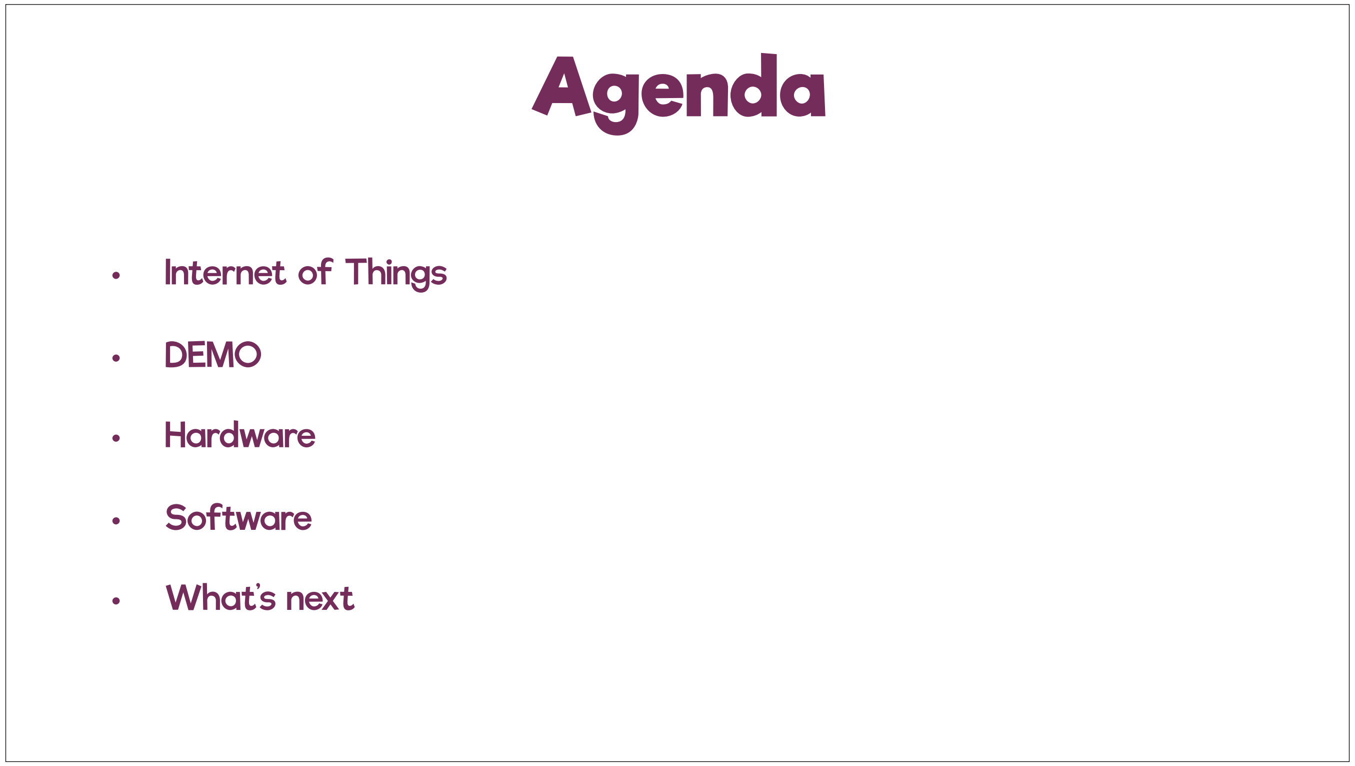
My Doorbell Runs Swift

iOSDevCampDC 2017

Rob Huebner

@huebnerob

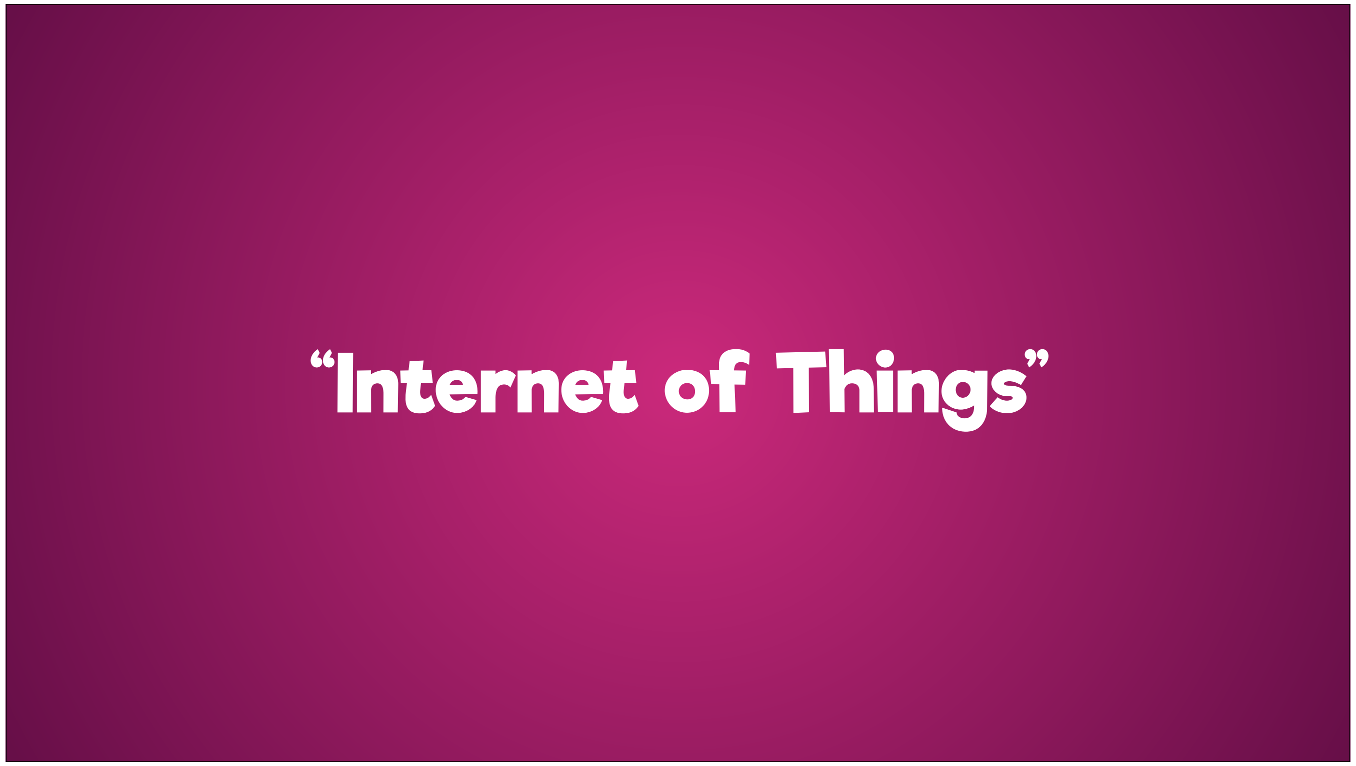
- Hey y’all! My name is Rob! I’m an iOS engineer at Blue Apron.
- Really excited to be here at iOSDevCampDC 2017!
- Now I know this is *iOS* Dev Camp, but today I’m *actually* going to get outside that box, pretty far outside actually
- I’m going to talk about a hobby project I built on a whim
- It touches on a broad range of topics, and we’re going to go a bit deeper into each one.
- By the end, your “full stack” is going to be just a little bit deeper



Agenda

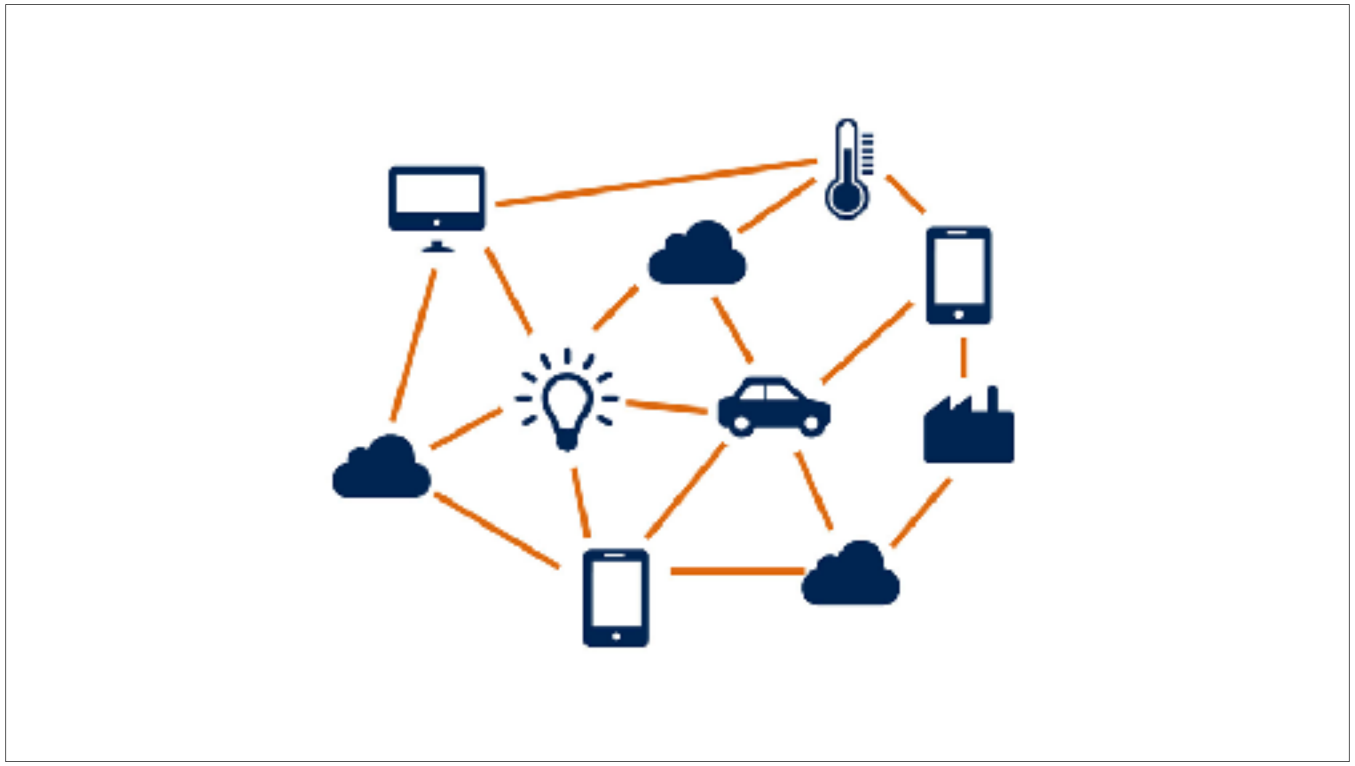
- Internet of Things
- DEMO
- Hardware
- Software
- What's next

- So here’s a short summary of what we’re going to talk about today
- We have a lot to cover (but I’ll try to keep things light)
- First, we’ll ease in with a discussion of what “Internet of Things” means to us
- Then, I’ll jump into a DEMO
- From there, I’ll go over the hardware and software concepts that we’ll need to start executing our project
- And I’ll close with some thoughts about potential futures, what you can check out next



“Internet of Things”

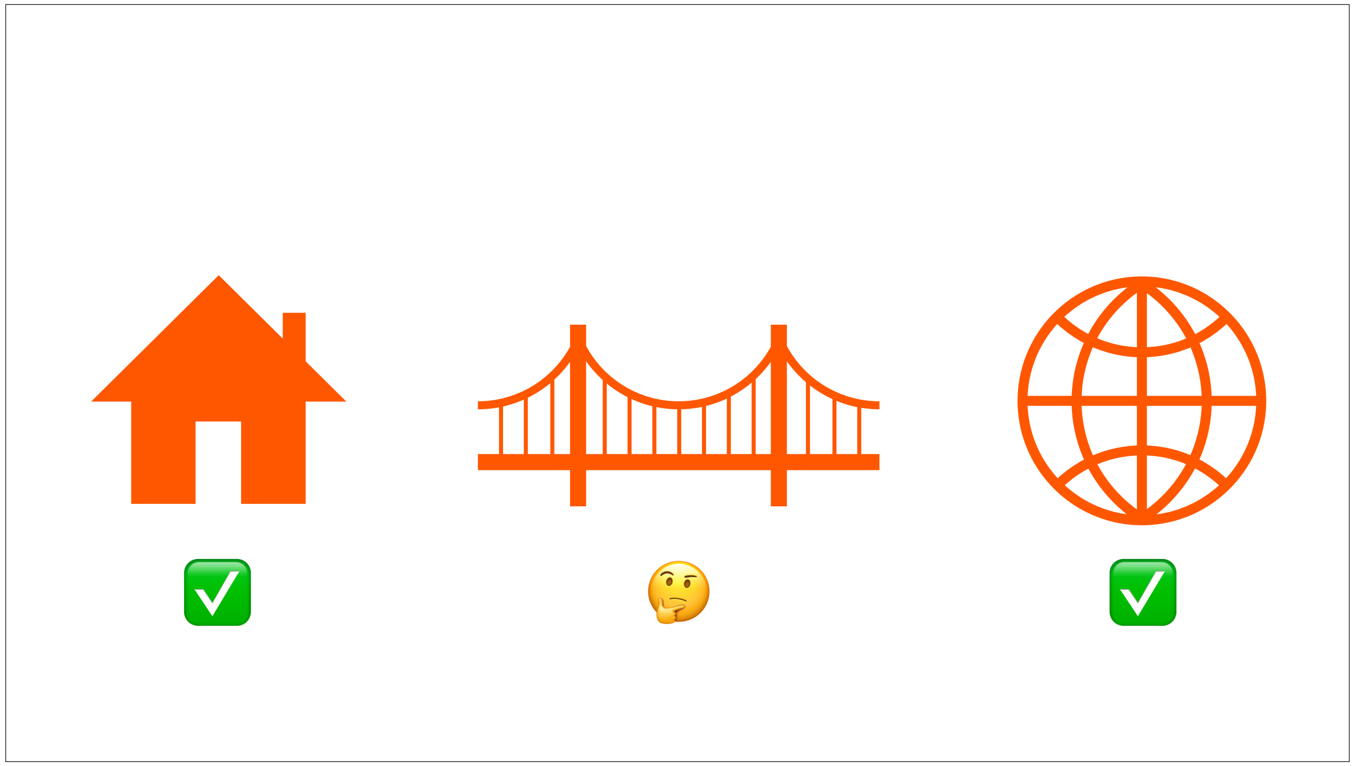
- Before we talk specifics, I wanted to bring up some terminology
- This term gets thrown around a lot, “Internet of Things”
- What does it *actually* mean? Why should we actually care?



- When I google image searched “Internet of Things”, this was one of the first results
- But what are these other things? Here’s the infamous cloud, a Philips hue light bulb, that also sounds right, A factory?
- What is the goal here? It looks like the “internet of things” is just a bunch of stuff with WiFi
- The internet of things ***isn’t just*** a bunch of stuff with WiFi

Example: Philips Hue Lights

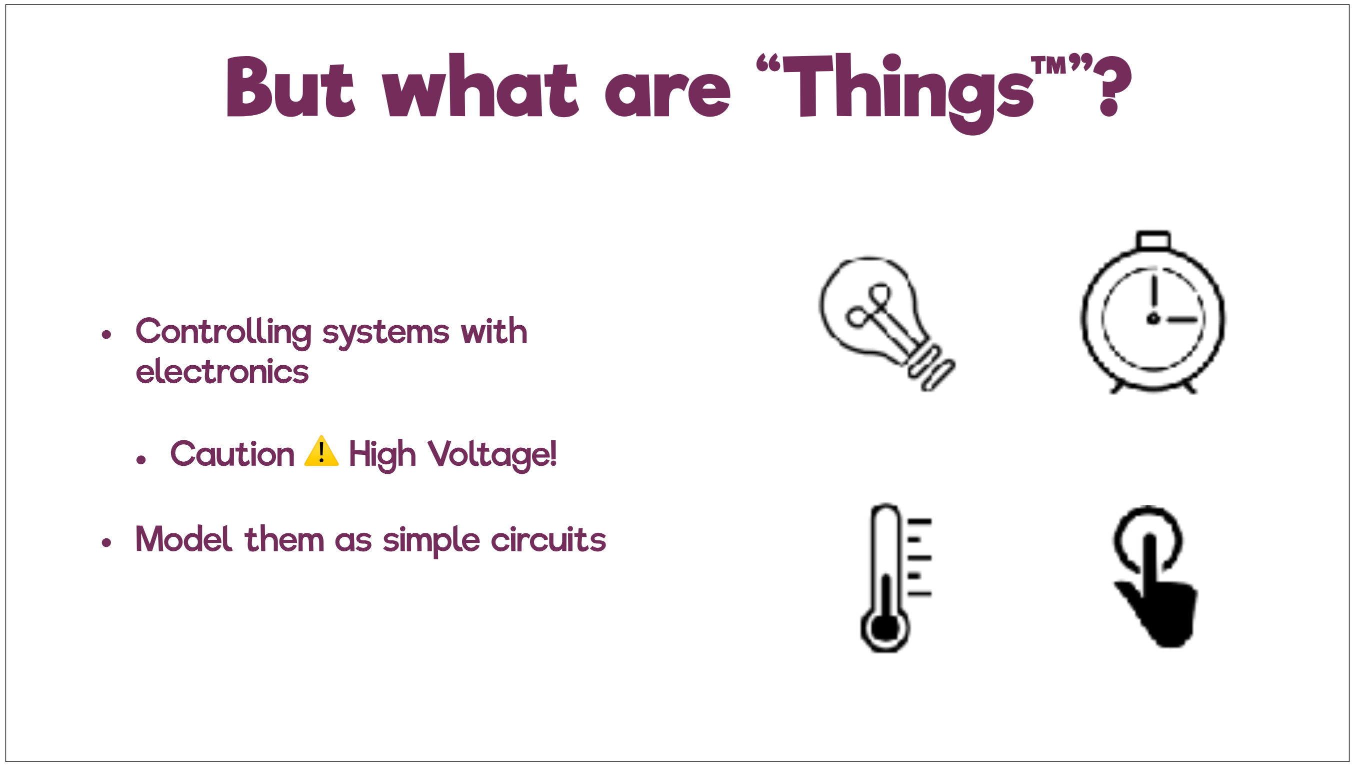
- So what does IoT look like in practice? Let’s get a bit more concrete
- Here’s an example: Philips Hue Lights
    - Multicolored LED lights that replace conventional bulbs
    - You need to plug in this weird box thing
    - Then you can control them with an app
- So in the high level:
    - We have the physical thing
    - Some kind of a “bridge”
    - Then the internet/app world we all know and love



- This is a pattern we see whenever we talk about the “Internet of Things”
- In between the “Internet” and the “Things”, there’s some kind of a ‘bridge’.
- But therein lies all the magic of IoT, it’s really all about the bridge
- We have the internet already, we understand what Real Things™ are, but what is this “**bridge**”? What does it mean to us, in a general sense?

