

CS 2261: Media Device Architecture - Week 7, part 2

Overview

- Some more Mode 4 examples

- Using Usenti



- Sprites

Mode 4 Examples/Demos

- `fillScreen4()`
- Simple `drawSquare4()`
- `drawVerticalLine4()`
- `drawHorizontalLine4()`
- `drawImage4()`

fillScreen4

```
void fillScreen4(volatile u8 paletteI){
    // let's set 4 pixels at a time!
    volatile unsigned int color = paletteI << 24 | paletteI << 16 |
                                   paletteI << 8 | paletteI;

    DMA[3].cnt = 0; // clear it first!
    DMA[3].src = &color;
    DMA[3].dst = videoBuffer;
    DMA[3].cnt = 1 << 31 | // turn it on
                1 << 24 | // with src fixed
                1 << 26 | // with chunk size 32
                (38400 / 4);
}
```

drawSquare4

(the lazier/slower one)

```
void drawSquare4(int x, int y, int size, u8 paletteI){  
    for (u8 i=0; i<size; i++){  
        for (u8 j=0; j<size; j++){  
            setPixel4(x+i, y+j, paletteI);  
        }  
    }  
}
```

drawVerticalLine4

```
void drawVerticalLine4(int x, int y, int length, u8 paletteI) {  
    for(u8 i=0; i<length; i++){  
        setPixel4(x, y + i, paletteI);  
    }  
}
```

drawHorizontalLine4

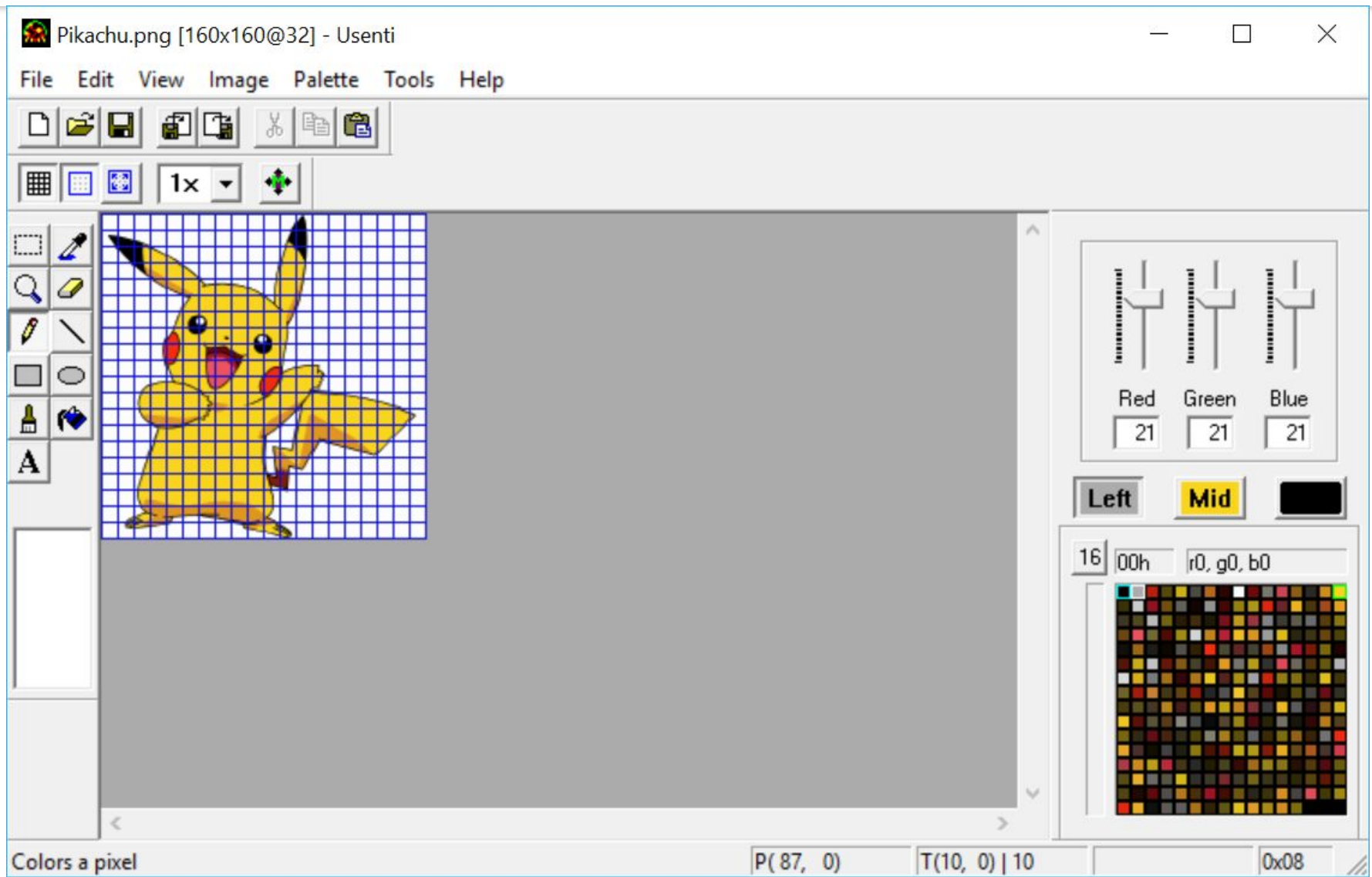
```
void drawHorizontalLine4(int x, int y, int length, u8 paletteI)
{
    for(u8 i=0; i<length; i++){
        setPixel4(x + i, y, paletteI);
    }
}
```

drawImage4

(also, slower/lazier way)

```
void drawImage4(int x, int y, int width, int height, const u8 *img){  
    for (int i=0; i<width; i++){  
        for(int j=0; j<height; j++){  
            setPixel4(x+i, y+j, img[i + width*j]);  
        }  
    }  
}
```


