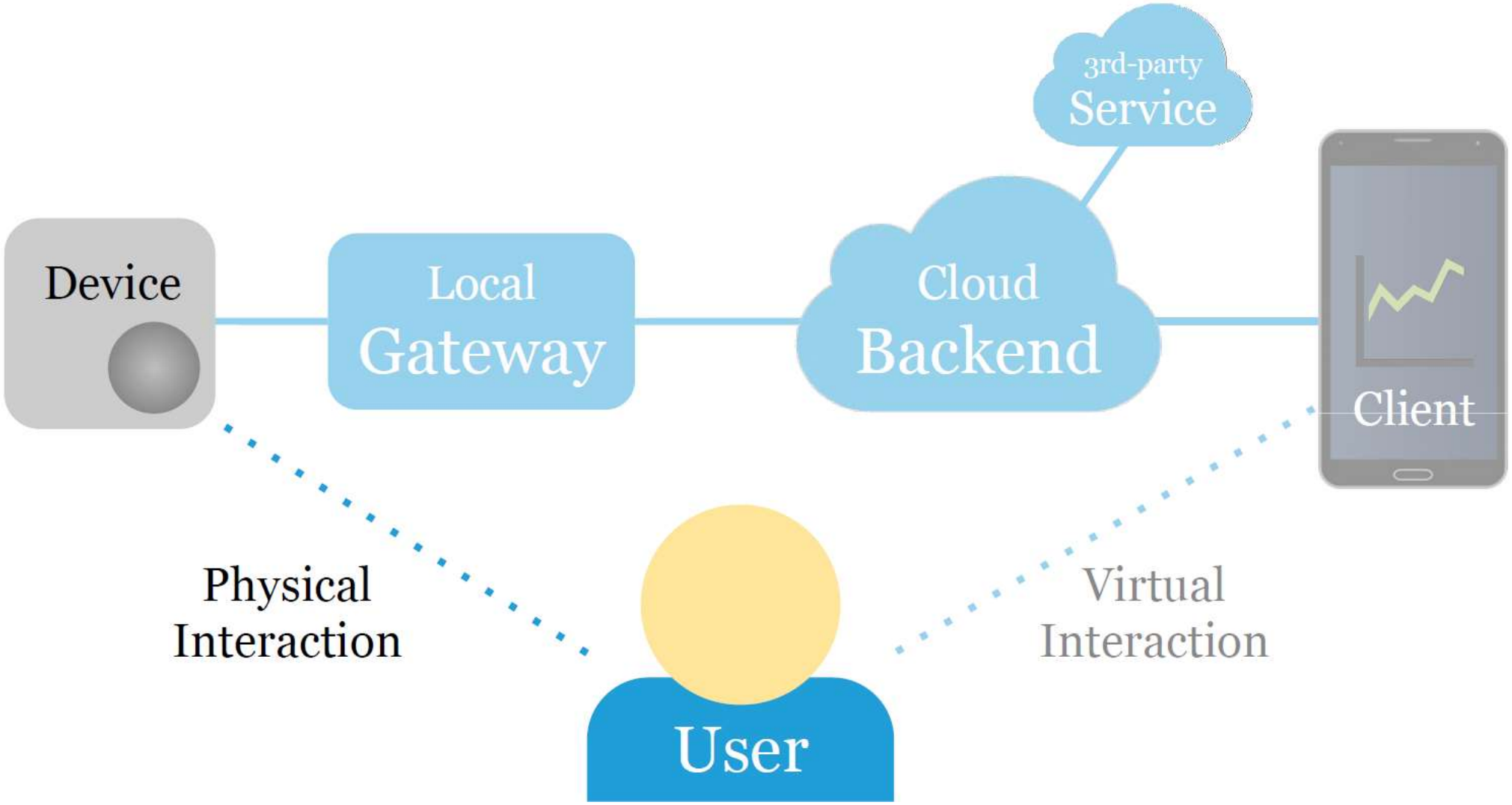
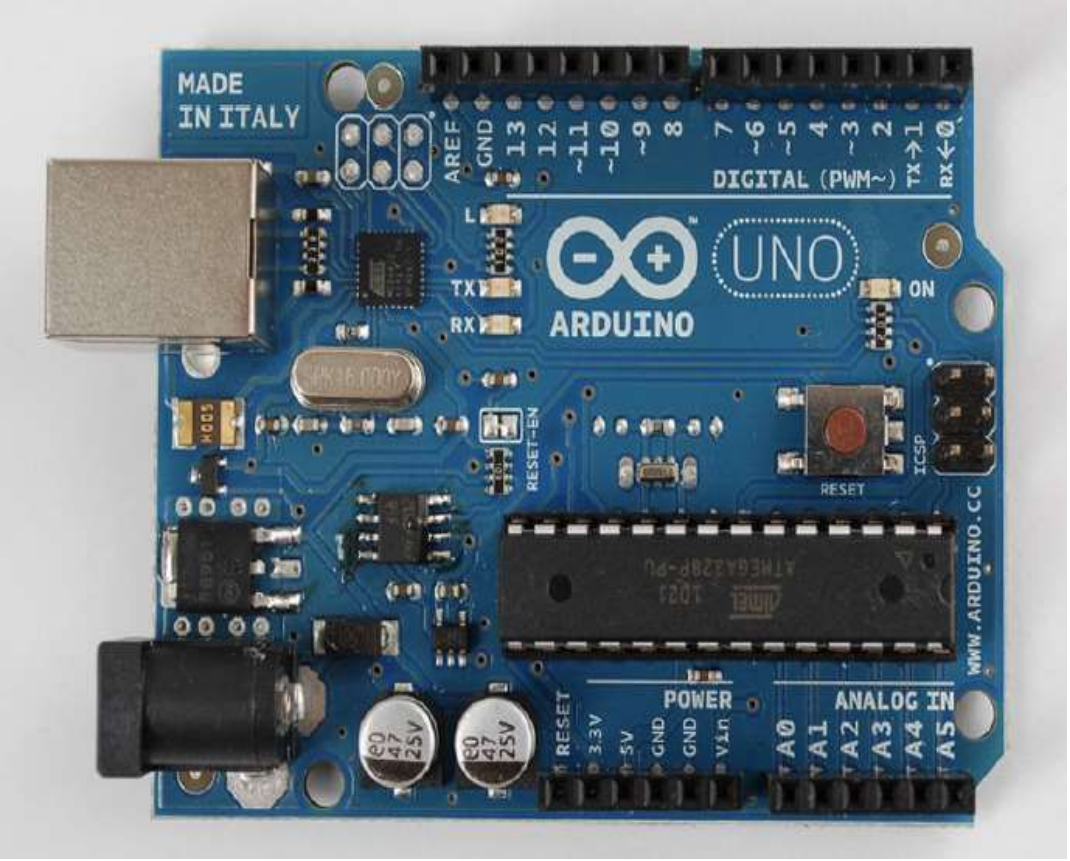
Microcontroller & SoC

