IOT: Introductory Workshop

FunGram LLC
June 2015

Introductions

(Around the room)

Instructors

- **Sasi Movva**
  - Entrepreneur building a startup in the Internet of Things domain
  - Masters (S/W Engg) from CMU, Pittsburgh and MBA (Leeds School of Business), CU Boulder
  - Extensive software programming experience including embedded software seen in most mobile phones
  - Loves hiking, biking and camping with his wife and 2 boys

---

- **Shashank Shekhar**
  - 4 years as a co-founder of a startup (failed)
  - Masters (CS) from CU, Boulder and B.Tech. (EE) from IIT Kanpur, INDIA
  - Expert at hardware chip design (VLSI)
  - Software development includes everything from websites (RoR) to embedded software (C)
  - Loves golfing, working on getting his wife and 2 boys into it

IOT Development Platforms

Platform comparisons

	Spec./Features	Conne ctivity	IDE
Arduino Uno	• ATmega328 (8-bit) microcontroller • 14 Digital I/O Pins (6 PWM outputs) • 32k Flash Memory	Shields	Arduino IDE
Spark Core	• Fits in a standard breadboard • FCC/CE/IC certified • STM32F103 ARM Cortex M3	WiFi	SparkBu ild Web IDE
Raspberry Pi	• Broadcom BCM2835 700MHz ARM1176JZFS processor with FPU and Videocore 4 GPU • 512 Megabytes of RAM • HDMI, SD Card socket	Ethernet /USB/ Shields	Multiple IDEs

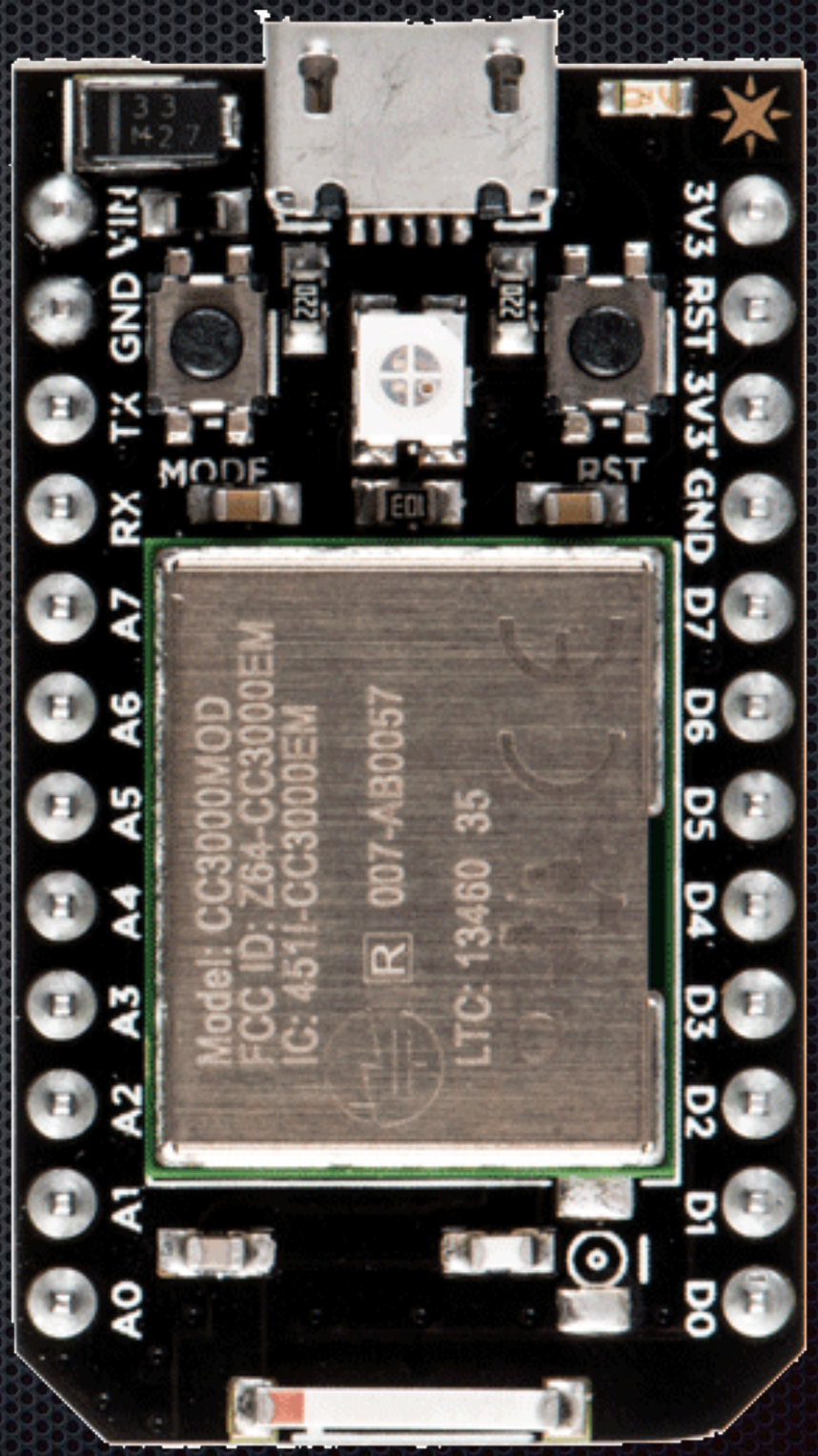
Platform comparisons

	Spec./Features	Connect ivity	IDE
Beagle Bone	• AM335x 1GHz ARM® Cortex-A8 • 2GB of on-board flash and a microSD card reader • HDMI • 2x 46 pin headers	Ethernet	Eclipse, Visual Studio
Intel Edison	• Dual-core CPU (500 MHz) + Single-core micro-controller (100 MHz) • 40 multiplexed GPIOs • 4 GB eMMC flash	WiFi, Bluetooth LE	Eclipse, Intel XDK IoT Edition, Arduino

Design Trade-offs

- Processing power => CPU MHz, Dual core/Single core
- Memory => on board storage, expansion capabilities
- Connectivity => wireless, wired, low energy or powered
- Physical dimensions => TinyDuino, LilyPad etc.
- S/W development ease => community, s/w libraries

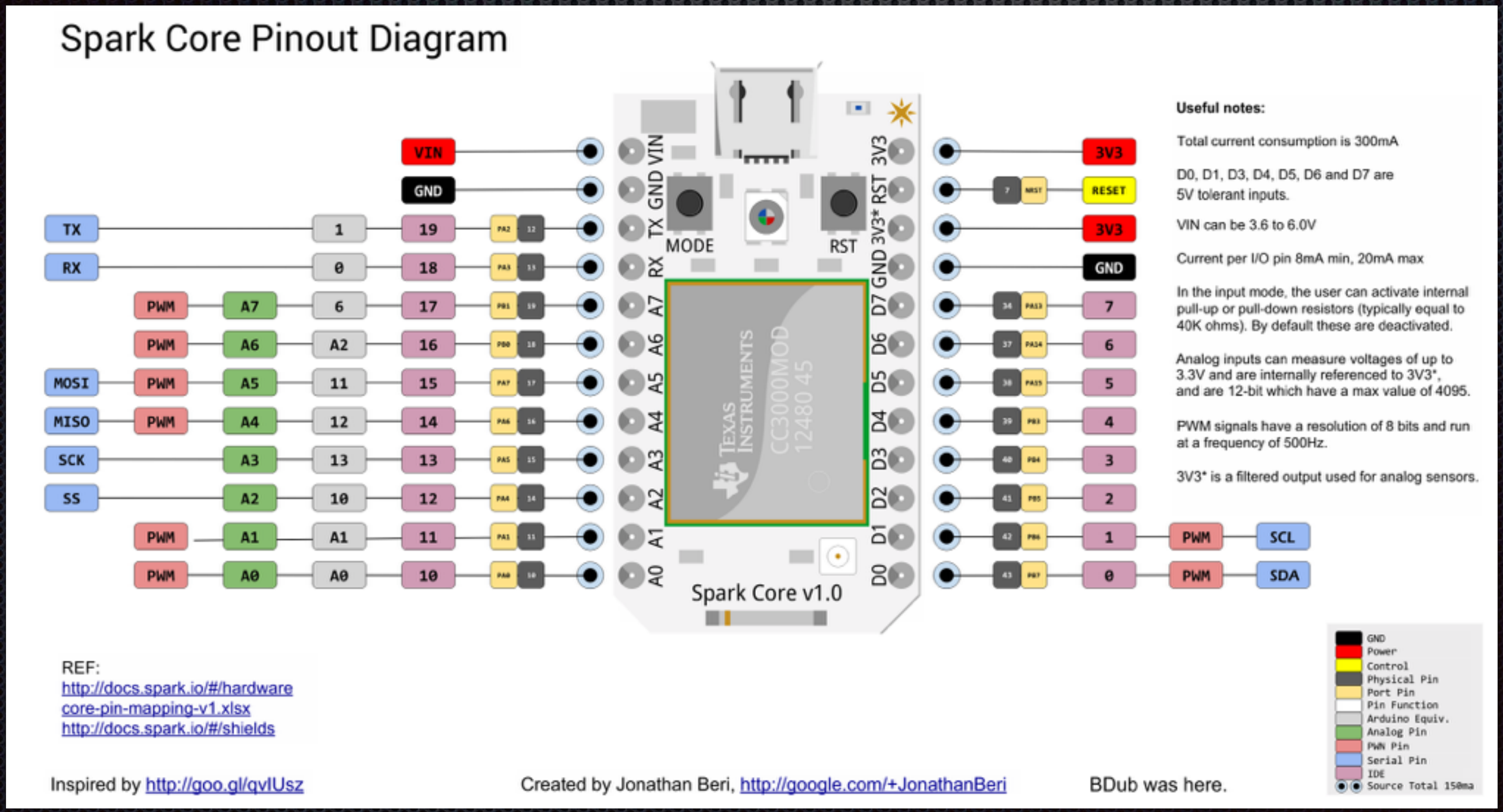
Spark Core


A0 A1 A2 A3 A4 A5 A6 A7 RX TX GND VIN
MODE
Model: CC3000MOD
FCC ID: Z64-CC3000EM
IC: 451I-CC3000EM
R 007-AB0057
LTC: 13460 35
RST
3V3 RST 3V3 GND D7 D6 D5 D4 D3 D2 D1 D0

Hardware - Overview

- ARM 32-bit Cortex™-M3 CPU Core (72 MHz)
- 128KB internal Flash, 2MB external Flash, 20KB of SRAM
- 12 bit ADC
- Onboard power regulator runs the device at 3.3V DC
  - 500mA output current
- Maybe powered by: 3.6V LiPo battery, 4AA battery pack, backup USB battery or an USB wall charger.

