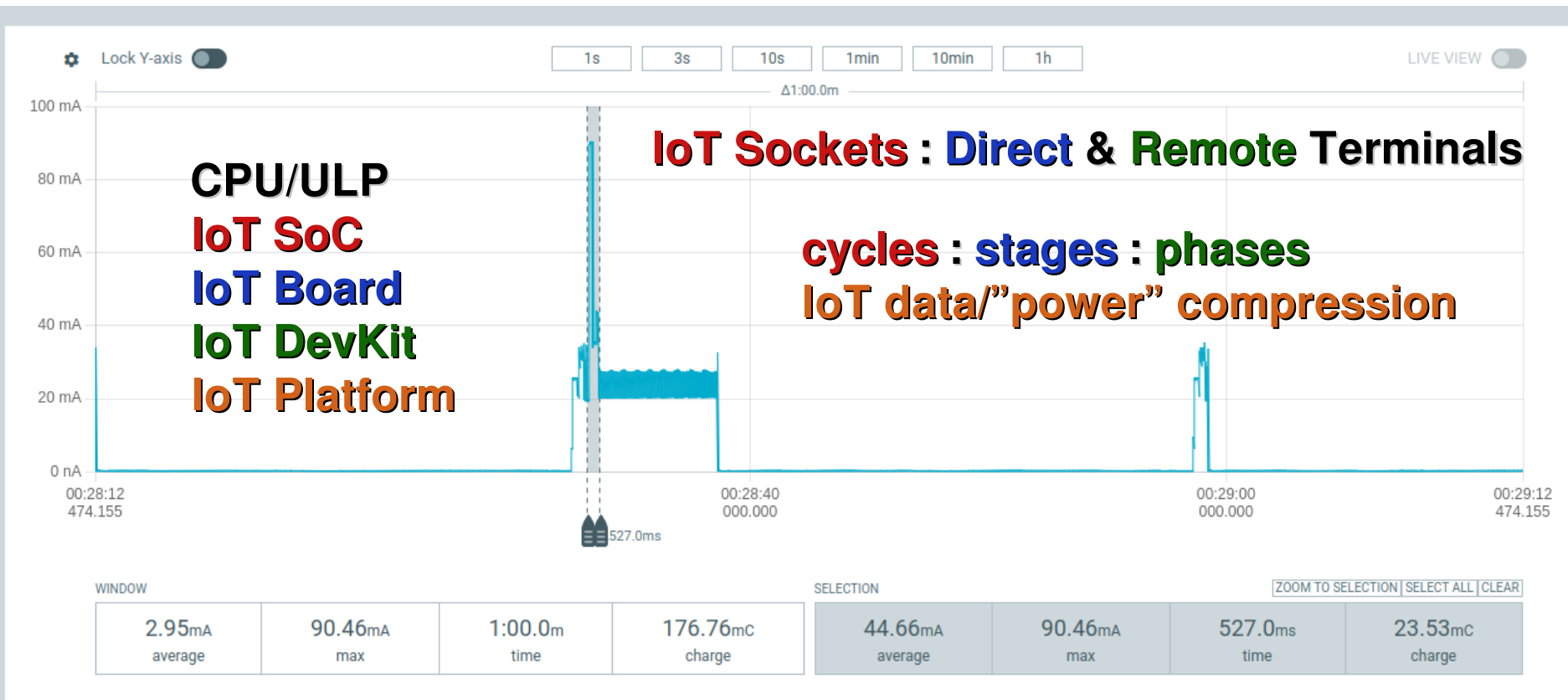
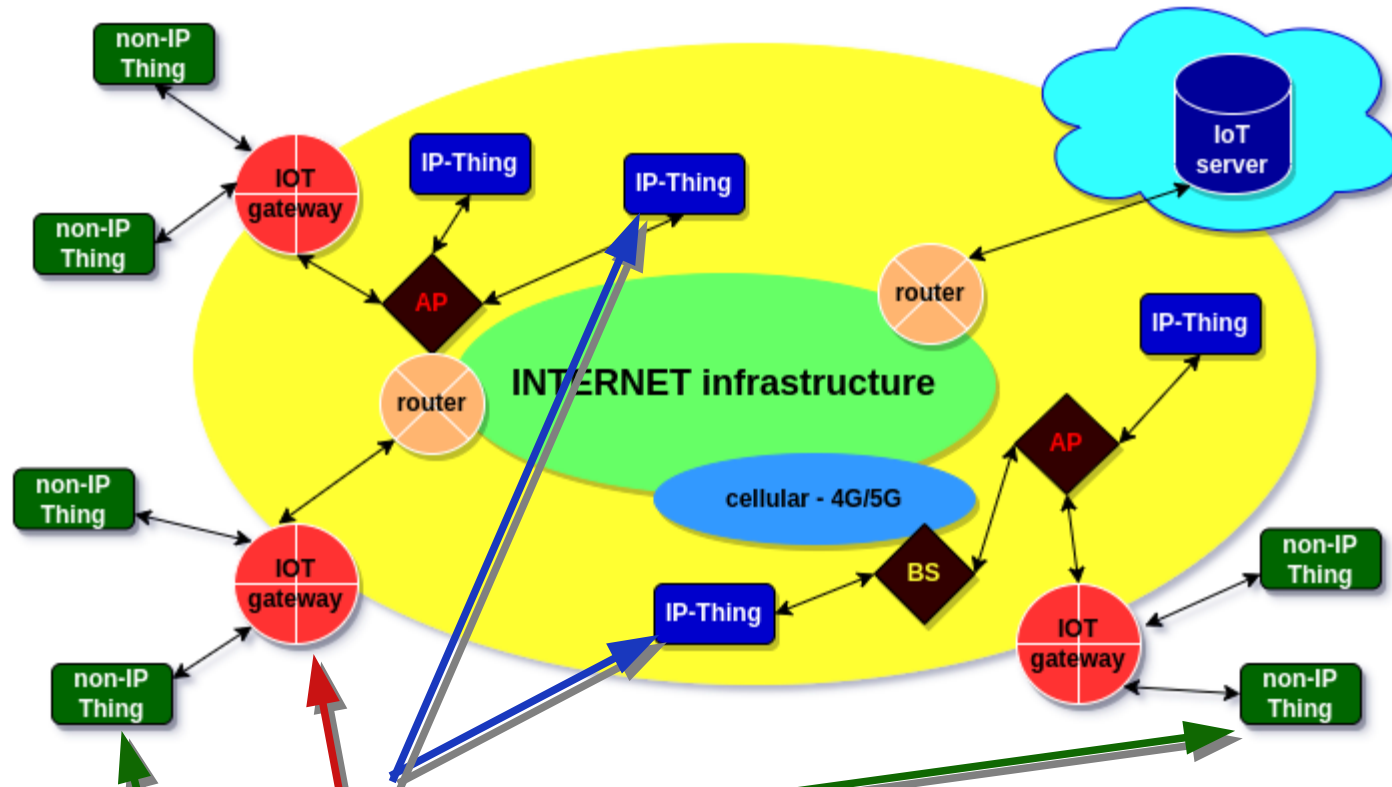


