



# Secure Software Systems

CYBR 200 | Fall 2018 | University of the Pacific | Jeff Shafer

## Architectural Approaches to Security

---

# Schedule

## This Week

- Tue September 11
- Thur September 13
- *Architectural Approaches to Security*

## Next Week

- Tue, September 18
- *Architectural Approaches to Security*
- *Project 1, Chapter 1 Due*

# Architectural Approaches to Security

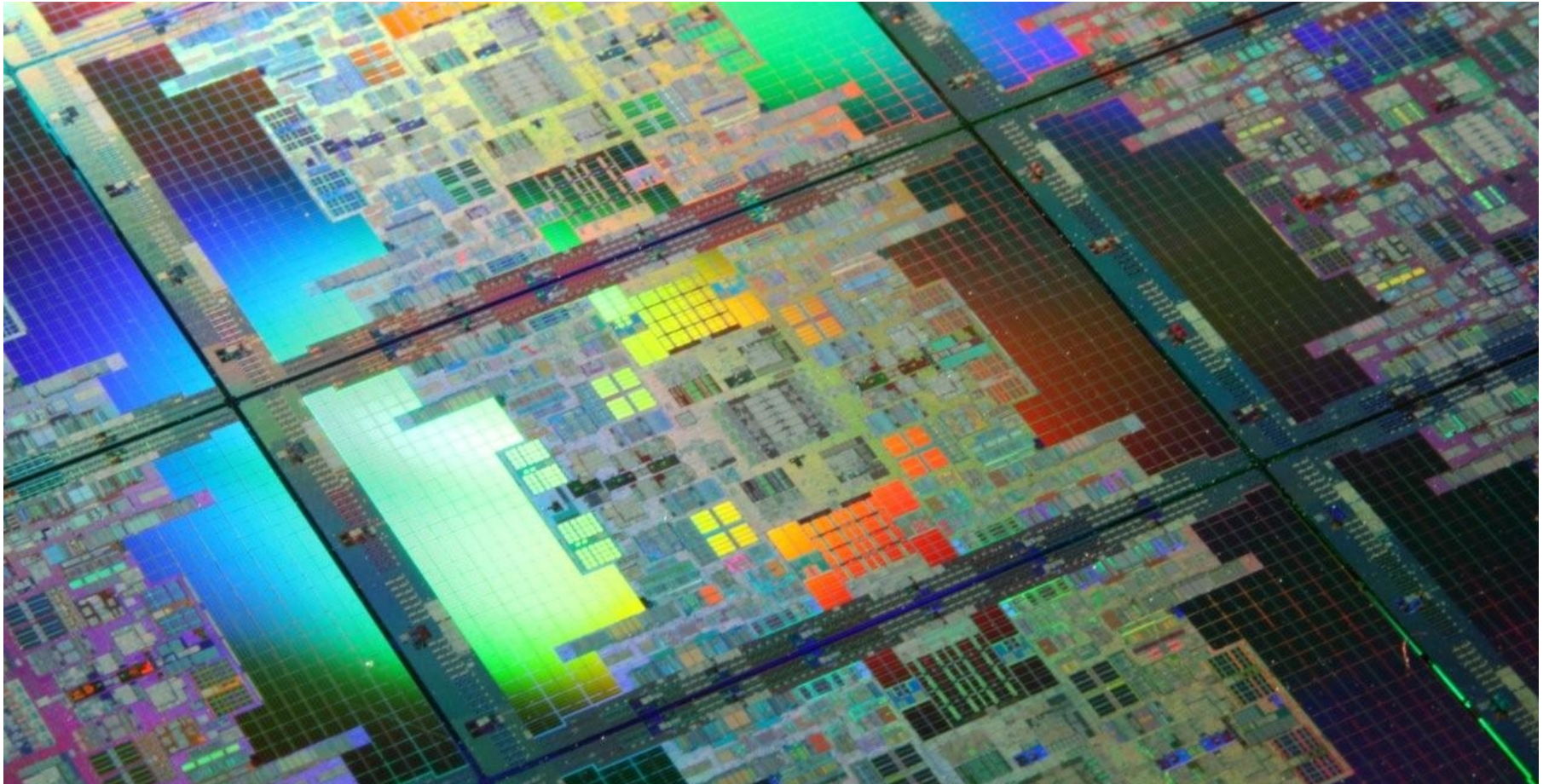
## ➤ Hardware

- Protection Rings
- Page level protection (Virtual Memory)
- Random Number Generation
- Cryptography instructions
- Management Engine
- Trusted Platform Module

## ➤ Software

- Virtualization
- Containers
- Sandboxing
- ASLR

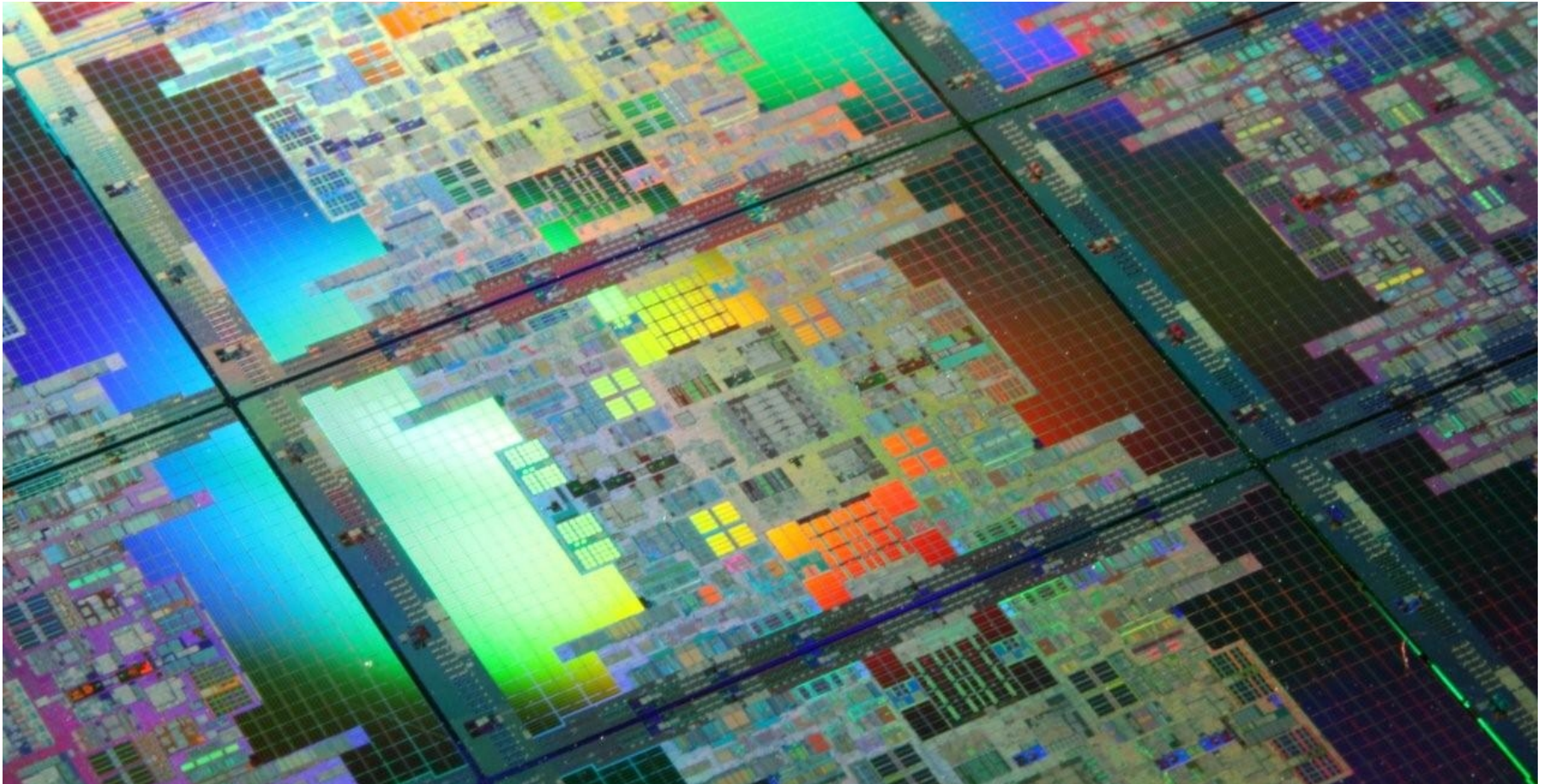
*Division between HW & SW is not this clear cut!  
Many techniques have elements of both*



# Hardware Security Mechanisms







# What can the Chip(s) do for Security?



# Hardware Security Mechanisms

**Motivating Question:** What protection/security mechanisms do modern hardware platforms (x86-64, ARM, ...) provide?

## ➤ Basic Features

- Protection Rings / Privilege Levels
- Page level protection (Virtual memory)

## ➤ Advanced Features

- Random number generation
- Crypto instructions
- Management Engine
- Trusted Platform Module
- Virtualization (*discuss in software section*)



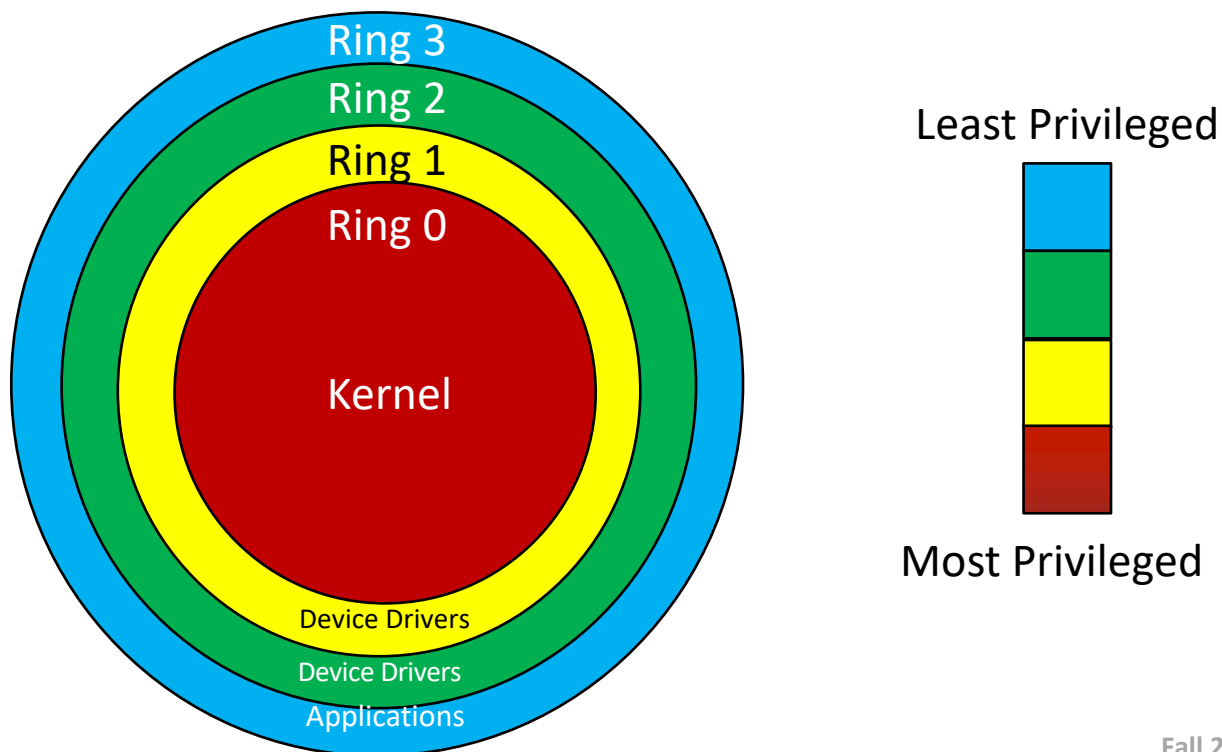


# Protection Rings



# Protection Rings

- Concept: Domains with varying privilege levels
- Privilege = ability to do actions



# Protection Rings

- Hardware protection mechanism
  - Ring 0 – most privileged
  - Ring  $n$  - least privileged
- Goals
  - Protect system integrity
  - Protect OS kernel from device drivers / services
  - Protect device drivers / services from applications
  - Etc...
- **Program can not call code of higher privilege directly**



# History of Protection Rings

- ***Multics* operating system**  
(Multiplexed Information and Computing Service)
  - Pioneering operating system released in 1969 – *vintage!*
- Honeywell 6180 mainframe supported 8 protection rings in hardware
  - 0 – supervisor
  - 1-3 – System levels
  - 4-7 – User levels, customized per application



# Protection Rings – x86-64

- Hardware rings
  - Ring 0 – Kernel mode (operating system)
  - Ring 1 – Privileged mode (device drivers)
    - Only present in x86, **removed** in x86-64
  - Ring 2 – Privileged mode
    - Only present in x86, **removed** in x86-64
  - Ring 3 – User Mode (applications)
- Restrictions based on protection ring
  - Which assembly instructions (or instruction options) are available?
  - What memory addresses can I read/write?
  - What I/O ports can I read/write?
  - What registers can I read/write?
  - ...

# x86-64 Q&A

- **Which ring would be part of the trusted computing base? (TCB)**
  - Ring 0 (kernel mode)

# x86-64 Q&A

- **Q:** How does software know its current ring level?
  - **A:** CS (code selector) register indicates current privilege level
- **Q:** How does user code get the attention of system code?
  - **A:** Code cannot call code of higher privilege directly. Instead must use system calls (aka “gates”)
    - SYSENTER / SYSEXIT (32-bit x86)
    - SYSCALL / SYSRET (64-bit x86-64)


# x86-64 Q&A

➤ **Q:** What instructions only work in ring-0?

➤ **A:**

- HLT – Halt CPU till next interrupt
- INVLPG – Invalidate page entry in TLB
- LIDT – Load Interrupt Descriptor Table
- LGTD – Load Global Descriptor Table
- RDMSR / WRMSR – Read/Write Model Specific Register
- MOV CR – Load or store control registers
- VMENTER / VMEXIT – Enter/exit hypervisor



- 
- **Q:** What can't a process do in user mode? (ring-3)
  - **A:** Open files, send/receive network packets, print to the screen, allocate memory, etc...

# Protection Rings – ARM

- ARM doesn't have protection rings, but does have “privilege levels” or “exception levels”
- ARMv7 privilege levels
  - User/Application (PL0)
  - Operating System (PL1)
  - Hypervisor (PL2)
- ARMv8 exception levels
  - User/Application (EL0)
  - Operating System (EL1)
  - Hypervisor (EL2)
- Restrictions based on privilege levels
  - Which assembly instructions (or instruction options) are available?
  - What memory addresses can I access?
  - What I/O ports can I access?
  - ...

# Page Level Protection



# Uniprogramming

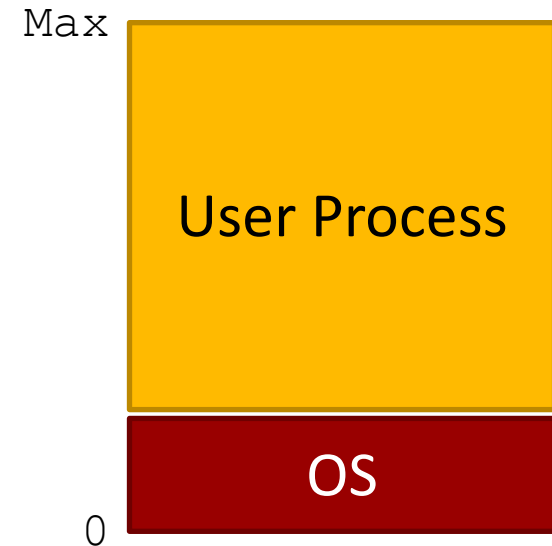
## ➤ Back in the dark ages: **Uniprogramming**

- Single user process running on computer 😞
- User process can destroy OS (read/write OS memory) 😞

## ➤ **Multiprogramming**

- Multiple user processes 😊
- Protection 😊

## ➤ Uniprogramming physical memory map:



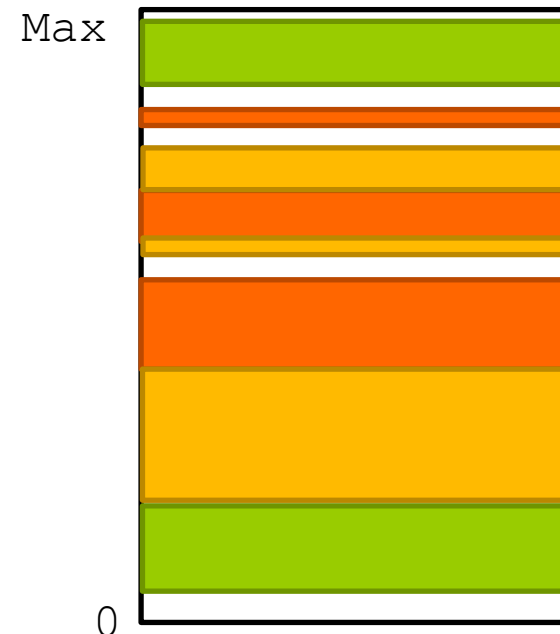
# Multiprogramming

**Multiple address spaces exist on same computer system**

**Logical View**



**Physical View**





# Multiprogramming Design Requirements

- Sharing
  - Multiple processes coexist in main memory
- Transparency
  - Processes are not aware that memory is shared
  - Processes can run regardless of the number or locations in physical memory of other processes
- Protection
  - Cannot access data of OS or other processes
- Performance
  - Fast (accelerated by hardware?)
  - Does not waste memory (fragmentation = bad)

# Multiprogramming Design: Segmentation

➤ Segmentation is a system where....

