IEEE SMP 2025 - D03

BACON

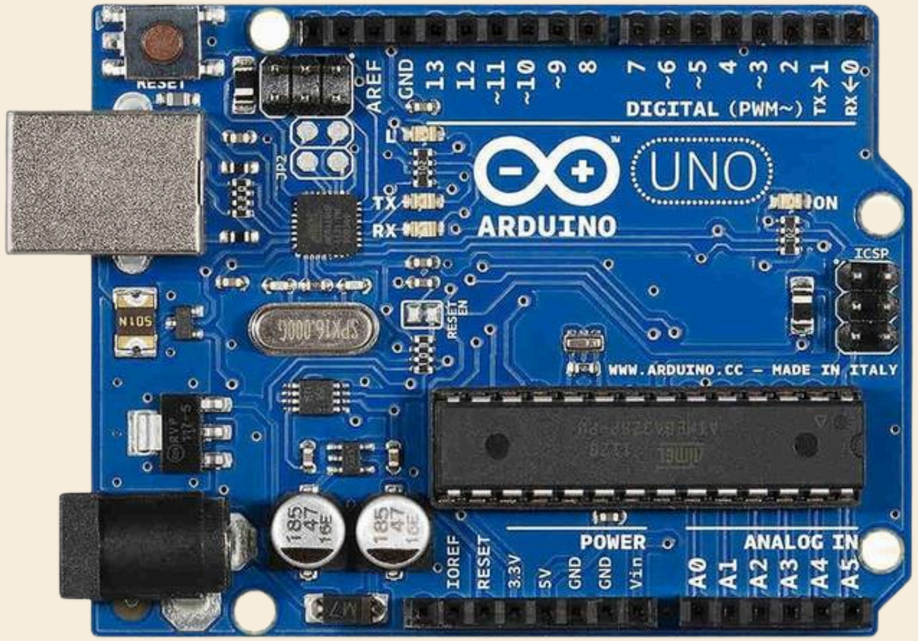
Building embedded Applications with Cloud, MQTT, and Node-RED

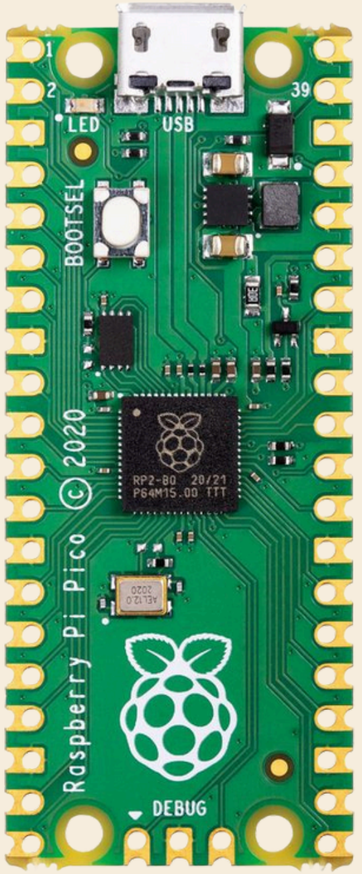
Week 2

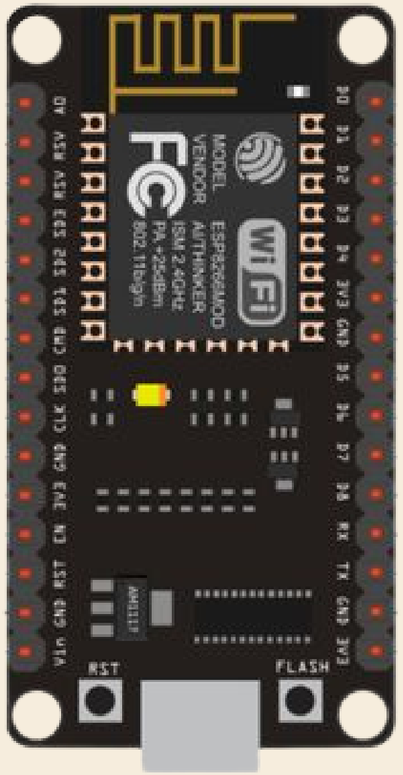
MICROCONTOLLERS

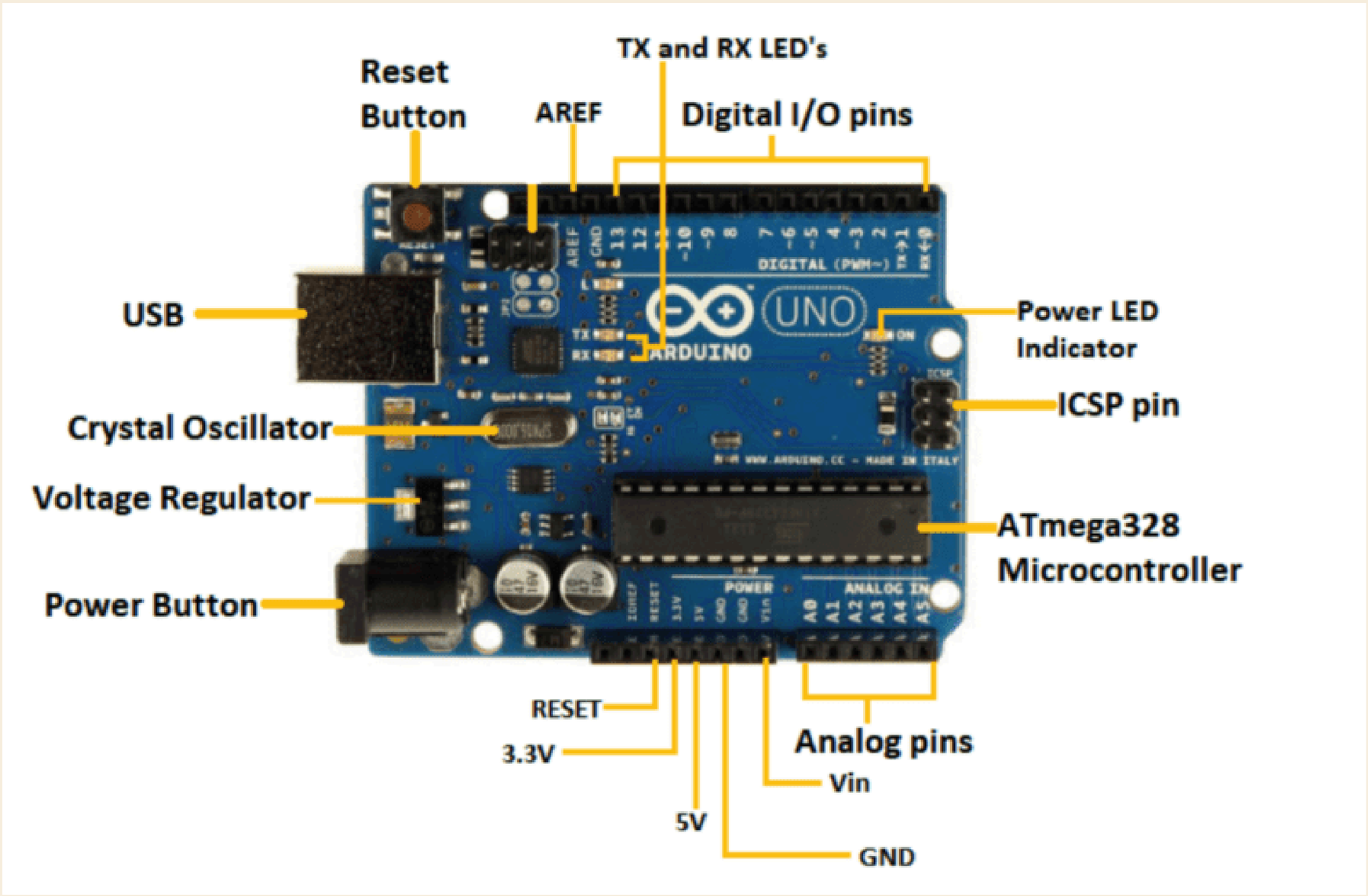
popular microcontrollers:

- Arduino UNO – Beginner-friendly, based on ATmega328P.
- Raspberry Pi Pico – Dual-core ARM Cortex M0+, supports C/C++ and MicroPython.
- ESP8266 / ESP32 – Built-in Wi-Fi/Bluetooth, ideal for IoT.

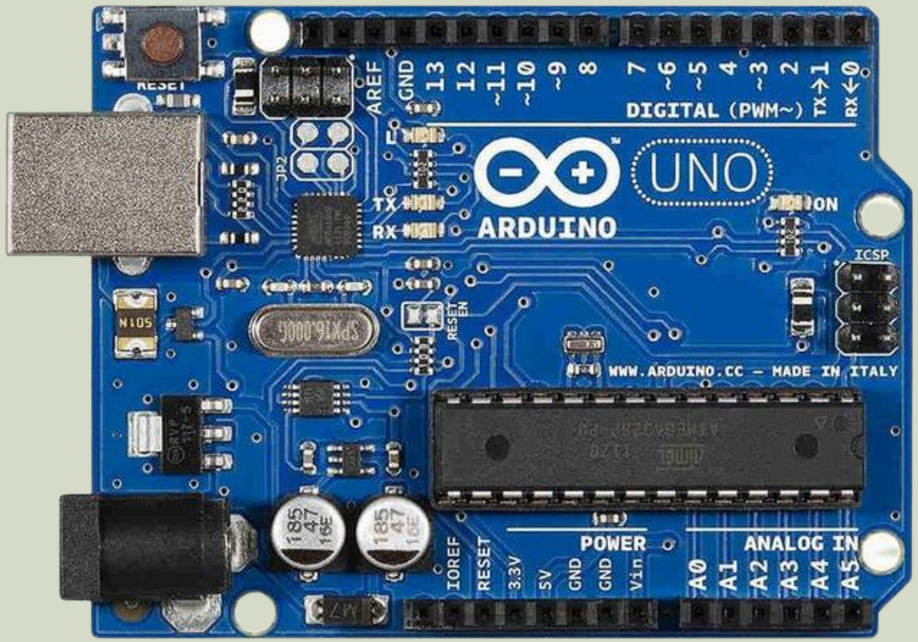

ARDUINO
UNO
DIGITAL (PWM~)
POWER
ANALOG IN
WWW.ARDUINO.CC – MADE IN ITALY


Raspberry Pi Pico
© 2020
BOOTSEL
LED
USB
DEBUG


RST
FLASH

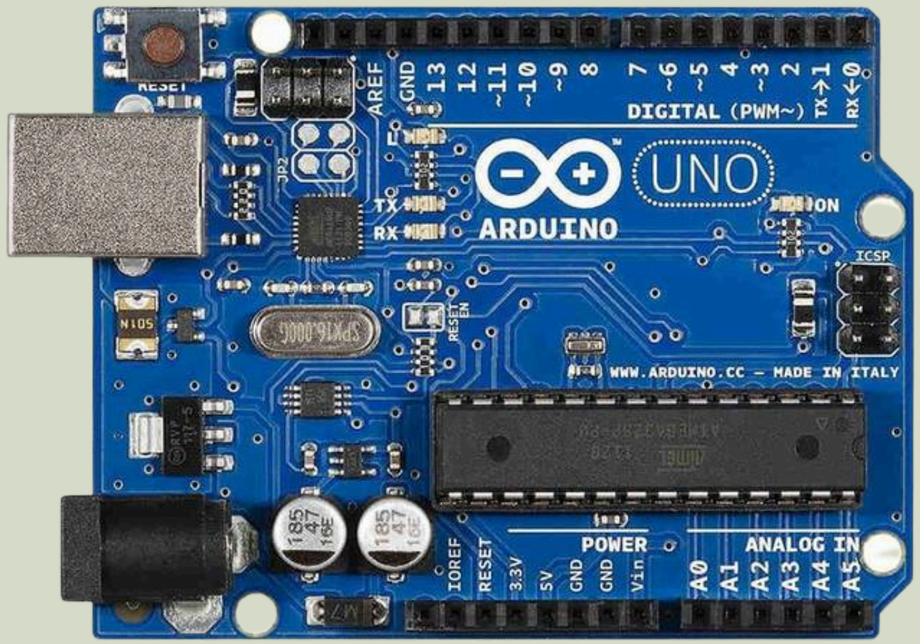

Reset Button
AREF
TX and RX LED's
Digital I/O pins
USB
Power LED Indicator
ICSP pin
Crystal Oscillator
Voltage Regulator
ATmega328 Microcontroller
Power Button
RESET
3.3V
5V
GND
Analog pins
Vin
UNO
ARDUINO
DIGITAL (PWM~)
POWER
ANALOG IN

