Baremetal on the Pi

Raspberry Pi A+

ARM processor and memory

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

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 (rbi-install.py, bootloader)

Starting (start.s, cstart.c)

Tools (git, bash, make, brew)

The Force Awakens in You

Nest thermostat

Building a Personal Computer

In the next 3 weeks you will start to make a full-fledged personal computer.

Approach: Connecting your CPU to the graphics processor (GPU) and keyboard

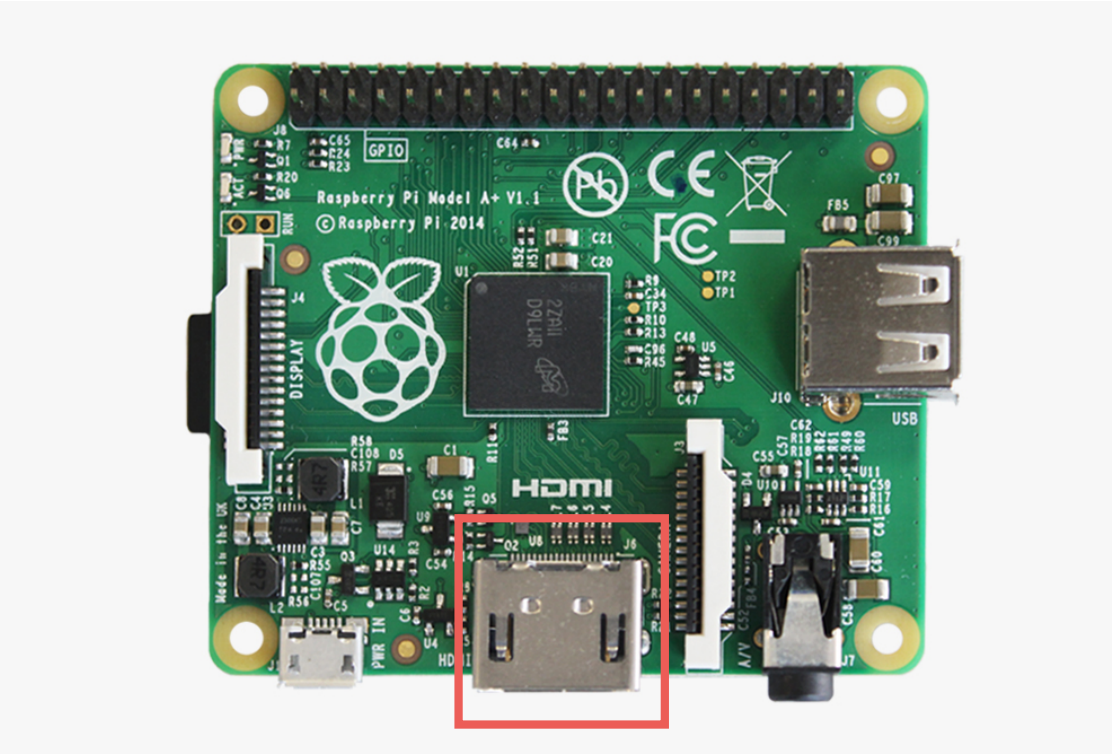
Goal: a command console, you can type in commands, and see them on a display.

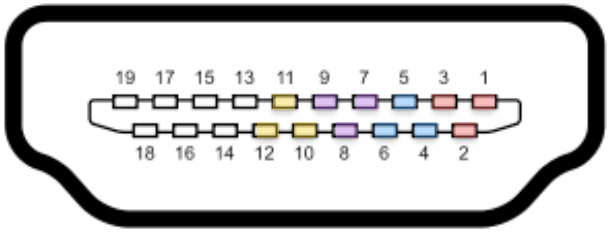
Schedule

Fri	Framebuffer	
Mon	Performance	
Lab/assignment		Console
Fri	Keyboard	
Mon	holiday	
Lab/assignment		Keyboard / Console
Fri	Interrupts	
Mon	Interrupts	
Lab/assignment		Fast Console

Graphics and Framebuffers

HDMI





Clock
Data 0
Data 1
Data 2
Control

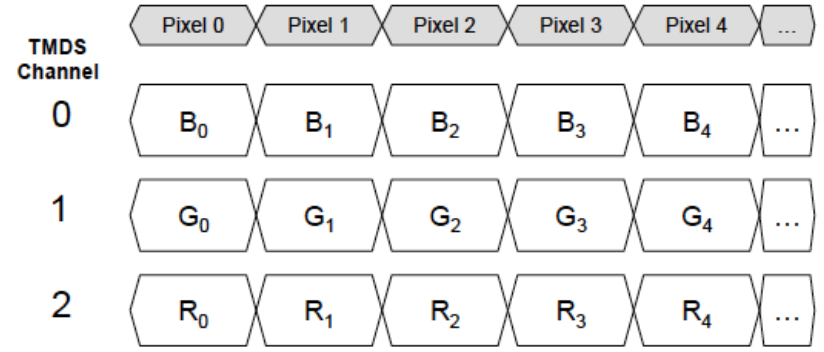
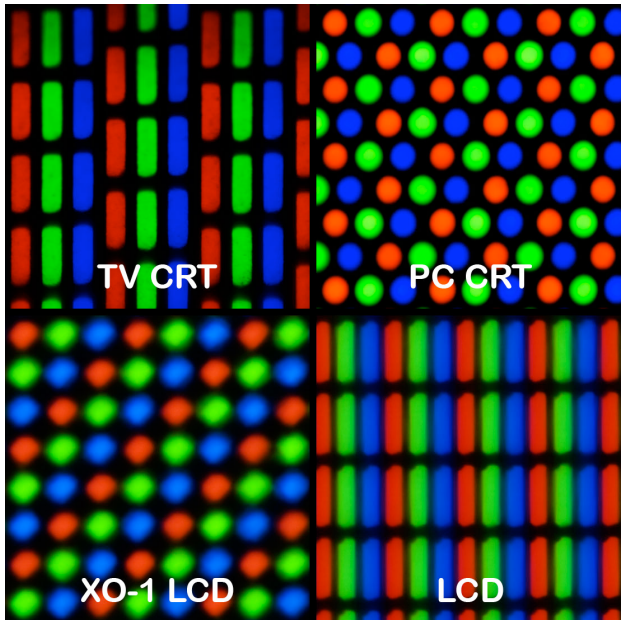