# Low Power IoT Architectures

## *Principles and Practices*

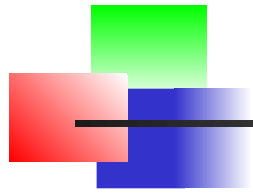


# IoT : Direct & Remote Terminals

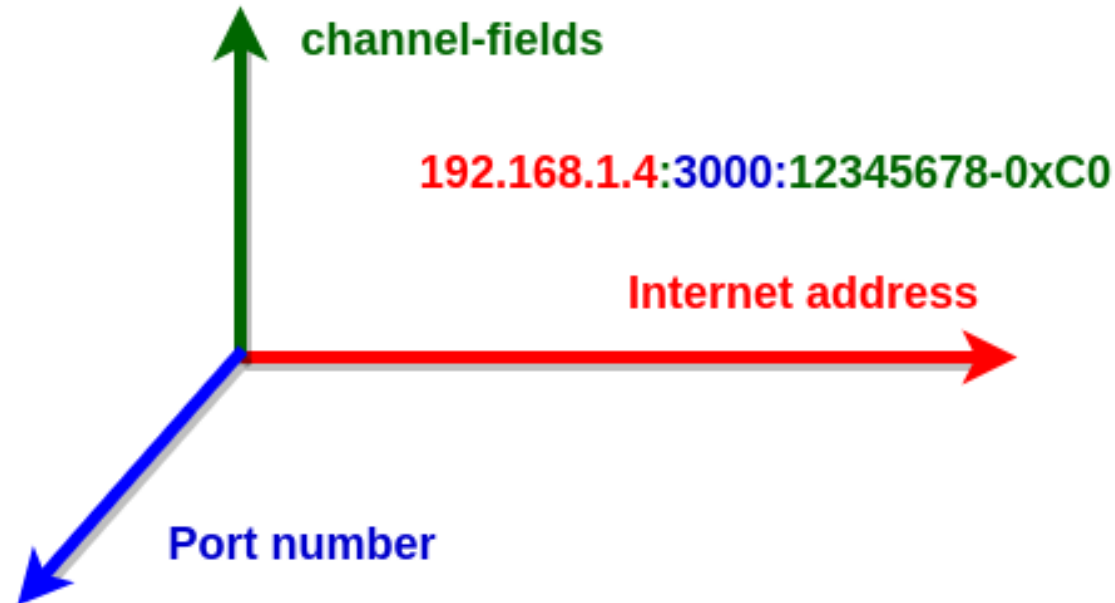


**IP-Thing : Direct Terminal** connected to INET via **WiFi/4G/5G** link

**non-IP-Thing : Remote Terminal** connected to INET via **LoRa** link and **IoT gateway**



# IoT Sockets : @IP:port:channel



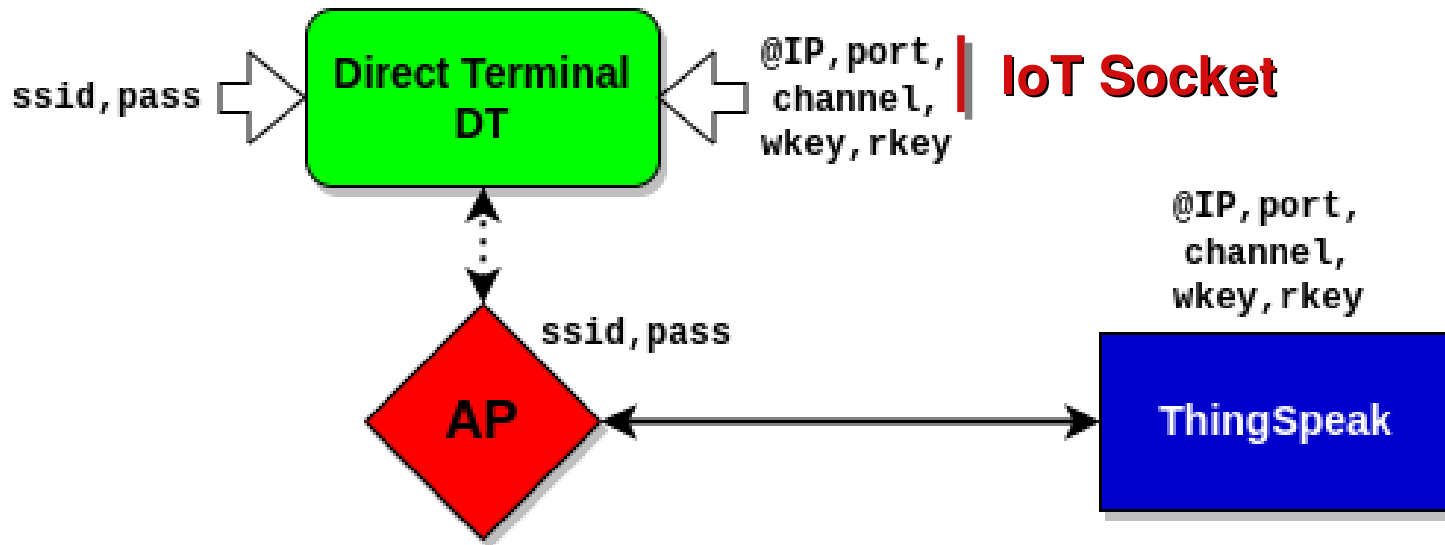
IoT socket => **IP address**: **Service port**: **Channel number-fields**

Direct Terminals know: **IP address**: **Service port**: **Channel number**

Gateways know: **IP address**: **Service port**

Remote Terminals know only: **Channel number** (identifier)

# Direct Terminals and IoT Sockets

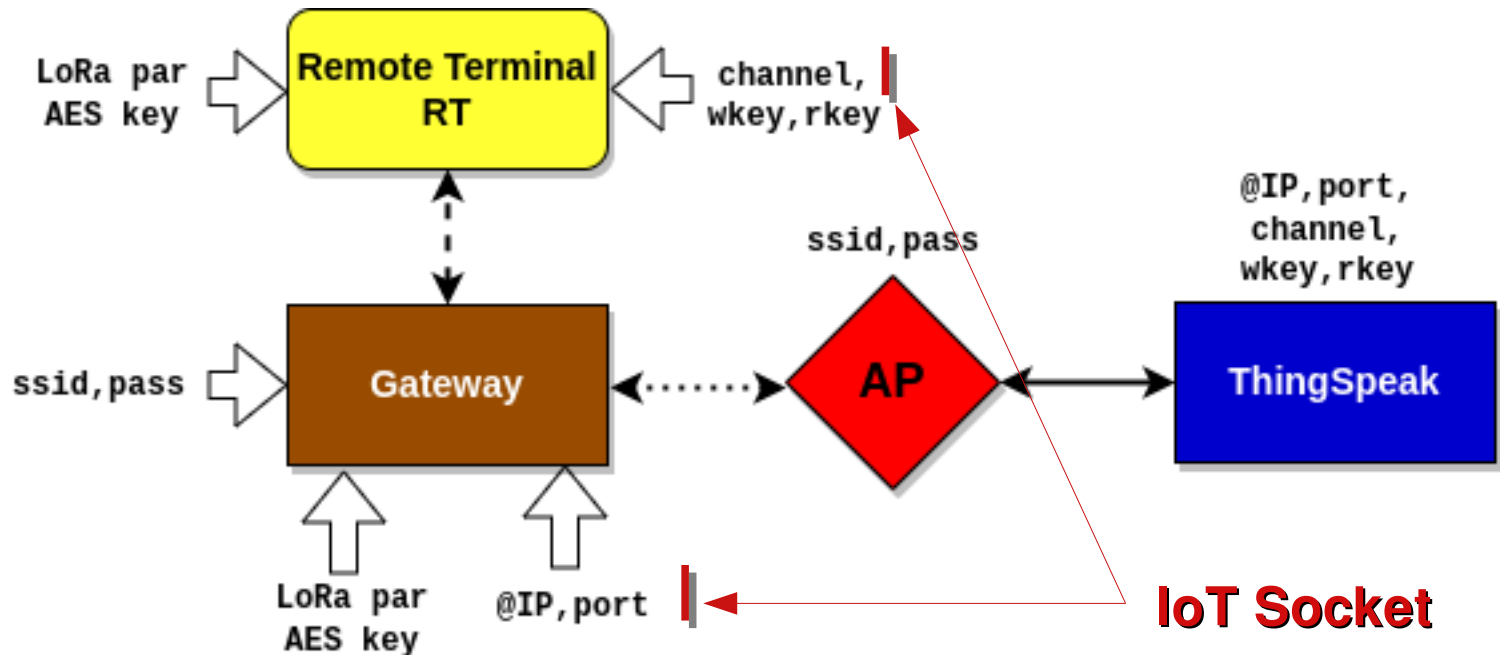


Direct Terminals know: **IP address:Service port:Channel number**

plus: write and optionally read key

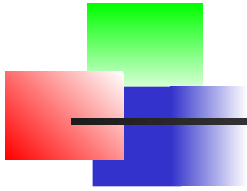
A channel contains fields (max.8) that may be interpreted as **IoT data streams to be “compressed”**.

