

Sistemi Embedded e IoT

Ingegneria e Scienze Informatiche - UNIBO

a.a 2020/2021

Docente: Prof. Alessandro Ricci

[modulo 1.1]

INTRODUZIONE AI SISTEMI EMBEDDED

SOMMARIO

- In questo modulo viene fornita una panoramica introduttiva relativa ai concetti essenziali che concernono i sistemi embedded
 - concetti generali
 - architettura sistemi embedded
 - progettazione e sviluppo di sistemi embedded

SISTEMI EMBEDDED: DEFINIZIONE

- Sistemi di elaborazione **special-purpose** - ovvero che svolgono una specifica funzione o compito - *incorporati* (embedded) in sistemi o dispositivi elettronici di diverse dimensioni
 - il compito tipicamente richiede di *interagire con il mondo fisico* mediante opportuni **sensori** e **attuatori**
- Tipicamente costituiti da una parte hardware e una parte software
 - in alcuni casi la parte software può non esser presente
- Sistemi software embedded (**embedded software**)
 - parte software di un sistema embedded

DIFFUSIONE E PERVASIVITÀ

- Elettronica di consumo
 - cellulari, smartphone, telecamere,...
- Home appliances
 - termostati, impianti di illuminazione intelligenti, forno a microonde, robot pulisci pavimenti...
- Office automation
 - fotocopiatrici, stampanti,...
- Business equipment
 - registratori di cassa, sistemi di allarme,...
- Automobili, velivoli
 - sistema di trasmissione, sistema di navigazione,...
- ...

ESEMPI: HOME APPLIANCES



ESEMPI: INTERNET OF THINGS e SMART THINGS



Nest smart thermostat

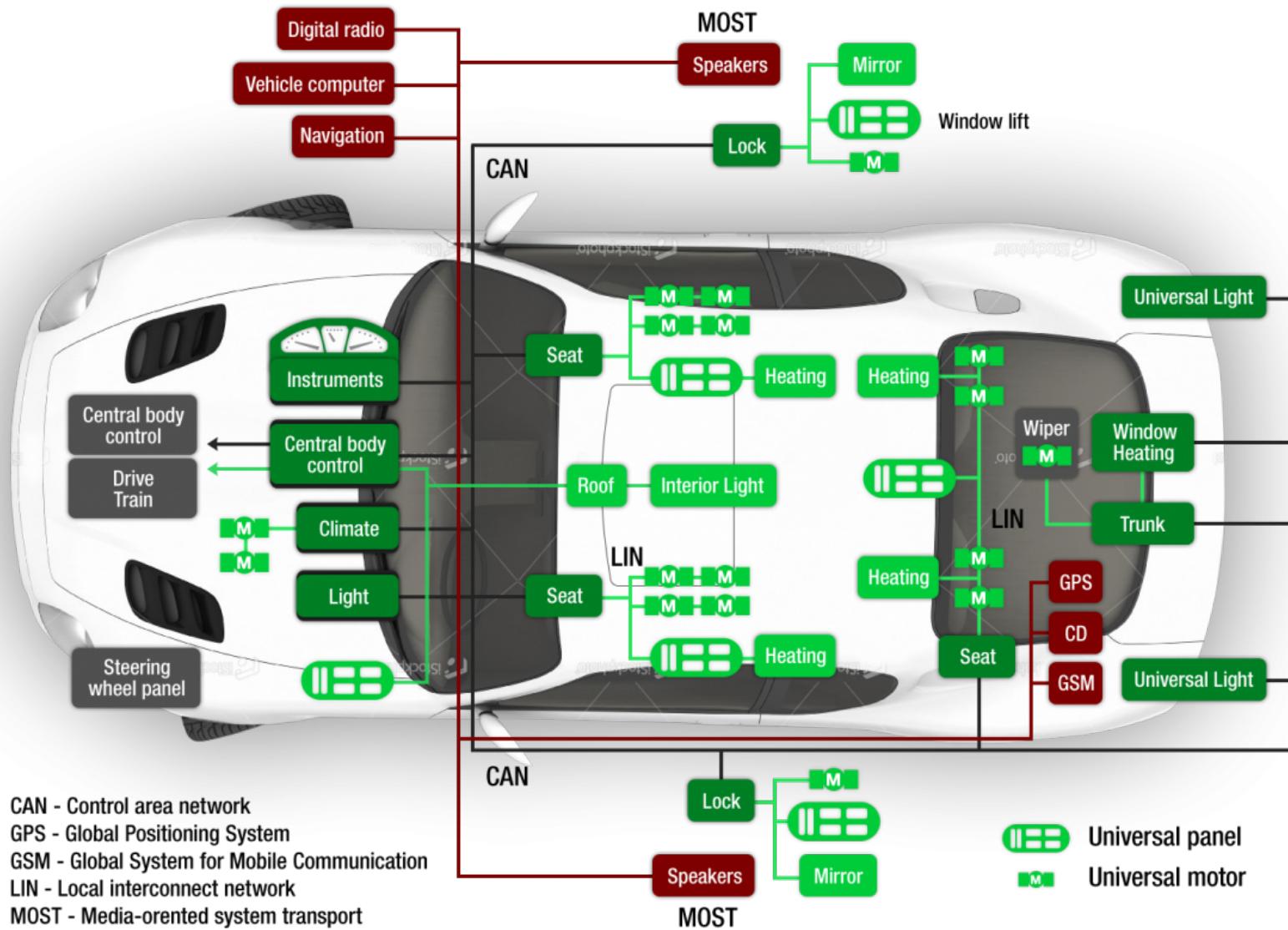


Kevo smart lock

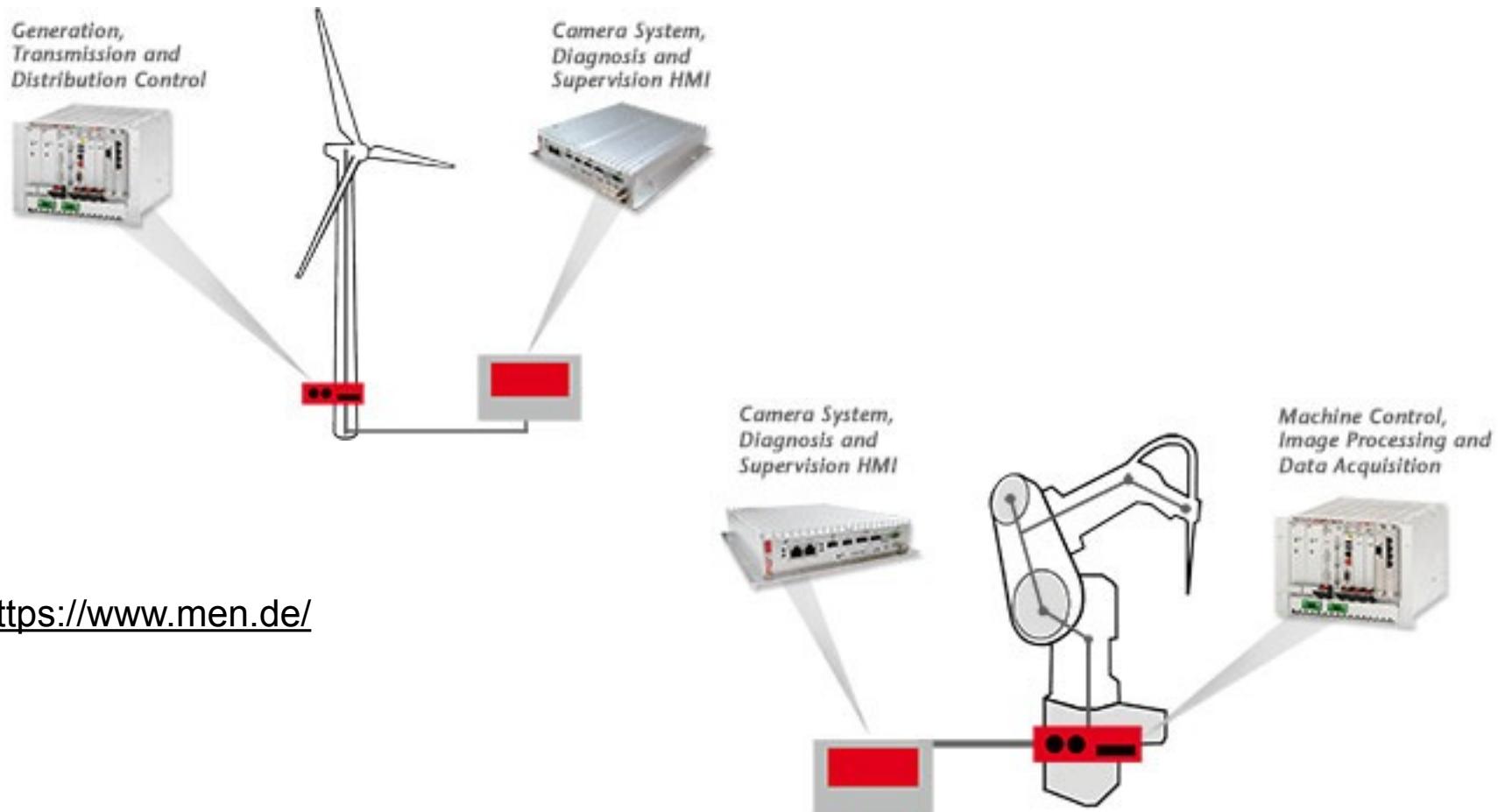


WeMo smart things

ESEMPI: AUTOMOTIVE



ESEMPI: IMPIANTI INDUSTRIALI

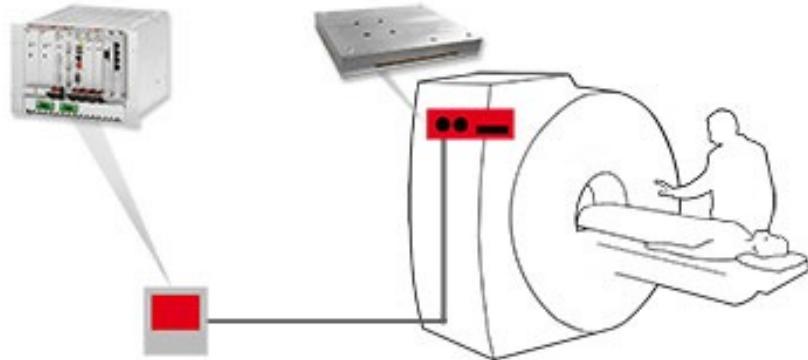


<https://www.men.de/>

ESEMPI: HEALTH CARE

Diagnosis, Monitoring and Patient Communication Systems

Surgical Control, Processing and Acquisition Components



Fitbit

ESEMPI: MOBILE E WEARABLE DEVICES



smart-glasses



smart-watch

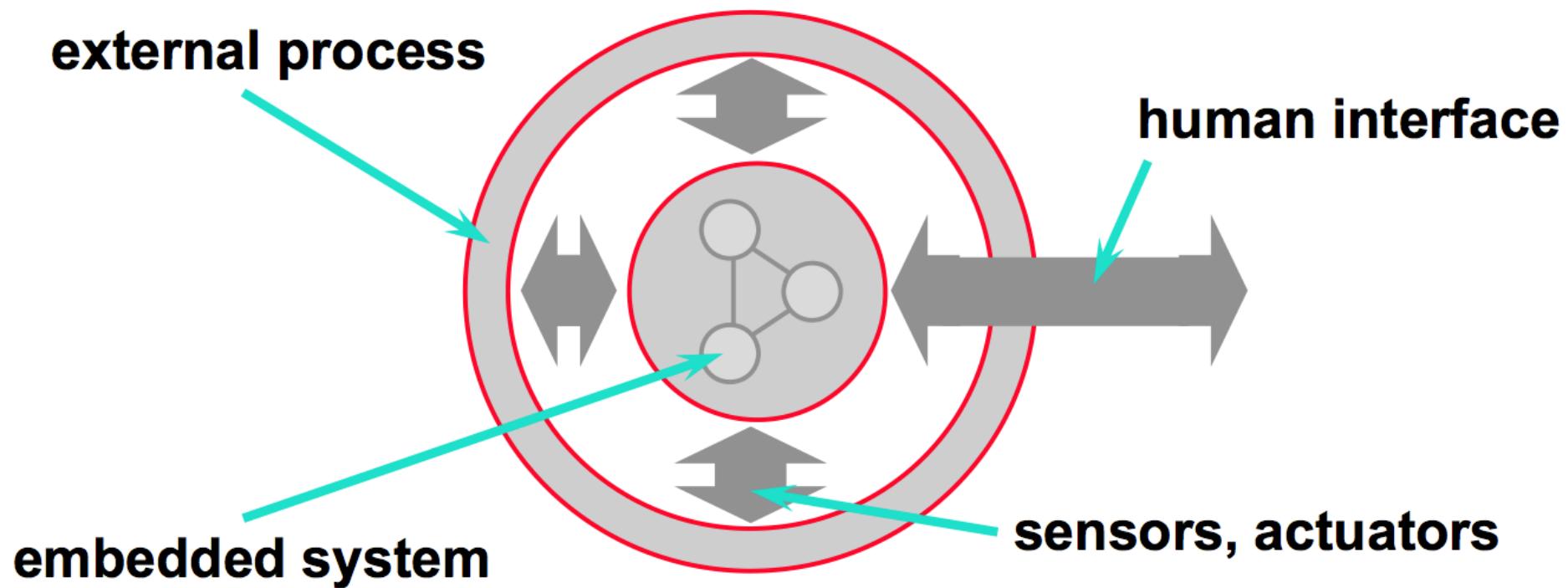


smart-helmet



smart armband

UNA RAPPRESENTAZIONE



(Embedded system introduction, http://www.tik.ee.ethz.ch/education/lectures/ES/slides/1_introduction)

CARATTERISTICHE

- **Funzionalità specifica**
 - un sistema embedded tipicamente esegue uno *specifico programma ripetutamente*
 - è quindi progettato per eseguire un'applicazione specifica
 - questo permette a design time di minimizzare le risorse da utilizzare e massimizzare la robustezza
 - user interface dedicata

CARATTERISTICHE

- **Risorse limitate (tightly constrained) ed efficienza**
 - tutti i sistemi di elaborazione hanno vincoli progettuali in merito alle risorse disponibili (memoria, CPU..), tuttavia quelle relative ai sistemi embedded sono tipicamente più stringenti
- Uso di metriche di progettazione (*design metrics*)
 - misura di una caratteristica dell'implementazione/realizzazione
 - **costo** - inclusi costi **NRE**, non-recurring-engineering = costi one time
 - **dimensioni, performance, consumo di energie**
- Progettazione orientata **all'efficienza**
 - energy efficient, code-size efficient, run-time efficient, weight efficient, cost efficient

CARATTERISTICHE

- **Affidabilità** (*dependability, reliability, availability*)
 - tipiche applicazioni dei sistemi embedded includono sistemi critici
 - ad esempio in ambito bio-medicale, in ambito di trasporto (es: automobili, aerei), etc
- ..per cui si richiede
 - **dependability, reliability, availability**
 - alta probabilità di corretto e continuo funzionamento
 - **safety and security**
 - alta probabilità che non rechi danni agli utenti o all'ambiente in cui è immerso
 - obtemperi norme di sicurezza e privacy

CARATTERISTICHE

- **Reattività e real-time**
 - spesso i sistemi embedded sono utilizzati in contesti dove devono prontamente *reagire a stimoli che provengono dall'ambiente (fisico)* e quindi eseguire elaborazioni ed eventualmente azioni in real-time, senza ritardi
 - nel caso di hard real-time, l'eventuale violazione di deadline può risultare in problemi critici per l'intero sistema

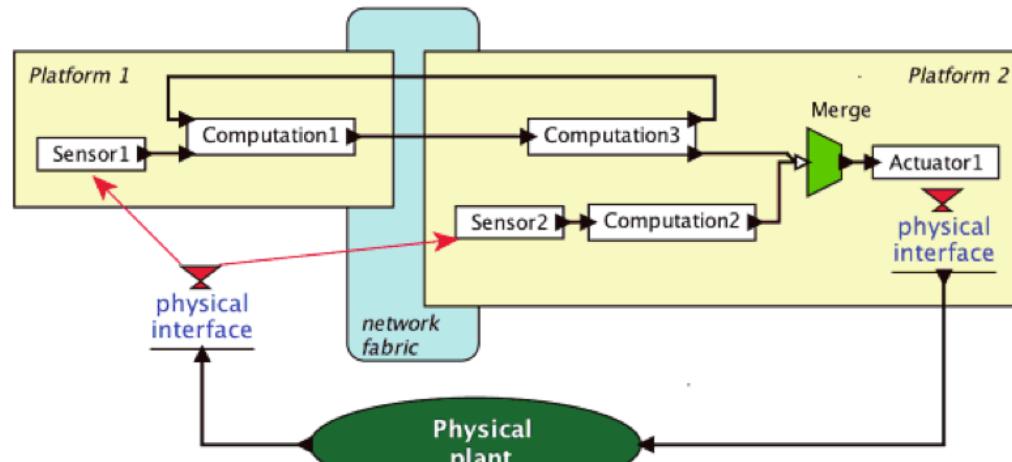
SISTEMI CYBER-PHYSICAL (CPS)

- **Cyber-Physical Systems (CPS)**
 - sistemi che *integrano computazione con processi fisici*
 - vs. information processing systems
- Aspetti specifici rispetto ai sistemi di pura elaborazione delle informazioni
 - gestione del **tempo**
 - non solo performance, ma correttezza
 - **concorrenza**
 - processi fisici sono tipicamente in parallelo
 - **reattività**, eventi asincroni
 - sistemi reattivi vs. trasformazionali

SISTEMI CYBER-PHYSICAL (CPS)

- 3 parti o sottosistemi
 - **la parte fisica**
 - può includere dispositivi/sistemi meccanici, biologici, o processi chimici, o operatori (utenti) umani
 - **la parte computazionale/embedded**
 - data da una o più piattaforme di elaborazione, ognuna contenente sensori, attuatori e uno o più computer S.
 - **la parte di rete**
 - fornisce i meccanismi e supporti per fare in modo che i computer comunichino

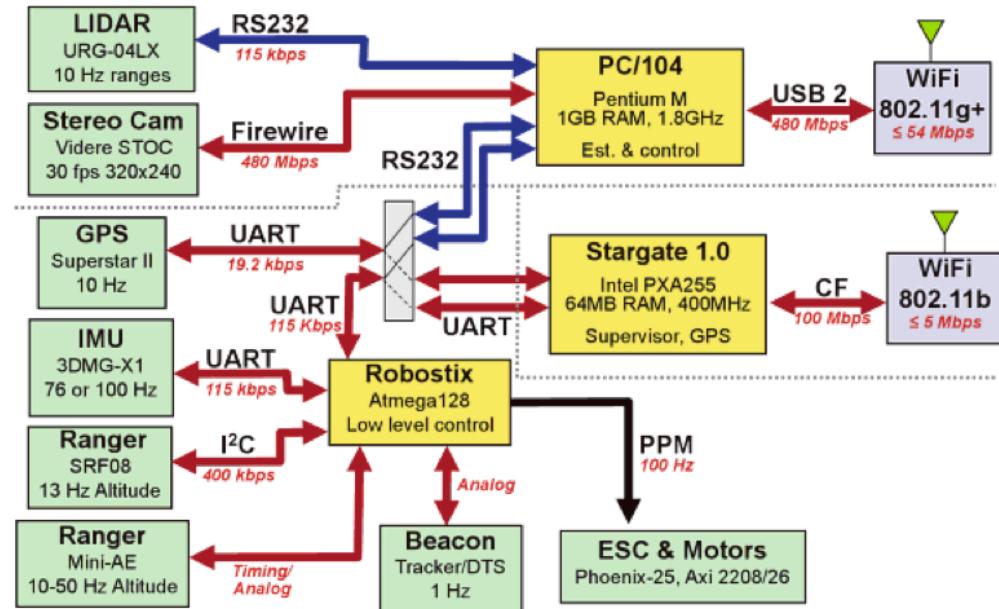
(IES book, p. 6)



ESEMPIO: DRONE QUADRIROTORE



(IES book, p. 7)



CPS: INTER-DISCIPLINARITY

Software

```

4) Based on above measures, which are the basic measures of
   the value creation of the customer?
      a) Increasing the price of the product by getting the customer to pay.
      b) ...
private static void calculateCustomerValue() {
    System.out.println("Customer analysis - Customer performance analysis");
    for (Customer customer : customers) {
        calculateCustomerValue(customer);
    }
}

private void calculateCustomerValue(Customer customer) {
    System.out.println("Customer analysis - Customer performance analysis");
    if (customer.isLoyal()) {
        System.out.println("...Adding additional basic measures of value creation.");
    }
}

Repeating (C) measures = simple profit maximization;
if (customer == null) {
    for (Customer c : C) {
        if (customer.equals(c)) {
            c.setCustomerValue();
        }
    }
}
for (Customer c : C) {
    if (customer.equals(c)) {
        if (customer.getCustomerValue() > customer.getCustomerValue()) {
            customer.setCustomerValue();
        }
    }
}

```

Control/ Physics

$$\dot{\mathbf{x}}(t) = \dot{\mathbf{x}}(0) + \frac{1}{M} \int_0^t \mathbf{F}(\tau) d\tau$$



