CSE413 – Security of Information Systems 2020

PhD Furkan Gözükara, Toros University

https://github.com/FurkanGozukara/Security-of-Information-Systems-CSE413-2020

Lecture 6

Computer Security

Composed from Prof. Audun Jøsang, University of Oslo, Information Security 2018 Lectures

Source : https://www.uio.no/studier/emner/matnat/ifi/INF3510/v18/lectures/

Lecture Overview

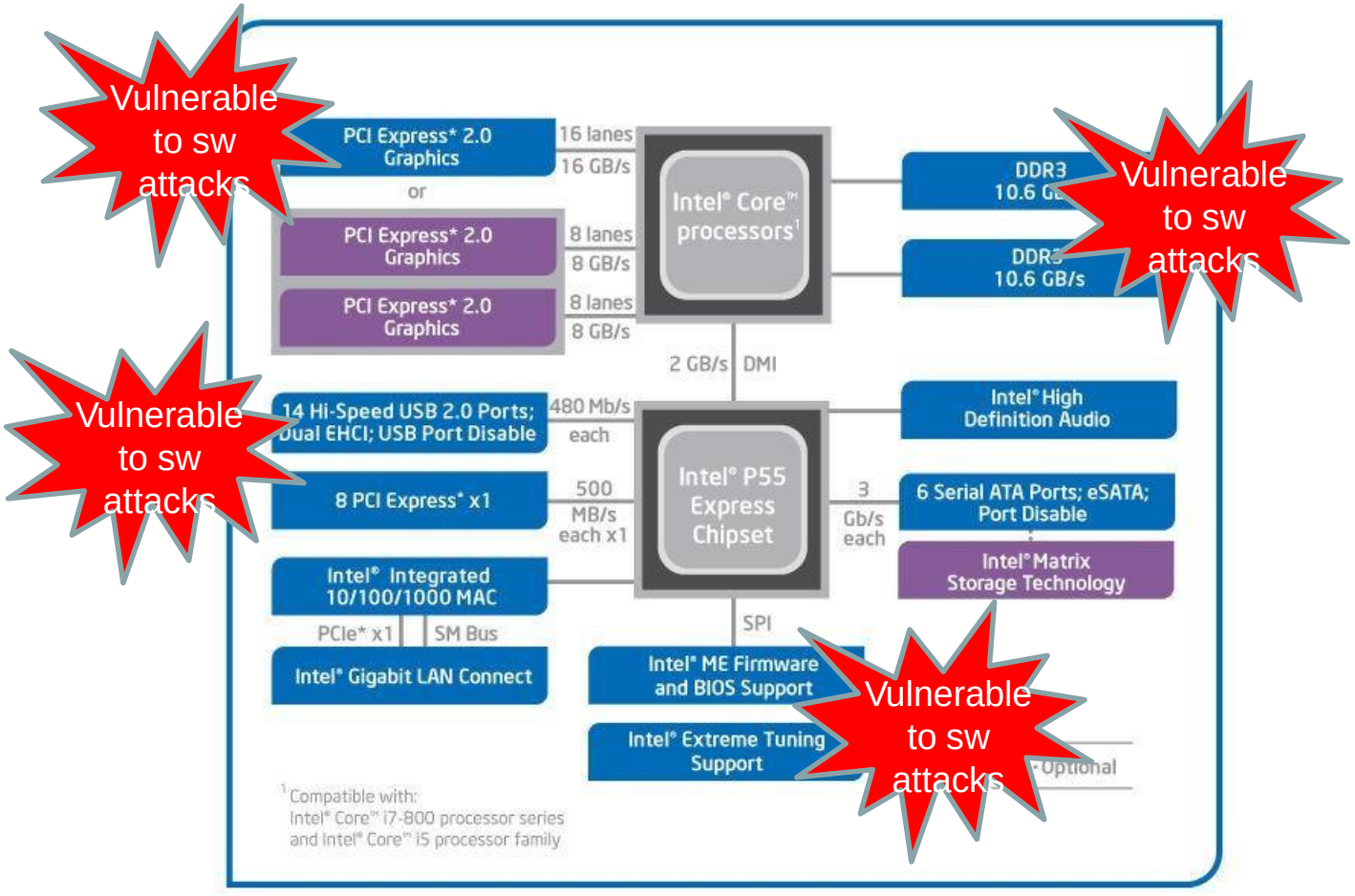
- Secure computer architectures
- Virtualization architectures
- Trusted computing

System & Communication Security

- "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.“

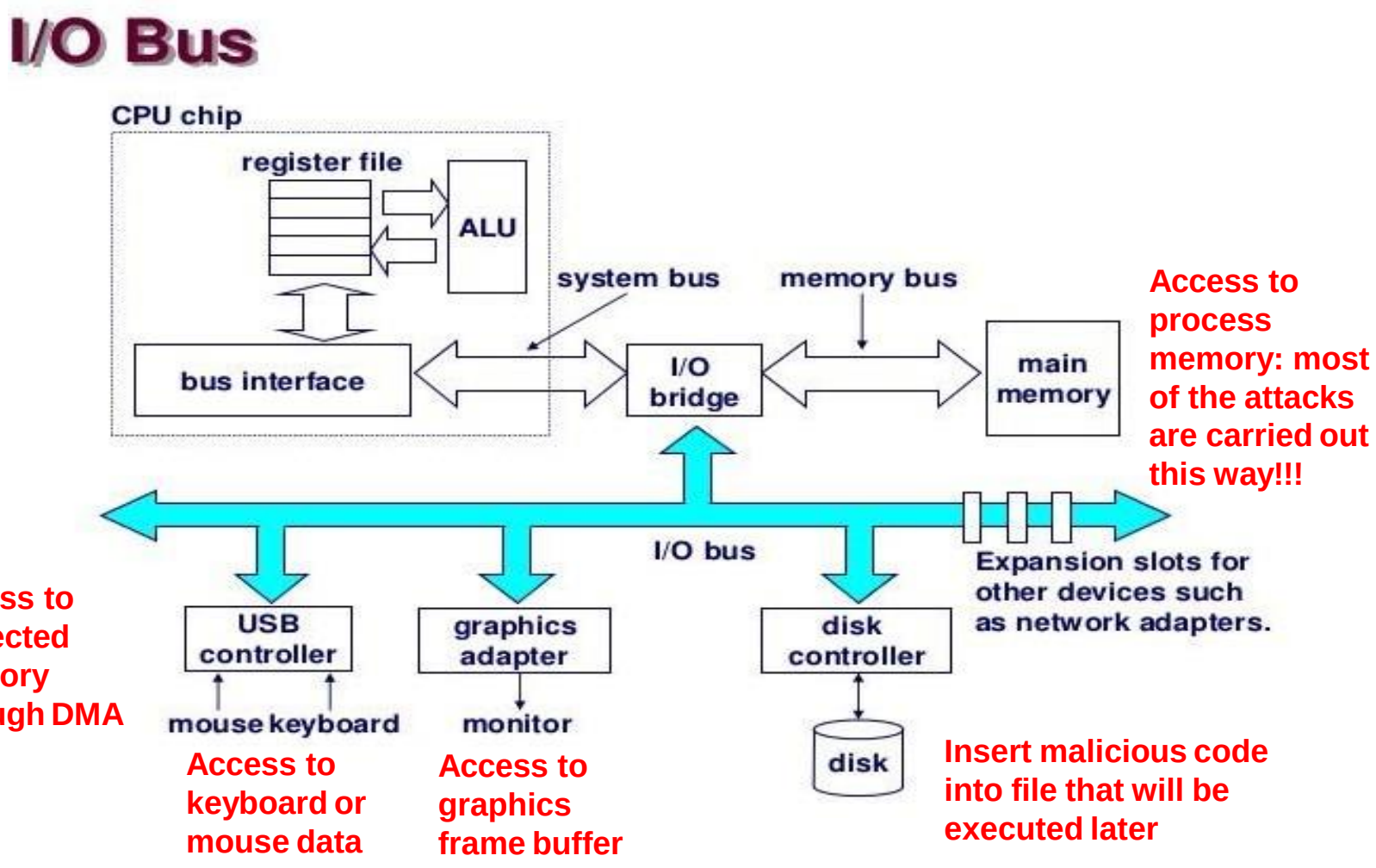
  (Gene Spafford)

Vulnerabilities of the PC today



Intel® P55 Express Chipset Platform Block Diagram

Vulnerabilities of the PC today



Approaches to strengthening platform security

- Harden the operating system
  - SE (Security Enhanced) Linux, Trusted Solaris, Windows 7/8/10
- 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

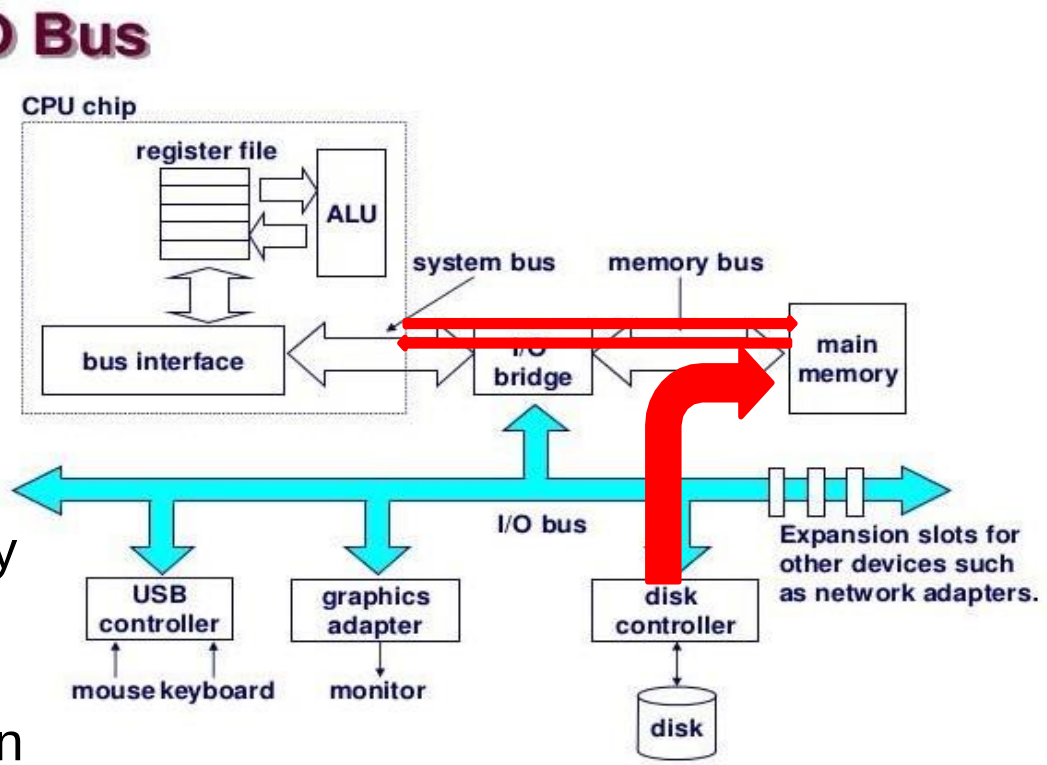
OS protections – Launching a process

When the system runs an executable, the code and the data of the executable are loaded into the Random Access Memory

