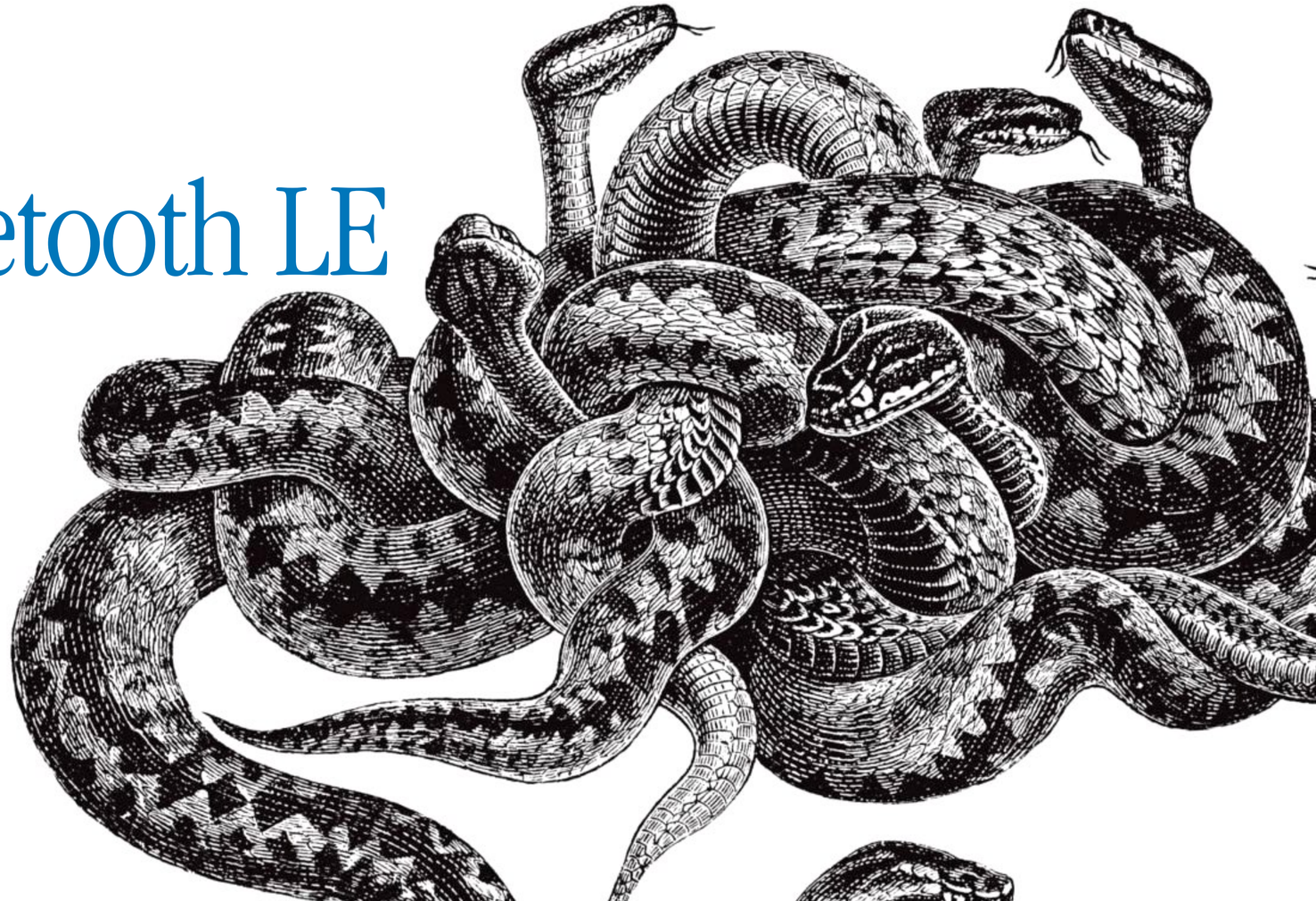
FermoLUG
linuxfm.org

Non solo blink: MicroPython e Bluetooth LE



Giacomo Vallorani
25 Ottobre 2025

Su di me

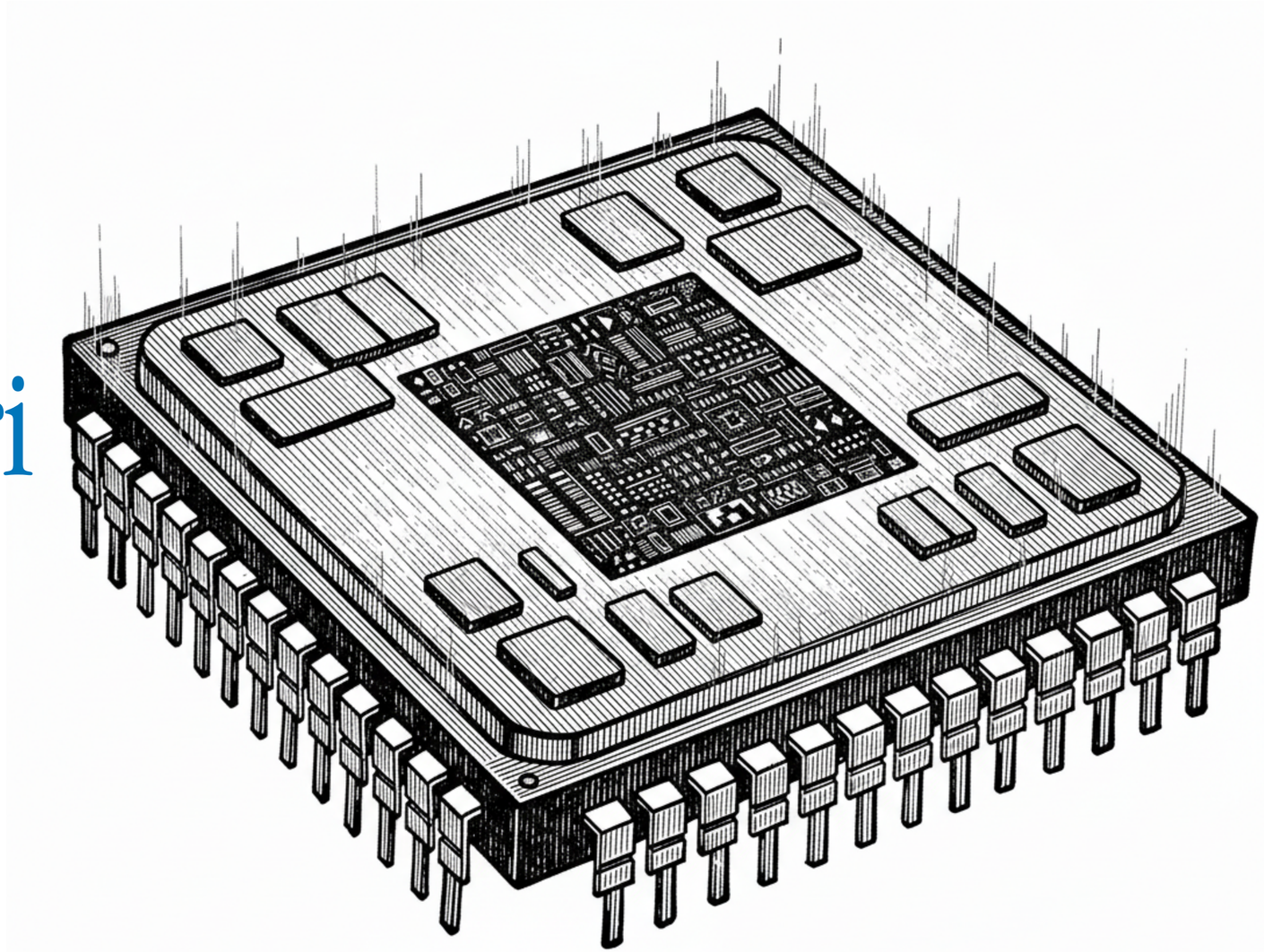
Giacomo Vallorani

- Backend Software Engineer @ PagoPA
- Maker nel tempo libero
- Sistemi distribuiti e IoT