Hardware - Pin Diagram

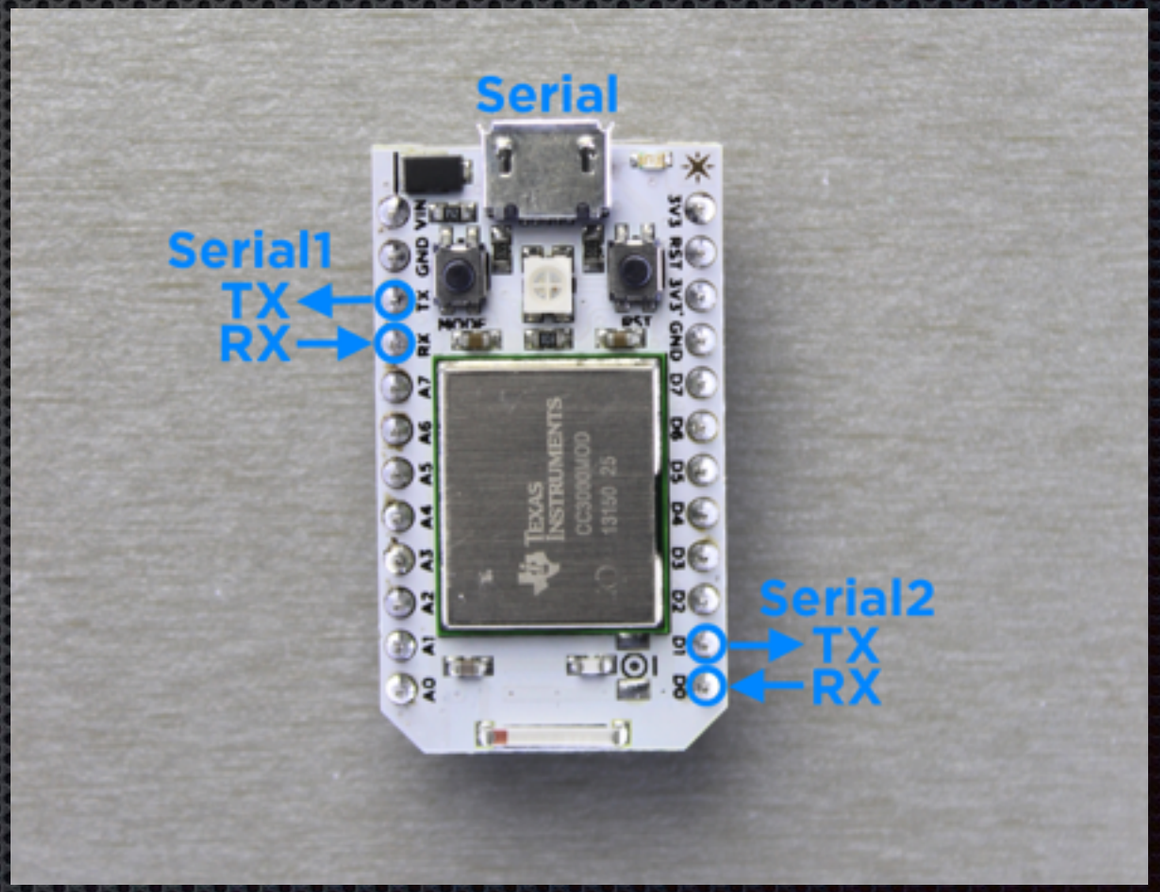


Hardware - Connectivity

- WiFi
  - IEEE 802.11 b/g compliant
  - Wireless security protocols
    - WEP/WPA Personal/ WPA2 Personal

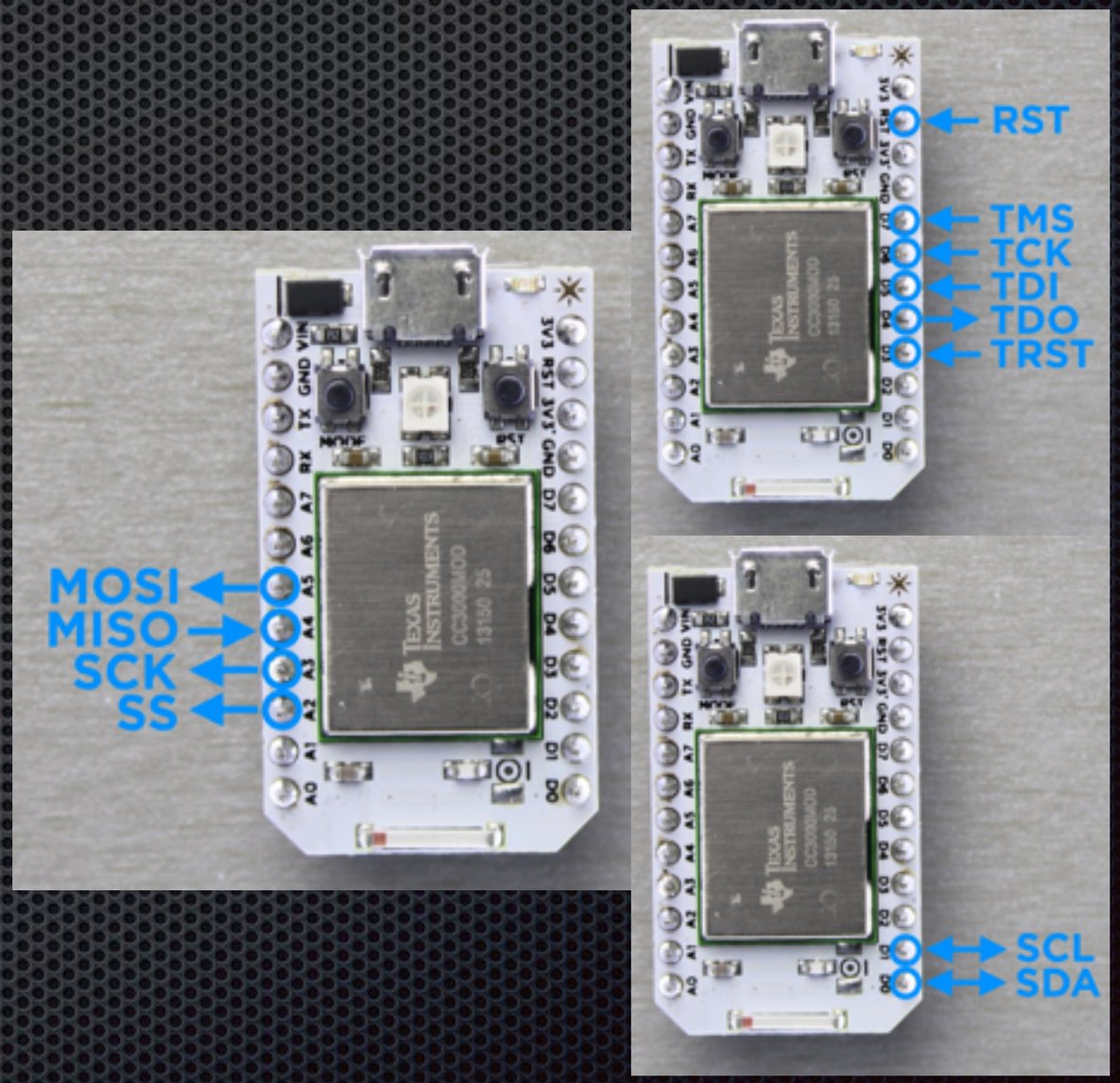
Hardware - Connectivity

- Three serial interfaces
  - A CDC (Communications Device Class) available over the USB port, shows up as a virtual COM port on the computer. <u>*Usage:*</u> *Serial.begin(9600)*
  - A hardware USART available via the TX and RX pins on the Core. <u>*Usage:*</u> *Serial1.begin(9600)*
  - A hardware USART available via the D1(Tx) and D0(Rx) pins on the Core <u>*Usage:*</u> *Serial.begin(9600)*



Hardware - Connectivity

- SPI
- I2C
- JTAG
- Additional information about hardware: http://docs.particle.io/core/hardware/



System Modes

- Three system modes for connectivity to the cloud
  - AUTOMATIC: automatic connect, automatic message processing.
  - SEMI-AUTOMATIC: no automatic connect, automatic message processing; User code needs to invoke *Spark.connect*, if required.
  - MANUAL: no automatic connect, no message processing; User code needs to invoke *Spark.connect* and *Spark.process*, as appropriate.

Software - Code Layout

```
/*
 Fade

 This example shows how to fade an LED on pin 9
 using the analogWrite() function.

 This example code is in the public domain.
 */

int led = 9;           // the pin that the LED is attached to
int brightness = 0;    // how bright the LED is
int fadeAmount = 5;    // how many points to fade the LED by

// the setup routine runs once when you press reset:
void setup()  {
  // declare pin 9 to be an output:
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop()  {
  // set the brightness of pin 9:
  analogWrite(led, brightness);

  // change the brightness for next time through the loop:
  brightness = brightness + fadeAmount;

  // reverse the direction of the fading at the ends of the fade:
  if (brightness == 0 || brightness == 255) {
    fadeAmount = -fadeAmount ;
  }
  // wait for 30 milliseconds to see the dimming effect
  delay(30);
}
```

Variables

Initial Setup

Forever Loop

Software - Dev Environment

- Particle Build (Web IDE): https://build.particle.io/build
  - Create new code
  - Flash the code onto the device
  - Create/use libraries
- Particle Dev (https://www.particle.io/dev):
  - Desktop application, needs Internet
  - Compiles code in the cloud
  - Work in progress

Software - CLI

- Particle CLI (http://docs.particle.io/core/cli/)
  - Command line interface
  - Uses Node.js
  - Can be used to provision, claim, connect, update, compile and flash the Particle Core

Software - Cloud SDK

- ParticleJS Javascript SDK
  - Can be used to interact with the Core
  - Can be used to provision, claim, connect, update, compile and flash the Core
- iOS Cloud SDK
  - Enables iOS apps to interact with the Core
  - Can be used to provision, claim, connect, update, compile and flash the Core

Project

Part 1:
Adding sensors/actuators to the Spark Core

Part 2:
Connecting and controlling your Core via the Cloud

Let us have some fun!

Project (part 1) Steps

1. Setup
   a. Connect (USB)
   b. Claim

2. Flash the code
   a. Control an LED
   b. Serial.println
   c. Pushbutton

3. Core inputs/outputs
   a. Temperature
   b. Photo resistors
   c. Push-button
   d. LED
   e. Piezo buzzer

4. Interrupts
   a. Connect an interrupt to the push-button

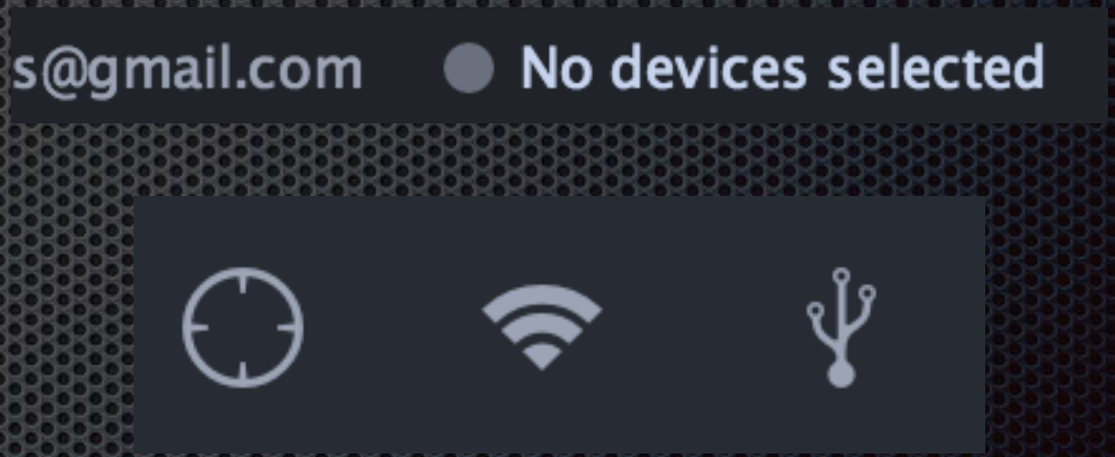
5. Event Notifications
   a. Spark.publish
   b. Spark.subscribe

