



Behind the Scenes of iOS and Mac Security

Ivan Krstić

Head of Security Engineering and Architecture, Apple

Mac secure boot

iOS code integrity protection

Find My

Mac secure boot

iOS code integrity protection

Find My



Gatekeeper



User Privacy
Protection

Gatekeeper

macOS Catalina



First use, quarantined



First use, quarantined



Non-quarantined

Malicious content scan

No known malicious content

No known malicious content

No known malicious content

Signature check

No tampering

No tampering

–

Local policy check

All new software requires notarization

All new software requires notarization

–

First launch prompt

User must approve

Users must approve software in bundles

–

User Data Protections

Data that requires user consent to access

Contacts

Calendars

Reminders

Photos

User Data Protections

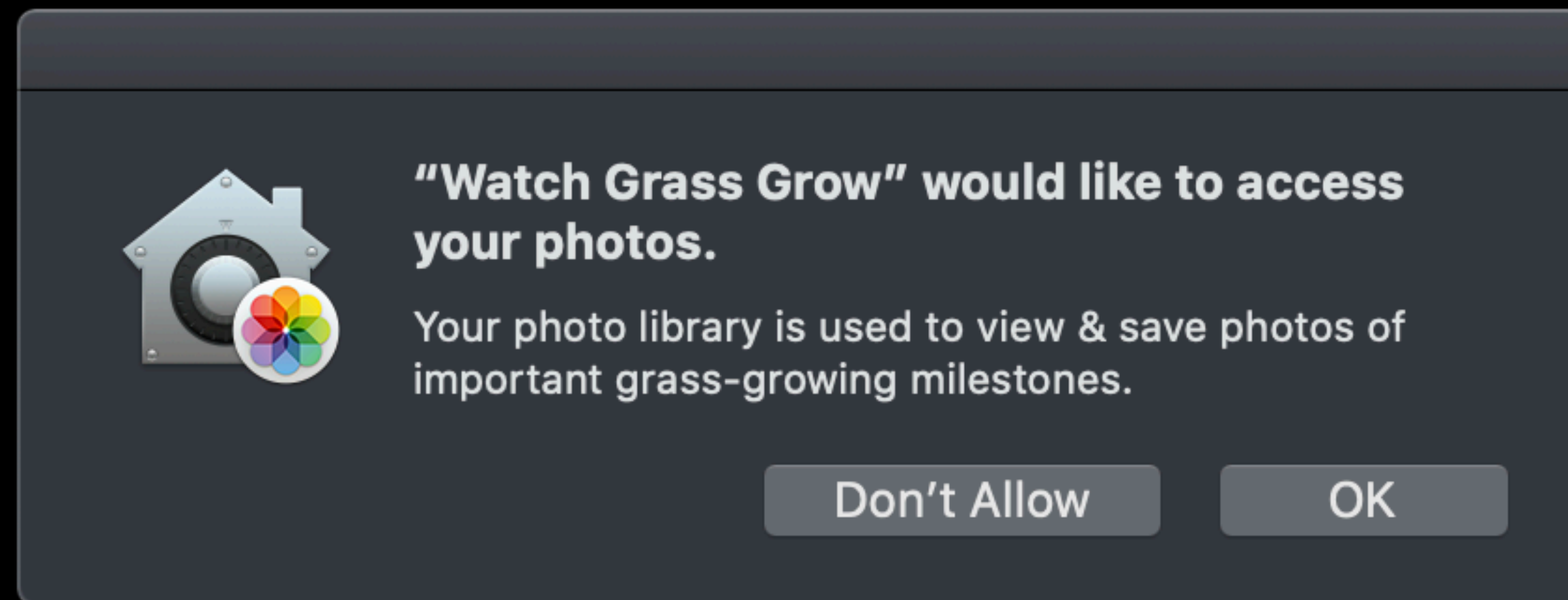
Data that requires user consent to access

Contacts

Calendars

Reminders

Photos



User Data Protections

Data that requires user consent to access

Contacts

Calendars

Reminders

Photos

Desktop

Documents

Downloads

iCloud Drive

Third-party cloud storage

Removable volumes

Network volumes

What about secure boot?

Apple Requirement**UEFI**

Signature verification of complete boot chain



System Software Authorization (server-side downgrade protection)



Authorization "personalized" for the requesting device (not portable)



User authentication required to downgrade secure boot policy



Secure boot policy protected against physical tamper



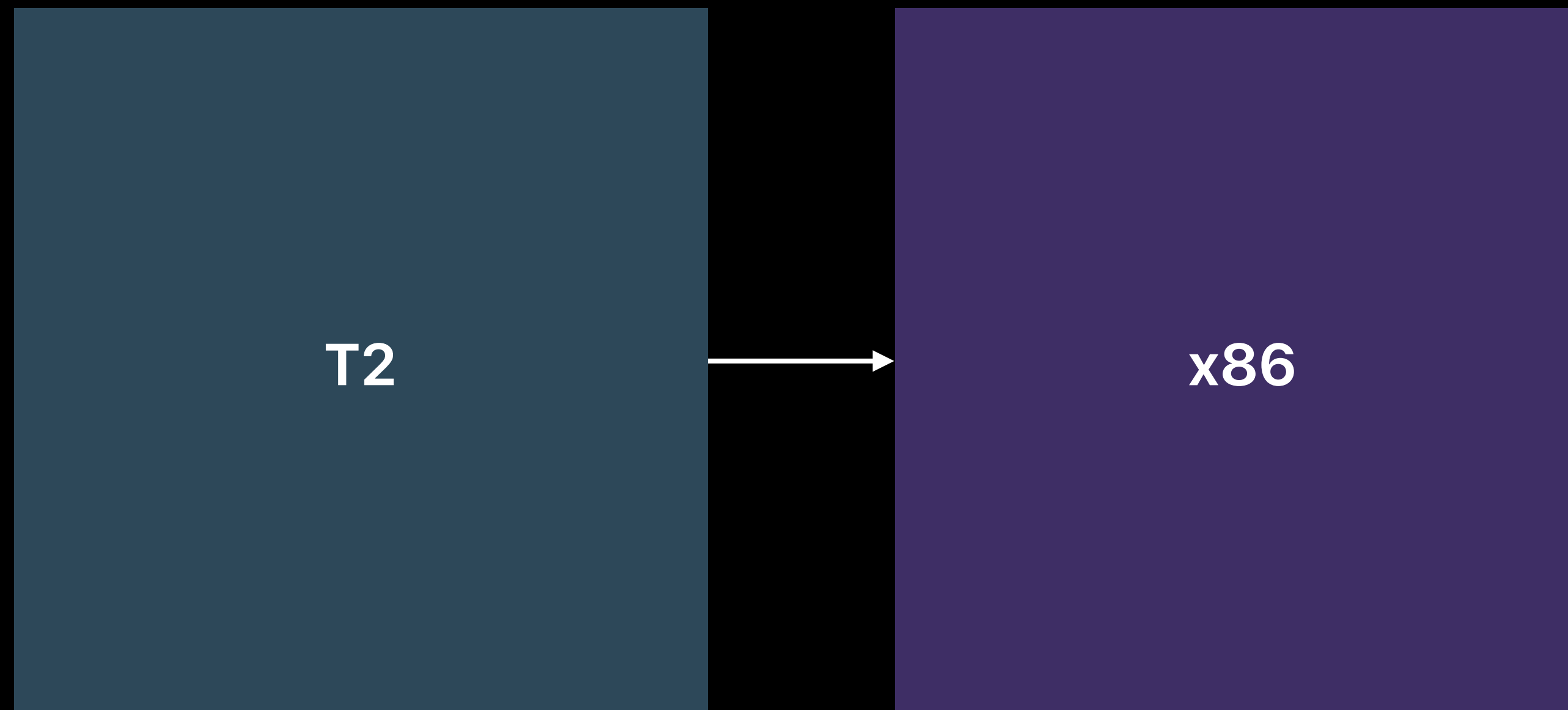
System can always be restored to known-good state



Mac Secure Boot



Mac Secure Boot

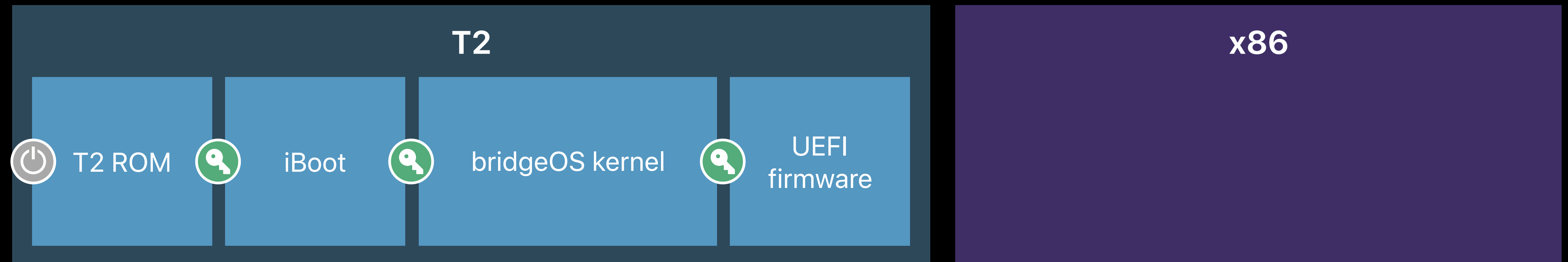


Mac Secure Boot

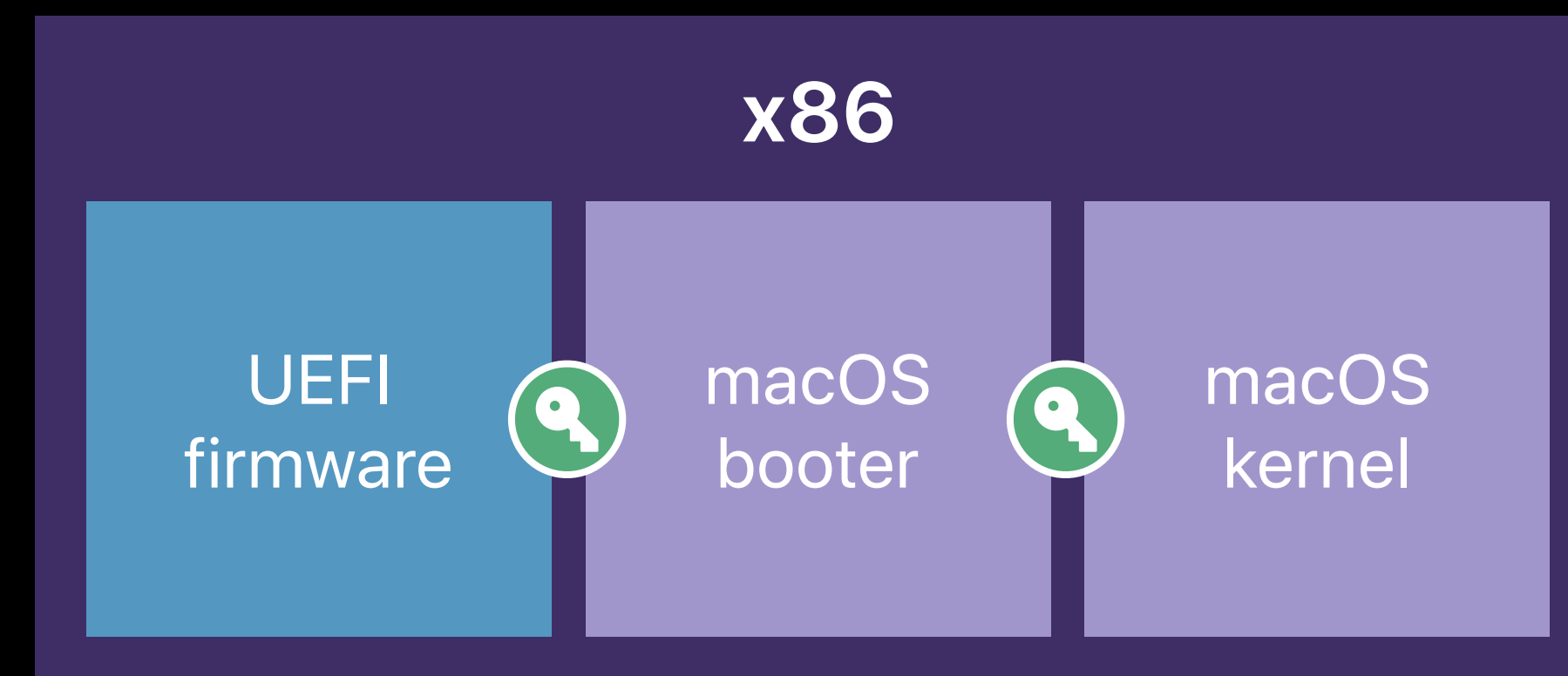
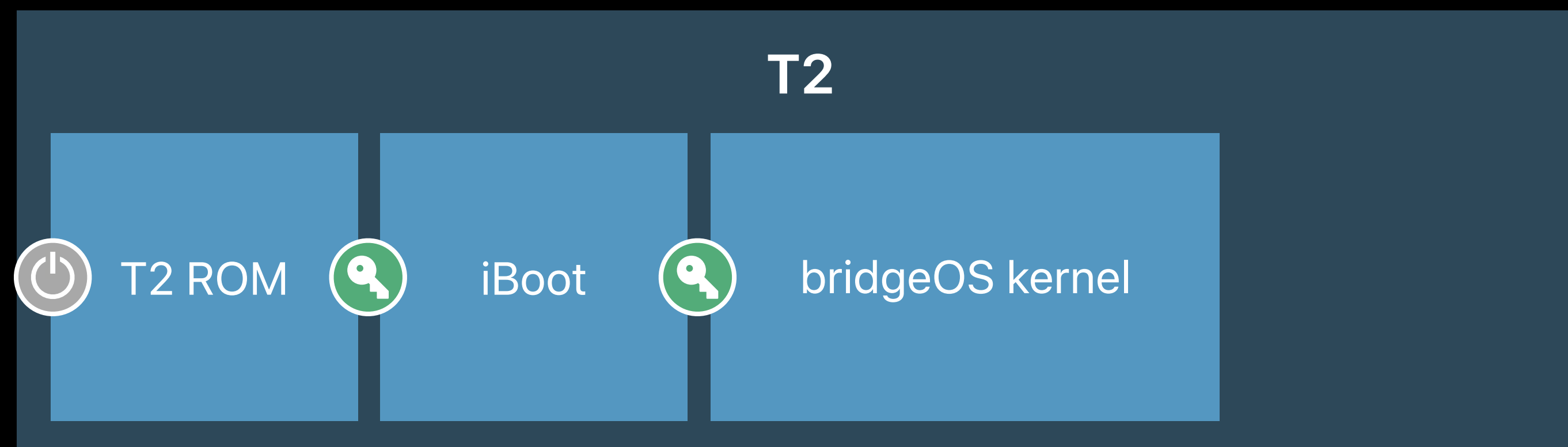
T2

x86

Mac Secure Boot



Mac Secure Boot



Two Critical Challenges

Thunderbolt and PCIe Direct Memory Access (DMA)

- Accessories can read/write host memory without the involvement of the CPU

PCIe Option ROMs (OROMs)

- Device-specific drivers for the early boot environment

Refresher — OS Page Tables

UEFI

macOS

x86 CPU

64 bit protected mode

“Read 4 bytes
from address
0x0000030000000000”

MMU hardware

Virtual memory enabled

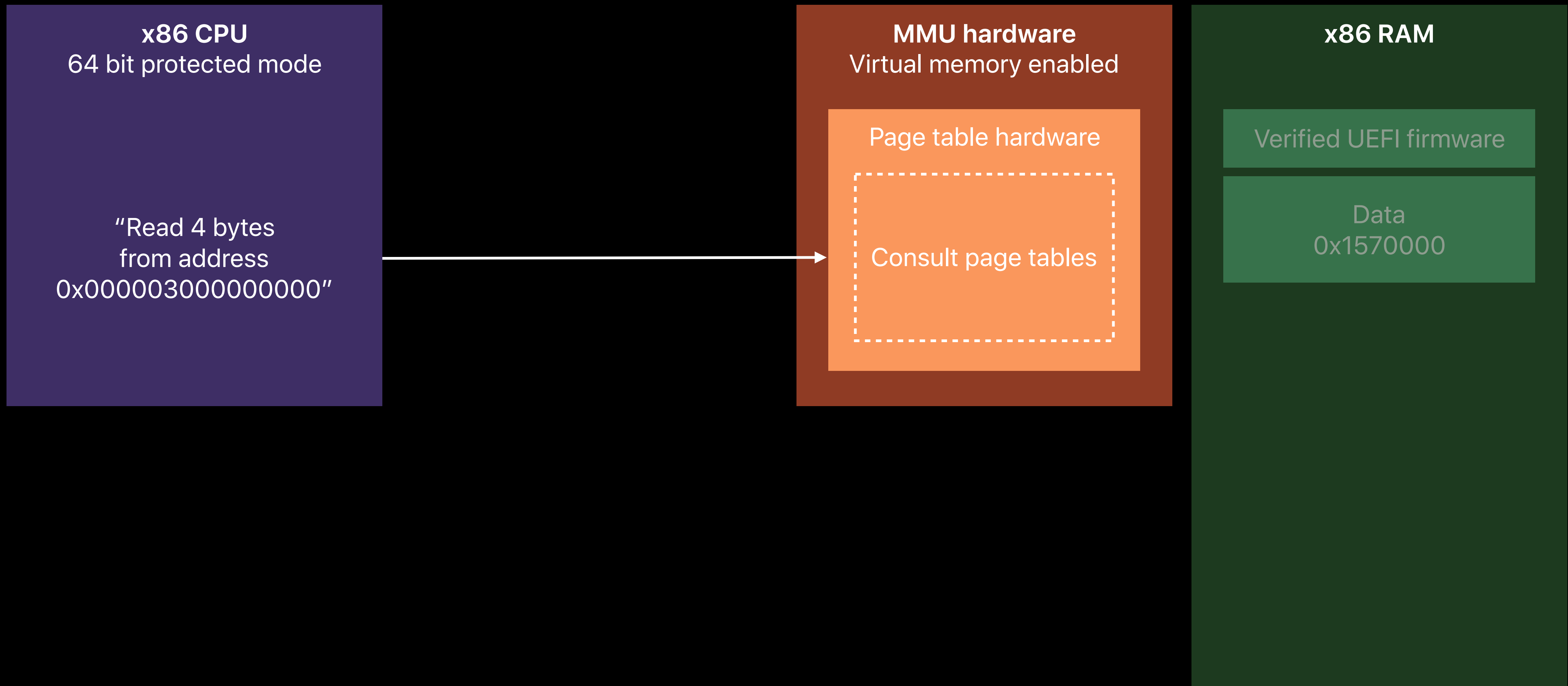
Page table hardware

x86 RAM

Verified UEFI firmware

Data
0x1570000

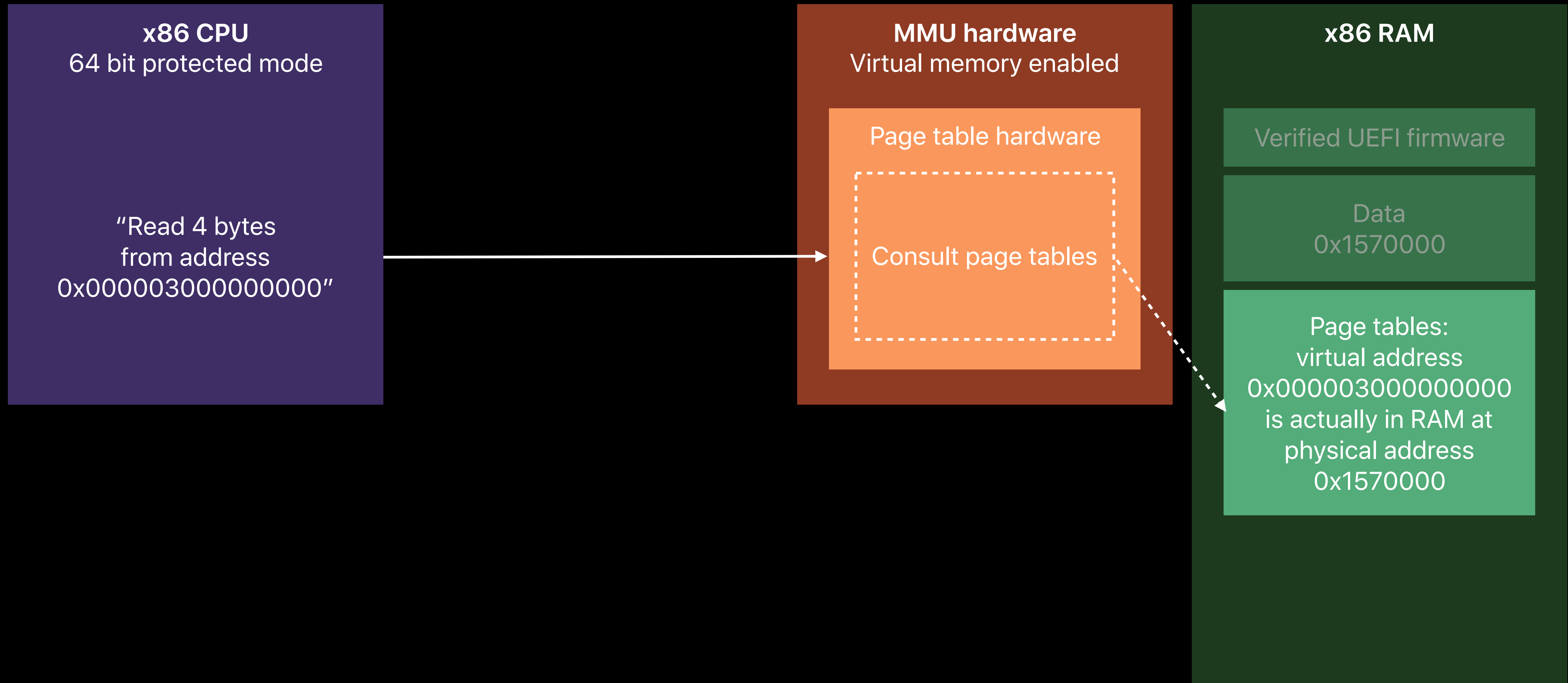
Refresher — OS Page Tables



Refresher — OS Page Tables

UEFI

macOS



Refresher — OS Page Tables

UEFI

macOS

x86 CPU

64 bit protected mode

“Read 4 bytes
from address
0x0000030000000000”

MMU hardware

Virtual memory enabled

Page table hardware

x86 RAM

Verified UEFI firmware

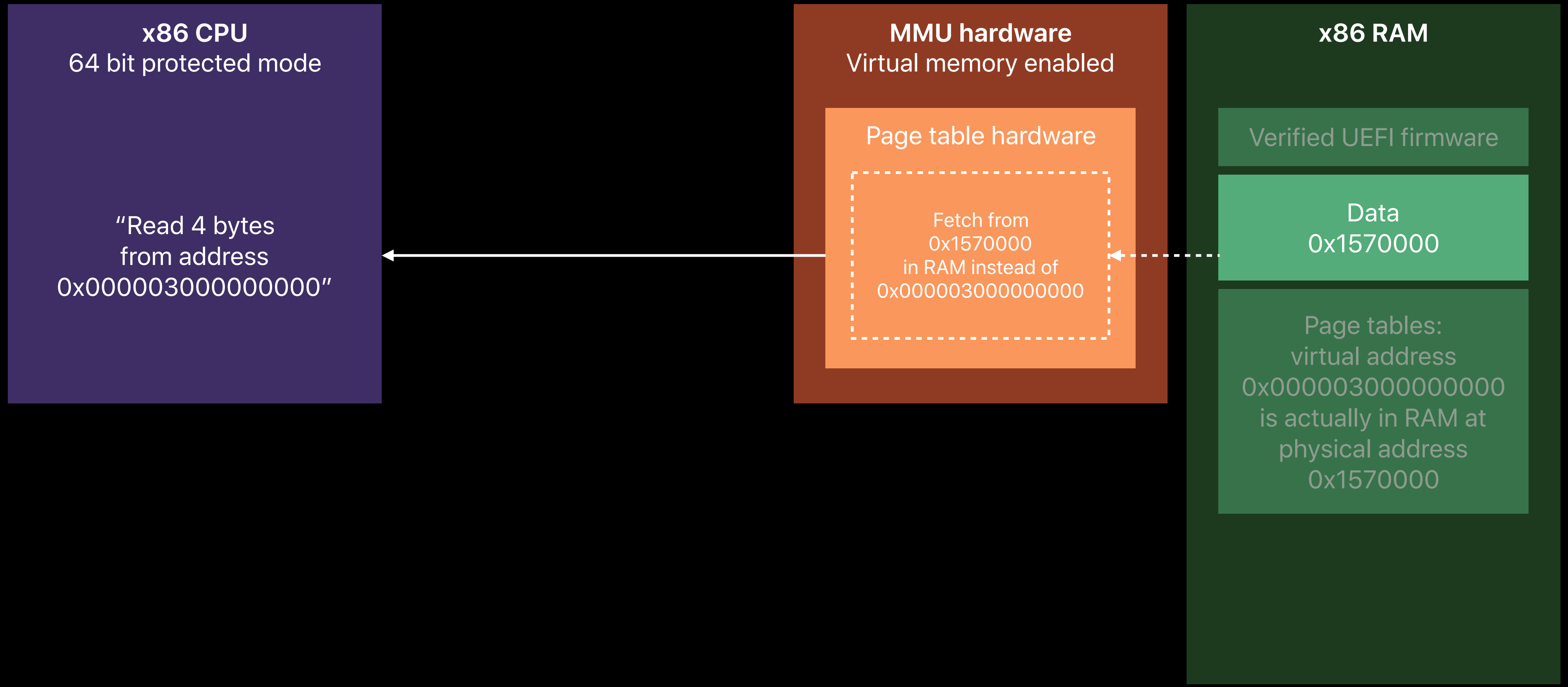
Data
0x1570000

Page tables:
virtual address
0x0000030000000000
is actually in RAM at
physical address
0x1570000