The OS analyze the code and also copies the required dependencies and the necessary OS API files into the memory

Each process has an own Virtual Address Space, where all necessary code and data are loaded

The process can access only its own Virtual Address Space by CPU instructions



OS protections – Process Virtual Address Space

- The process code can access its own address space, but this addressing is different from the global physical addressing. Each process has the maximum addressable memory range: a virtual memory in a virtual address space (this is 232 (4GB) i.e. in 32bit systems 0x00000000 – 0xffffffff)

- To ensure the correct memory operations the OS has to provide a runtime address translation between virtual memory to the physical memory

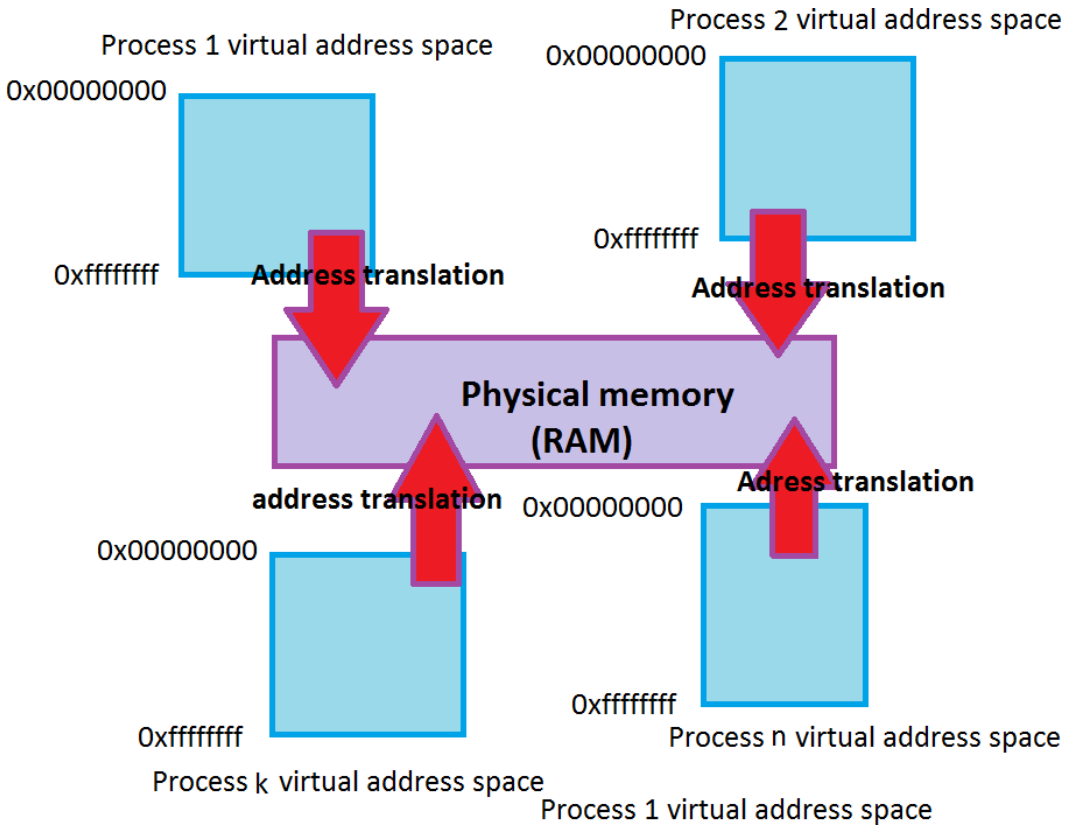
OS protections – Process Virtual Address Space

- Because of this way of operation the OS provides 2 protections by design: The program code that is executed cannot access the physical memory directly (We cannot write code to access physical memory in none of the programming languages)

- And also the OS protects the process code and data from each other. Normally a process cannot have access to another process memory (except debugging a process by another process or using specific OS features such as CreateRemoteThread)

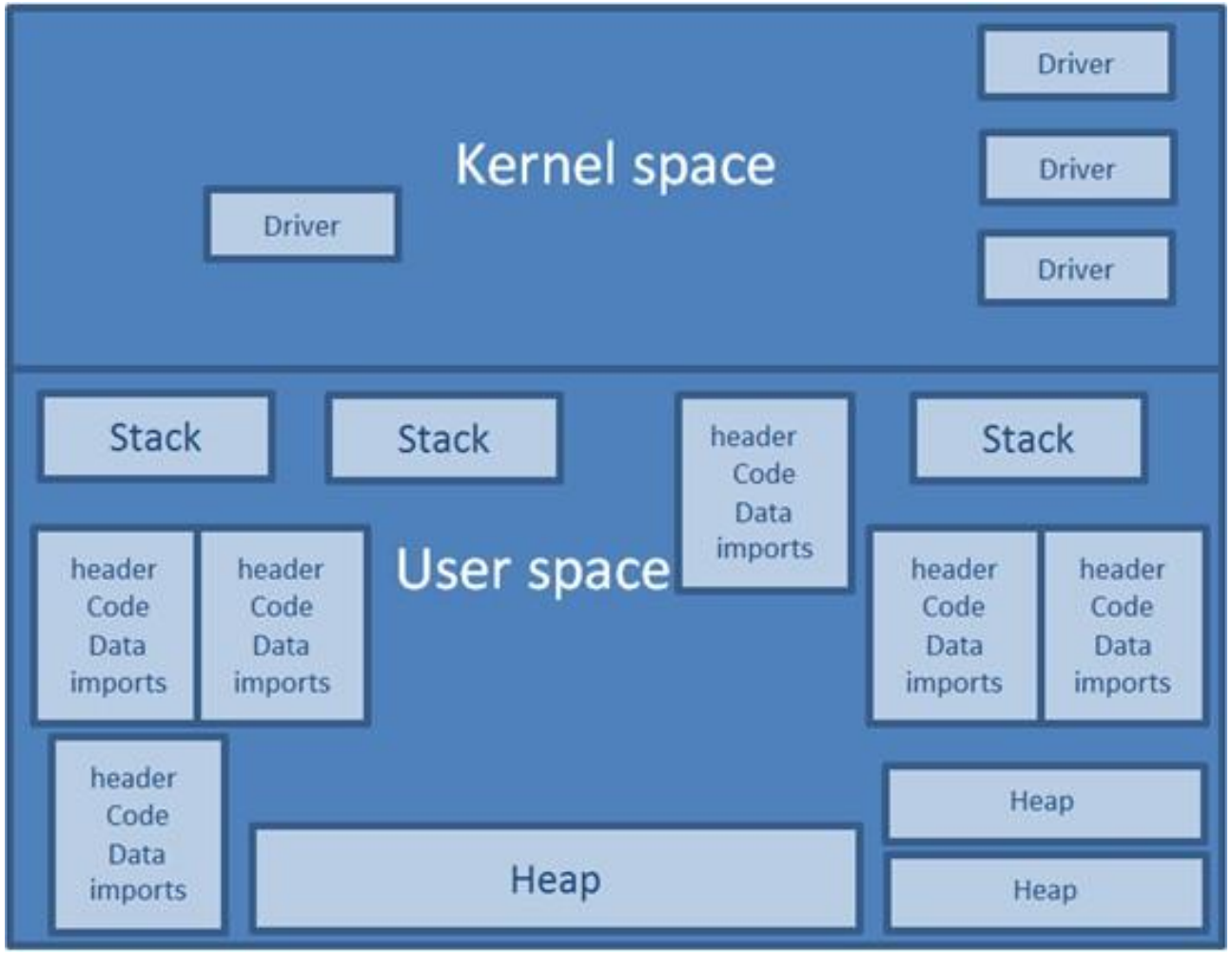
OS protections – Runtime Address Translation

- The runtime address translation is to optimize memory usage of processes and also to protect the process data



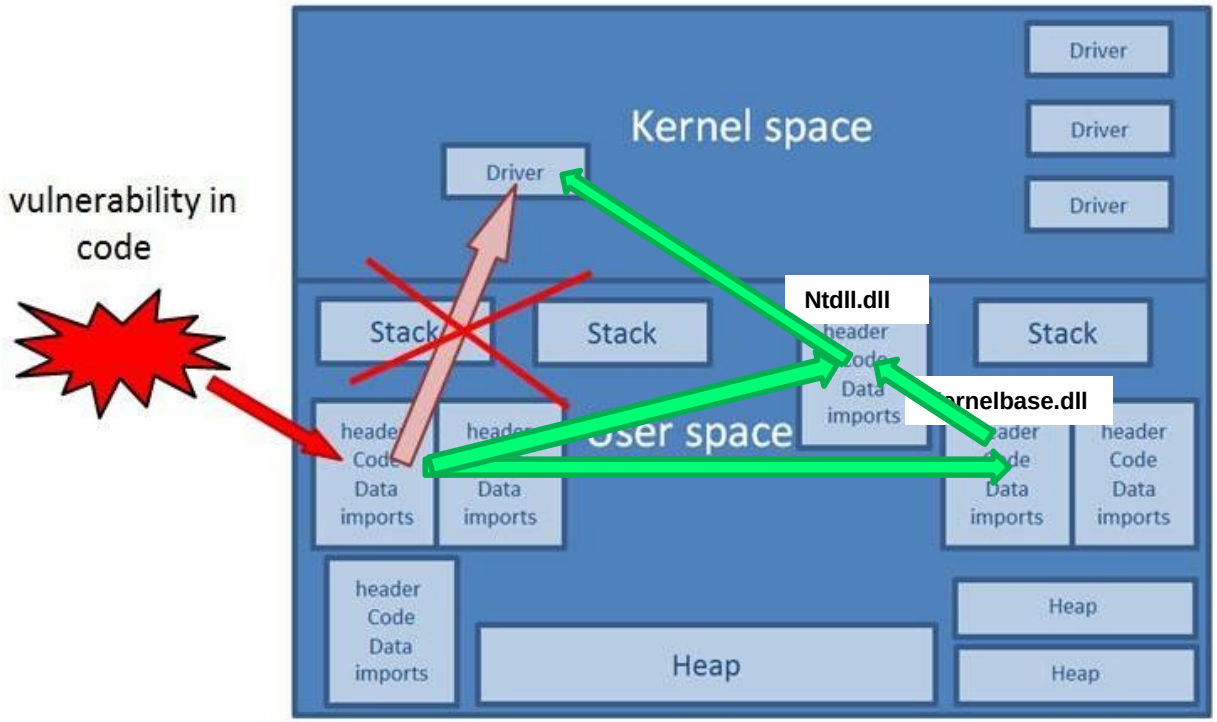
OS protections – Virtual Address Space

- The Virtual Address Space is separated into kernel and user space



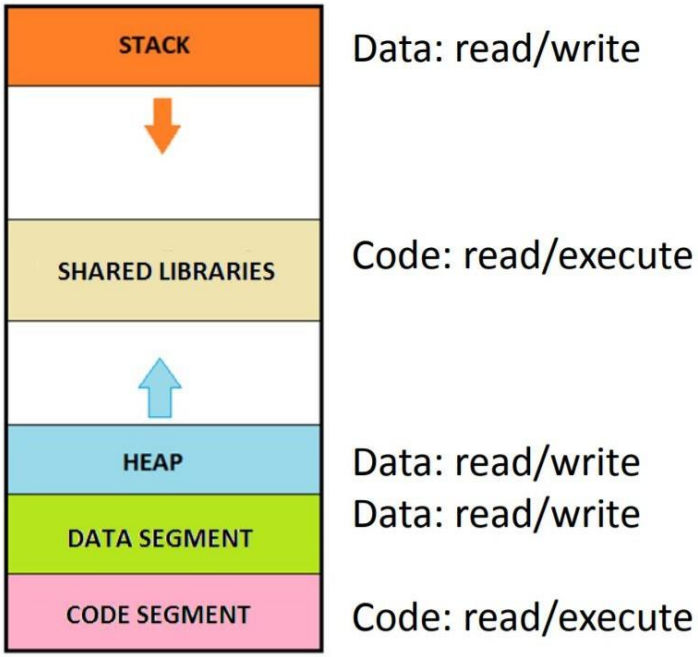
OS protections – Code vulnerabilities

- The kernel space can be accessed through only specific OS APIs (e.g. in Windows using the native API, Hardware protection CPU rings - see later)



