INF3510 Information Security
Spring 2014

Lecture 4

Computer Security

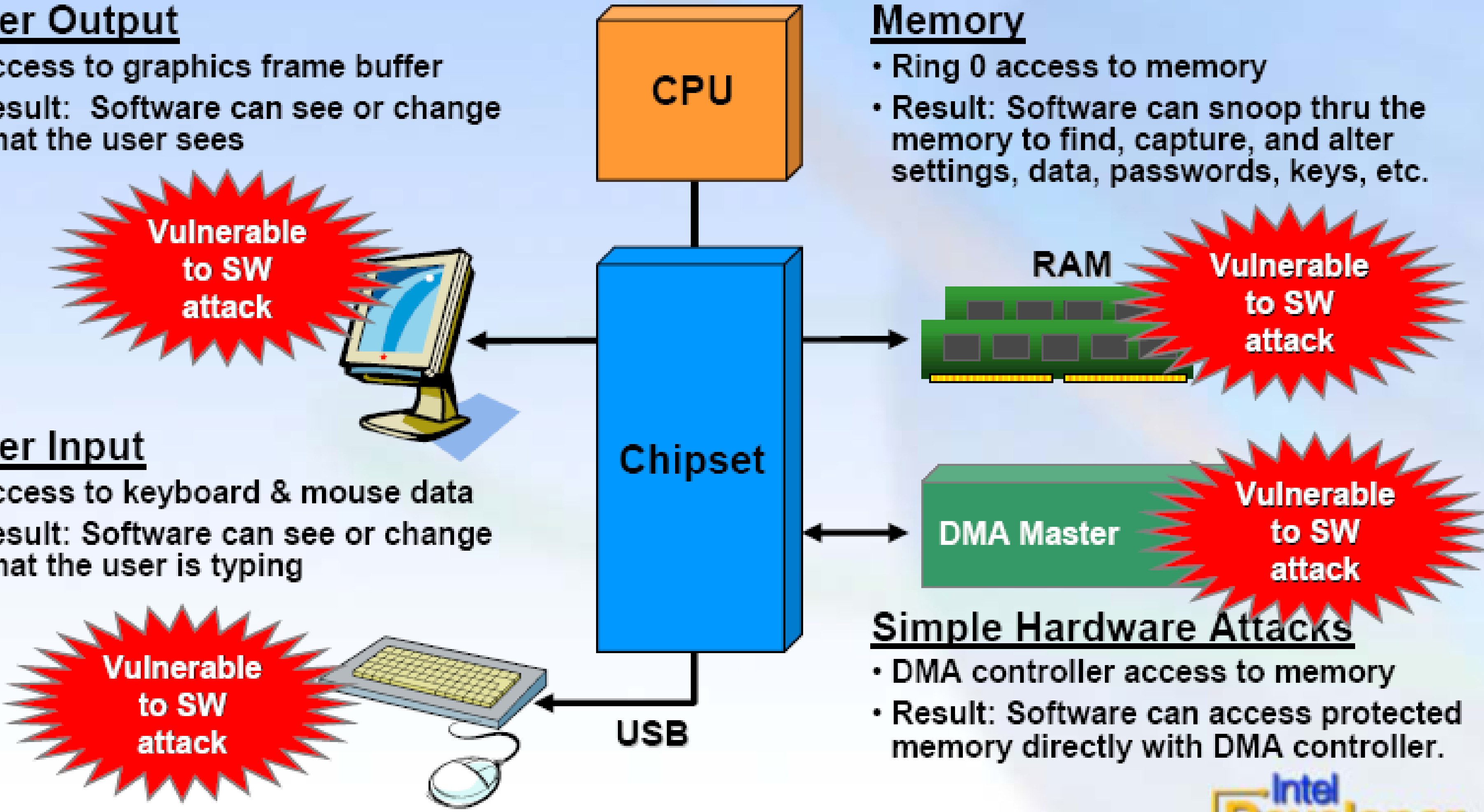
University of Oslo
Audun Jøsang

Lecture Overview

- Fundamental computer security concepts
- CPU and OS kernel security mechanisms
- Virtualization
- Memory Protection
- Trusted computing and TPM

Vulnerabilities of the PC Today

Sample of Common Vulnerabilities



User Output

- Access to graphics frame buffer
- Result: Software can see or change what the user sees

User Input

- Access to keyboard & mouse data
- Result: Software can see or change what the user is typing

Memory

- Ring 0 access to memory
- Result: Software can snoop thru the memory to find, capture, and alter settings, data, passwords, keys, etc.

Simple Hardware Attacks

- DMA controller access to memory
- Result: Software can access protected memory directly with DMA controller.

Meaningless transport defences when endpoints are insecure

"Using encryption on the Internet is the equivalent of arranging an armored car to deliver credit card information from someone living in a cardboard box to someone living on a park bench.“

(Gene Spafford)

Approaches to strengthening platform security

- Harden the operating system
  - SE (Security Enhanced) Linux, Trusted Solaris, Windows Vista/7/8
- Add security features to the CPU
  - Protection Layers, NoExecute, ASLR
- Virtualisation technology
  - Separates processes by separating virtual systems
- Trusted Computing
  - Add secure hardware to the commodity platform
  - E.g. TPM (Trusted Platform Module)
- Rely on secure hardware external to commodity platform
  - Smart cards
  - Hardware tokens

TCB – Trusted Computing Base

- The trusted computing base (TCB) of a computer system is the set of all hardware, firmware, and/or software components that are critical to its security, in the sense that bugs or vulnerabilities occurring inside the TCB might jeopardize the security properties of the entire system.
- By contrast, parts of a computer system outside the TCB must not be able to breach the security policy and may not get any more privileges than are granted to them in accordance to the security policy

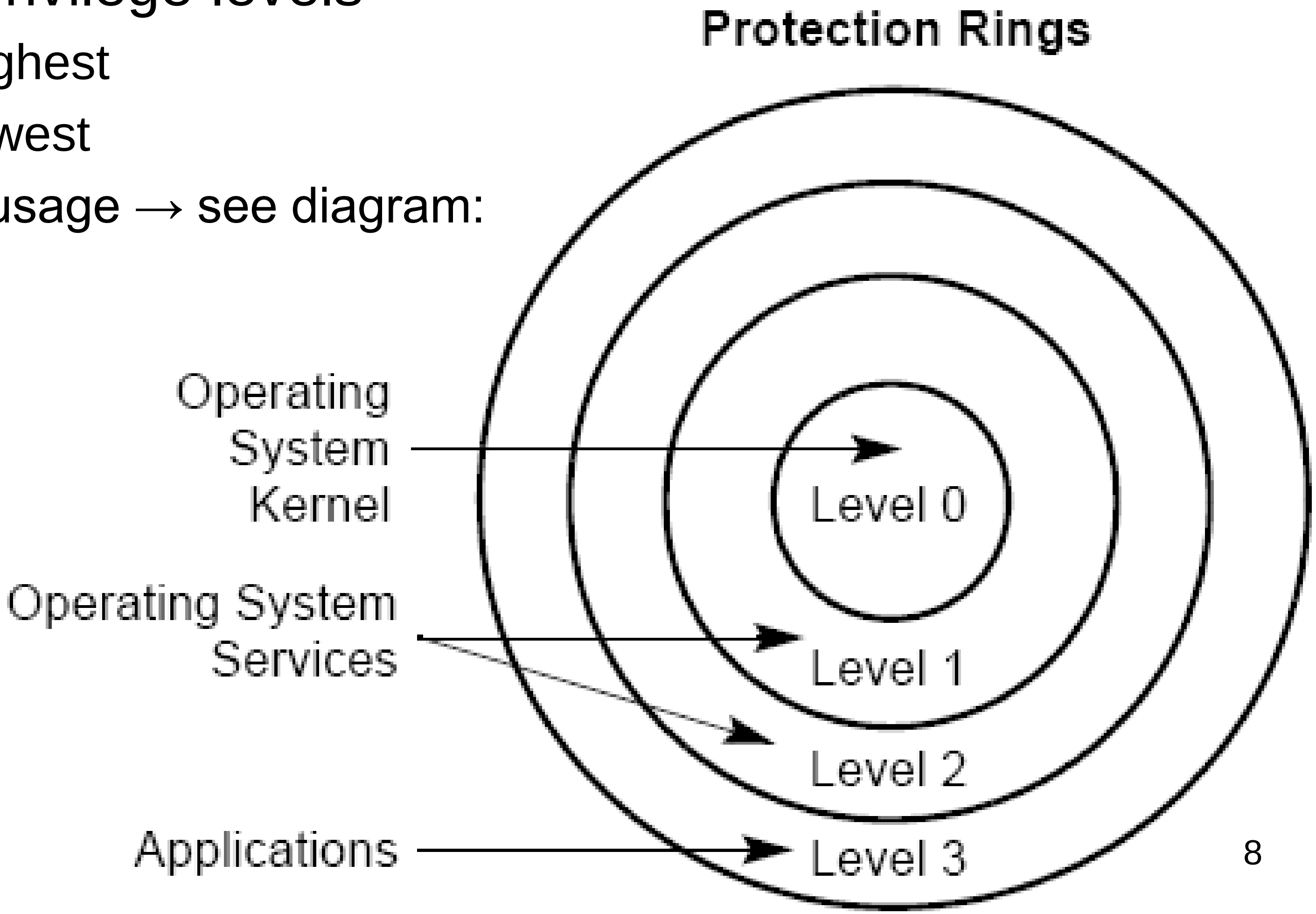
(TCSEC – Trusted Computer Evaluation Criteria, 1985).