Refresher — OS Page Tables

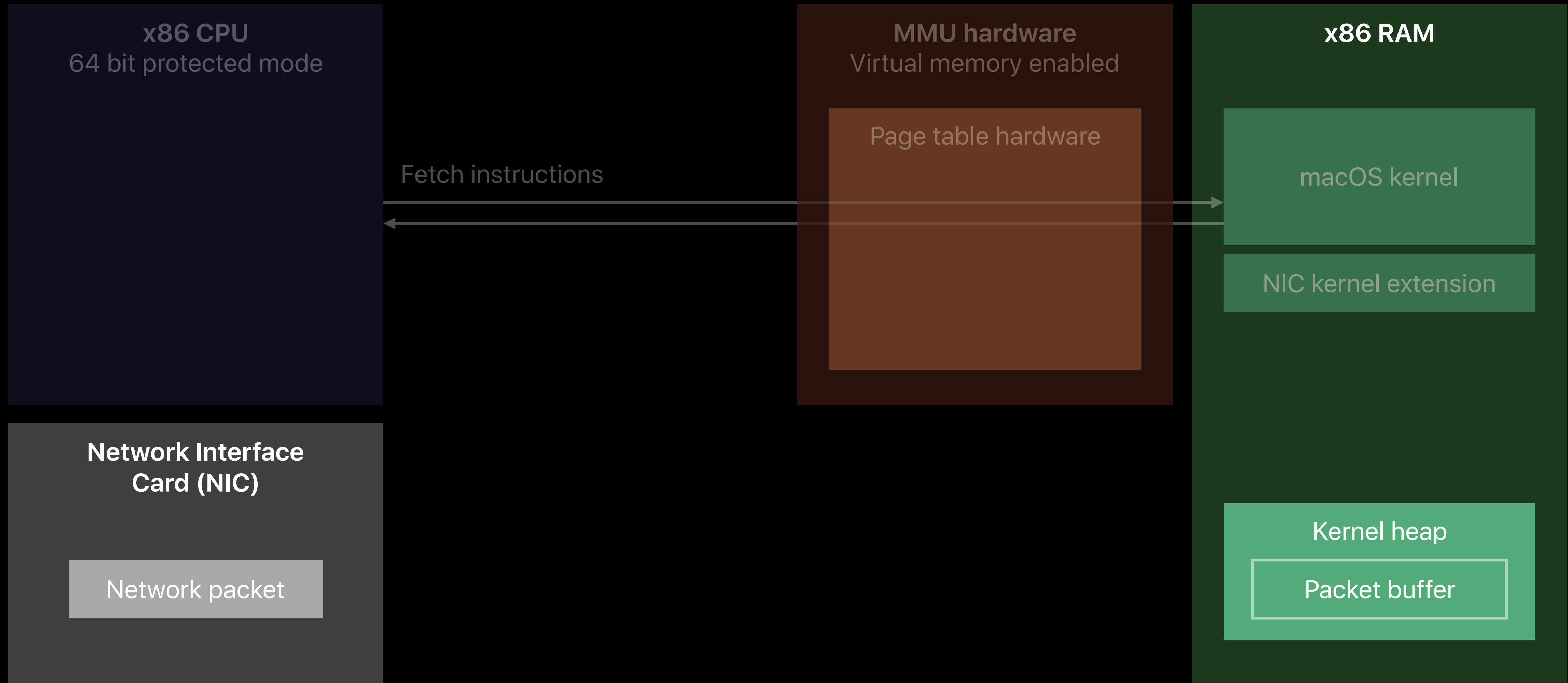
UEFI

macOS



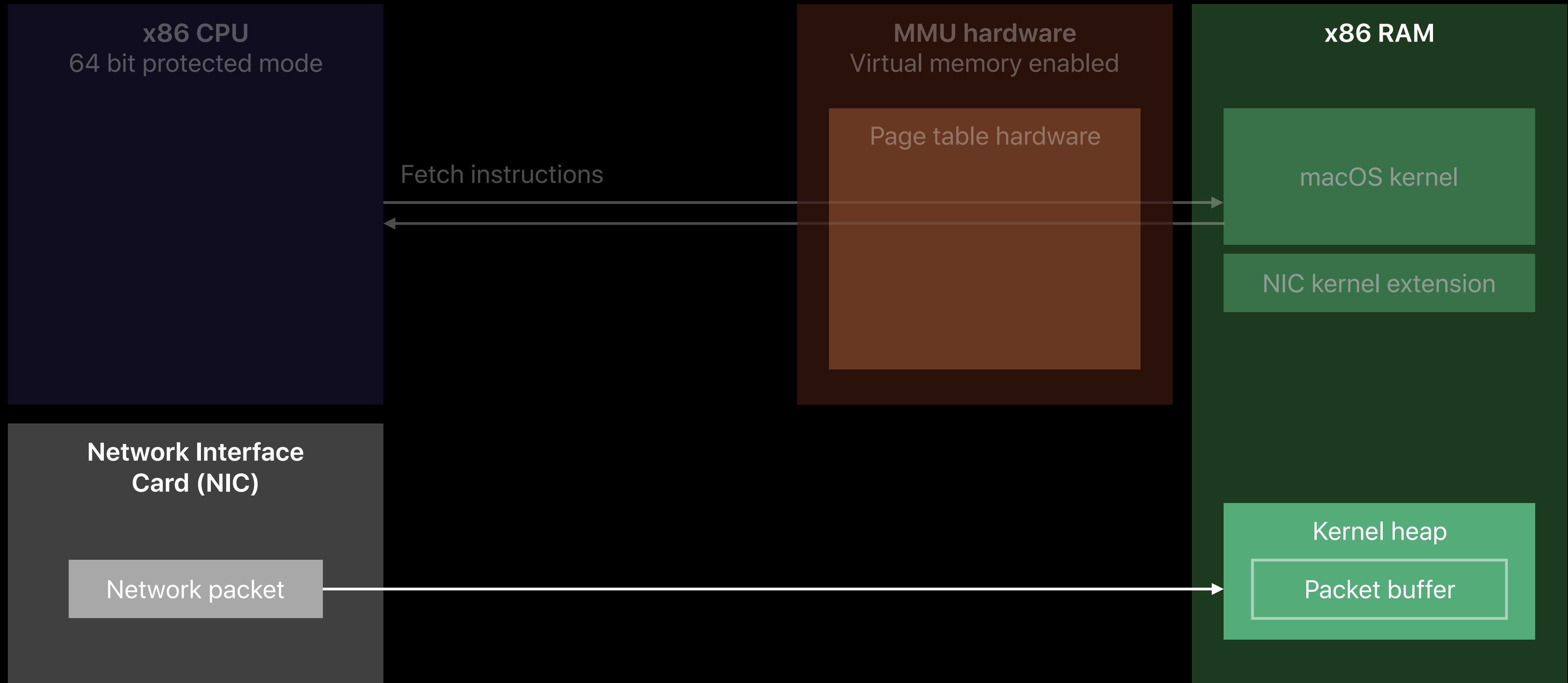
Unrestricted Direct Memory Access

UEFI macOS



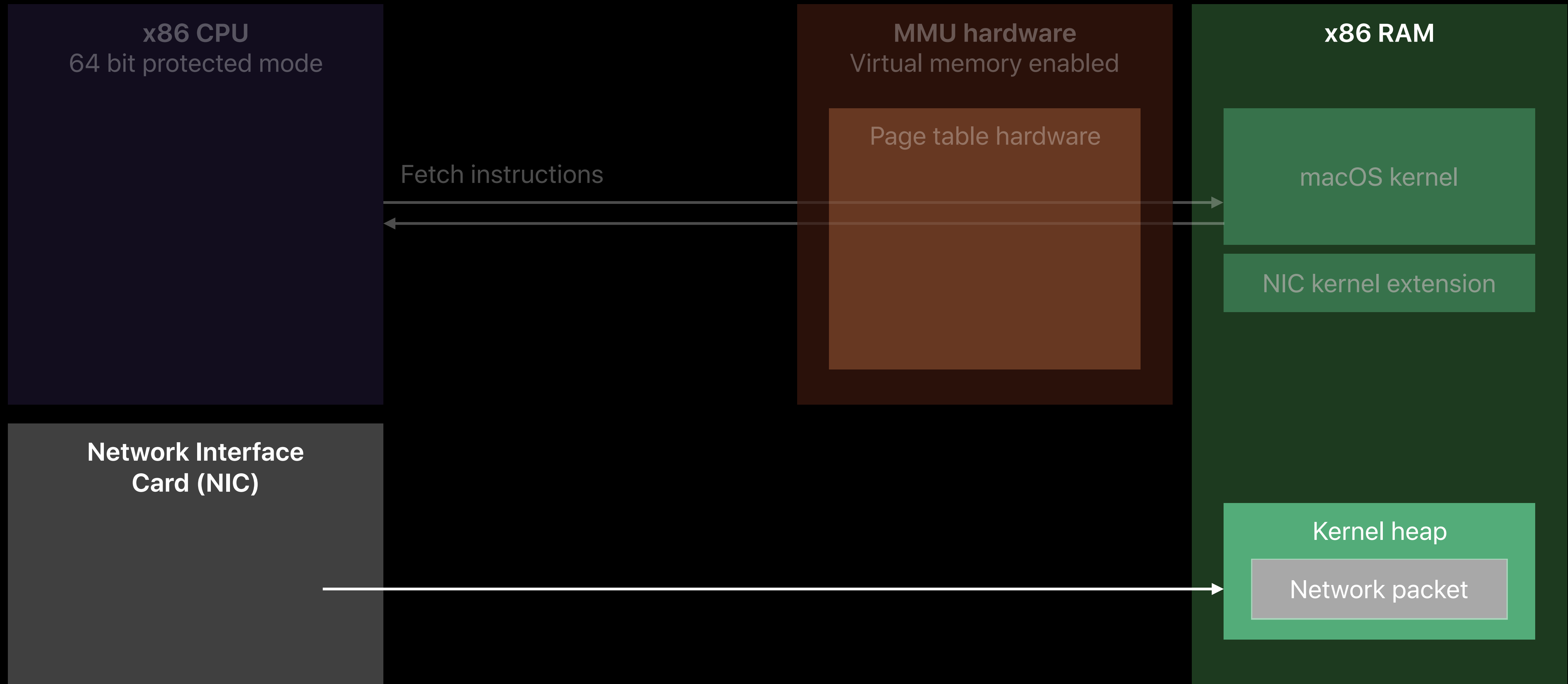
Unrestricted Direct Memory Access

UEFI macOS



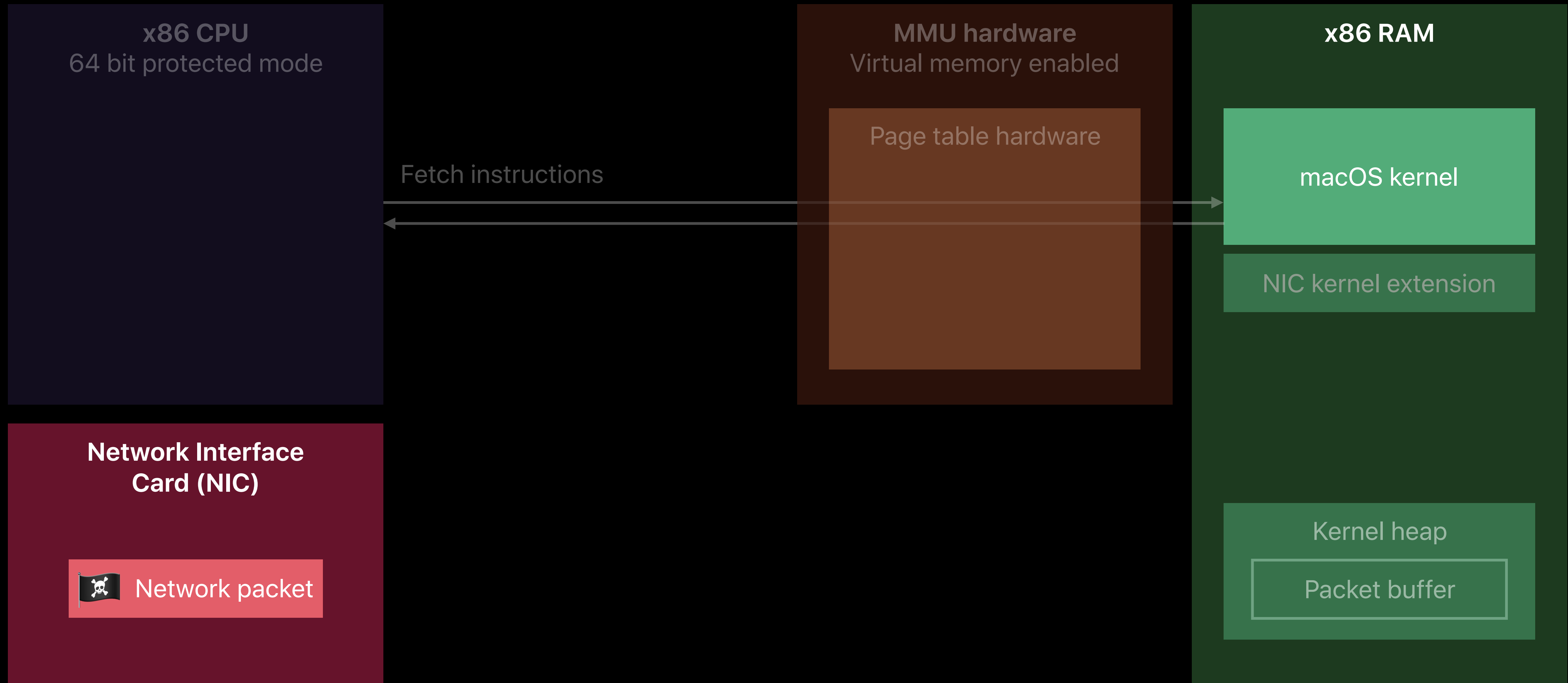
Unrestricted Direct Memory Access

UEFI macOS



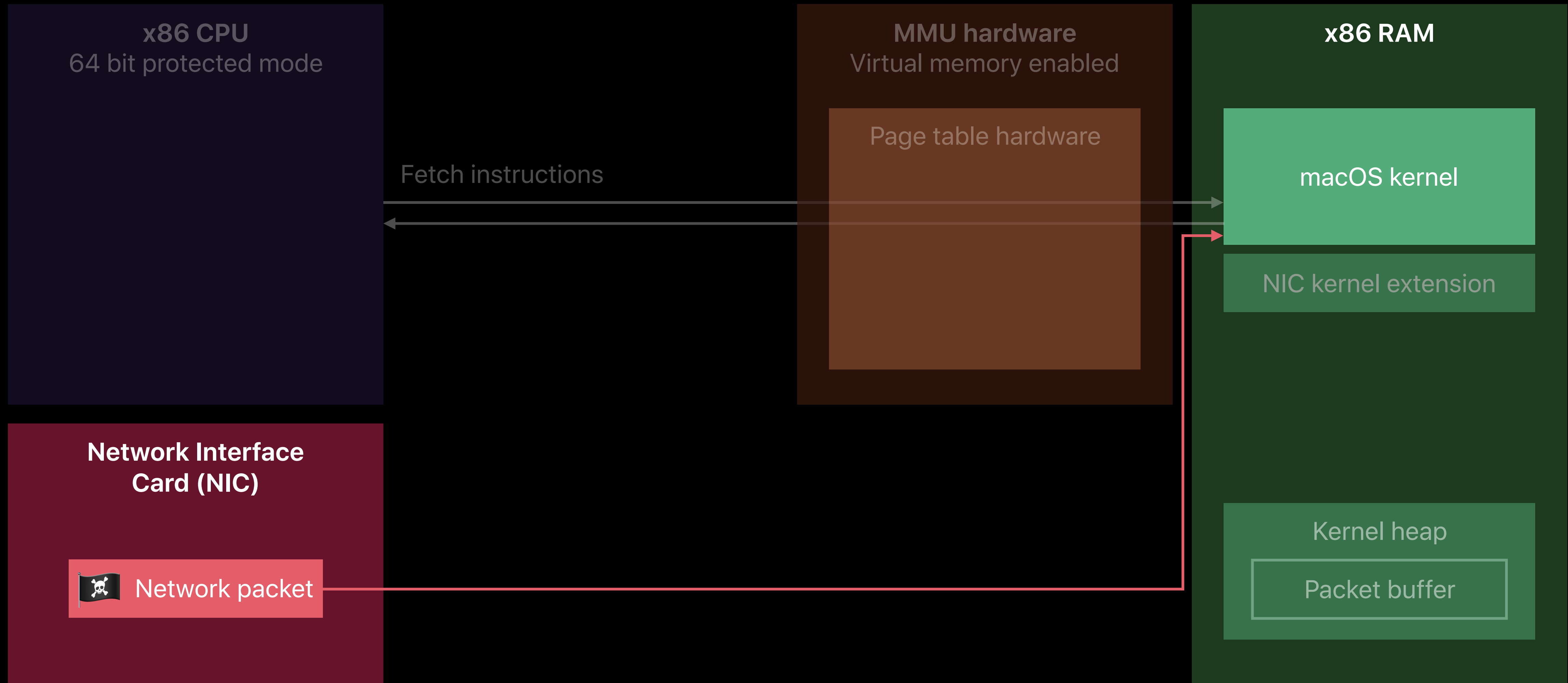
Unrestricted Direct Memory Access

UEFI macOS



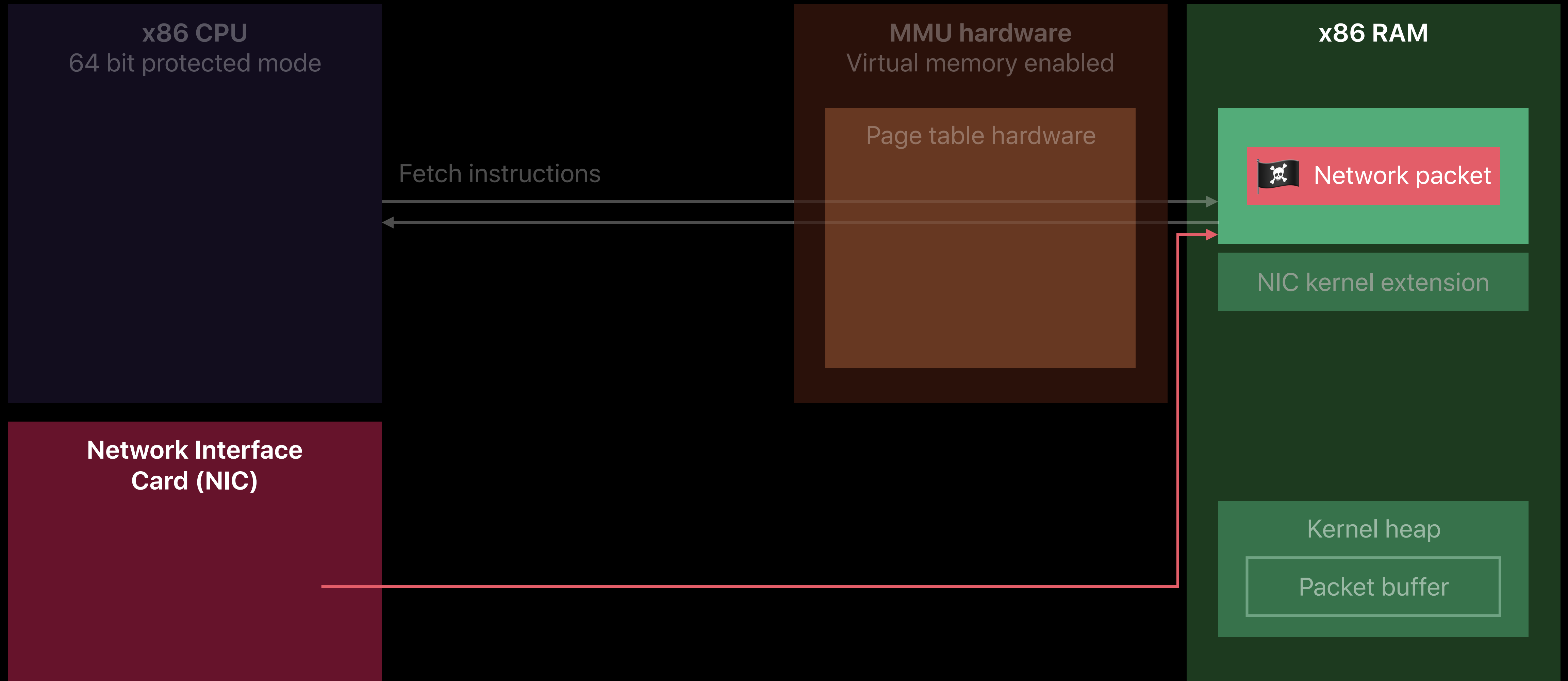
Unrestricted Direct Memory Access

UEFI macOS



Unrestricted Direct Memory Access

UEFI macOS



VT-d

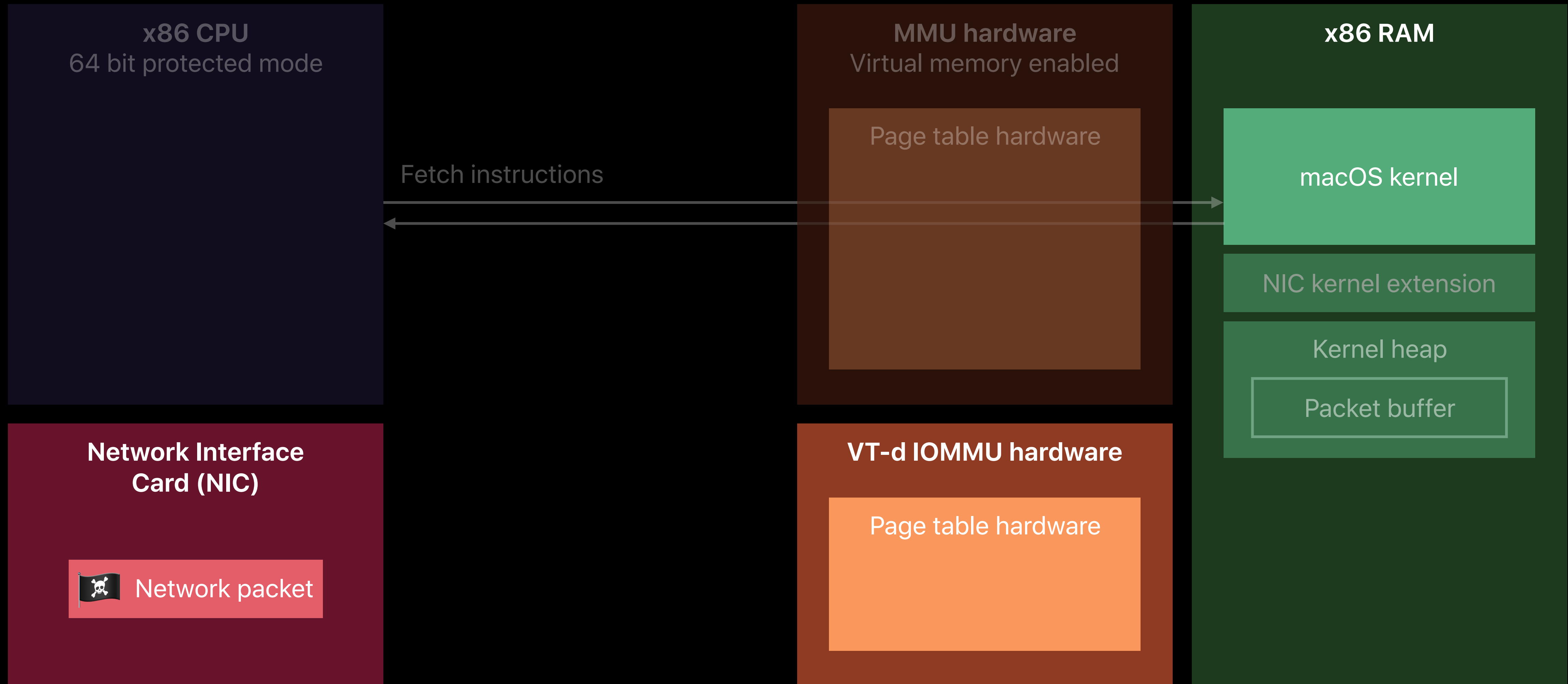
Intel Virtualization Technology for Directed I/O (VT-d) is a mechanism by which the host can place restrictions on DMA from peripherals

