P7. failed AG?

Exam Review

Engr 315: Hardware / Software Codesign

Andrew Lukefahr
Indiana University

Some material taken from EECS370 at U. of Michigan

Announcements

- P7: Due Friday.
  - New SD Card / Linux Image
- Exam: on Monday
- P8: Coming Soon.

Exam Details

- Main 5 sections
  - Multiple questions / section
- Some short answer
- Some fill-in-the blank/code/table

A “Cheat” Sheet is Allowed

- 2-sided
- 8.5”x11” paper
- Handwritten (not photocopied)

Major Topics

- Performance Profiling
- Data Structures

- Bus Interfaces
- MMIO
- DMA

Performance Profiling

- What is profiling?
- What function should we be optimizing here?
- How do you know?

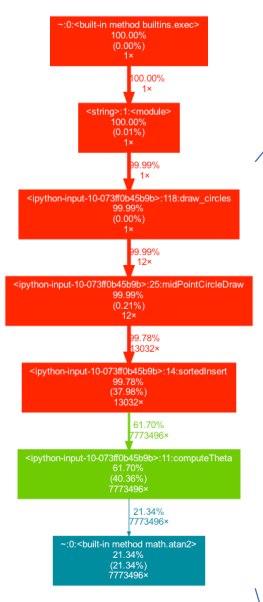
understand where application is spending time

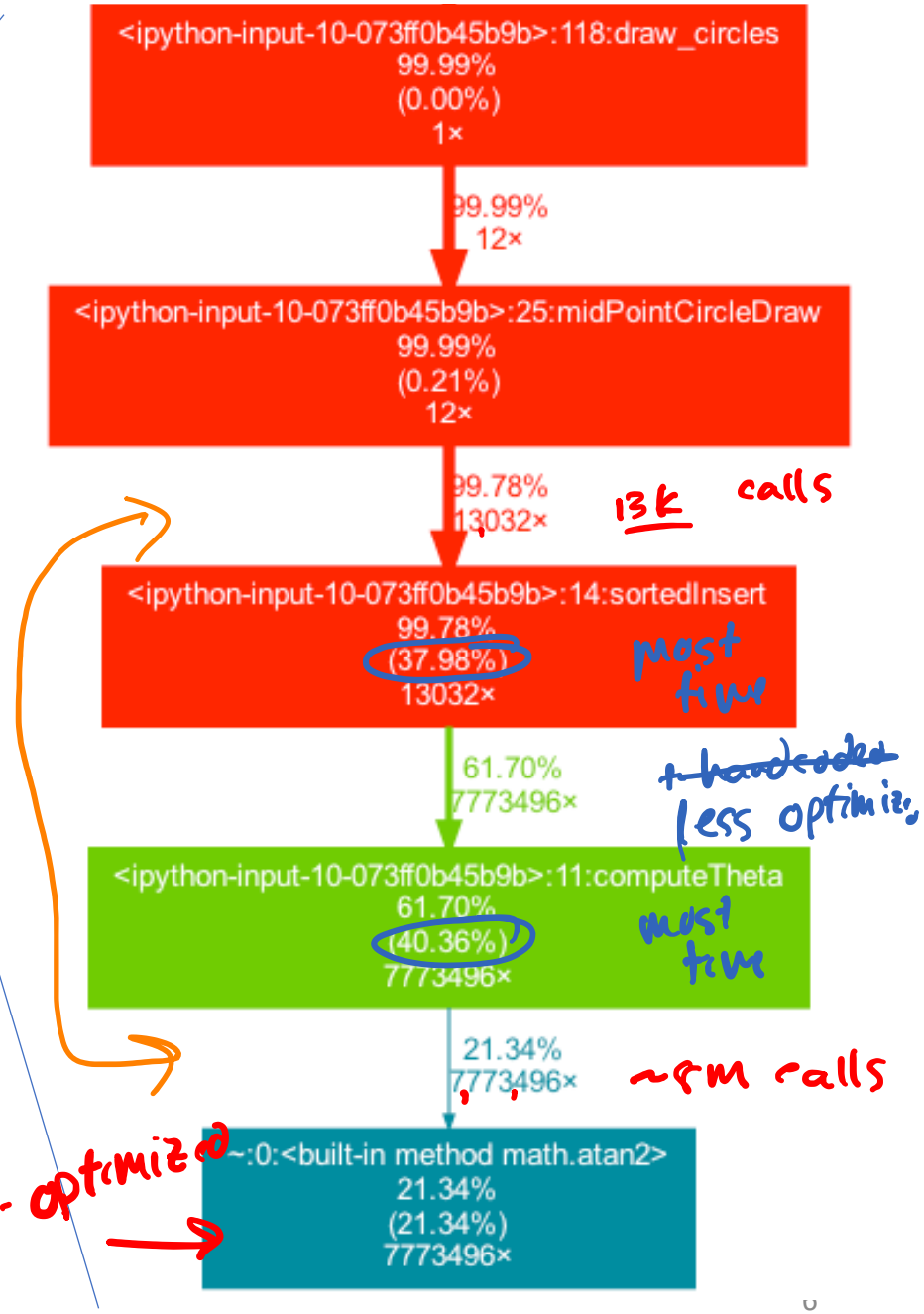
optimize places where spending time?

