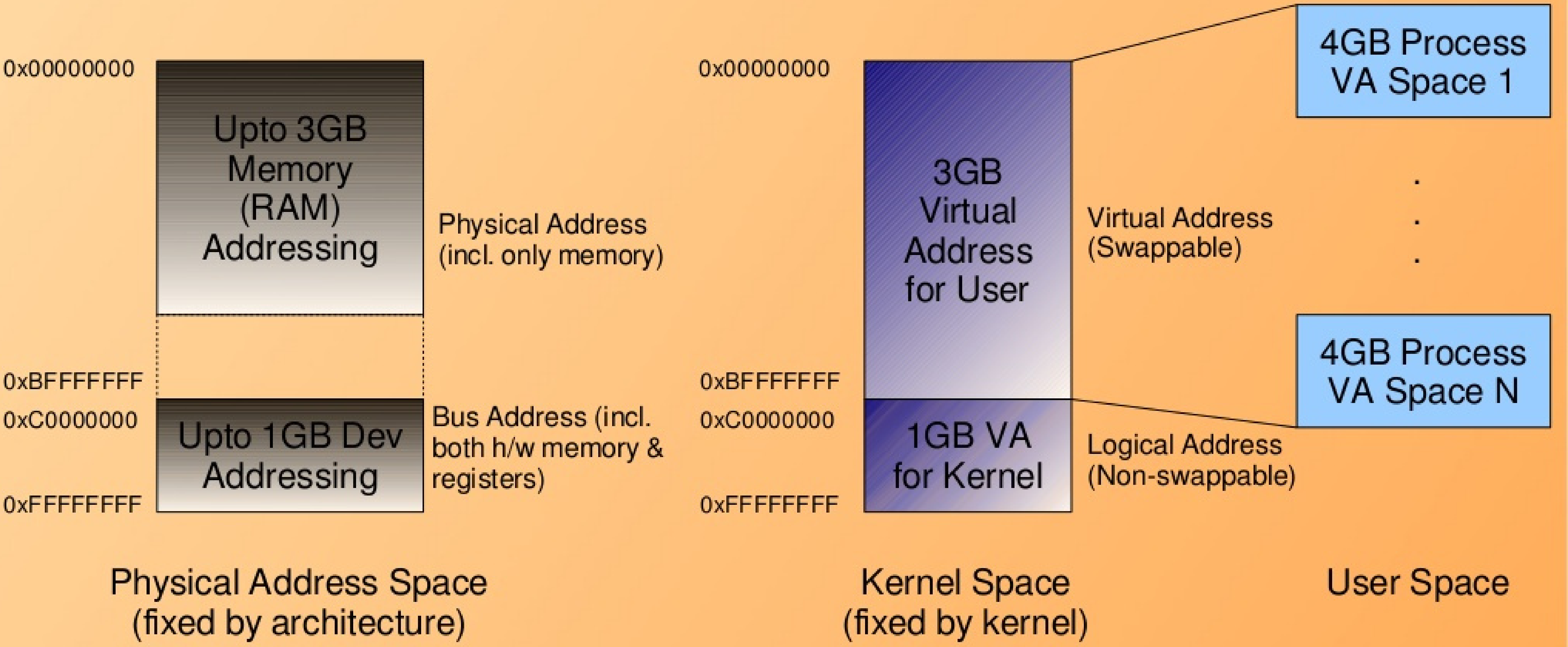
Low-level Accesses

What to Expect?

