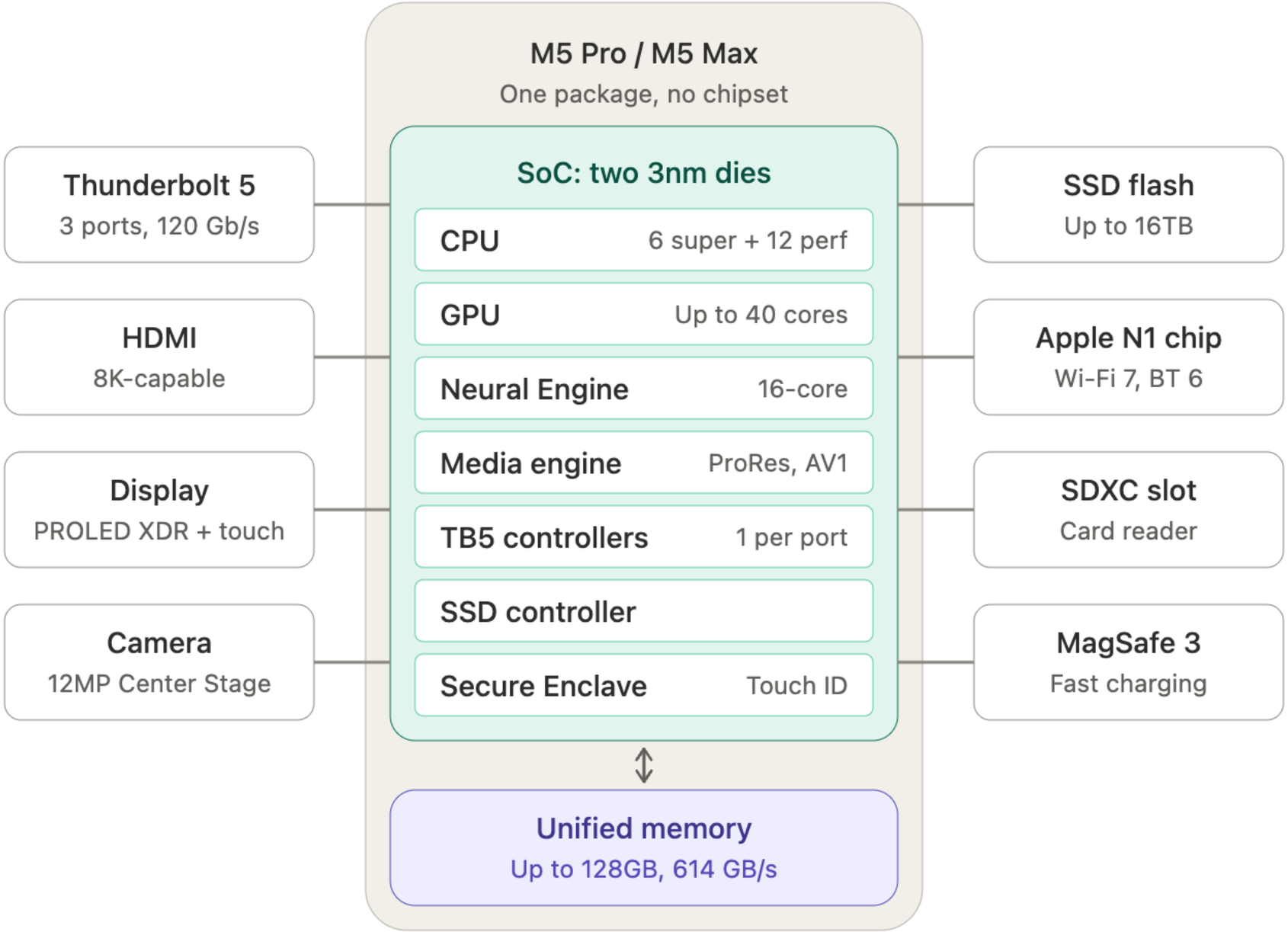
2550 Intro to systems security

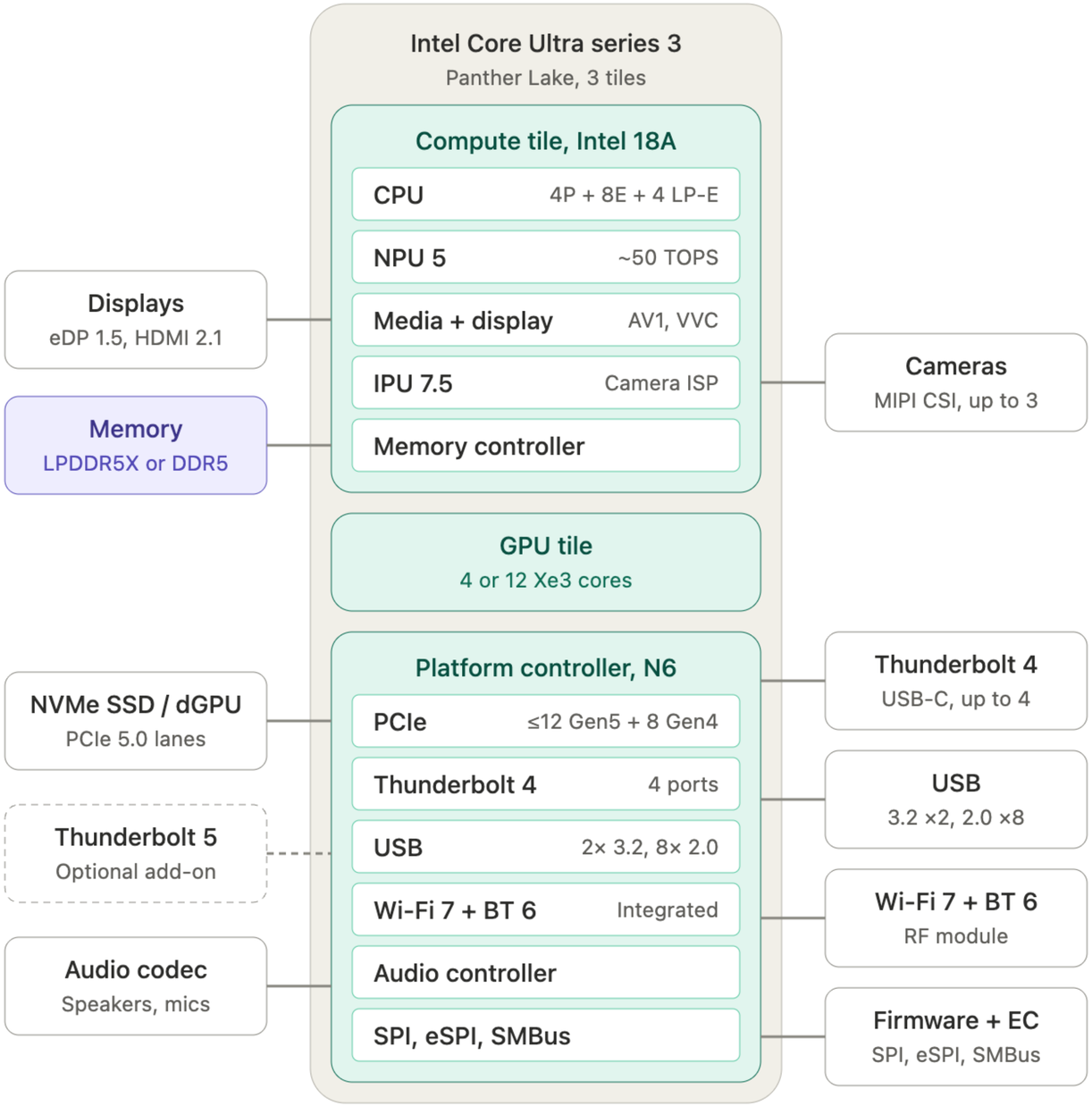
L15

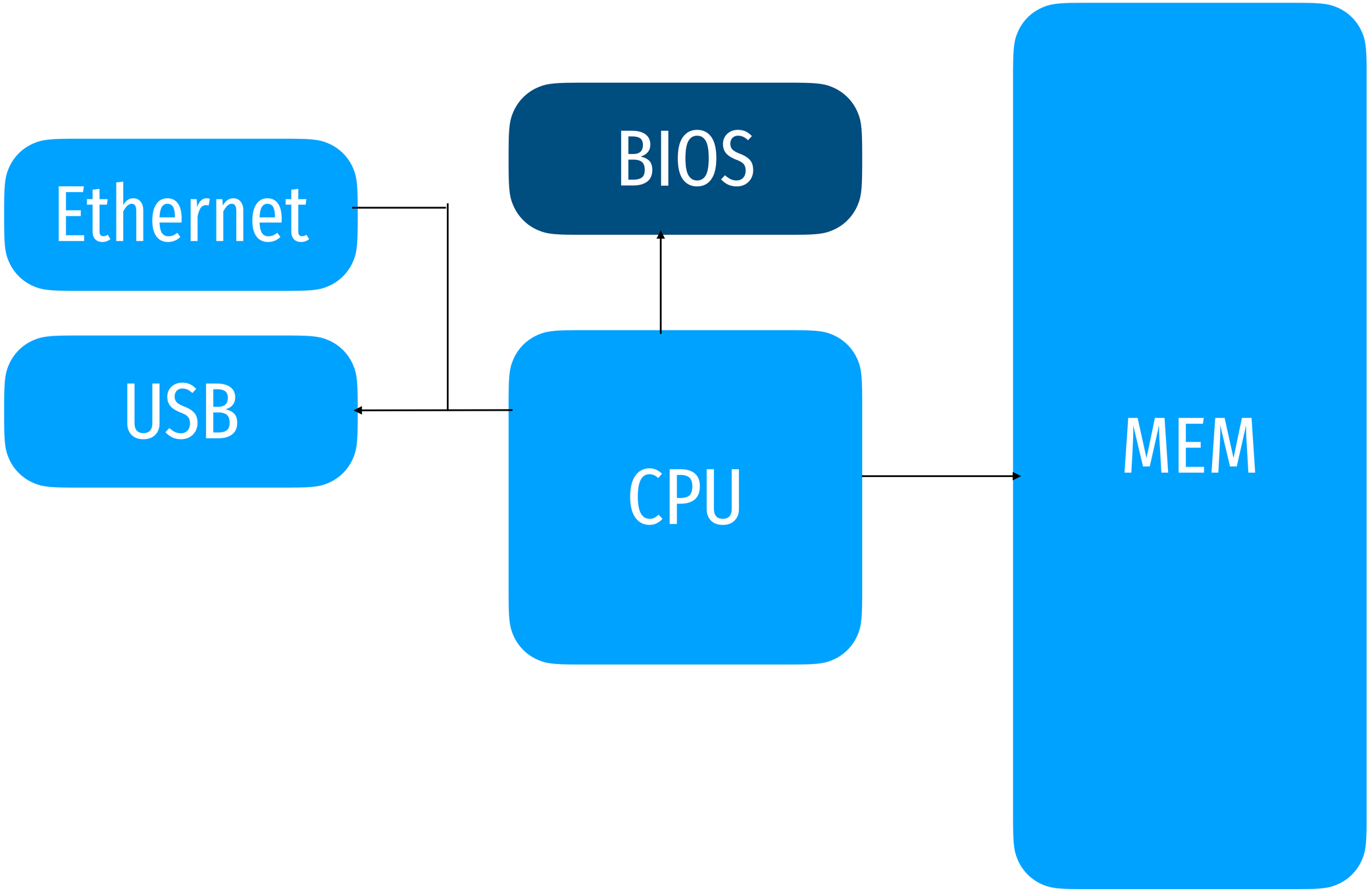
abhi shelat


IBM
IBM
IBM


M5


M5 Pro / M5 Max
One package, no chipset
SoC: two 3nm dies
CPU
6 super + 12 perf
GPU
Up to 40 cores
Neural Engine
16-core
Media engine
ProRes, AV1
TB5 controllers
1 per port
SSD controller
Secure Enclave
Touch ID
Unified memory
Up to 128GB, 614 GB/s
Thunderbolt 5
3 ports, 120 Gb/s
HDMI
8K-capable
Display
PROLED XDR + touch
Camera
12MP Center Stage
SSD flash
Up to 16TB
Apple N1 chip
Wi-Fi 7, BT 6
SDXC slot
Card reader
MagSafe 3
Fast charging


Intel Core Ultra series 3
Panther Lake, 3 tiles
Compute tile, Intel 18A
CPU
4P + 8E + 4 LP-E
NPU 5
~50 TOPS
Media + display
AV1, VVC
IPU 7.5
Camera ISP
Memory controller
GPU tile
4 or 12 Xe3 cores
Platform controller, N6
PCIe
≤12 Gen5 + 8 Gen4
Thunderbolt 4
4 ports
USB
2× 3.2, 8× 2.0
Wi-Fi 7 + BT 6
Integrated
Audio controller
SPI, eSPI, SMBus
Displays
eDP 1.5, HDMI 2.1
Memory
LPDDR5X or DDR5
NVMe SSD / dGPU
PCIe 5.0 lanes
Thunderbolt 5
Optional add-on
Audio codec
Speakers, mics
Cameras
MIPI CSI, up to 3
Thunderbolt 4
USB-C, up to 4
USB
3.2 ×2, 2.0 ×8
Wi-Fi 7 + BT 6
RF module
Firmware + EC
SPI, eSPI, SMBus
Dashed = optional, added by the laptop maker


Ethernet
USB
BIOS
CPU
MEM

What is Memory?

Memory is a “spreadsheet" with a single column

- Every row has a number, or address
- Every cell holds 1 byte of data

Address	Contents
114	
113	
112	
111	
110	
109	
108	
107	
106	
105	
104	
103	
102	
101	
100	

What is Memory?

Memory is a “spreadsheet" with a single column

- Every row has a number, or address
- Every cell holds 1 byte of data

Integers are typically four bytes

Address	Contents
114	
113	0
112	0
111	0
110	8
109	
108	
107	
106	
105	
104	
103	
102	
101	
100	

What is Memory?

Memory is a “spreadsheet" with a single column

- Every row has a number, or address
- Every cell holds 1 byte of data

Integers are typically four bytes

Each ASCII character is one byte, Strings are null terminated

Address	Contents
114	
113	0
112	0
111	0
110	8
109	
108	0
107	C
106	B
105	A
104	
103	
102	
101	
100	

What is Memory?

Memory is a “spreadsheet" with a single column

- Every row has a number, or address
- Every cell holds 1 byte of data

Integers are typically four bytes

Each ASCII character is one byte, Strings are null terminated

CPUs understand instructions in assembly language

Address	Contents
114	
113	0
112	0
111	0
110	8
109	
108	0
107	C
106	B
105	A
104	
103	0xAF
102	0x3C
101	0x91
100	0xE3

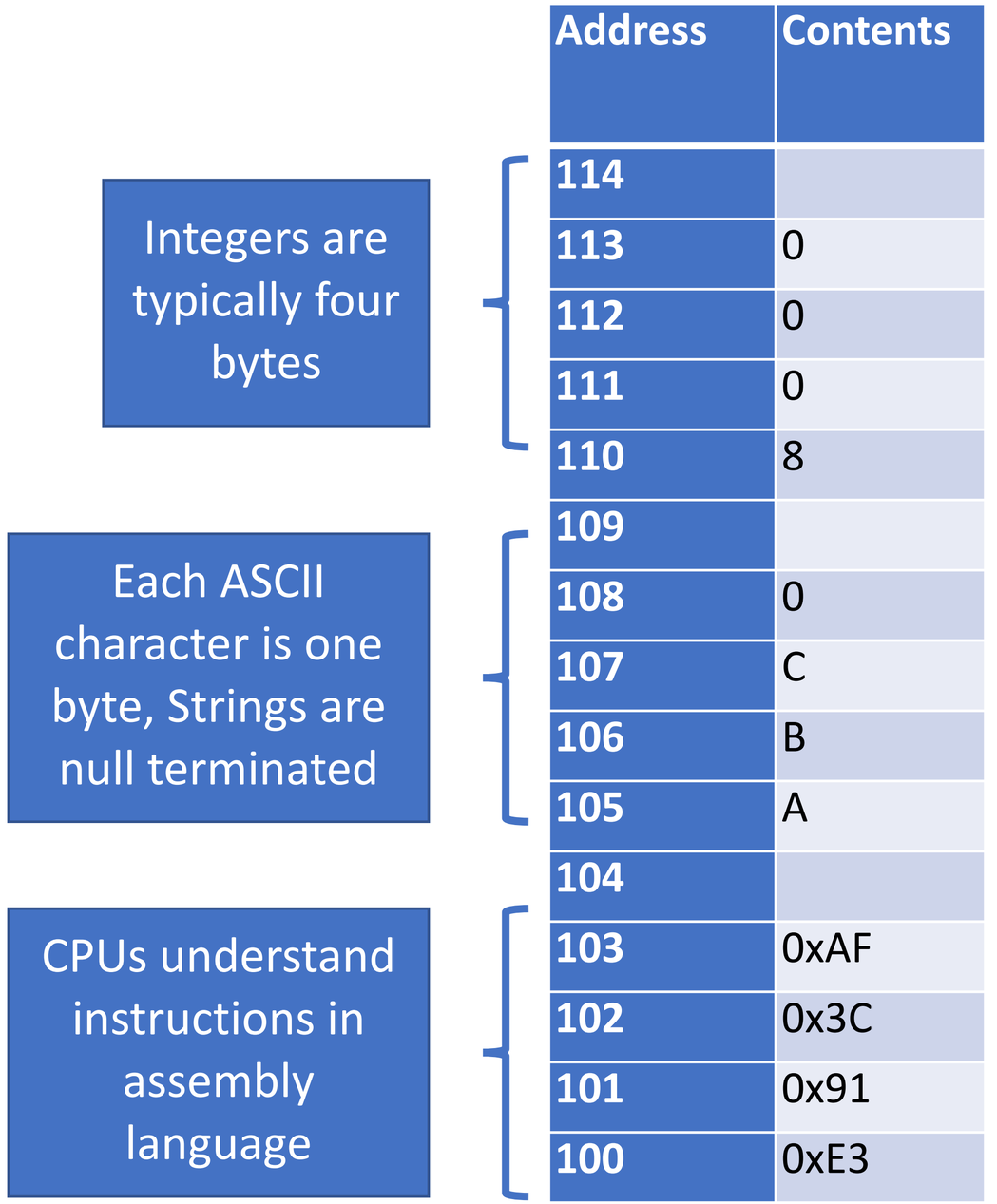
What is Memory?

Memory is a “spreadsheet" with a single column

- Every row has a number, or address
- Every cell holds 1 byte of data

All data and code are held in memory

```
int my_num = 8;
```



What is Memory?

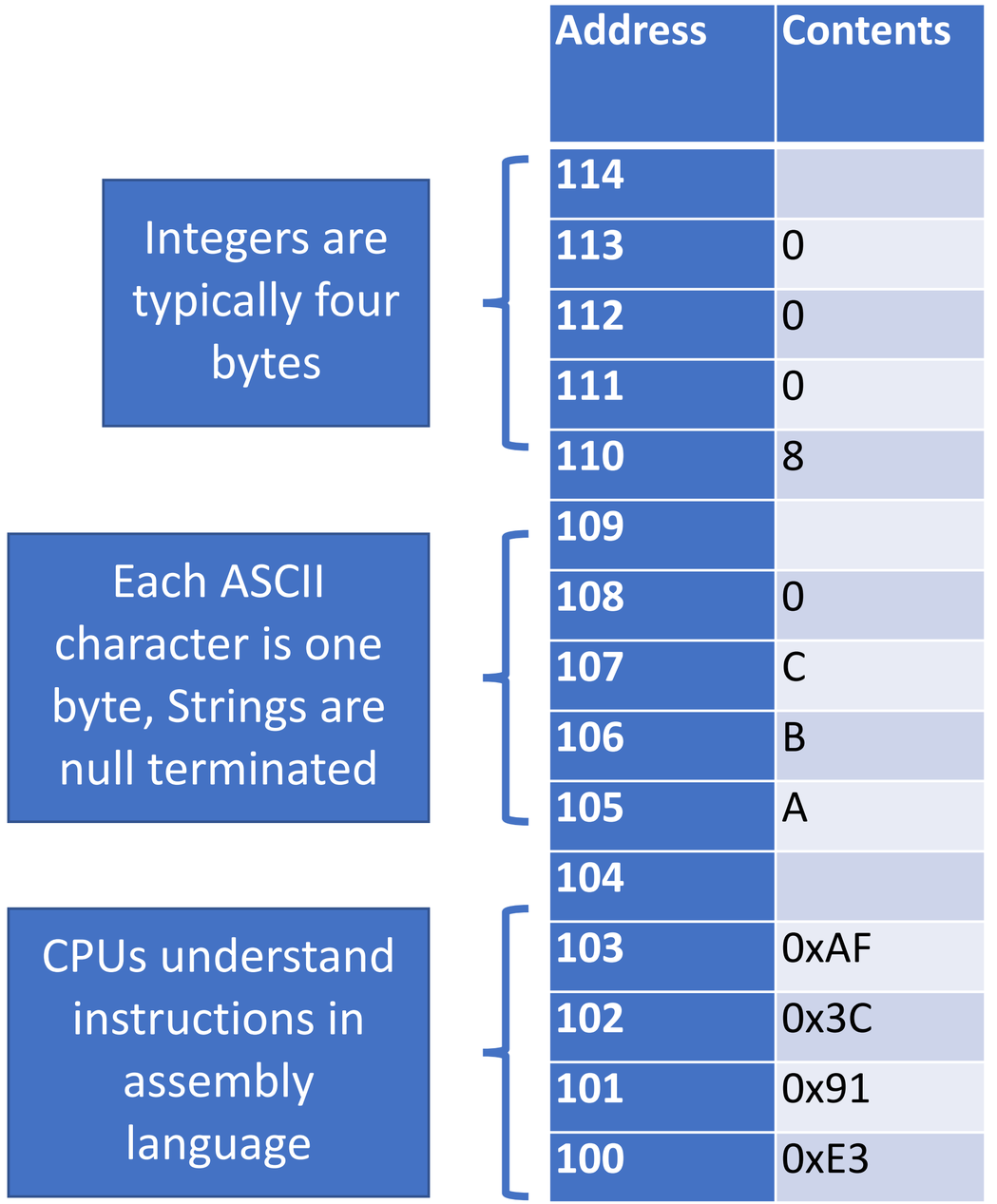
Memory is a “spreadsheet" with a single column

- Every row has a number, or address
- Every cell holds 1 byte of data

All data and code are held in memory

```
int my_num = 8;

String my_str = “ABC”;
```



What is Memory?

Memory is a “spreadsheet" with a single column

- Every row has a number, or address
- Every cell holds 1 byte of data

All data and code are held in memory

```
int my_num = 8;

String my_str = “ABC”;

while (my_num > 0) my_num--;
```

Integers are typically four bytes

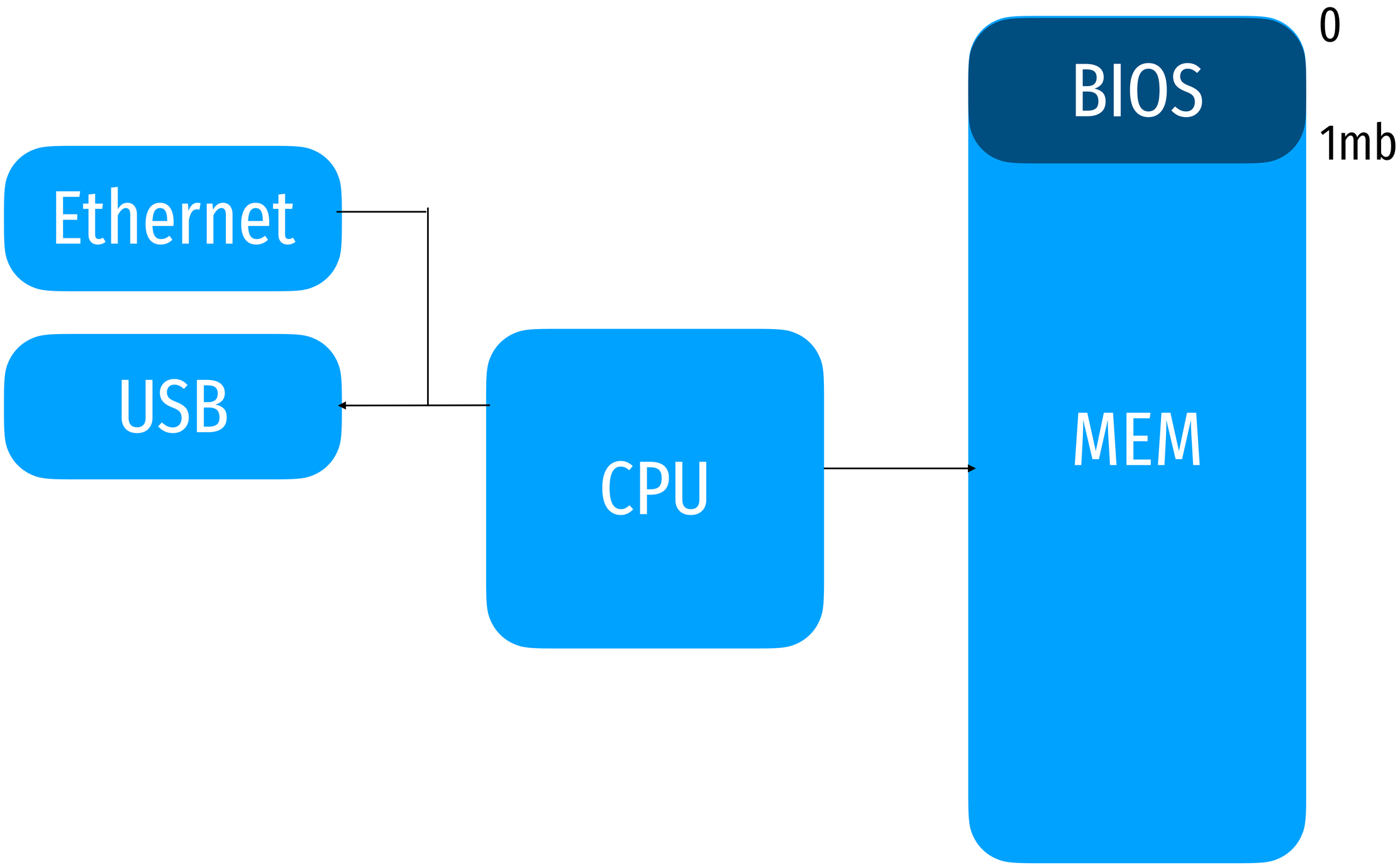
Each ASCII character is one byte, Strings are null terminated

CPUs understand instructions in assembly language

Address	Contents
114	
113	0
112	0
111	0
110	8
109	
108	0
107	C
106	B
105	A
104	
103	0xAF
102	0x3C
101	0x91
100	0xE3

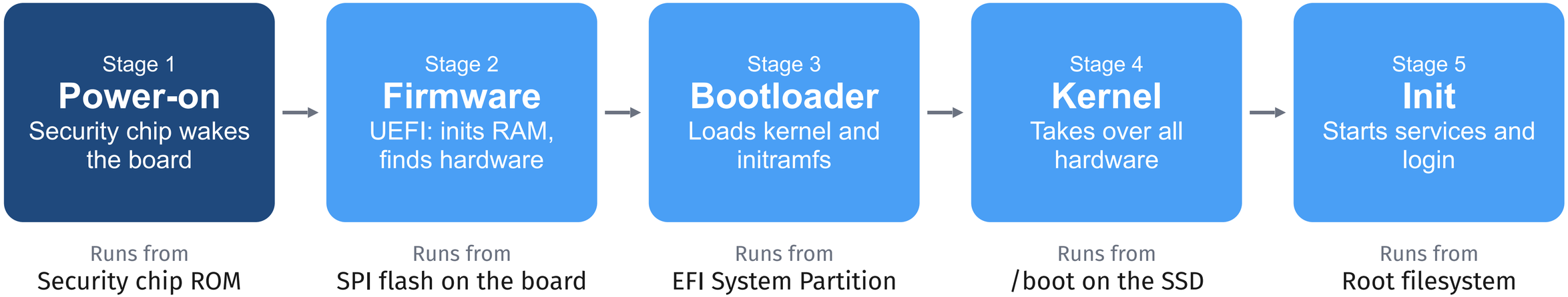
How does a computer boot?

https://youtu.be/MsKb0gR-4AM?t=36


0
BIOS
1mb
Ethernet
USB
CPU
MEM

How a computer boots: the big picture

Each stage does just enough to load, check, and start the next one: “relay race”



Capability grows

No RAM → RAM → storage and screen → every driver → users

Trust is handed on

Each stage checks the next stage’s signature before starting it.