# Remote Terminals and IoT Sockets

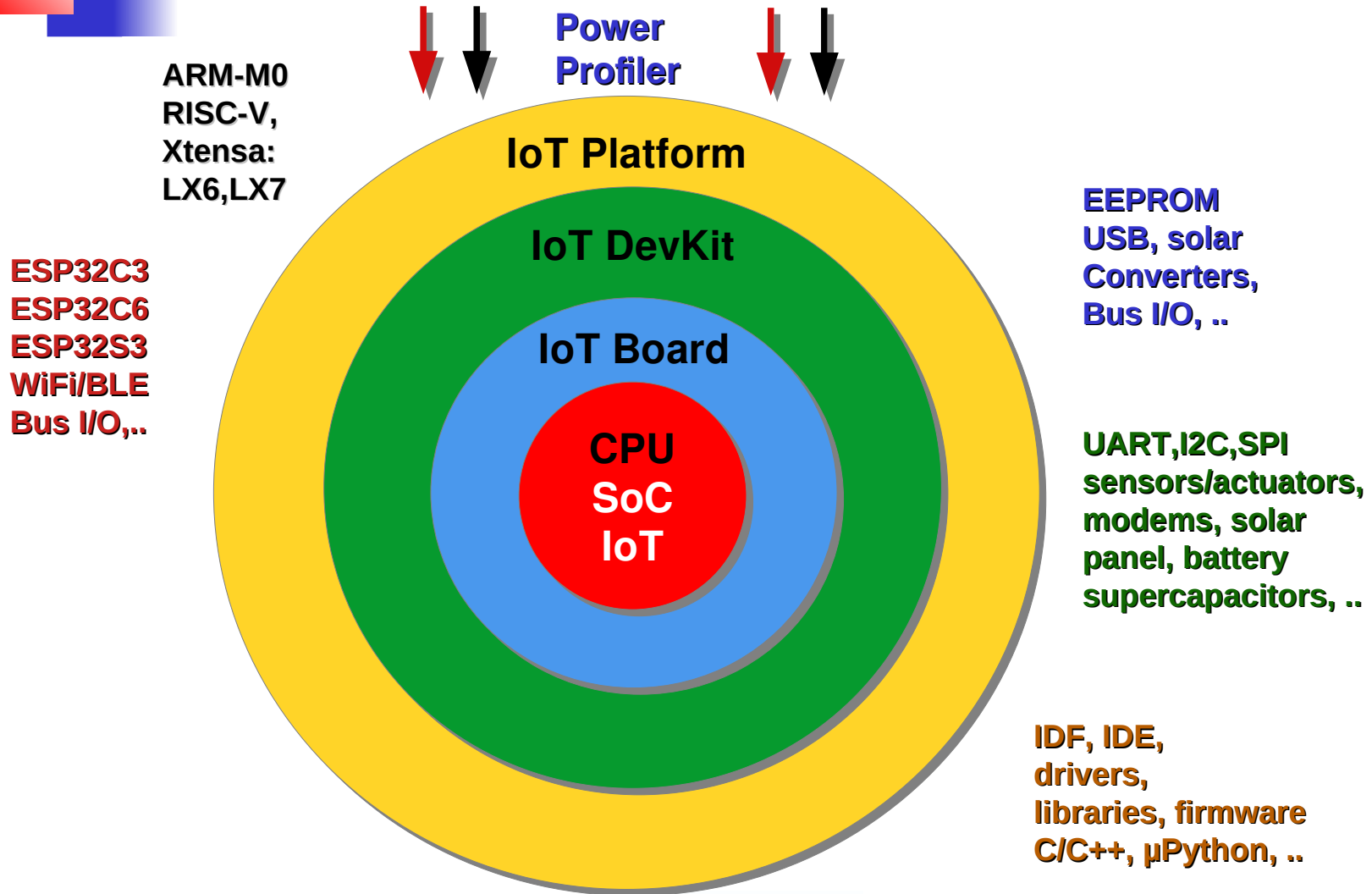


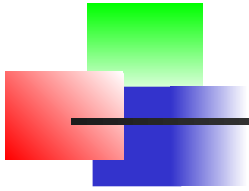
Gateways know **IP address** : **Service port**

Remote Terminals know only **Channel number** (identifier)