Sort.Insert + computeTheta

most time

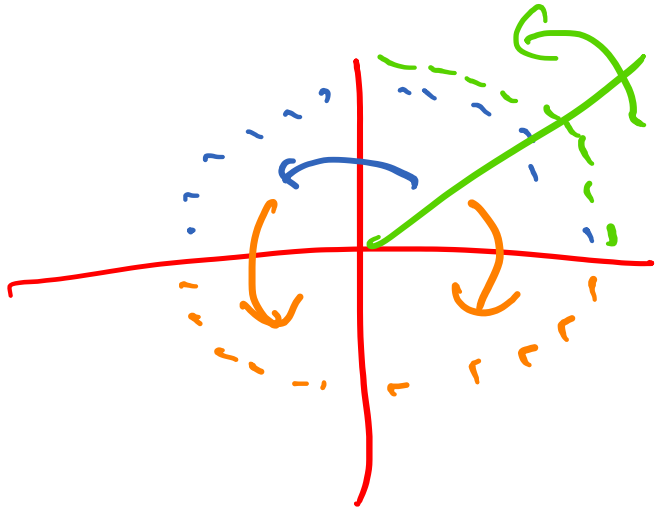
most calls





fctn calls expands

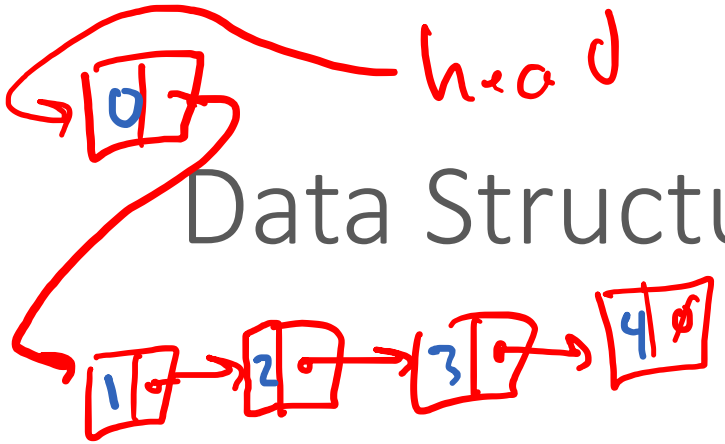
Algorithm Tuning



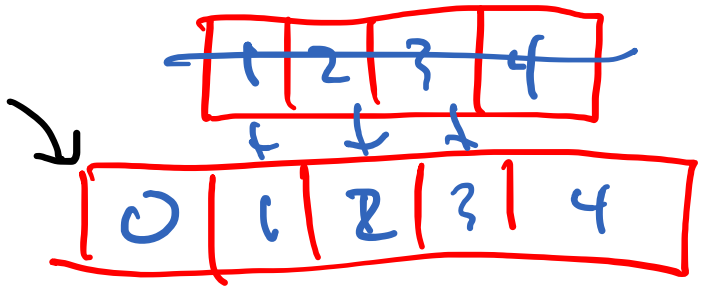
```
def computeTheta(self, x,y, x_centre, y_centre):
    return math.atan2(x-x_centre, y-y_centre)

def sortedInsert(self, theList, x, y, x_centre, y_centre):
    for index, value in enumerate(theList):
        oldTheta = self.computeTheta(value[0],value[1],x_centre,y_centre)
        newTheta = self.computeTheta(x,y, x_centre, y_centre)
        if oldTheta > newTheta:
            theList.insert(index, (x,y))
            return theList
    theList.append((x,y))
    return theList
```

Old sort

Sort here

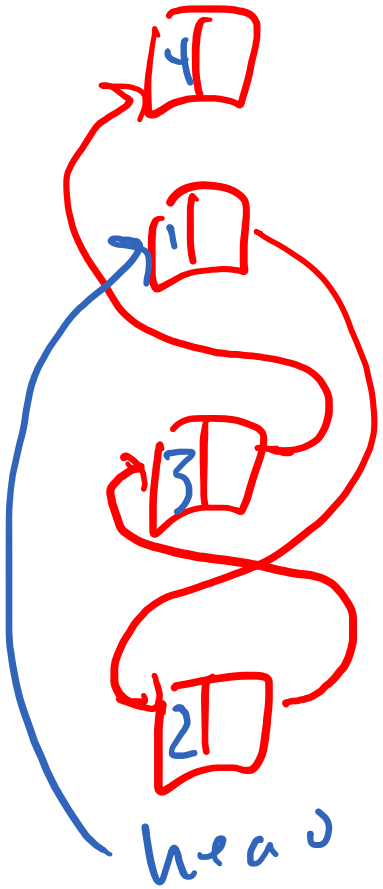


Data Structures

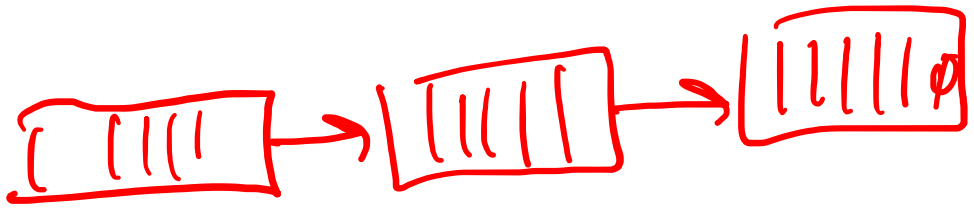


- When is better here? list or array?
  - Inserting at the beginning? → list
  - Accessing the element at position N (i.e. values[n]) Array
  - Accessing elements sequentially?

Array (barely)



- ~~What's funny about Python's lists~~?



Bus Interfaces

P3 - EMA (Stream)

