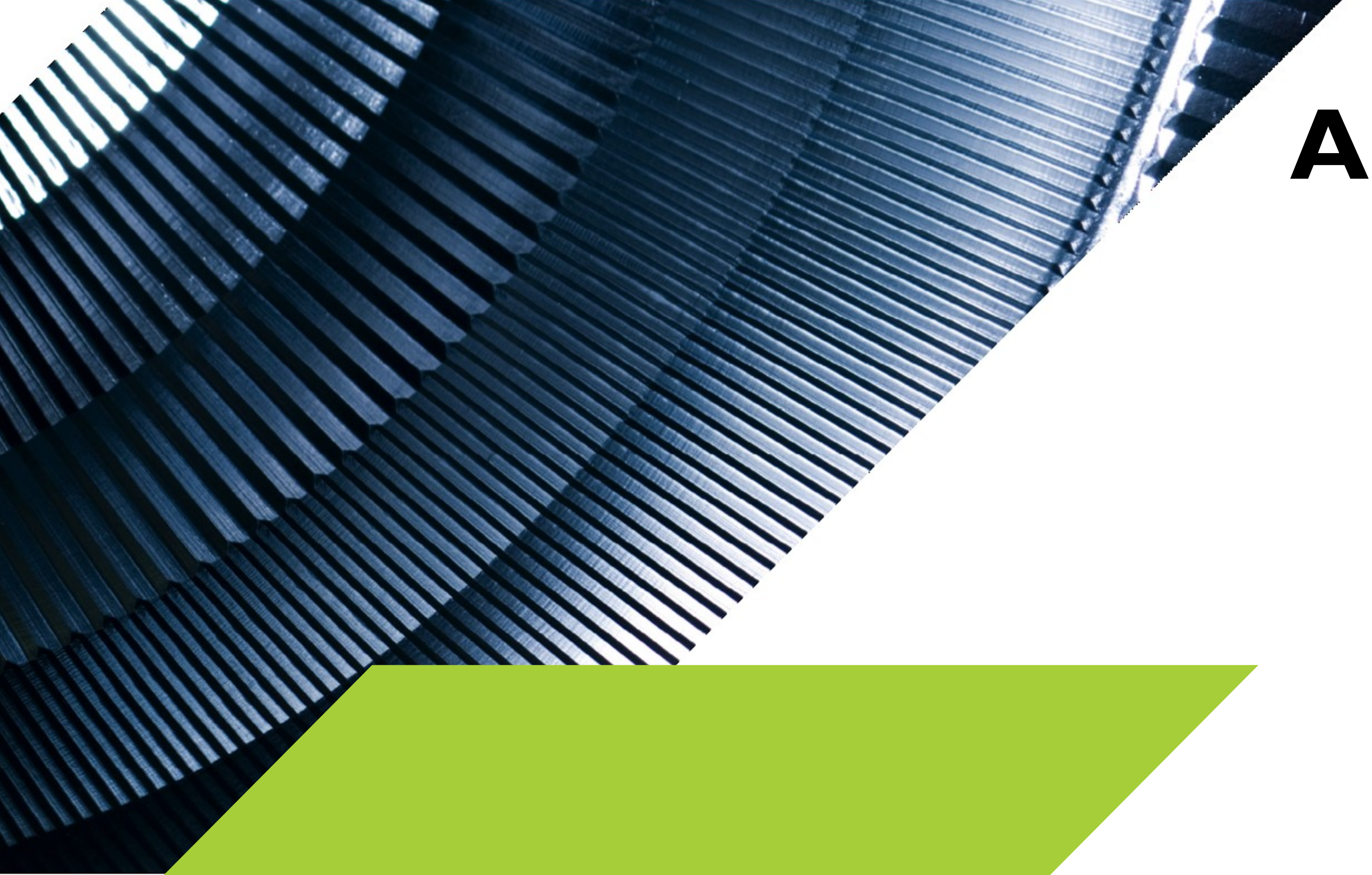


VIRTUALIZING IO THROUGH THE IO MEMORY MANAGEMENT UNIT (IOMMU)

ANDY KEGEL, PAUL BLINZER, ARKA BASU, MAGGIE CHAN

ASPLOS 2016

WHAT THIS TUTORIAL WILL AND WILL NOT COVER

- **Definition of “IO” or “Device” or “IO Device” :**
  - Traditional IO includes GPU for graphics, NIC, storage controller, USB controller, etc.
  - New IO (accelerators) includes general-purpose computation on a GPU (GPGPU), encryption accelerators, digital signal processors, etc.

- **Two Parts in Virtualizing an IO Device**
  - **Device specific: Virtual instances of device**
    - Virtual functions and Physical function in devices (PCIE® SR-IOV, MR-IOV)
  - **System defined: IO Memory Management Unit or IOMMU**
    - Virtualizing DMA accesses (Address Translation and Protection)
    - Virtualizing Interrupts (Interrupt Remapping and Virtualizing)

WHAT THIS TUTORIAL WILL AND WILL NOT COVER

AMD

- **Definition of “IO” or “Device” or “IO Device” :**
  - Traditional IO includes GPU for graphics, NIC, storage controller, USB controller, etc.
  - New IO (accelerators) includes general-purpose computation on a GPU (GPGPU), encryption accelerators, digital signal processors, etc.

- **Two Parts in Virtualizing an IO Device**
  - **System defined: IO Memory Management Unit or IOMMU**
    - Virtualizing DMA accesses (Address Translation and Protection)
    - Virtualizing Interrupts (Interrupt Remapping and Virtualizing)

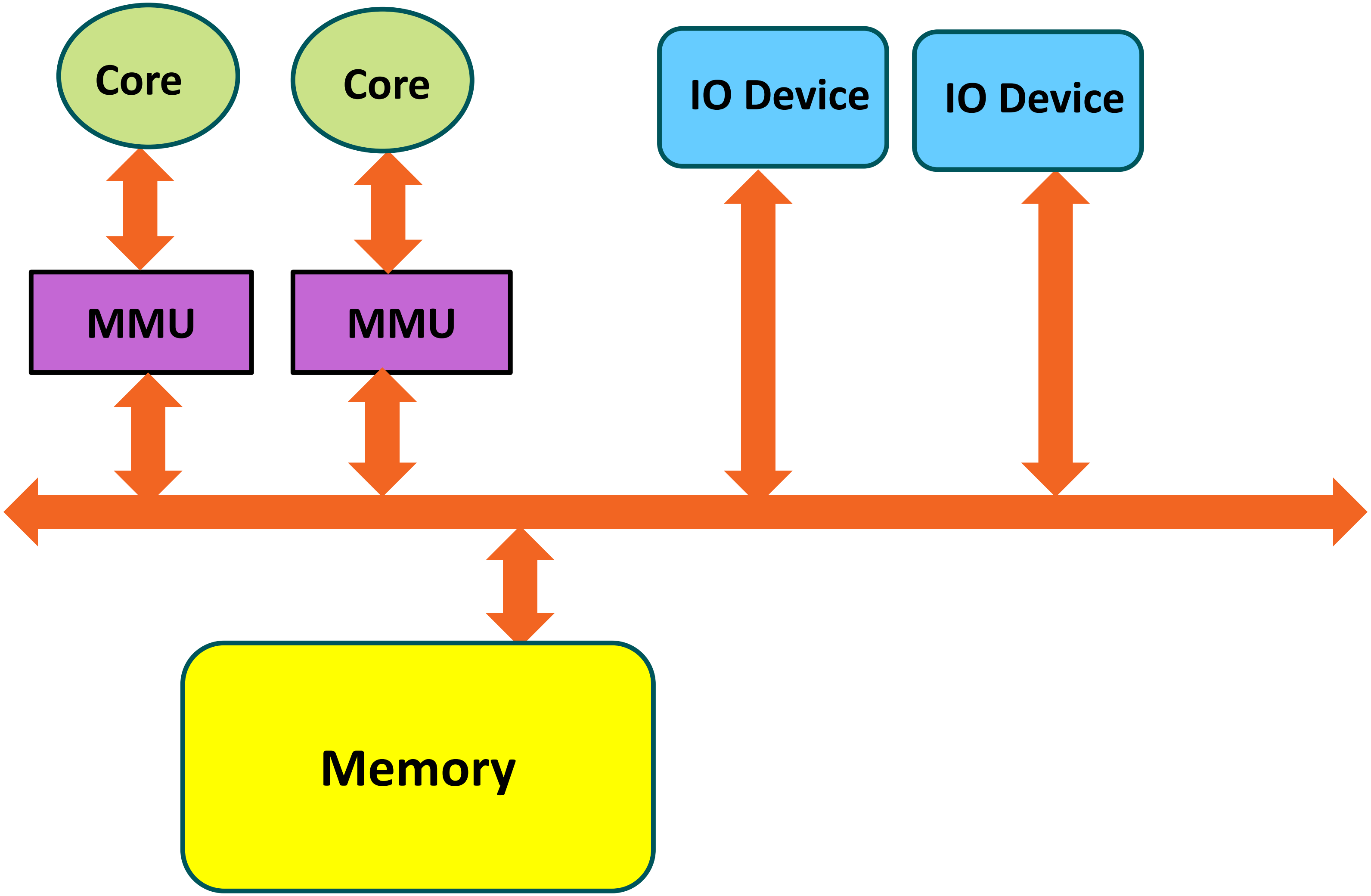
Focus

AGENDA

MOTIVATION & INTRODUCTION	What is IOMMU? -- *Andy Kegel*
USE CASES & DEMOSTRATION	Where can IOMMU help? -- *Paul Blinzer*
INTERNALS	How does IOMMU work? -- *Arka Basu, Maggie Chan*
RESEARCH	Research Opportunities and Discussion – *Arka Basu*

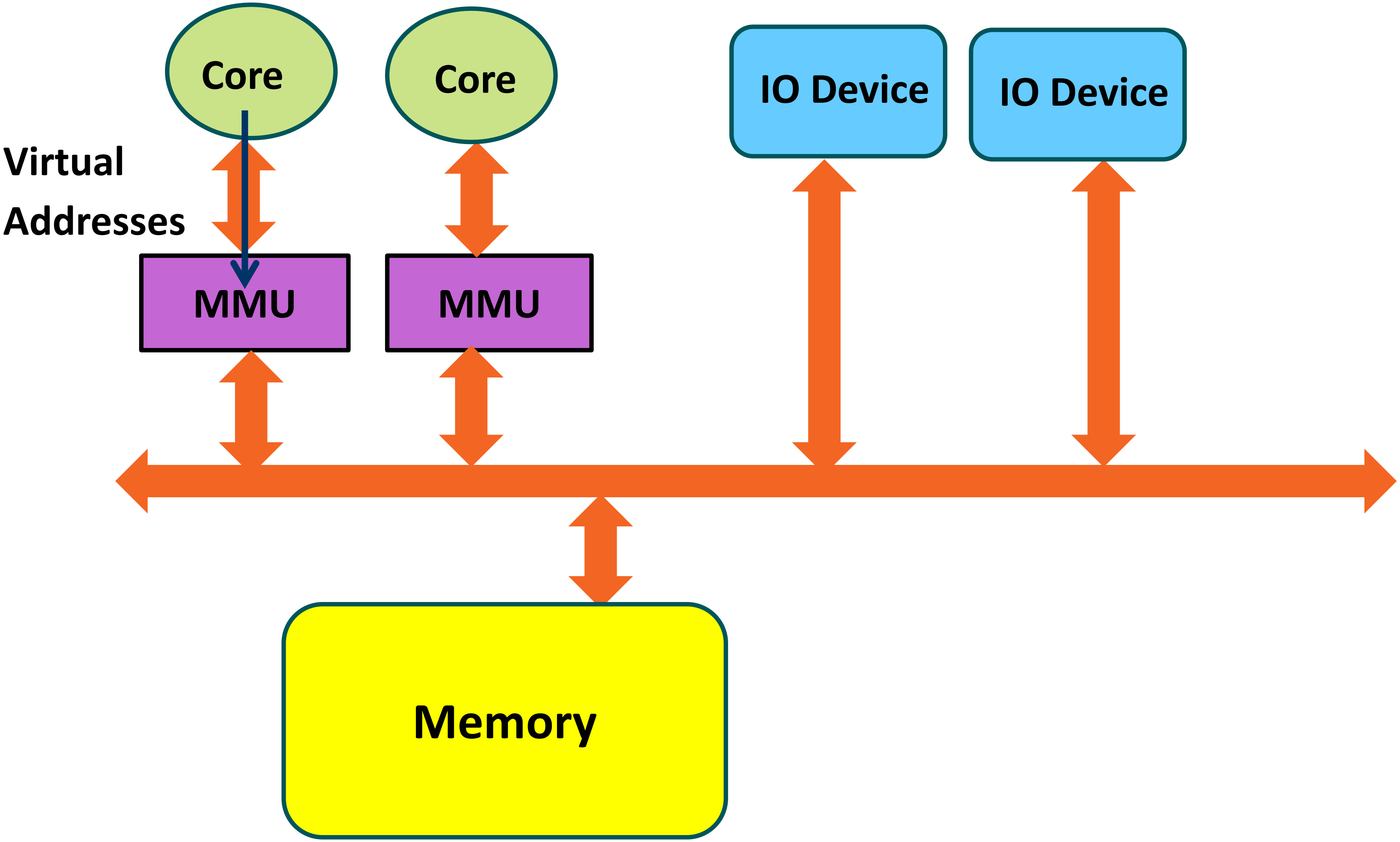
MOTIVATION: TRADITIONAL DMA BY IO

NO SYSTEM VIRTUALIZATION



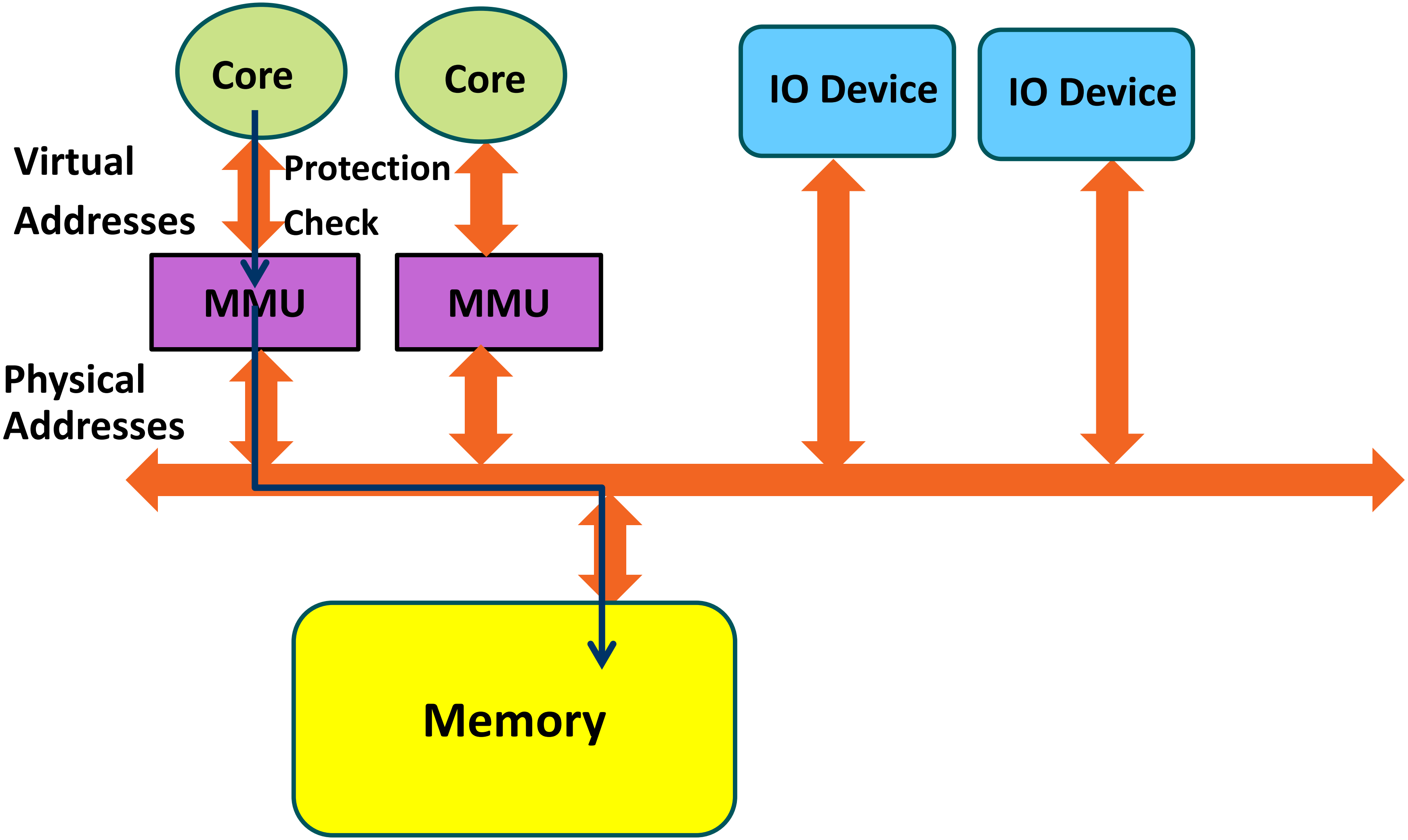
MOTIVATION: TRADITIONAL DMA BY IO

NO SYSTEM VIRTUALIZATION



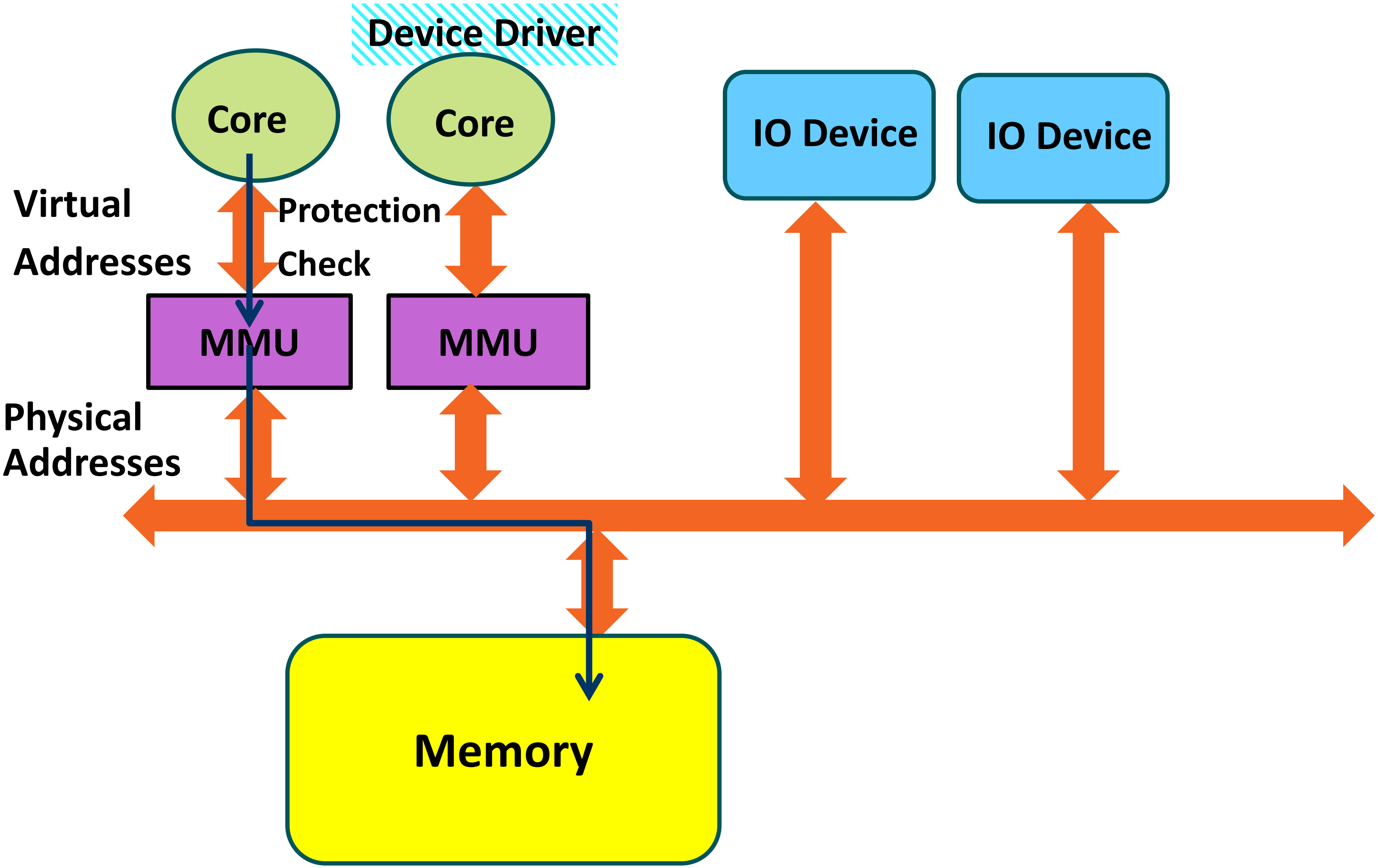
MOTIVATION: TRADITIONAL DMA BY IO

NO SYSTEM VIRTUALIZATION



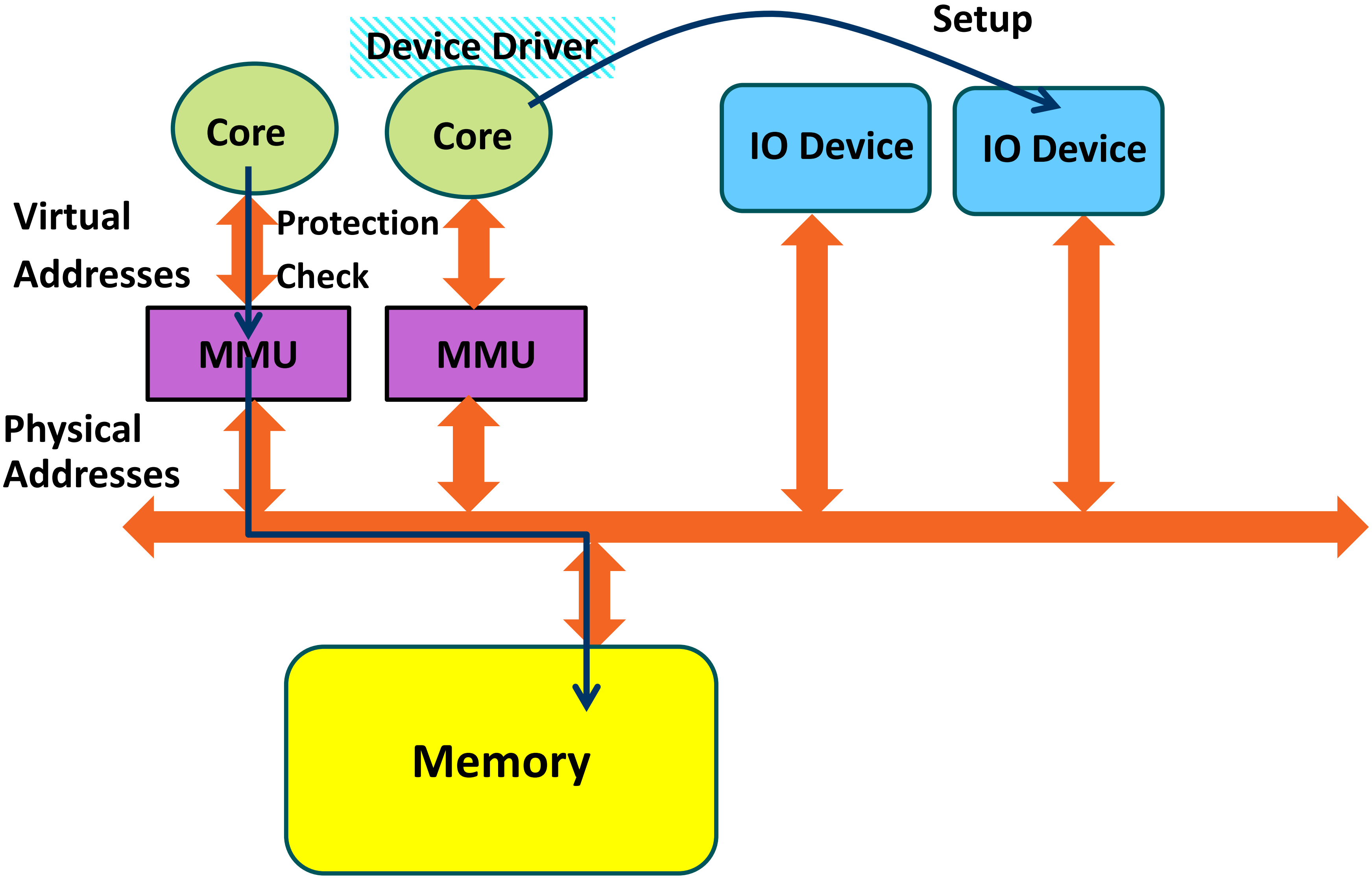
MOTIVATION: TRADITIONAL DMA BY IO

NO SYSTEM VIRTUALIZATION



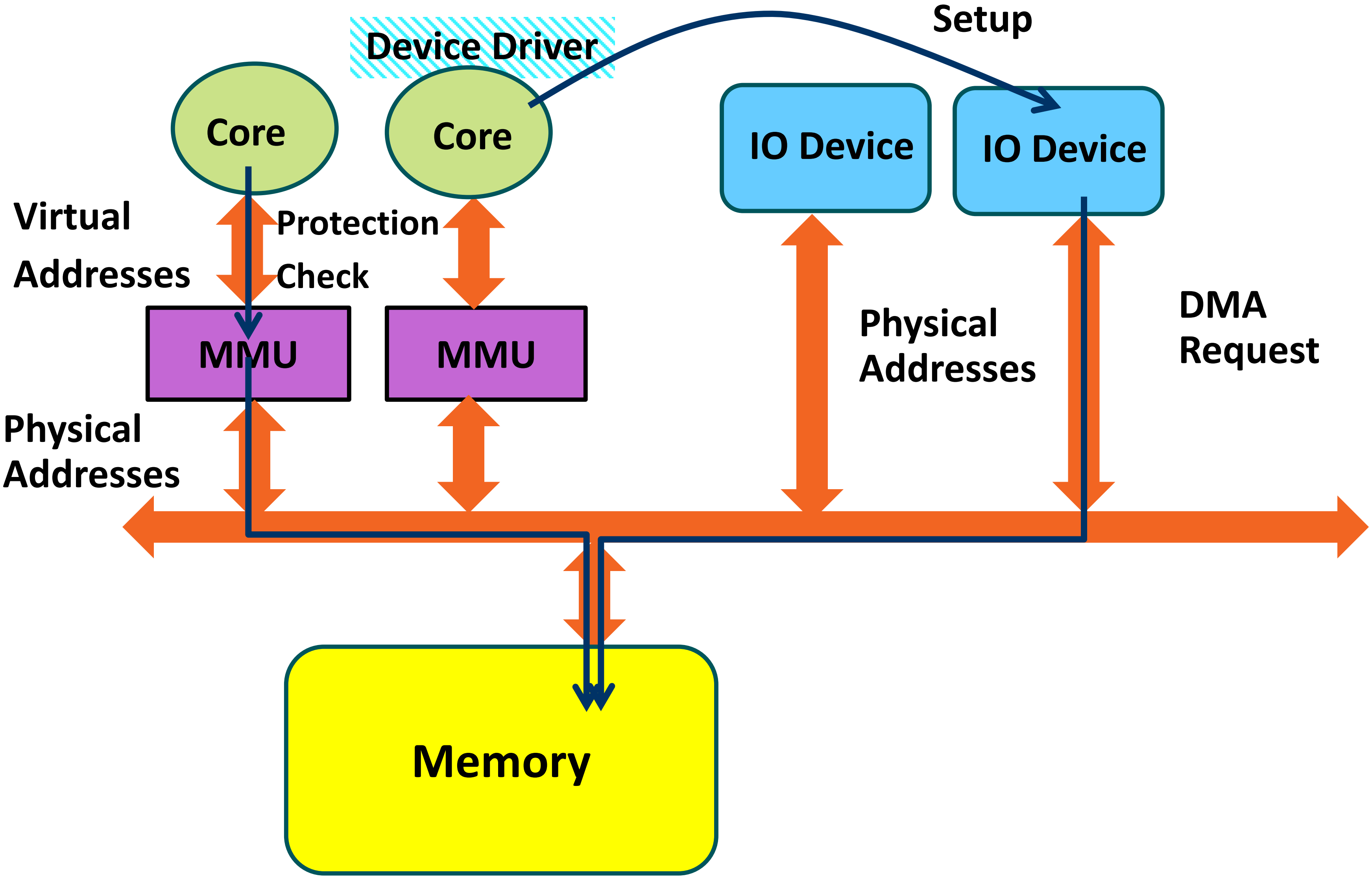
MOTIVATION: TRADITIONAL DMA BY IO

NO SYSTEM VIRTUALIZATION



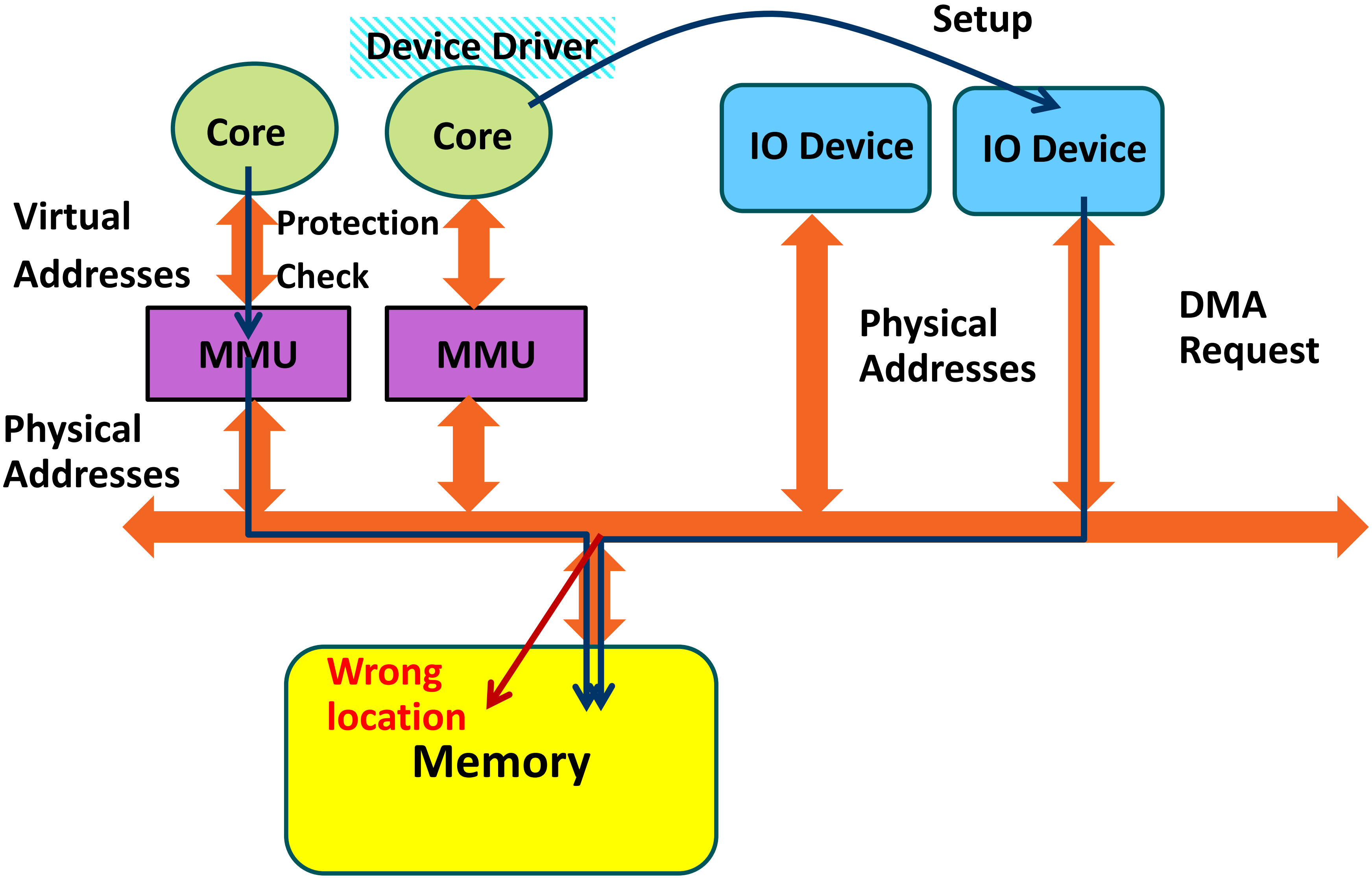
MOTIVATION: TRADITIONAL DMA BY IO

NO SYSTEM VIRTUALIZATION



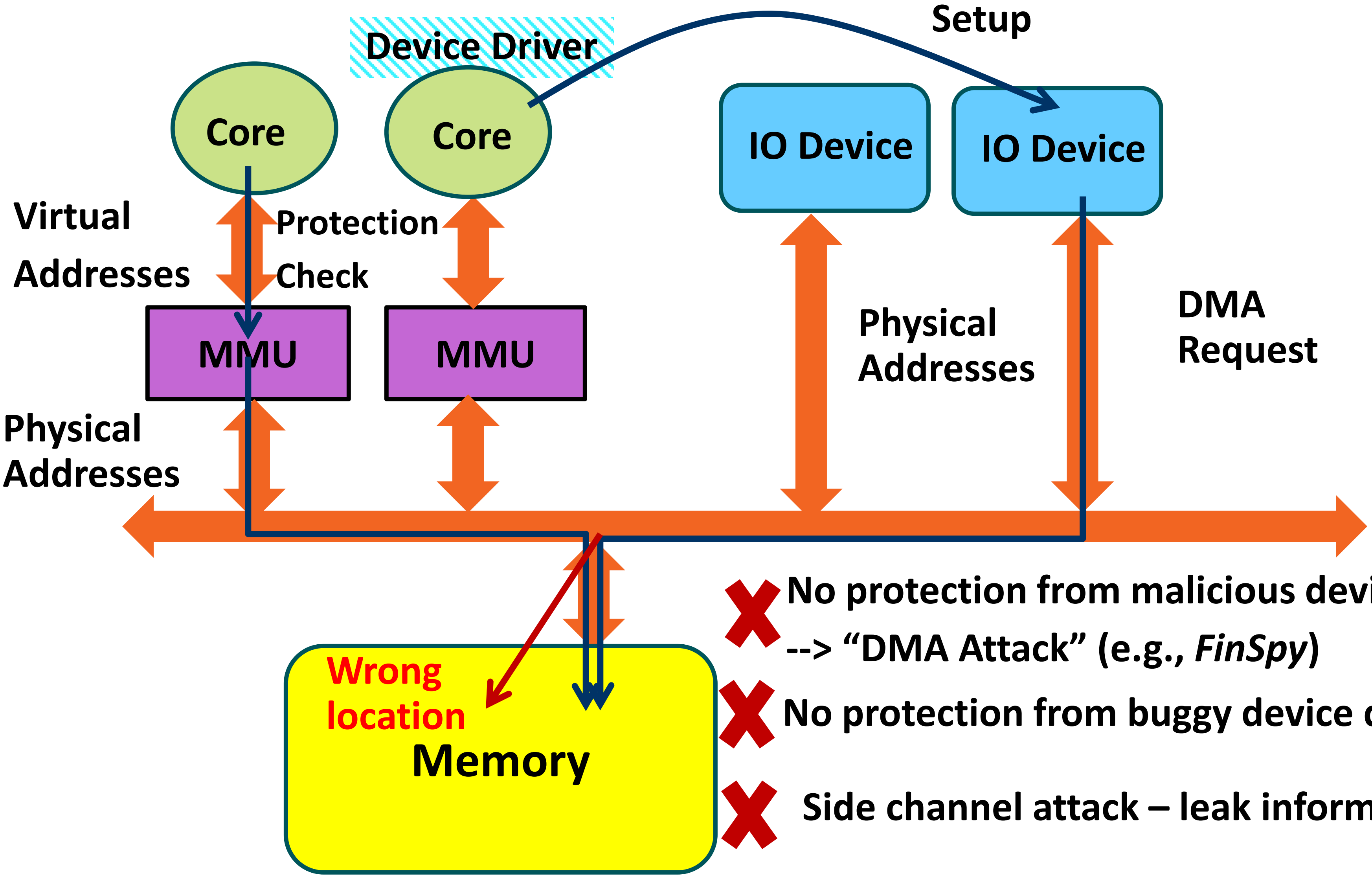
MOTIVATION: TRADITIONAL DMA BY IO

NO SYSTEM VIRTUALIZATION



MOTIVATION: TRADITIONAL DMA BY IO

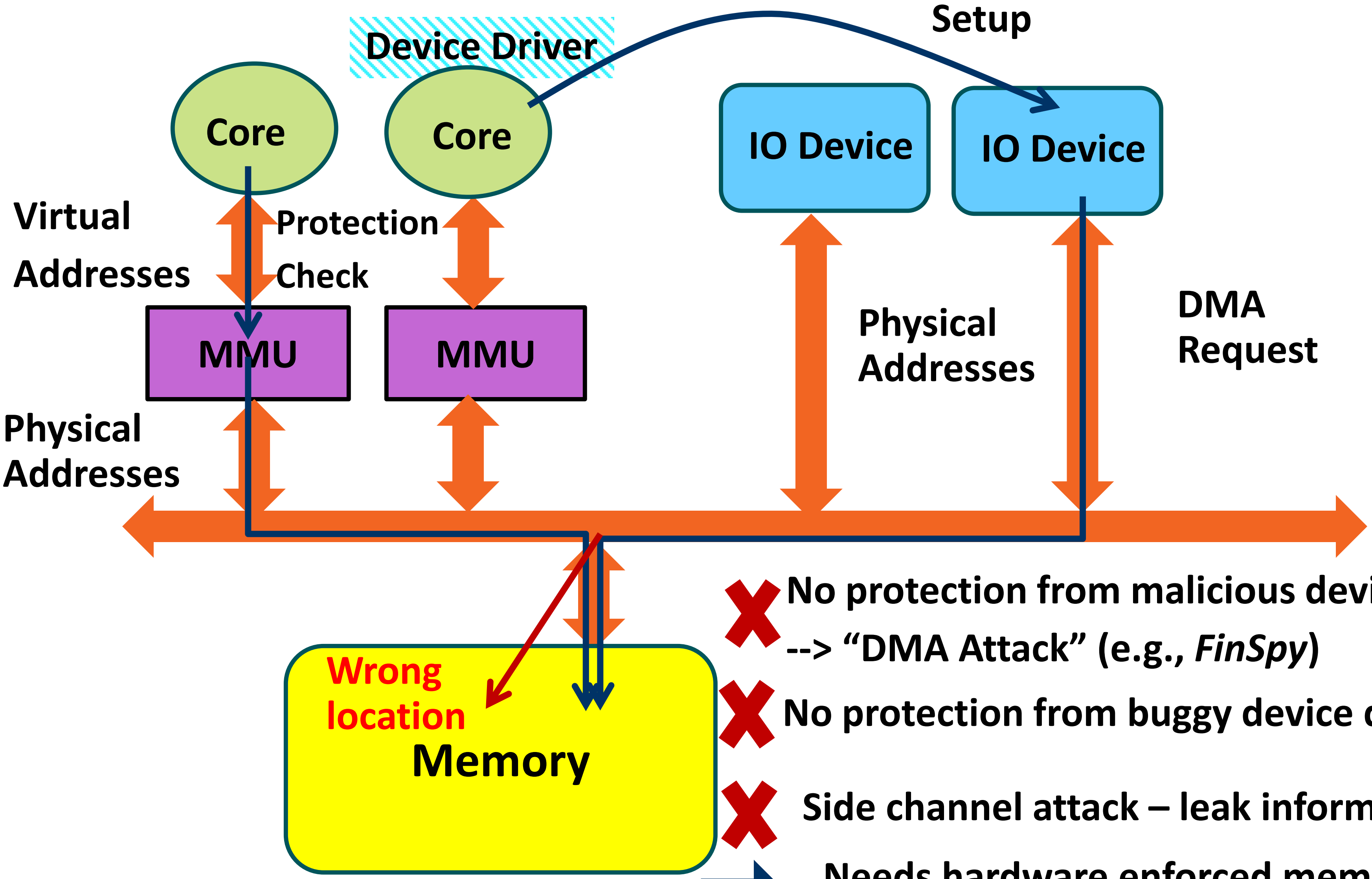
NO SYSTEM VIRTUALIZATION



- No protection from malicious devices --> “DMA Attack” (e.g., *FinSpy*)
- No protection from buggy device driver
- Side channel attack – leak information

MOTIVATION: TRADITIONAL DMA BY IO

NO SYSTEM VIRTUALIZATION

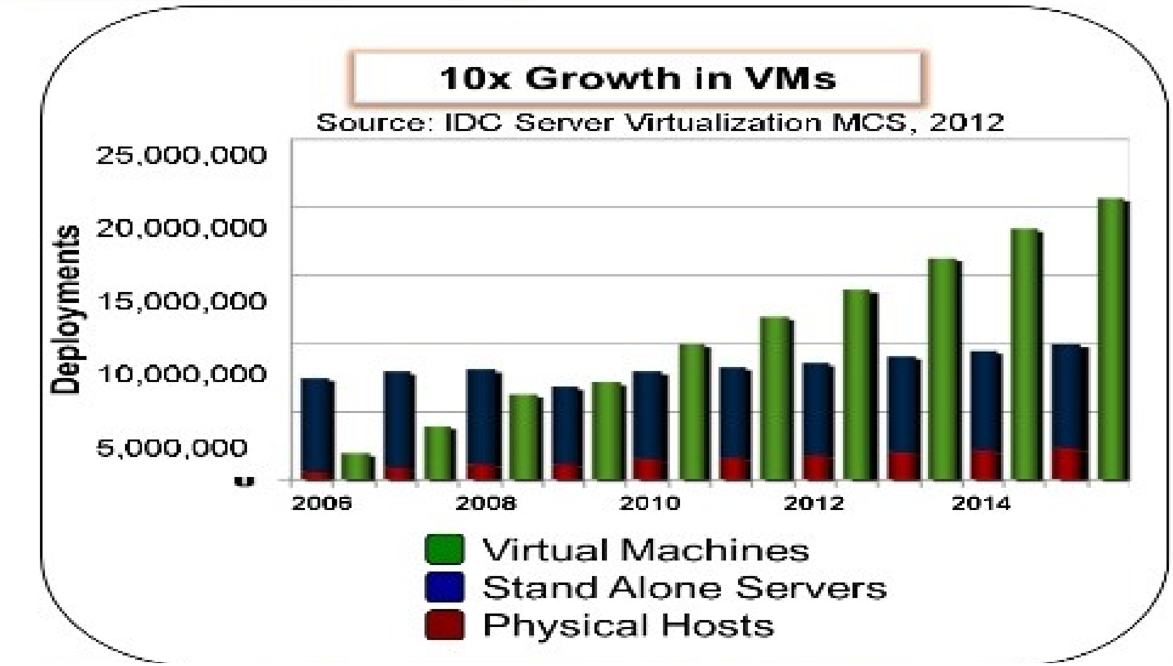


- No protection from malicious devices --> “DMA Attack” (e.g., *FinSpy*)
- No protection from buggy device driver
- Side channel attack – leak information

→ Needs hardware enforced memory protection

MOTIVATION: VIRTUAL MACHINES ARE TRENDING

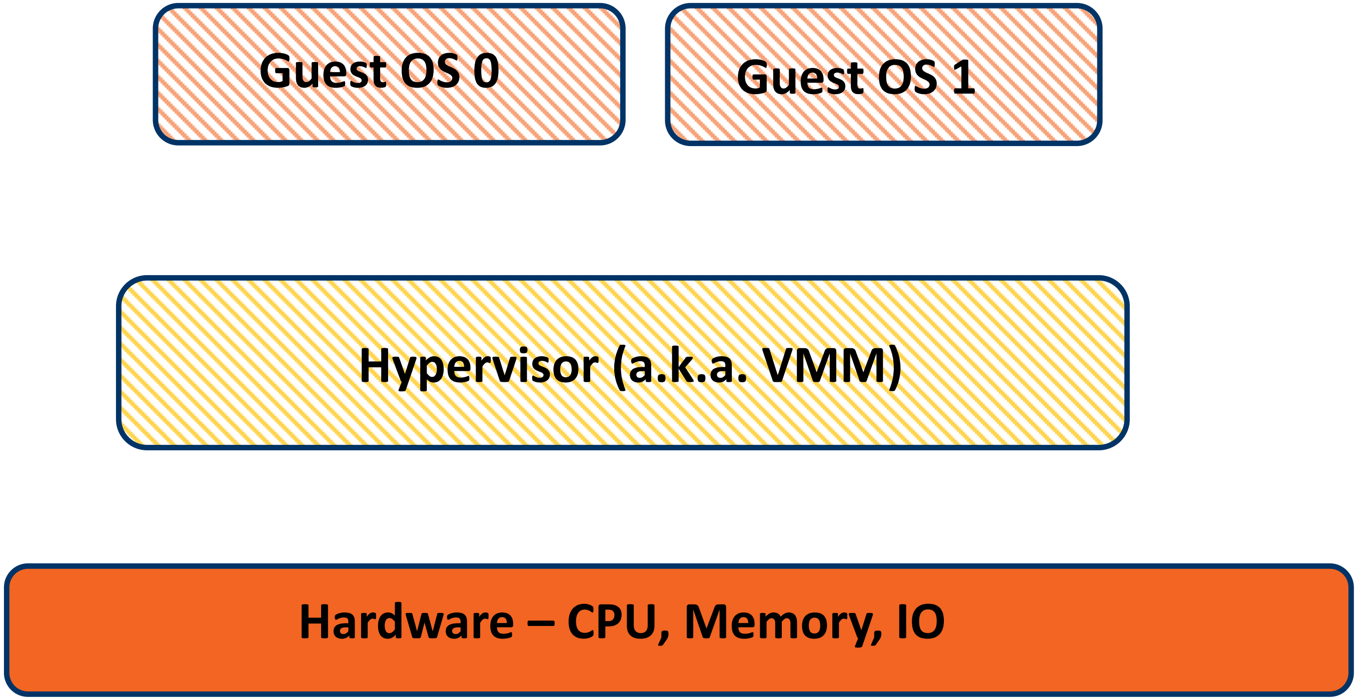
Tremendous growth in virtualization in server



Efficient access to IO under virtualization is important

Source: IDC Server Virtualization, MCS 2012

BACKGROUND: TRANSLATIONS IN VIRTUALIZED SYSTEM



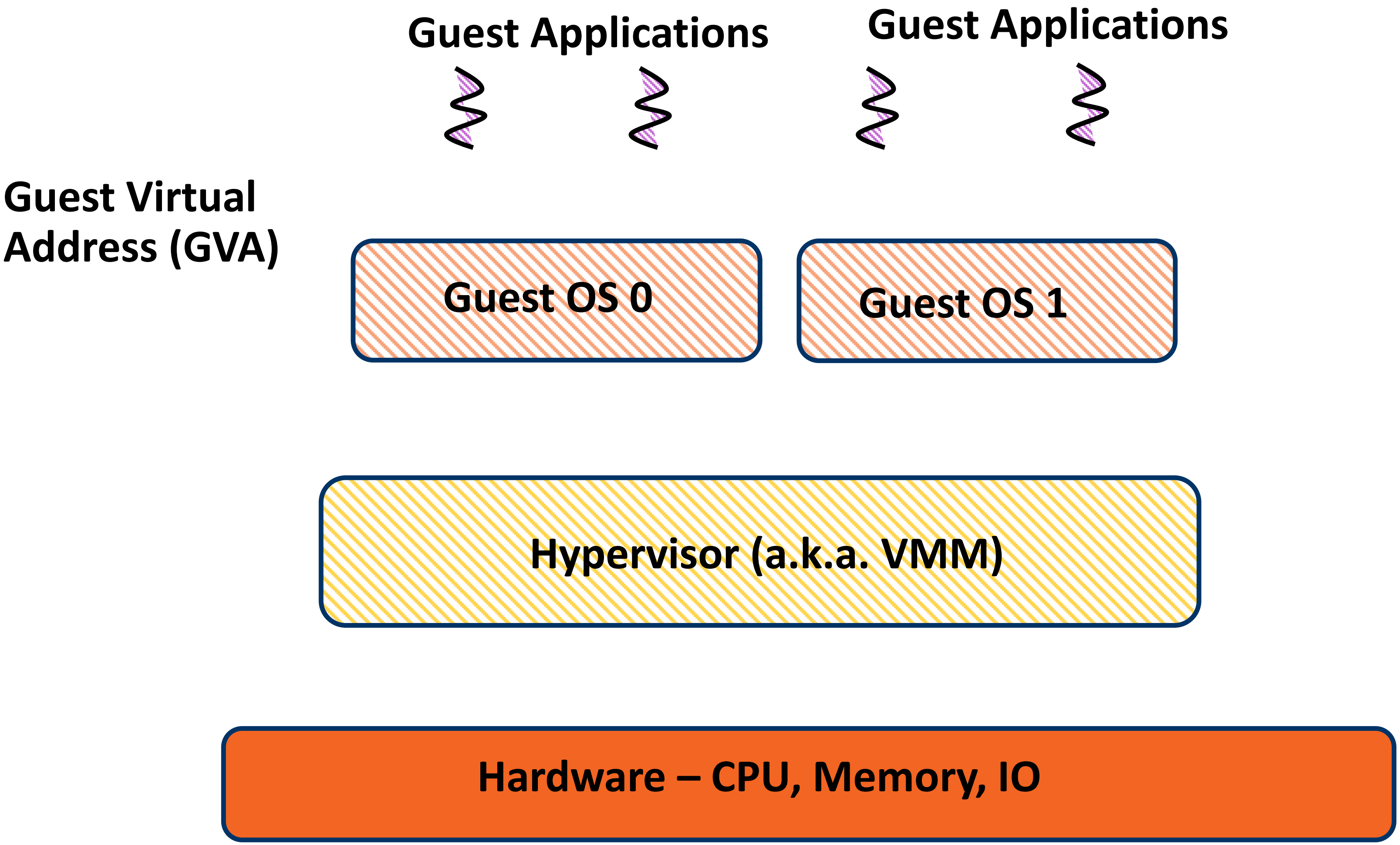
Guest OS 0

Guest OS 1

Hypervisor (a.k.a. VMM)

Hardware – CPU, Memory, IO

BACKGROUND: TRANSLATIONS IN VIRTUALIZED SYSTEM



Guest Applications

Guest Applications

Guest Virtual Address (GVA)

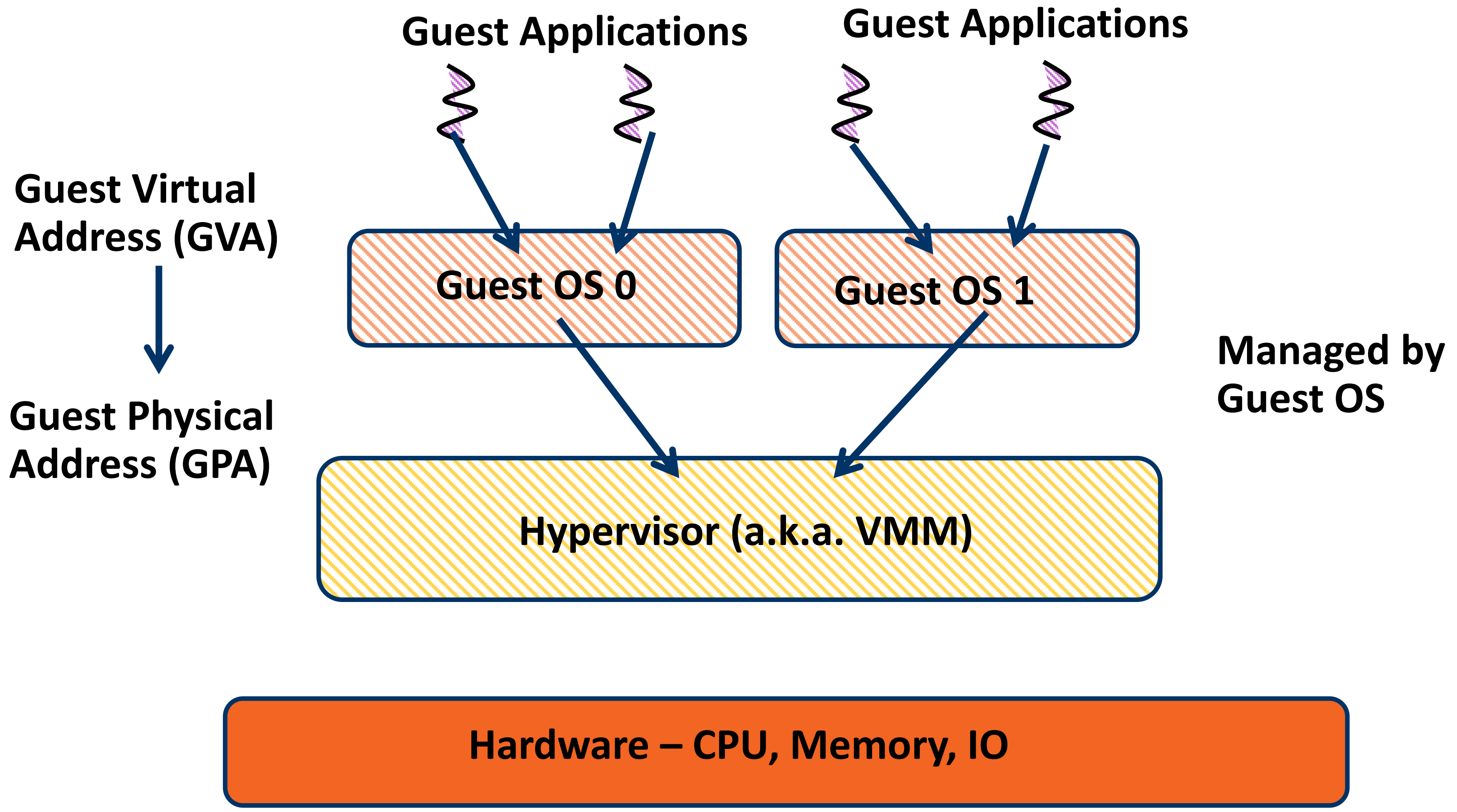
Guest OS 0

Guest OS 1

Hypervisor (a.k.a. VMM)

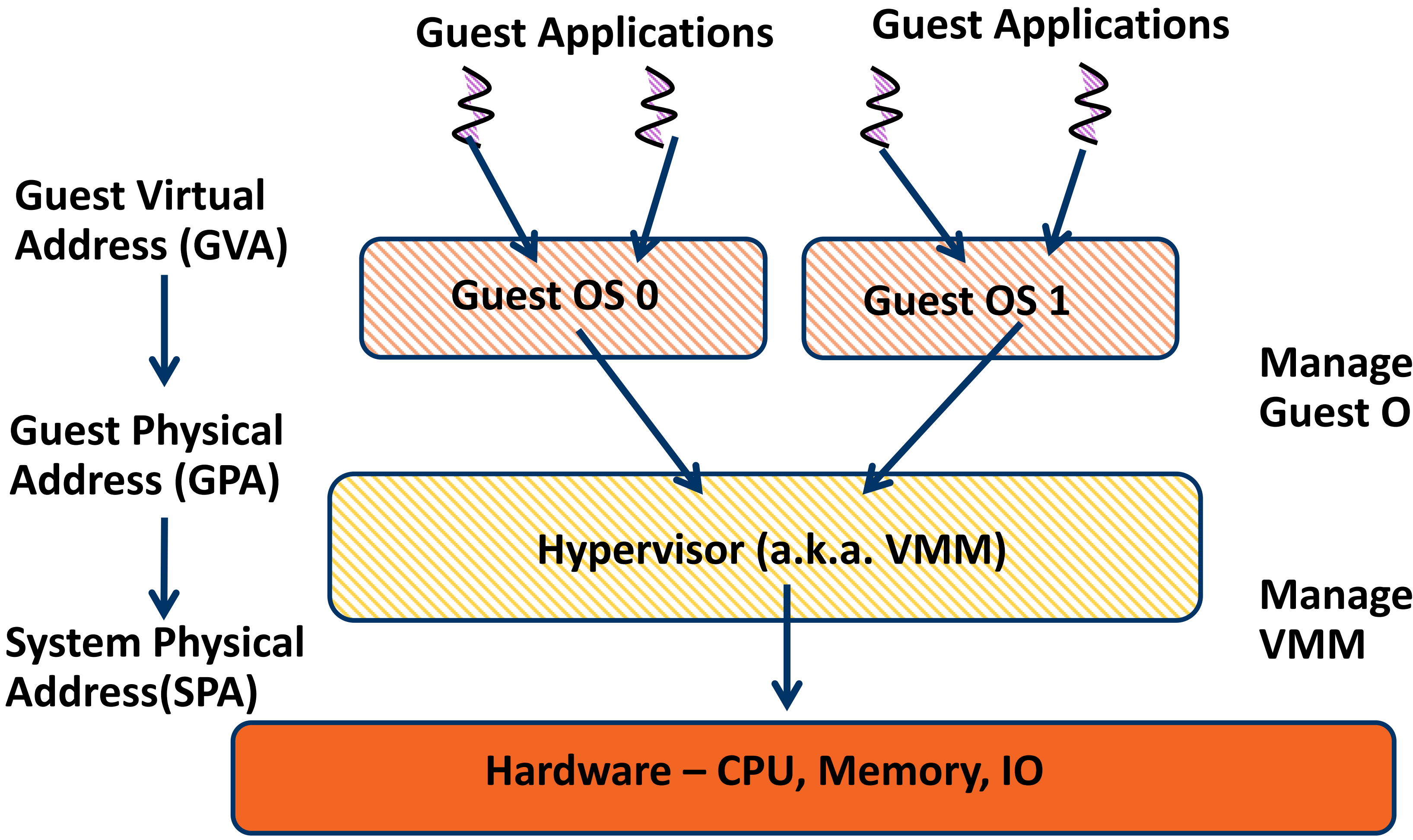
Hardware – CPU, Memory, IO

BACKGROUND: TRANSLATIONS IN VIRTUALIZED SYSTEM

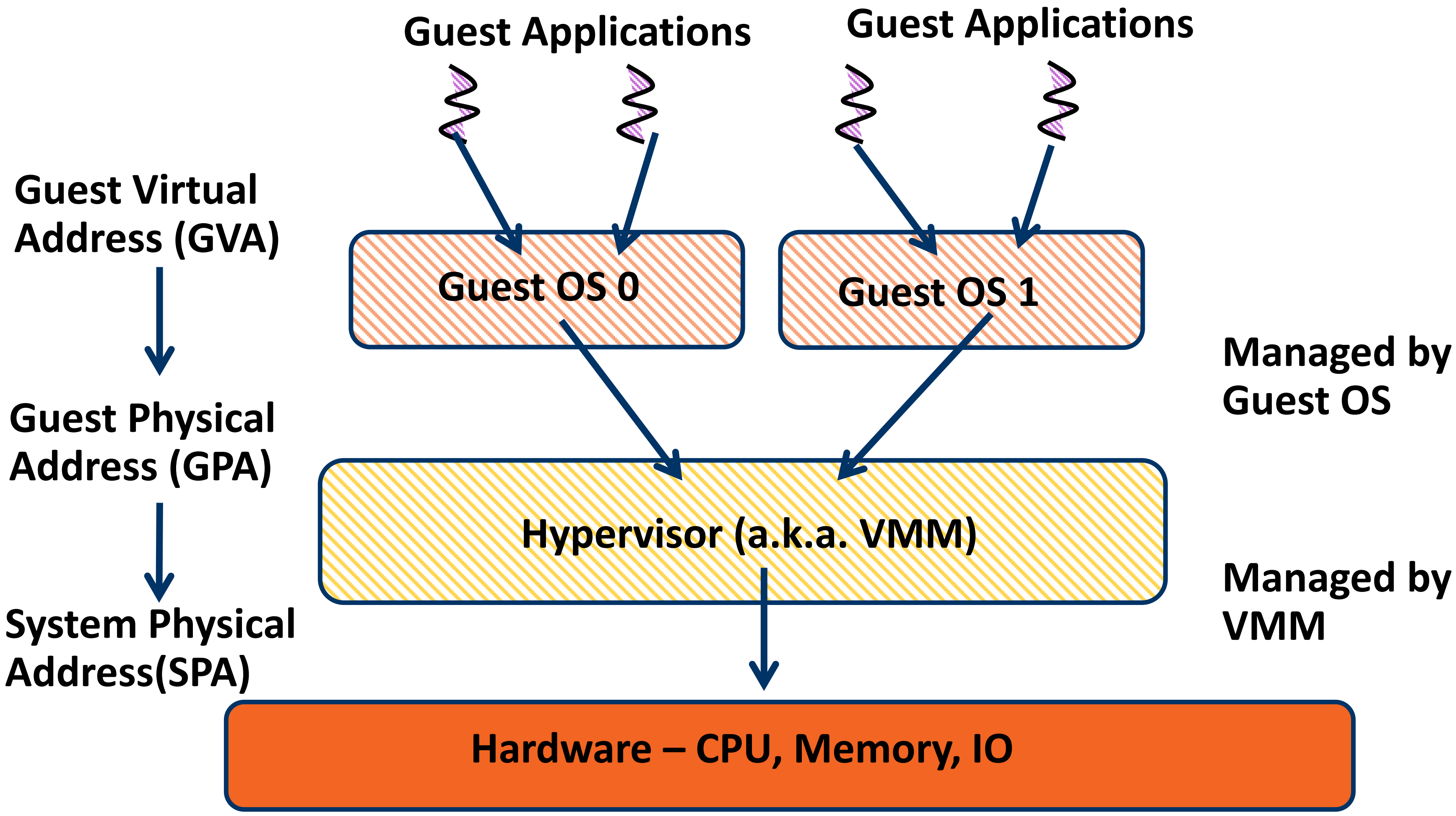


BACKGROUND: TRANSLATIONS IN VIRTUALIZED SYSTEM

AMD



BACKGROUND: TRANSLATIONS IN VIRTUALIZED SYSTEM



Guest Applications

Guest Applications

Guest Virtual Address (GVA)

Guest OS 0

Guest OS 1

Managed by Guest OS

Guest Physical Address (GPA)

Hypervisor (a.k.a. VMM)

Managed by VMM

System Physical Address(SPA)

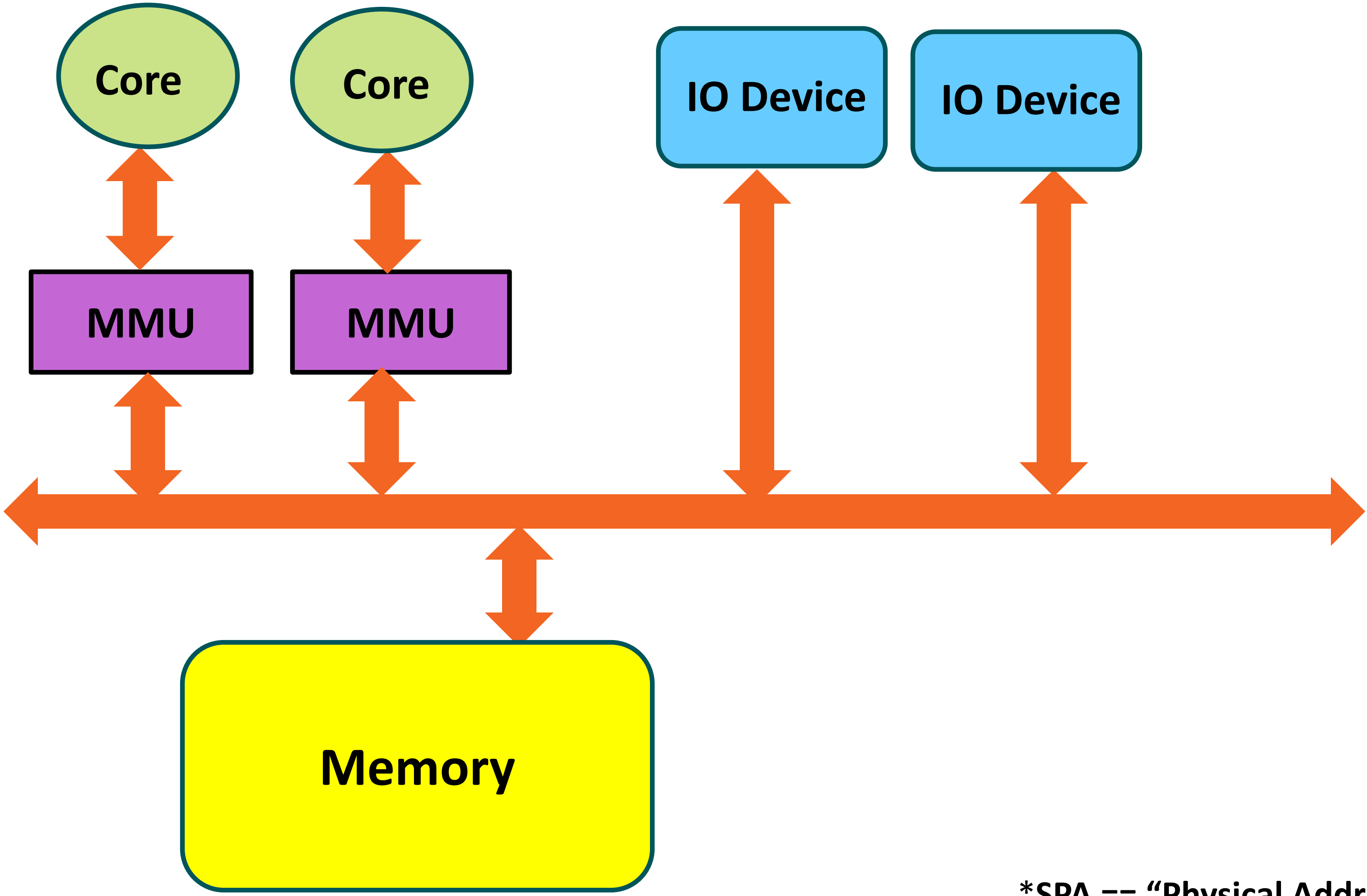
Hardware – CPU, Memory, IO

Isolation across Guest OS => No access to (system) physical address from Guest OS

MOTIVATION: TRADITIONAL DMA IN VIRTUAL MACHINES

AMD

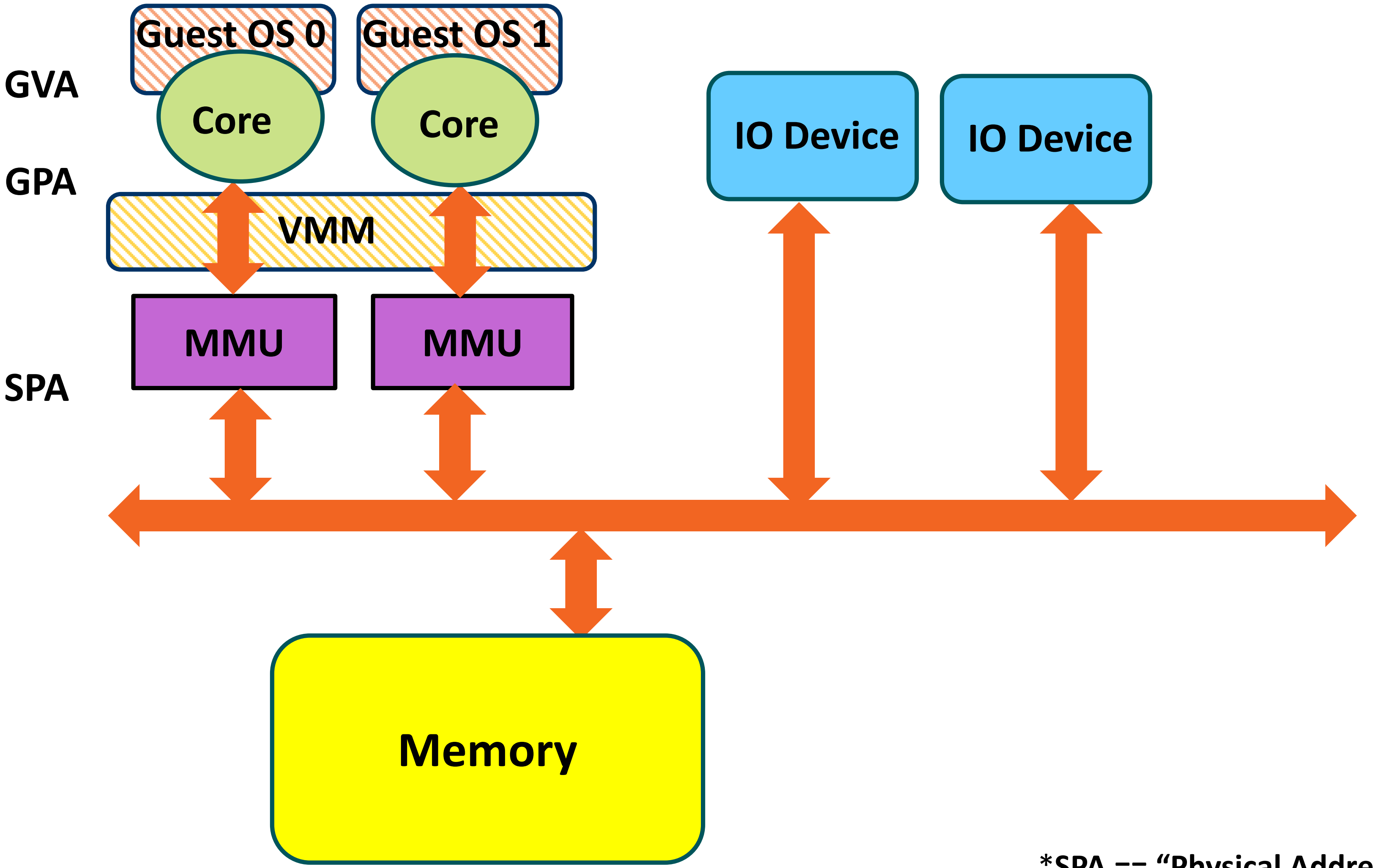
VIRTUALIZED SYSTEM



*SPA == “Physical Address”

MOTIVATION: TRADITIONAL DMA IN VIRTUAL MACHINES

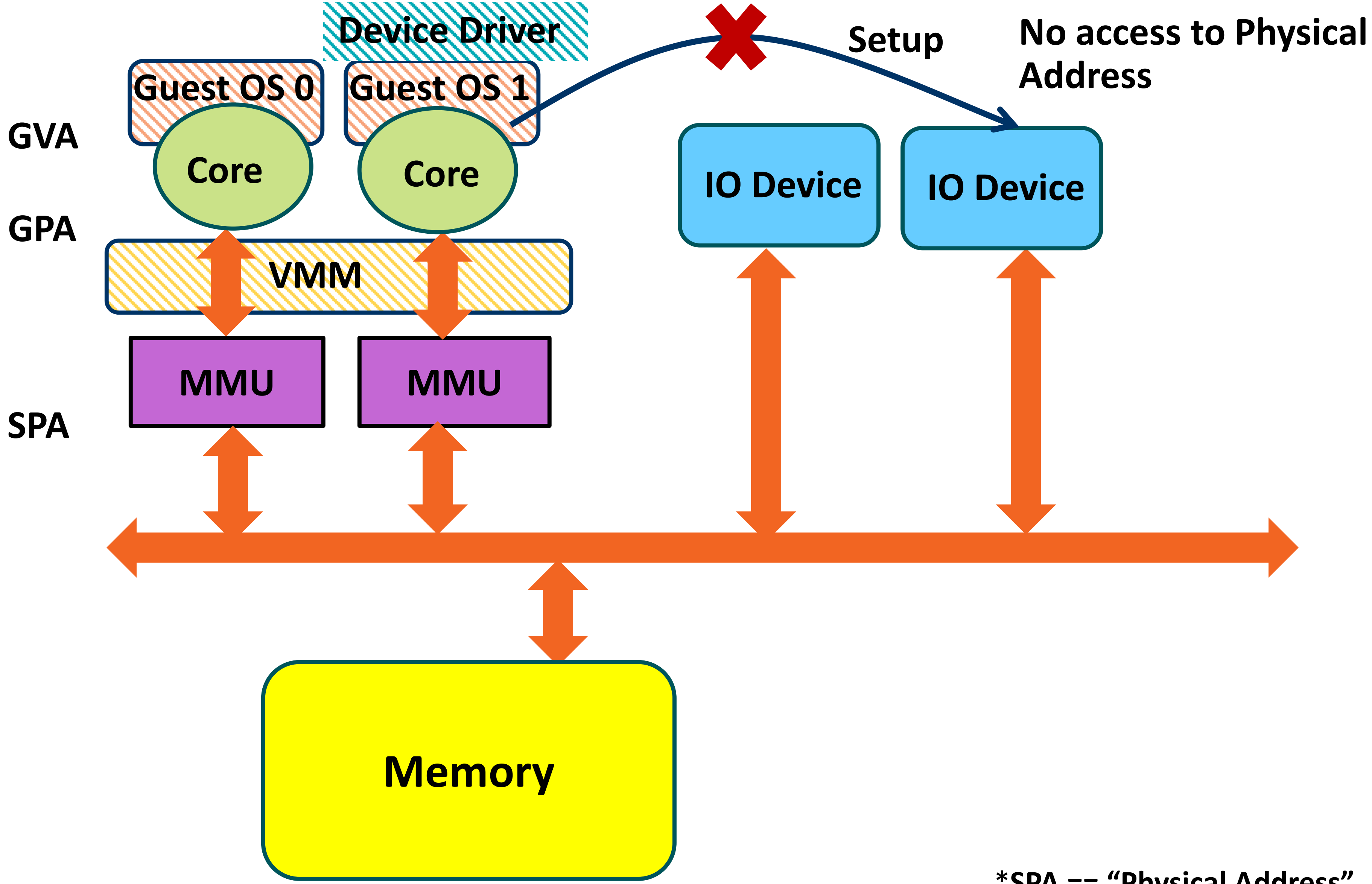
VIRTUALIZED SYSTEM



*SPA == “Physical Address”

MOTIVATION: TRADITIONAL DMA IN VIRTUAL MACHINES

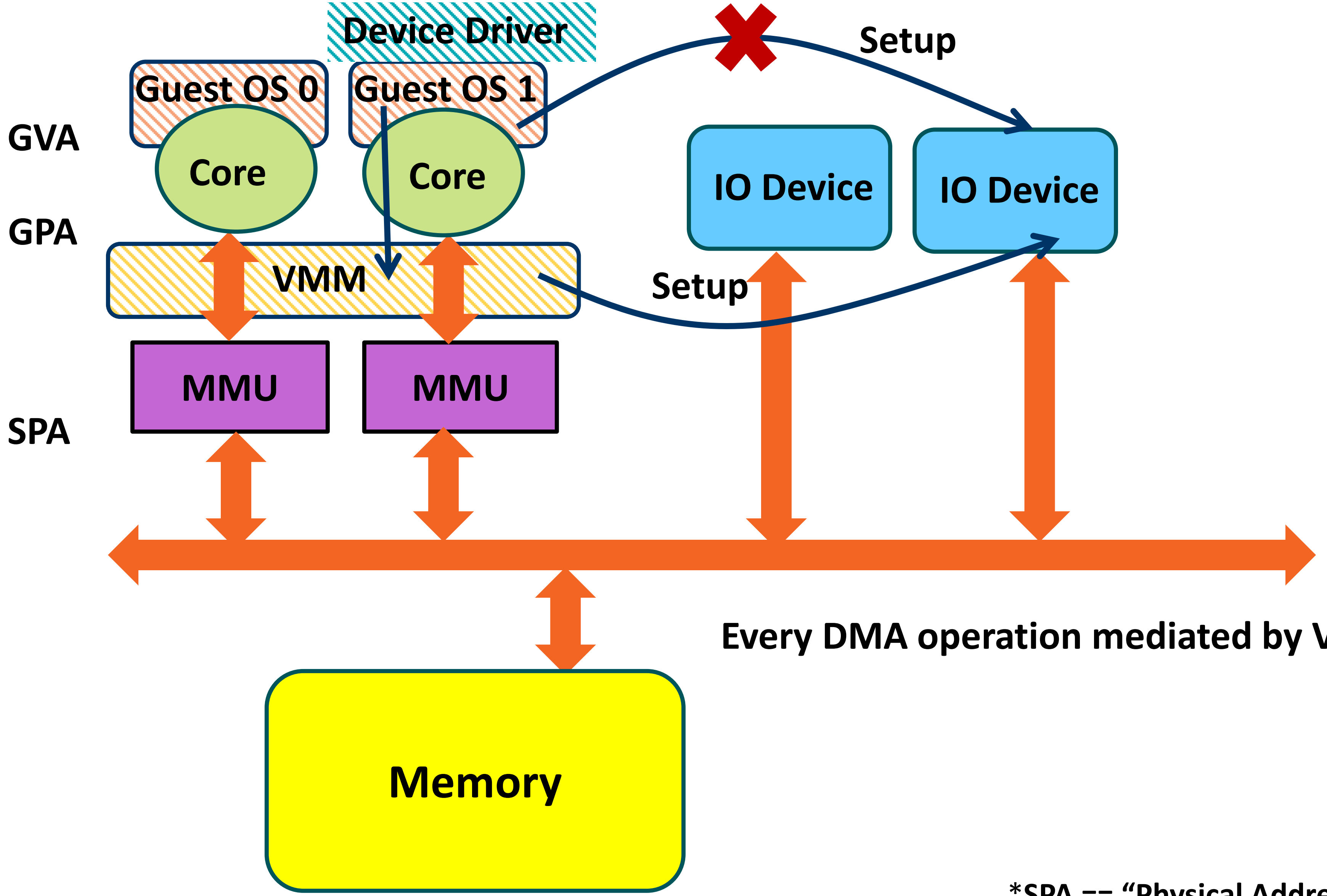
VIRTUALIZED SYSTEM



*SPA == "Physical Address"

MOTIVATION: TRADITIONAL DMA IN VIRTUAL MACHINES

VIRTUALIZED SYSTEM

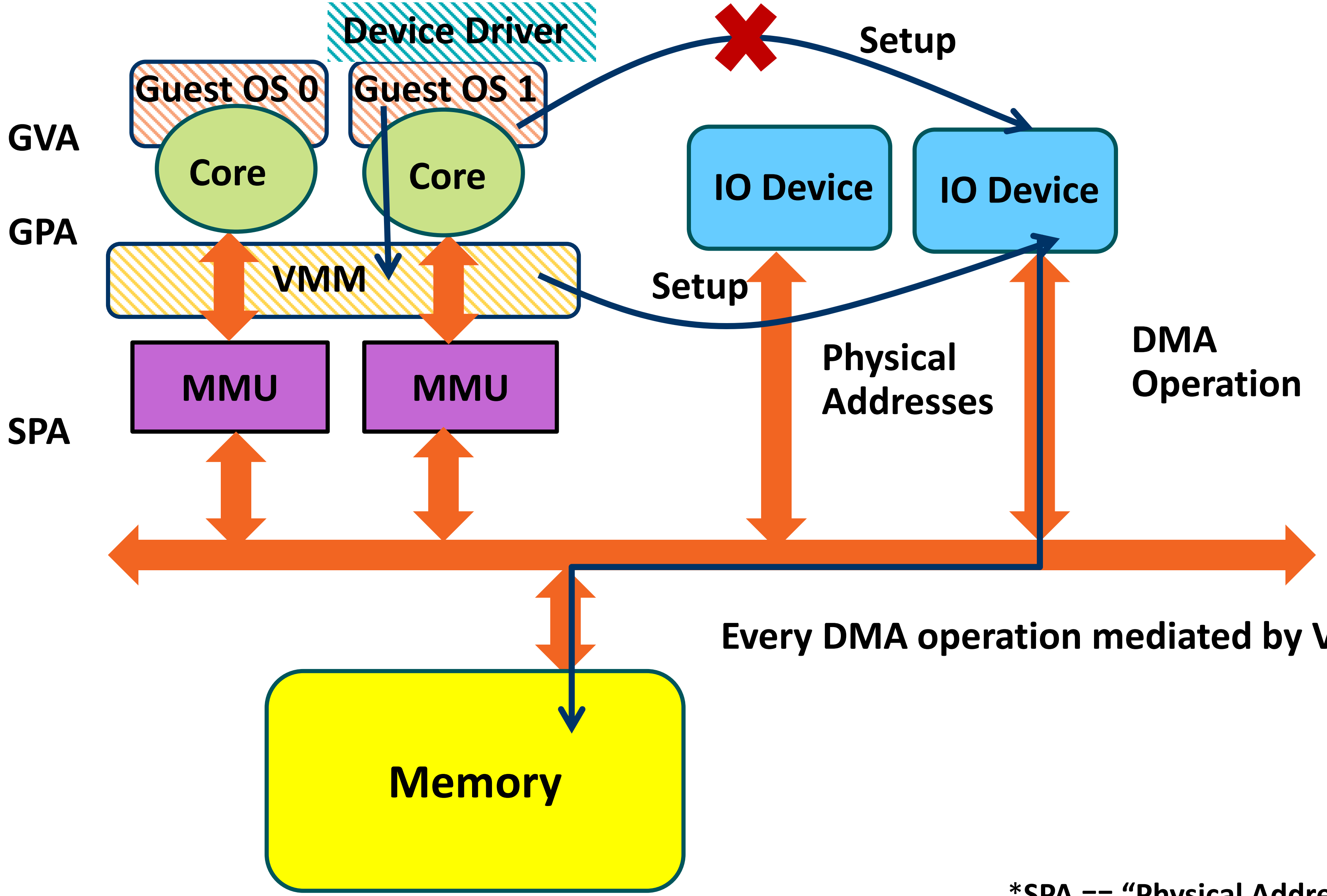


Every DMA operation mediated by VMM

*SPA == “Physical Address”

MOTIVATION: TRADITIONAL DMA IN VIRTUAL MACHINES

VIRTUALIZED SYSTEM

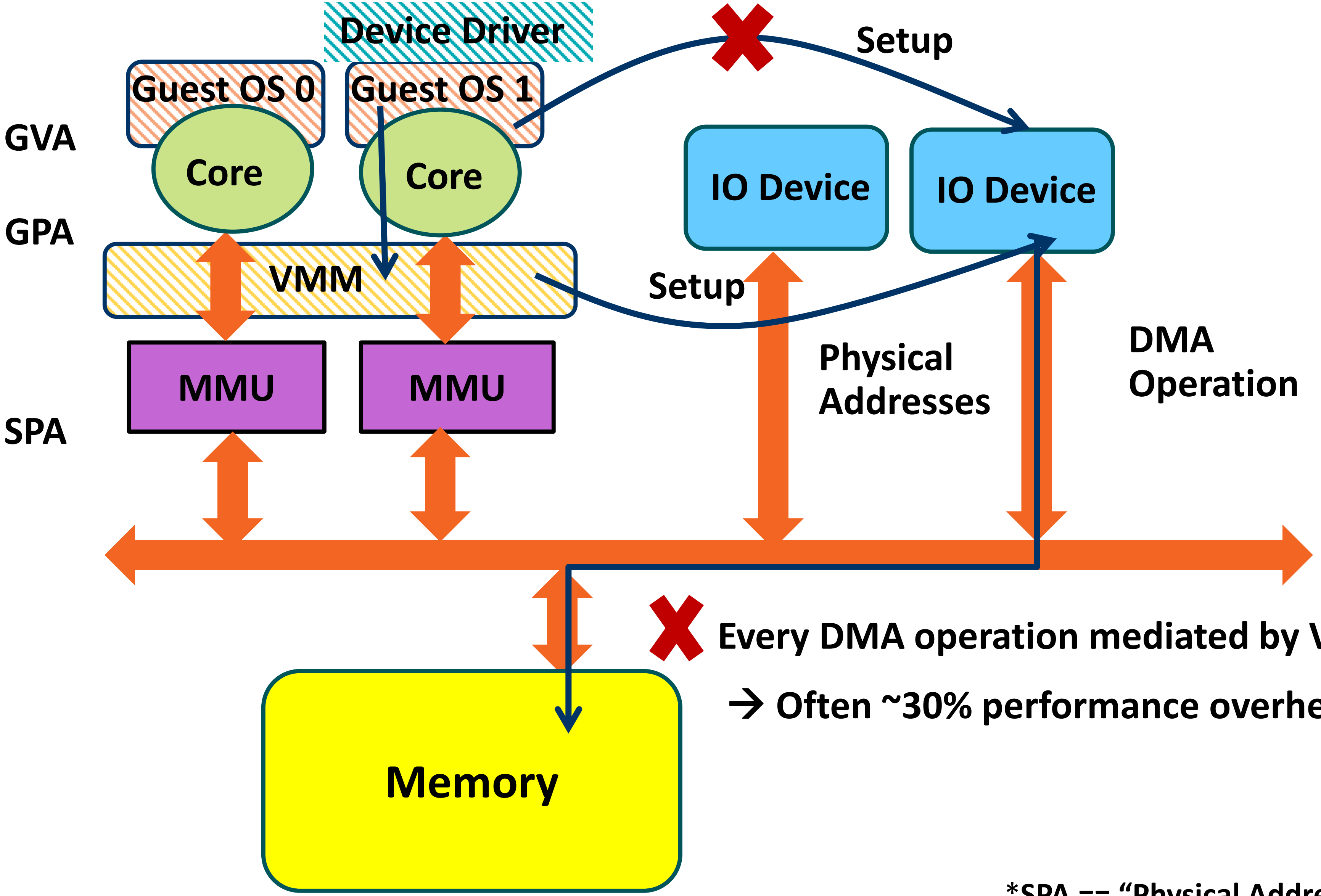


Every DMA operation mediated by VMM

*SPA == “Physical Address”

MOTIVATION: TRADITIONAL DMA IN VIRTUAL MACHINES

VIRTUALIZED SYSTEM



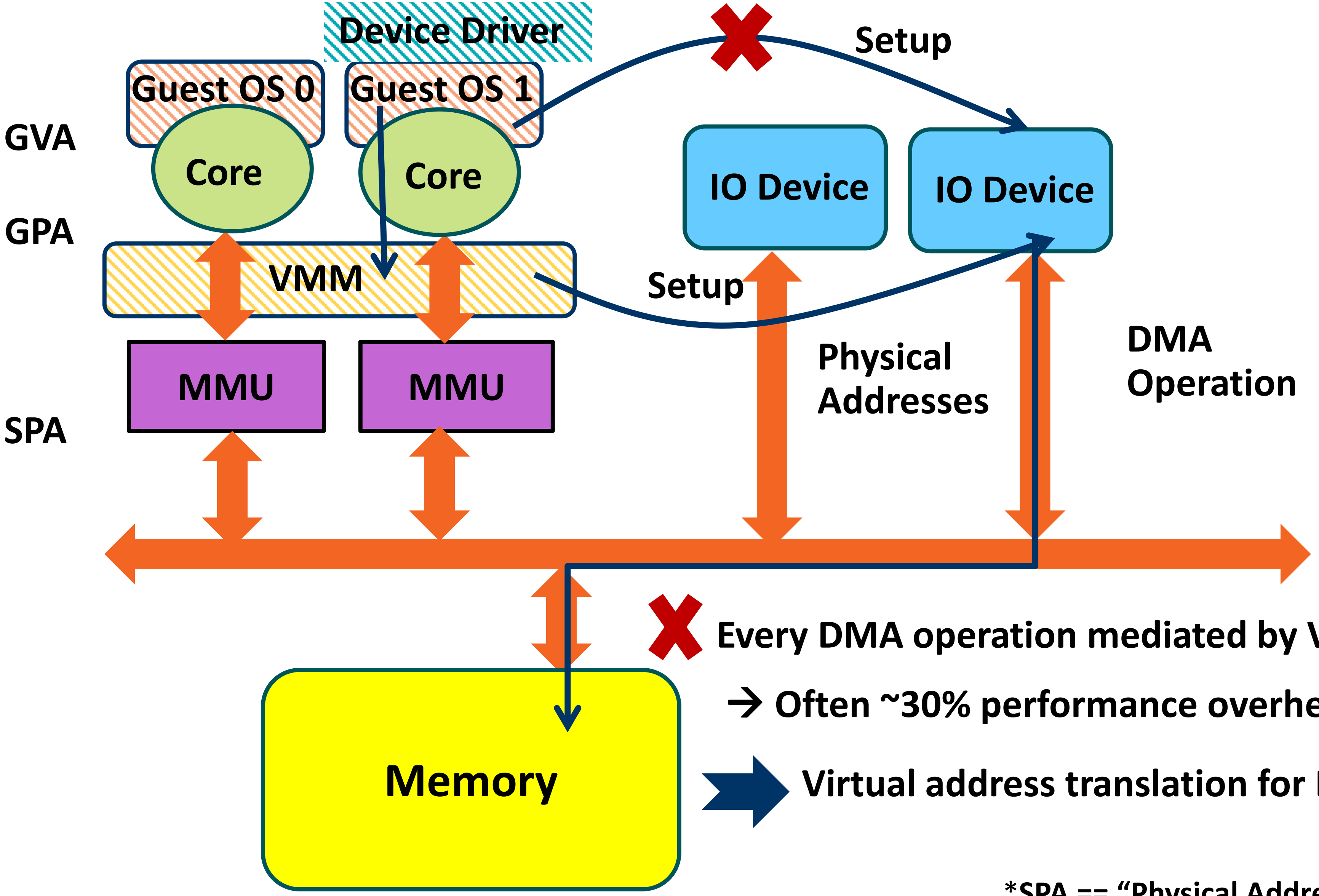
Every DMA operation mediated by VMM

→ Often ~30% performance overhead

*SPA == "Physical Address"

MOTIVATION: TRADITIONAL DMA IN VIRTUAL MACHINES

VIRTUALIZED SYSTEM



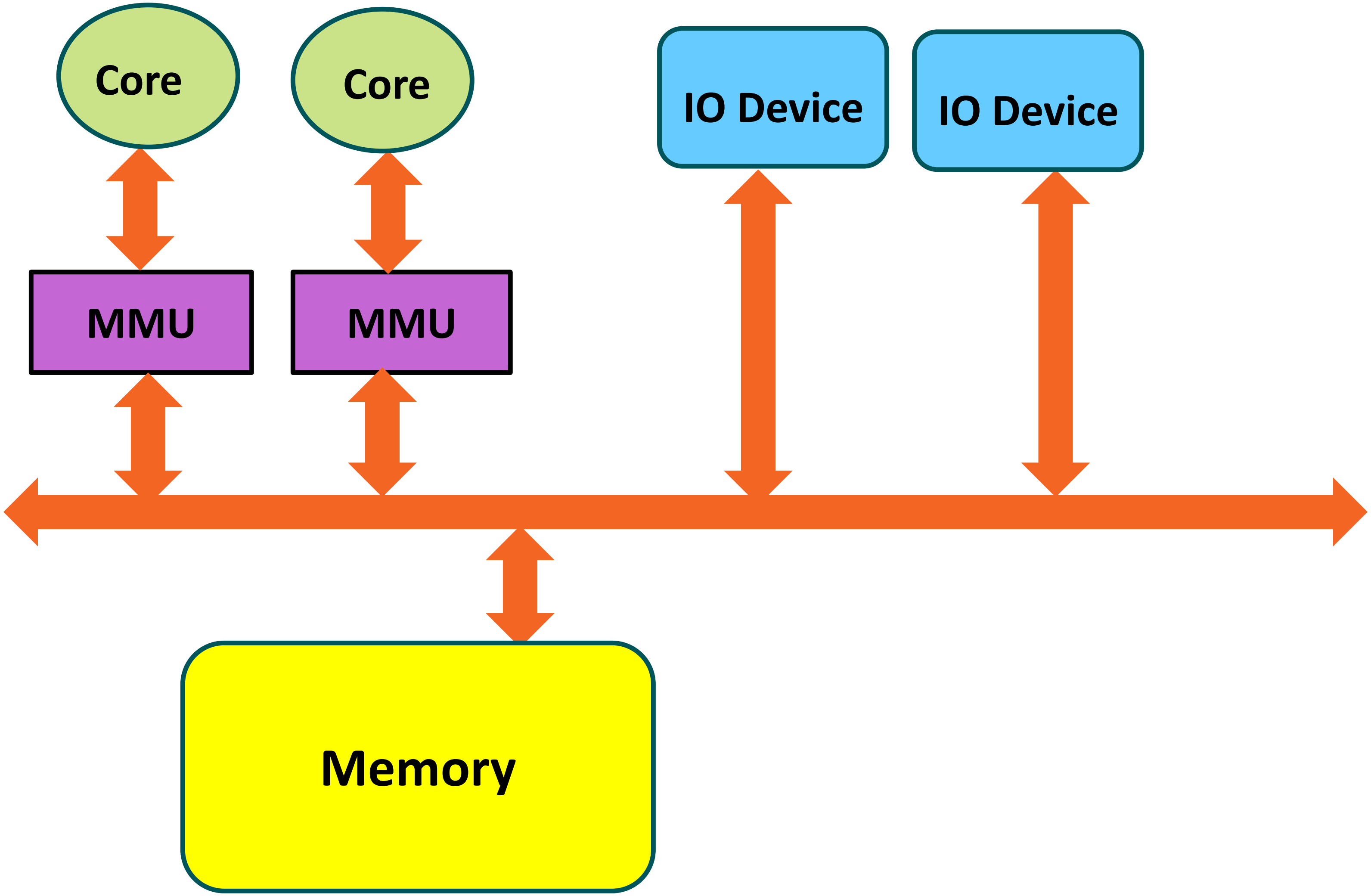
Every DMA operation mediated by VMM

→ Often ~30% performance overhead

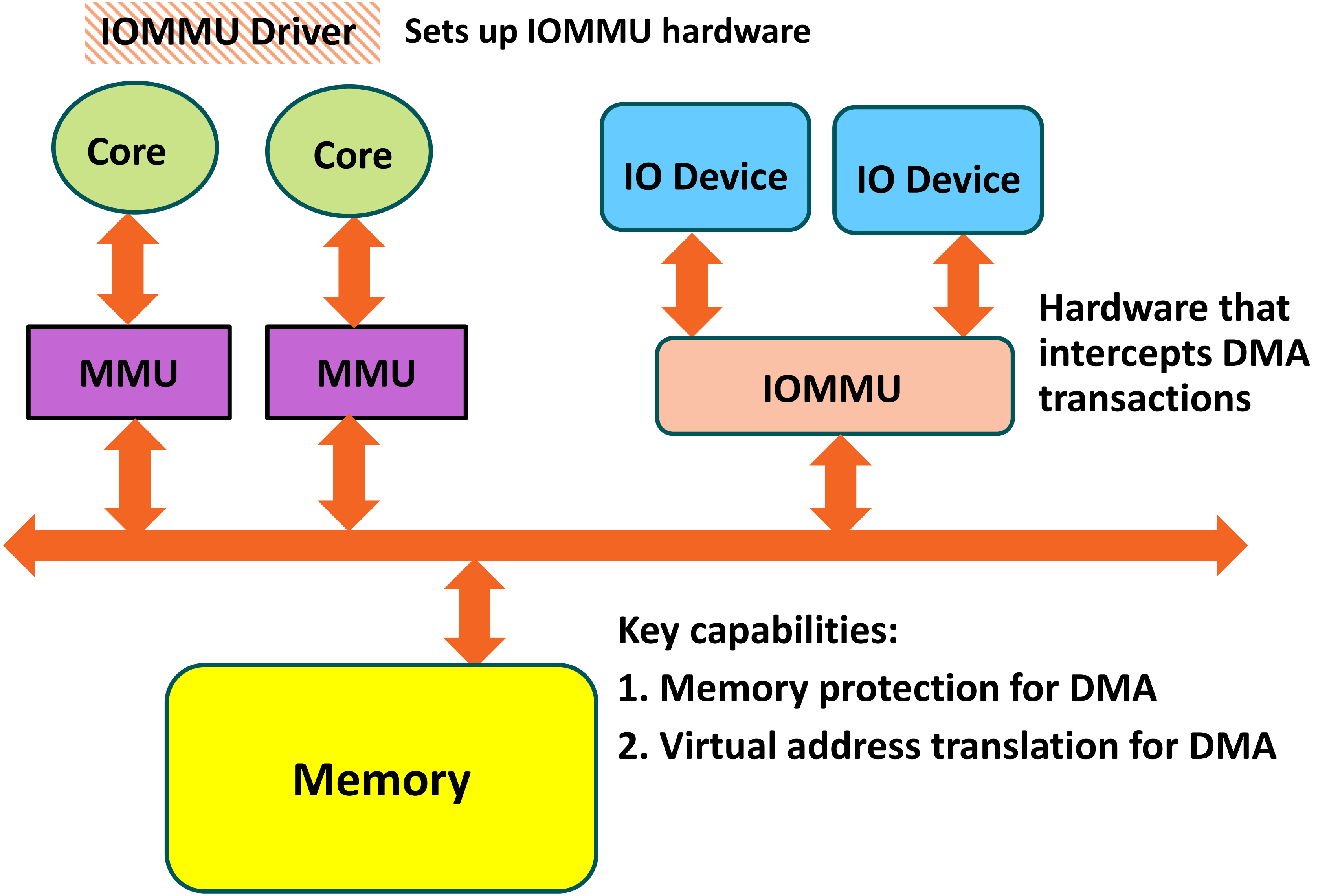
Virtual address translation for DMA

*SPA == "Physical Address"

INTRODUCTION OF IOMMU: THE LOGICAL VIEW



INTRODUCTION OF IOMMU: THE LOGICAL VIEW



Sets up IOMMU hardware

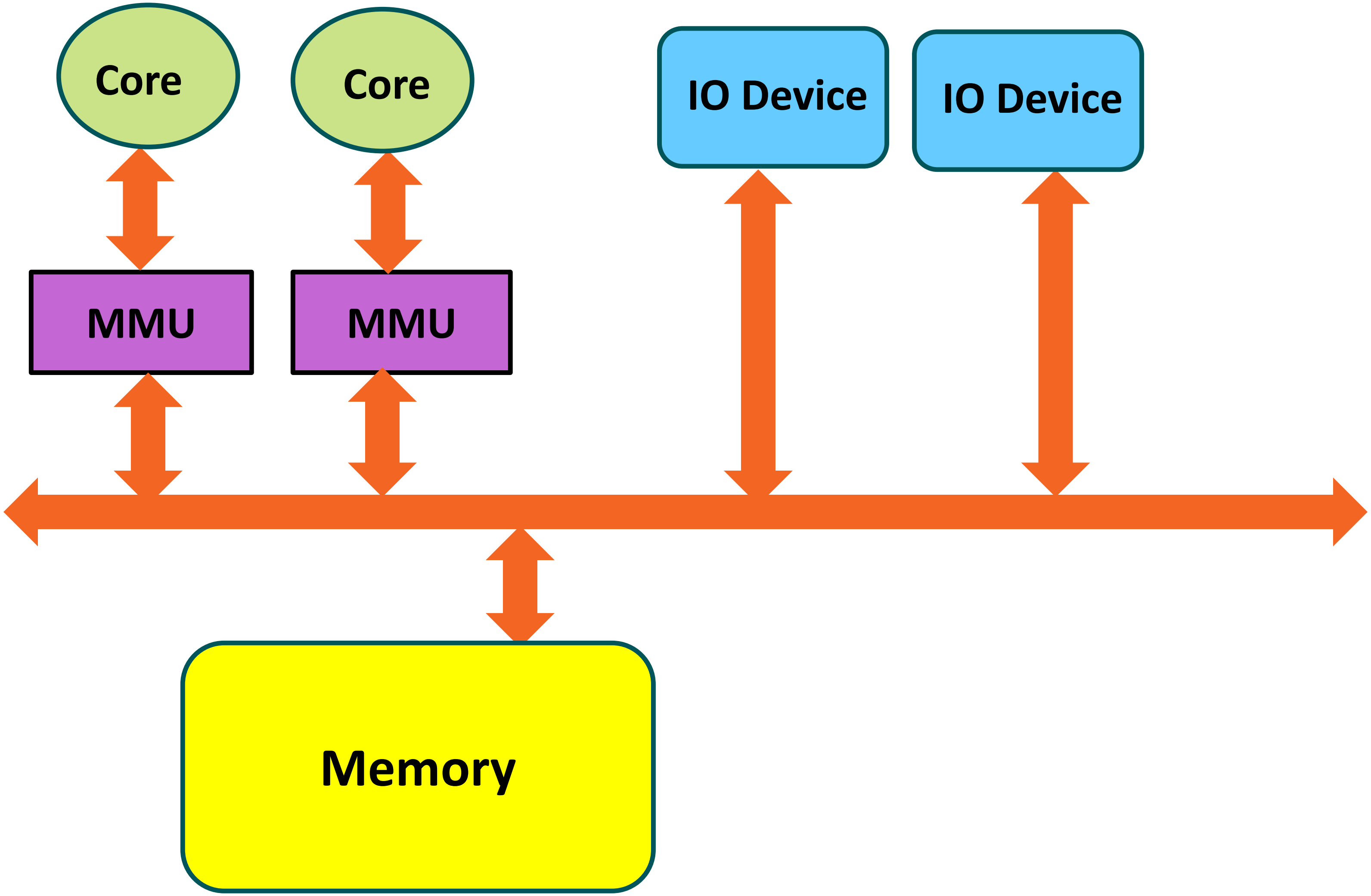
Hardware that intercepts DMA transactions

Key capabilities:

