Graphics and the Framebuffer

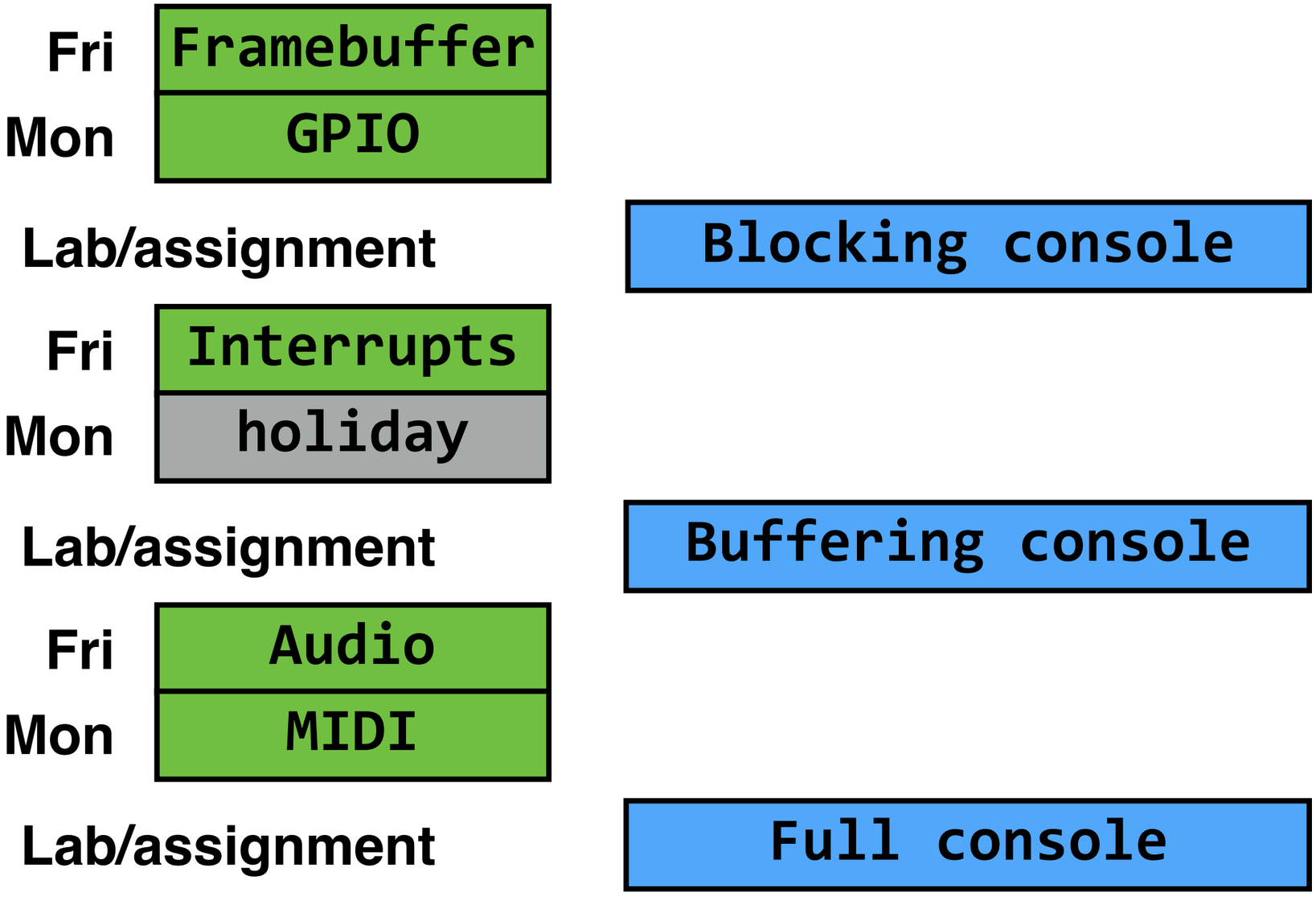
Next 3 weeks

Building on your central processing unit, in these next 3 weeks you will start to make a full-fledged computer.

Theme: input/output (I/O), connecting your CPU to interesting devices that increase its functionality.

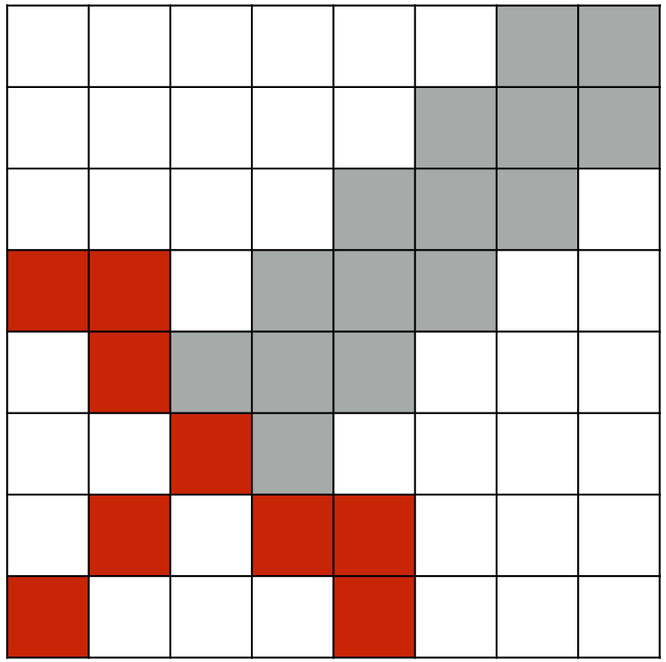
Goal: a command console, you can type in commands and see them on a display.

Schedule

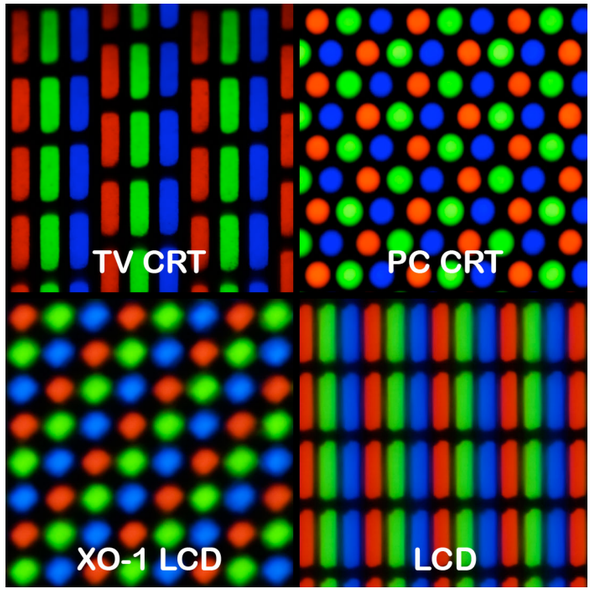

Fri
Framebuffer
Mon
GPIO
Lab/assignment
Blocking console
Fri
Interrupts
Mon
holiday
Lab/assignment
Buffering console
Fri
Audio
Mon
MIDI
Lab/assignment
Full console

