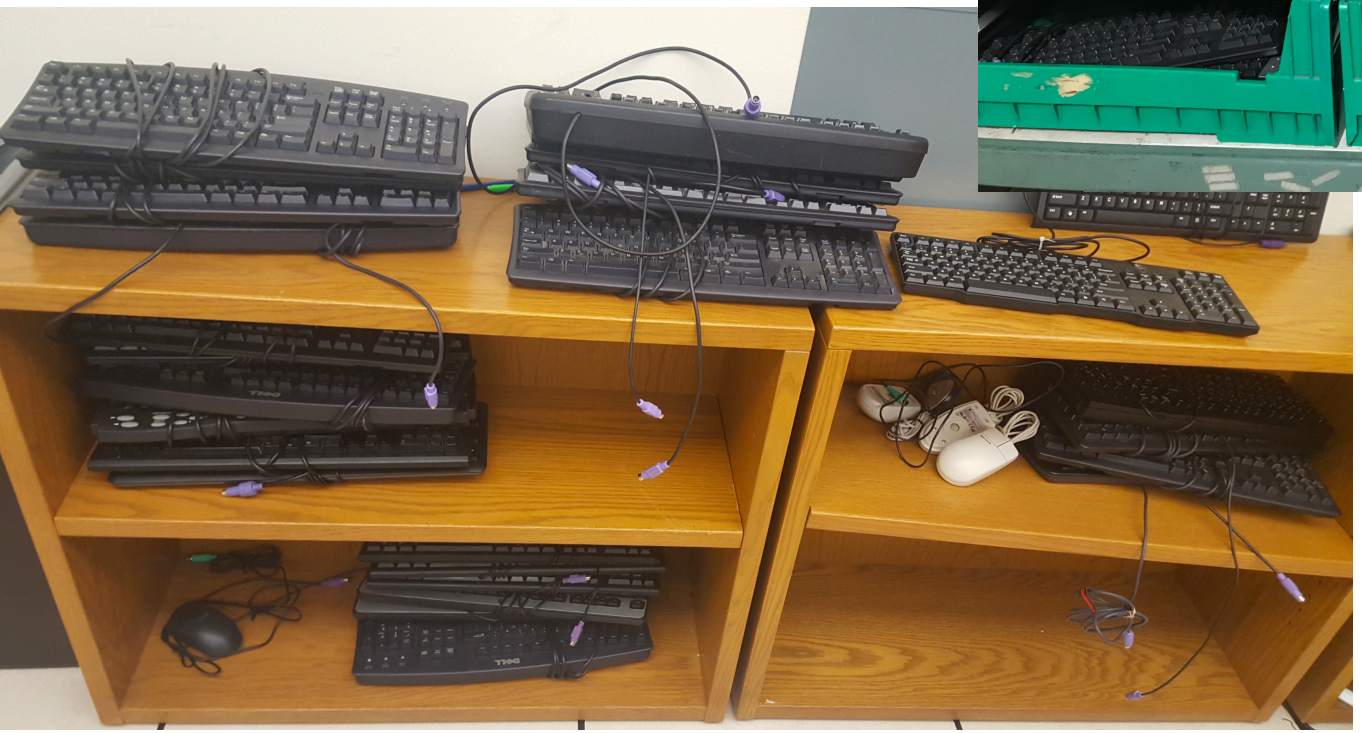
Graphics and Framebuffers

Thanks for all the feedback!

Some changes

- **2 CAs in some OHs; experiment with different ways of allocating time to students in OH**
- **Better preview of the upcoming assignment in lecture. YEAH OH on Thu (Michelle).**
- **No lecture on President's Day**



Lab catch-up
Julie and Pat
OH at 1pm today

Layers (keyboard.h)

```
unsigned char keyboard_read_scancode(void)
```

- returns scancode

```
int keyboard_read_sequence(unsigned char seq[])
```

- returns sequence of up to 3 scan codes (including PS2_CODE_RELEASE=0xF0, PS2_CODE_EXTEND = 0xE0)

```
key_event_t keyboard_read_event(void)
```

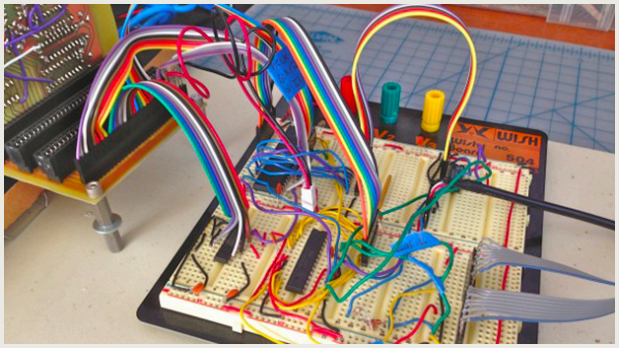
- returns key_event struct : scancode, modifiers, action, …
- keeps track of modifier keys

```
unsigned char keyboard_read_next(void)
```

- returns ASCII characters and special keys (>= 0x90)
- maps key events to ASCII characters

gpio
timer
uart
printf
malloc
keyboard
fb
gl
console
shell

lab64 Speaker Series: Nothing Half Way - Being Fearless With Your Projects



Topic: Nothing Half Way - Being Fearless With Your Projects

Friday, February 16, 2018 - 4:00pm

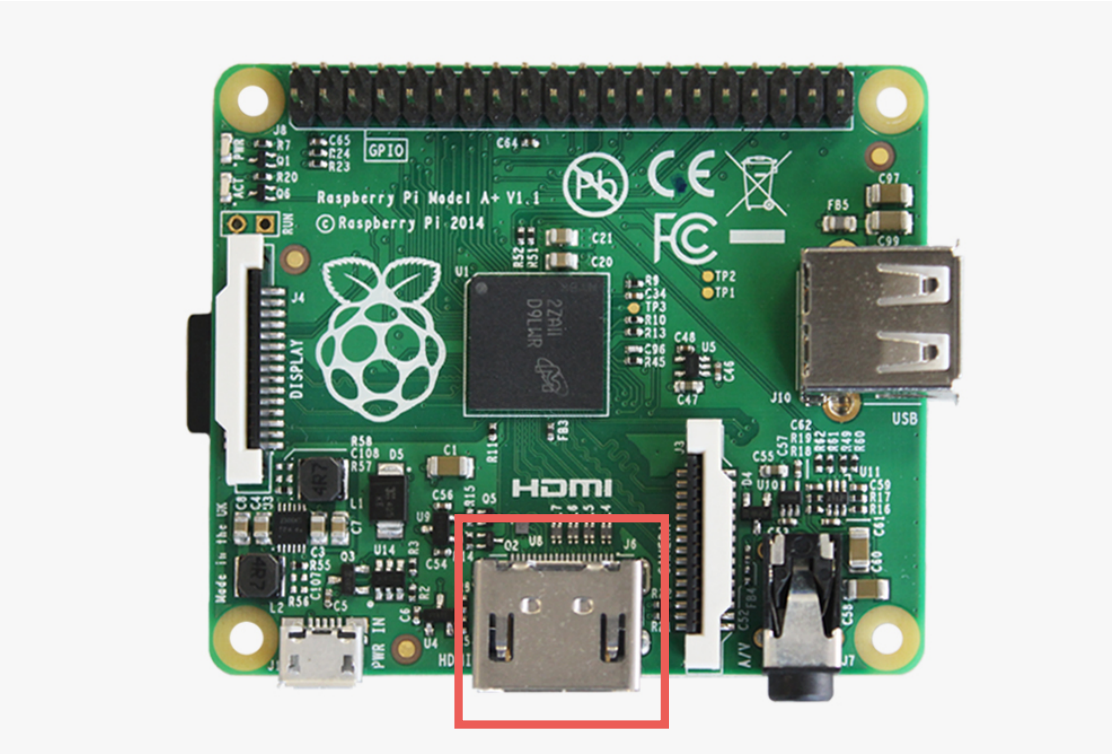
Venue: lab64 (Packard 064)