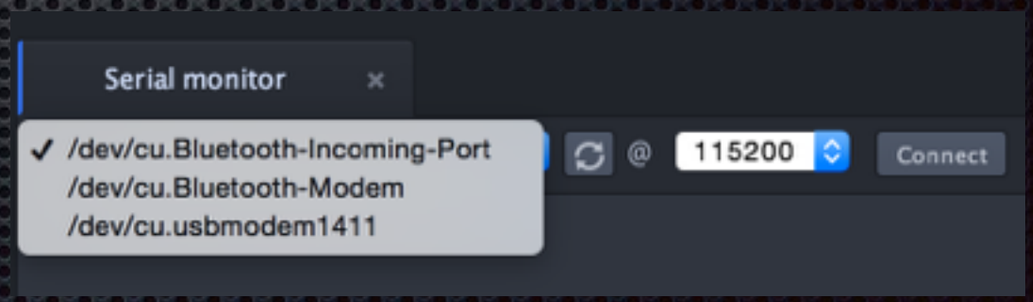
1. Setup

1. Listening Mode
    a. Press and hold the “MODE” button => RGB LED starts blinking blue
2. Connect to your computer via USB, then follow the steps below:
    1. Mac OSX (type the commands below in a terminal)
        1. `'npm install -g spark-cli'`
        2. Ensure the RGB LED is blinking blue
        3. `'spark setup'`
        4. Follow the rest of the prompts (Particle account credentials needed)
    2. Windows
        a. Download/Unzip Particle drivers from here: http://bit.ly/1Egg1DA
        b. Device Manager -> Other Devices -> double click on your Spark device
        c. Update driver -> browse for driver software on your computer -> Navigate to the dowloaded folder (Note that right now, the drivers are in a Spark folder and are named spark_core)
        d. Type the commands below on the command prompt
            i. `'npm install -g spark-cli'`
            ii. Ensure you see [serialport] Success
            iii. `‘spark setup’`
            iv. Follow the rest of the prompts
3. BE PATIENT!!

Flashing code to the Core (part 1)

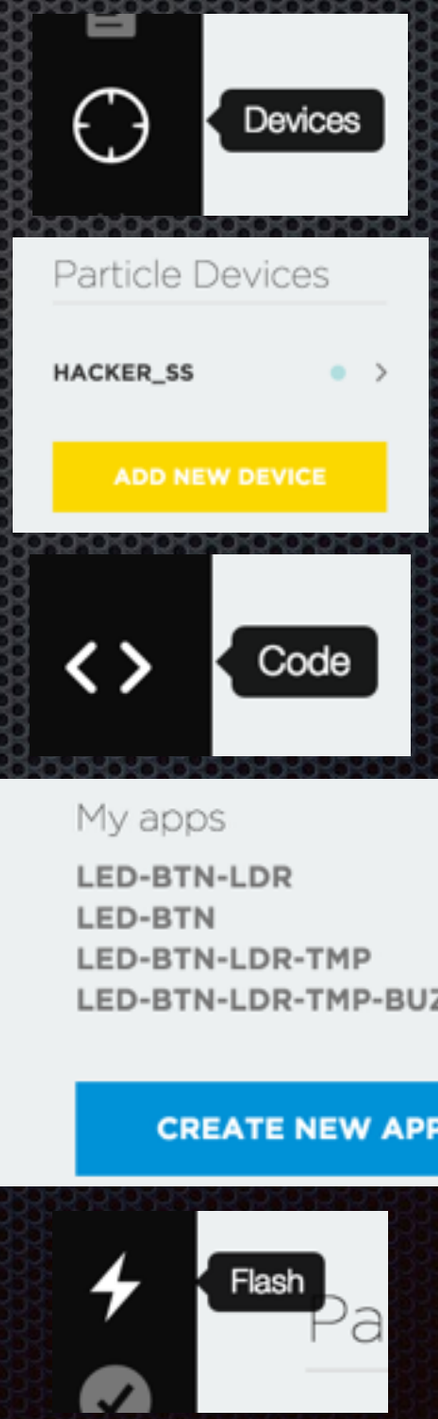
1. Run Spark Dev (installation instructions provided earlier)
    a. Connect to your device (click on “No devices selected”)
    b. Click the USB symbol in the top menu to show the Serial Monitor
        i. Connect to the Core by selecting the USB port from the dropdown and choosing a baud-rate in the other dropdown





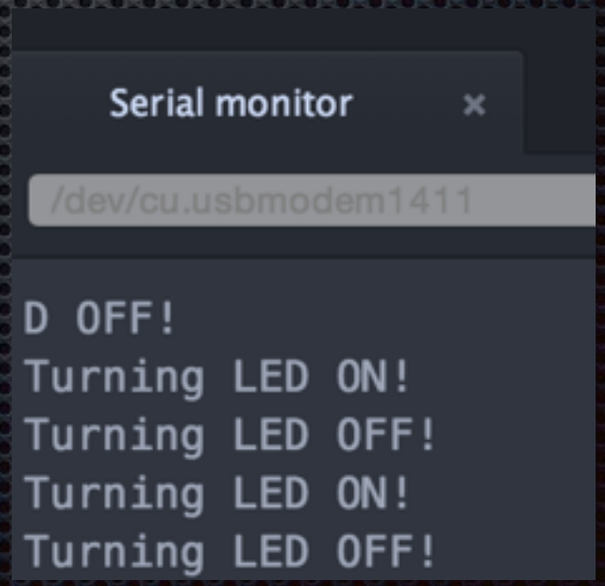
Flashing code to the Core (part 2)

1. Open Spark Build in your browser (https://build.spark.io/build/)
    a. Click on the "Devices" icon on the left
    b. Make sure your Spark Core is listed there (it would if the connecting-claiming was successful in setup)
    c. Click on the "Code" icon on the left
    d. Click on "Create New App" button
    e. Copy the content of the '*LED.INO*' file into the code area on the right side of the browser window
    f. Click on "Flash" icon on the left



Flashing code to the Core (part 3)

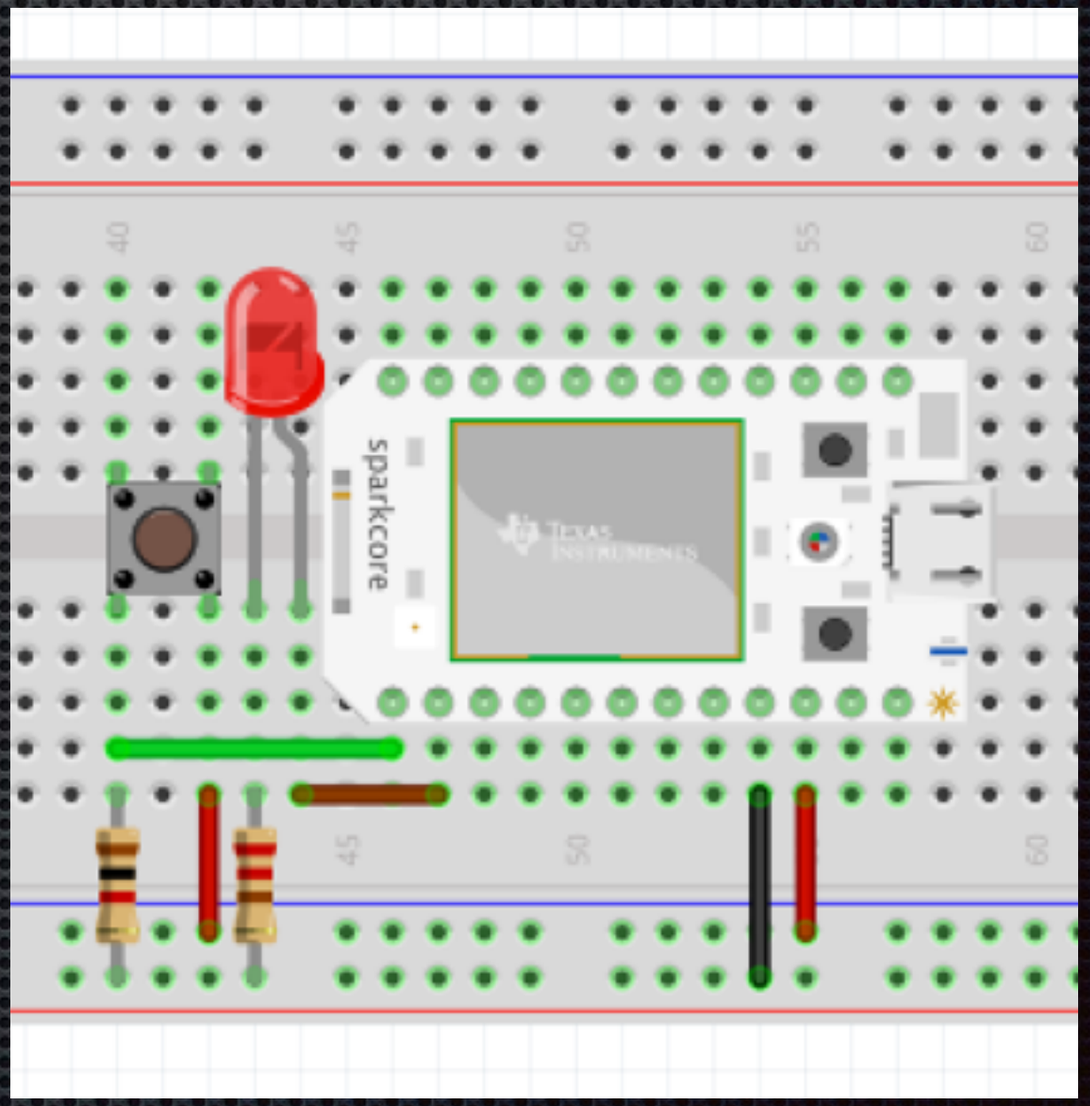
1. Make sure the blue LED on the Spark Core board turns on and off
2. Make sure the Serial.println statements print out on the Spark Dev serial monitor



Challenge #1

GOAL: Push-button control of LED

a. This challenge requires use of an LED and a push-button
b. The LED is to blink once every second ('Hello World' for circuits)
c. LED stays OFF for as long as the push-button is kept depressed
d. Log the LED state changes in Spark Dev Serial Monitor



STEPS:

a. Build the circuit on the right on your breadboard
b. Using Spark Build (Web IDE) write software to meet the challenge goals.

