Architectures de sécurité

concepts logiciels et matériels pour la sécurité des systèmes

Stéphane Duverger

Airbus Group Innovations
Toulouse, FRANCE

TLS-SEC 2017/2018

But de ce cours

Objectif

- aborder la sécurité au niveau de l'architecture
- à la fois matérielle (périphériques, extensions du cpu)
- mais également logicielle (conception, design)
- pas d'isolation applicative dans ce cours (ie. sandbox)

Isolation et Cloisonnement

Qu'es aquò ?

Cloisonnement : à quel niveau ?

- application elle-même :
  - plugins, applets java
  - sandbox navigateur pour javascript
  - chrome native client
- entre applications :
  - chroot
  - bsd jail
  - seccomp-bpf
- de l'OS (partie utilisateur) : un noyau pour plusieurs instances userland
- de l'OS (partie noyau) : plusieurs OS complets (Virtual Machines)
- ségrégation matérielle complète :
  - communication distante
  - protocole d'interfaçage restreint

But de ce cours

Plan d'action

- rappels sur la sécurité au niveau du CPU (ring, syscalls, ...)
- éléments d'architecture matérielle (North/South bridges, DMA)
- simulation, émulation (QEmu, Bochs, DosBox)
- virtualisation/conteneurisation
  - mutualisation noyau (openVZ, LXC, Vserver, BSD Jails, Zones Solaris)
  - Para et Full virtualization (Xen, VMware, VirtualBox)
- architectures de noyaux d'OS
- isolation des périphériques (ACPI, attaques DMA, I/O MMU)
- boot sécurisé (TPM, Intel TXT)
- extensions de sécurité pour x86

Rappels de sécurité au niveau CPU

Résumé

- notion de niveau de privilèges (rings)
- transition entre rings (Gates)
- concept d'appel système (interruption ou instruction dédiée)
- pagination (user/supervisor)

Rappels de sécurité au niveau CPU

Résumé

- notion de niveau de privilèges (rings)
- transition entre rings (Gates)
- concept d'appel système (interruption ou instruction dédiée)
- pagination (user/supervisor)

Vision du système complet

- insuffisant pour securiser un système complet
- quid des périphériques ?
- des firmwares incontrolés de ces périphériques ?
- ... nombreuses solutions envisagées

Exemple de périphérique :

contrôleur IDE-(S)ATA

Fonctionnement d'un contrôleur de Disque

Globalement

- Disques attachés à un contrôleur
- différents types : IDE, SCSI, SATA
- parlons de la norme ATA

Fonctionnement d'un contrôleur de Disque

Globalement

- Disques attachés à un contrôleur
- différents types : IDE, SCSI, SATA
- parlons de la norme ATA

Commander un périphérique sous x86

- `in`/`out`, ou *memory mapped*
- différents ports/registres (`0x1F0`)
- configuration :
  - nombre de secteurs (SCNT)
  - depuis quel secteur (LBA, accès logique au CHS)
  - lecture/écriture
  - quel disque ?

Fonctionnement d'un contrôleur de Disque

Globalement

- Disques attachés à un contrôleur
- différents types : IDE, SCSI, SATA
- parlons de la norme ATA

Commander un périphérique sous x86

- `in`/`out`, ou *memory mapped*
- différents ports/registres (`0x1F0`)
- configuration :
  - nombre de secteurs (SCNT)
  - depuis quel secteur (LBA, accès logique au CHS)
  - lecture/écriture
  - quel disque ?
- opération d'accès au(x) secteur(s)

Fonctionnement d'un contrôleur de Disque

Accès historique

- string I/O (`rep ins[b,w,d]/outs[b,w,d]`) sur chaque secteur
- très lent, monopolise le CPU

Fonctionnement d'un contrôleur de Disque

Accès historique

- string I/O (`rep ins[b,w,d]/outs[b,w,d]`) sur chaque secteur
- très lent, monopolise le CPU

Alternative rapide : DMA (Direct Memory Access)

- décharger le CPU pendant la copie des secteurs du disque
- utilise des MMI/O
- plus ou moins spécifique à chaque périphérique
- configuration de PRDT (Physical Region Descriptor Tables)
- tables de pointeurs vers des buffers de l'OS
- l'OS prépare ses PRDTs et les donne au pépriphérique
- le périphérique effectue les accès et previent l'OS une fois terminés

Fonctionnement d'un contrôleur de Disque

Accès historique

- string I/O (`rep ins[b,w,d]/outs[b,w,d]`) sur chaque secteur
- très lent, monopolise le CPU

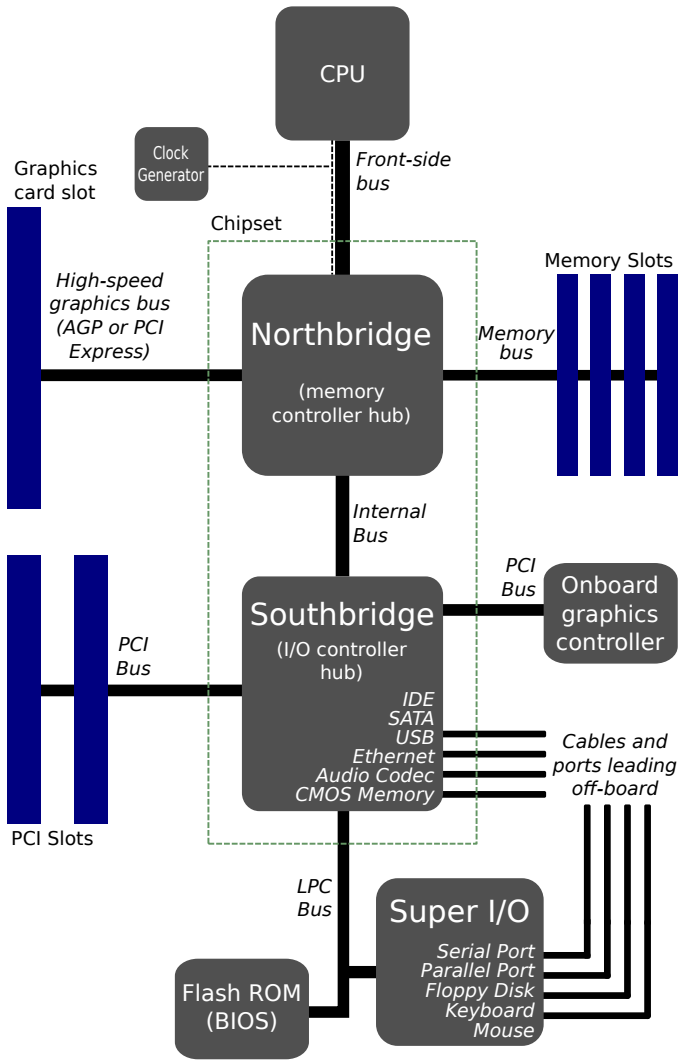
Alternative rapide : DMA (Direct Memory Access)

- décharger le CPU pendant la copie des secteurs du disque
- utilise des MMI/O
- plus ou moins spécifique à chaque périphérique
- configuration de PRDT (Physical Region Descriptor Tables)
- tables de pointeurs vers des buffers de l'OS
- l'OS prépare ses PRDTs et les donne au pépriphérique
- le périphérique effectue les accès et previent l'OS une fois terminés

... vous voyez le problème arriver ?

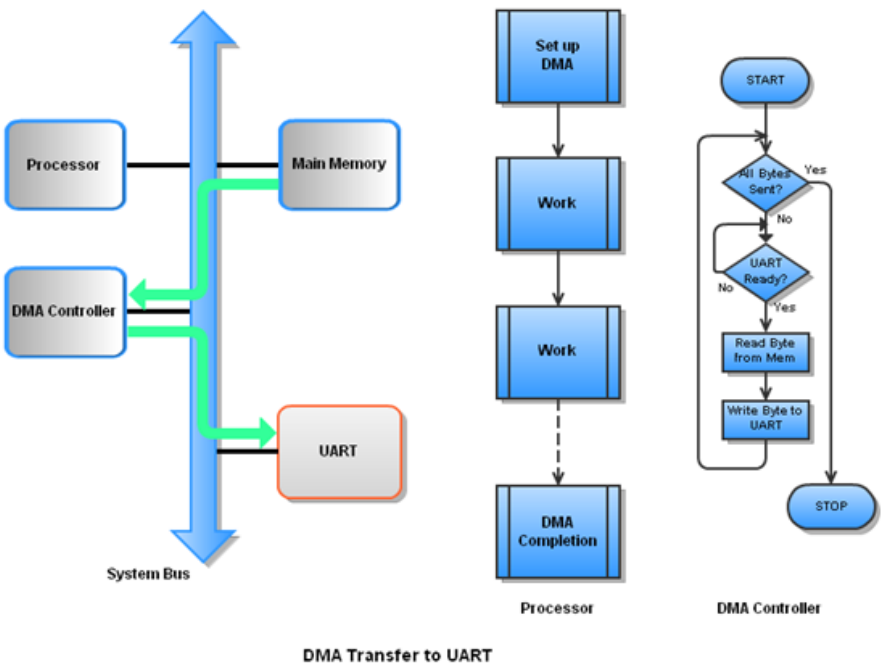
Fonctionnement d'un contrôleur de Disque

- le contrôleur accède tout seul à la mémoire physique
- requêtes du south-bridge vers le north-bridge
- en dehors du contrôle du CPU

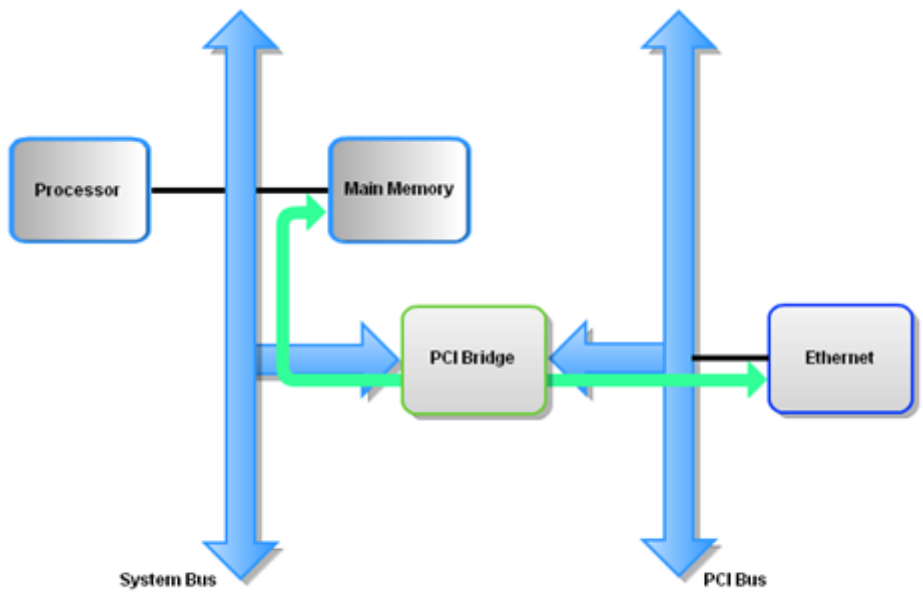


Direct Memory Access

- certains périphériques accèdent directement à la mémoire
- on parle de DMA (Direct Memory Access) :
  - historiquement *Third Party DMA* :
    - un controleur 8237 central
    - bus ISA
  - bus mastering *Firt Party DMA* :
    - chaque périphérique possède un *DMA engine*
    - PCI DMA capable



DMA Transfer to UART



DMA Transfer Across PCI Bus

Préambule

Pourquoi virtualiser

- différences entre émulation et virtualisation
- compromis sécurité/performance
- isolation entre VMs

Préambule

Pourquoi virtualiser

- différences entre émulation et virtualisation
- compromis sécurité/performance
- isolation entre VMs

Petit Glossaire

- VMM (Virtual Machine Monitor), Hyperviseur
- VM (Virtual Machine)

Émulation

Émuler des architectures

- Java et autres langages à ByteCode
- x86 : Bochs, SimNow
- consoles : MAME, SNes9x, PCSX, NeoRageX
- multiples : QEmu (ARM, PPC, Mips, x86 ...)

Émulation

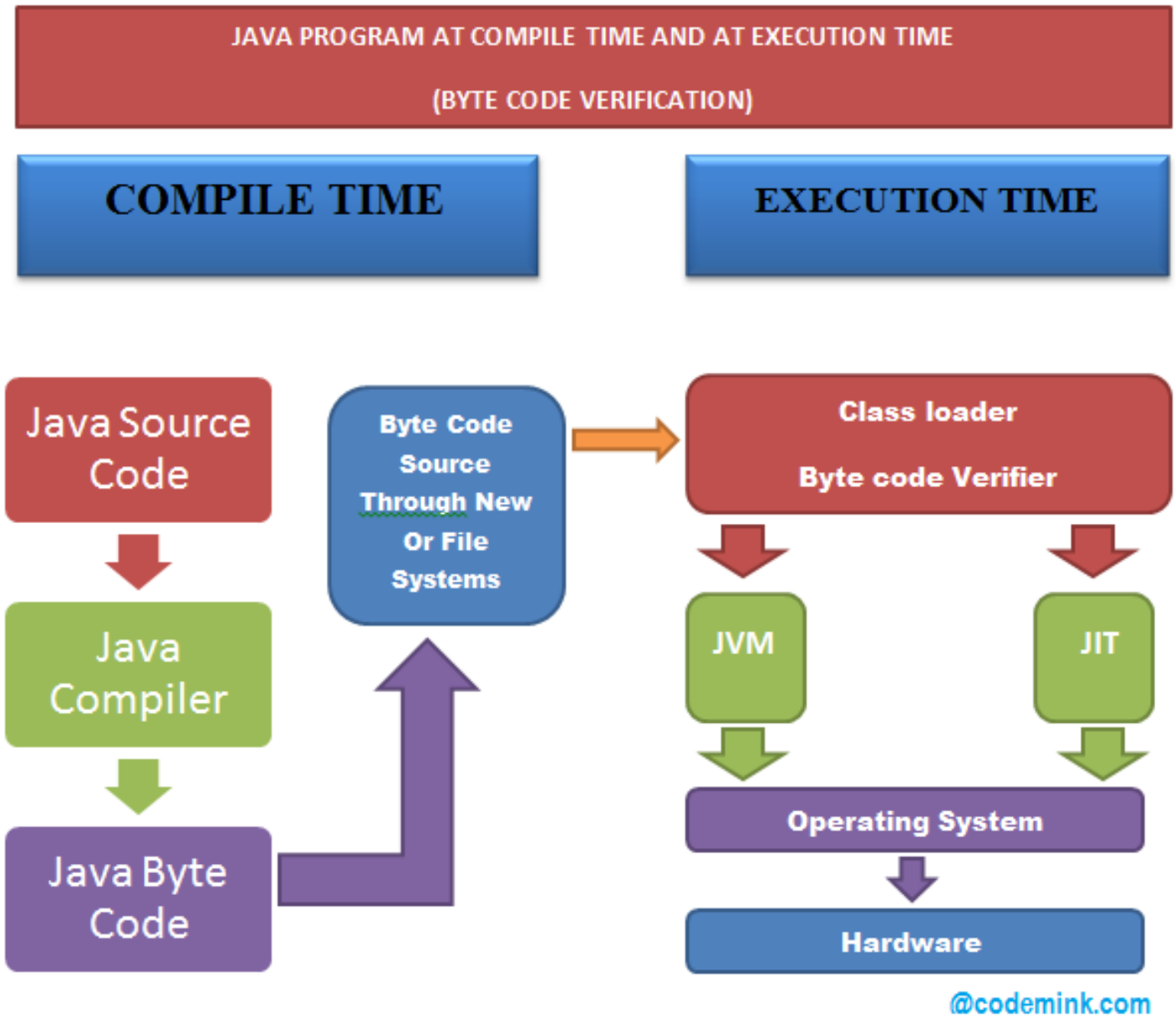
Émuler des architectures

- Java et autres langages à ByteCode
- x86 : Bochs, SimNow
- consoles : MAME, SNes9x, PCSX, NeoRageX
- multiples : QEmu (ARM, PPC, Mips, x86 ...)