1. Memory protection for DMA
2. Virtual address translation for DMA

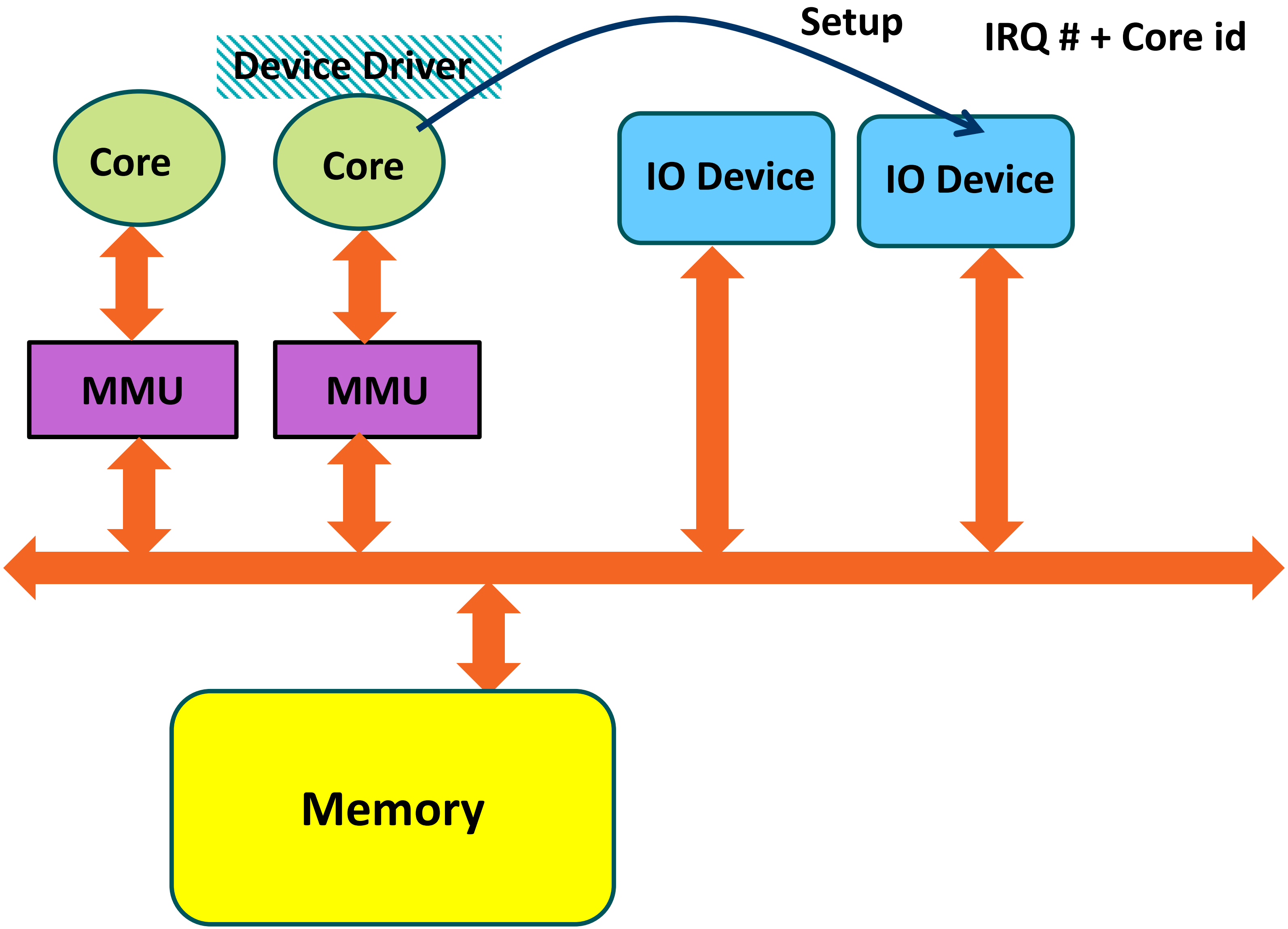
MOTIVATION: TRADITIONAL IO INTERRUPT

NON-VIRTUALIZED SYSTEM



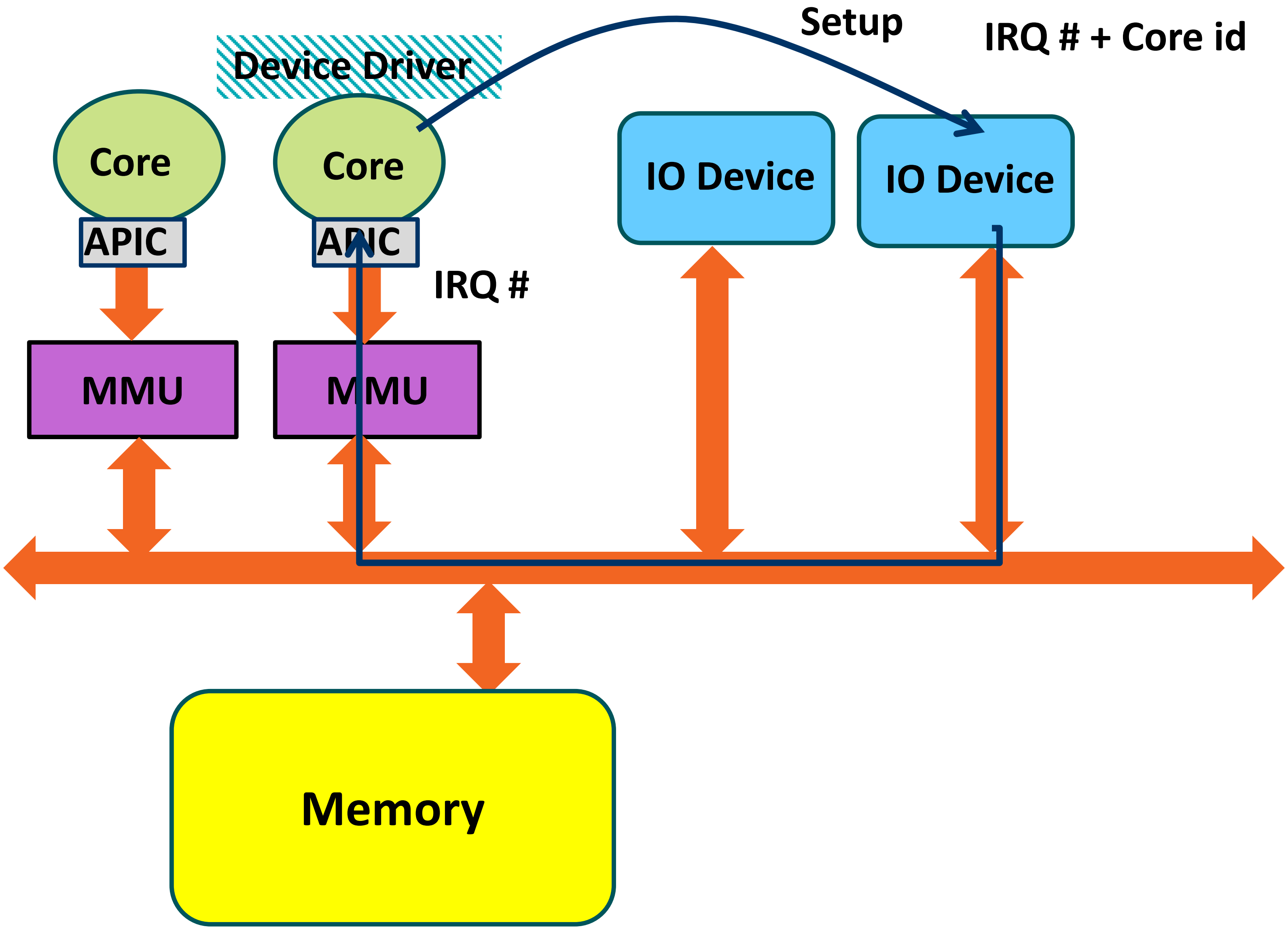
MOTIVATION: TRADITIONAL IO INTERRUPT

NON-VIRTUALIZED SYSTEM



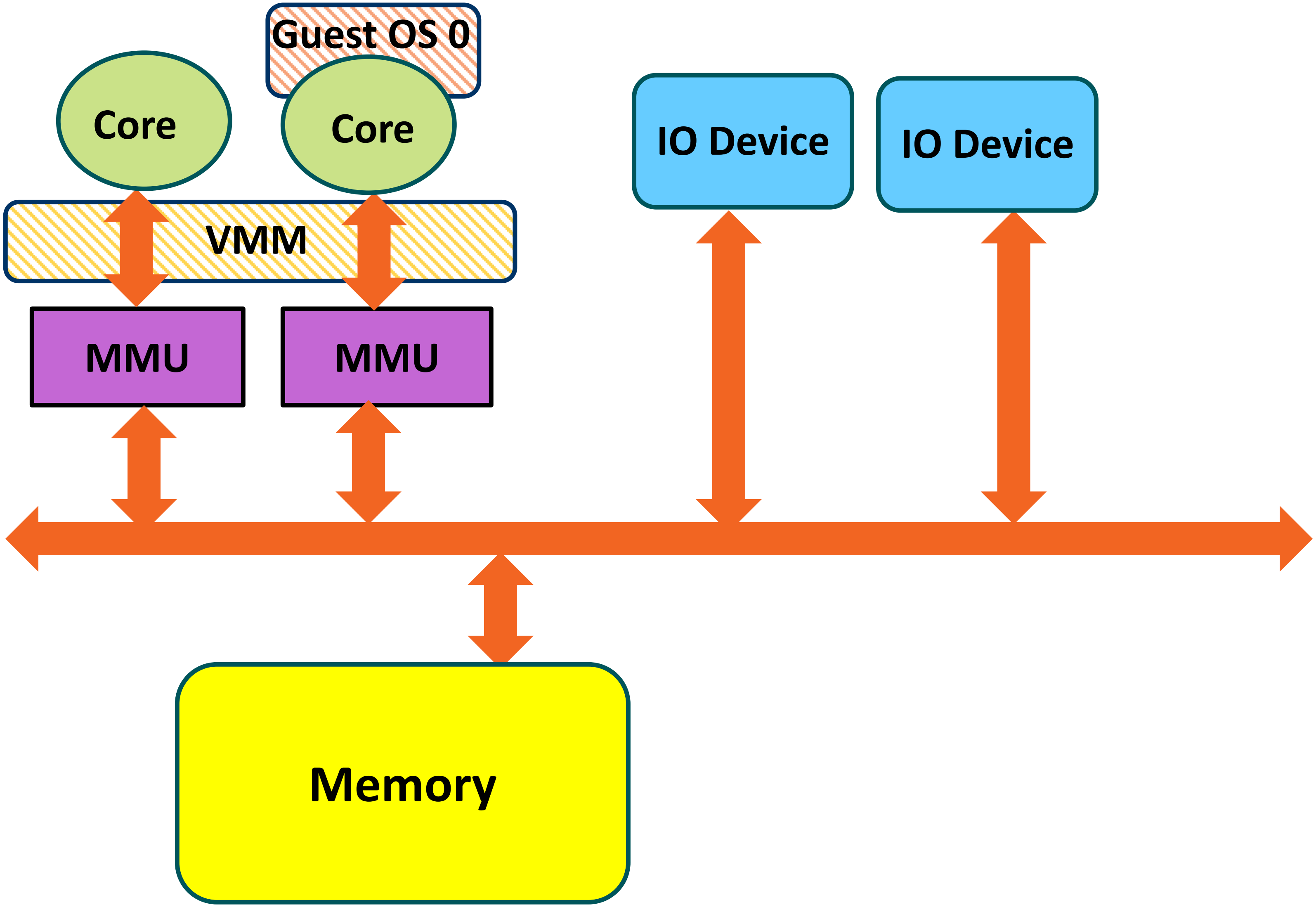
MOTIVATION: TRADITIONAL IO INTERRUPT

NON-VIRTUALIZED SYSTEM



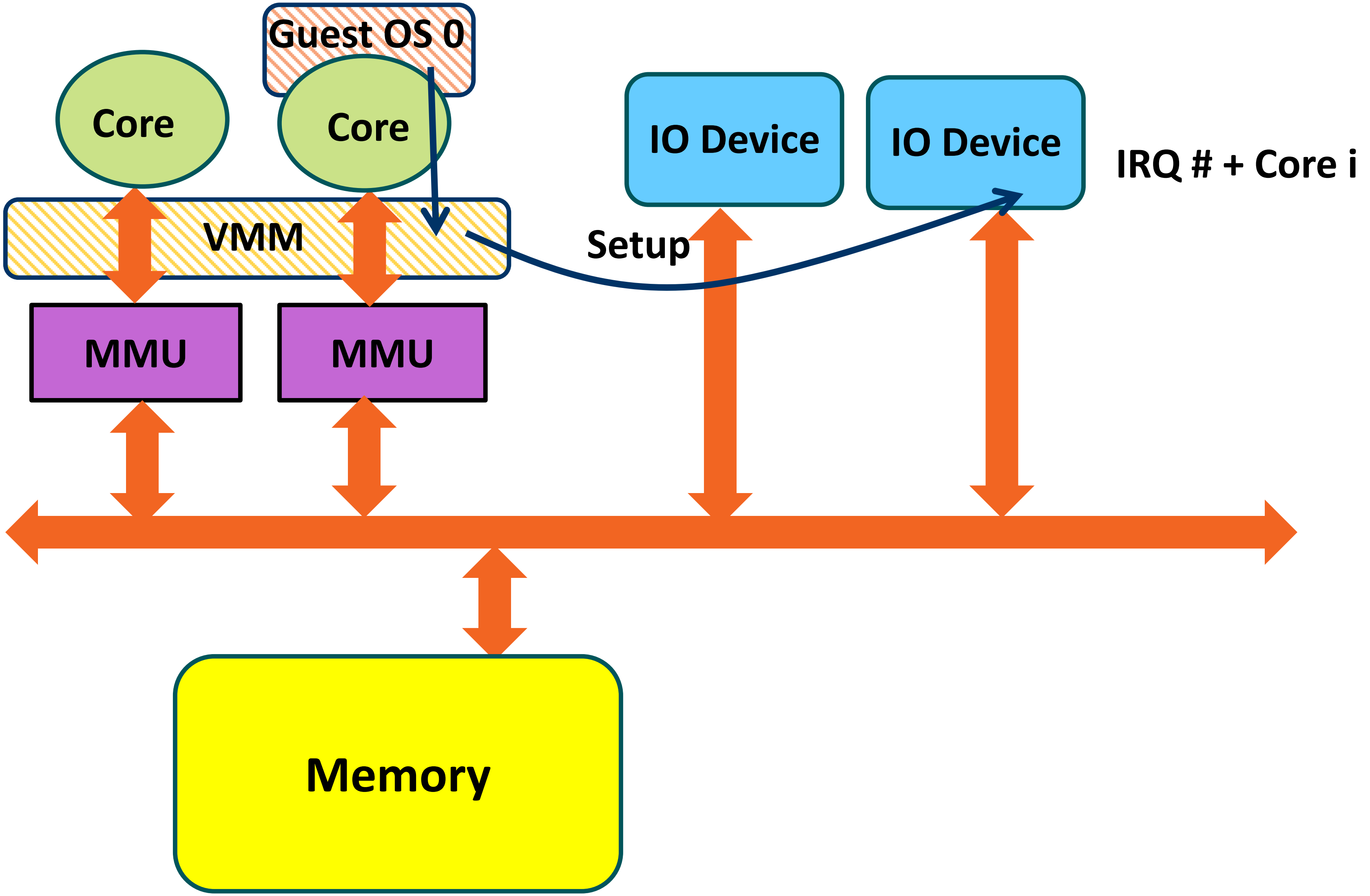
MOTIVATION: TRADITIONAL IO INTERRUPT

VIRTUALIZED SYSTEM



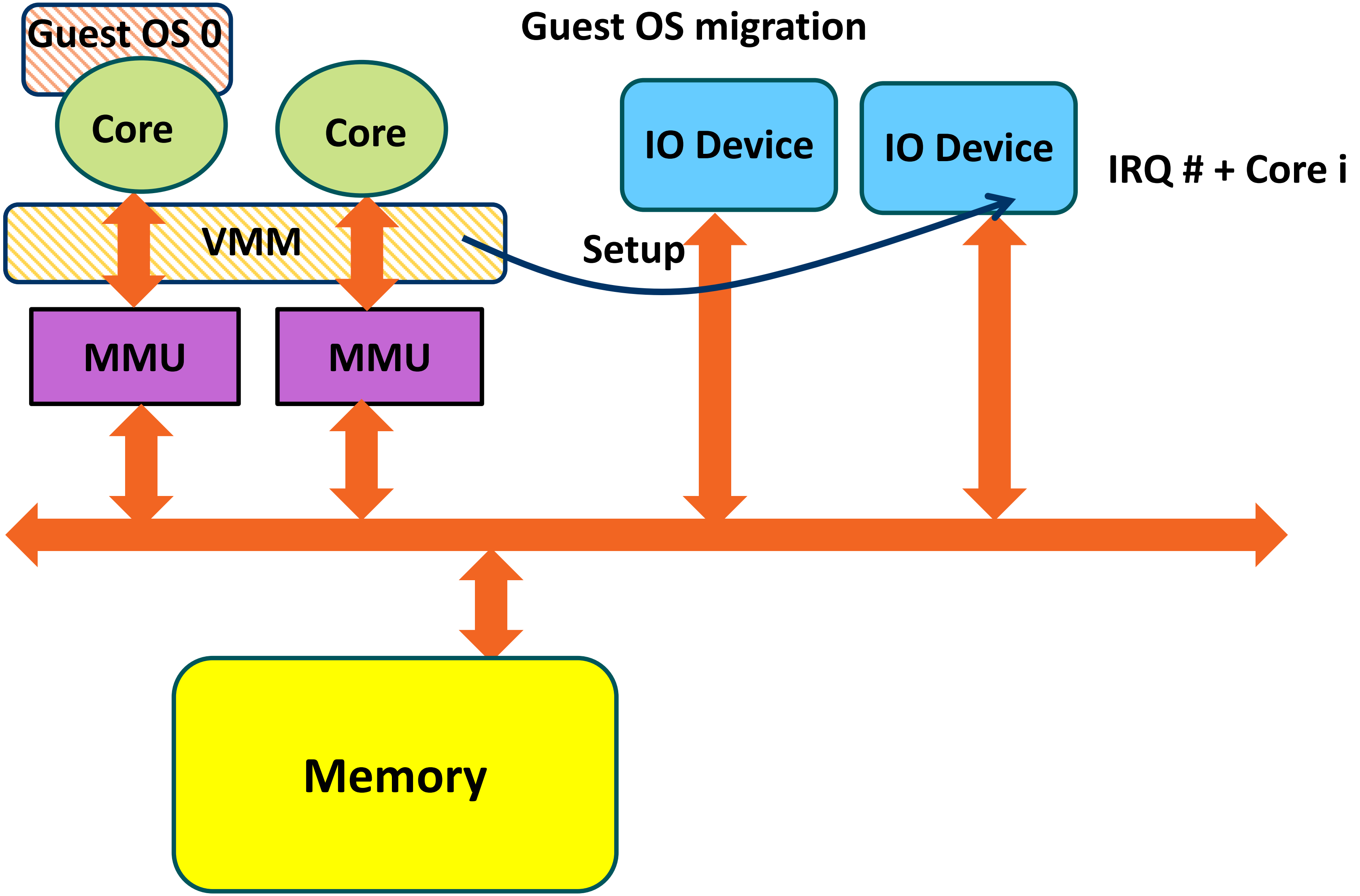
MOTIVATION: TRADITIONAL IO INTERRUPT

VIRTUALIZED SYSTEM



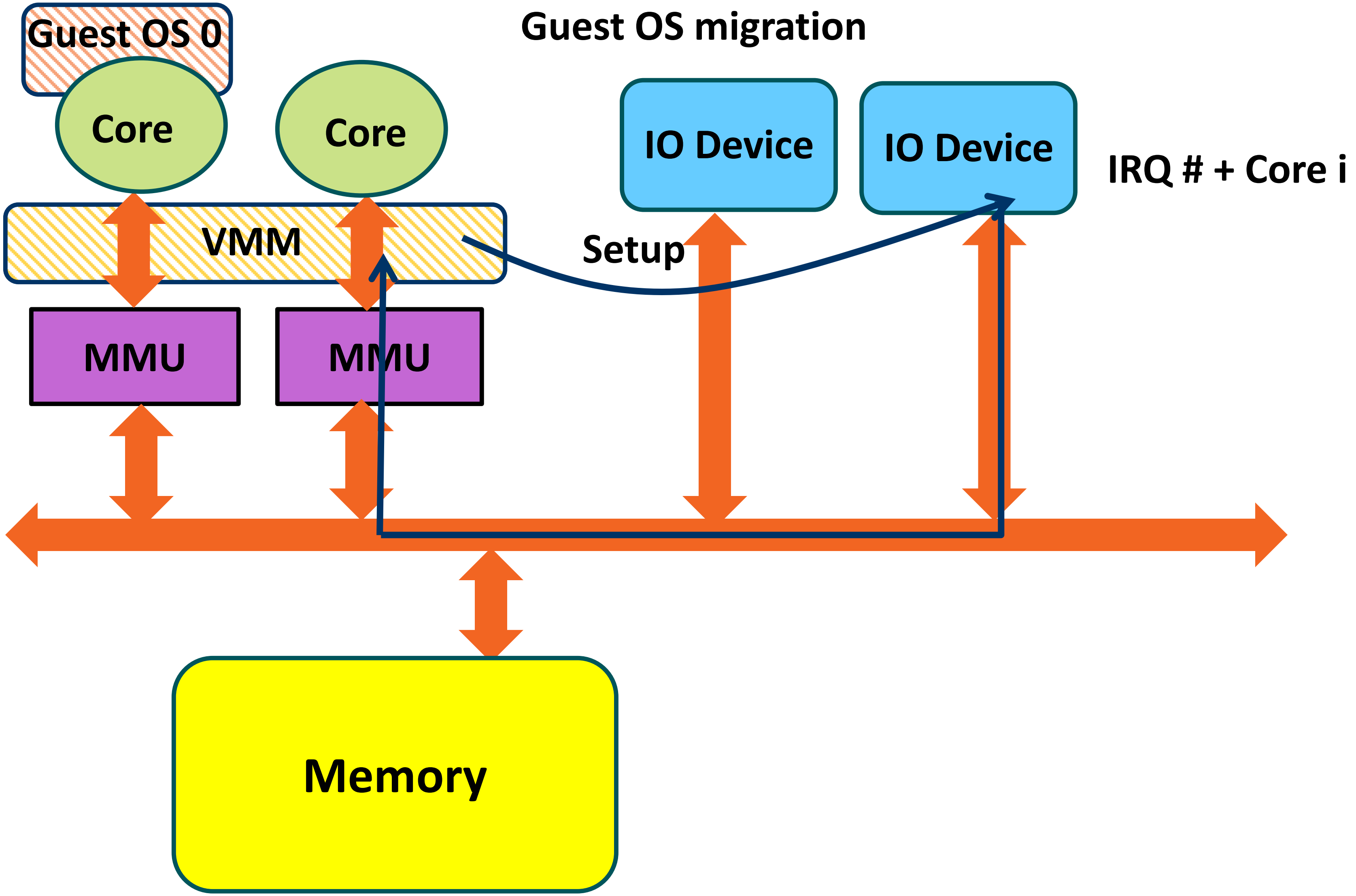
MOTIVATION: TRADITIONAL IO INTERRUPT

VIRTUALIZED SYSTEM



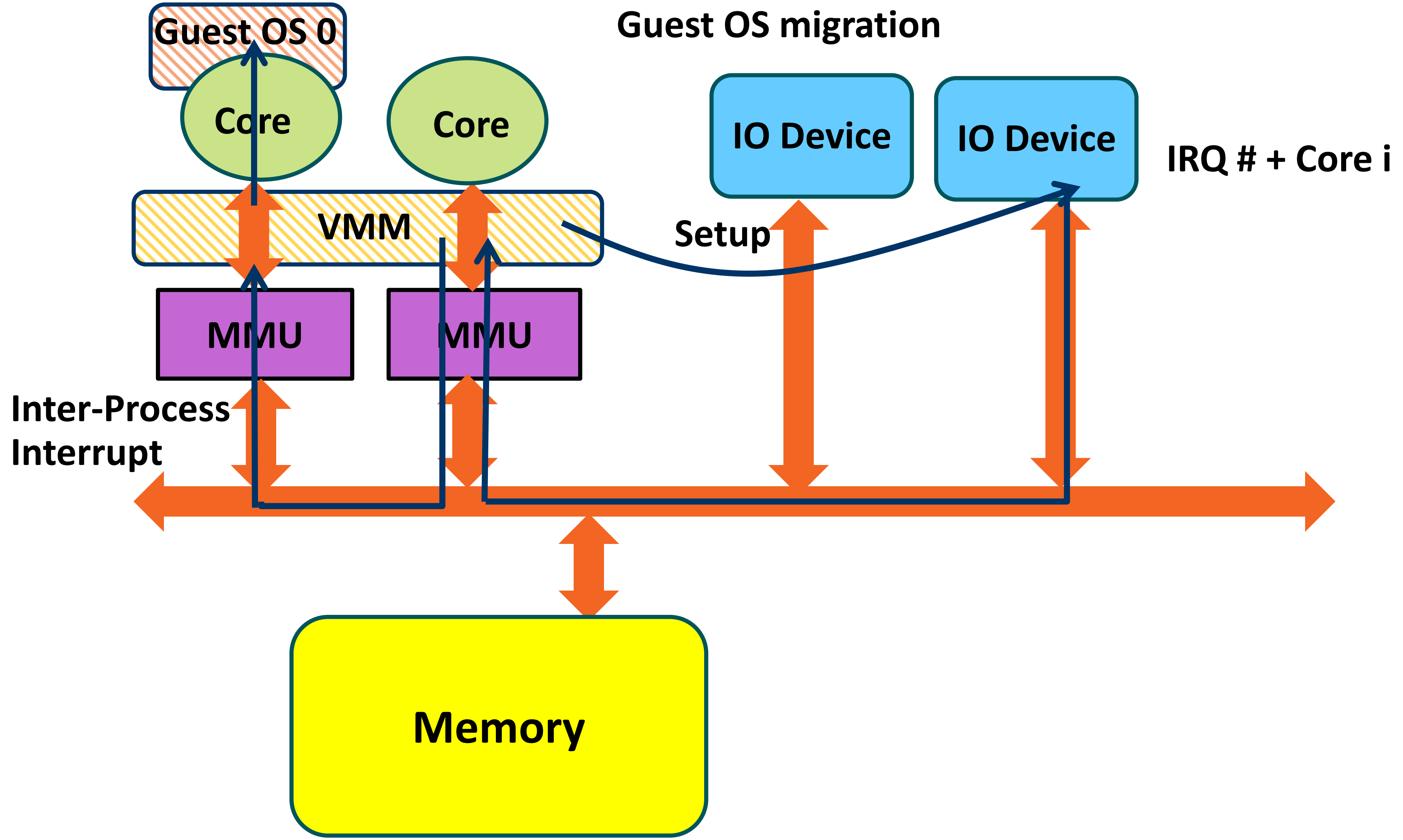
MOTIVATION: TRADITIONAL IO INTERRUPT

VIRTUALIZED SYSTEM



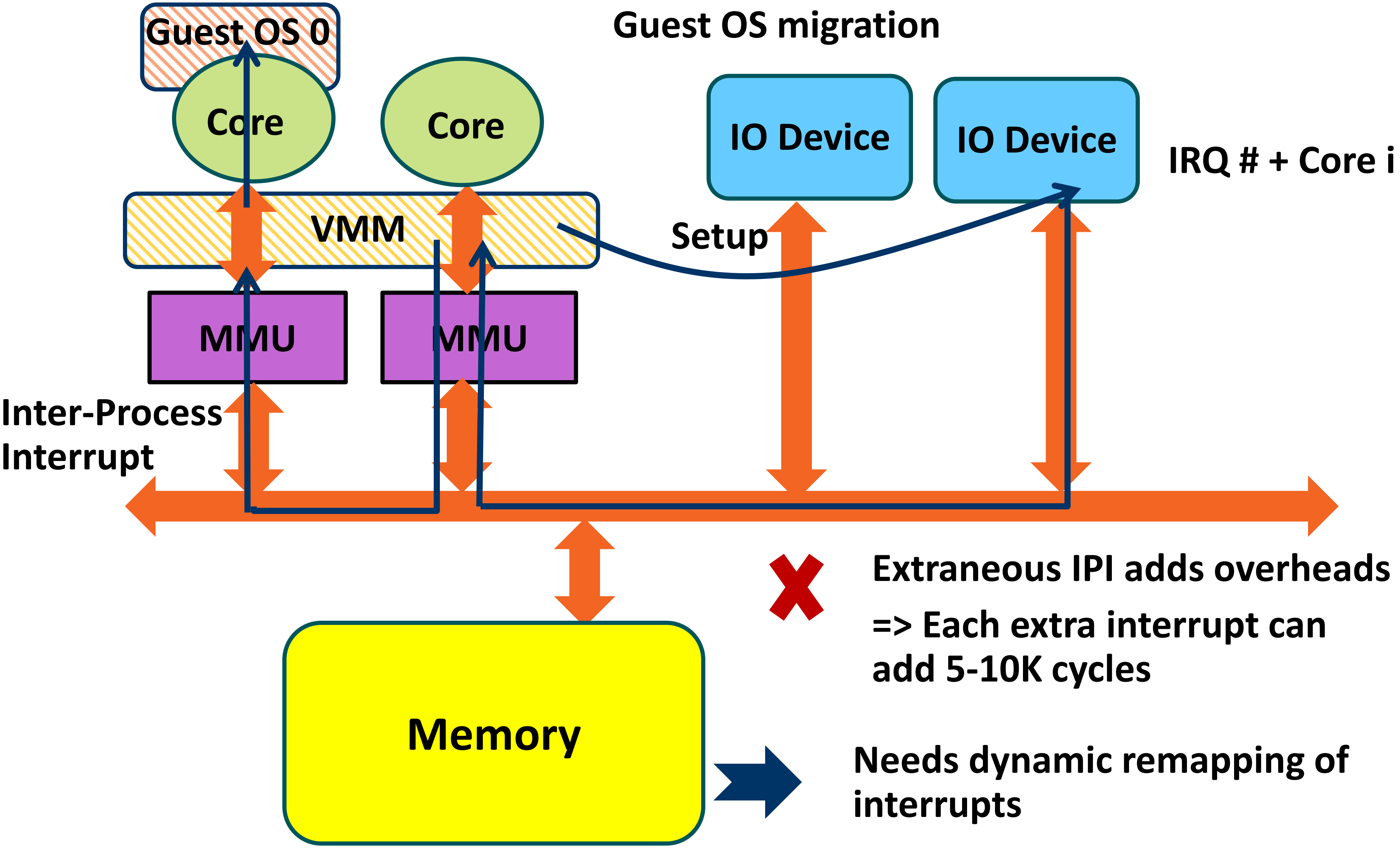
MOTIVATION: TRADITIONAL IO INTERRUPT

VIRTUALIZED SYSTEM



MOTIVATION: TRADITIONAL IO INTERRUPT

VIRTUALIZED SYSTEM



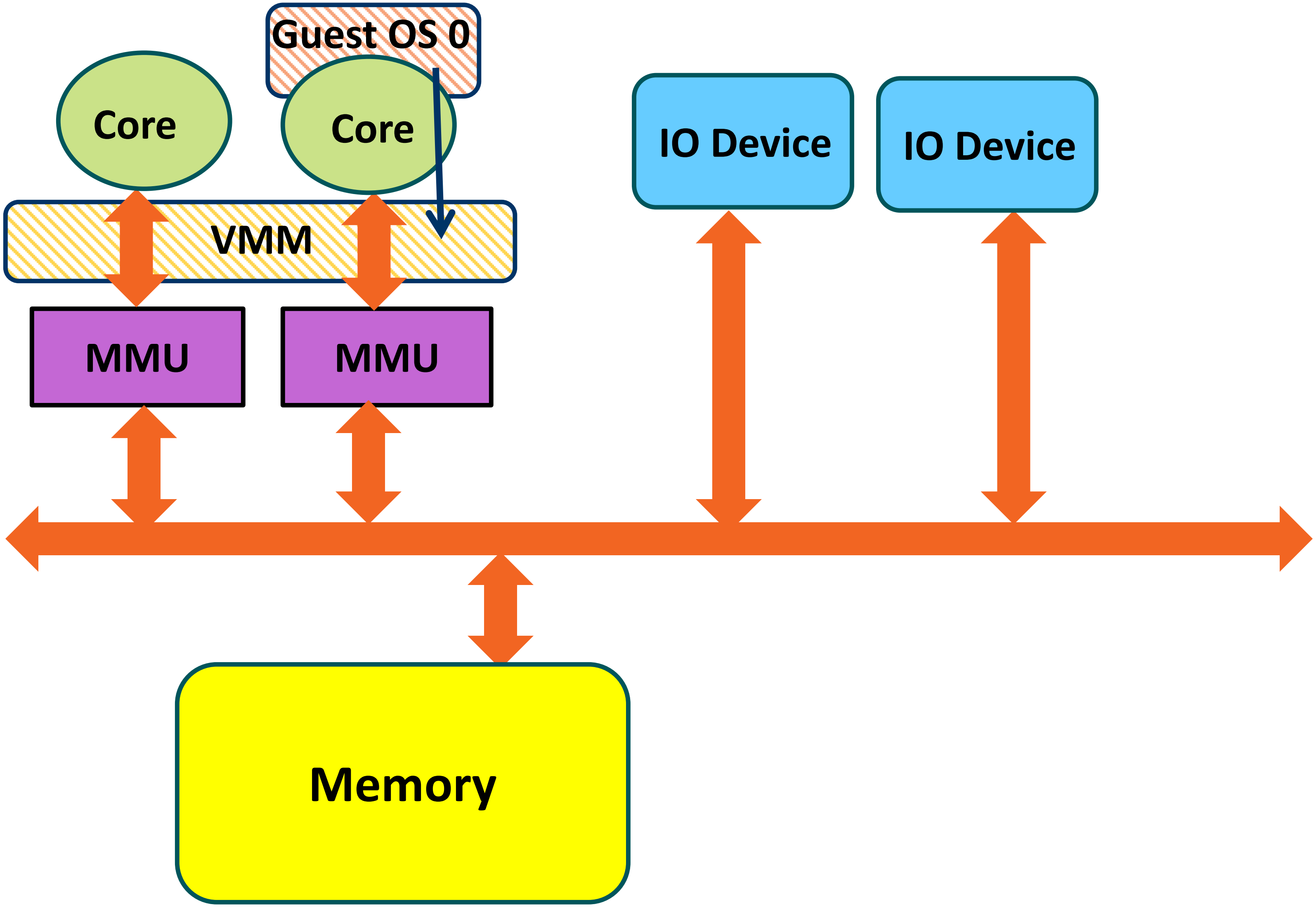
Extraneous IPI adds overheads

=> Each extra interrupt can add 5-10K cycles

Needs dynamic remapping of interrupts

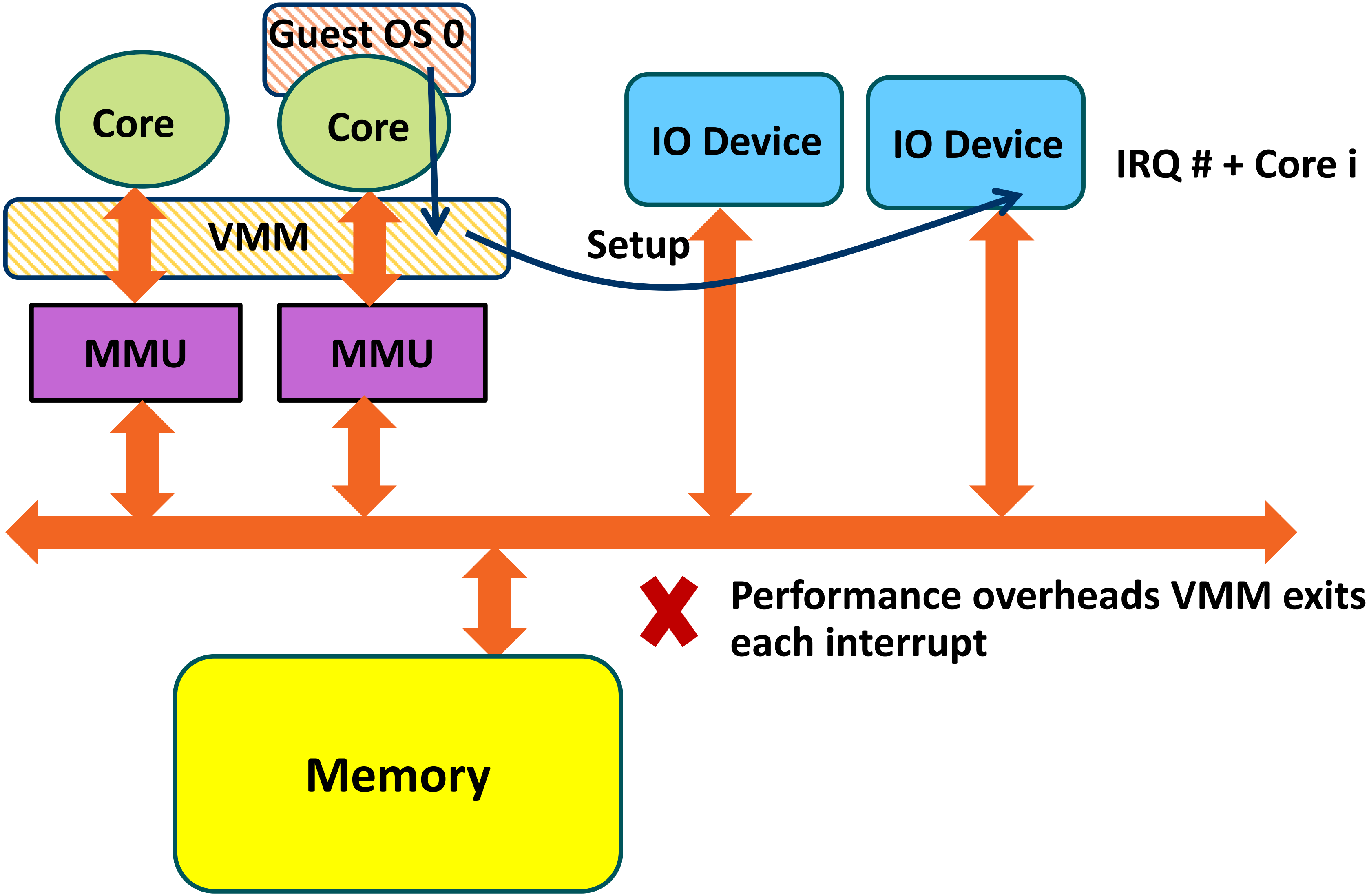
MOTIVATION: TRADITIONAL IO INTERRUPT

VIRTUALIZED SYSTEM



MOTIVATION: TRADITIONAL IO INTERRUPT

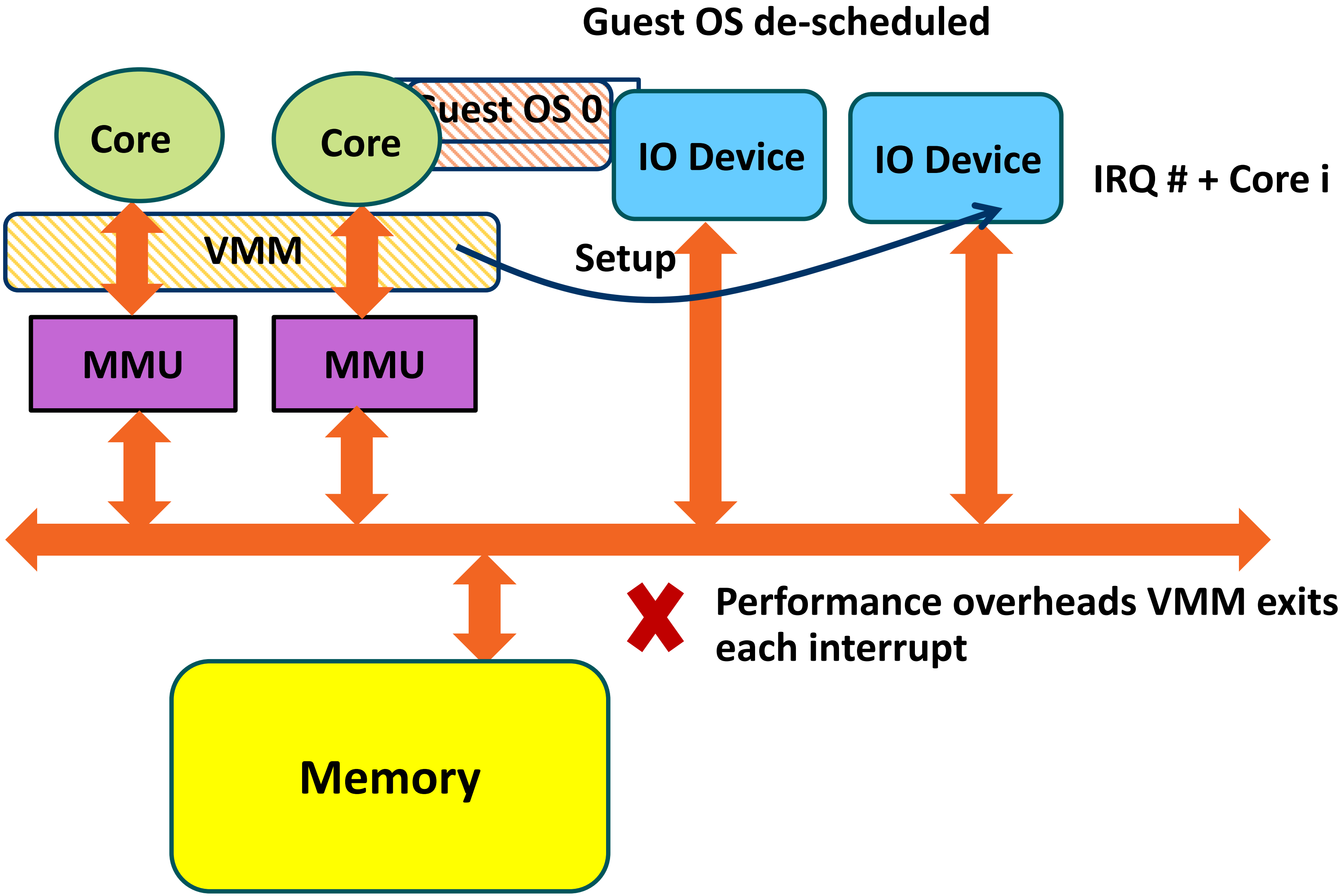
VIRTUALIZED SYSTEM



✗ Performance overheads VMM exits on each interrupt

MOTIVATION: TRADITIONAL IO INTERRUPT

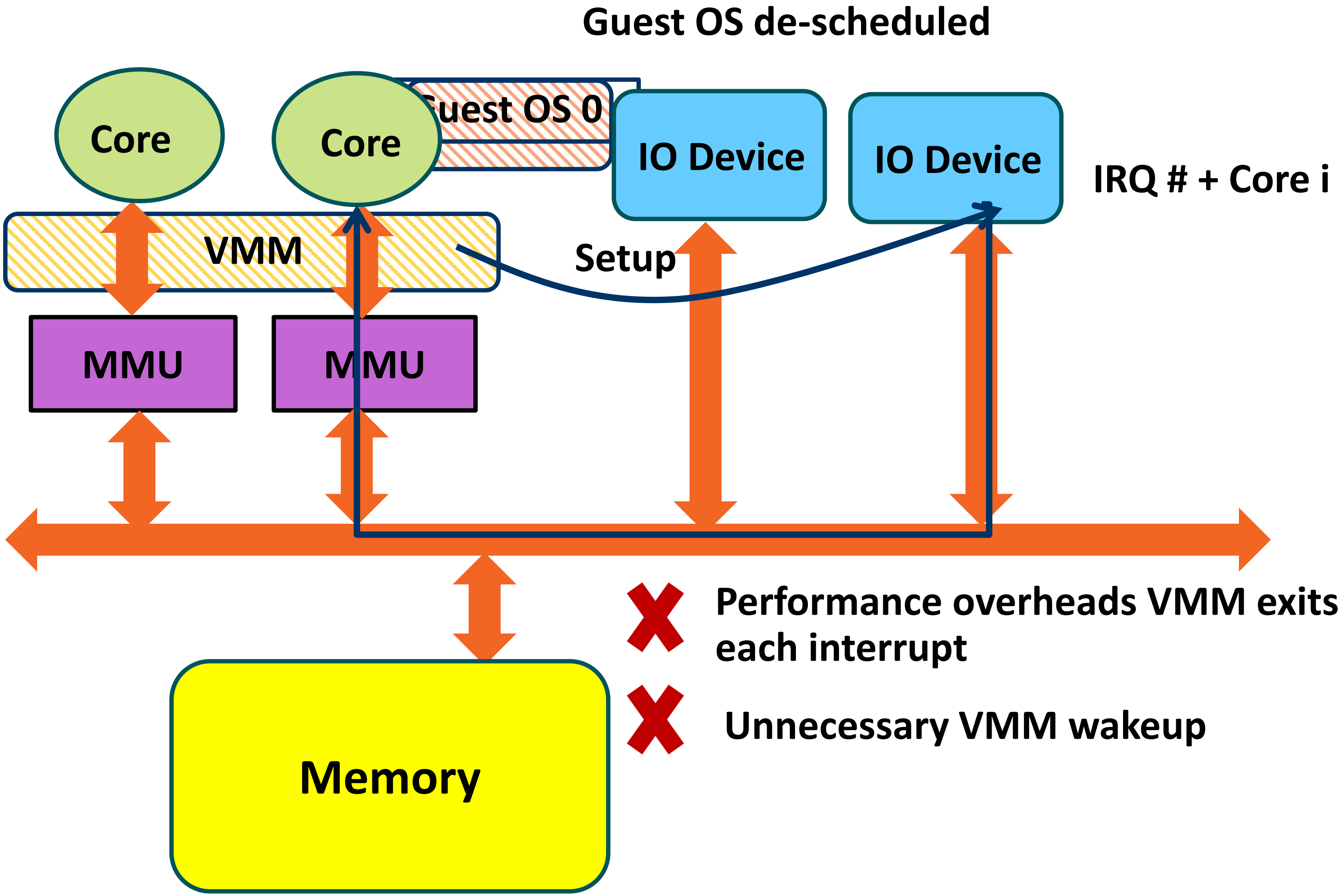
VIRTUALIZED SYSTEM



✗ Performance overheads VMM exits on each interrupt

MOTIVATION: TRADITIONAL IO INTERRUPT

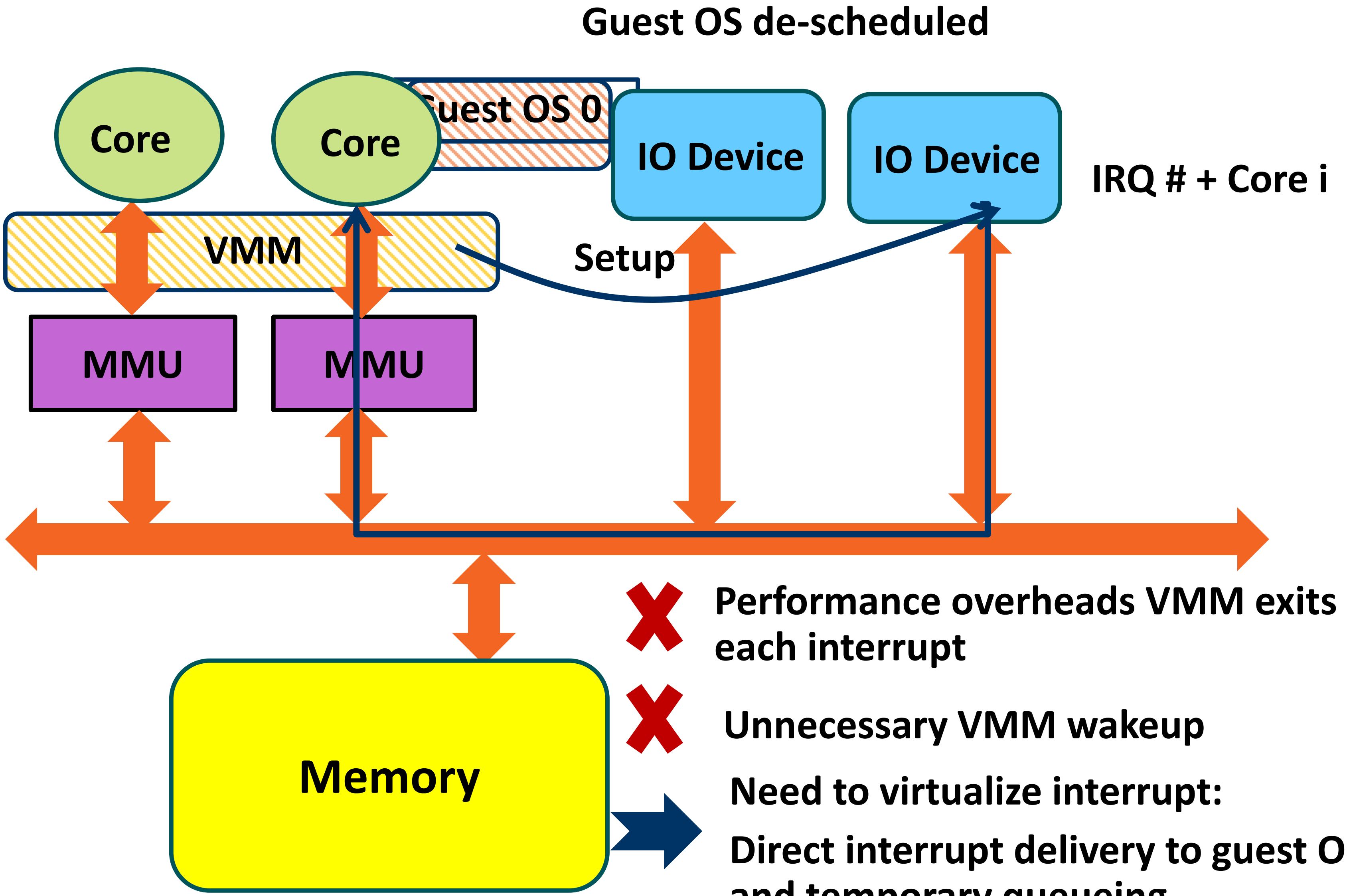
VIRTUALIZED SYSTEM



- ☒ Performance overheads VMM exits on each interrupt
- ☒ Unnecessary VMM wakeup

MOTIVATION: TRADITIONAL IO INTERRUPT

VIRTUALIZED SYSTEM



❌ Performance overheads VMM exits on each interrupt

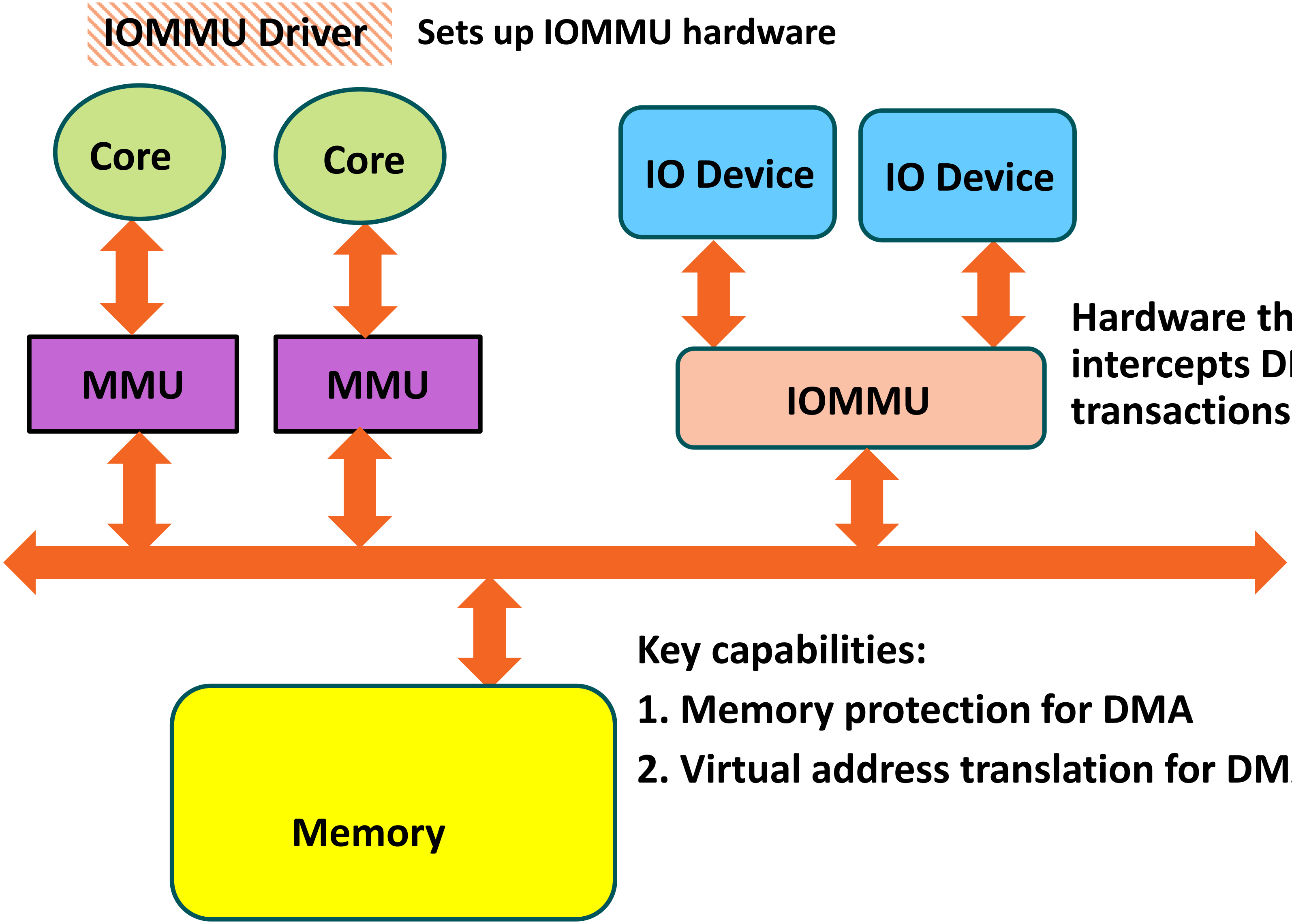
❌ Unnecessary VMM wakeup

Need to virtualize interrupt:

Direct interrupt delivery to guest OS and temporary queueing

INTRODUCTION OF IOMMU: THE LOGICAL VIEW

ADDING INTERRUPT HANDLING CAPABILITY



Sets up IOMMU hardware

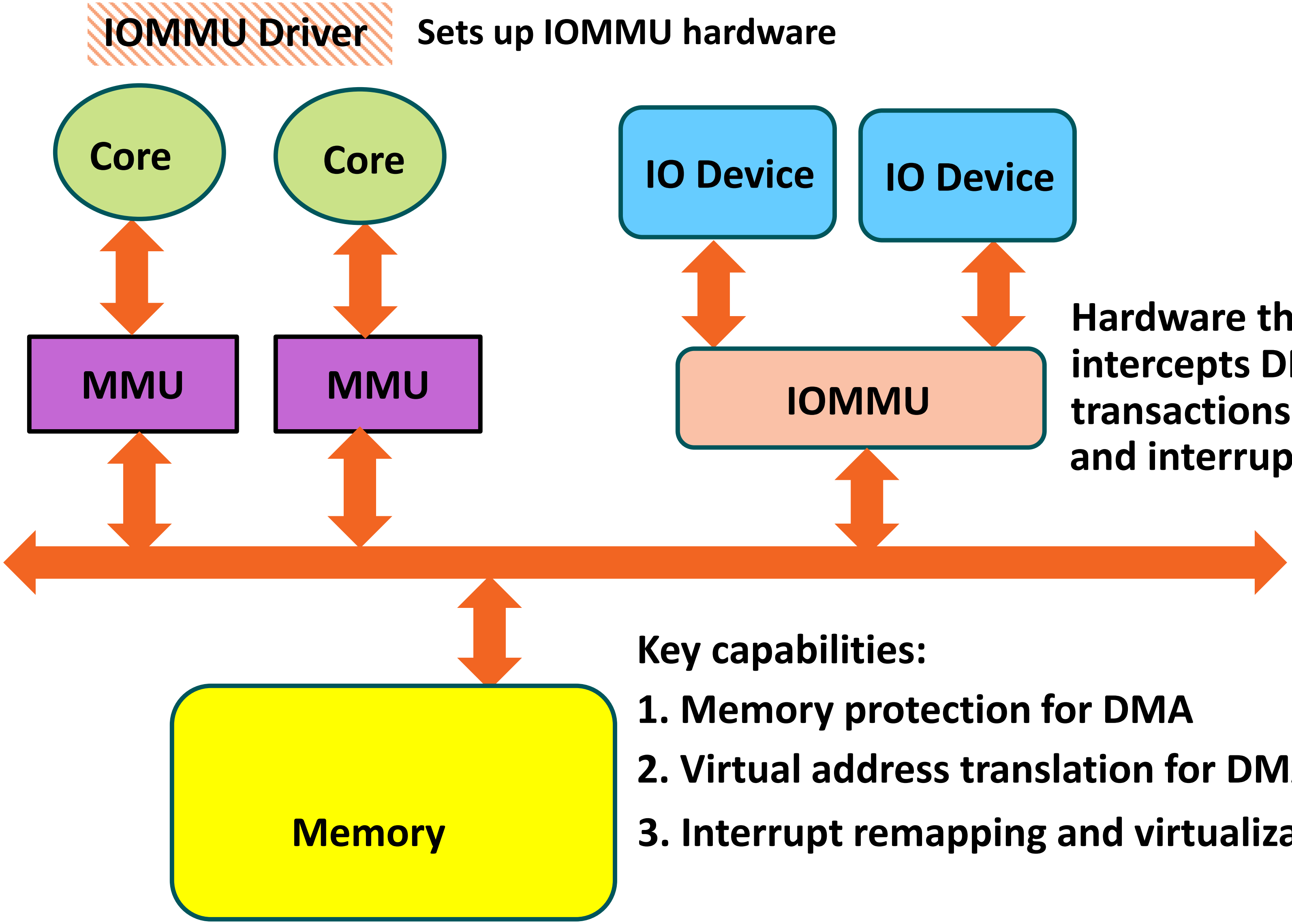
Hardware that intercepts DMA transactions

Key capabilities:

1. Memory protection for DMA
2. Virtual address translation for DMA

INTRODUCTION OF IOMMU: THE LOGICAL VIEW

ADDING INTERRUPT HANDLING CAPABILITY



Sets up IOMMU hardware

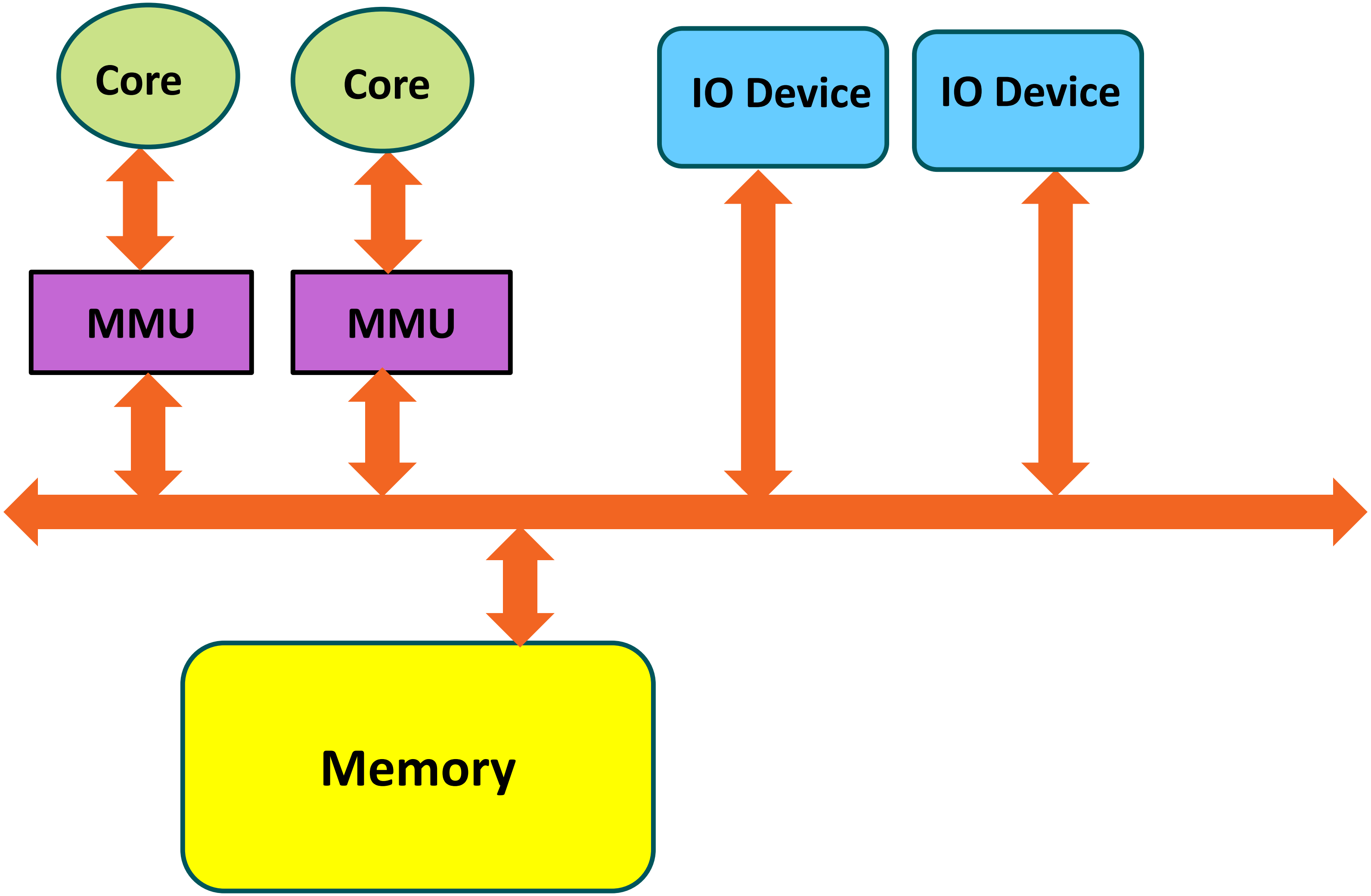
Hardware that intercepts DMA transactions and interrupts

Key capabilities:

1. Memory protection for DMA
2. Virtual address translation for DMA
3. Interrupt remapping and virtualization

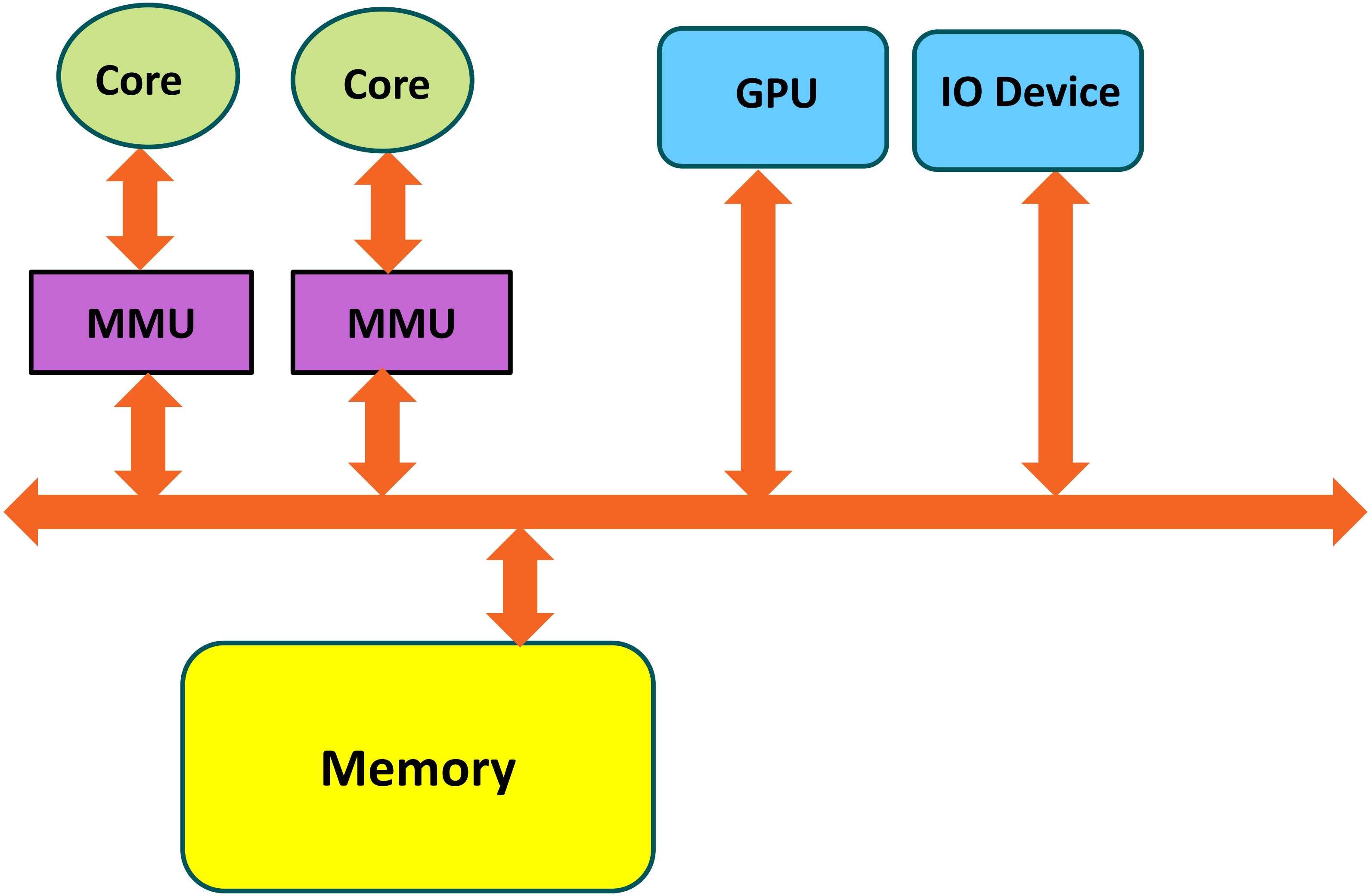
MOTIVATION: EMERGENCE OF HETEROGENEOUS SYSTEMS

HETEROGENEOUS SYSTEM ARCHITECTURE (HSA)



MOTIVATION: EMERGENCE OF HETEROGENEOUS SYSTEMS

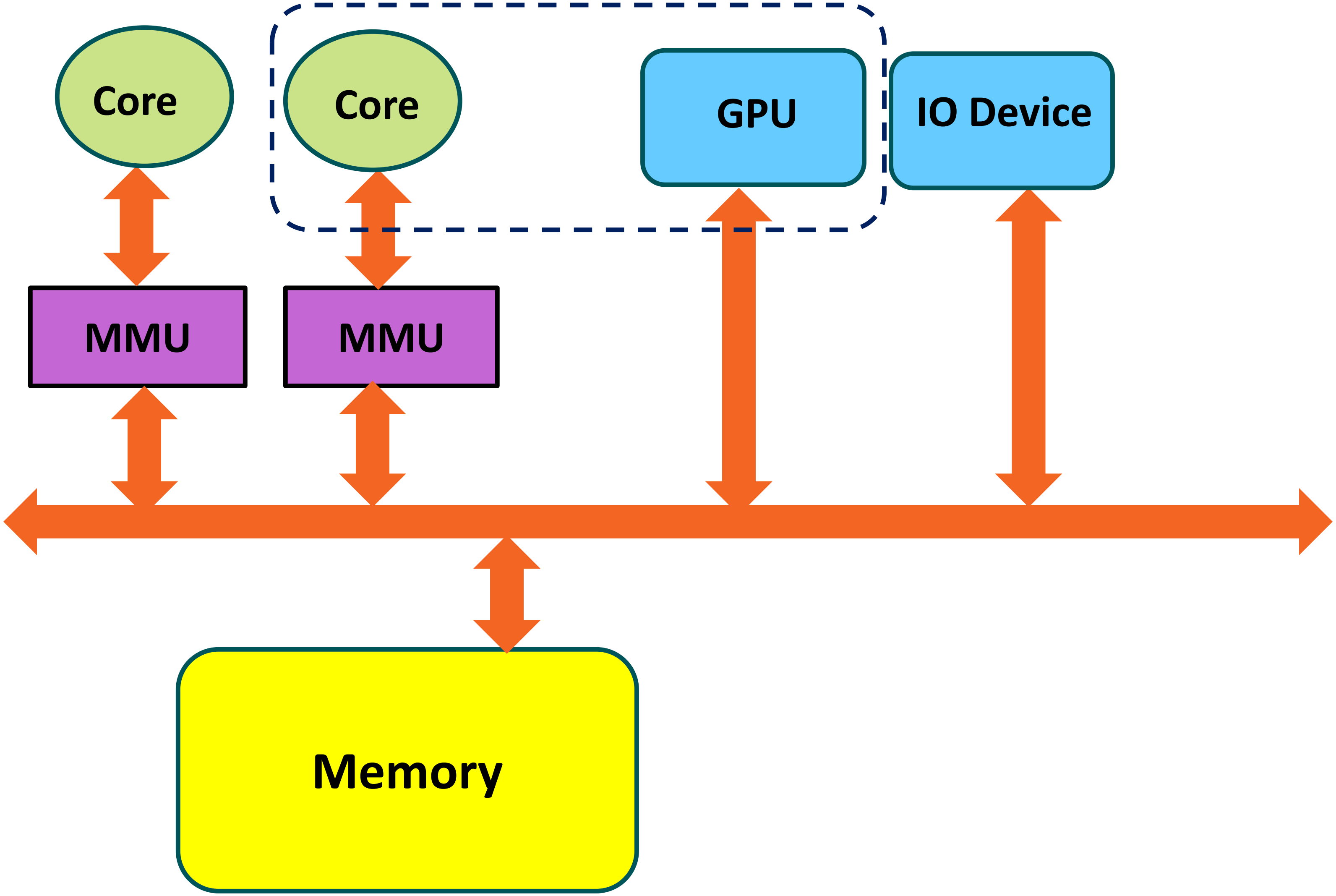
HETEROGENEOUS SYSTEM ARCHITECTURE (HSA)



MOTIVATION: EMERGENCE OF HETEROGENEOUS SYSTEMS

HETEROGENEOUS SYSTEM ARCHITECTURE (HSA)

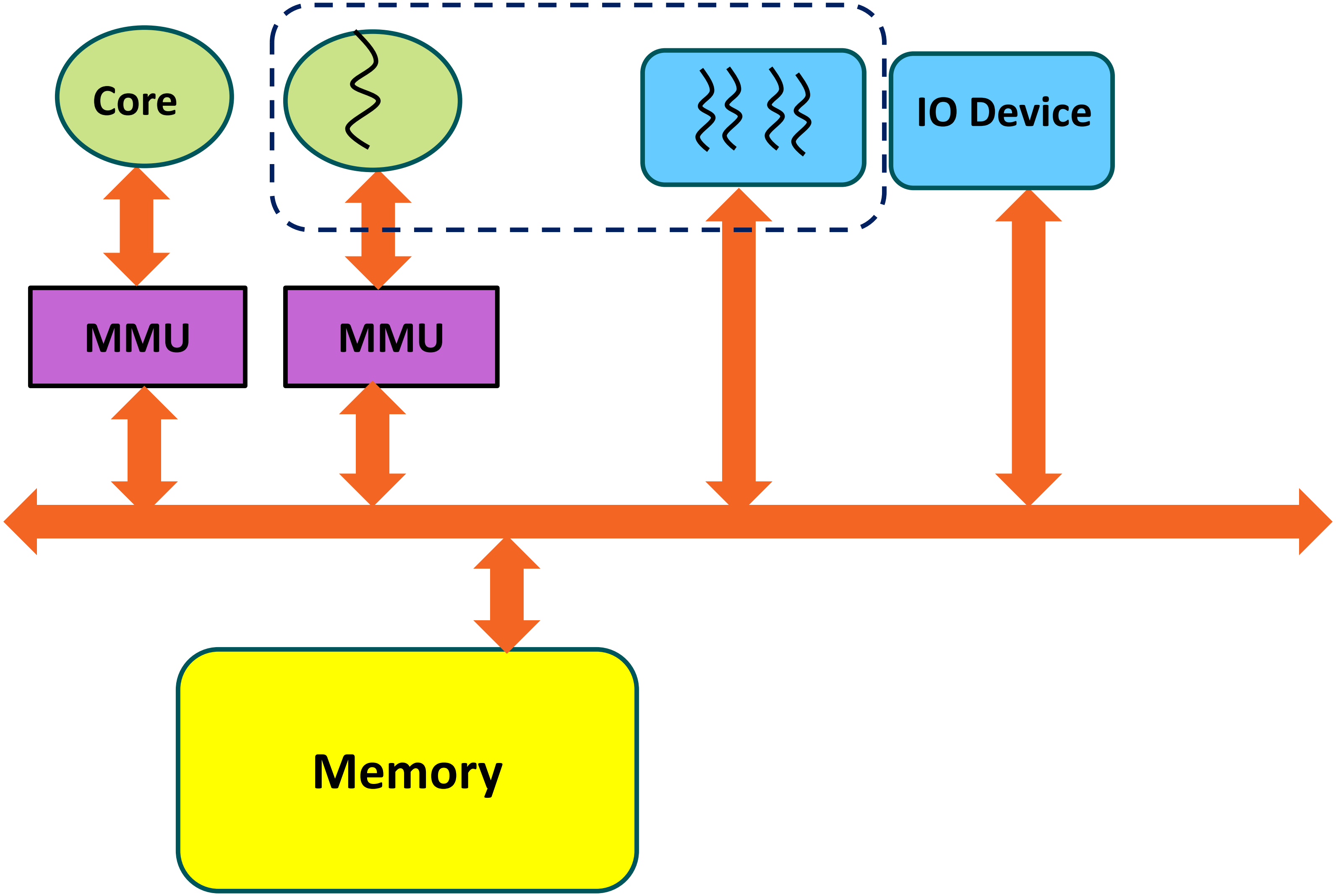
Shared virtual addressing is key to ease of programming



MOTIVATION: EMERGENCE OF HETEROGENEOUS SYSTEMS

HETEROGENEOUS SYSTEM ARCHITECTURE (HSA)

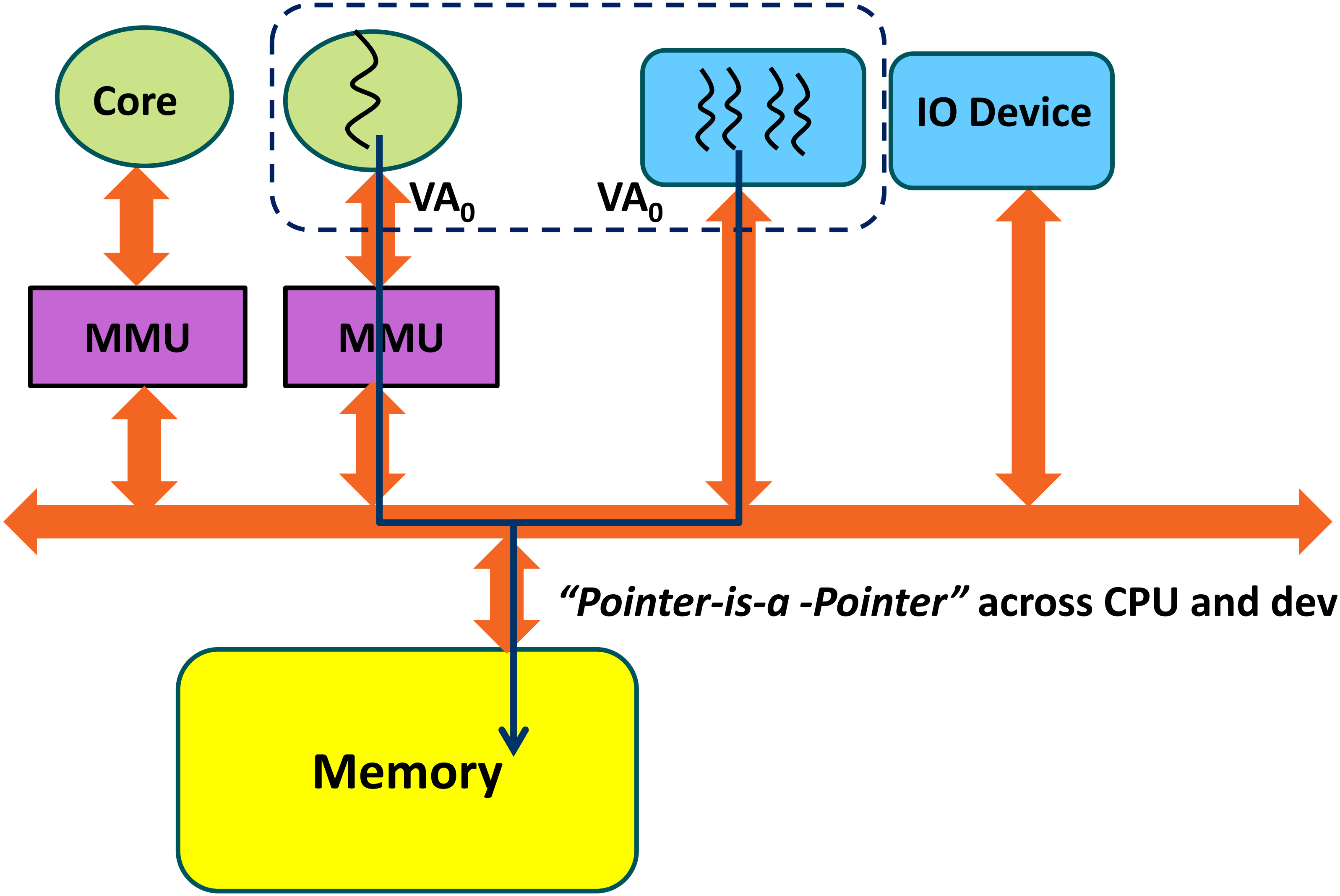
Shared virtual addressing is key to ease of programming



MOTIVATION: EMERGENCE OF HETEROGENEOUS SYSTEMS

HETEROGENEOUS SYSTEM ARCHITECTURE (HSA)

Shared virtual addressing is key to ease of programming

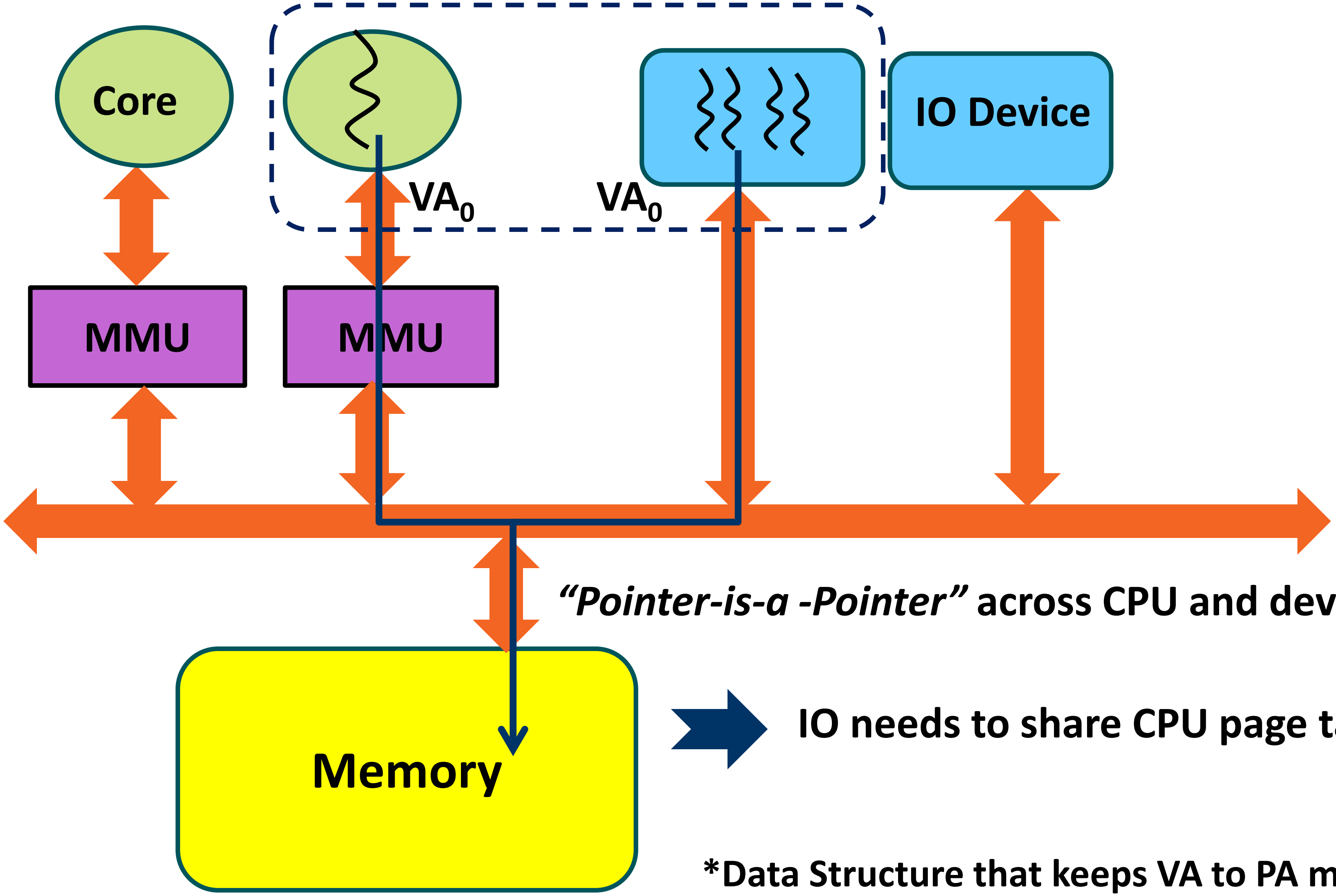


***"Pointer-is-a -Pointer"* across CPU and devices**

MOTIVATION: EMERGENCE OF HETEROGENEOUS SYSTEMS

HETEROGENEOUS SYSTEM ARCHITECTURE (HSA)

Shared virtual addressing is key to ease of programming



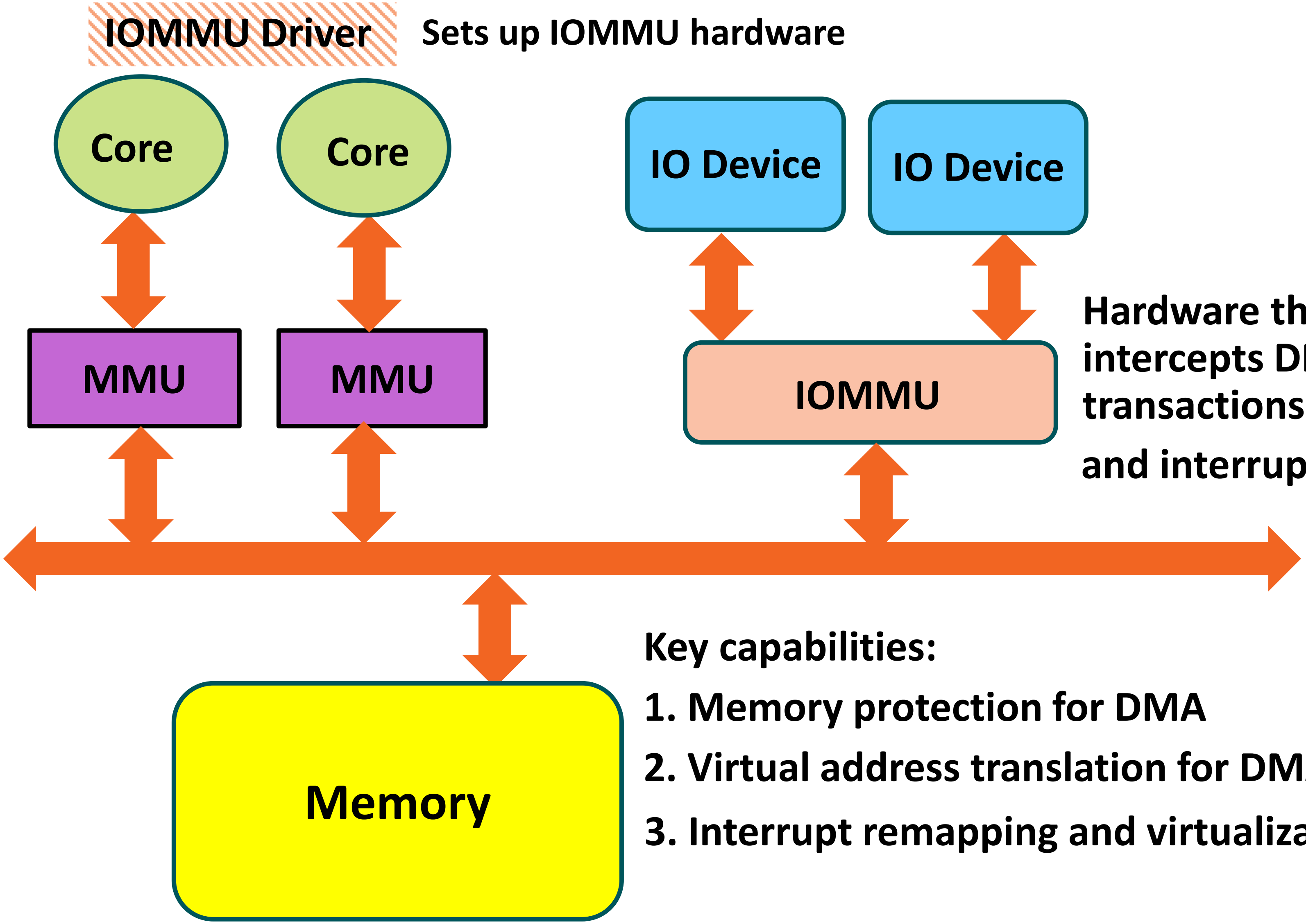
"Pointer-is-a -Pointer" **across CPU and devices**

IO needs to share CPU page table*

***Data Structure that keeps VA to PA mapping**

INTRODUCTION OF IOMMU: THE LOGICAL VIEW

ADDING ABILITY TO SHARE ADDRESS SPACE IN HETEROGENEOUS SYSTEM



Sets up IOMMU hardware

Hardware that intercepts DMA transactions

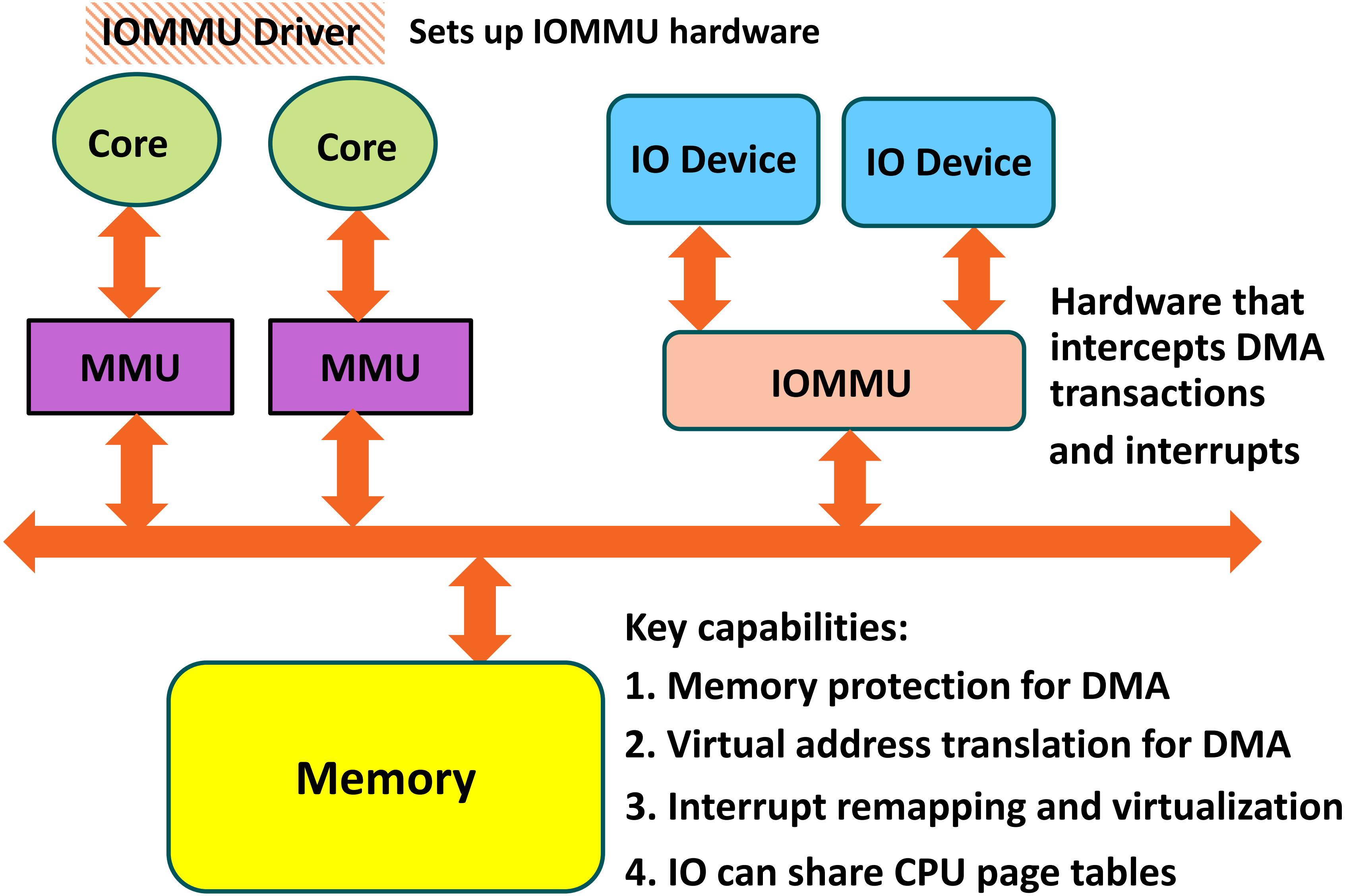
and interrupts

Key capabilities:

1. Memory protection for DMA
2. Virtual address translation for DMA
3. Interrupt remapping and virtualization

INTRODUCTION OF IOMMU: THE LOGICAL VIEW

ADDING ABILITY TO SHARE ADDRESS SPACE IN HETEROGENEOUS SYSTEM



Sets up IOMMU hardware

Hardware that intercepts DMA transactions

and interrupts

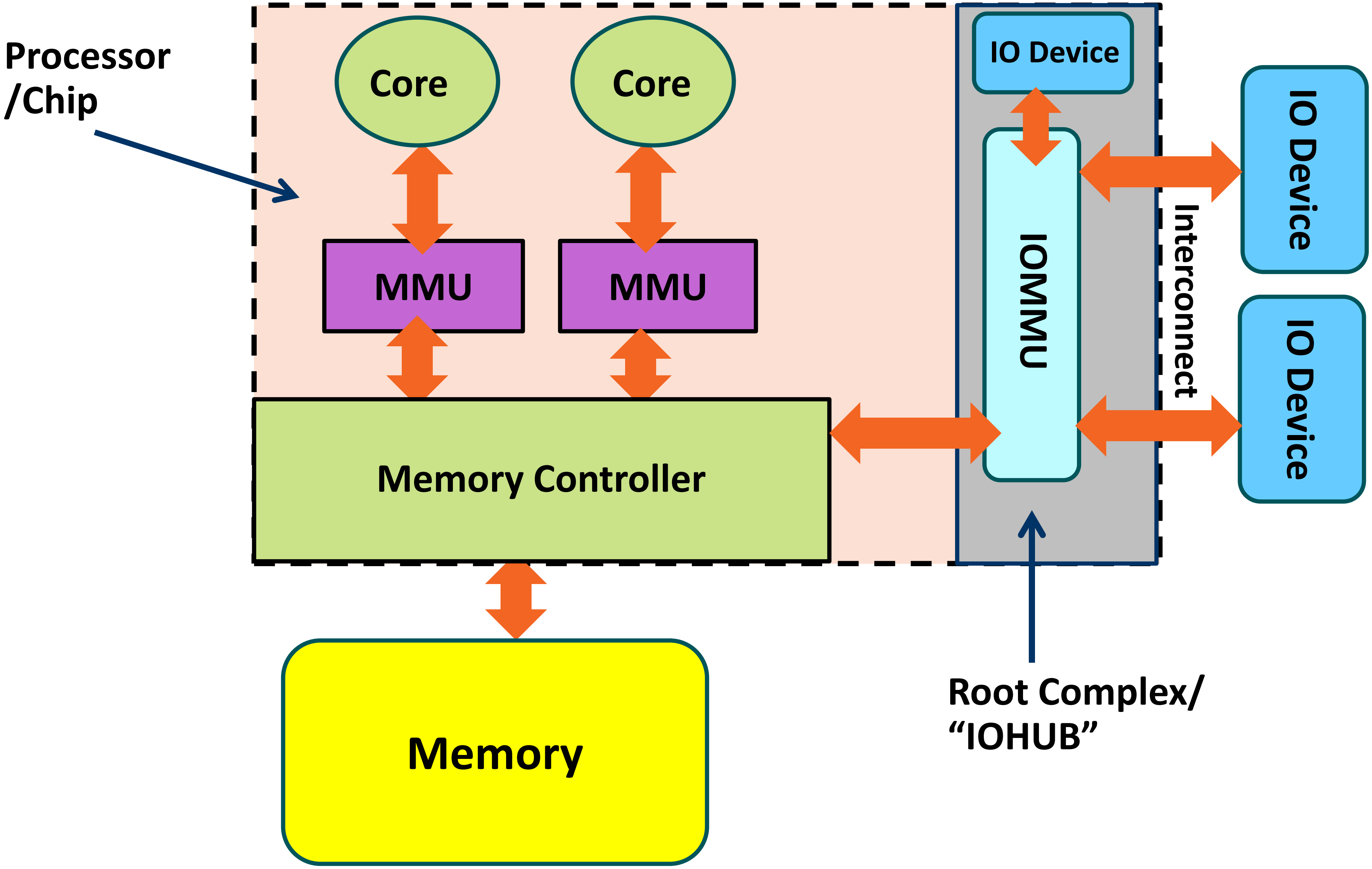
Key capabilities:

1. Memory protection for DMA
2. Virtual address translation for DMA
3. Interrupt remapping and virtualization
4. IO can share CPU page tables

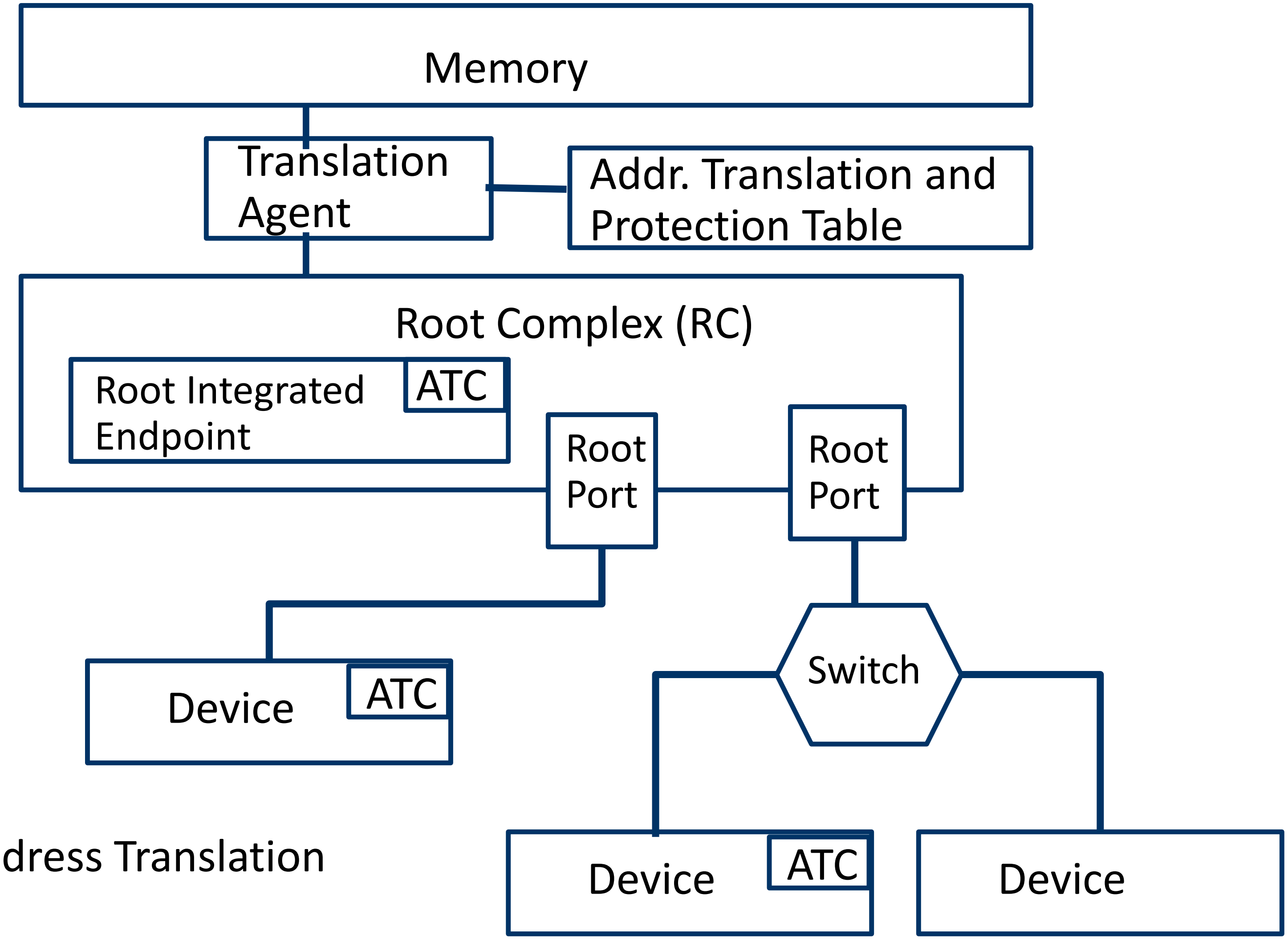
INTRODUCTION OF IOMMU: (TYPICAL) PHYSICAL VIEW

AMD

IOMMU IS PART OF PROCESSOR COMPLEX



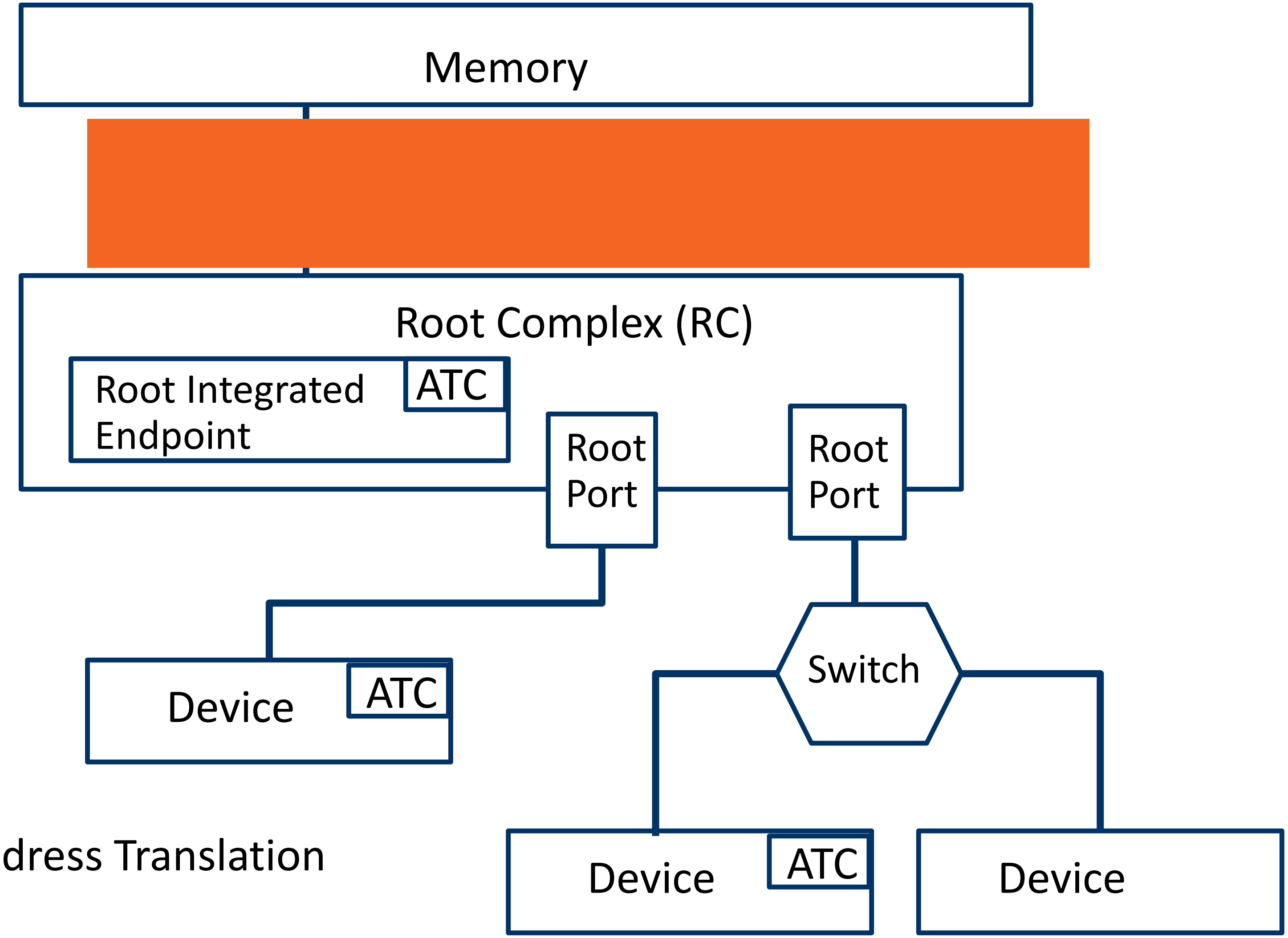
IOMMU FROM THE PERSPECTIVE OF DEVICE (PCIE® SPEC)



ATC – Address Translation Cache

IOMMU FROM THE PERSPECTIVE OF DEVICE (PCIE® SPEC)

IOMMU → Translation Agent and uses the Address Translation and Protection Table



ATC – Address Translation Cache

COMPARING CPU MMU AND IOMMU

AMD

	CPU MMU	IOMMU
Address Translation	VA → PA and GVA → GPA → SPA	VA → PA and GVA → GPA → SPA
Memory Protection	Read/Write etc.	Read/Write etc.
Interrupt Handling	No	Remapping and Virtualization Support
Parallelism	Mostly Single Threaded	Highly Multithreaded
Page Faults, Events, etc.	Synchronous Handling	Asynchronous Handling

HISTORY

A SIMPLIFIED VIEW

V1, c. 2004

Technology created to translate and vet memory accesses by peripherals, replacing software

V1.2, c. 2006

Interrupt remapping added for IO virtualization

V2, c. 2008

Nested paging, interrupt virtualization, and improved management features added

V3, c. 2010

Features added for full heterogeneous computing and further efficiencies

Whither next?

IOMMU TECHNOLOGY FAMILIES

REFERENCES

AMD IOMMU®	IO Memory Management Unit
Intel VT-d®	Virtualization Technology for Directed IO
ARM SMM®	System Memory Management Unit
IBM CAPI®	Coherent Accelerator Processor Interface

AGENDA

AMD

USE CASES & DEMOSTRATION

Where can IOMMU help?

INTERNALS

How does IOMMU work?

RESEARCH

Research Opportunities and Tools

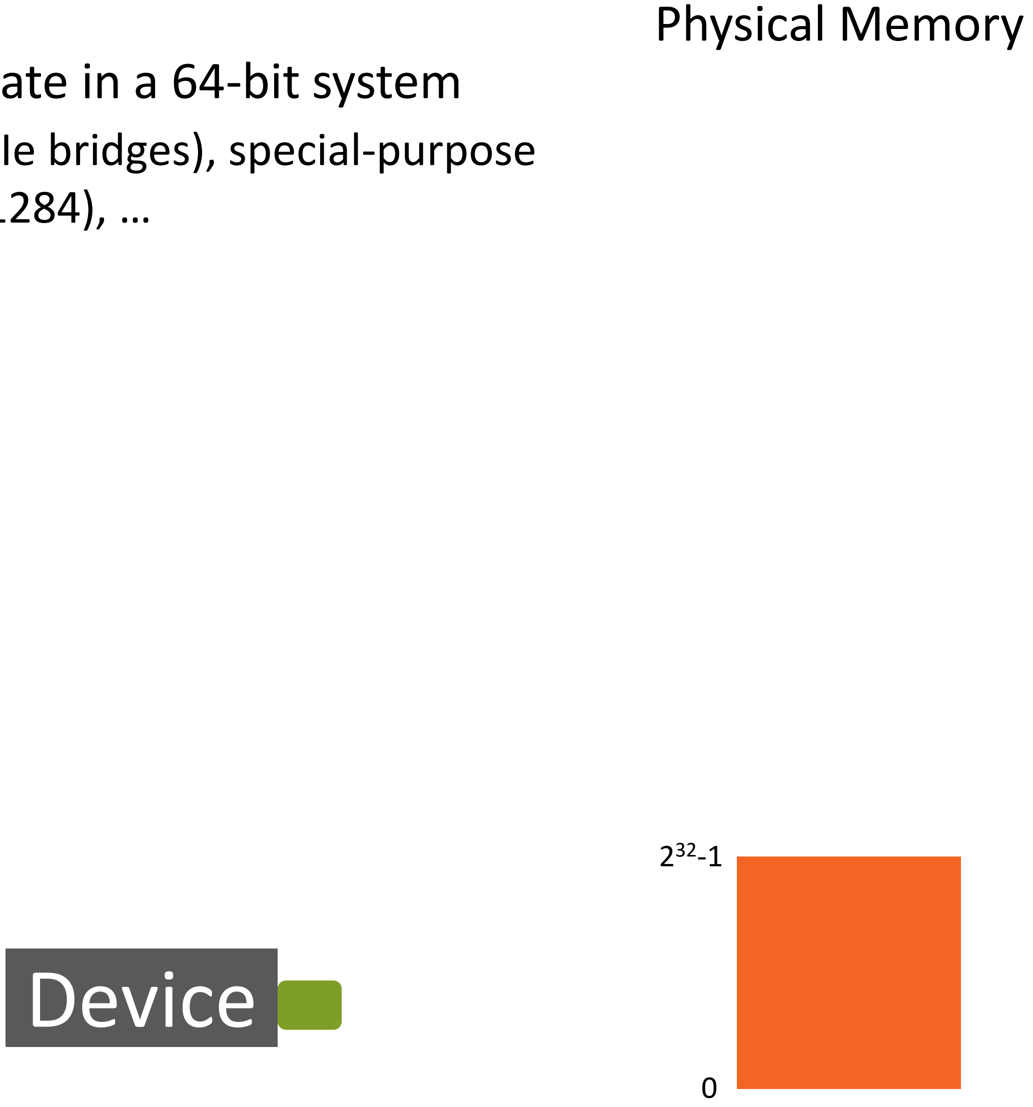
FIVE USE CASES OF IOMMU

LEGACY I/O	Supporting legacy devices – Extending DMA “beyond reach”
SECURITY AND PROTECTION	Preventing uncontrolled memory access
SECURE BOOT	Enforcing secure boot
DIRECT I/O DEVICES	Secure and efficient IO from Guest OS
HETEROGENEOUS COMPUTING	Enabling shared virtual memory

SUPPORTING LEGACY DEVICES

HOW CAN AN IOMMU HELP?

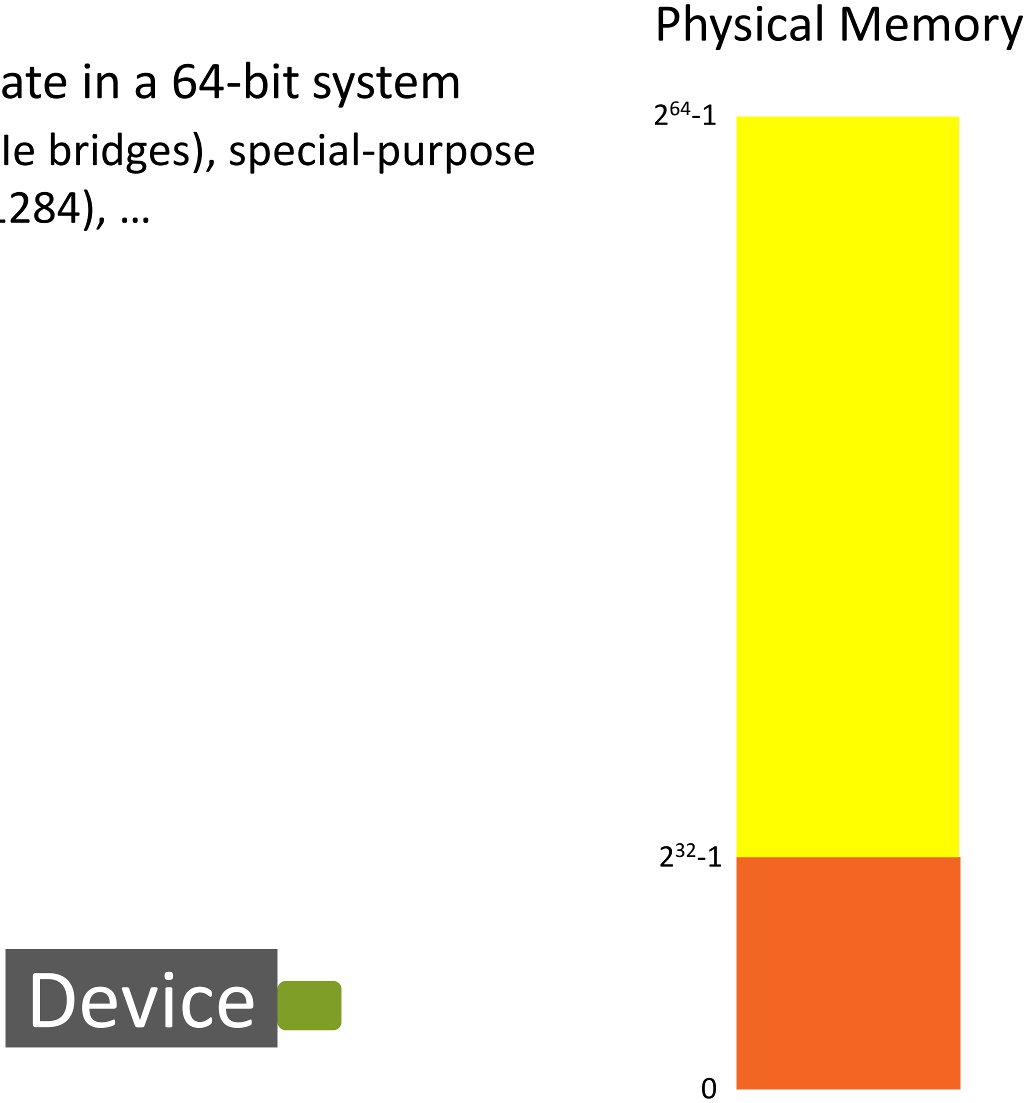
- Many 32-bit DMA devices operate in a 64-bit system
  - Older PCI cards (through PCI-PCIe bridges), special-purpose controllers, parallel ports (IEEE-1284), …



SUPPORTING LEGACY DEVICES

HOW CAN AN IOMMU HELP?

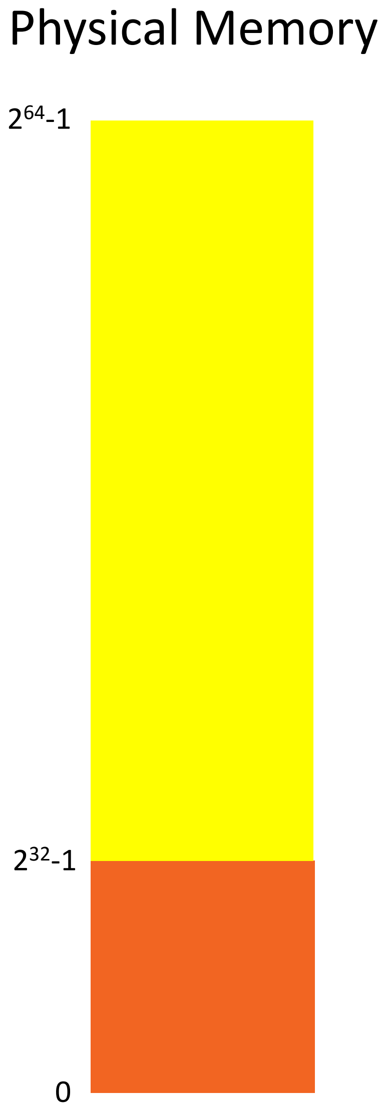
- Many 32-bit DMA devices operate in a 64-bit system
  - Older PCI cards (through PCI-PCIe bridges), special-purpose controllers, parallel ports (IEEE-1284), …



SUPPORTING LEGACY DEVICES

HOW CAN AN IOMMU HELP?

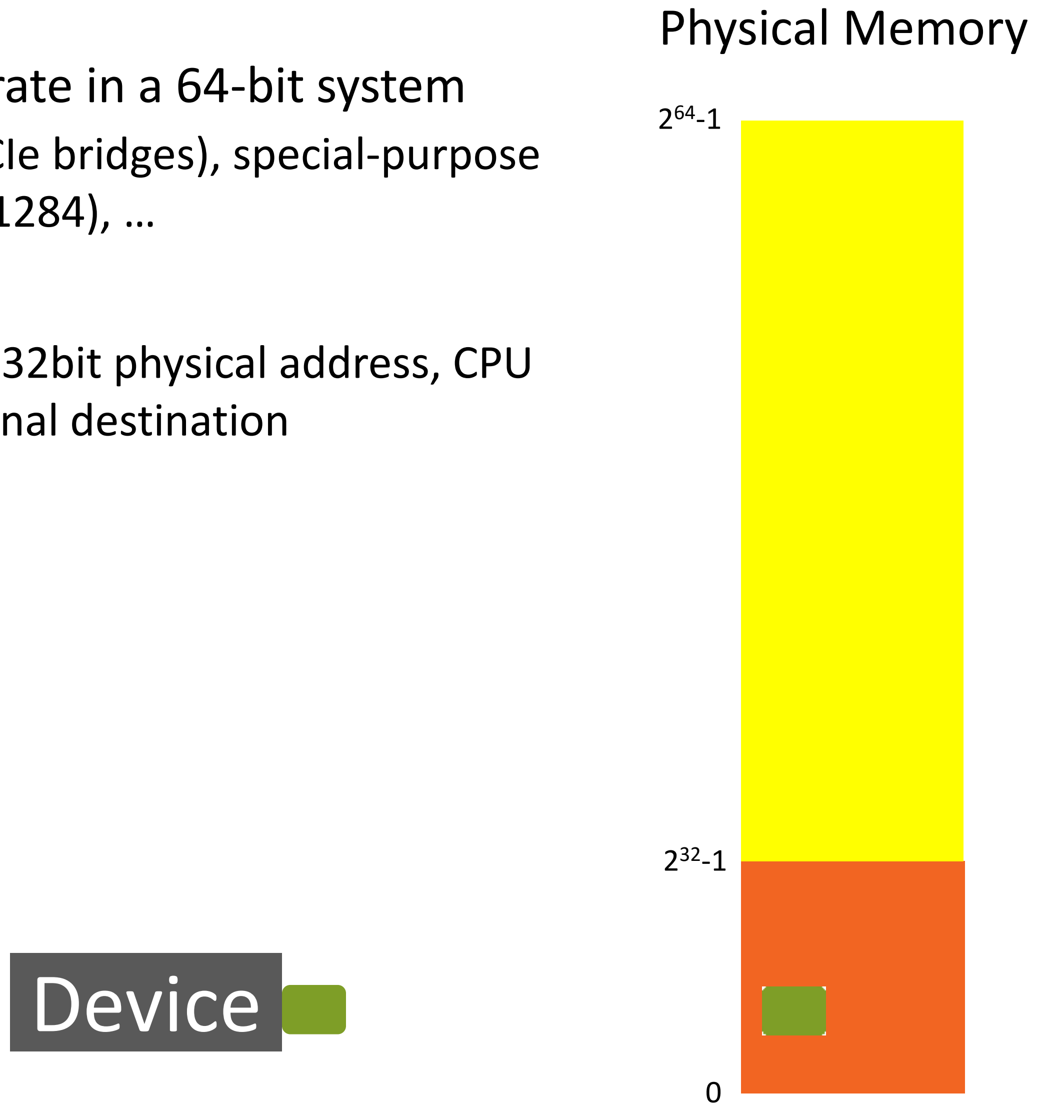
- Many 32-bit DMA devices operate in a 64-bit system
  - Older PCI cards (through PCI-PCIe bridges), special-purpose controllers, parallel ports (IEEE-1284), …
- SW Solution: Bounce buffers
  - Device does DMA to a region in 32bit physical address, CPU copies data from buffer to the final destination



SUPPORTING LEGACY DEVICES

HOW CAN AN IOMMU HELP?

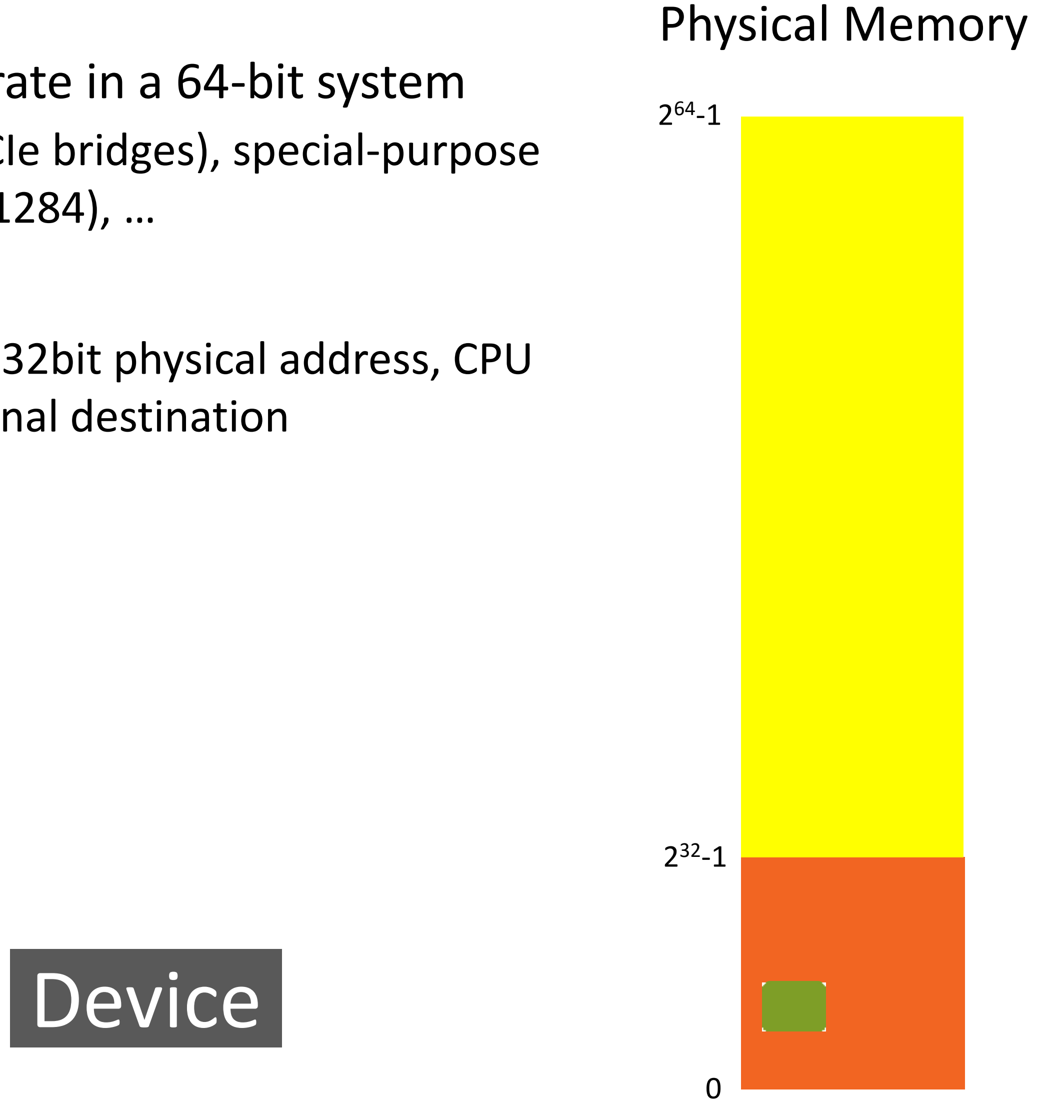
- Many 32-bit DMA devices operate in a 64-bit system
  - Older PCI cards (through PCI-PCIe bridges), special-purpose controllers, parallel ports (IEEE-1284), …
- SW Solution: Bounce buffers
  - Device does DMA to a region in 32bit physical address, CPU copies data from buffer to the final destination



SUPPORTING LEGACY DEVICES

HOW CAN AN IOMMU HELP?

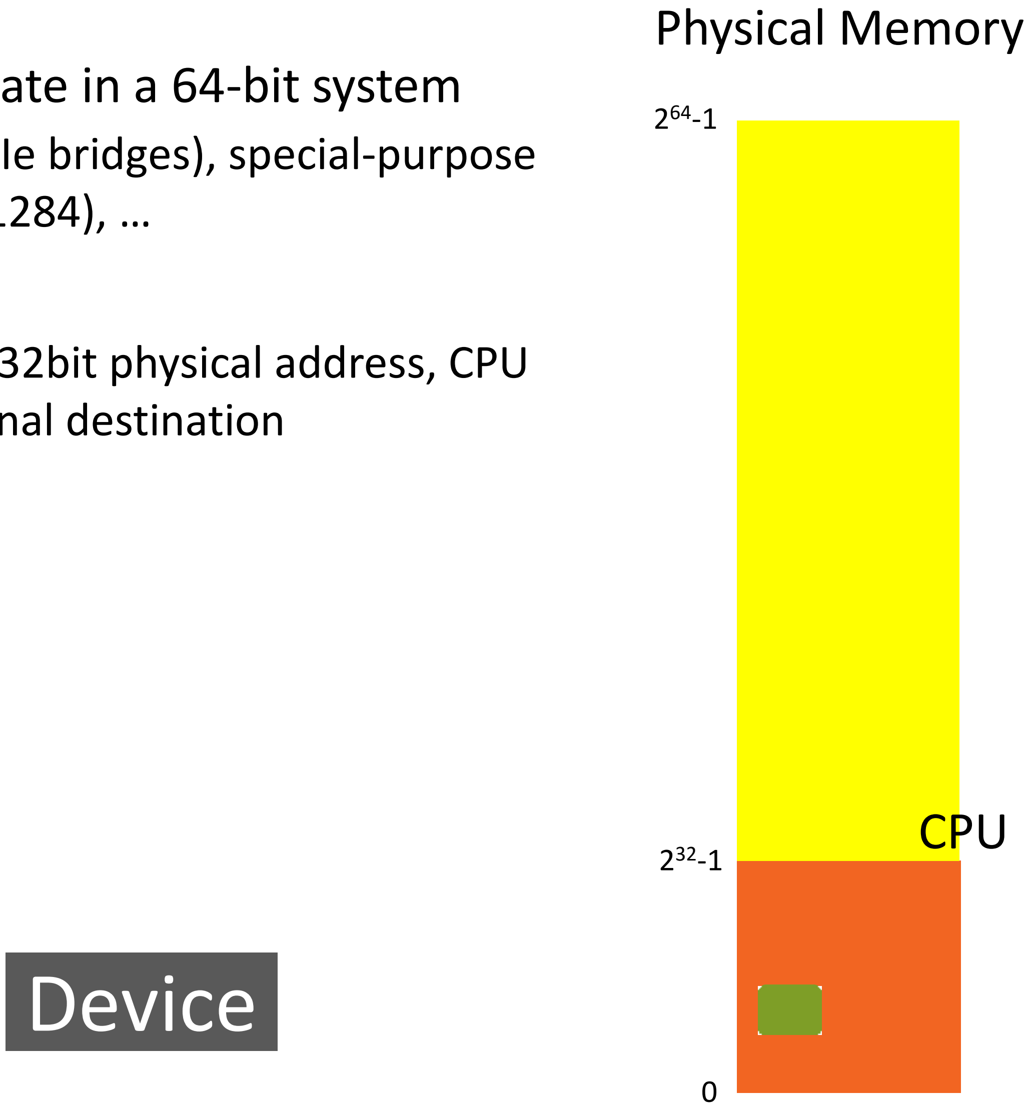
- Many 32-bit DMA devices operate in a 64-bit system
  - Older PCI cards (through PCI-PCIe bridges), special-purpose controllers, parallel ports (IEEE-1284), …
- SW Solution: Bounce buffers
  - Device does DMA to a region in 32bit physical address, CPU copies data from buffer to the final destination



SUPPORTING LEGACY DEVICES

HOW CAN AN IOMMU HELP?

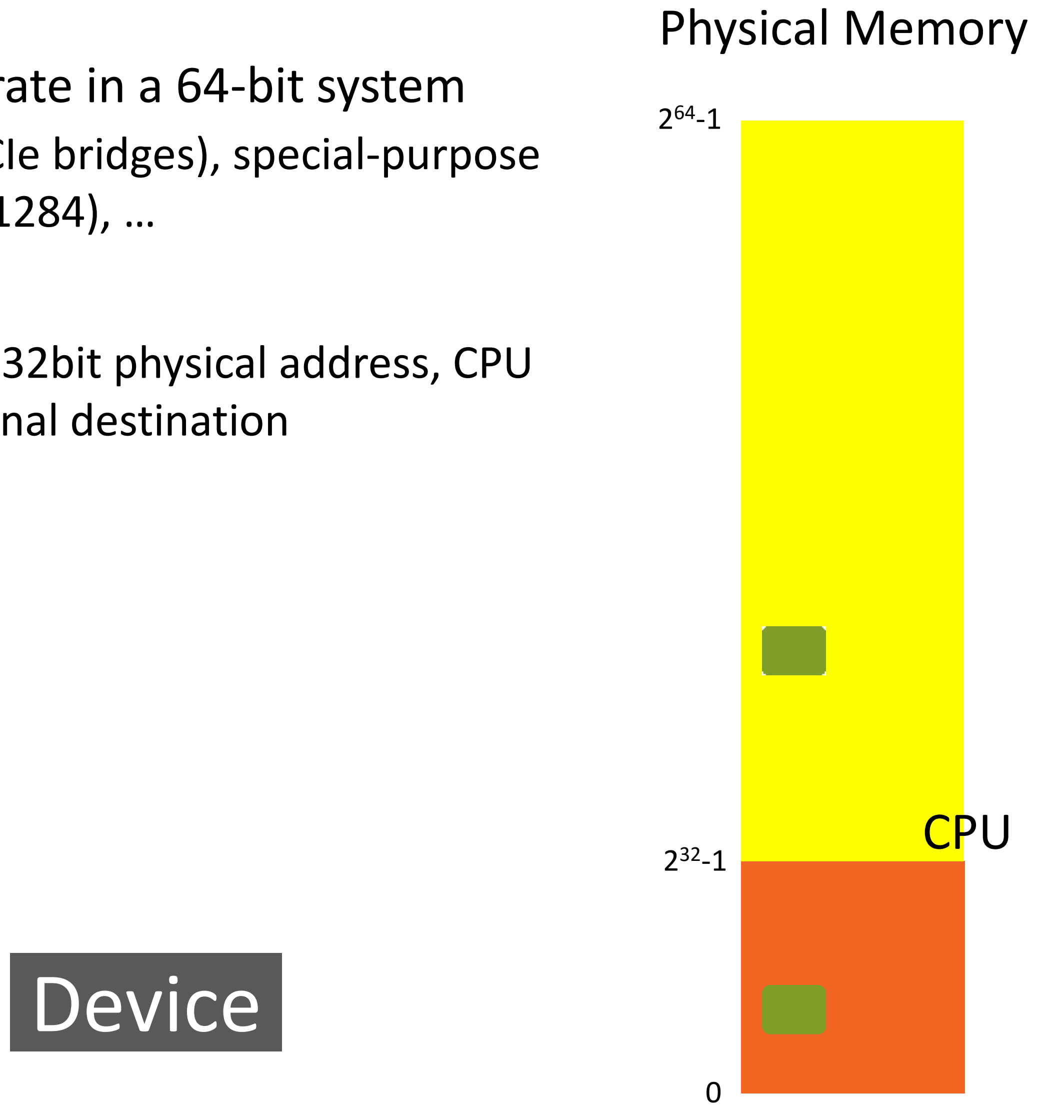
- Many 32-bit DMA devices operate in a 64-bit system
  - Older PCI cards (through PCI-PCIe bridges), special-purpose controllers, parallel ports (IEEE-1284), …
- SW Solution: Bounce buffers
  - Device does DMA to a region in 32bit physical address, CPU copies data from buffer to the final destination



SUPPORTING LEGACY DEVICES

HOW CAN AN IOMMU HELP?

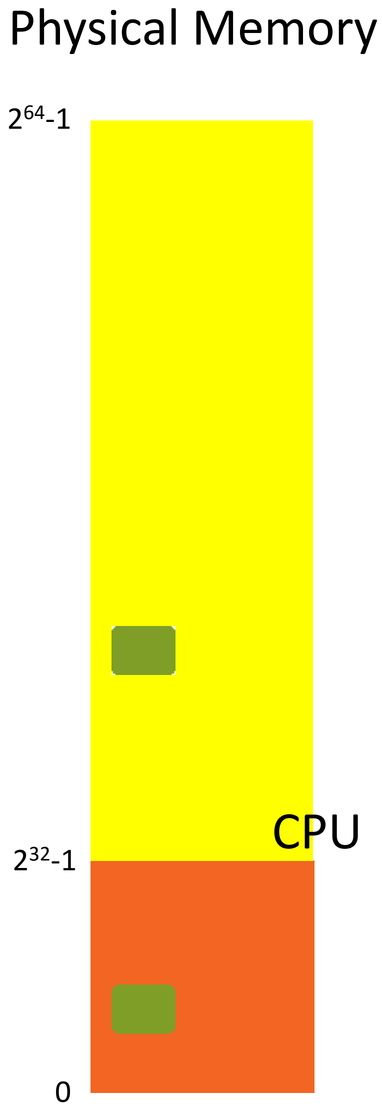
- Many 32-bit DMA devices operate in a 64-bit system
  - Older PCI cards (through PCI-PCIe bridges), special-purpose controllers, parallel ports (IEEE-1284), …
- SW Solution: Bounce buffers
  - Device does DMA to a region in 32bit physical address, CPU copies data from buffer to the final destination



SUPPORTING LEGACY DEVICES

HOW CAN AN IOMMU HELP?