IoT reference model



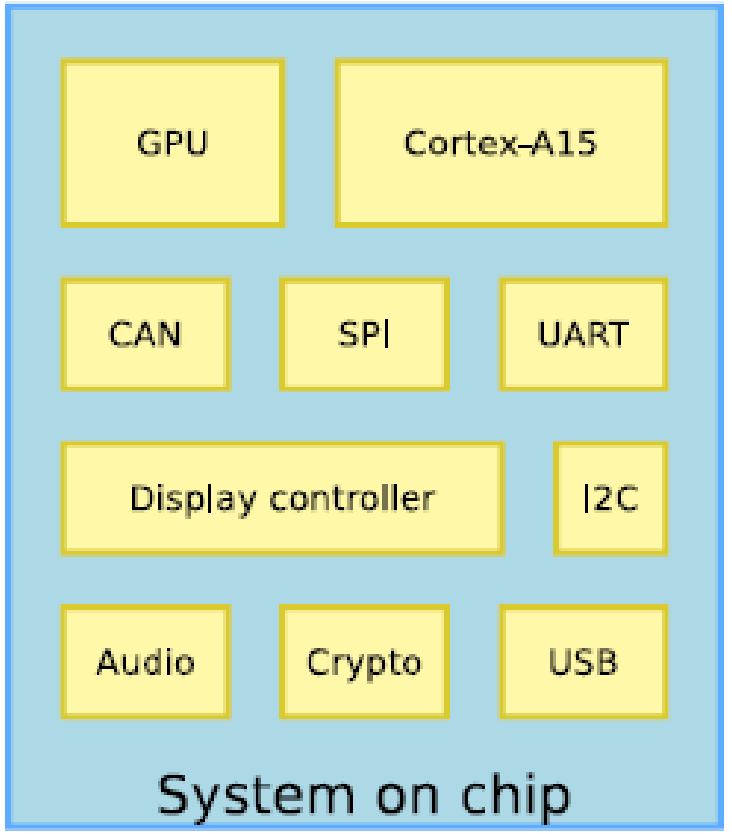
Microcontroller

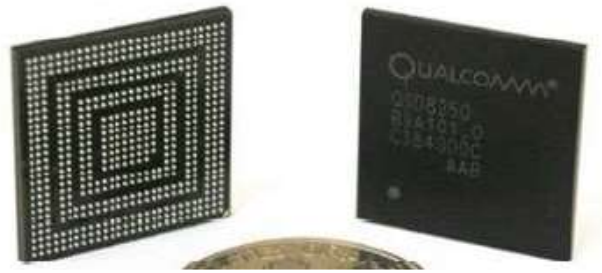
- *Microcontrollers* (MCU) are small computers that run a single program.
- *Arduino* is an MCU for electronics prototyping.



