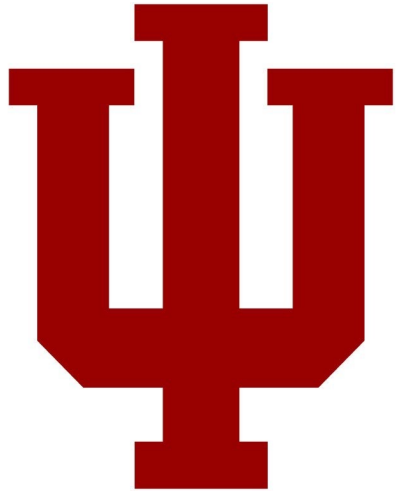
# 09: Memory Translation

Engr 315: Hardware / Software Codesign

Andrew Lukefahr
*Indiana University*



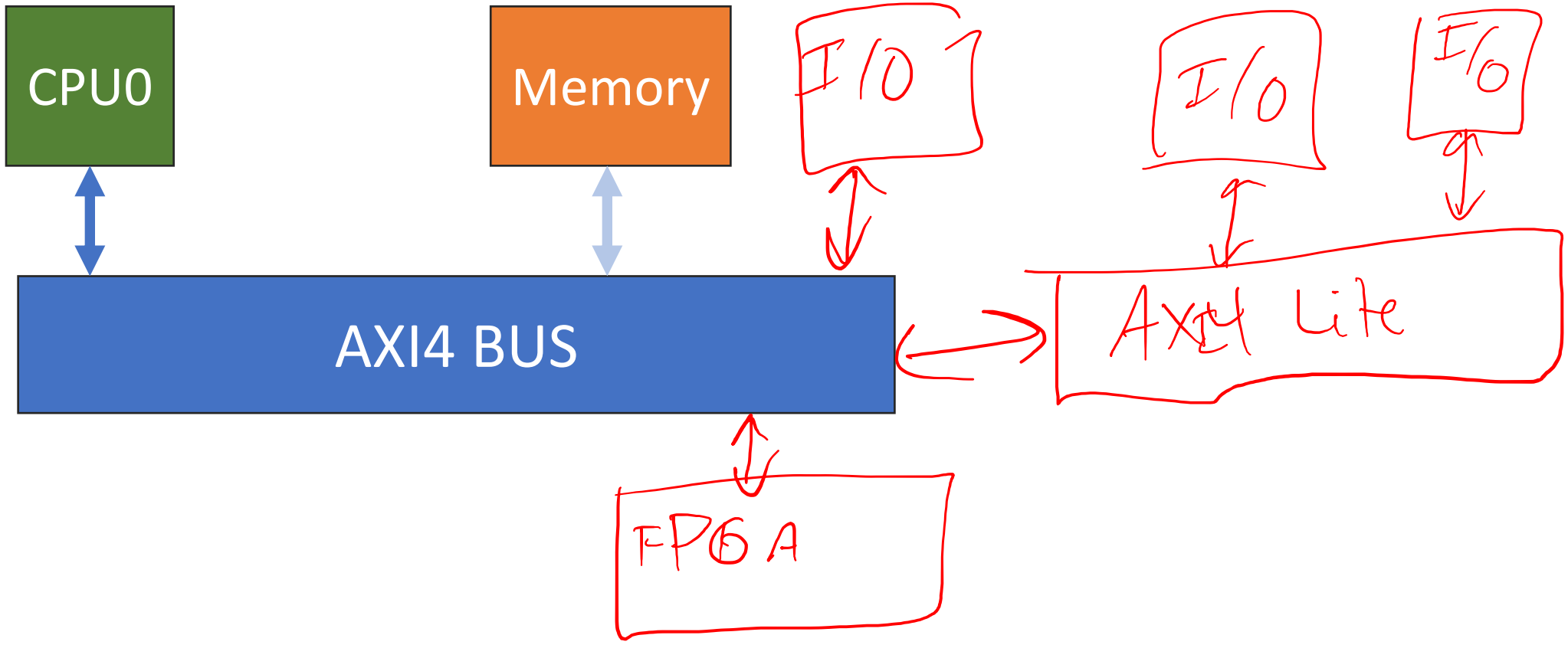
Some material taken from EECS370 at U. of Michigan

# Announcements

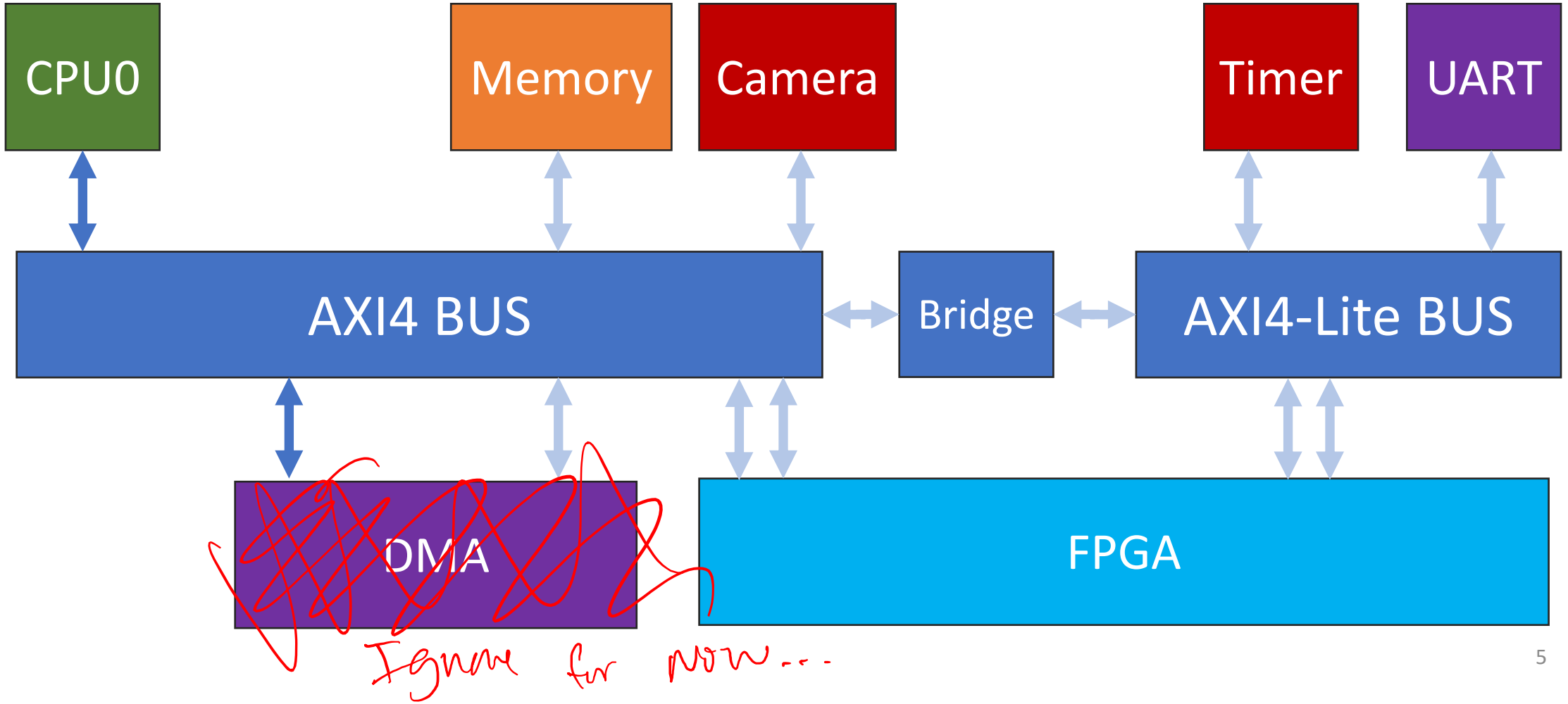
- P3 is out
  - Pushed back to 11:59 Wednesday (2/23)
- P4 is out.
  - Expect some revisions