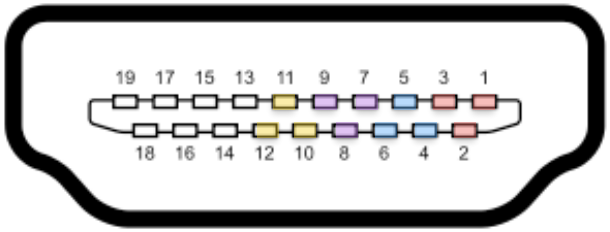
Speaker: Quinn Dunki (Scopely)

Abstract / Description:
Quinn has been making games for 36 years, on platforms ranging from the Apple II to all manner of newfangled things. She currently manages engineering for mobile games at Scopely. She also pursues consulting, independent development, mixed-media engineering projects, and writing. In her spare time she welds things, races cars, hacks electronics, and berates her friends with sarcasm.

www.quinndunki.com
www.quinndunki.com/blondihacks

HDMI





Clock

Data 0

Data 1

Data 2

Control

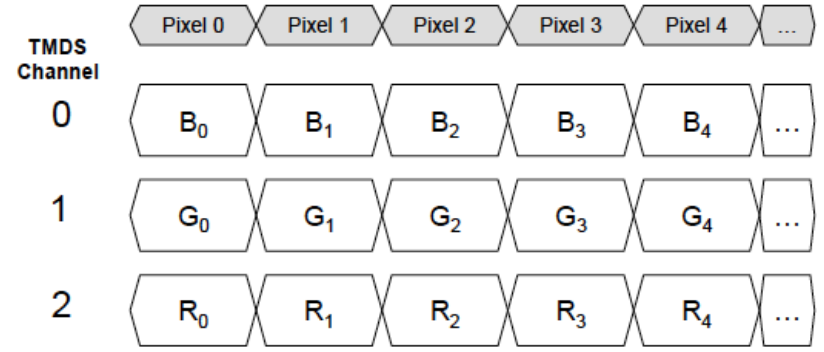
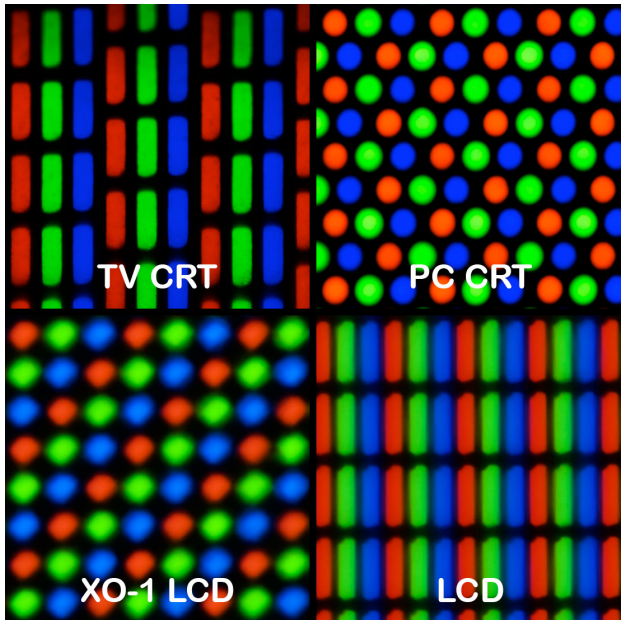


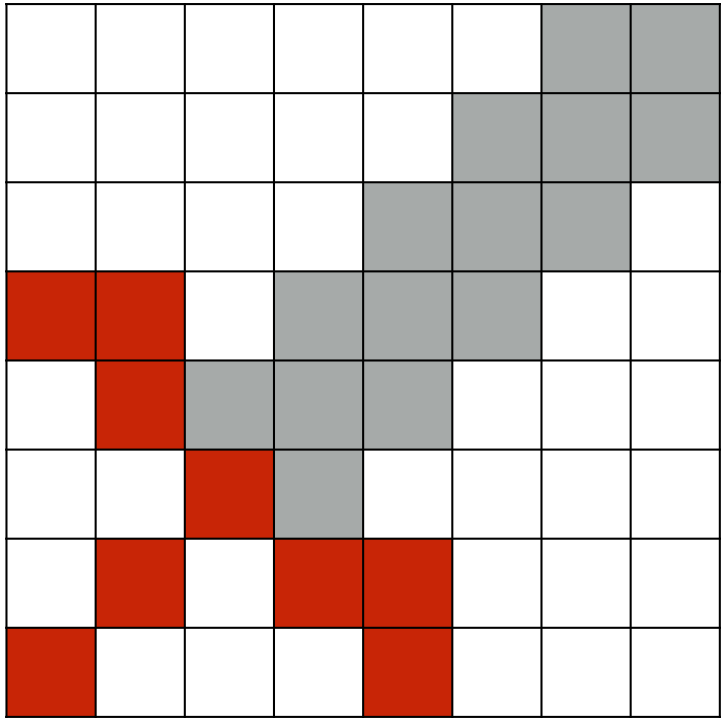
Figure 6-1 Default pixel encoding: RGB 4:4:4, 8 bits/component