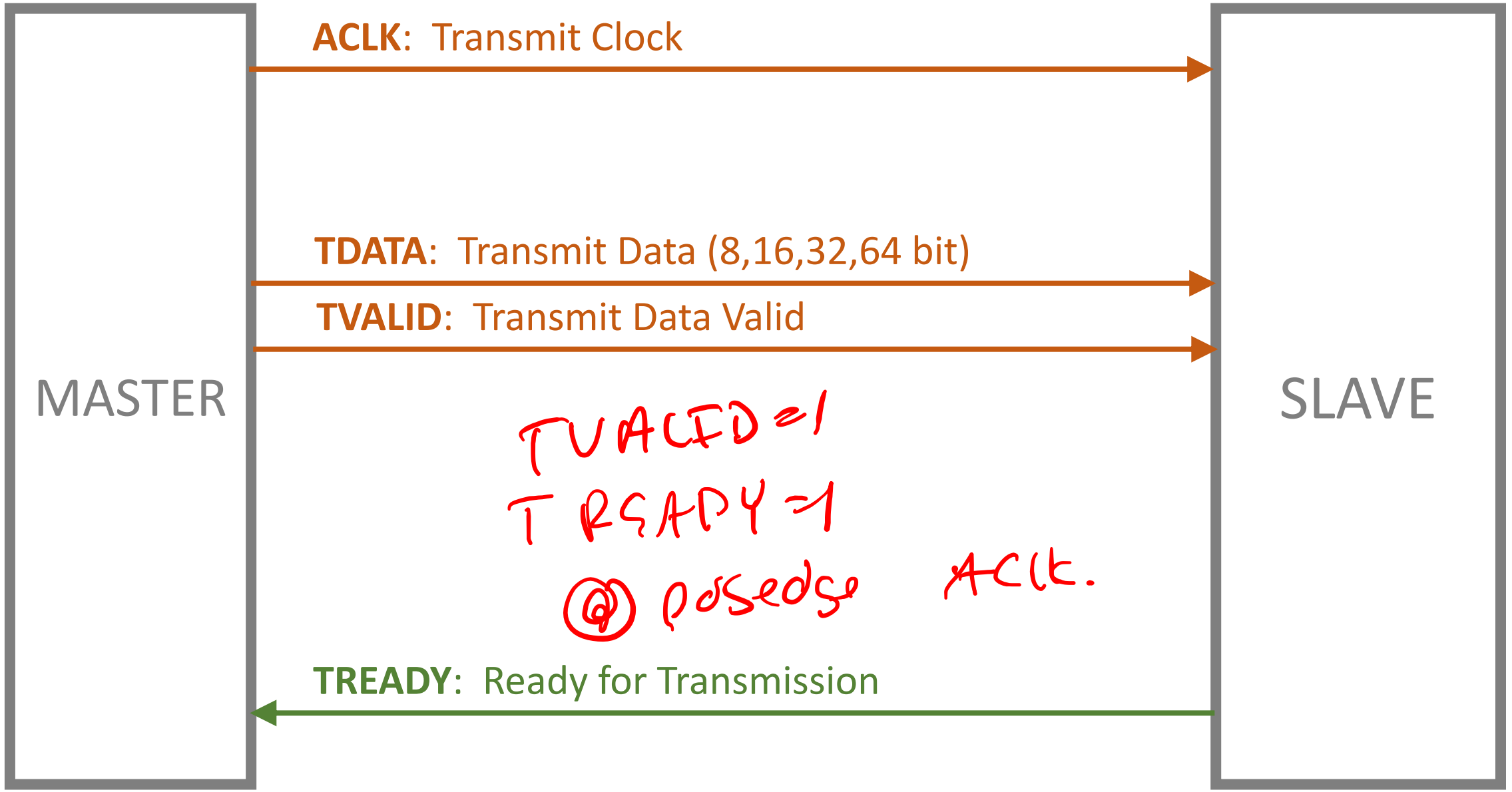
P4/P5 Popcut MMIO (Lite)

P6/P7 PC. DMA (Full)

- AXI4 “Full” vs. AXI4 Lite vs. AXI4 Stream

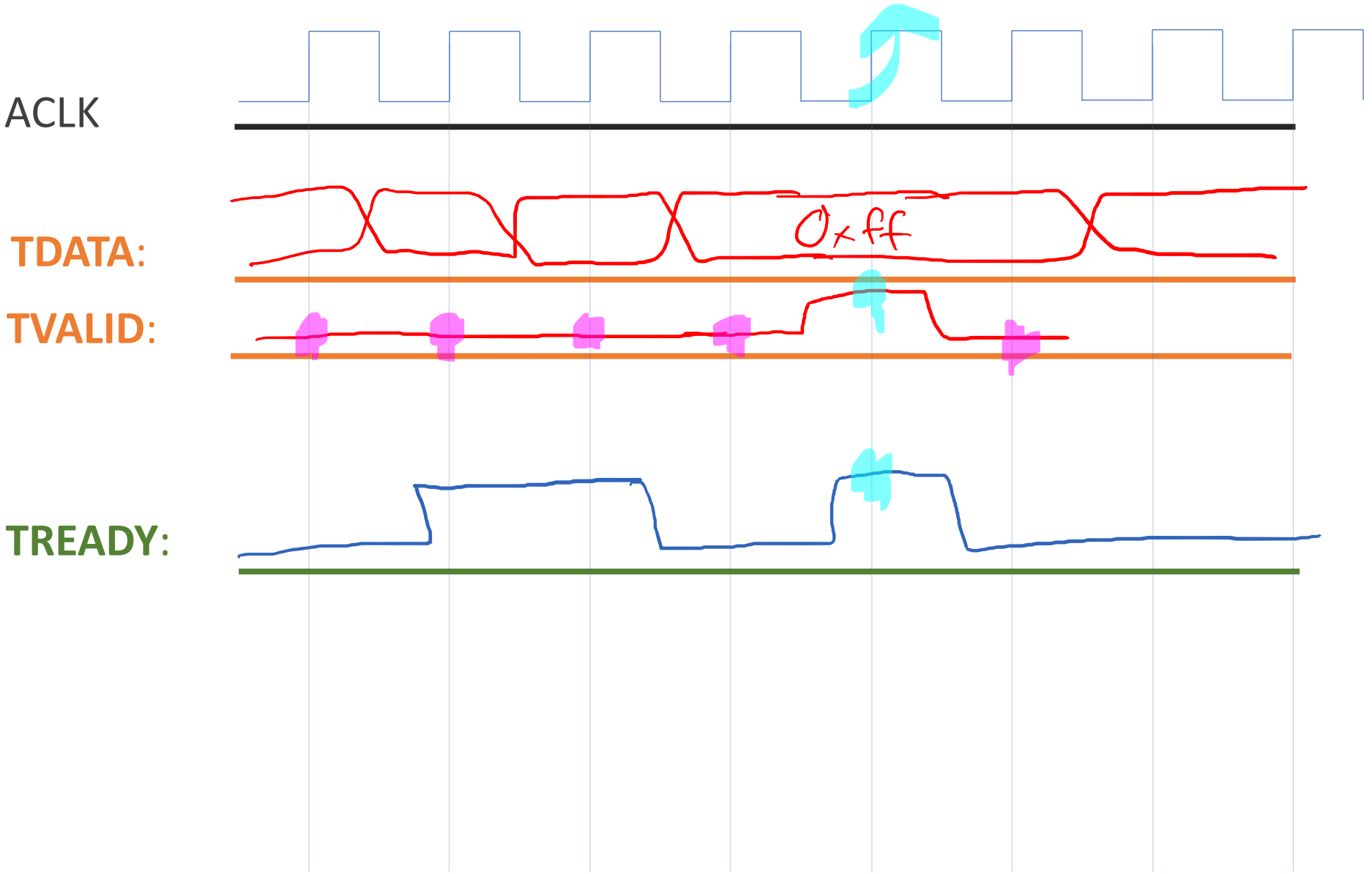
- What are the benefits of each?
- Where do we use them?