- Many 32-bit DMA devices operate in a 64-bit system
  - Older PCI cards (through PCI-PCIe bridges), special-purpose controllers, parallel ports (IEEE-1284), …
- SW Solution: Bounce buffers
  - Device does DMA to a region in 32bit physical address, CPU copies data from buffer to the final destination
  - Slow, needs SW synchronization, ties up CPU core

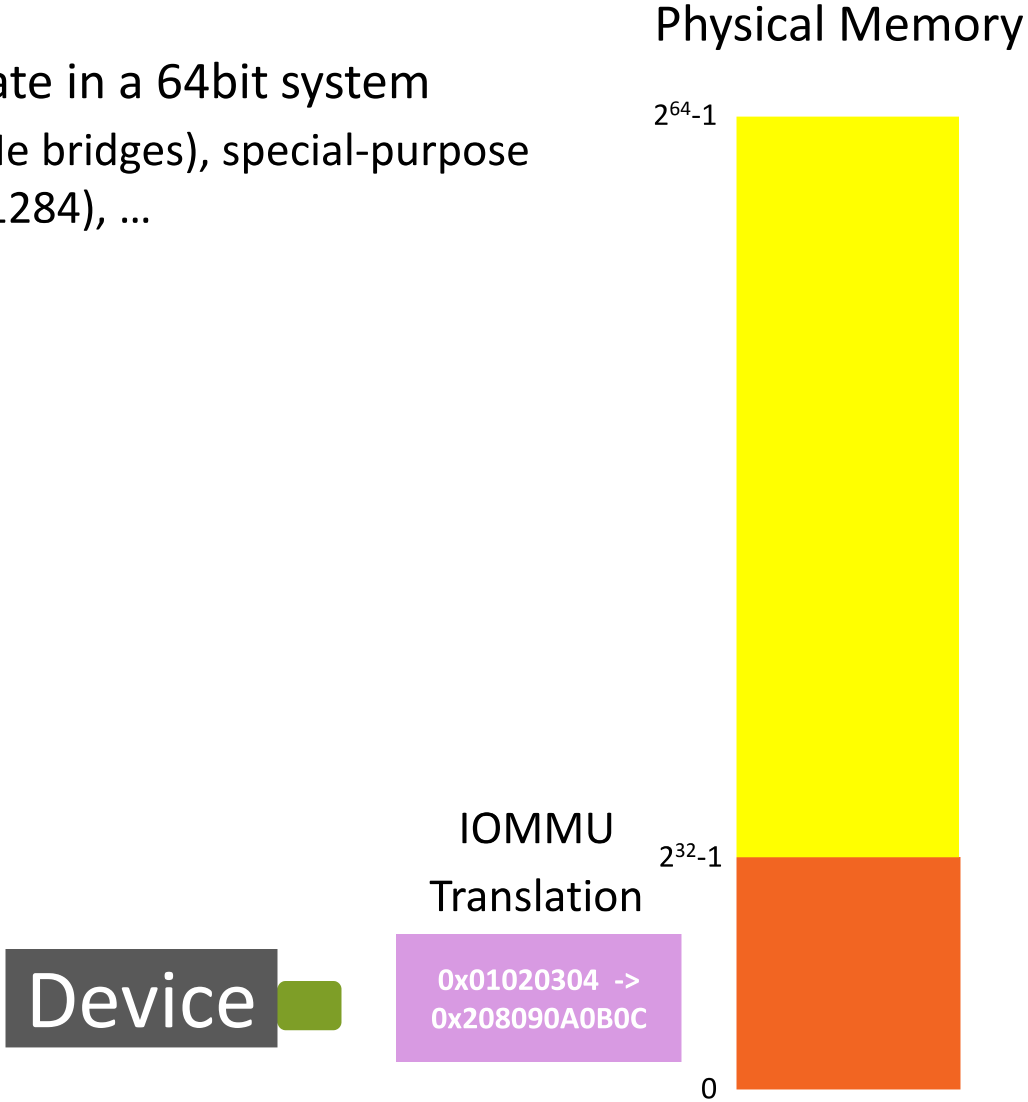


Device

SUPPORTING LEGACY DEVICES

HOW CAN AN IOMMU HELP?

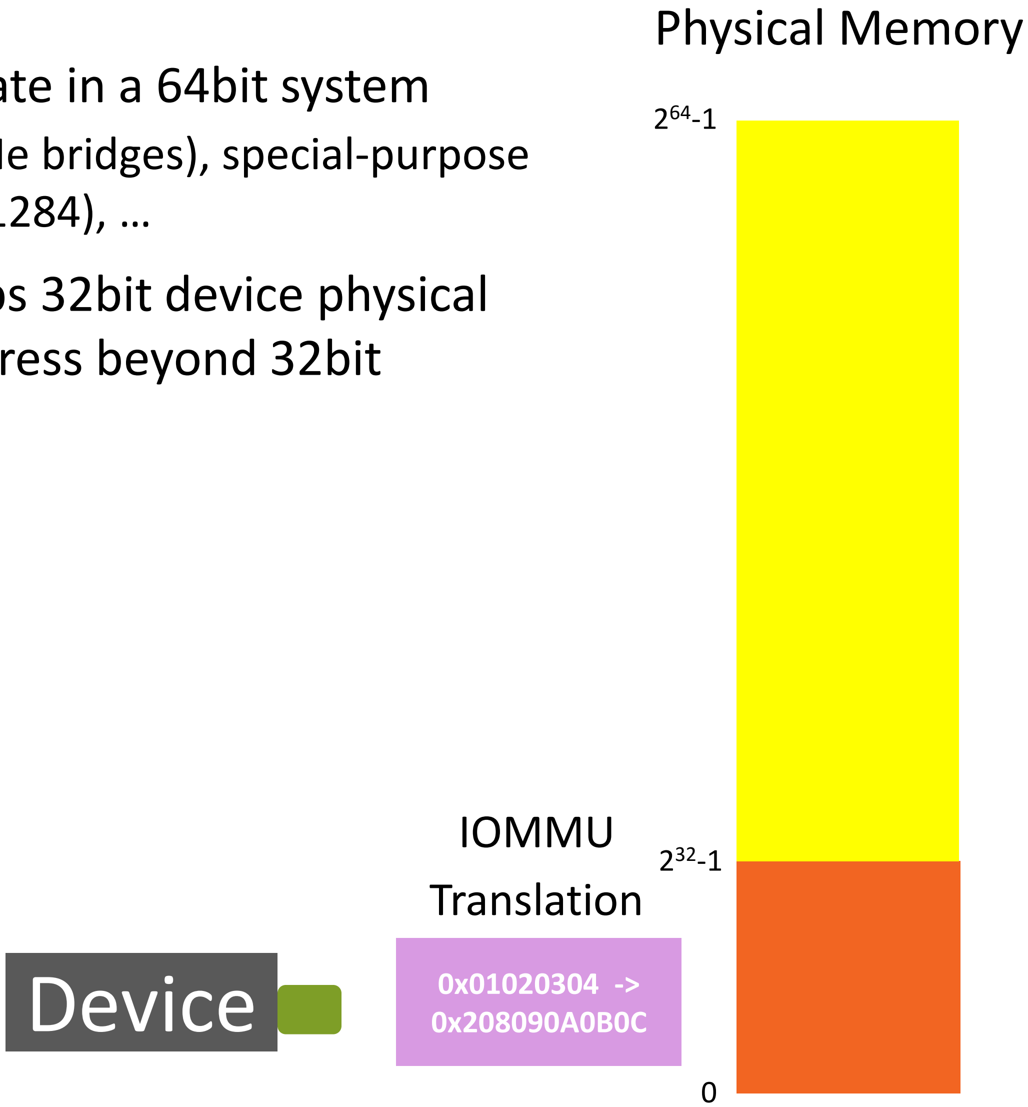
- Many 32bit DMA devices operate in a 64bit system
  - older PCI cards (through PCI-PCIe bridges), special-purpose controllers, parallel ports (IEEE-1284), …



SUPPORTING LEGACY DEVICES

HOW CAN AN IOMMU HELP?

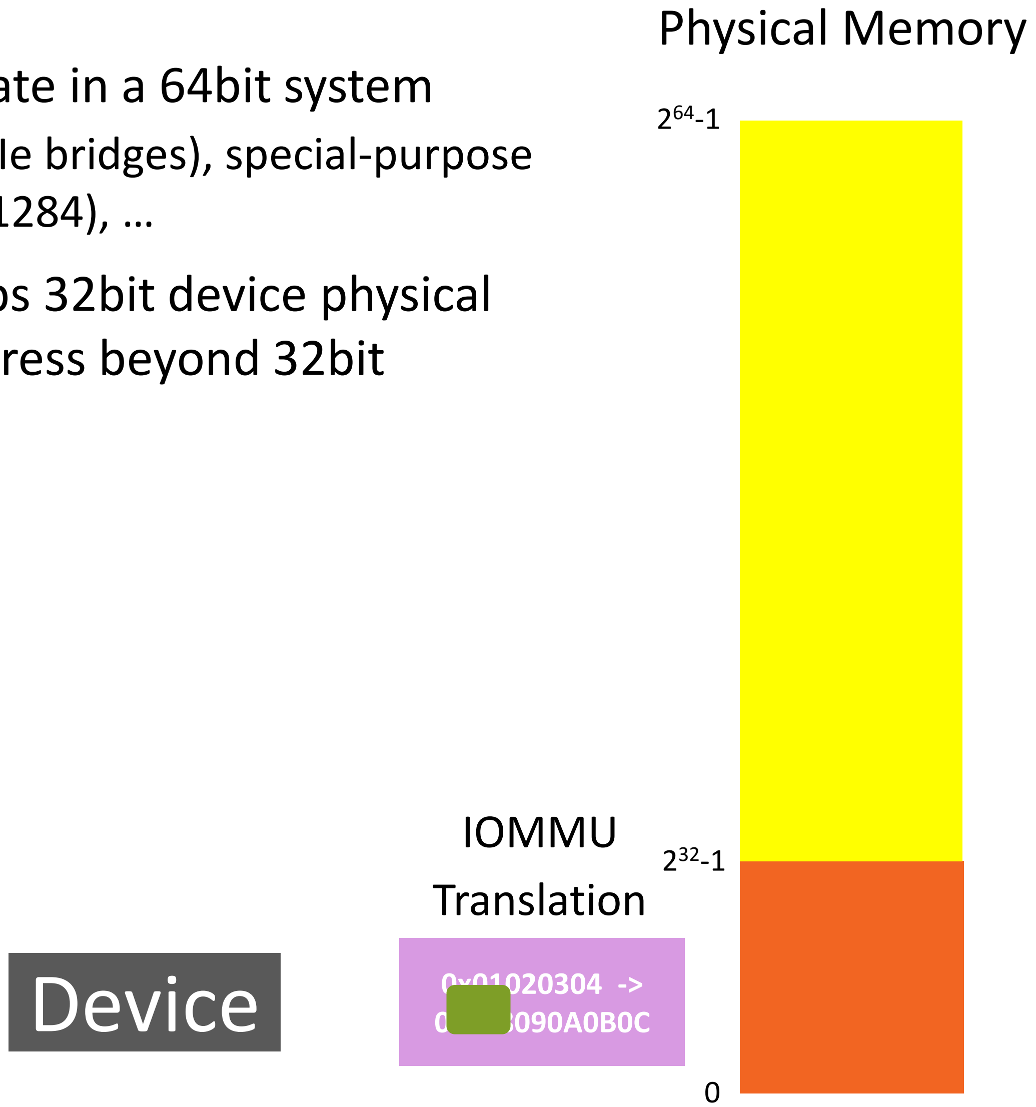
- Many 32bit DMA devices operate in a 64bit system
  - older PCI cards (through PCI-PCIe bridges), special-purpose controllers, parallel ports (IEEE-1284), …
- Better solution: IOMMU remaps 32bit device physical address to system physical address beyond 32bit



SUPPORTING LEGACY DEVICES

HOW CAN AN IOMMU HELP?

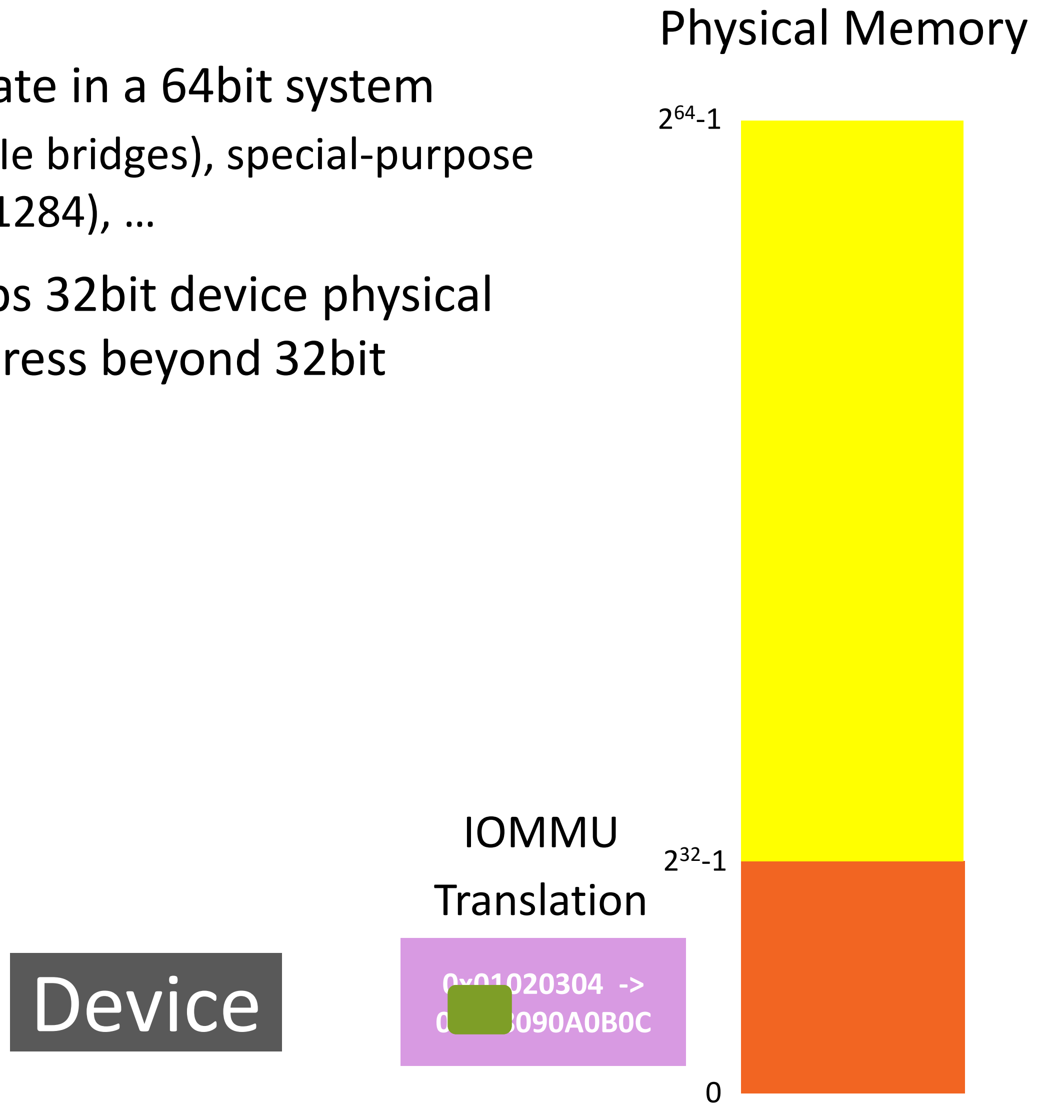
- Many 32bit DMA devices operate in a 64bit system
  - older PCI cards (through PCI-PCIe bridges), special-purpose controllers, parallel ports (IEEE-1284), …
- Better solution: IOMMU remaps 32bit device physical address to system physical address beyond 32bit



SUPPORTING LEGACY DEVICES

HOW CAN AN IOMMU HELP?

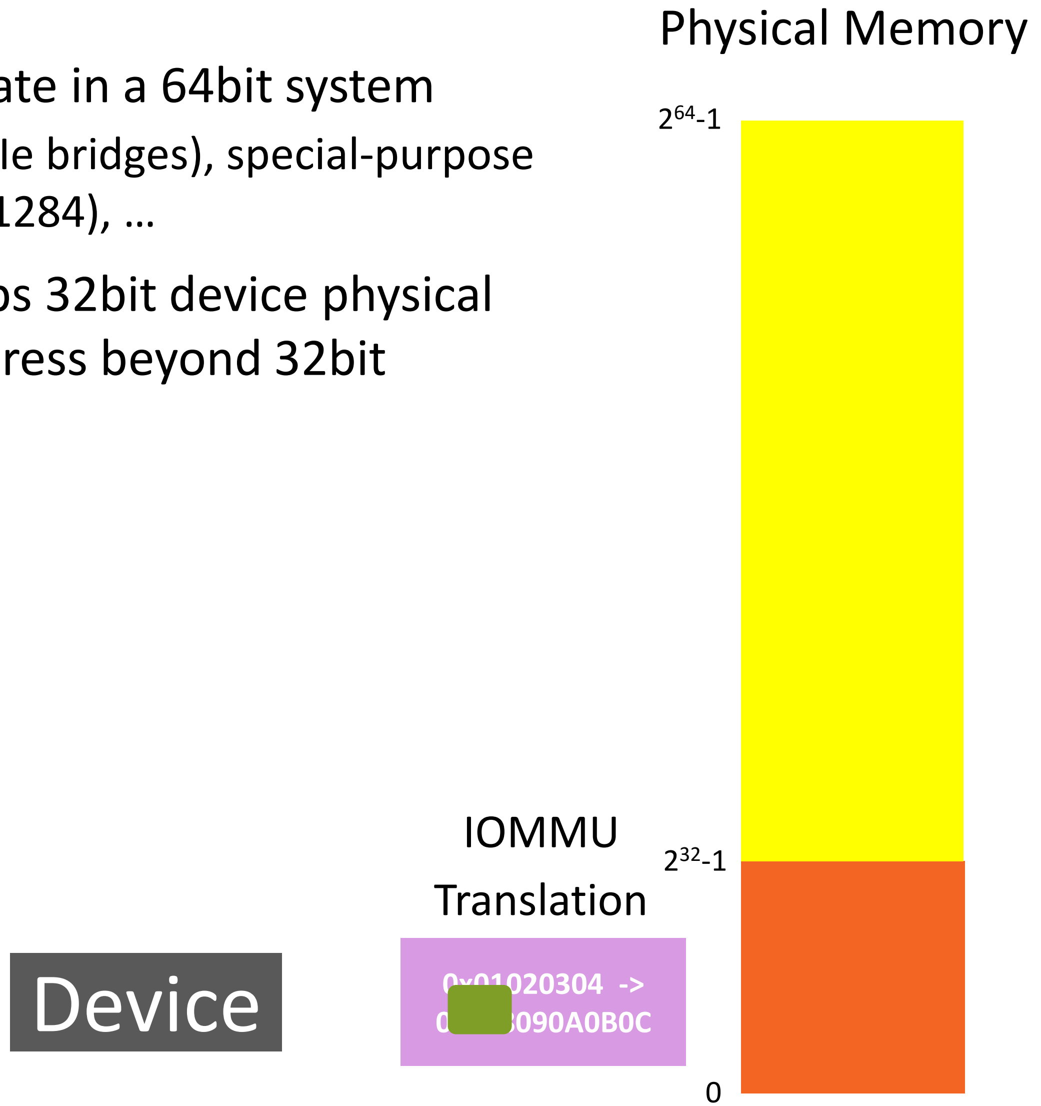
- Many 32bit DMA devices operate in a 64bit system
  - older PCI cards (through PCI-PCIe bridges), special-purpose controllers, parallel ports (IEEE-1284), …
- Better solution: IOMMU remaps 32bit device physical address to system physical address beyond 32bit



SUPPORTING LEGACY DEVICES

HOW CAN AN IOMMU HELP?

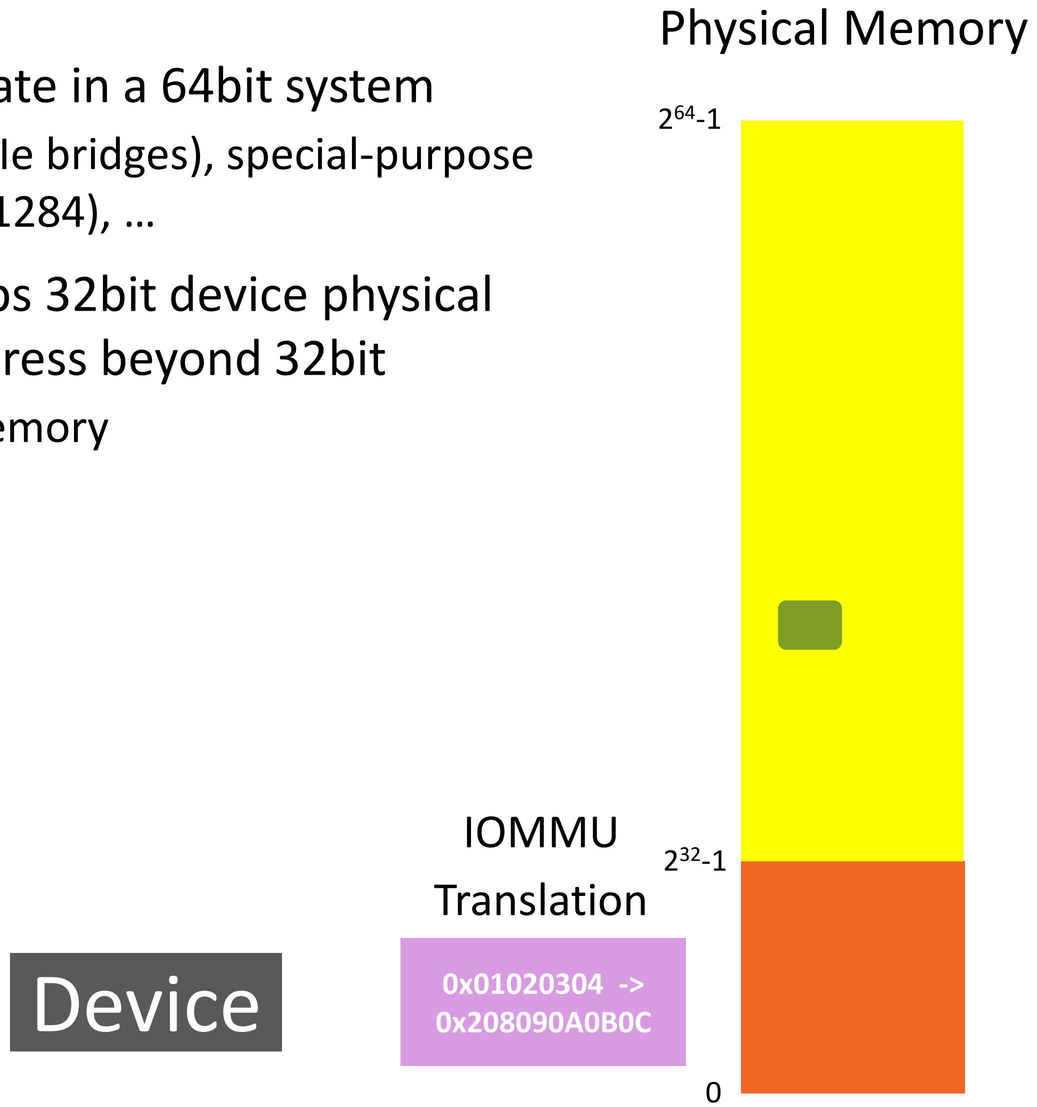
- Many 32bit DMA devices operate in a 64bit system
  - older PCI cards (through PCI-PCIe bridges), special-purpose controllers, parallel ports (IEEE-1284), …
- Better solution: IOMMU remaps 32bit device physical address to system physical address beyond 32bit



SUPPORTING LEGACY DEVICES

HOW CAN AN IOMMU HELP?

- Many 32bit DMA devices operate in a 64bit system
  - older PCI cards (through PCI-PCIe bridges), special-purpose controllers, parallel ports (IEEE-1284), …
- Better solution: IOMMU remaps 32bit device physical address to system physical address beyond 32bit
  - DMA goes directly into 64bit memory
  - No CPU transfer
  - More efficient



Physical Memory

2^{64}-1

2^{32}-1

0

IOMMU Translation

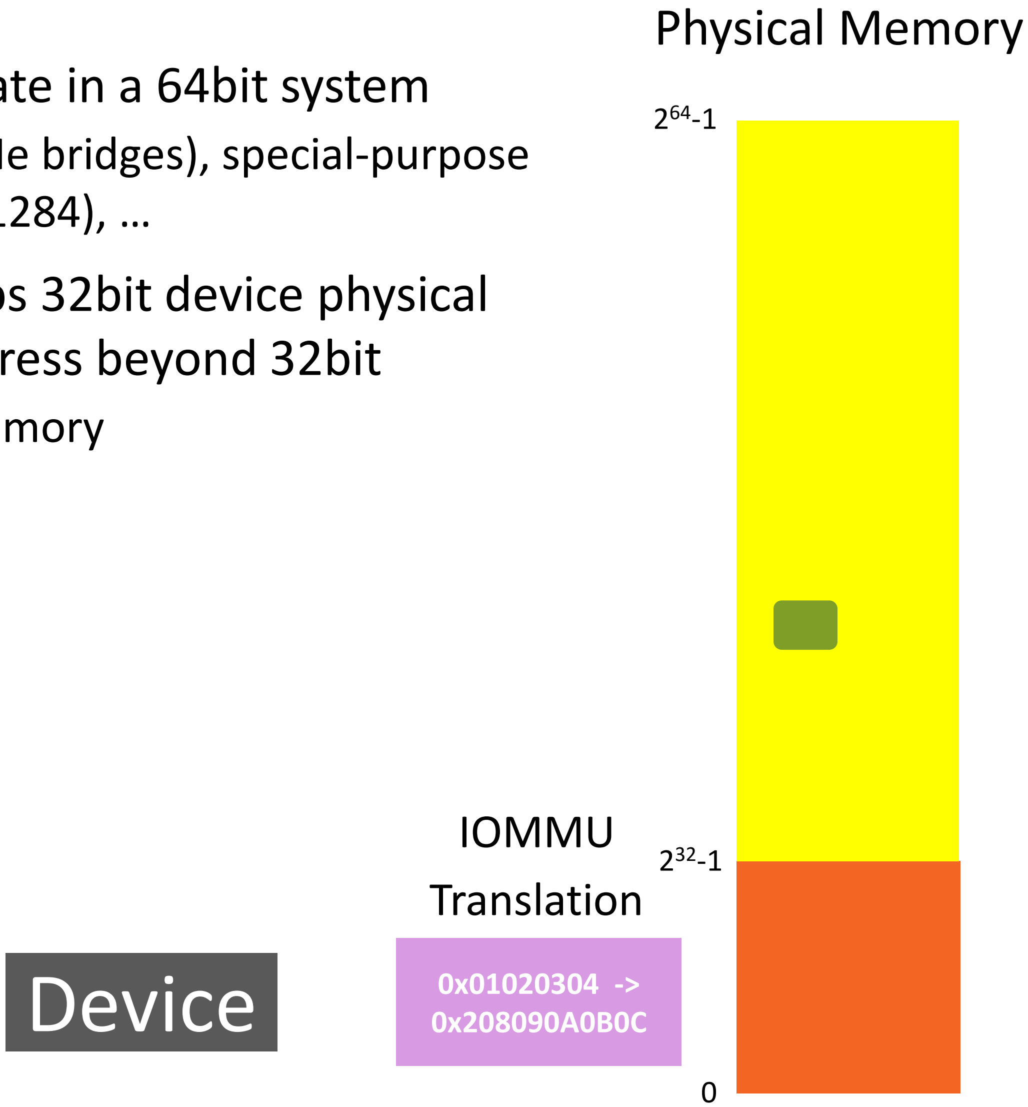
0x01020304 -> 0x208090A0B0C

Device

SUPPORTING LEGACY DEVICES

HOW CAN AN IOMMU HELP?

- Many 32bit DMA devices operate in a 64bit system
  - older PCI cards (through PCI-PCIe bridges), special-purpose controllers, parallel ports (IEEE-1284), …
- Better solution: IOMMU remaps 32bit device physical address to system physical address beyond 32bit
  - DMA goes directly into 64bit memory
  - No CPU transfer
  - More efficient
- Linux: DMA redirect feature



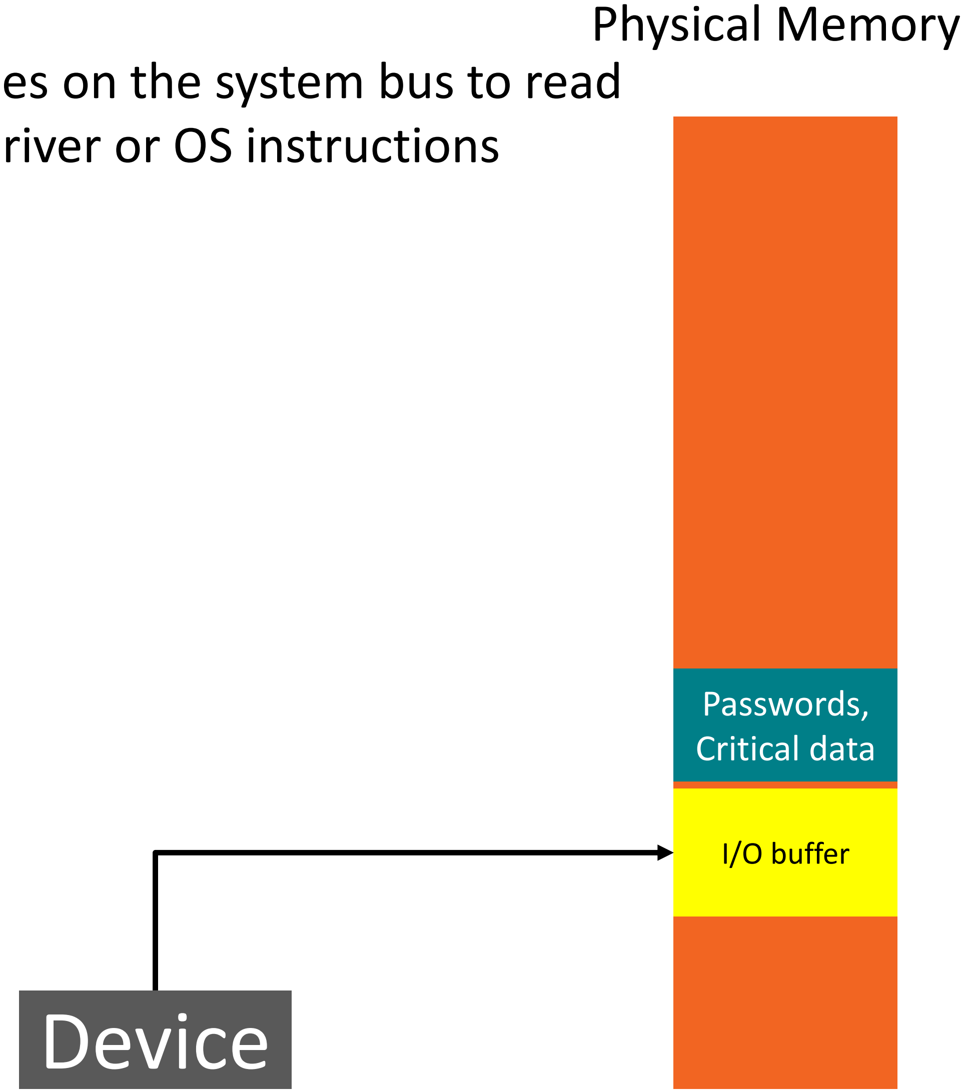
IOMMU USECASE: SECURITY AND PROTECTION

SECURE BOOT

SECURITY AND PROTECTION

THE TRADITIONAL IOMMU USE

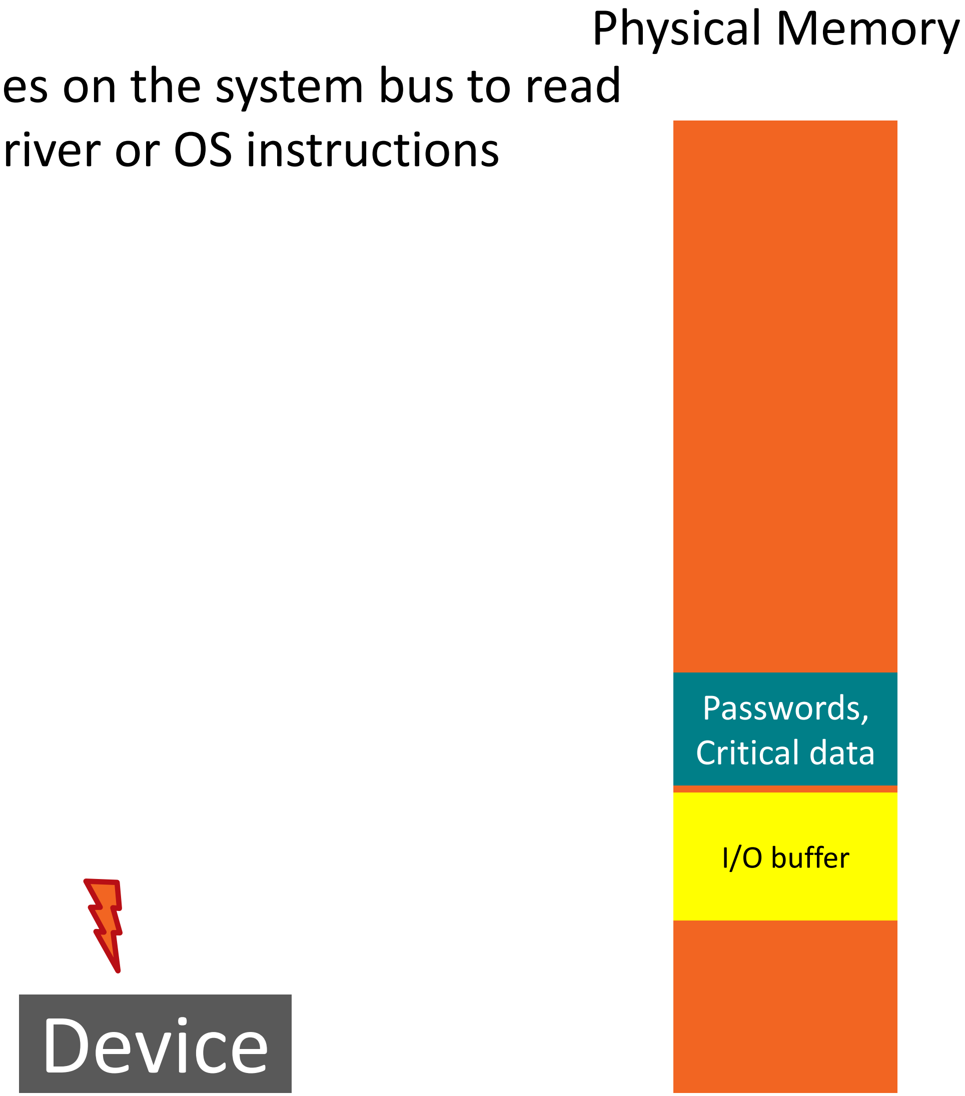
- DMA devices use physical addresses on the system bus to read and write memory based on SW driver or OS instructions



SECURITY AND PROTECTION

THE TRADITIONAL IOMMU USE

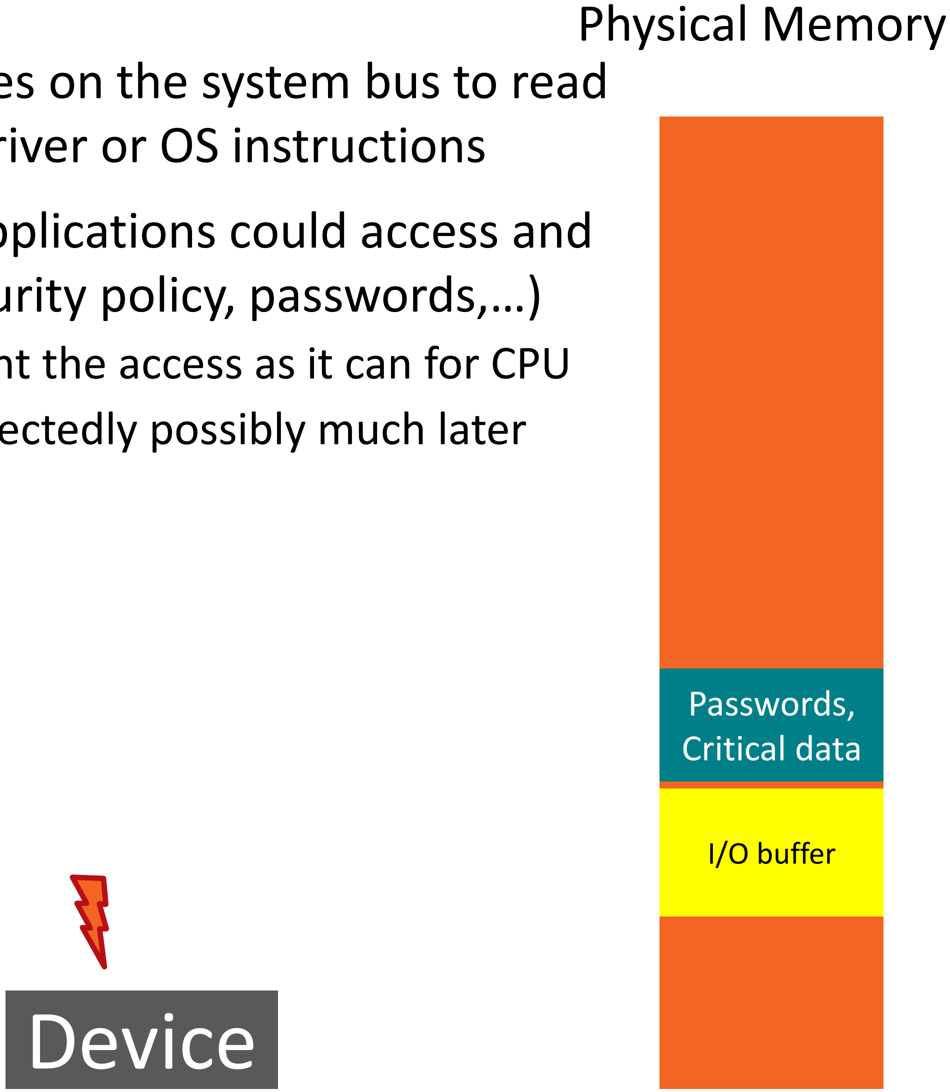
- DMA devices use physical addresses on the system bus to read and write memory based on SW driver or OS instructions



SECURITY AND PROTECTION

THE TRADITIONAL IOMMU USE

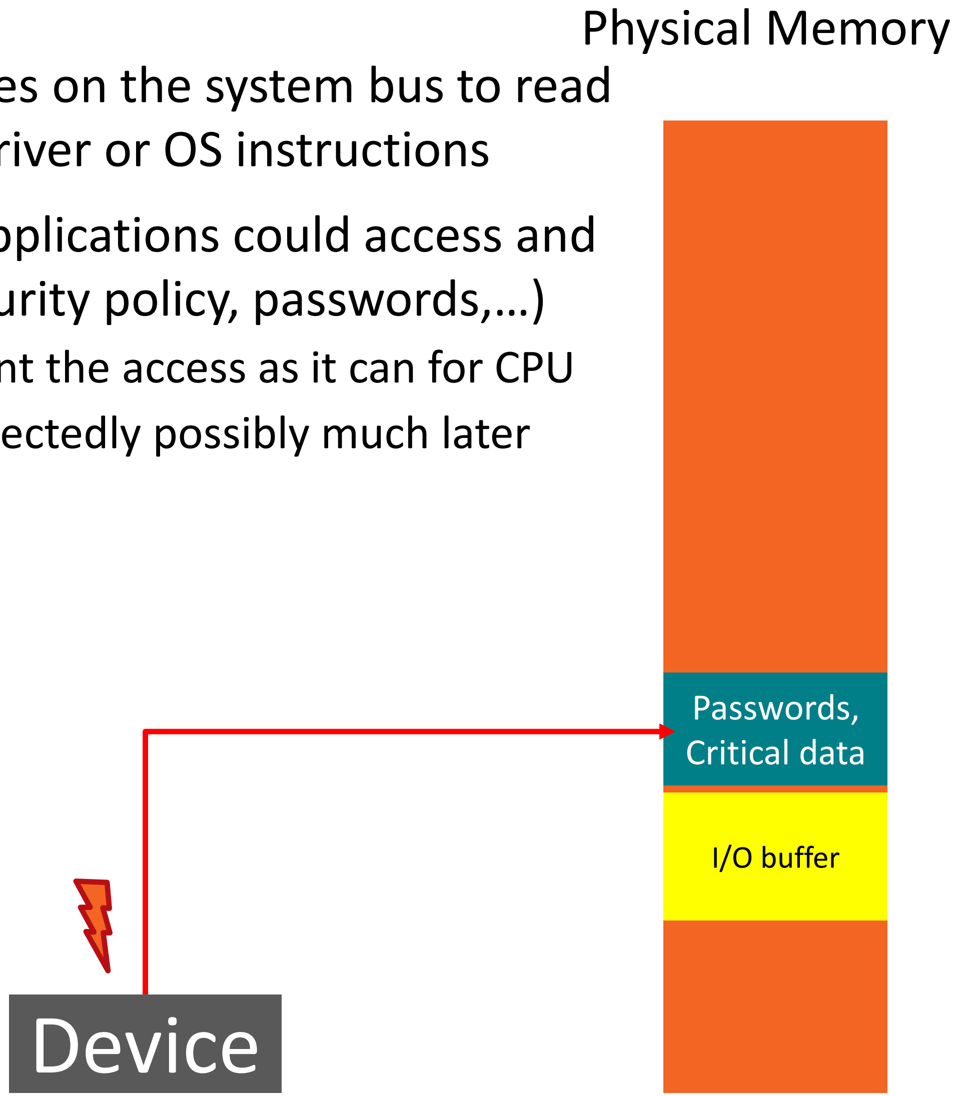
- DMA devices use physical addresses on the system bus to read and write memory based on SW driver or OS instructions
- SW bugs or attacks by malicious applications could access and modify important OS data (OS security policy, passwords,…)
  - Without OS able to detect or prevent the access as it can for CPU
  - Latent problem until it shows unexpectedly possibly much later



SECURITY AND PROTECTION

THE TRADITIONAL IOMMU USE

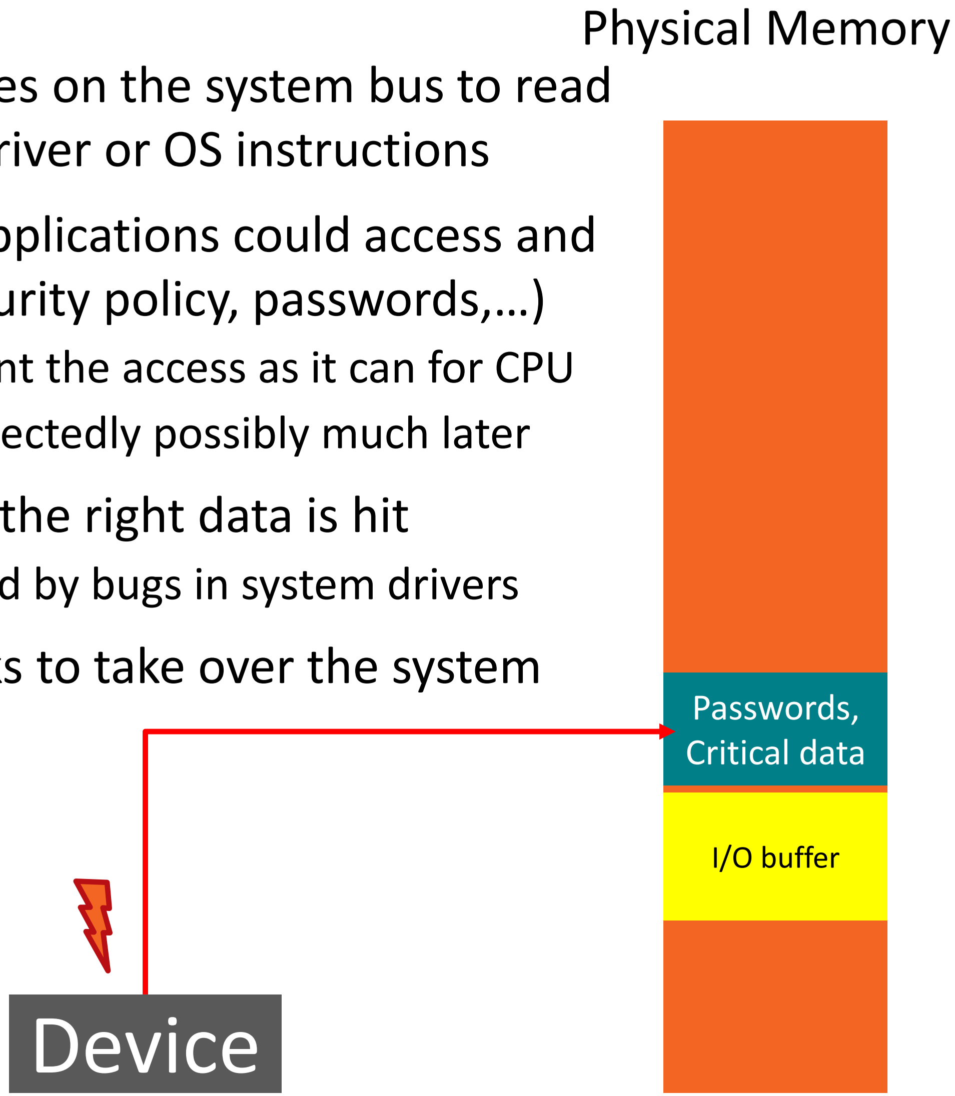
- DMA devices use physical addresses on the system bus to read and write memory based on SW driver or OS instructions
- SW bugs or attacks by malicious applications could access and modify important OS data (OS security policy, passwords,…)
  - Without OS able to detect or prevent the access as it can for CPU
  - Latent problem until it shows unexpectedly possibly much later



SECURITY AND PROTECTION

THE TRADITIONAL IOMMU USE

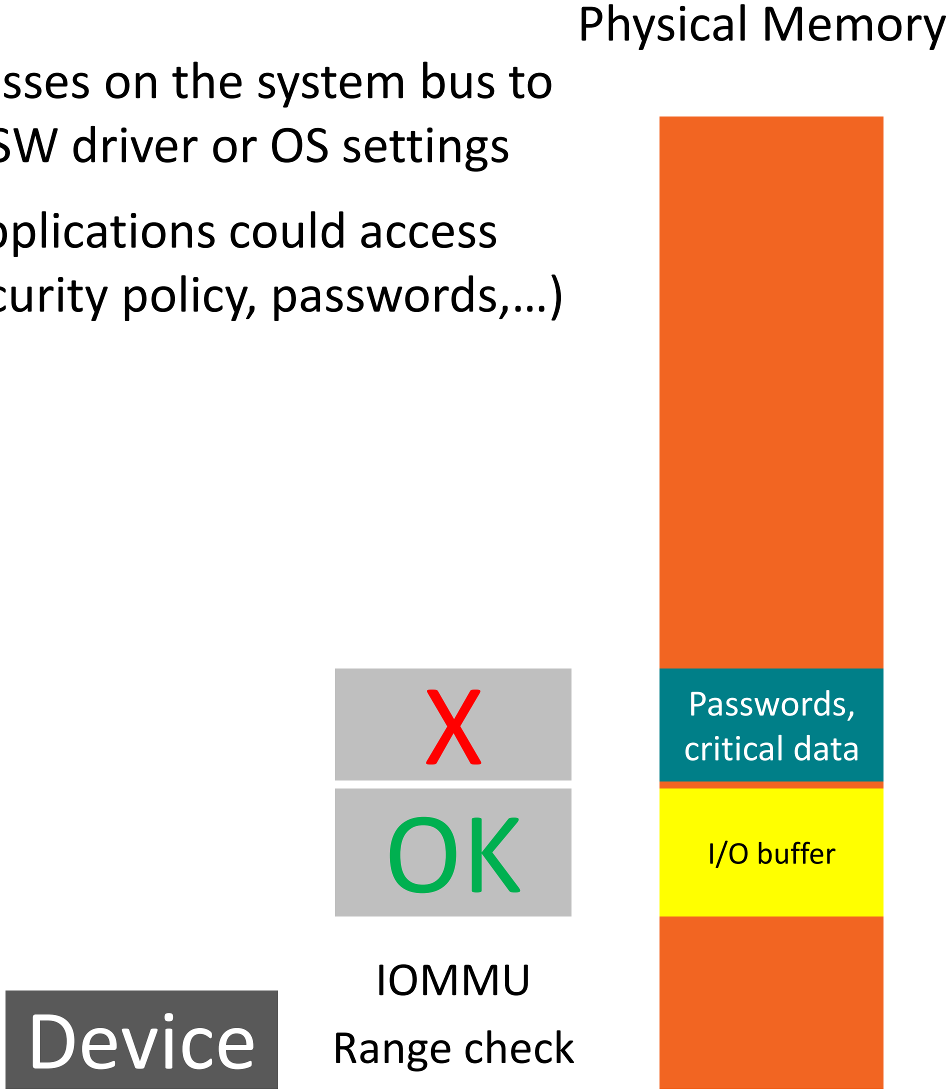
- DMA devices use physical addresses on the system bus to read and write memory based on SW driver or OS instructions
- SW bugs or attacks by malicious applications could access and modify important OS data (OS security policy, passwords,…)
  - Without OS able to detect or prevent the access as it can for CPU
  - Latent problem until it shows unexpectedly possibly much later
- This affects system stability, if just the right data is hit
  - "Heisenbugs" are sometimes caused by bugs in system drivers
- Or it allows malicious driver attacks to take over the system



SECURITY AND PROTECTION

THE TRADITIONAL IOMMU USE

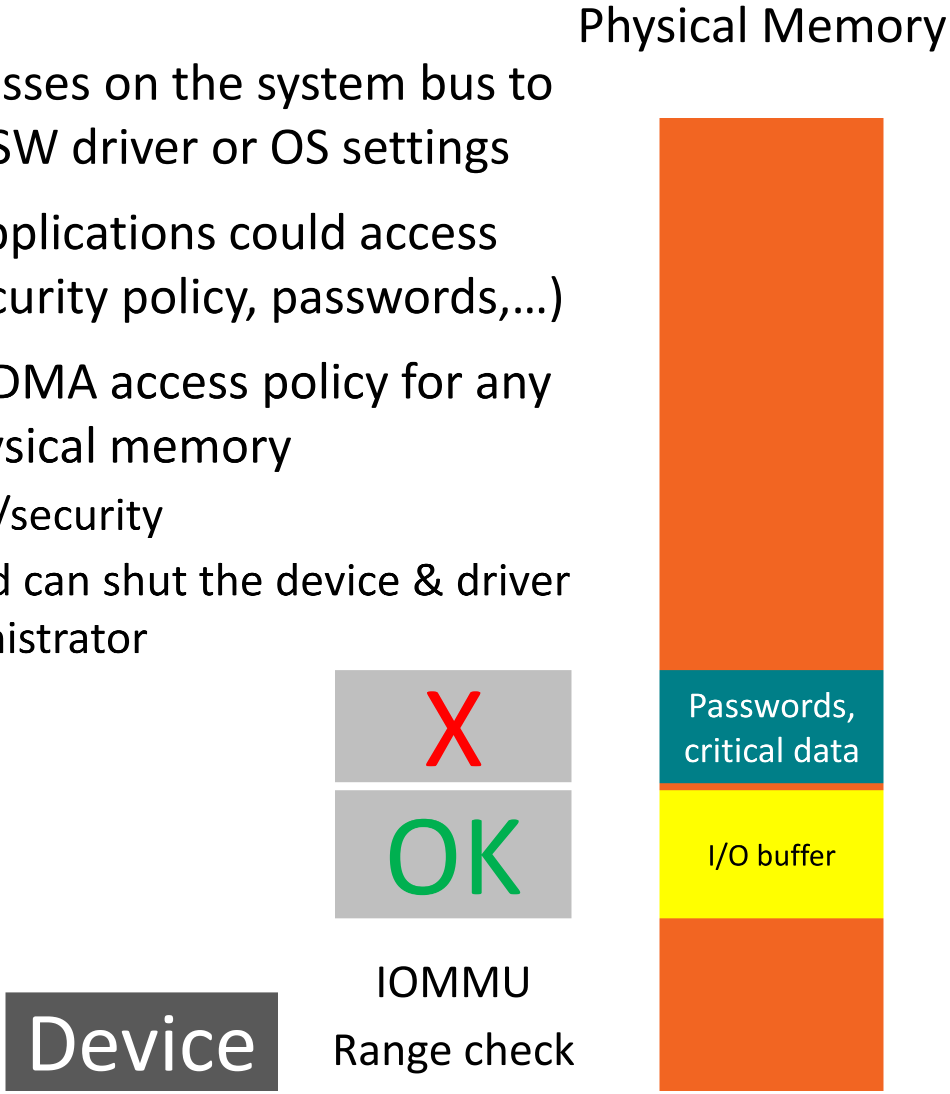
- DMA devices assert physical addresses on the system bus to read and write memory based on SW driver or OS settings
- SW bugs or attacks by malicious applications could access and modify important data (OS security policy, passwords,…)



SECURITY AND PROTECTION

THE TRADITIONAL IOMMU USE

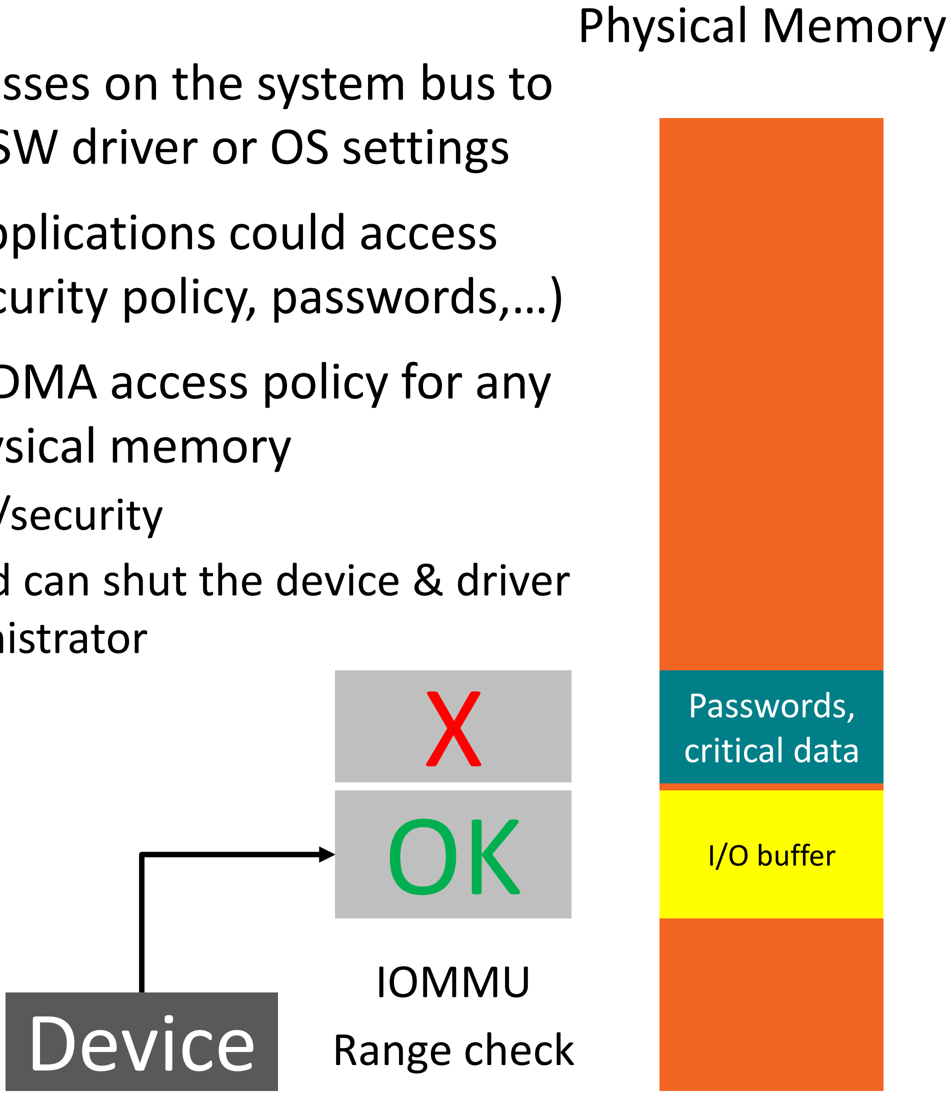
- DMA devices assert physical addresses on the system bus to read and write memory based on SW driver or OS settings
- SW bugs or attacks by malicious applications could access and modify important data (OS security policy, passwords,…)
- The IOMMU allows OS to enforce DMA access policy for any DMA capable device accessing physical memory
  - Memory state important to stability/security
  - If access occurs, OS gets notified and can shut the device & driver down and notifies the user or administrator



SECURITY AND PROTECTION

THE TRADITIONAL IOMMU USE

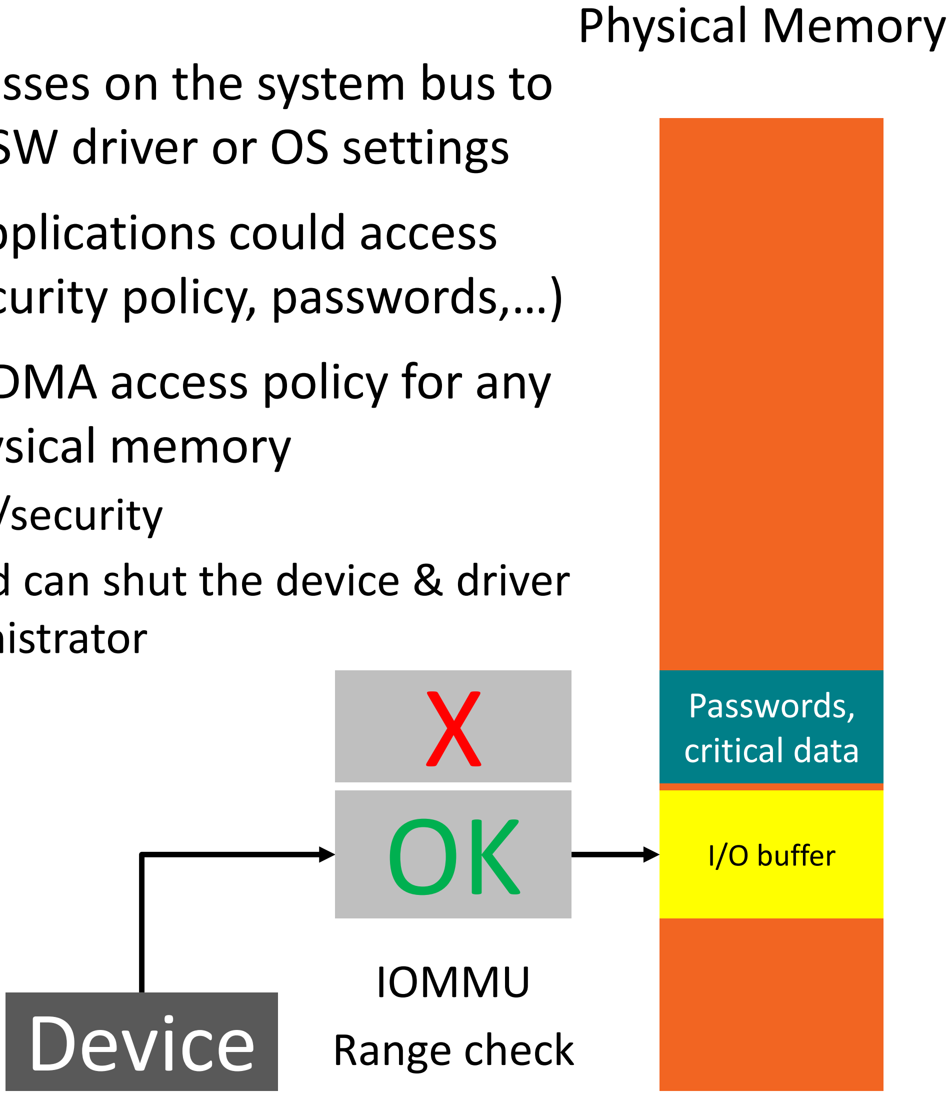
- DMA devices assert physical addresses on the system bus to read and write memory based on SW driver or OS settings
- SW bugs or attacks by malicious applications could access and modify important data (OS security policy, passwords,…)
- The IOMMU allows OS to enforce DMA access policy for any DMA capable device accessing physical memory
  - Memory state important to stability/security
  - If access occurs, OS gets notified and can shut the device & driver down and notifies the user or administrator



SECURITY AND PROTECTION

THE TRADITIONAL IOMMU USE

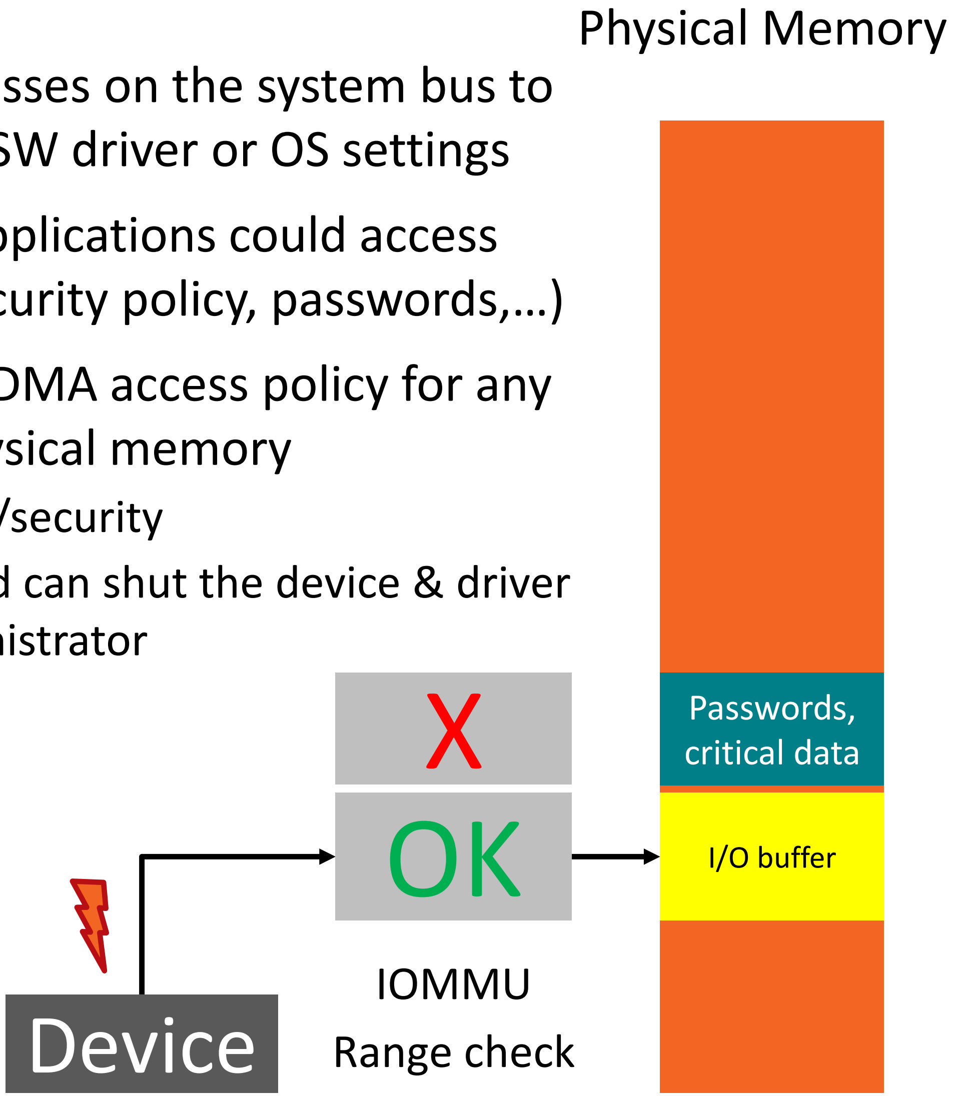
- DMA devices assert physical addresses on the system bus to read and write memory based on SW driver or OS settings
- SW bugs or attacks by malicious applications could access and modify important data (OS security policy, passwords,…)
- The IOMMU allows OS to enforce DMA access policy for any DMA capable device accessing physical memory
  - Memory state important to stability/security
  - If access occurs, OS gets notified and can shut the device & driver down and notifies the user or administrator



SECURITY AND PROTECTION

THE TRADITIONAL IOMMU USE

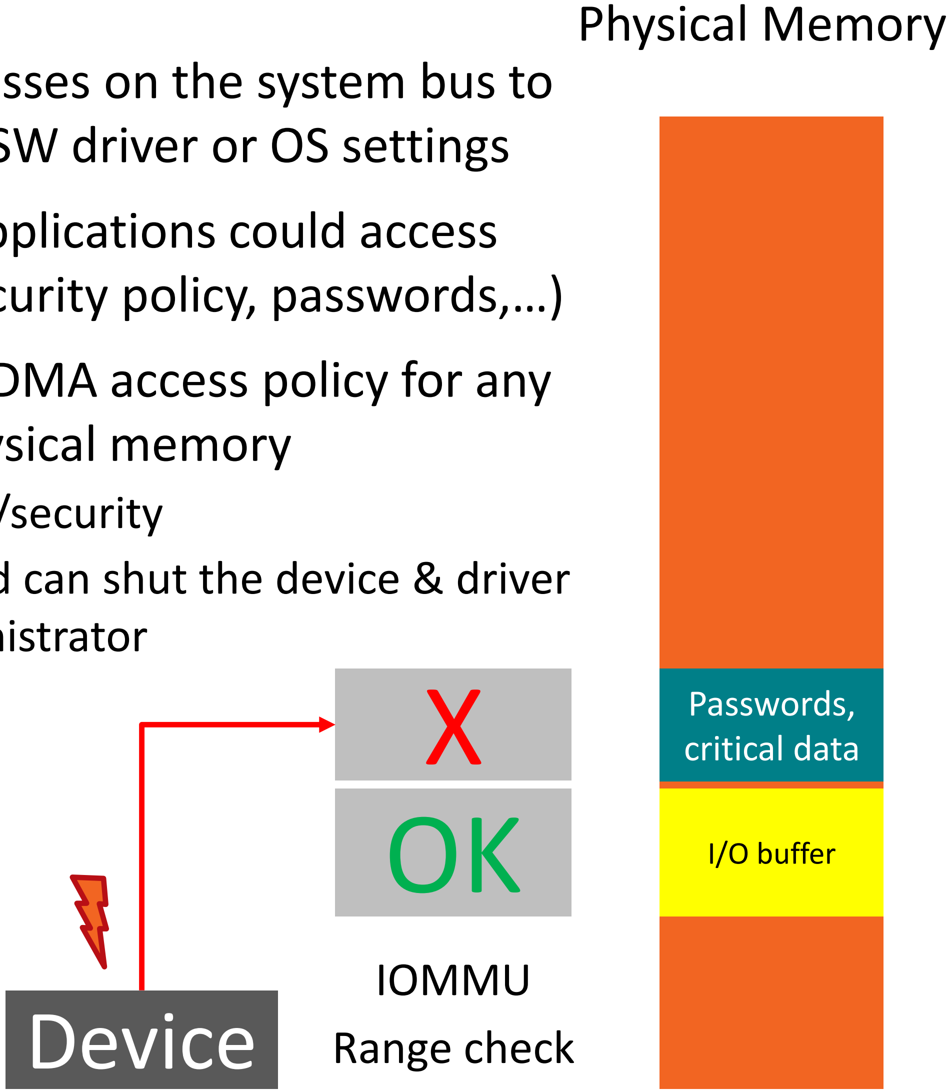
- DMA devices assert physical addresses on the system bus to read and write memory based on SW driver or OS settings
- SW bugs or attacks by malicious applications could access and modify important data (OS security policy, passwords,…)
- The IOMMU allows OS to enforce DMA access policy for any DMA capable device accessing physical memory
  - Memory state important to stability/security
  - If access occurs, OS gets notified and can shut the device & driver down and notifies the user or administrator



SECURITY AND PROTECTION

THE TRADITIONAL IOMMU USE

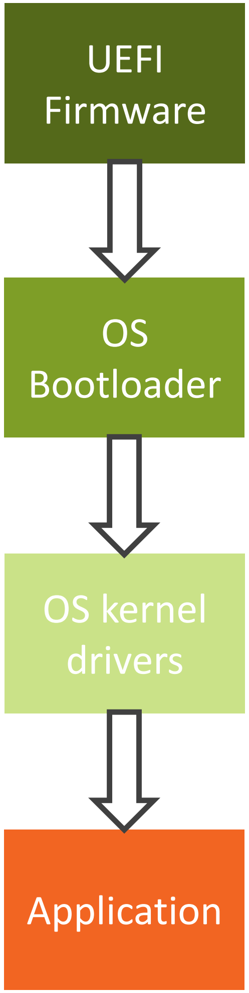
- DMA devices assert physical addresses on the system bus to read and write memory based on SW driver or OS settings
- SW bugs or attacks by malicious applications could access and modify important data (OS security policy, passwords,…)
- The IOMMU allows OS to enforce DMA access policy for any DMA capable device accessing physical memory
  - Memory state important to stability/security
  - If access occurs, OS gets notified and can shut the device & driver down and notifies the user or administrator



SECURE BOOT

YET ANOTHER USE FOR AN IOMMU

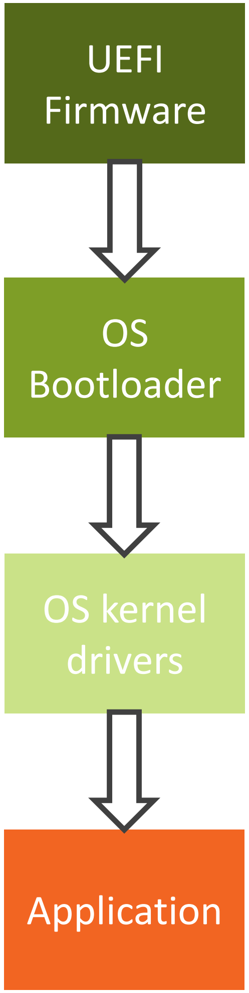
- Ensuring that a system is not doing more than it’s supposed to
  - e.g., being part of a botnet, provide banking data or other personal info to impersonators or other attackers
  - The earliest time for attack and defense is at firmware startup
  - From there critical memory regions are protected from invalid access



SECURE BOOT

YET ANOTHER USE FOR AN IOMMU

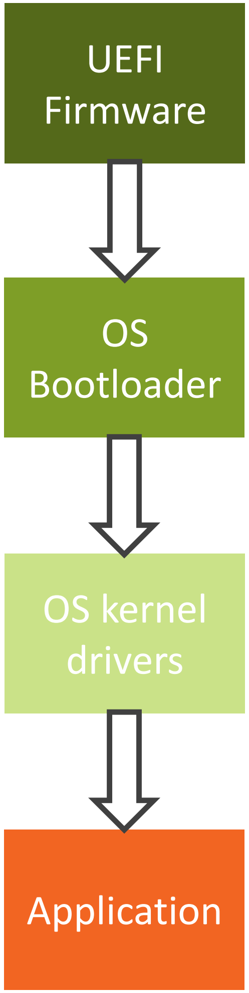
- Ensuring that a system is not doing more than it's supposed to
  - e.g., being part of a botnet, provide banking data or other personal info to impersonators or other attackers
  - The earliest time for attack and defense is at firmware startup
  - From there critical memory regions are protected from invalid access
- The Secure Boot architecture ensures that no non-vetted OS kernel code runs on the system, changing critical settings



SECURE BOOT

YET ANOTHER USE FOR AN IOMMU

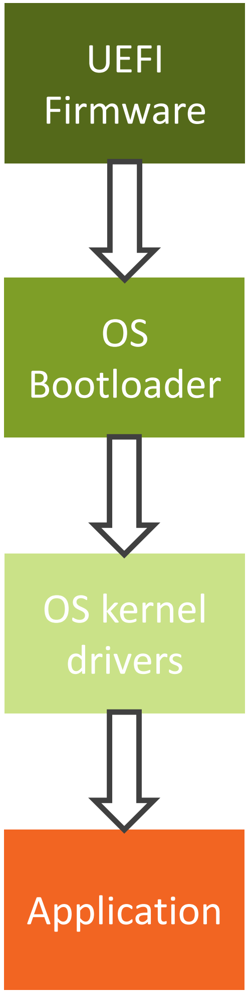
- Ensuring that a system is not doing more than it’s supposed to
  - e.g., being part of a botnet, provide banking data or other personal info to impersonators or other attackers
  - The earliest time for attack and defense is at firmware startup
  - From there critical memory regions are protected from invalid access
- The Secure Boot architecture ensures that no non-vetted OS kernel code runs on the system, changing critical settings
- Some I/O devices can issue DMA requests to system memory directly, without OS or Firmware intervention
  - e.g.,1394/Firewire, network cards, as part of network boot
  - That allows attacks to modify memory before even the OS has a chance to protect against the attacks



SECURE BOOT

YET ANOTHER USE FOR AN IOMMU

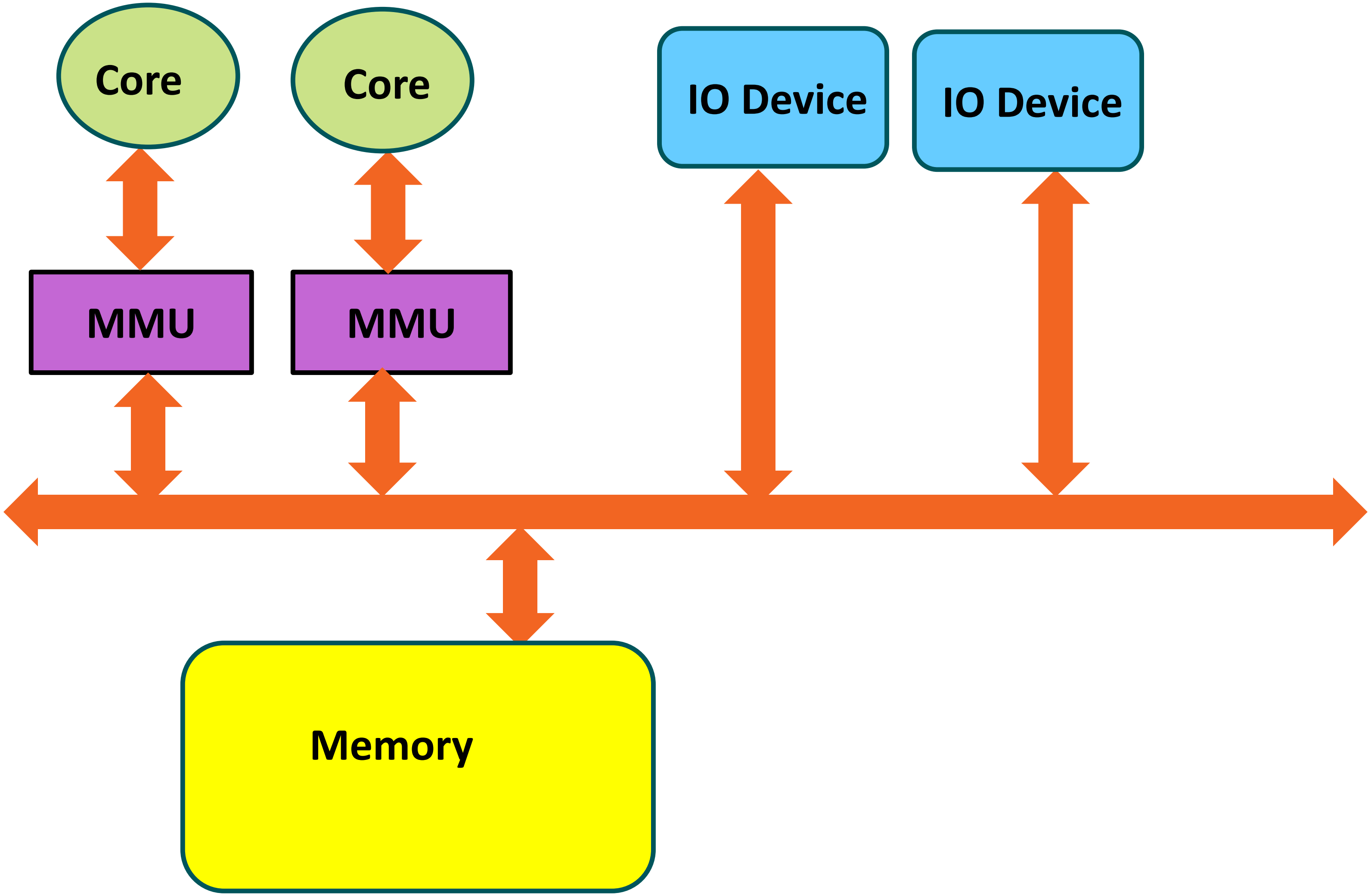
- Ensuring that a system is not doing more than it's supposed to
  - e.g., being part of a botnet, provide banking data or other personal info to impersonators or other attackers
  - The earliest time for attack and defense is at firmware startup
  - From there critical memory regions are protected from invalid access
- The Secure Boot architecture ensures that no non-vetted OS kernel code runs on the system, changing critical settings
- Some I/O devices can issue DMA requests to system memory directly, without OS or Firmware intervention
  - e.g.,1394/Firewire, network cards, as part of network boot
  - That allows attacks to modify memory before even the OS has a chance to protect against the attacks
- As outlined earlier, using the IOMMU prevents DMA access to important memory regions



IOMMU USECASE: EFFICIENT IO IN VIRTUALIZED ENVIRONMENT

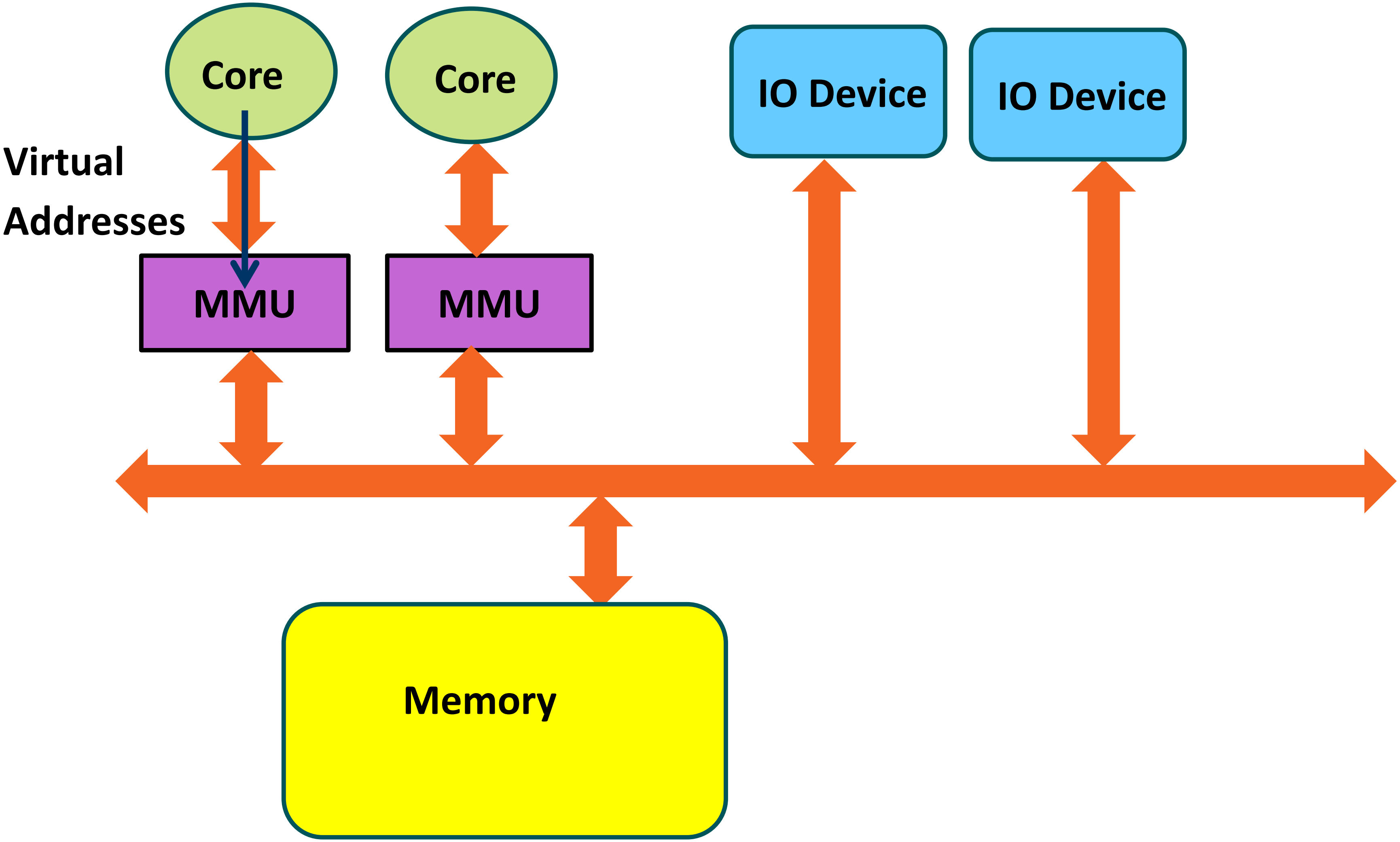
BACKGROUND: TRADITIONAL DMA BY IO

(NO SYSTEM VIRTUALIZATION)



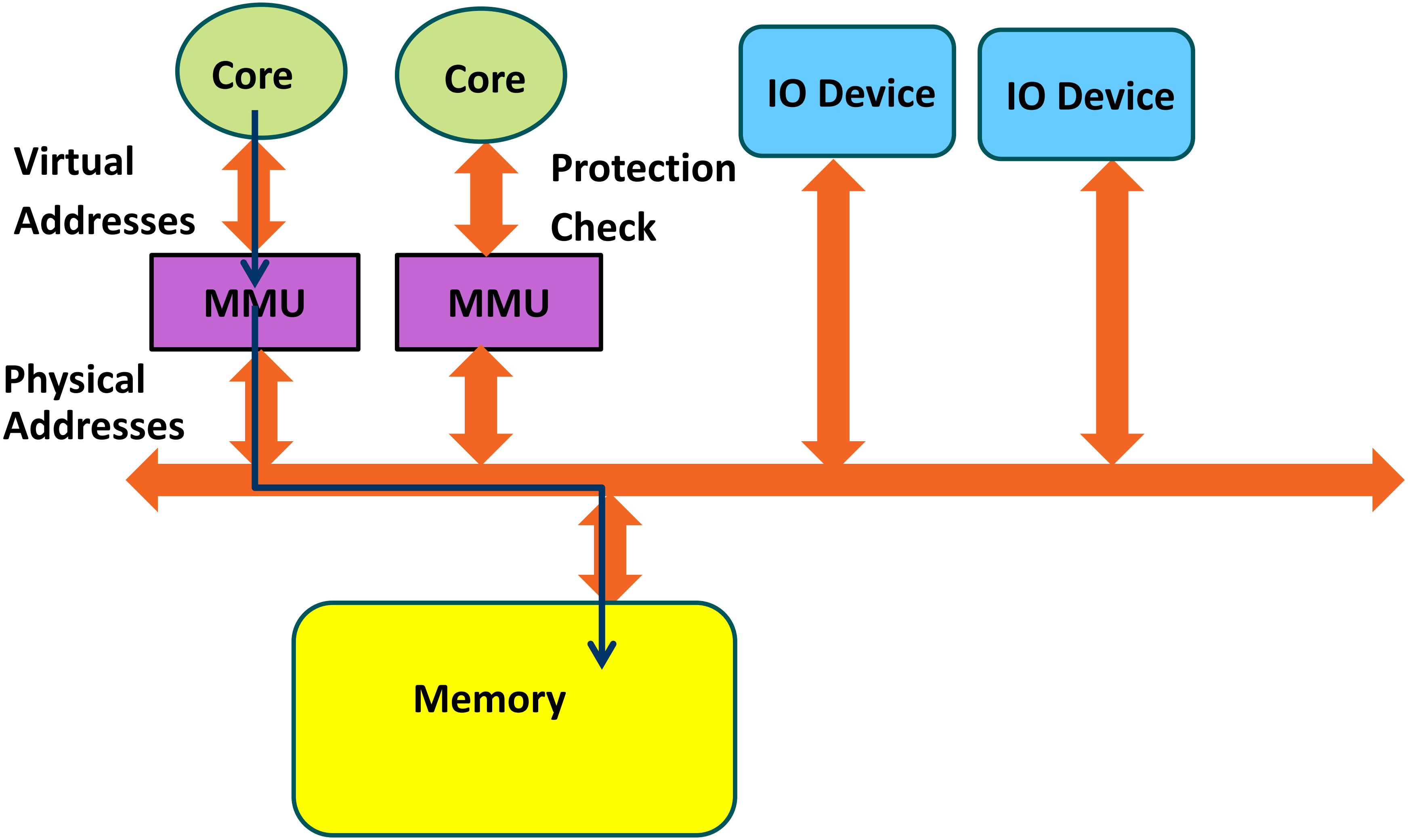
BACKGROUND: TRADITIONAL DMA BY IO

(NO SYSTEM VIRTUALIZATION)



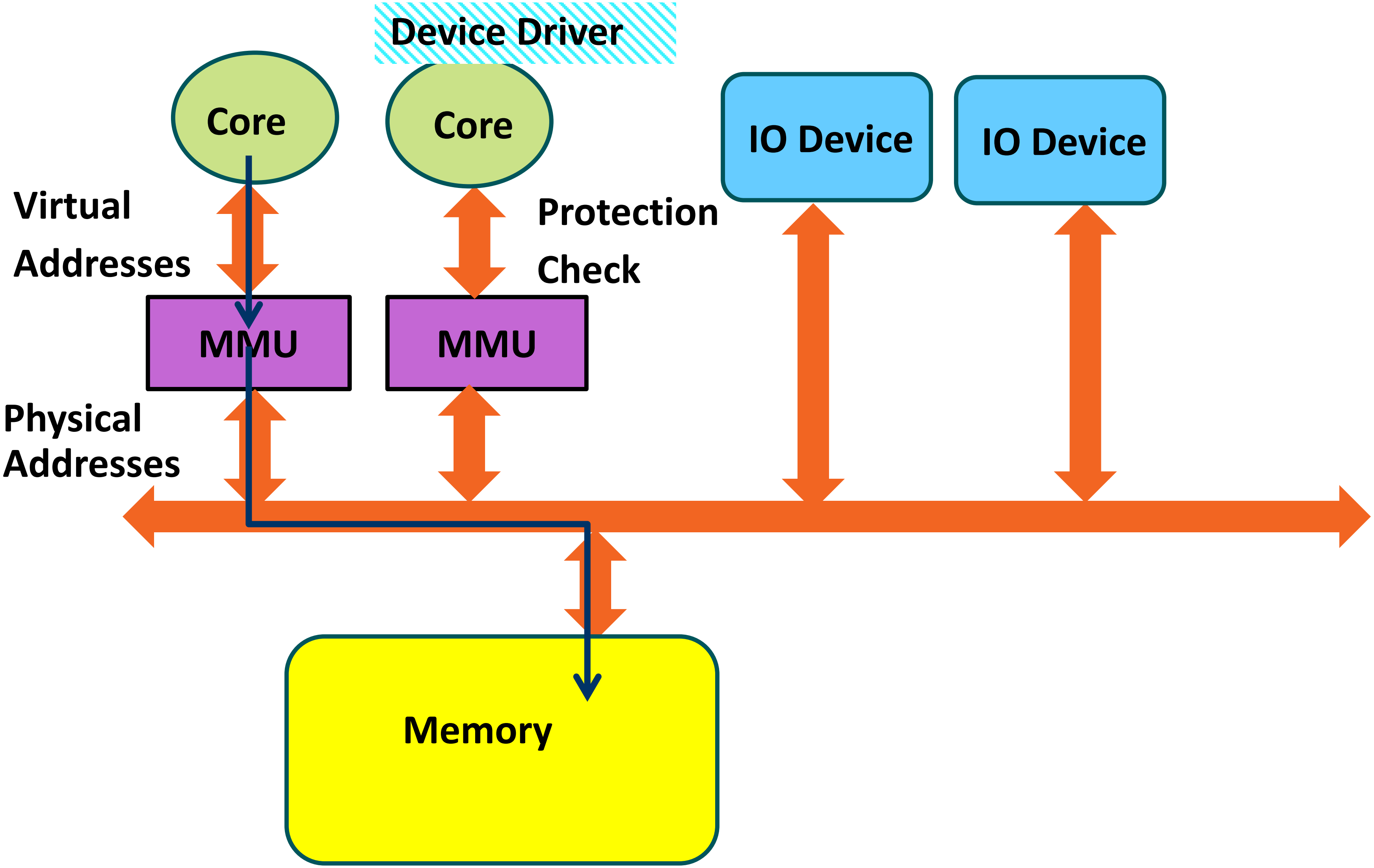
BACKGROUND: TRADITIONAL DMA BY IO

(NO SYSTEM VIRTUALIZATION)



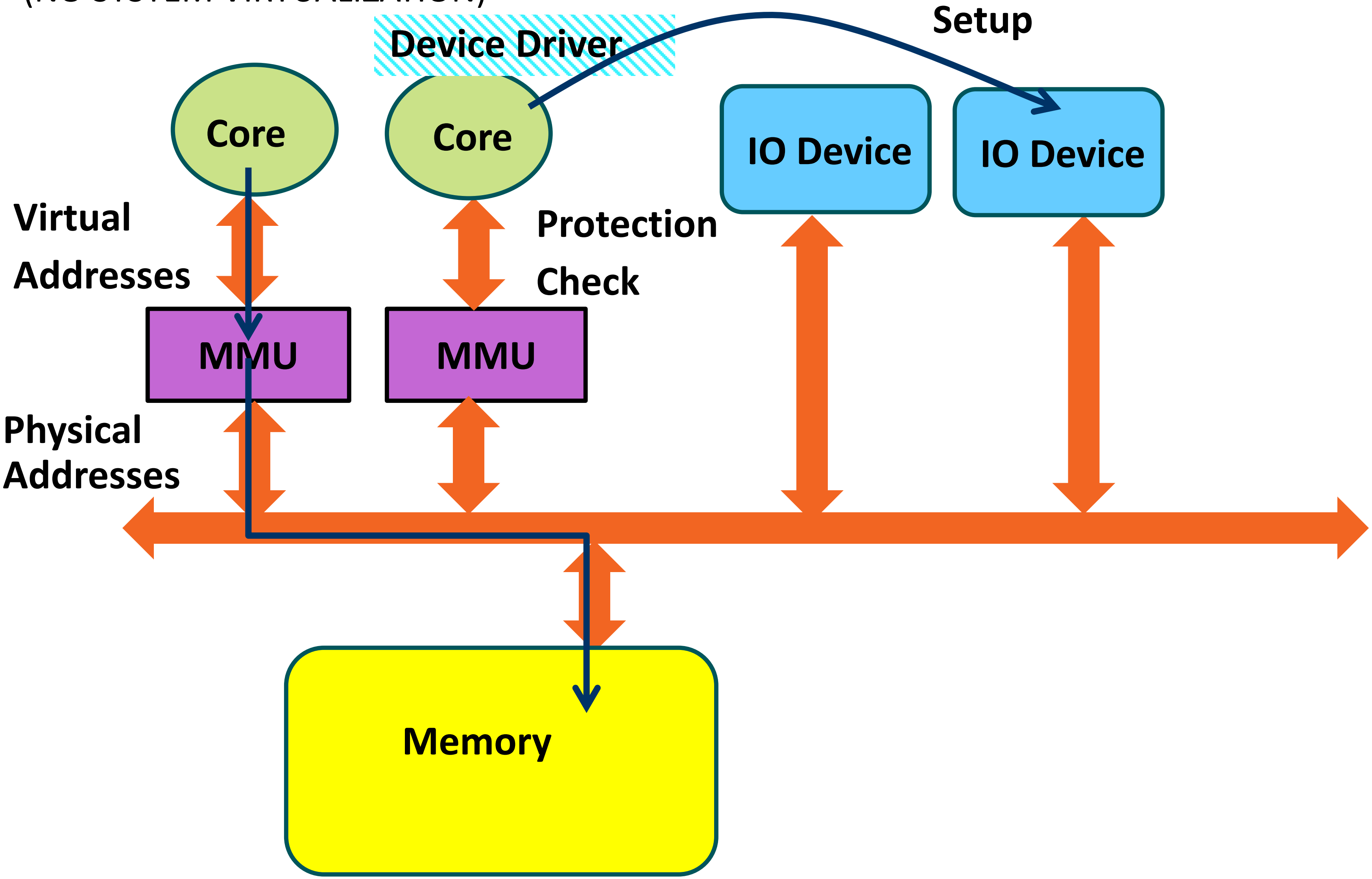
BACKGROUND: TRADITIONAL DMA BY IO

(NO SYSTEM VIRTUALIZATION)



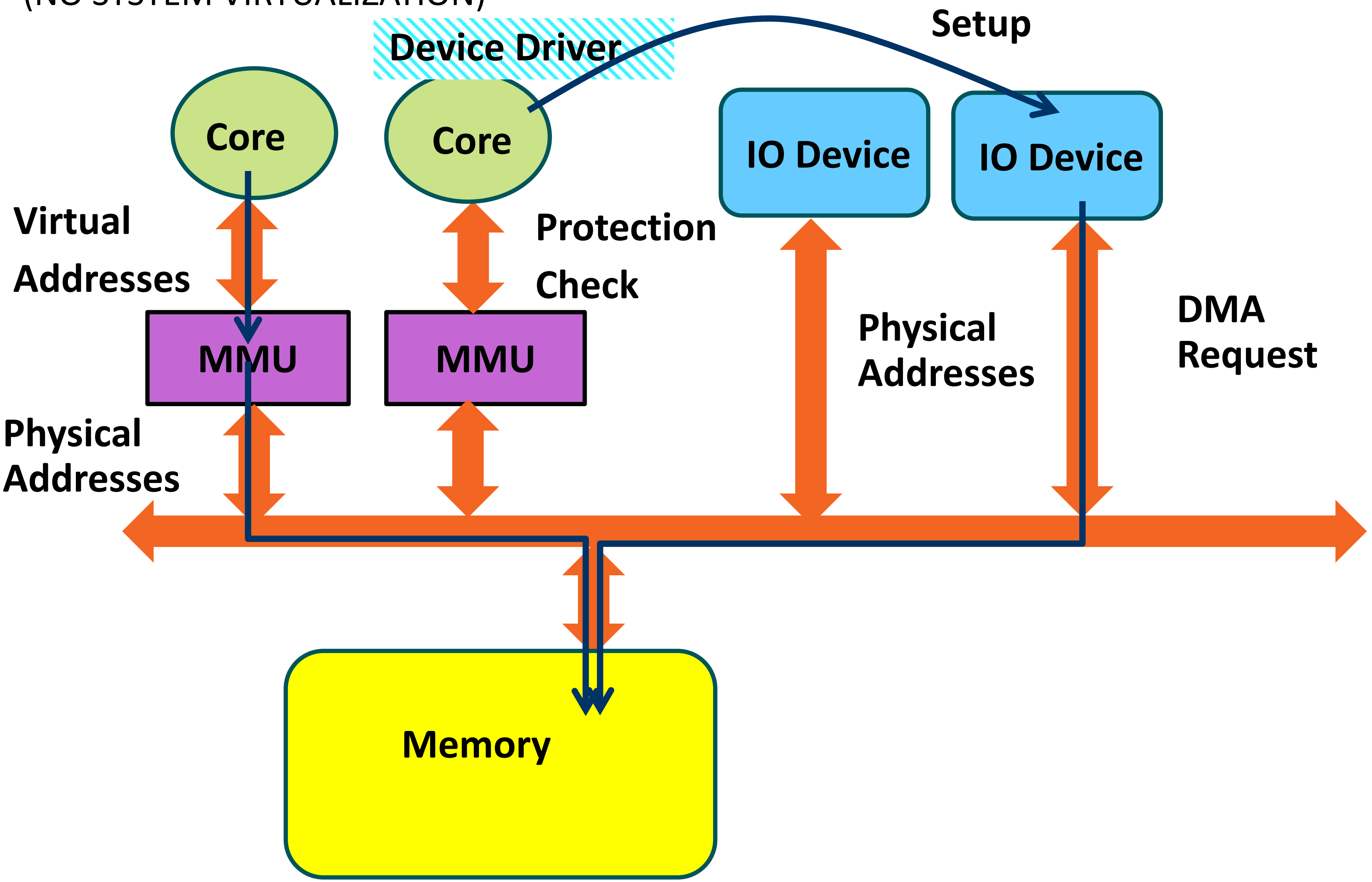
BACKGROUND: TRADITIONAL DMA BY IO

(NO SYSTEM VIRTUALIZATION)



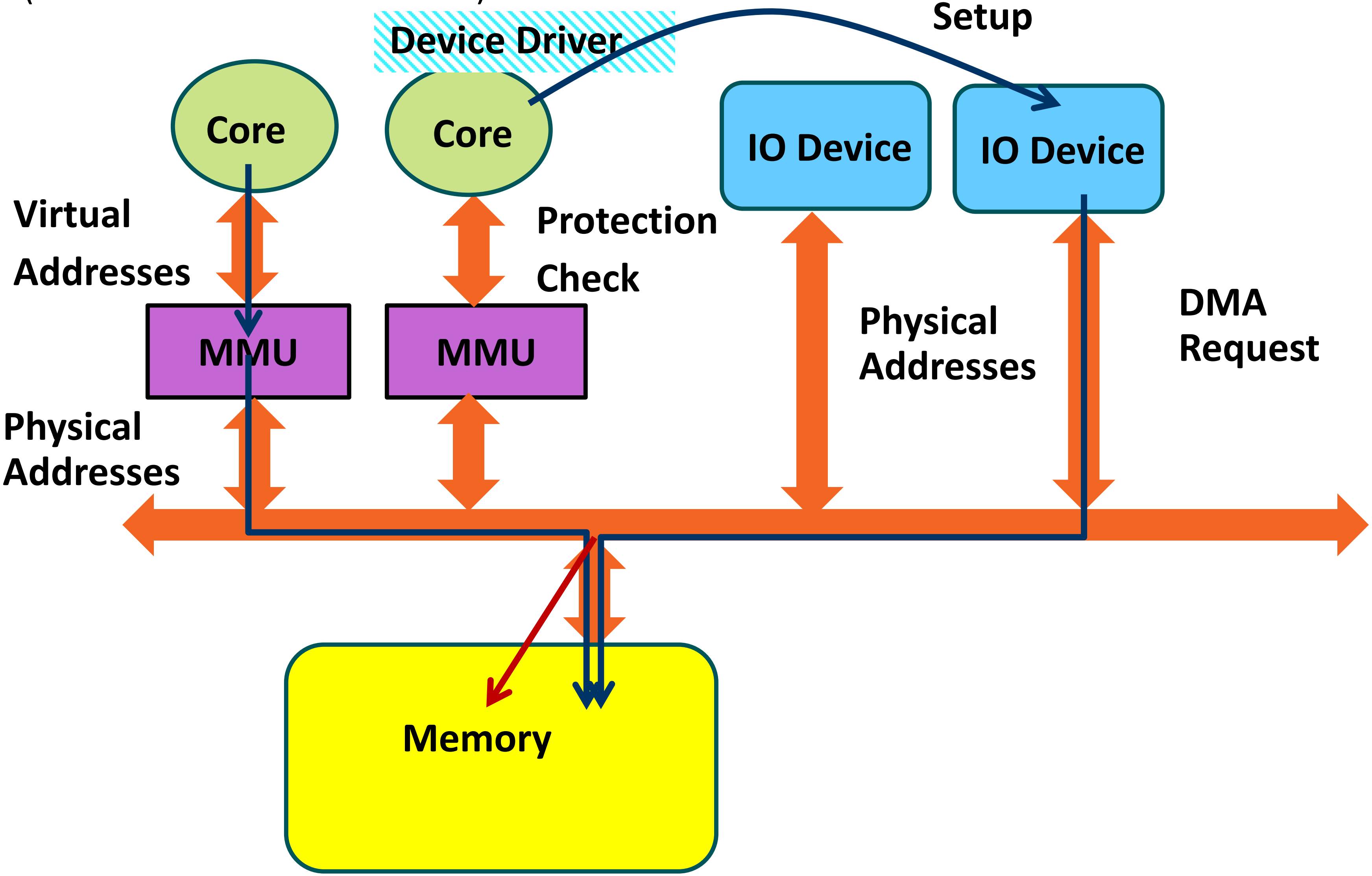
BACKGROUND: TRADITIONAL DMA BY IO

(NO SYSTEM VIRTUALIZATION)



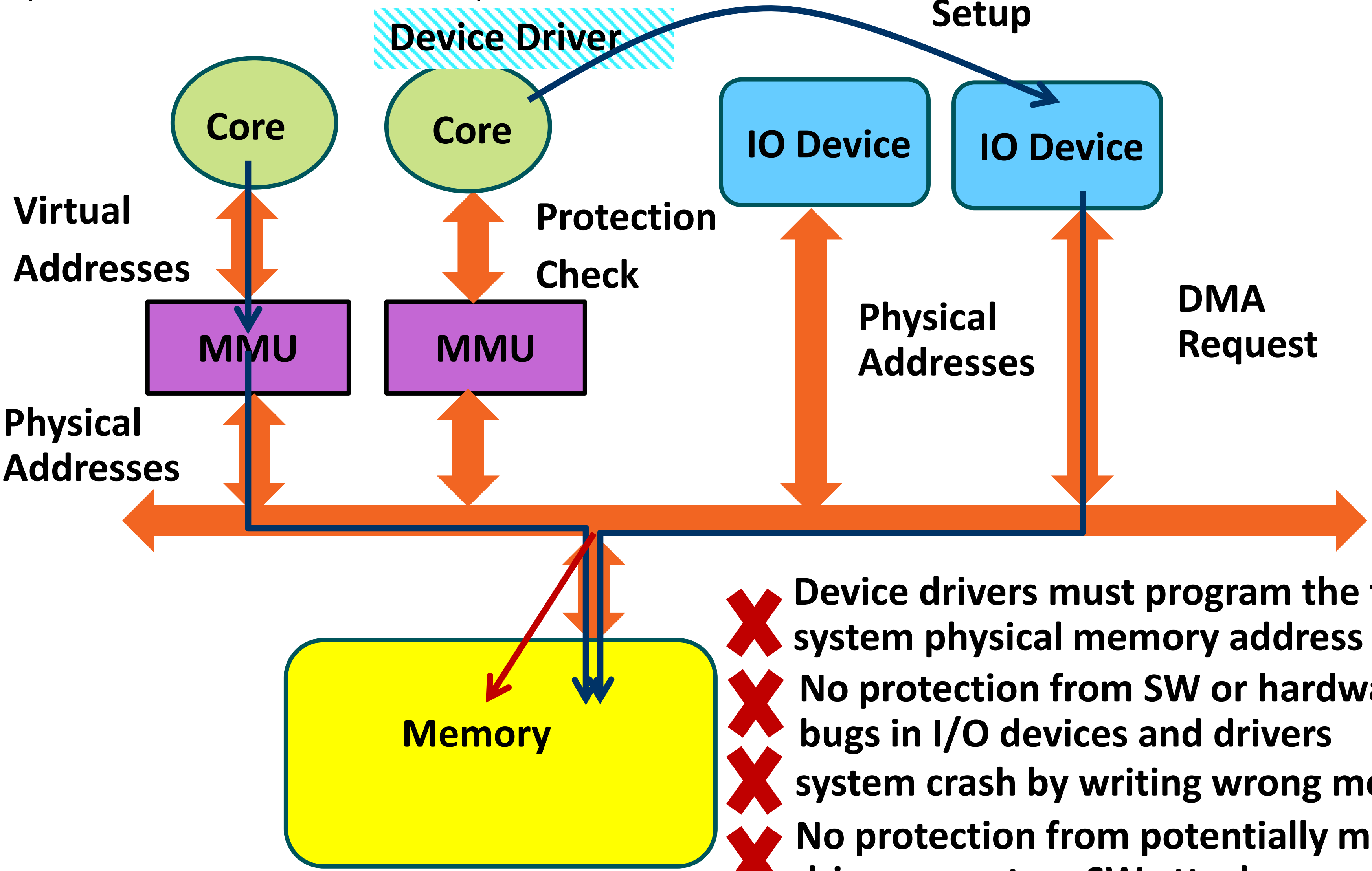
BACKGROUND: TRADITIONAL DMA BY IO

(NO SYSTEM VIRTUALIZATION)



BACKGROUND: TRADITIONAL DMA BY IO

(NO SYSTEM VIRTUALIZATION)



- ✗ Device drivers must program the true system physical memory address
- ✗ No protection from SW or hardware bugs in I/O devices and drivers
- ✗ system crash by writing wrong memory
- ✗ No protection from potentially malicious driver or system SW attacks

VIRTUALIZATION OF A SYSTEM IN SOFTWARE

IT HAS TO LOOK REAL TO AN OPERATING SYSTEM

- Each OS assumes full access to the platform hardware
  - Memory, Interrupts, Devices, CPU cores, etc.