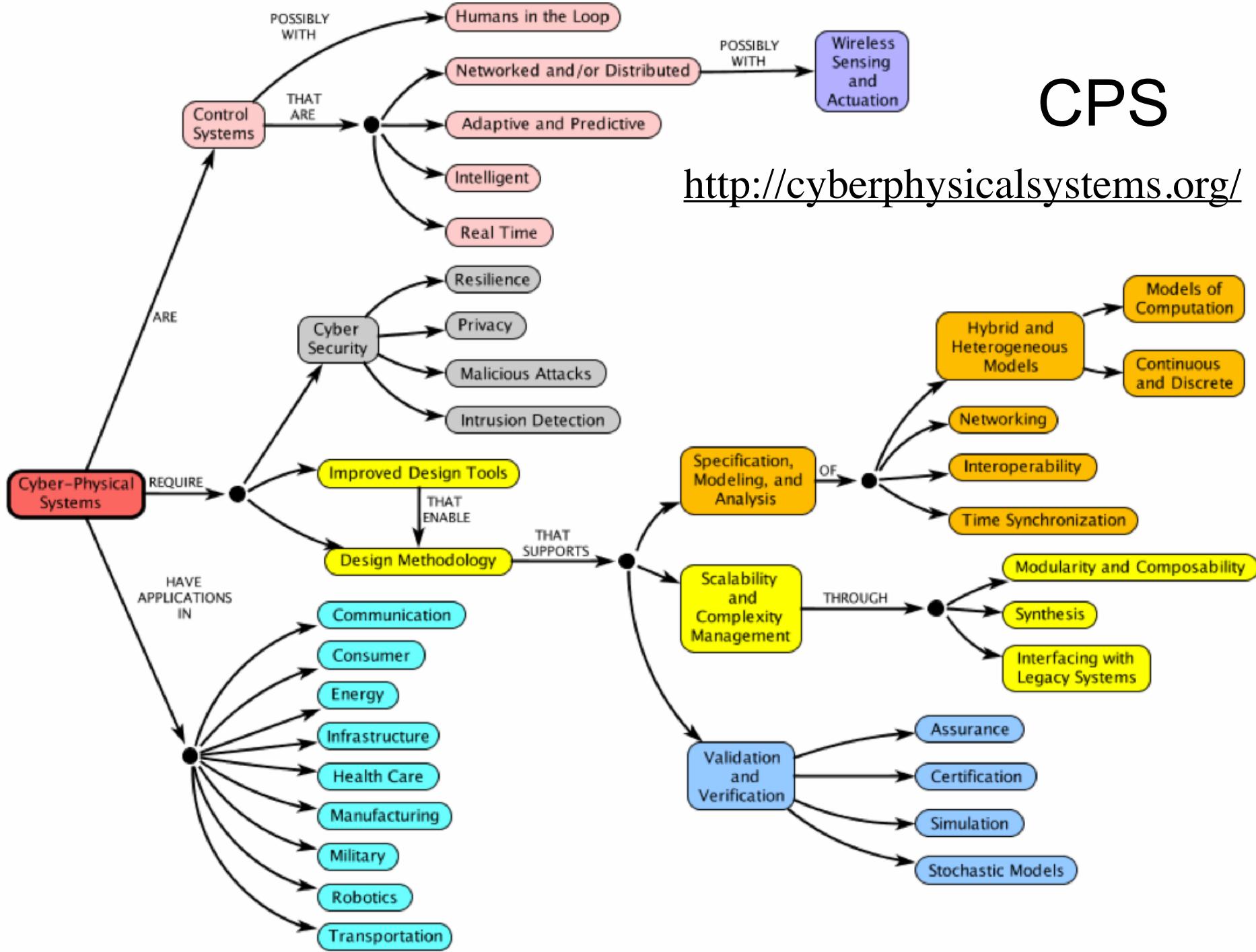
Systems



Overcome interdisciplinary boundaries

CPS

<http://cyberphysicalsystems.org/>

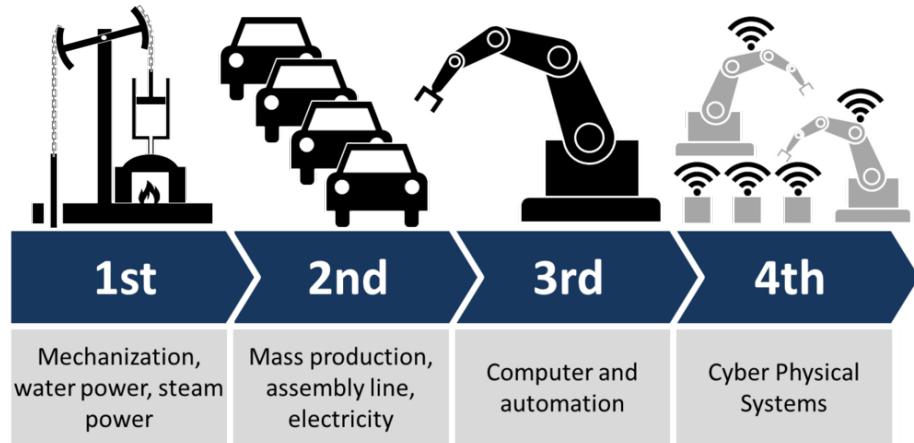


INDUSTRIA 4.0

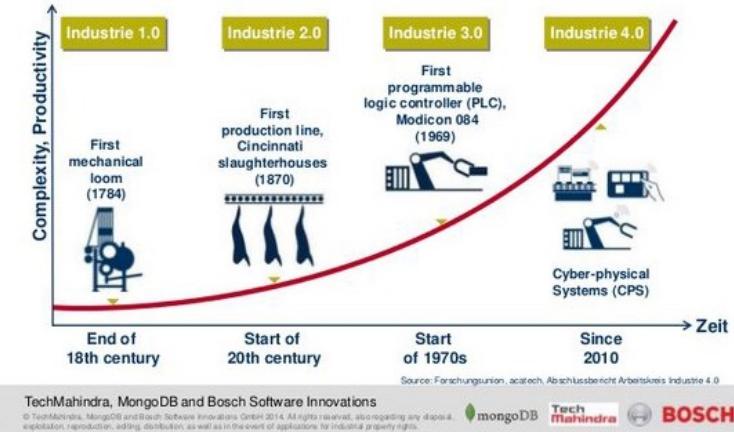
- Industry 4.0, Industrie 4.0 o quarta rivoluzione industriale
 - trend corrente in ambito automazione nell'industria manifatturiera
 - include **sistemi cyber-physical, Internet of Things (IoT) e cloud computing**
- Chiamato anche *smart factory*
 - fabbriche digitalizzate e interconnesse
 - **sistemi cyber-physical** monitorano i processi fisici, creando una rappresentazione virtuale del mondo fisici e permettendo decentralizzazione delle decisioni
 - sfruttando **IoT**, i sistemi cyber-physical comunicano e cooperano tra loro e con gli operatori umani in real time
 - sfruttando **Cloud** e Internet dei Servizi, vengono resi disponibili servizi sia all'interno dell'organizzazione, sia all'esterno in termini cross-organizational, per partner industriali che fan parte della value chain

https://en.wikipedia.org/wiki/Industry_4.0

INDUSTRIA 4.0 E SOCIETA' 4.0



Industrie 4.0: The next Industrial Revolution



- Questione importante
 - perdita posti di lavoro, nuove professionalità
 - aspetti etici

>> Society e Governance 4.0

<http://www.eesc.europa.eu/?i=portal.en.group-1-new-news.34501>



PHYSICAL COMPUTING

- Accezione usata per indicare sempre sistemi computazionale che interagiscono con il mondo fisico, tipicamente orientati in questo caso alla costruzione ed esplorazione di framework per *interazione human-machine*

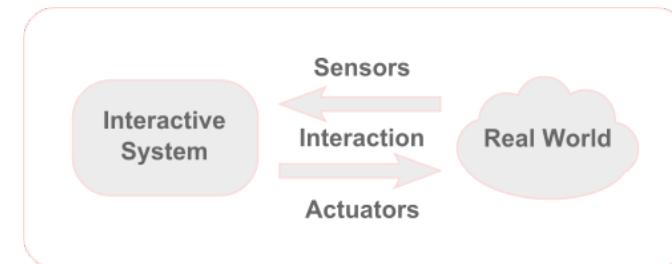
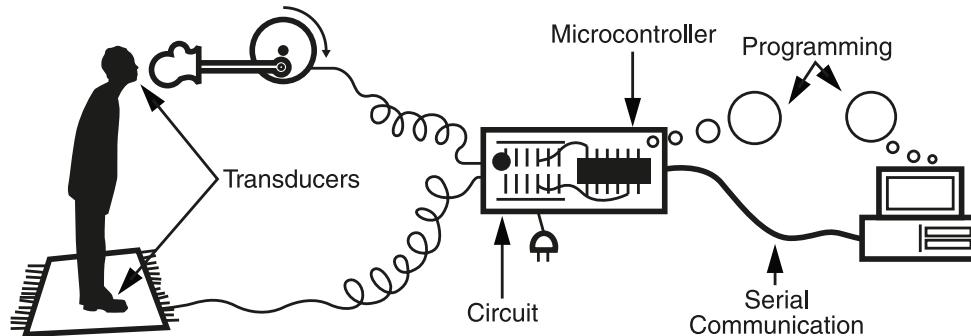


Figure I.4

The parts of a physical computing system.



(“Physical Computing”. Dan O’Sullivan, Tom Igoe. Thomson)

WEARABLE COMPUTING

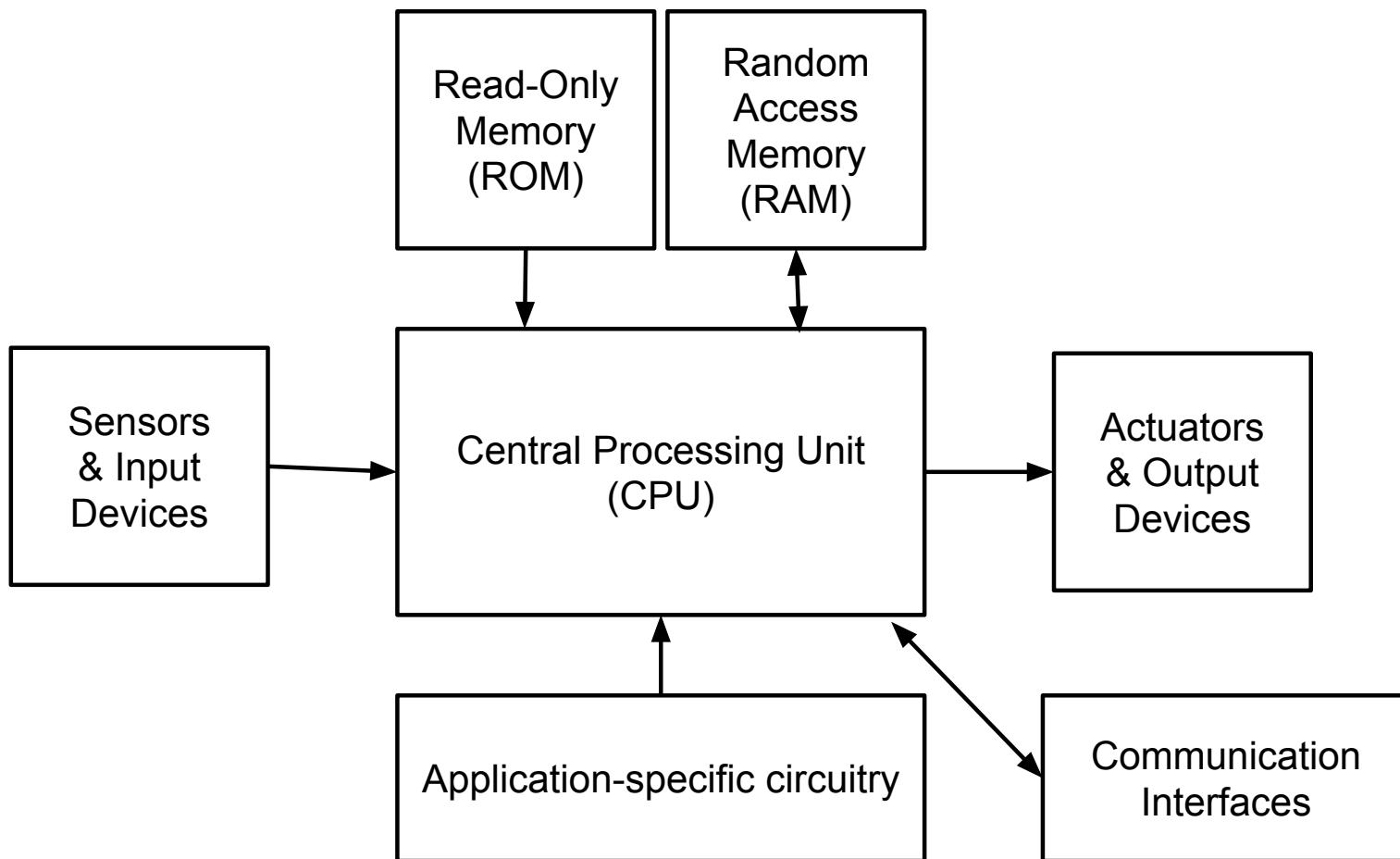
- Sistemi embedded *indossabili*
 - esempi
 - braccialetti (es: fitbit), smart-watch, smart-glasses
 - eyewear computing
 - basati su smartglass
- Tecnologie abilitanti per
 - sistemi **hands-free**
 - sistemi *realtà aumentata e mondi aumentati*
 - visione “human augmentation”



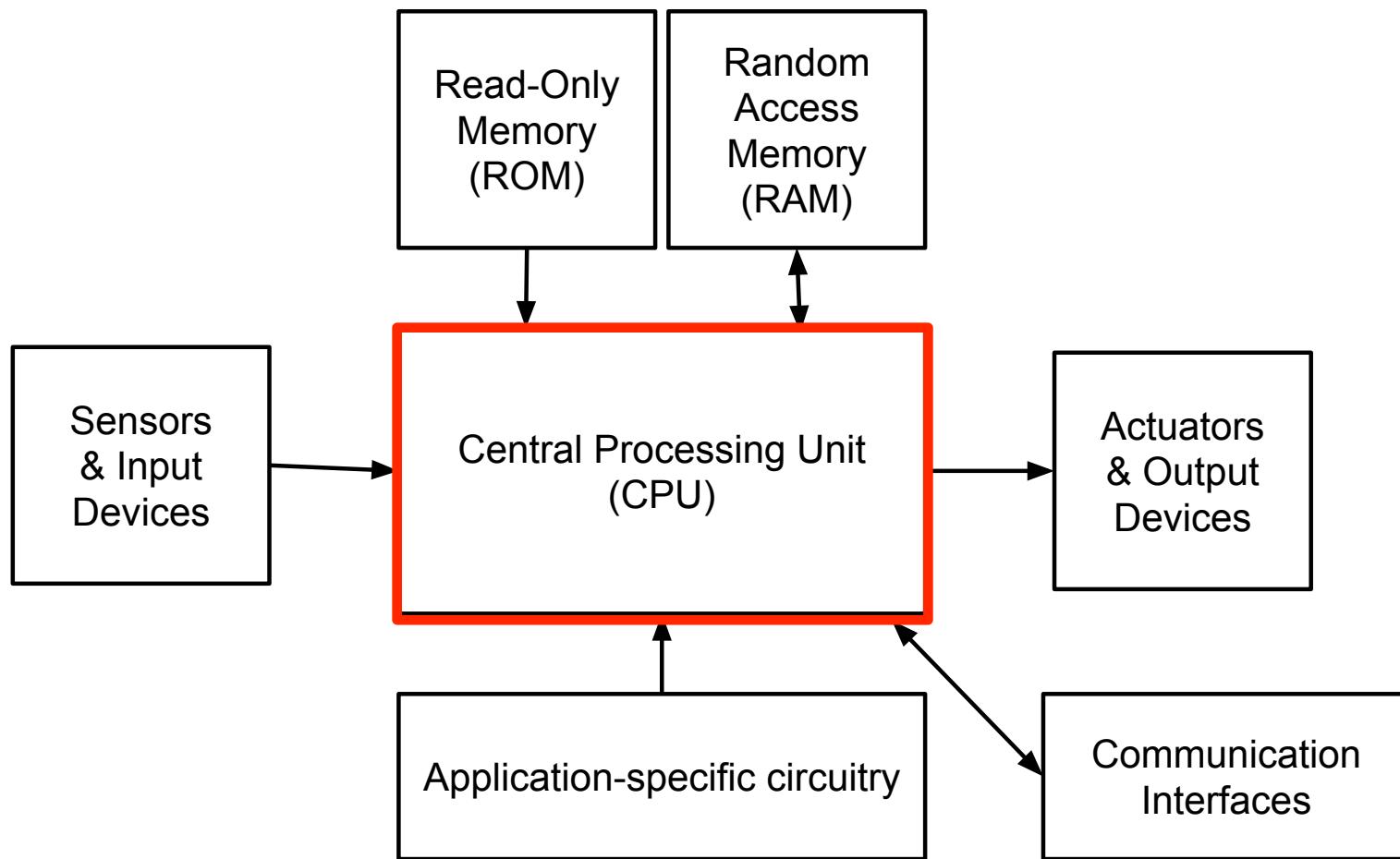
ARCHITETTURA SISTEMA EMBEDDED



ARCHITETTURA HARDWARE



ARCHITETTURA HARDWARE: LA CPU



PROCESSORI: TECNOLOGIE

- Per i sistemi embedded, esistono sostanzialmente tre tipi di tecnologie di processori
 - **processori general-purpose** (=> software)
 - processori che hanno una architettura e un insieme di istruzioni (instruction set architecture, ISA) predefinito e lo specifico comportamento è definito dal **programma (software)** in esecuzione
 - **processori single-purpose** completamente implementato via hardware
 - circuiti digitali progettati per implementare la specifica funzionalità o programma
 - **ASIC** = Application-Specific Integrated Circuit
 - **Application-Specific Processors (ASIP)**
 - una via intermedia
 - processori programmabili, ottimizzati per una specifica classe di applicazioni avente caratteristiche comuni
 - esempio: **micro-controllori (MCU)**, **SoC (System-on-a-Chip)**, **DSP (Digital Signal Processors)**
 - anche in questo caso la logica viene specificata a livello software

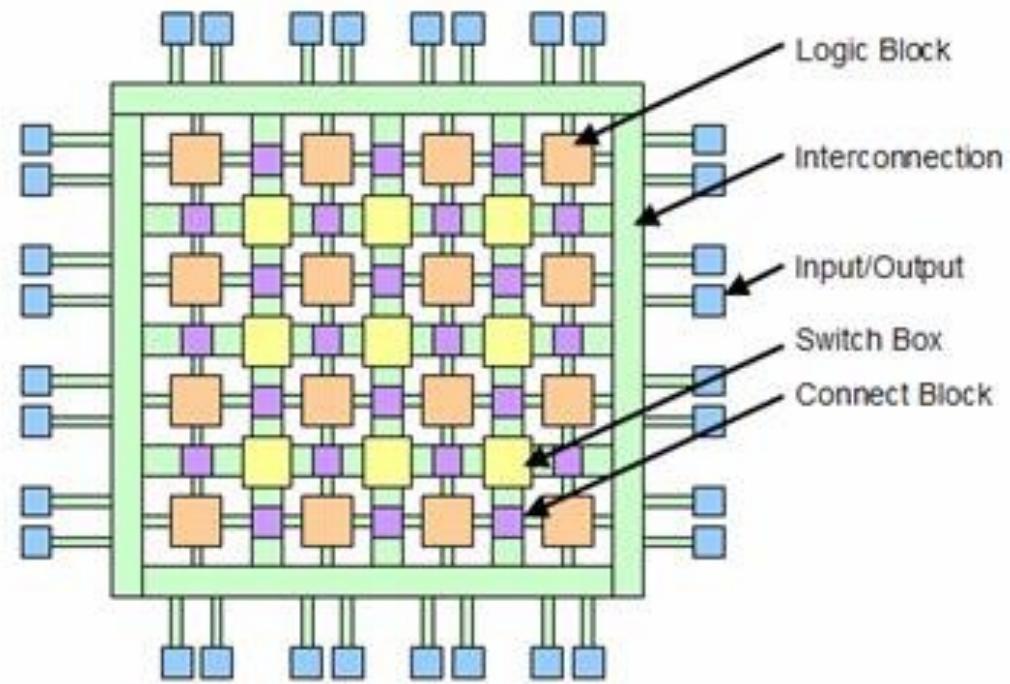
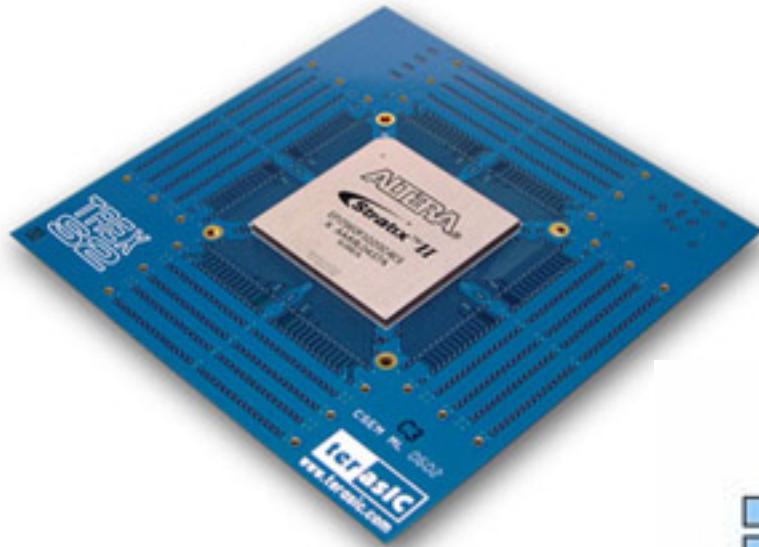
PROCESSORI SINGLE-PURPOSE

- Diversi tipi di implementazioni
 - **Full-Custom/VLSI**
 - in questo caso, l'implementazione avviene progettando l'intero processore in modo custom, ottimizzando tutti i livelli
 - **Semi-custom**
 - **ASIC** - application specific IC - in questo caso si parte da un certo insieme di livelli parzialmente o totalmente costruiti, implementandovi sopra il processore
 - **PLD - programmable logic device**
 - in questo caso tutti i livelli sono già presenti, quindi è possibile in questo caso acquistare un IC per poi “programmarlo” opportunamente.
 - i livelli implementano circuiti programmabili, dove la programmability è data dalla possibilità di creare/distruggere connessioni...

PROCESSORI SINGLE-PURPOSE

- PLA (Programmable Logic Array)
 - array programmabili di porte AND e array programmabili porte OR
- PAL (Programmable Array Logic)
 - solo un tipo di array
- **FPGA (Field Programmable Gate Array)**
 - circuiti integrati con funzionalità programmabili via software.
 - permettono un livello di connettività/programmabilità più generale rispetto a PLA e PAL
 - implementazione di funzioni logiche anche molto complesse, elevata scalabilità.
 - milioni di porte logiche per singolo dispositivo FPGA
 - *linguaggi di programmazione*: Verilog, **VHDL**
 - modalità di programmazione visuale schematic-entry
 - **LabView**

FPGA



PROGRAMMAZIONE FPGA

Xilinx - ISE - C:\Dokumente und Einstellungen\wfischer\mrc-nand\mrc-nand.ise - [mrc-nand.vhd*]

File Edit View Project Source Process Window Help

Sources Sources for: Synthesis/Implementation
mrc-nand xc3s50-5pq208 nand_gate - algorithm nand_gate - dataflow (mrc)

Processes Add Existing Source Create New Source View Design Summary Design Utilities User Constraints Synthesize -XST Implement Design Generate Programming File

```
30
31  Library IEEE;
32  use IEEE.std_logic_1164.all;
33
34
35  ENTITY nand_gate IS
36    port
37    (
38      a,b : in std_logic;
39      c : out std_logic
40    );
41  END nand_gate;
42
43
44  ARCHITECTURE dataflow OF nand_gate IS
45
46  BEGIN -- dataflow
47    c <= not ( a and b ) ;
48  END dataflow;
49
50
51  ARCHITECTURE dataflow_delay OF nand_gate IS
52
53  BEGIN -- dataflow_delay
54    c <= not ( a and b ) after 10 ns ;
55  END dataflow_delay;
56
57
58  ARCHITECTURE algorithmic OF nand_gate IS
59
60  BEGIN -- algorithmic
61    PROCESS (a,b)
62    BEGIN -- PROCESS
63      IF ( ( a='1' ) and ( b='1' ) ) THEN c<='0'; ELSE
64    END PROCESS;
65  END algorithmic;
```

Started : "Launching Design Summary".

Console Errors Warnings Find in Files

1000 ns

CompactRIO Sensor Demo (FPGA) vi Block Diagram

The block diagram is divided into several sections:

- Acquisition and Buffering:** Includes an Analog Input block connected to a Slot1/RIO-9215/A[0] block, which then connects to a Read Array block. The Read Array block has four outputs labeled Slot1/RIO-9215/A[0], Slot1/RIO-9215/A[1], Slot1/RIO-9215/A[2], and Slot1/RIO-9215/A[3]. These outputs feed into an FIFO block.
- Communication to Host:** Contains a FIFO Read block, a Timeout block, and an Empty block. It also includes a process for reading array data and a handshake mechanism involving a Wait for host to acknowledge data read block and a Handshake block.
- Read g-Range and Alarms:** This section includes logic for selecting g-range and triggering events. It features a Select g-Range for MMA7260Q Sensor block with two options: Slot2/RIO-genericTrigger (0300.0) and Slot2/RIO-generic/PC Adelay (0103). It also includes logic for reading AC and DC limits.

Module 1.1 Introduzione ai Sistemi Embedded

32

MICRO-CONTROLLORI (MICRO CONTROLLER UNIT, MCU)

- Dispositivi elettronici, nati come evoluzione alternativa ai *microprocessori integrando su singolo chip un sistema di componenti* che permette di avere la massima autosufficienza funzionale per applicazioni embedded
 - il processore, la memoria permanente, la memoria volatile e i canali (pin) di I/O, gestore interrupt, oltre ad eventuali altri blocchi specializzati
 - generalmente dotati di CPU CISC con architettura von Neumann, di recente sono apparsi microcontrollori con CPU ad architettura RISC, come ad esempio il Texas Instruments MSP430
 - a 8, 16, 32 bit
- Esempi
 - storici: MCU a 8 bit Motorola 68000 e Intel 8080 (1975), Zilog Z80, Intel 8051
 - più recenti: Atmel AVR, Texas Instruments MSP430, Microchip PIC16C84, ARM a 32 bit, Hitachi H8, PowerPC
- Mercato molto frammentato, con molti produttori e molte architetture, nessuna dominante

MICROCONTROLLORI - ESEMPI

- Lista: https://en.wikipedia.org/wiki/List_of_common_microcontrollers

TI MPS430	<ul style="list-style-type: none">Microcontrollore della Texas Instruments molto diffuso<ul style="list-style-type: none">16 bit, ultra-low-powerLaunchPad Development Kits<ul style="list-style-type: none">da 16 fino a 25 MHz, da 32 KB fino a 512 KB Flash, da 2 fino a 66 KB SRAM	 A red printed circuit board (PCB) labeled "EMULATION HSP-EXP430G2 Rev A". It features a central Texas Instruments MSP430 microcontroller chip. Various pins are labeled with their functions: UCC, P1.0 (LED1), P1.1 (TXD), P1.2 (RXD), P1.3 (S2), P1.4, P1.5, P1.6, P2.0, P2.1, P2.2, P2.3, P2.4, P2.5, P2.6, P2.7, P2.8, I2C Pin, 28pin, RESET, GND, and VCC. There are also several push buttons and LED indicators.
PIC 16505	PIC16C Microchip - MCU CMOS a 8 bit - 5 MIPS, 1,5 KB di memoria di programma con memoria dati fino a 72 byte di RAM, clock 4 MHz	 A black plastic package containing a PIC16C505 microcontroller. The package has a rectangular shape with a metal lead frame (MLF) and a "MICROCHIP" logo and part number "PIC16C505" printed on it.
ATMega328P	8 bit MCU, but instruction takes one or two 16-bit words, 2KB SRAM, 16KB FLASH, 16 MHz	 A black plastic package containing an ATMega328P microcontroller. The package has a rectangular shape with a metal lead frame (MLF) and a "ATMEGA" logo and part number "ATMega328P-PU" printed on it.