Techniques d'émulation

- interprétation des instructions du CPU
- émulation d'un chipset et de périphériques (board)
- lent pour des instructions fréquentes et non-sensibles
- solution : traduction dynamique ou anticipée
  - JIT : Just-In-Time Compiler
  - AOT : Ahead-Of-Time Compiler

Just-In-Time Compiler



Émulation

Du point de vue sécurité

- en théorie plus sûr
- pas d'instructions executées sans contrôle
- instructions sensibles simulées
- périphériques simulés

Émulation

Du point de vue sécurité

- en théorie plus sûr
- pas d'instructions executées sans contrôle
- instructions sensibles simulées
- périphériques simulés

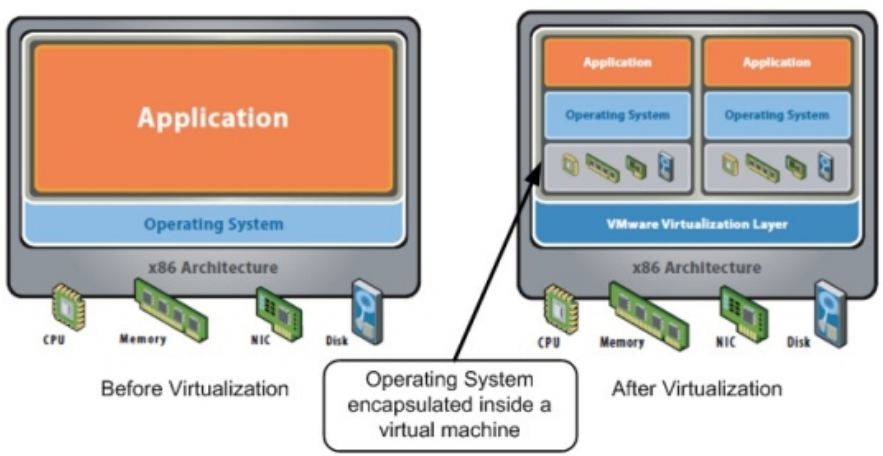
Intérêt

- quid d'x86 à x86 ? lenteur inutile, pas besoin de JIT
- . . . ah si pour les instructions sensibles !
- naissance de la virtualisation

Virtualisation

Intérets multiples

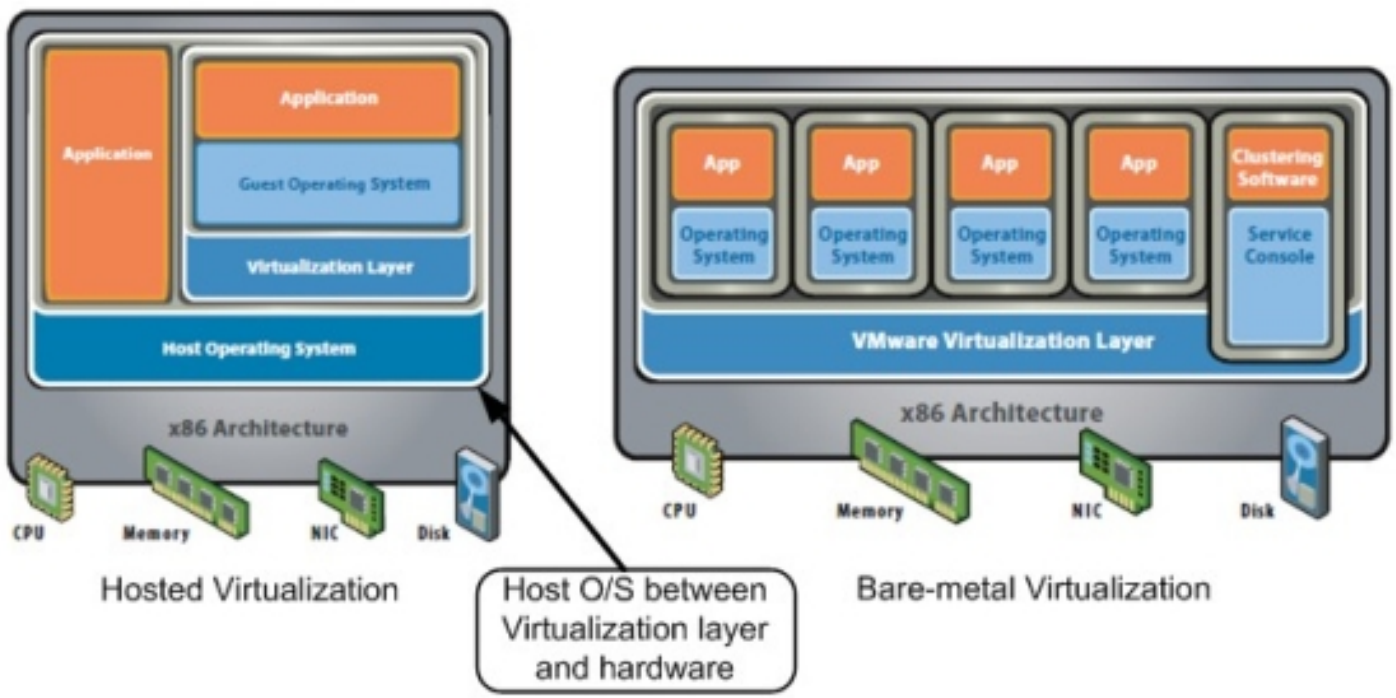
- permet l'exécution d'un (plusieurs) système(s) complet(s) sur une machine physique
- à architecture équivalente (ex : x86 vers x86)
  - performances quasiment natives
  - pas d'emulation d'instructions (autant que possible)
- mutualiser/distribuer des ressources physiques (Cloud)
- migrer des systèmes virtuels vers différentes machines physiques
- déployer/standardiser des environnements (uniformité matériel virtuel)
- idéalement ... cloisonner des composants logiciels



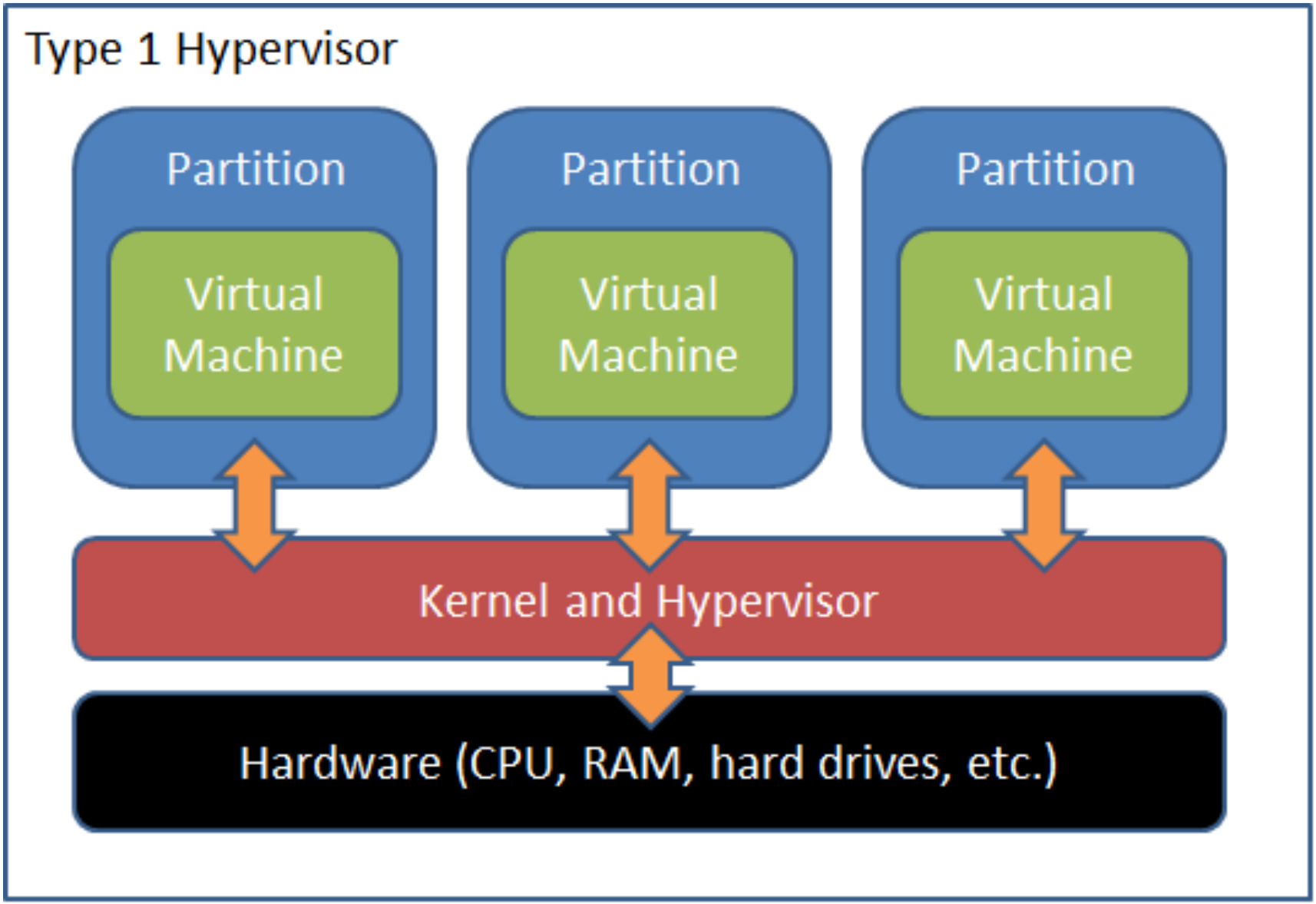
Virtualisation : architectures

Différentes architectures de VMM

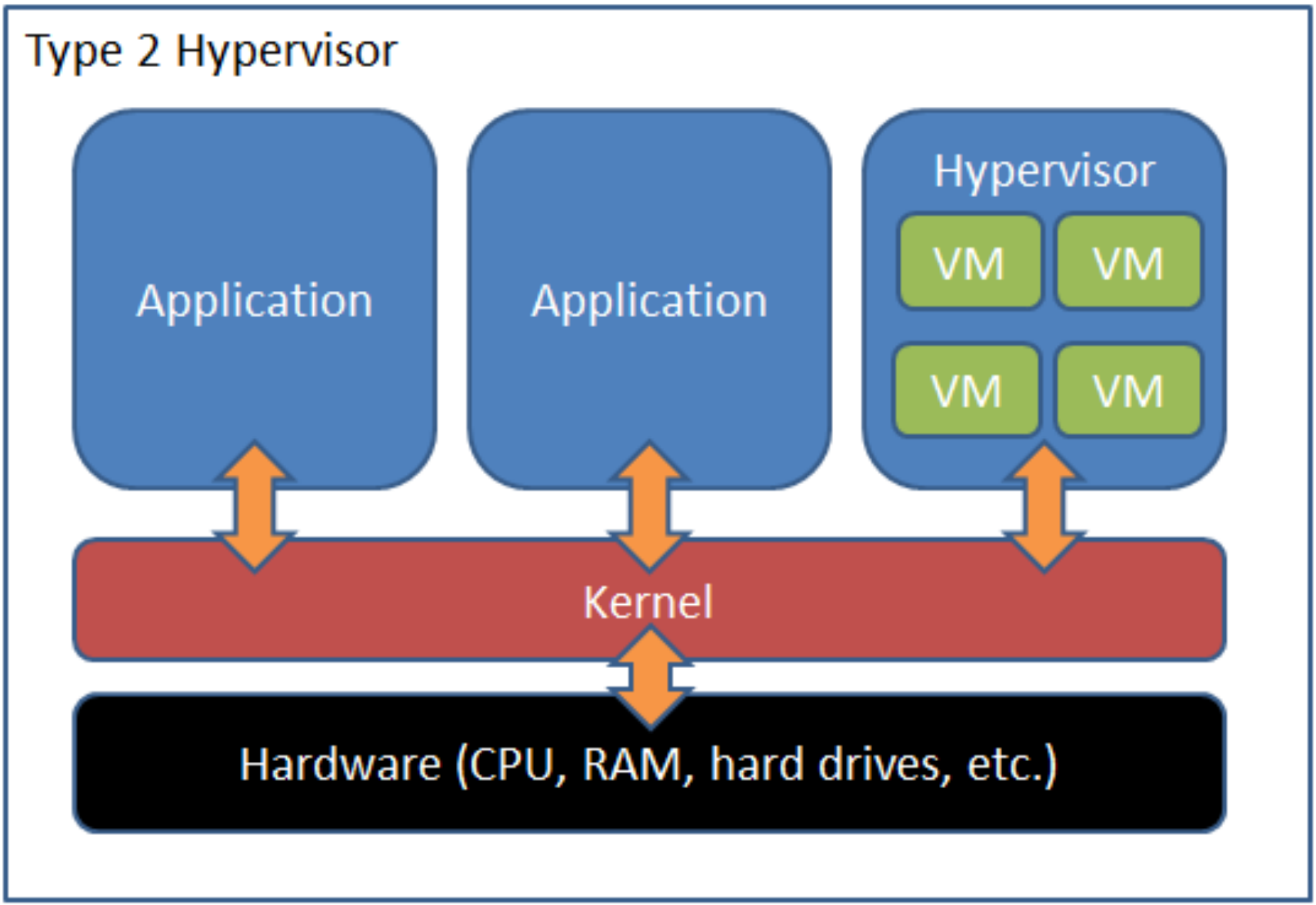
- Type 1, *bare-metal* : VMware ESX, Xen, Hyper-V, Ramooflax
- Type 2, *hosted* : VMware Workstation/Fusion, KVM, Virtualbox
- On-the-Fly virtualisation : BluePill, Vitriol, ... rootkits



Architectures de VMM : type 1



Architectures de VMM : type 2



Virtualisation : comment ça marche ?

Différentes natures de virtualisation

- Binary translation
- Full ou Hardware Assisted virtualisation (HVM)
- Para virtualisation (PV)
- Para-Hardware-Assisted Virtualisation (PVHVM)

`http://www.vmware.com/techpapers/2007/understanding-full-virtualization-paravirtualizat-1008.html`

Virtualisation : comment ça marche ?

Binary Translation

- l'OS ne sait pas qu'il est virtualisé
- exécution native des instructions non-sensibles
- instructions privilégiées :
  - interception par fautes (*#GP*,*#PF*, ...)
  - émulation
  - injection d'un *hypercall* ou équivalent
  - l'os est modifié dynamiquement
- inventé par VMWare pour x86

Virtualisation : comment ça marche ?

Full virtualisation

- l'os ne sait pas qu'il est virtualisé
- il n'est pas modifié
- toutes instructions exécutés nativement
- nécessité d'une aide matérielle à la virtualisation
- Intel VT ou AMD V
- VMWare Workstation, VirtualBox, KVM

Virtualisation : comment ça marche ?

Para virtualisation : Xen

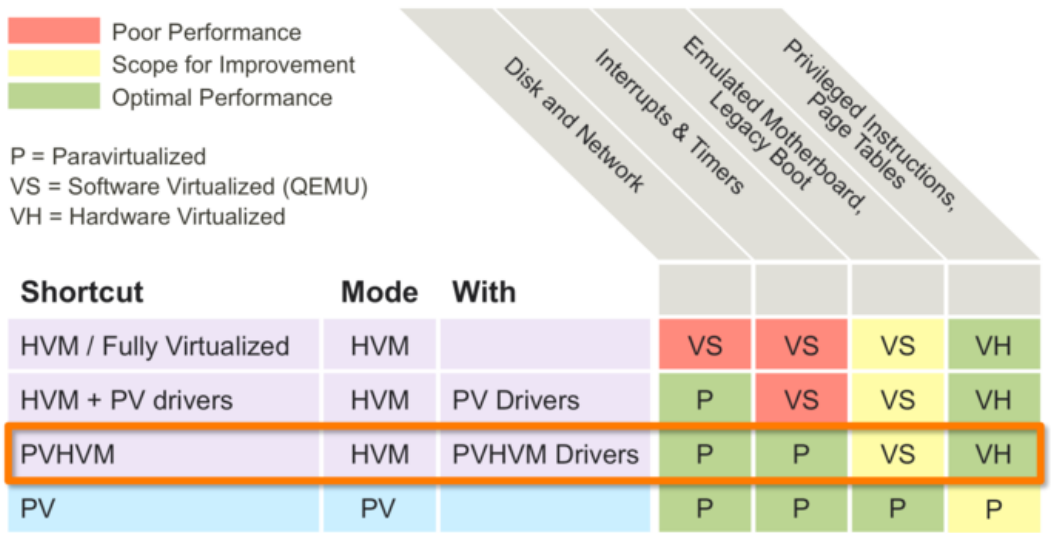
- l'OS sait qu'il est virtualisé
- préparé spécifiquement pour l'hyperviseur
- instructions privilégiées effectuent des *hypercall*
- Linux : `paravirt_ops`

```
struct pv_cpu_ops pv_cpu_ops = {
        .cpuid = native_cpuid,
...
        .read_cr0 = native_read_cr0,
        .write_cr0 = native_write_cr0,
        .read_cr4 = native_read_cr4,
        .read_cr4_safe = native_read_cr4_safe,
        .write_cr4 = native_write_cr4,
...
        .read_msr = native_read_msr,
        .write_msr = native_write_msr,
        .load_tr_desc = native_load_tr_desc,
        .set_ldt = native_set_ldt,
        .load_gdt = native_load_gdt,
        .load_idt = native_load_idt,
...
        .write_gdt_entry = native_write_gdt_entry,
        .write_idt_entry = native_write_idt_entry,
...
        .alloc_ldt = paravirt_nop,
        .free_ldt = paravirt_nop,
...
        .load_sp0 = native_load_sp0,
        .iret = native_iret,
...
};
```

Virtualisation : comment ça marche ?

Para-Hardware-Assisted virtualisation

- Xen propose ce mode hybride
- utilise la Full virtualisation pour le CPU/MMU
- utilise la Para virtualisation pours périphériques
- frontend/backend très optimisés pour le réseau par exemple



Virtualisation : comment ça marche ?

Et les périphériques ?

- émulation (ex : via Qemu pour KVM, Xen, VirtualBox)
- *pass-through* : VMM dédie un périphérique physique à une VM
  - souvenez-vous des accès DMA ? :)
- SR-IOV : Single Root IO Virtualisation, périphérique supporte la virtualisation matérielle

Popek & Goldberg requirements (1974)

Définition de VMM

- Equivalence/Fidelity : pas de divergences comportementales entre une exécution virtualisée et non-virtualisée (machine réelle)
- Resource control/Safety : contrôle des ressources par le VMM
- Efficiency/Performance : minimum d'intervention du VMM

Popek & Goldberg requirements (1974)

Définition de VMM

- Equivalence/Fidelity : pas de divergences comportementales entre une exécution virtualisée et non-virtualisée (machine réelle)
- Resource control/Safety : contrôle des ressources par le VMM
- Efficiency/Performance : minimum d'intervention du VMM

Classification des instructions

- Privileged : trap if user mode
- Control sensitive : change configuration of resources
- Behavior sensitive : behavior depends on configuration of resources

Problème de l'architecture x86

Nativement pas *virtualisable*

- ne respecte pas Popek & Goldberg
- instructions privilégiées (`mov cr`, `lidt`) et sensibles (`sidt`, `popf`, `cpuid`)
- extension de l'ISA : Intel VT (VMX) et AMD-V (SVM)
- instructions spécifiques d'aide à la virtualisation

Éléments communs entre Intel-VT (vmx) et AMD-V (svm)

Intérêt

- simplifier le développement d'hyperviseurs
- jeu d'instructions réduit ($\sim$ 10)
- notion de *vm-entry/vm-exit*
  - *vm-entry* charge la VM, sauve le VMM
  - *vm-exit* charge le VMM et sauve la VM

Configuration d'une structure de données

- AMD *VMCB*, Intel *VMCS* (asynchrone accès via `vmread, vmwrite`)
- préparation des registres systèmes (`cr, dr, gdtr, idtr, ...`)
- injection d'évènements (interruptions, exceptions)
- mise en place de bitmaps d'interceptions
  - évènements
  - instructions sensibles
  - accès aux I/O, MSRs ...

De nombreuses limitations

- Intel/AMD non compatibles
- fonctionnalités différentes selon modèle de CPU
- difficile de savoir ce que le CPU propose avant de l'acheter !

  `http ://cpuid.intel.com ?`

De nombreuses limitations

- Intel/AMD non compatibles
- fonctionnalités différentes selon modèle de CPU
- difficile de savoir ce que le CPU propose avant de l'acheter !

  `http ://cpuid.intel.com ?`

- informations insuffisantes lors des vm-exit
- nécessaire d'embarquer un désassembleur/émulateur
- interception des interruptions matérielles et logicielles de type on/off
- pas d'interception des interruptions logicielles sous Intel
- interruptions matérielles *pending* sous AMD
- cas des SMIs (bugs CPU, bugs BIOS, virtualisation SMM nécessaire, ...)

Gestion du mode réel catastrophique sous Intel
problématique pour le BIOS !