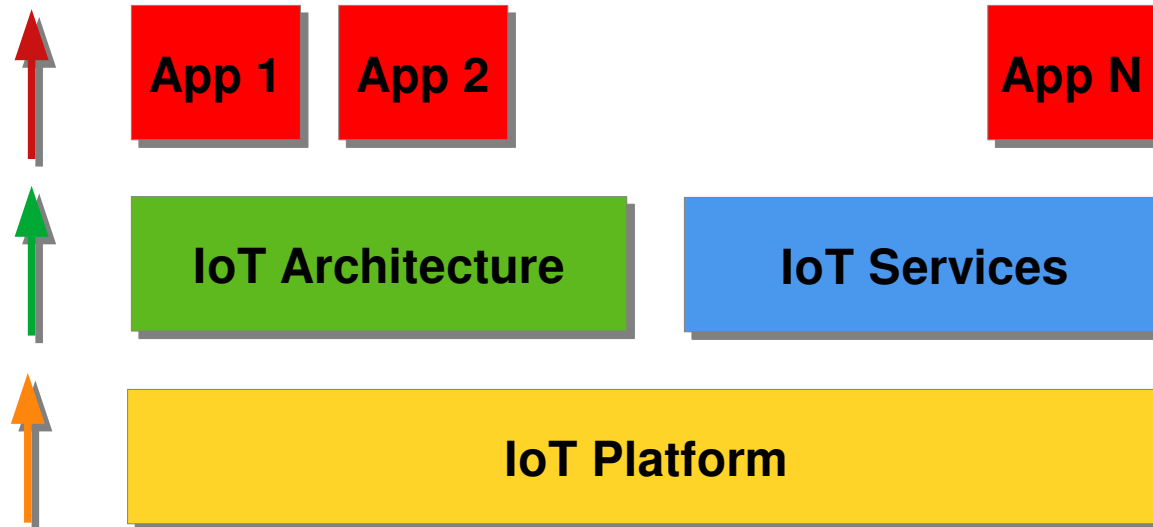


# From IoT SoC to IoT Platform





# From Platform to Application



**AI assisted** - Generation/Development process

# ESP32C3 IoT SoC

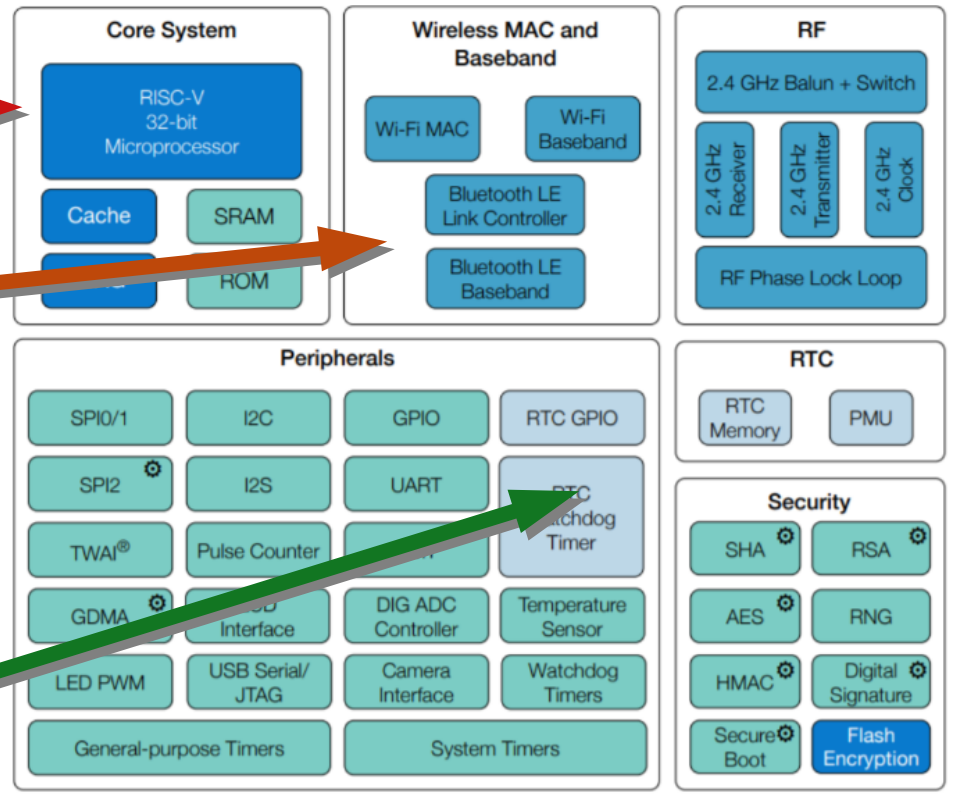
**RISC-V : 5-stages pipeline**

**Radio: WiFi, BT/BLE**

**Serial interfaces: I2C, SPI, UART, I2S, ..**

**low power : deepsleep  
RTC clock, memory, ..**

Espressif's ESP32-C3 Wi-Fi + Bluetooth® Low Energy SoC



Modules having power in specific power modes:



- Active
- Active and Modem-sleep
- Active, Modem-sleep, and Light-sleep; optional in Light-sleep
- All modes



# Heltec – ESP32C3 board

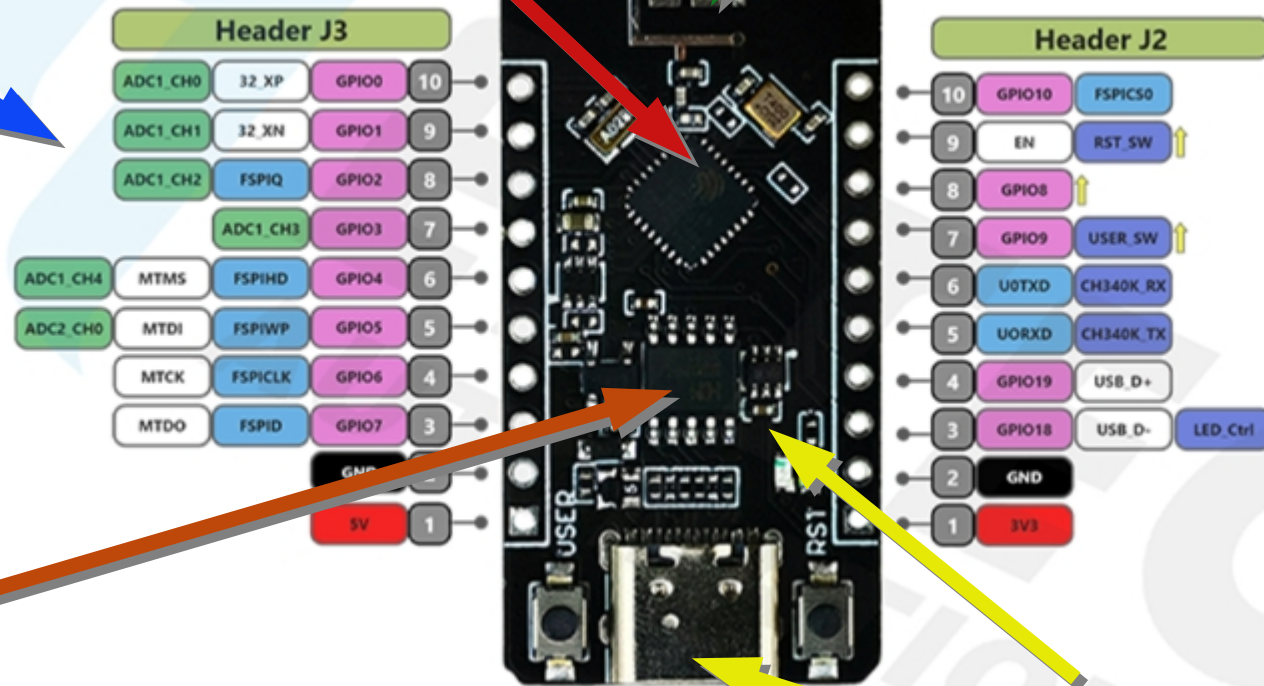
**ESP32C3 SoC**

**WiFi/BT/BLE – 2.4GHz antenna**

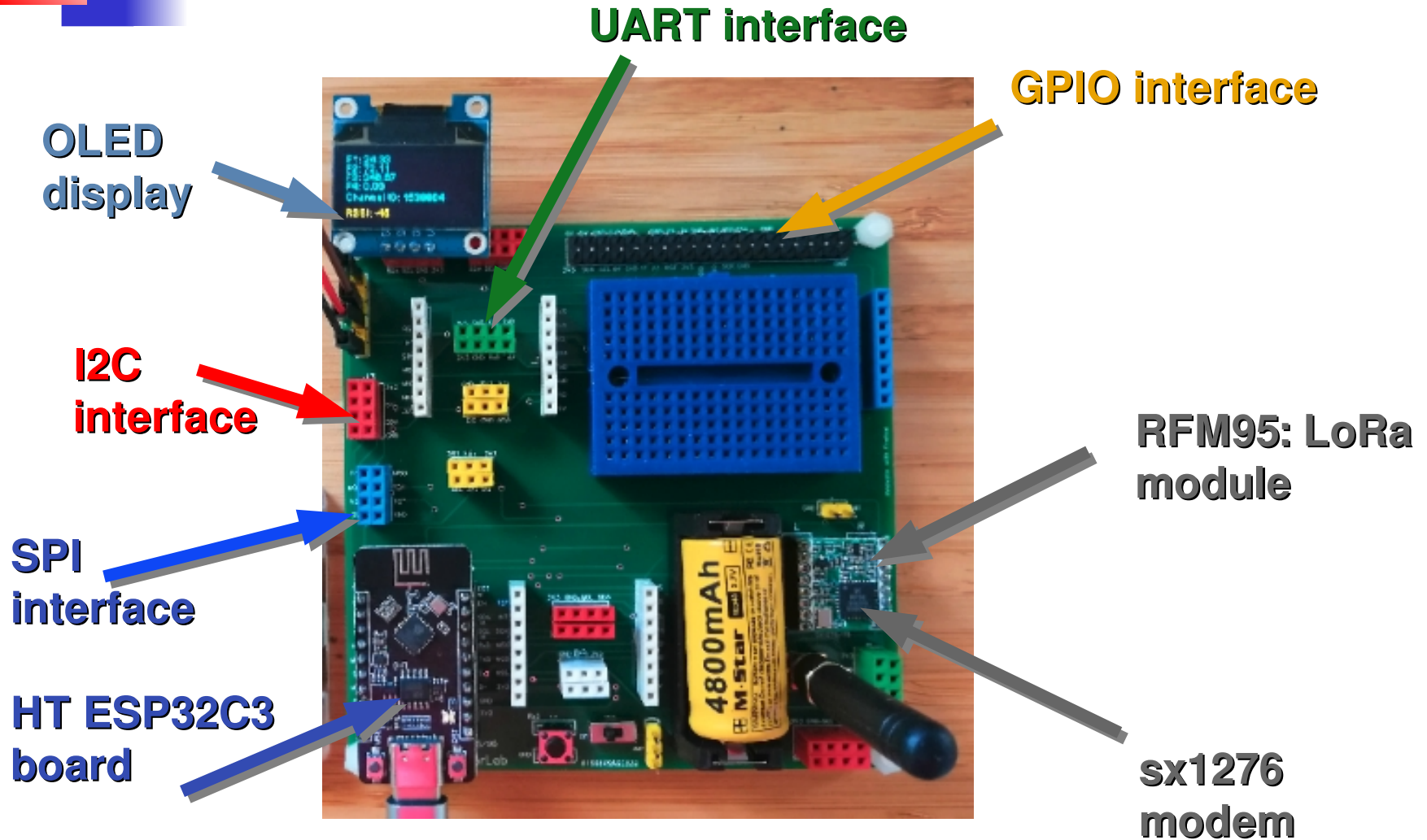
**Serial interfaces: I2C, SPI,  
UART, I2S, ..**