OS protections – Software supported Data Execution Prevention (DEP)

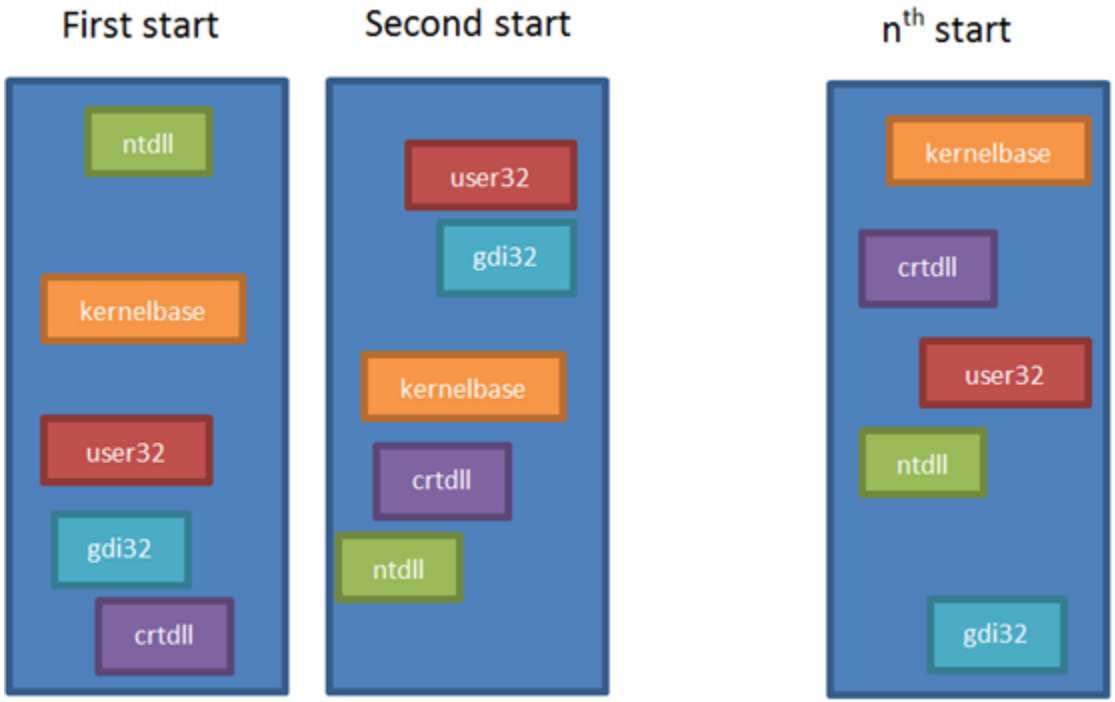
- The vulnerable code has no access directly to kernel space, but without DEP it can modify the code in the user space (even the OS API in user space) or can execute an attacker placed malicious code



- The OS can enforce DEP (i.e nxAlwaysOn settings on Windows)

OS protections – Address Space Layout Randomization (ASLR)

- Because of DEP the attackers turned into code reuse. To prevent it OS provides code randomization in the Virtual Address Space



OS advanced protections – new directions

- Increasing the entropy of ASLR makes brute-forcing less effective (High Entropy ASLR)

- Address Space Layout Permutation (ASLP) the place of the libraries are randomized as well as the order of the methods in the Virtual Address Space

- Code diversity (OS protection ?)

OS advanced protections – new directions

- Multiple heaps and protected heap segments (e.g. late free, MS Edge)

- Execute no Read segments (XnR protection, hardware support later?)

- Control Flow Integrity (observation of unintended use-cases)

CPU protection

- Protection rings
- Hardware supported Data Execution Prevention

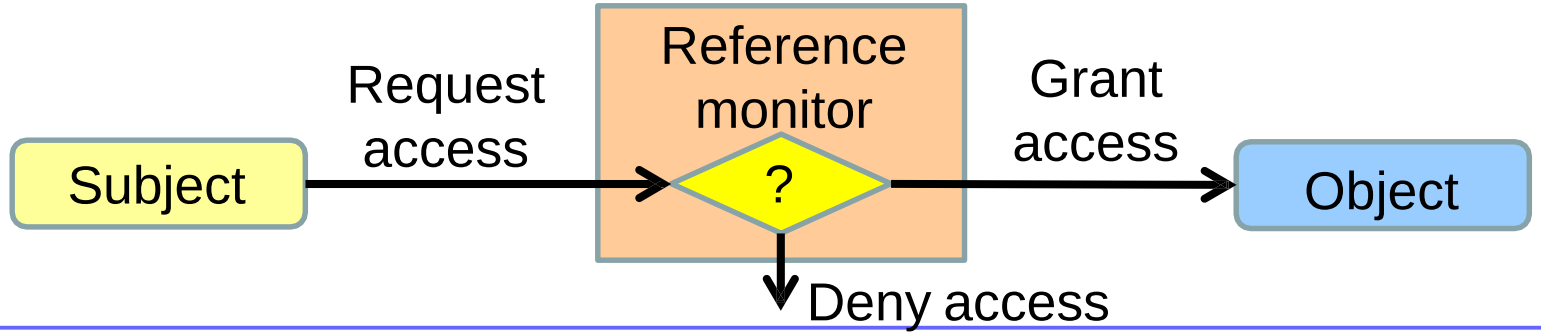
- Hardware supported Control Flow Integrity (not in use yet, e.g. Intel's Control Flow Enforcement)

- Intel sgx (Protecting memory regions from higher privileged access)

Reference Monitor

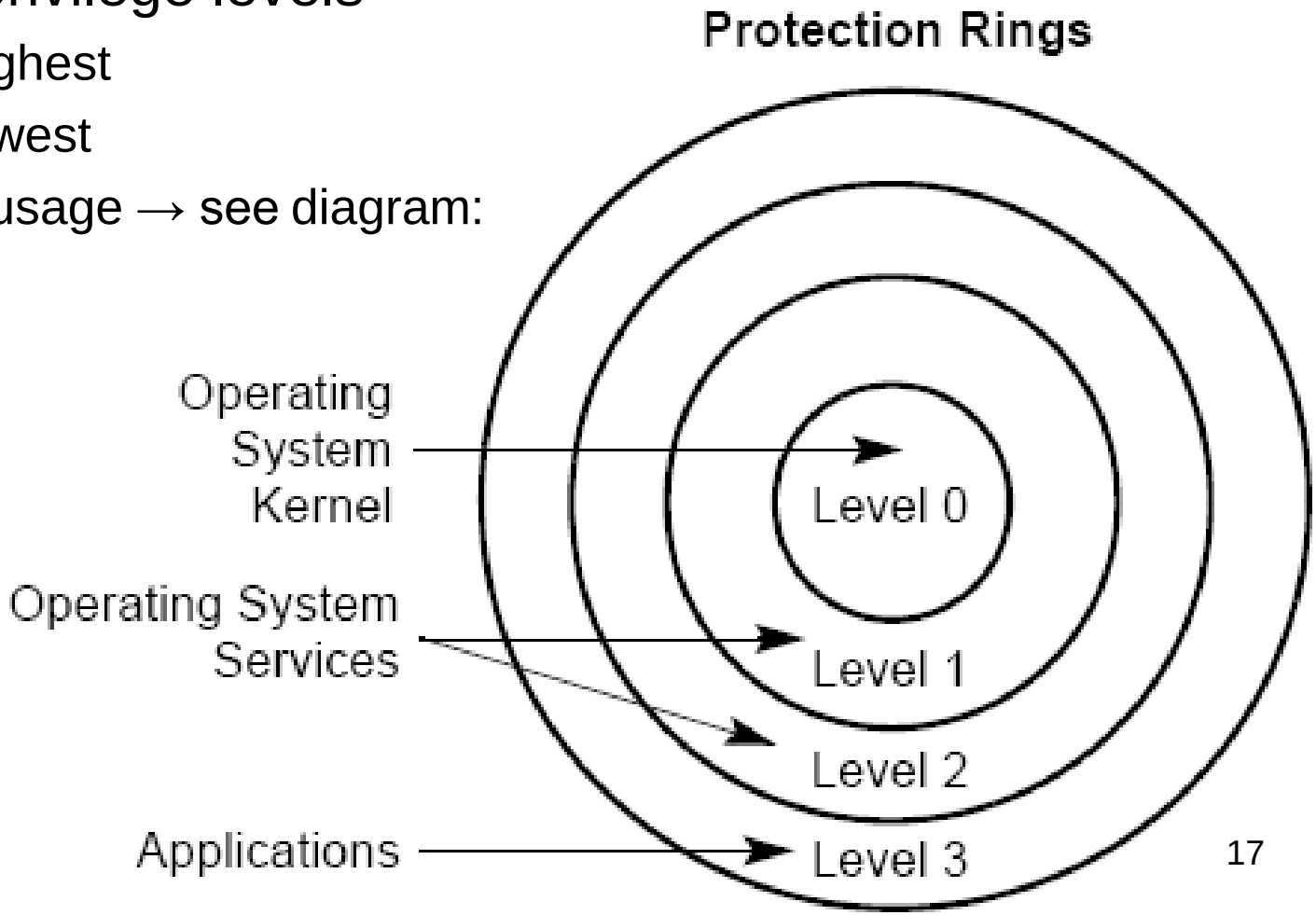
- A reference monitor is any security model for enforcing an access control policy over subjects' (e.g., processes and users) ability to perform operations (e.g., read and write) on objects (e.g., files, memory and sockets) on a system.
  - The reference monitor must always be invoked (complete mediation).
  - 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) implementation 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:



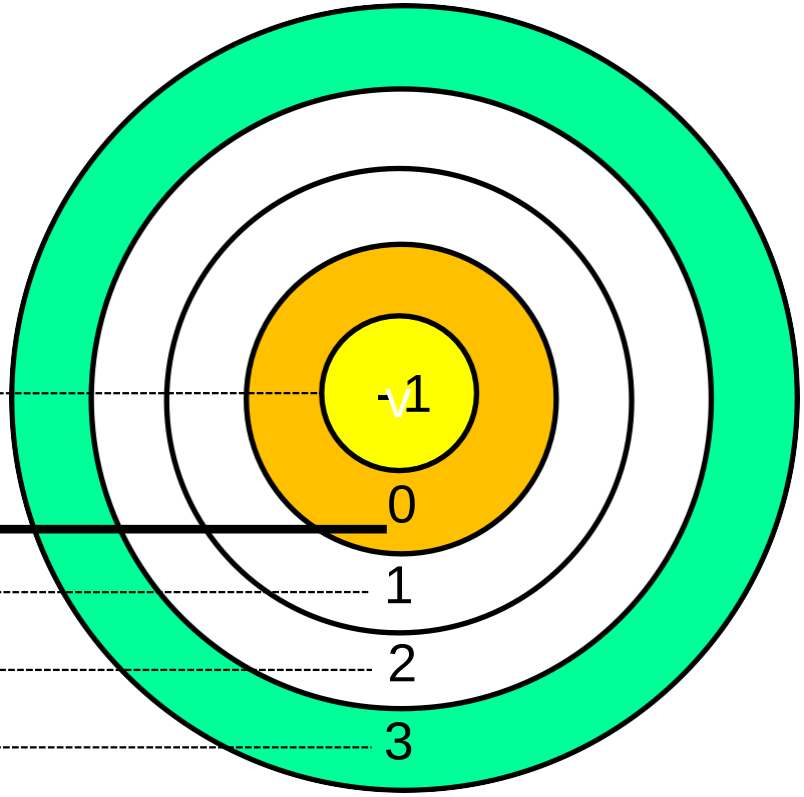
What happened to rings 1 & 2 ?