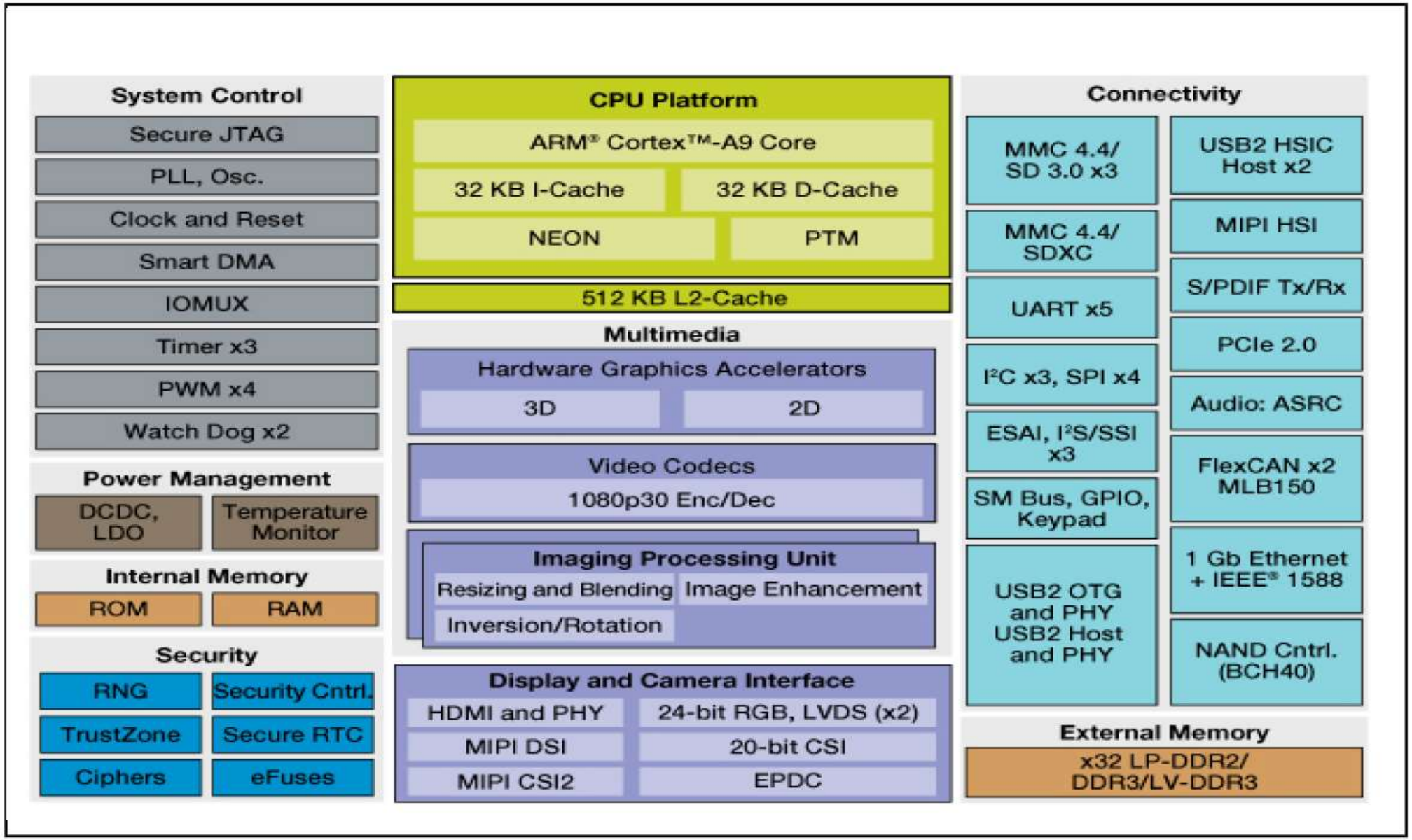
System-on-Chip(SoC)

- **System-on-chip**: integrated circuit that integrates all components of a computer system
  - CPU, but also peripherals: Ethernet, USB, UART, SPI, I2C, GPU, display, audio, etc.
  - Integrated in a single chip: easier to use, more cost effective
- **SoC vendors**
  - **Buy an ARM core** from ARM
  - Integrate **other IP blocks**, either designed internally, or purchased from other vendors
  - **Create and sell silicon**
- **Large** spectrum of SoCs available, addressing very different markets: automotive, mobile, industrial, low-power, set-top box, etc.

