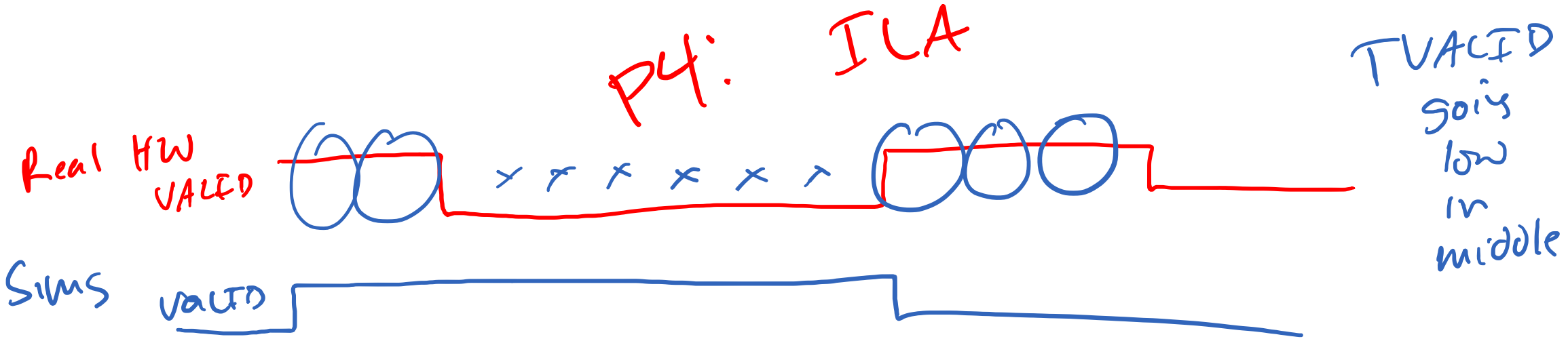


11: Memory Translation

Engr 315: Hardware / Software Codesign

Andrew Lukefahr
Indiana University

Some material taken from:
EECS 373 & EECS 370 University of Michigan
https://developer.arm.com/documentation/102202/0300/Transfer-behavior-and-transaction-ordering

Announcements

- P4: Due Wednesday.
  demo Friday
- P5: Out soon.
  ↳ Matteo's new AG

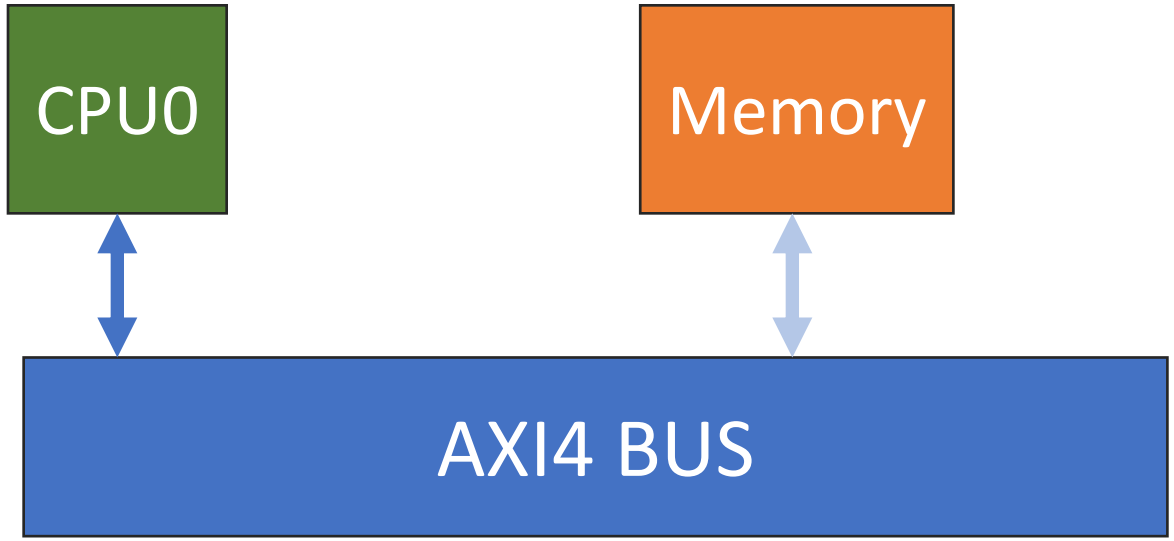
DS tip

Use `volatile` for MMIO addresses!

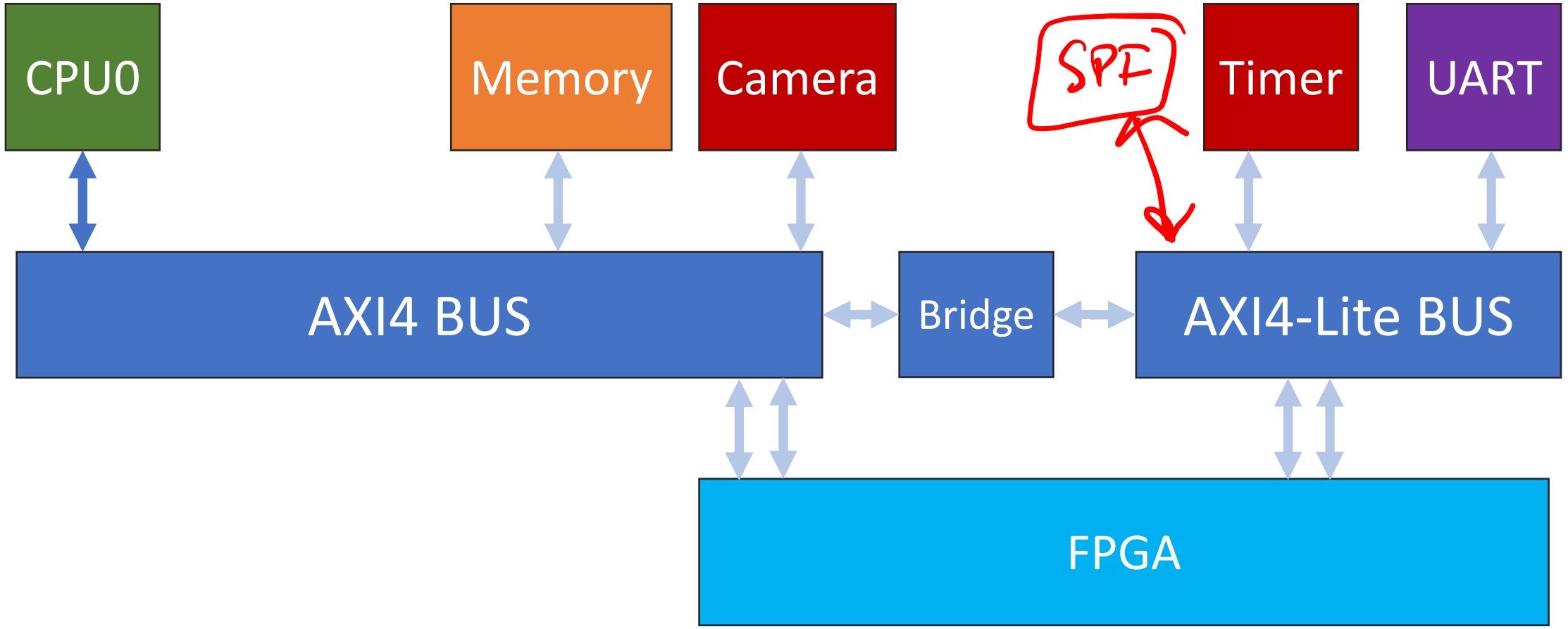
```
#define SW_ADDR 0xfffe
volatile uint32_t * SW_REG = (uint32_t * SW_ADDR);

int quit = (*SW_REG);
while(!quit)
{
     //more code
     quit = (*SW_REG);
}
```

Machine Model, Version 0

Machine Model, V1



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 = (volatile 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.