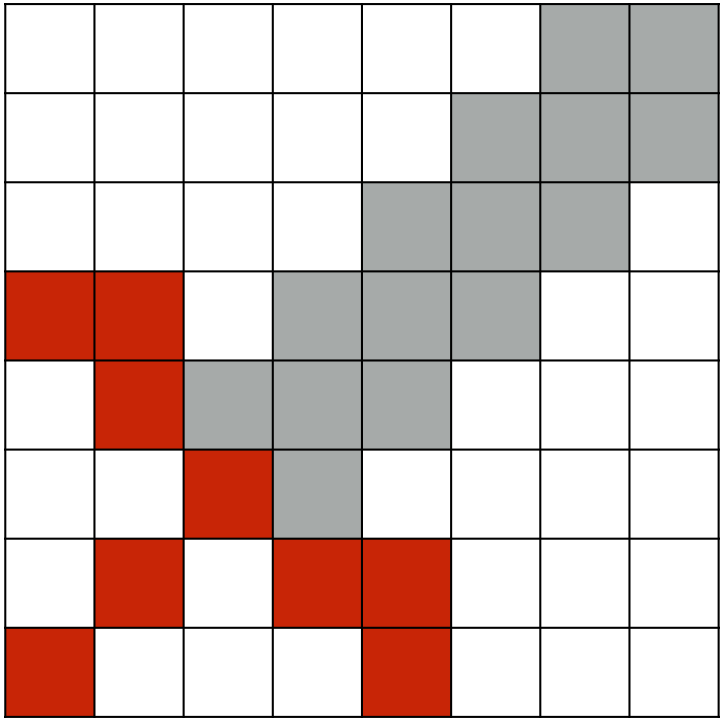
Figure 6-1 Default pixel encoding: RGB 4:4:4, 8 bits/component