Virtualisation du BIOS

BIOS et mode réel

- mode par défaut du CPU
- 16 bits, adressage mémoire 20 bits (1Mo), pas de protection
- utilisé intensivement par le BIOS

Virtualiser le mode réel *historiquement*

- virtualisation matérielle existe depuis 80386 : mode v8086
- cpu émule mécanismes du mode réel (interruptions, far call, ...)
- redirection/interception des I/O, interruptions

Virtualisation du BIOS

BIOS et mode réel

- mode par défaut du CPU
- 16 bits, adressage mémoire 20 bits (1Mo), pas de protection
- utilisé intensivement par le BIOS

Virtualiser le mode réel *historiquement*

- virtualisation matérielle existe depuis 80386 : mode v8086
- cpu émule mécanismes du mode réel (interruptions, far call, ...)
- redirection/interception des I/O, interruptions

Virtualiser le mode réel avec `vmx/svm`

- AMD propose un *paged real mode* (`CR0.PE=0 && CR0.PG=1`)

Virtualisation du BIOS

BIOS et mode réel

- mode par défaut du CPU
- 16 bits, adressage mémoire 20 bits (1Mo), pas de protection
- utilisé intensivement par le BIOS

Virtualiser le mode réel *historiquement*

- virtualisation matérielle existe depuis 80386 : mode v8086
- cpu émule mécanismes du mode réel (interruptions, far call, ...)
- redirection/interception des I/O, interruptions

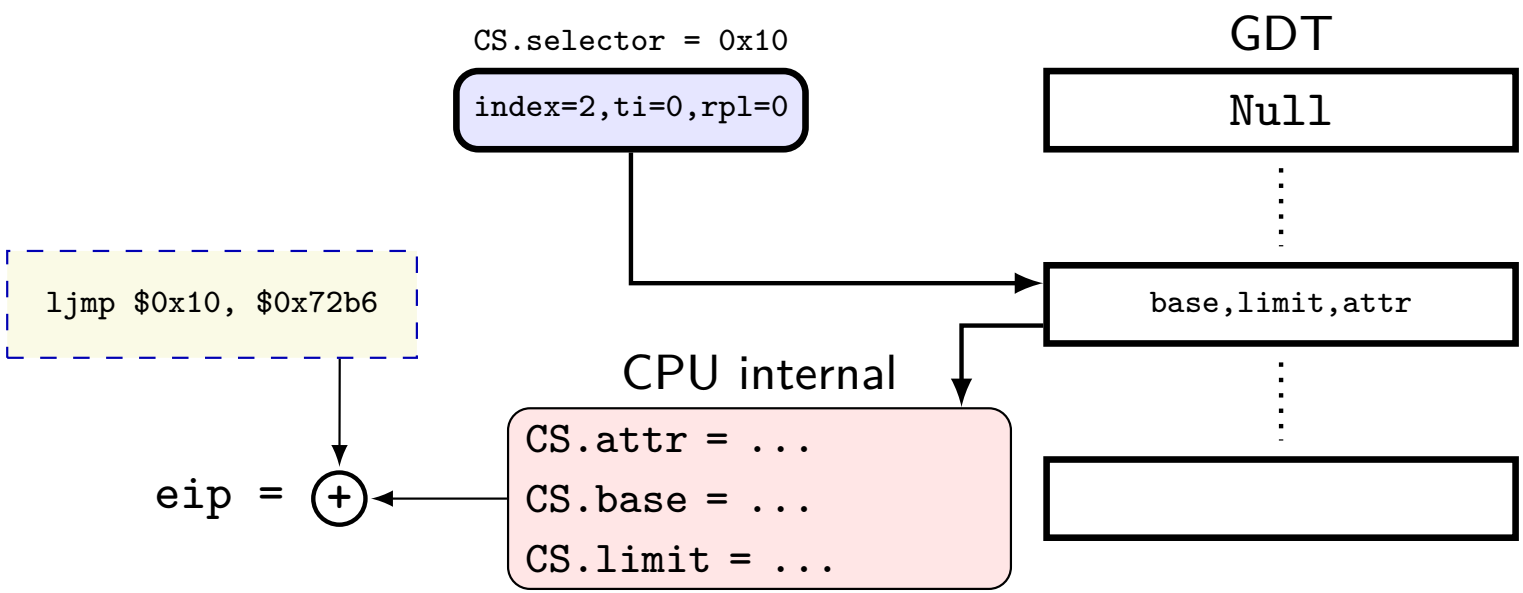
Virtualiser le mode réel avec `vmx/svm`

- AMD propose un *paged real mode* (`CR0.PE=0 && CR0.PG=1`)
- Intel interdit `CR0.PG=0` et donc `CR0.PE=0`
  - préconise l'utilisation du mode v8086
  - vm-entry en v8086 sont très restrictifs
  - surtout concernant la segmentation

Petit rappel sur la segmentation

Gestion des registres de segment

- partie visible accessible au programmeur (sélecteur)
- partie cachée gérée par le cpu (base, limite, attributs)
- mode réel : $base = selector * 16$, $limit = 64K$
- mode protégé : descripteurs de segments



Virtualisation du BIOS

Le mode irréel (*unreal, flat real, big real mode*)

- permet d'accéder à plus d'1Mo en mode réel
- transition mode protégé vers mode réel en laissant base=0 et limite=4Go
- utilisé par les BIOS pour accéder aux *memory mapped devices* . . .

```
seg000 :F7284          mov     bx, 20h
seg000 :F7287          cli
seg000 :F7288          mov     ax, cs
seg000 :F728A          cmp     ax, 0F000h
seg000 :F728D          jnz     short near ptr unk_7297
seg000 :F728F          lgdt    fword ptr cs :byte_8163     (1)
seg000 :F7295          jmp     short near ptr unk_729D
seg000 :F7297          lgdt    fword ptr cs :byte_8169
seg000 :F729D          mov     eax, cr0
seg000 :F72A0          or      al, 1
seg000 :F72A2          mov     cr0, eax                   (2)
seg000 :F72A5          mov     ax, cs
seg000 :F72A7          cmp     ax, 0F000h
seg000 :F72AA          jnz     short near ptr unk_72B1
seg000 :F72AC          jmp     far ptr 10h :72B6h          (3)
seg000 :F72B1          jmp     far ptr 28h :72B6h
seg000 :F72B6          mov     ds, bx                     (4)
seg000 :F72B8          mov     es, bx
seg000 :F72BA          mov     eax, cr0
seg000 :F72BD          and     al, 0FEh
seg000 :F72BF          mov     cr0, eax                   (5)
seg000 :F72C2          mov     ax, cs
seg000 :F72C4          cmp     ax, 10h                    (6)
seg000 :F72C7          jnz     short near ptr unk_72CE
seg000 :F72C9          jmp     far ptr 0F000h :72D3h
seg000 :F72CE          jmp     far ptr 0E000h :72D3h
```

Virtualisation du BIOS

L'échec d'Intel

- vm-entry en mode v8086 vérifie $base = selector * 16$
- impossible de virtualiser mode irréel avec v8086

Virtualisation du BIOS

L'échec d'Intel

- vm-entry en mode v8086 vérifie $base = selector * 16$
- impossible de virtualiser mode irréel avec v8086

Sans les extensions de virtualisation matérielle récentes

- émulation du mode réel en mode protégé
- intercepter accès aux registres de segment : *far call/jump*, *mov/pop seg*, *iret*
- double fail : Intel ne permet pas d'intercepter l'écriture de registres de segments
- solution : forcer limite GDT et IDT à 0 et intercepter les `#GP`

Une technologie qui évolue

Dans les CPUs récents et suffisament haut de gamme

- nouveau mode *Unrestricted Guest* (autorise `CR0.PE=0 && CR0.PG=0`)
- nécessite d'avoir Intel EPT pour protéger la mémoire du VMM
- on entend parler de VT-x2

Une technologie qui évolue

Dans les CPUs récents et suffisament haut de gamme

- nouveau mode *Unrestricted Guest* (autorise `CR0.PE=0 && CR0.PG=0`)
- nécessite d'avoir Intel EPT pour protéger la mémoire du VMM
- on entend parler de VT-x2

Performances

- de nombreuses limitations dans les premières versions
- pas de virtualisation de la MMU (EPT)
- VMWare Binary Translation plus rapide que VT-x sans EPT
- à lire absolument :

A Comparison of Software and Hardware Techniques for x86 Virtualization

Keith Adams & Ole Agesen

`https://www.vmware.com/pdf/asplos235_adams.pdf`

Nested Virtualization

Un hyperviseur dans un hyperviseur

- concept d'OS dans un OS résolue par la virtualisation
- niveau d'*inception* supplémentaire : un VMM dans un VMM
- la question se pose pour la virtualisation matérielle

Nested Virtualization

Un hyperviseur dans un hyperviseur

- concept d'OS dans un OS résolue par la virtualisation
- niveau d'*inception* supplémentaire : un VMM dans un VMM
- la question se pose pour la virtualisation matérielle

Bingo !

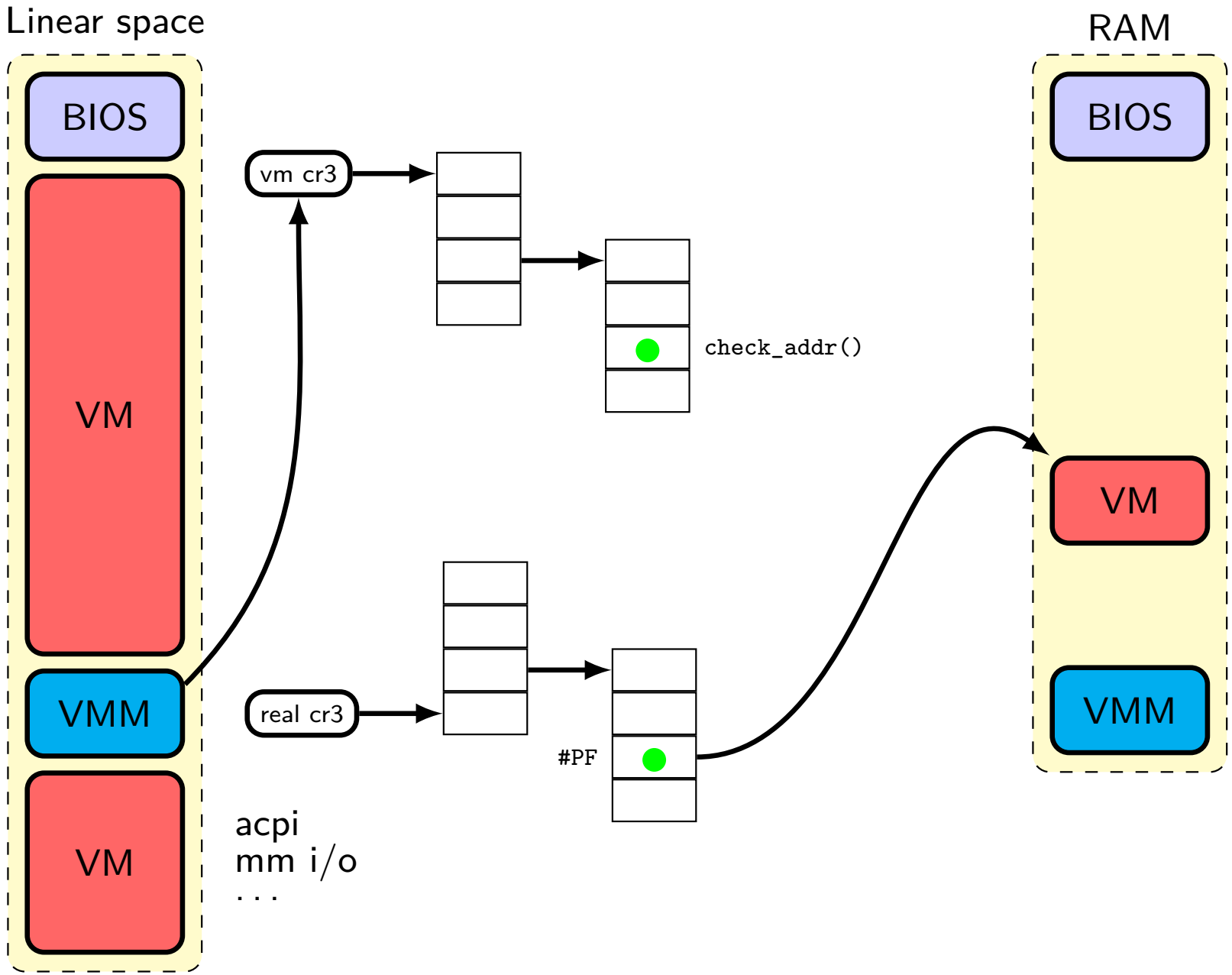
- pas possible nativement avec VT-X
- supporté par VMware en mode émulation si EPT présent
- apparition récente (2014) des *shadow* VMCS
- virtualisation des instructions de virtualisation

Méthodes de virtualisation de la mémoire

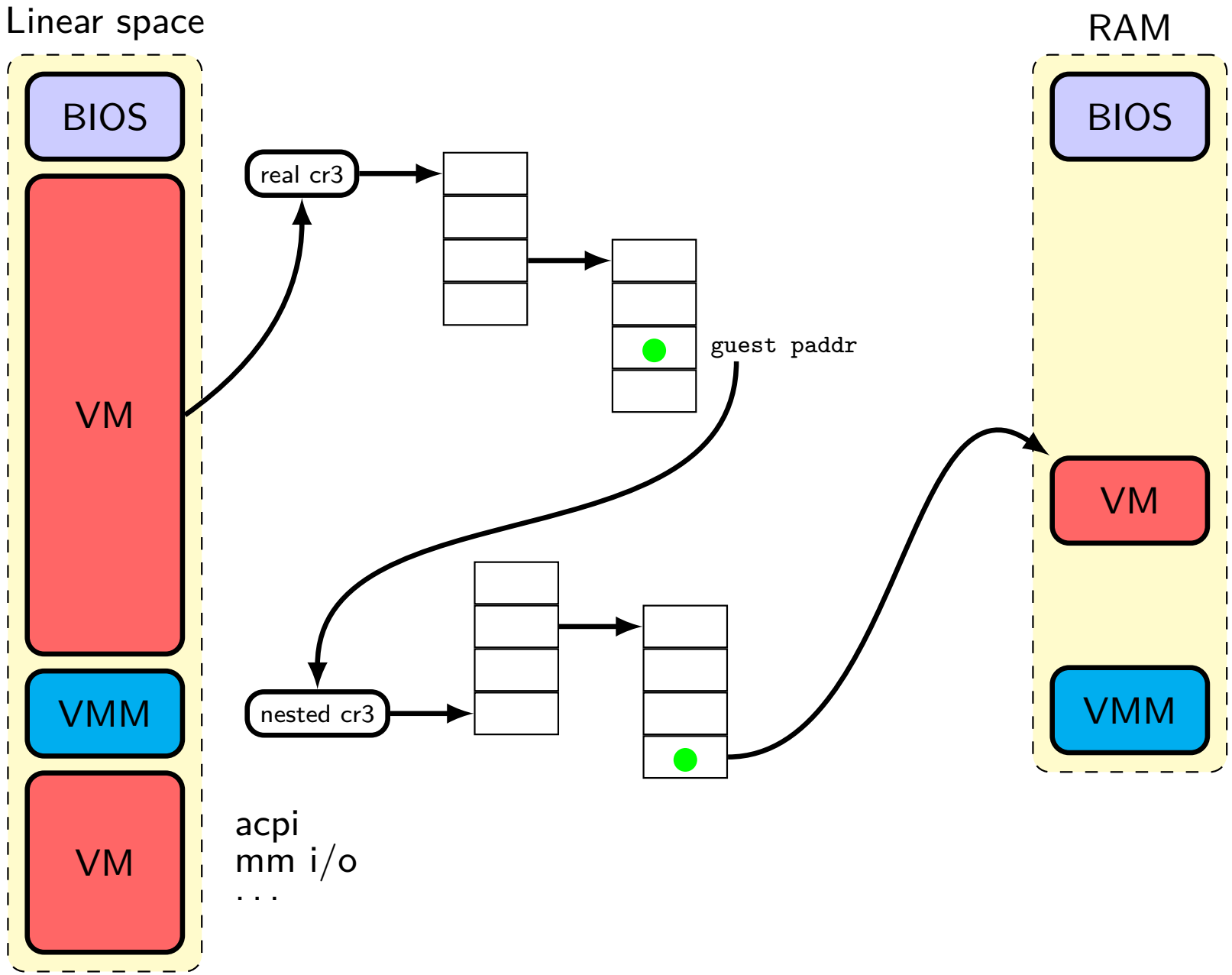
comment virtualiser la MMU ?