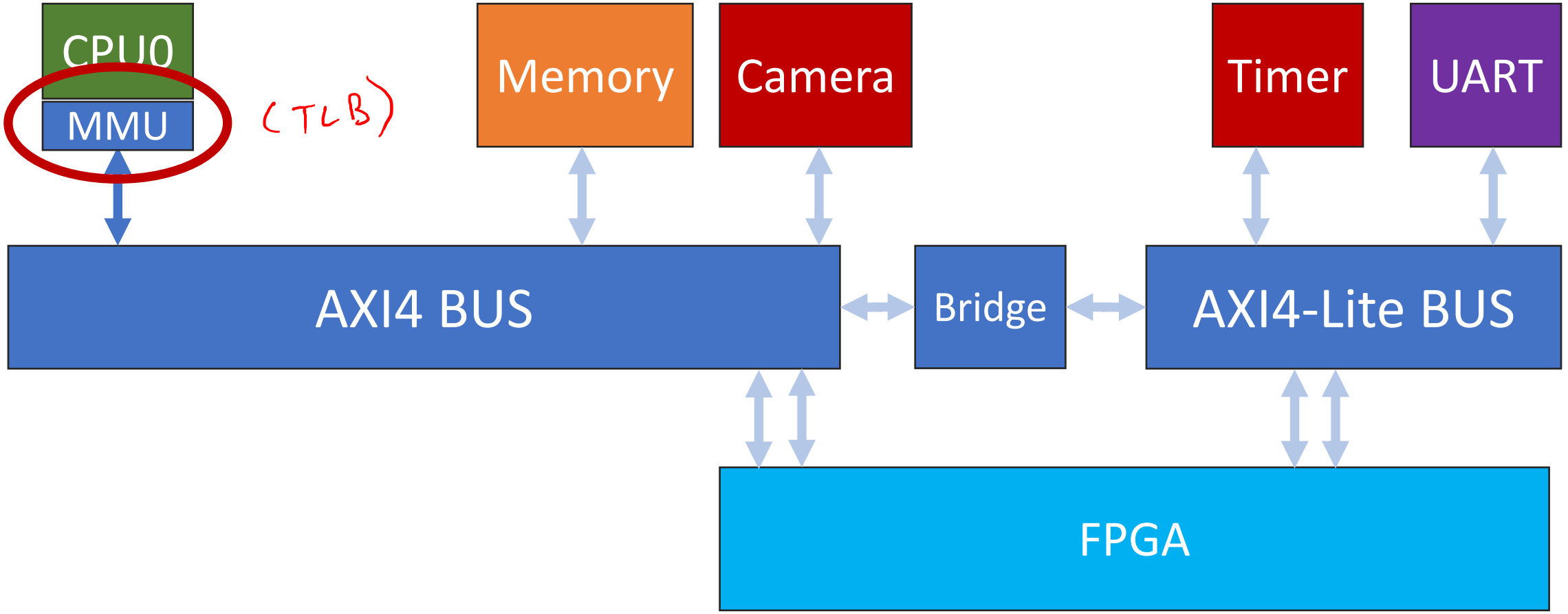
```
#define EMA_MMIO 0x40000000
volatile uint32_t * ema_ptr = (volatile uint32_t*)(EMA_MMIO);
//push new value into EMA
        *ema_ptr = val;
        //load value from EMA
        val = *ema_ptr;
```

```
   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

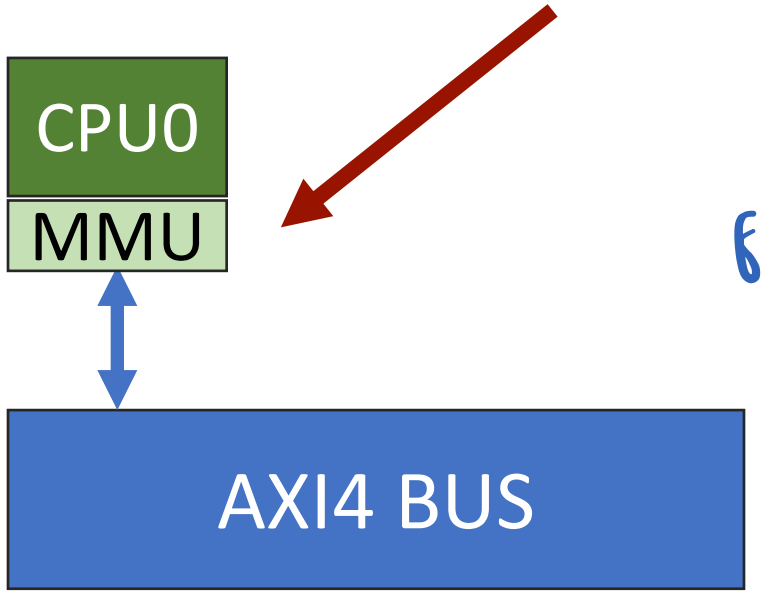
- WONT WORK!
- Why?
- Linux… and MMIOs

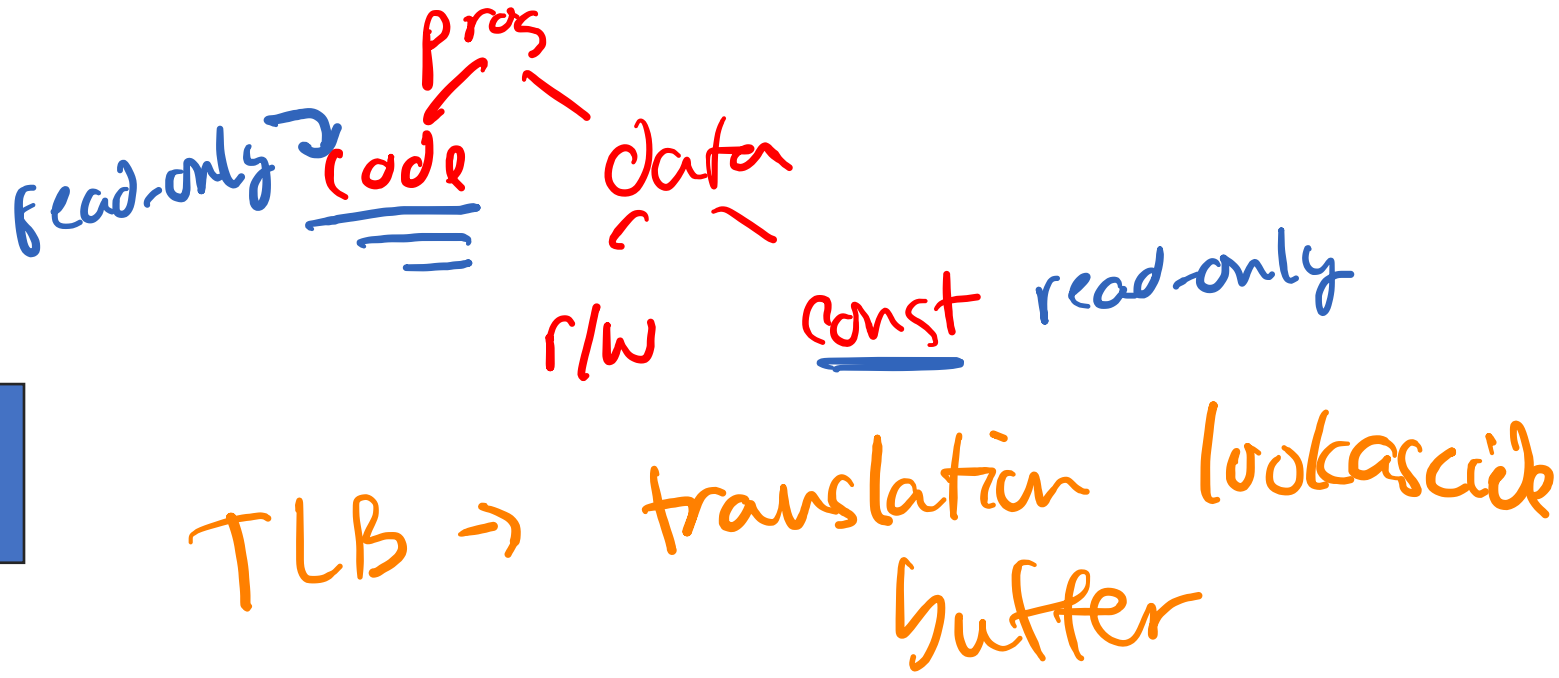
Machine Model, V3: MMUs



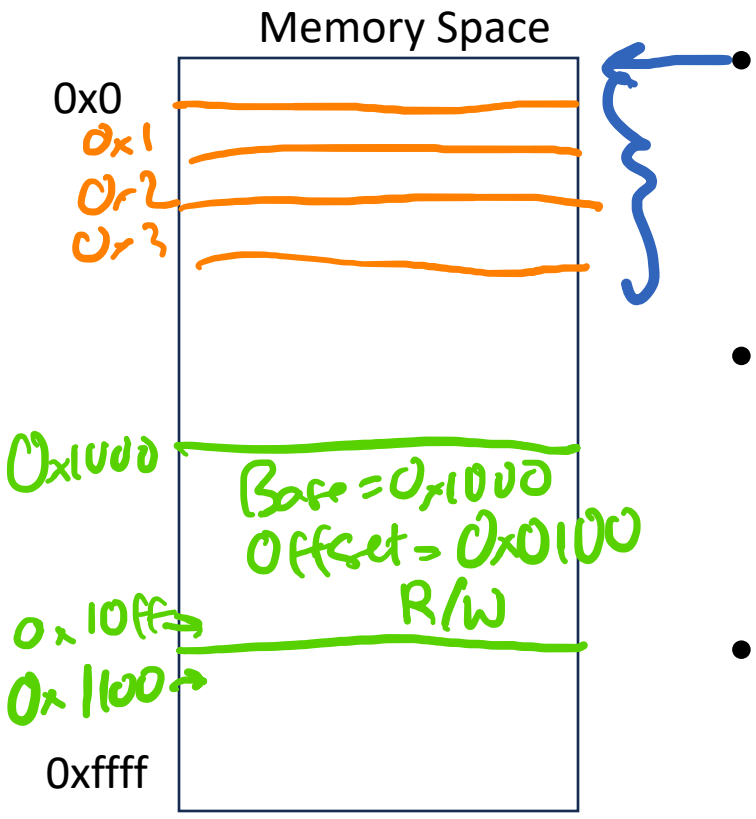