VIRTUALIZATION OF A SYSTEM IN SOFTWARE

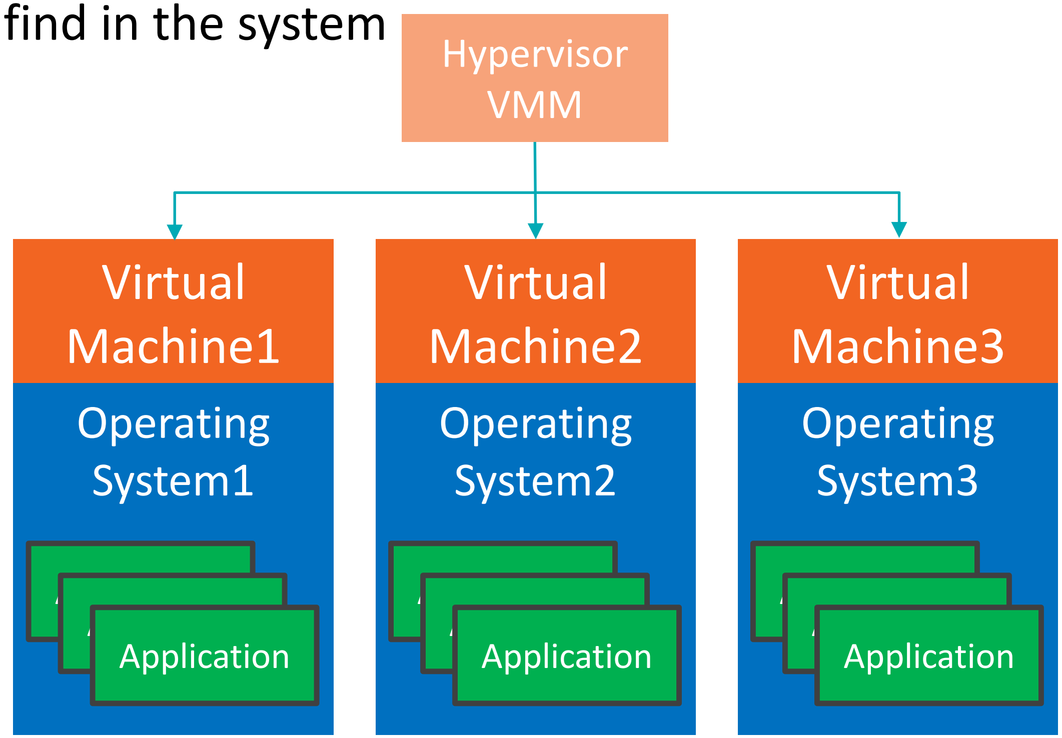
IT HAS TO LOOK REAL TO AN OPERATING SYSTEM

- Each OS assumes full access to the platform hardware
  - Memory, Interrupts, Devices, CPU cores, etc.
- A Virtual Machine Manager (VMM) or Hypervisor (HV) is tasked to manage the physical hardware and define a “virtual machine” (VM) that represents the resources an OS expects to find in the system

VIRTUALIZATION OF A SYSTEM IN SOFTWARE

IT HAS TO LOOK REAL TO AN OPERATING SYSTEM

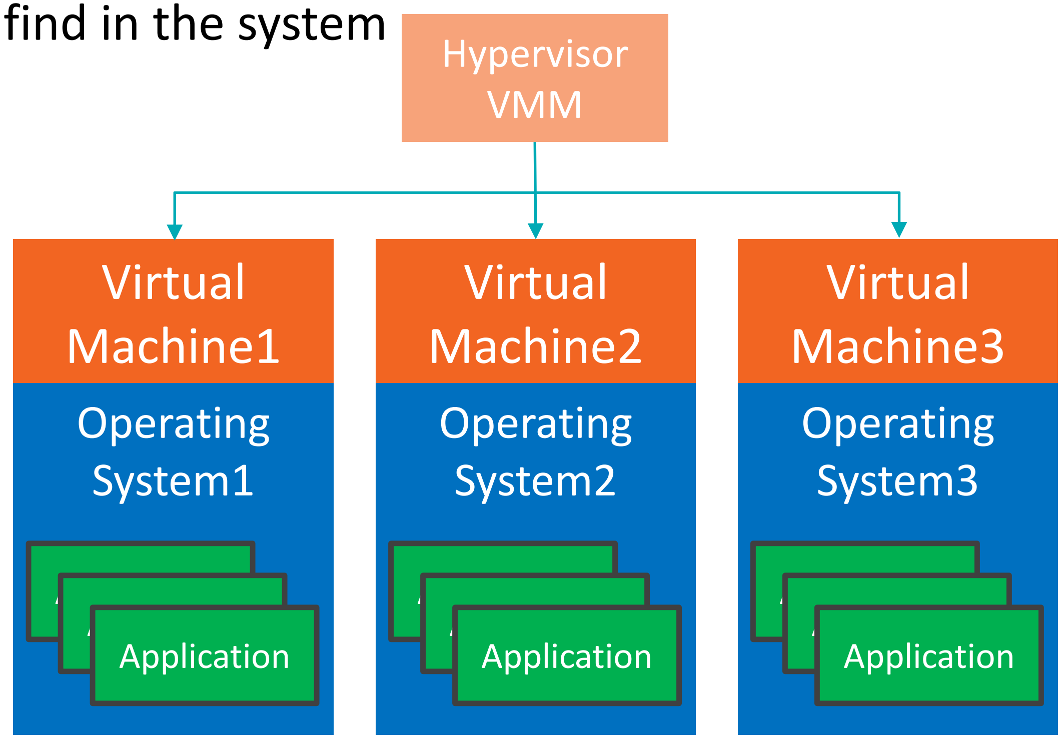
- Each OS assumes full access to the platform hardware
  - Memory, Interrupts, Devices, CPU cores, etc.
- A Virtual Machine Manager (VMM) or Hypervisor (HV) is tasked to manage the physical hardware and define a “virtual machine” (VM) that represents the resources an OS expects to find in the system



VIRTUALIZATION OF A SYSTEM IN SOFTWARE

IT HAS TO LOOK REAL TO AN OPERATING SYSTEM

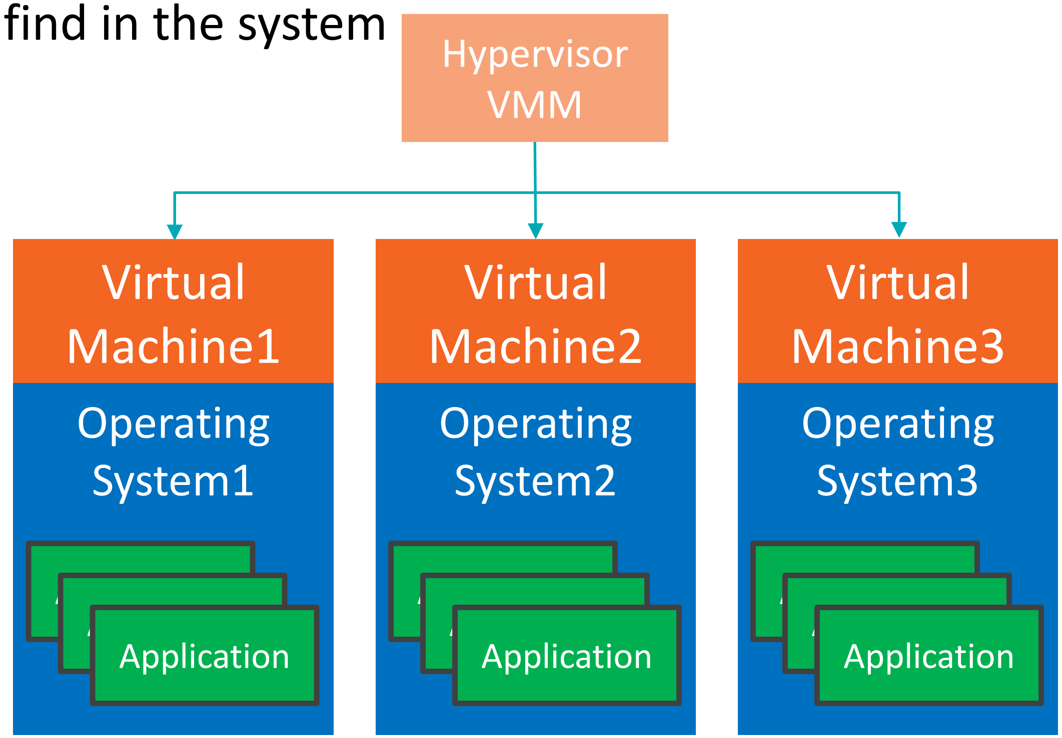
- Each OS assumes full access to the platform hardware
  - Memory, Interrupts, Devices, CPU cores, etc.
- A Virtual Machine Manager (VMM) or Hypervisor (HV) is tasked to manage the physical hardware and define a “virtual machine” (VM) that represents the resources an OS expects to find in the system
- Use cases:



VIRTUALIZATION OF A SYSTEM IN SOFTWARE

IT HAS TO LOOK REAL TO AN OPERATING SYSTEM

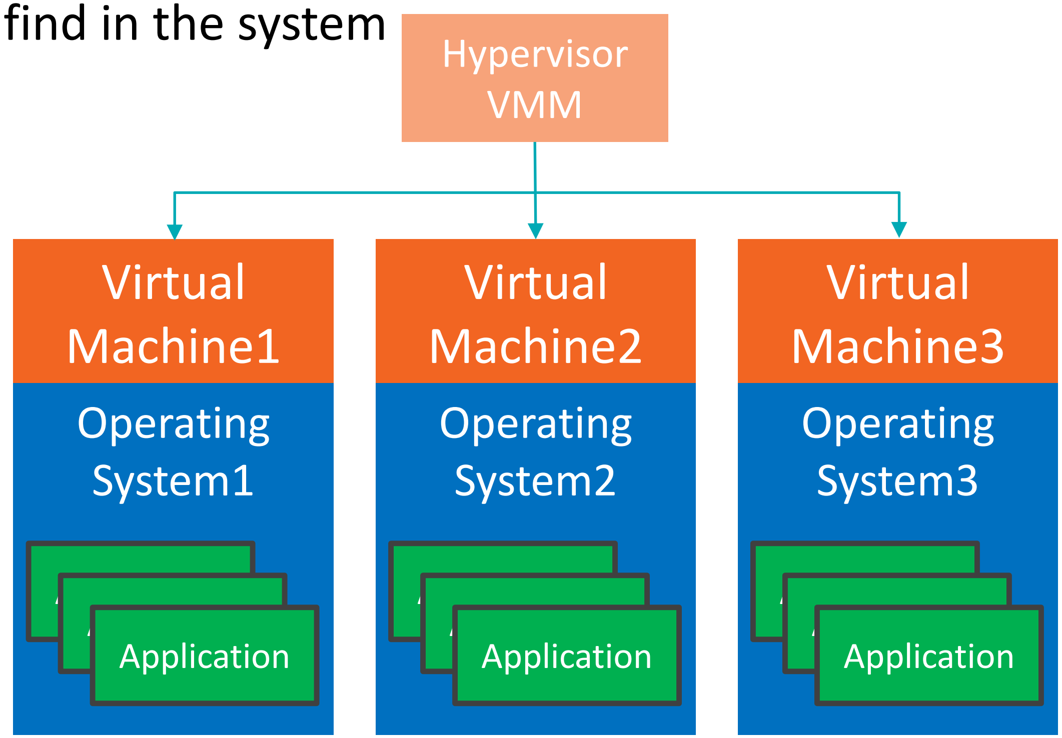
- Each OS assumes full access to the platform hardware
  - Memory, Interrupts, Devices, CPU cores, etc.
- A Virtual Machine Manager (VMM) or Hypervisor (HV) is tasked to manage the physical hardware and define a "virtual machine" (VM) that represents the resources an OS expects to find in the system
- Use cases:
  - System consolidation



VIRTUALIZATION OF A SYSTEM IN SOFTWARE

IT HAS TO LOOK REAL TO AN OPERATING SYSTEM

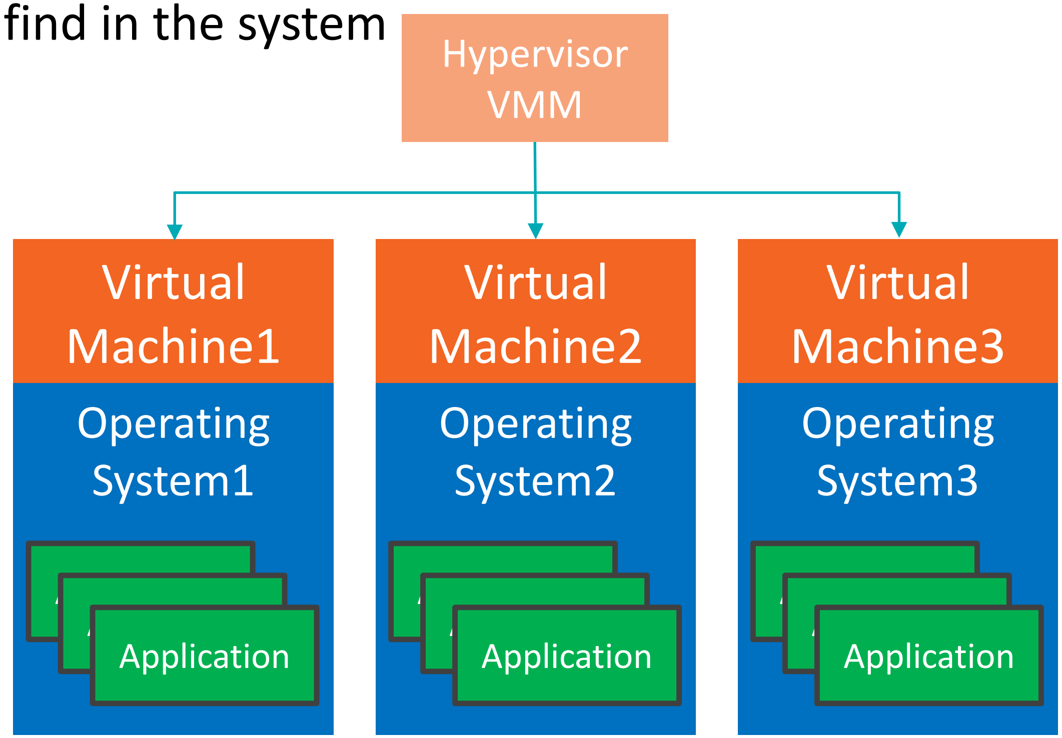
- Each OS assumes full access to the platform hardware
  - Memory, Interrupts, Devices, CPU cores, etc.
- A Virtual Machine Manager (VMM) or Hypervisor (HV) is tasked to manage the physical hardware and define a “virtual machine” (VM) that represents the resources an OS expects to find in the system
- Use cases:
  - System consolidation
  - OS/application compatibility



VIRTUALIZATION OF A SYSTEM IN SOFTWARE

IT HAS TO LOOK REAL TO AN OPERATING SYSTEM

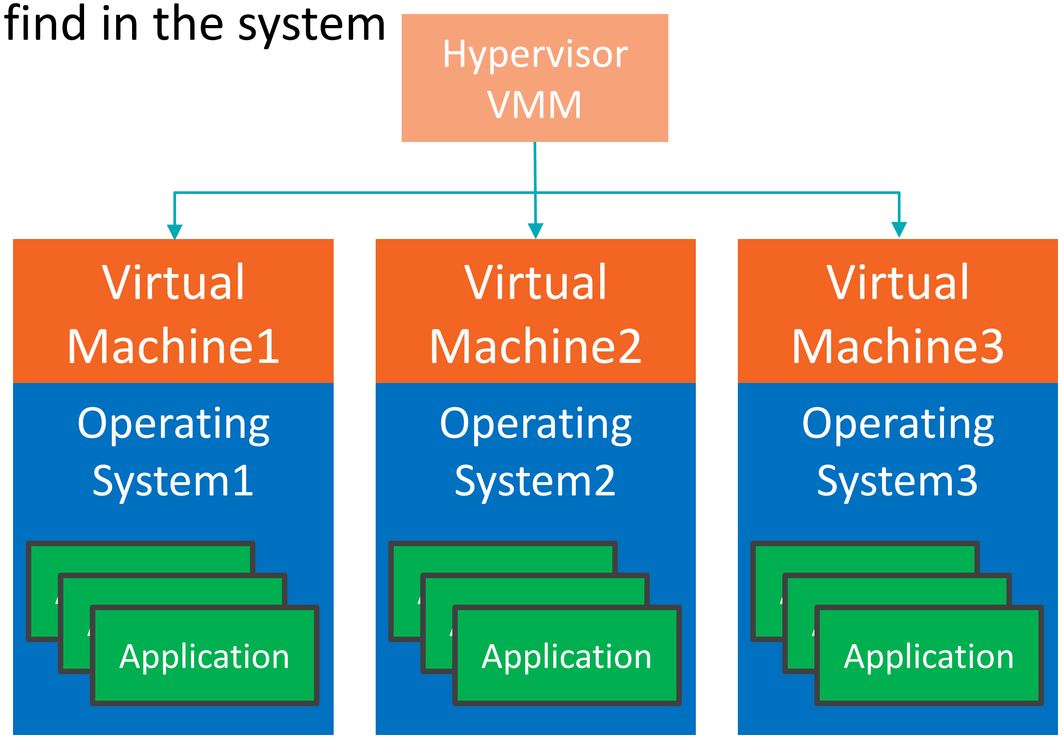
- Each OS assumes full access to the platform hardware
  - Memory, Interrupts, Devices, CPU cores, etc.
- A Virtual Machine Manager (VMM) or Hypervisor (HV) is tasked to manage the physical hardware and define a “virtual machine” (VM) that represents the resources an OS expects to find in the system
- Use cases:
  - System consolidation
  - OS/application compatibility
  - Security / Stability



VIRTUALIZATION OF A SYSTEM IN SOFTWARE

IT HAS TO LOOK REAL TO AN OPERATING SYSTEM

- Each OS assumes full access to the platform hardware
  - Memory, Interrupts, Devices, CPU cores, etc.
- A Virtual Machine Manager (VMM) or Hypervisor (HV) is tasked to manage the physical hardware and define a "virtual machine" (VM) that represents the resources an OS expects to find in the system
- Use cases:
  - System consolidation
  - OS/application compatibility
  - Security / Stability
  - Cloud Infrastructure



VIRTUALIZATION OF A SYSTEM

- Most CPUs today have support for system virtualization
  - Nested page tables (HV & OS levels), allow VMM/HV to assign and manage system memory and interrupts to Virtual Machines

VIRTUALIZATION OF A SYSTEM

- Most CPUs today have support for system virtualization
  - Nested page tables (HV & OS levels), allow VMM/HV to assign and manage system memory and interrupts to Virtual Machines
- I/O devices are typically managed by HV/VMM software, either by…

VIRTUALIZATION OF A SYSTEM

- Most CPUs today have support for system virtualization
  - Nested page tables (HV & OS levels), allow VMM/HV to assign and manage system memory and interrupts to Virtual Machines
- I/O devices are typically managed by HV/VMM software, either by…

Para-Virtualization
Guest device driver uses HV “hypercalls” Hypervisor manages HW operation (DMA)
Hypervisor SW validates and redirects I/O requests from Guest OS (overhead, slow)
Hypervisor arbitrates and schedules requests from multiple guest OS, allows VM migration
Most common operation for today’s virtualization Software Works well for CPU-heavy workloads I/O, graphics or compute-heavy workloads

VIRTUALIZATION OF A SYSTEM

AMD

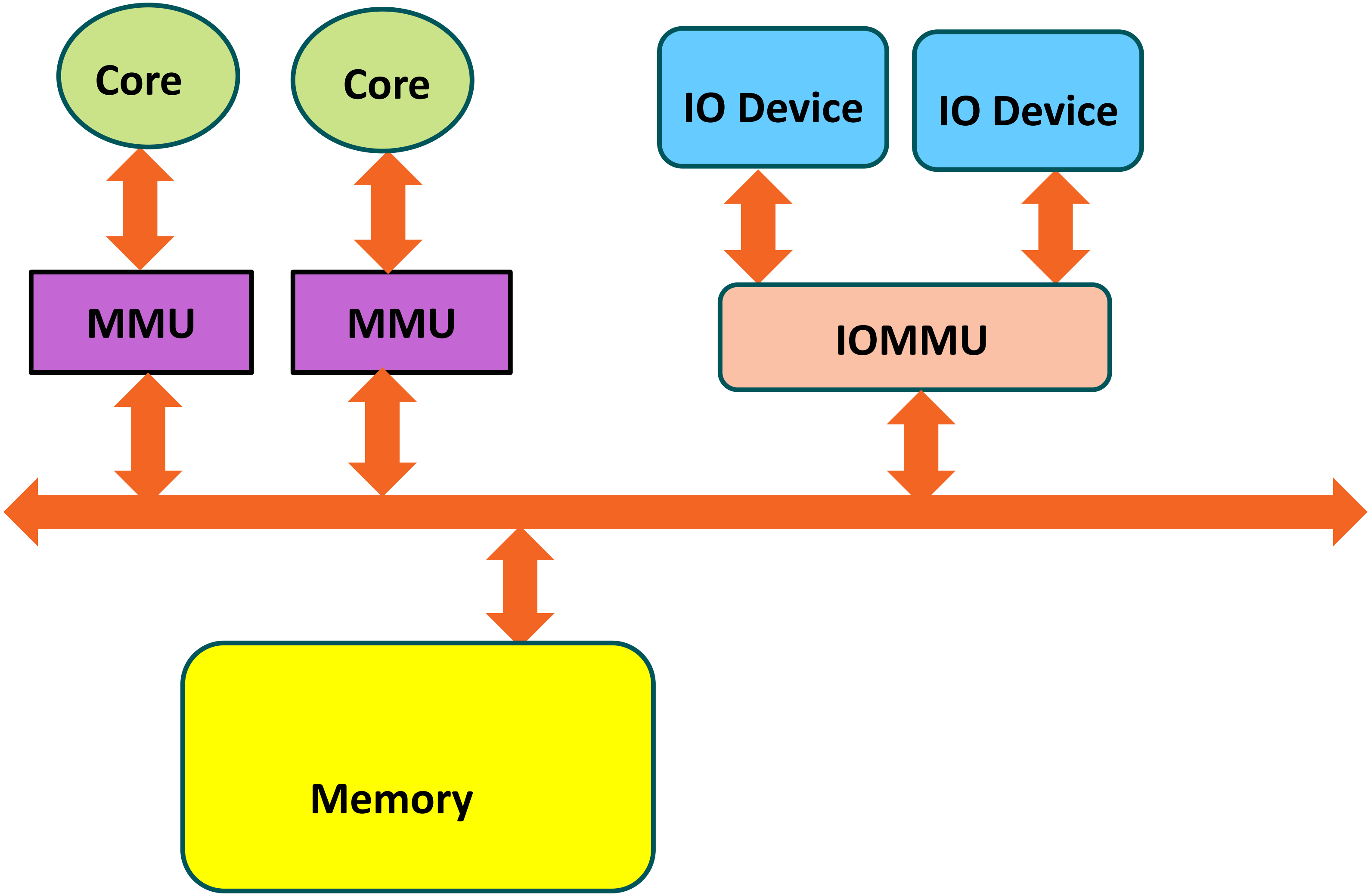
- Most CPUs today have support for system virtualization
  - Nested page tables (HV & OS levels), allow VMM/HV to assign and manage system memory and interrupts to Virtual Machines
- I/O devices are typically managed by HV/VMM software, either by…

Para-Virtualization	**Direct-Mapped Device & SR-IOV**
Guest device driver uses HV “hypercalls” Hypervisor manages HW operation (DMA)	Device function is mapped to guest OS Guest OS uses native HW drivers
Hypervisor SW validates and redirects I/O requests from Guest OS (overhead, slow)	Physical Device DMA must be limited and redirected by Hypervisor (via IOMMU),
Hypervisor arbitrates and schedules requests from multiple guest OS, allows VM migration	One device function per guest OS, physical memory must be committed
Most common operation for today’s virtualization Software Works well for CPU-heavy workloads I/O, graphics or compute-heavy workloads	I/O device must be resettable by HV when guest error puts it in undefined state SR-IOV is a variant of direct mapped I/O device provides 1 - n “virtual” devices in HW (PCI-SIG standard)

EFFICIENT I/O VIRTUALIZATION

HARDWARE IMPLEMENTED TECHNIQUE THROUGH IOMMU

- IOMMU validates DMA accesses and validates device interrupts



EFFICIENT IO VIRTUALIZATION WITH IOMMU

WHAT ARE THE BENEFITS?

- Using the IOMMU allows a Hypervisor to assign a physical device exclusively to a Guest VM without danger of memory corruption to other VMs

EFFICIENT IO VIRTUALIZATION WITH IOMMU

WHAT ARE THE BENEFITS?

- Using the IOMMU allows a Hypervisor to assign a physical device exclusively to a Guest VM without danger of memory corruption to other VMs
  - Beneficial if one VM requires near native performance

EFFICIENT IO VIRTUALIZATION WITH IOMMU

WHAT ARE THE BENEFITS?

- Using the IOMMU allows a Hypervisor to assign a physical device exclusively to a Guest VM without danger of memory corruption to other VMs
  - Beneficial if one VM requires near native performance
  - Or if OS needs to be “sandboxed” (because of suspected malware)

EFFICIENT IO VIRTUALIZATION WITH IOMMU

WHAT ARE THE BENEFITS?

- Using the IOMMU allows a Hypervisor to assign a physical device exclusively to a Guest VM without danger of memory corruption to other VMs
  - Beneficial if one VM requires near native performance
  - Or if OS needs to be “sandboxed” (because of suspected malware)
- Native driver can operate in the Guest OS

EFFICIENT IO VIRTUALIZATION WITH IOMMU

WHAT ARE THE BENEFITS?

- Using the IOMMU allows a Hypervisor to assign a physical device exclusively to a Guest VM without danger of memory corruption to other VMs
  - Beneficial if one VM requires near native performance
  - Or if OS needs to be "sandboxed" (because of suspected malware)
- Native driver can operate in the Guest OS
- IOMMU enforces Hypervisor policy on memory and system resource isolation for each of the Guest Virtual Machines

EFFICIENT IO VIRTUALIZATION WITH IOMMU

WHAT ARE THE BENEFITS?

- Using the IOMMU allows a Hypervisor to assign a physical device exclusively to a Guest VM without danger of memory corruption to other VMs
  - Beneficial if one VM requires near native performance
  - Or if OS needs to be “sandboxed” (because of suspected malware)
- Native driver can operate in the Guest OS
- IOMMU enforces Hypervisor policy on memory and system resource isolation for each of the Guest Virtual Machines
- IOMMU redirects device physical address set up by Guest OS driver (= Guest Physical Addresses) to the actual Host System Physical Address (SPA)

EFFICIENT IO VIRTUALIZATION WITH IOMMU

WHAT ARE THE BENEFITS?

- Using the IOMMU allows a Hypervisor to assign a physical device exclusively to a Guest VM without danger of memory corruption to other VMs
  - Beneficial if one VM requires near native performance
  - Or if OS needs to be “sandboxed” (because of suspected malware)
- Native driver can operate in the Guest OS
- IOMMU enforces Hypervisor policy on memory and system resource isolation for each of the Guest Virtual Machines
- IOMMU redirects device physical address set up by Guest OS driver (= Guest Physical Addresses) to the actual Host System Physical Address (SPA)
  - Useful for platform resources that have “well-known” addresses like legacy devices or system resources like APIC (Advanced Programmable Interrupt Controller)

EFFICIENT IO VIRTUALIZATION WITH IOMMU

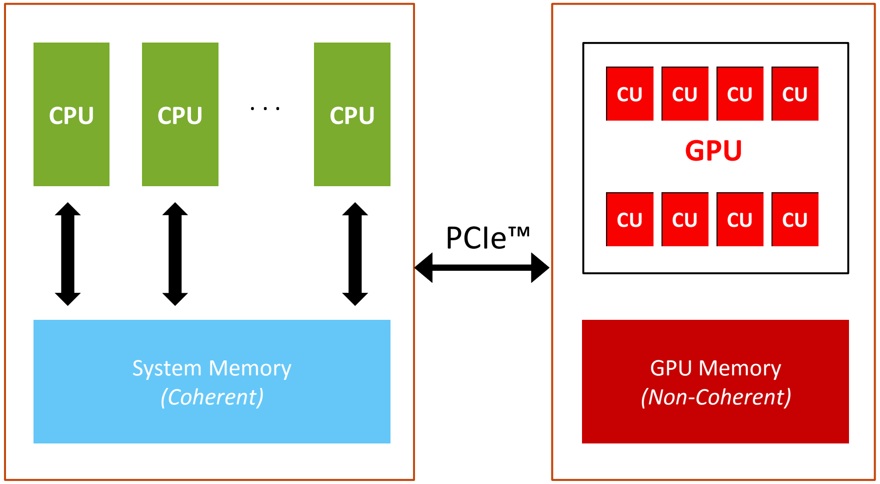
WHAT ARE THE BENEFITS?

- Using the IOMMU allows a Hypervisor to assign a physical device exclusively to a Guest VM without danger of memory corruption to other VMs
  - Beneficial if one VM requires near native performance
  - Or if OS needs to be “sandboxed” (because of suspected malware)
- Native driver can operate in the Guest OS
- IOMMU enforces Hypervisor policy on memory and system resource isolation for each of the Guest Virtual Machines
- IOMMU redirects device physical address set up by Guest OS driver (= Guest Physical Addresses) to the actual Host System Physical Address (SPA)
  - Useful for platform resources that have “well-known” addresses like legacy devices or system resources like APIC (Advanced Programmable Interrupt Controller)
- Allows near-native device performance for high-performance devices with low system impact

IOMMU USECASE: ENABLING HETEROGENEOUS COMPUTING

LEGACY GPU COMPUTE

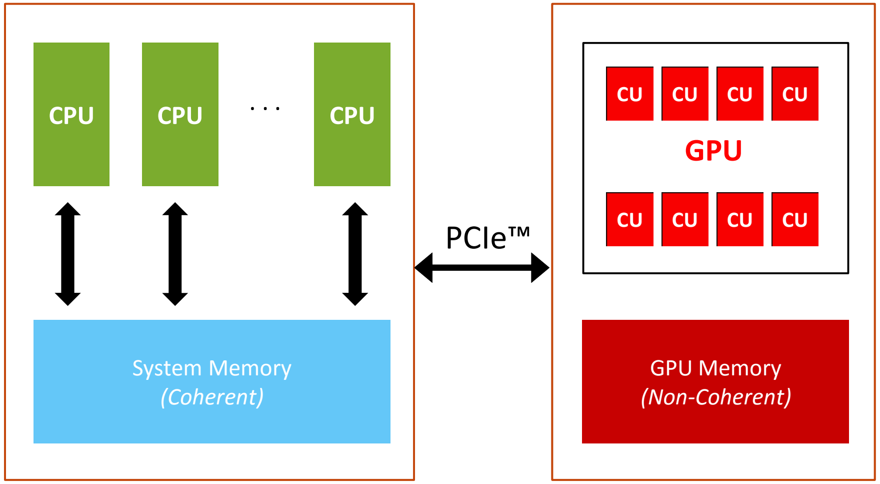
The limiters that need to be fixed to unleash programmers:



LEGACY GPU COMPUTE

The limiters that need to be fixed to unleash programmers:

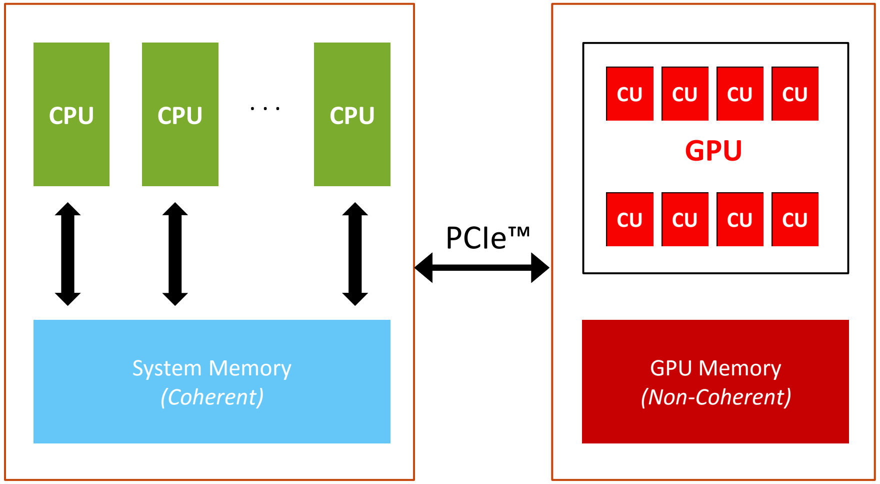
- Multiple memory pools, multiple address spaces



LEGACY GPU COMPUTE

The limiters that need to be fixed to unleash programmers:

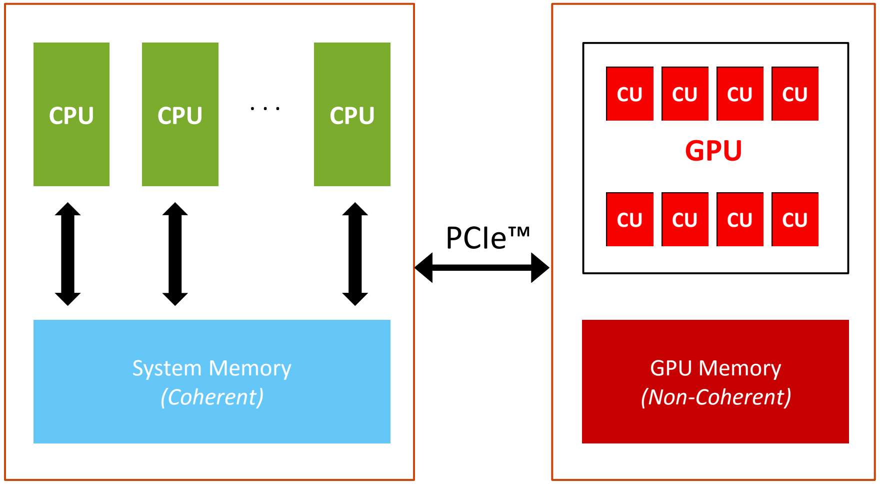
- Multiple memory pools, multiple address spaces
- High overhead dispatch, data copies across PCIe



LEGACY GPU COMPUTE

The limiters that need to be fixed to unleash programmers:

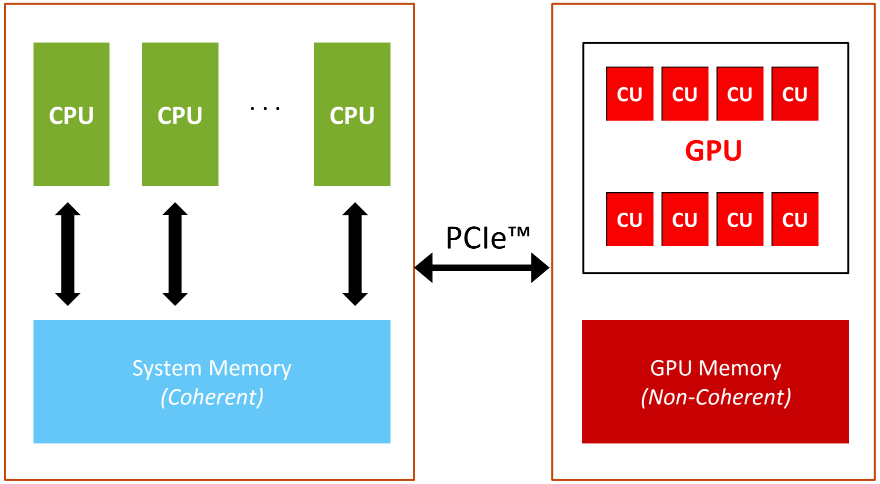
- Multiple memory pools, multiple address spaces
- High overhead dispatch, data copies across PCIe
- New languages and APIs for GPU programming necessary (OpenCL, etc.)



LEGACY GPU COMPUTE

The limiters that need to be fixed to unleash programmers:

- Multiple memory pools, multiple address spaces
- High overhead dispatch, data copies across PCIe
- New languages and APIs for GPU programming necessary (OpenCL, etc.)
  - And sometimes proprietary environments

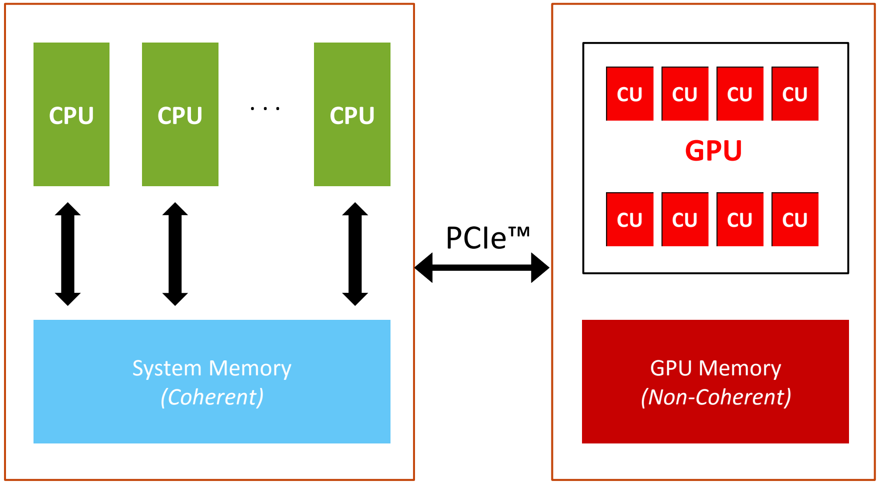


LEGACY GPU COMPUTE

The limiters that need to be fixed to unleash programmers:

- Multiple memory pools, multiple address spaces
- High overhead dispatch, data copies across PCIe
- New languages and APIs for GPU programming necessary (OpenCL, etc.)
  - And sometimes proprietary environments

➔ Dual source development



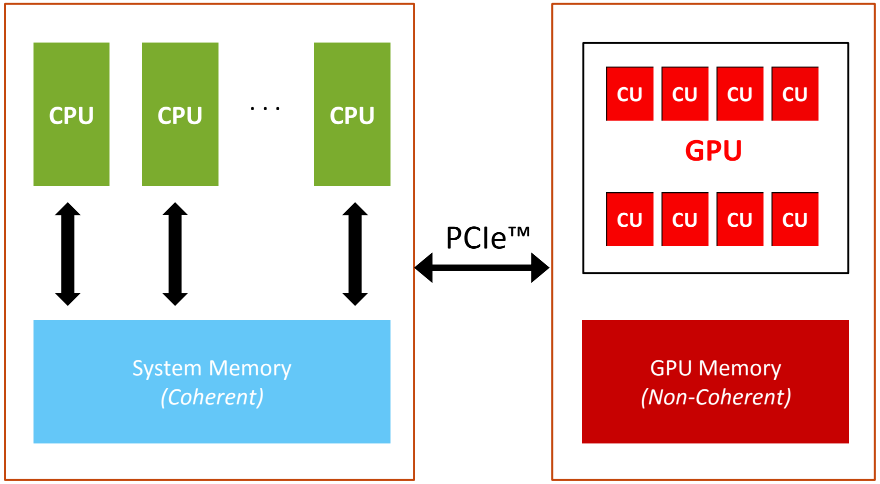
LEGACY GPU COMPUTE

The limiters that need to be fixed to unleash programmers:

- Multiple memory pools, multiple address spaces
- High overhead dispatch, data copies across PCIe
- New languages and APIs for GPU programming necessary (OpenCL, etc.)
  - And sometimes proprietary environments

➔ Dual source development

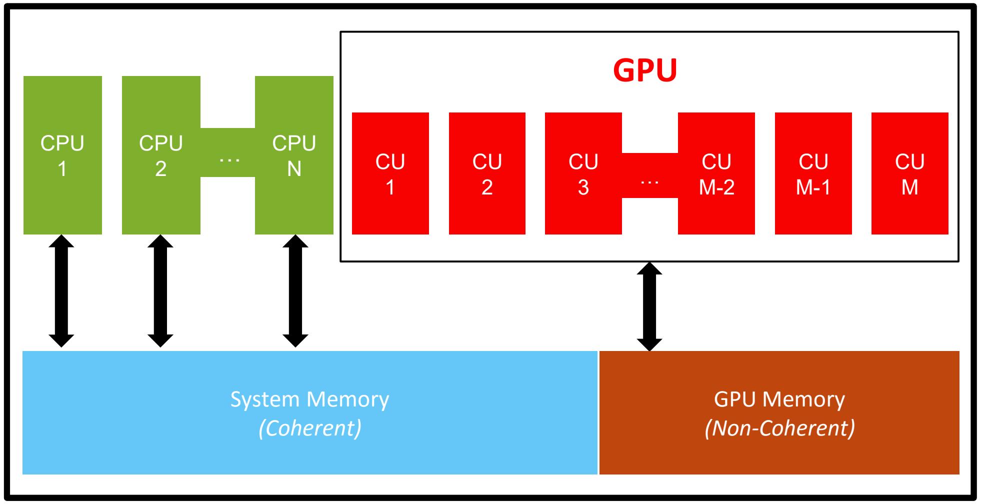
- Expert programmers only



THE PREVIOUS APUS AND SOCS, PHYSICAL INTEGRATION

- Some memory copies are gone, because the same memory is accessed

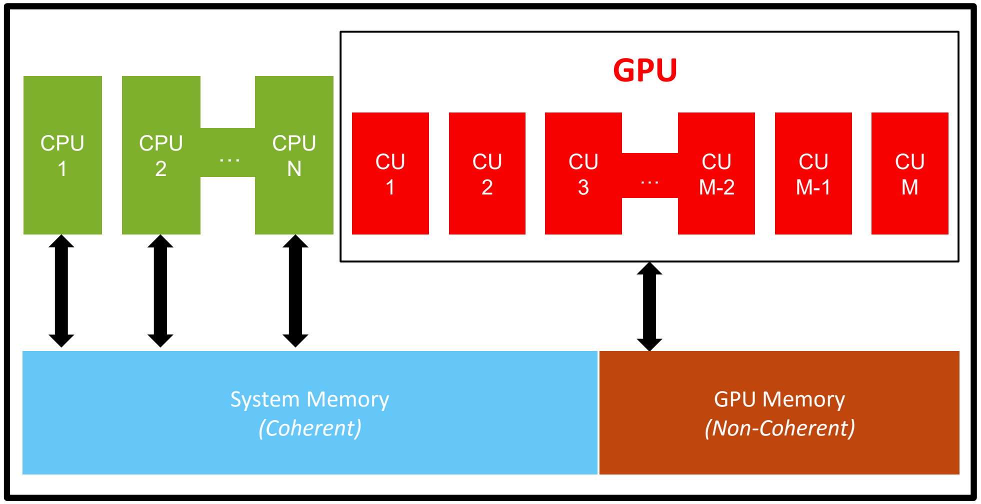
Physical Integration



THE PREVIOUS APUS AND SOCS, PHYSICAL INTEGRATION

- Some memory copies are gone, because the same memory is accessed
  - But the memory is not accessible concurrently, because of cache policies

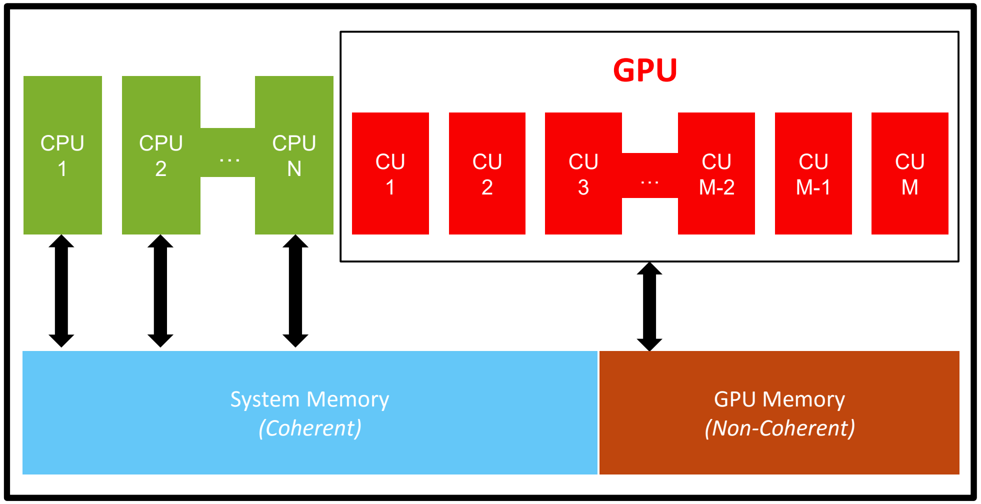
Physical Integration



THE PREVIOUS APUS AND SOCS, PHYSICAL INTEGRATION

- Some memory copies are gone, because the same memory is accessed
  - But the memory is not accessible concurrently, because of cache policies
- Two memory pools remain (cache coherent + non-coherent memory regions)

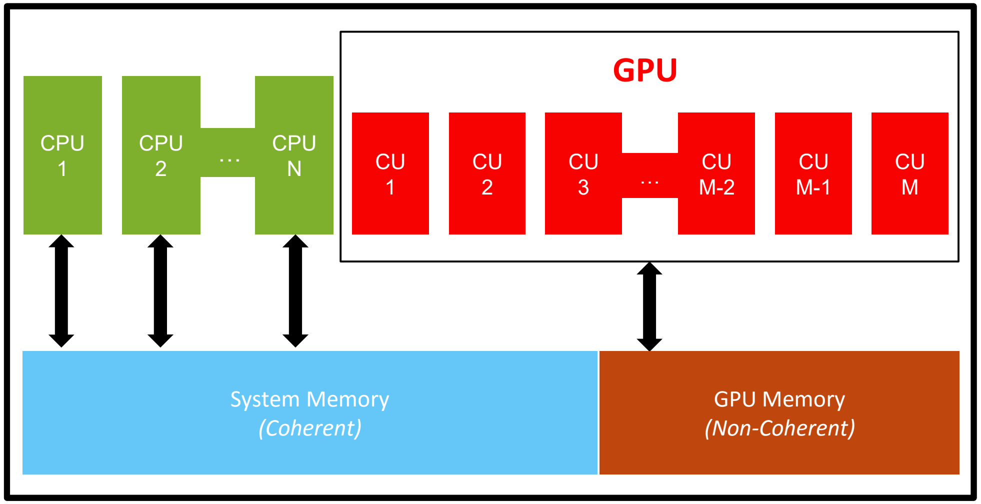
Physical Integration



THE PREVIOUS APUS AND SOCS, PHYSICAL INTEGRATION

- Some memory copies are gone, because the same memory is accessed
  - But the memory is not accessible concurrently, because of cache policies
- Two memory pools remain (cache coherent + non-coherent memory regions)

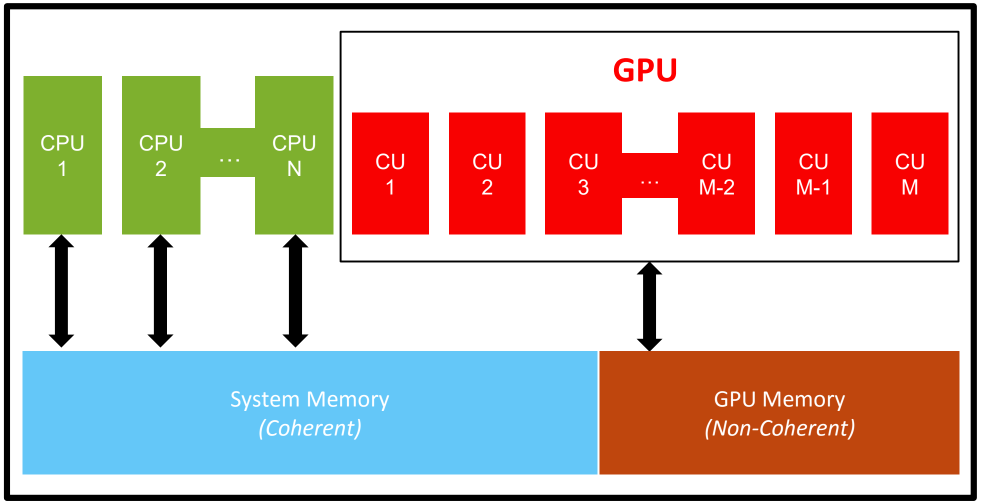
Physical Integration



THE PREVIOUS APUS AND SOCS, PHYSICAL INTEGRATION

- Some memory copies are gone, because the same memory is accessed
  - But the memory is not accessible concurrently, because of cache policies
- Two memory pools remain (cache coherent + non-coherent memory regions)
- Jobs are still queued through the OS driver chain and suffer from overhead

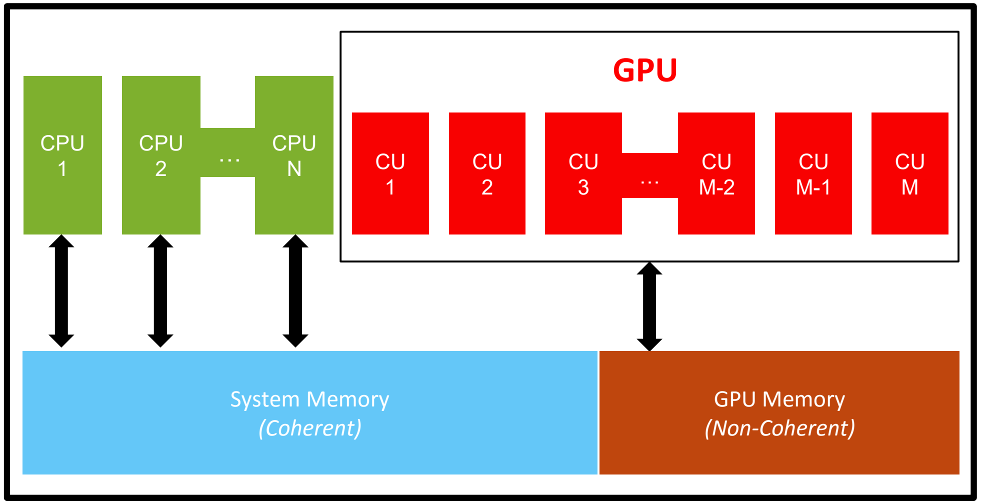
Physical Integration



THE PREVIOUS APUS AND SOCS, PHYSICAL INTEGRATION

- Some memory copies are gone, because the same memory is accessed
  - But the memory is not accessible concurrently, because of cache policies
- Two memory pools remain (cache coherent + non-coherent memory regions)
- Jobs are still queued through the OS driver chain and suffer from overhead
- Still requires expert programmers to get performance

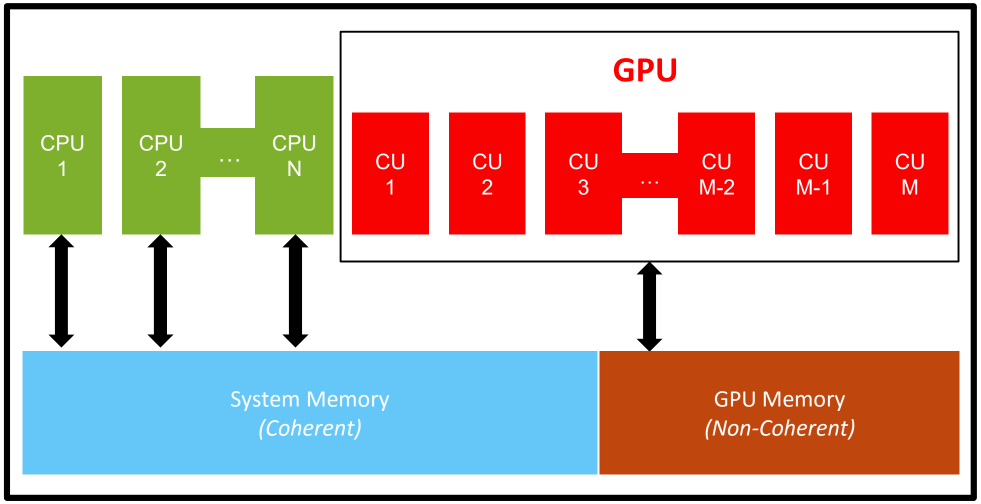
Physical Integration



THE PREVIOUS APUS AND SOCS, PHYSICAL INTEGRATION

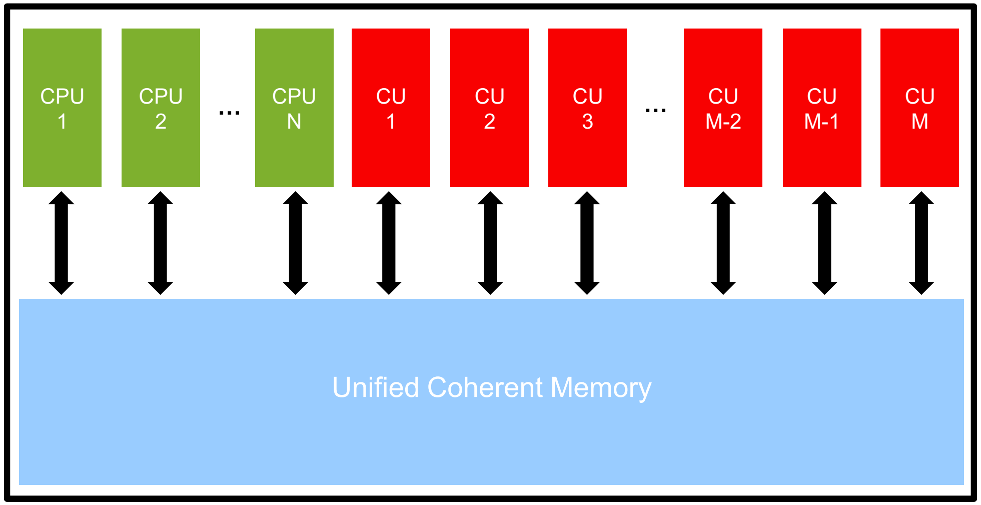
- Some memory copies are gone, because the same memory is accessed
  - But the memory is not accessible concurrently, because of cache policies
- Two memory pools remain (cache coherent + non-coherent memory regions)
- Jobs are still queued through the OS driver chain and suffer from overhead
- Still requires expert programmers to get performance
- **This is only an intermediate step in the journey**

Physical Integration



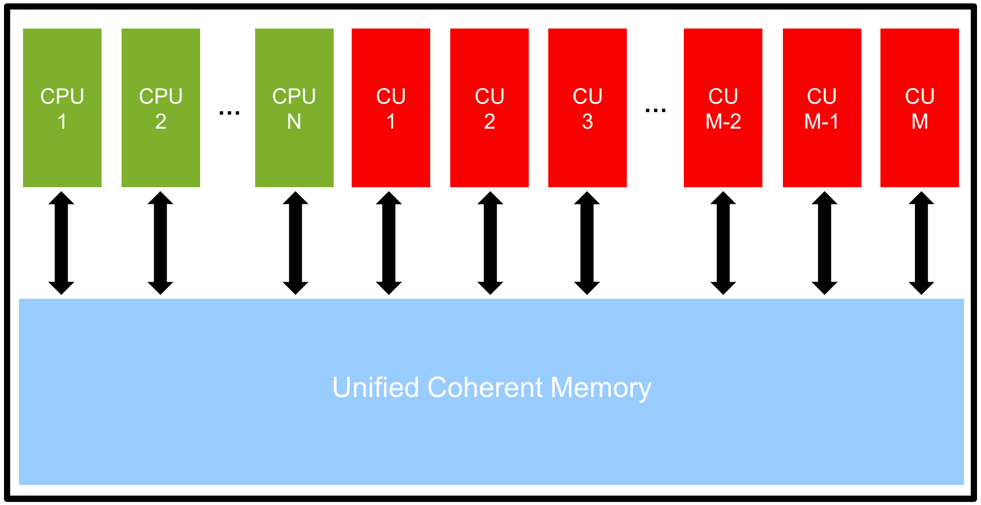
AN HSA ENABLED SOC

- **Unified Coherent Memory enables data sharing across all processors**



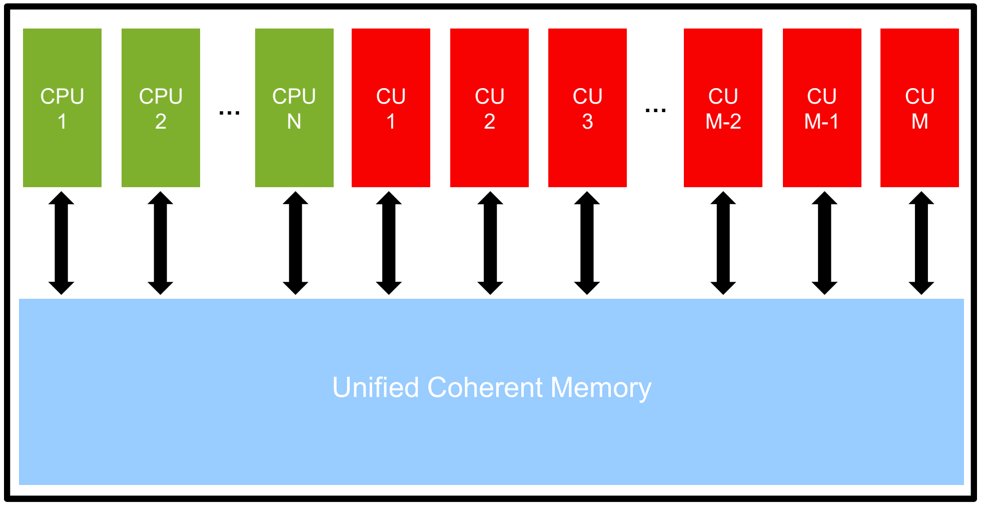
AN HSA ENABLED SOC

- **Unified Coherent Memory enables data sharing across all processors**



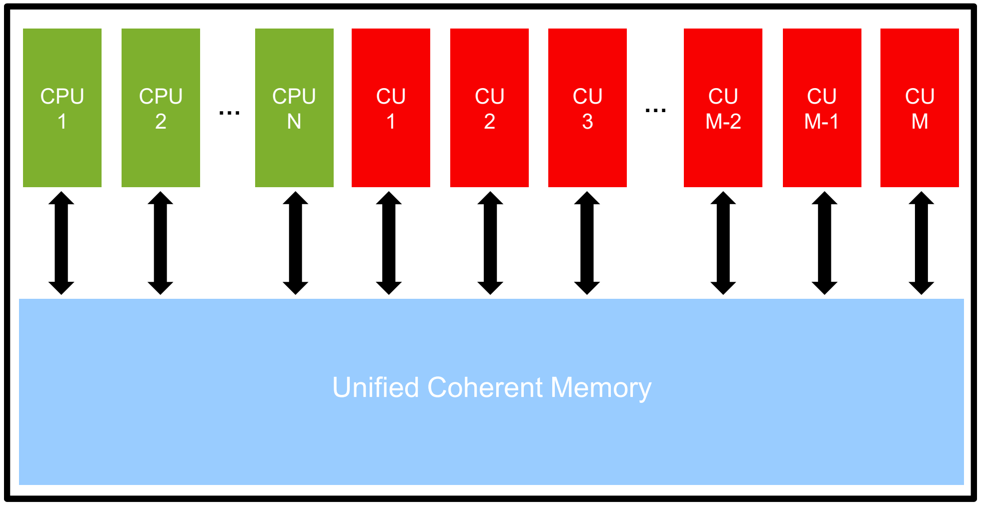
AN HSA ENABLED SOC

- **Unified Coherent Memory enables data sharing across all processors**
- Processors architected to operate cooperatively



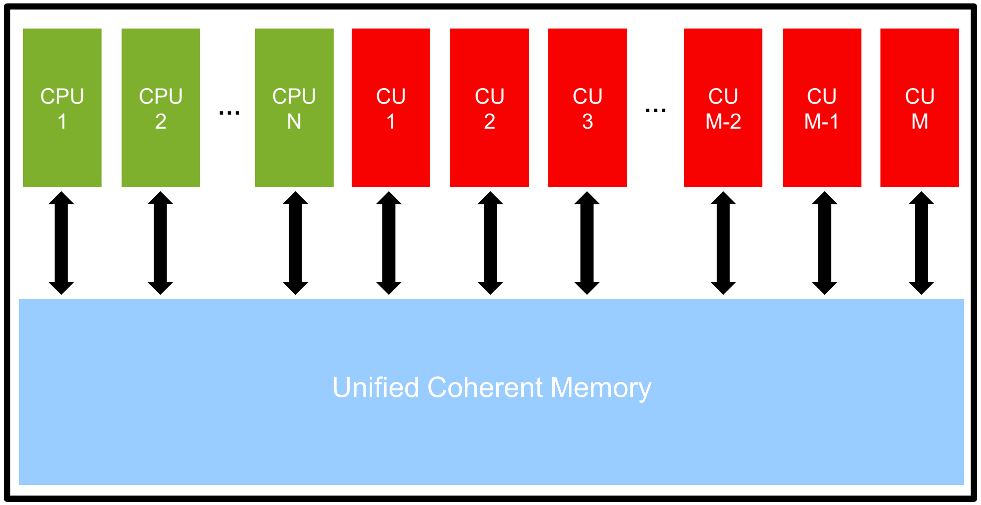
AN HSA ENABLED SOC

- **Unified Coherent Memory enables data sharing across all processors**
- Processors architected to operate cooperatively
  - – Can exchange data “on the fly”, similar to what CPU threads do



AN HSA ENABLED SOC

- **Unified Coherent Memory enables data sharing across all processors**
- Processors architected to operate cooperatively
  - Can exchange data “on the fly”, similar to what CPU threads do
  - The lower job dispatch overhead allows tasks to be handled by the GPU that previously were “too costly” to transfer over
- Designed to enable the application running on different processors without substantially changing the programming logic



IOMMU: A BUILDING BLOCK FOR HSA

REDUCING THE OVERHEAD TO CALL THE GPU OR OTHER ACCELERATORS

The goals of the Heterogeneous System Architecture (HSA)
and where the IOMMU helps:

IOMMU: A BUILDING BLOCK FOR HSA

REDUCING THE OVERHEAD TO CALL THE GPU OR OTHER ACCELERATORS

The goals of the Heterogeneous System Architecture (HSA) **and where the IOMMU helps:**

- Use of accelerators as a first-class, peer processor within the system

IOMMU: A BUILDING BLOCK FOR HSA

REDUCING THE OVERHEAD TO CALL THE GPU OR OTHER ACCELERATORS

The goals of the Heterogeneous System Architecture (HSA) **and where the IOMMU helps:**

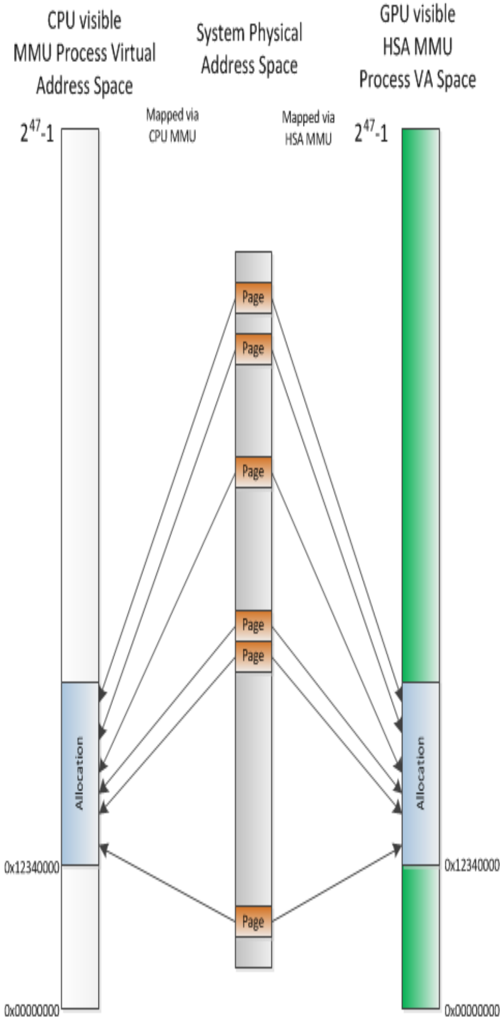
- Use of accelerators as a first-class, peer processor within the system
  - **Unified process address space access across all processors**
    - **Shared Virtual Memory (SVM), “GPU ptr == CPU ptr”**

IOMMU: A BUILDING BLOCK FOR HSA

REDUCING THE OVERHEAD TO CALL THE GPU OR OTHER ACCELERATORS

The goals of the Heterogeneous System Architecture (HSA) **and where the IOMMU helps:**

- Use of accelerators as a first-class, peer processor within the system
  - **Unified process address space access across all processors**
    - **Shared Virtual Memory (SVM), "GPU ptr == CPU ptr"**

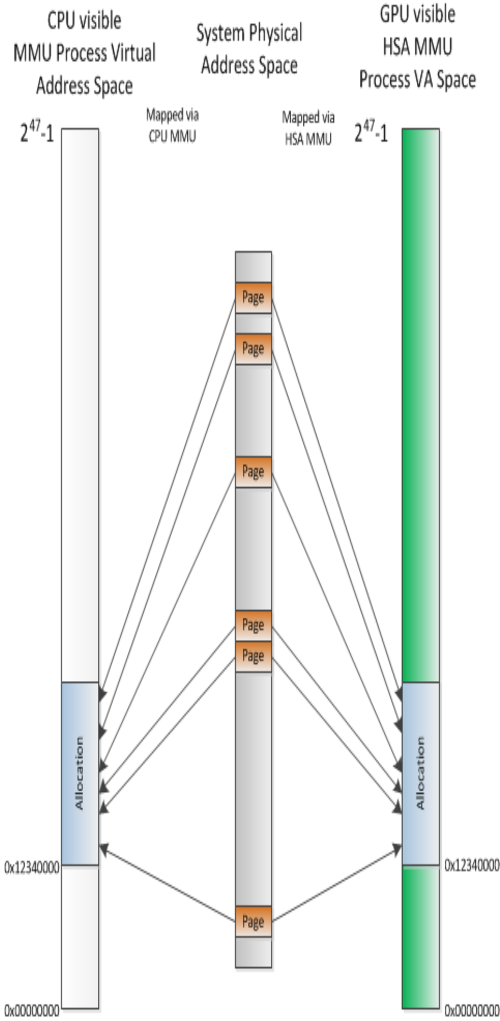


IOMMU: A BUILDING BLOCK FOR HSA

REDUCING THE OVERHEAD TO CALL THE GPU OR OTHER ACCELERATORS

The goals of the Heterogeneous System Architecture (HSA) **and where the IOMMU helps:**

- Use of accelerators as a first-class, peer processor within the system
  - **Unified process address space access across all processors**
    - **Shared Virtual Memory (SVM), "GPU ptr == CPU ptr"**
  - Accelerator operates in **pageable system memory***

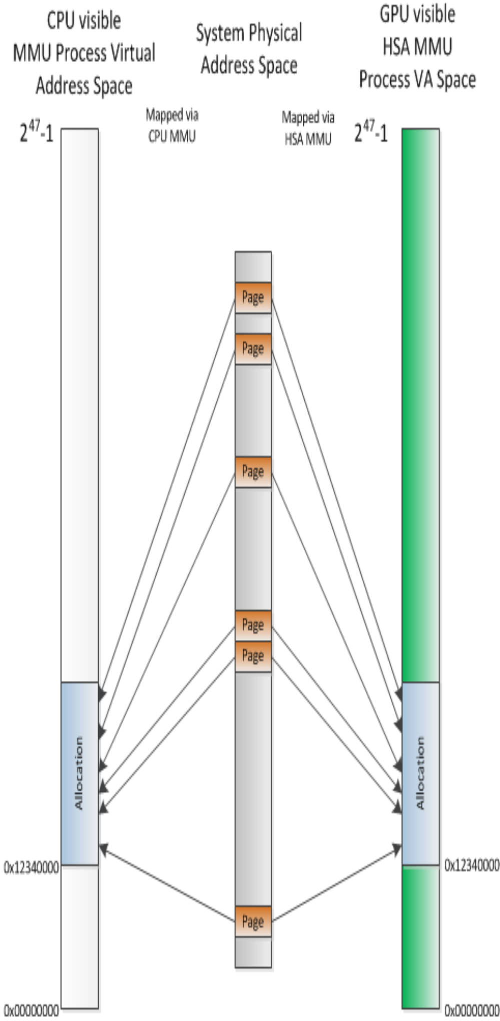


IOMMU: A BUILDING BLOCK FOR HSA

REDUCING THE OVERHEAD TO CALL THE GPU OR OTHER ACCELERATORS

The goals of the Heterogeneous System Architecture (HSA) **and where the IOMMU helps:**

- Use of accelerators as a first-class, peer processor within the system
  - **Unified process address space access across all processors**
    - **Shared Virtual Memory (SVM), "GPU ptr == CPU ptr"**
  - Accelerator operates in **pageable system memory***



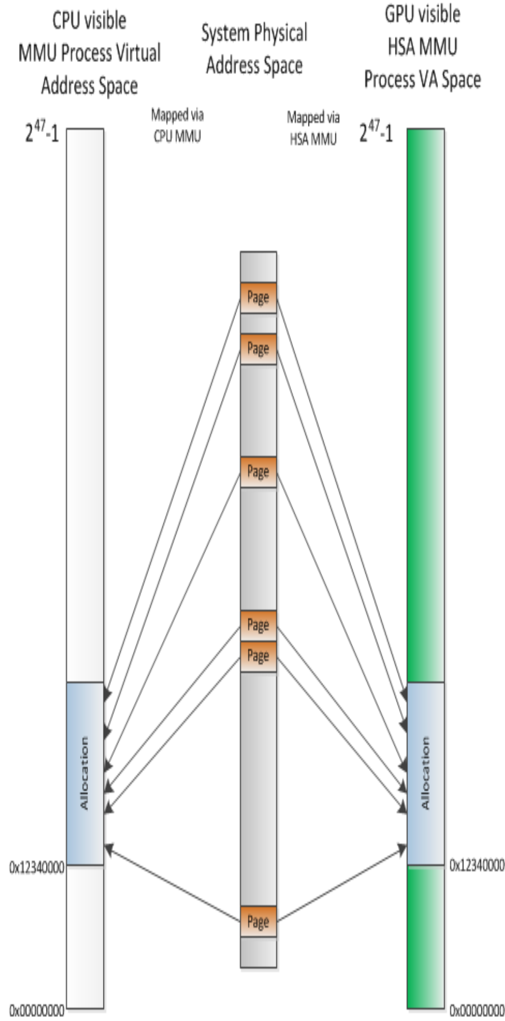
*with OS support & ATS/PRI

IOMMU: A BUILDING BLOCK FOR HSA

REDUCING THE OVERHEAD TO CALL THE GPU OR OTHER ACCELERATORS

The goals of the Heterogeneous System Architecture (HSA) **and where the IOMMU helps:**

- Use of accelerators as a first-class, peer processor within the system
  - **Unified process address space access across all processors**
    - **Shared Virtual Memory (SVM), “GPU ptr == CPU ptr”**
  - Accelerator operates in **pageable system memory***
  - Cache coherency between the CPU and accelerator caches
  - User mode dispatch/scheduling reduces job-dispatch overhead
  - QoS with preemption/context switch of GPU Compute Units



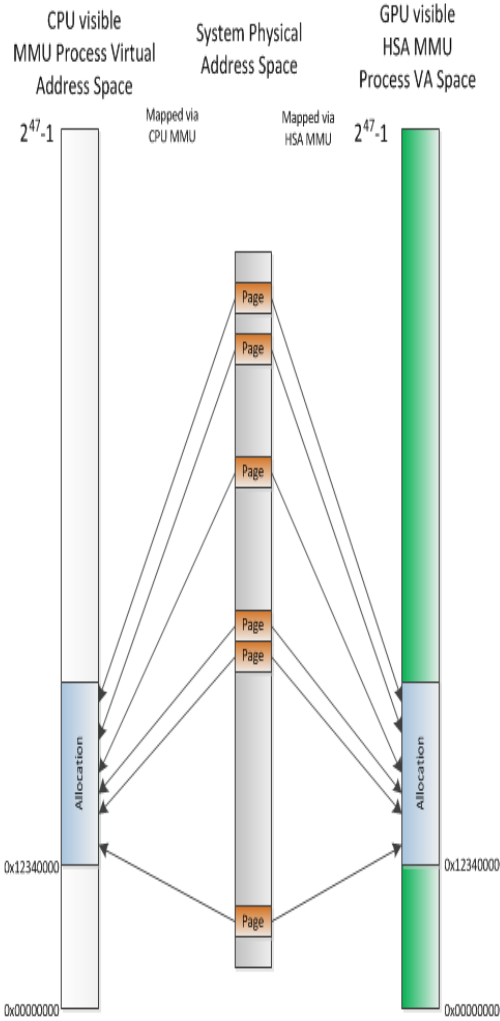
*with OS support & ATS/PRI

IOMMU: A BUILDING BLOCK FOR HSA

REDUCING THE OVERHEAD TO CALL THE GPU OR OTHER ACCELERATORS

The goals of the Heterogeneous System Architecture (HSA) **and where the IOMMU helps:**

- Use of accelerators as a first-class, peer processor within the system
  - **Unified process address space access across all processors**
    - **Shared Virtual Memory (SVM), “GPU ptr == CPU ptr”**
  - Accelerator operates in **pageable system memory***
  - Cache coherency between the CPU and accelerator caches
  - User mode dispatch/scheduling reduces job-dispatch overhead
  - QoS with preemption/context switch of GPU Compute Units
- **The IOMMU enforces control of GPU access to memory**



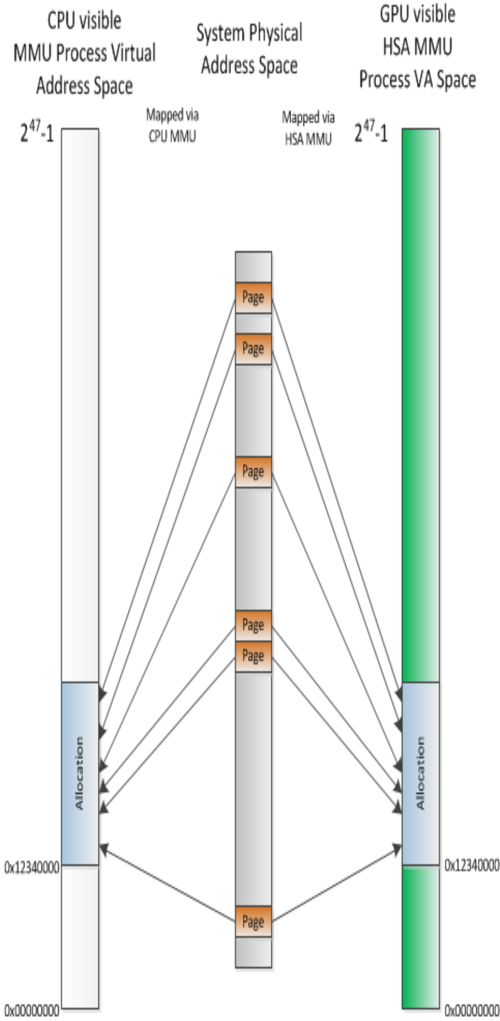
*with OS support & ATS/PRI

IOMMU: A BUILDING BLOCK FOR HSA

REDUCING THE OVERHEAD TO CALL THE GPU OR OTHER ACCELERATORS

The goals of the Heterogeneous System Architecture (HSA) **and where the IOMMU helps:**

- Use of accelerators as a first-class, peer processor within the system
  - **Unified process address space access across all processors**
    - **Shared Virtual Memory (SVM), “GPU ptr == CPU ptr”**
  - Accelerator operates in **pageable system memory***
  - Cache coherency between the CPU and accelerator caches
  - User mode dispatch/scheduling reduces job-dispatch overhead
  - QoS with preemption/context switch of GPU Compute Units
- **The IOMMU enforces control of GPU access to memory**
  - OS can efficiently and safely share process page tables with accelerators (requires ATS/PRI protocol support)



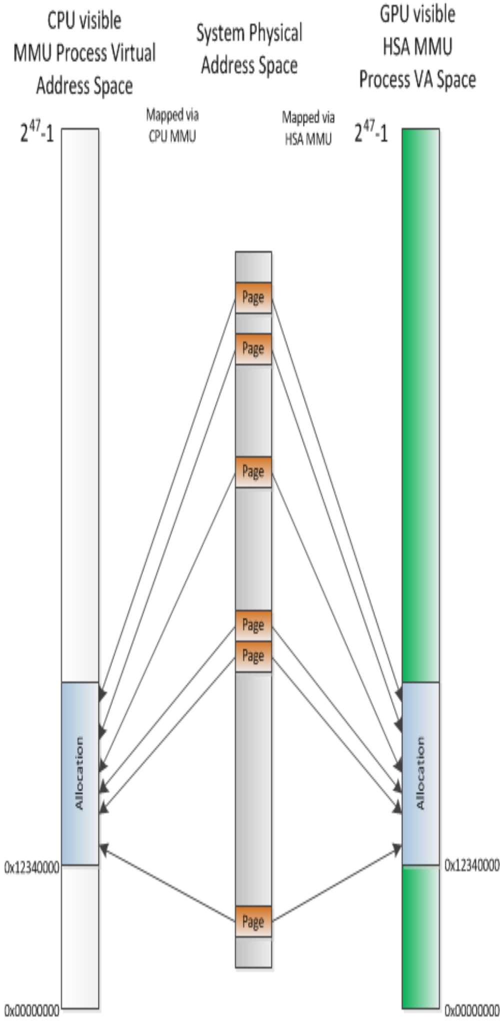
*with OS support & ATS/PRI

IOMMU: A BUILDING BLOCK FOR HSA

REDUCING THE OVERHEAD TO CALL THE GPU OR OTHER ACCELERATORS

The goals of the Heterogeneous System Architecture (HSA) **and where the IOMMU helps:**

- Use of accelerators as a first-class, peer processor within the system
  - **Unified process address space access across all processors**
    - **Shared Virtual Memory (SVM), “GPU ptr == CPU ptr”**
  - Accelerator operates in **pageable system memory***
  - Cache coherency between the CPU and accelerator caches
  - User mode dispatch/scheduling reduces job-dispatch overhead
  - QoS with preemption/context switch of GPU Compute Units
- **The IOMMU enforces control of GPU access to memory**
  - OS can efficiently and safely share process page tables with accelerators (requires ATS/PRI protocol support)
  - Accelerators can’t step outside of the OS-set boundaries

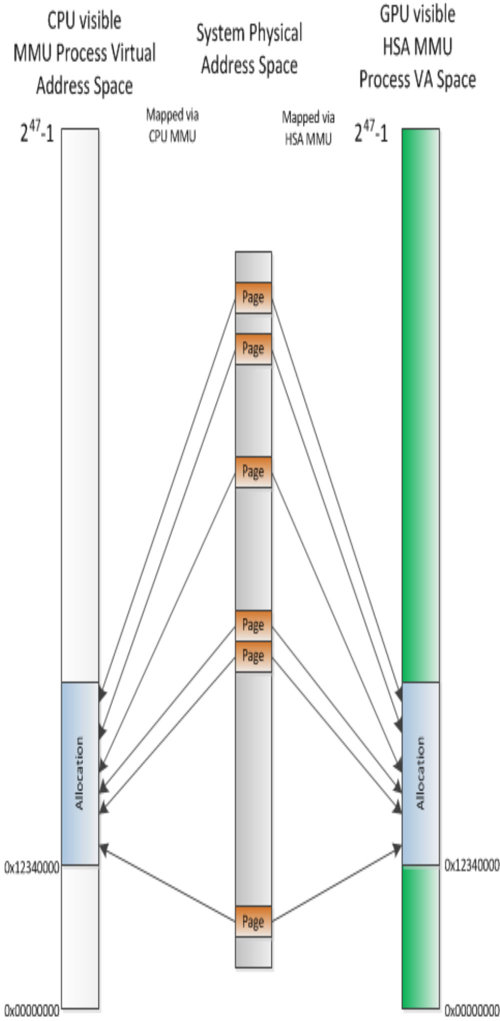


*with OS support & ATS/PRI

IOMMU: A BUILDING BLOCK FOR HSA

REDUCING THE OVERHEAD TO CALL THE GPU OR OTHER ACCELERATORS

The benefits of the Heterogeneous System Architecture:

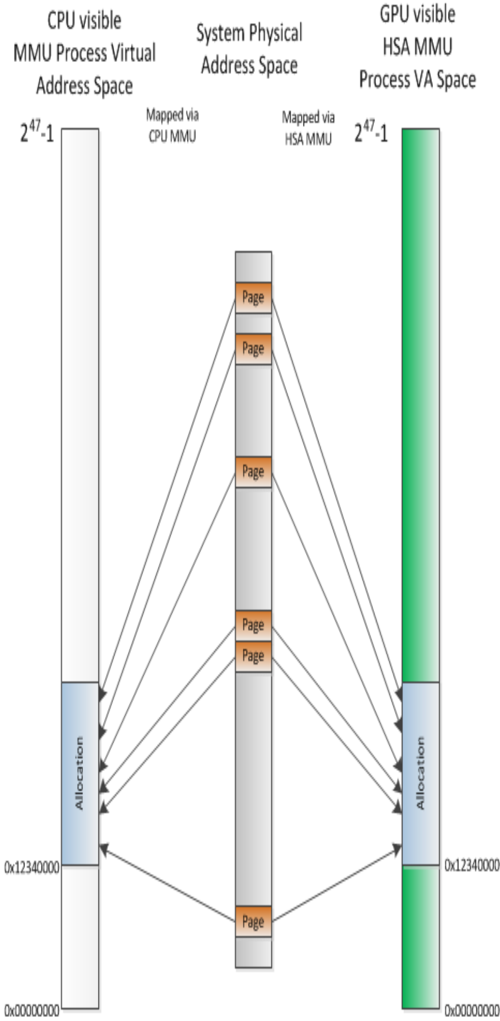


IOMMU: A BUILDING BLOCK FOR HSA

REDUCING THE OVERHEAD TO CALL THE GPU OR OTHER ACCELERATORS

The benefits of the Heterogeneous System Architecture:

- Pageable memory access is validated and handled directly by the OS memory manager via AMD IOMMU

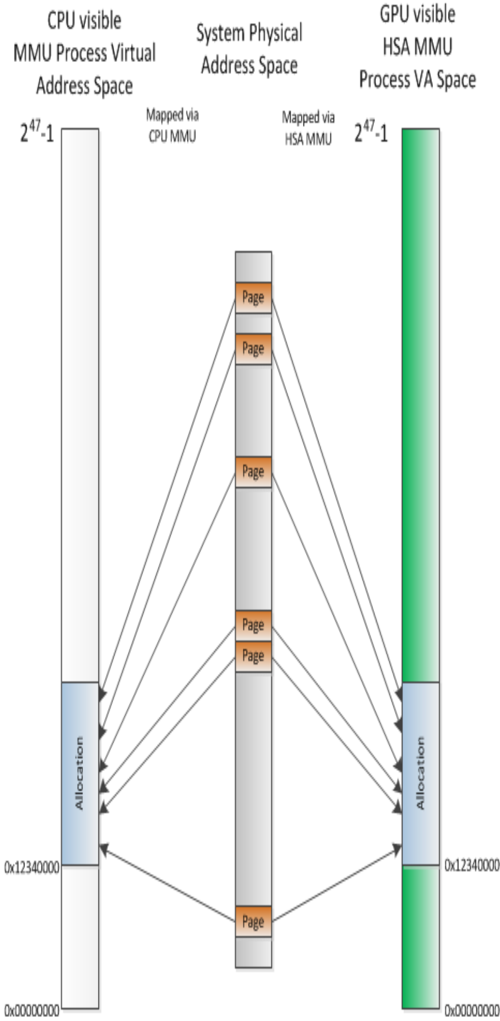


IOMMU: A BUILDING BLOCK FOR HSA

REDUCING THE OVERHEAD TO CALL THE GPU OR OTHER ACCELERATORS

The benefits of the Heterogeneous System Architecture:

- Pageable memory access is validated and handled directly by the OS memory manager via AMD IOMMU
- Application data structures can be directly parsed by the accelerator and pointer links followed without CPU help

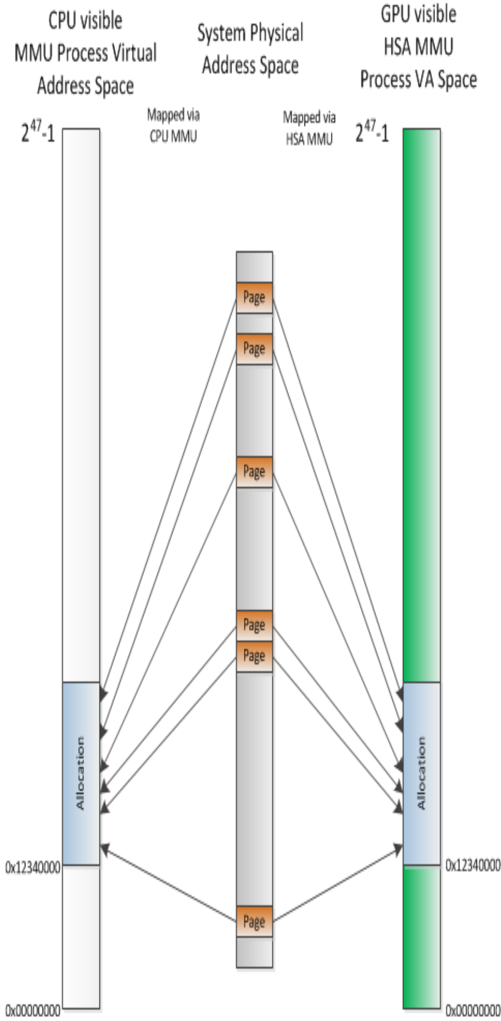


IOMMU: A BUILDING BLOCK FOR HSA

REDUCING THE OVERHEAD TO CALL THE GPU OR OTHER ACCELERATORS

The benefits of the Heterogeneous System Architecture:

- Pageable memory access is validated and handled directly by the OS memory manager via AMD IOMMU
- Application data structures can be directly parsed by the accelerator and pointer links followed without CPU help
- Common high level languages and tools (compilers, runtimes, …) port easily to accelerators

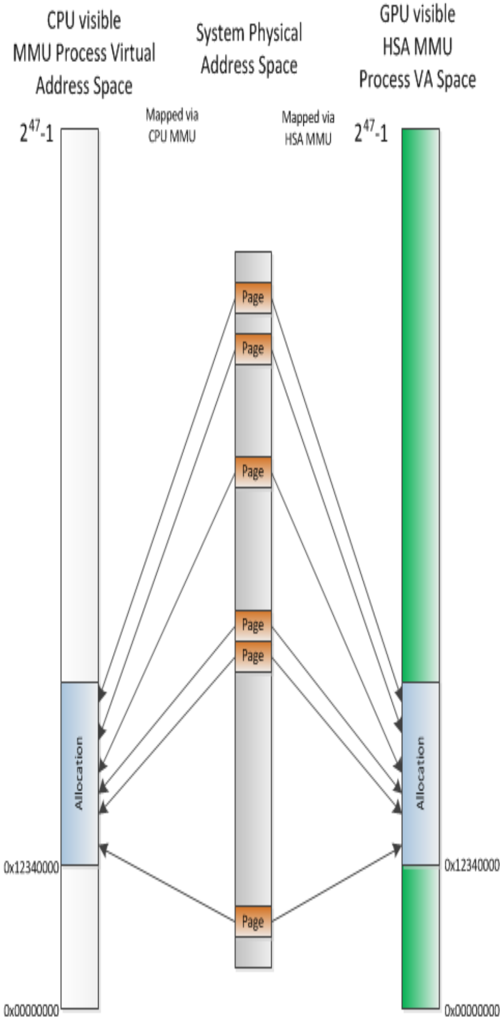


IOMMU: A BUILDING BLOCK FOR HSA

REDUCING THE OVERHEAD TO CALL THE GPU OR OTHER ACCELERATORS

The benefits of the Heterogeneous System Architecture:

- Pageable memory access is validated and handled directly by the OS memory manager via AMD IOMMU
- Application data structures can be directly parsed by the accelerator and pointer links followed without CPU help
- Common high level languages and tools (compilers, runtimes, …) port easily to accelerators
  - C/C++, Python, Java, … already have open source implementations

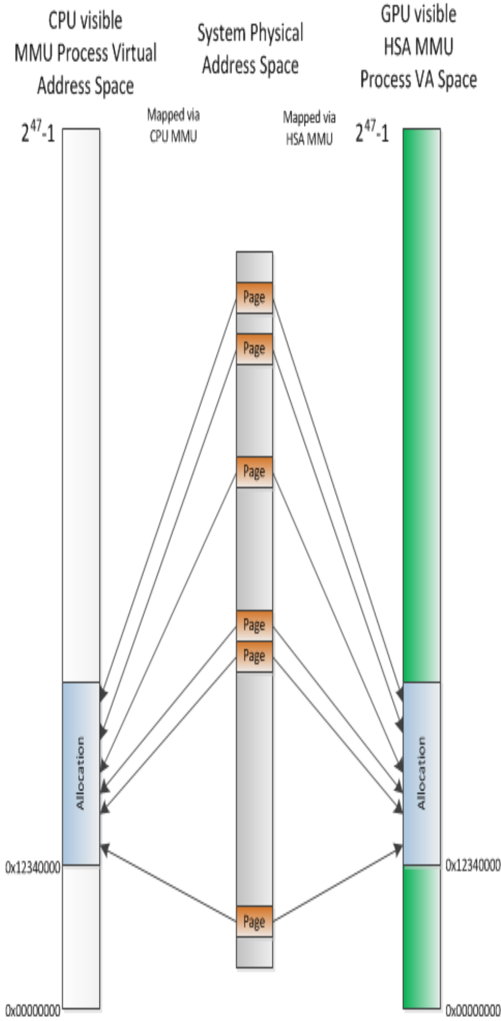


IOMMU: A BUILDING BLOCK FOR HSA

REDUCING THE OVERHEAD TO CALL THE GPU OR OTHER ACCELERATORS

The benefits of the Heterogeneous System Architecture:

- Pageable memory access is validated and handled directly by the OS memory manager via AMD IOMMU
- Application data structures can be directly parsed by the accelerator and pointer links followed without CPU help
- Common high level languages and tools (compilers, runtimes, …) port easily to accelerators
  - C/C++, Python, Java, … already have open source implementations
  - Many more languages to follow

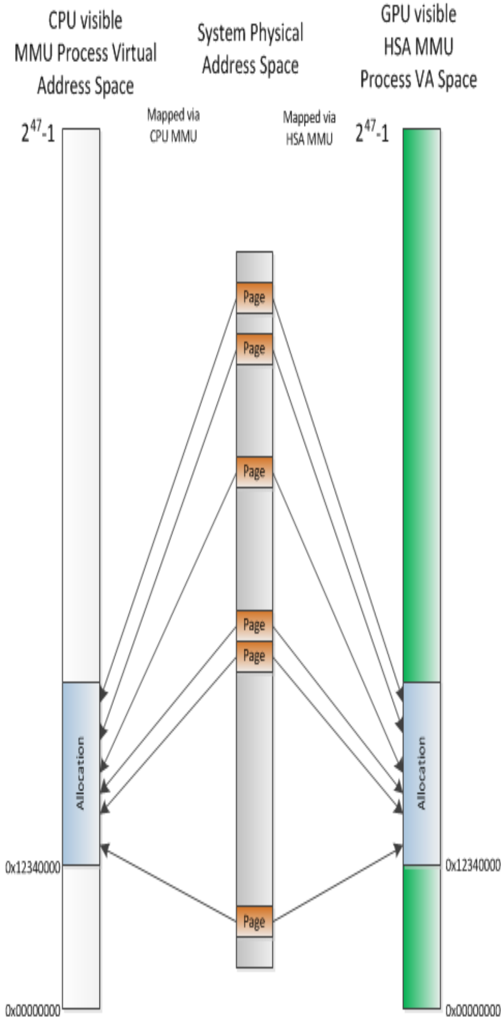


IOMMU: A BUILDING BLOCK FOR HSA

REDUCING THE OVERHEAD TO CALL THE GPU OR OTHER ACCELERATORS

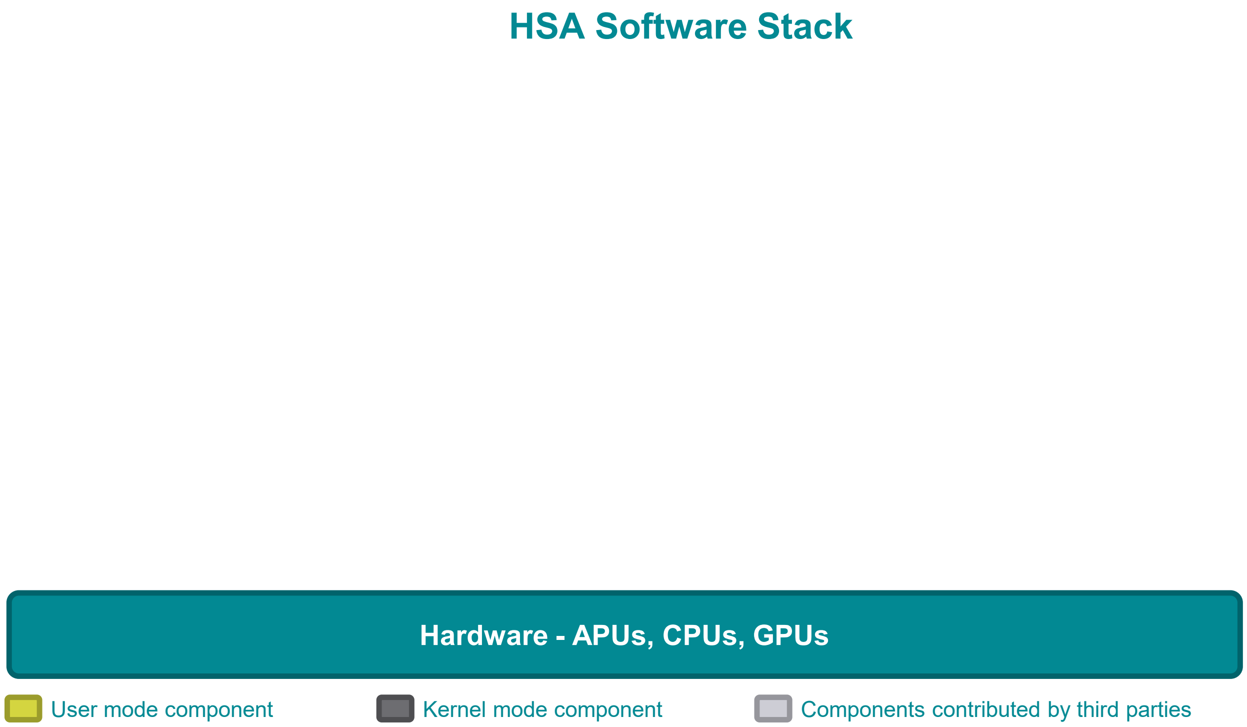
The benefits of the Heterogeneous System Architecture:

- Pageable memory access is validated and handled directly by the OS memory manager via AMD IOMMU
- Application data structures can be directly parsed by the accelerator and pointer links followed without CPU help
- Common high level languages and tools (compilers, runtimes, …) port easily to accelerators
  - C/C++, Python, Java, … already have open source implementations
  - Many more languages to follow
- **IOMMU making it easier for programmers to use GPUs and other accelerators safely and efficiently**



EVOLUTION OF THE SOFTWARE STACK – A COMPARISON

- Goal of the software stack is to focus on high-level language support



HSA Software Stack

Hardware - APUs, CPUs, GPUs

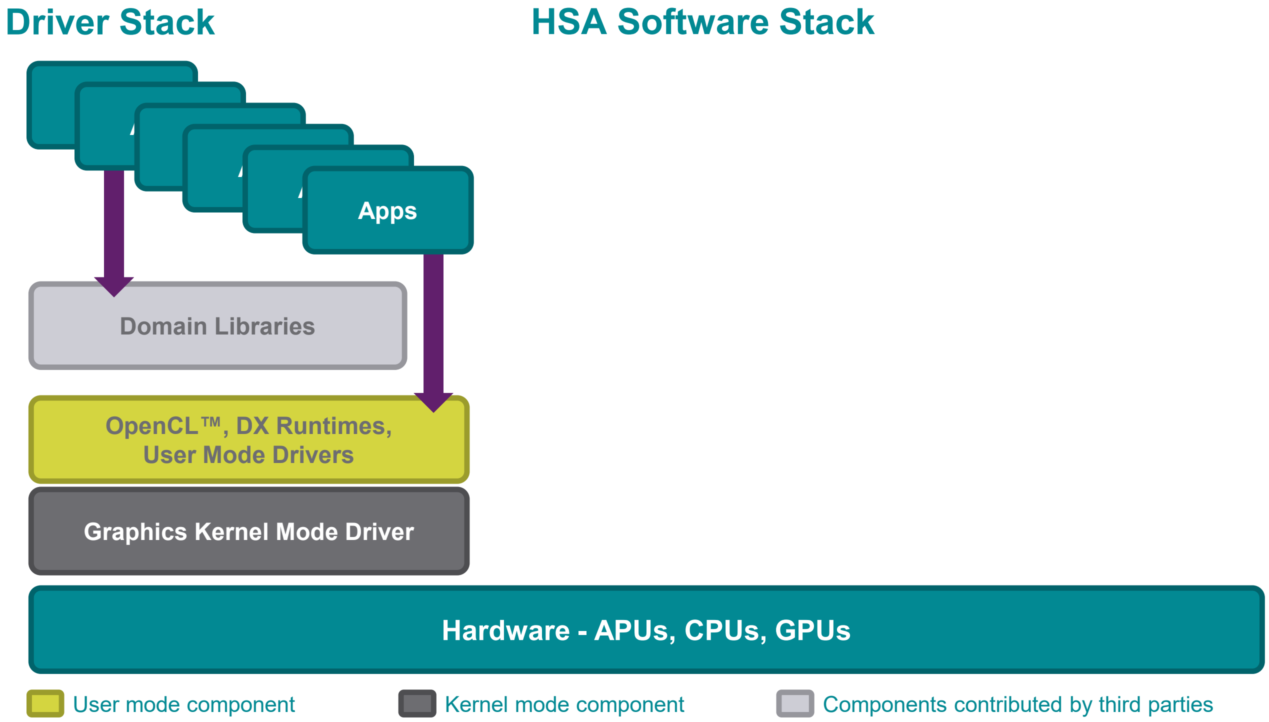
🟨 User mode component

⬛ Kernel mode component

⬜ Components contributed by third parties

EVOLUTION OF THE SOFTWARE STACK – A COMPARISON

- Goal of the software stack is to focus on high-level language support



Driver Stack

HSA Software Stack

Apps

Domain Libraries

OpenCL™, DX Runtimes, User Mode Drivers

Graphics Kernel Mode Driver

Hardware - APUs, CPUs, GPUs

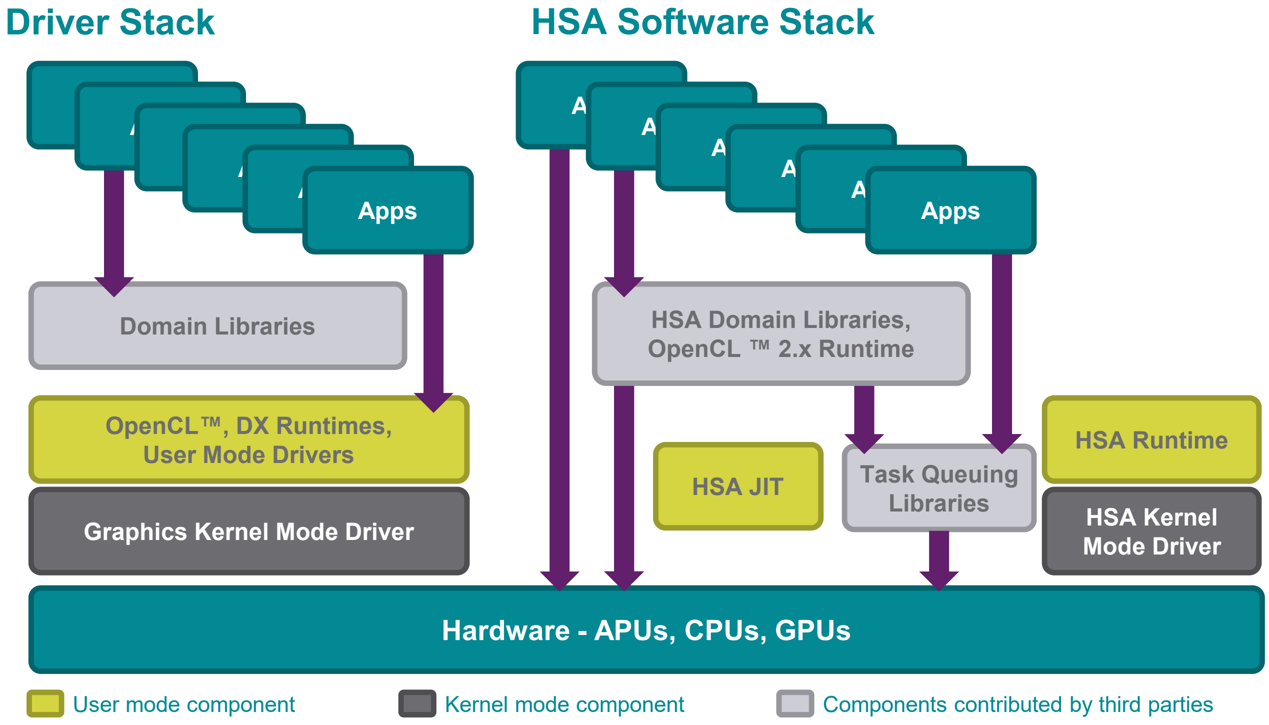
🟨 User mode component

⬛ Kernel mode component

⬜ Components contributed by third parties

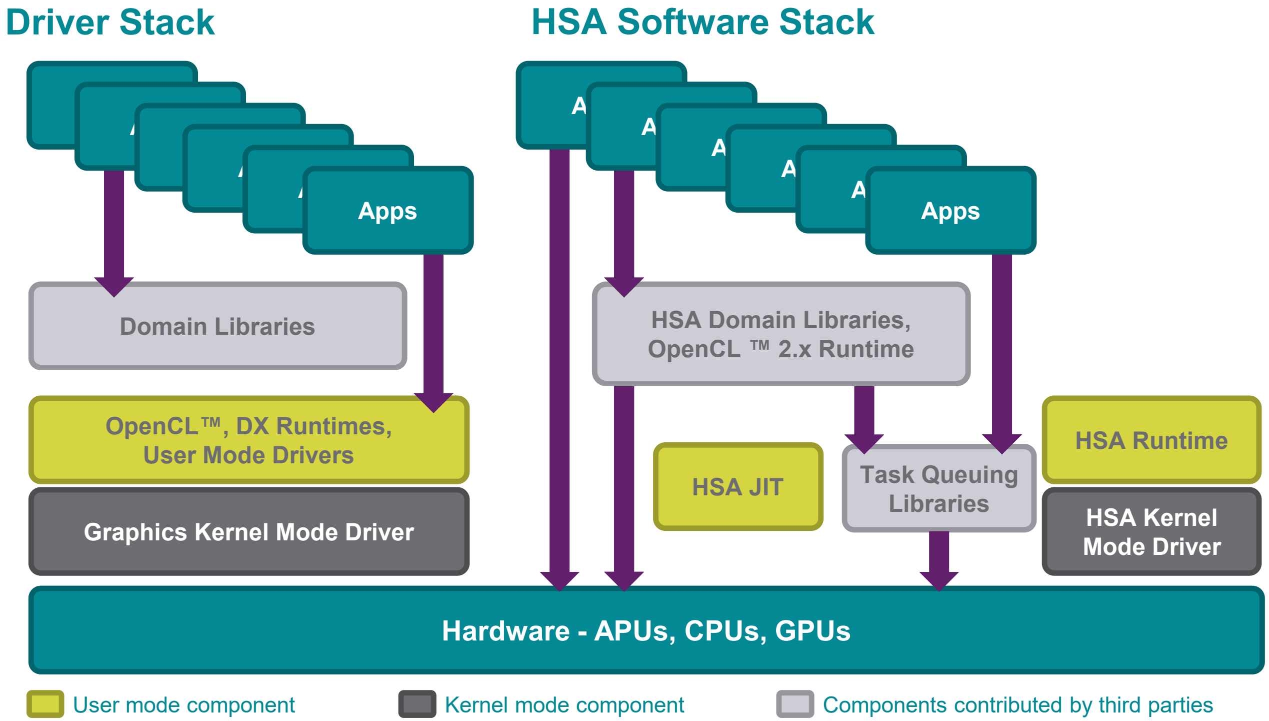
EVOLUTION OF THE SOFTWARE STACK – A COMPARISON

- Goal of the software stack is to focus on high-level language support
  - Allow to target the GPU directly by SW


Driver Stack
HSA Software Stack
Apps
Apps
Domain Libraries
OpenCL™, DX Runtimes, User Mode Drivers
Graphics Kernel Mode Driver
HSA Domain Libraries, OpenCL ™ 2.x Runtime
HSA JIT
Task Queuing Libraries
HSA Runtime
HSA Kernel Mode Driver
Hardware - APUs, CPUs, GPUs
User mode component
Kernel mode component
Components contributed by third parties

EVOLUTION OF THE SOFTWARE STACK – A COMPARISON

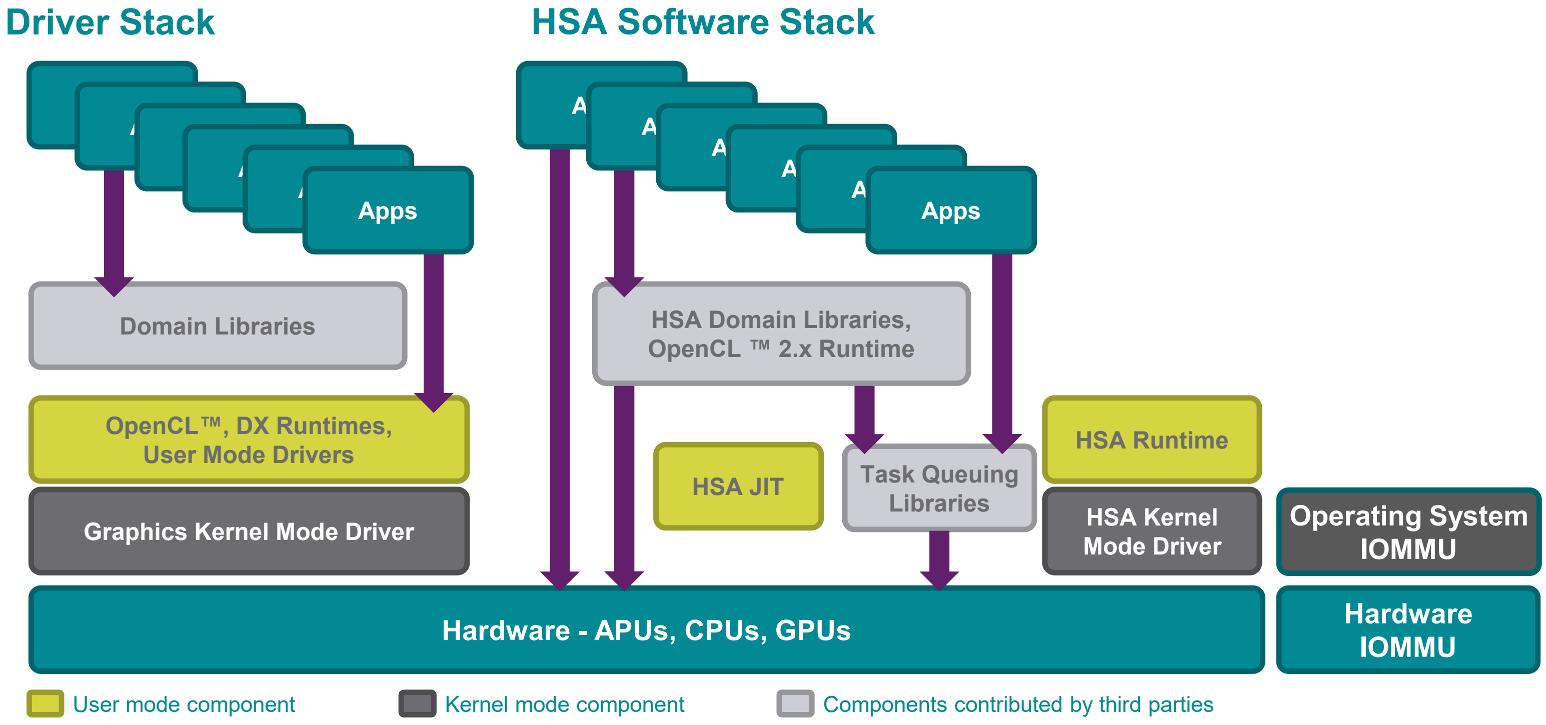
- Goal of the software stack is to focus on high-level language support
  - Allow to target the GPU directly by SW
  - Drivers are setting up the HW and policies, then go out of the way



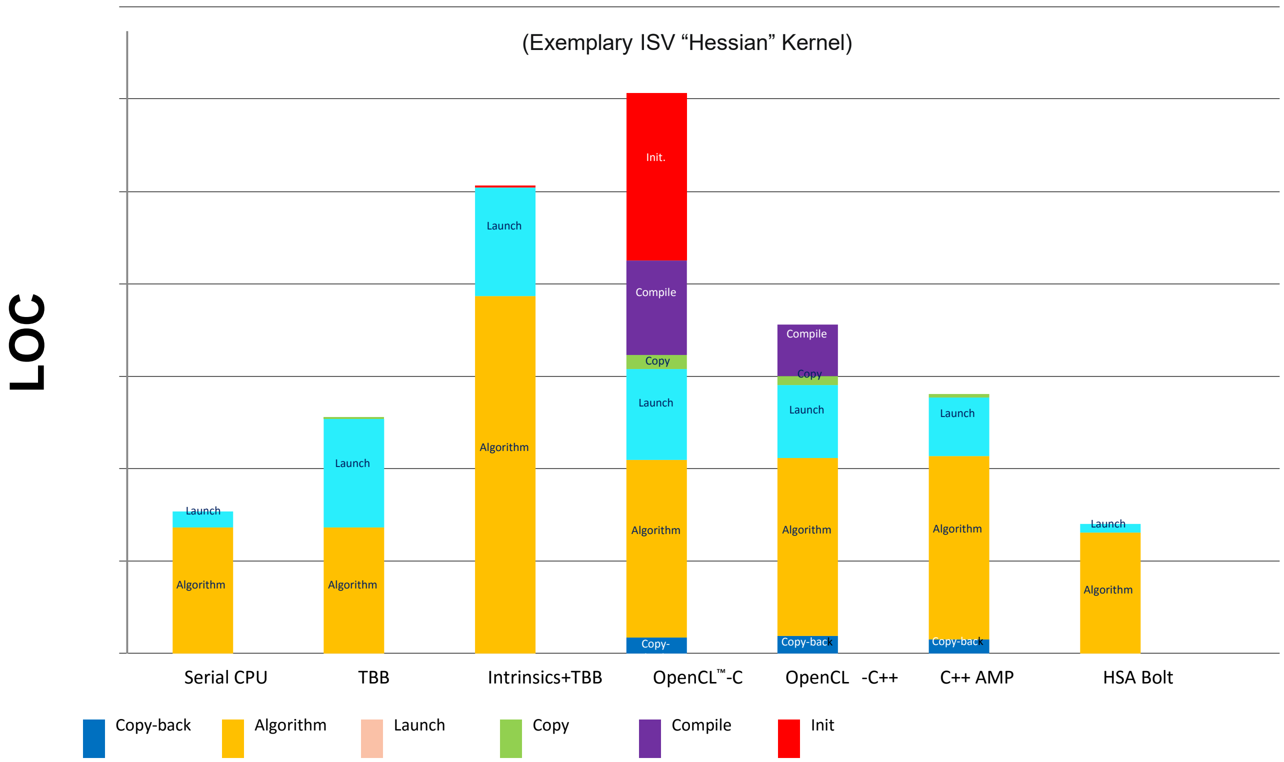
Driver Stack
HSA Software Stack
Apps
Apps
Domain Libraries
OpenCL™, DX Runtimes, User Mode Drivers
Graphics Kernel Mode Driver
HSA Domain Libraries, OpenCL ™ 2.x Runtime
HSA JIT
Task Queuing Libraries
HSA Runtime
HSA Kernel Mode Driver
Hardware - APUs, CPUs, GPUs
User mode component
Kernel mode component
Components contributed by third parties

EVOLUTION OF THE SOFTWARE STACK – A COMPARISON

- Goal of the software stack is to focus on high-level language support
  - Allow to target the GPU directly by SW
  - Drivers are setting up the HW and policies, then go out of the way
  - IOMMU support provide hardware enforced protections for Operating System

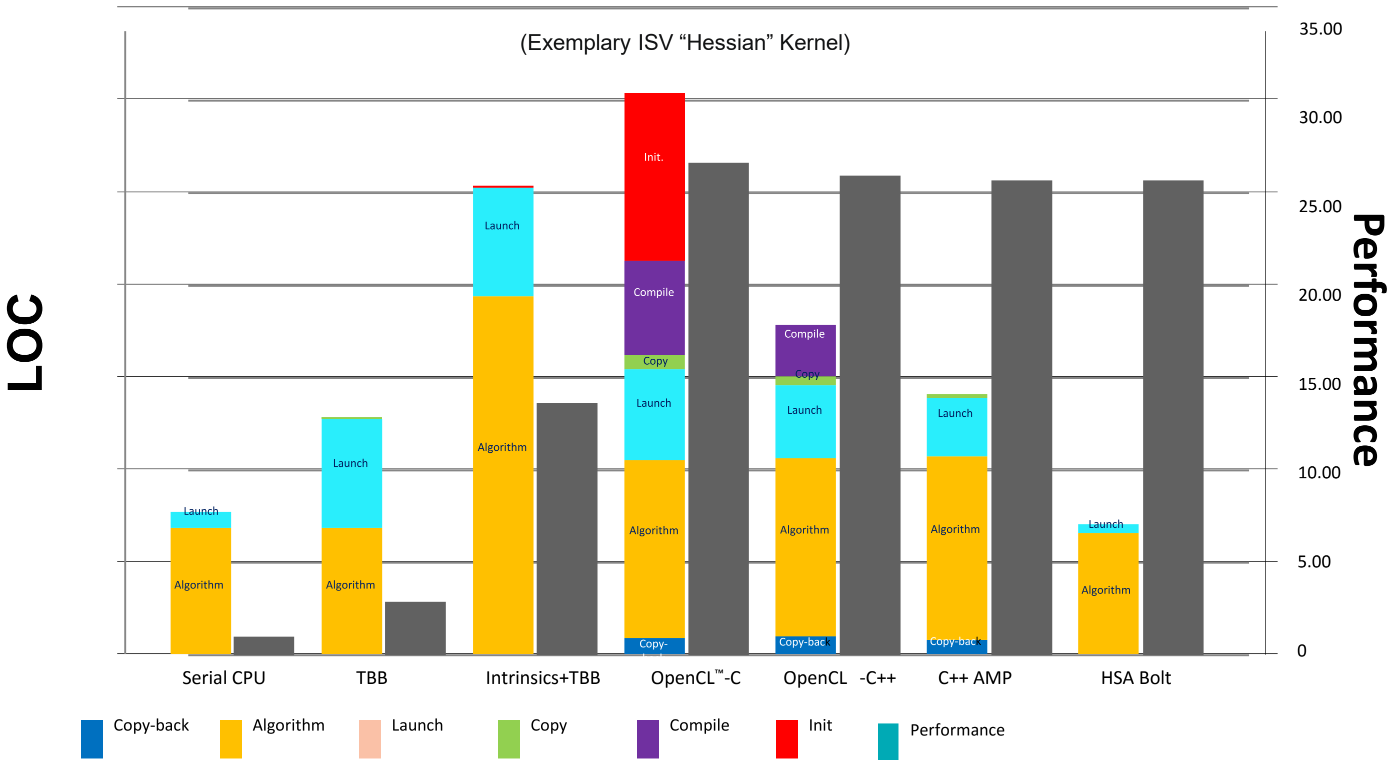


LINES-OF-CODE AND PERFORMANCE COMPARISONS



AMD A10-5800K APU with Radeon™ HD Graphics – CPU: 4 cores, 3800MHz (4200MHz Turbo); GPU: AMD Radeon HD 7660D, 6 compute units, 800MHz; 4GB RAM.
Software – Windows 7 Professional SP1 (64-bit OS); AMD OpenCL™ 1.2 AMD-APP (937.2); Microsoft Visual Studio 11 Beta

LINES-OF-CODE AND PERFORMANCE COMPARISONS



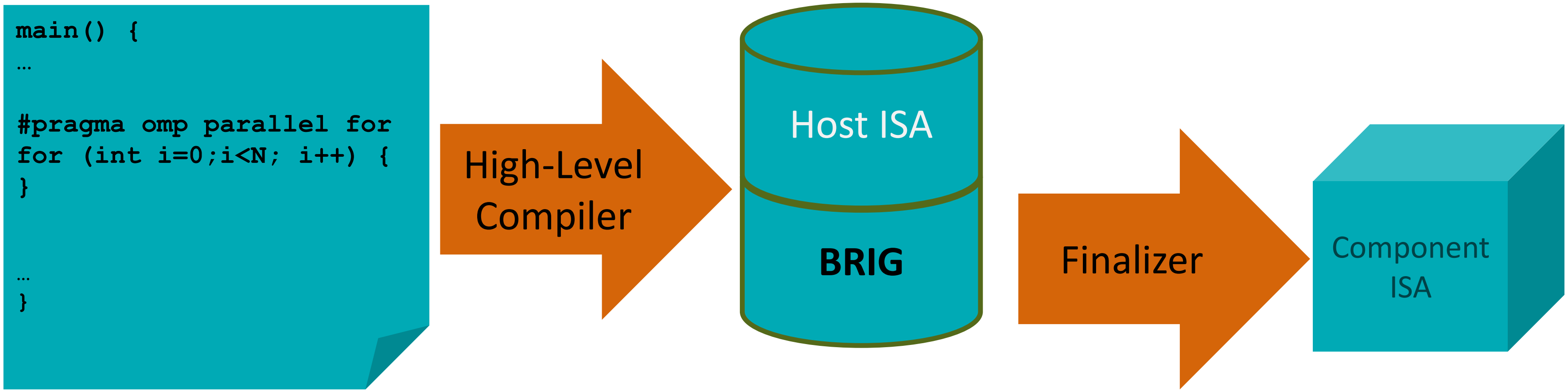
AMD A10-5800K APU with Radeon™ HD Graphics – CPU: 4 cores, 3800MHz (4200MHz Turbo); GPU: AMD Radeon HD 7660D, 6 compute units, 800MHz; 4GB RAM.
Software – Windows 7 Professional SP1 (64-bit OS); AMD OpenCL™ 1.2 AMD-APP (937.2); Microsoft Visual Studio 11 Beta

ACCELERATORS: THE PORTABILITY CHALLENGE

- CPU ISAs
  - ISA innovations added incrementally (i.e., NEON, AVX, etc)
    - ISA retains backwards-compatibility with previous generation
  - Two dominant instruction-set architectures: ARM and x86
- GPU ISAs
  - Massive diversity of architectures in the market
    - Each vendor has its own ISA - and often several in the market at same time
  - No commitment (or attempt!) to provide any backwards compatibility
    - Traditionally graphics APIs (OpenGL, DirectX) provide necessary abstraction

WHAT IS HSA INTERMEDIATE LANGUAGE (HSAIL)?

- Intermediate language for parallel compute in HSA
  - Generated by a “High Level Compiler” (GCC, LLVM, Java VM, etc.)
  - Expresses parallel regions of code
  - Binary format of HSAIL is called “BRIG”
  - *Goal: Bring parallel acceleration to mainstream programming languages*
- IOMMU based pointer translation is key to enabling an efficient IL Implementation



```
main() {
…

#pragma omp parallel for
for (int i=0;i<N; i++) {
}


…
}
```

MEMBERS DRIVING HAS FOUNDATION

http://www.hsafoundation.com/

Founders

AMD, ARM, Imagination, MEDIATEK, QUALCOMM, SAMSUNG

Promoters

LG Electronics, GPT HX

Supporters

allinea, Argonne National Laboratory, UChicago Argonne LLC, Arteris, codeplay, FABRICENGINE, KISHONTI, Lawrence Livermore National Laboratory, Linaro, MULTICOREWARE, Oak Ridge National Laboratory, Sandia National Laboratories, Stream Computing Performance Engineers, SUSE

Contributors

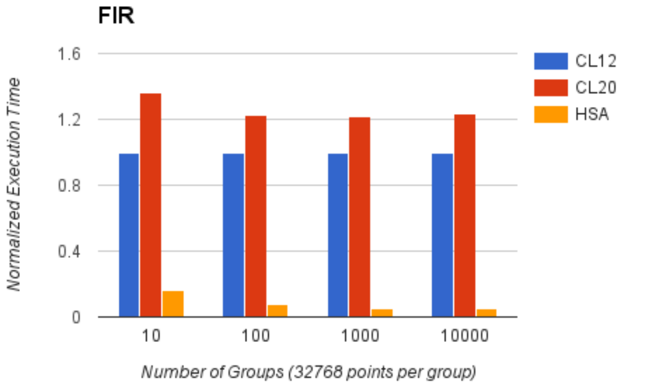
ANALOG DEVICES, CANONICAL, CEVA, DMP, ETRI 한국전자통신연구원, HUAWEI, ITRI Industrial Technology Research Institute, LUXOFT, MARVELL, mobica, SONICS, SONY make.believe, Swarm64, SYNOPSYS, tensilica, Texas Instruments, TOSHIBA Leading Innovation, VIA

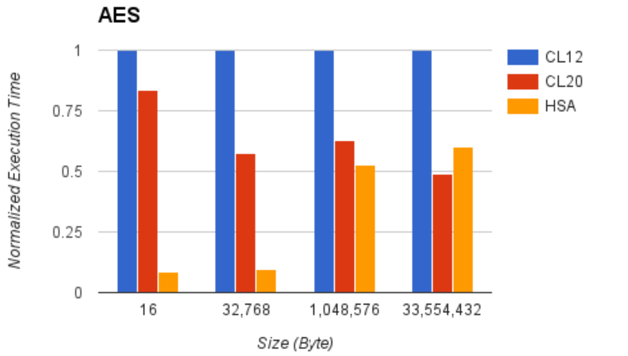
Academic

icsa, NTHU Programming Language Lab, NTHU System Software Lab, NMAMIT, Northeastern, RICE Unconventional Wisdom, 清華大學 Tsinghua University, Tampere University of Technology, T.E.I. of Crete, The University of Mississippi, UNT University of North Texas, Alma Mater Studiorum Università di Bologna, University of Bristol, The University of Edinburgh informatics, Illinois Computer Science, UWE Bristol

GEN1: FIR & AES

- FIR is a memory-intensive streaming workload
- AES is a compute-intensive streaming workload
- CL12 – cl_mem buffer
  - Copy to/from the device
- CL20 – SVM buffer – Coarse Grain Sync
  - Copy to/from SVM
  - Data copy cannot be avoided, since the space for SVM is limited
- HSA – Unified Memory Space – Fine Grained Sync
  - Regular pointer
  - No explicit copy
- Results
  - HSA compute abstraction
  - NO performance penalty
- Not all algorithms run faster
  - Measured on Kaveri (A pre-HSA 1.0 device)
  - Limited Coherent throughput

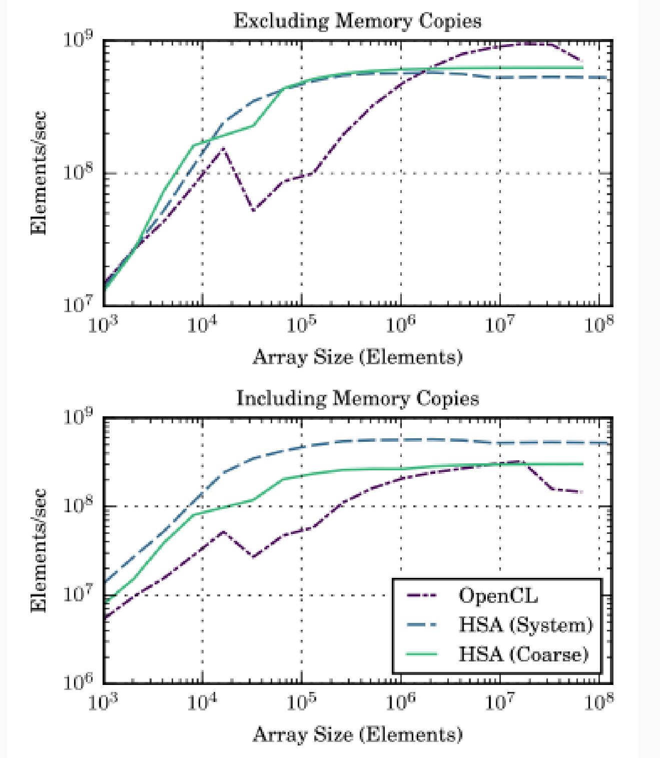




Saoni Mukherjee, Yifan Sun, Paul Blinzer, Amir Kavyan Ziabari, David Kaeli,*A Comprehensive Performance Analysis of HSA and OpenCL 2.0,* **Proceedings of the 2016 International Symposium on Program Analysis and System Software,** April 2016, to appear.

BLACKSCHOLES

- C++ on HSA
  - Matches or outperforms OpenCL
- Course Grained SVM
  - Matches OpenCL buffers for bandwidth
  - More predictable performance
- Fine Grained SVM
  - Faster kernel dispatch
  - Larger allocations
  - Shared data structure
- Results
  - HSA compute abstraction
  - NO performance penalty



SOURCE: RALPH POTTER – CODEPLAY. PRESENTATION MADE TO SG14 C++ WORKGROUP

ENABLING HETEROGENEOUS COMPUTING

SUMMARY AND DEMONSTRATION

- **Key Takeaways:**
  - To further scale up compute performance, software must take better advantage of system accelerators like GPUs and DSPs in high level languages
  - Accelerators following the HSA Foundation specification requirements allow programmers to write or port programs easily using common high level languages
  - AMD IOMMU is key to efficiently and safely access process virtual memory!
    - Does translation of both process address space via PASID and device physical accesses
    - Enforces OS allocation policy, deals with virtual memory page faults, and much more

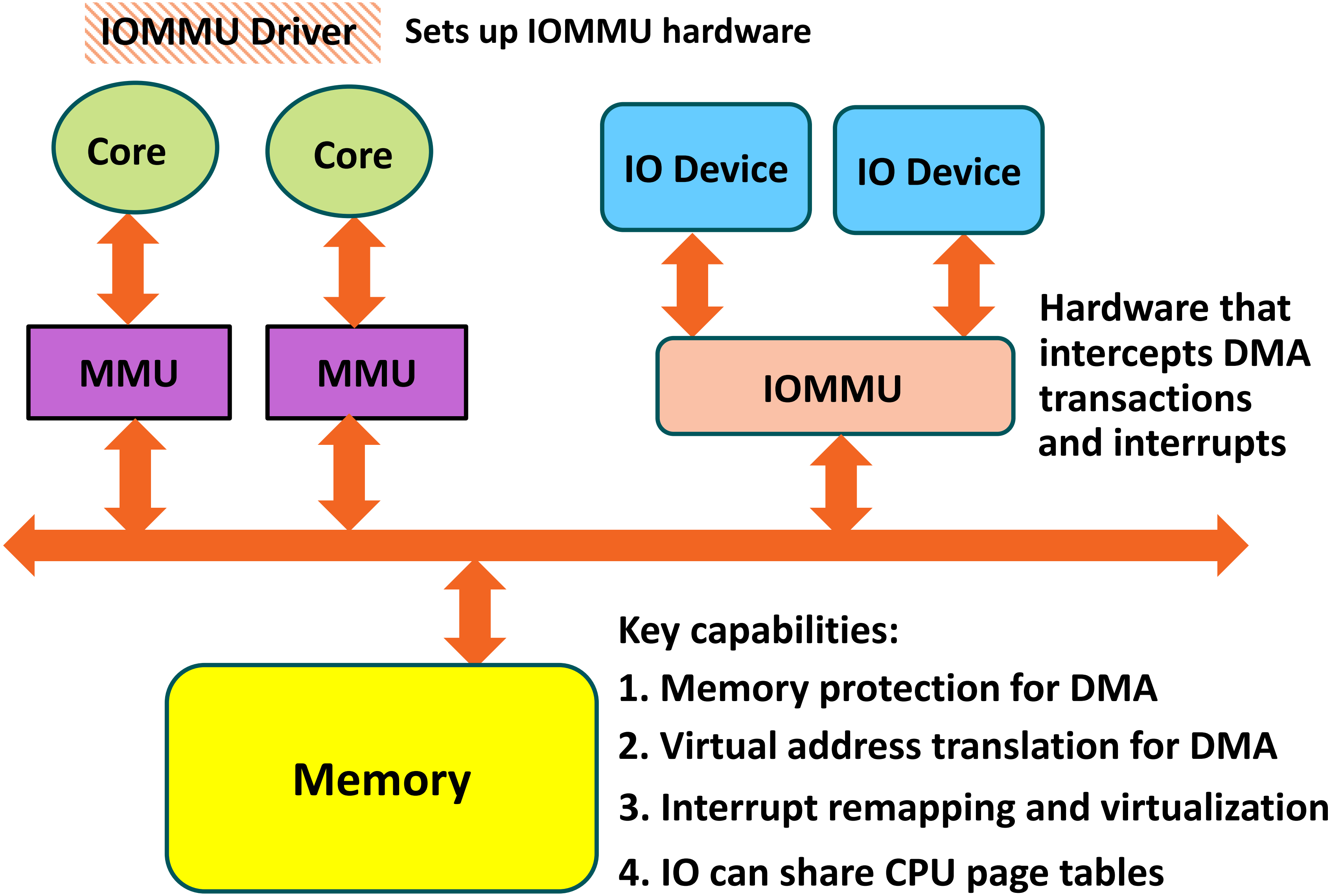
AGENDA

AMD

INTERNALS How does IOMMU work?

