



# Programmed I/O accesses: a threat to Virtual Machine Monitors?

Loïc Duflot & Laurent Absil  
Central Department for  
Information Systems Security

SGDN/DCSSI 51 boulevard de la Tour Maubourg 75007 Paris  
[loic.duflot@sgdn.pm.gouv.fr](mailto:loic.duflot@sgdn.pm.gouv.fr)

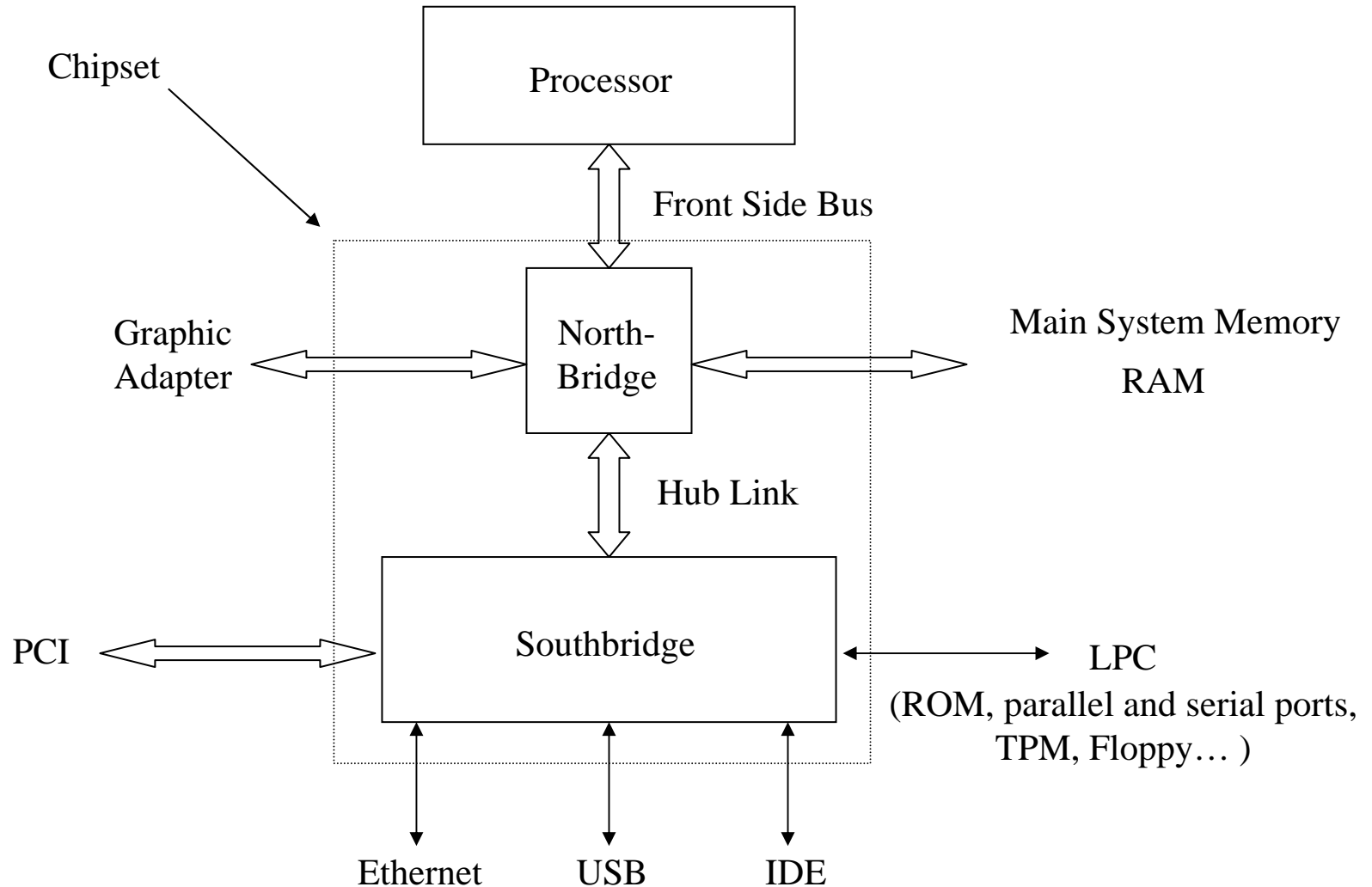
# Introduction

- Main goal of the presentation: show (once more) how hardware functionalities can be misused as a means for privilege escalation over a PC (x86, x86-64) system.
- Documented hardware mechanisms working according to their specifications can be used from the application level to bypass operating system security mechanisms (no physical access to the system is required).
- We show a proof of concept use of such privilege escalations in the context of virtualization.
- What is the level of trust that can be achieved even when the hardware works strictly as documented?

# Outline

- Introduction
- Quick recap on PC architectures and I/O accesses
- Using chipset mechanisms from the application level to bypass security functions
  - AGP graphics aperture
  - UHCI-compliant USB host controllers
  - Sample exploits on OpenBSD systems
- Intel® VT virtualization mechanism
- Circumventing VT isolation mechanisms
  - Experimental settings
  - Proof of concept
- Countermeasures and conclusion

# Simplified PC architecture



# I/O accesses

- IRQ to the processor.
  - Functionally unidirectional.
  - Asynchronous.
- I/O access mechanisms:
  - Memory mapped I/O (MMIO).
    - I/O registers are mapped in physical address space.
  - Programmed I/O (PIO).
    - I/O registers are mapped on a separate 16-bit bus.
- Direct Memory Access (DMA)
  - Peripherals act as masters on the PCI bus.

# Memory protection in protected/ IA-32e mode

- Hardware security mechanisms:
  - Privilege rings:
    - Most privileged: Ring 0 (kernel execution)
    - Least privileged: Ring 3 (user space code)
  - Segmentation and Paging.
- Hardware-based memory protection checks.
- Restricted assembly language instructions (HLT, LGDT, INVD).

# Programmed I/O accesses

- In protected or IA-32e modes of operation:
  - In/out instructions.
- No restriction to ring 0 code.
- Ring 3 code must be delegated I/O privileges.
  - Ring 0 can delegate such privileges using one of the two following mechanisms:
    - IOPL (full delegation).
    - I/O Bitmap in TSS (delegation of I/O privileges on selected ports only).
- Most operating systems provide system calls to software running with superuser privileges.
  - iopl (i386\_iopl, amd64\_iopl).
  - ioperm (i386\_ioperm, amd64\_ioperm).

# Outline

- Introduction
- Quick recap on PC architectures and I/O accesses
- Using chipset mechanisms from the application level to bypass security functions
  - AGP graphics aperture
  - UHCI-compliant USB host controllers
  - Sample exploits on OpenBSD systems
- Intel<sup>®</sup> VT virtualization mechanism
- Circumventing VT isolation mechanisms
  - Experimental settings
  - Proof of concept
- Countermeasures and conclusion