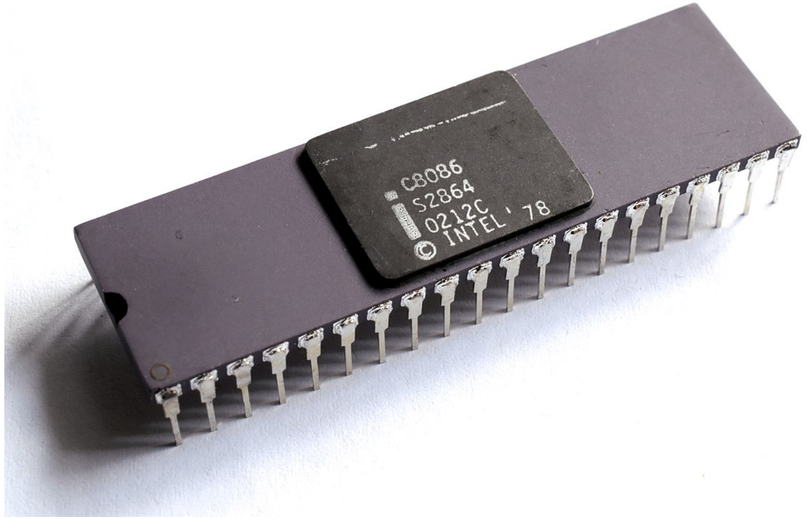
@Vallasc

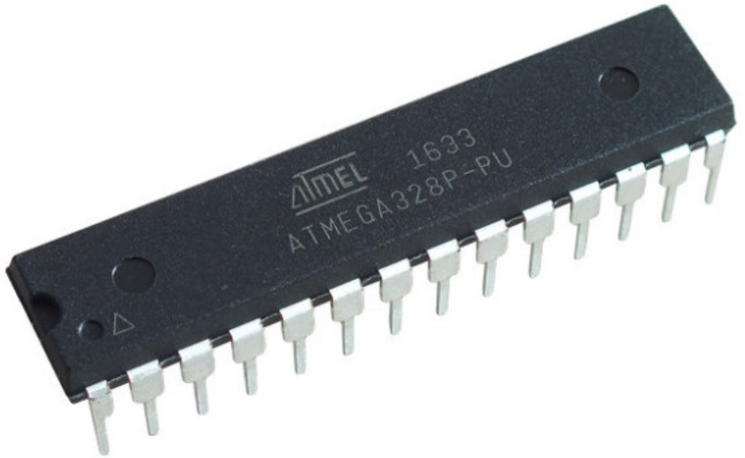
giacomovallorani

1 Programmazione di microcontrollori

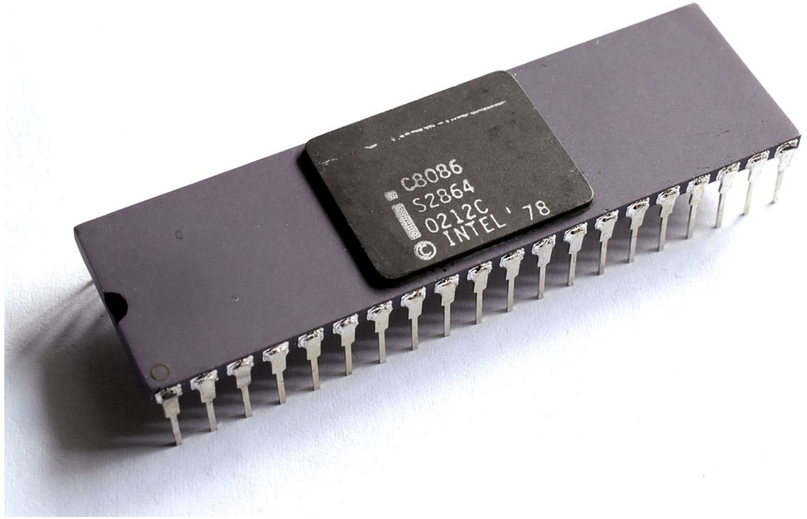


Microprocessore vs Microcontrollore

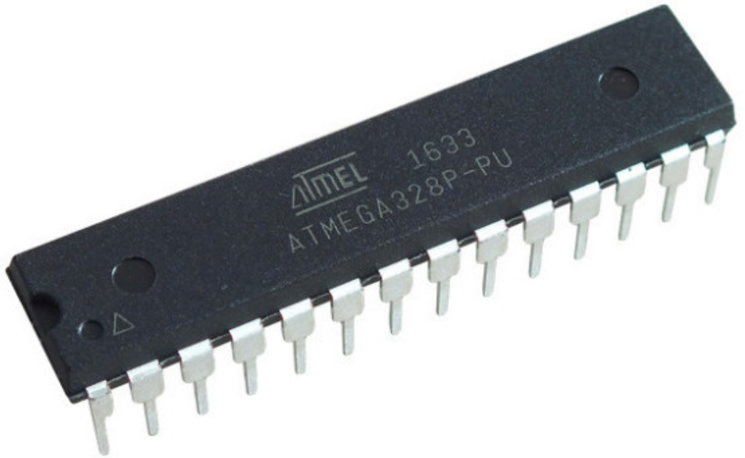




Microprocessore vs Microcontrollore

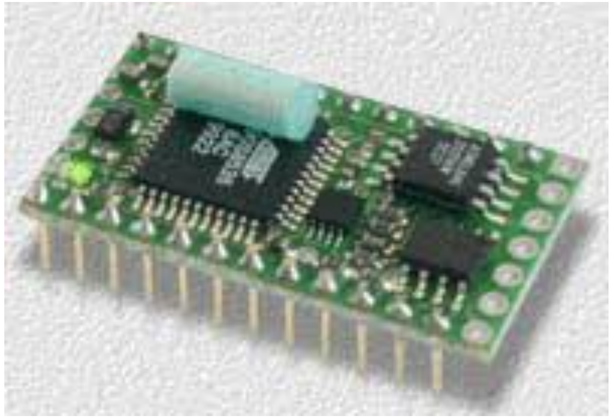


- Solo CPU
- Multitasking
- Consumi in Watt
- Costoso

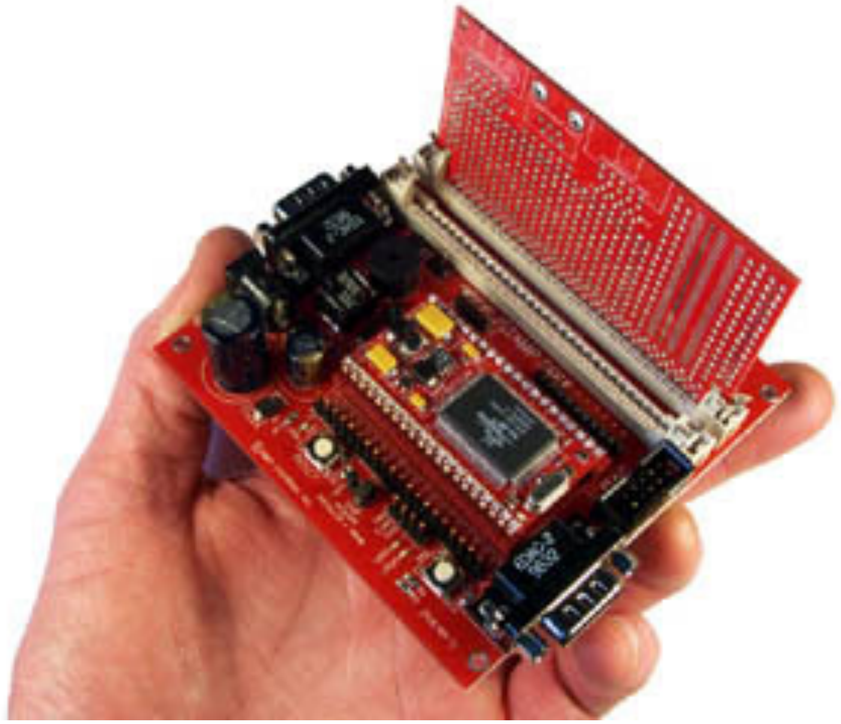


- CPU + memoria + periferiche
- Single task
- Consumi in Milliwatt
- Economico

