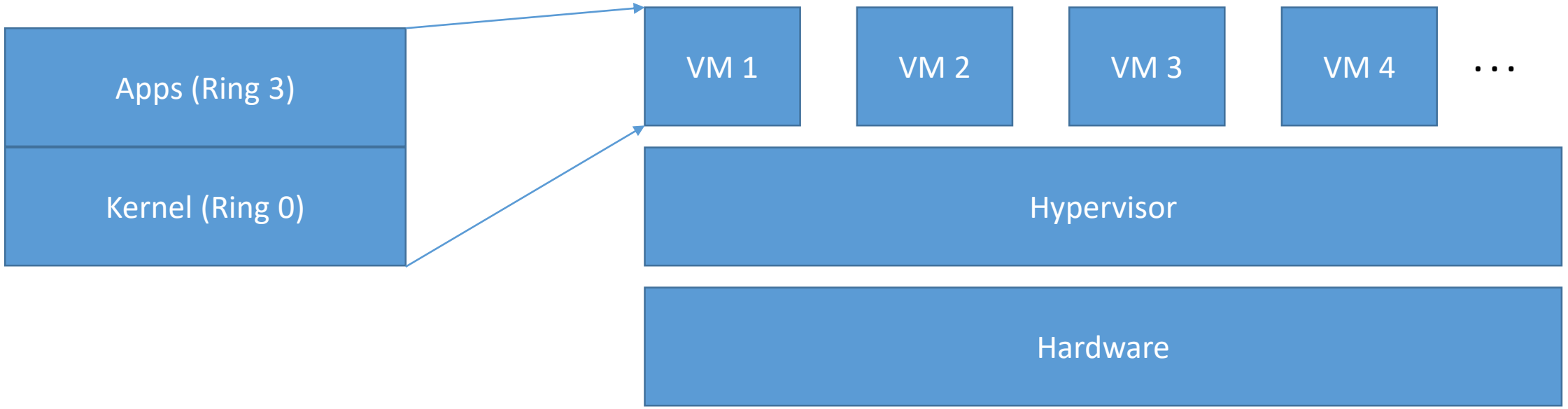
# Intel VMCS – VM-Entry Control Fields – Exemplos

- VM-entry MSR-load count
- VM-entry MSR-load address

# Intel VMCS – VM-Exit Information Fields – Exemplos

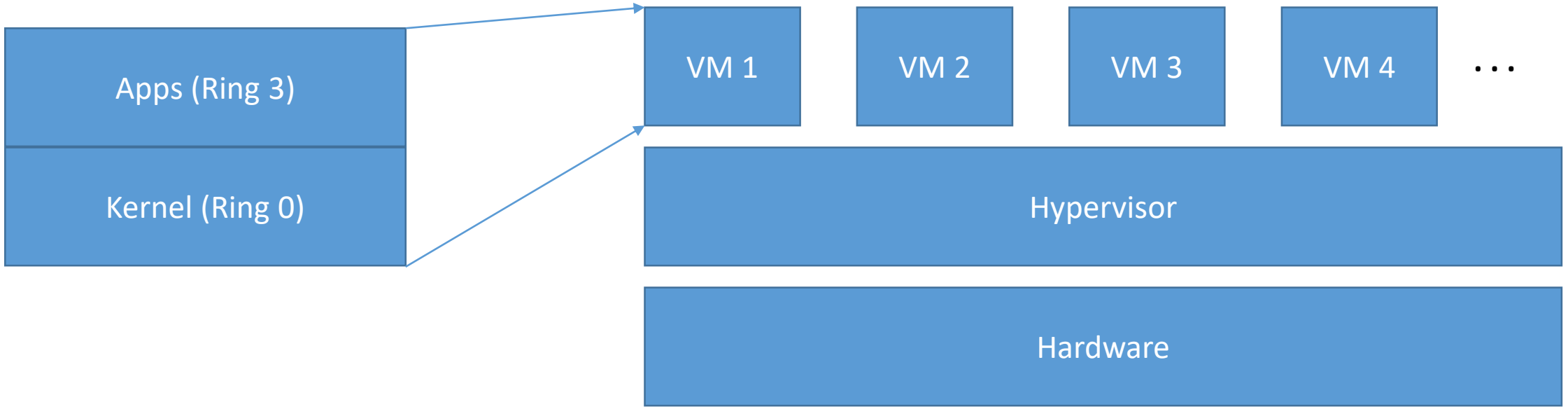
- Motivo do VM-Exit ☺

# Memória




**E se o kernel de uma VM configurar suas tabelas de página para acessar um endereço físico do hypervisor ou de outra VM?**

