Graphics and Framebuffers

Response to Mid-Quarter Feedback

Thanks for all the feedback!

Some changes

- **Tweak OH schedule; dual OHs; move evenly divide time between students**
- **Will try hard to limit labs to 2 hours**
- **Better preview of the upcoming assignment in lecture**

gpio
timer
uart
printf
malloc
keyboard
fb
gl
console
shell

Baremetal on the Pi

Raspberry Pi A+

ARM processor and memory

Peripherals: GPIO, timers, UART (gpio, uart), keyboard

Assembly and machine language (as)

C and pointers (gcc)

Functions and the stack (gdb)

Serial communication and strings (uart, printf)

Linking and the memory map (ld, memmap, objcopy)

Loading using the bootloader (rpi-install.py, bootloader)

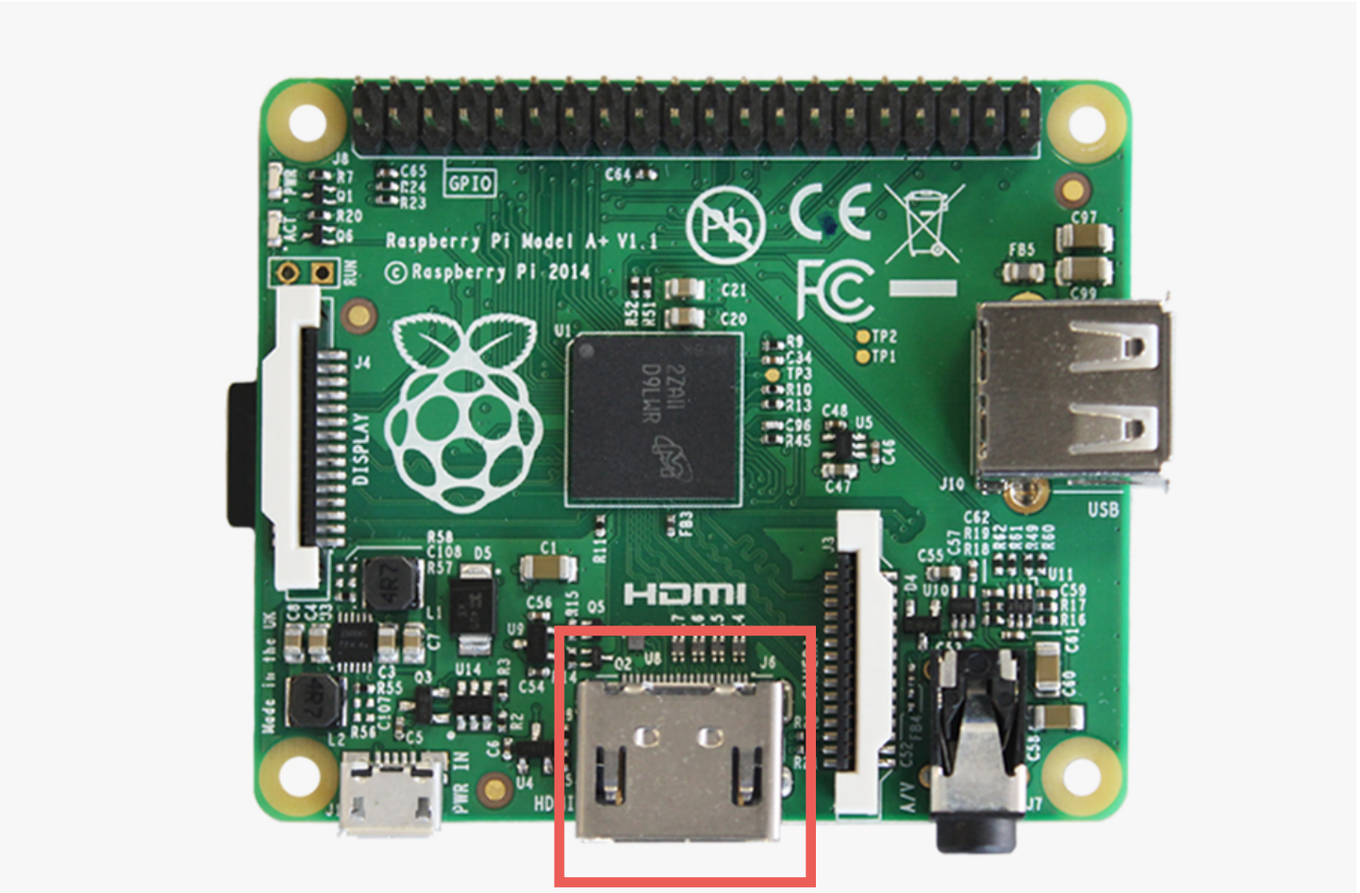
Starting (start.s, cstart.c)

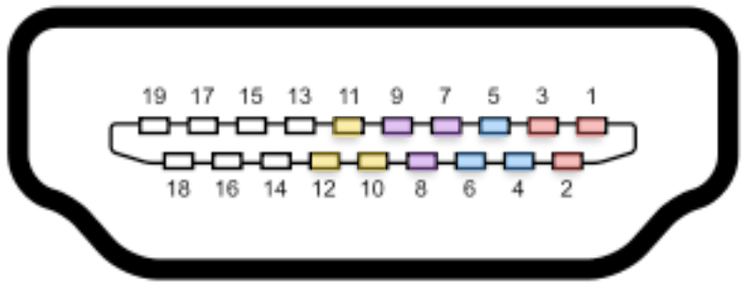
Memory managements

Tools (git, bash, make, brew)

The Force Awakens in You

HDMI





- Clock
- Data 0
- Data 1
- Data 2
- Control

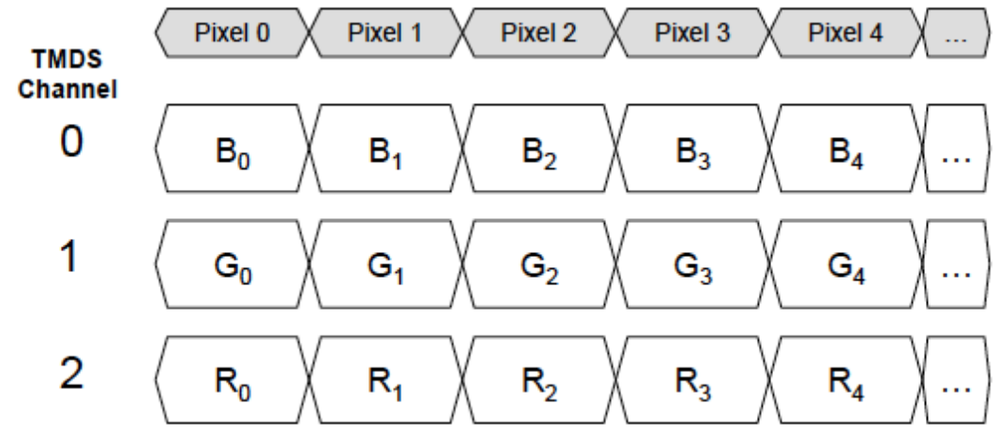
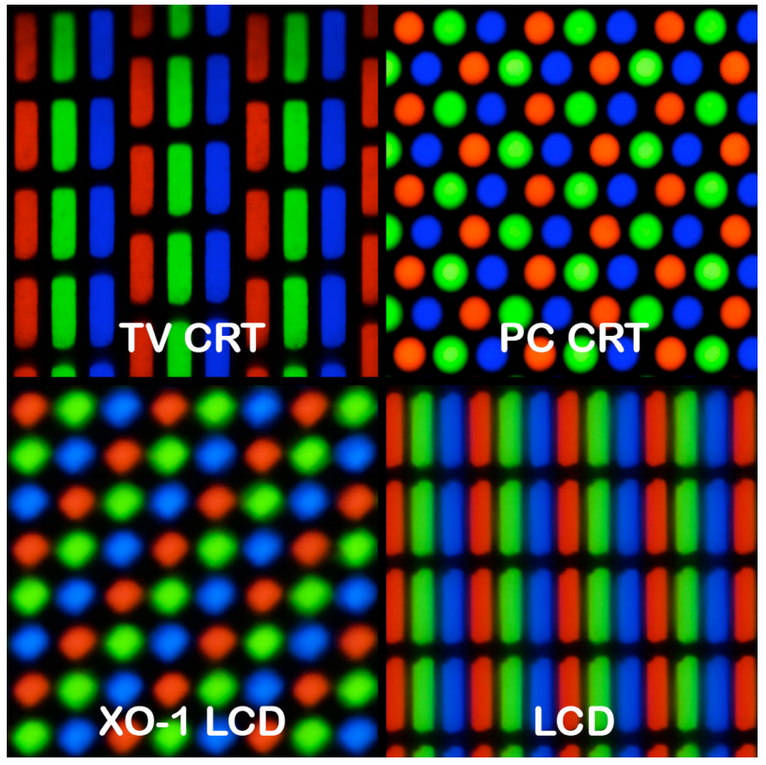