# Skip!

*Used on Intel 8086 and 80286 processors*

*(Know that there was a legacy design, and the legacy design still has traces in modern design/terminology)*

- |        |        |        |        |        |        |        |        |
|--------|--------|--------|--------|--------|--------|--------|--------|
|        |        |        |        |        |        |        |        |
| Yellow | Yellow | Yellow | Green  | Green  | Green  | Green  | Orange |
|        | Yellow | Yellow |        |        |        |        |        |
|        | Orange |        |        |        |        |        |        |
|        |        |        | Orange | Orange |        |        |        |
|        |        |        |        |        |        |        |        |
|        | Green  | Green  |        | Yellow | Yellow |        |        |
|        |        | Green  |        |        | Yellow | Yellow |        |
|        |        |        |        |        |        |        |        |
|        |        |        | Yellow |        |        |        |        |
|        | Orange | Yellow | Yellow | Yellow | Orange |        |        |
|        |        |        |        |        |        |        |        |
|        |        |        | Orange |        |        |        |        |
|        |        |        |        |        |        |        |        |
|        |        | Green  | Green  | Green  | Green  | Green  | Green  |
| Green  | Green  |        |        |        |        |        | Yellow |

# Paging

- Paging optimizations?
  - Super page / huge page / large page
  - 2MB or 1GB in x86-64
  - 64kB, 1MB, 16MB in ARMv7
  - Reduces amount of translation metadata needed

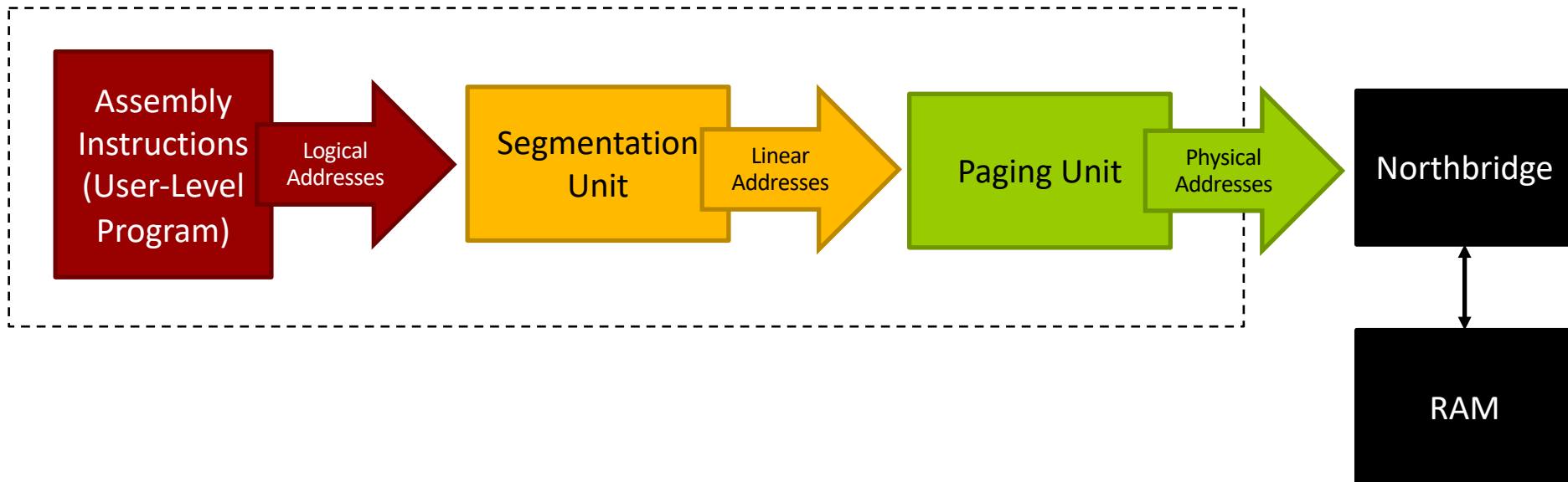
# Memory Address Translation

- Map program-generated address (*virtual address*) to hardware address (*physical address*)
  - Done dynamically at runtime
  - Done at the assembly code level
  - Done at every memory access (read or write)
- Accelerated by hardware (e.g. *Translation Lookaside Buffer* – TLB) using data structures that are managed by operating system
  - Hardware + Software = Virtual Memory system

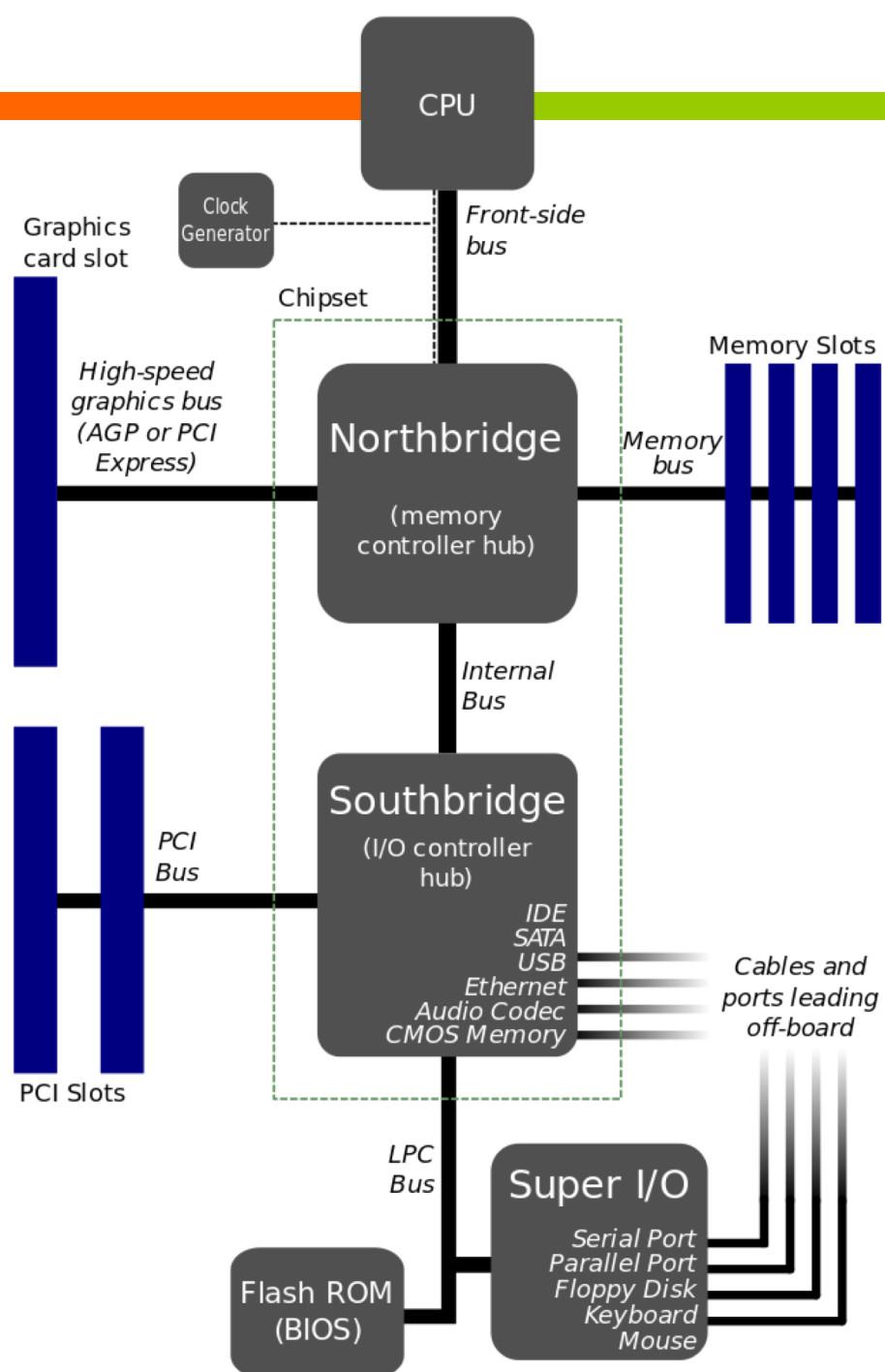


# Memory Address Translation

x86 CPU with paging enabled

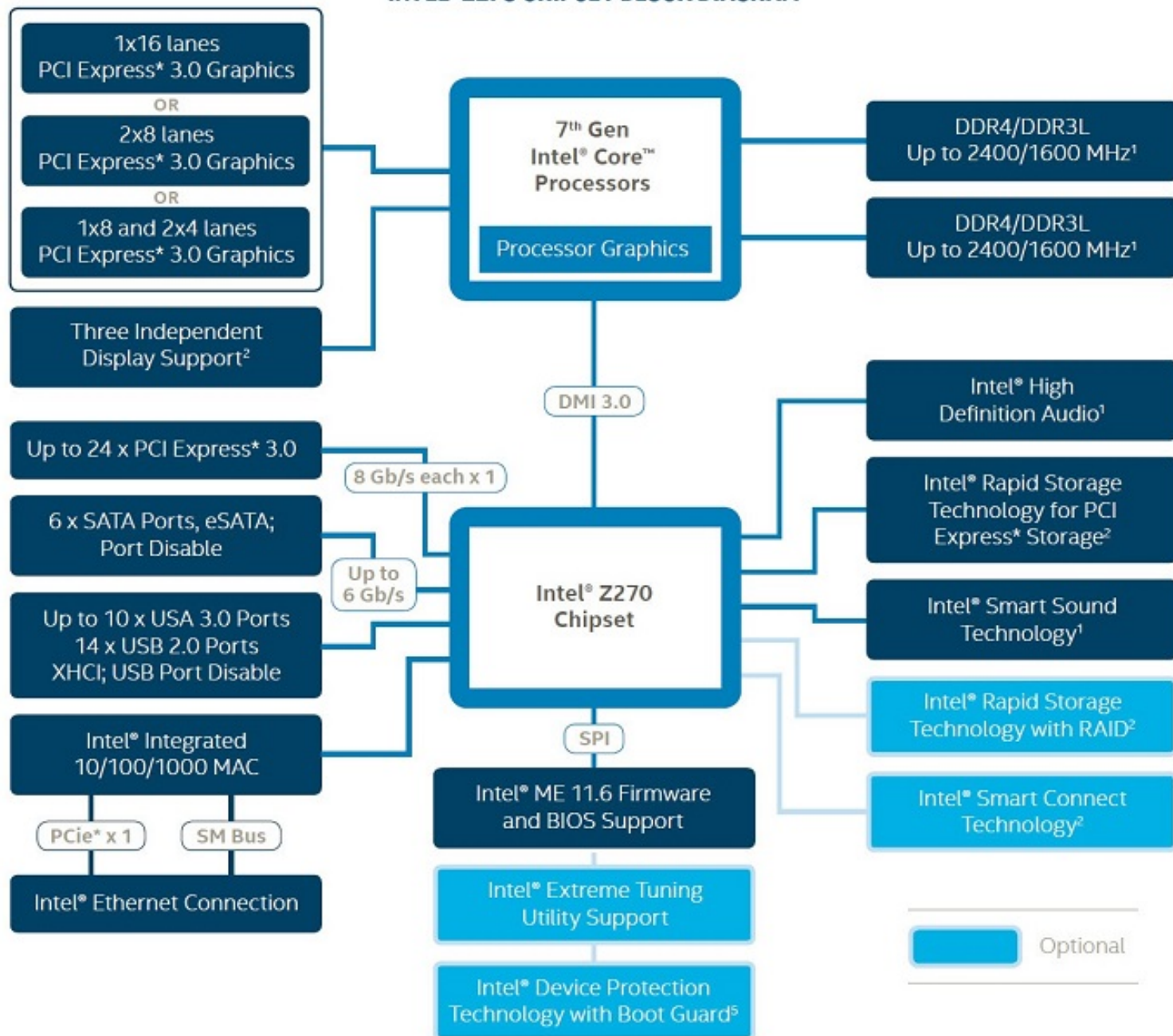


<http://duartes.org/gustavo/blog/post/memory-translation-and-segmentation/>



# INTEL® Z270 CHIPSET BLOCK DIAGRAM

27



# Memory Address Translation

- Page Table
  - Data structure storing virtual address->physical address mapping
  - Consulted on every memory access!
  - Not one table, but a hierarchical sequence of tables
- Translation Lookaside Buffer (TLB)
  - Hardware cache to accelerate page table lookups

# Security

- **Q:** How does this impact security?
- **A:** Process X will *only*\* be able to get virtual addresses for memory assigned to it!
  - Hardware will translate these virtual addresses to physical addresses that are only assigned to that process
- (\*) Process can ask operating system for greater memory access, e.g. creating a memory region that can be shared between multiple processes

# Security

**Protection bits in page table:** Checked by hardware (MMU) on each memory access

- Present bit – Is there a mapping to a valid physical page?
- Read/write/execute bits
- User/Supervisor bit – Can this page be accessed in user mode? (Ring 3)
  - User process can not read kernel memory

<http://hypervsir.blogspot.com/2014/10/introduction-on-hardware-security.html>

# Security

**Protection bits in page table:** Checked by hardware (MMU) on each memory access

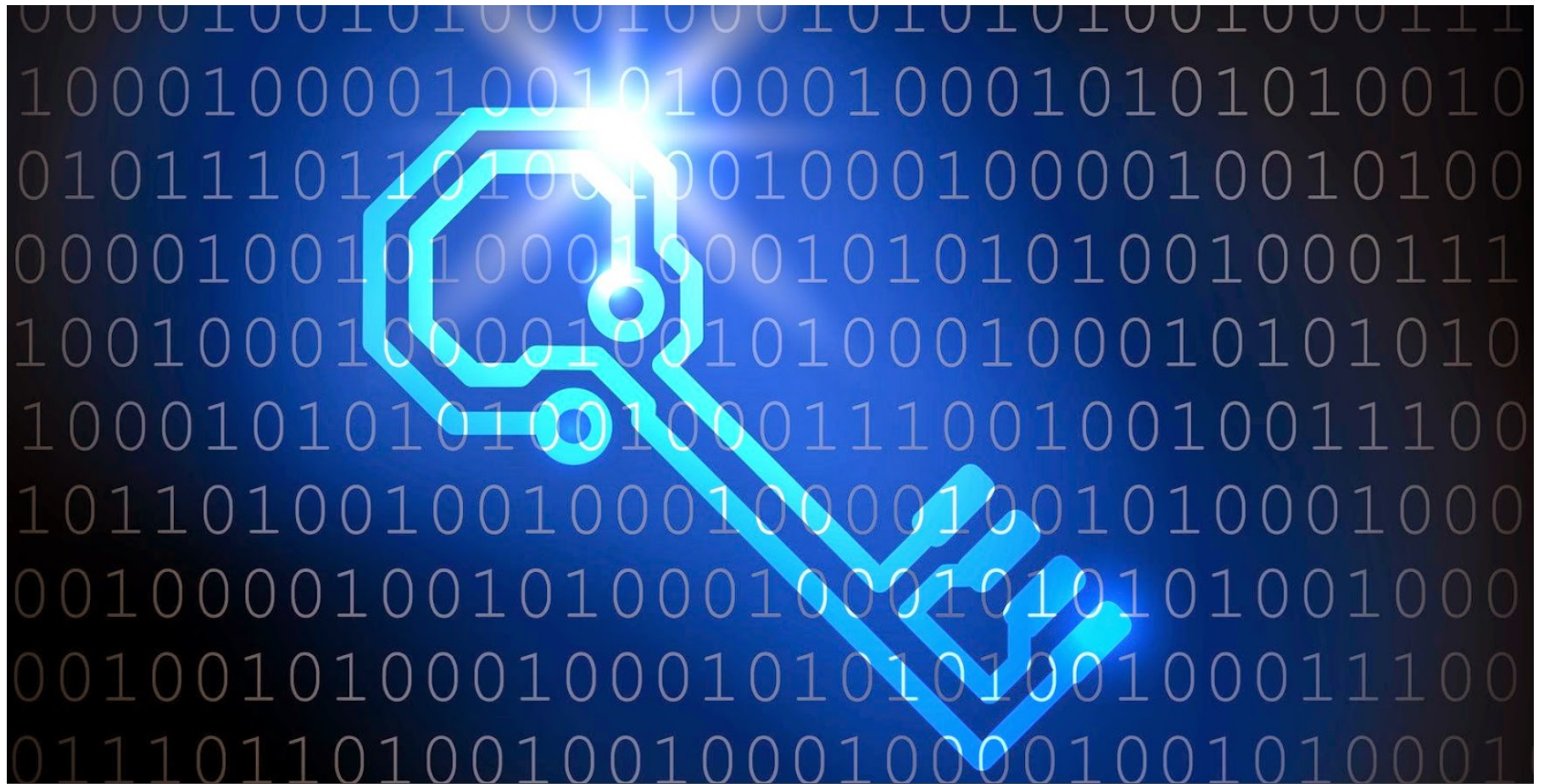
- **XD** – eXecute Disable / **NX** – No-eXecute (Intel)  
**XN** – eXecute-Never (ARM)
- Can we execute code from this page? (or is it only data?)
- Aka Windows Data Execution Prevention (DEP)
- Useful to stop buffer overflows turning (easily) into exploits

# Security

**Protection bits in page table:** Checked by hardware (MMU) on each memory access

- **SMAP** – Supervisor Mode Access Protection (Intel)
- SMEP** – Supervisor Mode Execution Protection (Intel)
- PAN** – Privileged Access Never (ARM)
- PXN** – Privileged eXecute Never (ARM)
  - Can be used to block access or execution to user-space pages while running in privileged mode (i.e. restrict what the kernel can do!)
  - **Q:** Why do we want to limit the kernel?
  - **Ans:** What if the attack fills a user-space region with malicious code and tricks the kernel into accessing a pointer to it?





# Random Number Generation



# Random Number Generation

**Motivating Question:** How do I get a **good** random number when computer hardware is deterministic?

- Will talk *at length* about importance of random number (entropy) generation, and many potential methods, in cryptography section of course
- Discuss hardware method today

# Random Number Generators

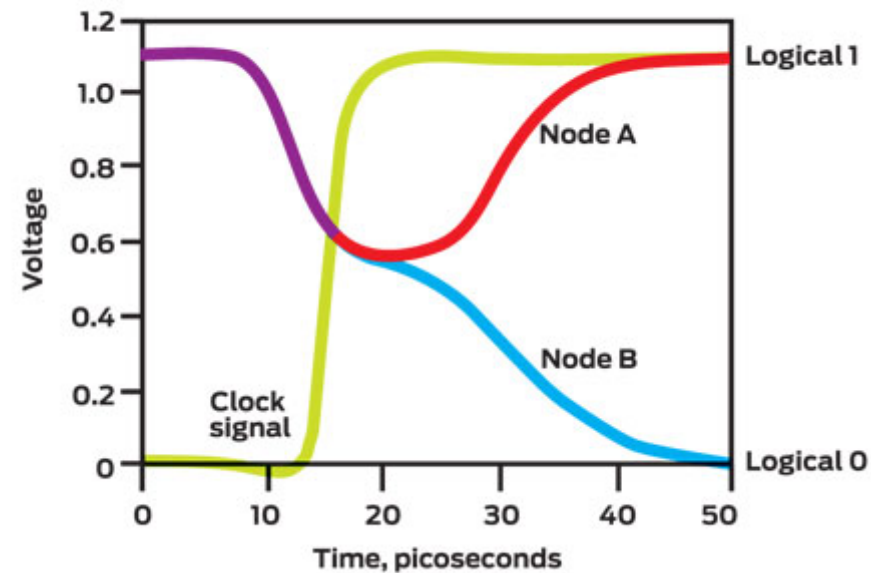
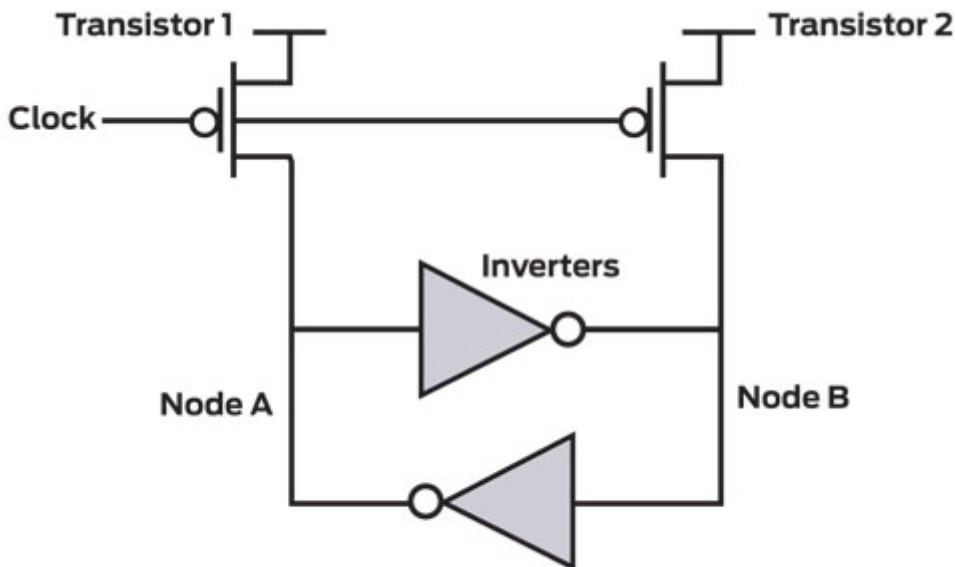
- Produce a sequence of numbers with following properties:
  - New value must be *statistically independent* of previous value
    - Particular values should not be more or less likely
  - Distribution of numbers is *uniformly distributed*
    - Cannot have some values more or less likely
  - Sequence is *unpredictable*
    - Cannot guess next value based on current or past values
    - Cannot guess previous values based on current value

# x86 Randomness

- Other desired features for a hardware implementation?
  - Fast! (can produce many random numbers quickly)
  - Secure against attackers (cannot observe/modify underlying state)
- Implementations
  - Intel Digital Random Number Generator (DRNG)
    - [https://software.intel.com/sites/default/files/m/d/4/1/d/8/441\\_Intel\\_R\\_DRNG\\_Software\\_Implementation\\_Guide\\_final\\_Aug7.pdf](https://software.intel.com/sites/default/files/m/d/4/1/d/8/441_Intel_R_DRNG_Software_Implementation_Guide_final_Aug7.pdf)
    - Introduced in Ivy Bridge architecture
  - AMD Secure Random Number Generator (RNG)

# Intel DRNG

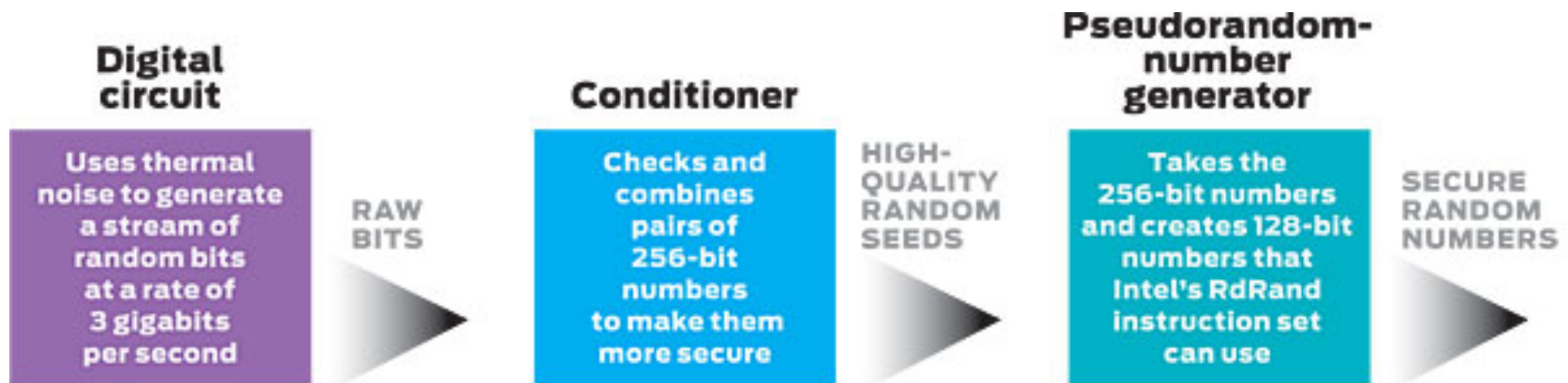
- Idea: Break key digital design rule of “Circuit should always be in known state”. *Metastability* is a feature!
- **What happens when transistors are ON? And then OFF?**



