# Javascript and Hardware

Kristina Durivage
@gelicia

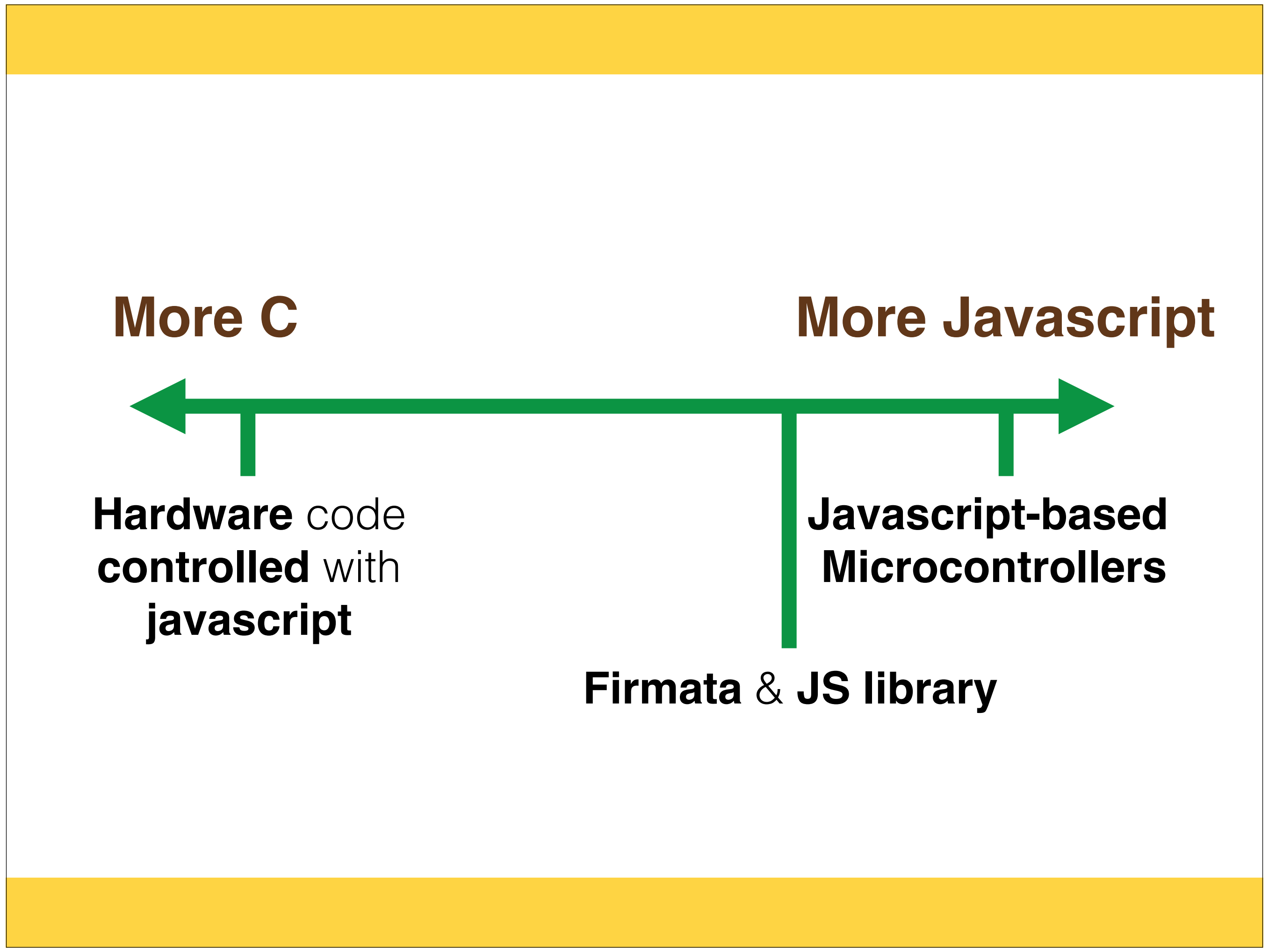
## Why?

+ **Abstraction**

+ **Use a language you already know**

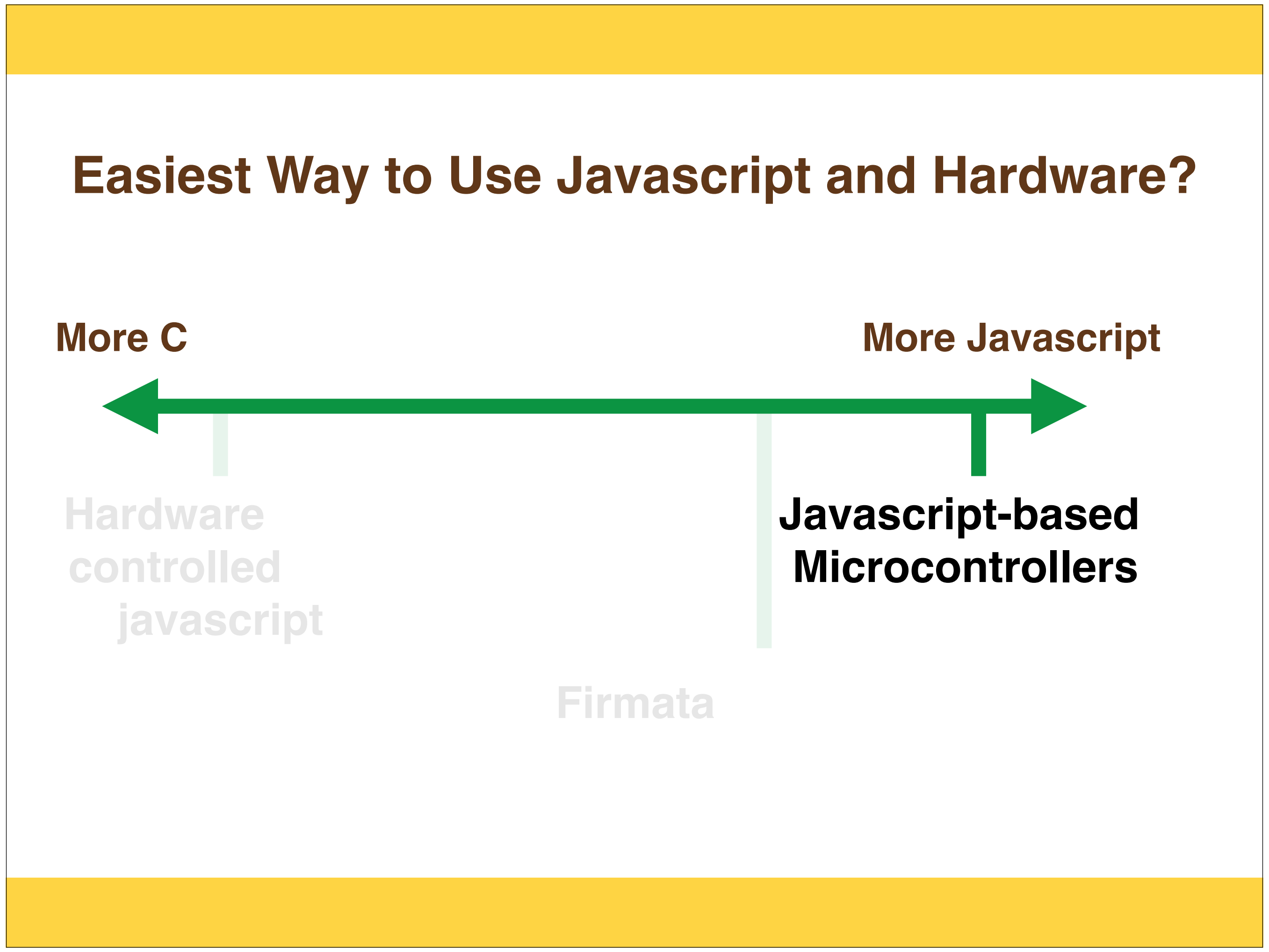
- Different ecosystem

- **Not production ready**


More C
More Javascript
Hardware code controlled with javascript
Javascript-based Microcontrollers
Firmata & JS library

What are our options for working with hardware and javascript

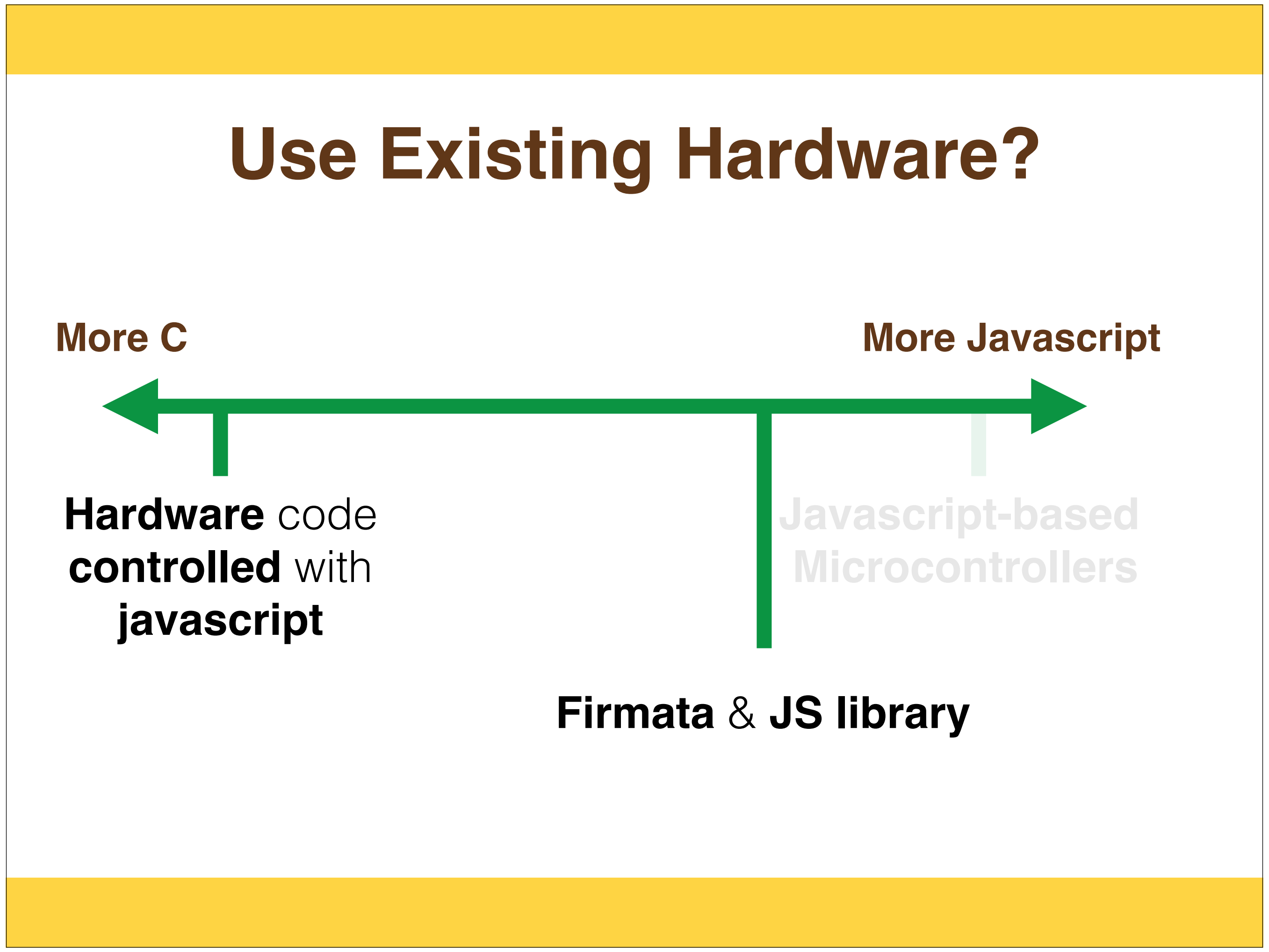
Firstly, if you're not too familiar with the arduino environment, the programming language used for it is called Wiring, which is a variant of C. C is pretty low level and can be hard to get into if you're used to javascript, and I'm guessing since you're here, a lot of you would like to do as much javascript as possible.