Displays, RGB Color

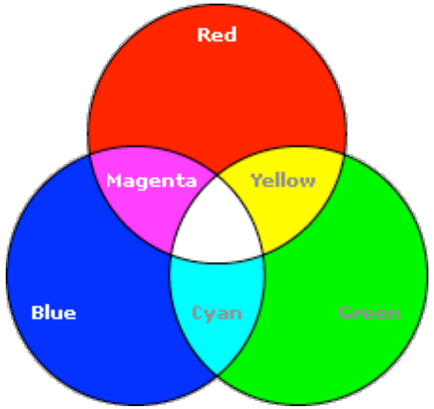
Pixels



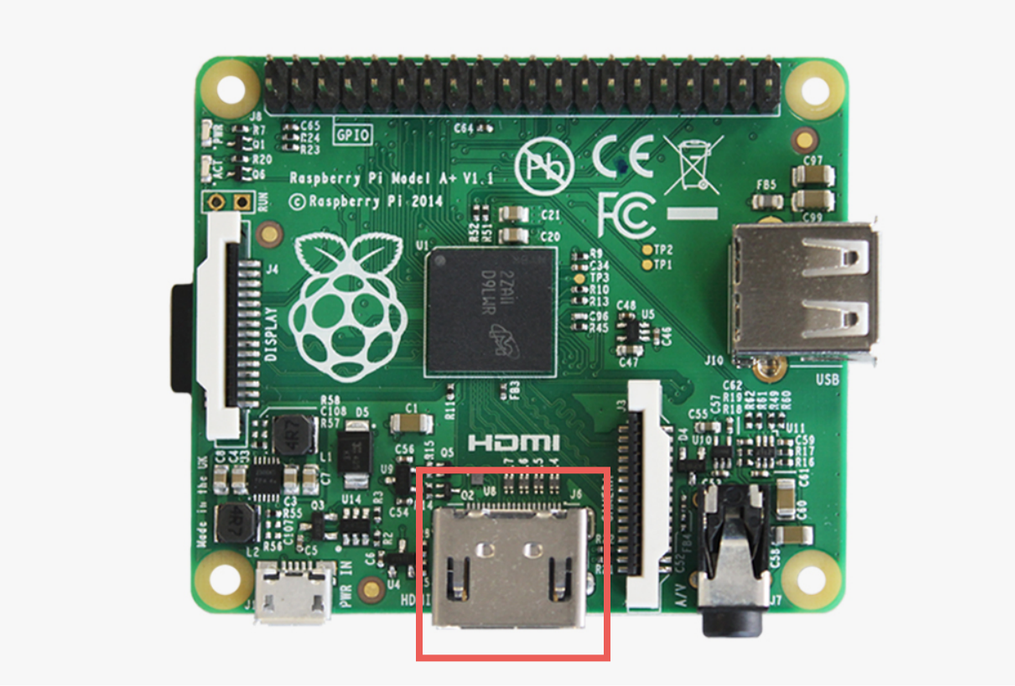
Displays

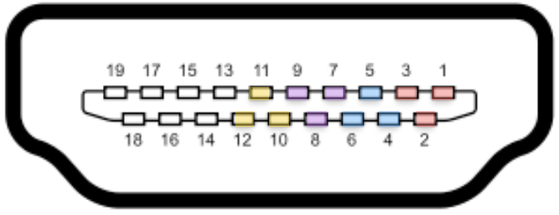


Light



HDMI





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

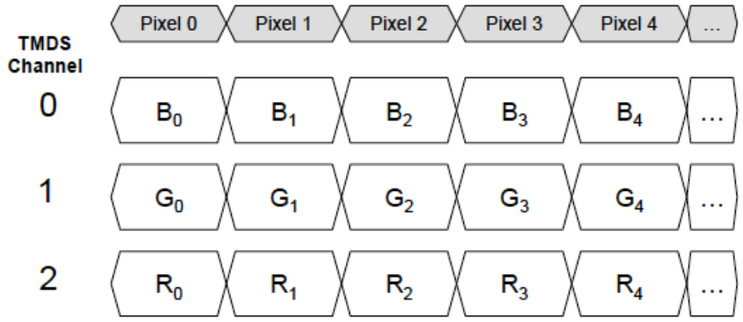
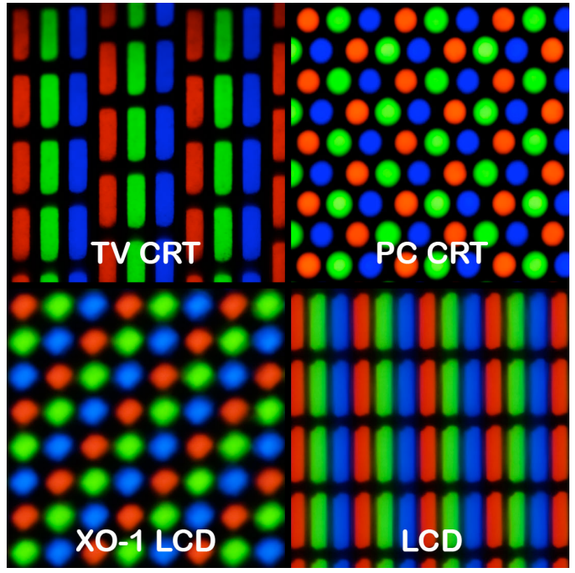


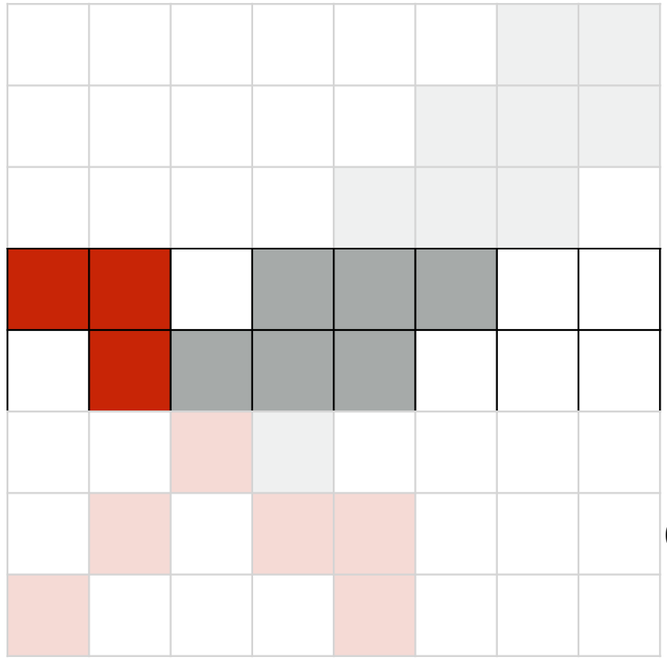
Figure 6-1 Default pixel encoding: RGB 4:4:4, 8 bits/component