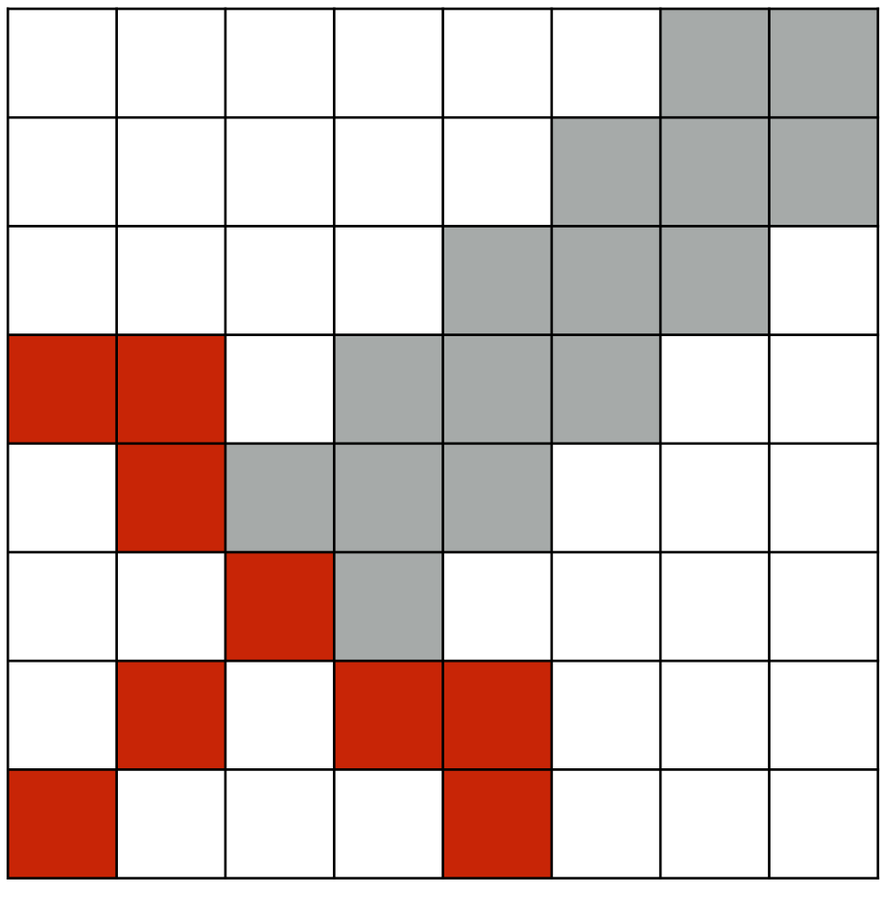
Figure 6-1 Default pixel encoding: RGB 4:4:4, 8 bits/component

Figure from High-Definition Multimedia Interface Specification Version 1.3a

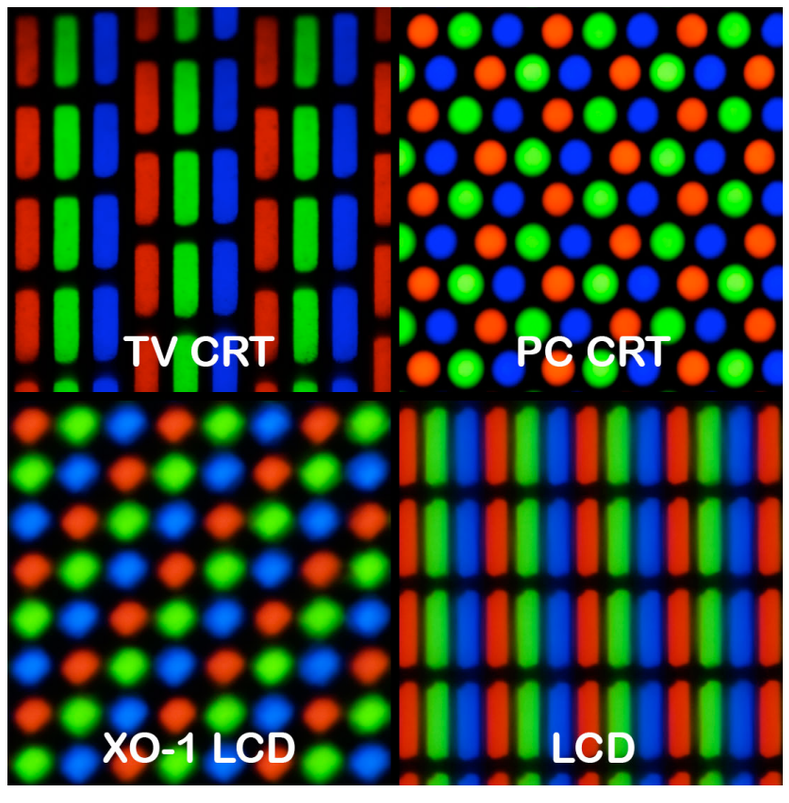


Displays

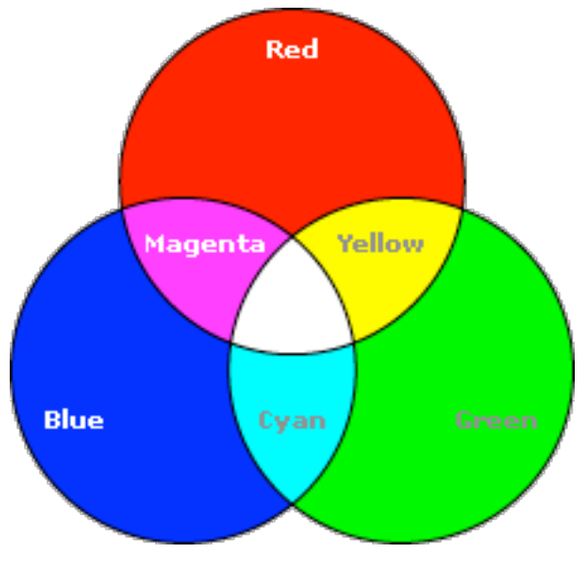
Pixels

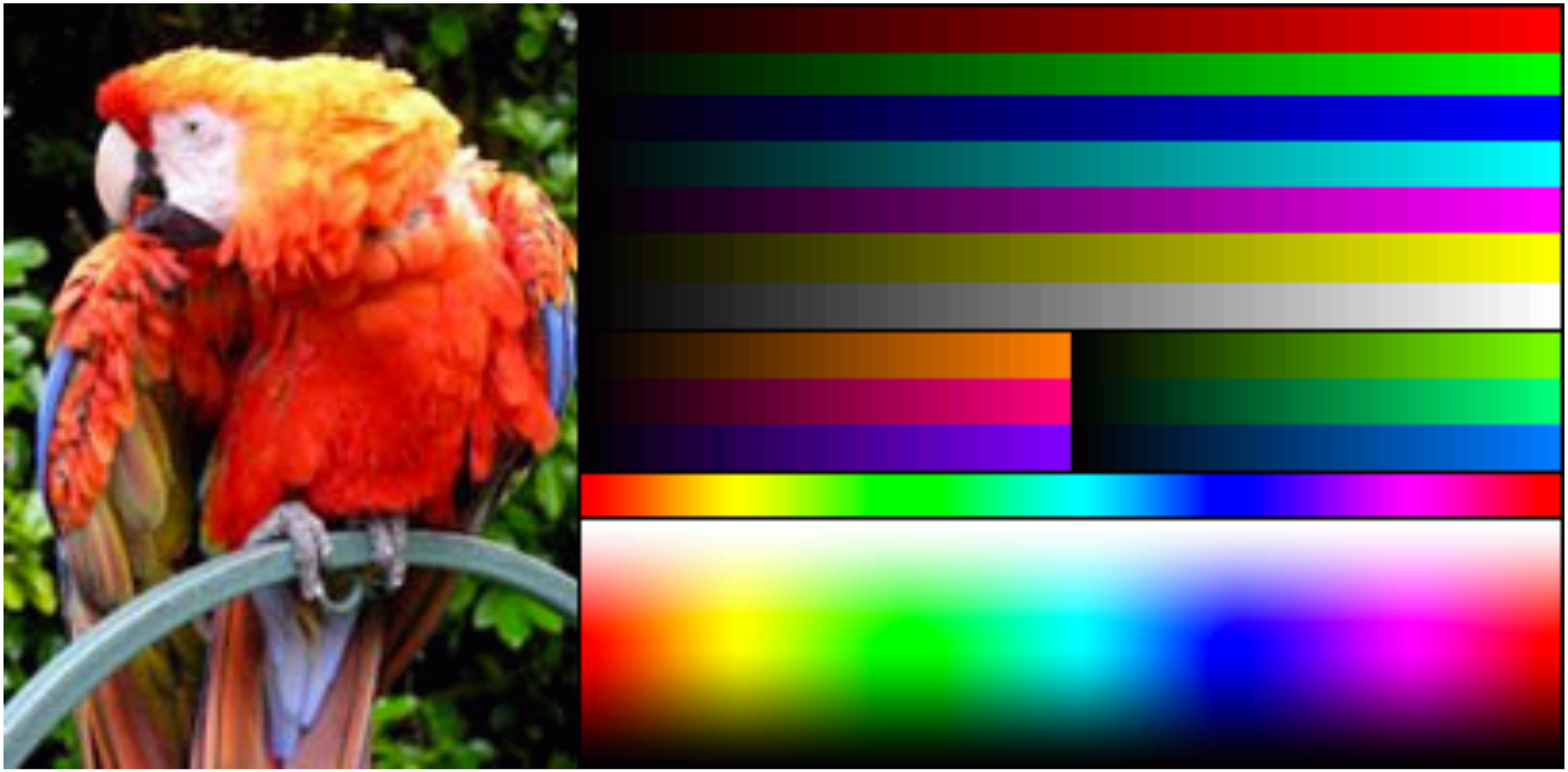


Displays



Light





The framebuffer is an image

An image is a 2D array of pixels

RGBA pixel (depth=32 bits)