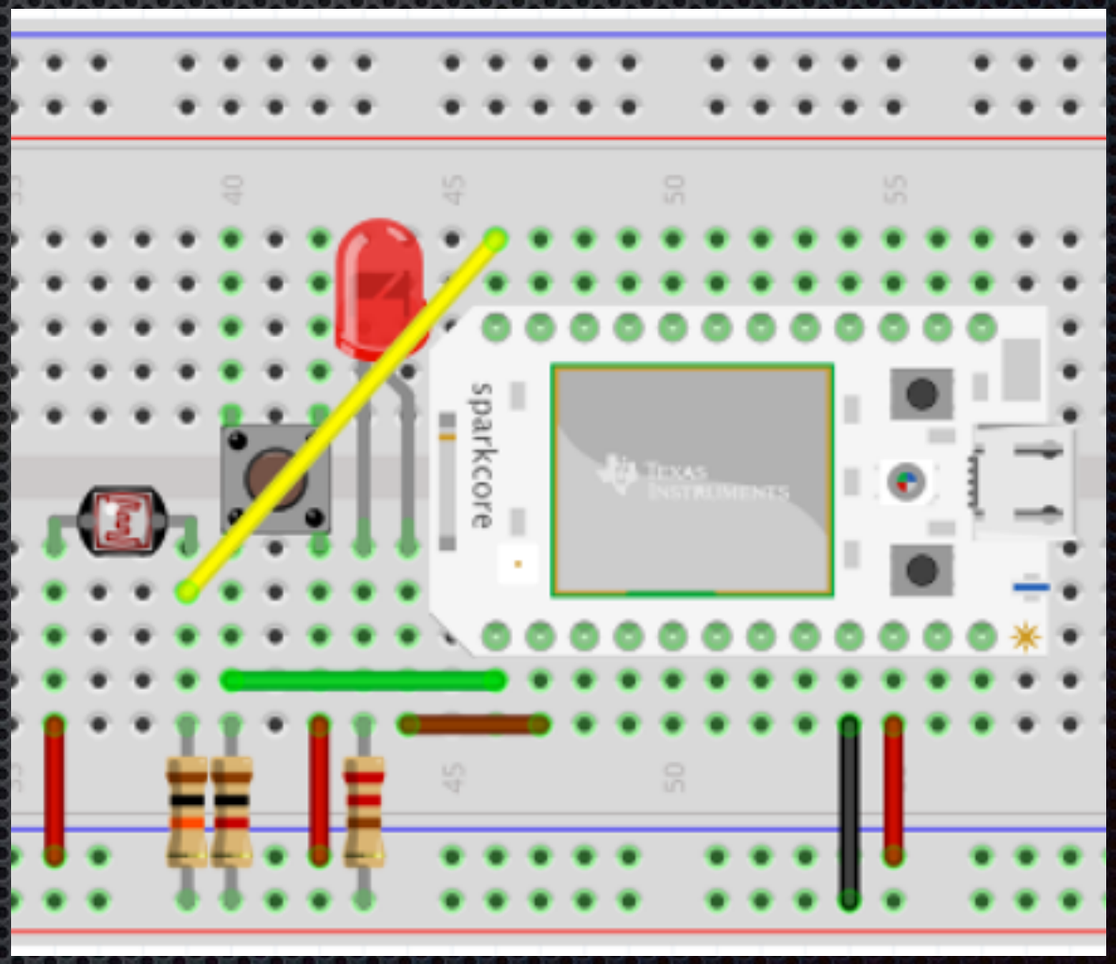
TIP:

a. Use ~220 ohm resistor for the LED and ~1 Kohm resistor for the pushbutton
b. Visit http://docs.particle.io/core/firmware/
c. Learn about *pinMode(), Serial.begin(), digitalRead(), digitalWrite(), Serial.println(), delay()*

Challenge #2

GOAL: LED brightness control using an LDR

a. Add a photo resistor to the circuit
b. Make the LED glow brighter (no blinking) when the photo resistor reports a low light condition (high resistance) and vice-versa.
c. Log the LDR sensor and LED brightness values



STEPS:

a. Enhance the previous circuit to create the circuit in the picture
b. Using Spark Build (Web IDE) write software to meet the challenge goals.

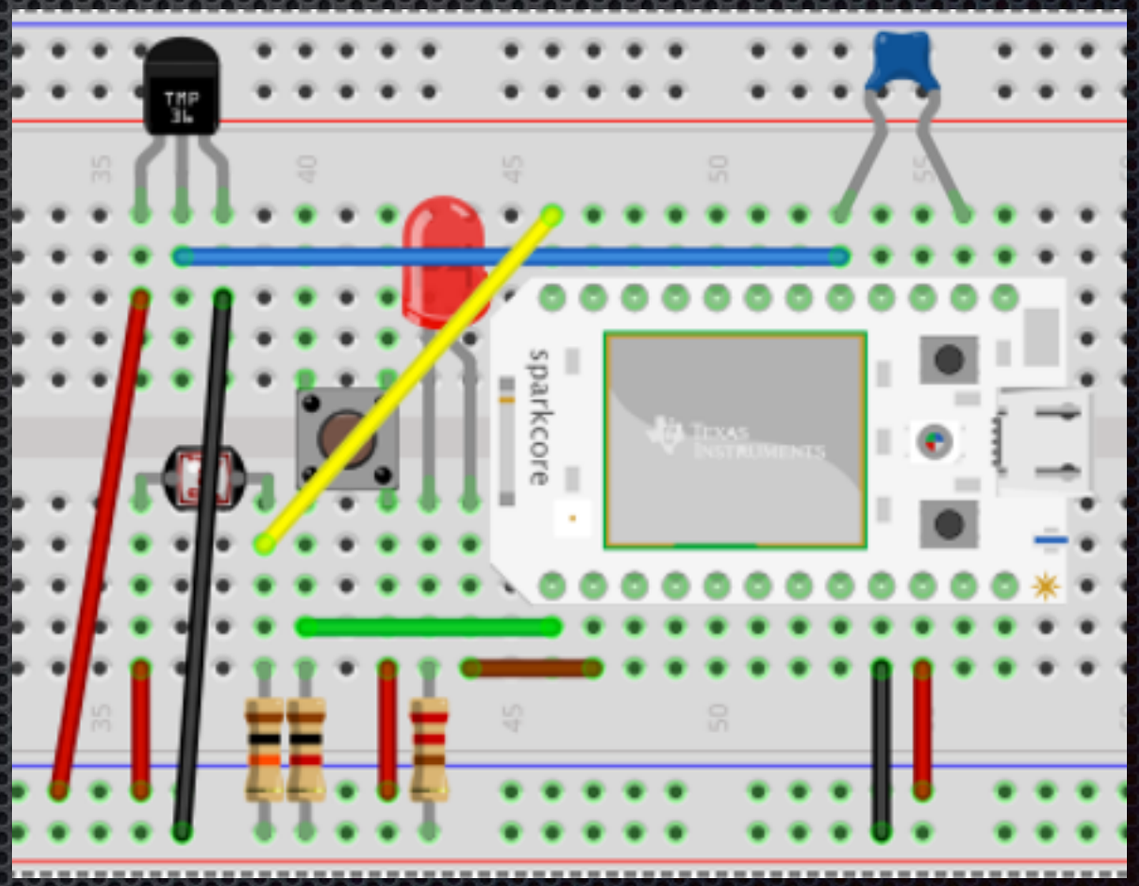
TIP:

a. Use ~10 Kohm resistor for the LDR (photo-resistor)
b. Visit http://docs.particle.io/core/firmware/
c. Learn about *analogRead(), analogWrite()*

Challenge #3

GOAL: Temperature controlled LED blink rate

a. Add a temperature sensor to the circuit
b. Make the LED blink at 1 sec at room temperature
c. Change the blink duration by 50 msec for every deg C change in temperature (blink faster for higher temperatures, slower otherwise)
d. Log the temperature value and LED blink duration.



STEPS:

a. Enhance the previous circuit to create the circuit in the picture
b. Using Spark Build (Web IDE) write a function '*void blink_led(int blink_duration)*'
c. Use it to complete the challenge goals

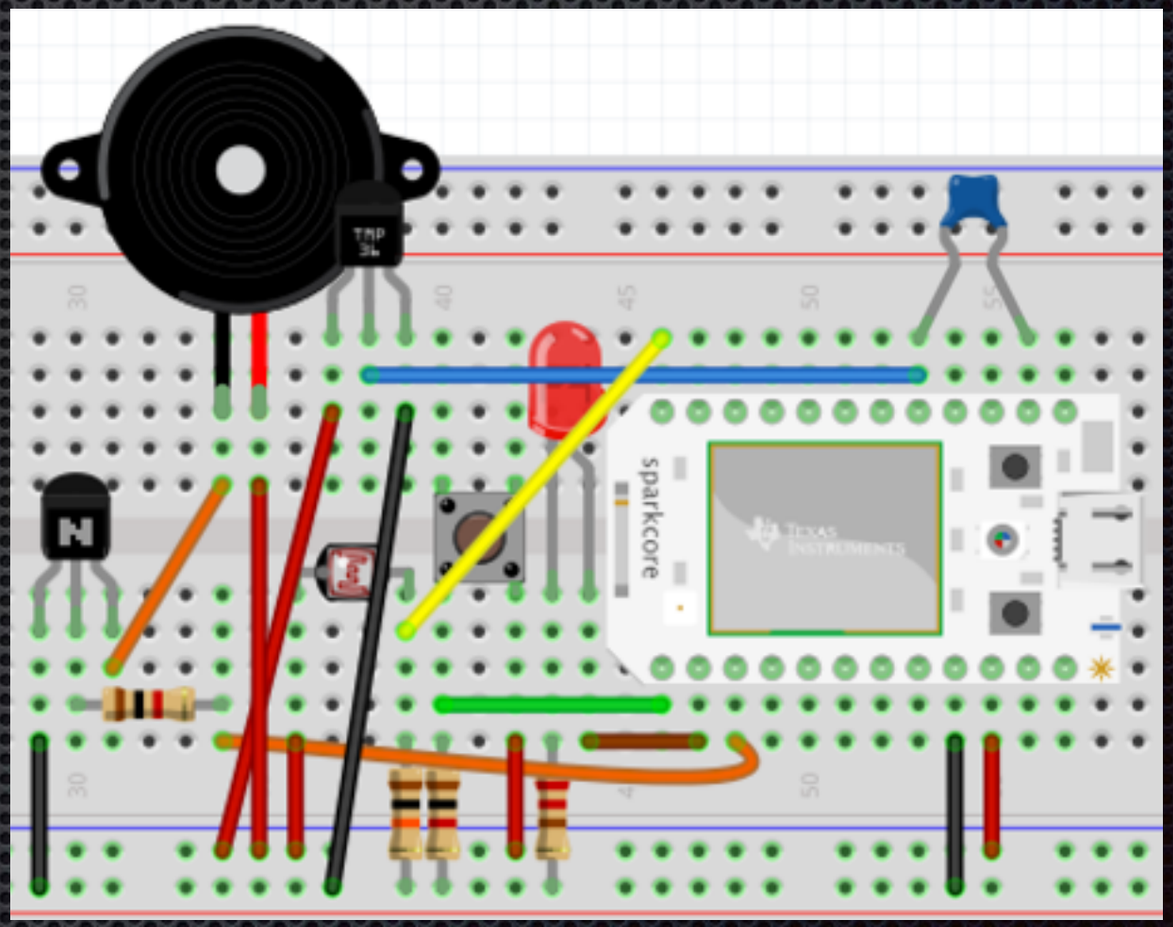
TIP:

a. Use ~100 nF capacitor for the TMP36 temperature sensor
b. Visit http://docs.particle.io/core/firmware/#spark-variable
c. Look at the code snippet on the right to convert the *analogRead()* value to temperature.
d. Use the *double* datatype for the math and *round()* function to convert back to *integer*.