Build Our Own Bridge

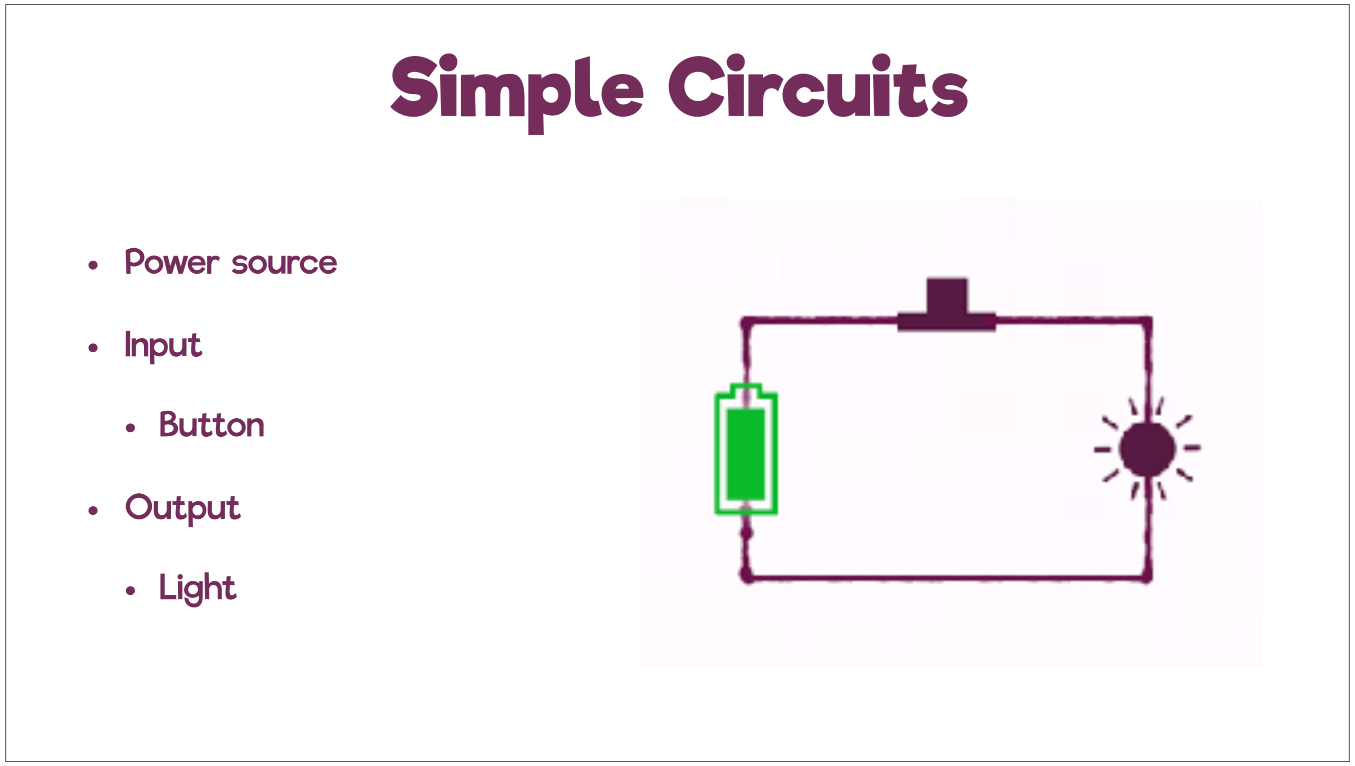
- And that's our ultimate goal, we're going to *Build Our Own Bridge*
    - Honestly, this should be the *real* title of the talk
- But we're at *iOS* Dev Camp, why do we care?
    - Well, two reasons:
        - 1. People interact with the entire world mostly through their phones these days.
            - The IoT is no exception.
            - The bridges you build are between phones and the physical world, not desktop computers
            - This is actually great for us though, because there's *way more awesome* to be had on mobile
            - We're deepening the relationship with the user
        - 2. Also, as iOS Developers, there's something here for us too
            - We can do this all in Swift
            - At the end of the day, code is code, but it's nice to be able to use a language you know in a new and unexpected way



But what are “Things™”?

- Controlling systems with electronics
  - Caution ⚠ High Voltage!
- Model them as simple circuits

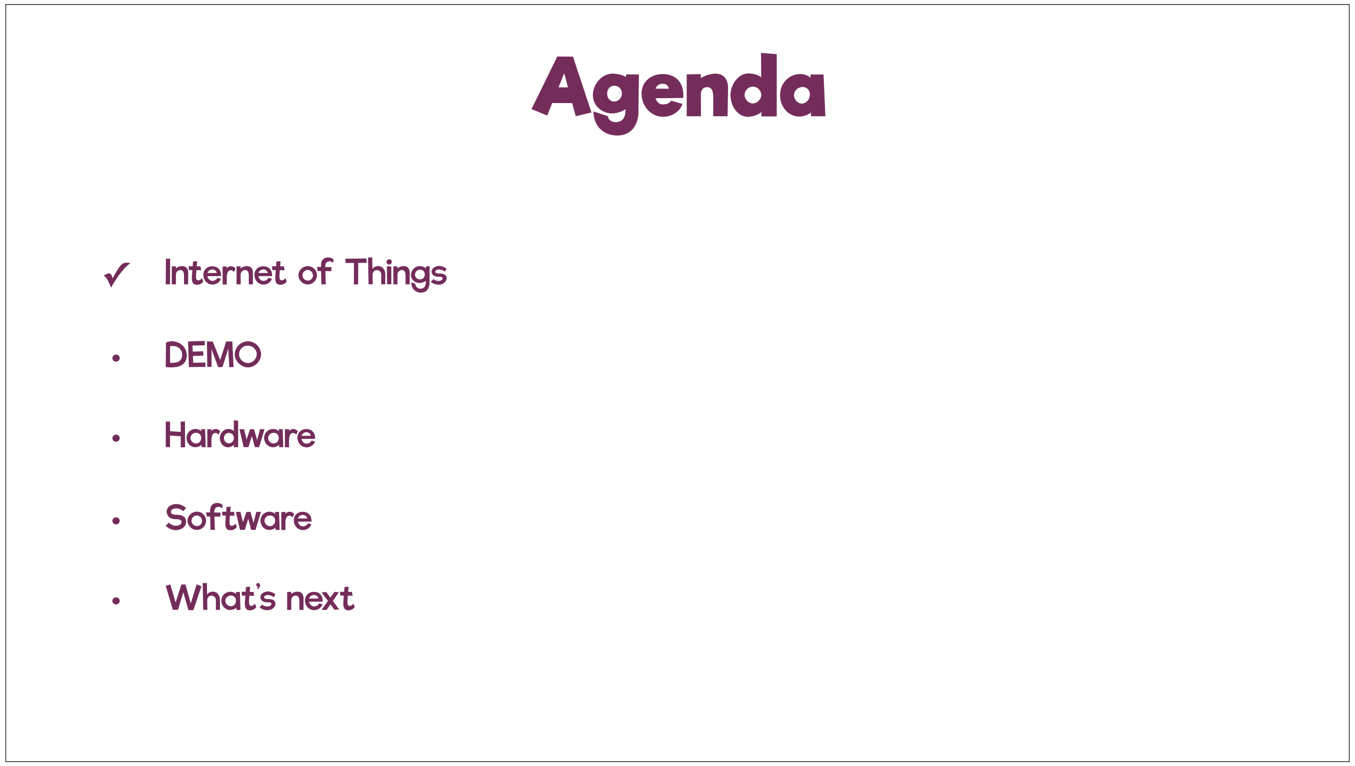
- You might still have questions about the left hand of that equation though
- What are “things”?
- Well, really, it’s all about interfacing with electronics at some level
- So many modern everyday objects are controlled via electronics, we just need to ‘get on their level’
- You can build all kinds of advanced circuits to control appliances and use sensors, this universe is *enormous*
- Ultimately, building circuits is out of scope for this presentation
- So, for today, we can just model things as a simple circuit



Simple Circuits

- Power source
- Input
  - Button
- Output
  - Light

- Here it is, a simple circuit, the most distilled form of a “Thing”
- We have a power source, a button, and a light
- When you push the button, the light illuminates, easy peasy
- If you think about it, a lot of stuff isn’t really much more complicated than this, though
  - Turning something on and off goes very far
  - A button is kind of like the simplest ‘sensor’
- We’ll use this simple metaphor for inputs and outputs as we continue to build our bridge



Agenda

- ✓ Internet of Things
- DEMO
- Hardware
- Software
- What's next

- So that's the scoop on "Internet of Things"

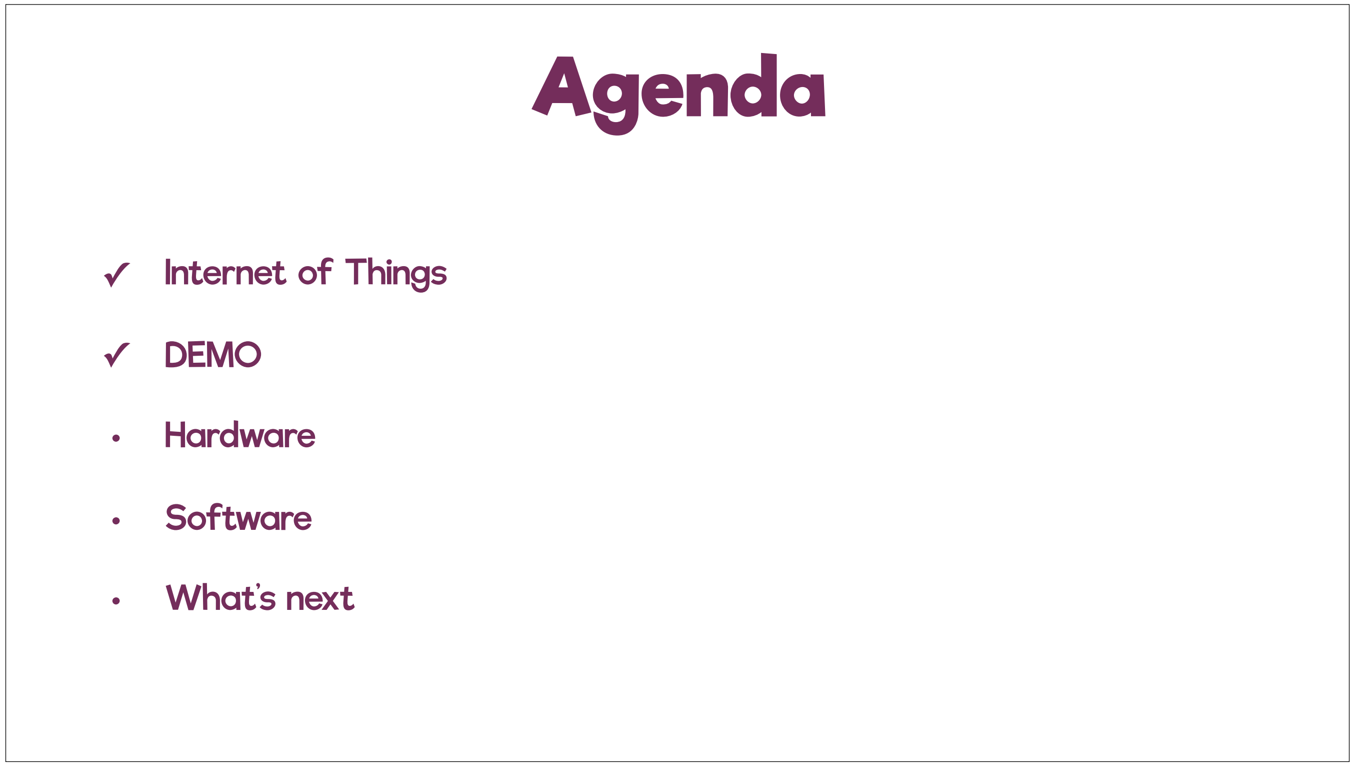
DEMO

- Now the fun part
- Like a cooking show, I’ll show you the finished pot roast, then I’ll show you all the hard cooking you need to do
- Without further ado, this is my doorbell running Swift…

- Demo:
    - here comes the guest
    - they press doorbell
    - Doorbell in apt ringing!
    - magic behind the curtain (pi detecting bell)
    - notification received on phone
    - homeowner presses door unlock quick action
    - pi gets door unlock api call
    - door buzzing open
    - guest walks in

DEMO

- So that’s it, my doorbell running swift.
    - I’ll get into what that even means in a second
- But I think the key thing to take away from this demo is the mundanity of the interaction and the simplicity of the solution
- This is a sequence of events that a lot of us go through often, and it’s improved by *juust* the right amount of technology.
- There are a lot of these sorts of interactions all around us.
- I’m going to be talking about my doorbell, because that’s what I did, but I want you to keep your eyes open
- Figure out how all this applies to something **YOU** care about



Agenda

✓ Internet of Things

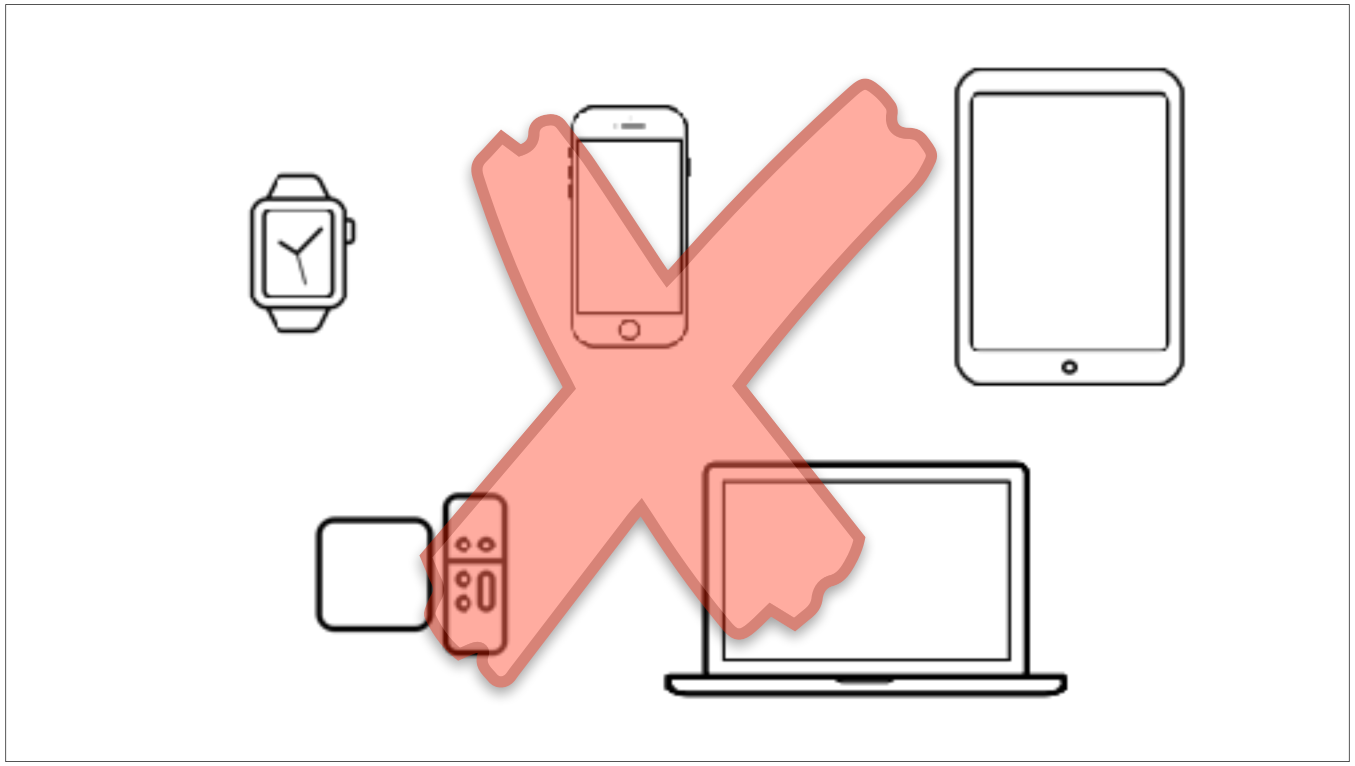
✓ DEMO

- Hardware
- Software
- What's next

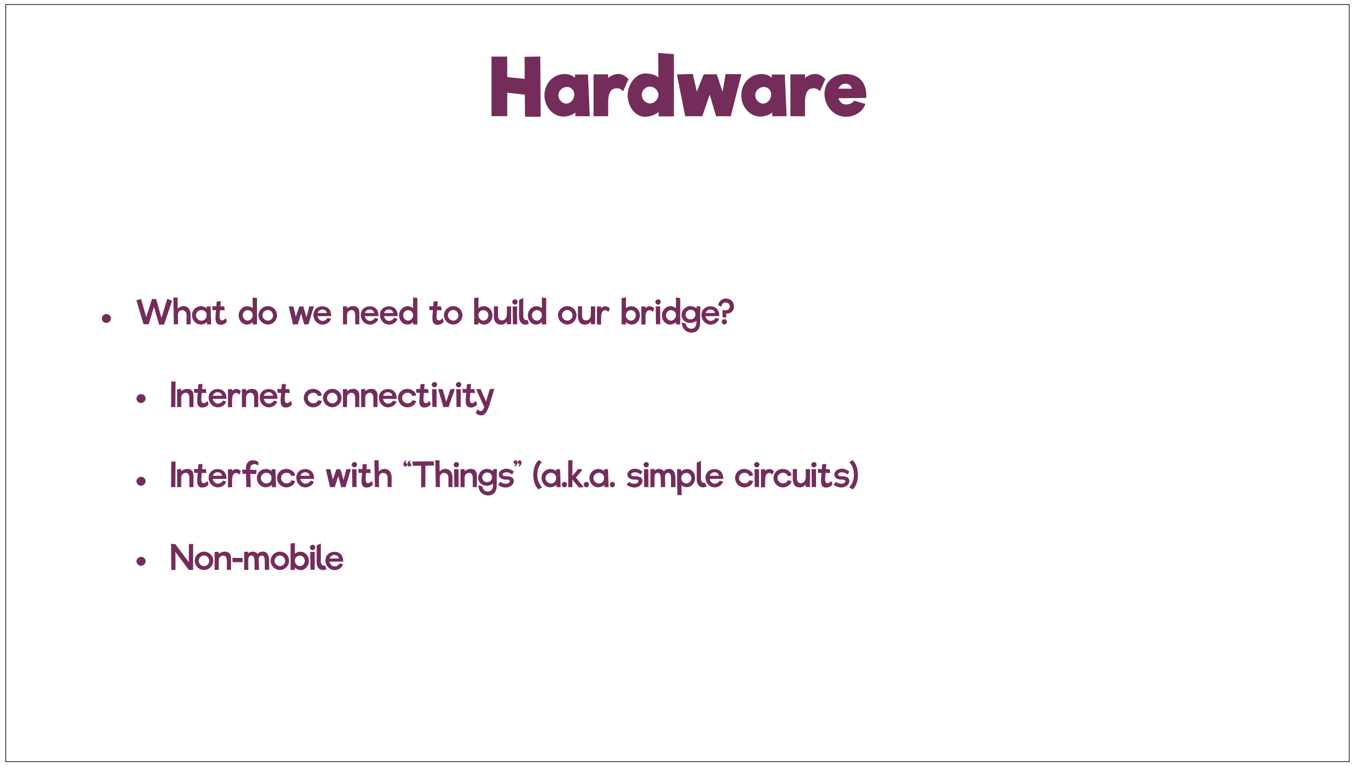
- Well that was fun, but *how do we build it?*

Hardware

- Let’s start with the hardware
- Kind of uncommon to be worried about hardware in the context of Apple development