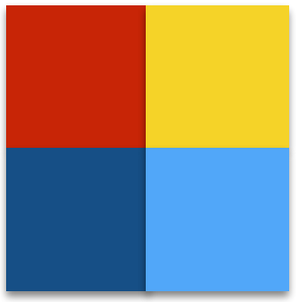
Red = 8 bits
Green = 8 bits
Blue = 8 bits
Alpha = 8 bits

Framebuffer Resolution

read physical size
read virtual size
read pixel depth

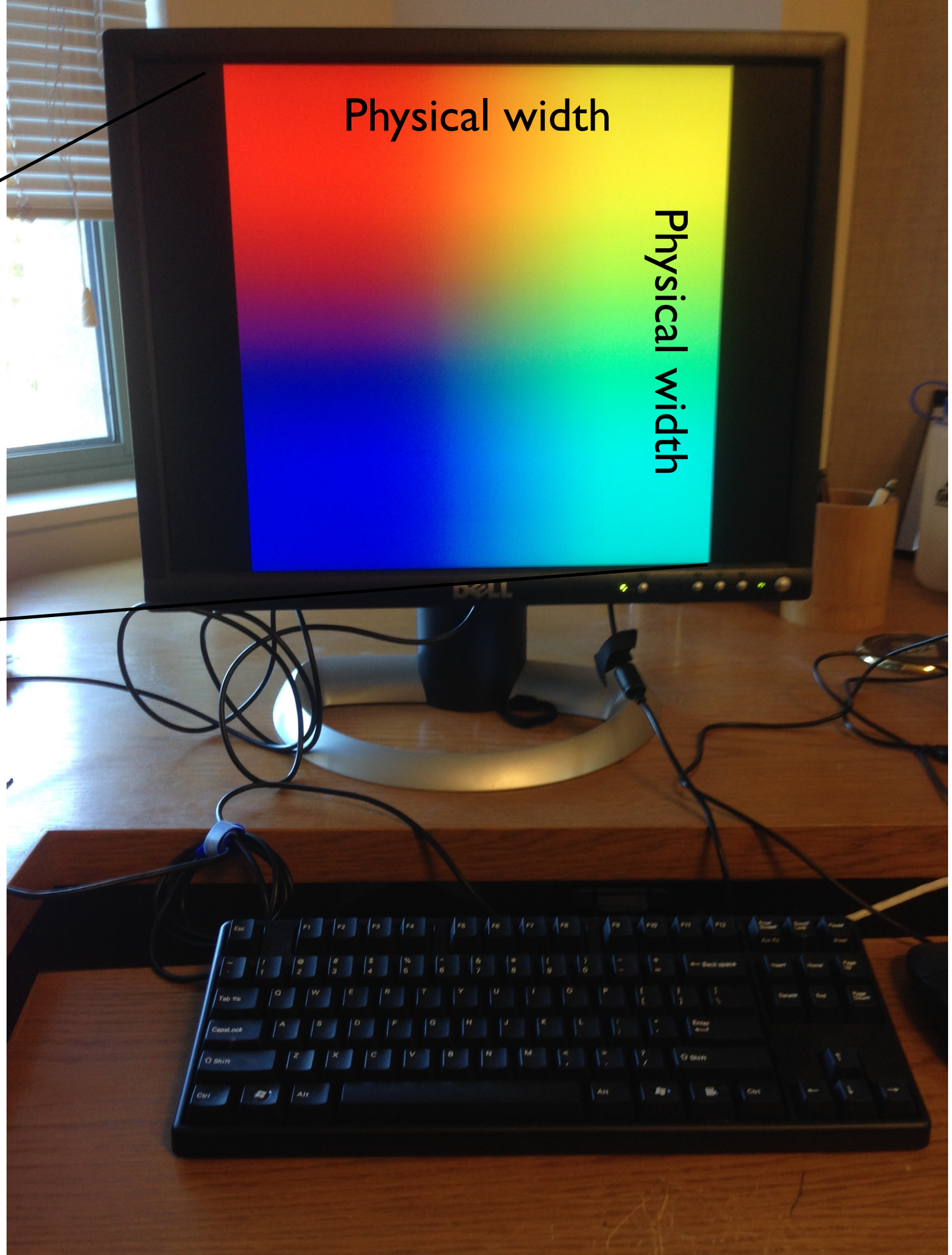
code/video

Virtual height = 2



Virtual width = 2

2x2 image is interpolated to 1600x1200 to fill monitor



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

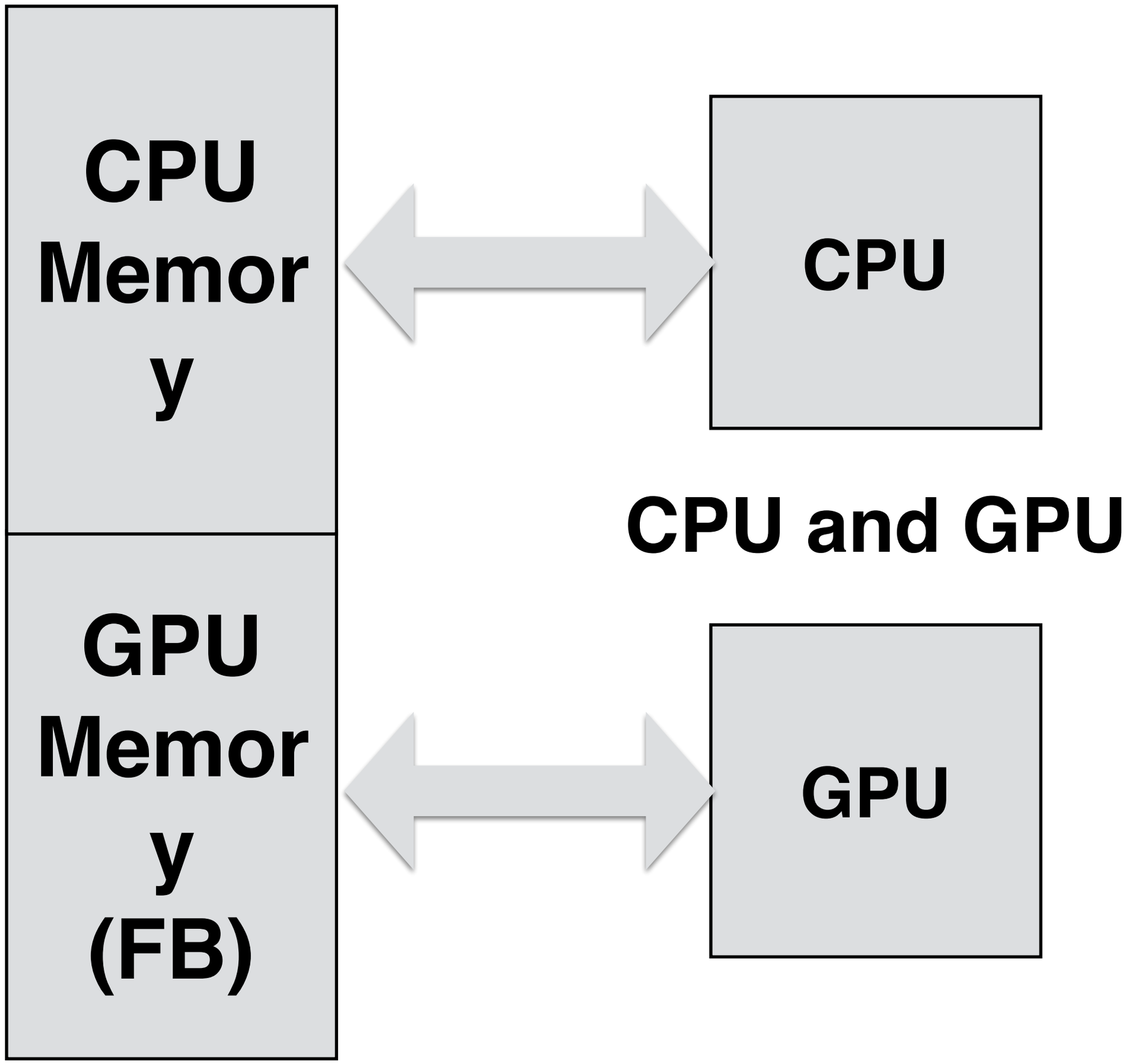
code/fb

Shared Memory

frame buffer memory

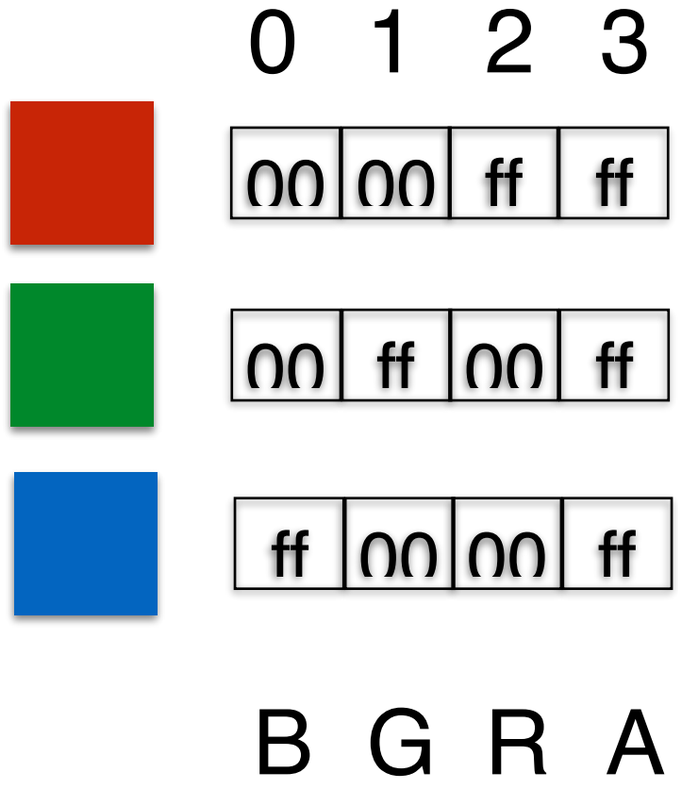
split between cpu & cpu

code/video


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

RGBA (BGRA) Pixel/Color

RGBA (BGRA) pixels are four bytes

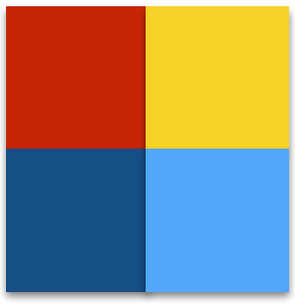


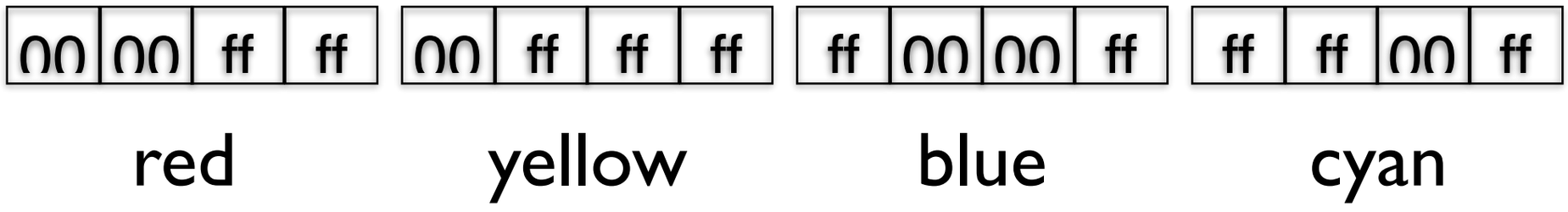
Beware: blue is the first byte (lowest address)

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
```

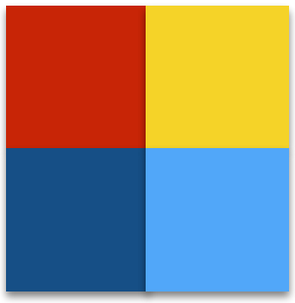


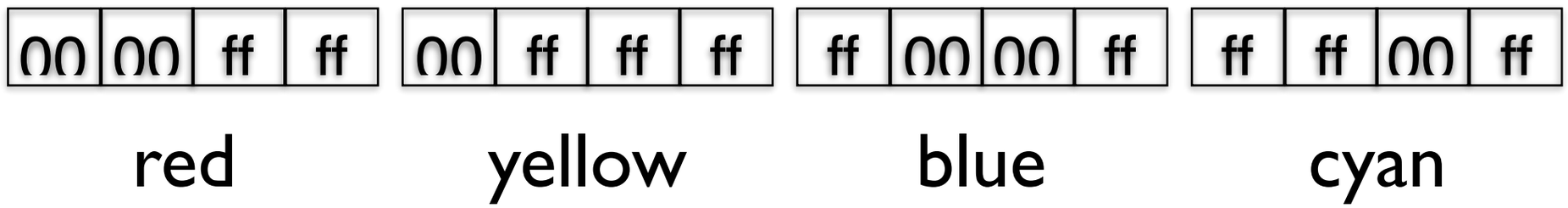


Note: (0,0) is at the upper left corner of the monitor

Array of unsigned char