Figure from High-Definition Multimedia Interface Specification Version 1.3a

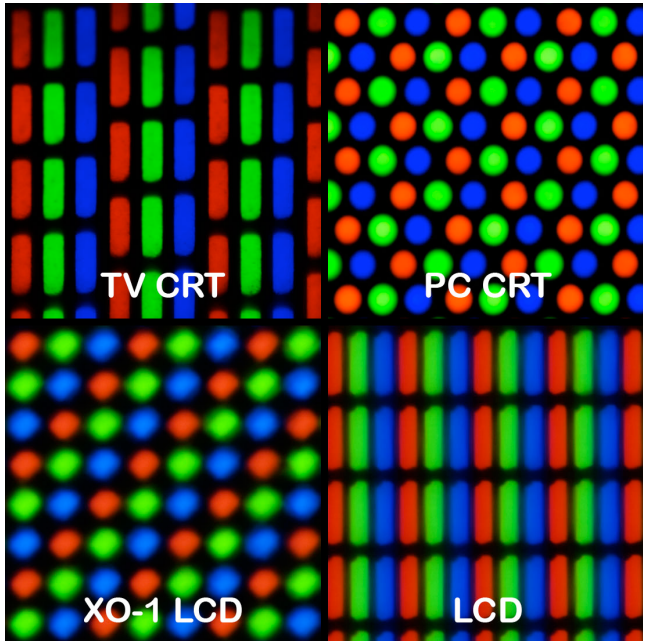


Displays

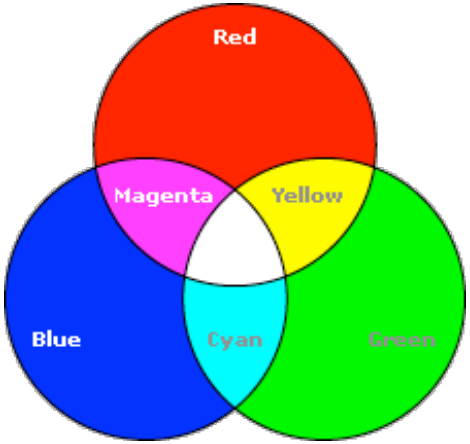
Pixels



Displays



Light



The framebuffer is an image

An image is a 2D array of pixels

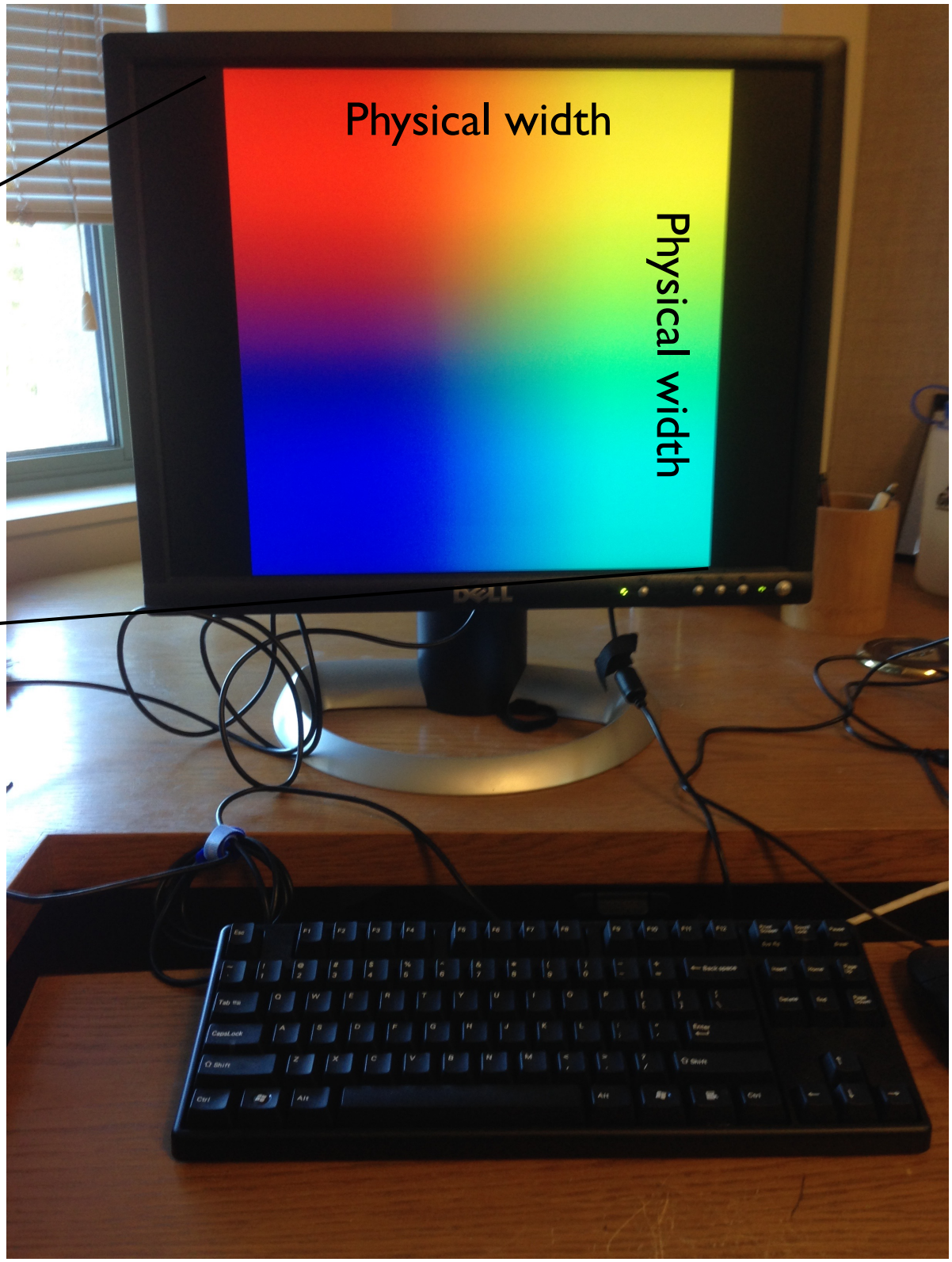
RGBA pixel (depth=32 bits)

Red = 8 bits
Green = 8 bits
Blue = 8 bits
Alpha = 8 bits

Framebuffer Resolution

read physical size
read virtual size
read pixel depth

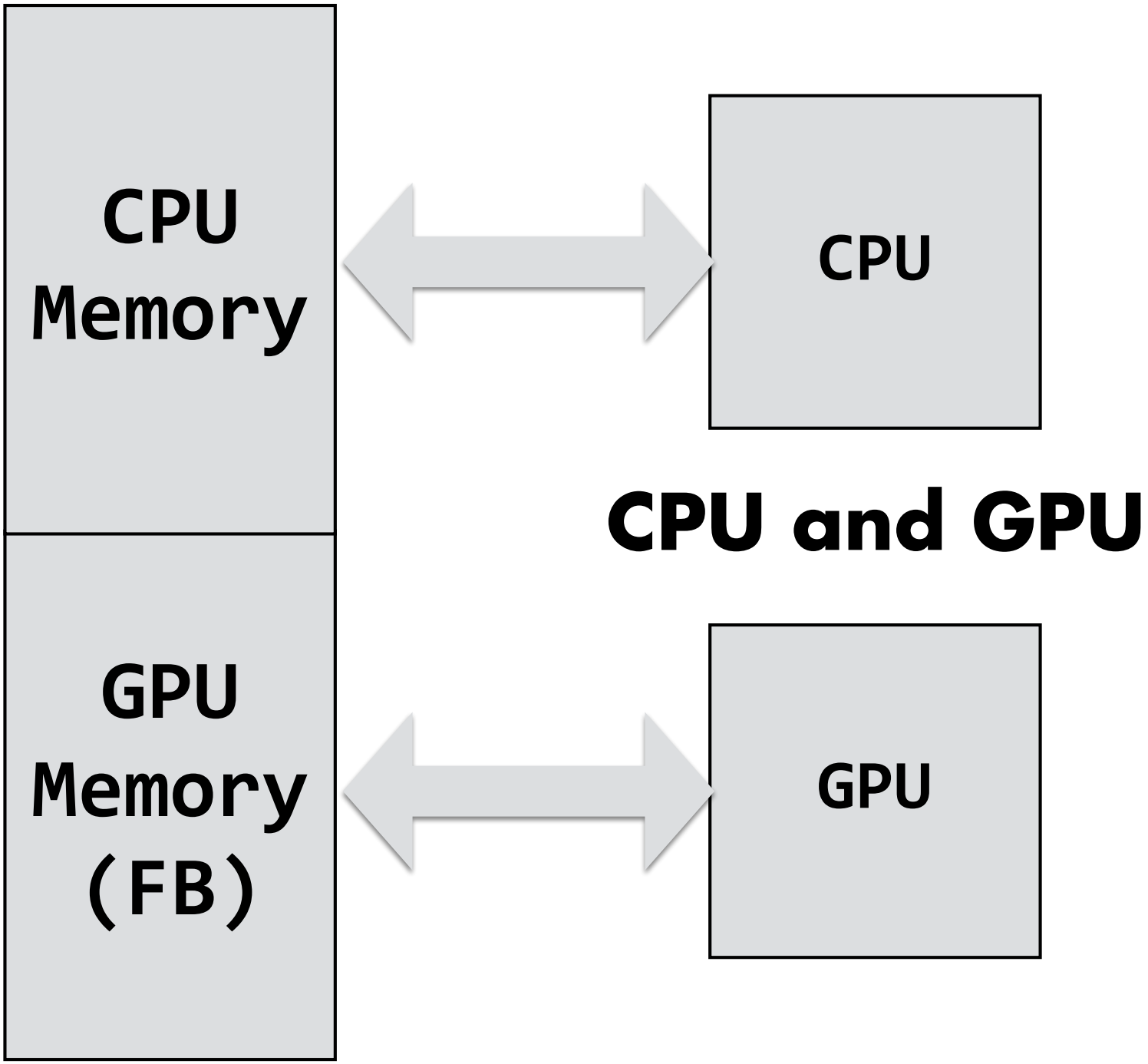
`code/video`



Virtual height = 2

Virtual width = 2

2x2 image is interpolated to 1600x1200 to fill monitor


CPU
Memory
CPU
CPU and GPU
GPU
Memory
(FB)
GPU

FB Config Structure

40 bytes long, specifies 10 parameters

Field	CPU	GPU	Description
width	write	read	Width of physical screen
height	write	read	Height of physical screen
virtual_width	write	read	Width of framebuffer
virtual_height	write	read	Height of framebuffer
pitch	read	write	Bytes/row of framebuffer
depth	write	read	Bits/pixel of framebuffer
x_offset	write	read	X offset of screen in framebuffer
y_offset	write	read	Y offset of screen in framebuffer
pointer	read	write	Pointer to framebuffer
size	read	write	Size of framebuffer in bytes