- After this session, you would know
  - Various Address Spaces in Linux
  - Role of Memory Manager in Linux
  - Accessing the Memory in Kernel Space
  - Accessing the Device or Hardware
    - Memory
    - Registers
  - Low-level Access in Drivers

Address Spaces in Linux

- An Example assuming 32-bit architecture


0x00000000
Upto 3GB Memory (RAM) Addressing
Physical Address (incl. only memory)
0xBFFFFFFF
0xC0000000
Upto 1GB Dev Addressing
Bus Address (incl. both h/w memory & registers)
0xFFFFFFFF
Physical Address Space (fixed by architecture)
0x00000000
3GB Virtual Address for User
Virtual Address (Swappable)
0xBFFFFFFF
0xC0000000
1GB VA for Kernel
Logical Address (Non-swappable)
0xFFFFFFFF
Kernel Space (fixed by kernel)
4GB Process VA Space 1
.
.
.
4GB Process VA Space N
User Space

Linux Memory Manager

- Provides Access Control to h/w & memory resources
- Provides Dynamic Memory to kernel sub-system
  - Drivers
  - File Systems
  - Stacks
- Provides Virtual Memory to Kernel & User space
  - Kernel & User Processes run in their own virtual address spaces
  - Providing the various features of a Linux system
    - System reliability, Security
    - Communication
    - Program Execution Support