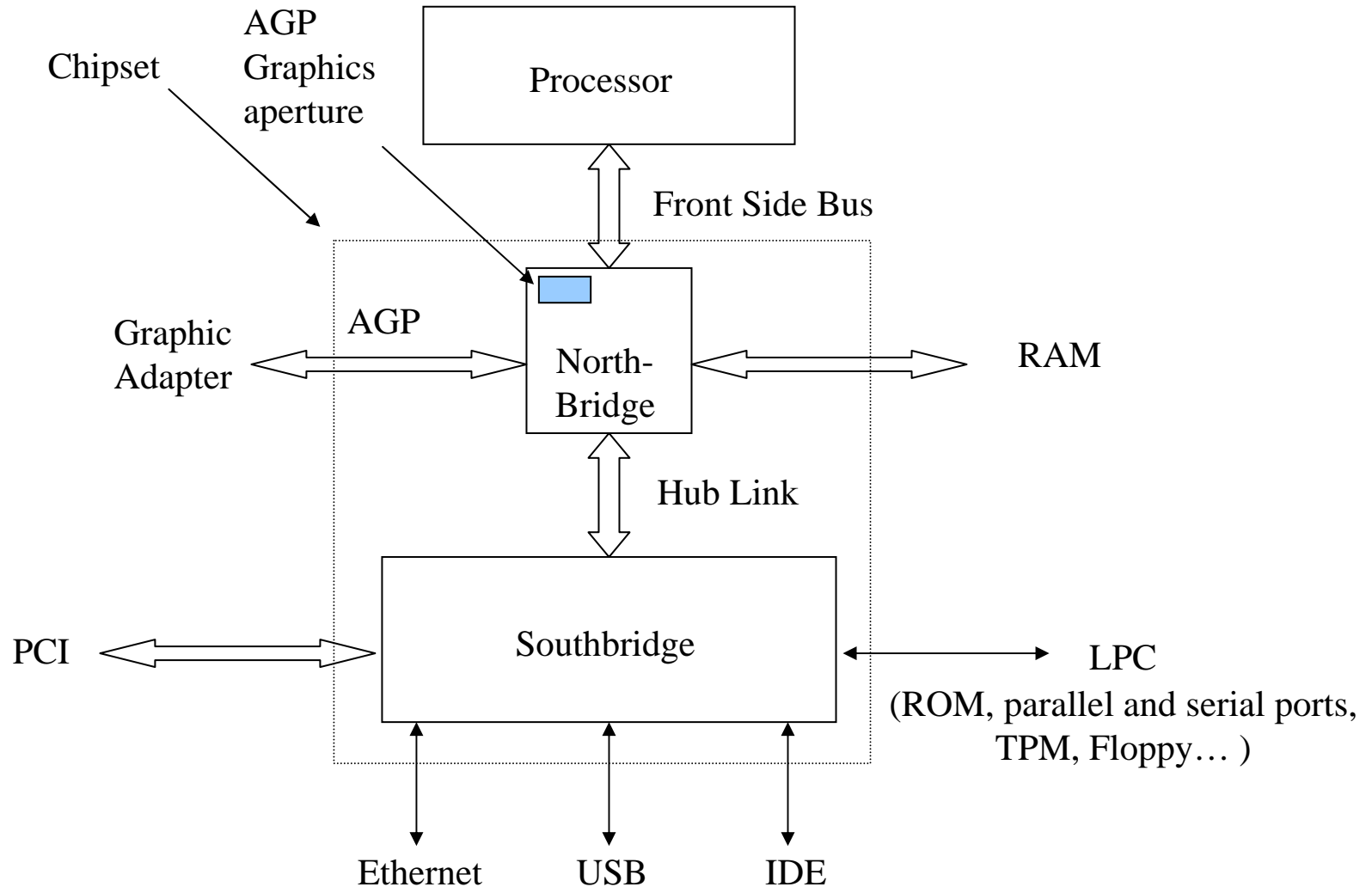
# Using hardware mechanisms to circumvent operating systems security functions

- Which hardware mechanism? Any of them!
  - Examples:
    - AGP graphics aperture (GART).
    - UHCI USB host controller.
    - System Management Mode related functionalities (see CanSecWest 2006 presentation).
- Which security function? Any privilege restriction mechanism where the privilege-reduced object can be delegated I/O privileges.
  - Examples:
    - OpenBSD securelevel.
    - VT isolation mechanism in given configurations.

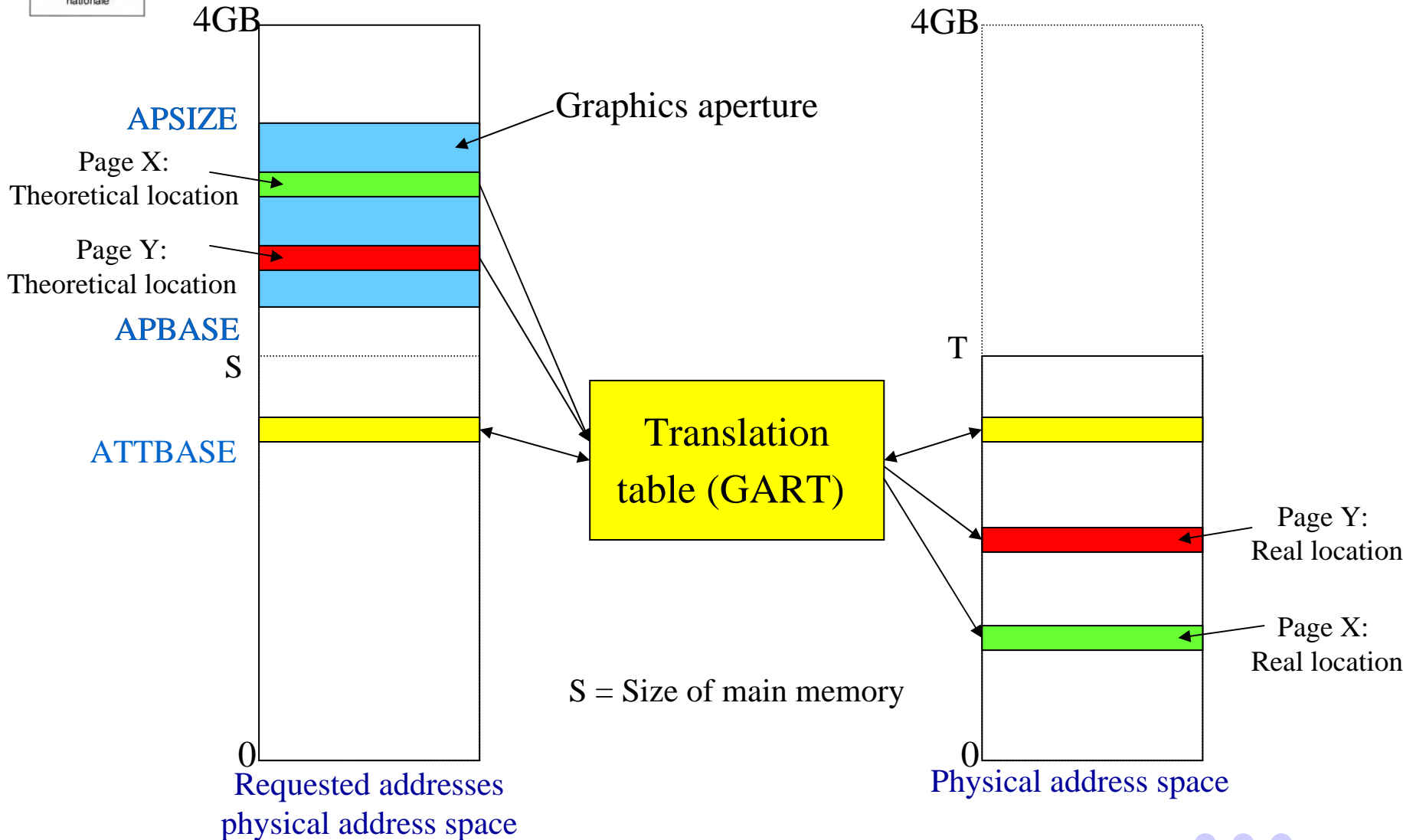
# Outline

- Introduction
- Quick recap on PC architectures and I/O accesses
- Using chipset mechanisms from the application level to bypass security functions
  - AGP graphics aperture
  - UHCI-compliant USB host controllers
  - Sample exploits on OpenBSD systems
- Intel® VT virtualization mechanism
- Circumventing VT isolation mechanisms
  - Experimental settings
  - Proof of concept
- Countermeasures and conclusion

