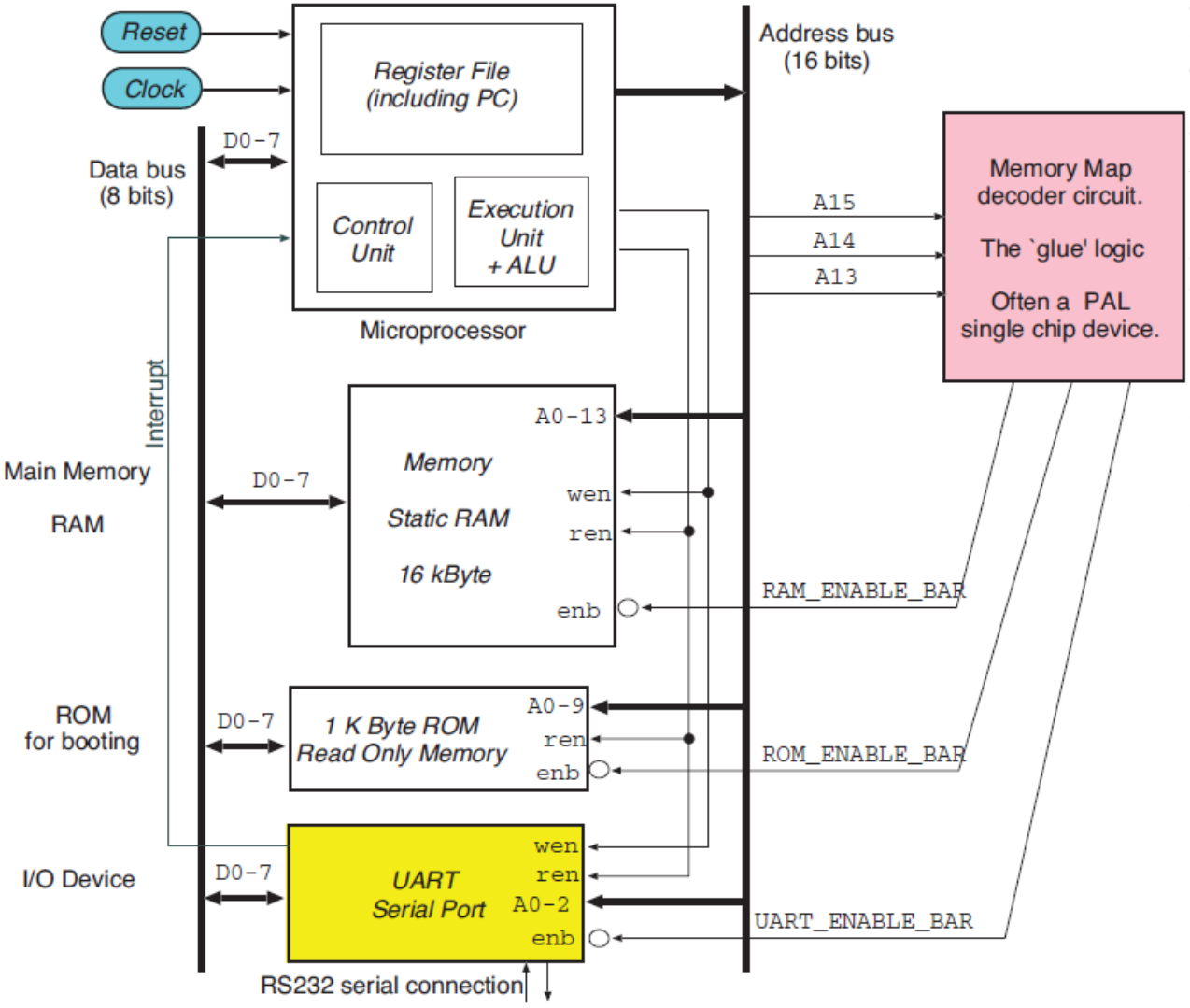
Gestione Periferiche - DMA e BUS

Andrea Bartolini – a.bartolini@unibo.it

Fin ora abbiamo detto che il processore è un blocchetto che ha delle unità funzionali all interno e che con dei fili trasferisce dati da e verso la memoria.

Architettura dei calcolatori T



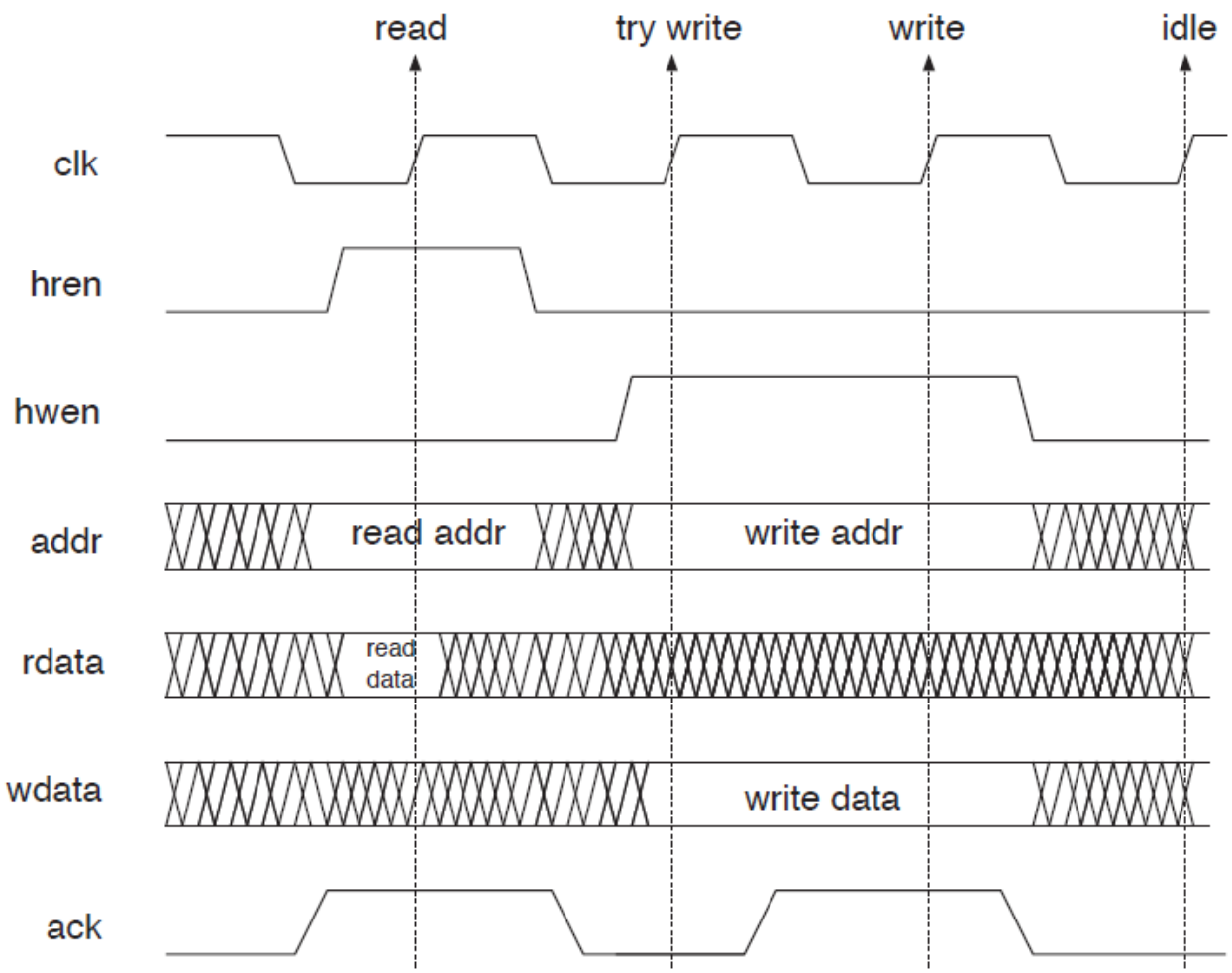
Il bus e sincrono nel senso che vengono condivisi dei segnali di handshake e il segnale di clock tra i due attori della comunicazione

Quando si parla di spazio di indirizzamento, si fa riferimento alla larghezza del bus degli indirizzi, e quando si parla di datapath con un certo parallelismo (32, 64,.. bit) si fa riferimento al parallelismo del bus dati.
C'è poi una logica di mapping che definisce i segnali di enable sulle ferie periferiche che erano funzione degli indirizzi che venivano emessi.
Questo tipo di bus non è più usato, se non s livello di connessione di componenti su una scheda madre. Questo per il fatto che la logica threestate è onerosa in termini di consumo.
Al giorno d'oggi in un sistema on chip, o in una situazione in cui più core sono interconnessi si utilizzano interconnessioni punto punto. Tipicamente queste connessioni vengono fatte con bus sincroni che connettono due punti di comunicazione, in cui uno è master e l'altro è slave.
Quando il processore vuole parlare con una periferica o con la memoria inizia una transazione sul bus, la transazione é sincrona, ed e sempre il master che può decidere di scrivere o leggere verso un dispositivo che è il target della comunicazione.
Quindi la comunicazione viene sempre iniziata da uno dei due attori

Tri-state bus not used in on-chip interconnect.

Point-to-point wiring are preferred to achieve a lower switched-capacitance and reducing the leakage energy in logic gates where the input is floating between logic levels.

Simple syncronous bus



Read followed by a write. The write is extended by one clock cycle as the ack signal was not present on the first positive edge when hwen was asserted.

Synchronous BUS => transactions happens on positive edge of clk signal

Handshake

1. Initiator issues hwen/ hren signals
2. Target acknowledge the command with ack signal

- addr[31..0]: select address
- hwen: asserted by initiator during a write to target operation
- hren: asserted by initiator during a read from target operation
- wdata[31..0]: data from initiator during a write operation
- rdata[31..0]: data from target during a read operation
- ack: asserted by target when ready.

Nello stadio di fetch si fa solo lettura in questo caso la memoria riceve un indirizzo e dovrà emettere un dato.

Segnale di hand shake in lettura e in scrittura. Chi vuole iniziare una comunicazione, tiene alto il segnale di handshake finché la transazione non é conclusa.
Nel caso di lettura, il processore che inizia la comunicazione, dovrà asserire il segnale di handshake di read enable. Contestualmente mette sul filo l'indirizzo rispetto al quale vuole fare l'operazione di lettura.
In una comunicazione che dev4 gestire sia operazioni di lettura che di scrittura Avremo segnali di controllo che sono pilotati dal master, un bus su cui viaggiano degli indirizzi (da master verso slave), due canali diversi, uni su cui viaggeranno i dati che vanno dalla memoria verso la cpu, è un canale che tiene i dati che vanno dalla cpu verso la memoria, poi ci sarà un altro canale di handshake di acknowledge gestito dallo slave verso il master.
I fili sono tutti unidirezionali, questo perché non si vogliono più avere porte logiche threetate che possono essere usate sia in lettura che scrittura perché costano troppo in hw, costa meno aumentare il numero di fili.

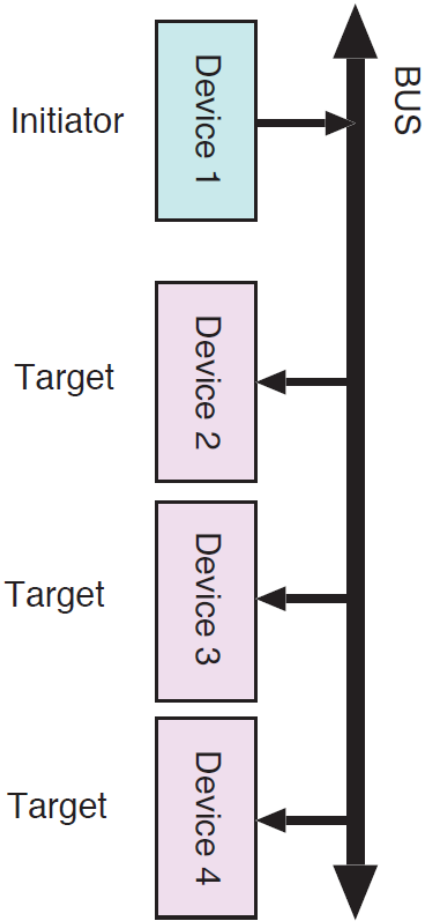
Quando un dispositivo vuole iniziare a comunicare, manda un segnale in cui comunica se vuole fare lettura o scrittura, e con i suoi tempi il dispositivo target comincia di essere pronto e la comunicazione avviene sul primo fronte di salita che c'è da quando sia il segnale di handshake é alto e sia e alto il segnale di acknowledge che dice che lo slave è pronto. Quando sono entrambi alti , nel primo fronte alto successivo inizia la comunicazione.
Nell'operazione di scrittura, chi inizia la comunicazione asserisce una richiesta di scrittura, ovvero il segnale di handshake di write enable ed emette sul canale degli indirizzi l'indirizzo associato alla locazione di memoria su cui vuole scrivere il dato e con il suo tempo lo slave (memoria) risponde con l'ack. Quando sia richiesta che ack. Sono alti e c'è fronte di salita alto del Ck, avviene il trasferimento dei dati, che possono essere campionati.