## Easiest Way to Use Javascript and Hardware?

More C
More Javascript
Hardware controlled javascript
Firmata
Javascript-based Microcontrollers

The easiest way to use Javascript and hardware is to buy a special board and use that ecosystem to program. The Espirino and the Tessel are both boards that do that. I will be doing an example with the Tessel today.



## Use Existing Hardware?

More C
More Javascript
Hardware code controlled with javascript
Javascript-based Microcontrollers
Firmata & JS library

Lets say that you don't want to use a special board, you want to use the hardware everyone else is using. There's a couple different options for you.

The first is what most people I know think of when they think Javascript and hardware. It involves using libraries to allow javascript to talk to the arduino over USB or wireless. The two libraries I know of for this are johnny-five and cylon.js. I'll be showing an example with Johnny five since it's the one I know more people are using and there doesn't seem to be much of a difference besides syntax.
The second is what I've done with some of the wearable clothing I make. You program the hardware part in Wiring, but use a board to expose the functionality to the web, and interact with it with a webserver. This is using the Particle ecosystem, and I'll be giving a demo with that last.

24 x **Neopixel** (WS2812) Ring

$20 from **Adafruit**
http://www.adafruit.com/products/1586

$9 from **Ebay**

LEDs are **RGB** (any color)

**Individually addressable** (can turn any one on or off)

Covered with a 3d printed ring for diffusion

I wanted to talk a bit about what I'll be using for these next two demos. This is a ring of 24 LEDs that can be any color, and are individually addressable, meaning I can turn any one of them to any color or off. If you'd like to play with one, I'd recommend you check out the documentation on Adafruit and buy from there if it helps you, but if you know what you're looking for ebay has them for less than half the price.

Some quick information about the LEDs - these kind, and probably the most popular LED out there for people who want one LED that can change color. The chip is the WS2812, but they were branded "neopixel" by Adafruit.

These LEDs are connected to each other in a chain, and the chip in them is so smart, I can tell any LED in that chain to be any color with just one set of wires. One wire for power, one for ground, and one for data in.

I'm using this as an example because I like LEDs, but this isn't necessarily a happy path example. These LEDs work on only one wire of data, so the arduino needs a special library to translate instructions on that one wire so the LEDs know what color to be or if they should be off. But, I think it's worthwhile to explore the non happy path for this, but know that a lot of other parts of hardware might be easier to use.

## Javascript-Based Microcontrollers

Let's start by talking about the most straightforward way to program hardware with Javascript - Javascript based microcontrollers. This involves buying a special board that combines with a library to let you program in javascript. You run the code using a special command that will translate that code to something that board can understand, then push that code to the board.

**Tessel**

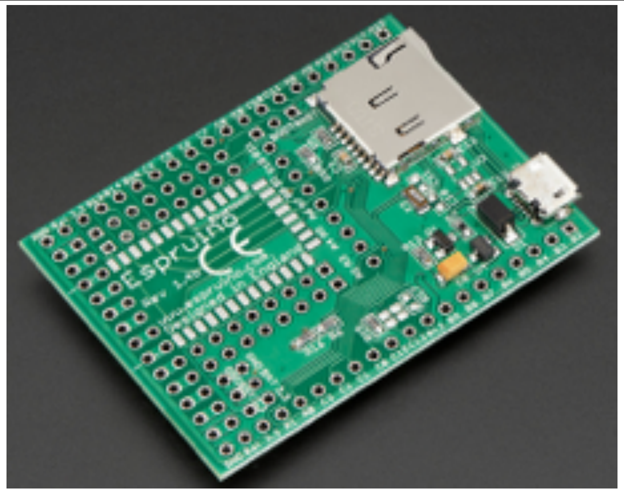
**$35** preorder for Tessel 2
(Original Tessel not in their shop)

Use **npm**

Has wifi

**Non standard pins**

https://tessel.io/



**Espruino**

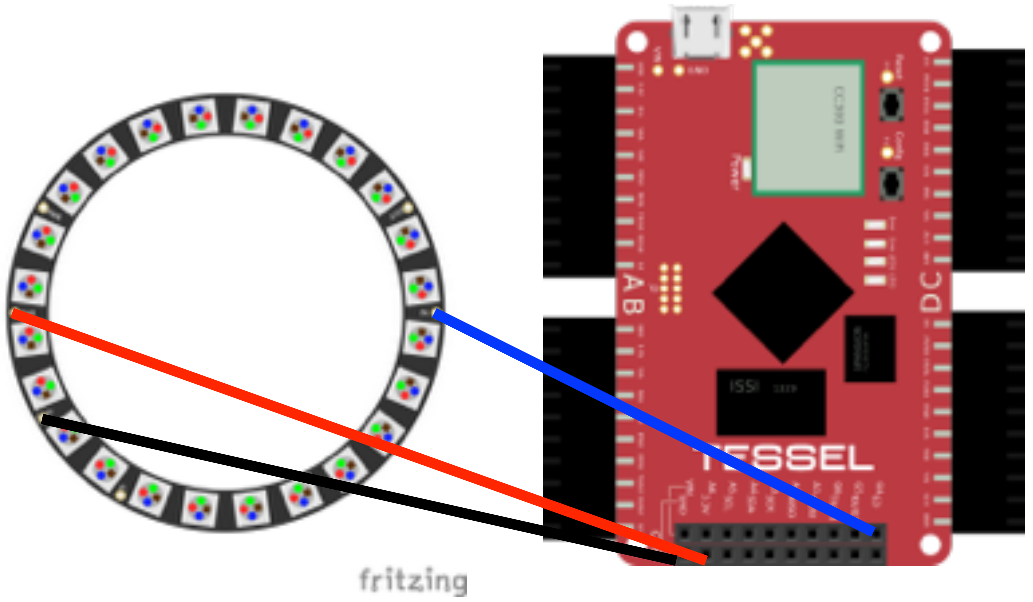
**$40** ($29 for mini version)

More **complex**
(you have to set up what bits to send)

**Standard pins**

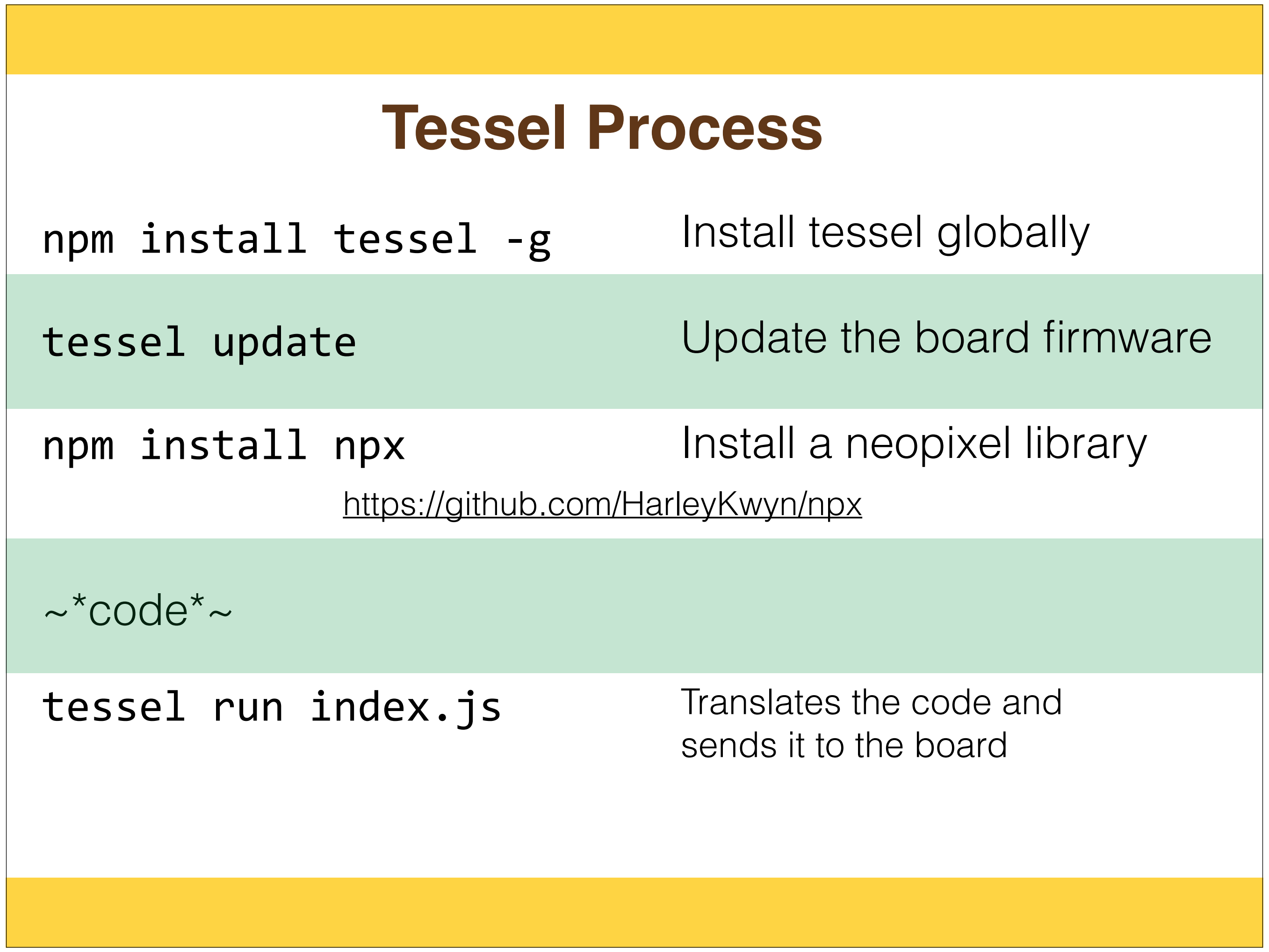
http://www.espruino.com/

There are two main options here that I've heard of - the Tessel and the Espruino. The downside of the Tessel is you're supposed to use their modules to run things as opposed to the standard pin set that the Arduino provides, and the downside to the Espruino is you have to set up what to send pins yourself which doesn't provide the level of abstraction I'd want. I'll be showing a demo with the Tessel today because I don't have an Espruino. Another side note though, is the Tessel I'm showing isn't available anymore it seems like, and the Tessel 2 is out in September and available for preorder now.