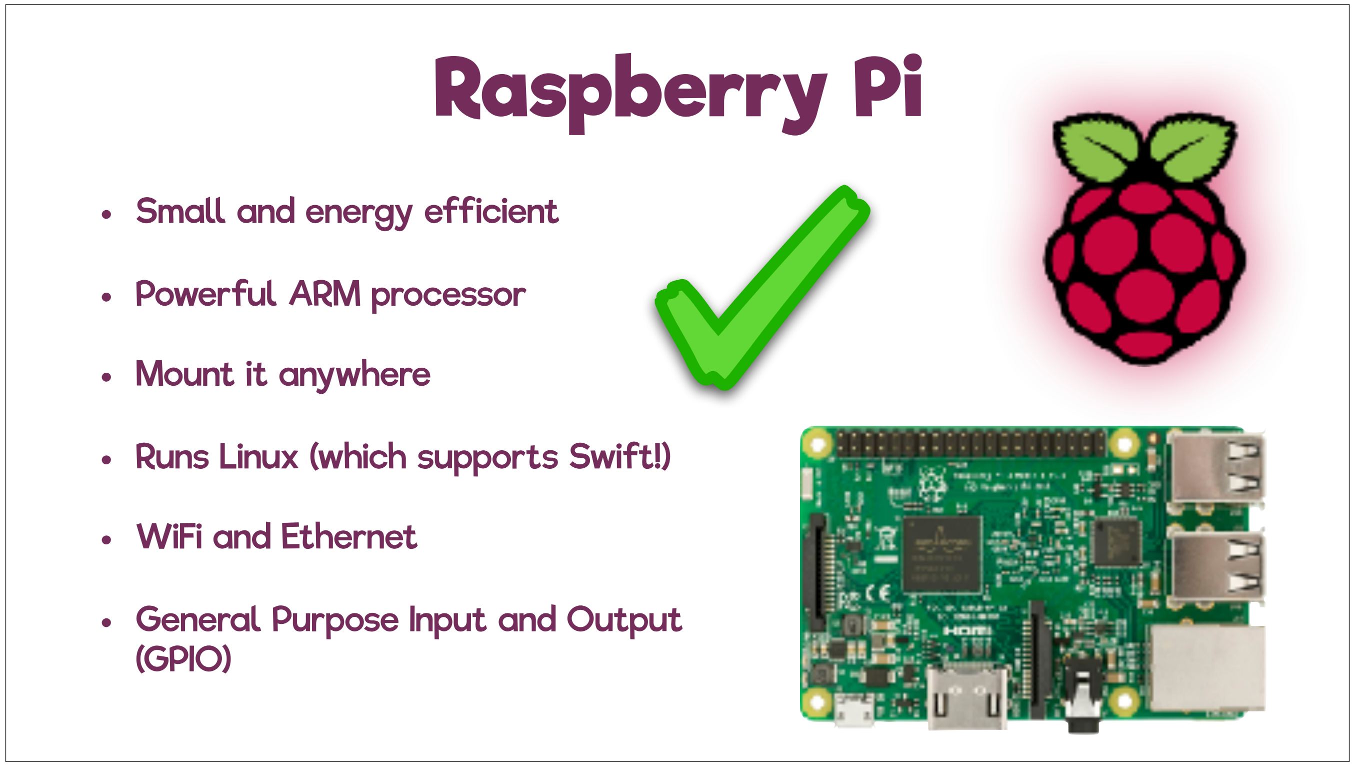
- We’ve been taught that this is a closed system, Apple has already given us all the hardware we’ll ever know
- Certainly there’s no lack of diversity, there’s plenty of different classes of Apple hardware to develop for
    - iPhone, iPad, watch, mac, TV
- The issue is that none of these devices quite capture ***everything*** we need to build our bridge for the internet of things



Hardware

- What do we need to build our bridge?
  - Internet connectivity
  - Interface with "Things" (a.k.a. simple circuits)
  - Non-mobile

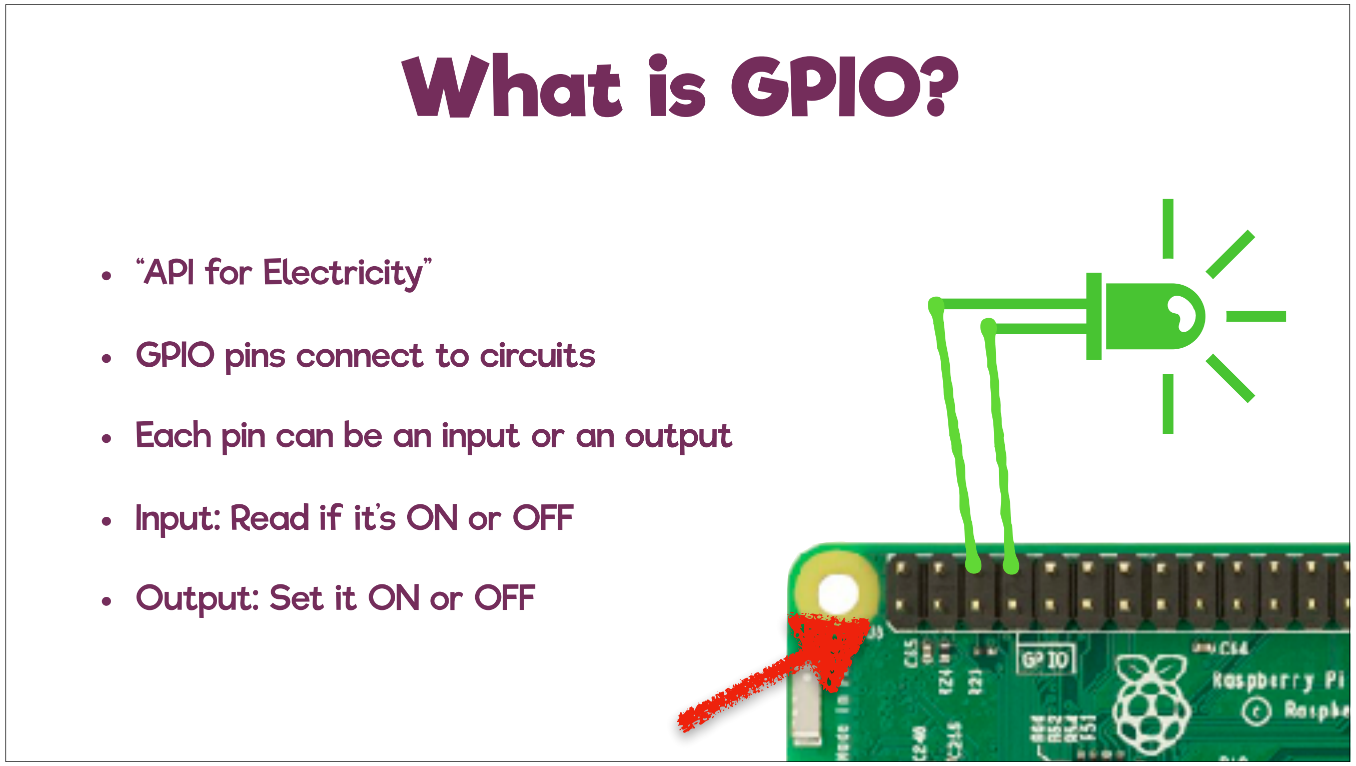
- Why aren't there really any Apple devices that are good bridges?
- To understand why, ask the basic question, What do we *need* to build our bridge?
  - We need to talk to the internet
  - We need to talk to real things
  - And, at the end of the day, our bridge is basically just going to sit there,
    - so it would be great if it were optimized for non-mobile use
- Well, iOS devices are too mobile, we're not going to hook them up to our doorbell
- Macs and Apple TVs come closer, but they still don't have the right interface to interact with "Things"
- All of these devices are lacking a way to interact with simple electronic circuits at a low level
- no USB ports on my doorbell, or wifi, that's the thing I'm trying to build!
- What do we use?
- It turns out, we actually need to leave the Apple store to find this hardware!



Raspberry Pi

- Small and energy efficient
- Powerful ARM processor
- Mount it anywhere
- Runs Linux (which supports Swift!)
- WiFi and Ethernet
- General Purpose Input and Output (GPIO)

- Turns out, there’s a device that checks all of our boxes, and it only costs 35 dollars
- It’s called a Raspberry Pi!
- It’s a tiny computer with an efficient ARM processor, like your iPhone
- It’s so small that you can install it virtually anywhere you want, which is naturally amazing for connecting to “Things”
- It runs Linux, which gives you access to a wealth of software, including Swift
- It has both WiFi and Ethernet, so internet connectivity is not a problem.
- But the feature that *really* sets the Raspberry Pi apart for us, is this thing called **general purpose input and output**, or “**GPIO**”



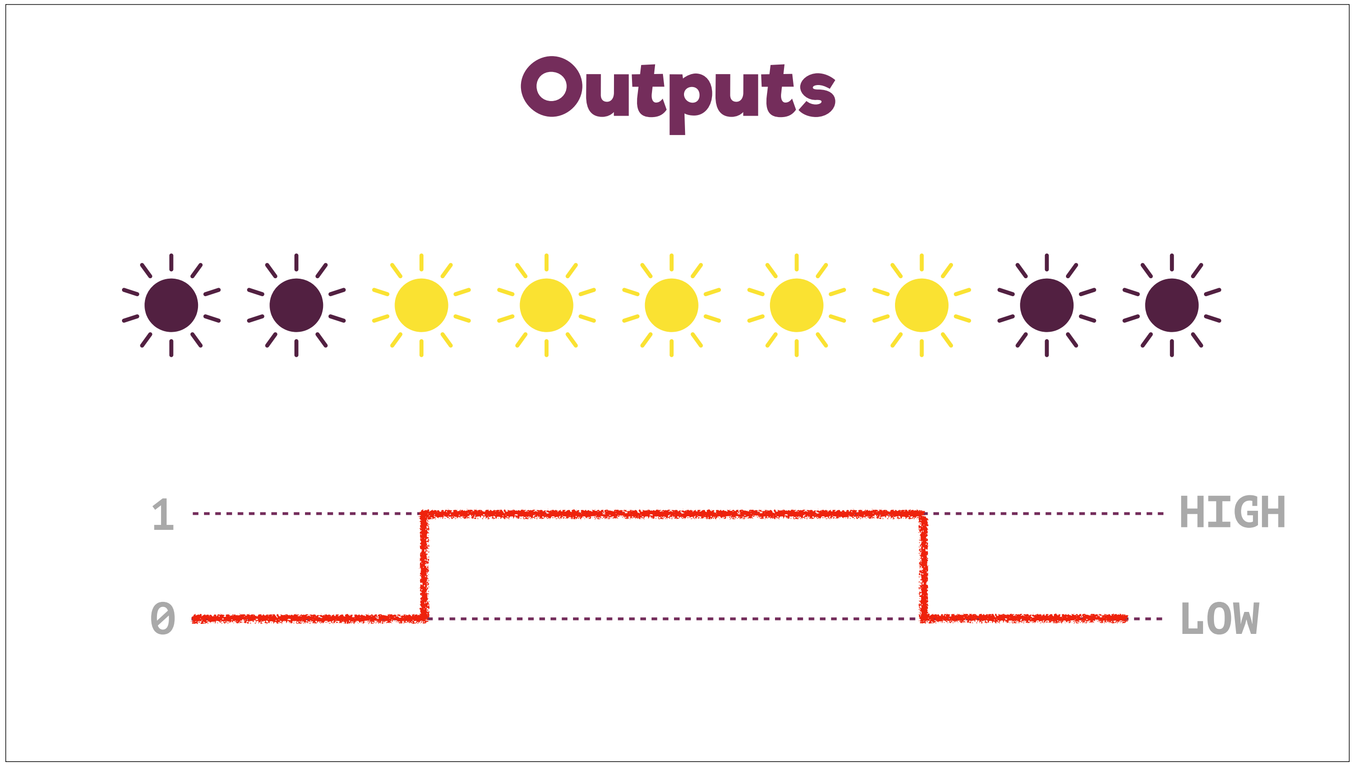
What is GPIO?

- “API for Electricity”
- GPIO pins connect to circuits
- Each pin can be an input or an output
- Input: Read if it’s ON or OFF
- Output: Set it ON or OFF

- So what is **GPIO**?
- It’s this port that looks like a series of pins on the edge of your Raspberry Pi
- I like to call it an “API for Electricity”
- GPIO allows you to interact with circuits at the lowest level possible.
- That means that you can do things like turn a light on and off, or read the value from a sensor
- With relatively small effort, you can design circuits that interact with your everyday household objects

- “At the simplest level, you can think of them as switches that you can turn on or off (input) or that the Pi can turn on or off (output).”

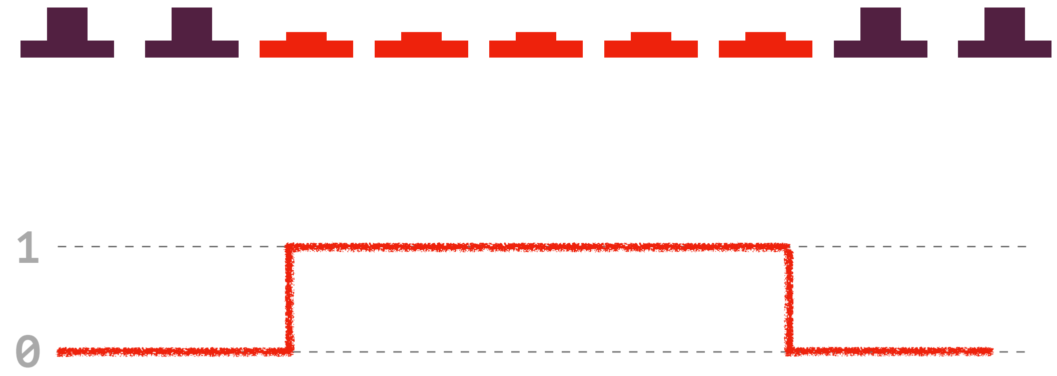
Resource: https://www.raspberrypi.org/documentation/usage/gpio/



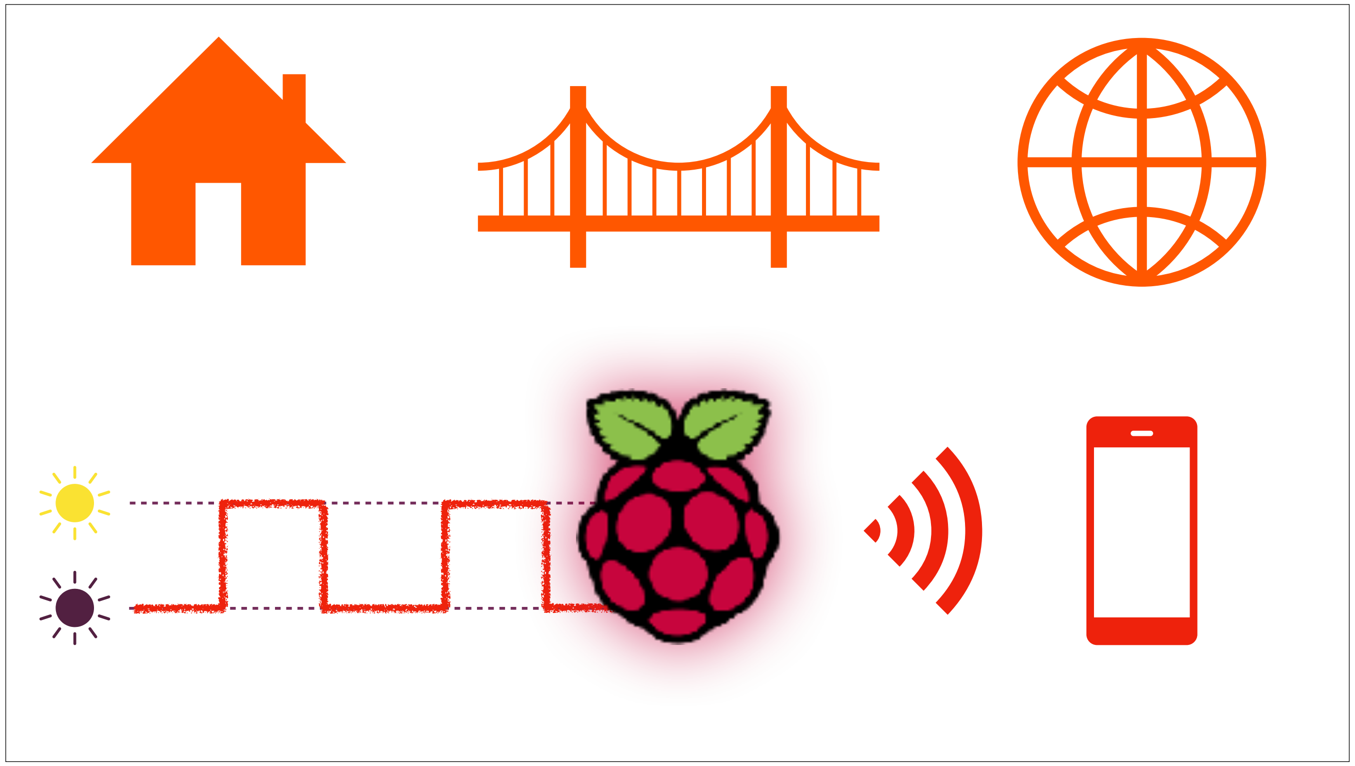
Outputs

- To illustrate
- We can set up a GPIO pin as an output
- As our program executes, we can set the value of the pin to either 1 or 0
- This turns the light on and off, respectively
- Sidenote about terminology, in the electrical world this can also be referred to as HIGH and LOW
  - It's the same thing though, if you encounter it don't be alarmed

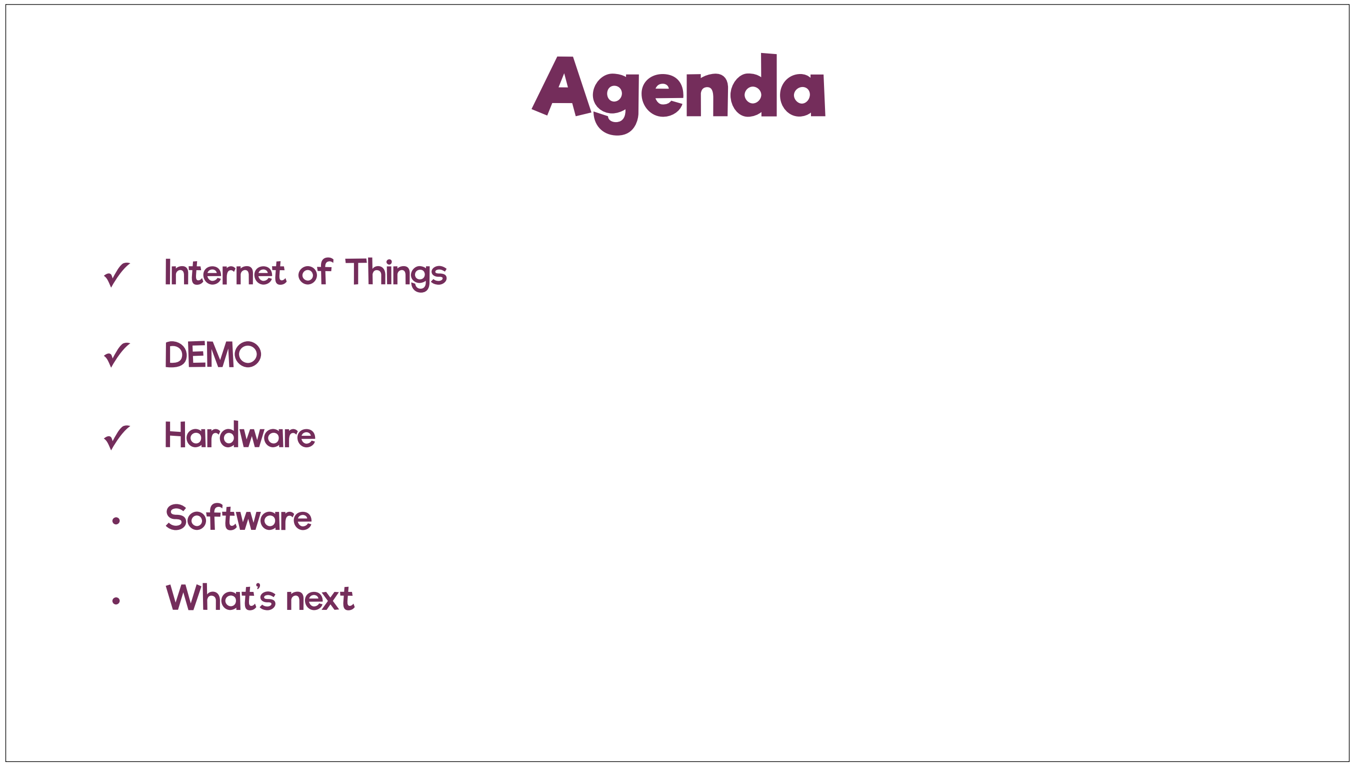
Inputs



- Alternatively, we can also set up the pin as an input
- In this case, we can read the value of the pin as the program executes
- Say we had a circuit with a button connected, kind of like the one example we had before
- Our pin would have a value of one when the button is pressed, and zero when its released