# AGP Graphics aperture



# AGP Graphics aperture



# Graphics aperture basics

- Configuration registers are “PCI configuration registers” available in the chipset.
- They can be accessed using PIO accesses on 0xcf8 and 0xcfc PIO registers (PCI configuration mechanism).
- Map for a Intel MCH Northbridge:

**ATTBASE**

Bus 0 Dev. 0 Func 0 Off 0xb8  
outl(0xcf8, 0x800000b8)  
inl/outl(0xcfc)

**APSIZE**

Bus 0 Dev. 0 Func 0 Off 0xb4  
outl(0xcf8, 0x800000b4)  
inl/outl(0xcfc)

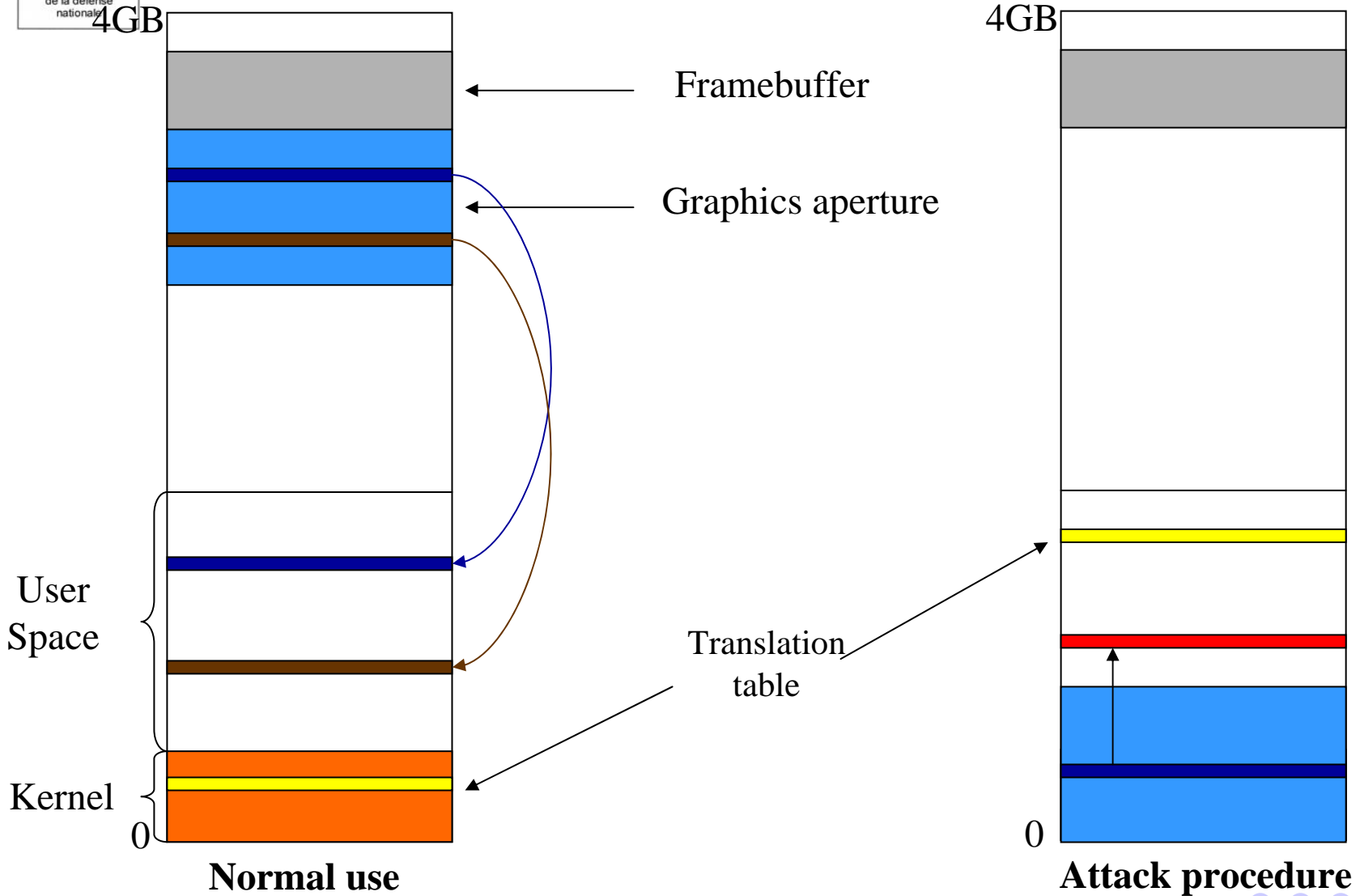
**APBASE**

Bus 0 Dev. 0 Func 0 Off 0x10  
outl(0xcf8, 0x80000010)  
inl/outl(0xcfc)

**AGPM**

Bus 0 Dev. 0 Func 0 Off 0x50  
outl(0xcf8, 0x80000050)  
inl/outl(0xcfc)

# Various uses of the mechanism



# Graphics aperture attack subtleties

- The attacker needs to determine the physical address (motherboard level) of allocated buffers:
  - One way to do it is to use read access on physical memory (through the `/dev/mem` pseudo file on Unix/Linux systems).
  - Fill the buffer with a recognizable pattern and look for the pattern in physical memory.
- Buffers may be swapped and their physical address may change during the course of the exploit.
- The graphics aperture mechanism may be already in use by another software component (display server most likely).