PARTS OF AN UNO



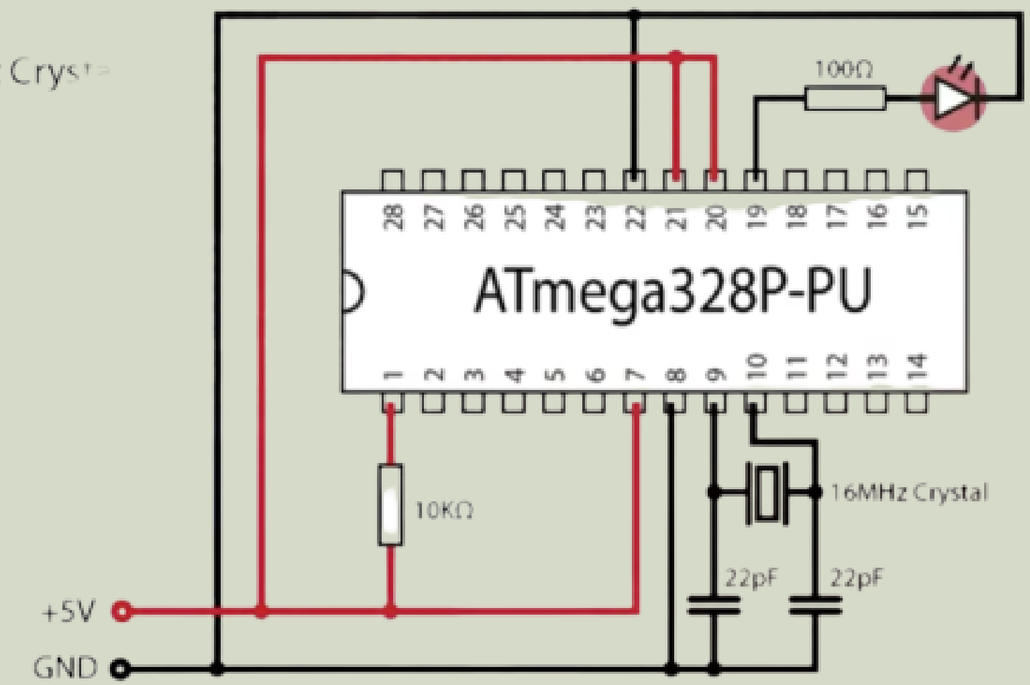
- Microcontroller – ATmega328P (main processing chip)
- Crystal Oscillator – 16 MHz clock source
- USB Interface – ATmega16U2 handles USB to serial conversion
- Voltage Regulator – Converts 7–12V to 5V/3.3V
- Reset Button – Manually restarts the MCU
- Power Pins – 3.3V, 5V, GND
- I/O Pins – 14 digital, 6 analog inputs
- AREF- Reference voltage for the analog inputs.
- The ICSP (In-Circuit Serial Programming) - Used for programming the microcontroller directly without the need for a bootloader.

ATMEGA328P



- Digital I/O Pins: 14 (of which 6 provide PWM output)
- 20 MHz max supported clock speed, runs typically at 16 MHz
- Operates at 5V logic level
- DC Current per I/O Pin: 40 mA
- Flash Memory: 32 KB
- SRAM: 2 KB
- EEPROM: 1 KB
- Before, the only way to load program code into their memory was with a programmer – a special hardware tool built for the purpose. Now, the microcontroller receives new program data via some communication method – be that USB, UART, etc – and writes that new program data onto the chip via a bootloader.

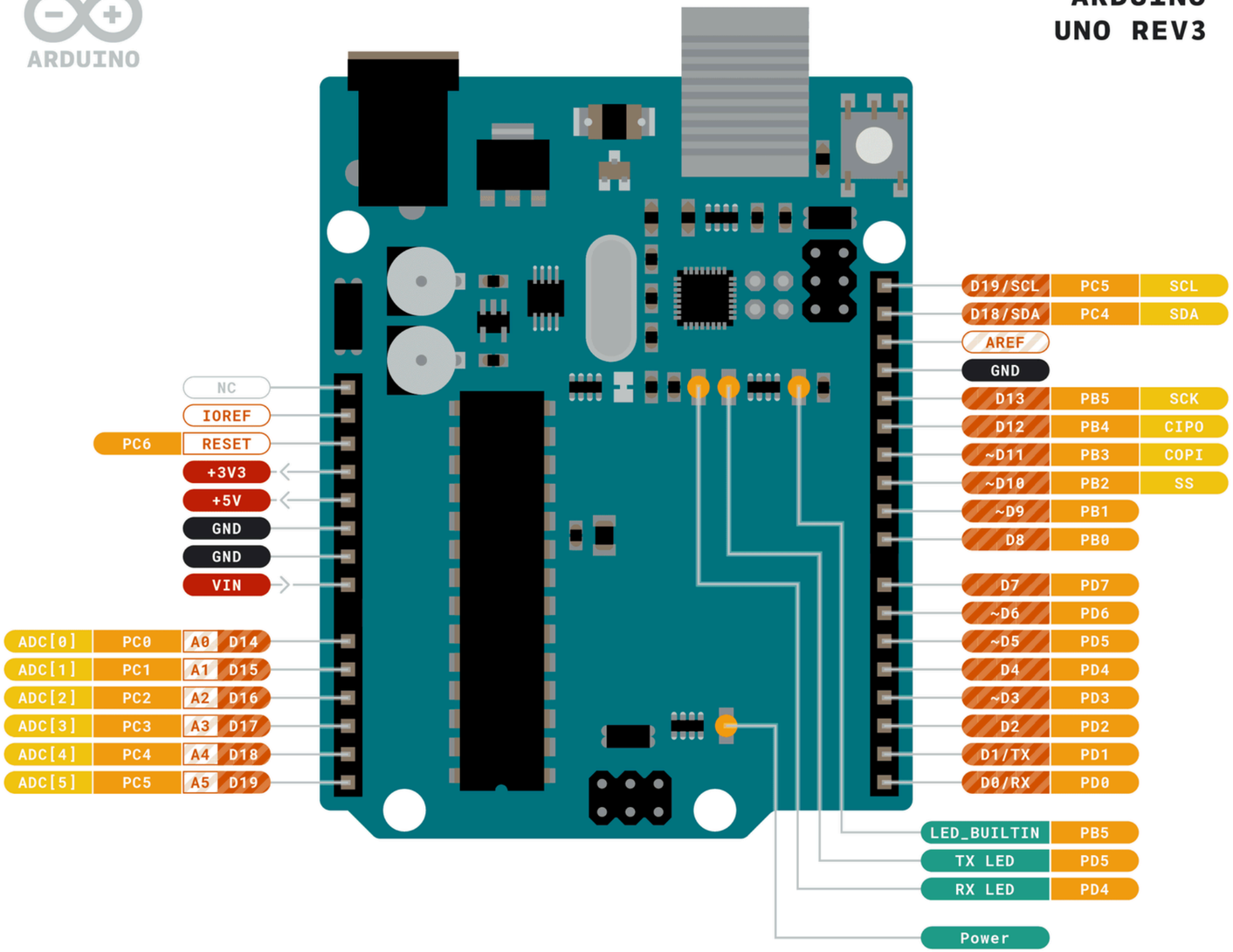
CLOCK GENERATION



- The Crystal Oscillator (16 MHz) gives a stable frequency input to the ATmega328P.
- Starts vibrating when an electric field is applied.
- Resonates at a fixed frequency (16 million cycles per second).
- The chip uses the vibrating crystal to generate a square wave.
- This wave becomes the system clock for all operations (instruction execution, peripheral timing, etc.).
- Arduino Uno also has a small 8 MHz ceramic resonator near the USB interface (ATmega16U2), used for USB communication timing.

ARDUINO

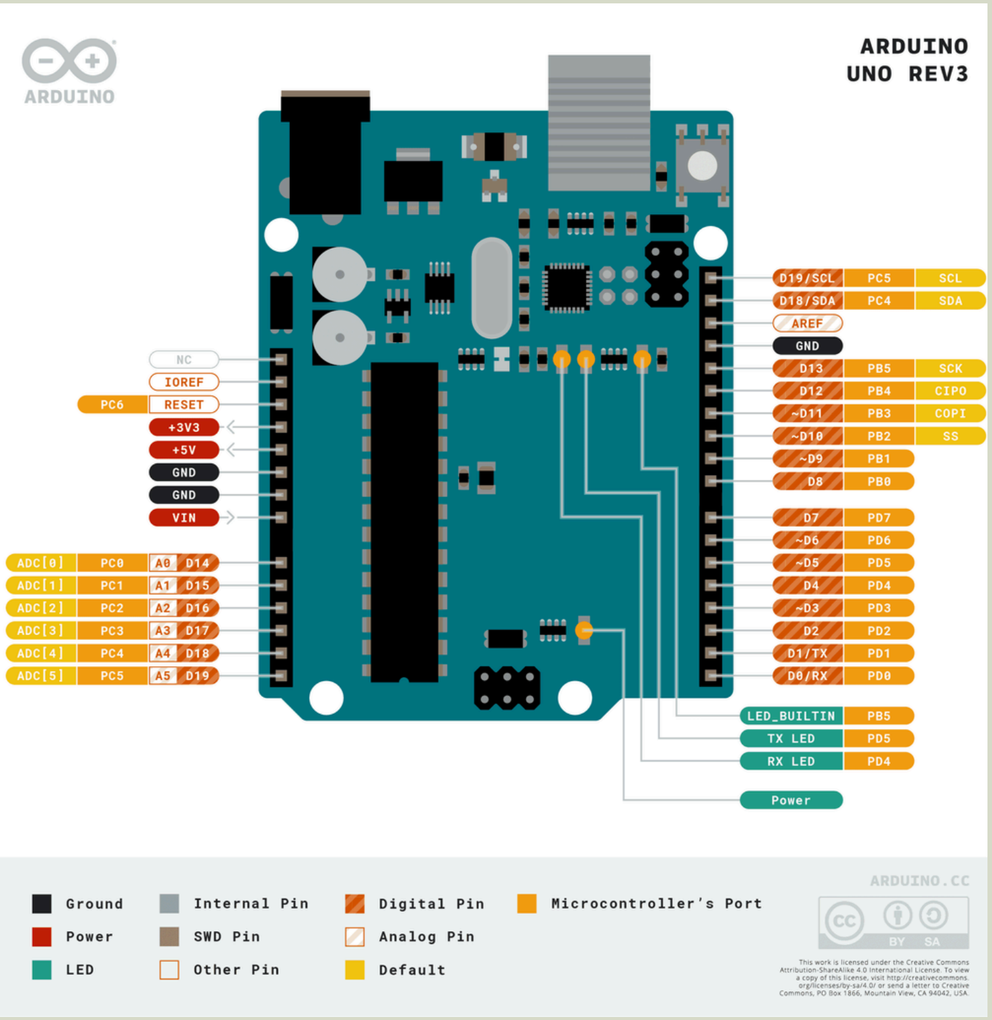
ARDUINO UNO REV3



Ground
Power
LED
Internal Pin
SWD Pin
Other Pin
Digital Pin
Analog Pin
Default
Microcontroller's Port

ARDUINO.CC

ANALOG AND DIGITAL PINS



Digital Pins (0–13):

- HIGH or LOW logic, some support PWM.
- 2 states (0V or 5V), used for on/off signals.

Analog Pins (A0–A5):

- Connect to sensors; input to ADC.
- Reads variable voltages (0–5V) as 10-bit values (0–1023).
- While the main function of the analog pins for most Arduino users is to read analog sensors, the analog pins also have all the functionality of general-purpose input/output (GPIO) pins (the same as digital pins 0 - 13).