# Intel EPT (Extended Page-Table) – Visão de Alto Nível



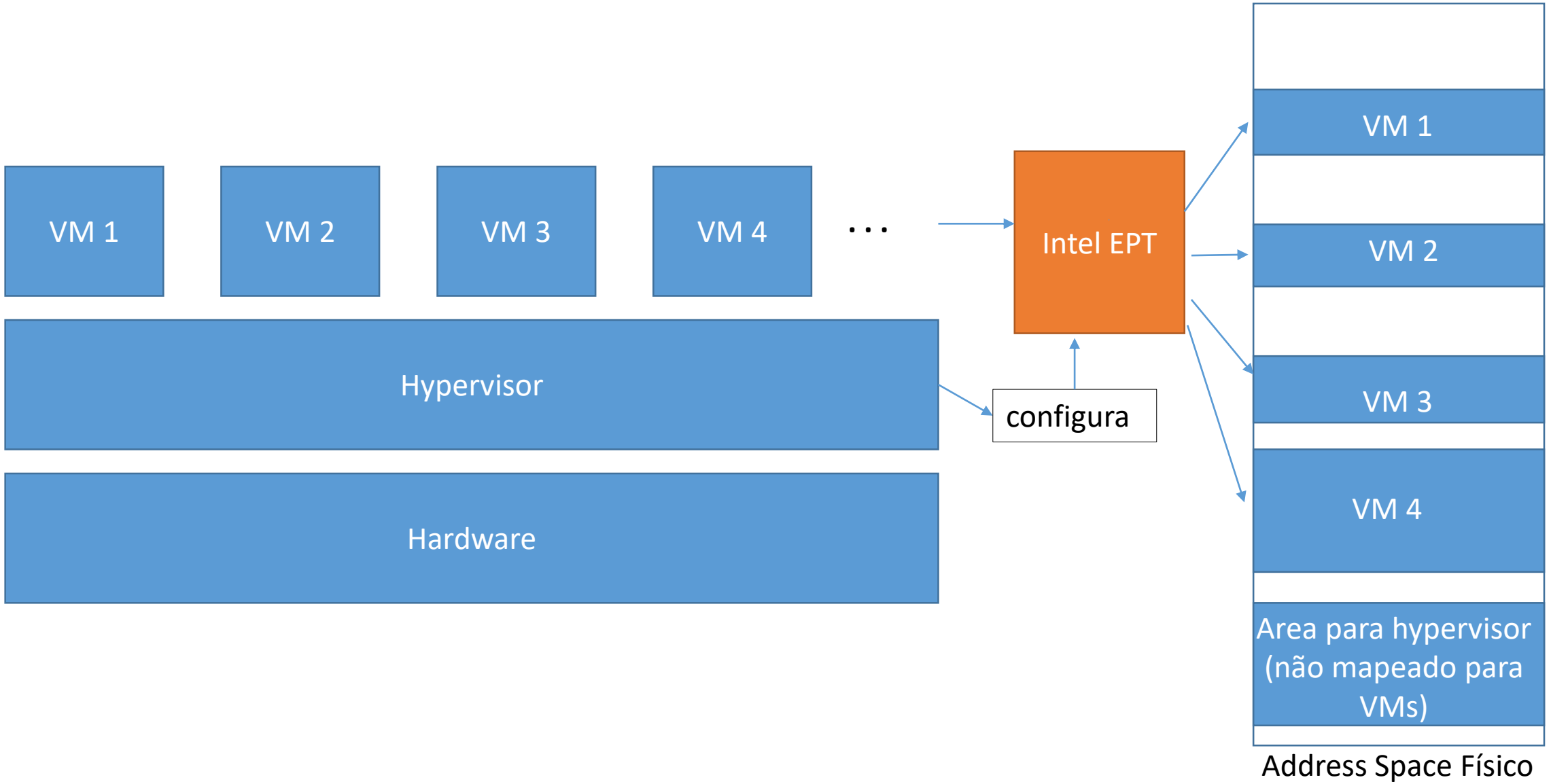

| Guest Physical Address (GPA) | System Physical Address (SPA) | Atributos |
|---|---|---|
| 0x0 | 0x12345678 | Read-Only |
| 0xdeadbeef | Not mapped | - |

# Intel EPT (Extended Page Tables)

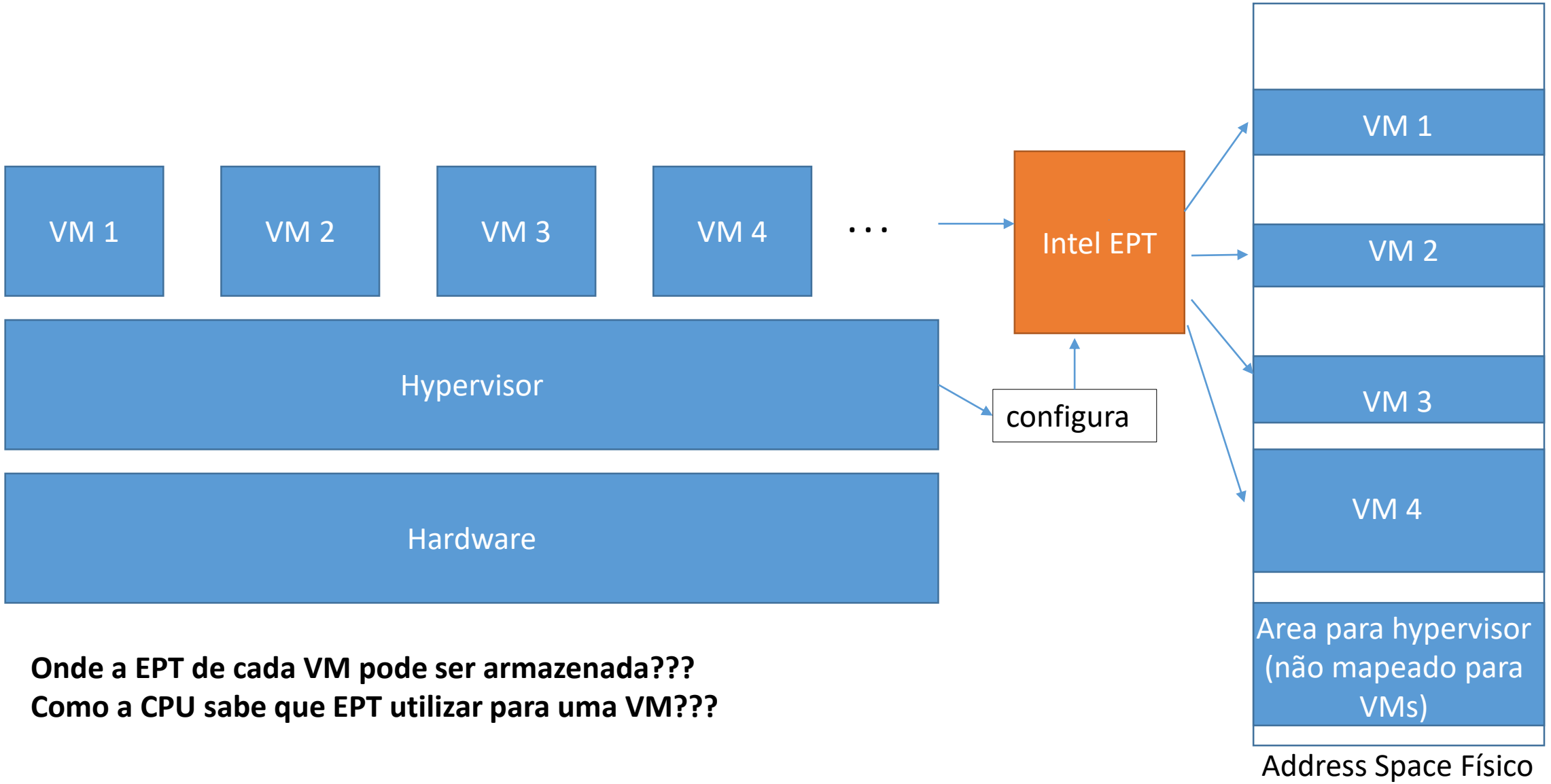
| 63 62 61 60 59 58 57 56 55 54 53 52 51 | M[1] | M-1 | 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 | 9 8 7 | 6 | 5 4 3 | 2 1 0 | |
|---|---|---|---|---|---|---|---|---|
| Reserved | | | Address of EPT PML4 table | Rsvd. | A/D | EPT PWL-1 | EPT PS MT | EPTP[2] |
| Ignored | Rsvd.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