Stage 1: Before the first instruction runs

1. **You press power**
   The embedded controller switches on the power rails in order and holds the CPU in reset.

2. **A security processor boots first**
   Intel CSME or AMD PSP starts from its own ROM and loads its firmware from the SPI flash chip.

3. **Firmware is checked**
   Intel Boot Guard or AMD Platform Secure Boot verifies the first firmware block against a key fused into the chip.

4. **Microcode is patched**
   On Intel, the CPU loads a microcode update listed in the flash's Firmware Interface Table.

5. **Reset is released**
   Core 0 fetches its first instruction from 0xFFFF_FFF0. The other cores stay asleep.

The x86 cores are not the first processor to run

On AMD Ryzen, the PSP (a small Arm core) even trains the DRAM before the x86 cores start.

Stage 2: the four UEFI phases

SEC
Security

- Fetch first instruction at 0xFFFF_FFF0
- Switch real → 32-bit → 64-bit mode
- Use CPU cache as RAM (no DRAM yet)
- Find and verify PEI

PEI
Pre-EFI init

- Run memory init code (Intel FSP-M)
- Train DRAM: timing and voltage per channel
- Move the stack from cache into real RAM
- Copy and decompress firmware into RAM

DXE
Driver execution

- Init chipset, power, clocks (FSP-S)
- Set up interrupts and timers
- Wake other cores, patch microcode, park them
- Enumerate PCIe, assign addresses
- Load drivers: USB, NVMe, GPU
- Lock down SMM
- Build ACPI tables and memory map

BDS
Boot device selection

- Read boot order from NVRAM
- Mount the EFI System Partition
- Secure Boot: check the loader’s signature
- Record hashes in the TPM
- Run the OS loader

FSP = Intel Firmware Support Package, the binary “blob” Intel ships for silicon init (FSP-T, FSP-M, FSP-S). AMD’s equivalent is AGESA.

Handing off to the operating system

Stage 3
Bootloader
GRUB, systemd-boot

- Reads its config, shows the menu
- Loads kernel and initramfs into RAM
- Passes the kernel command line
- shim checks the kernel's signature

Stage 4
Kernel
vmlinuz

- ExitBootServices(): firmware steps aside
- Decompresses itself, builds page tables
- Reads the memory map and ACPI tables
- Wakes the other cores (again)
- Sets up interrupts, timers, PCI, drivers

Stage 5
Early userspace → init
initramfs, then systemd

- Loads disk drivers, unlocks LUKS, finds LVM
- Switches to the real root filesystem
- Starts PID 1 (systemd)
- Services, then the login screen

Windows, same shape: `bootmgfw.efi → winload.efi → ntoskrnl.exe → smss.exe → winlogon.exe`

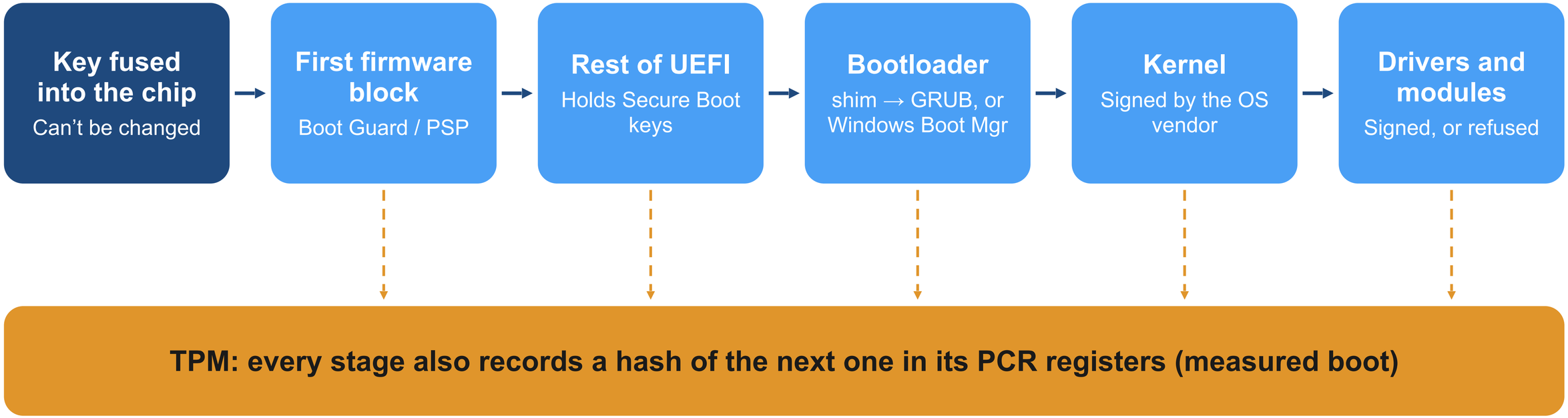
So much can go wrong in this process

Modern devices use a “secure boot” process to ensure that the steps complete as designed.

This process is bootstrapped by a “root of trust” in your hardware.

Chain of trust: each stage checks the next

Each arrow means “check the signature, then jump”. A bad signature stops the boot.



Verified boot

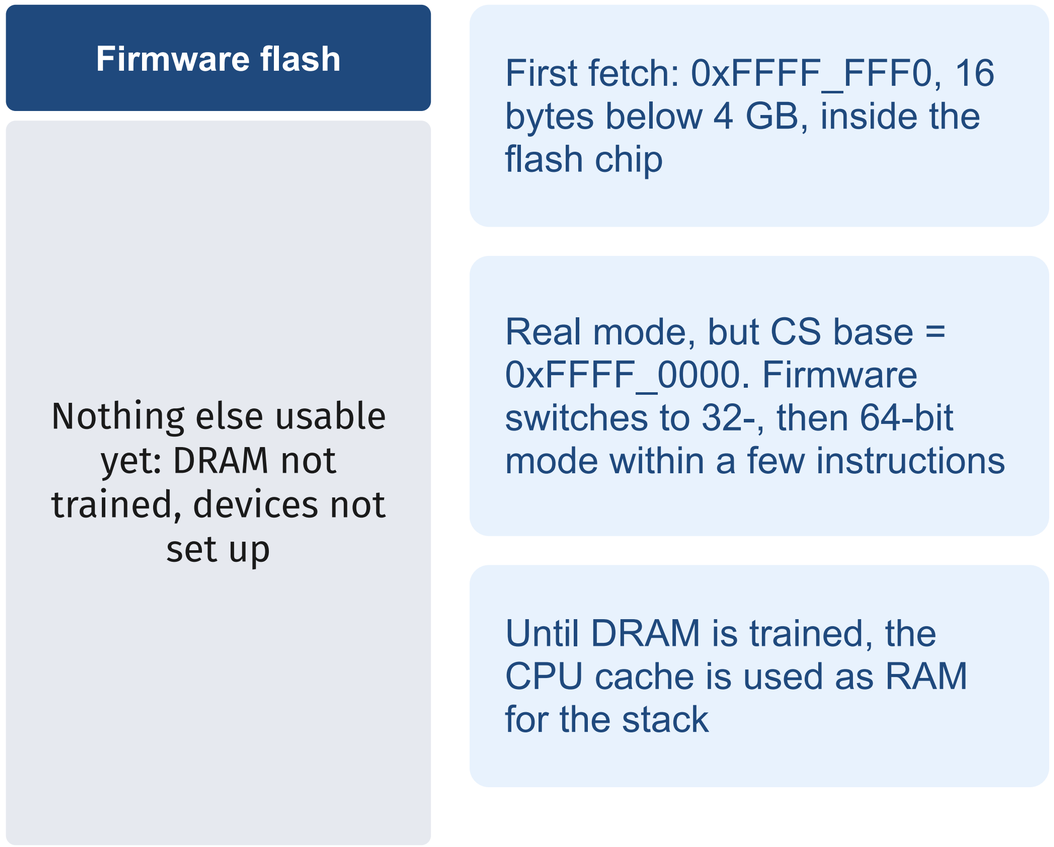
Refuse to run anything unsigned. This is UEFI Secure Boot.

Measured boot

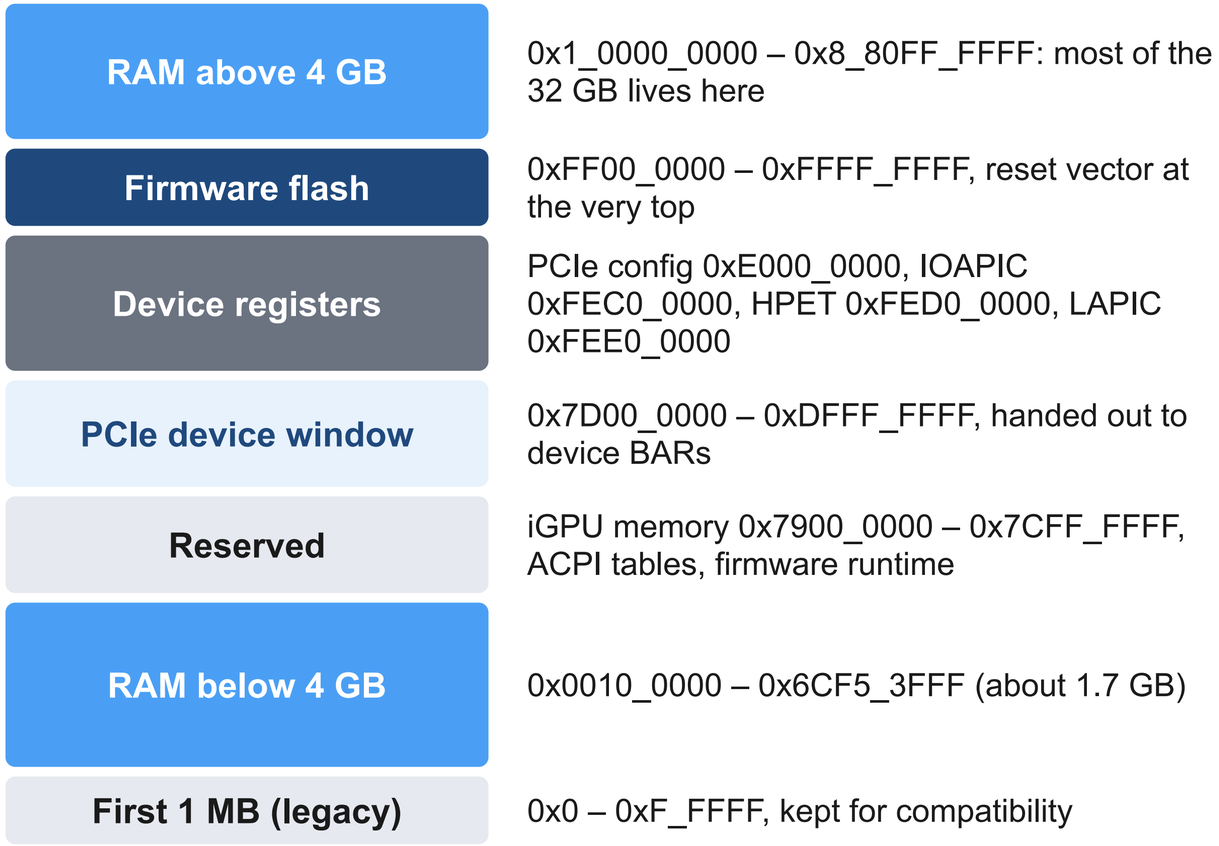
Keep a tamper-proof record. BitLocker and LUKS release disk keys only if it matches.

Memory at reset vs. when the OS takes over

At reset



When the OS takes over (Intel NUC9i5, 32 GB)



Addresses from this machine’s boot log (see “Reading a real boot log”). Not to scale.

Reading a real boot log

Intel NUC9i5, AMI firmware, Ubuntu kernel 5.3. Most early lines report what earlier stages left behind.