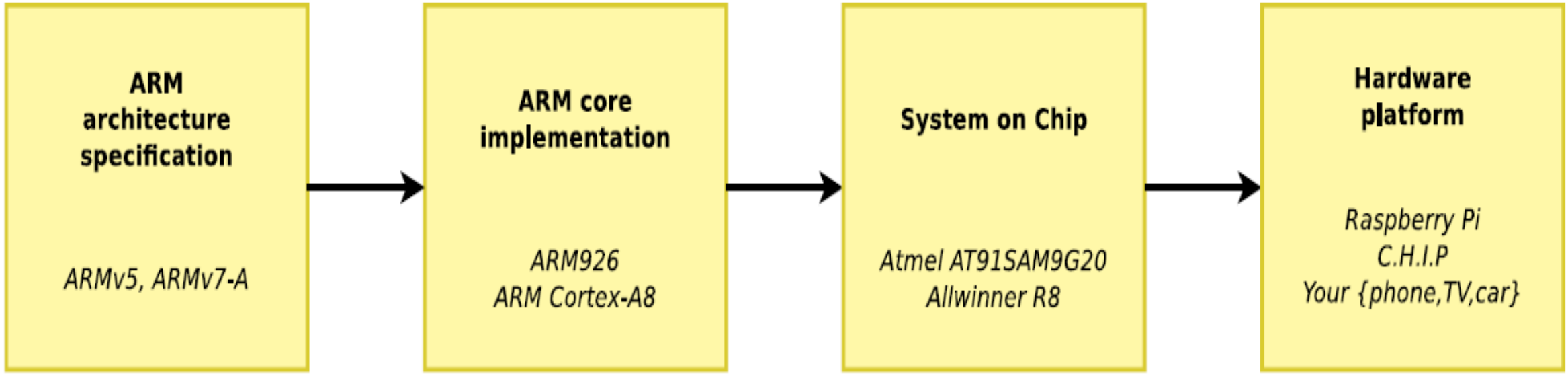




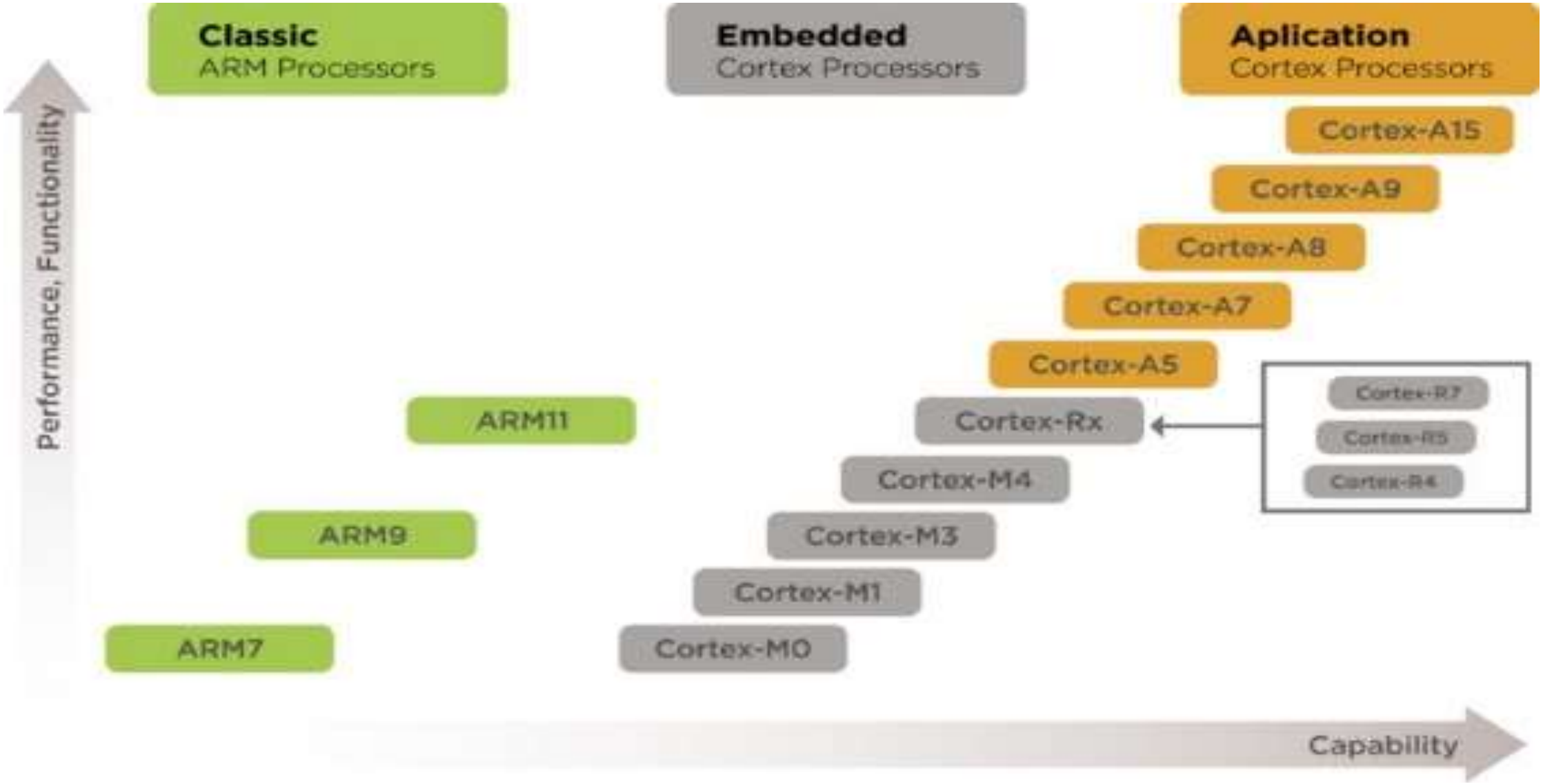
SoC Example: Freescale iMX6



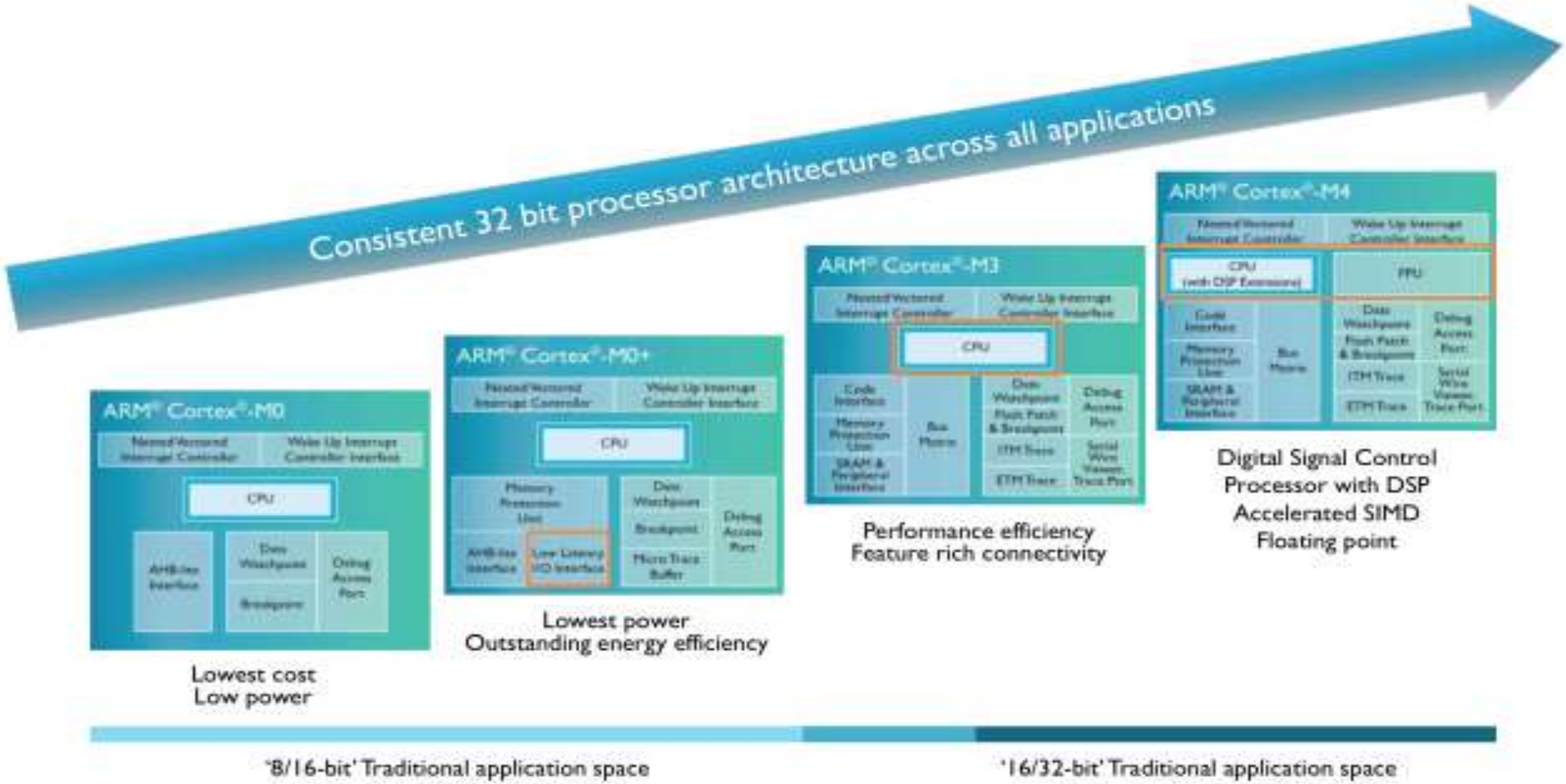
ARM: from the architecture to the board



Cortex-A, Cortex-R, or Cortex-M



ARM Cortex-M Product Line



Examples of ARM board

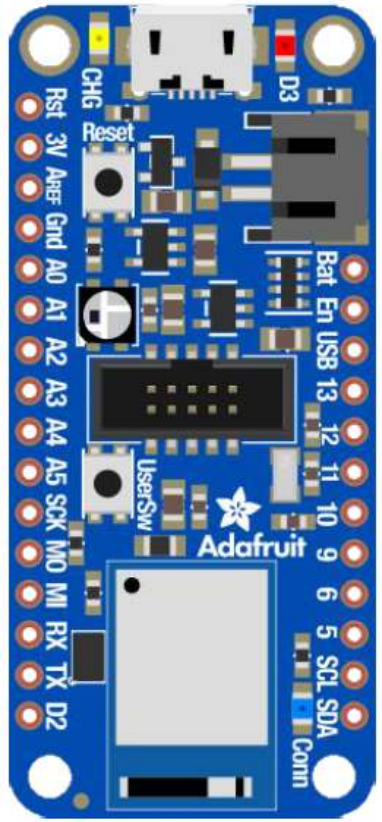
Feather nRF52840 Express

Microcontroller with Bluetooth 5 (and more).

Nordic nRF52840 System on Chip (SoC).

32-bit ARM Cortex-M4 CPU with FPU.

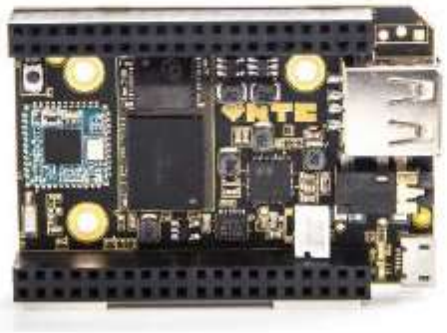