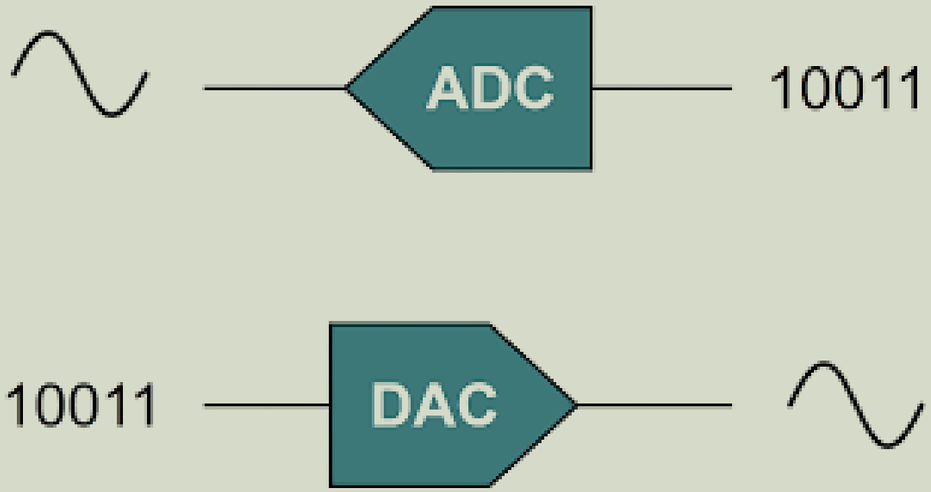
PWM Pins (~):

- Output simulated analog voltages using PWM.

ANALOG AND DIGITAL PINS

- **Serial:** These pins are categorized into two types, namely receive (RX) and transmit (TX) serial data. The board has TX and RX LEDs. TX flashes the LED while data is sent, and RX flashes when data is being received.
- **External Interrupts:** External interrupts are used to trigger an interrupt when required. This interruption can be due to a rising or falling edge, or a change in value. Once an interrupt is called, the Arduino will come to a halt and begin working only when told. These pins are PIN '2' and '3', which are controlled using the attachInterrupt() function.
- **SPI:** This is a synchronous serial data protocol generally used by microcontrollers. This is present at pin number 10 (SS), 11 (MOSI), 12 (MISO), and 13 (SCK), which are used by microcontrollers for communicating with different devices.
- **LED:** Present at PIN 13 in some Arduino, the LED is often used for testing purposes. The LED glows when the pin is HIGH, and turns off when the pin is LOW.
- **I2C**: These pins are present at numbers 4 (SDA) and 5 (SCL) and are used to perform I2C (TWI) communication.

ADC AND DAC



ADC (Analog to Digital Converter)

- Converts analog input to digital values (0–1023 on 10-bit ADC).
- Arduino Uno has 6 on-board ADC channels which can be used to read analog signals in the range 0- 5V.
- Enables reading values from analog sensors like potentiometers.

DAC (Digital to Analog Converter)

- UNO lacks true DAC — uses PWM to simulate analog output by toggling pins rapidly.

ANALOG TO DIGITAL CONVERTER

It has a 10-bit ADC means it will give a digital value in the range of 0 – 1023 (2^10). This is called as a resolution, which indicates the number of discrete values it can produce over the range of analog values.

Digital Output Value Calculation:

ADC Resolution = Vref/ ((2^n) - 1)

Digital Output = Vin / Resolution

Vref- Maximum value that the ADC can convert.

Example:

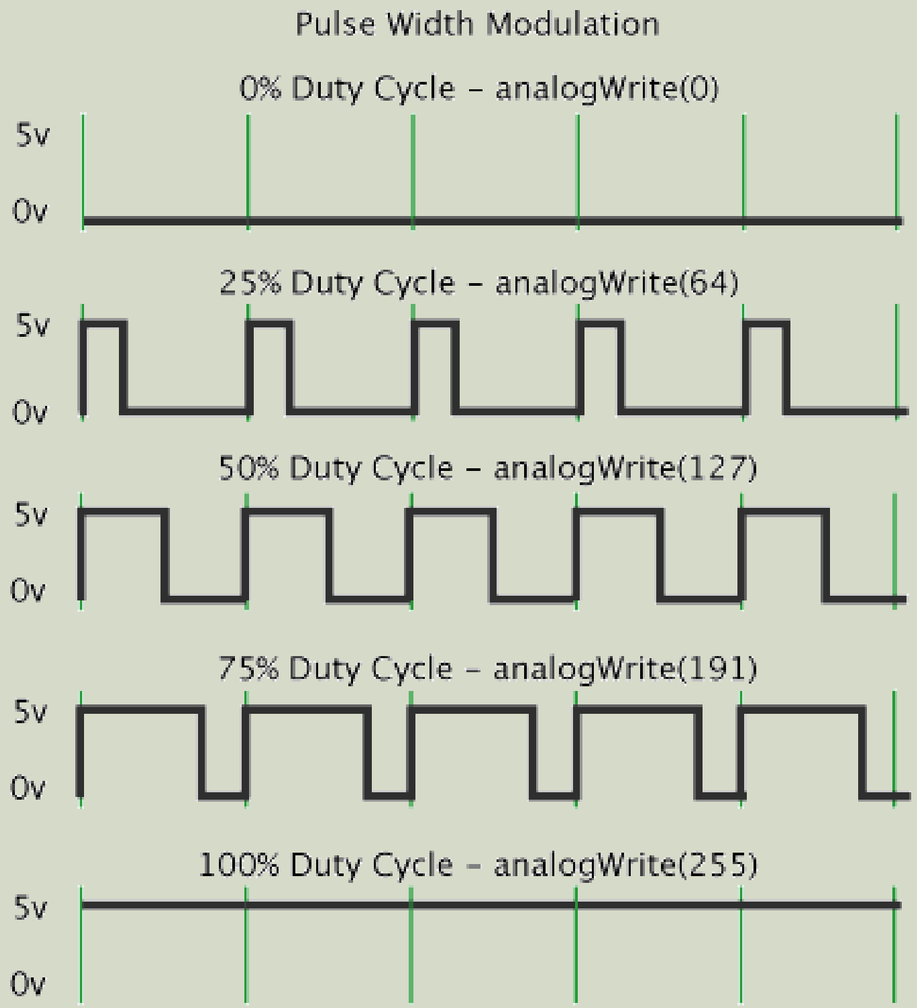
Vref is 5V and n is 10,

For 0 Vin, digital o/p value = 0

For 5 Vin, digital o/p value = 1023 (10-bit)

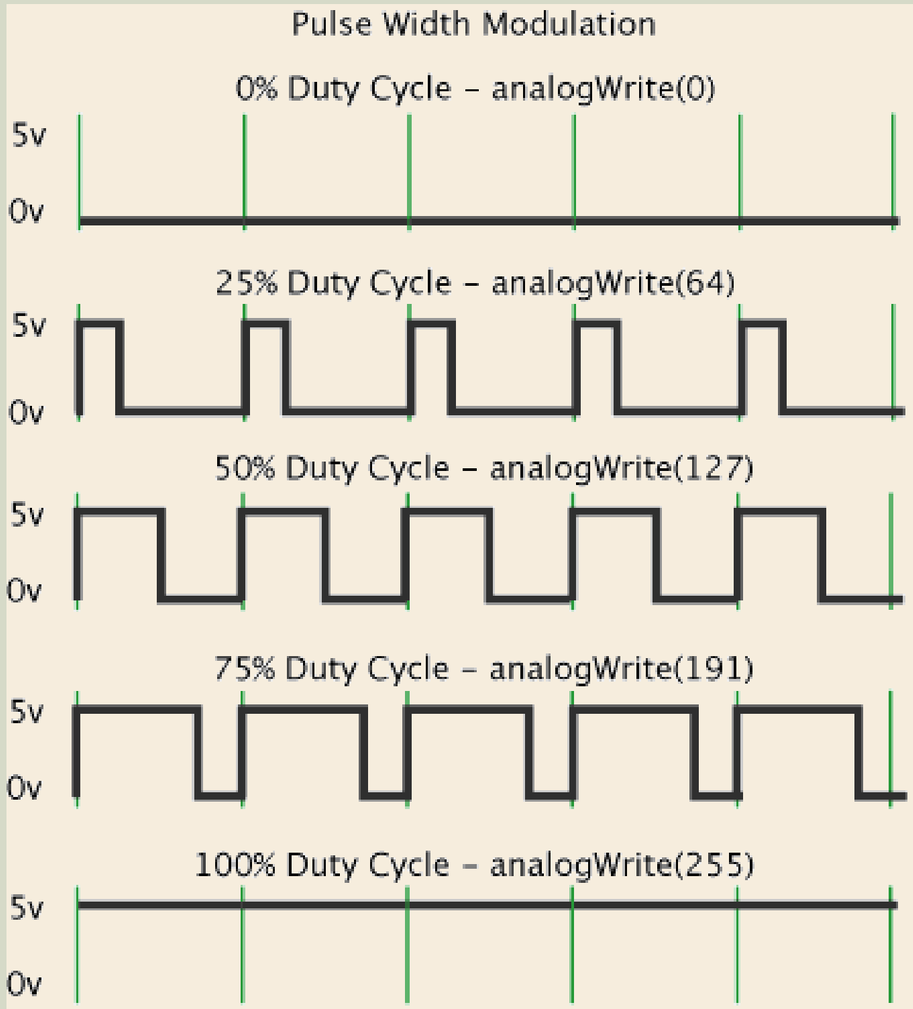
For 2.5 Vin, digital o/p value = 512 (10-bit)

PULSE WIDTH MODULATION



- Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means.
- Simulates voltages between the board's full Vcc(5 volts)and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off.
- The duration of "on time" is called the pulse width. To get varying analog values, you change, or modulate, that pulse width.
- Used for dimming LEDs, controlling motors, etc.
- E.g., 50% duty cycle → 2.5V average output over time.

PULSE WIDTH MODULATION



Avg voltage value:

$$\frac{1}{T}\int_0^{T_{ON}} V_{ref}\,dt = \frac{1}{T}\left[V_{ref}\,t\right]_0^{T_{ON}}$$