- logicielle : Shadow Page Tables
- matérielle : Nested Page Tables
  - Intel EPT (Extended Page Tables)
  - AMD NPT/RVI (Nested Page Tables, Rapid Virtualization Indexing)

Shadow Page Tables



Nested Page Tables



Outils de sécurité fondés sur la virtualisation matérielle

Virtualiser pour instrumenter

- concept de virtualisation légère : micro-virtualisation
- VMM dédié à une unique VM
- généralement en passthrough vis-à-vis des périphériques
- instrumentation d'un OS plutôt que mutualisation de ressources

Quelques exemples

- debuger discret (Ramooflax, VirtDBG)
- protection de programmes (AppVSWild : Ramooflax based)
- analyse de programmes (MAVMM, Ramooflax)
- sandbox (Capsule, Ramooflax)
- rootkit (Blue Pill, Ramooflax)

Xen Security Advisories (XSA) : 190 vulnérabilités Octobre 2016

Advisory	Release	CVE	Title
XSA-151	2015-10-29 11 :59	CVE-2015-7969	x86 : leak of per-domain profiling-related vcpu pointer array
XSA-148	2015-10-29 11 :59	CVE-2015-7835	x86 : Uncontrolled creation of large page mappings by PV guests
XSA-147	2015-10-29 11 :59	CVE-2015-7814	arm : Race between domain destruction and memory allocation decrease
XSA-142	2015-09-22 10 :00	CVE-2015-7311	libxl fails to honour readonly flag on disks with qemu-xen
XSA-140	2015-08-03 12 :00	CVE-2015-5165	QEMU leak of uninitialized heap memory in rtl8139 device model
XSA-139	2015-08-03 12 :00	CVE-2015-5166	Use after free in QEMU/Xen block unplug protocol
XSA-138	2015-07-27 12 :00	CVE-2015-5154	QEMU heap overflow flaw while processing certain ATAPI commands.
XSA-136	2015-06-11 12 :00	CVE-2015-4164	vulnerability in the iret hypercall handler
XSA-135	2015-06-10 13 :10	CVE-2015-3209	Heap overflow in QEMU PCNET controller, allowing guest->host escape
XSA-134	2015-06-11 12 :00	CVE-2015-4163	GNTTABOP_swap_grant_ref operation misbehavior
XSA-133	2015-05-13 11 :15	CVE-2015-3456	Privilege escalation via emulated floppy disk drive
XSA-131	2015-06-02 12 :00	CVE-2015-4106	Unmediated PCI register access in qemu
XSA-130	2015-06-02 12 :00	CVE-2015-4105	Guest triggerable qemu MSI-X pass-through error messages
XSA-126	2015-03-31 12 :00	CVE-2015-2756	Unmediated PCI command register access in qemu
XSA-123	2015-03-10 12 :00	CVE-2015-2151	Hypervisor memory corruption due to x86 emulator flaw
XSA-121	2015-03-05 12 :00	CVE-2015-2044	Information leak via internal x86 system device emulation
XSA-119	2015-03-12 12 :00	CVE-2015-2152	HVM qemu unexpectedly enabling emulated VGA graphics backends
XSA-112	2014-11-27 11 :25	CVE-2014-8867	Insufficient bounding of "REP MOVS" to MMIO emulated inside the hypervisor
XSA-110	2014-11-18 12 :00	CVE-2014-8595	Missing privilege level checks in x86 emulation of far branches
XSA-109	2014-11-18 12 :00	CVE-2014-8594	Insufficient restrictions on certain MMU update hypercalls
XSA-108	2014-10-01 12 :00	CVE-2014-7188	Improper MSR range used for x2APIC emulation
XSA-106	2014-09-23 12 :00	CVE-2014-7156	Missing privilege level checks in x86 emulation of software interrupts
XSA-105	2014-09-23 12 :00	CVE-2014-7155	Missing privilege level checks in x86 HLT, LGDT, LIDT, and LMSW emulation
XSA-90	2014-03-24 13 :00	CVE-2014-2580	Linux netback crash trying to disable due to malformed packet
XSA-85	2014-02-06 12 :00	CVE-2014-1895	Off-by-one error in FLASK_AVC_CACHESTAT hypercall
XSA-78	2013-11-20 17 :08	CVE-2013-6375	Insufficient TLB flushing in VT-d (iommu) code
XSA-76	2013-11-26 12 :00	CVE-2013-4554	Hypercalls exposed to privilege rings 1 and 2 of HVM guests
XSA-75	2013-11-08 16 :20	CVE-2013-4551	Host crash due to guest VMX instruction execution
XSA-59	2013-08-20 12 :00	CVE-2013-3495	Intel VT-d Interrupt Remapping engines can be evaded by native NMI interrupts
XSA-58	2013-06-26 12 :00	CVE-2013-1432	Page reference counting error due to XSA-45/CVE-2013-1918 fixes —> LOL ;)
XSA-44	2013-04-18 12 :00	CVE-2013-1917	Xen PV DoS vulnerability with SYSENTER
XSA-42	2013-02-12 12 :00	CVE-2013-0228	Linux kernel hits general protection if %ds is corrupt for 32-bit PVOPS.
XSA-41	2013-01-16 14 :50	CVE-2012-6075	qemu (e1000 device driver) : Buffer overflow when processing large packets

Surface d'attaque d'un hyperviseur

- moteur de désassemblage
  - instructions x86 complexes
  - taille variable
  - préfixes : segment, address size, operand size,
  - comportement différents selon mode CPU

Surface d'attaque d'un hyperviseur

- moteur de désassemblage
  - instructions x86 complexes
  - taille variable
  - préfixes : segment, address size, operand size,
  - comportement différents selon mode CPU
- émulateur :
  - instructions CPU ring0 : `mov cr0`
  - *boundary conditions* : exceptions dans tel ou tel cas
  - vérification du niveau de privilège
  - optimisation en tout genre (vitesse == ennemie de la sécurité)
  - instructions liées aux I/O : `rep insw`
  - périphériques : vga, ide, sata, réseau, cdrom, floppy

Surface d'attaque d'un hyperviseur

- moteur de désassemblage
  - instructions x86 complexes
  - taille variable
  - préfixes : segment, address size, operand size,
  - comportement différents selon mode CPU
- émulateur :
  - instructions CPU ring0 : `mov cr0`
  - *boundary conditions* : exceptions dans tel ou tel cas
  - vérification du niveau de privilège
  - optimisation en tout genre (vitesse == ennemie de la sécurité)
  - instructions liées aux I/O : `rep insw`
  - périphériques : vga, ide, sata, réseau, cdrom, floppy
- mémoire virtuelle/physique :
  - shadow page tables
  - DMA (I/O mmu)

Serveurs Privés Virtuels (VPS)

Operating System Level Virtualisation

- virtualisation très légère
- spécifique à un OS (Linux)
- un noyau pour plusieurs serveurs virtuels userland
- on ne duplique que l'applicatif userland (*la distribution*)

Quelques exemples

- Linux V-Server
- LXC
- Docker

Serveurs Privés Virtuels (VPS) : V-server

Linux V-Server

- patch kernel + utilitaires userland
- l'implémentation est basée sur :
  - Linux Capabilities (`CAP_SYS_MODULE, CAP_SYS_PTRACE`)
  - quotas (Resource Limits) : core, process numbers, cpu time
  - chroot
  - security contexts
  - routage
  - ... concepts liés au kernel

Serveurs Privés Virtuels (VPS) : LXC

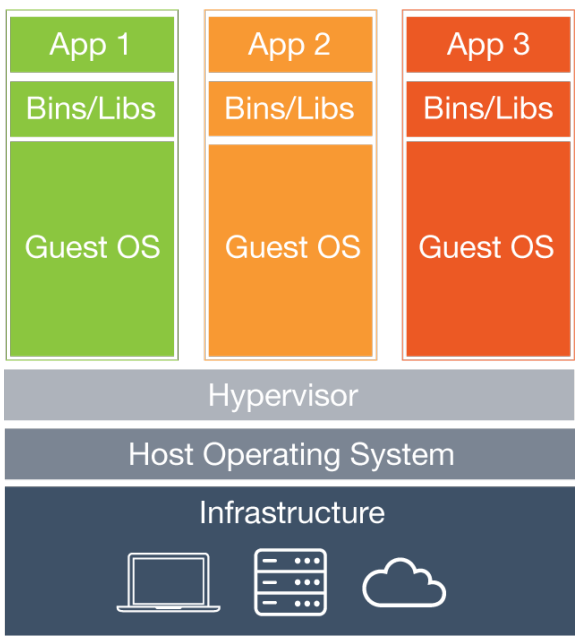
LinuX Containers : LXC

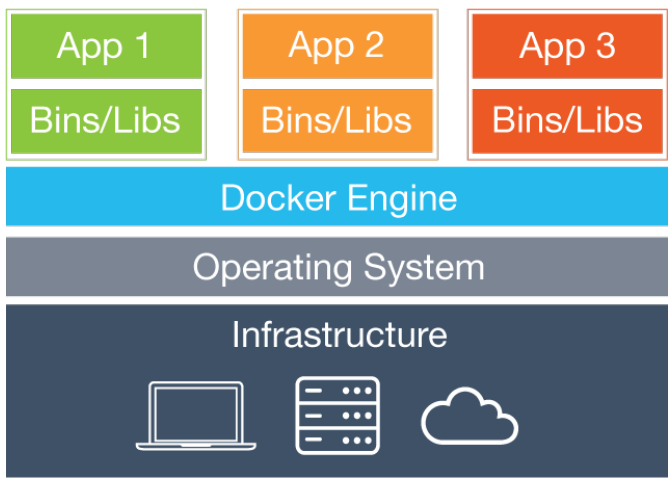
- solution de VPS basée sur “Kernel containers features”
- via bibliothèque userland (libLXC) :
  - Kernel namespaces (ipc, uts, mount, pid, network and user)
  - Apparmor/SELinux profiles
  - Seccomp policies
  - Chroots (pivot_root)
  - Kernel capabilities
  - CGroups (control groups)

Serveurs Privés Virtuels (VPS) : docker

Docker : container engine

- intialement basé sur LXC, à présent sur `runC` (*libcontainer*)
- s'adapte à toute API bas-niveau de container
- déployer un système complet (distrib Linux, comme précédemment)
- idéal pour le déploiement applicatif avec dépendences
- sous-ensemble d'une distrib
- partage de *docker images* : `https://hub.docker.com`

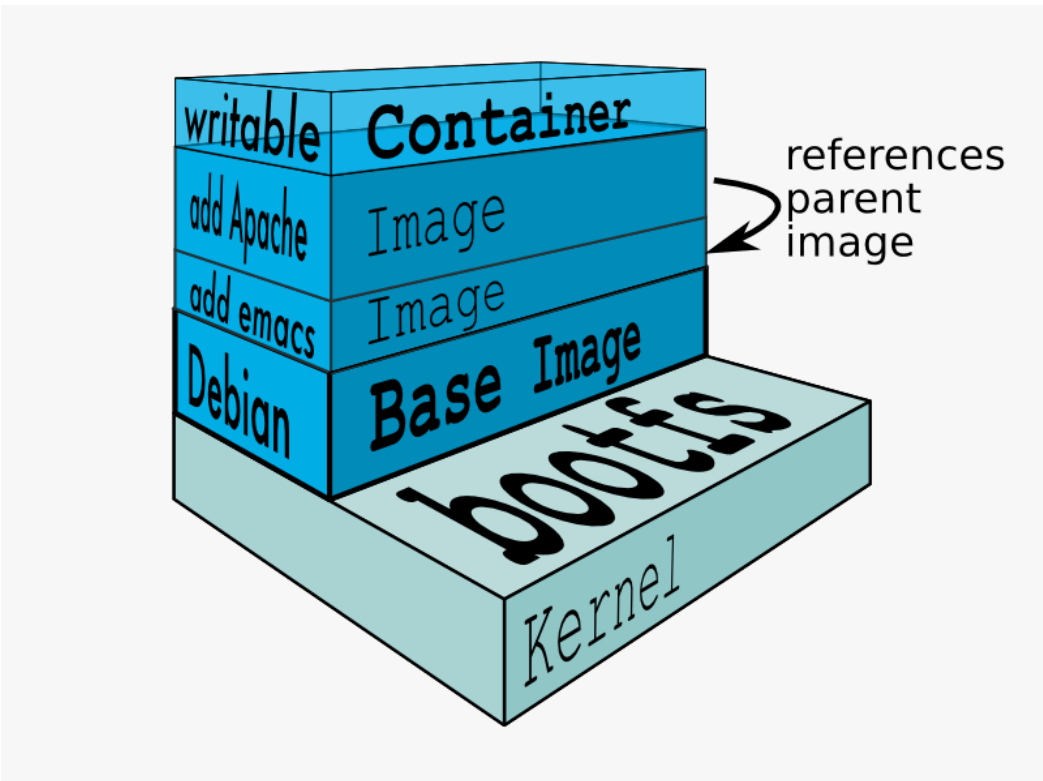




Serveurs Privés Virtuels (VPS) : docker

Docker : container layer

- notion de conteneur parent
- on peut partir d'un conteneur de base
- lui ajouter/empiler des éléments
- basé sur UnionFS



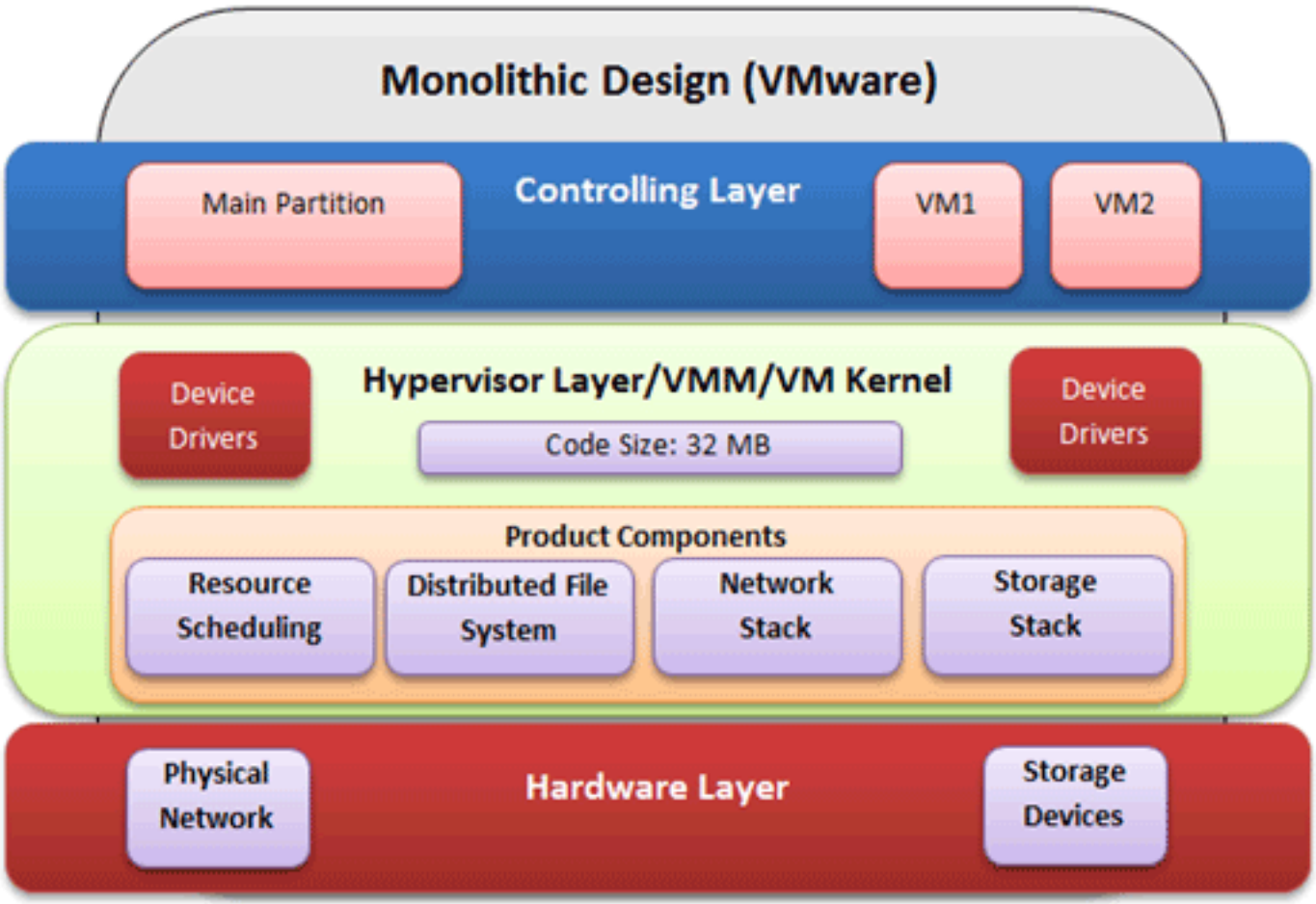
Serveurs Privés Virtuels (VPS) : docker

```
$ sudo apt-get install docker
$ sudo docker search ubuntu
NAME                            DESCRIPTION                                     STARS     OFFICIAL   AUTOMATED
ubuntu                          Ubuntu is a Debian-based Linux operating s...   2712      [OK]
ubuntu-upstart                  Upstart is an event-based replacement for ...   46        [OK]
torusware/speedus-ubuntu        Always updated official Ubuntu docker imag...   25                   [OK]
sequenceiq/hadoop-ubuntu        An easy way to try Hadoop on Ubuntu             24                   [OK]
ubuntu-debootstrap              debootstrap --variant=minbase --components...   20        [OK]
tleyden5iwx/ubuntu-cuda         Ubuntu 14.04 with CUDA drivers pre-installed    18                   [OK]
neurodebian                     NeuroDebian provides neuroscience research...   15        [OK]
rastasheep/ubuntu-sshd          Dockerized SSH service, built on top of of...   15                   [OK]
n3ziniuka5/ubuntu-oracle-jdk    Ubuntu with Oracle JDK. Check tags for ver...   5                    [OK]
sameersbn/ubuntu                                                                5                    [OK]
nuagebec/ubuntu                 Simple always updated Ubuntu docker images...   4                    [OK]
nimmis/ubuntu                   This is a docker images different LTS vers...   3                    [OK]
ioft/armhf-ubuntu               [ABR] Ubuntu Docker images for the ARMv7(a...   3                    [OK]
azukiapp/ubuntu                 Docker image to run Linux Ubuntu (trusty) ...   2                    [OK]
maxexcloo/ubuntu                Docker base image built on Ubuntu with Sup...   2                    [OK]
seetheprogress/ubuntu           Ubuntu image provided by seetheprogress us...   1                    [OK]
sylvainlasnier/ubuntu           Ubuntu 15.04 root docker images with commo...   1                    [OK]
zoni/ubuntu                                                                     0                    [OK]
rallias/ubuntu                  Ubuntu with the needful                         0                    [OK]
esycat/ubuntu                   Ubuntu LTS                                      0                    [OK]
teamrock/ubuntu                 TeamRock's Ubuntu image configured with AW...   0                    [OK]
tvaughan/ubuntu                 https ://github.com/tvaughan/docker-ubuntu       0                    [OK]
konstruktoid/ubuntu             Ubuntu base image                               0                    [OK]
vicamo/ubuntu-phablet-jiexi     Dockerfile for developing Ubuntu JieXi PDK.     0                    [OK]
```