VT-d creates an I/O Memory Management Unit (IOMMU) to manage DMA

We've used VT-d to protect the kernel since OS X Mountain Lion in 2012

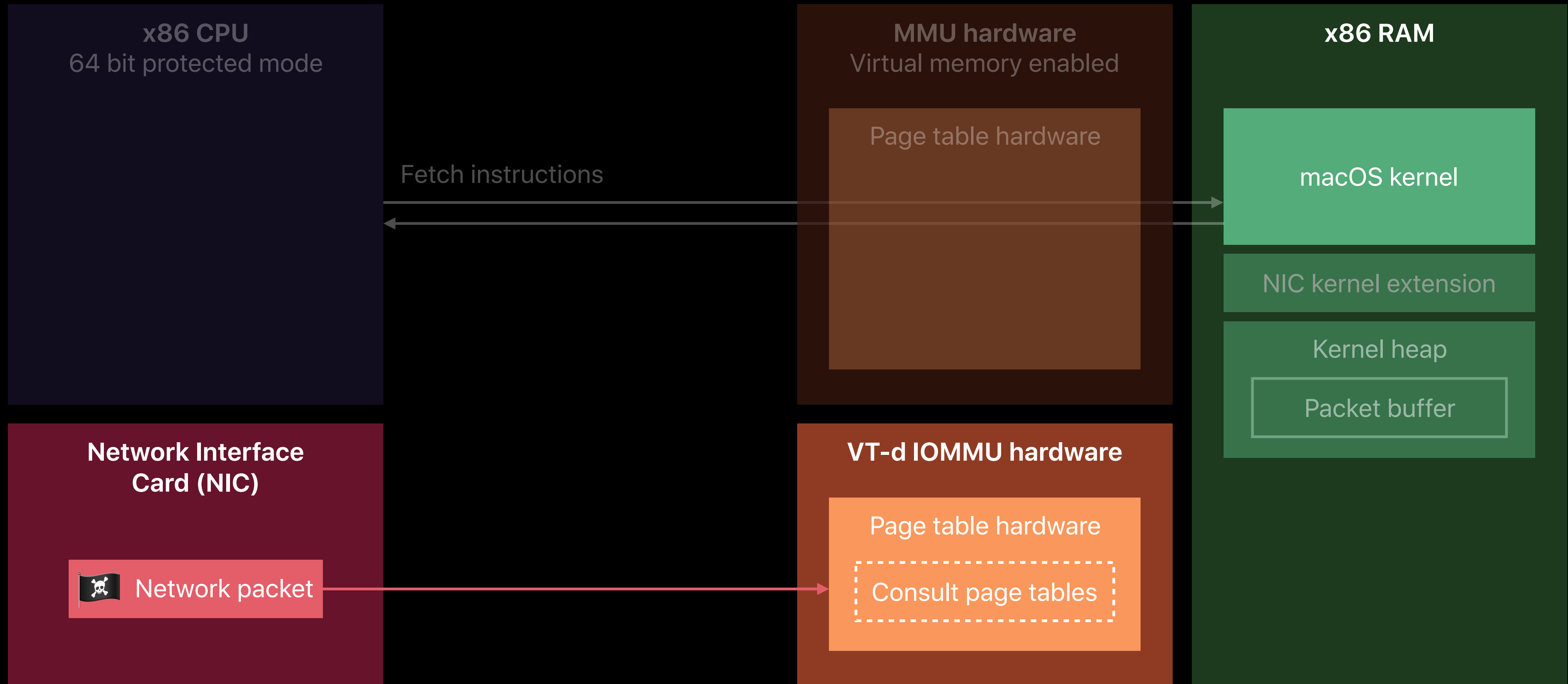
Direct Memory Access with VT-d

UEFI macOS



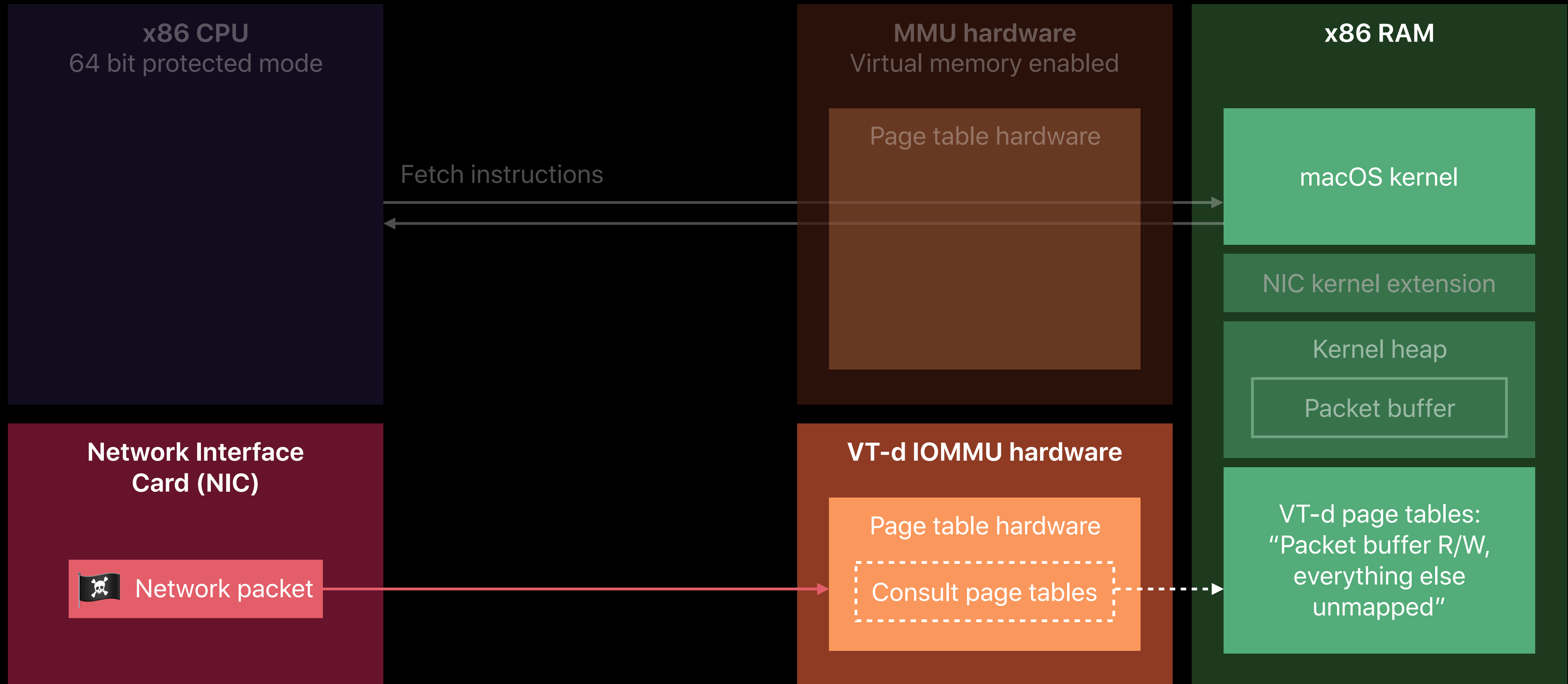
Direct Memory Access with VT-d

UEFI macOS



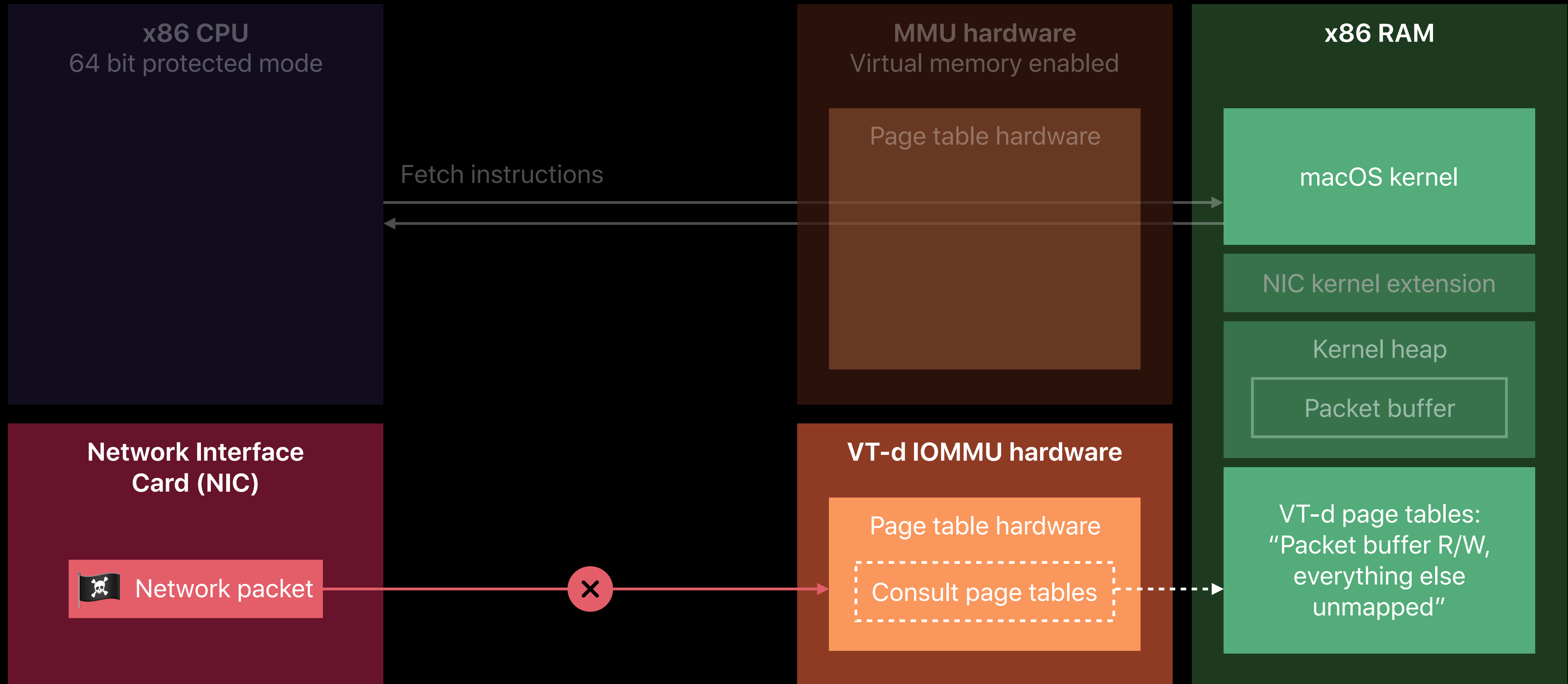
Direct Memory Access with VT-d

UEFI macOS



Direct Memory Access with VT-d

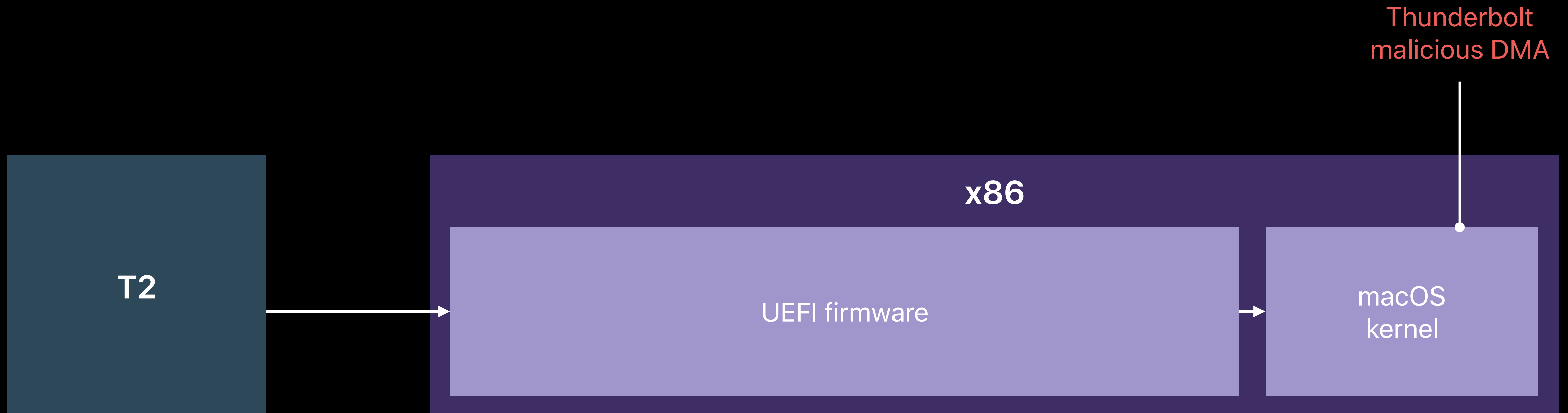
UEFI macOS



DMA Protection for Thunderbolt



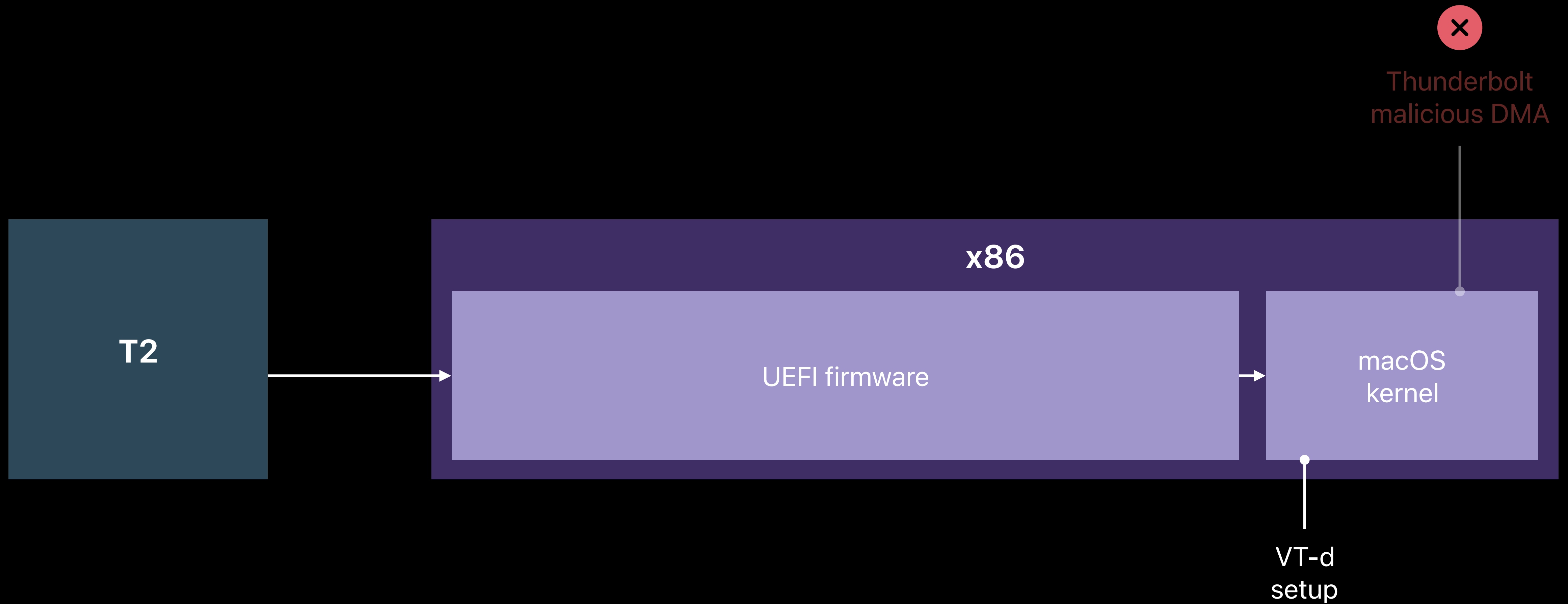
DMA Protection for Thunderbolt



DMA Protection for Thunderbolt



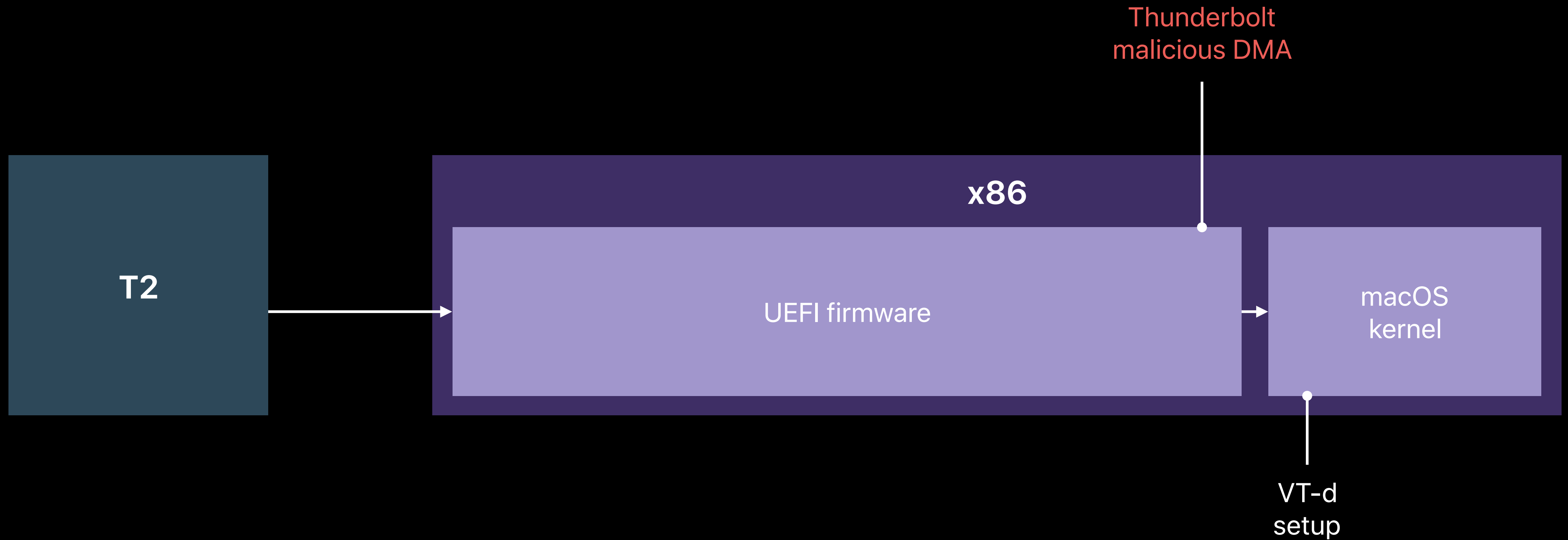
DMA Protection for Thunderbolt



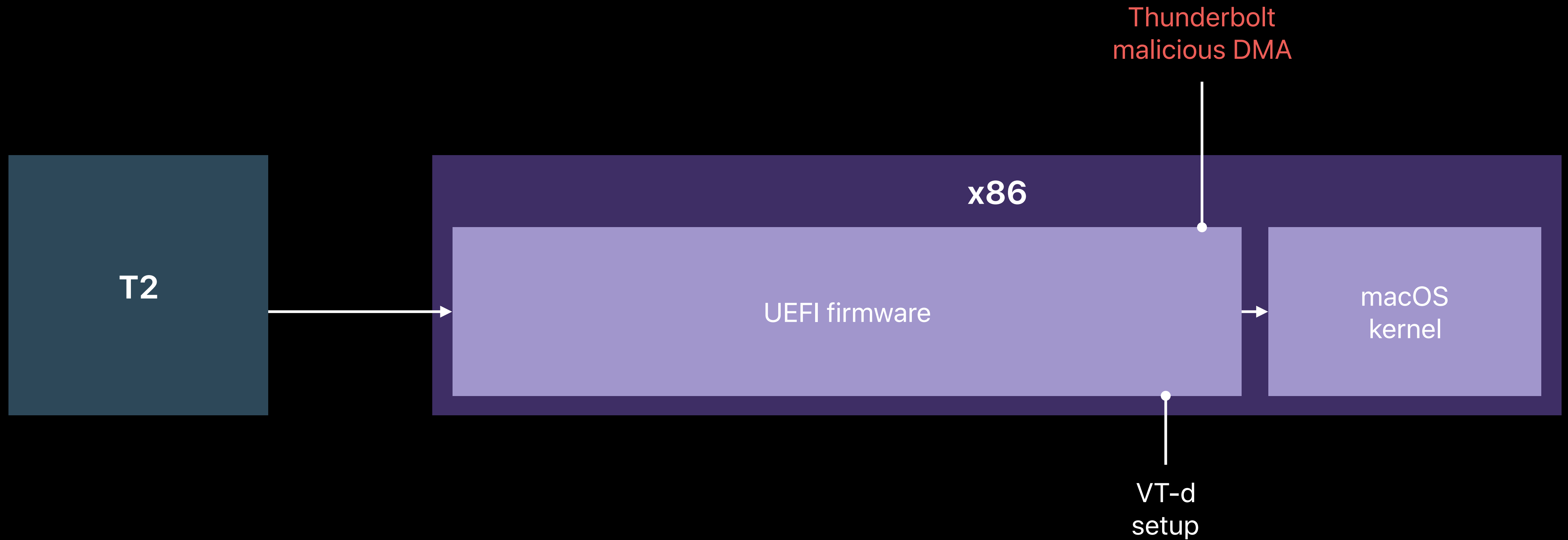
DMA Protection for Thunderbolt



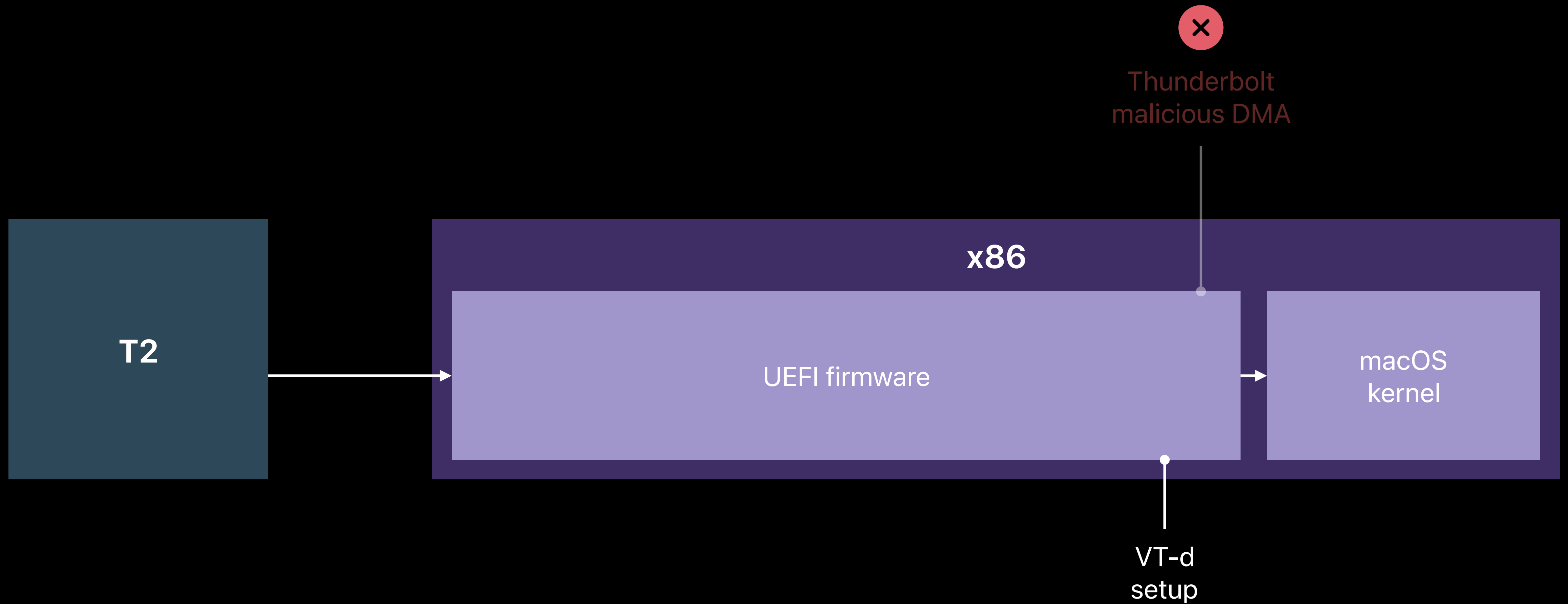
DMA Protection for Thunderbolt



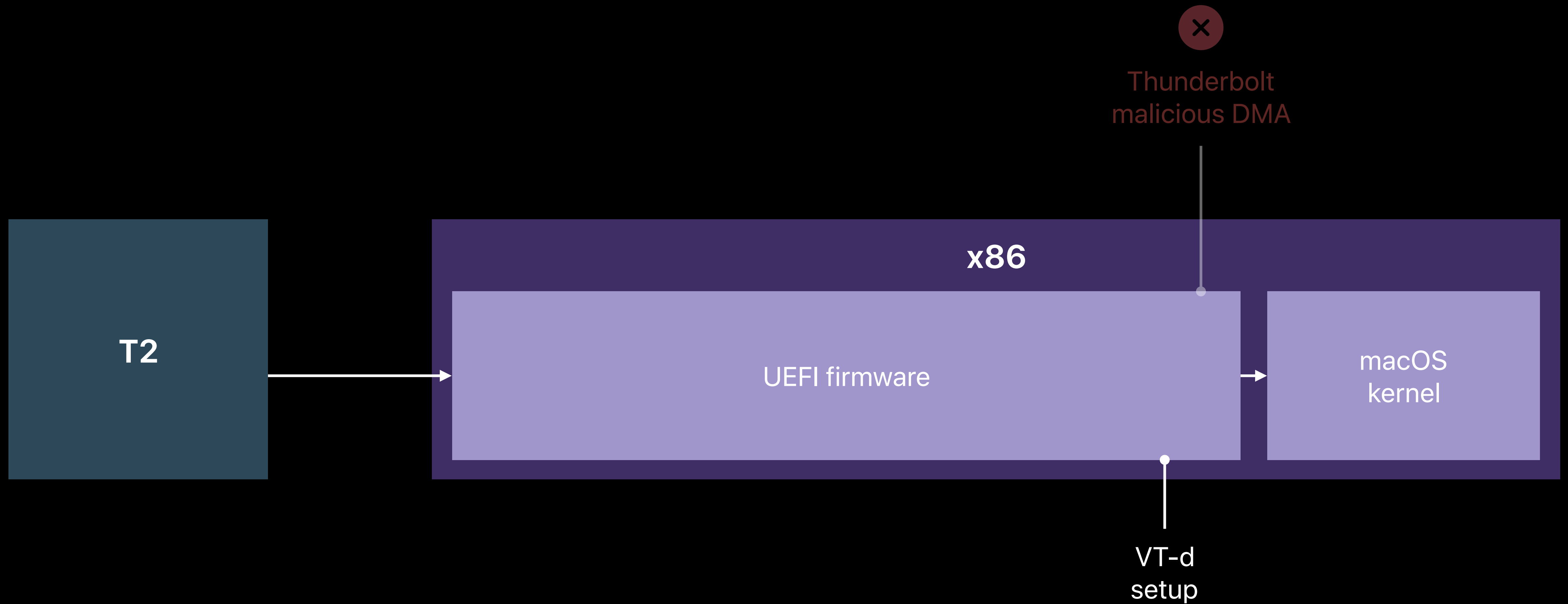
DMA Protection for Thunderbolt



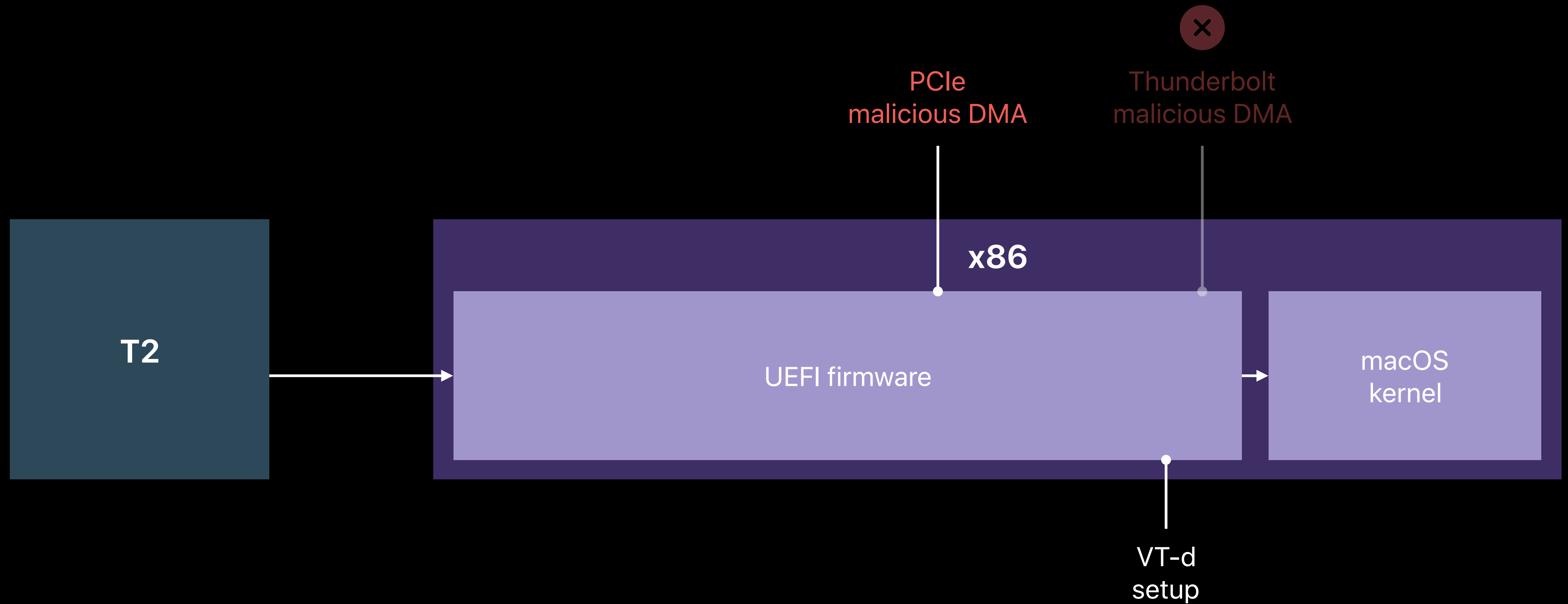
DMA Protection for Thunderbolt



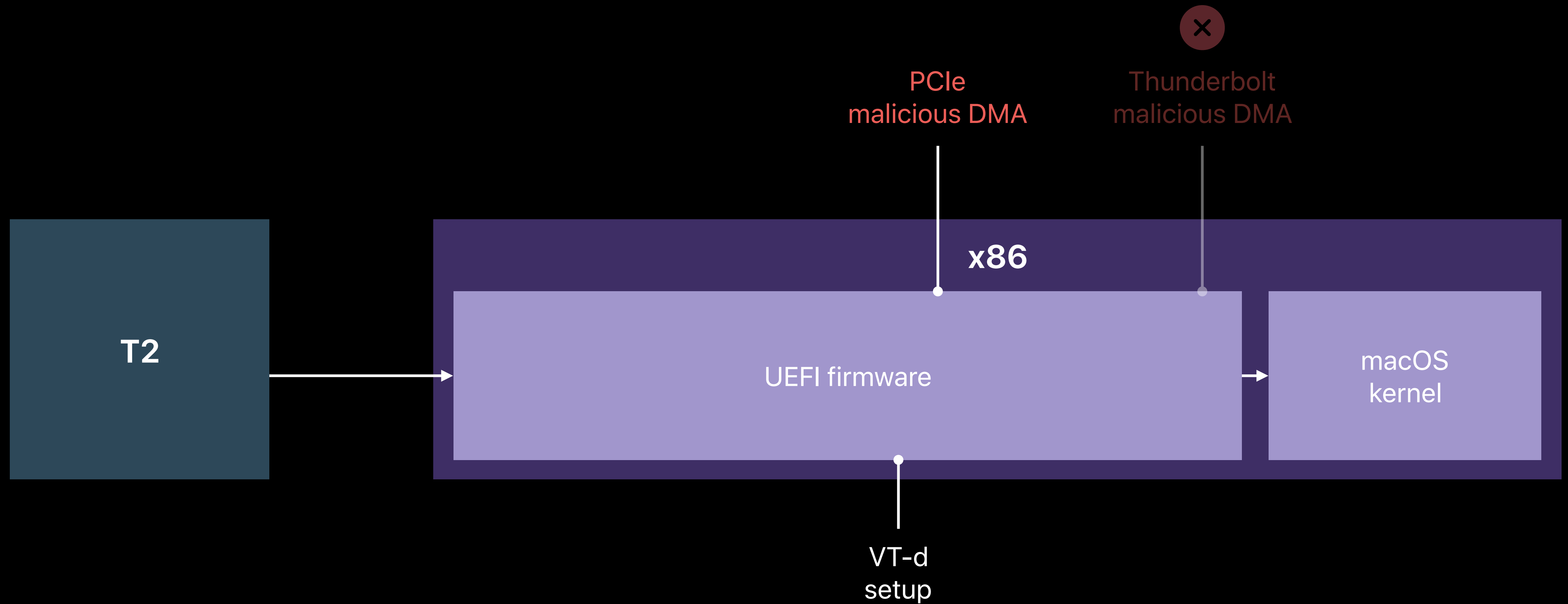
DMA Protection for PCIe



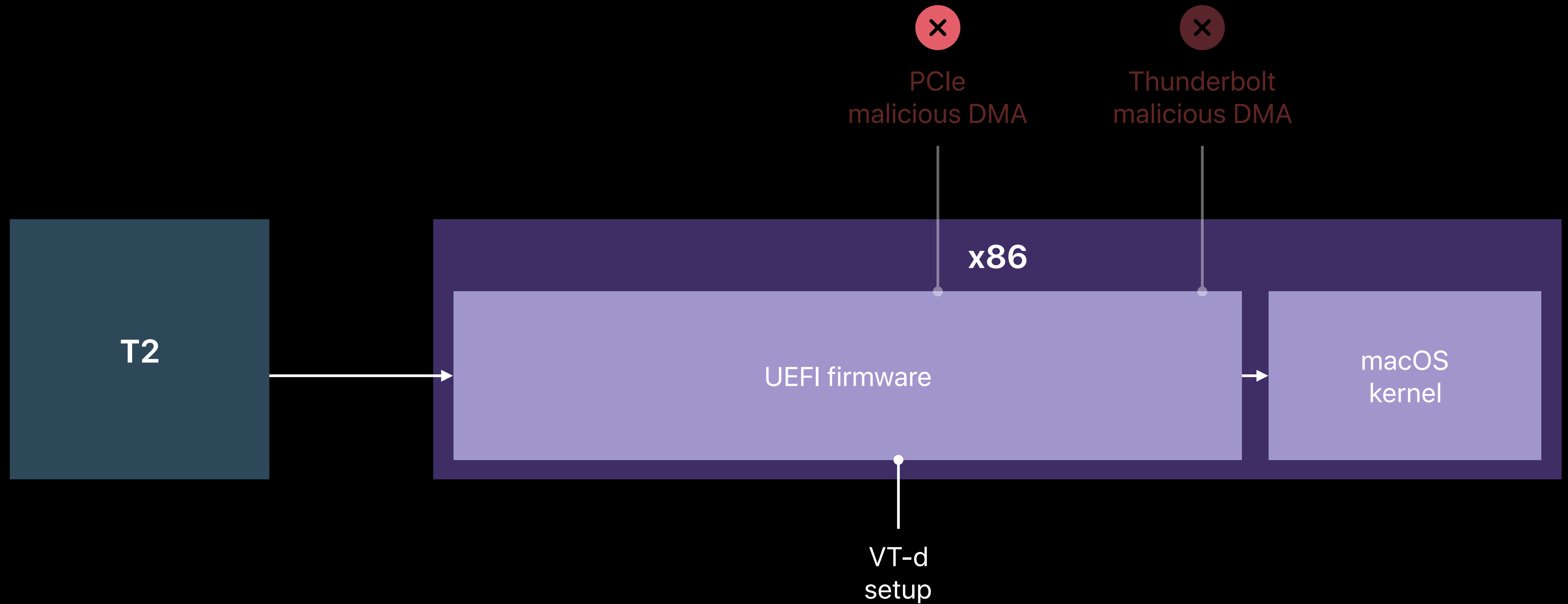
DMA Protection for PCIe



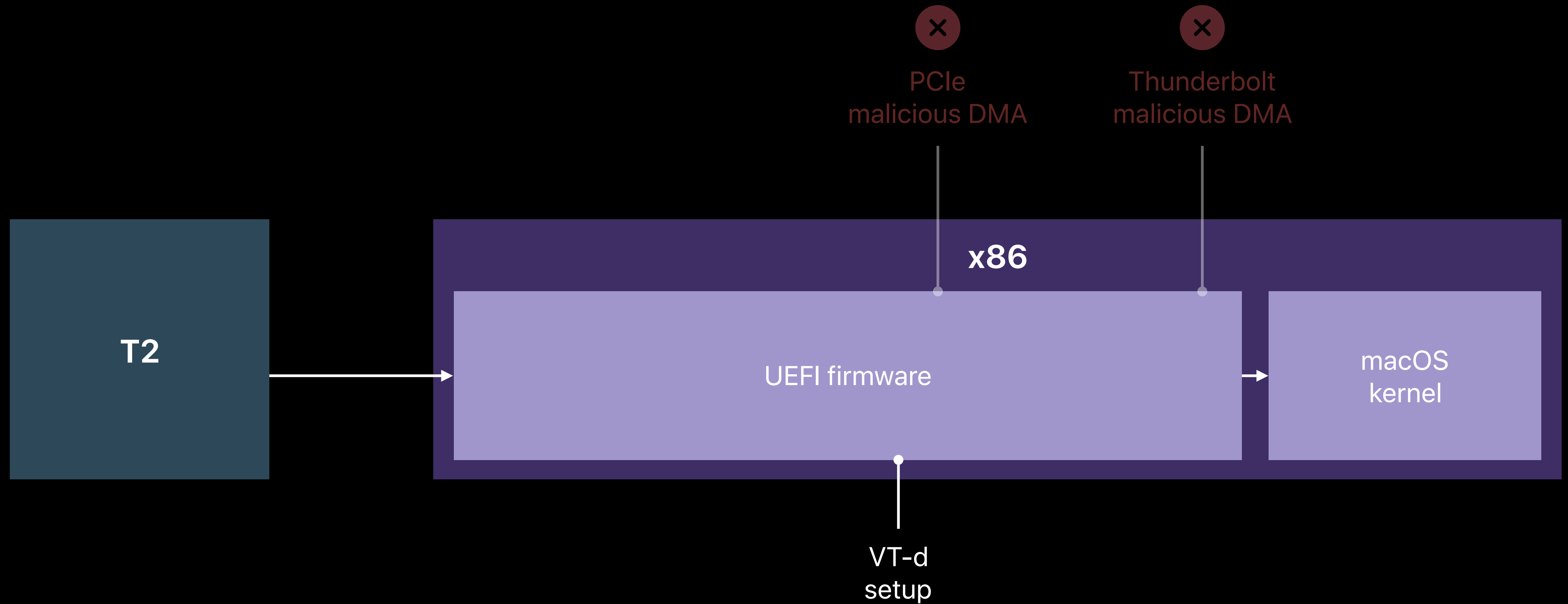
DMA Protection for PCIe



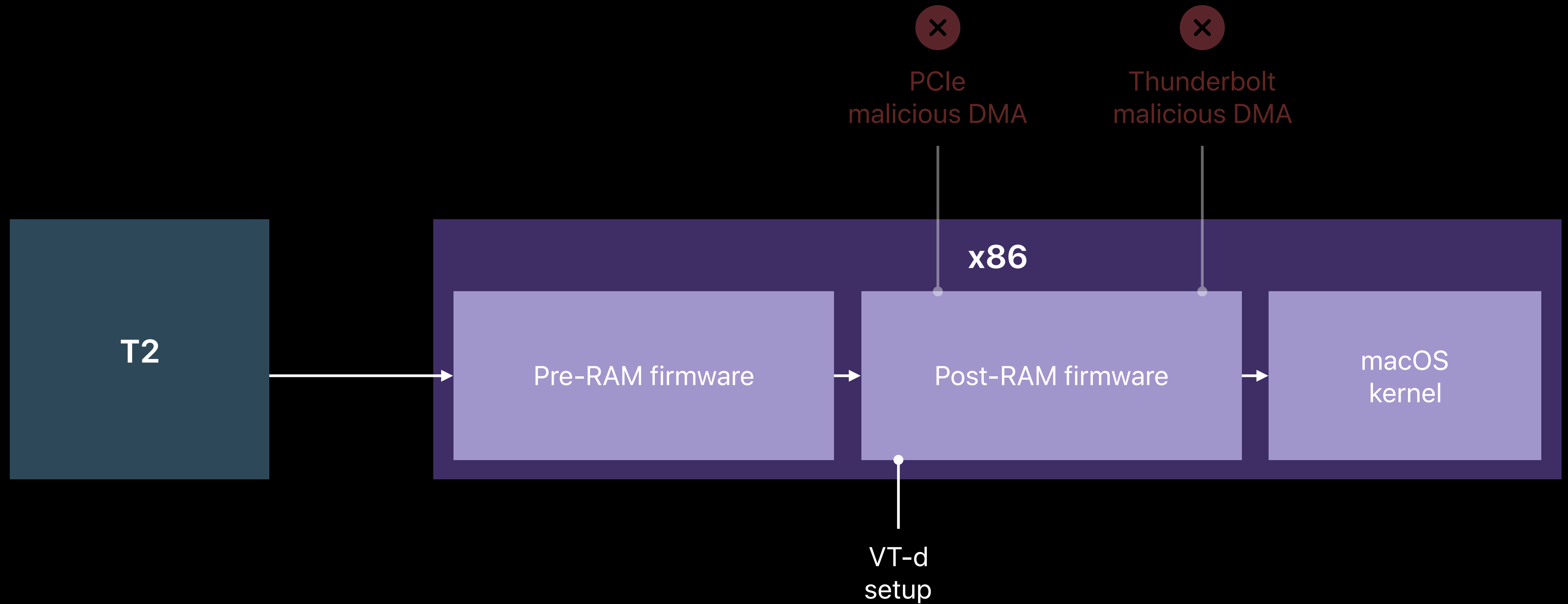
DMA Protection for PCIe



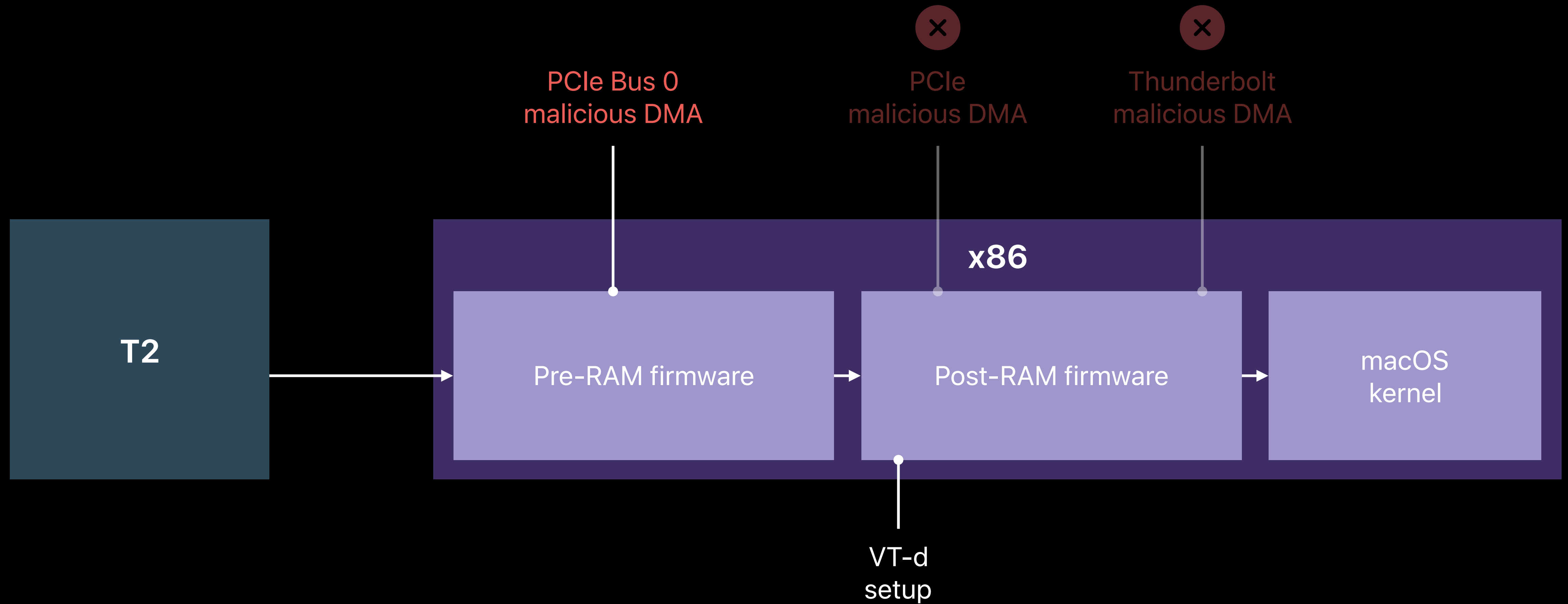
DMA Protection for PCIe Bus 0



DMA Protection for PCIe Bus 0

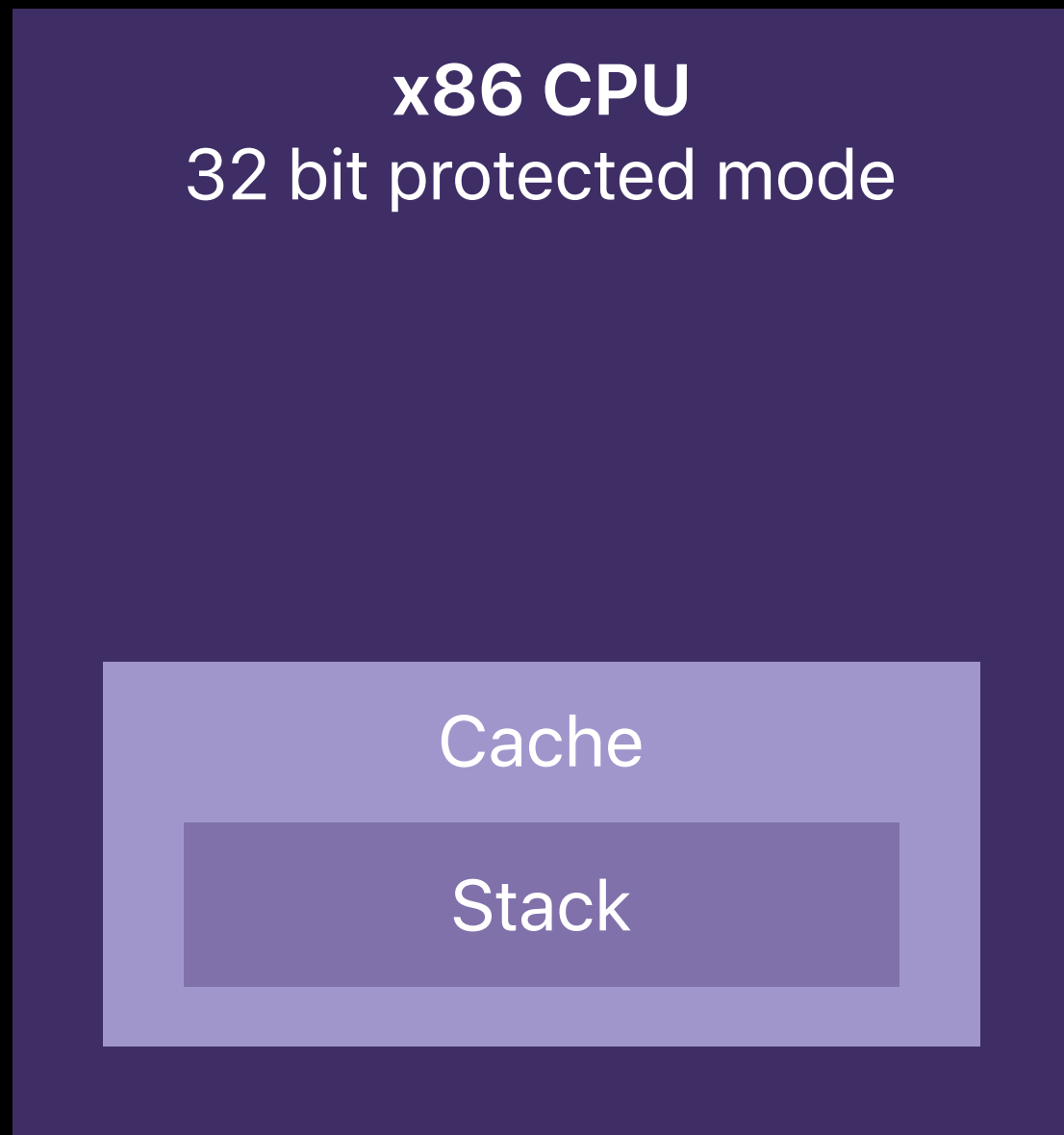


DMA Protection for PCIe Bus 0

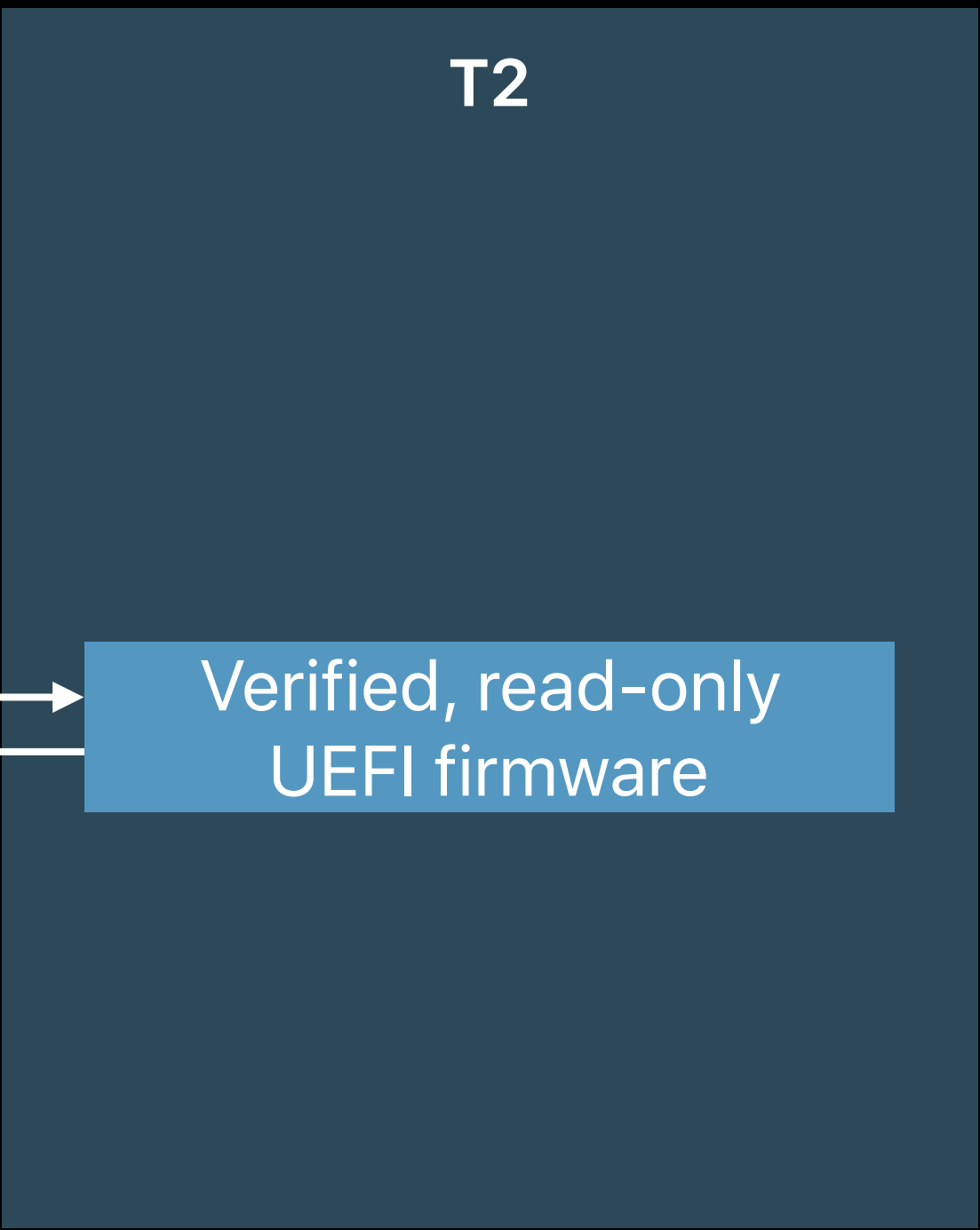
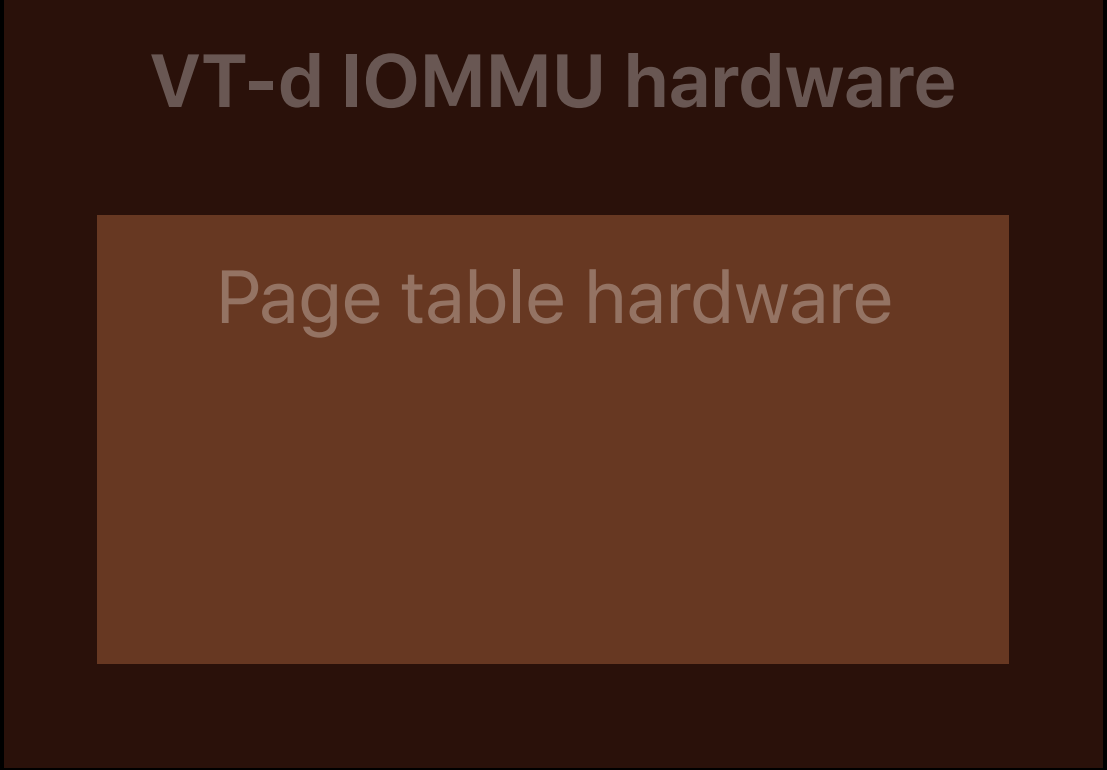
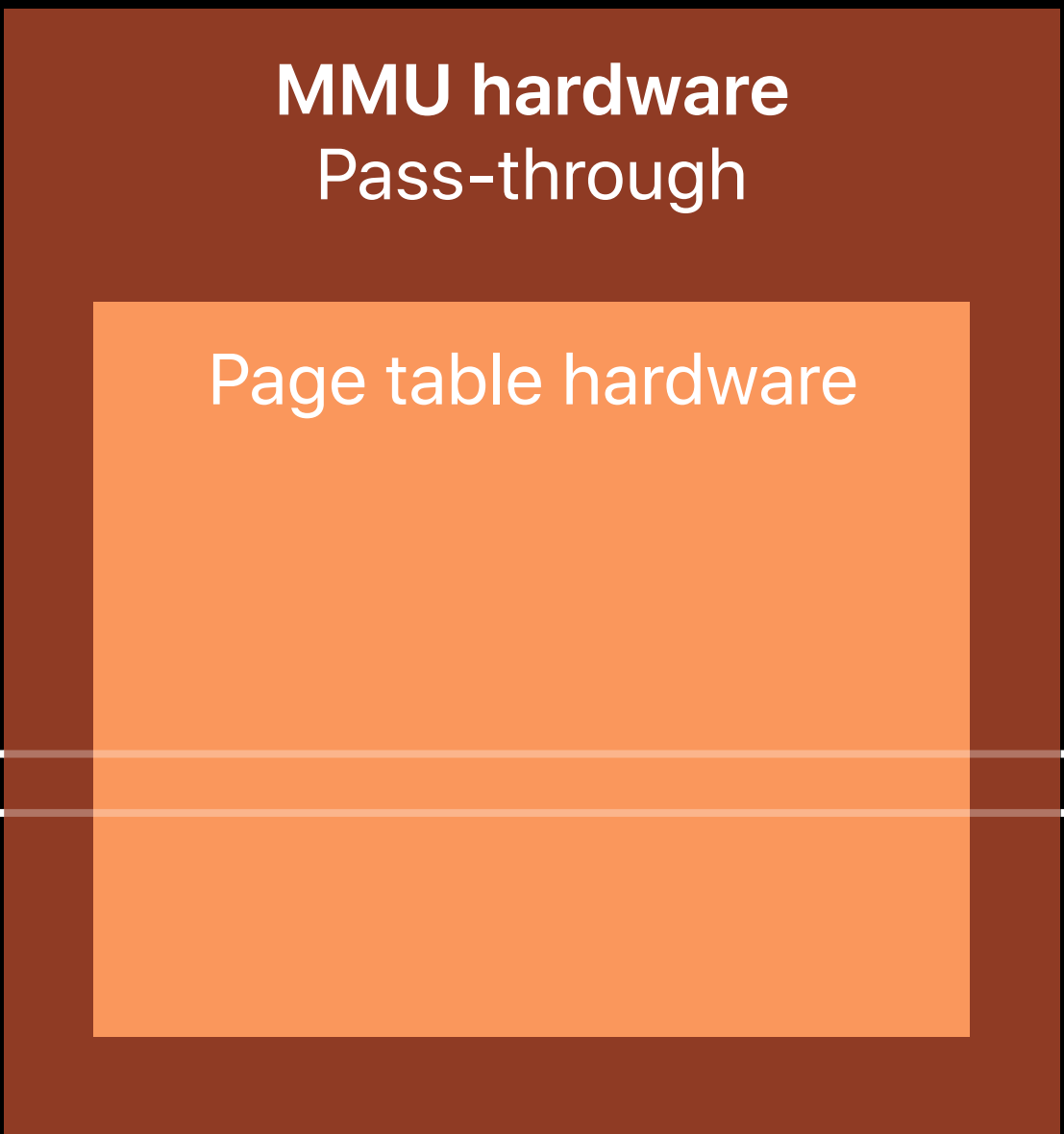