# Machine Model, Version 0

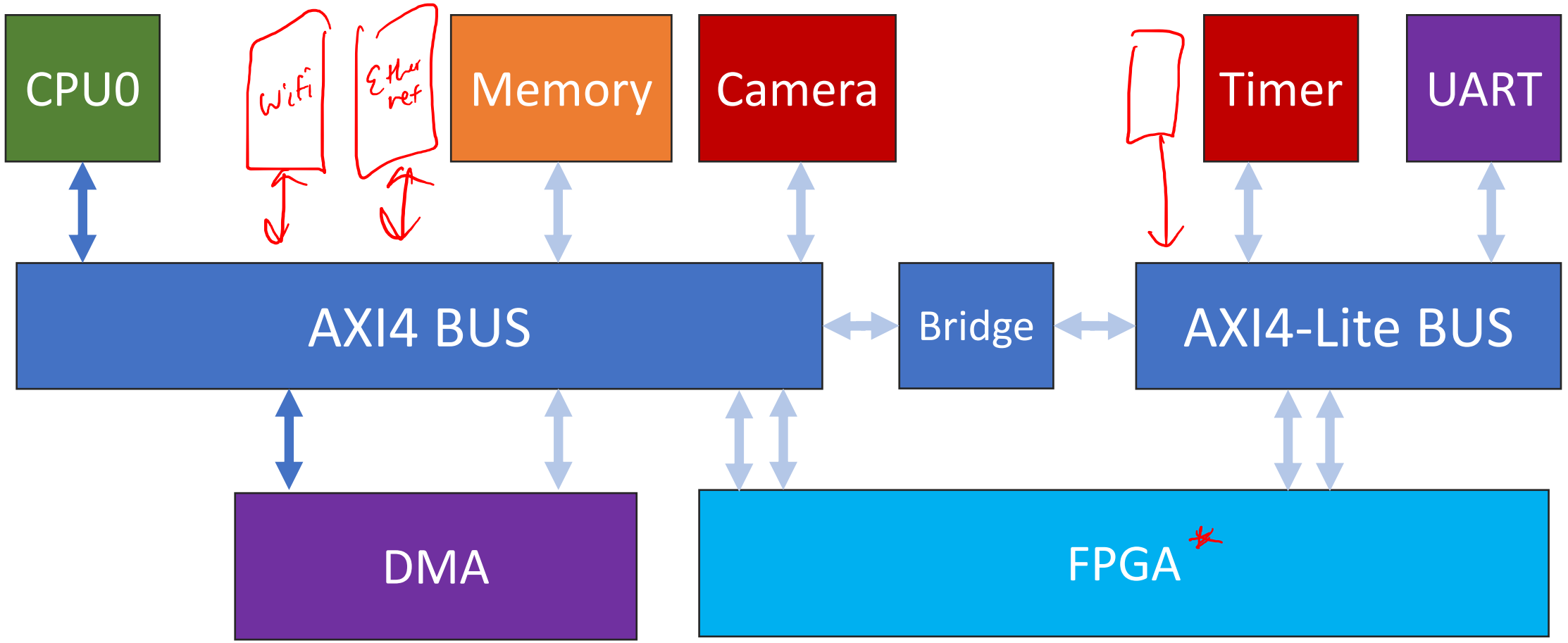

# Machine Model, V1



# Machine Model, V2

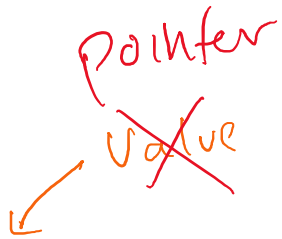


# Machine Model, V2



# MMIO from C.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#define EMA_MMIO 0x40000000
int main () {
    volatile uint32_t * ema_ptr = (uint32_t*)(EMA_MMIO);
    int32_t val = 0x1000;
    while (1) {
        //push new value into EMA
        *ema_ptr = val;
        //load value from EMA
        val = *ema_ptr;
        printf("Val: %d\n", val);
    }
    return 0;
}
```



# MMIO from C.

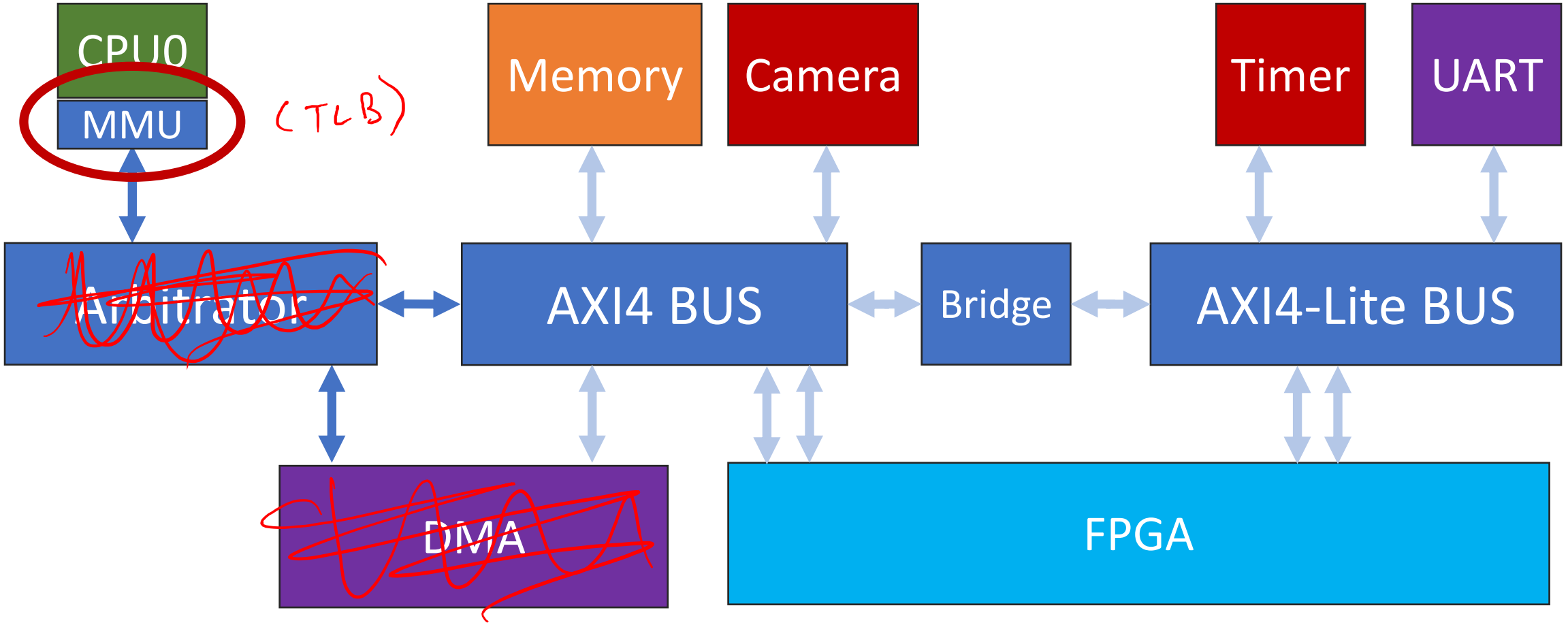
```
#define EMA_MMIO 0x40000000
volatile uint32_t * ema_ptr = (uint32_t*)(EMA_MMIO);
//push new value into EMA
        *ema_ptr = val;
        //load value from EMA
        val = *ema_ptr;
```