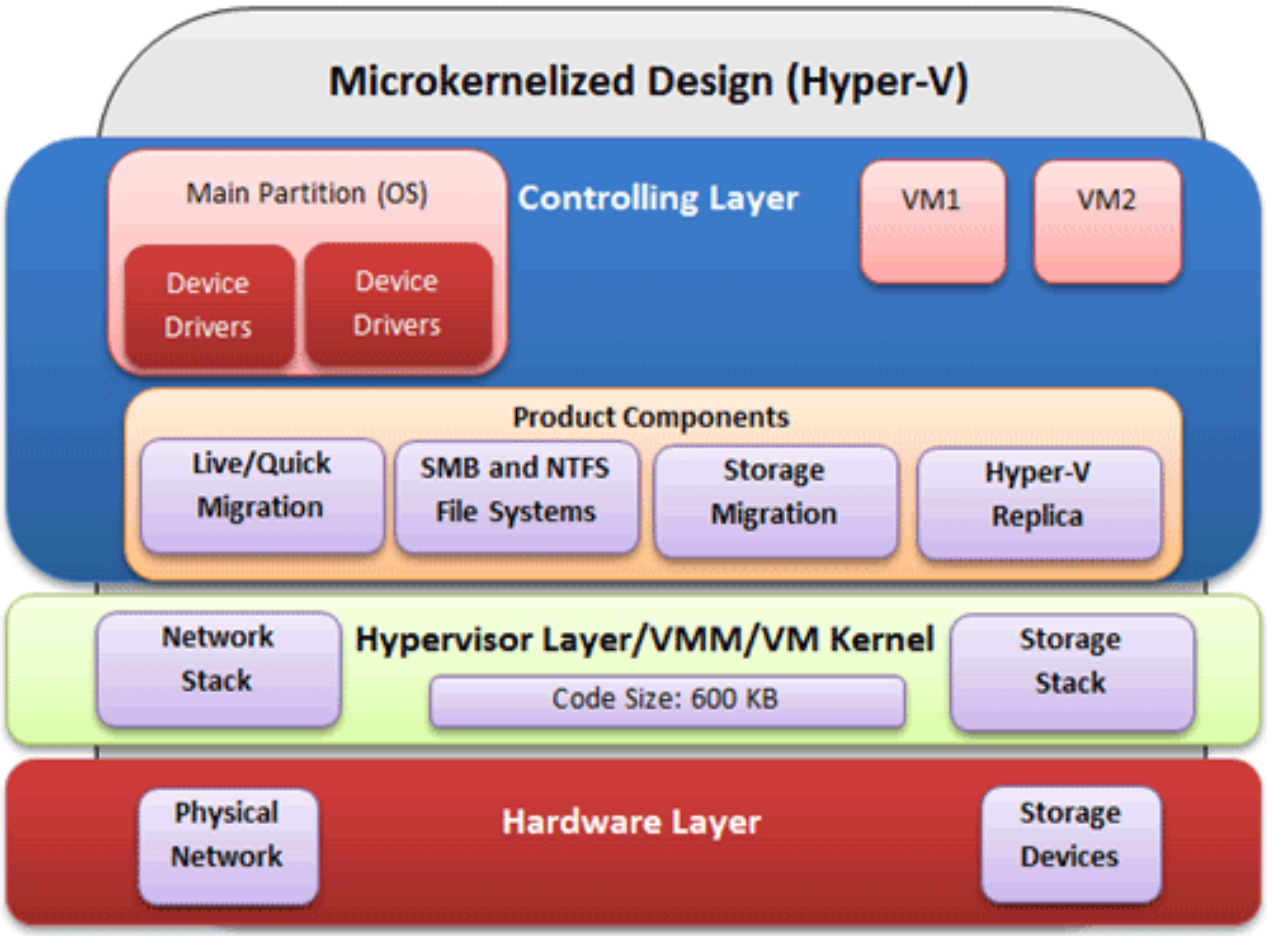
Serveurs Privés Virtuels (VPS) : docker

```
$ sudo docker run -it ubuntu bash
Unable to find image 'ubuntu' locally
Pulling repository ubuntu
ca4d7b1b9a51 : Download complete
2332d8973c93 : Download complete
ea358092da77 : Download complete
a467a7c6794f : Download complete
root@c383ca2bbc83 :/# cat /etc/passwd
root :x :0 :0 :root :/root :/bin/bash
daemon :x :1 :1 :daemon :/usr/sbin :/usr/sbin/nologin
[...]
syslog :x :101 :104 : :/home/syslog :/bin/false

root@8ab6c2bed5d4 :/# ps faux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0  18148  1944 ?        Ss   17 :46   0 :00 bash
root        11  0.0  0.0  15560  1108 ?        R+   17 :48   0 :00 ps faux
root@8ab6c2bed5d4 :/#

root@8ab6c2bed5d4 :/# mount
none on / type aufs (rw,relatime,si=9910fa2461a316b9)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
sysfs on /sys type sysfs (ro,relatime)
tmpfs on /dev type tmpfs (rw,nosuid,mode=755)
shm on /dev/shm type tmpfs (rw,nosuid,nodev,noexec,relatime,size=65536k)
devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=666)
/dev/disk/by-uuid/3cb0facf-6f7c-41ef-b43a-249001b6cb12 on /.dockerinit type ext4 (ro,relatime,errors=remount-ro,data=ordered)
/dev/disk/by-uuid/3cb0facf-6f7c-41ef-b43a-249001b6cb12 on /etc/resolv.conf type ext4 (ro,relatime,errors=remount-ro,data=ordered)
/dev/disk/by-uuid/3cb0facf-6f7c-41ef-b43a-249001b6cb12 on /etc/hostname type ext4 (ro,relatime,errors=remount-ro,data=ordered)
/dev/disk/by-uuid/3cb0facf-6f7c-41ef-b43a-249001b6cb12 on /etc/hosts type ext4 (ro,relatime,errors=remount-ro,data=ordered)
devpts on /dev/console type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=000)
proc on /proc/sys type proc (ro,nosuid,nodev,noexec,relatime)
proc on /proc/sysrq-trigger type proc (ro,nosuid,nodev,noexec,relatime)
proc on /proc/irq type proc (ro,nosuid,nodev,noexec,relatime)
proc on /proc/bus type proc (ro,nosuid,nodev,noexec,relatime)
tmpfs on /proc/kcore type tmpfs (rw,nosuid,mode=755)

^D
$
```

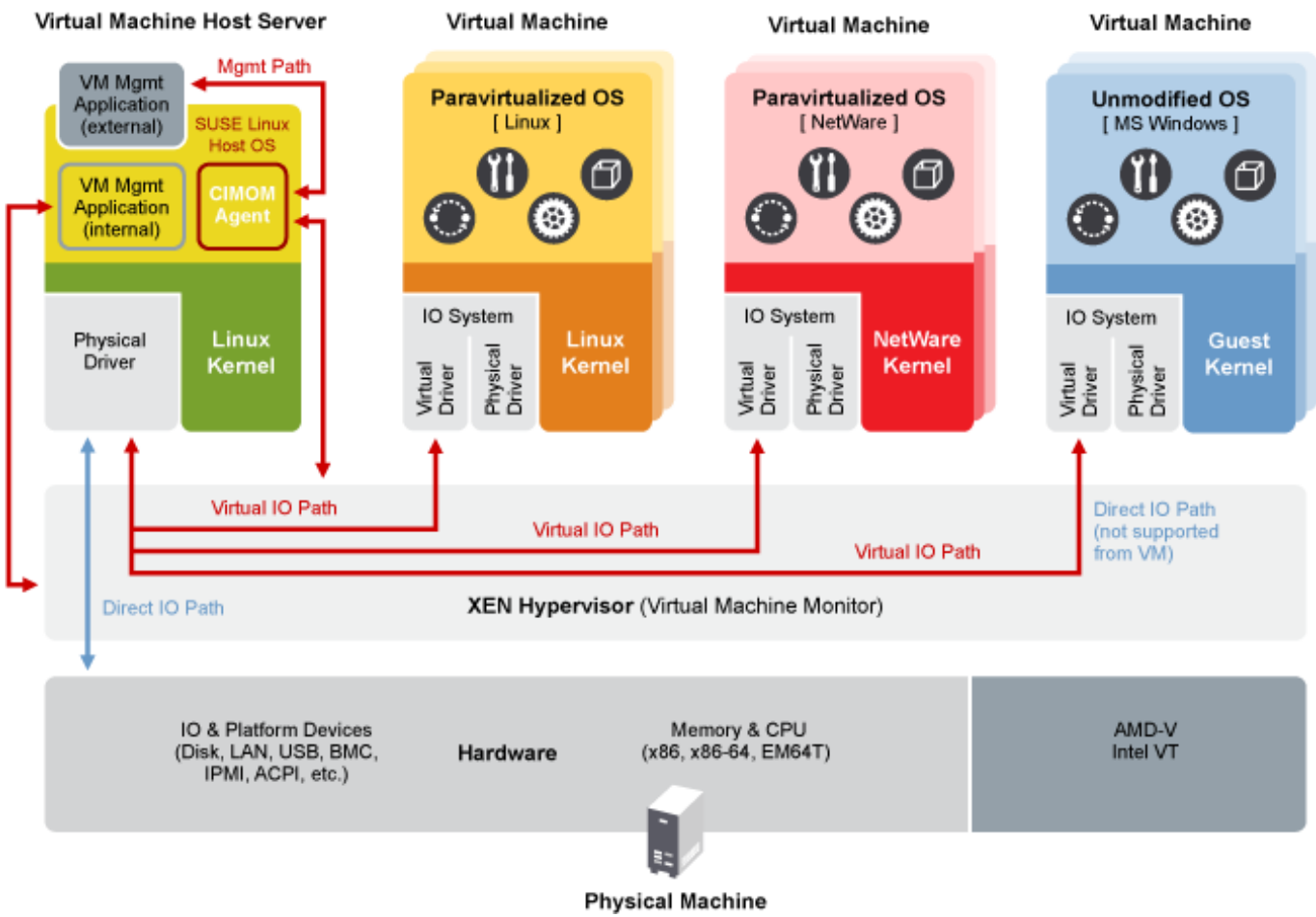
Type 1 VMM - monolithique



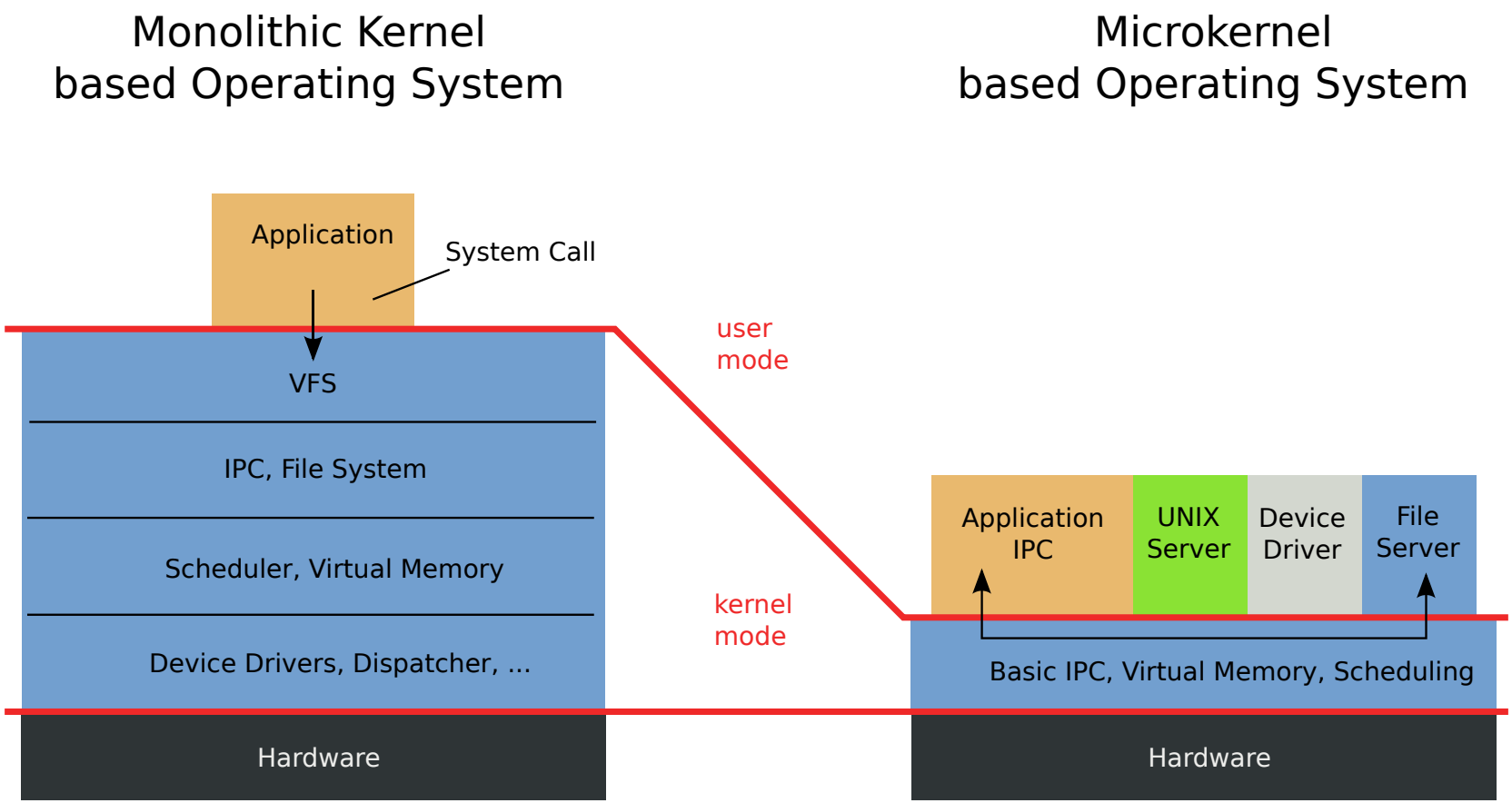
Type 1 VMM - micro-noyau



Xen : micro-kernel + dom0 + domU



Architecture d'un micro-noyau



Micro-noyaux

Principe

- *Liedtke's minimality principle*
- architecture d'OS où le noyau est minimaliste
- uniquement les composants nécessaires au fonctionnement de l'OS
- ring 0 :
  - gestion memoire (entité d'adressage, espace)
  - threads (entité d'execution, temps)
  - communication inter-taches (IPC)
- ring 3 :
  - drivers
  - pile ip/tcp/...
  - file-system
  - ...

Intérêt

- base de code du noyau est plus réduite
- permet une éventuelle certification (EAL, CC)
- sécurité améliorée (surface d'attaque réduite)

Micro-noyaux

Principe

- *Liedtke's minimality principle*
- architecture d'OS où le noyau est minimaliste
- uniquement les composants nécessaires au fonctionnement de l'OS
- ring 0 :
  - gestion memoire (entité d'adressage, espace)
  - threads (entité d'execution, temps)
  - communication inter-taches (IPC)
- ring 3 :
  - drivers
  - pile ip/tcp/...
  - file-system
  - ...

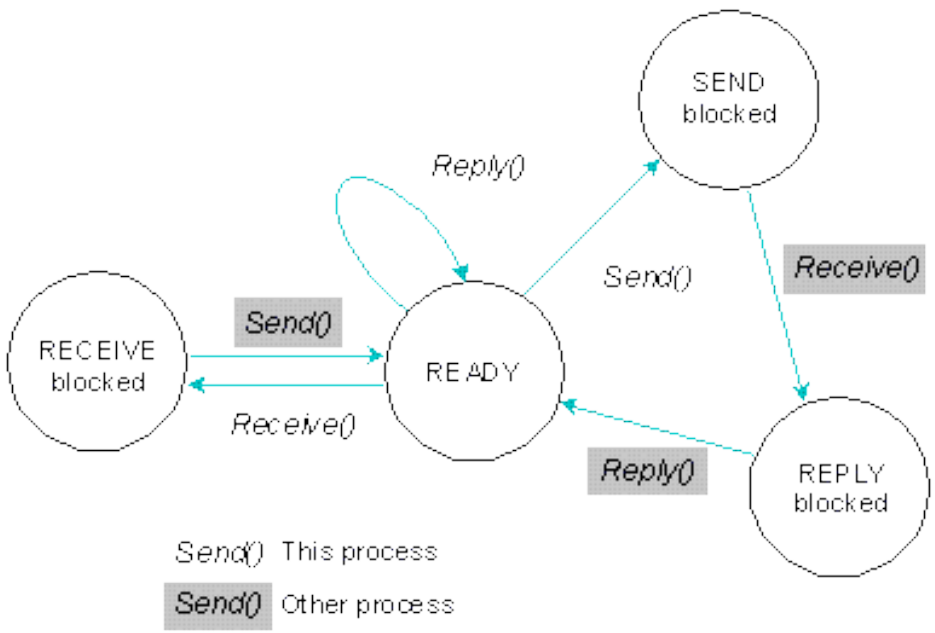
Intérêt

- base de code du noyau est plus réduite
- permet une éventuelle certification (EAL, CC)
- sécurité améliorée (surface d'attaque réduite)
- beaucoup plus lent (allers/retours user/kernel)

Micro-noyaux : IPC

Inter Process Communication

- synchrones ou asychrones
- ni plus ni moins qu'un système d'échange de messages
- entre d'applications à application
- en passant par le noyau
- via ses quelques appels systemes :
  - `ipc_send()`
  - `ipc_receive()`
  - `ipc_wait()`



Une liste de micro-kernels

- L4 : seL4, OKL4, Fiasco, NOVA
- PikeOS, QNX, LynxOS
- Minix, Mach, GNU Hurd, XNU (Darwin OSX)

L4 micro-kernel

Micro-Kernel optimisé : 2nd génération

- Fast IPC :
  - send/receive enchainé
  - passage des valeurs par registres
  - mapping mémoire entre espace d'adressage plutot que recopie de buffer
  - scheduling immédiat de récepteur (ipc synchrone)
- bénéfice pour les systèmes temps-réel
- famille nombreuse :
  - OKL4 pour les mobiles (embedded virtualization)
  - SEL4 preuve formelle de son API kernel

PikeOS

Yet Another Micro Kernel

- développé par SYSGO
- racheté par THALES
- inspiré de L4 meme s'ils ne le disent pas :)
- micro-kernel temps réel
- hyperviseur (embedded virtualization)
- propose de nombreuses API :
  - support linux (eLinOS)
  - applications natives pikeOS
  - applications POSIX
  - ARINC 653 (standard securité avionique : partionnement temps/espace)

PikeOS

Virtualisation par Partitions

- pas de notion de VM mais de partitions
  - time partitions : scheduling temps réel
  - resources partitions : mémoire, devices
- une sorte de lib kernel commune aux partitions : `PSSW`

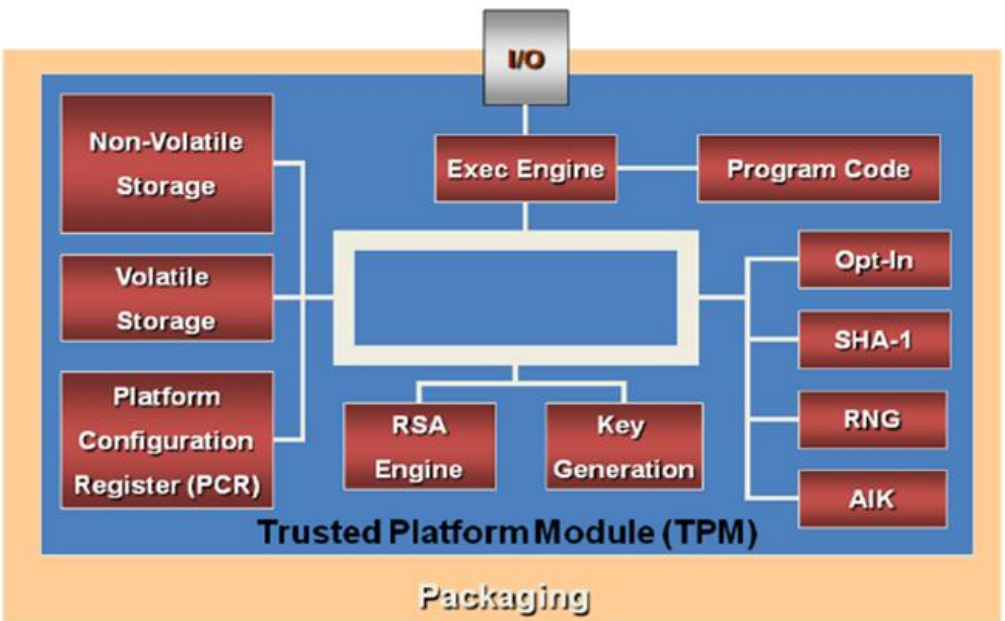
Platform System SoftWare : PSSW

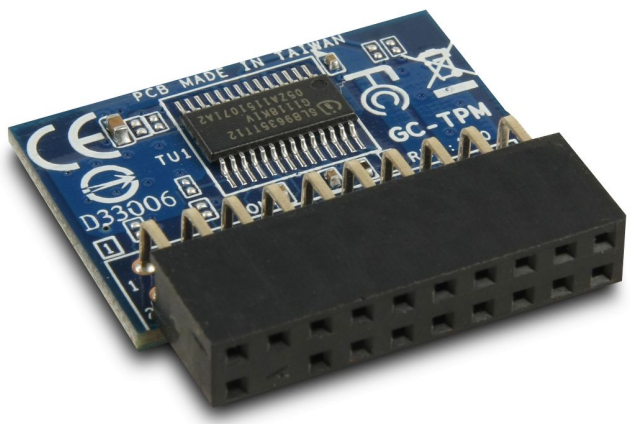
- dans une tâche spécifique d'une partition système
- chaque partition dispose d'un daemon thread qui tourne dans l'espace d'adressage de la tâche `PSSW`
- la `PSSW` est un noyau monolithique ring 3 qui tourne au dessus du micro-noyau et de ses services minimalistes (IPC, scheduling, mapping memoire)
- abstrait les IPC via des servcices : memoire partagee, serveur de fichiers
- nombreuses restrictions :
  - daemon thread d'une partition peut recevoir des IPC
  - uniquement des taches de sa ressource partition
  - daemon threads communiquement entre eux grace a des events
  - elles ne peuvent s'echanger qu'au sein d'une meme tache.