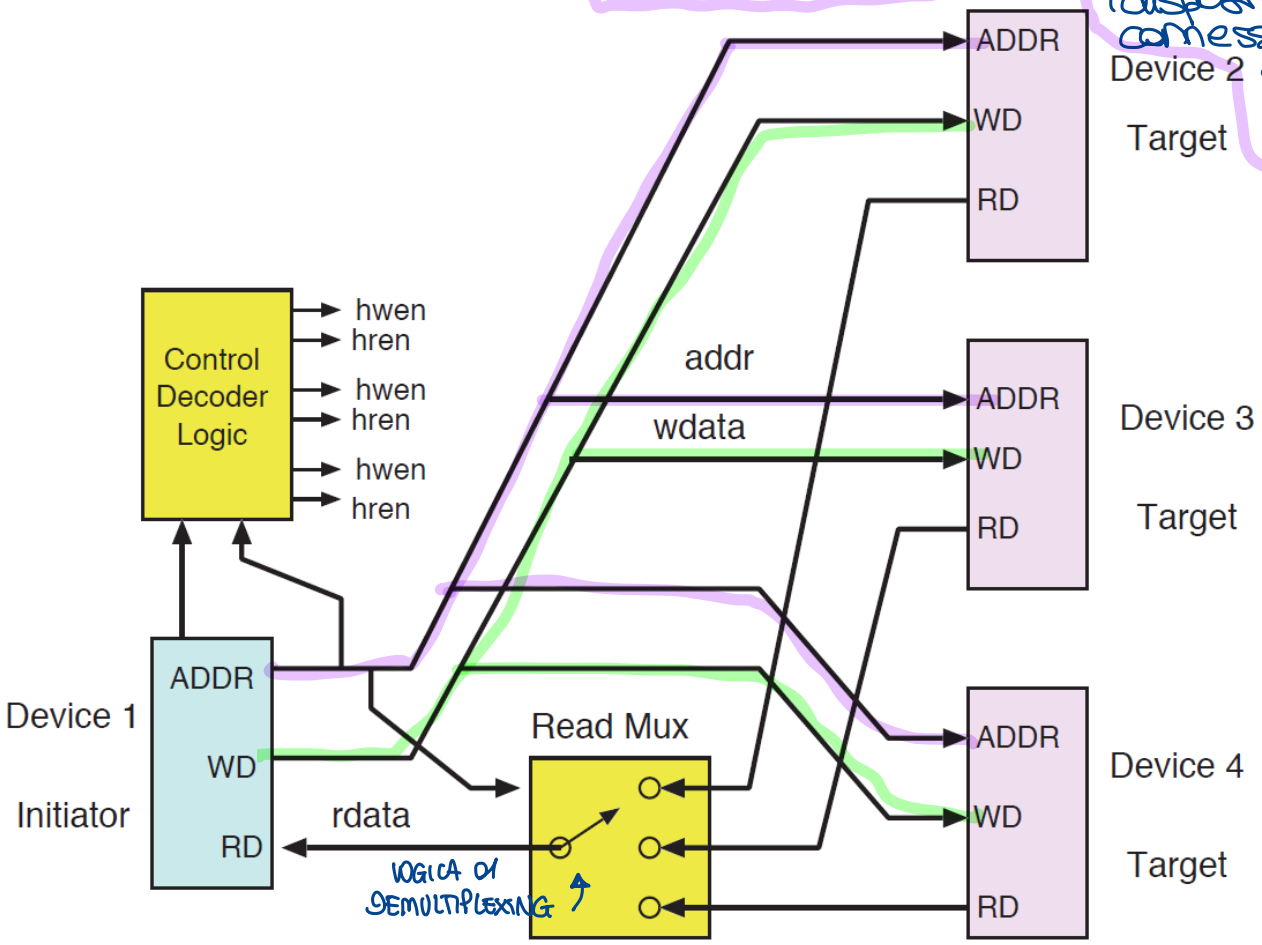
Esiste sempre un dispositivo che può essere iniziatore e slave di una comunicazione, il dispositivo ha porte di acceso distinte ed inviare e ricevere.

Bus Fabbric – Simple w. one initiator

caso 1 iniziatore e multipli target : si ha bisogno di un demux

Il canale degli indirizzi è pilotato dal dispositivo che inizia la comunicazione ed è condiviso alle porte target di tutti i dispositivi connessi sullo stesso bus





Il canale che fa muovere i dati da chi inizia la comunicazione verso i dispositivi target è connesso a tutte le porte write data delle porte target

Logica di decode: Connette elettricamente i segnali di write enable é read enable emessi dal master , in modo selettivo ai singoli dispositivi target. Avremo una logica che in funzione dell'indirizzo va a generare dei segnali di handske in lettura e scrittura specifici per ciascun target. Quindi duplico i segnali di controllo per ciascun target, che saranno attivi in modo selettivo, quindi quando voglio parlare con il dispositivo 2 verranno attivati solo i segnali di controllo che vanno verso al periferica 2 e non quelli verso 1 o 3.

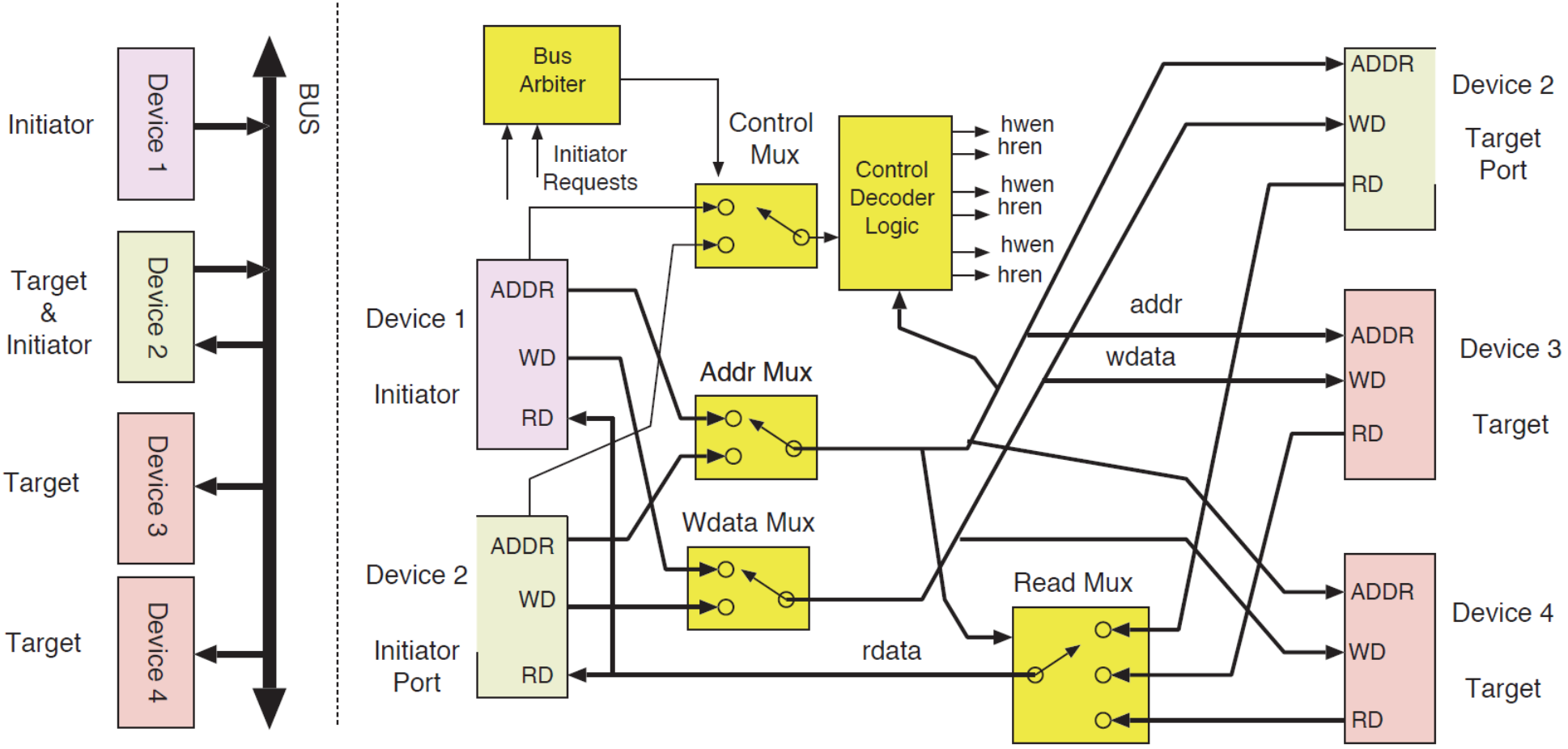
Se. Operazione di lettura. Il master emetterà un segnale di handshake read enable, questo verrà combinato con i bit di indirizzo per decidere a quale periferica di vuole comunicare. Quindi le copie di segnali di handshake di lettura e scrittura sono duplicate per ogni periferica e sono attive in modo esclusivo.

Logica di demux. Serve per il canale di risposta, permette in caso di operazione dj lettura di decidere quale dei canali di read data è elettricamente connesso al canale read data dell' inizializzatore.
Non abbiamo più canali threestate, quindi non possiamo connettere elettricamente 3 uscite sullo stesso filo. Possiamo connettere un uscita a multipli ingressi, ma non possiamo connettere più uscite allo stesso ingresso, quindi c'è bisogno di logica demux, non ci sono più bus threestate nel sistema.

Quando abbiamo più iniziatori nel sistema'? (Slide sotto)
Ci sono due dispositivi che possono decidere di iniziare la comunicazione. Sara necessario un arbitro che riceve le richieste fatte da più iniziatori e le serializza. In funzione dell'esito dell'arbitraggio avremo che solo il canale di indirizzi di 2 può essere connesso verso le periferiche, quindi si fa una logica di demux che seleziona quale campo, indirizzo e segnale di controllo viene connesso con la logica di decodifica. Poi si ha un altro demux che eve decidere quale tra i due iniziatori della comunicazione può emettere indirizzi verso le periferiche e quale dei dispositivi iniziatori potrà inviare dati in scrittura verso le periferiche. In un momento, in modo esclusivo ci sarà solo o 1 o 2 che é master dell'itnerconnect che può comunicare con le periferiche.
Il canale di risposta passa attraverso un demux, e siccome possiamo avere un uscita connessa a più ingressi, abbiamo che il segnale di risposta ad esempio nel caso di read, è commesso elettricamente sia al device 1 che 2. Solo il device che aveva iniziato la comunicazione vedrà il segnale di ack.

Questo tipo di comunicazione e bloccante

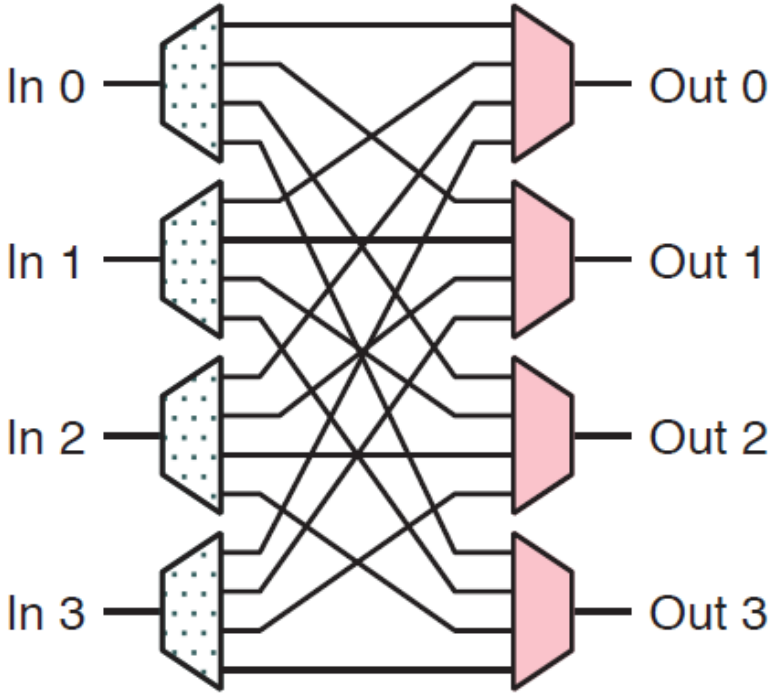
Bus Fabbric – Simple w. multiple initiator



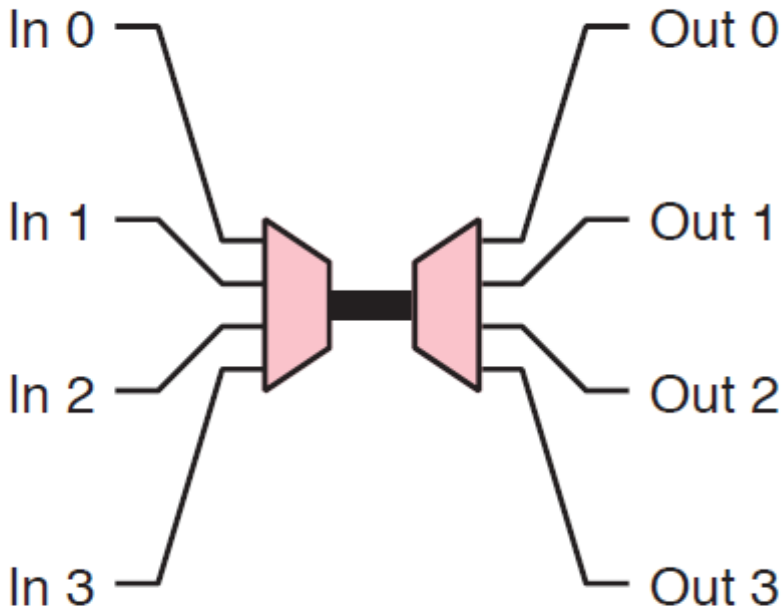
Crossbar

: Meccanismo con una serie di porte di iniziatori e una serie di porte target. Si vuole una comunicazione che non serva una sola comunicazione alla volta, ma avere contemporaneamente comunicazione tra più master e slave contemporaneamente.