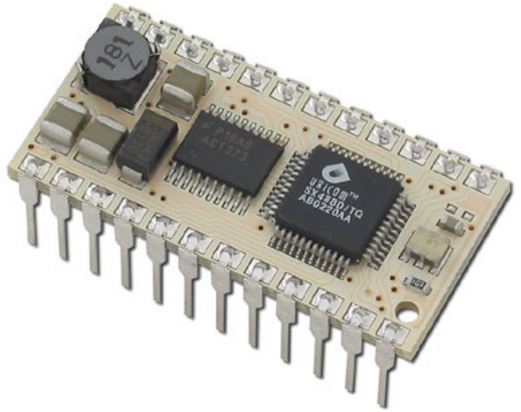
Il mondo prima di Arduino



Netmedia BasicX-24

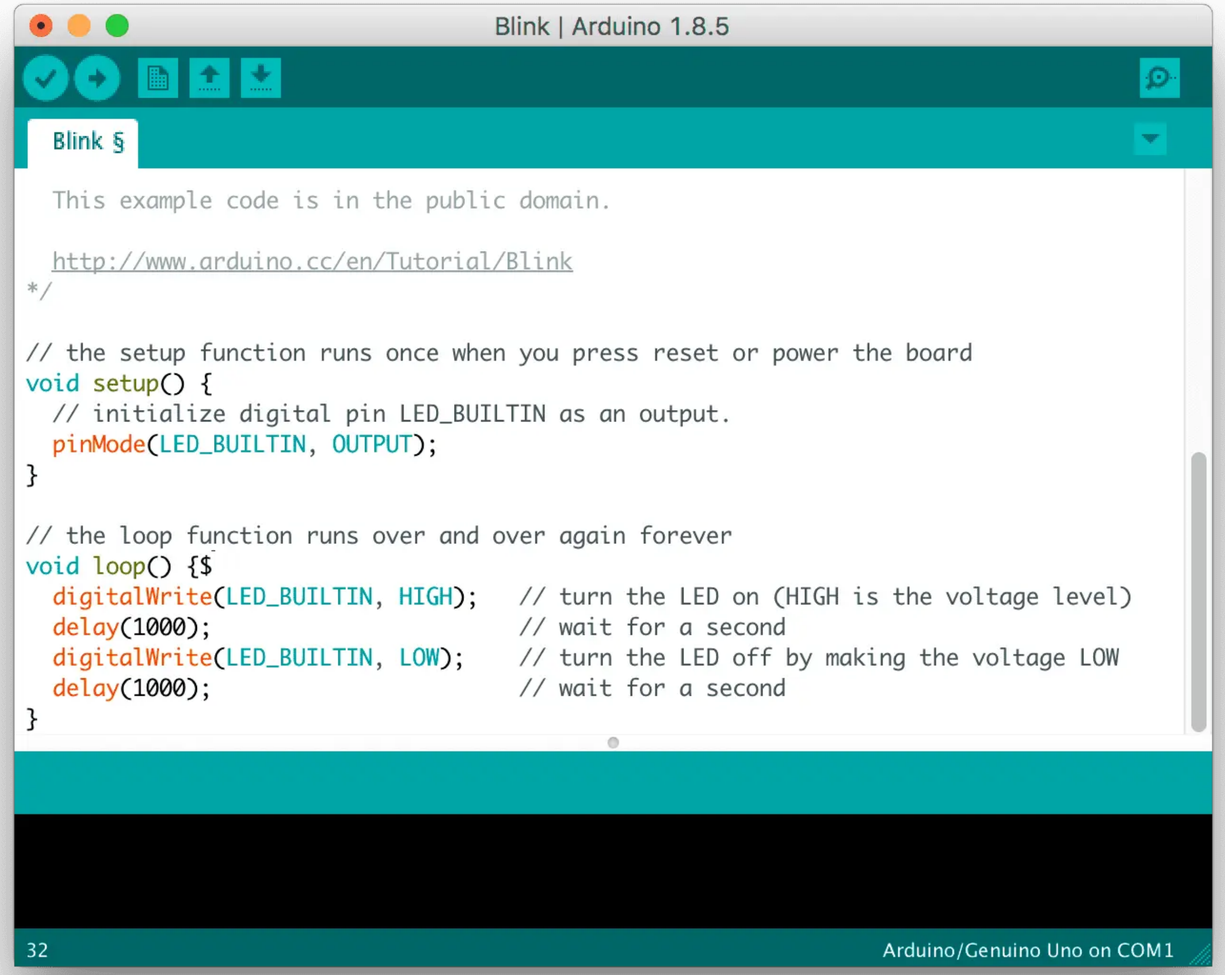


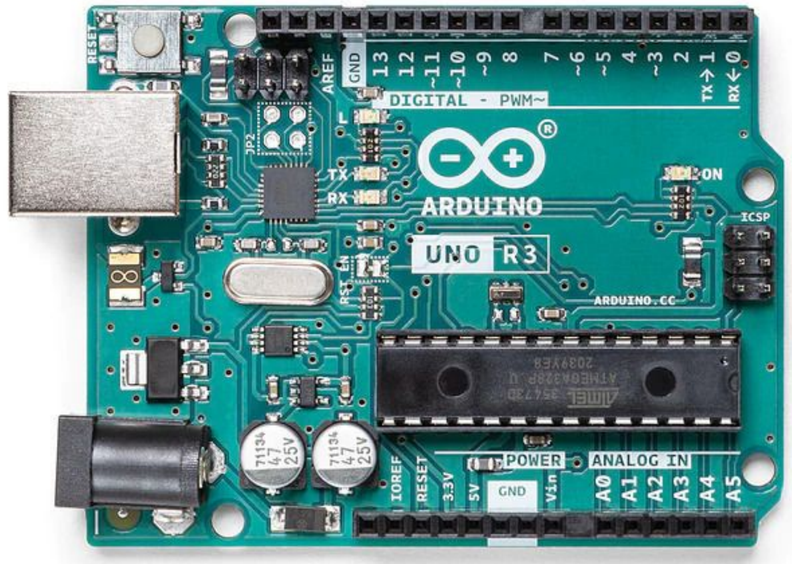
Systronix JStamp



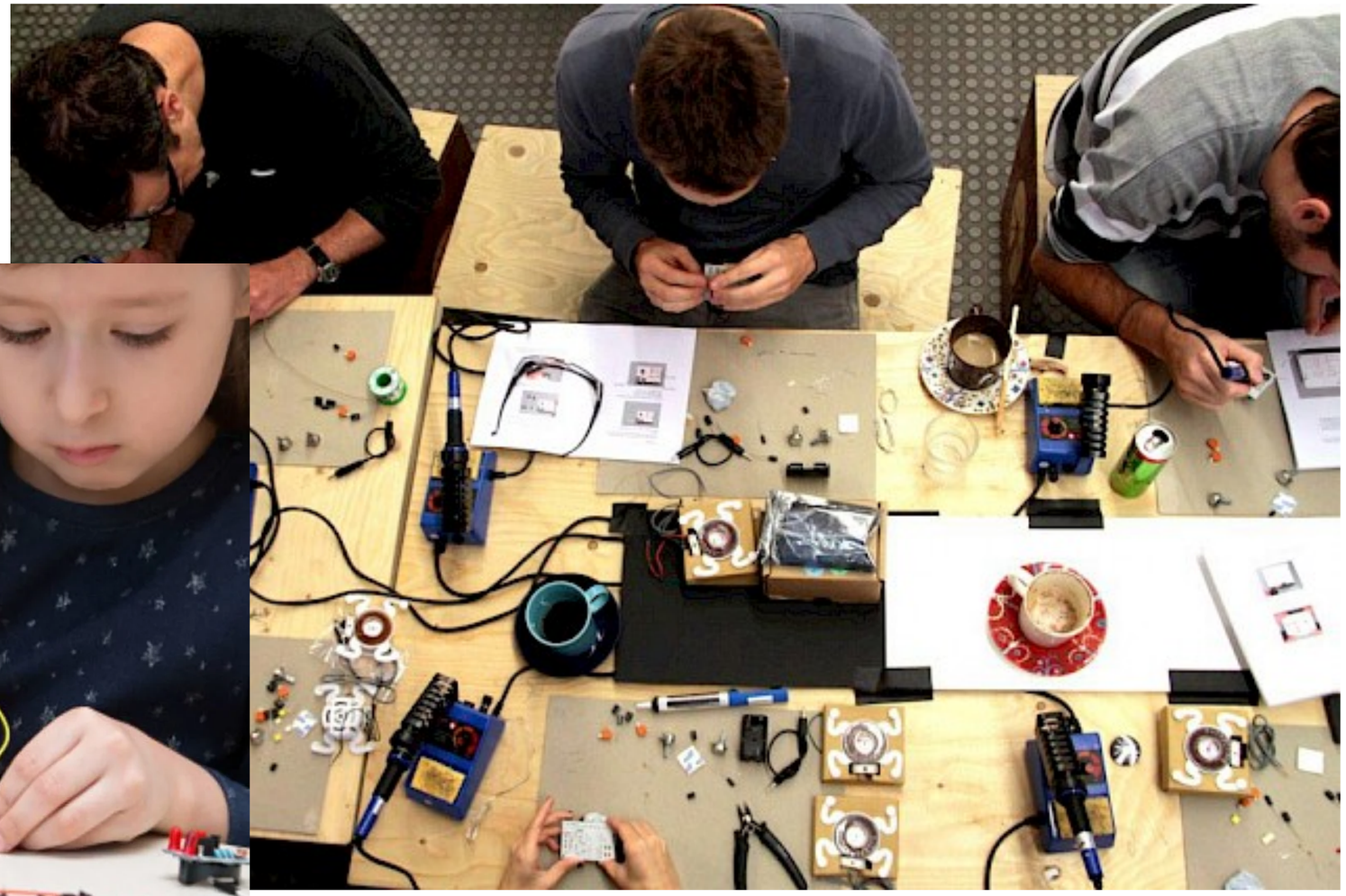
Parallax Javelin Stamp

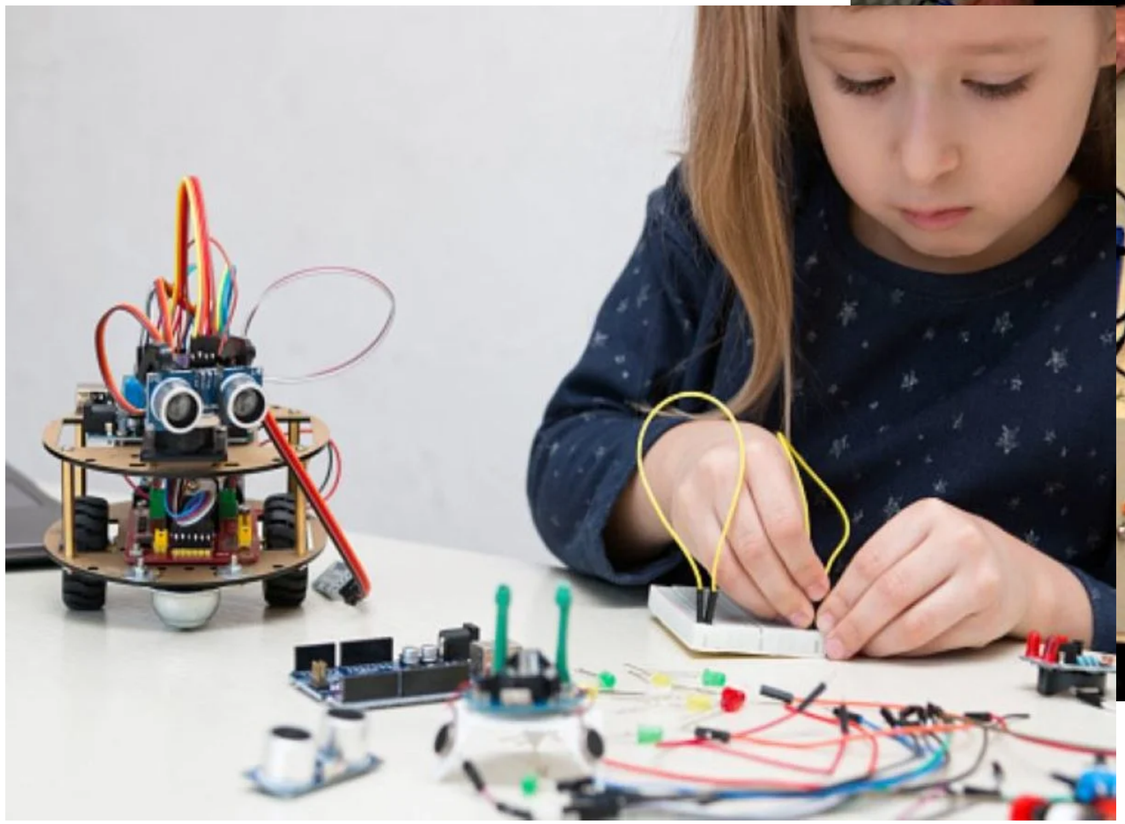
La democratizzazione dell'hardware





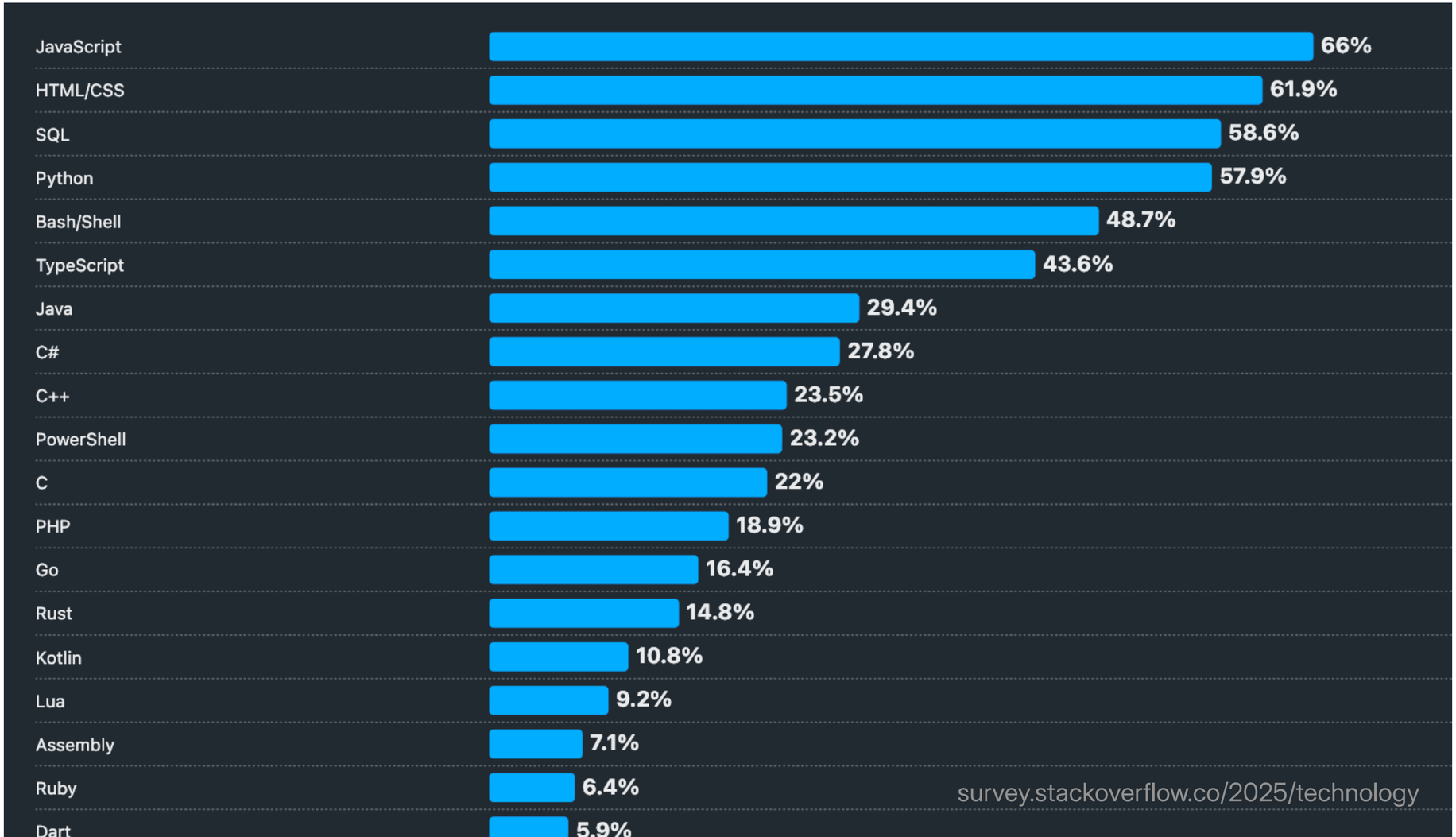
Makers



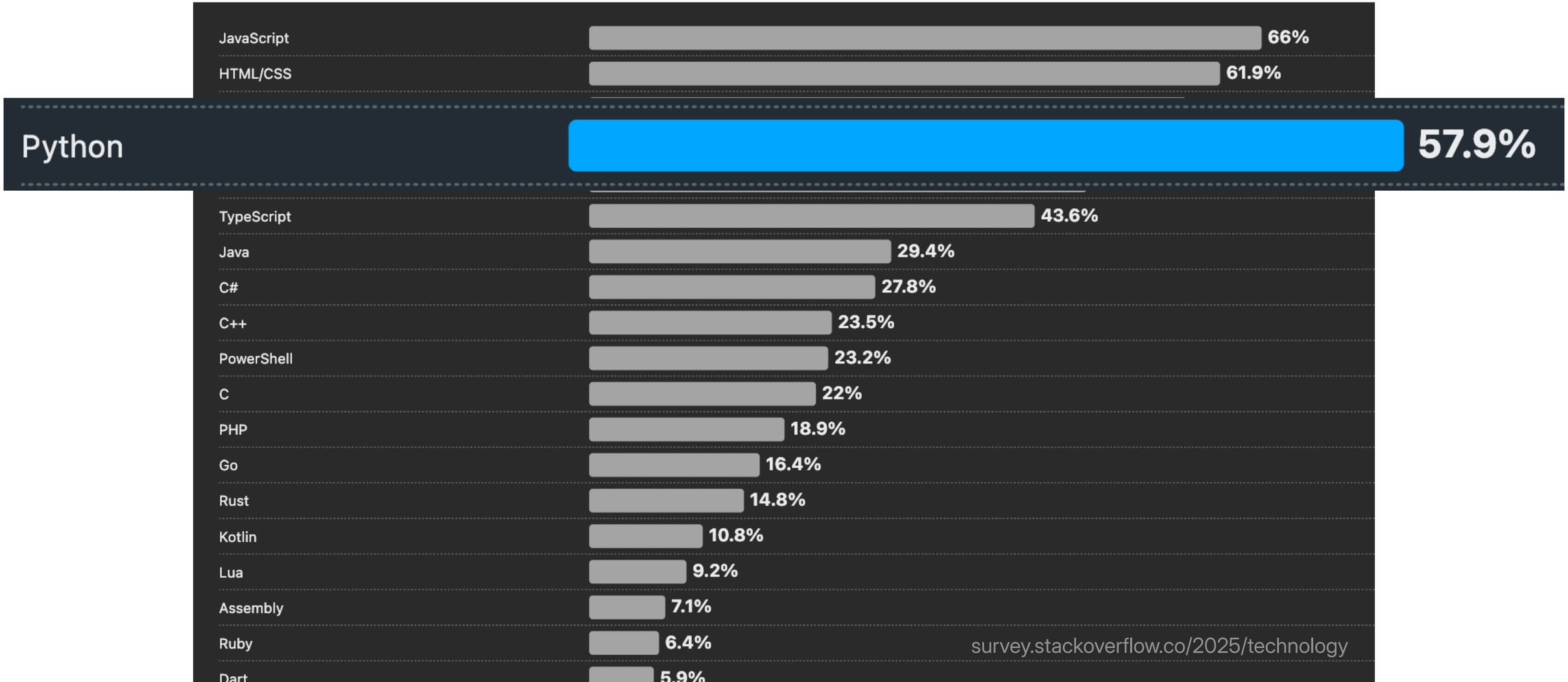


2 Perché (Micro)Python?

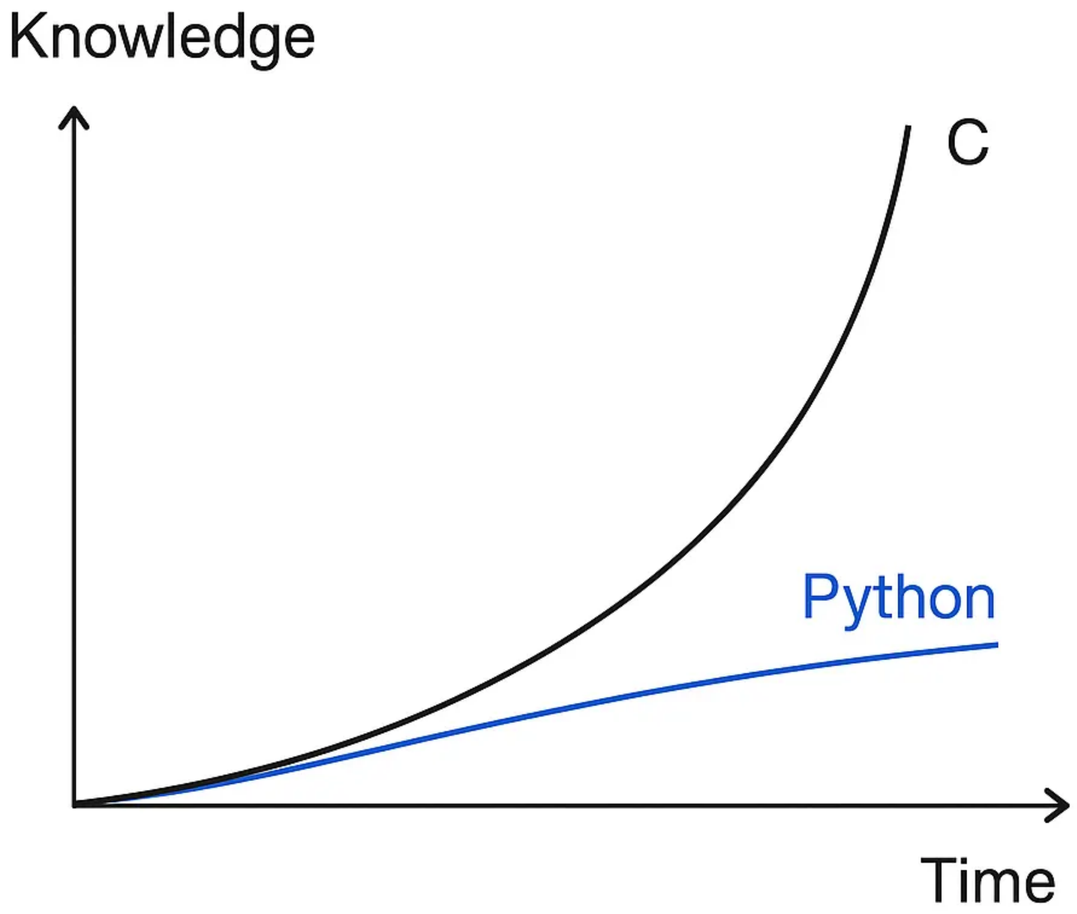
Adozione


JavaScript 66%