UEFI

macOS



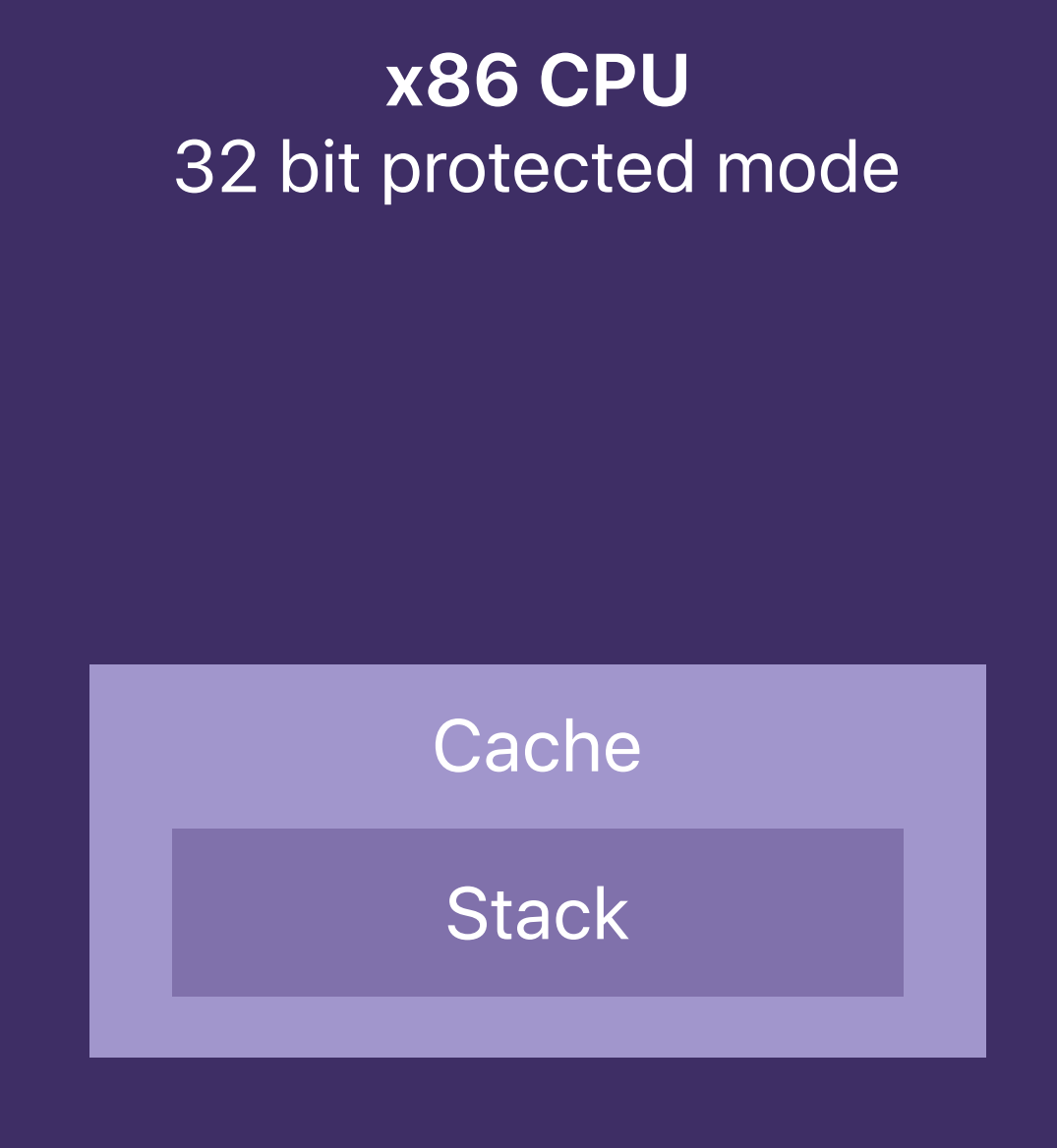
Fetch instructions



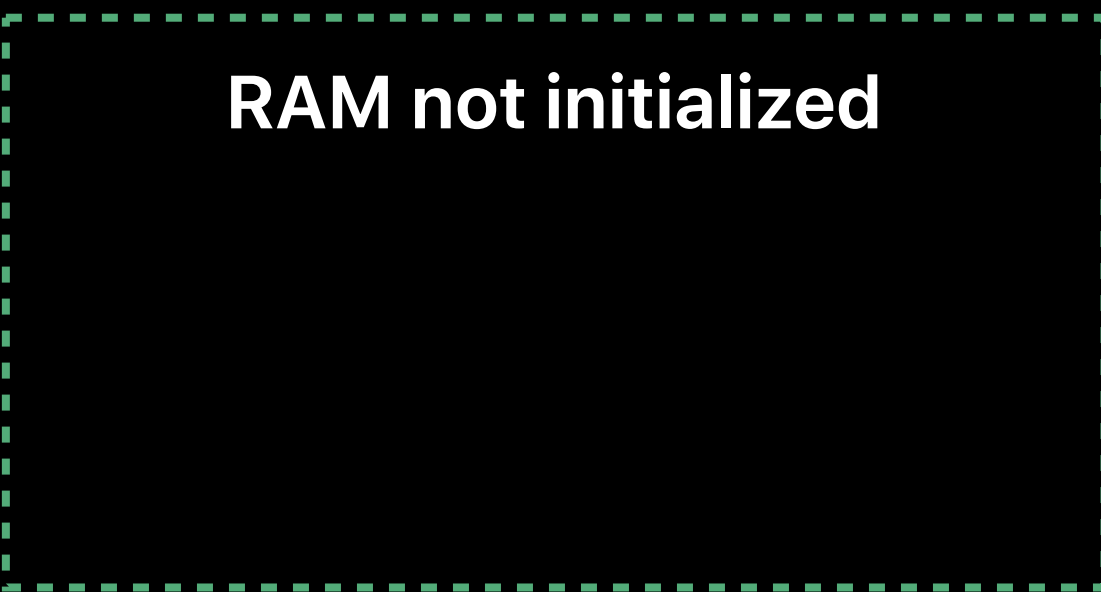
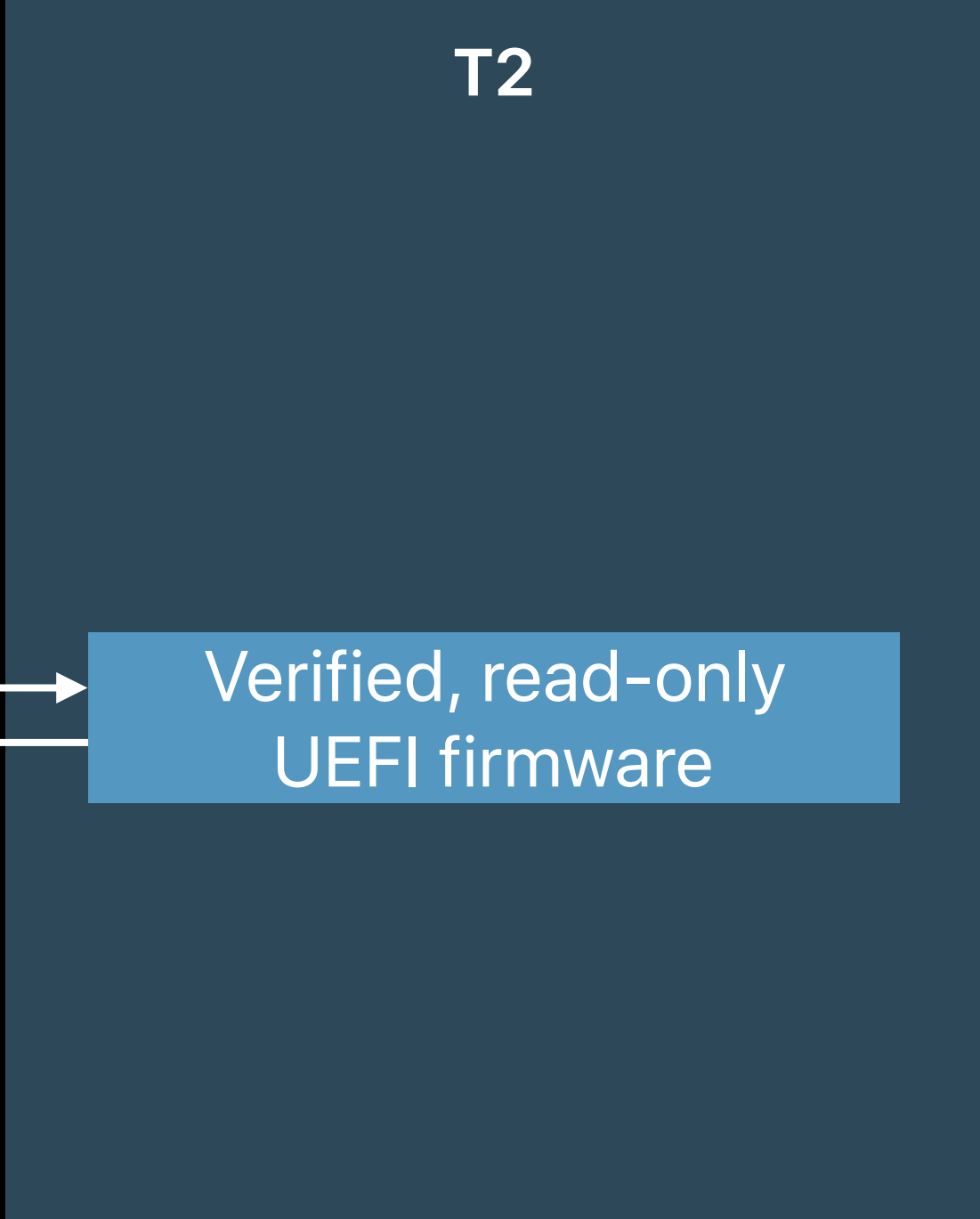
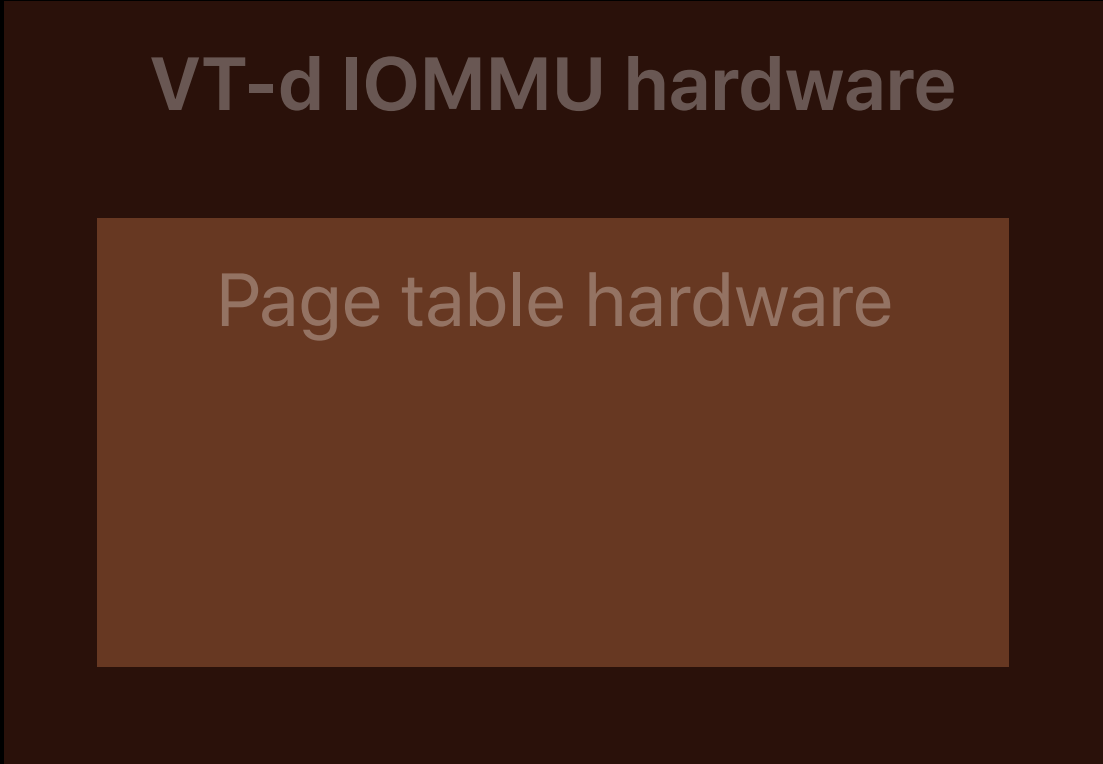
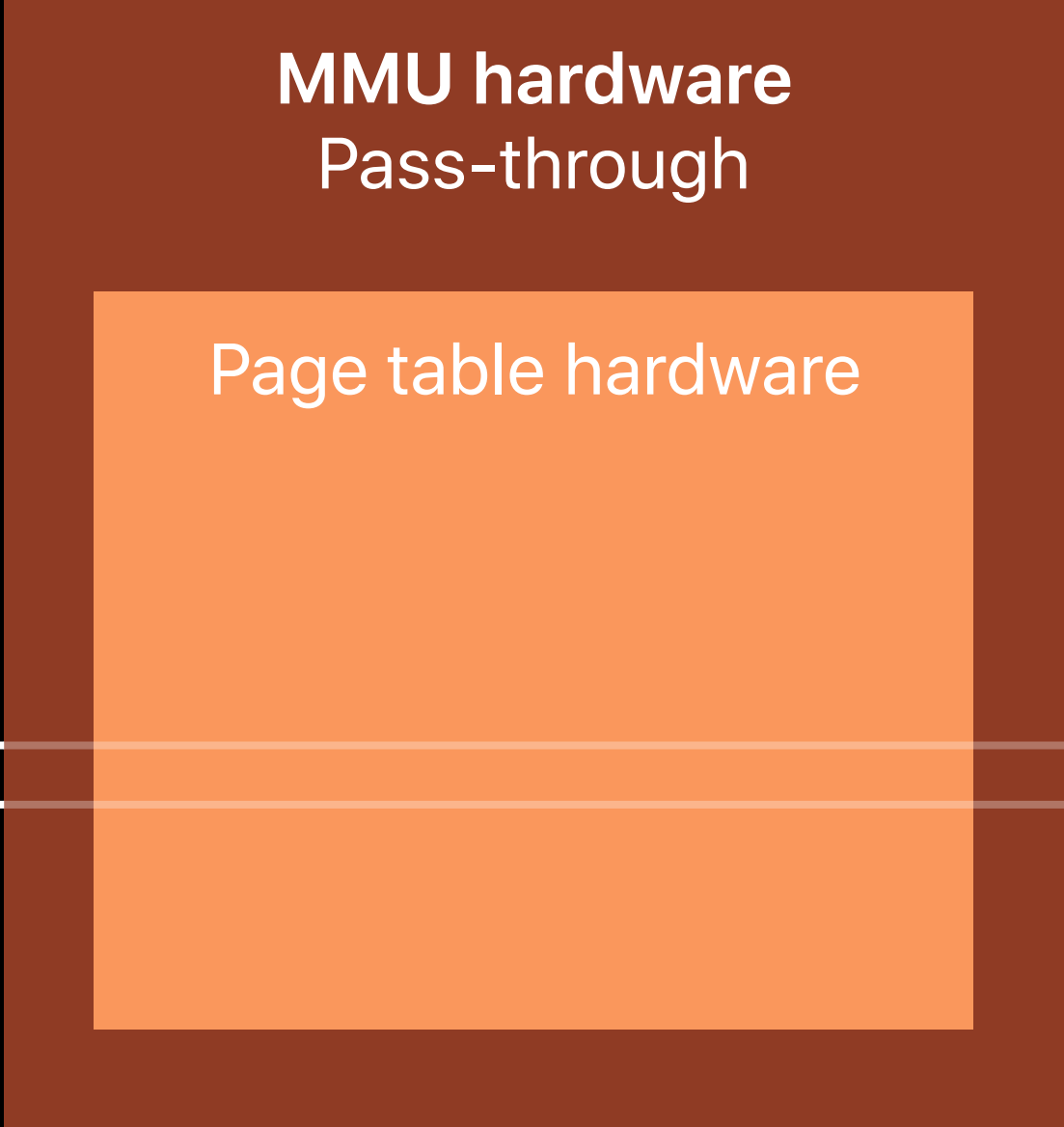
Pre-RAM

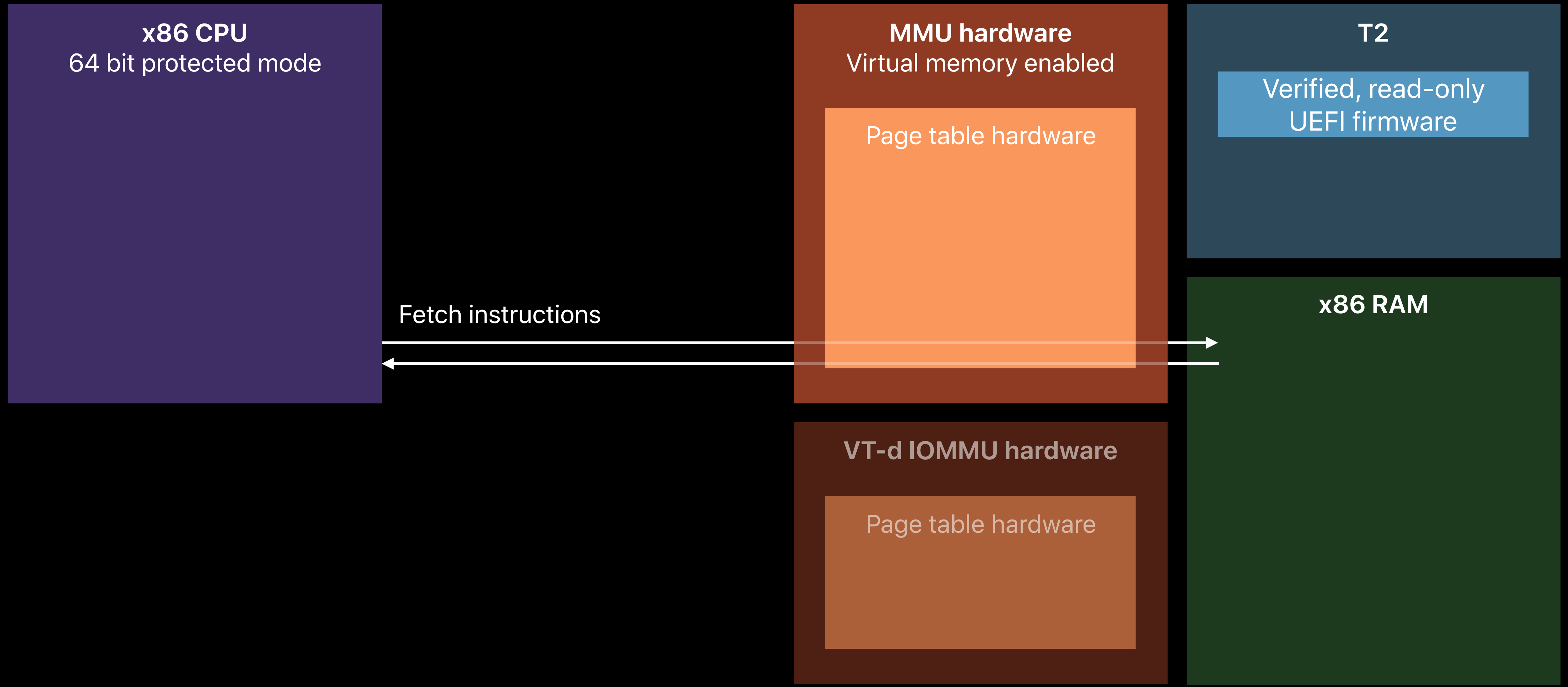
UEFI

macOS



Fetch instructions





x86 CPU
64 bit protected mode

MMU hardware
Virtual memory enabled

Page table hardware

T2

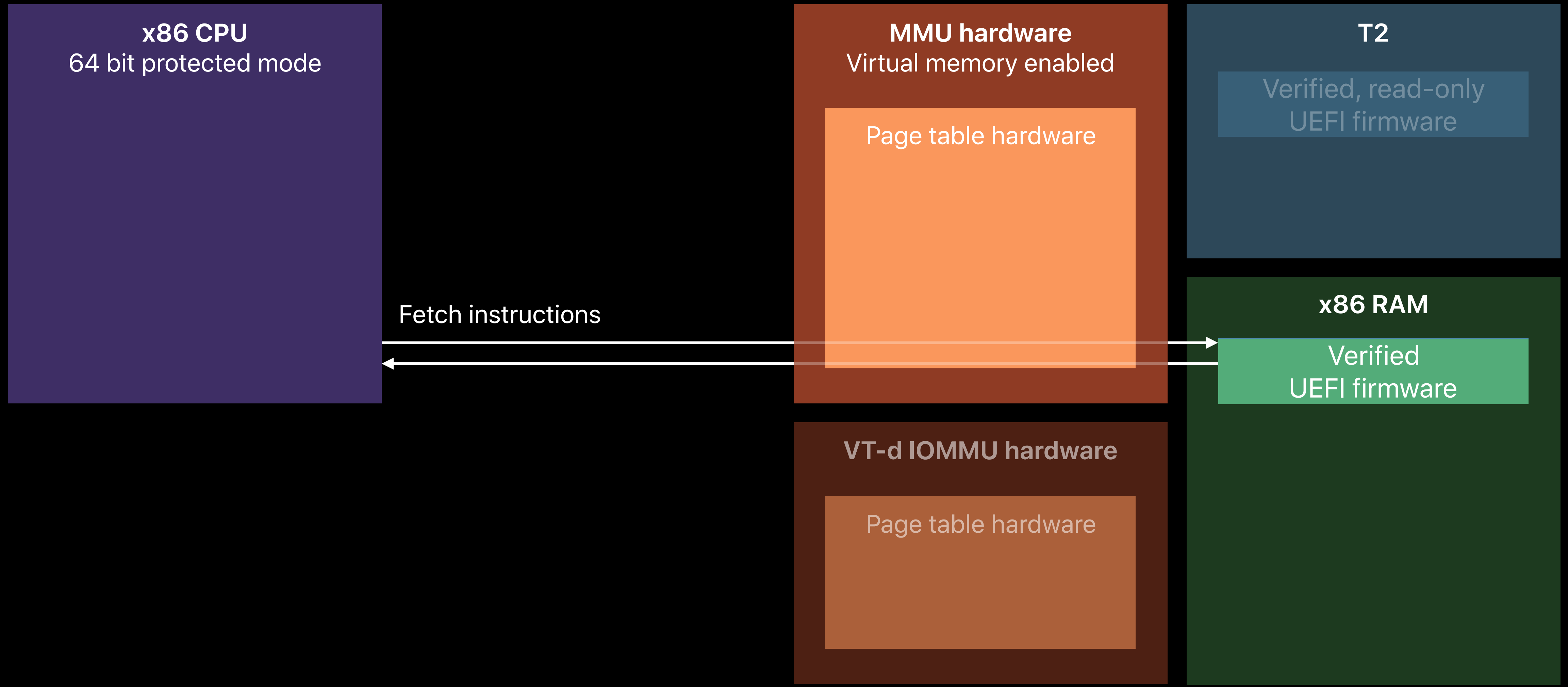
Verified, read-only
UEFI firmware

Fetch instructions

x86 RAM

VT-d IOMMU hardware

Page table hardware



Pre-RAM

UEFI

macOS

x86 CPU
64 bit protected mode

PCIe Bus 0 device

Malicious data

MMU hardware
Virtual memory enabled

Page table hardware

VT-d IOMMU hardware

Page table hardware

T2

Verified, read-only UEFI firmware

x86 RAM

Verified UEFI firmware

Fetch instructions



Pre-RAM

UEFI

macOS

x86 CPU
64 bit protected mode

MMU hardware
Virtual memory enabled

Page table hardware

T2

Verified, read-only UEFI firmware

PCIe Bus 0 device

Malicious data

VT-d IOMMU hardware

Page table hardware

x86 RAM

Verified UEFI firmware

Fetch instructions



Pre-RAM

UEFI

macOS

x86 CPU
64 bit protected mode

MMU hardware
Virtual memory enabled

Page table hardware

T2

Verified, read-only
UEFI firmware

PCIe Bus 0 device

VT-d IOMMU hardware

Page table hardware

x86 RAM

Malicious data

Fetch instructions



```
// This array contains the root and interrupt remapping tables. Each table is
// 4kB, and must be 4kB aligned as well. We can only guarantee the alignment by
// manually mapping our 2 4kB tables into this 12kB array. By initializing the
// array to all zeros, every bus is marked as not present, and no interrupts
// are allowed.
```

```
STATIC UINT8 mTables[TABLE_SIZE * 3] = {0};
```

```
STATIC
```

```
EFI_STATUS
```

```
EFIAPI
```

```
VTdBlockDMAForUnit(UINTN VTdBar)
```

```
{
```

```
    EFI_STATUS Status;
```

```
    VTD_ECAP_REG ExtCapabilities;
```

```
    UINT64 RootTable;
```

```
    UINT64 InterruptTable;
```

```
    CHECKED_VTD_CALL(CheckCapabilities(VTdBar));
```

```
    // ExtCap needed for IOTLB register offset
```

```
// This array contains the root and interrupt remapping tables. Each table is
// 4kB, and must be 4kB aligned as well. We can only guarantee the alignment by
// manually mapping our 2 4kB tables into this 12kB array. By initializing the
// array to all zeros, every bus is marked as not present, and no interrupts
// are allowed.
```

```
—————→ STATIC UINT8 mTables[TABLE_SIZE * 3] = {0};
```

```
STATIC
EFI_STATUS
EFIAPI
VTdBlockDMAForUnit(UINTN VTdBar)
{
    EFI_STATUS Status;
    VTD_ECAP_REG ExtCapabilities;
    UINT64 RootTable;
    UINT64 InterruptTable;

    CHECKED_VTD_CALL(CheckCapabilities(VTdBar));

    // ExtCap needed for IOTLB register offset
```



```
UINT64 InterruptTable;  
  
CHECKED_VTD_CALL(CheckCapabilities(VTdBar));
```

```
// ExtCap needed for IOTLB register offset
```

```
ExtCapabilities.Uint64 = MmioRead64(VTdBar + R_ECAP_REG);
```

```
—————→ RootTable = (UINT64)mTables;
```

**Use mTable
as RootTable**

```
// Align the root table to a 4kB boundary within the table buffer.
```

```
RootTable = (RootTable + TABLE_SIZE - 1) & ~(TABLE_SIZE - 1);
```

```
// Set deny-all root table
```

```
SetRootTable(VTdBar, RootTable);
```

```
// Put the interrupt remapping table right after the root table
```

```
InterruptTable = RootTable + TABLE_SIZE;
```

```
// Set deny-all interrupt table
```

```
SetInterruptRemapTable(VTdBar, InterruptTable);
```

```
RootTable = (UINT64)mTables;
```

```
// Align the root table to a 4kB boundary within the table buffer.
```

```
RootTable = (RootTable + TABLE_SIZE - 1) & ~(TABLE_SIZE - 1);
```

```
// Set deny-all root table
```

```
SetRootTable(VTdBar, RootTable);
```

**Use RootTable
for DMA VT-d**

```
// Put the interrupt remapping table right after the root table
```

```
InterruptTable = RootTable + TABLE_SIZE;
```

```
// Set deny-all interrupt table
```

```
SetInterruptRemapTable(VTdBar, InterruptTable);
```

```
// Set deny-all table
SetRootTable(VTdBar, RootTable);

// Put the interrupt remapping table right after the root table
InterruptTable = RootTable + TABLE_SIZE;

// Set deny-all interrupt table
SetInterruptRemapTable(VTdBar, InterruptTable);
```

—————→
**Same for MSI VT-d
interrupts**

Structure

Name	Action	Type	Subtype	Text
▶ A3C7A8BA-094A-47C6-9F66-F6DE5E42A6E7		File	PEI module	
Pad-file		File	Pad	
▶ ApplePlatformInfoDatabaseDxe		File	PEI module	
▶ C24A946F-8BC6-412B-9ACE-307E08E68125		File	PEI module	
Pad-file		File	Pad	
▶ D072670B-DC2C-4768-8102-99B4A9EF5EDC		File	PEI module	
Pad-file		File	Pad	
▶ 3FB1A55F-DDEF-42D9-8FAF-891039769F8D		File	PEI module	
Pad-file		File	Pad	
▶ 636826D4-BFA3-4D8B-B3A8-9535384932BE		File	PEI module	
▼ A1F39391-B841-4C3E-A458-71E312DD6CB9		File	PEI module	
PEI dependency section		Section	PEI dependency	
Raw section		Section	Raw	
TE image section		Section	TE image	
Pad-file		File	Pad	
▶ 53EF228E-BF8B-489B-818A-57034BA88F60		File	PEI module	
Pad-file		File	Pad	
▶ PlatformInitPreMem		File	PEI module	
▶ PlatformInit		File	PEI module	
Pad-file		File	Pad	
▶ PiSmmCommunicationPei		File	PEI module	
Pad-file		File	Pad	
▶ S3Resume2Pei		File	PEI module	
Pad-file		File	Pad	
▶ SiInitPreMem		File	PEI module	
▶ CpuMpPei		File	PEI module	
▶ SiInit		File	PEI module	
Pad-file		File	Pad	
▶ 63D5F28B-0F76-40A5-A54E-D235DDFDD1C2		File	PEI module	
Pad-file		File	Pad	
▶ 308FC263-7672-45C8-8096-6D0288E94EEE		File	PEI module	
▶ ADA7DBB8-2E6F-4FF6-8963-7CD5C0040C52		File	PEI module	
▶ DB0116EE-5135-4920-AB3C-A289FDAAB583		File	PEI module	
▶ F25EC99D-564F-473B-BFFB-671BE412886E		File	PEI module	
▶ 75135AB5-D466-452B-A6D9-028F9FA60762		File	PEI module	
Pad-file		File	Pad	
▶ DxeIplPei		File	PEI module	
▶ EfiBiosIdGuid		File	Freeform	
▶ AppleRomInformation		File	Freeform	
Volume free space		Free space		
▶ 04ADEEAD-61FF-4D31-B6BA-64F8BF901F5A		Volume	FFSv2	
▶ 04ADEEAD-61FF-4D31-B6BA-64F8BF901F5A		Volume	FFSv2	

Information

Fixed: No
 Base: E0EFACh
 Header address: FFE0EFACh
 Data address: FFE0EFB0h
 Offset: 44h
 Type: 12h
 Full size: 4634h (17972)
 Header size: 4h (4)
 Body size: 4630h (17968)
 Signature: 5A56h
 Machine type: x86
 Number of sections: 4
 Subsystem: 0Bh
 Stripped size: 178h (376)
 Base of code: 240h
 Address of entry point: 325h
 Image base: FFE0EE60h
 Adjusted image base: FFE0EFB0h

Hex view: TE image section

15A0	81	E5	DF	4B	B8	13	ED	59	11	64	23	DC	00	00	00	00	ã	ß	K	,	í	Y	.	d	#	Ü
15B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
15C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
15D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
15E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
15F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
1600	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
1610	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
1620	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
1630	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
1640	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
1650	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
1660	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
1670	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
1680	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
1690	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
16A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
16B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
16C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
16D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
16E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
16F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
1700	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
1710	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
1720	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
1730	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
1740	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
1750	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
1760	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
1770	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
1780	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
1790	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
17A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
17B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
17C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
17D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
17E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
17F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										
1800	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00										