Le TPM et ses implications

Trusted Platform Module

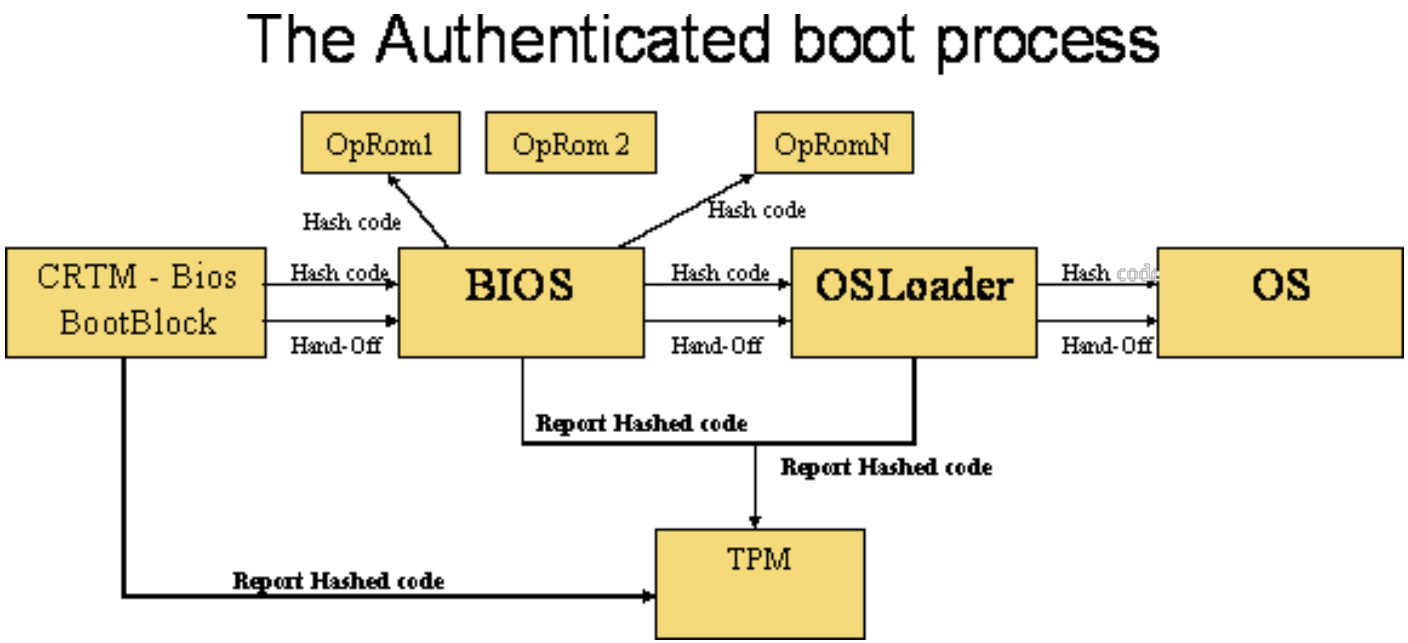
- définit par le TCG (Trusted Computing Group)
- standard d'informatique de confiance
- puce cryptographique qui vérifie (via signature) un composant logiciel
- le TPM implique une gestion de clés et des protocoles



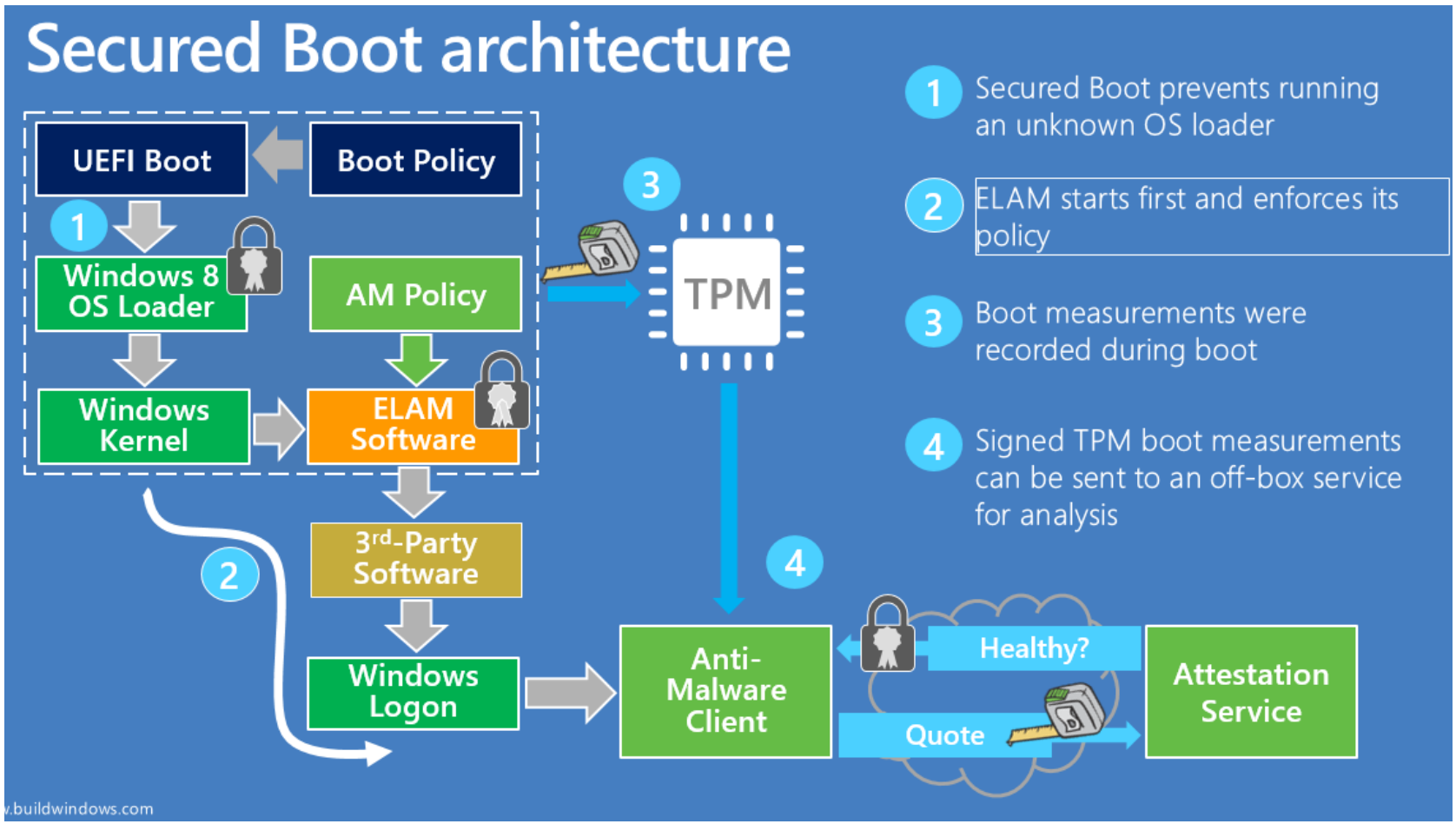


Le TPM et ses implications

- savoir ce que l'on boot
- chaine de démarrage de confiance
- qui par la suite valide un kernel ... jusqu'a démarrage complet de l'OS
- previent les attaques à froid (backdooring)
- Root of Trust for Measuring (RTM,RTR,RTS), gestion des hashs des éléments de la phase de boot



Exemple ; Windows 8 Secured Boot



I/O MMU

Intérêt du point de vue de la sécurité

- principe d'une MMU au niveau des périphériques
- contrôler les accès DMA
- cloisonner des périphériques

I/O MMU

Intérêt du point de vue de la sécurité

- principe d'une MMU au niveau des périphériques
- contrôler les accès DMA
- cloisonner des périphériques

Implémentation

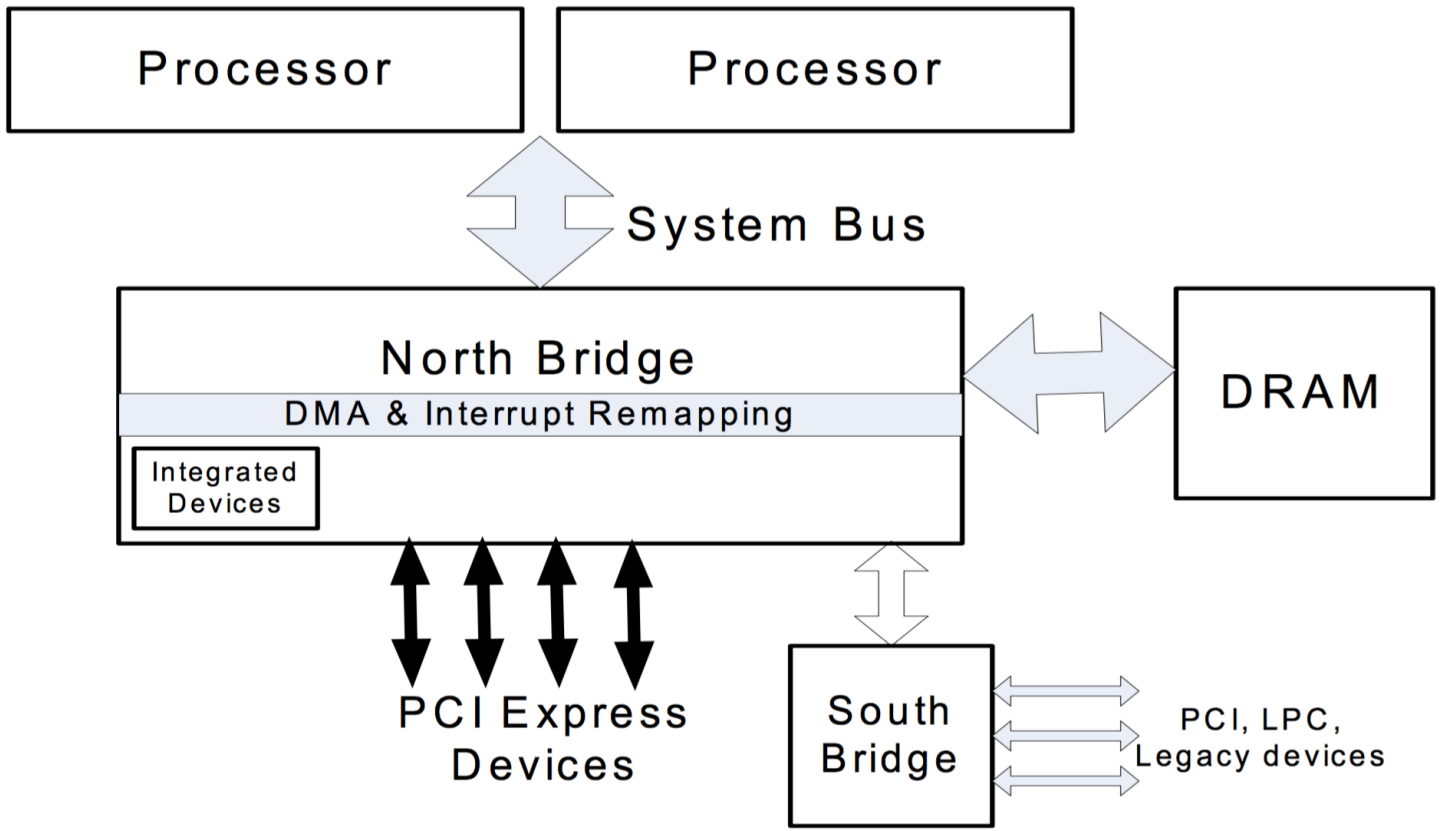
- AMD : DEV (Device Exclusion Vectors), lié à SVM
- Intel : Intel vt-d (VT for Directed I/O) :
  - pas lié au CPU
  - mais au chipset (X58, Q87, Z77) de la carte mère
  - propose *Interrupt/DMA Remapping*

Liens utiles

- `https://en.wikipedia.org/wiki/List_of_IOMMU-supporting_hardware`
- `http://www.intel.fr/content/www/fr/fr/support/boards-and-kits/desktop-boards/000005758.html`
- `http://www.intel.com/content/dam/www/public/us/en/documents/product-specifications/vt-directed-io-spec.pdf`

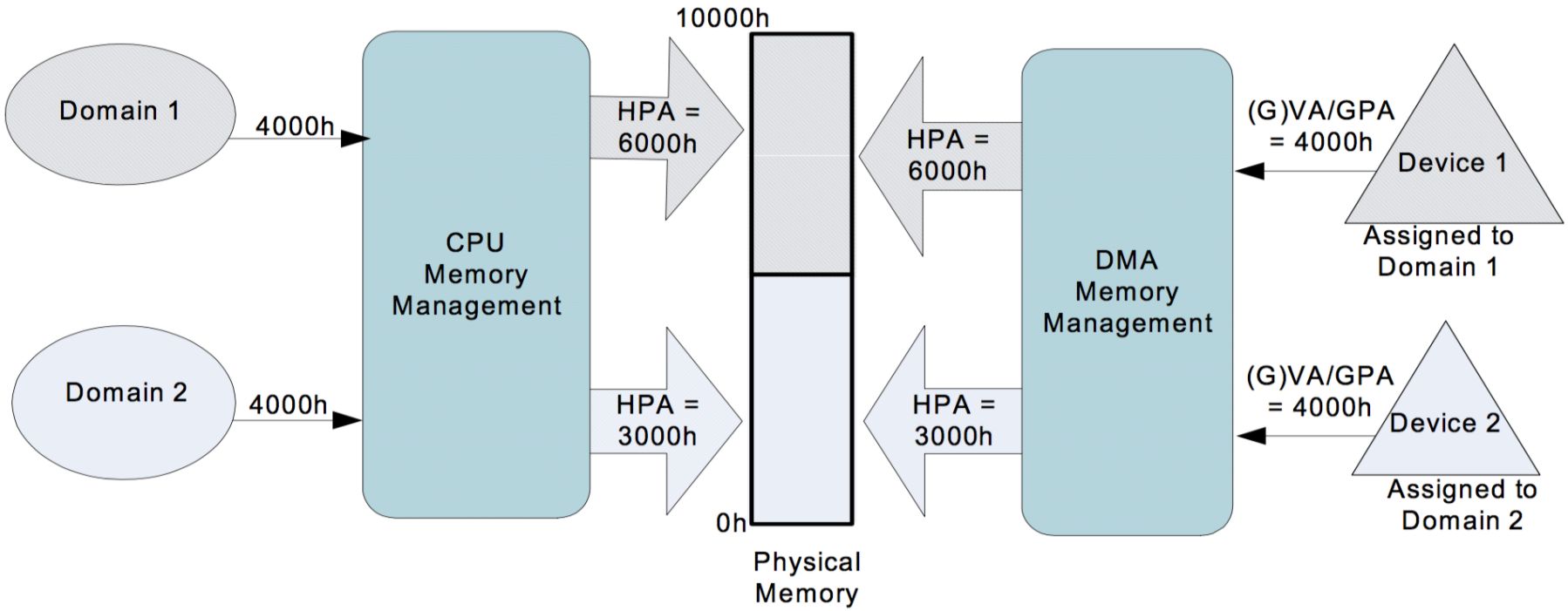
I/O MMU : localisation

- avant le contrôleur de DRAM
- dans le north-bridge à cause des périphériques intégrés, GPUs