# Intel DRNG

- Hardware produces 1 random bit per clock cycle
- Scale up with **parallel circuits** for multiple random bits per cycle
- Q: Is this sufficient?
- A: No – each inverter circuit is not identical when fabricated at scale
  - What if some prefer 0's more than 1's?
    - Bias! ☹ Lacking a uniform distribution
  - Need a solution that can be certified as meeting NIST standards

# Intel DRNG



- <https://spectrum.ieee.org/computing/hardware/behind-intels-new-randomnumber-generator>
- [https://software.intel.com/sites/default/files/m/d/4/1/d/8/441 Intel R DRNG Software Implementation Guide final Aug7.pdf](https://software.intel.com/sites/default/files/m/d/4/1/d/8/441_Intel_R_DRNG_Software_Implementation_Guide_final_Aug7.pdf)

# Intel DRNG

- RDSEED instruction
- Purpose: Seed a software PRNG (pseudo-random number generator) with arbitrary-length stream of bits
- Produced by conditioner circuit (not directly from raw inverter circuits that toggle based on thermal noise)
- Slower than next instruction, RDRAND



# Intel DRNG

- RDRAND instruction
- Purpose: Generate random 16, 32, or 64-bit number for use in software applications
- Warning: Instruction might (rarely!) fail if random value not available— Must check the CF (carry flag) to make verify result before using. Read the docs!
  - Any wrapped API on this instruction would handle retrying automatically for you

# Intel DRNG

## ➤ RDRAND generation method

1. Hardware entropy source (2 256-bit numbers)
2. Advanced Encryption Standard (AES) conditioner produces single 256-bit entropy sample
3. *Deterministic* random bit generator produces samples for RDRAND from hardware seed
  1. This allows RDRAND to be much faster than the underlying hardware generator, which runs at fixed rate
  2. Hardware seed replaced every 511 sample

# Trust



- Risk – Do we trust the hardware implementation? (black box, impossible to audit/verify)
  - Critical part of *trusted computing base*
  - Serious concerns in Linux, FreeBSD development community
  - Are we *sure* that the NSA hasn't influenced these hardware designs?
    - Are conspiracy theorists paranoid or prescient?
- Consensus – Hardware entropy should not be the *only* source of randomness in the system
  - `/dev/random` in Linux

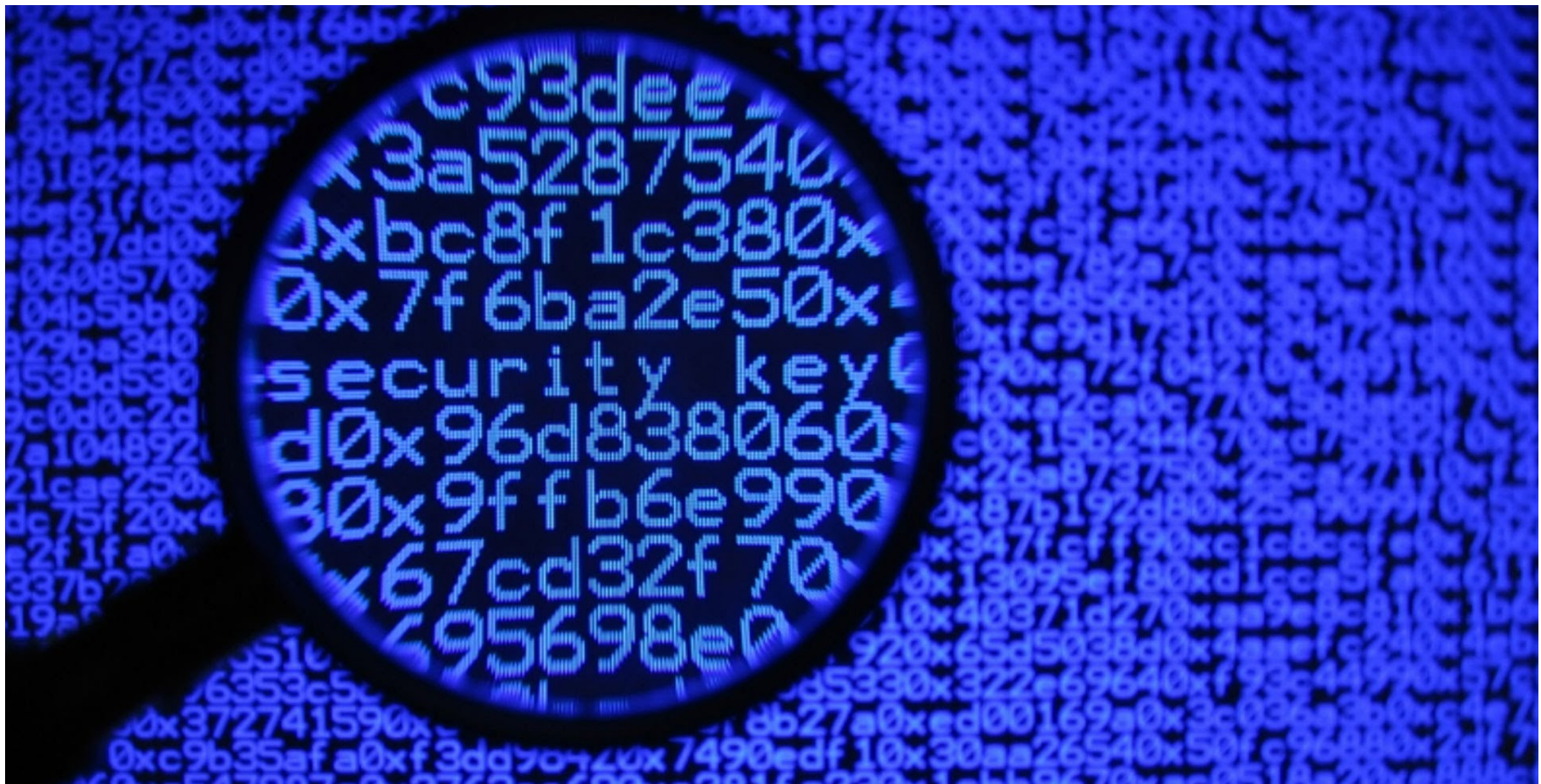
# Aside: Linux /dev/random

- **Q:** Other sources of entropy for Linux /dev/random?
- **A:** Kernel has variety of “environmental noise” factors that are mixed together. SHA hash of “entropy pool”
  - Inter-interrupt timings  
`add_interrupt_randomness()`
  - Input layer interrupt timing (e.g. inter-keyboard)  
`add_input_randomness()`
  - Disk seek time per-disk/per-request – not good for SSDs!  
`add_disk_randomness()`
  - Serial numbers / Ethernet MAC addresses – only for initialization of pool! `add_device_randomness()`

<https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/drivers/char/random.c>

# Assurance – Intel DRNG

1. **Compliant with Standards** *(Who verified this?)*
  1. **NIST SP 800-90A** (2015) - “Recommendation for Random Number Generation Using Deterministic Random Bit Generators”
  2. **FIPS 140-2** (2002) – “Security Requirements for Cryptographic Modules” NIST standard for cryptography modules with HW & SW components
  3. **ANSI X9.82 (2006)** – “Random Number Generation”
2. **Third-party Review**
  1. Jun, Benjamin; Kocher, Paul (1999-04-22). "The Intel Random Number Generator". Cryptography Research, Inc.  
<http://www.cryptography.com/public/pdf/IntelRNG.pdf>
  2. Hamburg, Mike; Kocher, Paul; Marson, Mark (2012-03-12). "Analysis of Intel's Ivy Bridge Digital Random Number Generator". Cryptography Research, Inc.  
[https://www.cryptography.com/public/pdf/Intel\\_TRNG\\_Report\\_20120312.pdf](https://www.cryptography.com/public/pdf/Intel_TRNG_Report_20120312.pdf)



# Cryptography Instructions



# Cryptography Instructions

- *Whines.* “Cryptography is slow – can’t we make it faster?”



# Cryptography Instructions

## ➤ AES Instruction Set

- Assembly instructions added in 2008 by Intel and AMD
- Accelerate AES (Advanced Encryption Standard) encryption and decryption
- Similar functionality in ARM via *Security System*

AESENC, AESENCLAST, AESDEC,  
AESDECLAST, AESKEYGENASSIST,  
AESIMC, PCLMULQDQ



# Cryptography Instructions

## ➤ SHA Extensions

- Assembly instructions added in 2016 by Intel and AMD
- Secure Hash Algorithm (SHA)
- Accelerate SHA-1 and SHA-256 calculations

```
SHA1RND4, SHA1NEXTE, SHA1MSG1, SHA1MSG2  
SHA256RND2, SHA256MSG1, SHA256MSG2
```

# Cryptography Instructions

- **Q:** Should I be writing inline assembly code to use these instructions?
- **A: No!** Your crypto library (*which you didn't write yourself*) should use these acceleration techniques
  - But you may want to consider availability of hardware acceleration and performance when weighing tradeoffs between crypto algorithms

# Trusted Computing



# Attacks

- Many attacks are challenging for conventional software to detect & address
- Unauthorized software running on computer
  - Root kits / Boot sector / BIOS virus
- Theft of cryptographic keys
  - If I have read access to your disk, I can steal your private keys
  - Impersonation attacks?
  - Data theft?
- Users behaving badly (or software running under their privileges)
  - Super-user / abuse of privilege attacks

# Trusted Computing

- Goal: Build computer systems that
  - Strongly identify themselves (uniquely)
  - Strongly identify their current configuration and running software
- Identity
  - Hardware identity will be based on public-key cryptography
  - Software identity will be based on cryptographic hashes of program byte code

# Trustworthy State

- Take a measurement (cryptographic hash) of each component that contributes to platform state
  - Firmware? Kernel? Library? Application binary? Configuration file?
- Administrator can use measurements to decide if platform is in a trustworthy state
  - Same state as last boot time?
  - Using components without known vulnerabilities?
  - Using components approved by administrator?
- Obtain measurements in hardware (which is *hard to alter*)

# Trusted Computing

- 'Trusted Computing' developed by the Trusted Computing Group
  - Founding members: Microsoft, HP, IBM, Intel, AMD
  - Current members: Cisco, Lenovo, Infineon, Juniper Networks, many more...
  - <https://trustedcomputinggroup.org/>
- Proposed six technology *concepts* in 2001 v1.0 spec release
  1. Endorsement key
  2. Secure input & output
  3. Memory curtaining / protected execution
  4. Sealed storage
  5. Remote attestation
  6. Trusted Third Party (TPP)

# Trusted Computing

## Endorsement Key

- 2048-bit RSA public/private key pair created at manufacture time
- Fixed in hardware, never leaves the chip

## Secure Input & Output

- Control where data (audio, video, files...) are sent to



# Trusted Computing

## Memory Curtaining

- Provides full isolation of sensitive memory areas (e.g. locations of cryptographic keys)
- Operating System access to curtained memory is limited

## Sealed Storage

- Protects private information by binding it to platform configuration information (i.e. hardware and software being used)
  - e.g. can only decrypt files on specific hardware devices

# Trusted Computing

## Remote Attestation

- Verifies to remote parties that specific software is running on computer
  - And that software is not tampered with

## Trusted Third Party

- Alice wants to assure Bob that she is running un-tampered hardware and software
  - But she doesn't want to reveal her unique identifying information to Bob (i.e. wants anonymity)
- Idea: Trusted Third Party to vouch for Alice

# Use Case – Authenticated Boot

- Keep a tamper-evident log of boot process
- Power on
  - Compute cryptographic hash of boot ROM, write to log, run boot ROM
  - Compute cryptographic hash of next stage of boot process, write to log, run stage
  - Repeat until full OS is loaded
- Log provides history of *exactly* what software was loaded on the machine

# Use Case – Digital Rights Management

- *Sealed storage* prevents using from opening file with unauthorized computer
- *Remote attestation* ensures that only media company approved software players are loaded
- *Curtained memory* ensures that decrypted media file cannot be copied out of memory
- *Secure Input/Output* ensures copy of decrypted file cannot be captured from audio/video devices

# Controversy

- Hardware is not only secured *for* its owner, but also secured *against* its owner.
- “Treacherous Computing” –  
Richard Stallman (GNU, Free Software Foundation)  
<https://www.gnu.org/philosophy/can-you-trust.html>
- Examples (circa 2003)
  - Can “Internet Explorer-only” websites force you to attest that you are running IE?
  - Can Microsoft file sharing servers force you to attest that you are running MS clients, not open source Samba clients?
  - Can you trust the hardware vendors providing the trusted computing platform?

# Controversy

- Microsoft *Palladium* (aka Next-Generation Secure Computing Base)
  - Parallel Windows architecture to add trusted mode / secure paths alongside untrusted legacy paths. Announced in 2002
- Consumer reaction was “mixed”
  - A plot to take over cyberspace?
  - A plot to keep users from running any software not personally approved by Bill Gates?
  - <http://www.cl.cam.ac.uk/~rja14/tcpa-faq.html>
- Developed over a decade, but cancelled prior to release of Windows Vista
- Surviving elements
  - BitLocker disk encryption
  - Measured Boot (Windows 8)
  - Device Guard (Windows 10)

# Modern Computing

- **So is the trusted computing mindset dead?**
- Can I run any OS I want on my iPhone or iPad?
  - How is this enforced?
- Can I run any application I want on my iPhone?
  - How is this enforced?
- Can't run Linux and OpenOffice on iPad hardware

# Trusted Platform Module





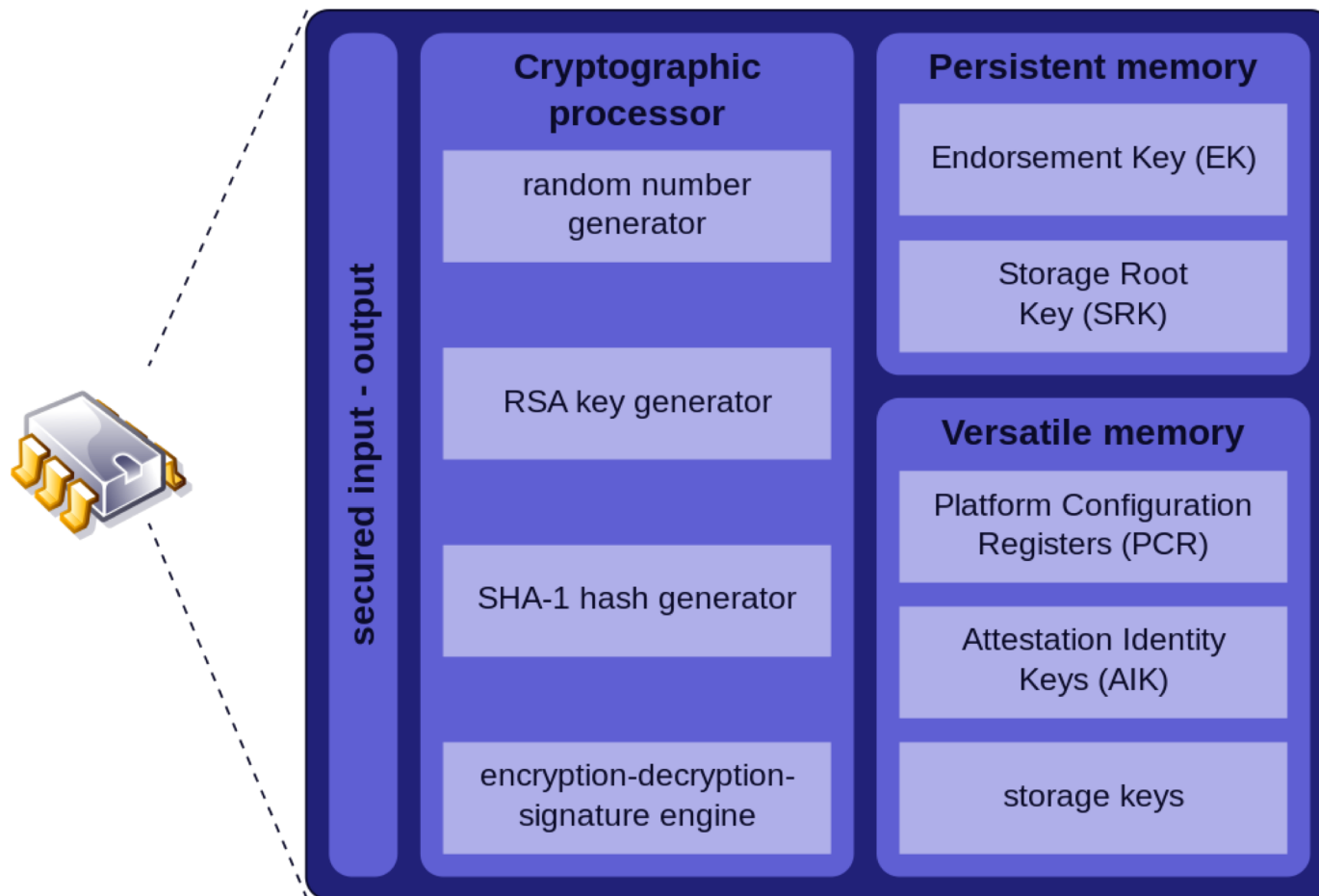
# Trusted Platform Module

- Key security challenge: knowing exactly what software is running now (so that computing platform is in *defined state*)
  - Must monitor boot process
  - Must provide an anchor for “Root of Trust”
  - Must have safe place to take measurements from
  - Must be able to report measurements to 3<sup>rd</sup> party (attestation)

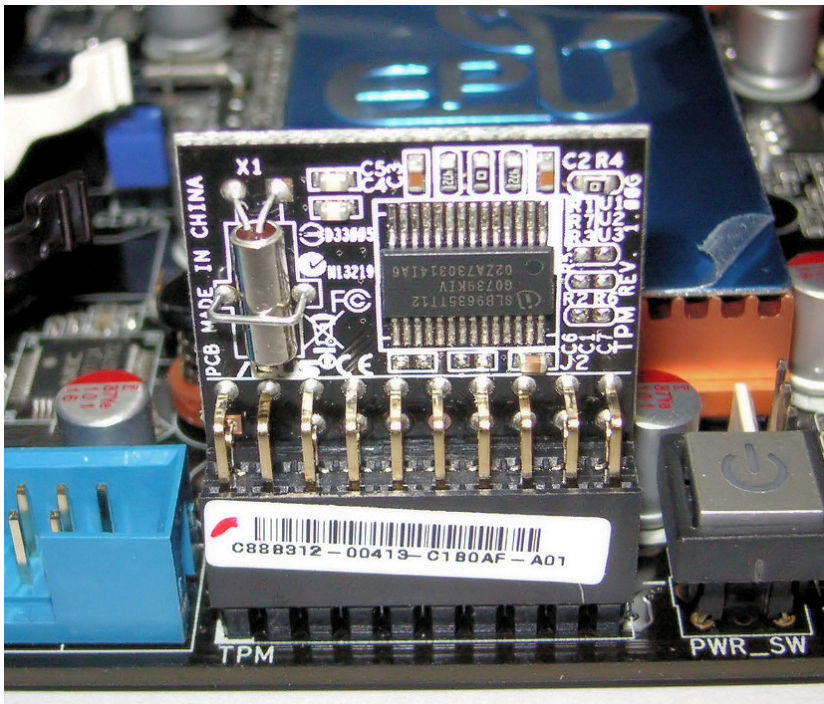
# Trusted Platform Module

- Current implementation of Trusted Computing concepts
  - Standardized in 2009 – ISO/IEC 11889
  - Updated in 2014 (v2.0)
- Ideally a dedicated microcontroller (e.g. hardware)
- Features
  - Random number generator
  - Cryptographic key generator
  - Remote attestation of HW/SW
  - Binding / Sealing data

# Trusted Platform Module v1.2



# Implementations



- Full hardware
  - Discrete TPM
  - Integrated TPM
- Software
  - Firmware TPM
  - Software TPM
  - Virtual TPM
- Confidence may vary with implementation type