Address	Size	Version	Checksum	Type
---------	------	---------	----------	------

Pre-RAM

UEFI

macOS

x86 CPU
64 bit protected mode

MMU hardware
Virtual memory enabled

Page table hardware

T2

Verified, read-only
UEFI firmware

VT-d IOMMU hardware

Page table hardware

x86 RAM

Fetch instructions



Pre-RAM

UEFI

macOS

x86 CPU
64 bit protected mode

MMU hardware
Virtual memory enabled

Page table hardware

T2

Verified, read-only
UEFI firmware

Bus0 VT-d page tables
"deny all"

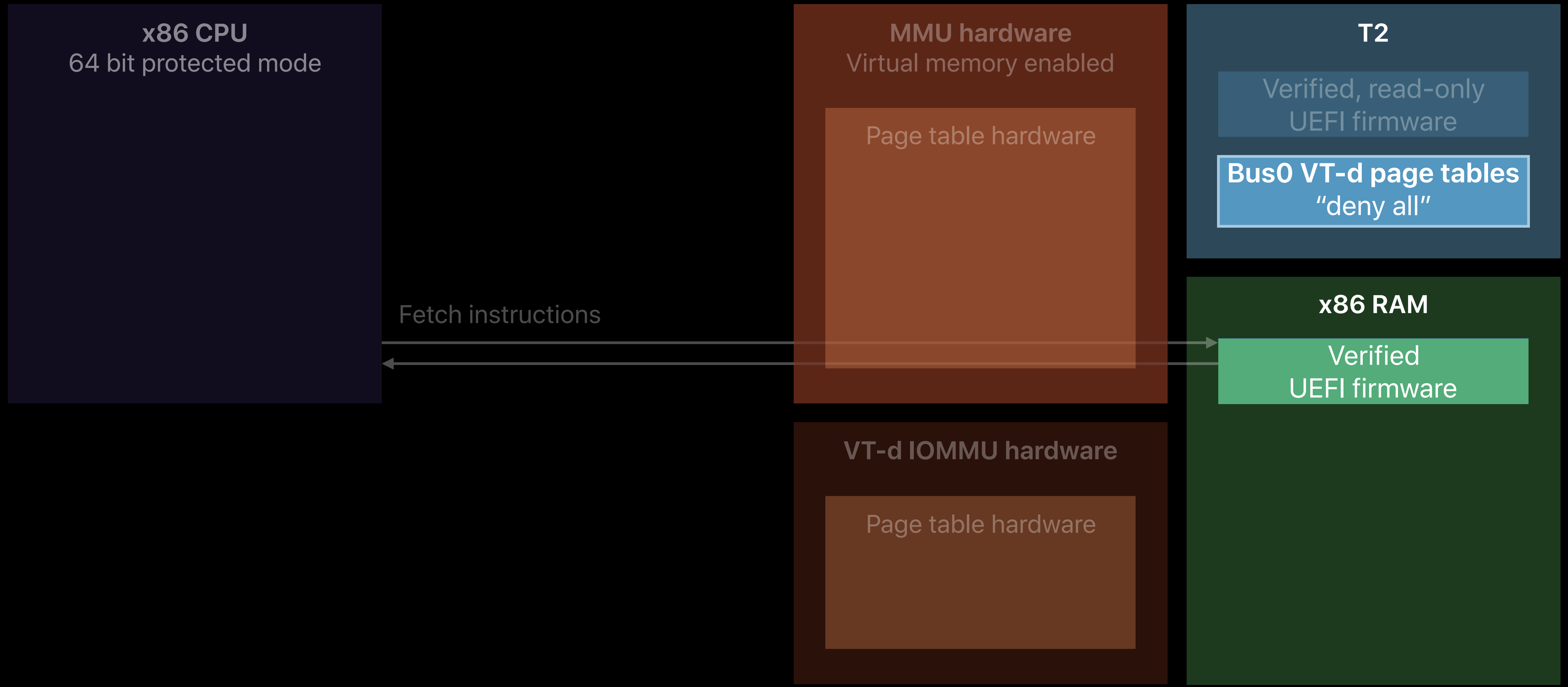
VT-d IOMMU hardware

Page table hardware

x86 RAM

Fetch instructions





Pre-RAM

UEFI

macOS

x86 CPU
64 bit protected mode

MMU hardware
Virtual memory enabled

Page table hardware

T2

Verified, read-only UEFI firmware

Bus0 VT-d page tables
"deny all"

PCIe Bus 0 device

Malicious data

VT-d IOMMU hardware

Page table hardware

x86 RAM

Verified UEFI firmware

Fetch instructions



Pre-RAM

UEFI

macOS

x86 CPU
64 bit protected mode

MMU hardware
Virtual memory enabled

Page table hardware

T2

Verified, read-only UEFI firmware

Bus0 VT-d page tables
"deny all"

PCIe Bus 0 device

Malicious data

VT-d IOMMU hardware

Page table hardware

Consult page tables

x86 RAM

Verified UEFI firmware

Fetch instructions



Pre-RAM

UEFI

macOS

x86 CPU
64 bit protected mode

MMU hardware
Virtual memory enabled

Page table hardware

T2

Verified, read-only UEFI firmware

Bus0 VT-d page tables
"deny all"

x86 RAM

Verified UEFI firmware

PCIe Bus 0 device

Malicious data

VT-d IOMMU hardware

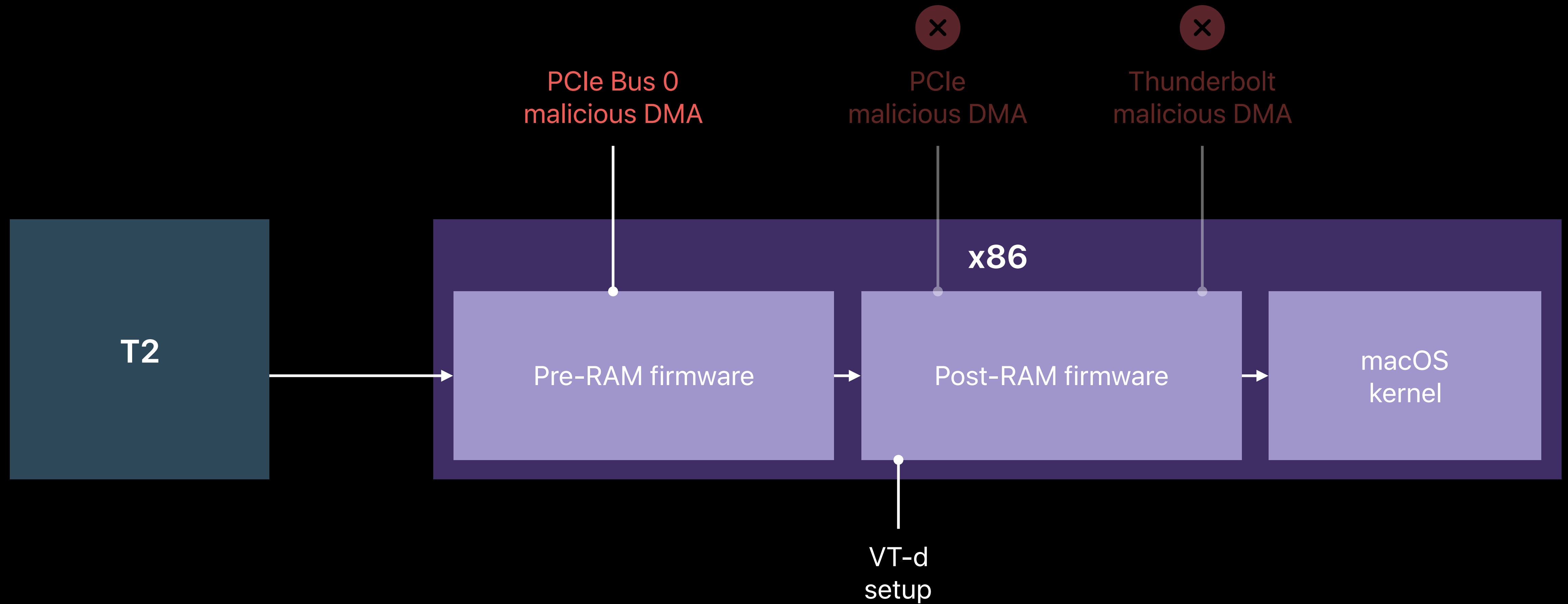
Page table hardware

Consult page tables

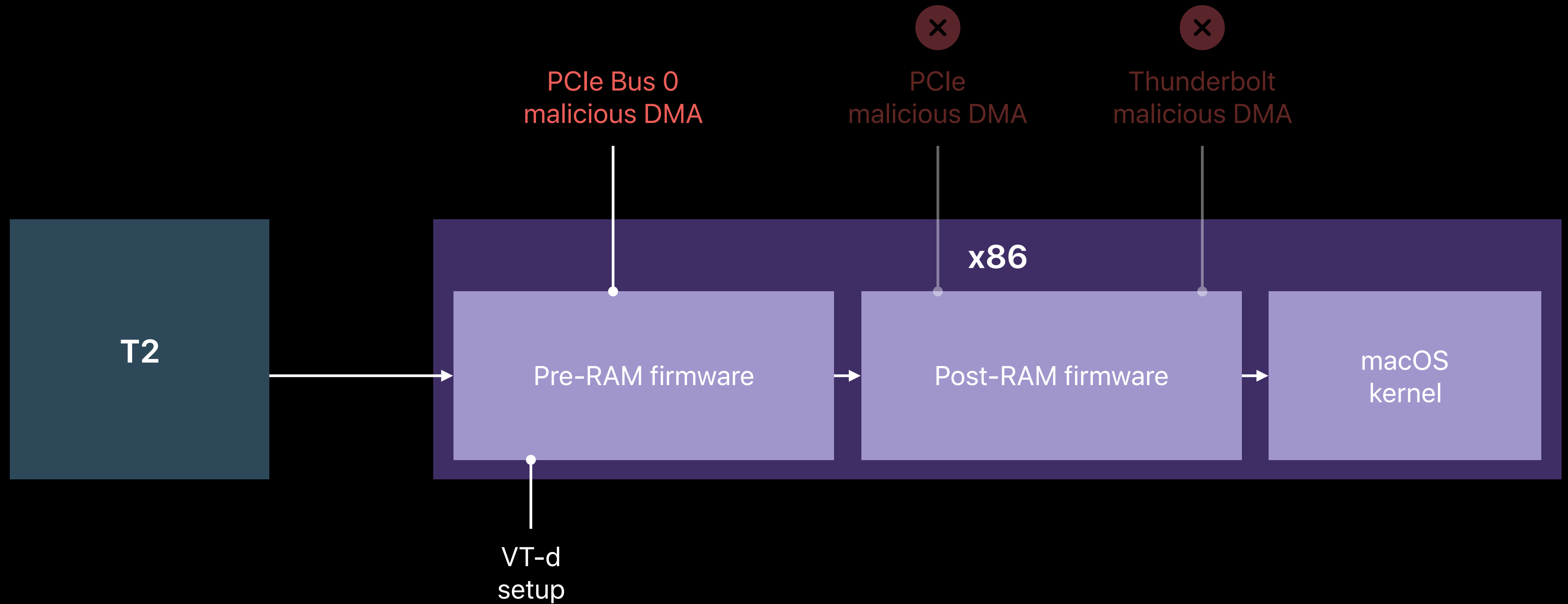
Fetch instructions



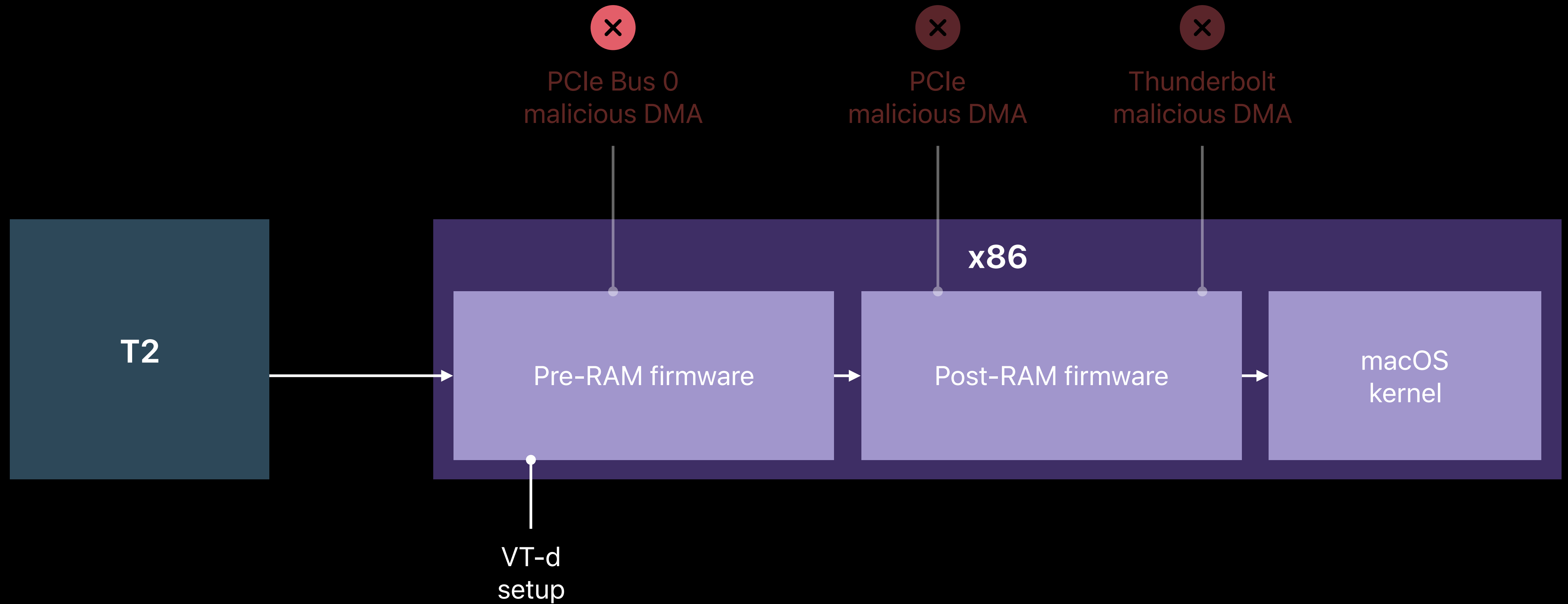
DMA Protection for PCIe Bus 0



DMA Protection for PCIe Bus 0



DMA Protection for PCIe Bus 0



PCIe Option ROMs

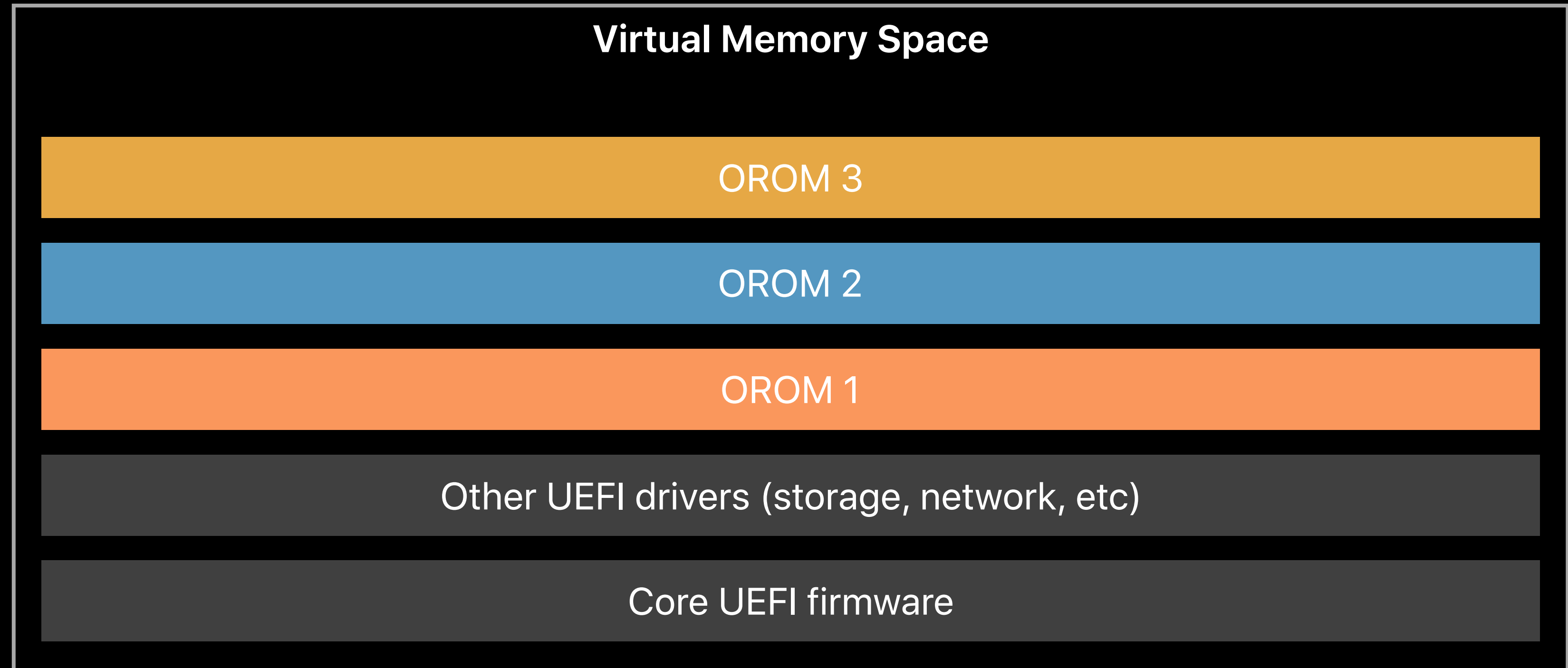
PCIe Option ROMs

Device drivers which PCIe devices supply to UEFI

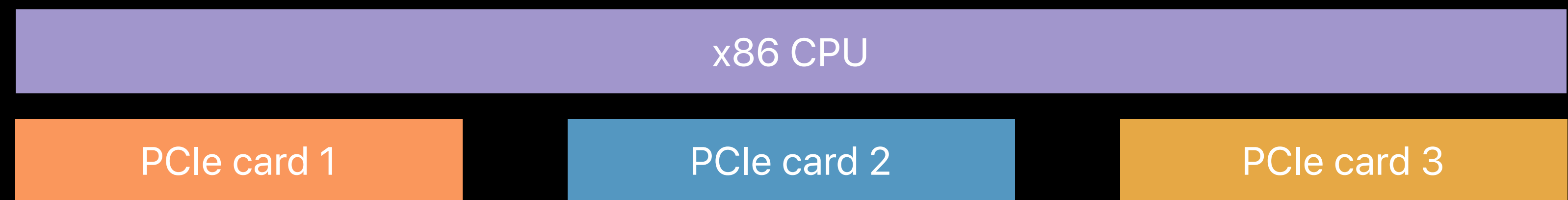
UEFI firmware, including OROMs, mostly all run at the same x86 privilege level: Ring 0

All code loaded after OROMs, including the booter and kernel, is vulnerable to overwrite

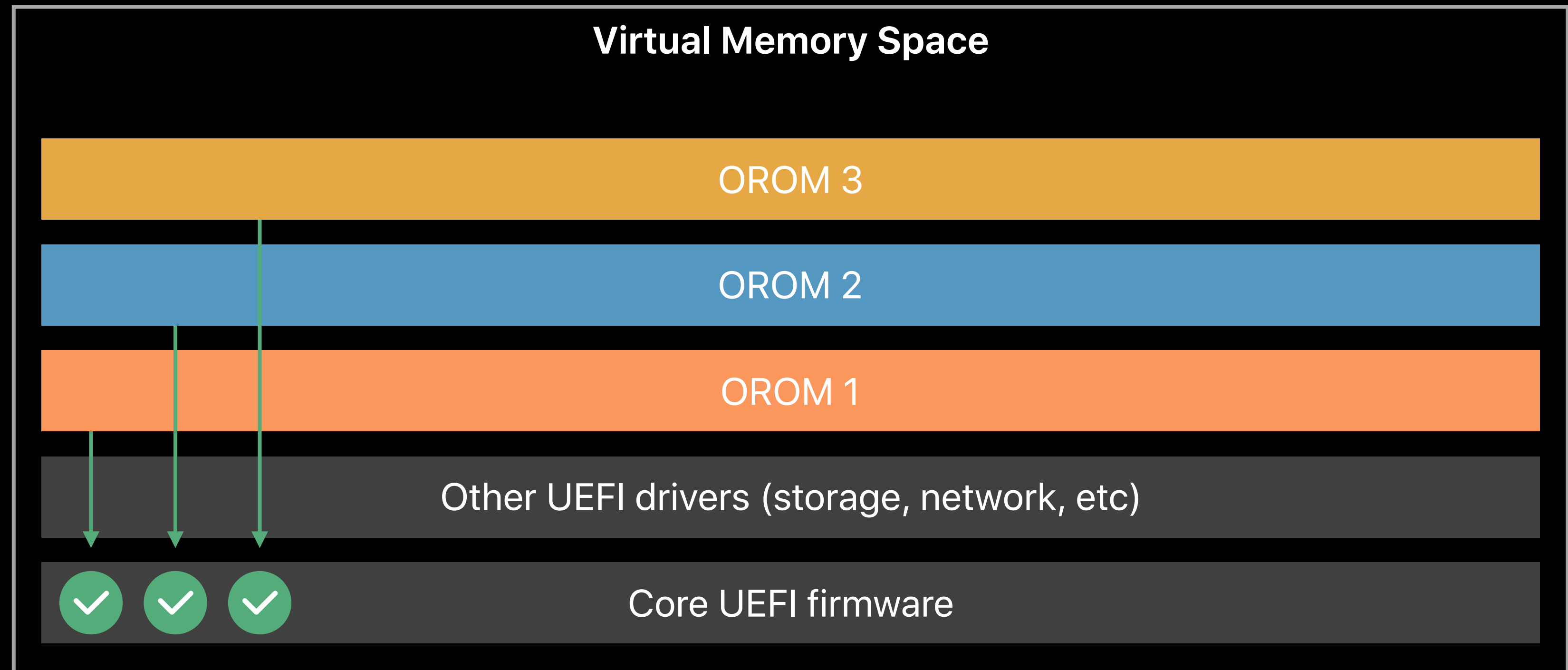
Ring 0
(More privileged)



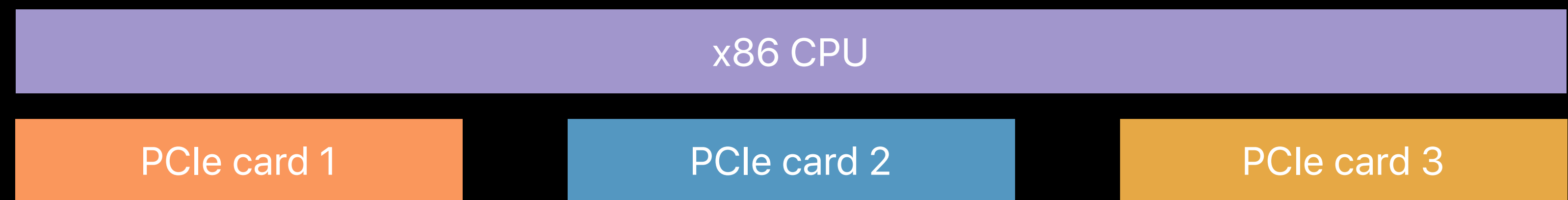
Hardware
(More privileged)



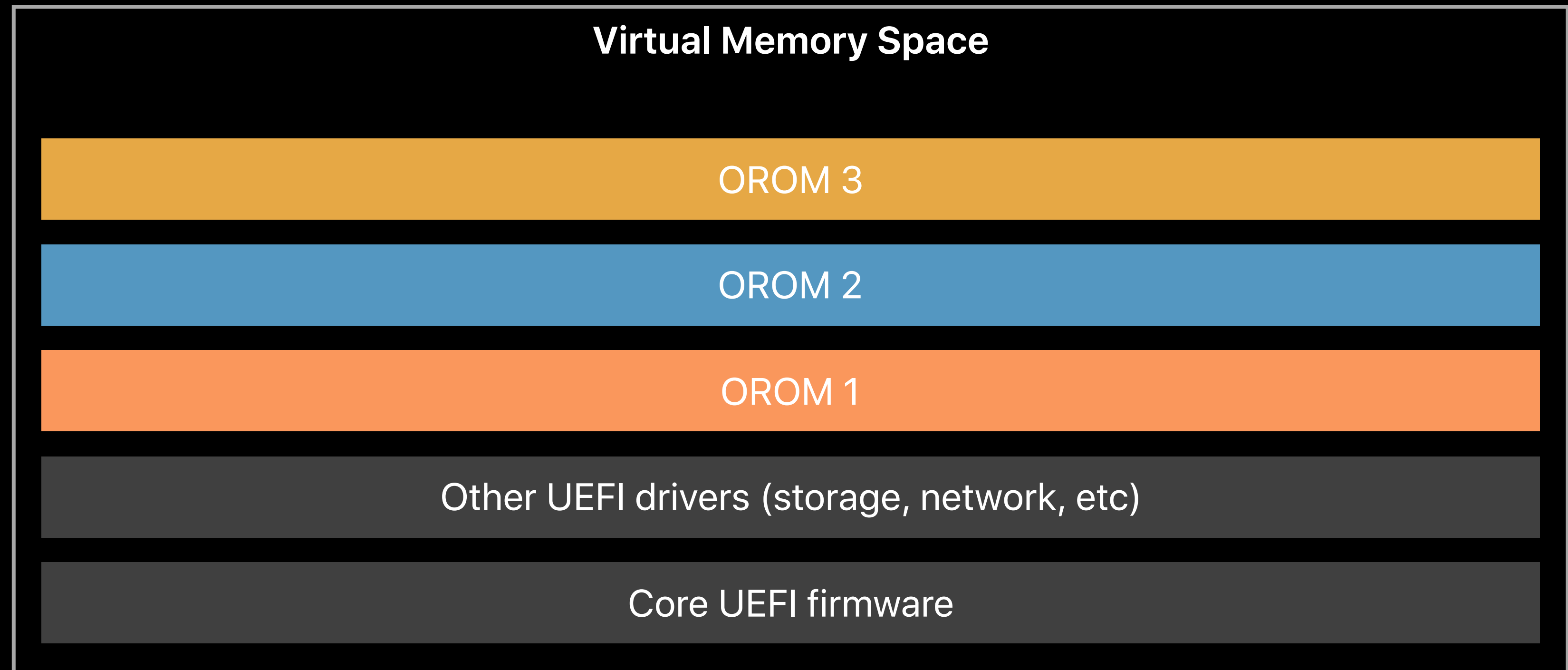
Ring 0
(More privileged)



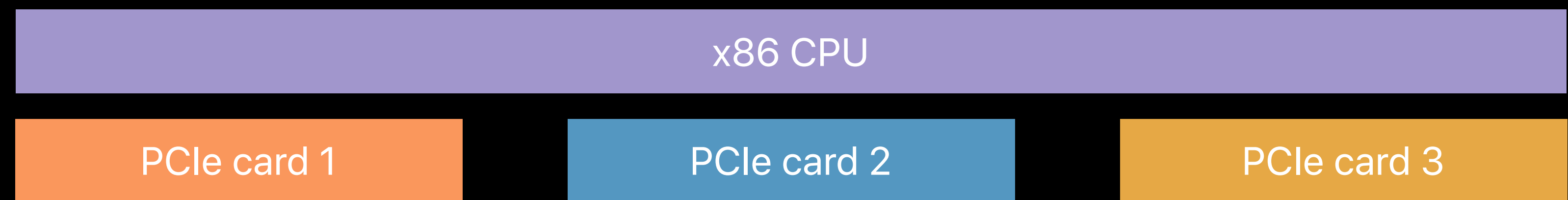
Hardware
(More privileged)



Ring 0
(More privileged)



Hardware
(More privileged)

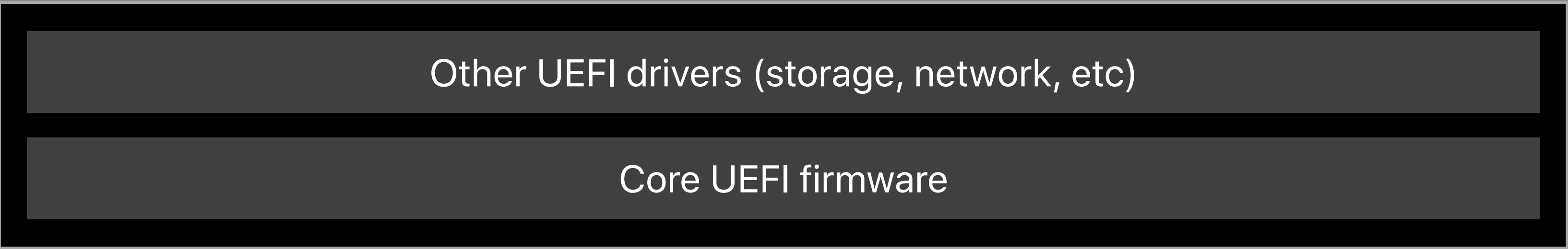


Virtual Memory Space

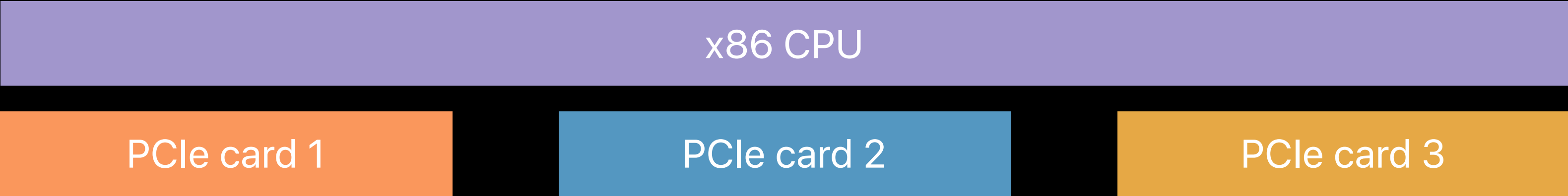
Ring 3
(Less privileged)

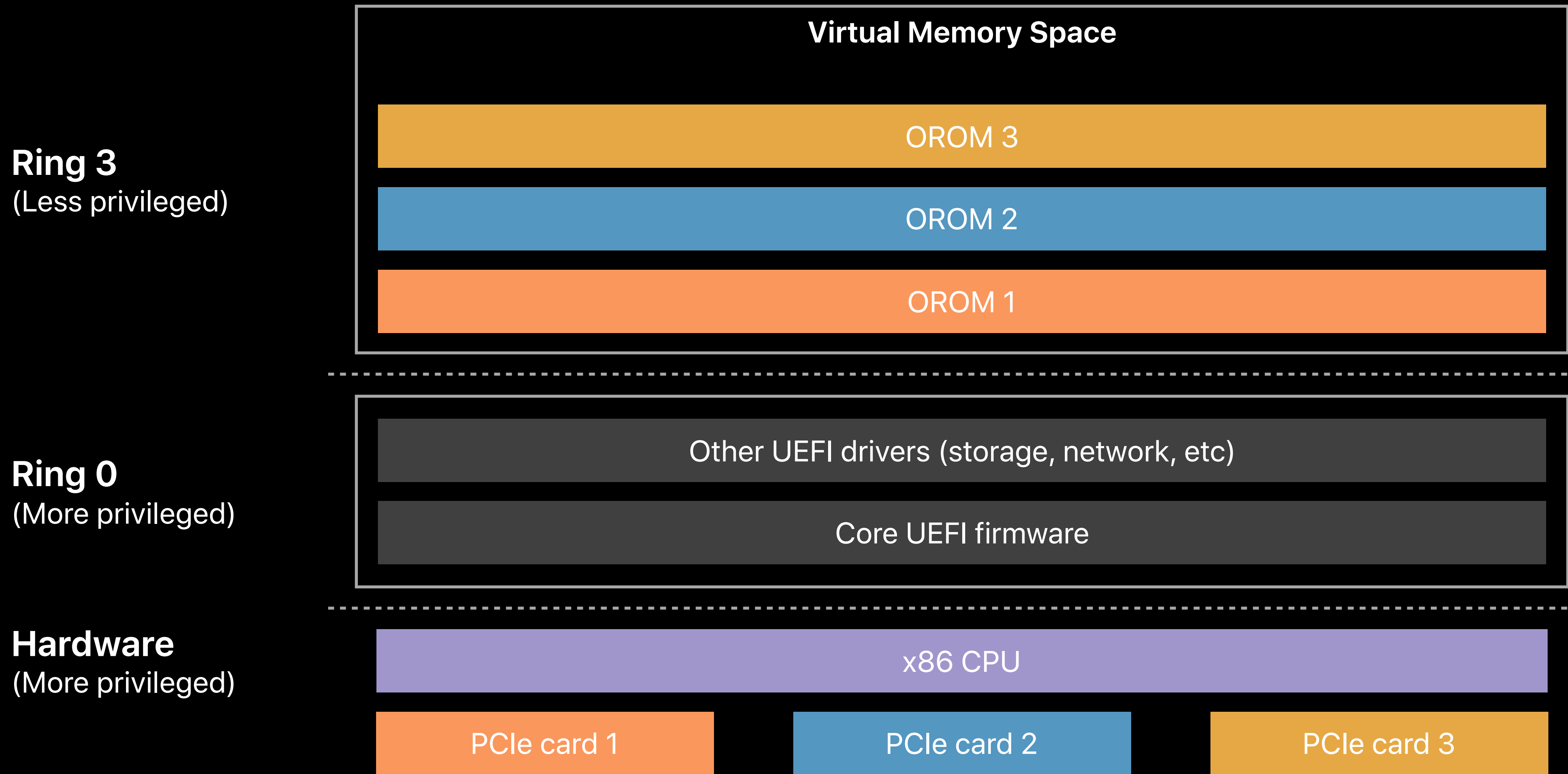


Ring 0
(More privileged)



Hardware
(More privileged)





Ring 3
(Less privileged)



OROM 1

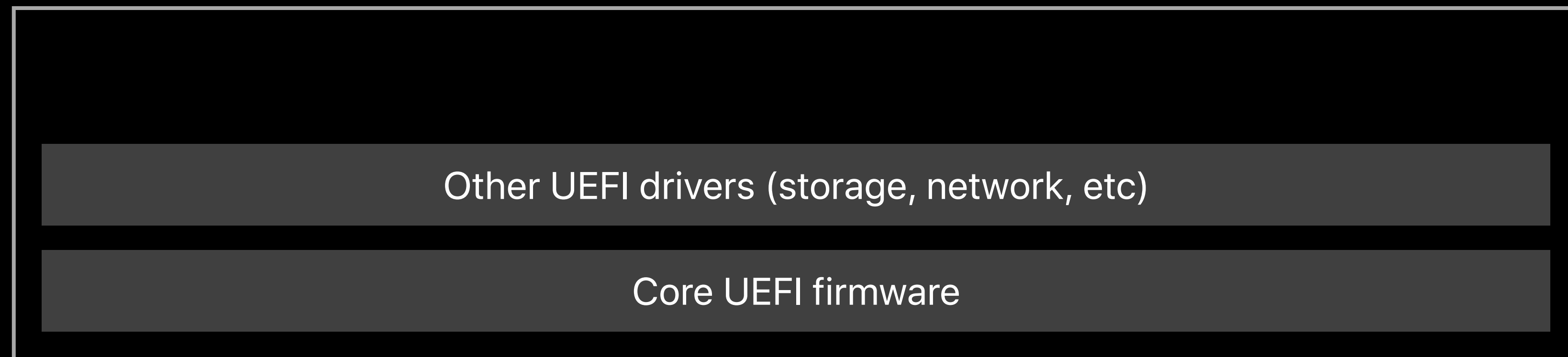


OROM 2



OROM 3

Ring 0
(More privileged)