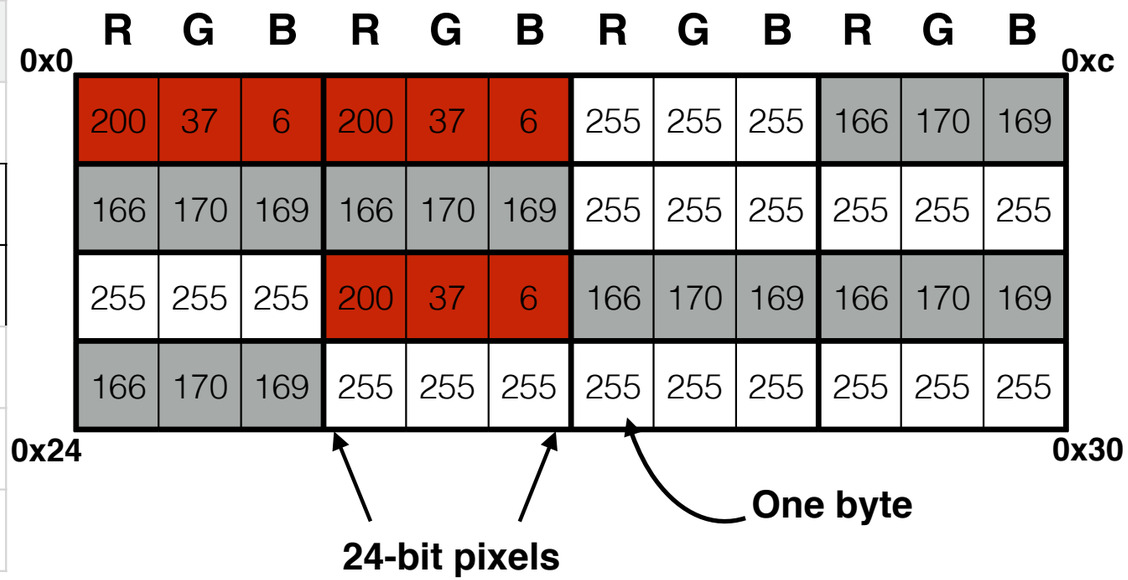
Figure from High-Definition Multimedia Interface Specification Version 1.3a



Framebuffer

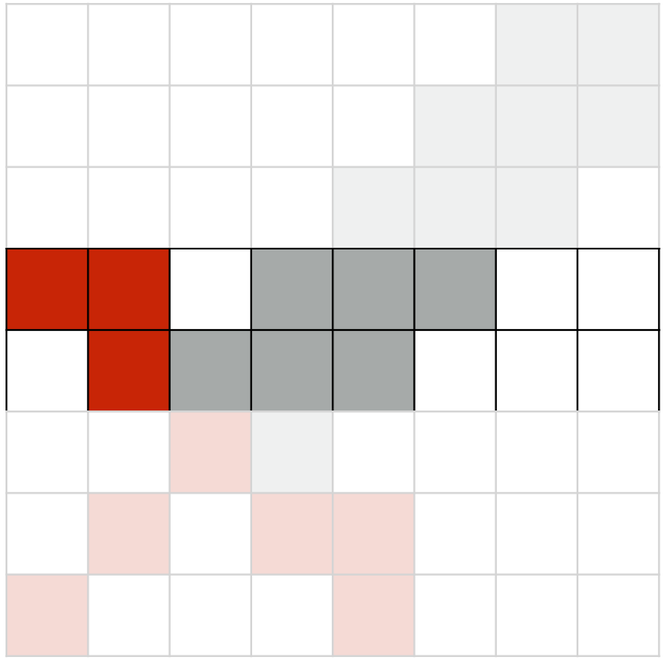
The display is a block of memory





Framebuffer

The display is a block of memory



R	G	B	R	G	B	R	G	B	R	G	B
200	37	6	200	37	6	255	255	255	166	170	169
166	170	169	166	170	169	255	255	255	255	255	255
255	255	255	200	37	6	166	170	169	166	170	169
166	170	169	255	255	255	255	255	255	255	255	255

```
#define X 640
#define Y 480
#define WIDTH 3
#define SIZE (X * Y * WIDTH)
char* buffer; // explain how you get the pointer shortly

// access pixel x,y
char red = buffer[y*WIDTH + x];
char blue = buffer[y*WIDTH + x + 1];
```

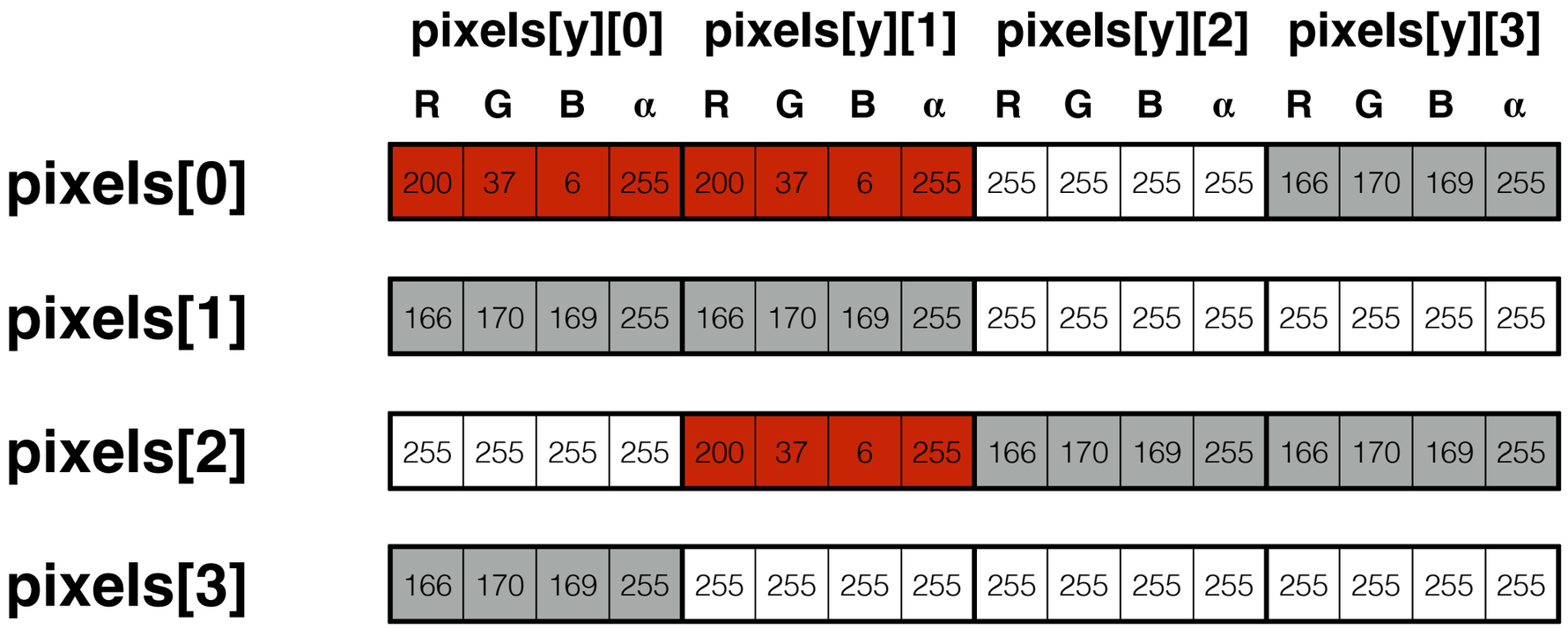
Multi-Dimensional Arrays

```
unsigned int pixels[4][4];      // y, x
pixels[2][1] = 0xff0625c8;
pixels[2][2] = 0xffa9aaa6;
```

