

How to enable AMD IOMMU in coreboot

OSFC 2018

Piotr Król







- Introduction
- Motivation
- What is IOMMU and why we may need it?
- AMD IOMMU features
- ACPI tables brief explanation
- Implementation state
 - Broken IVRS from AGESA
 - Initial patches
 - Current state and further work
- Summary



Piotr Król

Founder & Embedded Systems Consultant

-  @pietrushnic
-  piotr.krol@3mdeb.com
-  [linkedin.com/in/krolpiotr](https://www.linkedin.com/in/krolpiotr)
-  [facebook.com/piotr.krol.756859](https://www.facebook.com/piotr.krol.756859)
- PC Engines platforms maintainer
- interested in:
 - advanced hardware and firmware features
 - security and updateability

Fascination with security-focused OSes

- QubesOS - personal computing
- OpenXT - critical infrastructure
- ViryaOS - automotive

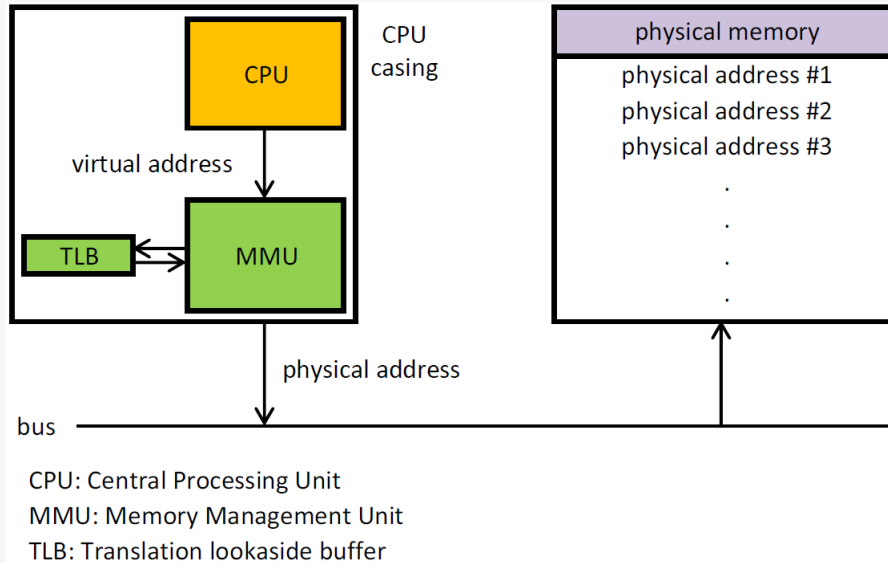
Customer requests

- isolation of user facing web from rest of the system
- separation of NICs for different purposes

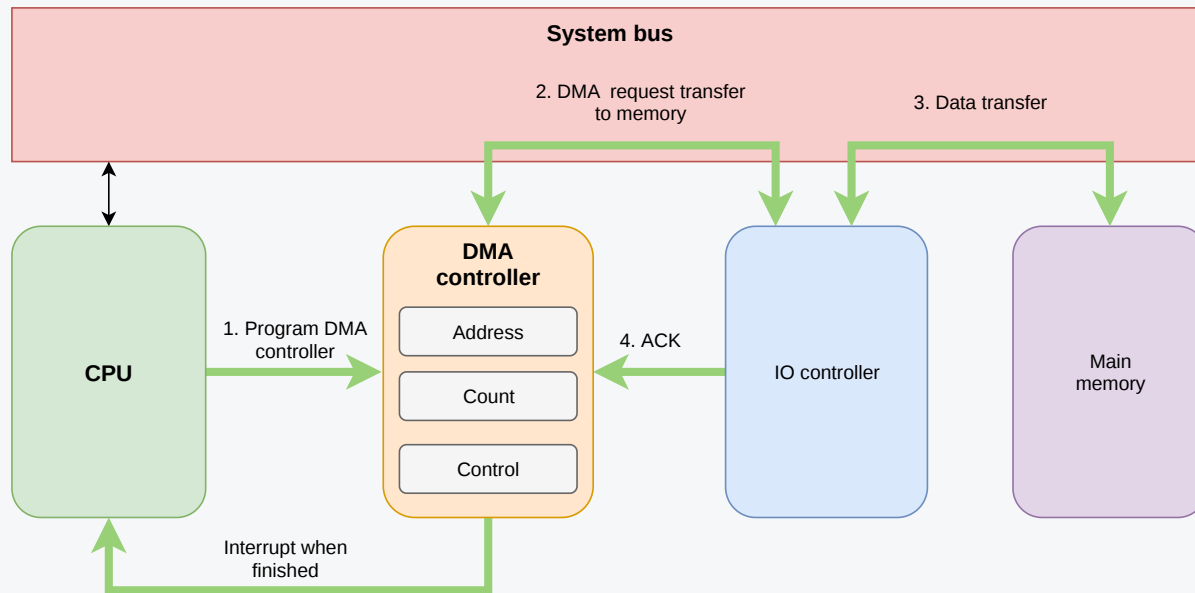
Community work

- virtualized firewalls (pfSense, OPNSense, IPfire)
- Proxmox
- CoreOS
- KVM

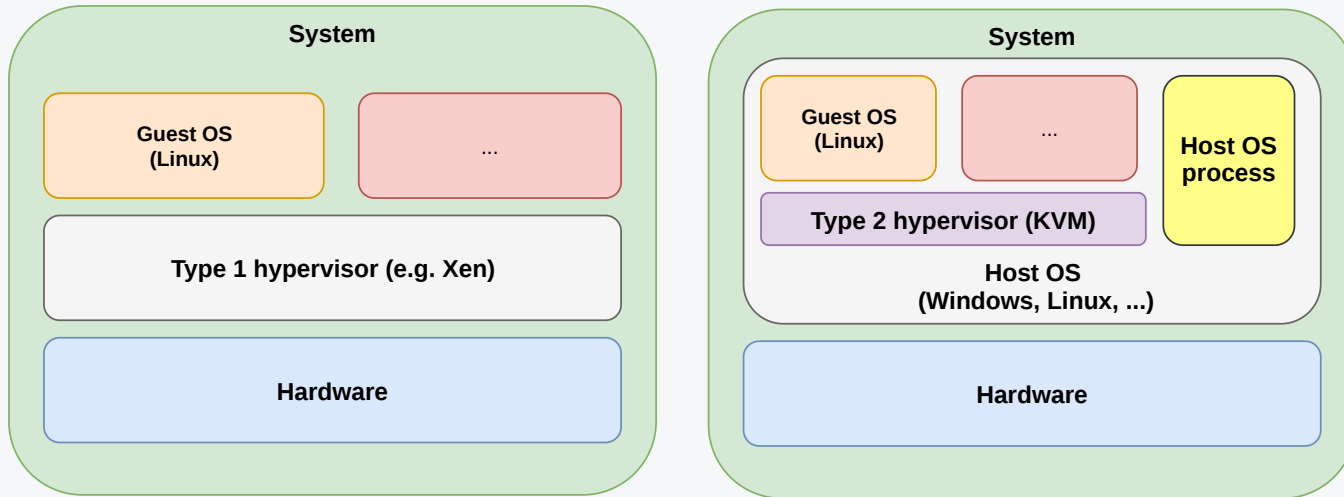
- MMU
- DMA
- Virtualization
- IOMMU



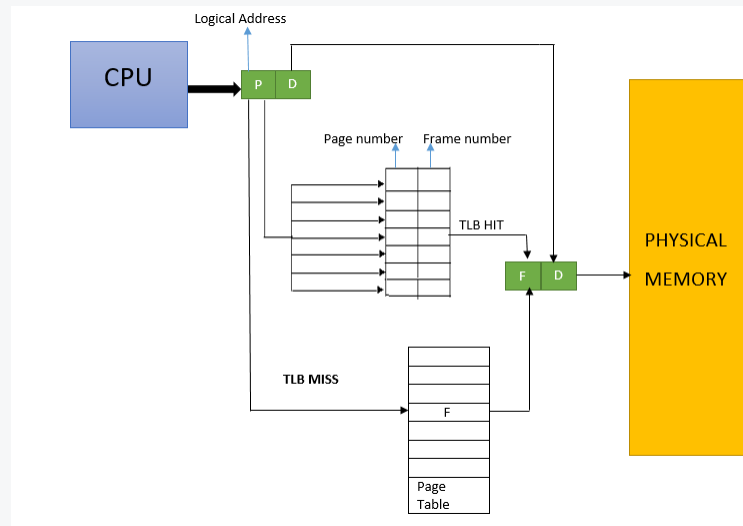
- MMU (Memory Management Unit)
 - virtual memory management
 - memory protection
 - cache control
 - bus arbitration