RESEARCH Research Opportunities and Tools

RECAP: IOMMU AND ITS CAPABILITIES



Key capabilities:

1. Memory protection for DMA
2. Virtual address translation for DMA
3. Interrupt remapping and virtualization
4. IO can share CPU page tables

AGENDA: WHAT IS COMING UP?

- DMA Address Translation
  - Address translation and memory protection in un-virtualized System
  - Making address translation faster through caching
  - Enabling shared address space in heterogeneous system
  - Enabling pre-translation through IOMMU
  - Enabling demand paging from devices (dynamic page fault)
  - Nested address translation in virtualized system
  - Invalidating IOMMU mappings

Address translation, memory protection, HSA

AGENDA: WHAT IS COMING UP?

- DMA Address Translation
  - Address translation and memory protection in un-virtualized System
  - Making address translation faster through caching
  - Enabling shared address space in heterogeneous system
  - Enabling pre-translation through IOMMU
  - Enabling demand paging from devices (dynamic page fault)
  - Nested address translation in virtualized system
  - Invalidating IOMMU mappings
- Interrupt Handling
  - Interrupt filtering and remapping
  - Interrupt virtualization

Address translation, memory protection, HSA

Interrupts

AGENDA: WHAT IS COMING UP?

- DMA Address Translation
  - Address translation and memory protection in un-virtualized System
  - Making address translation faster through caching
  - Enabling shared address space in heterogeneous system
  - Enabling pre-translation through IOMMU
  - Enabling demand paging from devices (dynamic page fault)
  - Nested address translation in virtualized system
  - Invalidating IOMMU mappings
- Interrupt Handling
  - Interrupt filtering and remapping
  - Interrupt virtualization
- Summary
  - A peek inside a typical IOMMU implementation
  - Data structures and their Interactions

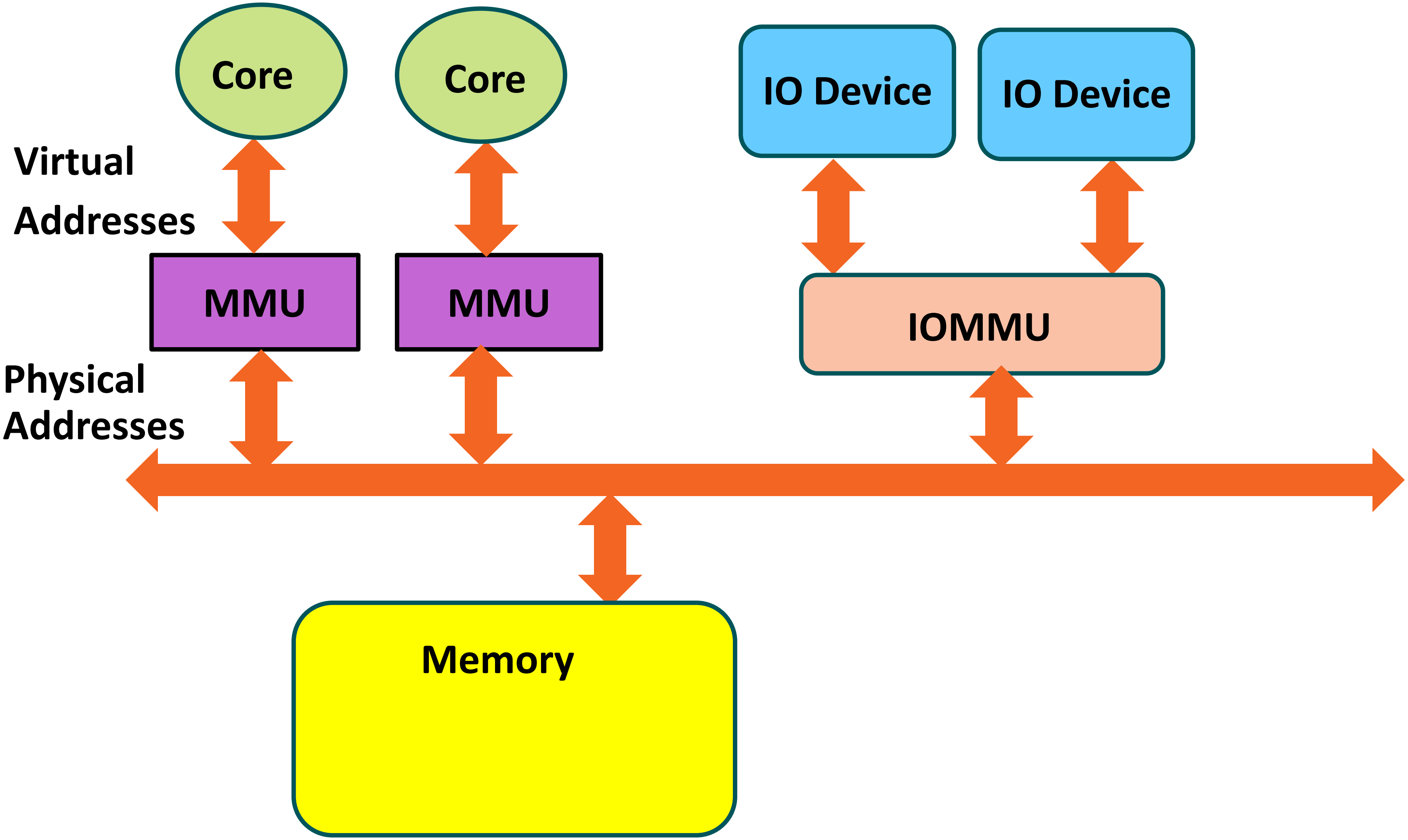
Address translation, memory protection, HSA

Interrupts

IOMMU Internals:
Address Translation and Memory Protection

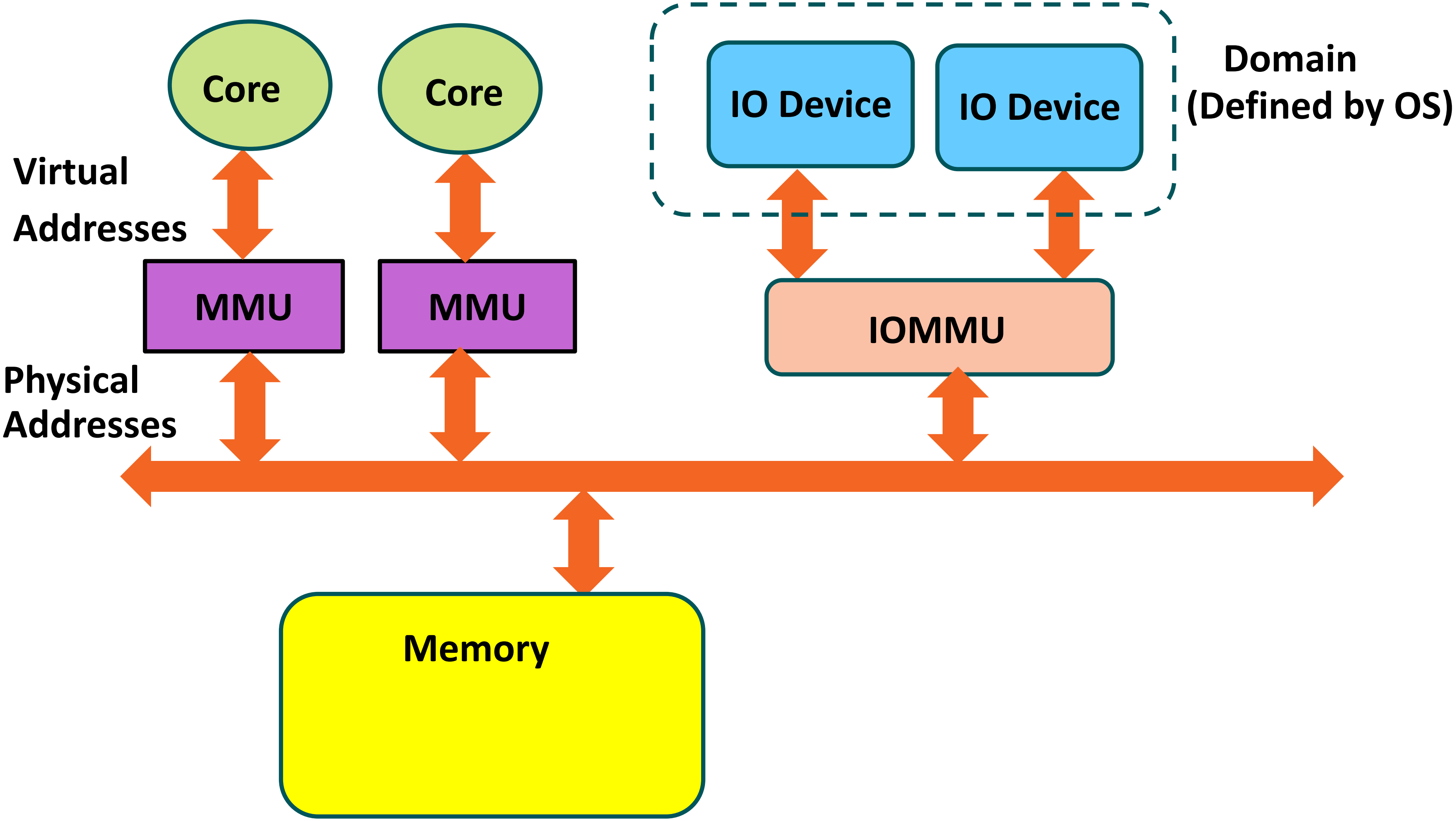
ADDRESS TRANSLATION AND MEMORY PROTECTION

NON-VIRTUALIZED SYSTEM



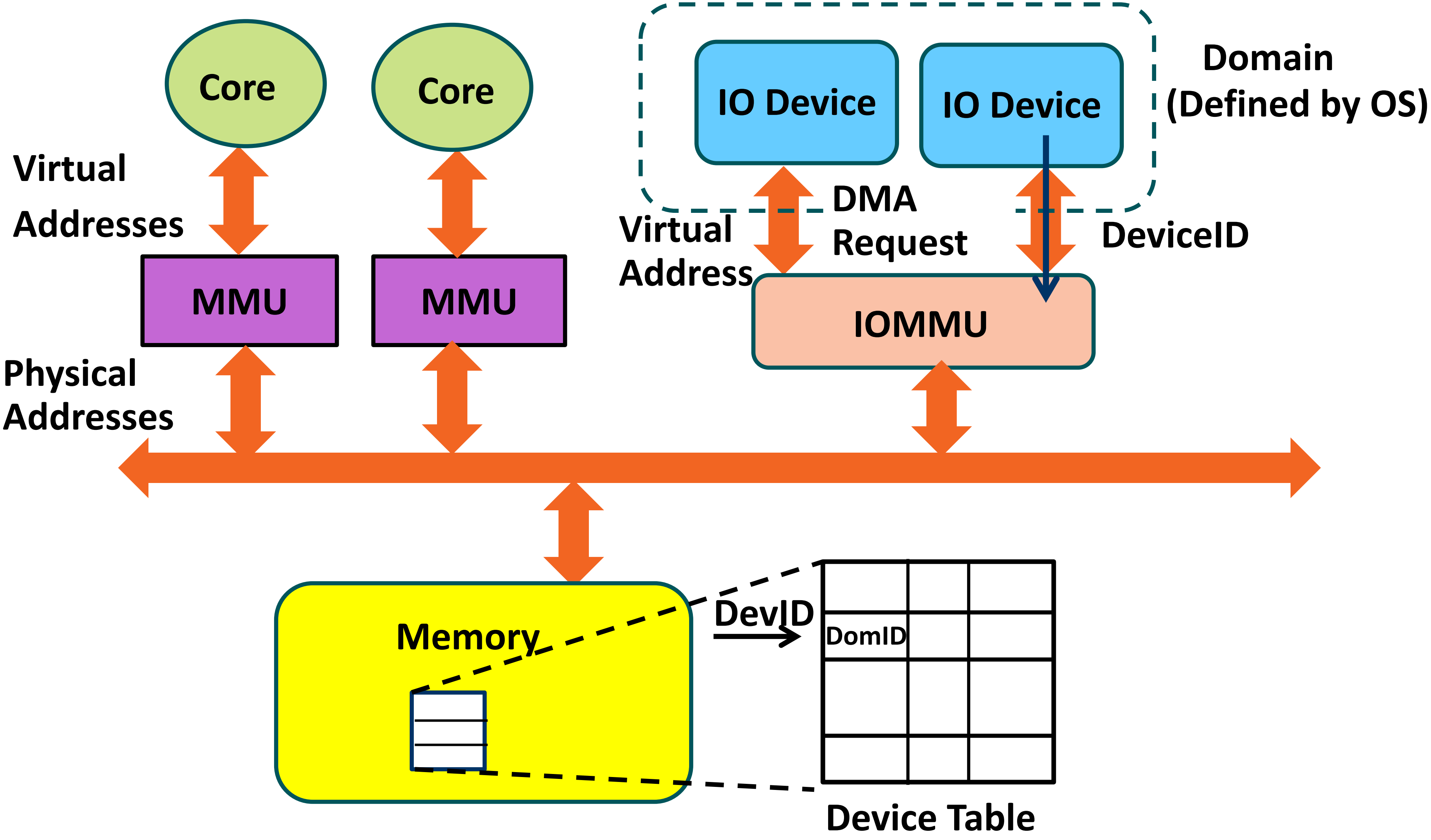
ADDRESS TRANSLATION AND MEMORY PROTECTION

NON-VIRTUALIZED SYSTEM



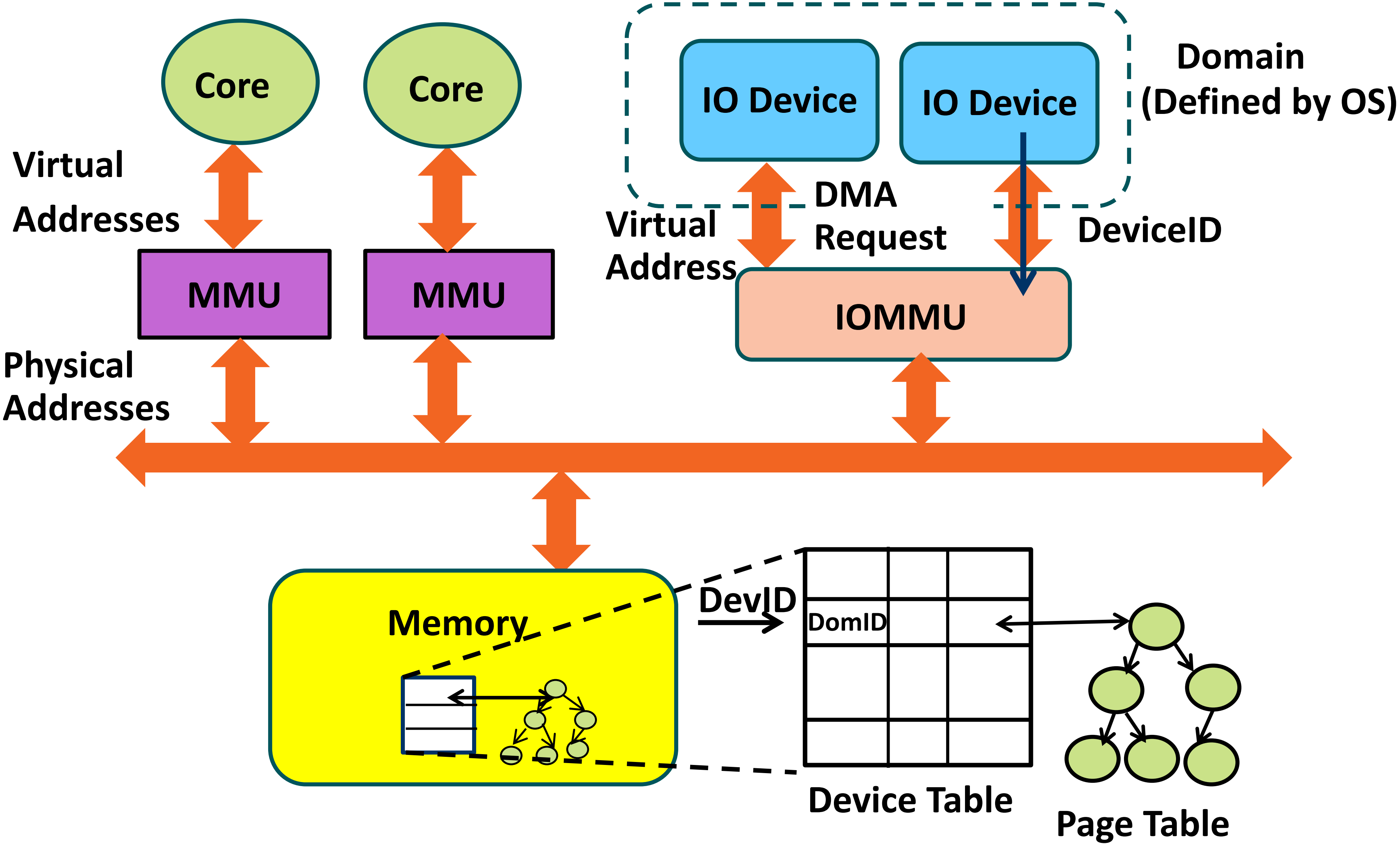
ADDRESS TRANSLATION AND MEMORY PROTECTION

NON-VIRTUALIZED SYSTEM



ADDRESS TRANSLATION AND MEMORY PROTECTION

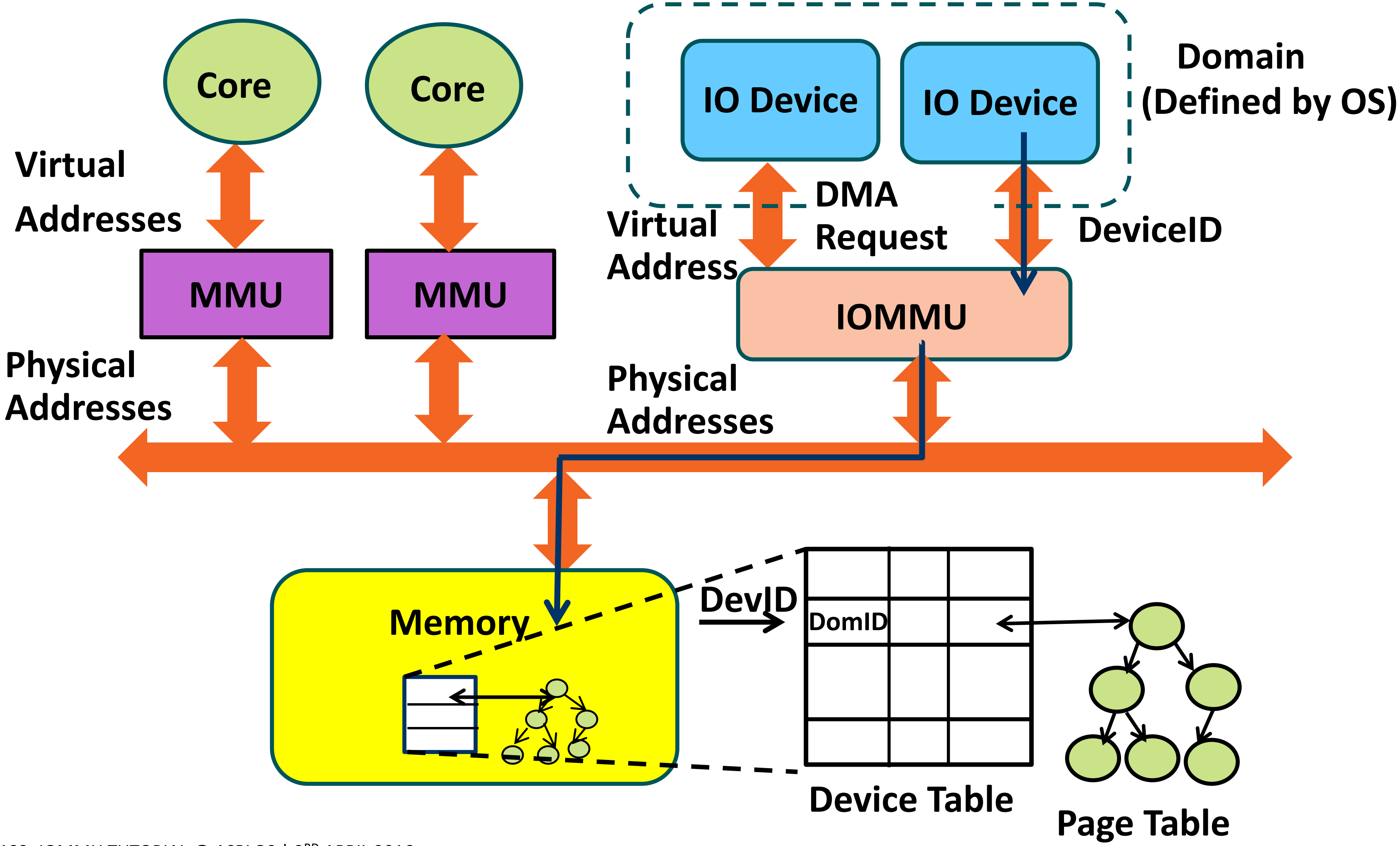
NON-VIRTUALIZED SYSTEM



ADDRESS TRANSLATION AND MEMORY PROTECTION

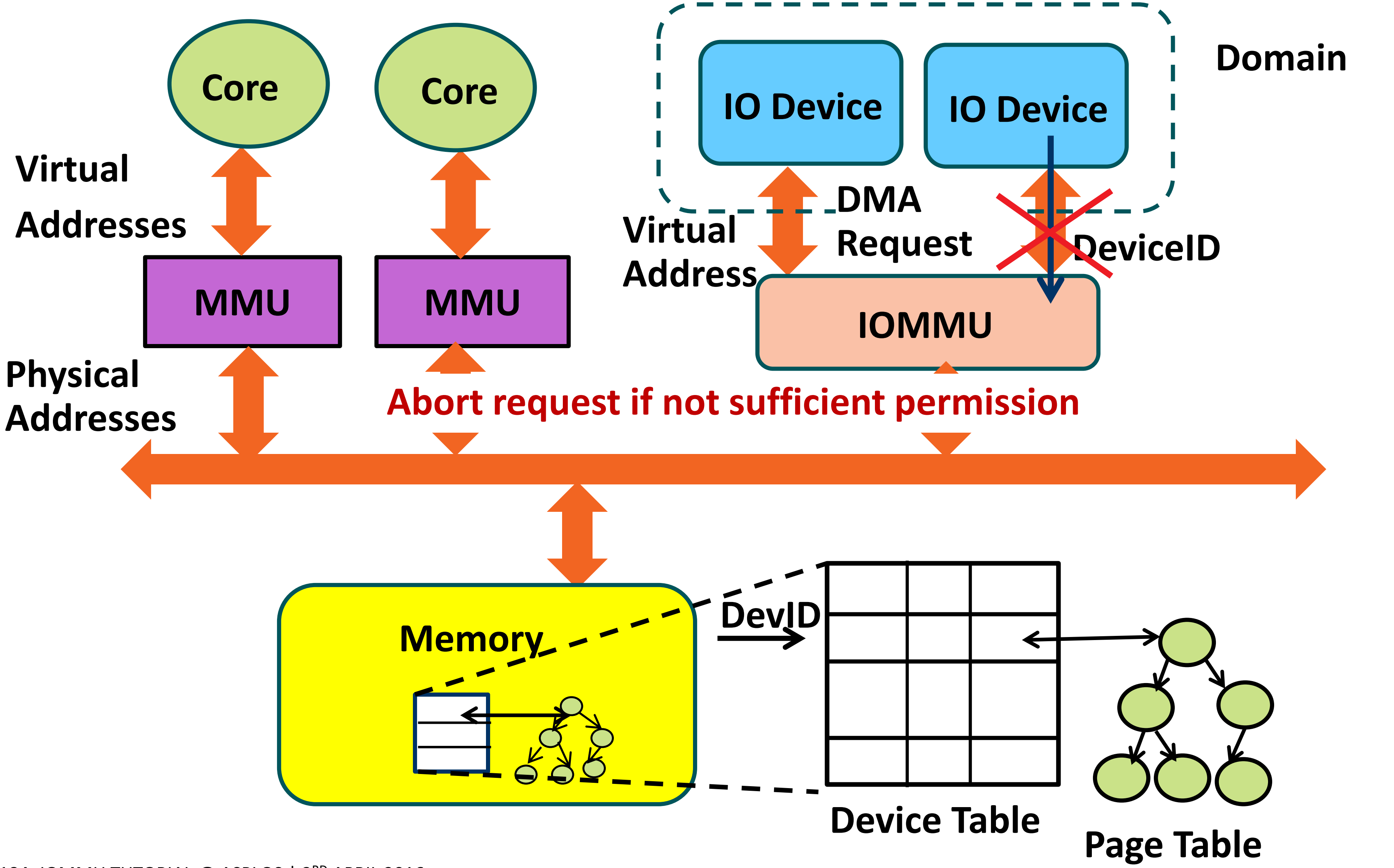
NON-VIRTUALIZED SYSTEM

AMD



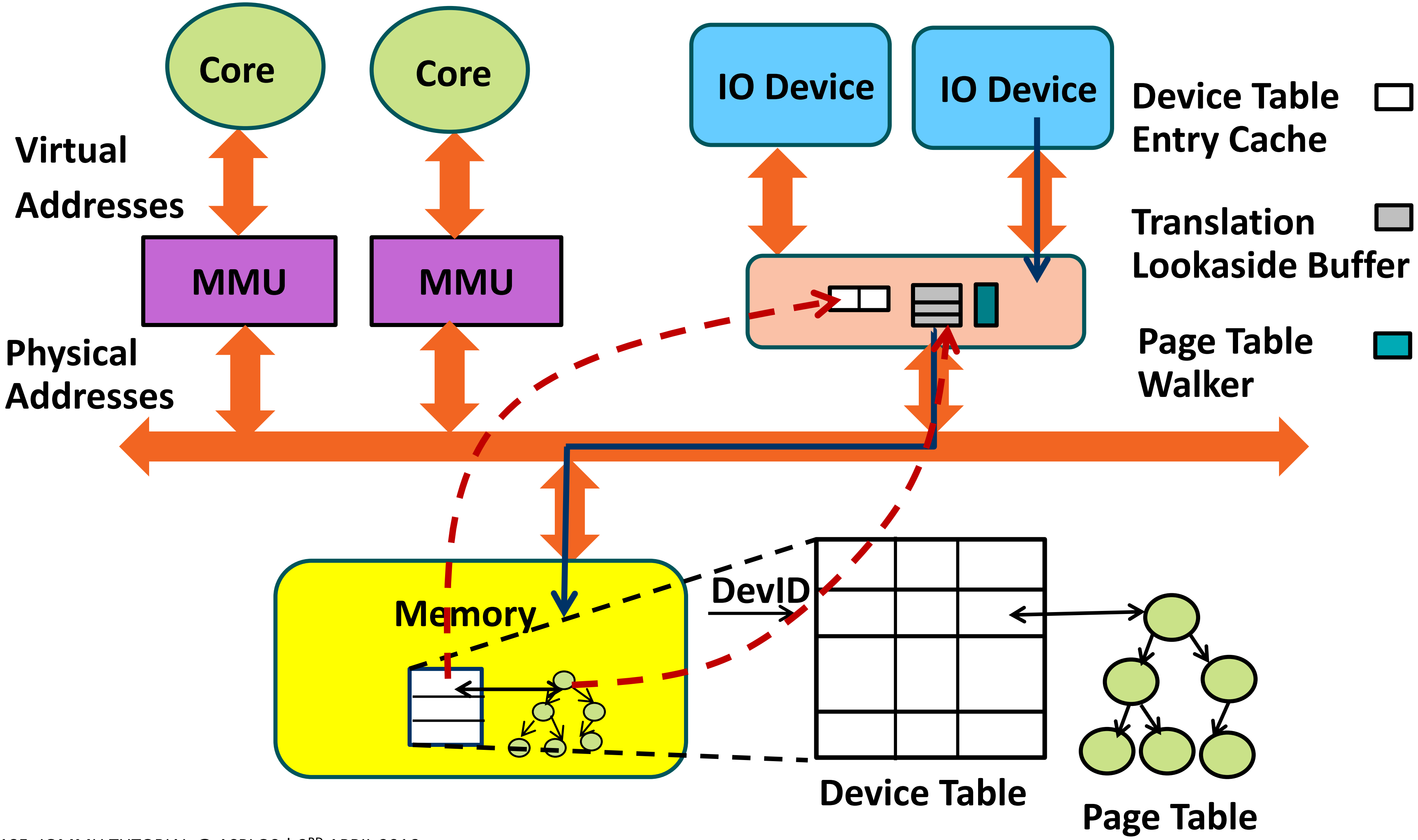
ADDRESS TRANSLATION AND MEMORY PROTECTION

NON-VIRTUALIZED SYSTEM



MAKING TRANSLATION FAST

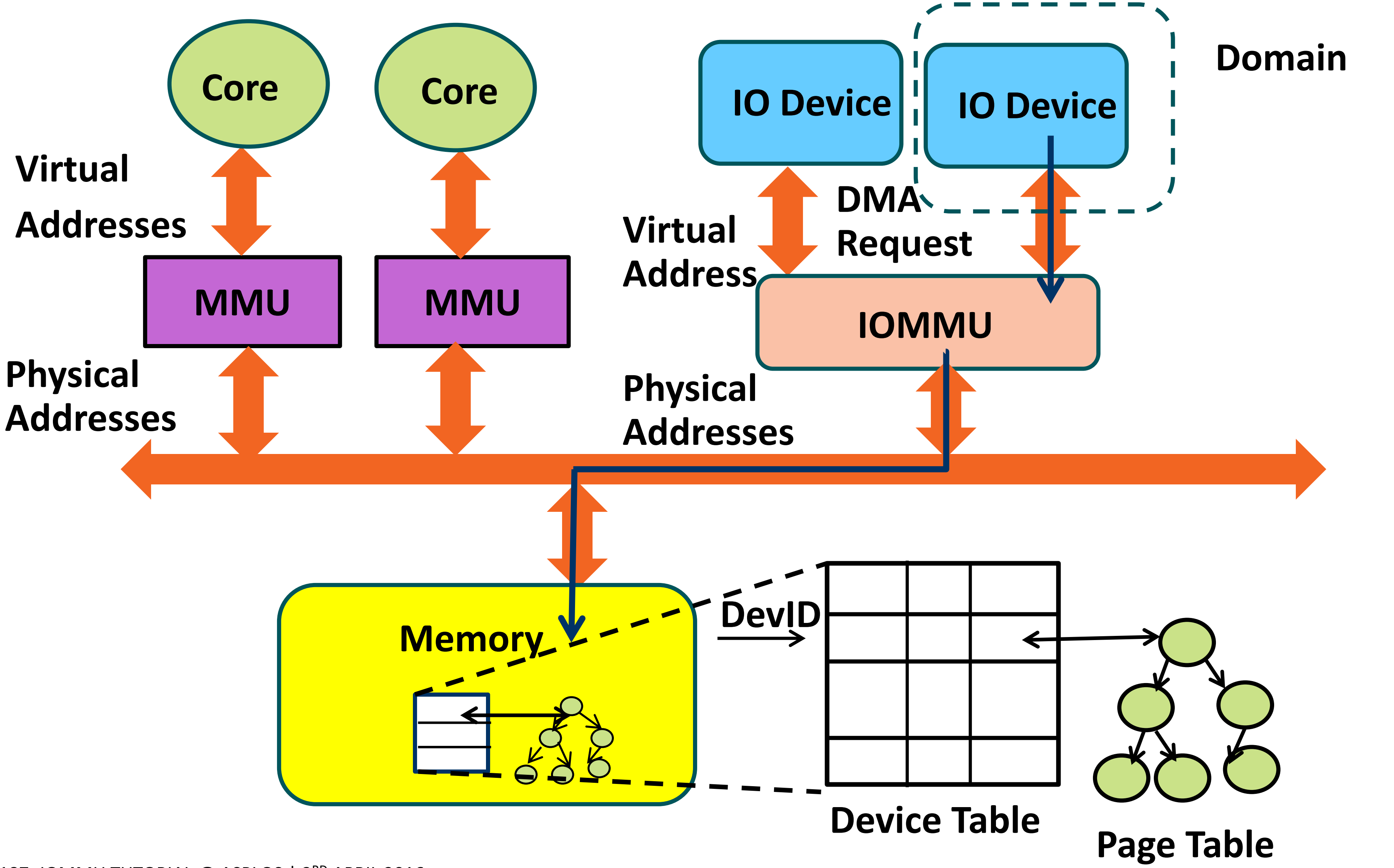
CACHING TRANSLATION IN IOMMU



IOMMU Internals: Enabling “Pointer-is-a-Pointer” in Heterogeneous Systems

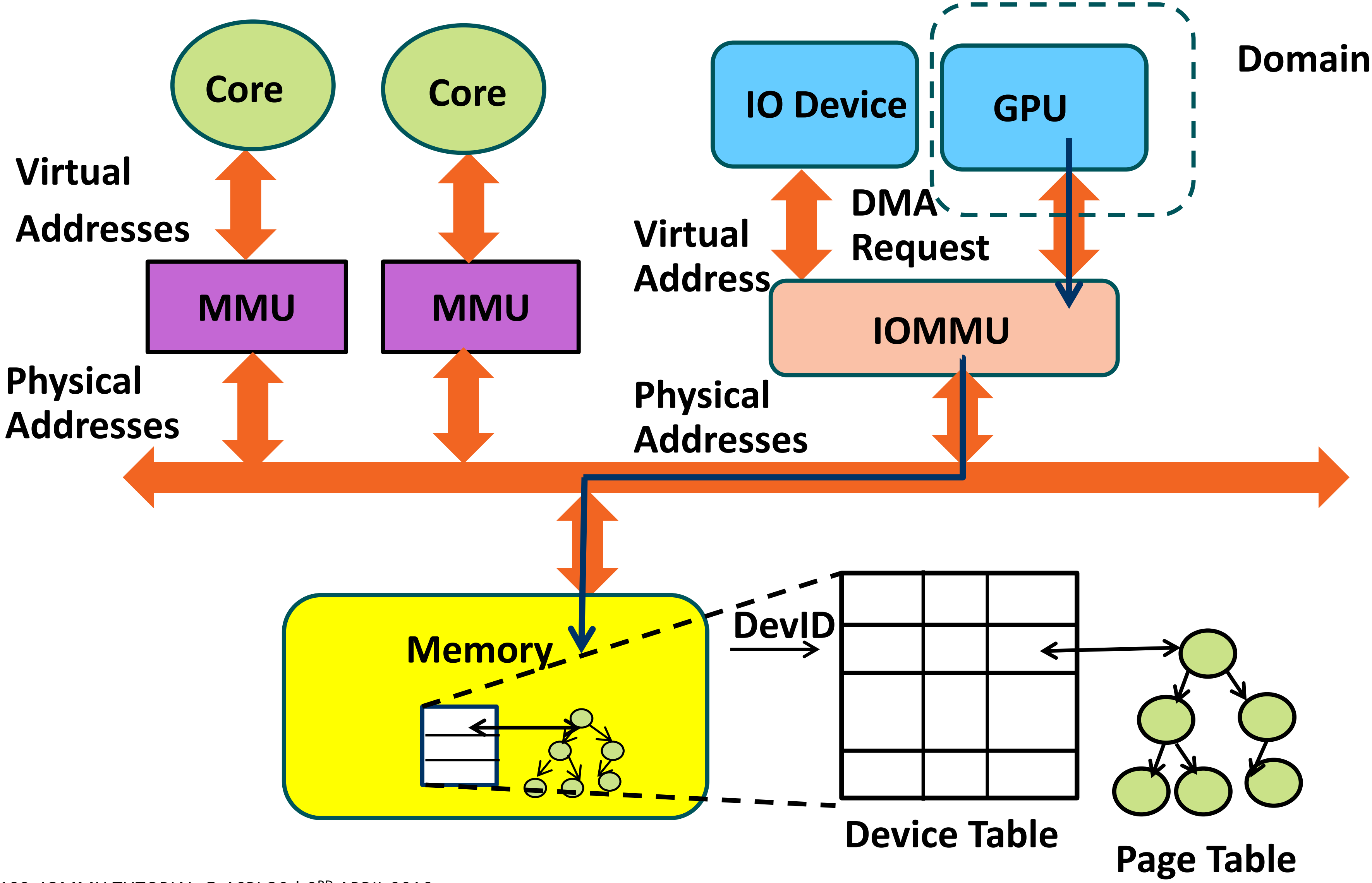
SHARING ADDRESS SPACE WITH CPU

ENABLING POINTER AS POINTER IN HETEROGENEOUS SYSTEMS



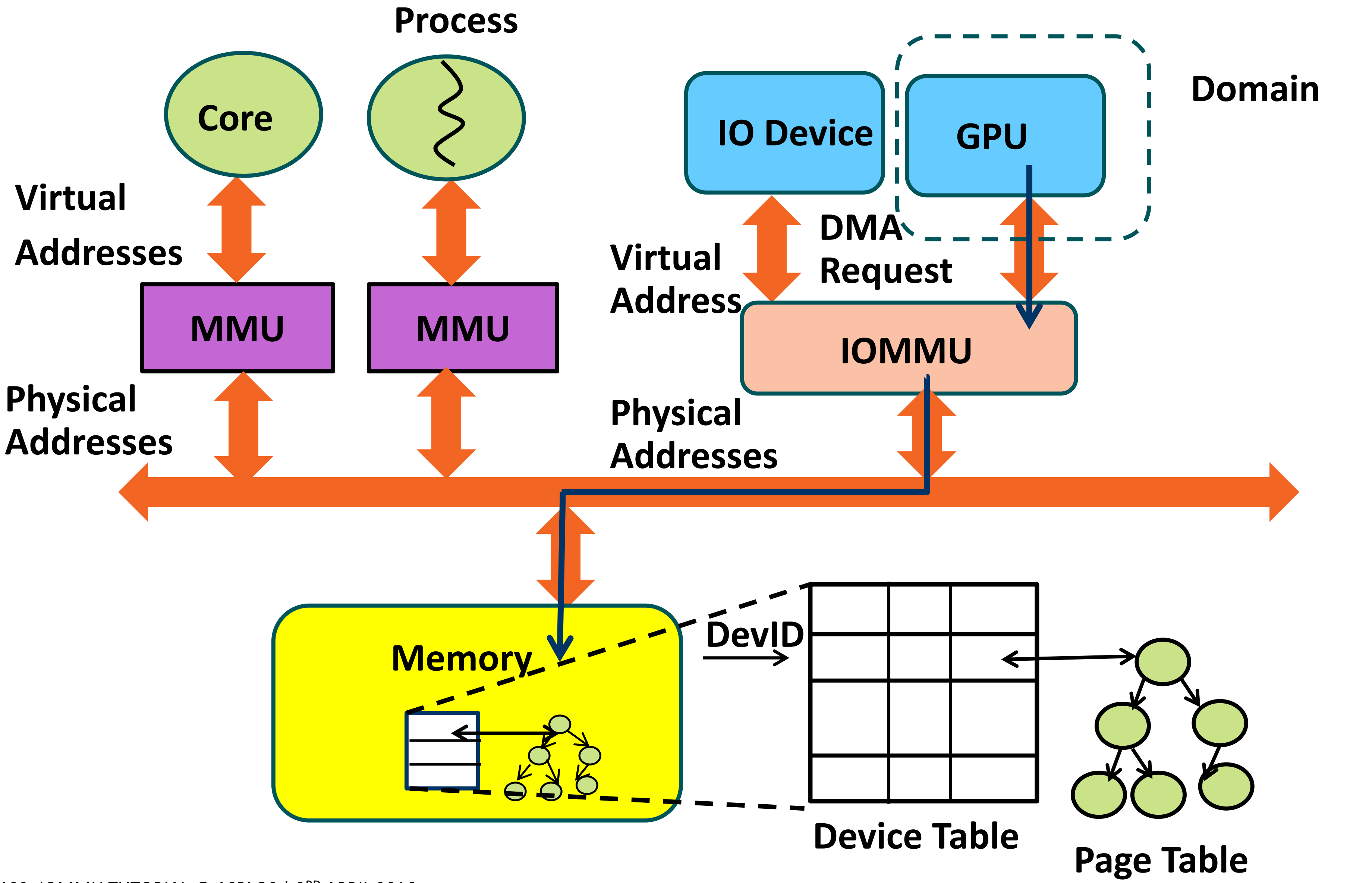
SHARING ADDRESS SPACE WITH CPU

ENABLING POINTER AS POINTER IN HETEROGENEOUS SYSTEMS



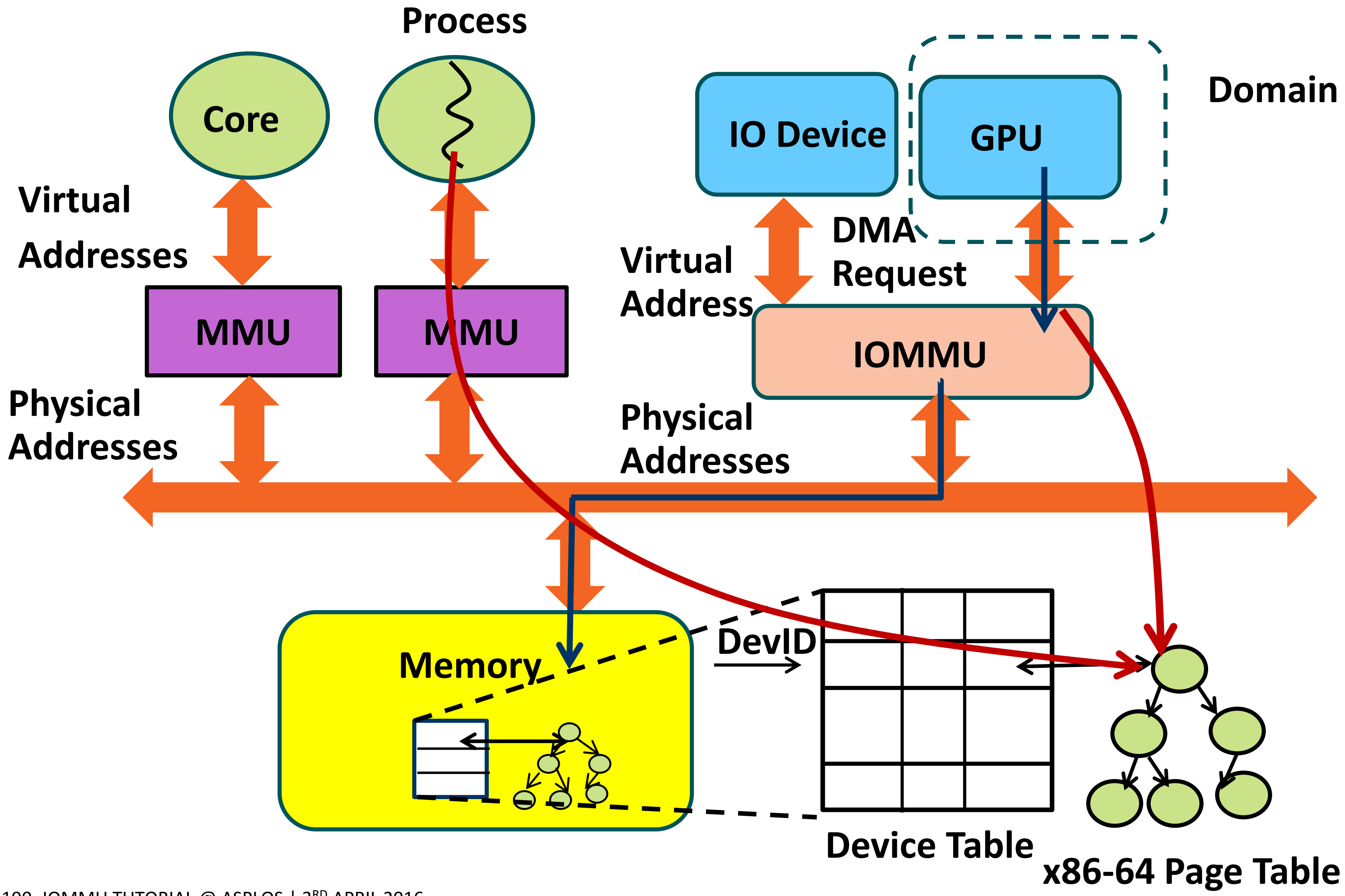
SHARING ADDRESS SPACE WITH CPU

ENABLING POINTER AS POINTER IN HETEROGENEOUS SYSTEMS



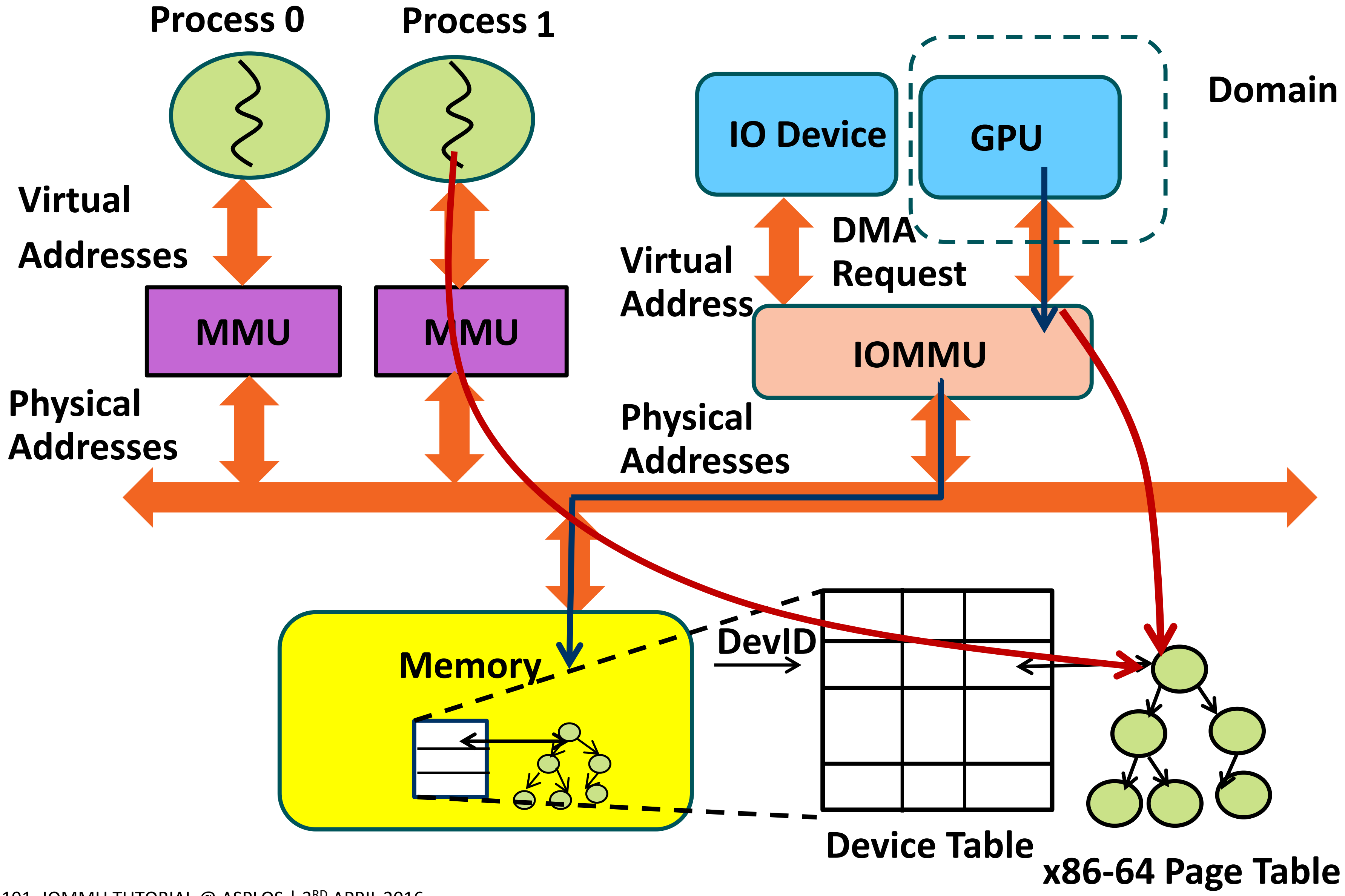
SHARING ADDRESS SPACE WITH CPU

ENABLING POINTER AS POINTER IN HETEROGENEOUS SYSTEMS



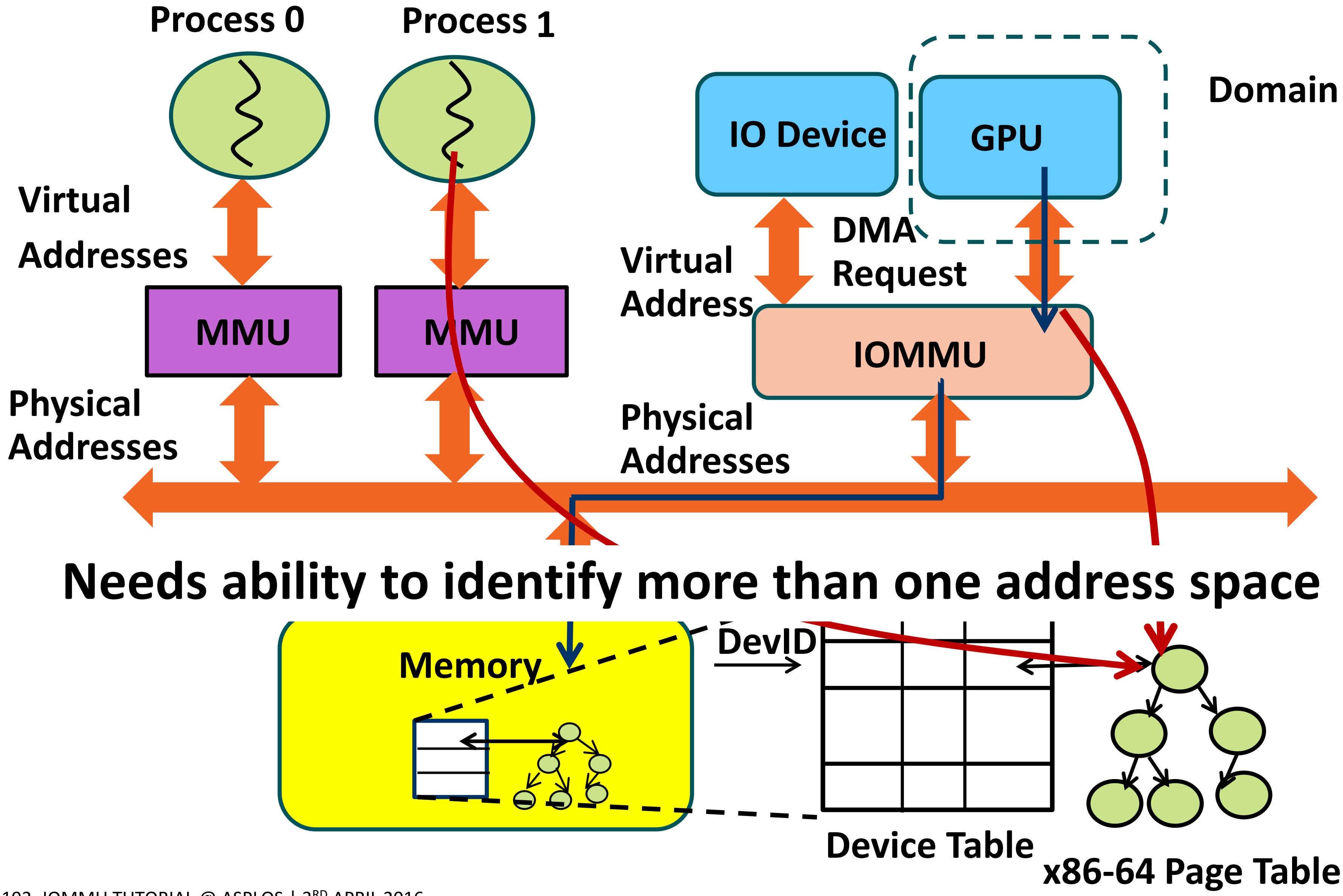
SHARING ADDRESS SPACE WITH CPU

ENABLING POINTER AS POINTER IN HETEROGENEOUS SYSTEMS



SHARING ADDRESS SPACE WITH CPU

ENABLING POINTER AS POINTER IN HETEROGENEOUS SYSTEMS

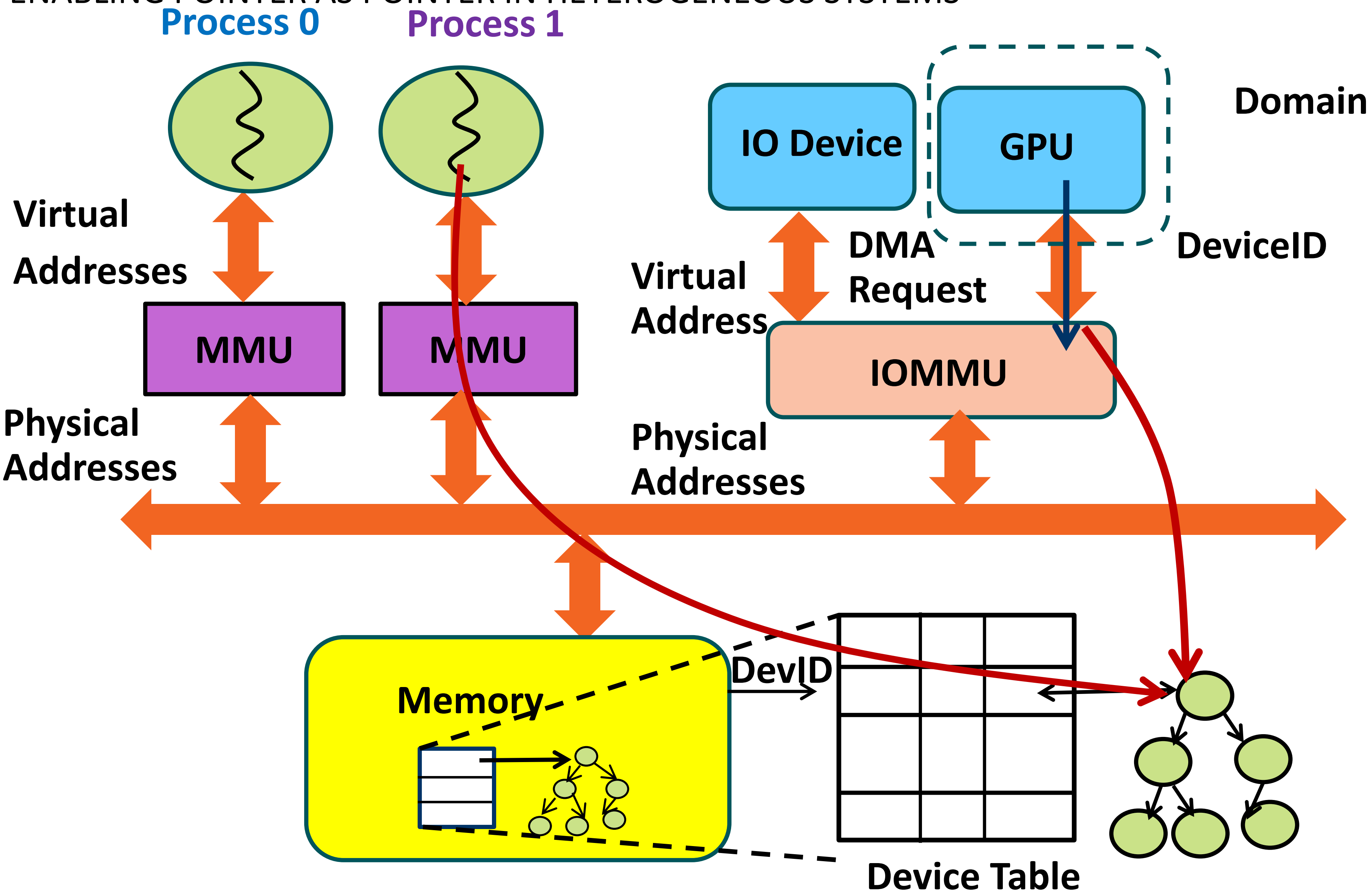


Needs ability to identify more than one address space

SHARING ADDRESS SPACE WITH CPU

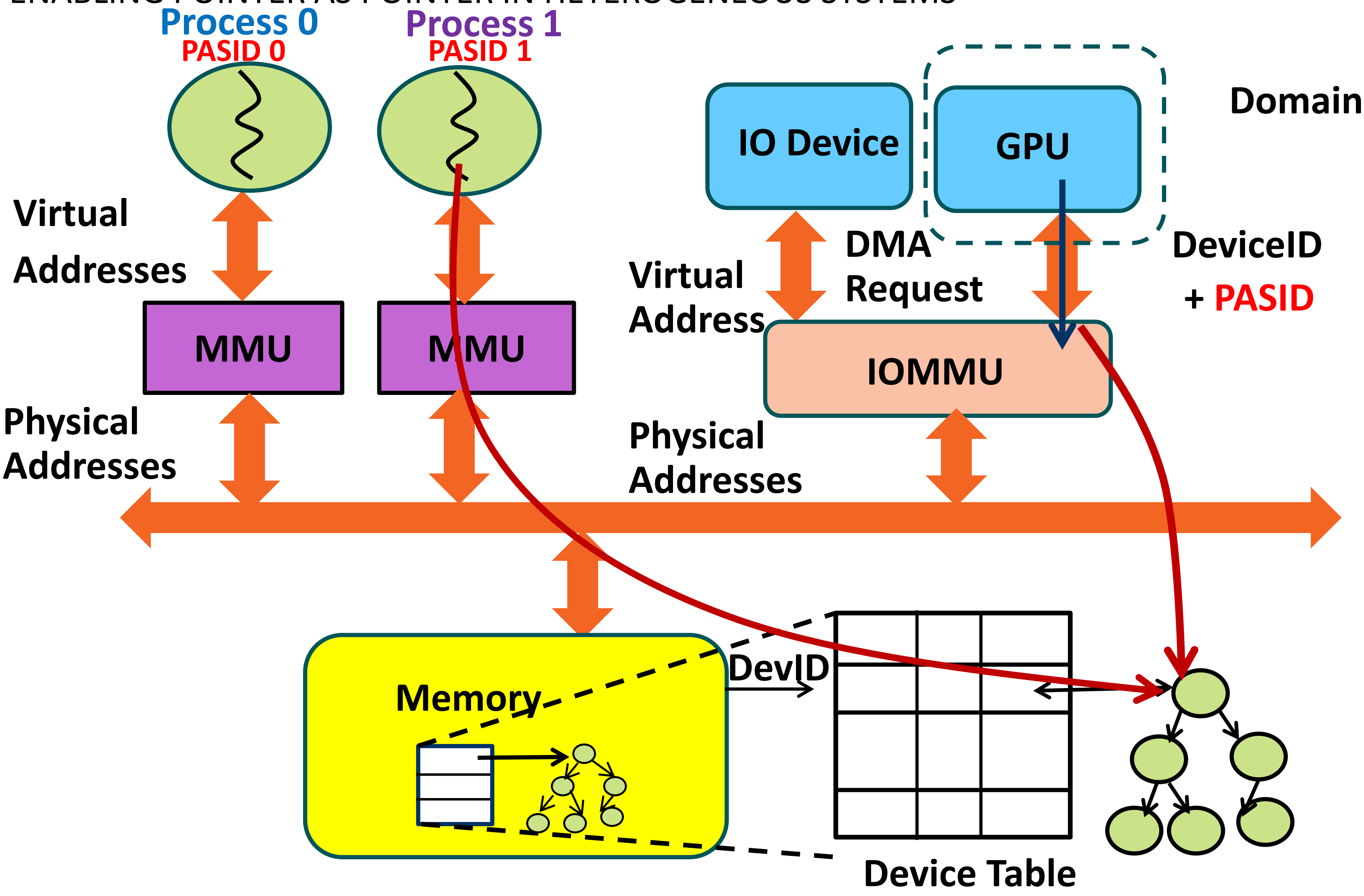
AMD

ENABLING POINTER AS POINTER IN HETEROGENEOUS SYSTEMS



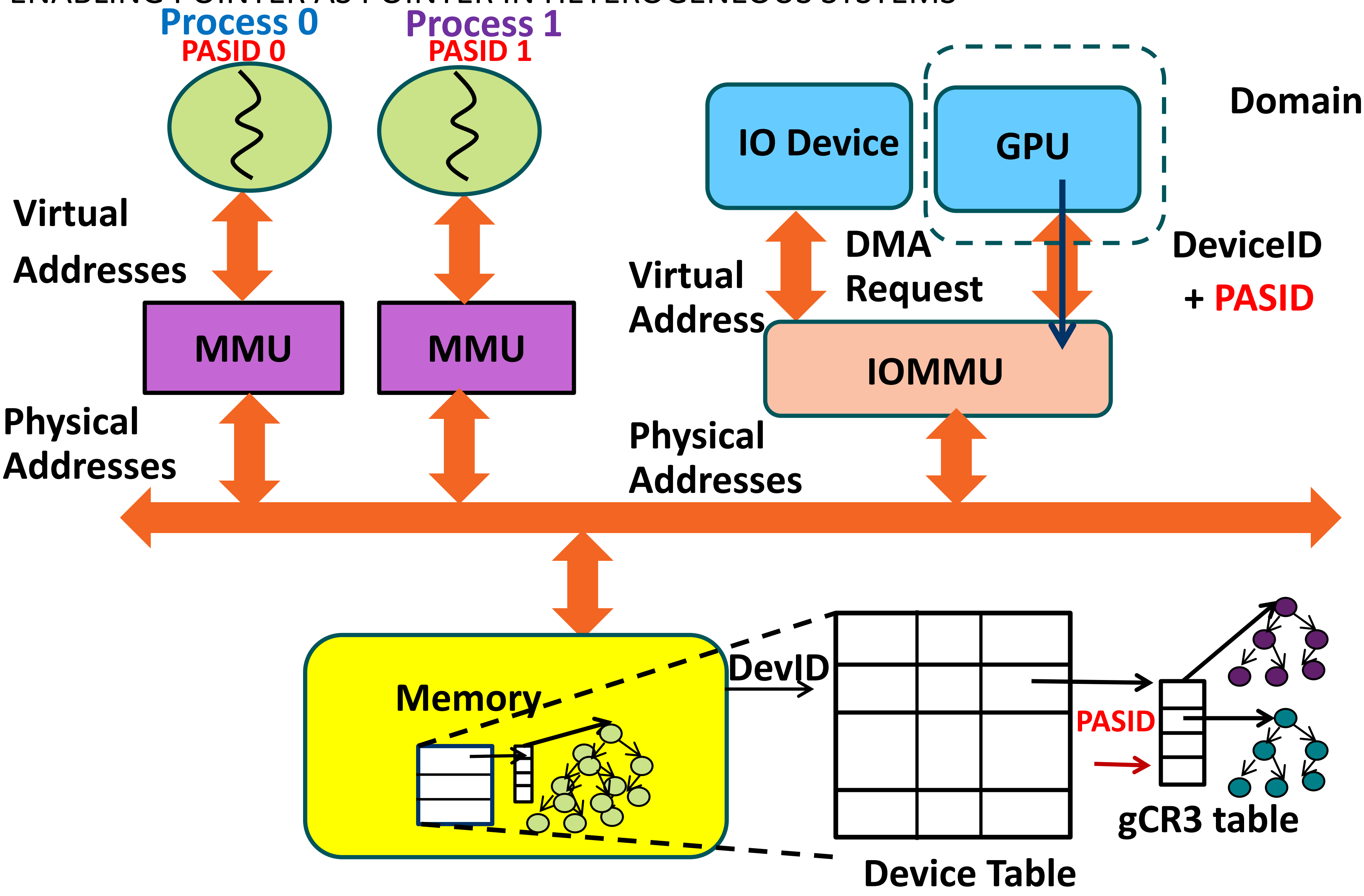
SHARING ADDRESS SPACE WITH CPU

ENABLING POINTER AS POINTER IN HETEROGENEOUS SYSTEMS



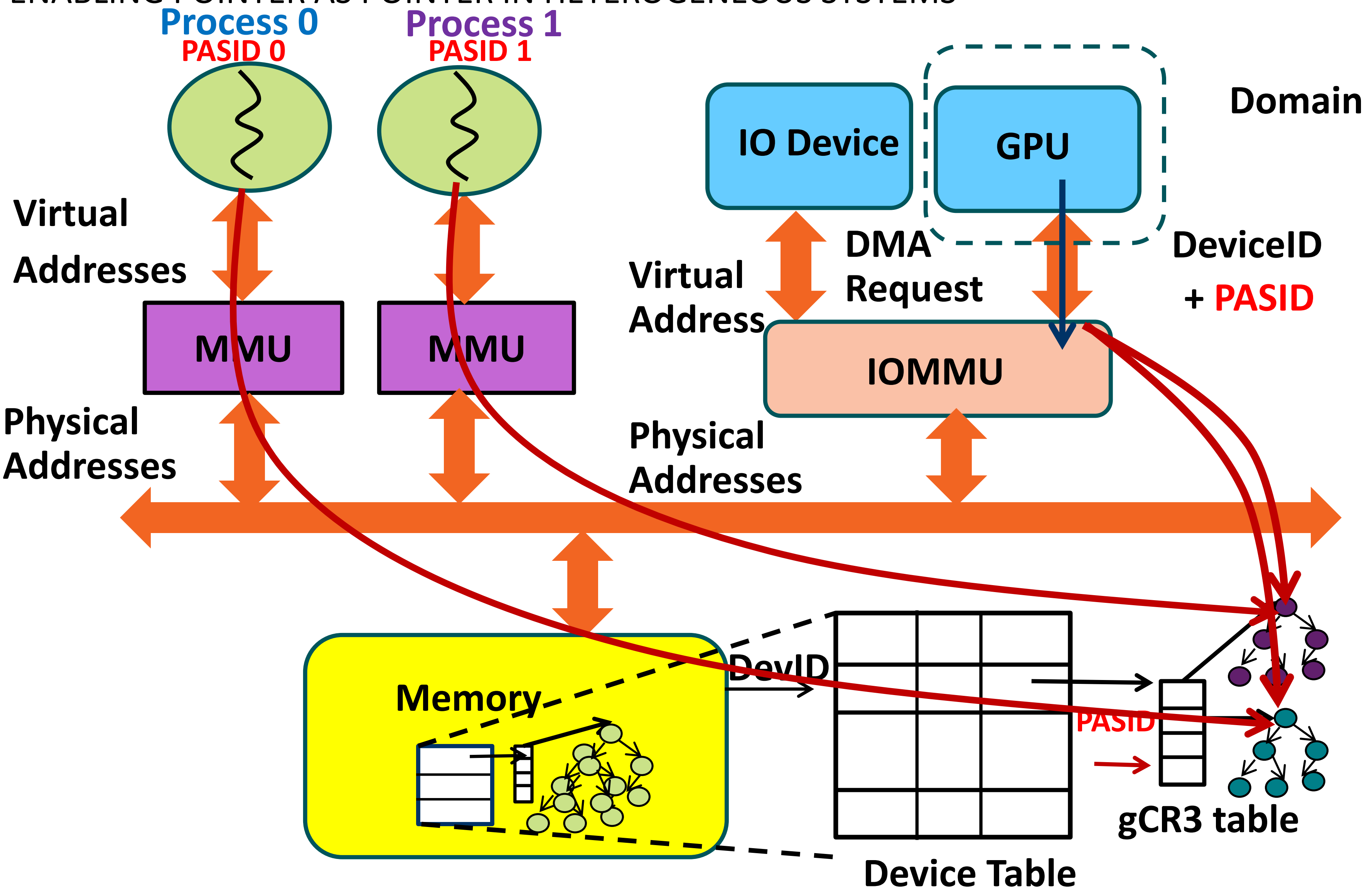
SHARING ADDRESS SPACE WITH CPU

ENABLING POINTER AS POINTER IN HETEROGENEOUS SYSTEMS



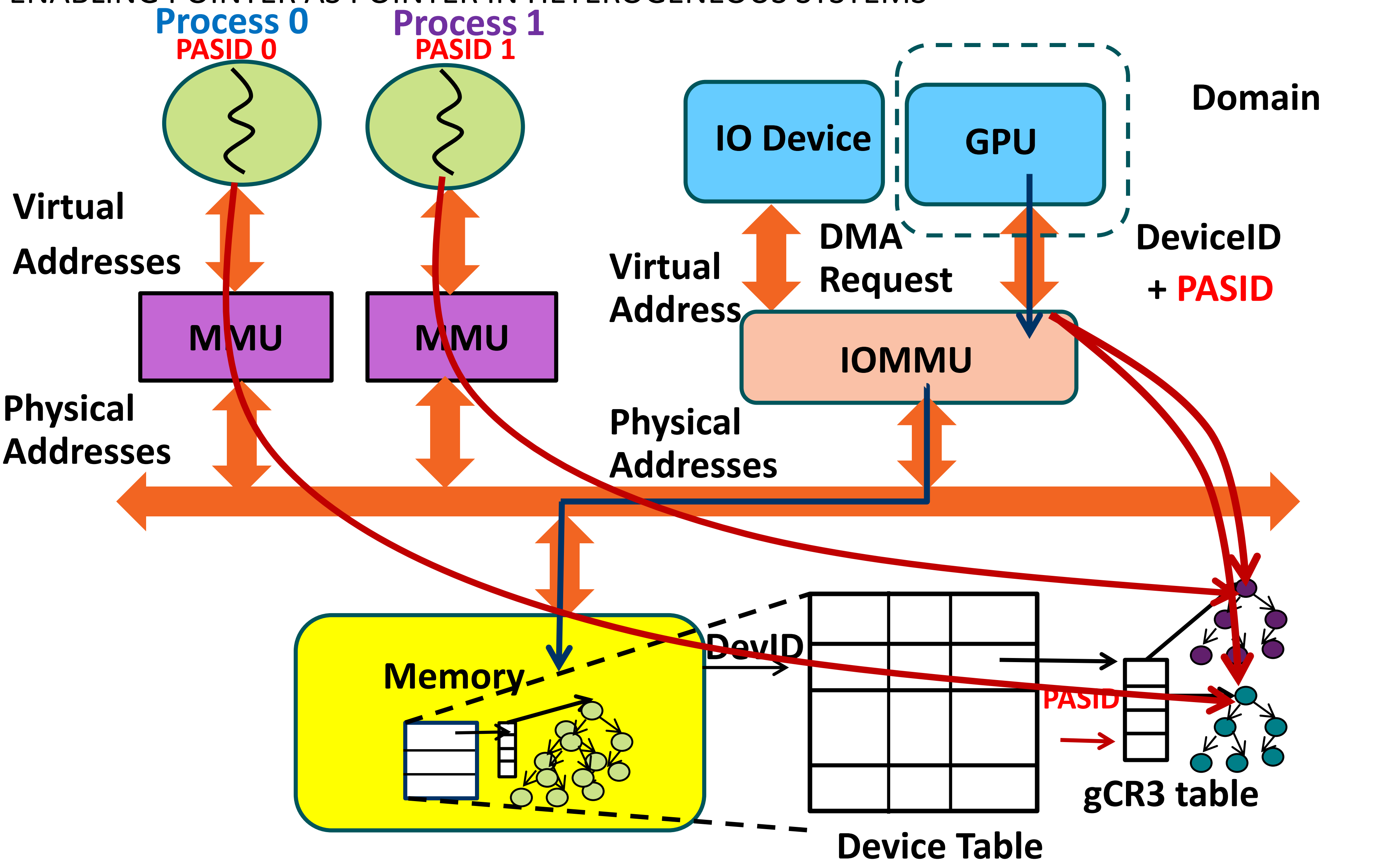
SHARING ADDRESS SPACE WITH CPU

ENABLING POINTER AS POINTER IN HETEROGENEOUS SYSTEMS



SHARING ADDRESS SPACE WITH CPU

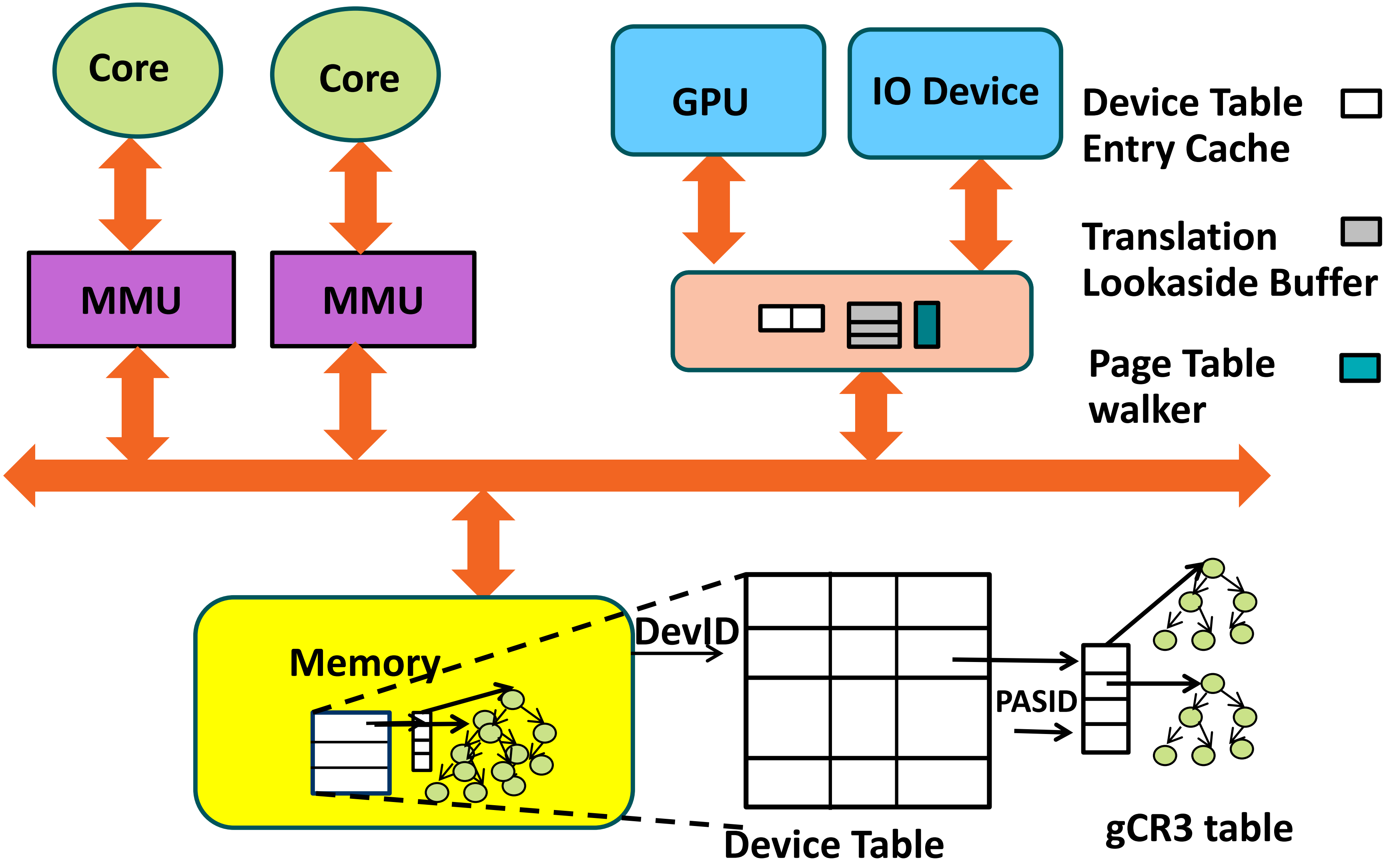
ENABLING POINTER AS POINTER IN HETEROGENEOUS SYSTEMS



IOMMU Internals:
Enabling Translation Caching in Devices

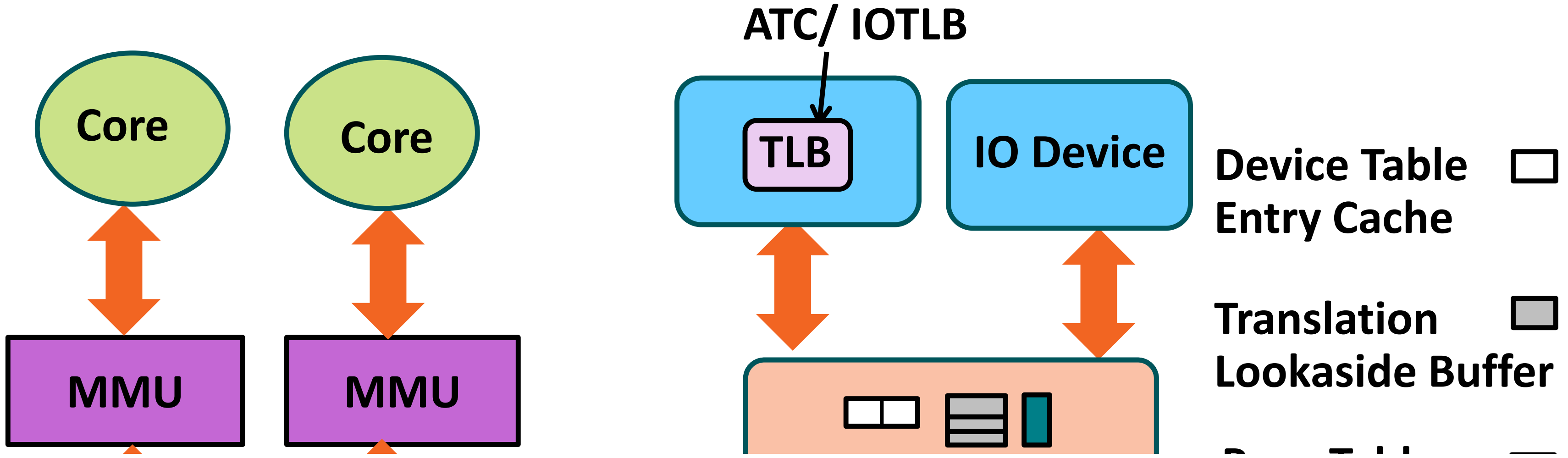
CACHING ADDRESS TRANSLATION IN DEVICES

ENABLING MORE CAPABLE DEVICE/ACCELERATORS

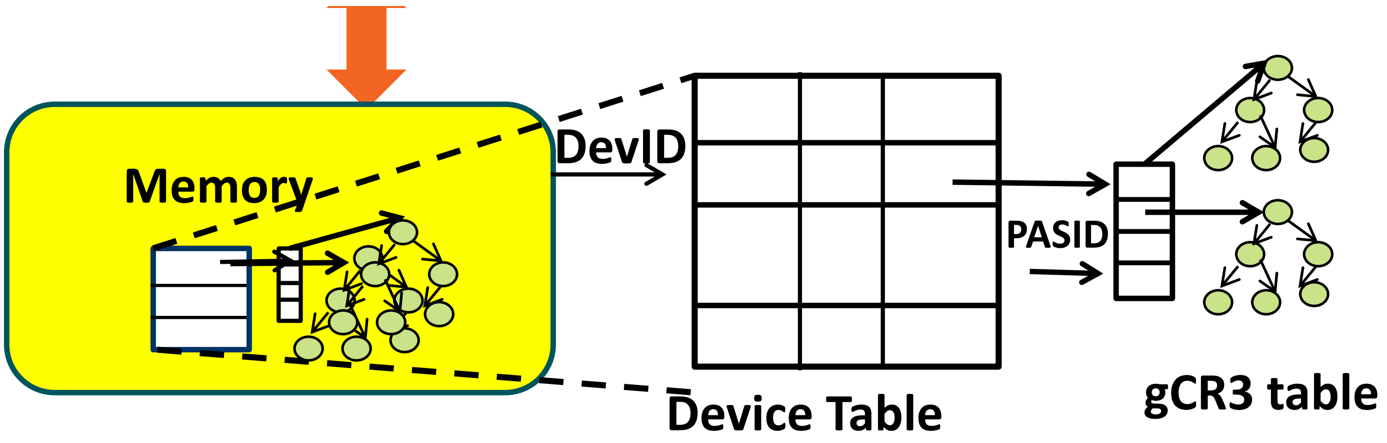


CACHING ADDRESS TRANSLATION IN DEVICES

ENABLING MORE CAPABLE DEVICE/ACCELERATORS

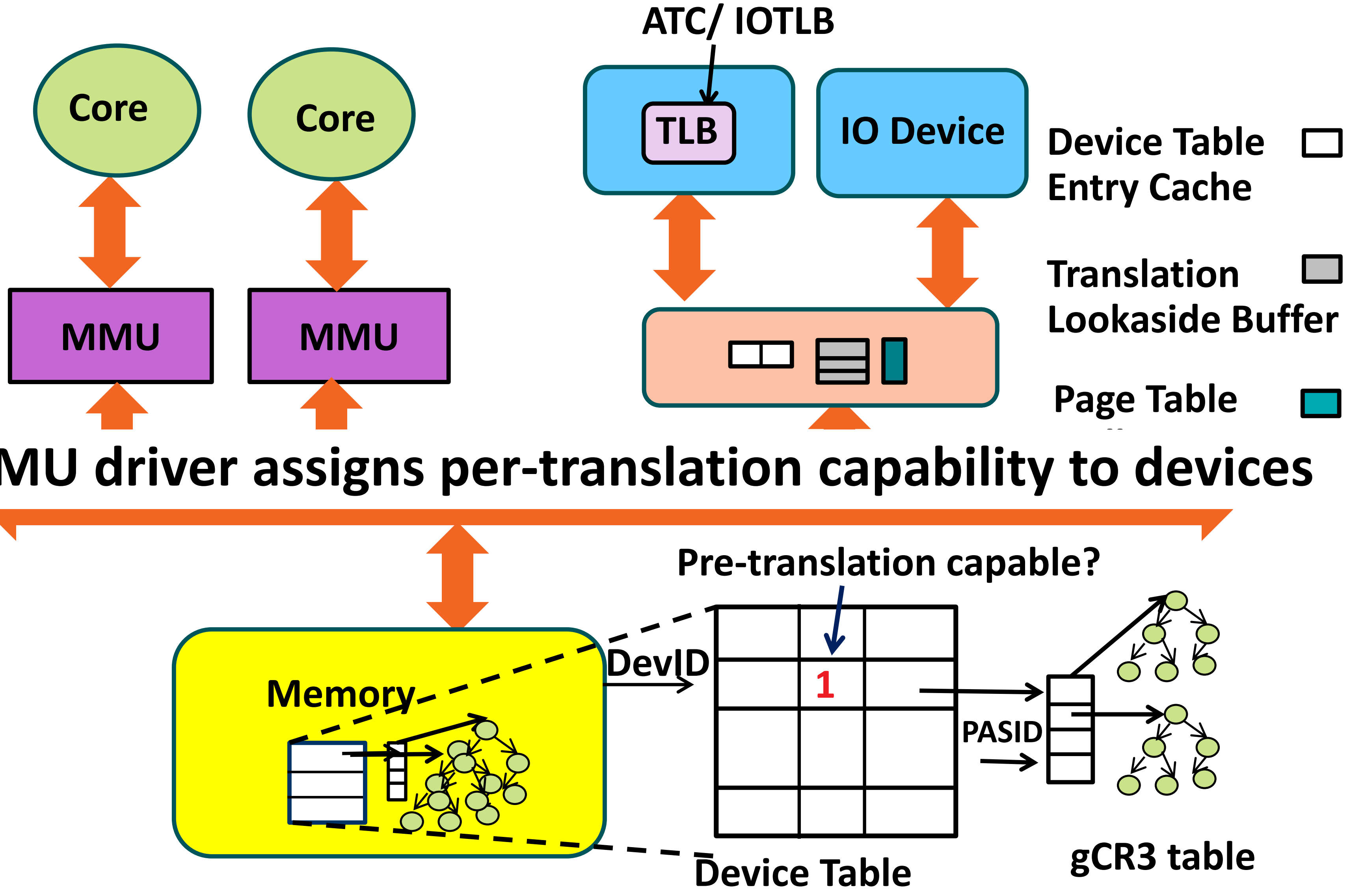


Locally caching address translation in device reduces trips to IOMMU



CACHING ADDRESS TRANSLATION IN DEVICES

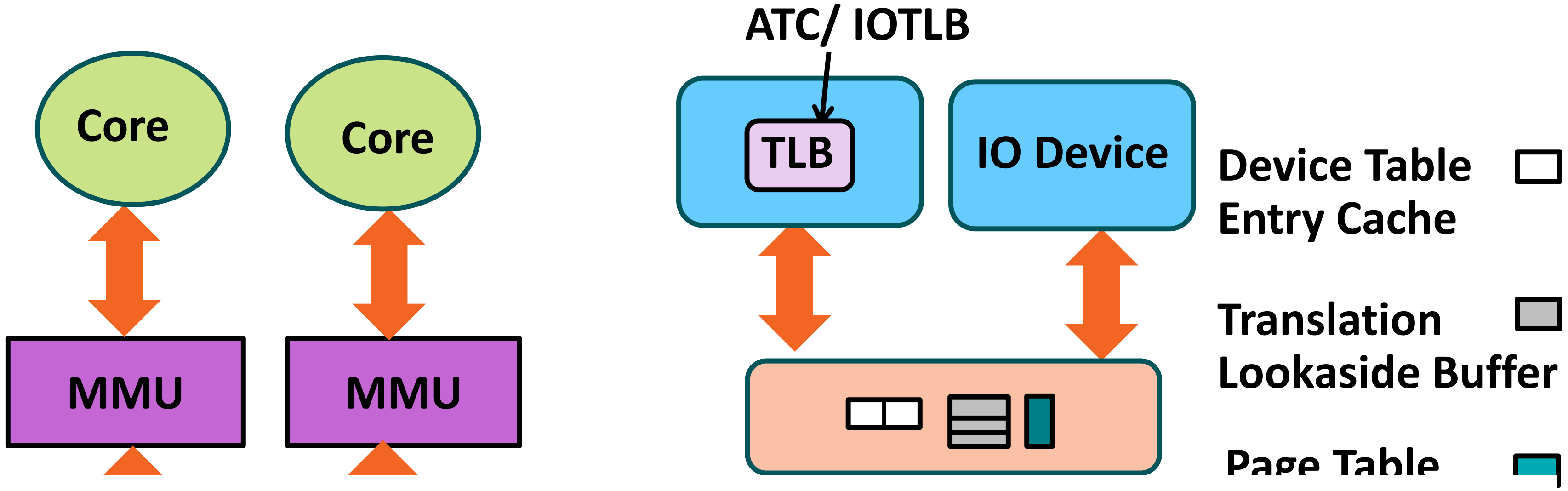
ENABLING MORE CAPABLE DEVICE/ACCELERATORS



IOMMU driver assigns per-translation capability to devices

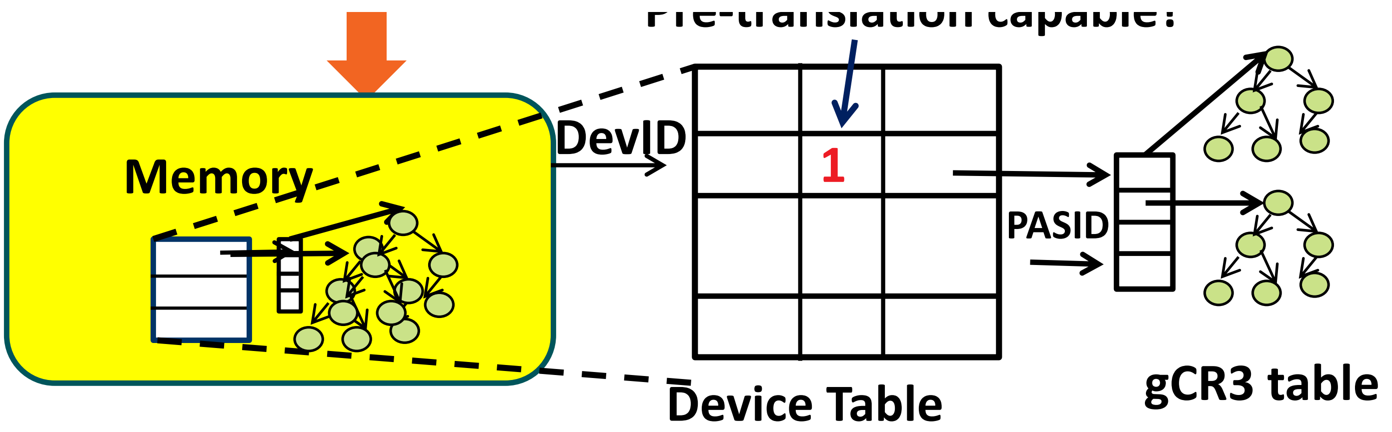
CACHING ADDRESS TRANSLATION IN DEVICES

ENABLING MORE CAPABLE DEVICE/ACCELERATORS



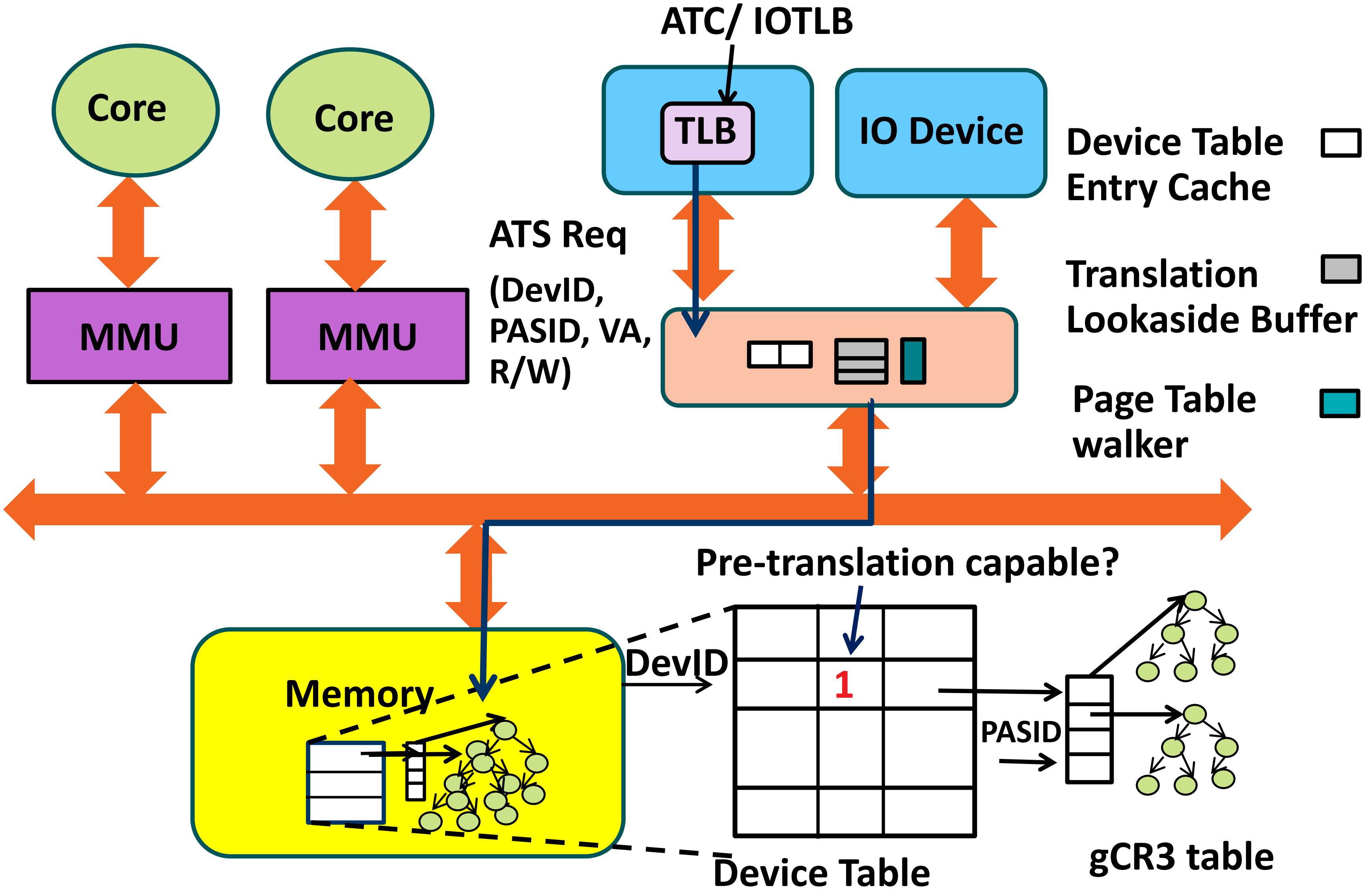
Introduce new message ype:

Address Translation Service (ATS)



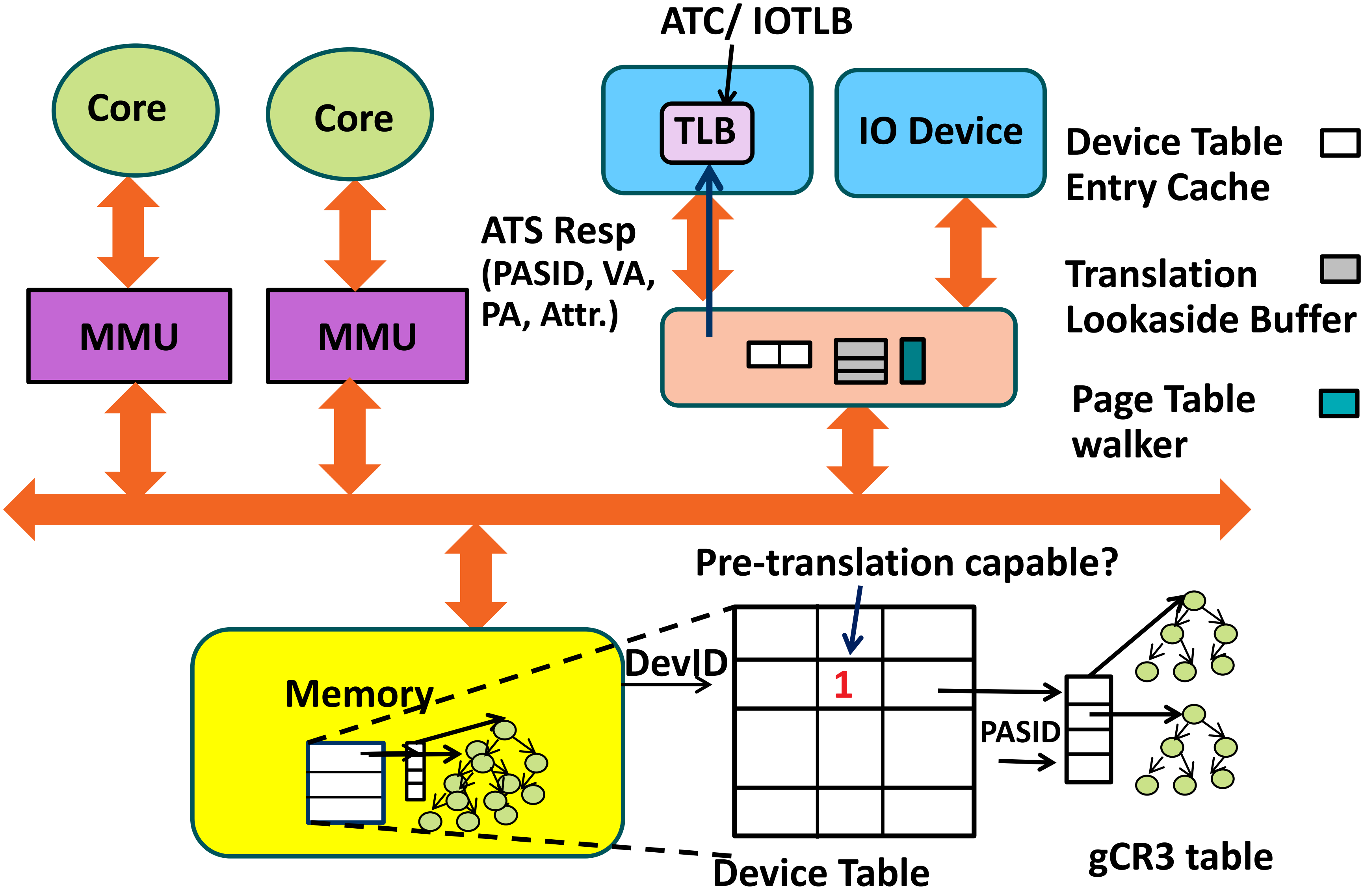
CACHING ADDRESS TRANSLATION IN DEVICES

ENABLING MORE CAPABLE DEVICE/ACCELERATORS



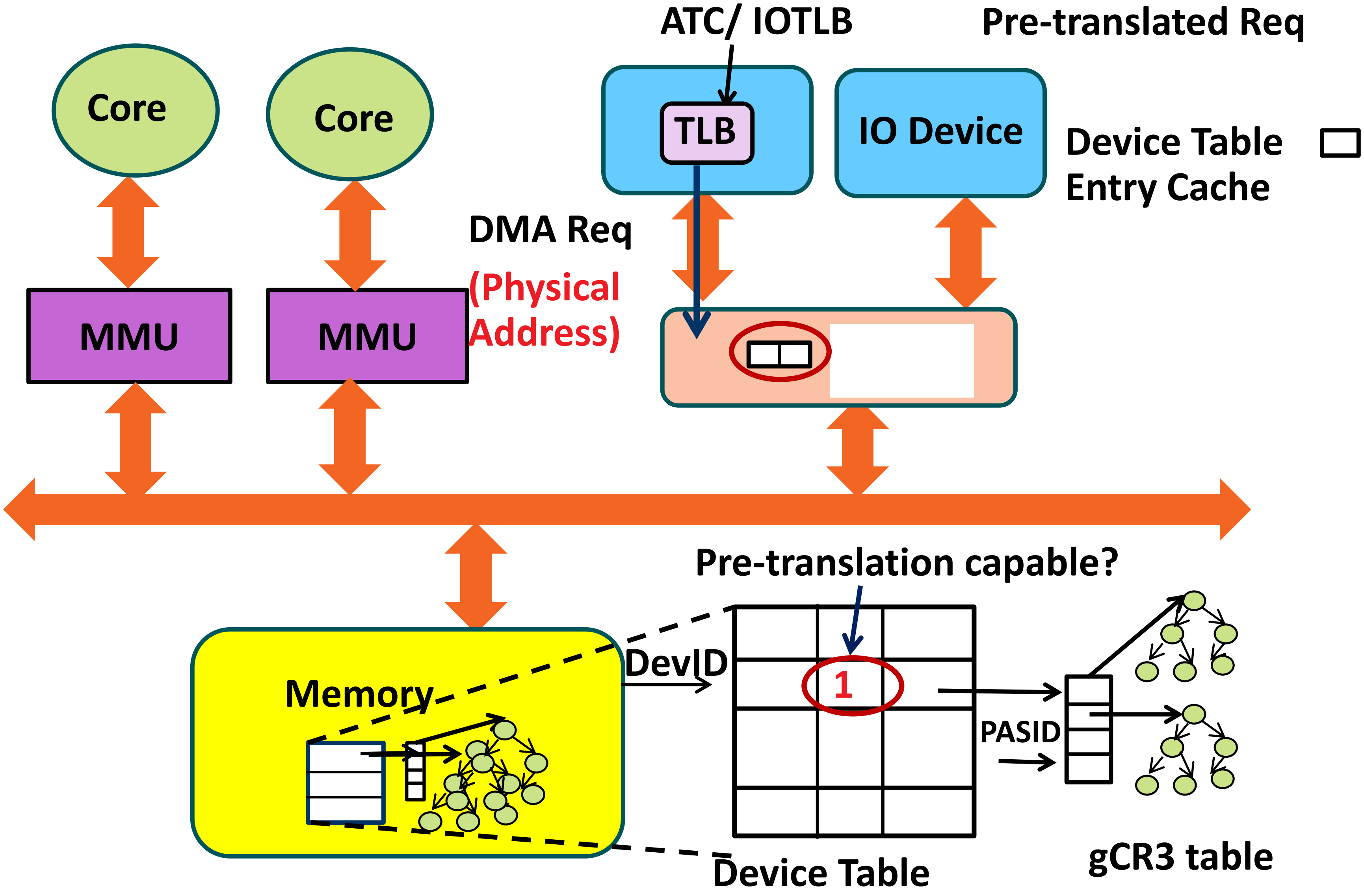
CACHING ADDRESS TRANSLATION IN DEVICES

ENABLING MORE CAPABLE DEVICE/ACCELERATORS



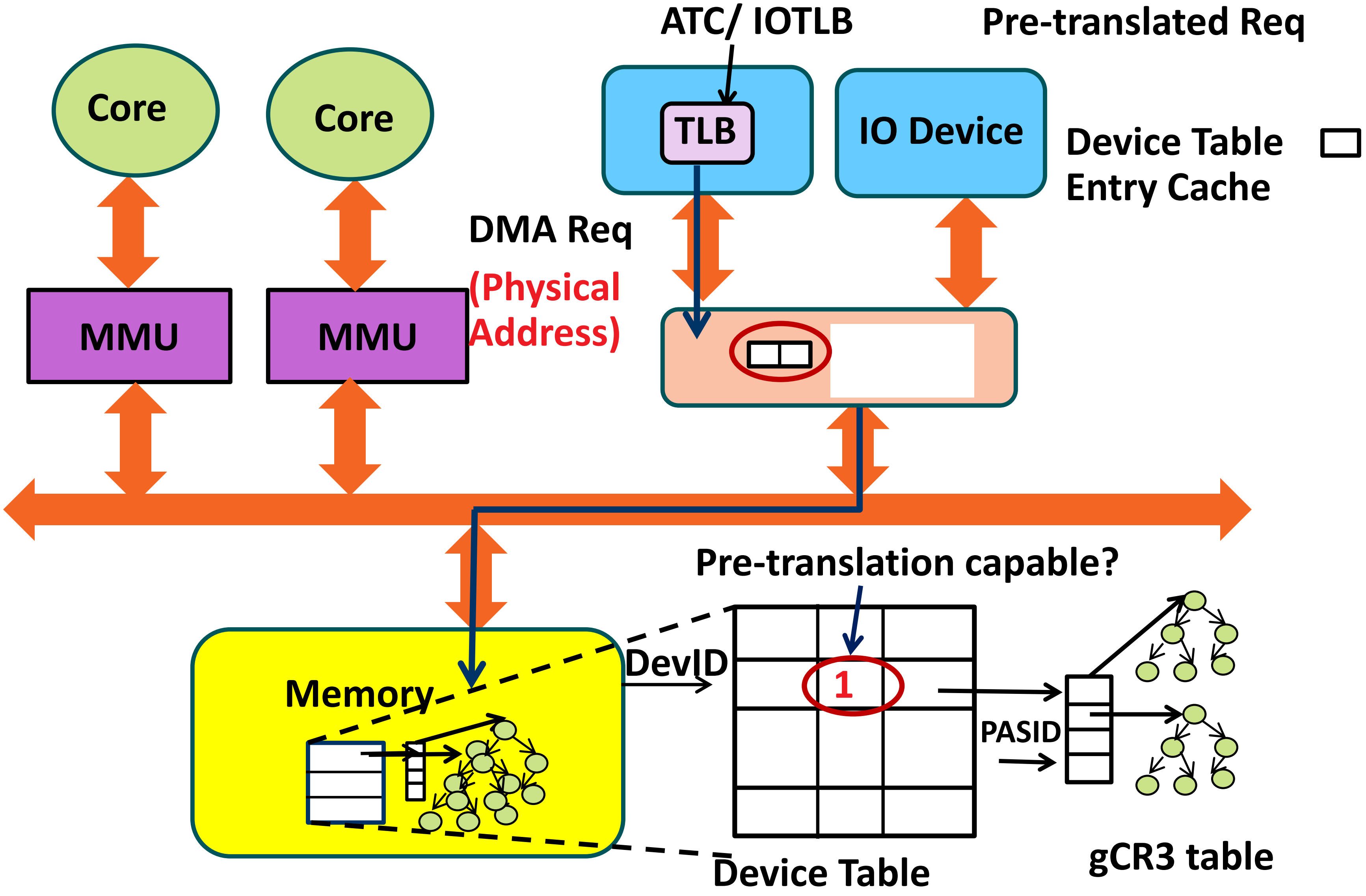
CACHING ADDRESS TRANSLATION IN DEVICES

ENABLING MORE CAPABLE DEVICE/ACCELERATORS



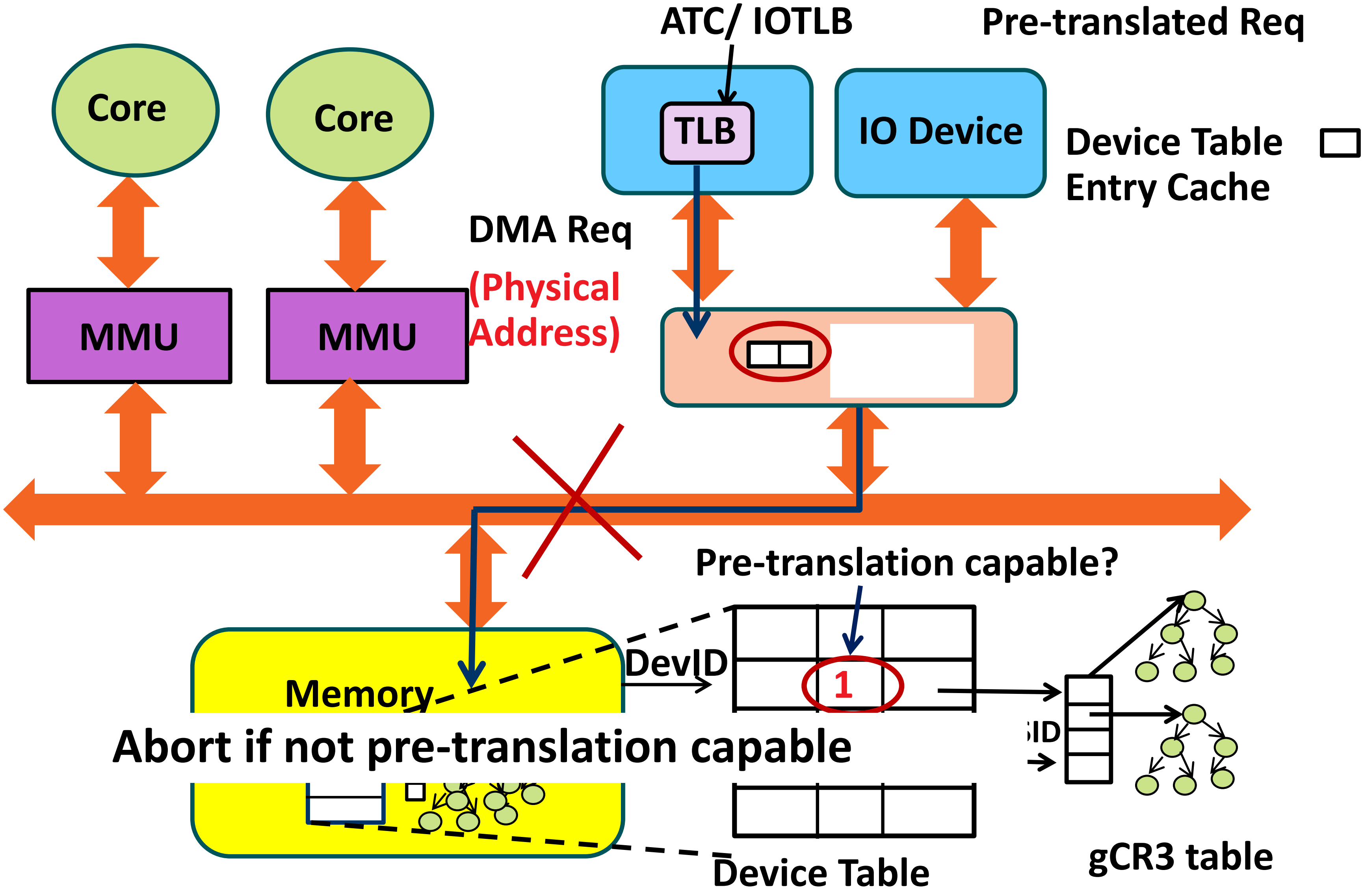
CACHING ADDRESS TRANSLATION IN DEVICES

ENABLING MORE CAPABLE DEVICE/ACCELERATORS



CACHING ADDRESS TRANSLATION IN DEVICES

ENABLING MORE CAPABLE DEVICE/ACCELERATORS

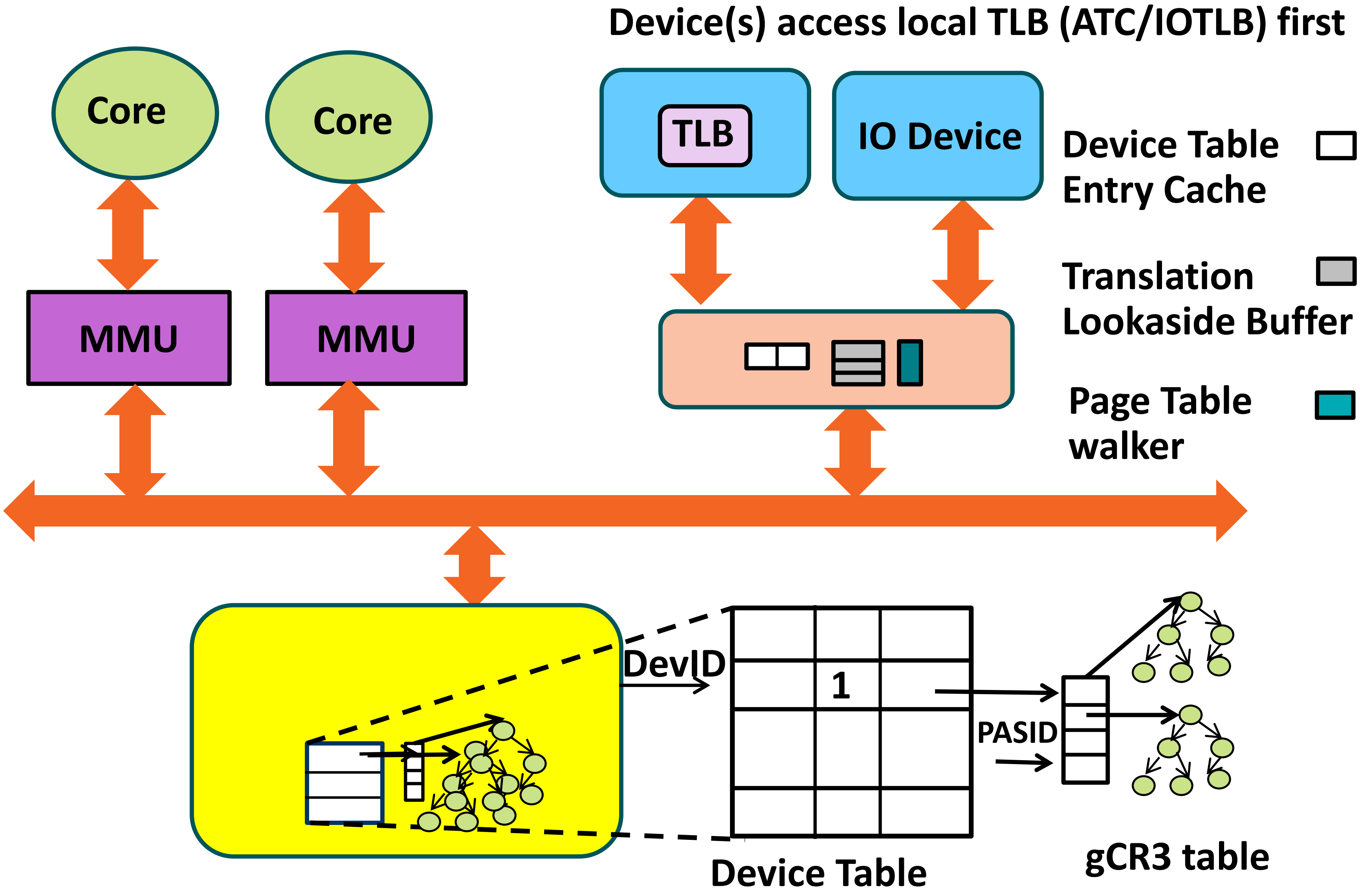


Abort if not pre-translation capable

IOMMU Internals:
Enabling Demand Paging from IO
→ No Need to Pin Memory

ENABLING DEMAND PAGING FROM DEVICE

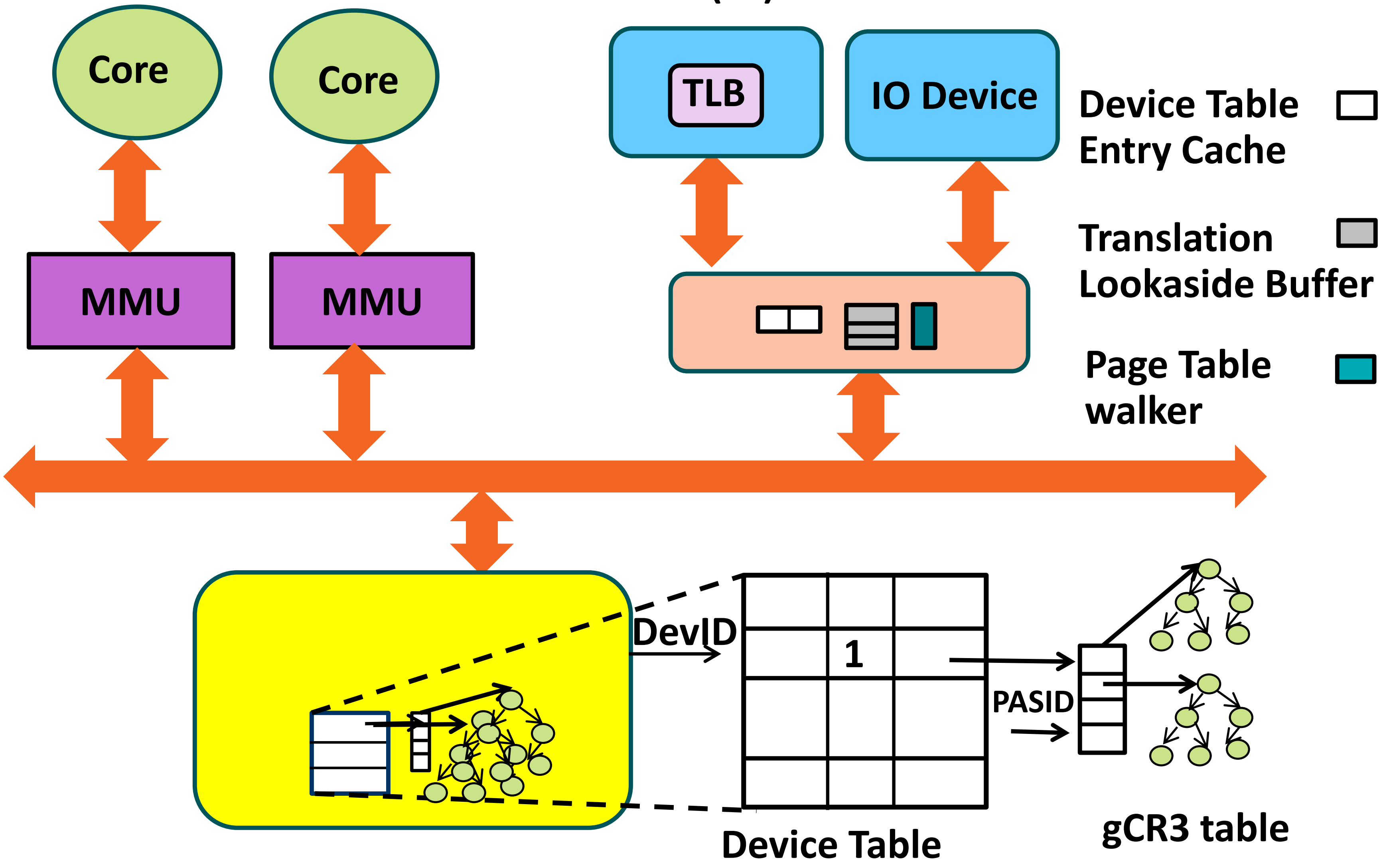
SERVICING DEVICE PAGE FAULT


Device(s) access local TLB (ATC/IOTLB) first
Core
Core
MMU
MMU
TLB
IO Device
Device Table Entry Cache □
Translation Lookaside Buffer ▨
Page Table walker ■
DevID
1
PASID
Device Table
gCR3 table