Log line	What it tells you	Set up by
`e820: update [mem 0x68146000-0x68146fff] usable ==> reserved`	Kernel adjusting the memory map the firmware handed over	**Firmware**
`ACPI: DSDT 0x000000006F08F000 042561 (v02 INTEL NUC9i5FN …)`	Hardware description tables, written by the firmware	**Firmware**
`RAMDISK: [mem 0x3ce54000-0x3fffffff]`	Where GRUB placed the initramfs in RAM	**Bootloader**
`Kernel command line: BOOT_IMAGE=/boot/vmlinuz-5.3.0-64-generic root=/dev/mapper/vgubuntu-root …`	Options from GRUB; the root filesystem is on LVM	**Bootloader**
`Using ACPI (MADT) for SMP configuration information` `smpboot: Allowing 12 CPUs, 0 hotplug CPUs`	Kernel learns the cores from a firmware table, then wakes them	**Firmware → kernel**
`[mem 0x7d000000-0xdfffffff] available for PCI devices`	The hole below 4 GB for device registers	**Firmware**
`Memory: 32410828K/33258444K available`	About 830 MB held back by firmware and kernel	**Kernel**

```
[    0.000000] microcode: microcode updated early to revision 0xca, date = 2019-10-03
[    0.000000] Linux version 5.3.0-64-generic (buildd@lcy01-amd64-026) (gcc version 9.2.1 20191008 (Ubuntu 9.2.1-9ubuntu2)) #58-Ubuntu SMP Fri Jul 10 19:33:51 UTC 2020 (Ubuntu 5.3.0-64.58-generic 5.3.18)
[    0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-5.3.0-64-generic root=/dev/mapper/vgubuntu-root ro quiet splash vt.handoff=7
[    0.000000] KERNEL supported cpus:
[    0.000000]   Intel GenuineIntel
[    0.000000]   AMD AuthenticAMD
[    0.000000]   Hygon HygonGenuine
[    0.000000]   Centaur CentaurHauls
[    0.000000]   zhaoxin   Shanghai
[    0.000000] x86/fpu: Supporting XSAVE feature 0x001: 'x87 floating point registers'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x002: 'SSE registers'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x004: 'AVX registers'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x008: 'MPX bounds registers'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x010: 'MPX CSR'
[    0.000000] x86/fpu: xstate_offset[2]:  576, xstate_sizes[2]:  256
[    0.000000] x86/fpu: xstate_offset[3]:  832, xstate_sizes[3]:   64
[    0.000000] x86/fpu: xstate_offset[4]:  896, xstate_sizes[4]:   64
[    0.000000] x86/fpu: Enabled xstate features 0x1f, context size is 960 bytes, using 'compacted' format.
[    0.000000] BIOS-provided physical RAM map:
[    0.000000] BIOS-e820: [mem 0x0000000000000000-0x000000000009efff] usable
[    0.000000] BIOS-e820: [mem 0x000000000009f000-0x00000000000fffff] reserved
[    0.000000] BIOS-e820: [mem 0x0000000000100000-0x000000006cf53fff] usable
[    0.000000] BIOS-e820: [mem 0x000000006cf54000-0x000000006f057fff] reserved
[    0.000000] BIOS-e820: [mem 0x000000006f058000-0x000000006f069fff] ACPI data
[    0.000000] BIOS-e820: [mem 0x000000006f06a000-0x000000006f06afff] reserved
[    0.000000] BIOS-e820: [mem 0x000000006f06b000-0x000000006f0d7fff] ACPI data
[    0.000000] BIOS-e820: [mem 0x000000006f0d8000-0x000000006f1b1fff] ACPI NVS
[    0.000000] BIOS-e820: [mem 0x000000006f1b2000-0x000000006fc4dfff] reserved
[    0.000000] BIOS-e820: [mem 0x000000006fc4e000-0x000000006fc4efff] usable
[    0.000000] BIOS-e820: [mem 0x000000006fc4f000-0x000000007cffffff] reserved
[    0.000000] BIOS-e820: [mem 0x00000000e0000000-0x00000000efffffff] reserved
[    0.000000] BIOS-e820: [mem 0x00000000fe000000-0x00000000fe010fff] reserved
[    0.000000] BIOS-e820: [mem 0x00000000fec00000-0x00000000fec00fff] reserved
[    0.000000] BIOS-e820: [mem 0x00000000fed00000-0x00000000fed03fff] reserved
[    0.000000] BIOS-e820: [mem 0x00000000fee00000-0x00000000fee00fff] reserved
[    0.000000] BIOS-e820: [mem 0x00000000ff000000-0x00000000ffffffff] reserved
[    0.000000] BIOS-e820: [mem 0x0000000100000000-0x0000000880ffffff] usable
[    0.000000] NX (Execute Disable) protection: active
[    0.000000] e820: update [mem 0x657d3018-0x657e3457] usable ==> usable
[    0.000000] e820: update [mem 0x657d3018-0x657e3457] usable ==> usable
[    0.000000] extended physical RAM map:
[    0.000000] reserve setup_data: [mem 0x0000000000000000-0x000000000009efff] usable
[    0.000000] reserve setup_data: [mem 0x000000000009f000-0x00000000000fffff] reserved
[    0.000000] reserve setup_data: [mem 0x0000000000100000-0x00000000657d3017] usable
[    0.000000] reserve setup_data: [mem 0x00000000657d3018-0x00000000657e3457] usable
[    0.000000] reserve setup_data: [mem 0x00000000657e3458-0x000000006cf53fff] usable
[    0.000000] reserve setup_data: [mem 0x000000006cf54000-0x000000006f057fff] reserved
[    0.000000] reserve setup_data: [mem 0x000000006f058000-0x000000006f069fff] ACPI data
[    0.000000] reserve setup_data: [mem 0x000000006f06a000-0x000000006f06afff] reserved
[    0.000000] reserve setup_data: [mem 0x000000006f06b000-0x000000006f0d7fff] ACPI data
[    0.000000] reserve setup_data: [mem 0x000000006f0d8000-0x000000006f1b1fff] ACPI NVS
[    0.000000] reserve setup_data: [mem 0x000000006f1b2000-0x000000006fc4dfff] reserved
[    0.000000] reserve setup_data: [mem 0x000000006fc4e000-0x000000006fc4efff] usable
[    0.000000] reserve setup_data: [mem 0x000000006fc4f000-0x000000007cffffff] reserved
[    0.000000] reserve setup_data: [mem 0x00000000e0000000-0x00000000efffffff] reserved
[    0.000000] reserve setup_data: [mem 0x00000000fe000000-0x00000000fe010fff] reserved
[    0.000000] reserve setup_data: [mem 0x00000000fec00000-0x00000000fec00fff] reserved
[    0.000000] reserve setup_data: [mem 0x00000000fed00000-0x00000000fed03fff] reserved
[    0.000000] reserve setup_data: [mem 0x00000000fee00000-0x00000000fee00fff] reserved
[    0.000000] reserve setup_data: [mem 0x00000000ff000000-0x00000000ffffffff] reserved
[    0.000000] reserve setup_data: [mem 0x0000000100000000-0x0000000880ffffff] usable
[    0.000000] efi: EFI v2.70 by American Megatrends
[    0.000000] efi:  ACPI=0x6f117000  ACPI 2.0=0x6f117014  TPMFinalLog=0x6f11f000  SMBIOS=0x6f9de000  SMBIOS 3.0=0x6f9dd000  MEMATTR=0x66372018  ESRT=0x68146518  TPMEventLog=0x657e4018
[    0.000000] secureboot: Secure boot enabled
[    0.000000] Kernel is locked down from EFI secure boot; see man kernel_lockdown.7
[    0.000000] SMBIOS 3.2.0 present.
[    0.000000] DMI: Intel(R) Client Systems NUC10i7FNH/NUC10i7FNB, BIOS FNCML357.0032.2019.1021.1624 10/21/2019
[    0.000000] tsc: Detected 1600.000 MHz processor
[    0.000376] tsc: Detected 1599.960 MHz TSC
[    0.000376] e820: update [mem 0x00000000-0x00000fff] usable ==> reserved
[    0.000379] e820: remove [mem 0x000a0000-0x000fffff] usable
[    0.000391] last_pfn = 0x881000 max_arch_pfn = 0x400000000
[    0.000398] MTRR default type: write-back
[    0.000399] MTRR fixed ranges enabled:
[    0.000401]   00000-9FFFF write-back
[    0.000403]   A0000-BFFFF uncachable
[    0.000405]   C0000-FFFFF write-protect
```

```
[    0.000406] MTRR variable ranges enabled:
[    0.000409]   0 base 0080000000 mask 7F80000000 uncachable
[    0.000410]   1 base 007C000000 mask 7FFC000000 uncachable
[    0.000412]   2 base 007A000000 mask 7FFE000000 uncachable
[    0.000413]   3 base 0079000000 mask 7FFF000000 uncachable
[    0.000415]   4 base 0078800000 mask 7FFF800000 uncachable
[    0.000417]   5 base 2000000000 mask 6000000000 uncachable
[    0.000418]   6 base 1000000000 mask 7000000000 uncachable
[    0.000420]   7 base 4000000000 mask 4000000000 uncachable
[    0.000421]   8 disabled
[    0.000422]   9 disabled
[    0.001256] x86/PAT: Configuration [0-7]: WB  WC  UC- UC  WB  WP  UC- WT
[    0.001570] last_pfn = 0x6fc4f max_arch_pfn = 0x400000000
[    0.021713] esrt: Reserving ESRT space from 0x0000000068146518 to 0x0000000068146550.
[    0.021729] e820: update [mem 0x68146000-0x68146fff] usable ==> reserved
[    0.021847] check: Scanning 1 areas for low memory corruption
[    0.021853] Using GB pages for direct mapping
[    0.022564] RAMDISK: [mem 0x3ce54000-0x3fffdfff]
[    0.022580] ACPI: Early table checksum verification disabled
[    0.022584] ACPI: RSDP 0x000000006F117014 000024 (v02 INTEL )
[    0.022589] ACPI: XSDT 0x000000006F116728 0000CC (v01 INTEL  NUC9i5FN 00000020 AMI  01000013)
[    0.022598] ACPI: FACP 0x000000006F0D2000 000114 (v06 INTEL  NUC9i5FN 00000020 AMI  00010013)
[    0.022606] ACPI: DSDT 0x000000006F08F000 042561 (v02 INTEL  NUC9i5FN 00000020 INTL 20160527)
[    0.022610] ACPI: FACS 0x000000006F1B1000 000040
[    0.022614] ACPI: MCFG 0x000000006F0D5000 00003C (v01 INTEL  NUC9i5FN 00000020 MSFT 00000097)
[    0.022618] ACPI: SSDT 0x000000006F0D3000 001B4A (v02 INTEL  NUC9i5FN 00000020 INTL 20160527)
[    0.022623] ACPI: FIDT 0x000000006F08E000 00009C (v01 INTEL  NUC9i5FN 00000020 AMI  00010013)
[    0.022627] ACPI: SSDT 0x000000006F08A000 0031C6 (v02 INTEL  NUC9i5FN 00000020 INTL 20160527)
[    0.022631] ACPI: HPET 0x000000006F0D7000 000038 (v01 INTEL  NUC9i5FN 00000020 AMI  01000013)
[    0.022635] ACPI: SSDT 0x000000006F086000 0033B4 (v02 INTEL  NUC9i5FN 00000020 INTL 20160527)
[    0.022639] ACPI: SSDT 0x000000006F084000 00147B (v02 INTEL  NUC9i5FN 00000020 INTL 20160527)
[    0.022643] ACPI: SSDT 0x000000006F080000 0032BD (v02 INTEL  NUC9i5FN 00000020 INTL 20160527)
[    0.022648] ACPI: NHLT 0x000000006F0D6000 00002D (v00 INTEL  NUC9i5FN 00000020 AMI  01000013)
[    0.022652] ACPI: LPIT 0x000000006F07F000 000094 (v01 INTEL  NUC9i5FN 00000020 AMI  01000013)
[    0.022656] ACPI: SSDT 0x000000006F07B000 002720 (v02 INTEL  NUC9i5FN 00000020 INTL 20160527)
[    0.022660] ACPI: SSDT 0x000000006F07A000 00087C (v02 INTEL  NUC9i5FN 00000020 INTL 20160527)
[    0.022664] ACPI: DBGP 0x000000006F079000 000034 (v01 INTEL  NUC9i5FN 00000020 AMI  01000013)
[    0.022668] ACPI: DBG2 0x000000006F078000 000054 (v00 INTEL  NUC9i5FN 00000020 AMI  01000013)
[    0.022672] ACPI: SSDT 0x000000006F076000 001B66 (v02 INTEL  NUC9i5FN 00000020 INTL 20160527)
[    0.022677] ACPI: TPM2 0x000000006F074000 00004C (v04 INTEL  NUC9i5FN 00000020 AMI  00000000)
[    0.022681] ACPI: DMAR 0x000000006F075000 0000A8 (v01 INTEL  NUC9i5FN 00000020      01000013)
[    0.022685] ACPI: WSMT 0x000000006F07E000 000028 (v01 INTEL  NUC9i5FN 00000020 AMI  00010013)
[    0.022689] ACPI: APIC 0x000000006F073000 0000F4 (v04 INTEL  NUC9i5FN 00000020 AMI  00010013)
[    0.022693] ACPI: FPDT 0x000000006F072000 000044 (v01 INTEL  NUC9i5FN 00000020 AMI  01000013)
[    0.022707] ACPI: Local APIC address 0xfee00000
[    0.023236] No NUMA configuration found
[    0.023238] Faking a node at [mem 0x0000000000000000-0x0000000880ffffff]
[    0.023254] NODE_DATA(0) allocated [mem 0x880fd5000-0x880ffffff]
[    0.023698] Zone ranges:
[    0.023700]   DMA      [mem 0x0000000000001000-0x0000000000ffffff]
[    0.023702]   DMA32    [mem 0x0000000001000000-0x00000000ffffffff]
[    0.023703]   Normal   [mem 0x0000000100000000-0x0000000880ffffff]
[    0.023705]   Device   empty
[    0.023706] Movable zone start for each node
[    0.023711] Early memory node ranges
[    0.023713]   node   0: [mem 0x0000000000001000-0x000000000009efff]
[    0.023714]   node   0: [mem 0x0000000000100000-0x000000006cf53fff]
[    0.023716]   node   0: [mem 0x000000006fc4e000-0x000000006fc4efff]
[    0.023717]   node   0: [mem 0x0000000100000000-0x0000000880ffffff]
[    0.024270] Zeroed struct page in unavailable ranges: 41229 pages
[    0.024272] Initmem setup node 0 [mem 0x0000000000001000-0x0000000880ffffff]
[    0.024274] On node 0 totalpages: 8314611
[    0.024276]   DMA zone: 64 pages used for memmap
[    0.024277]   DMA zone: 25 pages reserved
[    0.024278]   DMA zone: 3998 pages, LIFO batch:0
[    0.024379]   DMA32 zone: 6910 pages used for memmap
[    0.024380]   DMA32 zone: 442197 pages, LIFO batch:63
[    0.039990]   Normal zone: 122944 pages used for memmap
[    0.039991]   Normal zone: 7868416 pages, LIFO batch:63
[    0.207139] Reserving Intel graphics memory at [mem 0x79000000-0x7cffffff]
[    0.207882] ACPI: PM-Timer IO Port: 0x1808
[    0.207884] ACPI: Local APIC address 0xfee00000
[    0.207894] ACPI: LAPIC_NMI (acpi_id[0x01] high edge lint[0x1])
[    0.207895] ACPI: LAPIC_NMI (acpi_id[0x02] high edge lint[0x1])
[    0.207896] ACPI: LAPIC_NMI (acpi_id[0x03] high edge lint[0x1])
[    0.207897] ACPI: LAPIC_NMI (acpi_id[0x04] high edge lint[0x1])
[    0.207897] ACPI: LAPIC_NMI (acpi_id[0x05] high edge lint[0x1])
[    0.207898] ACPI: LAPIC_NMI (acpi_id[0x06] high edge lint[0x1])
[    0.207899] ACPI: LAPIC_NMI (acpi_id[0x07] high edge lint[0x1])
[    0.207900] ACPI: LAPIC_NMI (acpi_id[0x08] high edge lint[0x1])
[    0.207901] ACPI: LAPIC_NMI (acpi_id[0x09] high edge lint[0x1])
[    0.207902] ACPI: LAPIC_NMI (acpi_id[0x0a] high edge lint[0x1])
[    0.207902] ACPI: LAPIC_NMI (acpi_id[0x0b] high edge lint[0x1])
[    0.207903] ACPI: LAPIC_NMI (acpi_id[0x0c] high edge lint[0x1])
[    0.207957] IOAPIC[0]: apic_id 2, version 32, address 0xfec00000, GSI 0-119
[    0.207960] ACPI: INT_SRC_OVR (bus 0 bus_irq 0 global_irq 2 dfl dfl)
[    0.207962] ACPI: INT_SRC_OVR (bus 0 bus_irq 9 global_irq 9 high level)
[    0.207963] ACPI: IRQ0 used by override.
[    0.207965] ACPI: IRQ9 used by override.
[    0.207968] Using ACPI (MADT) for SMP configuration information
[    0.207969] ACPI: HPET id: 0x8086a201 base: 0xfed00000
[    0.207974] smpboot: Allowing 12 CPUs, 0 hotplug CPUs
[    0.208001] PM: Registered nosave memory: [mem 0x00000000-0x00000fff]
[    0.208003] PM: Registered nosave memory: [mem 0x0009f000-0x000fffff]
[    0.208005] PM: Registered nosave memory: [mem 0x657d3000-0x657d3fff]
[    0.208008] PM: Registered nosave memory: [mem 0x657e3000-0x657e3fff]
[    0.208010] PM: Registered nosave memory: [mem 0x68146000-0x68146fff]
[    0.208012] PM: Registered nosave memory: [mem 0x6cf54000-0x6f057fff]
[    0.208013] PM: Registered nosave memory: [mem 0x6f058000-0x6f069fff]
[    0.208014] PM: Registered nosave memory: [mem 0x6f06a000-0x6f06afff]
[    0.208015] PM: Registered nosave memory: [mem 0x6f06b000-0x6f0d7fff]
[    0.208015] PM: Registered nosave memory: [mem 0x6f0d8000-0x6f1b1fff]
[    0.208016] PM: Registered nosave memory: [mem 0x6f1b2000-0x6fc4dfff]
[    0.208018] PM: Registered nosave memory: [mem 0x6fc4f000-0x7cffffff]
[    0.208019] PM: Registered nosave memory: [mem 0x7d000000-0xdfffffff]
[    0.208020] PM: Registered nosave memory: [mem 0xe0000000-0xefffffff]
[    0.208021] PM: Registered nosave memory: [mem 0xf0000000-0xfdffffff]
[    0.208022] PM: Registered nosave memory: [mem 0xfe000000-0xfe010fff]
[    0.208022] PM: Registered nosave memory: [mem 0xfe011000-0xfebfffff]
[    0.208023] PM: Registered nosave memory: [mem 0xfec00000-0xfec00fff]
[    0.208024] PM: Registered nosave memory: [mem 0xfec01000-0xfecfffff]
[    0.208025] PM: Registered nosave memory: [mem 0xfed00000-0xfed03fff]
[    0.208025] PM: Registered nosave memory: [mem 0xfed04000-0xfedfffff]
[    0.208026] PM: Registered nosave memory: [mem 0xfee00000-0xfee00fff]
[    0.208027] PM: Registered nosave memory: [mem 0xfee01000-0xfeffffff]
[    0.208028] PM: Registered nosave memory: [mem 0xff000000-0xffffffff]
[    0.208030] [mem 0x7d000000-0xdfffffff] available for PCI devices
[    0.208033] Booting paravirtualized kernel on bare hardware
[    0.208036] clocksource: refined-jiffies: mask: 0xffffffff max_cycles: 0xffffffff, max_idle_ns: 7645519600211568 ns
[    0.208045] setup_percpu: NR_CPUS:8192 nr_cpumask_bits:12 nr_cpu_ids:12 nr_node_ids:1
[    0.208519] percpu: Embedded 54 pages/cpu s184320 r8192 d28672 u262144
[    0.208530] pcpu-alloc: s184320 r8192 d28672 u262144 alloc=1*2097152
[    0.208531] pcpu-alloc: [0] 00 01 02 03 04 05 06 07 [0] 08 09 10 11 -- -- -- --
[    0.208576] Built 1 zonelists, mobility grouping on.  Total pages: 8184668
[    0.208577] Policy zone: Normal
[    0.208579] Kernel command line: BOOT_IMAGE=/boot/vmlinuz-5.3.0-64-generic root=/dev/mapper/vgubuntu-root ro quiet splash vt.handoff=7
[    0.211269] Dentry cache hash table entries: 4194304 (order: 13, 33554432 bytes, linear)
[    0.212573] Inode-cache hash table entries: 2097152 (order: 12, 16777216 bytes, linear)
[    0.212724] mem auto-init: stack:off, heap alloc:on, heap free:off
[    0.217959] Calgary: detecting Calgary via BIOS EBDA area
[    0.217961] Calgary: Unable to locate Rio Grande table in EBDA - bailing!
[    0.326171] Memory: 32410828K/33258444K available (14339K kernel code, 2387K rwdata, 4732K rodata, 2680K init, 5040K bss, 847616K reserved, 0K cma-reserved)
[    0.326180] random: get_random_u64 called from kmem_cache_open+0x2d/0x410 with crng_init=0
[    0.326431] SLUB: HWalign=64, Order=0-3, MinObjects=0, CPUs=12, Nodes=1
[    0.326453] ftrace: allocating 43632 entries in 171 pages
[    0.355621] rcu: Hierarchical RCU implementation.
[    0.355623] rcu:     RCU restricting CPUs from NR_CPUS=8192 to nr_cpu_ids=12.
[    0.355624]  Tasks RCU enabled.
[    0.355626] rcu: RCU calculated value of scheduler-enlistment delay is 25 jiffies.
[    0.355627] rcu: Adjusting geometry for rcu_fanout_leaf=16, nr_cpu_ids=12
[    0.361328] NR_IRQS: 524544, nr_irqs: 2152, preallocated irqs: 16
[    0.361842] random: crng done (trusting CPU's manufacturer)
[    0.361879] vt handoff: transparent VT on vt#7
[    0.361889] Console: colour dummy device 80x25
[    0.361895] printk: console [tty0] enabled
[    0.361918] ACPI: Core revision 20190703
[    0.362625] clocksource: hpet: mask: 0xffffffff max_cycles: 0xffffffff, max_idle_ns: 79635855245 ns
[    0.362761] APIC: Switch to symmetric I/O mode setup
[    0.362764] DMAR: Host address width 39
[    0.362766] DMAR: DRHD base: 0x000000fed90000 flags: 0x0
[    0.362774] DMAR: dmar0: reg_base_addr fed90000 ver 1:0 cap 1c0000c40660462 ecap 19e2ff0505e
```

```
[    0.696153] PCI host bridge to bus 0000:00
[    0.696156] pci_bus 0000:00: root bus resource [io  0x0000-0x0cf7 window]
[    0.696158] pci_bus 0000:00: root bus resource [io  0x0d00-0xffff window]
[    0.696159] pci_bus 0000:00: root bus resource [mem 0x000a0000-0x000bffff window]
[    0.696161] pci_bus 0000:00: root bus resource [mem 0x000e0000-0x000e3fff window]
[    0.696162] pci_bus 0000:00: root bus resource [mem 0x000e4000-0x000e7fff window]
[    0.696163] pci_bus 0000:00: root bus resource [mem 0x000e8000-0x000ebfff window]
[    0.696165] pci_bus 0000:00: root bus resource [mem 0x000ec000-0x000efff window]
[    0.696166] pci_bus 0000:00: root bus resource [mem 0x000f0000-0x000fffff window]
[    0.696167] pci_bus 0000:00: root bus resource [mem 0x7d000000-0xdfffffff window]
[    0.696169] pci_bus 0000:00: root bus resource [mem 0x4000000000-0x7fffffffff window]
[    0.696170] pci_bus 0000:00: root bus resource [mem 0xfc800000-0xfe7fffff window]
[    0.696172] pci_bus 0000:00: root bus resource [bus 00-fe]
[    0.696187] pci 0000:00:00.0: [8086:9b51] type 00 class 0x060000
[    0.696942] pci 0000:00:02.0: [8086:9bca] type 00 class 0x030000
[    0.696959] pci 0000:00:02.0: reg 0x10: [mem 0x6022000000-0x6022ffffff 64bit]
[    0.696967] pci 0000:00:02.0: reg 0x18: [mem 0x4000000000-0x400fffffff 64bit pref]
[    0.696973] pci 0000:00:02.0: reg 0x20: [io  0x3000-0x303f]
[    0.697334] pci 0000:00:08.0: [8086:1911] type 00 class 0x088000
[    0.697352] pci 0000:00:08.0: reg 0x10: [mem 0x6023120000-0x6023120fff 64bit]
[    0.697682] pci 0000:00:12.0: [8086:02f9] type 00 class 0x118000
[    0.697708] pci 0000:00:12.0: reg 0x10: [mem 0x602311f000-0x602311ffff 64bit]
[    0.698071] pci 0000:00:14.0: [8086:02ed] type 00 class 0x0c0330
[    0.698096] pci 0000:00:14.0: reg 0x10: [mem 0x6023100000-0x602310ffff 64bit]
[    0.698176] pci 0000:00:14.0: PME# supported from D3hot D3cold
[    0.698639] pci 0000:00:14.2: [8086:02ef] type 00 class 0x050000
[    0.698662] pci 0000:00:14.2: reg 0x10: [mem 0x6023118000-0x6023119fff 64bit]
[    0.698675] pci 0000:00:14.2: reg 0x18: [mem 0x602311e000-0x602311efff 64bit]
[    0.699033] pci 0000:00:14.3: [8086:02f0] type 00 class 0x028000
[    0.699130] pci 0000:00:14.3: reg 0x10: [mem 0x6023114000-0x6023117fff 64bit]
[    0.699383] pci 0000:00:14.3: PME# supported from D0 D3hot D3cold
[    0.699892] pci 0000:00:15.0: [8086:02e8] type 00 class 0x0c8000
[    0.700034] pci 0000:00:15.0: reg 0x10: [mem 0x00000000-0x00000fff 64bit]
[    0.700757] pci 0000:00:15.2: [8086:02ea] type 00 class 0x0c8000
[    0.700899] pci 0000:00:15.2: reg 0x10: [mem 0x00000000-0x00000fff 64bit]
[    0.701605] pci 0000:00:16.0: [8086:02e0] type 00 class 0x078000
[    0.701636] pci 0000:00:16.0: reg 0x10: [mem 0x602311b000-0x602311bfff 64bit]
[    0.701726] pci 0000:00:16.0: PME# supported from D3hot
[    0.702164] pci 0000:00:17.0: [8086:02d3] type 00 class 0x010601
[    0.702186] pci 0000:00:17.0: reg 0x10: [mem 0x96220000-0x96221fff]
[    0.702195] pci 0000:00:17.0: reg 0x14: [mem 0x96223000-0x962230ff]
[    0.702204] pci 0000:00:17.0: reg 0x18: [io  0x3090-0x3097]
[    0.702213] pci 0000:00:17.0: reg 0x1c: [io  0x3080-0x3083]
[    0.702222] pci 0000:00:17.0: reg 0x20: [io  0x3060-0x307f]
[    0.702230] pci 0000:00:17.0: reg 0x24: [mem 0x96222000-0x962227ff]
[    0.814539] NET: Registered protocol family 2
[    0.814767] tcp_listen_portaddr_hash hash table entries: 16384 (order: 6, 262144 bytes, linear)
[    0.815038] TCP established hash table entries: 262144 (order: 9, 2097152 bytes, linear)
[    0.815477] TCP bind hash table entries: 65536 (order: 8, 1048576 bytes, linear)
[    0.815587] TCP: Hash tables configured (established 262144 bind 65536)
[    0.815676] UDP hash table entries: 16384 (order: 7, 524288 bytes, linear)
[    0.815819] UDP-Lite hash table entries: 16384 (order: 7, 524288 bytes, linear)
[    0.815965] NET: Registered protocol family 1
[    0.815971] NET: Registered protocol family 44
[    0.815984] pci 0000:00:02.0: Video device with shadowed ROM at [mem 0x000c0000-0x000dffff]
[    0.816626] pci 0000:01:00.0: CLS mismatch (64 != 128), using 64 bytes
[    0.816828] pci 0000:01:00.0: enabling device (0002 -> 0003)
[    0.817224] Trying to unpack rootfs image as initramfs...
[    0.986156] Initramfs unpacking failed: Decoding failed
[    0.993397] Freeing initrd memory: 50856K
[    0.993425] DMAR: Intel-IOMMU force enabled due to platform opt in
[    0.993459] DMAR: No ATSR found
[    0.993546] DMAR: dmar0: Using Queued invalidation
[    1.103257] usb usb1: New USB device found, idVendor=1d6b, idProduct=0002, bcdDevice= 5.03
[    1.103259] usb usb1: New USB device strings: Mfr=3, Product=2, SerialNumber=1
[    1.103261] usb usb1: Product: xHCI Host Controller
[    1.103263] usb usb1: Manufacturer: Linux 5.3.0-64-generic xhci-hcd
[    1.103264] usb usb1: SerialNumber: 0000:00:14.0
[    1.103469] hub 1-0:1.0: USB hub found
[    1.103489] hub 1-0:1.0: 12 ports detected
[    1.106061] xhci_hcd 0000:00:14.0: xHCI Host Controller
[    1.106066] xhci_hcd 0000:00:14.0: new USB bus registered, assigned bus number 2
[    1.106071] xhci_hcd 0000:00:14.0: Host supports USB 3.1 Enhanced SuperSpeed
[    1.106121] usb usb2: New USB device found, idVendor=1d6b, idProduct=0003, bcdDevice= 5.03
[    1.106122] usb usb2: New USB device strings: Mfr=3, Product=2, SerialNumber=1
[    1.106124] usb usb2: Product: xHCI Host Controller
[    1.106126] usb usb2: Manufacturer: Linux 5.3.0-64-generic xhci-hcd
[    1.106127] usb usb2: SerialNumber: 0000:00:14.0
[    1.106307] hub 2-0:1.0: USB hub found
[    1.106321] hub 2-0:1.0: 6 ports detected
[    1.107605] usb: port power management may be unreliable
[    1.107867] xhci_hcd 0000:39:00.0: xHCI Host Controller
[    1.107874] xhci_hcd 0000:39:00.0: new USB bus registered, assigned bus number 3
[    1.109069] xhci_hcd 0000:39:00.0: hcc params 0x200077c1 hci version 0x110 quirks 0x0000000200009810
[    1.109318] usb usb3: New USB device found, idVendor=1d6b, idProduct=0002, bcdDevice= 5.03
[    1.109320] usb usb3: New USB device strings: Mfr=3, Product=2, SerialNumber=1
[    1.109322] usb usb3: Product: xHCI Host Controller
[    1.109323] usb usb3: Manufacturer: Linux 5.3.0-64-generic xhci-hcd
[    1.109325] usb usb3: SerialNumber: 0000:39:00.0
[    1.109508] hub 3-0:1.0: USB hub found
[    1.109520] hub 3-0:1.0: 2 ports detected
[    1.109708] xhci_hcd 0000:39:00.0: xHCI Host Controller
[    4.346737] idma64 idma64.1: Found Intel integrated DMA 64-bit
[    4.351728] mei_me 0000:00:16.0: hbm: dma setup response: failure = 3 REJECTED
[    4.352889] Bluetooth: Core ver 2.22
[    4.352898] NET: Registered protocol family 31
[    4.352898] Bluetooth: HCI device and connection manager initialized
[    4.352901] Bluetooth: HCI socket layer initialized
[    4.352902] Bluetooth: L2CAP socket layer initialized
[    4.352903] Bluetooth: SCO socket layer initialized
[    4.389562] 88x2bu: loading out-of-tree module taints kernel.
[    4.403328] cryptd: max_cpu_qlen set to 1000
[    4.406575] Intel(R) Wireless WiFi driver for Linux
[    4.406576] Copyright(c) 2003- 2015 Intel Corporation
[    4.406641] iwlwifi 0000:00:14.3: enabling device (0000 -> 0002)
[    4.415431] iwlwifi 0000:00:14.3: TLV_FW_FSEQ_VERSION: FSEQ Version: 43.2.23.17
[    4.415434] iwlwifi 0000:00:14.3: Found debug destination: EXTERNAL_DRAM
[    4.415435] iwlwifi 0000:00:14.3: Found debug configuration: 0
[    4.415592] iwlwifi 0000:00:14.3: loaded firmware version 48.4fa0041f.0 op_mode iwlmvm
[    4.424971] AVX2 version of gcm_enc/dec engaged.
[    4.424972] AES CTR mode by8 optimization enabled
[    4.427194] usbcore: registered new interface driver btusb
[    4.428100] Bluetooth: hci0: Firmware revision 0.0 build 62 week 31 2019
[    8.658281] audit: type=1400 audit(1602118928.642:68): apparmor="STATUS" operation="profile_
docker.dockerd" name="docker-default" pid=1506 comm="apparmor_parser"
[    8.810186] Initializing XFRM netlink socket
[   10.867709] e1000e: eno1 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: Rx/Tx
[   10.867863] IPv6: ADDRCONF(NETDEV_CHANGE): eno1: link becomes ready
[   98.291764] rfkill: input handler disabled
[  142.368628] audit: type=1400 audit(1602119062.356:69): apparmor="DENIED" operation="open" p
ompose" name="/proc/2958/mounts" pid=2958 comm="python3" requested_mask="r" denied_mask="r" fs
```

See each stage on your own machine

`systemd-analyze`

Time spent in kernel and userspace, plus firmware and loader if the loader reports them

All stages

`efibootmgr -v`

Boot entries and boot order stored in firmware NVRAM

Firmware → loader

`sudo cat /proc/iomem`

The physical memory map from the memory slide (needs root to show addresses)

Firmware

`ls /sys/firmware/acpi/tables`

The ACPI tables the firmware built: DSDT, MADT, HPET, …

Firmware

`mokutil --sb-state`

Whether Secure Boot is on

Chain of trust

`journalctl -k -b`

This boot's kernel log, like the one on the previous slide

Kernel

Windows: msinfo32 shows BIOS mode and Secure Boot state; bcdedit /enum firmware lists boot entries.
Mac (Apple silicon): bputil -d shows the boot security policy.

Same relay, different runners

	Power-on	Firmware	Bootloader	Kernel	First process
Linux PC	CSME / PSP	UEFI (AMI, Insyde, Phoenix)	shim → GRUB	vmlinuz + initramfs	systemd
Windows PC	CSME / PSP	UEFI	bootmgfw.efi → winload.efi	ntoskrnl.exe	smss.exe
Mac (Apple silicon)	Boot ROM in the SoC	LLB	iBoot	XNU	launchd

Describing the hardware

PCs hand the kernel ACPI tables. Macs pass a device tree from iBoot.

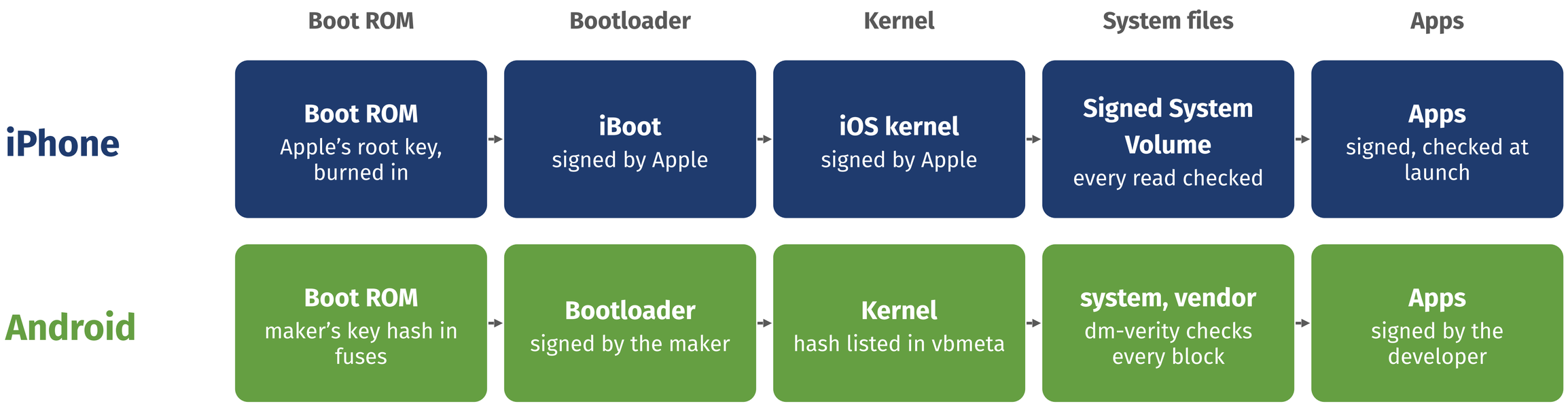
Root of trust

No BIOS or UEFI on Apple silicon: the Boot ROM verifies LLB, which verifies iBoot.

ssh <github-username>@34.45.78.138

Phones use the same chain of trust

Each stage checks the next stage's signature before running it, just like on the PC



Plus a separate security chip

iPhone: the Secure Enclave boots its own signed software and guards your keys and Face ID data.

...on Android too

A security chip such as Google's Titan M2, or a secure area of the main chip (TEE), boots separately and guards keys.

iOS: Apple signs every step

One key, held by Apple, and every installation is approved for one specific phone

Boot ROM
in the chip; can never change

↓

iBoot
checked against Apple’s root key

↓

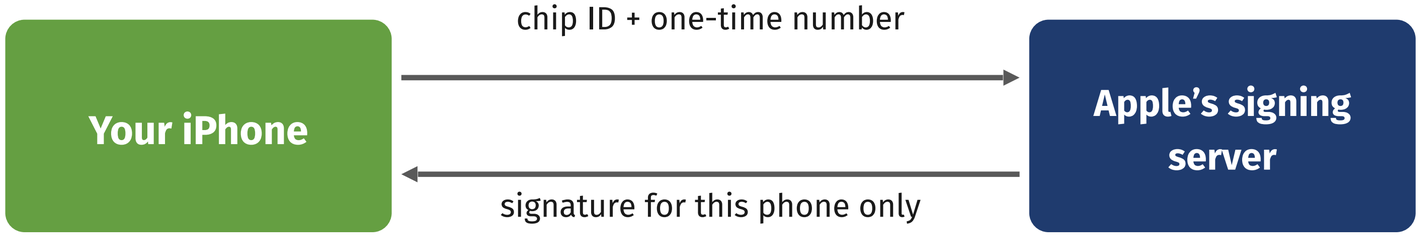
iOS kernel
checked by iBoot

↓

Signed System Volume
kernel checks every system file it reads

A failed check stops the boot. The phone must be restored from a computer.