- DMA (Direct Memory Access)
 - provide performance optimization for IO reads and writes

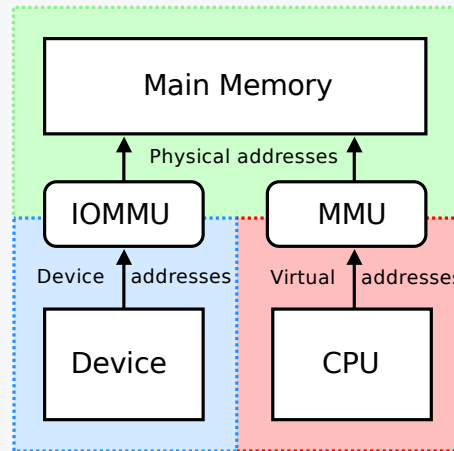


- IOTLB is TLB for IO devices
- TLB is cache of entries that matching virtual address to physical address
- there 2 important things about TLB:
 - when match was found in TLB (**hit**) - we're good
 - when it was not found (**miss**) - things complicate little bit and behavior may be platform depending

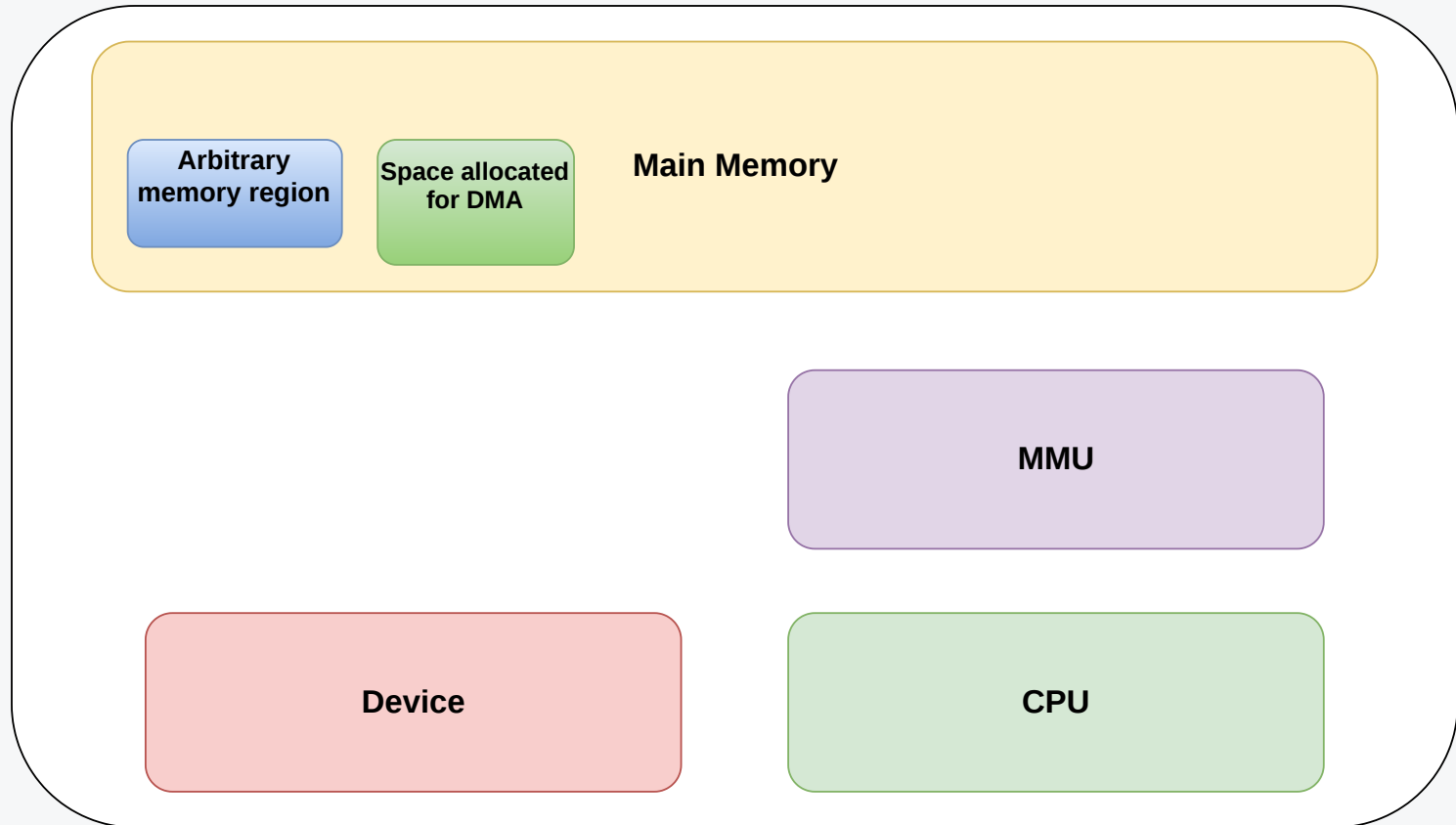