Figure from High-Definition Multimedia Interface Specification Version 1.3a

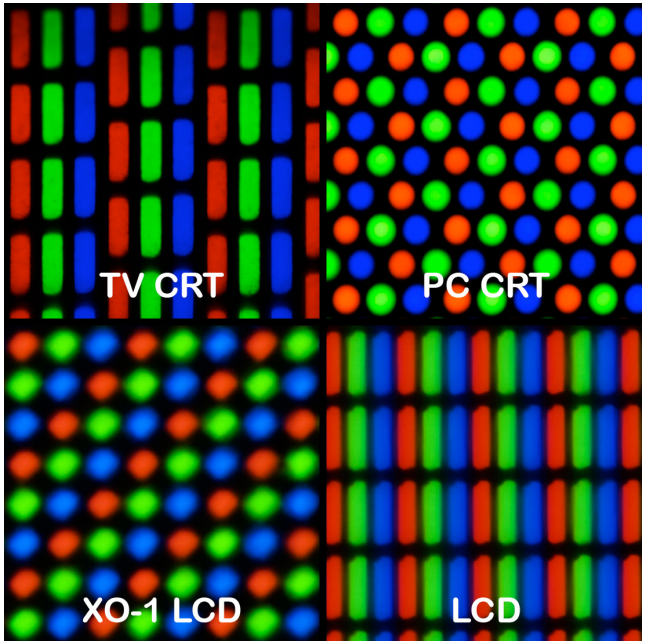


Displays

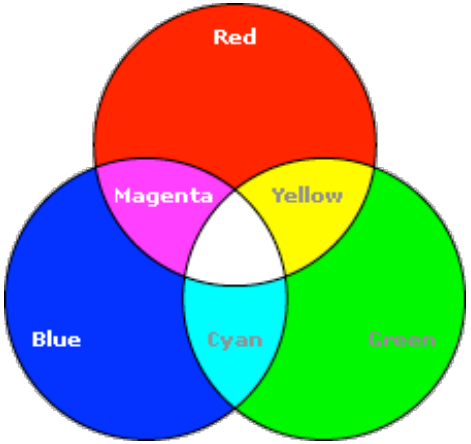
Pixels



Displays



Light



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

`video.c`

Virtual height

Virtual width

Note interpolation

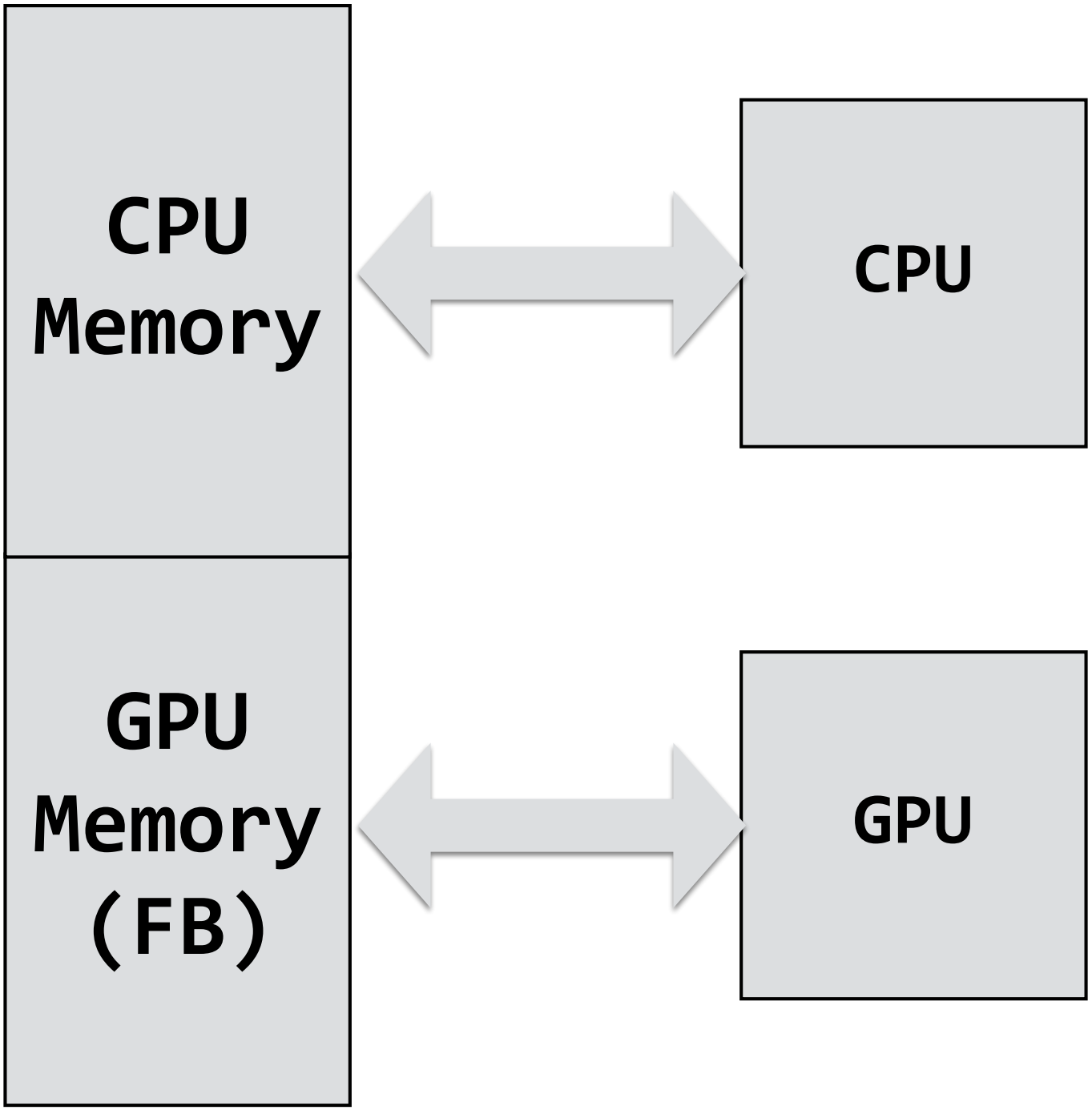
FB Config Structure

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

`fb.c`

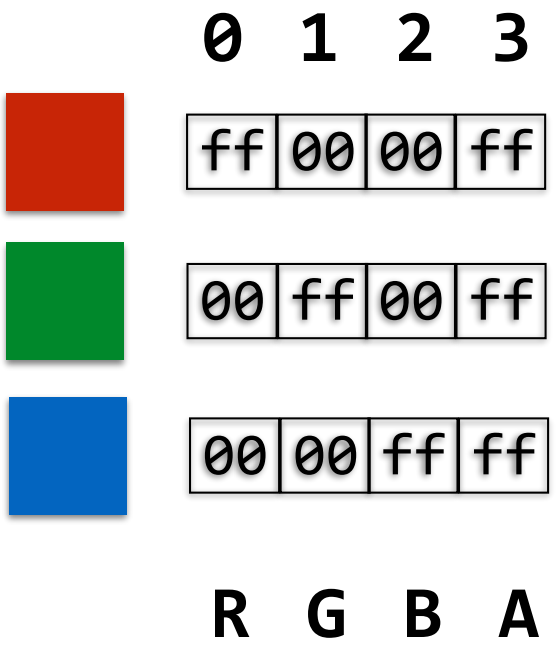

CPU
Memory
CPU
GPU
Memory
(FB)
GPU

Shared Memory

memory split between cpu and cpu
framebuffer memory is shared

`config.txt`
`memory.c`
`memmap`

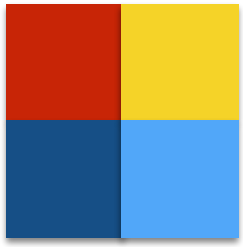
RGBA Pixel/Color



Note: red is the first byte (lowest address)

Array of `unsigned char`

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



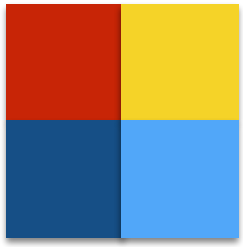
ff	00	00	ff	ff	ff	00	ff	00	00	ff	ff	00	ff	ff	ff

red **yellow** **blue** **cyan**

Note: (0,0) is the upper left on the monitor

Array of `unsigned char`

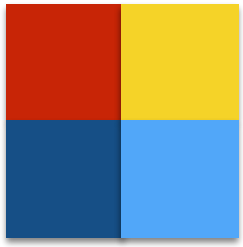
```
unsigned char fb[2*2*4];
fb[rgba + 4*(x + 2*y)] = …
rgba = 0 or 1 or 2 or 3