Challenge #4

GOAL: Temperature controlled audio-visual alarm

a. Add a Piezo Buzzer to the circuit
b. Make the buzzer beep if the temperature goes above a threshold (eg. 78.8 deg F or 26 deg C). Blow on the sensor to increase the temperature.
c. Increase the beep frequency with temperature.



STEPS:

a. Enhance the previous circuit to create the circuit in the picture
b. Using Spark Build (Web IDE) add a function '*void beep_buzzer(int beep_duration)*'
c. Use it to complete the challenge goals

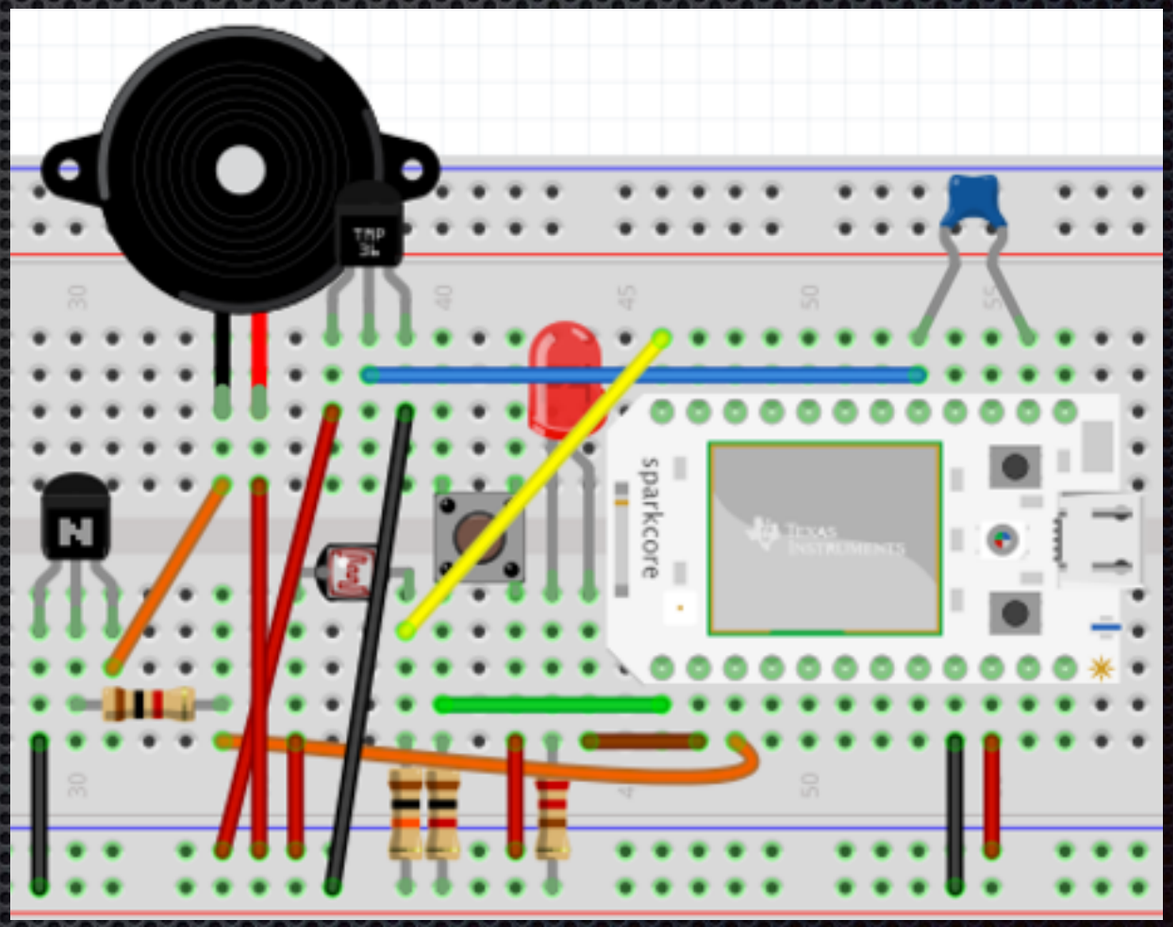
TIP:

a. Use the NPN diode with a ~1K resistor to drive the Piezo Buzzer.
b. You may want to keep the LED and the Buzzer ON constantly beyond a certain temperature.
c. The buzzer beeps are annoying, so be mindful.

Challenge #5

GOAL: Simluate fire hazard that is averted when the push-button is pressed.

a. Detect the presence of high light (choose an appropriate threshold) and high temperature (104 deg F/40 deg C) condition.
b. Leave the LED and the Buzzer ON and log a FIRE HAZARD message to the serial monitor.
c. Stop the Buzzer when the push-button is pressed.



STEPS:

a. Add the code to detect the above conditions to complete the challenge.
b. Log messages to indicate hazardous conditions and when the danger is averted.

TIP:

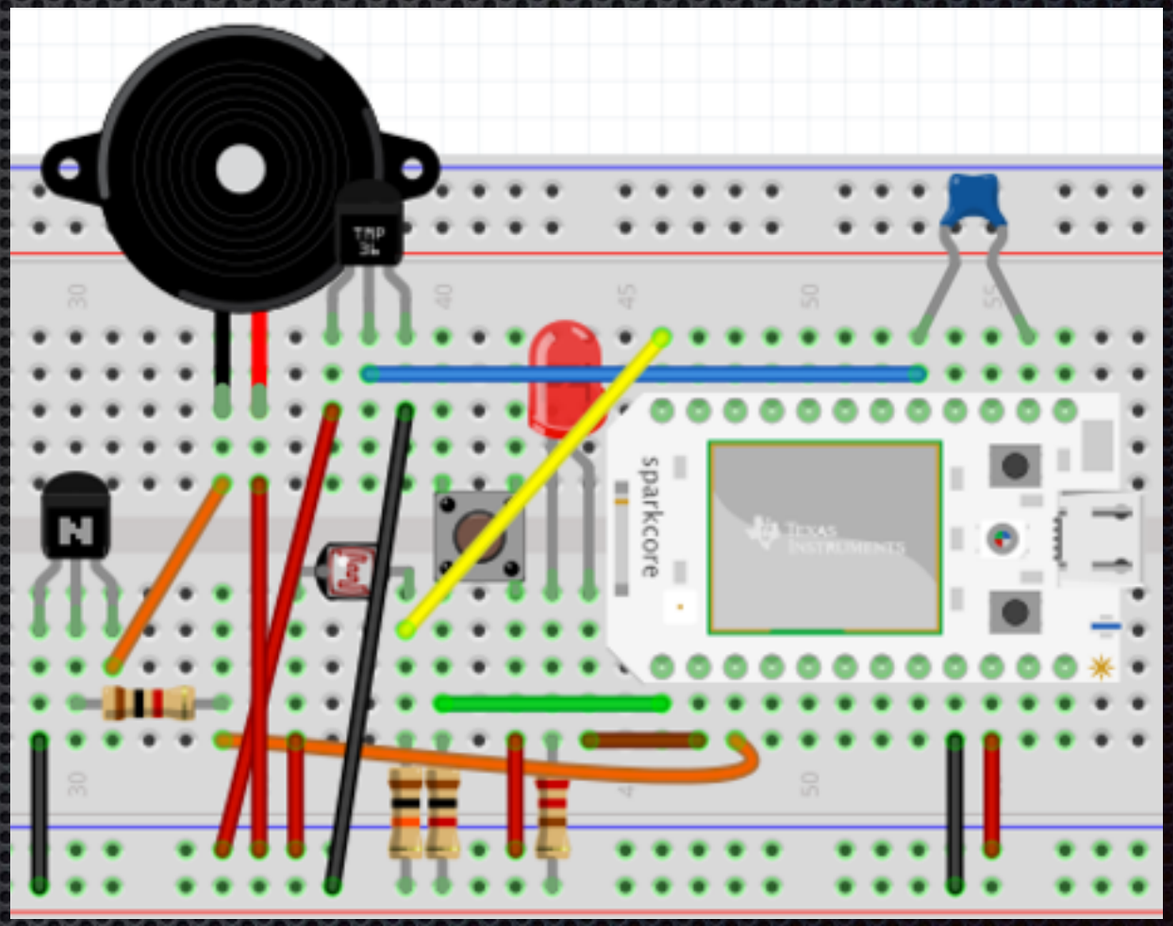
a. The buzzer beeps are annoying, so be mindful.

Challenge #6

GOAL: Trigger a software interrupt when push-button is depressed.

STEPS:

a. Use the software from the previous challenge.
b. Change the polling for push-button status to be interrupt driven.
c. Write an ISR to process the interrupt.
d. Log messages to indicate interrupt processing

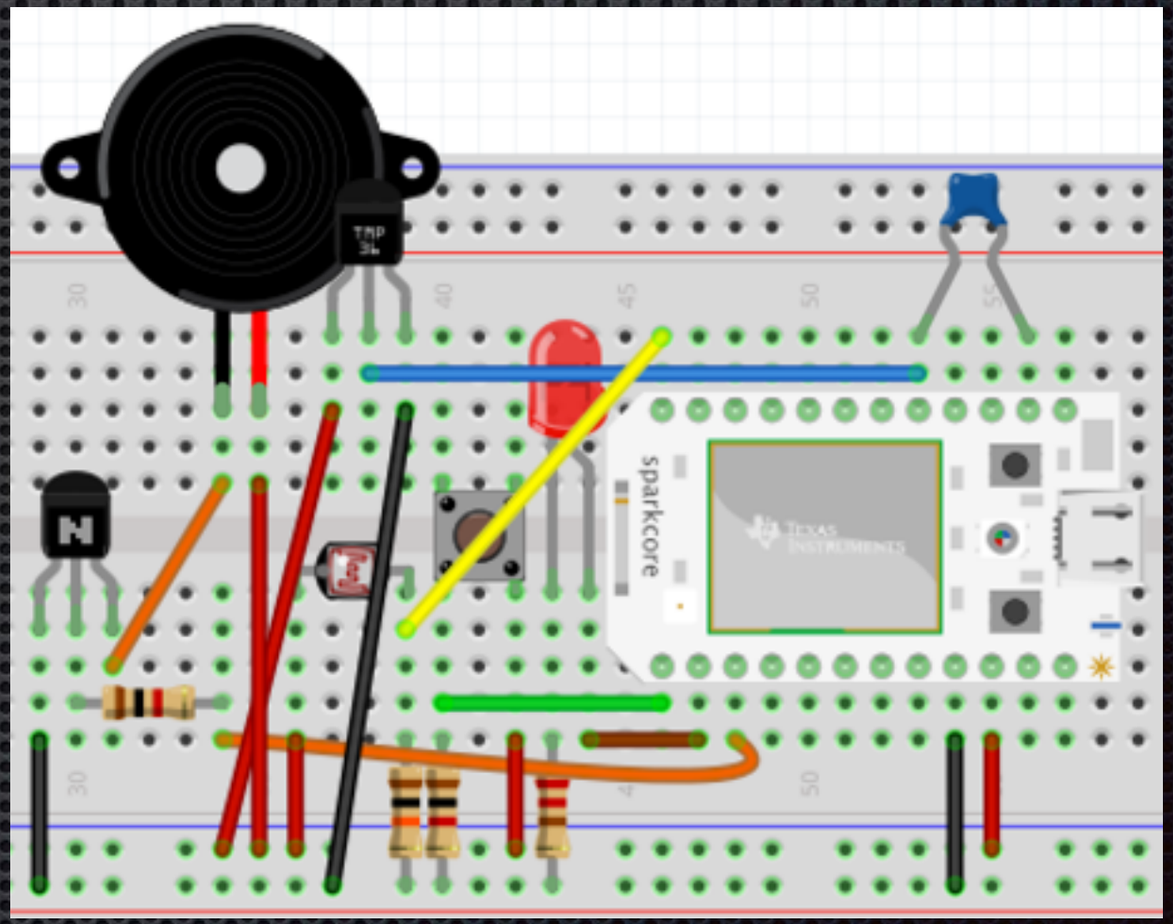


TIP:

a. Visit http://docs.particle.io/core/firmware/#other-functions-interrupts

Challenge #7

GOAL: Listen to an event from another Spark Core using *Spark.publish()* and *Spark.subscribe()*



STEPS:

a. Find a partner with his/her own Spark Core.
b. Choose a Core (yours or his/hers) to *publish* the photo resistor data.
c. Choose the other one to *publish* the temperature sensor data.
d. Write the *Spark.publish()* routines to accomplish the same.
e. Write the *Spark.subscribe()* routines to listen to other's published events.
f. Temperature sensor triggers the LED.
g. Photo resistor (LDR) event triggers the Buzzer.