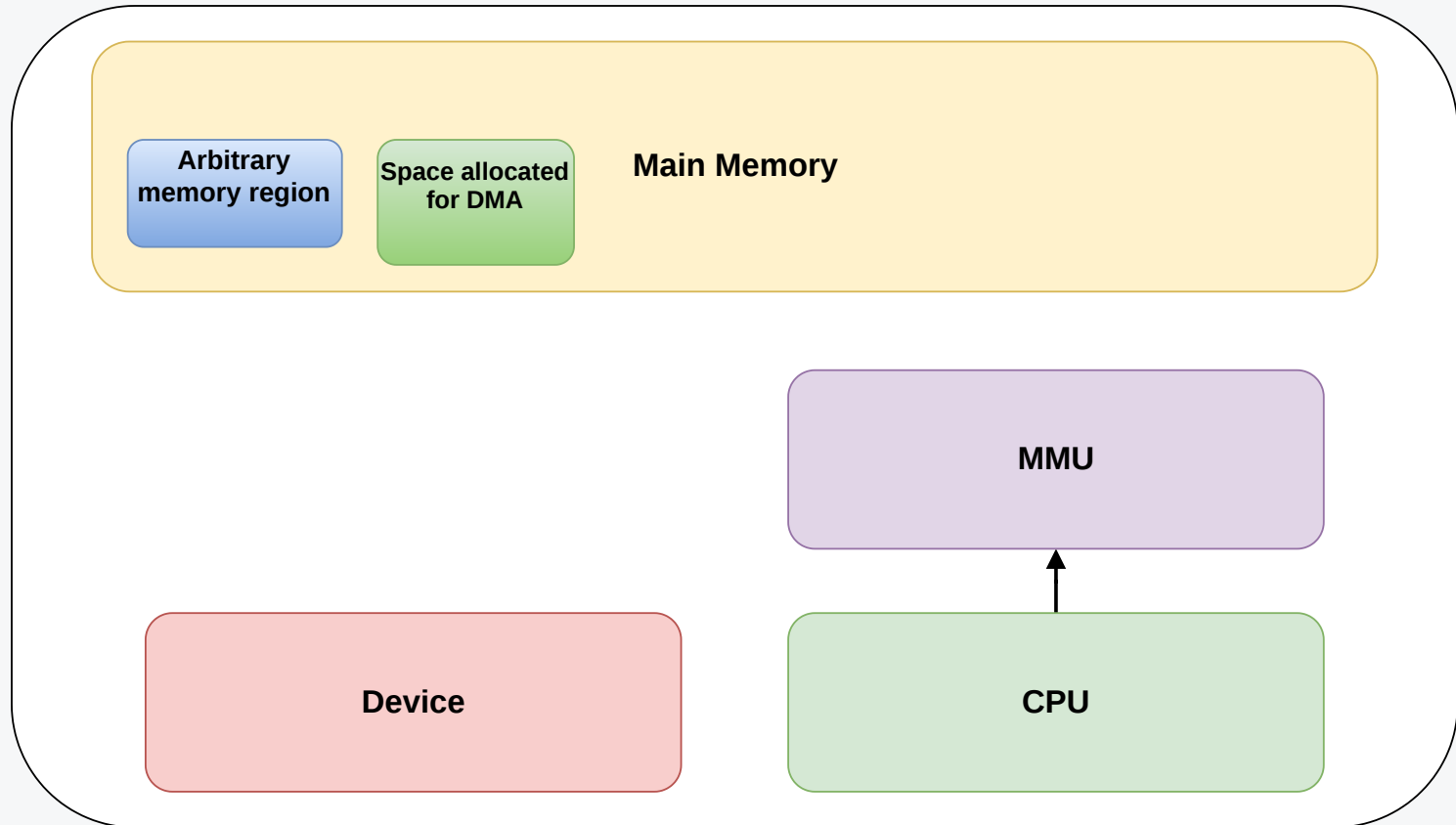
Wikipedia

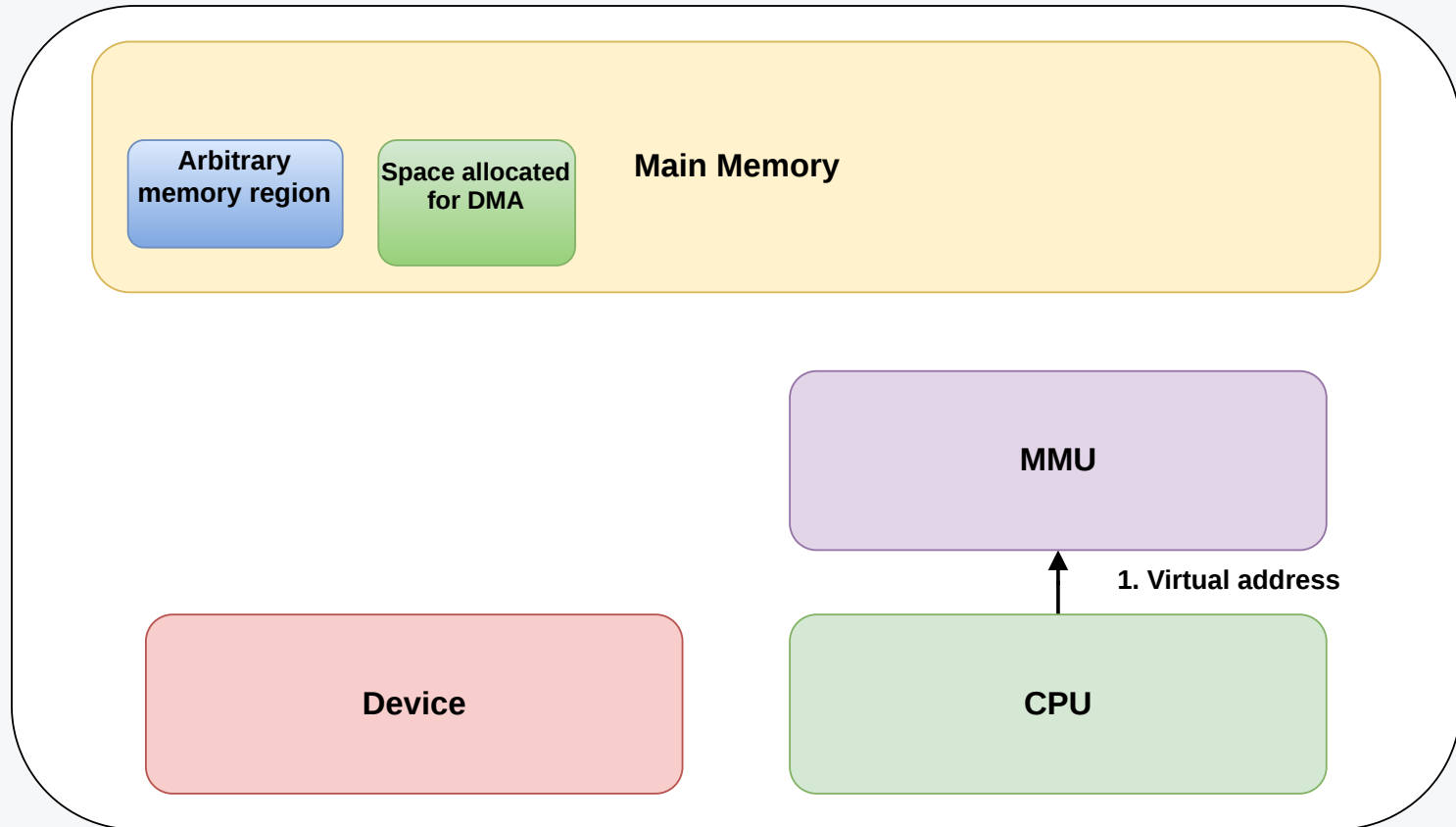
- **I/O Memory Management Unit (IOMMU):** MMU for I/O devices
- most important features:
 - hardware enforced memory protection - security
 - virtual address translation for DMA - performance
 - interrupts remapping and virtualization - performance
 - page table sharing - e.g. graphics performance
 - PCI passthrough - security and performance