Es: contemp. $1 \rightarrow 2$, $3 \rightarrow 3$



Multiplexer - demultiplexer

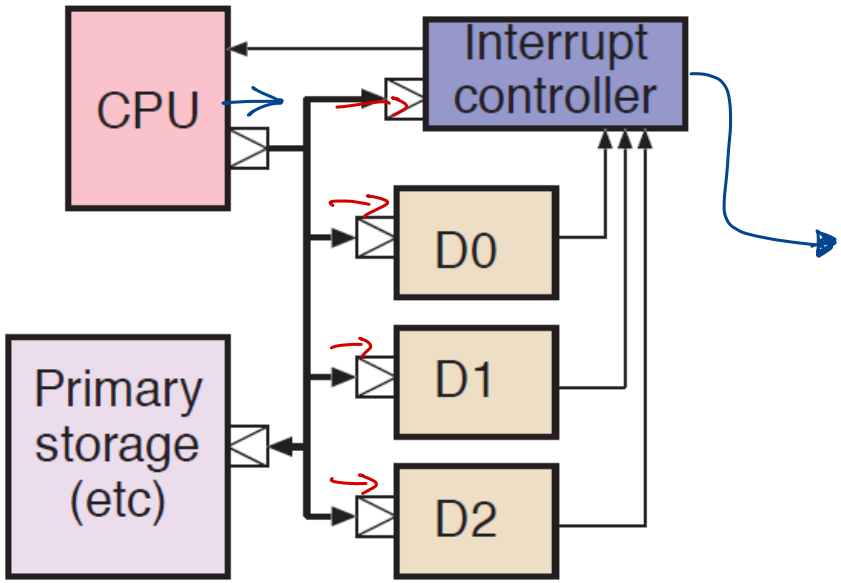


TDM: Time-division multiplexed

Canale singolo, che va a una frequenza du clock molto elevata.

Il blocco crossbar e il più usato nei SoC.
lo stesso dispositivo può avere più porte in ingresso che in uscita.

Interrupt Controllers

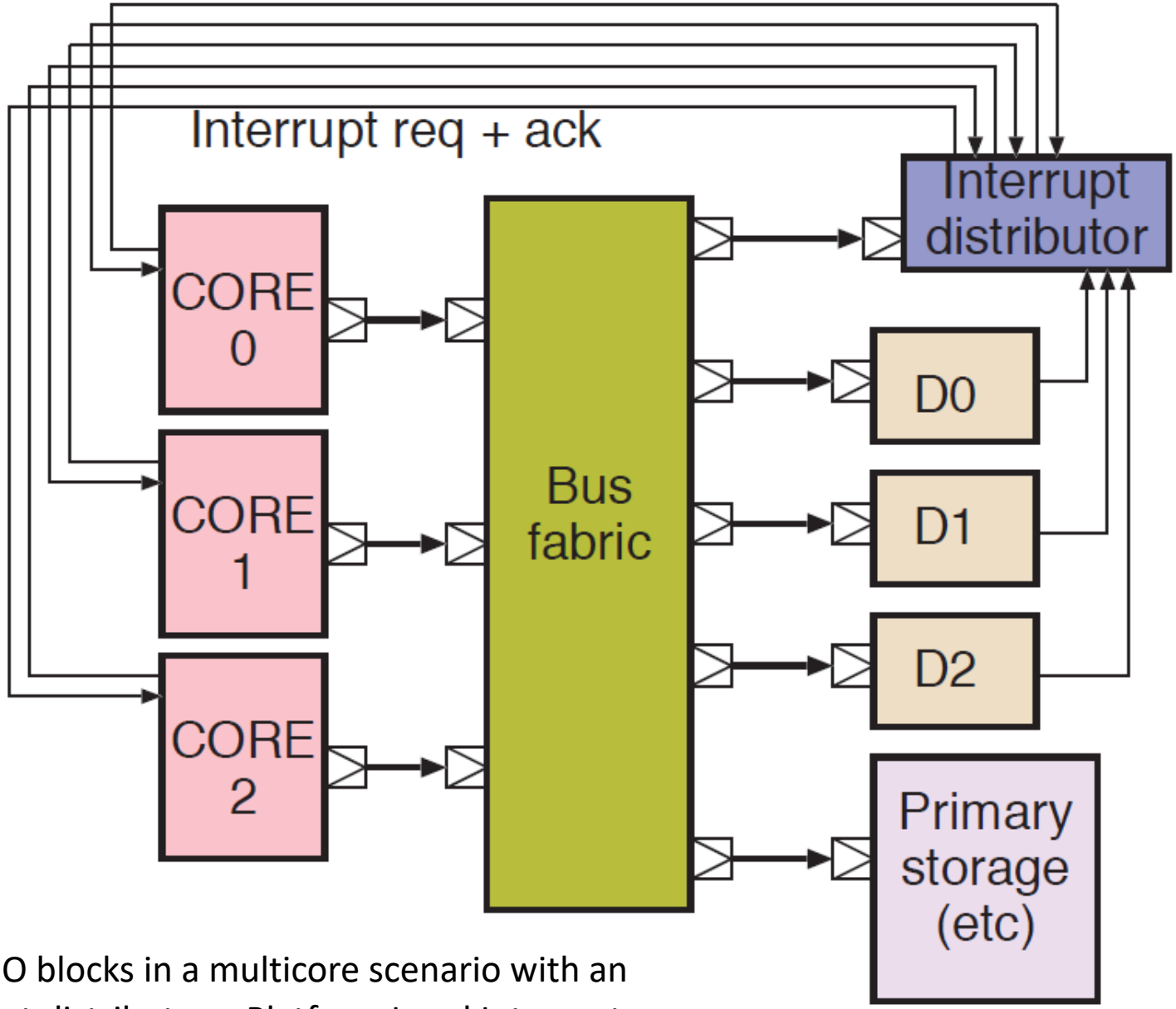


CPU interconnessa con un gruppo di interconnect, quindi verosimilmente una crossbar, che ha una singola porta di iniziazione di comunciazione e multipli ricevitori .

Serve per fare multiplexing tra diversi segnali di intrrrupt emessi dalle periferiche e una singola linea di interrupt che può essere ricevuta dalla cpu. L'interrupt controller è generalmente programmabile, permette di definire sia delle priorità sulle linee di interrupt. Sia a fare clean della linea di interrupt sia comunicare un indice relativo alla periferica che ha generato l'interrupt e quindi gestire le interrupt vetttorizzate, che permettono all' interrupt service routin di sapere già all'intero di un registro qual è il dispositivo che ha generato l'interrupt senza dover chiedere qual è all'interrupt control.
C'è un interrupt controller per ciascun core.

Three IO blocks connected to
a CPU, MEM, Core level
Interupt Controller (PIC)

Interrupt Controllers



Three IO blocks in a multicore scenario with an interrupt distributor – Platform Level Interrupt Controller (PLIC)

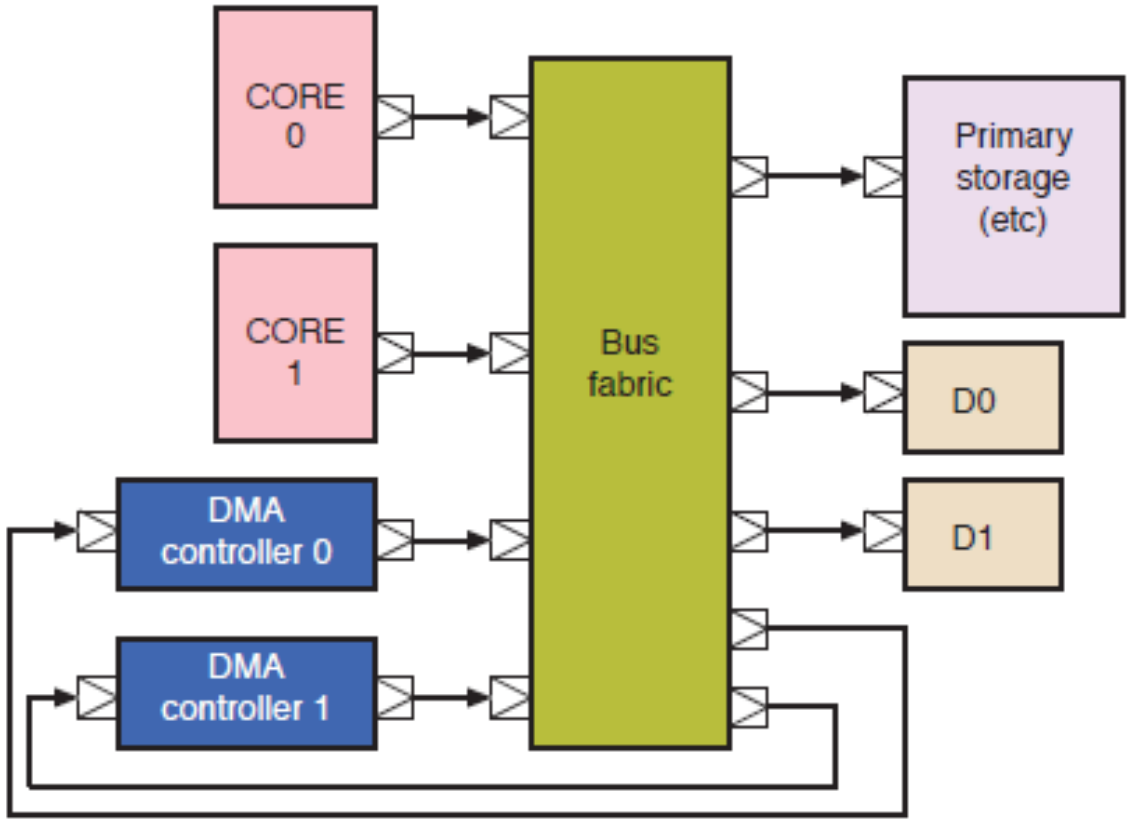
Gestione periferiche

È un dispositivo che può leggere e scrivere da e verso memoria, può quindi essere iniziatore di comunicazioni, ma può anche ricevere comunicazioni ed essere programmato. Gestisce anche gli interrupts.

:

Direct Memory Access:

- Introduciamo un controllore HW per gestire direttamente i trasferimenti dati tra periferica e memoria centrale. Senza coinvolgimento della CPU.
- Direct Memory Access Controller (DMAC)



Direct Memory Access Controller (DMAC):

L'architettura di un DMAC prevede come minimo:

I. Un contatore del numero di parole da trasferire;

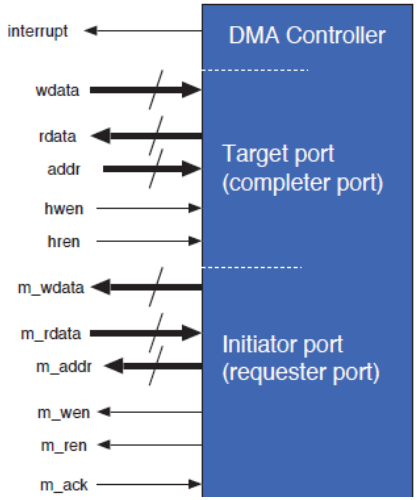
II. Un puntatore alla posizione della memoria in cui verrà letto/scritto il prossimo dato;

III. Un registro di comando che indichi il tipo di trasferimento;

IV. Un eventuale registro di stato.

V. Un registro per mascherare eventi (Mask Register)

Questo fa riferimento ad un ad un DMAC con un singolo canale ciascuno dei quali presenta un suo registro contatore ed un suo registro di puntatore.



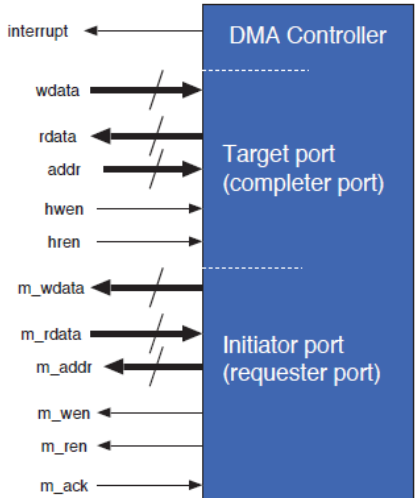
Ha una porta che riceve comunicazioni, per configurare il registro interno. La porta poi che permette di comunicare serve per fare trasferimenti da e verso la memoria.
I canali del DMA consentono di generare flussi di indirizzi disgiunti, il dma più semplice avrà registro che permette di definire un indirizzo base, un registro che consente di definire un contatore, un registro di comando che definisce che tipo di trasferimento il dma controller può supportare, un registro di stato che dice quanti trasferimenti sono stati fatti, a ciascuno di questi canali sarà associato l'interrupt di una periferica e la dove l'interrupt viene generato, il dma effettua un trasferimento da un indirizzo verso un altro.