Figure 28-1. Formats of EPTP and EPT Paging-Structure Entries

# EPT na Prática (Alto Nível)

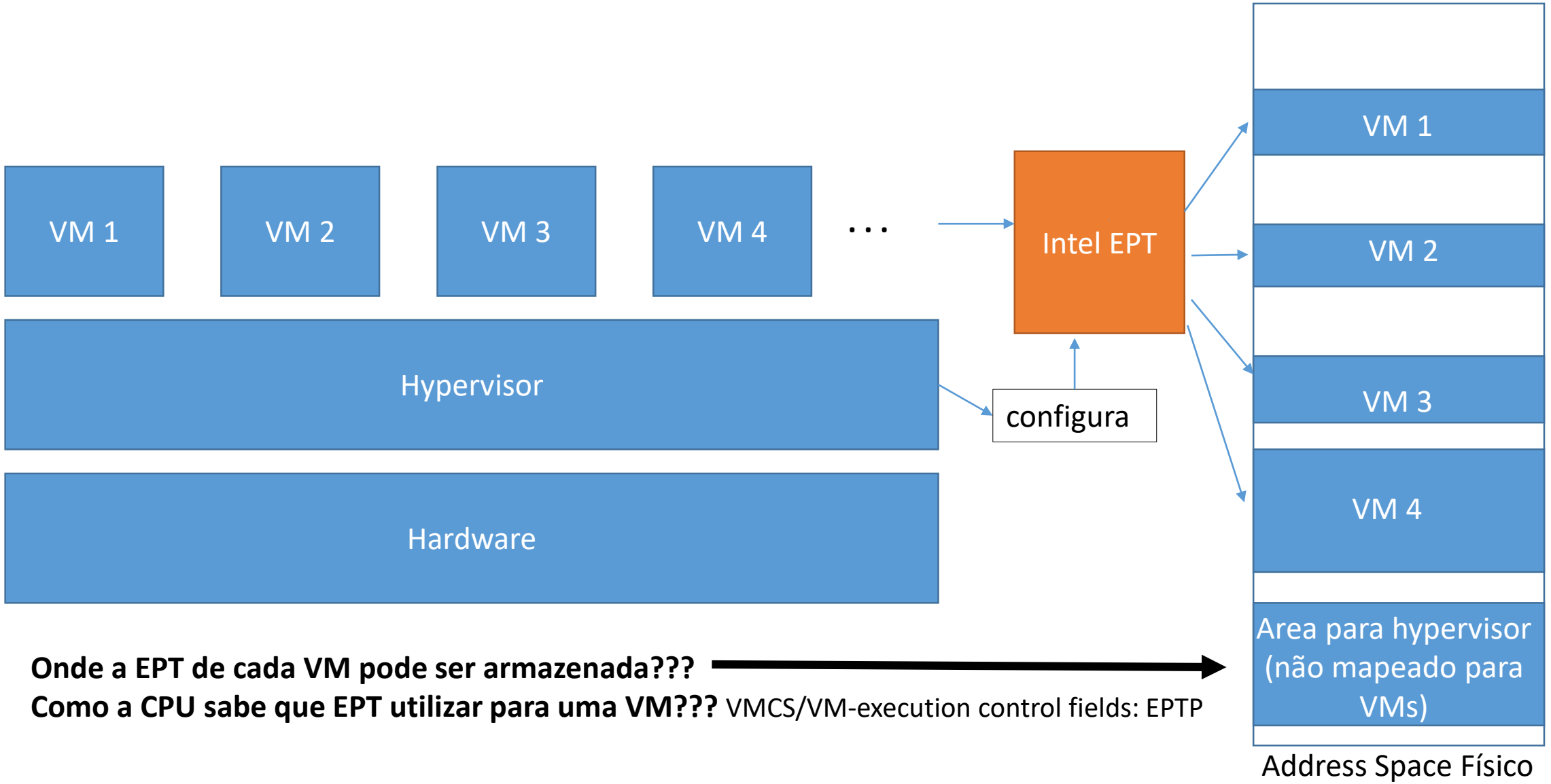


# EPT na Prática (Alto Nível)




**Onde a EPT de cada VM pode ser armazenada???**
**Como a CPU sabe que EPT utilizar para uma VM???**

# EPT na Prática (Alto Nível)




**Onde a EPT de cada VM pode ser armazenada???**

**Como a CPU sabe que EPT utilizar para uma VM???** VMCS/VM-execution control fields: EPTP

# Instruções VMX

- **VMLAUNCH —** This instruction launches a virtual machine managed by the VMCS. A VM entry occurs, transferring control to the VM.
- **VMRESUME —** This instruction resumes a virtual machine managed by the VMCS. A VM entry occurs, transferring control to the VM.
- **VMXOFF —** This instruction causes the processor to leave VMX operation.
- **VMXON —** This instruction takes a single 64-bit source operand that is in memory. It causes a logical processor to enter VMX root operation and to use the memory referenced by the operand to support VMX operation.
- **VMCALL —** This instruction allows software in VMX non-root operation to call the VMM for service. A VM exit occurs, transferring control to the VMM.