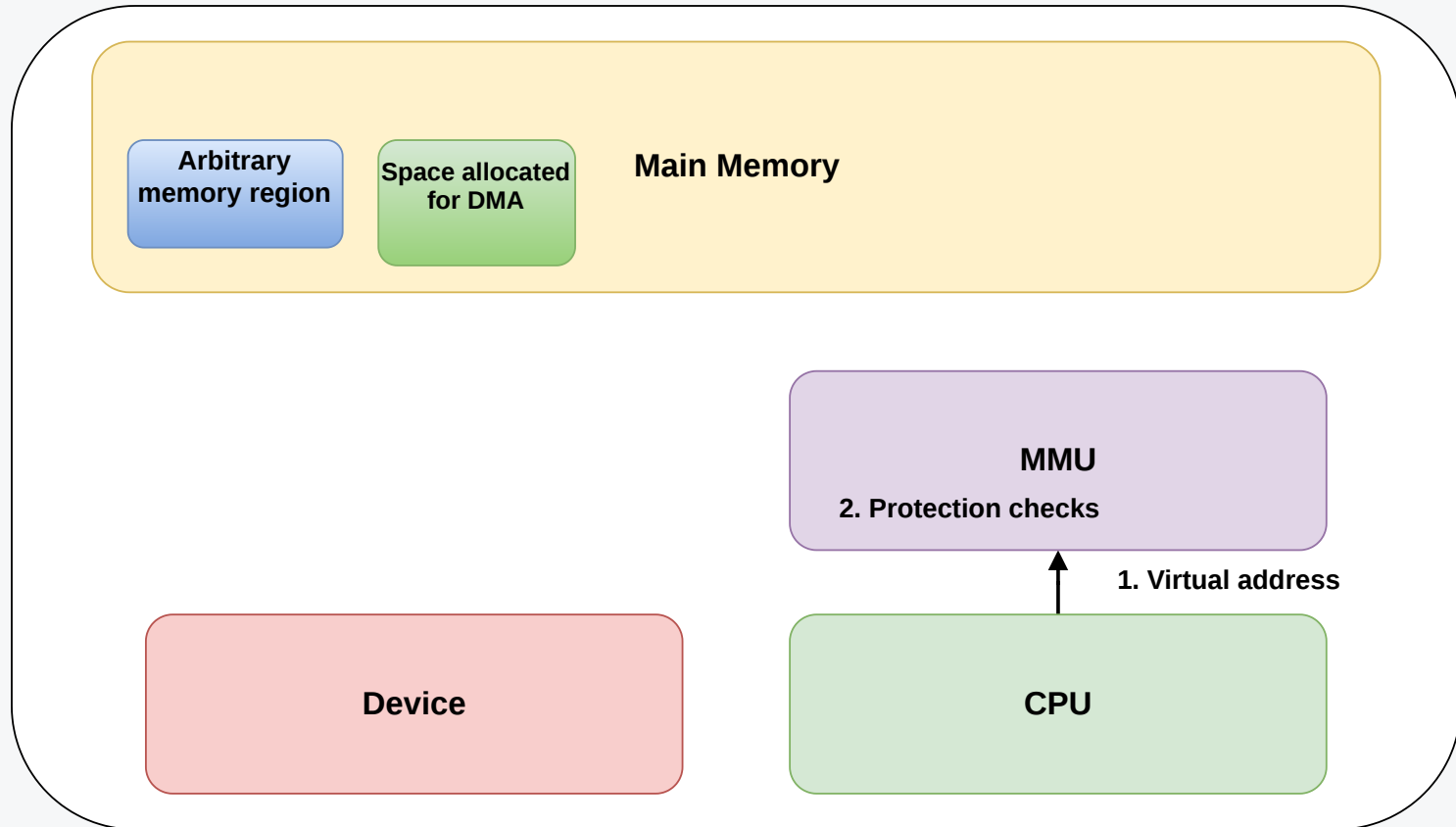


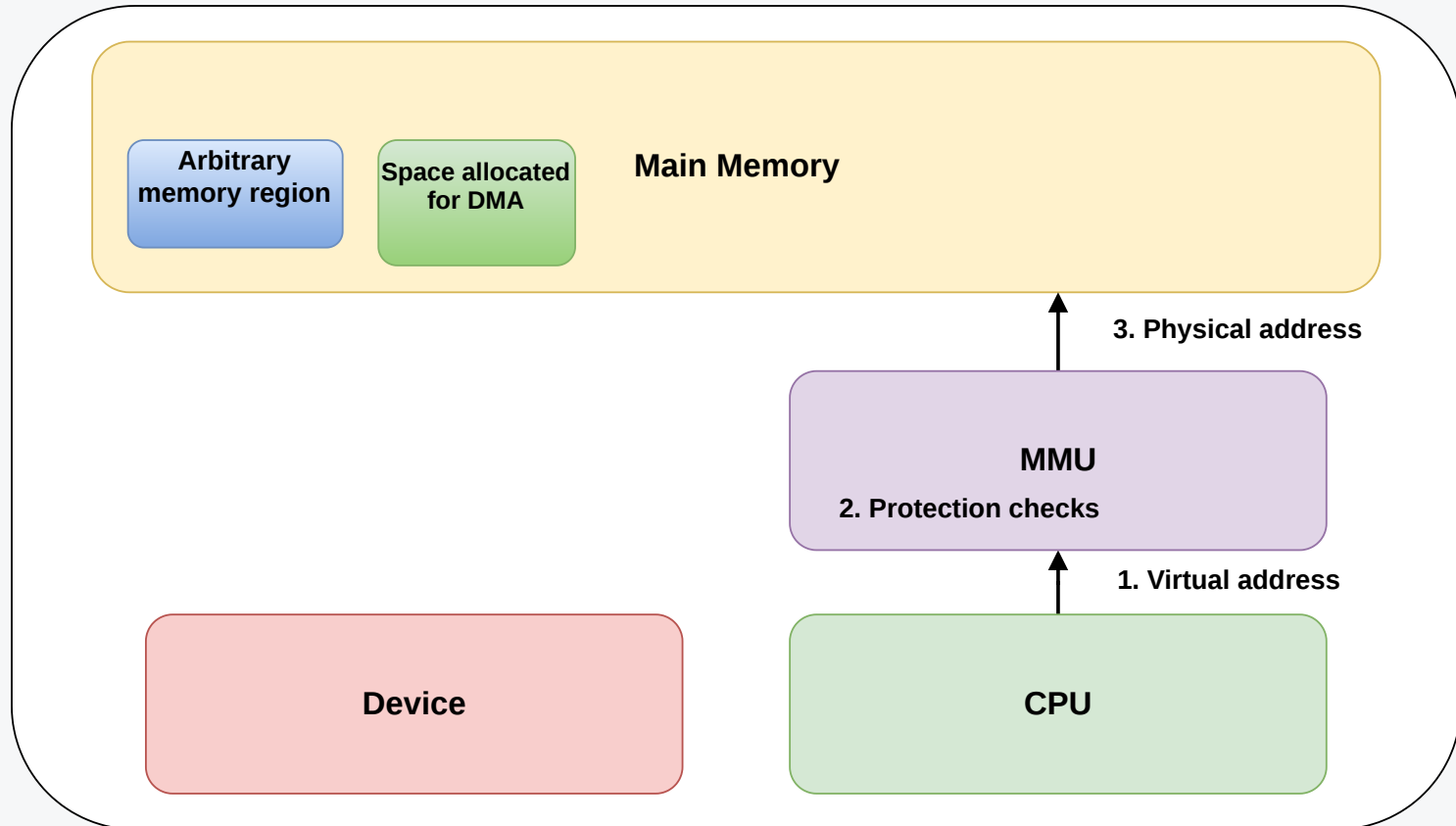
Wikipedia

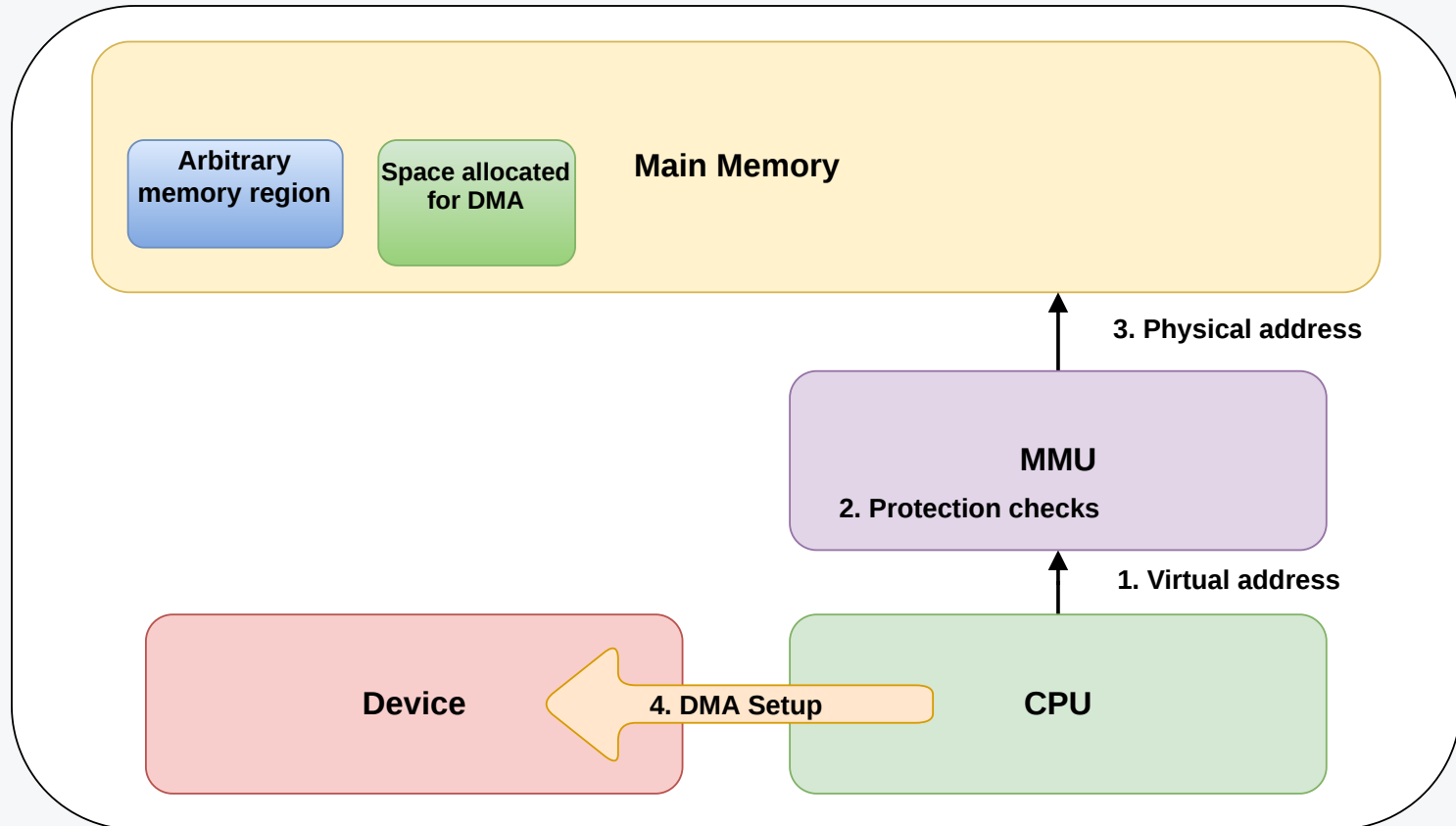


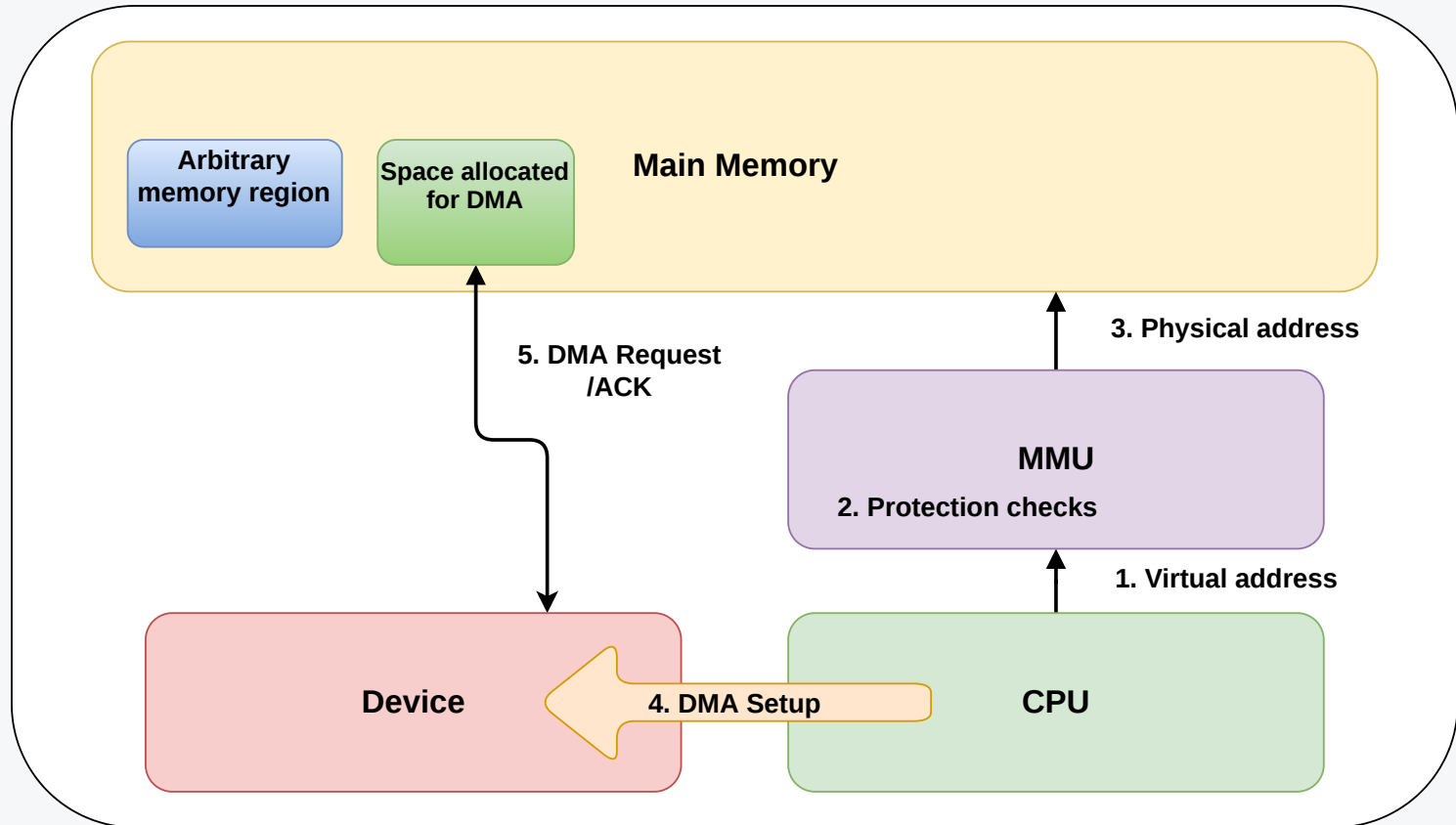


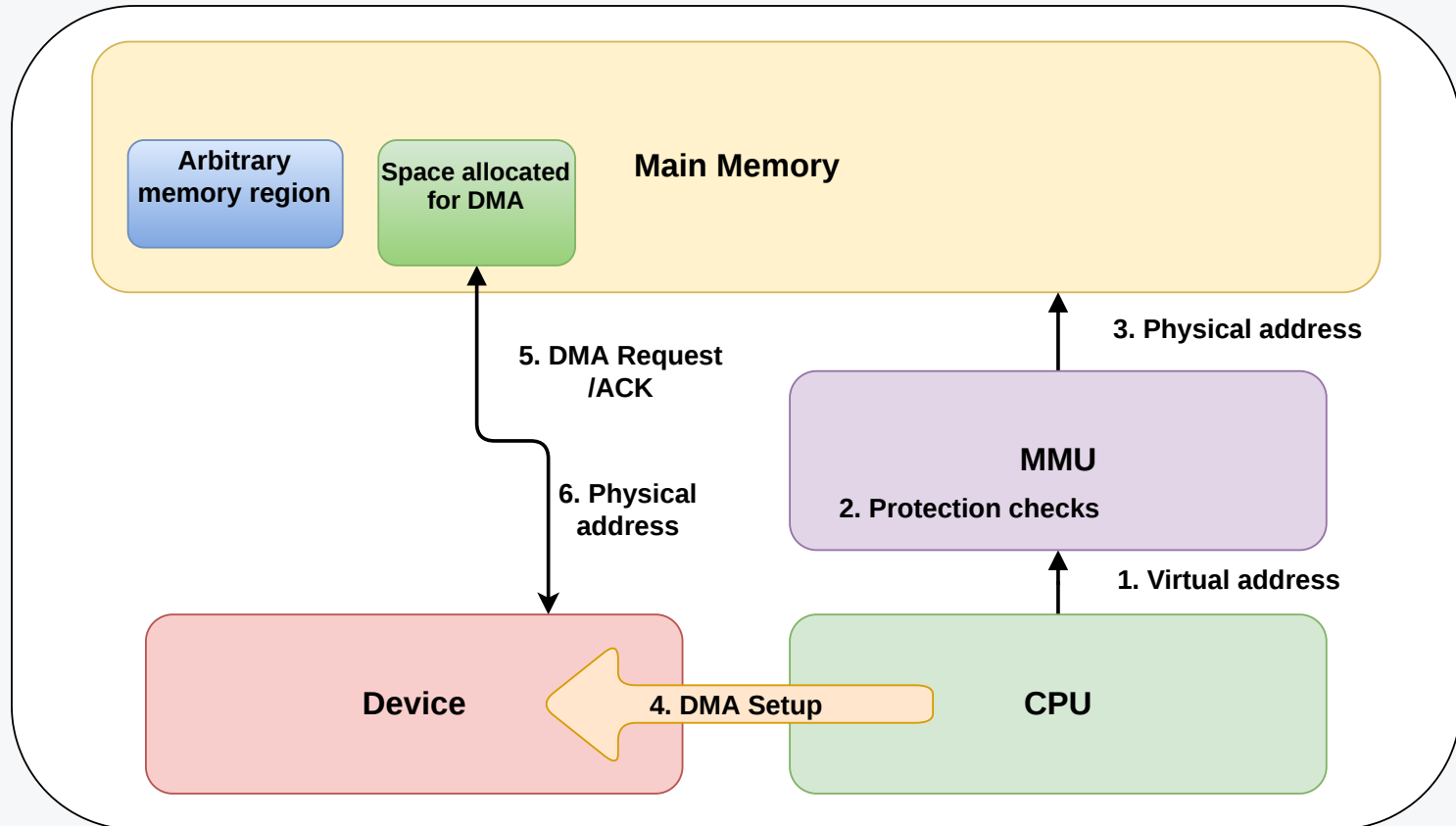


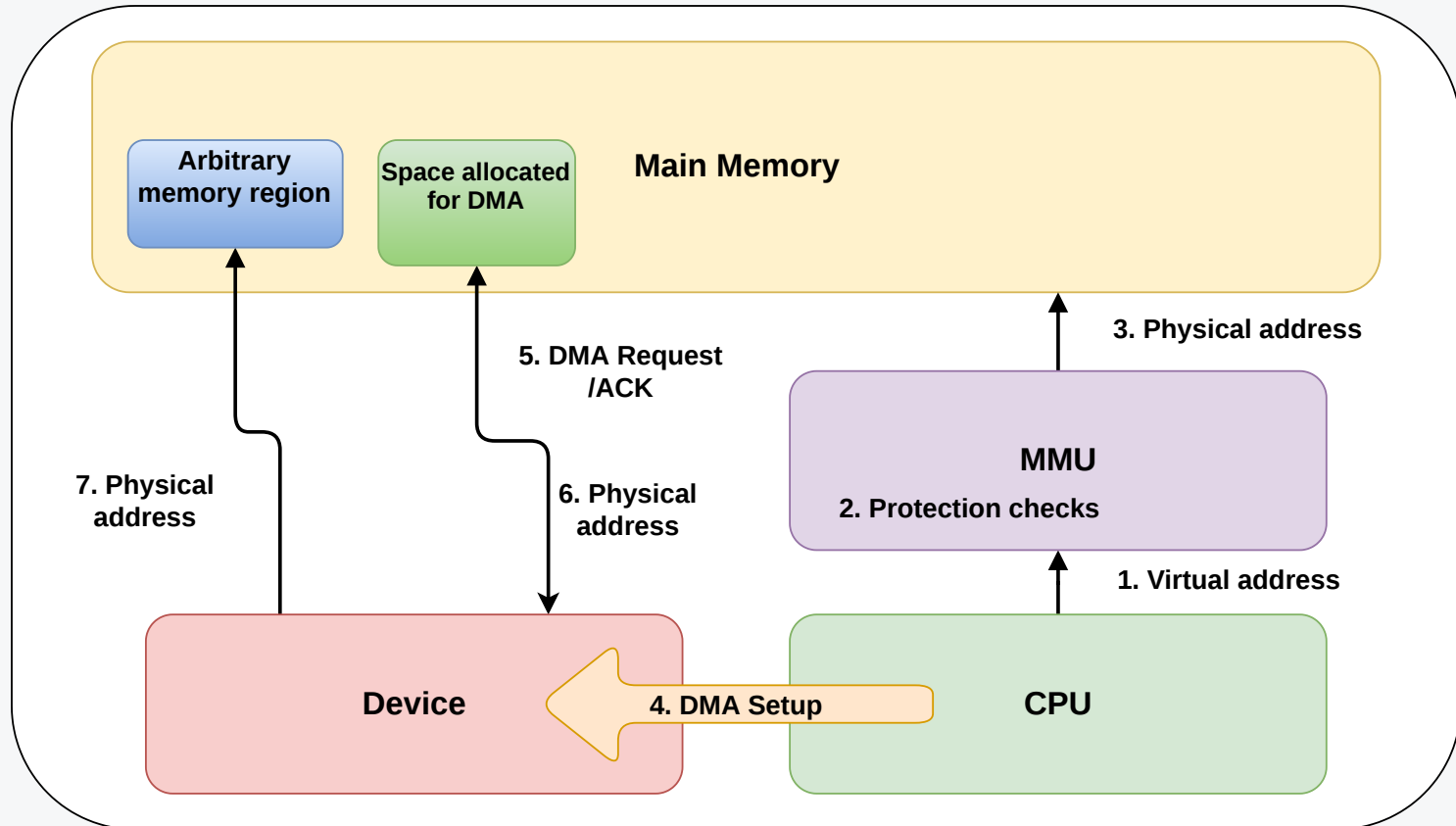


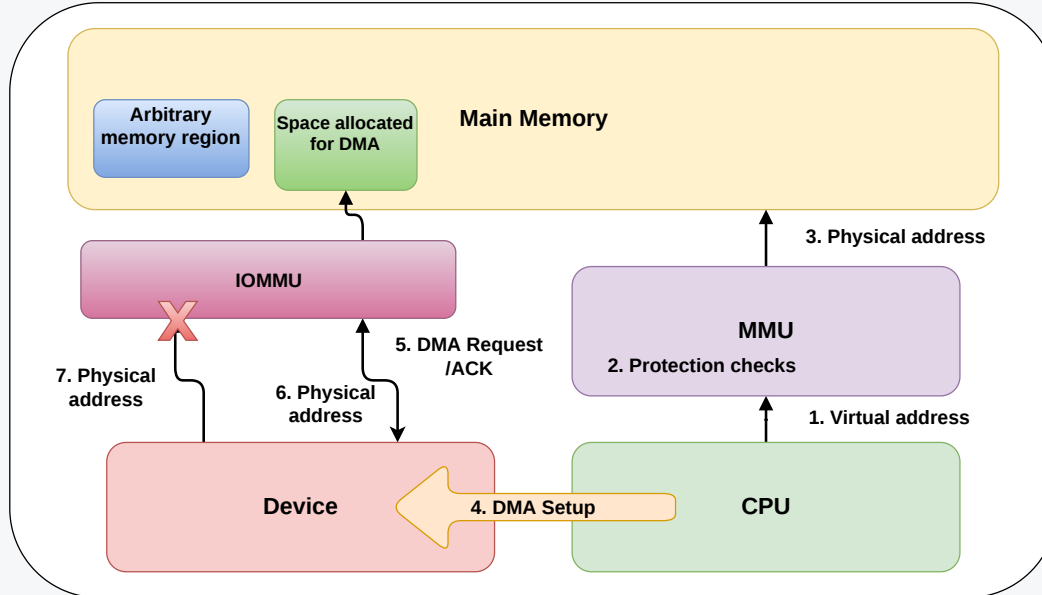












- can be initiated by
 - malicious device
 - buggy device driver
- hardware enforced memory protection mitigate DMA attacks - its one of main functions of IOMMU

- each guest need access to interrupt controller for IPI and device interrupts
- high rate of device and interprocessor interrupts may cause significant overhead
- IOMMU accelerates delivery of virtual interrupts from I/O devices to virtual processor