	Full	Lite	Stream
Benefits -	MMIO High. Perf.	MMIO Simple	Simple, High Perf.
Drawbacks -	High Complexity	Low perf.	NO MMIO

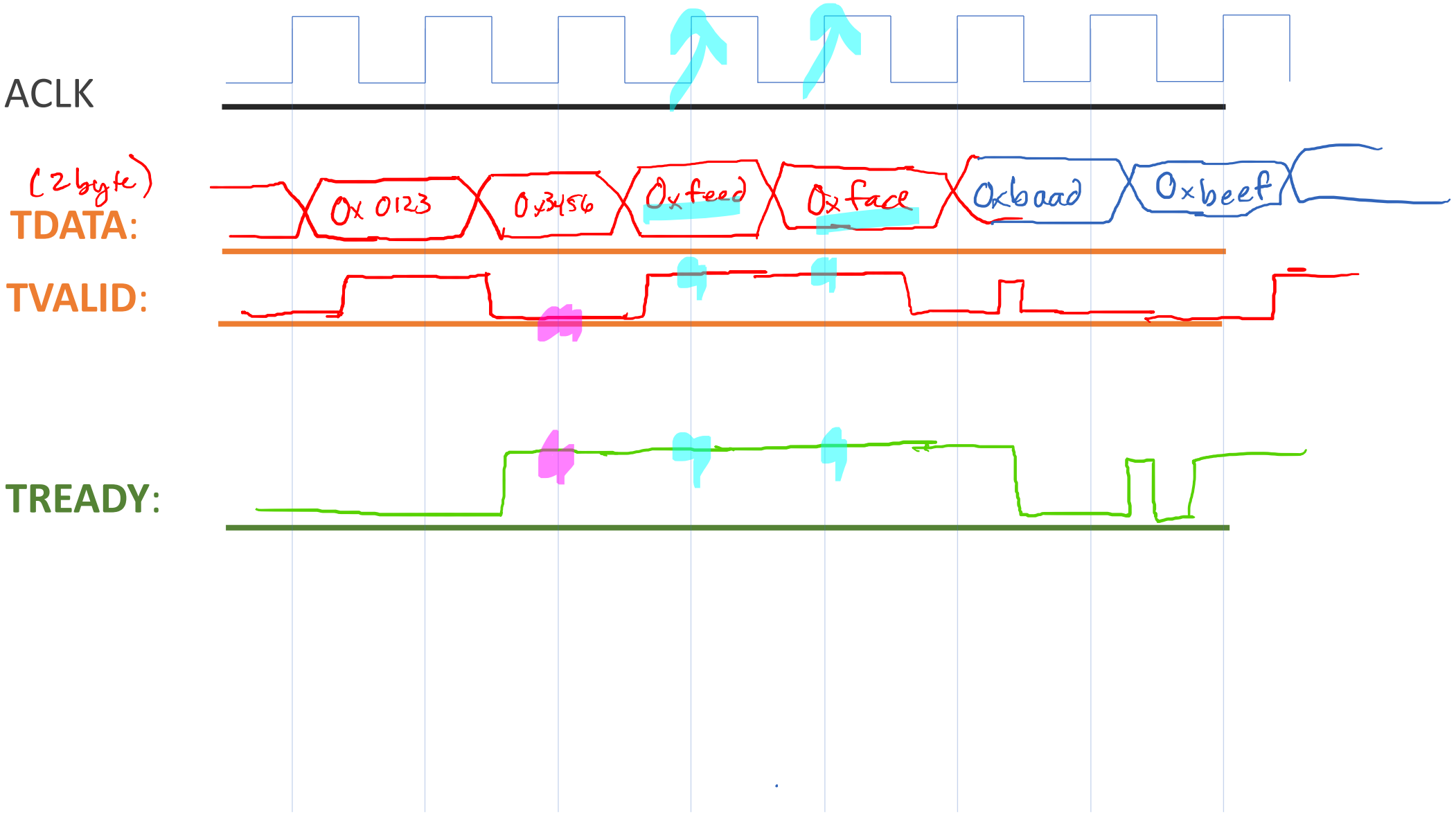


When is a transmission valid?

Transferring data on a AXI4-Stream Bus.