- All I’d want to stress here is how uniquely situated the Raspberry Pi is
- It has Swift, WiFi, and GPIO!
- Raspberry Pi is the *perfect* bridge
- It sits right in the middle of these two worlds
- It has all the interfaces we need to connect both of them



Agenda

- ✓ Internet of Things
- ✓ DEMO
- ✓ Hardware
- • Software
- • What's next

- That's what our bridge hardware will look like

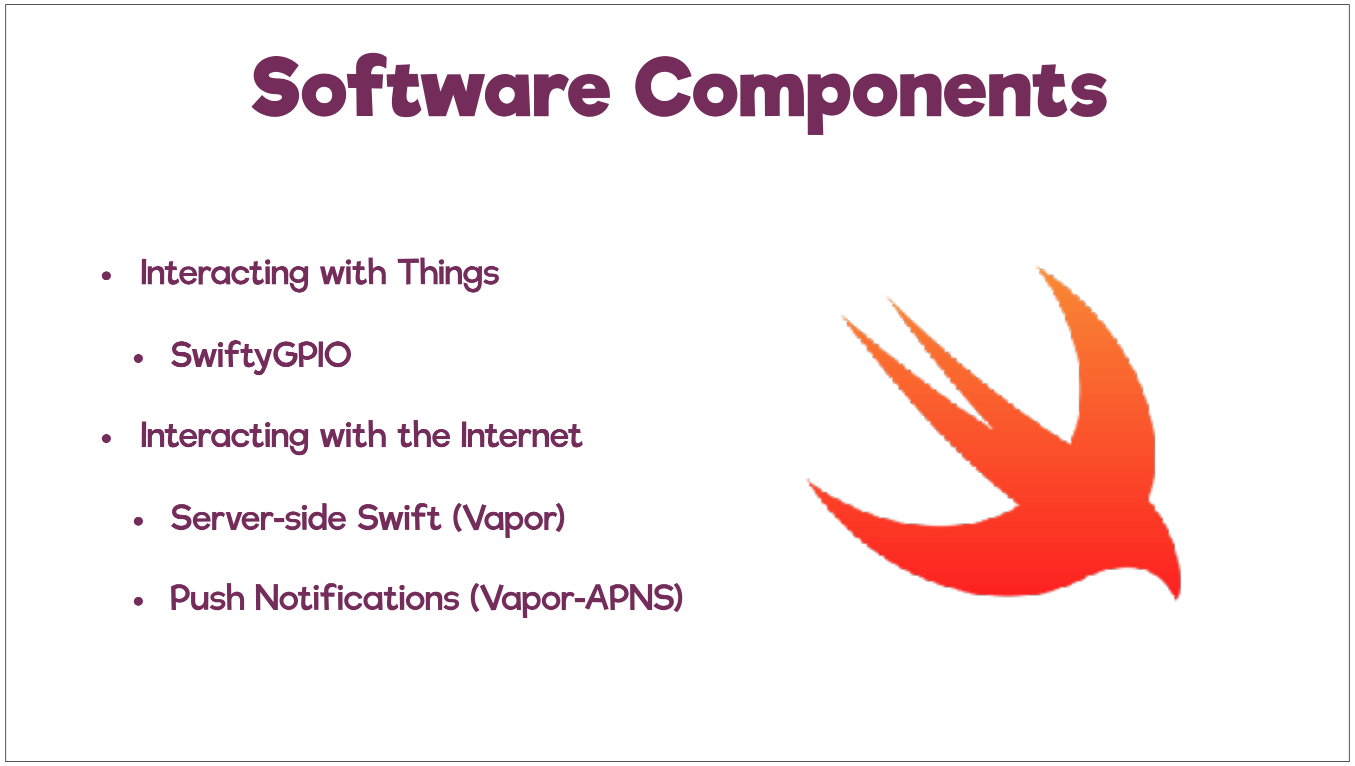
Software

- Let’s move on to the software

Setting up your Pi

- Ubuntu Linux
- Installing the Swift Toolchain
- Installing dependencies
- Network configuration
- Helper guide
  - goo.gl/P5JKcy

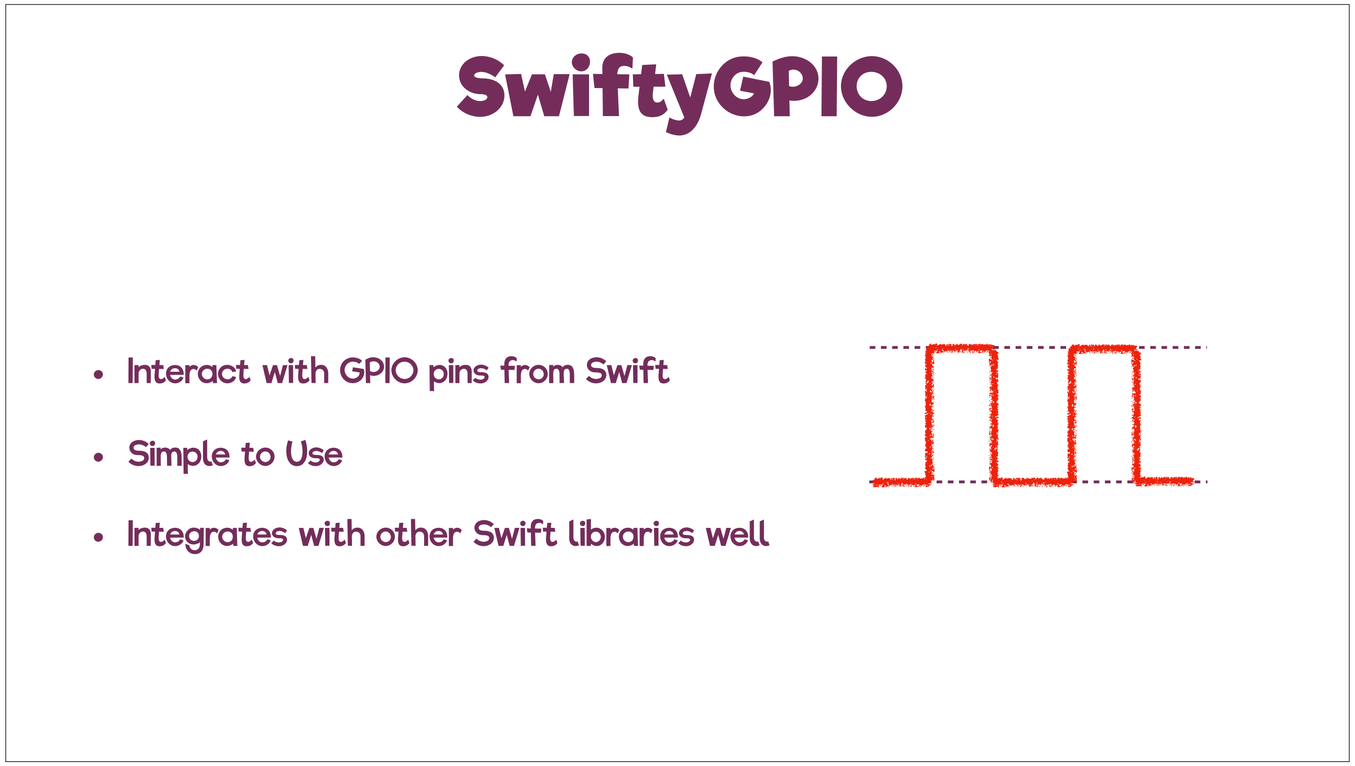
- We have our Pi, but there's still a bit of setup we need to do
- Sadly, this isn't key as turnkey of a solution as iOS
- We'll install a suitable version of Linux
- We'll install the open-source Swift toolchain
  - Major props to the Swift team for making this Linux-compatible
- There are a few dependencies here and there that our software stack will require
- To make this all very simple, I've written up a short guide on my website that will get you up and running quickly
- Check it out when you get your Pi, or feel free to reach out to me on twitter if you need help! (@huebnerob)



Software Components

- Interacting with Things
  - SwiftyGPIO
- Interacting with the Internet
  - Server-side Swift (Vapor)
  - Push Notifications (Vapor-APNS)

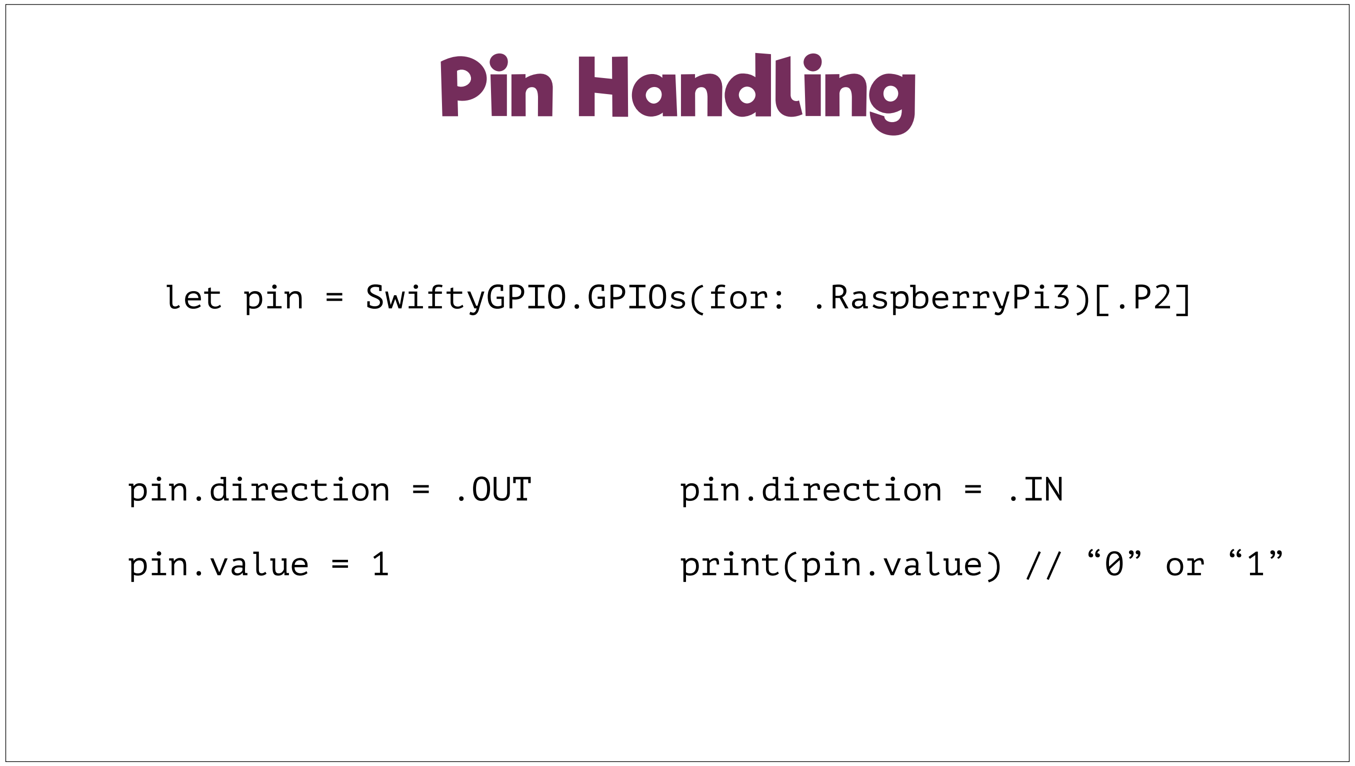
- Once we’re all configured, we start building our Bridge’s software stack
- I’ve broken it down into two main categories of software to deal with
- The first category I’ll call “Interacting with Things”
  - This is where we write the code that interacts with our GPIO pins
- The second, I’ll call “Interacting with the Internet”
  - This is where we stand up a web server and send push notifications
- Putting both these worlds together is what enables such powerful interactions



SwiftyGPIO

- Interact with GPIO pins from Swift
- Simple to Use
- Integrates with other Swift libraries well

- Returning to a topic near and dear to my heart: GPIO
- We know now that GPIO is a way for devices to interact with simple circuits
- SwiftyGPIO is a great Swift library to interact with your GPIO pins from your Swift code
- Of course, because it’s all in Swift, we can program our GPIO pins from in concert with other Swift libraries
- Let’s see what that code actually looks like



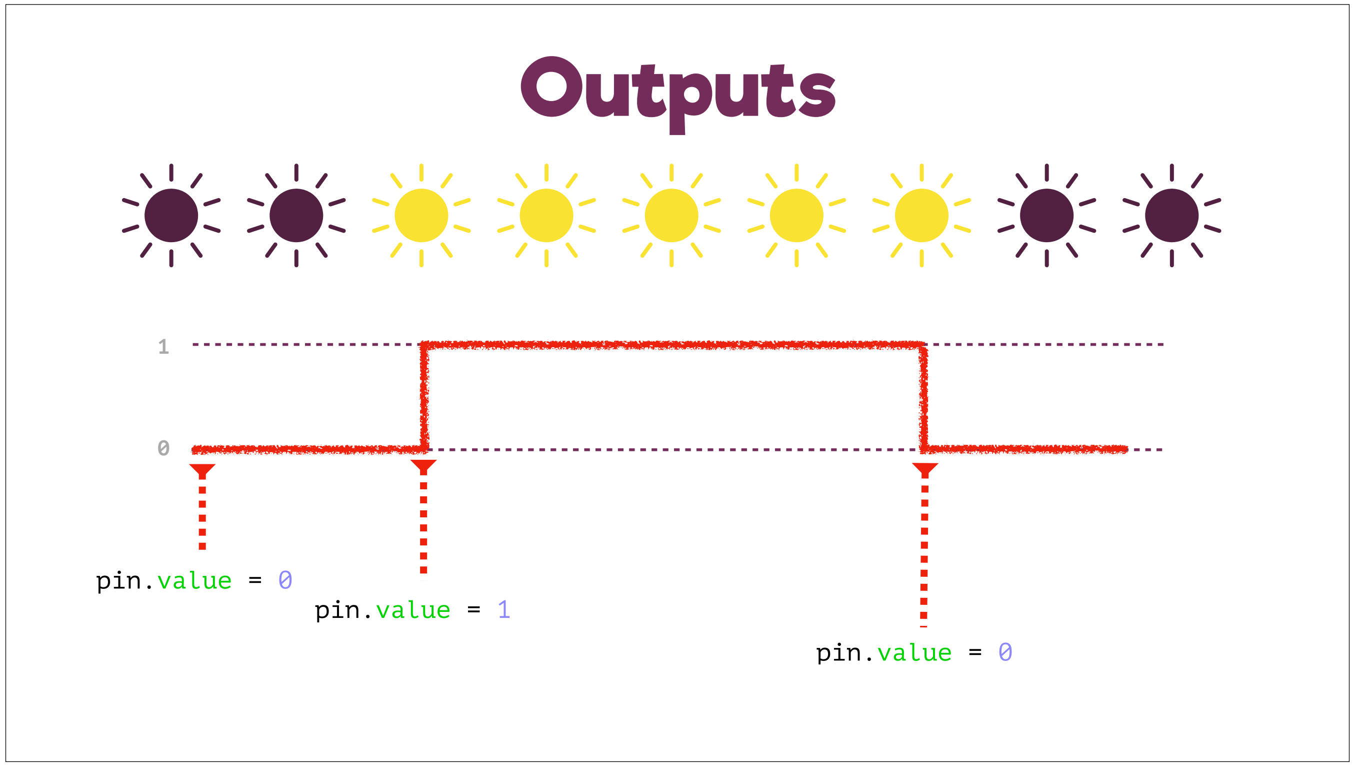
Pin Handling

```
  let pin = SwiftyGPIO.GPIOs(for: .RaspberryPi3)[.P2]


pin.direction = .OUT          pin.direction = .IN

pin.value = 1                 print(pin.value) // “0” or “1”
```

- First, you initialize a GPIO object using the type of device your running on and the physical pin number
- Second, you set the direction for the pin, which is whether it’s an INput or an OUTput
- If it’s an OUTput, you can set the value of the pin
- If it’s an INput, you can read the value of the pin



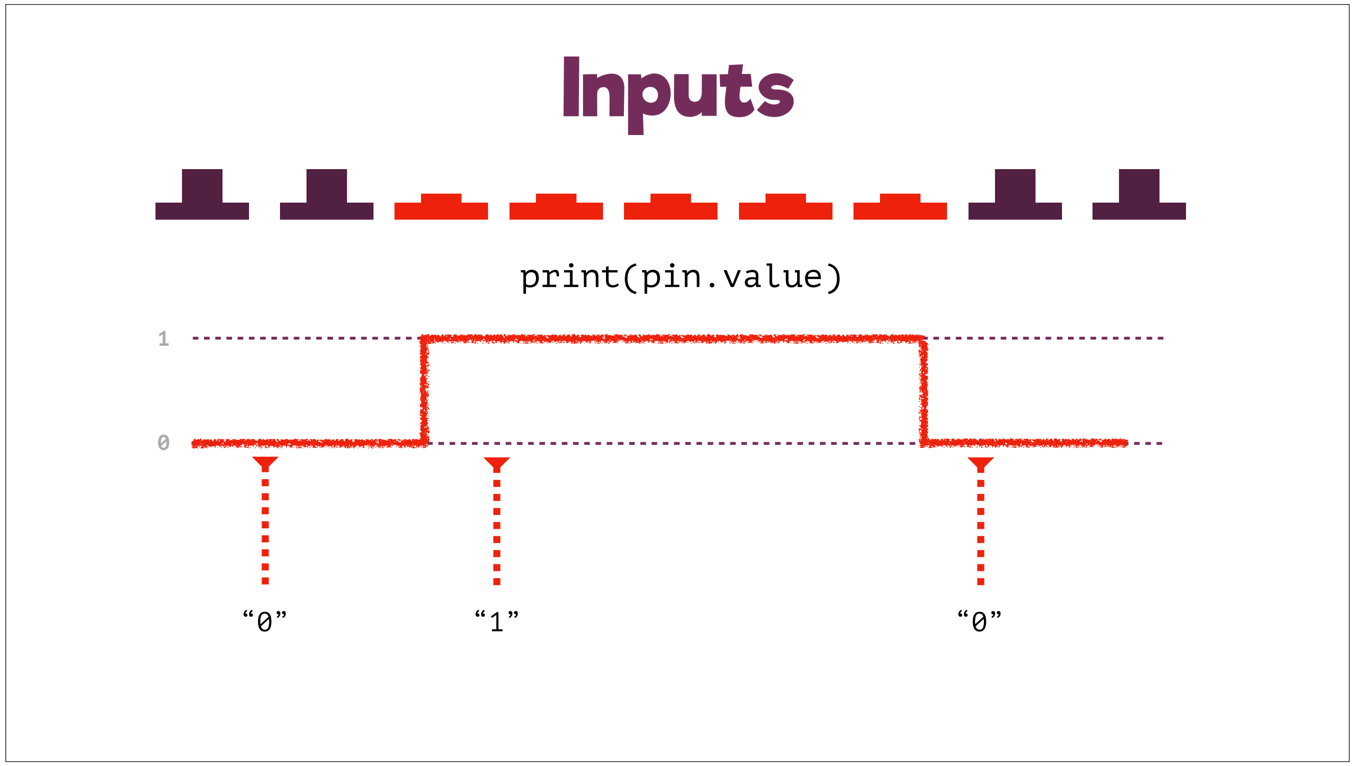
Outputs

```
pin.value = 0
```

```
pin.value = 1
```

```
pin.value = 0
```