TIP:

a. Visit http://docs.particle.io/core/firmware/#spark-publish

Break (10 mins)

Web Access to your Particle Core

Core - Web Interactions

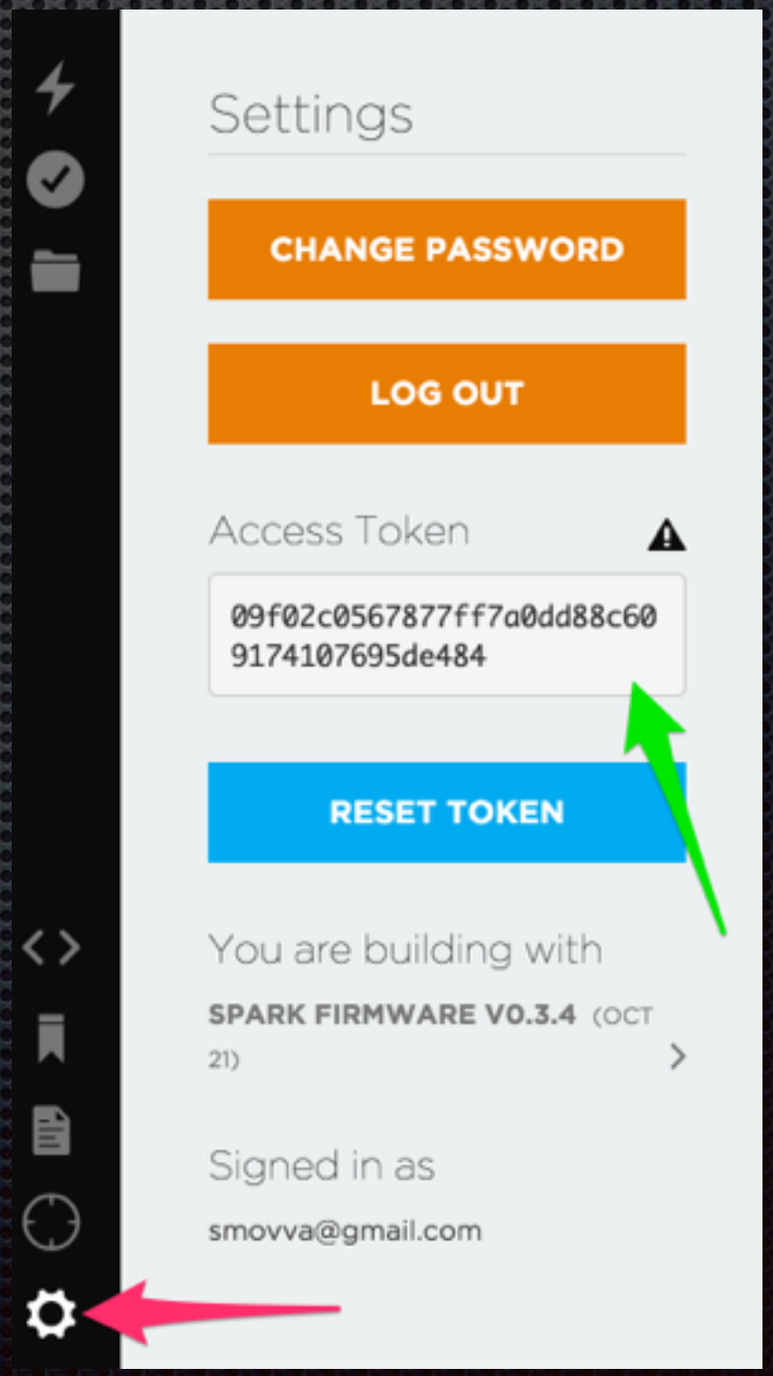
- We will use two frameworks to Interact with Spark Core
  - Spark Cloud API
  - PushingBox
- Lots more out there...

Particle Cloud API

- All requests to the device come through Spark's API server using TLS security.
- Every device has a URL, which can be used to
  - GET variables,
  - POST a function call, or
  - PUT new firmware.

REST API access setup

- Permissions for controlling and communicating with your Particle device are managed with OAuth2.
- Your access token can be found in the Particle Build web IDE on the 'Settings' page.
- Install Postman - REST Client if not already done earlier

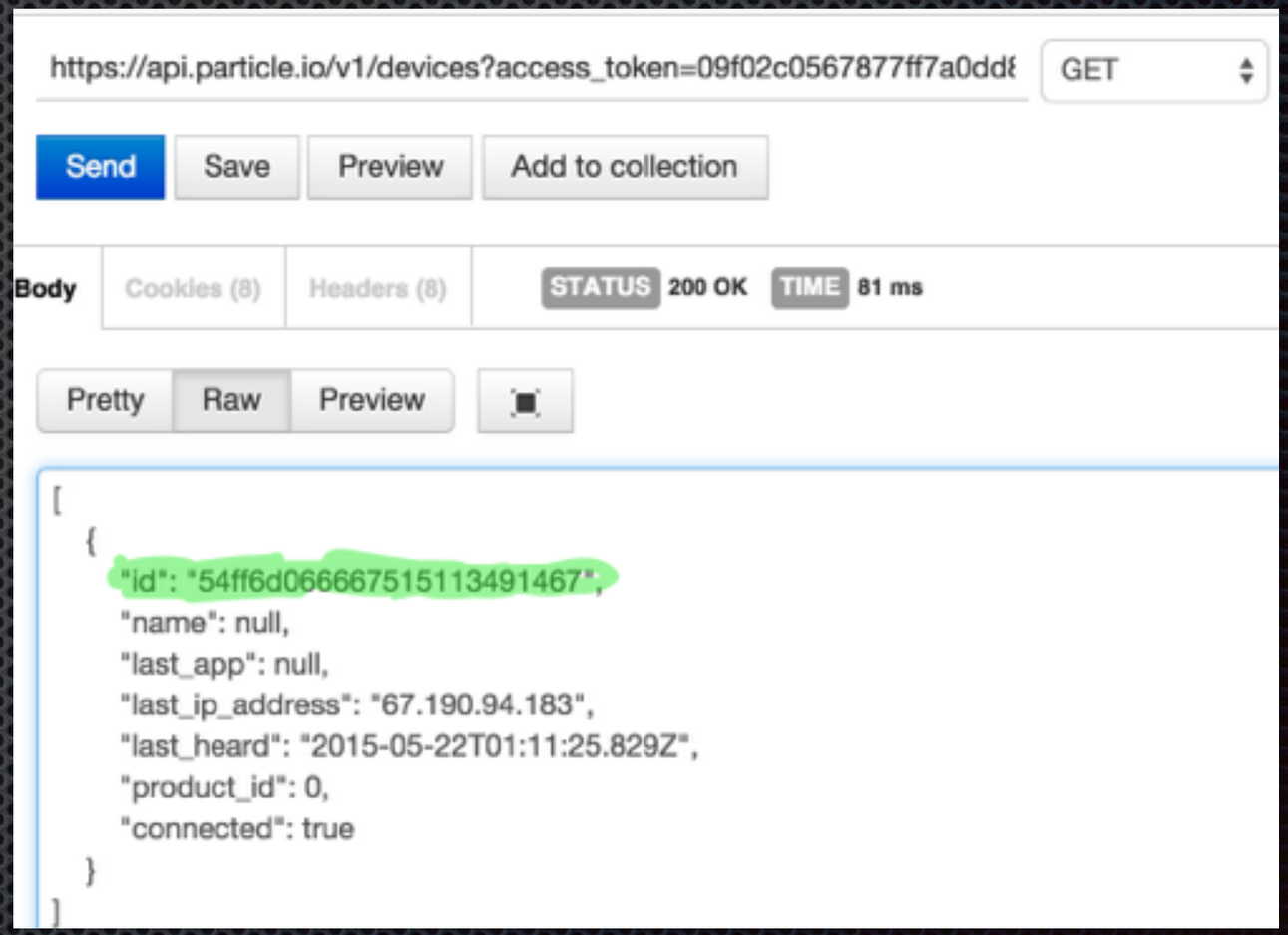


Get All Devices

- Query

  https://api.particle.io/v1/devices?access_token=XX

- Returns the ID of all the devices you have access to



Closer Look at a REST Call

- https://api.particle.io/v1/devices?access_token=09f02c0567877ff7a0dd88c609174107695de484
- Server
  - https://api.particle.io
- Version
  - /v1Current Spark support is only for v1 API
- Resource
  - /devices?
  - Gets list of all devices accessible using the access token
- Access Token
  - access_token=09f02c0567877ff7a0dd88c609174107695de484
  - Authentication token from Settings page

Expose a Variable

```
int temperature = 0;

void setup()
{
  Spark.variable("temperature", &temperature, INT);
  // ?? Why the &temperature?
}
```

- Make your variable a Spark variable using the function
  - Spark.variable(“Name in REST API”,

    address of local variable,

    type);
  - Make sure that the variable you are using is something that is changing in your code like photo sensor output or temperature
- Compile and Flash the build

Get the Device Details

- Query
  - GET https://api.particle.io/v1/devices/DEV_ID?access_token=ACC_TKN
- Output


```
{
    "id": "54ff6d0666675151134914 67",
    "name": null,
    "connected": true,
    "variables": {
        "temperature": "int32"
    },
    ...
}
```


- Notice the variable you added shows up in device details
- Also notice the connected flag
  - Turn off your core and see what the query returns
  - What does that tell us?

Get the Variable

- Query
  - GET https://api.particle.io/v1/devices/DEV_ID/temperature?access_token=ACC_TKN
- Output


```
{
    "cmd": "VarReturn",
    "name": "temperature",
    "result": 1,
    "coreInfo": {
        ...
    }
}
```


- Use the query above to GET a variable.
- Try it a couple of times to see if the value is changing
  - What happens if the core is turned of when the query is made?
  - What happens if you try to get a variable that is not a spark variable?