... 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

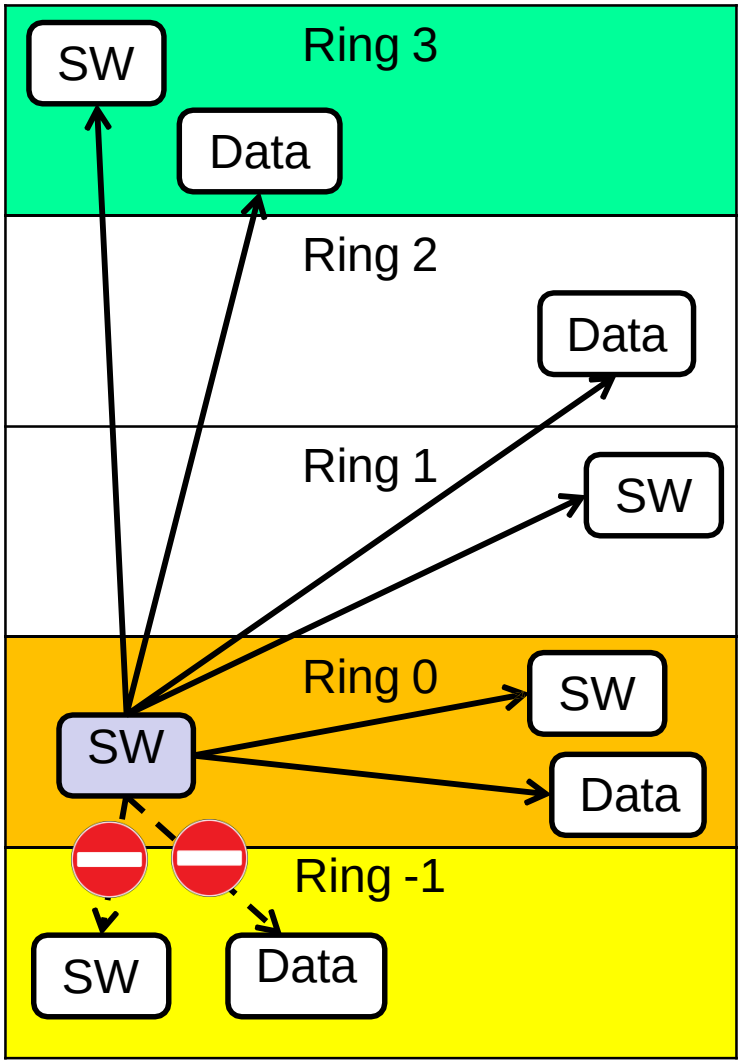
Ring 3: User Mode

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

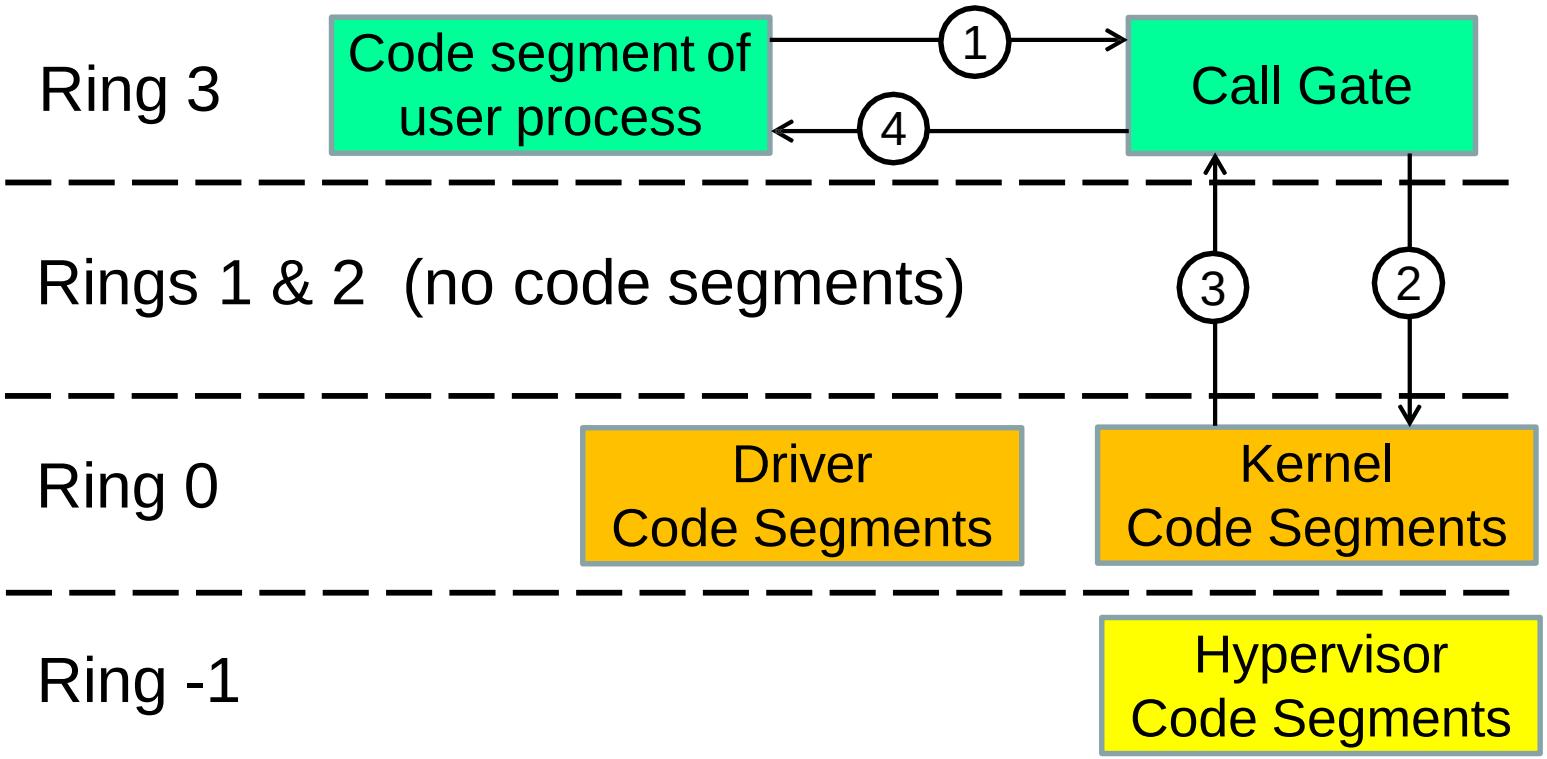


User processes access to system resources

- 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 (Current Privilege Level, CPL).

- 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



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.

Hardware supported Data Execution Prevention

- Intel Processors with XD support
- AMD processors with NX support

Hardware-enforced DEP marks all memory locations as non-executable (you cannot execute code in this portion of memory) unless the location explicitly contains executable code.

Hardware-enforced DEP relies on processor hardware to mark memory with an attribute that indicates that code should not be executed from that memory.

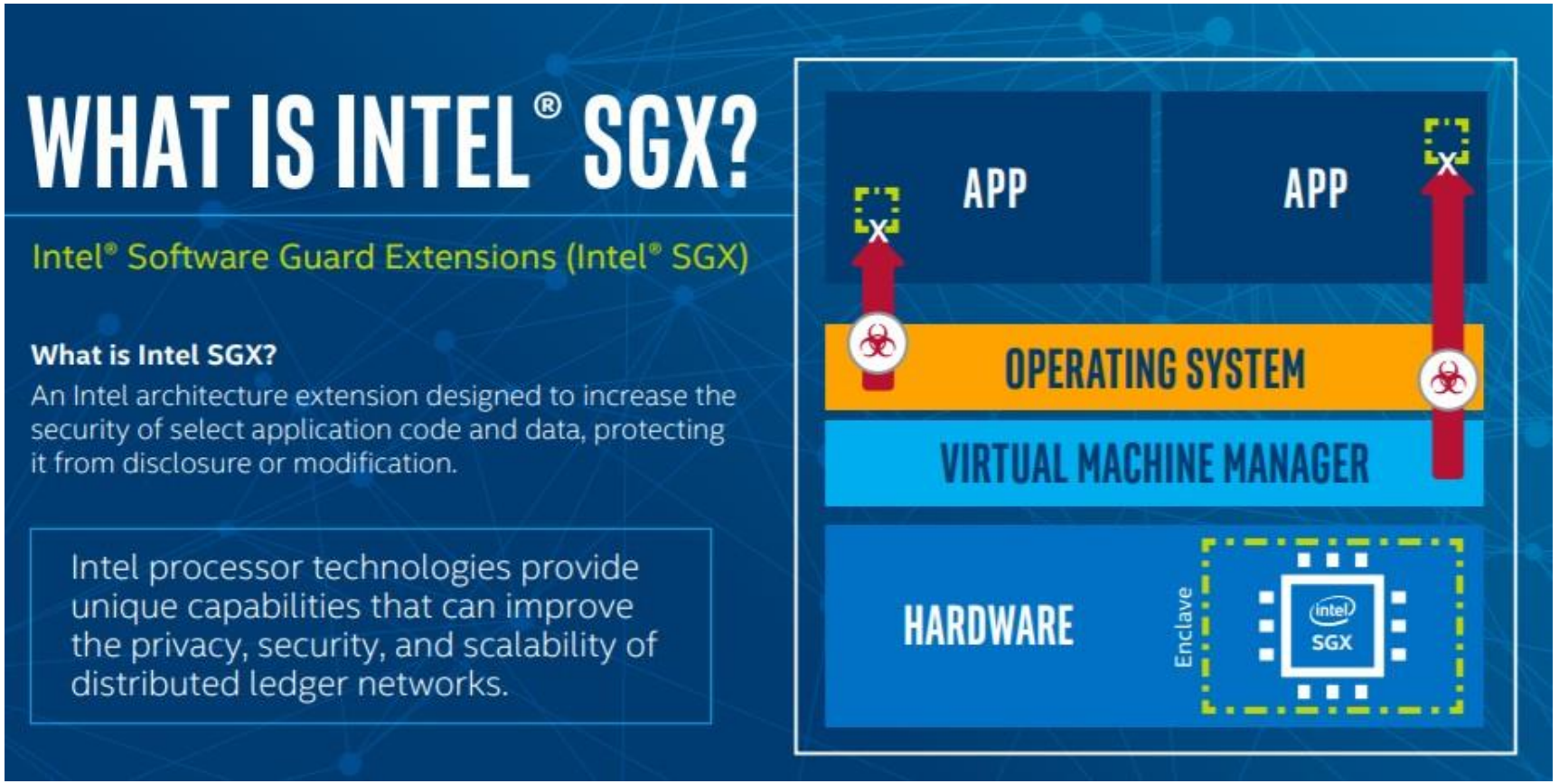
Processors that support hardware-enforced DEP are capable of raising an exception when code is executed from a memory location where it should not be executed.