Updates are personalized



- Apple stops signing an old version soon after a new one ships, so you can’t go back to a version with known bugs
- A signed update copied from another phone won’t verify on yours
- The Secure Enclave verifies its own software separately, so even a hacked kernel can’t read its keys

Android Verified Boot

Same idea, but the boot state is visible, and it isn't always green

Boot ROM
checks the bootloader against the maker's key hash in fuses

↓

Bootloader
checks vbmeta's signature, and refuses older versions

↓

vbmeta
a signed list of hashes for boot, system and vendor

↓

Kernel + dm-verity
checks each block of system files as it's read

Every boot ends in one of four states

GREEN	Locked, with the phone maker's key. Boots normally.
YELLOW	Locked, with the owner's own key. Warning on every boot.
ORANGE	Unlocked. Nothing is verified. Warning on every boot.
RED	Verification failed. Won't boot, or warns the system is corrupt.

Apps can check the state: the security chip signs the boot state into key attestation, which banking apps and Google's Play Integrity read.

Android: the owner can unlock it

1. **Turn on OEM unlocking**
   In Developer options. Some phones hide or grey out this switch.

2. **Reboot into the bootloader**
   and run fastboot flashing unlock from a computer

3. **Confirm on the phone itself**
   with the volume and power keys, so it can’t be done remotely

4. **All your data is wiped**
   so a thief can’t unlock the phone to read it

5. **Every boot shows an ORANGE warning**
   Now you can install another OS, or root the phone

```
$ adb reboot bootloader
$ fastboot flashing unlock
  …confirm on the phone…

# Optional, on a Pixel: relock with your own key
$ fastboot flash avb_custom_key avb_pkmd.bin
$ fastboot flashing lock
```

Relocking with your own key gives YELLOW: full verified boot, with your key instead of Google’s.

What it costs

- Banking and payment apps may refuse to run on an unlocked phone.
- Not every phone allows it: Samsung removed the option worldwide in One UI 8 (2025), and many carrier phones block it.

iOS has no unlock switch

Why not

- Apple holds the only signing key. There is no slot for an owner’s key.
- Every installation is approved by Apple’s server, for one phone
- Apple argues an unlock option would be used by thieves, abusers and malware, not just owners
- The trade-off: you can’t run another OS, even on a phone you own

The cracks that have existed

A Boot ROM bug: checkm8 (2019)

Works on iPhone 4S through iPhone X. It’s in ROM, so it can never be patched, but it needs a USB cable and the phone in hand.

Another OS: a stunt, not a product

Project Sandcastle (2020) used checkm8 to boot Android on an iPhone 7, re-run from a computer on every boot.

Approved exceptions

Apple lends special research iPhones to vetted researchers. In the EU, other app stores are allowed, but iOS itself stays Apple’s.

Can you jailbreak iOS today?

A jailbreak is a chain of exploits. It runs inside iOS; it doesn't replace it.

Run code via a browser or app bug → **Escape the app sandbox** → **Exploit a kernel bug** → **Defeat hardware checks (PAC, PPL)** → **Root, inside Apple's iOS**

Hobbyist jailbreaks

None public for any iOS after 17.0 (September 2023) on iPhone XS or newer. iOS 27.2 shipped on October 26, 2026.

Attackers still have them

Google found the Coruna kit (2026): 23 exploits in five chains for iOS 13 to 17.2.1, passed from a spyware vendor to spies to criminals. Up-to-date phones were safe.

Replacing the OS

Only on checkm8 phones (iPhone X and older), and only while tethered to a computer. On any newer iPhone: no.

Why so hard? Each link in the chain is a rare, expensive bug, Apple patches quickly, and hardware defenses break whole exploit techniques.

Latest security, side by side

As of October 2026

	iOS 27	Android 17
Released	September 2026; 27.2 (October 26) fixed about 160 security bugs	June 2026 on Pixels; other brands follow later
Memory-safety hardware	Memory Integrity Enforcement: always-on memory tagging on iPhone 17 (A19)	Memory tagging on Pixel 8 and later, mainly through Advanced Protection
High-risk mode	Lockdown Mode	Advanced Protection, now including scam detection
Theft and scams	Stolen Device Protection; new Impersonation Risk Detection	Theft protection on by default; Mark as lost; scam detection
Installing apps	App Store; other stores only in the EU	Sideloading allowed, but developers must be verified: 4 countries from September 30, 2026, worldwide in 2027
Your own OS	Not possible on current iPhones	Possible on Pixels; blocked on Samsung and many carrier phones

Who holds the keys?

iPhone

Apple holds the only key

Strong security by default, and no override for the owner

Android

The maker holds the key; some let owners add theirs

Freedom with a visible cost: wiped data, warning screens, app checks

The two are converging. Samsung removed unlocking and Google now verifies developers, while Apple had to allow other app stores in the EU.

Discussion: should the owner of a phone be able to replace its operating system?

What happens after boot?

The kernel starts exactly one program. Every other process descends from it.

```
Ubuntu 24.04 LTS nuc tty1

nuc login: alice
Password:
alice@nuc:~$ pstree -p
systemd(1)─┬─systemd-journal(298)
           ├─systemd-udevd(341)
           ├─NetworkManager(702)
           ├─sshd(905)
           └─login(788)───bash(1402)───pstree(1530)
```

Every command you run is a child of your shell, which starts from PID 1.

1. **Kernel finishes booting**
   Memory protection, user mode and the timer are now switched on

2. **Kernel starts PID 1: systemd**
   The first user-space program; it starts all the others

3. **systemd starts agetty, sushi**
   agetty prints the “login:” prompt on the console
   sshd allows remote login

4. **login checks your password**
   Then it switches to your user ID and starts your shell

What is the OS for?



Referee

Shares the CPU, memory and devices between programs, and stops them from harming each other

One program can't read another's passwords



Illusionist

Makes each program feel like it has the whole machine: its own CPU, its own memory

300 programs share 12 cores



Glue

Provides common services, so every program doesn't reinvent them

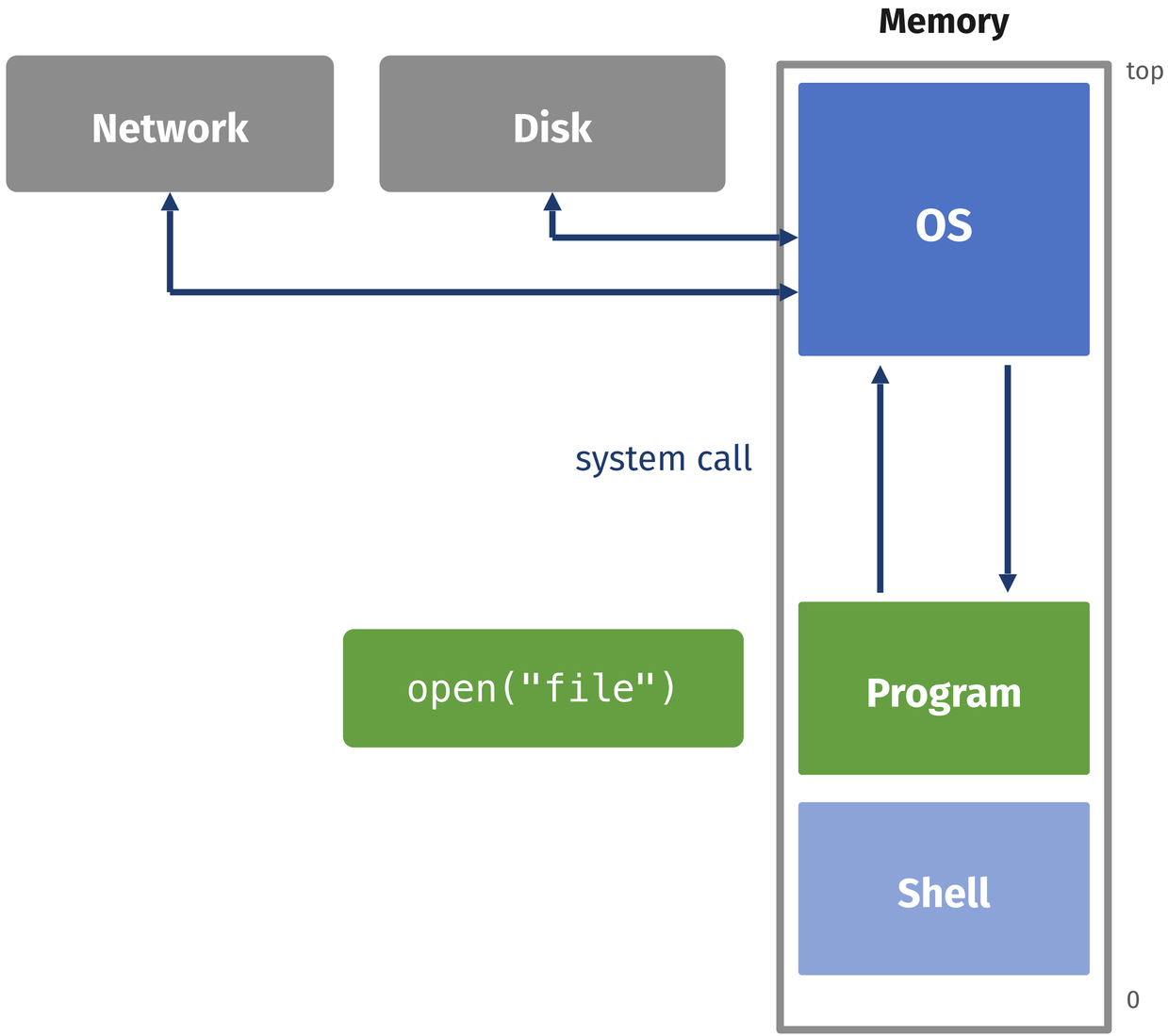
Files, networking, windows

The shell is not the OS. It's an ordinary program that uses these same services.

Framing from Anderson & Dahlin, Operating Systems: Principles and Practice.

The system model

- The OS reserves a region of memory for itself
- The shell is just another program
- Programs ask the OS for services through system calls, like open("file")
- On DOS this was optional: programs often skipped the OS and drove the hardware directly



What if nothing enforces the rules?

That’s how the CPU wakes up: real mode, no protection. DOS ran like this.

Three ways one program can hurt all the others:



Memory

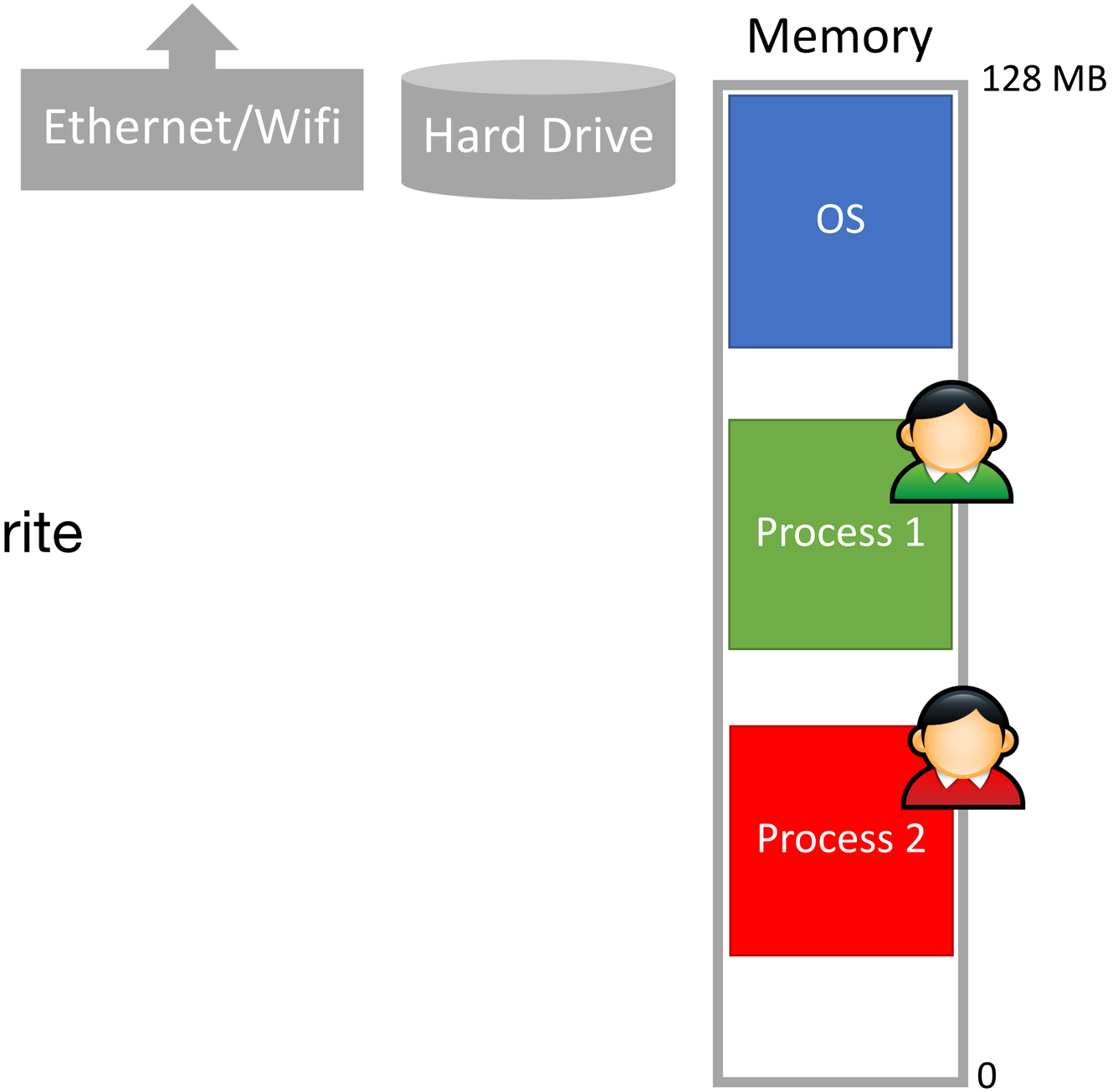


Devices



CPU time

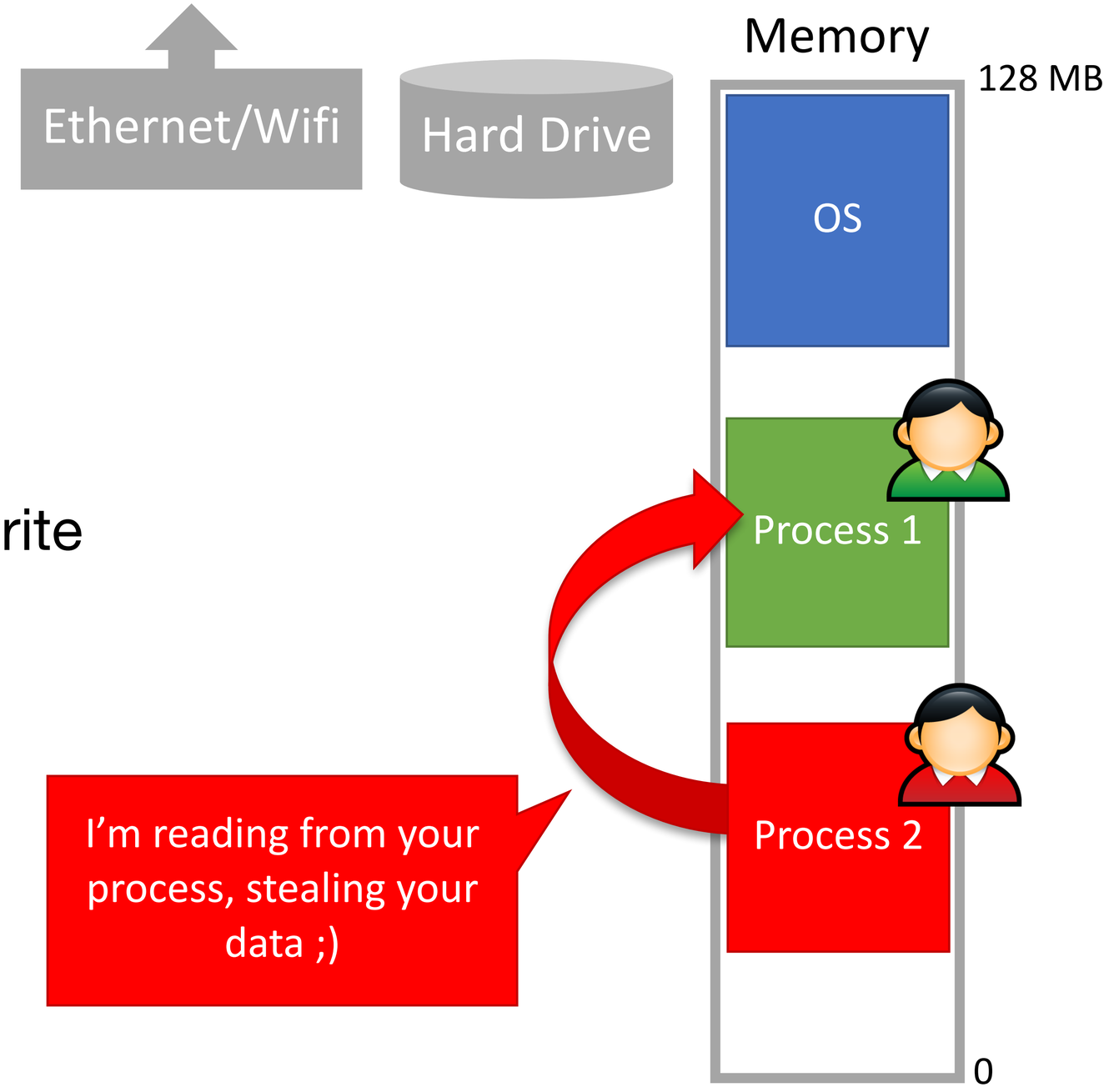
Memory Unsafety



Problem: any process can read/write any memory

Memory Unsafety

Problem: any process can read/write any memory



Problem 1: anyone can touch any memory

Malice

A malicious program can steal other programs' secrets, or plant its own code inside the OS.

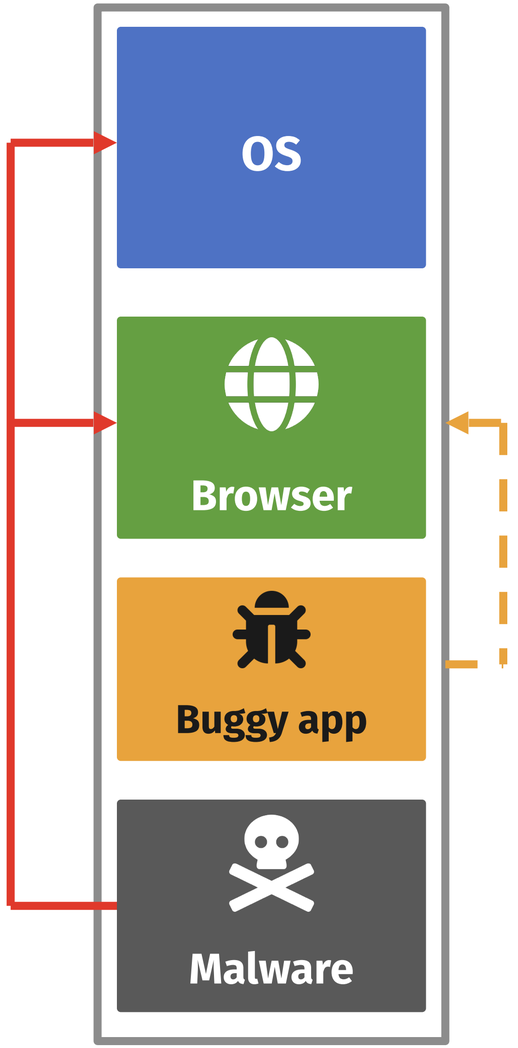
Bugs

A buggy program can corrupt another program or crash the whole machine. The crash shows up far from the cause, which makes it very hard to debug.

Overwrite the OS with malicious code

Scan the browser for passwords and card numbers

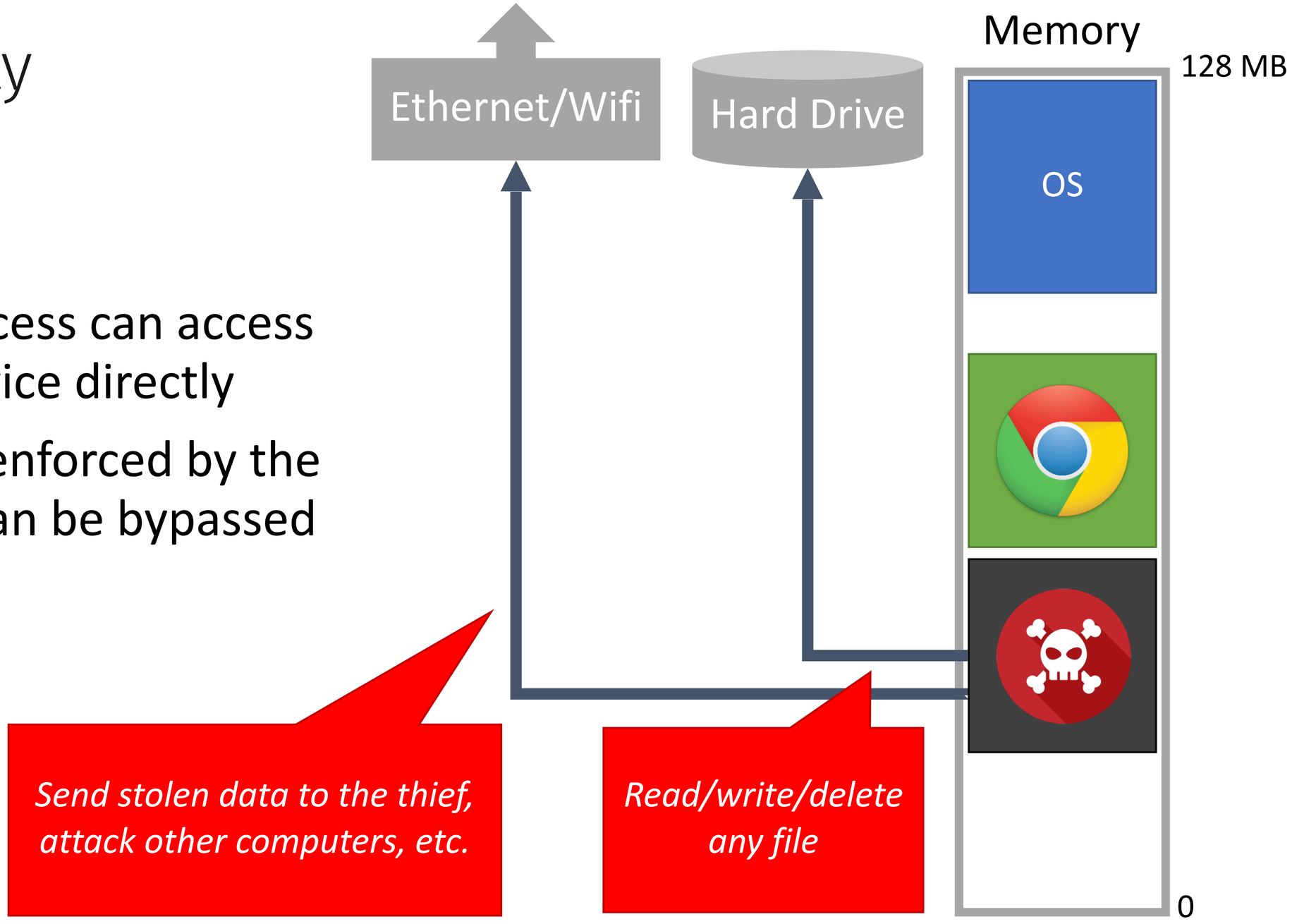
Oops: a stray pointer overwrites the browser's data



Device Unsafety

Problem: any process can access any hardware device directly

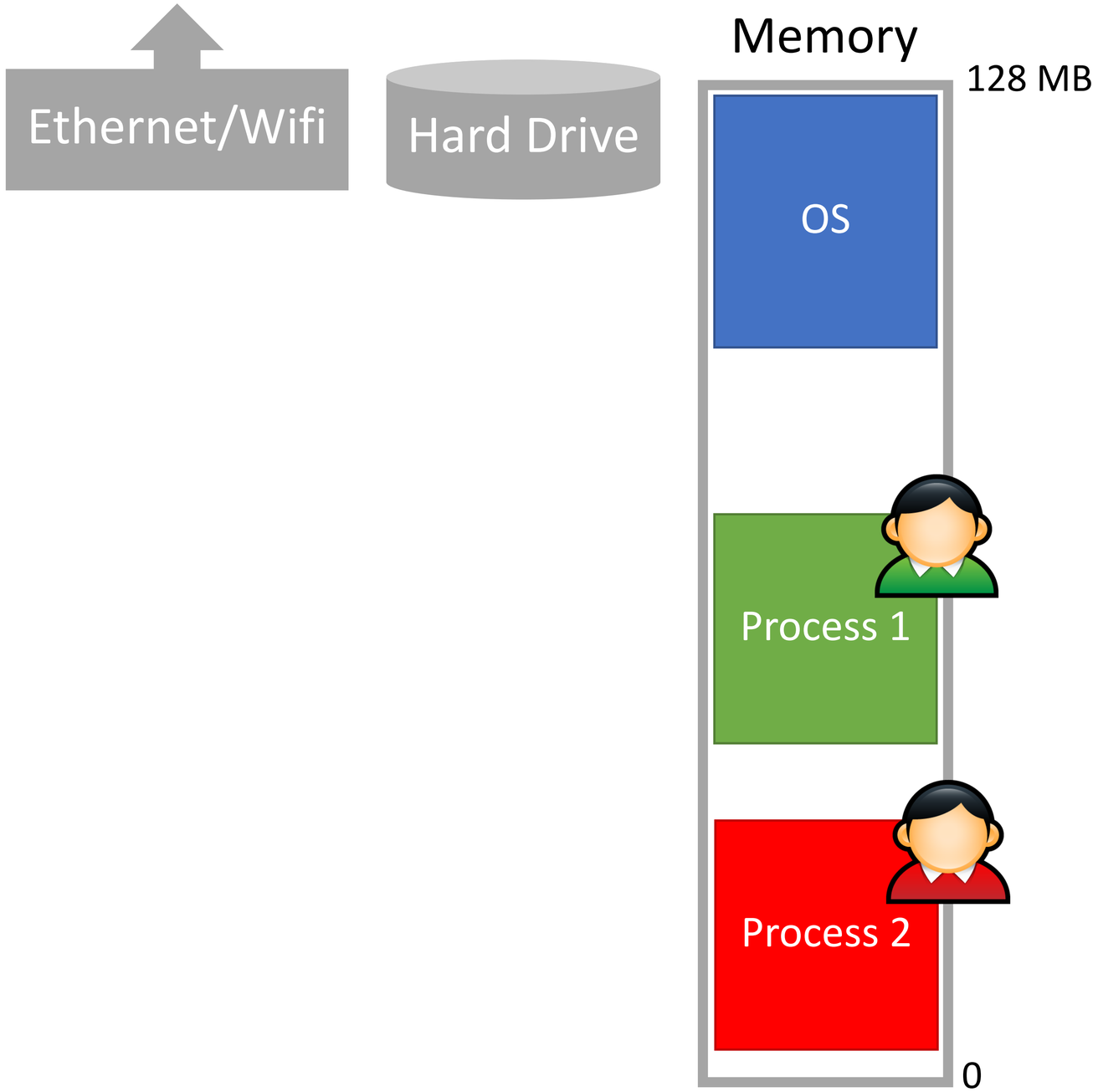
Access control is enforced by the OS, but OS APIs can be bypassed