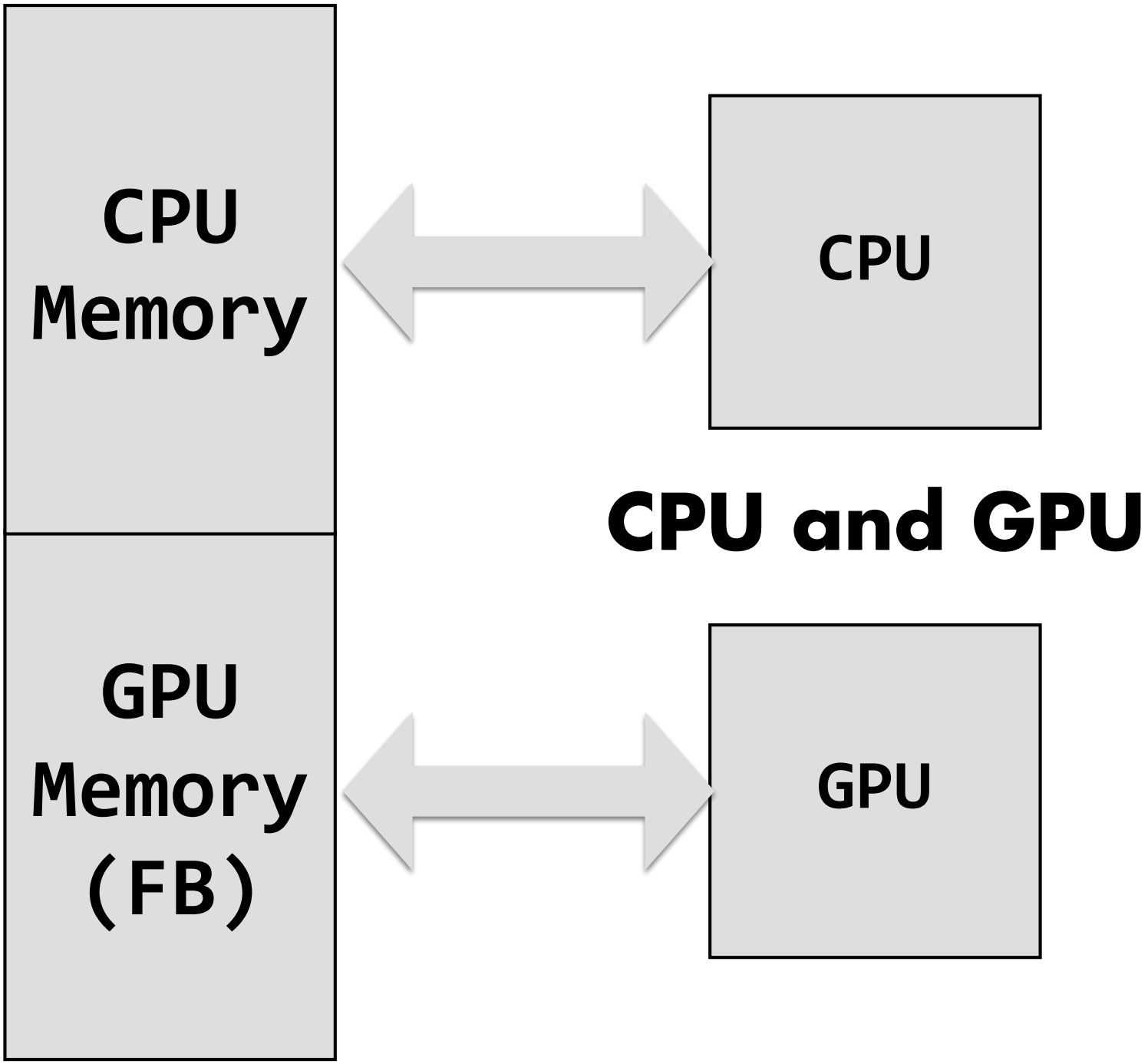
Configure Framebuffer Resolution

set physical size
set virtual size
set depth

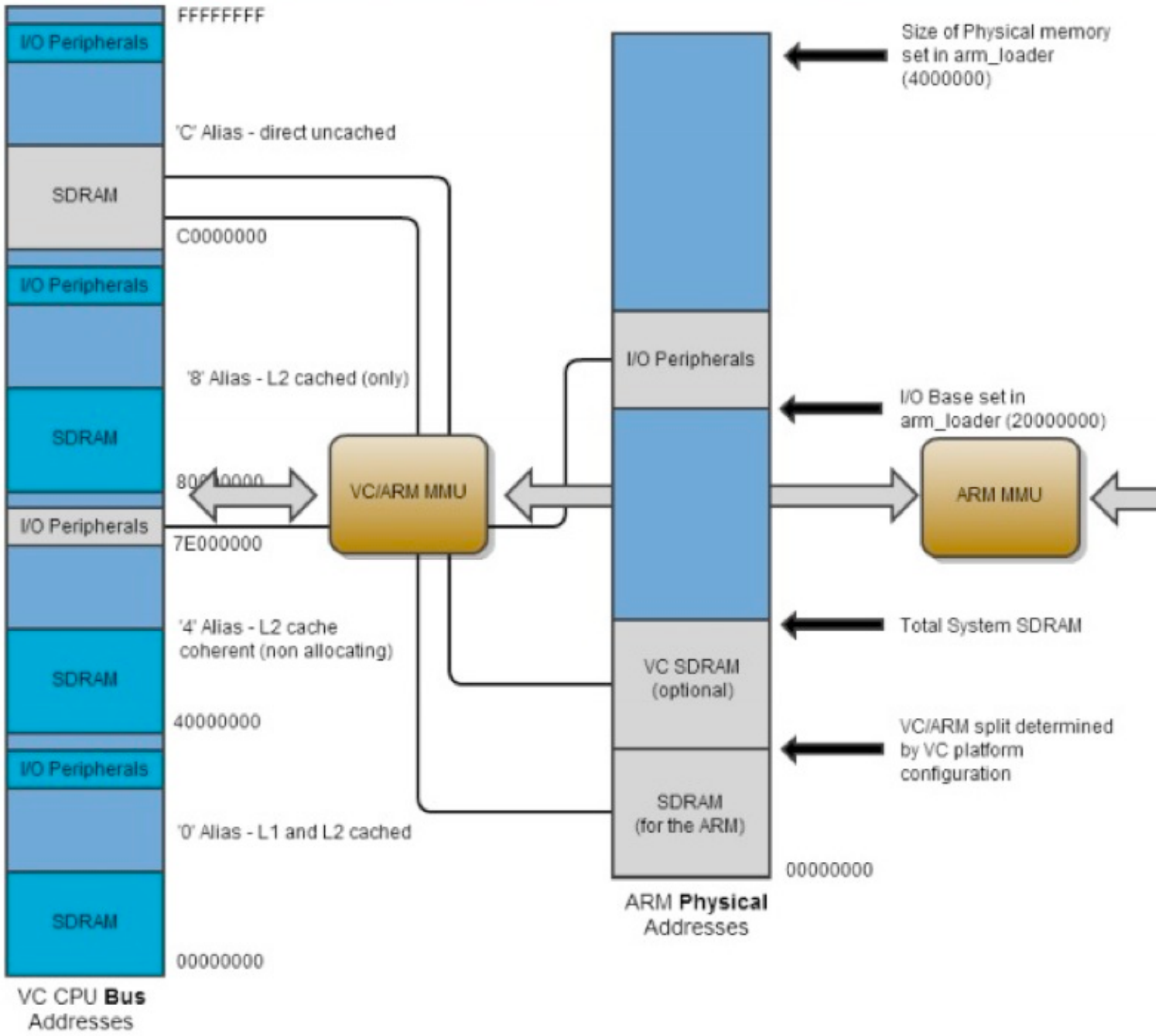
read pitch, size, fb pointer

`code/fb`

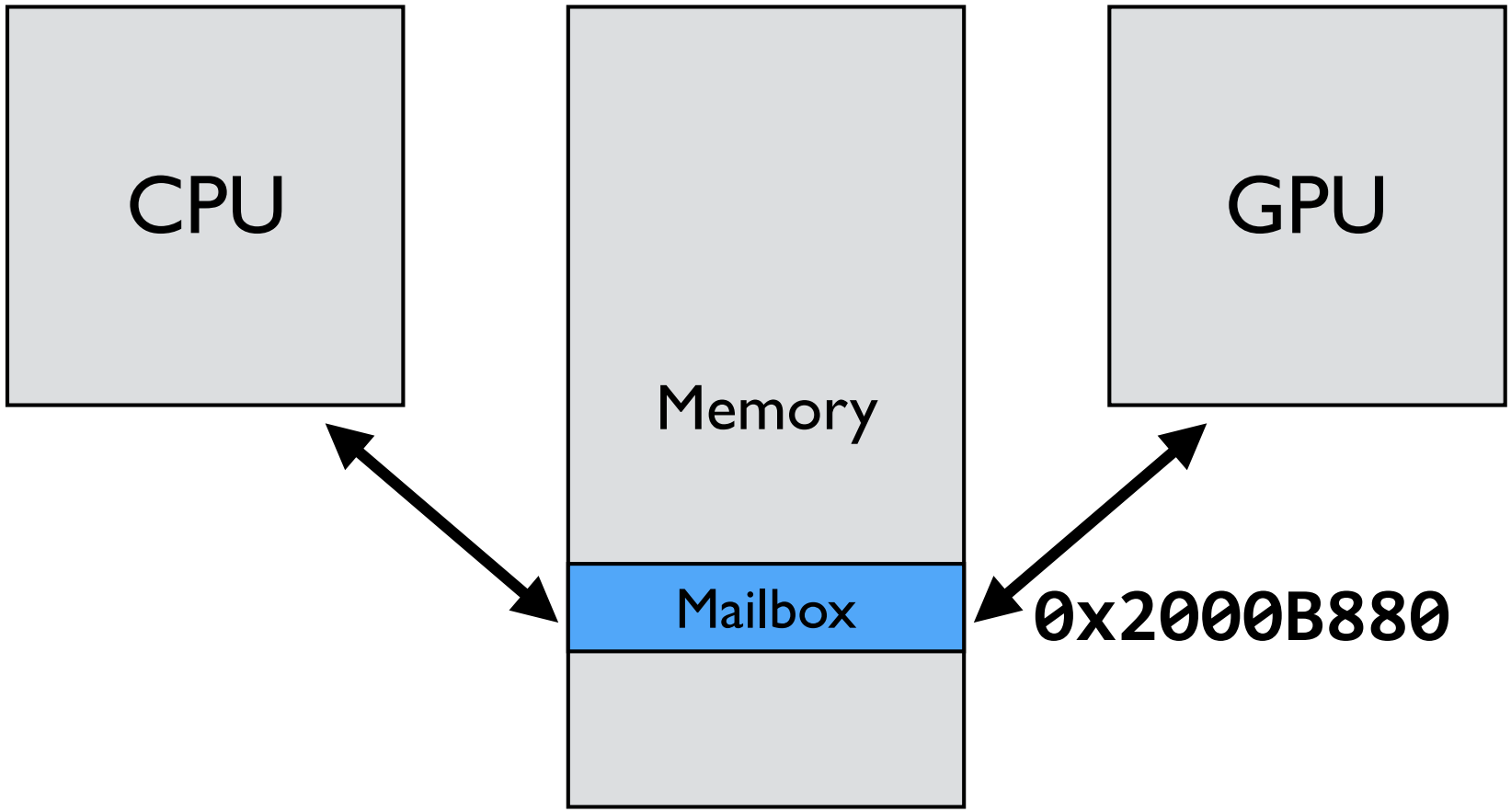
Mailbox


CPU
Memory
CPU
CPU and GPU
GPU
Memory
(FB)
GPU

BCM2835 ARM Peripherals

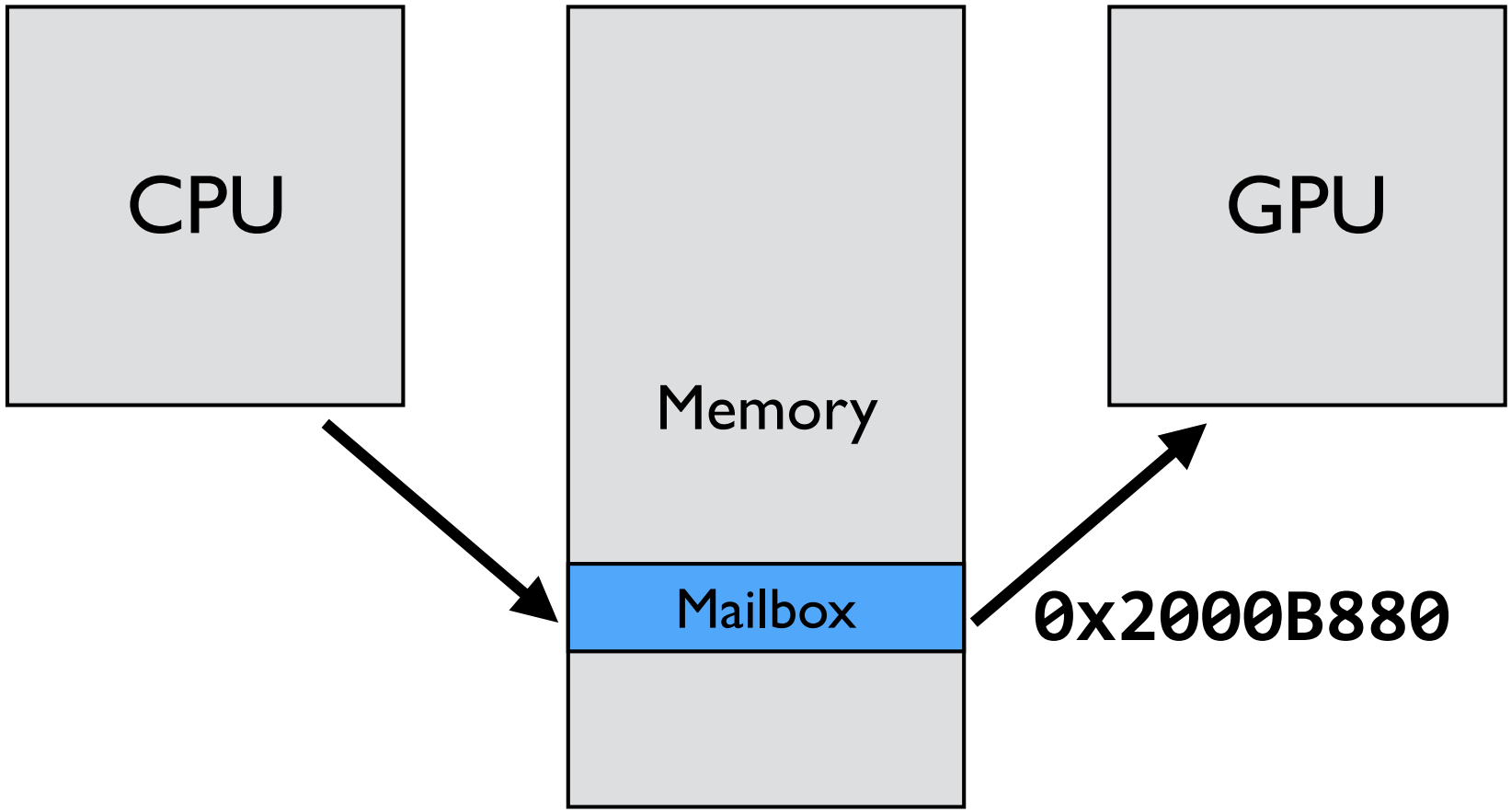