To use these processor features, the processor must run in Physical Address Extension (PAE) mode. HP ships Windows XP with PAE enabled.

Control Flow Enforcement (in plan)

- Processors will directly support: Shadow (call) stack tracking. Method return addresses are stored in data stack and the shadow stack too. Shadow stack is not accessible, the processor checks the return addresses, if there's un-matching return then control protection exception is raised
- Processors also support: indirect branch tracking. After each legitimate indirect branch instruction the code must contain a *nop-like* special instruction. If the program execution is redirected by an attacker, the nop-like instruction is missing and a control flow protection error is raised.
- Control Flow Enforcement Technology is announced by Intel in June 2016. Release date is unknown.

Intel Software Guard Extension (SGX)



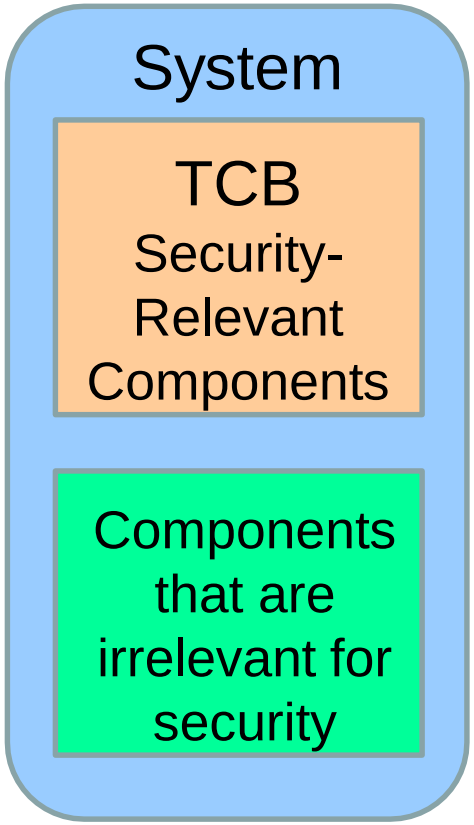
https://newsroom.intel.com/news/intel-microsoft-enterprise-blockchain-service/

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

From TCSEC

Trusted Computer Evaluation Criteria, 1985.

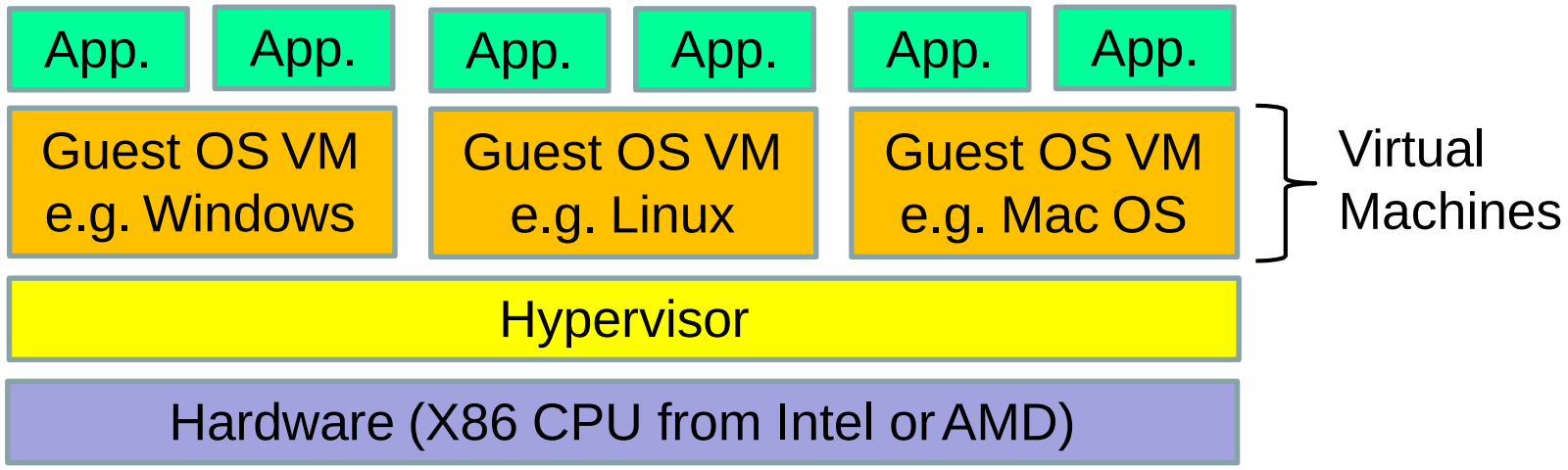


Platform Virtualization

Platform Virtualization

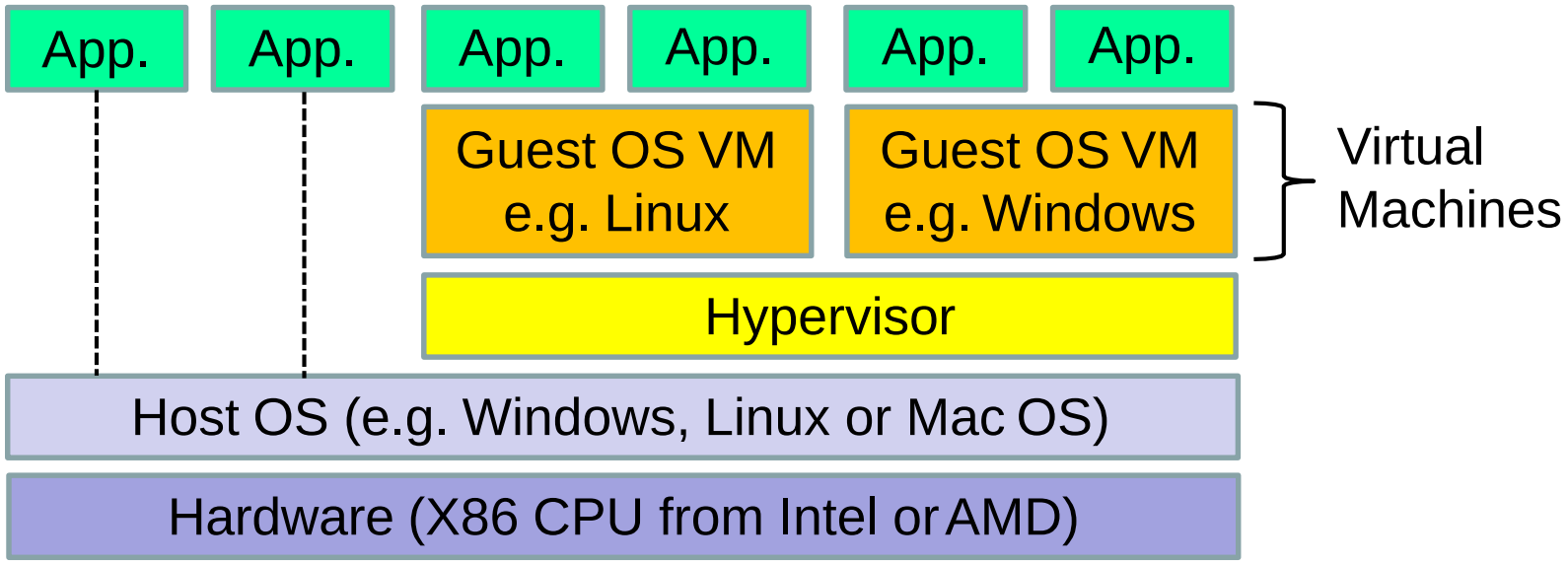
- Hypervisor (aka. VMM - Virtual Machine Monitor) is needed to manage multiple guest OSs (virtual machines) in the same hardware platform.
- Many types of hypervisors available
  - VMWare is most known Commercial product (Type 1&2)
    - Free version comes with a limitations
  - VirtualBox is a hypervisor for x86 virtualization
    - It is freely availably under GPL, Type 2
    - Runs on Windows, Linux, OS X and Solaris hosts
  - Hyper-V is Microsoft's hypervisor technology (Type 1)
    - Requires Windows Server
  - Xen, powerful open source hypervisor, (Type 1)

Type 1 VM Architecture (native)



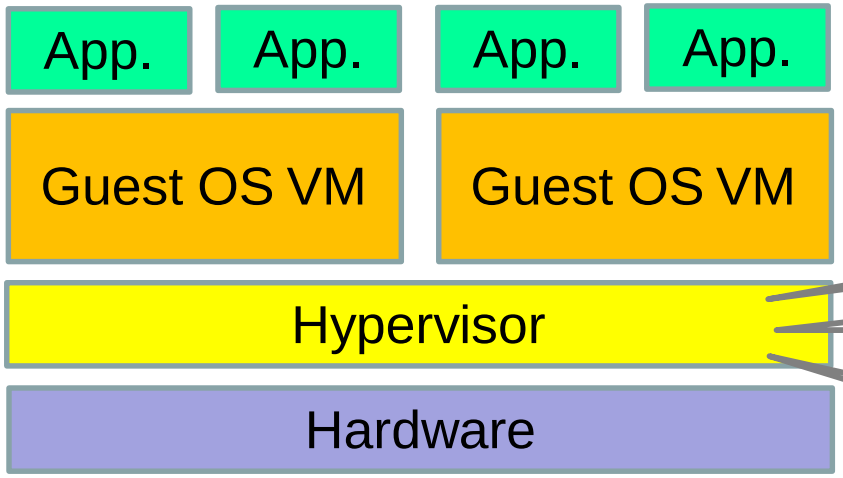
- No host OS
- Hypervisor runs directly on hardware
- High performance
- Traditionally limited GUI, but is improved in modern versions
- HW support can be an issue

Type 2 VM Architecture (hosted)