CONFRONTO MICROCONTROLLORI E MICROPROCESSORI

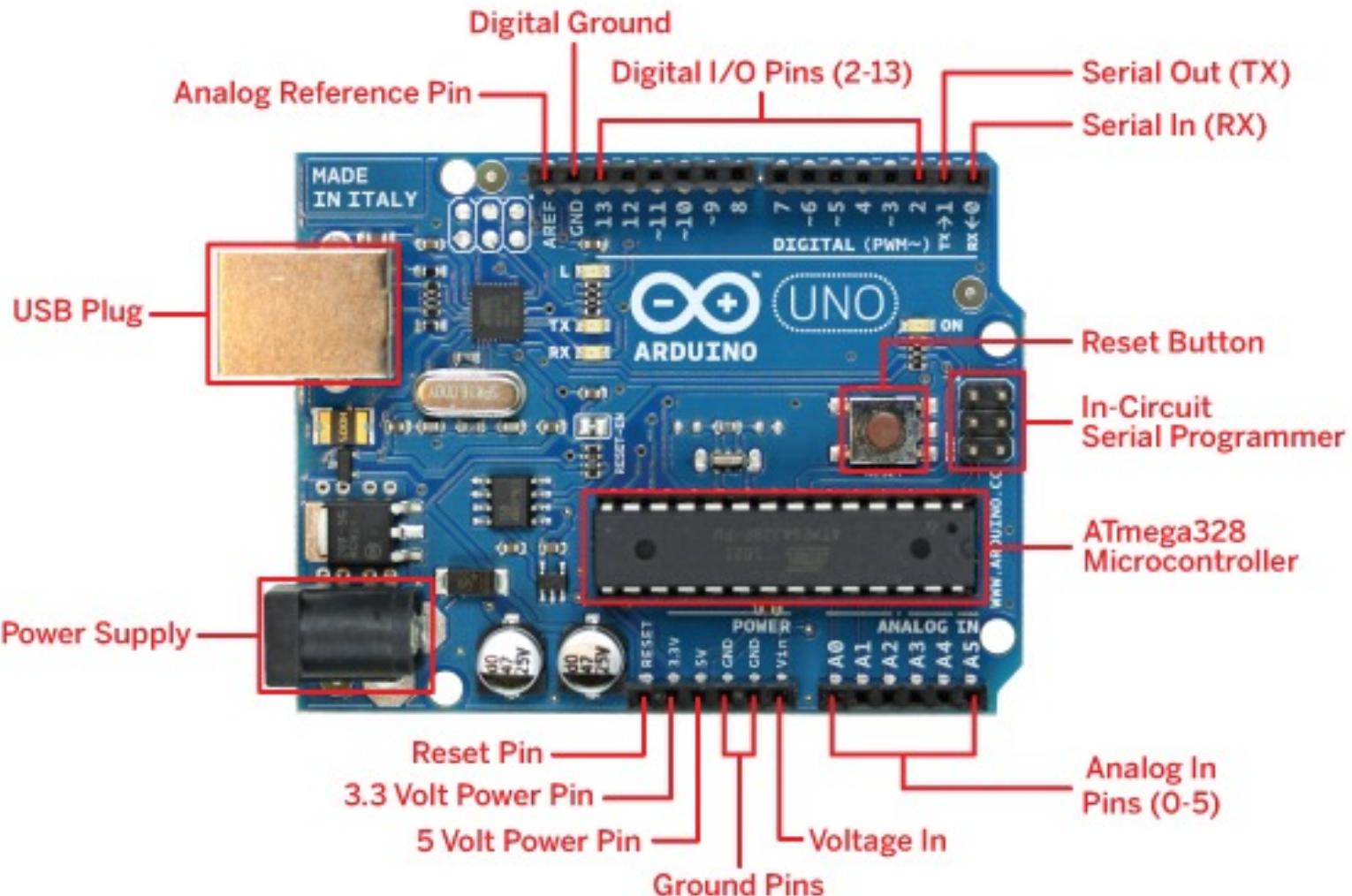
Caratteristica	Microcontrollore	Microprocessore
Velocità massima di clock	200 Mhz	4GHz
Capacità elaborativa massima in MegaFLOPS	200	5000
Potenza minima dissipata in watt (in stato di elaborazione)	1	50
Prezzo minimo per singola unità in USD	0.5	50
Numero pezzi venduti annualmente (in milioni)	11000	1000

<http://it.wikipedia.org/wiki/Microcontrollore>

SINGLE-BOARD MICRO-CONTROLLER

- Soluzioni che incorporano in unica scheda il microcontrollore *e tutta la circuiteria necessaria per eseguire dei compiti di controllo*
 - microprocessore, I/O, generatore di clock, RAM, memoria per contenere il programma, etc
 - la scheda può essere così immediatamente utilizzabile dallo sviluppatore di applicazioni, evitando di dover occuparsi della parte di progettazione e sviluppo dell'HW di controllo
- Esempi
 - A livello maker: Arduino
 - A livello industriale: Famiglia XMC

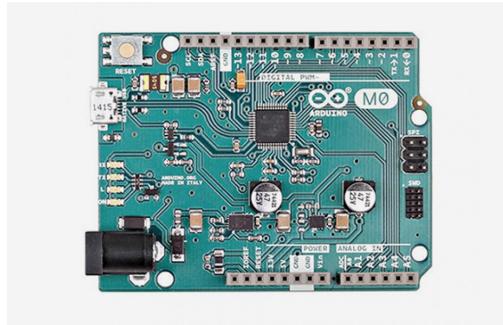
ARDUINO (UNO)



FAMIGLIA ARDUINO <http://arduino.cc>



Uno - 8 bit 16 Mhz, 2+32KB, 14+6 pin



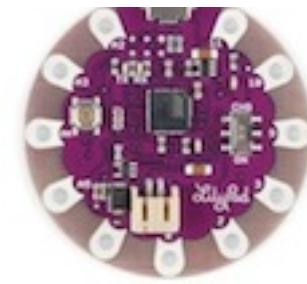
M0 - 32 bit 48 Mhz, 32+256KB



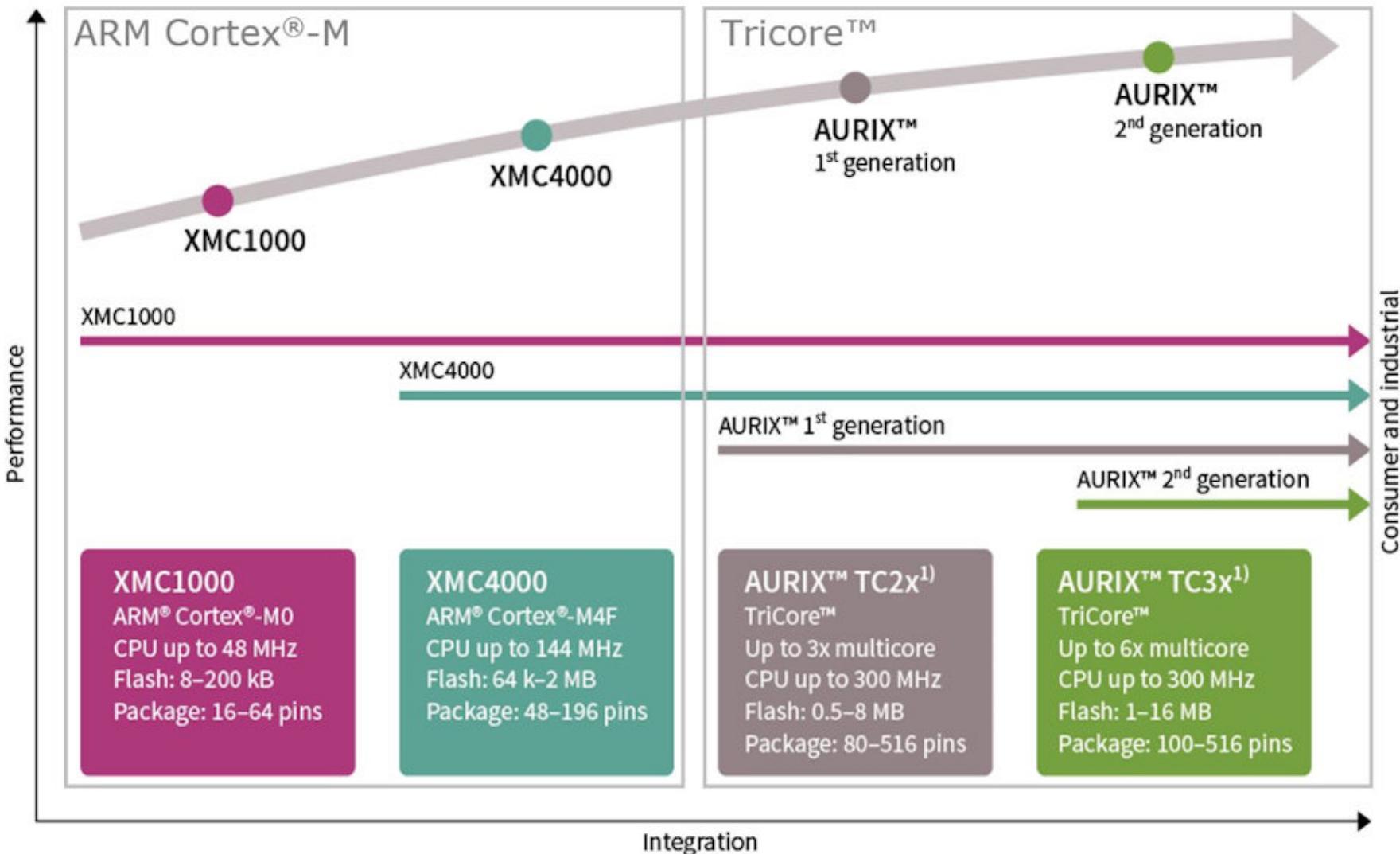
Mega - 8 bit 16 Mhz, 8+256KB, 54+16 pin



Yun - 32 bit 400 Mhz, 64+16MB



ESEMPIO INDUSTRIALE



<https://www.infineon.com/cms/en/product/microcontroller/32-bit-industrial-microcontroller-based-on-arm-cortex-m/>

SOC E SINGLE-BOARD CPU

- **SOC = System-on-a-Chip**
 - In questo caso è un chip stesso che incorpora un sistema completo, che include CPI, memoria, controllori, etc
 - tipicamente usati per creare delle single-board CPU
- Esempi:
 - BROADCOM BCM2837 64bit ARMv8 Cortex A53 Quad Core (1.2Ghz)
 - usato in Raspberry Pi 3
 - ARM Sitara AM335x SoC - including ARM Cortex-A8 processor
 - usato in BeagleBone, Arduino Tre
 - Texas Instruments OMAP3530
 - ...

FAMIGLIA RASPBERRY

Raspberry Pi (2012)

- SOC: Broadcom
BCM2835
- 700 MHz single-core **ARM1176JZF-S**
- 256 MB



Raspberry Pi 2 (2015)

- SOC: Broadcom
BCM2836
- 900 MHz 32-bit quad-core **ARM Cortex-A7** (900 Mhz, quad-core)
- 1 GB RAM

CPU:
ARM Cortex A7
(900 Mhz, quad-core)
1 GiB RAM

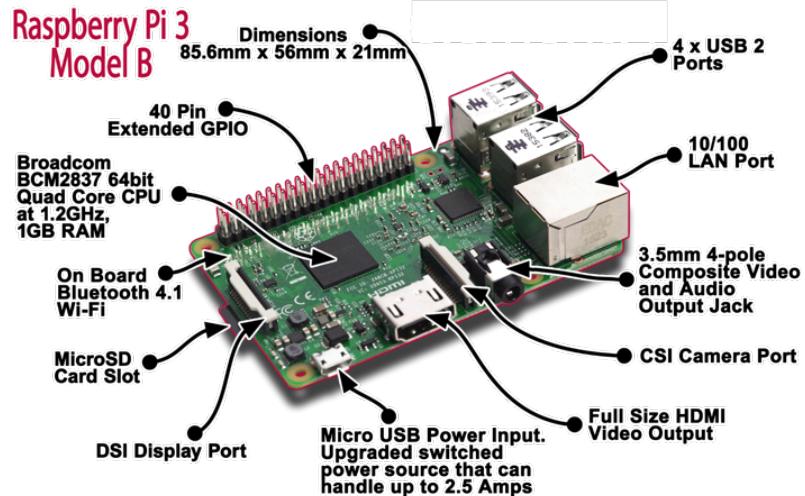


Full HDMI port

FAMIGLIA RASPBERRY

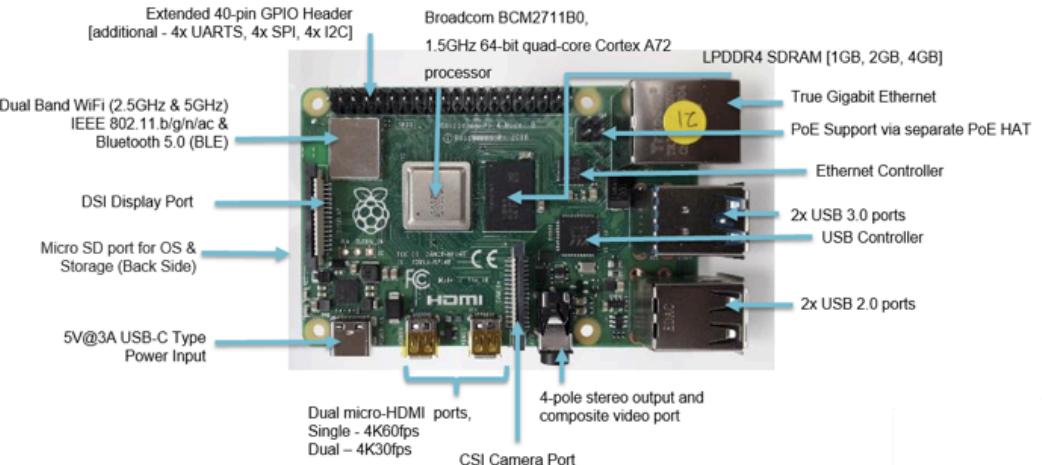
Raspberry Pi 3 (2016)

- Broadcom BCM2837
- 1.2 GHz 64-bit quad-core ARM Cortex-A53
- 1 GB RAM



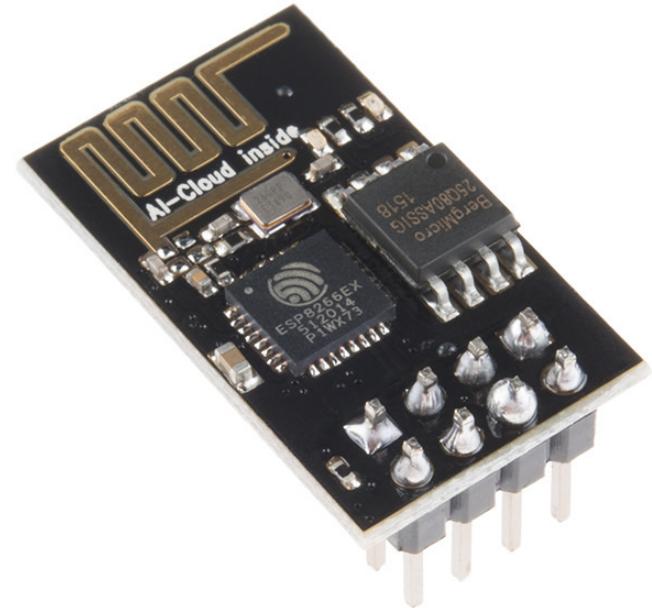
Raspberry Pi 4 (2019)

- Broadcom BCM2711
- 4x Cortex-A72 1.5 GHz
- 1,2, 4 GB RAM

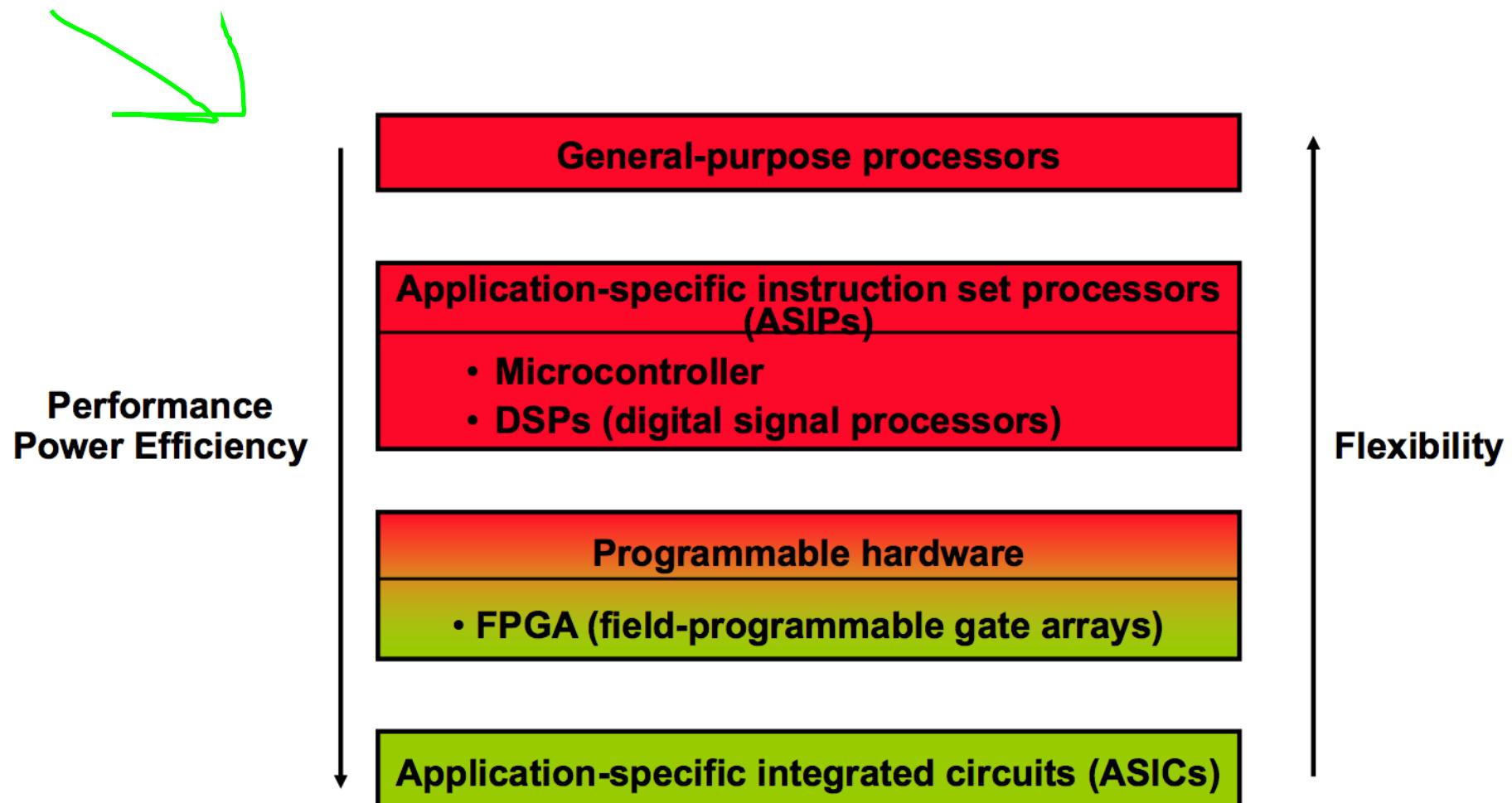


ESP 8266

- Dispositivo pensato per IoT
- Hardware
 - Processor: L106 32-bit RISC microprocessor core 80 MHz*
 - 64 KiB of instruction RAM, 96 KiB of data RAM
 - 16 GPIO pins
 - SPI, I²C, I²S + UART
 - 10-bit ADC
 - IEEE 802.11 b/g/n Wi-Fi
- SDK e firmware diversi, fra cui
 - GCC toolchain, Wiring / Arduino, NodeMCU Lua-based, MicroPython Python based, ESP-Open-RTOS Open source FreeRTOS-based...



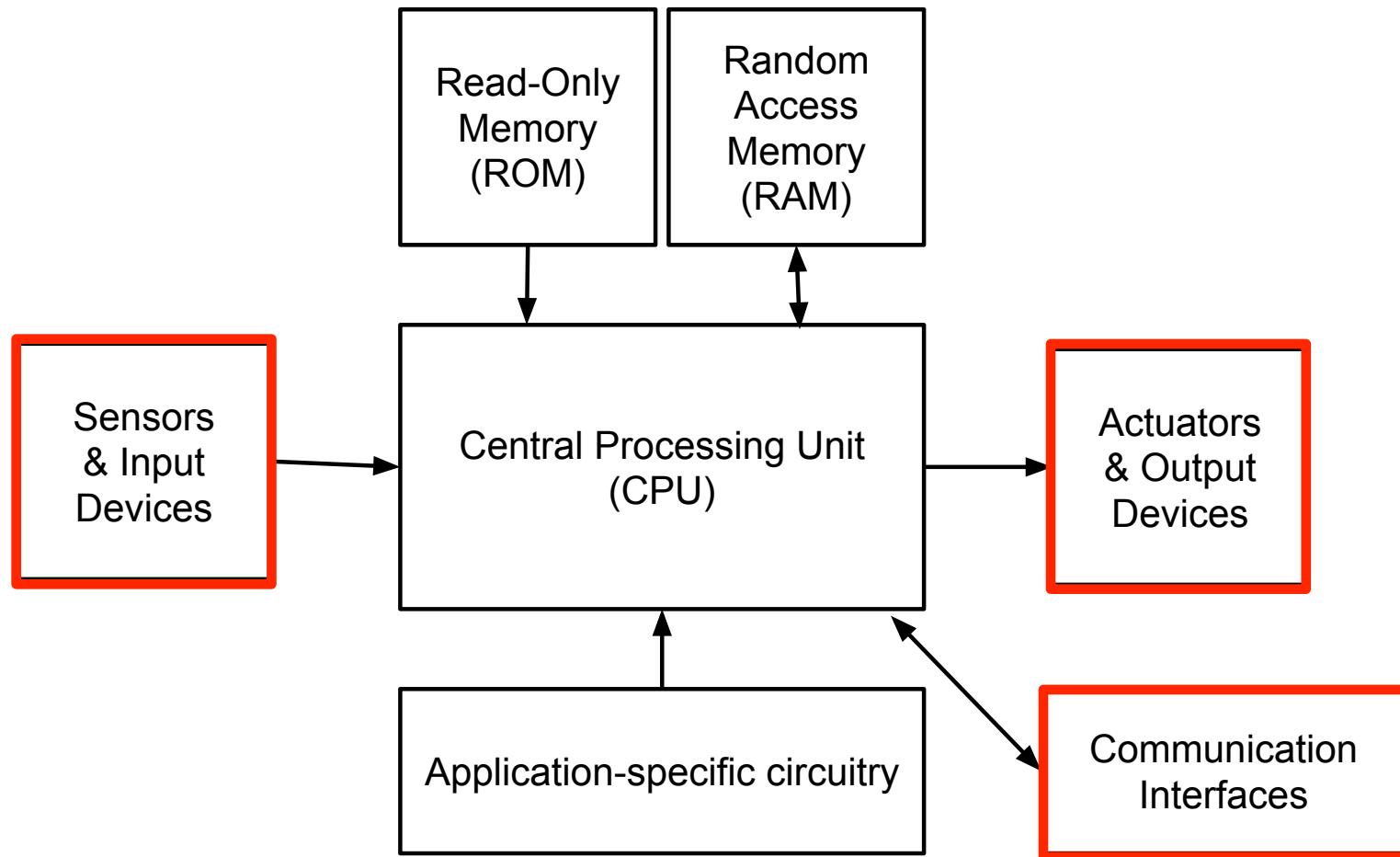
QUALE TECNOLOGIA SCEGLIERE



TENDENZA: MCU e SOC

- La scelta dipende dai requisiti del progetto ed è sempre un trade-off
- Detto questo, il continuo sviluppo tecnologico ha portato oggigiorno ad un utilizzo sempre più frequente e pervasivo di soluzioni basate su microprocessori/microcontrollori, SoC e single-board computers
 - facilita lo sviluppo di famiglie di prodotti che possono essere sviluppate per fornire tipi diversi di funzionalità, a costi diversi
 - flessibilità, efficienza, scalabile con la complessità
 - l'uso di processori con set predefinito di istruzioni spesso conduce ad una implementazione più rapida (che non l'uso di logica custom)
 - questo è dovuto al continuo progresso tecnologico relativamente alle CPU ad opera delle case produttrici
- **La programmabilità dei microcontrollori e microprocessori porta quindi benefici sostanziali a tutto il processo di progettazione e sviluppo**