- virtualized interrupts are delivered to VM without hypervisor intervention
- CPU and IOMMU maintain interrupt state in Guest Virtual APIC table

- **Virtual address translation for DMA** - this is for guest operating systems under hypervisor control, if guest OS want to perform DMA operation for given device, this operation needs continuous supervision of hypervisor, what cause big overhead (even 30%), having IOMMU gives ability to program DMA operation once and make it work without performance overhead
- **Page tables sharing** - if 2 or more devices can use the same physical memory, what essentially means both can use VA which translate to the same PA, then it can simplify programming, IOMMU gives that ability
- **PCI passthrough** - using mentioned features we can assign given hardware (example NIC) to one VM and do not give access to that hardware to anything else, that means PCI passthrough, performance of this device should be not much different then real hardware

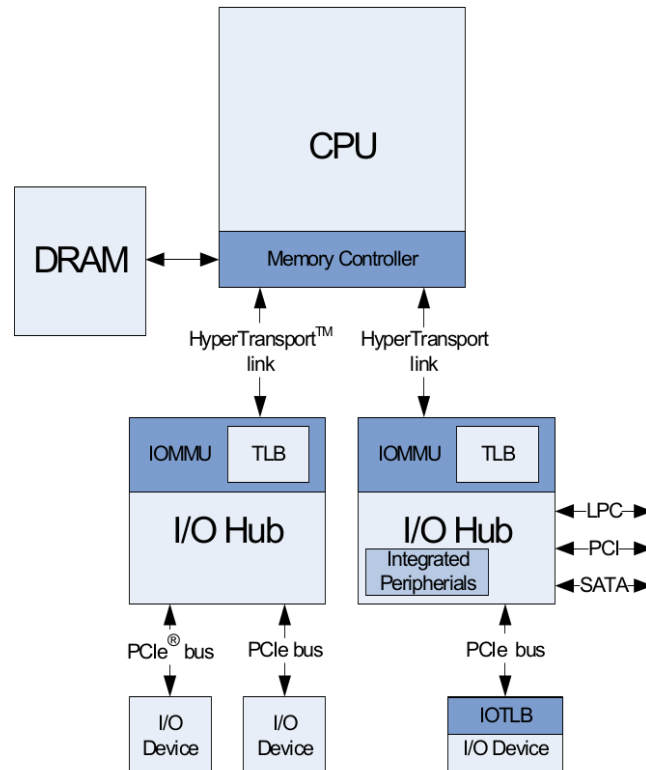


Figure 1: Example Platform Architecture

AMD I/O Virtualization Technology (IOMMU) Specification

- All AMD IOMMU implementations support basic features
 - device virtual to physical address translation
 - interrupt remapping
 - access permission checking
- Extended features are described through IOMMU Extended Feature Register (EFRSup)