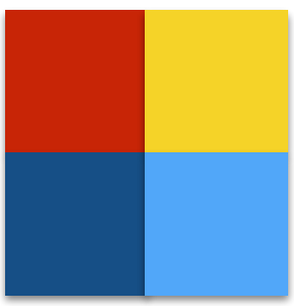
unsigned char fb[2*2*4];
fb[bgra + 4*(x + 2*y)] = …
bra = 0, 1, 2, or 3





Array of unsigned char

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



Array of unsigned

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

unsigned fb[WIDTH*HEIGHT];
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

2D Array of unsigned

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

unsigned (*fb)[2] = (unsigned (*)[2])frame;
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
```

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

Demo

code/grid

Double-Buffering

Double Buffering

Writing directly to screen can cause flickering

Solution: Double buffering

- **Two buffers: back-buffer and front-buffer**
- **Front-buffer is being display**
- **Draw into back-buffer**
- **Swap buffers to update display**

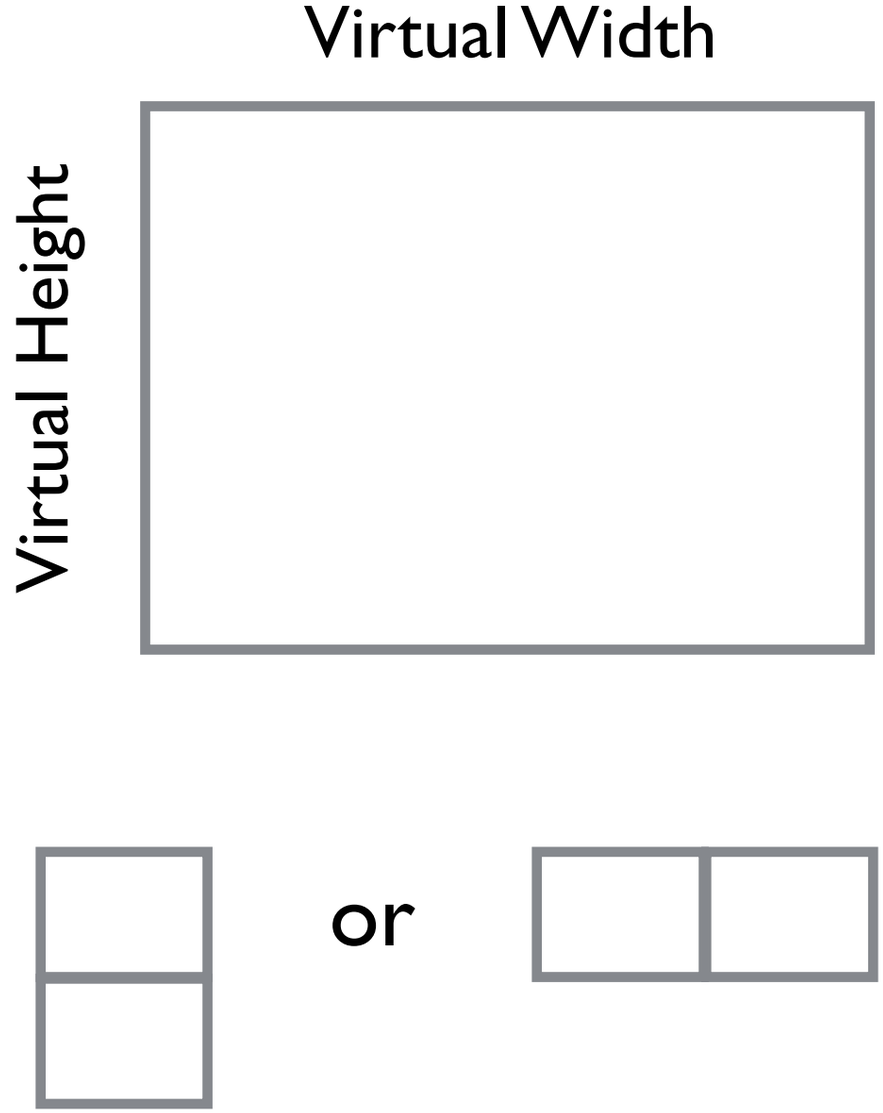
Single Buffer

Drawing directly to framebuffer lets you see the graphics as it is draw (good for debugging!)