ARCHITETTURA HARDWARE: SENSORI, ATTUATORI, BUS



SENSORI E ATTUATORI

- Un sistema embedded interagisce con l'ambiente in cui è situato mediante sensori e attuatori
- **Sensori**
 - dispositivi *trasduttori* che permettono di *misurare* una certo fenomeno fisico (come la temperatura, radiazioni, umidità, etc) o rilevare e quantificare una concentrazione chimica (es: il fumo)
 - fornisce una rappresentazione misurabile di un fenomeno su una certa specifica scala o intervallo
 - possono essere **analogici** o **digitali**
 - *analogici* => la grandezza elettrica prodotta in uscita - tensione o corrente - varia con continuità in risposta alla variazione della grandezza misurata
 - nel caso di sensori analogici, nel micro-controllore è tipicamente incluso *un convertitore analogico digitale* (ADC)
 - *digitali* => due soli valori o insieme discreto di valori
- **Attuatori**
 - dispositivi che producono un qualche effetto misurabile sull'ambiente, a partire da una specifica condizione o evento chiamato *trigger*

ESEMPI DI SENSORI PER ARDUINO



Sens. Prossimità ultrasuoni



Sens. Temperatura



GPS



Sens. PIR



Sens. luminosità



MPU-6050
3 assi Gyro+Accelerometro

BUS E PROTOCOLLI DI COMUNICAZIONE

- Interazione fra MCU/microprocessore e sensori/attuatori avviene mediante canali di comunicazione (bus) con specifici protocolli di comunicazione
- Nel caso dei sistemi embedded i protocolli più usati implementano una **trasmissione seriale**
 - ovvero: parole multi-bit vengono inviate sequenzialmente come serie di bit
- Esempi specifici rilevanti per sistemi embedded
 - I2C, SPI, JTAG
 - CAN-bus

UART (E USART)

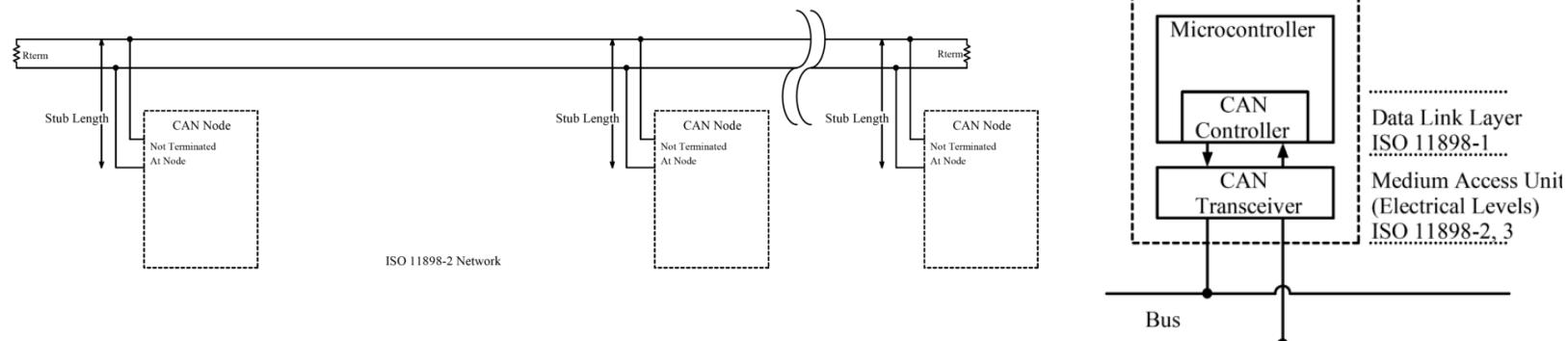
- Universal Asynchronous Receiver-Transmitter (ricevitore-trasmettitore asincrono universale)
 - il più datato dei protocolli seriali, supportato da qualsiasi MCU
 - consente di convertire flussi di bit di dati da un formato parallelo a un formato seriale asincrono o viceversa
 - di per sé UART non genera o riceve direttamente i segnali da convertire: questo tipicamente viene fatto da dispositivi di interfacciamento separati come RS-232, Bluetooth, USB,...
- USART (Universal Synchronous/Asynchronous Receiver-Transmitter) estende il protocollo con la trasmissione di un segnale di clock per la sincronizzazione

I²C, SPI e JTAG

- **I2C (Inter-Integrated Circuit)**
 - ulteriore protocollo/bus seriale *sincrono*, a due fili (uno per dati e uno per il clock)
 - denominato anche TW (Two Wire)
 - il più semplice da usare e più espandibile
- **SPI (Serial Peripheral Interface)**
 - protocollo/bus seriale *sincrono*, basato su master e slave
 - più veloce di I2C, meno semplice da usare
- **JTAG**
 - protocollo standard per il test funzionale di dispositivi
 - da usare in accoppiata con strumenti con il debugger

CAN-BUS

- Controller Area Network (CAN-bus)
 - protocollo standard seriale per bus usati nell'ambito automotive e in contesti industriali, introdotto da Bosch
 - pensato per collegare varie unità di controllo
 - progettato per funzionare in ambienti disturbati dalla presenza di onde elettromagnetiche
 - basato su scambio di messaggi in **multicast**
- Architettura



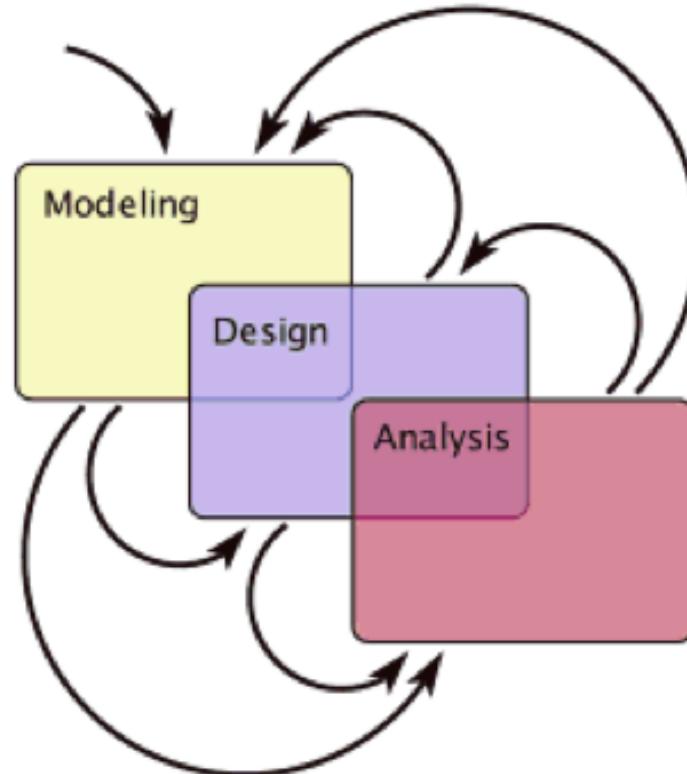
- Il nodo può essere anche un sensore/attuatore + host processor

INTERFACCE E TECNOLOGIE DI COMUNICAZIONE

- Tecnologie e standard che permettono la comunicazione wired e wireless con altri sistemi (embedded e non)
- Wireless in particolare
 - **Bluetooth** and Bluetooth Low-Energy (**BLE**)
 - ZigBee
 - Z-Wave
 - LoRoWan
 - Wifi
 - ...

PROGETTAZIONE SISTEMI EMBEDDED

- Processo iterativo
 - **modeling**
 - **design**
 - **analysis**



(IES book, p. 8)

PROGETTAZIONE SISTEMI EMBEDDED: PROCESSO

- **Modeling / Modellazione**

- processo finalizzato ad ottenere un'approfondita comprensione o conoscenza relativamente al sistema da costruire
- tale conoscenza è rappresentata da *modelli*, che sono il risultato di questo processo
- rappresentano cosa il sistema deve fare

- **Design / Progettazione**

- processo finalizzato alla creazione degli artefatti tecnologici che rappresentano il sistema
- rappresentano come il sistema fa quello che deve fare
- la modellazione può avere un ruolo importante anche in questa fase
 - fornendo una descrizione astratta di come funziona il sistema, che prescinde dalla specifica implementazione

- **Analysis / Analisi**

- processo finalizzato ad ottenere un'approfondita comprensione e conoscenza del comportamento del sistema
- specifica perché un sistema fa quello che fa (o non fa quello che dovrebbe fare)

MODELLAZIONE

- **Modello** = descrizione degli aspetti rilevanti del sistema, utili la comprensione di proprietà del sistema stesso e della sua dinamica in particolare
 - dinamica = evoluzione nel tempo
- Sistemi embedded e CPS sono sistemi composti da sottosistemi fisici integrati con sistemi computazioni e di rete => un modello di un sistema embedded include tutte le tre parti (*physical, computing, networking*)
 - modello della parte fisica
 - modello della parte logica
 - software, algorithms

MODELLAZIONE DEL COMPORTAMENTO DINAMICO

- *Modellazione della parte fisica => modellazione di comportamenti dinamici **tempo-continui***
 - modelli matematici
 - teoria del controllo
 - oggetto di studio di corsi di controlli e automazione
- *Modellazione della parte logica => modellazione di comportamenti dinamici **tempo-discreti***
 - **macchine a stati** e loro composizione
 - modelli **concorrenti**
 - oggetto di studio di questo corso
- Modellazione di sistemi ibridi
 - modelli ad attori (libro IES)

DESIGN/PROGETTAZIONE E SVILUPPO

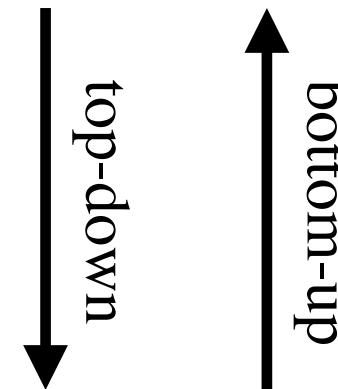
- Progettazione del sistema embedded
 - scegliere la tecnologia e architettura HW
 - +
 - scegliere l'architettura software più appropriata

ANALYSIS

- Ogni sistema è progettato per soddisfare certi requisiti
 - ciò è particolarmente vero per sistemi critici (*safety-critical systems*)
- I requisiti di un sistema sono espressi in termini di proprietà o specifiche, espresse in qualche linguaggio rigoroso (e formale)
 - “*a design without specification cannot be right or wrong, it can be only surprising!*” (Young et al., 1985)
- Strumenti
 - linguaggi formali per esprimere in modo rigoroso le proprietà
 - logica temporale
 - tecniche per confrontare le specifiche
 - tecniche per analizzare le specifiche e il risultato della fase di design
- Aspetto non approfondito in questo corso
 - contenuti affrontati in corsi della magistrale

TOP-DOWN AND BOTTOM-UP DESIGN

- Il design può essere top-down e bottom up, attraverso livelli diversi
 - **requisiti**
 - **specifiche** (Specification)
 - **architettura**
 - **componenti**
 - **integrazione di sistema**



REQUISITI

- Raccolta della descrizione informale da parte del customer circa il sistema e cosa si supponga che faccia
 - requisiti ***funzionali***
 - funzioni che deve svolgere il sistema
 - requisiti ***non-funzionali***
 - performance, costo (manufacturing and NRE), dimensioni fisiche, peso, consumo energia e potenza,..
- Uso di *requirement forms*
 - nome, scopo, inputs, outputs, funzioni, performance, costo di fabbricazione, consumo potenza, dimensioni

SPECIFICHE

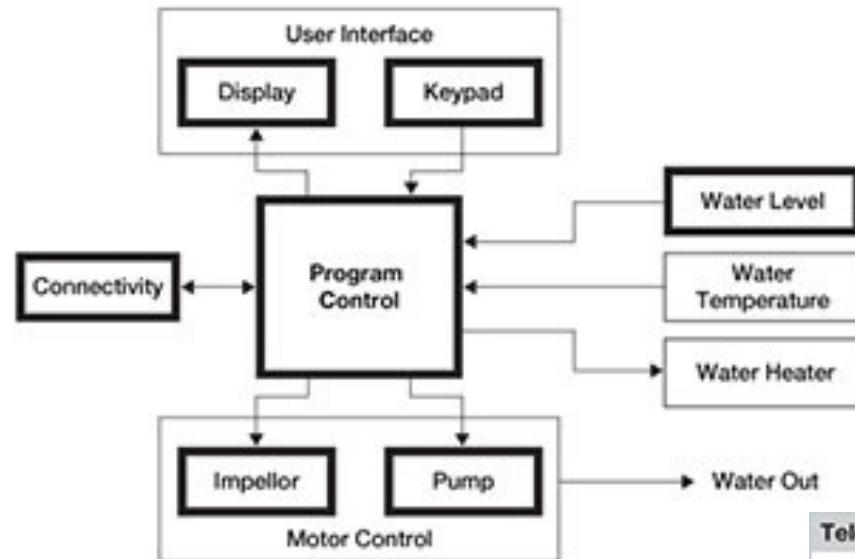
- Descrizione più precisa e rigorosa dei requisiti
 - serve come *contratto* fra cliente e progettisti
- Dovrebbe essere sufficientemente comprensibile da poter verificare se soddisfa i requisiti e le aspettative del cliente
 - non ambiguo
- **Linguaggio UML**, linguaggi formali
 - basati ad esempio sulla logica

PROGETTAZIONE ARCHITETTURA

- Descrive come il sistema implementa le funzioni descritte nella specifiche
- **Architettura**
 - piano della struttura complessiva del sistema, che verrà usata poi per guidare lo sviluppo dei componenti
 - descrive quindi quali componenti servono e come essi interagiscono
- **Diagrammi a Blocchi**
 - hardware and software

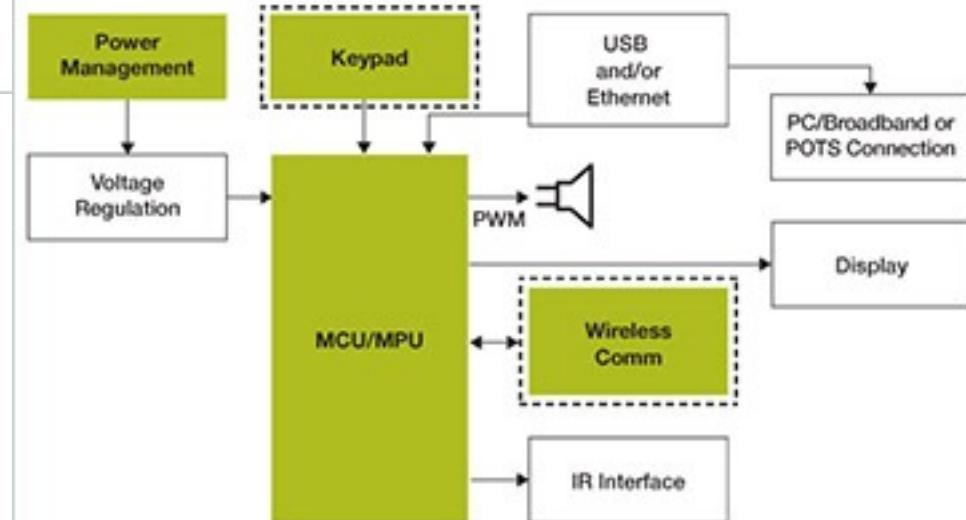
DIAGRAMMI A BLOCCHI

Dishwasher



■ Freescale Technology

Telehealth Gateway



PROGETTAZIONE COMPONENTI

- Progettazione e sviluppo dei singoli componenti
- Sia della parte software e hardware
 - ad esempio: FPGA, boards, SW modules
- Alcuni componenti possono essere ready-made, ovvero già disponibili
 - CPU, GPS receivers, topographic databases, services..

INTEGRAZIONE DI SISTEMA

- Integrazione dei componenti seguendo l'architettura
- Parte critica del design

SVILUPPO E PROGRAMMAZIONE

- Programmazione e sviluppo del sistema software embedded (sintesi)
 - manuale o *model-driven*
- Avviene su un computer host
 - tool-chain per la creazione dell'eseguibile, che viene trasferito e mandato in esecuzione poi sul sistema embedded
- Linguaggi/piattaforme di alto livello
 - linguaggio C/C++ e librerie di sistema
 - usati in modo pervasivo in soluzioni basate su microcontrollori
 - su sistemi embedded con sistema operativo => diffuso utilizzo di linguaggi/piattaforme di più alto livello, con architettura runtime
 - linguaggi: Java, C#, JavaScript, Python, ...
 - piattaforme: Embedded Java, Android,...

DEPLOYMENT, DEBUGGING e TESTING

- Lo sviluppo del software avviene tipicamente su un computer host
 - upload del programma compilato sul sistema embedded target ed esecuzione
- Supporti per debugging
 - collegamento seriale per scrittura di messaggi relativi allo stdout sul computer host
 - interfaccia/protocollo JTAG

PUNTO IMPORTANTE DEL CORSO

- **Importanza della modellazione e progettazione**
 - ogni sistema che non sia banale richiede una opportuna fase di modellazione e progettazione prima di passare alla pura implementazione
 - richiamo ai principi di base dell'*ingegneria del software*
- L'implementazione deve esser sviluppata il più possibile mantenendo **il livello di astrazione** individuato in fase di modellazione
 - non dall'hardware sottostante
 - vari vantaggi
 - maggiore riusabilità, comprensibilità, manutenibilità,
 - ...
- Aspetto sviluppato e approfondito in futuri moduli del corso

BIBLIOGRAFIA E SITOGRAFIA

- Wayne Wolf. *Computers as Components - Principles of Embedded Computer System Design*. Morgan Kaufman
 - chapter 1
- [IES Book] Ed Lee and S. Seshia. *Introduction to Embedded Systems. A Cyber-Physical Systems Approach*. UC Berkley
 - chapter 1
- Frank Vahid and Tony Givargis. *Embedded System Design - A unified Hardware/Software perspective - Introduction*. Wiley
 - chapter 1

Sistemi Embedded e IoT

Ingegneria e Scienze Informatiche - UNIBO

a.a 2020/2021

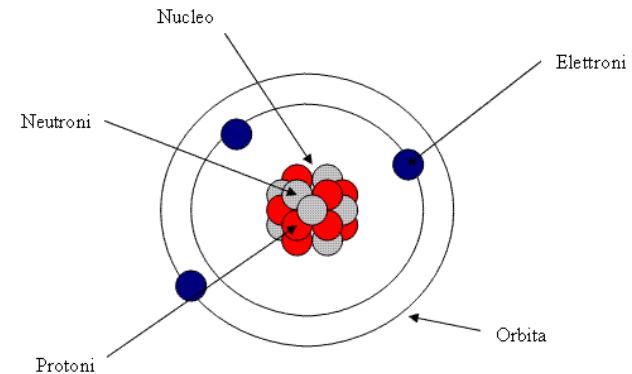
Docente: Prof. Alessandro Ricci

[modulo-lab-1.1]

CENNI DI ELETTRONICA
DI BASE

CARICA ELETTRICA

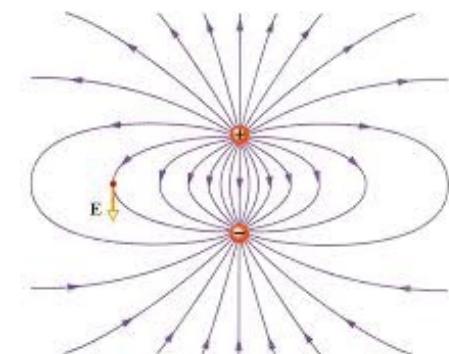
- Modello elementare dell'atomo
 - **elettroni** - particelle con **carica negativa**
 - **protoni** - particelle con carica positiva
 - neutroni - senza carica
- La carica dell'elettrone è pari a quella del protone (con segno opposto), e si indica con Q e si misura in **Coulomb**. Per definizione: 1 C è *la carica elettrica posseduta da 6.25×10^{18} elettroni*
- Nel loro stato normale, gli atomi hanno un pari numero di elettroni e protoni (*numero atomico*). In tutti i materiali esistono tuttavia **elettroni liberi**
 - elettroni che si trovano nel livello più esterno e che possono liberarsi dall'atomo a fronte di energia che prendono dall'ambiente
- Materiali **conduttori, semiconduttori, isolanti**
 - i materiali definiti *conduttori* (es: argento, rame) hanno un gran numero di elettroni liberi. I materiali definiti *semi-conduttori* (es: silicio) hanno un numero inferiore. I materiali isolanti hanno un numero molto ridotto di elettroni liberi.



CAMPO ELETTRICO

- Quando in un materiale esiste *un eccesso di elettroni*, vi è una carica elettrica complessivamente negativa; quando esiste un eccesso di protoni, vi è una carica elettrica positiva.
 - *l'elettricità statica* è la presenza di una carica netta positiva o negativa su un materiale
- Esiste un **campo elettrico** che esercita una **forza** fra cariche
 - per cui materiali aventi cariche elettriche di segno opposto si attraggono reciprocamente, mentre quelli dello stesso segno si respingono
- Cariche positive o negative si possono creare a partire da atomi neutri sottoposti ad una certa energia (ad esempio, via reazioni chimiche) per cui elettroni di valenza (più esterni) sfuggono e l'atomo rimane con carica netta positiva - detto *ione positivo*.
 - se un atomo acquista elettroni, diventa uno *ione negativo*

$$\mathbf{E} = \lim_{q \rightarrow 0} \frac{\mathbf{F}}{q}$$



LA TENSIONE

- Data la forza di attrazione che esiste fra due cariche positiva e negativa, per portare tali cariche ad una certa distanza *bisogna spendere una certa quantità di energia (lavoro)*
 - tutte le cariche di segno opposto posseggono una certa quantità di *energia potenziale* dovuta alla distanza che le separa.
- La differenza fra l'energia potenziale fra le cariche è detta **differenza di potenziale**, o **tensione (V)**, ed è espressa come energia (W) per unità di carica (Q):

$$V ::= W/Q$$

$$V_b - V_a = - \int_a^b \vec{E} \cdot d\vec{l}$$

la d.d.p. tra due cariche distanti (che si attirerebbero), una volta collegate queste ultime con un cavo conduttore, permette la loro attrazione (creazione di corrente)

espresso in **Volt**, ove W è espresso in Joule e Q è espresso in Coulomb

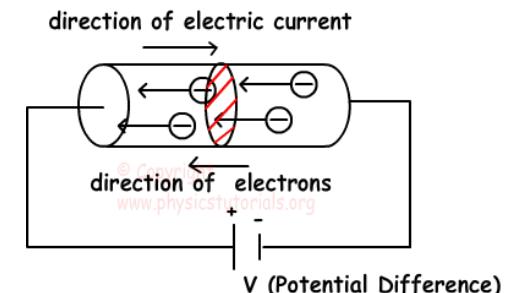
- *1 Volt è la tensione (o d.d.p) tra due punti quando è impiegata l'energia di un Joule per spostare la carica di un Coulomb da un punto all'altro.*

GENERATORI DI TENSIONE

- Un **generatore di tensione** è una sorgente di energia potenziale che viene detta anche **forza elettromotrice** (f.e.m)
 - è il lavoro per unità di carica compiuto dalla batteria per muovere le cariche dal polo a basso potenziale al polo a potenziale più alto
 - essa è numericamente uguale alla differenza di potenziale massima ai capi di un generatore elettrico sconnesso dal circuito elettrico
- Esempi:
 - *batteria*
 - è un generatore di tensione che trasforma energia chimica in elettrica
 - *alimentatore elettronico*
 - generatore che trasforma la tensione alternata delle prese elettriche in tensione continua
 - *celle solare*
 - generatore che trasforma energia della luce in elettrica (azione fotovoltaica)
 - *generatori*
 - trasformano energia meccanica in elettrica, sfruttando il principio di induzione elettromagnetica

LA CORRENTE ELETTRICA

- In tutti i materiali esistono elettroni liberi, che si muovono casualmente in tutte le direzioni, passando da un atomo all'altro
- Se applichiamo una tensione agli estremi di un materiale conduttore o semiconduttore, un estremo diventa positivo e l'altro negativo
 - le forze di repulsione e attrazione *determina un movimento di elettroni liberi dall'estremo negativo del materiale all'estremo positivo.*
 - questo fenomeno prende il nome di **corrente elettrica** - simbolo **I**
 - moto ordinato di carica elettrica dal + al -
- La corrente elettrica è definita come *l'intensità del flusso degli elettroni in un materiale conduttore o semiconduttore* ed è misurata dal numero di elettroni che attraversano un dato punto nell'unità di tempo:



$$I ::= Q/t$$

$$I = \lim_{\Delta t \rightarrow 0} \frac{\Delta Q}{\Delta t} = \frac{dQ}{dt}$$

La corrente si misura in **Ampère (A)**.