**PWR** to **3.3V**
**GND** to **GND**
**DI** to **G3 on GPIO**

Firstly, this is how you would wire it up. I mentioned the Tessel doesn't have standard pins like an arduino - you are meant to buy tessel modules that plug in to the four sockets. However, since neopixels are very popular, someone has figured out how to make it work. You need to wire them up this way to have it work with the library - the library doesn't allow for specification of pins.



## Tessel Process

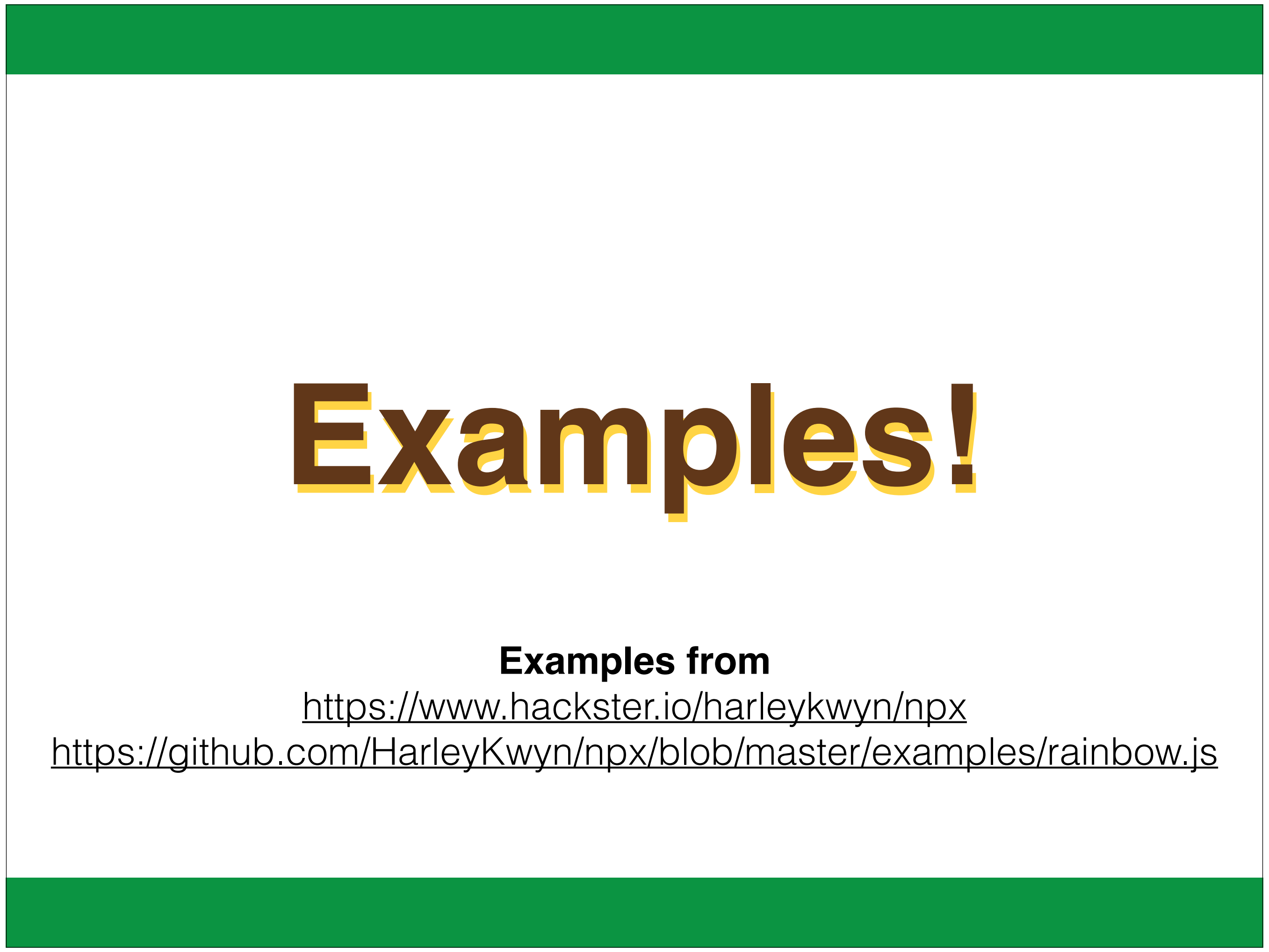
| | |
|---|---|
| `npm install tessel -g` | Install tessel globally |
| `tessel update` | Update the board firmware |
| `npm install npx` | Install a neopixel library |

https://github.com/HarleyKwyn/npx

~*code*~

| | |
|---|---|
| `tessel run index.js` | Translates the code and sends it to the board |

The general process for programming with the tessel is the following. Firstly, you need to install the tessel package globally on your computer. Then you update your board's firmware if you haven't done anything with it for a while. Install the library you want to use - in this case, someone has written a very nice neopixel library for the tessel called npx. Finally, tessel run will have the tessel package look for a tessel connected on USB, take care of any code manipulation and send the code to it.



## Examples!

**Examples from**

https://www.hackster.io/harleykwyn/npx

https://github.com/HarleyKwyn/npx/blob/master/examples/rainbow.js

Examples are here: https://github.com/gelicia/midwestJSHW/tree/master/tessel-example

+ **All in One** Solution

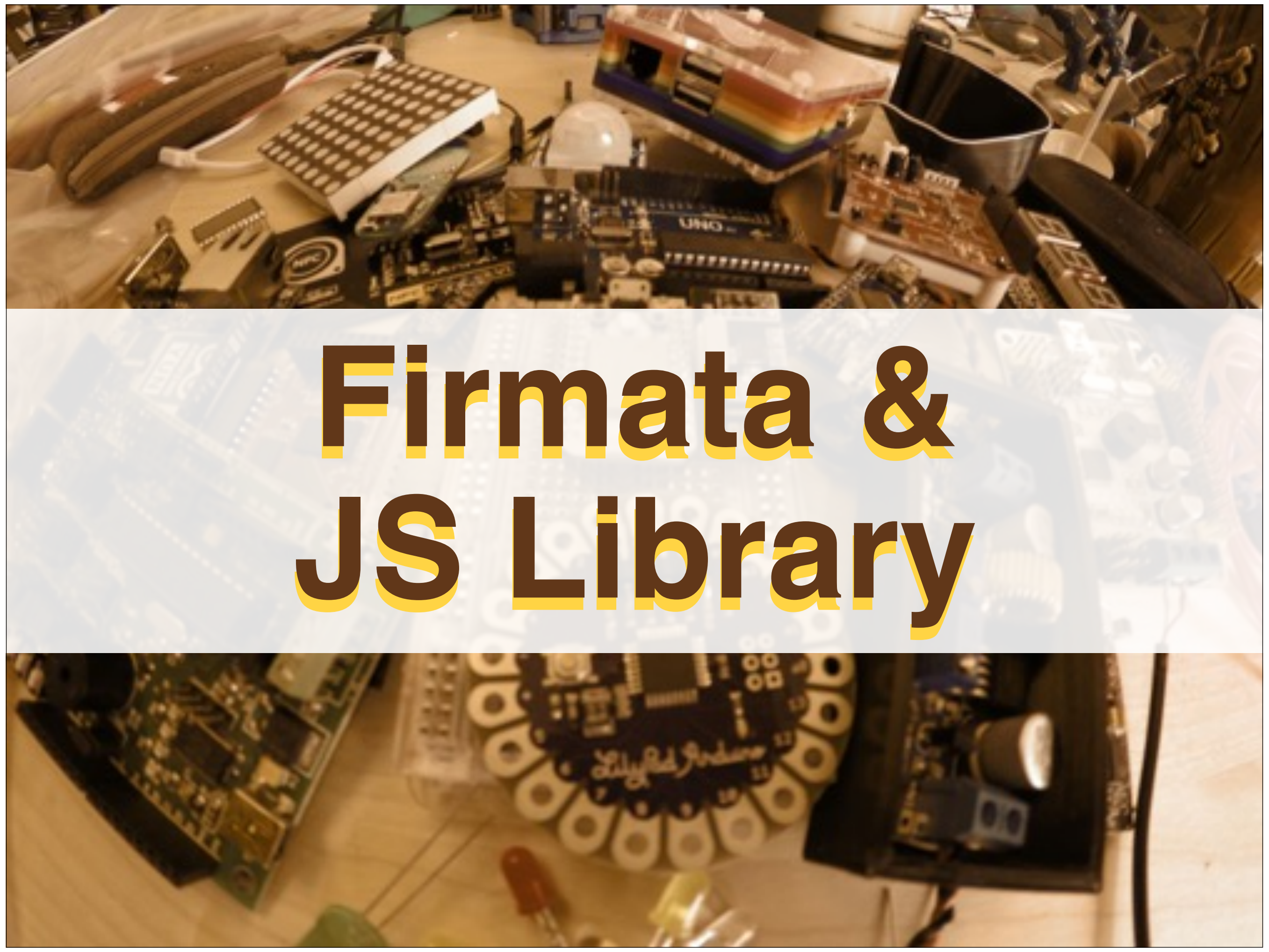
+ **Easy** to get up and running

+ Tessel has wifi!