Reference Monitor

- Reference monitor is the security model for enforcing an access control policy over subjects' (e.g., processes and users) ability to perform operations (e.g., read and write) on objects (e.g., files and sockets) on a system.
  - The reference monitor must always be invoked (complete mediation).
  - The reference monitor must be tamperproof (tamperproof).
  - The reference monitor must be small enough to be subject to analysis and tests, the completeness of which can be assured (verifiable).
- The security kernel of an OS is a low-level (close to the hardware) implememtation of a reference monitor.

OS security kernel as reference monitor

- Hierarchic security levels were introduced in X86 CPU architecture in 1985 (Intel 80386)
- 4 ordered privilege levels
  - Ring 0: highest
  - Ring 3: lowest
  - Intended usage → see diagram:



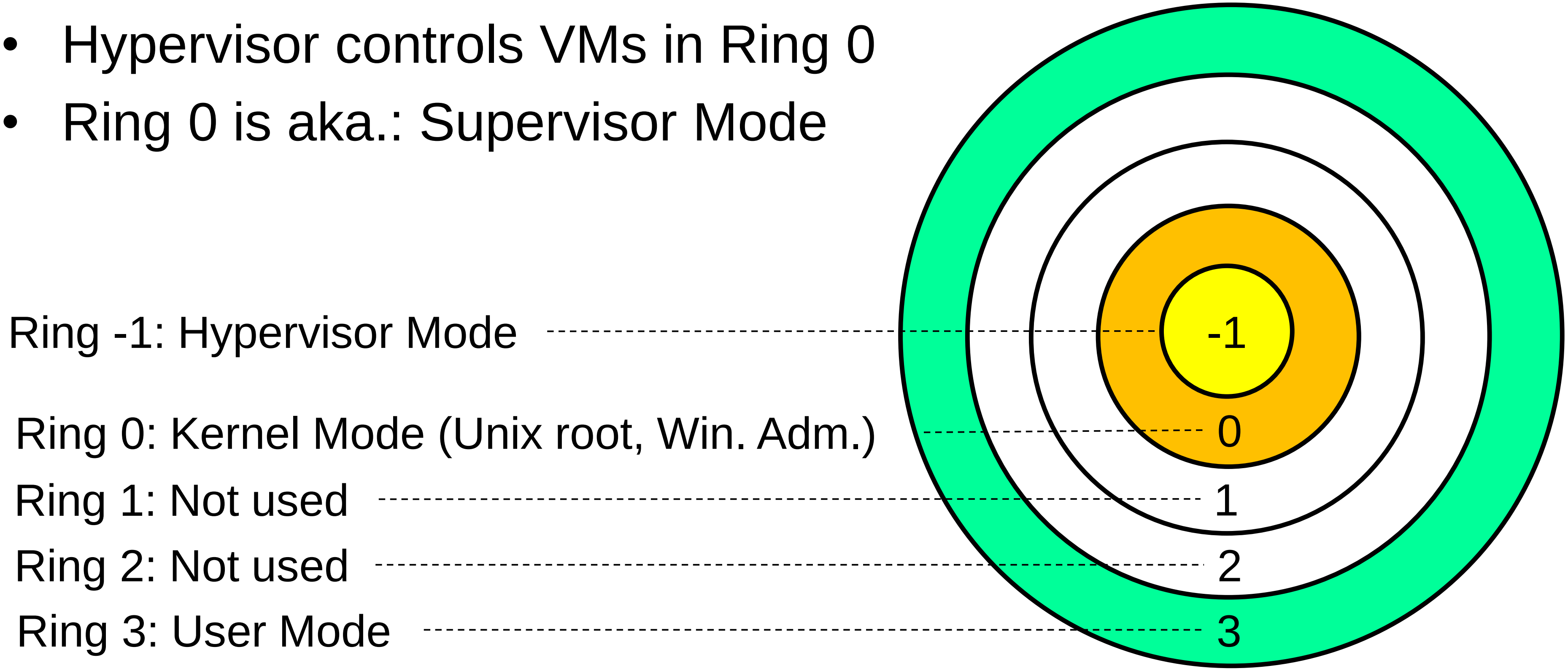
What happened to rings 1 & 2 ?

... it eventually became clear that the hierarchical protection that rings provided did not closely match the requirements of the system programmer and gave little or no improvement on the simple system of having two modes only. Rings of protection lent themselves to efficient implementation in hardware, but there was little else to be said for them. [...]. This again proved a blind alley...

Maurice Wilkes (1994)

CPU Protection Ring structure from 2006

- New Ring -1 introduced for virtualization.
- Necessary for protecting hypervisor from VMs (Virtual Machines) running in Ring 0.
- Hypervisor controls VMs in Ring 0
- Ring 0 is aka.: Supervisor Mode



Ring -1: Hypervisor Mode

Ring 0: Kernel Mode (Unix root, Win. Adm.)

Ring 1: Not used

Ring 2: Not used

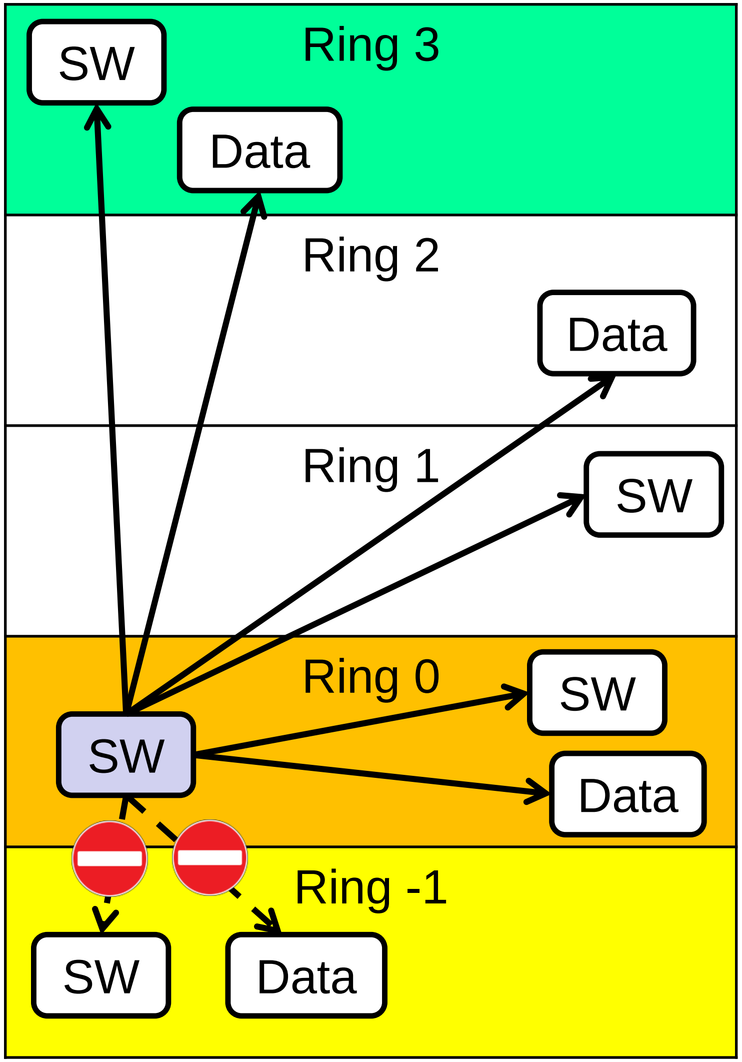
Ring 3: User Mode

Privileged Instructions

- Some of the system instructions (called “privileged instructions”) are protected from use by application programs.
- The privileged instructions control system functions (such as the loading of system registers). They can be executed only when the Privilege Level is 0 or -1 (most privileged).
- If one of these instructions is attempted when the Privilege Level is not 0 or -1, then a general-protection exception (#GP) is generated, and the program crashes.

Principle of protection ring model

- A process can access and modify any data and software at the same or less privileged level as itself.
- A process that runs in kernel mode (Ring 0) can access data and SW in Rings 0, 1, 2 and 3
  - but not in Ring -1
- The goal of attackers is to get access to kernel or hypervisor mode.
  - through exploits
  - by tricking users to install software