- 1 A è la quantità di corrente che fluisce quando un numero di elettroni aventi la carica di 1 C attraversano un punto determinato nel tempo di 1 secondo.

IL CONCETTO DI POTENZA

- La potenza (P) è in generale *l'intensità con cui l'energia viene impiegata*. Rappresenta la quantità di energia impiegata in un certo intervallo di tempo:

$$P ::= W / t$$

si misura in **Watt**, come Joule/secondi.

- Per definizione: *1 Watt è la quantità di potenza che si ha quando l'energia di un Joule viene consumata in 1 secondo*
- Il **consumo di energia** si può quindi ricavare come la potenza utilizzata in un certo periodo di tempo
 - $W = P*t$
 - modo alternativo di esprimere l'energia rispetto al Joule è il wattsecondo (Ws), wattora (Wh), kilowattora (kWh).
 - es: una lampadina da 100W che rimane accesa per 10 ore consuma 1 kWh.

CORRENTE E POTENZA

- Ricapitolando
 - se nel conduttore vi sono cariche elettriche, la forza generata dal campo elettrico è direttamente proporzionale alla differenza tra l'energia potenziale delle cariche nei due punti.
 - il moto ordinato di carica è quindi dovuto al fatto che le cariche *minimizzano* la loro energia potenziale, spostandosi dal punto a potenziale maggiore al punto a potenziale minore.
- Il **campo elettrico** nel conduttore compie pertanto un **lavoro** sulle cariche, *realizzando un trasferimento di potenza dal campo alle cariche in moto*

$$dL = dq\Delta V = I\Delta V dt$$

- La **potenza** è il lavoro sviluppato nell'unità di tempo. La potenza sviluppata dal campo elettrico è quindi:

$$P = \frac{dL}{dt} = I\Delta V$$

CORRENTE CONTINUA E ALTERNATA

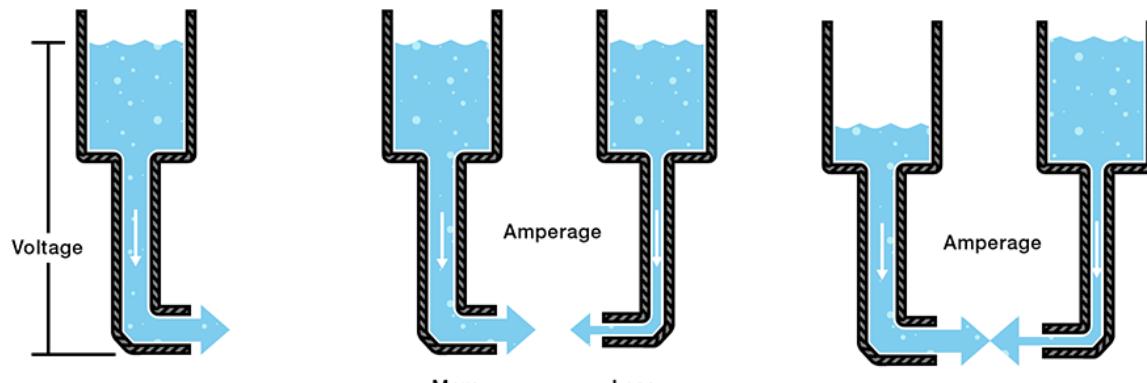
- **Corrente Continua** (CC o DC, Direct Current):
 - corrente il cui verso non varia nel tempo
 - è la corrente prodotta dalle batterie, nei dispositivi elettronici
- **Corrente Alternata** (CA o AC, Alternating Current):
 - il verso della corrente varia periodicamente nel tempo
 - legge: $I = I_0 \sin(2\pi f*t)$, dove f è la frequenza.
 - È la corrente prodotta dalle centrali elettriche, con frequenza $f = 50$ Hz in Europa, $f = 60$ Hz negli Stati Uniti

LA RESISTENZA

- Quando la corrente elettrica attraversa un materiale, gli elettroni liberi si muovono attraverso il materiale urtando occasionalmente contro gli atomi
 - le collisioni provocano una parziale perdita dell'energia posseduta dagli elettroni
- La proprietà di un materiale di limitare il passaggio degli elettroni è detta **resistenza (R)**
 - la resistenza è l'opposizione al passaggio della corrente. I conduttori hanno una resistenza molto piccola.
- L'unità di misura della resistenza è **l'Ohm**.
 - *si ha la resistenza di 1 Ohm quando un conduttore è attraversato dalla corrente di 1 ampere (A) con una tensione applicata di 1 volt*
- Inverso della **conduttanza (G)** - $G = 1/R$
 - espressione quantitativa di un conduttore ad essere percorso da corrente elettrica. Si misura in siemens (S)

ANALOGIA “IDRAULICA”

- Carica => acqua
- Tensione => pressione
- Corrente => flusso



(batteria)

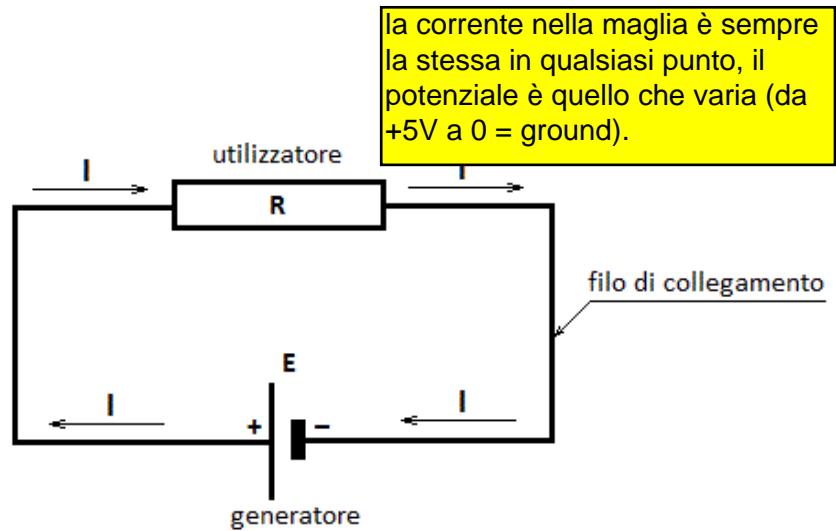
Resistance

Less resistance

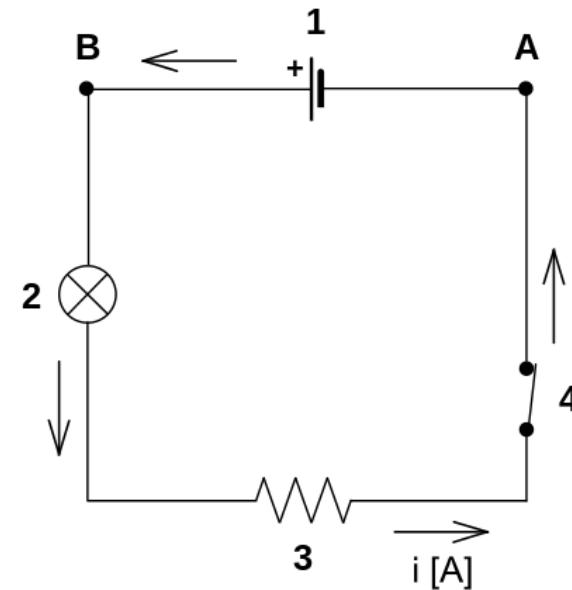
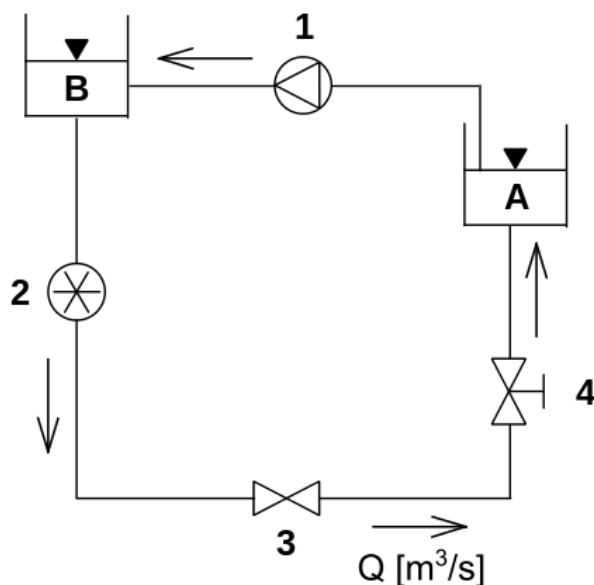
More resistance

CIRCUITO ELETTRICO

- Costituito essenzialmente da
 - uno o più generatori di tensione
 - uno o più **carichi** (componenti) elettrici/elettronici
 - un collegamento affinché la corrente possa fluire
- Lo *schema elettrico* è una rappresentazione con segni grafici standard del circuito
- Affinché la corrente possa fluire il circuito deve essere **chiuso**
 - ovvero il percorso/collegamento deve essere completo.
 - se è interrotto, il *circuito* è *aperto* e la corrente non fluisce.
- I componenti sono collegati secondo schemi che rispettano le specifiche e **polarità** dei vari componenti
 - **verso in cui deve scorrere la corrente**



ANALOGIA “IDRAULICA”



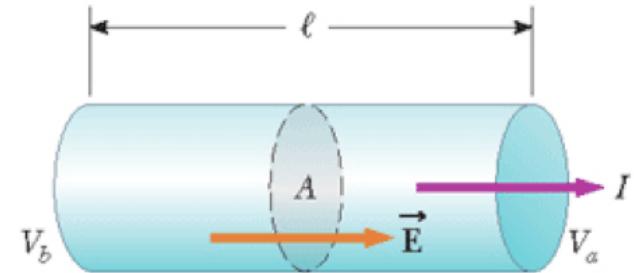
Analogia tra un **circuito idraulico** (a sinistra) e un **circuito elettrico** (a destra): la differenza di potenziale elettrico tra i due punti A e B del circuito elettrico è analoga alla differenza di pressione tra i due punti A e B del circuito idraulico corrispondente.

Nella figura sono indicati inoltre i seguenti dispositivi, tra loro analoghi:

- 1 - pompa idraulica / generatore di tensione;
- 2 - turbina / lampadina;
- 3 - valvola di laminazione / resistore;
- 4 - valvola di chiusura / interruttore.

LEGGE DI OHM

- Fra i capi di un conduttore affinché ci sia un campo elettrico E che causa una corrente, ci deve essere una differenza di potenziale $V = V_b - V_a$
- La relazione fra differenza di potenziale V e corrente I dipende dal materiale e dalle condizioni in cui è usato
- Per un gran numero di casi vale la **Legge di Ohm: $V = I \cdot R$**
 - dove R è la resistenza
 - dipende dal materiale e dalla geometria del conduttore.
 - come detto in precedenza, la resistenza R si misura in V/A , ovvero **Ohm (Ω)**: $1 \Omega = 1 V/A$.
- Quindi
 - in un circuito con resistenza costante, all'aumentare/diminuire della tensione, aumenterà/diminuirà linearmente anche la corrente
 - in un circuito a tensione costante, aumentando/diminuendo la resistenza, diminuirà/aumenterà la corrente.



LA POTENZA IN UN CIRCUITO ELETTRICO (EFFETTO JOULE)

- Quando la corrente fluisce in un materiale conduttore con una certa resistenza, le collisioni contro gli atomi sviluppano calore con il risultato di una perdita di energia.
- In una resistenza che segue la legge di Ohm la potenza dissipata è data dalla formula:

$$P = V * I = I^2 * R = V^2 / R$$

COMPONENTI ELETTRONICI

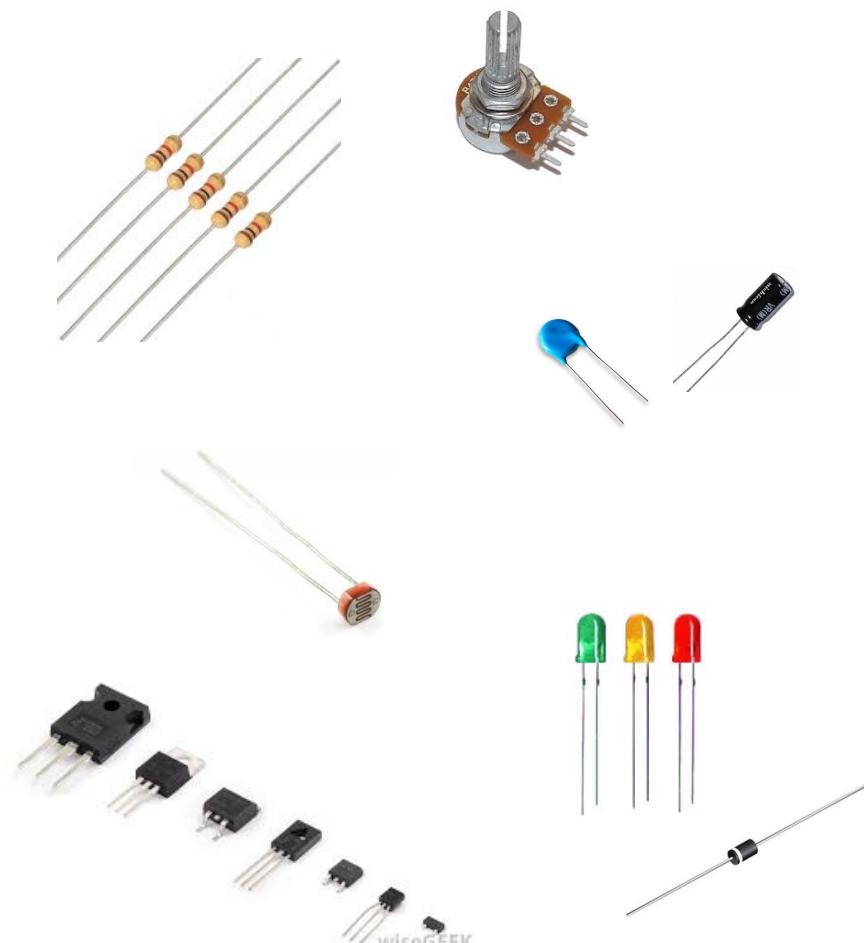
- In generale un componente elettronico in un circuito è un dispositivo atti a regolare il passaggio di corrente elettrica attraverso di esso e/o il valore di tensione elettrica ai suoi capi
 - la legge di natura matematica che lega correnti e tensioni ai capi di un componente componente è detta **caratteristica del componente**
- I componenti elettronici sono classificati in *attivi* e *passivi*.
 - **passivi**: resistori, condensatori, induttanze, diodi, ...
 - non introducono energia, non necessitano di alimentazione esterna
 - **attivi**: transistor, circuiti integrati stessi...
- Resistori, i condensatori, le induttanze sono chiamate anche **impedenze**



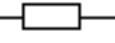
quando introducono loro stessi energia (a volte necessitano di un'alimentazione).

ALCUNI COMPONENTI ELETTRONICI

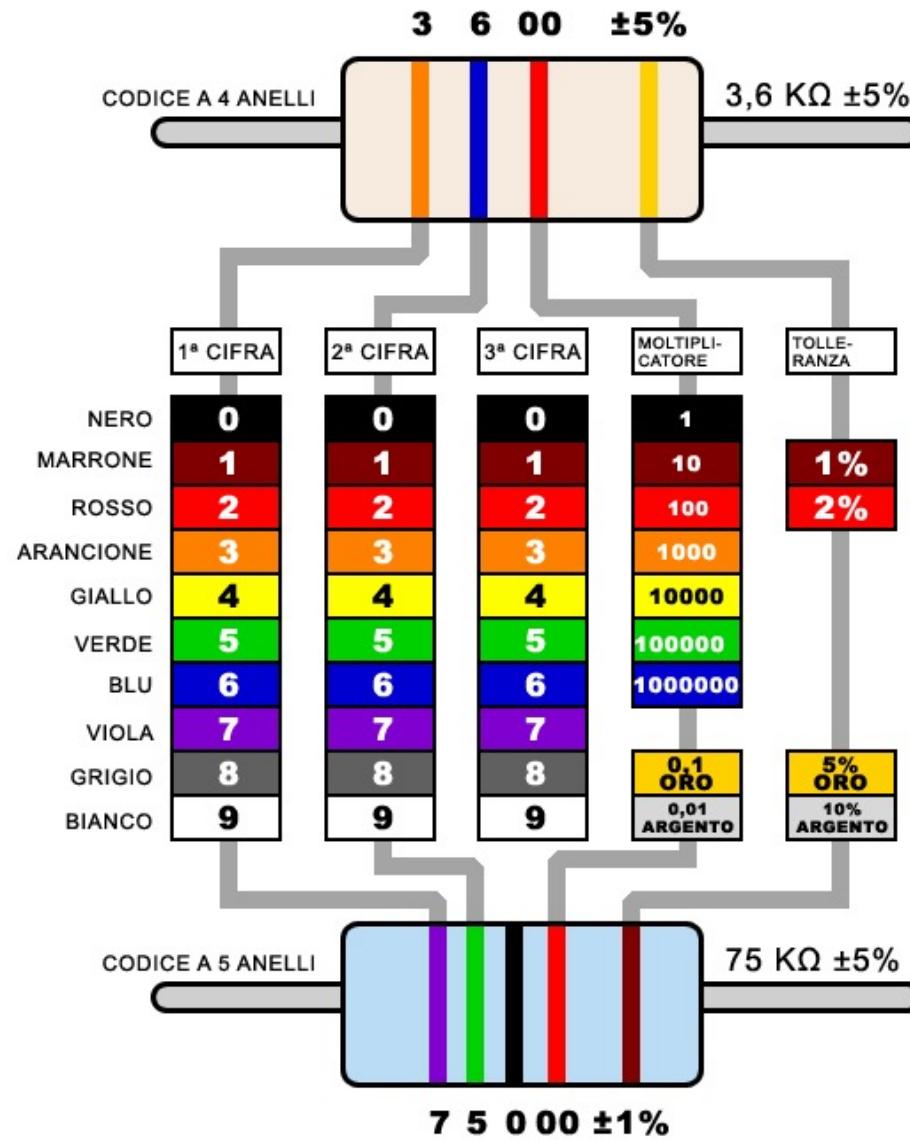
SIMBOLO	SIGLA	DESCRIZIONE	COME SI PRESENTA
-W-	R	RESISTENZA	
-W-↑	R	TRIMMER	
-W-↑	R	POTENZIOMETRO	
±	FR	FOTORESISTENZA	
+	C	CONDENSATORE CERAMICO o POLIEST.	
+	C	COMPENSATORE	
+	C	CONDENSATORE ELETTROLITICO	
A K	DS	DIODO AL SILICIO	
A K	DZ	DIODO ZENER	
A K	DV	DIODO VARICAP	
A K	DL	DIODO LED	
A K	FD	FOTODIODO TRASMITTENTE	
R F	TR	TRANSISTOR	
B S	FT	FET	



RESISTORI

- **Resistore**
 - componente costruito con un materiale che determina una caduta di potenziale elettrico al passaggio di una corrente attraverso di esso.
 - si misura in Ohm (Ω), ed è determinata da caratteristiche geometriche oltre che dalla capacità conduttriva del materiale.
 - simboli  
- Vari tipi
 - fissi, variabili (es: potenziometri)
- Codice colore nei resistori fissi - a 4 anelli o 5 anelli
 - definisce il valore della resistenza
 - prime tre strisce (codice a 4 anelli)
 - ogni colore rappresenta una cifra
 - prima cifra, seconda cifra, moltiplicatore (numero zeri)
 - quarta striscia: tolleranza (5%, 10%, 20%)

RESISTORI - CODICE COLORE



POTENZA NOMINALE DI UN RESISTORE

- *Potenza nominale* di un resistore
 - massima potenza che un resistore può dissipare senza essere danneggiato dall'eccessivo sviluppo di calore
 - dipende dalle dimensioni fisiche e dalla forma
 - maggiore è la superficie, maggiore è la potenza che riesce a smaltire
- In linea di massima è opportuno scegliere resistori con potenza nominale doppia rispetto alla potenza che dovrà dissipare nel suo funzionamento normale

CONDENSATORI

- **Condensatore**
 - immagazzina una carica elettrica, accumulando proporzionalmente una tensione ai suoi capi.
 - la capacità si misura in Farad e suoi sottomultipli
 - le capacità dei condensatori commerciali più comuni sono dell'ordine del milionesimo di farad o meno.
 - sono usati per accoppiare o disaccoppiare segnali tempovarianti e per immagazzinare energia
 - simbolo 

INDUTTORI

- **Induttore**
 - genera un campo magnetico al passare della corrente elettrica.
 - viene utilizzata nelle macchine e nei motori elettrici, ad esempio trasformatori, relè, ecc ...
 - L'induttore con un campo magnetico costante si lascia attraversare da corrente elettrica senza reagire; invece, in un campo magnetico variabile (quindi con un flusso variabile), l'induttore genera il campo magnetico e non fa passare la corrente alternata
 - simbolo: 

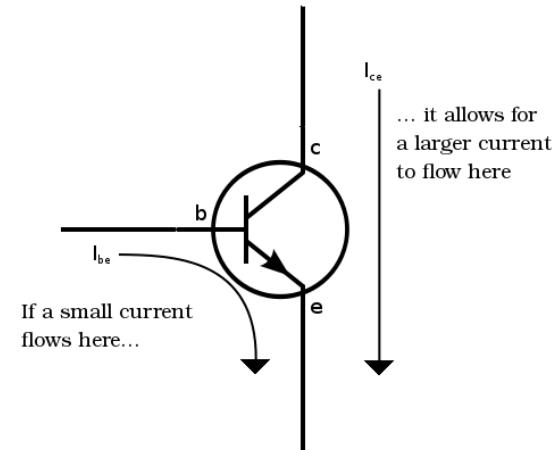
DIODI E LED

- **Diodo**
 - componente semiconduttore e, come tale, ha la proprietà di lasciare passare la corrente elettrica in una sola direzione.
 - componente polarizzato => va inserito nel circuito rispettando detta polarità
 - estremi denominati anodo (+) e catodo (-)
 - corrente fluisce dall'anodo al catodo
 - varie applicazioni in ambito elettronico: funzione *rettificatrice* della corrente in alimentatori, negli stadi rivelatori per separare il segnale dalla portante...
 - simbolo: 
- **LED (acronimo di Light Emitting Diode)**
 - diodi utilizzati come segnalatore luminoso
 - simbolo: 

TRANSISTORI

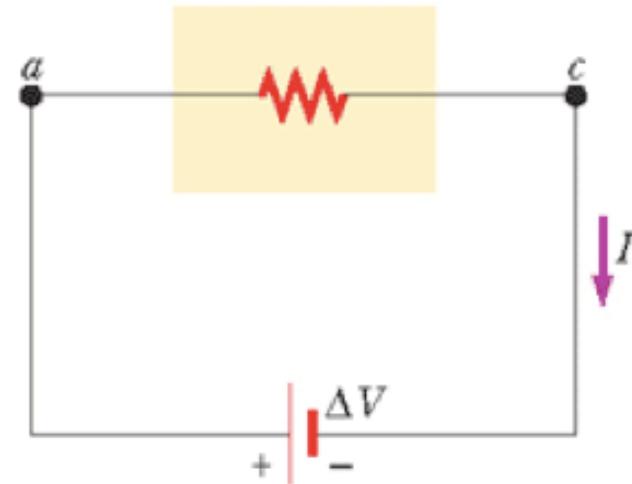
- **Transistor**

- componente attivo, costituente fondamentale della maggior parte dei dispositivi elettronici esistenti e dei circuiti integrati
- a 3 terminali:
collettore, base, emettitore
 - la corrente fluisce da collettore all'emettitore a seconda del segnale alla base
- Viene utilizzato per amplificare un segnale, per pilotare dispositivi di potenza superiore o anche come interruttore elettronico
- simboli:   



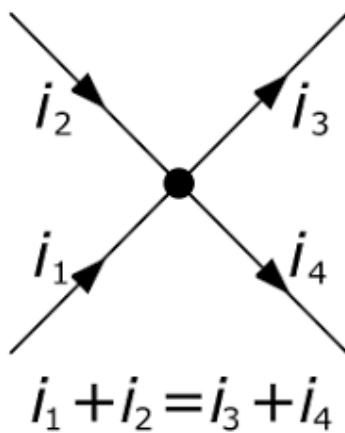
ANALISI CIRCUITI ELEMENTARI

- La corrente I scorre da dove il potenziale è più alto a dove più basso
 - anche se gli elettroni fanno il percorso inverso
 - il potenziale in (a) è ΔV più alto che in (c)
- Dato un componente con una resistenza elettrica per cui valga la legge di Ohm, allora:
 $I = \Delta V/R$
- Assunzioni
 - resistenza trascurabile dei collegamenti fra i vari elementi di circuiti (i fili metallici)
 - la batteria ha una resistenza interna trascurabile
 - generatore ideale => resistenza interna nulla