Device Unsafety

Problem: any process can access any hardware device directly

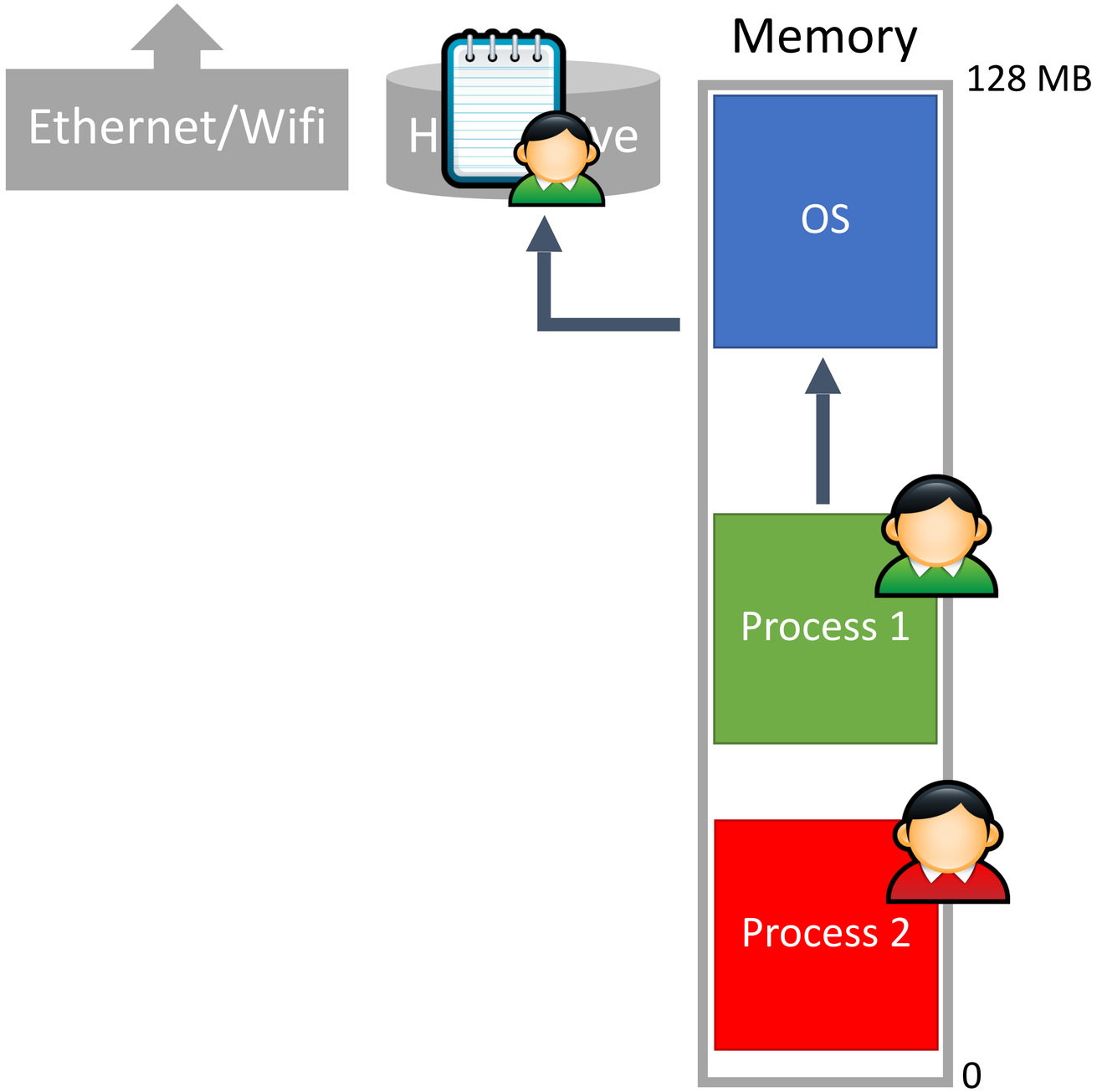
Access control is enforced by the OS, but OS APIs can be bypassed



Device Unsafety

Problem: any process can access any hardware device directly

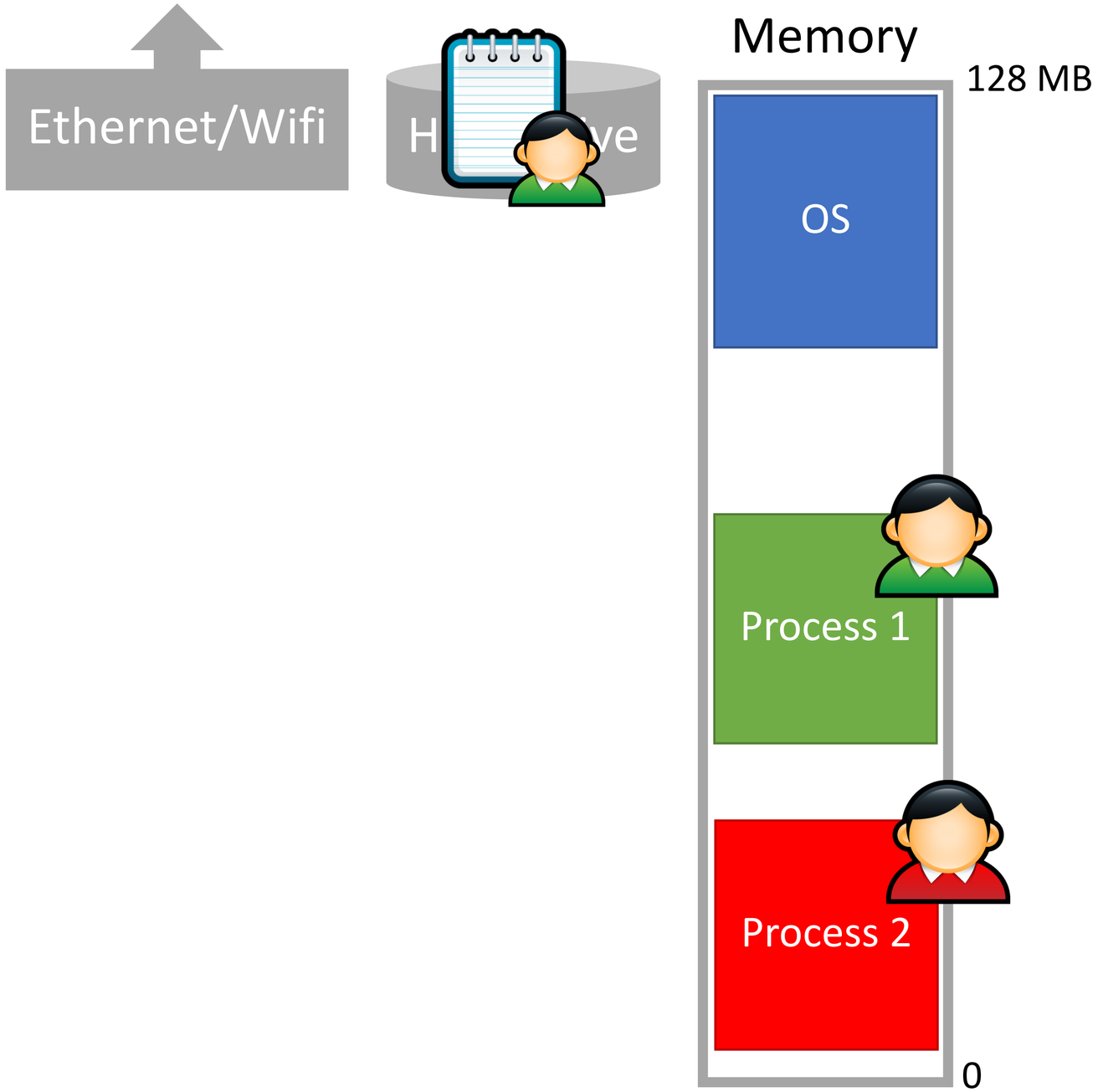
Access control is enforced by the OS, but OS APIs can be bypassed



Device Unsafety

Problem: any process can access any hardware device directly

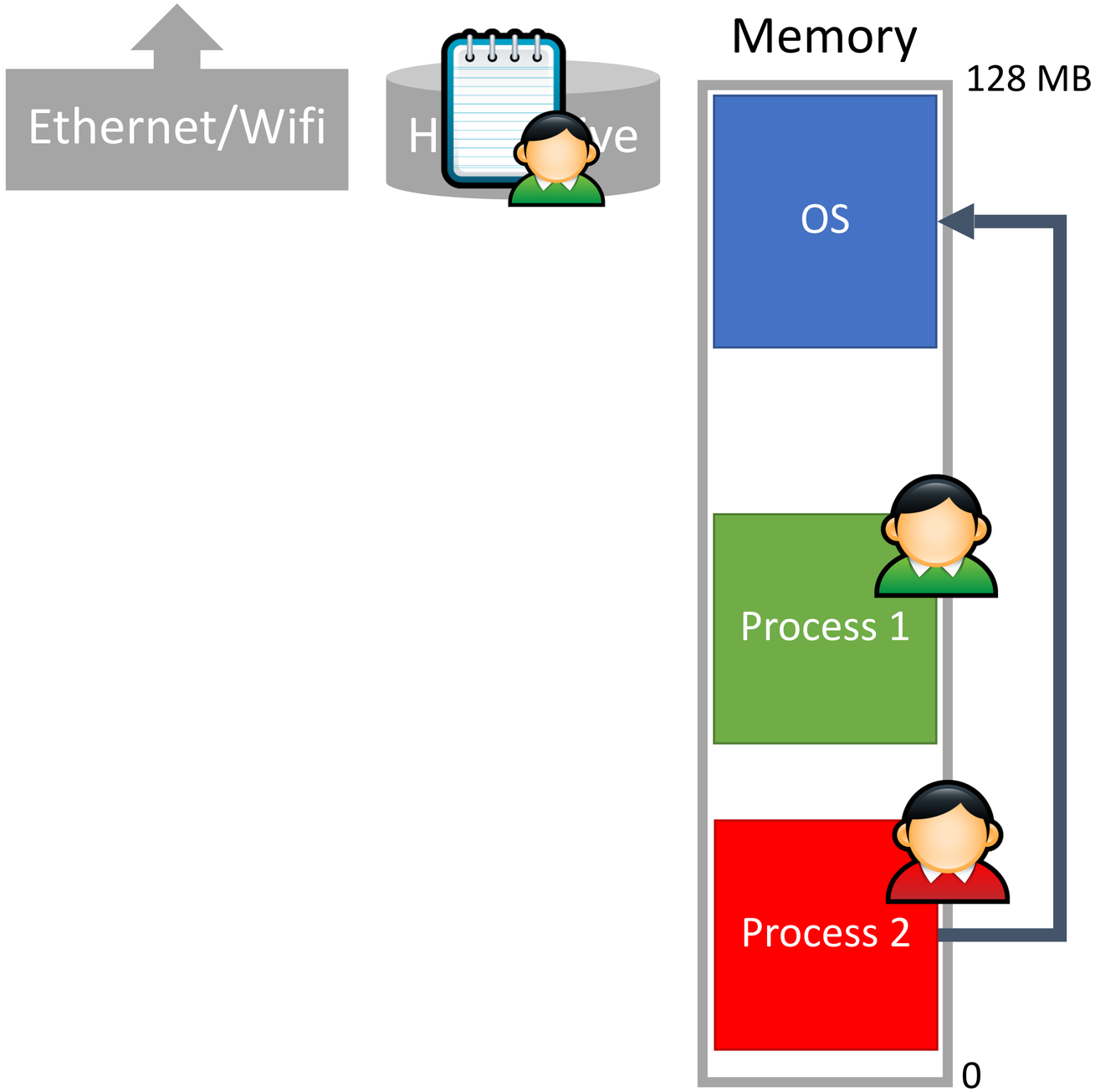
Access control is enforced by the OS, but OS APIs can be bypassed



Device Unsafety

Problem: any process can access any hardware device directly

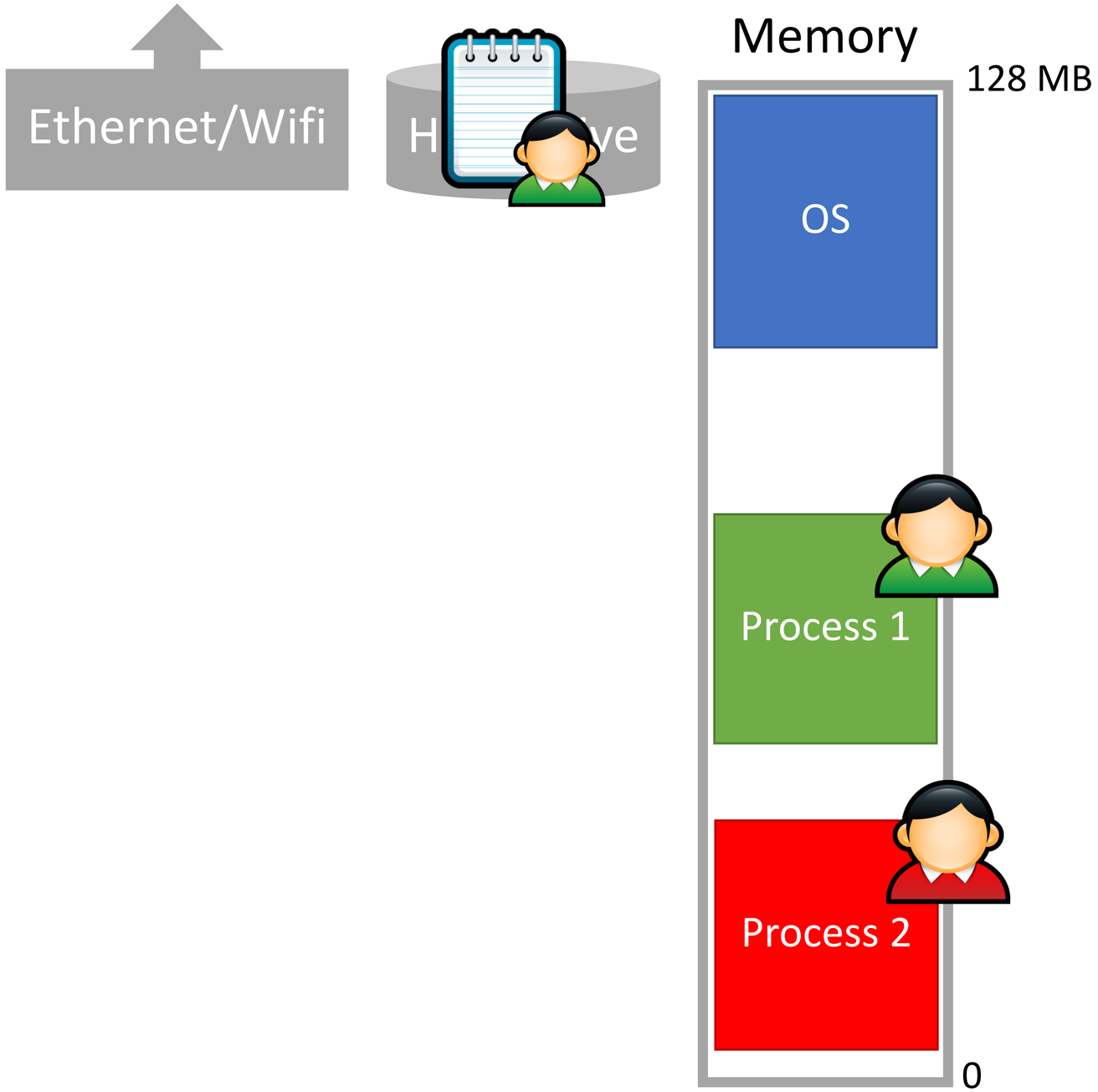
Access control is enforced by the OS, but OS APIs can be bypassed



Device Unsafety

Problem: any process can access any hardware device directly

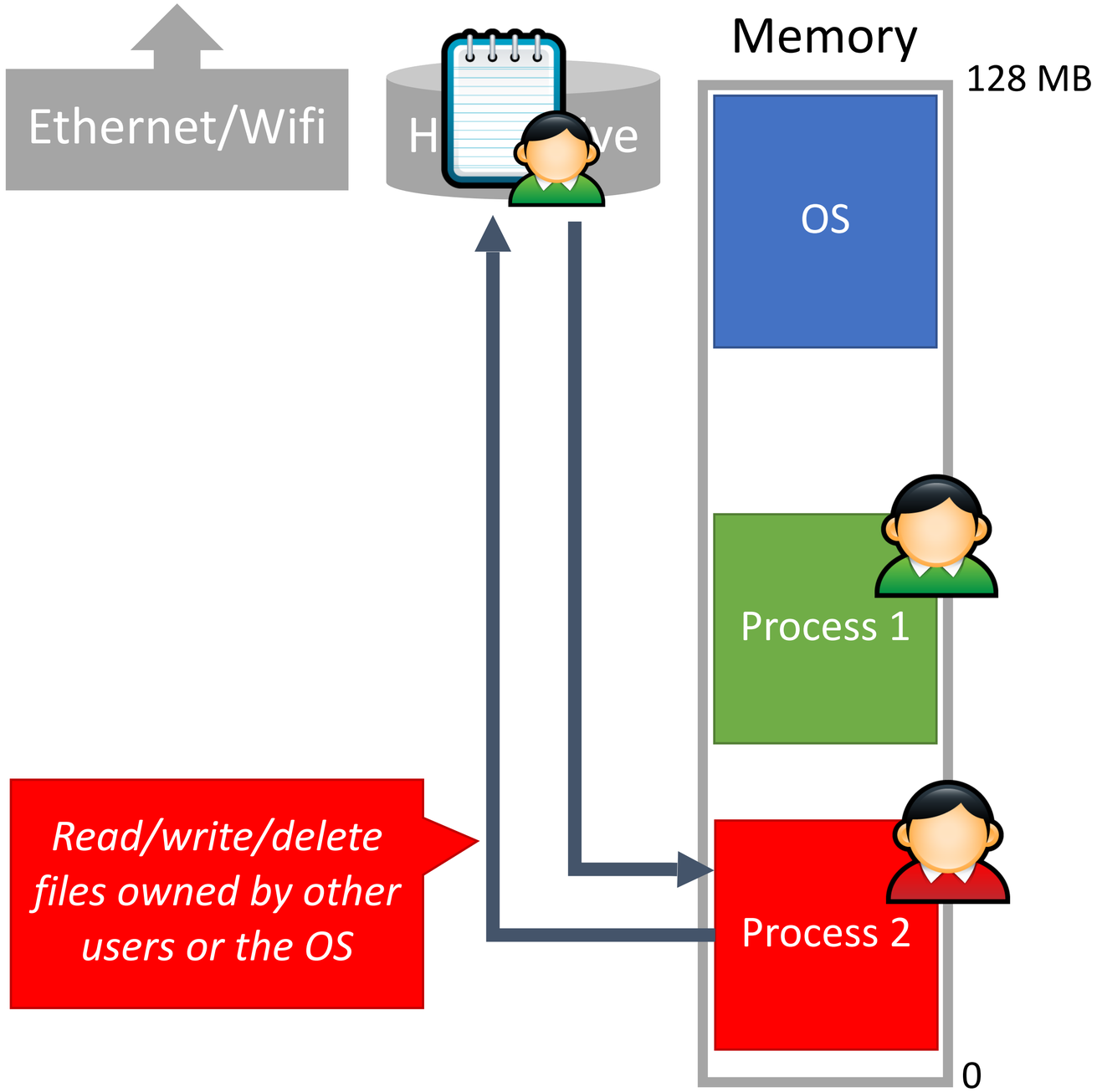
Access control is enforced by the OS, but OS APIs can be bypassed



Device Unsafety

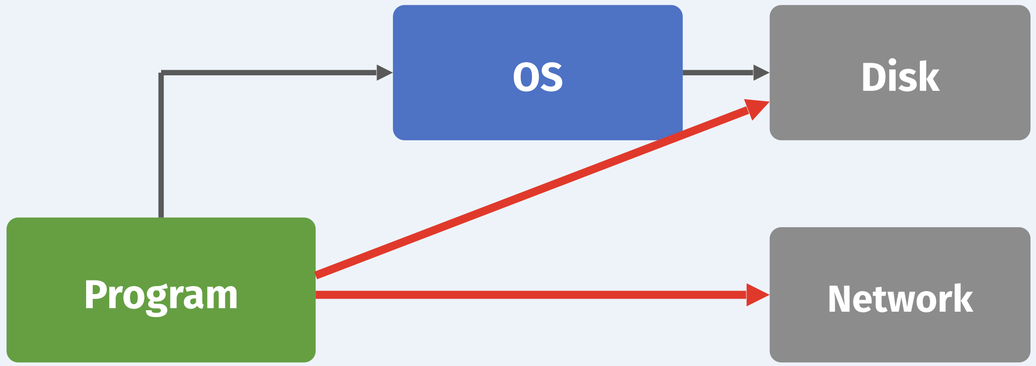
Problem: any process can access any hardware device directly

Access control is enforced by the OS, but OS APIs can be bypassed



Two more ways one program can hurt the rest

Problem 2: direct device access

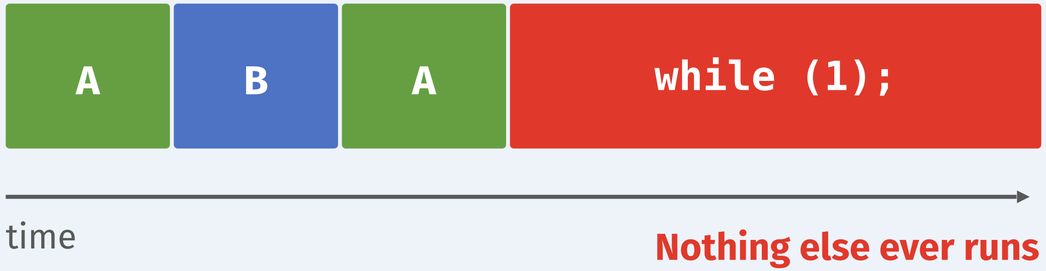


Direct, skipping the OS

- Read raw disk sectors: everyone's files, ignoring permissions
- Send any packet on the network
- Tell a device to copy memory for it (DMA)

Problem 3: hogging the CPU

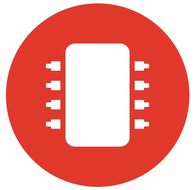
Who is running on the CPU



Nothing else ever runs

With cooperative multitasking, as in Windows 3.1 and classic Mac OS, programs had to hand the CPU back voluntarily. One infinite loop froze the whole machine.

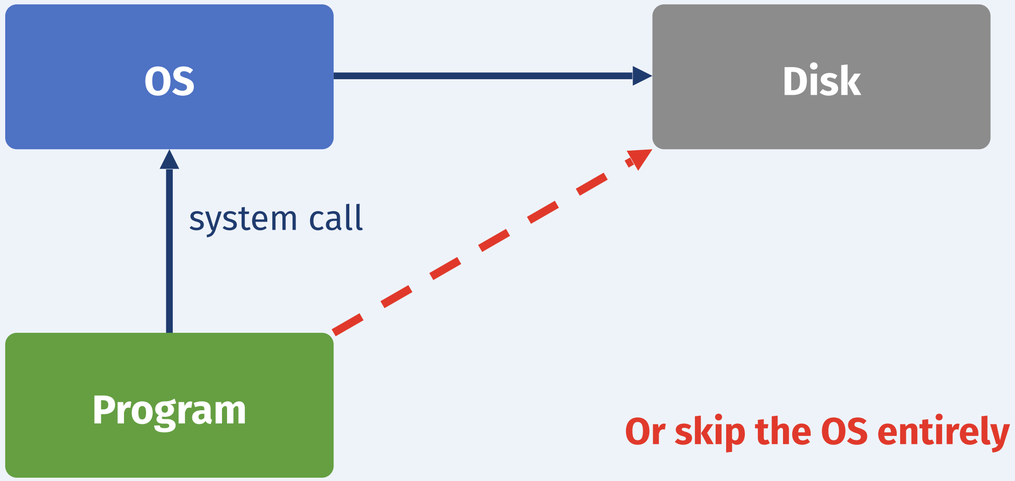
Three problems, three hardware fixes

Problem	Hardware feature	What the OS can now enforce
 **Any memory**	**MMU** address translation	Each program sees only its own memory. The OS's memory is off-limits.
 **Any device**	**User and kernel mode** privileged instructions	Only the kernel can touch devices. Programs must ask through system calls.
 **Keeps the CPU**	**Timer interrupt** fires every few ms	The kernel regains control and can switch to another program.

The kernel switches on all three during boot, before it starts PID 1.