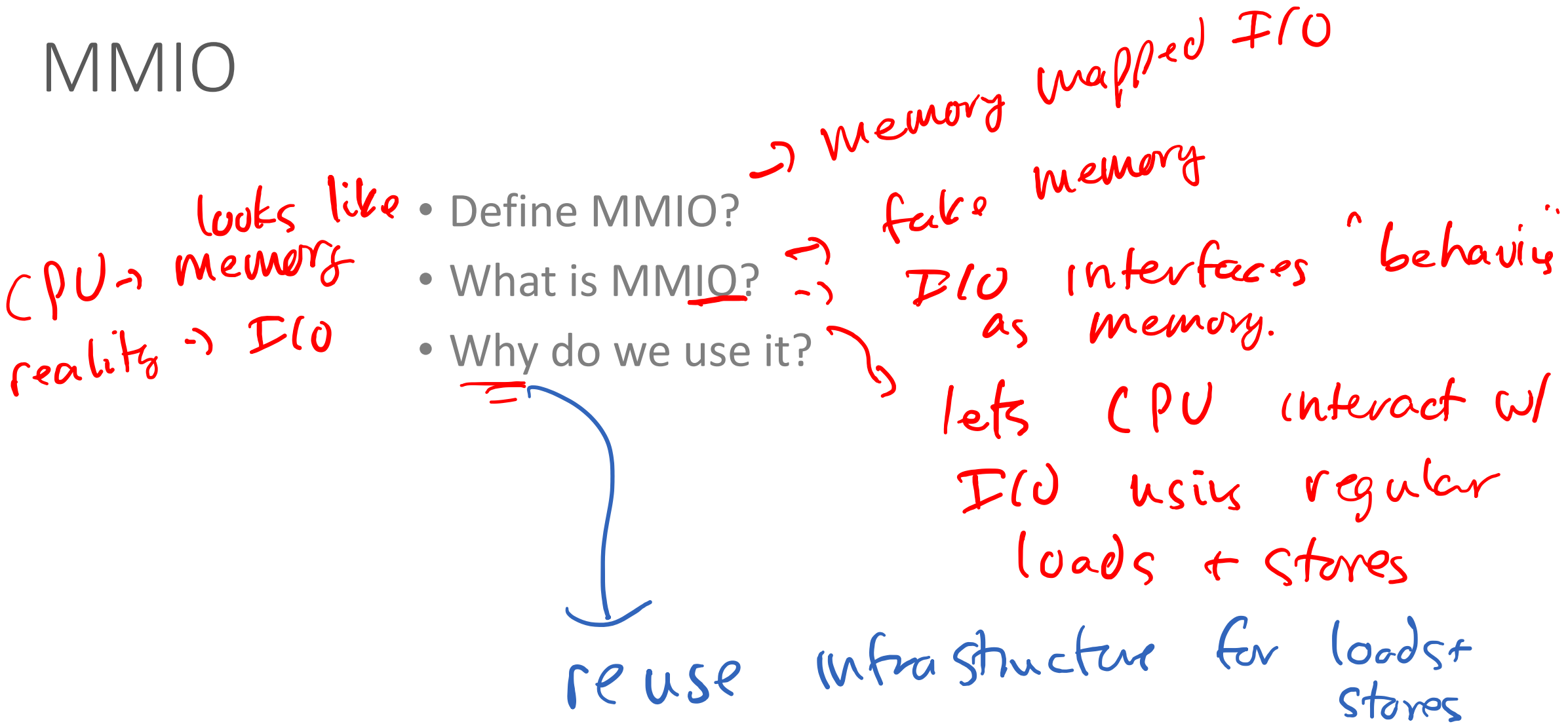


Transferring data on a AXI4-Stream Bus.



MMIO

- Define MMIO?
- What is MMIO?
- Why do we use it?



MMIO Loads

volatile uint32_t * count_reg = vbase_ptr + 0;

*count_reg;

- In ASM?

```
mov r2, 0x40000000 ;
ldr r3, [r2, +0x144];
```

load from 0x4000_0140

- In C?

#define ADDR 0x4000_0144

int x = *((volatile uint32_t *)(ADDR 0x4000 0140))

↑ why? subject to change value randomly

MMIO Loads and Stores

- In ASM?

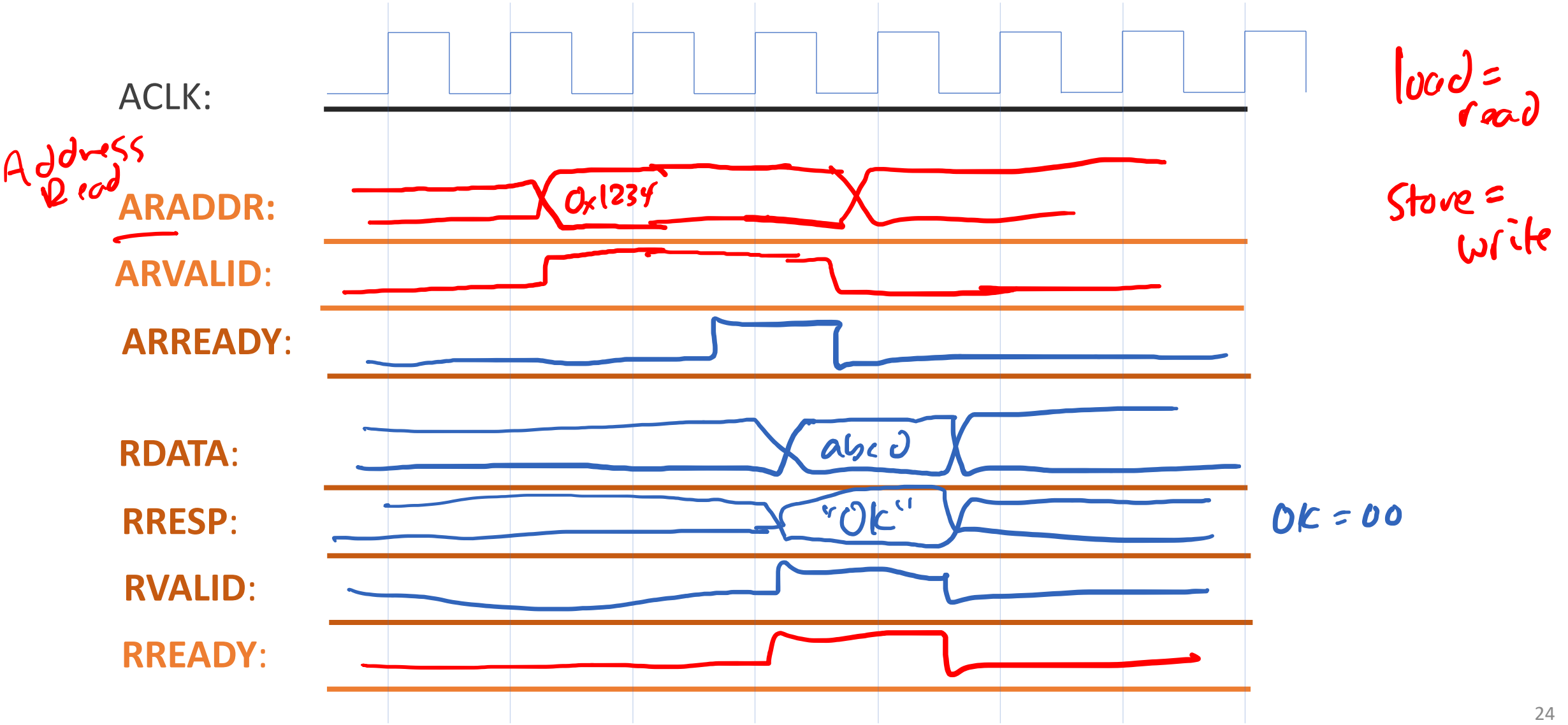
```
mov r2, 0x40000000 ;
ldr r3, [r2, 0x144];
```