```
   volatile uint32_t * ema_ptr = (uint32_t*)(EMA_MMIO);
8224:   e3a05101    mov r5, #1073741824 ; 0x40000000
   *ema_ptr = val;
822c:   e5854000    str r4, [r5]
    val = *ema_ptr;
8230:   e5954000    ldr r4, [r5]
```

# MMIO from C

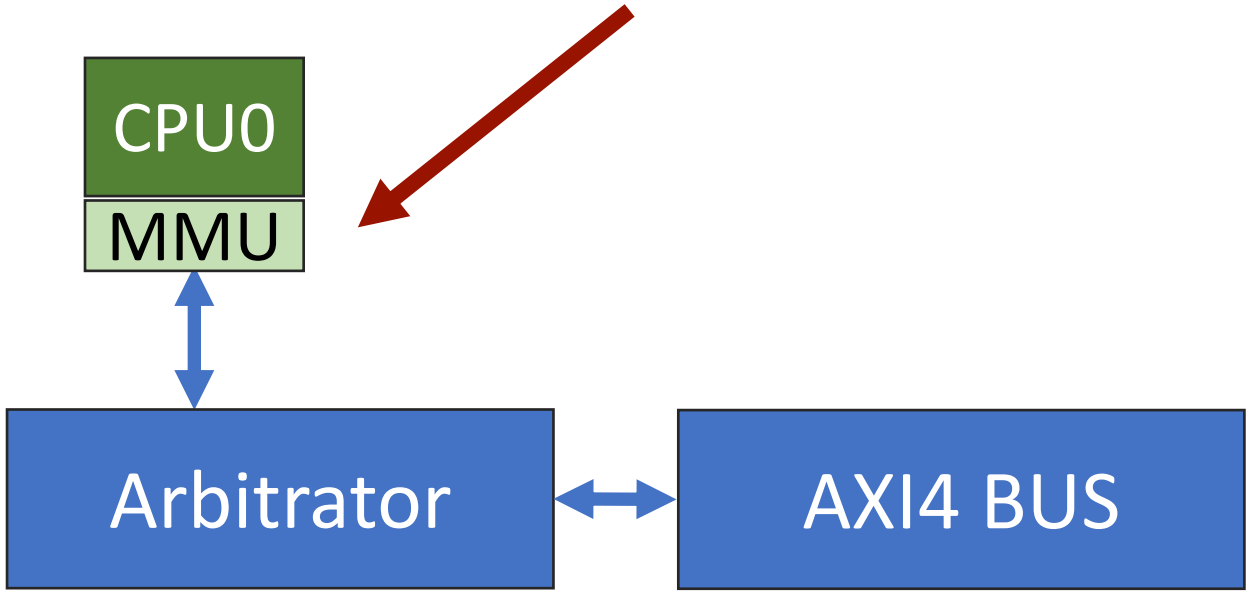
- DON’T WORK!
- Why?
- Linux… and MMIOs

# Machine Model, V3: MMUs

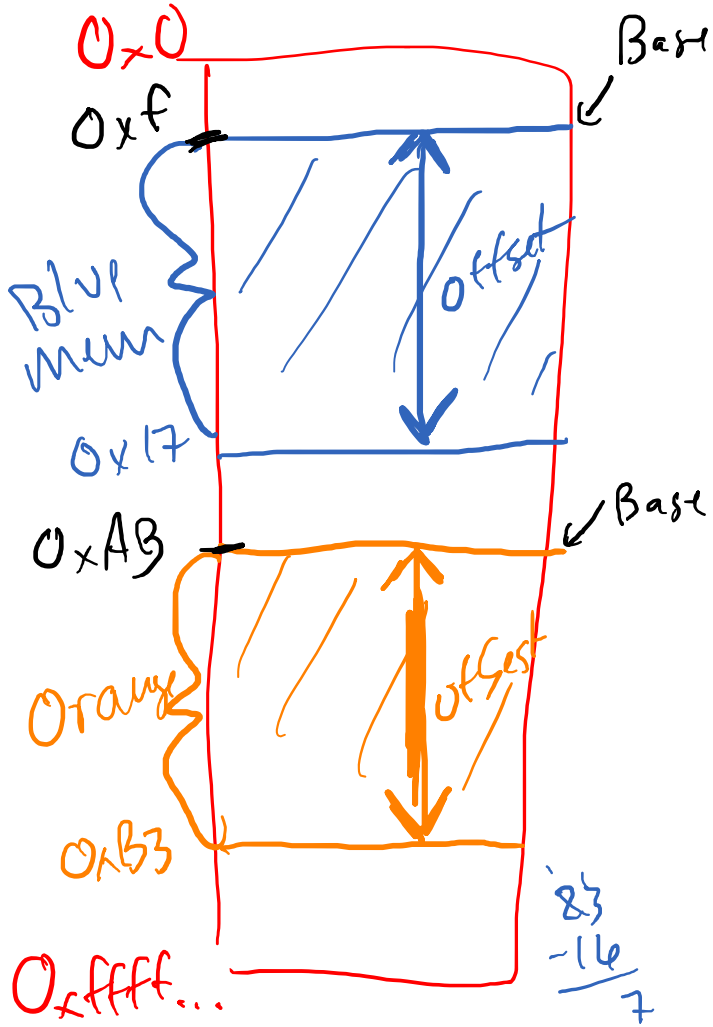


# MMU: Memory Management Unit (TLB)

- Rejects load + stores that are “unauthorized”
- Translates addresses (Later)



# MMUs track the following things




- **BASE ADDRESS**: the start of a memory region that is allowed through the MMU

  Blue: 0xf

  Or: 0xAB

- **OFFSET**: the size of a memory region that is allowed through the MMU