Other UEFI drivers (storage, network, etc)

Core UEFI firmware

Hardware
(More privileged)



x86 CPU



PCIe card 1

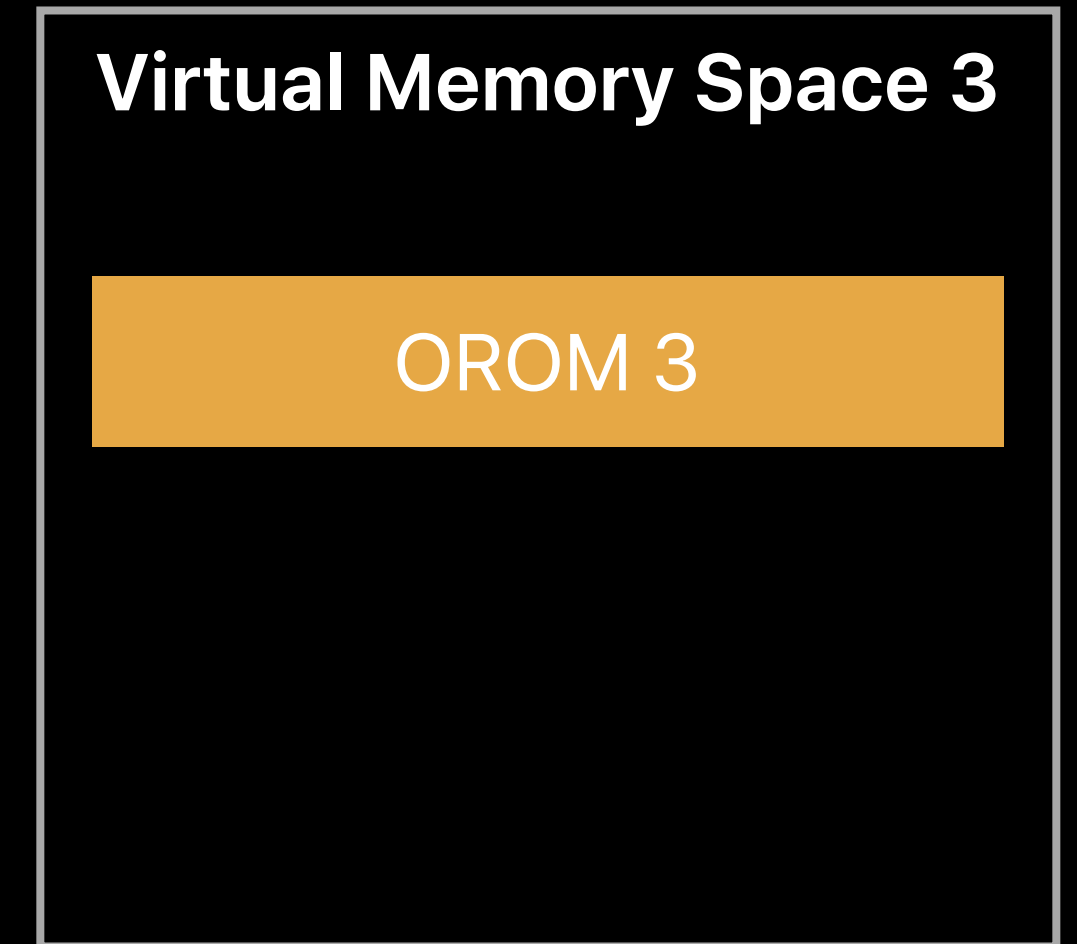
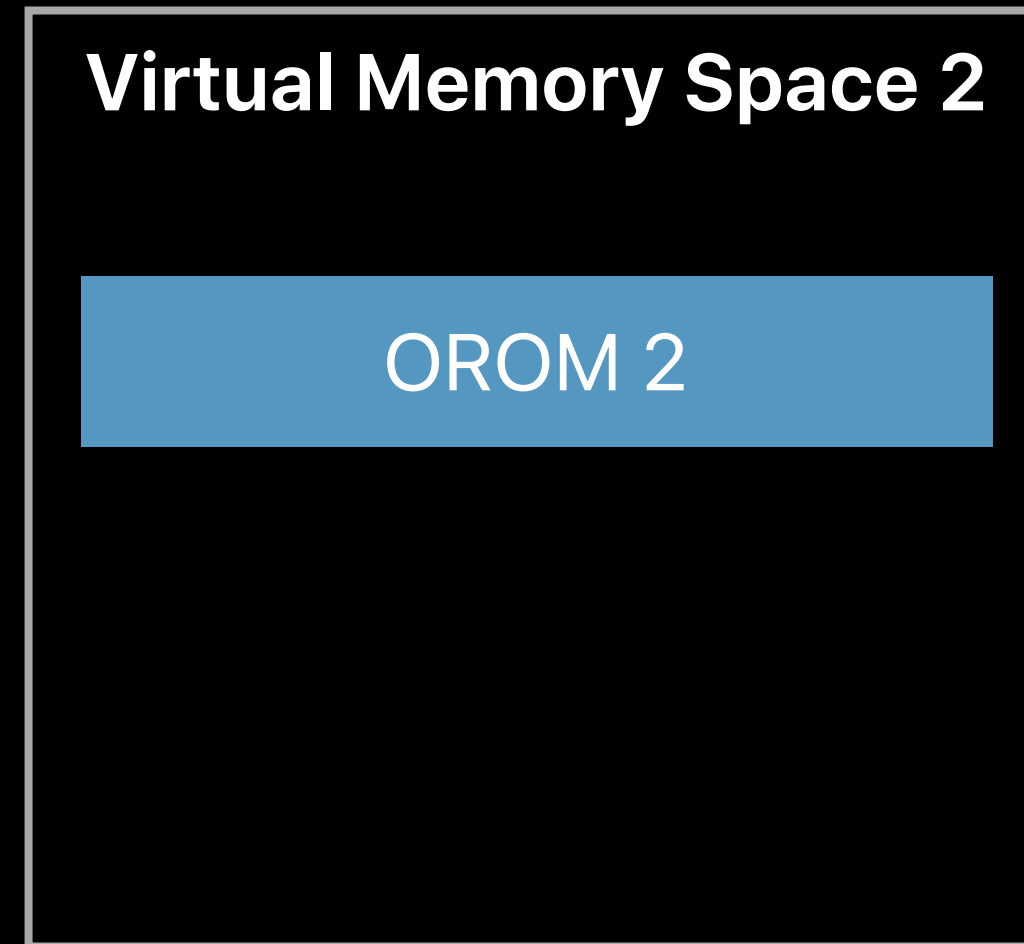
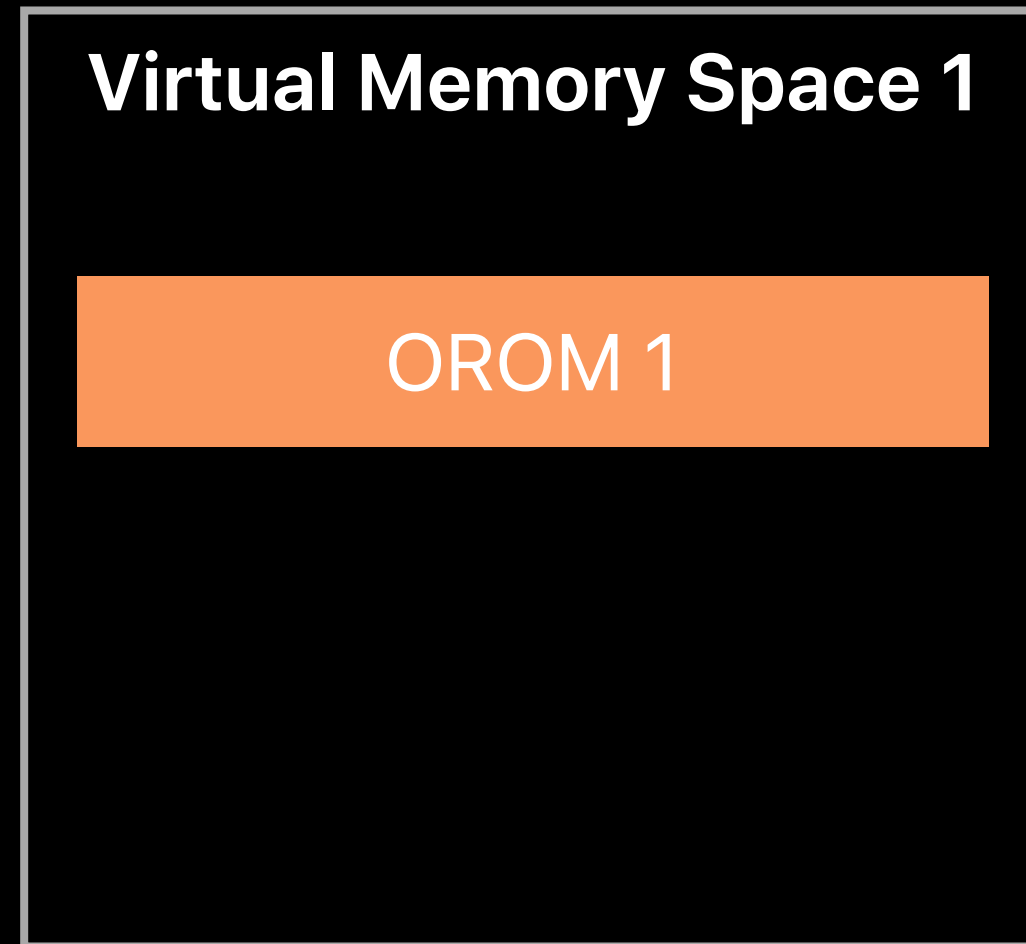


PCIe card 2

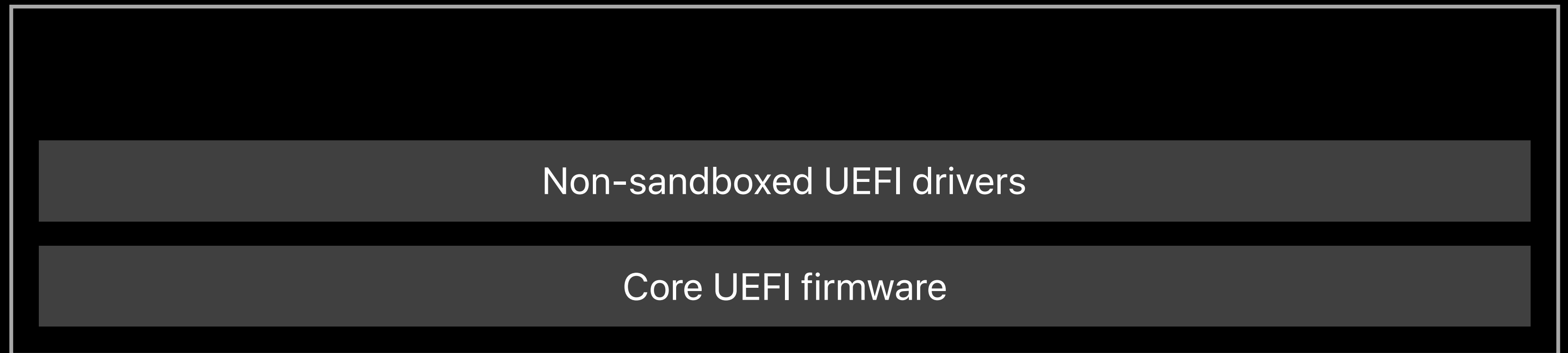


PCIe card 3

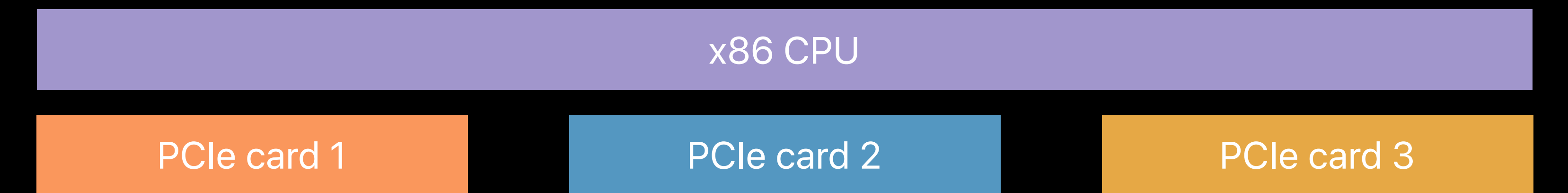
Ring 3
(Less privileged)



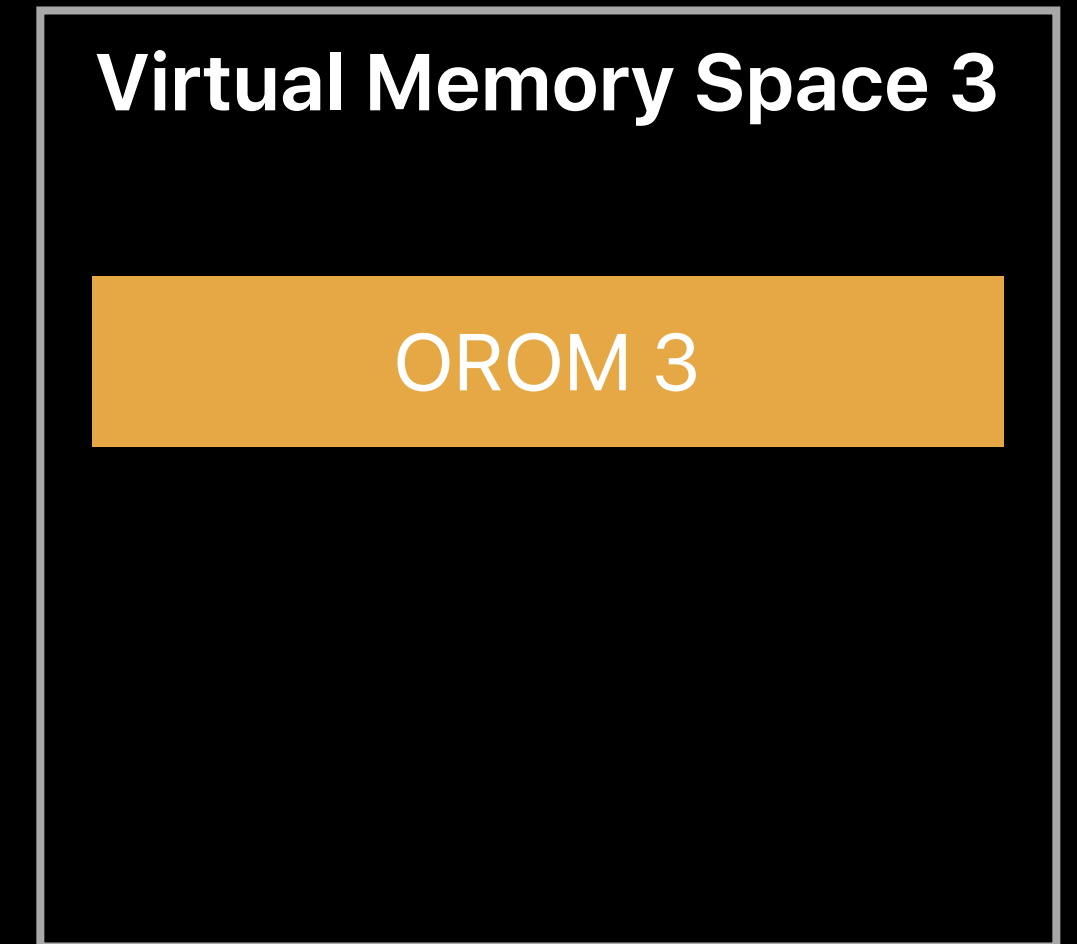
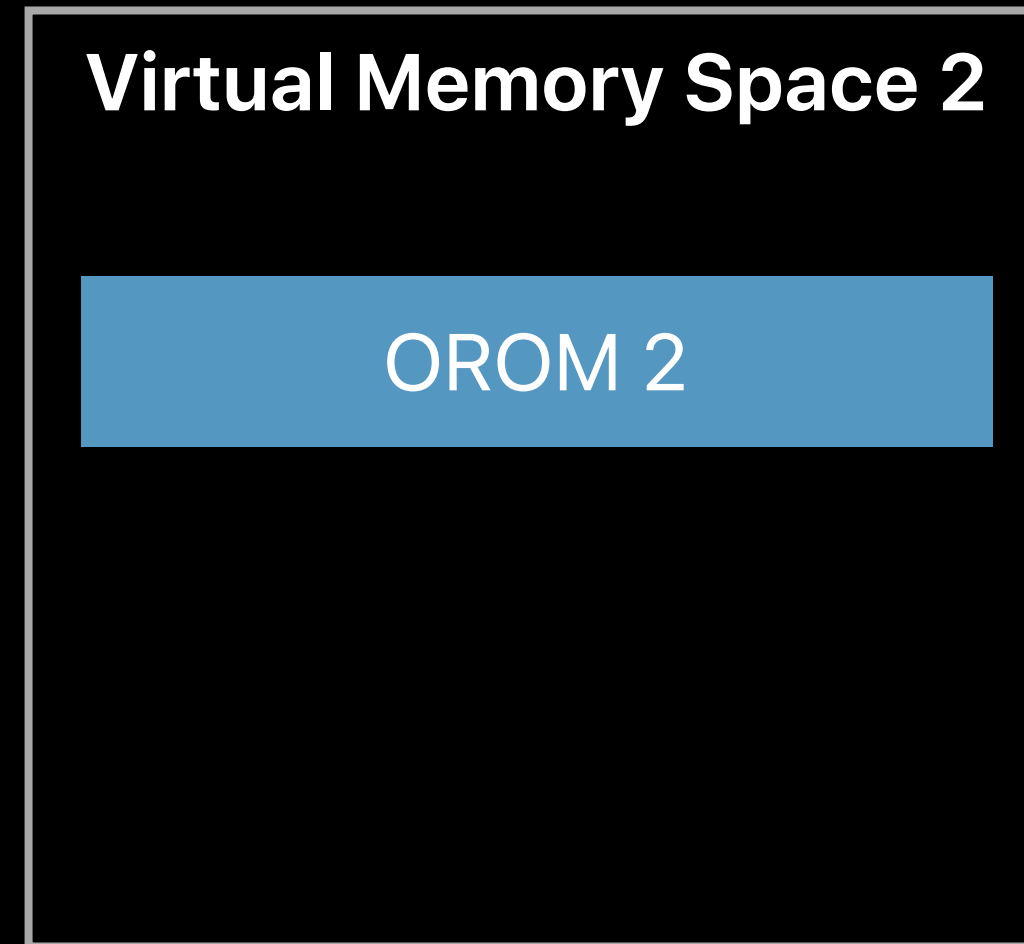
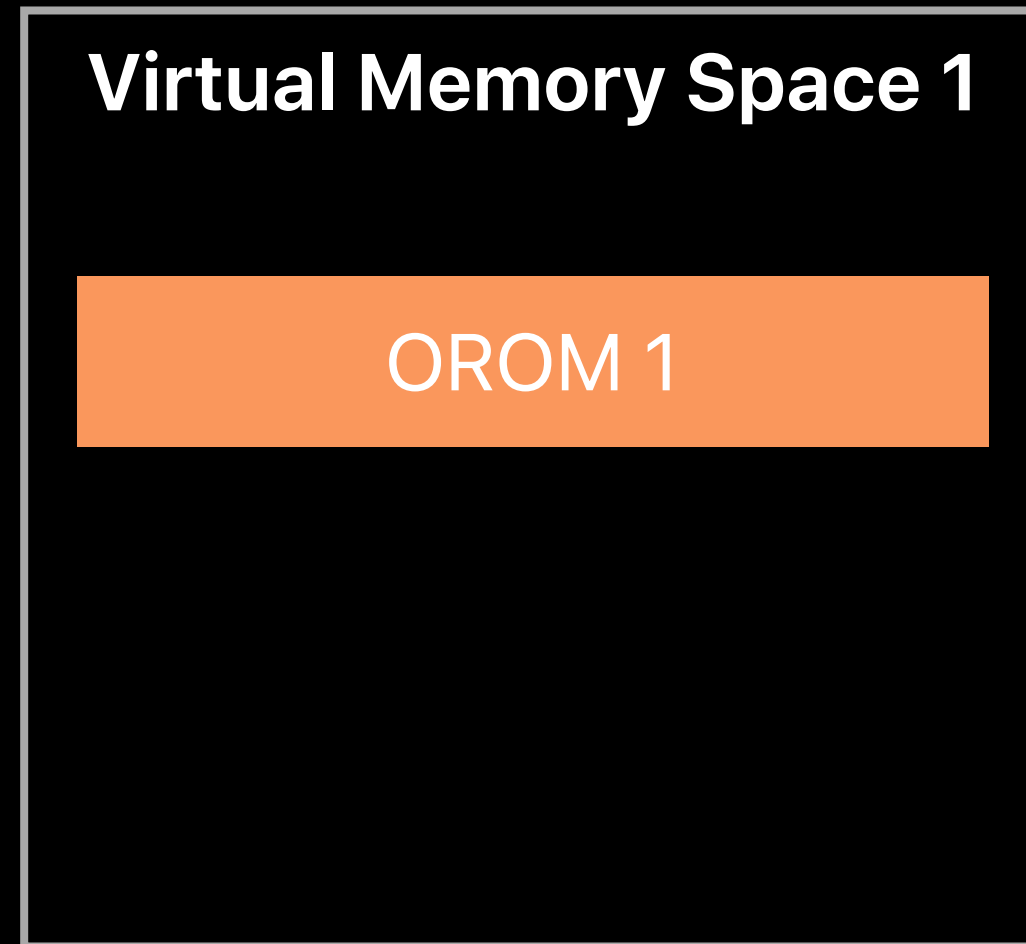
Ring 0
(More privileged)



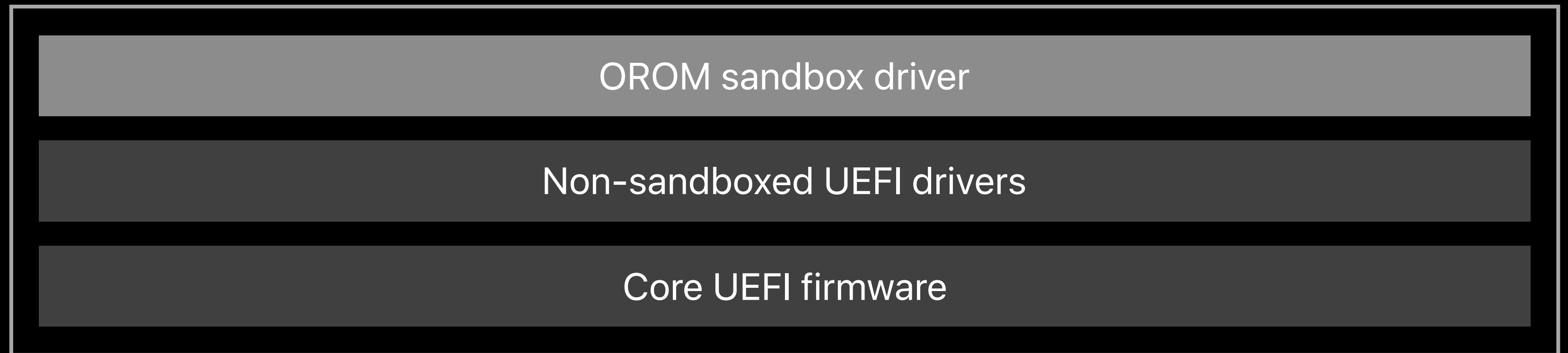
Hardware
(More privileged)



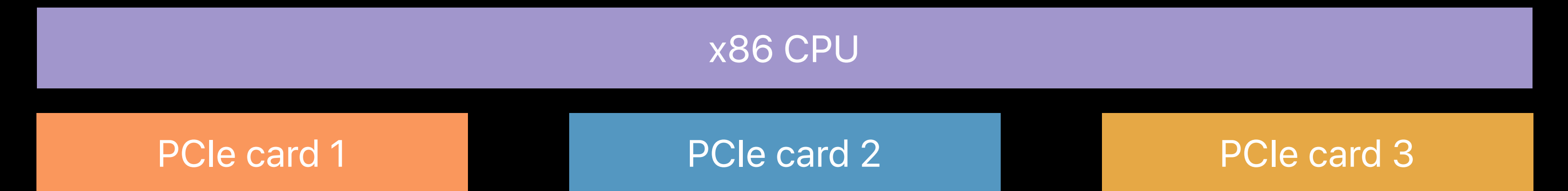
Ring 3
(Less privileged)



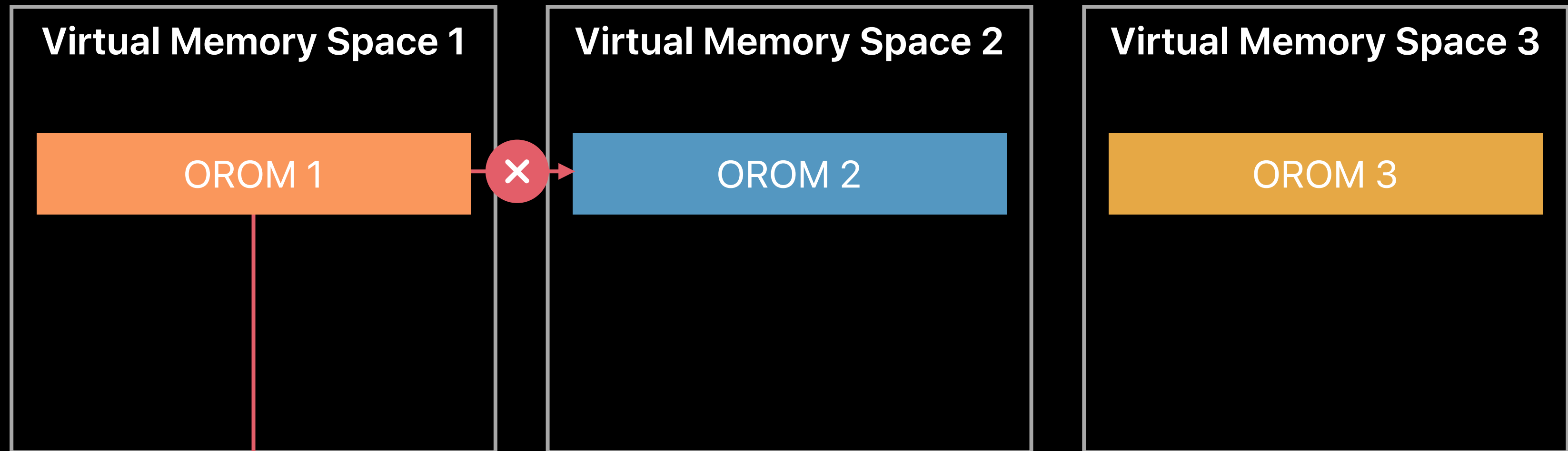
Ring 0
(More privileged)



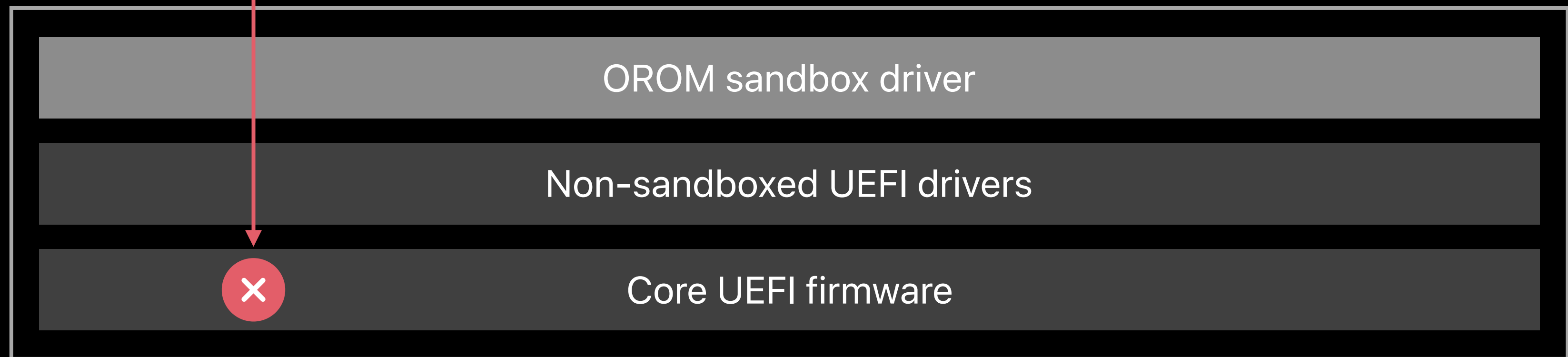
Hardware
(More privileged)



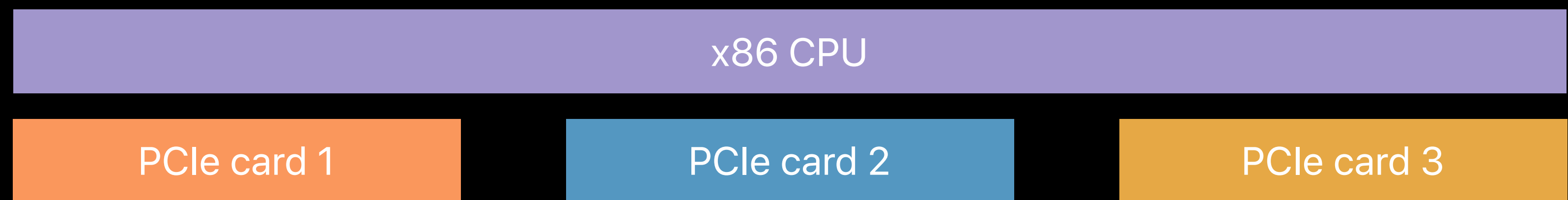
Ring 3
(Less privileged)



Ring 0
(More privileged)



Hardware
(More privileged)