Using Usenti to get a picture for Mode4



Resize Fairly Small (don't forget to click "stretch")

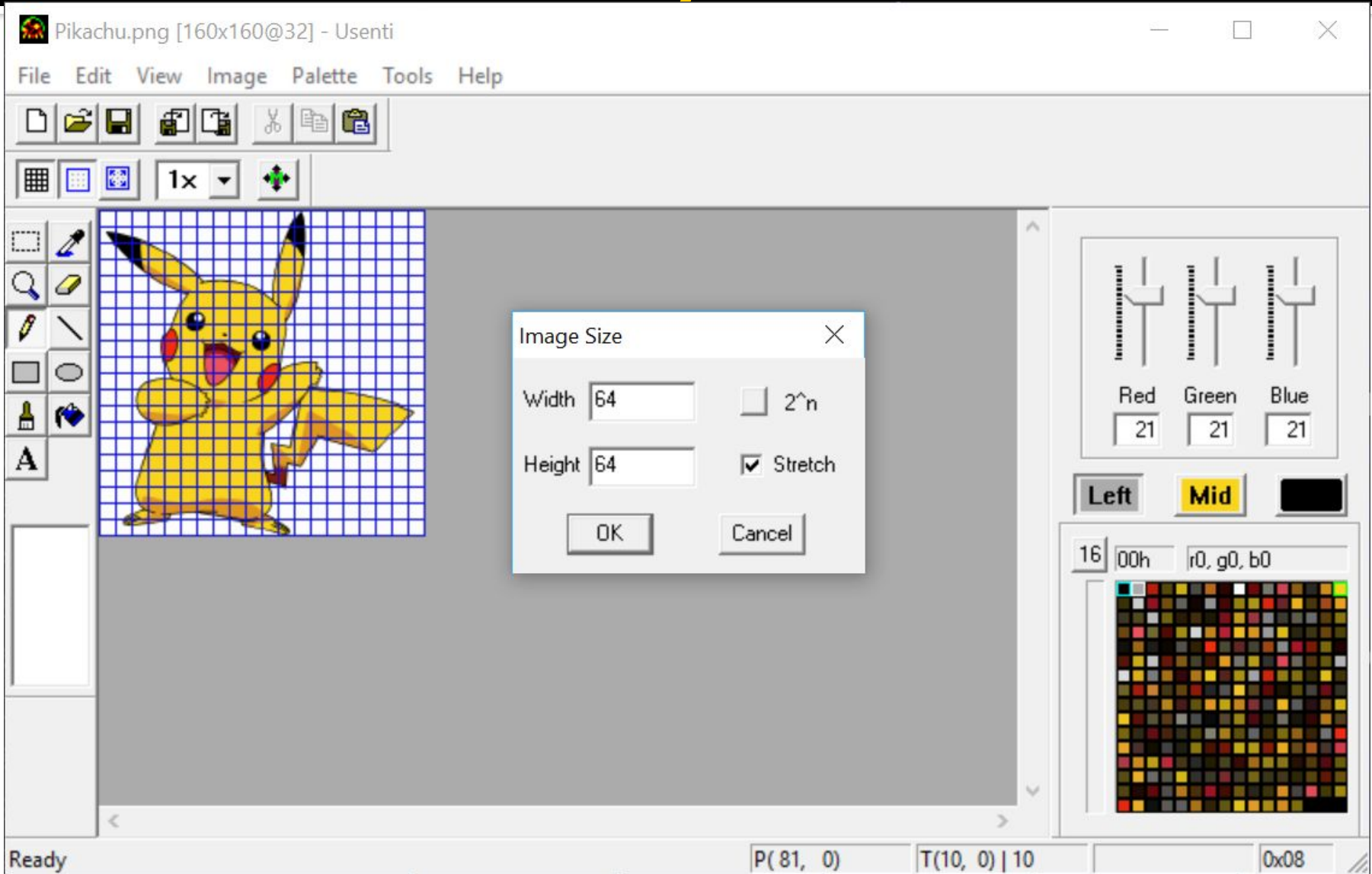
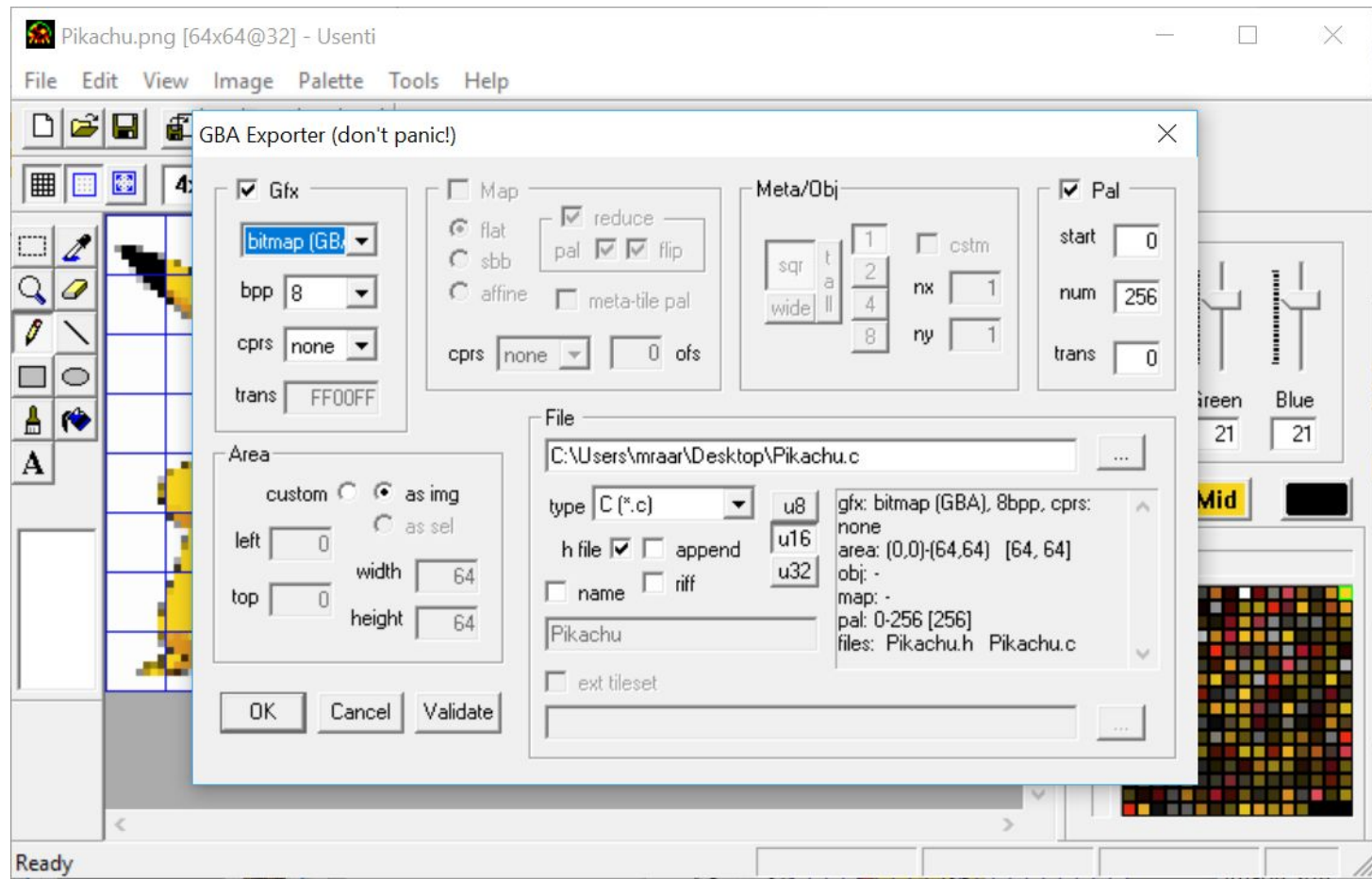