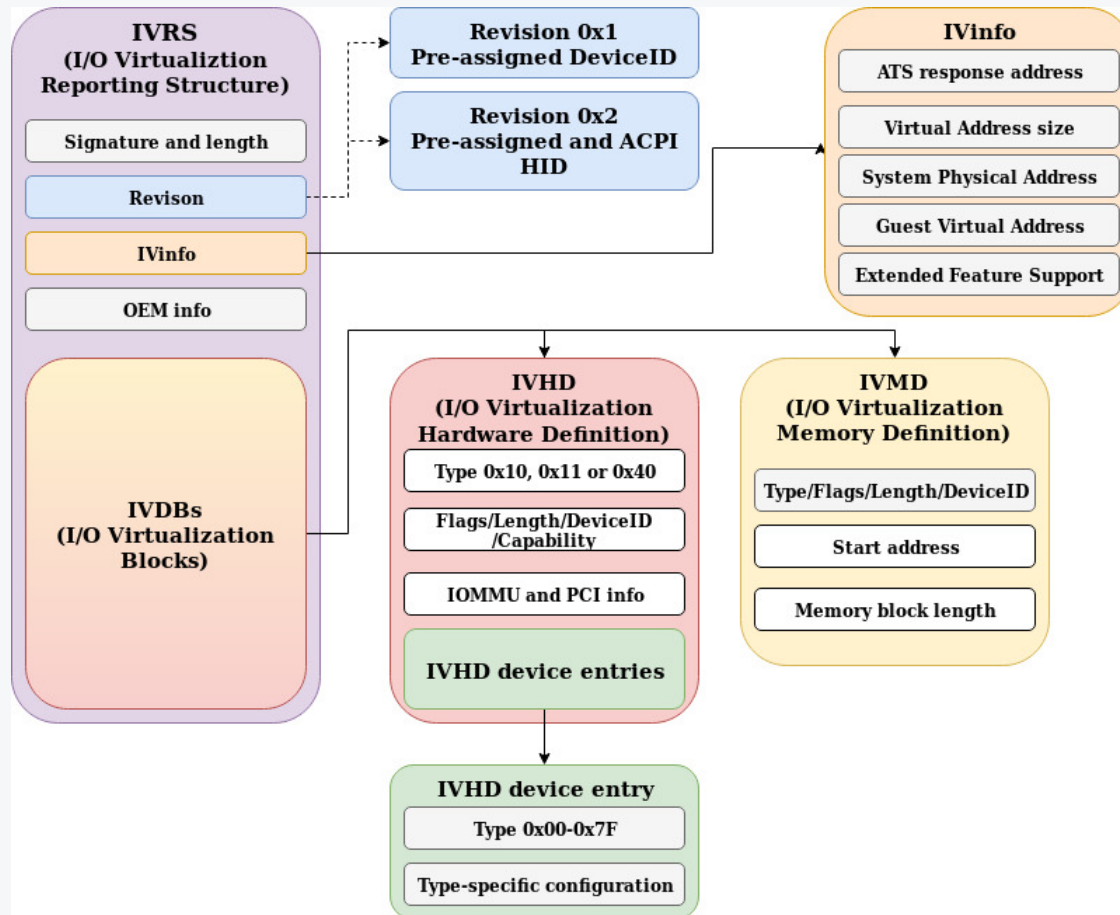
63	62	61											55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36							32
Reserved	ForcePhyDestSup	Reserved										InvtIbTypeSup	Reserved	HDSup	AttrFWSup	EPHSup	HASup	GIOSup	Reserved	MslCapMmioSup	PerfOptSup	BlkStopMrkSup	MarcSup		PPRAutoRspSup		PPREarlyOF Sup	DevTblSegSup		USSup	PASmax[4:0]								
31	30	29	28	27	26	25	24	23			21	20			18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0						
Reserved	DualEventLogSup		Reserved		DualPprLogSup		GAMSup		SmiFRC		SmiFSup		GLXSup		GATS		HATS		PCSup		HESup		GASup		IASup		Reserved	GTSup		NXSup		Rsv		PPRSup		PreFSup			

AMD I/O Virtualization Technology (IOMMU) Specification

- AMD IOMMU spec mention 28 software-visible features
 - Base support: I/O Page Tables for Host Translations, Interrupt Remapping
 - Capabilities header: EFRSup enable/disable, IotlbSup enable/disable
 - EFRSup: all other features
- not enough time to present all, but we will mention most interesting ones
- SMI Filter
 - intercept unexpected SMIs
 - blocks or defers suspected SMI sources
- Guest Page Table NX and Access Protection
 - works the same as usual protection but applied to I/O initiated by device
- Memory Address Routing and Control (MARC)
 - give device ability to bypass IOMMU for low-latency needs

- Disable IOMMU simply pass whole traffic
- Enabled IOMMU intercepts traffic from downstream devices
 - perform permission checks
 - translate addresses on the request
 - send translated to system memory
- IOMMU reads 3 tables to perform above functions
 - permission checks table
 - address translation table
 - interrupt remapping table
- all tables are cached and it is software responsibility to invalidate cache when configuration changes
- IOMMU detects following events and log them in system memory
 - IOMMU event responses - all IOMMU responses are logged
 - I/O Page Faults - e.g. lack of permissions, page not present
 - Memory Access Errors - e.g. uncorrectable ECC errors

- there is no way one system can utilize all features and options
- it is recommended that:
 - IOMMU initialization is performed by firmware - system software/hypervisor should face configured IOMMU
 - firmware should describe IOMMU in dedicated ACPI tables
 - firmware must preserve or restore IOMMU configuration across power management state transition
- IOMMU configuration should be performed as early as possible
- ACPI tables creation have to be done after PCI enumeration



- March 2016 -Initial work published by Kyösti Mälkki
 - relied on AGESA returned values
 - custom IVRS patching
 - single isolation domain
 - not stable across reboots
- June 2016 - Independent implementation by Timothy Pearson from Raptor Engineering
 - deeply nested and recursive code
 - quite a lot of hard coded values
 - IVHD Type 10 used
- Both implementation had its own problems, but without above work we would not be able to move forward, so thanks to initial authors
- June 2018 - patches picked by 3mdeb
 - based on combined work - some values came from AGESA and some came from hand-crafted ACPI tables
 - tested under Xen 4.8 and Debian with 4.14.y Dom0

- Code path

```
src/include/device/pci_ids.h  
src/mainboard/pcengines/apu2/variants/apu{2,3,4,5}/devicetree.cb  
src/northbridge/amd/pi/00730F01/Makefile.inc  
src/northbridge/amd/pi/00730F01/iommu.c  
src/northbridge/amd/pi/00730F01/northbridge.c  
src/northbridge/amd/pi/agesawrapper.c  
9 files changed, 326 insertions(+), 17 deletions(-)
```

- IOMMU PCI ID
- Enable IOMMU
- Add IOMMU driver compilation
- Basic IOMMU driver
- Key changes (IVRS ACPI table)
- Correct AGESA init

```
static struct boot_state boot_states[] = {
    BS_INIT_ENTRY(BS_PRE_DEVICE, bs_pre_device),
    BS_INIT_ENTRY(BS_DEV_INIT_CHIPS, bs_dev_init_chips),
    BS_INIT_ENTRY(BS_DEV_ENUMERATE, bs_dev_enumerate),
    BS_INIT_ENTRY(BS_DEV_RESOURCES, bs_dev_resources),
    BS_INIT_ENTRY(BS_DEV_ENABLE, bs_dev_enable),
    BS_INIT_ENTRY(BS_DEV_INIT, bs_dev_init),
    BS_INIT_ENTRY(BS_POST_DEVICE, bs_post_device),
    BS_INIT_ENTRY(BS_OS_RESUME_CHECK, bs_os_resume_check),
    BS_INIT_ENTRY(BS_OS_RESUME, bs_os_resume),
    BS_INIT_ENTRY(BS_WRITE_TABLES, bs_write_tables), <- HERE
    BS_INIT_ENTRY(BS_PAYLOAD_LOAD, bs_payload_load),
    BS_INIT_ENTRY(BS_PAYLOAD_BOOT, bs_payload_boot),
};
```

```
bs_write_tables - src/lib/hardwaremain.c
-> write_tables - src/lib/coreboot_table.c
-> arch_write_tables - src/arch/x86/tables.c
-> write_acpi_table - src/arch/x86/acpi.c
-> write_acpi_tables (aka agesa_write_acpi_tables)
    - src/northbridge/amd/pi/00730F01/northbridge.c
```

```

1. acpi_dump_ivrs: ivrs->header.signature: IVRSx
2. acpi_dump_ivrs: ivrs->header.length: 0x78
3. acpi_dump_ivrs: ivrs->header.revision: 0x2
4. acpi_dump_ivrs: ivrs->header.checksum: 0x9a
5. acpi_dump_ivrs: ivrs->header.oem_id: AMD AGESA
6. acpi_dump_ivrs: ivrs->header.oem_table_id: AGESA
7. acpi_dump_ivrs: ivrs->header.oem_revision: 0x1
8. acpi_dump_ivrs: ivrs->header.asl_compiler_id: AMD
9. acpi_dump_ivrs: ivrs->header.asl_compiler_revision: 0x0
10. acpi_dump_ivrs: ivrs->iv info: 0x203040
11. acpi_dump_ivrs: ivrs->ivhd.type: 0x10
12. acpi_dump_ivrs: ivrs->ivhd.flags: 0xfe
13. acpi_dump_ivrs: ivrs->ivhd.length: 0x48
14. acpi_dump_ivrs: ivrs->ivhd.device_id: 0x2
15. acpi_dump_ivrs: ivrs->ivhd.capability_offset: 0x40
16. acpi_dump_ivrs: ivrs->ivhd.iommu_base_low: 0xf7f00000
17. acpi_dump_ivrs: ivrs->ivhd.iommu_base_high: 0x0
18. acpi_dump_ivrs: ivrs->ivhd.pci_segment_group: 0x0
19. acpi_dump_ivrs: ivrs->ivhd.iommu info: 0x1300
20. acpi_dump_ivrs: ivrs->ivhd.iommu feature info: 0x48824
21. 1001962f: 03 08 00 00 04 fe ff 00 43 00 ff 00 00 a4 00 00 .....C.....
22. 1001963f: 04 ff ff 00 00 00 00 48 00 00 00 a0 00 02 .....H.....
23. 1001964f: 48 00 00 d7 04 a0 00 01 48 00 00 00 05 00 00 01 H.....H.....