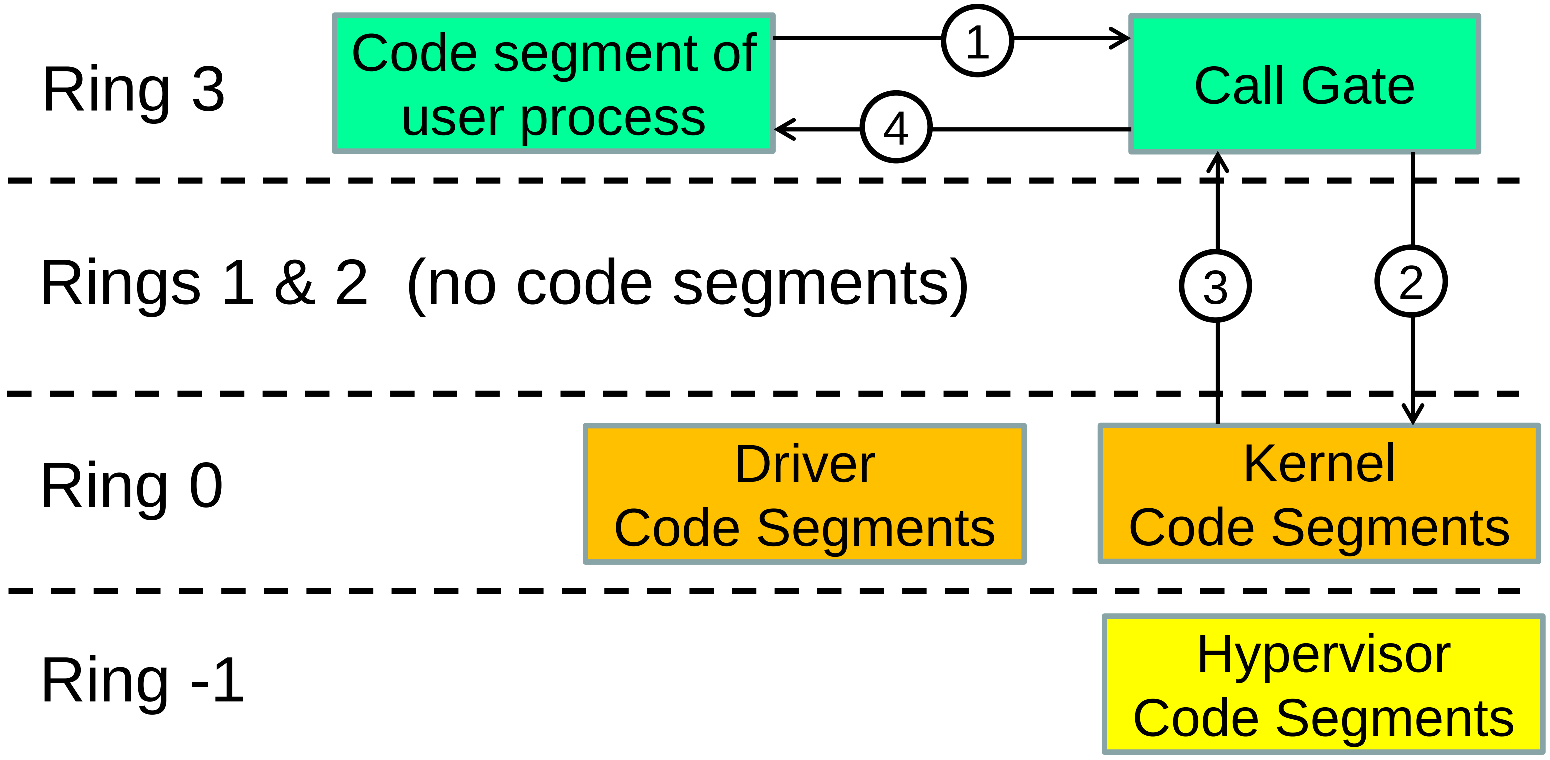


User processes access to system resources

- User processes need to access system resources (memory and drivers)
- User application processes should not access system memory directly, because they could corrupt memory.
- The CPU must restrict direct access to memory segments and other resources depending on the privilege level.

- Question 1: How can a user process execute instructions that require kernel mode, e.g. for writing to memory ?
  – Answer: The CPU must switch between privilege levels
- Question 2: How should privilege levels be switched?
  – Answer: Through Controlled invocation of code segments

Controlled Invocation of code segments



Ring 3

Code segment of user process

Call Gate

Rings 1 & 2 (no code segments)

Ring 0

Driver Code Segments

Kernel Code Segments

Ring -1

Hypervisor Code Segments

Controlled Invocation

- The user process executes code in specific code segments.
- Each code segment has an associated mode which dictates the privilege level the code executes under.
- Simply setting the mode of user process code to Kernel would give kernel-privilege to user process without any control of what the process actually does. Bad idea!
- Instead, the CPU allows the user process to call kernel code segments that only execute a predefined set of instructions in kernel mode, and then returns control back to the user-process code segment in user mode.
- We refer to this mechanism as controlled invocation.

Platform Virtualization

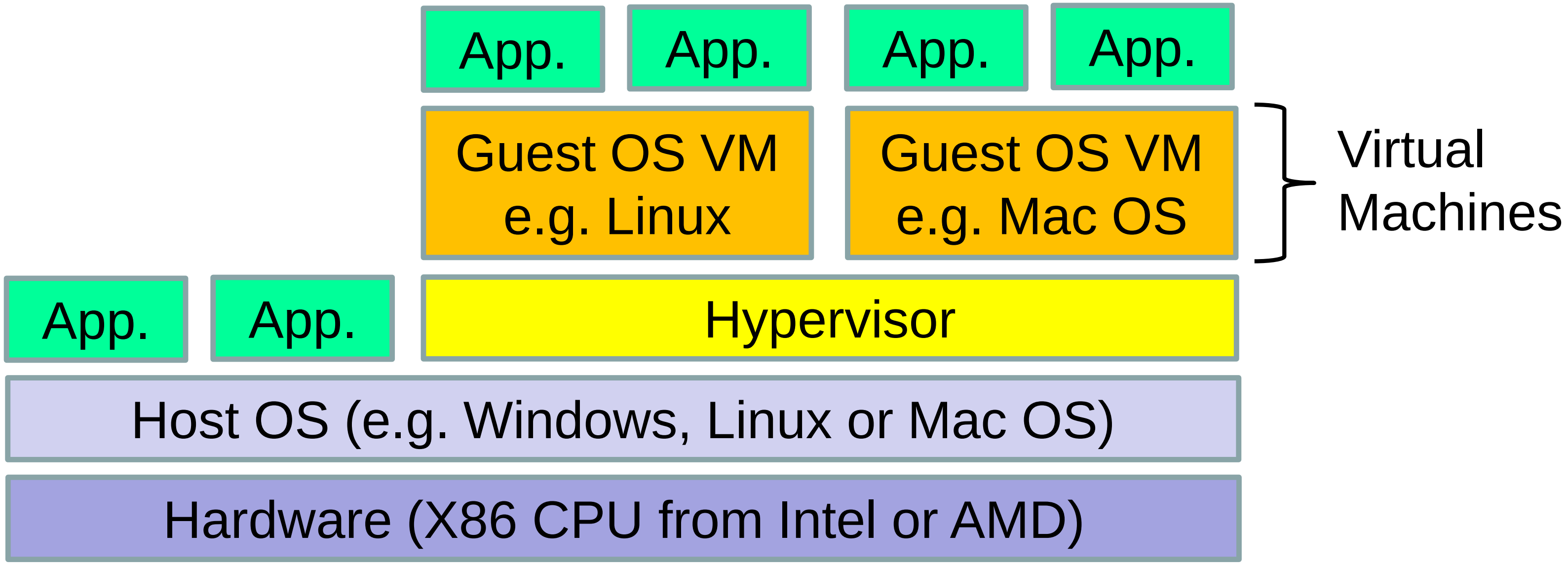
Virtual machines (VM)

- A software implementation of a machine (OS) that executes programs like a real machine (traditional OS)
- Example:
- Java Virtual Machine (JVM)
  - JVM accepts a form of computer intermediate language commonly referred to as Jave bytecode.
    - "compile once, run anywhere“
  - The JVM translates the bytecode to executable code on the fly
- Platform Virtualization
  - Simultaneous execution of multiple OSs on a single computer hardware, so each OS becomes a virtual computing platform

Platform Virtualization

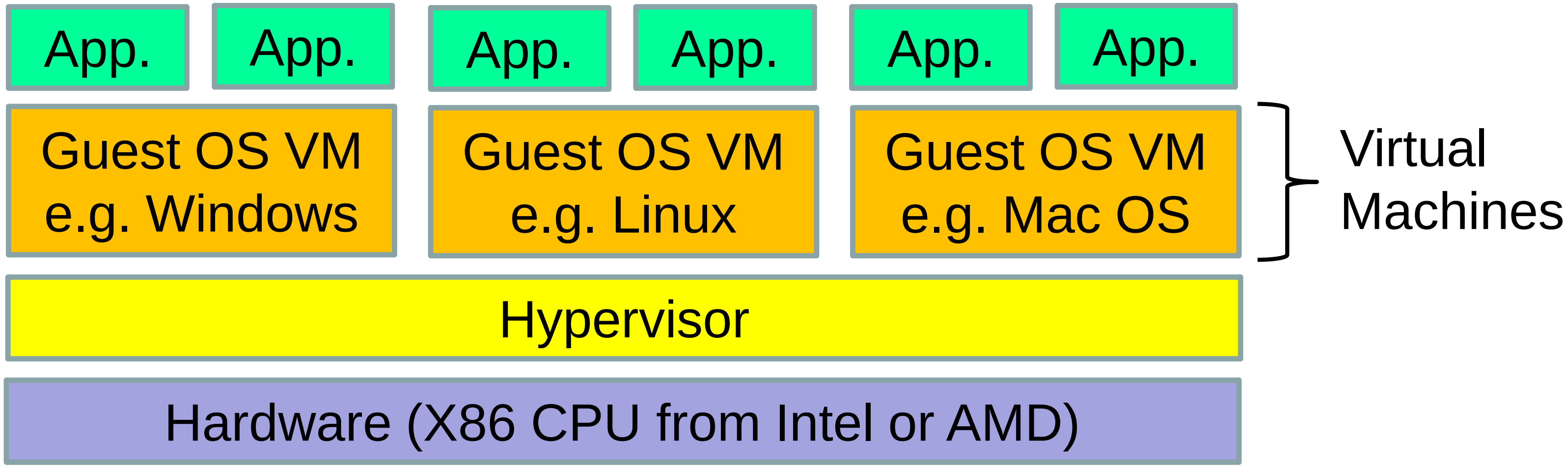
- Hypervisor (aka. VMM - Virtual Machine Monitor) is needed to manage multiple guest OSs (virtual machines) in the same hardware platform.
- Many types of hypervisors available
  - VMWare is most known Commercial product
    - Free version comes with a limitations
  - VirtualBox is a hypervisor for x86 virtualization
    - It is freely availably under GPL
    - Runs on Windows, Linux, OS X and Solaris hosts
  - Hyper-V is Microsoft’s hypervisor technology
    - Requires Windows Server

Type 2 VM Architecture (simple virtualization)



