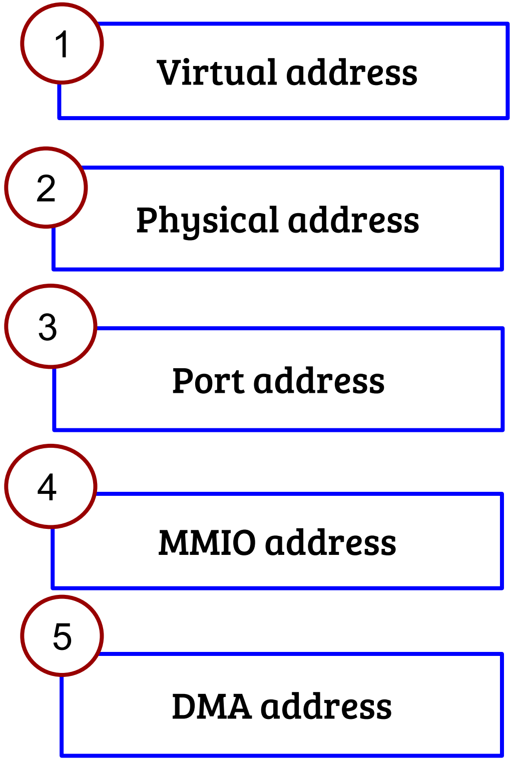
Topics in Operating Systems

Addressing in kernel

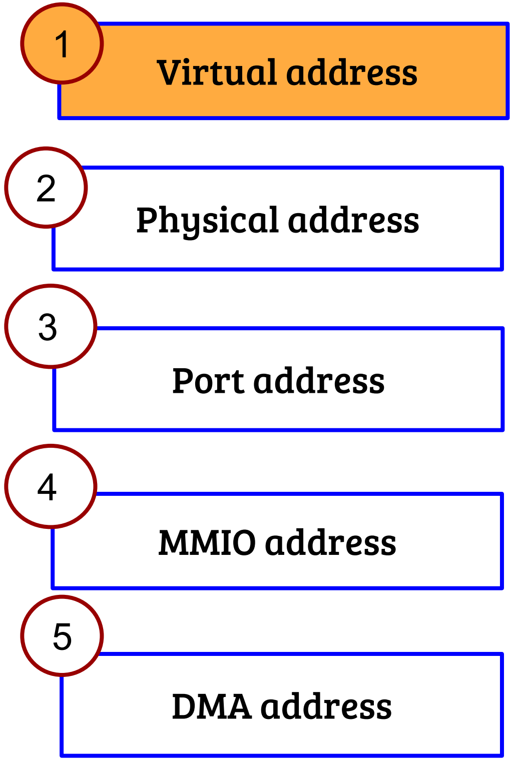
Debadatta Mishra, CSE, IITK

Address types in kernel



1. Virtual address
2. Physical address
3. Port address
4. MMIO address
5. DMA address

Kernel virtual address



1 Virtual address

2 Physical address

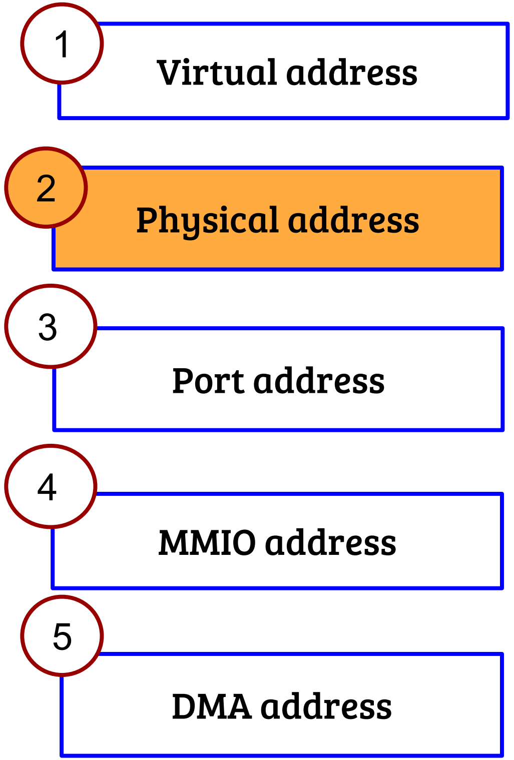
3 Port address

4 MMIO address

5 DMA address

- Direct mapping of physical memory (64TB)
  - Conversion from virtual to physical and vice-a-versa can be done using macros like __va(paddr) and __pa(vaddr)
- Physically discontinuous virtual address
  - Allocated used vmalloc( )
  - Useful when you allocate large contiguous kernel virtual address
  - Legacy: 32-bit systems required temporary virtual addresses a lot (check out highmem)