Direct Memory Access Controller (DMAC):

Il funzionamento di un DMAC prevede due fasi mutualmente esclusive (una esclude l'altra):

1) Fase di programmazione: Serve a trasmettere al DMAC le informazioni che definiscono la modalità di trasferimento sul canale.
   - Il controllore viene visto come una periferica dotato di un insieme di registri.
   - Si programmano i registri puntatore, contatore e di comando. Si controlla lo stato.

2) Fase di trasferimento dei dati: Si distinguono due modalità:
   - Trasferimento singolo (cycle stealing): il DMAC occupa il bus per un ciclo (pochi cicli) necessari al trasferimento di un dato da/verso la memoria.
   - Trasferimento a blocchi (burst): prevede l'occupazione del bus per tutto il tempo richiesto a trasferire il numero di parole scritto nel contatore in fase di programmazione.



Virtual Memory e periferiche

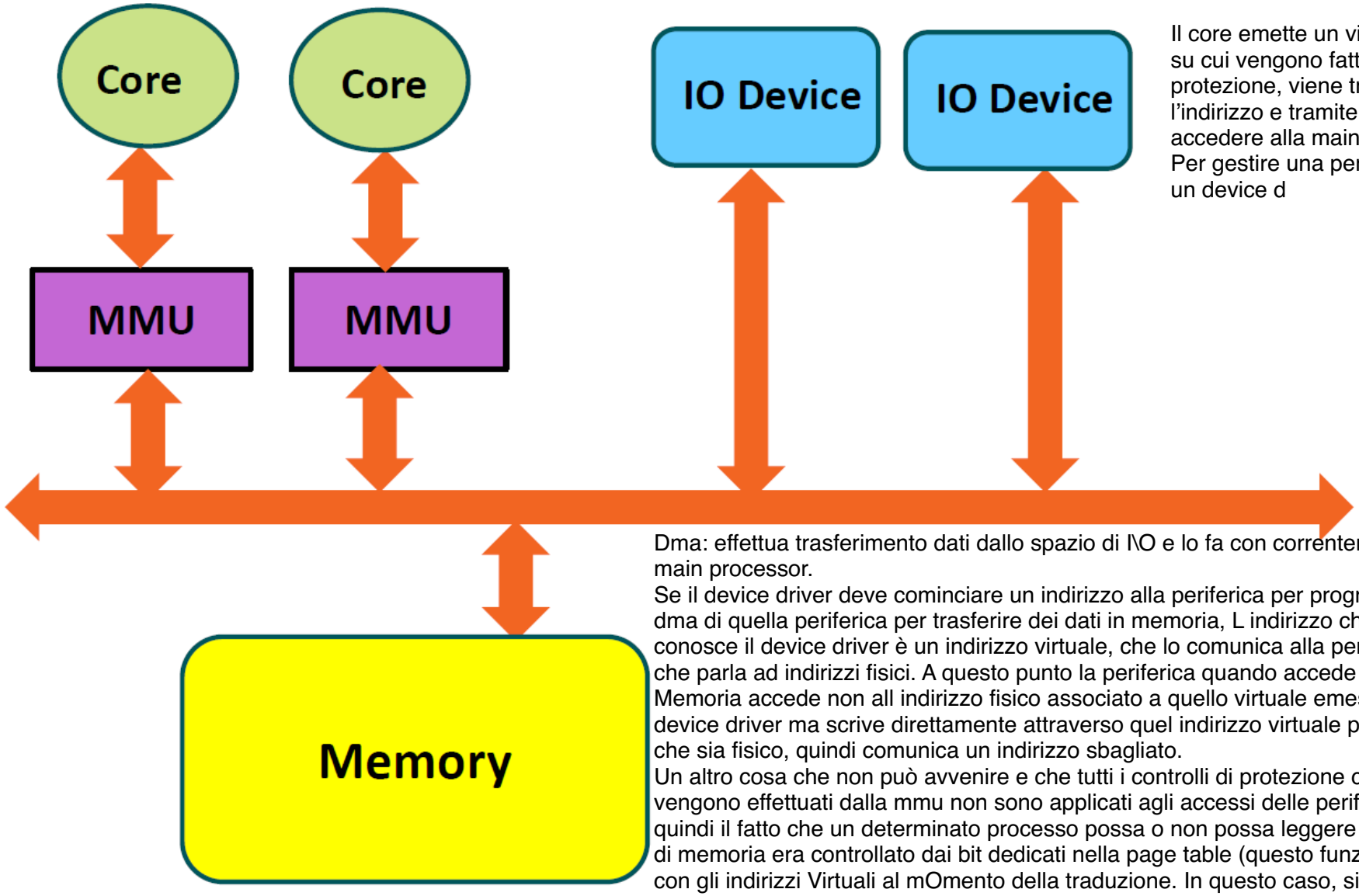
Andrea Bartolini – a.bartolini@unibo.it

MOTIVATION: TRADITIONAL DMA BY IO

AMD

NO SYSTEM VIRTUALIZATION

I core emettono indirizzi virtuali quando vogliono communire con la memoria. La memoria comunica con i bus con indirizzi fisici. Abbiamo poi periferiche che sono connesse a bus con indirizzi fisici. Quando i core eseguono load e store, la traduzione degli indirizzi viene emessa dalla mmu.

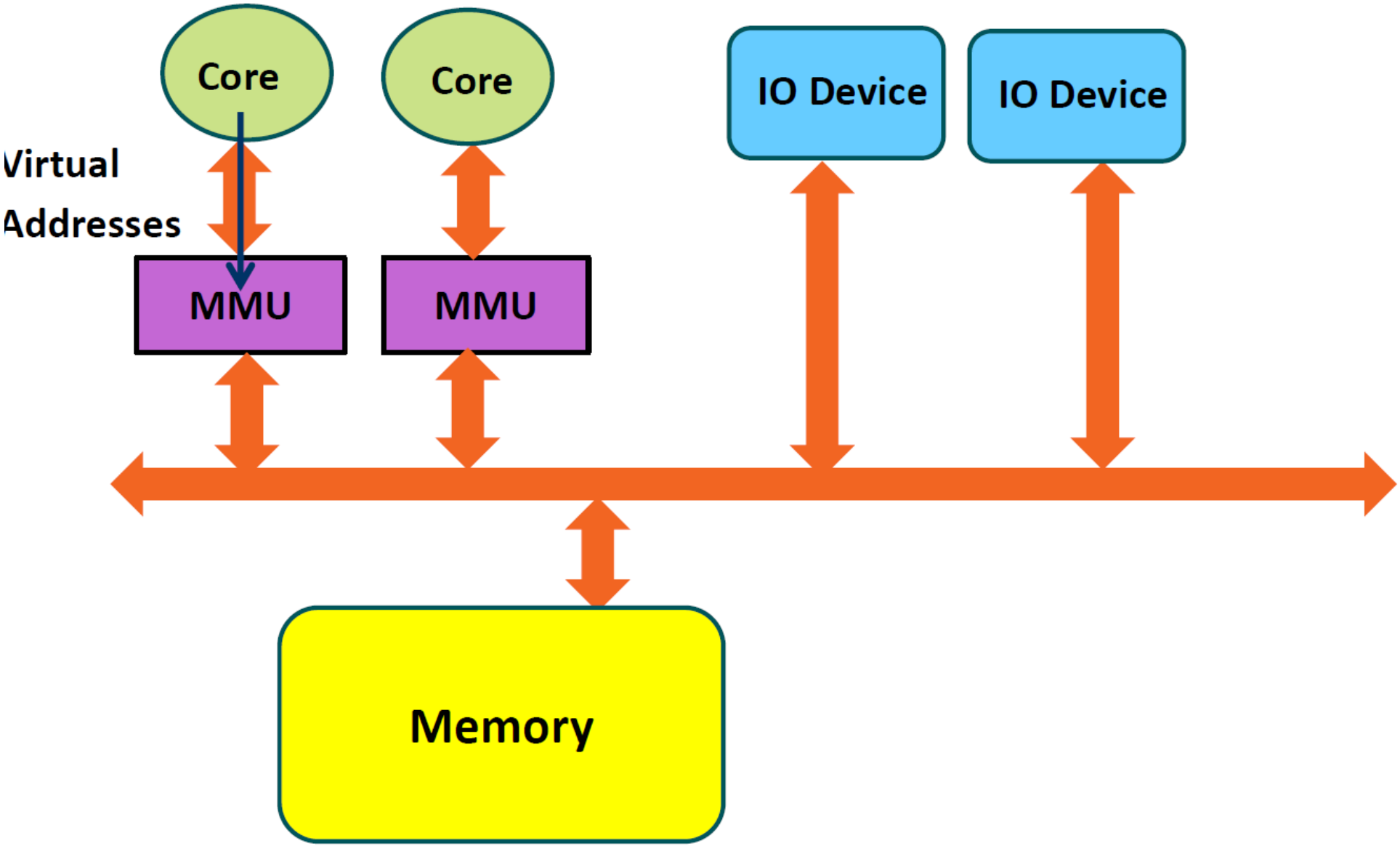


Il core emette un virtual address, su cui vengono fatti controlli di protezione, viene tradotto l'indirizzo e tramite questo si può accedere alla main memory.
Per gestire una periferica avremo un device d

Dma: effettua trasferimento dati dallo spazio di I\O e lo fa con correntemente al main processor.
Se il device driver deve cominciare un indirizzo alla periferica per programmare il dma di quella periferica per trasferire dei dati in memoria, L indirizzo che conosce il device driver è un indirizzo virtuale, che lo comunica alla periferica che parla ad indirizzi fisici. A questo punto la periferica quando accede i. Memoria accede non all indirizzo fisico associato a quello virtuale emesso dal device driver ma scrive direttamente attraverso quel indirizzo virtuale pensando che sia fisico, quindi comunica un indirizzo sbagliato.
Un altro cosa che non può avvenire e che tutti i controlli di protezione che vengono effettuati dalla mmu non sono applicati agli accessi delle periferiche, quindi il fatto che un determinato processo possa o non possa leggere una zona di memoria era controllato dai bit dedicati nella page table (questo funzionava con gli indirizzi Virtuali al mOmento della traduzione. In questo caso, siccome il dispositivo di i/o parla a indirizzi fisici non è possibile garantire che quel dispositivo di i/o non vada ad accedere in zona e in cui non può. Quindi va esteso l'hw per poter prevenire queste condizioni.