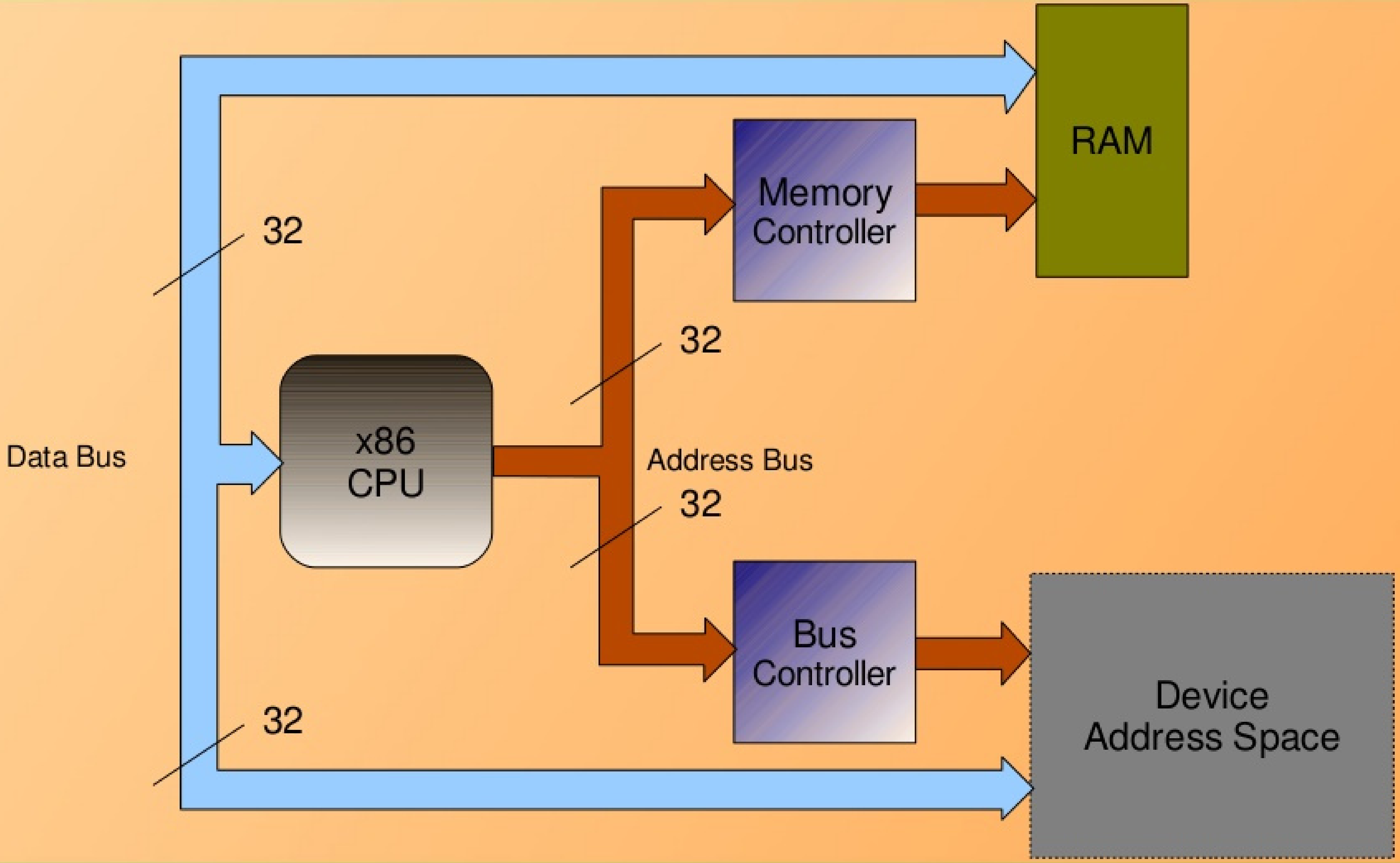
Kernel Space Memory Access

- Virtual Address for Physical Address
  - Header: <linux/gfp.h>
    - unsigned long __get_free_pages(flags, order); etc
    - void free_pages(addr, order); etc
  - Header: <linux/slab.h>
    - void *kmalloc(size_t size, gfp_t flags);
      - GFP_USER, GFP_KERNEL, GFP_DMA
    - void kfree(void *obj);
  - Header: <linux/vmalloc.h>
    - void *vmalloc(unsigned long size);
    - void vfree(void *addr);

Kernel Space Device Access

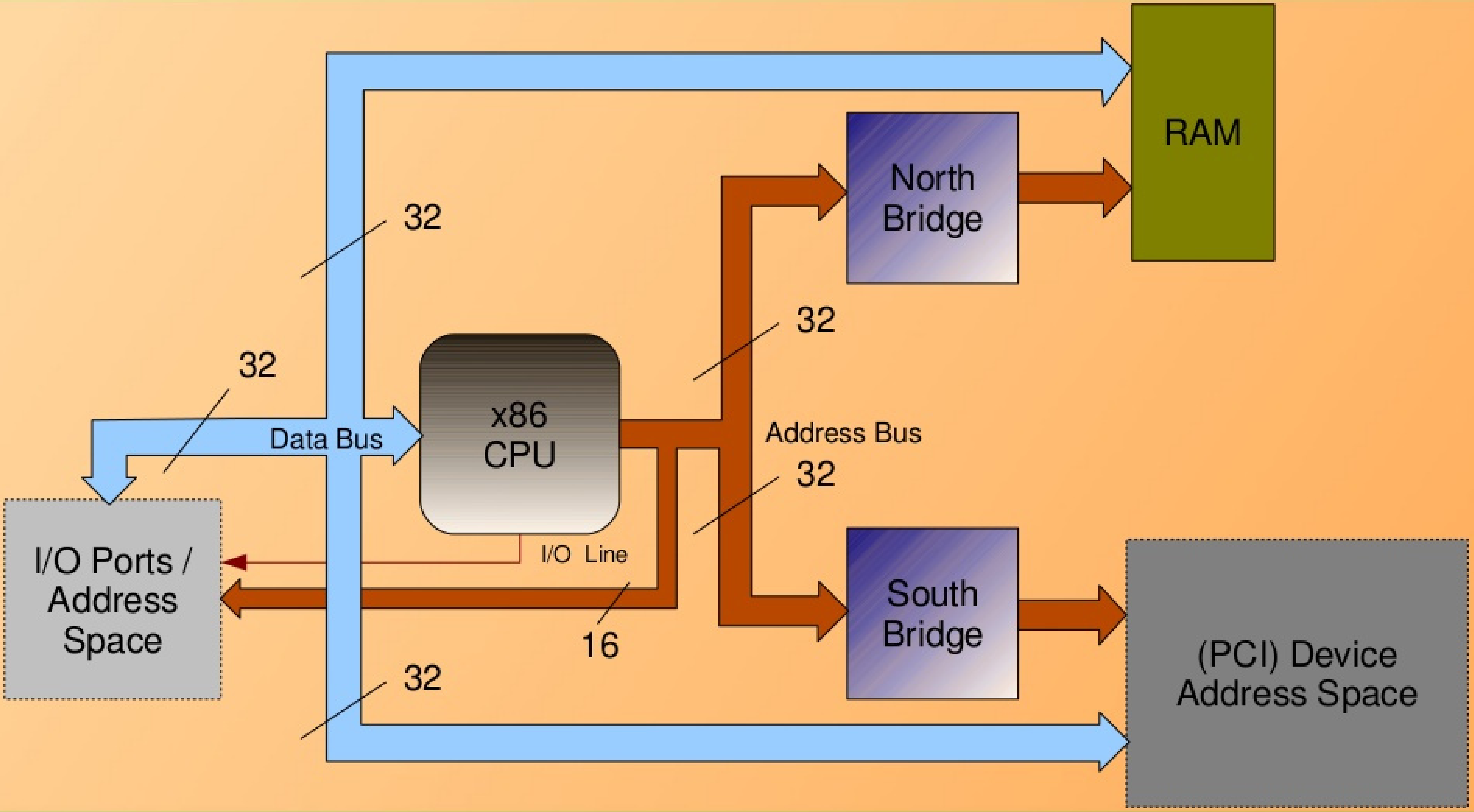
- Virtual Address for Bus/IO Address
  - Header: <asm/io.h>
    - void *ioremap(phys_addr_t bus_addr, unsigned long size);
    - void iounmap(void *addr);
- I/O Memory Access
  - Header: <asm/io.h>
    - u[8|16|32] ioread[8|16|32](void *addr);
    - void iowrite[8|16|32](u[8|16|32] value, void *addr);
- Kernel Window: /proc/iomem
- Access Permissions
  - Header: <linux/ioport.h>
    - struct resource *request_mem_region(resource_size_t start, resource_size_t size, label);
    - void release_mem_region(resource_size_t start, resource_size_t size);