Mailbox



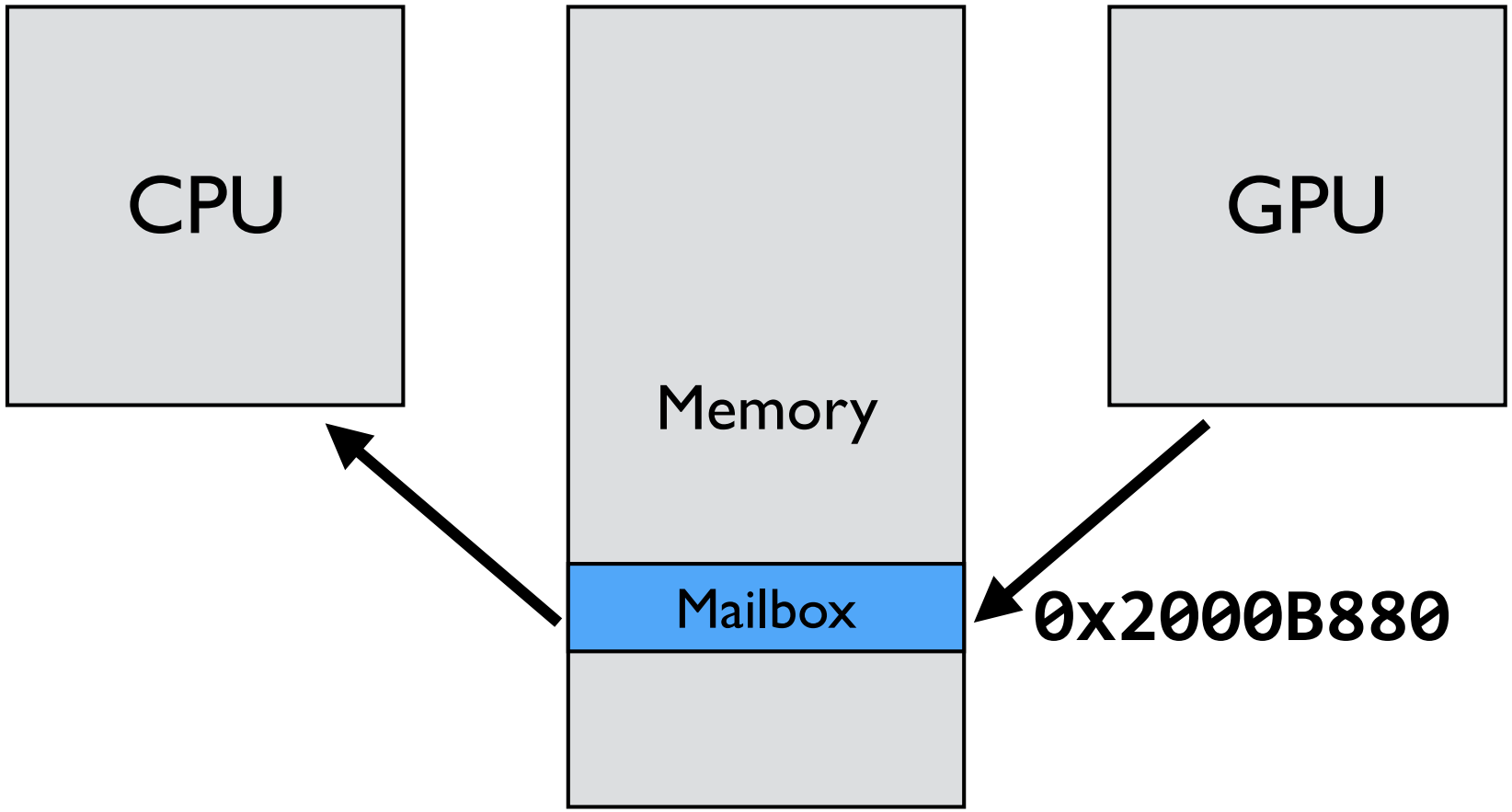
code/clear/mailbox.c

CPU "Mails" Message to GPU



code/fb/mailbox.c

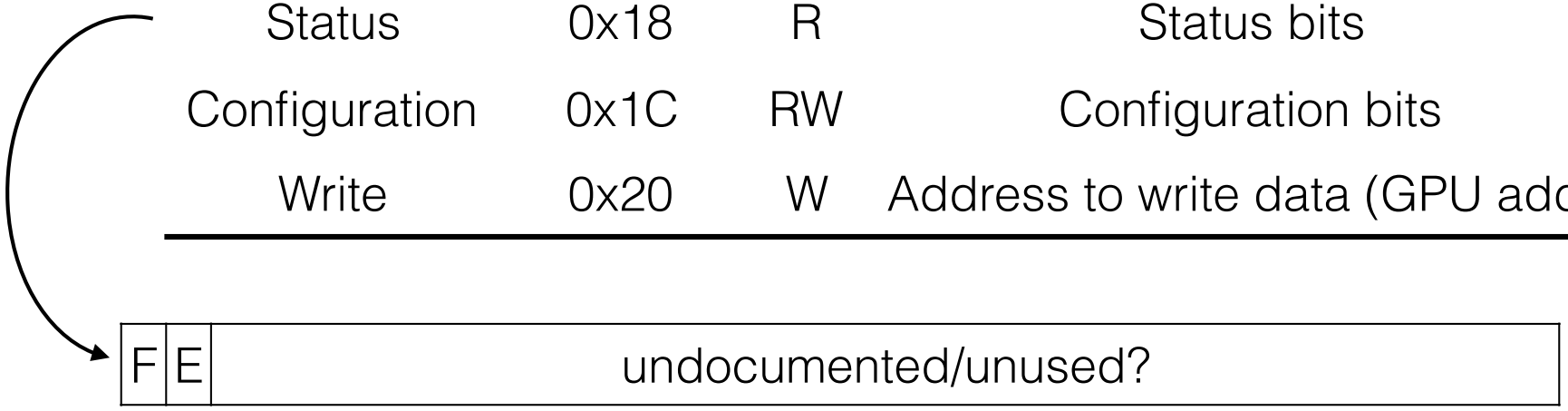
GPU Mails Reply to CPU



code/fb/mailbox.c

Mailbox Format

Register	Offset	R/W	Use
Read	0x00	R	Destructively read value
Peek	0x10	R	Read without removing data
Sender	0x14	R	Sender ID (bottom 2 bits)
Status	0x18	R	Status bits