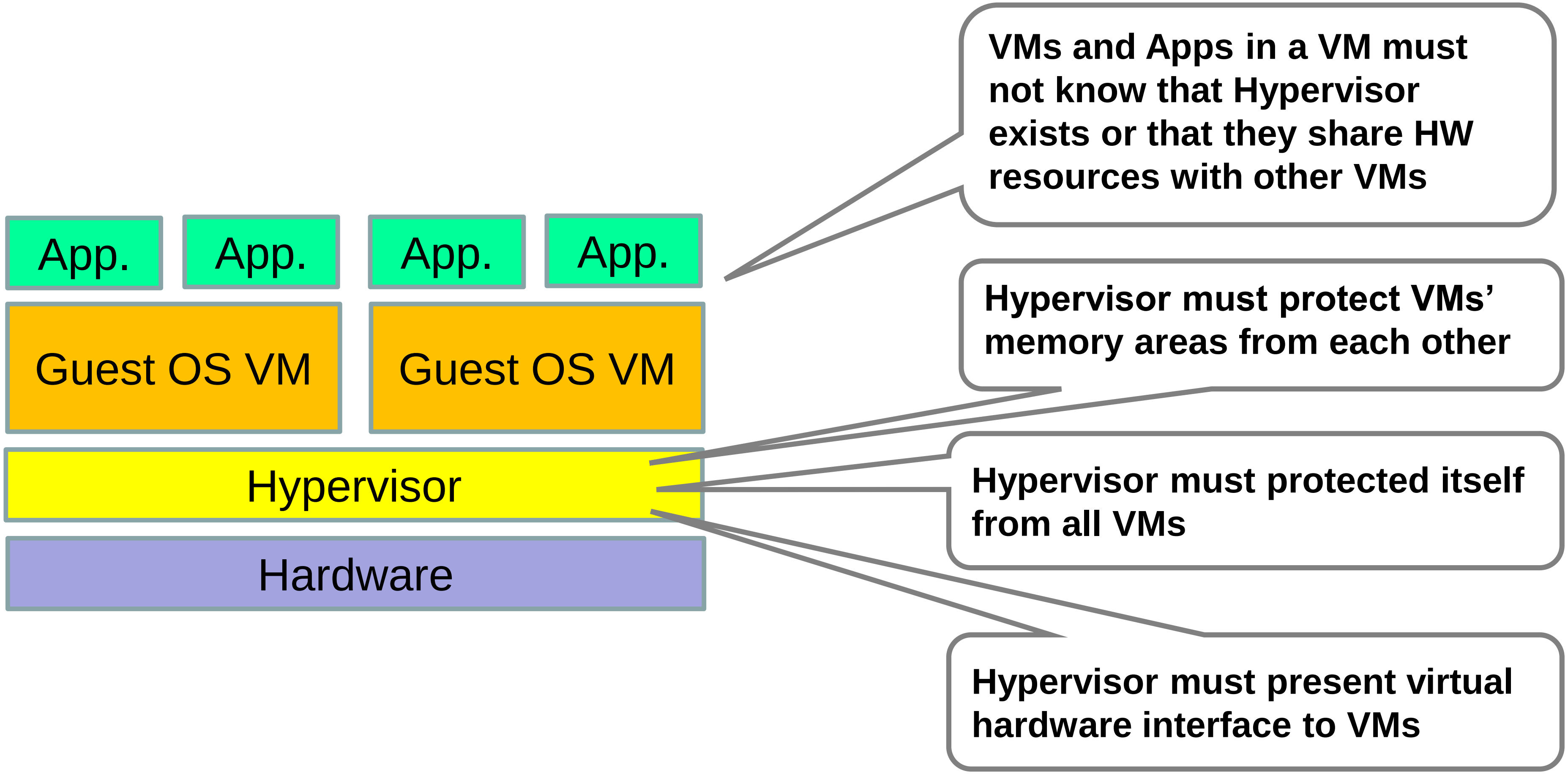
- Hypervisor runs on top of host OS
- Performance penalty, because hardware access goes through 2 OSs
- Traditionally good GUI
- Traditionally good HW support, because host OS drivers available

Type 1 VM Architecture (advanced virtualization)



- No host OS
- Hypervisor runs directly on hardware
- High performance
- Traditionally limited GUI, but is improved in modern versions
- HW support can be an issue

Challenges of Running VMs



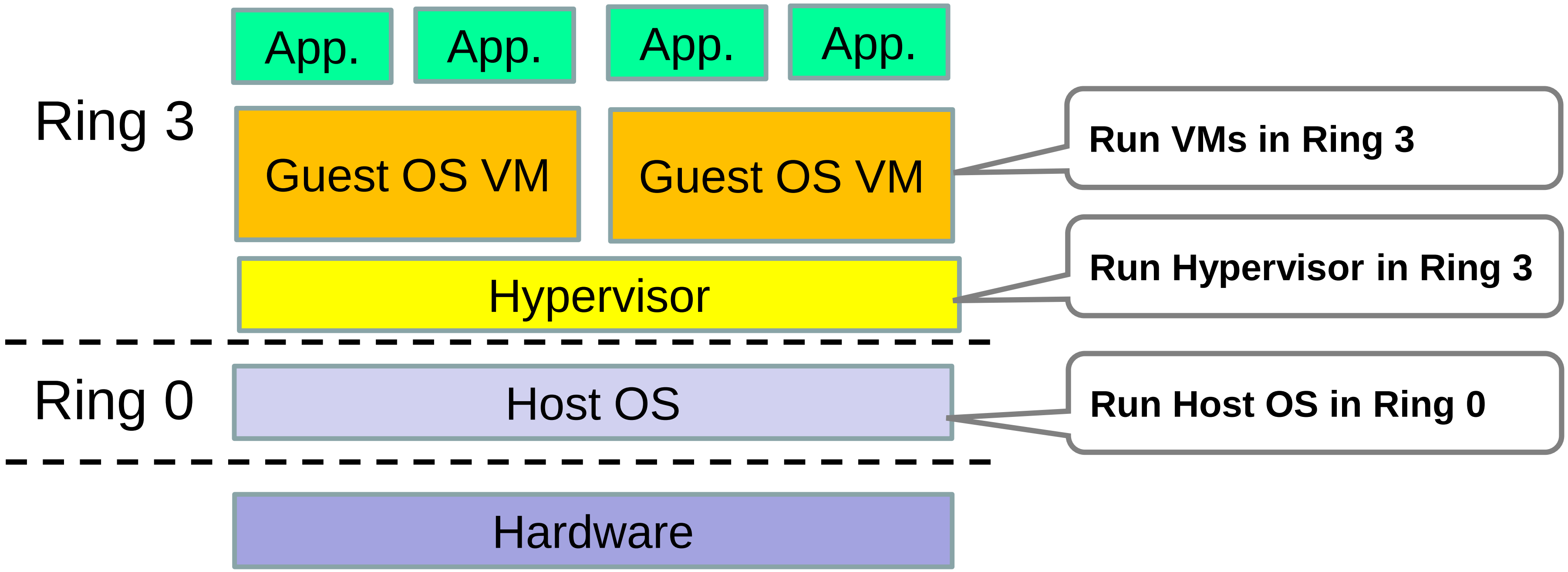
VMs and Apps in a VM must not know that Hypervisor exists or that they share HW resources with other VMs

Hypervisor must protect VMs' memory areas from each other

Hypervisor must protected itself from all VMs

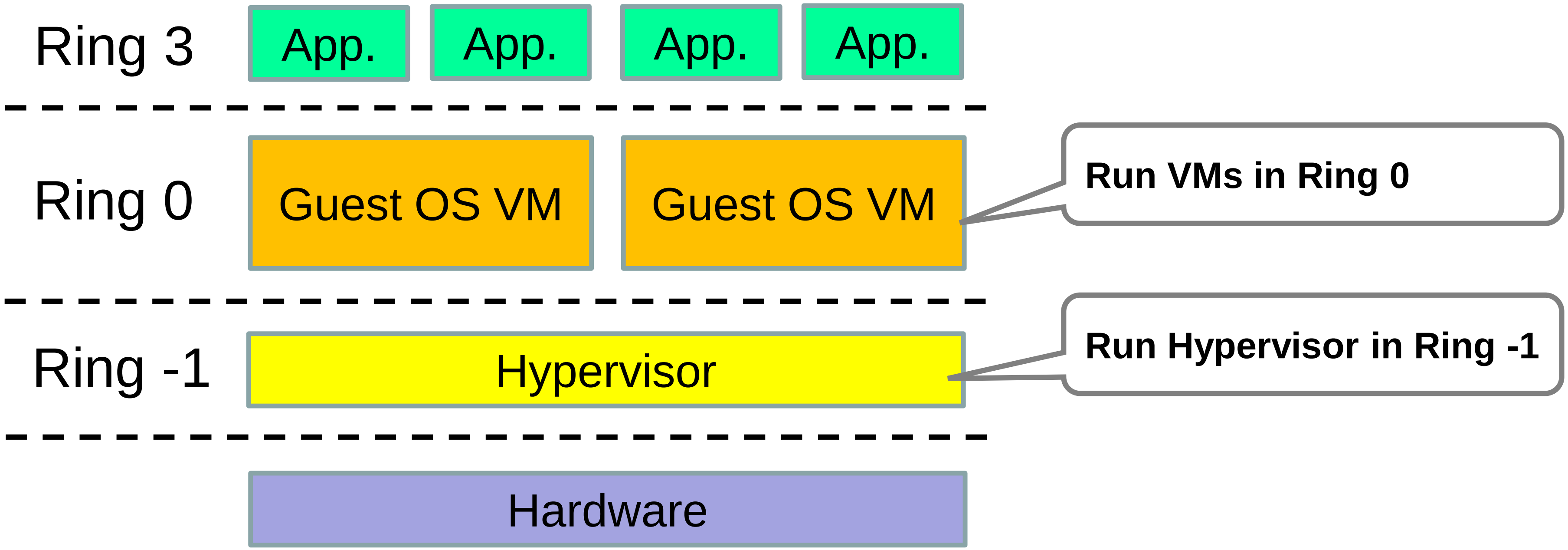
Hypervisor must present virtual hardware interface to VMs

Type 2 VM Architecture Ring Allocation



- Guest OS VMs run in Ring 3.
- Guest OS VMs call privileged instructions that are forbidden in Ring 3.
- Forbidden instructions cause exceptions that are handled by interrupt/exception handler to be executed.
- Slow performance !