# Misconceptions

- TPM does not measure, monitor, or control anything
  - Passive component in system
  - Measurements are made by host software and sent to TPM
  - Cannot alter execution flow of system
- TPM does not perform bulk encryption (e.g. full disk encryption)
- Open specification / Open API
  - Trusted Computing Group

# Applications

## ➤ Google Chromebooks

- Verified Boot – provides cryptographic assurance that only Google-approved code is running
- Encrypted store – each user of Chromebook has private encrypted data store – keys are protected by TPM
- <https://chrome.googleblog.com/2011/07/chromebook-security-browsing-more.html>

# Applications

- iPhones / iPads / TouchID
  - Apple uses a “Secure Enclave” co-processor for security critical functions
  - Similar in concept to TPM, but proprietary

# Management Engines





# How do you keep your computer secure?

- Key tip: Run the latest version of software (with all security fixes)
  - You might only have *days* (hours?) between a security bug being fixed and attackers actively exploiting on the Internet
- Keep Windows up-to-date
- Keep Office up-to-date
- Keep Chrome up-to-date

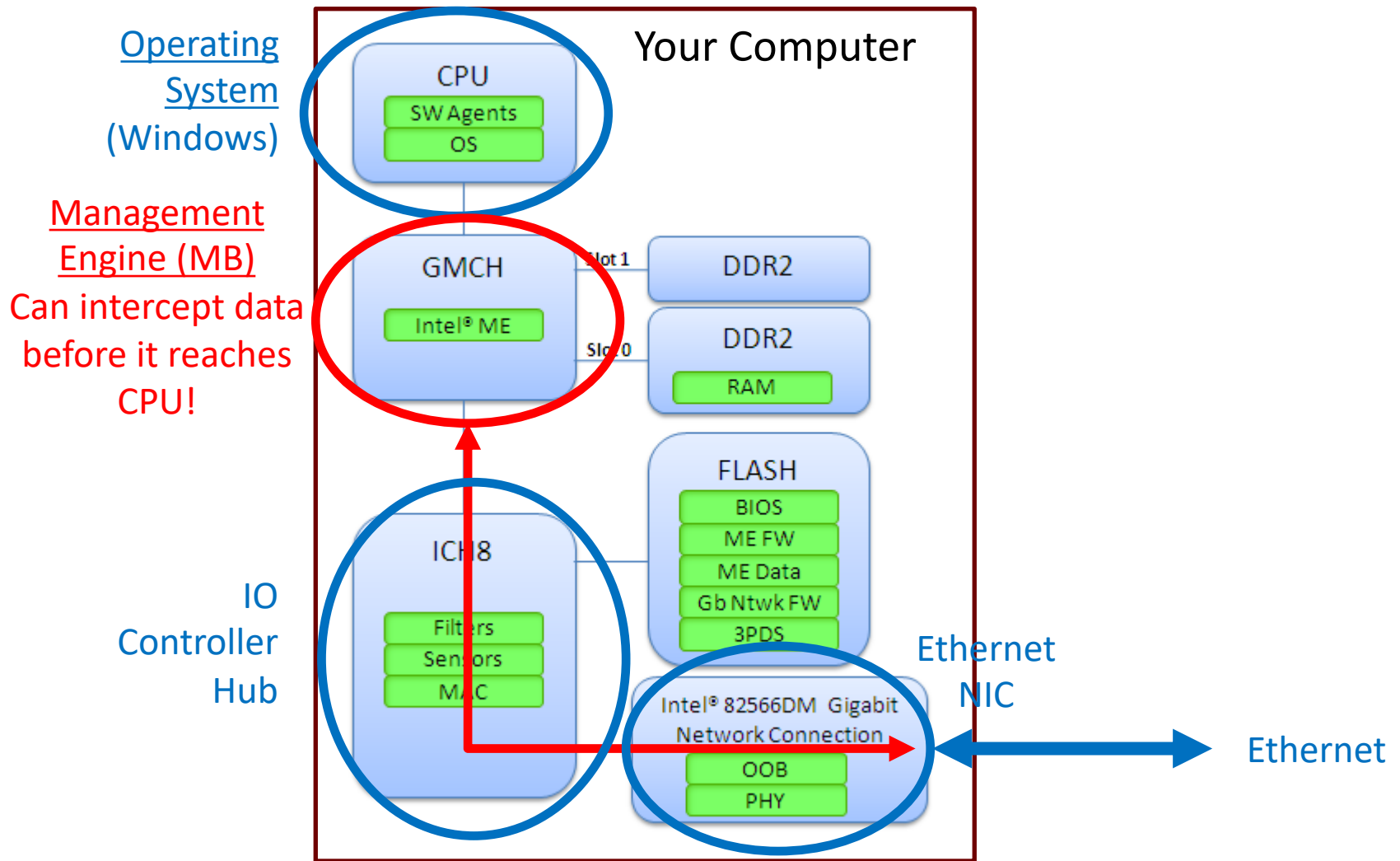
Is this all the software we need to worry about?

# What about the *hidden computer* inside your computer?

- **Intel™ Management Engine (ME)**
- Completely independent system embedded in Intel platforms
  - Has its own processor and memory
  - Has its own operating system (proprietary, signed)
  - Runs its own programs (proprietary, signed)
  - Direct access to memory, screen, keyboard, network
- Works if computer has no operating system install
- Works if computer is powered off (but still plugged into power and network)

# Management Engines

- Remote management functionality
  - Remote power up/power down – WOL
  - Remote boot (i.e. boot from remote disk)
  - Console redirection (serial over LAN)
  - Keyboard, video, mouse (KVM) over network
  - Network monitoring/filtering
  - Access hardware asset information
  - Protected audio/video path for DRM-protected media



# Management Engines

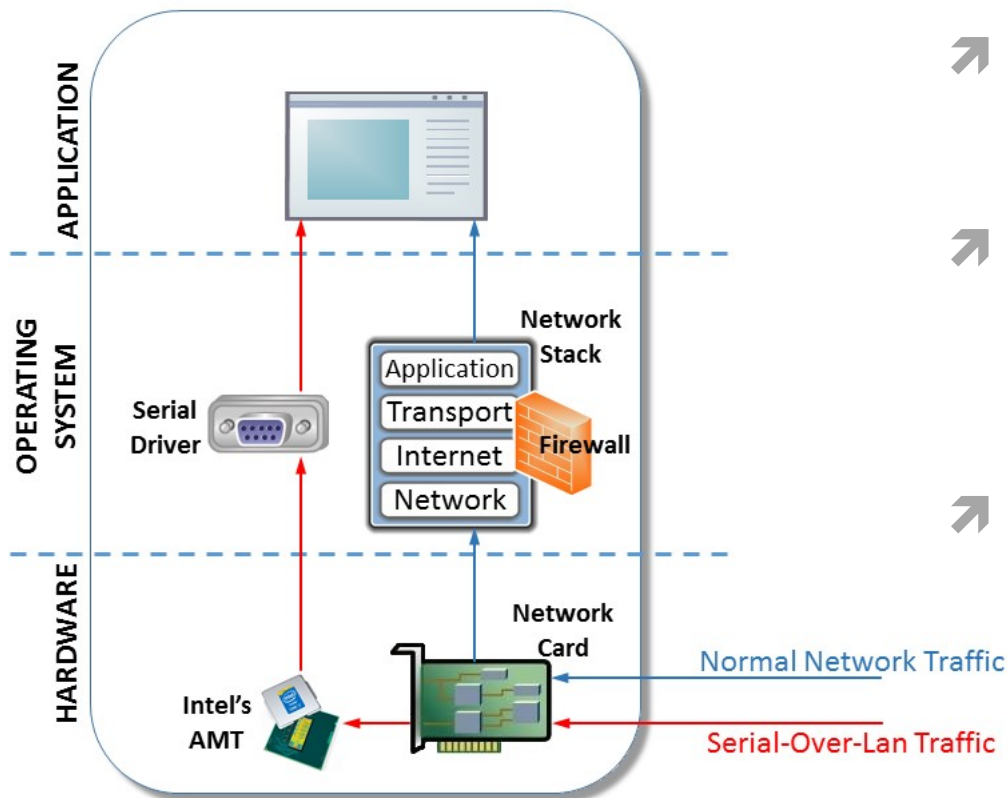
## ➤ Examples

- Intel Active Management Technology (AMT)  
vPro-enabled processors
- AMD Secure Processor  
(Formerly Platform Security Processor – PSP)

# Security Challenge

- CVE-2017-5689  
<https://nvd.nist.gov/vuln/detail/CVE-2017-5689>
  - “An unprivileged local attacker could provision manageability features gaining unprivileged network or local system privileges”
- Active Management (AMT) module - part of Management Engine
- Allows remote administrator to interact with keyboard/screen
- Supposed to require a password for access, but you can send a blank password instead!

# Security Challenge



- Data exfiltration technique nicknamed PLATINUM
- Evades firewalls and other traditional network monitoring tools
- Uses AMT-provided virtual serial port

<https://arstechnica.com/information-technology/2017/06/sneaky-hackers-use-intel-management-tools-to-bypass-windows-firewall/>  
<https://blogs.technet.microsoft.com/mmpc/2017/06/07/platinum-continues-to-evolve-find-ways-to-maintain-invisibility/>



# Software Security Mechanisms





# Software Security Architectures

**Motivating Question:** What architectures can we build in software on top of underlying hardware platform?

## ➤ Basic Features

- Process Isolation

## ➤ Advanced Features

- SELinux
- Virtualization
- Containers
- Sandboxes
- DEP
- ASLR

# Process Isolation



# Process Isolation

Resource	Limitation / Isolation	Enforced By
Memory	Cannot access memory of other processes	<i>Virtual memory system</i> (CPU+OS)
CPU	Execution may be paused or throttled at any time. No control over scheduling.	<i>Pre-emptive multitasking</i> (CPU+OS)
Disk I/O	Access may be granted, denied, or throttled at any time.	Operating System
Network I/O	Access may be granted, denied, or throttled at any time.	Operating System
Runtime	Each process has separate file descriptors, socket descriptors, ...	Operating System
Hardware (general)	No direct access to hardware	Operating System

# Process Isolation

- What *can* I do as a user process?
  - Run user-mode assembly instructions
  - Read/write to addresses within my virtual memory space
  - Anything else requires asking the kernel (via syscall)
- Kernel must be part of *trusted computing base* for process isolation to be successful
- Assuming we trust the kernel, what else can it do for us security-wise beyond these standard features?

# Protection in Linux

1. `fork() + setuid() + exec()`
2. `chroot()`
3. `seccomp()`
4. `prctl()`
5. SELinux
6. Namespaces

[http://nmap.gnutls.org/2015/06/software-isolation-in-linux\\_15.html](http://nmap.gnutls.org/2015/06/software-isolation-in-linux_15.html)

# Protection in Linux – fork/setuid/exec()

- Create a new process with separate memory space
- `fork()` – create a child process by duplicating the calling process
- `setuid()` – set the user ID of the new process
- `exec()` – replaces the current process with new process image
- Standard feature of Unix-like operating systems for decades

# Protection in Linux – chroot()

- Change the apparent root directory available to the currently running process and its children
- Files and commands *above* this point on the tree are no longer accessible
- Standard feature of Unix-like operating systems for decades

# Protection in Linux – seccomp()

- Filter (limit) system calls that are available to process
  - **Reduces the kernel attack surface!**
  - List of Linux 64-bit syscalls:  
[http://blog.rchapman.org/posts/Linux\\_System\\_Call\\_Table\\_for\\_x86\\_64/](http://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/)
  - Management library  
<https://github.com/seccomp/libseccomp>
- Example: only allow `read()/write()/ioctl()` syscalls but nothing else
  - If malicious code is injected into our process, it will be limited in what it can do



# Protection in Linux – seccomp()

```
#include <seccomp.h>
```

```
scmp_filter_ctx ctx;  
ctx = seccomp_init(SCMP_ACT_ERRNO(EPERM))  
assert(ctx == 0);
```

*return -1 and set errno to EPERM  
if unpermitted syscall run*

```
assert(seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(read), 0) == 0);  
assert(seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(write), 0) == 0);  
assert(seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(ioctl), 1,  
SCMP_A1(SCMP_CMP_EQ, (int)SIOCGIFMTU)) == 0);  
  
assert(seccomp_load(ctx) == 0);
```

*What does assert() do?*

# Protection in Linux – seccomp()

## ➤ Challenge: What syscalls does my program need?

- Tedious to find and tedious to enumerate
- Tip: Use `strace` utility to see syscalls, along with their arguments and return values
  - Be careful of syscalls only used for error conditions, or different syscalls in 32 and 64-bit code

```
$ strace ls
execve("/bin/ls", ["ls"], [/* 21 vars */]) = 0
brk(0)                                = 0x8c31000
access("/etc/ld.so.nohwcap", F_OK)     = -1 ENOENT (No such file or directory)
mmap2(NULL, 8192, PROT_READ, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb78c7000
access("/etc/ld.so.preload", R_OK)     = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)     = 3
...
```

<http://www.thegeekstuff.com/2011/11/strace-examples/>

# Aside – Fun with strace!

## Only show specific syscall(s):

```
$ strace -e open ls
open("/etc/ld.so.cache", O_RDONLY)      = 3
open("/lib/libselinux.so.1", O_RDONLY)  = 3
open("/lib/librt.so.1", O_RDONLY)       = 3
open("/lib/libacl.so.1", O_RDONLY)      = 3
open("/lib/libc.so.6", O_RDONLY)        = 3
open("/lib/libdl.so.2", O_RDONLY)       = 3
open("/lib/libpthread.so.0", O_RDONLY)  = 3
open("/lib/libattr.so.1", O_RDONLY)     = 3
open("/proc/filesystems", O_RDONLY|O_LARGEFILE) = 3
open("/usr/lib/locale/locale-archive", O_RDONLY|O_LARGEFILE) = 3
open(".", O_RDONLY|O_NONBLOCK|O_LARGEFILE|O_DIRECTORY|O_CLOEXEC) = 3
Desktop  Documents  Downloads  examples.desktop  libflashplayer.so
Music    Pictures    Public    Templates  Ubuntu_OS  Videos
```

<http://www.thegeekstuff.com/2011/11/strace-examples/>

# Aside – Fun with strace!

**Attach to a currently running process  
(If not same user as you, must be root...)**

```
$ ps -C firefox-bin
PID TTY TIME CMD 1725 ? 00:40:50 firefox-bin

$ sudo strace -p 1725
...
...
...
```

<http://www.thegeekstuff.com/2011/11/strace-examples/>

# Aside – Fun with strace!

**Print timestamp for each syscall:**

```
$ strace -t -e open ls /home
20:42:37 open("/etc/ld.so.cache", O_RDONLY) = 3
20:42:37 open("/lib/libselinux.so.1", O_RDONLY) = 3
20:42:37 open("/lib/librt.so.1", O_RDONLY) = 3
20:42:37 open("/lib/libacl.so.1", O_RDONLY) = 3
20:42:37 open("/lib/libc.so.6", O_RDONLY) = 3
20:42:37 open("/lib/libdl.so.2", O_RDONLY) = 3
20:42:37 open("/lib/libpthread.so.0", O_RDONLY) = 3
20:42:37 open("/lib/libattr.so.1", O_RDONLY) = 3
20:42:37 open("/proc/filesystems", O_RDONLY|O_LARGEFILE) = 3
20:42:37 open("/usr/lib/locale/locale-archive", O_RDONLY|O_LARGEFILE) = 3
20:42:37 open("/home", O_RDONLY|O_NONBLOCK|O_LARGEFILE|O_DIRECTORY|O_CLOEXEC) = 3
shafer
```

<http://www.thegeekstuff.com/2011/11/strace-examples/>

# Aside – Fun with strace!

**Print relative timestamp for each syscall:**

```
$ strace -r ls
0.000000 execve("/bin/ls", ["ls"], [/ * 37 vars */]) = 0
0.000846 brk(0) = 0x8418000
0.000143 access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or
directory)
0.000163 mmap2(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0xb787b000
0.000119 access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or
directory)
0.000123 open("/etc/ld.so.cache", O_RDONLY) = 3
0.000099 fstat64(3, {st_mode=S_IFREG|0644, st_size=67188, ...}) = 0
0.000155 mmap2(NULL, 67188, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb786a000 ... ..
```

<http://www.thegeekstuff.com/2011/11/strace-examples/>

# Protection in Linux – prctl()

## ➤ “Process Control”

```
#include <prctl.h>

prctl(PR_SET_DUMPABLE, 0);
```

- Disabling the `PR_SET_DUMPABLE` features prevents other processes (at the same privilege level, not root) from using `ptrace()` to observe memory, file descriptors, registers, and to control execution
  - `ptrace()` is often used by debuggers and code analysis tools (*and evildoers*)
- Enabled by default on some systems (but not all)

# Protection in Linux

## ➤ SELinux

➤ Will devote a full section to discussing

## ➤ Namespaces

➤ Will discuss in *Containers* section





SELinux



# SELinux

- “Security Enhanced Linux”
- Developed by NSA and RedHat
- First release in 2000
  - Merged into mainline kernel in 2003  
(so it’s not just a random set of NSA patches)
- Goal: Supplement existing *discretionary access control (DAC)* in Linux with *mandatory access control (MAC)*
  - **What’s the difference between these two?**
- <https://selinuxproject.org/>

# SELinux

- What does SELinux do to the Linux kernel?
  - MAC provides granular permissions for **subjects** (users, processes) and **objects** (files, pipes, network interfaces, devices)
  - Added security fields to kernel data structures
  - Added calls to hook kernel functions at critical points
  - Added functions for registering and unregistering security modules
  - Added kernel component (Security Server) to enforce security policies

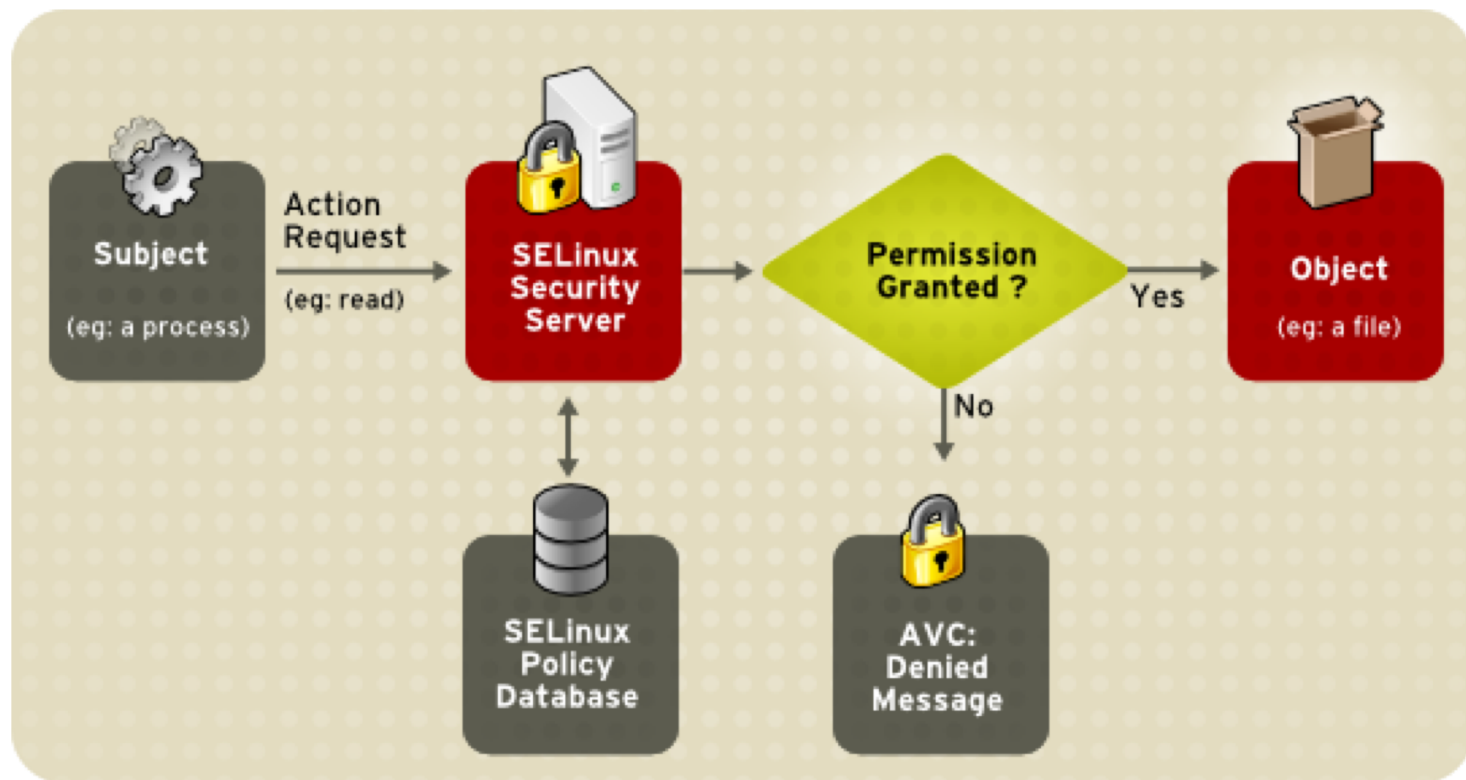
# Hooks (*Linux Security Modules*)