If 25% of Duty cycle, $T_{ON} = T/4$

$$\Rightarrow \frac{V_{ref}\,[T/4]}{T} = \frac{V_{ref}}{4}$$

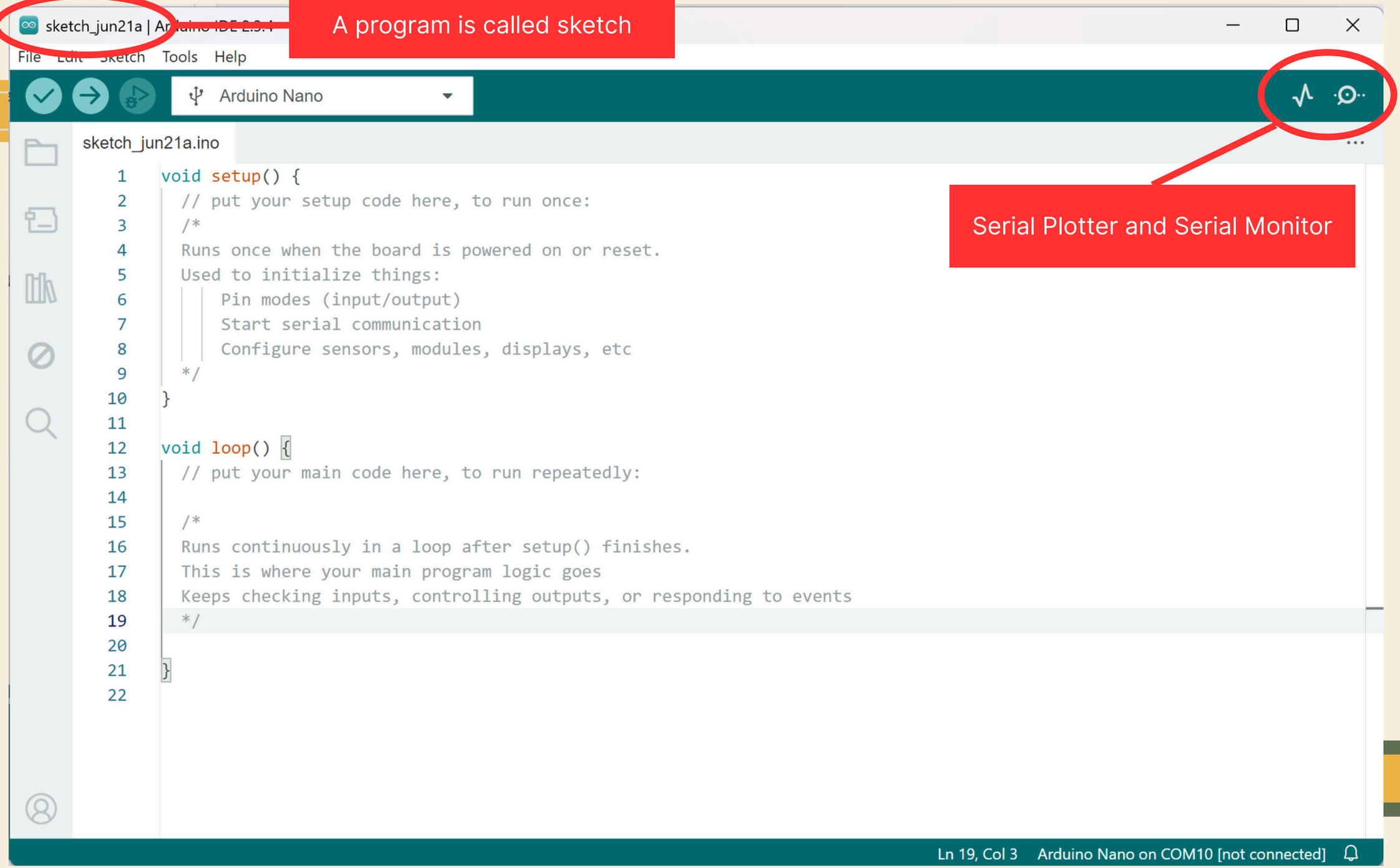
If 50% of Duty cycle, $T_{ON} = T/2$

$$\Rightarrow \frac{V_{ref}\,(T/2)}{T} = \frac{V_{ref}}{2}$$

ARDUINO IDE

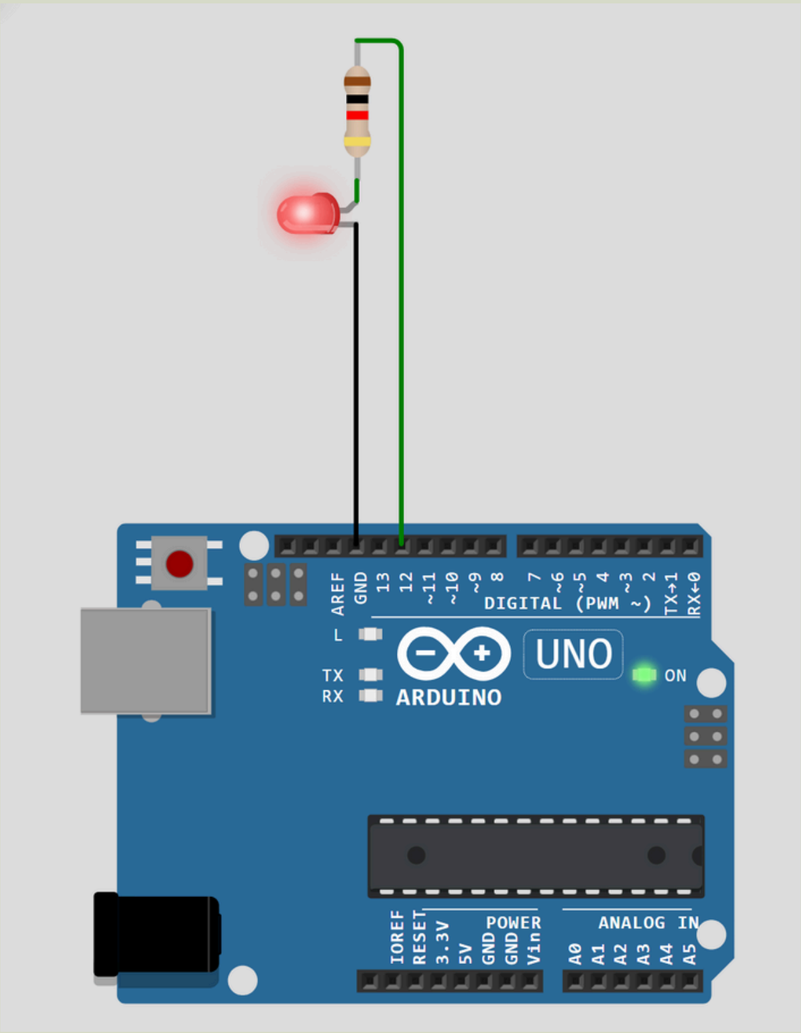
Official software used to write, compile, and upload code to Arduino boards and many other microcontrollers

Based on C/C++ with a simplified structure and some custom libraries provided by Arduino


sketch_jun21a | Ar
A program is called sketch
File Edit Sketch Tools Help
Arduino Nano
Serial Plotter and Serial Monitor
sketch_jun21a.ino
1 void setup() {
2 // put your setup code here, to run once:
3 /*
4 Runs once when the board is powered on or reset.
5 Used to initialize things:
6 Pin modes (input/output)
7 Start serial communication
8 Configure sensors, modules, displays, etc
9 */
10 }
11
12 void loop() {
13 // put your main code here, to run repeatedly:
14
15 /*
16 Runs continuously in a loop after setup() finishes.
17 This is where your main program logic goes
18 Keeps checking inputs, controlling outputs, or responding to events
19 */
20
21 }
22
Ln 19, Col 3 Arduino Nano on COM10 [not connected]

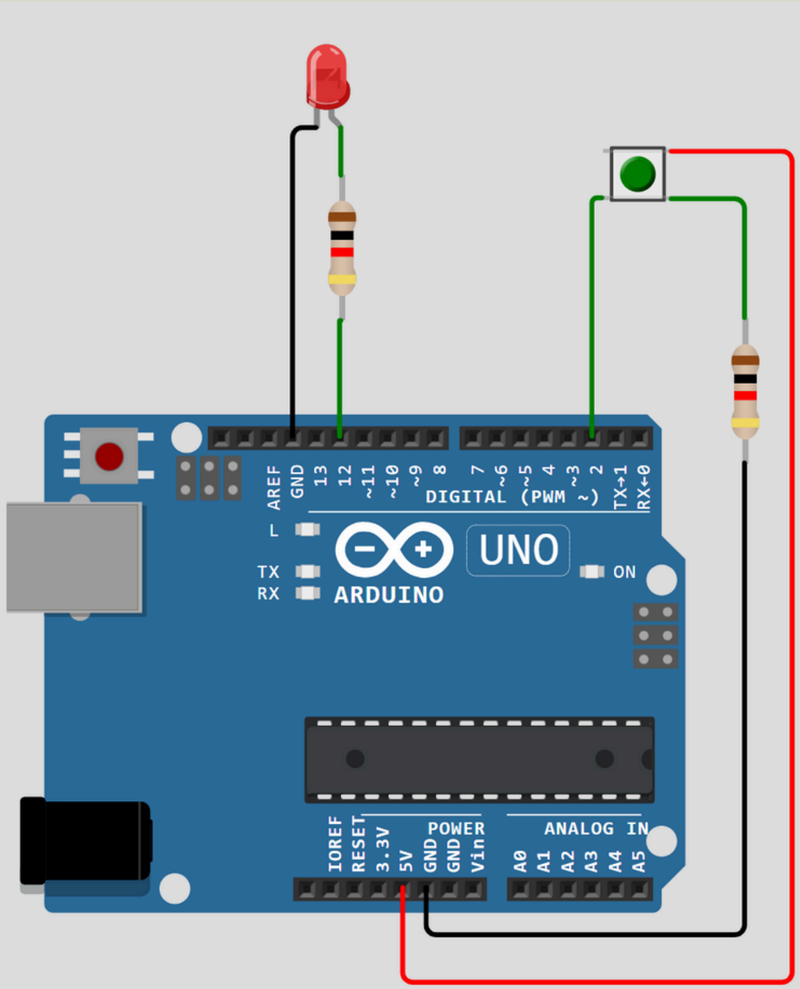
LED BLINKER

The ‘Hello World’ of Embedded programming




```
int LED = 12;
int delayTime = 1000;

void setup() {
 pinMode(LED, OUTPUT);
}

void loop() {
 digitalWrite(LED, LOW);
 delay(delayTime);
 digitalWrite(LED, HIGH);
 delay(delayTime);
}
```

LED AND BUTTON



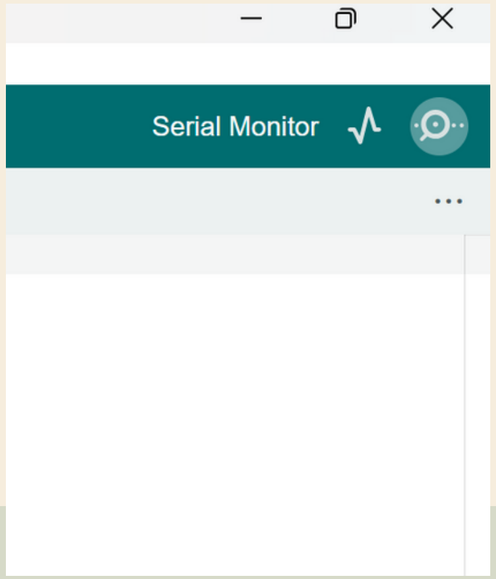

```
int button = 2;
int led = 12;

void setup() {
 pinMode(button, INPUT);
 pinMode(led, OUTPUT);
}

void loop() {
 if (digitalRead(button) == HIGH){
  digitalWrite(led, HIGH);
 }
 else {
  digitalWrite(led, LOW);
 }
}
```

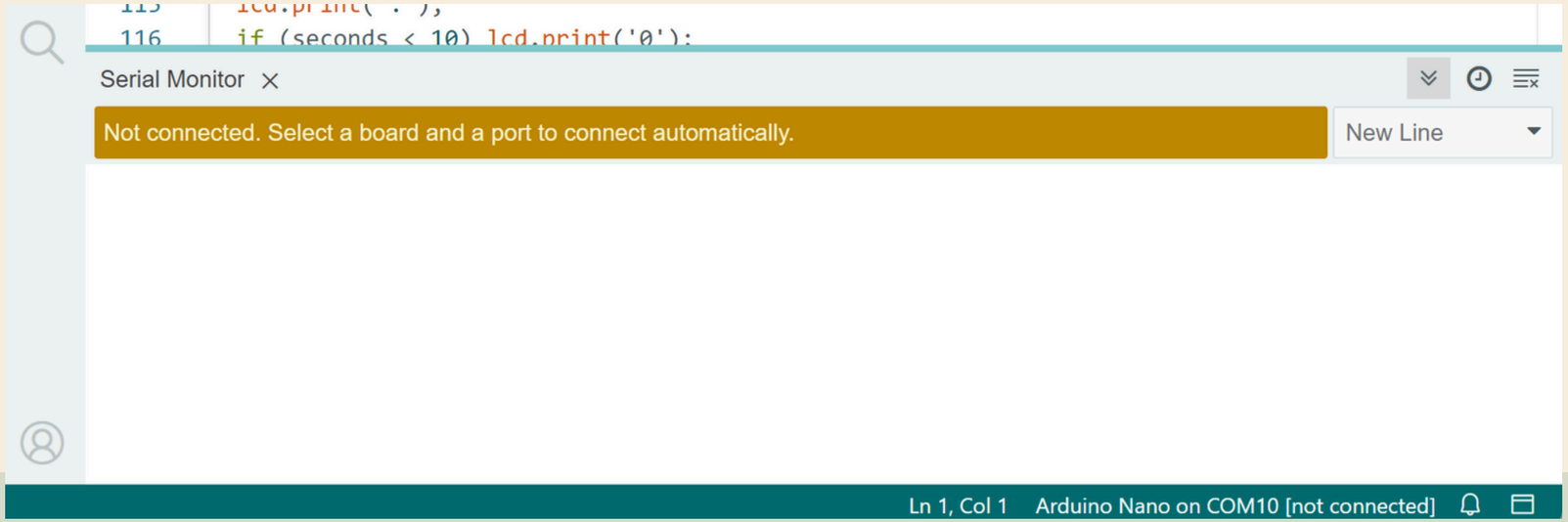
SERIAL PRINTING

Serial Monitor is a built-in tool in the Arduino IDE that lets you communicate with your Arduino board by sending and receiving text data over USB.