- In C?

```
uint32_t  x = *(volatile uint32_t *)(0x40000144);
```

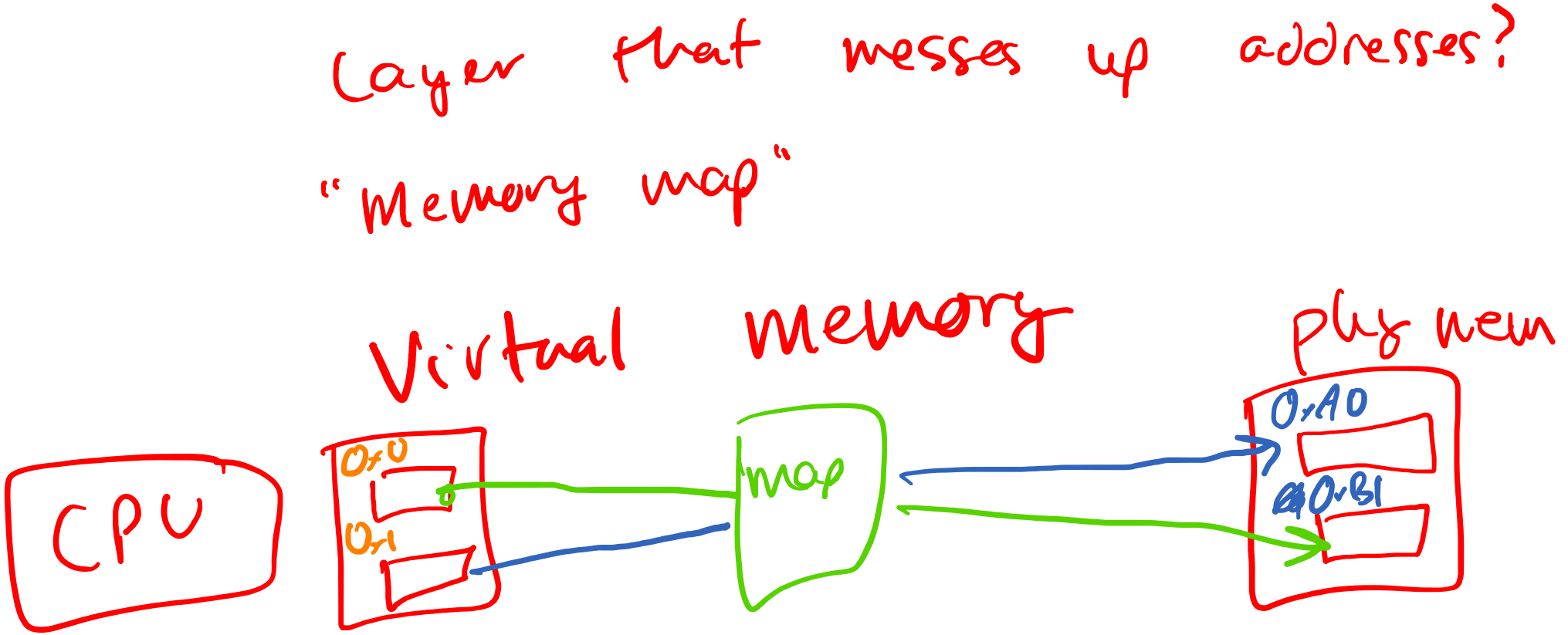
Store: *(volatile uint32_t *)(0x40000144) = 32;

AXI4Lite: Load 0x1234, response: 0xabcd



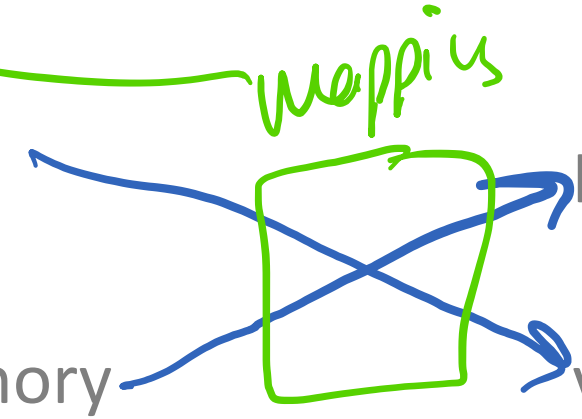
Linux MMIO?

- What's weird about C/MMIO with Linux?



Virtual Memory

- Linux/Hardware “translates” CPU’s memory addresses into real memory addresses.

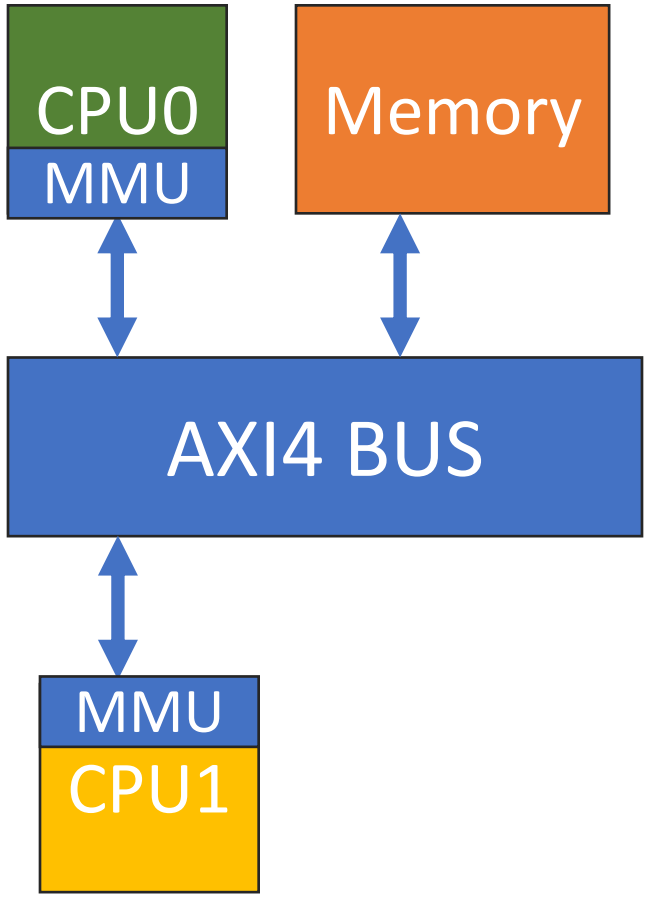


(0x4000-0000)