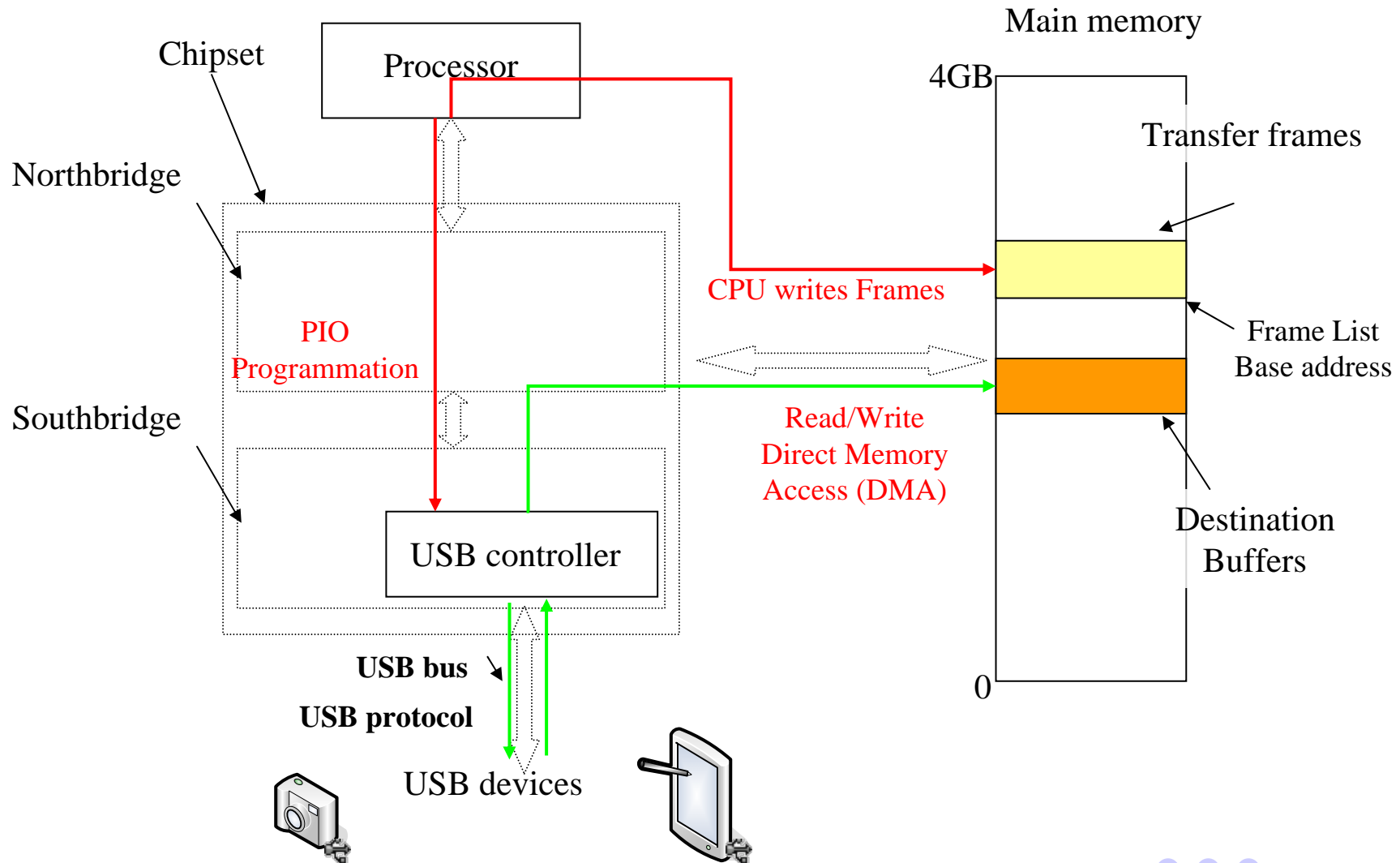
# Graphics aperture attack recap

- An attacker with:
  - PIO access privileges on 0xcf8, 0xcfc PIO registers (ATTBASE, APBASE, APSIZE, AGPM)
  - Read access to physical memory (via the /dev/mem pseudo-file on Unix/Linux systems)
- Can get to kernel privileges no matter how his other privileges are restricted.

# Outline

- Introduction
- Quick recap on PC architectures and I/O accesses
- Using chipset mechanisms from the application level to bypass security functions
  - AGP graphics aperture
  - UHCI-compliant USB host controllers
  - Sample exploits on OpenBSD systems
- Intel<sup>®</sup> VT virtualization mechanism
- Circumventing VT isolation mechanisms
  - Experimental settings
  - Proof of concept
- Countermeasures and conclusion

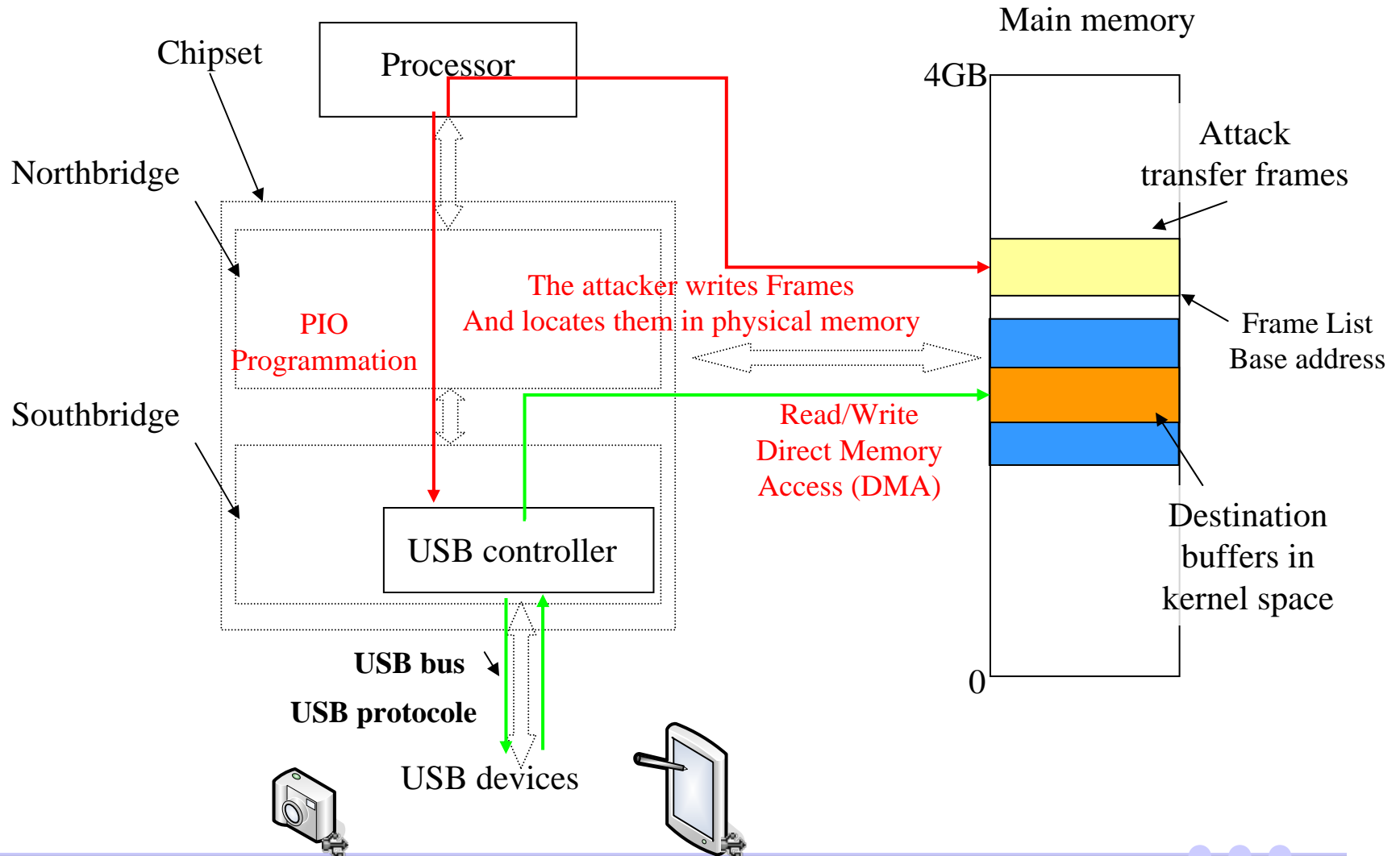
# USB Host Controller



# Useful configuration registers

- Important configuration registers:
  - FRAME\_LIST\_BASE\_ADDRESS.
  - USBCMD Command register.
  - USBSTA USB Status register.
- All of the are accessible using PIO accesses (Register in variable I/O range).
- If the attacker gets lucky (i.e. it is the case on most operating systems), command registers are already configured.

# Offensive use



# Privileges recap

- An attacker with I/O privileges on:
  - The FRAME\_LIST\_BASE\_ADDRESS register of a USB controller with any USB device plugged in.
- Read privileges on physical memory /dev/mem.
- Can get to kernel privileges.
- Read privileges are not even necessary if the attacker can guess the physical addresses of the allocated buffers.
  - Only 2 page-wide buffers need to be allocated.
  - Allocate multiple copies of those buffers and try to guess the address of one buffer of each type.
  - Works well in practice even if virtual address space is randomized...