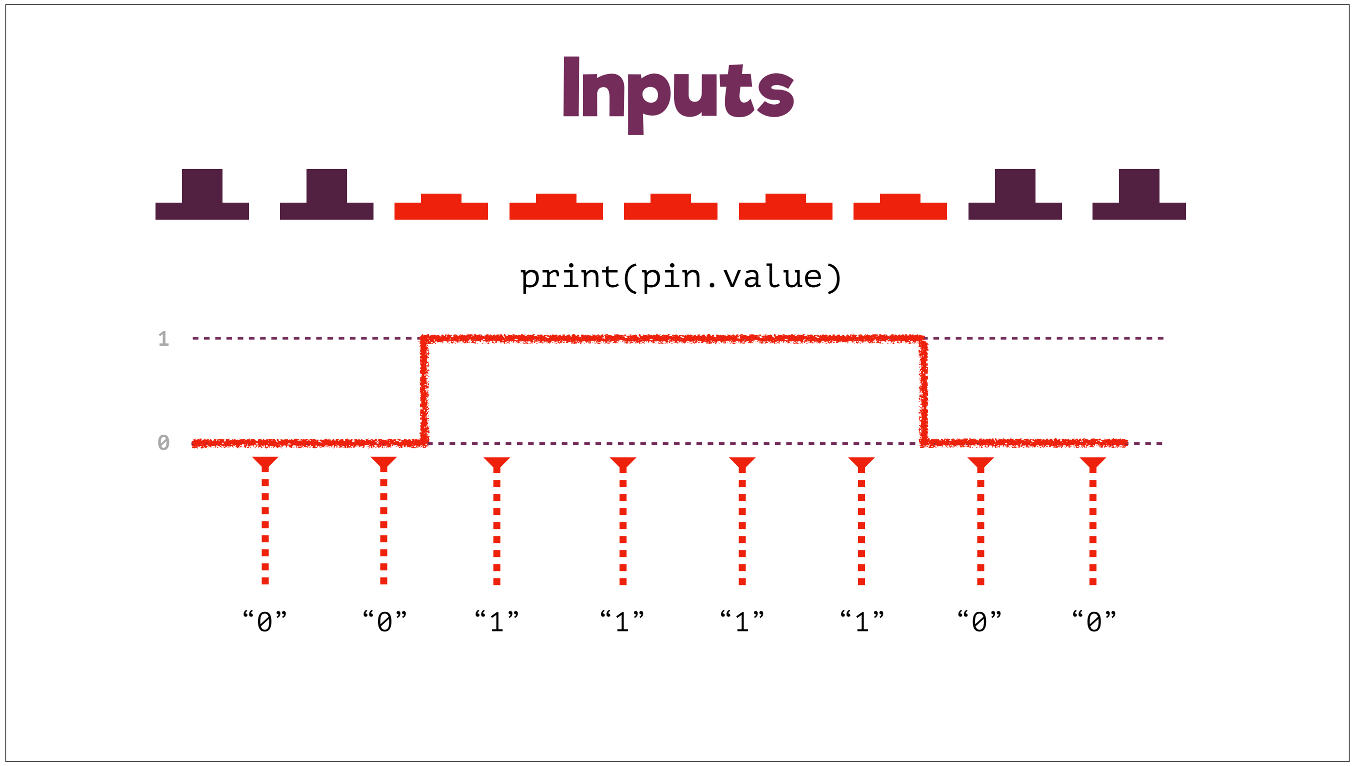
- Looking back to our light example
- Now, we can turn this light on and off in code



Inputs

```
print(pin.value)
```

- If we go back to our button example, we see that you can query the value of the pin
- Well, what if we wanted to do something *when* the pin changed value?

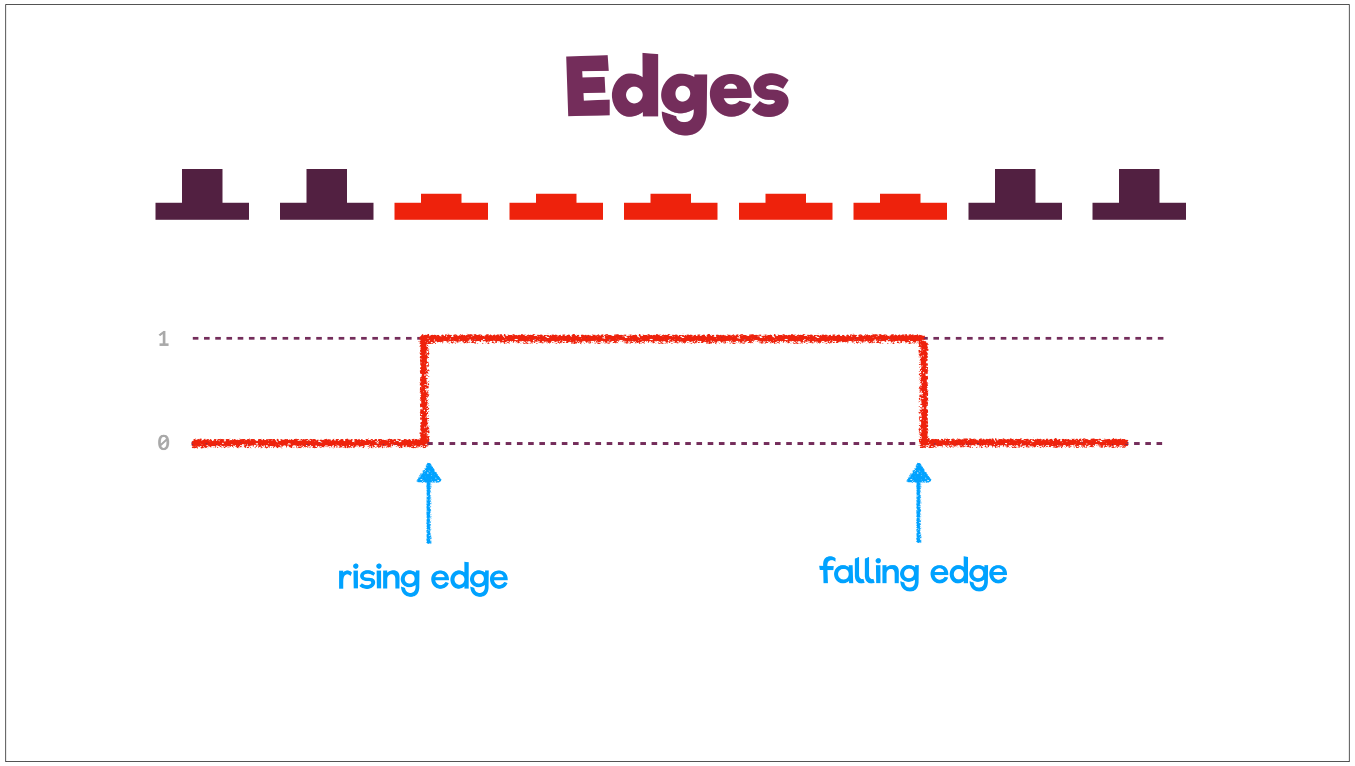


Inputs

```
print(pin.value)
```

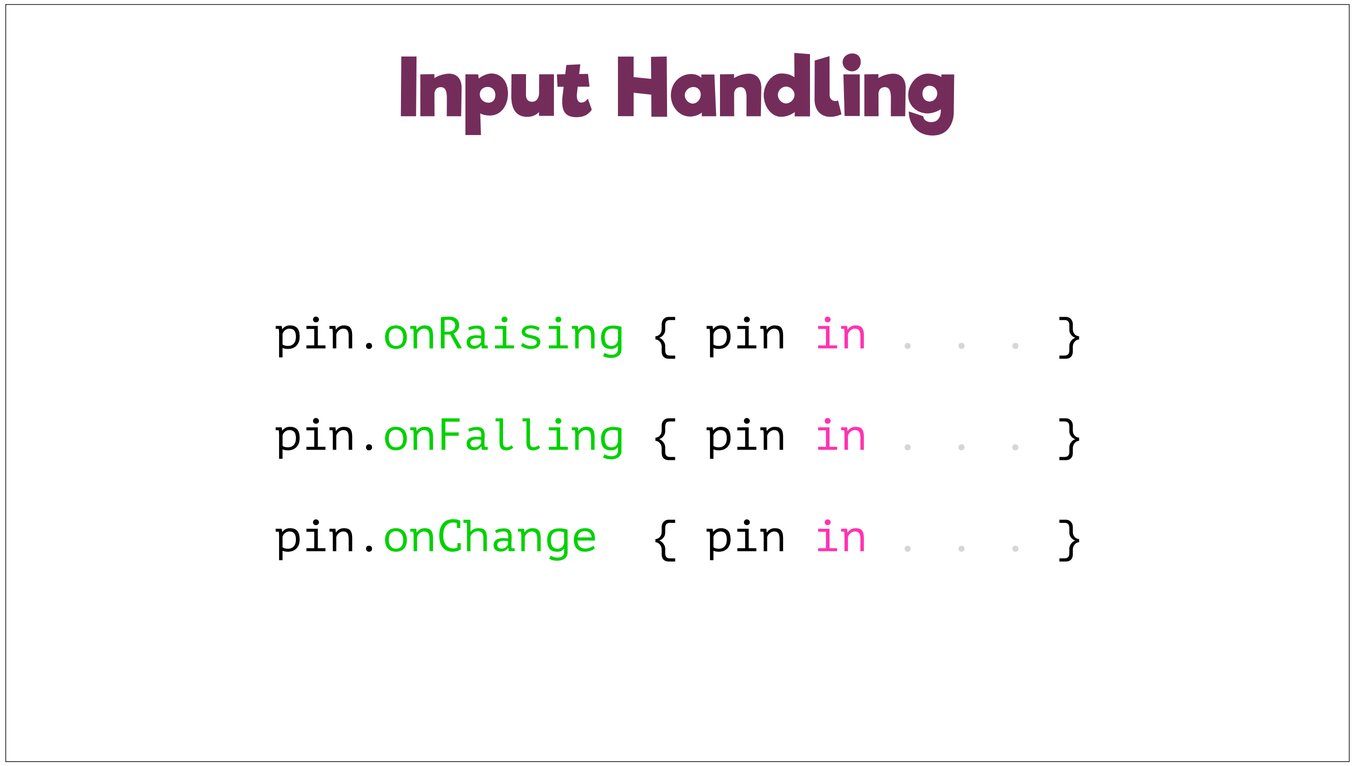
“0” “0” “1” “1” “1” “1” “0” “0”

- You could set up a simple polling cycle, checking the value of the pin every so often
- But this doesn’t seem very efficient
- There’s a better way



Edges

- If we look at the logic diagram, it’s very obvious to us when an event happens that we care about
    - There’s a vertical line
- These are called *edges*
- When the value of the pin is going from 0 to 1, this is a *rising edge*
- When it’s going from 1 to 0, it’s called a *falling edge*



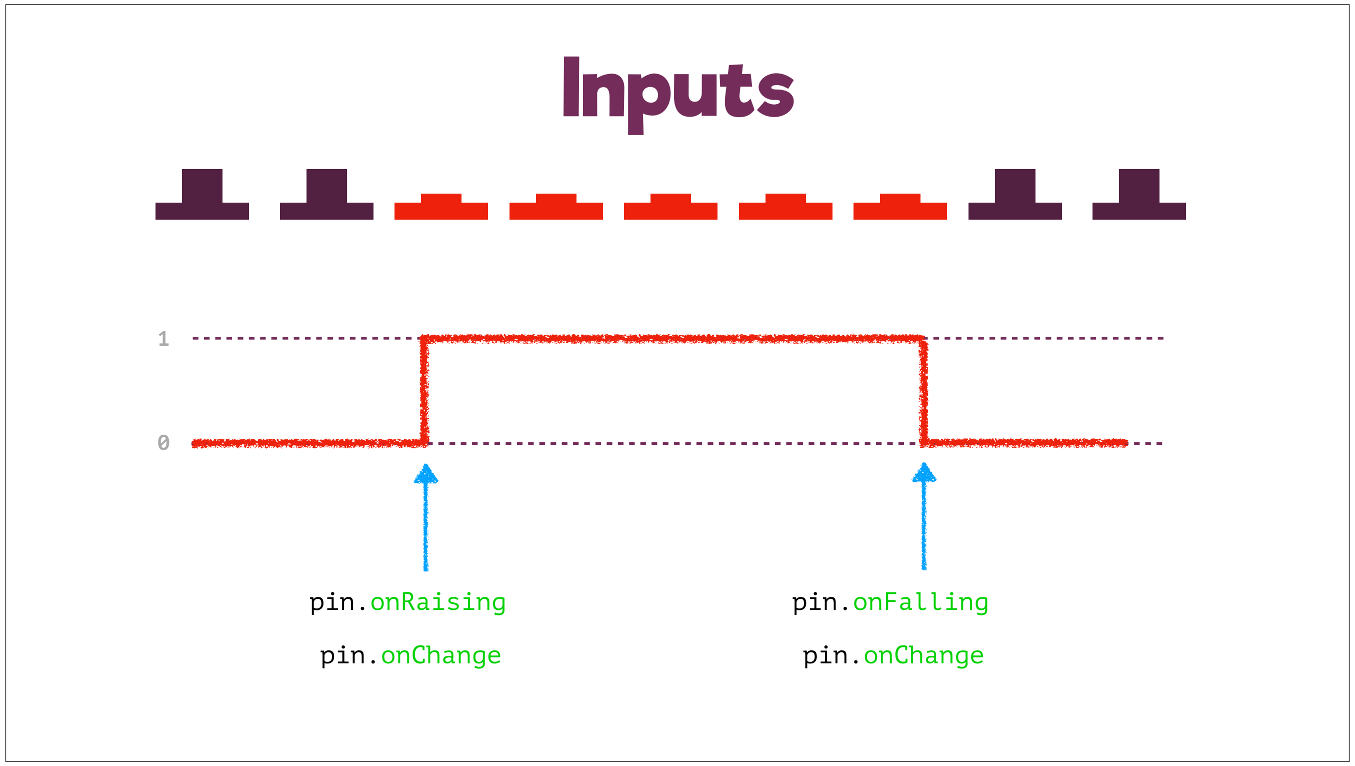
Input Handling

```
pin.onRaising { pin in . . . }

pin.onFalling { pin in . . . }

pin.onChange  { pin in . . . }
```

- SwiftyGPIO gives us a convenient way to listen for edge events on pins
- You can register these handler blocks on any INPUT pin, and they’ll run each time the respective edge is encountered
- There’s also a general `onChange` handler, this will get run whenever the value changes, whether rising or falling



Inputs

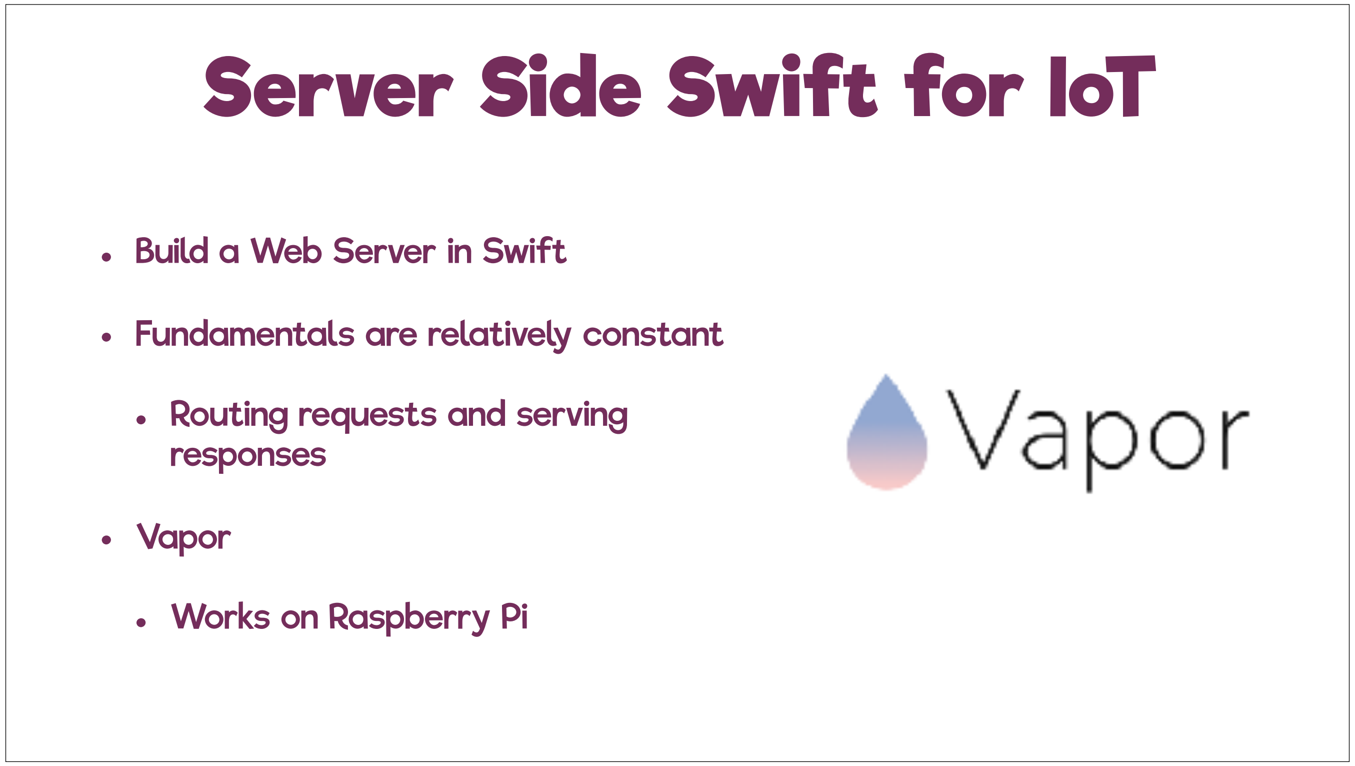
pin.onRaising

pin.onChange

pin.onFalling

pin.onChange

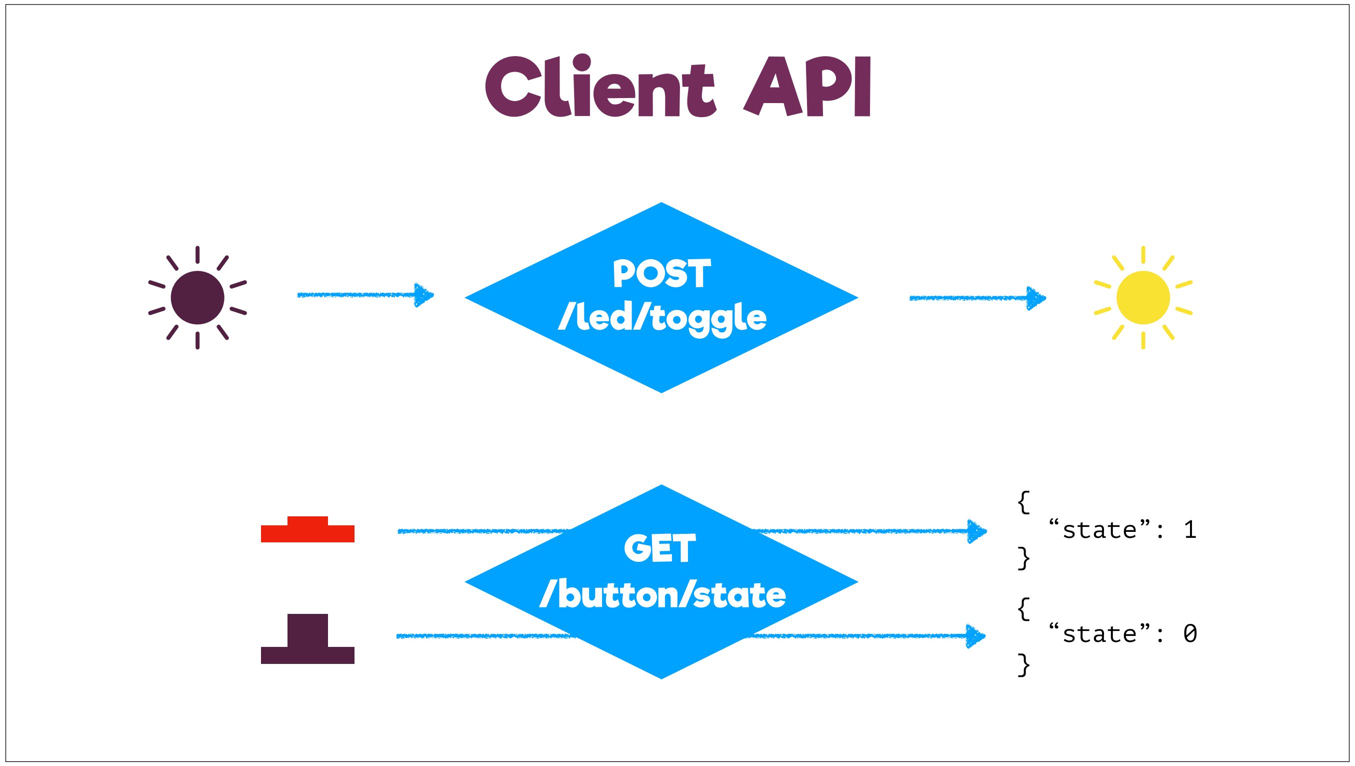
- Now our code is super concise
- Using these callbacks, we only execute code when an event happens that we care about