MOTIVATION: TRADITIONAL DMA BY IO

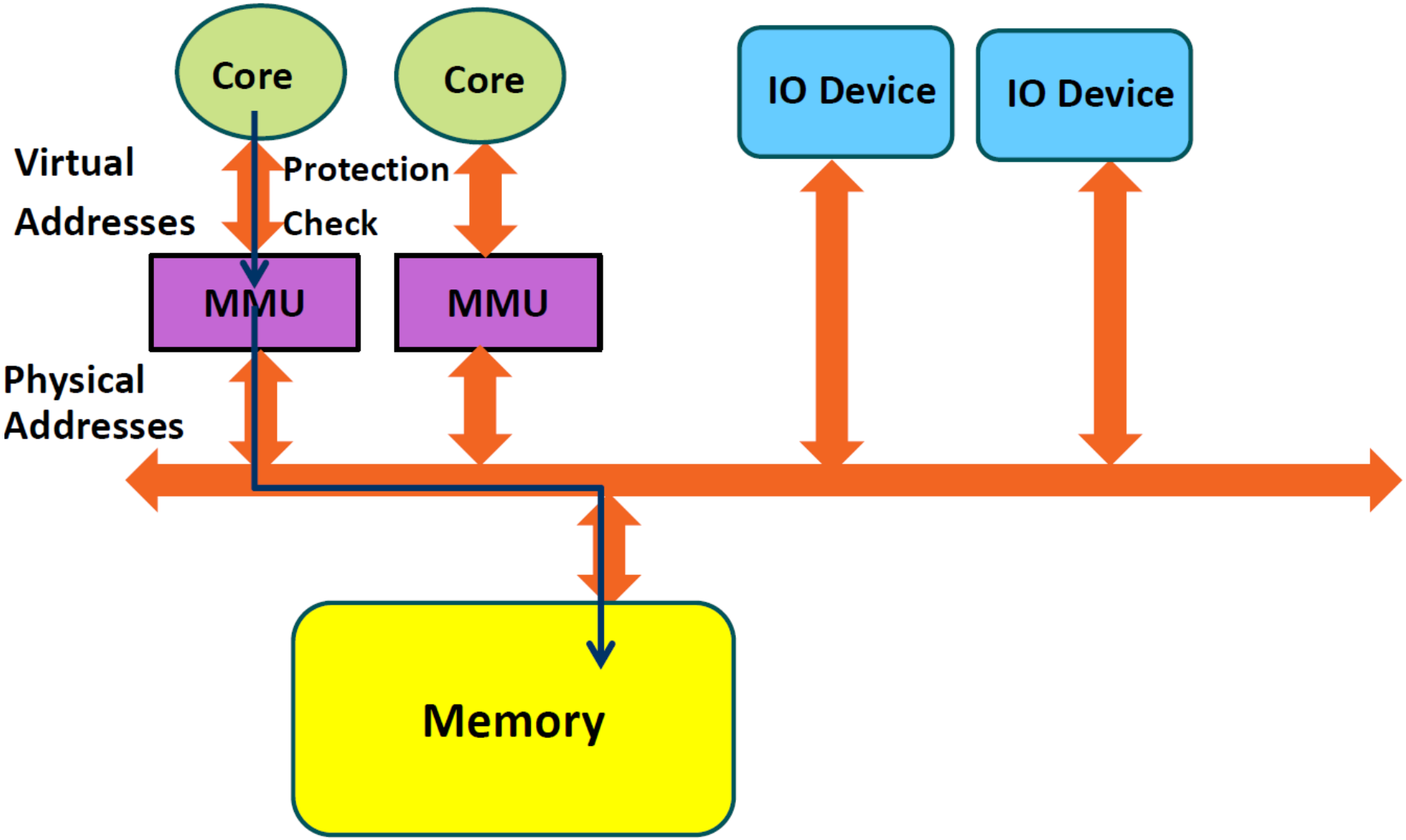
NO SYSTEM VIRTUALIZATION



MOTIVATION: TRADITIONAL DMA BY IO

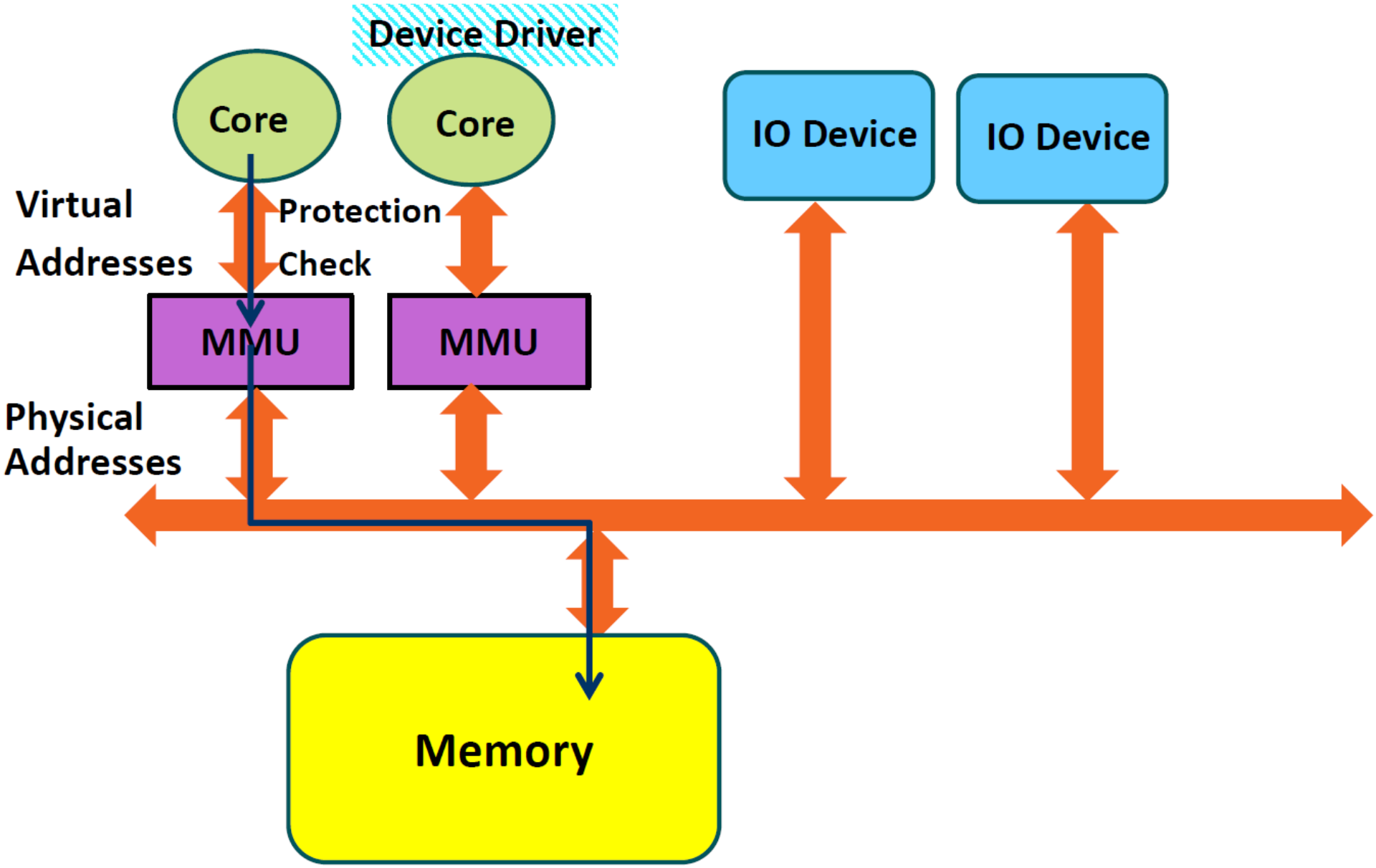
NO SYSTEM VIRTUALIZATION

AMD



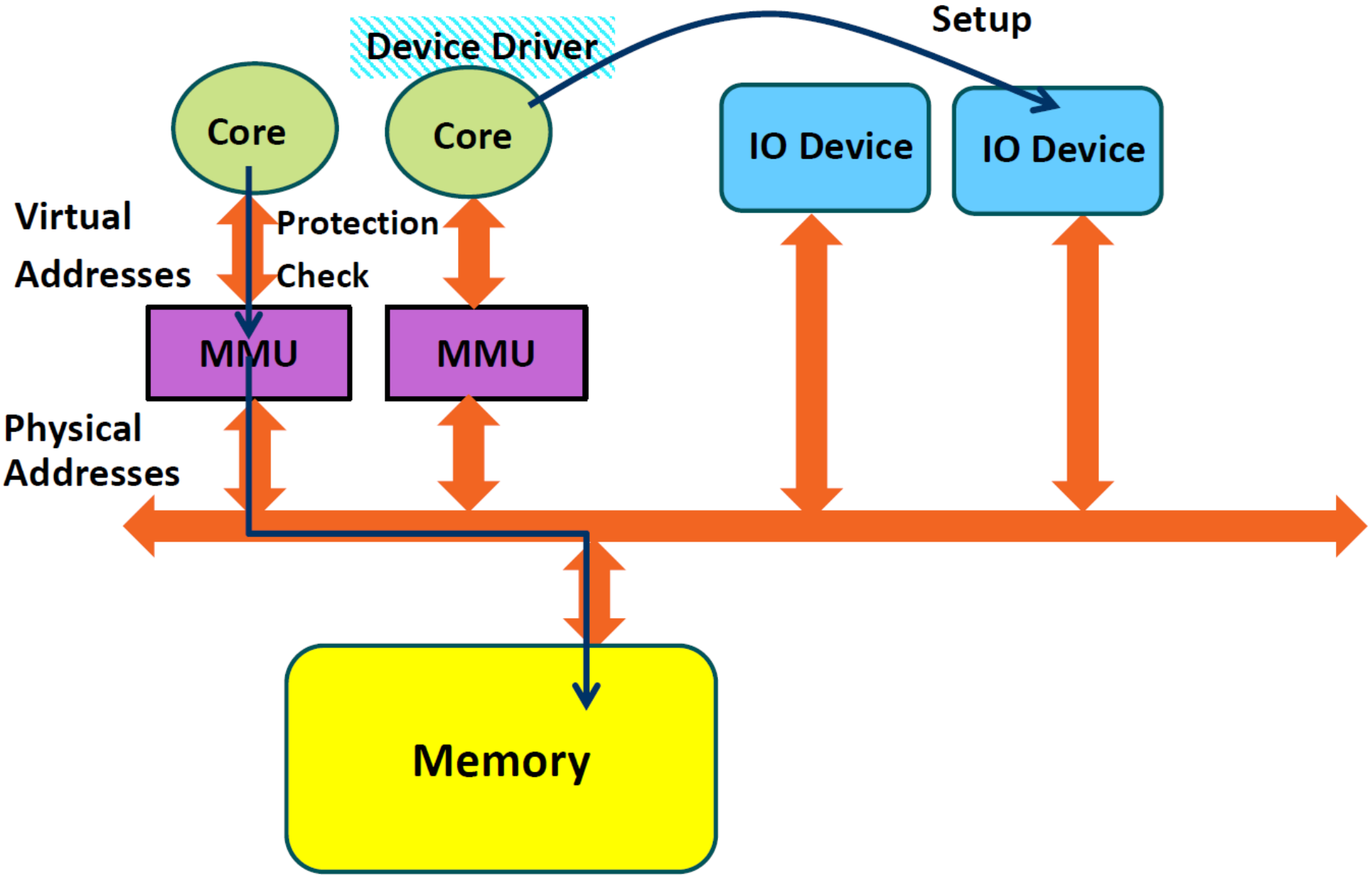
MOTIVATION: TRADITIONAL DMA BY IO

NO SYSTEM VIRTUALIZATION



MOTIVATION: TRADITIONAL DMA BY IO

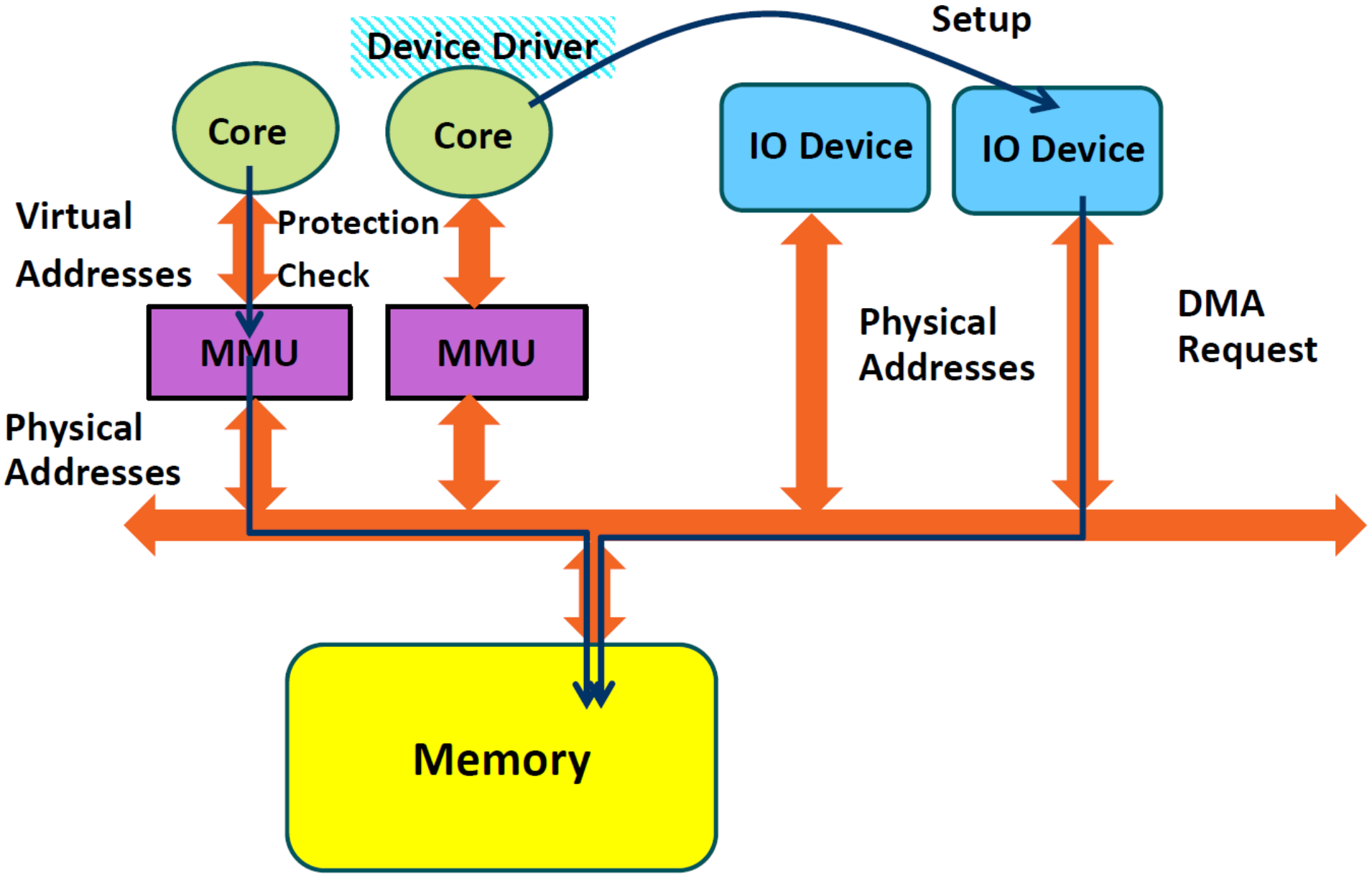
NO SYSTEM VIRTUALIZATION



MOTIVATION: TRADITIONAL DMA BY IO

NO SYSTEM VIRTUALIZATION

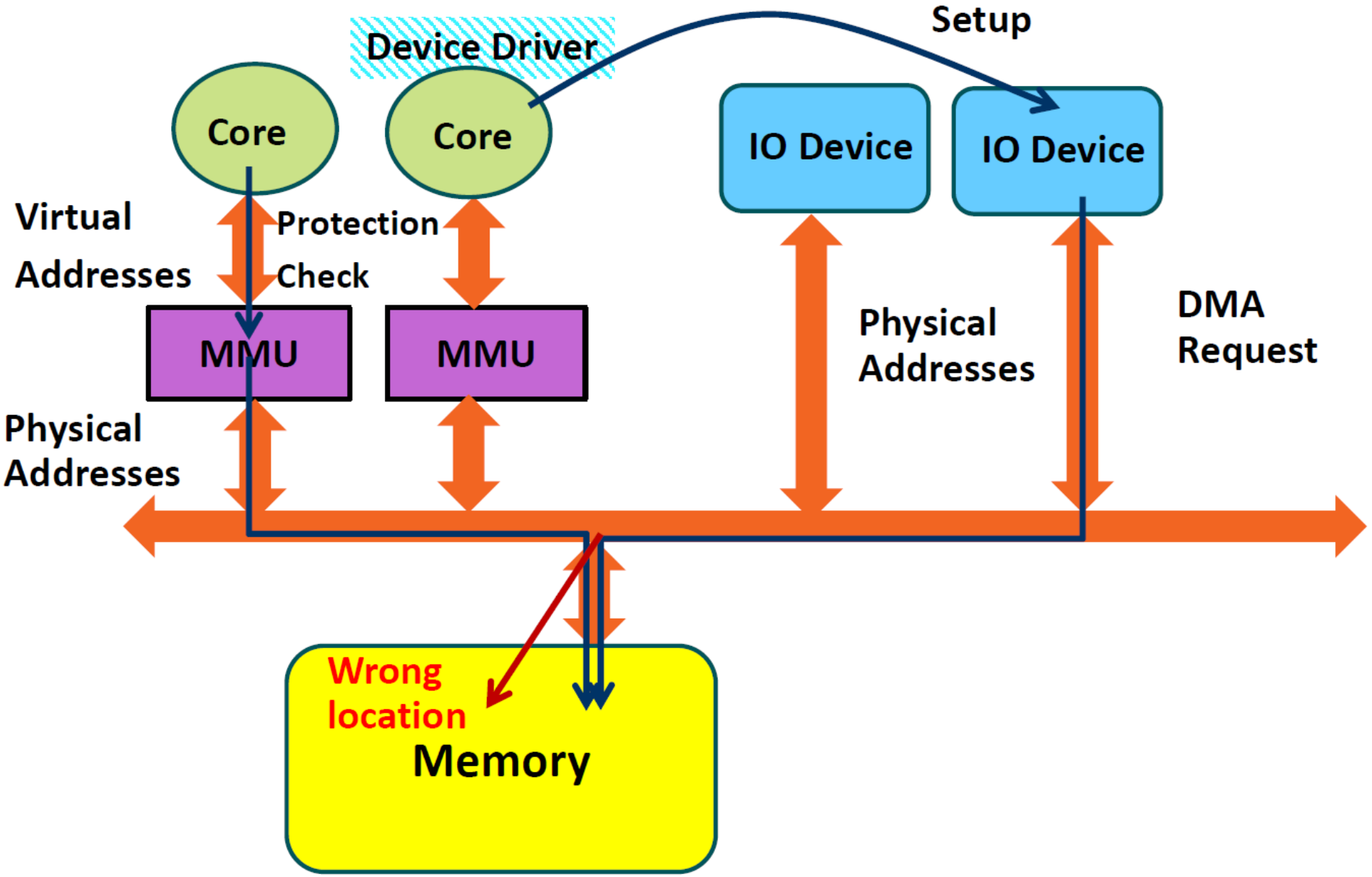
AMD



MOTIVATION: TRADITIONAL DMA BY IO

NO SYSTEM VIRTUALIZATION

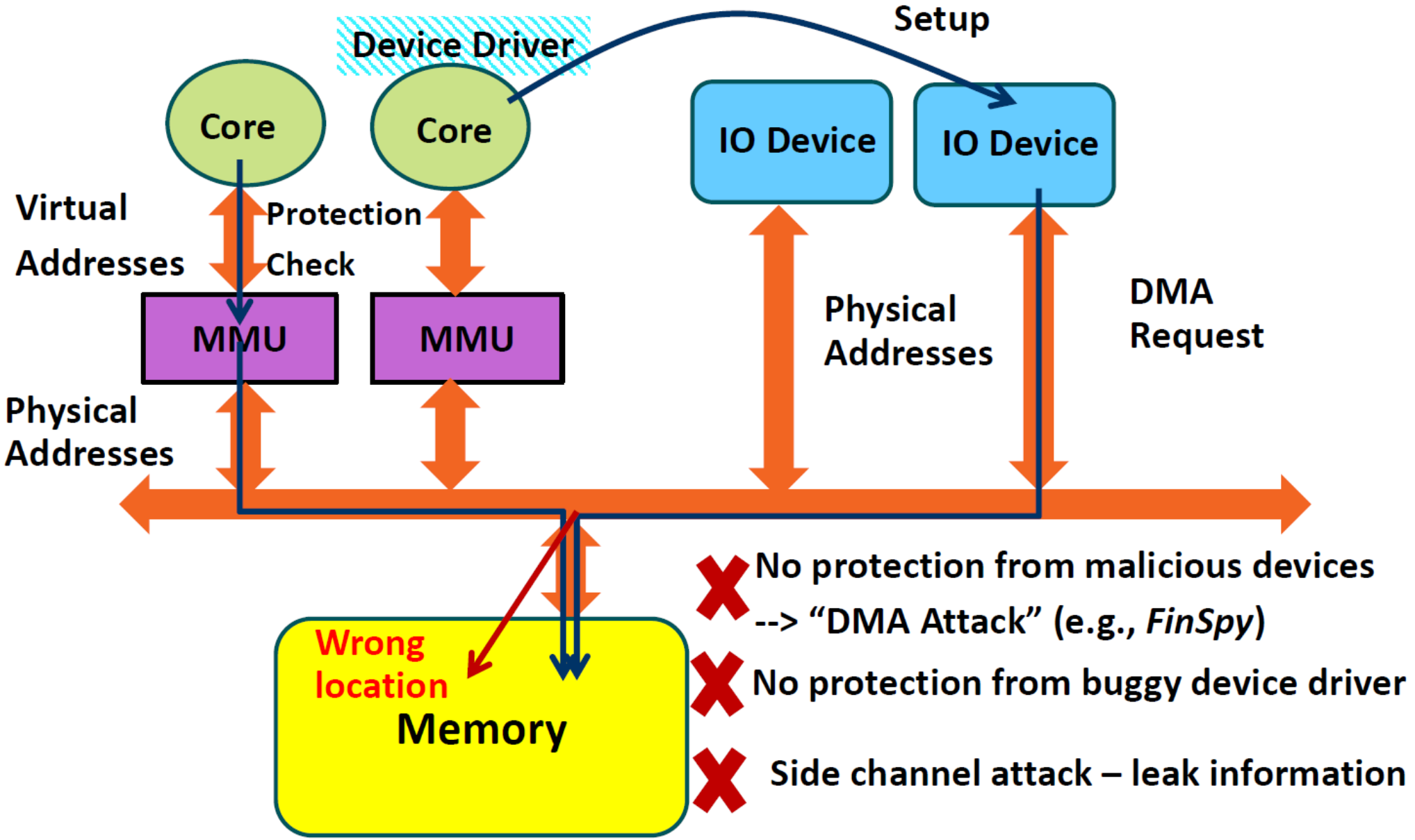
AMD



MOTIVATION: TRADITIONAL DMA BY IO

NO SYSTEM VIRTUALIZATION

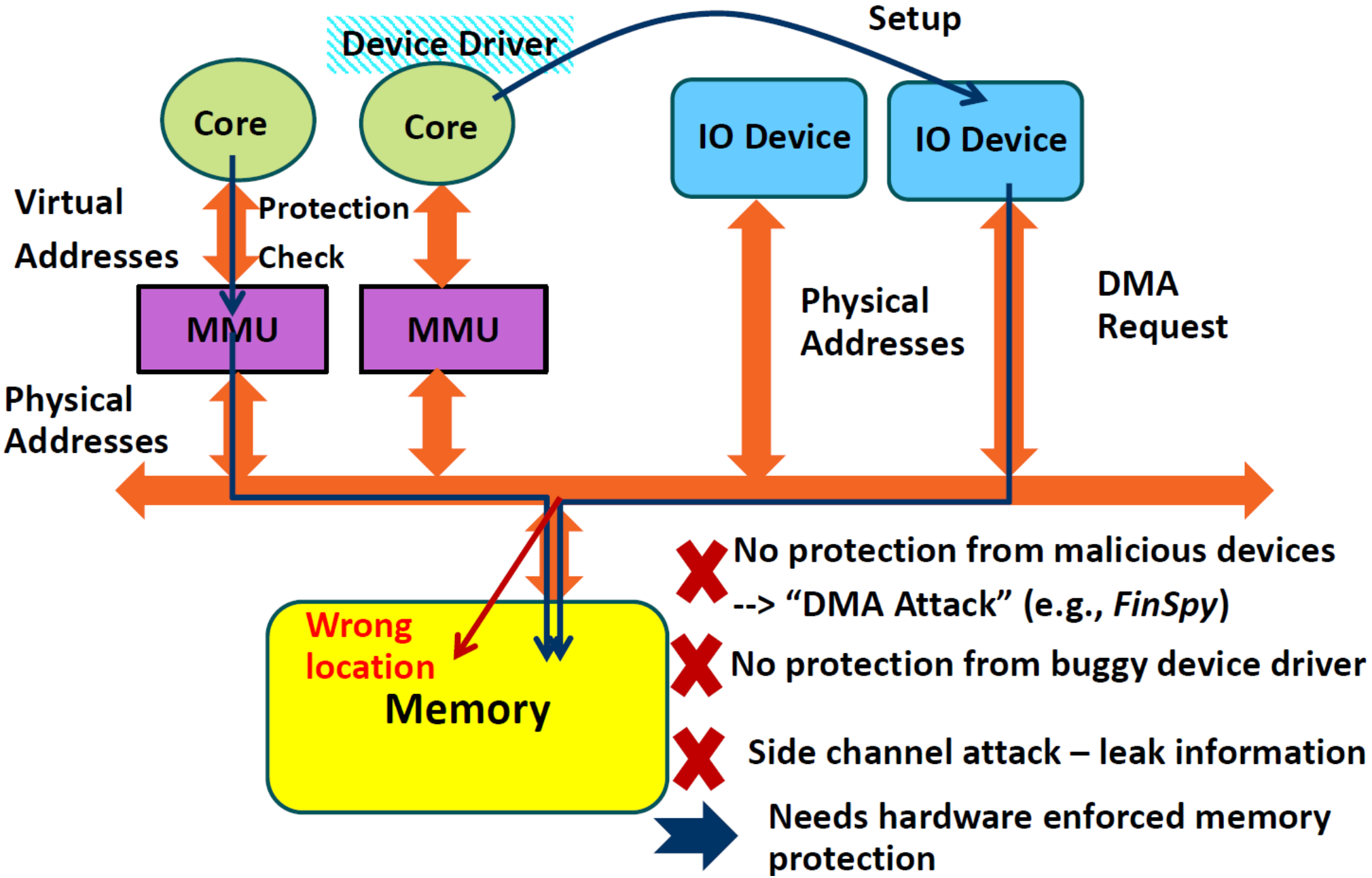
AMD



MOTIVATION: TRADITIONAL DMA BY IO

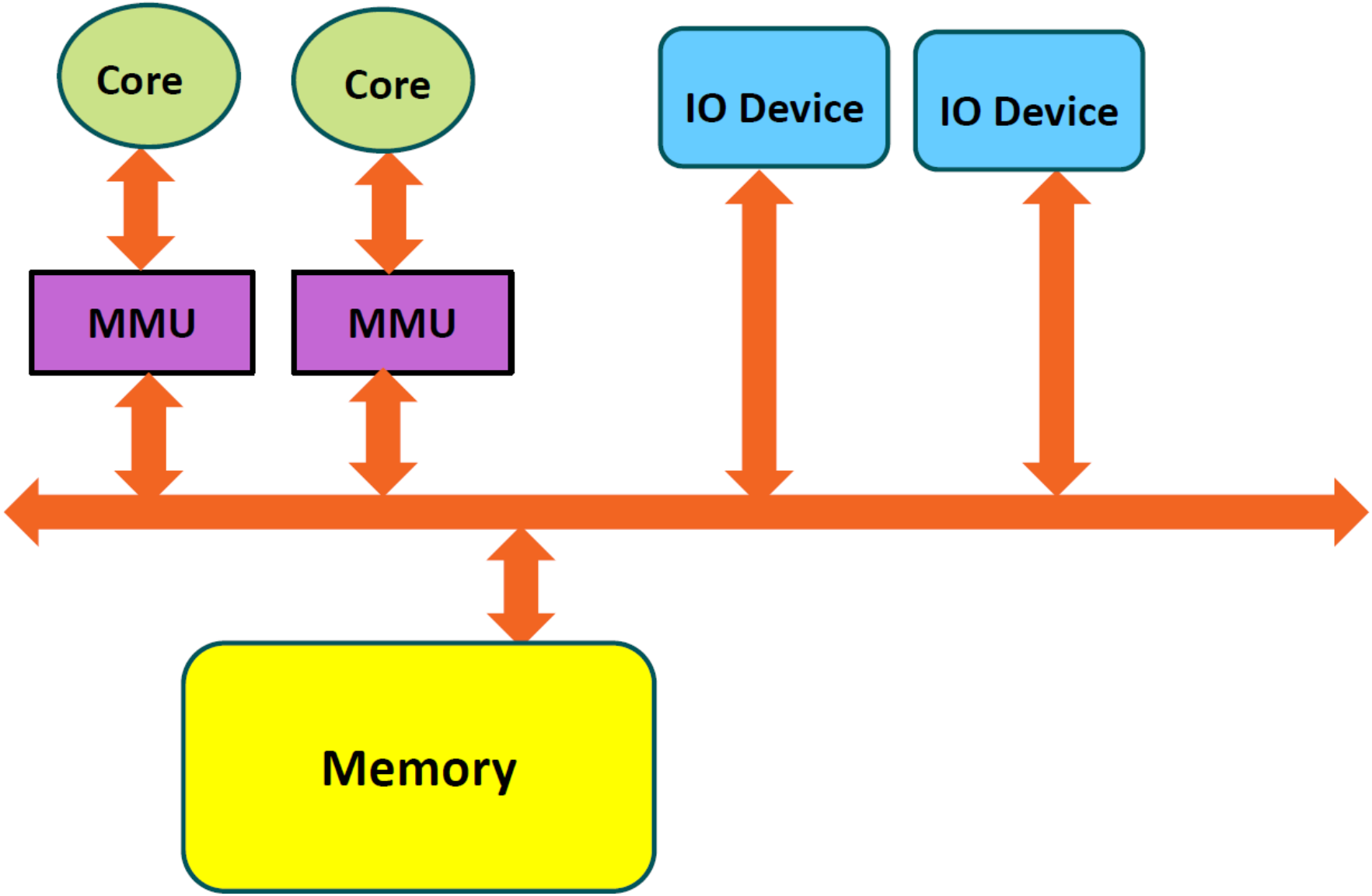