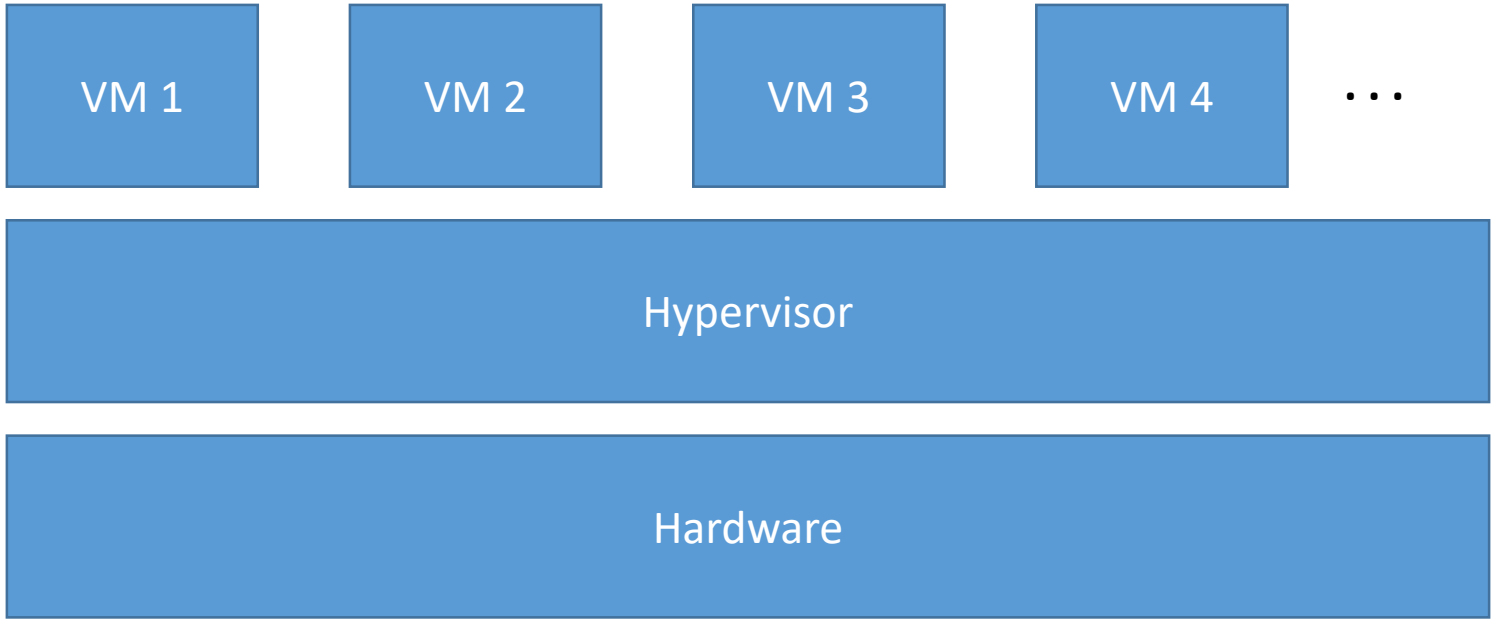
# Instruções VMX – VMCS

- **VMPTRLD —** This instruction takes a single 64-bit source operand that is in memory. It makes the referenced VMCS active and current, loading the current-VMCS pointer with this operand and establishes the current VMCS based on the contents of VMCS-data area in the referenced VMCS region. Because this makes the referenced VMCS active, a logical processor may start maintaining on the processor some of the VMCS data for the VMCS.
- **VMPTRST —** This instruction takes a single 64-bit destination operand that is in memory. The current-VMCS pointer is stored into the destination operand.
- **VMCLEAR —** This instruction takes a single 64-bit operand that is in memory. The instruction sets the launch state of the VMCS referenced by the operand to "clear", renders that VMCS inactive, and ensures that data for the VMCS have been written to the VMCS-data area in the referenced VMCS region. If the operand is the same as the current-VMCS pointer, that pointer is made invalid.
- **VMREAD —** This instruction reads a component from a VMCS (the encoding of that field is given in a register operand) and stores it into a destination operand that may be a register or in memory.
- **VMWRITE —** This instruction writes a component to a VMCS (the encoding of that field is given in a register operand) from a source operand that may be a register or in memory.

# Agenda

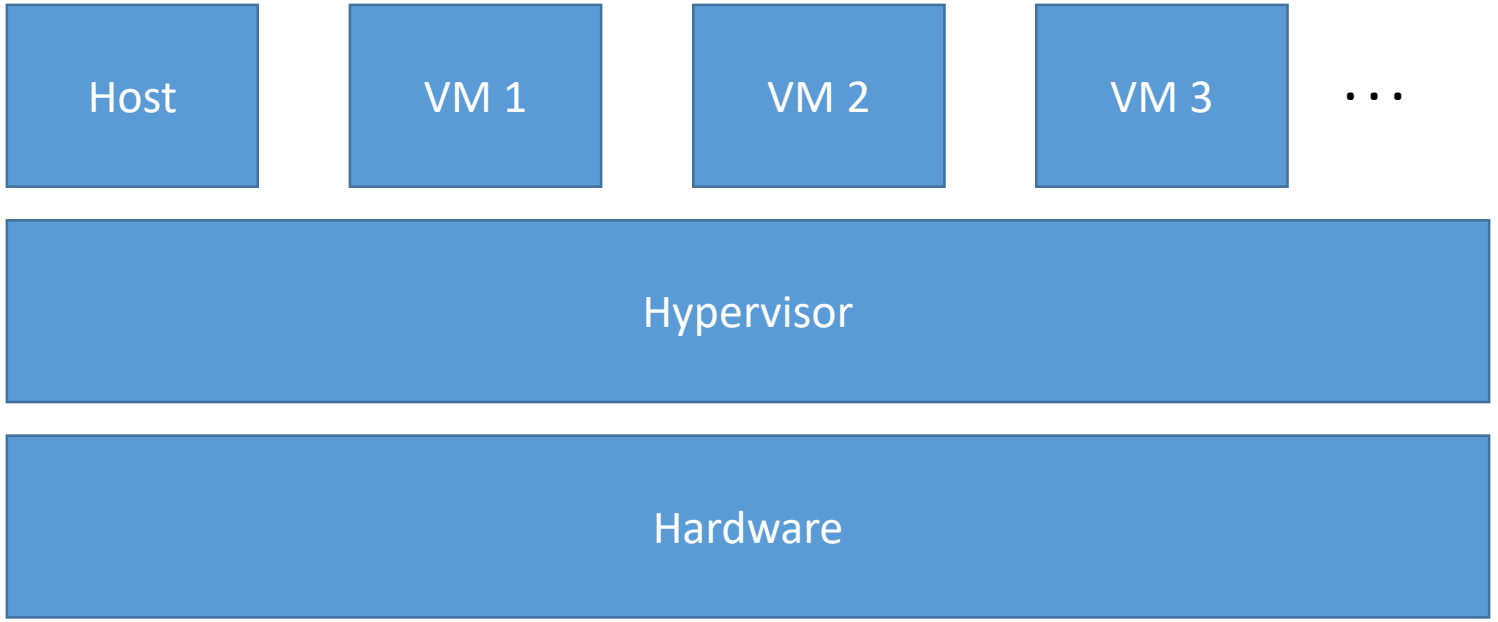
- Discussão de conceitos de virtualização
- Aspectos de segurança
- Conclusão