- Hooks actually provided by *Linux Security Modules* (generic framework for MAC, not specific to SELinux)
  - Mediate security-sensitive operations
    - Files, directories, IPC, network, semaphores, shared memory, ...
  - Variety of operations per data type
    - Control access to read of file data and file metadata separately
- Statically compiled into kernel to prevent tampering

# SELinux Policy Example

- For user to run passwd program
  - Only passwd should have permission to modify /etc/shadow
- Need permission to execute the passwd program
  - `allow user_t passwd_exec_t:file execute` (user can exec /usr/bin/passwd)
  - `allow user_t passwd_t:process transition` (user gets passwd perms)
- Must transition domains to passwd\_t from user\_t
  - `allow passwd_t passwd_exec_t:file entrypoint` (run w/ passwd perms)
  - `type_transition user_t passwd_exec_t:process passwd_t`
- Passwd can the perform the operation
  - `allow passwd_t shadow_t:file {read write}` (can edit passwd file)

# SELinux Architecture



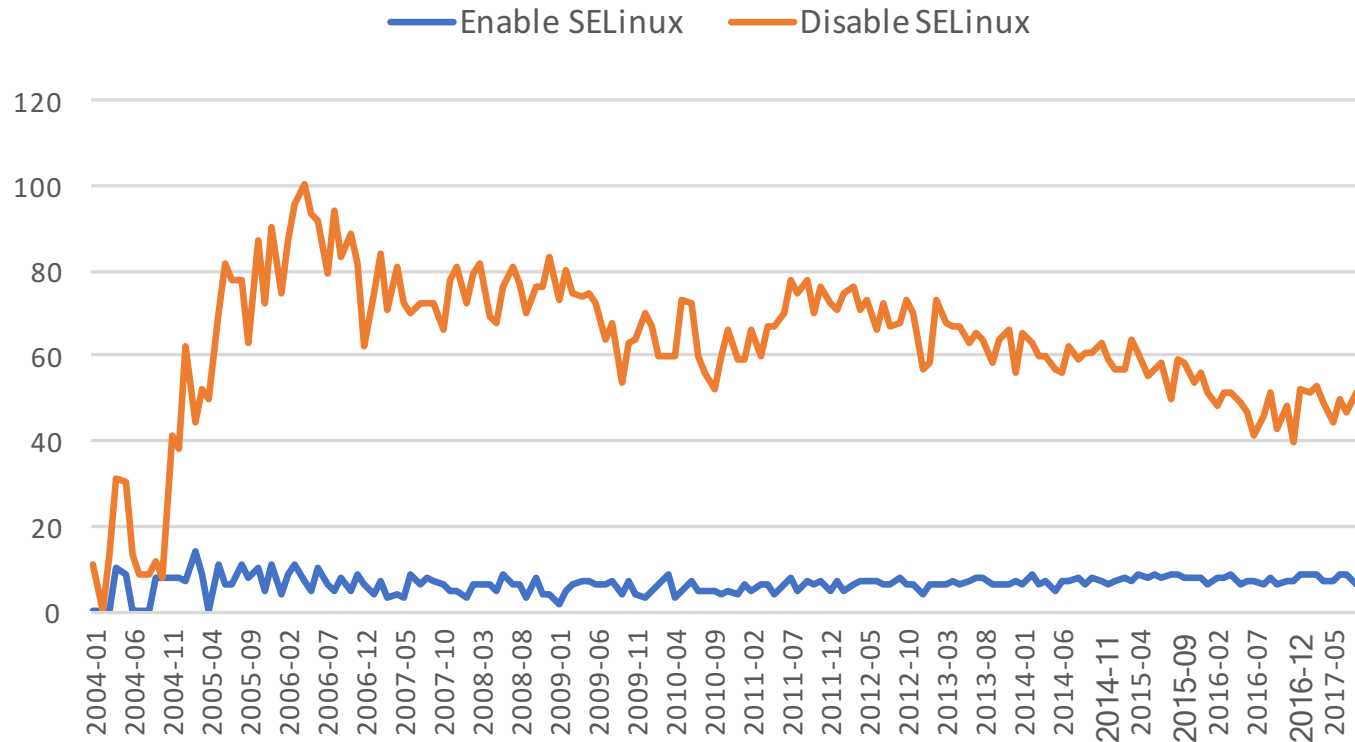
Design Goal: Separation of *policy* specification from *enforcement* of policy

# SELinux

- Complementary with previous security mechanisms
  - Use `seccomp()` to only allow `read()` syscall
  - Use SELinux to only accept certain file descriptors for `read()`
- Key difference: SELinux policy is configured at the system
  - Cannot be changed by an application
  - Centralized security policy under administrator control
  - `seccomp()` is configured at the programmer level

# SELinux

## Google Trends, 2004-Present



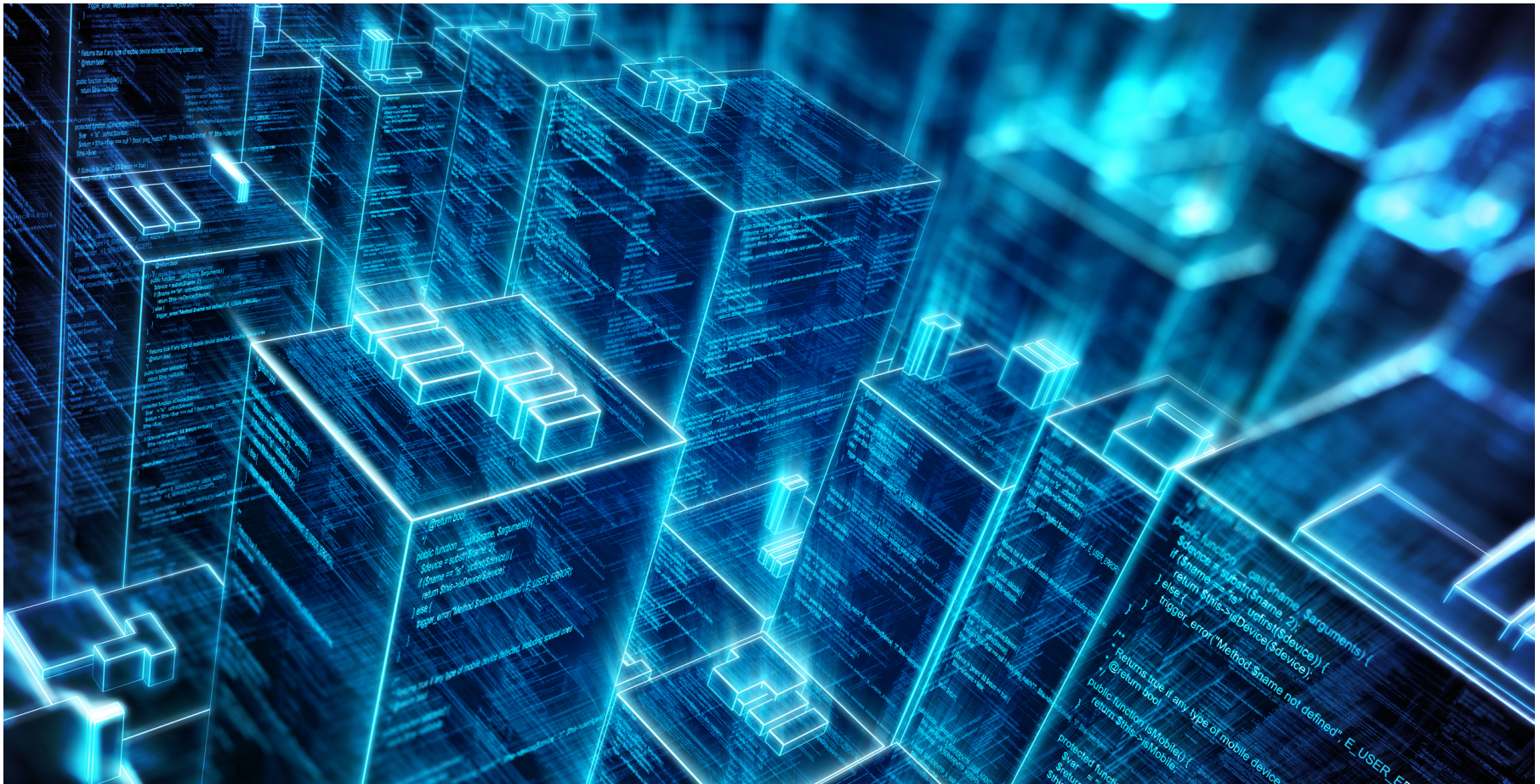
*#(\*#%\$^ Security!*

*Just turn the  
whole system off!*



# Other Examples

- Mandatory Access Control
  - Linux: SELinux, AppArmor, Smack, TOMOYO  
(all using Linux Security Modules)
  - BSD: TrustedBSD
  - Windows: Integrity Levels



# Virtualization



# Virtualization

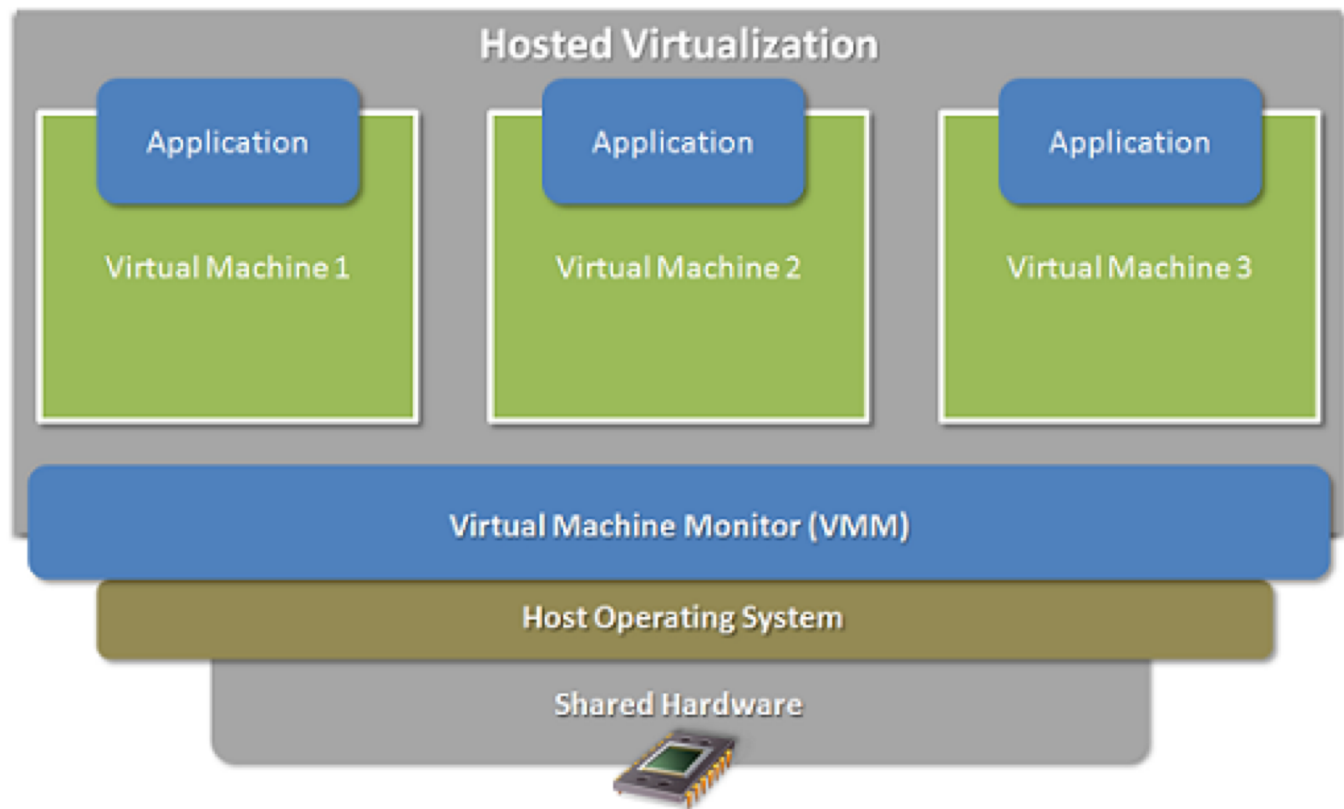
- **Q:** Does *virtualization* belong in the hardware or software section of our discussion?
- **A:** Yes!

# Terminology

- Hypervisor / Virtual Machine Monitor
  - Hardware + software that creates and runs virtual machines
  - Think “supervisor”, but where “hyper” is a stronger type of supervisor
- Classic terminology
  - Type 1 vs Type 2 hypervisor
    - Popek, Gerald J.; Goldberg, Robert P. (1974). “*Formal requirements for virtualizable third generation architectures*”. *Communications of the ACM*. 17 (7): 412–421. doi:10.1145/361011.361073

# Hosted Virtualization

*Type 2*  
Hypervisor



# Hosted Virtualization

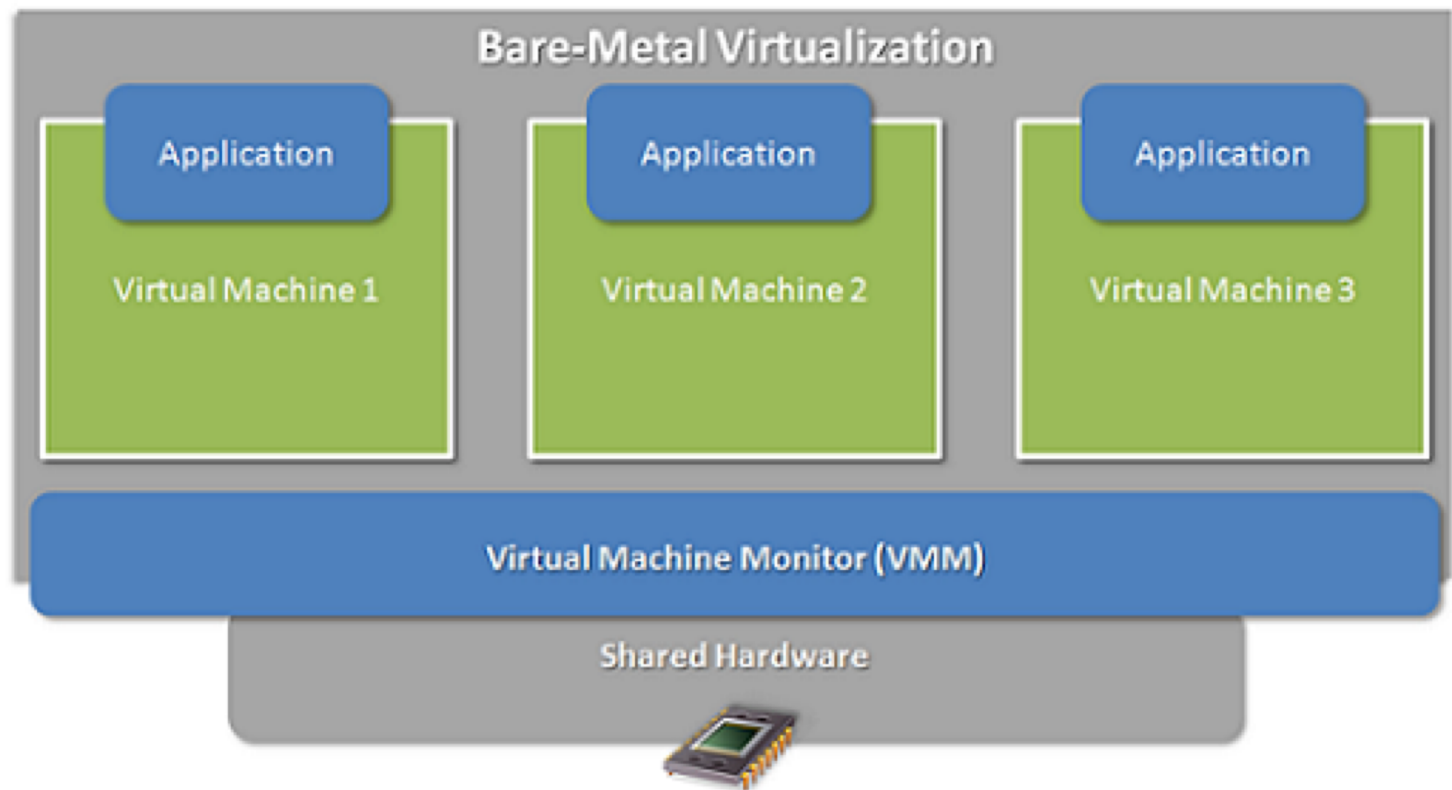
- Run on top of commodity OS
- Examples
  - VMWare Workstation, Player, Fusion
  - Virtualbox
  - Parallels Desktop

# Native / Bare Metal Virtualization

## Type 1 Hypervisor

More efficient, but not as easy to install.

The virtual machine monitor acts like an operating system itself!



# Native Virtualization

- Replace commodity OS
- Examples
  - Xen
  - VMWare ESX/ESXi
  - Linux kvm (kernel module that acts like Type 1 hypervisor)



# x86 Virtualization

- **Q: How do we support Virtual Machines?**
  - Multiple operating systems running on single computer
  - Each OS kernel thinks they “own” the hardware – how do we share?