  8 bytes

  Blue: 0xf + 0x8 = 0x17 offset = 0x8

  ORange: offset = 8 => 0xAB ⊖ B3

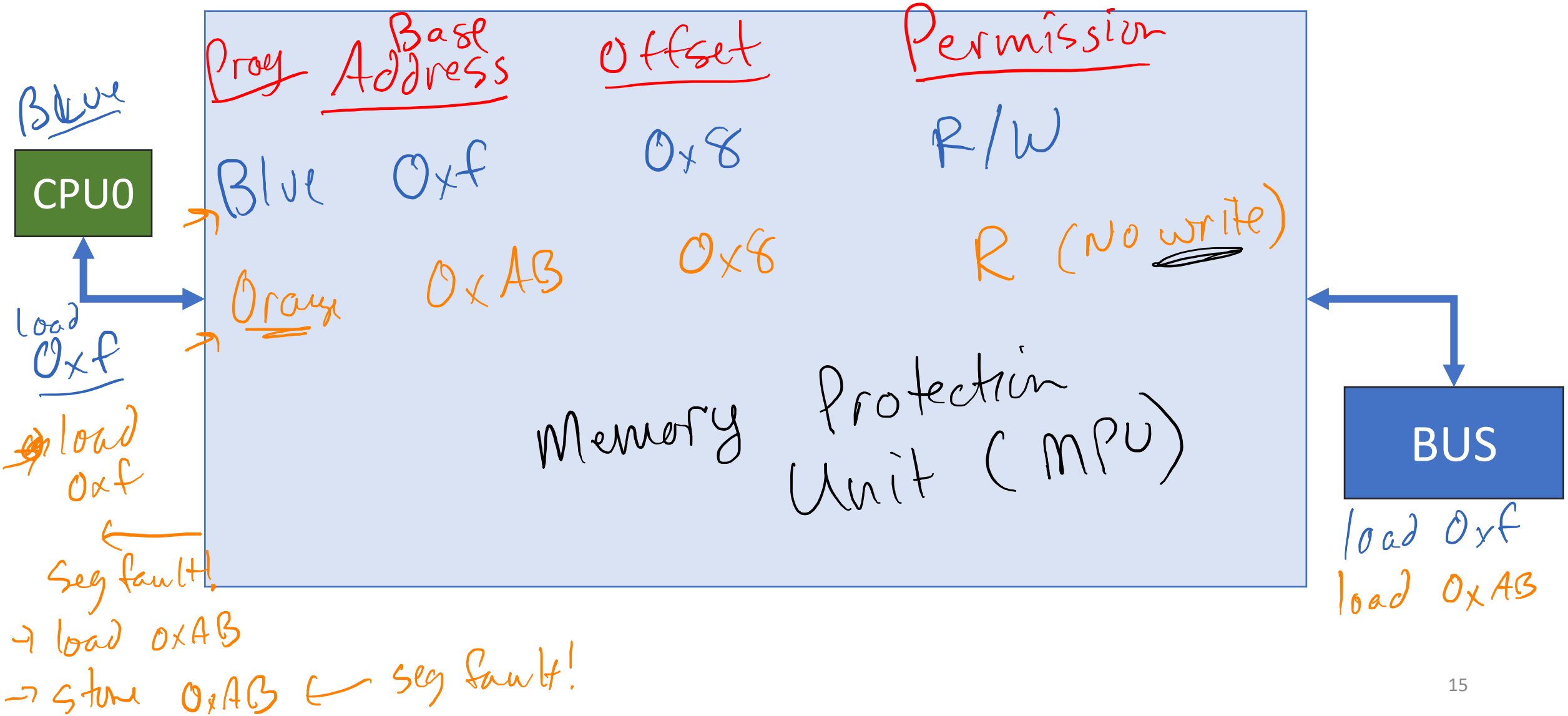
- **Permission**: the type of access that is allowed through the MMU

  Read / write

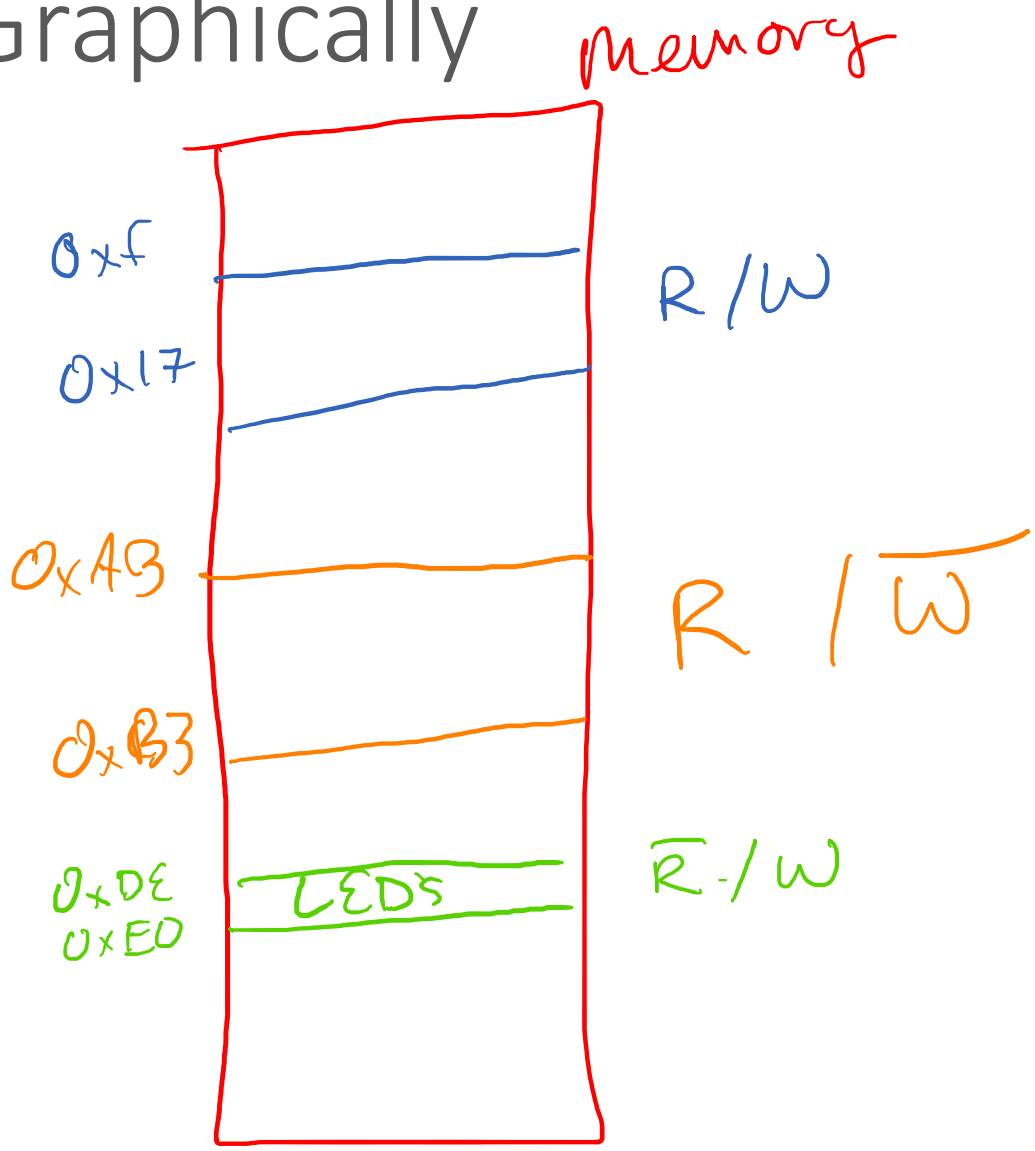
  Blue: R/W for Blue None for Orange

  Orange: None for Blue R/W for orange

# Basic MMU Table



# Memory Protections Graphically



# Why?

- Security
  - Keep you from modifying the code
  - Keep you from executing the data
- Separate multiple applications

# Separating multiple applications

- A) What if two applications want to use the same memory address?
- B) How do I prevent your application from modifying my memory?

# Two application test..