# Outline

- Introduction
- Quick recap on PC architectures and I/O accesses
- Using chipset mechanisms from the application level to bypass security functions
  - AGP graphics aperture
  - UHCI-compliant USB host controllers
  - **Sample exploits on OpenBSD systems**
- Intel® VT virtualization mechanism
- Circumventing VT isolation mechanisms
  - Experimental settings
  - Proof of concept
- Countermeasures and conclusion

# Important question

- Alright, it seems nice on the paper but does it actual work in practice?
- Proof of concept schemes on a OpenBSD based computer with a Intel® MCH/ICH2 chipset.
- The goal is to circumvent one of OpenBSD security functions.
- Applications to virtualization systems also (later).

# Application to OpenBSD systems: OpenBSD overview

- Security-aware operating system.
- Use of the securelevel mechanism to restrict superuser privileges.
- Securelevel model:
  - In Highly Secure mode, the superuser (root) cannot get to kernel privileges (no module loading, no writing on /dev/mem, no raw accesses to disks).
  - securelevel cannot be rolled back to Insecure once in Highly Secure mode.
- We assume that the machdep.allowaperture variable is non zero (required for root to be able to use i386\_iopl or i386\_set\_ioperm system calls).
  - Default value up to 3.9. Changed in 4.0.

# AGP attack

- OpenBSD is running in Highly Secure mode.
- We assume that the attacker has already found a way to run code with superuser privileges thanks to some previous “remote to root” privilege escalation.
- In the model, the attacker cannot get to kernel privileges.
- But:
  - The attacker can use the `i386_iopl` call and thus get write access to PIO `0xcf8/0xcfc` registers.
  - Can read `/dev/mem`.
- Can use the privilege escalation attack to get to kernel privileges.

# USB-host based attack

- OpenBSD is running in Highly secure mode.
- We assume that the attacker has already found a way to run code with superuser privileges thanks to some previous “remote to root” privilege escalation.
- Cannot get to kernel privileges in the model.
- But:
  - Can use the `i386_iopl` call and thus get write access to PIO `0xcf8/0xcfc` registers.
  - Can read `/dev/mem` (optional).
- Can use the privilege escalation attack to get to kernel privileges.

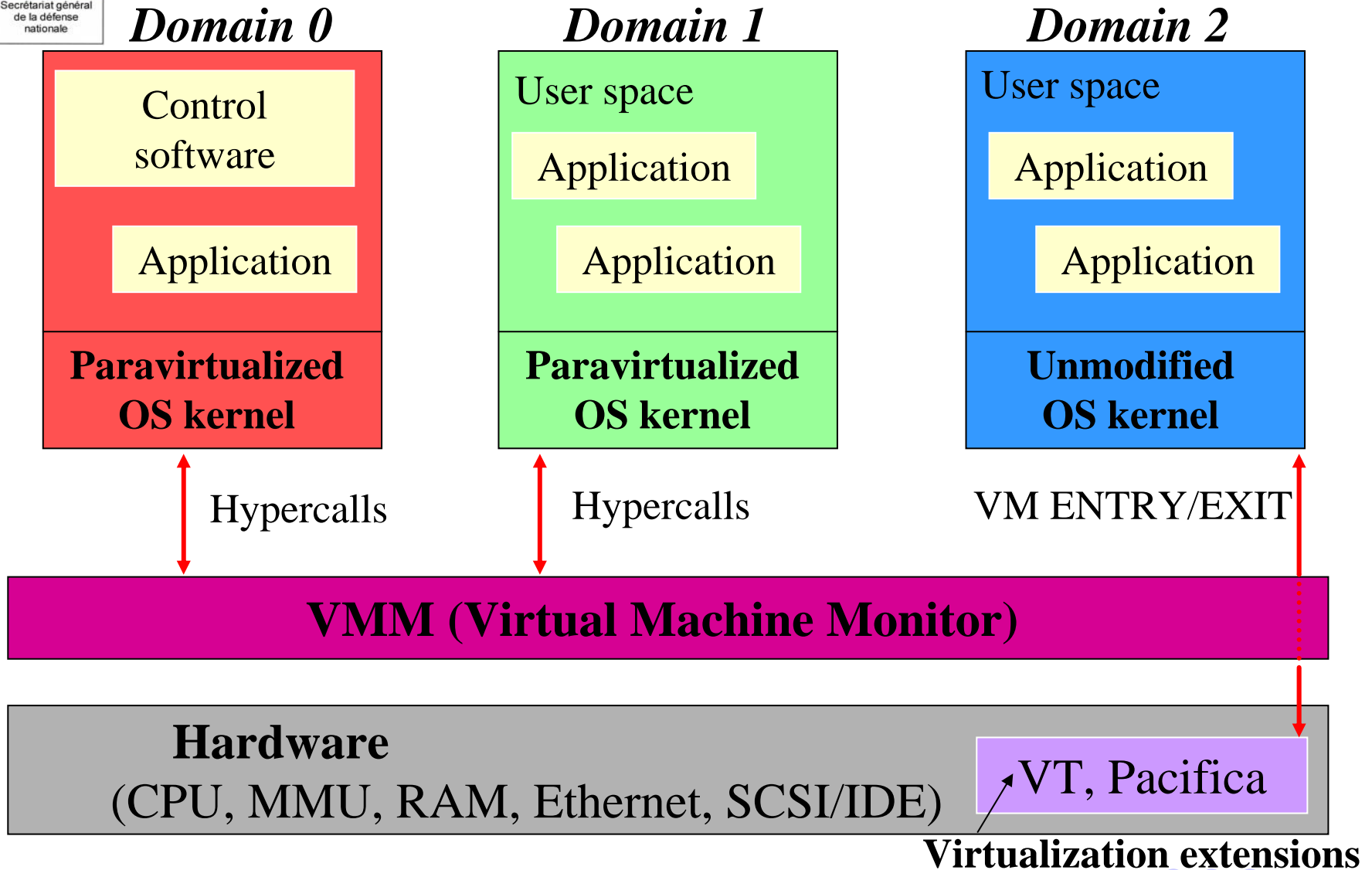
# What's been shown so far

- Processes with reduced privileges but that can be granted (one way or another) I/O accesses to some PI/O ports (USB host controllers ones for instance) can escalate to kernel privileges on the system.
- Can this be used to bypass virtualization isolation mechanisms?

# Outline

- Introduction
- Quick recap on PC architectures and I/O accesses
- Using chipset mechanisms from the application level to bypass security functions
  - AGP graphics aperture
  - UHCI-compliant USB host controllers
  - Sample exploits on OpenBSD systems
- **Intel® VT virtualization mechanism**
- Circumventing VT isolation mechanisms
  - Experimental settings
  - Proof of concept
- Countermeasures and conclusion