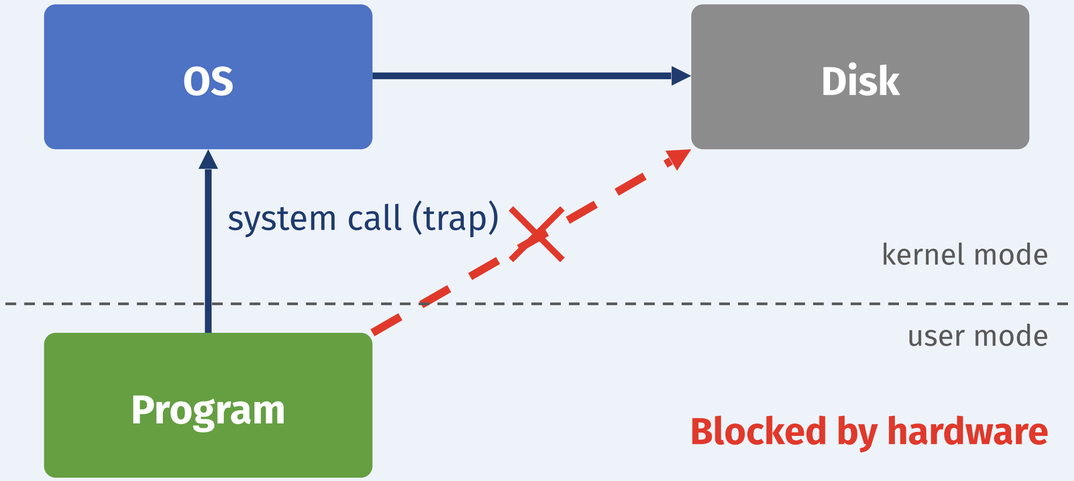
The API becomes the only door

Without protection (DOS)



System calls are a convenience

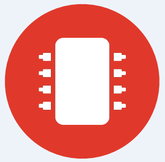
With protection



System calls are the only way in

That's why open(), read() and write() are system calls. Each one crosses from user mode into the kernel, which checks your permissions before it touches the disk.

Isolation still matters today



Meltdown (2018)

A CPU flaw let ordinary programs read kernel memory through speculative execution. The fix, KPTI, stopped mapping most of the kernel into each program's page tables.



Devices that copy memory

A Thunderbolt device can read memory directly using DMA, as the Thunderspy attacks (2020) showed. The IOMMU blocks this: the DMAR lines in our boot log describe it.



Your browser

Chrome runs each website in its own process (Site Isolation, 2018), so the OS keeps a malicious page away from your bank's tab.



Tiny computers

Many microcontrollers have no MMU. Every task shares one memory, just like DOS.

Review

Old systems did not protect memory or devices

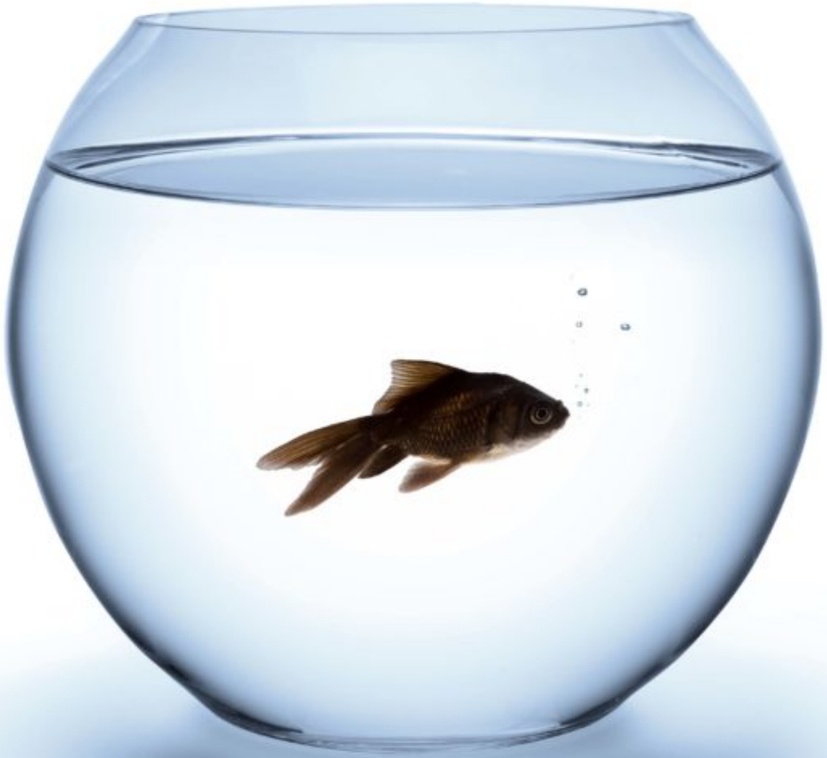
- Any process could access any memory
- Any process could access any device

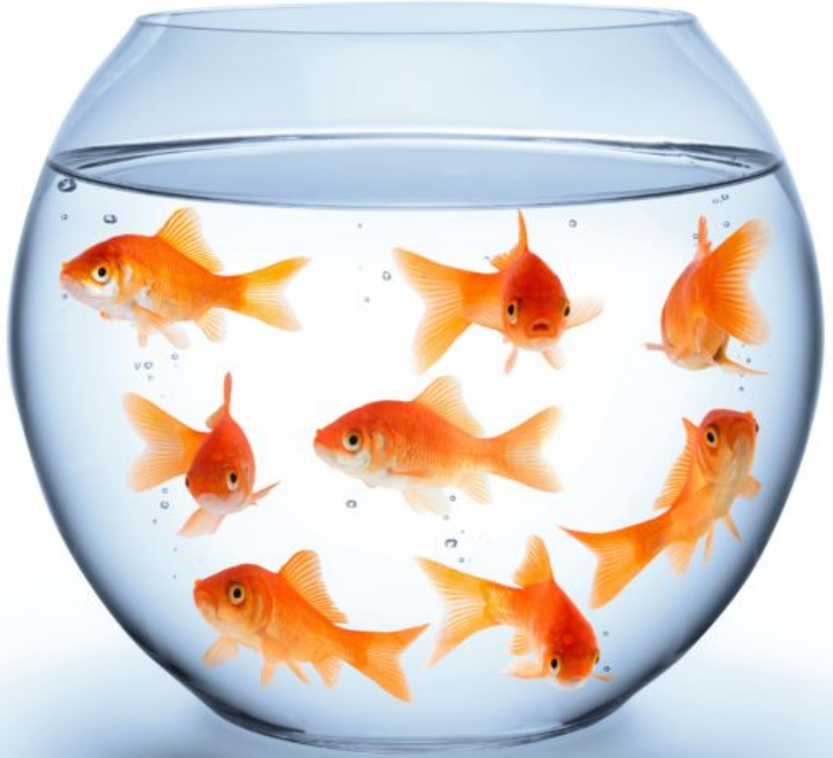
Problems

- No way to enforce access controls on users or devices
- Processes can steal from or destroy each other
- Processes can modify or destroy the OS

On old computers, systems security was literally impossible

ISOLATION





Threat Model
Principles
Intro to System Architecture
Hardware Support for Isolation
Examples

Towards Modern Architecture

To achieve systems security, we need process isolation

- Processes cannot read/write memory arbitrarily
- Processes cannot access devices directly

How do we achieve this?

Hardware support for isolation

1. Protected mode execution (a.k.a. process rings)
2. Virtual memory



Privilege rings on x86



- x86 has four rings, 0 to 3. A lower number means more privilege.
- Ring 0: the kernel, including every device driver, on Linux, Windows and macOS
- Ring 3: every program, even ones running as root
- Rings 1 and 2 are essentially unused: 64-bit page tables only know “user” (ring 3) and “supervisor” (rings 0–2)

Same idea on other CPUs

Arm: EL0 apps, EL1 kernel, EL2 hypervisor, EL3 secure firmware

RISC-V: U user, S kernel, M firmware

Why drivers in ring 0 matter

In July 2024, one faulty CrowdStrike kernel driver crashed about 8.5 million Windows PCs at once.

Below ring 0

Nicknames, not real rings: code even more privileged than the kernel, and invisible to it

Ring 3	Programs
Ring 0	Kernel
“Ring –1”	Hypervisor: runs whole operating systems as guests (VT-x, AMD-V; KVM, Hyper-V)
“Ring –2”	System Management Mode: firmware code the CPU switches to on special interrupts. The OS can’t see it.

not part of the CPU at all

“Ring –3”	A separate processor: Intel CSME or AMD PSP. Runs whenever the board has power.

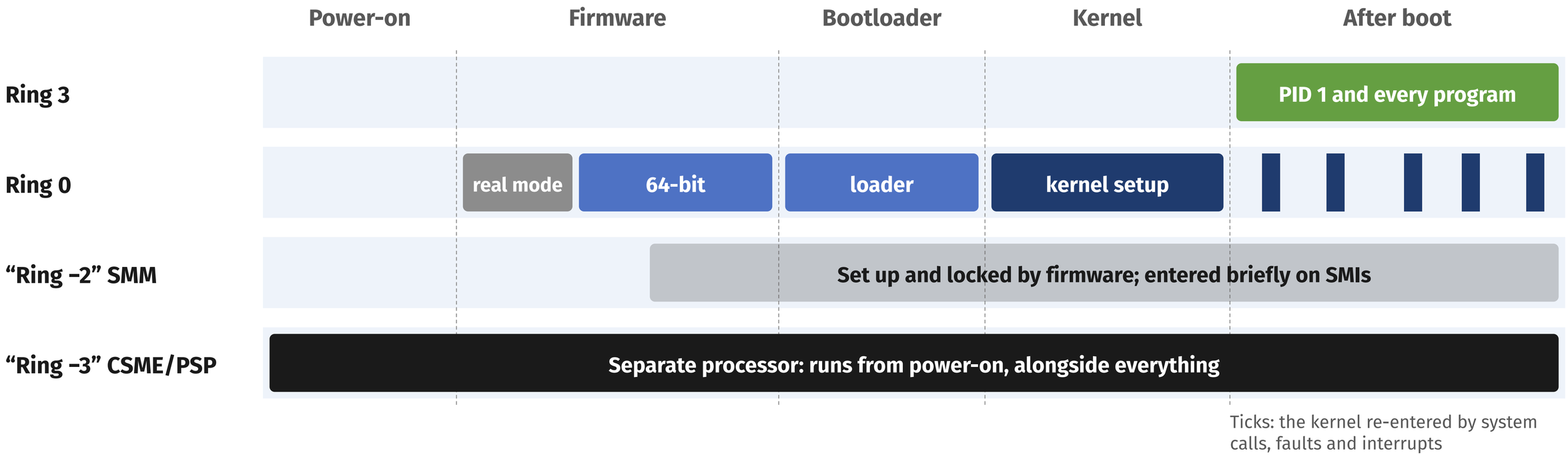
Case study: Intel’s management engine

From 2015 (ME 11) it ran its own operating system, MINIX 3, with a network stack and a web server for remote management.

In 2017, Google engineers warned that this layer sat below any OS they controlled. The same year, a flaw in that web server’s login (CVE-2017-5689) let attackers take over managed PCs over the network.

Isolation is only as strong as the most privileged code.

Privilege levels during boot



No mode switch for the OS

UEFI firmware is already in 64-bit mode before any OS code runs.

Going down is deliberate

The kernel enters ring 3 for the first time when it starts PID 1.

Coming back up is controlled

Only through doors the kernel set up: system calls, exceptions and interrupts.

What ring 3 can’t do

Instruction	Controls	If a program could run it…
`mov to cr0`	Whether protection and paging are on	Switch protection off
`mov to cr3`	Which page tables are active	Map any memory, including the kernel’s
`lidt, wrmsr`	Interrupt handlers and the system call entry point	Point the kernel’s doors at its own code
`cli, sti`	Whether interrupts are delivered	Block the timer and keep the CPU forever
`hlt`	Stops the CPU until the next interrupt	Freeze a core
`in, out`	I/O ports, used by older devices	Talk to devices directly

Most devices don’t use in and out anymore

Their registers sit in the PCIe window on the memory map. The kernel protects them with page tables: they simply aren’t mapped into a program’s memory.

If a program tries any of these, the CPU raises a fault and the kernel takes over.

Try it: run a privileged instruction

```
$ cat hlt.c
int main(void) {
    __asm__("hlt");   // ring 0 only
    return 0;
}
$ gcc hlt.c -o hlt && ./hlt
Segmentation fault (core dumped)
$ sudo dmesg | tail -1
traps: hlt[4021] general protection
  fault ip:55d4c1a0d131 sp:7ffc3a1e9f60
  error:0 in hlt[…]
```

1. **The CPU sees hlt in ring 3**
   and refuses to run it
2. **It raises a protection fault**
   and jumps into the kernel's fault handler, in ring 0
3. **The kernel decides what to do**
   Linux sends the program the signal SIGSEGV
4. **The program dies**
   and the shell prints "Segmentation fault"

Green: ring 3. Blue: ring 0.

A program never just crashes: the CPU hands control to the kernel, and the kernel decides.

Three ways into the kernel

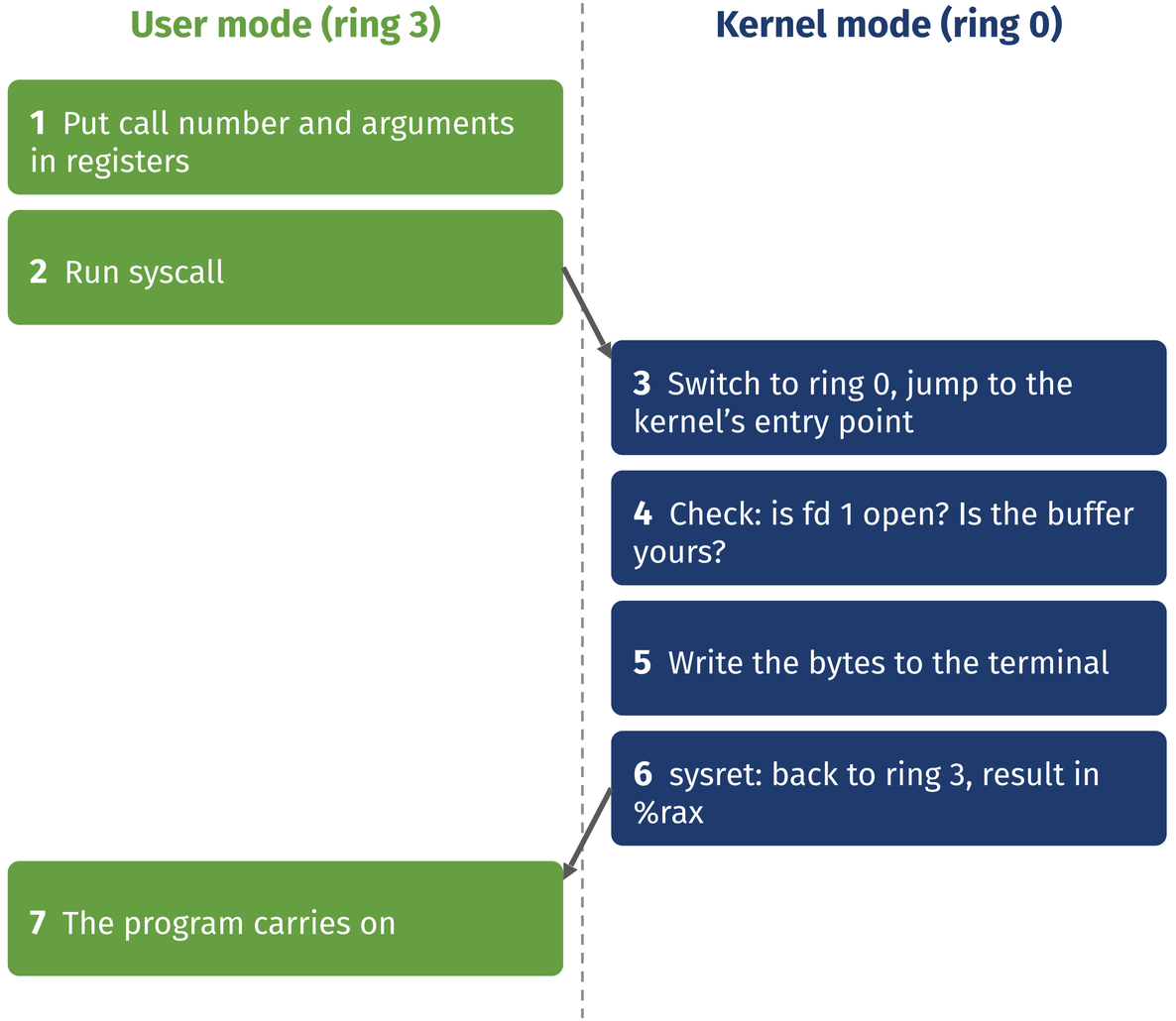
User mode (ring 3)

System call	Exception	Interrupt
The program asks for something: open a file, send a packet	The program did something the CPU won't allow or can't finish: hlt, divide by zero, unmapped memory	The outside world needs attention: the timer, a key press, a network packet
deliberate ↓	*accidental* ↓	*from hardware* ↓
Kernel mode (ring 0)		
System call entry point, set at boot	Fault handler, listed in the interrupt table	Device handler, listed in the same table

In all three, the CPU jumps to an address the kernel registered at boot, so a program can't choose where it lands in the kernel. When the kernel is done, it returns to ring 3.

A system call, step by step

```
# write(1, "hi\n", 3) on x86-64 Linux
mov  $1, %eax          # call number: write
mov  $1, %edi          # fd 1: standard output
lea  msg(%rip), %rsi   # address of "hi\n"
mov  $3, %edx          # length in bytes
syscall                # enter the kernel
# %rax now holds the result: 3
```



Older and other ways in

int 0x80 and sysenter on 32-bit x86, svc on Arm, ecall on RISC-V. Same idea everywhere: one special instruction, one fixed entry point.

See it yourself: strace

```
$ strace -e trace=openat,read,write cat /etc/hostname > /dev/null
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0"..., 832) = 832
…
openat(AT_FDCWD, "/etc/hostname", O_RDONLY) = 3
read(3, "nuc\n", 131072)                = 4
write(1, "nuc\n", 4)                    = 4
read(3, "", 131072)                     = 0
+++ exited with 0 +++
```

One line, one round trip

Each line is a trip into ring 0 and back, with the result after the =

Most of it is setup

cat first loads its libraries. The real work is the last four calls.

Try: strace -c ls

Counts every system call a program makes

System calls are the only door, so strace sees nearly everything a program does to the outside world.

System Boot Sequence

1. On startup, the CPU starts in 16-bit real mode
   - Protected mode is disabled
   - Any process can access any device

System Boot Sequence

1. On startup, the CPU starts in 16-bit real mode
   - Protected mode is disabled
   - Any process can access any device
2. BIOS executes, finds and loads the OS

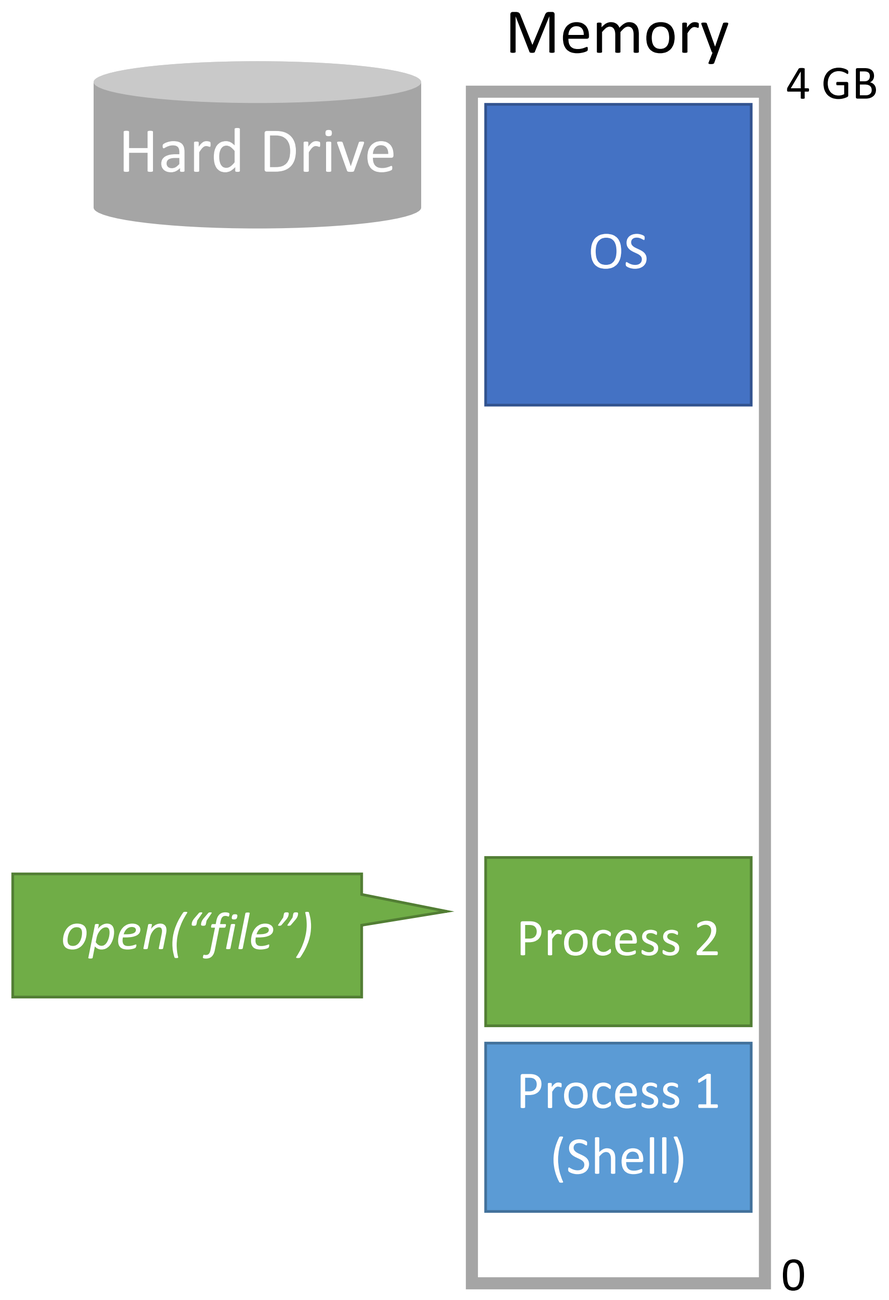
System Boot Sequence

1. On startup, the CPU starts in 16-bit real mode
   - Protected mode is disabled
   - Any process can access any device
2. BIOS executes, finds and loads the OS
3. OS switches CPU to 32-bit protected mode
   - OS code is now running in Ring 0
   - OS decides what Ring to place other processes in

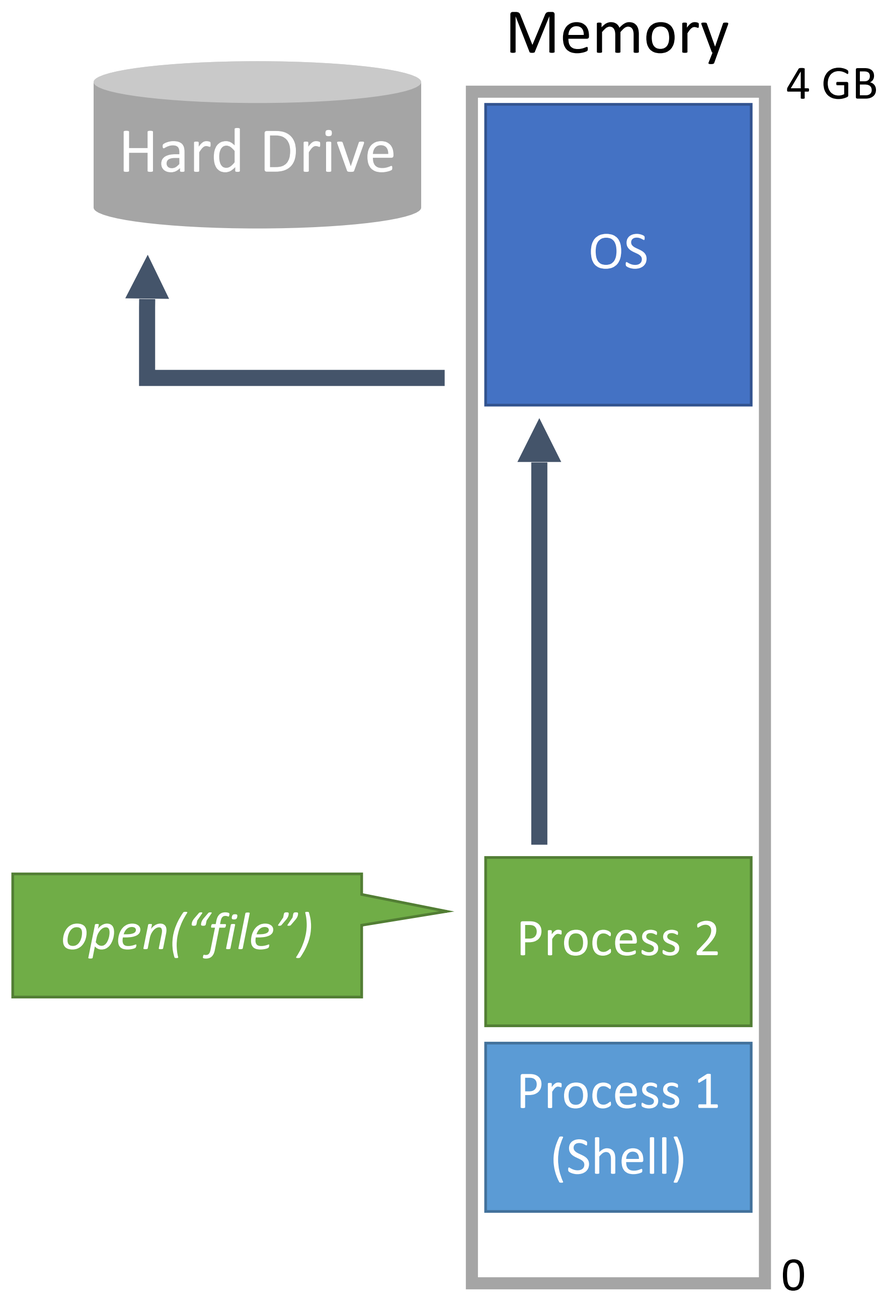
System Boot Sequence

1. On startup, the CPU starts in 16-bit real mode
   - Protected mode is disabled
   - Any process can access any device
2. BIOS executes, finds and loads the OS
3. OS switches CPU to 32-bit protected mode
   - OS code is now running in Ring 0
   - OS decides what Ring to place other processes in
4. Shell gets executed, user may run programs
   - User processes are placed in Ring 3

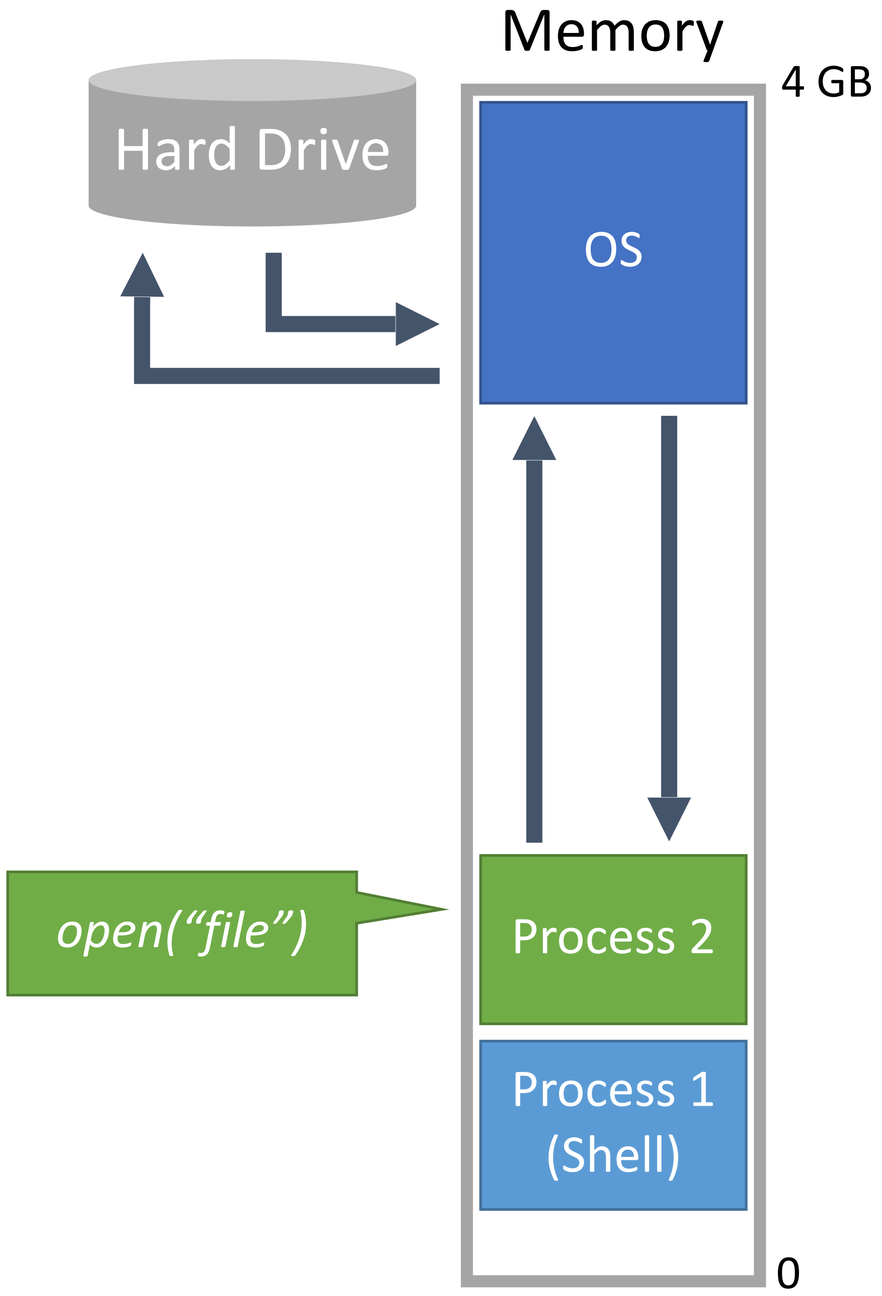
How to change modes



How to change modes



How to change modes



Changing Modes

Applications often need to access the OS APIs

- Writing files
- Displaying things on the screen
- Receiving data from the network
- etc...