```
#include <stdio.h>
#include <stdlib.h>

volatile int avalue = 2;

int main ()
{

    while (avalue == 2) { ; }
    return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>

volatile int avalue = 3;

int main ()
{

    while (avalue == 3) { ; }
    return 0;
}
```

# Two application test..

```
gcc -g -O0 test_1.c -o test_1.out
objdump -DSs test_1.out > test_1.dis
gcc -g -O0 test_2.c -o test_2.out
objdump -DSs test_2.out > test_2.dis

vi test_1.dis test_2.dis
```

# Two application test..

```
00011008 <avalue>:
volatile int avalue = 2;
   11008:       00000002        andeq   r0, r0, r2
```

FF11008

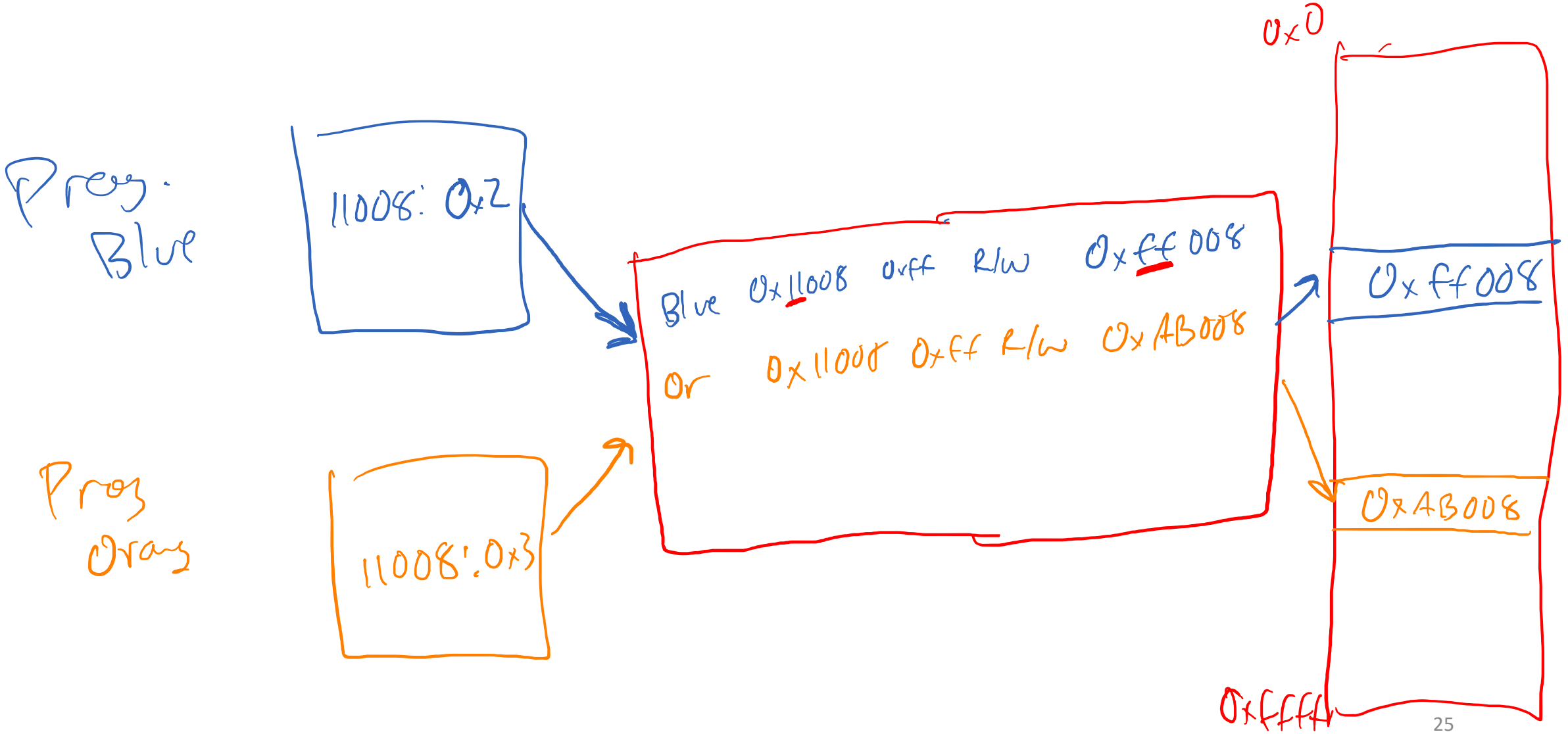
```
00011008 <avalue>:
volatile int avalue = 3;
   11008:       00000003        andeq   r0, r0, r3
```

AA11008

# Two application test..

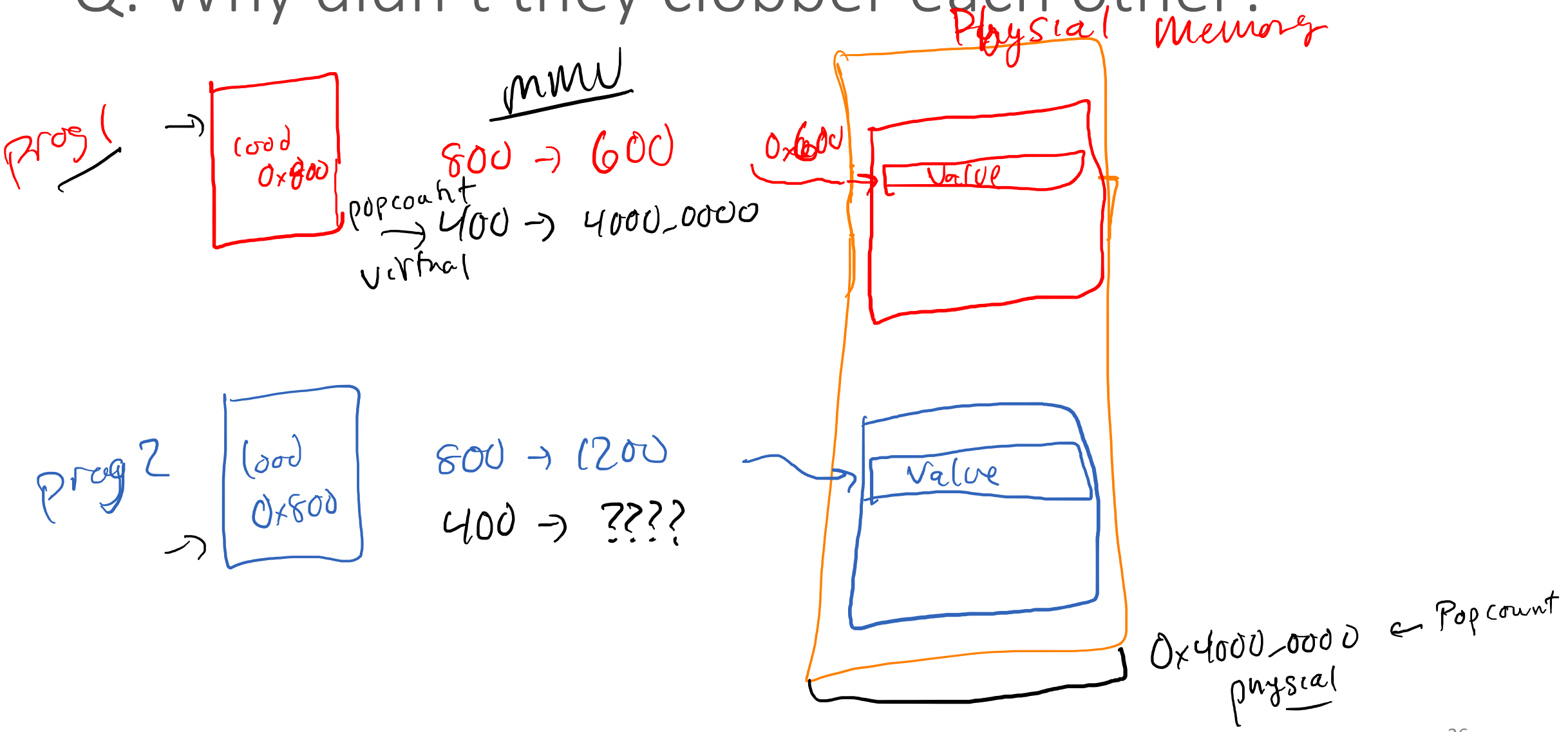
```
./test_1.out &
./test_2.out &