OROM Sandbox

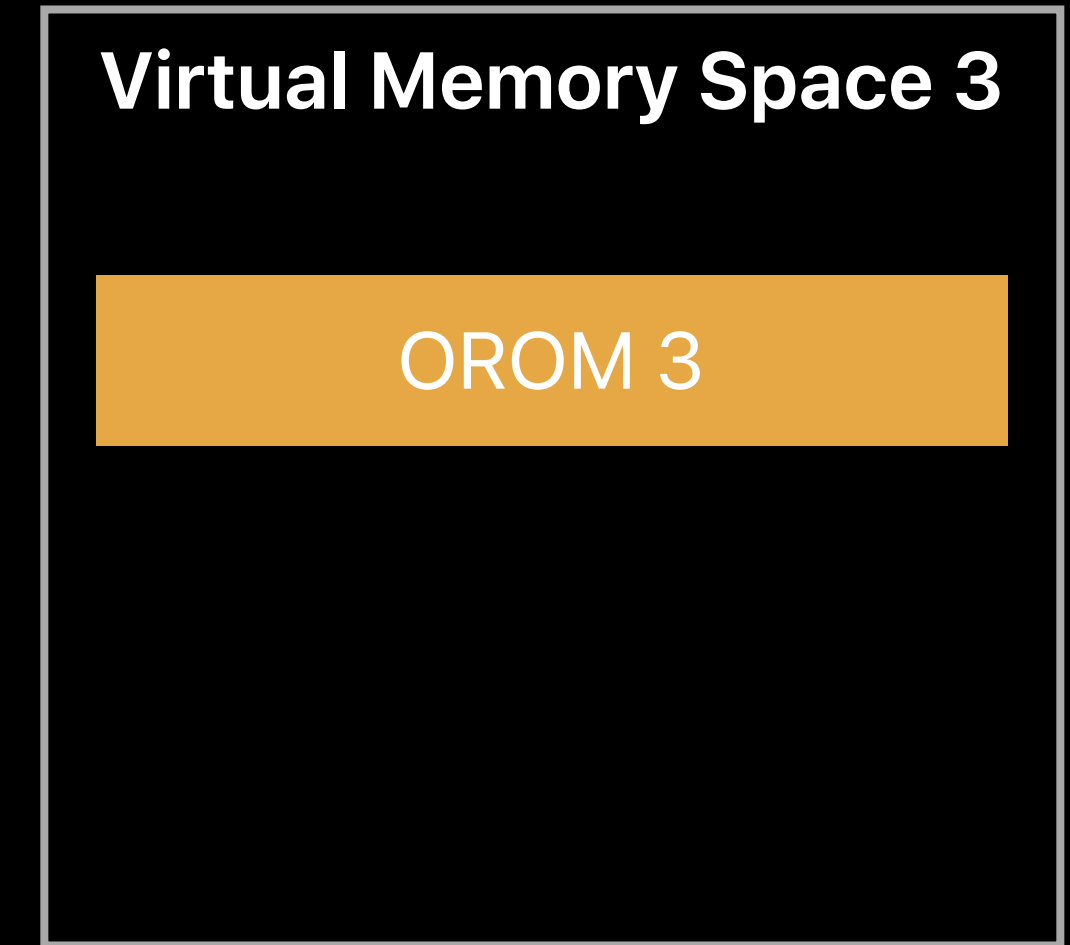
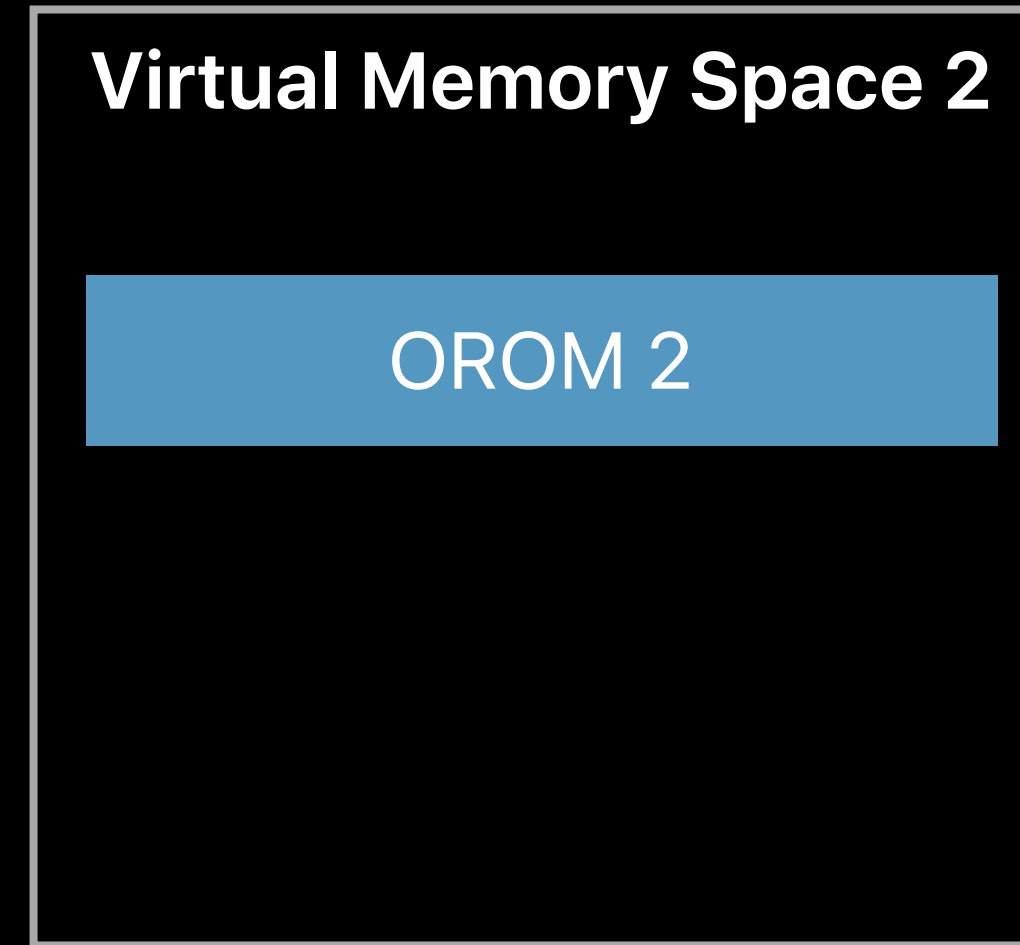
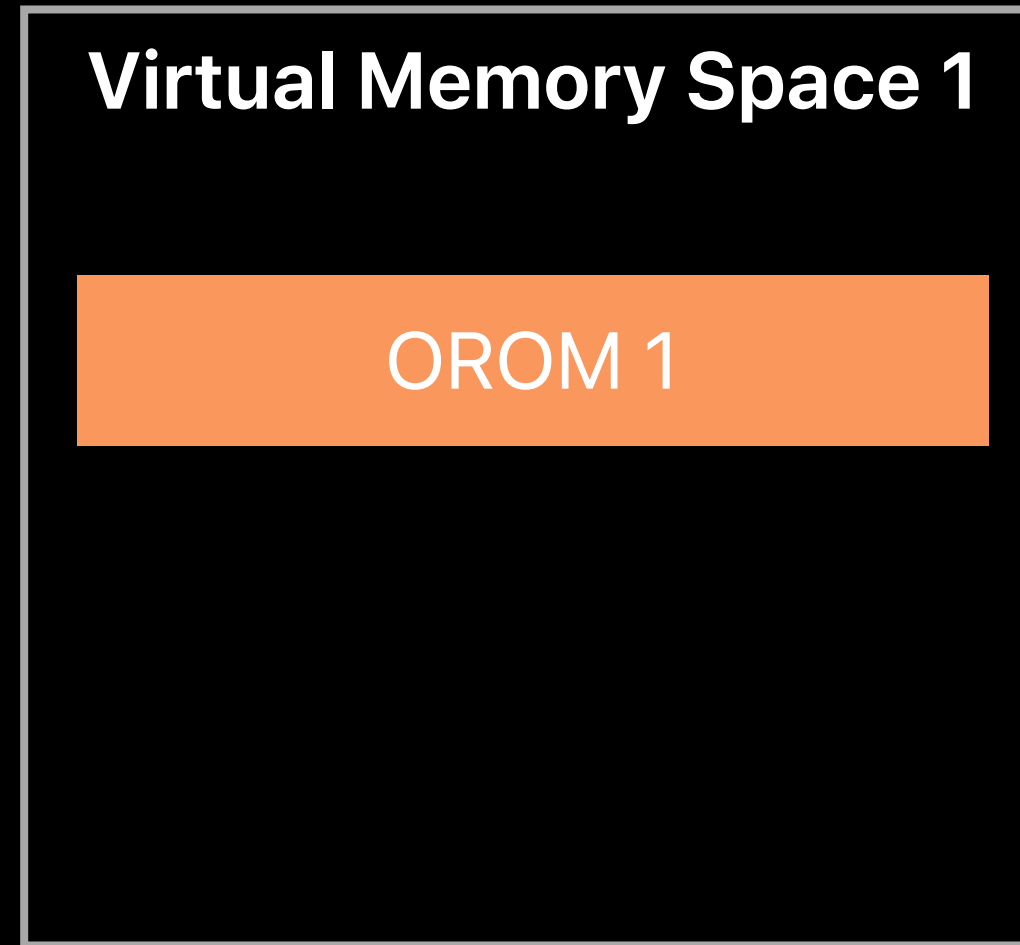
OROMs can only *call* a limited subset of expected UEFI interfaces

- Similar to system call filtering

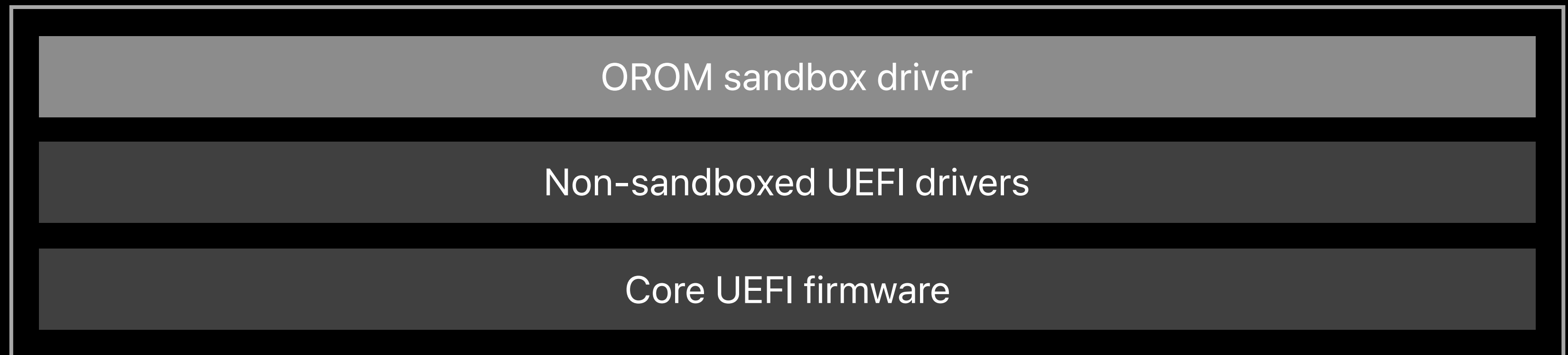
OROMs can only *install* a limited subset of expected UEFI interfaces

- E.g. read and write to disk blocks, or draw to graphics

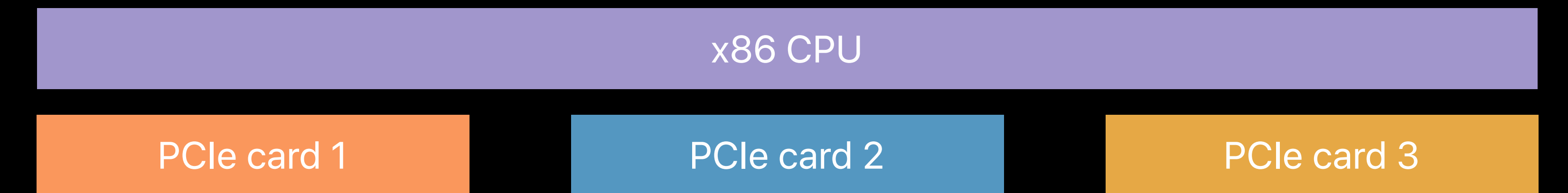
Ring 3
(Less privileged)



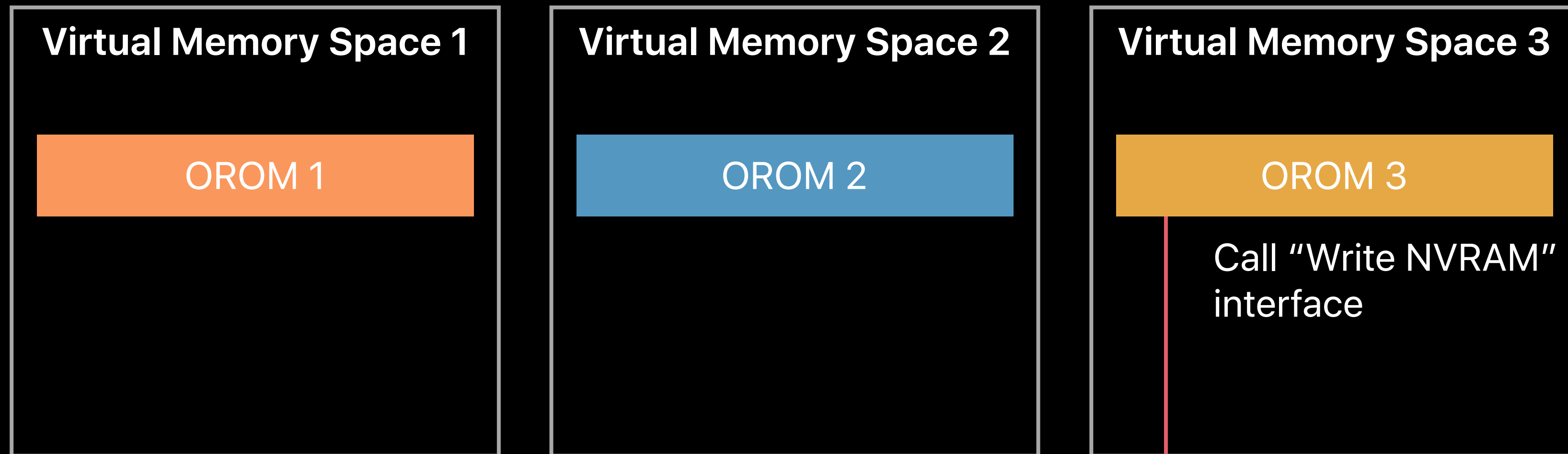
Ring 0
(More privileged)



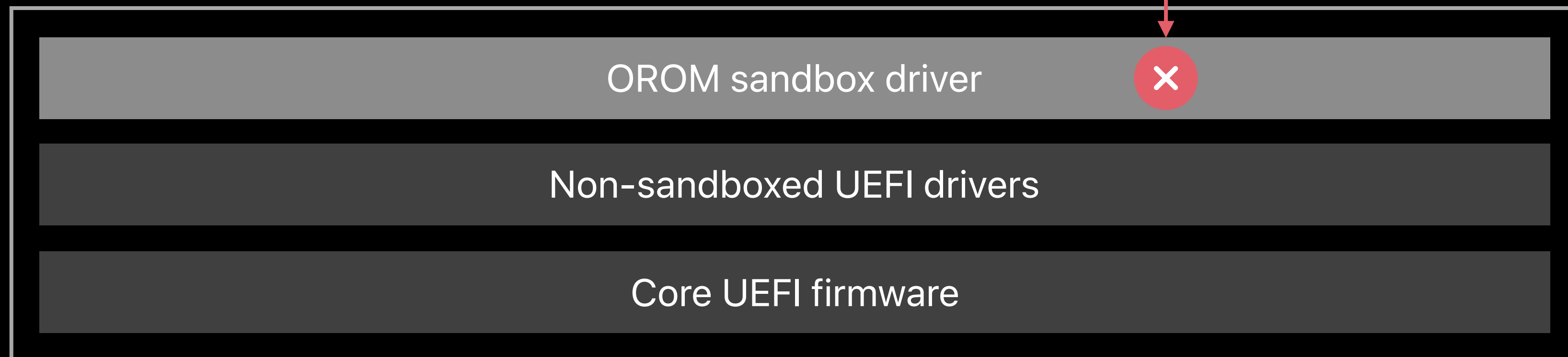
Hardware
(More privileged)



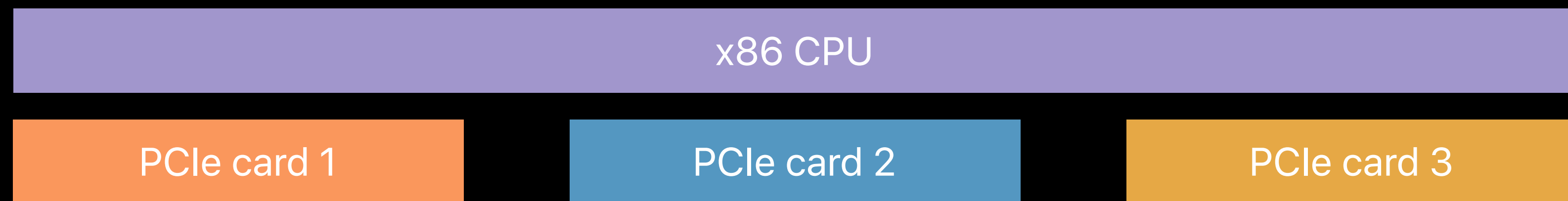
Ring 3
(Less privileged)



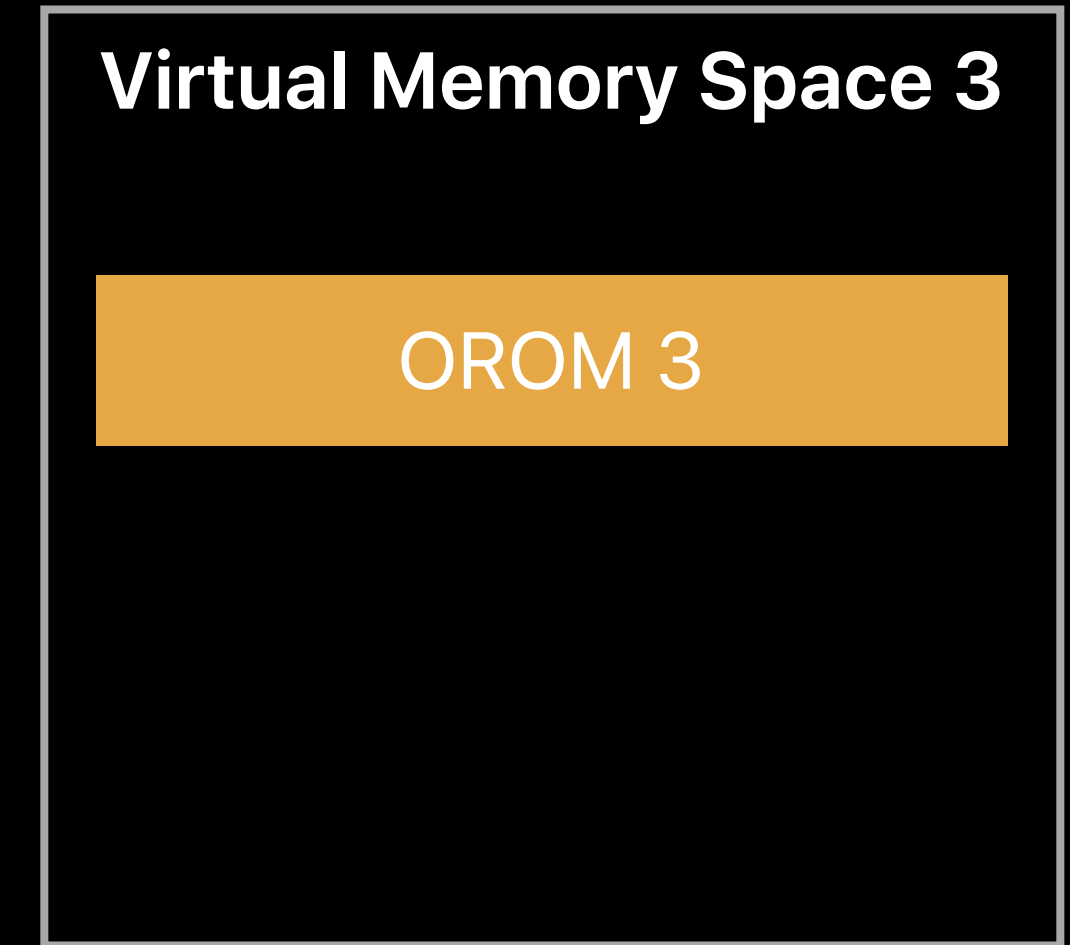
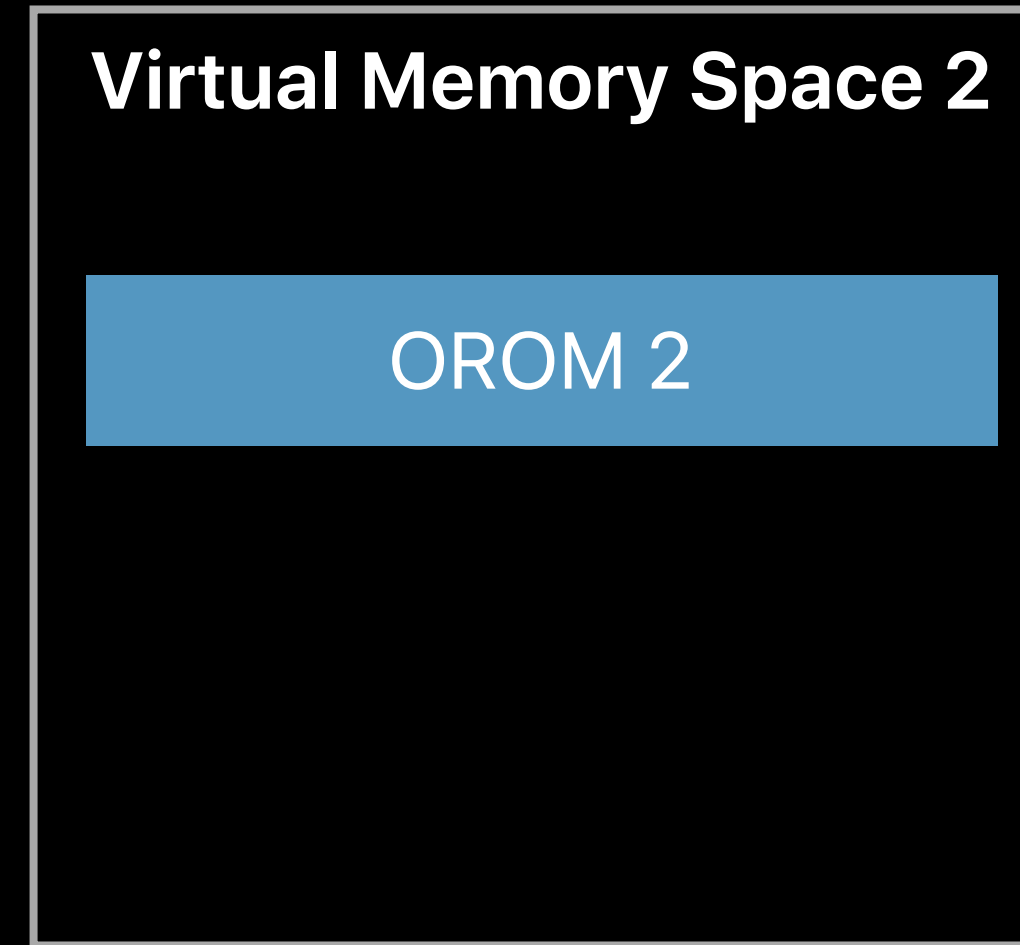
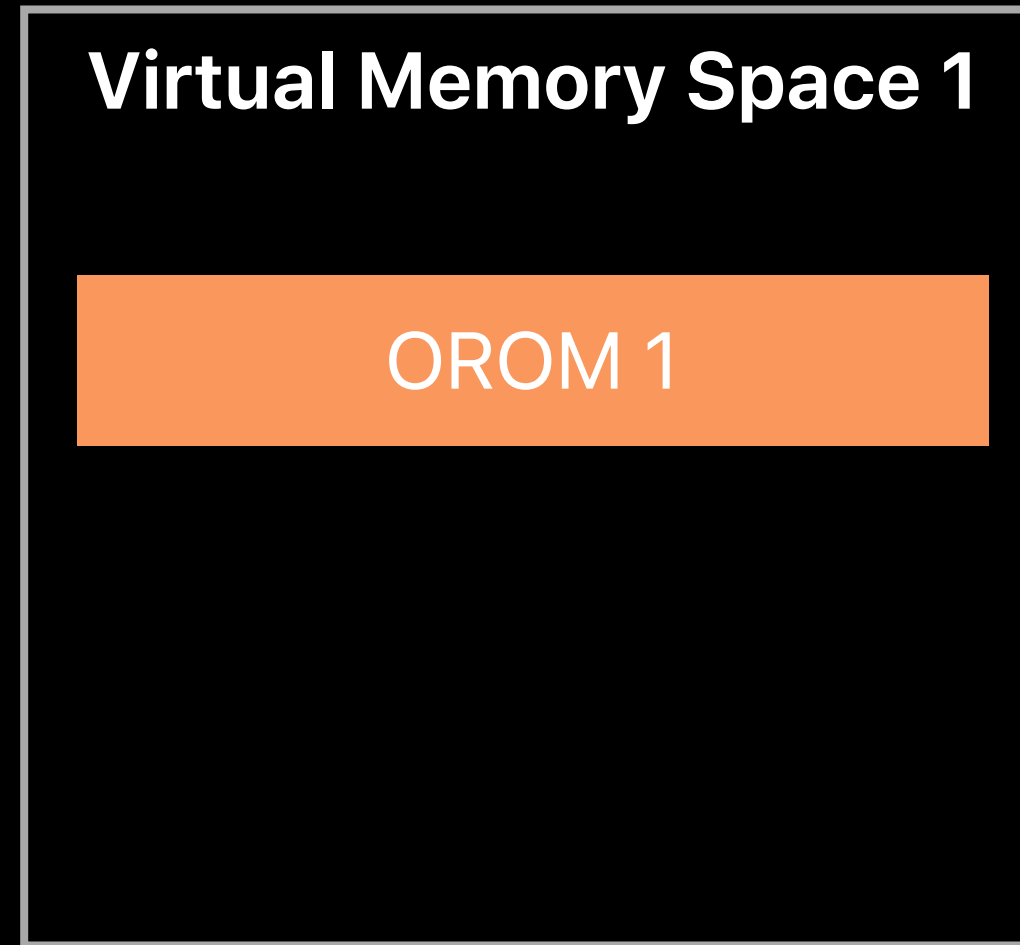
Ring 0
(More privileged)



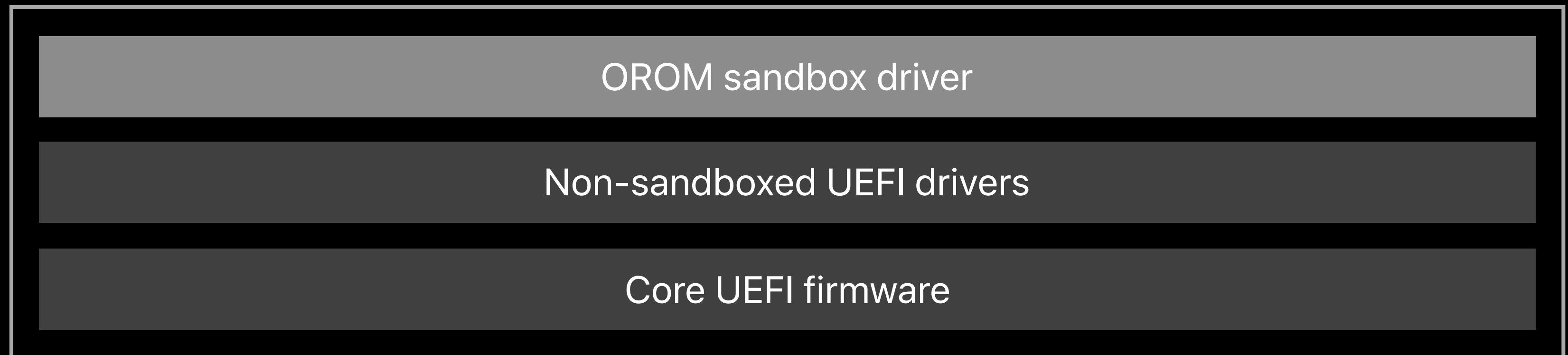
Hardware
(More privileged)



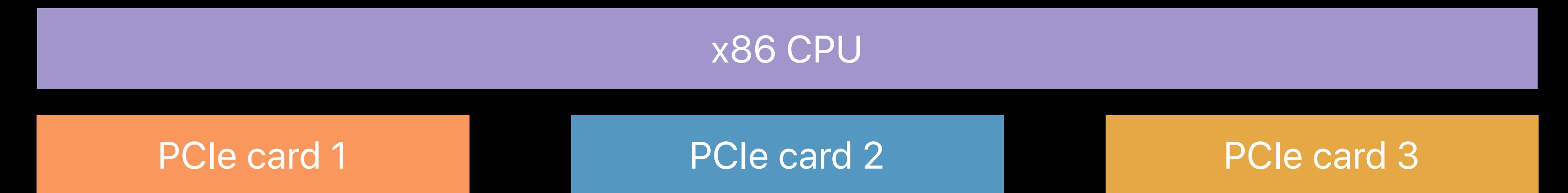
Ring 3
(Less privileged)

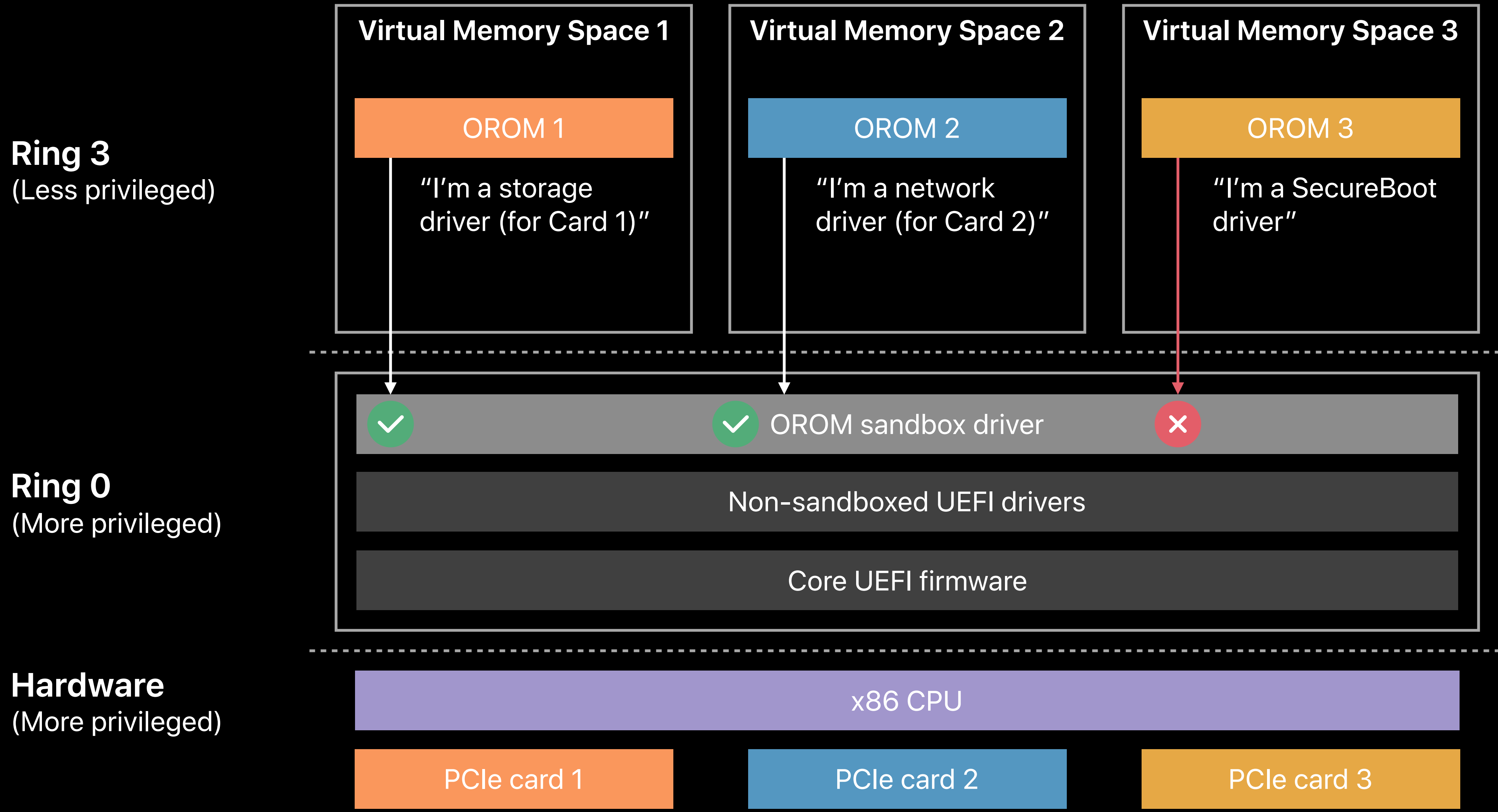


Ring 0
(More privileged)



Hardware
(More privileged)





OROM Sandbox

OROM can only talk to the assigned device in its sandbox

- This is the device it was embedded on

The VT-d policy allows a device to DMA to any memory allocated within its OROM's sandbox

- Preserve high-throughput DMA but with strong VT-d protection
- OROM doesn't even have to be VT-d aware!

EFI Exploit Mitigations

OROM Sandbox will drive attackers to privilege escalation and sandbox escapes

We added a strong set of exploit mitigations to EFI on T2 systems

- Stack Cookies
- All EFI memory W^X with read-only page tables
- SMAP: Ring 0 can't directly read/write Ring 3 data
- SMEP: Ring 0 can't execute Ring 3 code
- Some Spectre/Meltdown mitigations

Mac Secure Boot Summary

The T2 Security Chip brings key secure boot properties from iOS to the Mac, far outclassing UEFI SecureBoot-based systems

Our DMA protection for PCIe Bus 0 provides state-of-the-art protection against DMA attacks targeting firmware

The Mac OROM Sandbox provides unprecedented defense against malicious PCIe Option ROMs compromising the secure boot process

Mac secure boot

iOS code integrity protection

Find My

Software Enforced Code Integrity

Before iOS 9

Kernelcache signature verified by iBoot at load time

Userland `__TEXT` pages code signed

- CodeDirectory checked at load time (or static)
- Pages checked at fault time

Compromised kernel could change its own `__TEXT`

Compromised kernel could disable codesigning altogether,
or alter userland pages

Kernel Integrity Protection

Goal

- Maintain integrity of kernel code and read-only data after secure boot

Threat model

- Kernel arbitrary read/write
- Arbitrary kernel instruction pointer control
- Arbitrary read/write by DMA agents and system coprocessors

Out of scope

- Secure boot bypass

Kernel Integrity Protection v0

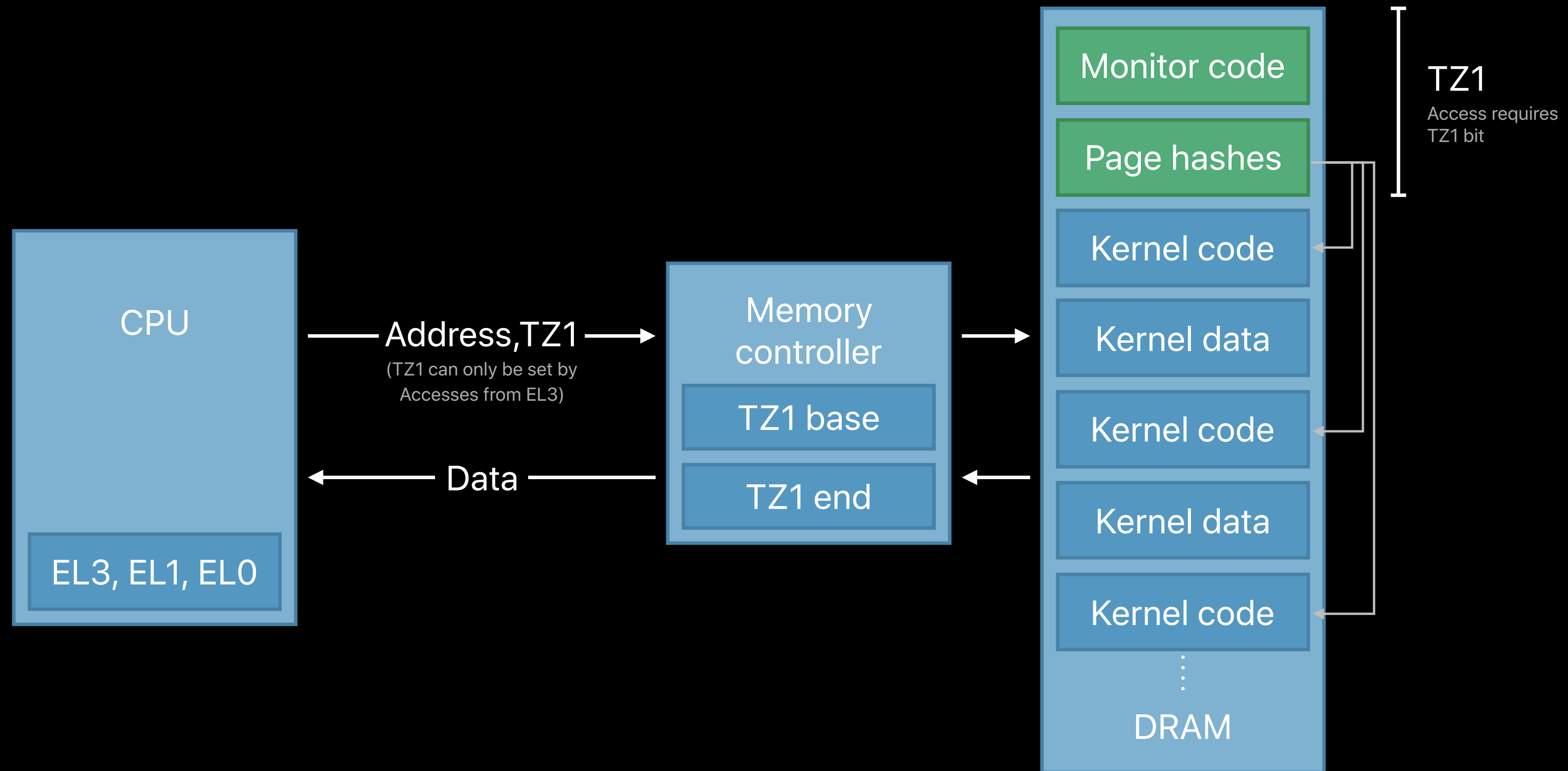
iOS 9

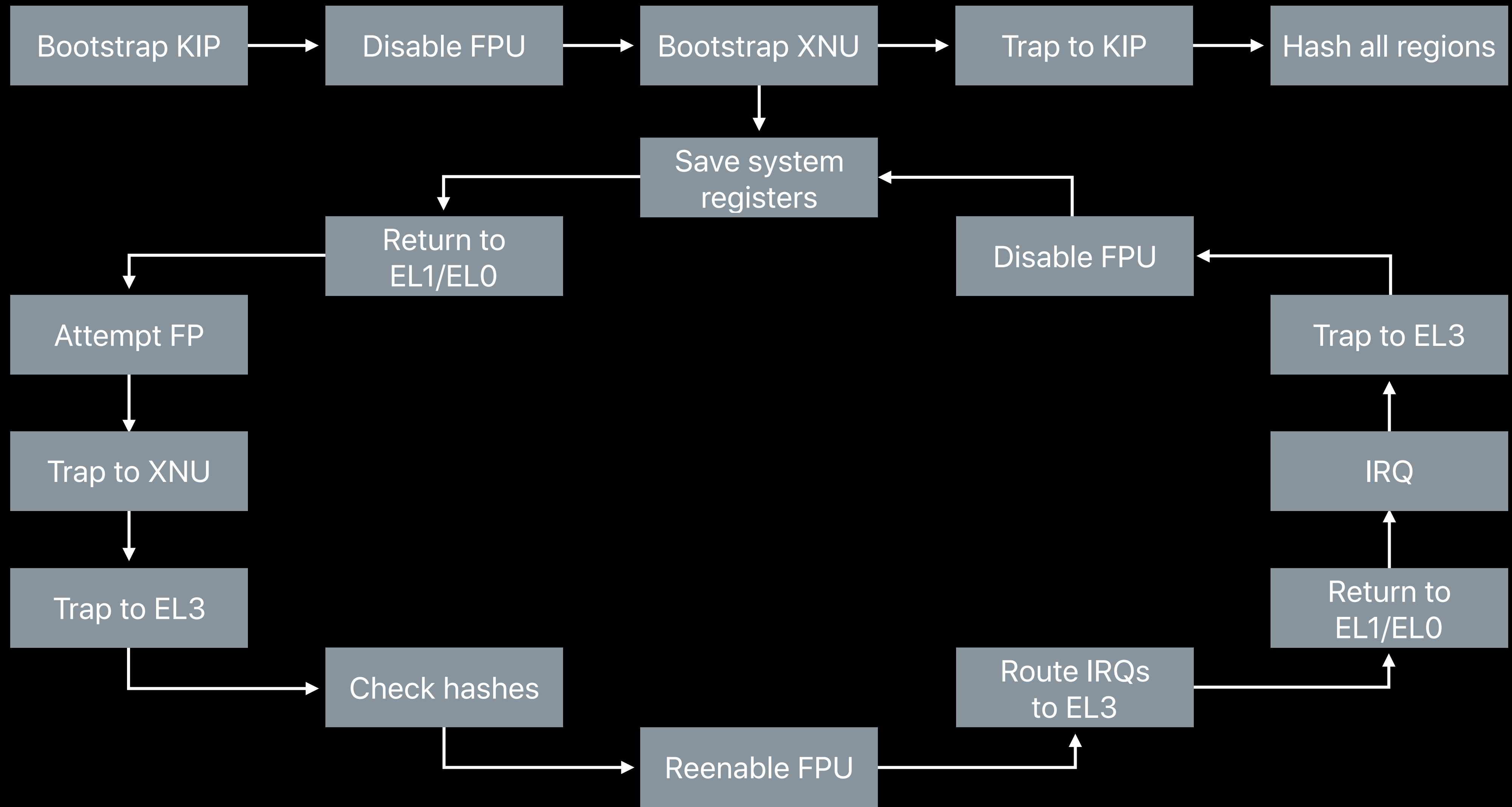
At system initialization, EL3 monitor creates array of kernel page table and text hashes in TZ1