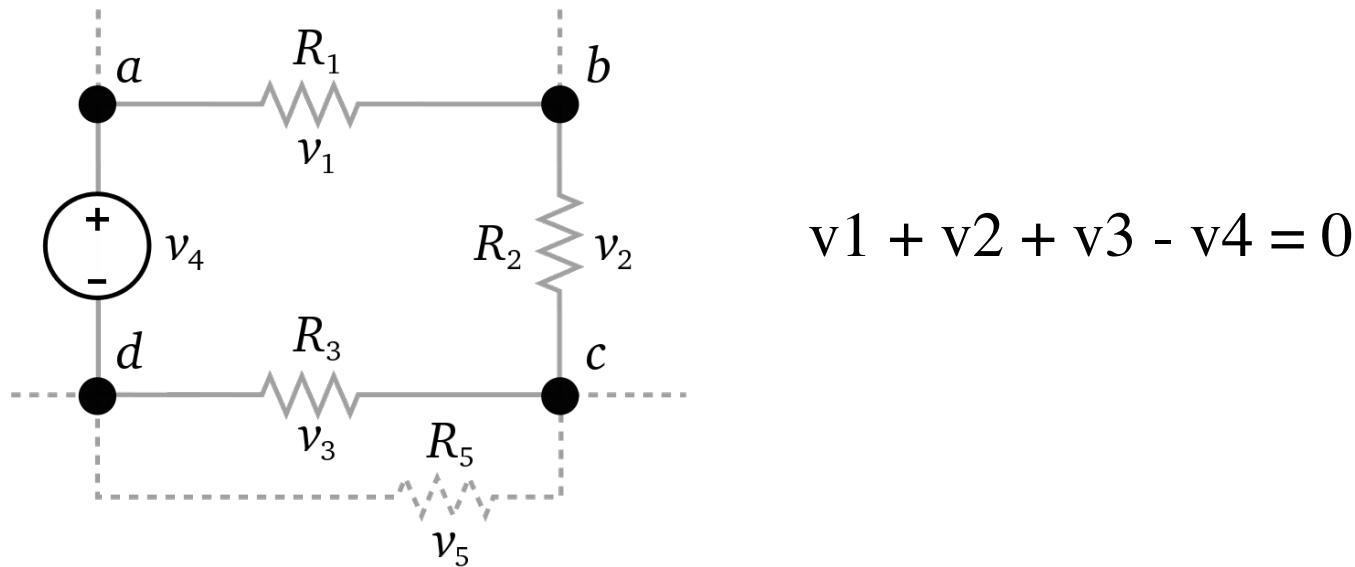
LEGGI DI KIRCHHOFF

- **Prima legge - legge delle correnti (o dei nodi)**
 - in generale, definita una superficie chiusa che contenga un circuito elettrico in regime stazionario, la somma algebrica delle correnti che attraversano la superficie (con segno diverso se entranti o uscenti) è nulla
 - in una formulazione semplificata, e definendo una superficie che racchiuda un singolo nodo del circuito, si può dire che in esso *la somma delle correnti entranti è uguale alla somma delle correnti uscenti*



LEGGI DI KIRCHHOFF

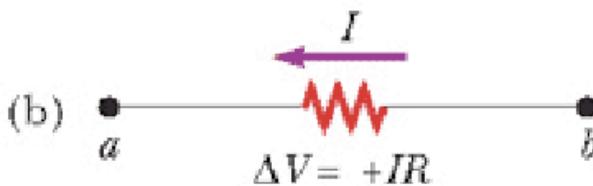
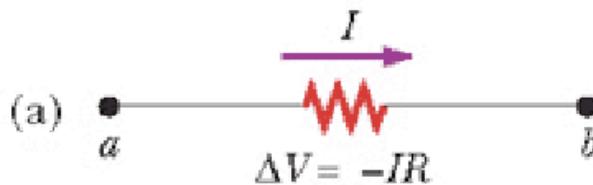
- **Seconda legge - legge delle tensioni (o delle maglie)**
 - in generale, la somma algebrica delle tensioni agenti tra le coppie di punti nello spazio che formano una qualsiasi sequenza chiusa (orientata) è uguale a zero.
 - in una formulazione semplificata, *la somma algebrica delle tensioni lungo una linea chiusa (con il segno appropriato in funzione del verso di percorrenza della maglia stessa) è pari a zero.*



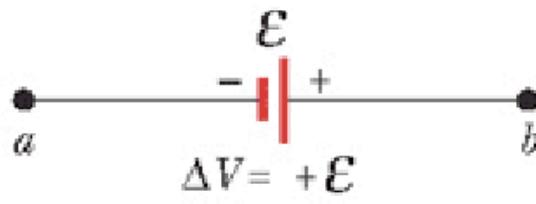
LEGGI DI KIRCHHOFF

- La caduta di potenziale attraverso un elemento di circuito è pari alla differenza di potenziale ai capi

- $\Delta V = V_b - V_a$

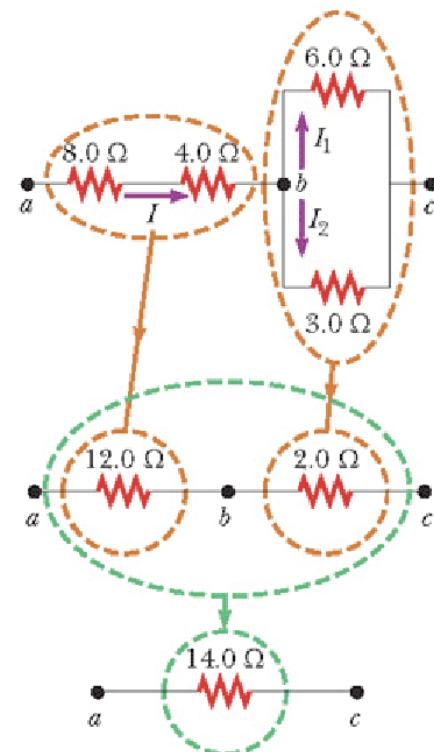
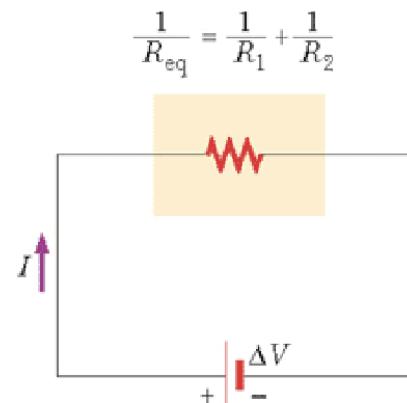
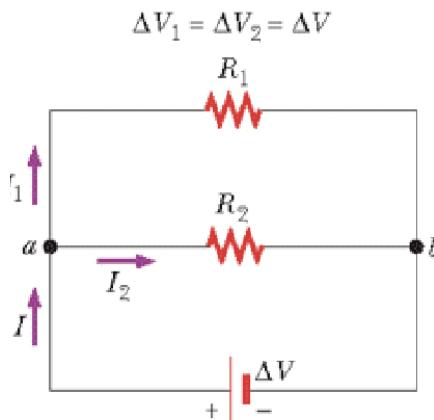


- Per le batterie:



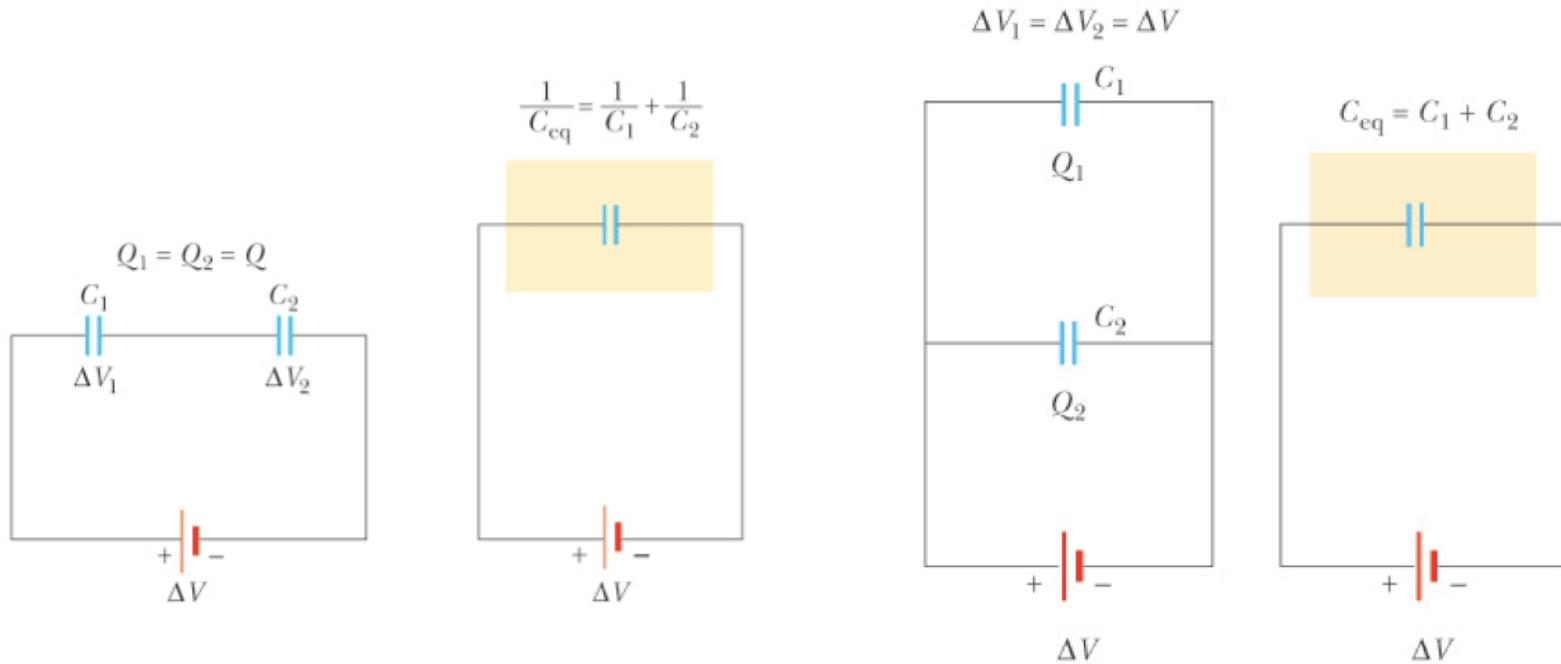
RESISTENZE IN SERIE E IN PARALLELO

- Mediante la legge dei nodi, è possibile calcolare la resistenza equivalente a 2 o più resistenze in serie e in parallelo
 - in serie: $R_{eq} = R_1 + R_2$
 - in parallelo: $R_{eq} = 1/(1/R_1 + 1/R_2) = R_1 * R_2 / (R_1 + R_2)$



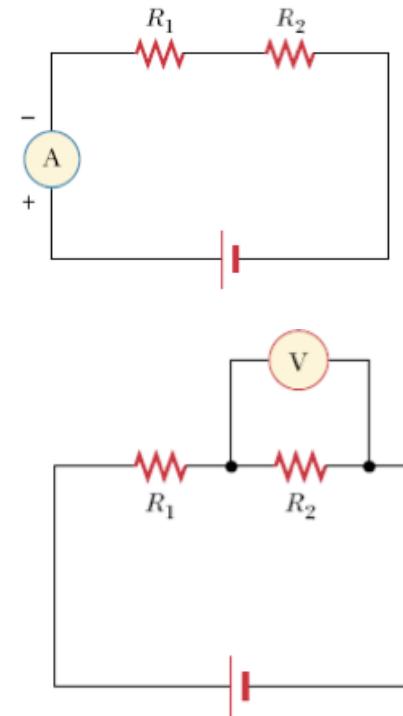
CONDENSATORI IN SERIE E IN PARALLELO

- Analogamente per i condensatori, sfruttando la legge $V = Q/C$ e Kirchhoff, si può determinare il C_{eq} :
 - in serie: $C_{\text{eq}} = 1/(1/C_1 + 1/C_2) = C_1 * C_2 / (C_1 + C_2)$
 - in parallelo: $C_{\text{eq}} = C_1 + C_2$



STRUMENTI DI MISURA

- **Amperometro**
 - misura la corrente che scorre in un circuito
 - deve essere montato in serie
 - resistenza interna piccola, per non perturbare il sistema sotto misura
- **Voltmetro**
 - misura la differenza di potenziale fra due punti di un circuito
 - deve essere montato in parallelo
 - resistenza interna molto grande
- Tester/multimetri



Sistemi Embedded e IoT

Ingegneria e Scienze Informatiche - UNIBO

a.a 2020/2021

Docente: Prof. Alessandro Ricci

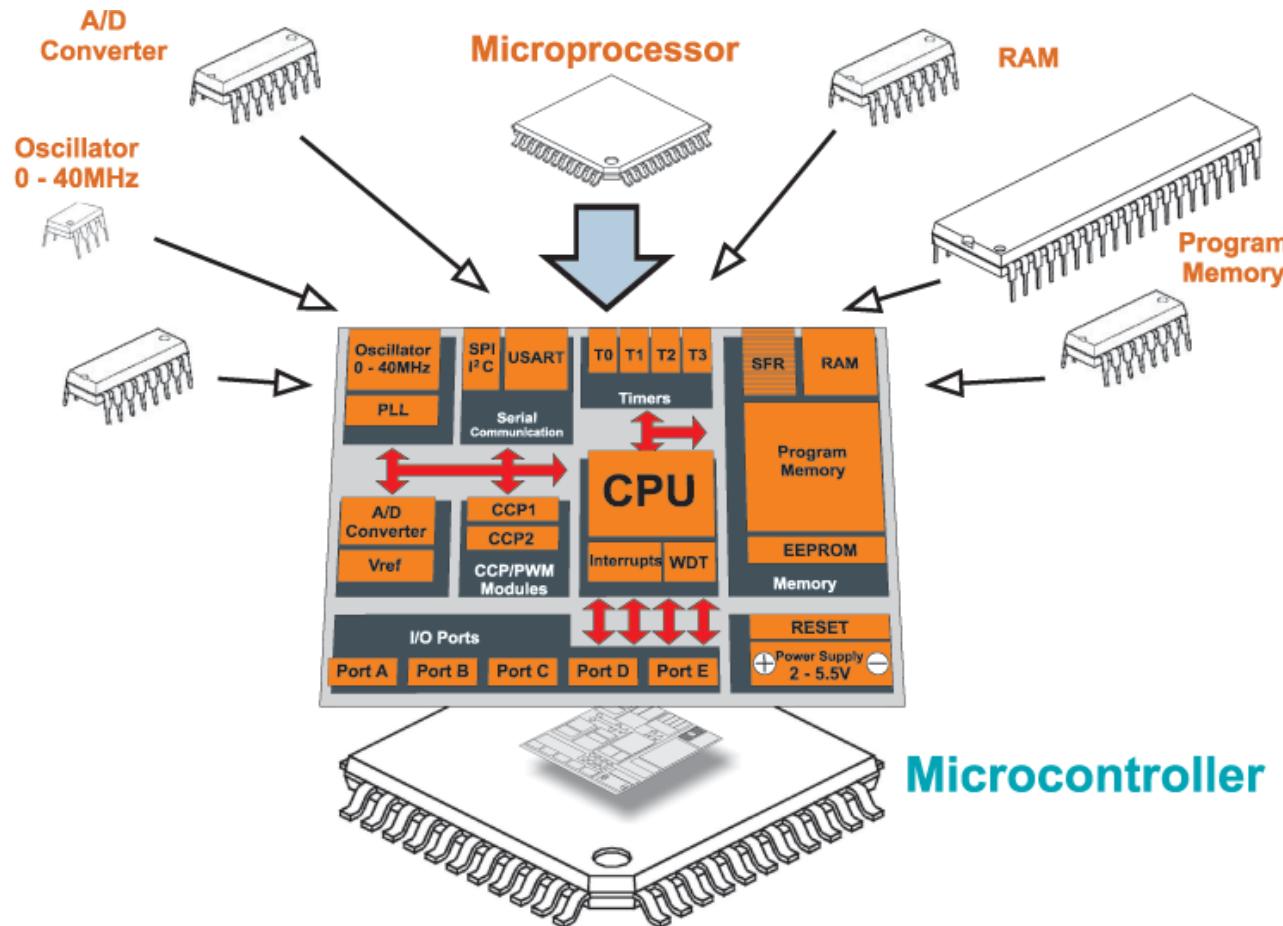
[modulo 1.2]

**SISTEMI EMBEDDED BASATI
SU MICRO-CONTROLLORE**

SOMMARIO

- In questo modulo vengono discussi i componenti principali di un micro-controllore e la sua programmazione di base
 - CPU e Memoria
 - GPIO
 - Interruzioni
 - Timer
 - Seriale
- Si considera come caso di studio Arduino Uno e il framework di programmazione Wiring
- In laboratorio verranno ripresi concetti ed esempi inclusi in questo modulo

QUADRO DEGLI ELEMENTI DI UN MICRO-CONTROLLORE

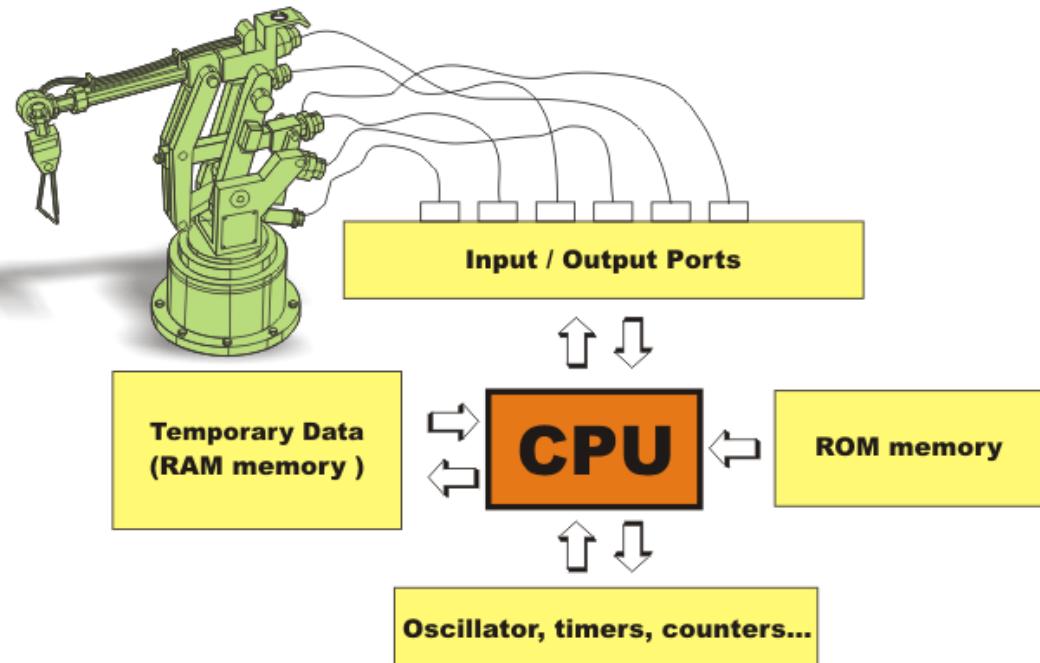


(immagine tratta da:

<http://www.mikroe.com/chapters/view/74/pic-basic-book-chapter-1-world-of-microcontrollers/>)

QUADRO DEGLI ELEMENTI DI UN MICRO-CONTROLLORE

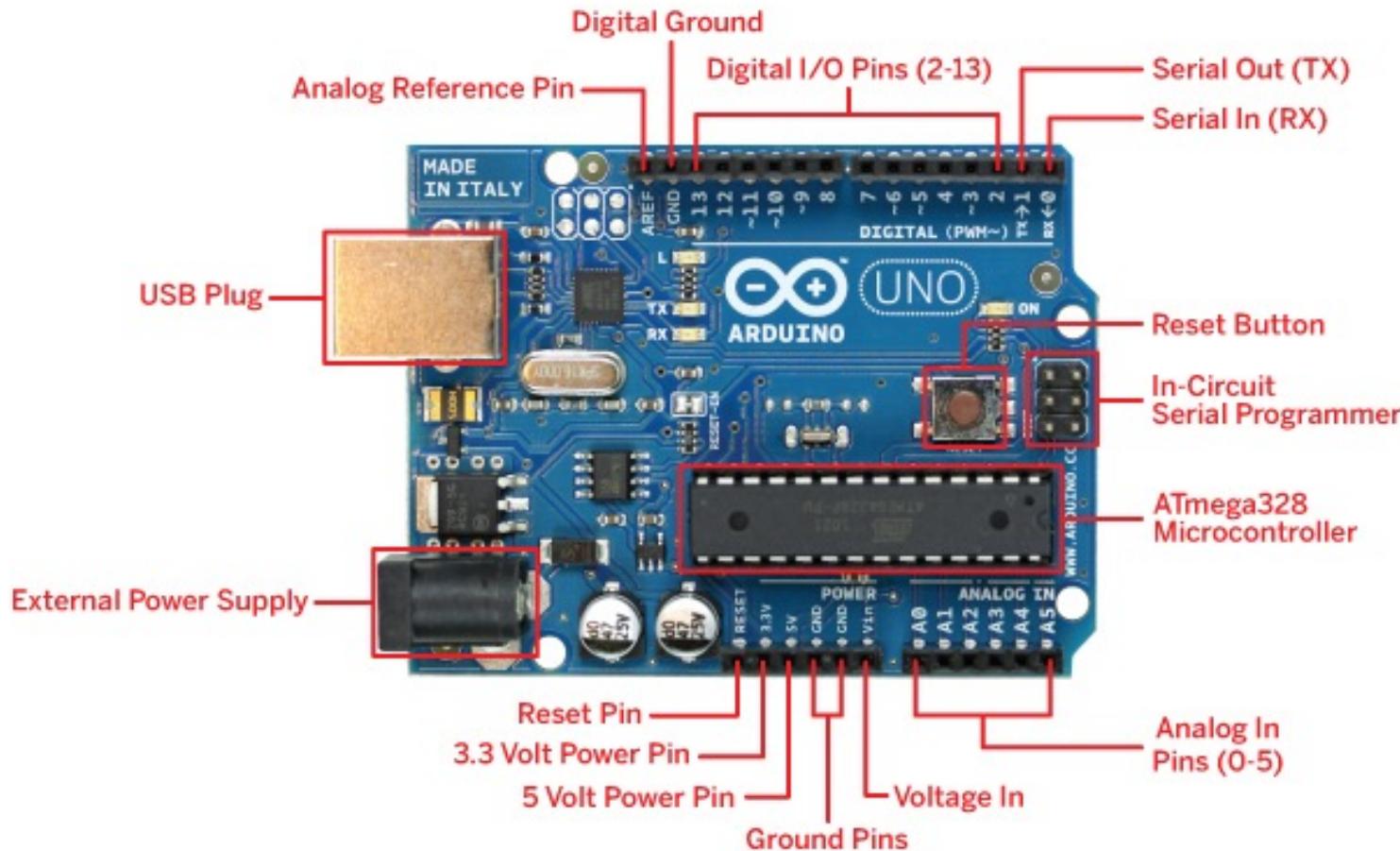
- CPU
- Unità di memoria
- Porte di Input/Output (GPIO)
- Convertitori analogico/digitali
- Timers
- Bus e Comunicazione seriale
- Oscillatore/clock
- Circuito di alimentazione