/dev/uio0
+ mmap

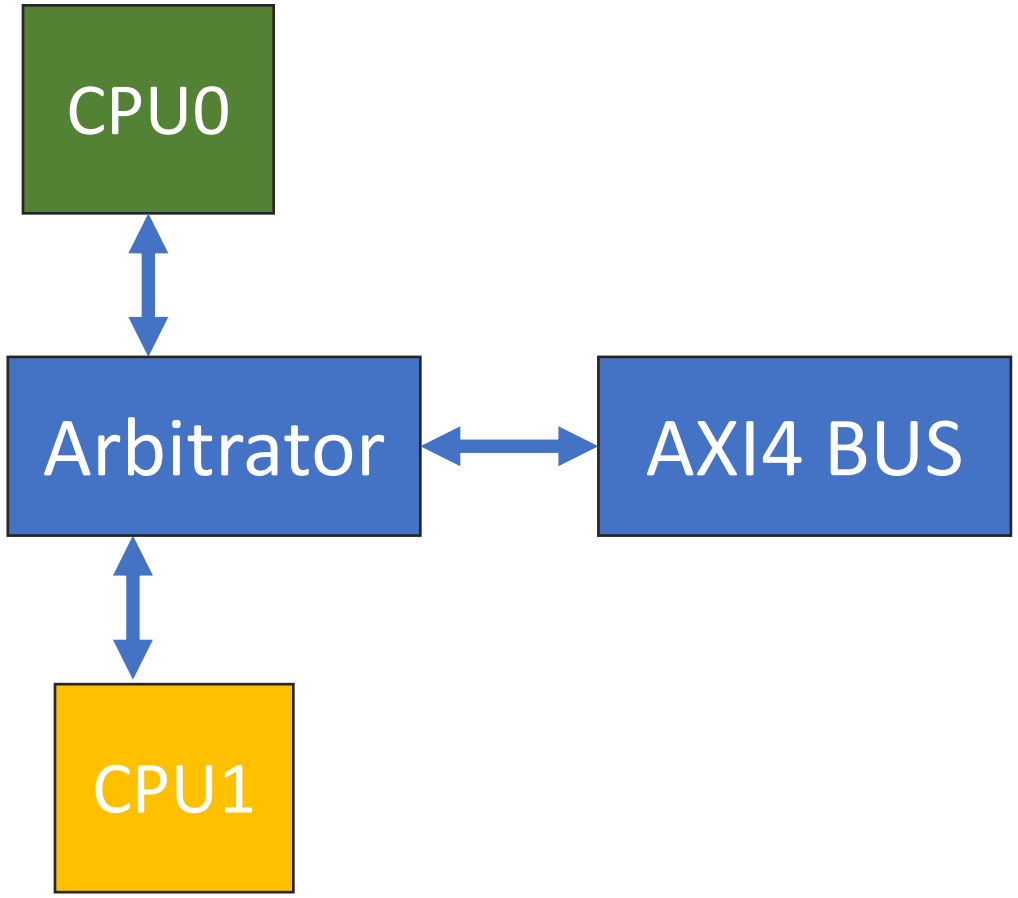
Workaround
Virtual → Fixed Physical Address

Multiple Masters



- What happens if both CPUs request a transaction at the same time?

An Arbitrator selects who gets to use the bus



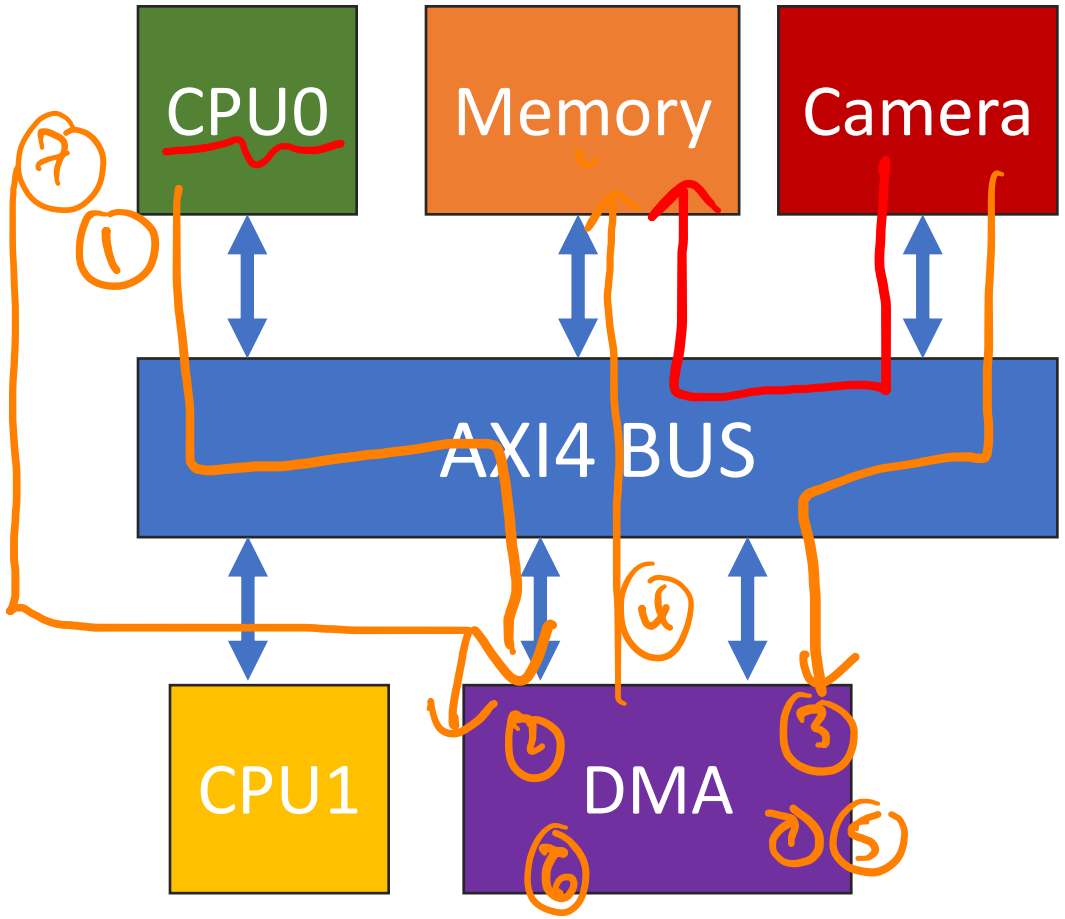
- What happens if both request a transaction at the same time?
- Arbitration: Pick a winner!
- What Arbitration scheme to use?

Fixed Priority: CPU 0 always wins

Round Robin: changes winner

DMA

- Define DMA?
- What's the goal of DMA?
- What steps are involved?