```



ff	00	00	ff	ff	ff	00	ff	00	00	ff	ff	00	ff	ff	ff
red				**yellow**				**blue**				**cyan**			

Array of `unsigned char`

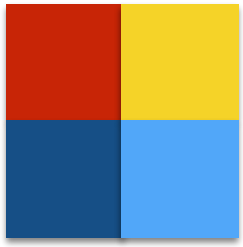
```
unsigned char fb[WIDTH*HEIGHT*DEPTH];
fb[rgba + DEPTH*(x + WIDTH*y)] = …
```



ff	00	00	ff	ff	ff	00	ff	00	00	ff	ff	00	ff	ff	ff
red				**yellow**				**blue**				**cyan**			

Array of `unsigned`

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



ff	00	00	ff	ff	ff	00	ff	00	00	ff	ff	00	ff	ff	ff

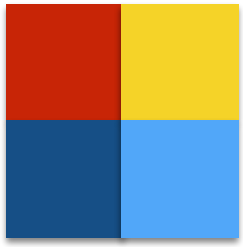
red **yellow** **blue** **cyan**

Drawing

`clear.c`
`cleari.c`
`clear2d.c`

2D Array of unsigned

```
unsigned fb[2][2];
fb[0][0] = 0xff0000ff;
fb[0][1] = 0xff00ffff;
fb[1][0] = 0xffff0000;
fb[1][1] = 0xffffff00;
```



ff	00	00	ff

red

ff	ff	00	ff

yellow

00	00	ff	ff

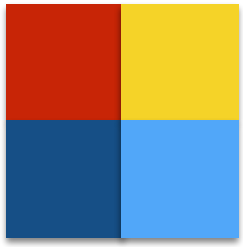
blue

00	ff	ff	ff

cyan

2D Array of unsigned

```
unsigned (*fb)[2] = (unsigned (*)[2])frame;
fb[0][0] = 0xff0000ff;
fb[0][1] = 0xff00ffff;
fb[1][0] = 0xffff0000;
fb[1][1] = 0xffffff00;
```



ff	00	00	ff	ff	ff	00	ff	00	00	ff	ff	00	ff	ff	ff
red				**yellow**				**blue**				**cyan**			

What is `unsigned *fb[2]`?