Physical address in kernel



1 Virtual address

2 Physical address

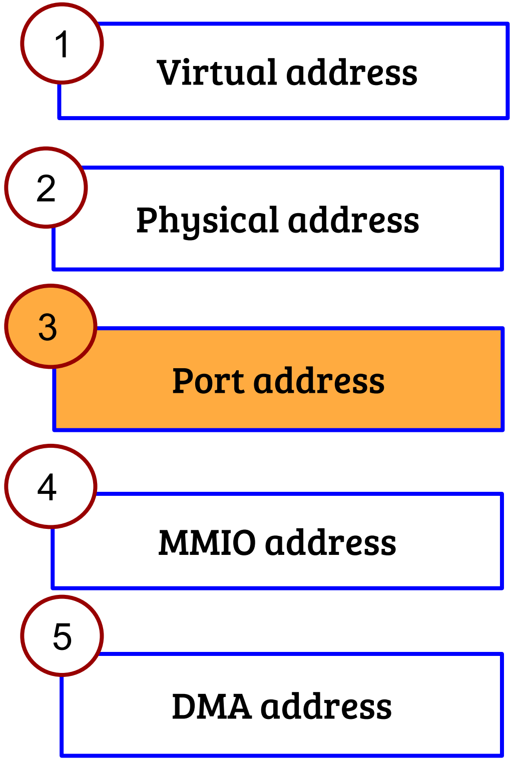
3 Port address

4 MMIO address

5 DMA address

- Two commonly used (almost interchangeable) terms
  - Page: A *struct page* type
  - Physical Frame Number (PFN): *unsigned long*
  - APIs: pfn_to_page, page_to_pfn etc.
  - How does the conversion happen?
- At the lowest level, physical allocation done through page allocation APIs (alloc_page, free_page etc.)
- Page structure contains information like mapcount, usage count etc.

Port addressing


1 Virtual address

2 Physical address

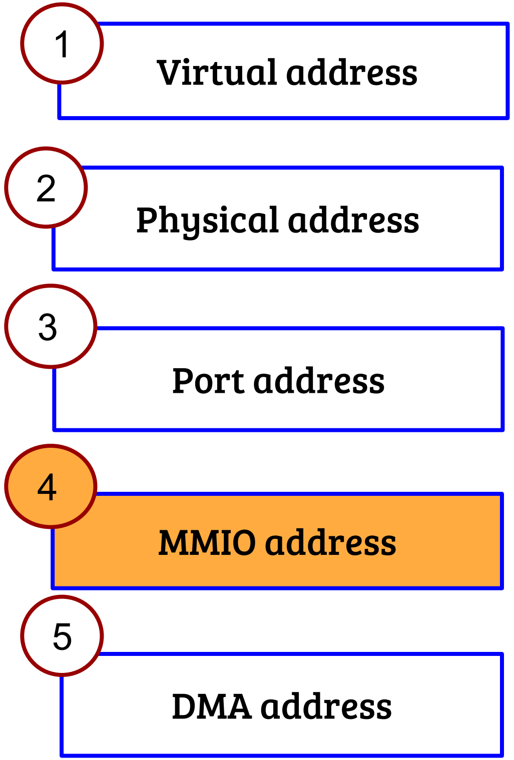
3 Port address

4 MMIO address

5 DMA address

- Device registers mapped by BIOS to port addresses
- Port addresses can be accessed directly w/o page table mapping
- However, port addresses are
    - Not memory addresses
    - Only I/O instructions (in, out) are allowed
- $cat /proc/ioports
- OSes have to use some hard coded port addresses, it is unavoidable!
- Example: Serial console in gemOS