Type 1 VM Architecture Ring Allocation



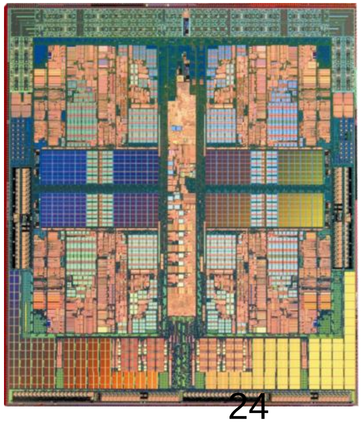
- Guest OS VMs are less privileged than the hypervisor.
- Hypervisor is well protected from the VMs.
- Good security !

Hardware support for virtualization

- Modern Intel and AMD X86 CPUs support virtualization
  - Intel-VT (Intel Virtualization Technology)
  - AMD-V (AMD Virtualization)
- Must be enabled in BIOS
  - Can be enabled and disabled
  - Computers with single OS typically have virtualization disabled
- Access to data- and code segments for hypervisor can be restricted to processes running in hypervisor mode
- Some instructions are reserved for hypervisor mode

Intel Core i7 CPU

AMD Phenom CPU

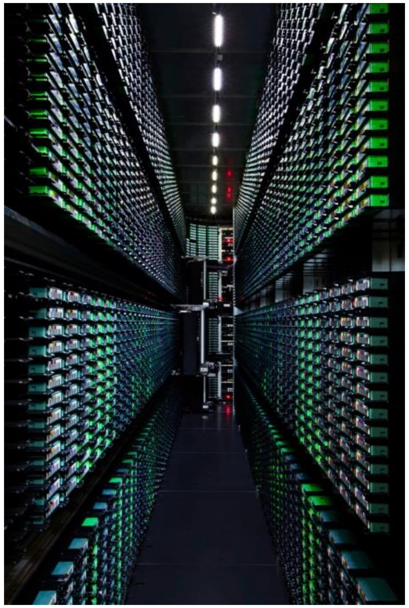


Why use platform virtualization

- Efficient use of hardware and resources
  - Improved management and resource utilization
  - Saves energy
- Improved security
  - Malware can only infect the VM
  - Safe testing and analysis of malware
  - Isolates VMs from each other
- Distributed applications bundled with OS
  - Allows optimal combination of OS and application
  - Ideal for cloud services
- Powerful debugging
  - Snapshot of the current state of the OS
  - Step through program and OS execution
  - Reset system state

Hypervisor examples of use

- Cloud providers run large server parks
  - Each customer gets its own VM
  - Many customers share the same hardware
  - Migrated VMs between servers to increase/reduce capacity



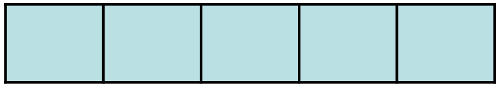
Google data center

- Testing and software analysis
  - Potentially damaging experiments can be executed in isolated environment
  - Take a snapshot of the current state of the OS
  - Use this later on to reset the system to that state
  - Malware Analysis

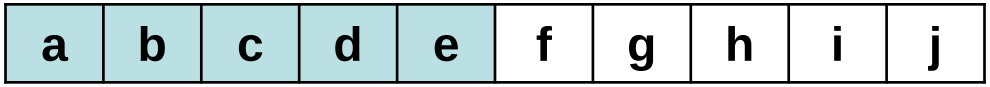
Memory Protection

Buffer overflow

- A program tries to store more data in a buffer than it was intended to hold.
- Example:
  - Assume a 5 bytes buffer to store a variable in memory:



  - Write10 bytes to buffer, then 5 extra bytes get overwritten



a	b	c	d	e	f	g	h	i	j

  - If the overwritten part contained a return pointer or software, it is possible for the attacker to execute his own instructions.
- Many attacks are based on buffer overflow techniques

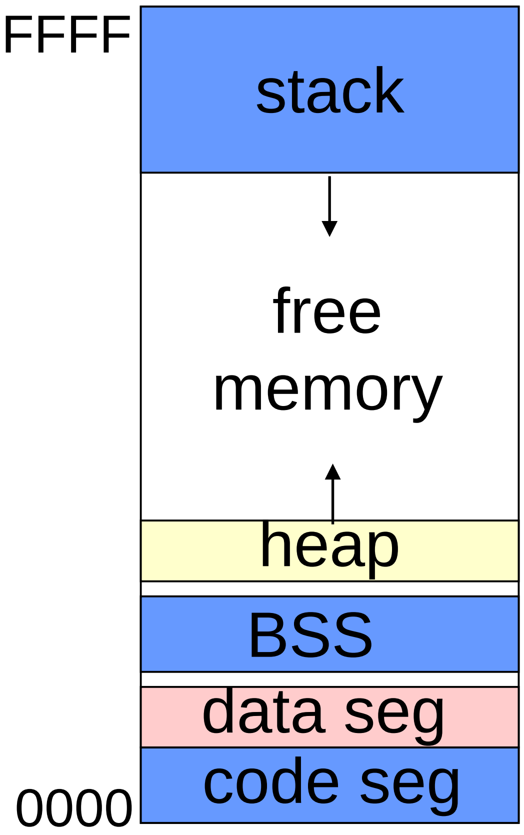
Buffer Overflow

- Buffer overflow is when written data size > buffer size
  - Results in neighbouring buffers being overwritten
- Unintentional buffer overflow crashes software, and results in unreliability software.
- Intentional buffer overflow is when an attacker modifies specific data in memory to execute malware
- Attackers target return addresses (specify the next piece of code to be executed) and security settings.
- In languages like C or C++ the programmer allocates and de-allocates memory.
- Type-safe languages like Java guarantee that memory management is ‘error-free’.

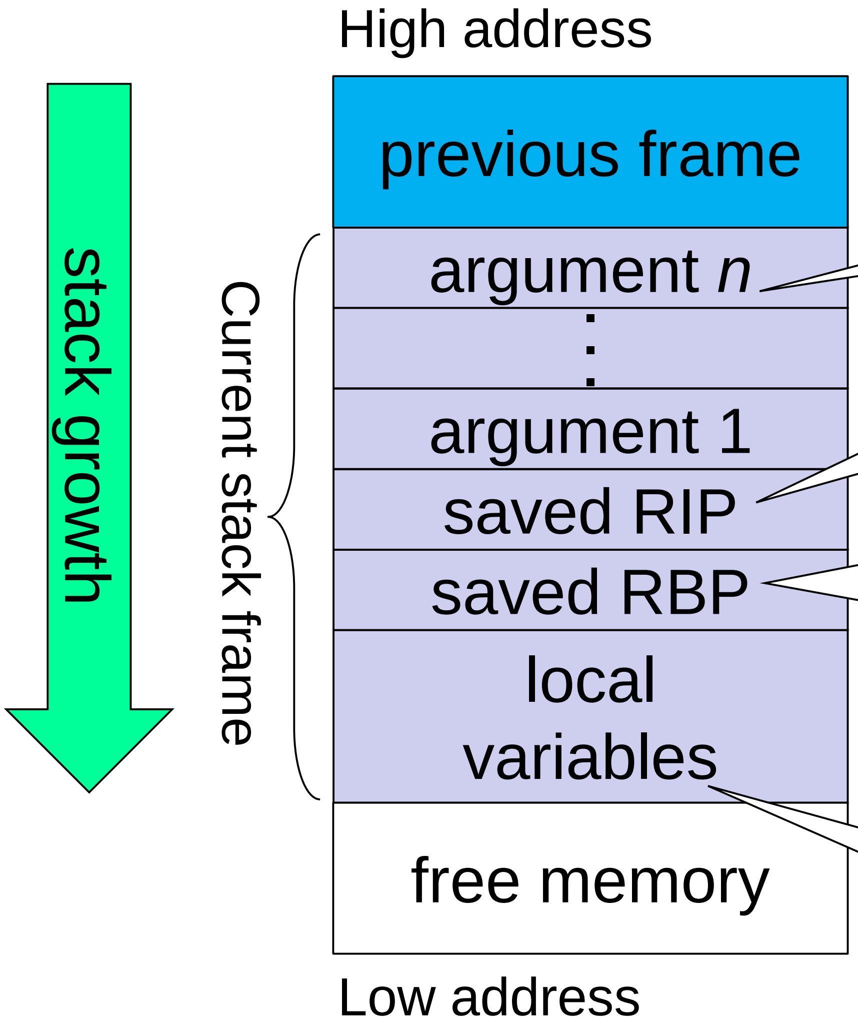
Memory corruption and buffer overflow

- The stack contains memory buffers that hold return address, local variables and function arguments. It is possible to decide in advance where a particular buffer will be placed on the stack.