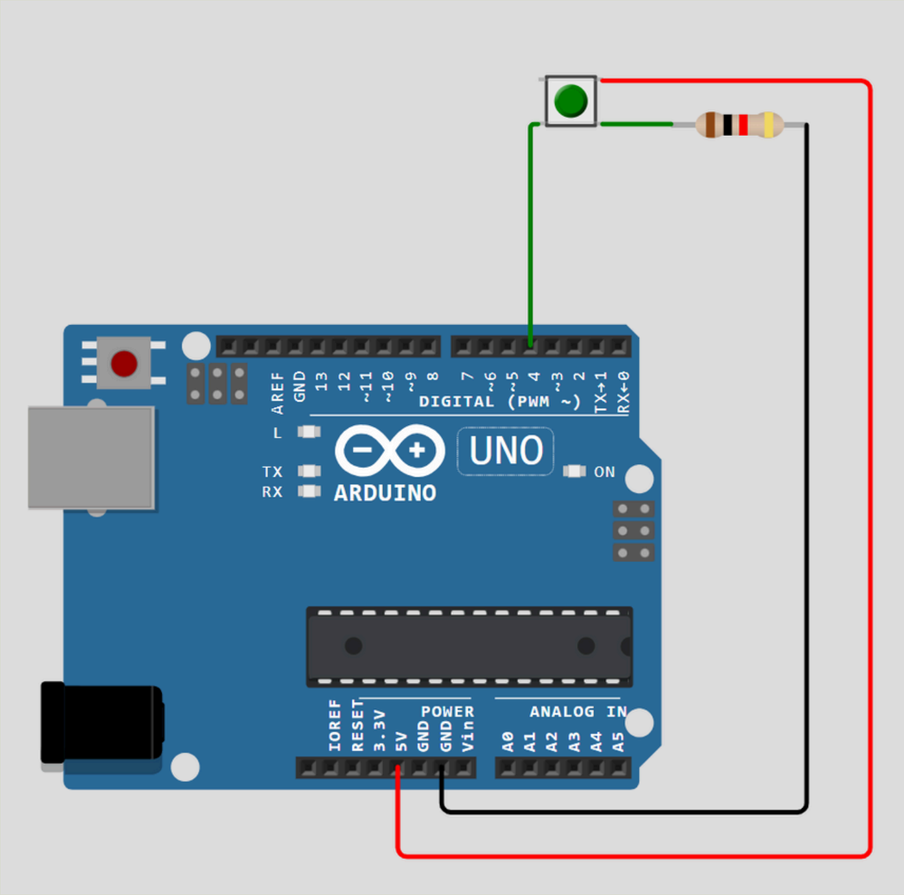
- Displays data sent from the Arduino using Serial.print() or Serial.println()
- Sends text or commands from your computer to the Arduino
- Helps with debugging, monitoring sensors, or checking program behavior

Baud Rate or the Bit Rate is the number of Bits sent every second.





SERIAL PRINTING




```
int button = 4;
int buttonVal;

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

void loop() {
 buttonVal = digitalRead(button);
 Serial.println(buttonVal);
 delay(500);
}
```

ASSIGNMENT

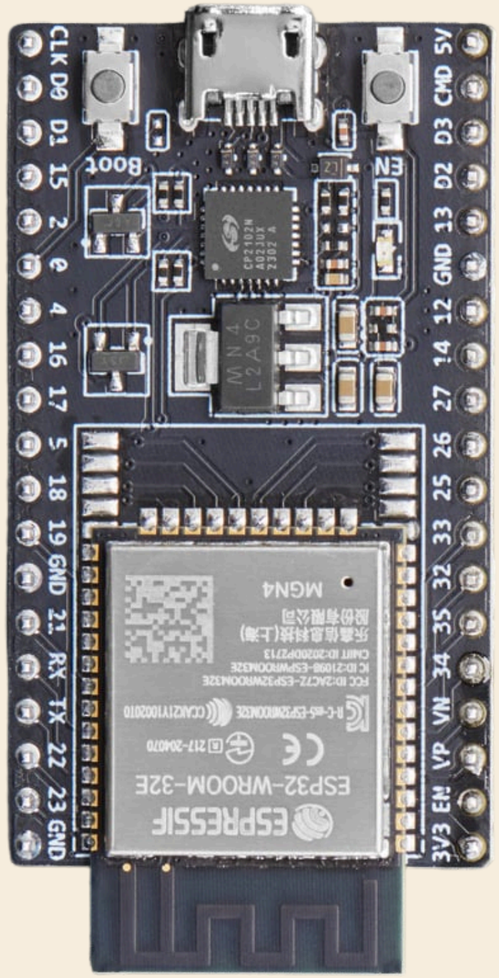
Simulate **AND** and **OR** gates using an Arduino board, two push buttons and two LEDS. One LED should glow for AND condtions and one for OR.
Also print the gate status for both on the Serial Monitor.

Submit a recording of the circuit in action and the code by Wednesday (25/06/26).

Tip: If you are doing a simulation on Wokwi, set a keyboard shortcut for each button so that you can simultaneously press both buttons.

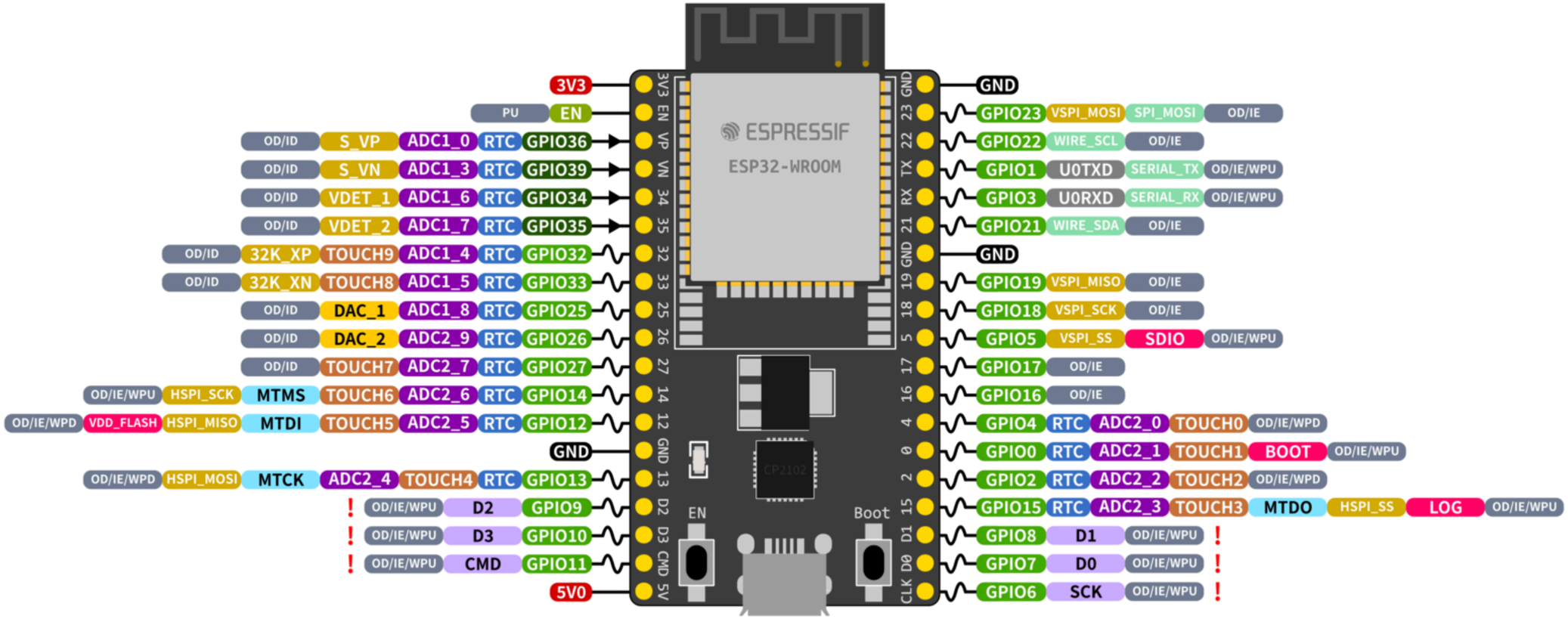
ESP 32 DESCRIPTION

- Esp-32 series is successor to esp8266 and and it just represent every microcontroller built using the esp32 chipset, manufactured by expressif and has different variants
- most common ones are devkit,wroom,cam - these have different features and slightly vary for different purposes
- we will look into dev kit for now
- all esp 32 boards and wifi and BLE enabled hence its a cheap and powerful wireless board
- https://www.adafruit.com/product/3269
- pinout indicates the purpose and capability of each pin
- the physical pin number does not correspond to its gpio
- ie to reference a pin in the code refer to the pinout or the markings on the board instead of counting the pins because they are two different meanings
- gpio numbers usually correspond to the chipset pin number and do not have to correspond to the board’s pin number



ESP32-DevKitC

ESP 32 PINOUT



ESP32 Specs

32-bit Xtensa® dual-core @240MHz
Wi-Fi IEEE 802.11 b/g/n 2.4GHz
BLuetooth 4.2 BR/EDR and BLE
520 KB SRAM (16 KB for cache)
448 KB ROM
34 GPIOs, 4x SPI, 3x UART, 2x I2C,
2x I2S, RMT, LED PWM, 1 host SD/eMMC/SDIO,
1 slave SDIO/SPI, TWAI®, 12-bit ADC, Ethernet

PWM Capable Pin
GPIOX GPIO Input Only
GPIOX GPIO Input and Output
DAC_X Digital-to-Analog Converter
DEBUG JTAG for Debugging
FLASH External Flash Memory (SPI)
ADCX_CH Analog-to-Digital Converter
TOUCHX Touch Sensor Input Channel
OTHER Other Related Functions
SERIAL Serial for Debug/Programming
ARDUINO Arduino Related Functions
STRAP Strapping Pin Functions

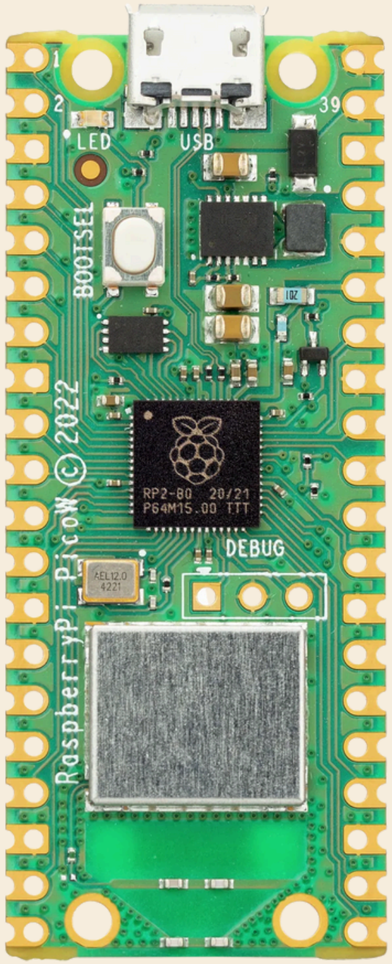
RTC RTC Power Domain (VDD3P3_RTC)
GND Ground
PWD Power Rails (3V3 and 5V)
! Pin Shared with the Flash Memory
Can't be used as regular GPIO

GPIO STATE

WPU: *Weak Pull-up (Internal)*
WPD: *Weak Pull-down (Internal)*
PU: *Pull-up (External)*
IE: *Input Enable (After Reset)*
ID: *Input Disabled (After Reset)*
OE: *Output Enable (After Reset)*
OD: *Output Disabled (After Reset)*