Memory mapped I/O



1 Virtual address

2 Physical address

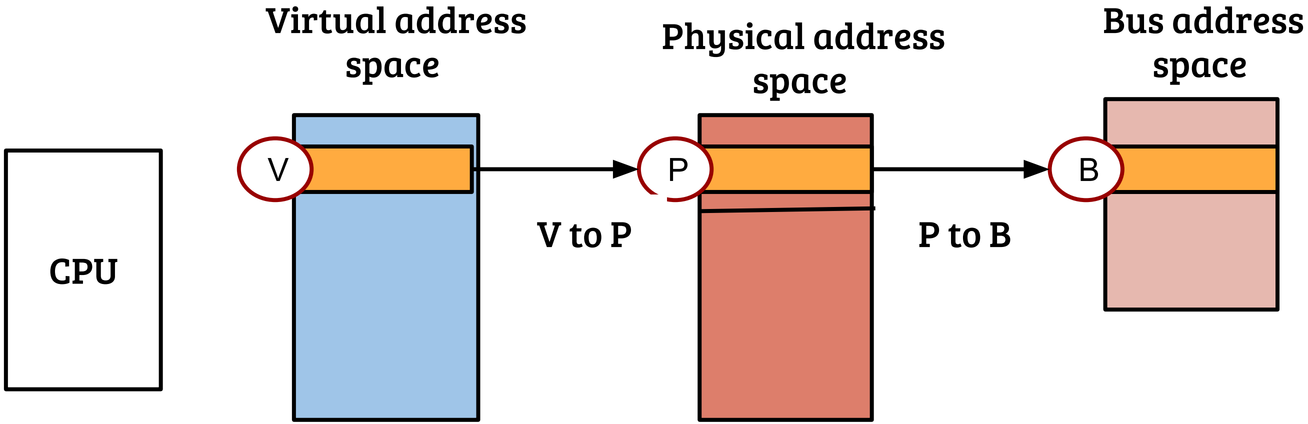
3 Port address

4 MMIO address

5 DMA address

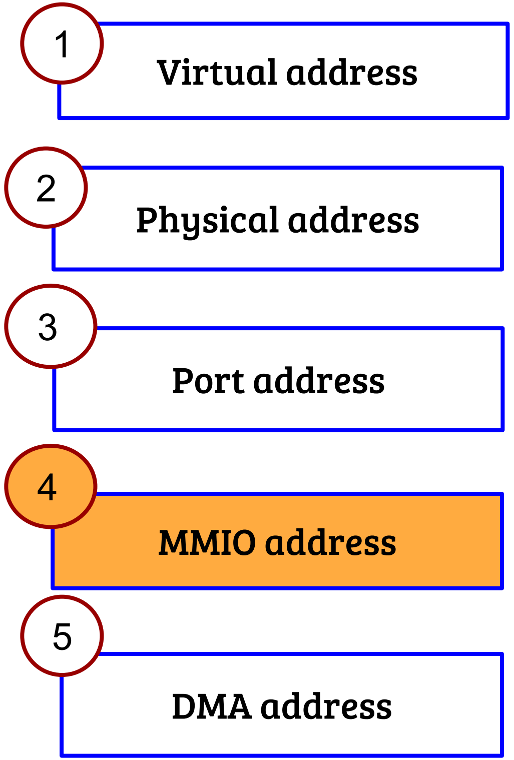
- I/O registers/memory mapped into physical address space, can be accessed like memory
- What address to use, virtual or physical?
- What extra care to be taken while accessing MMIO addresses?

Memory mapped I/O



- During device discovery, kernel maintains a device to MMIO space (/proc/iomem)
- Device driver must map the PA to V before access
- Kernel source: ioremap( ), ioread32( )
- Example: gemOS APIC setup

Memory mapped I/O



1 Virtual address

2 Physical address

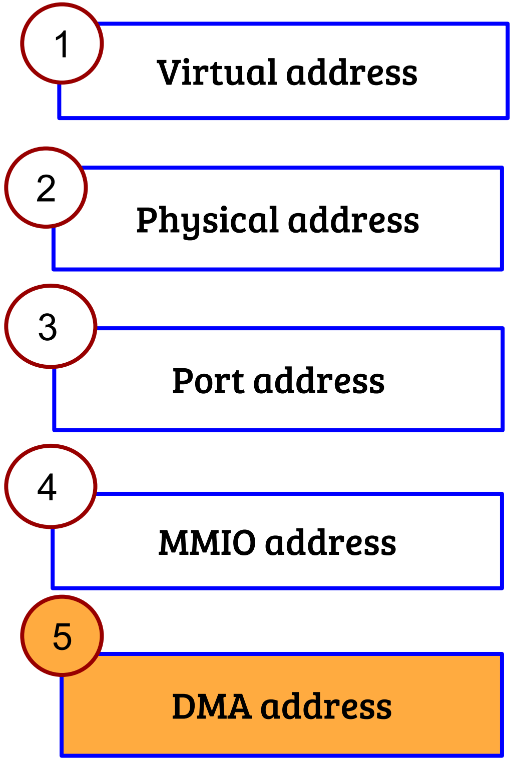
3 Port address

4 MMIO address

5 DMA address

- I/O registers/memory mapped into physical address space, can be accessed like memory
- What address to use, virtual or physical?
- Virtual
- What extra care to be taken while accessing MMIO addresses?
- Correctly timing the accesses, compiler optimizations, OOO processing

Direct memory access (DMA)