Direct Memory Access

→ Separate thing CPU used for memory copy

free-up a real CPU, use thing-one instead for memory copies

① CPU tells DMA what to copy
② DMA sets "start" bit
③ Load from camera
④ store to memory
⑤ loop for all memory
⑥ DMA clears "start" bit
⑦ CPU polls for "start" bit clear

DMA Control Design

start →

```
void dma (uint32_t *from,
          uint32_t *to,
          uint32_t size)
{
   register uint32_t reg;

   for (int i = 0; i < size; ++i){

      reg = from[i]; //load

      to[i] = reg; //store
   }
}
```

= source address

= destination address

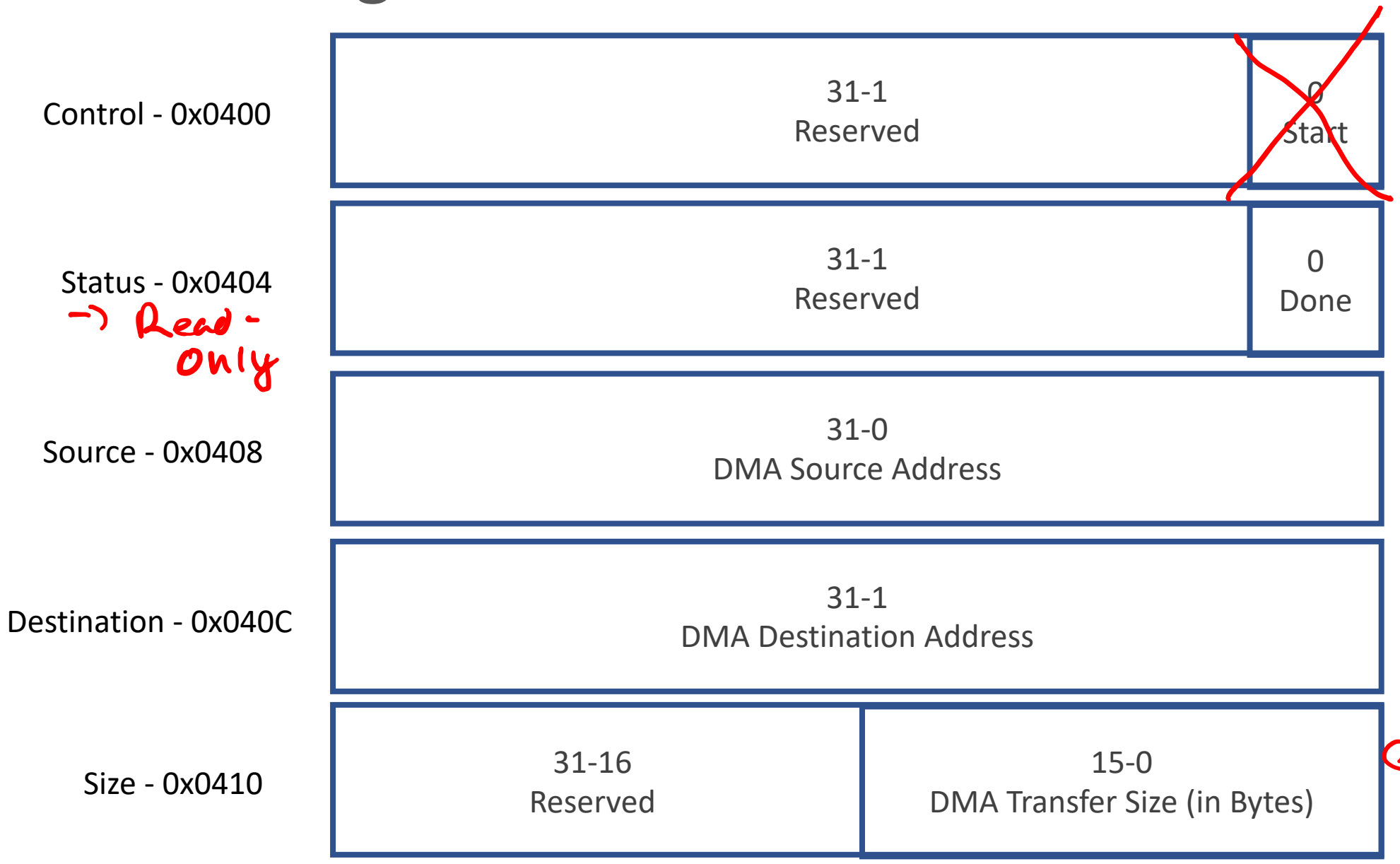
stop.

DMA Control Design

```
void dma (uint32_t * from,
          uint32_t * to,
          uint32_t size)
{
   register uint32_t reg;

   for (int i = 0; i < size; ++i){

      reg = from[i]; //load

      to[i] = reg; //store
   }
}
```

- What interfaces do you need?
- How do you start/stop DMA?
- Design a DMA state machine?

All DMA Registers



DMA Control

```
void dma (uint32_t * from,
          uint32_t * to,
          uint32_t size)
{
   register uint32_t reg;

   for (int i = 0; i < size; ++i){

      reg = from[i]; //load

      to[i] = reg; //store
   }
}
```

- AXI4 Master Interface
  - Loads + Stores
- 5 MMIO registers
  - Control (Start)
  - Status (Done)
  - Source (`from`)
  - Destination (`to`)
  - `size` (in Bytes)

Using DMA from CPU's side:

```
void dma (uint32_t * from,
          uint32_t * to,
          uint32_t size)
{




}
```

0x0400: Control Register
0x0404: Status Register
0x0408: Source Address
0x040C: Destination Address
0x0410: Transfer Size in Bytes

Using DMA from CPU's side:

0x0400: Control Register
0x0404: Status Register
0x0408: Source Address
0x040C: Destination Address
0x0410: Transfer Size in Bytes

```
void dma_copy ( uint32_t * src,
                uint32_t * dest,
                uint32_t size){

    *((volatile uint32_t *)(0x0408))=src;
    *((volatile uint32_t *)(0x040C))=dest;
    *((volatile uint32_t *)(0x0410))=size;
    *((volatile uint32_t *)(0x0400))= 0x1; //start

    //spin until copy done
    while( *((volatile uint32_t *)(0x0404)) != 0x1){;}
}
```

DMA System Interface

Exam Review

Engr 315: Hardware / Software Codesign

Andrew Lukefahr
Indiana University