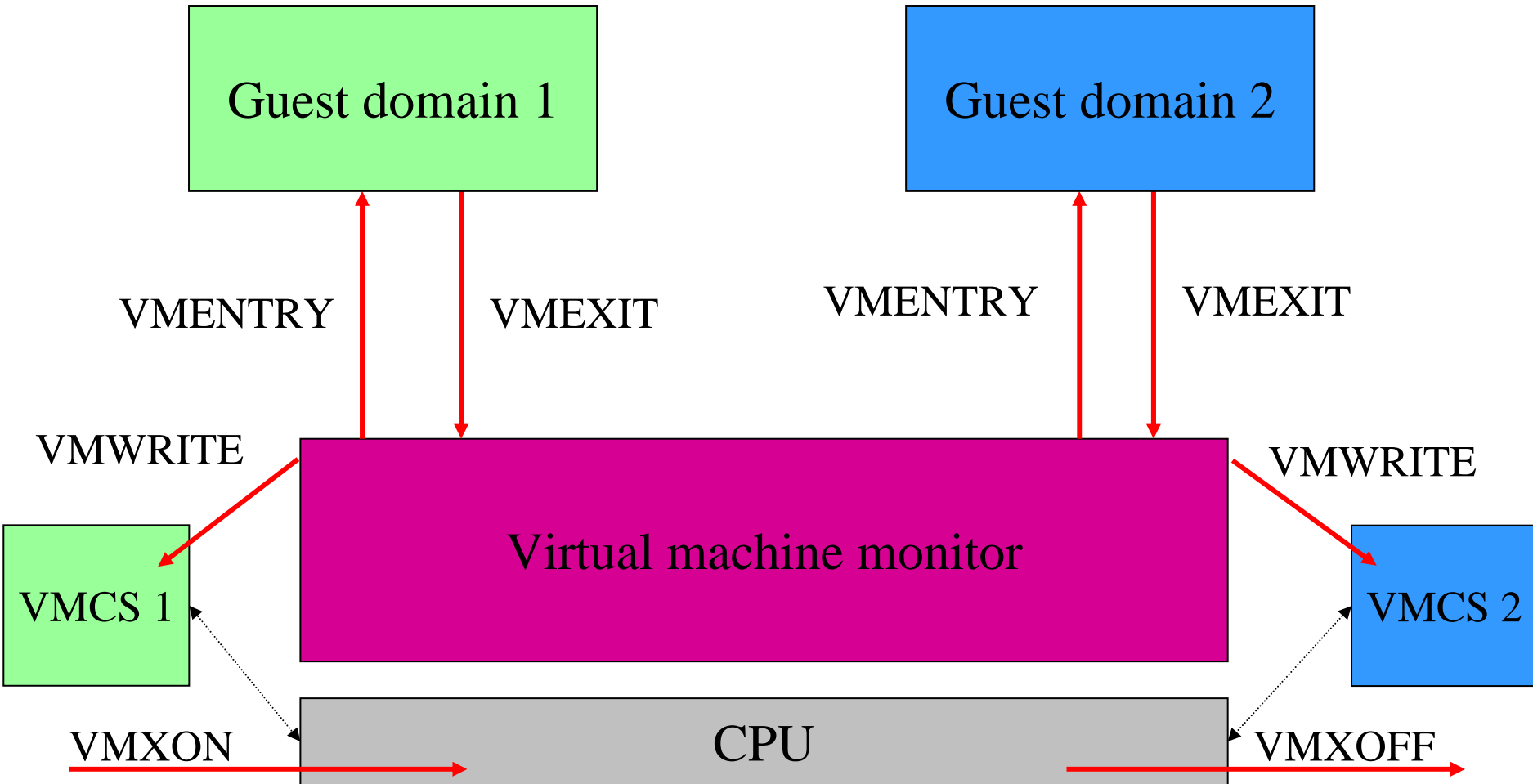
# (para)Virtualization principle



# Intel® VT technology

- Allows full virtualization of guest operating systems.
- Run several independent operating systems in parallel without modification, but virtual machine monitors must be “VT aware” .
- Confinement/Isolation model: it is impossible to get any access from one operating system to resources allocated to other operating systems.
- Parametrized through VMCS (Virtual Machine Control Structures)

# Virtual Machine Control Structures

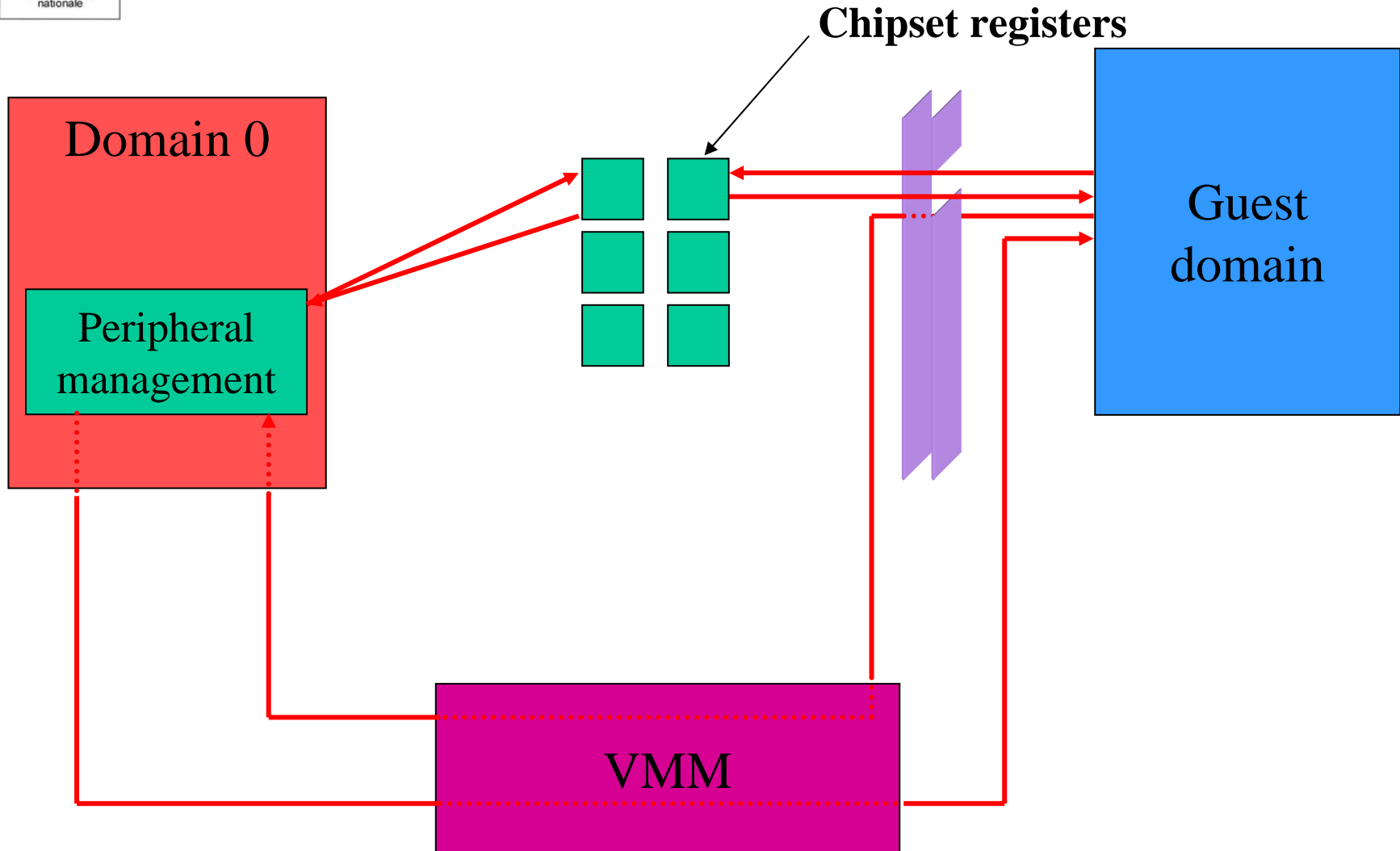


# Inside the VMCS

Bit index	Name	Description
3	TSC offsetting	RDTSC returns a value modified by the TSC offset field.
7	HLT exiting	VMEXIT triggered on HLT instruction.
9	INVLPG exiting	VMEXIT on cache flushes.
12	RDTSC exiting	VMEXIT on RDTSC instruction.
24	Unconditional I/O exiting	Each in/out instruction causes VMEXIT.
25	Use I/O bitmaps	I/O Bitmaps are use to determine I/O registers on which I/O accesses cause VMEXIT.