top
```

# Q: Why didn't they clobber each other?



Swamy: Mentioned 201

# Q: Why didn't they clobber each other?



# Q: Why didn’t they clobber each other?

- A:  MMUs are doing something else… “Virtual Memory”

# Virtual memory with an MMU

- MMU automatically translates each memory reference from a

(CPU) virtual address

(which the programmer sees as a huge array of bytes)

to a

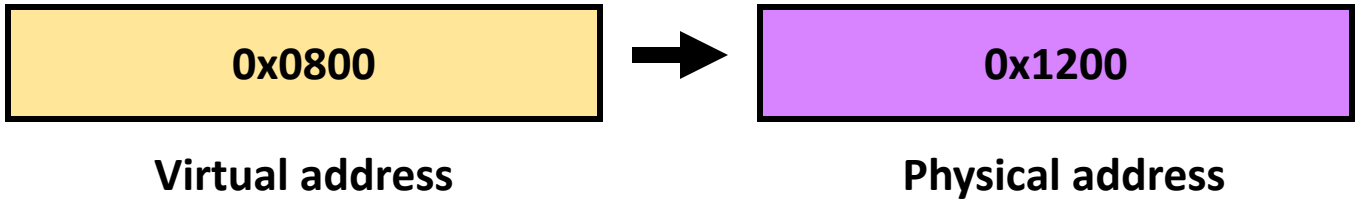
(Memory) physical address

(which the hardware uses to identify where the storage actually resides)

# Basics of Virtual Memory

- Any time you see the word virtual in computer science and architecture it means "using a level of indirection"

- Virtual memory hardware changes the virtual address the programmer sees into the physical one the memory chips see



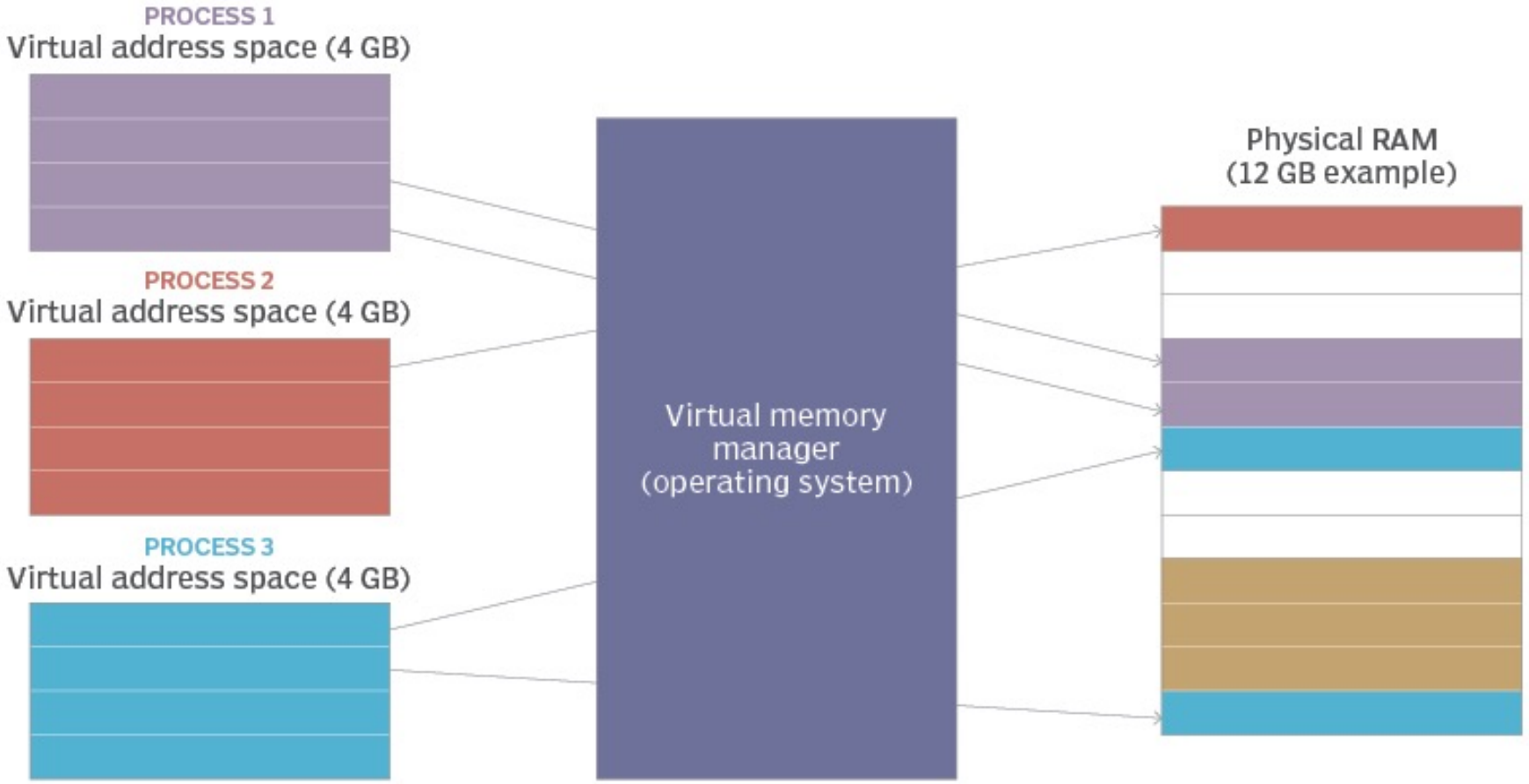
# Virtual Memory View

- Virtual memory lets the programmer address a memory array larger than the DRAM available on a particular computer system