MMU: Memory Management Unit (TLB)

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



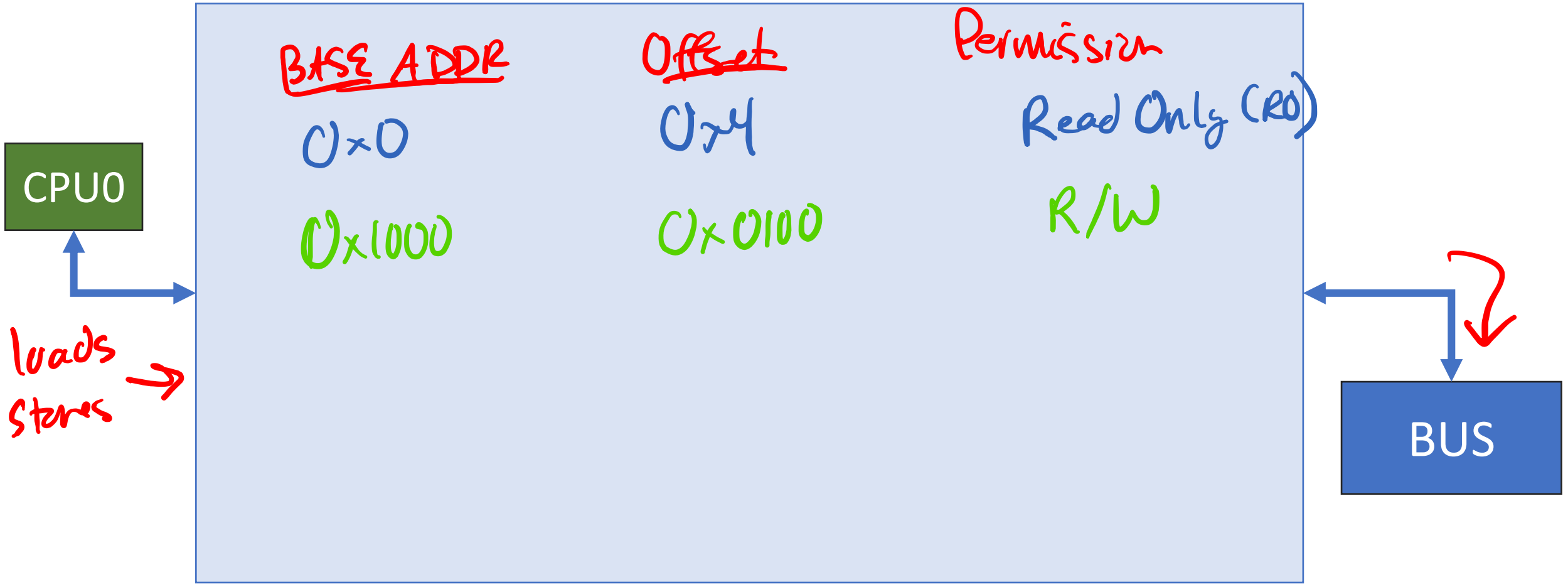


MMUs track the following things



- **BASE ADDRESS**: the start of a memory region that is allowed through the MMU
- **OFFSET**: the size of a memory region that is allowed through the MMU
- **Permission**: the type of access that is allowed through the MMU
  - Write Only (WO)
  - Read/Write (R/W)

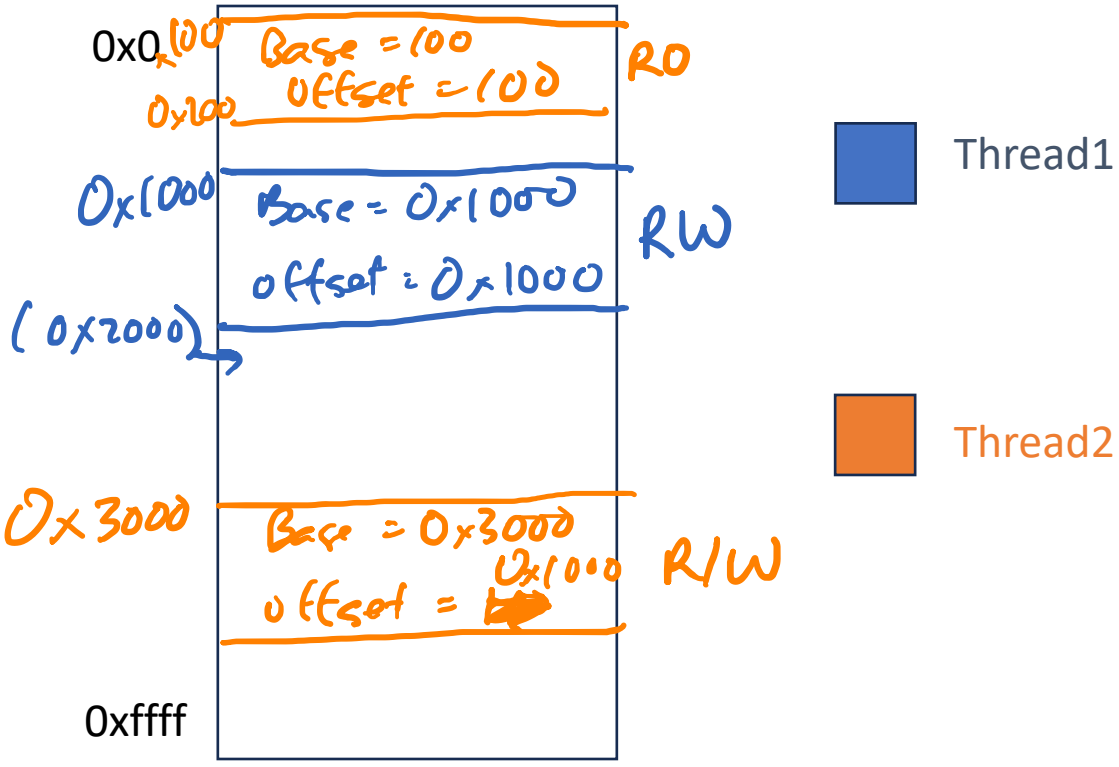
Basic MMU Table