1 MB flash memory, 265 kB RAM.



Examples of ARM board

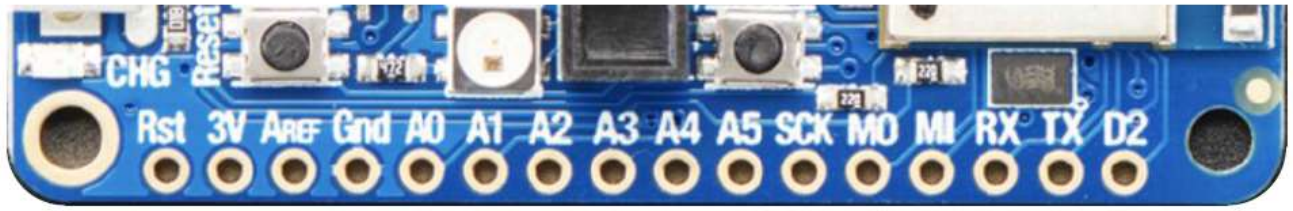
- **RaspberryPi 1**
  - SoC: Broadcom 2835
  - ARM core: ARM1176JZF (single)
  - ARM architecture: ARMv6
- **RaspberryPi 2**
  - SoC: Broadcom 2836
  - ARM core: Cortex-A7 (quad)
  - ARM architecture: ARMv7-A
- **C.H.I.P**
  - SoC: Allwinner R8
  - ARM core: Cortex-A8 (single)
  - ARM architecture: ARMv7-A
- **ESPRESSOBin**
  - SoC: Marvell Armada 3700
  - ARM core: Cortex-A53 (dual)
  - ARM architecture: ARMv8-A