HTML/CSS 61.9%
SQL 58.6%
Python 57.9%
Bash/Shell 48.7%
TypeScript 43.6%
Java 29.4%
C# 27.8%
C++ 23.5%
PowerShell 23.2%
C 22%
PHP 18.9%
Go 16.4%
Rust 14.8%
Kotlin 10.8%
Lua 9.2%
Assembly 7.1%
Ruby 6.4%
Dart 5.9%
survey.stackoverflow.co/2025/technology

Adozione


JavaScript 66%
HTML/CSS 61.9%
Python 57.9%
TypeScript 43.6%
Java 29.4%
C# 27.8%
C++ 23.5%
PowerShell 23.2%
C 22%
PHP 18.9%
Go 16.4%
Rust 14.8%
Kotlin 10.8%
Lua 9.2%
Assembly 7.1%
Ruby 6.4%
Dart 5.9%
survey.stackoverflow.co/2025/technology

Curva di apprendimento



~~Puntatori~~

~~Gestione della memoria~~

~~Compilazione~~

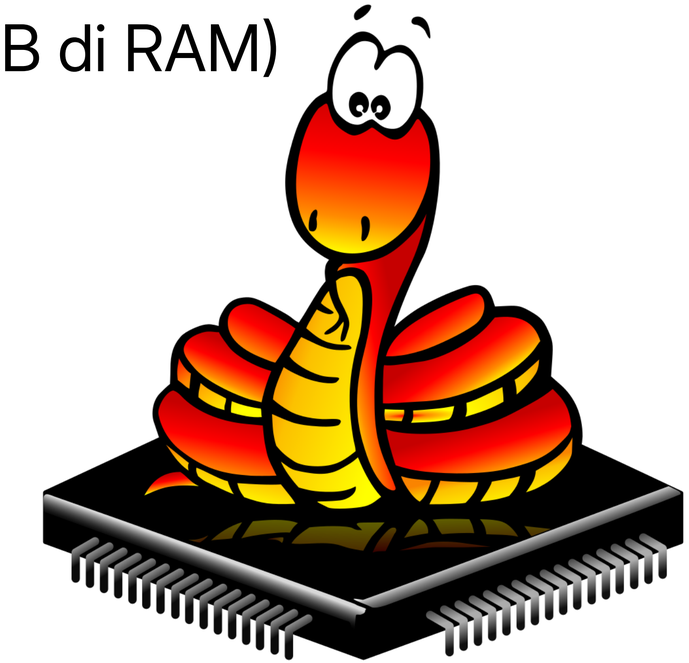
list/dict/tuple

Garbage collector

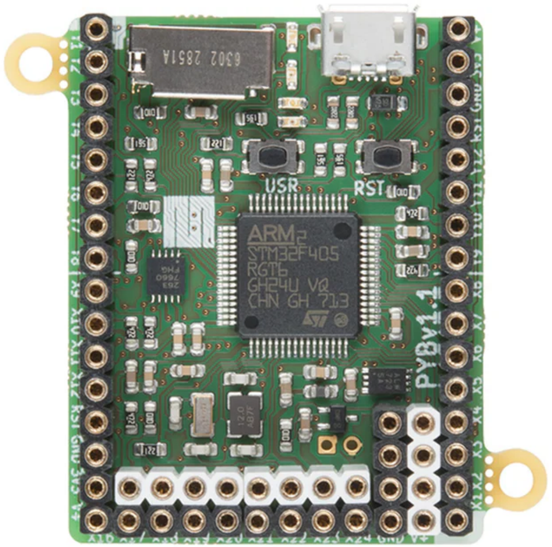
try/except

MicroPython

- Progetto Kickstarter(2013) by Damien George
- Python 3.x per microcontrollori
- Progettato per essere efficiente con le risorse (minimo 8KB di RAM)
- Progettato per essere eseguito bare metal
- Libreria **`machine`** per accesso hardware / GPIO
- Punta alla compatibilità con librerie CPython
- Open source: MIT license