x86 Memory & Device Access



x86 Hardware Architecture

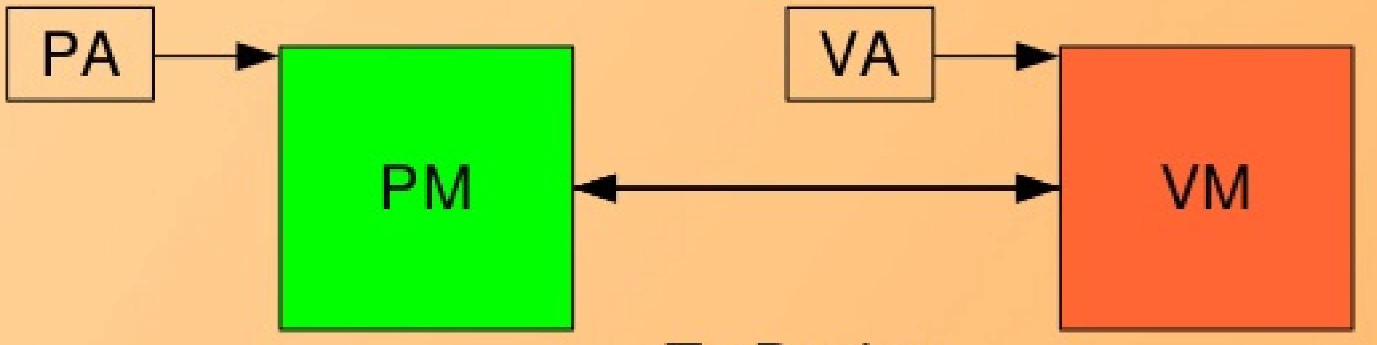
complete



I/O Access (x86* specific)