# Host



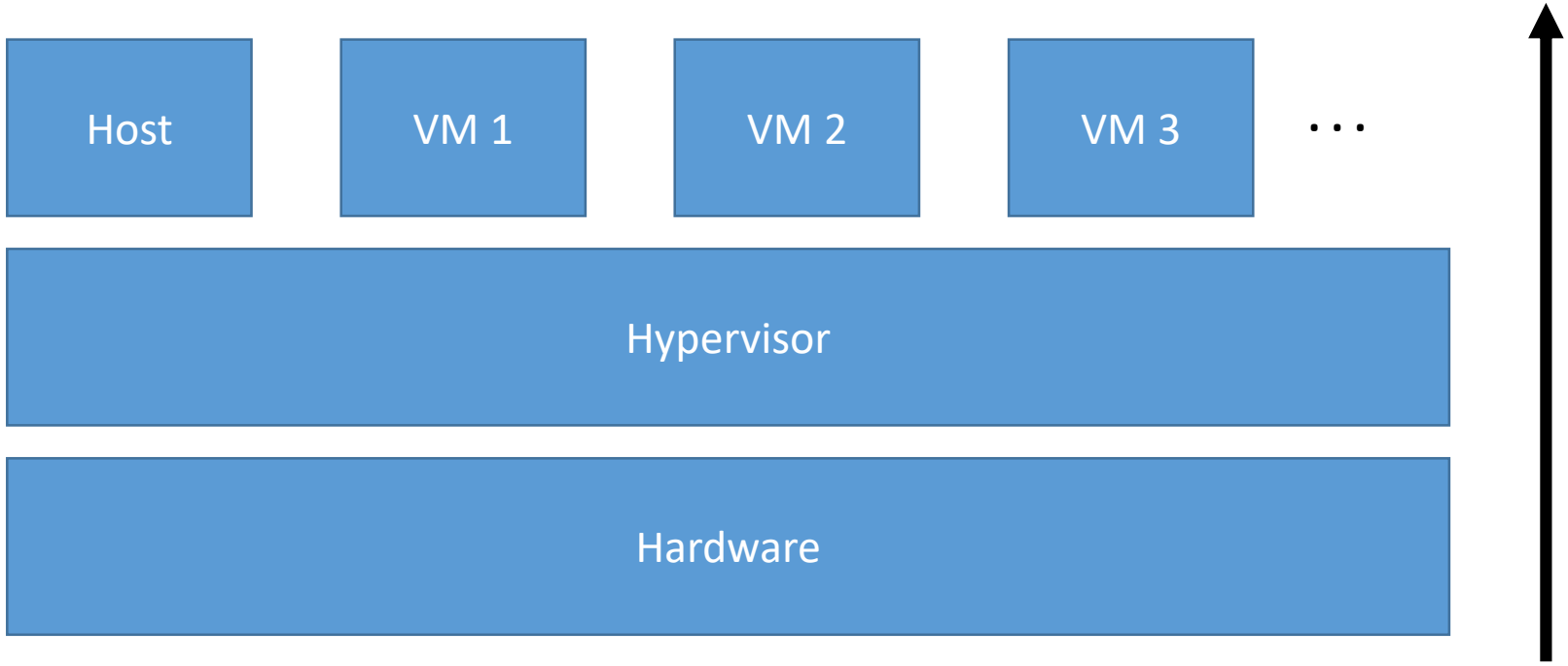

**Onde está o “Windows”???**

# Host



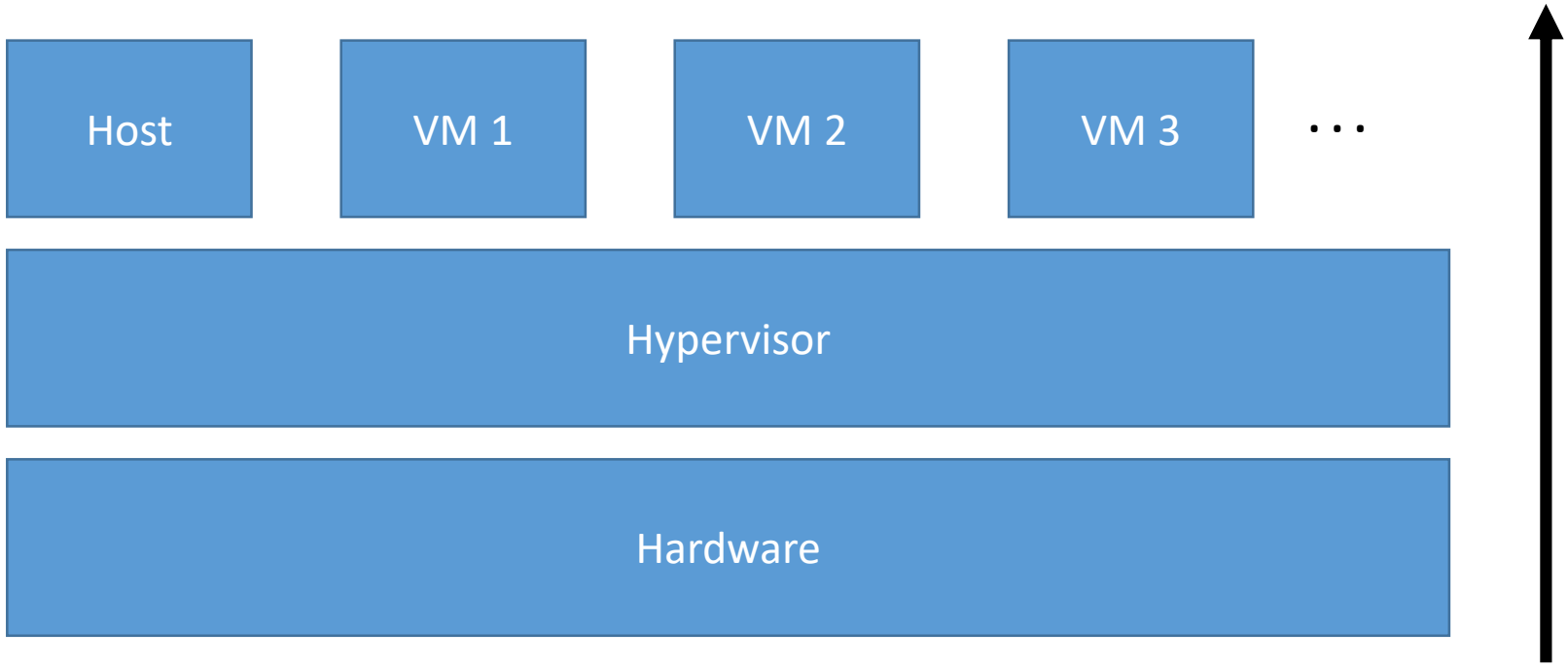

**Em versões modernas, o Windows é uma VM!**
**(Mais detalhes ao longo da apresentação)**

# Attack Surface – Brainstorm



# Attack Surface – Brainstorm

- VM to VM
- VM to Host
- VM to Hypervisor
- DoS
- *Host to Hypervisor*
- Hardware




**Hypervisor to Guest???**
**Host to Guest?**

# Attack Surface – Mais detalhes

- VM Exit handling
- Hypercall
- Hardware bug
- Hypervisor backdoor
- Configuração da EPT
  - Mapeamento
  - Dados previamente armazenados nas páginas
- Configuração do VMCS

# Attack Surface – “VMEXIT handling”

- https://xenbits.xen.org/xsa/advisory-239.html

```
--- a/xen/arch/x86/hvm/emulate.c
+++ b/xen/arch/x86/hvm/emulate.c
@@ -129,7 +129,7 @@ static int hvmemul_do_io(
         .count = *reps,
         .dir = dir,
         .df = df,
-        .data = data,
+        .data = data_is_addr ? data : 0,
         .data_is_ptr = data_is_addr, /* ioreq_t field name is misleading */
         .state = STATE_IOREQ_READY,
     };
--- a/xen/arch/x86/hvm/intercept.c
+++ b/xen/arch/x86/hvm/intercept.c
@@ -127,6 +127,7 @@ int hvm_process_io_intercept(const struc
             addr = (p->type == IOREQ_TYPE_COPY) ?
                    p->addr + step * i :
                    p->addr;
+            data = 0;
             rc = ops->read(handler, addr, p->size, &data);
             if ( rc != X86EMUL_OKAY )
                 break;
@@ -161,6 +162,7 @@ int hvm_process_io_intercept(const struc
         {
             if ( p->data_is_ptr )
             {
+                data = 0;
                 switch ( hvm_copy_from_guest_phys(&data, p->data + step * i,
                                                   p->size) )
                 {
```

# Attack Surface – Hypercall

- https://xenbits.xen.org/xsa/advisory-122.html

```
--- a/xen/common/kernel.c
+++ b/xen/common/kernel.c