R	G	B	α	R	G	B	α	R	G	B	α	R	G	B	α
200	37	6	255	200	37	6	255	255	255	255	255	166	170	169	255
166	170	169	255	166	170	169	255	255	255	255	255	255	255	255	255
255	255	255	255	200	37	6	255	166	170	169	255	166	170	169	255
166	170	169	255	255	255	255	255	255	255	255	255	255	255	255	255

Multi-Dimensional Arrays

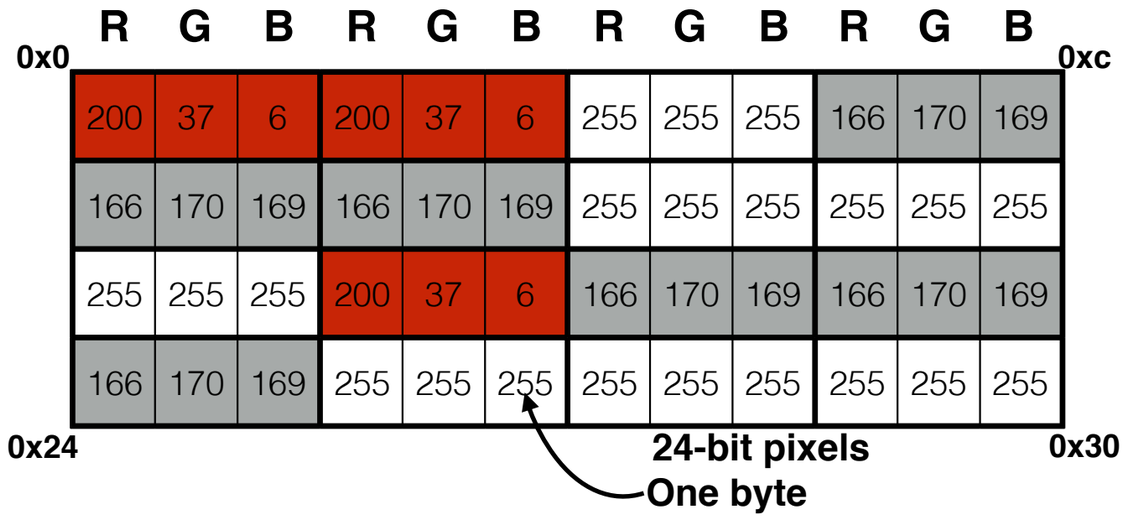
```
unsigned int pixels[4][4];      // y, x
pixels[2][1] = 0xff0625c8;
pixels[2][2] = 0xffa9aaa6;
```



More Multi-Dimensional

unsigned char pixels[4][4][3];

```
pixels[2][1][0] = 0xc8;  // 200
pixels[2][1][1] = 0x25;  // 37
pixels[2][1][2] = 0x06;  // 6
```

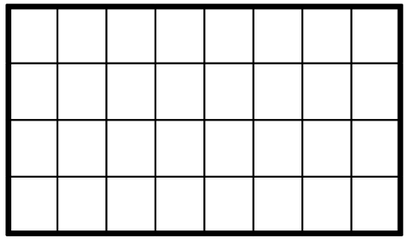


Casting

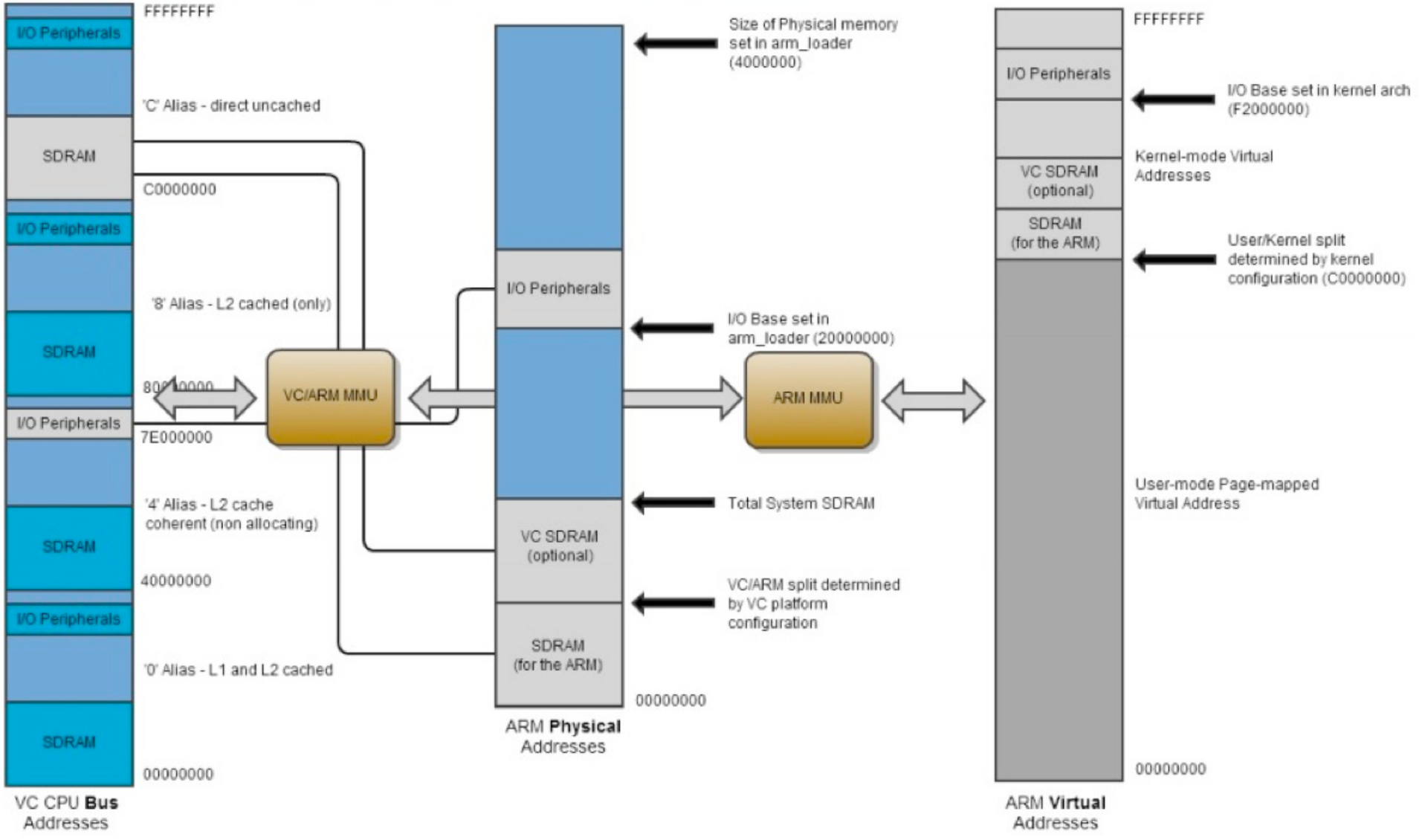
```
int a[32];
int (*b)[8] = (int(*)[8])a;
```