Expose a Function

```
int toggleLight(String args)
{
  ledState = ledState==TRUE?FALSE:TRUE;
  digitalWrite(led_p, ledState);
  return ledState;
}

void setup() {
{
   Spark.function("toggleLight", toggleLight);
}
```

- Create a function to toggle the LED Status
- Make it a spark function
  - Spark.function("toggleLight", toggleLight);
  - Let the function return the current status of the LED
- Compile and Flash the build

Call the Function

- Query
  - POST https://api.particle.io/v1/devices/DEV_ID/toggleLights?access_token=ACC_TKN
- Output


```
{
    "id": "54ff6d06666751511349146 7",
    "last_app": "",
    "connected": true,
    "return_value": 1
}
```


- Use the query to get the device details and see if function is listed
- Call the function using the post API above
  - Notice the return value and make sure that it is changing

Using API in a website

- Rest API can be used from Web apps using javascript
  - Never put access tokens and device IDs where other people can read it.
  - The current code does that for simplicity
- Create a simple HTML file
  - Java script can be used to make a REST API Call
- Get code sample from <<TODO>>

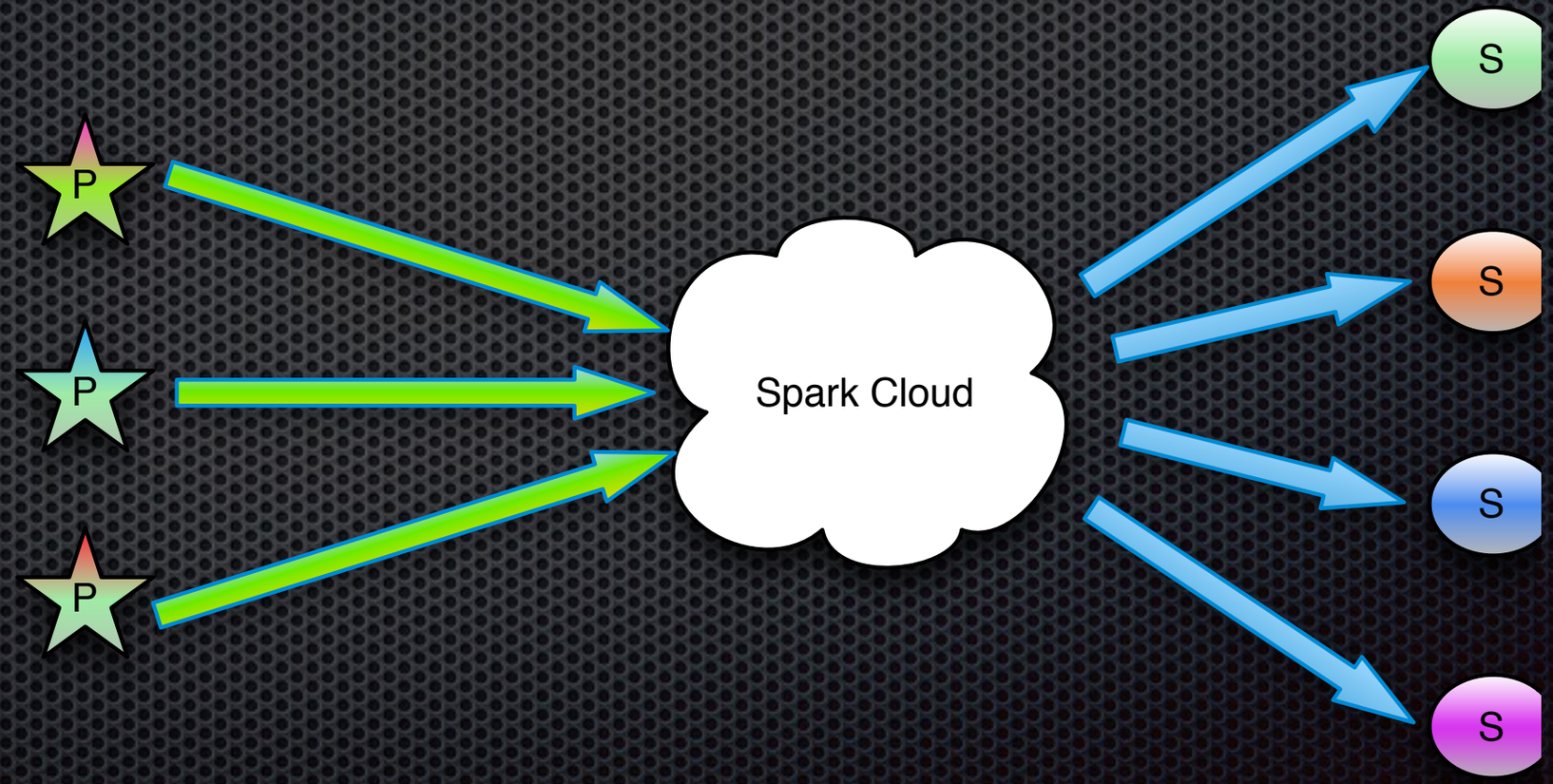
Using API in a website(2)

```
$.ajax({
        type: "GET",
        dataType: "json",
        url: “https://api.particle.io/v1/devices/DEVID/lights?access_token=ACC_TKN",
        success: function(data){
          alert(data.result);
        }
    });
```

- Example uses jQuery to send GET and POST requests
- Try to change the text button to say
  - Turn Off -> if lights are currently ON
  - Turn On -> if lights are currently off

Publish Subscribe

- Publish/Subscribe Model for realtime messaging and signaling
- Multiple Subscribers to each event and
- Multiple Publishers of same event



Publishing events - FW

```
if(ledState)
{
  Spark.publish("myLightStatus", "ON");
 }
 else
 {
  Spark.publish("myLightStatus", "OFF");
 }
```

- Spark allows for publishing events
  - Spark.publish(“Light-Status-Changed“); ← Basic event
  - Spark.publish("Light-Status", "ON"); ← Event with details
  - Spark.publish("lake-depth/1", "28", 21600); ← TTL for the event
  - Spark.publish("front-door-unlocked", NULL, 60, PRIVATE); ← Make event private

Listen to Events

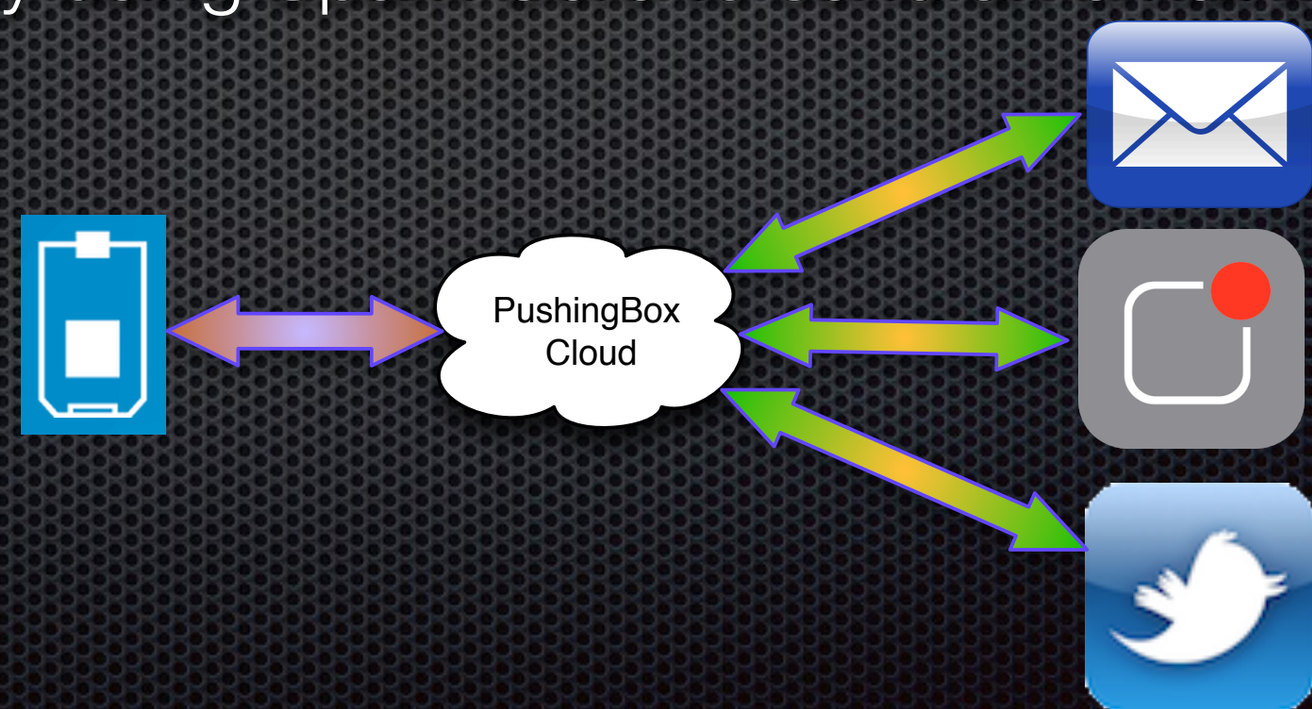
```
var eventSource = new EventSource("https://api.spark.io/v1/devices/" + deviceID + "/events/?access_token=" + accessToken);

eventSource.addEventListener('myLightStatus', function(e) {
  var parsedData = JSON.parse(e.data);
  var lightSpan   = document.getElementById("lightStatus");
  lightSpan.innerHTML = "Light is currently " + parsedData.data;
}, false);
});
```

- Listen to events in JS code
  - Change HTML based on event
- Try adding more events to see if this works

Pushing Box

- PushingBox is a cloud that can send notifications based on API calls.
- A single request to pushing box can can trigger several notifications like a Push, a Tweet, an Email etc.
- Let us try using Spark Core to send an email if temperate is up



Pushing Box - Initial Setup

- Sign into pushing box using your Google account
- Create a service under my services tab
  - for the current exercises we will choose email but it can be anything
- Create a Scenario
  - Example: TooHot
  - Add an Action for the scenario - to email when the condition is met

Core + PushingBox

- Call the function when some condition is met
  - Say when temperature is above a threshold
- See if you got the email
- You can add other notifications and see if that works for you as well

Add details to notifications

Spark Code

Add the following code to where the pushing box request is sent

```
client.print("&temperature=");
client.print(temperature);
```

PushingBox Changes

Edit your message to include the following

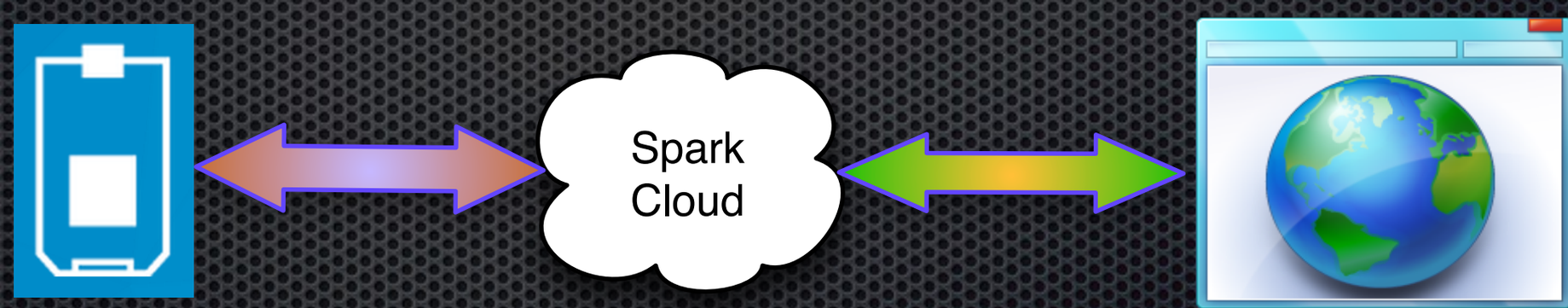
Temperature in the room is **$temperature$**