@@ -240,6 +240,8 @@ DO(xen_version)(int cmd, XEN_GUEST_HANDL
     case XENVER_extraversion:
     {
         xen_extraversion_t extraversion;
+
+        memset(extraversion, 0, sizeof(extraversion));
         safe_strcpy(extraversion, xen_extra_version());
         if ( copy_to_guest(arg, extraversion, ARRAY_SIZE(extraversion)) )
             return -EFAULT;
@@ -249,6 +251,8 @@ DO(xen_version)(int cmd, XEN_GUEST_HANDL
     case XENVER_compile_info:
     {
         struct xen_compile_info info;
+
+        memset(&info, 0, sizeof(info));
         safe_strcpy(info.compiler,       xen_compiler());
         safe_strcpy(info.compile_by,     xen_compile_by());
         safe_strcpy(info.compile_domain, xen_compile_domain());
@@ -284,6 +288,8 @@ DO(xen_version)(int cmd, XEN_GUEST_HANDL
     case XENVER_changeset:
     {
         xen_changeset_info_t chgset;
+
+        memset(chgset, 0, sizeof(chgset));
         safe_strcpy(chgset, xen_changeset());
         if ( copy_to_guest(arg, chgset, ARRAY_SIZE(chgset)) )
             return -EFAULT;
```

# Attack Surface – CPU bug

| | |
|---|---|
| **SKL031** | **A VMX Transition Attempting to Load a Non-Existent MSR May Result in a Shutdown** |
| **Problem** | A VMX transition may result in a shutdown (without generating a machine-check event) if a non-existent MSR is included in the associated MSR-load area. When such a shutdown occurs, a machine check error will be logged with IA32_MCi_STATUS.MCACOD (bits [15:0]) of 406H, but the processor does not issue the special shutdown cycle. A hardware reset must be used to restart the processor. |
| **Implication** | Due to this erratum, the hypervisor may experience an unexpected shutdown. |
| **Workaround** | Software should not configure VMX transitions to load non-existent MSRs. |
| **Status** | For the steppings affected, see the Summary Table of Changes. |

Source: http://www.intel.com/content/www/us/en/processors/core/desktop-6th-gen-core-family-spec-update.html