Board compatibili

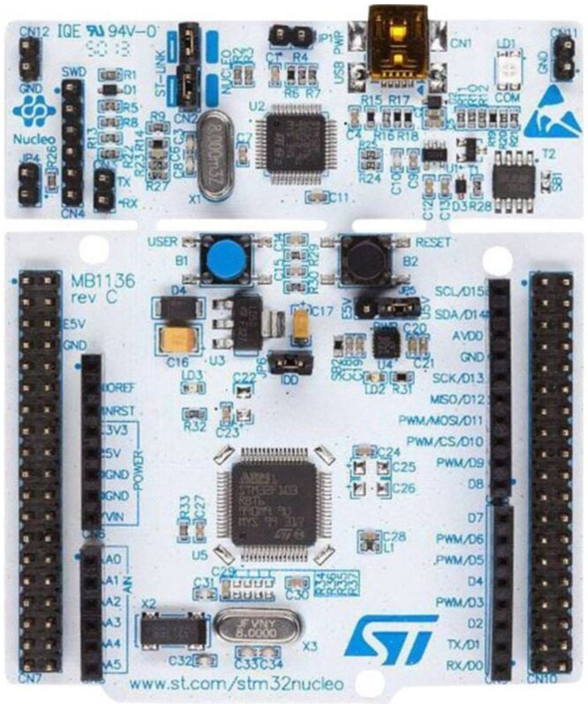


PyBoard (STM32)

ESP32/ESP8266

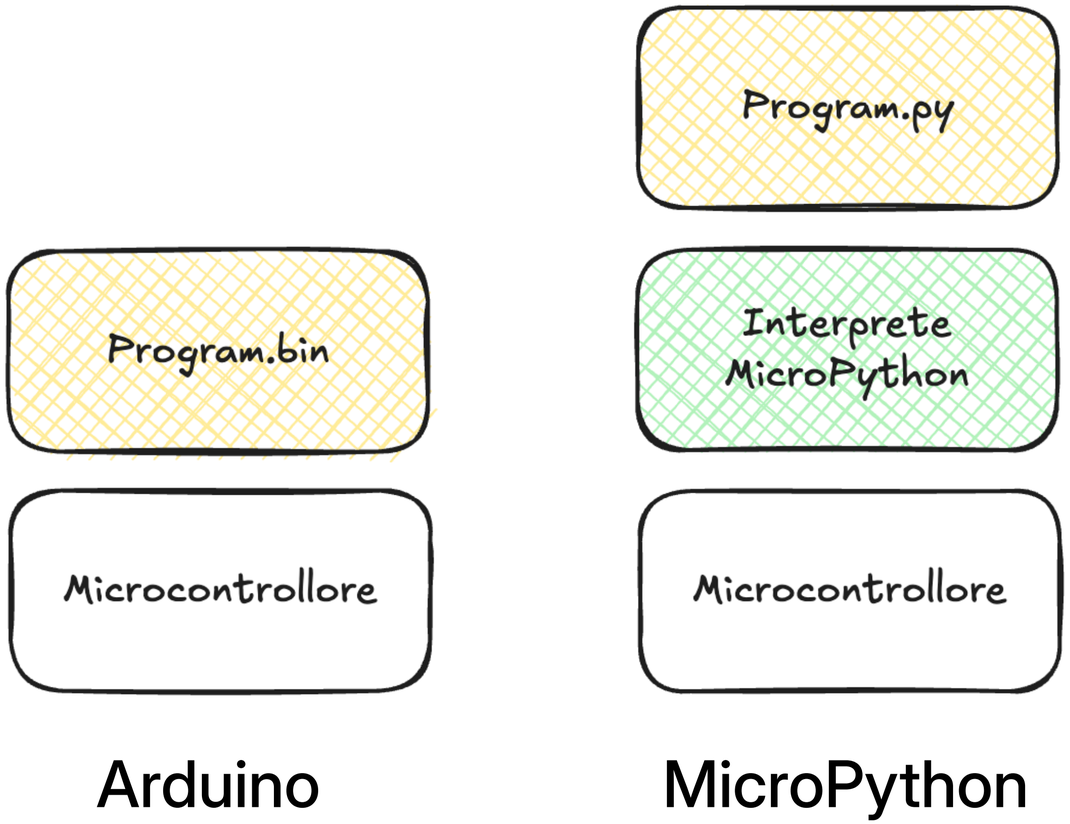
Raspberry Pi PICO (RP2040)

pca100XX



STM32

Arduino vs MicroPython



Read

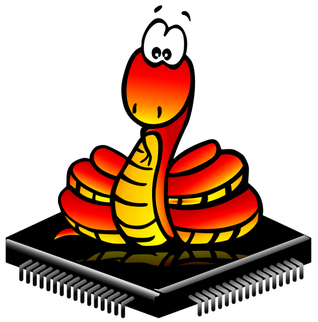
Evaluate

Print

Loop

Read

Evaluate



Print

Loop

Package Management

```
import mip
# Installa l'ultima versione dei "pkgname" (and dependencies)
mip.install("pkgname")
# Installa una versione specifica
mip.install("pkgname", version="x.y")

# Installa una libreria custom
mip.install("http://example.com/x/y/foo.py")
# Installa una libreria custom sotto forma di bytecode
mip.install("http://example.com/x/y/foo.mpy")
```

Moduli

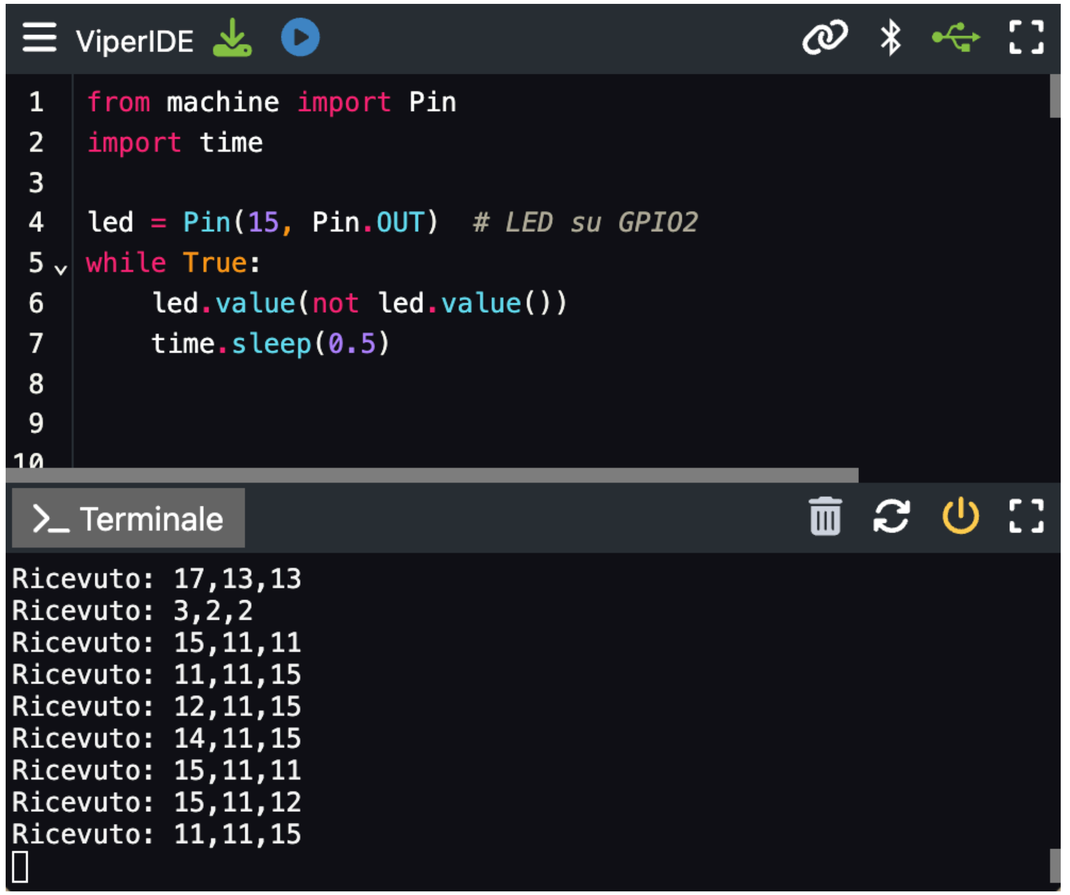
- Moduli built-in che replicano le funzionalità di Python (es. os, time)
- Moduli specifici per l'hardware sottostante (es. bluetooth, machine, network)
- Moduli di utilità (es. umqtt, dht)

```
import dht
import machine

sensor = dht.DHT22(machine.Pin(4))
sensor.measure()
print("Temperatura:", sensor.temperature(), "°C")
```