- 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

Challenges of Running VMs



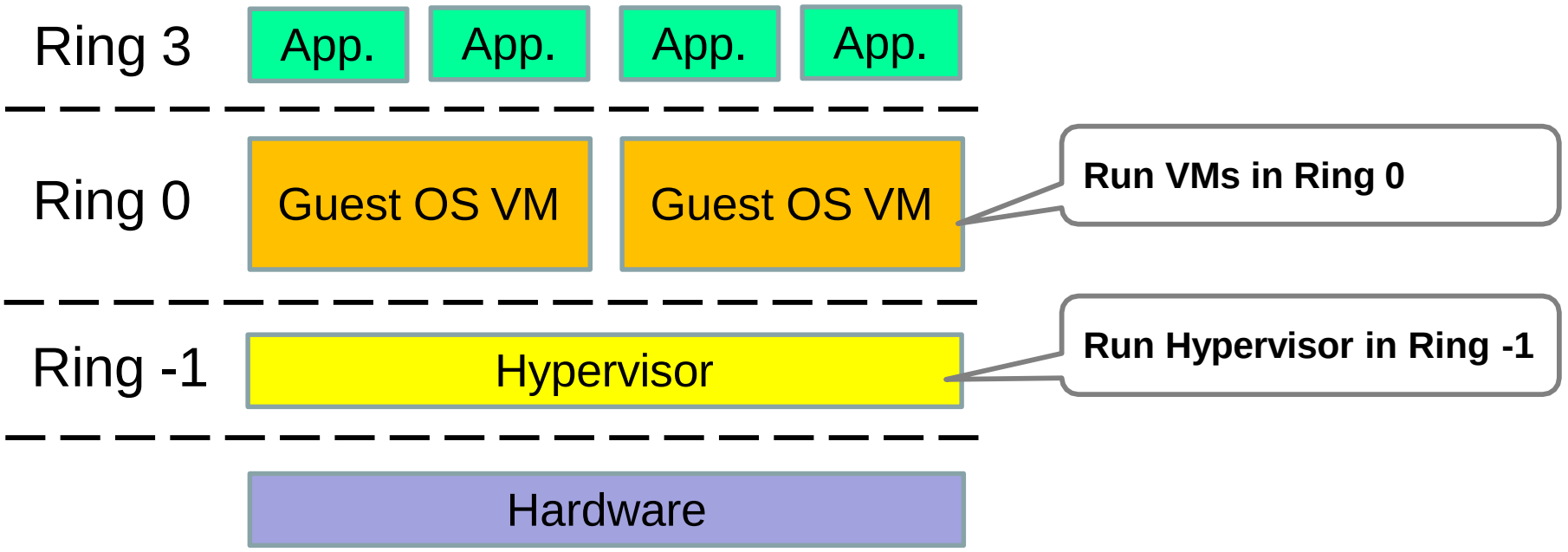
VMs and Apps in a VM must not know that Hypervisor exists or that they share HW resources with other VMs

Hypervisor must protect VMs' memory areas from each other

Hypervisor must protected itself from all VMs

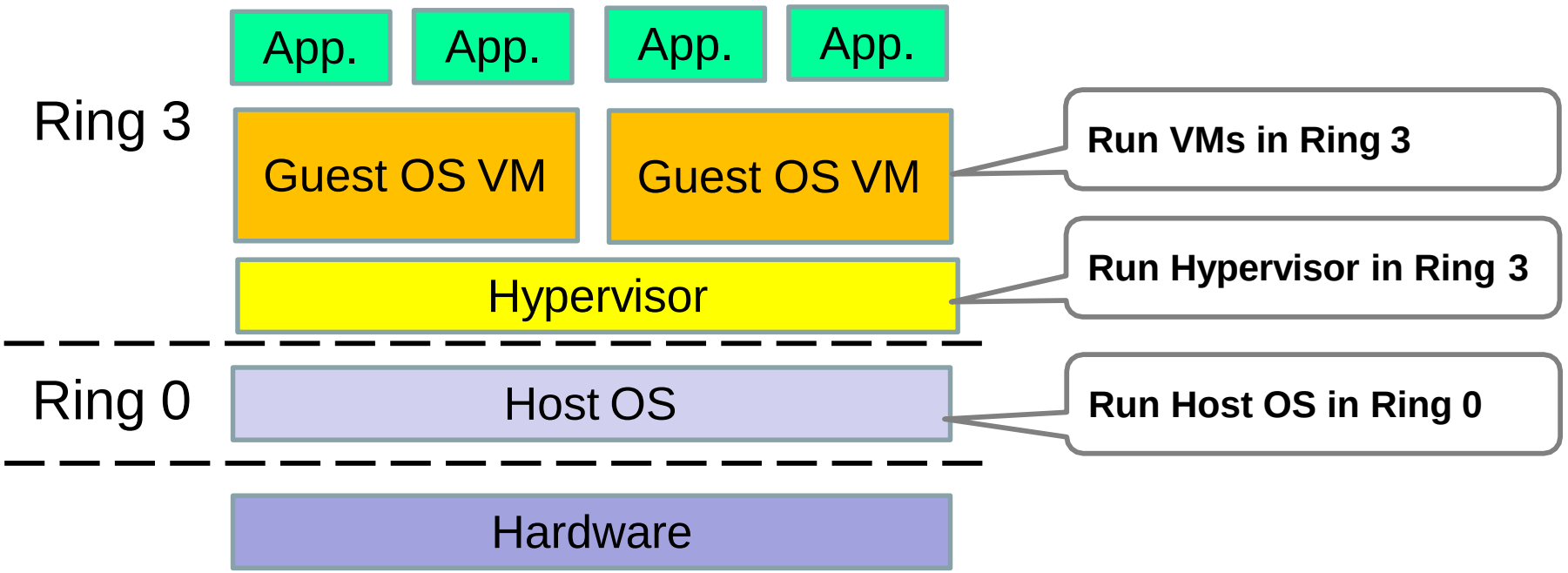
Hypervisor must present virtual hardware interface to VMs

Type 1 VM Architecture Ring Allocation



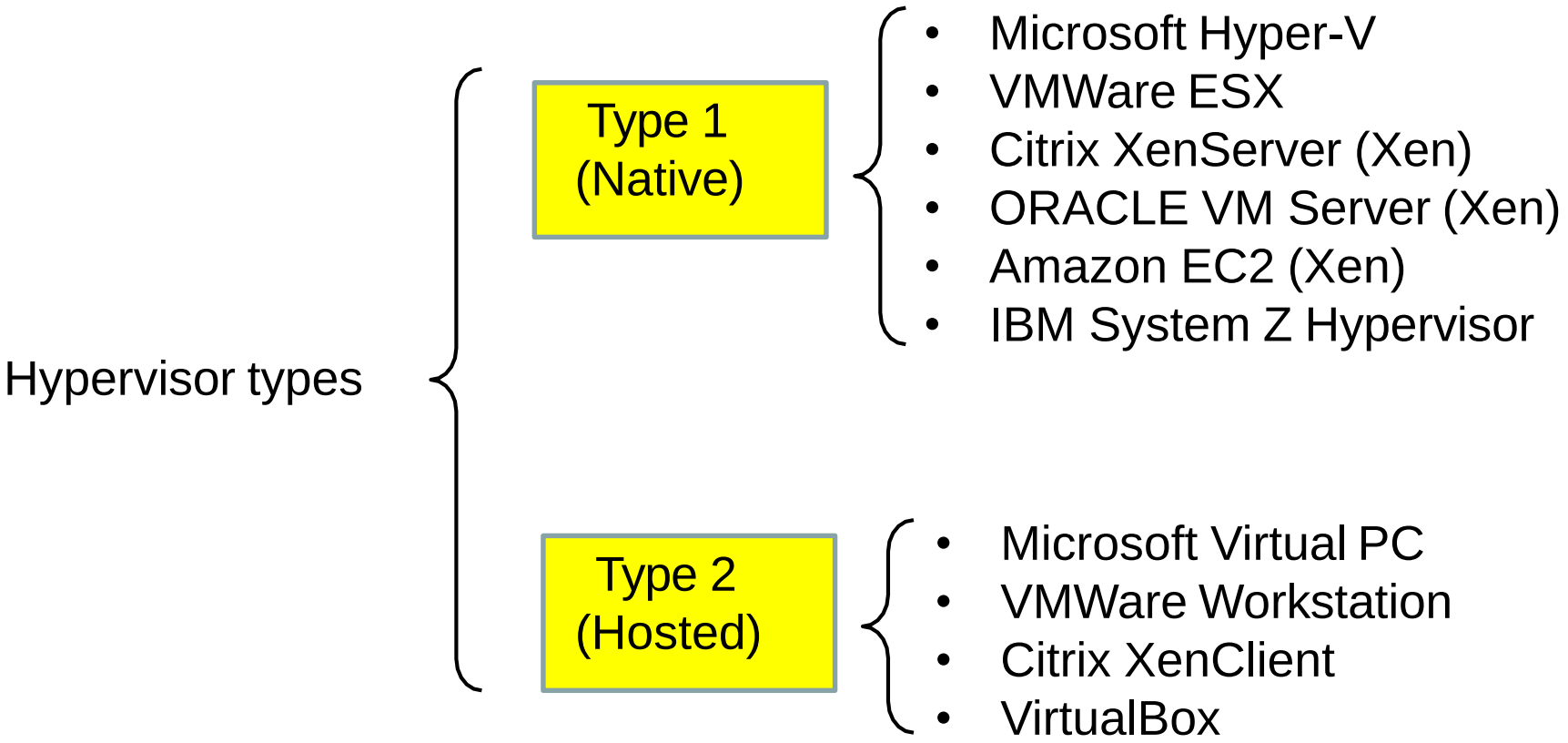
- Guest OS VMs are less privileged than the hypervisor.
- Hypervisor is well protected from the VMs.
- Good performance and good security !

Type 2 VM Architecture Ring Allocation

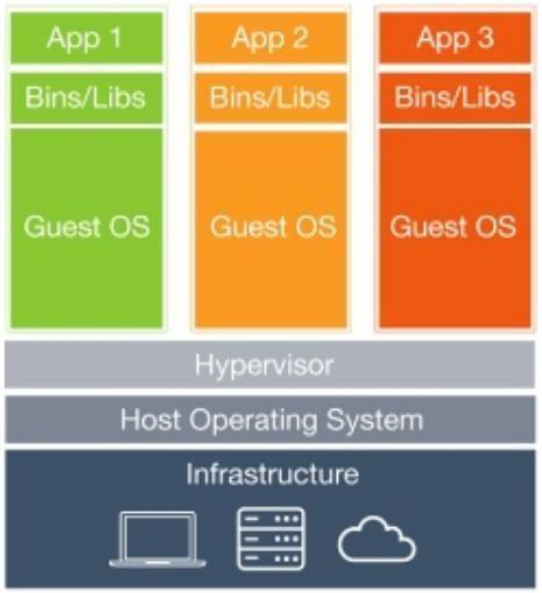


- Guest OS VMs run in Ring 3.
- Guest OS VMs call privileged instructions that are forbidden in Ring 3.
- Forbidden instructions cause exceptions that are handled by interrupt/exception handler to be executed.
- Slow performance !