Image -> Export (GBA source)

- Pick "bitmap" 8bpp in Gfx column
- Make sure Pal is clicked



Profit (Note: constant is really helpful here! -- ends up in big ROM memory)

■ Pikachu.h

```
#define PikachuBitmapLen 4096
extern const unsigned short PikachuBitmap[2048];
```

```
#define PikachuPalLen 512
extern const unsigned short PikachuPal[256];
```

■ Pikachu.c

```
const unsigned short PikachuBitmap[2048] __attribute__((aligned(4)))=
{
    0x0808,0x0808,0x0808,0x0808,0x0808,0x0808,0x0808,0x0808,
    0x0808,0x0808,0x0808,0x0808,0x0808,0x0808,0x0808,0x0808,
    //...
const unsigned short PikachuPal[256] __attribute__((aligned(4)))=
{
    0x0000,0x56B5,0x0496,0x052A,0x0ED9,0x2129,0x1197,0x0005,
    0x7FFF,0x042D,0x35EF,0x2919,0x0971,0x10A5,0x0E3A,0x0F5F,
```

Adding to project

```
// in main.c  
#include "Pikachu.h"
```

Add Pikachu.c to sources in Makefile:

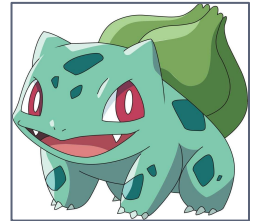
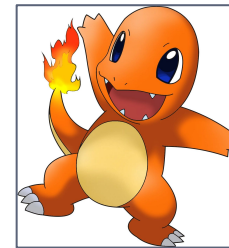
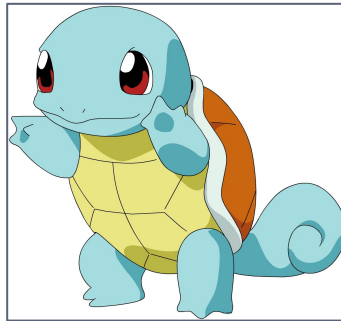
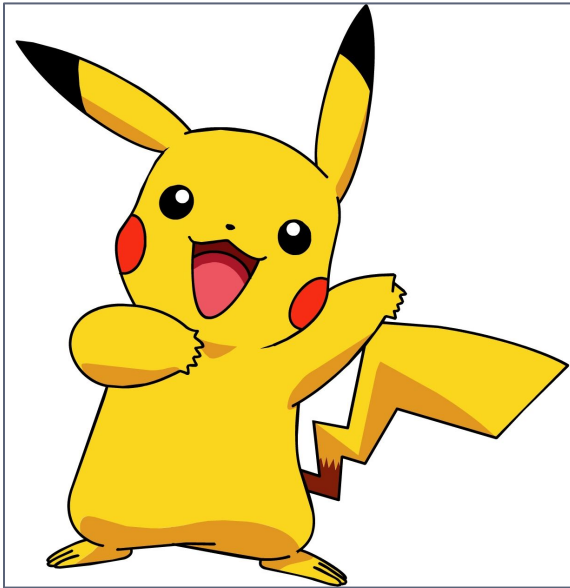
```
SOURCES          = main.c mylib.c font.c palette.c Pikachu.c
```

Demo Actually using

PikachuBitmap from Pikachu.c is a series of shorts, it actually is "easier" (certainly much faster) to use that way (when doing DMA), but I'm using it byte by byte for the purposes of this demo.

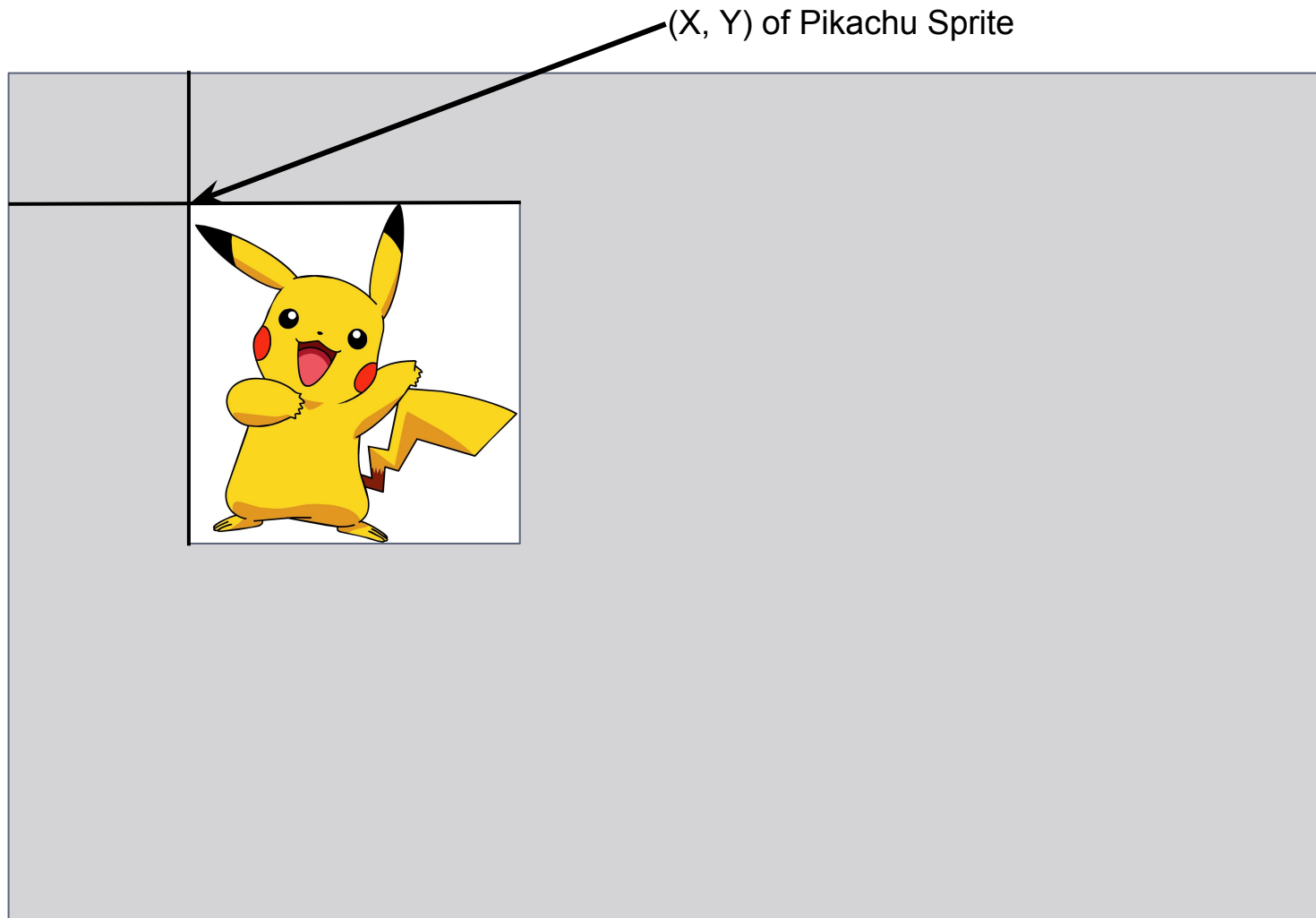
Sprites

This is the relative ranking of Pokémon favorites for Quiz 2 (by size):



You voted for 29 different Pokémon, but all but 4 got a single vote.