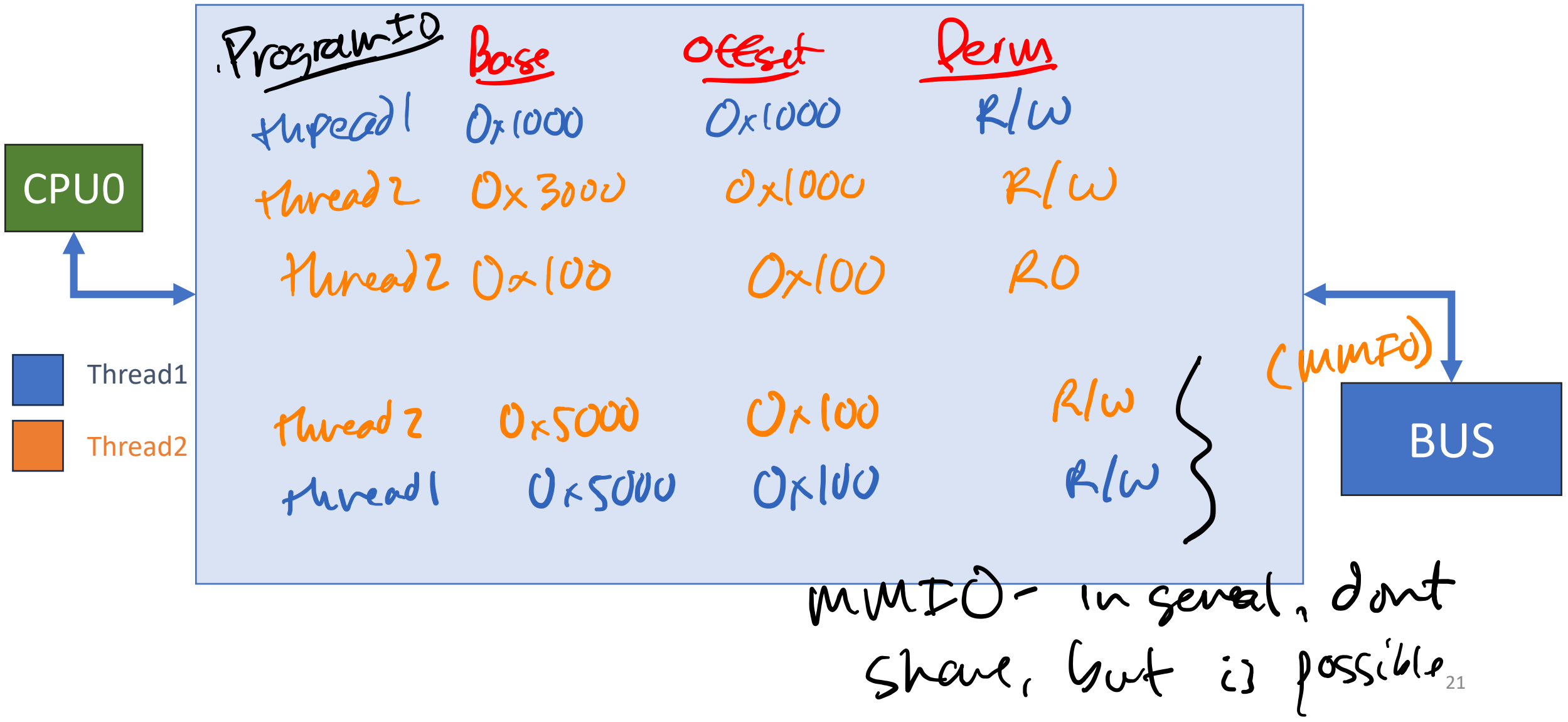
Memory Protection

- CPUs run multiple applications
- Q: How do I prevent one application from modifying another's memory?
- A: Add program-specific access rules to the MMU
- Often called a Memory Protection Unit (MPU)

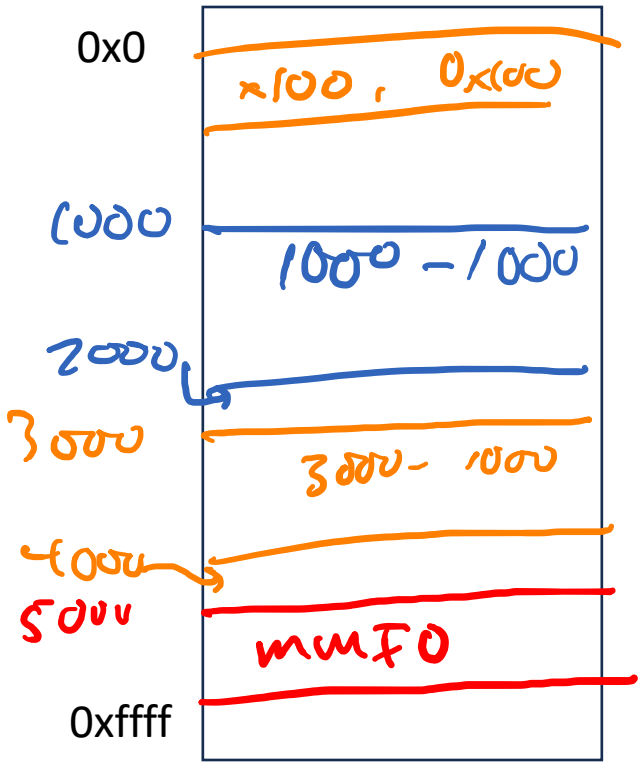
Memory Protections Regions



Memory-Protection Table



MMIO with Memory Protections Regions



Thread1

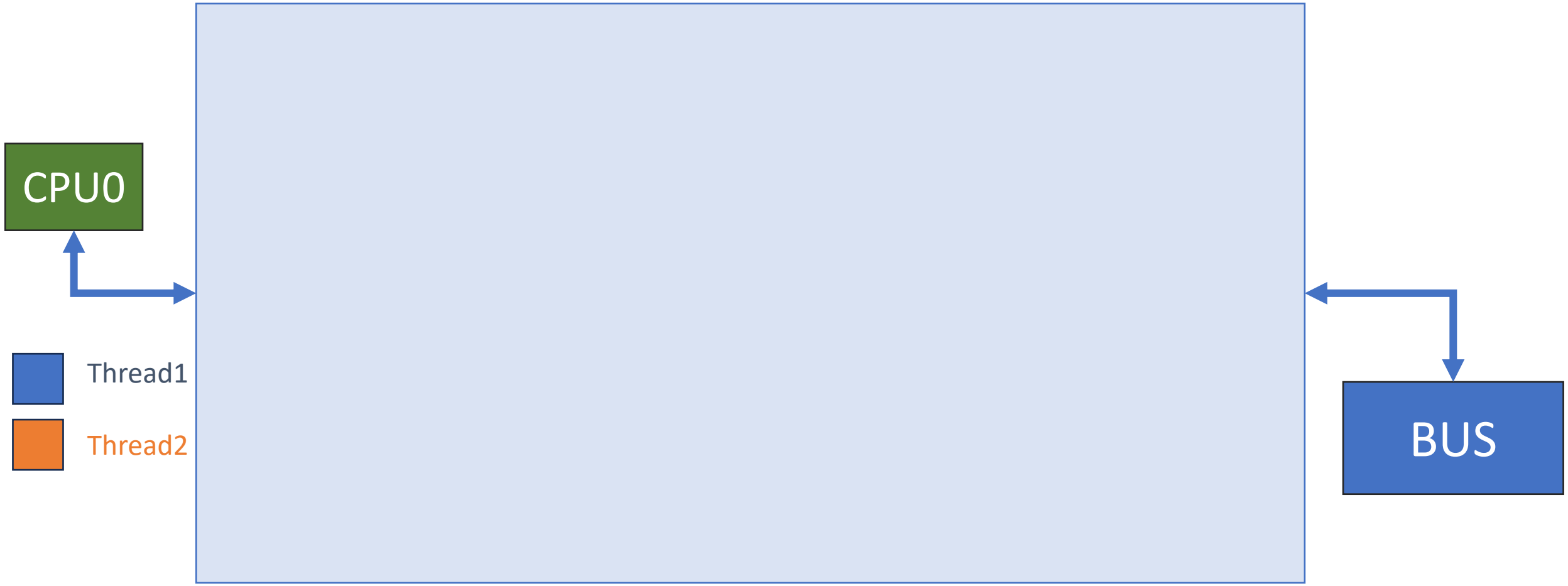
Thread2

MMIO Options

1) fine-grained mapping

2) Overlap

Memory-Protection Table



Why?

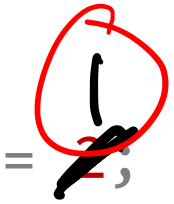
- Security
  - Keep you from modifying the code
  - Keep you from executing the data
- Separate multiple applications

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 = 2;

int main ()
{

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

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

Two application test..

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

top
```