- Virtual memory enables multiple programs to share the physical memory without:
  - Knowing other programs exist (**transparency**)
  - Worrying about one program modifying the data contents of another (**protection**)

# Managing virtual memory

- Managed by hardware logic ***and*** operating system software
  - Hardware for speed
  - Software for flexibility and because disk storage is controlled by the operating system
- The hardware must be designed to support Virtual Memory

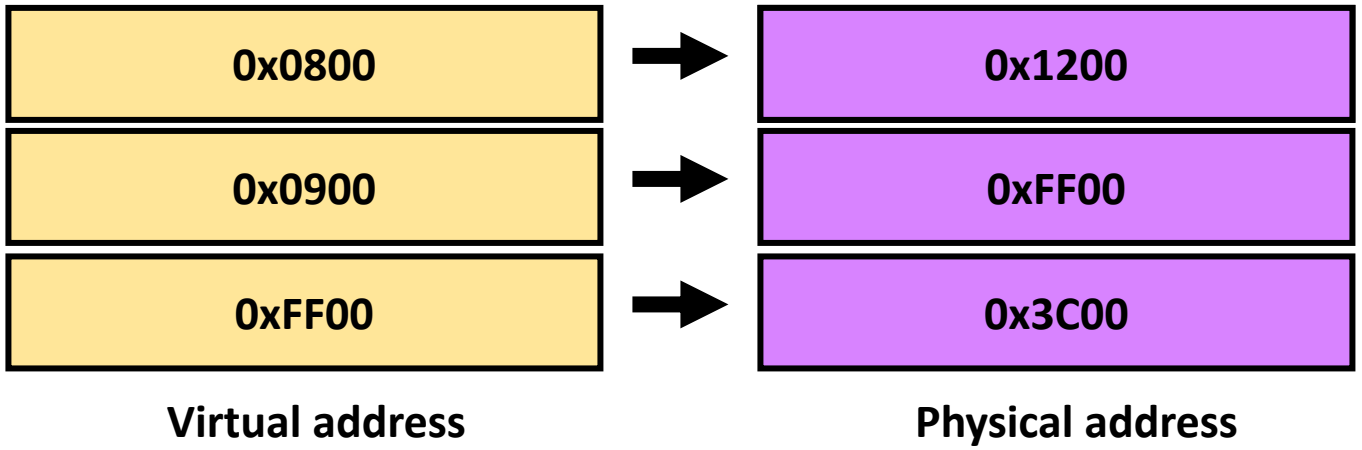
# OS (Linux) mains full Virtual->Physical Mappings



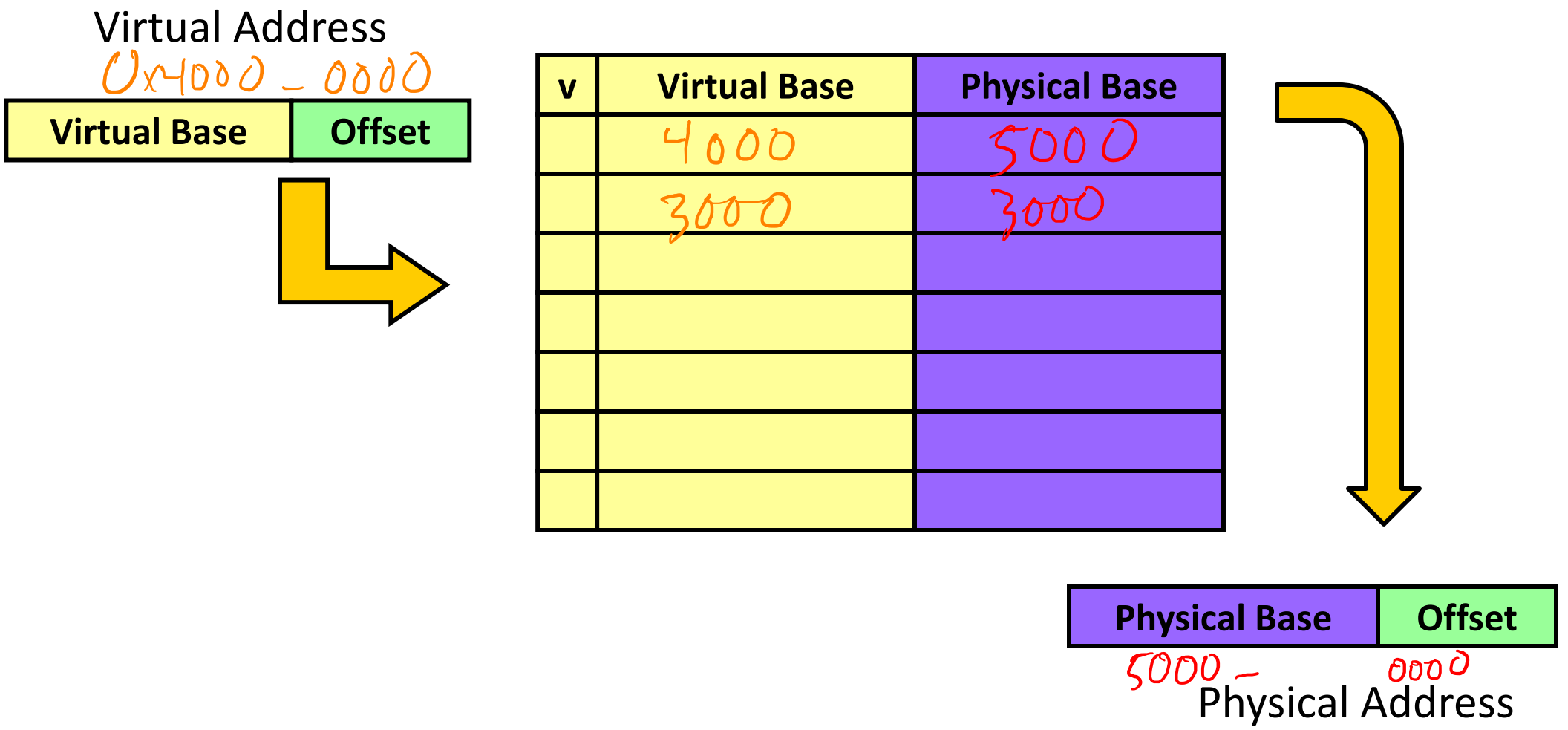

https://www.techtarget.com/searchstorage/definition/virtual-memory

# Hardware uses TLBs (Translation Look-aside Buffers)

- Buffer common Virtual->Physical translations in a **Translation Look-aside Buffer (TLB)**, a fast cache memory dedicated to storing a small subset of valid translations
- 16-512 entries common
- Generally has low miss rate (< 1%)