Demo

`grid.c`

Double-Buffering

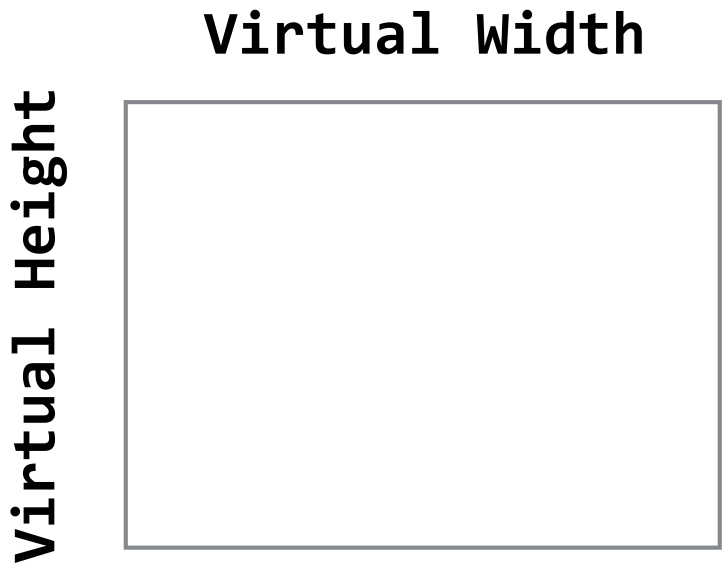
Double Buffering

Writing directly to screen can cause flickering

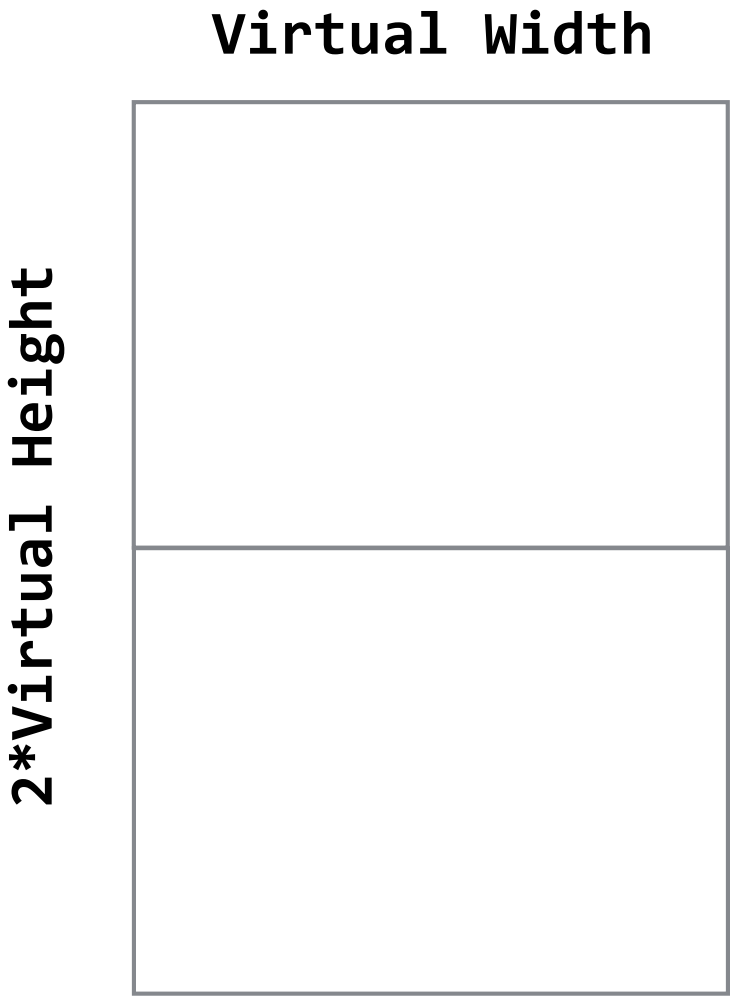
Solution: Double buffering

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

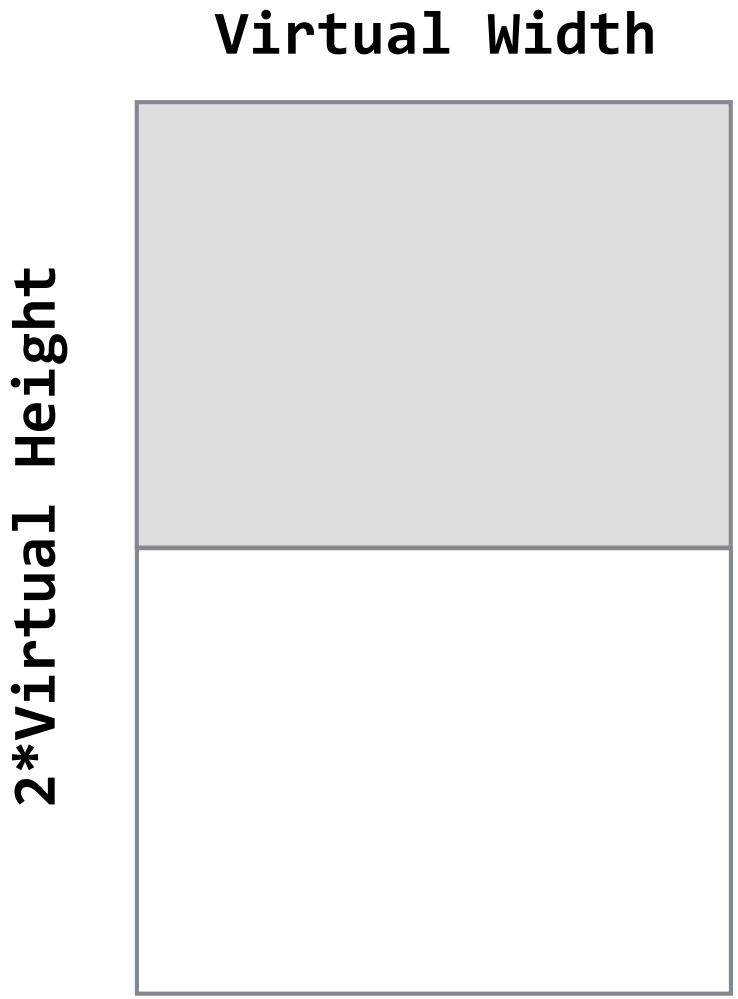
Single Buffer



Double Buffer

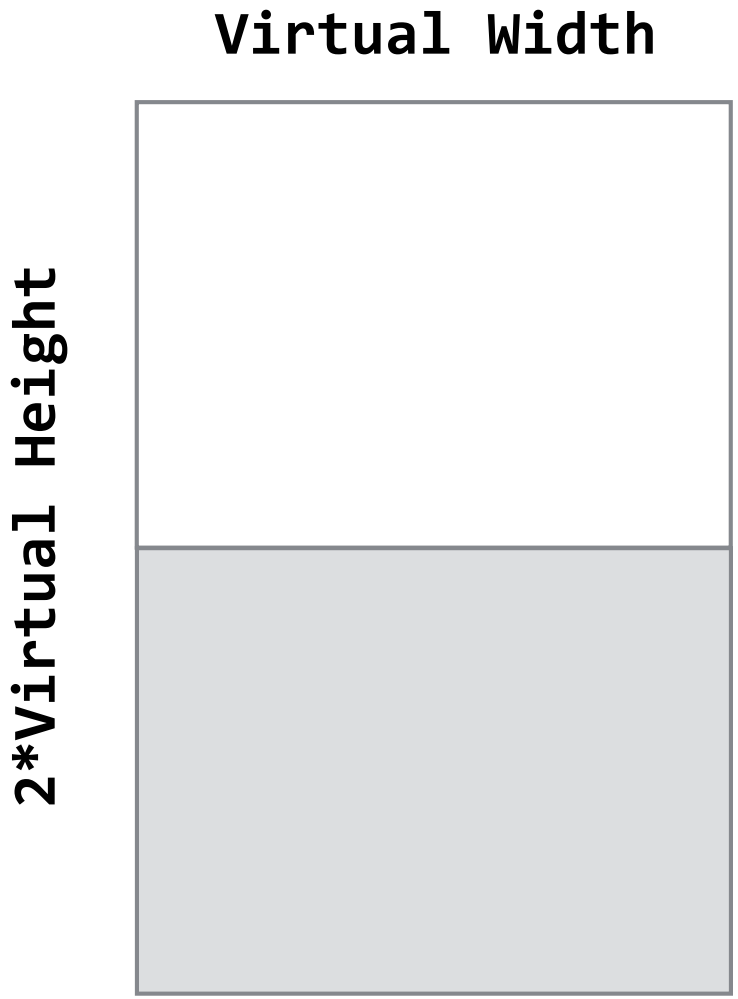


Display Top Buffer




```
x_offset = 0;
y_offset = 0;
```

Display Bottom Buffer



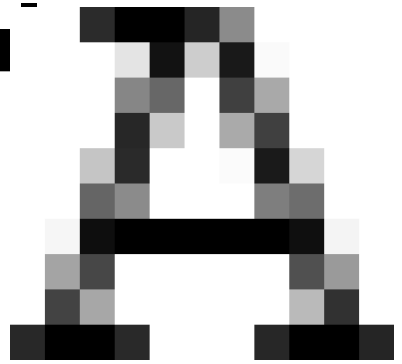

```
x_offset = 0;
y_offset =
    vheight;
```