# x86 Virtualization

## ➤ A1: Software Emulation

- **Dynamically recompile guest OS** and emulate hardware instructions in software
- Very complex / high overhead
- Examples
  - QEMU
    - <https://www.qemu.org/>
  - PearPC (PowerPC emulator on x86)
    - <http://pearpc.sourceforge.net/>
  - Unicorn (ARM, 68K, MIPS, SPARC)
    - <http://www.unicorn-engine.org/>

# x86 Virtualization

## ➤ A2: Paravirtualization

### ➤ Modify guest OS code

- Find all code that requires ring 0 permission
- Emulate that code in hypervisor
- Replace OS code with call to hypervisor

### ➤ Pros: High performance

### ➤ Cons:

- Limited OS support
- Must keep up with OS kernel development

### ➤ Example

- Xen VMM paravirtualization

# x86 Virtualization

## ➤ A3: Crazy software tricks

- Move the guest OS into ring 1
- Set “traps” on all instructions that cannot run at ring 1 or require adjustments to share resources with other guests
  - Trap into virtual machine manager and emulate in software
- Pros: Works with any OS
- Cons:
  - Slow due to emulation of some instructions (better than full emulation)
  - Very complicated - <http://www.virtualbox.org/manual/ch10.html#idp13729504>
- Example
  - VirtualBox (x86-only in software virtualization mode)

# x86 Virtualization

## ➤ A4: Hardware Virtualization

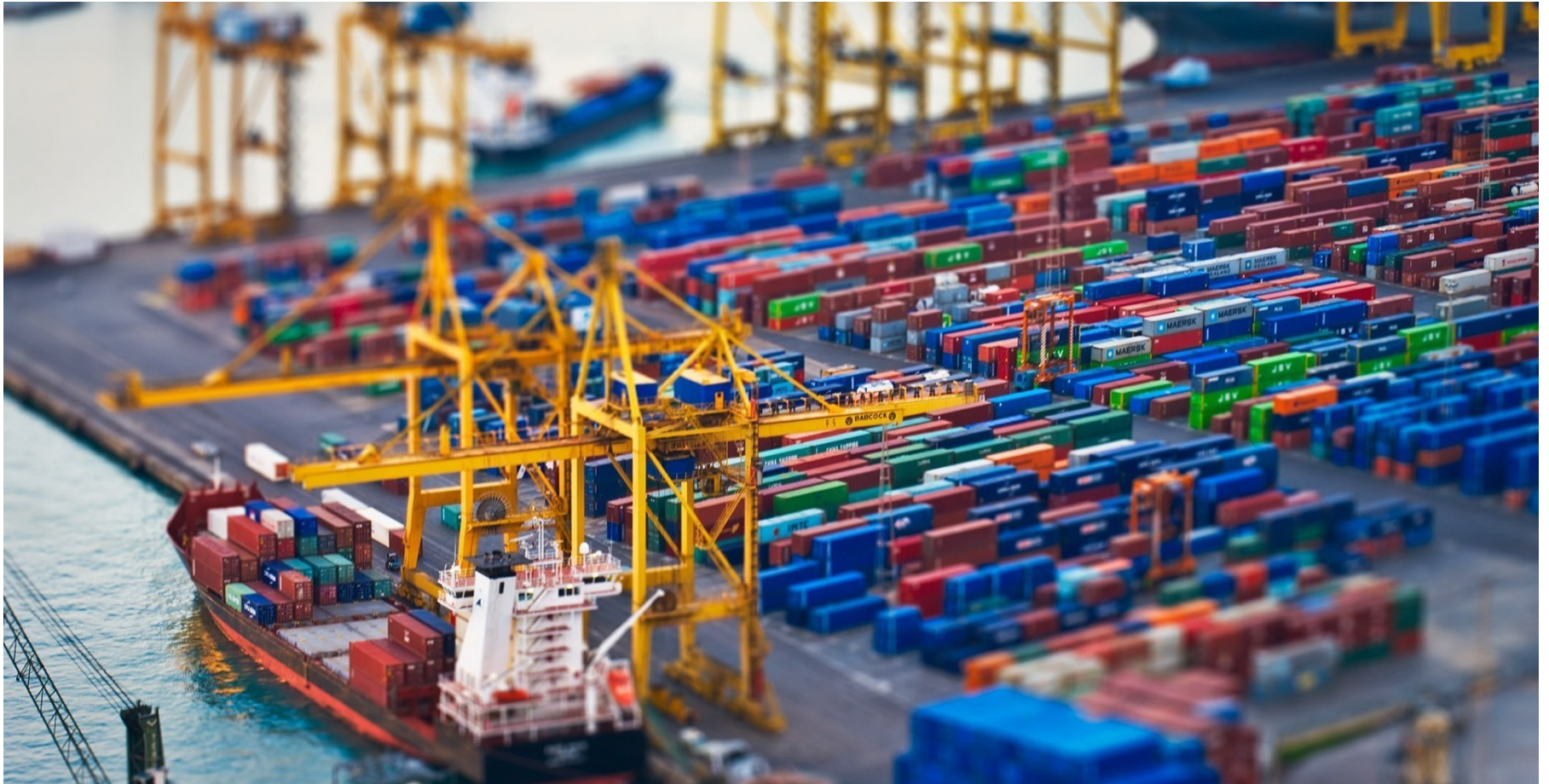
- Modify CPU to provide native virtualization support for un-modified operating systems
- **Ring -1 : Hypervisor Mode**
- Examples
  - Intel VT-x – Virtualization Technology for x86
  - AMD-V
- New machine instructions that only work at ring -1

# Hardware Virtualization

- Two new operating modes for CPU
  - Root mode – “classic” – Used by non-virtualized systems and hypervisor
  - Non-Root mode – Adds Virtual Machine Control Structure (VMCS) to control instruction behavior. Used by guest OS
- Two new events:
  - VMENTRY – Guest OS did something requiring hypervisor oversight (Root->Non-root mode)
  - VMEXIT – Hypervisor finished / time to resume guest (Non-root -> Root mode)

# Hardware-Assisted Virtualization

- Guest OS has address space not shared with hypervisor 😊
- Guest OS kernel runs at ring 0 in non-root mode 😊
- Minimal software emulation 😊
- No need to re-write paravirtualization code to keep up with changes in guest OS kernel 😊



# Containers





# Containers

- Operating System Level Virtualization
    - Aka “Containerization”
    - OS allows multiple isolated user-space instances
  - Modern examples
    - Docker - <https://www.docker.com/> (released 2013)
    - Linux LXC - <https://linuxcontainers.org/>
    - Linux OpenVZ - <https://openvz.org/>
    - FreeBSD Jails
    - Solaris Containers
- Not a new idea! These existed before virtual machines were the next big thing...*

# Container Features

- **Q:** What can I do inside a container?
  - Unique users (including root users!)
  - Unique memory
  - Unique files
  - Unique applications (processes)
  - Unique network configuration
  - Unique system libraries
  - Reboot containers independently

# Container Features

- **Q:** What can I *not* do in a container?
- **A:** Run a different OS than the host OS
  - If your host is Linux, your containers are Linux
  - If you host is FreeBSD, your containers are FreeBSD
- **A:** Change the kernel
  - It's the same kernel underlying all containers and the host OS

# Container Implementation

- How does this work? (in Linux)
  - `cgroups` (“control groups”) is kernel feature that isolates *hardware resources* used by a collection of processes
  - Resource = CPU, memory, disk I/O, network I/O, ...
    - Can set limits on groups
    - Can prioritize groups
    - Can freeze/checkpoint/research groups

# Container Implementation

- How does this work? (in Linux)
  - `namespaces` is kernel feature that virtualizes *system resources* used by a collection of processes
  - Resource = Process IDs, hostname, user IDs, interprocess communication, filesystem mount points, network interface names...
- Container systems like LXC, OpenVZ, Docker rely on kernel features like `cgroups` and `namespaces`

# Container Implementation - Filesystem

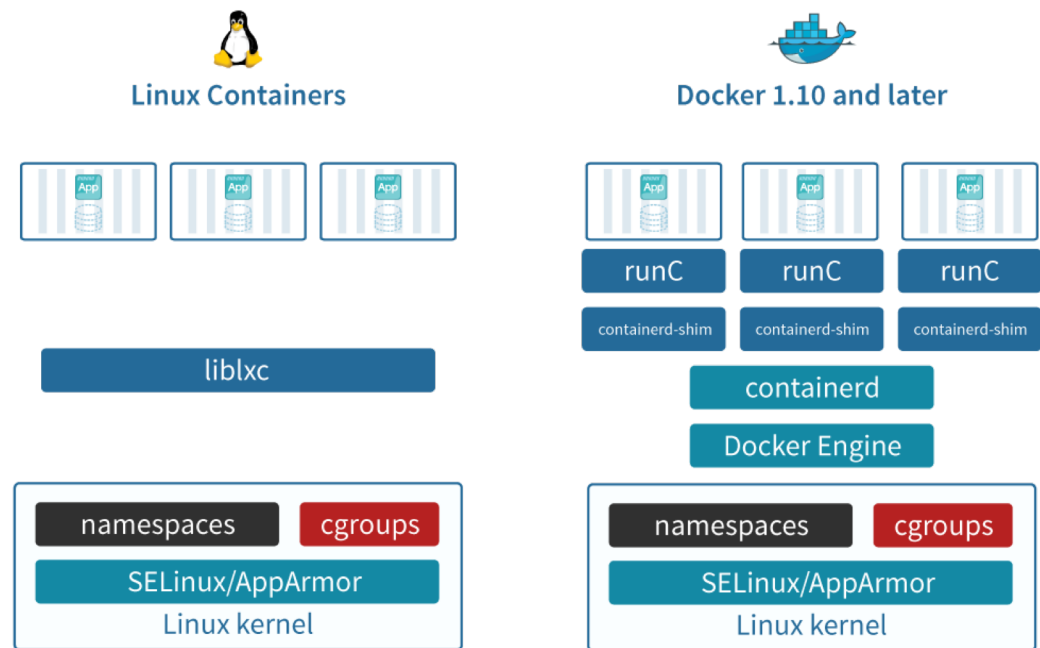
## ➤ Layers

- Original filesystem is marked *read-only*, and each container layers a new filesystem on top storing only modifications
  - Space efficient!
  - Easy to distribute! (no huge VM image)
- **Union mount** – combine multiple directories into single directory with unified view



# Docker vs LXC

- Share common kernel features (namespaces, cgroups)
- Different runtime engines/libraries
- LXC is an OS container
  - Designed to run multiple processes/services
- Docker is an Application container
  - Designed to package and run a **single service**

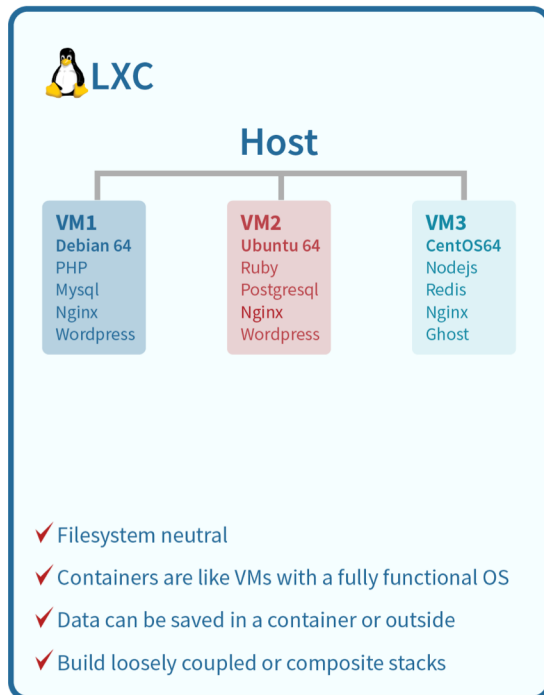


<https://robinsystems.com/blog/containers-deep-dive-lxc-vs-docker-comparison/>

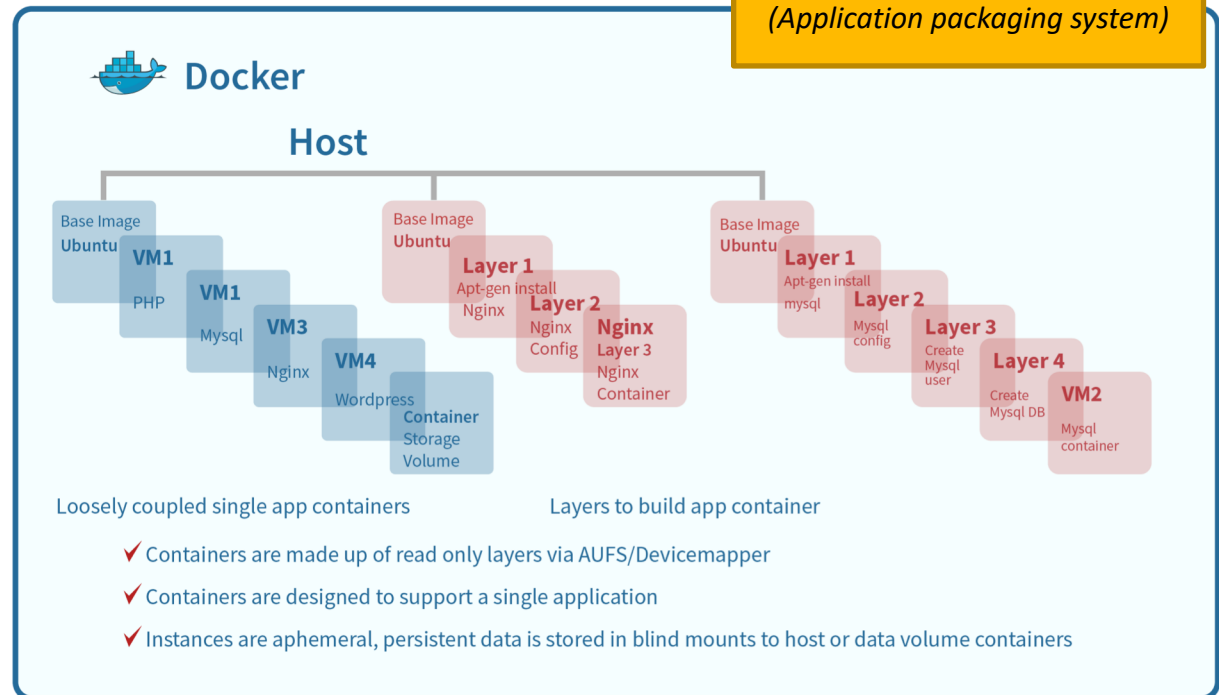


# Docker vs LXC

## Key differences between LXC and Docker



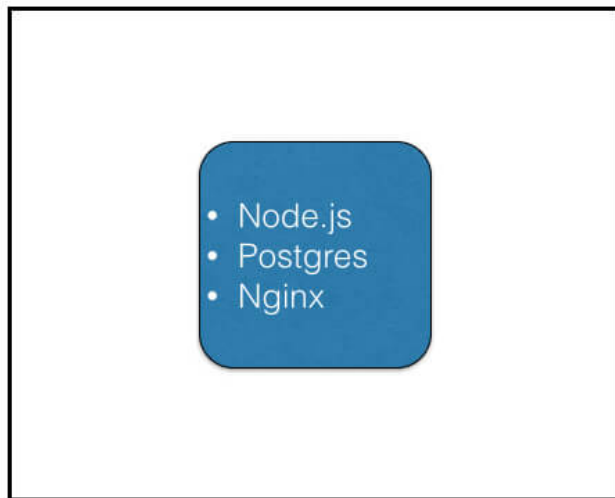
*LXC: Can run different Linux distributions  
but the kernel has to be identical*



**Microservices!**  
(Application packaging system)

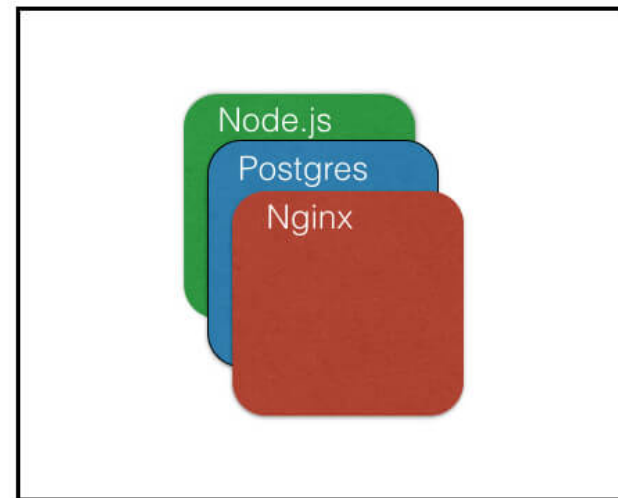
<https://robinsystems.com/blog/containers-deep-dive-lxc-vs-docker-comparison/>

# Docker vs LXC



OS containers

- Meant to be used as an OS - run multiple services
- No layered filesystems by default
- Built on cgroups, namespaces, native process resource isolation
- Examples - LXC, OpenVZ, Linux VServer, BSD Jails, Solaris Zones



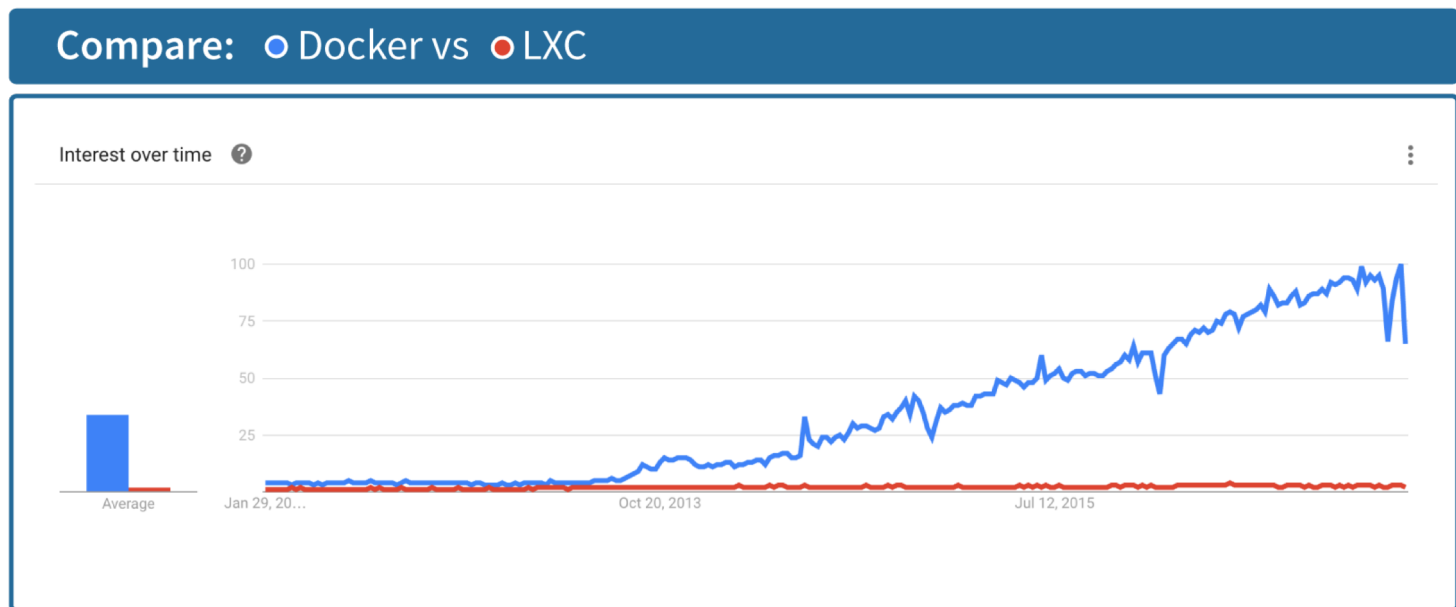
App containers

- Meant to run for a single service
- Layered filesystems
- Built on top of OS container technologies
- Examples - Docker, Rocket

<https://blog.risingstack.com/operating-system-containers-vs-application-containers/>

# Docker vs LXC

Google search trends 2017 – who is winning?



<https://trends.google.com>

<https://robinsystems.com/blog/containers-deep-dive-lxc-vs-docker-comparison/>

# Containers vs Virtual Machines

Server Virtualization -> Virtual Machines, *as is*  
Operating System Virtualization -> Containers

- Overhead?
  - Containers have less overhead  
(VMs must virtualize the hardware + run full guest OS)
- Requirements?
  - Containers require no hardware support
- Flexibility?
  - VMs are more flexible (can run multiple guest operating systems of different types on the same host)

# Containers vs Virtual Machines

## ➤ Security?

### ➤ Containers are not fully isolated! ☹️

- Filesystems under /sys not virtualized
- Devices not virtualized (/dev/mem, dev/sd\*)
- Kernel modules not virtualized
- SELinux not virtualized

### ➤ “Containers do not contain”

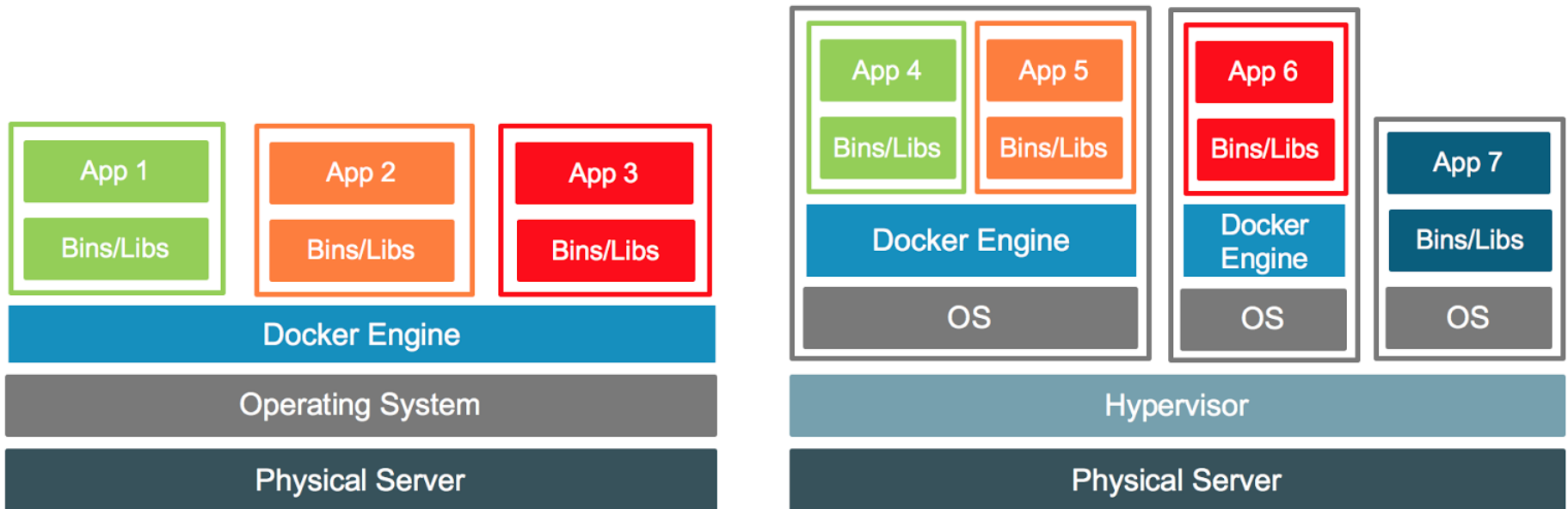
<https://opensource.com/business/14/7/docker-security-selinux>

### ➤ **Must apply same security practices as with applications running *outside* of a container**

- Drop privileges as quickly as possible
- Run services as non-root whenever possible
- Treat root *within* container as if it was root *outside* of container