- This would trigger an email with data from the spark
  - Twitter/Notification support can be added by adding service on pushing box

Conclusion

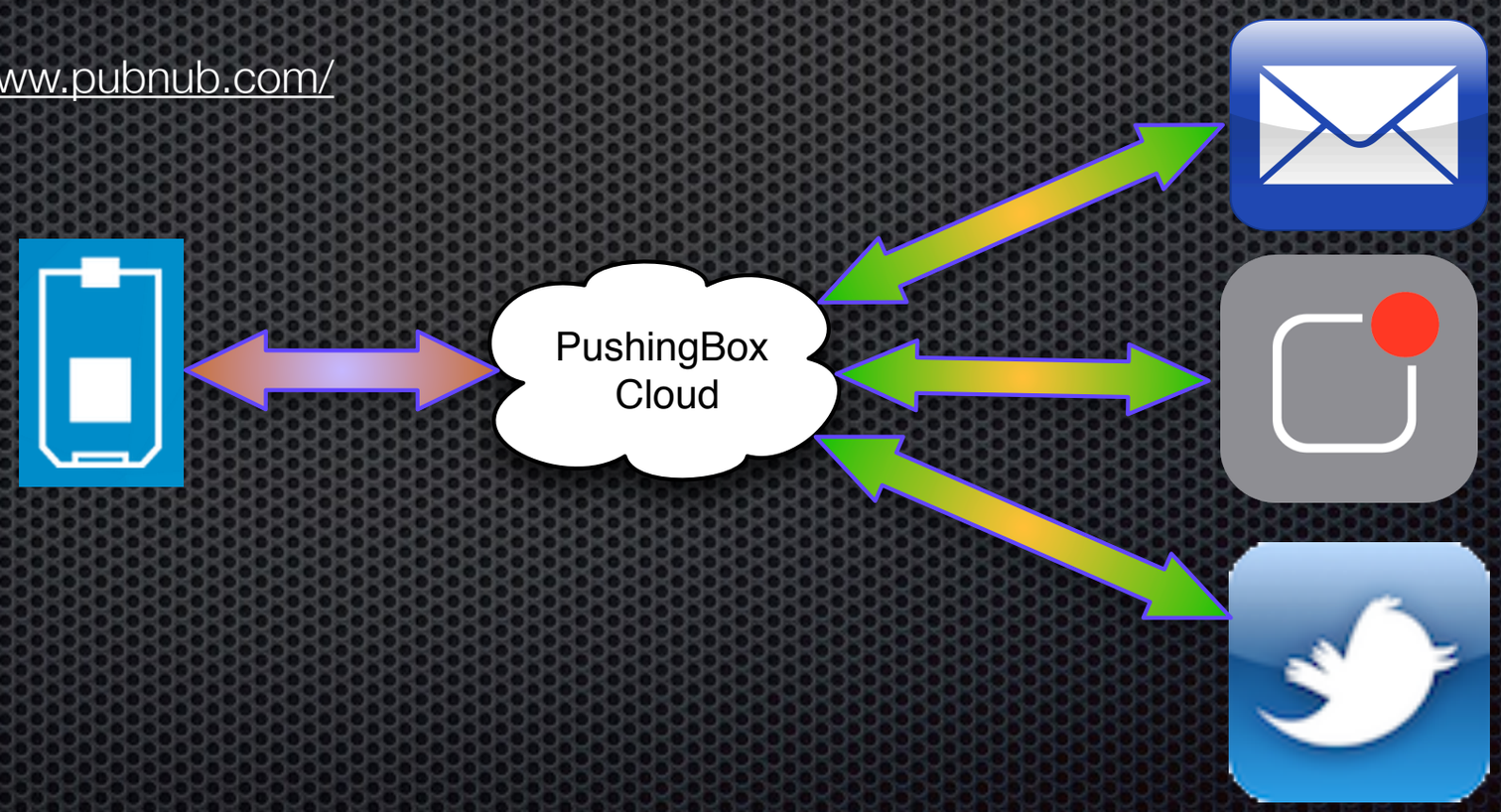
Three ways to connect(1)

- Use Spark REST API to build your apps
  - Can get data/call functions/upload FW etc.
- Integration with other services will happen on web server



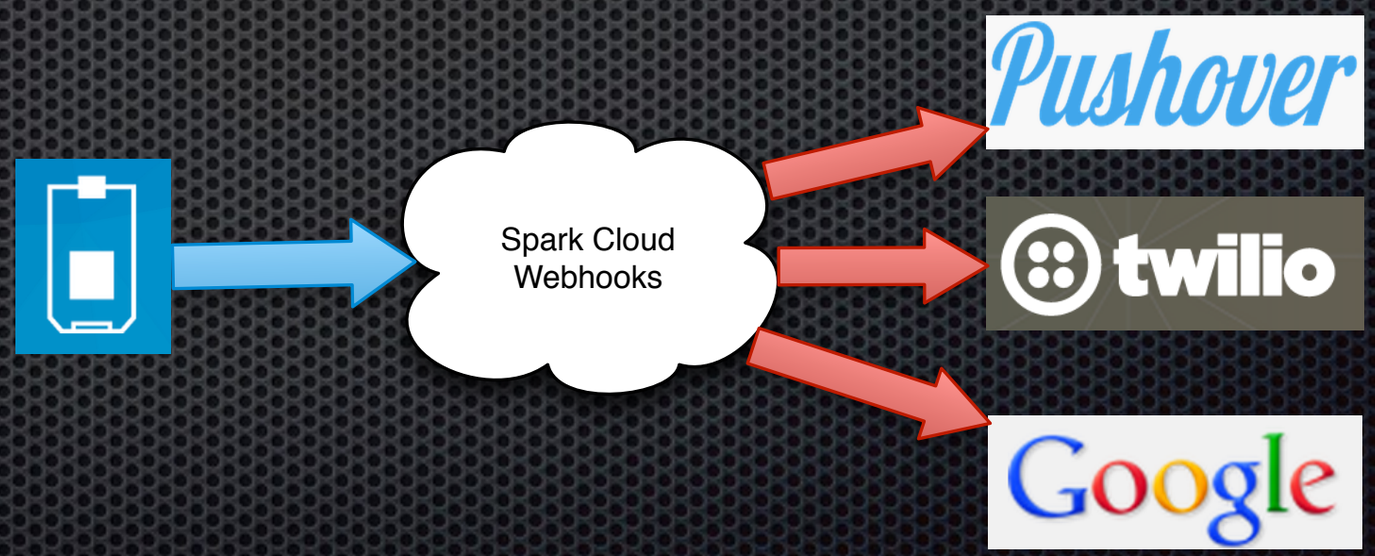
Three ways to connect(2)

- Spark Directly talks to the service provider
  - using service Provider API
- Some examples include
  - https://thingspeak.com/
  - https://temboo.com/
  - https://www.pubnub.com/

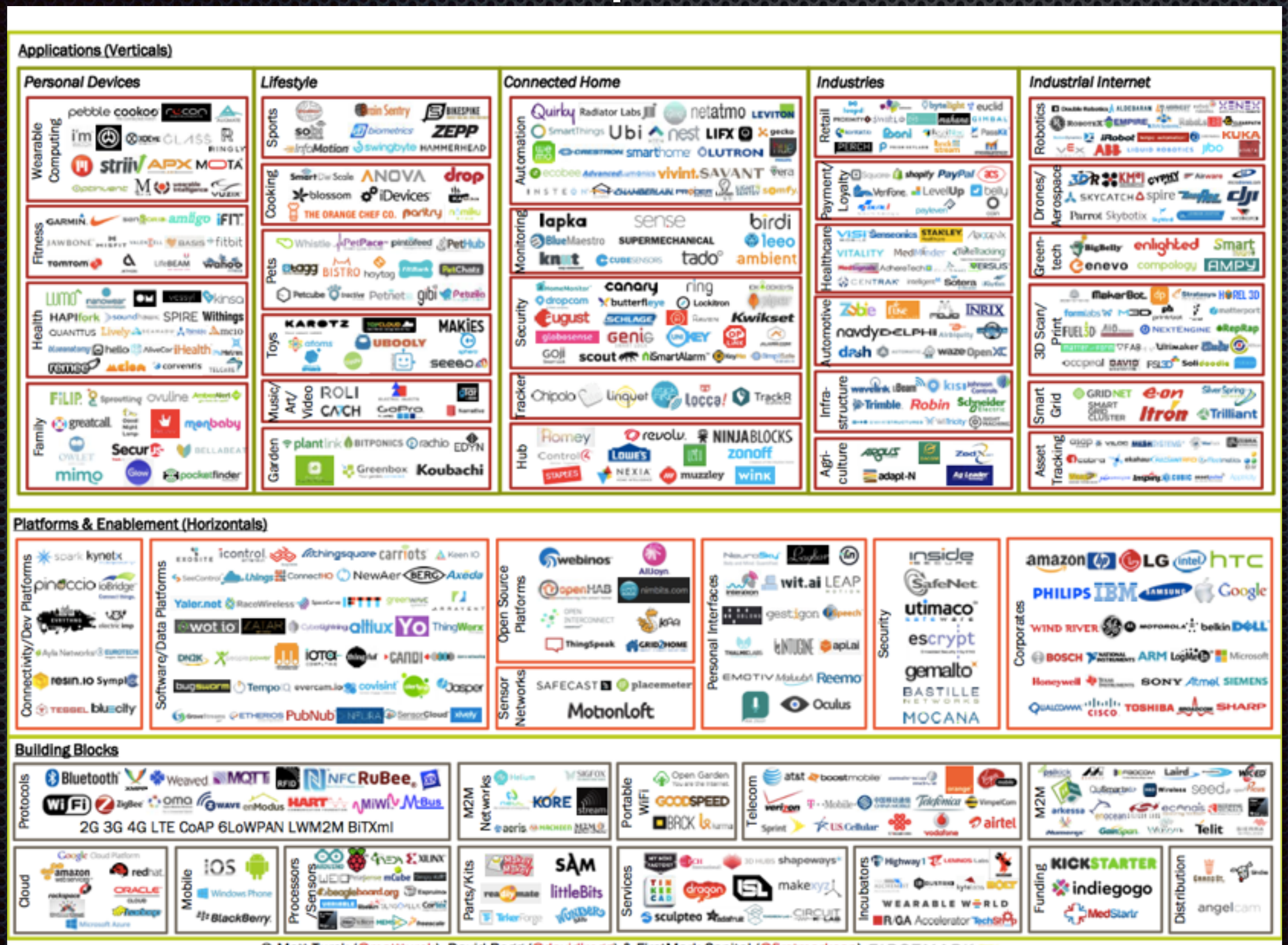


Three ways to connect(3)

- Spark Core interactions using web hooks
  - Add a web hook on spark cloud
  - Publish an event from the core
  - Cloud makes a call to API from other service providers



IOT Landscape



Thank You!