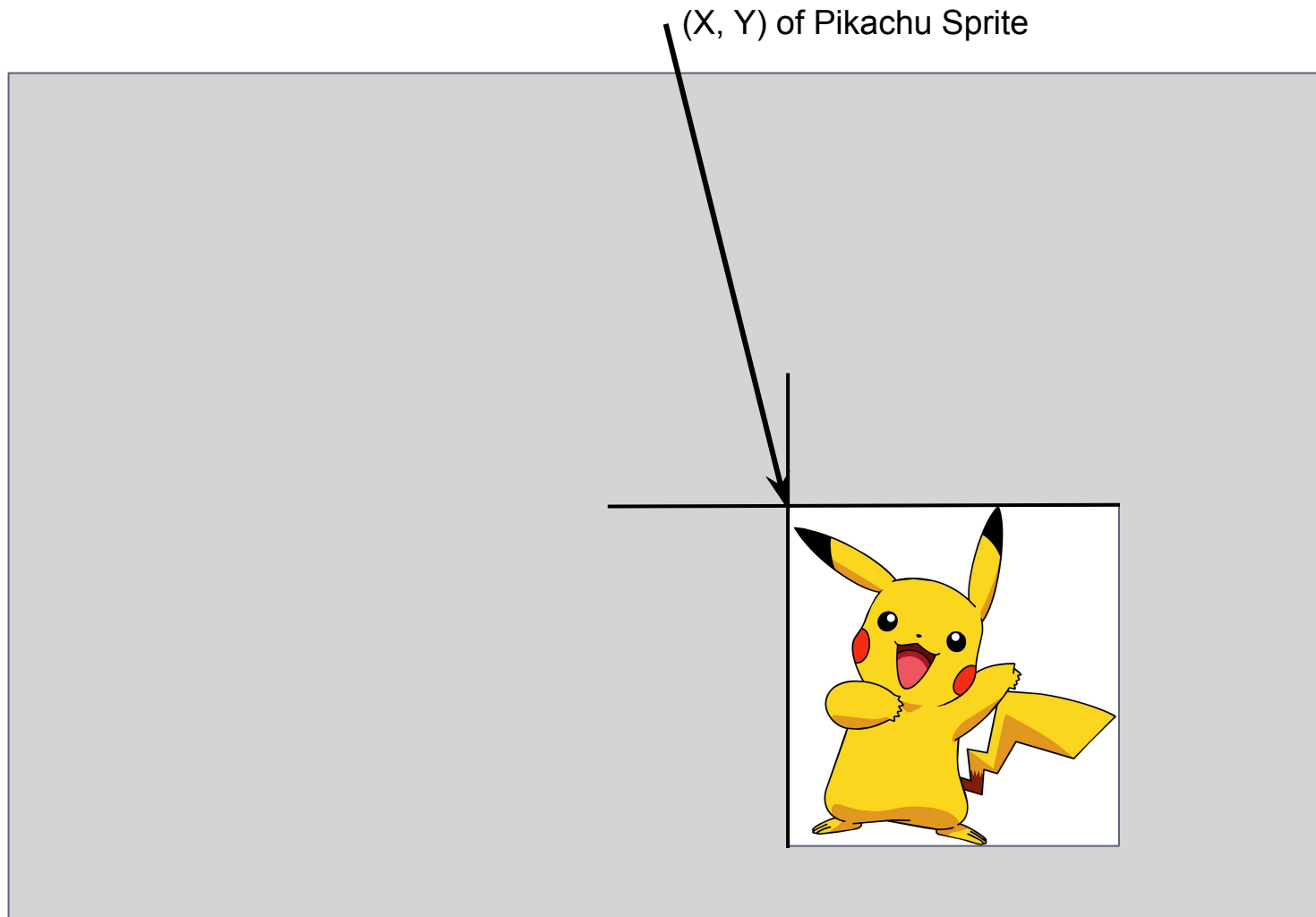
Pikachu Sprite



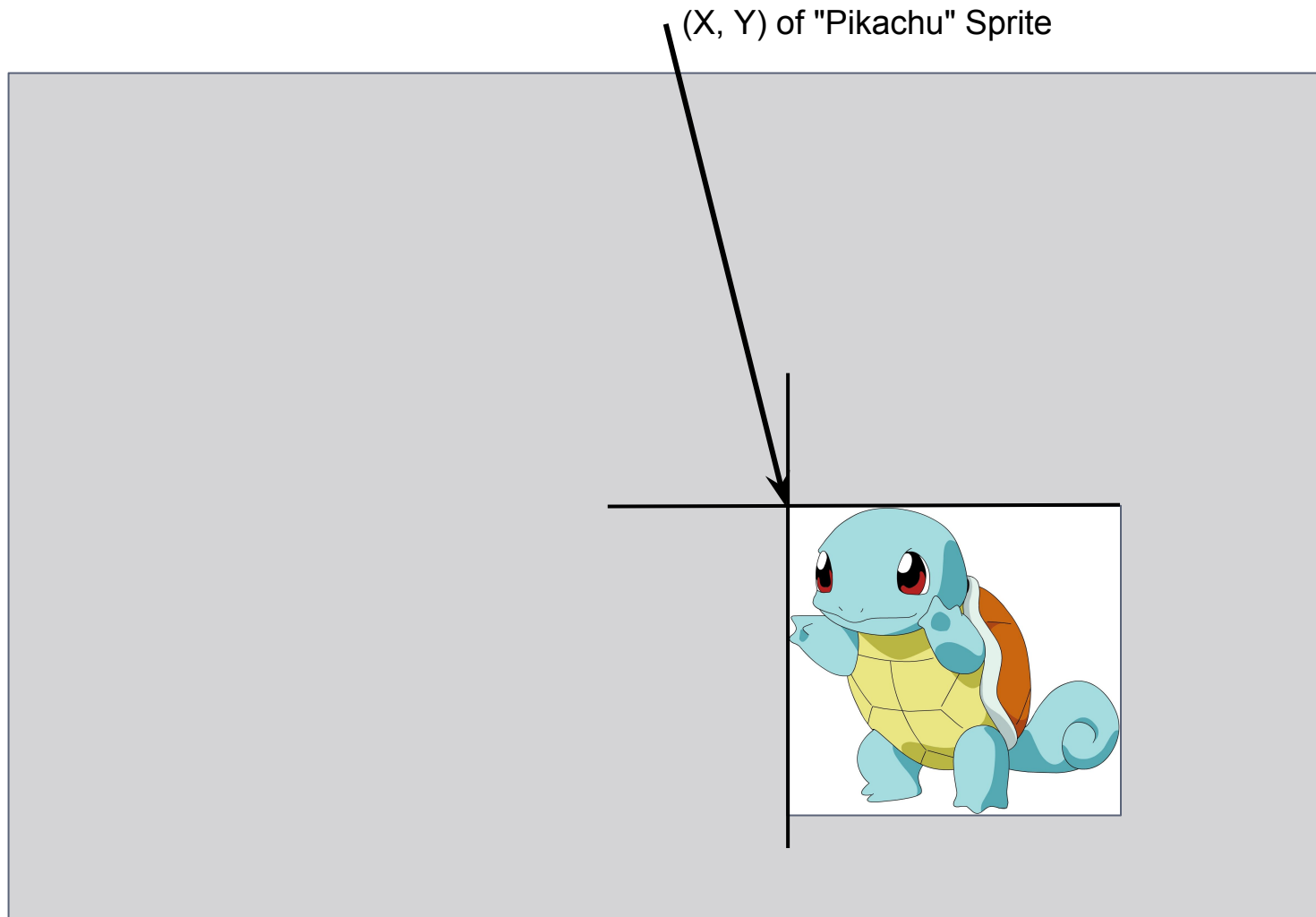
Sprites

- These days, "sprite" is a generic term for an image that can be placed into a larger scene.
 - Originally, sprites were images that could be inserted without disturbing the scene. In this sense, they are implemented with hardware.
 - GBA sprites fall under the original definition
 - They sit "on top of" the background behind them.
 - They can move independently of the background!
 - Examples of sprites:
 - Balls, paddles, pacmen, pikachus...
 - Basically anything that's not part of the background is likely best implemented as a sprite.

Moved within the scene