Configuration	0x1C	RW	Configuration bits
Write	0x20	W	Address to write data (GPU addr)

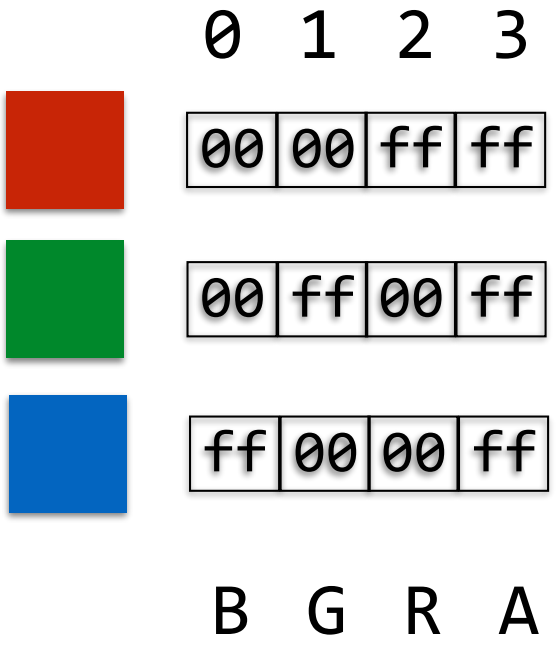


F = Full
E = Empty

code/fb/mailbox.c

RGBA (BGRA) Pixel/Color

RGBA (BGRA) pixels are four bytes

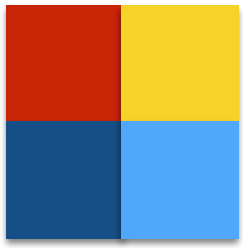


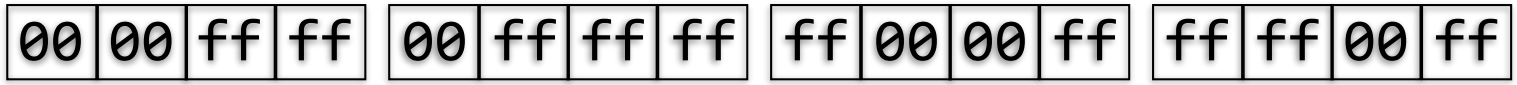
Beware: blue is the first byte (lowest address)