**EEPROM (SPI)**

**USB/UART**



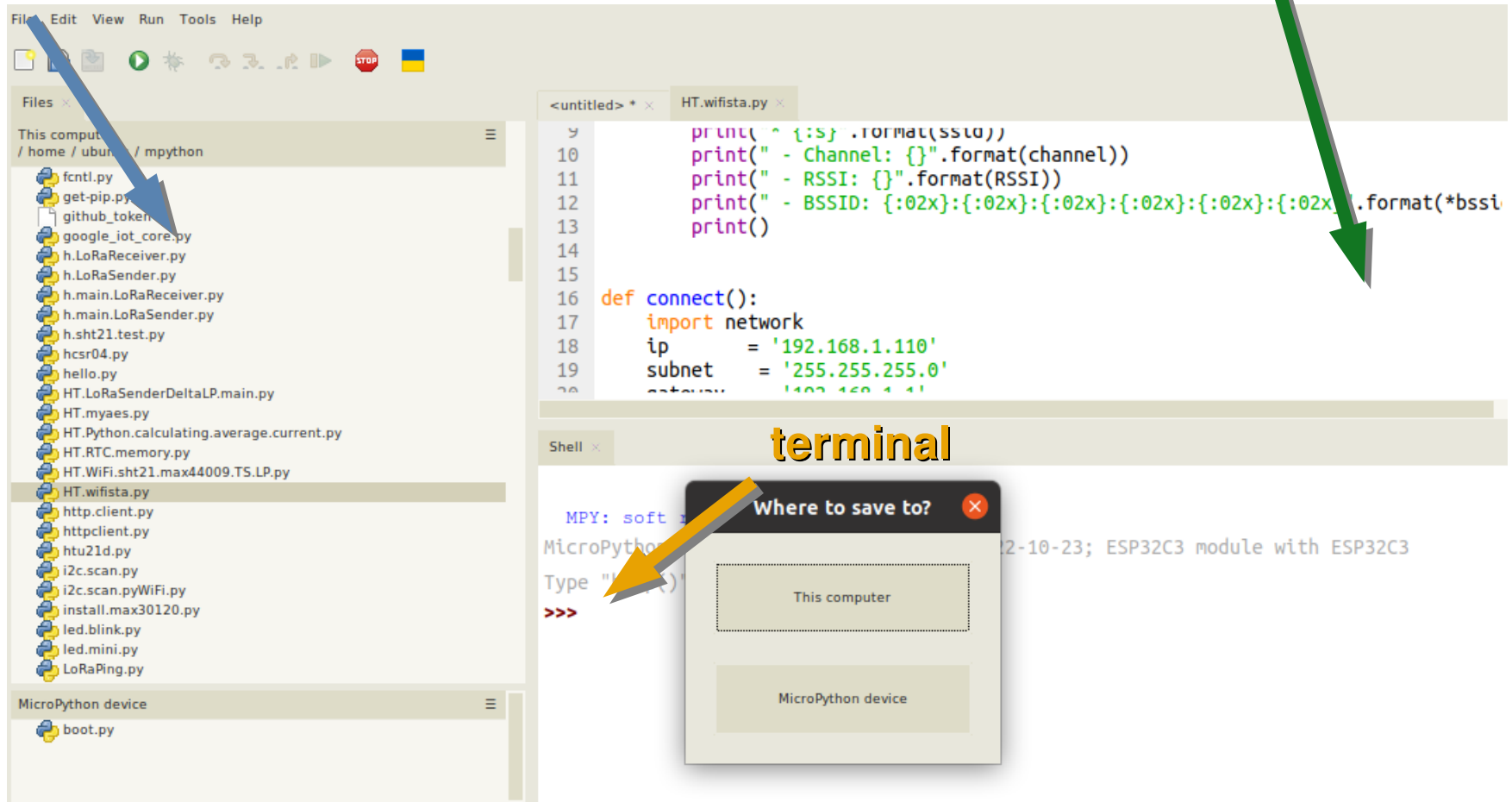
# IoT DevKit – HT ESP32C3



# Platform – Thonny IDE

editor

files



# Platform – Thonny IDE

The image shows two overlapping windows from the Thonny IDE. The background window is 'Thonny options', and the foreground window is 'Install MicroPython (esptool)'. Annotations include a blue arrow pointing to the 'MicroPython (ESP32)' dropdown in the options window, a green arrow pointing to the 'Target port' dropdown in the install window, and a red arrow pointing to the 'firmware.esp32c3.all.221023.bin' option in the install window's variant dropdown.

**Thonny options**

General Interpreter Editor Theme & Font Run & Debug Terminal Shell Assistant

Which kind of interpreter should Thonny use for running your code?

MicroPython (ESP32)

Details

Connecting via USB cable:  
Connect your device to the computer and select corresponding port below (look for your device name, "USB Serial" or "UART").  
If you can't find it, you may need to install proper USB driver first.

Connecting via WebREPL:  
If your device supports WebREPL, first connect via serial, make sure WebREPL (import webrepl\_setup), connect your computer and device to same network and use < WebREPL > below

Port or WebREPL

USB Serial @ /dev/ttyUSB0

- ☒ Interrupt working program on connect
- ☒ Synchronize device's real time clock
- ☒ Use local time in real time clock
- ☒ Restart interpreter before running a script

[Install or update MicroPython](#)

**Install MicroPython (esptool)**

Click the ≡ button to see all features and options. If you're stuck then check the variant's 'info' page for details or ask in MicroPython forum.

NB! Some boards need to be put into a special mode before they can be managed here (e.g. by holding the BOOT button while plugging in). Some require hard reset after installing.

You may need to tweak the install options (≡) if the selected MicroPython variant doesn't match your device precisely. For example, you may need to set flash-mode to 'dio' or flash-size to 'detect'.

Target port: USB Serial @ /dev/ttyUSB0

☒ Erase all flash before installing (not just the write area)

MicroPython family: ESP32-C3

variant: <local file>

version: firmware.esp32c3.all.221023.bin

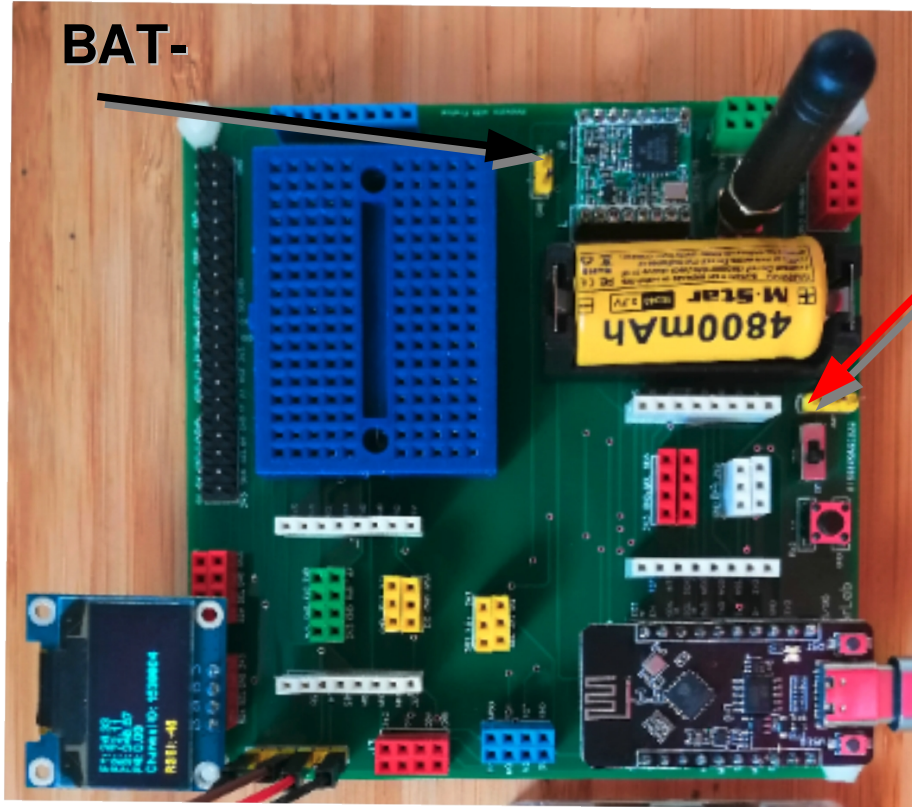
info: </media/ubuntu/0969-2ABE/microPython.firmware/esp32c3>