Server Side Swift for IoT

- Build a Web Server in Swift
- Fundamentals are relatively constant
  - Routing requests and serving responses
- Vapor
  - Works on Raspberry Pi

- Now, let’s move up the stack a bit into the web server component
- Writing a web server in Swift is a pretty hot topic right now
- It’s especially great for iOS developers
  - we can write our backend and our app in the same language
  - potentially share code between the two
- There’s a lot to say about this topic and I’m nowhere near qualified to say it
  - In fact, we have another talk here at Dev Camp today about the topic
- But I do want to talk about how this relates to Raspberry Pi
  - There are many frameworks that enable SSS, but getting them to work on Pi can be a challenge
  - Vapor is a good one, it works on Pi so that’s what we’re going to use



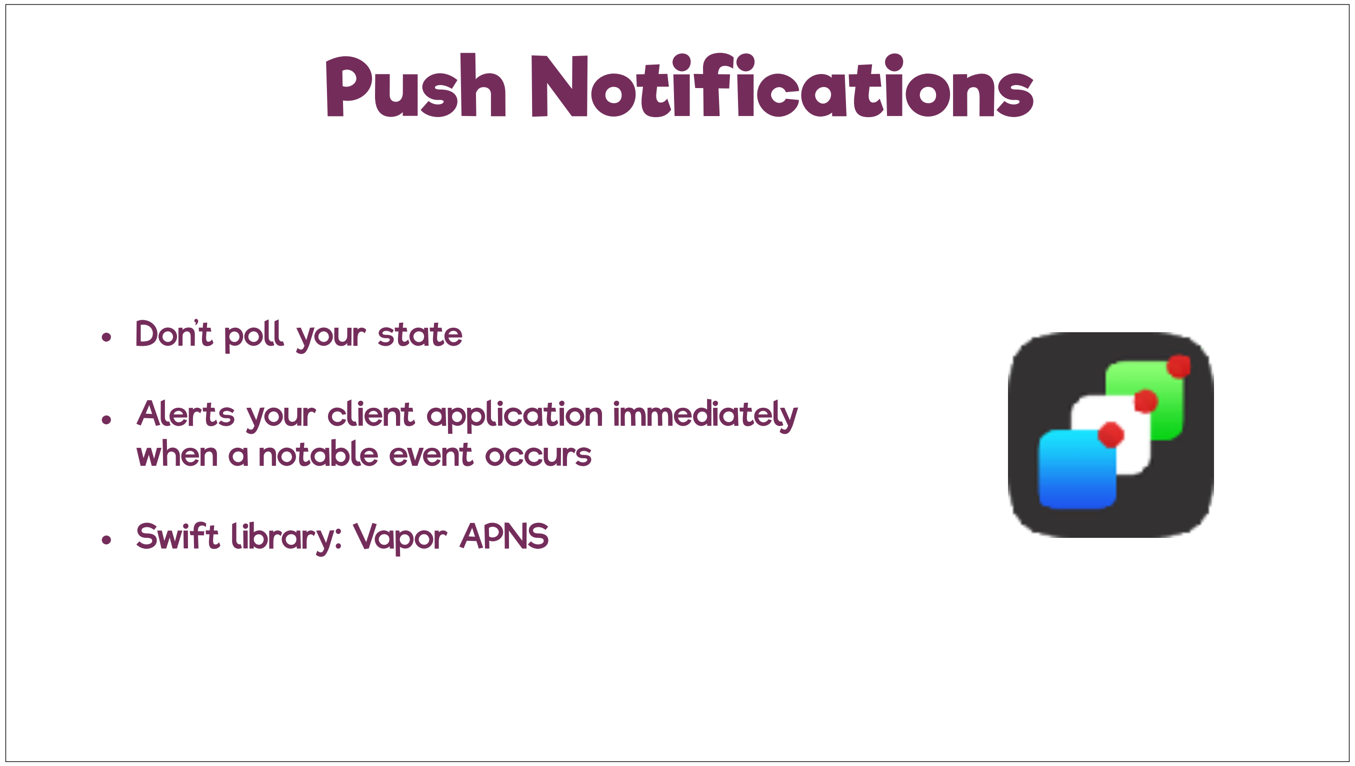
Client API

POST
/led/toggle

GET
/button/state

```
{
  “state”: 1
}
{
  “state”: 0
}
```

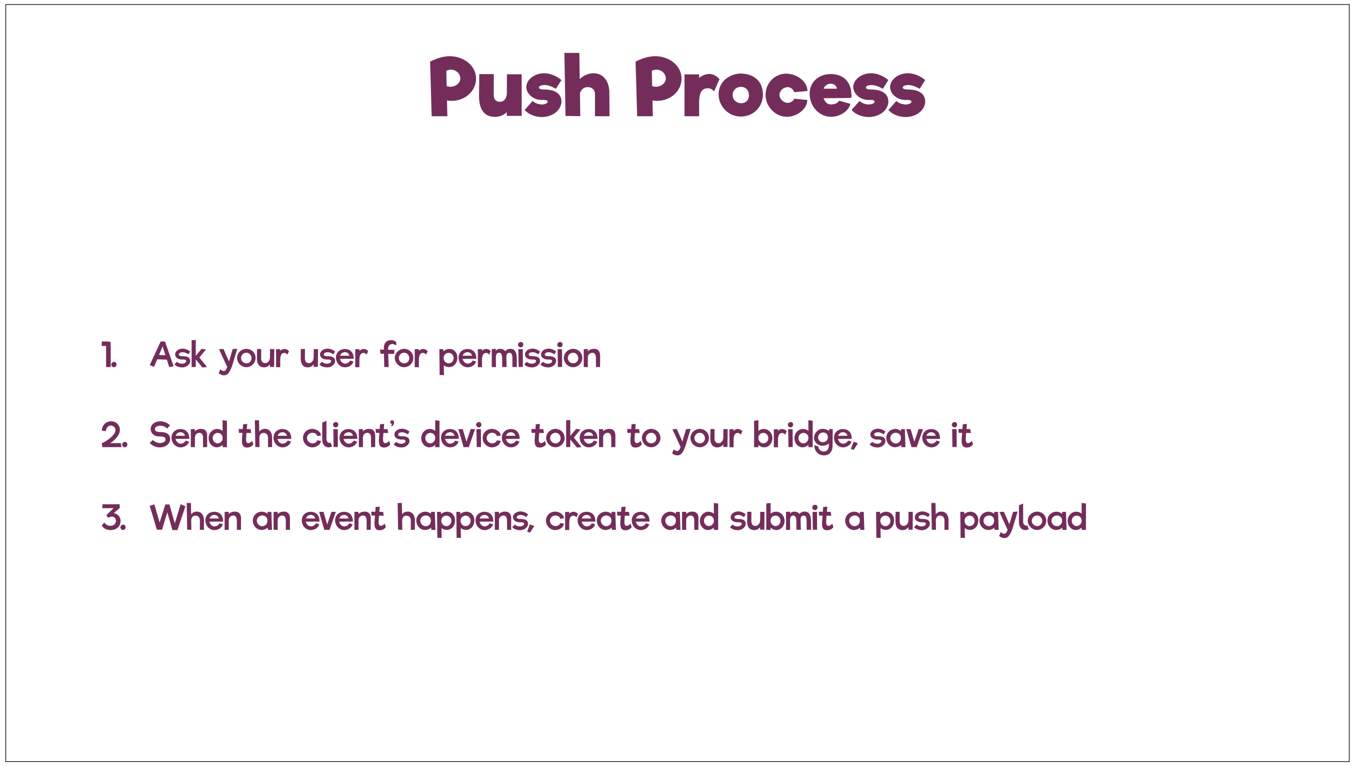
- One thing that’s going to be universal is understanding how to design your API
- When we talk about Internet of Things, the interface can be pretty simple
    - It essentially maps to your electrical interface
- You can use a POST endpoint to set or toggle your outputs
- If you have sensors to read, you can setup some GET endpoints
- Though we’re going to handle our doorbell sensing a different way
    - Remember how we didn’t want to keep polling the GPIO input pin to figure out when it had changed?
    - Well we can use push notifications to provide a similar event-driven model for receiving inputs in our client application on iOS



Push Notifications

- Don't poll your state
- Alerts your client application immediately when a notable event occurs
- Swift library: Vapor APNS

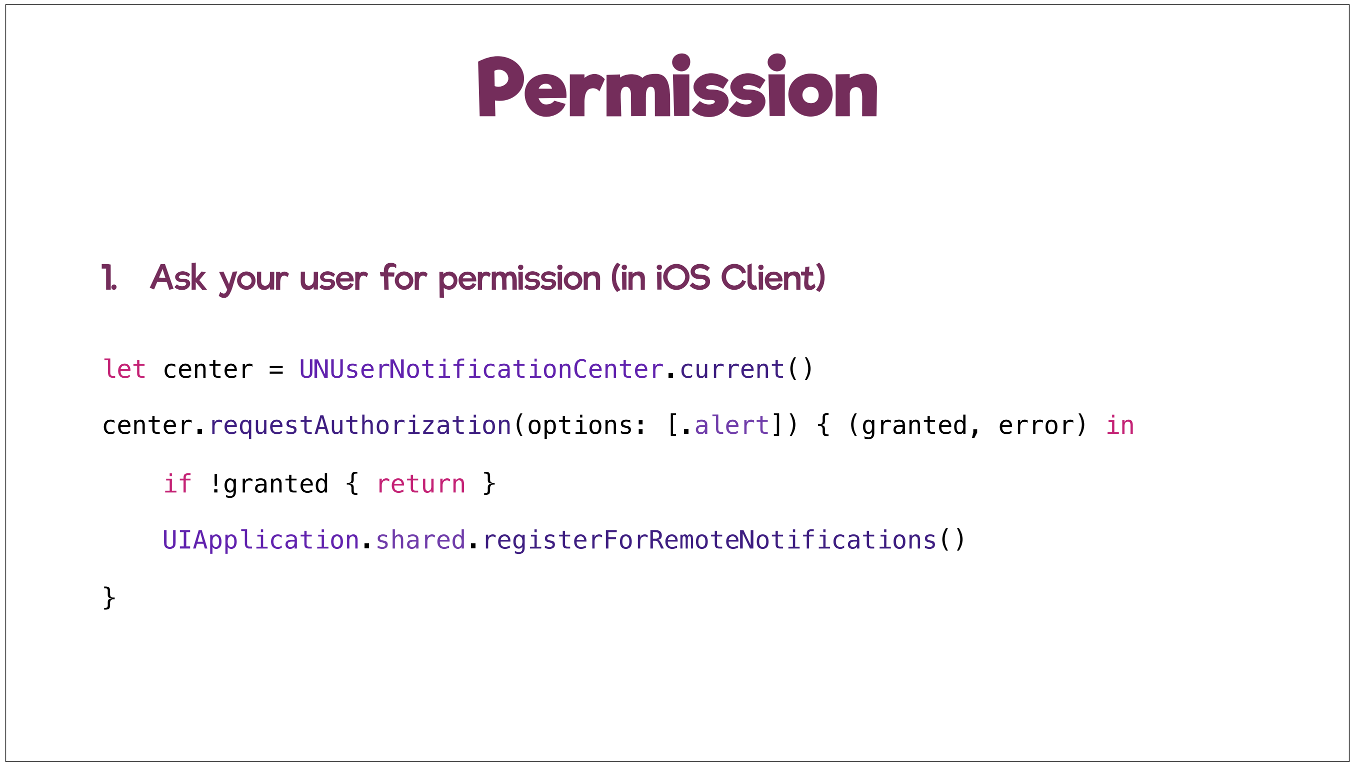
- Push Notifications, what are they?
- They alert us of external events and our app doesn’t necessarily need to be running to receive them
- We know this, many of us iOS developers may be familiar with push notifications, but from the client side, where they come in almost like magic
- Today, we're going to be the ones *sending* the pushes, from our very own server
- There's a great library to help with this, **Vapor-APNS**



Push Process

1. Ask your user for permission
2. Send the client's device token to your bridge, save it
3. When an event happens, create and submit a push payload

- Sending a push isn’t so difficult actually, there’s three main steps
- First, naturally, we always need to ask our users if they want to receive push notifications from us
- Second, you’ll get a device token from iOS —> send that to your bridge and save it
- Finally, when “something” happens (whatever that means for you), send the push!

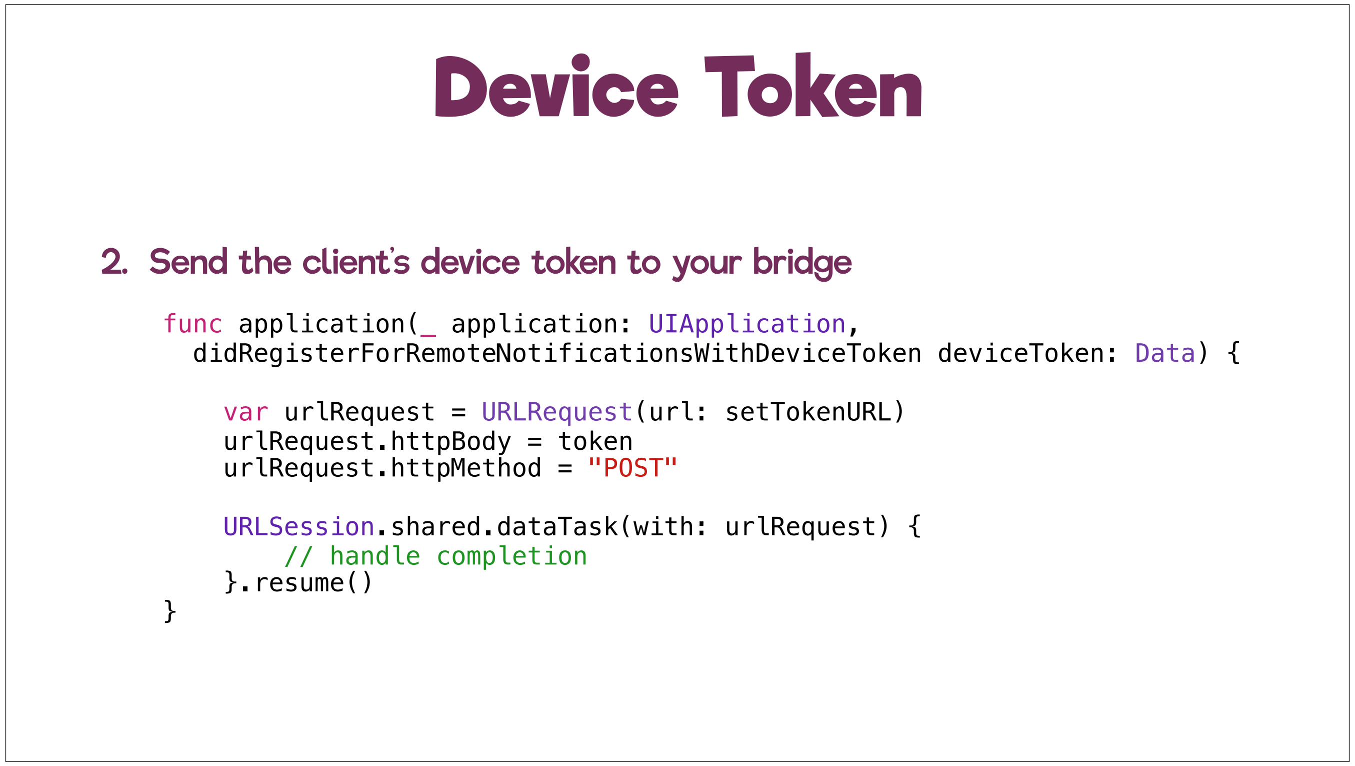


Permission

1. Ask your user for permission (in iOS Client)

```
let center = UNUserNotificationCenter.current()
center.requestAuthorization(options: [.alert]) { (granted, error) in
    if !granted { return }
    UIApplication.shared.registerForRemoteNotifications()
}
```

- I’ll give you just a tasting menu of what this looks like in practice, just some sample code
- Here were in the iOS client
- We’re using the new UNUserNotificationCenter APIs (iOS 10)
- We request authorization to present an alert, this also applies to a banner
- If we’re granted, we can tell our application to register for remote notifications



Device Token

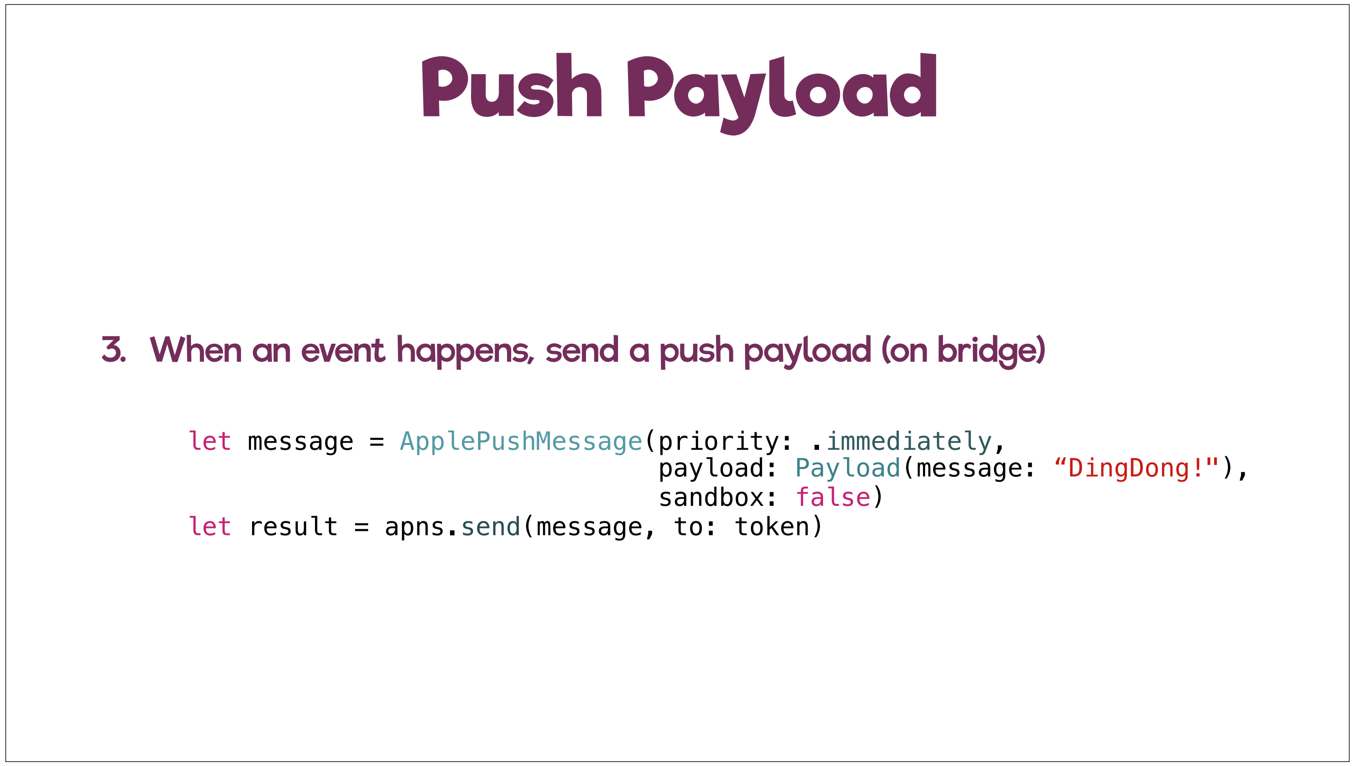
2. Send the client's device token to your bridge

```
func application(_ application: UIApplication,
  didRegisterForRemoteNotificationsWithDeviceToken deviceToken: Data) {

    var urlRequest = URLRequest(url: setTokenURL)
    urlRequest.httpBody = token
    urlRequest.httpMethod = "POST"

    URLSession.shared.dataTask(with: urlRequest) {
        // handle completion
    }.resume()
}
```