Platform Virtualisation Products

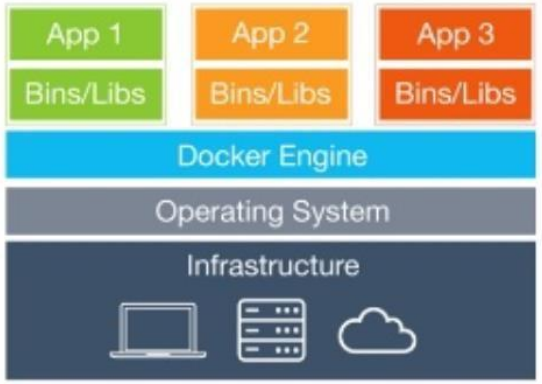


Operating system level virtualization



Virtual Machines

Each virtual machine includes the application, the necessary binaries and libraries and an entire guest operating system - all of which may be tens of GBs in size.



Containers

Containers include the application and all of its dependencies, but share the kernel with other containers. They run as an isolated process in userspace on the host operating system. They're also not tied to any specific infrastructure – Docker containers run on any computer, on any infrastructure and in any cloud.

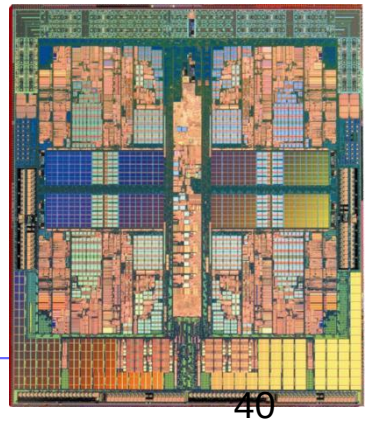
https://www.slideshare.net/GiacomoVacca/docker-from-scratch

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

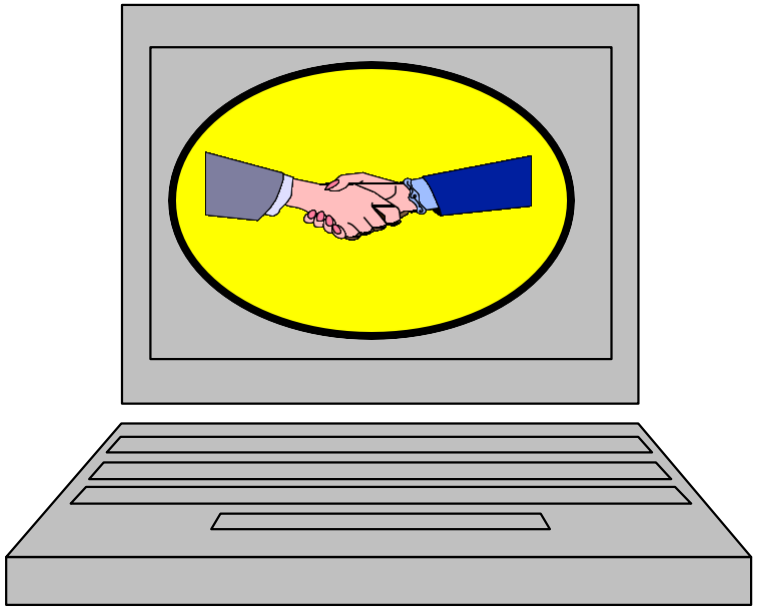
Hypervisor examples of use

- 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

Amazon EC2 Data Centre

- 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

Trusted Computing



Trusted Computing Motivation

- Software alone can not be trusted.
- Malware infection in OS kernel remains undetected by anti-malware tools.
- Physical access to computers opens up for attacks that can circumvent traditional TCBs (Trusted Computing Base), e.g. secure operating systems.
- Remote parties do not know the status of systems they are communicating with.
- Remote parties do not know the physical identity of hosts they are communicating with.

Basic idea of Trusted Computing

- Use specialised **security hardware** as part of TCB in a computer system
  - Can not be compromised by malware
  - Can verify the integrity of OS kernel
  - Can make physical tampering difficult
  - Can report status of system to remote parties
  - Can report identity of system to remote parties
- Gives increased level of trust that the system will perform as expected/specified

What is “trust” in the sense of TC?

- To have confidence in assumptions about security
- Trust is to believe that security assertions will hold

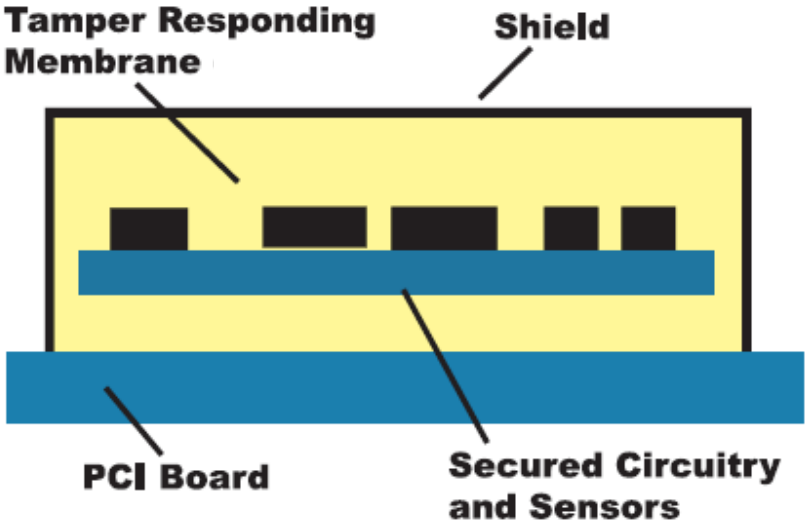
  “*A trusted component, operation, or process is one whose behaviour is assumed to be correct under any operating condition, and which is assumed to resist subversion by malicious software, viruses, and manipulations*”
- A trusted component enforces the security policy as long as these assumptions hold
- A trusted component violates the security policy if it breaks
- Q1: How do you know that a component is ‘trustworthy’, i.e. that it will not break ? A1: Through ‘assurance’
- Q2: Trusted by whom to do what ?
  - Trusted by **user**, by **vendor**, or by **3rd party (NSA)**
  - What if they have conflicting interests ?

Characteristics of Trusted Hardware

- Physically secure hardware component
  - Assumed not to break because it's hardware
- Environmental monitoring (temperature, power supply, structural integrity)
- Tamper responsive
- Implementations
  - CPU
  - ROM for OS and application code
  - Specialized hardware for cryptography and for storing secrets

Trusted Hardware – Example

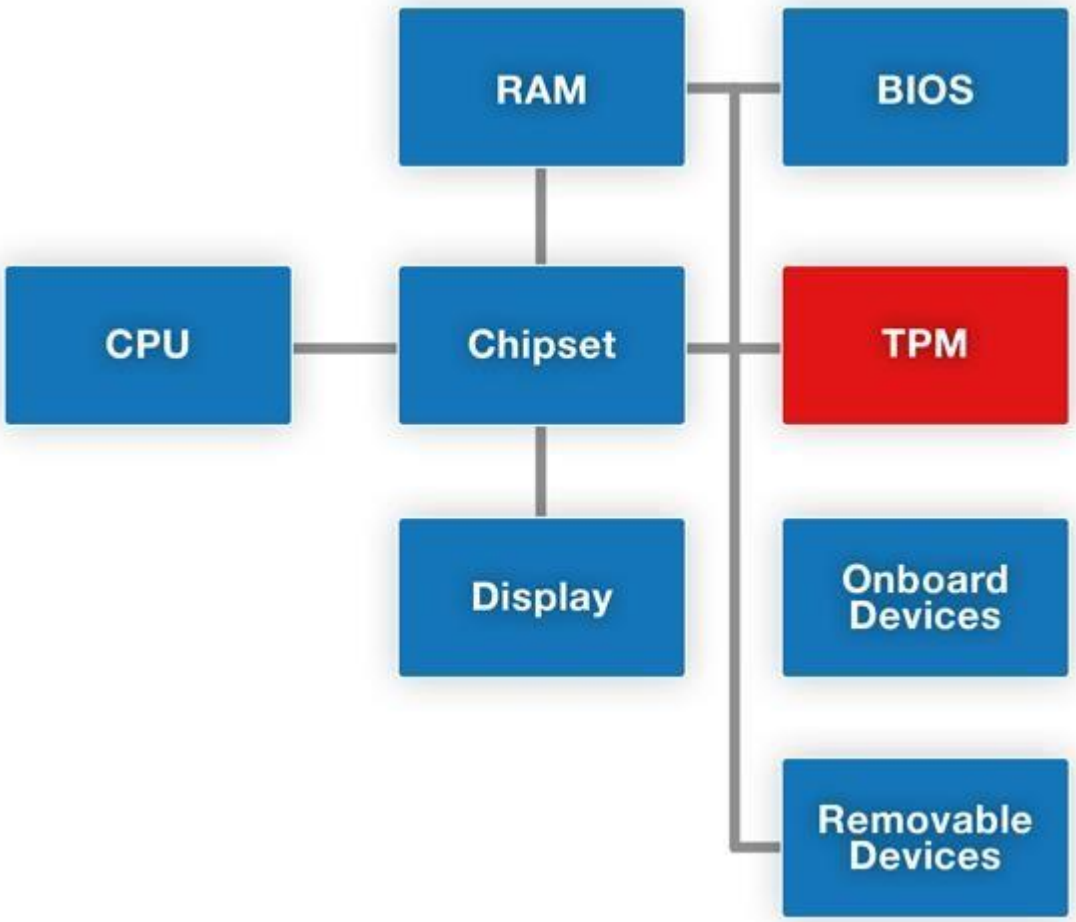
- IBM 4765 Secure Coprocessor



Trusted Computing Group
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
  - Industry standards organization
- 2003: TCPA TPM spec. adopted by TCG as TPM 1.2
- 2012: Draft TPM Specification 2.0 published
  - TPM 2.0 spec. not compatible with TPM 1.2 spec.
- 2015: Official TPM specification 2.0

Pervasiveness of the TPM



- The TPM chip sits on the motherboard
- Installed in 2 billion devices per 2015
- Relatively obscure technology for most people

TPM 1.2 Functionality

secured input - output

Cryptographic processor

- random number generator
- RSA key generator
- SHA-1 hash generator
- encryption-decryption-signature engine

Persistent memory

- Endorsement Key (EK)
- Storage Root Key (SRK)

Versatile memory

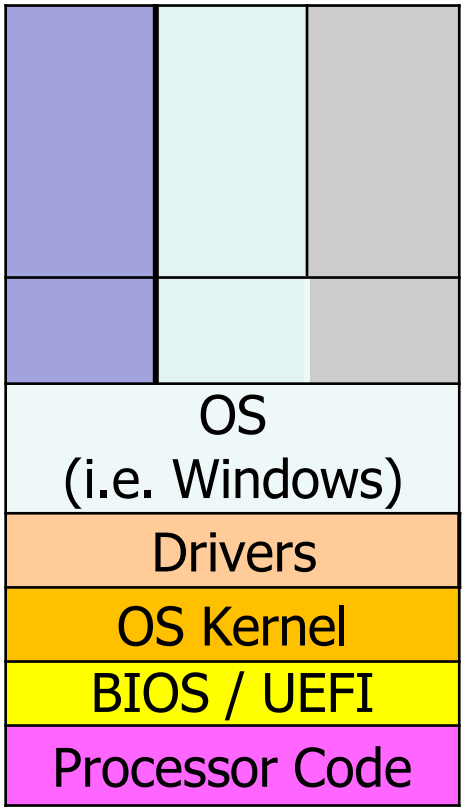
- Platform Configuration Registers (PCR)
- Attestation Identity Keys (AIK)
- storage keys