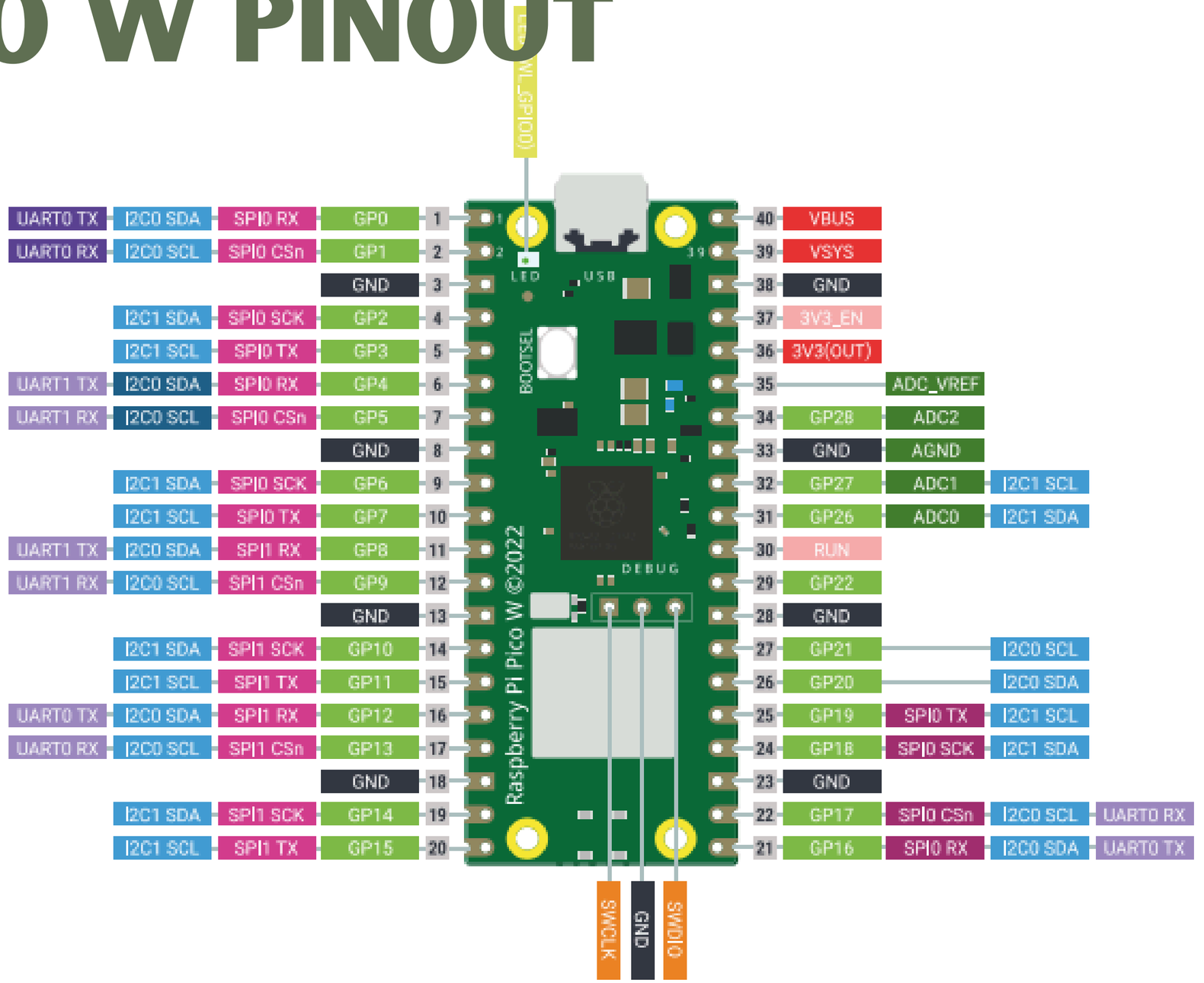
PICO W DESCRIPTION

- its the third variant of the Pico series, manufactured by raspberry pi
- it is the same as the base pico but with the addon of a wifi and ble module, this is external to the chipset but in esp 32 its embodied within the same chip
- form factors and speed performances are different, choosing between the two boards comes down to specifics but in general they are mutually replaceable
- same as esp 32 but a bit more streamlined and has additional capabilities here
- uses the rp2040 chipset
- detailed info here
  https://datasheets.raspberrypi.com/picow/pico-w-datasheet.pdf



The pinout of
Pico W board

PICO W PINOUT



RP2040

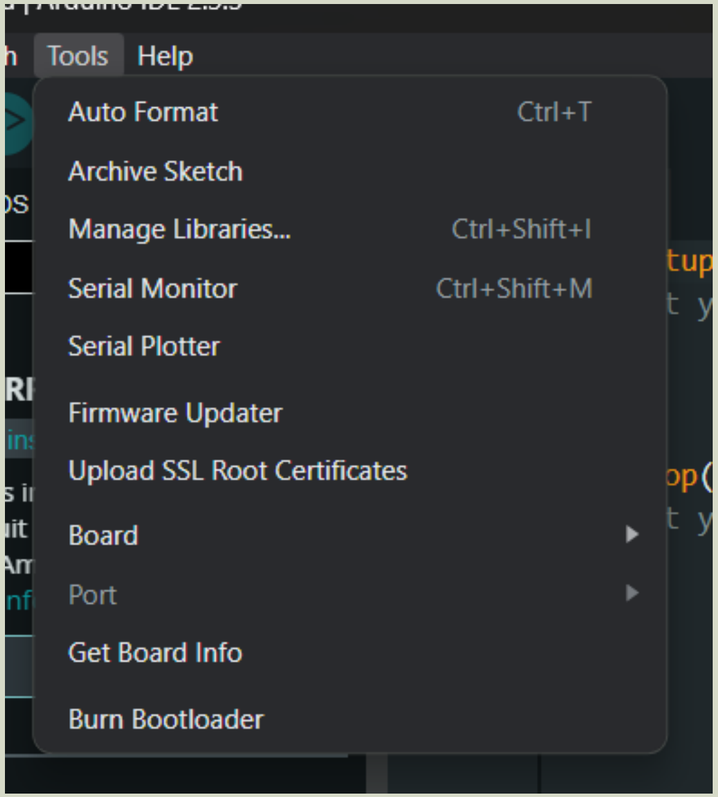
- Power
- Ground
- UART / UART (default)
- GPIO, PIO, and PWM
- ADC
- SPI / SPI (default)
- I2C / I2C (default)
- System Control
- Debugging