Normally we just show the result. Don't see the drawing process

Double buffering: display "front-buffer" while drawing into "back"-buffer. Swap buffers when you are done drawing.

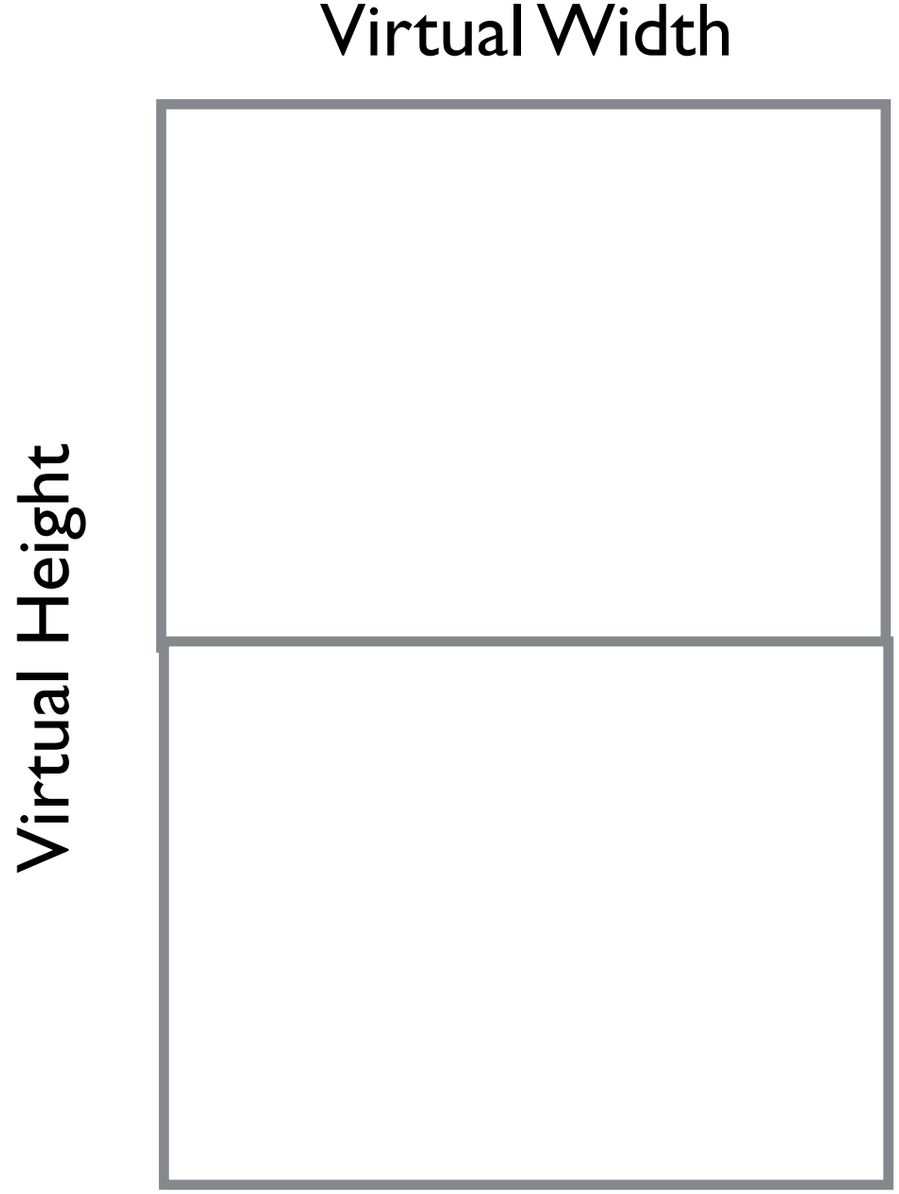
Which arrangement is better?



Double Buffer

Double buffering:

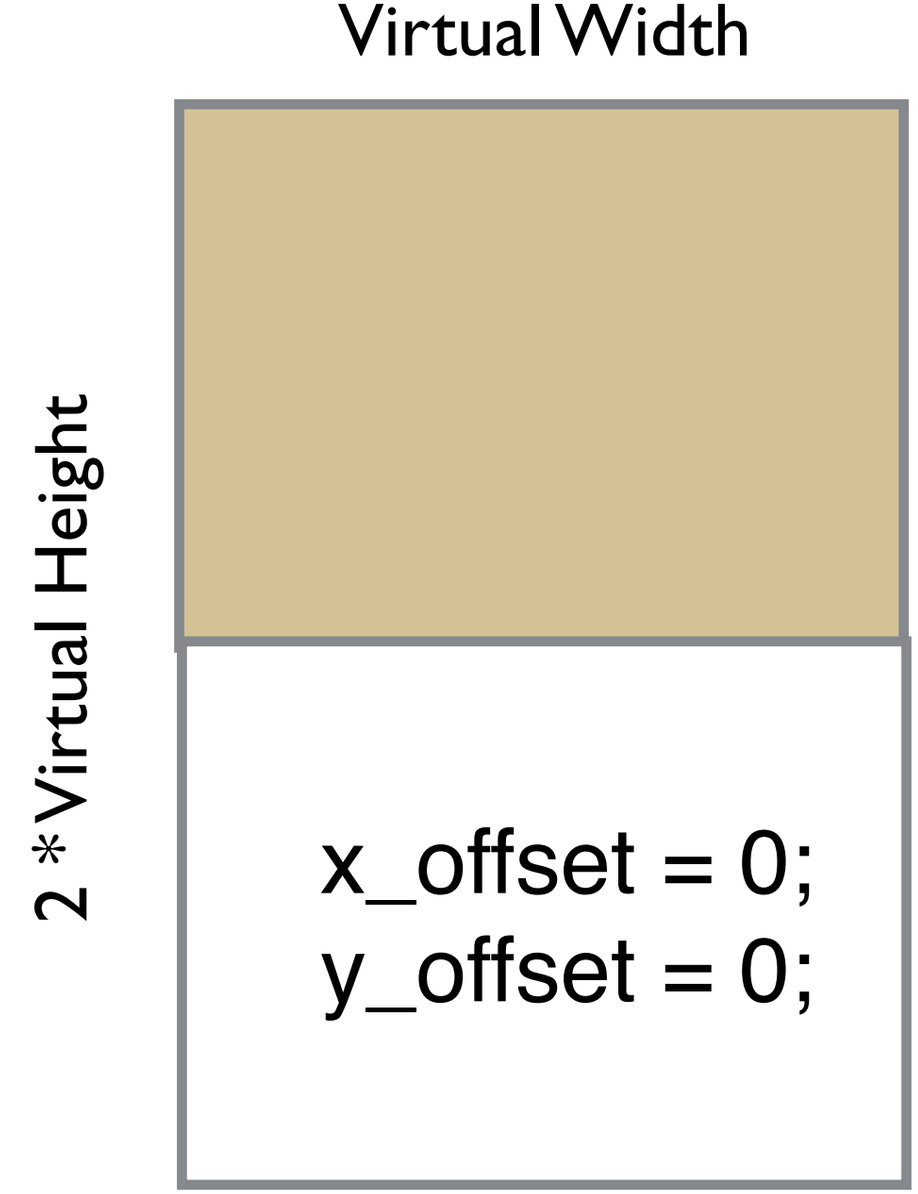
- **Display the "front"-buffer**
- **Draw into the "back"-buffer**
- **Swap front and back when you are done drawing**
- **Requires 2 frame buffers**



Display Top Buffer

Double buffering:

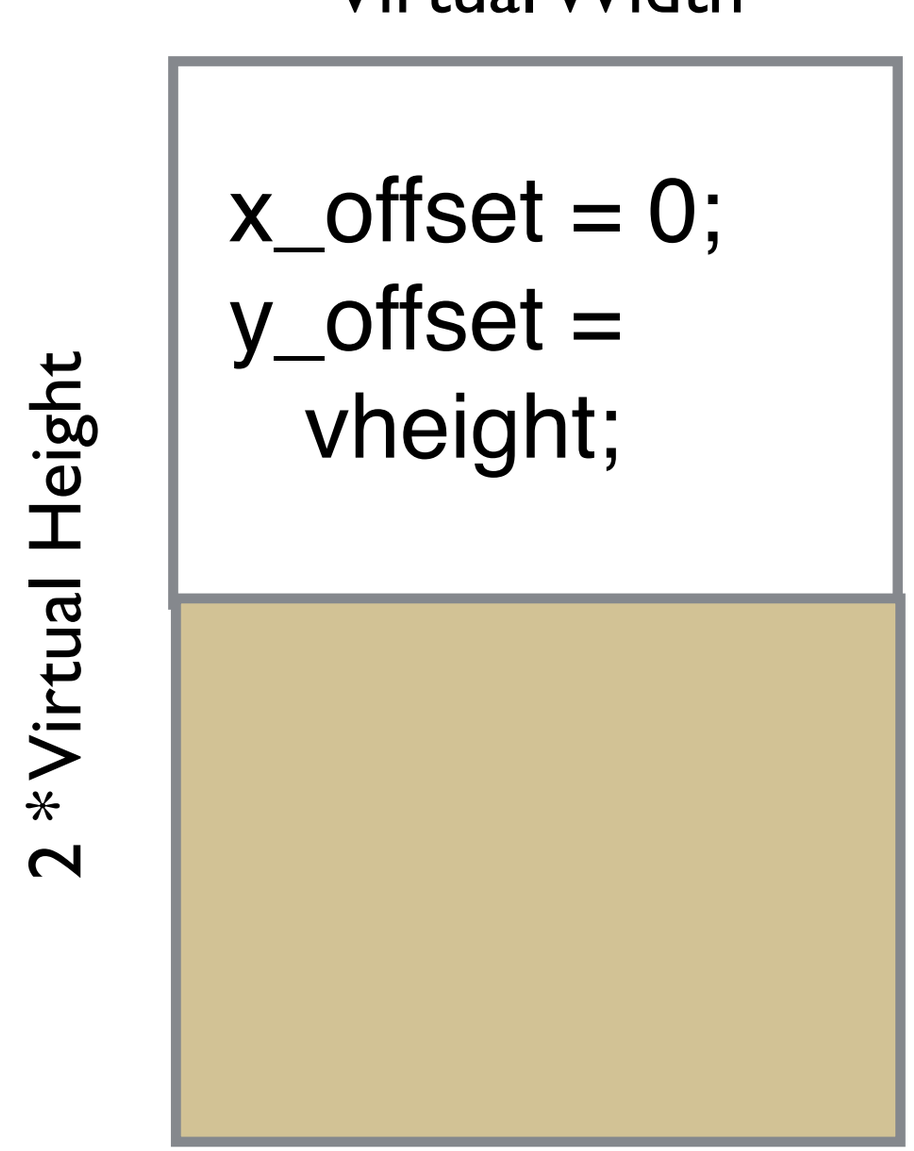
- **Display the "front"-buffer**
- **Draw into the "back"-buffer**
- **Swap front and back when you are done drawing**
- **Requires 2 frame buffers**