Text

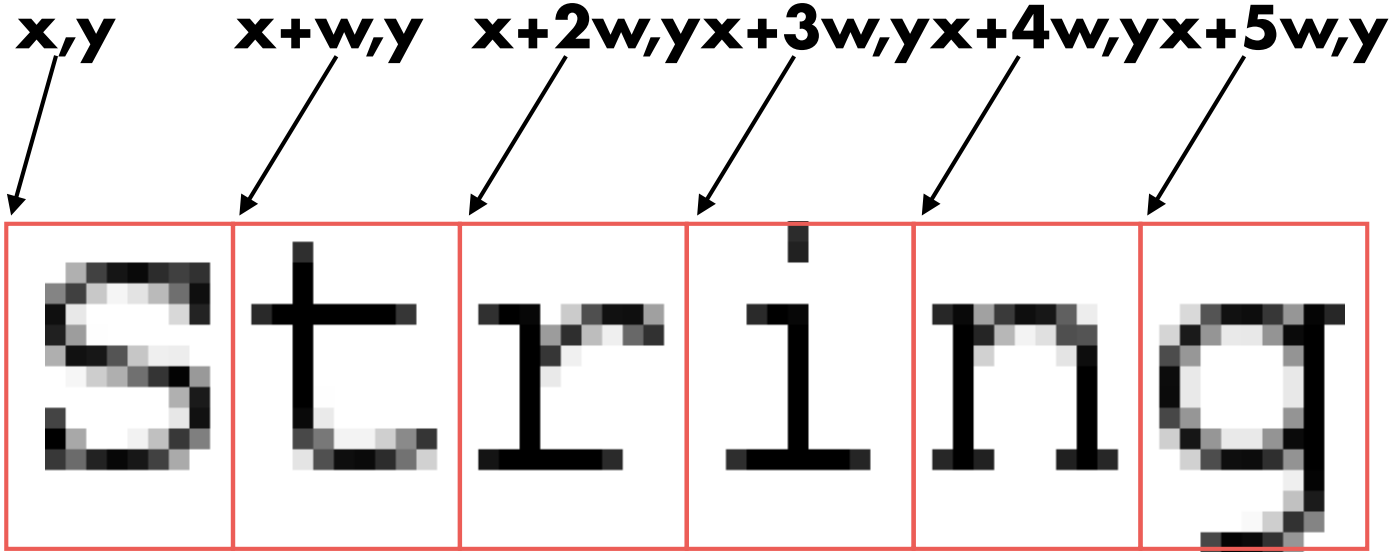
`hello.c`

Drawing Text

Fonts: monospaced vs. proportiona

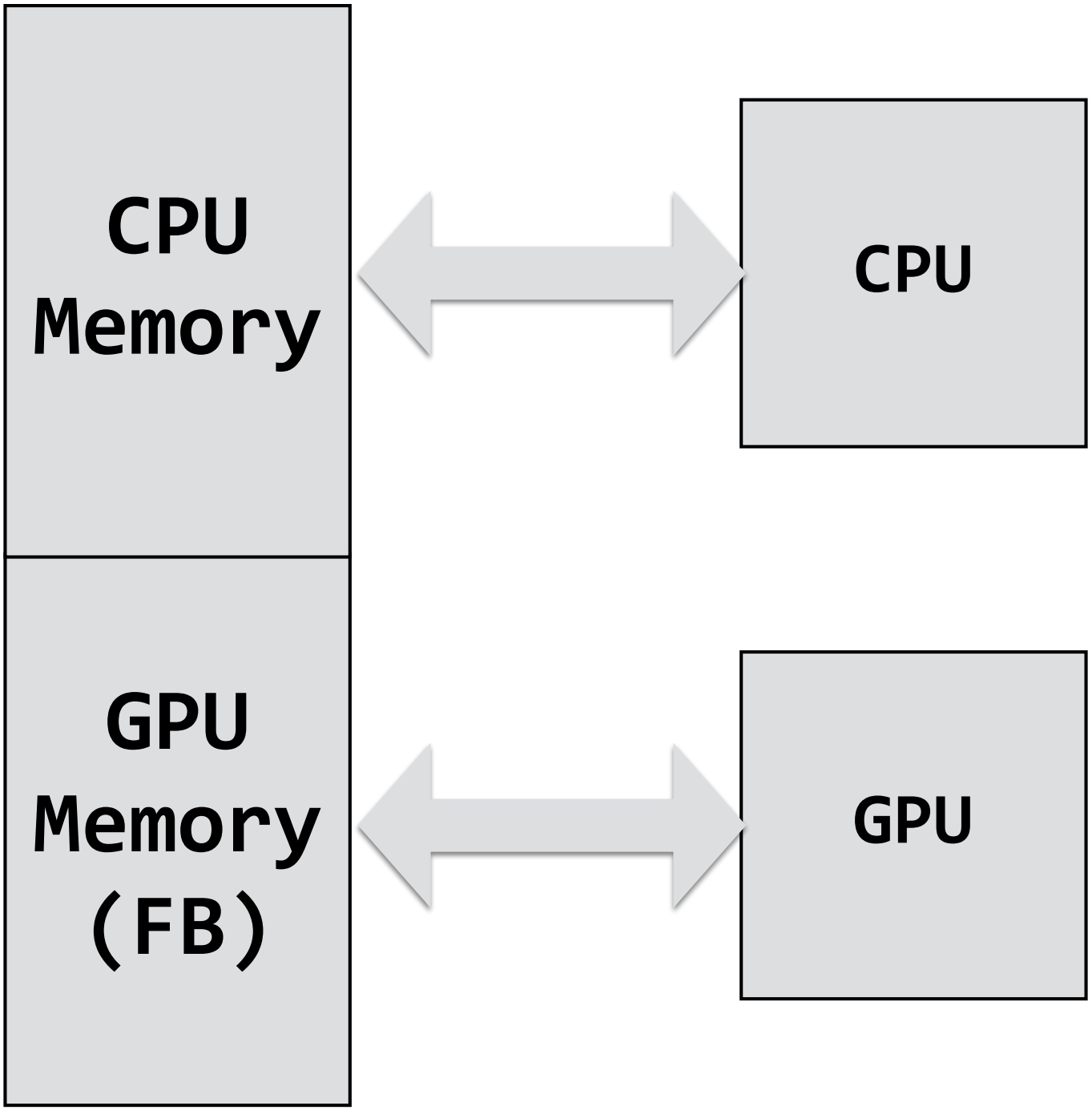


String: a series of characters

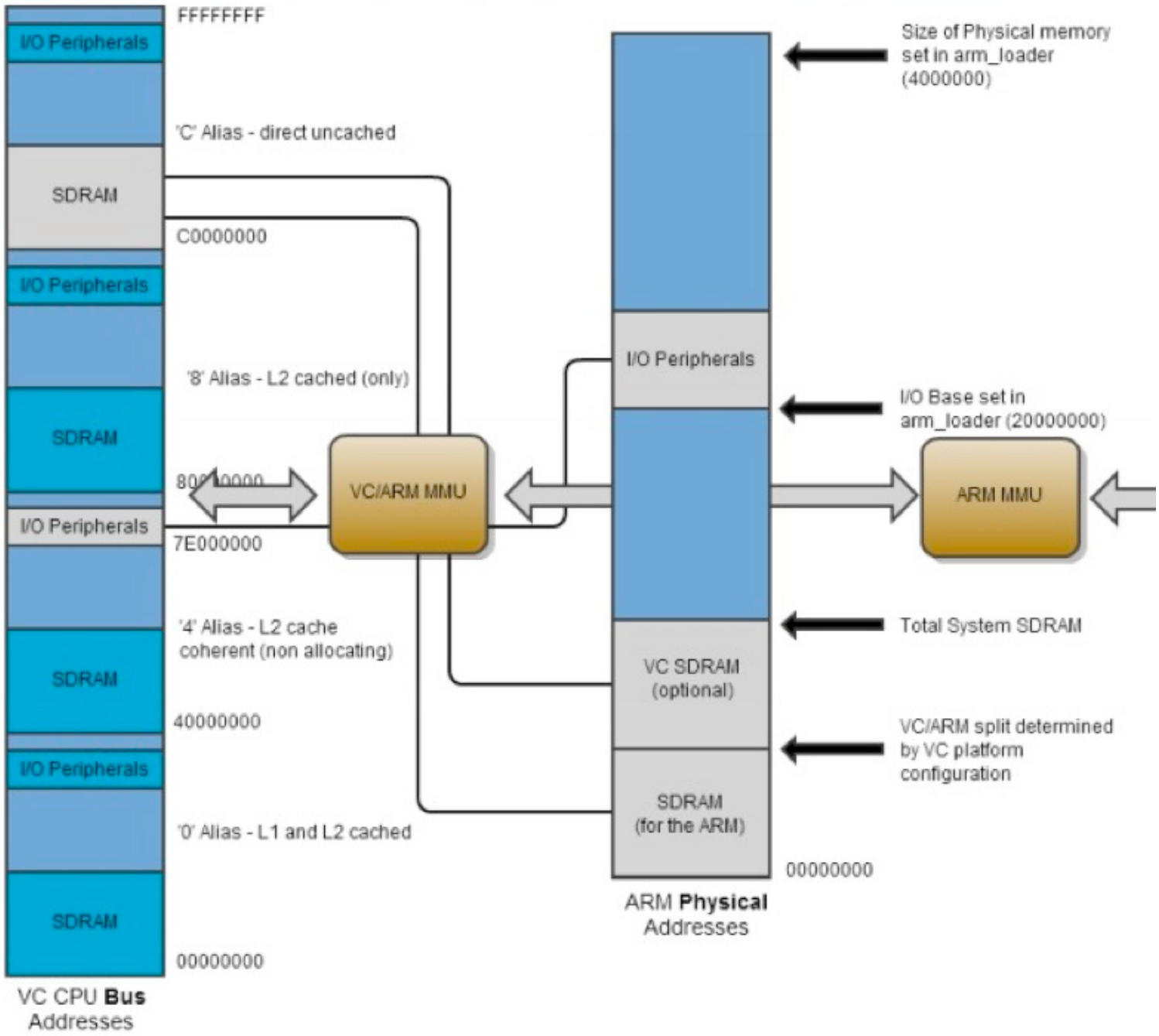



```
char str[7];
str[0] = 'S';
str[1] = 't';
str[2] = 'r';
str[3] = 'i';
str[4] = 'n';
str[5] = 'g';
str[6] = 0;
```