Replaced by Updating Underlying Memory

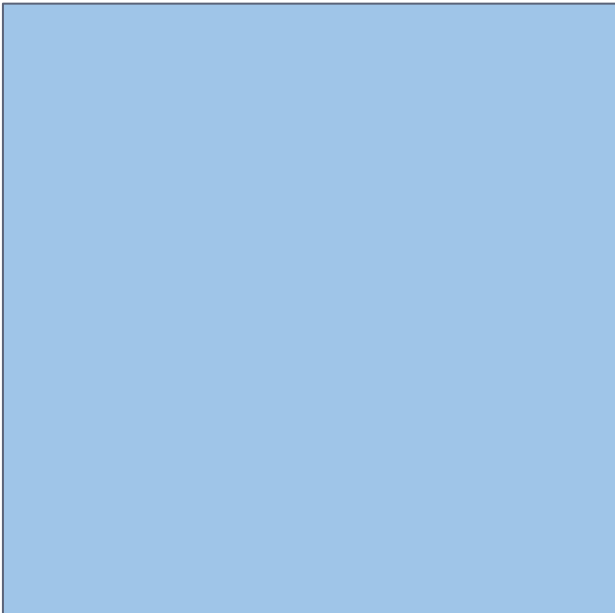


GBA Sprite Overview

- Standard component of games
 - Basically anything that's not part of the background should probably be a sprite.
- Composed of 8x8 tiles
- Hardware-supported movement and drawing (including transforms) that doesn't change the background.
- The GBA supports up to 128 sprites at a given time, ranging in size from 8x8 to 64x64 pixels.
- Sometimes referred to as "video objects", especially in GBA documentation (see some of the following slides).

Size comparisons

Single Pixel:  Smallest Sprite:  8x8

Largest Sprite:  64x64

Screen Size: ~20% wider than this entire screen, at this scale.