a

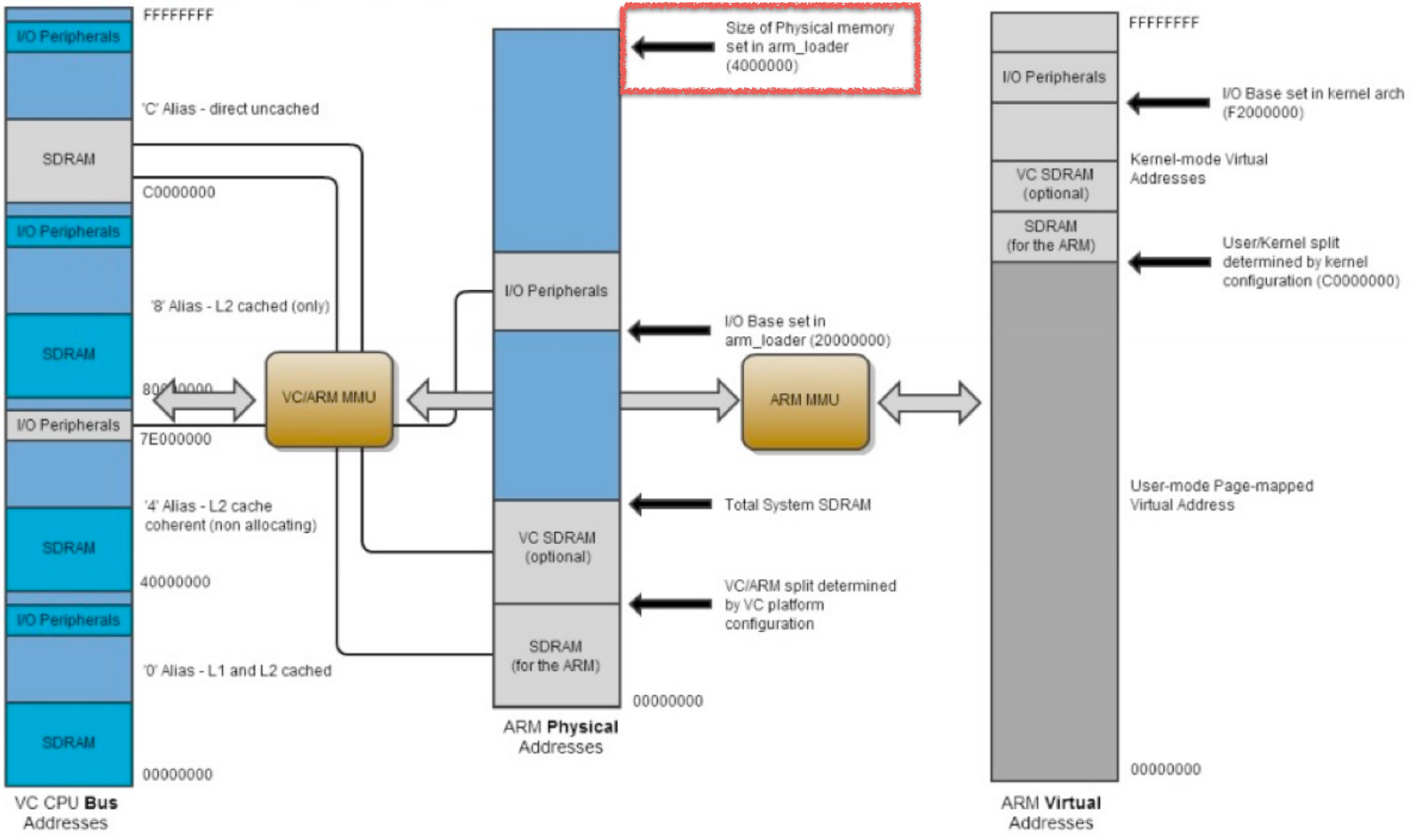
b

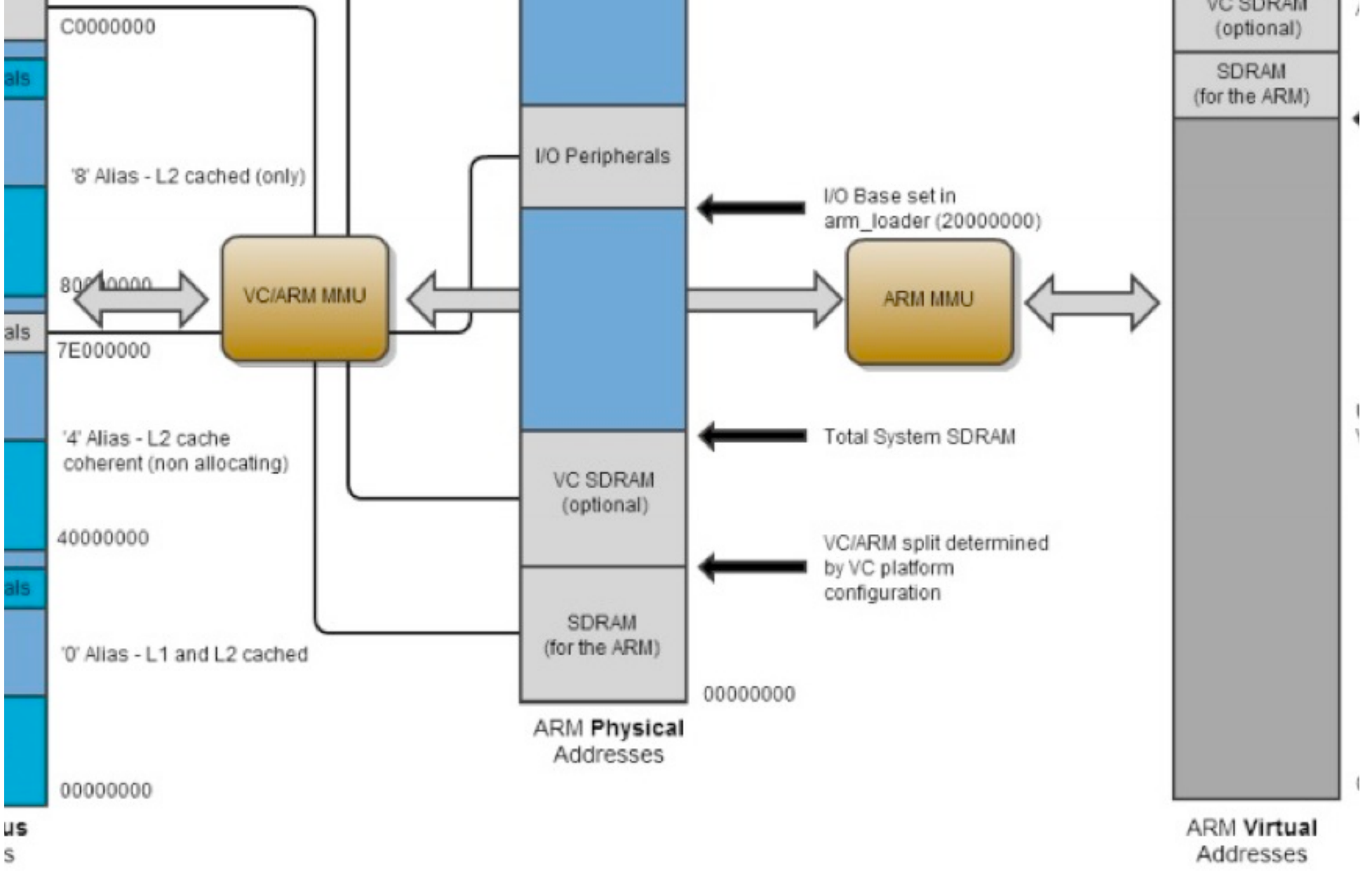


BCM2835 ARM Peripherals

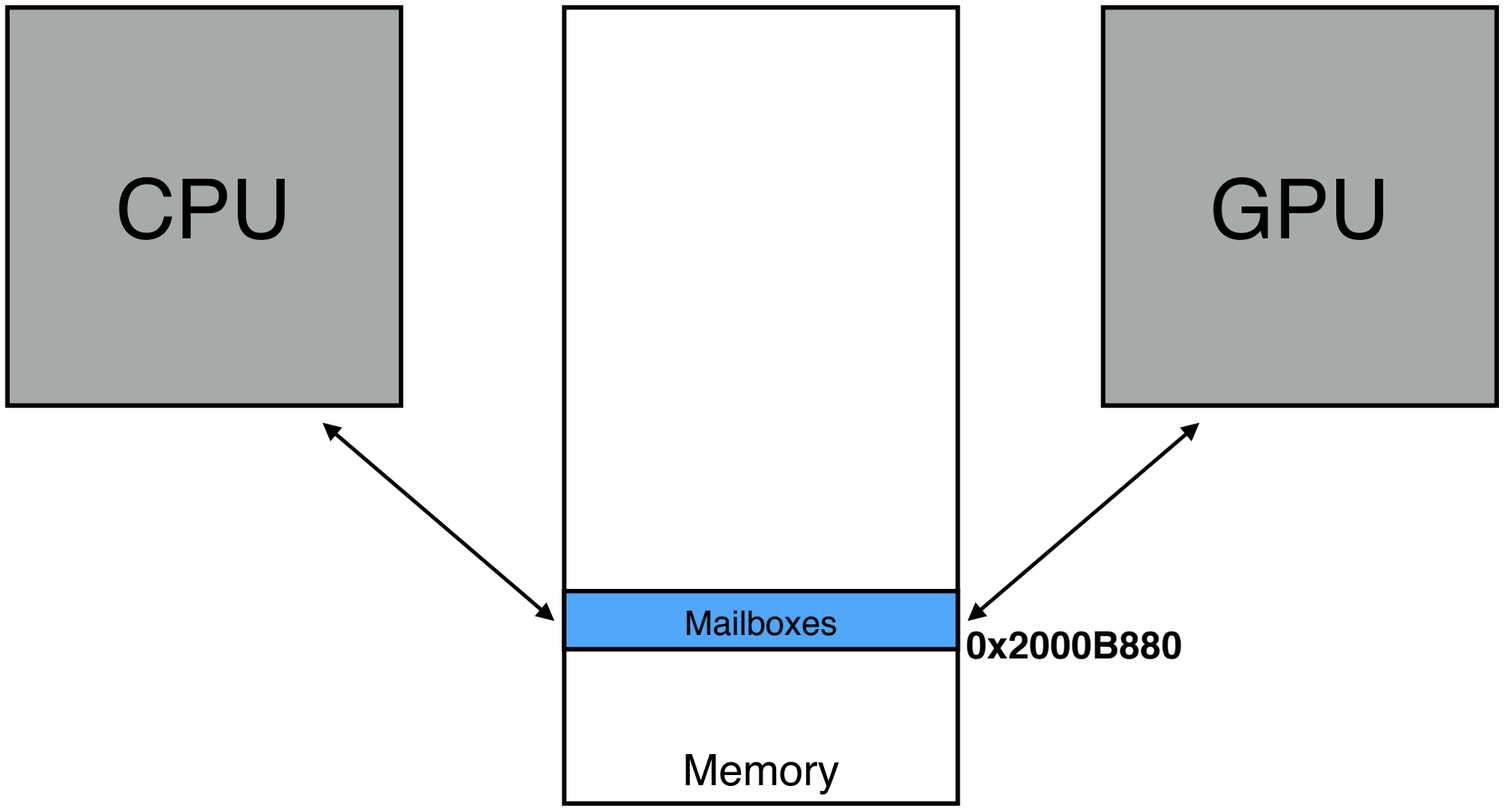


BCM2835 ARM Peripherals



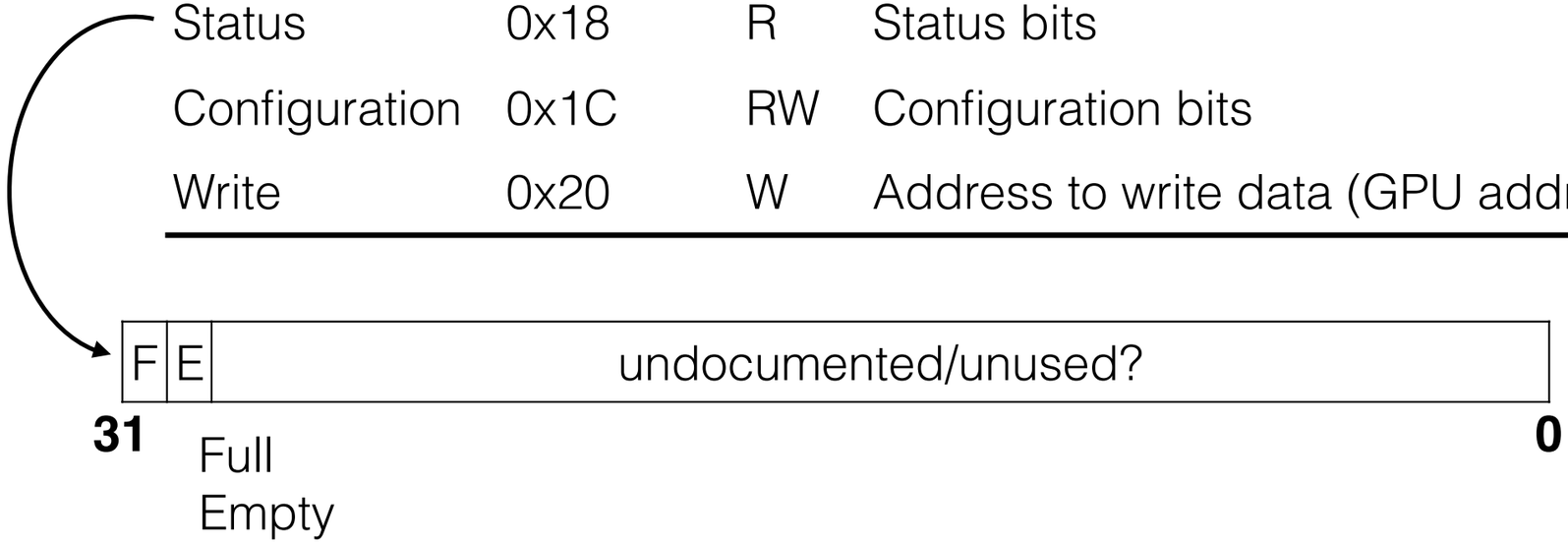

C0000000
'8' Alias - L2 cached (only)
80000000
7E000000
'4' Alias - L2 cache coherent (non allocating)
40000000
'0' Alias - L1 and L2 cached
00000000
VC/ARM MMU
I/O Peripherals
VC SDRAM (optional)
SDRAM (for the ARM)
00000000
ARM Physical Addresses
I/O Base set in arm_loader (20000000)
Total System SDRAM
VC/ARM split determined by VC platform configuration
ARM MMU
VC SDRAM (optional)
SDRAM (for the ARM)
ARM Virtual Addresses