General purpose input and output

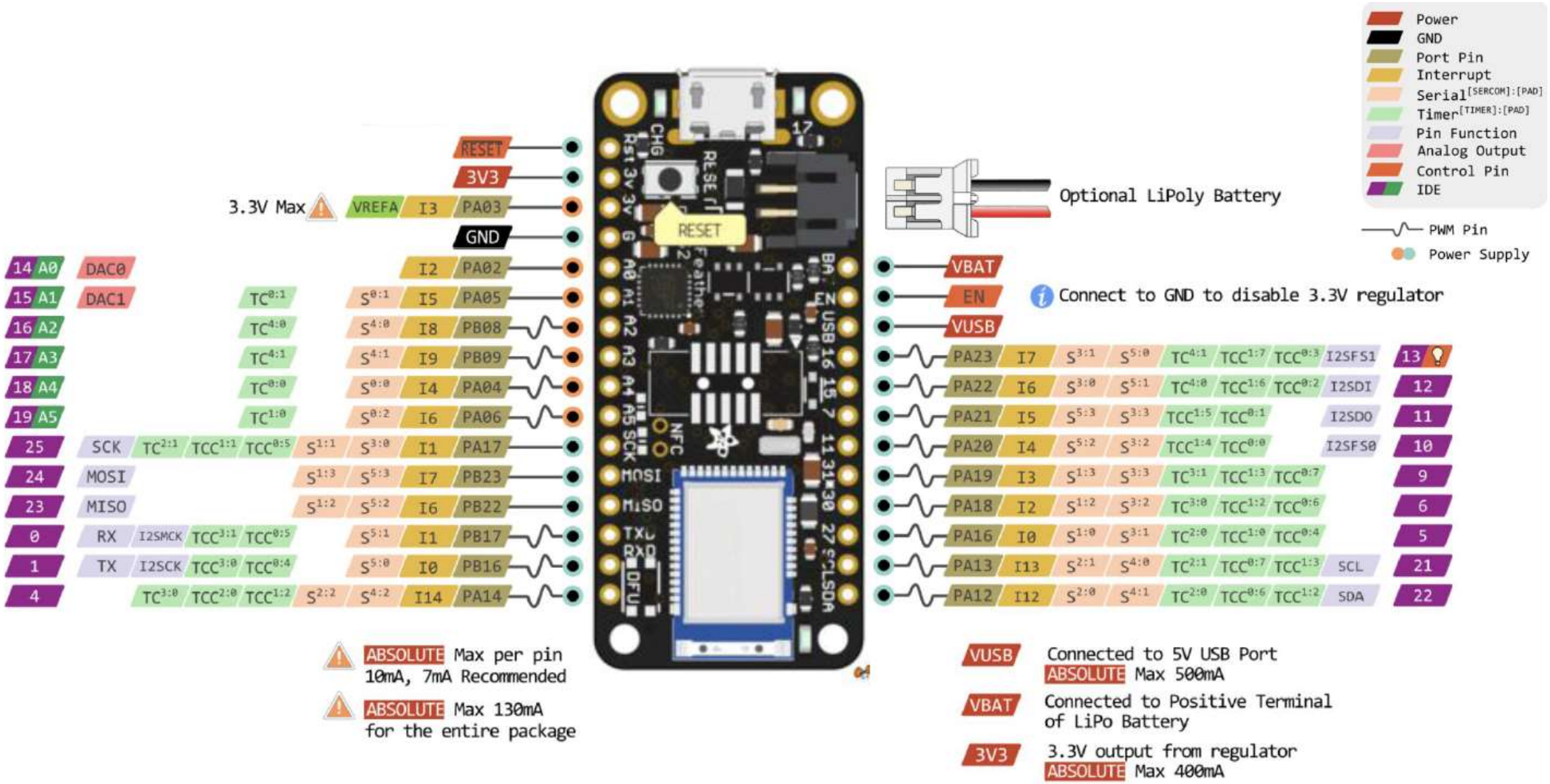
- Microcontrollers can "talk to" the physical world through general-purpose input and output (GPIO).
- GPIO *pins* allow an MCU to measure/control signals.



E.g. power, ground, analog pins, digital pin

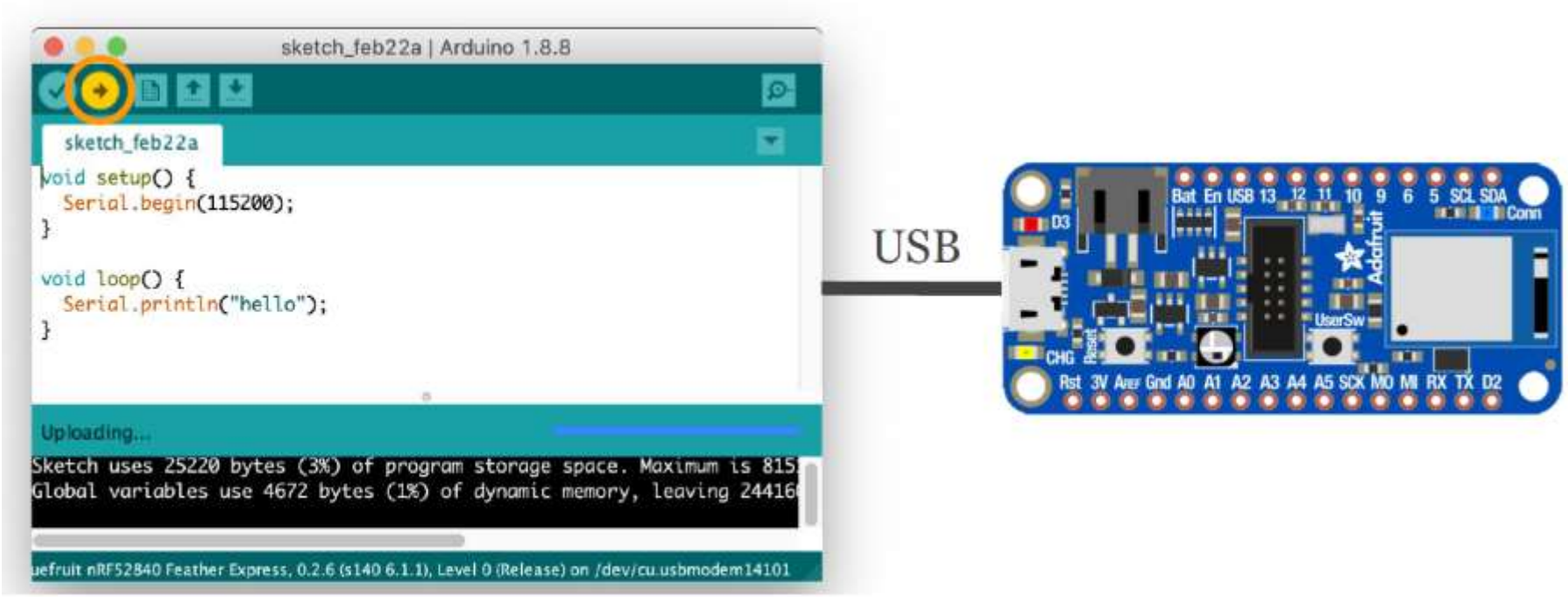
The map of available pins is called *pinout*