+ Easy to **communicate with the web**

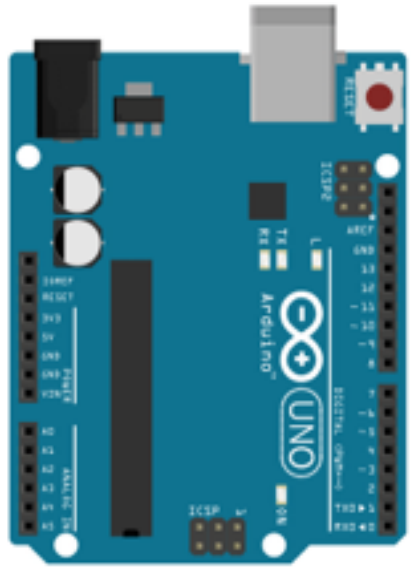
- Requires **special hardware**
(both of which aren't great)

- **Locked** into one ecosystem



## Firmata & JS Library

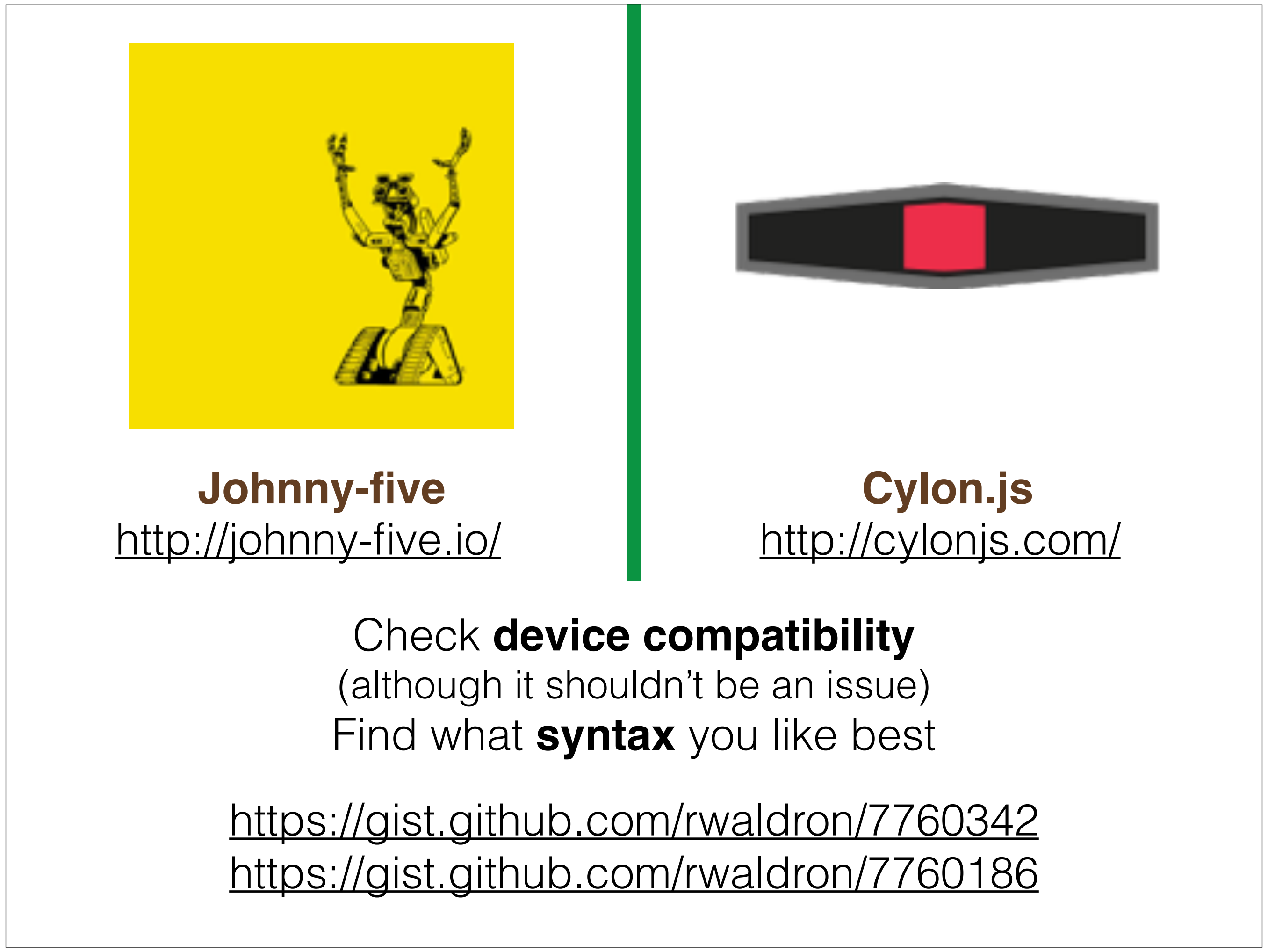
Next, I'm going to talk about Firmata & JS libraries that will let you use your existing hardware and program it with javascript.



$$\begin{tikzcd}\text{\textbf{JS Library} Abstracting Firmata} & \text{node-serialport} \arrow[l] \arrow[r] & \text{Firmata}\end{tikzcd}$$

Using cats to liven up the process of explaining this shamelessly taken from Suz Hinton, link at the end :3

Firstly I’ll go over how this is possible.
To begin with, there’s a protocol called firmata designed to let microcontrollers talk with non-microcontrollers such as computers or mobile devices. This part is still very low level and a bit over my head so I’m not going to go into it too much. The firmata protocol was implemented in Wiring, the language for the Arduino. You upload that program to the Arduino and it can start understanding commands in the firmata protocol sent to the board.
Secondly, theres a npm package called serialport which will let you send and receive data over USB to the arduino. So we can send firmata instructions from javascript to the arduino using that.
Thirdly, there are javascript libraries that make it so we don’t have to communicate in firmata’s protocol directly. It abstracts the bits and bytes of the commands to something easier for us to program in, translates it to firmata protocol, sends it over with serialport, where the firmata program running on the arduino receives it, translates it into instructions the microcontroller understands, and everything works.



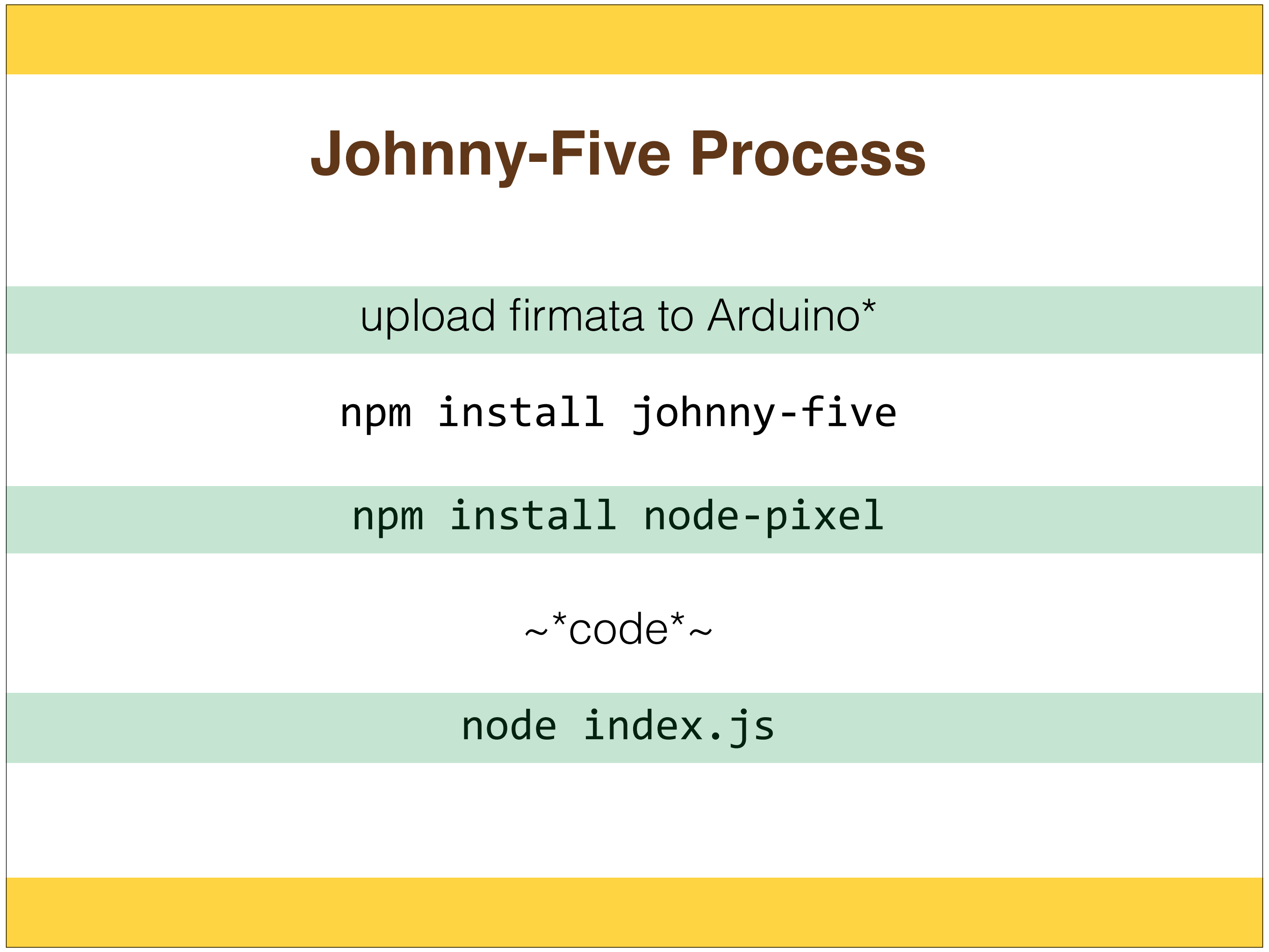
**Johnny-five**
http://johnny-five.io/

**Cylon.js**
http://cylonjs.com/

Check **device compatibility**
(although it shouldn't be an issue)
Find what **syntax** you like best

https://gist.github.com/rwaldron/7760342
https://gist.github.com/rwaldron/7760186

So now you understand how it works, lets look at some options for javascript libraries that do the part you'll be directly interacting with. The two main ones are Johnny Five and Cylon.JS. Both of them do the same stuff with a slightly different syntax, but most people I know use Johnny Five, so thats what my example will be using. The author of johnny-five has a couple gists comparing the syntax of the two libraries.



## Johnny-Five Process

upload firmata to Arduino*

```
npm install johnny-five
```

```
npm install node-pixel
```

~*code*~

```
node index.js
```

The process goes like this, but again, there are some added complexities due to my using neopixels for this example. For one, you can’t upload the standard firmata program that comes with the arduino IDE - you need to use a special version that has the libraries packaged with it. You can find this in the folder for the node-pixel library under firmware.
We begin by uploading that to the arduino. Like I said, this is special for neopixels and if you’re doing a robotics project you won’t need to do this.
Secondly, we install johnny-five and the node-pixel libraries to run both the framework and the neopixel library on the computer side. Notice you do not install this globally - you run the program like a normal node program.
You code everything up using the node-pixel API to write to the LEDs
Run it all like a regular node program.