ENABLING DEMAND PAGING FROM DEVICE

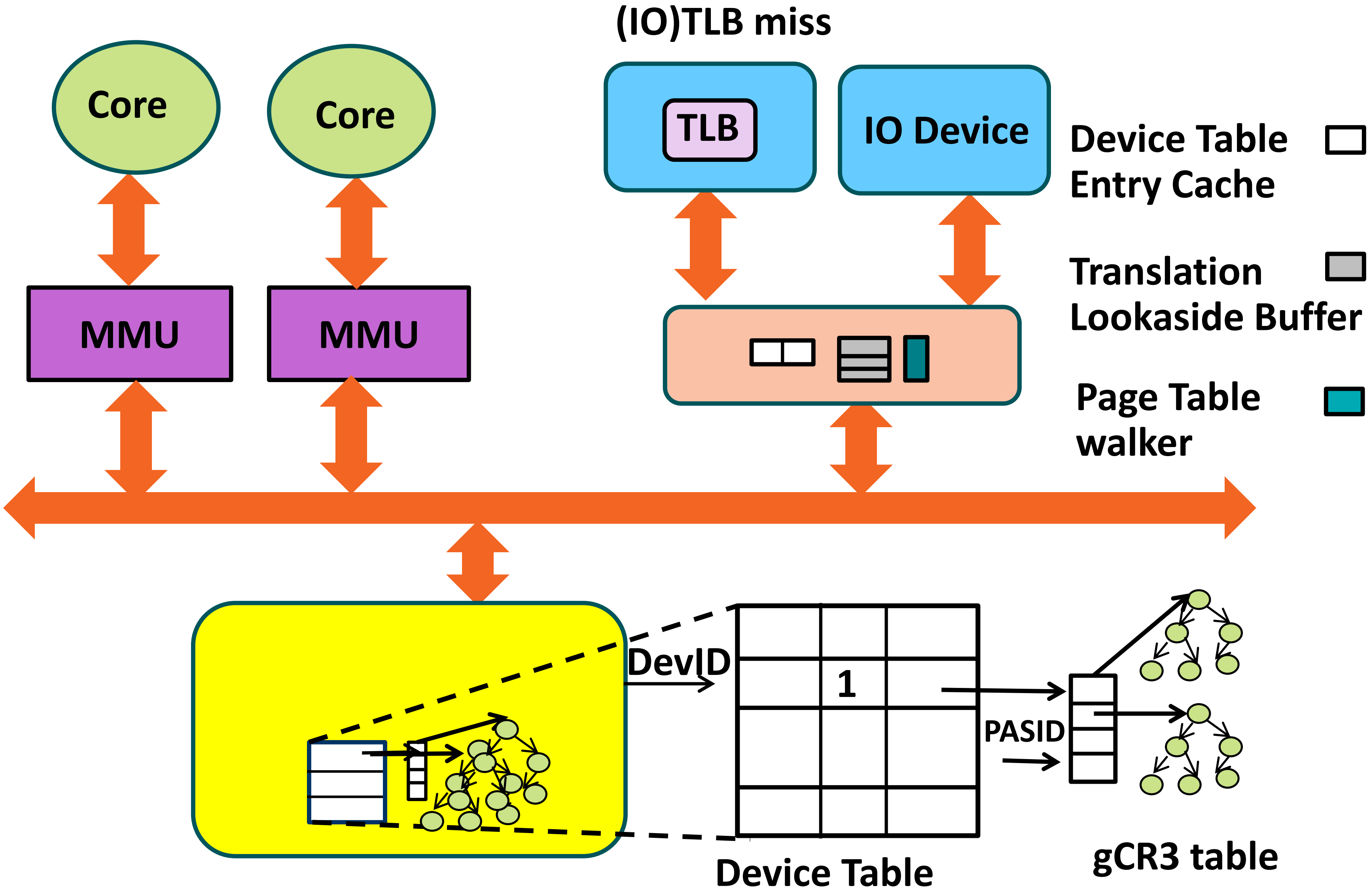
SERVICING DEVICE PAGE FAULT

On a (IO)TLB hit no access to IOMMU



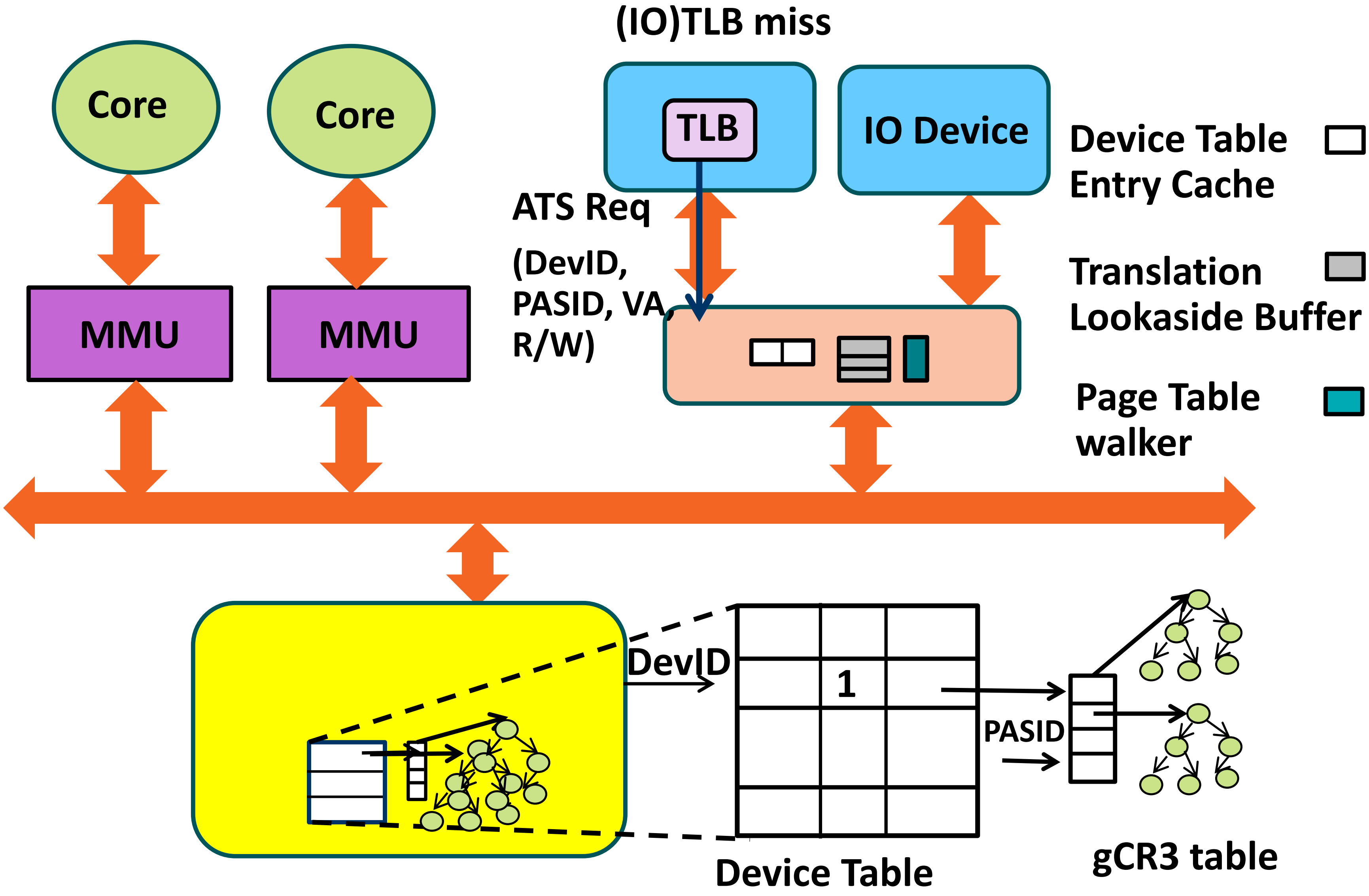
ENABLING DEMAND PAGING FROM DEVICE

SERVICING DEVICE PAGE FAULT



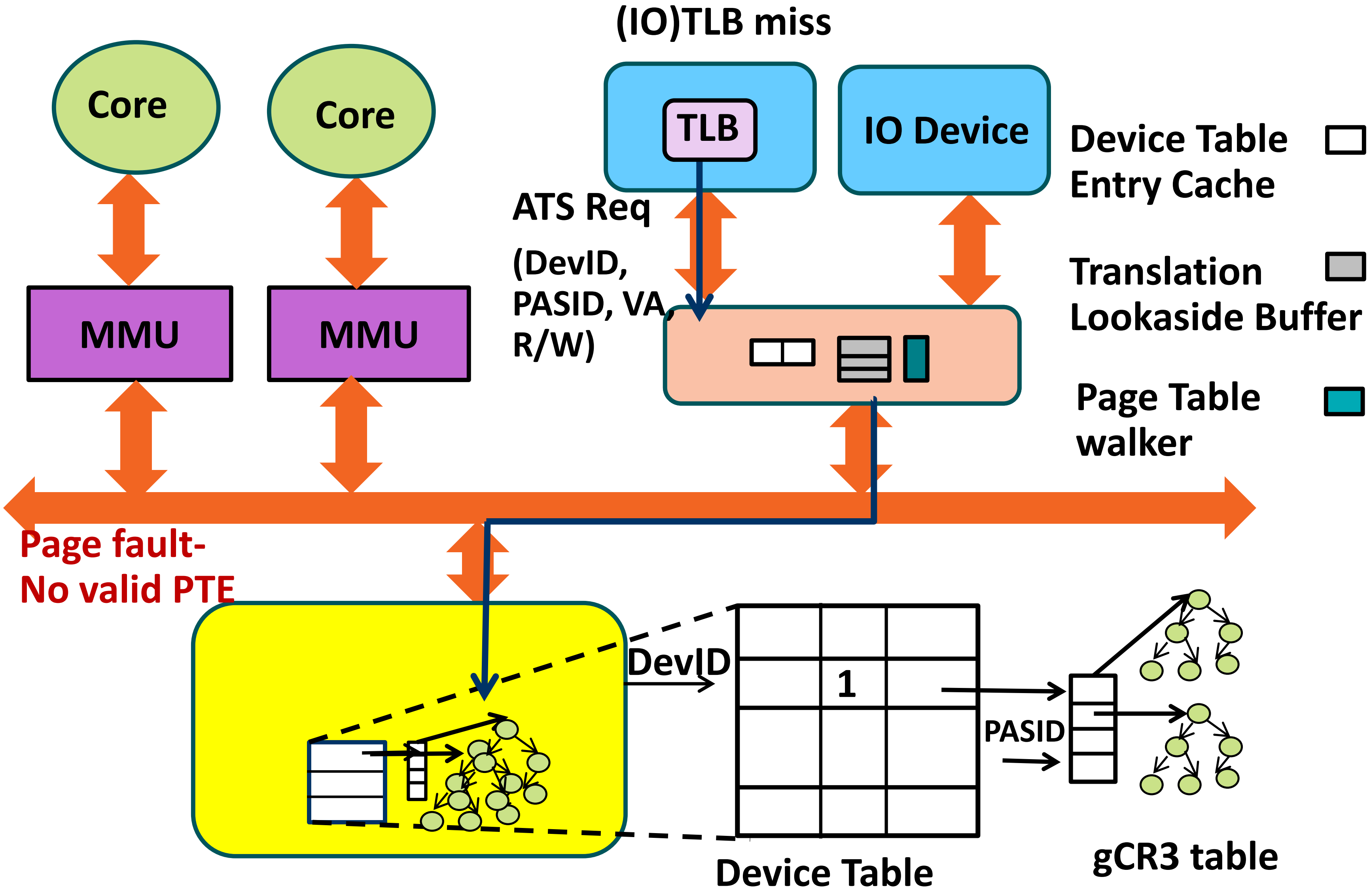
ENABLING DEMAND PAGING FROM DEVICE

SERVICING DEVICE PAGE FAULT



ENABLING DEMAND PAGING FROM DEVICE

SERVICING DEVICE PAGE FAULT



(IO)TLB miss

Core

Core

MMU

MMU

TLB

IO Device

ATS Req

(DevID, PASID, VA, R/W)

Device Table Entry Cache □

Translation Lookaside Buffer ▨

Page Table walker ■

Page fault- No valid PTE

DevID

1

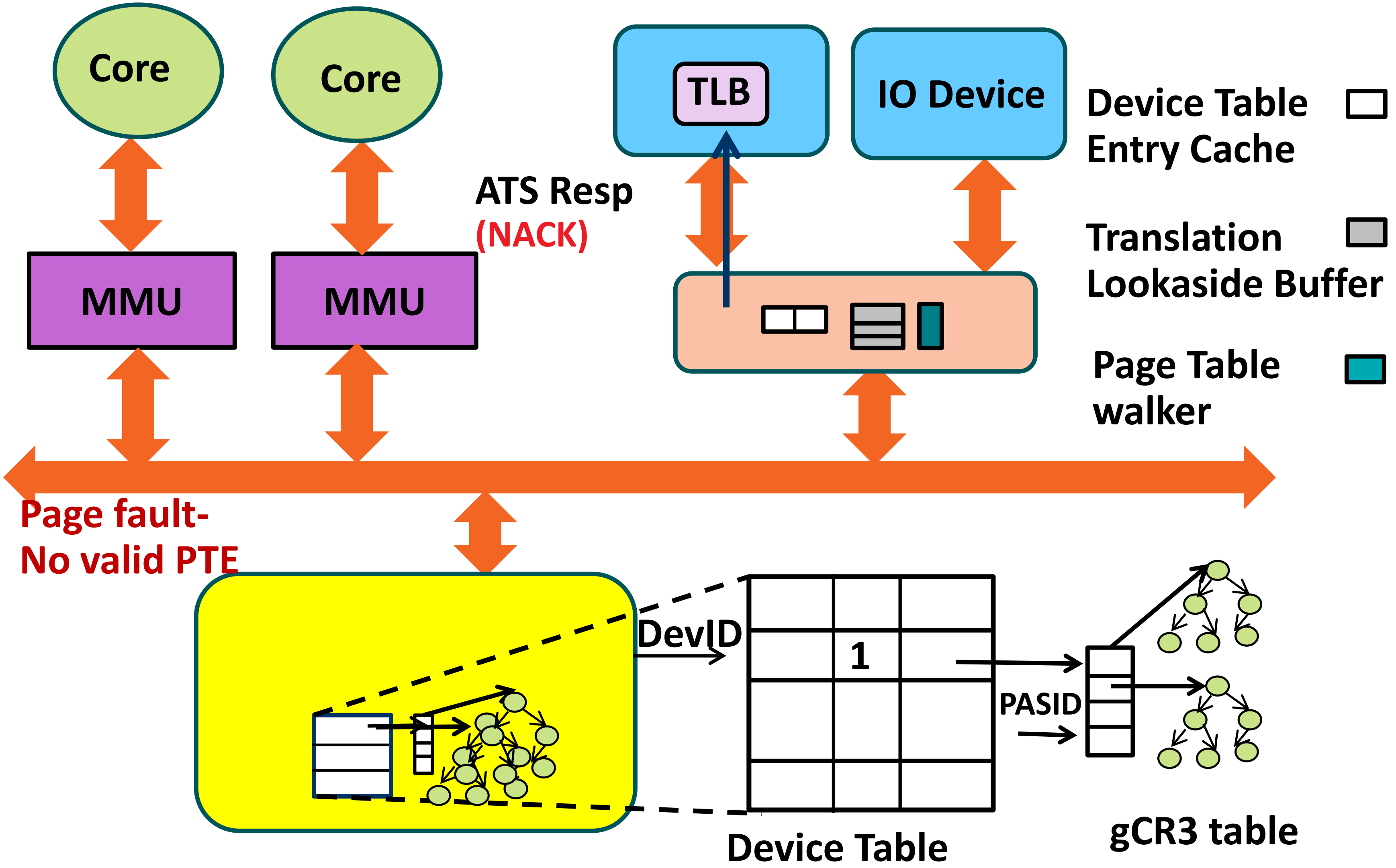
PASID

Device Table

gCR3 table

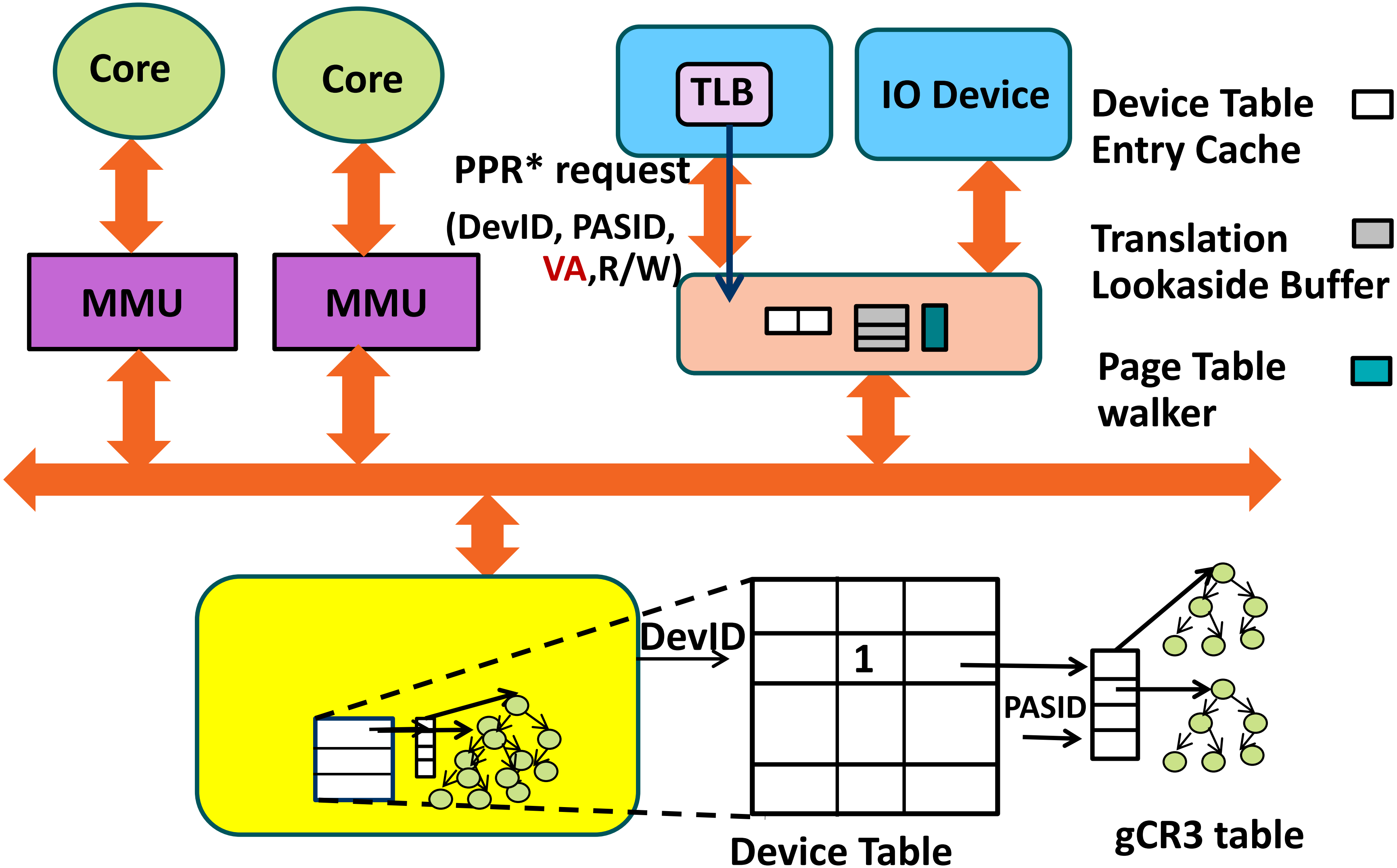
ENABLING DEMAND PAGING FROM DEVICE

SERVICING DEVICE PAGE FAULT



ENABLING DEMAND PAGING FROM DEVICE

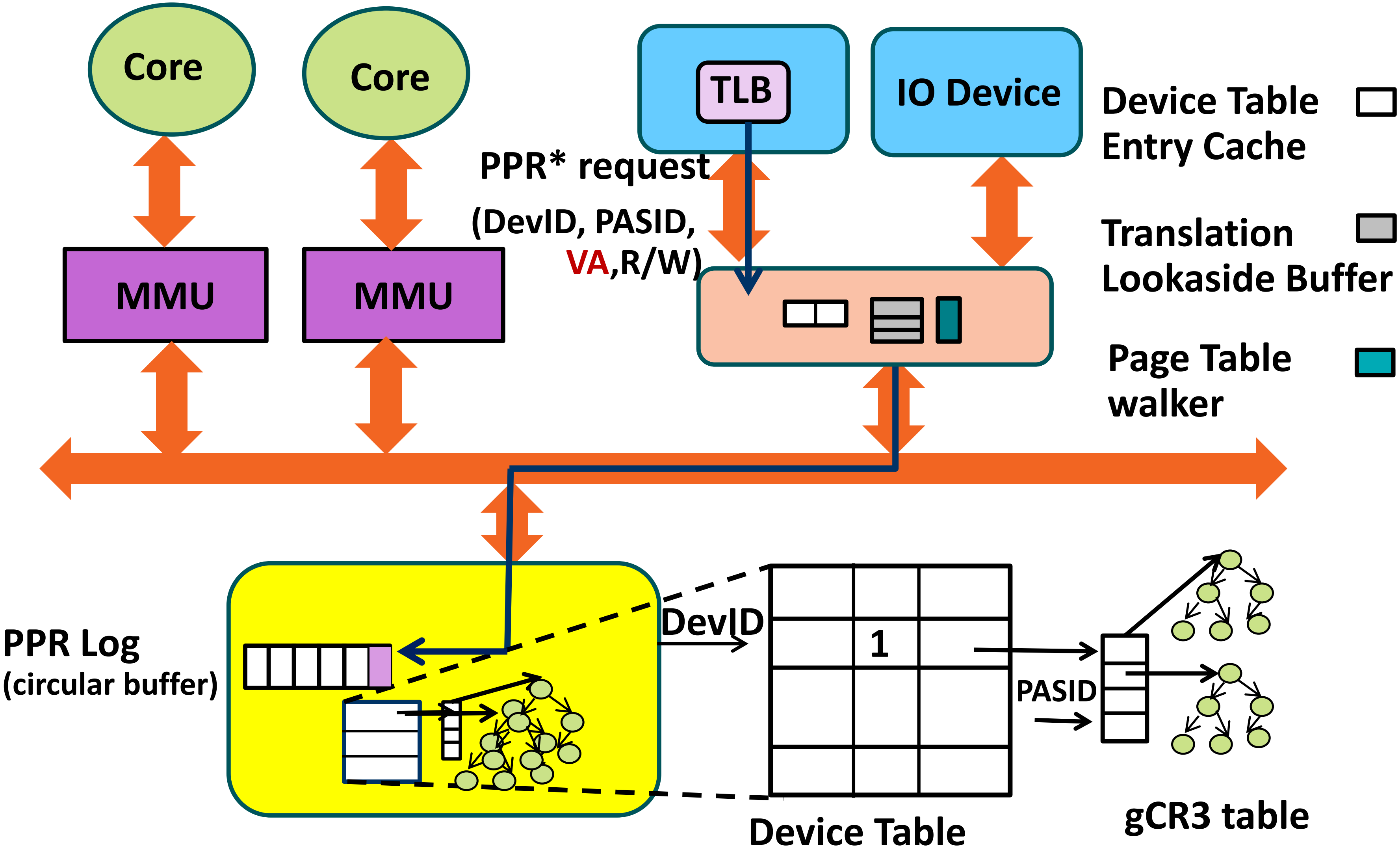
SERVICING DEVICE PAGE FAULT



*PPR= Page Peripheral Request

ENABLING DEMAND PAGING FROM DEVICE

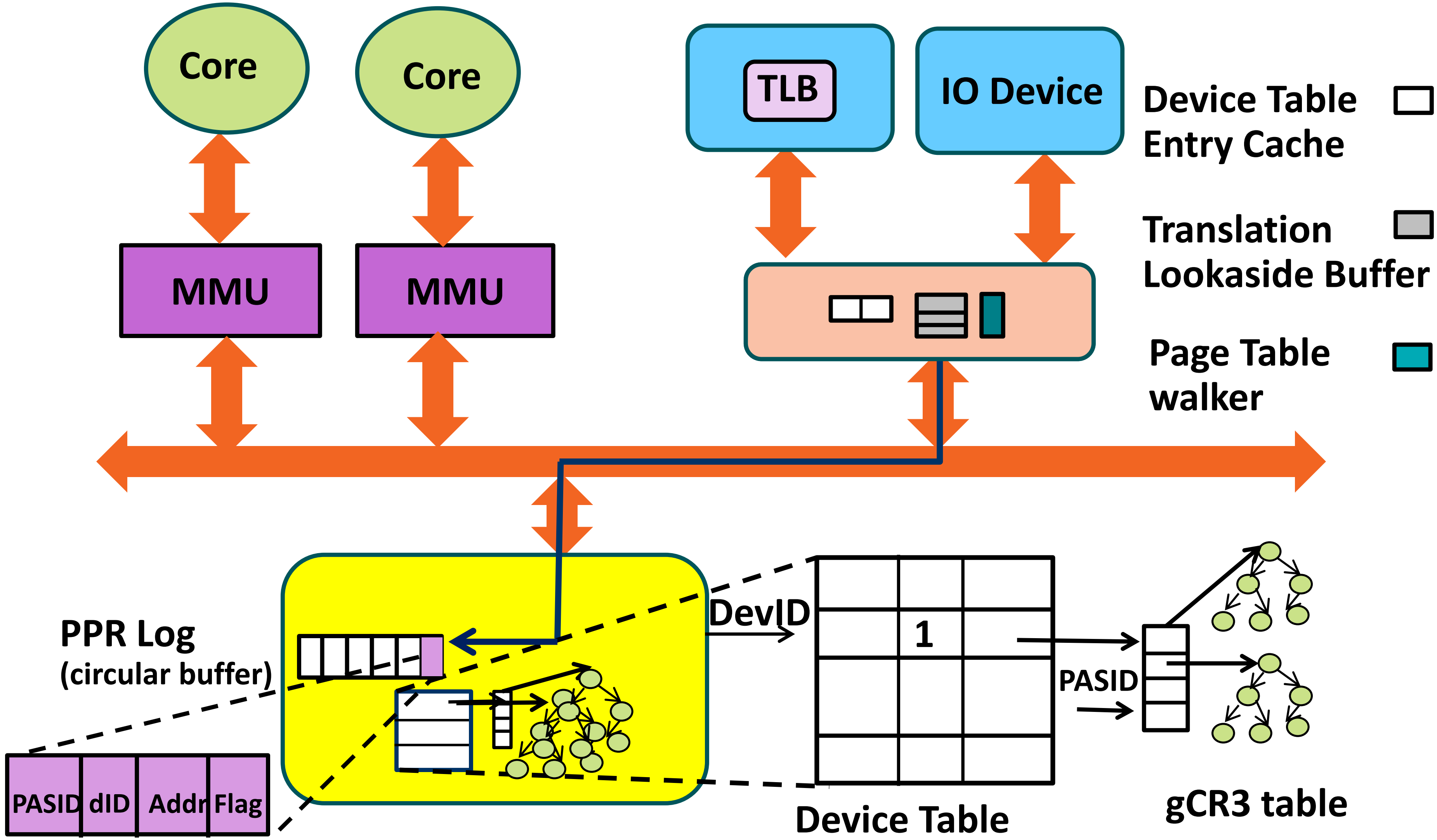
SERVICING DEVICE PAGE FAULT



*PPR= Page Peripheral Request

ENABLING DEMAND PAGING FROM DEVICE

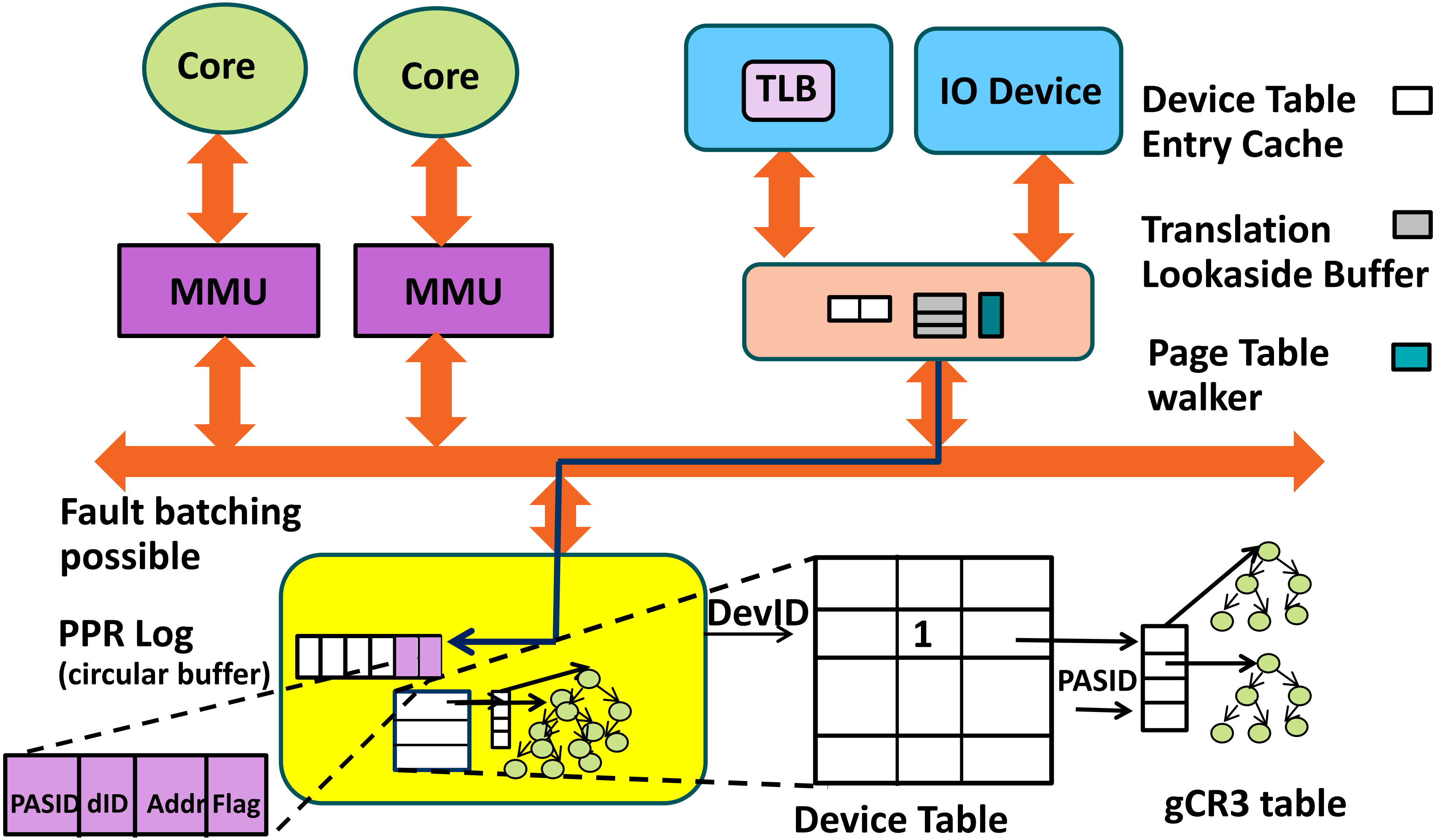
SERVICING DEVICE PAGE FAULT



*PPR= Page Peripheral Request

ENABLING DEMAND PAGING FROM DEVICE

SERVICING DEVICE PAGE FAULT



Core

Core

MMU

MMU

TLB

IO Device

Device Table Entry Cache □

Translation Lookaside Buffer ▨

Page Table walker ■

Fault batching possible

PPR Log (circular buffer)

PASID	dID	Addr	Flag

DevID

1

PASID

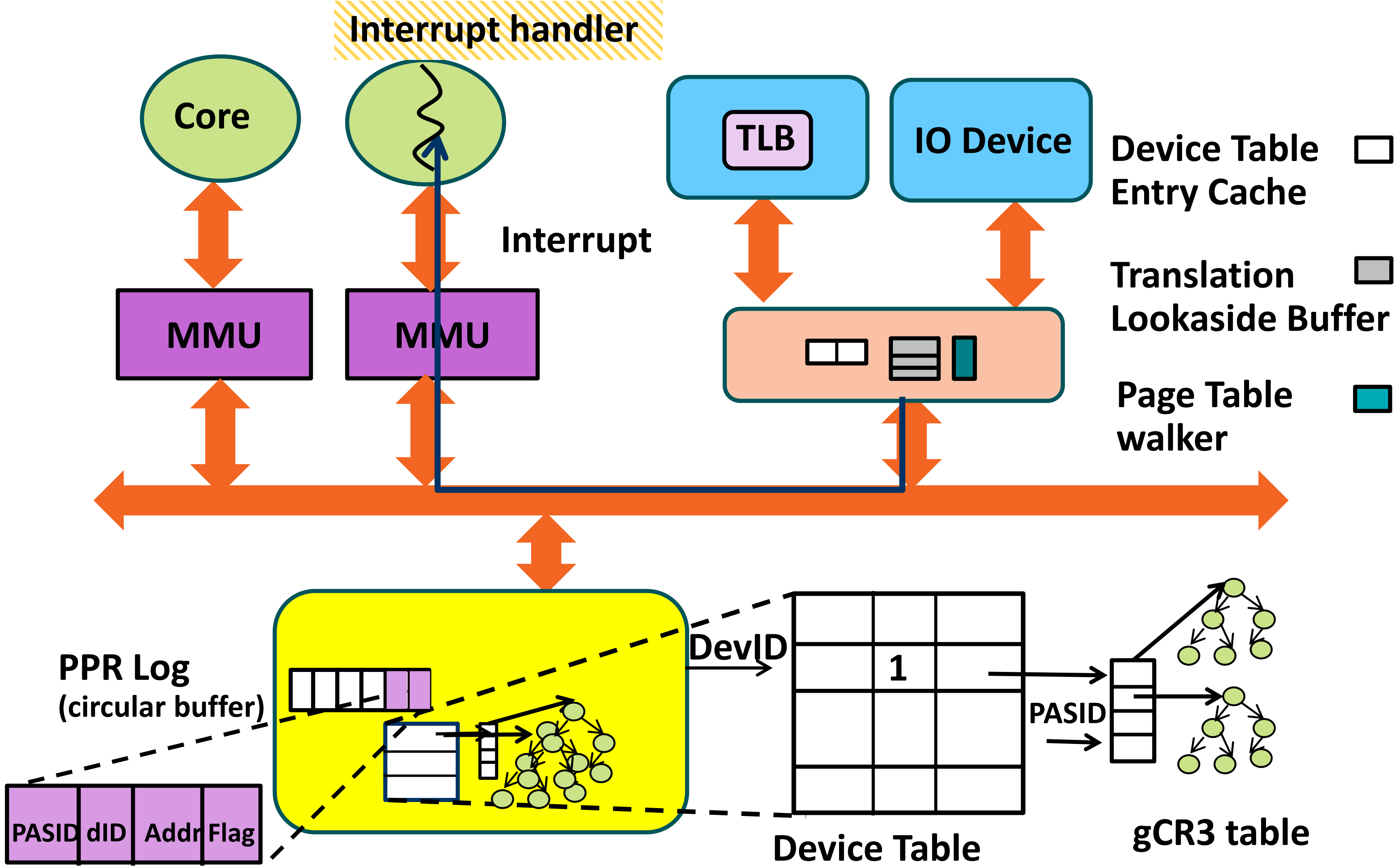
Device Table

gCR3 table

*PPR= Page Peripheral Request

ENABLING DEMAND PAGING FROM DEVICE

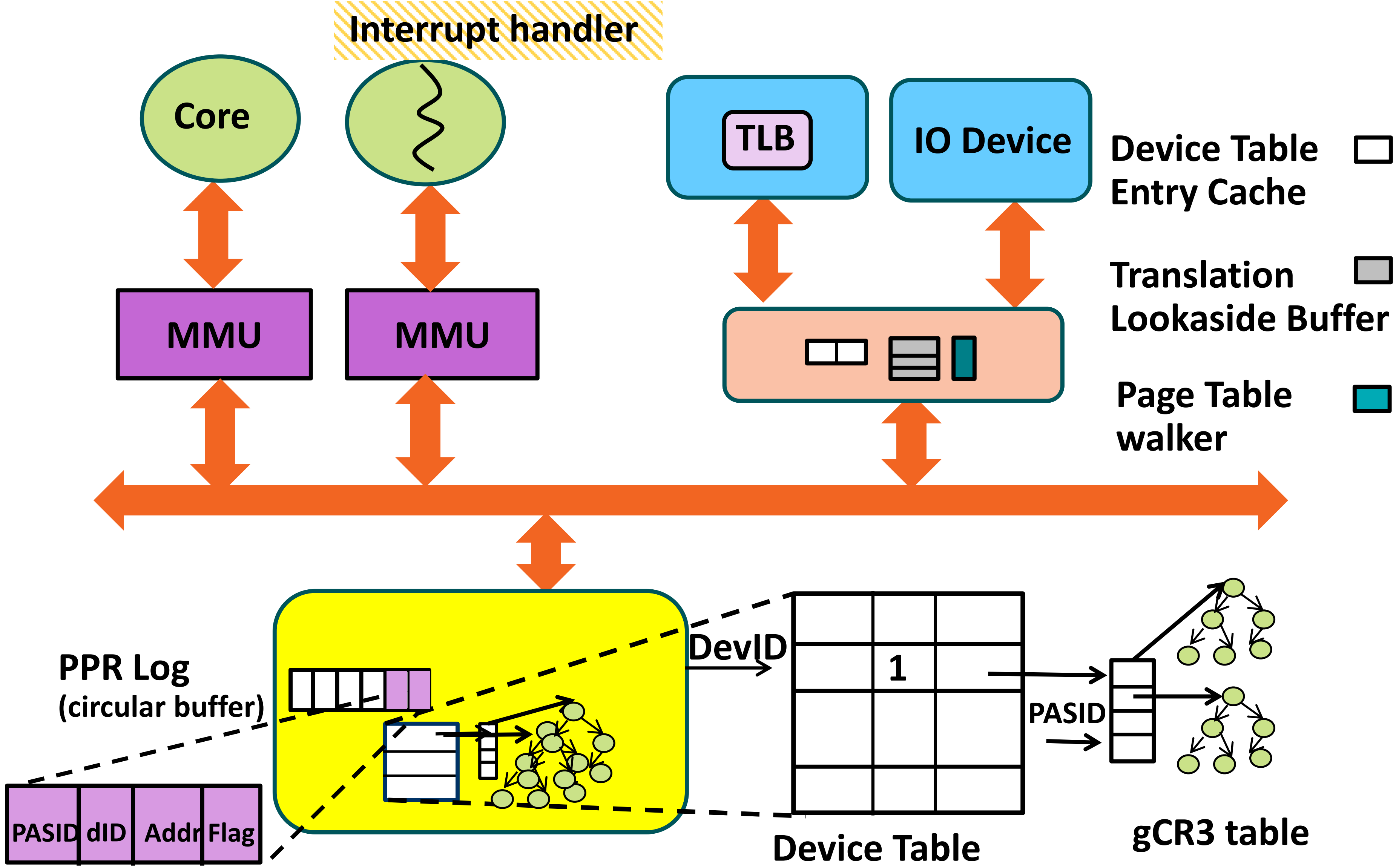
SERVICING DEVICE PAGE FAULT



*PPR= Page Peripheral Request

ENABLING DEMAND PAGING FROM DEVICE

SERVICING DEVICE PAGE FAULT

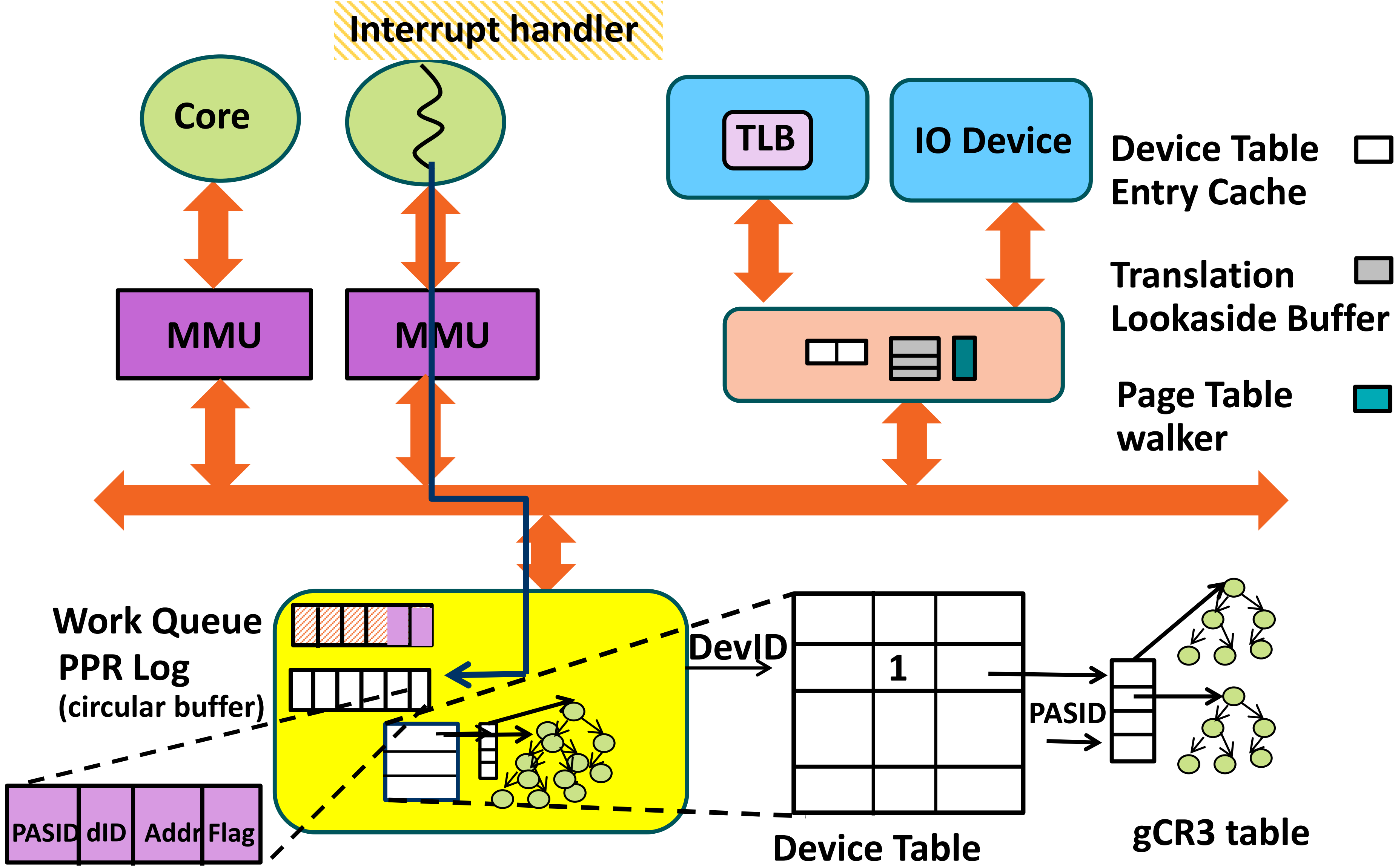


*PPR= Page Peripheral Request

ENABLING DEMAND PAGING FROM DEVICE

AMD

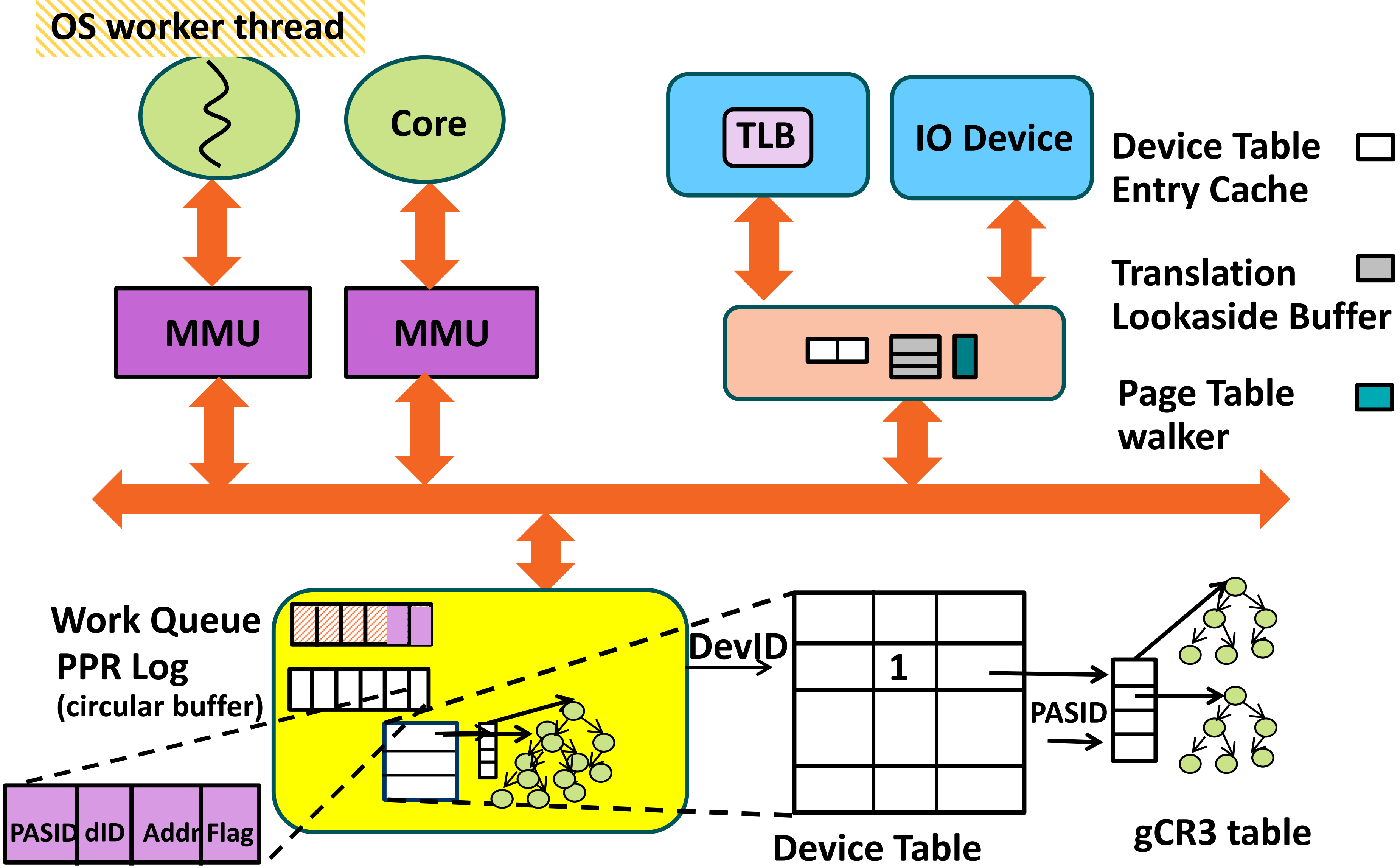
SERVICING DEVICE PAGE FAULT



*PPR= Page Peripheral Request

ENABLING DEMAND PAGING FROM DEVICE

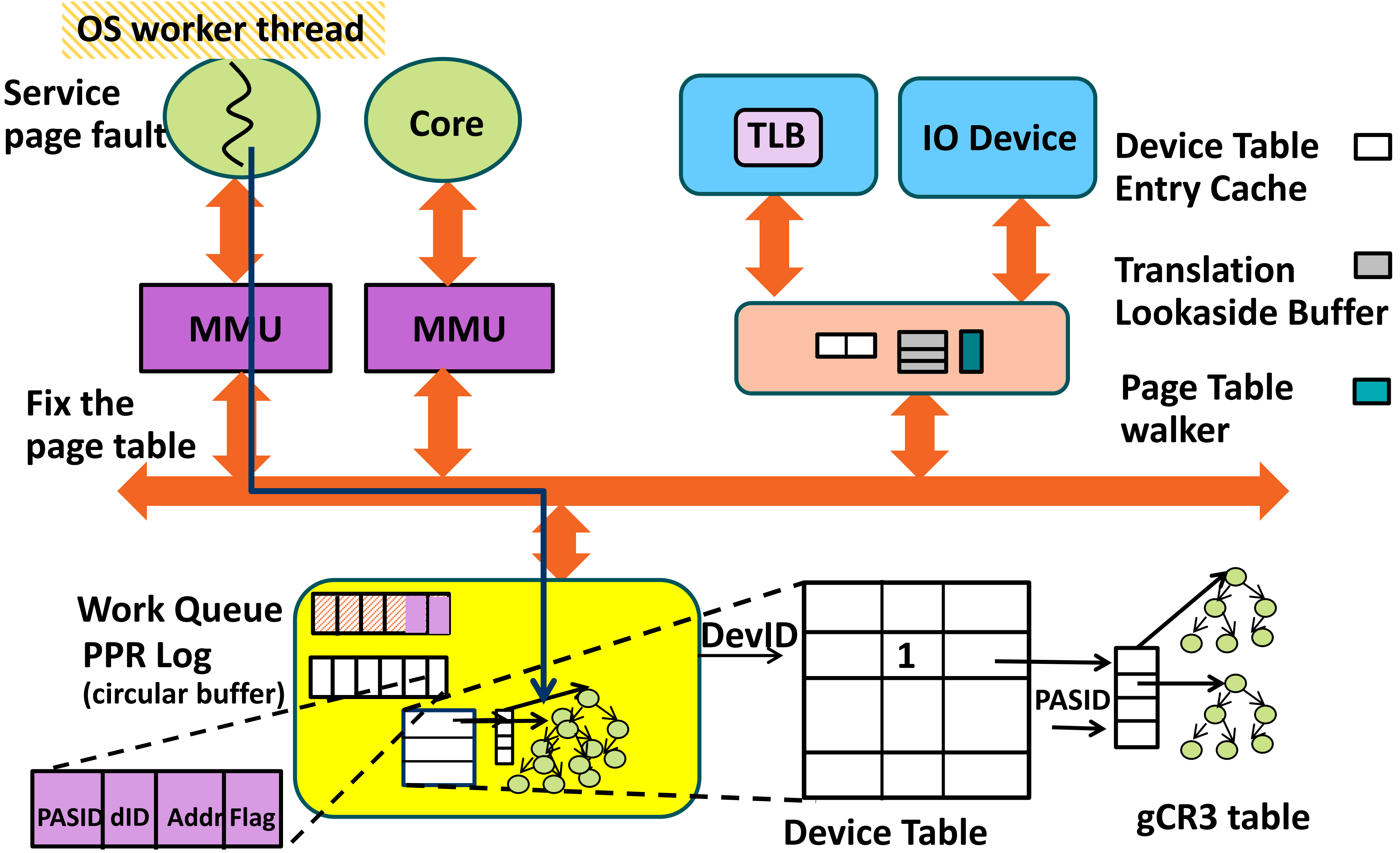
SERVICING DEVICE PAGE FAULT



*PPR= Page Peripheral Request

ENABLING DEMAND PAGING FROM DEVICE

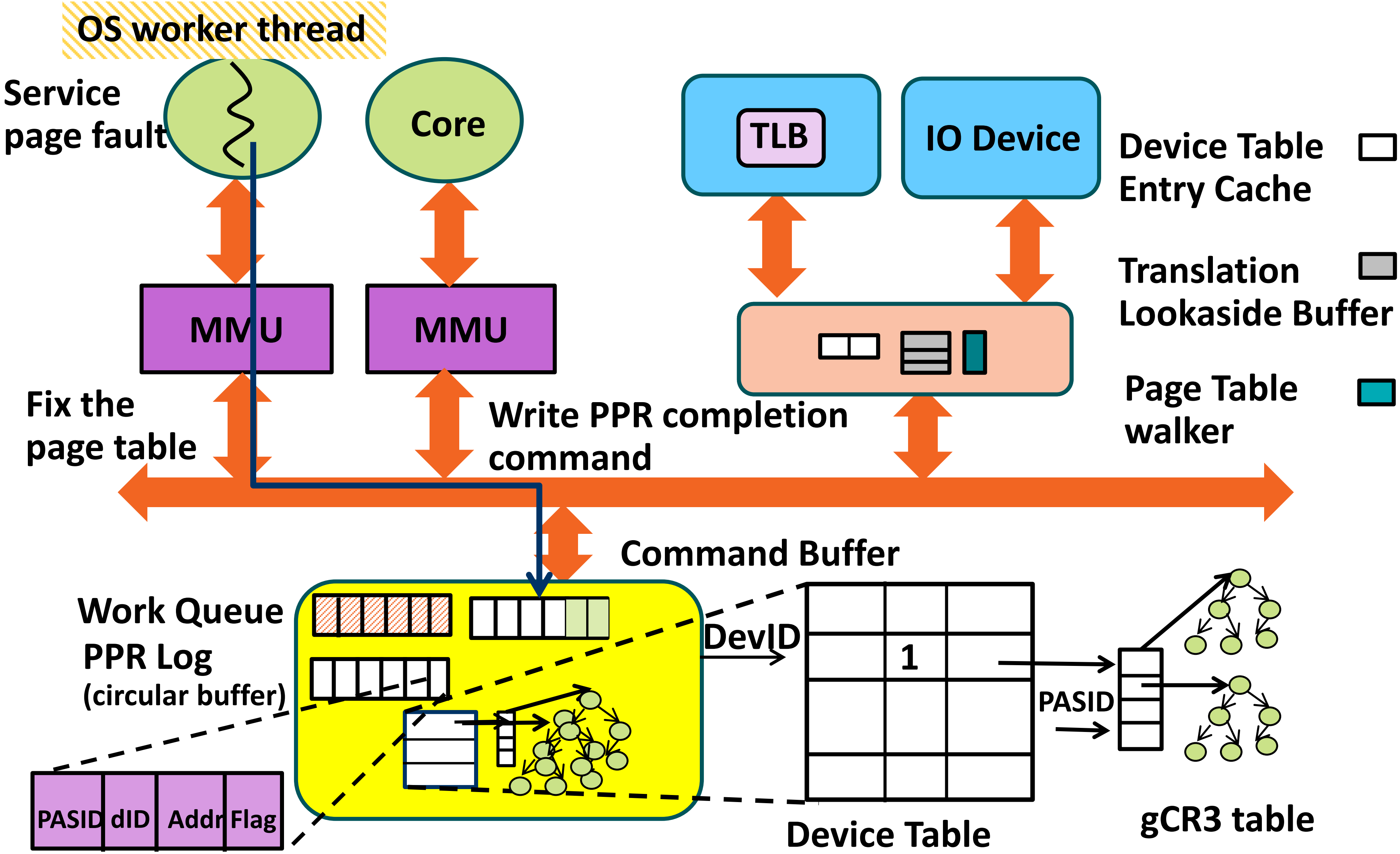
SERVICING DEVICE PAGE FAULT



*PPR= Page Peripheral Request

ENABLING DEMAND PAGING FROM DEVICE

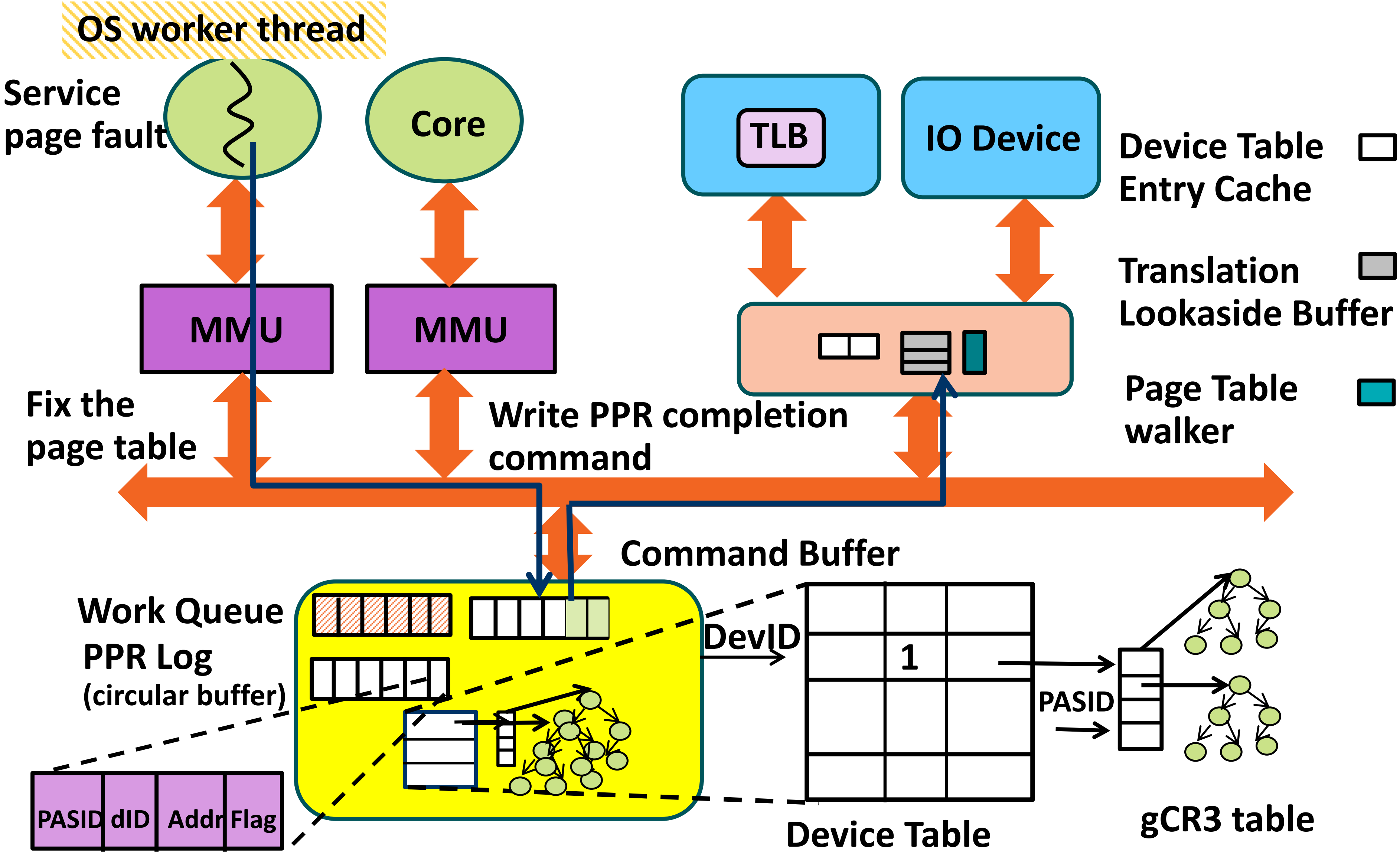
SERVICING DEVICE PAGE FAULT



*PPR= Page Peripheral Request

ENABLING DEMAND PAGING FROM DEVICE

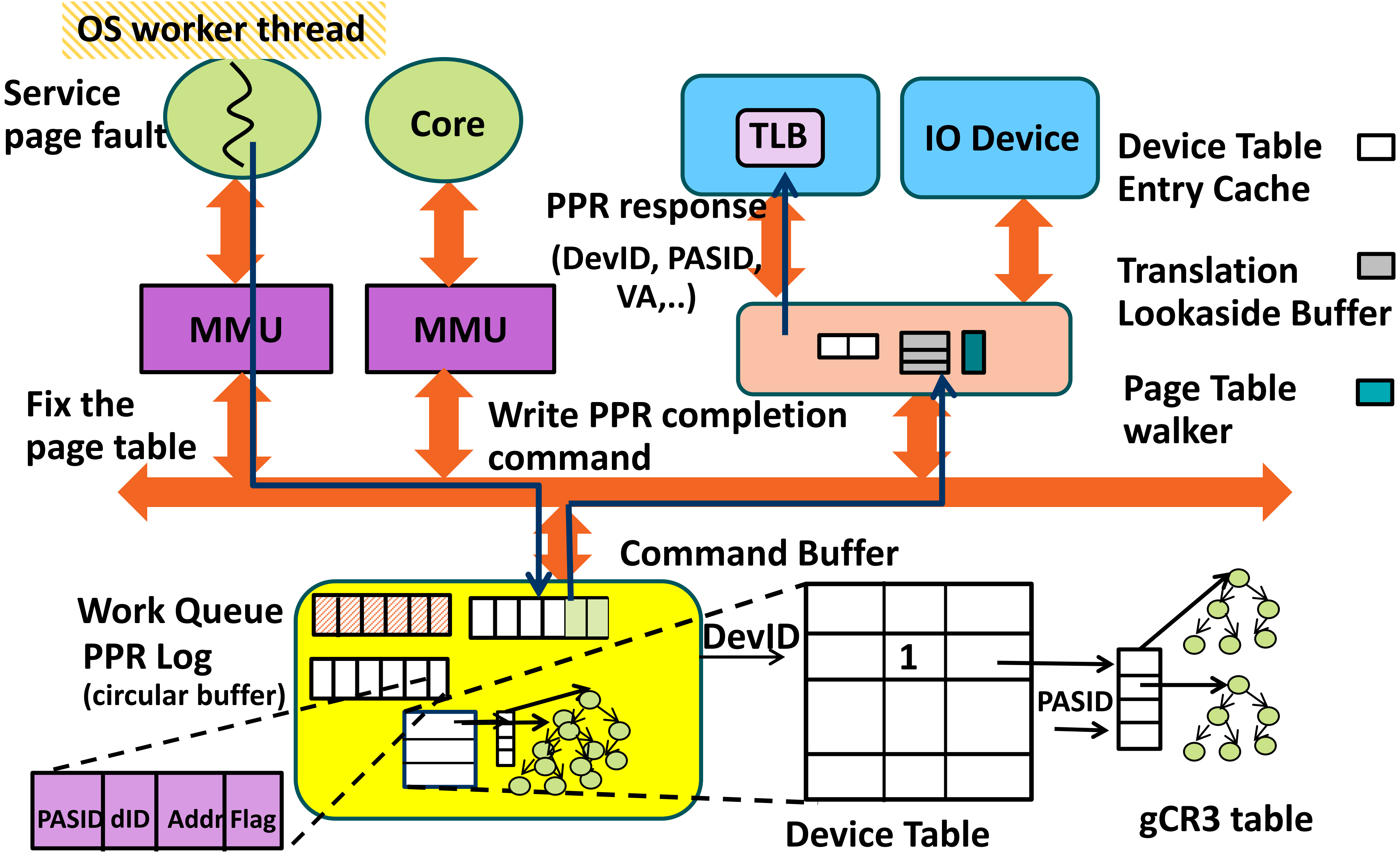
SERVICING DEVICE PAGE FAULT



*PPR= Page Peripheral Request

ENABLING DEMAND PAGING FROM DEVICE

SERVICING DEVICE PAGE FAULT



OS worker thread

Service page fault

Core

MMU

MMU

TLB

IO Device

PPR response

(DevID, PASID, VA,..)

Fix the page table

Write PPR completion command

Command Buffer

Work Queue

PPR Log (circular buffer)

PASID | dID | Addr | Flag

DevID

1

PASID

Device Table

gCR3 table

Device Table Entry Cache □

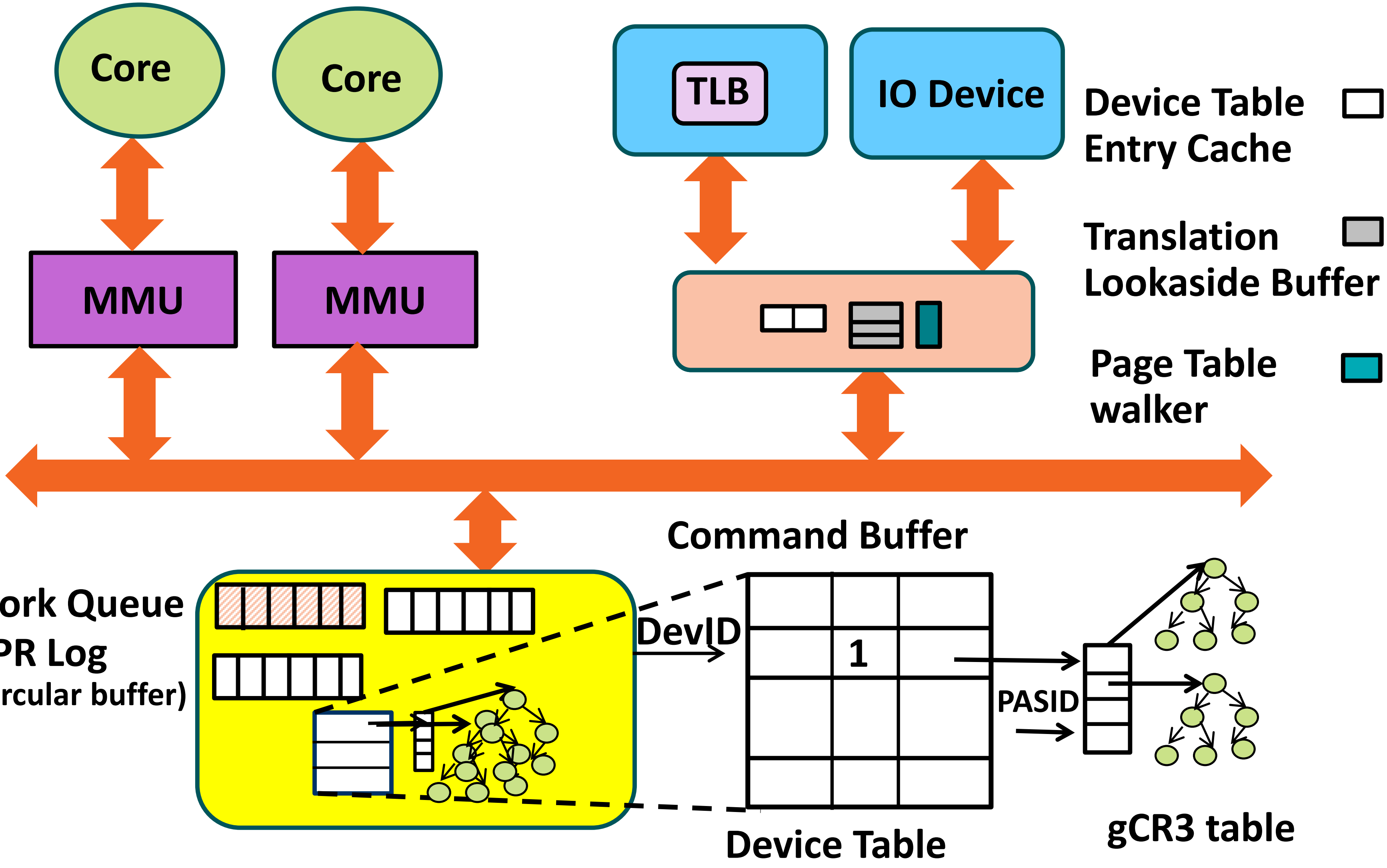
Translation Lookaside Buffer ▨

Page Table walker 🟦

*PPR= Page Peripheral Request

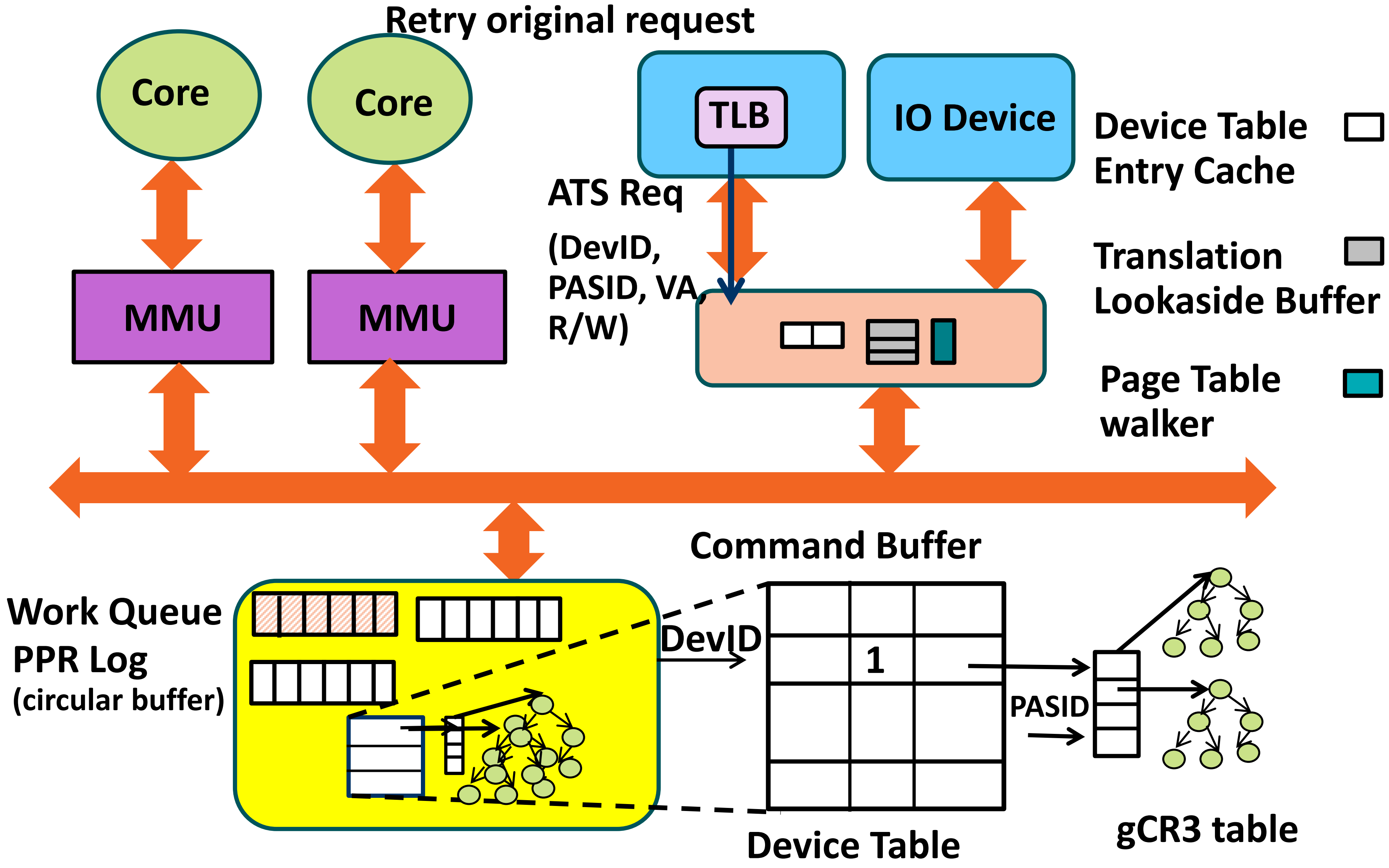
ENABLING DEMAND PAGING FROM DEVICE

SERVICING DEVICE PAGE FAULT



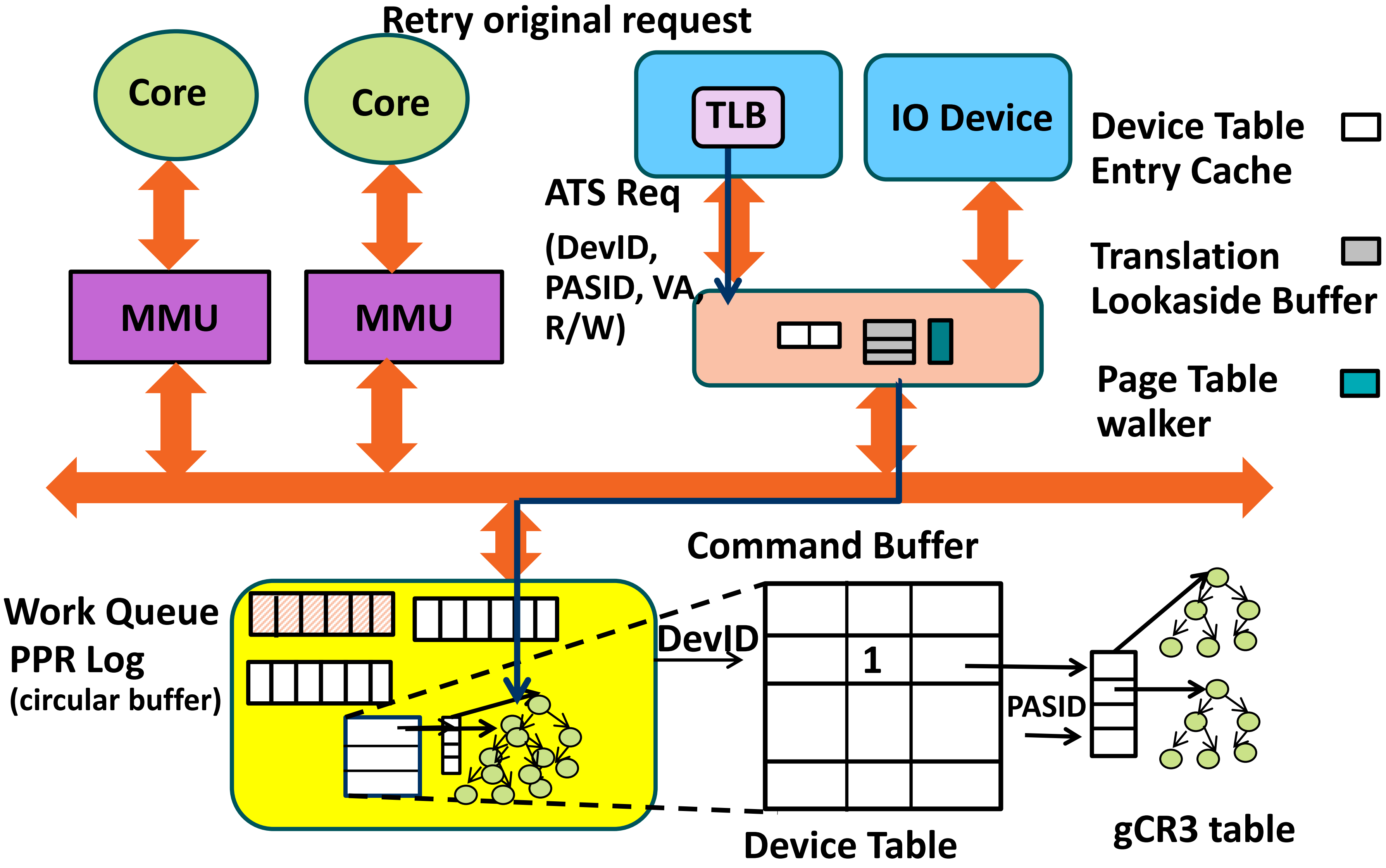
ENABLING DEMAND PAGING FROM DEVICE

SERVICING DEVICE PAGE FAULT



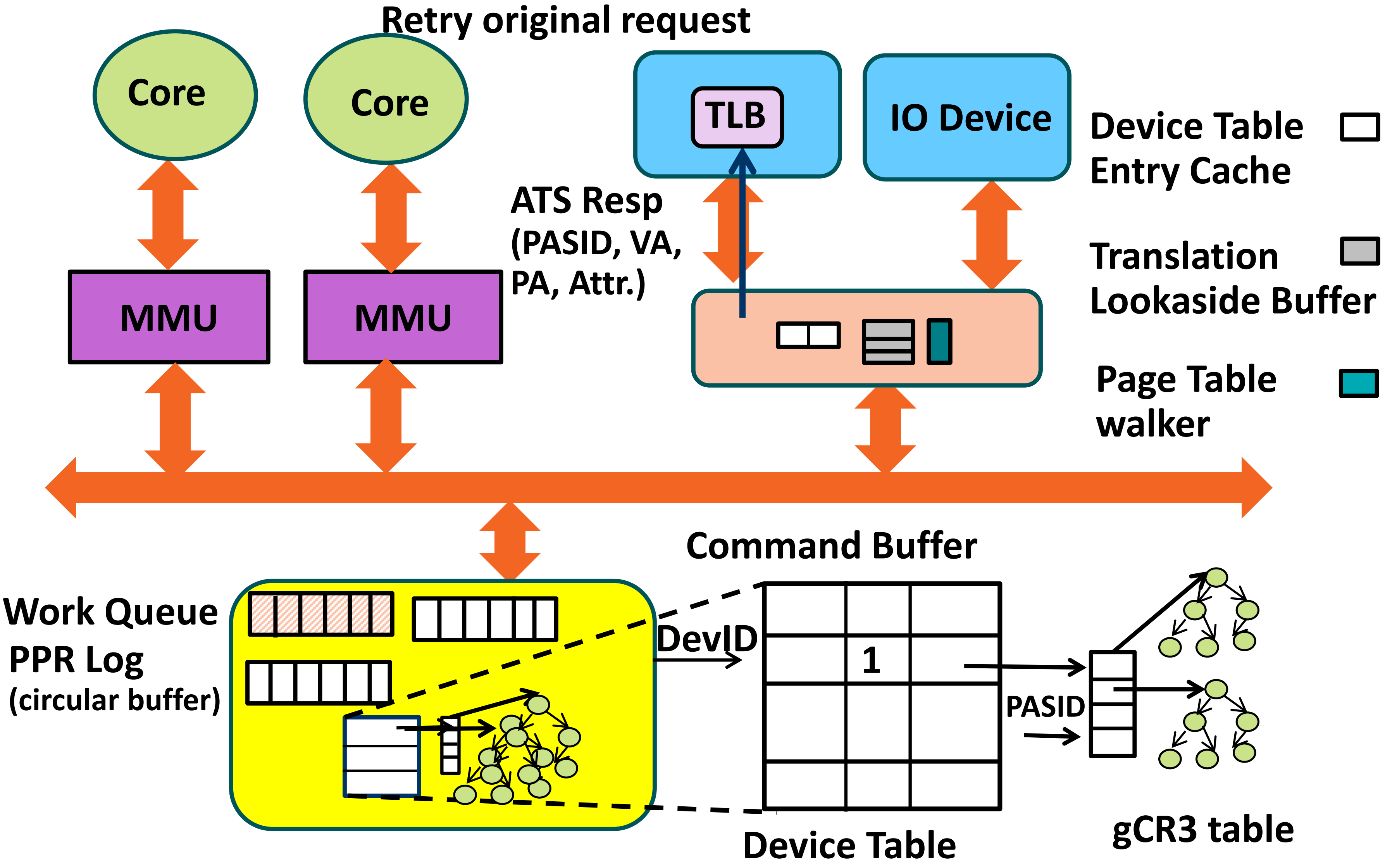
ENABLING DEMAND PAGING FROM DEVICE

SERVICING DEVICE PAGE FAULT



ENABLING DEMAND PAGING FROM DEVICE

SERVICING DEVICE PAGE FAULT



Retry original request

Core

Core

MMU

MMU

ATS Resp (PASID, VA, PA, Attr.)

TLB

IO Device

Device Table Entry Cache □

Translation Lookaside Buffer ▨

Page Table walker 🟦

Command Buffer

Work Queue

PPR Log (circular buffer)

DevID

1

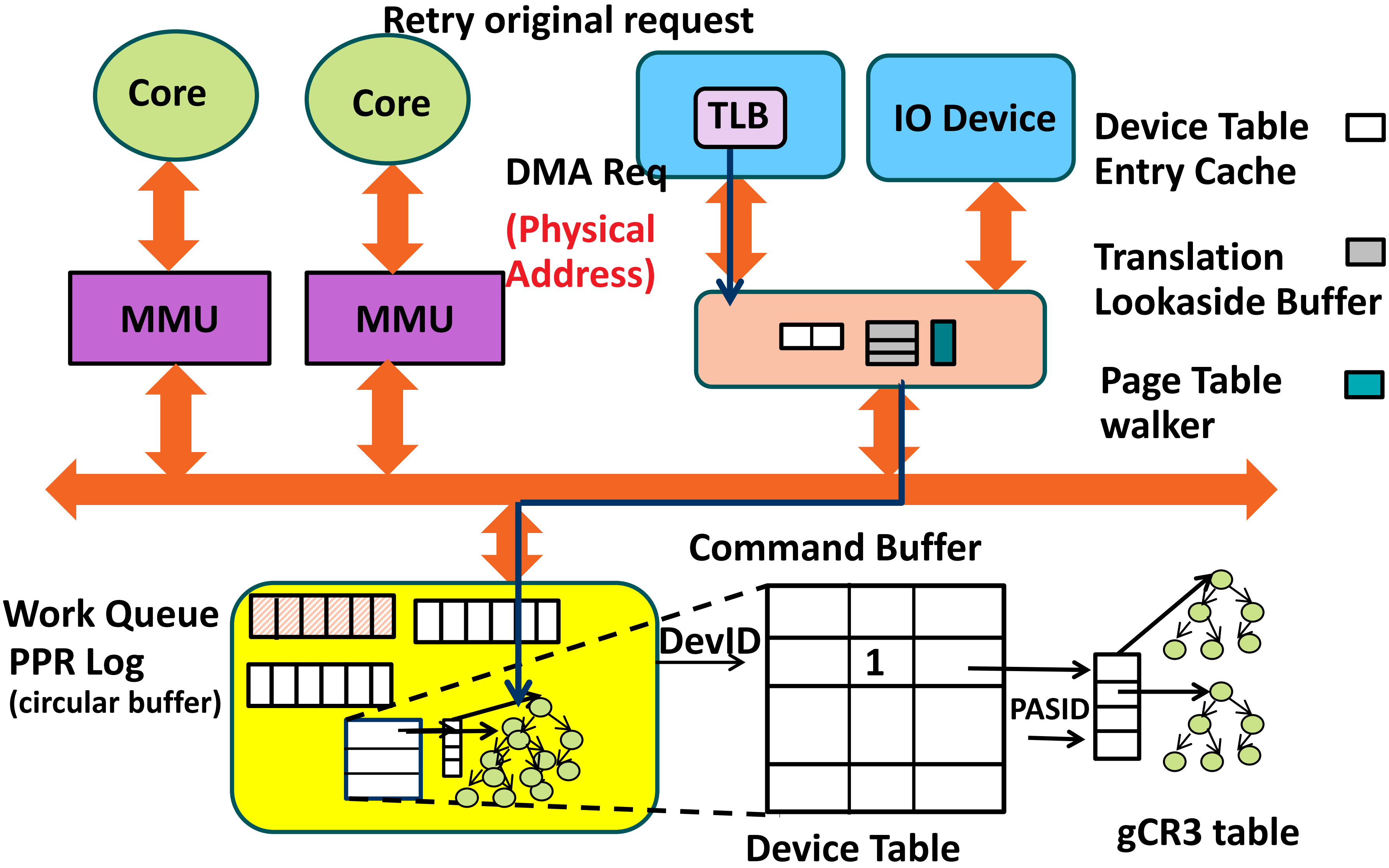
PASID

Device Table

gCR3 table

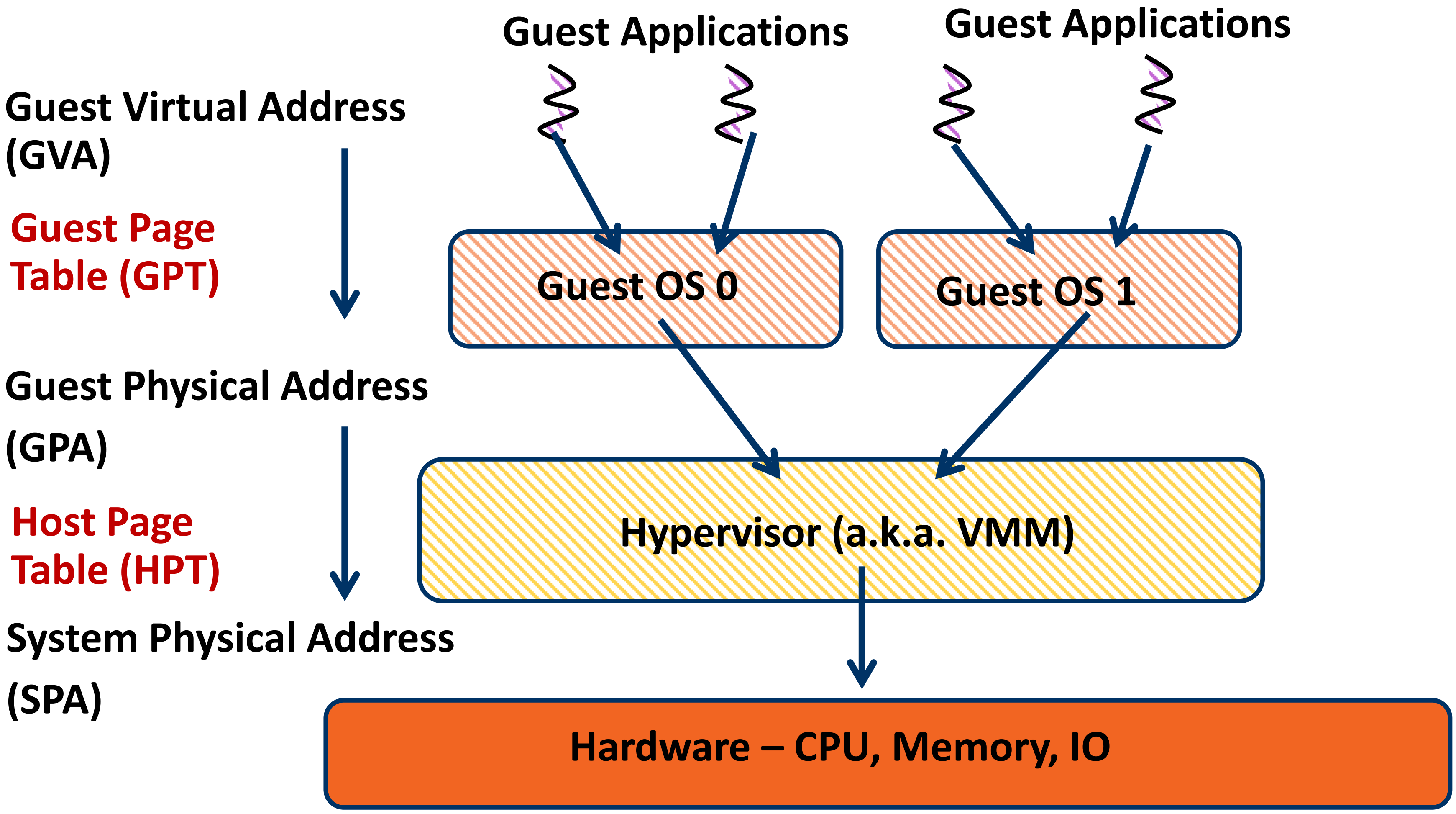
ENABLING DEMAND PAGING FROM DEVICE

SERVICING DEVICE PAGE FAULT



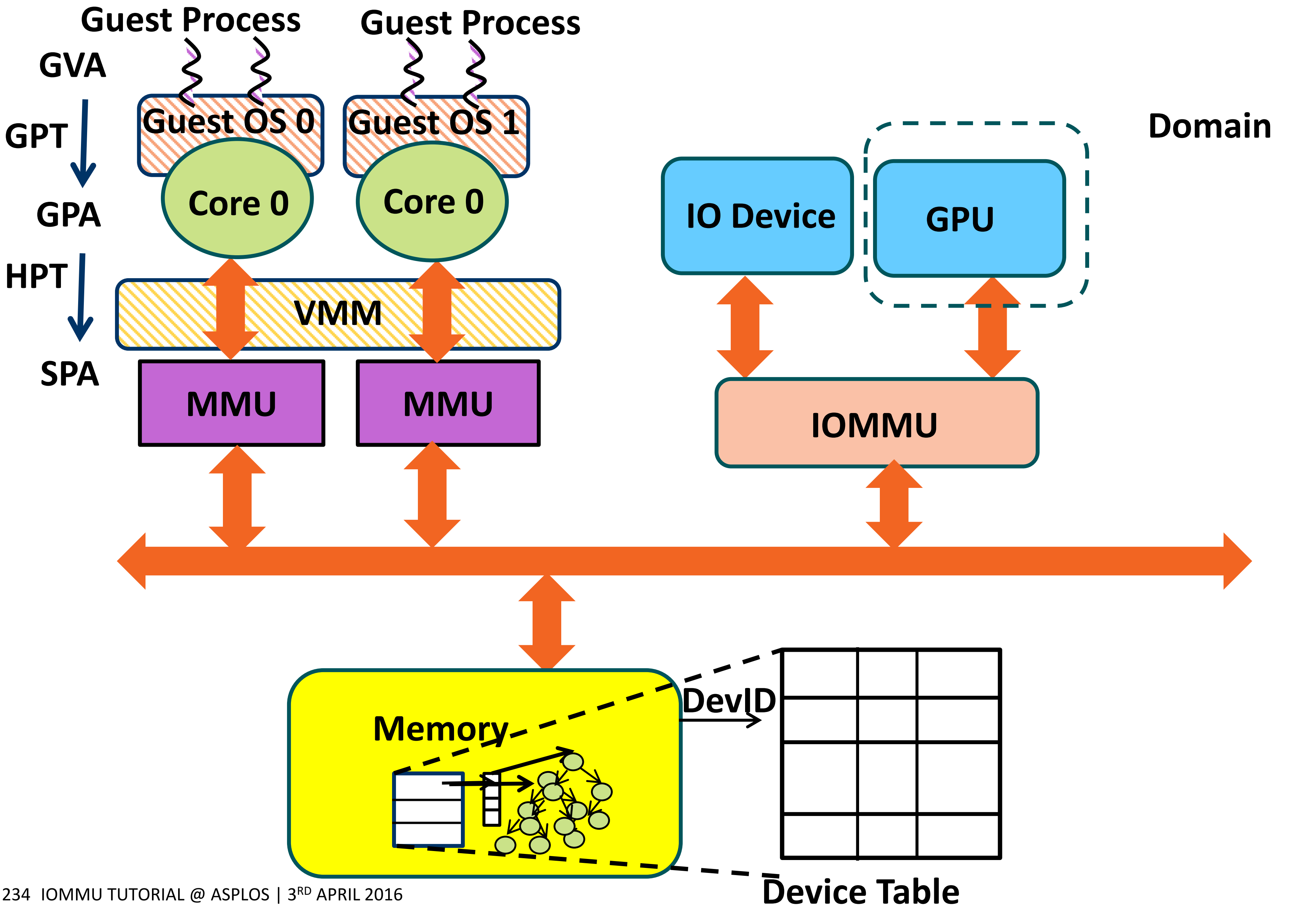
IOMMU Internals:
Nested (Two-Level) Address Translation

RECAP: ADDRESS TRANSLATION IN VIRTUALIZED SYSTEMS

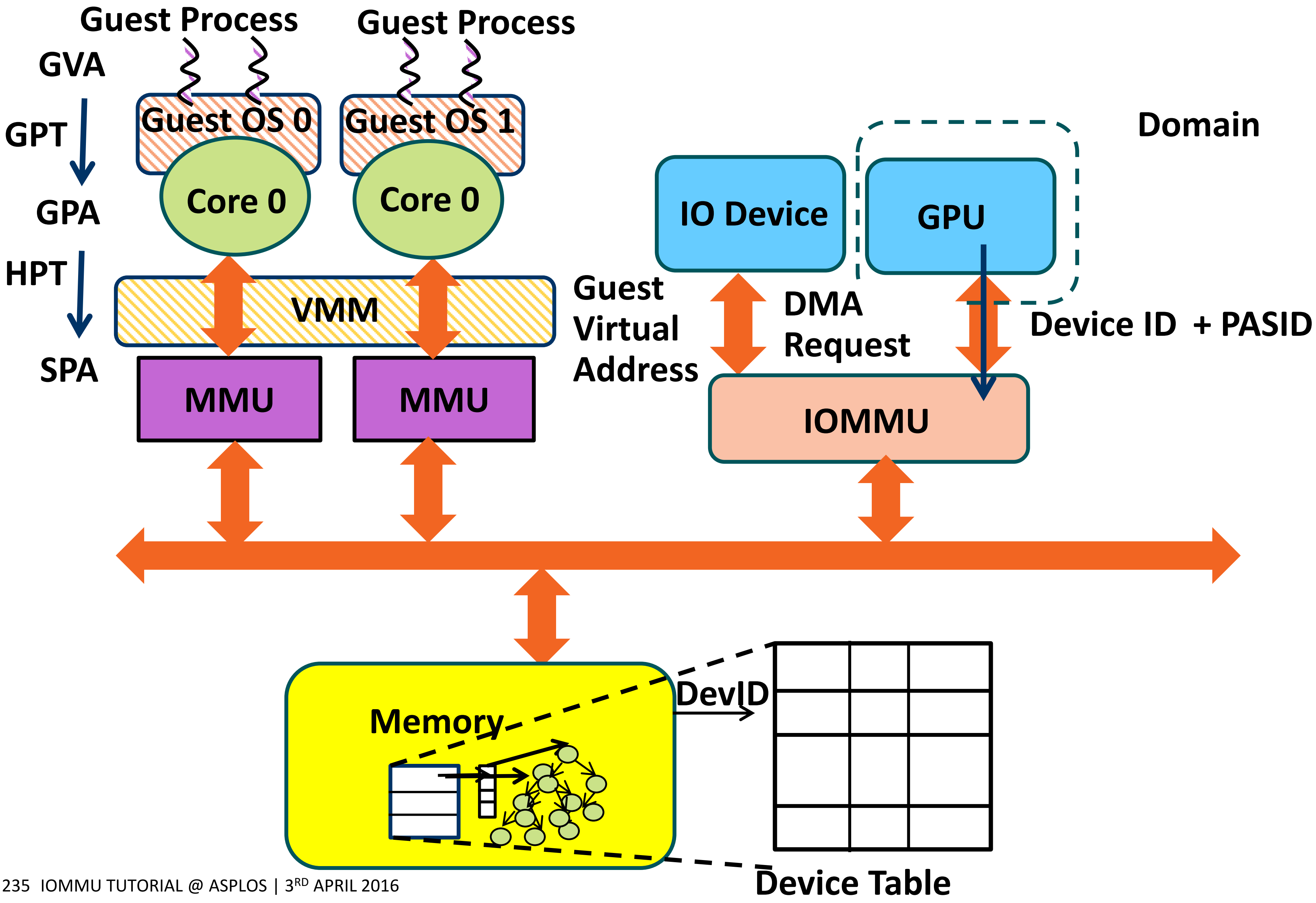


Guest OS does not have access to (system) physical address

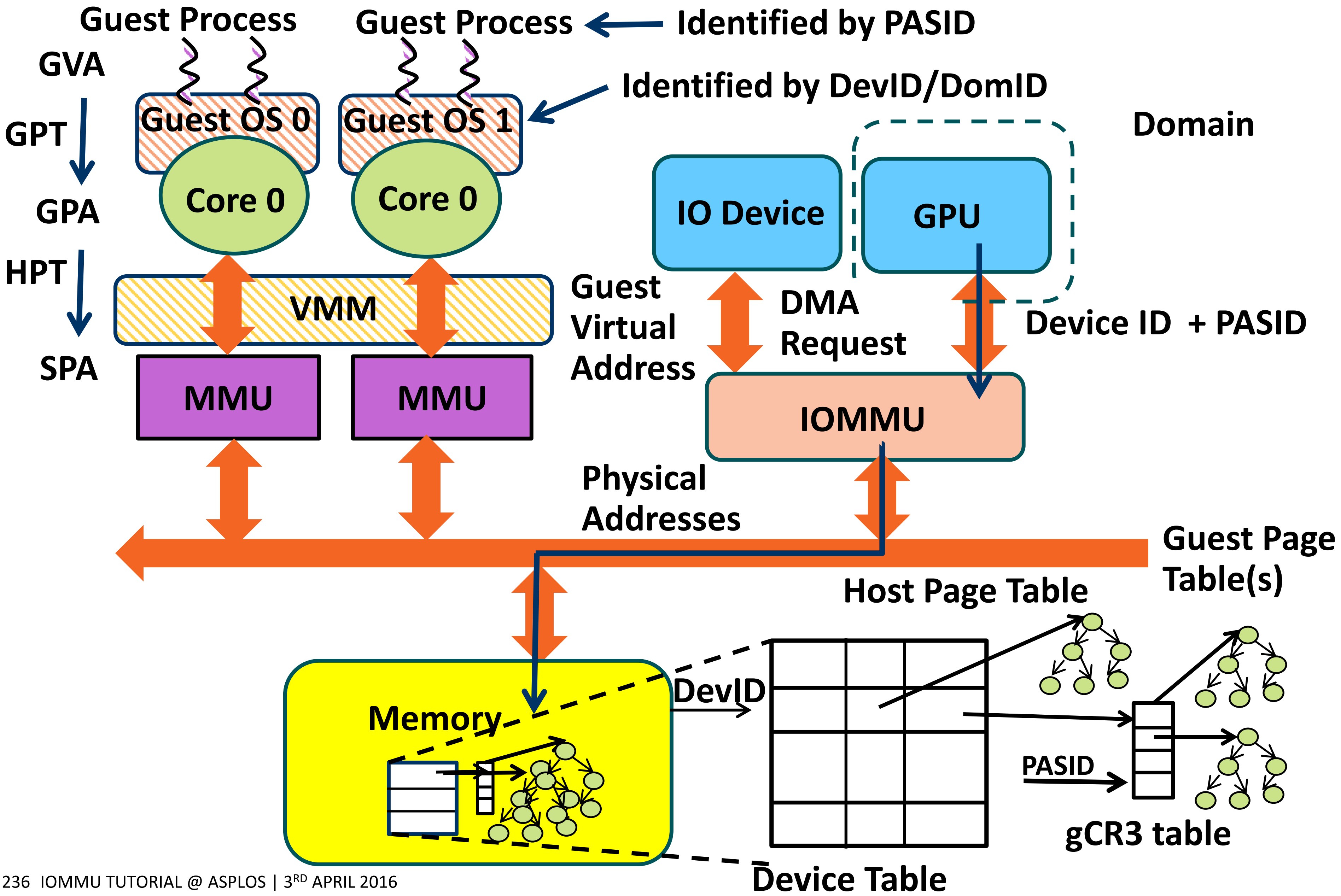
NESTED ADDRESS TRANSLATION BY IOMMU



NESTED ADDRESS TRANSLATION BY IOMMU

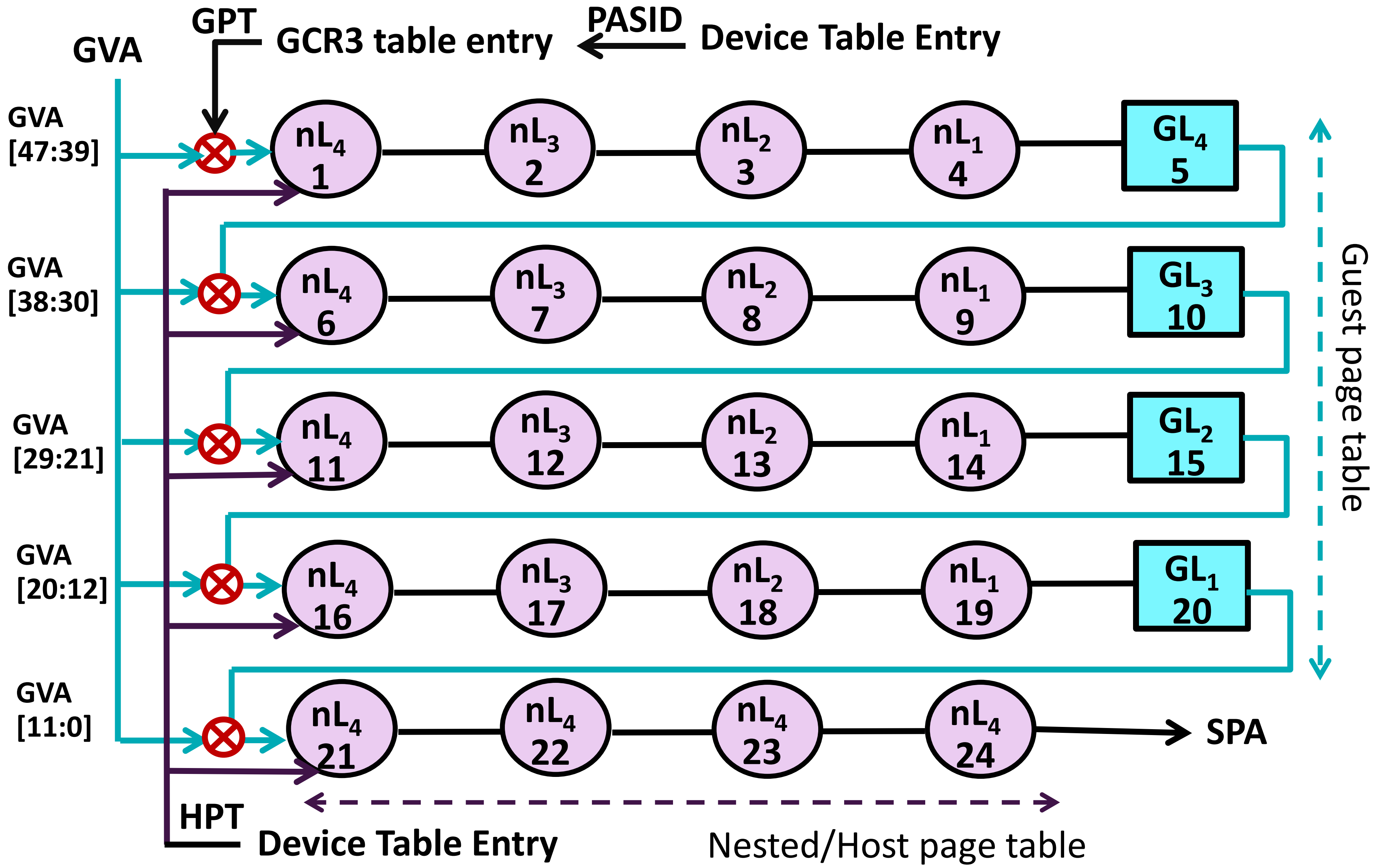


NESTED ADDRESS TRANSLATION BY IOMMU



NESTED ADDRESS TRANSLATION BY IOMMU

AMD



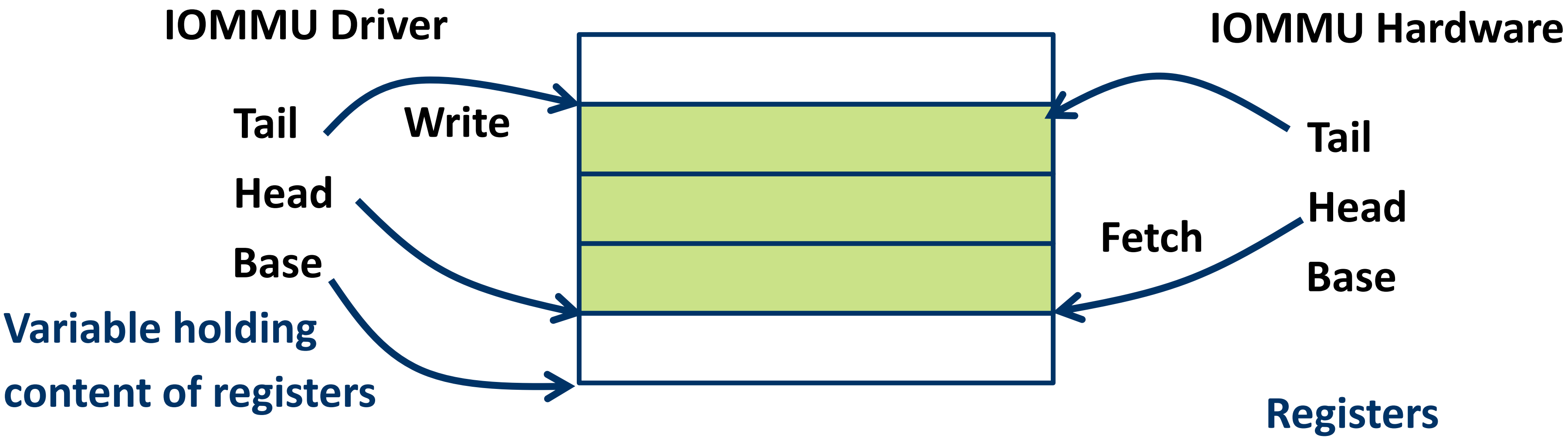
IOMMU Internals:
Sending Commands to IOMMU

COMMANDS TO IOMMU

- IOMMU Driver (running on CPU) issues commands to IOMMU
  - e.g., Invalidate IOMMU TLB Entry, Invalidate IOTLB Entry
  - e.g., Invalidate Device Table Entry
  - e.g., Complete PPR, Completion Wait , etc.
- Issued via **Command Buffer**
  - Memory resident circular buffer
  - MMIO registers: Base, Head, and Tail register

COMMANDS TO IOMMU

- IOMMU Driver (running on CPU) issues commands to IOMMU
  - e.g., Invalidate IOMMU TLB Entry, Invalidate IOTLB Entry
  - e.g., Invalidate Device Table Entry
  - e.g., Complete PPR, Completion Wait , etc.
- Issued via **Command Buffer**
  - Memory resident circular buffer
  - MMIO registers: Base, Head, and Tail register

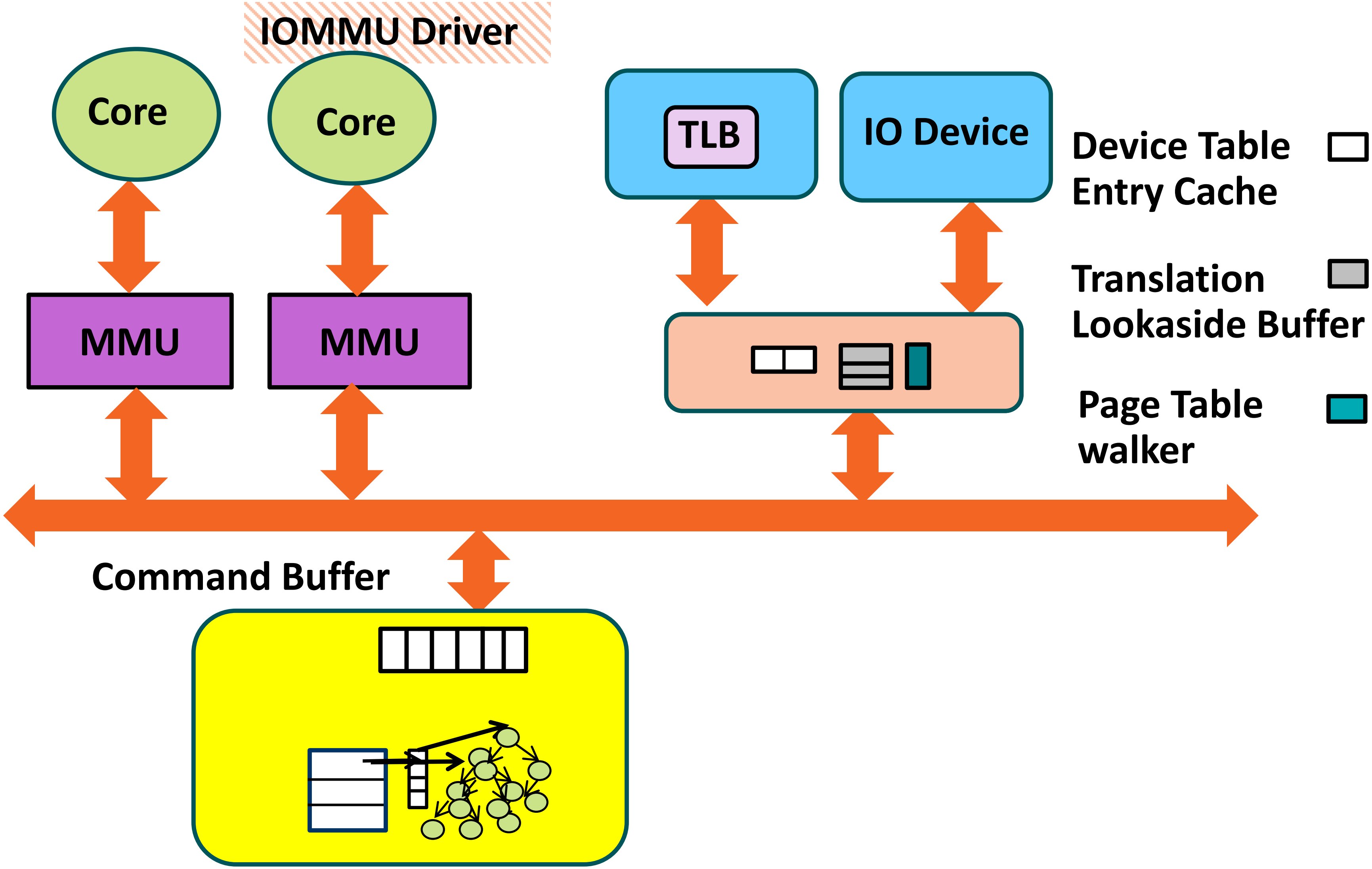


EXAMPLE: IOMMU TLB SHOOTDOWN

- IOMMU TLB Shootdown
  - Update page table information
  - Flush TLB Entry(s) containing stale information

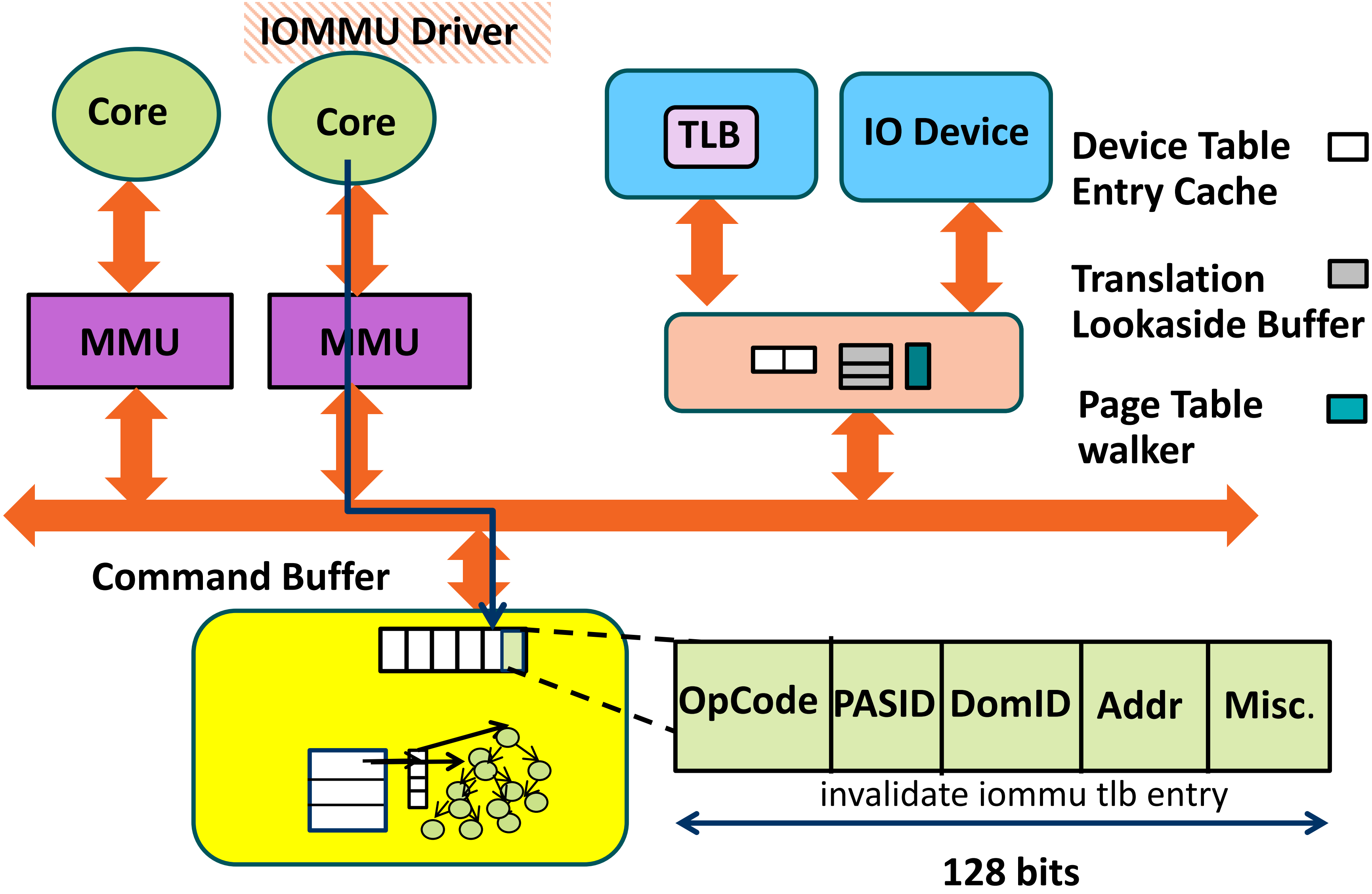
- Three steps in IOMMU TLB shootdown
  - Invalidating IOMMU TLB entry
  - Invalidating IO TLB (Device TLB) entry
  - Wait for completion