Ambiente di sviluppo

- Linea di comando **mpremote**
- **Thonny IDE**
- **VSCode** + micropython-stubs + MPRemote
- **ViperIDE** - viper-ide.org



Blink

```
from machine import Pin
import time

led = Pin(15, Pin.OUT)  # LED su GPIO15
while True:
    led.value(not led.value())
    time.sleep(0.5)
```

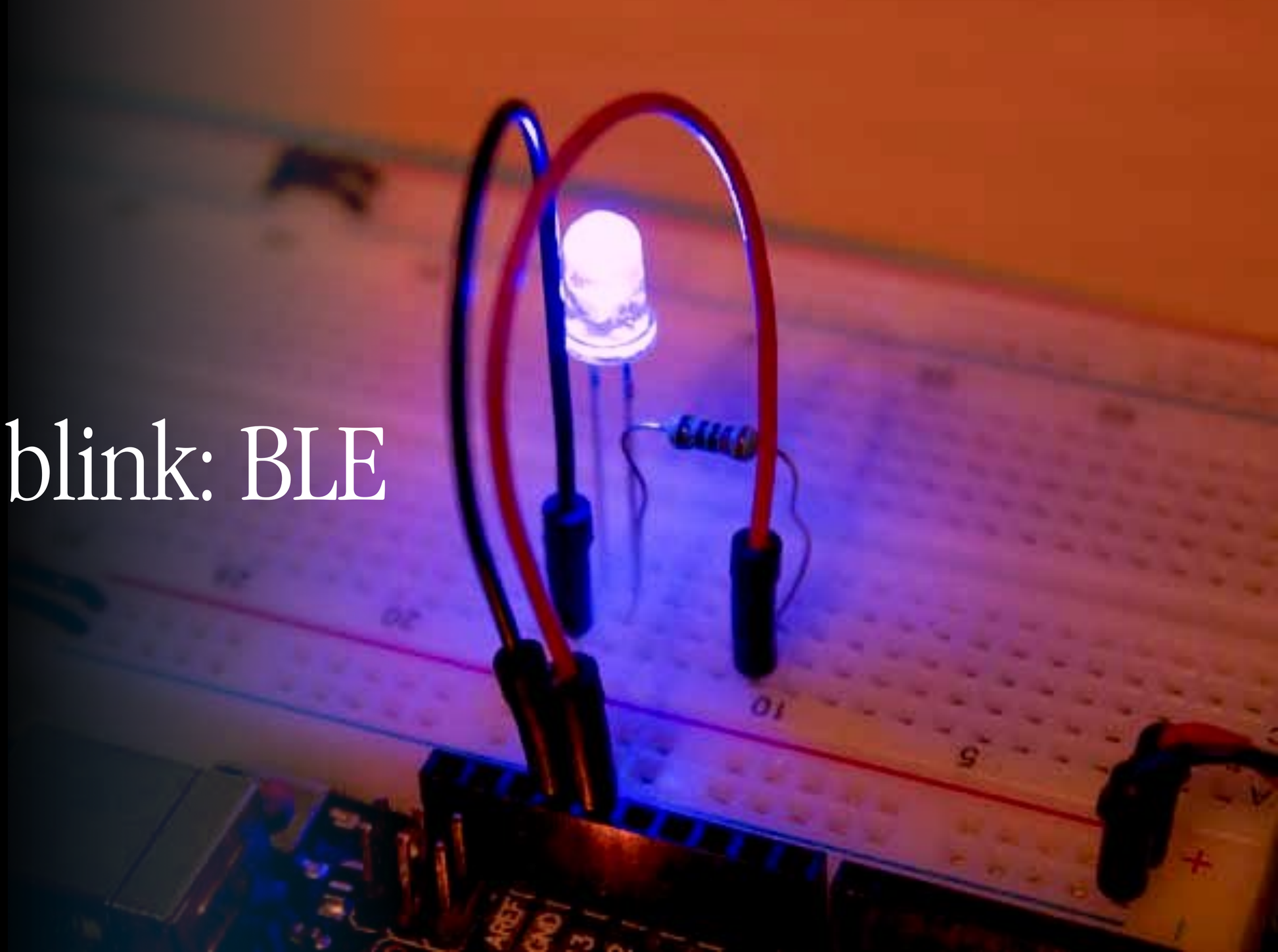
Non è tutto oro quel che luccica

- Prestazioni inferiori a C/C++:
  - il codice è più lento, anche 5–10 volte.
- Maggiore utilizzo della memoria
  - Interprete + programma .py (o .mpy)
  - Garbage collector
- Meno documentazione

MicroPython = sviluppo più rapido
C/C++ = codice ottimizzato e più veloce in esecuzione