# All of the Above: Containers and VMs

Your Datacenter or VPC



<https://blog.docker.com/2016/04/containers-and-vm-together/>



# Sandboxes



# Sandbox

- Field of **Software Development**
  - Sandbox is safe place to test code isolated from production environment



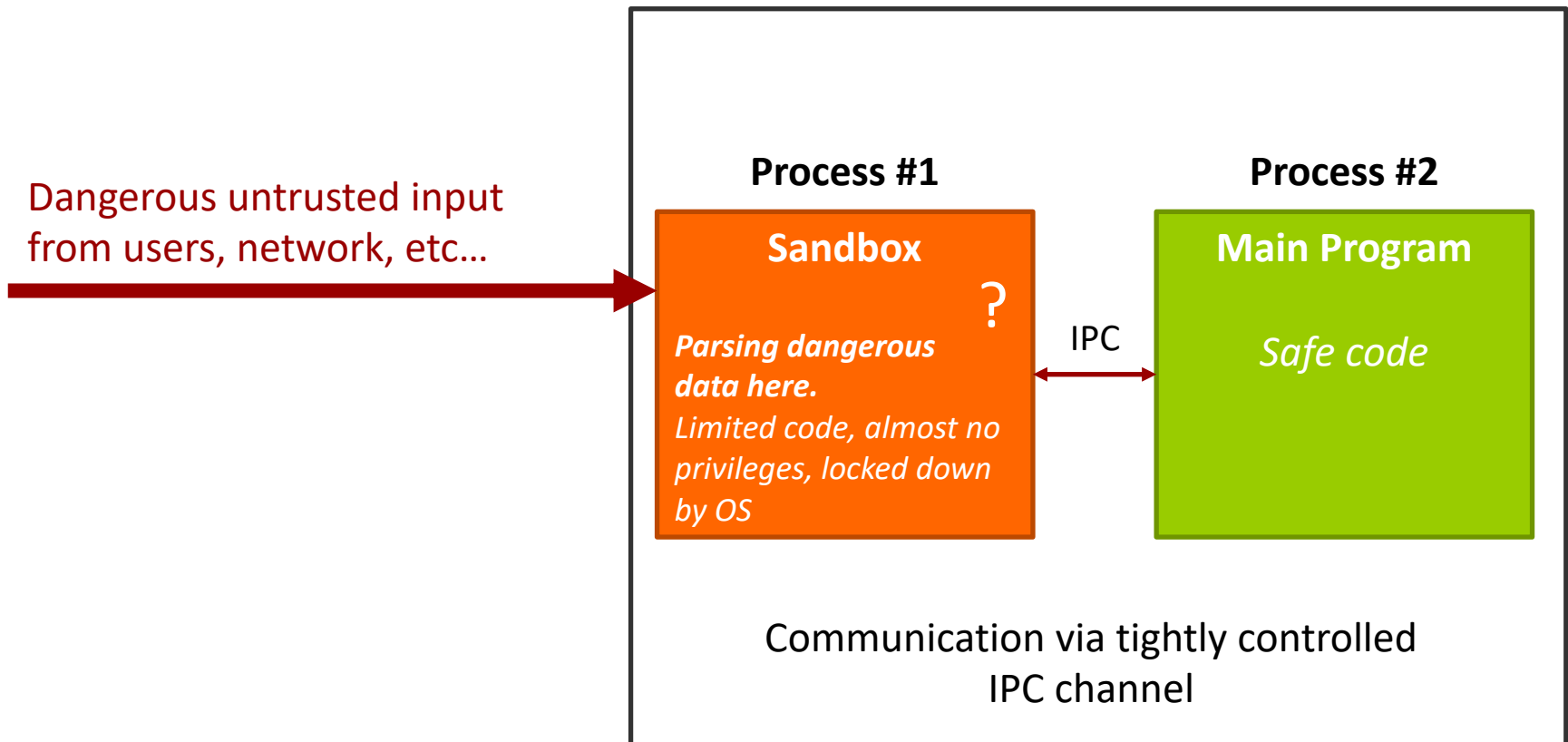
# Sandbox

## ➤ Field of **Computer Security**

- Sandbox is safe place to run code at high risk of exploitation
- Idea: We assume *arbitrary code execution* (by attacker) inside of sandbox
- Idea: We *try* to write perfect code (of course), but design *defense-in-depth* so that if we fail, the attacker will have difficulty leveraging our vulnerability into a useful exploit. No single point of failure!

# Sandbox (Generic)

## Your System



# Sandbox (Generic)

- **Q: What should our sandbox not be allowed to do?**
- **A:** Will vary by design, but potentially
  - Read/write to disk? (beyond scratch storage)
  - Read/write to network?
  - Create new processes or threads?
  - Use more than X% of CPU or memory?
  - Apply the *least privileges* principle very strongly here and lock it down!

# Sandbox Examples

- “Security In-Depth for Linux Software”
  - [https://www.cr0.org/paper/jt-ce-sid\\_linux.pdf](https://www.cr0.org/paper/jt-ce-sid_linux.pdf)
- Web browsers: Google Chrome, IE, Edge
- File viewers: Adobe Acrobat
- ...

# Data Execution Prevention (DEP)



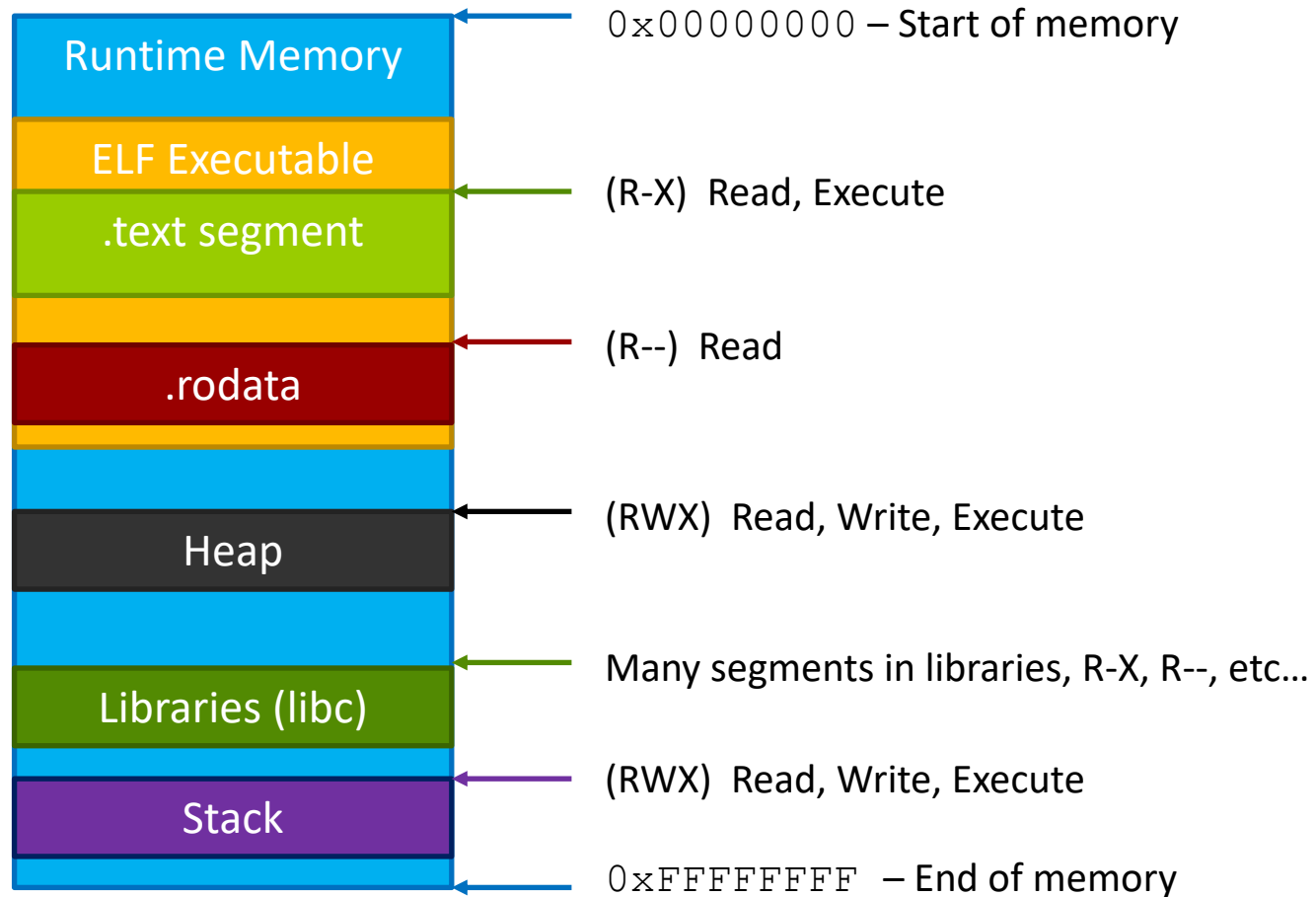
# Data Execution Prevention (DEP)

- Exploit mitigation technique
  - Make blackhat's job harder by preventing malicious code from executing out of any random memory location they might find
- Idea: Only *legitimate* code segments should be marked as executable
  - Also known as DEP, NX, XN, XD, W^X...
  - *Think back to discussion on memory paging and enforcement by hardware*

# Data Execution Prevention (DEP)

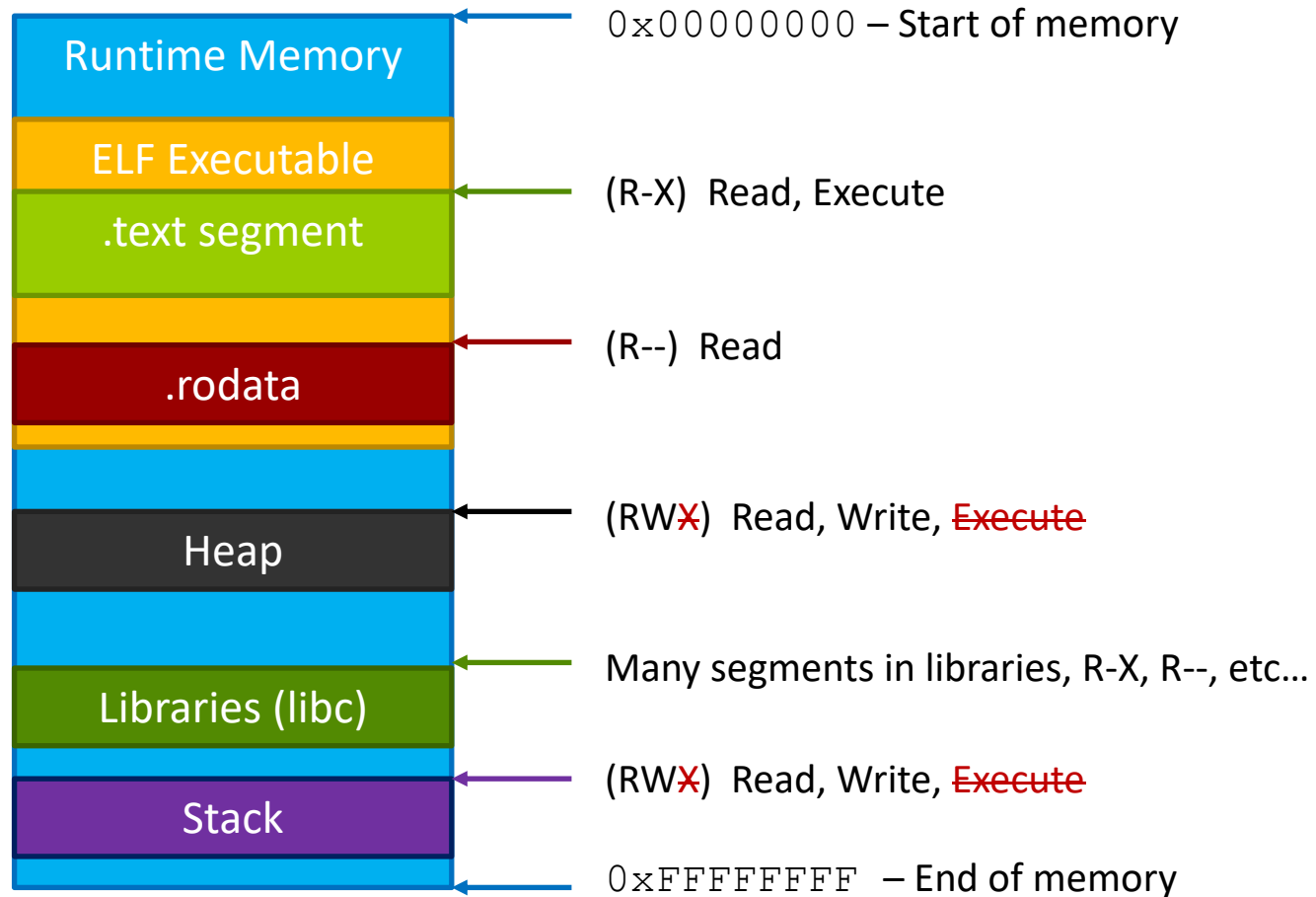
- Goal: No segment of memory should ever be **writeable** and **executable** at the same time
  - $W \wedge X$
  - *Unless you're writing a JIT compiler, you have no excuse for writing code onto the heap and then executing it*
- Data segments
  - Stack, Heap
  - (others)
- Code segments
  - .text
  - (others)

# Runtime Process without DEP

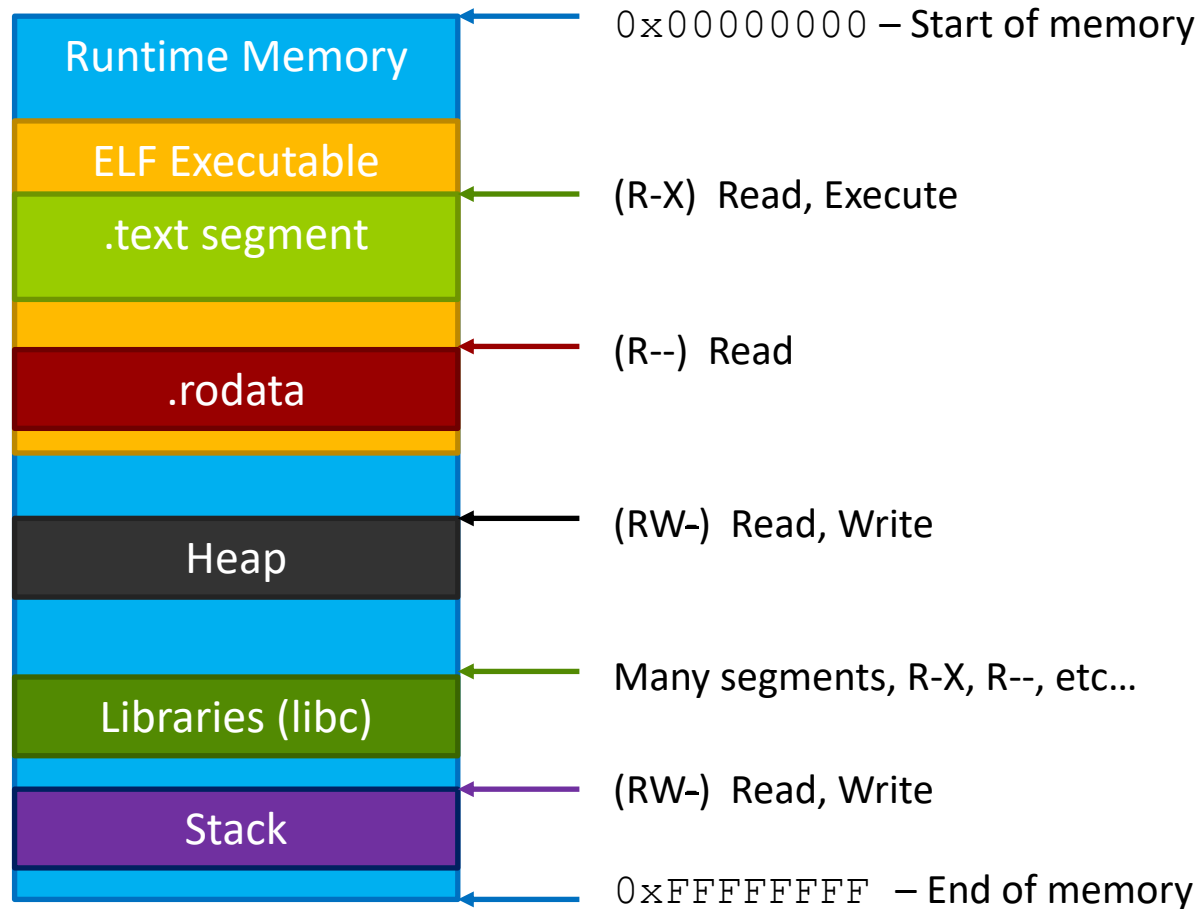




# Runtime Process with DEP



# Runtime Process with DEP



# Data Execution Prevention (DEP)

- Widespread adoption
  - Linux: Kernel 2.6.8 (2004)
  - Windows: Windows XP SP2 (2004)
  - Mac: OS X 10.5 (2006)
  - iOS: Always?
  - Android: 2.3

# Address Space Layout Randomization (ASLR) ↗

# ASLR

- Exploit mitigation technique
  - Make blackhat's job harder by preventing identical code from working on all victim machines
- Idea: Address ranges for important memory segments should be **random** at every execution
  - Memory segments are no longer in static address ranges
  - Attackers have no expectation where anything is in memory

# ASLR

➤ Q: What are we randomizing?