NO SYSTEM VIRTUALIZATION

AMD



INTRODUCTION OF IOMMU: THE LOGICAL VIEW

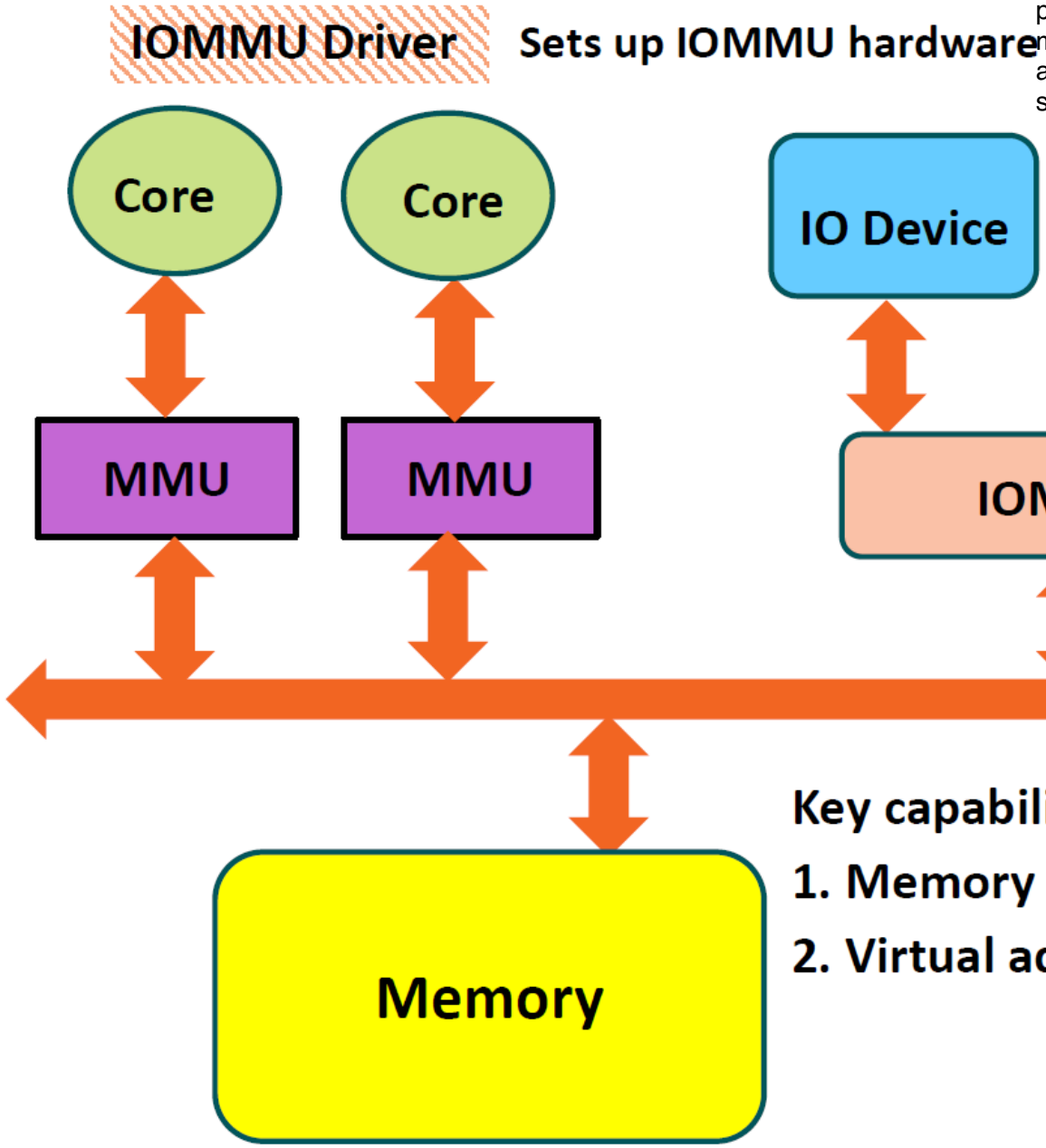
AMD



INTRODUCTION OF IOMMU: THE LOGICAL VIEW

AMD

Quello che negli anni è emersa o e permettere ai dispositivo di I/ˆo di avere una loro MMU, che può essere configurato e serve ai dispositivi di i o di tradurre gli indirizzi da virtuali a fisici e garantire protezione negli accesi che i dispositivi di i/o fanno verso la memoria.Quindi eve gestire la protezione della memoria verso gli accessi emessi da un DMA e permettere che gli accessi dal DMA siano tradotti in indirizzi fisici.



Questo consente di creare la unified memory, una memoria può essere condivisa dal core, cpu, gpu… altrimenti non potrebbero de non ci fossero garanzie di memory protection.