- I/O Port Access
  - u8 inb(unsigned long port);
  - u16 inw(unsigned long port);
  - u32 inl(unsigned long port);
  - void outb(u8 value, unsigned long port);
  - void outw(u16 value, unsigned long port);
  - void outl(u32 value, unsigned long port);
- Header: <asm/io.h>
- Kernel Window: /proc/ioports
- Access Permissions
  - Header: <linux/ioport.h>
    - struct resource *request_region(resource_size_t start, resource_size_t size, label);
    - void release_region(resource_size_t start, resource_size_t size);

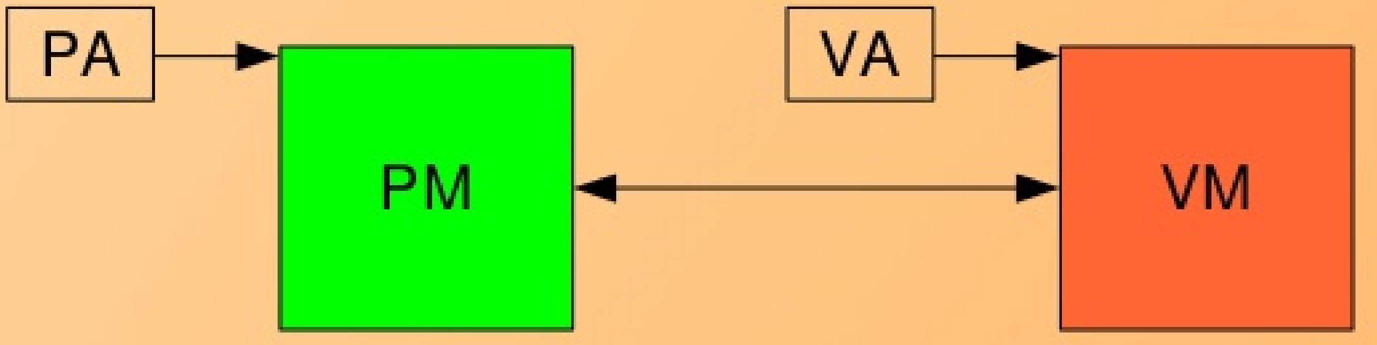
DMA Mapping



To Device
From Device

- APIs
  - dma_addr_t dma_map_single(struct device *, void *, size_t, enum dma_data_direction);
  - void dma_unmap_single(struct device *, dma_addr_t, size_t, enum dma_data_direction);
  - Directions
    - DMA_BIDIRECTIONAL
    - DMA_TO_DEVICE
    - DMA_FROM_DEVICE
    - DMA_NONE
- Header: <linux/dma-mapping.h>

DMA Allocation



- APIs
  - void *dma_alloc_coherent(struct device *, size_t, dma_addr_t *, gfp_t);
  - void dma_free_coherent(struct device *, size_t, void *, dma_addr_t);
  - int dma_set_mask(struct device *, u64 mask);
- Header: <linux/dma-mapping.h>

Barriers

- Heard about Processor Optimization?
- void barrier(void);
  - For surrounding instructions
  - Header: <linux/kernel.h>
- void [r|w|]mb(void);
  - For surrounding read/write instructions
  - Header: <asm/system.h>

Memory & Character Driver

- Dynamic Memory Experiments
  - Preserve latest write in /dev/memory
  - Control the preserve size using ioctl
  - Implement seek

Hardware & Character Driver

- Digital/Analog I/O Control on the Board
- Figure out
  - Operation Relevant Registers
  - Hardware Access Addresses
  - Relevant low-level access APIs to be used
- Driver for I/O access over /dev/io

What all have we learnt?

- Various Address Spaces in Linux
- Role of Memory Manager in Linux
- Accessing the Memory in Kernel Space
- Accessing the Device or Hardware
  - Memory
  - Registers
- Barriers
- Low-level Access in Drivers

Any Queries?