3

Non solo blink: BLE

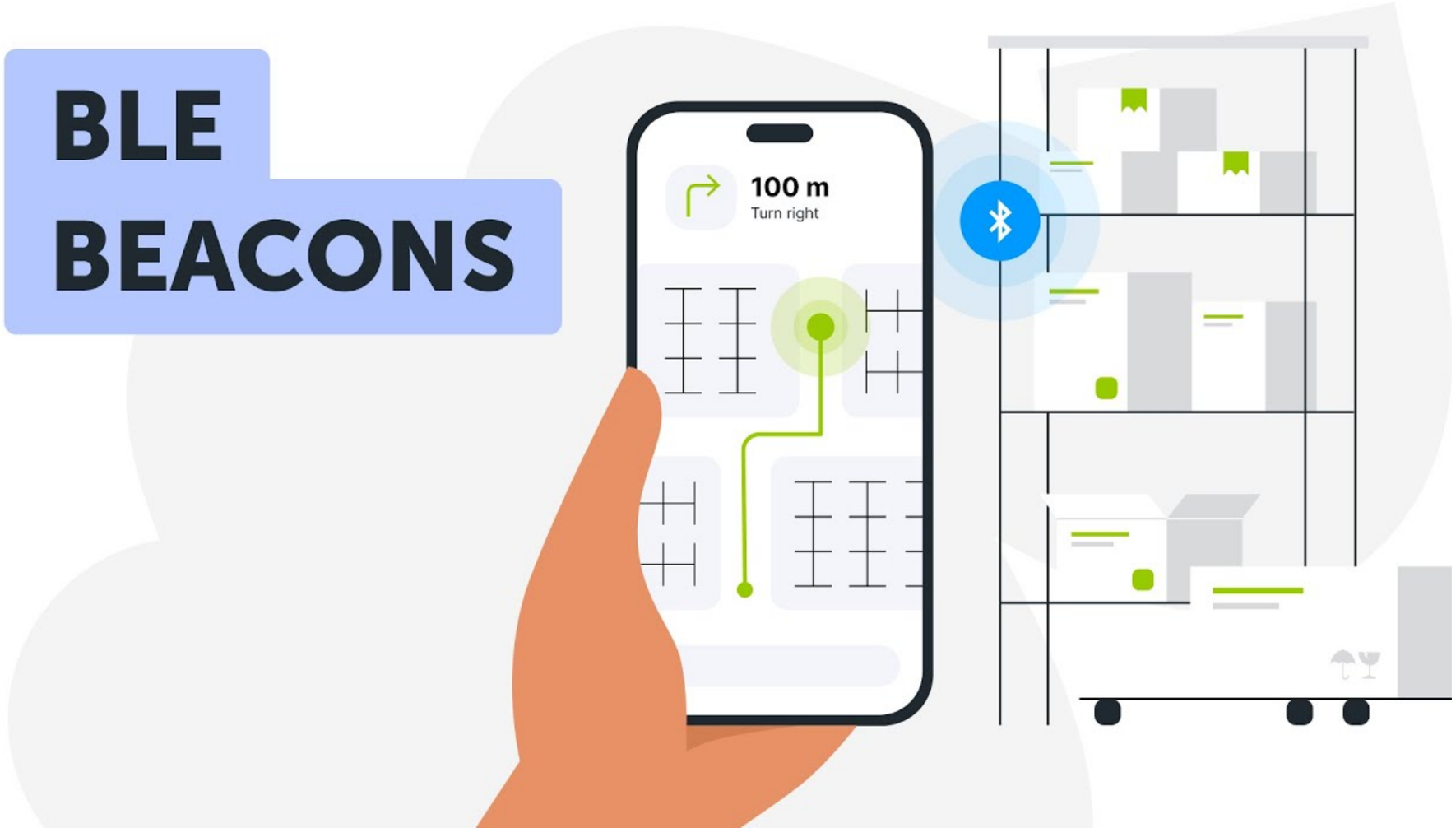


Cos’è il Bluetooth Low Energy

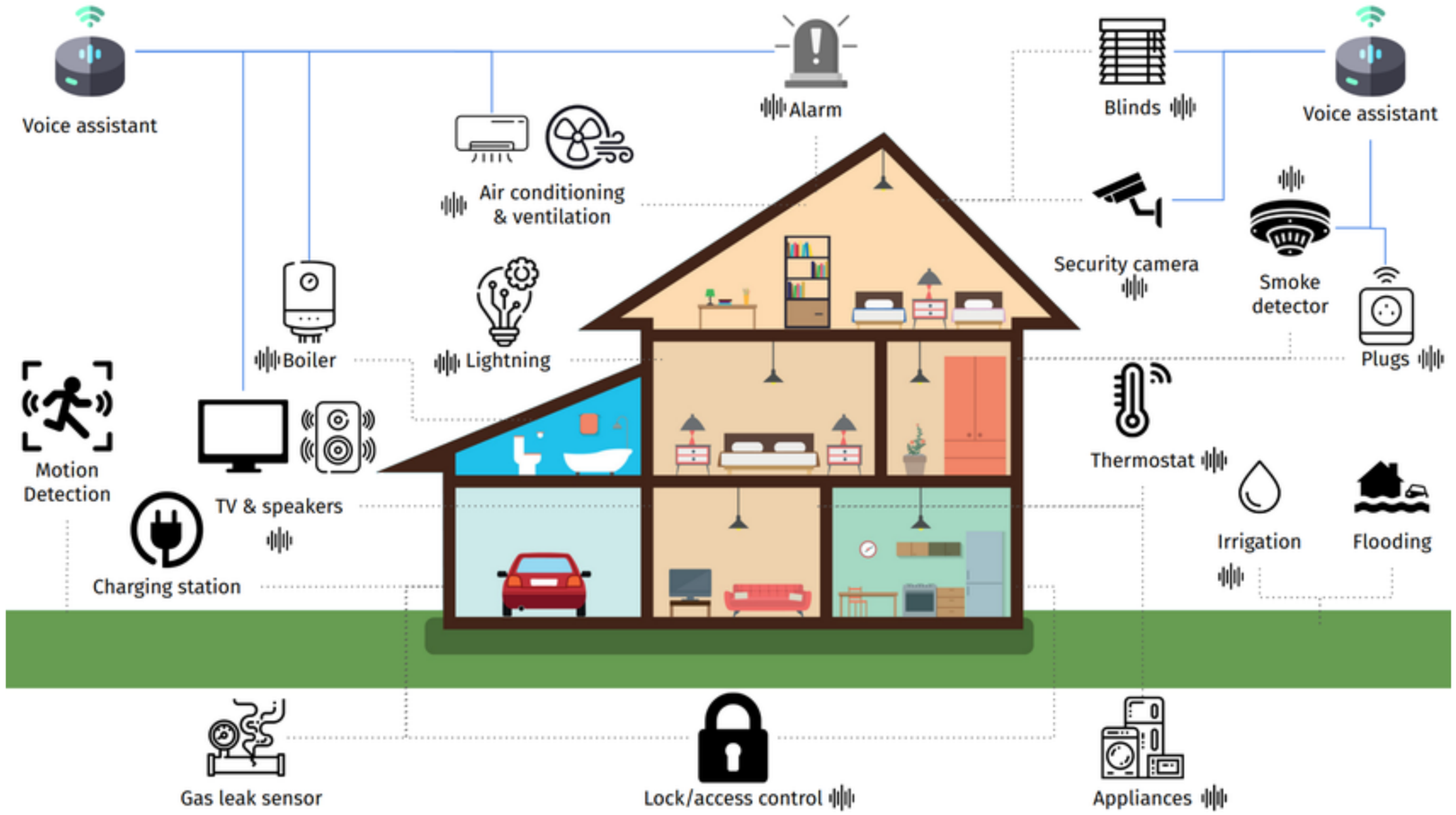
Device wireless che trasmettono rich-content come dati, video e audio.

Sensori che inviano pochi dati e low power

Asset Tracking



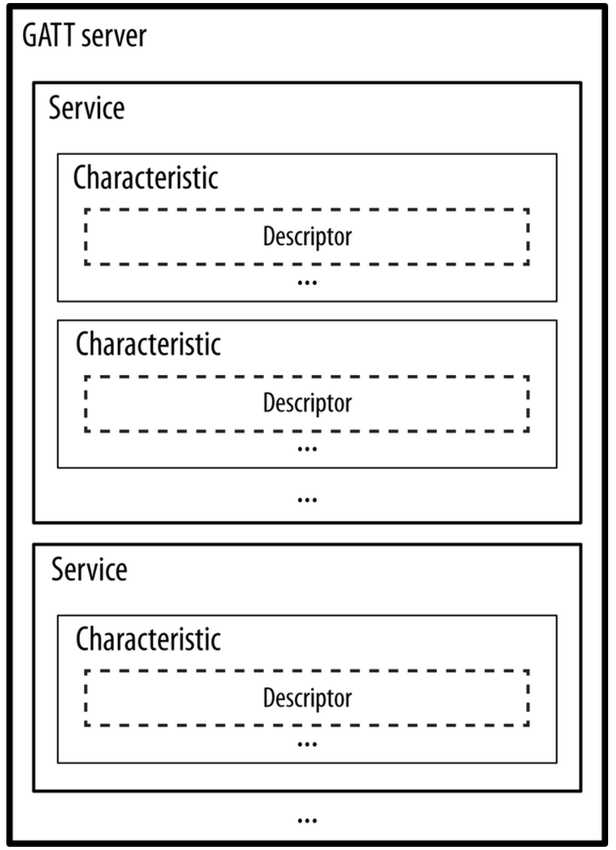
Smart Home


Voice assistant
Alarm
Blinds
Voice assistant
Air conditioning & ventilation
Security camera
Smoke detector
Boiler
Lightning
Plugs
Motion Detection
Thermostat
TV & speakers
Irrigation
Flooding
Charging station
Gas leak sensor
Lock/access control
Appliances

Wearables

BLE: Architettura Generale

- Due tipi di dispositivi BLE:
  - **Peripheral**: piccolo dispositivo che espone dati o sensori
    - (es. smartwatch, beacon, termometro).
  - **Central**: dispositivo che legge o controlla i peripheral
    - (es. smartphone, gateway, PC).
- Dopo la connessione, si scambiano dati tramite il protocollo GATT (Generic Attribute Profile).



Advertising e Scanning

- Il **Peripheral** invia periodicamente pacchetti di advertising
  - I pacchetti contengono informazioni base come nome, UUID, servizi offerti, ecc.
- Il **Central** esegue lo scan, riceve i pacchetti e decide se:
  - Connettersi al dispositivo
  - Leggere i dati direttamente (nel caso di beacon)

aioble

- `aioble` è una libreria MicroPython asincrona (basata su uasyncio) per gestire il Bluetooth Low Energy
- Fornisce un'API **più semplice e ad alto livello** rispetto al modulo `bluetooth`
- Gestisce automaticamente:
  - Advertising e scanning
  - Connessioni (Central ↔ Peripheral)
  - Servizi e caratteristiche GATT

Nordic UART Service (NUS)

- È un servizio BLE standard (sviluppato da Nordic Semiconductor) che emula una **seriale UART** via Bluetooth.
- Consente di inviare e ricevere stringhe di testo o dati binari tra due dispositivi BLE.
- Utilizza due caratteristiche:
  - **RX Characteristic**: per ricevere dati (dal Central → Peripheral)
  - **TX Characteristic**: per inviare dati (dal Peripheral → Central)
- Molto usato per debug, comandi remoti o comunicazione tra microcontrollore e smartphone.


DEMO