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**



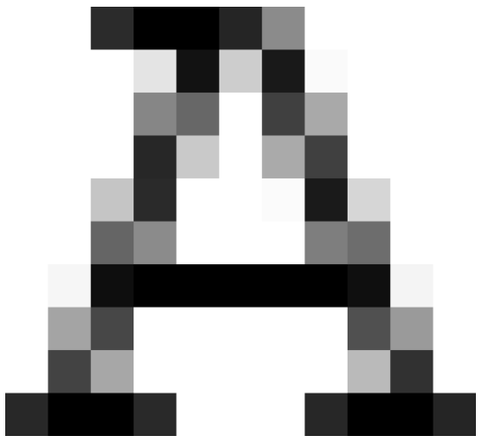
code/singlebuffer

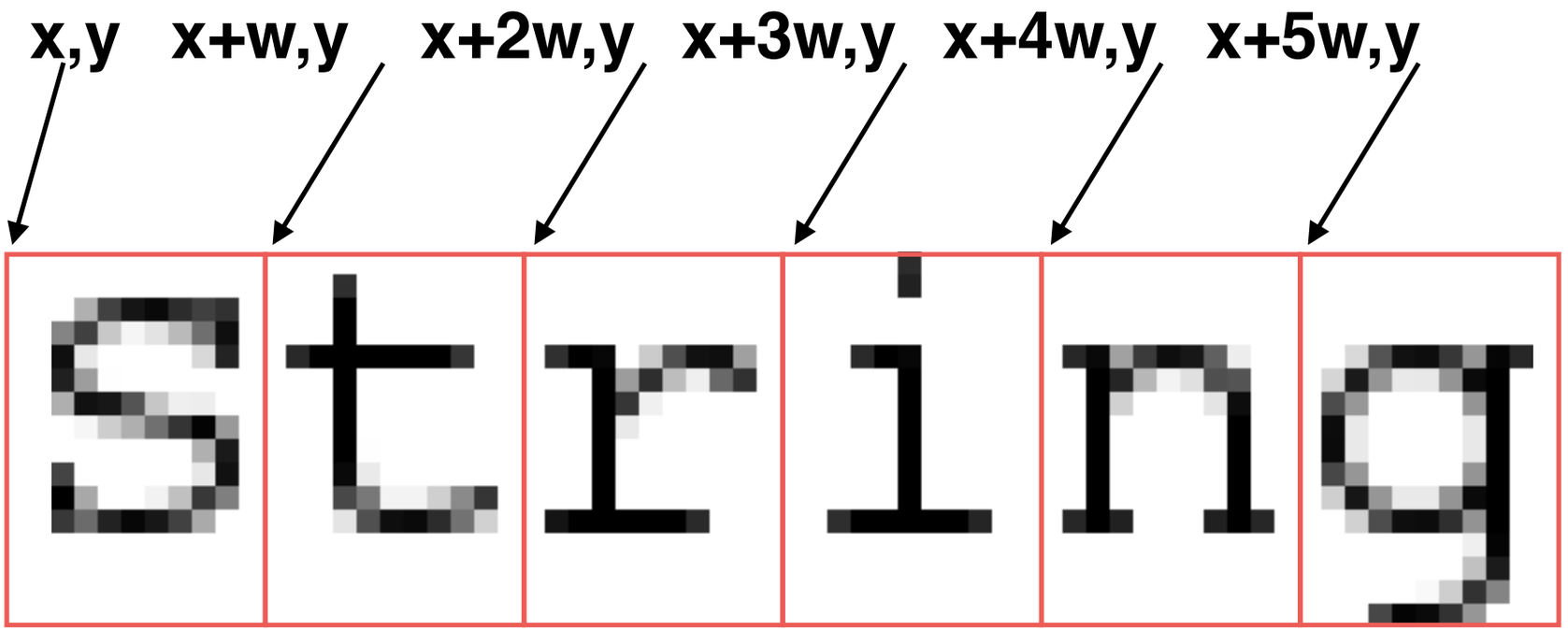
code/doublebuffer

Drawing Text

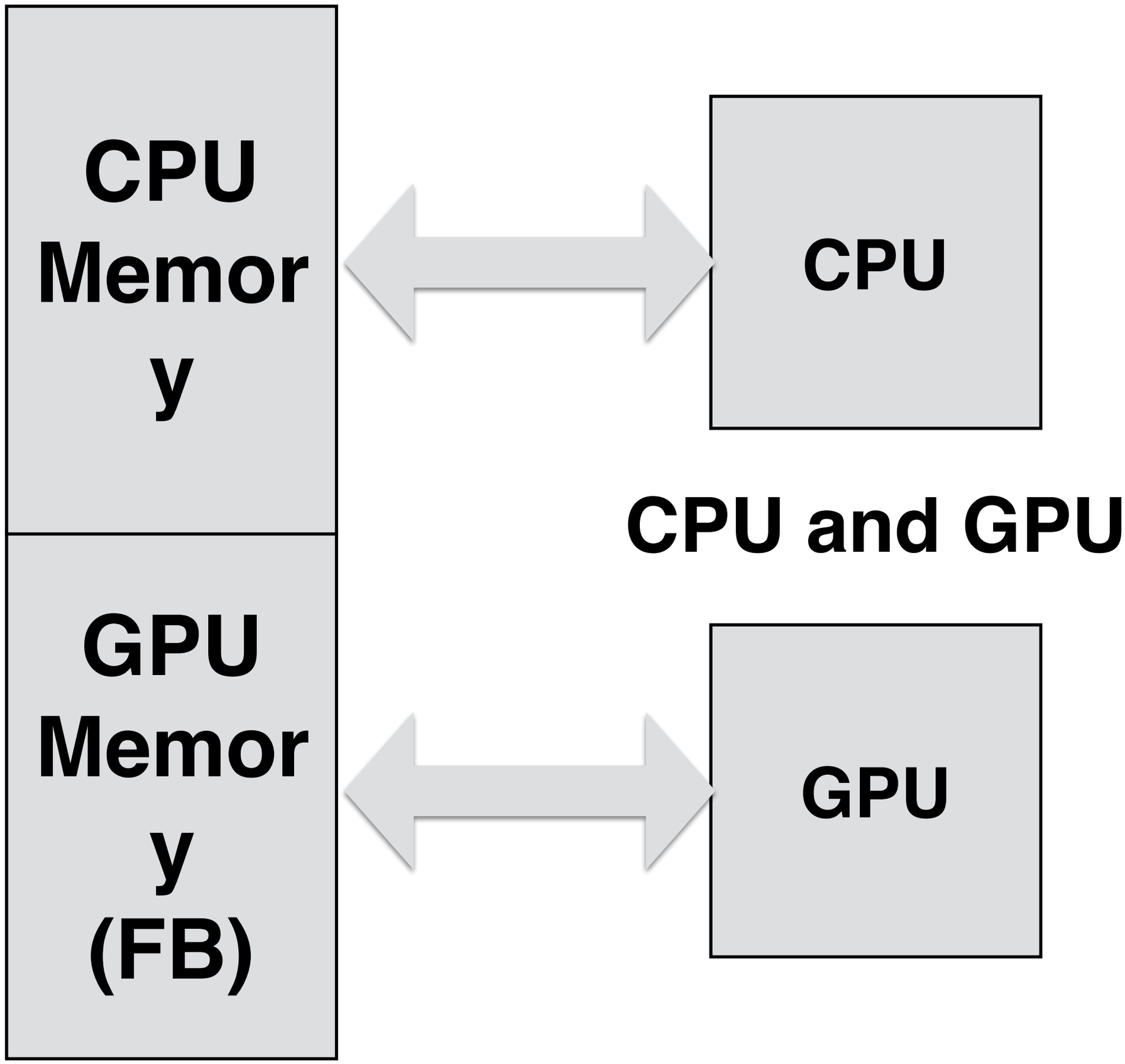
Fonts: monospaced vs. proportional

Font a set of "glyphs"