☐ Writing at 0x00105112... (68 %)

**Annotations:**

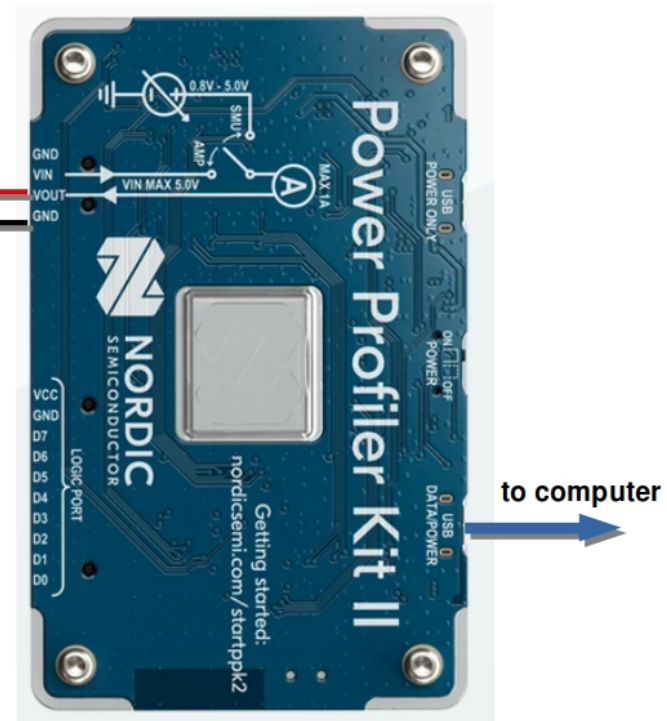
- Blue arrow: **μPython interpreter type**
- Green arrow: **port to flash**
- Red arrow: **firmware to flash**

# Power Profiler Kit 2 (PPK2)



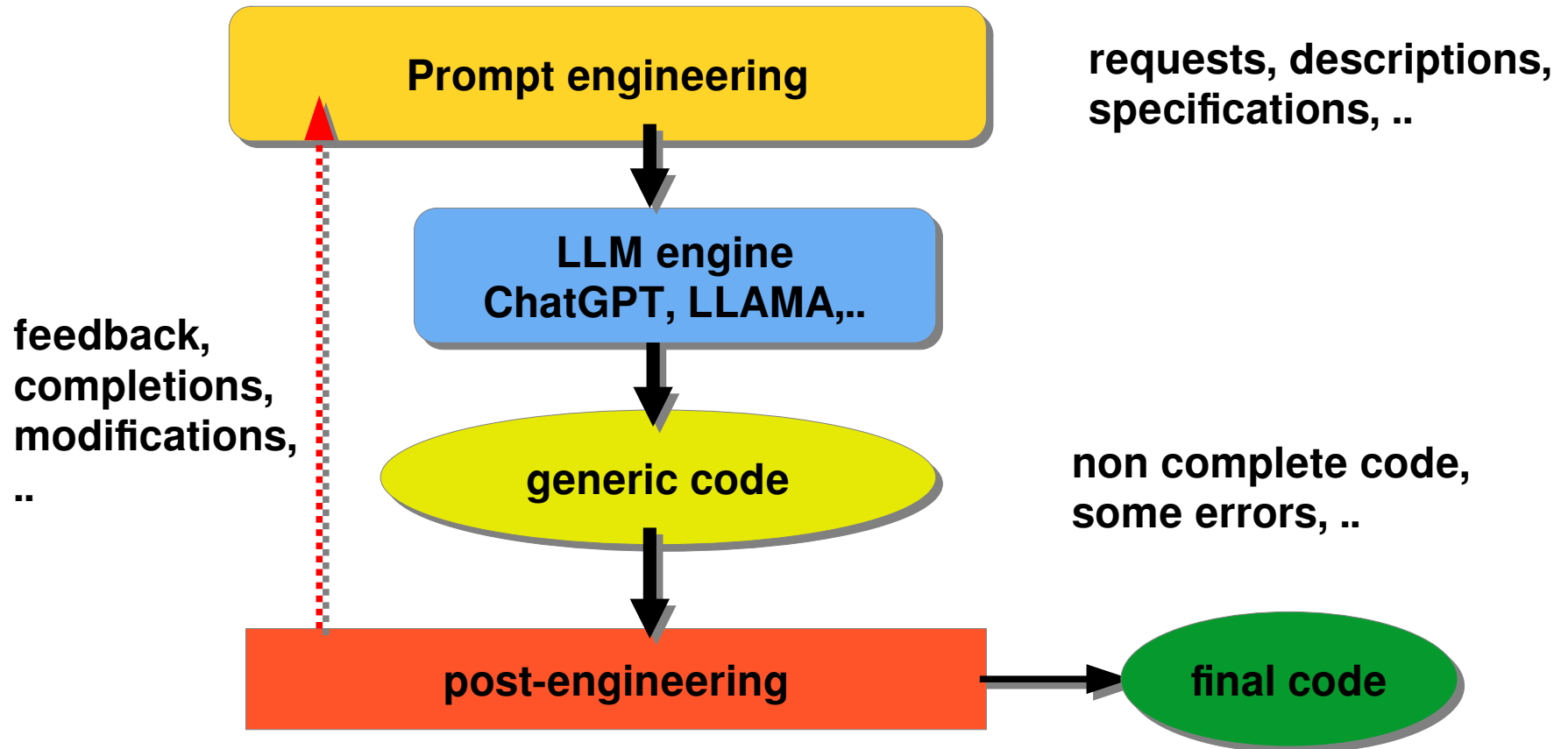
**BAT+**

to DevKit  
to BAT+  
to BAT-  
or GND





# Chat GPT development assistance

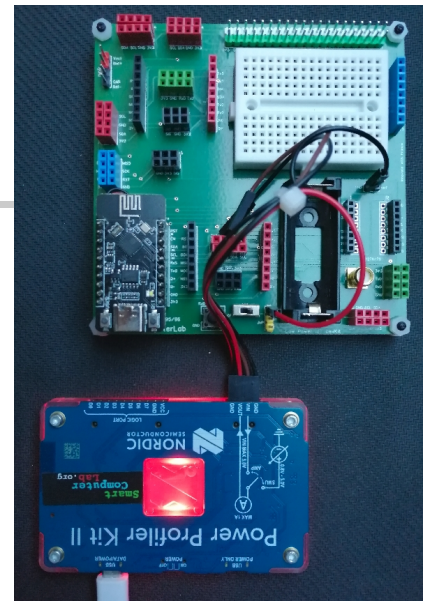


# Simple example

## Prompt:

Write MicroPython code for ESP32 to switch on the signal (buzzer) on Pin 3 for 3 seconds. Then switch off the signal and go to deepsleep for 6 seconds.