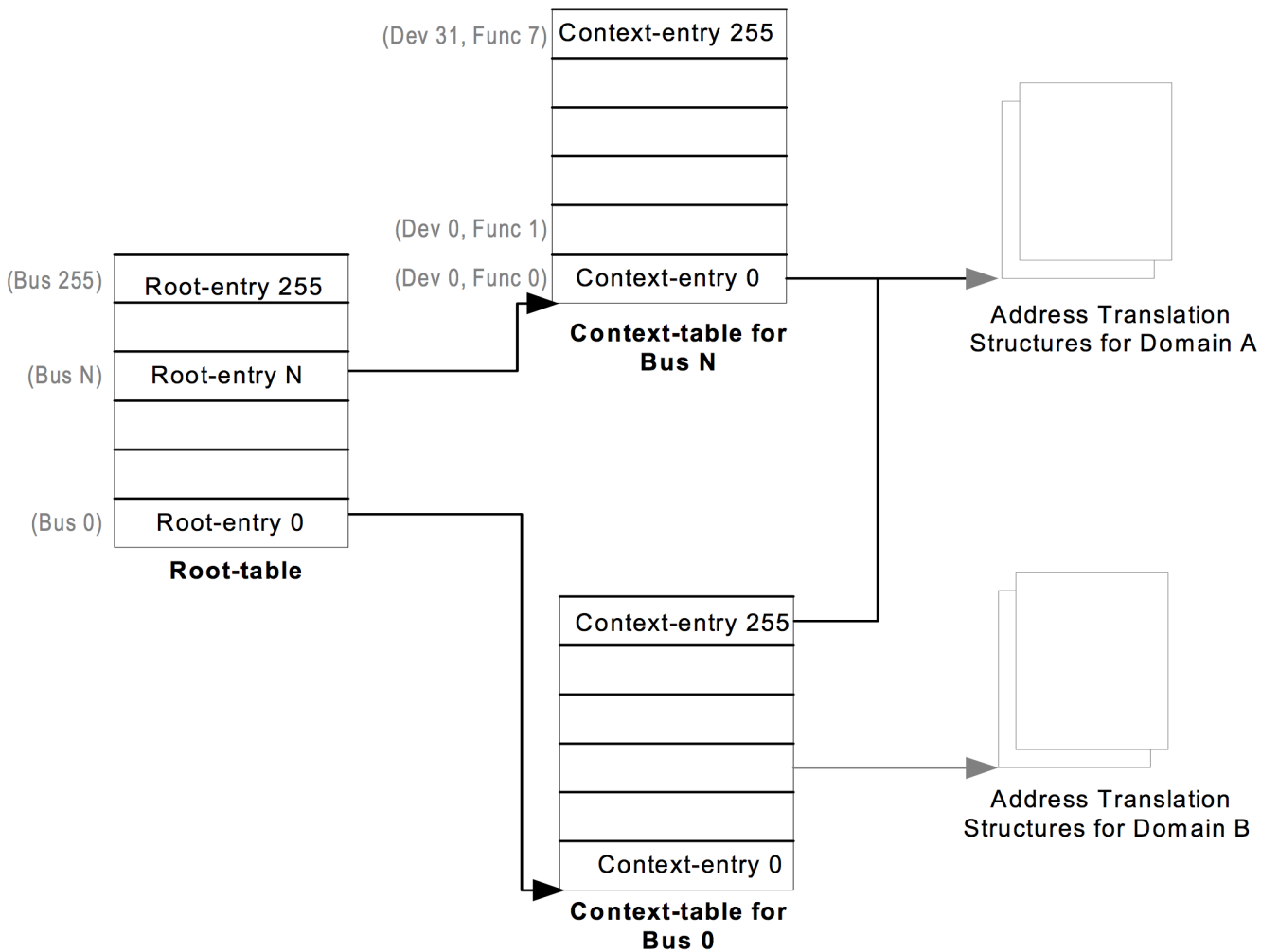
I/O MMU : isolation

- mimer les mécanismes d'une MMU
- adresses physiques des périphériques 1 & 2 deviennent virtuelles



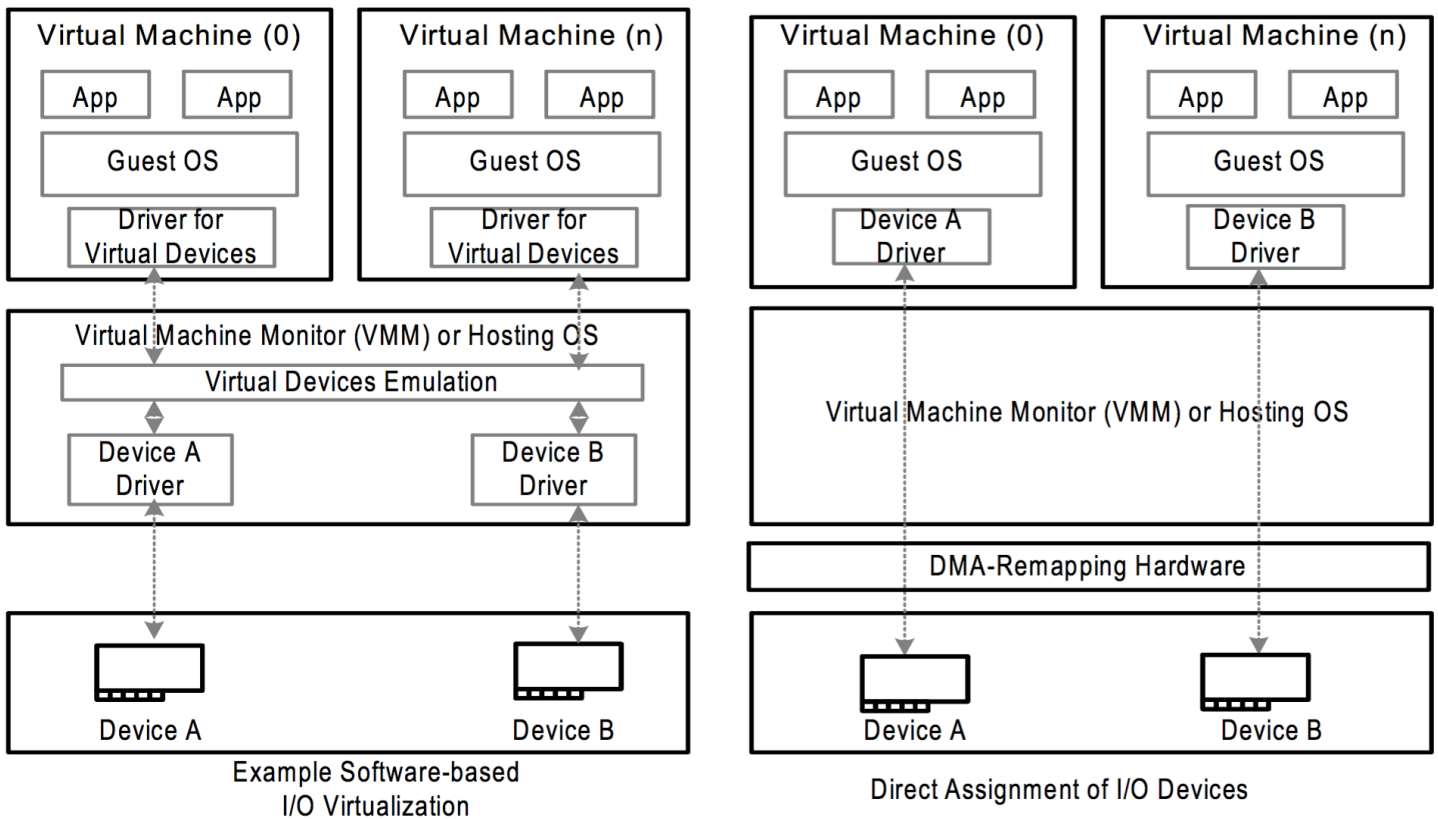
Intel VT-d : fonctionnalités

- plusieurs DRHD (DMA Remapping Hardware Device) par chipset
- un DRHD gère un ou plusieurs périphérique(s)
- les DRDH sont listés dans une table ACPI : DMAR
- traduction d'adresses par *source ID* (PCI bus :dev :fnc)



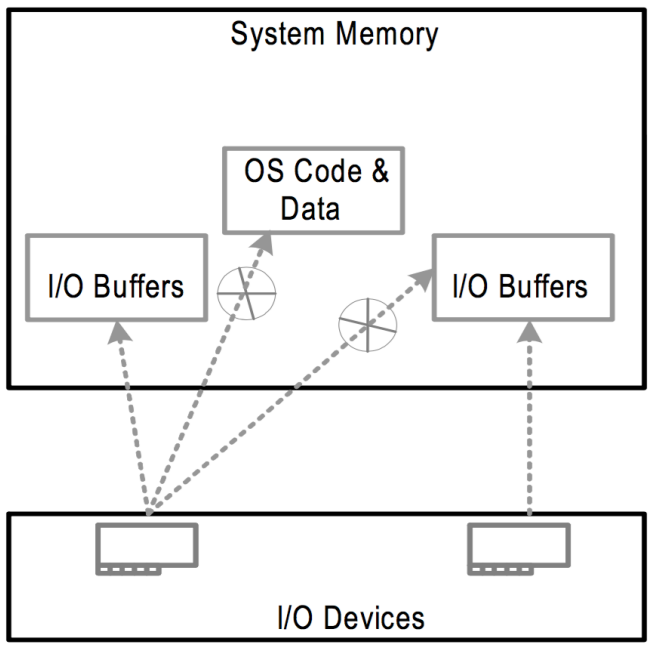
Intel VT-d : Intérêt pour un hyperviseur

- DMA remapping : traduction d'adresse au niveau des périphériques
- I/O device assignment : allocation de périphériques par VM
- Interrupt remapping : redirection d'interruptions par VM
- Gestion d'erreurs : *#PF* des périphériques remontées au logiciel (vmm/os)

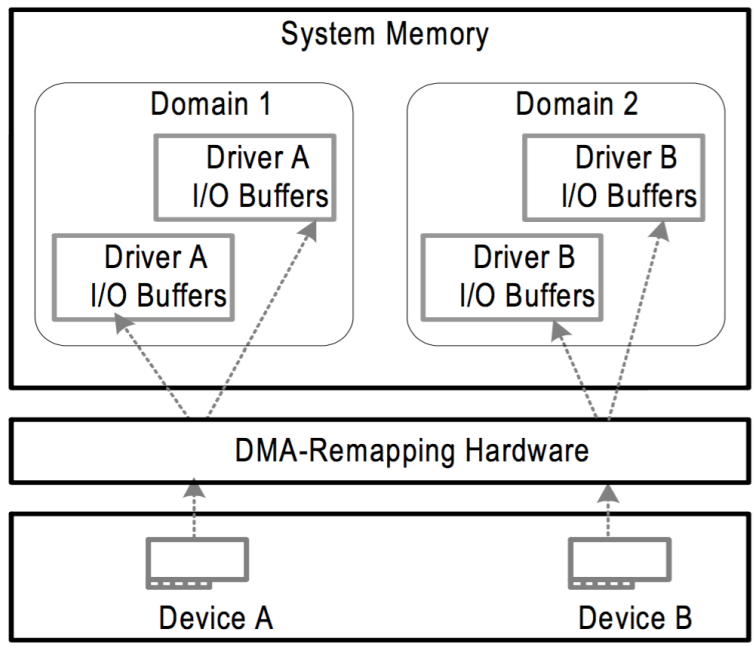


Intel VT-d : Intérêt pour un OS

- protection : isoler le code et les données de l'OS, des drivers de périphériques et de leurs I/O buffers (pensez 3rd party)
- DMA isolation : isoler les périphériques les uns des autres, créer des domaines



Device DMA without isolation



Device DMA isolated using DMA remapping hardware

ACPI

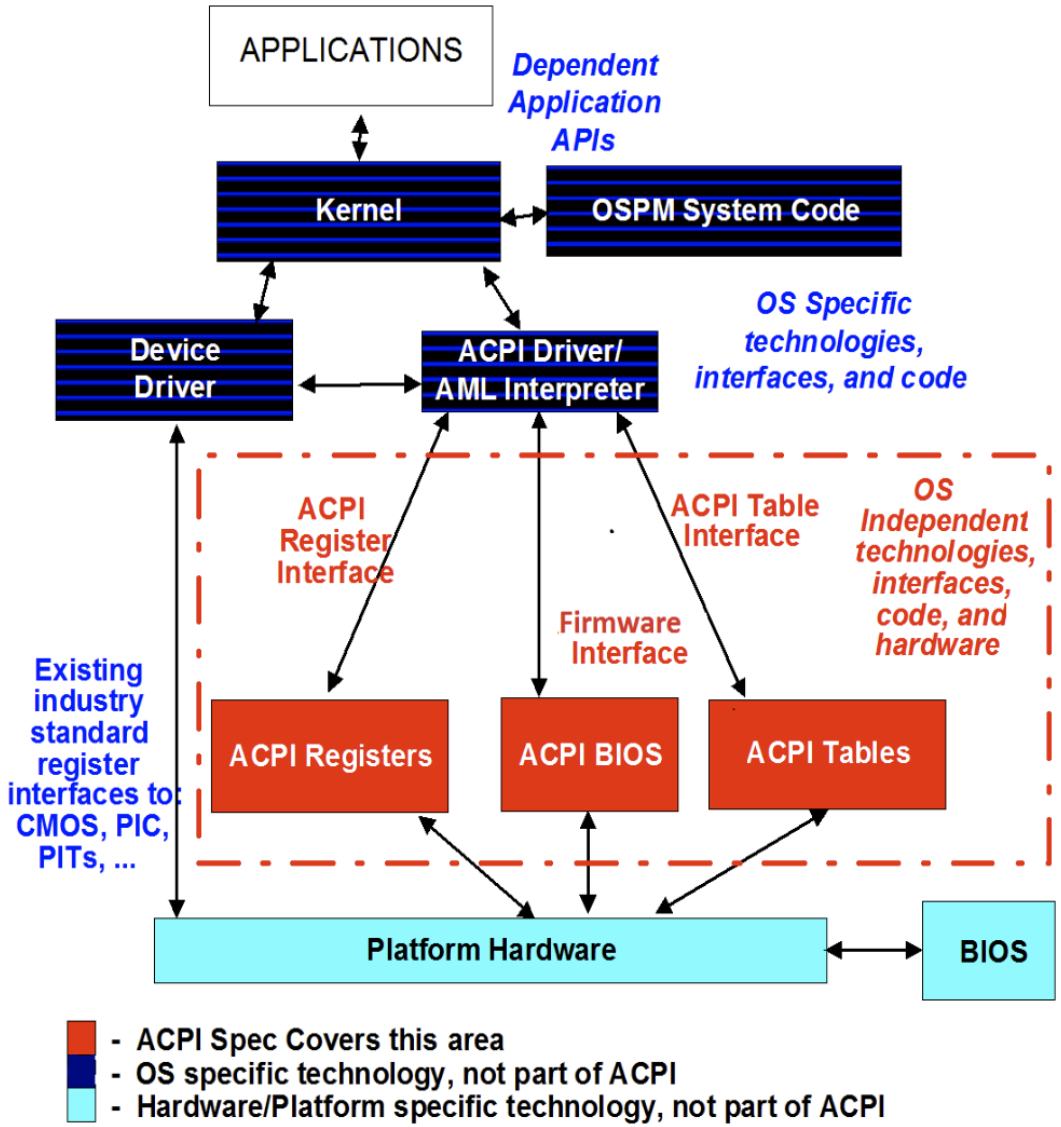
Historique

- nombreux périphériques et contrôleurs initialisés par le bios
- APM, Plug'n'Play, ... dépassés
- volonté de standardiser la détection/configuration des périphériques sur PC
- gestion avancée de l'énergie dans le même temps (CPU et périphériques)
- ACPI : Advanced Configuration & Power Interface
- norme ACPI 6.1a (Janvier 2016)

ACPI : core features

Composants principaux

- ACPI System Description Tables
- ACPI Registers
- ACPI Platform Firmware



ACPI

OSPM

- OS compliant avec la spécification ACPI
- embarque une interpréteur ASL/AML
- AML : ACPI Machine Language (Bytecode)
- ASL : ACPI Source Language (compilé vers AML)

ACPI

OSPM

- OS compliant avec la spécification ACPI
- embarque une interpréteur ASL/AML
- AML : ACPI Machine Language (Bytecode)
- ASL : ACPI Source Language (compilé vers AML)

ACPI Component Architecture

- implémentation de référence
- indépendante de l'OS
  - interpréteur AML
  - parcours des tables
  - accès aux registres

ACPI

Gestion de l'énergie

- Gx/Sx pour le système global
- Cx pour le CPU
- Dx pour les périphériques

Description des états

- S0, en marche
- S3, *suspend to ram*. Sauve l'état du système en RAM. Seule la RAM est alimentée
- S4, *suspend to disk*. Sauve l'état du système sur le disque.
- Réveil : firmware ACPI appelle un *wake up vector* configuré par l'OSPM dans la table FACS.

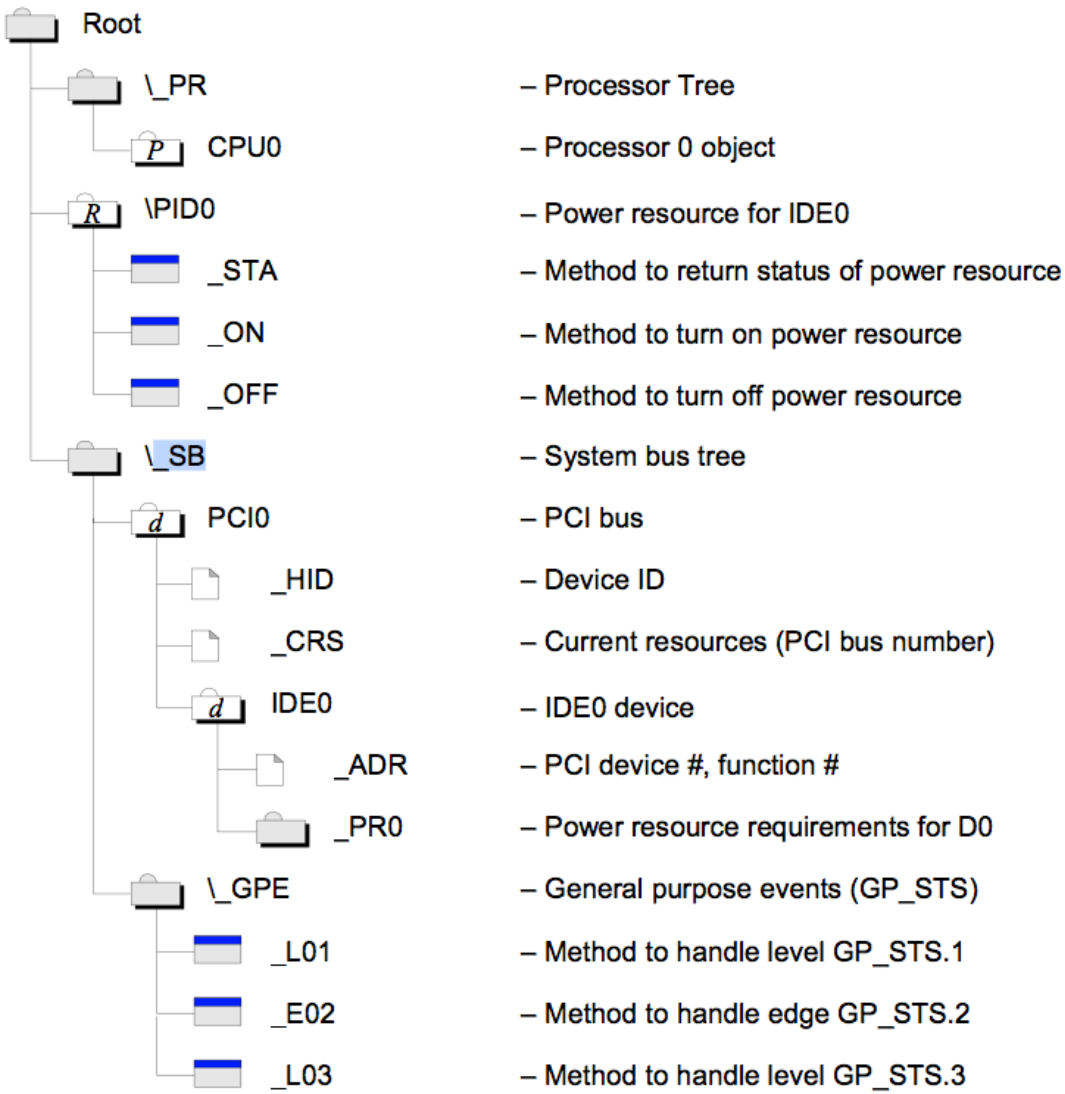
`https://wiki.ubuntu.com/Kernel/Reference/S3`