ESEMPIO DI RIFERIMENTO: ARCHITETTURA ARDUINO UNO

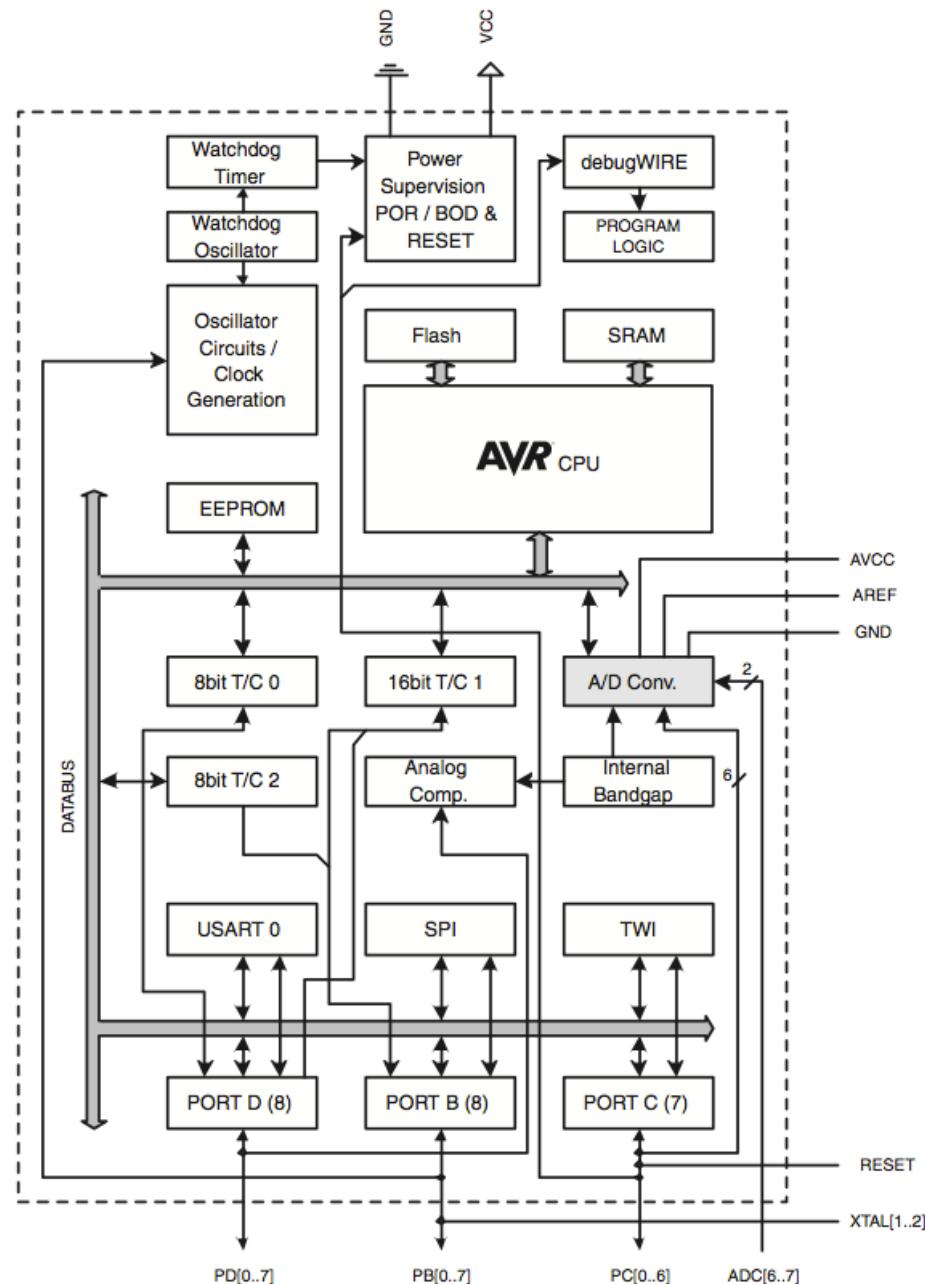
- Componenti principali
 - MCU ATMega 328P
 - 8 bit, 16 MHz
 - Flash memory: 32 KB
 - SRAM memory: 2 KB, EEPROM: 1 KB
 - 14 pin digitali input/output
 - 6 possono essere usati come uscite PWM
 - 6 input analogici
 - connettore USB
 - power jack
 - ICSP header
 - pulsante di reset
- Altre specifiche
 - Operating voltage: 5V
 - Input voltage (recom.): 7-12 V
 - DC current per I/O pin: 40 mA
 - DC current per 3.3V: 50mA

ESEMPIO DI RIFERIMENTO: ARCHITETTURA ARDUINO UNO



MCU ATMega328P

- CPU a 8 bit, 16 MHz
 - 32 registri
- memoria a 16 bit
 - Flash memory: 32 KB,
 - 0.5 usati dal bootloader
 - SRAM memory: 2 KB
 - EEPROM: 1 KB



PROGRAMMAZIONE DI UN MICRO-CONTROLLORE

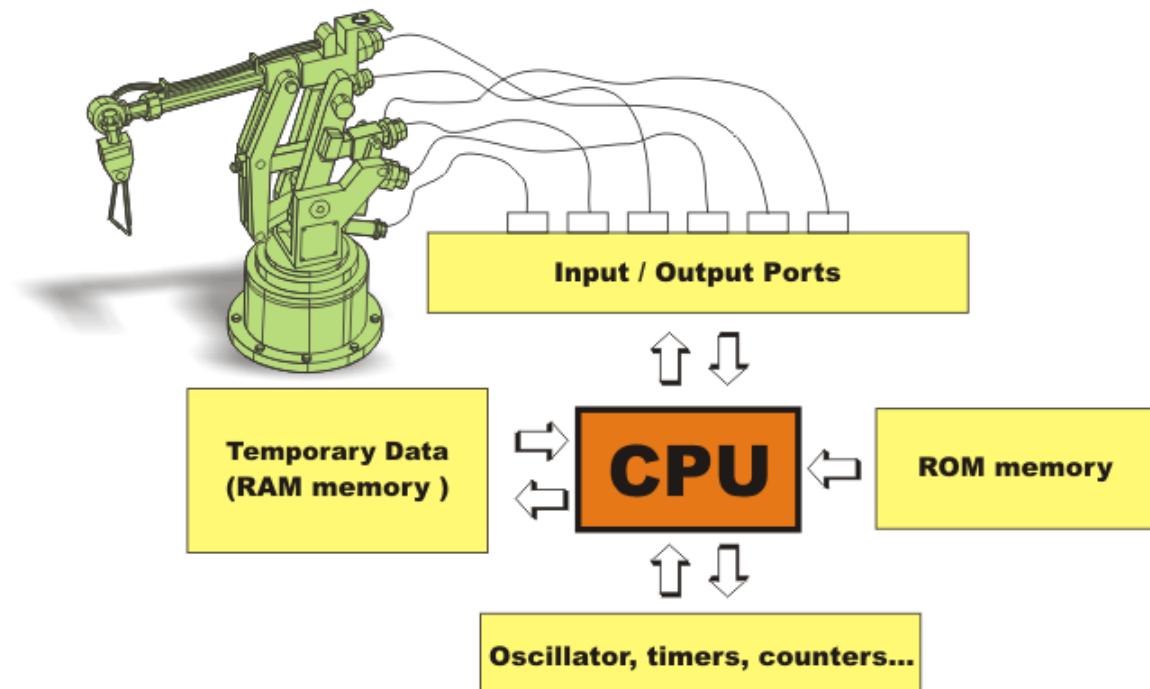
- Tipicamente la programmazione avviene utilizzando un sistema esterno (es: un PC), ove i programmi vengono editati, compilati e viene creato l'eseguibile in codice binario
- L'eseguibile viene quindi trasferito sul micro-controllore mediante opportuni dispositivi HW, oppure direttamente mediante collegamento (es: seriale) con il PC host
- Lo stack software su micro-controllori **non** include alcun sistema operativo
 - il codice eseguibile viene trasferito in memoria e direttamente eseguito dal processore

ESEMPIO ARDUINO

- Su Arduino il programma può essere sviluppato su PC e quindi trasferito direttamente via collegamento **USB**
 - in alternativa è possibile usare l'interfaccia **ICSP** (In-Circuit Serial Programmer) e apposito dispositivo “programmer” esterno
- Il trasferimento mediante USB viene effettuato ad opera di un **bootloader**
 - un piccolo programma (~0.5KiB) pre-caricato (via ICSP)
- Ambiente di programmazione: **Wiring** e Arduino IDE
 - Wiring è un framework open-source C/C++ disponibile per diversi tipi di microcontrollori
 - <http://wiring.org.co/>
 - Arduino IDE come semplice tool per editing, compilazione e trasferimento eseguibile su micro
 - basati sui tool GNU e altri progetti open-source per Atmel AVR
 - include il compilatore **avr-gcc** e librerie open-source di base (es: AVR Libc)

ELEMENTI DI UNA MCU: LA CPU

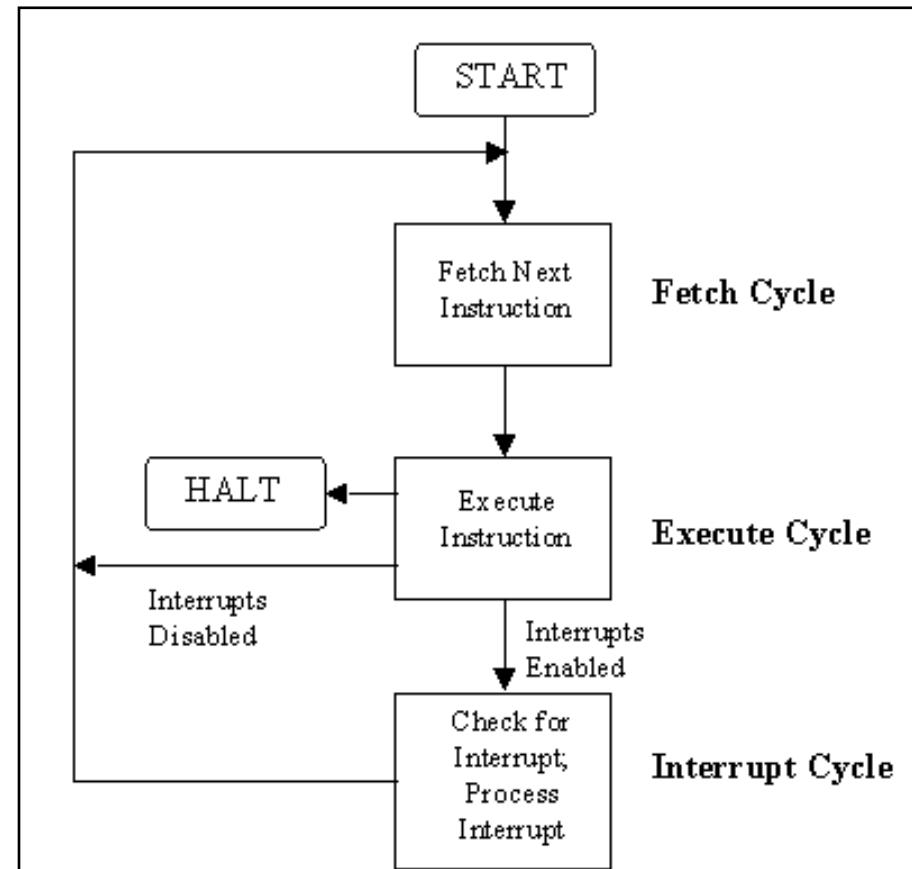
- CPU
- Unità di memoria
- Porte di Input/Output (GPIO)
- Convertitori analogico/digitali
- Timers
- Bus e Comunicazione ser
- Oscillatore/clock
- Circuito di alimentazione



ARCHITETTURA E FUNZIONAMENTO

CPU: RICHIAMI

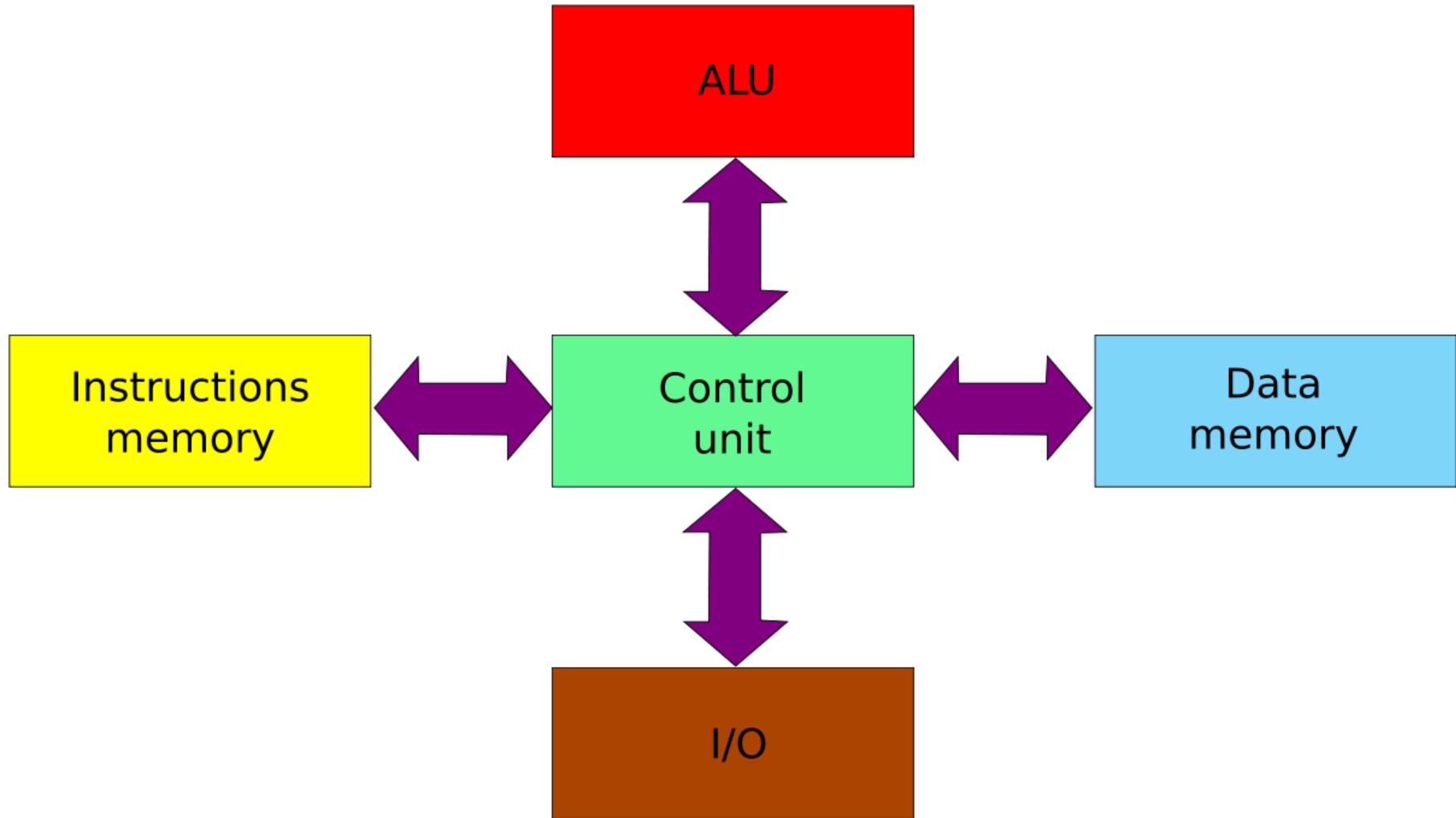
- Il funzionamento di una CPU è definito a livello di modello dalla **macchina di Von Neumann**
 - ciclo di esecuzione *fetch-decode-execute*
- L'**instruction-execution cycle** di un computer in quanto macchina di Von Neumann consiste nel:
 - caricamento dalla memoria (**fetch**) dell'istruzione da eseguire, depositata in un apposito registro (**instruction register**)
 - **decodifica** dell'istruzione, che può comportare il caricamento di altri operandi dalla memoria
 - **esecuzione** dell'istruzione, che può portare all'aggiornamento dei registri e alla scrittura di dati in memoria



ARCHITETTURA HARVARD

- Variante dell'architettura di Von Neumann in cui *istruzioni e dati sono memorizzate in memorie fisicamente separate*, con bus distinti
 - nome deriva dal Harvard Mark I relay-based computer
 - istruzioni memorizzate in schede perforate e dati in dispositivi elettro-meccanici
 - architettura tipicamente usata in micro-controllori e DSP, con estensioni e varianti
 - es: Atmel AVR
- Aspetti di rilievo:
 - le due memorie utilizzate possono avere caratteristiche molto diverse, ad esempio:
 - **codice in FLASH memory**
 - accesso veloce in lettura, lento in scrittura
 - **dati in SRAM**
 - accesso veloce sia in lettura sia scrittura, consumo di più
- Codice e dati possono essere letti/scritti parallelamente
 - dal momento che usano bus distinti

ARCHITETTURA HARVARD



ESEMPIO: MCU ATMega328P

- MCU con architettura **Harvard**
 - istruzioni e dati memorizzate in memorie fisiche separate
 - istruzioni
 - in **FLASH (non volatile)** - 32 KiB
 - di cui 0.5 KB sono occupati dal bootloader
 - dati
 - variabili / stack allocate in **SRAM (volatile)** - **2 KiB**
 - long-term / persistenti in **EEPROM (non volatile)** - **1 KiB**
- **In evidenza:**
 - **memoria utilizzabile per dati/variabili esigua**
=> minimizzare uso stringhe e strutture dati
 - lo stack è su SRAM
 - quindi anche il nesting di chiamate (es: chiamate ricorsive) deve essere usato con attenzione

CONFRONTO FRA MEMORIE...

- Memoria **Flash**
 - memoria non-volatile
 - veloce in lettura, lenta in scrittura
 - max ~100000 cicli di write
 - bassi consumi e costi
- Memoria **SRAM**
 - volatile
 - veloce sia in lettura sia in scrittura
 - su ATMega328P: ~2 cicli di clock
 - maggior consumo rispetto alle FLASH
- Memoria **EEPROM**
 - memoria non volatile
 - accesso molto meno performance
 - può essere letta e scritta usando la libreria <http://arduino.cc/en/Reference/EEPROM>

ARCHITETTURA E FUNZIONAMENTO CPU

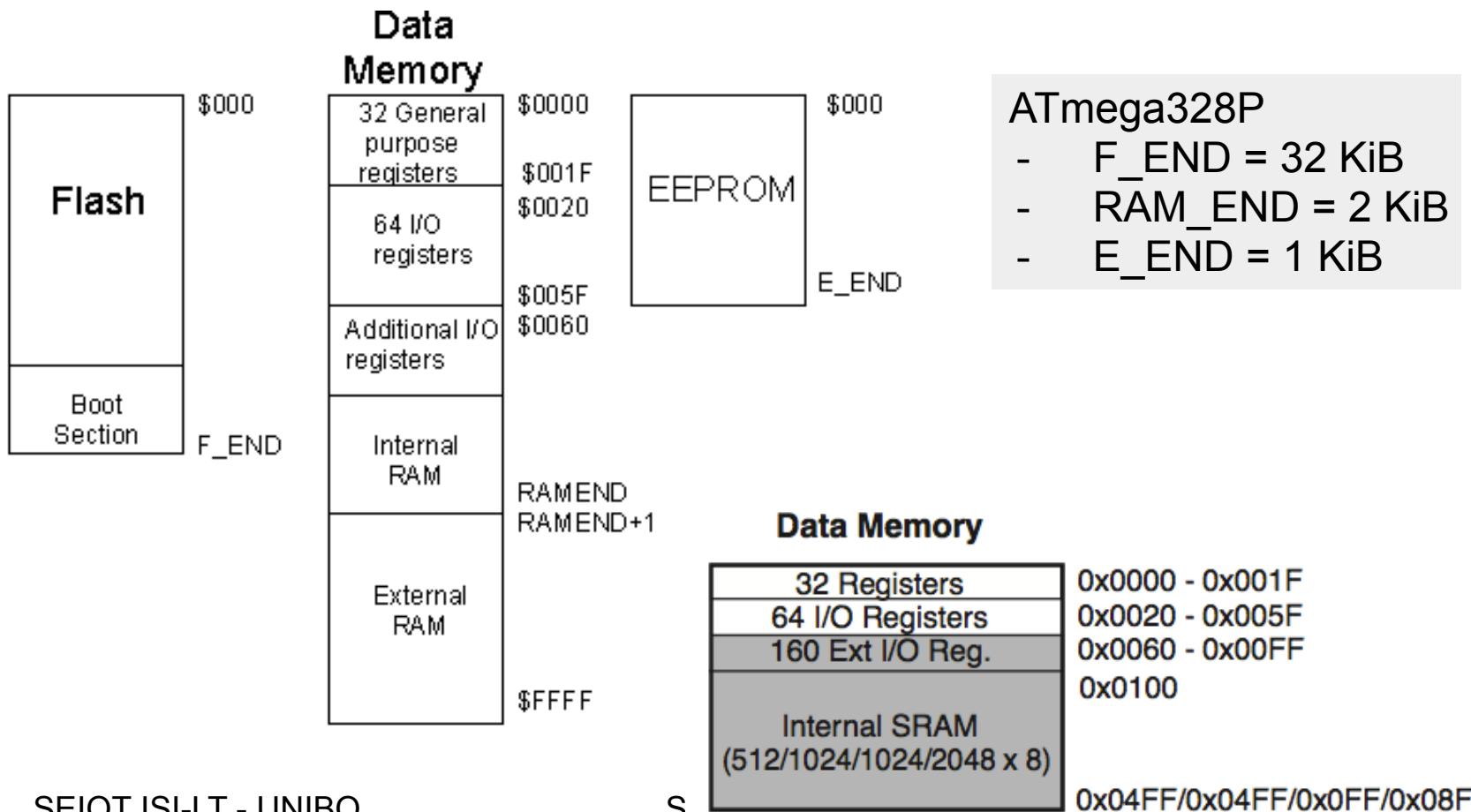
- La CPU esegue le istruzioni del programma codificate in **linguaggio macchina**
- Caratterizzato da un proprio **Instruction Set Architecture (ISA)**
 - insieme di possibili istruzioni riconosciute dalla CPU
 - insieme di registri
 - registri generali o programmabili (**GPR**)
 - es: accumulatore
 - registri dedicati / special purpose (**SFR**)
 - **Program counter (PC)**
 - » l'indirizzo della prossima istruzione da eseguire.
 - **Program Status Word (PS)**
 - » informazioni sullo stato del processore
- CPU a XX bit (XX = 8, 16, 32, 64)
 - numero bit utilizzati nei registri e specificare indirizzi di memoria e valori

ESEMPIO: ARCHITETTURA AVR8

- Il microprocessore presente nei micro-controllore ATMega328P è un AVR8
 - architettura RISC a 16 bit dati, 8 bit codice - 16 MHz
- Registri del processore
 - **32 registri general-purpose ad 8 bit (R0–R31)**
 - tutte le operazioni aritmetiche e logiche avvengono usando questi registri
 - solo le operazioni di load e store accedono alla memoria RAM
 - insieme ristretto di **registri dedicati**, che includono:
 - **PC**: 16- or 22-bit program counter
 - **SP**: 8- or 16-bit stack pointer
 - **SREG**: 8-bit status register - tra i bit:
 - C Carry flag - riporto, con istruzioni di sub
 - Z Zero flag - settato quando un risultato aritmetico è 0
 - I Interrupt flag - settato quando le interruzioni sono abilitate

INDIRIZZAMENTO

- Sia i registri general-purpose del processore, sia quelli relativi all'I/O (descritti in seguito) sono mappati in memoria, accessibili nei primi indirizzi dello spazio di indirizzamento



UNO SGUARDO ALL'ASSEMBLY LANGUAGE AVR

- Insieme delle istruzioni su AVR - categorie:
 - **aritmetiche**
 - **manipolazione bit**
 - **trasferimento**
 - **salto**
 - **test/selezione (branch)**
 - **chiamata (call)**
- Vedere
 - https://en.wikipedia.org/wiki/Atmel_AVR_instruction_set
 - manuale completo:
 - <http://www.atmel.com/images/Atmel-0856-AVR-Instruction-Set-Manual.pdf>

SULLE PERFORMANCE

- Frequenza del clock supportato dal microprocessore: 16 MHz
 - periodo: 0.0625 us (microsec) = 62.5 ns (nanosec)
- La maggior parte delle istruzioni del processore vengono eseguite in 1 o 2 cicli di clock ovvero impiegano ~60ns, ~125ns
 - i dati precisi e nel dettaglio sono disponibili nel datasheet
- Queste informazioni permettono di prevedere in modo piuttosto accurato il tempo di esecuzione di programmi o di porzioni di essi, in particolare del caso peggiore (in programmi non deterministici)
 - importante aspetto per lo sviluppo di sistemi real-time

ARCHITETTURA DI CONTROLLO DI BASE: IL SUPER LOOP

- Il super-loop è l'architettura di controllo (framework) più semplice che si può adottare nella programmazione di micro-controllori
 - molto diffusa perché non richiede supporti HW specifici
 - es: interruzioni
- Caratteristiche
 - inizializzazione
 - ciclo infinito che esegue ripetutamente un certo *task X*

libreria: insieme di procedure/funzioni
framework: oltre ad essere una libreria, impone un modo di mandare in esecuzione un'applicazione che viene gestita con un `while(true)`

```
/* Main.c */  
  
#include "x.h"  
  
void main(void){  
    X_Init(); // Prepare for task X  
    while (1) // for ever 'super loop'  
    {  
        X(); Perform the task  
    }  
}
```

```
/* x.h - header */  
  
void X_Init();  
void X(void);
```

```
/* x.c - implementation */  
  
void X_Init() {...}  
void X(void) {...}
```