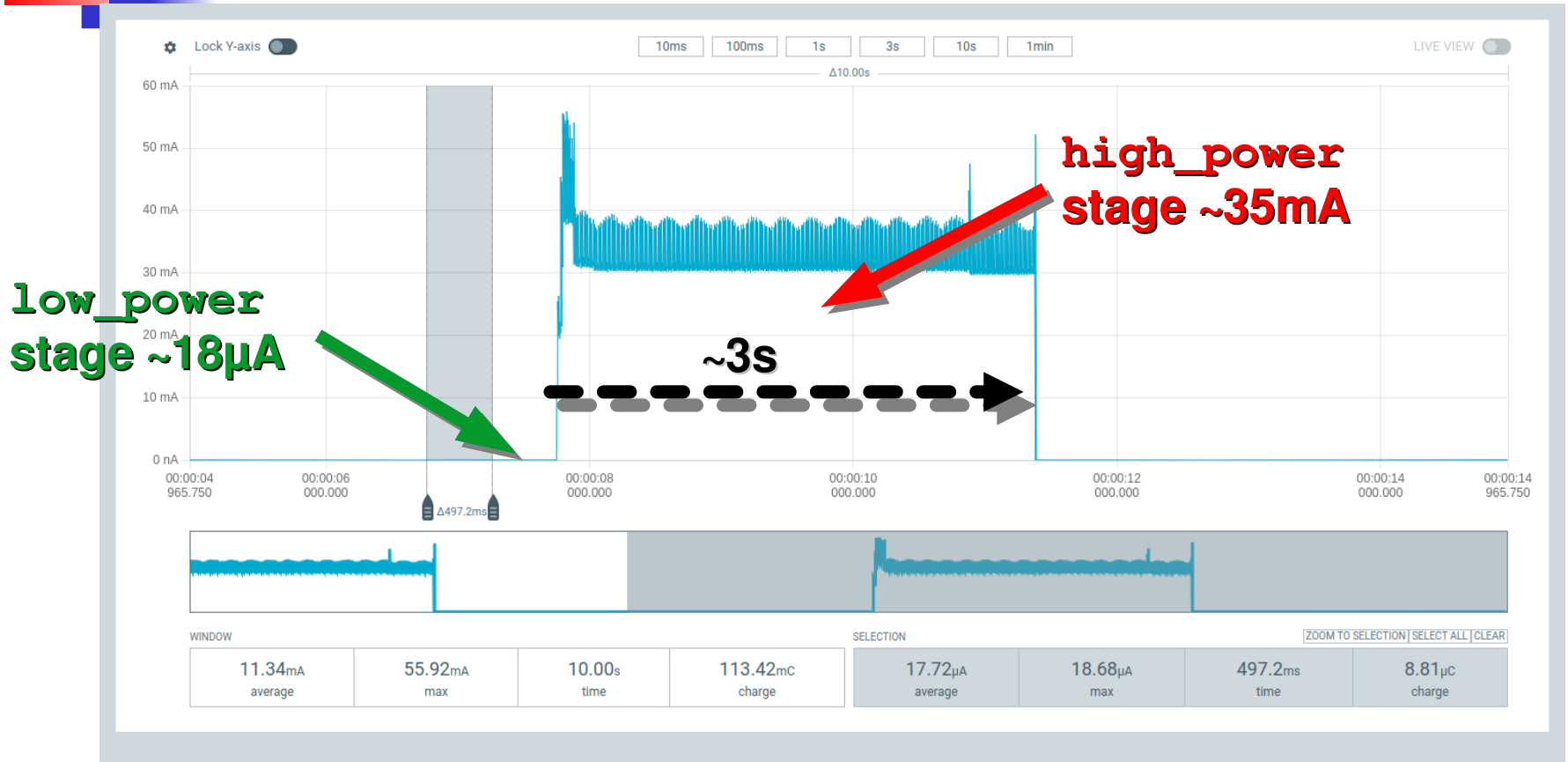
```
from machine import Pin, deepsleep
import time
# Initialize the SIG pin
led = Pin(3, Pin.OUT)
# Turn on the SIG
led.value(1)
print("SIG is ON")
time.sleep(3)
# Turn off the LED
led.value(0)
print("SIG is OFF")
# Go to deep sleep for 6 seconds
print("Going to sleep for 10 seconds...")
time.sleep(0.2)      # added to see the print
deepsleep(6000)      # 6 seconds in milliseconds
```



## Remark:

The code is correct, no post-engineering and/or feedback

# Simple example – PPK2 diagram



Calculate the average current for a cycle with 600s low\_power stage ?

$$(18 \times 600 + 35000 \times 3) / (600 + 3) = 192 \mu A$$





# IoT Labs – Session 1

---

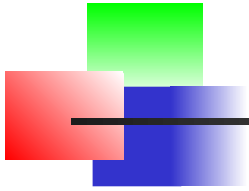
## Lab 1: Working with sensors and displays (I2C bus)

### Example Prompts

#### Prompt:

**With ESP32 that is programmed with MicroPython we use frequently I2C bus to connect the sensors. Explain the operations of I2C bus.**

```
from machine import Pin, I2C
# Configure I2C
i2c = I2C(0, scl=Pin(9), sda=Pin(8)) # I2C bus 0 with custom pins
# Scan for I2C devices
devices = i2c.scan()
print("I2C devices found:", devices)
# Assuming a device is found at address 0x3C
device_address = 0x3C
# Write to the device
data_to_write = bytearray([0x00, 0xFF]) # Example data
i2c.writeto(device_address, data_to_write)
# Read from the device
num_bytes = 2
data_read = i2c.readfrom(device_address, num_bytes)
print("Data read:", data_read)
```



# IoT Labs – Session 1

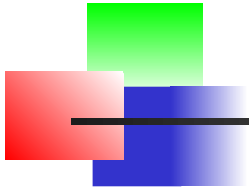
---

## Lab 1: Working with sensors and displays (I2C bus)

### Example Prompts

#### **Prompt:**

**Write for ESP32 in MicroPython the function and module sensors.py with drivers for max44009 (luminosity) and sht21 (temperature, humidity) sensors operating over I2C bus. The function takes two arguments : sda, scl to specify the pins of I2C bus. It returns 3 values: luminosity,temperature,humidity=sensors(sda,scl).**



# IoT Labs – Session 1

---

## Lab 2: Communicating with WiFi and IoT servers

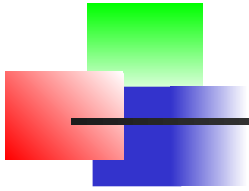
### Example Prompts

**Prompt:**

**Write in Micropython for ESP32 a module (`wifi_tools.py`) with 3 functions related to WiFi: `connect_WiFi(ssid,pass)`, `disconnect_WiFi()` and `scan_WiFi()`.**

**Prompt:**

**Write MicroPython code - function for ESP32 to get WiFi configuration: IP, port, .. after a successful connection to access point.**



# IoT Labs – Session 1 – Lab 2

---

**Prompt:**

**Write MicroPython programs for ESP32 to send and receive simple UDP datagrams with socket() functions**

**Prompt:**

**Explain the function and the use of NTP protocol and server.**

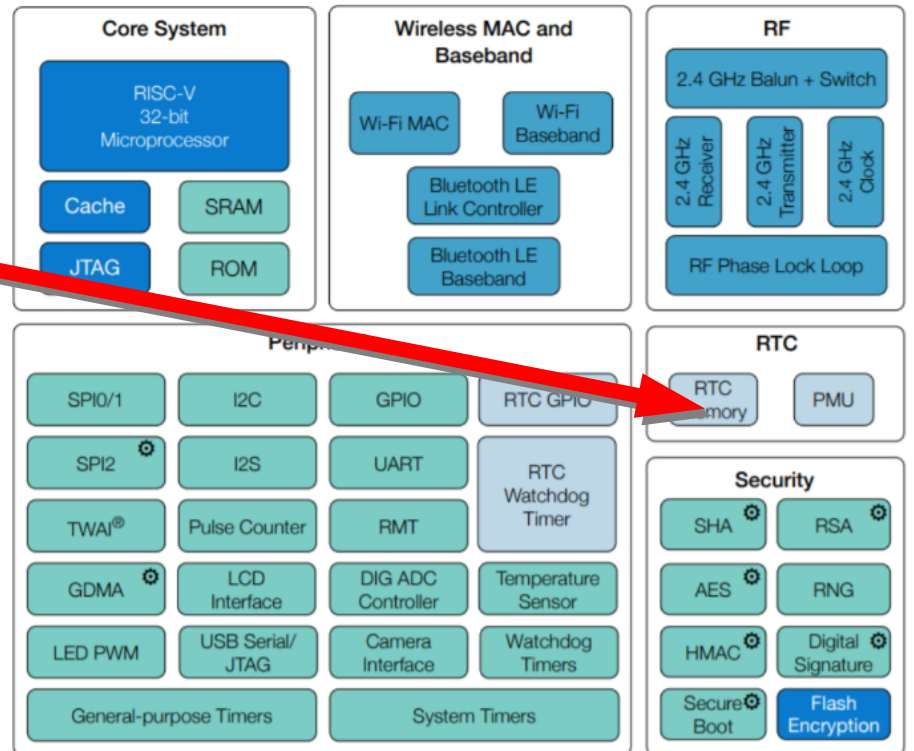
**Prompt:**

**Write micropython code for ESP32 using the generated wifi\_tools.py module to read time from an external NTP server. Read the time in a 10 second infinite cycle. Note that the time must be synchronized only in the first cycle. Print the values of hour, minute, and second.**