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

virtual address

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

to a

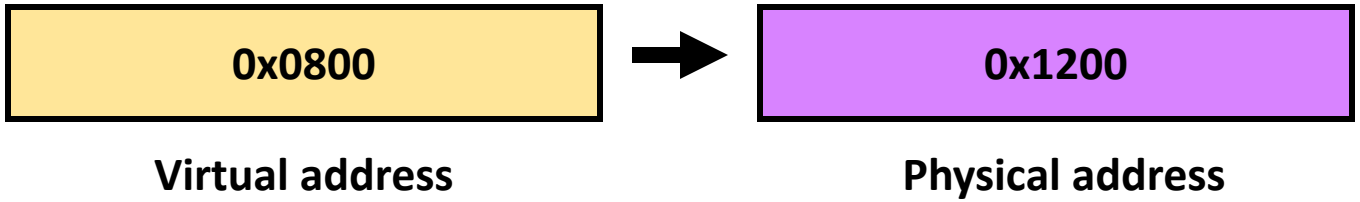
physical address

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

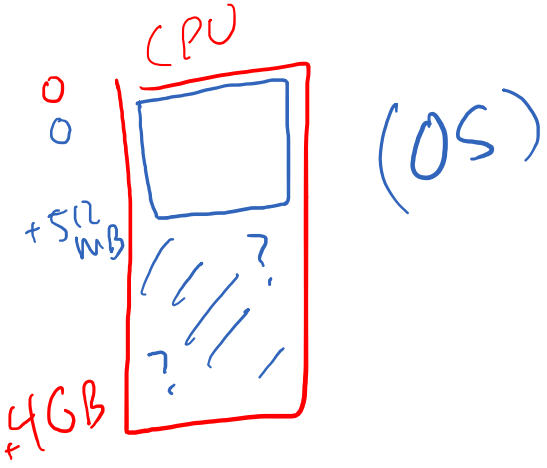
Basics of Virtual Memory

- 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

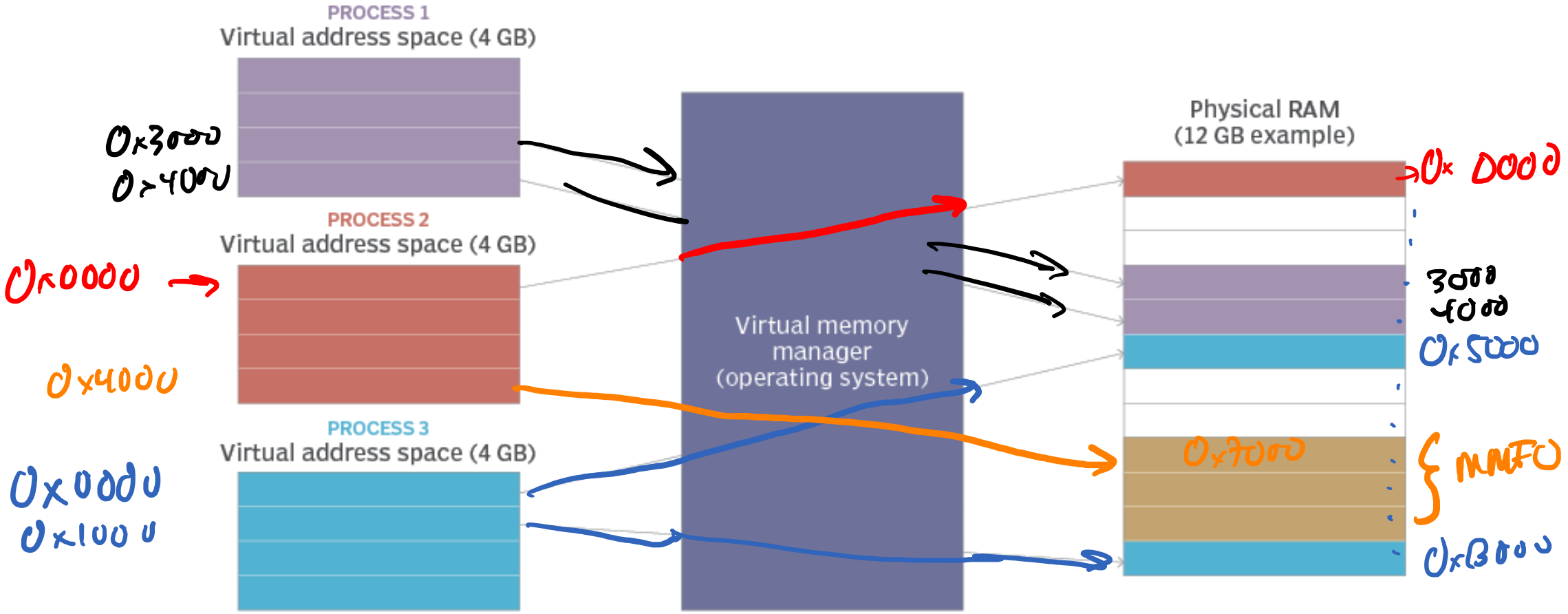
- Virtual memory enables multiple programs to share the physical memory without:
  - Knowing other programs exist (**transparency**)
  - Worrying about one program modifying the data contents of another (**protection**)