## Examples!

**First Example from**
https://www.npmjs.com/package/node-pixel/

https://github.com/gelicia/midwestJSHW/tree/master/johnnyfive-example

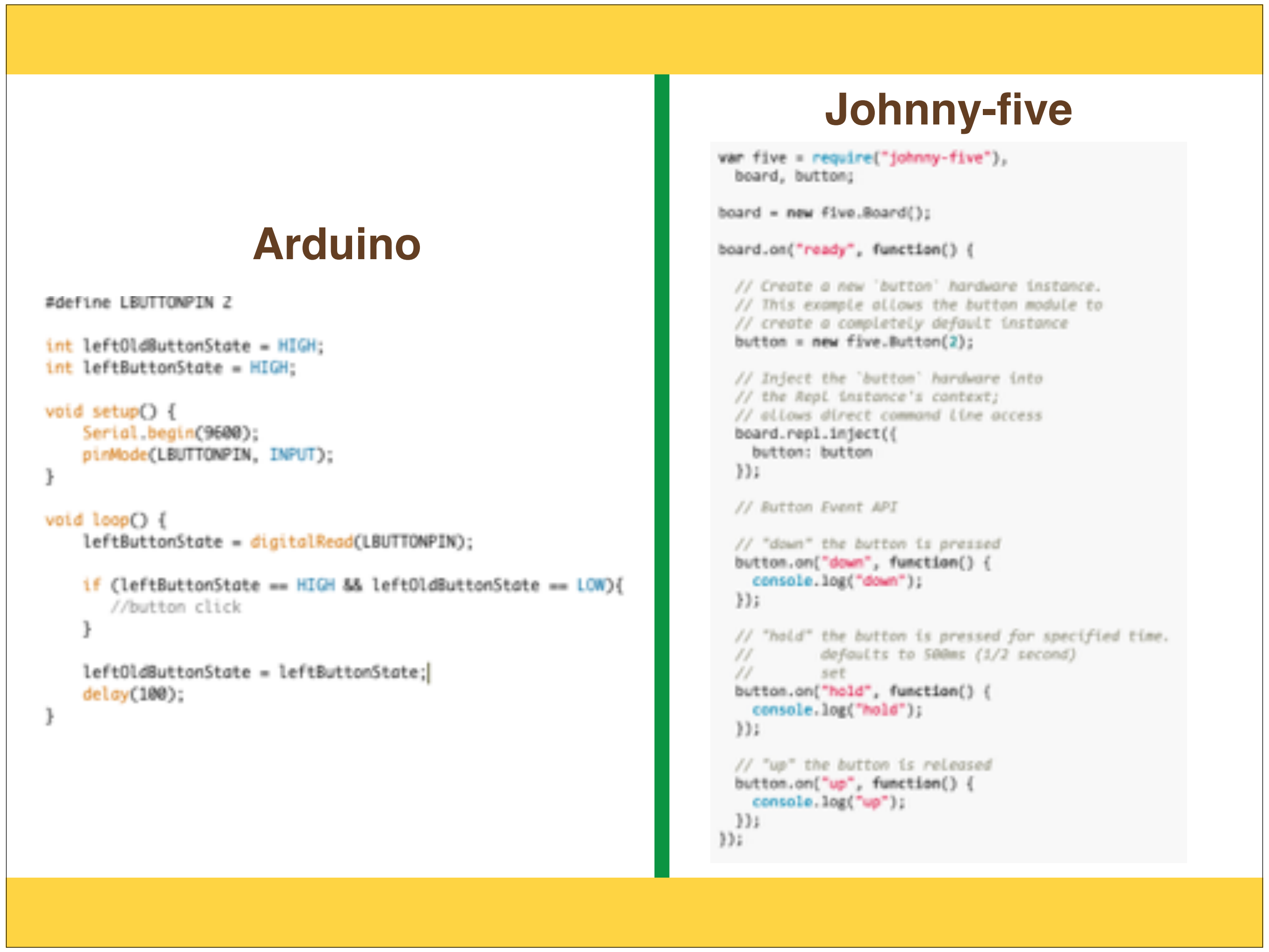
+ **Great for robotics**

+ **Easy** to get up and running

+ Easy to **communicate with the web**

- Uploading firmata is an **extra step**

- **Locked** into one ecosystem



## Arduino

```
#define LBUTTONPIN 2

int leftOldButtonState = HIGH;
int leftButtonState = HIGH;

void setup() {
    Serial.begin(9600);
    pinMode(LBUTTONPIN, INPUT);
}

void loop() {
    leftButtonState = digitalRead(LBUTTONPIN);

    if (leftButtonState == HIGH && leftOldButtonState == LOW){
      //button click
    }

    leftOldButtonState = leftButtonState;
    delay(100);
}
```

## Johnny-five

```
var five = require("johnny-five"),
  board, button;

board = new five.Board();

board.on("ready", function() {

  // Create a new `button` hardware instance.
  // This example allows the button module to
  // create a completely default instance
  button = new five.Button(2);

  // Inject the `button` hardware into
  // the Repl instance's context;
  // allows direct command line access
  board.repl.inject({
    button: button
  });

  // Button Event API

  // "down" the button is pressed
  button.on("down", function() {
    console.log("down");
  });

  // "hold" the button is pressed for specified time.
  //        defaults to 500ms (1/2 second)
  //        set
  button.on("hold", function() {
    console.log("hold");
  });

  // "up" the button is released
  button.on("up", function() {
    console.log("up");
  });
});
```

I wanted to talk a bit about one area where javascript abstraction is particularly helpful - with events. In arduino programs, there are two functions, a function to setup everything and a loop function that runs as fast as it can. Because that loop function runs super fast, it can get tricky to debounce buttons and make the events from inputs happen like you might want. Johnny five and Cylon both allow you to react to events more like you would with other javascript libraries, and abstract away the debouncing or sensitivity problems you might have with a loop function that's running really fast.

## More Resources

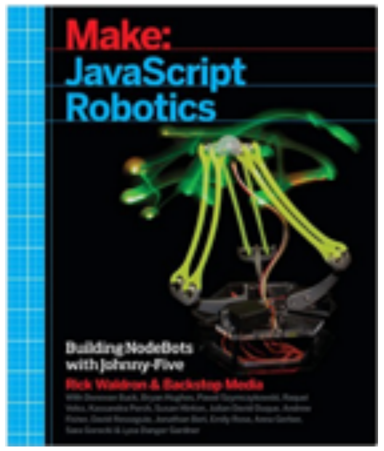
Suz Hinton **@noopcat** at **Web Rebels**