➤ A: ~~All of the things!~~

Base address of user **executable code**

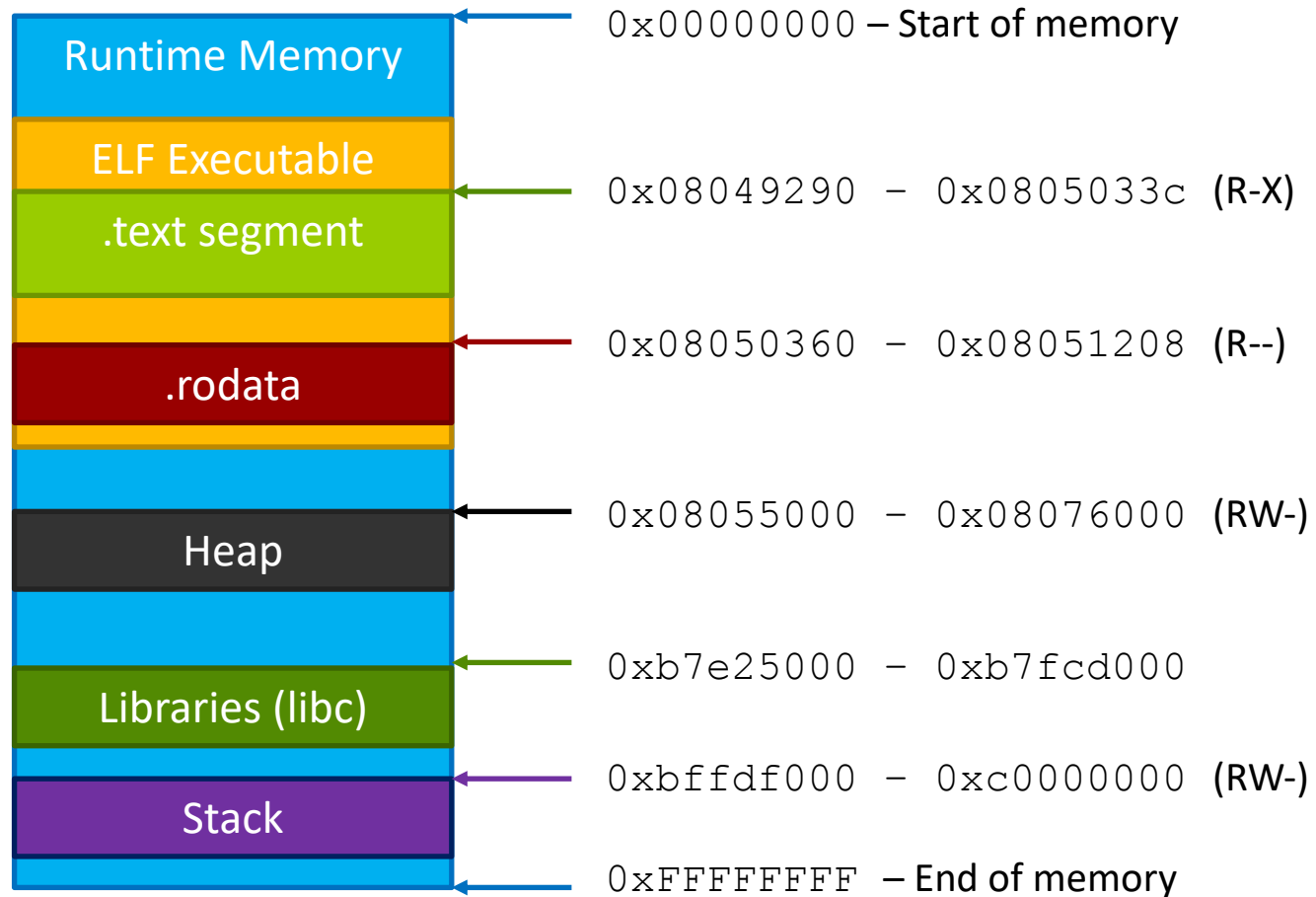
Base address of **stack**

Base address of **heap**

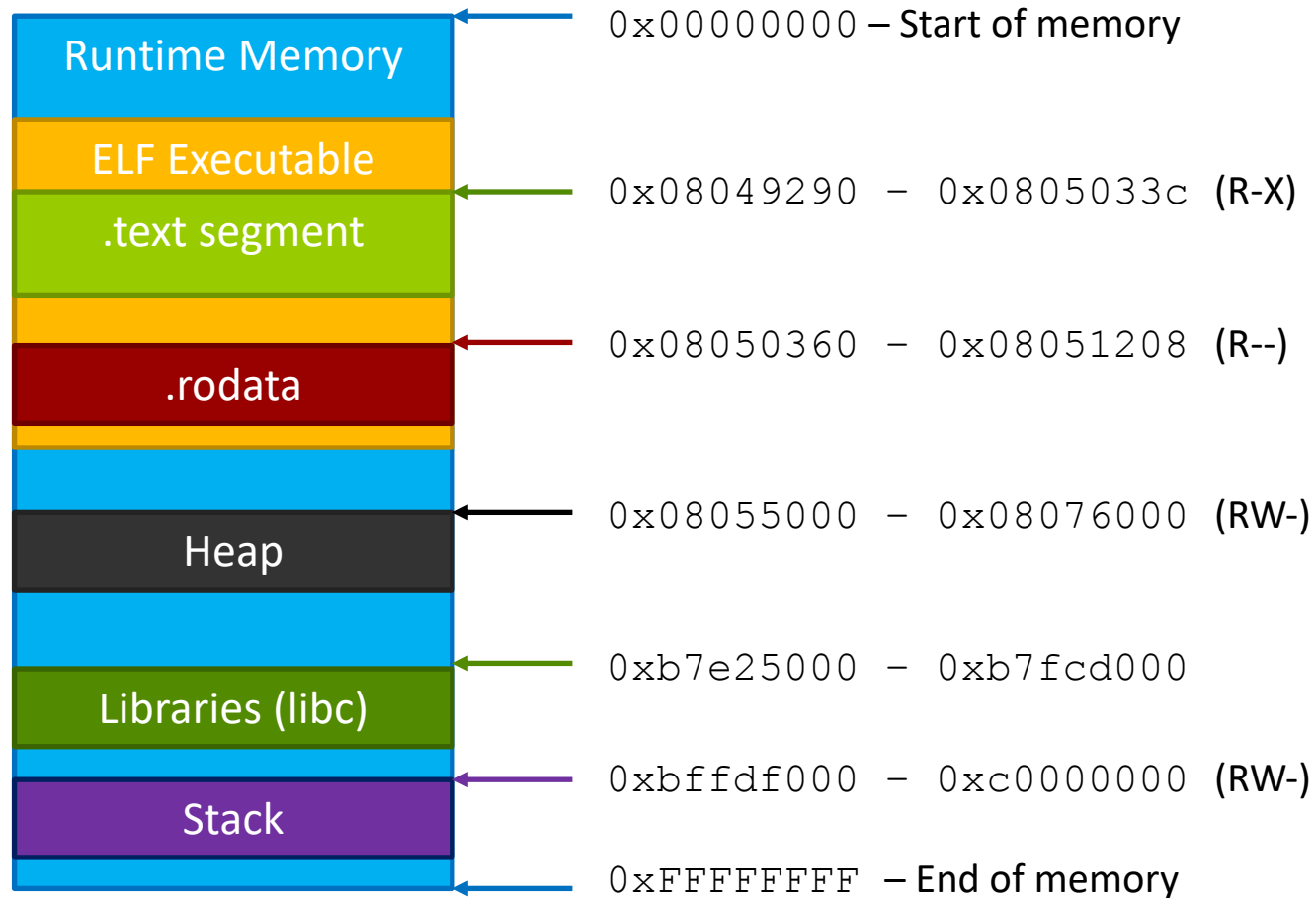
Base address of **libraries**

➤ These are all **virtual addresses**

# Runtime Process without ASLR

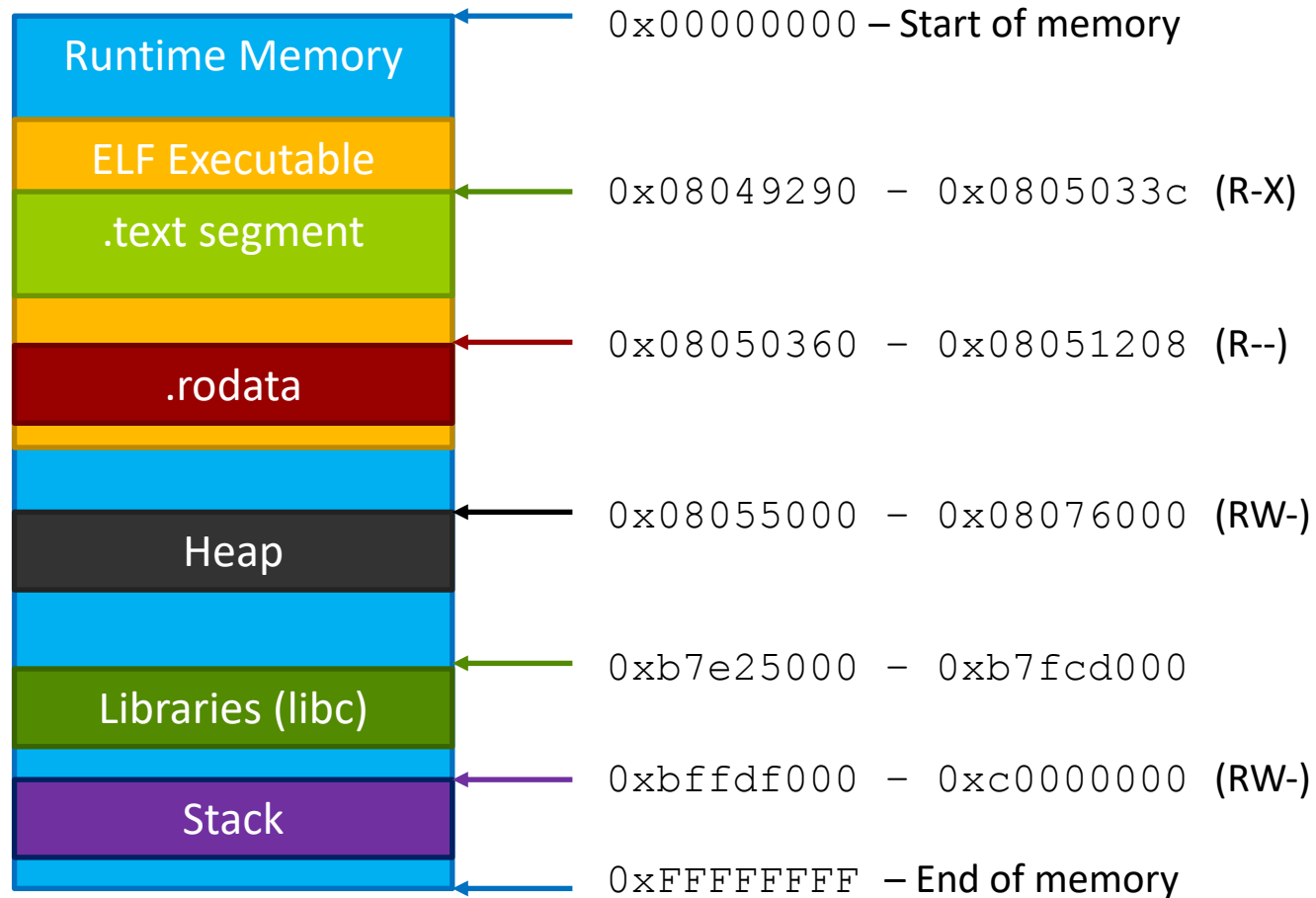


# Run #1 without ASLR

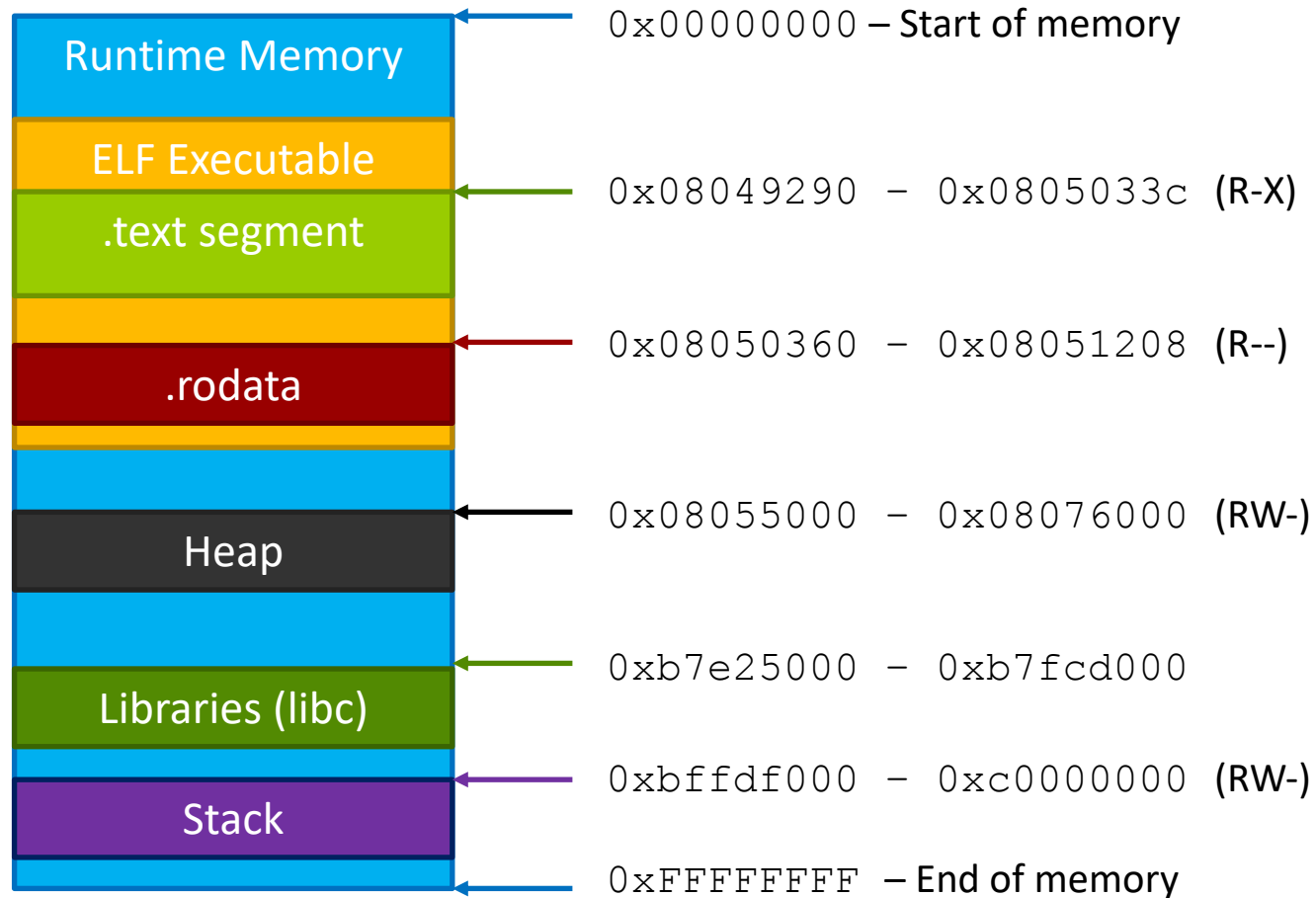




# Run #2 without ASLR



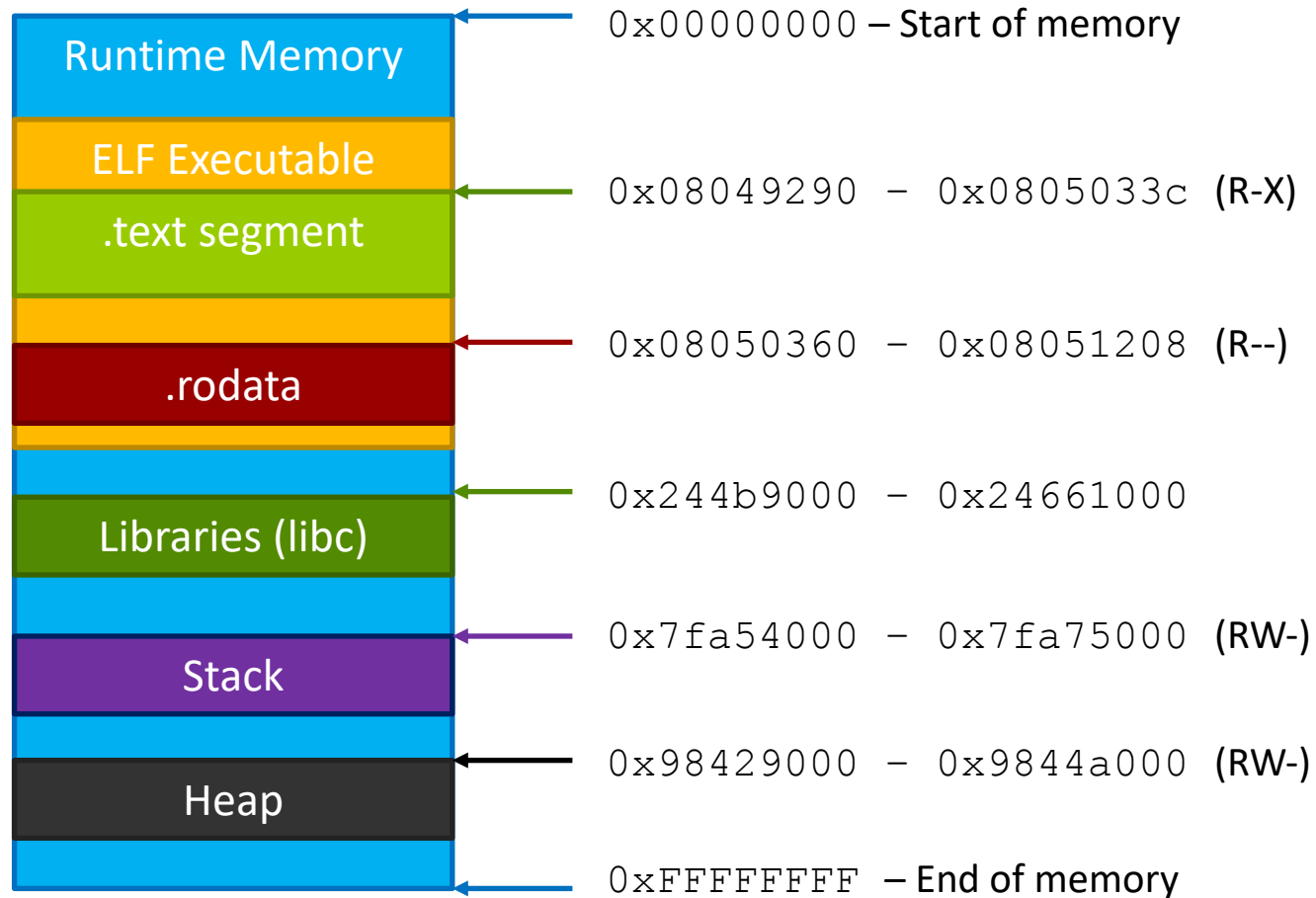
# Run #3 without ASLR



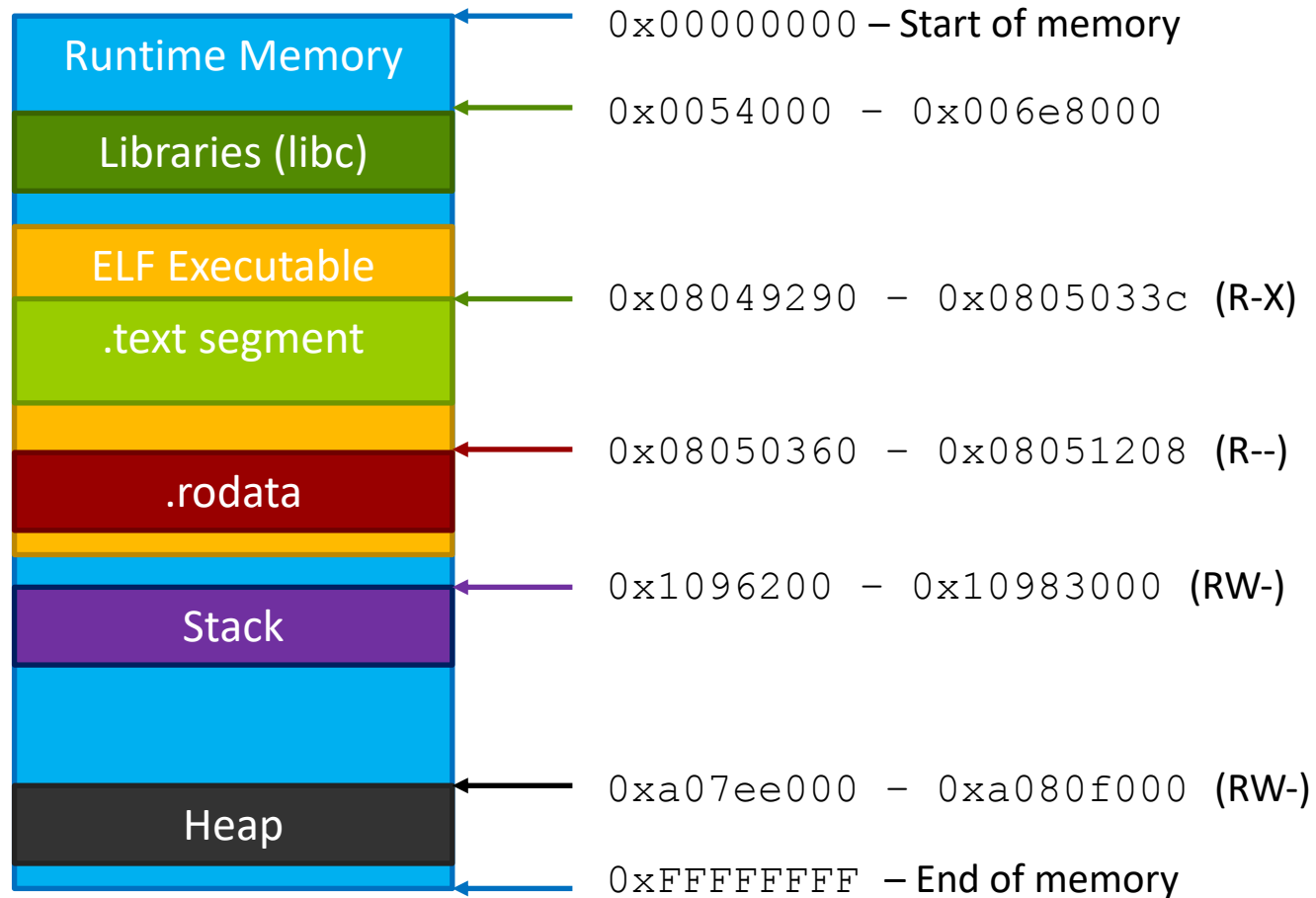
# ASLR

➤ Let's add randomization now with ASLR

# Run #1 with ASLR



# Run #1 with ASLR



# ASLR Support

- Widespread industry adoption
- Linux: 2005 (kernel 2.6.12+) - Stack & nmap
  - Compile-time option: *Position Independent Executable*  
`$ gcc -pie -fPIE -o program program.c`
  - Required for shared libraries, optional for programs
- Windows: 2007 (Windows Vista+) – Full ASLR
  - Link-time option: /DYNAMICBASE
  - <https://insights.sei.cmu.edu/cert/2014/02/differences-between-aslr-on-windows-and-linux.html>
- Mac: 2011 (Mac OS X 10.7+) – Full ASLR
- iOS: 2011 (iOS 4.3) – Full ASLR
- Android: 2012 (Android 4.0+) – Full ASLR