- In the app delegate, if all goes according to plan, we'll get a `deviceToken` back
- Using a simple POST request, we can send that token up to our bridge

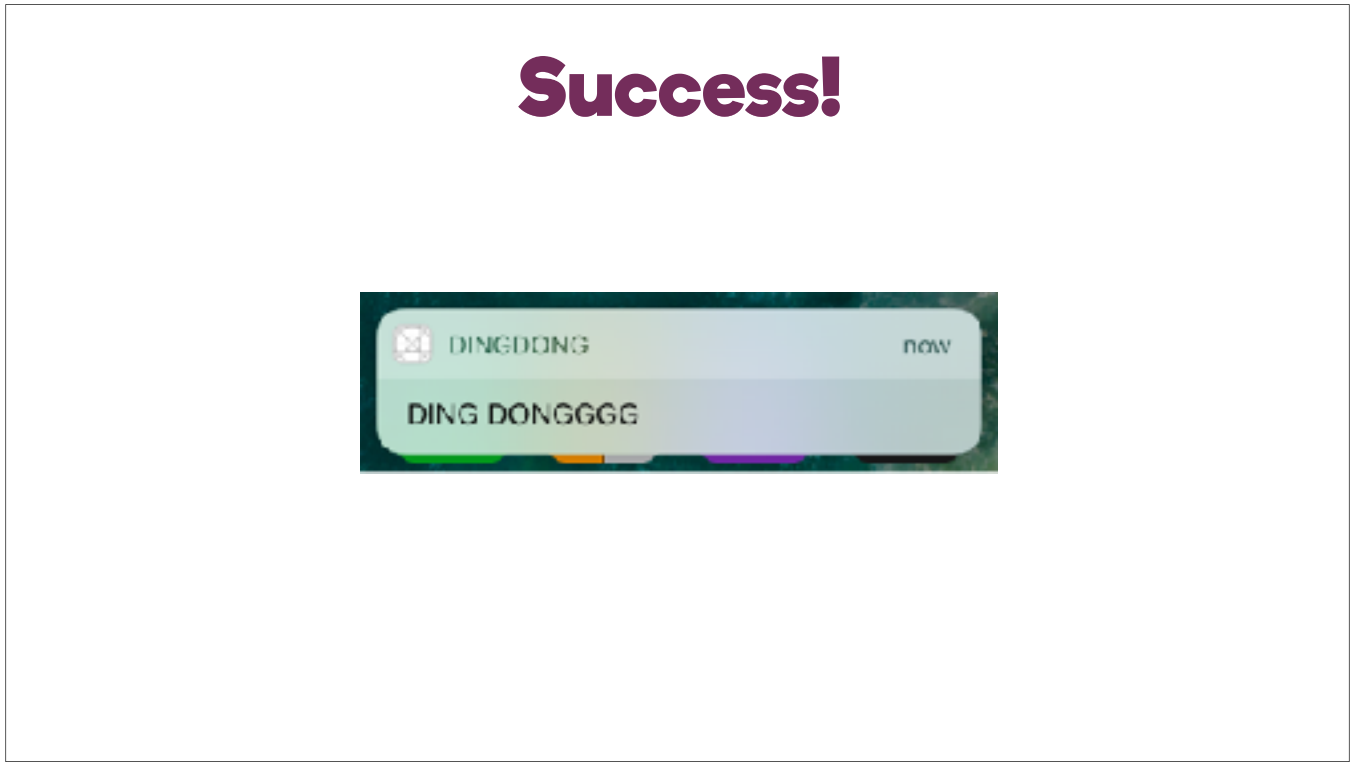


Push Payload

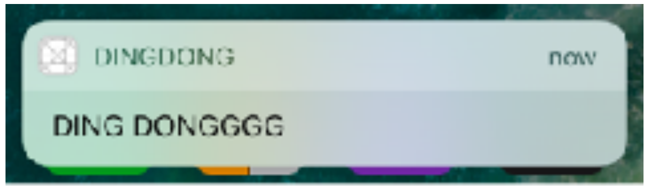
3. When an event happens, send a push payload (on bridge)

```
let message = ApplePushMessage(priority: .immediately,
                                payload: Payload(message: “DingDong!"),
                                sandbox: false)
let result = apns.send(message, to: token)
```

- After the bridge saves that token, we can use it to send a push payload
- There are other parameters available to add to the payload, but here we’re just sending “Ding Dong!”

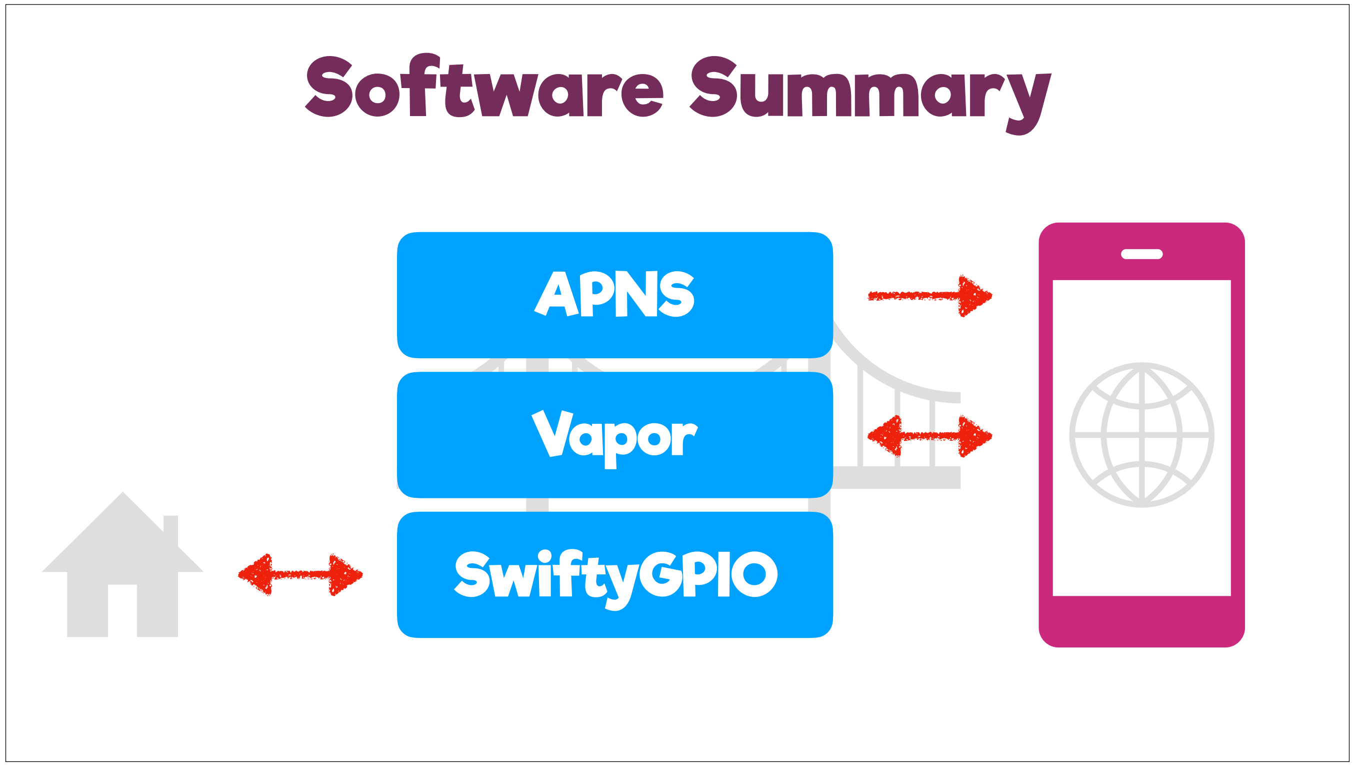


Success!



DING DONGGGG

- And that's more or less all there is to it!
- There was some other initial setup and provisioning I needed to do initially, but us iOS developers are used to that
- You'll also want to handle errors, obviously
- I walked through these code samples just to illustrate that the general flow of sending a push is not too complicated



Software Summary

APNS

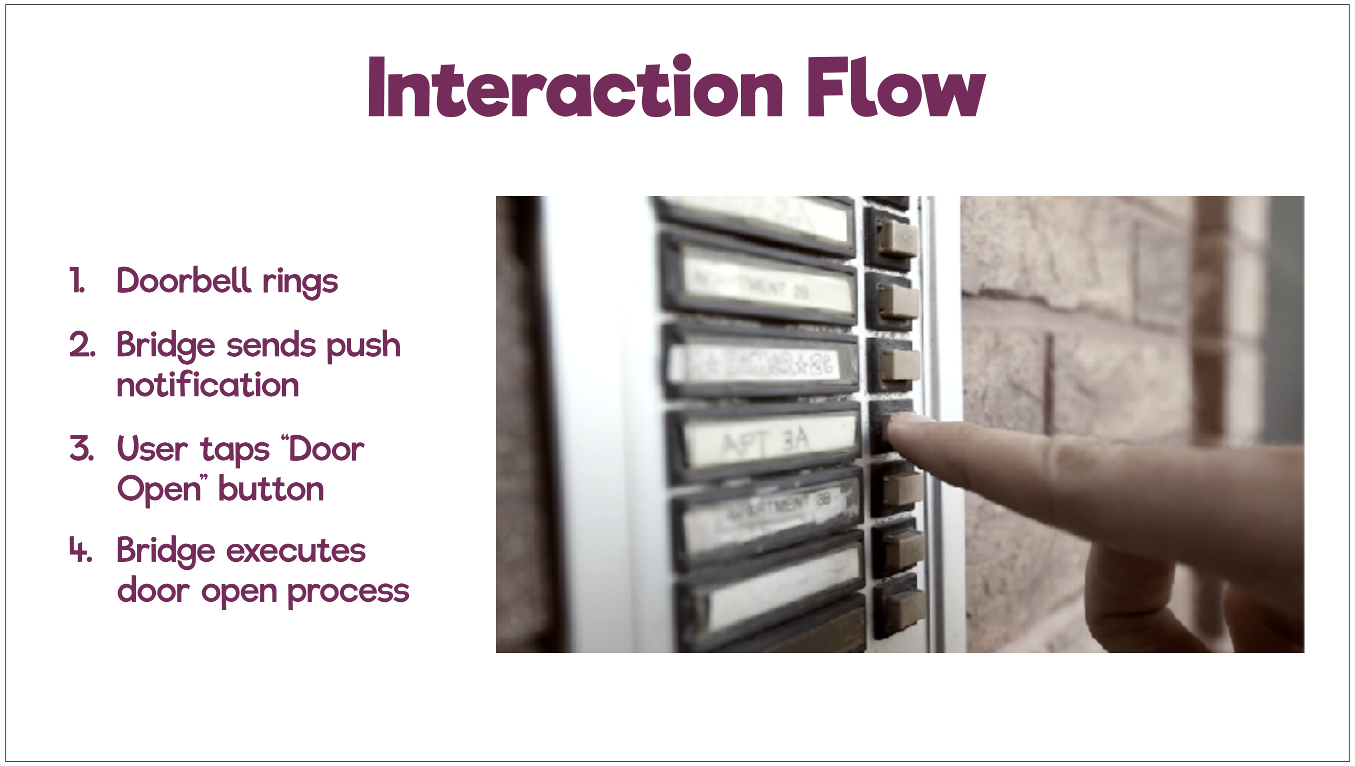
Vapor

SwiftyGPIO

- That was a whirlwind, let’s do a quick summary of software
- At the lowest level, we have SwiftyGPIO running our GPIO pins and giving us an interface to “Things”
- As the backbone of our bridge, we built a web API using Vapor, a Server-side Swift framework
- On top, we’re using push notifications to provide event-driven communications
- This all can communicate with a simple client app that we build for iOS

Swifty Doorbell

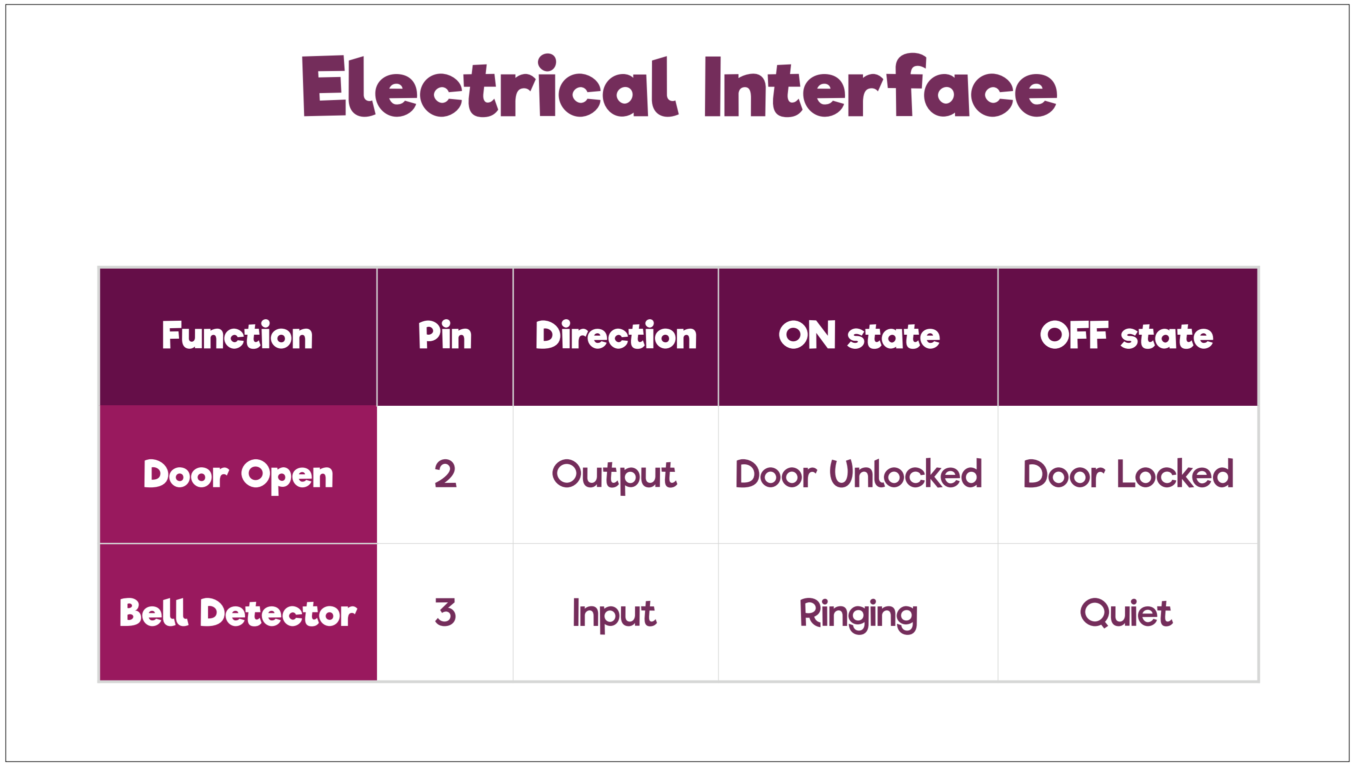
- It’s been a long road, but I can finally talk about how my doorbell runs Swift



Interaction Flow

1. Doorbell rings
2. Bridge sends push notification
3. User taps "Door Open" button
4. Bridge executes door open process

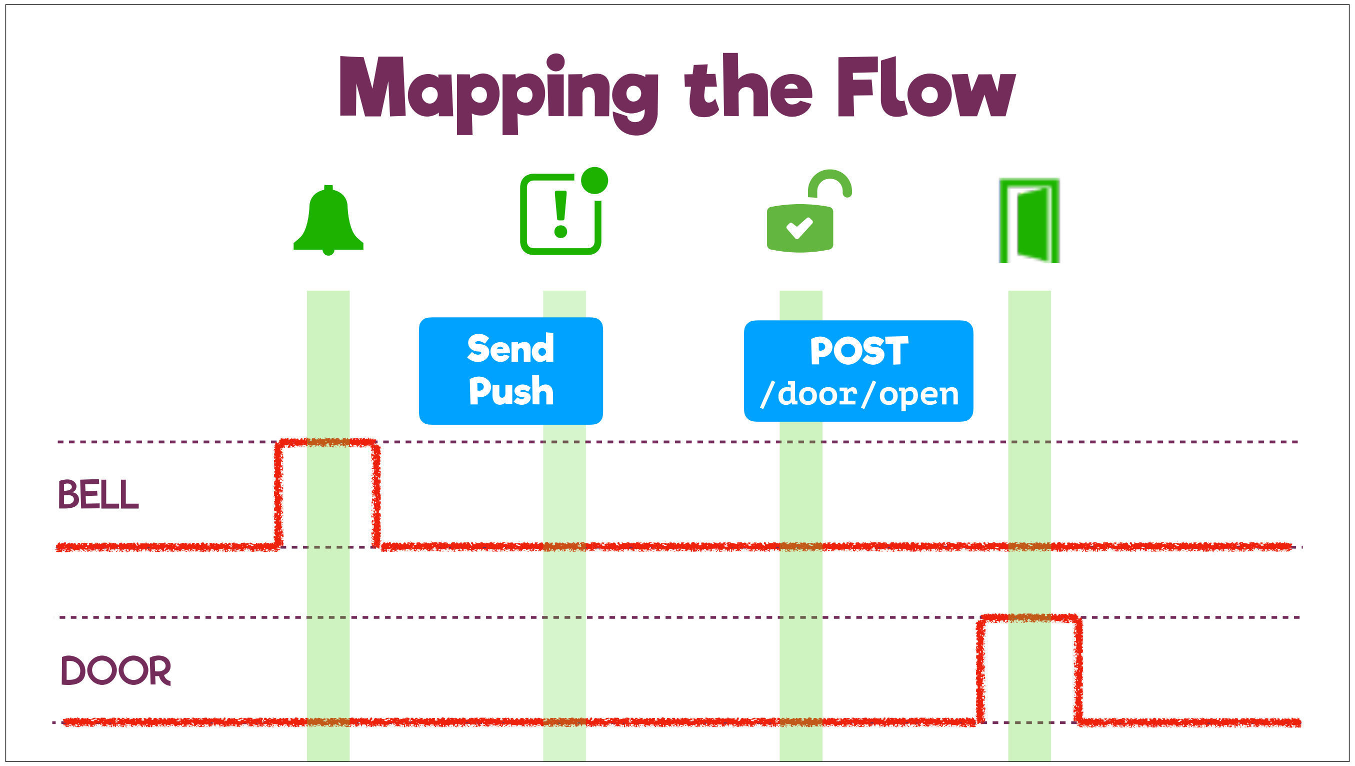
- Lets go back to the demo reel and tease apart the different steps of this interaction



Electrical Interface

Function	Pin	Direction	ON state	OFF state
Door Open	2	Output	Door Unlocked	Door Locked
Bell Detector	3	Input	Ringing	Quiet