## Extract of the processor-based VM-execution control structure

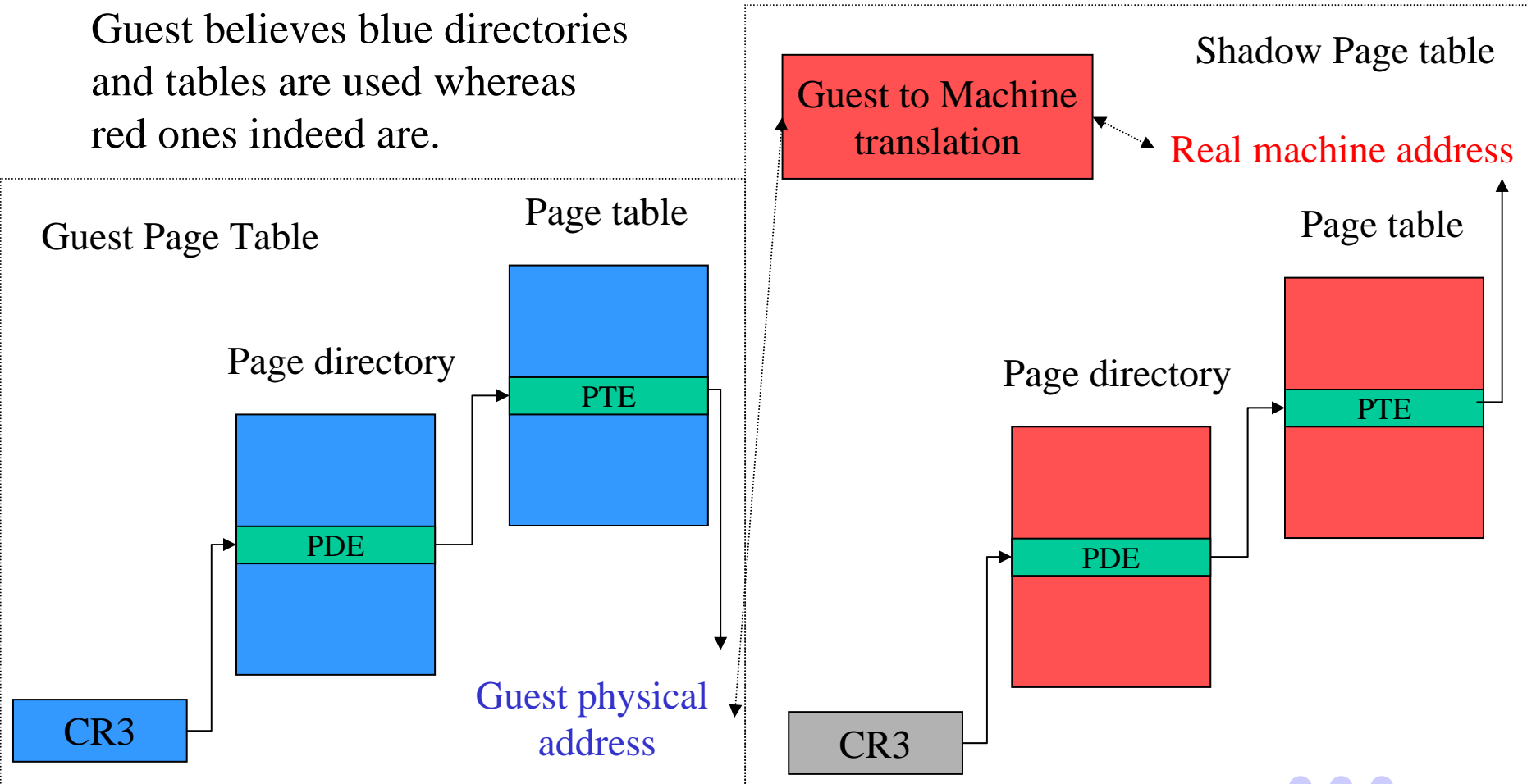
# I/O configuration



# Memory management model

- Use of shadow page tables.

Guest believes blue directories and tables are used whereas red ones indeed are.



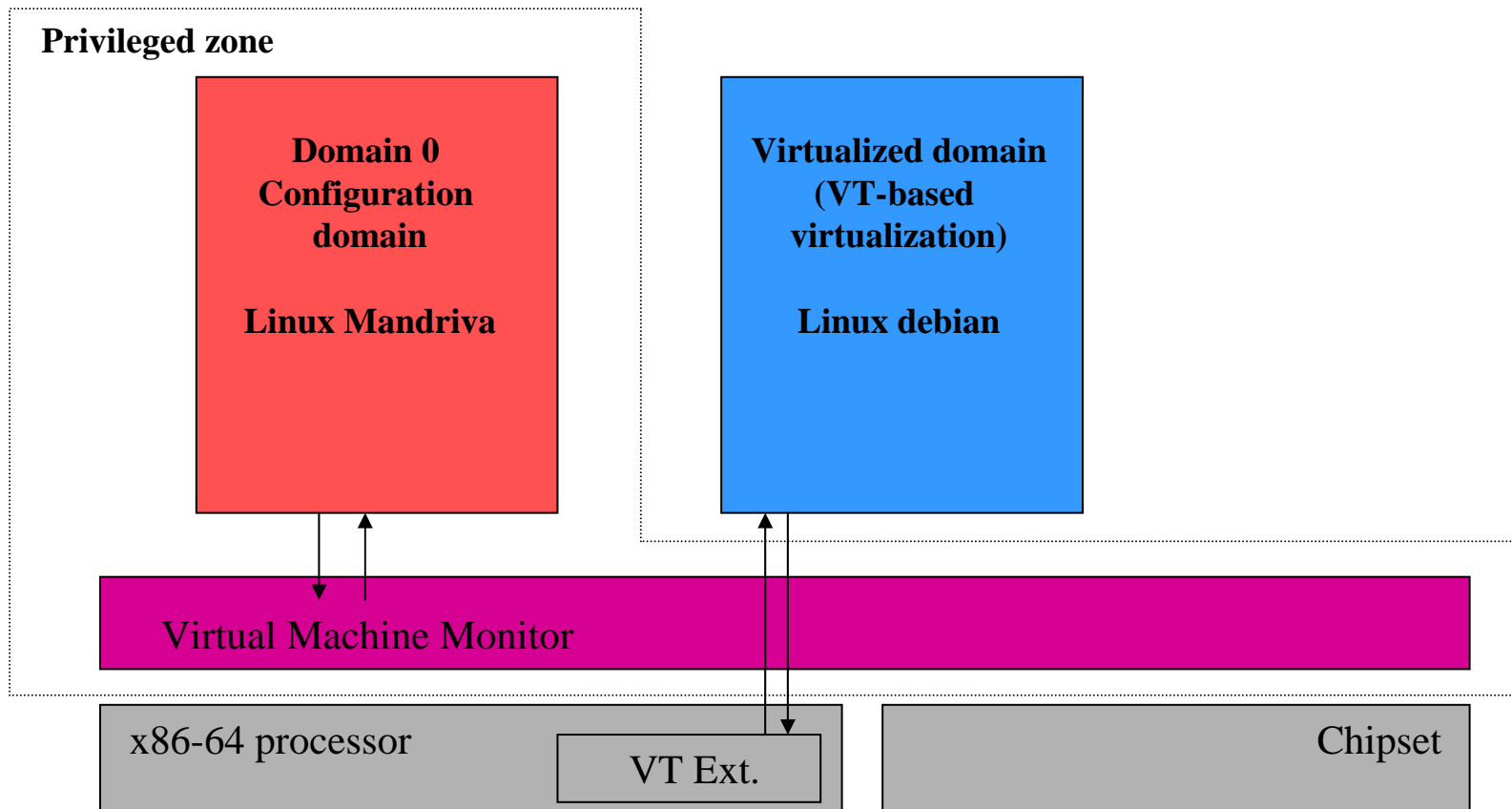
# Outline

- Introduction
- Quick recap on PC architectures and I/O accesses
- Using chipset mechanisms from the application level to bypass security functions
  - AGP graphics aperture
  - UHCI-compliant USB host controllers
  - Sample exploits on OpenBSD systems
- Intel VT virtualization mechanism
- **Circumventing VT isolation mechanisms**
  - **Experimental settings**
  - Proof of concept
- Countermeasures and conclusion