But the OS is Ring 0, and processes are Ring 3

How do processes get access to the OS?

Changing Modes

Applications often need to access the OS APIs

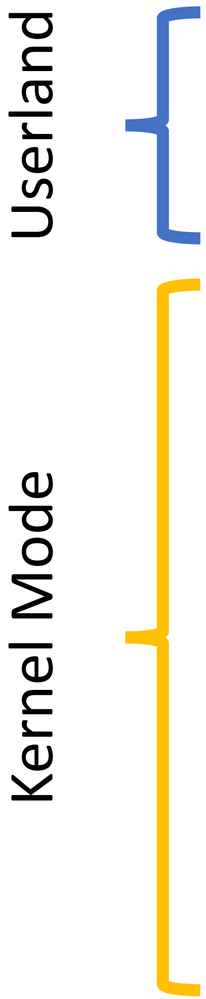
- Writing files
- Displaying things on the screen
- Receiving data from the network
- etc...

But the OS is Ring 0, and processes are Ring 3

How do processes get access to the OS?

- Invoke OS APIs with special assembly instructions
  - Interrupt: int 0x80
  - System call: sysenter or syscall
- int/sysenter/syscall cause a mode transfer from Ring 3 to Ring 0

Mode Transfer



1. Application executes trap (int) instruction
   - EIP, CS, and EFLAGS get pushed onto the stack
   - Mode switches from ring 3 to ring 0

Mode Transfer

Userland

1. Application executes trap (int) instruction
   - EIP, CS, and EFLAGS get pushed onto the stack
   - Mode switches from ring 3 to ring 0

Kernel Mode

2. Save the state of the current process
   - Push EAX, EBX, …, etc. onto the stack

Mode Transfer

Userland

1. Application executes trap (int) instruction
   - EIP, CS, and EFLAGS get pushed onto the stack
   - Mode switches from ring 3 to ring 0

Kernel Mode

2. Save the state of the current process
   - Push EAX, EBX, …, etc. onto the stack
3. Locate and execute the correct syscall handler

Mode Transfer

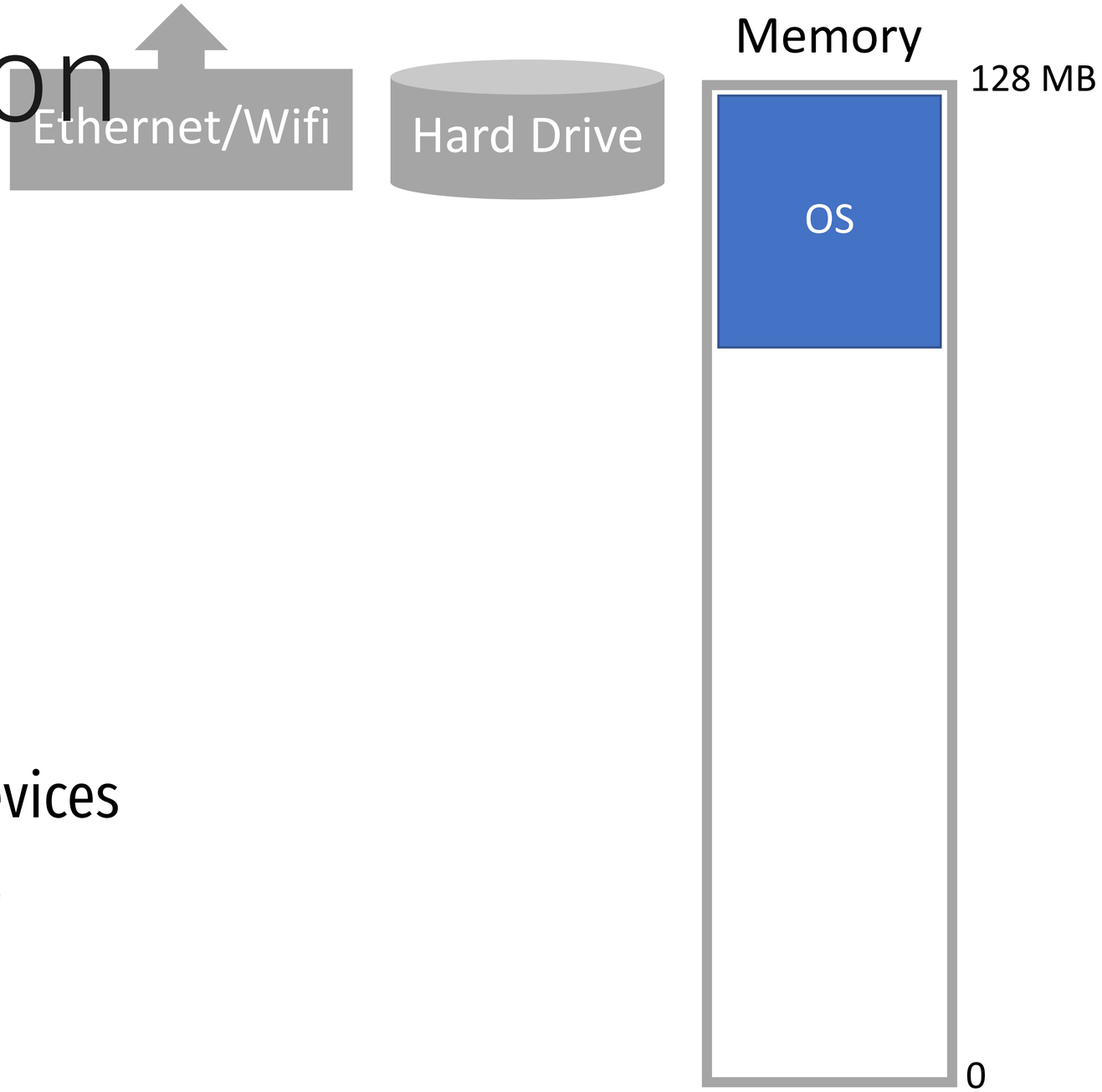
Userland

1. Application executes trap (int) instruction
   - EIP, CS, and EFLAGS get pushed onto the stack
   - Mode switches from ring 3 to ring 0

Kernel Mode

2. Save the state of the current process
   - Push EAX, EBX, …, etc. onto the stack
3. Locate and execute the correct syscall handler
4. Restore the state of process
   - Pop EAX, EBX, … etc.
5. Place the return value in EAX
6. Use iret to return to the process
   - Switches back to the original mode (typically 3)