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. 2.0
  - Latest version of TPM spec. 2.0 is from 2015
- TPM chip mounted on motherboard,
- TPM equipped computing platforms
  - Laptops, servers, pads, mobile phones
- Used by software platforms
  - Windows Vista / 7 / 8 / 10, Linux, and MAC OS
- Supports 3 basic services:
  - Authenticated/measured boot,
  - Sealed Storage / Encryption
  - Remote attestation,

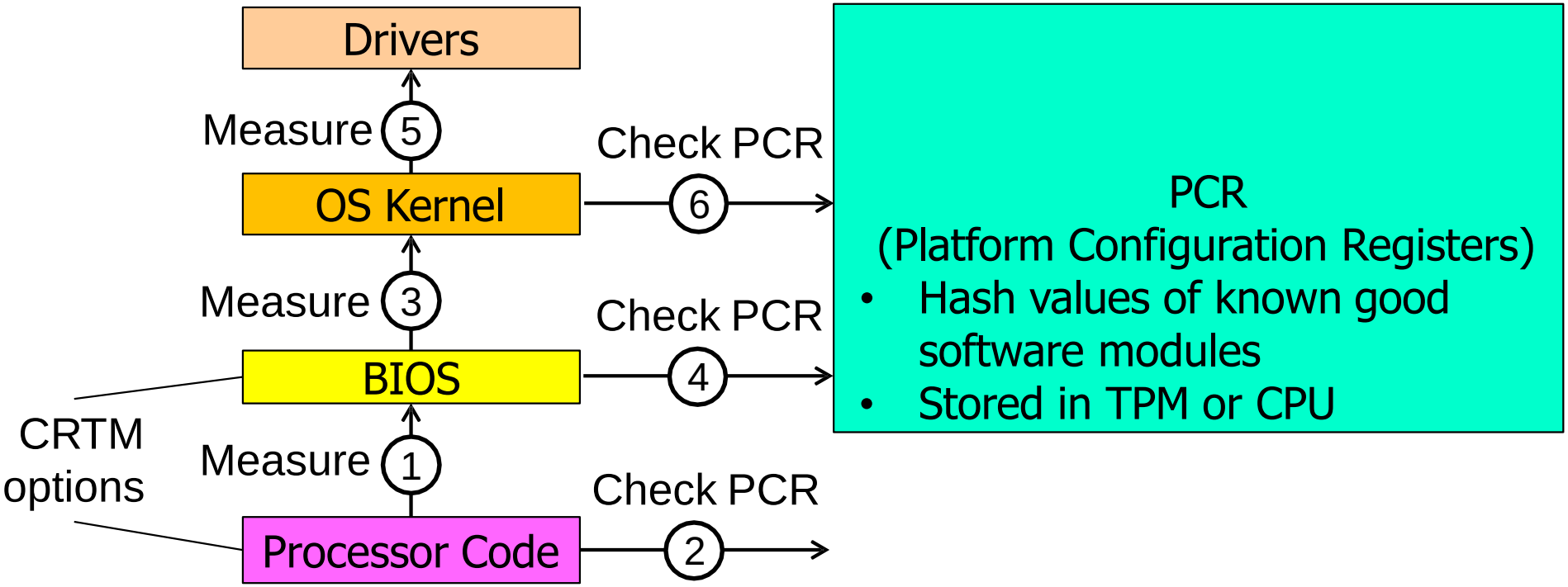
Boot protection

- BIOS /UEFI
  - Loaded by processor code at power-on
  - BIOS/UEFI initializes and identifies system devices such as display and keyboard
  - BIOS/UEFI then loads software (OS) held on a peripheral device such as a hard disk
  - BIOS/UEFI firmware is stored in ROM
- Boot protection
  - CRTM (Core Root-of-Trust for Measurement)
  - Boot protection focuses on verifying the integrity of the OS during boot.
  - CRTM can provide integrity assurance of OS
  - The usage of CRTM can be to check PCR or to check Digital Signatures of software.



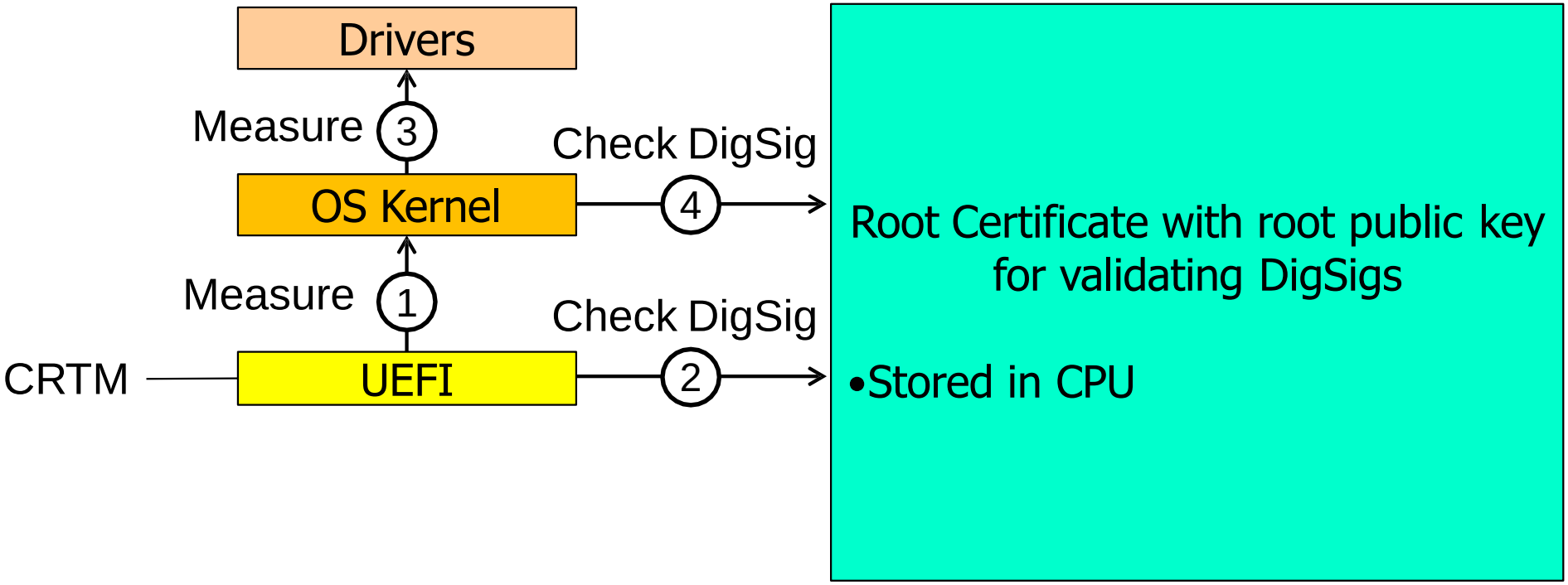
Boot Protection with TPM

- The CRTM is the initial code which starts running at CPU power-on.
- The boot integrity depends on the CRTM integrity.
- Multiple options for storing the CRTM, e.g. in CPU or external ROM
- Multiple options for storing PCR, e.g. in TPM or in other ROM chip.



Boot Protection with UEFI

- Unified Extensible Firmware Interface (replaces BIOS)
- The CRTM is the UEFI code which starts at CPU power-on.
- The boot integrity depends on the UEFI/CRTM integrity.
- The root public key must be stored in CPU.

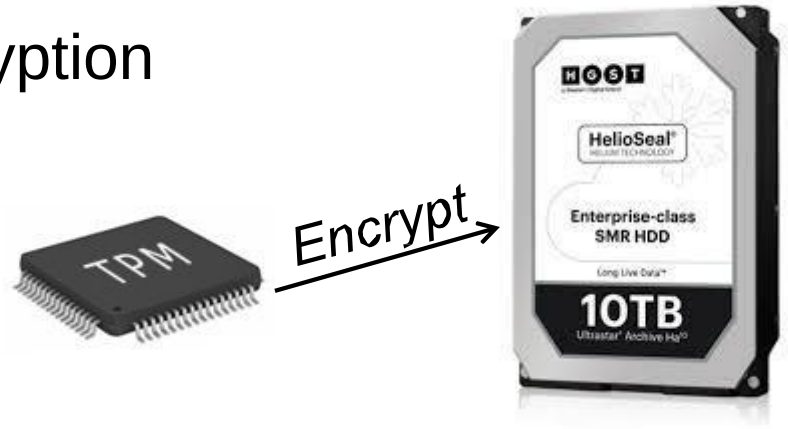


Two modes of boot protection

- Secure boot with UEFI (not with TPM, see UEFI later)
  - The platform owner can define expected (trusted) measurements (hash values) of OS software modules.
  - Hash values stored in memory signed by private PK (Platform Key).
  - Public PK stored in secure firmware on platform
  - Measured hash values can be compared with stored values.
  - Matching measurement values guarantee the integrity of the corresponding software modules.
  - Boot process terminates if a measurement does not match the stored value for that stage of the boot process.
- Authenticated/Measured boot with TPM
  - Records measured values in PCRs and reports to remote party
  - Does not terminate boot if measured values are wrong

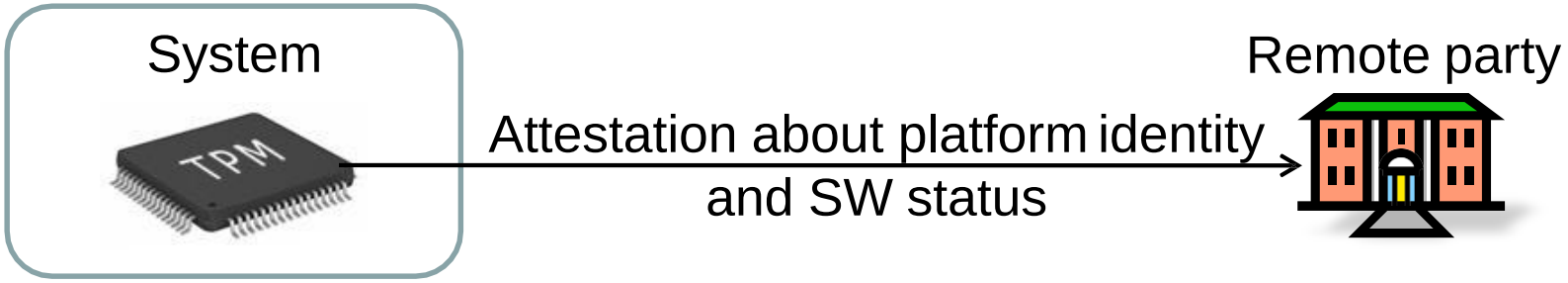
Sealed Storage / Encryption

- Encrypts data so it can be decrypted
  - by a certain machine in given configuration
- Depends on
  - Storage Root Key (SRK) unique to machine
  - Decryption only possible on unique machine
- 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 signature in configuration info
  - giving another user confidence in it
  - Based on Attestation Key (AK)
- Remote parties can validate signature based on a PKI
- Provides hierarchical certification approach
  - trust TPM, then trust the OS, then trust applications



End of lecture