Hardware that intercepts DMA transactions

Il page fault viene comunque gestito dalla cpu, la gestione della traduzione può essere fatta direttamente dal dispositivo i/o.
Quindi si hanno due versioni di mmu, una contiene solo un tlb,e la dov'è non c'è la traduzione di genera un interrupt verso il main processor che Andra a aggiornare quel sotto set di pagine di memoria virtuale conosciuta dai dispositivi di i\o. Nel caso più sofisticato la mmu ah anche page walker e può cercare da solo le traduzioni nella page table.

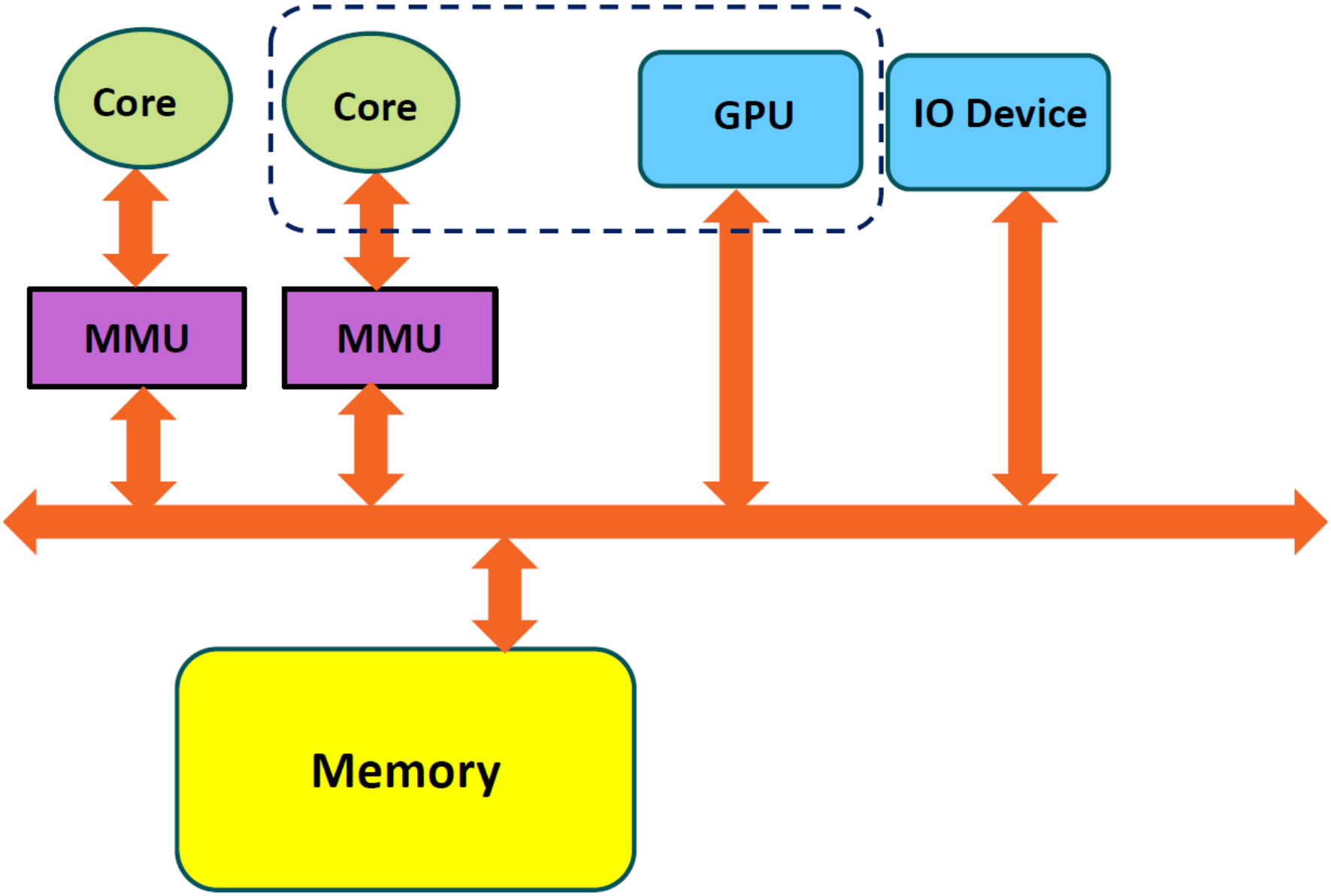
Key capabilities:

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

MOTIVATION: EMERGENCE OF HETEROGENEOUS SYSTEMS

HETEROGENEOUS SYSTEM ARCHITECTURE (HSA)

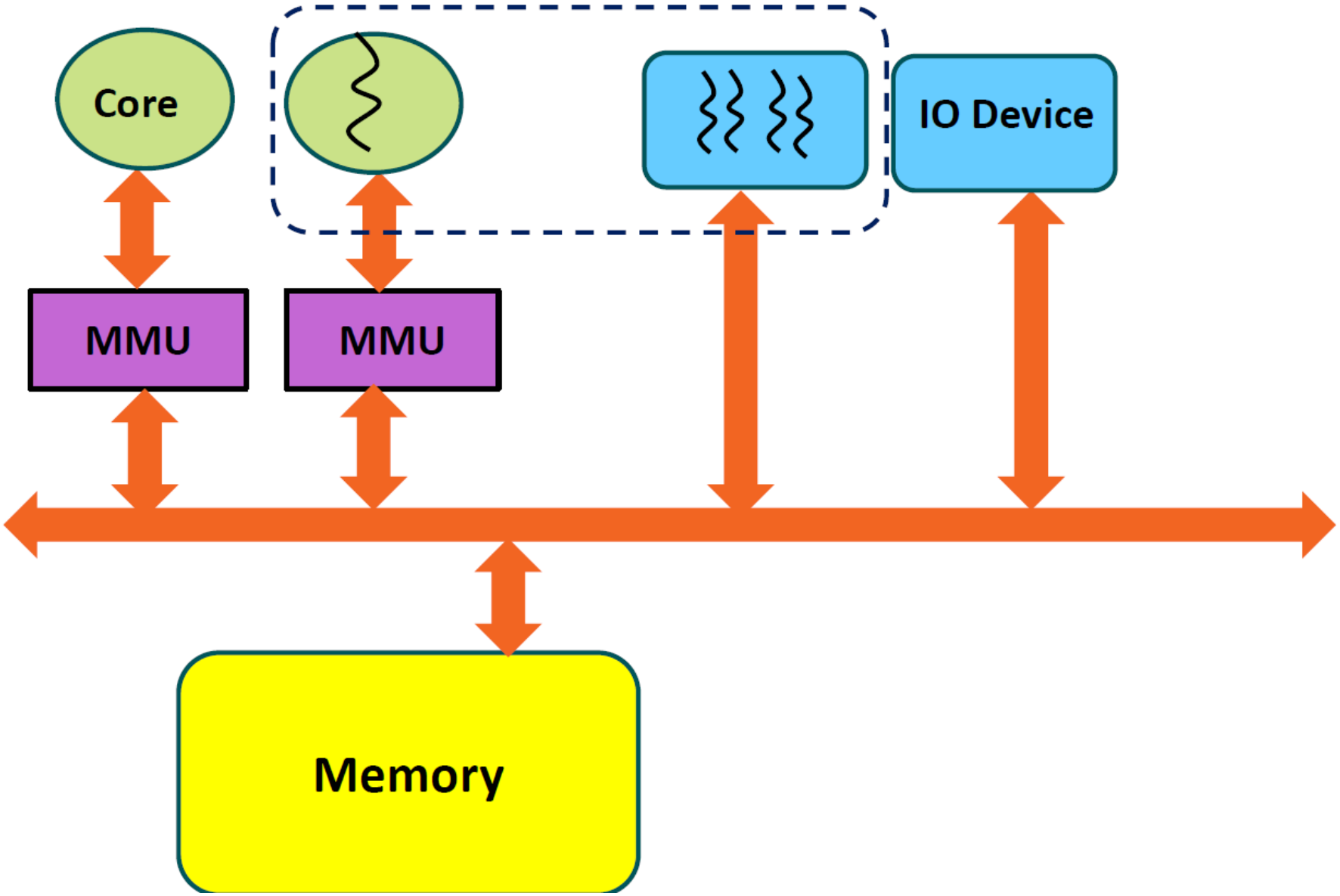
Shared virtual addressing is key to ease of programming