# Attack Surface – Backdoor

Table 24-7. Definitions of Secondary Processor-Based VM-Execution Controls

| Bit Position(s) | Name | Description |
|---|---|---|
| 0 | Virtualize APIC accesses | If this control is 1, the logical processor treats specially accesses to the page with the APIC-access address. See Section 29.4. |
| 1 | Enable EPT | If this control is 1, extended page tables (EPT) are enabled. See Section 28.2. |
| 2 | Descriptor-table exiting | This control determines whether executions of LGDT, LIDT, LLDT, LTR, SGDT, SIDT, SLDT, and STR cause VM exits. |
| 3 | Enable RDTSCP | If this control is 0, any execution of RDTSCP causes an invalid-opcode exception (#UD). |
| 4 | Virtualize x2APIC mode | If this control is 1, the logical processor treats specially RDMSR and WRMSR to APIC MSRs (in the range 800H–8FFH). See Section 29.5. |
| 5 | Enable VPID | If this control is 1, cached translations of linear addresses are associated with a virtual-processor identifier (VPID). See Section 28.1. |
| 6 | WBINVD exiting | This control determines whether executions of WBINVD cause VM exits. |
| 7 | Unrestricted guest | This control determines whether guest software may run in unpaged protected mode or in real-address mode. |
| 8 | APIC-register virtualization | If this control is 1, the logical processor virtualizes certain APIC accesses. See Section 29.4 and Section 29.5. |
| 9 | Virtual-interrupt delivery | This controls enables the evaluation and delivery of pending virtual interrupts as well as the emulation of writes to the APIC registers that control interrupt prioritization. |
| 10 | PAUSE-loop exiting | This control determines whether a series of executions of PAUSE can cause a VM exit (see Section 24.6.13 and Section 25.1.3). |
| 11 | RDRAND exiting | This control determines whether executions of RDRAND cause VM exits. |
| 12 | Enable INVPCID | If this control is 0, any execution of INVPCID causes a #UD. |
| 13 | Enable VM functions | Setting this control to 1 enables use of the VMFUNC instruction in VMX non-root operation. See Section 25.5.5. |
| 14 | VMCS shadowing | If this control is 1, executions of VMREAD and VMWRITE in VMX non-root operation may access a shadow VMCS (instead of causing VM exits). See Section 24.10 and Section 30.3. |
| 16 | RDSEED exiting | This control determines whether executions of RDSEED cause VM exits. |
| 17 | Enable PML | If this control is 1, an access to a guest-physical address that sets an EPT dirty bit first adds an entry to the page-modification log. See Section 28.2.5. |
| 18 | EPT-violation #VE | If this control is 1, EPT violations may cause virtualization exceptions (#VE) instead of VM exits. See Section 25.5.6. |
| 20 | Enable XSAVES/XRSTORS | If this control is 0, any execution of XSAVES or XRSTORS causes a #UD. |
| 25 | Use TSC scaling | This control determines whether executions of RDTSC, executions of RDTSCP, and executions of RDMSR that read from the IA32_TIME_STAMP_COUNTER MSR return a value modified by the TSC multiplier field (see Section 24.6.5 and Section 25.3). |

Source: http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html

# Attack Surface – Página não sanitizada

- https://xenbits.xen.org/xsa/advisory-100.html

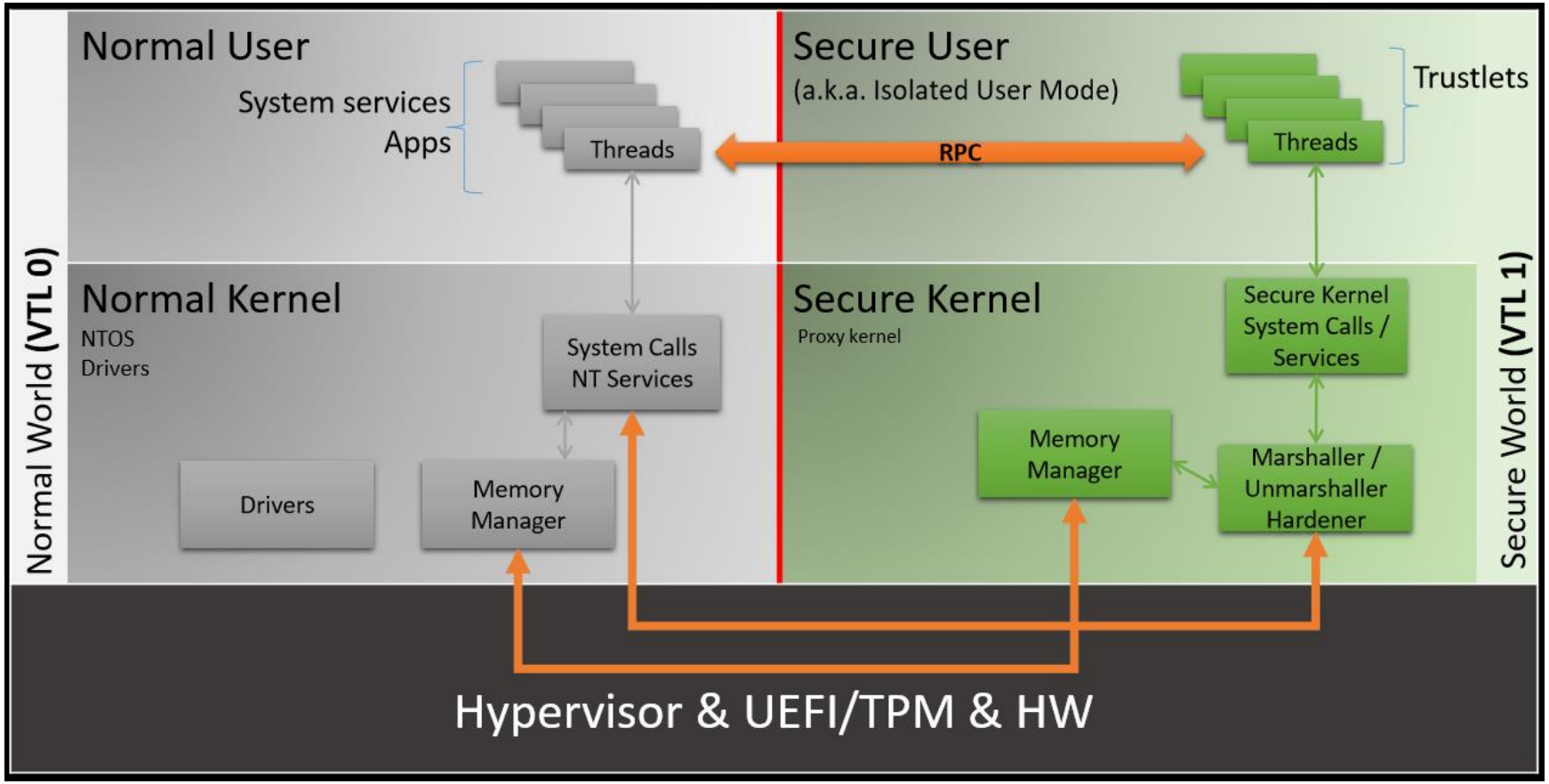
```
--- a/xen/common/page_alloc.c
+++ b/xen/common/page_alloc.c
@@ -1409,7 +1409,10 @@ void free_xenheap_pages(void *v, unsigne
     pg = virt_to_page(v);

     for ( i = 0; i < (1u << order); i++ )
+    {
+        scrub_one_page(&pg[i]);
         pg[i].count_info &= ~PGC_xen_heap;
+    }

     free_heap_pages(pg, order);
 }
@@ -1579,6 +1582,8 @@ void free_domheap_pages(struct page_info
     else
     {
         /* Freeing anonymous domain-heap pages. */
+        for ( i = 0; i < (1 << order); i++ )
+            scrub_one_page(&pg[i]);
         free_heap_pages(pg, order);
         drop_dom_ref = 0;
     }
```