Mailboxes



Mailbox Format

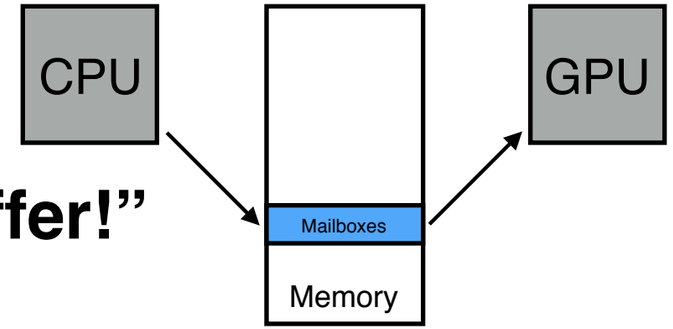
Register	Offset	R/W	Use
Read	0x00	R	Destructively read value
Peek	0x10	R	Read without removing data
Sender	0x14	R	Sender ID (bottom 2 bits)
Status	0x18	R	Status bits
Configuration	0x1C	RW	Configuration bits
Write	0x20	W	Address to write data (GPU addr)



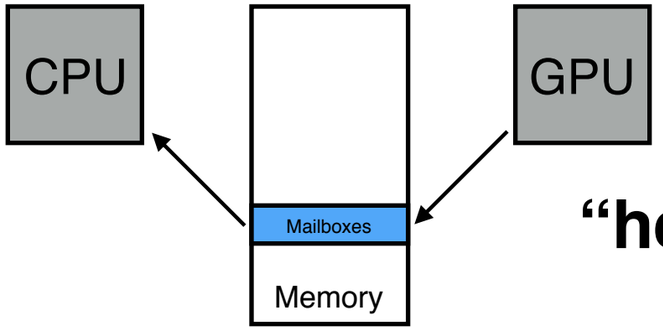
Getting a Framebuffer

Send a command to the GPU via mailbox

“give me a frame buffer!”



Wait for the response, providing the pointer



“here ya go”

Writing Pseudocode

```
// Wait while mailbox is full
while (status & full) {}

// Write the address of config structure
// Address must be 16-byte aligned
// Bottom 4 bits say channel, channel 1 is FB
write = address + 1;
```

Reading Pseudocode

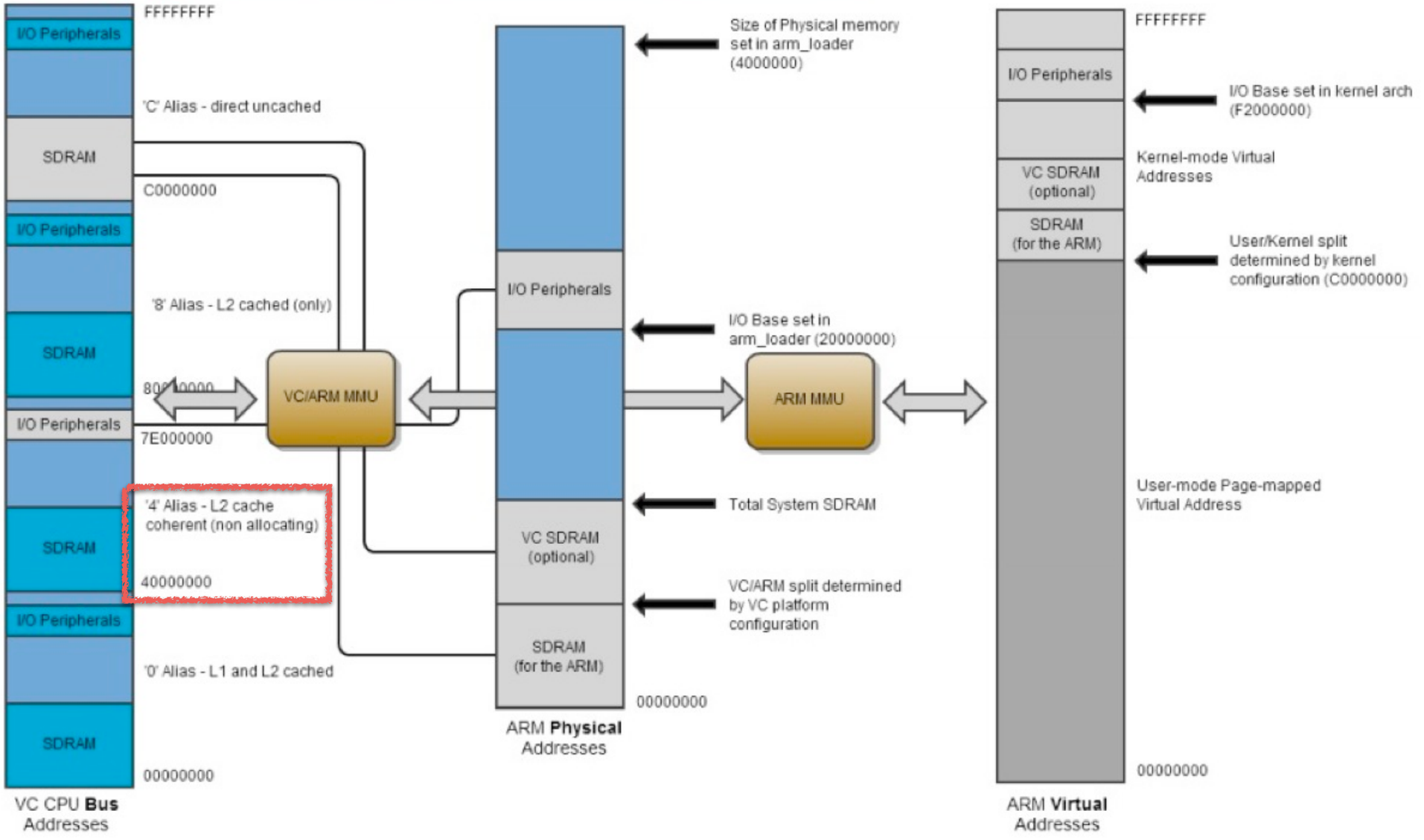
```
// Wait until expected message
while true {
  // Wait until mailbox not empty
  while (status & empty) {}
  val = read;

  // Expected channel, break out of while(1)
  if ((val & 0xF) == 1) break;
}
return val;
```