MOTIVATION: EMERGENCE OF HETEROGENEOUS SYSTEMS

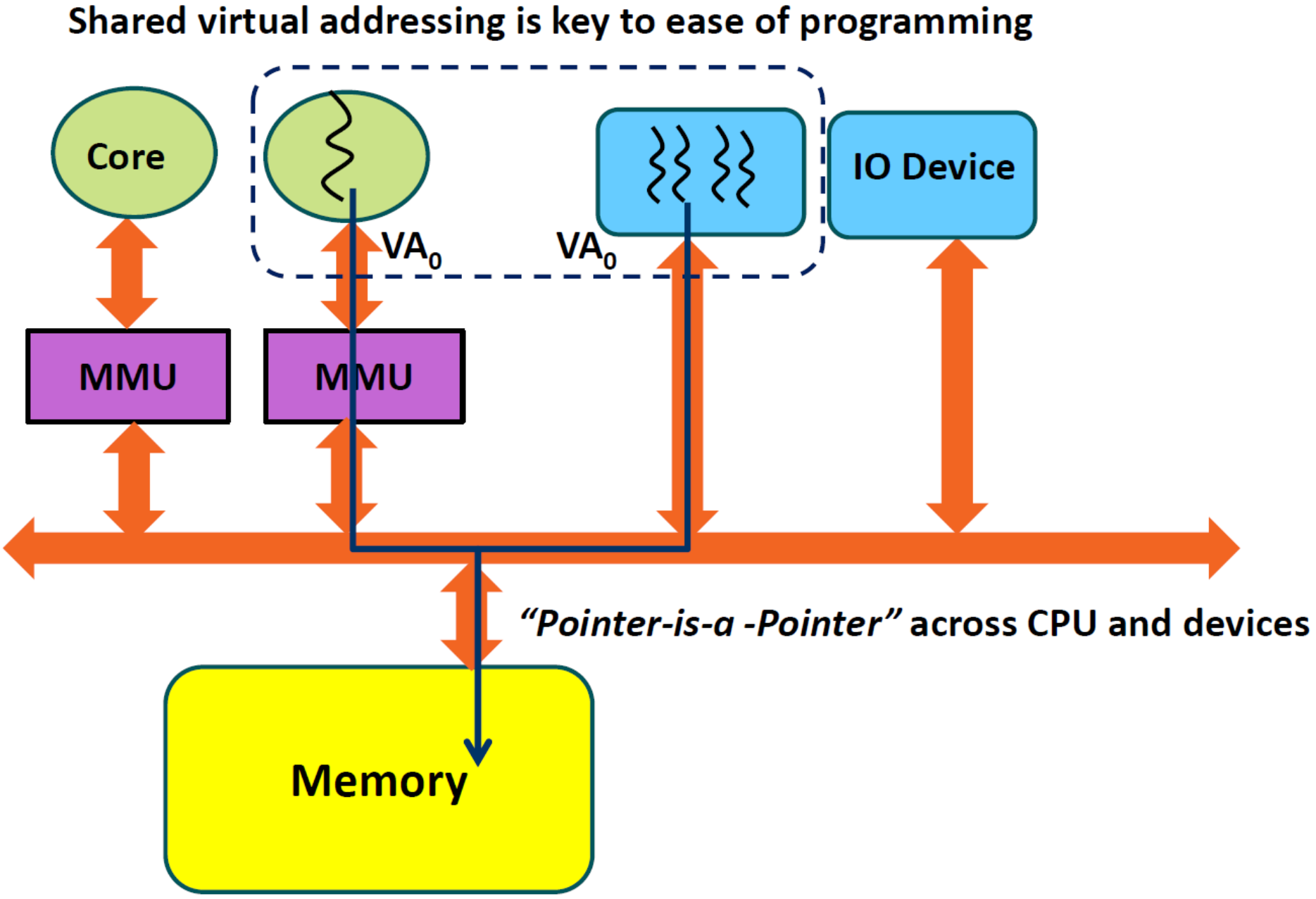
HETEROGENEOUS SYSTEM ARCHITECTURE (HSA)

Shared virtual addressing is key to ease of programming



MOTIVATION: EMERGENCE OF HETEROGENEOUS SYSTEMS

HETEROGENEOUS SYSTEM ARCHITECTURE (HSA)

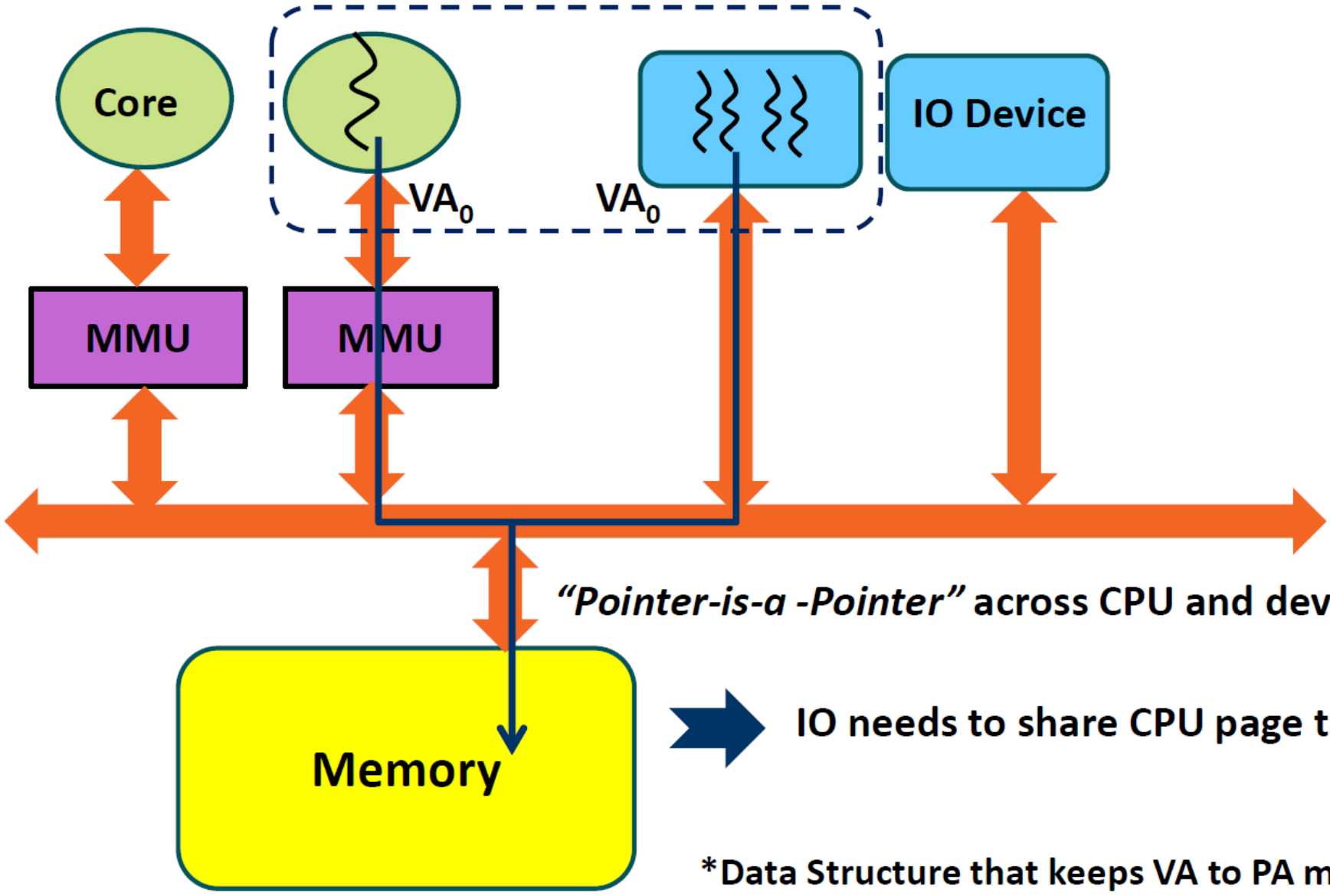


MOTIVATION: EMERGENCE OF HETEROGENEOUS SYSTEMS

AMD

HETEROGENEOUS SYSTEM ARCHITECTURE (HSA)

Shared virtual addressing is key to ease of programming



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

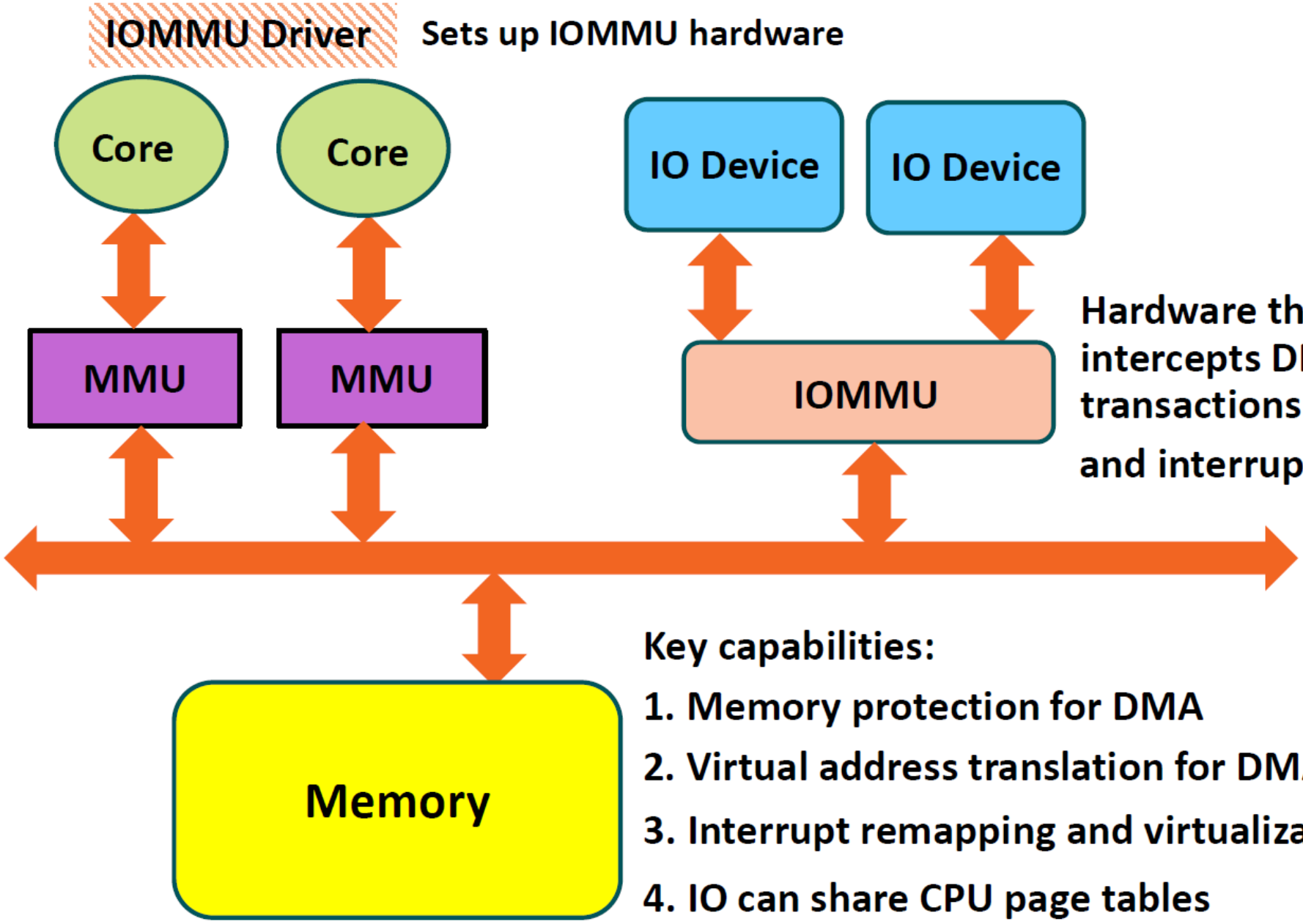
IO needs to share CPU page table*

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

INTRODUCTION OF IOMMU: THE LOGICAL VIEW

ADDING ABILITY TO SHARE ADDRESS SPACE IN HETEROGENEOUS SYSTEM

AMD



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