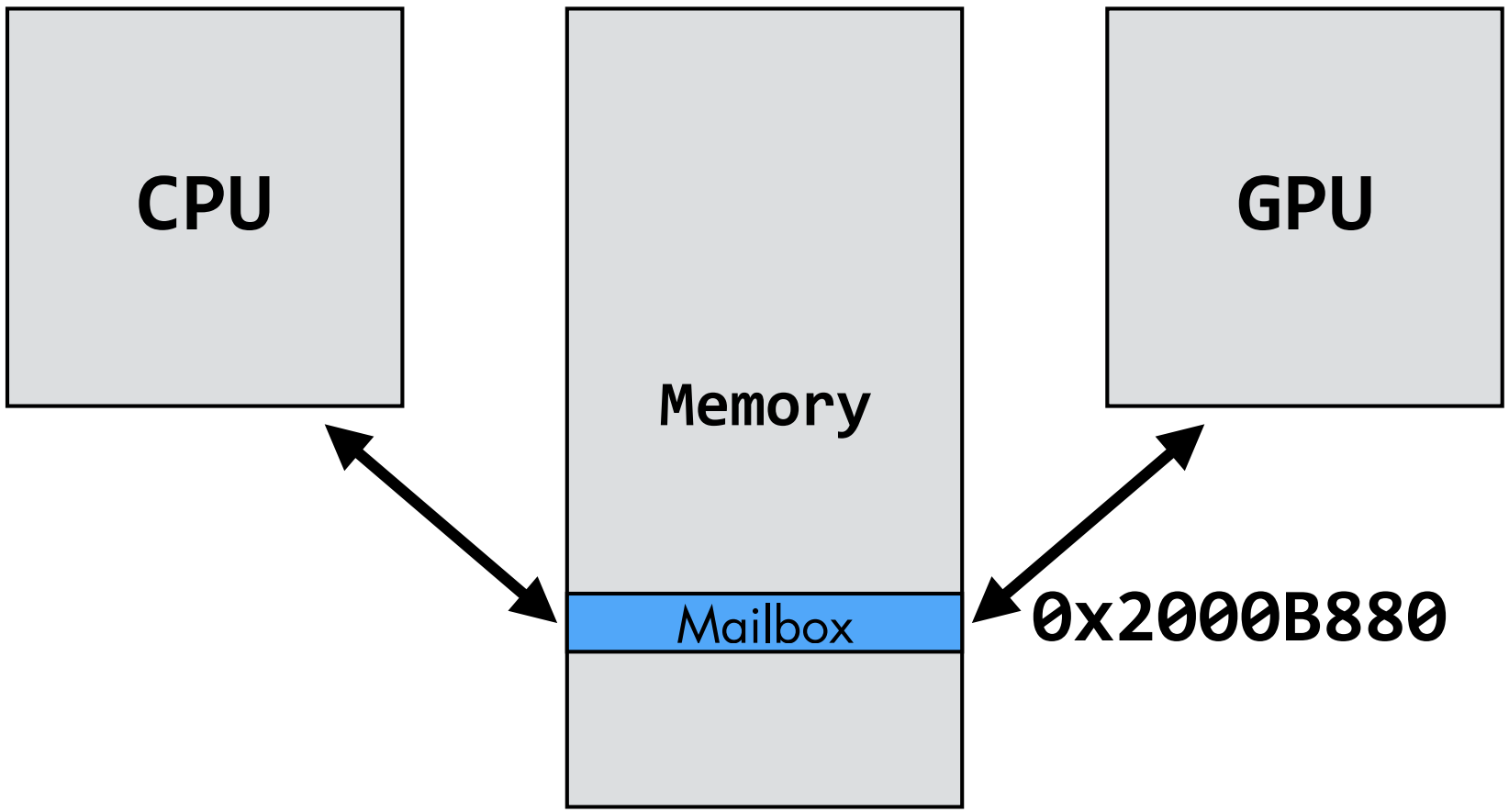
Mailbox


CPU
Memory
CPU
GPU
Memory
(FB)
GPU

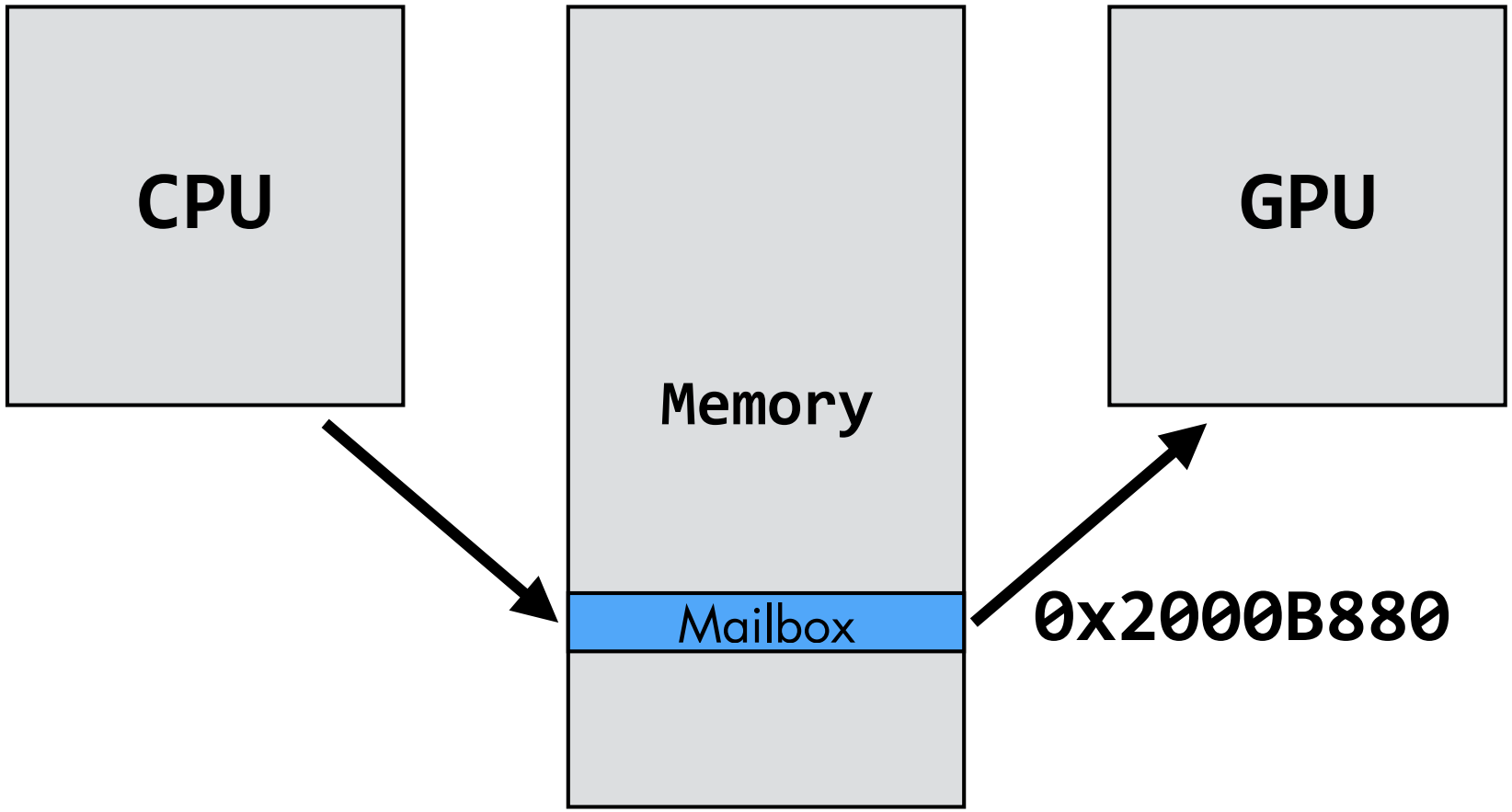
BCM2835 ARM Peripherals



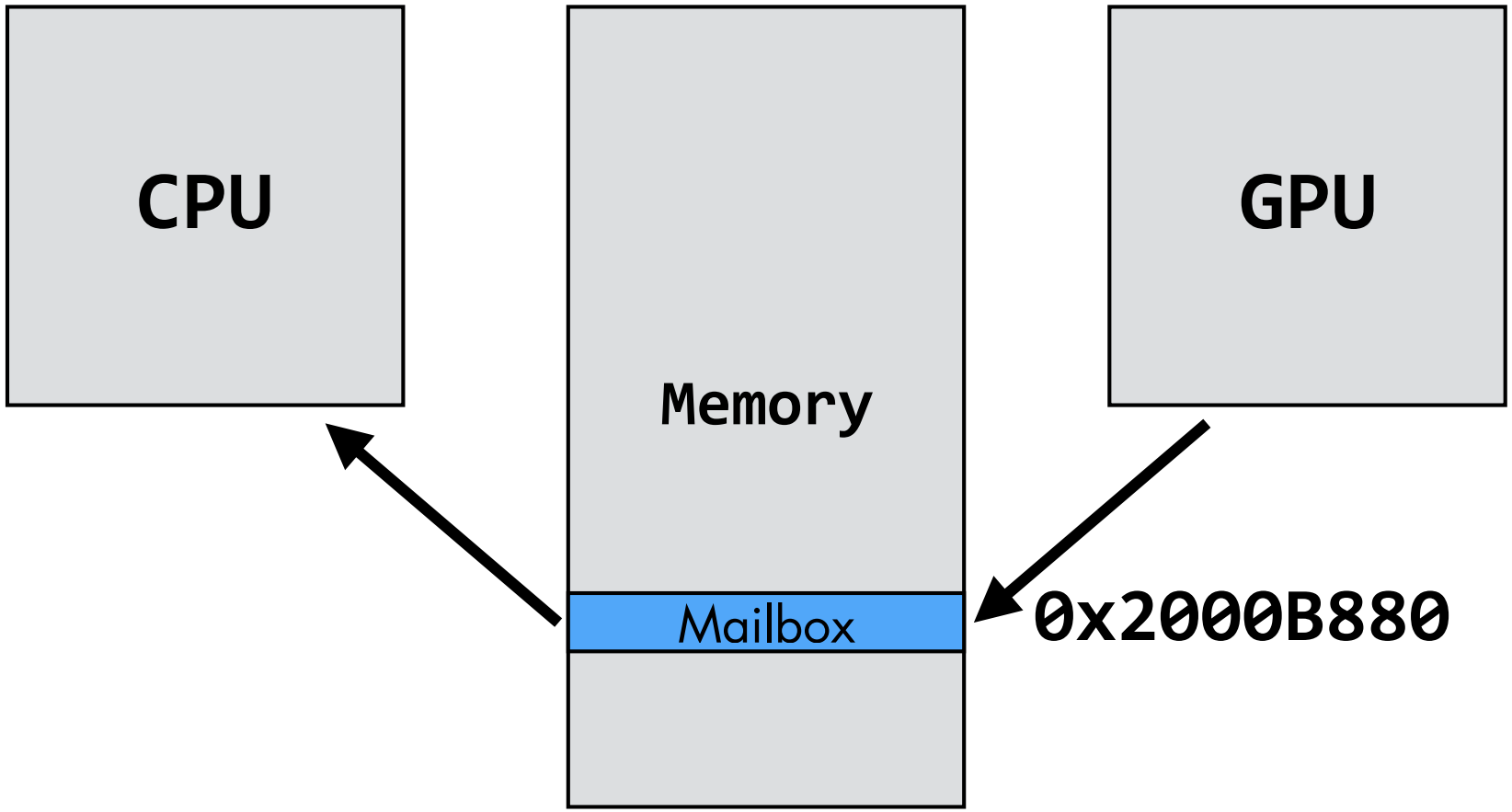
Mailbox



CPU Mails GPU