# SMM

- SMM
  - SMM is a special-purpose operating mode provided for handling system-wide functions like power management, system hardware control, or proprietary OEM-designed code. It is intended for use only by **system firmware**, not by applications software or general-purpose systems software. The main benefit of SMM is that it offers a distinct and easily isolated processor environment that **operates transparently to the operating system or executive and software applications**.

- Attack surface?

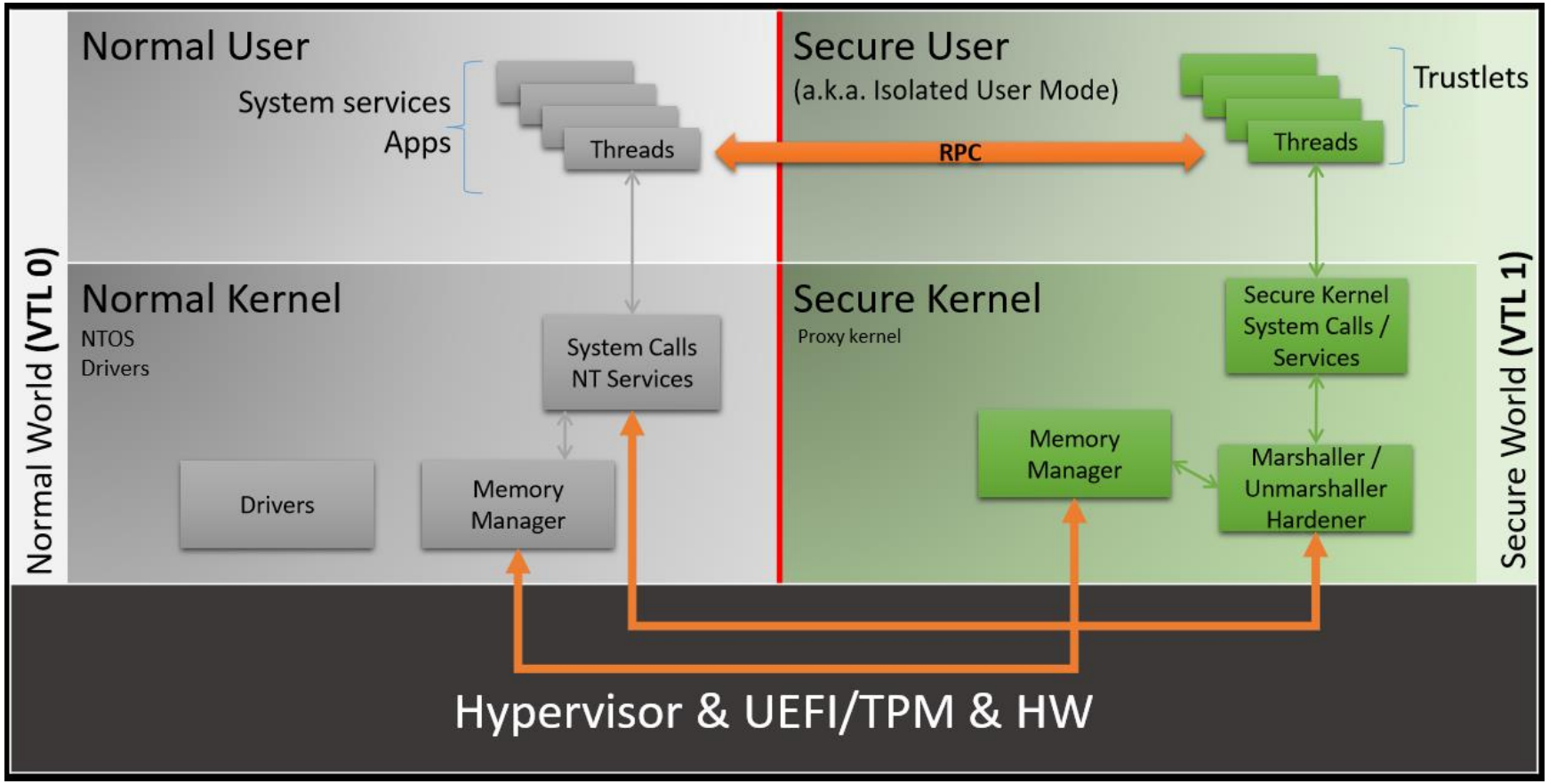
# Microsoft Virtualization Based Security (VBS) – Visão Geral




Source: Black Hat 2015 – "Defeating Pass-the-Hash – Separation of Powers" - https://www.blackhat.com/docs/us-15/materials/us-15-Moore-Defeating%20Pass-the-Hash-Separation-Of-Powers.pdf

# Microsoft Virtualization Based Security (VBS) – Visão Geral

**Host to Hypervisor???**




Source: Black Hat 2015 – "Defeating Pass-the-Hash – Separation of Powers" - https://www.blackhat.com/docs/us-15/materials/us-15-Moore-Defeating%20Pass-the-Hash-Separation-Of-Powers.pdf

# Agenda

- Discussão de conceitos de virtualização
- Aspectos de segurança
- Conclusão

# Conclusão

- A aplicação da virtualização mostra-se muito útil para diversos casos de uso
  - Segurança
  - Cloud
  - Etc

- No entanto, há diversos detalhes que precisam ser cautelosamente implementados para um melhor nível de segurança

# Obrigado!

Gabriel Negreira Barbosa
gabriel.negreira.barbosa [arroba] intel.com