- Heap: dynamically allocated memory; more difficult but not impossible to decide in advance where a particular buffer will be placed on the heap.
- BSS: Block Segment of Static Variables



Stack Frame – Layout



Inputs to function

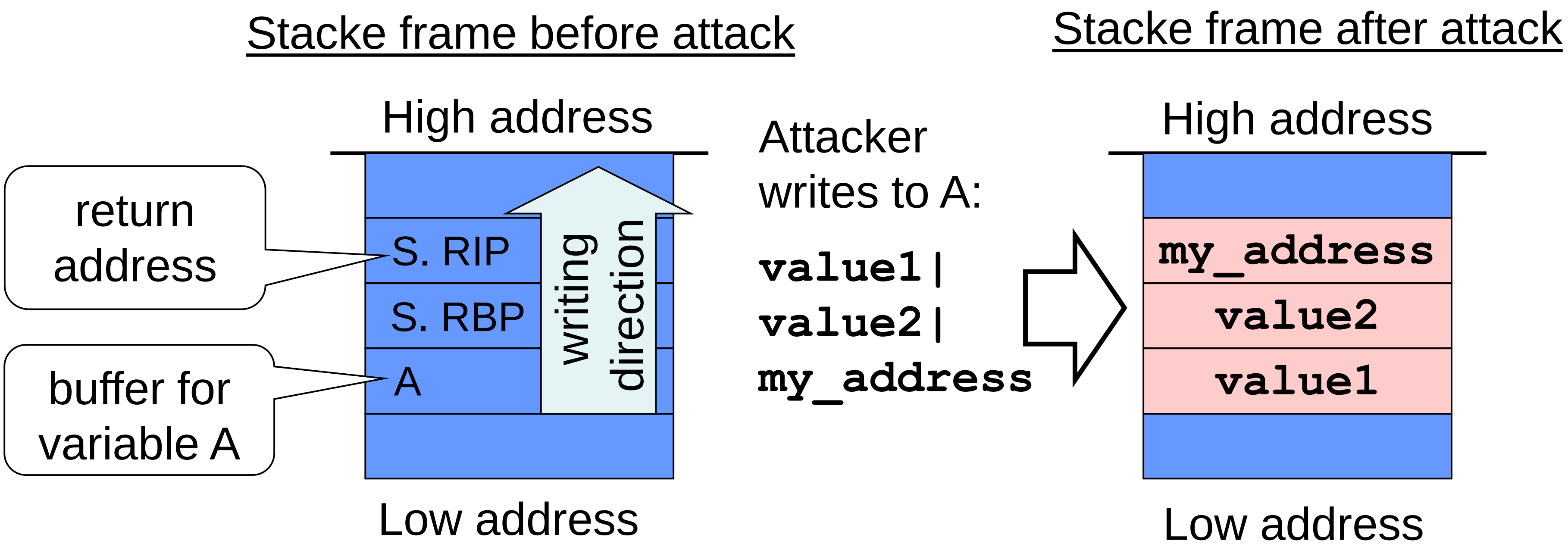
Saved RIP (Instruction Pointer) (return address) (EIP in 32 CPU)

Saved RBP (Base Pointer) (reference point for relative addressing, a.k.a. frame pointer) (EBP in 32 bit CPU)

Local variables stored in memory buffers (ranges) of specific sizes.

Stack-based Overflows

- Find a buffer on the runtime stack that can overflow.
- Overwrite the return address with the start address of the code you want to execute.
- The code can also be injected by overflowing buffers.
- You can now execute your own code.



Stacke frame before attack

Stacke frame after attack

Attacker writes to A:

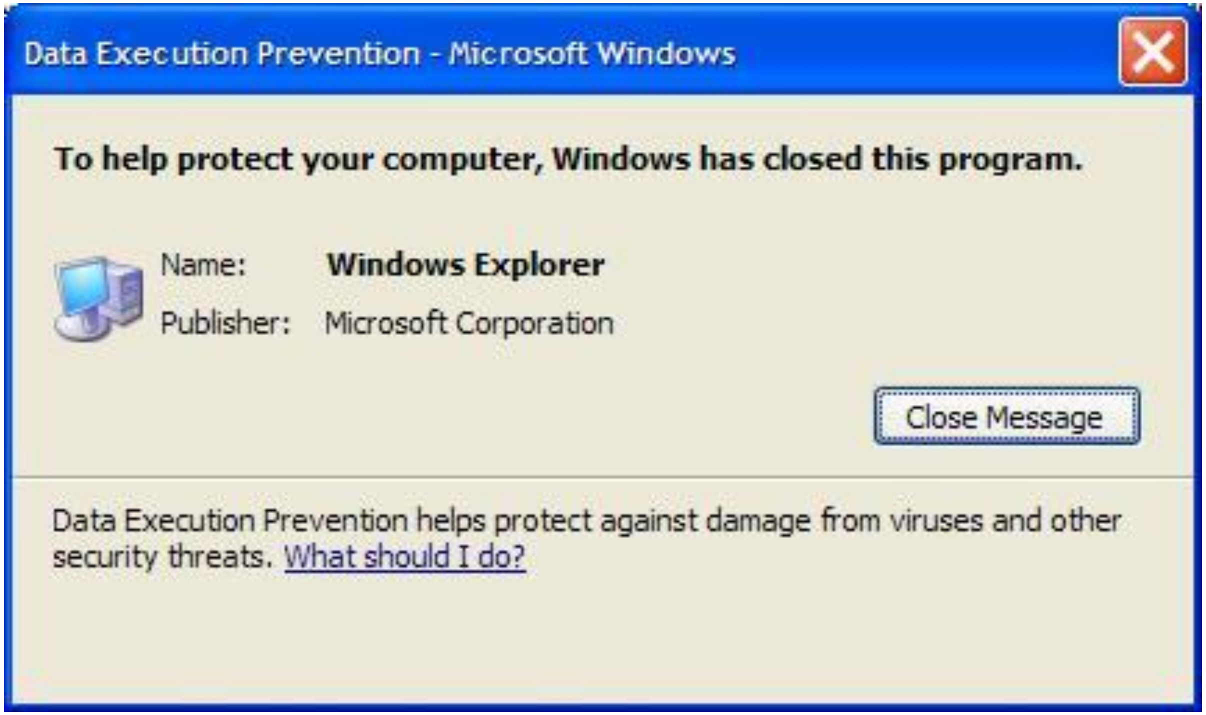
```
value1|
value2|
my_address
```

Defences against memory corruption

- Hardware mechanisms
  - NX (No eXecute) bit/flag in stack memory
    - Injected attacker code will not execute on stack
- OS / compiler mechanisms
  - Stack cookies: detects corruption at runtime
  - ASLR (Address Space Layout Randomization)
    - Makes it difficult to locate functions in memory
- Programming language
  - Type safe languages like Java and C#
- Programming rules
  - Avoid vulnerable functions like
    - strcpy (use strncpy instead)
    - gets (use fgets instead)

NX stops

- An attacker supplying his own shell code
  - storing it in writable memory …
  - and executing it, results in



Data Execution Prevention - Microsoft Windows

To help protect your computer, Windows has closed this program.

Name: **Windows Explorer**

Publisher: Microsoft Corporation

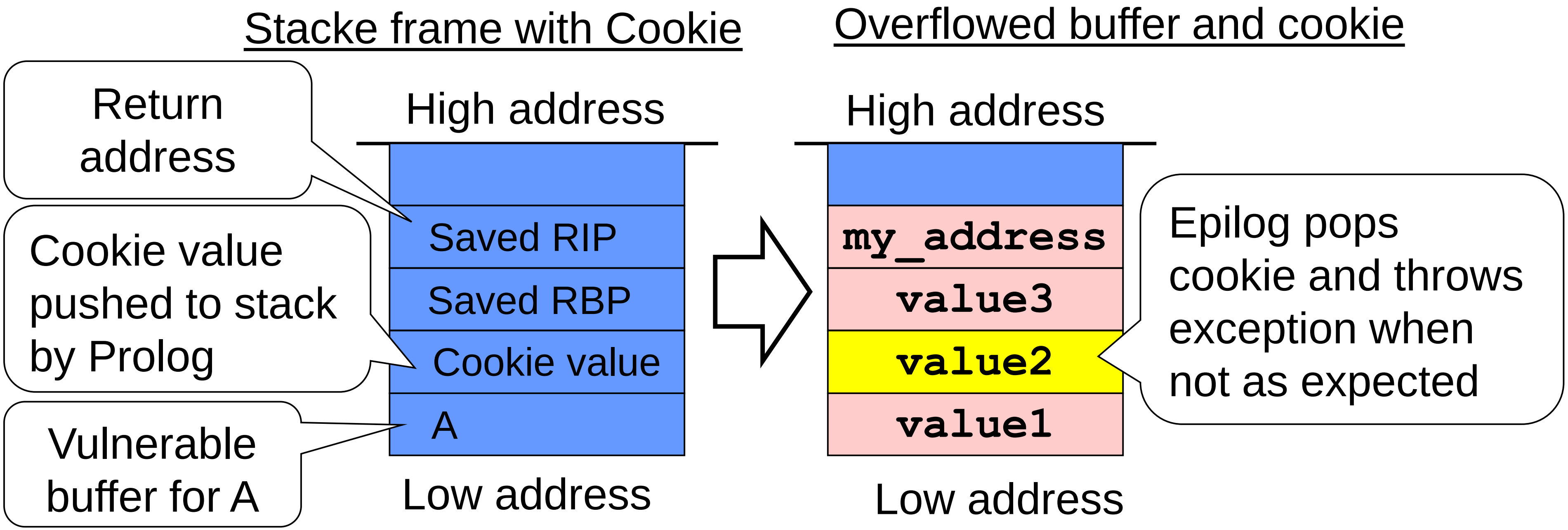
Data Execution Prevention helps protect against damage from viruses and other security threats. What should I do?