- I started by laying out what I needed to interface with on the “Things” side
- For each function I want to control, I’ll allocate a GPIO pin
- It’ll be either an input or an output
- Then, for each of the ON and OFF states, what does that mean for this function?
- First is “Door Open”
  - I’m connecting this to pin 2 because the Raspberry Pi’s GPIO pins actually start at 2
    - I don’t make the rules!
  - It’s an output
  - When I turn the pin on, it’ll be like I’m pressing the door open button, so the door will be unlocked
  - When it’s off, that’s the resting state, the front door is going to be locked like normal
- The other one is what I’ll call “Bell Detector”
  - I made a simple circuit that maps the ringing of my door bell into this ON/OFF state
  - This way I can hook it into GPIO, just like any other sensor
  - It’s going to be an input on pin 3



Mapping the Flow

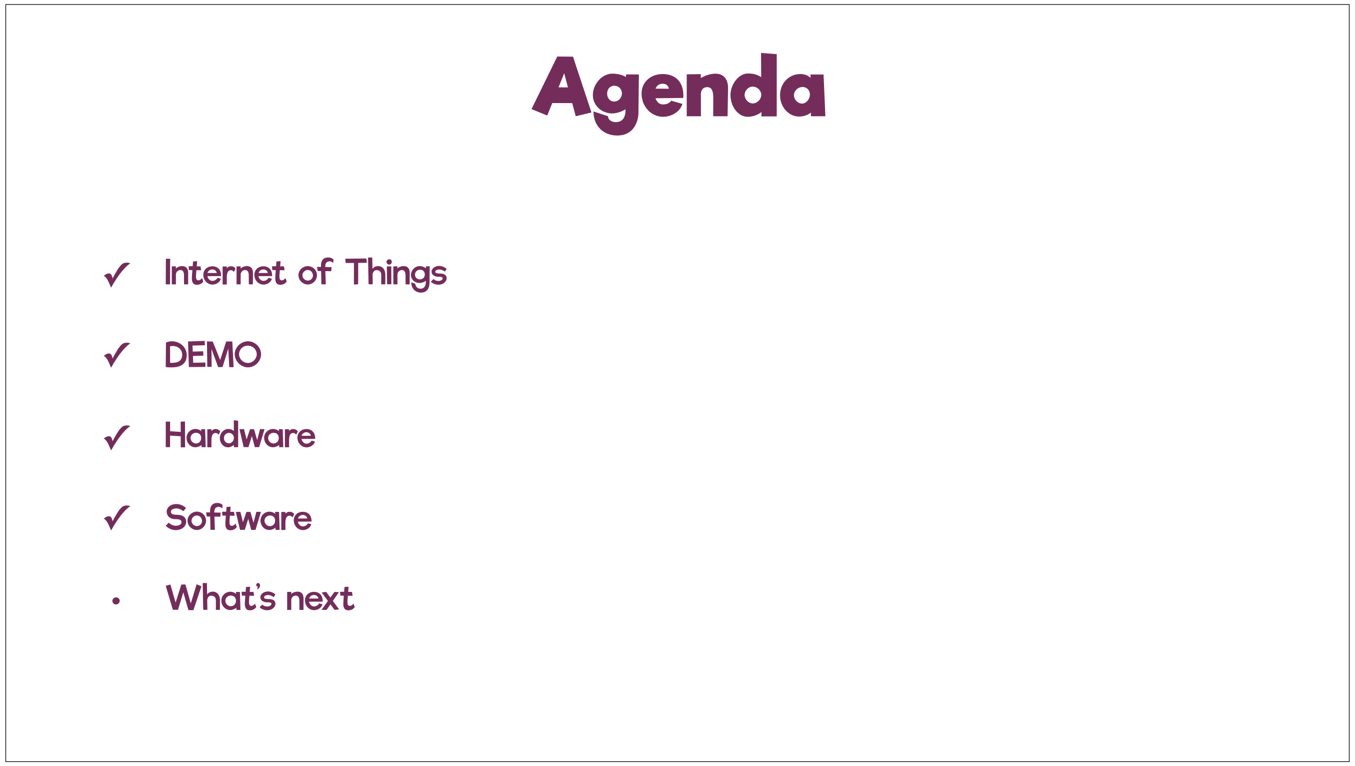
Send Push

POST /door/open

BELL

DOOR

- Going back to our 4 steps, we see the timeline of events and how everything interacts
  - Doorbell rings
    - The bell goes from LOW to HIGH for some time, then back to LOW
    - This triggers the push
  - Bridge sends push notification
    - Homeowner receives
  - User taps “Door Open” button
    - API request to our bridge to open the door
  - Bridge executes door open process
    - Holds the door open pin HIGH for some period of time, then brings it back LOW
- In conclusion, I hope this shows how some relatively simple technology can enable cool interactions



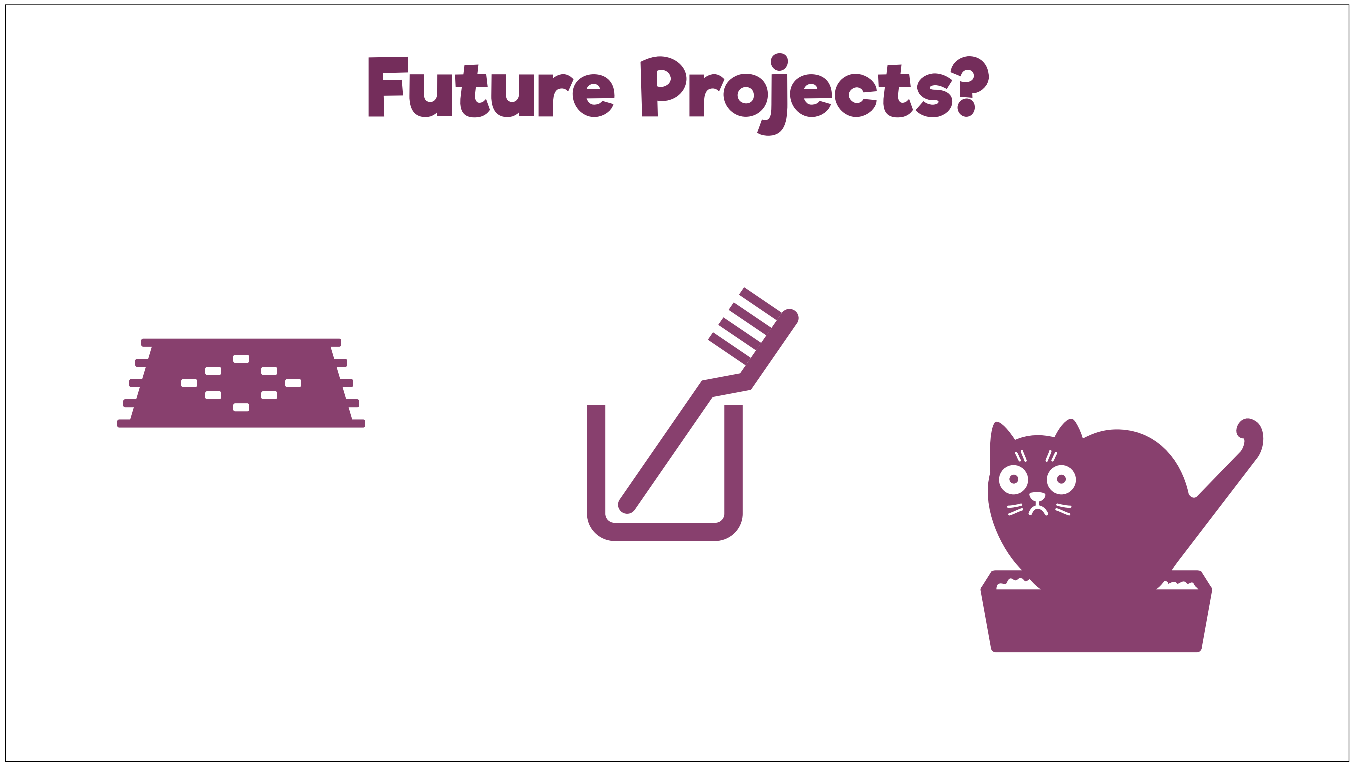
Agenda

- ✓ Internet of Things
- ✓ DEMO
- ✓ Hardware
- ✓ Software
- What's next

- That was our bridge software as well as a more specific peek into my doorbell running Swift

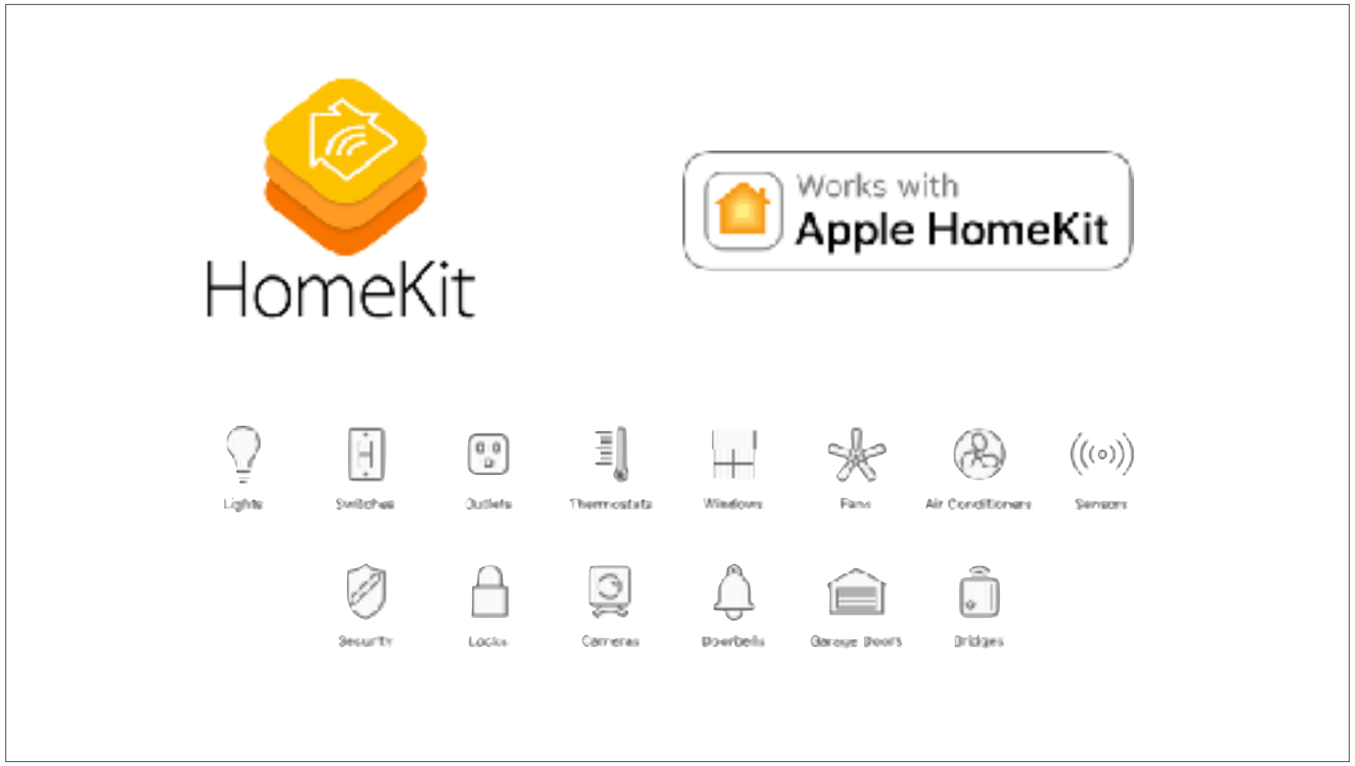
What's Next?

- But where does that leave you?
- What’s next?



Future Projects?

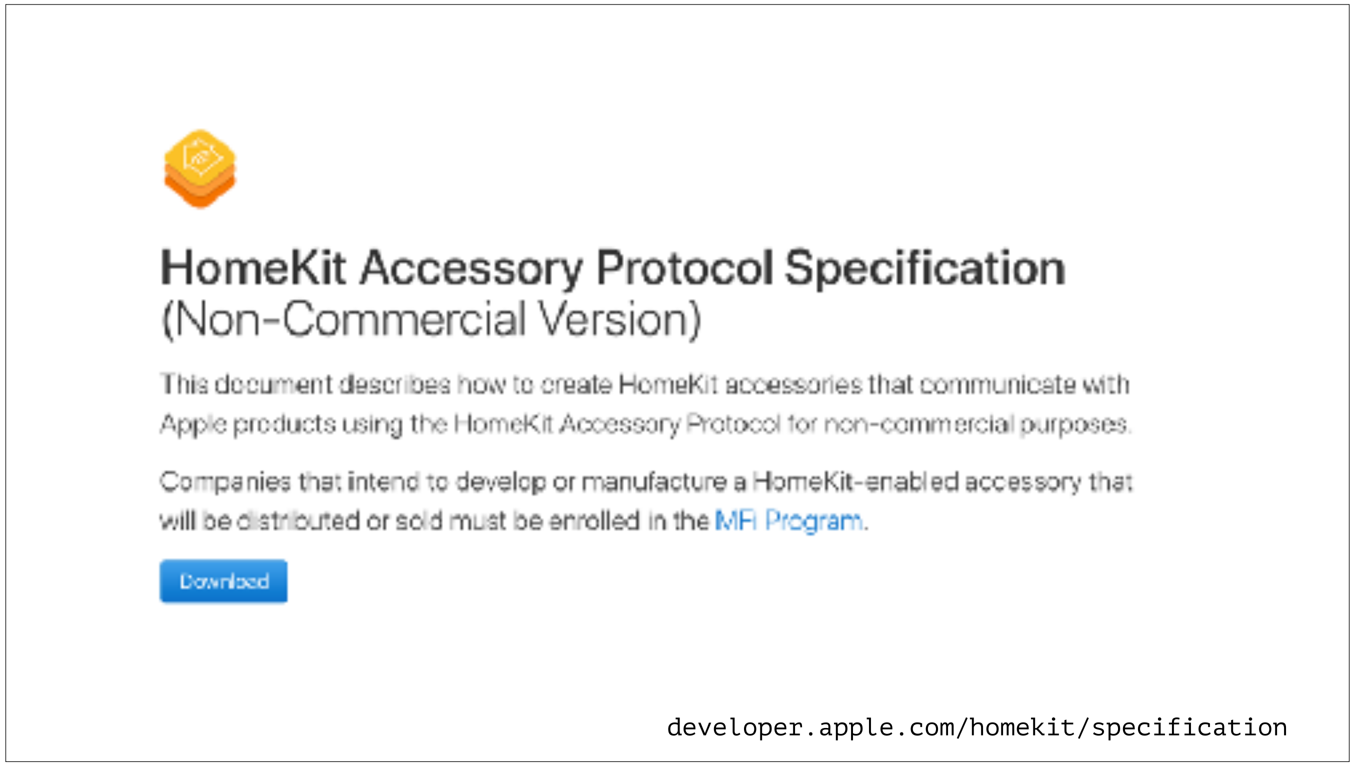
- Well, like I said in the beginning, this talk is nothing if it’s not a template for even cooler projects that could be built with a custom bridge
- Here’s just some thoughts I was spitballing the other day
    - Put a button under a mat next to your bed. When you step on it in the morning, send a push notification with the weather.
    - Hook a sensor up to your toothbrush cup. Measure the amount of time you spend brushing and shame yourself on Twitter if its not long enough
    - Detect motion near your cat’s litter box and be alerted if your attention is needed, because I'm sure you all would sign up for those notifications



HomeKit

Works with Apple HomeKit

- Maybe you’ve heard of HomeKit
- This is Apple’s framework for the “Internet of Things”
- There are tons of devices that support this standard
- Developers can create apps that interact with HomeKit devices



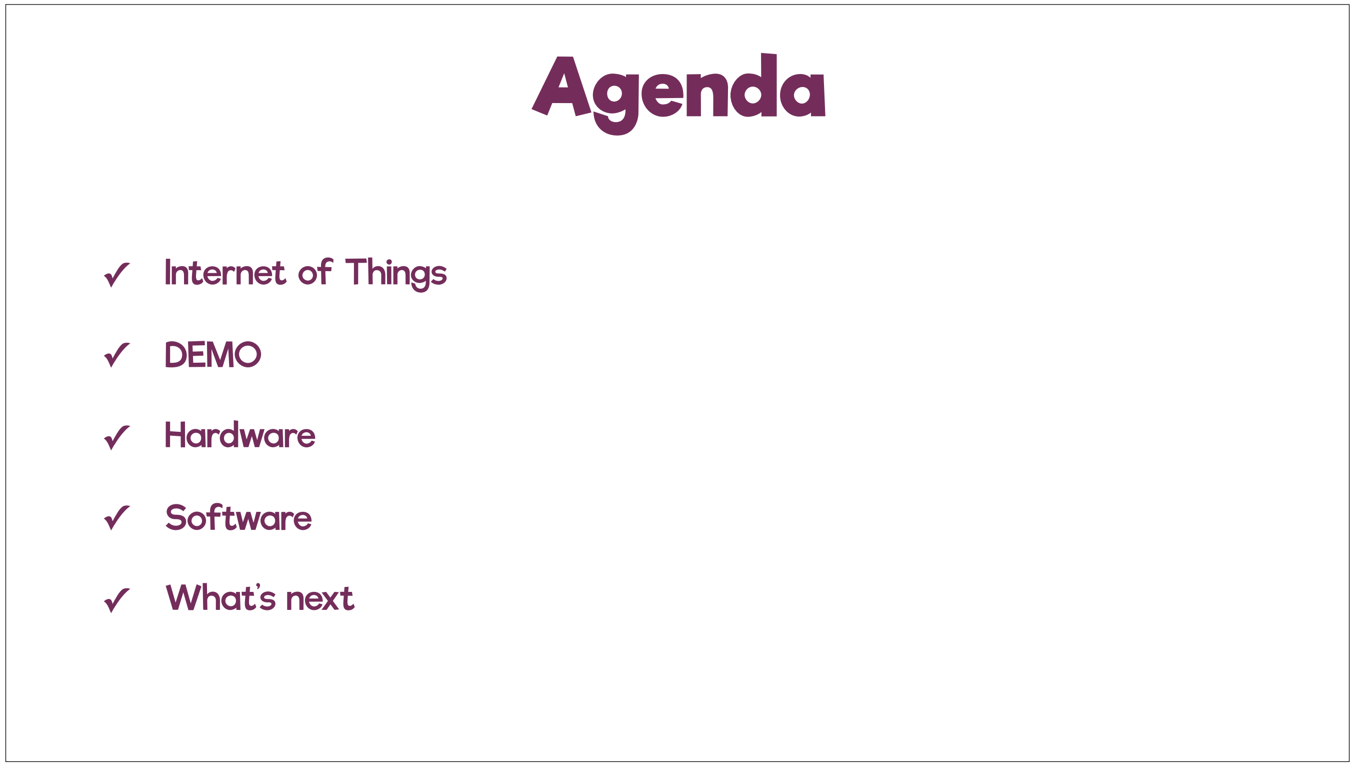
HomeKit Accessory Protocol Specification
(Non-Commercial Version)

This document describes how to create HomeKit accessories that communicate with Apple products using the HomeKit Accessory Protocol for non-commercial purposes.

Companies that intend to develop or manufacture a HomeKit-enabled accessory that will be distributed or sold must be enrolled in the MFi Program.

developer.apple.com/homekit/specification

- At WWDC this year, Apple threw us a curveball by announcing a Non-Commercial Version of their HomeKit Accessory Protocol
- What this means for you is that you now don’t need to be a Fortune 500 company with a manufacturing partner to participate in the HomeKit ecosystem
- The authentication can now all be implemented in software
- It *also* could be implemented on our Raspberry Pi bridge, then we could do all the neat HomeKit stuff for free



Agenda

- ✓ Internet of Things
- ✓ DEMO
- ✓ Hardware
- ✓ Software
- ✓ What's next

- That about wraps it up

Thanks!

@huebnerob

- So I hope y'all enjoyed this chat about IoT, Raspberry Pi's, my Doorbell, etc. etc.
- If you have any thoughts later, feel free to reach out to me on twitter
- Thank you so much to iOSDevCampDC, I hope you all have a great rest of the conference!