Config Structure

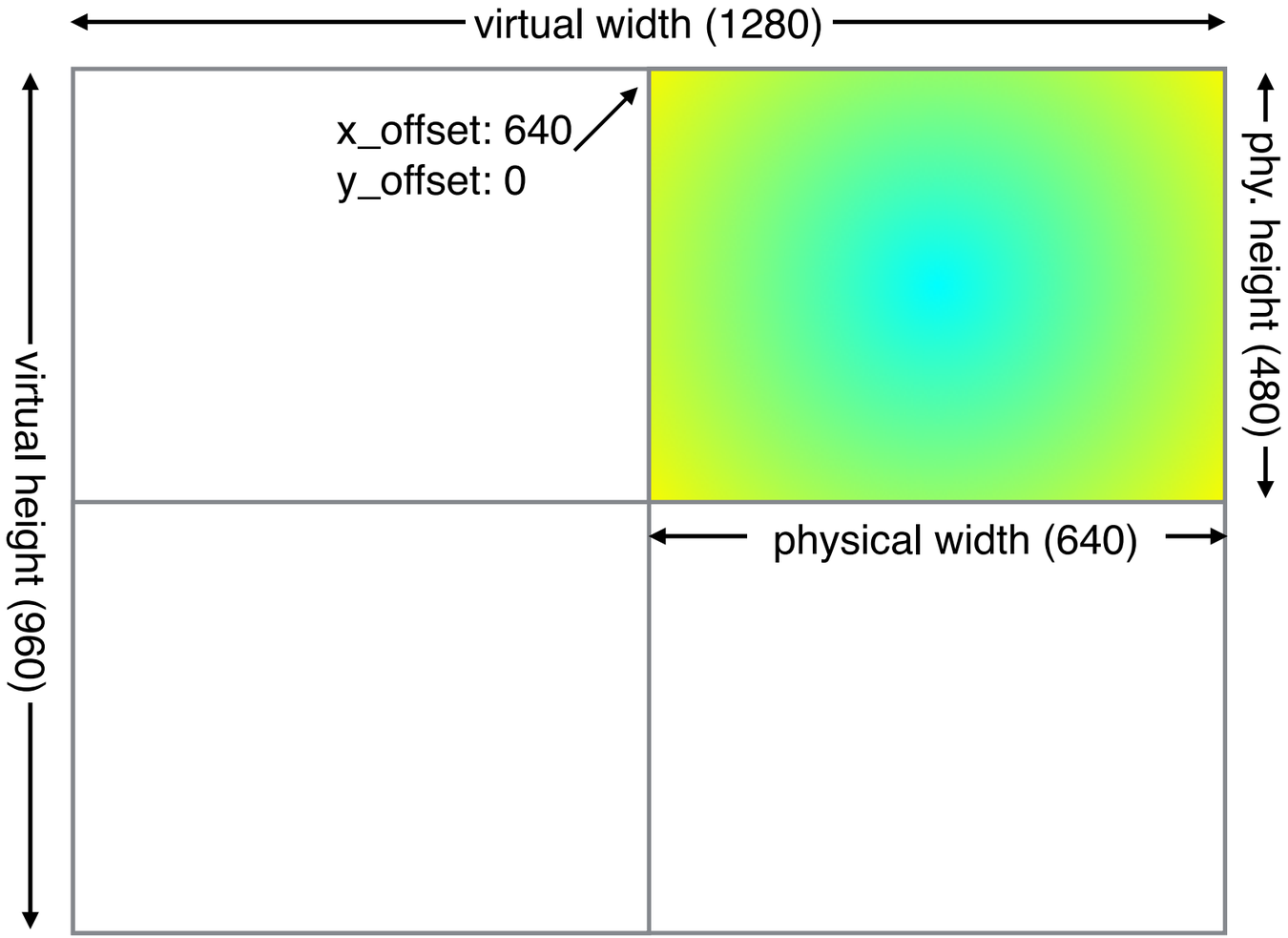
40 bytes long, specifies 10 parameters
Have to pass as *GPU address*

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

BCM2835 ARM Peripherals



Framebuffer Geometry

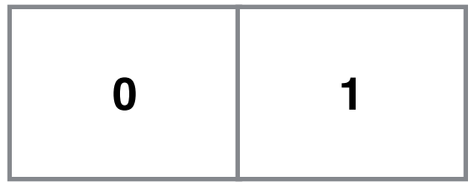


Double Buffering

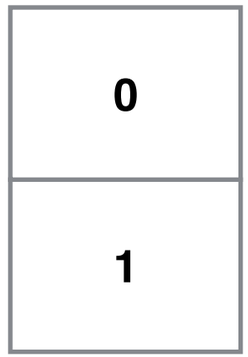
Writing directly to screen can look bad if it takes a lot of time, human can perceive it

Double buffering: write to an off-screen buffer, swap that buffer in to update screen

Which arrangement is better?



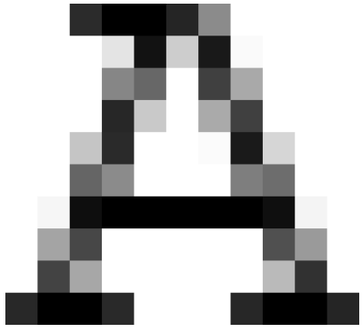
x



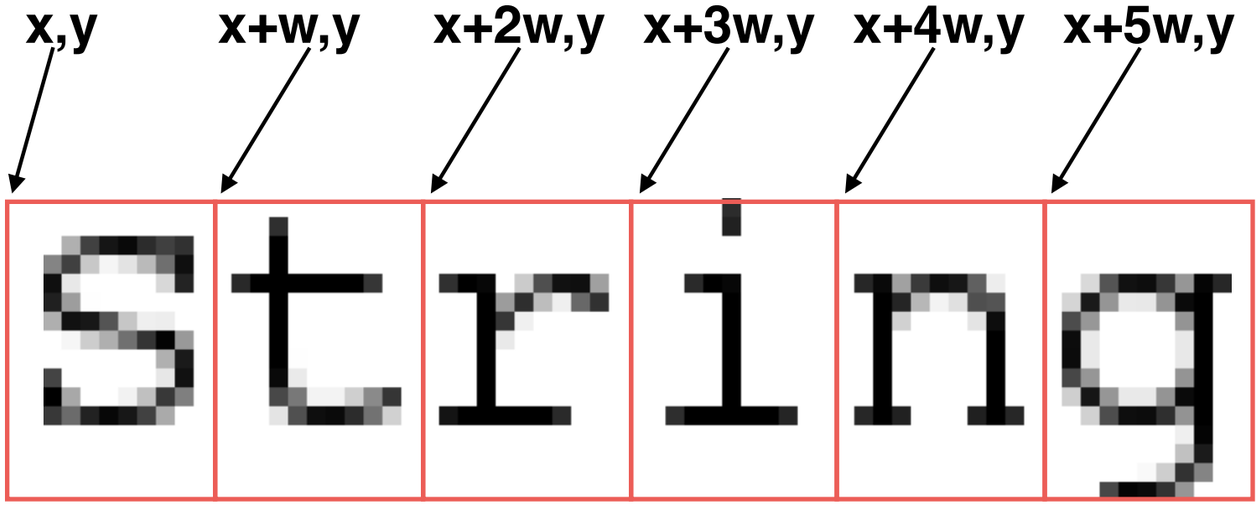
y

Drawing Text

Fonts: monospaced vs. proportional



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;
```

Framebuffer Overview

GPU is an I/O device whose data plane (frame buffer) is a shared memory region.

GPU's control plane is a set of memory mapped registers (mailboxes).

Framebuffer can distinguish physical and virtual size, enabling double buffering.

Text can be a series of character images.