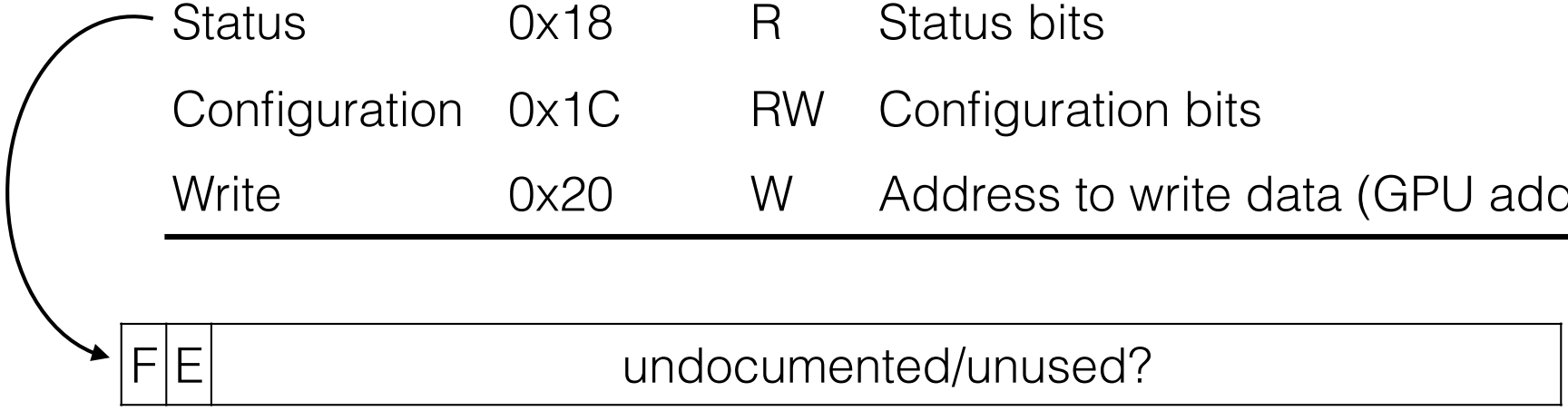


GPU Mails CPU



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

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 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