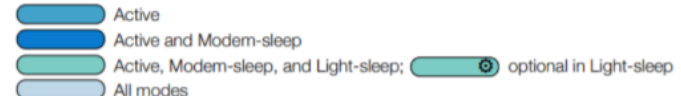
# IoT Labs – Session 1 – Lab 2

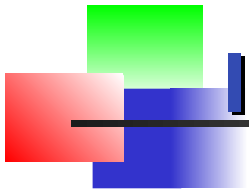
**Prompt:**  
Give simple example in micropython for ESP32 of the usage of **RTC memory**. The example provides a counter of the cycles built with deepsleep mode for 6 seconds. Print the value of the counter in each cycle.

Espressif's ESP32-C3 Wi-Fi + Bluetooth® Low Energy SoC



Modules having power in specific power modes:





# IoT Labs – Session 1 – Lab 3

**Prompt:**

**Write MicroPython code for ESP32 to connect to WiFi Access Point and to send (**publish**) **MQTT** message. The message contains readings from MAX44009 and SHT21 with `sensors.py` module. Use previously created `wifi_tools.py` module to connect to WiFi access point.**

**Prompt:**

**Write MicroPython code for ESP32 to send several sensor data values to ThingSpeak server. Use MAX44009 and SHT21 sensors – `sensors.py`. Use `wifi_tools.py` module predefined previously to connect to the WiFi and **ThingSpeak.com** server.**

# IoT Labs – Session 1 – Lab 2

ThingSpeak™

Channels ▾ Apps ▾ Devices ▾ Support ▾

ThingSpeak™

Channels ▾ Apps ▾ Devices ▾ Support ▾

## My Channels

New Channel

Search by tag

Name ↕	Created ↕	Updated ↕
Smart IoT 1 one, smartiotlabs <div>Private Public Settings Sharing API Keys Data Import / Export</div>	2021-10-16	2024-03-15
Smart IoT 2 smartiotlabs, two <div>Private Public Settings Sh</div>	2022-01-05	2024-03-17

## Smart IoT 2

Channel ID: 1626377  
Author: mwa0000024358098  
Access: Public

This channel is  
number SmartI  
 smartiotlab

Private View Public View Channel Settings Sharing API Keys

## Write API Key

Key 3IN09682SQX3PT4Z

Generate New Write API Key

## Smart IoT 2

Channel ID: 1626377  
Author: mwa0000024358098  
Access: Public

This channel is created to for  
number SmartIoTBox  
 smartiotlabs, two

Private View Public View Channel Settings Sharing API Keys Data Import

## Channel Settings

Percentage Complete 70%

Channel ID 1626377

Name Smart IoT 2

Description This channel is created to for Smart IoT DevKits. each  
terminal is identified by channel number SmartIoTBox

Field 1 temperature ☒

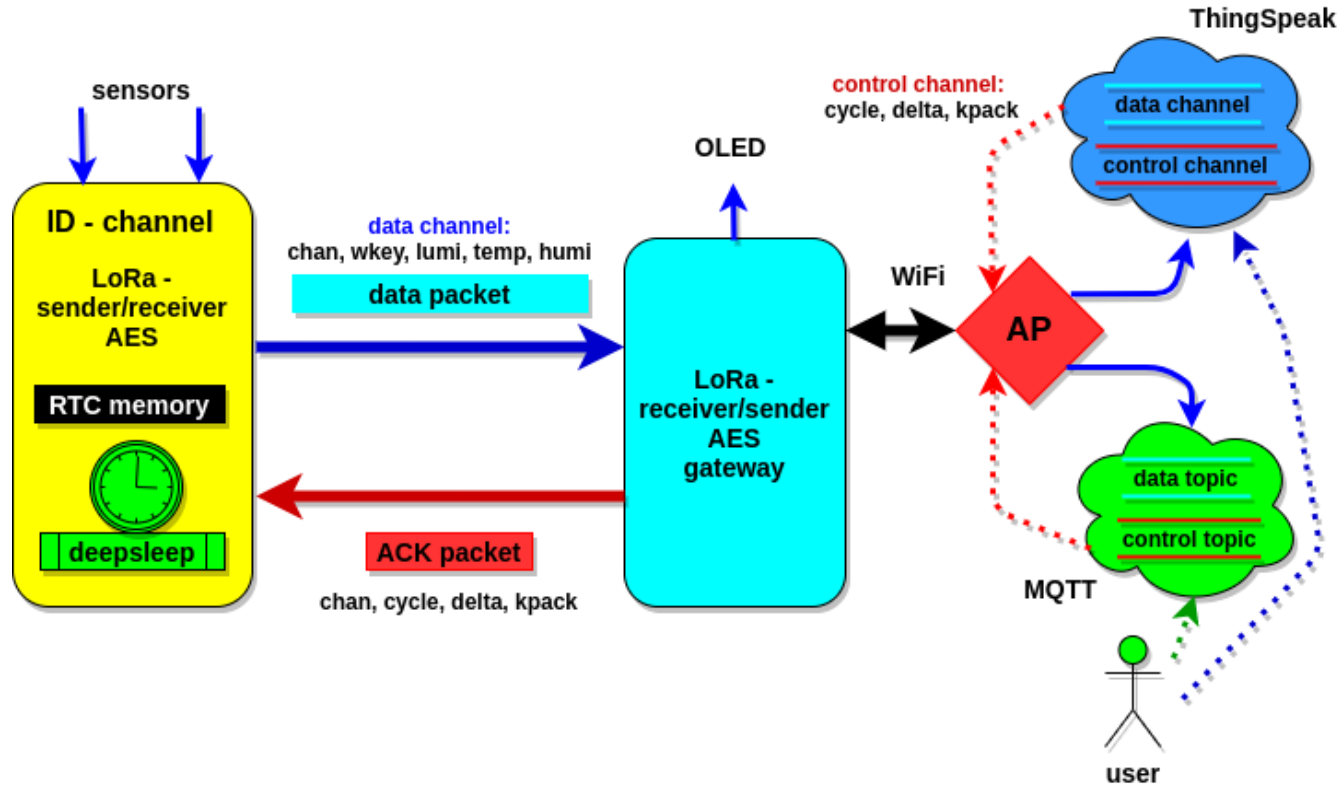
Field 2 humidity ☒

Field 3 luminosity ☒

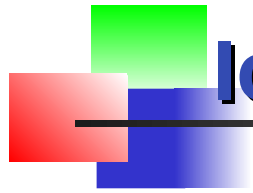
# IoT Labs – Session 1 – Lab 3 & Lab 4

Lab 3: Building Close Terminals and Gateways with MAC-WiFi

Lab 4: Long distance communication with Remote Terminals over LoRa radio links







# **IoT Labs – Session 2 - Lab 5 + Project**

---

## **Lab 5 : AIoT - EDGE**

### **Projects and Applications with advanced IoT boards**