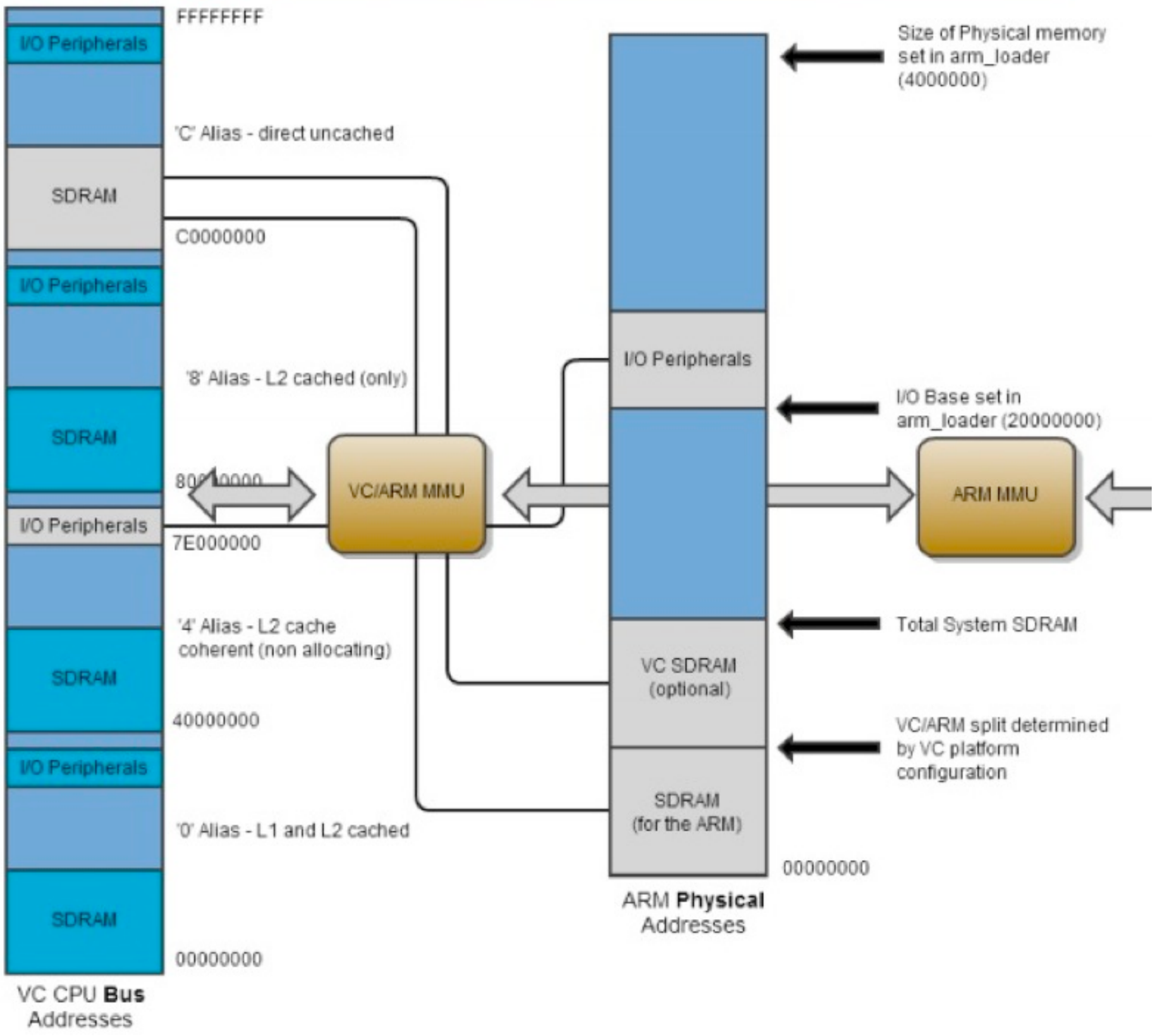




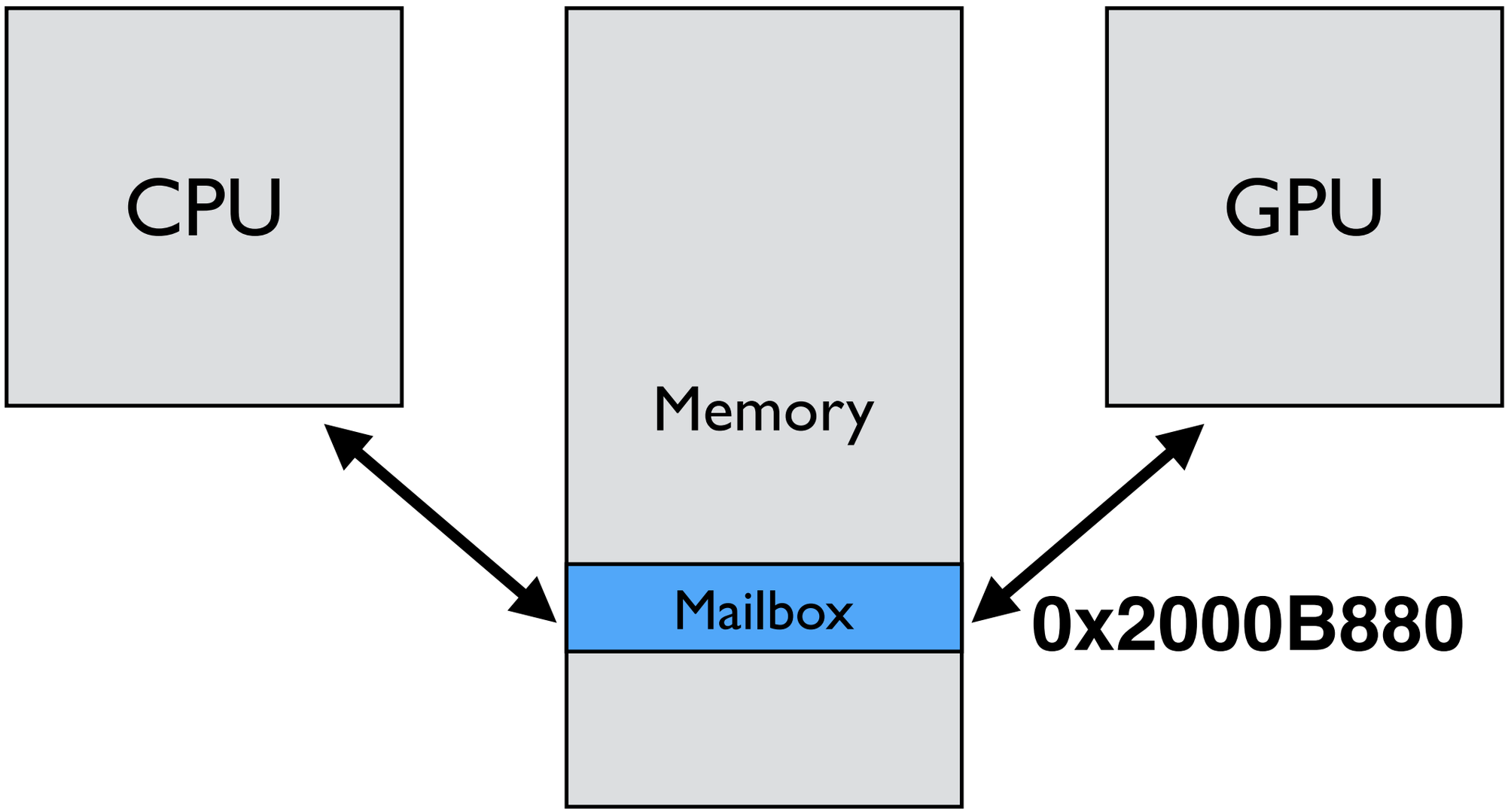
Mailbox


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

BCM2835 ARM Peripherals

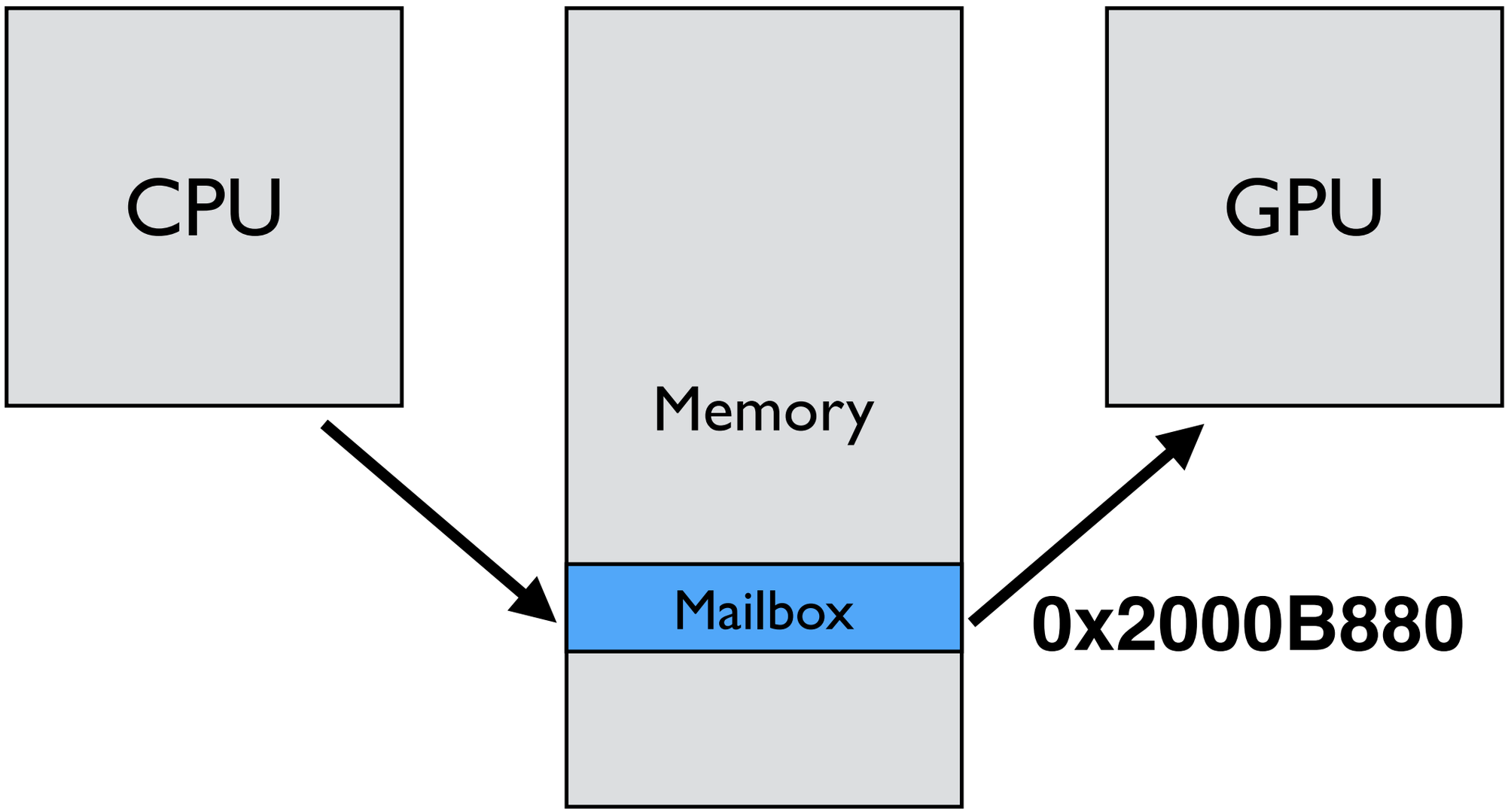


Mailbox



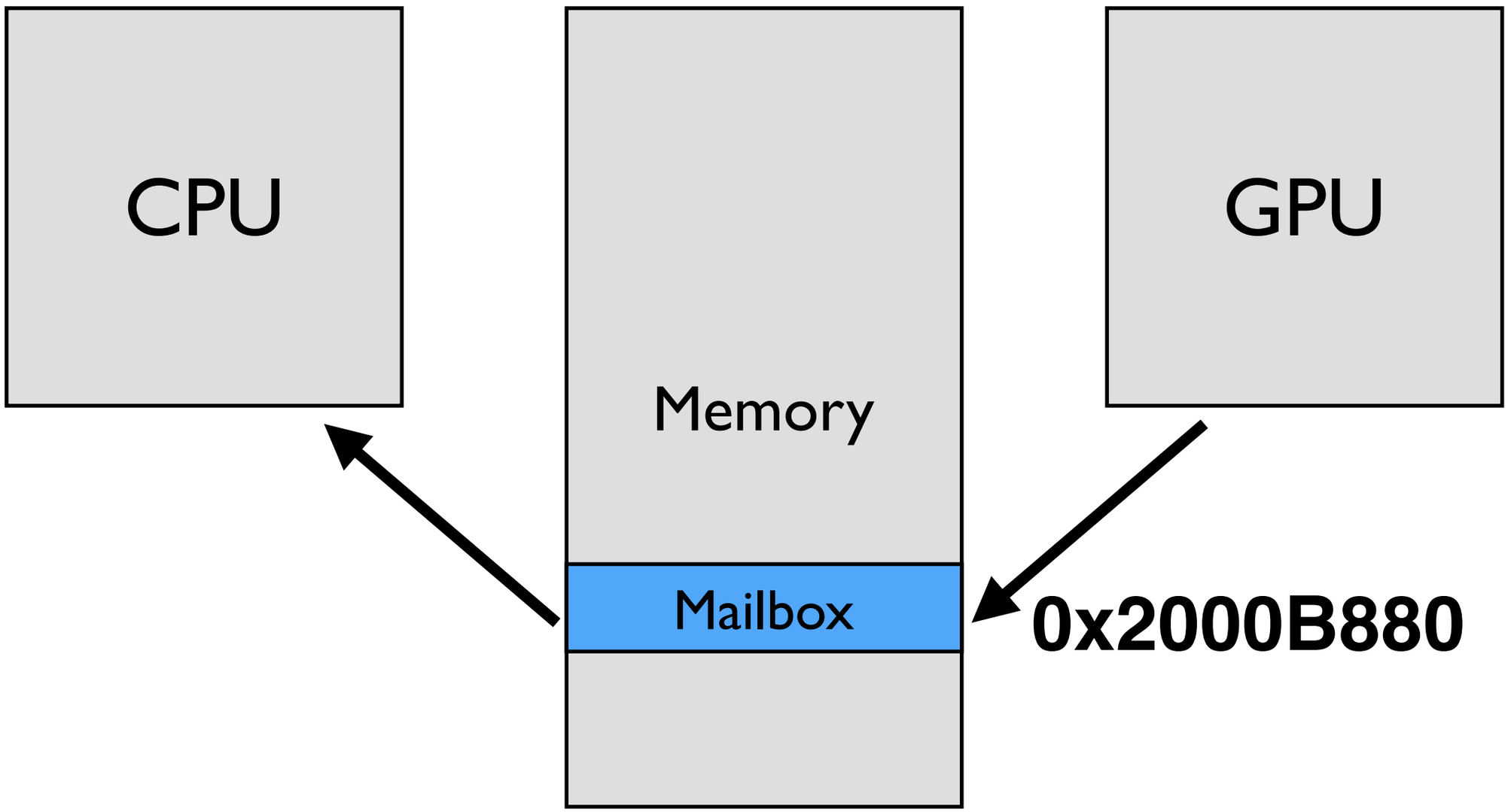
`code/clear/mailbox.c`

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`

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