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

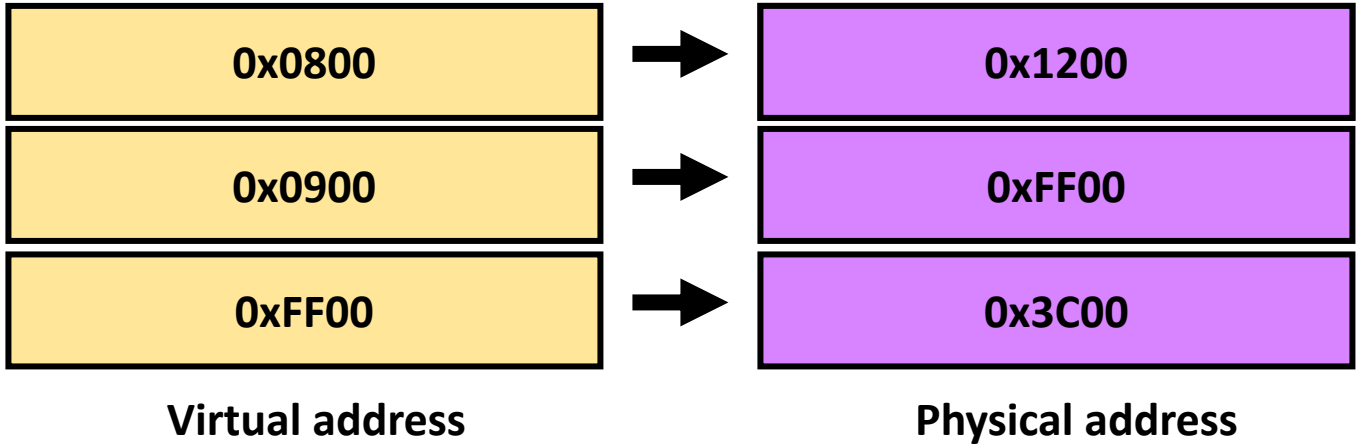
1) Need mapping



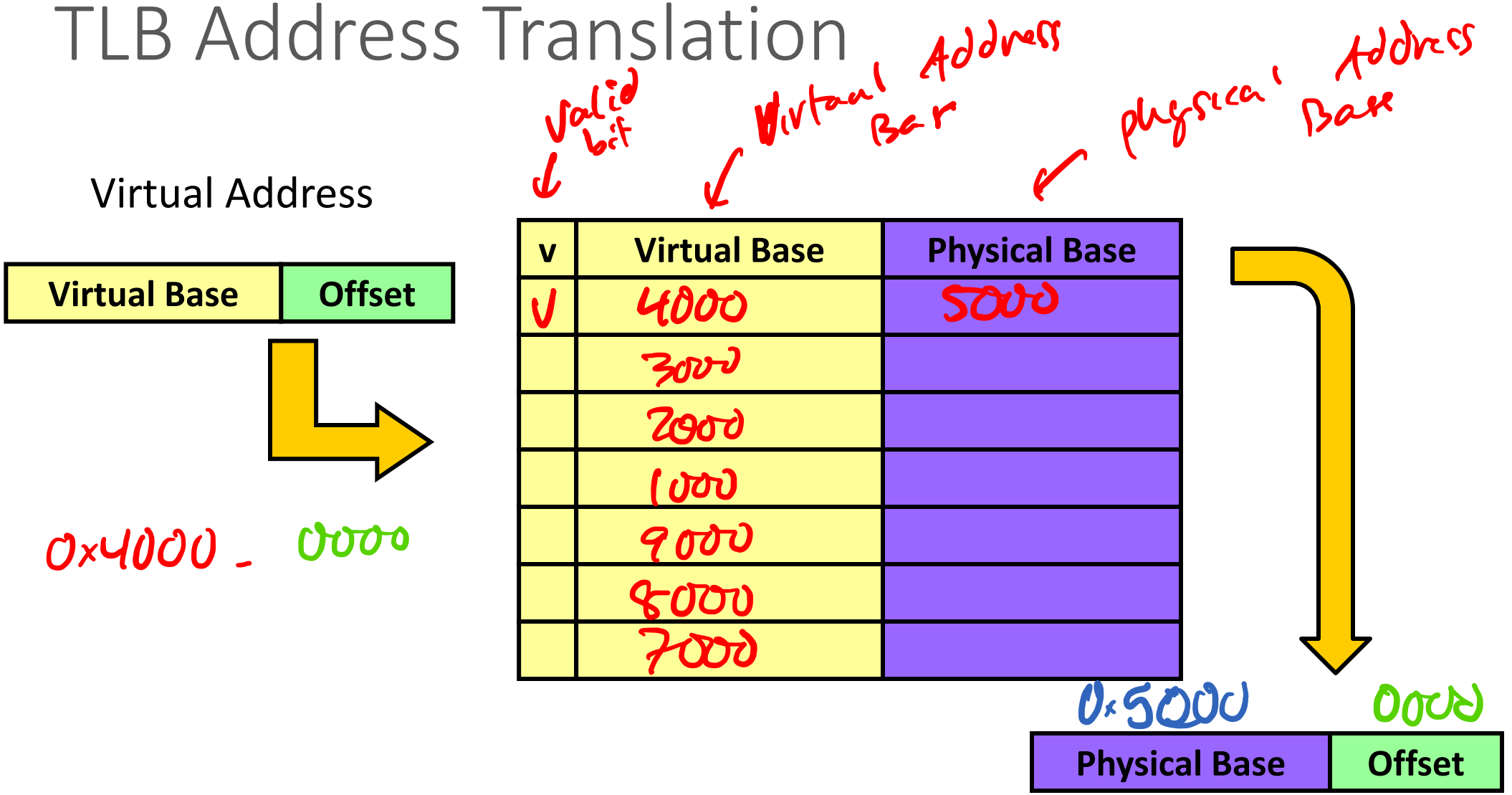
https://www.techtarget.com/searchstorage/definition/virtual-memory

Hardware uses TLBs (Translation Look-aside Buffers)

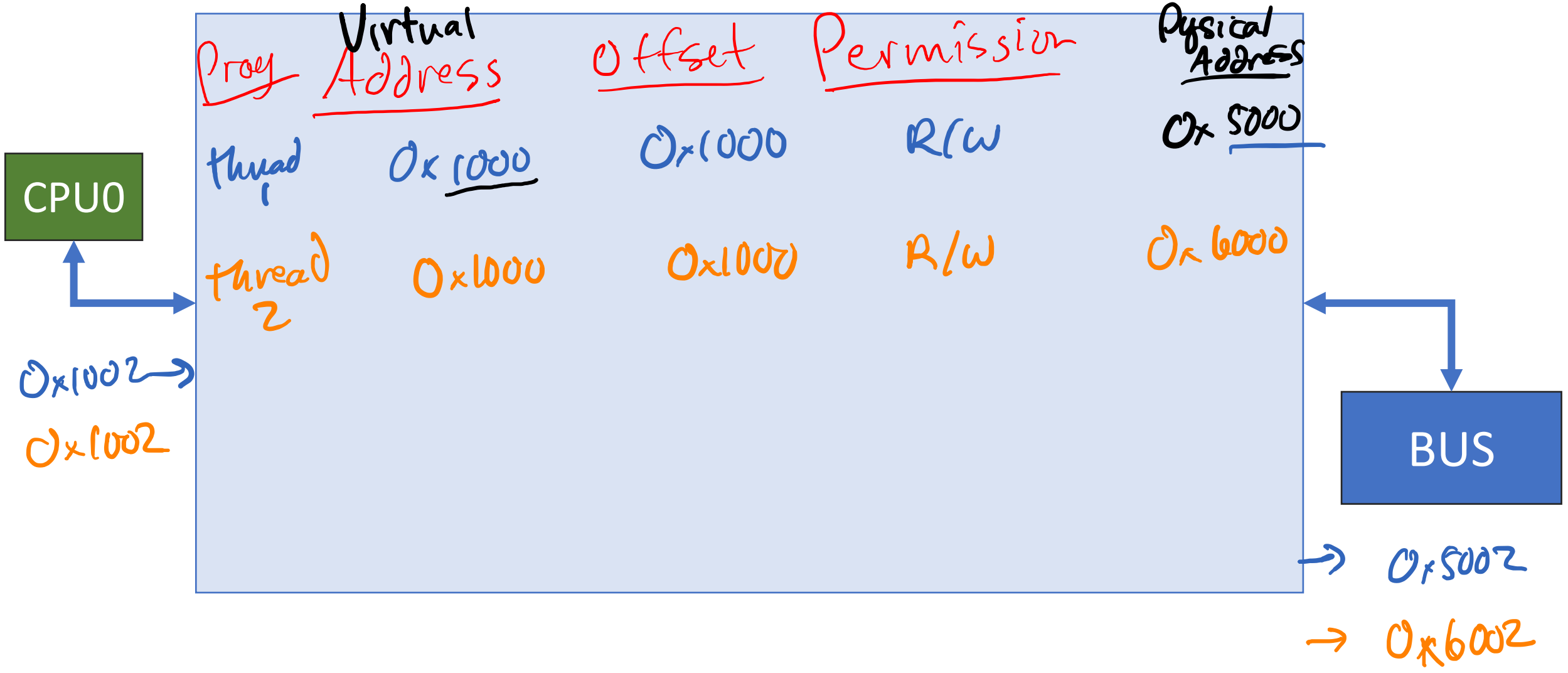
- Buffer common Virtual->Physical translations in a **Translation Look-aside Buffer (TLB)**, a fast cache memory dedicated to storing a small subset of valid translations
- 16-512 entries common
- Generally has low miss rate (< 1%)