Explaining the inner workings to get hardware and Javascript to talk

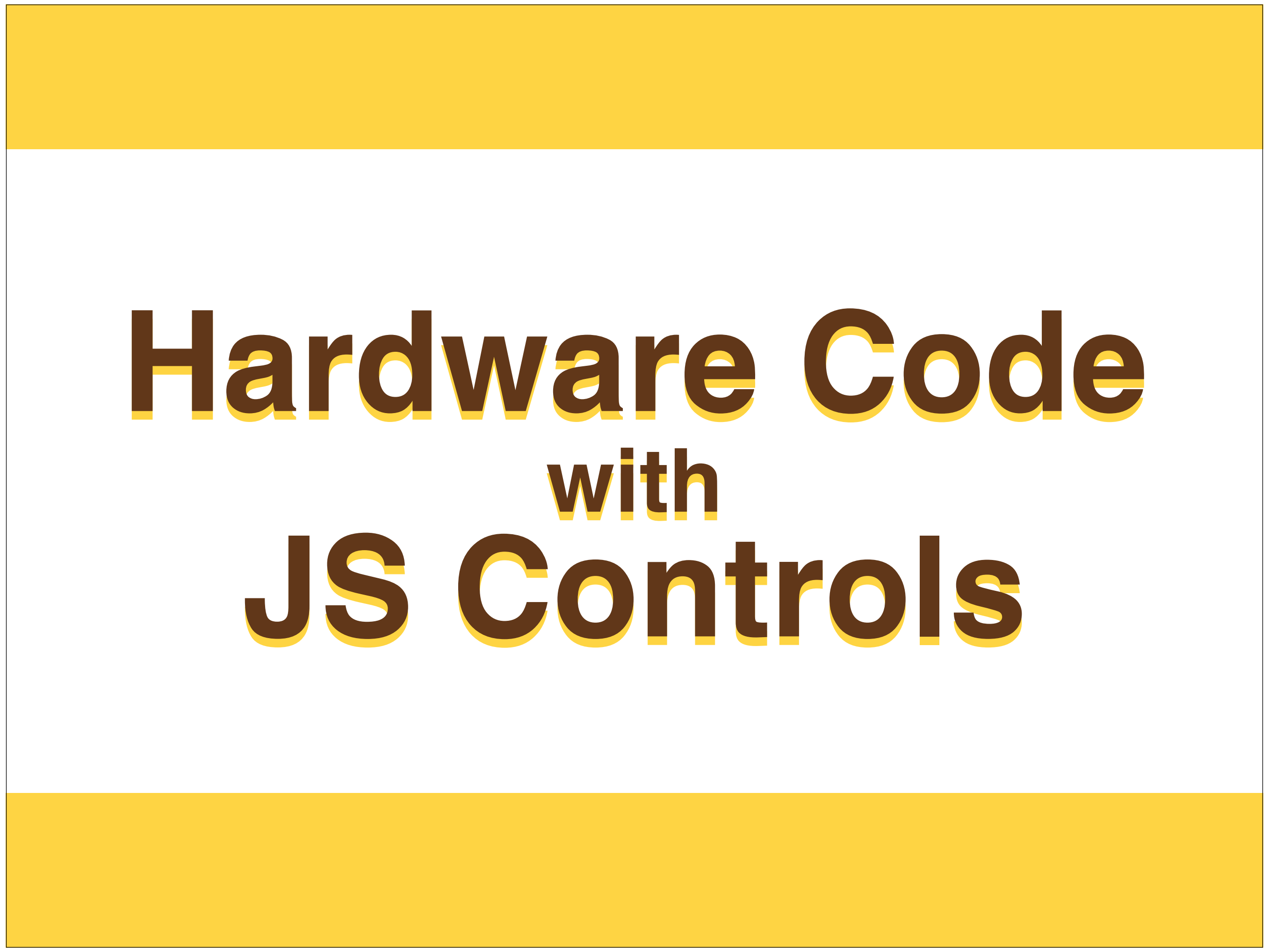
https://vimeo.com/129003513

**Make: Javascript Robotics**

Several hardware projects using Johnny-Five by the author and several others

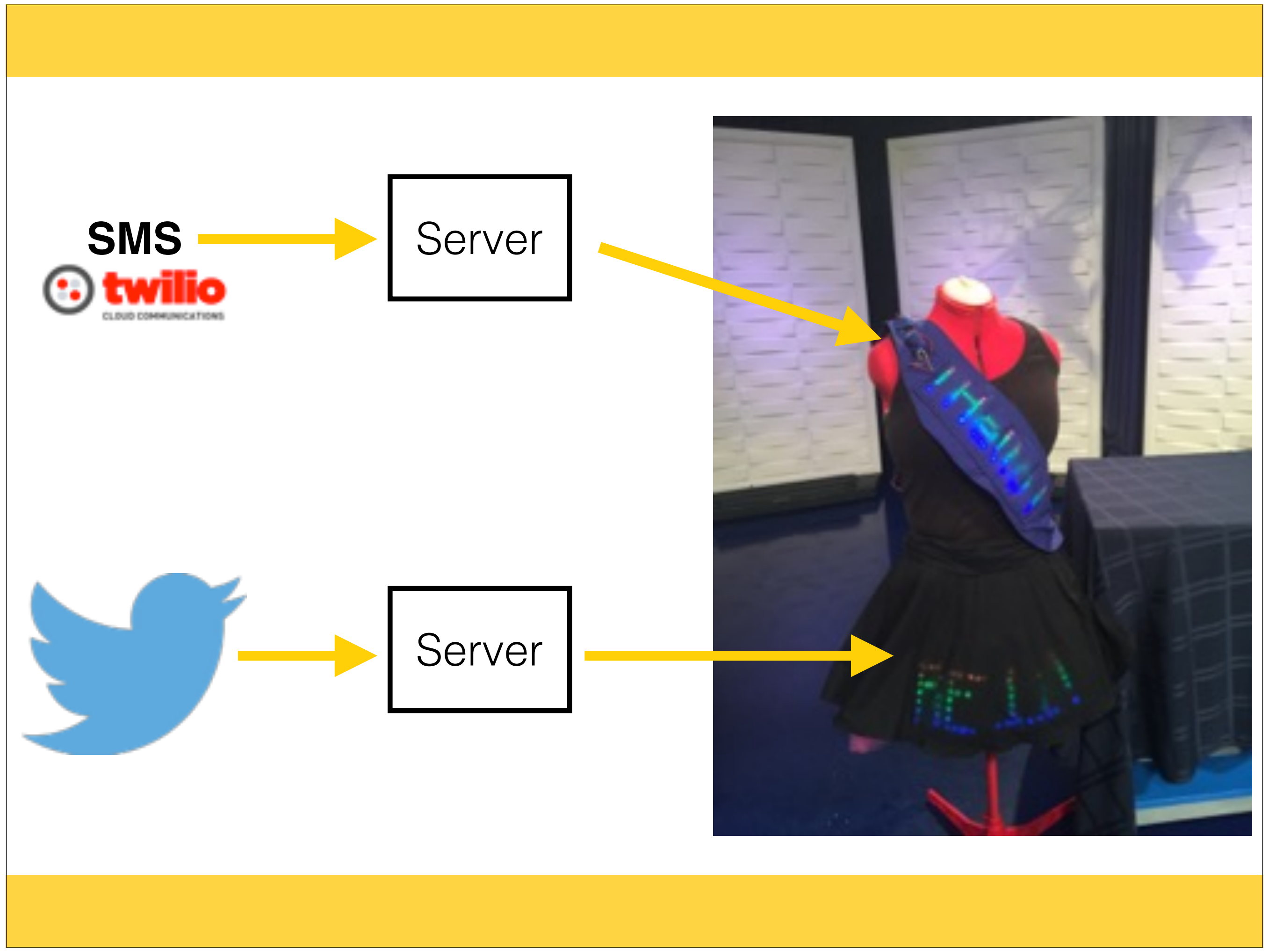

Make:
JavaScript Robotics
Building NodeBots with Johnny-Five
Rick Waldron & Backstop Media

Nodebots is all about making robotics more accessible, so there are resources to help people interested in getting into robotics. For one, Suz Hinton has a great talk about exactly how the arduino works at a hardware level, and the various things involved in getting Javascript to talk to the arduino. It has lots of cute cats, so I'm a huge fan. Secondly, a book of projects was released by Make publishing about javascript robotics by johnny-fives author and others in the nodebots community.

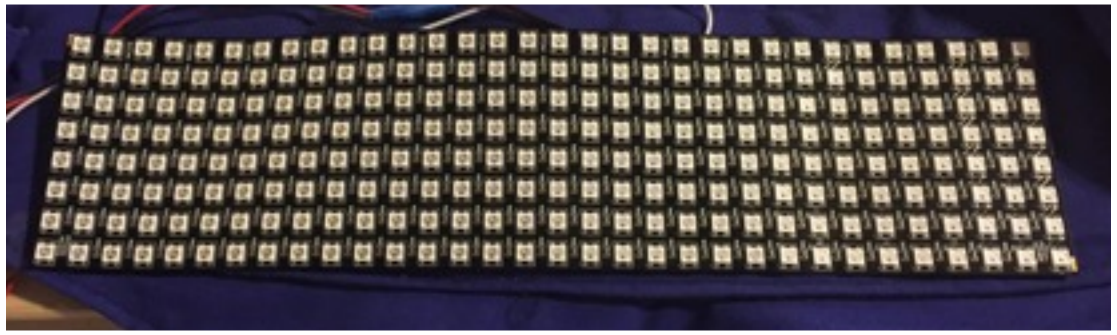


## Hardware Code with JS Controls

Lastly, we're going to talk about how to interact with normal hardware code with javascript. This one is a little less structured and there's a lot of paths it can take, but I'm going to take the one I'm most familiar with.



One of my weird hobbies and why I started getting mixed up in hardware despite being a software programmer is making clothing you can communicate with. I have a sash you can text to and a skirt you can tweet to. This last method is describing how I get that to work, focusing on the part that makes it possible to do with javascript. Because of this, my example will be a bit different - I'll be showing how to hook up the scrolling text code I wrote with IFTTT.

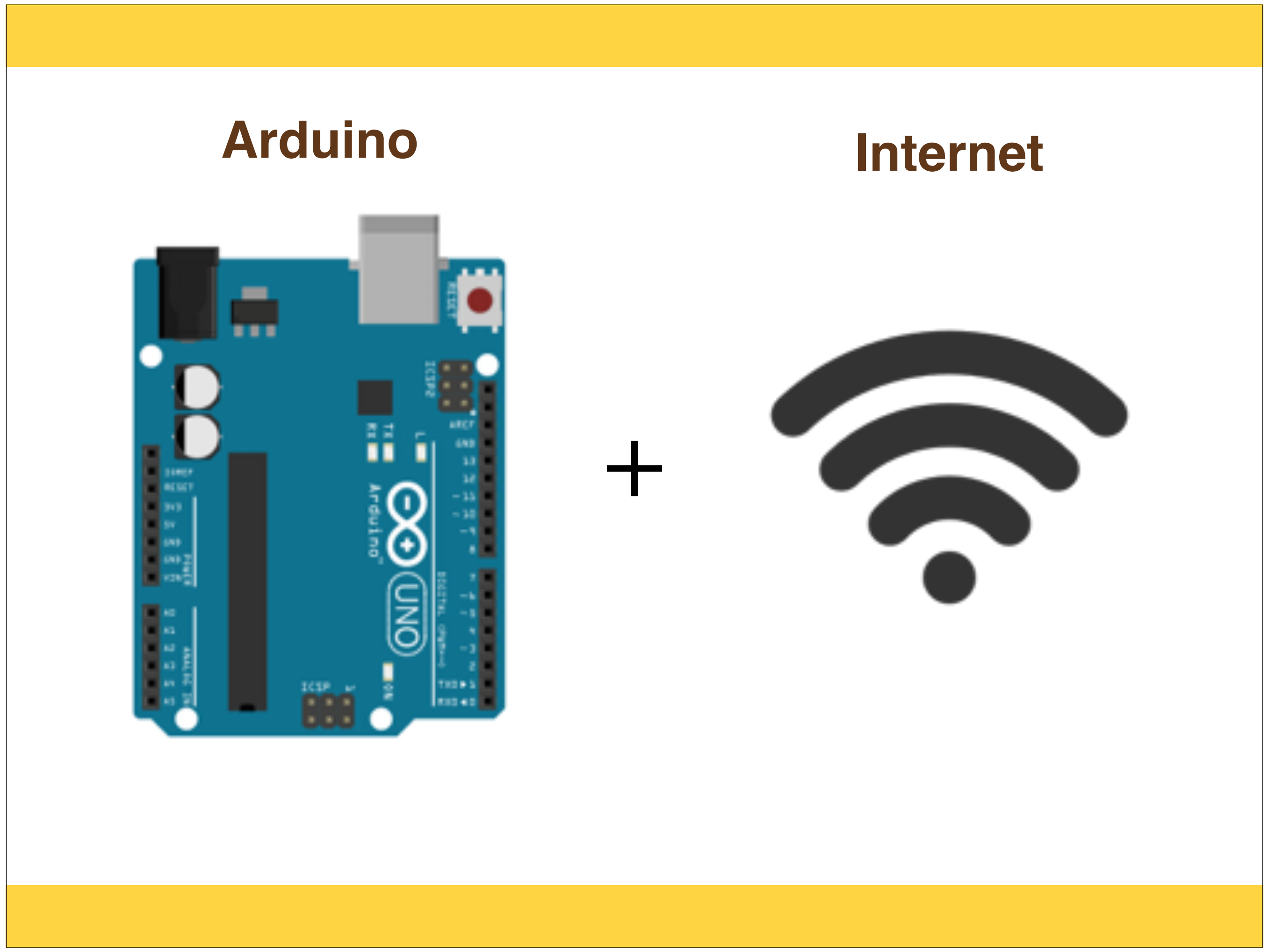


8 x 30 **Neopixel** (WS2812) array

$100 from **Adafruit**
http://www.adafruit.com/products/2294

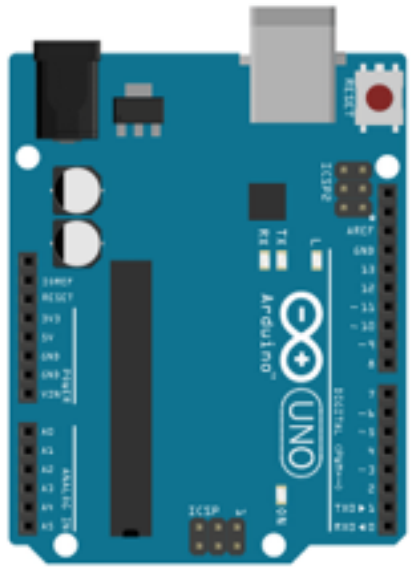
$60-70 from **Ebay**

The next examples have a different set of hardware, so I want to talk about that for a second. Instead of using the LED ring, I'll be using an array of LEDs. The individual LEDs are the same as in the ring, but there's a lot more of them! So I'm going to be uploading a program that I wrote last December that does scrolling code across this array. It's one of those things that I got to work and need some time to figure out how again. I really don't want anyone to write it ever again, it's all open source, so please just look at the source if you ever want it.