1 Virtual address

2 Physical address

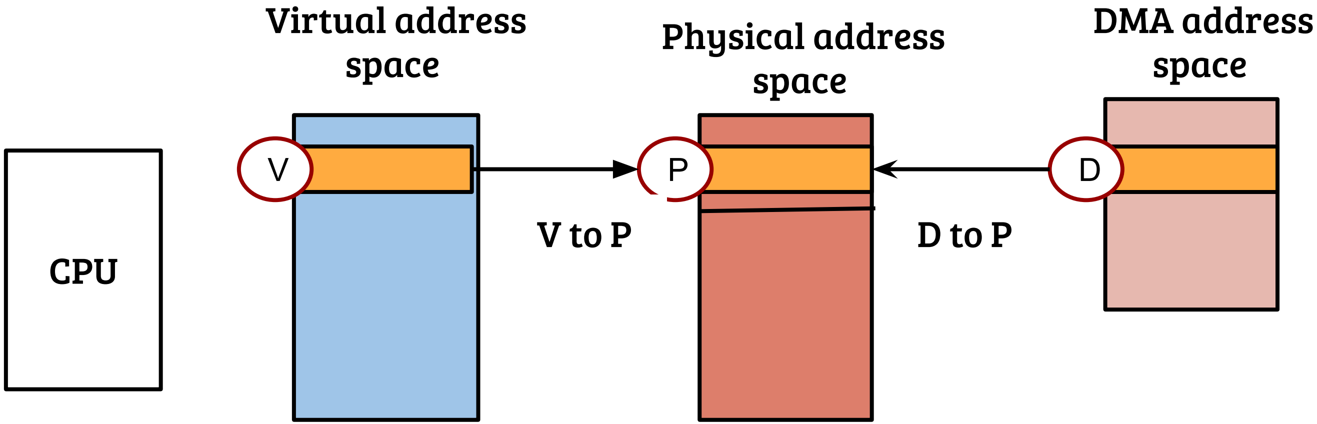
3 Port address

4 MMIO address

5 DMA address

- DMA can be used if
  - DMA controller is available
  - Device supports DMA
- DMA addresses are generated and used by DMA controller
- Can be different from physical address if IOMMU is used

DMA contd.



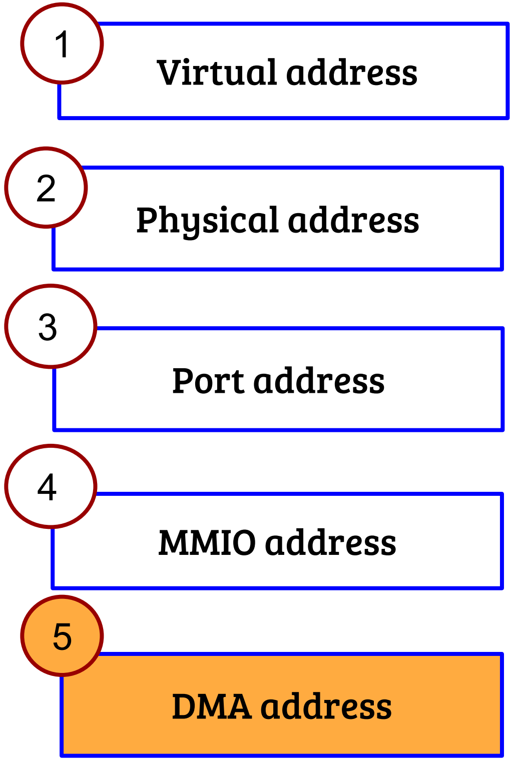
- Device driver allocates a buffer (VA = V, PA = P), no lazy allocation allowed!
- In non-IOMMU systems, device can use P directly
- With IOMMU, mapping must be setup between D → P using API's like *dma_map_single*
- Why device driver programmer has to worry about the DMA address?

DMA and interrupt handling example

```
setup_one_rcv(NIC *nic){
      dma_addr_t *mapping;
      mapping = dma_map_single(nic->dev, nic->buff_va, nic-> len,
DMA_FROM_DEVICE);
       nic->rcv_dma = mapping;
       mmio_nic(nic, DEVICE_SET_DMA);
}

irq_rcv_one(NIC *nic){
     dma_unmap_single(nic->dev, nic->buff_va, nic-> len, DMA_FROM_DEVICE);
   do_tcp_ip(nic->buff, nic->len);
 }
```

Direct memory access (DMA)



- Virtual addresses used by DMA should be mapped (don’t use vmalloc( ) address)
- DMA mapping can be of two types
  - Consistent/Coherent: mostly used throughout the driver lifetime
  - Streaming/inconsistent: used to configure receive buffer of a NIC
- Refer to kernel documentation (Documentation/DMA-API-HOWTO.txt) for details