TLB Address Translation



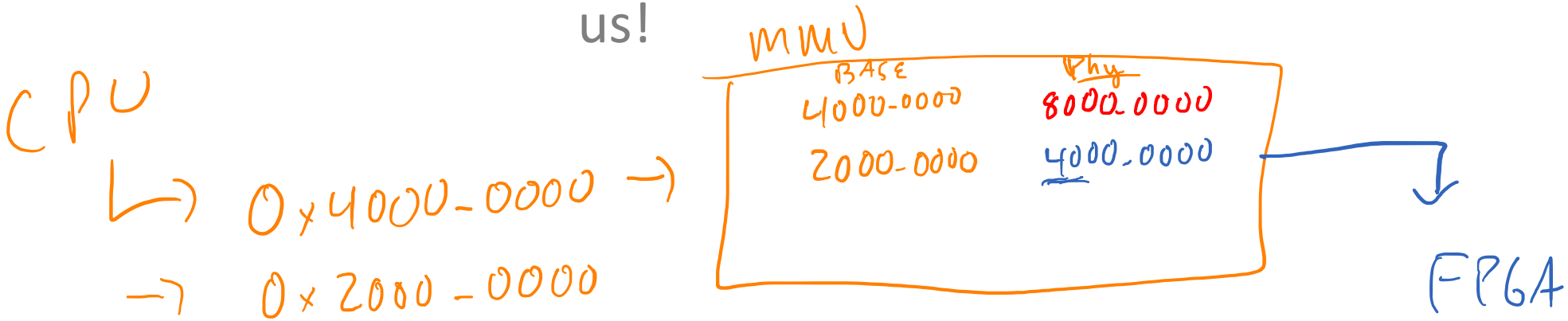
Adding TLB mappings in the MMU



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!

0x4000-0000



P4: MMIO Popcount

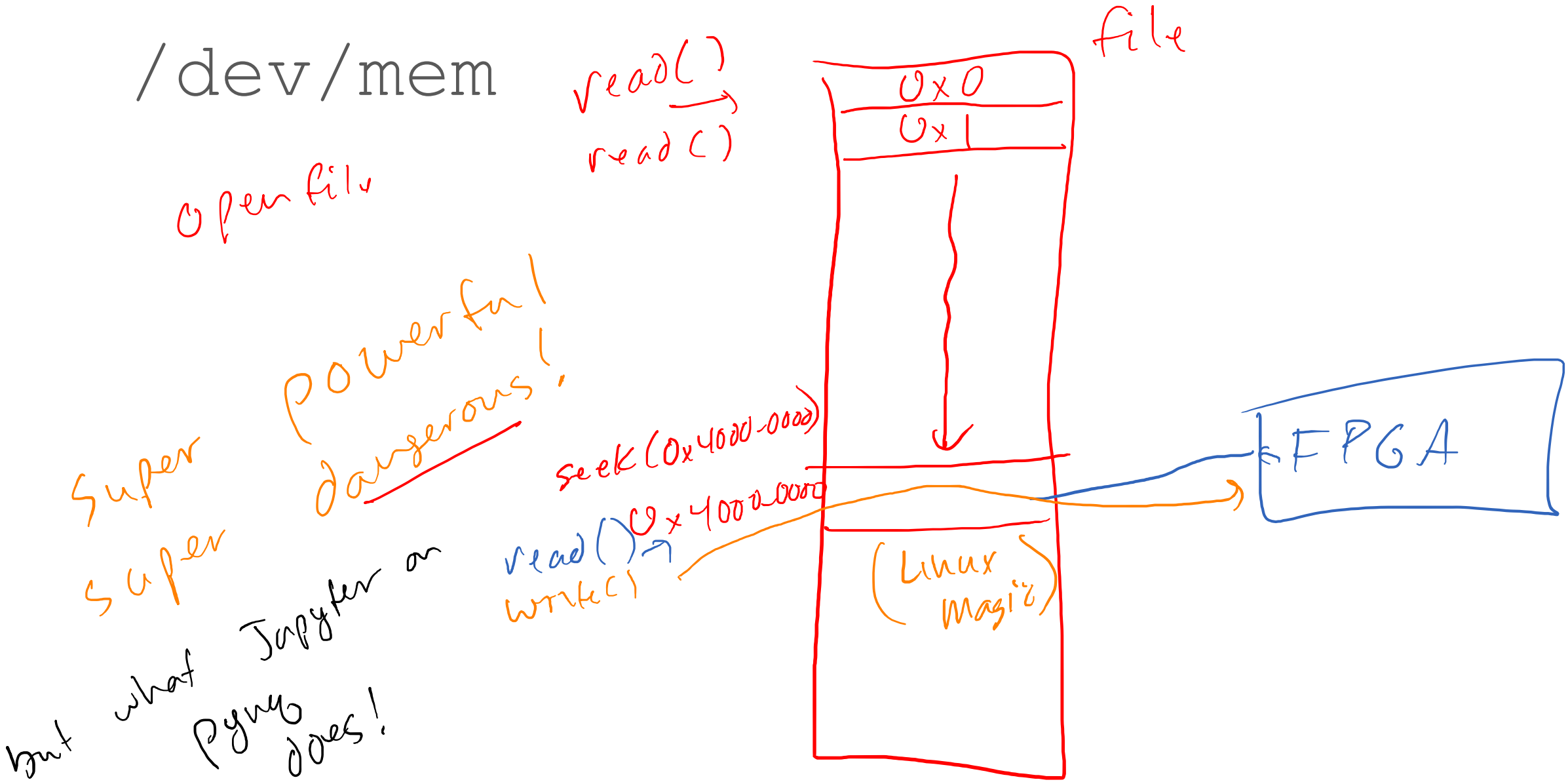
/dev/mem

next time...

- */dev/mem* is a character device file that is an image of the main memory of the computer. It may be used, for example, to examine (and even patch) the system.
- Byte addresses in */dev/mem* are interpreted as physical memory addresses. References to nonexistent locations cause errors to be returned.
- Requires root (sudo) access

/dev/mem

/dev/mem


file
0x0
0x1
read()
read()
open file
seek(0x4000-0000)
read()
write()
0x4000-0000
FPGA
(Linux
Magic)
Super
super
powerful
dangerous!
but what Jupyter or
Pynq does!

Python Example

```
from pynq import Overlay
from pynq import MMIO
class hw_ema():
    def __init__(self):
        self.overlay = Overlay('bitstream.bit')
        self.mmio = self.overlay.axi_popcount_0.S_AXI_LITE
    def ema(self, n):
        self.mmio.write(0x0, int(n))
        return self.mmio.read(0x0)
ema = hw_ema()
for i in range(1000,6000,1000):
    x = ema.ema(i)
    print ("In: ", i, " Out: ", x)
```

```
$ sudo python3 mmio_demo.py
In:  1000  Out:  250
In:  2000  Out:  687
In:  3000  Out:  1264
In:  4000  Out:  1948
In:  5000  Out:  2711
```

Next Time:

- /dev/mem
- /dev/uio

09: Memory Translation

Engr 315: Hardware / Software Codesign

Andrew Lukefahr
Indiana University