Sprite sheet

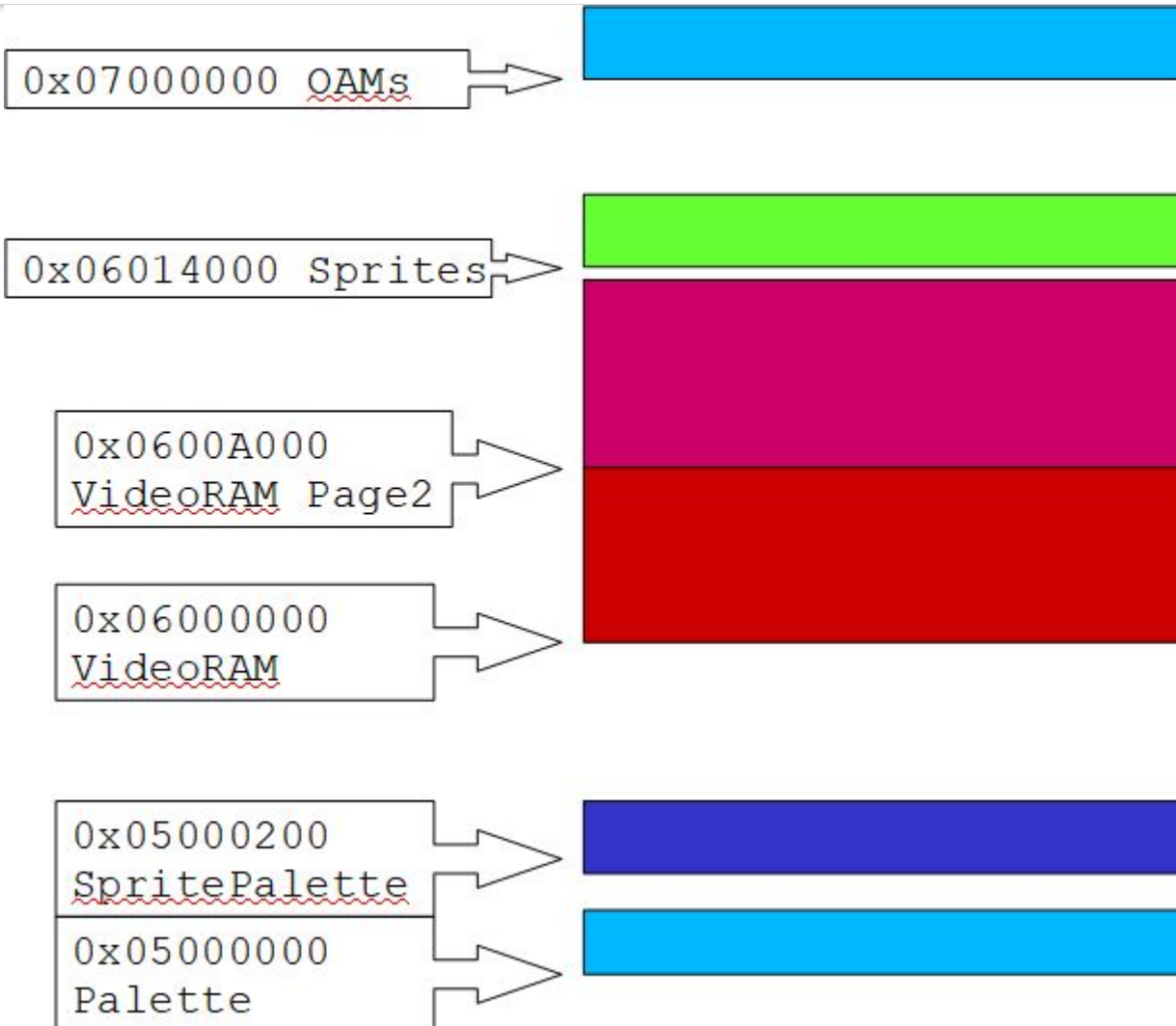
- Often, a mapping of many sprites are combined into a single bitmap
 - Typically for space/performance reasons.
 - Can also help keep related sprite values together, logically (as with sprite animations).
- Such a grouping is typically called a "sprite sheet" (also sometimes called a "texture atlas").