EXAMPLE: IOMMU TLB SHOOTDOWN

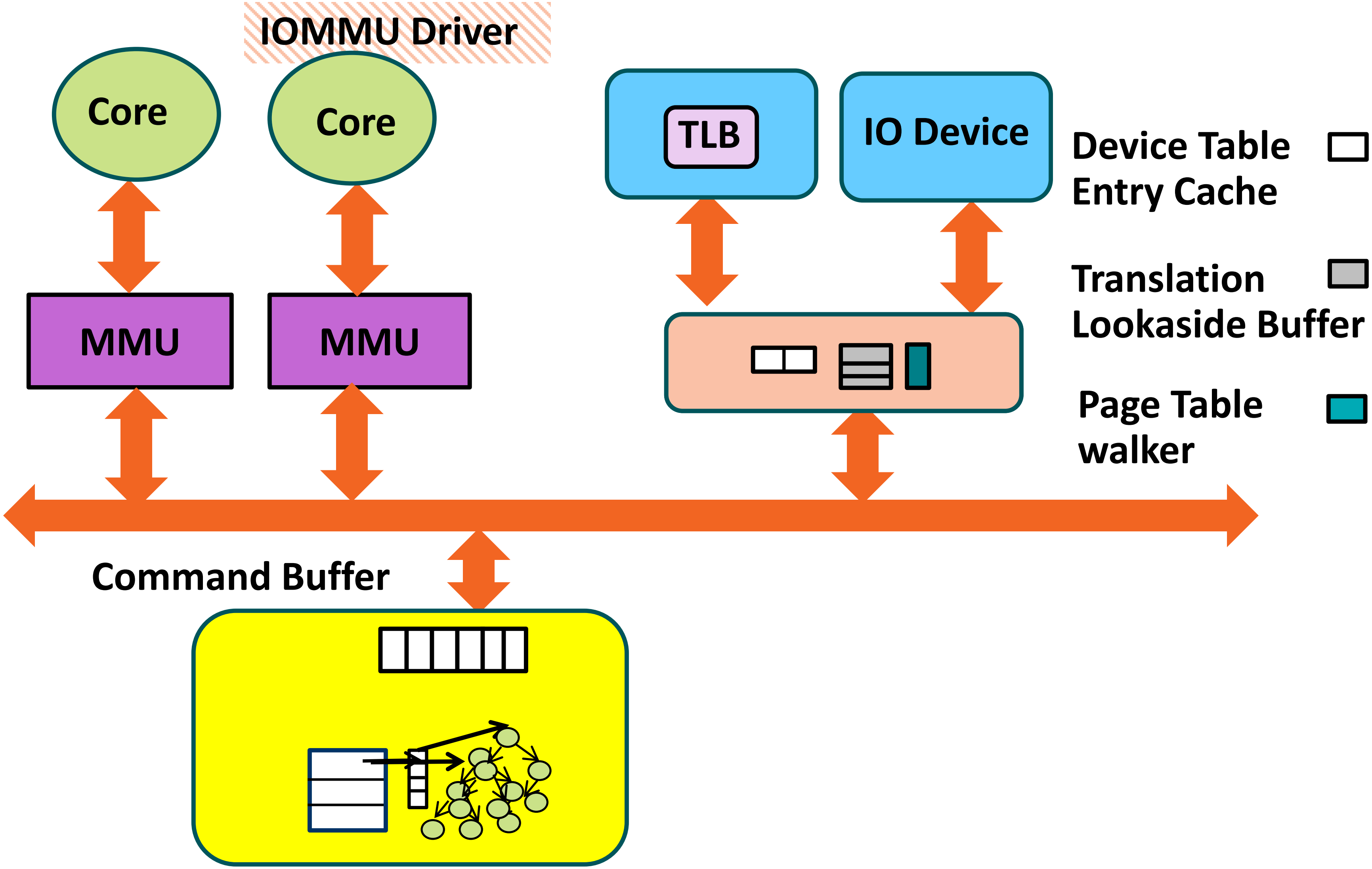


EXAMPLE: IOMMU TLB SHOOTDOWN

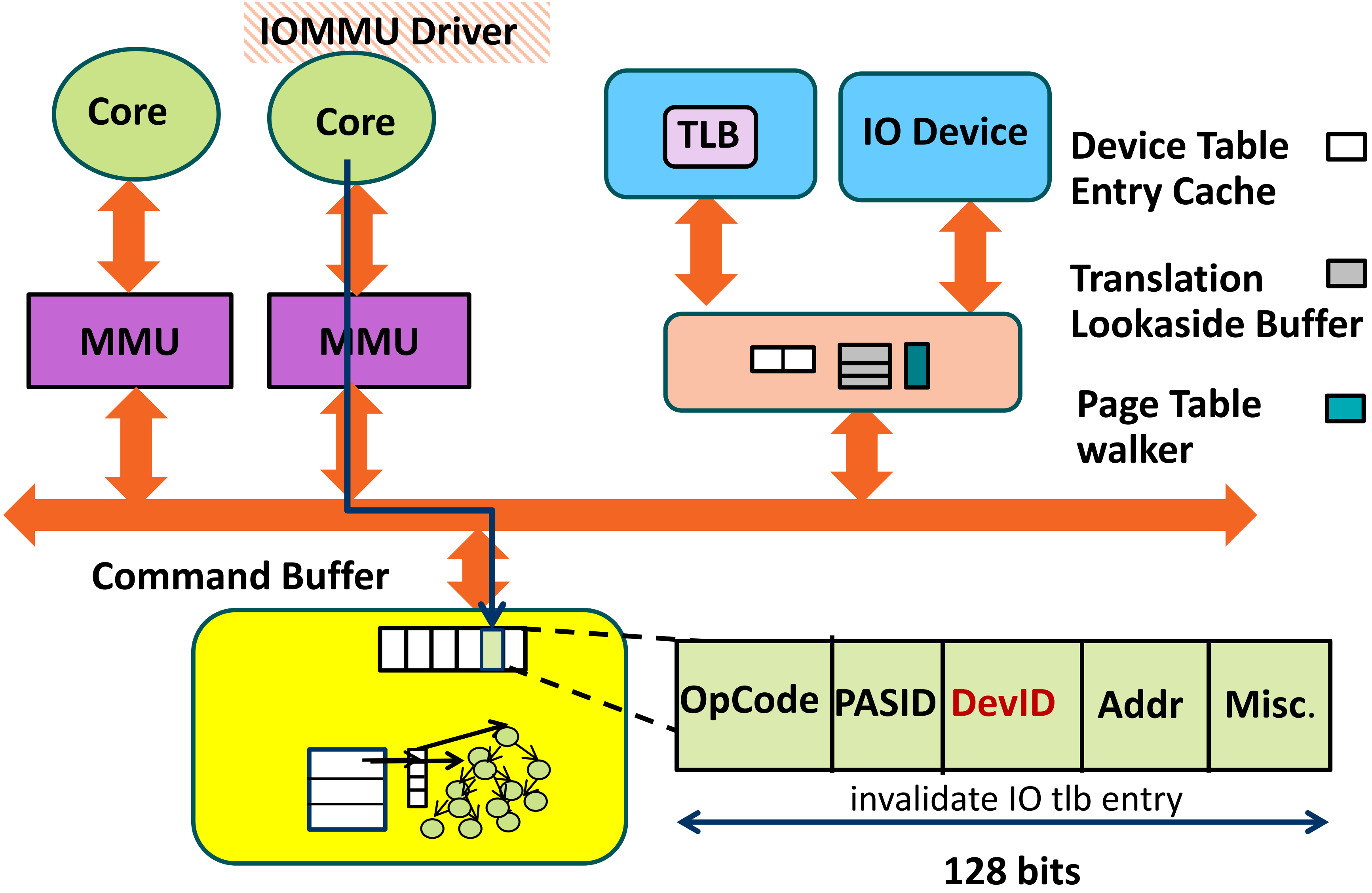
AMD



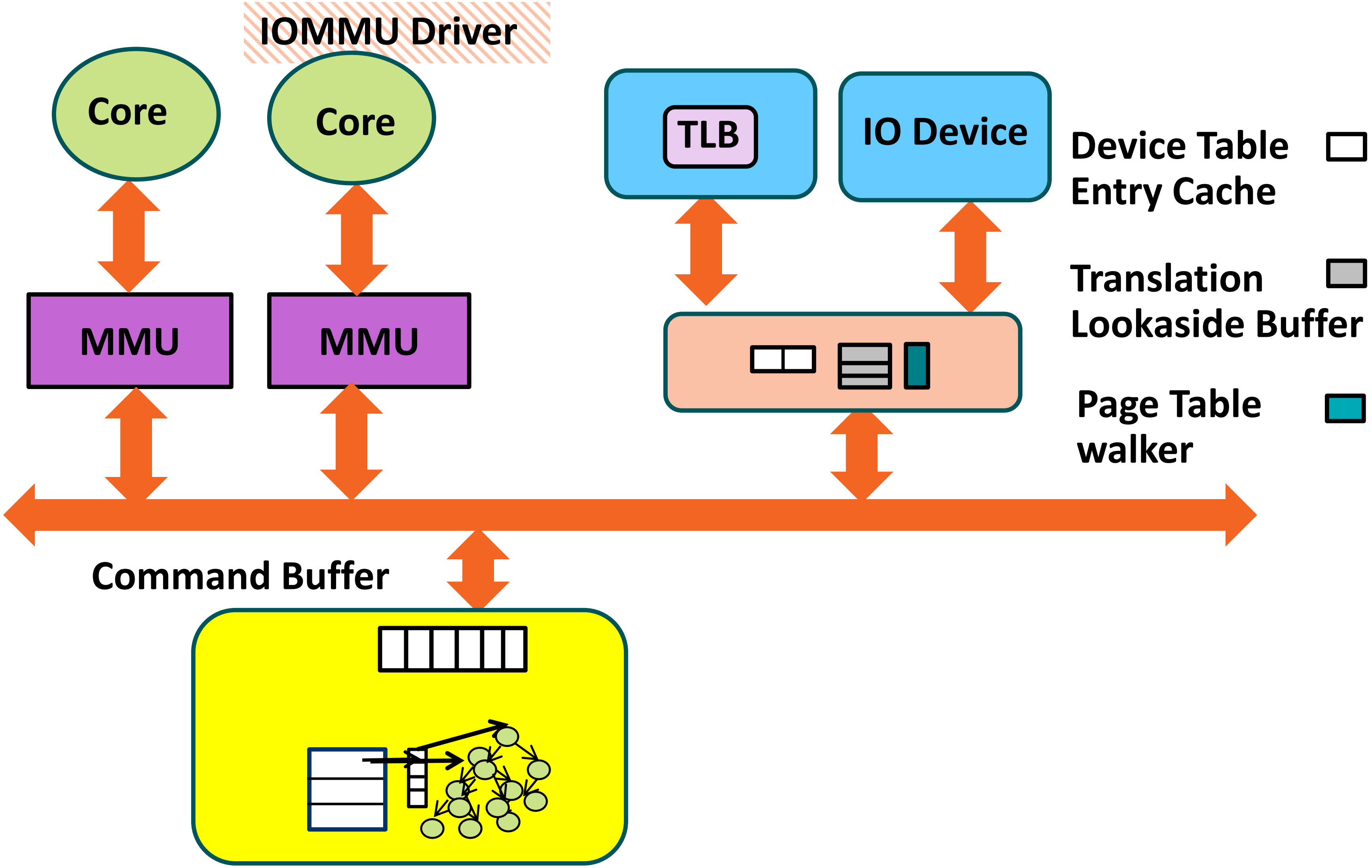
EXAMPLE: IOMMU TLB SHOOTDOWN



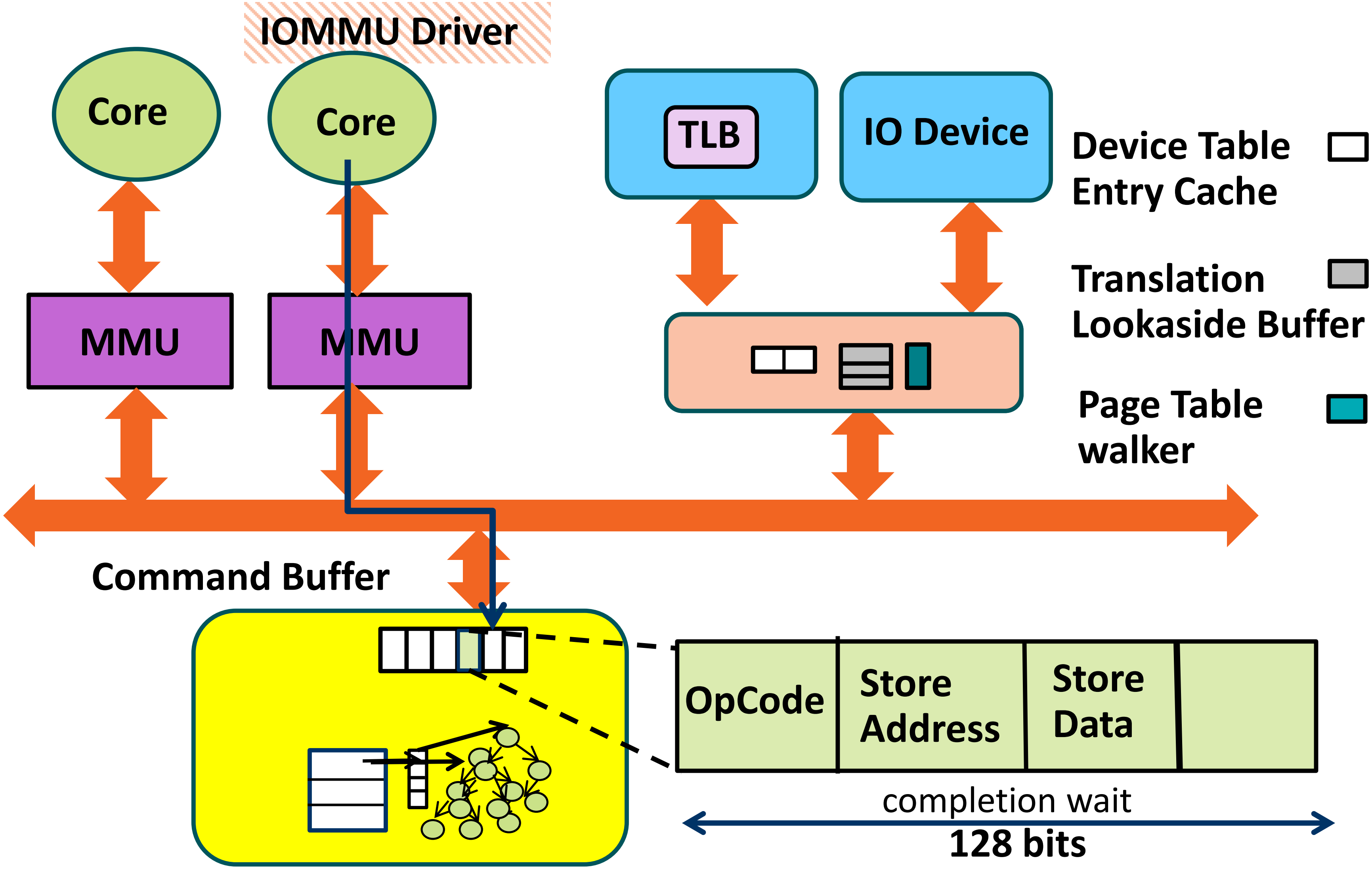
EXAMPLE: IOMMU TLB SHOOTDOWN



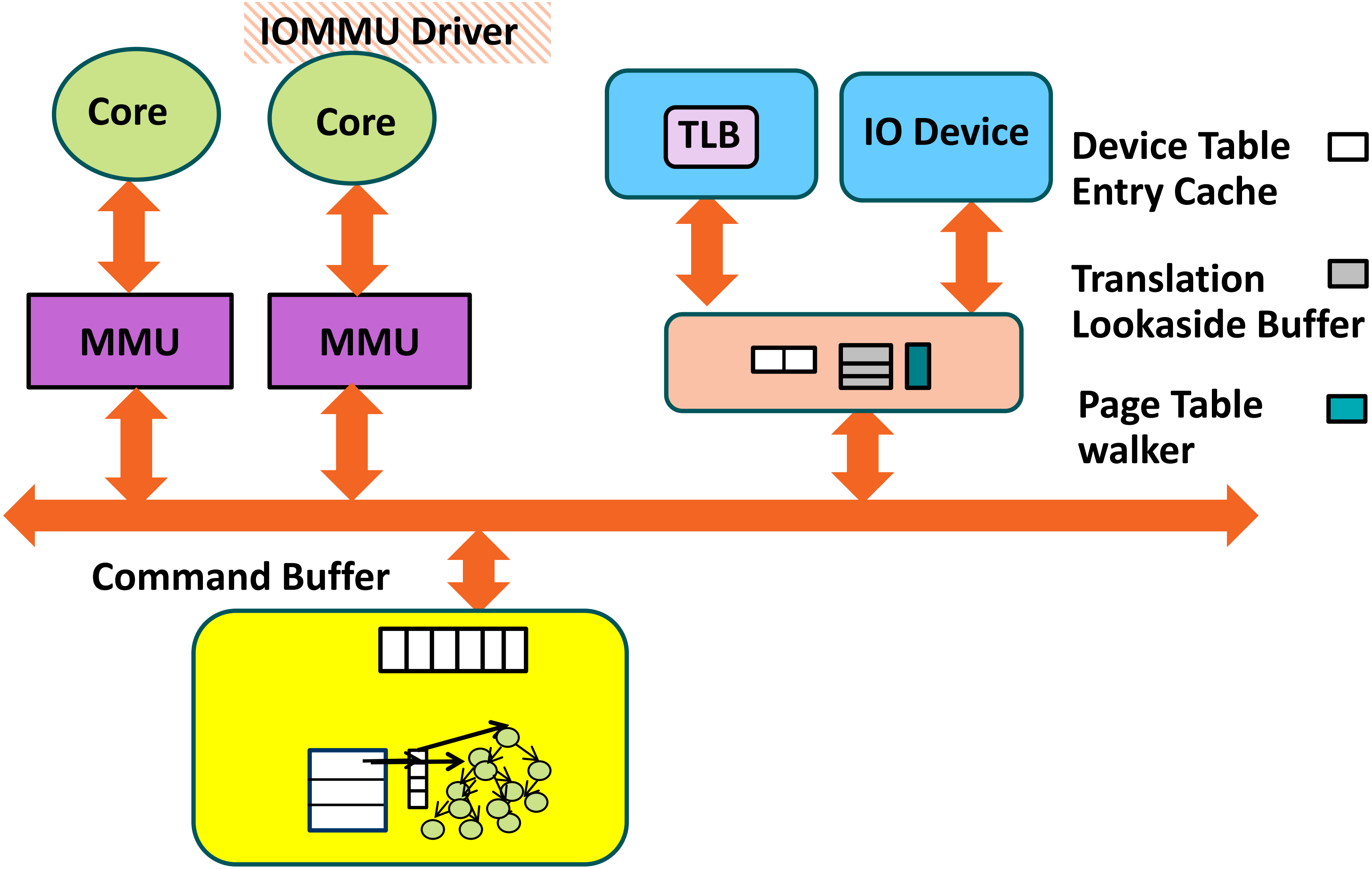
EXAMPLE: IOMMU TLB SHOOTDOWN



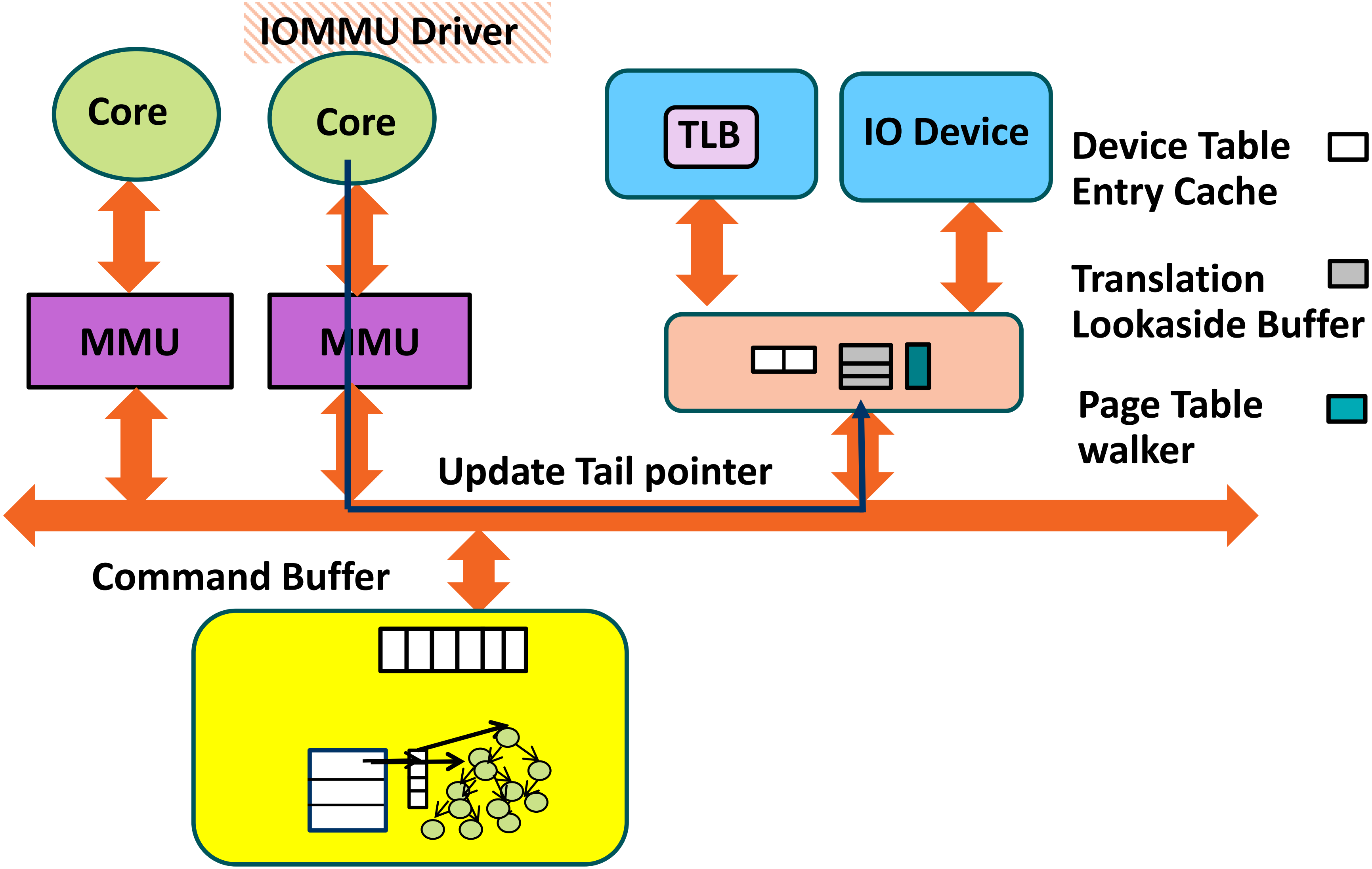
EXAMPLE: IOMMU TLB SHOOTDOWN



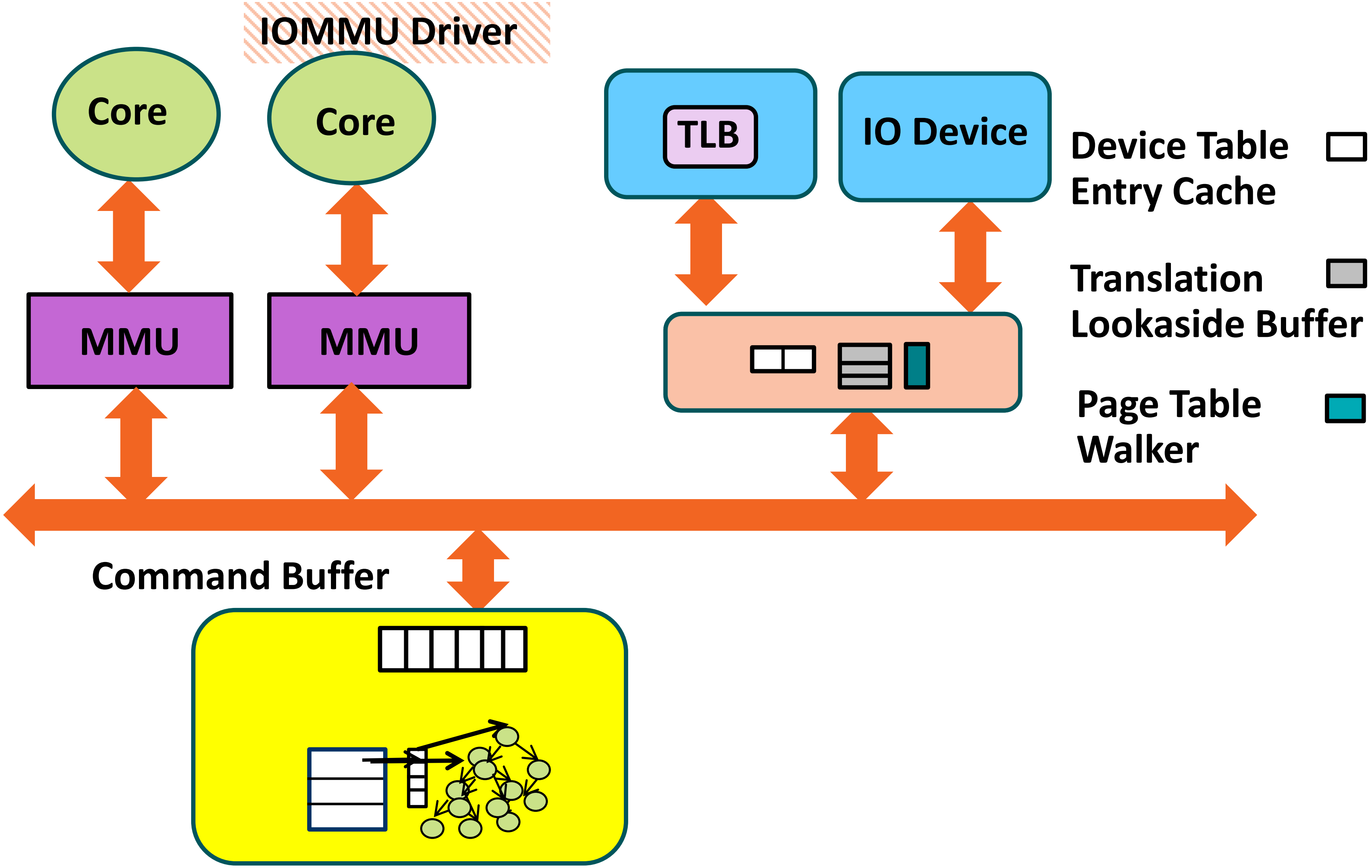
EXAMPLE: IOMMU TLB SHOOTDOWN



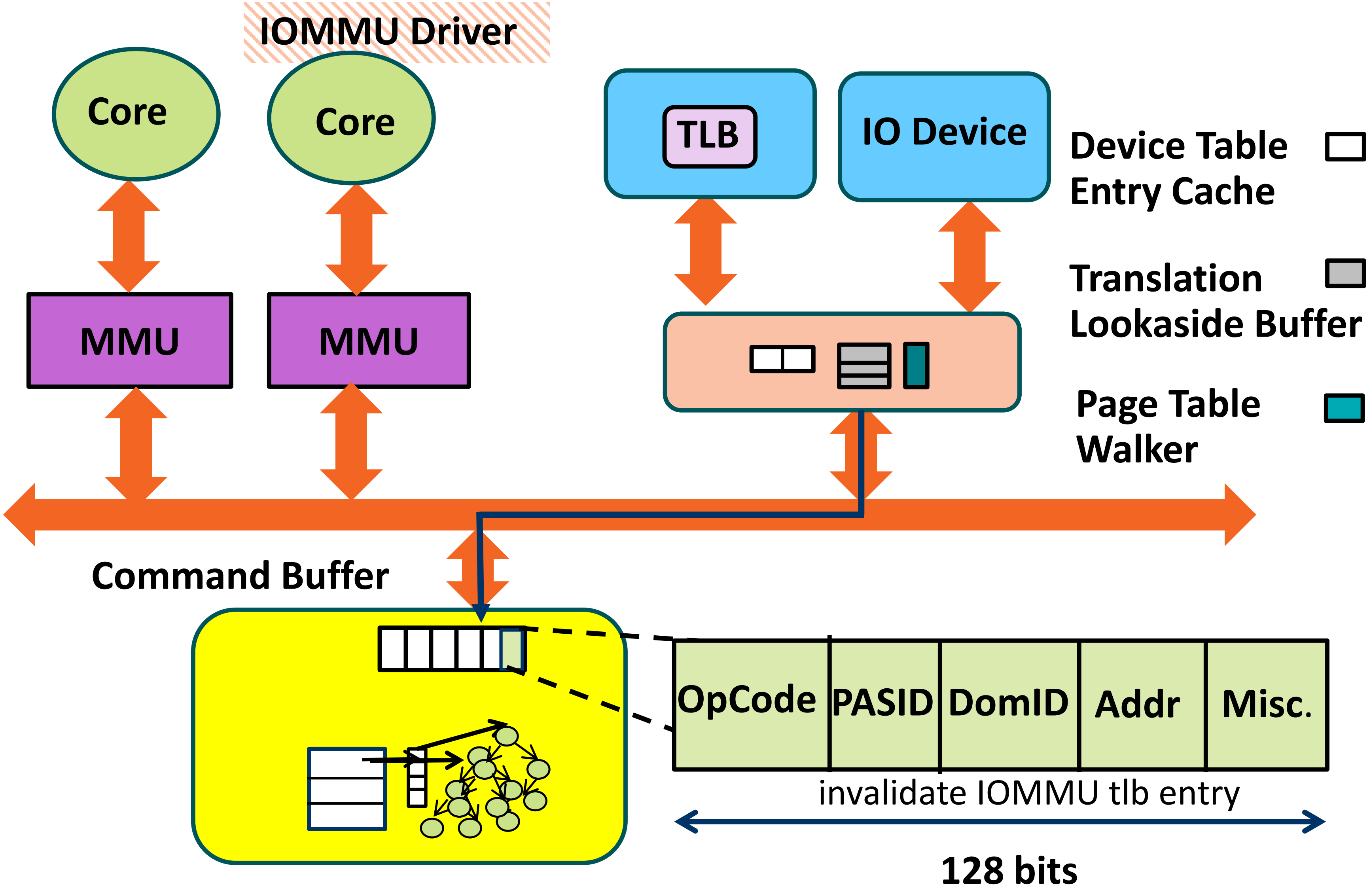
EXAMPLE: IOMMU TLB SHOOTDOWN



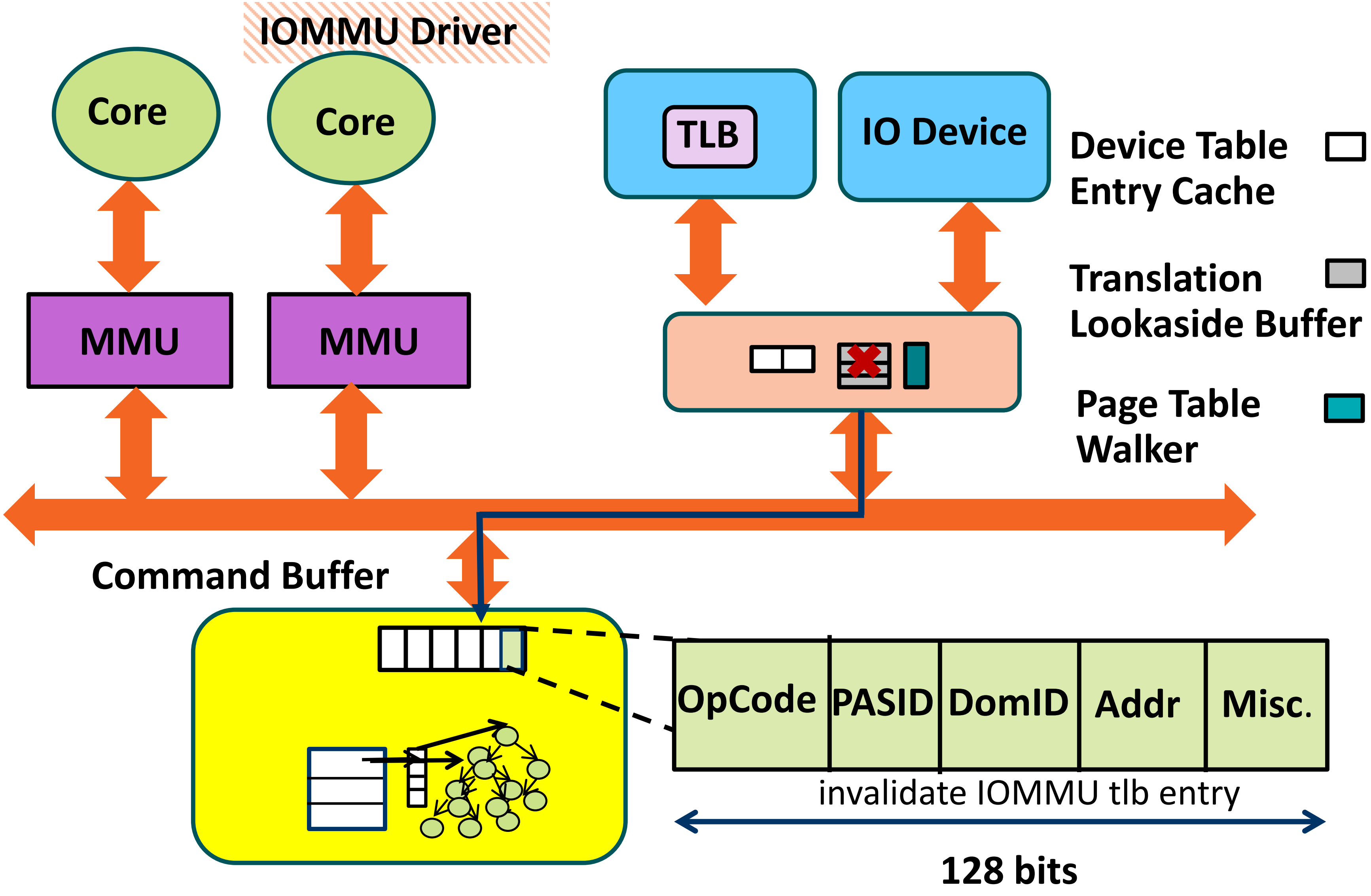
EXAMPLE: IOMMU TLB SHOOTDOWN



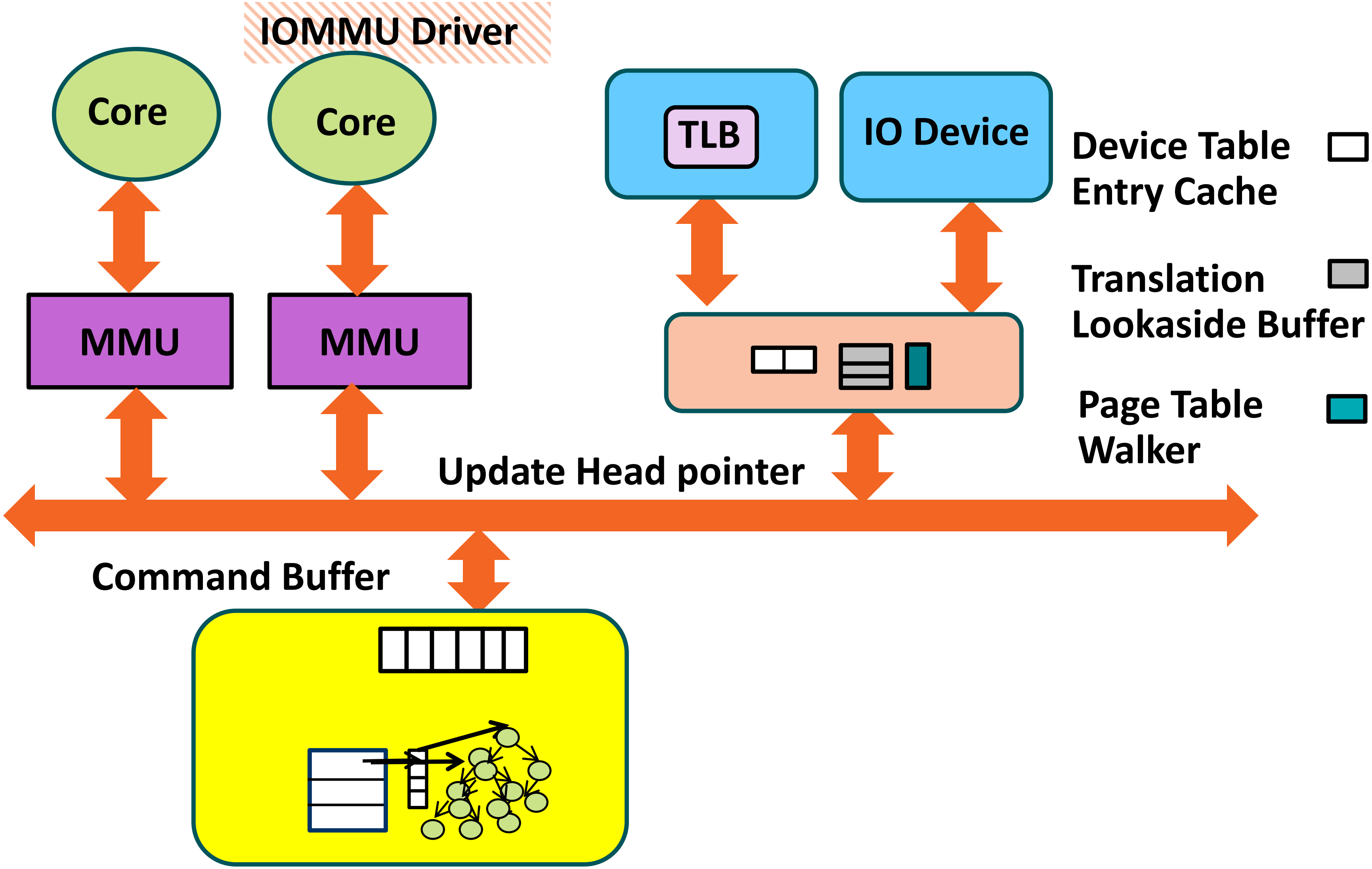
EXAMPLE: IOMMU TLB SHOOTDOWN



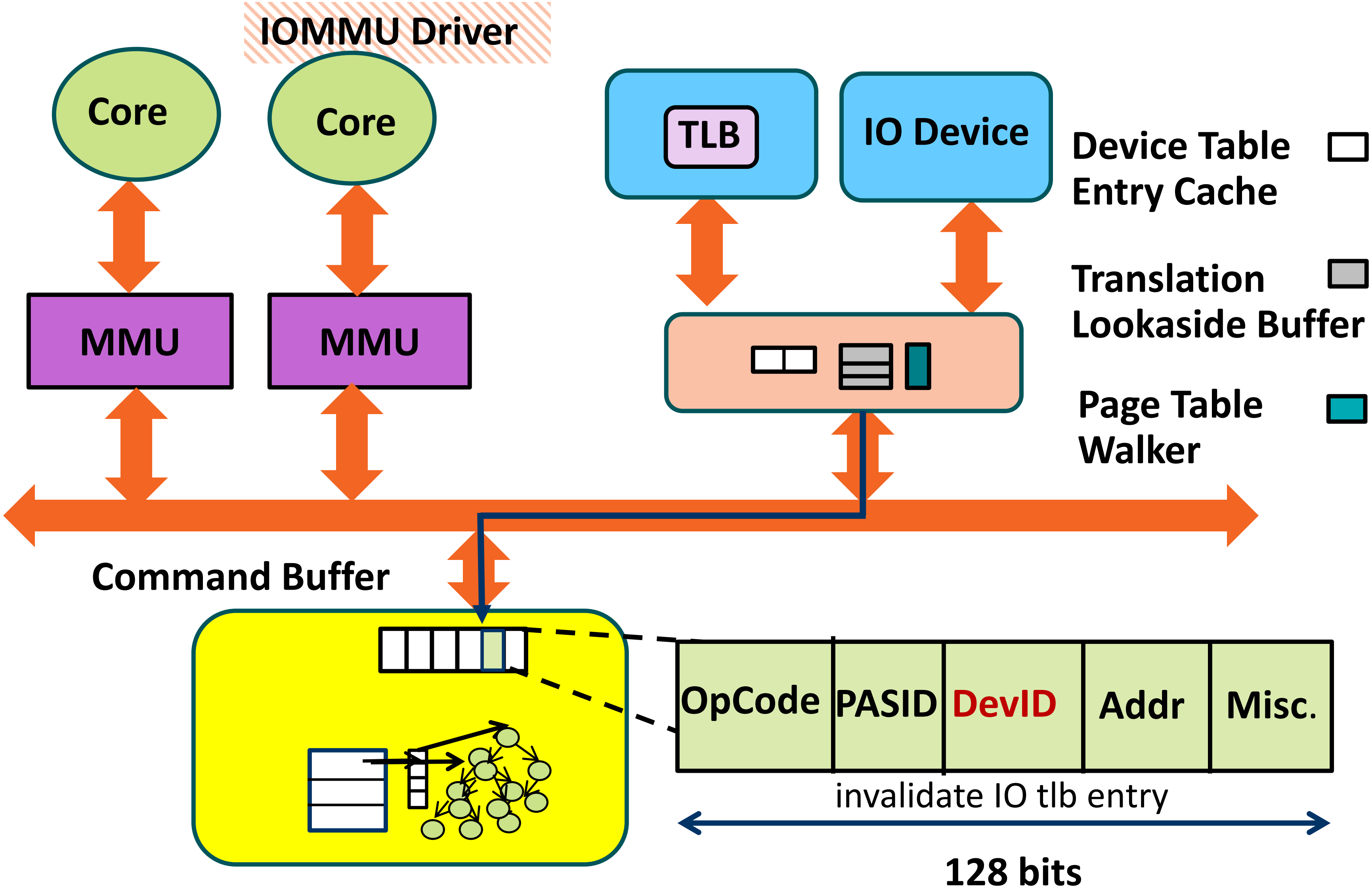
EXAMPLE: IOMMU TLB SHOOTDOWN



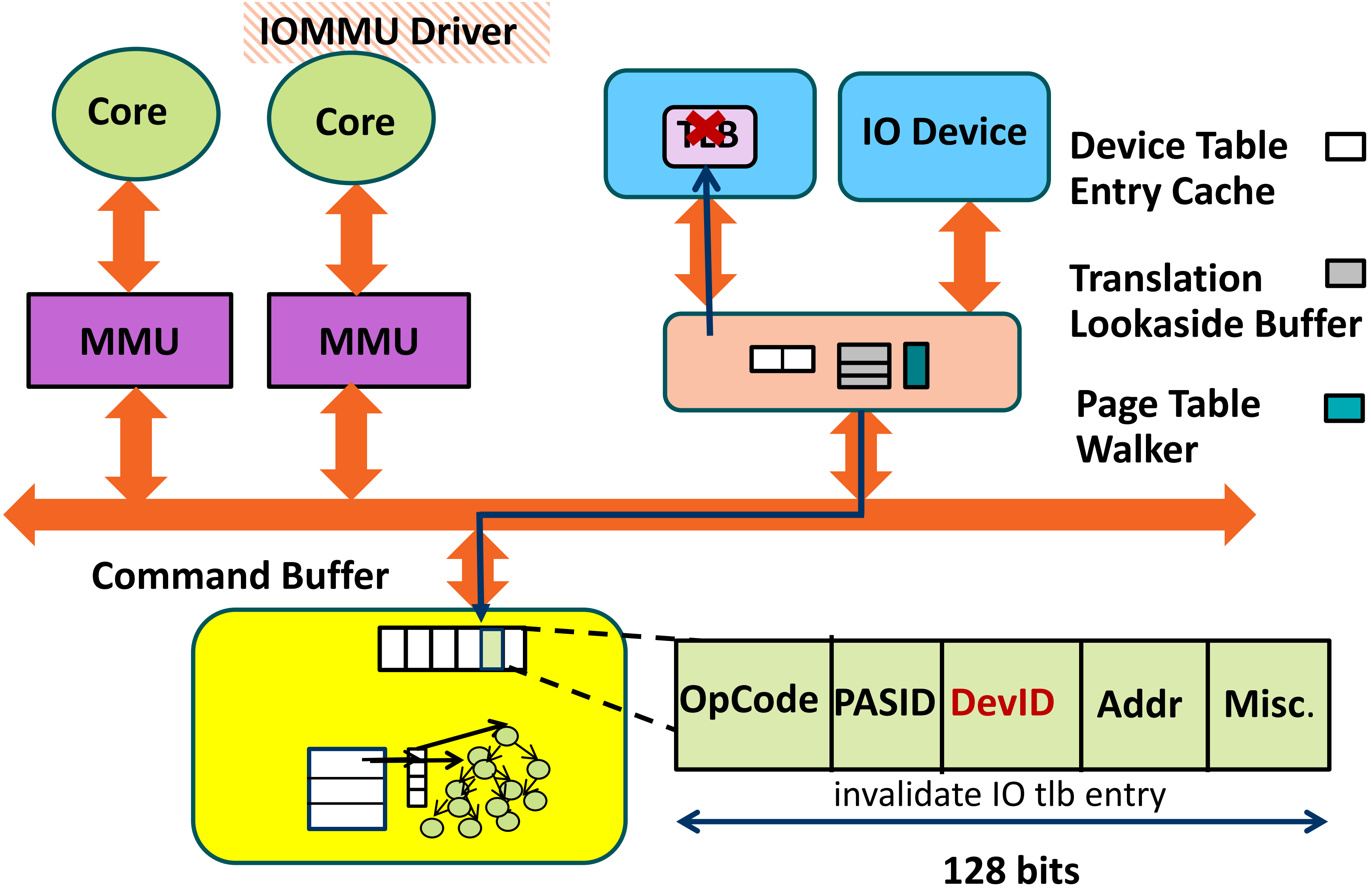
EXAMPLE: IOMMU TLB SHOOTDOWN



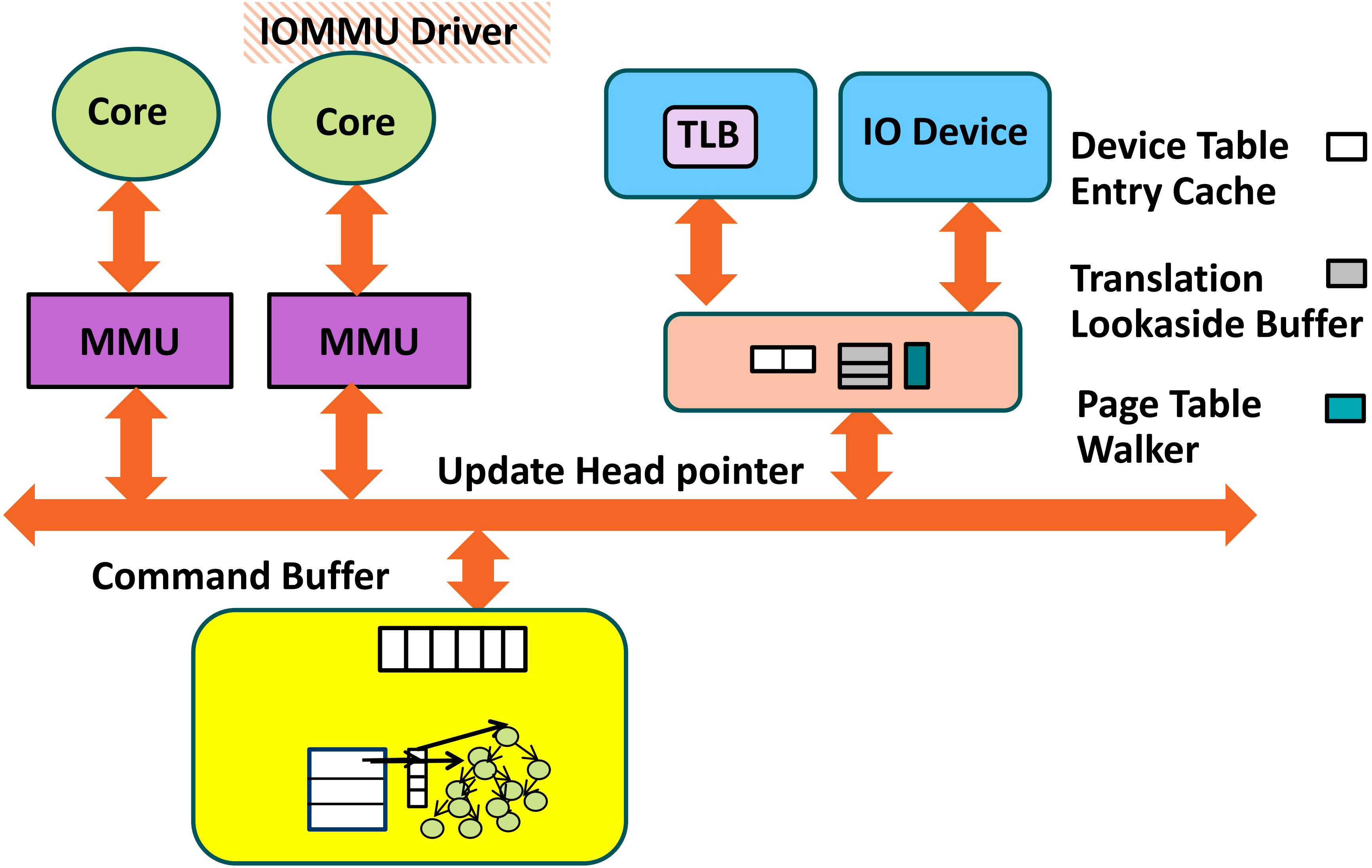
EXAMPLE: IOMMU TLB SHOOTDOWN



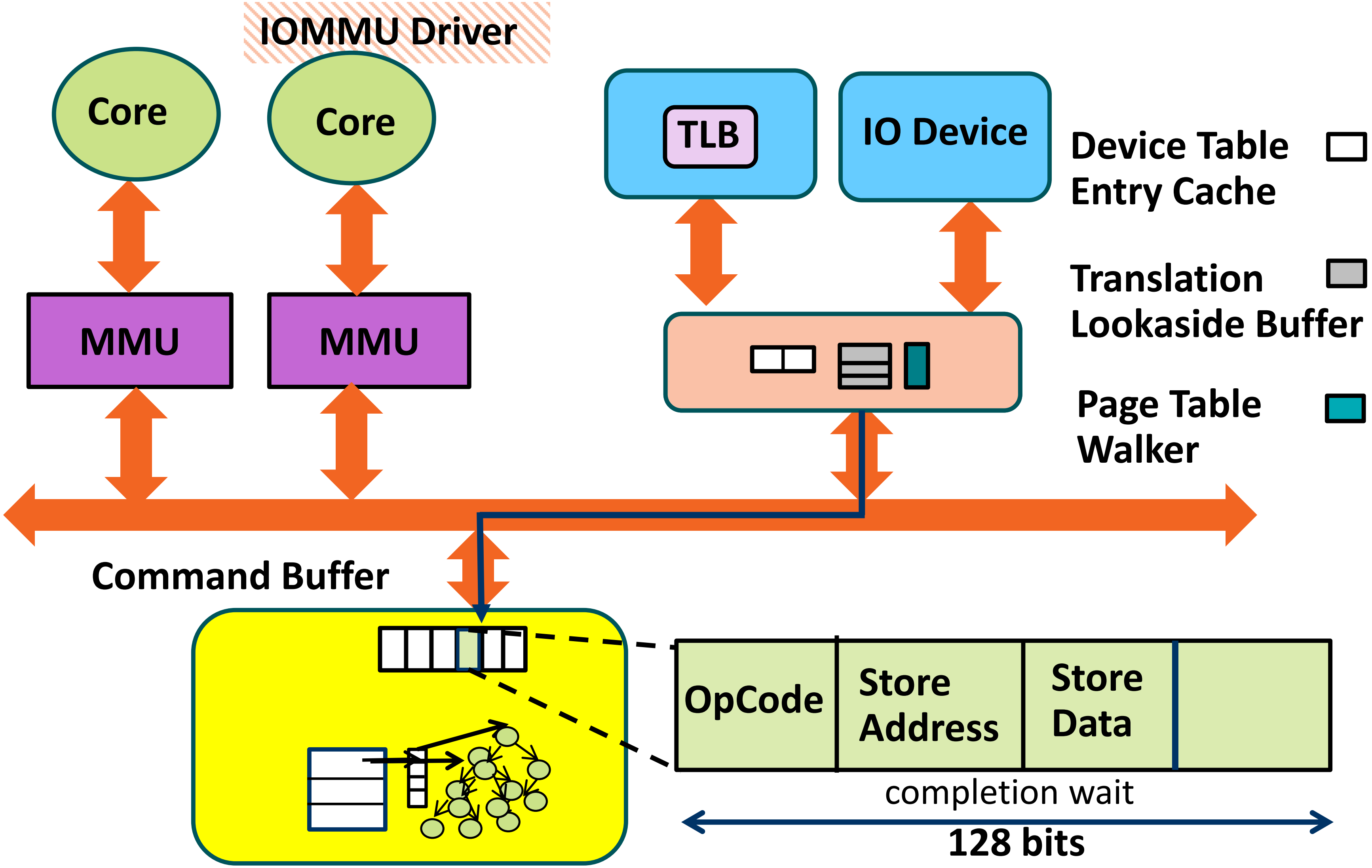
EXAMPLE: IOMMU TLB SHOOTDOWN



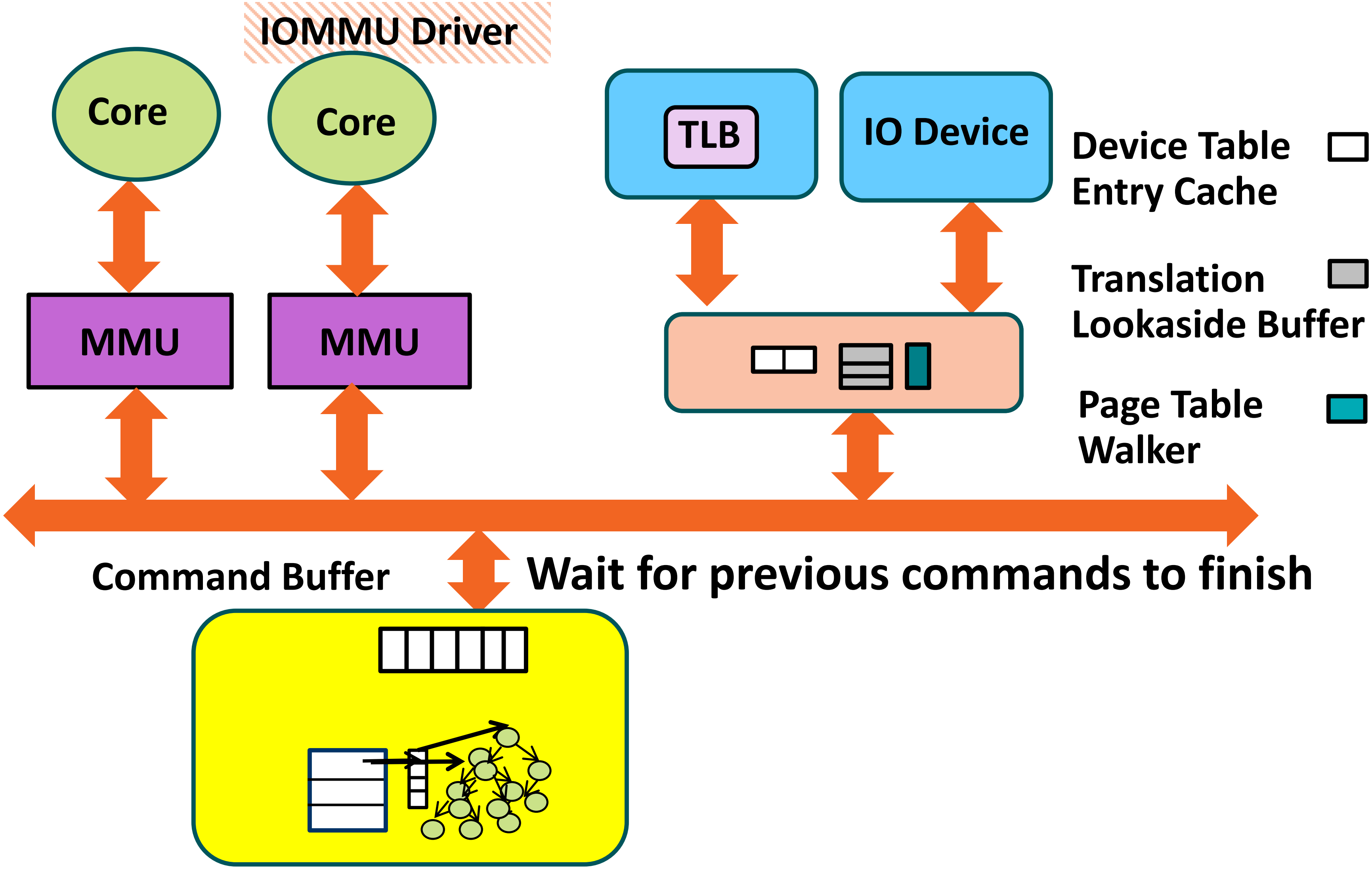
EXAMPLE: IOMMU TLB SHOOTDOWN



EXAMPLE: IOMMU TLB SHOOTDOWN

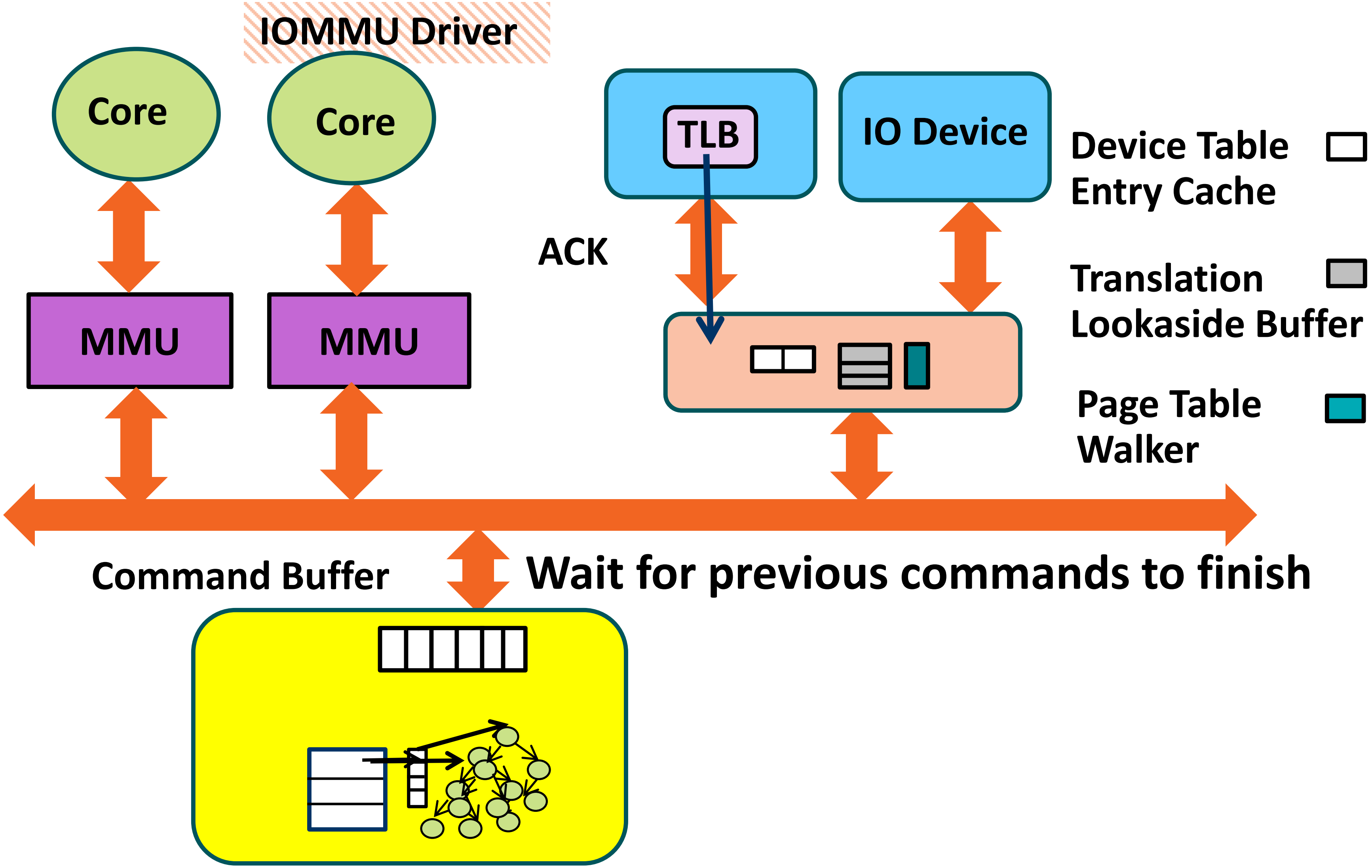


EXAMPLE: IOMMU TLB SHOOTDOWN



Wait for previous commands to finish

EXAMPLE: IOMMU TLB SHOOTDOWN



IOMMU Driver

Core

Core

MMU

MMU

TLB

IO Device

ACK

Device Table Entry Cache □

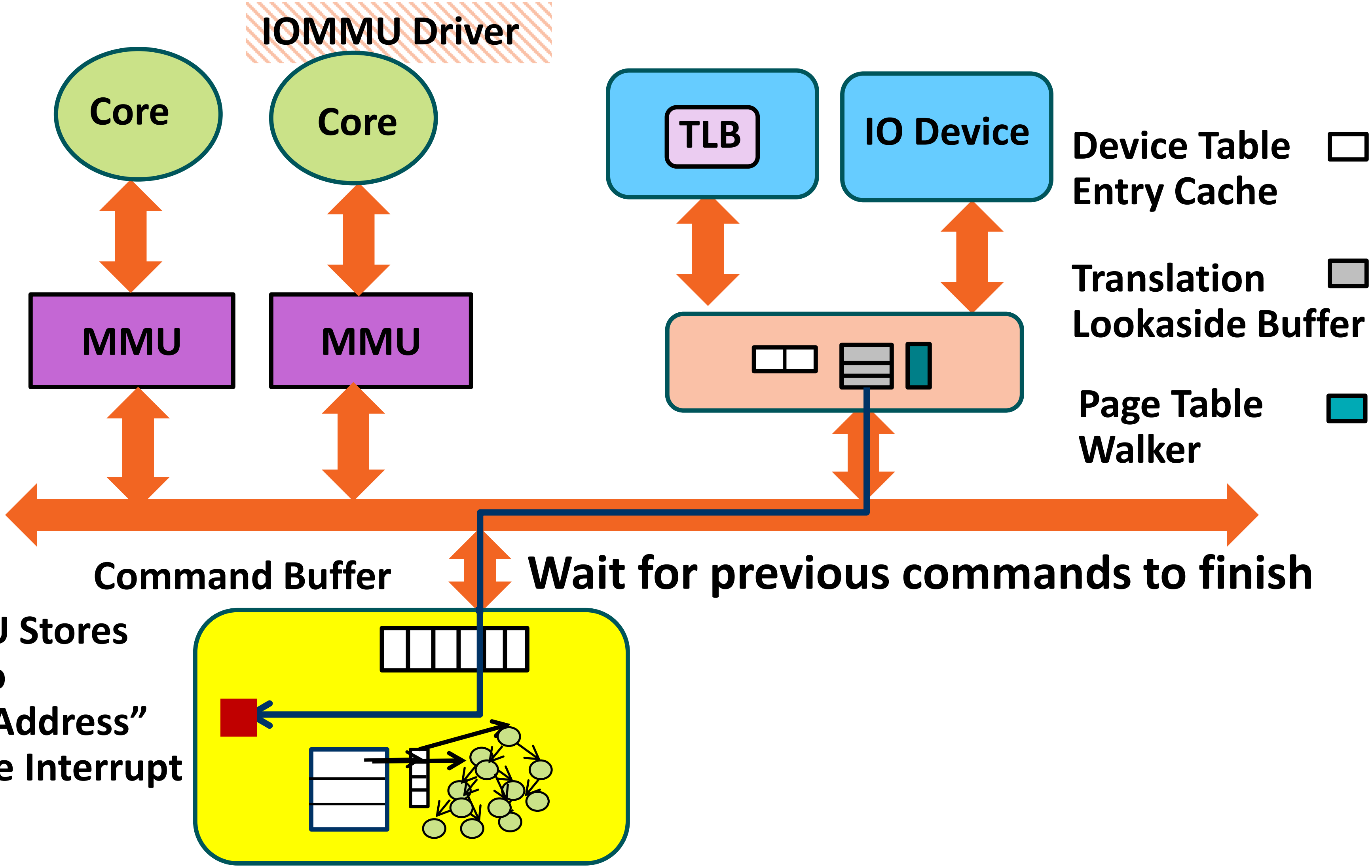
Translation Lookaside Buffer ▨

Page Table Walker 🟦

Command Buffer

Wait for previous commands to finish

EXAMPLE: IOMMU TLB SHOOTDOWN



IOMMU Driver

Core

Core

MMU

MMU

TLB

IO Device

Device Table Entry Cache □

Translation Lookaside Buffer ▨

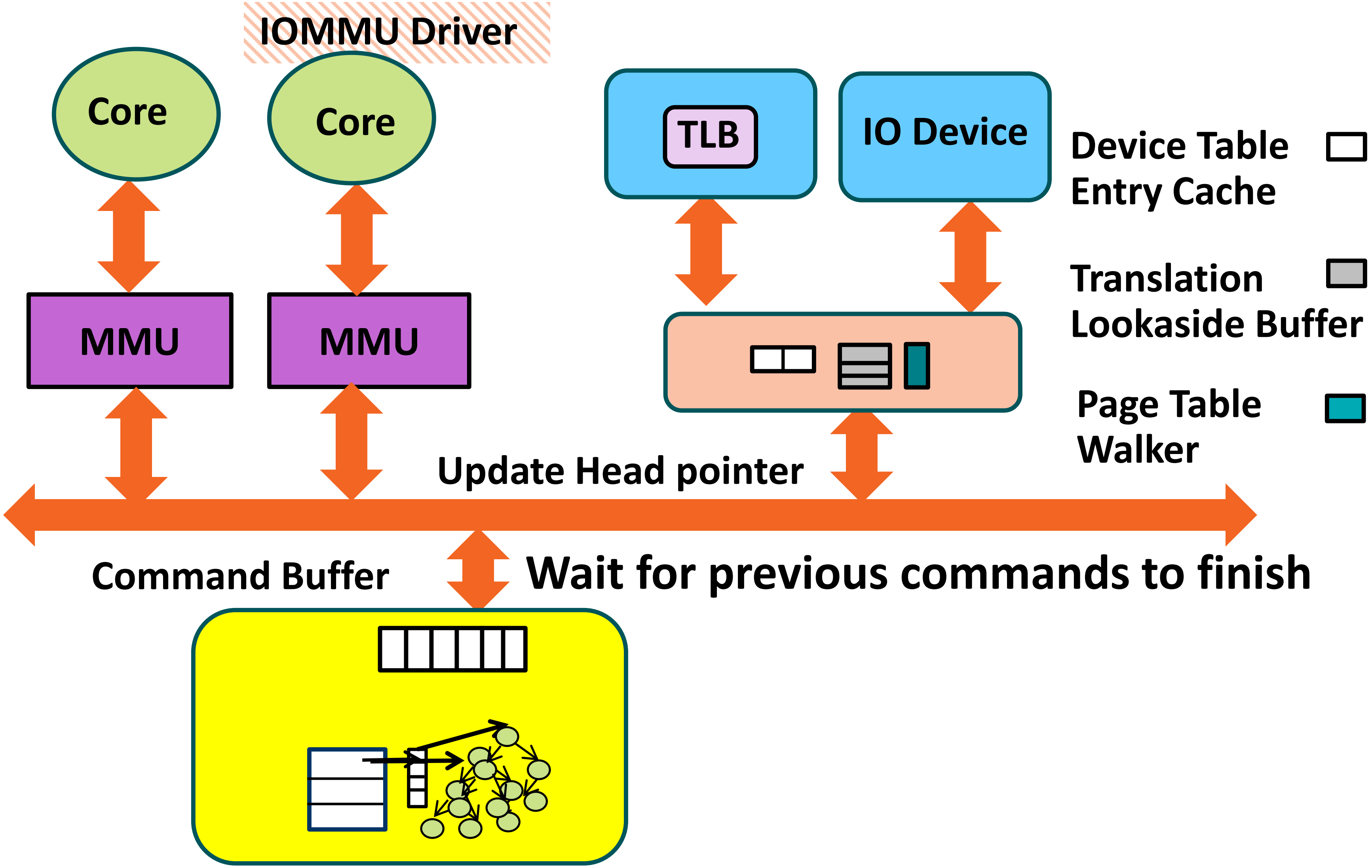
Page Table Walker 🟦

Command Buffer

Wait for previous commands to finish

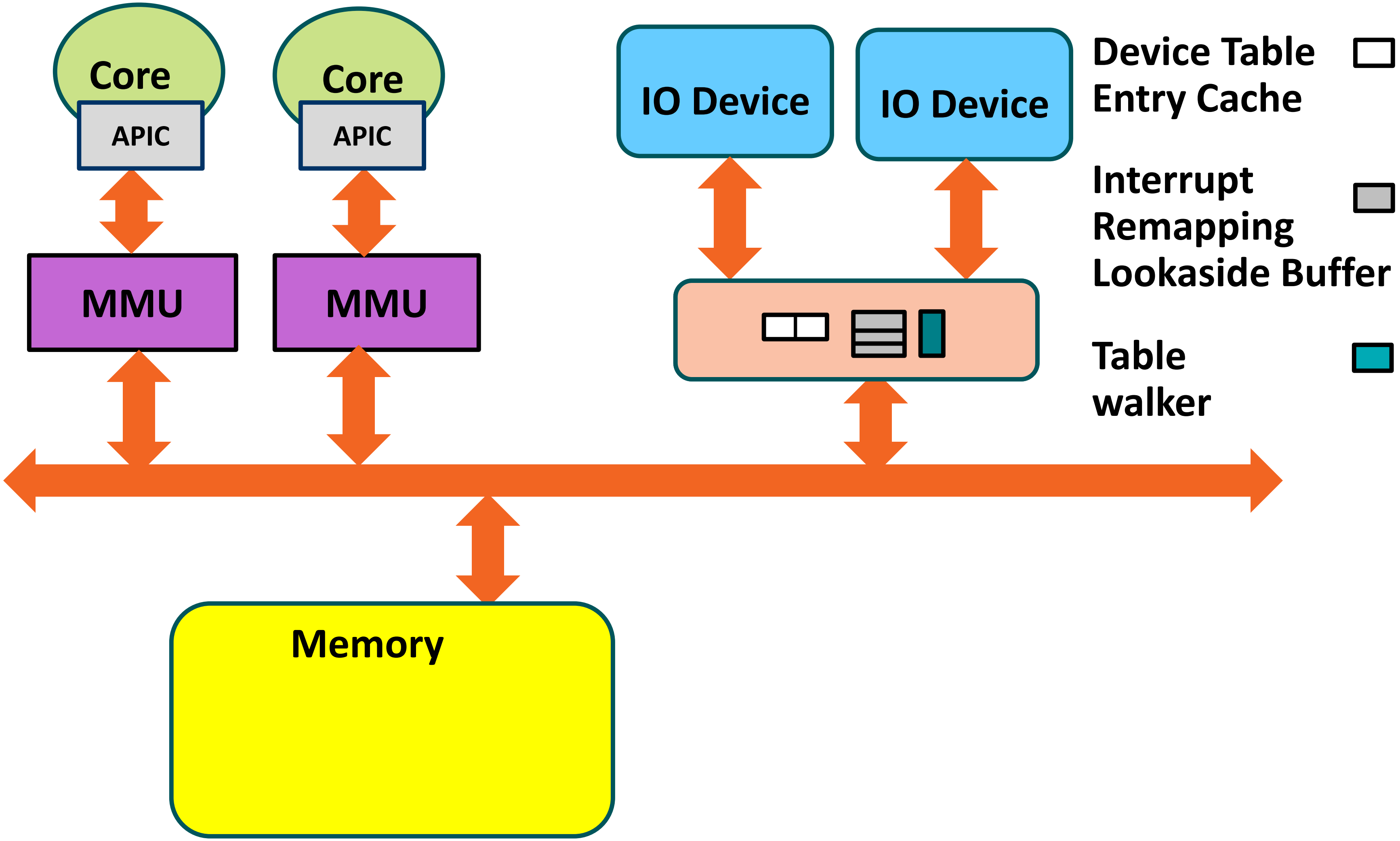
IOMMU Stores
Data to
“Store Address”
Or Raise Interrupt

EXAMPLE: IOMMU TLB SHOOTDOWN



IOMMU INTERNALS: INTERRUPT REMAPPING AND VIRTUALIZATION

INTERRUPT REMAPPING



Core

APIC

Core

APIC

MMU

MMU

IO Device

IO Device

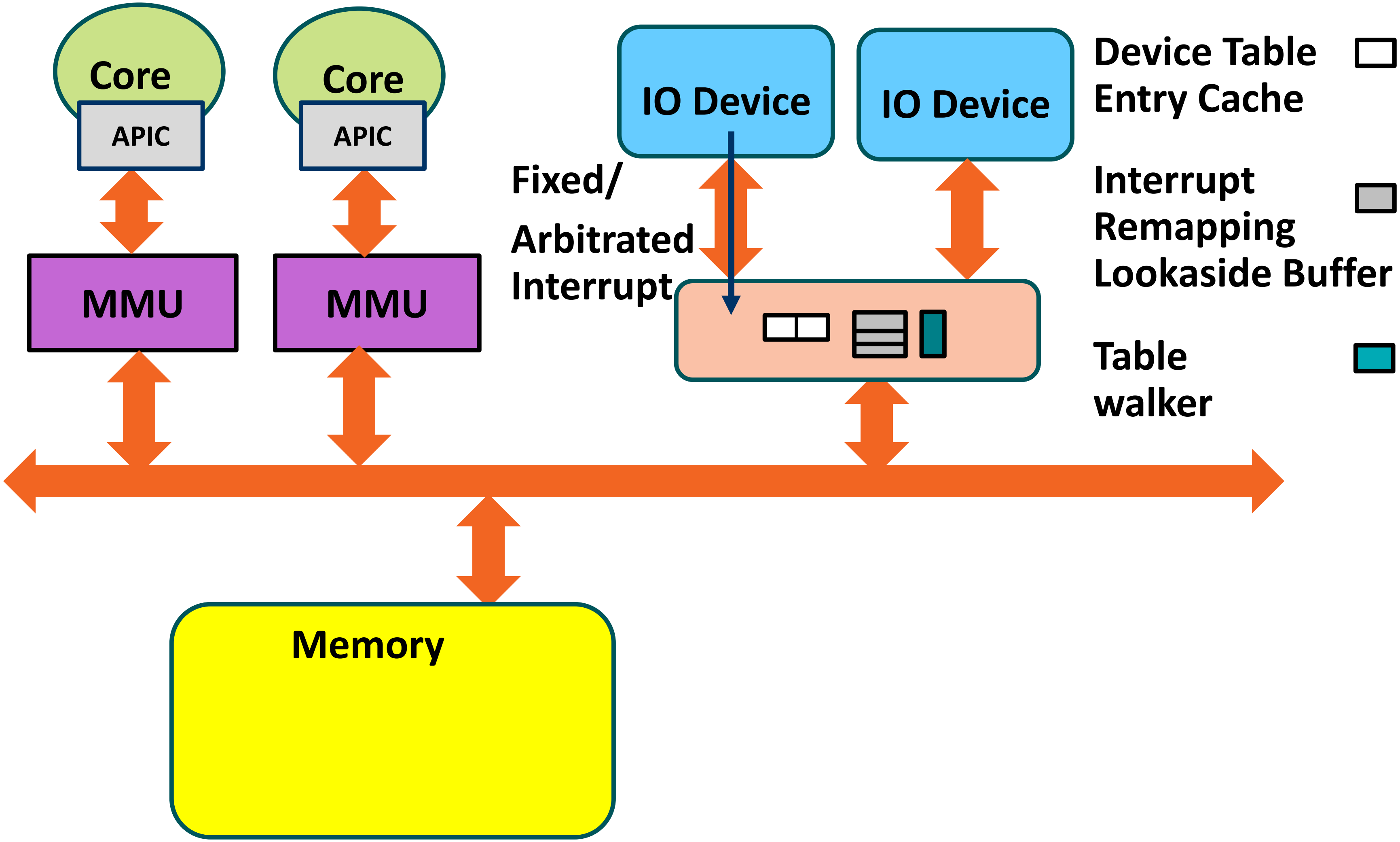
Memory

Device Table Entry Cache □

Interrupt Remapping Lookaside Buffer ▨

Table walker ■

INTERRUPT REMAPPING



Core

APIC

Core

APIC

MMU

MMU

IO Device

IO Device

Fixed/

Arbitrated

Interrupt

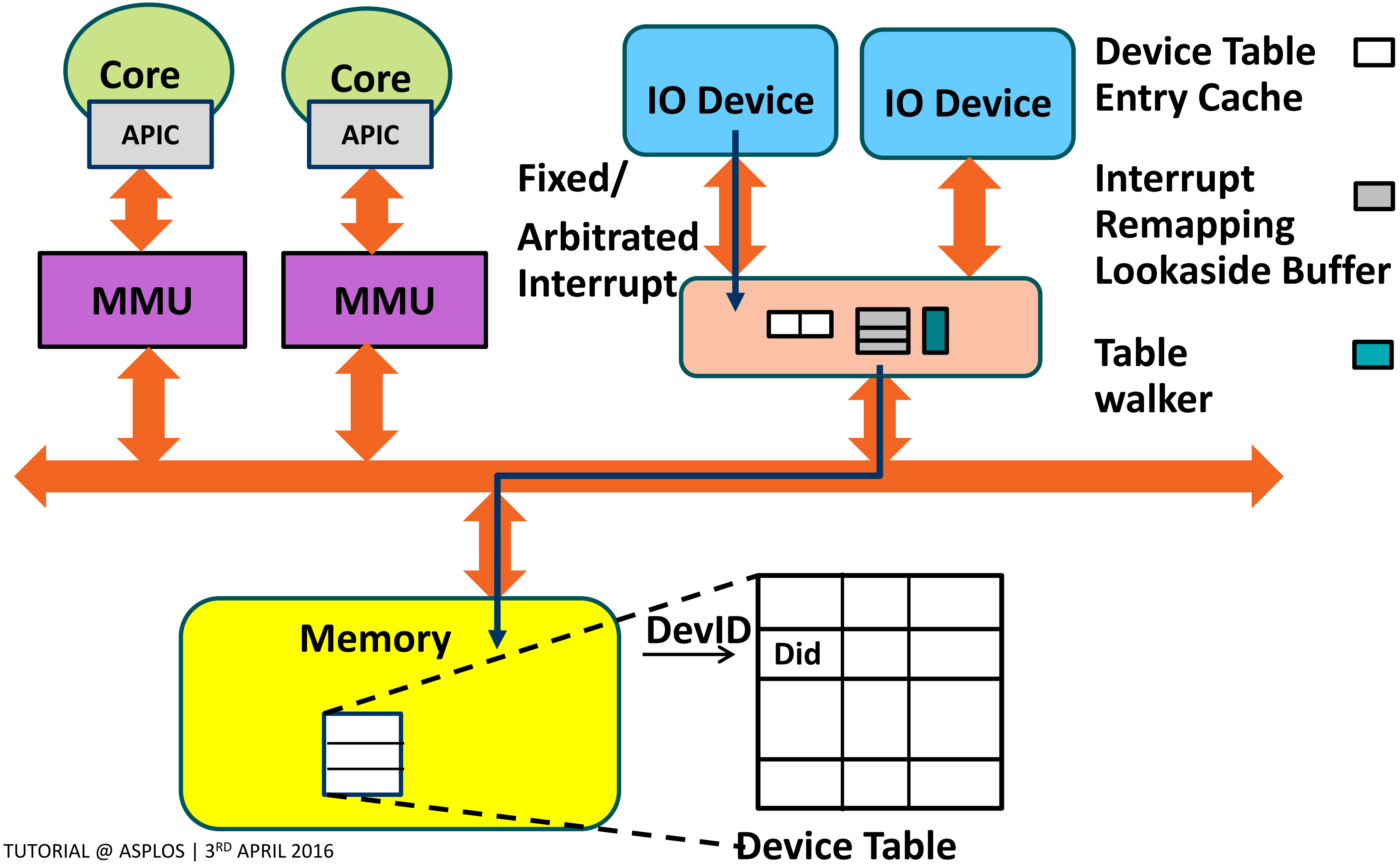
Memory

Device Table Entry Cache □

Interrupt Remapping Lookaside Buffer ▨

Table walker ■

INTERRUPT REMAPPING



Core

APIC

Core

APIC

MMU

MMU

IO Device

IO Device

Fixed/

Arbitrated

Interrupt

Memory

DevID

Did

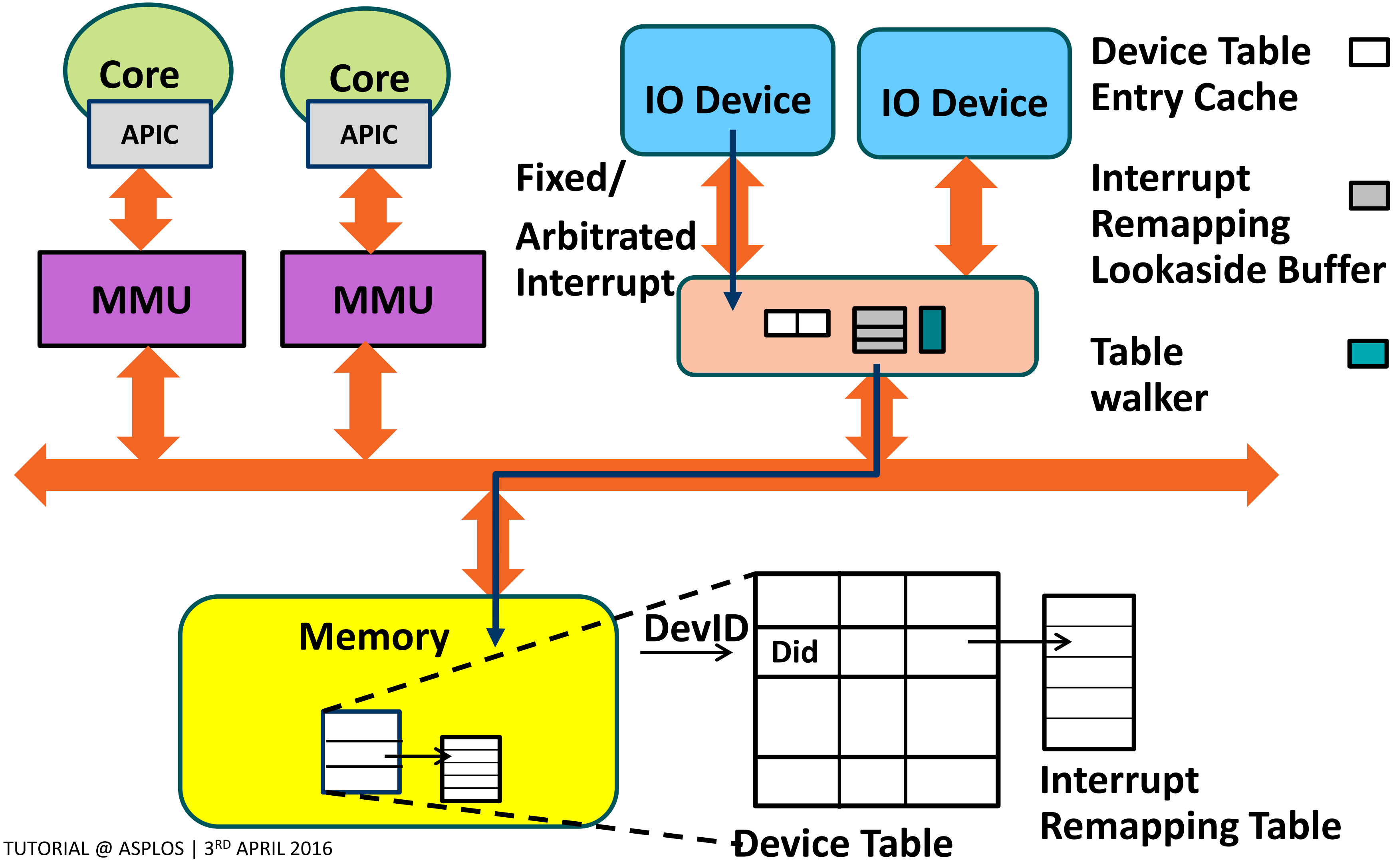
Device Table

Device Table Entry Cache □

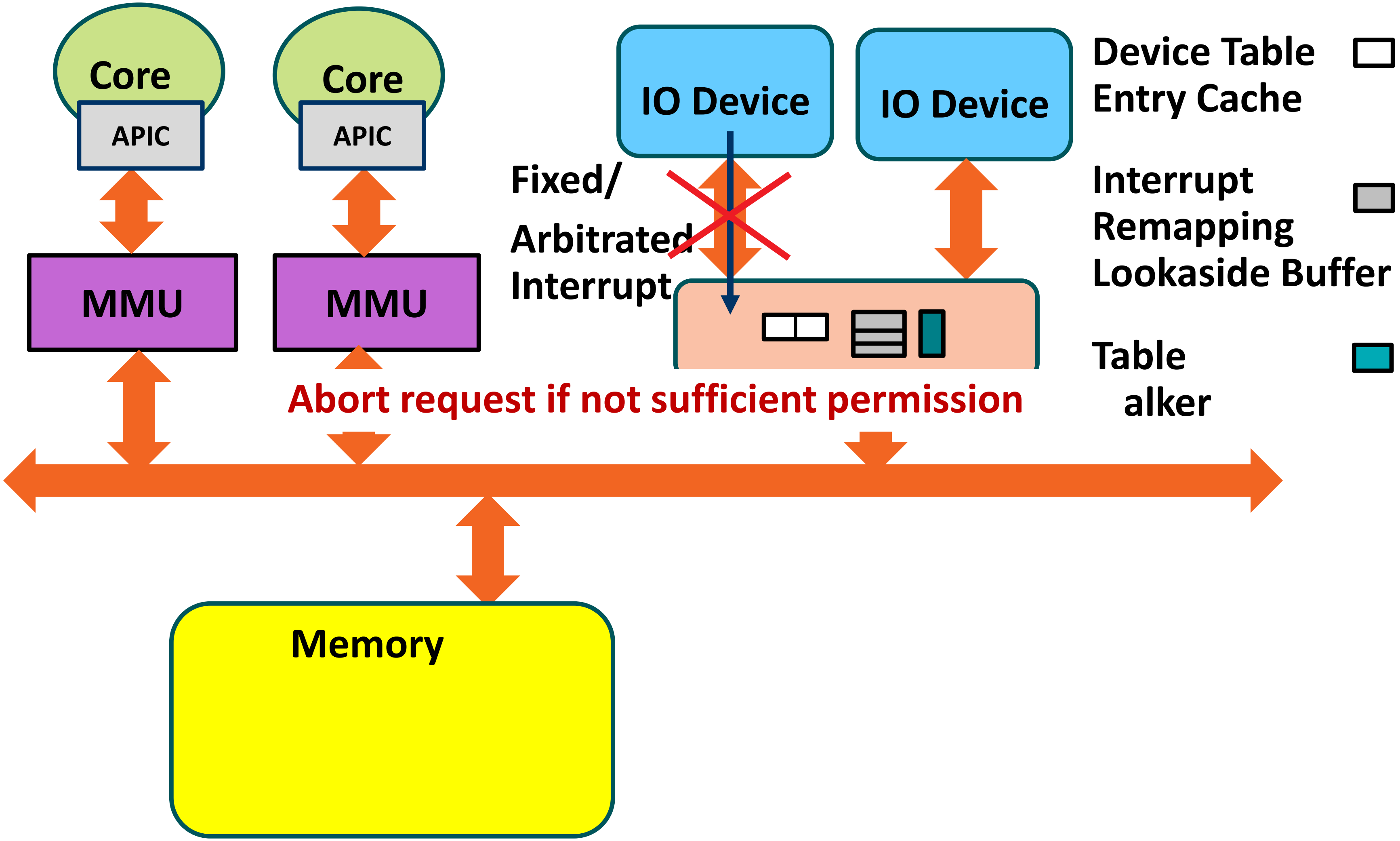
Interrupt Remapping Lookaside Buffer ▨

Table walker ■

INTERRUPT REMAPPING



INTERRUPT REMAPPING



Core

APIC

Core

APIC

MMU

MMU

IO Device

IO Device

Fixed/

Arbitrated

Interrupt

Abort request if not sufficient permission

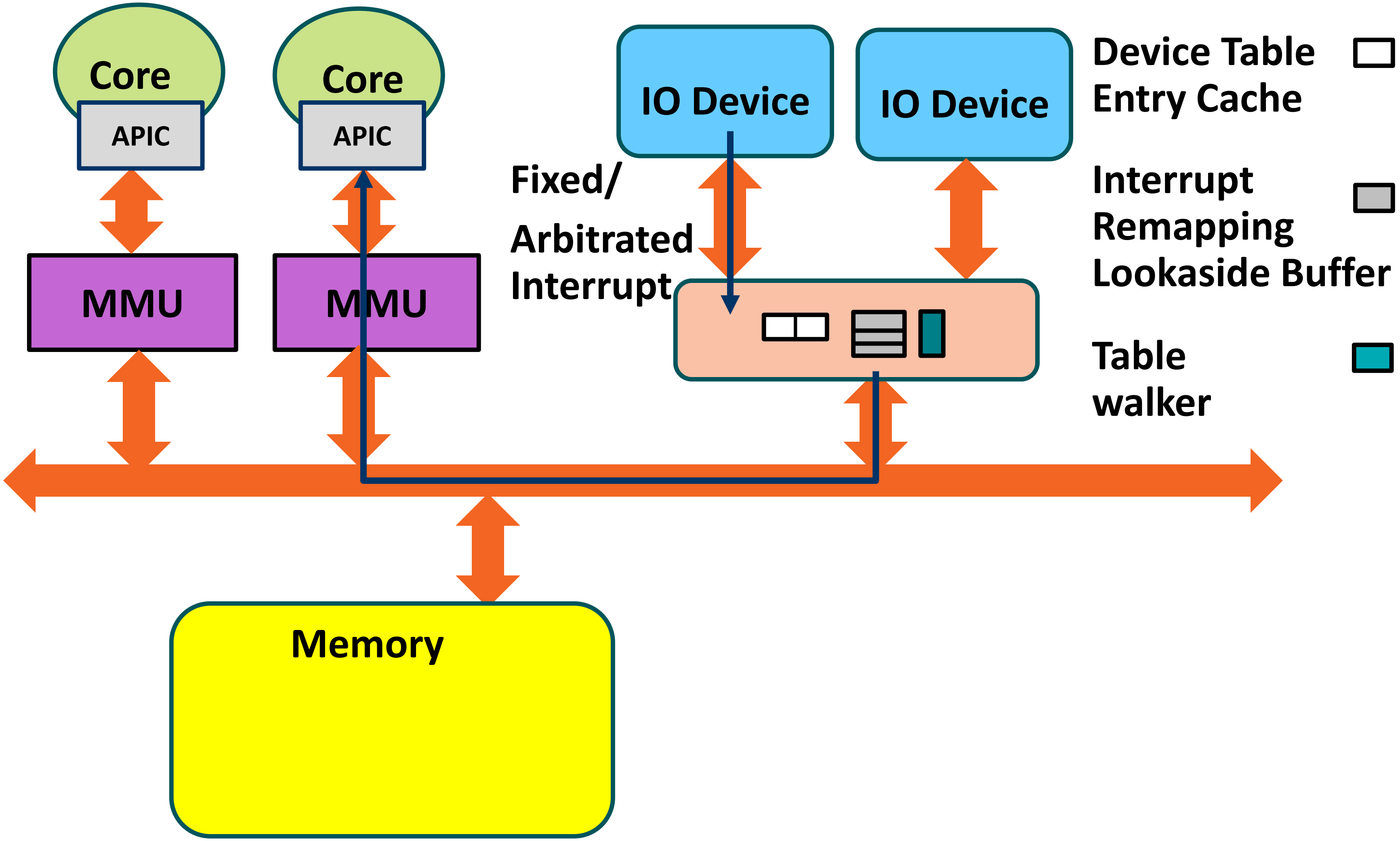
Memory

Device Table Entry Cache □

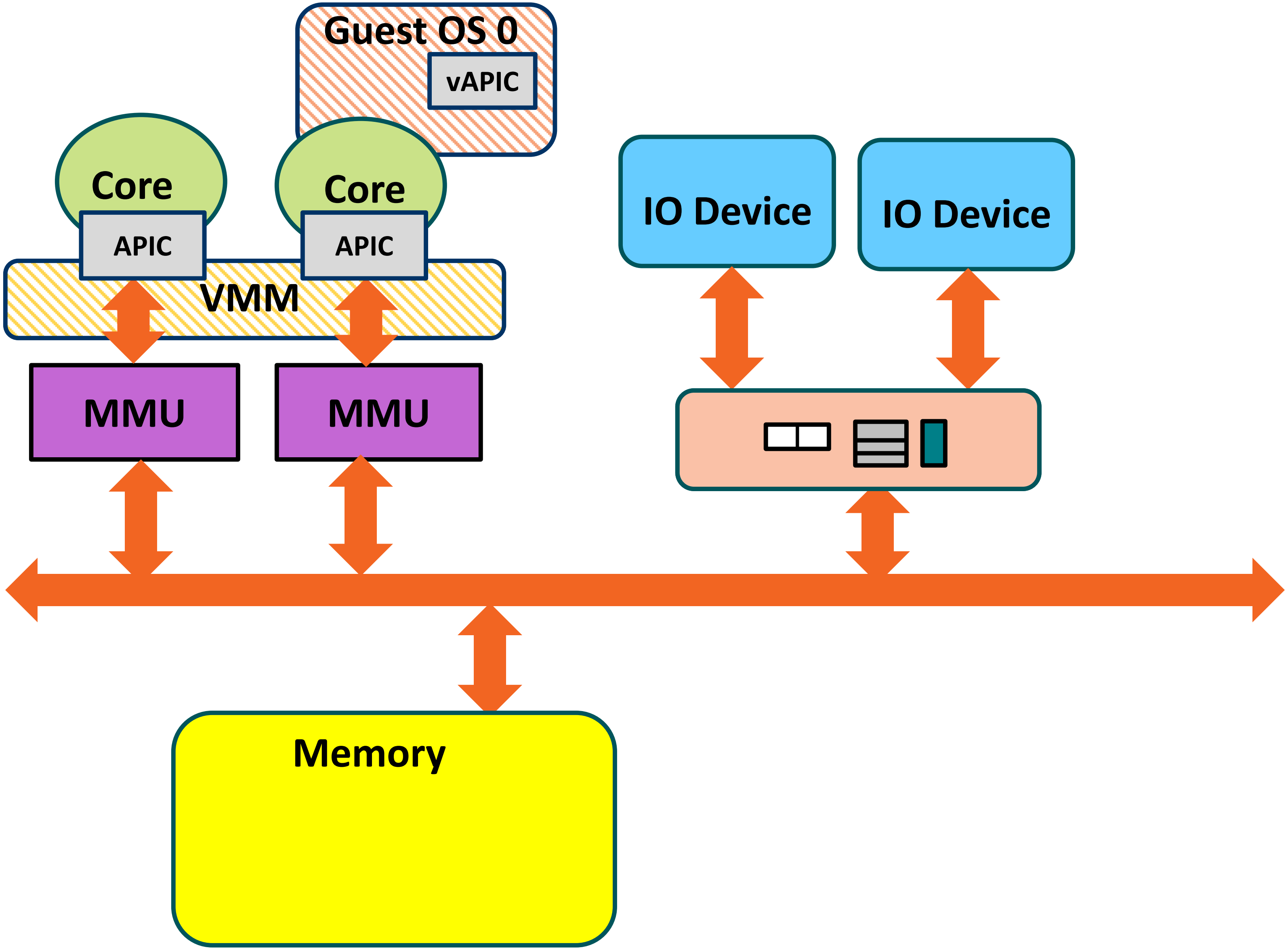
Interrupt Remapping Lookaside Buffer ▨

Table
alker ■

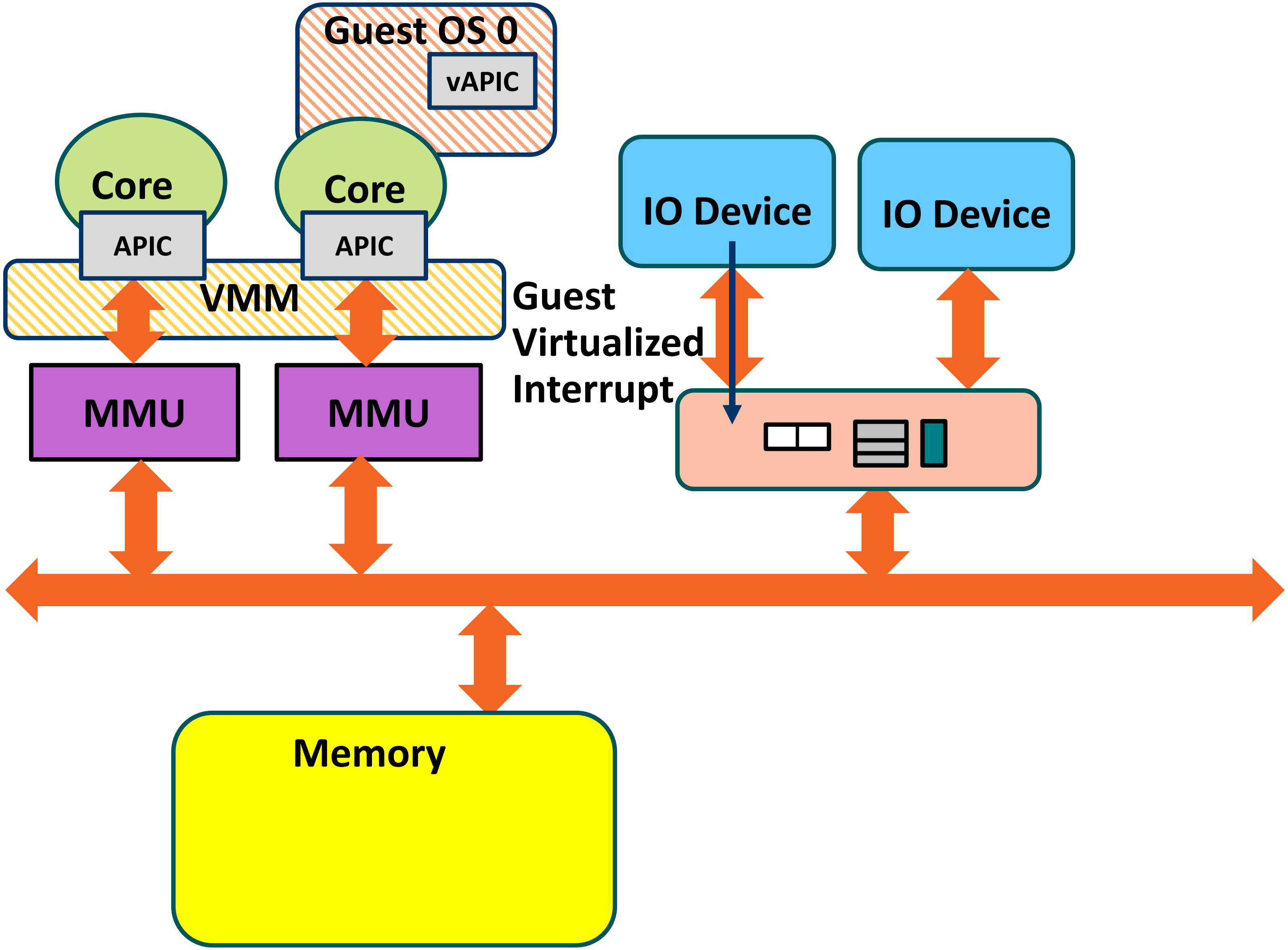
INTERRUPT REMAPPING



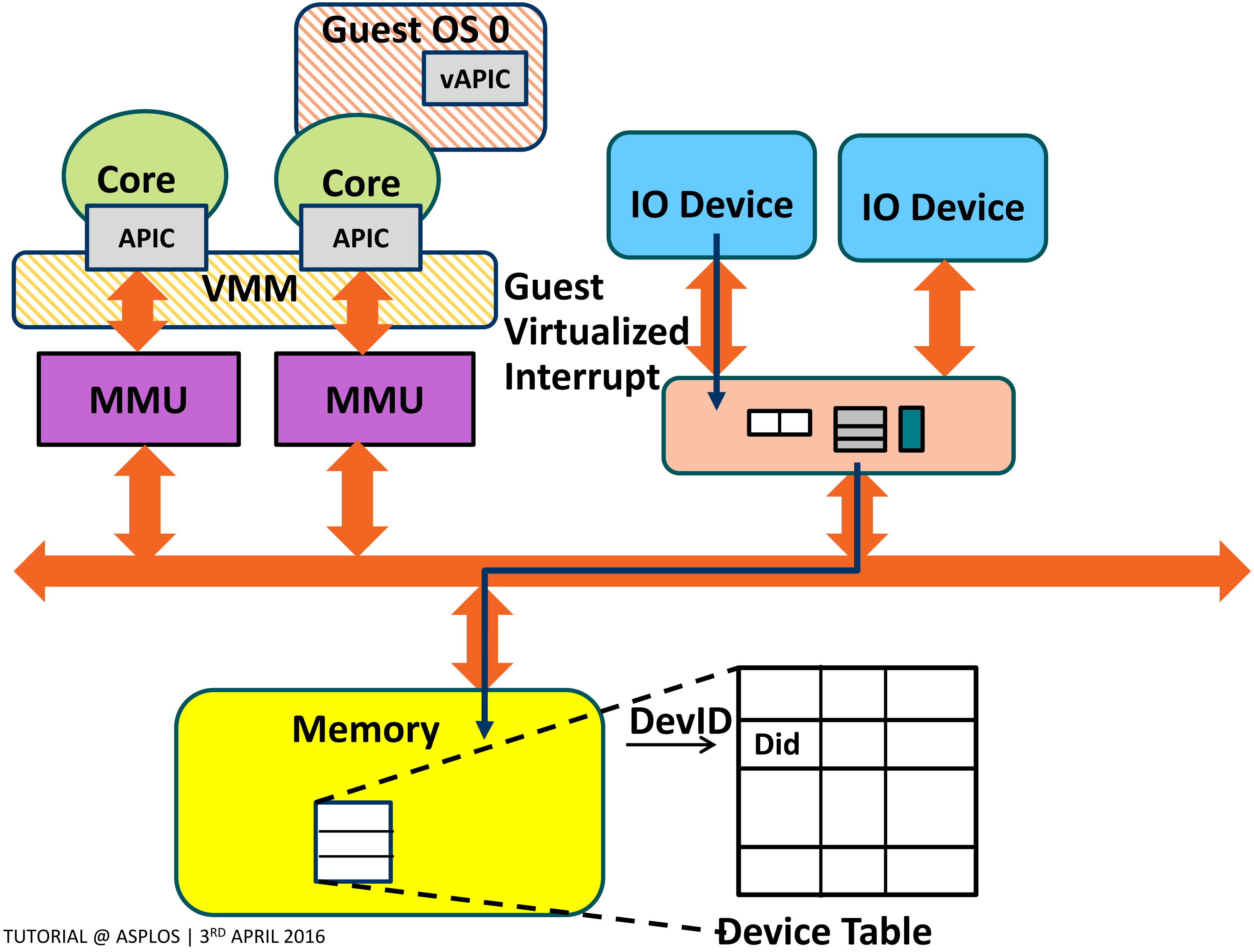
INTERRUPT VIRTUALIZATION



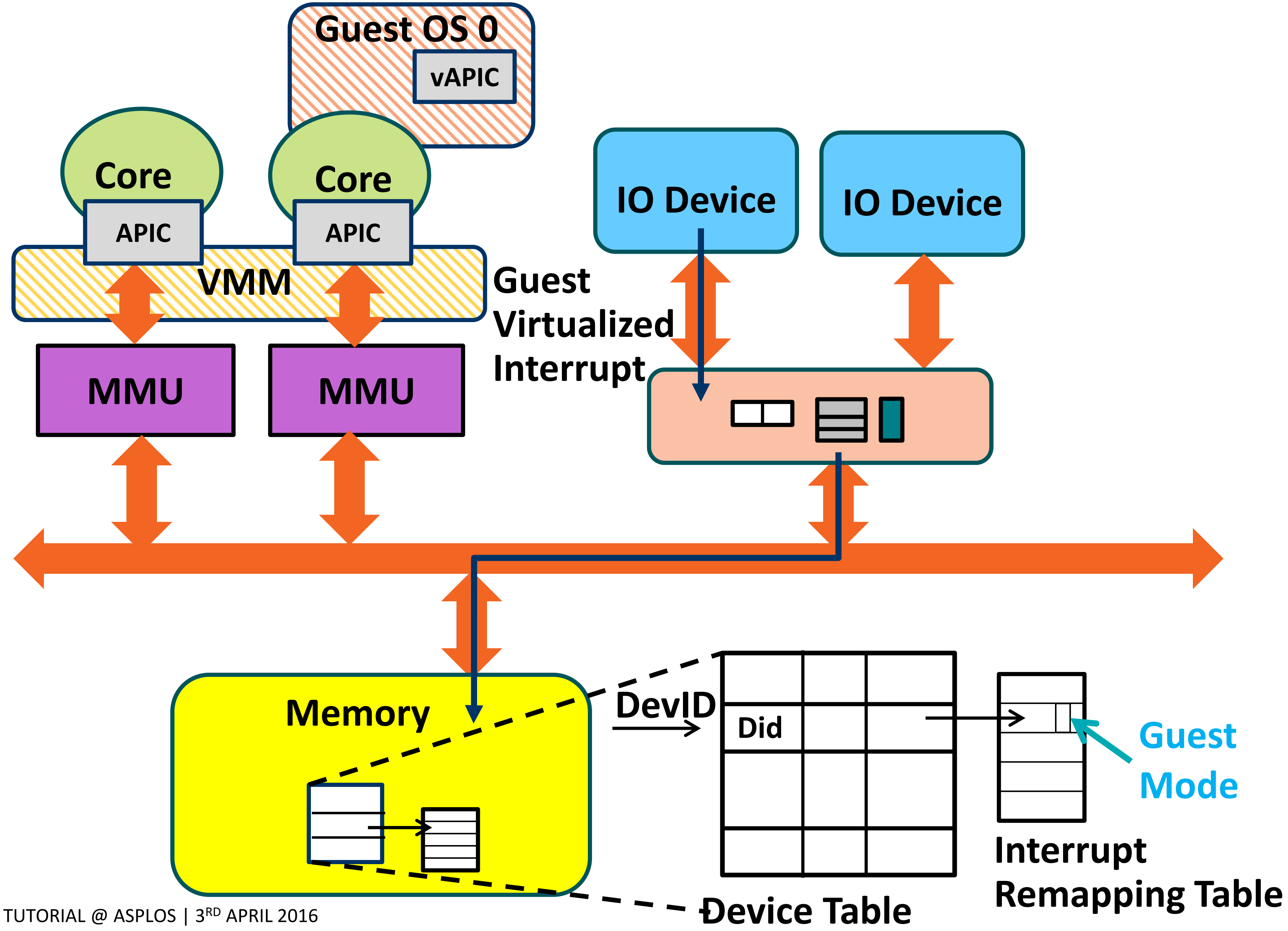
INTERRUPT VIRTUALIZATION



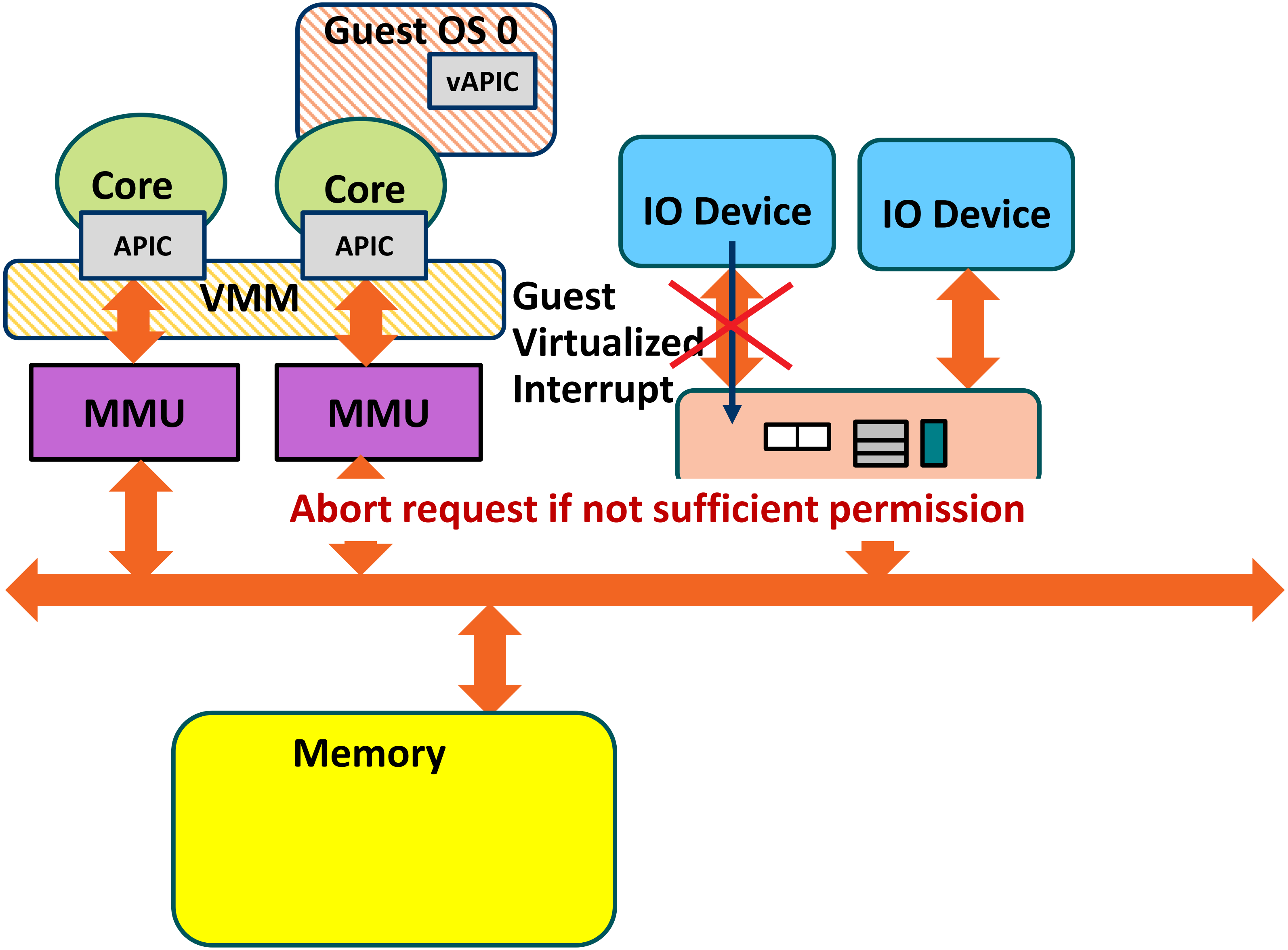
INTERRUPT VIRTUALIZATION



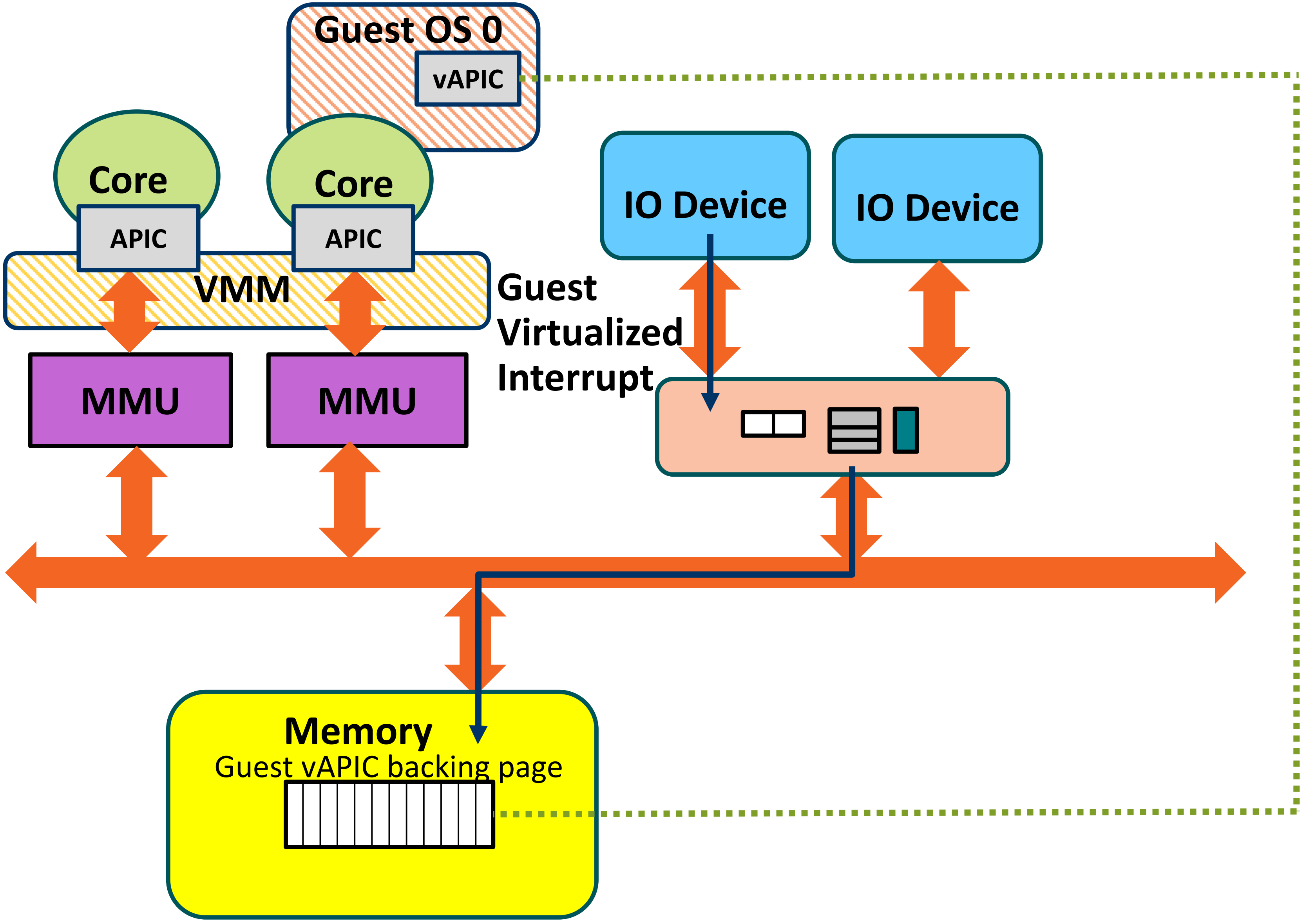
INTERRUPT VIRTUALIZATION



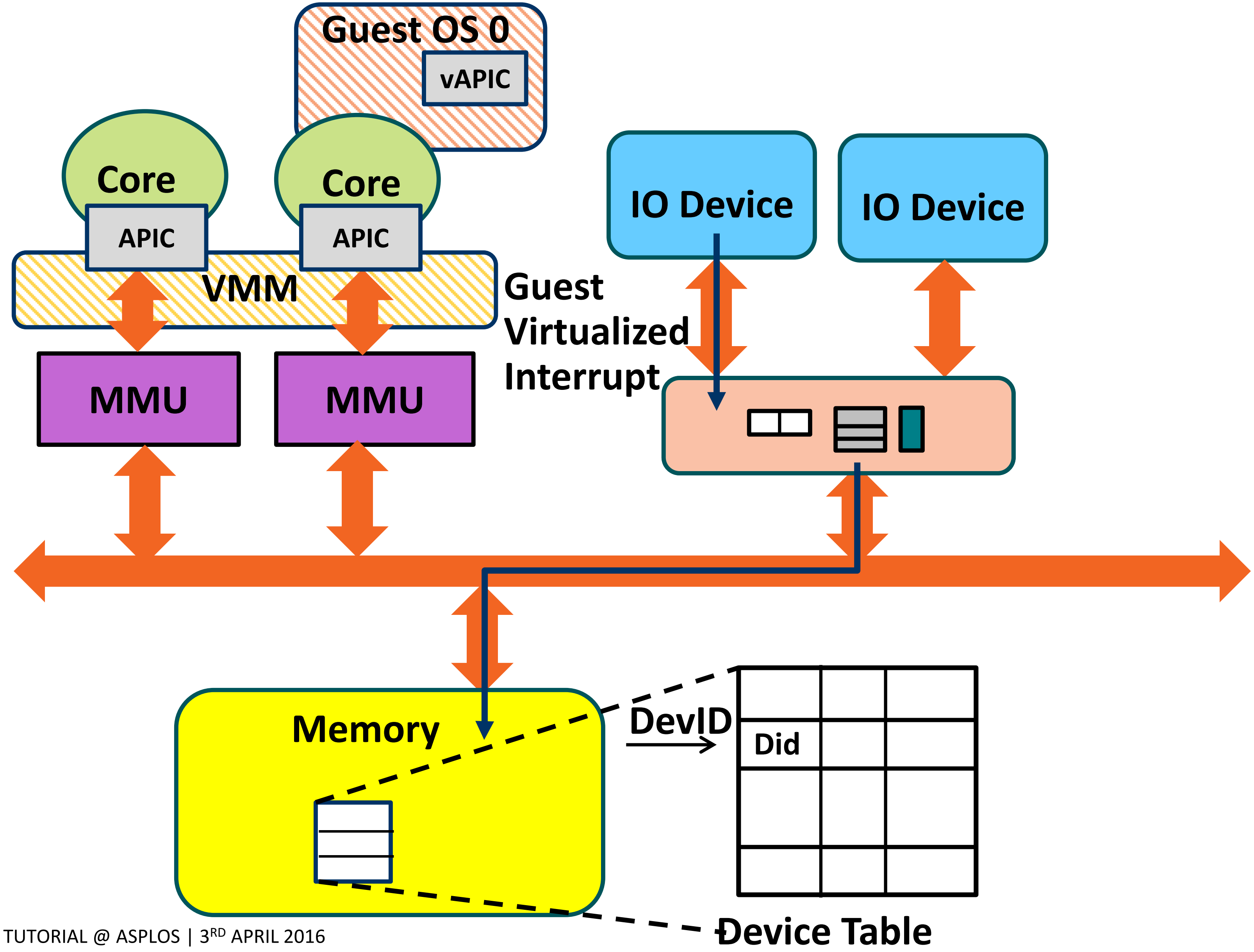
INTERRUPT VIRTUALIZATION


Guest OS 0
vAPIC
Core
APIC
Core
APIC
VMM
Guest Virtualized Interrupt
IO Device
IO Device
MMU
MMU
Abort request if not sufficient permission
Memory