nRF52840


Power
GND
Port Pin
Interrupt
Serial[SERCOM]:[PAD]
Timer[TIMER]:[PAD]
Pin Function
Analog Output
Control Pin
IDE
PWM Pin
Power Supply
RESET
3V3
3.3V Max
GND
Optional LiPoly Battery
VBAT
EN
VUSB
Connect to GND to disable 3.3V regulator
ABSOLUTE Max per pin
10mA, 7mA Recommended
ABSOLUTE Max 130mA
for the entire package
VUSB
Connected to 5V USB Port
ABSOLUTE Max 500mA
VBAT
Connected to Positive Terminal
of LiPo Battery
3V3
3.3V output from regulator
ABSOLUTE Max 400mA

Programming a microcontroller

- Microcontrollers are programmed via USB.
- Code is (cross-) *compiled* on your computer.
- The *binary* is *uploaded* to the microcontroller.
- The uploaded program then runs "stand-alone".



A typical program in Arduino C

```
void setup() { // called once at startup
  Serial.begin(115200); // set baud rate
}

void loop() { // called in a loop
  Serial.println("Hello, World!");
}
```

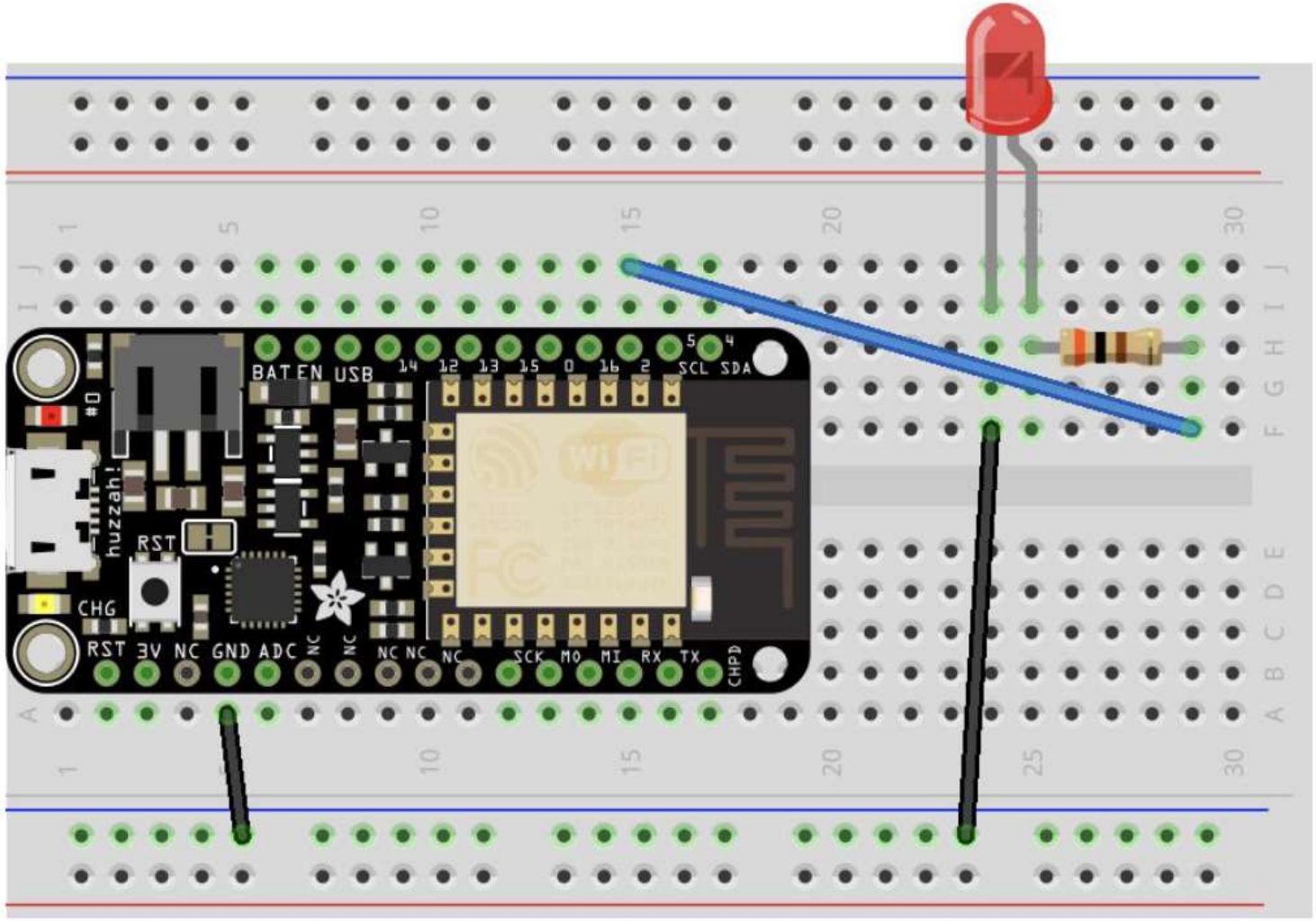
Arduino language

- The Arduino language uses a subset of C/C++.
- The user exposed code looks a bit like Java.
- There is a string type and a String class.
- Libraries are programmed in C++.