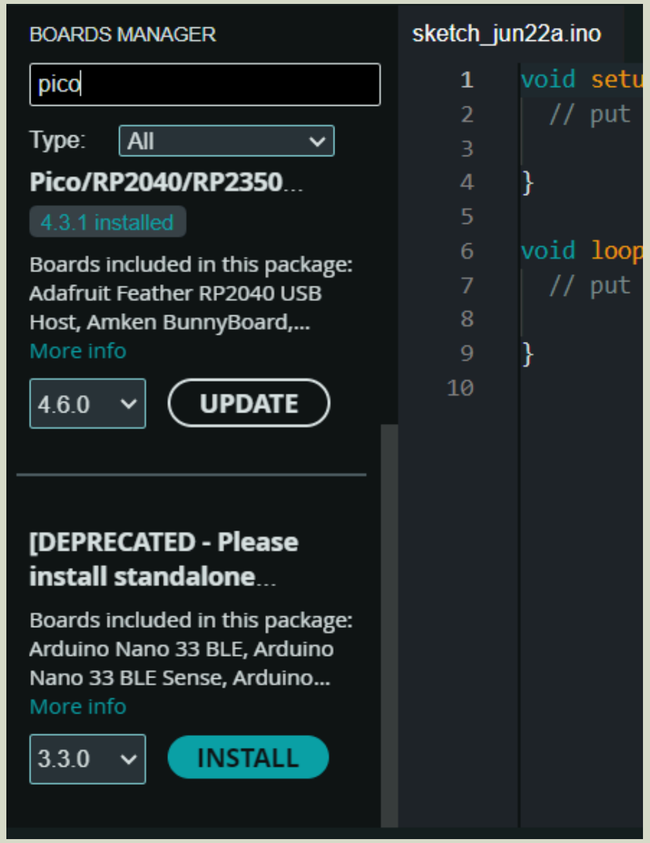
Infineon 43439

- GPIO

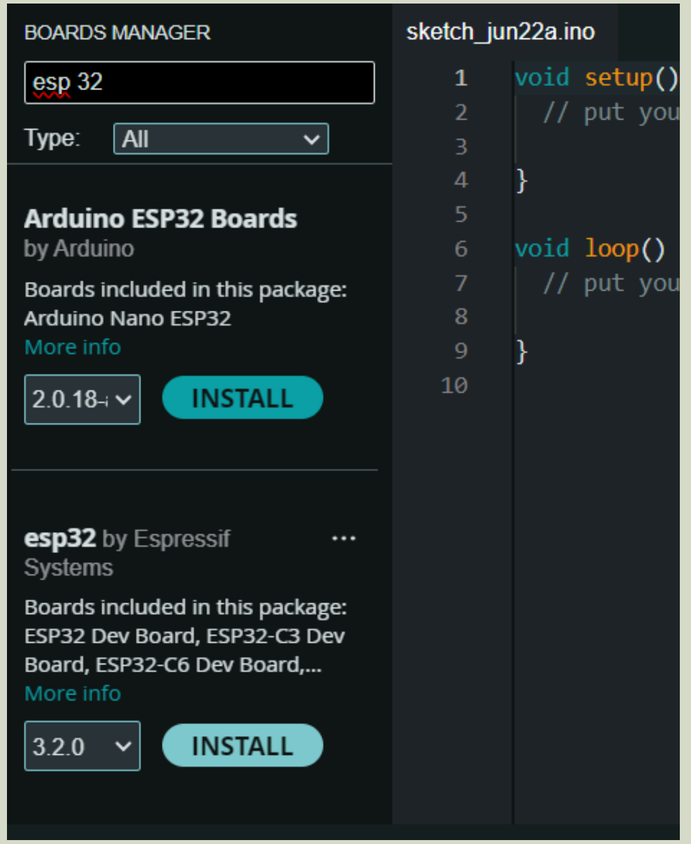
HOW TO ADD THESE TO ARDUINO IDE

- Goto tools
- Click on board manager and search for respective boards

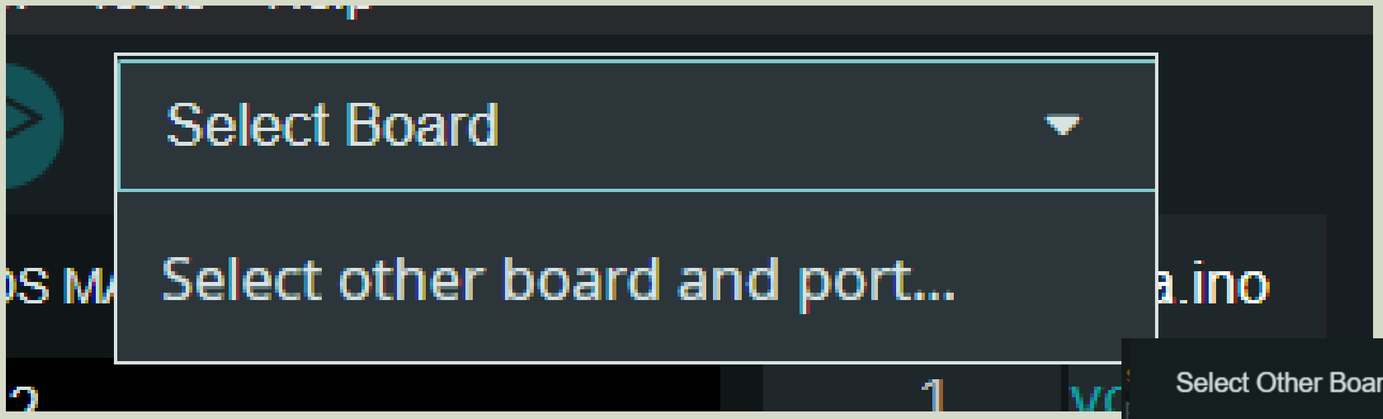


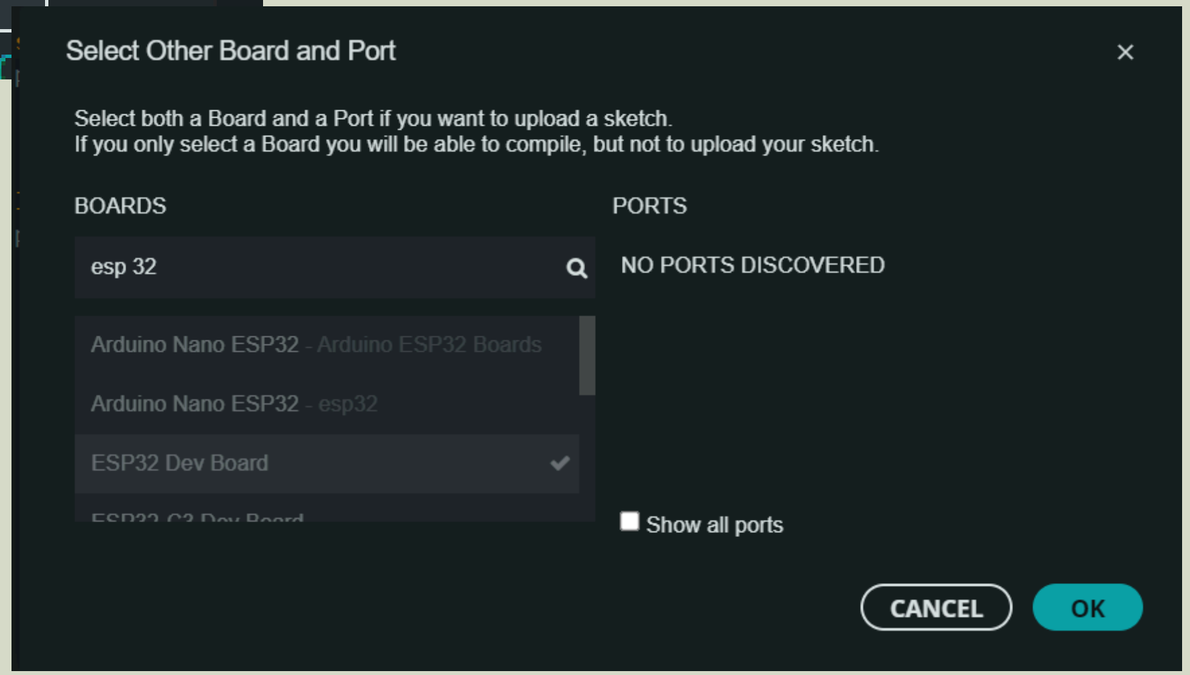


or



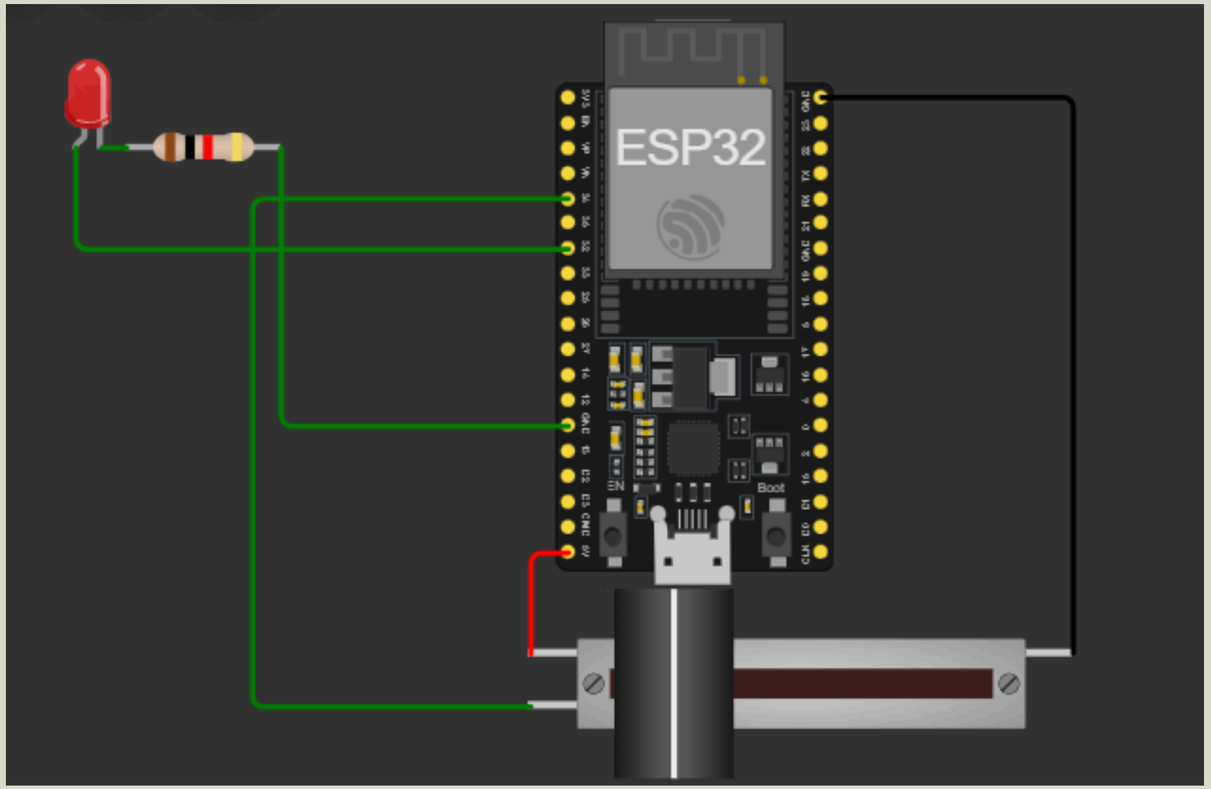
UPLOADING CODE





- after writing your code go onto select board and search for the board name
- once selected, select the right port and upload the code

ANALOG READ AND WRITE




```
int sensorPin = 34;   // select the input pin for the potentiometer
int ledPin = 35;      // select the pin for the LED
int sensorValue = 0;  // variable to store the value coming from the sensor

void setup() {
  // declare the ledPin as an OUTPUT:
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // read the value from the sensor:
  sensorValue = analogRead(sensorPin);
  // turn the ledPin on
  analogWrite(ledPin,sensorValue);
}
```

WHY WE ARE USING THESE TWO

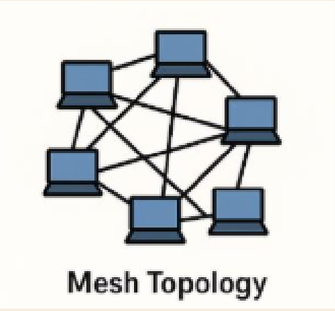
- both of these boards are really well documented and mostly commonly used and most of all with an inbuilt wifi and networking module they easily interface the net
- super low power and inexpensive and easily deployable

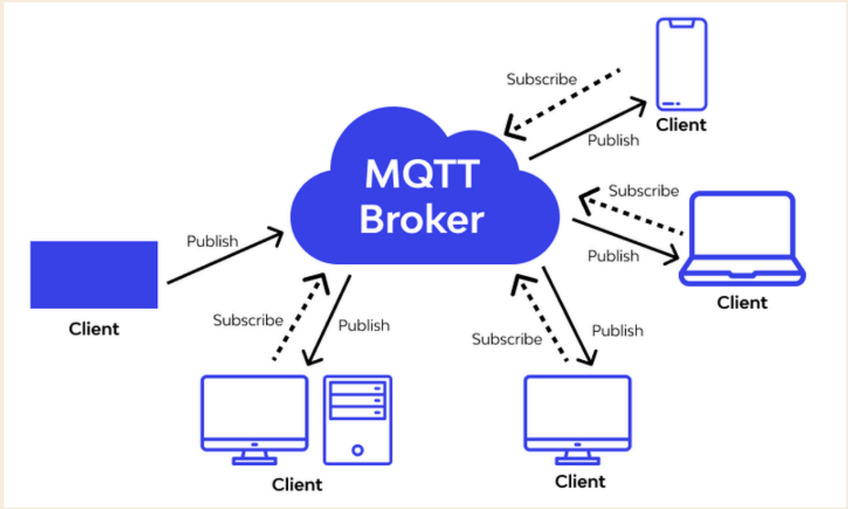
- a few lines of code and we have a working mqtt client
- simplest initialisation code

```
/*********
  Rui Santos
  Complete project details at https://randomnerdtutorials.com
*********/

#include <WiFi.h>
#include <PubSubClient.h>
#include <Wire.h>
#include <Adafruit_BME280.h>
#include <Adafruit_Sensor.h>

// Replace the next variables with your SSID/Password combination
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

// Add your MQTT Broker IP address, example:
//const char* mqtt_server = "192.168.1.144";
const char* mqtt_server = "YOUR_MQTT_BROKER_IP_ADDRESS";

WiFiClient espClient;
PubSubClient client(espClient);
long lastMsg = 0;
char msg[50];
int value = 0;
```

WHY MQTT

- lets say we have a broad eco system with multiple mcus and clients, if they were to connect traditionally ie one to one connection
- we end up realising a really complex mesh like network which is expensive to setup and manage
- each client needs to connect with every other client and keep listening to to every single one of them
- in a system dealing multiple devices keeping track of these and ensuring data reaches safely ends up being super hard and expensive
- and monitoring each datatype restriction for each channel becomes cumbersome

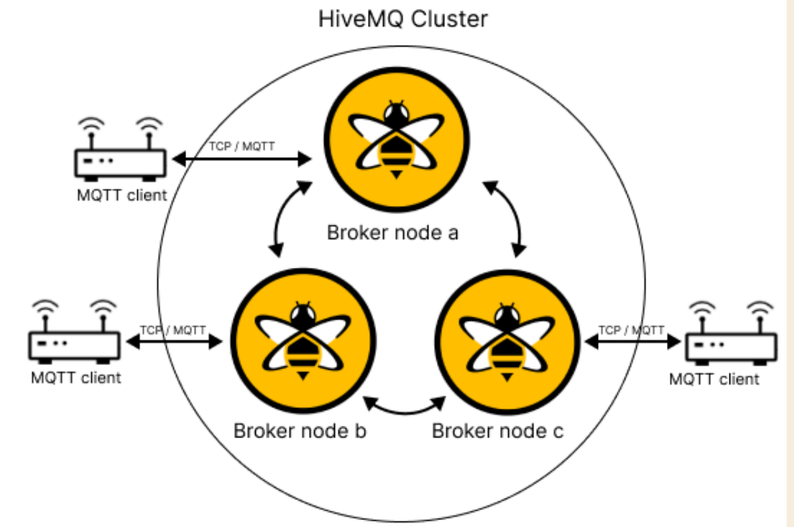




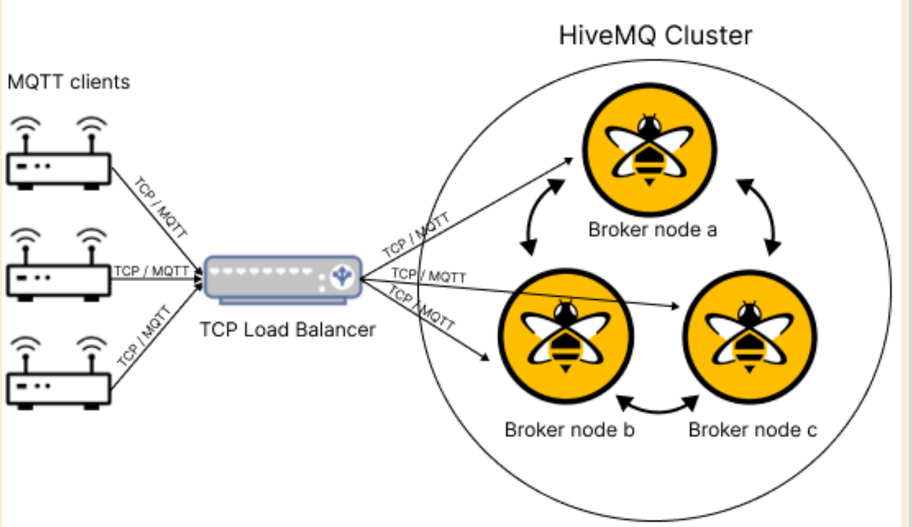
- in mqtt the work is offloaded onto a broker that makes sure to host all these topics and the need for these multiple individual connections is essentially nulled
- biggest disadvantage would be, in case of a broker breakdown it will disrupt the entire flow of data in which case we employ buffer brokers
- datatype has no restriction anything can be sent, as long as the listener/subscriber is listening data is transmitted
- the entire load of the system is handled by the main broker server

EXAMPLE OF A CLUSTERED BROKER SYSTEM

Without Load Balancing:



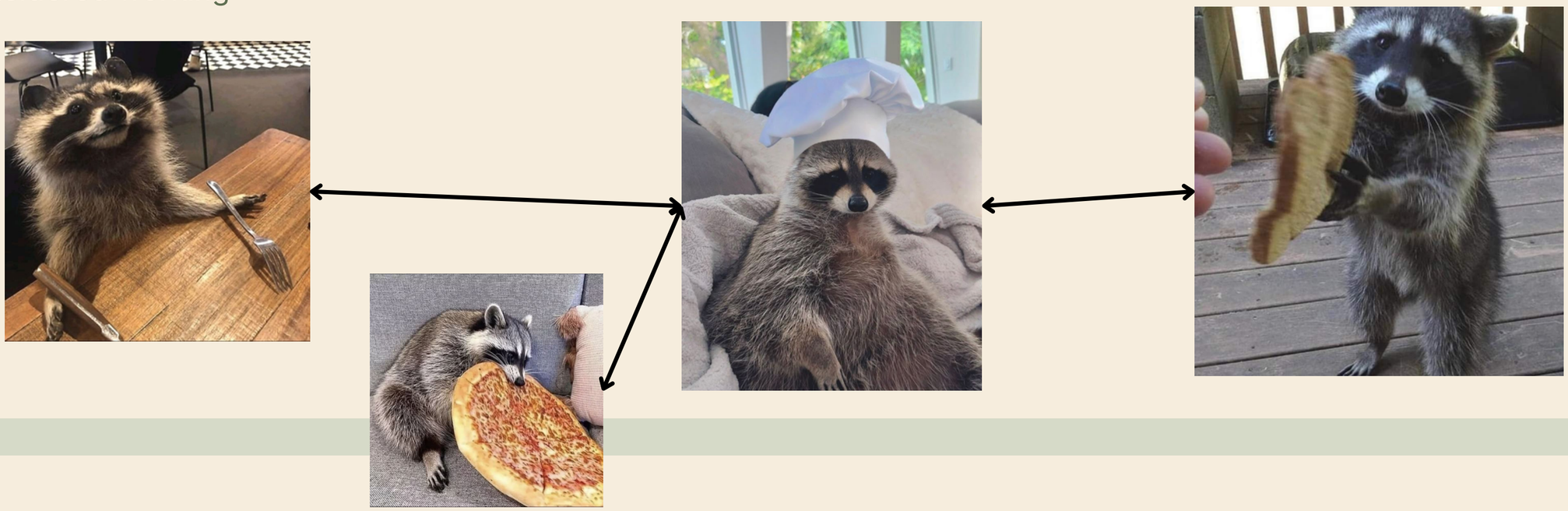
With Load Balancing:



- Different methods are employed to further ensure proper loading and unloading of data
- mqtt by default holds no data unless certain advanced msg types are enabled
- this allows lesser memory consumption and multiple brokers ensure multiple data sites and lesser loading onto a singular unit

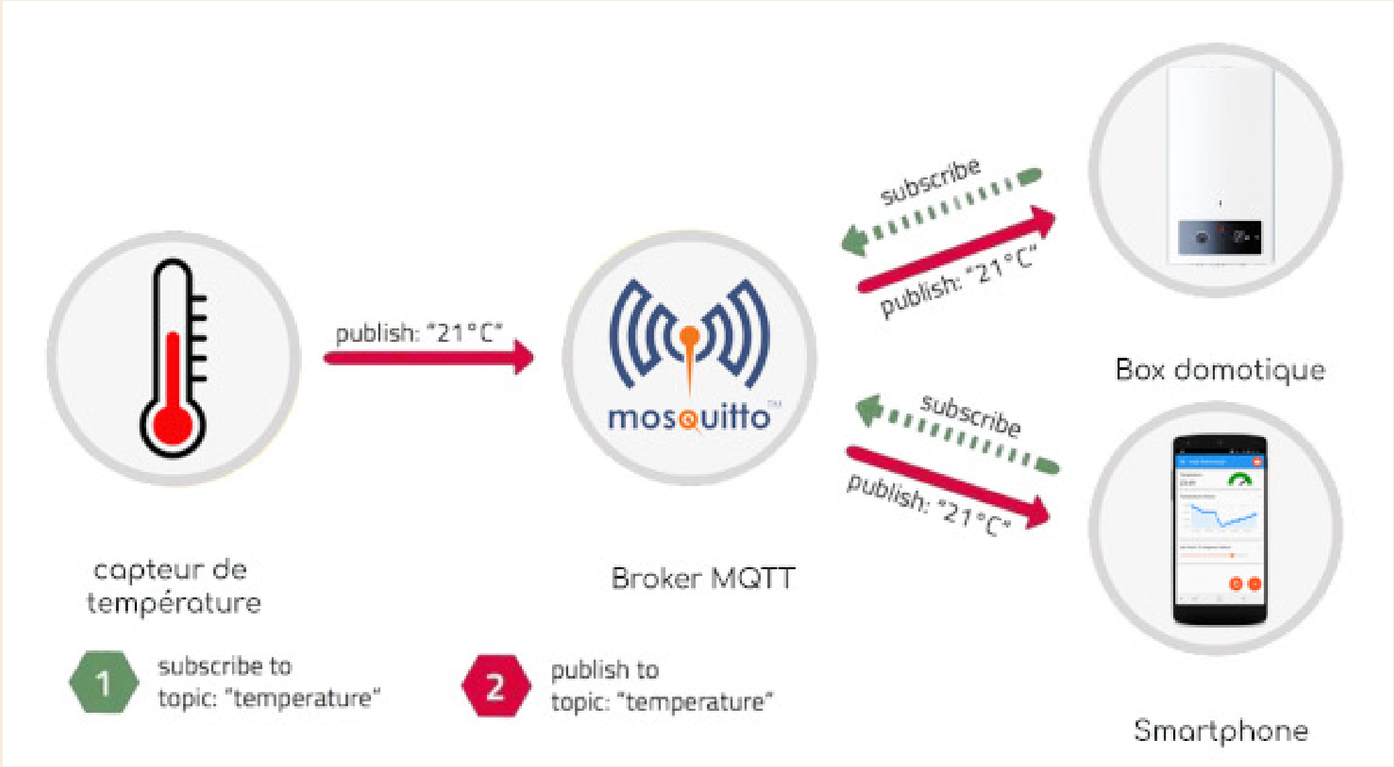
WHY MQTT IS SO LIGHT WEIGHT AND EFFICIENT

- practically because of how all load is put onto the server the mcu only needs to connect to a unique port and network to publish its respective data
- encryption is mainly server side and network handling is also server side
- client only connects and dumps data
- almost every language has a support for it and as long as the device connects and sends data the mqtt connection is considered working



WHAT MOSQUITTO REALLY IS

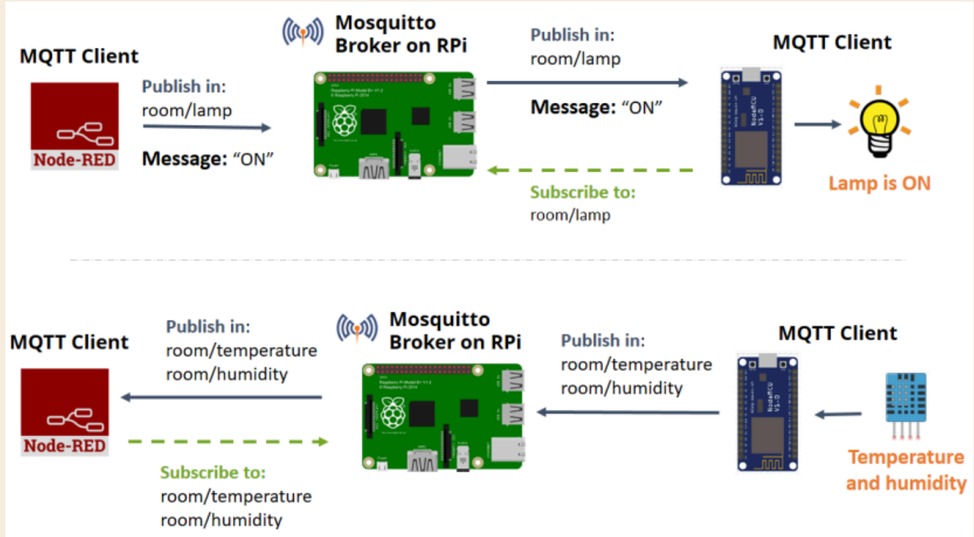
- when mqtt first came out as a concept, everyone had a different implementation of the broker server
- history aside mosquitto was one of the first such architectures made open source and is still mainstream with many commercial models based off of it



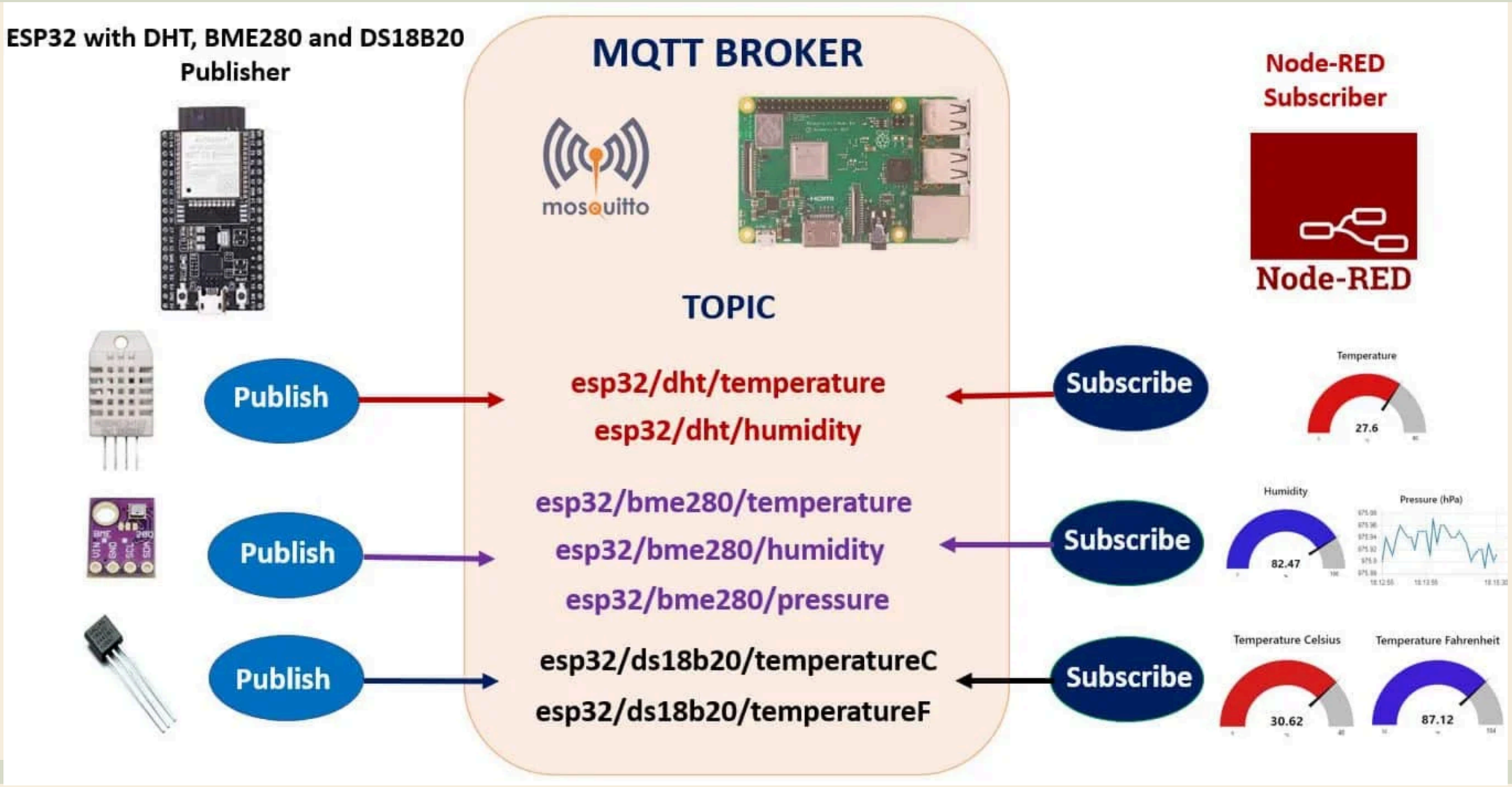
WHAT NODE RED REALLY IS

- node red is a super lightweight graphic visualiser for any sort of networking that happens on a device
- Node-RED is a powerful and versatile flow-based, low-code development tool for visual programming. Originally developed by IBM, it's designed to make it easy to wire together hardware devices, APIs, and online services, especially within the context of the Internet of Things (IoT)

- we install this on our broker to connect data points or simply visualise the network map
- it is not necessary to have node red to have an mqtt network
- mosquitto is the real backbone of mqtt for our case
- and node red is simply a tool to make this easier for us to connect and visualise
- it is technically a subscriber client that we run on the broker itself to gather data abt the connection and make changes locally



WHAT NODE RED REALLY IS



THANK YOU