Arduino

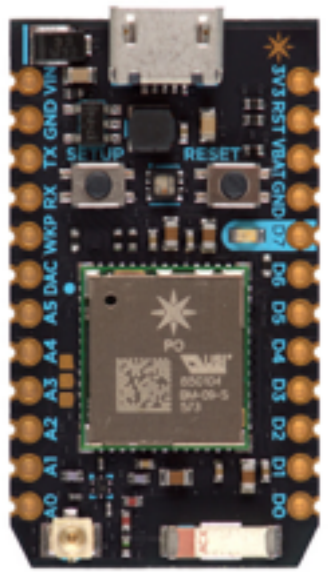
+

Internet



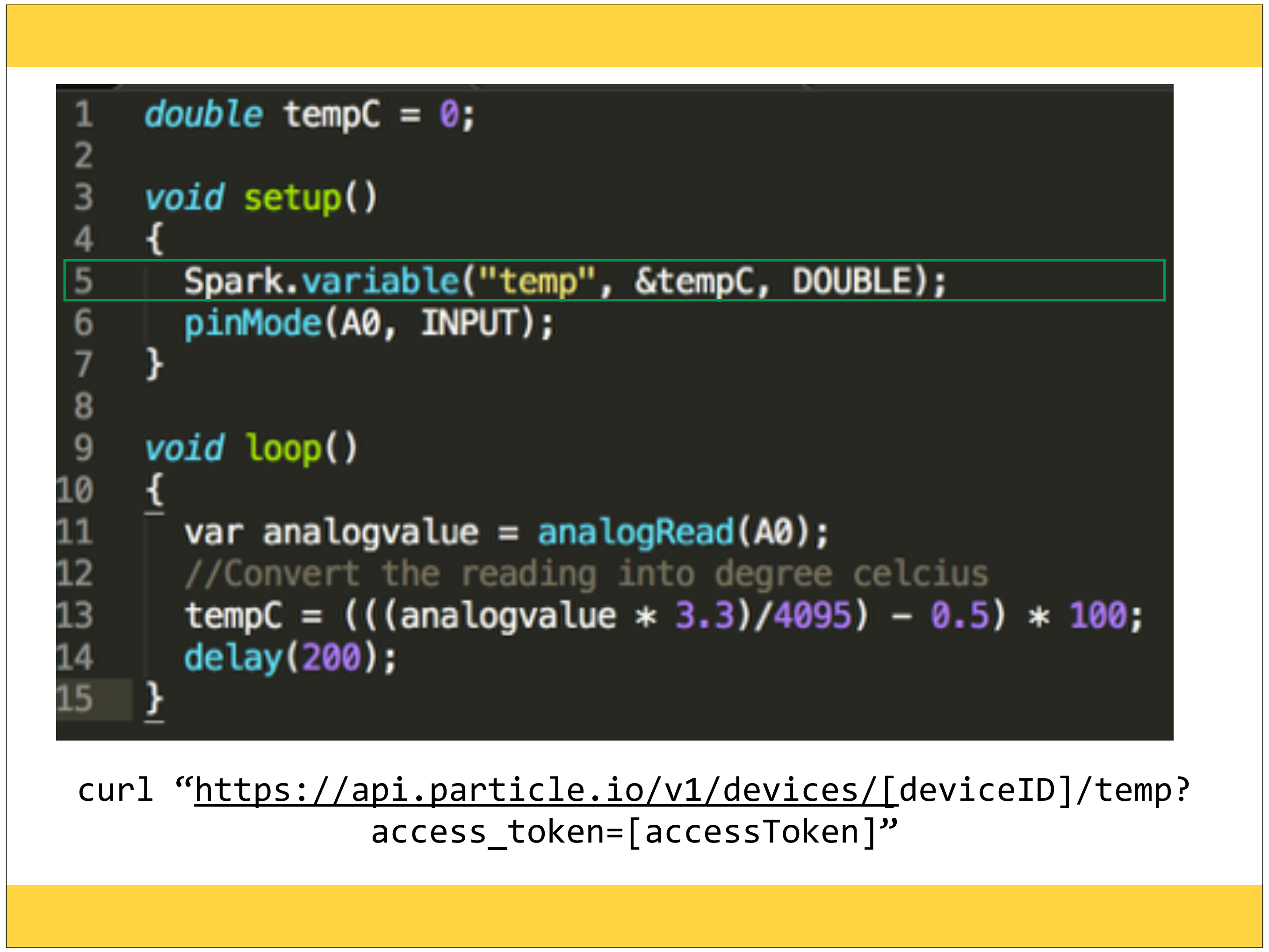
Anyway, this method involves programming the hardware in Wiring, the C-variant that the Arduino uses, but interacting with it in Javascript. To do this, youll need some sort of connection with the arduino and your Javascript. In previous examples we talked about using USB, but in this case, I want to talk about doing it wirelessly.

## Particle (formerly Spark)



- $19
- Very **similar to Arduino**
- “Lives” **online**
- **Expose** hardware **functions / variables** to **REST endpoints**
- Publish **events** from the device

This is where a new piece of hardware comes in - the particle photon. The photon is a small microcontroller with a wifi chip. Programming for it is very similar to the arduino, but the big difference is the ecosystem for the particle. Particle makes it very easy to set variables and functions to be accessible from the web.



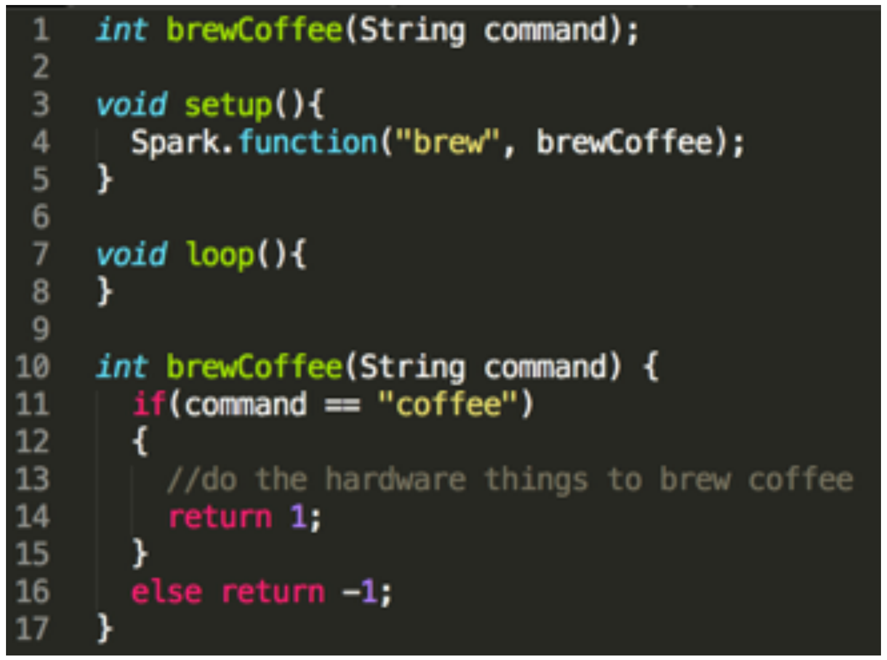
```
double tempC = 0;

void setup()
{
  Spark.variable("temp", &tempC, DOUBLE);
  pinMode(A0, INPUT);
}

void loop()
{
  var analogvalue = analogRead(A0);
  //Convert the reading into degree celcius
  tempC = (((analogvalue * 3.3)/4095) - 0.5) * 100;
  delay(200);
}
```

```
curl “https://api.particle.io/v1/devices/[deviceID]/temp?
access_token=[accessToken]”
```

For example, this is how you access a variable. Like the arduino, the particle has two functions required to run - a setup and a loop. In the setup function, you specify what to expose to the web. In this example, we’d have a temperature sensor hooked up, and then we could ping this device at any time for the temperature.



```
int brewCoffee(String command);

void setup(){
  Spark.function("brew", brewCoffee);
}

void loop(){
}

int brewCoffee(String command) {
  if(command == "coffee")
  {
    //do the hardware things to brew coffee
    return 1;
  }
  else return -1;
}
```

```
curl https://api.particle.io/v1/devices/[deviceID]/brew
     -d access_token=[accessToken]
     -d "args=coffee"
```

Secondly, and what I'll be demoing here, is how to expose a function. This will let you completely separate your hardware code from your server code to interact with it. So, you do have to figure out the hardware enough to do what you want, but you can trigger it with parameters from a webserver or whatever can send API calls out.