Stack cookies (canaries)

- Stack cookies are integrity values to detect overflow of saved RIP
- Compiler adds code to start (Prolog) and to end (Epilog) of every function.
- Prolog and epilog code is generated at compile-time.
- During run-time Prolog pushes cookie value to stack frame after saved RIP.
- Attacker can not guess cookie value.
- Buffer overflow destroys cookie.
- Epilog verifies correct cookie, or detects when cookie is destroyed.

Stack Cookie explained

- A cookie integrity check value is computed by OS
- Prolog pushes cookie onto stack frame after saved RIP/RBP.
- Before returning from function, epilog checks cookie.
- Exception if cookie value is different from original value.
- Disadvantage of stack cookies: Computation overhead.



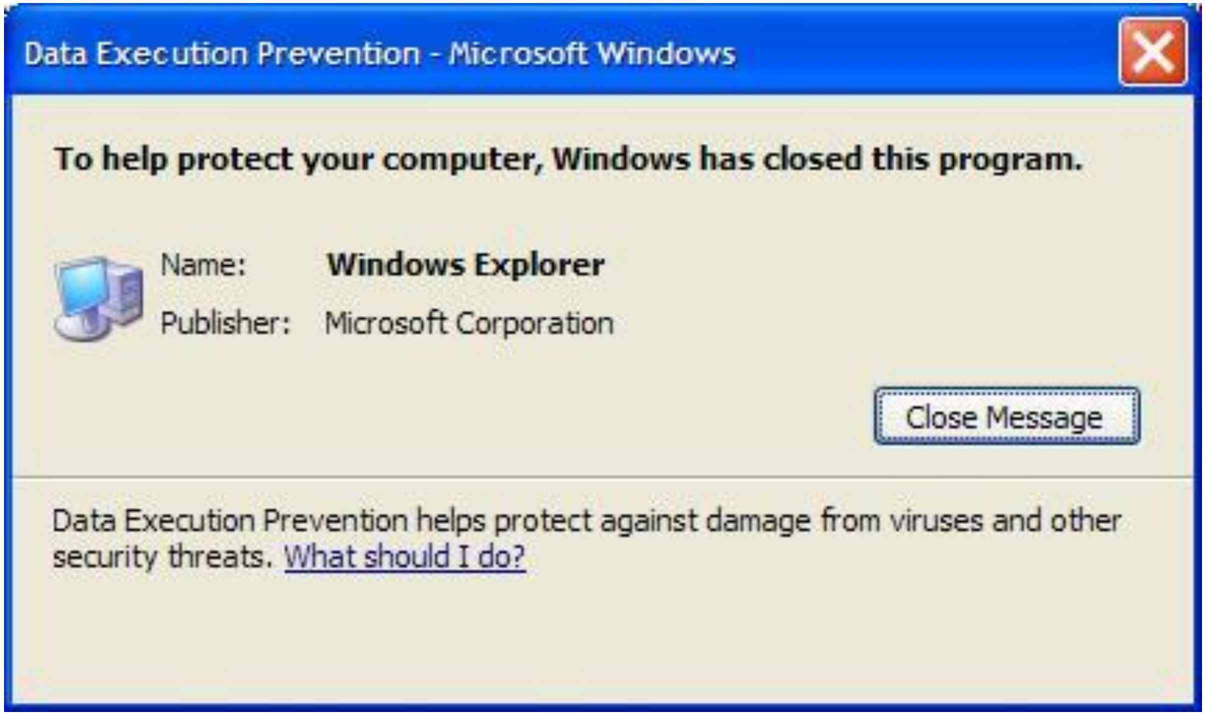
Non executable memory

- Run time mechanism
- Utilizes CPU support for NX/XD for marking memory pages RWX
- Hypotesis: A writable page should not need to be executable
  - Is this always true?

NX stops

STOP

- An attacker supplying his own shell code
  - – storing it in writable memory …
  - – and executing it, results in



Data Execution Prevention - Microsoft Windows

To help protect your computer, Windows has closed this program.

Name: **Windows Explorer**

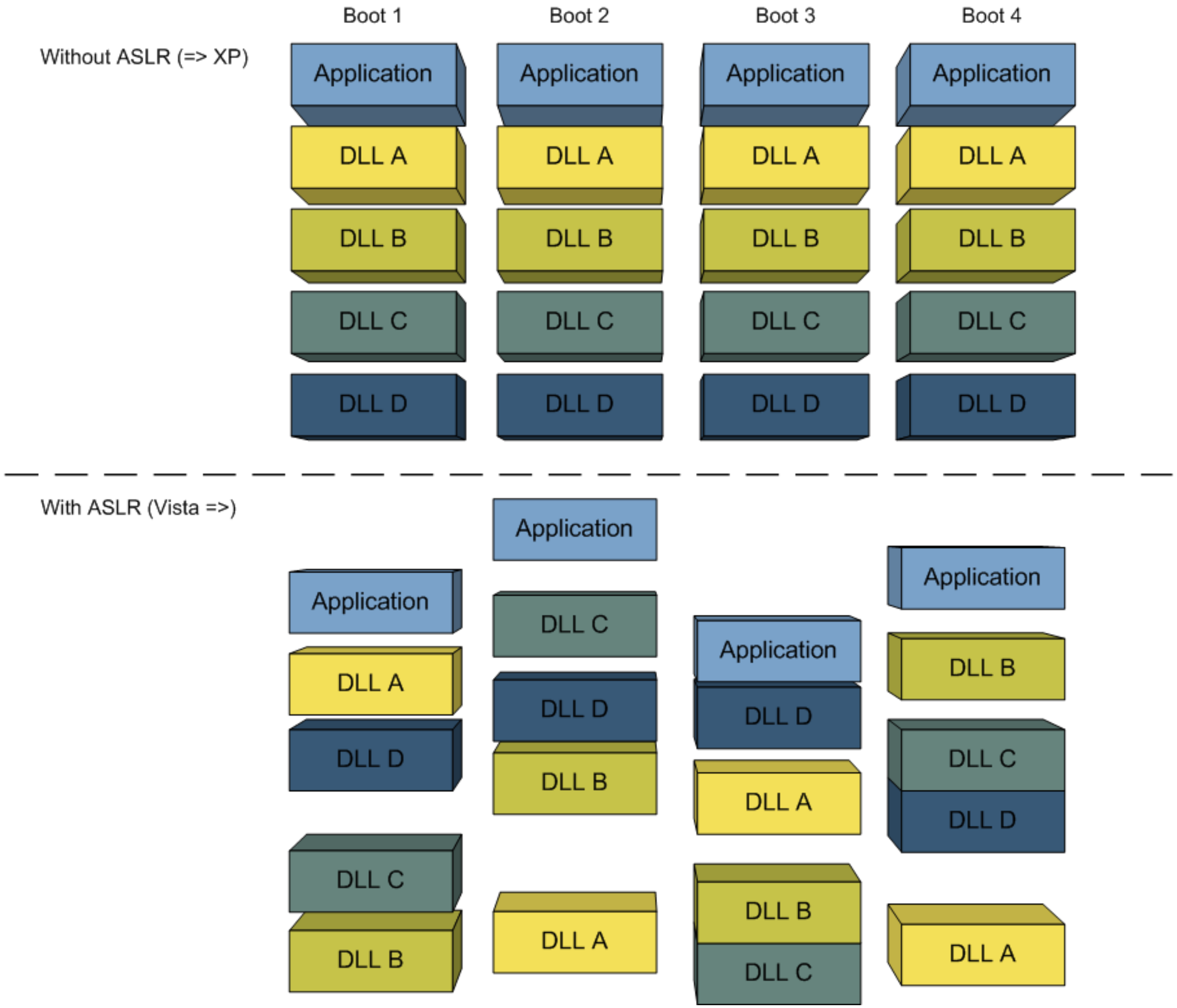
Publisher: Microsoft Corporation

Data Execution Prevention helps protect against damage from viruses and other security threats. What should I do?

Address Space Layout Randomization (ASLR)

- Traditionally all elements in the process memory has been loaded at predictable fixed addresses
- Fixed addresses can be exploited in buffer overflow attacks by jumping to specific existing functions
- ASLR causes elements to be loaded at random addresses
- ASLR makes it difficult (impossible) for attackers to know where exploitable functions are located in memory

ASLR illustrated



ASLR limitations

- ALL libraries must be ASLR-enabled
- Shellcode spraying is indifferent to layout
- Attacks relying on relative addressing
- It maybe possible to find non-randomized addresses
- Information leakage can reveal address to one specific libc module.
- ROP (Return-Oriented Programming) and JOP (Jump-Oriented Programming) possible with only one static libc code module

Trusted Computing

Trusted Computing: Basic idea

- Addition of security hardware functionality to a computer system to compensate for insecure software
- Enables external entities to have increased level of trust that the system will perform as expected/specified

- Trusted platform = a computing platform with a secure hardware component that forms a security foundation for software processes
- Trusted Computing = computing on a Trusted Platform

Trusted Hardware Examples

TPM Chip

iButton

IBM 4764

Fortezza PC Card

Smart Card

BankID OTP token

Characteristics of Trusted Hardware

- Physically secure module, two variants
  - Tamper resistant (difficult to penetrate physical protection)
  - Tamper proof (detection of physical penetration, self-destruction)
- Environmental monitoring (temperature, power supply, structural integrity)
- Optimized hardware support for cryptography
- I/O interface
- Secure manufacturing
- Secure customization

Trusted Computing Group (TCG)

TCG Promoters
intel.
AMD
Microsoft
hp HEWLETT PACKARD
Sun microsystems
IBM
SONY

... and many others.