```

```

1. acpi_dump_ivrs: ivrs->header.signature: IVRS
2. acpi_dump_ivrs: ivrs->header.length: 0x108
3. acpi_dump_ivrs: ivrs->header.revision: 0x1
4. acpi_dump_ivrs: ivrs->header.checksum: 0x0
5. acpi_dump_ivrs: ivrs->header.oem_id: CORE COREBOOT
6. acpi_dump_ivrs: ivrs->header.oem_table_id: COREBOOT
7. acpi_dump_ivrs: ivrs->header.oem_revision: 0x0
8. acpi_dump_ivrs: ivrs->header.asl_compiler_id: CORE
9. acpi_dump_ivrs: ivrs->header.asl_compiler_revision: 0x0
10. acpi_dump_ivrs: ivrs->iv info: 0x203040
11. acpi_dump_ivrs: ivrs->ivhd.type: 0x10
12. acpi_dump_ivrs: ivrs->ivhd.flags: 0x1e
13. acpi_dump_ivrs: ivrs->ivhd.length: 0xd8
14. acpi_dump_ivrs: ivrs->ivhd.device_id: 0x2
15. acpi_dump_ivrs: ivrs->ivhd.capability_offset: 0x40
16. acpi_dump_ivrs: ivrs->ivhd.iommu_base_low: 0xf7f00000
17. acpi_dump_ivrs: ivrs->ivhd.iommu_base_high: 0x0
18. acpi_dump_ivrs: ivrs->ivhd.pci_segment_group: 0x0
19. acpi_dump_ivrs: ivrs->ivhd.iommu info: 0x1300
20. acpi_dump_ivrs: ivrs->ivhd.iommu feature info: 0x48000
21. cfeb0eb8: 48 00 00 d7 00 a0 00 02 02 02 00 00 00 00 00 H.....
22. cfeb0ec8: 02 10 00 00 00 00 00 02 12 00 00 00 00 00 00 .....
23. cfeb0ed8: 02 00 01 00 00 00 00 02 13 00 00 00 00 00 00 .....
24. cfeb0ee8: 02 00 02 00 00 00 00 02 14 00 00 00 00 00 00 .....
25. cfeb0ef8: 02 00 03 00 00 00 00 02 40 00 00 00 00 00 00 .....@.....
26. cfeb0f08: 02 80 00 00 00 00 00 02 88 00 00 00 00 00 00 .....
27. cfeb0f18: 02 98 00 00 00 00 00 02 a0 00 97 00 00 00 00 .....
28. cfeb0f28: 02 a3 00 00 00 00 00 02 a7 00 00 00 00 00 00 .....
29. cfeb0f38: 02 c0 00 00 00 00 00 02 c1 00 00 00 00 00 00 .....
30. cfeb0f48: 02 c2 00 00 00 00 00 02 c3 00 00 00 00 00 00 .....
31. cfeb0f58: 02 c4 00 00 00 00 00 02 c5 00 00 00 00 00 00 .....
32. cfeb0f68: 48 00 00 05 00 00 01 48 00 00 d7 04 a0 00 01 H.....H.....

```


- install Xen or boot over PXE
- setup selected device for PCI passthrough

```
modprobe xen-pciback  
xl pci-assignable-add 02:00.0
```

- after above operations device should not be visible from dom0
- create Xen HVM guest config for installation purposes and perform OS installation in VM
- boot VM and verify `lspci`

```
debian@debian:~$ lspci -s 00:04.0 -vvv
00:04.0 Ethernet controller: Intel Corporation I210 Gigabit Network Connection (rev 03)
  Subsystem: Intel Corporation I210 Gigabit Network Connection
  Physical Slot: 4
  Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr- Stepping- SERR- FastB2B- DisINTx+
  Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort- <TAbort- <MAbort- >SERR- <PERR- INTx-
  Latency: 0
  Interrupt: pin A routed to IRQ 32
  Region 0: Memory at f3000000 (32-bit, non-prefetchable) [size=128K]
  Region 2: I/O ports at c100 [size=32]
  Region 3: Memory at f3030000 (32-bit, non-prefetchable) [size=16K]
  Capabilities: <access denied>
  Kernel driver in use: igb
  Kernel modules: igb

debian@debian:~$
```

```
name = "debian-9.5.0"
builder = "hvm"
vcpus = 2
memory = 2048
pci = [ '02:00.0' ]
disk=[ '/root/debian-9.5.0-amd64-netinst.iso,,hdc,cdrom', '/dev/vg0/debian,,hdb,rw' ]
vnc=1
vnclisten='apu2_ip_addr'
boot='d'
```

- Xen 4.8
- Debian stretch
- Linux 4.14.59 - requires well-crafted kernel with all Xen requirements filled
- pfSense HVM guest
- iperf

Detailed configuration you can find in blog post "pfSense as HVM guest on PC Engines apu2" on 3mdeb website.

Passthrough in VM

```
(speedtest-venv) root@debian:~# iperf -c 192.168.3.101
-----
Client connecting to 192.168.3.101, TCP port 5001
TCP window size: 85.0 KByte (default)
-----
[  3] local 192.168.3.100 port 35958 connected with 192.168.3.101 port 5001
[ ID] Interval           Transfer     Bandwidth
[  3]  0.0-10.0 sec   1.10 GBytes    941 Mbits/sec
(speedtest-venv) root@debian:~#
```

Server

```
(speedtest-venv) root@apu2:~# iperf -s -B 192.168.3.101
-----
Server listening on TCP port 5001
Binding to local address 192.168.3.101
TCP window size: 85.3 KByte (default)
-----
[  4] local 192.168.3.101 port 5001 connected with 192.168.3.102 port 34004
[ ID] Interval           Transfer     Bandwidth
[  4]  0.0-10.0 sec   1.10 GBytes    941 Mbits/sec
```

- DMA attack protection verification is not trivial
 - after consulting PCILeech maintainer we are in process of using PCIeScreamer to exercise PC Engines and test it against DMA attacks
- Google fuzzed Intel VT-d to check if it correctly protect memory, but no details about tools and methodology
- maybe good idea for next talk

- PCI pass through tested on Debian and pfSense
- performance prove that feature works correctly
- there were no longevity testing
- sporadic hangs in Xen hypervisor

(XEN) CPU1: No irq handler for vector e7 (IRQ -2147483648)

(XEN) CPU2: No irq handler for vector e7 (IRQ -2147483648)

- we are not sure what are the correct IOMMU setting - there are lot of them and correct configuration should be enforced as soon as possible
- there are still bugs that can cause problems

- *AMD I/O Virtualization Technology (IOMMU) Specification* - 48882 Rev 3.00 December 2016
- *AMD64 Architecture Programmer's Manual Volume 2: System Programming* - 24593 Rev 3.29 December 2017

- binary blobs are bad and working around defects they may introduce is time consuming
- Future work
 - stabilize the code and merge mainline
 - enable and verify advanced IOMMU features
 - GPU passthrough (?)
 - prove that AMD IOMMU protects against DMA attacks
 - "towards reasonably secure" router/IoT gateway

Q&A