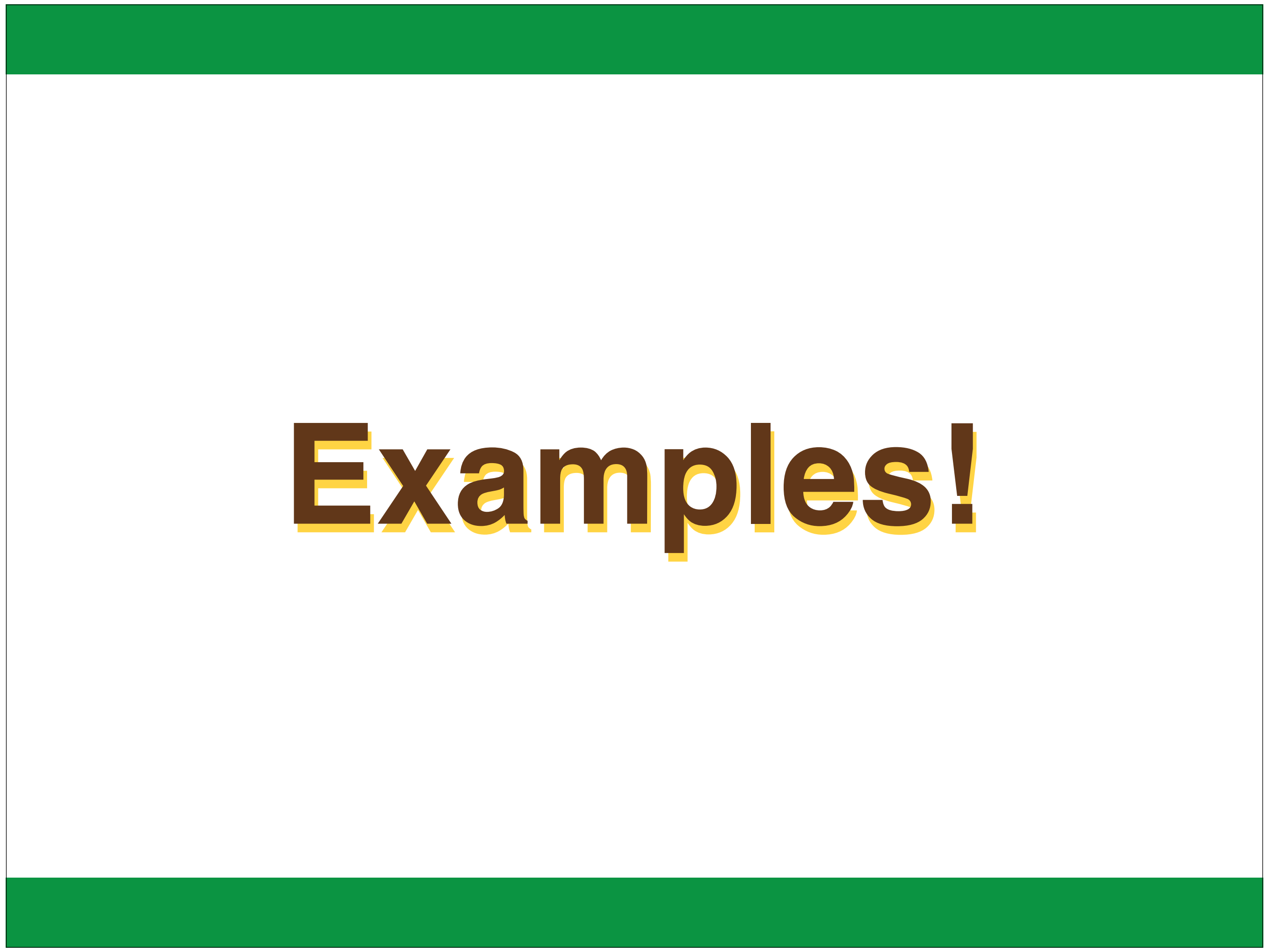
## Particle Process

**Program functionality** in hardware

**Expose** single **function** taking parameters

**Test function** by using curl, postman, etc

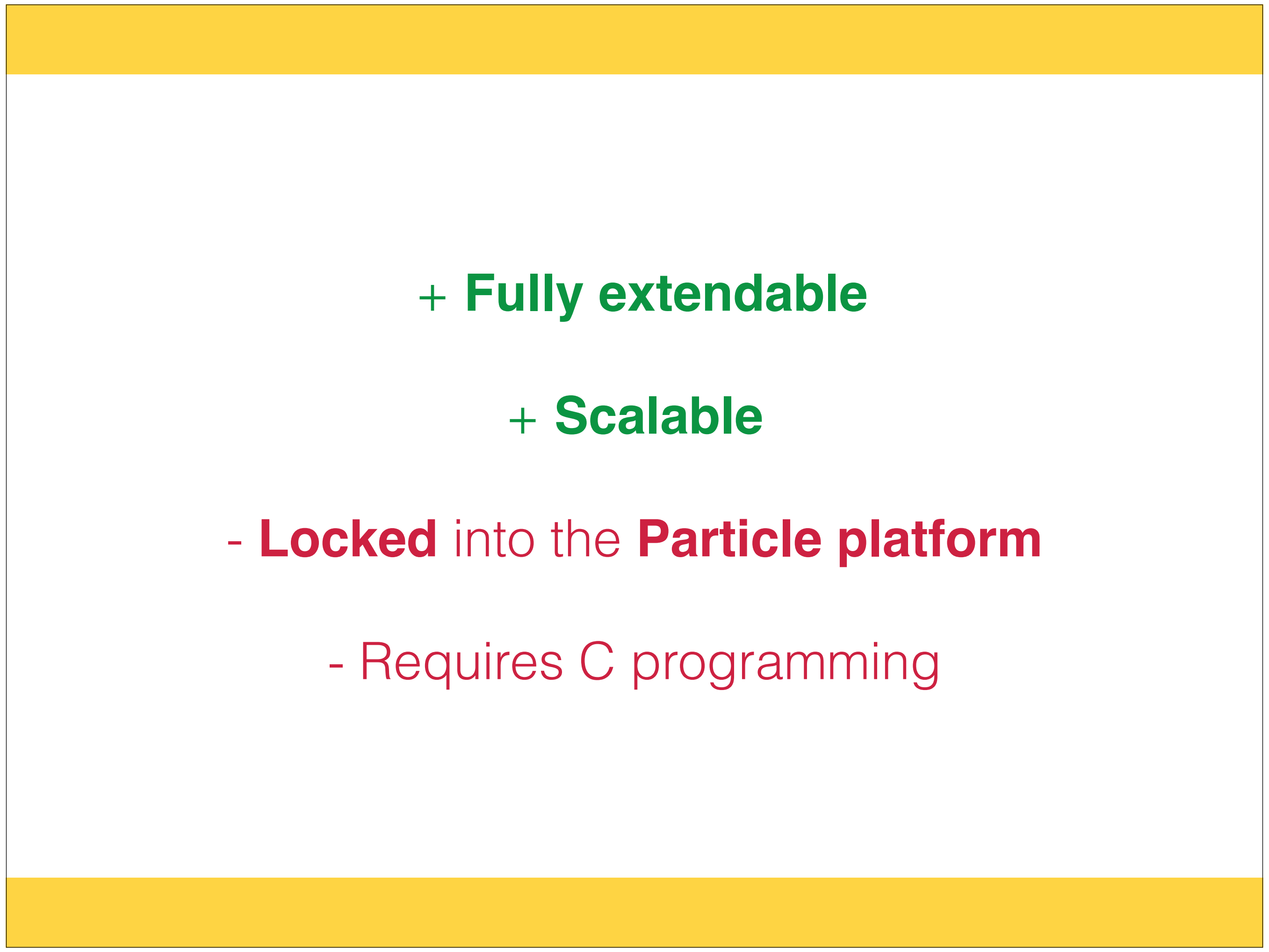
**Build app** calling function



## Examples!

The example is here, but please read the read me about this specific example. It’s non trivial to setup the power yourself. I’d recommend starting with blinking an LED on or off from a function and building up from there.

https://github.com/gelicia/midwestJSHW/tree/master/particle-example



+ **Fully extendable**

+ **Scalable**

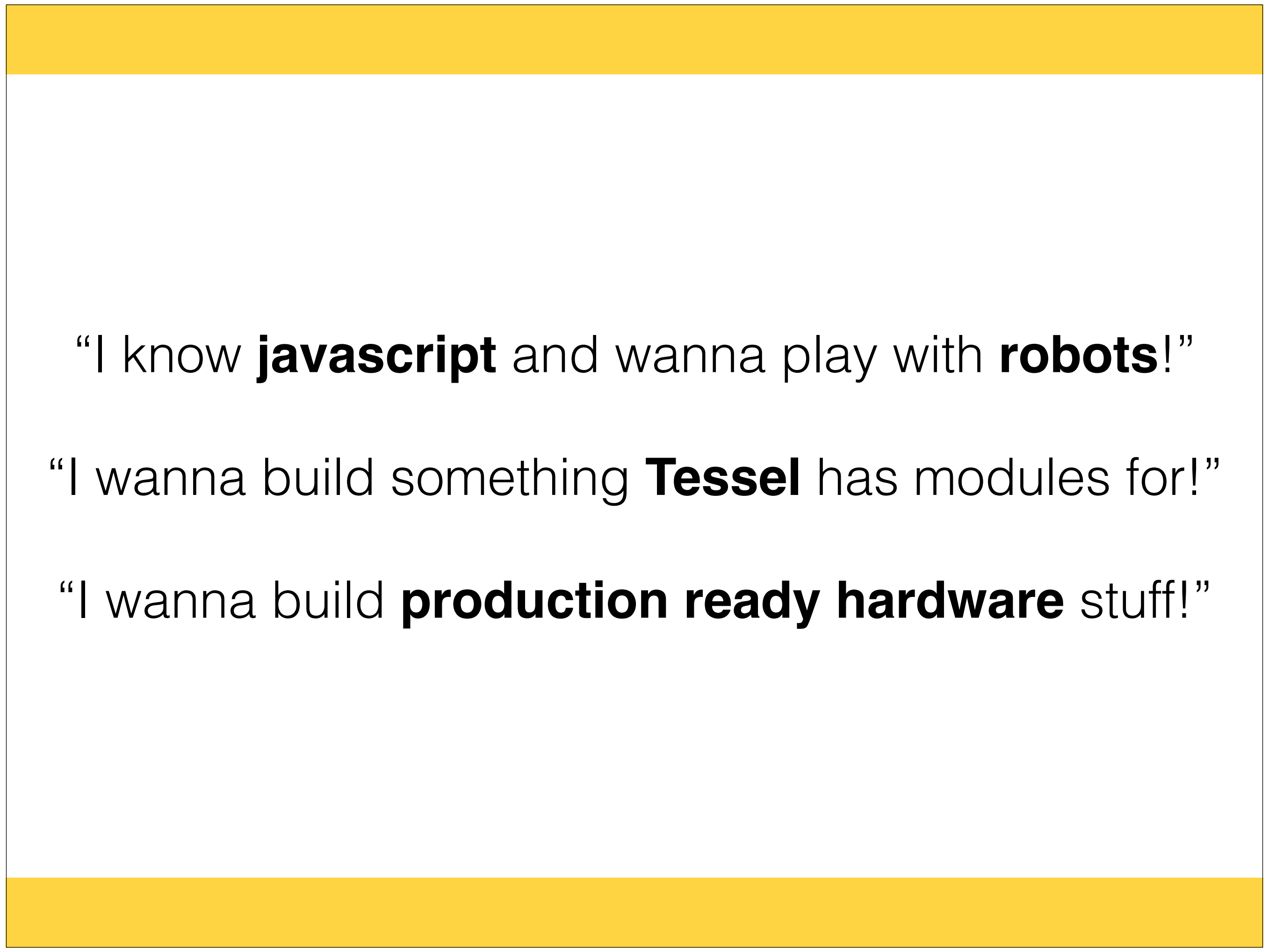
- **Locked** into the **Particle platform**

- Requires C programming

Like the other types, there are some benefits and drawbacks to this.
For one, it's really the best solution for someone who wants to build something to sell. Particle sells the wifi chips and has a full api solution for pairing up to their system. People have built stuff on the Particle system and sell it.
The downside is you are locked into the platform, and there are some drawbacks with this. For one that directly goes into the example, there's a 60 character limit out of the box with passing data to functions. I get around this in my app by having an admin message of BEGIN and END and chaining the rest calls to happen in the right order. There's a way to fix this in the firmware too, but I know javascript better than figuring that out.
There can also be some issues getting it to pair to the cloud. Improvements get made, but you are on Particle's schedule for it
Finally, you can't program everything in javascript, you'll have to suffer through the hardware code to build that part.

## Bottom Line

I wanted to close with a summary of these different ways to program hardware with javascript.


"I know **javascript** and wanna play with **robots**!"

"I wanna build something **Tessel** has modules for!"

"I wanna build **production ready hardware** stuff!"

For one, if you're looking for pure ease of use and the modules exist for what you want to do, the Tessel is a really good route to take.
If you want something with just one more but easy step, johnny five is great for using hardware you already own and a lot more flexible.
If making an IOT-thing appeals to you, and maybe you might even want to sell them someway and aren't scared of a lower level language, the Particle cloud is an excellent option.

https://github.com/gelicia/midwestJSHW

http://files.gelicia.com/midwestJSHW

# Thank you!

@gelicia kdurivage@gelicia.com