Essential Sprite Steps

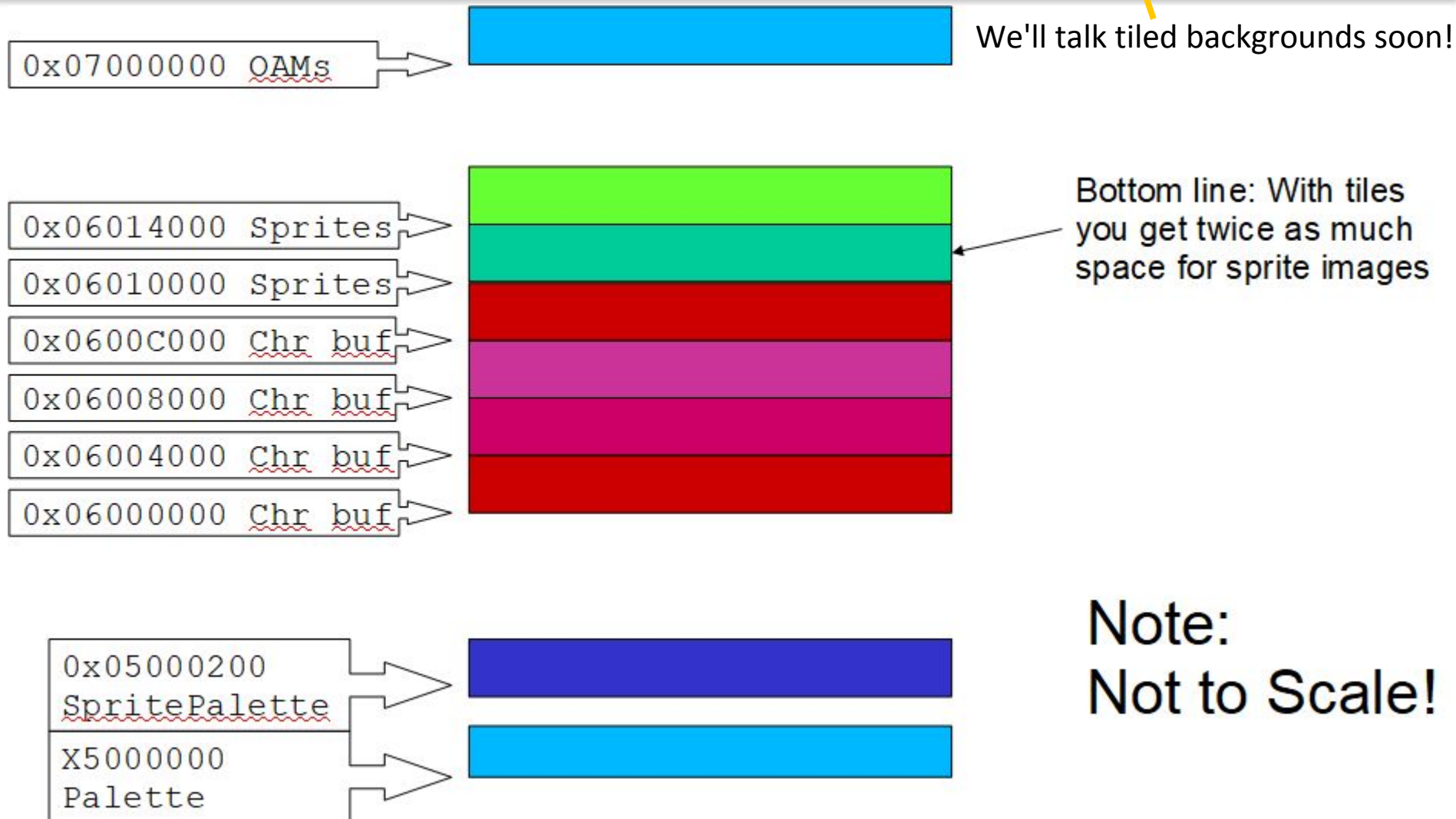
- Load sprite palette into sprite palette memory
 - Just after the Mode 4 pallet at 0x0500200
- Load sprite graphics into OVRAM (Object Video RAM)
- Set sprite attributes in OAM (Object Attribute Memory)
- Enable sprite objects and select correct mapping mode in REG_DISPCNT

Video Buffer Layout for Bitmap Display Modes & Sprites



Note:
Not to Scale!

Video Buffer Layout for Tiles Display Modes & Sprites



Sprites are composed of Tiles, not just bitmaps

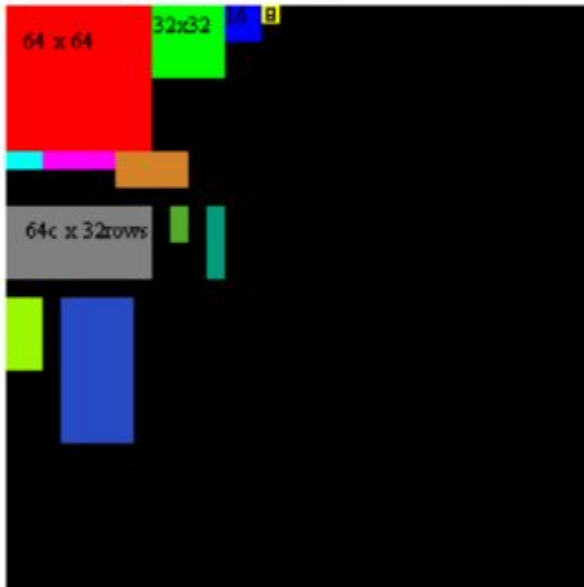
- Tiles themselves have either 4bits per byte (16 colors) or 8 bits per byte (256 colors)
- How many tiles and at what bpp is best?
- Sprite sheet of 32 tiles x 32 tiles = 1024 total tiles
 - 1024 x 64 pixels/tile = 65536 total pixels
 - With 8bpp that's 65536 bytes
 - With 4bpp that's half or 32768 bytes
 - That's 32KB, which is the size of Sprite Memory
 - So a 32 tile x 32 tile sprite sheet at 4bpp fits nicely (tightly) into
- Usenti will still export as shorts

Color Modes

- 4bpp
 - 16 colors
 - 32 bytes per 8x8 sprite (smallest)
- 8bpp
 - 256 colors
 - 64 bytes per 8x8 sprite
- For most applications, 16 color sprites are preferred
- - Tiles are smaller
 - Sprites are smaller
 - More to work with!

Visually

- Sprite image area 16 color sprites (4bpp)
- 256x256 (pixels)



- Sprite image area 256 color sprites (8bpp)
- 128x256 (pixels)



What is...

- A variable?
 - A symbol representing an address where we store something.
- An array variable?
 - A symbol representing an address where a block of memory has been reserved.
- A function name?
 - A symbol representing the address of a piece of code which we can jump to (and we normally expect to contain code that will return control back to the caller).
 - Remember, code is in memory too! (Von Neumann)