# Experimental setting

- Machine with Intel<sup>®</sup> x86-64 CPU and VT extensions activated.
- Virtual Machine Monitor emulating all I/O accesses except accesses to the Frame List Base Address Configuration register of one of the UHCI controllers of the computer:
  - Modified University of Cambridge's Xen hypervisor.
- Domain 0: Linux Mandriva
- Domain 1: Linux Debian (virtualized using VT)

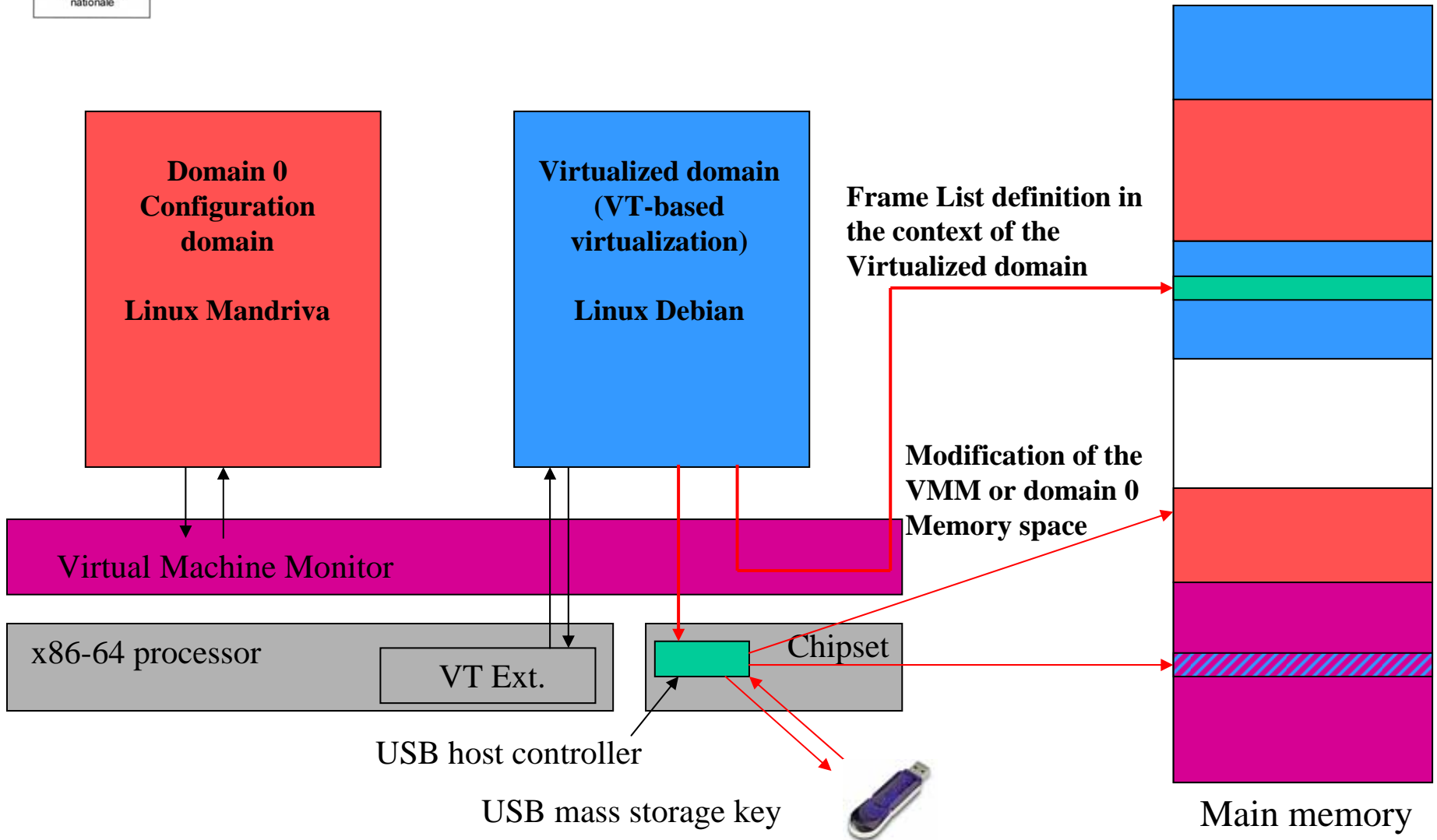
# Experimental setting



# Outline

- Introduction
- Quick recap on PC architectures and I/O accesses
- Using chipset mechanisms from the application level to bypass security functions
  - AGP graphics aperture
  - UHCI-compliant USB host controllers
  - Sample exploits on OpenBSD systems
- Intel VT virtualization mechanism
- **Circumventing VT isolation mechanisms**
  - Experimental settings
  - **Proof of concept**
- Countermeasures and conclusion

# Proof of concept attack summary



# Proof of concept attack

- It is possible:
  - From root privileges in domain 1,
  - To escalate to kernel privileges in domain 1, 0 or to Virtual Machine Monitor privileges.
- Proof of concept implementation:
  - Escalate to root privileges in domain 0.
- Theoretically possible as soon as one I/O port is not virtualized.

# Recap

- Don't get a wrong idea: neither VT nor Xen is “broken”.
- But can VMM developers be convinced that the settings or the control structures they are using for guest operating systems guaranty that there is no possibility whatsoever for one of the non privileged operating systems to get access to resources devoted to other guest operating systems or to the virtual machine monitor?
- We only show that in some configurations this is not true.
- Can we be convinced that there is a safe way to use the technology?

# Outline

- Introduction
- Quick recap on PC architectures and I/O accesses
- Using chipset mechanisms from the application level to bypass security functions
  - AGP graphics aperture
  - UHCI-compliant USB host controllers
  - Sample exploits on OpenBSD systems
- Intel VT virtualization mechanism
- Circumventing VT isolation mechanisms
  - Experimental settings
  - Proof of concept
- Countermeasures and conclusion

# Countermeasures

- For the particular schemes we describe:
  - Prevent I/O accesses to dangerous I/O ports.
    - What are the dangerous ports?
    - I/O Accesses filter
- Partial answer: use a correctly-configured I/OMMU
  - Prevents the scheme where the attacker uses UHCI USB host controllers or other schemes involving DMA.
  - Can we be sure every single attack scenario is prevented?
- VMM designers should virtualize each and every peripheral.
  - Performance issues?
  - Is that enough?

# Overall conclusion

- Showed that I/O privileges delegation is a dangerous mechanism.
  - One should not delegate I/O privileges to non privileged processes.
- Can be used as means of privilege escalation:
  - Proof of concept on OpenBSD systems.
- Can be used to bypass virtualization isolation mechanisms.
- This does not require physical access to hardware components or knowledge of hardware components implementation bugs.



Thank you for your attention

Any questions?

Contact address:

[loic.duflot@sgdn.pm.gouv.fr](mailto:loic.duflot@sgdn.pm.gouv.fr)