Breadboard prototyping

Wire electronic components, no soldering.

Under the hood, the columns are connected, also the power rails.



Arduino example code: Blinking a LED (digital output)

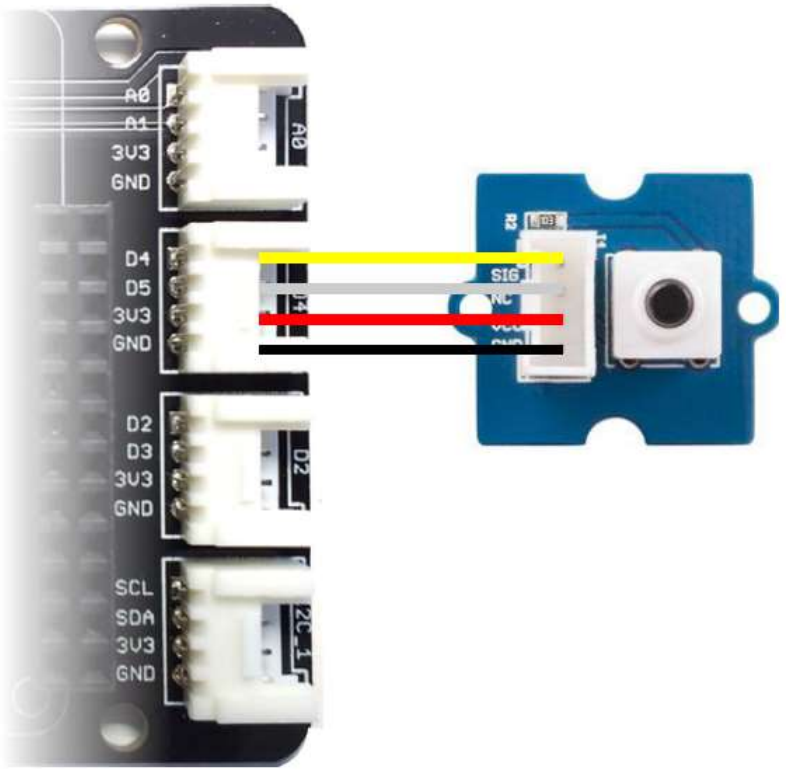
```
pin = 0; // for ESP8266, or 9 for nRF52840

void setup() { // called once
  pinMode(pin, OUTPUT); // configure pin
}

void loop() { // called in a loop
  digitalWrite(pin, HIGH); // switch pin on
  delay(500); // ms
  digitalWrite(pin, LOW); // switch pin off
  delay(500); // ms
}
```

Arduino example code: Reading a button (digital input)

Connect to Grove port *D4*.

It maps to ESP8266 pin *0*.

Or nRF52840 pin *9*.



Reading a button (digital input)

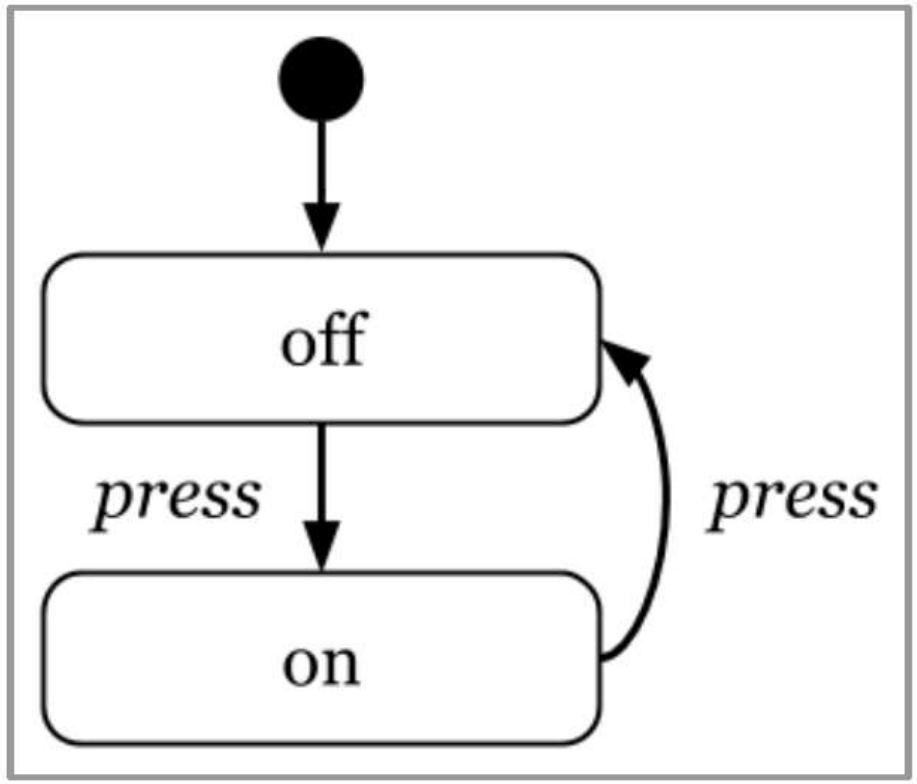
```
pin = 0; // for ESP8266, or 9 for nRF52840

void setup() { // called once
  pinMode(pin, INPUT); // configure pin
  Serial.begin(9600);
}

void loop() { // called in a loop
  int value = digitalRead(pin);
  Serial.println(value);
  delay(500); // ms
}
```

Button-triggered LED

A (finite-) state machine is a simple way to manage state in embedded programs.

System is in one state at a time, *events* trigger state *transitions*.

E.g. 1st button *press* => light *on*, 2nd button *press* => light *off*, 3rd => *on*, 4th => *off*, etc.



State machine (refined)

Button is *high* or *low*.

Light is *on* or *off*.

Pressed = *low* → *high*.