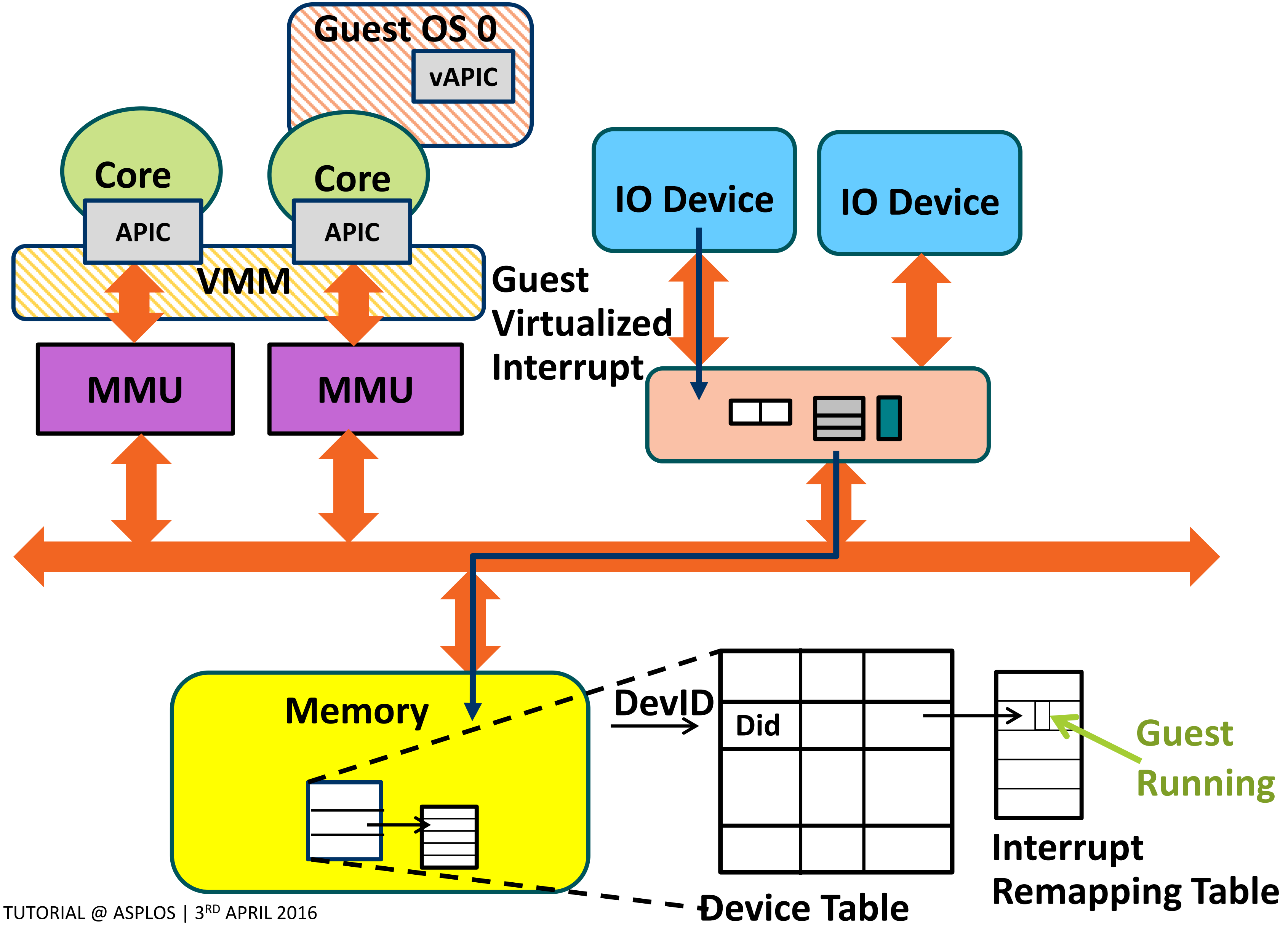
INTERRUPT VIRTUALIZATION



INTERRUPT VIRTUALIZATION

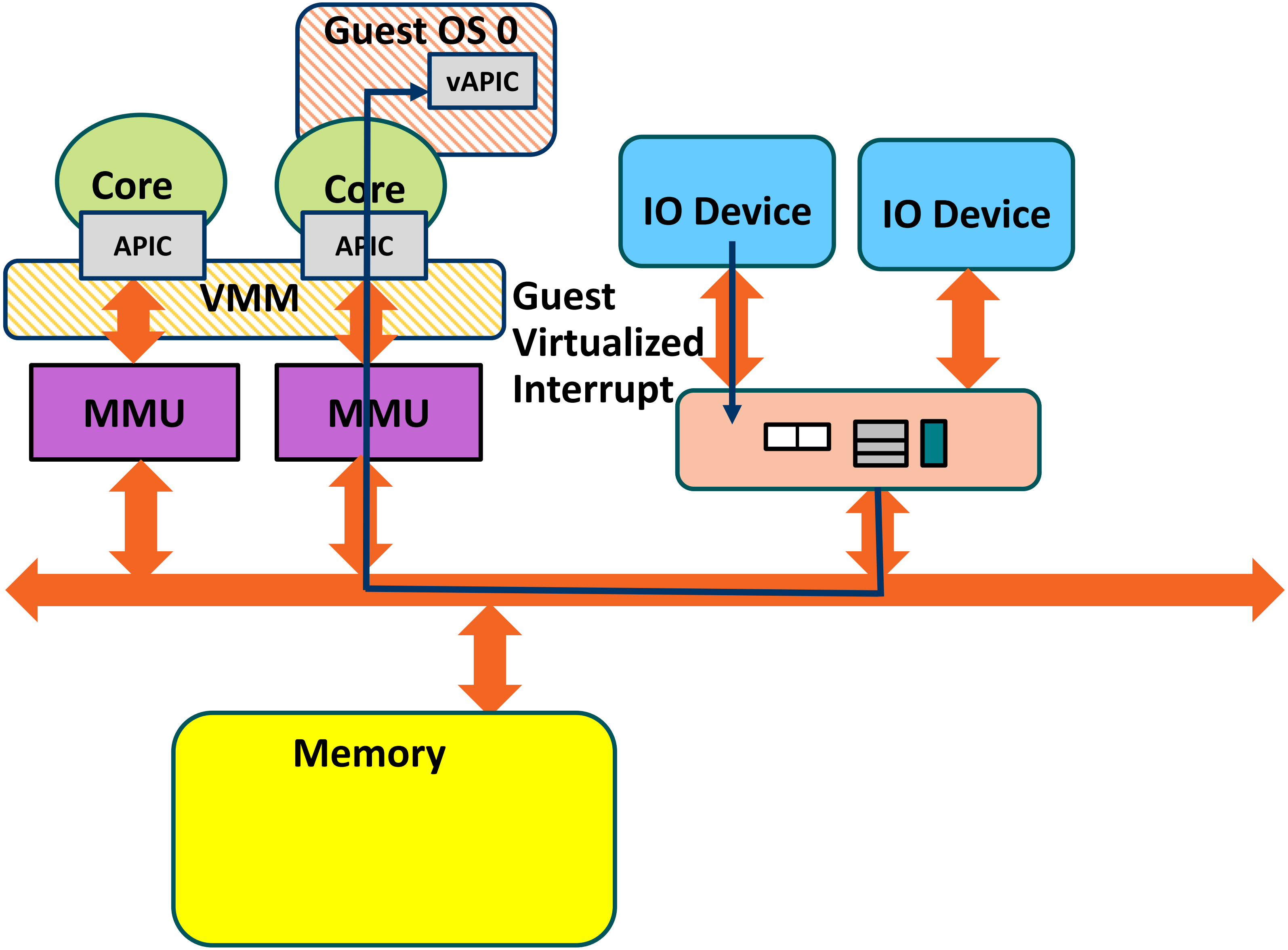


INTERRUPT VIRTUALIZATION

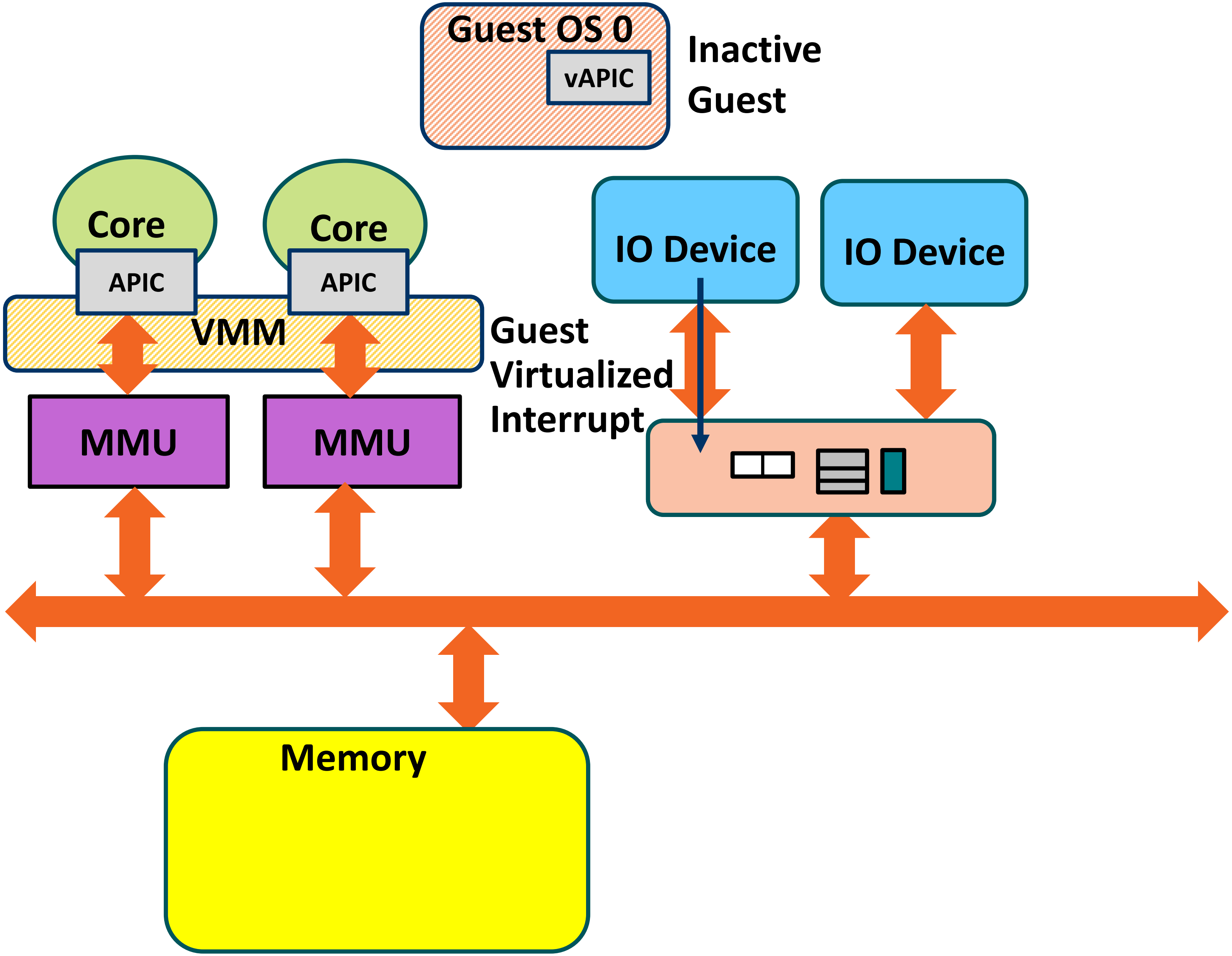


INTERRUPT VIRTUALIZATION

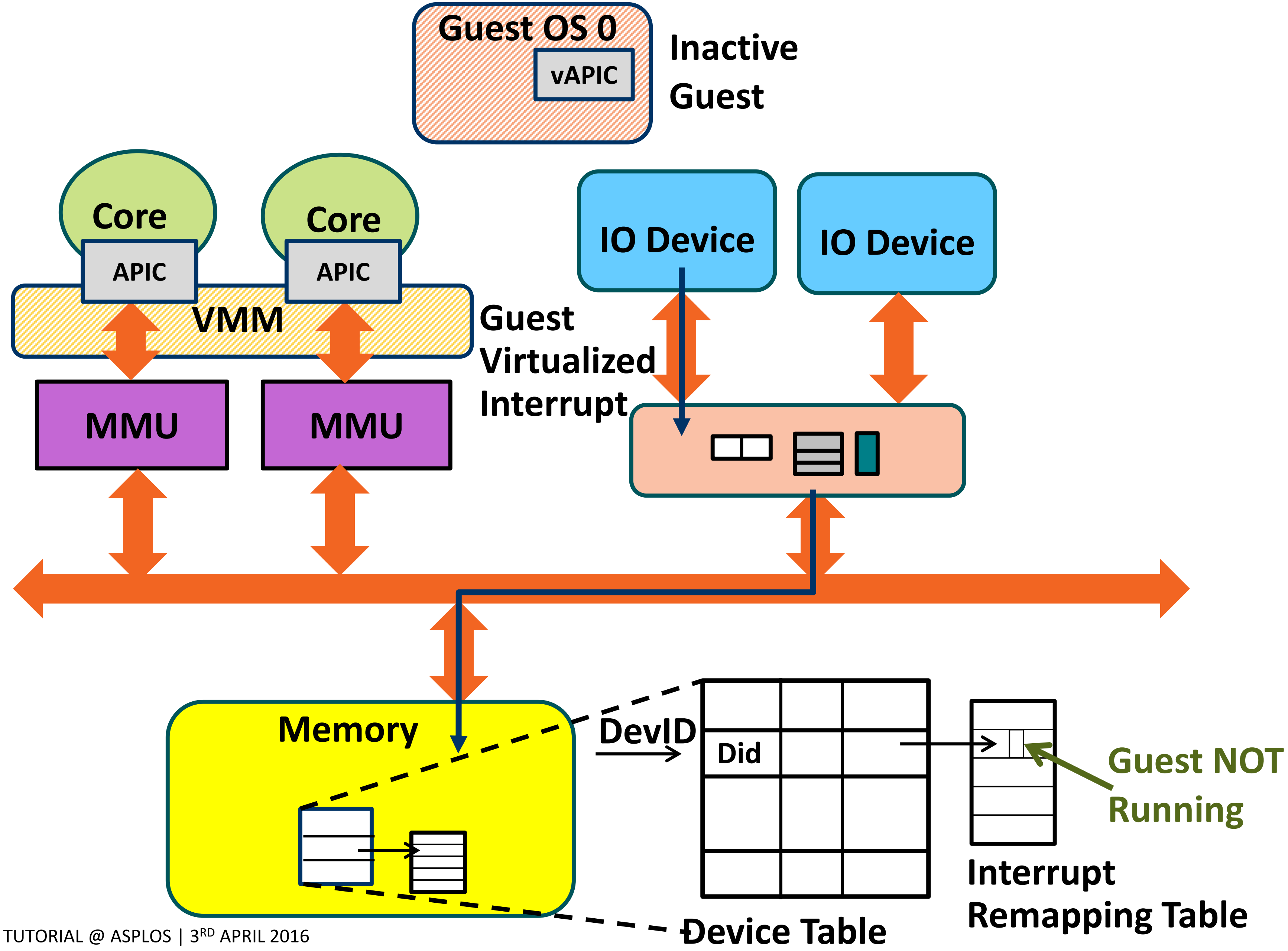
AMD



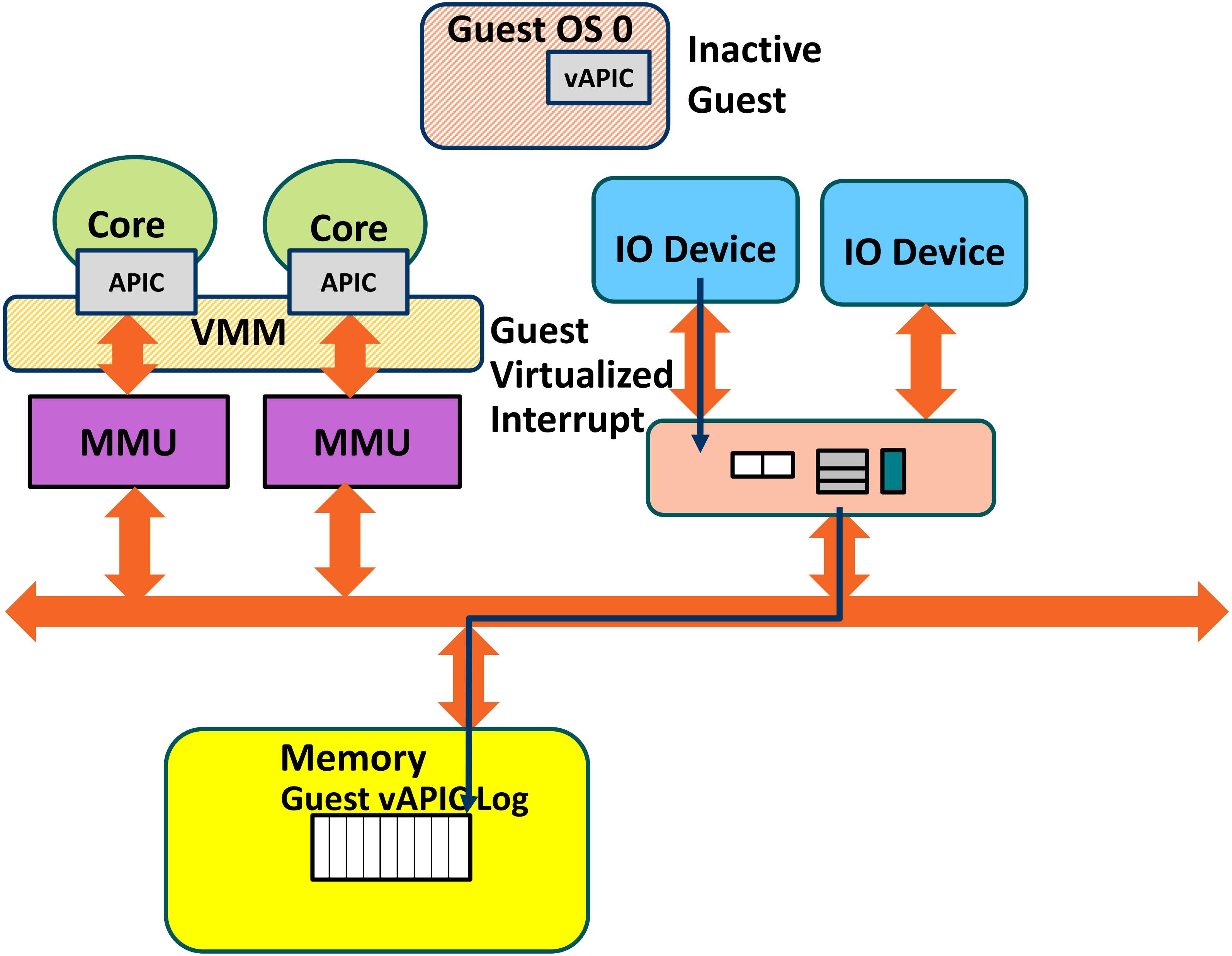
INTERRUPT VIRTUALIZATION



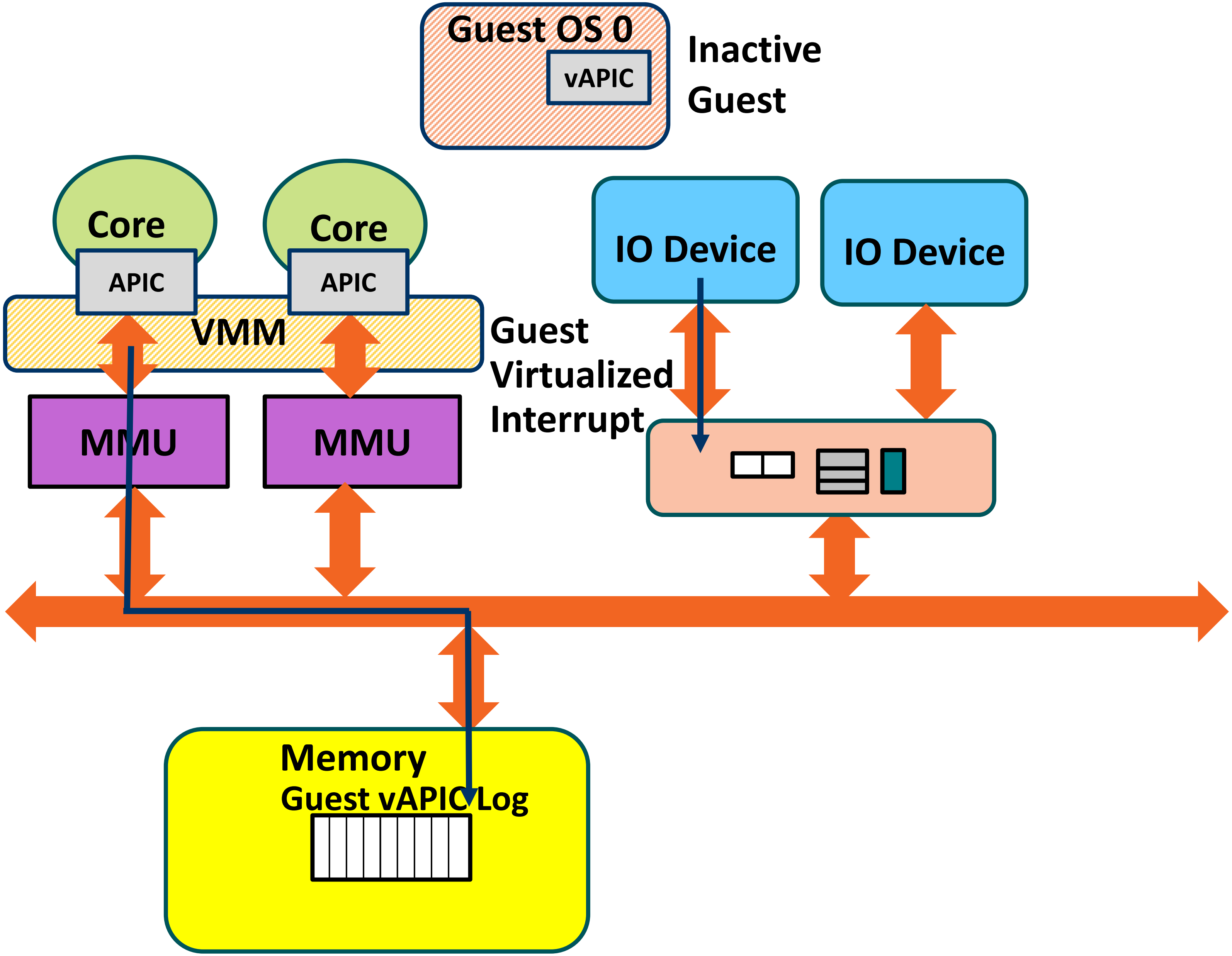
INTERRUPT VIRTUALIZATION



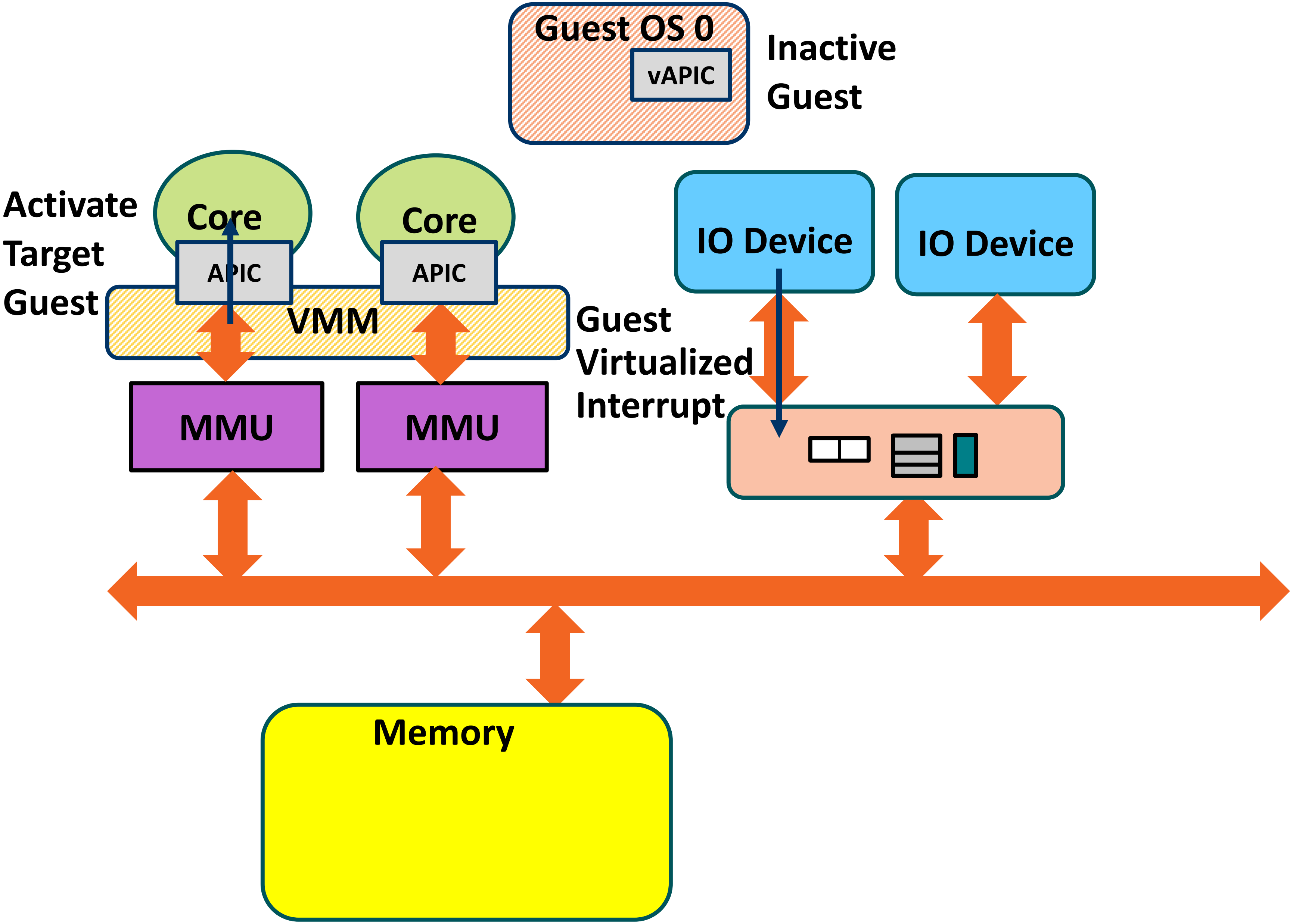
INTERRUPT VIRTUALIZATION



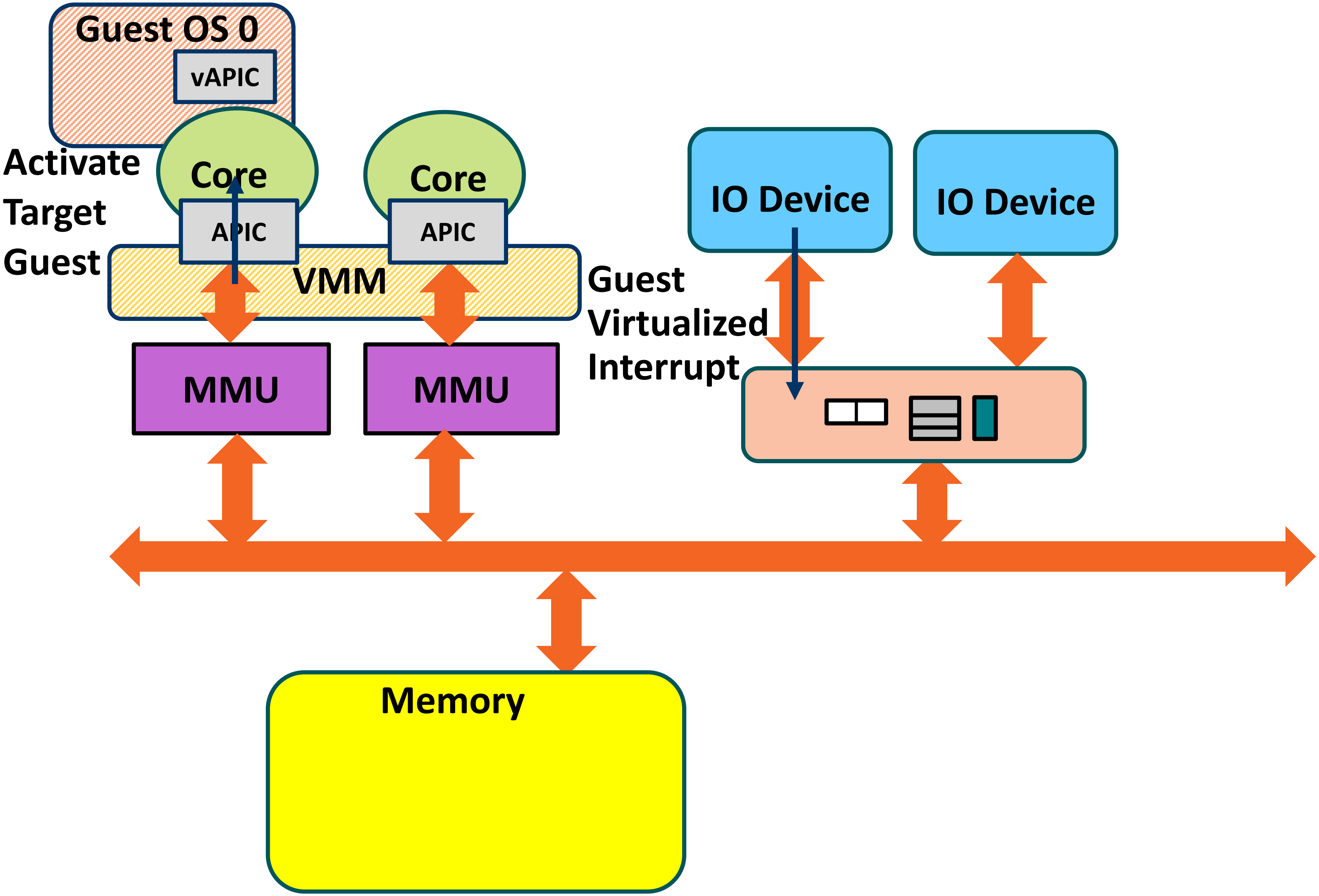
INTERRUPT VIRTUALIZATION



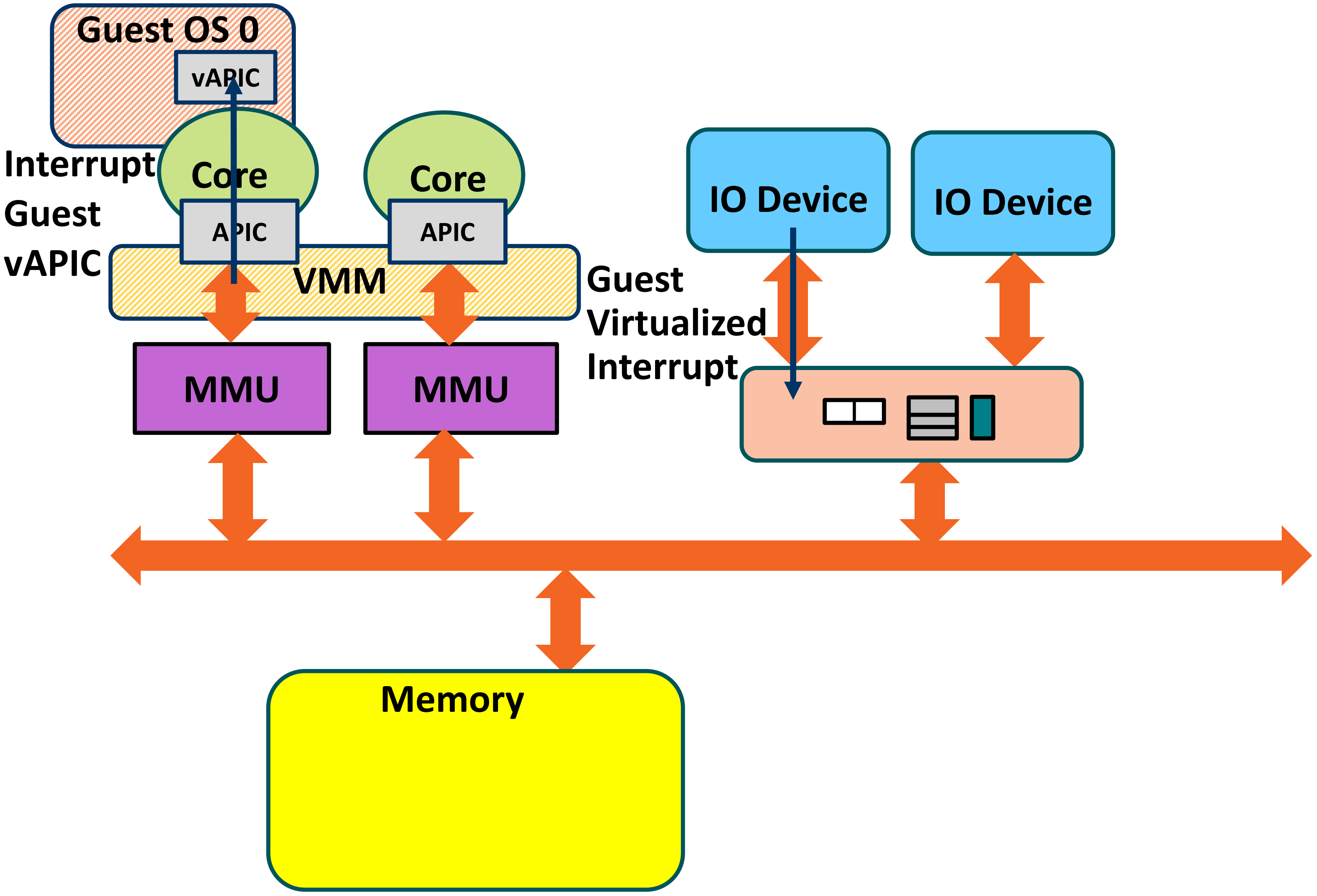
INTERRUPT VIRTUALIZATION



INTERRUPT VIRTUALIZATION

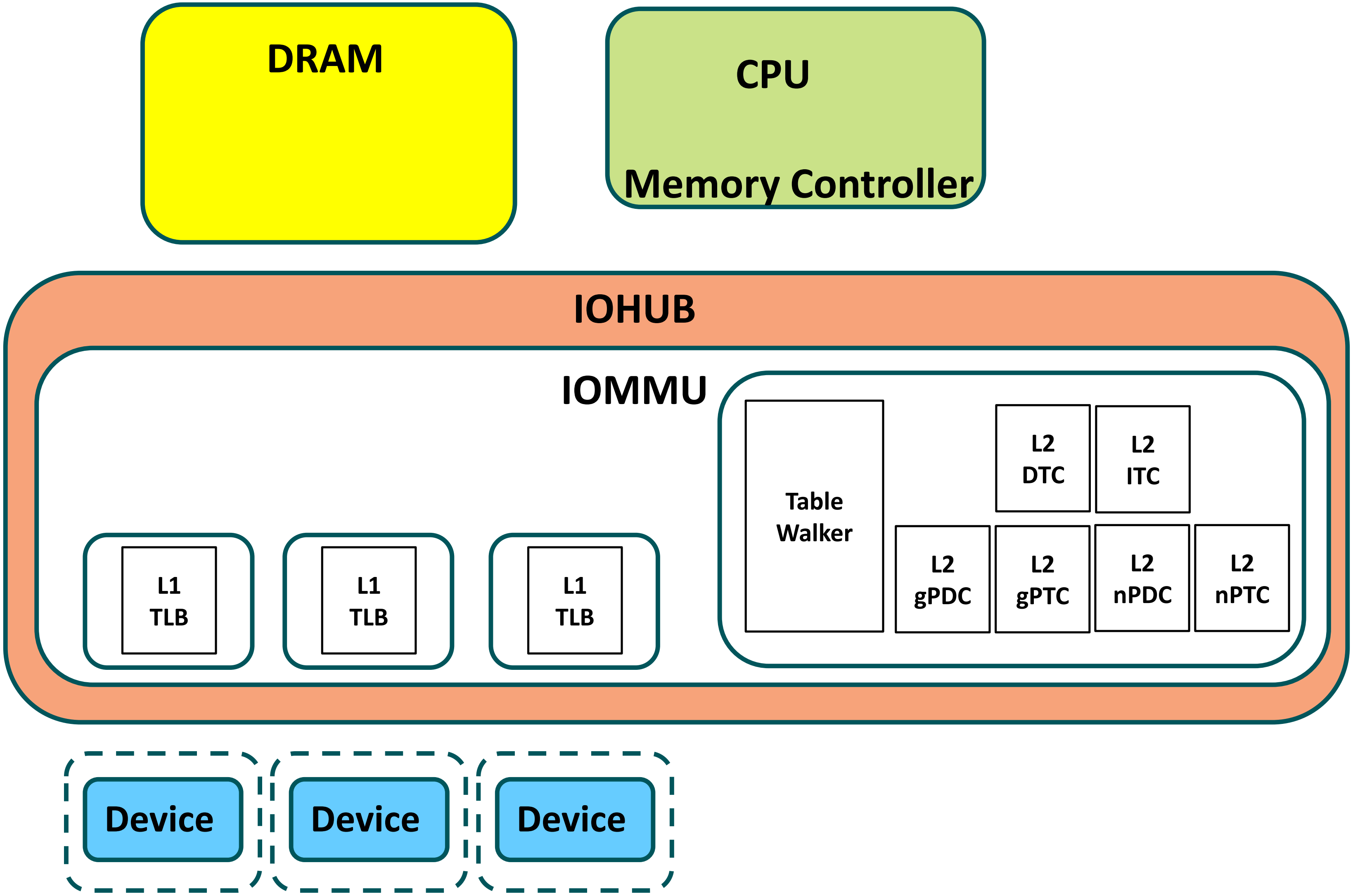


INTERRUPT VIRTUALIZATION



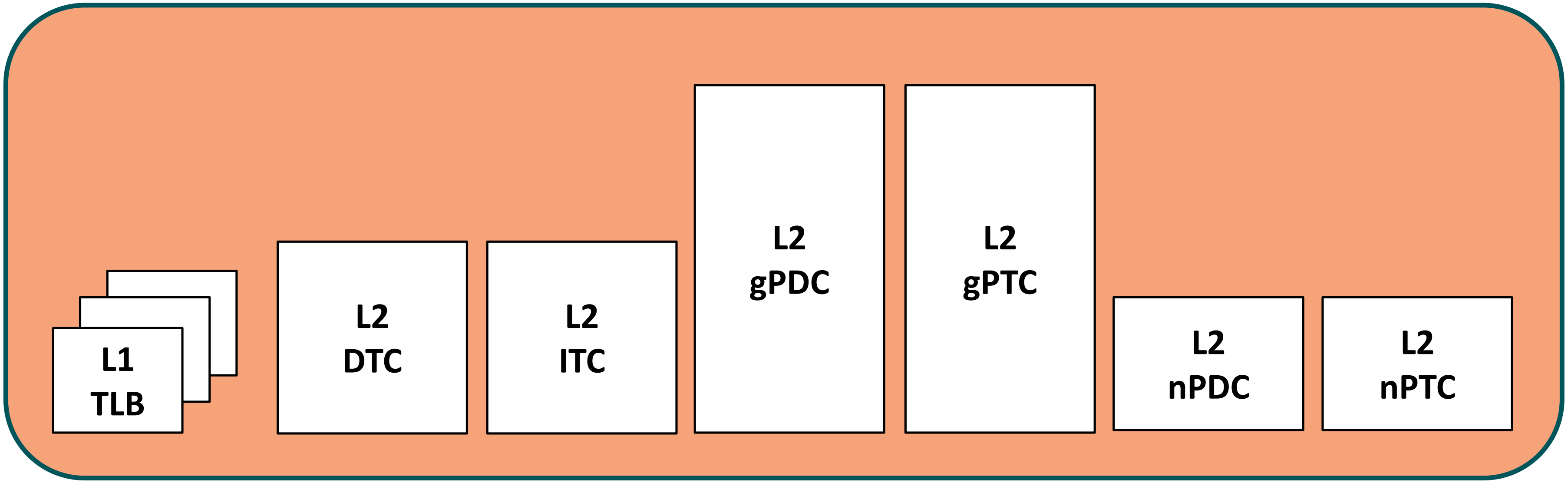
IOMMU INTERNALS: A TYPICAL IOMMU HARDWARE DESIGN

EXAMPLE OF IOMMU HARDWARE DESIGN



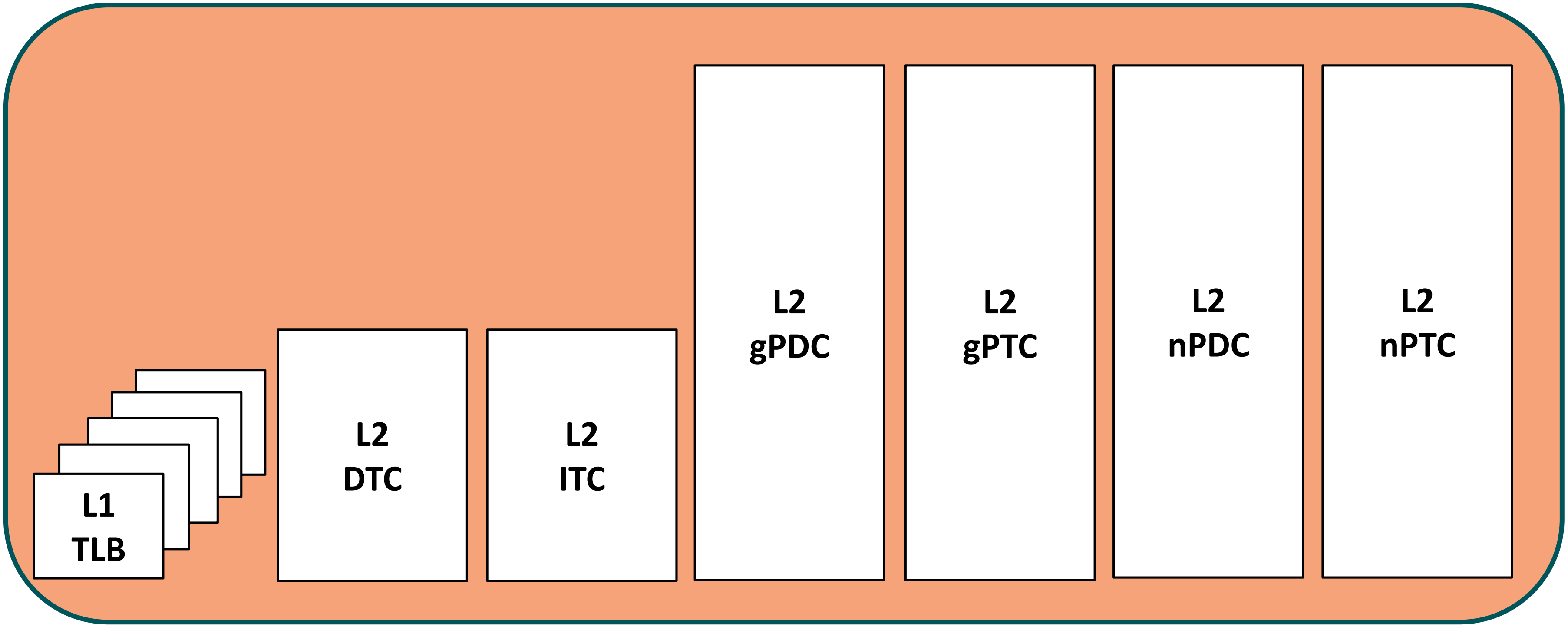
CACHE SIZING VS PRODUCT TYPE

- Typical Client Product
  - Non-Virtualized
  - I/O Isolation
  - Small Working Set



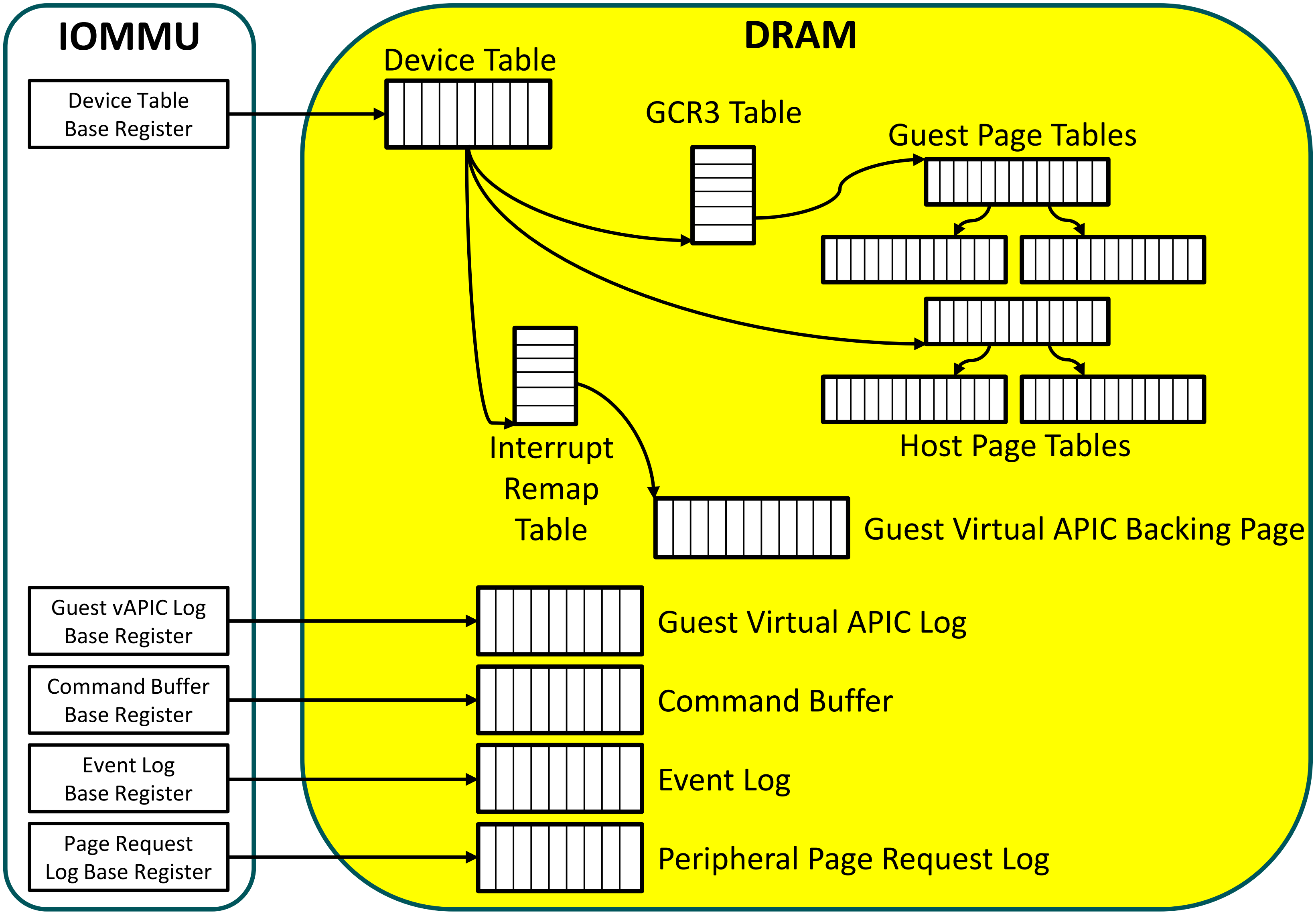
CACHE SIZING VS PRODUCT TYPE

- Typical Server Product
  - Virtualized
  - Large Working Set



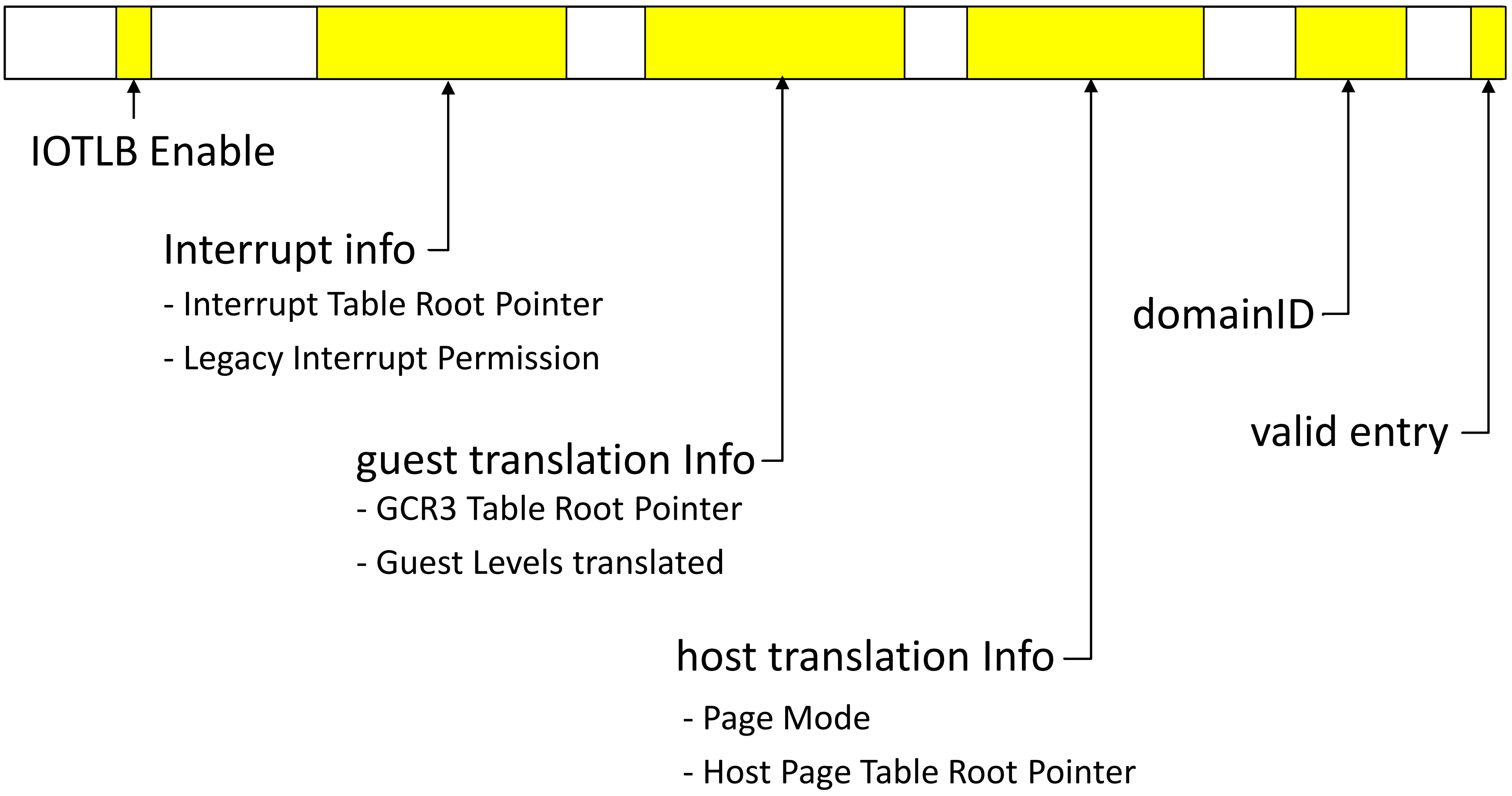
IOMMU INTERNALS: SUMMARY OF KEY DATA STRUCTURES

IOMMU'S KEY DATA STRUCTURES


IOMMU
Device Table Base Register
Guest vAPIC Log Base Register
Command Buffer Base Register
Event Log Base Register
Page Request Log Base Register
DRAM
Device Table
GCR3 Table
Guest Page Tables
Host Page Tables
Interrupt Remap Table
Guest Virtual APIC Backing Page
Guest Virtual APIC Log
Command Buffer
Event Log
Peripheral Page Request Log

DEVICE TABLE ENTRY

Each entry is 32B



IOTLB Enable

Interrupt info

- Interrupt Table Root Pointer
- Legacy Interrupt Permission

guest translation Info

- GCR3 Table Root Pointer
- Guest Levels translated

host translation Info

- Page Mode
- Host Page Table Root Pointer

domainID

valid entry

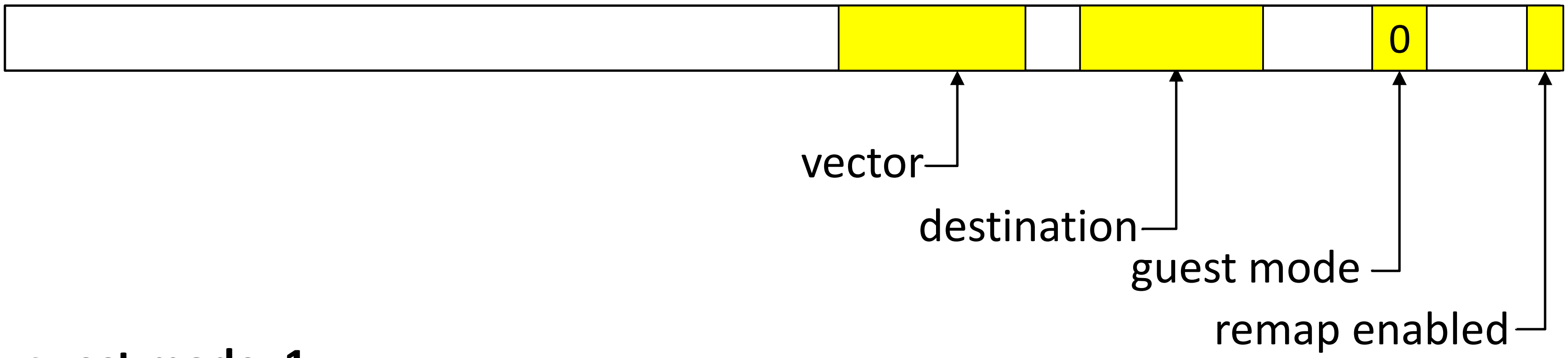
INTERRUPT REMAPPING TABLE ENTRY

Each entry is 128b. Two modes:

Interrupt Remapping (guest mode=0)

Interrupt Virtualization (guest mode=1)

guest mode=0:



0

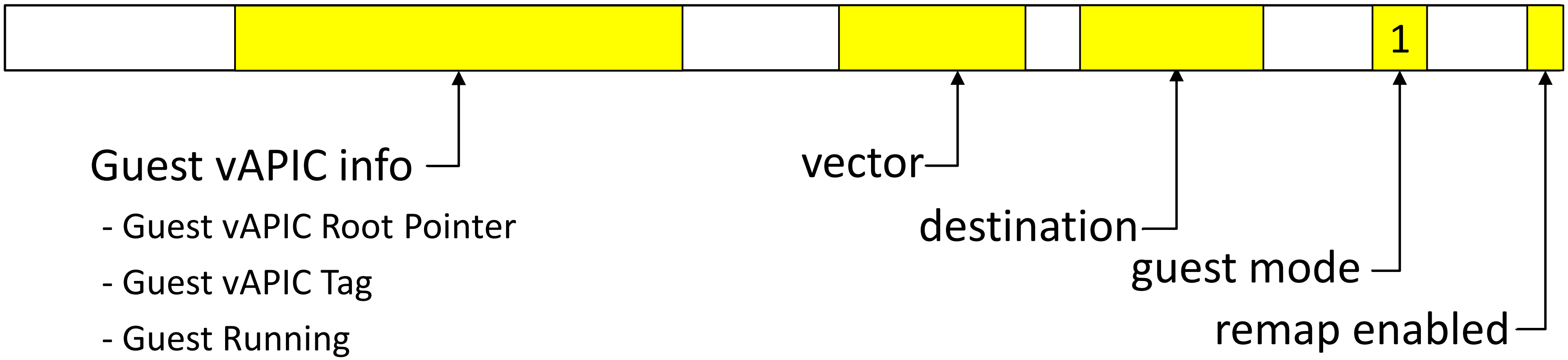
vector

destination

guest mode

remap enabled

guest mode=1:



1

Guest vAPIC info

- Guest vAPIC Root Pointer
- Guest vAPIC Tag
- Guest Running

vector

destination

guest mode

remap enabled

AGENDA

RESEARCH

Research Opportunities and Tools

RESEARCH DIRECTIONS

- Isolation from malicious or buggy third party accelerators
  - Can IOMMU ensure protection in-presence of untrusted accelerators?

RESEARCH DIRECTIONS

- Isolation from malicious or buggy third party accelerators
  - Can IOMMU ensure protection in-presence of untrusted accelerators?
- Specializing IOMMU for performance and power
  - Can IOMMU hardware exploit predictable access pattern of some accelerators?

RESEARCH DIRECTIONS

- Isolation from malicious or buggy third party accelerators
  - Can IOMMU ensure protection in-presence of untrusted accelerators?
- Specializing IOMMU for performance and power
  - Can IOMMU hardware exploit predictable access pattern of some accelerators?
- Trading memory protection for performance

RESEARCH DIRECTIONS

- Isolation from malicious or buggy third party accelerators
  - Can IOMMU ensure protection in-presence of untrusted accelerators?
- Specializing IOMMU for performance and power
  - Can IOMMU hardware exploit predictable access pattern of some accelerators?
- Trading memory protection for performance
  - Can selectively lowering protection enable better performance?

RESEARCH DIRECTIONS

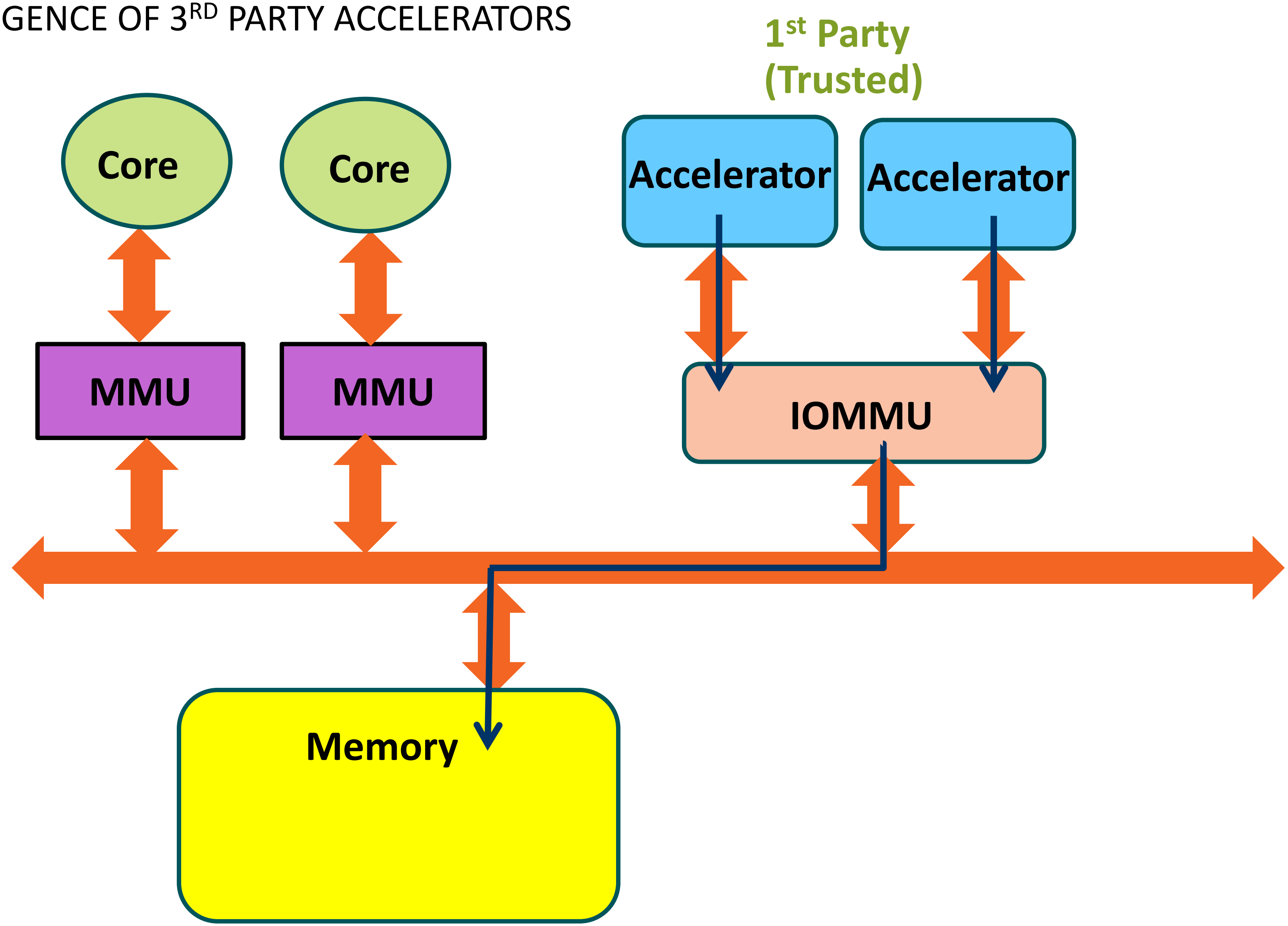
- Isolation from malicious or buggy third party accelerators
  - Can IOMMU ensure protection in-presence of untrusted accelerators?
- Specializing IOMMU for performance and power
  - Can IOMMU hardware exploit predictable access pattern of some accelerators?
- Trading memory protection for performance
  - Can selectively lowering protection enable better performance?
- Extending (limited) virtual memory to embedded accelerators
  - Can we design for IOMMU^{LITE} embedded low-power accelerators?

RESEARCH DIRECTIONS

- Isolation from malicious or buggy third party accelerators
  - Can IOMMU ensure protection in-presence of untrusted accelerators?
- Specializing IOMMU for performance and power
  - Can IOMMU hardware exploit predictable access pattern of some accelerators?
- Trading memory protection for performance
  - Can selectively lowering protection enable better performance?
- Extending (limited) virtual memory to embedded accelerators
  - Can we design for IOMMU$^{\text{LITE}}$ embedded low-power accelerators?
- Avoiding interference in the IOMMU
  - How to reduce interference among multiple devices accessing IOMMU?

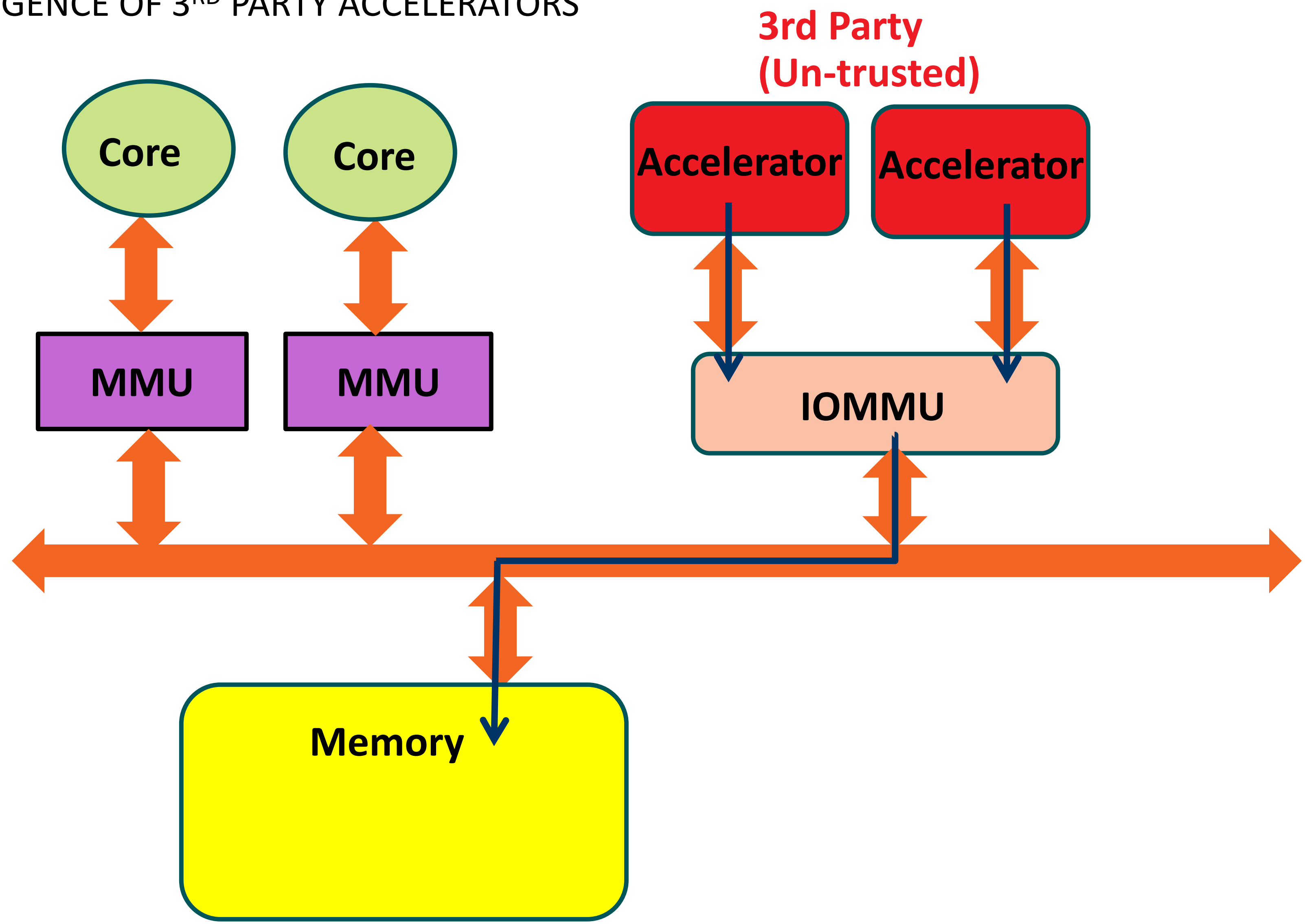
ISOLATION FROM THIRD PARTY ACCELERATORS

EMERGENCE OF 3RD PARTY ACCELERATORS



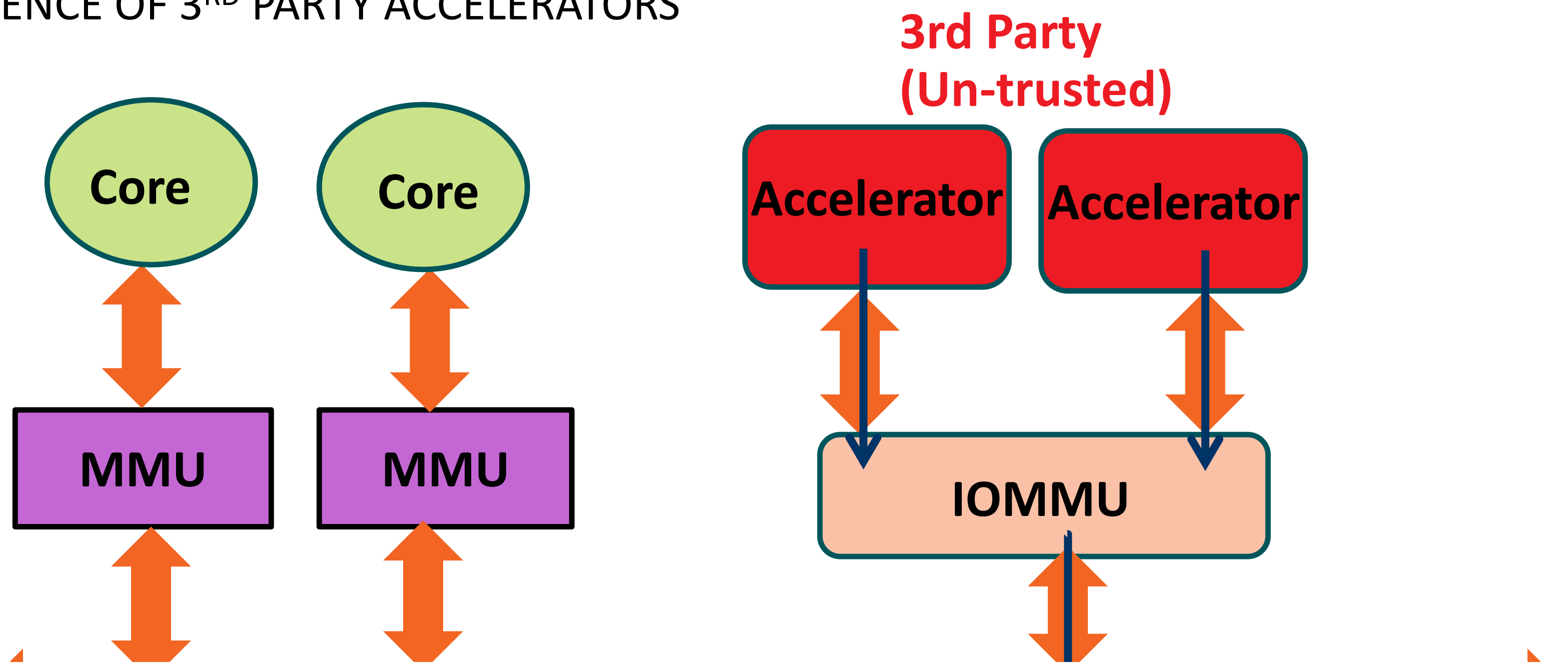
ISOLATION FROM THIRD PARTY ACCELERATORS

EMERGENCE OF 3RD PARTY ACCELERATORS



ISOLATION FROM THIRD PARTY ACCELERATORS

EMERGENCE OF 3RD PARTY ACCELERATORS



Q: How to integrate third party accelerators efficiently and securely?

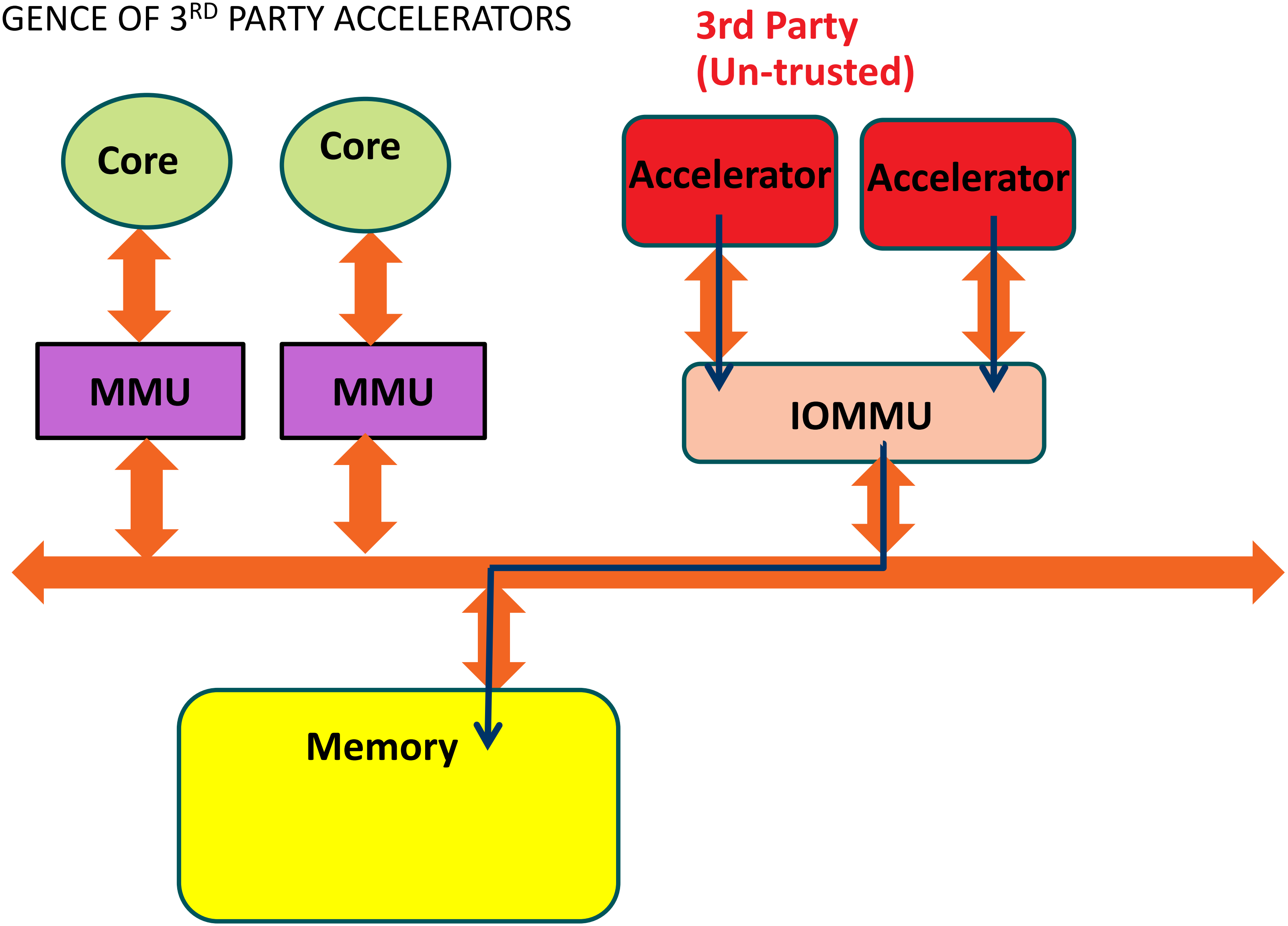
How to determine if a device is trustworthy and remains trustworthy?

May not be possible verify if 3rd party accelerator is not buggy.

ISOLATION FROM THIRD PARTY ACCELERATORS (CNTD.)

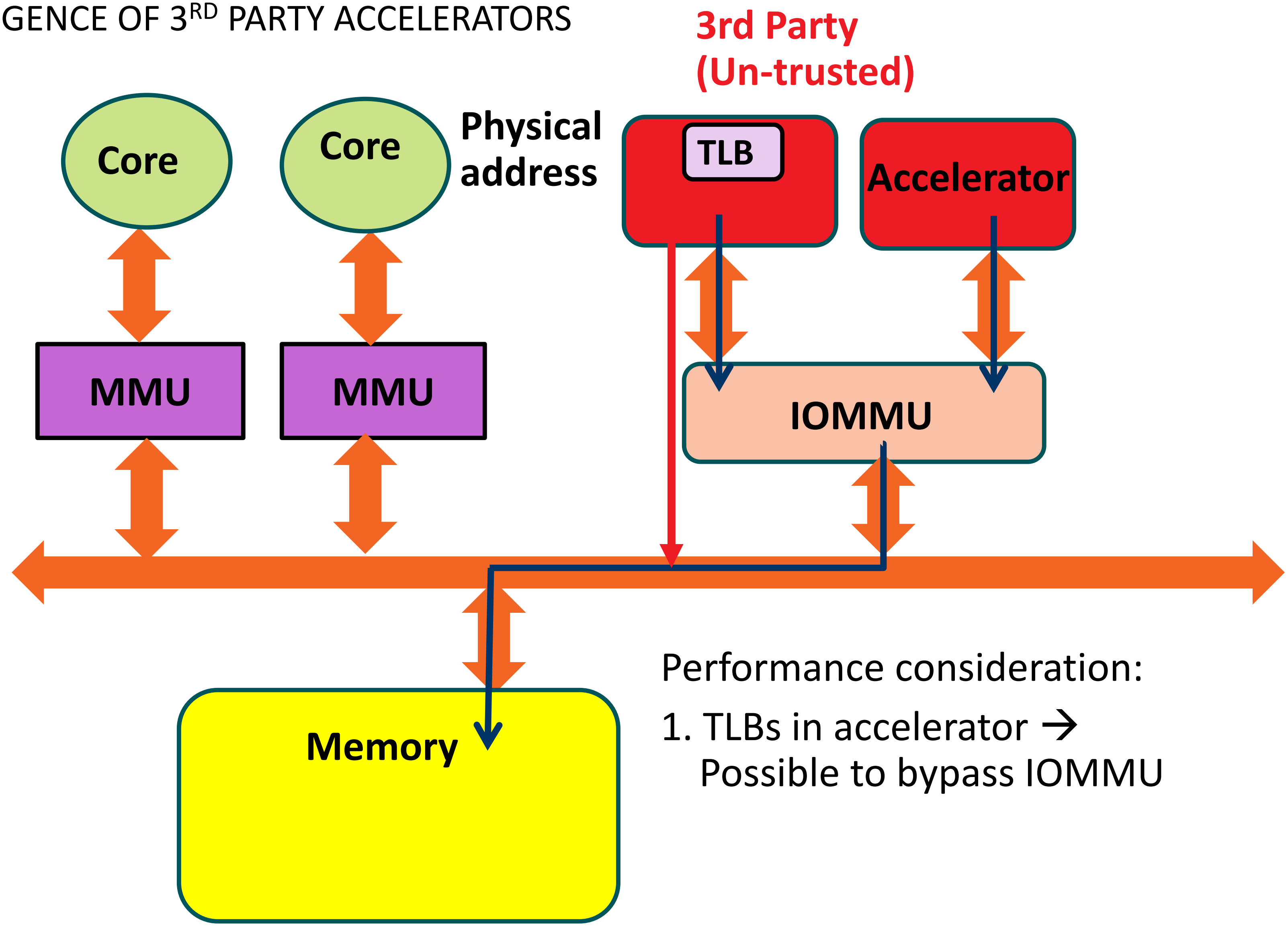
AMD

EMERGENCE OF 3RD PARTY ACCELERATORS



ISOLATION FROM THIRD PARTY ACCELERATORS (CNTD.)

EMERGENCE OF 3RD PARTY ACCELERATORS



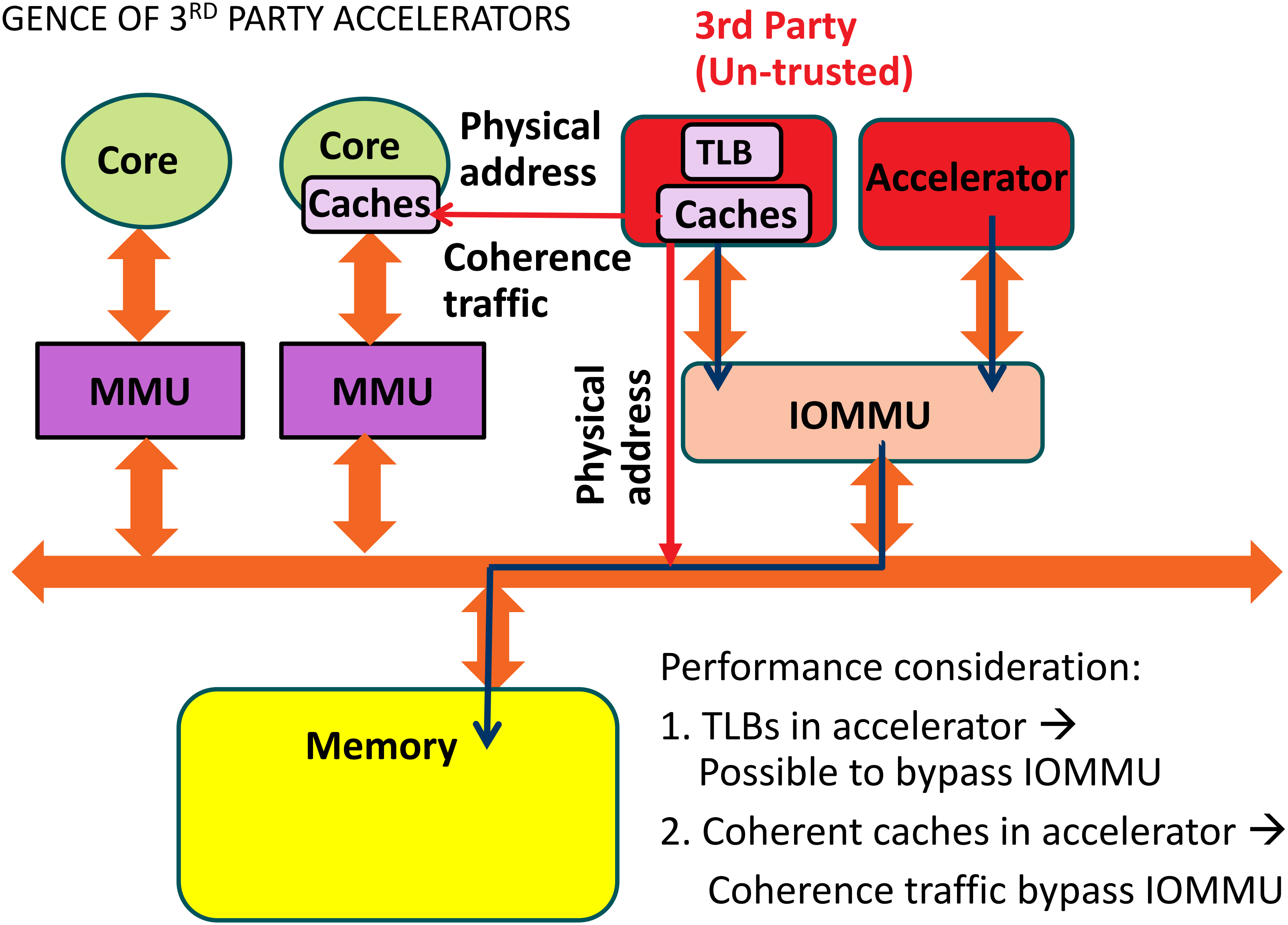
Performance consideration:

1. TLBs in accelerator → Possible to bypass IOMMU

ISOLATION FROM THIRD PARTY ACCELERATORS (CNTD.)

AMD

EMERGENCE OF 3^{RD} PARTY ACCELERATORS

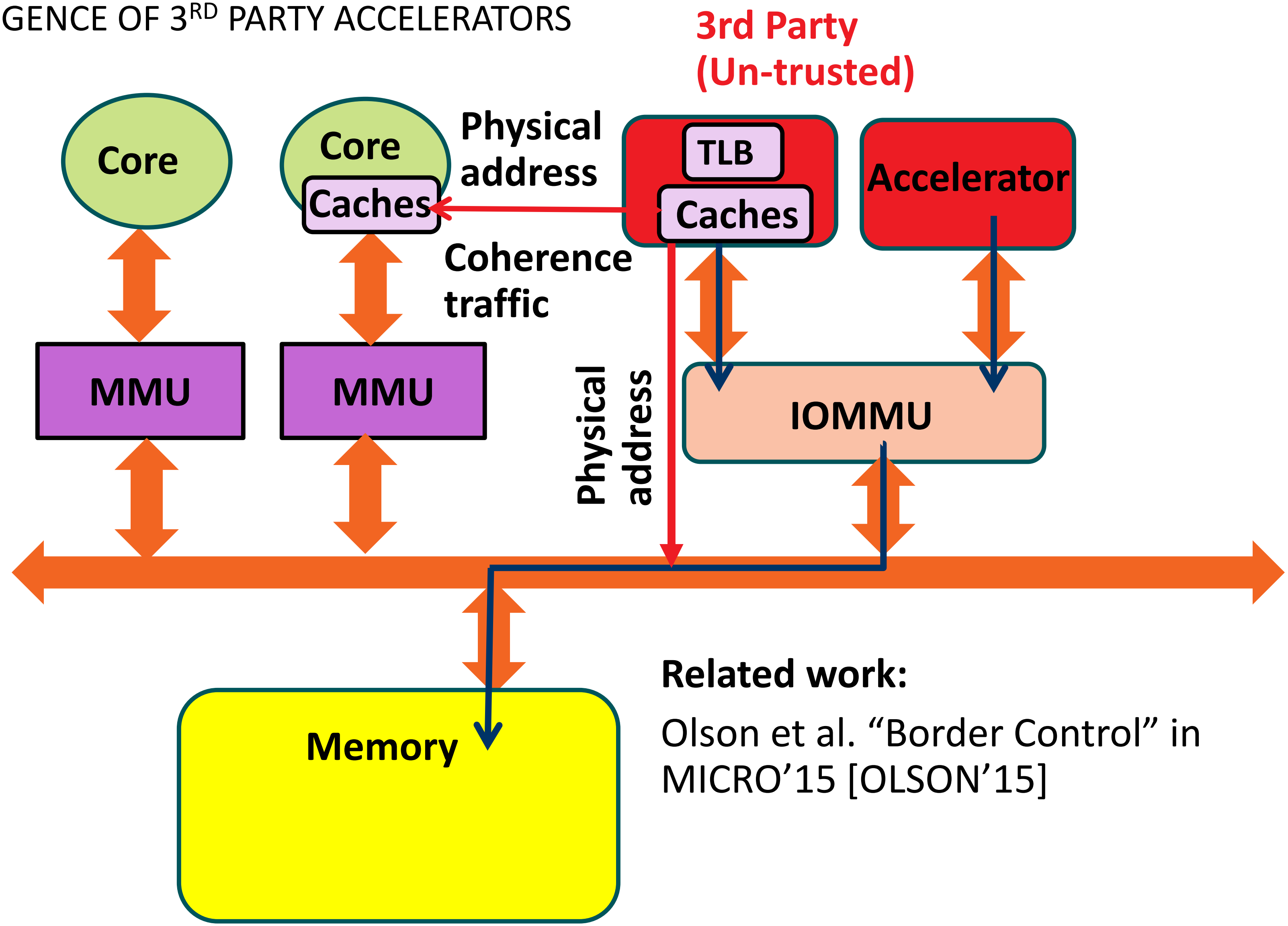


Performance consideration:

1. TLBs in accelerator → Possible to bypass IOMMU
2. Coherent caches in accelerator → Coherence traffic bypass IOMMU

ISOLATION FROM THIRD PARTY ACCELERATORS (CNTD.)

EMERGENCE OF 3RD PARTY ACCELERATORS

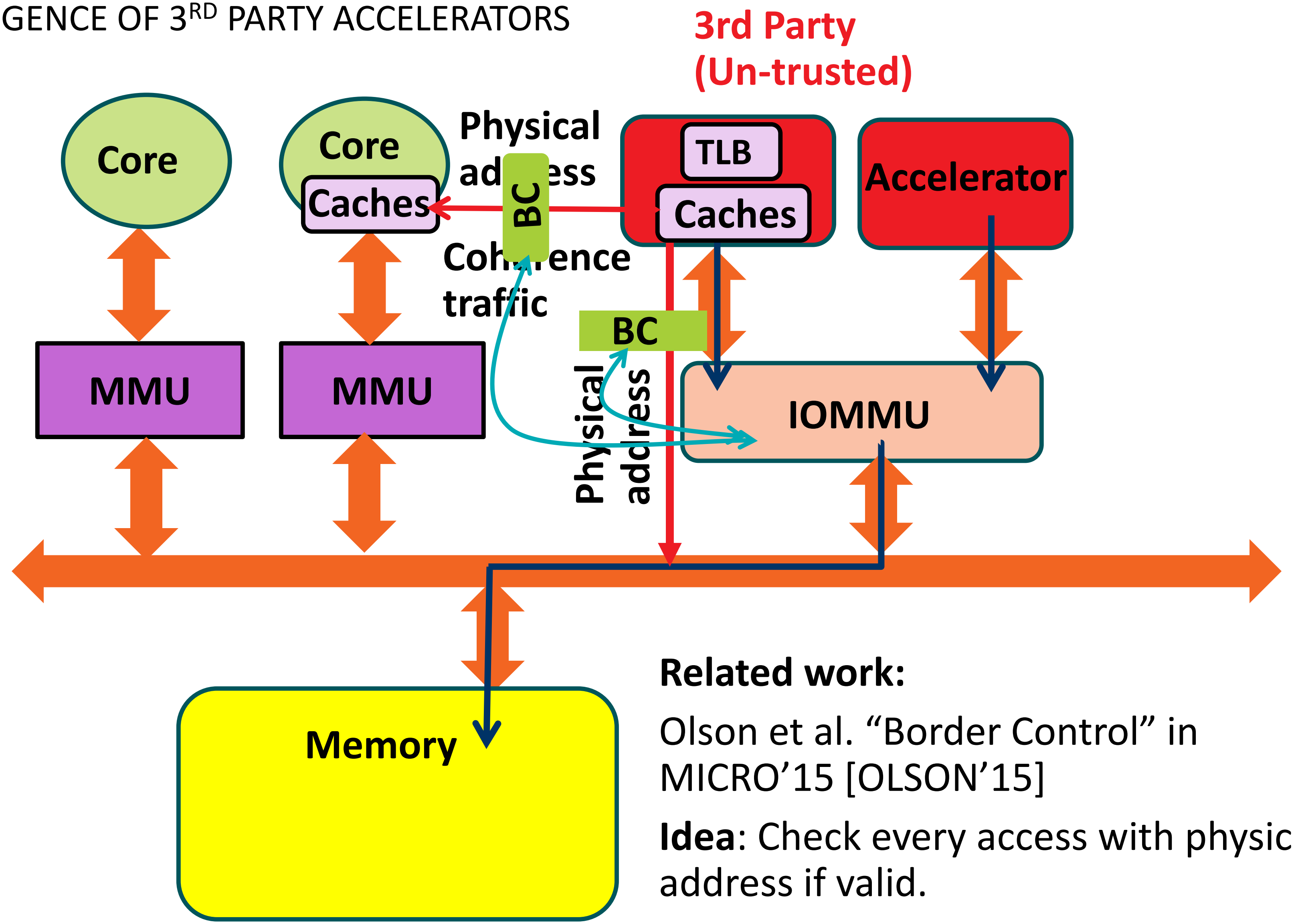


Related work:

Olson et al. "Border Control" in MICRO'15 [OLSON'15]

ISOLATION FROM THIRD PARTY ACCELERATORS (CNTD.)

EMERGENCE OF 3RD PARTY ACCELERATORS



Related work:

Olson et al. “Border Control” in MICRO’15 [OLSON’15]

Idea: Check every access with physical address if valid.

SPECIALIZING IOMMU FOR DEVICE/ ACCELERATOR

- IOMMU design(s) resembles CPU MMU design
  - But device/accelerator access patterns differs from CPU's
- IOMMU caters to disparate devices
  - Single design point may not be optimal for all
  - e.g., access pattern from GPU likely different from NIC's

SPECIALIZING IOMMU FOR DEVICE/ ACCELERATOR

AMD

- IOMMU design(s) resembles CPU MMU design
  - But device/accelerator access patterns differs from CPU’s
- IOMMU caters to disparate devices
  - Single design point may not be optimal for all
  - e.g., access pattern from GPU likely different from NIC’s

Study traffic pattern to IOMMU and specialize for common patterns

- **Related work**: Malka et al. ’s “rIOMMU” in ASPLOS’15.
  - **Idea**: Exploit ***predictable*** IOMMU accesses from devices using circular ring buffers

SPECIALIZING IOMMU FOR DEVICE/ ACCELERATOR

AMD

- IOMMU design(s) resembles CPU MMU design
  - But device/accelerator access patterns differs from CPU's
- IOMMU caters to disparate devices
  - Single design point may not be optimal for all
  - e.g., access pattern from GPU likely different from NIC's

Study traffic pattern to IOMMU and specialize for common patterns

- **Related work**: Malka et al. 's "rIOMMU" in ASPLOS'15.
  - **Idea**: Exploit ***predictable*** IOMMU accesses from devices using circular ring buffers
  - Replace page table with circular, flat table → Easy page walk
  - Predictable access → single entry IOTLB with no TLB miss and less invalidation

SPECIALIZING IOMMU FOR DEVICE/ ACCELERATOR

- IOMMU design(s) resembles CPU MMU design
  - But device/accelerator access patterns differs from CPU's
- IOMMU caters to disparate devices
  - Single design point may not be optimal for all
  - e.g., access pattern from GPU likely different from NIC's

Study traffic pattern to IOMMU and specialize for common patterns

- **Related work**: Malka et al. 's "rIOMMU" in ASPLOS'15.
  - **Idea**: Exploit ***predictable*** IOMMU accesses from devices using circular ring buffers
  - Replace page table with circular, flat table → Easy page walk
  - Predictable access → single entry IOTLB with no TLB miss and less invalidation
- Possible to use device-specific knowledge to optimize performance
  - IOMMU prefetching and TLB caching hints can be useful
  - Replacement policy coordination between IOTLB (Device TLB) and IOMMU TLB
  - Energy/power optimization in IOMMU

TRADING PROTECTION FOR PERFORMANCE

- IOMMU hardware allows lowering protection for performance
  - For example: pre-translated DMA transactions pass-through IOMMU
  - A *trusted* IO device can manipulate any address, including interrupt storms

TRADING PROTECTION FOR PERFORMANCE

- IOMMU hardware allows lowering protection for performance
  - For example: pre-translated DMA transactions pass-through IOMMU
  - A *trusted* IO device can manipulate any address, including interrupt storms

- OS policies for trading off protection for security
  - Should the sysadmin decide how much to trust a device/driver?
  - Exposing software knobs for dialing performance vs. protection
  - **Related work:** OS policies for ***Strict* vs *Deferred*** protection strategy [WILMANN’08, BEN-YEHUDA’07, AMIT’11]
  - **ASPLOS’16:** Strict, sub-page grain protection through Shadow DMA-buffer [MARKUZE’16]

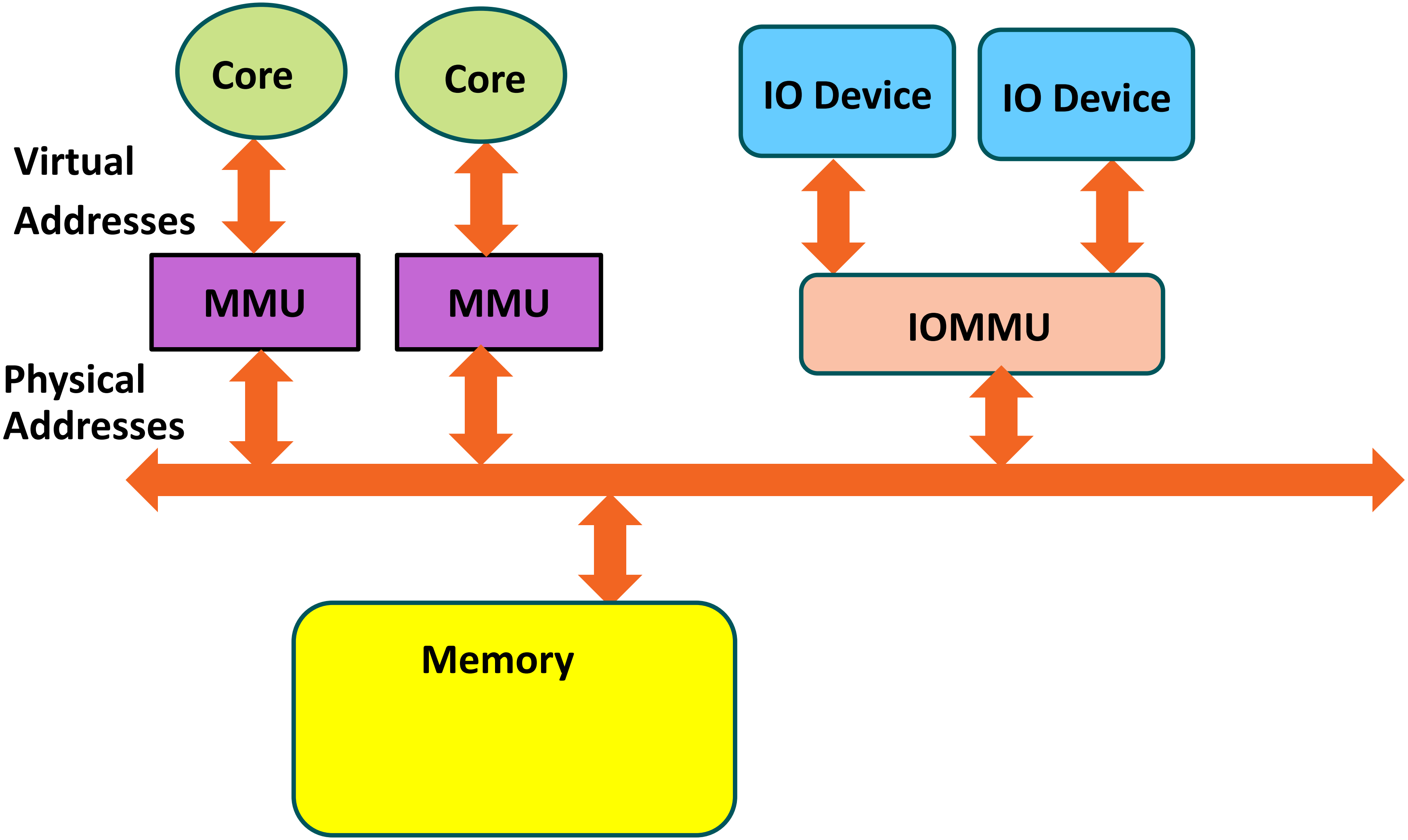
IOMMU$^{\text{LITE}}$ FOR EMBEDDED LOW-POWER ACCELERATORS

- Virtual memory eases programming (e.g., “pointer-is-pointer”)
  - But comes at performance and energy cost
- Stripped-down IOMMU for **ultra low-power** accelerators
  - Lower hardware, performance, power cost by stripping non-essential features
  - Example “non-essential” features: IO virtualization support, Interrupt remapping, Page fault handling, Nested page table walker, etc.

IOMMU^{LITE} FOR EMBEDDED LOW-POWER ACCELERATORS

- Virtual memory eases programming (e.g., “pointer-is-pointer”)
  - But comes at performance and energy cost
- Stripped-down IOMMU for **ultra low-power** accelerators
  - Lower hardware, performance, power cost by stripping non-essential features
  - Example “non-essential” features: IO virtualization support, Interrupt remapping, Page fault handling, Nested page table walker, etc.
- **Related work**:
  - Vogel et al.’s “Lightweight Virtual Memory” in CODES’15 [VOGEL’15]
    - Idea: Software managed IOMMU for FPGA → No translation miss handling in hardware
    - Simple design, high performance with effective software management

AVOIDING (DESTRUCTIVE-) INTERFERENCE IN IOMMU



Core

Core

IO Device

IO Device

Virtual Addresses

MMU

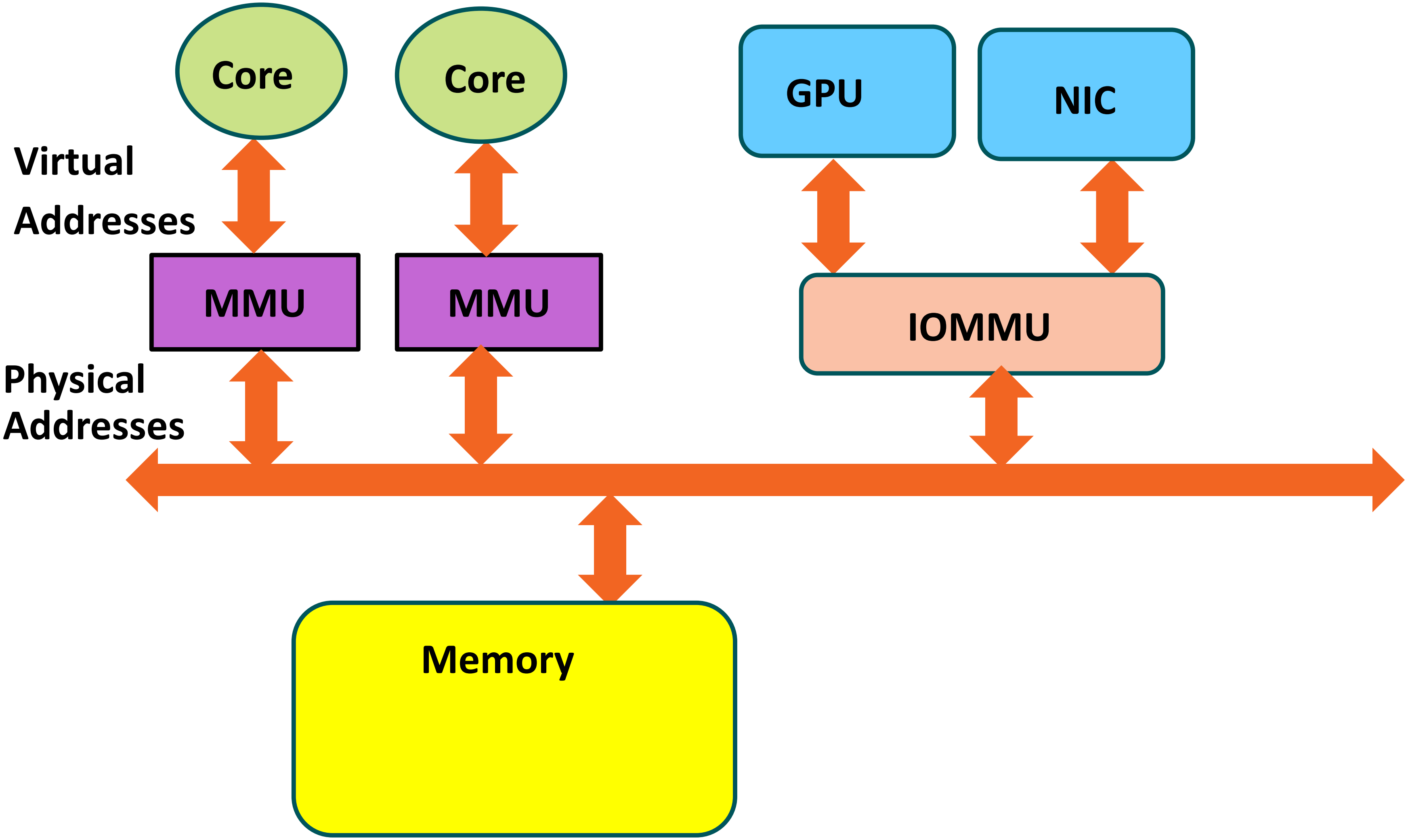
MMU

IOMMU

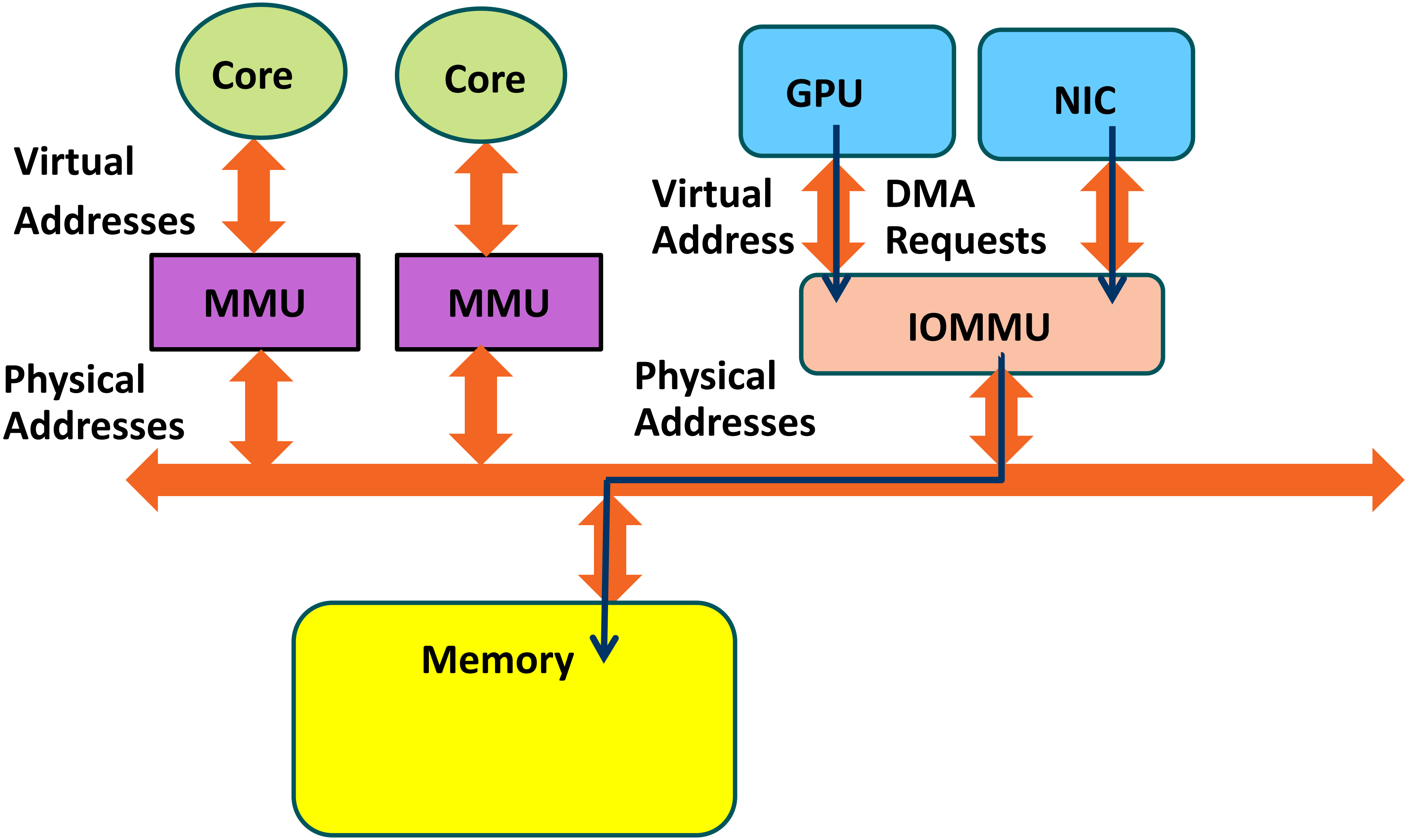
Physical Addresses

Memory

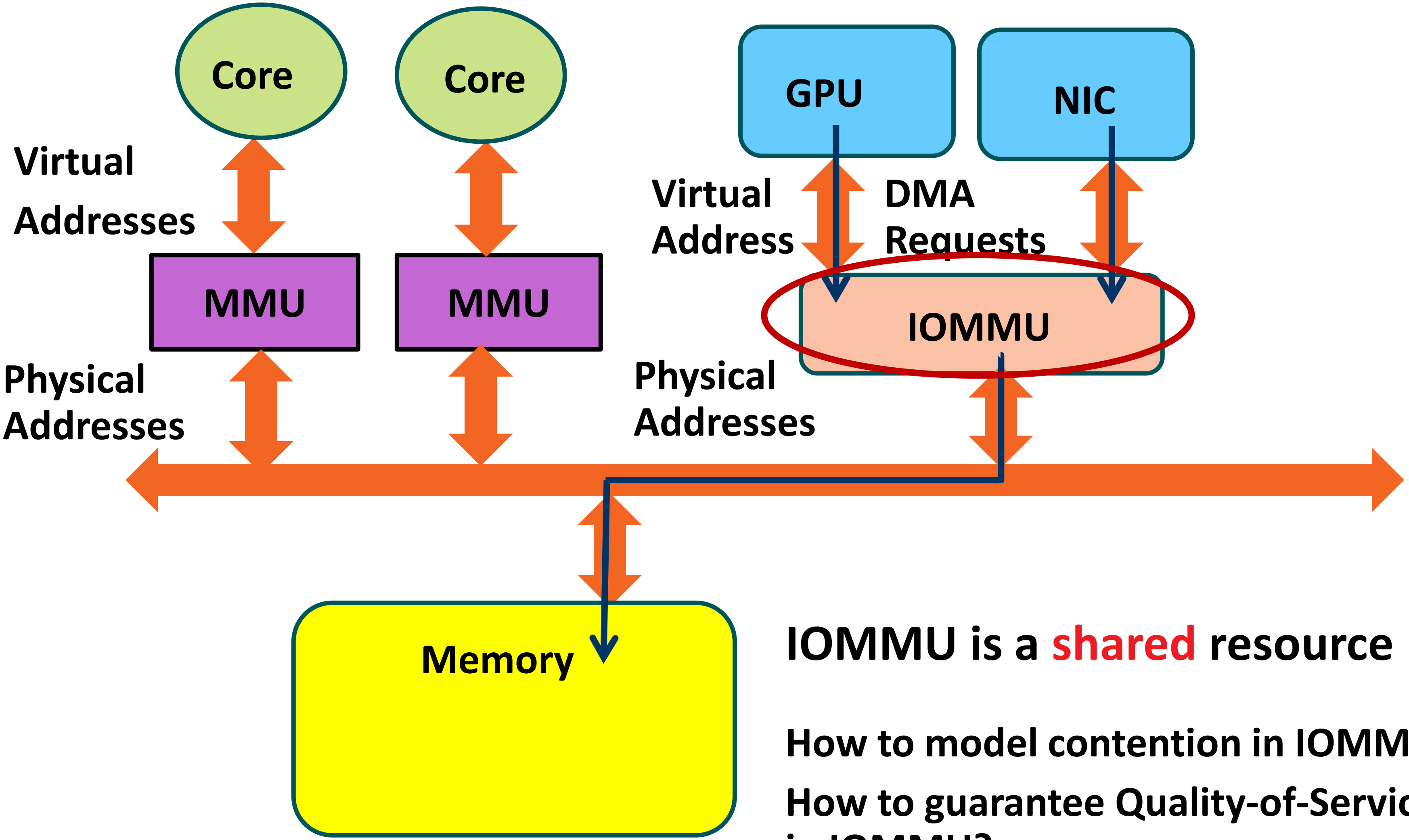
AVOIDING (DESTRUCTIVE-) INTERFERENCE IN IOMMU



AVOIDING (DESTRUCTIVE-) INTERFERENCE IN IOMMU



AVOIDING (DESTRUCTIVE-) INTERFERENCE IN IOMMU



IOMMU is a shared resource

How to model contention in IOMMU?

How to guarantee Quality-of-Service in IOMMU?

RESEARCH: TOOLS AND MODELING

- Software research: IOMMU driver/OS policies
  - Easy! Open source IOMMU Driver in Linux

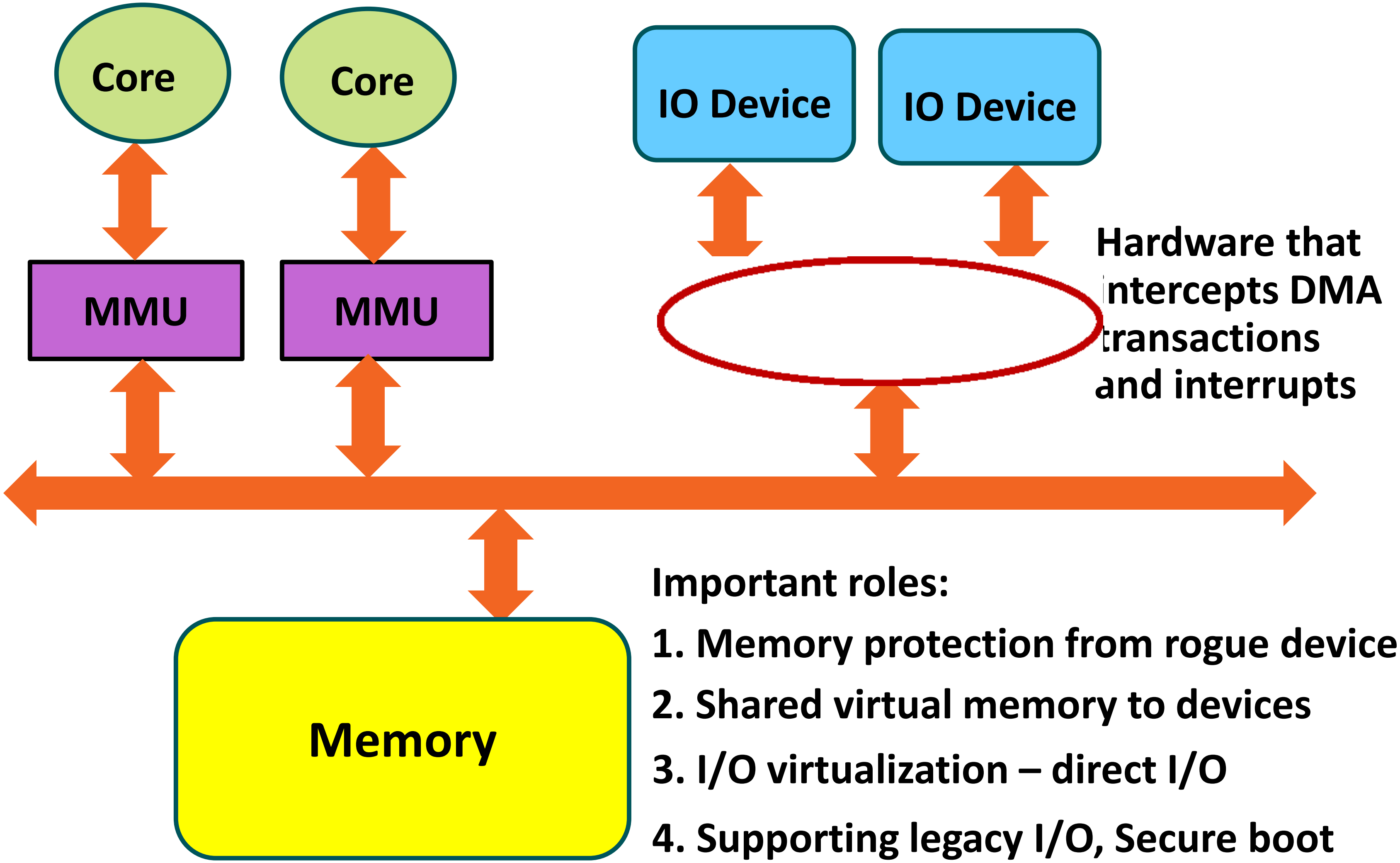
- Hardware research: Modifying IOMMU hardware behavior
  - Option 1: Hardware performance counter + Analytical models
  - Option 2: Simulator with IOMMU model
    - Work in progress to add IOMMU model in gem5
    - Write down in attendance sheet your email if interested

SUMMARY

AMD

IOMMU (kernel-mode) Driver:

Configuration/Setup IOMMU hardware



Hardware that intercepts DMA transactions and interrupts

Important roles:

1. Memory protection from rogue devices
2. Shared virtual memory to devices
3. I/O virtualization – direct I/O
4. Supporting legacy I/O, Secure boot

REFERENCES

- IOMMU specification: http://support.amd.com/TechDocs/48882_IOMMU.pdf
- OLSON’15: Lean Olson et. al. “Border Control: Sandboxing Accelerators” , MICRO 2015
- AMIT’11: Nadav Amit et al. “vIOMMU: Efficient IOMMU Emulation”, USENIX, ATC , 2011
- BEN-YEHUDA’07: Muli Ben-Yehuda et al. “The Price of Safety: Evaluating IOMMU Performance”, OLS 2007
- MALKA’15: Moshe Malka et al. “rIOMMU: Efficient IOMMU for I/O Devices That Employ Ring Buffers”, ASPLOS 2015.
- WILLMANN’08: Paul Willmann et al. “Protection Strategies for Direct Access to Virtualized I/O Devices”, USENIX, ATC 2008.
- VOGEl’15: Pirmin Vogel et. al. “Lightweight virtual memory support for many-core accelerators in heterogeneous embedded SoCs”, CODES’15
- MARKUZE’16: Markuze et al. “True IOMMU Protection from DMA Attacks”, ASPLOS’16.

QUESTIONS AND FEEDBACK

- Reachable @
  - Arka Basu: Arkaprava “dot” Basu “at” amd.com
  - Andy Kegel: Andrew “dot” Kegel “at” amd.com
  - Paul Blinzer: Paul “dot” Blinzer “at” amd.com
  - Maggie Chan: Maggie “dot” Chan “at” amd.com

DISCLAIMER & ATTRIBUTION

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

ATTRIBUTION