Monitor periodically verifies hashes, panics on mismatch

Effective against long-lived patches, inherently vulnerable to races

Kernel Integrity Protection v0





Lessons Learned

Must protect critical data in addition to code

- Page tables
- Global offset table entries
- Sandbox configuration

Integrity verification after boot is vulnerable to race conditions

Easier to adapt hardware architecture to fit security requirements

Kernel Integrity Protection v1

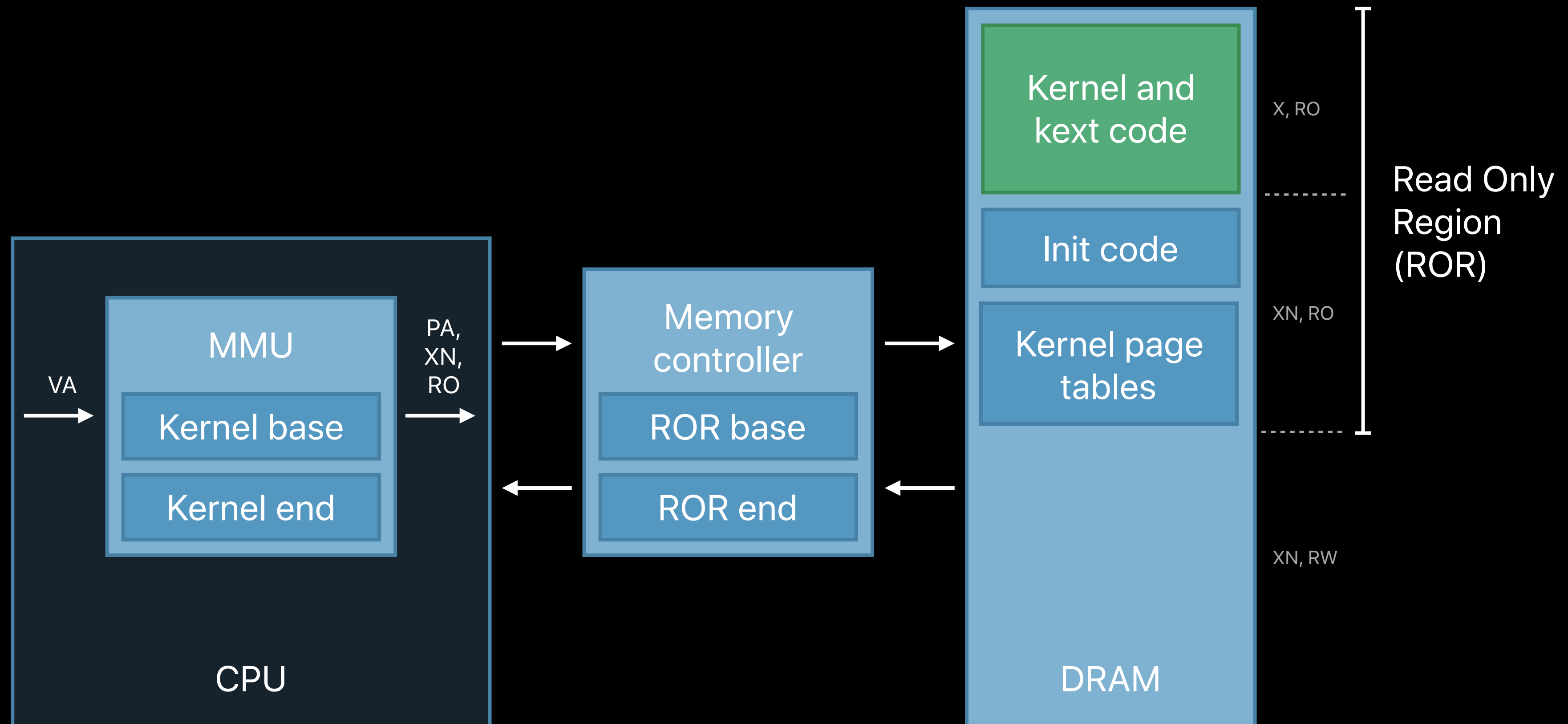
iPhone 7

New hardware design tailored to our goals

Our threat model had three hardware requirements

- CPU prevents modification of kernel memory
- CPU also prevents EL1 execution of non-kernel memory
- Memory controller prevents DMA writes to protected physical range

Kernel Integrity Protection v1



Kernel Integrity Protection v1: Read-Only Data

We have a strong design for code, but protecting data requires additional finesse

Neither KIP v0 nor KIP v1 prevent modification of TTBR1, which tells CPU where to find the kernel's page tables

By using a very careful initialization sequence, we make sure no instructions are available to modify TTBR1 after CPU finishes initializing

Kernel Integrity Protection v1

Required significant rework of kernelcache layout

Build time checks that no TTBR1 write gadget exists

Very effective at protecting kernel code integrity

Only public bypass was off-by-one error in our protection range calculation

Kernel Integrity Protection v2

iPhone Xs

Applied lessons learned from KIP v1

Control bits prevent changes to TTBR1, MMU enable, and exception vector addresses

- Guarantees in hardware that MMU configuration cannot be modified
- Replaces init-only instructions from KIP v1

Configuration is retained when CPU goes into idle power-off

- Less complexity in power management transitions

Kernel Integrity Protection

Summary

Robust enforcement of kernel code and read-only data integrity

Hardware implementation tailored to software security requirements

Essential foundation for next-generation security features

Fast Permission Restrictions (APRR)

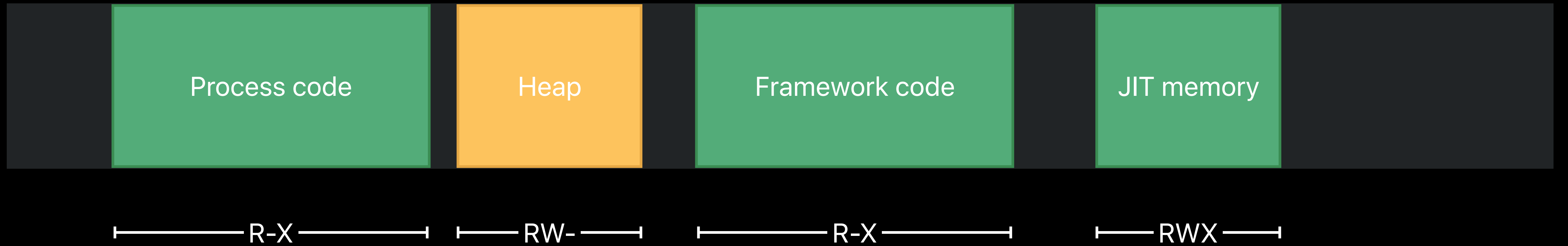
iPhone X

Builds upon software-only Hardened WebKit JIT Mapping in iOS 10

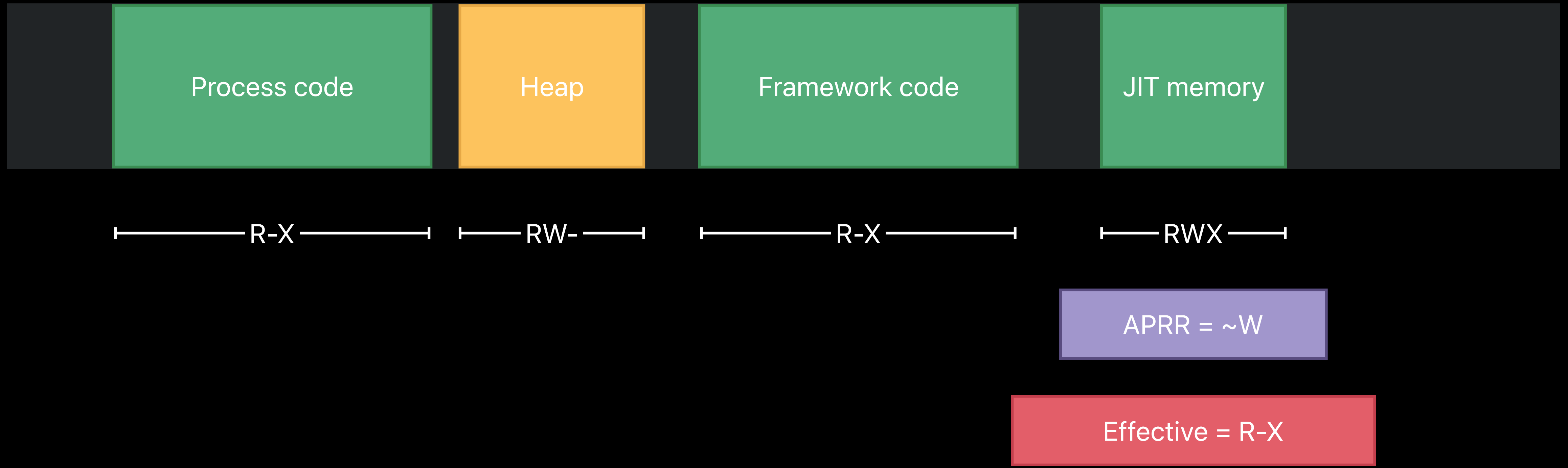
CPU register to quickly restrict permissions on RWX memory, per thread

Removes overhead of a syscall and walking page tables to change permissions

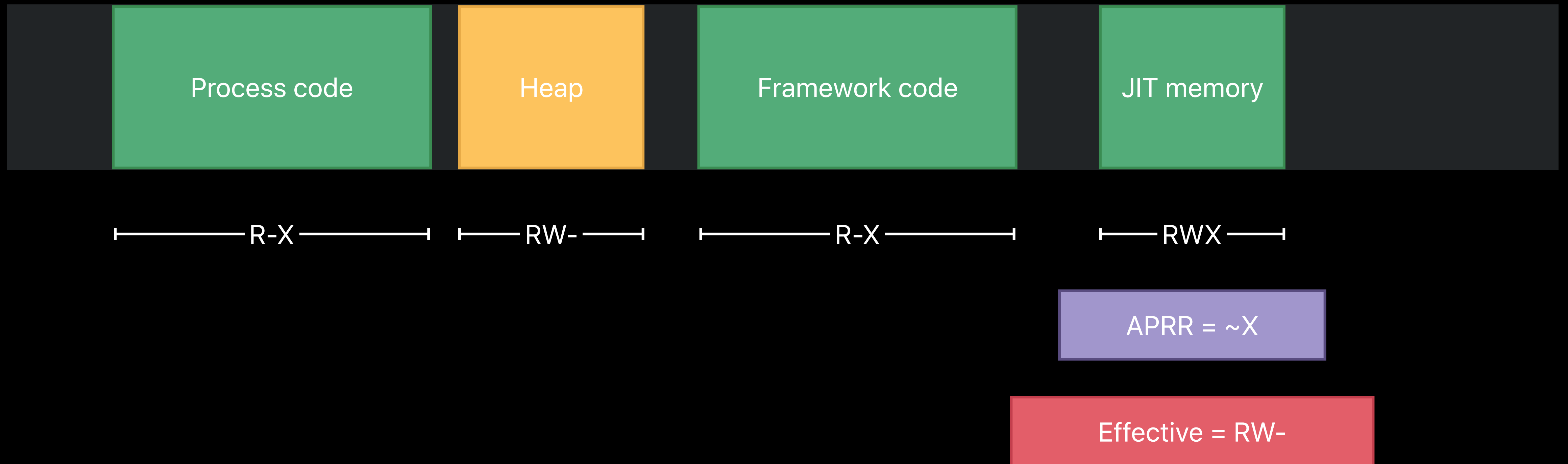
Pre-APRR VM Permissions



APRR: JavaScriptCore Execution Threads



APRR: JavaScriptCore JIT Compiler Thread



What about userland?

Protecting Userland Integrity

KIP gives us strong integrity protection for kernel text

Page table overrides with KIP rely on kernel code being static

Userland code is dynamically loaded, so we would need dynamic overrides

Page Protection Layer (PPL)

iPhone Xs

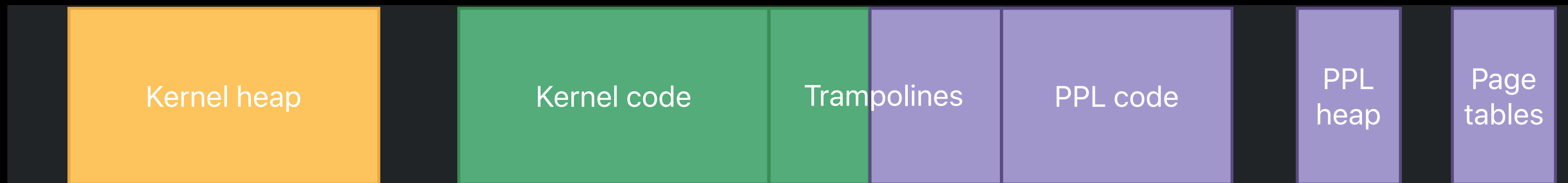
Ensures userland code can't be modified after code signature checks complete

Built upon KIP and APRR

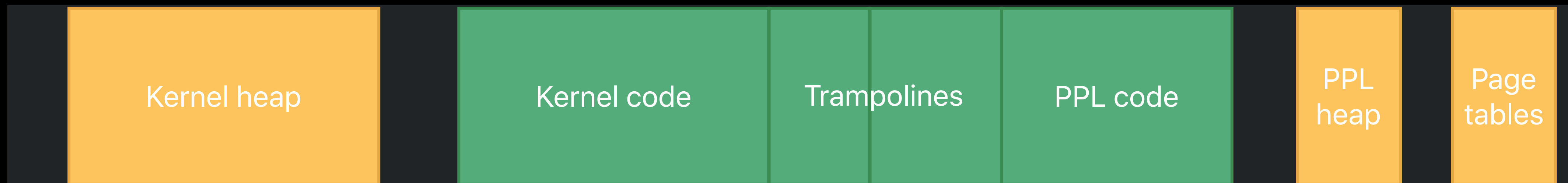
Manages page tables, code signing validation

Small TCB

Guarantees only code inside PPL can alter protected pages



Default	RW-	R-X	R-X	R-X	R-X	RW-	RW-
APRR				~X	~X	~W	~W
Effective	RW-	R-X	R-X	R--	R--	R--	R--



Default RW- R-X R-X R-X R-X RW- RW-

APRR

Effective RW- R-X R-X R-X R-X RW- RW-

Page Protection Layer

Summary

System-wide dynamic code integrity enforcement

- Even with a compromised kernel!

Massive attack surface reduction

Low overhead

- No hypervisor traps
- No nested page tables

With code integrity protected, how do we protect control flow?

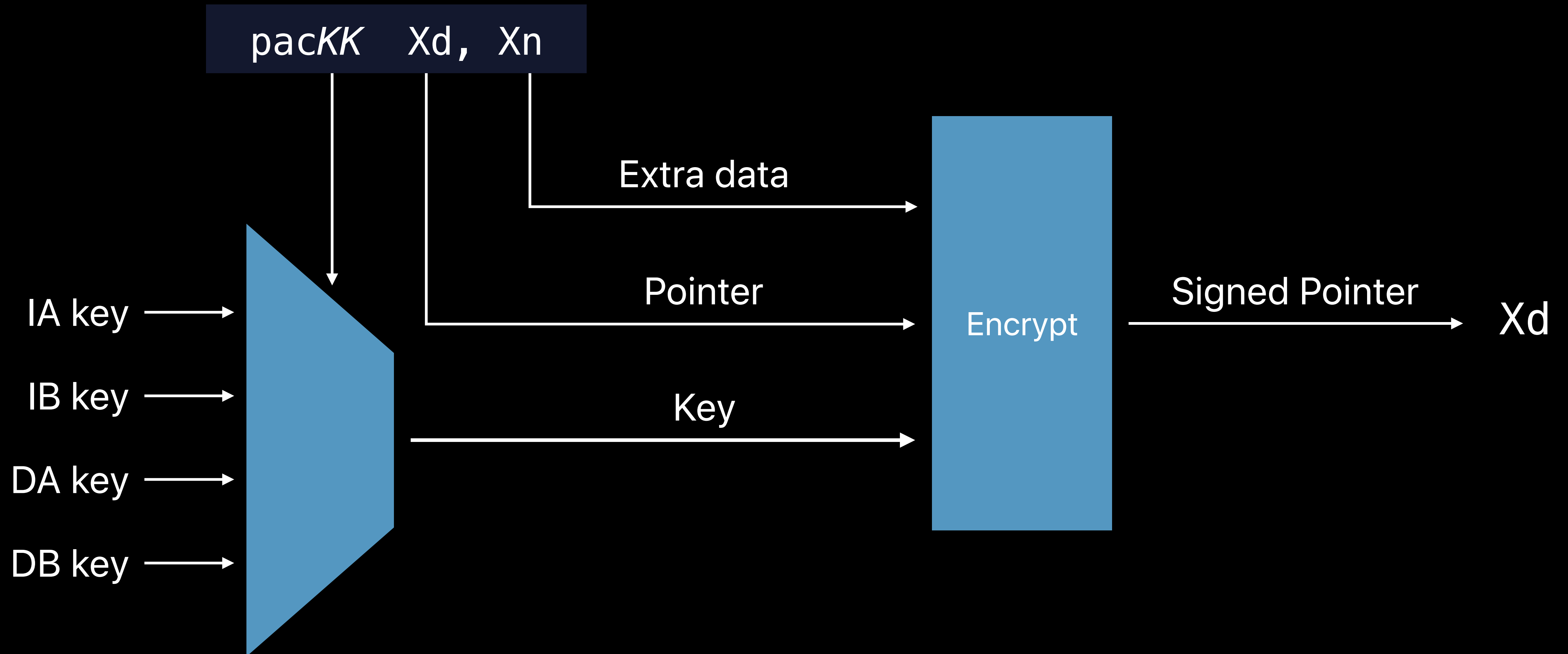
Pointer Authentication

New instructions in ARMv8.3

Uses spare bits in pointers to store a cryptographic hash

Designed to be robust in the presence of arbitrary read/write primitives

Pointer Authentication Instructions



Pointer Authentication

Sign

000000100a41238



Pointer

Pointer Authentication

Sign

000000100a41238

Padding Address

Pointer Authentication

Sign



7b9352e100a41238



Signature



Address

Pointer Authentication

Authenticate

7b9352e100a41238



Signature



Address

Pointer Authentication

Authenticate



000000100a41238

Padding Address

Pointer Authentication

Auth failure

7b9352f100a41238

Signature Address

Pointer Authentication

Auth failure



200000100a41238



Padding



Address

Pointer Authentication

Keys

5 secret 128-bit values

- IA, IB, DA, DB, and GA keys
- I keys for instructions, D keys for data
- GA key for data MAC

Randomly generated

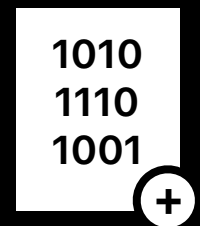
- At boot (A keys)
- At process creation (B keys)

Can't be read by attacker



Pointer Authentication

Pointers to code



	I	B	Storage Address
Function Return Address	I	B	Storage Address
Function Pointers	I	A	0
Block Invocation Function	I	A	Storage Address
Objective-C Method Cache	I	B	Storage Address + Class + Selector
C++ V-Table Entries	I	A	Storage Address + Hash(mangled method name)
Computed Goto Label	I	A	Hash(function name)

Pointer Authentication

Function return address before PAC

```
_func:
```

```
    stp x29, x30, [sp, #-16]!
```

```
    ...
```

```
    ldp x29, x30, [sp], #16
```

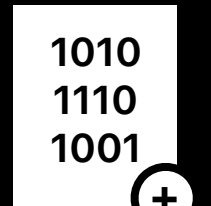
```
    ret
```

Pointer Authentication

Function return address after PAC



IB
Code
Process



Storage
Address

```
_func:
```

```
    pacibsp
```

```
    stp x29, x30, [sp, #-16]!
```

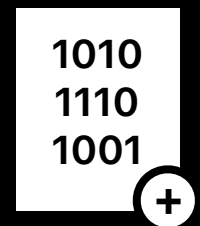
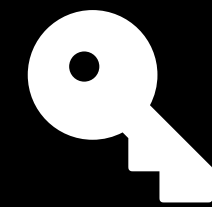
```
    ...
```

```
    ldp x29, x30, [sp], #16
```

```
    retab
```

Pointer Authentication

Pointers to data, code via data



Kernel Thread State

G

A

*

User Thread State Registers

I

A

Storage Address

C++ V-Table Pointers

D

A

0

Pointer Authentication

Improvements in iOS 13

Abort on all authentication failures
in kernel

Adoption across all Apple kexts

Hardened jump tables



Pointer Authentication

Improvements in iOS 13

ObjC method dispatch hardening

- Sign and authenticate IMP pointers in method cache tables

Hardened exception handling

- Hash and verify sensitive register state

JavaScriptCore JIT and extra data hardening



Pointer Authentication

Coming soon

Authenticated members of high value data structures

- Processes, tasks
- Codesigning
- Virtual Memory subsystem
- IPC structures



Mac secure boot

iOS code integrity protection

Find My

Helping users find lost devices, even when offline

Any device in proximity can help, even if stranger to the owner

Offline device communicates via Bluetooth with participating strangers (finders)

Finders report their location and a timestamp

Owner uses a second device to find the lost device

Challenges

A static device identifier makes the device trackable

Even with a rotated identifier, finder can't encrypt location end-to-end

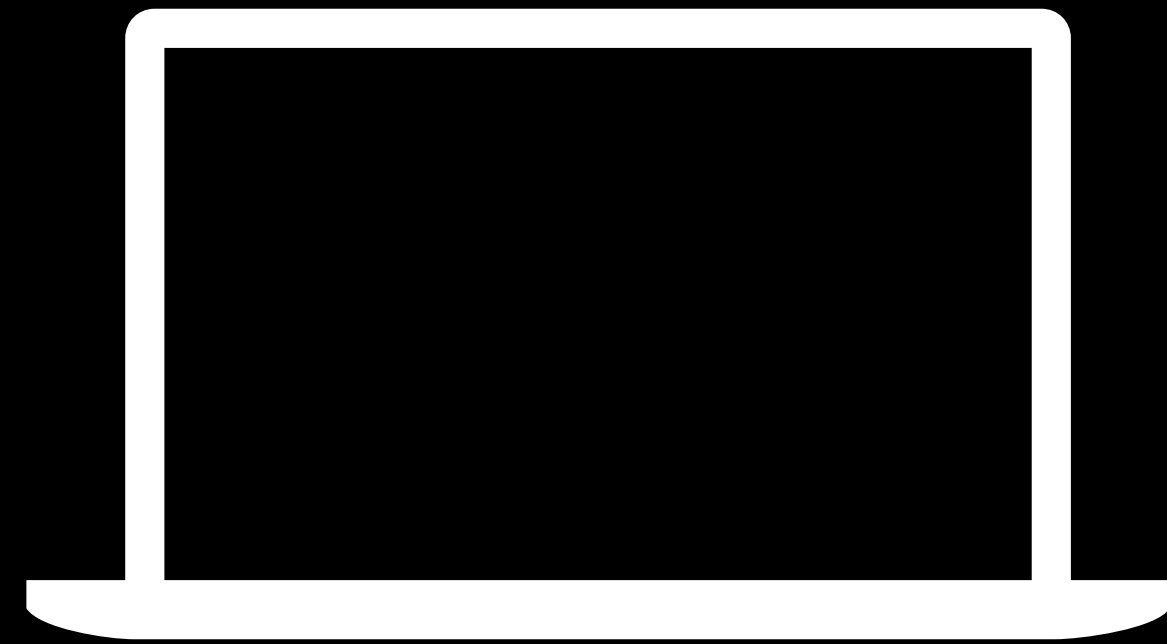
- Server would have access to the location information

Security and Privacy Goals

Protect owners, finders, and devices

- Location reports are not accessible to Apple servers
 - Cannot read, modify, or even add bogus reports
- Finder identities and location not revealed to Apple servers
 - No finder identifier recorded
 - Reported location is encrypted
- Information broadcasted by the lost device cannot be used to track it, except by the owner

Find My Setup



Encrypted $\{d, P, SK_0\}$
in iCloud Keychain



Generate EC P-224 key pair $\{d, P = d \cdot G\}$

Generate symmetric key SK_0

Store $\{d, P, SK_0\}$ in iCloud Keychain

Find My

Device broadcasting its location

A Find My time period, i , is 15 minutes long

Derive symmetric key SK_i

- $SK_i = \text{KDF}(SK_{i-1}, \text{"update"})$

Derive anti-tracking secret pair (u_i, v_i)

- $(u_i, v_i) = \text{KDF}(SK_i, \text{"diversify"})$

Unlinkably diversify public key P

- $P_i = u_i \cdot P + v_i \cdot G$

Broadcast P_i to nearby finders

Find My

Reporting location of a broadcasting device

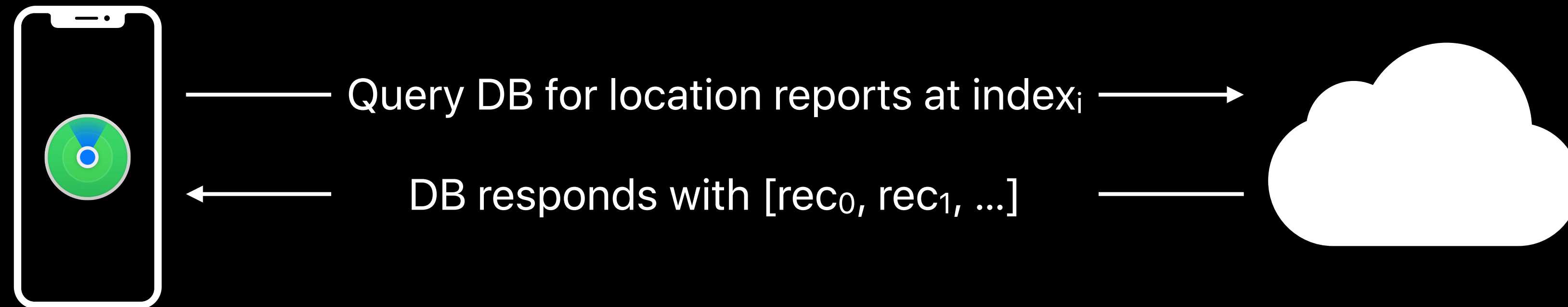
Finder ECIES-encrypts its location to public key P_i

Computes lookup $\text{index}_i = \text{SHA256}(P_i)$

Uploads encrypted report with index_i to Apple servers

Find My

Owner locating their device



Retrieve d_i from iCloud Keychain

Compute $P_i = d_i \cdot G$ for lookup period i

Compute lookup index _{i} = Hash(P_i)

ECIES decrypt $(pos_{i,0}, time_{i,0}) = D(d_i, rec_0)$

Find My Summary

Novel design to enable users to enlist the help of strangers to locate lost devices

Highly rigorous privacy properties to protect participating device owners and finders

Mac secure boot

iOS code integrity protection

Find My

Mac secure boot

iOS code integrity protection

Find My

Apple Security Bounty

Introduced in 2016

Platforms

iOS, iCloud

Categories

5

Participation

Very small invited
researcher audience

Maximum payout

\$200,000

50 High-Value Reports

What's next?

Apple Security Bounty will be open to
all researchers

iCloud

iOS

tvOS

iPadOS

watchOS

macOS

Revised and expanded categories

		Maximum Payout
Unauthorized access to iCloud account data on Apple servers		\$100,000
Attack via physical access	Lock screen bypass	\$100,000
	User data extraction	\$250,000
Attack via user-installed app	Unauthorized access to high-value user data	\$100,000
	Kernel code execution	\$150,000
	CPU side channel attack on high-value user data	\$250,000
Network attack requiring user interaction	One-click unauthorized access to high-value user data	\$150,000
	One-click kernel code execution	\$250,000
Network attack with no user interaction	Zero-click radio to kernel with physical proximity	\$250,000
	Zero-click access to high-value user data	\$500,000

Vulnerabilities in designated pre-
release builds

50%

bonus

What about getting started?

Making It Easier to Get Started with iOS Research

We want to attract exceptional researchers who have been focused on other platforms

New researchers shouldn't have to find a full chain to bootstrap research

Existing iOS researchers shouldn't have to hold back chains for research

iOS Security Research Device Program

iOS Security Research Device program

Unprecedented, Apple-supported iOS security research platform

Comes with ssh, a root shell, and advanced debug capabilities

New research fusing, neither production nor development

iOS Security Research Device program

Unprecedented, Apple-supported iOS security research platform

Comes with ssh, a root shell, and advanced debug capabilities

New research fusing, neither production nor development

Program applications open to everyone with a track record of high-quality systems security research on any platform

iOS Security Research Device program

Unprecedented, Apple-supported iOS security research platform

Comes with ssh, a root shell, and advanced debug capabilities

New research fusing, neither production nor development

Program applications open to everyone with a track record of high-quality systems security research on any platform

Coming next year

Apple Security Bounty Summary

Participation open to all researchers in the Fall

Expanded and revised categories

Highest maximum payouts in the industry

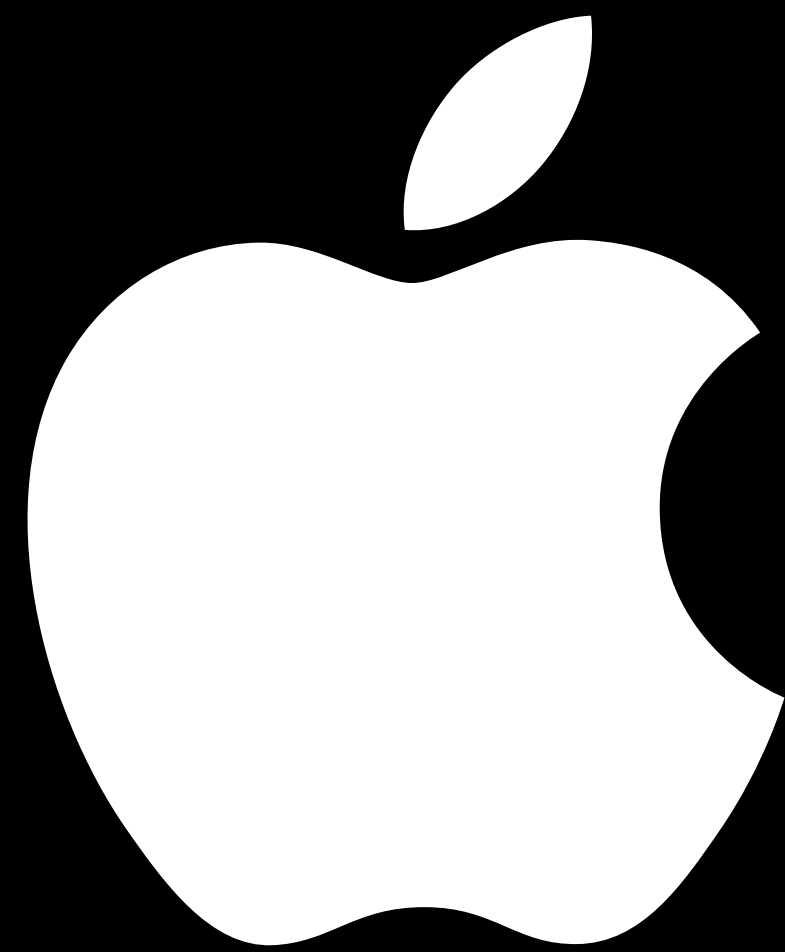
iOS Security Research Device Program for exceptional researchers new to our platform

What about a zero-click iOS full chain
with kernel code execution and
persistence?

\$1,000,000

	Maximum Payout
Unauthorized access to iCloud account data on Apple servers	\$100,000
Attack via physical access	Lock screen bypass \$100,000
	User data extraction \$250,000
Attack via user-installed app	Unauthorized access to high-value user data \$100,000
	Kernel code execution \$150,000
	CPU side channel attack on high-value user data \$250,000
Network attack requiring user interaction	One-click unauthorized access to high-value user data \$150,000
	One-click kernel code execution \$250,000
Network attack with no user interaction	Zero-click radio to kernel with physical proximity \$250,000
	Zero-click access to high-value user data \$500,000
	Zero-click kernel code execution with persistence \$1,000,000

We're excited to work with you!



TM and © 2019 Apple Inc. All rights reserved.