# MMU Address Translation



# MMU Address Translation


unity mapping

Virtual Address (32-bit)

| Virtual Base | Offset |
|---|---|

0x69ff | 0004

2000 | 0008

| v | Virtual Base | Physical Base |
|---|---|---|
| | 69ff | 4000 |
| | 3f00 | 4f00 |
| | 2000 | 2000 |
| | | |
| | 6000 | 8000 |
| | 7000 | 8000 |
| | | |

0x4000 | 0004

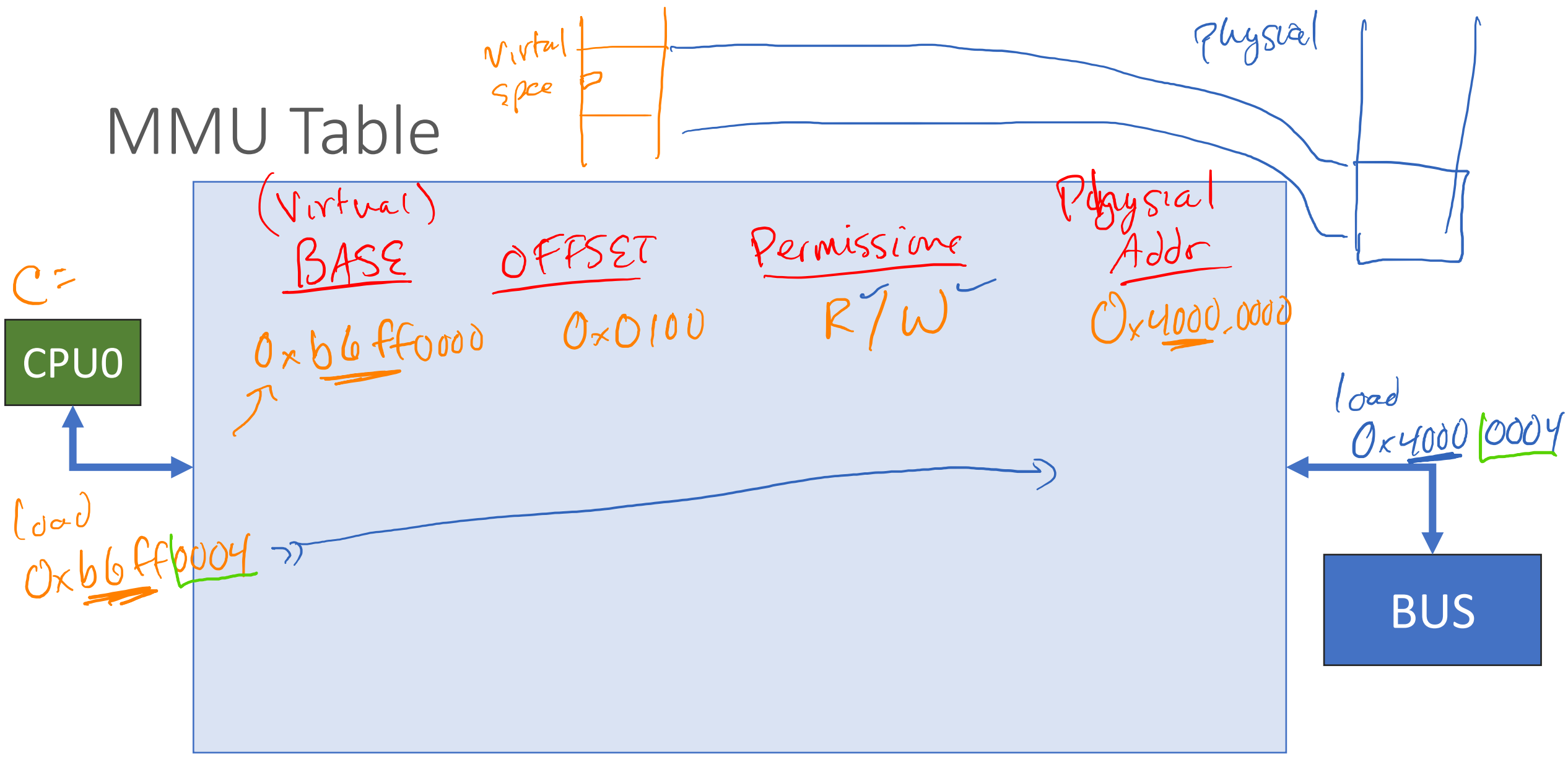
2000 | 0008

| Physical Base | Offset |
|---|---|

Physical Address

memory (mmio)

# MMU Table


virtual space
physical
(Virtual)
BASE
OFFSET
Permissions
Physical Addr
0x4000.0000
R/W
0x0100
0xb6ff0000
C:=
CPU0
load
0xb6ff0004
load
0x4000 0004
BUS

# So how do we access the FPGA for P4?

- FPGA uses physical address
- CPU (w/Linux) uses virtual address

- Q: How do I talk to a physical address with Linux?

- A: Linux provides a special /dev/mem file to help us!

# Next Time:

- /dev/mem
- /dev/uio

# 09: Memory Translation

Engr 315:  Hardware / Software Codesign

Andrew Lukefahr
*Indiana University*