1D Array of `unsigned char`

The framebuffer is an array

```
unsigned char fb[2*2*4];
fb[0] = 0x00; // b
fb[1] = 0x00; // g
fb[2] = 0xff; // r
fb[3] = 0xff; // a
```



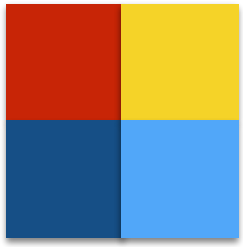


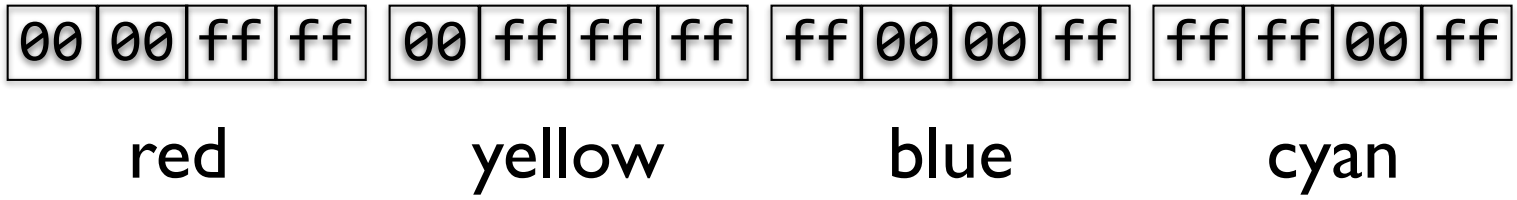
red yellow blue cyan

Note: (0,0) is at the upper left corner of the monitor

1D Array of `unsigned char`

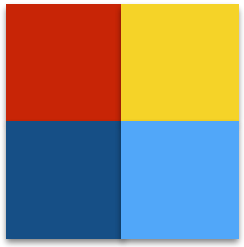
```
unsigned char fb[2*2*4];
fb[4*(2*y + x) + brga] = …
brga = 0, 1, 2, or 3
```





1D Array of unsigned char

```
#define HEIGHT 2
#define WIDTH 2
#define DEPTH 4
unsigned char fb[HEIGHT*WIDTH*DEPTH];
fb[DEPTH*(WIDTH*y + x) + bgra] = …
```



1D Array of unsigned

```
#define HEIGHT 2
#define WIDTH 2
#define DEPTH 4
unsigned char fb[HEIGHT*WIDTH*DEPTH];
fb[DEPTH*(WIDTH*y + x) + bgra] = …

unsigned fb[HEIGHT*WIDTH];
fb[0] = 0xffff0000; // x=0, y=0
fb[1] = 0xffffff00; // x=1, y=0
fb[2] = 0xff0000ff; // x=0, y=1
fb[3] = 0xff00ffff; // x=1, y=1
```

Drawing

`code/clear`
`code/cleari`

2D Array of unsigned

```
#define HEIGHT 2
#define WIDTH 2
#define DEPTH 4
unsigned char fb[WIDTH*HEIGHT*DEPTH];
fb[rgba + DEPTH*(x + WIDTH*y)] = …

unsigned fb[HEIGHT][WIDTH];
fb[0][0] = 0xffff0000; // x=0, y=0
fb[0][1] = 0xffffff00; // x=1, y=0
fb[1][0] = 0xff0000ff; // x=0, y=1
fb[1][0] = 0xff00ffff; // x=1, y=1
```

2D Array of unsigned

```
unsigned fb[HEIGHT][WIDTH];
fb[y][x] = …

unsigned (*fb)[WIDTH] =
    (unsigned (*)[WIDTH])frame;
fb[0][0] = 0xffff0000; // x=0, y=0
fb[0][1] = 0xffffff00; // x=1, y=0
fb[1][0] = 0xff0000ff; // x=0, y=1
fb[1][0] = 0xff00ffff; // x=1, y=1
```

What is `unsigned *fb[2], (*fb)[2]`**?**

Demo

`code/grid`

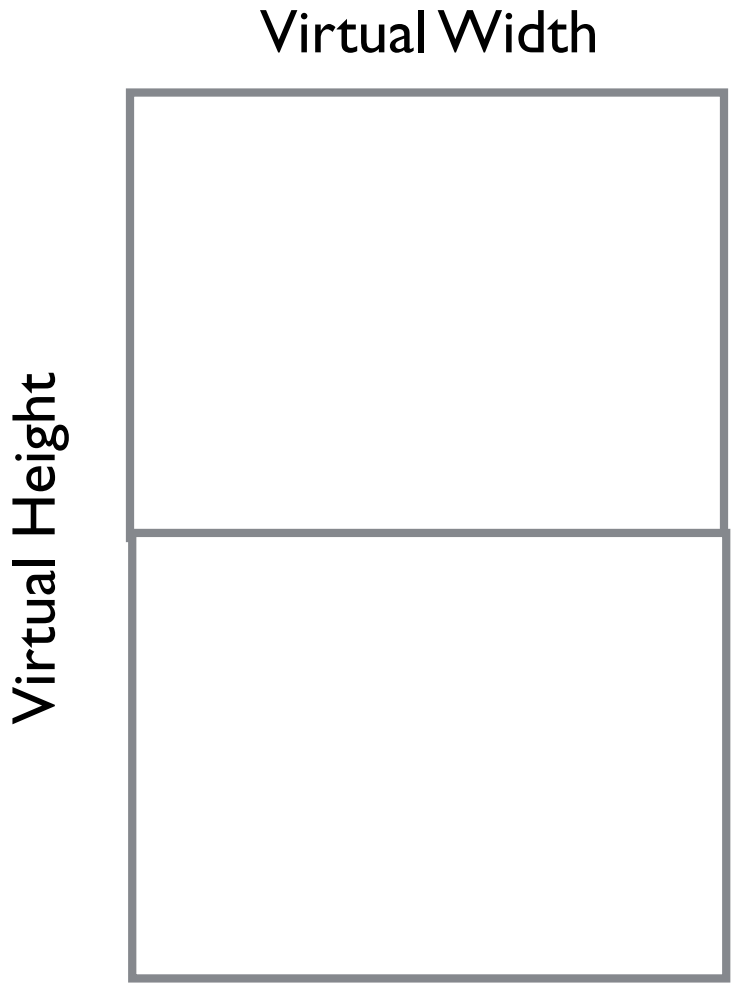
Double-Buffering

code/singlebuffer
code/doublebuffer

Double Buffer

Double buffering:

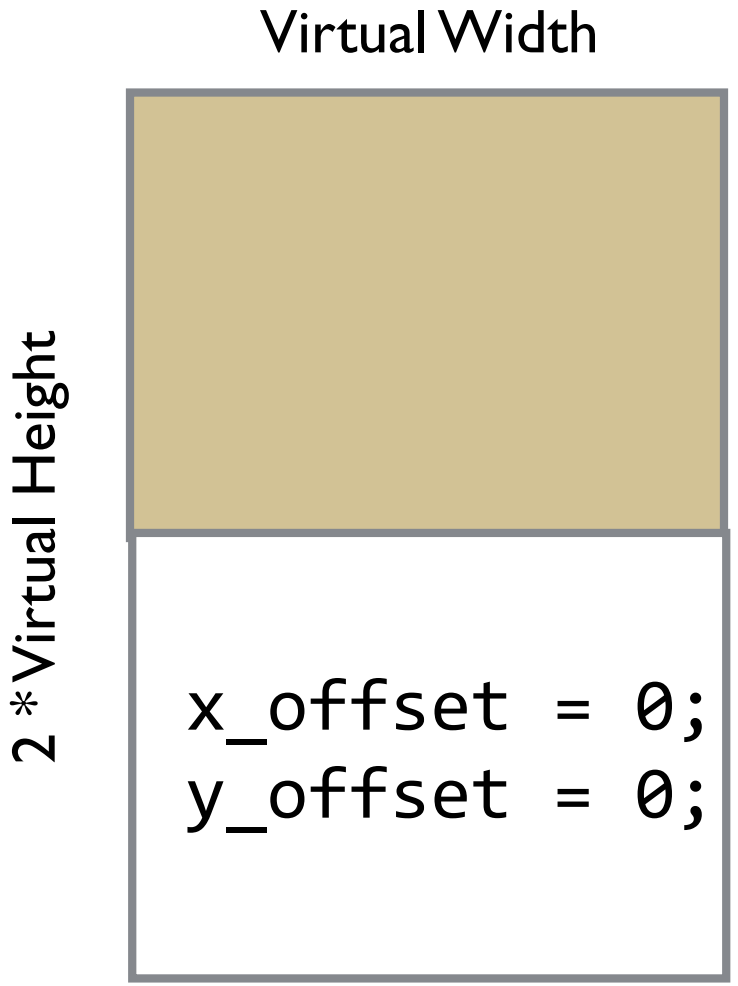
- **Display the "front"-buffer**
- **Draw into the "back"-buffer**
- **Swap front and back when you are done drawing**
- **Requires 2 frame buffers**



Display Top Buffer

Double buffering:

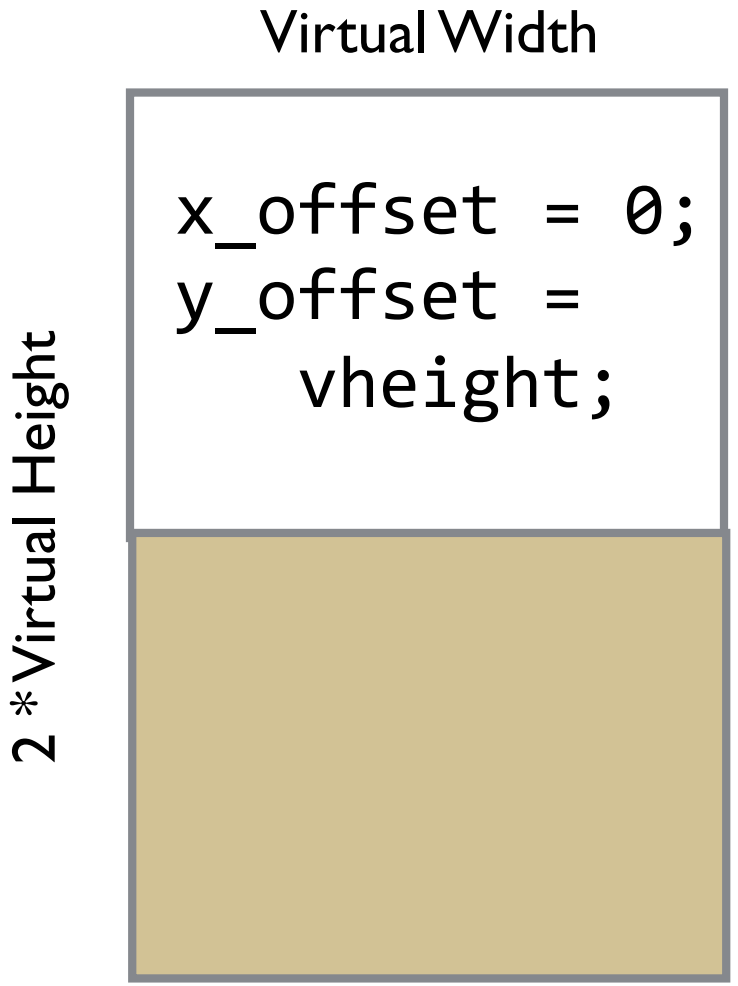
- **Display the "front"-buffer**
- **Draw into the "back"-buffer**
- **Swap front and back when you are done drawing**
- **Requires 2 frame buffers**



Display Bottom Buffer

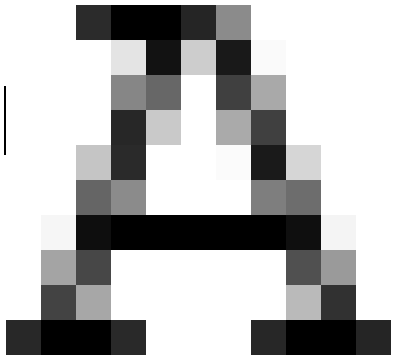
Double buffering:

- **Display the "front"-buffer**
- **Draw into the "back"-buffer**
- **Swap front and back when you are done drawing**
- **Requires 2 frame buffers**



Drawing Text

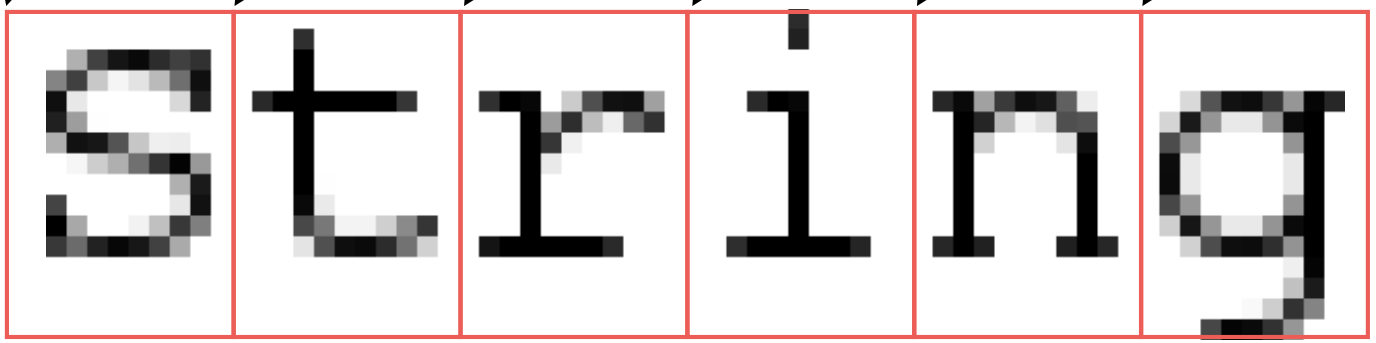
Fonts: monospaced vs. proportiona



Font a set of "glyphs"

```
 !"#$%&'( )*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
`abcdefghijklmnopqrstuvwxyz{|}~
```

x,y x+w,y x+2w,yx+3w,yx+4w,yx+5w,y



Framebuffer Overview

GPU refreshes the display using a framebuffer

The size of the image sent to the monitor is called the physical size

The size of the framebuffer image in memory is called the virtual size

The CPU and GPU share the memory, and hence the frame buffer

The CPU and GPU exchange messages using a mailbox

Lab 6 and Assignment 6

Lab

- **Understand mailbox and fb code**
- **First steps towards a graphics library**

Assignment

- **Complete fb code (including double buffering)**
- **Implement graphics library**
- **Draw fonts**
- **Implement console**
- **Port your shell to use the console**