Protection in Action

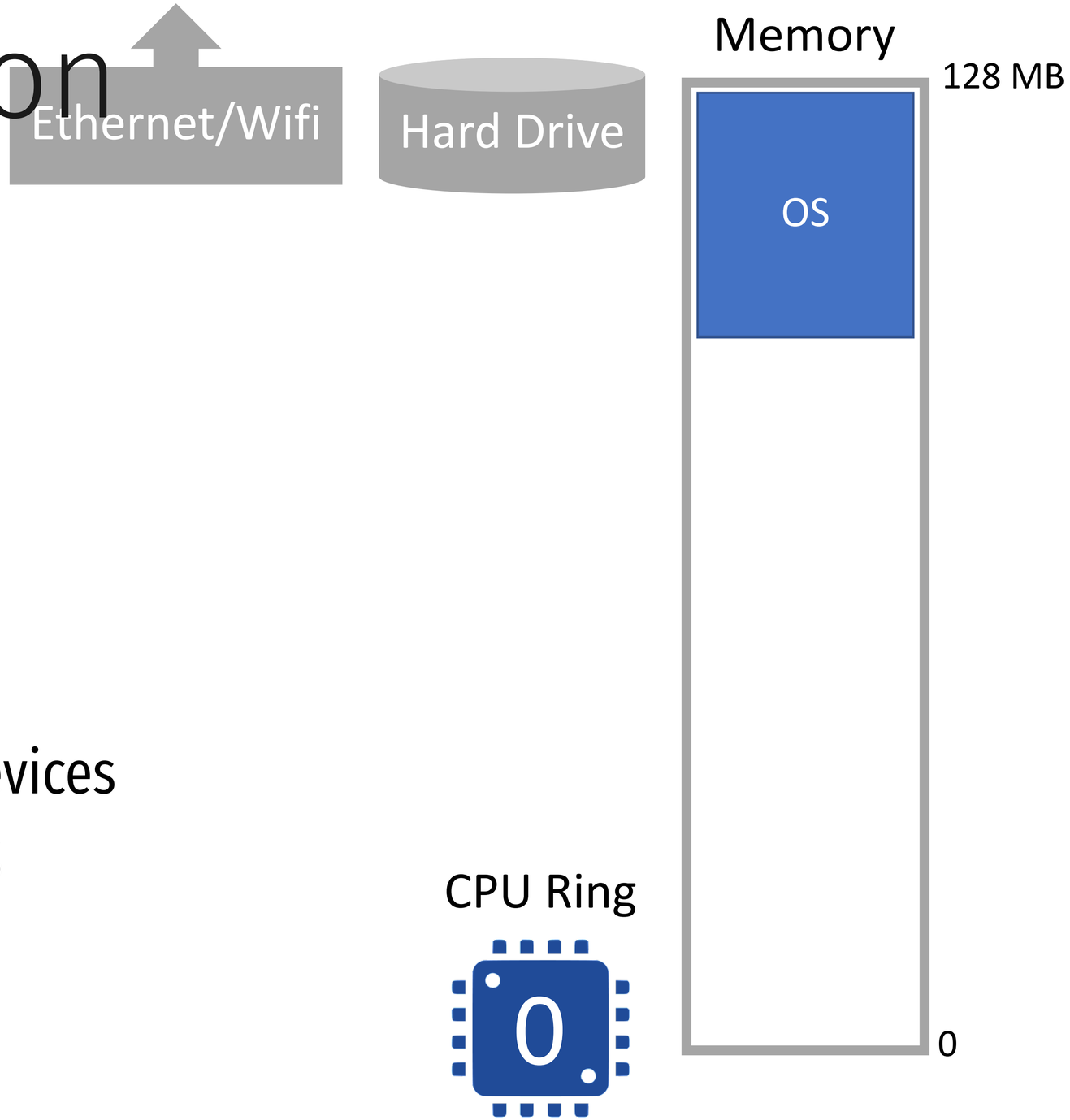


Protected mode stops direct access to devices

All device access must go through the OS

OS will impose access control checks

Protection in Action

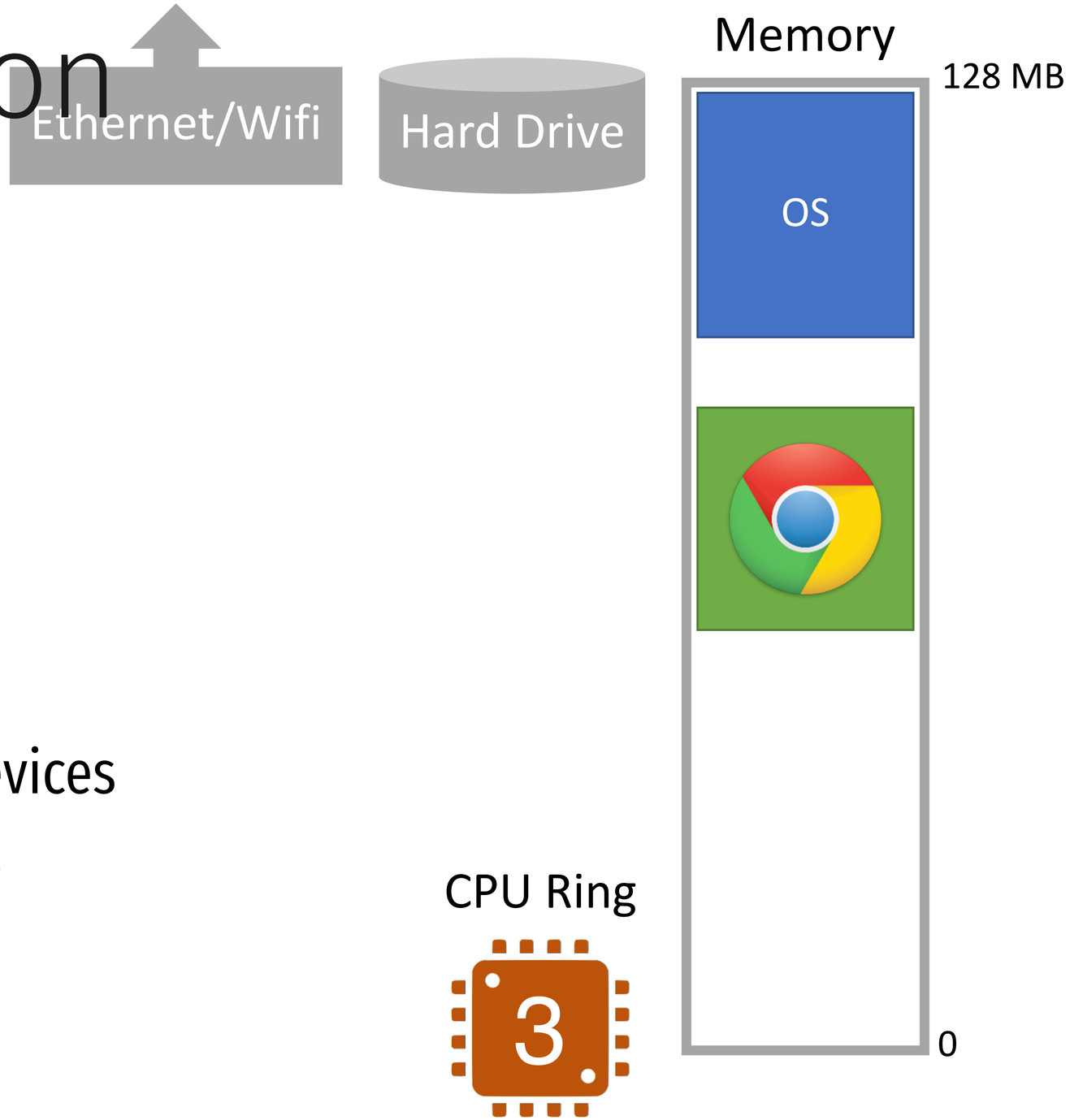


Protected mode stops direct access to devices

All device access must go through the OS

OS will impose access control checks

Protection in Action

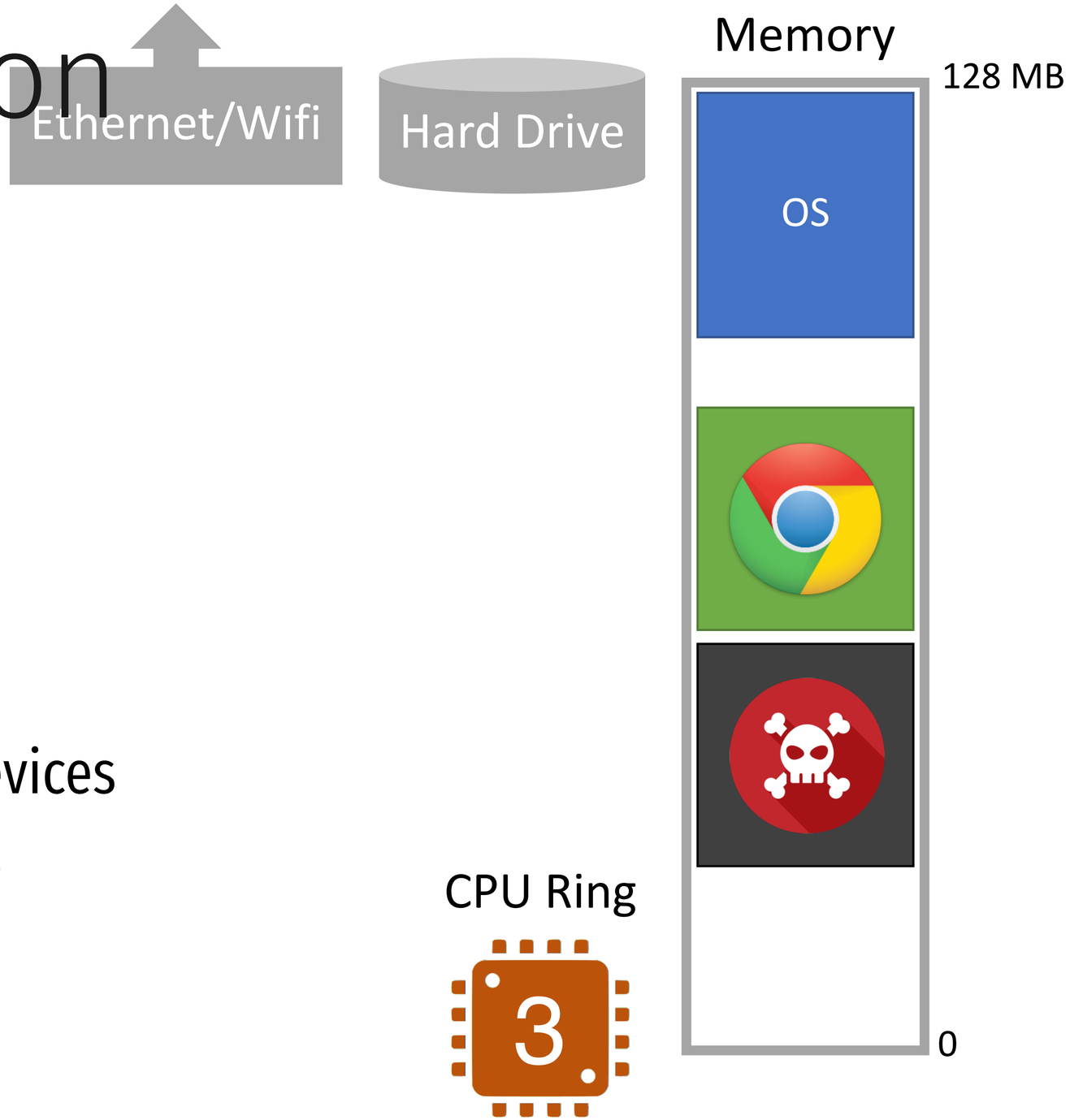


Protected mode stops direct access to devices

All device access must go through the OS

OS will impose access control checks

Protection in Action

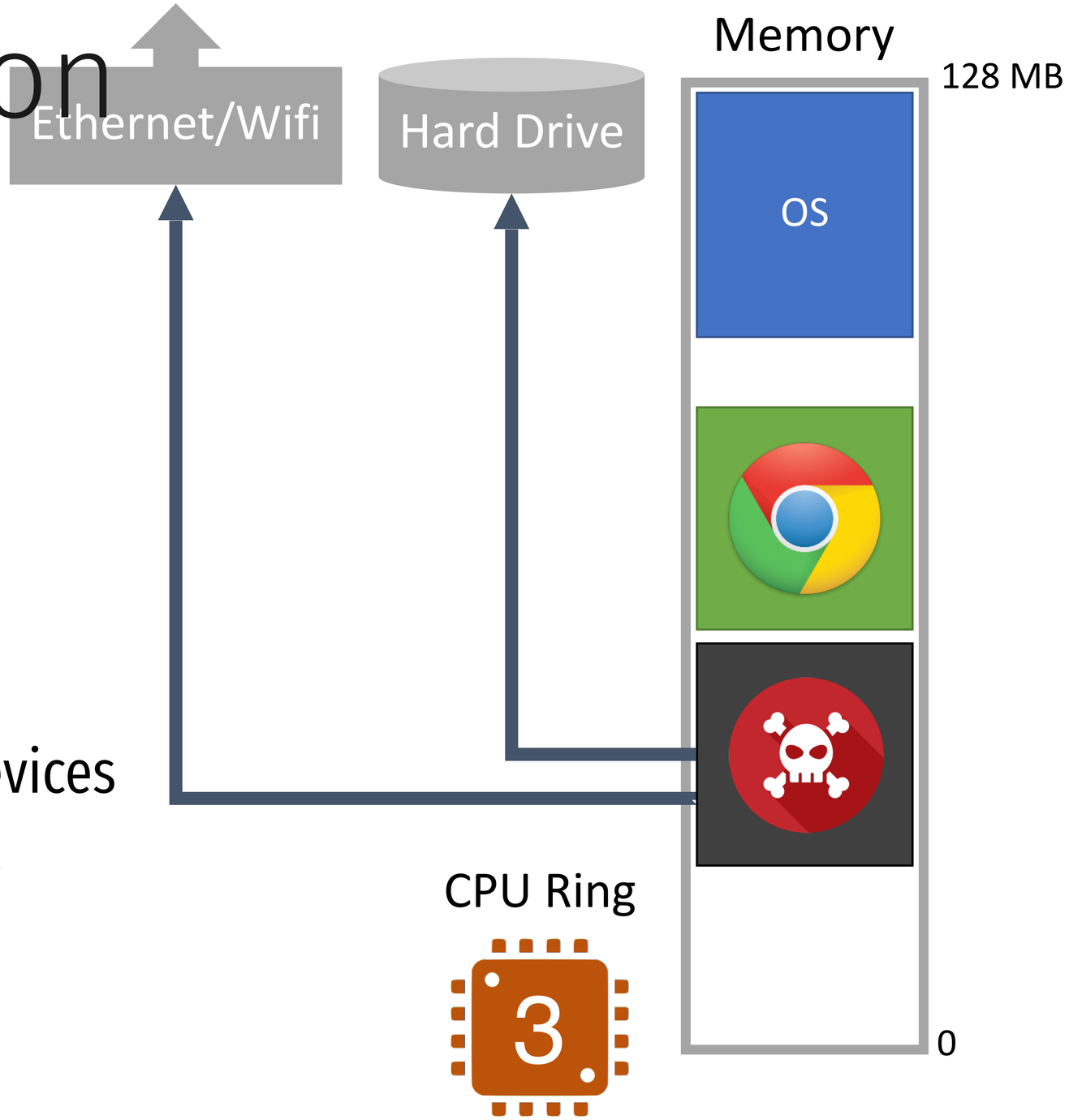


Protected mode stops direct access to devices

All device access must go through the OS

OS will impose access control checks

Protection in Action

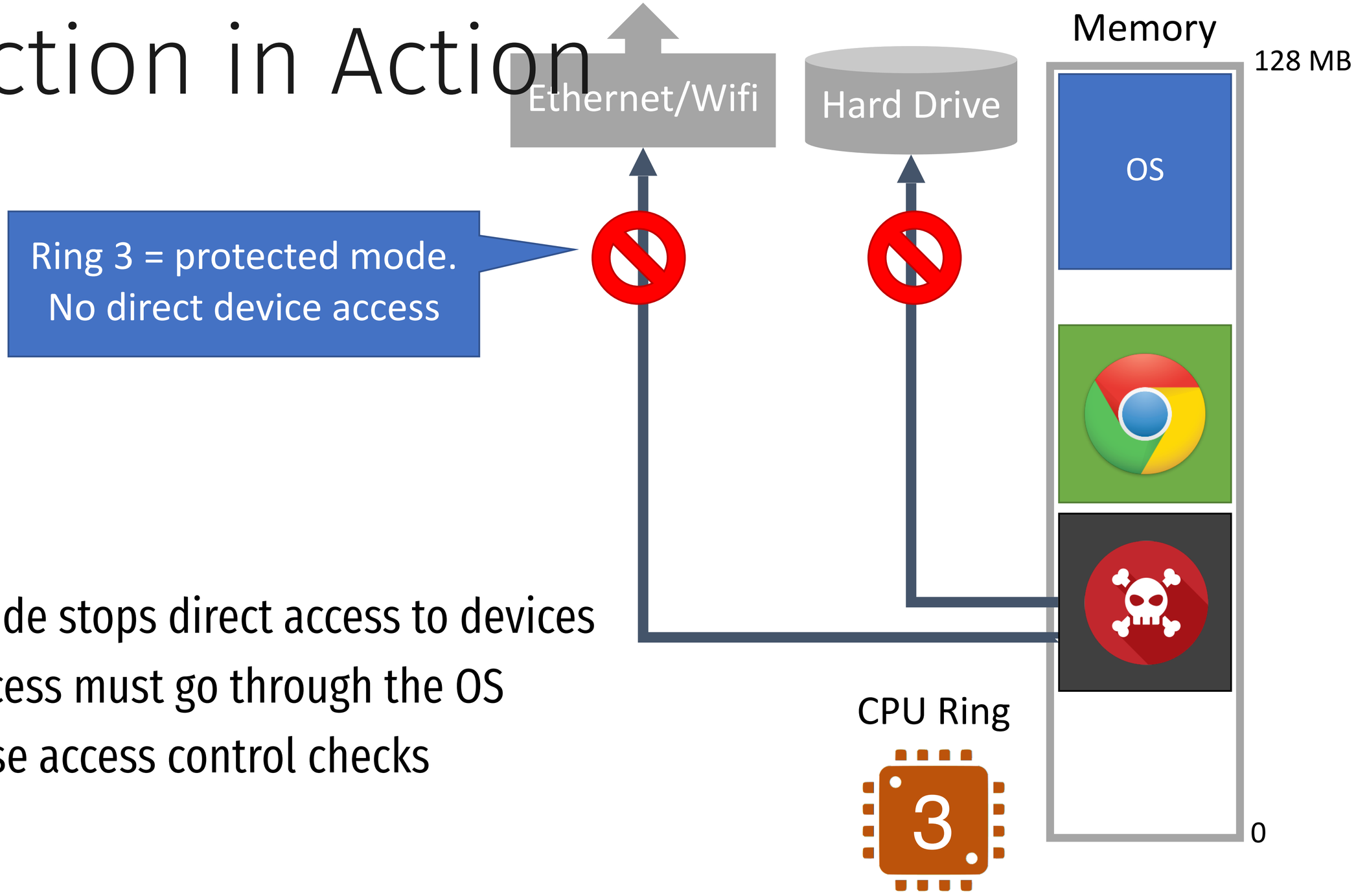


Protected mode stops direct access to devices

All device access must go through the OS

OS will impose access control checks

Protection in Action

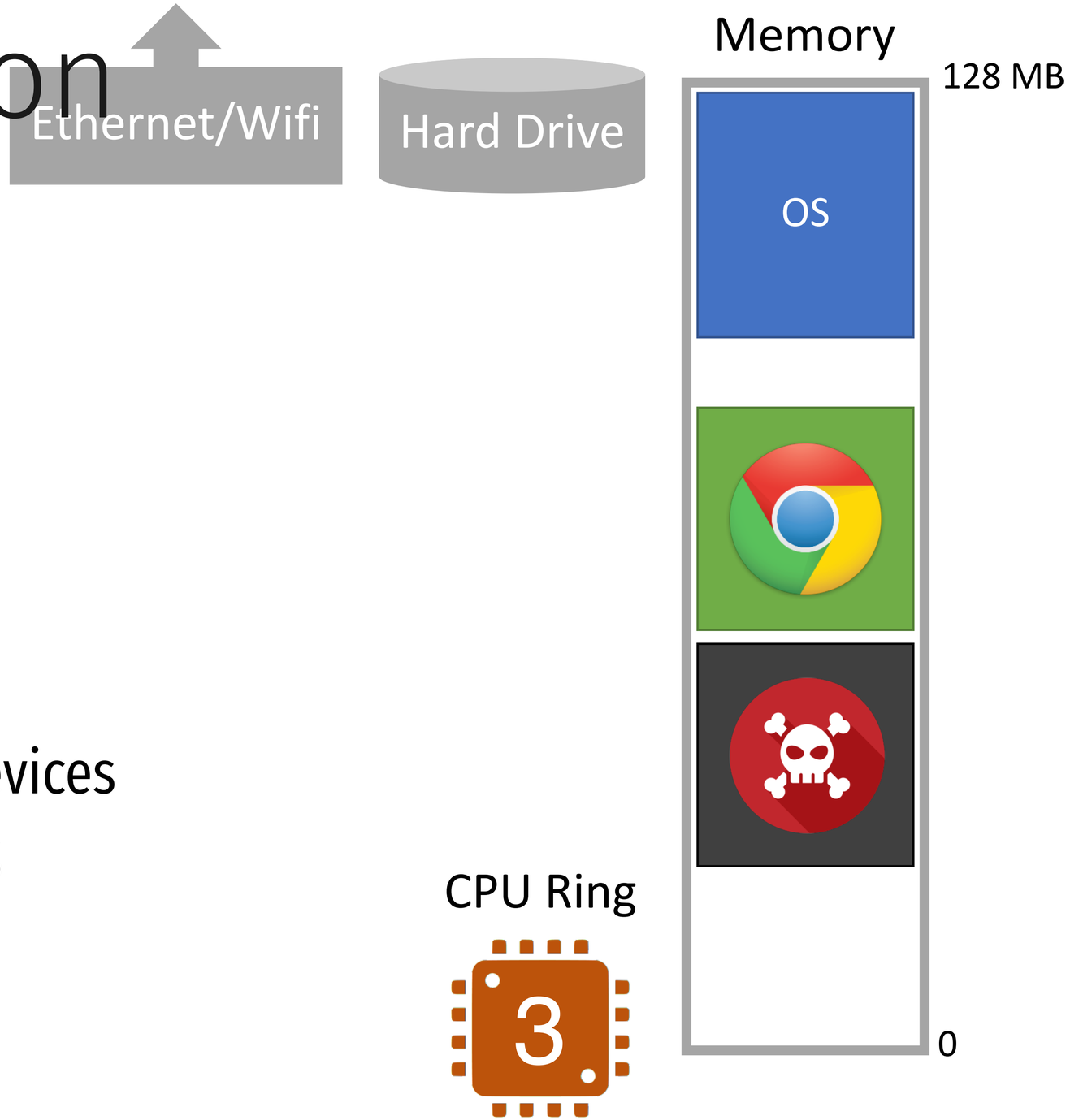


Protected mode stops direct access to devices

All device access must go through the OS

OS will impose access control checks

Protection in Action

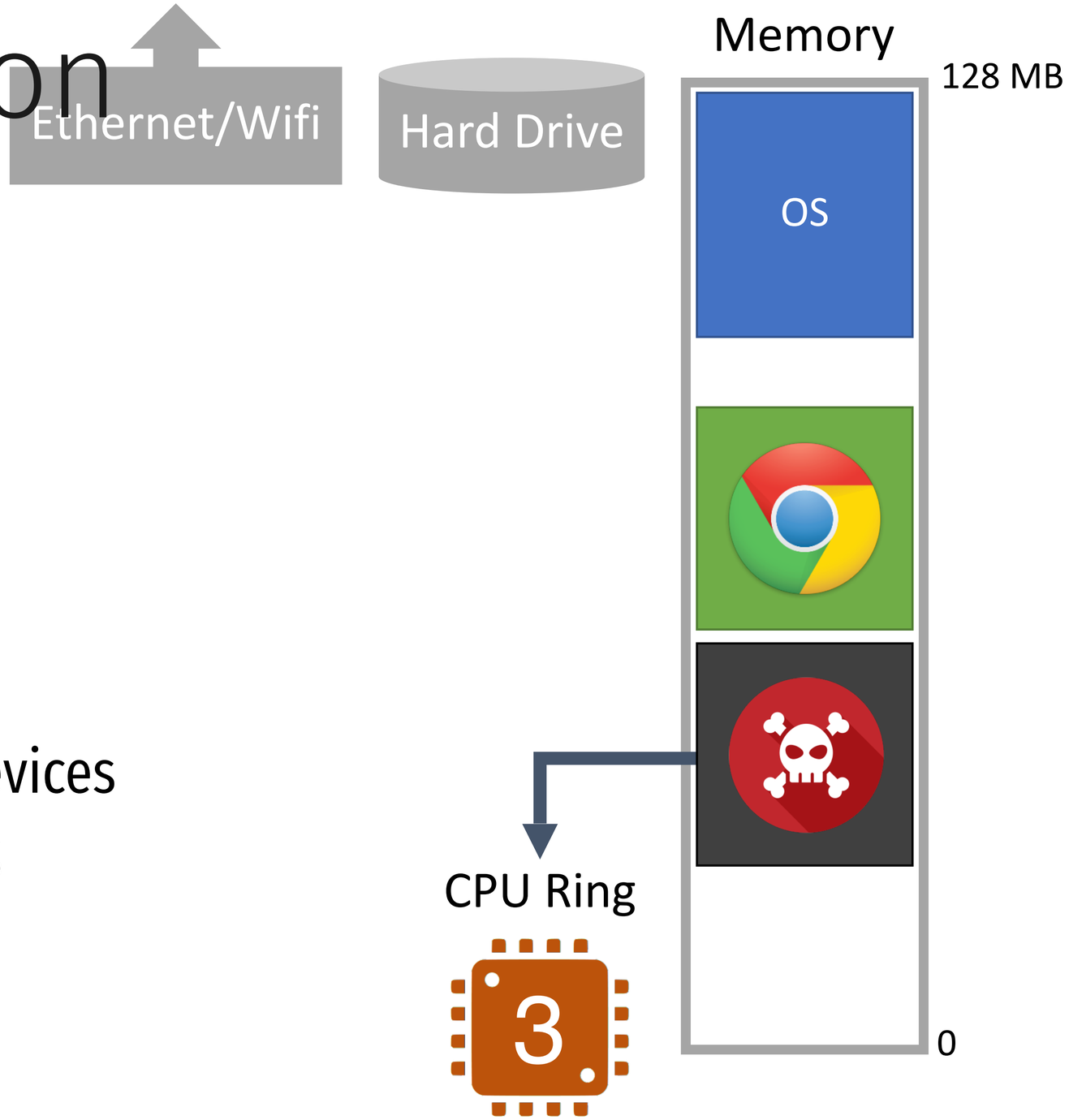


Protected mode stops direct access to devices

All device access must go through the OS

OS will impose access control checks

Protection in Action

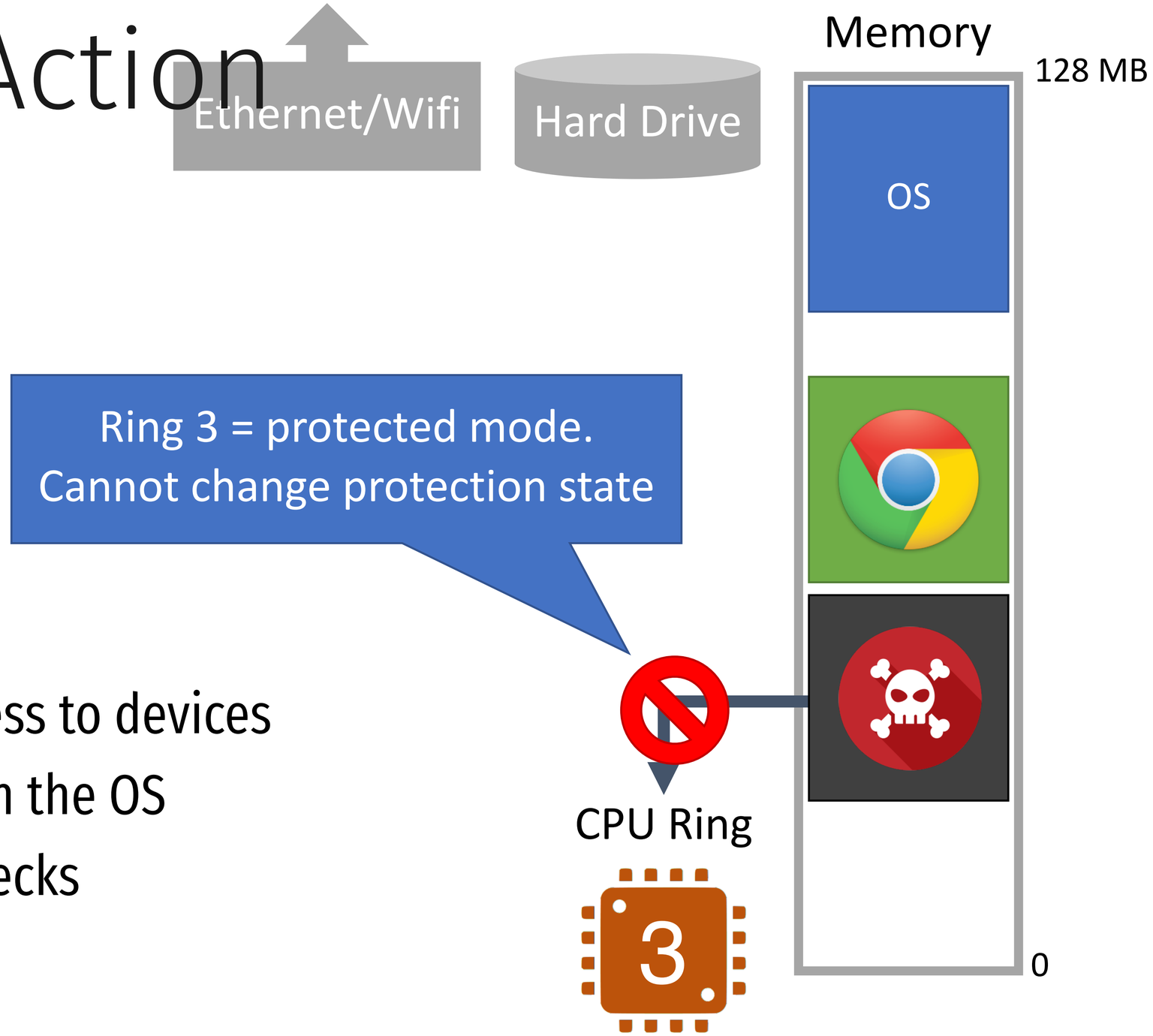


Protected mode stops direct access to devices

All device access must go through the OS

OS will impose access control checks

Protection in Action

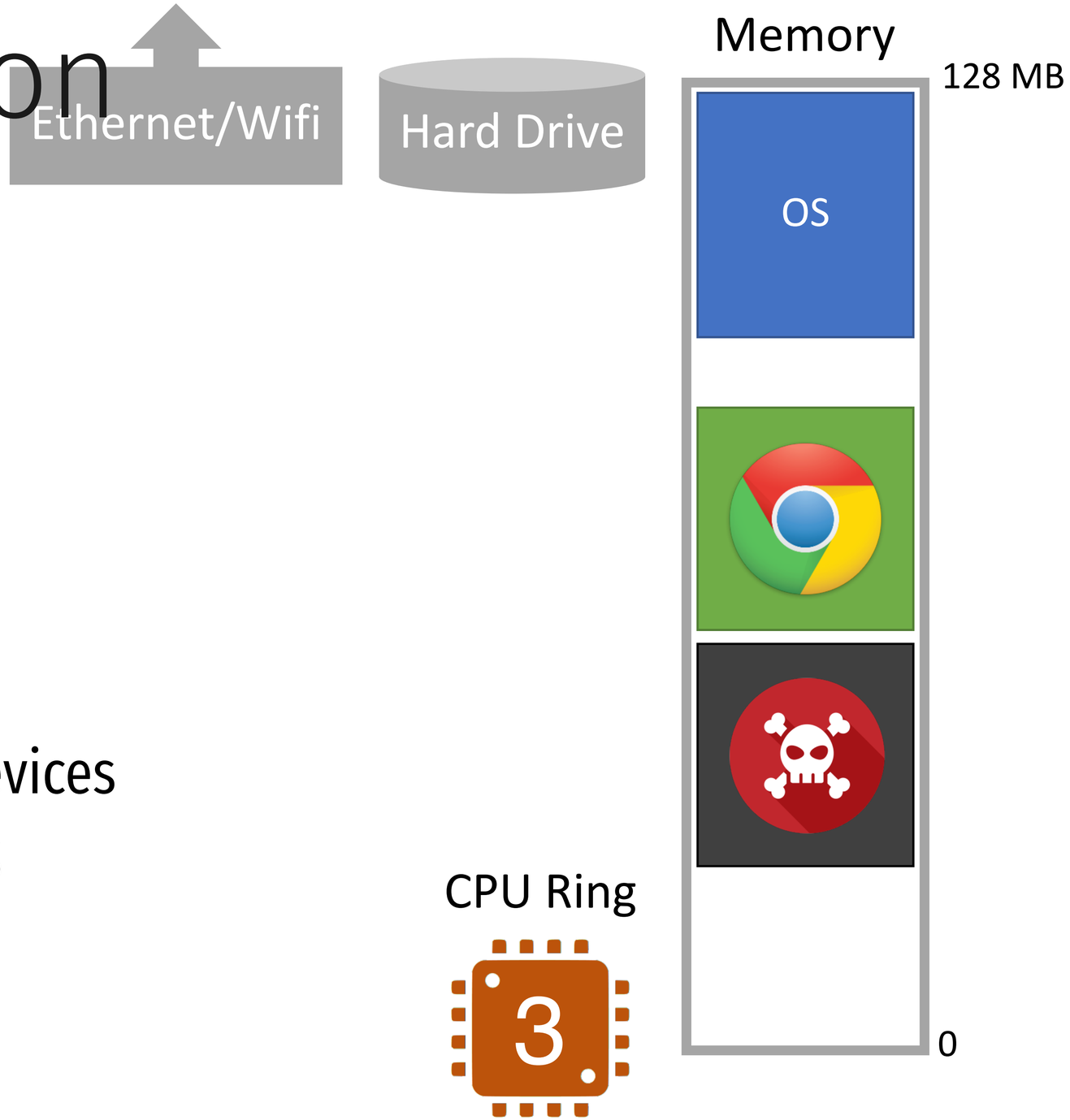


Protected mode stops direct access to devices

All device access must go through the OS

OS will impose access control checks

Protection in Action

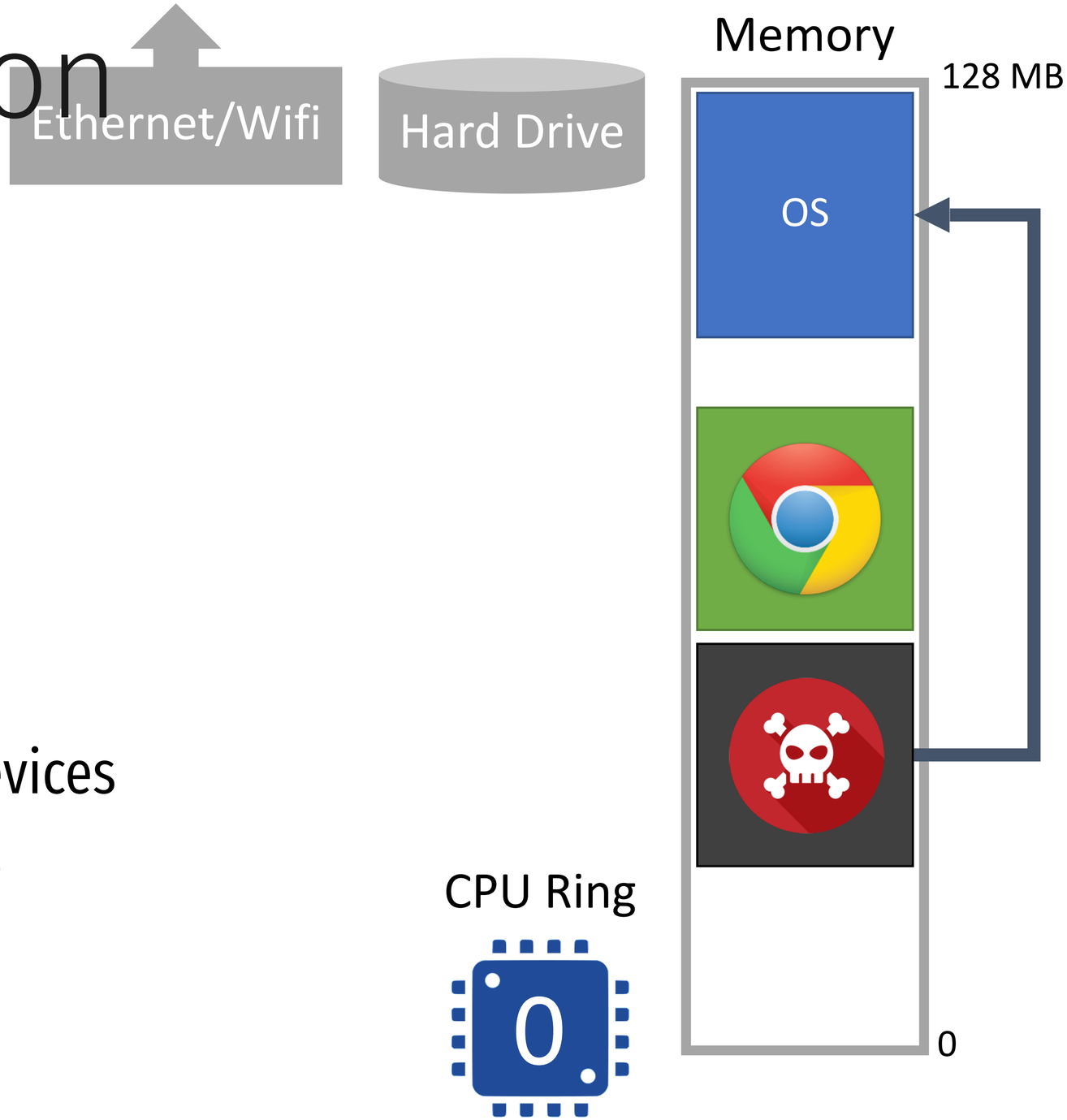


Protected mode stops direct access to devices

All device access must go through the OS

OS will impose access control checks

Protection in Action

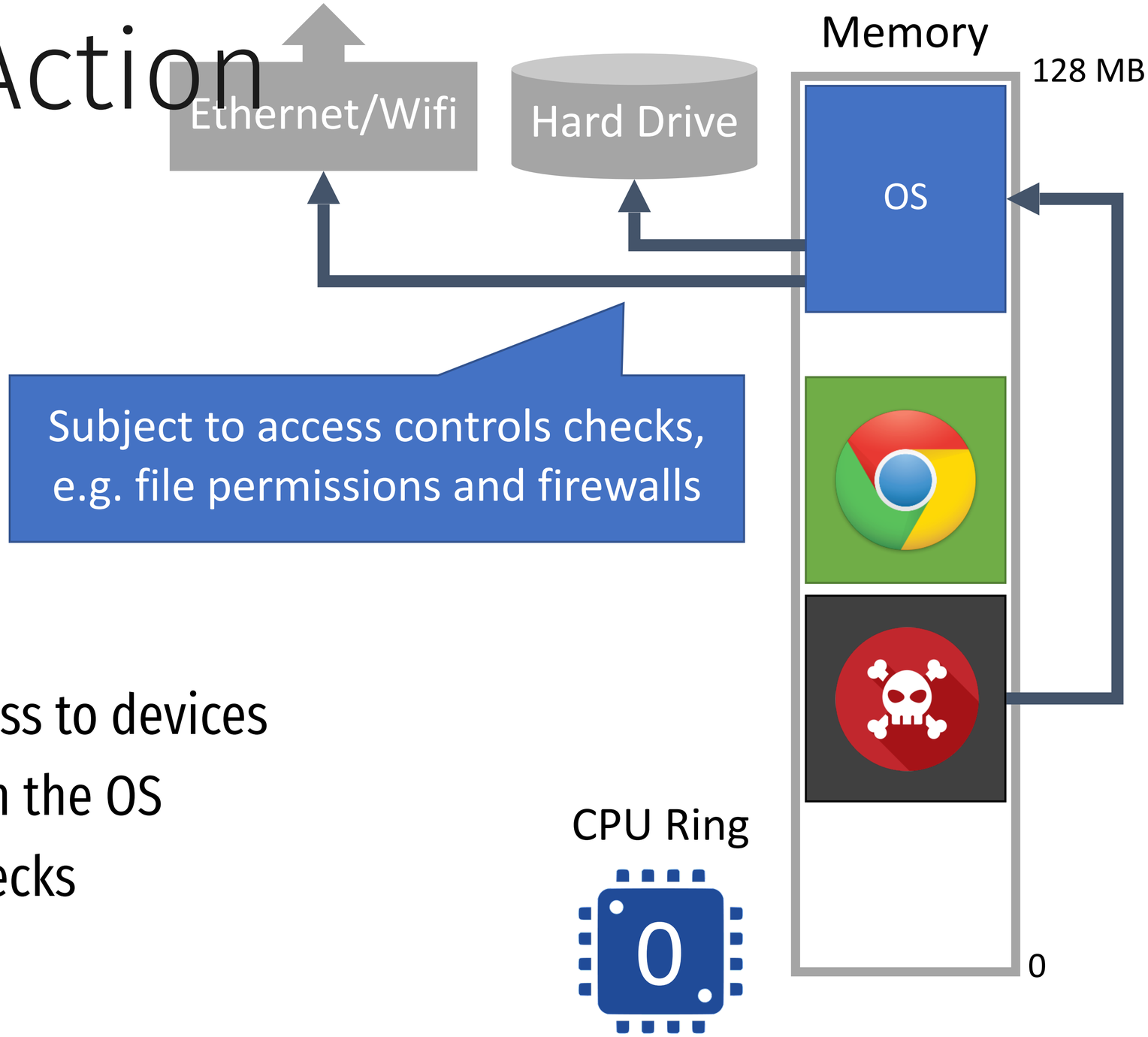


Protected mode stops direct access to devices

All device access must go through the OS

OS will impose access control checks

Protection in Action



Protected mode stops direct access to devices

All device access must go through the OS

OS will impose access control checks