ACPI

Quelques tables ACPI usuelles

- RSDP (Root System Description Pointer)
- RSDT (Root System Description Table)
- DSDT (Differentiated System Description Table)
- XSDT (Extended System Description Table)
- FADT (Fixed ACPI Description Table)
- FACS (Firmware ACPI Control Structure)
- SBST (Smart Battery Table)
- ECDT (Embedded Controller Boot Resources Table)
- MADT (Multiple APIC Description Table)
- SRAT (System Resource Affinity Table)
- SLIT (System Locality Distance Information Table)
- SLIC (Software Licensing Description Table)
- SSDT (Secondary System Descriptor Tables)
- THRM (CPU Thermal)

ACPI : organisation de l'espace ACPI



ACPI : exemple ASL

```
{
  OperationRegion(\GIO, SystemIO, 0x125, 0x1)
  Field(\GIO, ByteAcc, NoLock, Preserve) {
        CT01, 1,
  }

  Scope(\_SB)
     Device(PCI0) {
         PowerResource(FET0, 0, 0) {
             Method (_ON)   {
                 Store (Ones, CT01)
                 Sleep (30)
             }
             Method (_OFF) {
                 Store (Zero, CT01)
             }
             Method (_STA) {
                 Return (CT01)
             }
         }
     }
  }
}
```

ACPI : sécurité

Un impact certain

- vulnérabilités dans l'implémentation au niveau de l'OS ?
- validation/protection des tables ACPI ?
- confiance dans le firmware ACPI ?
- `http://actes.sstic.org/SSTIC09/ACPI_et_routine_de_traitement_de_la_SMI/SSTIC09-article-L-Duflot-_O-Levillain-ACPI_et_routine_de_traitement_de_la_SMI.pdf`
- `http://www.intelsecurity.com/advanced-threat-research/content/WP_Intel_ATR_S3_ResBS_Vuln.pdf`

Extensions de sécurité chez Intel

- WP : write protect
- NX : execute disable
- SMAP/SMEP : Supervisor Mode Access/Execution Protection
- MPX : Memory Protection Extensions
- CET : Control-flow Enforcement Technology
- SGX : Software Guard Extensions

`https://github.com/huku-/research/wiki/Intel-CPU-security-features`

Extensions de sécurité chez Intel

Write Protect

- au niveau de *CR0*
- empêche ring 0 d'écrire dans des pages *read-only*
- utile pour le *copy-on-write*

Extensions de sécurité chez Intel

Execute Disable

- au niveau du MSR : *IA32_EFER*
- empêche l'exécution par entrée de PDE/PTE
- uniquement en PAE ou LongMode (64 bits)

Extensions de sécurité chez Intel

Supervisor Mode Access Protection

- au niveau de CR4
- empêche ring 0 d'accéder à des pages *user :ring 1,2,3*
- désactivable via EFLAGS.AC

Supervisor Mode Execution Protection

- au niveau de CR4
- empêche ring 0 d'exécuter des pages *user :ring 1,2,3*

Extensions de sécurité chez Intel

Memory Protection Extensions

- protection contre les débordements de tampons
- jeu d'instructions supplémentaires : *BNDLDx, BNDSTx, BNDCL, BNDCU*
- et de nouveaux registres : *BNDx, BNDMK, BNDSTATUS*
- consultation de *bound tables* définissant des buffers (start,end)
- implémenté par les compilateurs (*gcc -mmpx -fcheck-pointer-bounds*)

Extensions de sécurité chez Intel

Control-flow Enforcement Technology

- protection contre le ROP/JOP
- *shadow stack*
- *indirect branch tracking*

Extensions de sécurité chez Intel

CET : Shadow Stack

- pile dédiée aux transferts de contrôle (adresses de retour)
- dissociée de la pile de données (variables locales, arguments)
- *CALL* sauvegarde l'adresse de retour sur les deux piles
- *RET* récupère l'adresse sur chaque pile et compare les valeurs
- si différentes, génère nouvelle exception *#CP*

Extensions de sécurité chez Intel

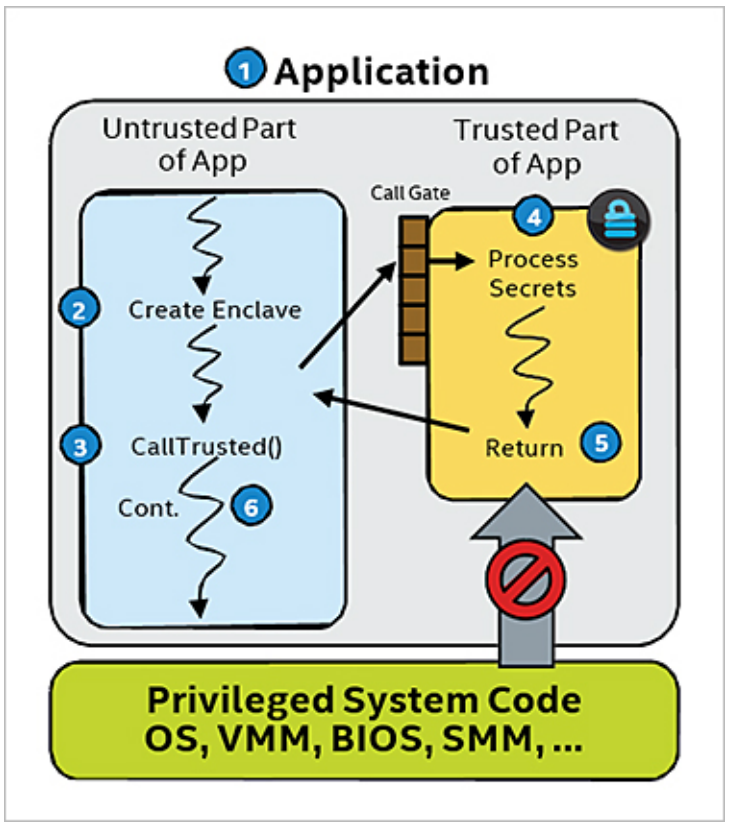
CET : Indirect Branch Tracking

- branchements indirects (*jmp [eax]*, *call *ecx*)
- concerne généralement l'implémentation des *switch case*
- nouvelle instruction *ENDBR*
- à chaque saut indirect :
  - le CPU passe d'*IDLE* à *WAIT_FOR_ENDBR*
  - si prochaine instruction n'est pas *ENDBR*, génère *#CP*
  - sinon repasse *IDLE*

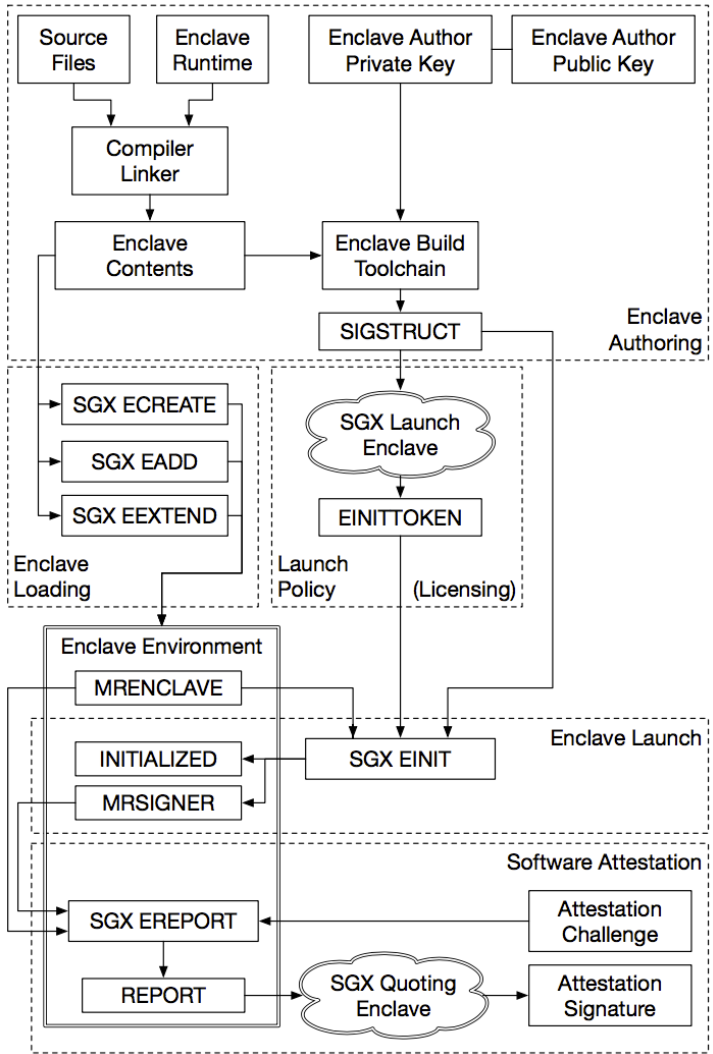
Extensions de sécurité chez Intel

Software Guard Extensions

- nécessiterait un cours dédié au sujet
- solution de protection d'applications (ring 3) en intégrité et confidentialité
- basé sur le concept d'*enclaves* chiffrées/signées
- toutes les opérations cryptographiques sont faites en *hardware*



Intel SGX


Source Files
Enclave Runtime
Enclave Author Private Key
Enclave Author Public Key
Compiler Linker
Enclave Contents
Enclave Build Toolchain
SIGSTRUCT
Enclave Authoring
SGX ECREATE
SGX EADD
SGX EEXTEND
Enclave Loading
SGX Launch Enclave
EINITTOKEN
Launch Policy
(Licensing)
Enclave Environment
MRENCLAVE
INITIALIZED
MRSIGNER
SGX EINIT
Enclave Launch
Software Attestation
SGX EREPORT
REPORT
SGX Quoting Enclave
Attestation Challenge
Attestation Signature

Intel SGX

Limitations

- inconnues sur les opérations crypto implémentées en hard ?
- quid d'une backdoor de la NSA ?
- initialement seul Intel pouvait signer des enclaves
- Intel SGX Memory Encryption Engine
  `https://eprint.iacr.org/2016/204.pdf`
- Intel SGX Explained
  `https://eprint.iacr.org/2016/086.pdf`

Attaques sur les protections d'Intel

Contourner SMEP

- faire du ROP pour désactiver *CR4.SMEP*
- utiliser un buffer ring0 qui pointe physiquement vers des données contrôlées en ring3
- *synonym mappings* : plusieurs mappings virtuels de différentes natures vers des pages de mémoire physiques identiques

Quelques pointeurs

- j00ru//vx tech blog : SMEP : What is it, and how to beat it on Windows
  `http://j00ru.vexillium.org/?p=783`
- Xen SMEP (and SMAP) bypass - NCC Group
  `https://www.nccgroup.trust/uk/about-us/newsroom-and-events/blogs/2015/april/xen-smep-and-smap-bypass/`
- Efficacité de CET : PaX RAP vs Intel CET
  `https://forums.grsecurity.net/viewtopic.php?f=7&t=4490`

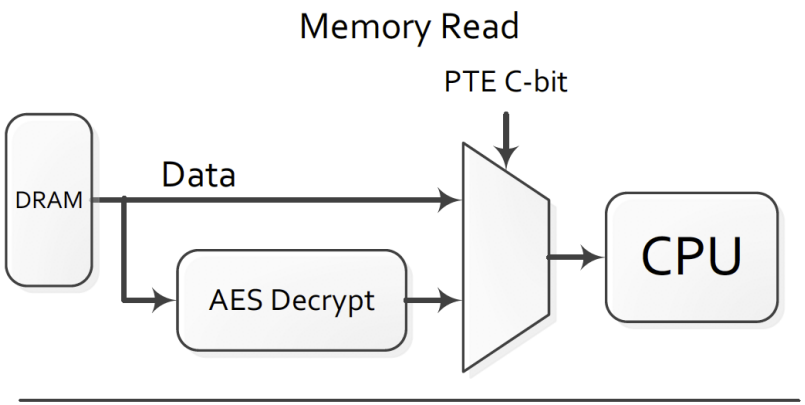
AMD Memory Encryption

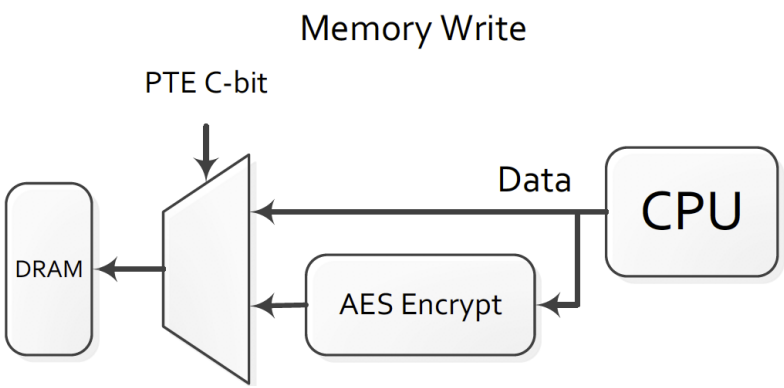
Chiffrement de la mémoire

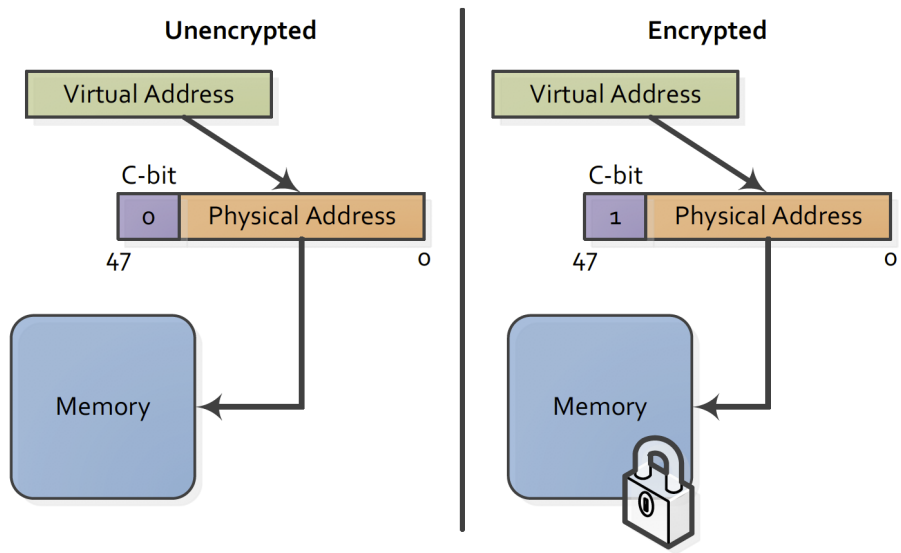
- SME : Secure Memory Encryption
- SEV : Secure Encrypted Virtualization

AMD SME : Secure Memory Encryption

- chiffrement à la volée des pages de mémoire en RAM
- clef invisible générée à chaque reset, pour chaque Core
- AES128 + mode (*address based tweak*)
- AMD Secure Processor (ARM Cortex A5) s'occupe des opérations crypto
- bit 'C' dans les entrées de PDE/PTE pour déclencher le chiffrement
- DMA supporté vers les pages chiffrées (simple accès mémoire)

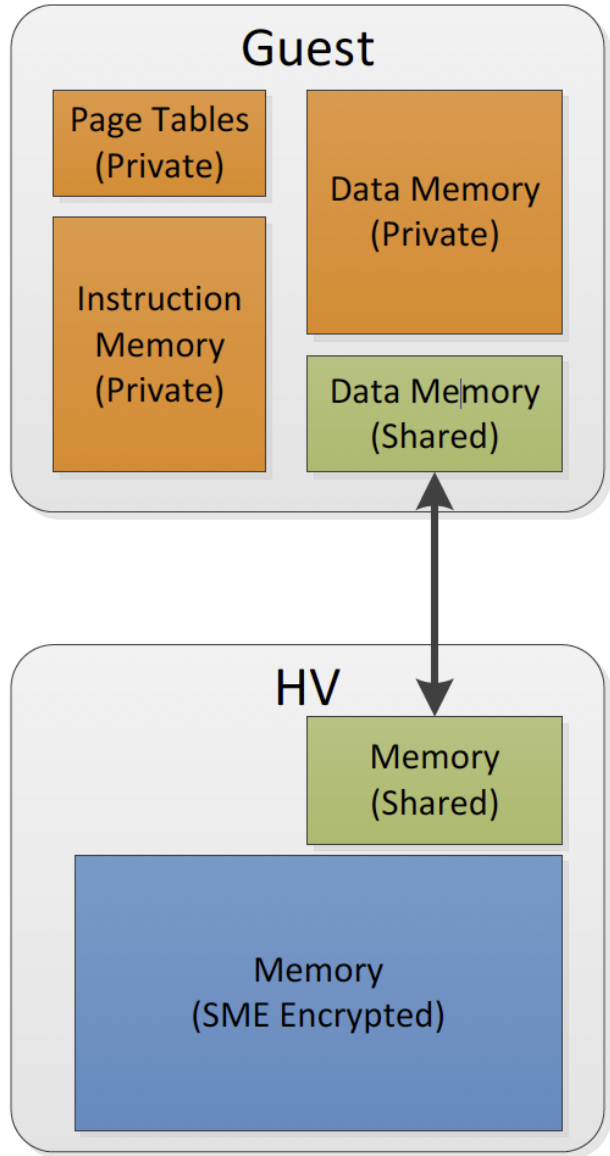






AMD SEV : Secure Encrypted Virtualization

- même technologie que SME
- protection des VMs entre elles
- protection contre leur hyperviseur
- un ASID par VM, une clef AES par ASID
- notion de pages privées (guest key)/partagées (vmm key)
- ***key management*** :
  - *firmware* SEV gère les clefs
  - VMM/VMs doivent communiquent avec SEV
- DMA uniquement sur des *shared pages*



AMD SEV : Secure Encrypted Virtualization

- Authentification : Root of Trust
  - *identity key* signée par AMD et l'admin de la machine
  - authentifie le *firmware* SEV
- Attestation :
  - signature des VMs (éléments SEV) avant leur démarrage
  - garantie l'état du Guest (measurement) au démarrage de SEV
- Confidentialité :
  - mémoire chiffrée grâce à SME
  - les clefs ne sortent pas du SOC
  - échange DH entre utiliseur (Guest Owner) et AMD-SP