TCG History & Evolution

- October 1999: TCPA formed
  - Trusted Computing Platform Alliance
  - Founders: IBM, HP, Compaq, Intel and Microsoft
- 2001: 1st TPM specification released
  - Trusted Platform Module
- 2002: TCPA changes its name to TCG
  - Trusted Computing Group
  - Incorporated not-for-profit industry standards organization
- 2003: TCPA TPM spec. adopted by TCG as TPM 1.2
- 2012: Draft TPM Specification 2.0 published
- 2014: Still draft TPM specification 2.0

TPM usage

- TPM is both the name of a standard and a chip
- TPM chip at the heart of hardware / software approach to trusted computing
- Current TPM chips implement TPM spec. 1.2
  - Latest version of TPM spec. 1.2 is from 2011
- TPM chip mounted on motherboard,
- TPM equipped computing platforms
  - Laptops, servers, pads, mobile phones
- Used by software platforms
  - Windows Vista / 7 / 8, Linux, and MAC OS
- Supports 3 basic services:
  - Authenticated/Secure boot,
  - Sealed Storage / Encryption
  - Remote attestation,

TPM supports two modes of booting

- Secure boot
  - The platform owner can define expected (trusted) measurements (hash values) of OS software modules.
  - Expected values are stored in special non-volatile PCR (Platform Configuration Registers) in the TPM.
  - Matching measurement values guarantee the integrity of the corresponding software modules.
  - If a measurement does not match the expected value for that stage of the boot process, TPM can signal a boot termination request.
- Authenticated boot
  - does not terminate boot, only check measured values against expected values from PCR, and records new values in PCRs

Sealed Storage / Encryption

- Encrypts data so it can be decrypted
  - by a certain machine in given configuration
- Depends on
  - master secret key unique to machine
  - used to generate secret encryption key for every possible configuration only usable in it
- Can also extend this scheme upward
  - create application key for desired application version running on desired system version
- Supports disk encryption

Remote Attestation

- TPM can certify configuration to others
  - with a digital certificate of configuration info
  - giving another user confidence in it
  - Based on endorsement credential and identity credential
- Include current challenge value in certificate to also ensure attestation is fresh
- Provides hierarchical certification approach
  - trust TPM, then OS, then applications

TPM Platform Identity

- ***Endorsement Key*** pair :
  - Public/private key pair generated during manufacture
  - Uniquely identifies each TPM
  - Optional support for EK reset
  - TPM can not export private part of endorsement key
- ***Endorsement Credential*:**
  - Certificate used to prove to external systems that they communicate with a genuine TPM
  - Anonymous, can not be used to identify unique TPM
  - Used for remote attestation
- ***Identity Credentials*:**
  - Derived from EK
  - Used to identify unique TPM'
  - Used for remote attestation

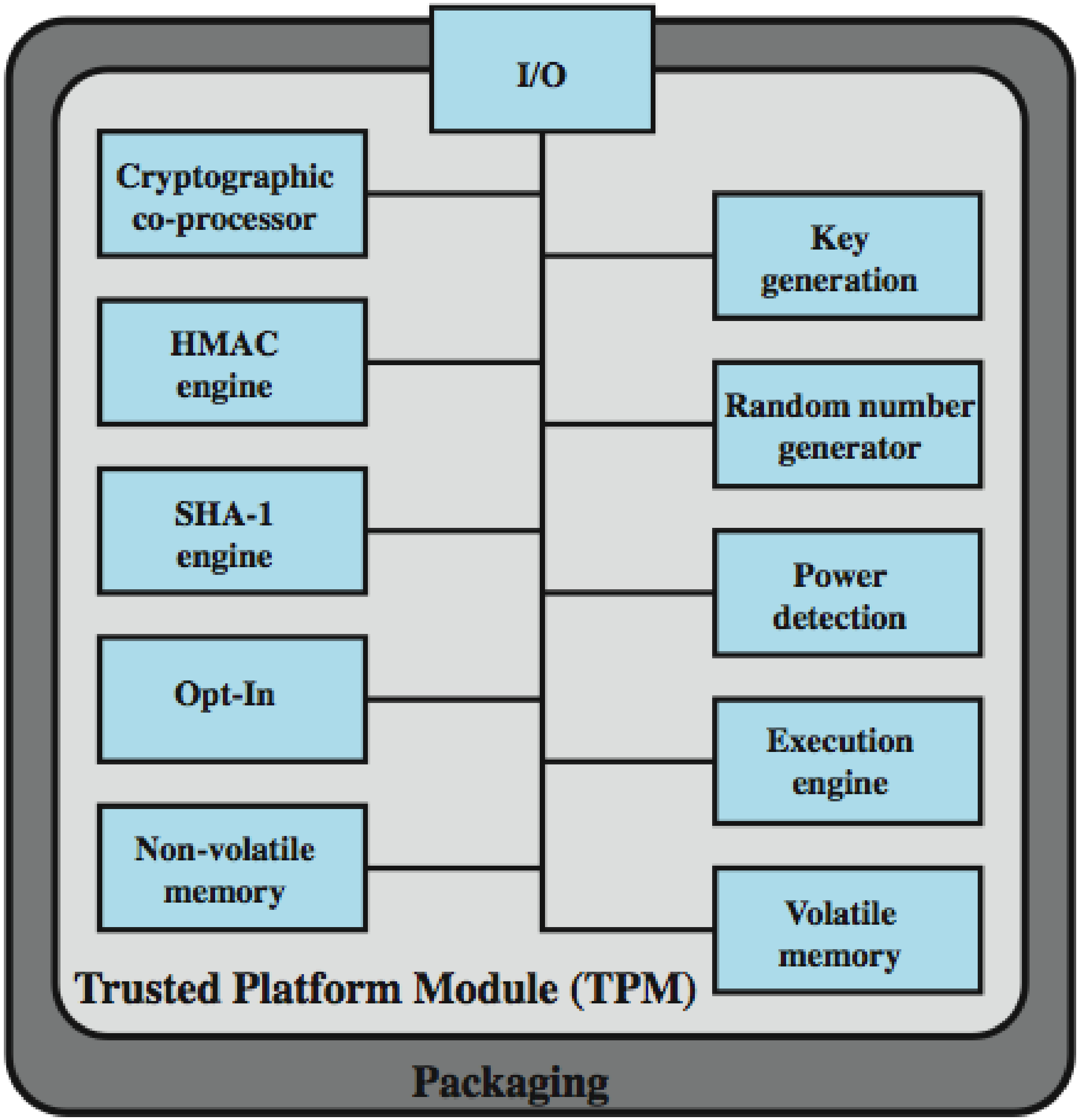
Endorsement key pair

Public/Private

Genuine and Anonymous TPM

Genuine and Unique TPM

TPM 1.2 hardware elements

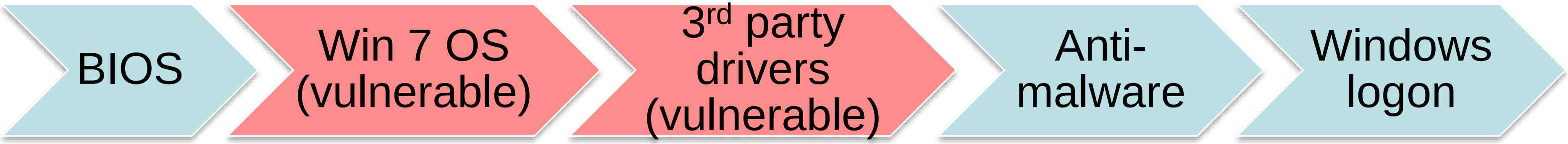


TPM Disadvantages

- TPM security functions depend on measurement (hash) values stored in PCR registers.
- Changes in system configuration can lead to measurements no longer matching PCR values.
  - E.g. patching and updates
- Non-matching measurement values can lead to:
  - booting no longer possible,
  - encrypted data not recoverable, data loss.
- Secure boot of pre-installed OS makes it difficult or impossible to install a new OS
  - Unfair market power
- Software integrity protection based on TPM does not protect against infection during run-time
  - But infection might be detected during next system boot.

TPM Secure Boot in Windows 8

Windows 7 boot



BIOS

Win 7 OS (vulnerable)

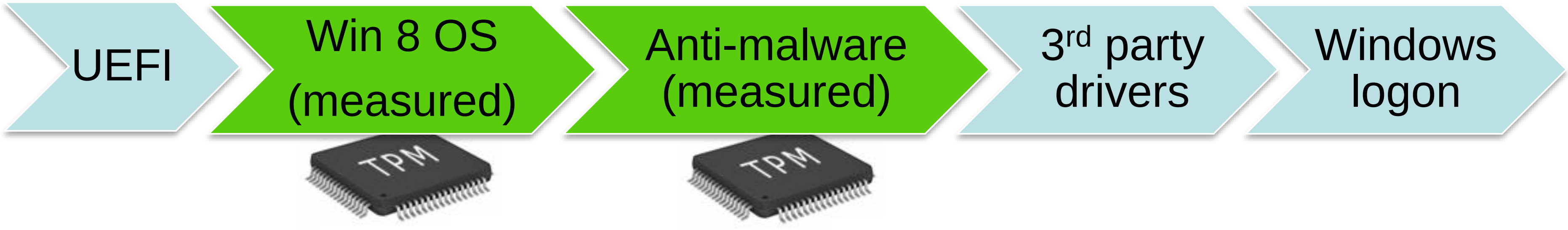
3rd party drivers (vulnerable)

Anti-malware

Windows logon

- Malware is able to start before Windows 7 and Anti-malware
  - Malware able to hide and remain undetected
  - Systems can be completely compromised

Windows 8 boot



UEFI

Win 8 OS (measured)

Anti-malware (measured)

3rd party drivers

Windows logon

- Possible to start anti-malware early in the Windows 8 boot process
  - Early Launch Anti-Malware (ELAM) driver is signed by Microsoft
  - Malware can no longer bypass Anti-Malware inspection
  - UEFI (Unified Extensible Firmware Interface) replaces BIOS

End of lecture