CARATTERISTICHE

- Pro
 - semplicità
 - framework che non richiede il sistema operativo per funzionare
 - portabilità
 - reliability e safety
 - efficienza
- Contro
 - temporizzazioni non accurate
 - fragilità e scarsa flessibilità
 - versione base => funzionamento ‘full-power’ => consumo

SUPER-LOOP IN WIRING

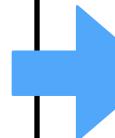
- Framework **wiring** (<http://wiring.org.co/>)
 - framework open-source per programmazione microcontrollori
 - linguaggio di riferimento: C/C++
 - sotto-insieme dello standard ANSI C++
 - architettura di controllo a **super-loop**
 - procedura **setup()** e procedura **loop()**
 - parti che devono essere implementate dal programmatore
 - c'è un main pre-definito in cui vengono chiamate le procedure
 - Wiring fornisce anche un insieme di API di base
 - classi, procedure e funzioni C/C++
- La compilazione e creazione dell'eseguibile avviene mediante compilatore/linker C/C++
 - **avr-gcc** (GNU)
- Tra gli altri tool inclusi nella distribuzione Arduino: **avr-objdump**
 - permette di fare il disassemblato dell'eseguibile

DA WIRING ALL'ESEGUIBILE: ESEMPIO

- Compilazione di un programma “vuoto” in Wiring
 - dal C al linguaggio macchina AVR8

```
void setup(){  
}  
  
void loop(){  
}
```

Programma “vuoto” in
Wiring (compilato con
avr-gcc)



```
00000090 <setup>:  
90: 08 95          ret  
  
00000092 <loop>:  
92: 08 95          ret  
  
00000094 <initVariant>:  
94: 08 95          ret  
  
00000096 <main>:  
96: 0e 94 a4 00    call 0x148      ; 0x148 <init>  
9a: 0e 94 4a 00    call 0x94       ; 0x94 <initVariant>  
9e: 0e 94 48 00    call 0x90       ; 0x90 <setup>  
a2: c0 e0          ldi r28, 0x00 ; 0  
a4: d0 e0          ldi r29, 0x00 ; 0  
a6: 0e 94 49 00    call 0x92       ; 0x92 <loop>  
aa: 20 97          sbiw r28, 0x00 ; 0  
ac: e1 f3          breq .-8       ; 0xa6 <main+0x10>  
ae: 0e 94 00 00    call 0          ; 0x0 <__vectors>  
b2: f9 cf          rjmp .-14     ; 0xa6 <main+0x10>  
...
```

Disassemblato dell'eseguibile
(ottenuto con avr-objdump, opzione -d)

IN EVIDENZA: IL LOOP DI CONTROLLO

- Con informazioni sul codice sorgente:

```
00000090 <setup>:  
 90: 08 95           ret  
00000092 <loop>:  
 92: 08 95           ret  
...  
00000096 <main>:  
int main(void)  
{  
    init();  
 96: 0e 94 a4 00      call   0x148    ; 0x148 <init>  
    initVariant();  
 9a: 0e 94 4a 00      call   0x94     ; 0x94 <initVariant>  
    setup();  
 9e: 0e 94 48 00      call   0x90     ; 0x90 <setup>  
    for (;;) {  
        loop();  
  a2: c0 e0           ldi    r28, 0x00    ; 0  
  a4: d0 e0           ldi    r29, 0x00    ; 0  
  a6: 0e 94 49 00      call   0x92     ; 0x92 <loop>  
            if (serialEventRun) serialEventRun();  
  aa: 20 97           sbiw  r28, 0x00    ; 0  
  ac: e1 f3           breq  .-8       ; 0xa6 <main+0x10>  
  ae: 0e 94 00 00      call   0         ; 0x0 <__vectors>  
  b2: f9 cf           rjmp  .-14      ; 0xa6 <main+0x10>
```

disassemblato con informazioni sul codice sorgente

ESEMPIO DI PROGRAMMA NON VUOTO

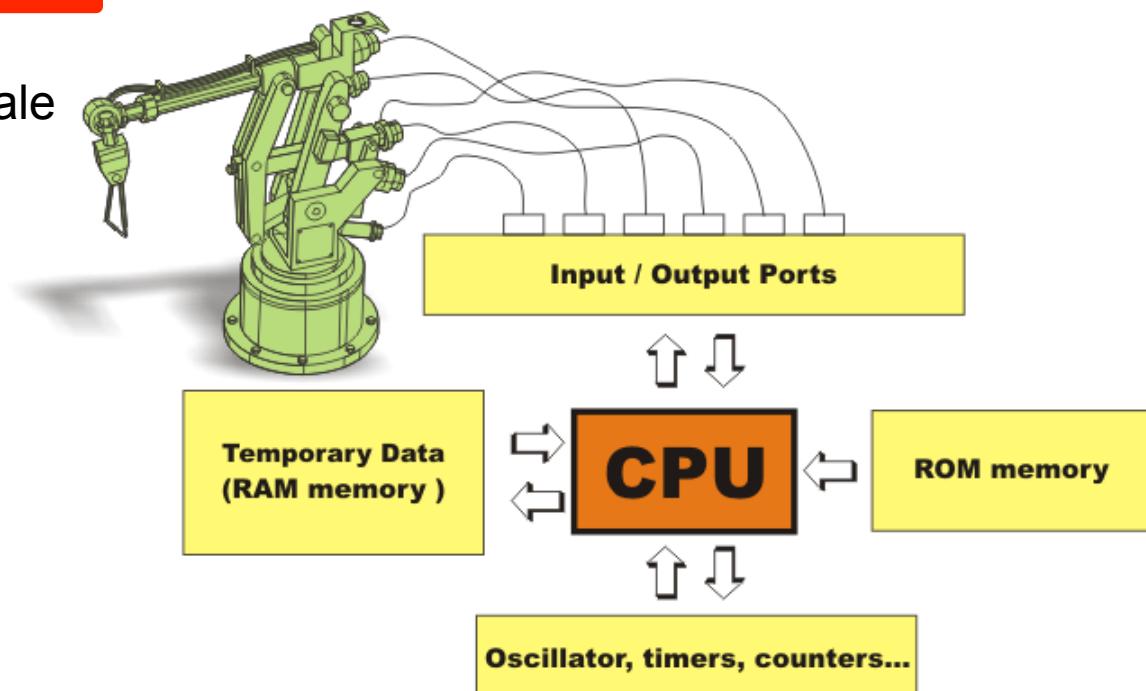
```
int c;  
  
void setup(){  
    c = 100;  
}  
  
void loop(){  
    c = c + 1;  
}
```

...	000000a6 <setup>:		
	a6: 84 e6	ldi	r24, 0x64 ; 100
	a8: 90 e0	ldi	r25, 0x00 ; 0
	aa: 90 93 01 01	sts	0x0101, r25
	ae: 80 93 00 01	sts	0x0100, r24
	b2: 08 95	ret	
...	000000b4 <loop>:		
	b4: 80 91 00 01	lds	r24, 0x0100
	b8: 90 91 01 01	lds	r25, 0x0101
	bc: 01 96	adiw	r24, 0x01 ; 1
	be: 90 93 01 01	sts	0x0101, r25
	c2: 80 93 00 01	sts	0x0100, r24
	c6: 08 95	ret	
...			

in evidenza accesso
a memoria (var c)

ELEMENTI DI UNA MCU: I/O

- CPU
- Unità di memoria
- Porte di Input/Output (GPIO)
- Convertitori analogico/digitali
- Timers
- Bus e Comunicazione seriale
- Oscillatore/clock
- Circuito di alimentazione



GENERAL-PURPOSE I/O (GPIO)

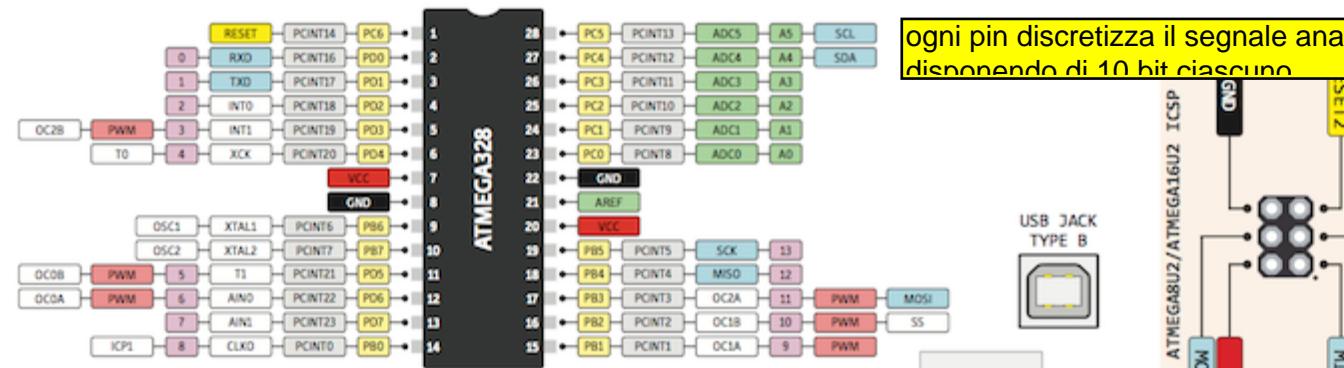
- I micro-controllori utilizzati in ambito embedded contengono usualmente un certo insieme di **pin** che possono essere usati direttamente per gestire input/output
- I pin tipicamente sono *general-purpose* nel senso che possono essere programmati per fungere da **input** o **output** a seconda delle necessità
 - input
 - il valore può essere letto via software
 - output
 - il valore può essere pilotato via software
- Possono essere **digitali** o **analogici**
 - un pin digitale può assumere solo due valori, HIGH o LOW (1 o 0)
 - il valore di un pin analogico può assumere un qualsiasi valore reale all'interno di un certo range

ESEMPIO: ARDUINO GPIO

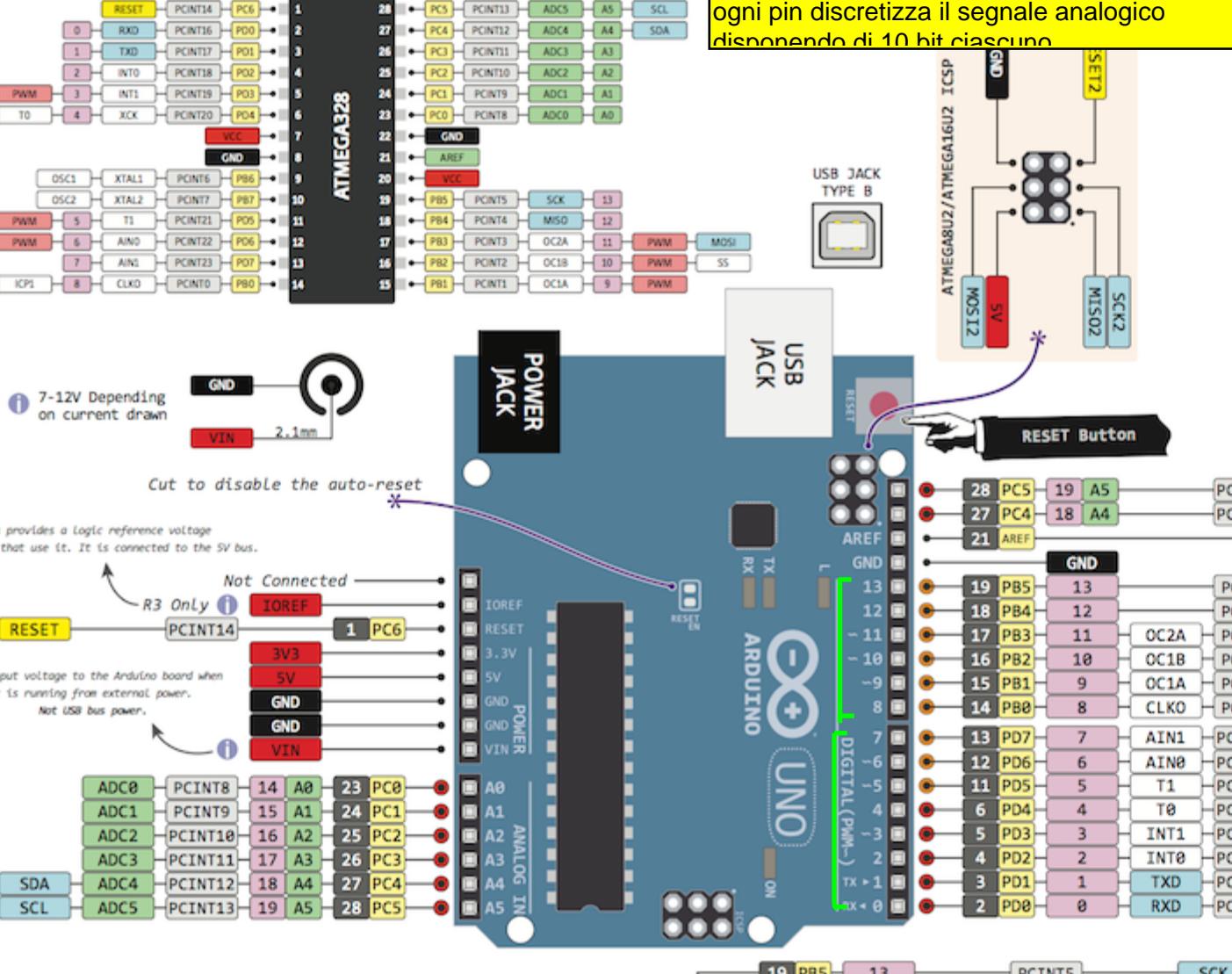
- Pin Arduino
 - **14 pin digitali**
 - configurabili sia come input, sia come output
 - **6 pin analogici**
 - solo input
- Alcuni pin sono usati per più funzioni (multiplexed)
 - esempio: pin 0 e pin 1 sono usati come porta seriale

ARDUINO UNO

PINOUT DIAGRAM

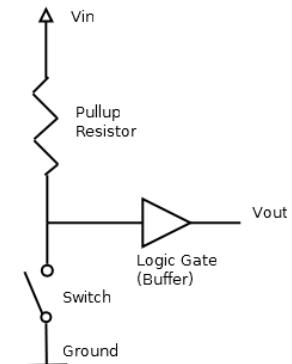


ogni pin discretizza il segnale analogico disponendo di 10 bit ciascuno



PARAMETRI DI FUNZIONAMENTO

- Parametri elettrici importanti di funzionamento per i pin sono:
 - la **tensione** (in Volt)
 - da applicare (nel caso di input) o prodotta in uscita (nel caso di output)
 - esempi 5 Volt (Arduino), 3.3 (Raspberry Pi e ESP)
 - la **corrente** (in Ampère)
 - che può ricevere (input) o che può fornire (output)
 - esempi: Arduino: 40 mA, ESP8266 12mA
- Presenza o meno di *circuiti/resistori di pull-up*
 - internamente i pin possono essere equipaggiati di circuiti di pull-up, per fissare il valore di tensione (a +VCC) anche quando il circuito attaccato al pin è aperto ed evitare malfunzionamenti dovuti al floating del segnale
 - uso resistori dell'ordine di decine di KOhm



PORTE DI INPUT/OUTPUT

- Le **porte** sono ciò che permette al micro-controllore di interagire con i pin e quindi i dispositivi esterni
 - come per i pin, possono essere di input o di output o entrambi, a seconda siano predisposti per ricevere o inviare segnali
- Sono costituite da uno o più **registri special purpose (SRF)** connessi ai pin che trasportano i *segnali digitali o analogici* inviati o ricevuti dai dispositivi esterni
 - il registro mantiene lo stato dei vari pin
 - ad ogni porta è tipicamente associato anche un registro special purpose i cui bit determinano la **configurazione/stato** dei corrispondenti pin (se input, output, etc.)

ESEMPIO: ATMega328P

- L'ATMega328P ha 23 linee di I/O raggruppati in **3 porte** da 8 bit l'una denominate **B, C e D**
 - la porta D contiene i pin 0..7, porta B i pin 8..13, porta C quelli analogici
- Ogni pin è identificato da una sigla come PD0, PC1 o PB2.
 - ad esempio, PD0 identifica il pin 0 della porta logica “D”

ATmega328 Pin Mapping		
Arduino function		Arduino function
reset	(PCINT14/RESET) PC8	1
digital pin 0 (RX)	(PCINT16/RXD) PD0	2
digital pin 1 (TX)	(PCINT17/TXD) PD1	3
digital pin 2	(PCINT18/INT0) PD2	4
digital pin 3 (PWM)	(PCINT19/OC2B/INT1) PD3	5
digital pin 4	(PCINT20/XCK/T0) PD4	6
VCC	VCC	7
GND	GND	8
crystal	(PCINT6/XTAL1/TOSC1) PB6	9
crystal	(PCINT7/XTAL2/TOSC2) PB7	10
digital pin 5 (PWM)	(PCINT21/OC0B/T1) PD5	11
digital pin 6 (PWM)	(PCINT22/OC0A/AIN0) PD6	12
digital pin 7	(PCINT23/AIN1) PD7	13
digital pin 8	(PCINT0/CLKO/ICP1) PB0	14
		28
	PC5 (ADC5/SCL/PCINT13)	29
	PC4 (ADC4/SDA/PCINT12)	27
	PC3 (ADC3/PCINT11)	26
	PC2 (ADC2/PCINT10)	25
	PC1 (ADC1/PCINT9)	24
	PC0 (ADC0/PCINT8)	23
	GND	22
	AREF	21
	AVCC	20
	PB5 (SCK/PCINT5)	19
	PB4 (MISO/PCINT4)	18
	PB3 (MOSI/OC2A/PCINT3)	17
	PB2 (SS/OC1B/PCINT2)	16
	PB1 (OC1A/PCINT1)	15
Digital Pins 11, 12 & 13 are used by the ICSP header for MISO, MOSI, SCK connections (Atmega 168 pins 17, 18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.		

ATMega328P - PORTE E REGISTRI

- Ogni porta è gestita da 3 registri: DDRx, PORTx, PINDx, dove x è il nome della porta. Ad esempio, nel caso della porta D:
 - **DDRD**
 - questo registro è di lettura/scrittura e contiene la **direzione** dei pin ad esso collegati (un bit a 0 => pin impostato come INPUT; un bit ad 1 => un pin impostato come OUTPUT)
 - **PORTD**
 - questo registro è di lettura/scrittura e contiene lo **stato** dei pin, che cambia a seconda della direzione del pin:
 - se il pin è impostato come INPUT, un bit ad 1 attiva la resistenza di PULL-UP, mentre un bit a 0 la disattiva;
 - se il pin è impostato come OUTPUT, un bit ad 1 indica uno stato HIGH sul relativo pin, mentre un bit a 0 indica la presenza dello stato LOW sullo stesso pin.
 - **PIND**
 - questo registro è di sola lettura e contiene, nel caso di un pin impostato come INPUT, la lettura del segnale collegato al pin: 1 per un segnale alto (HIGH); 0 per un segnale basso (LOW).

ATMega328P - PORTE E REGISTRI

- Quindi, per impostare un pin come OUTPUT con un livello HIGH basta:
 - impostare la direzione con il registro DDR_x
 - il suo stato con il registro PORT_x.
- Nel micro-controllore ATMega328, come in molti micro-controllori e anche microprocessori general-purpose, i registri di I/O sono mappati in memoria
 - per cui si accedono/manipolano in modo uniforme mediante scritture e letture di byte/word ad indirizzi di memoria ben definiti
 - nell'ATMega328P sono accessibili a partire dall'indirizzo 0x20, ovvero subito dopo i registri general purpose

IMPORTARE, SCRIVERE, LEGGERE UN PIN IN WIRING

- In Wiring sono fornite API di base
 - **pinMode()**
 - setta la direzione INPUT/OUTPUT di un pin
 - **digitalWrite()**
 - imposta il valore di un pin di OUTPUT
 - **digitalRead()**
 - legge il valore di un pin di INPUT
- Queste API internamente accedono ai registri delle porte come detto in precedenza

ESEMPIO OUTPUT: BLINKING LED

```
#define LED_PIN 13

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

void loop() {
    digitalWrite(LED_PIN, HIGH);
    delay(1000);
    digitalWrite(LED_PIN, LOW);
    delay(1000);
}
```

NOTE SUL CODICE: delay

- **delay()** è una procedura disponibile in Wiring che esegue un *busy waiting* per la quantità di millisecondi specificata:

```
void delay(unsigned long ms)
{
    unsigned long start = millis();
    while (millis() - start <= ms)
        ;
}
```

dal sorgente `wiring.c`,
accessibile in rete

- La procedura fa uso di una ulteriore procedura **millis()**, che restituisce il numero corrente di millisecondi trascorsi dall'accensione del sistema
 - accede a variabili del timer (vedere più avanti nel modulo)
- NOTA:
 - `delay()` e il busy waiting rendono di fatto *non reattivo* il loop di controllo ad altri possibili eventi
 - in generale da evitare, a meno che la non reattività sia voluta

ESEMPIO INPUT/OUTPUT: BUTTON-LED

```
const int buttonPin = 2;      // the number of the pushbutton pin
const int ledPin = 13;        // the number of the LED pin

// variables will change:
int buttonState = 0;          // variable for reading the pushbutton status

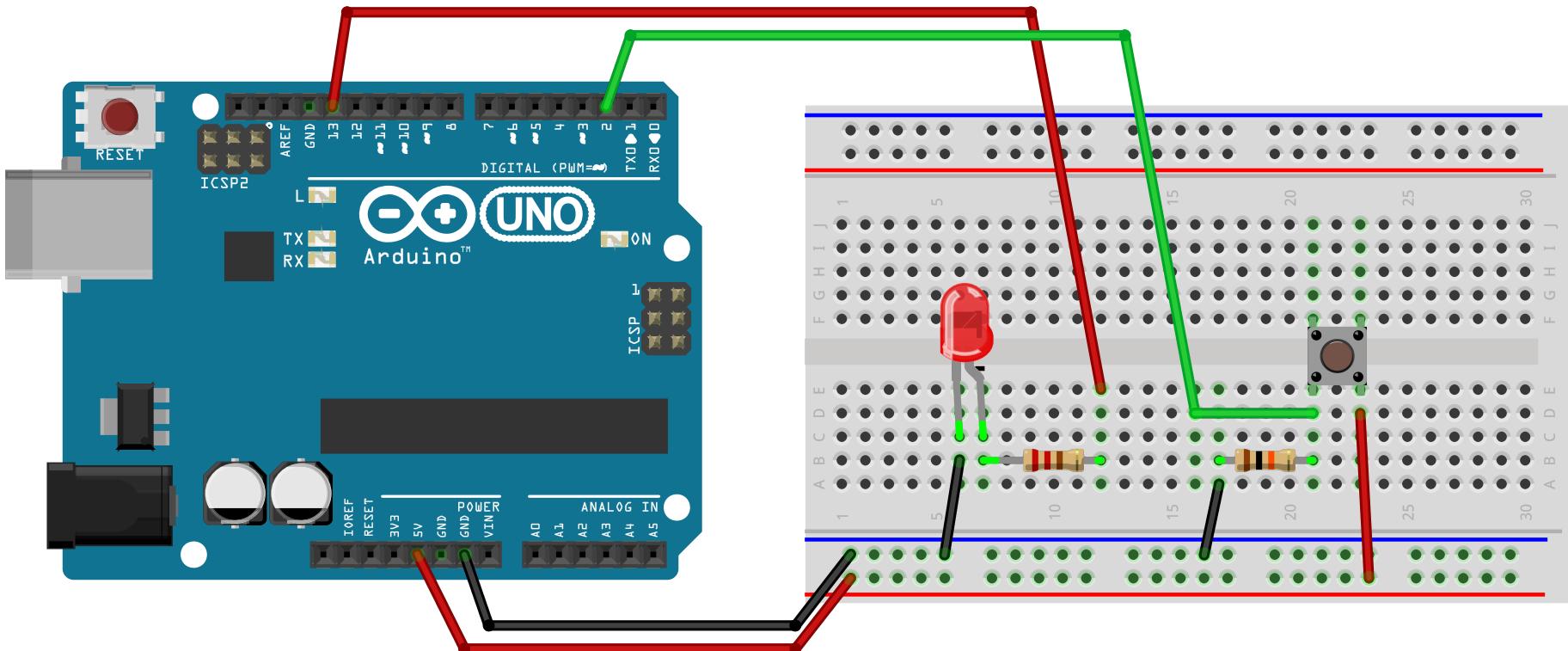
void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}

void loop(){
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  } else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

ESEMPIO INPUT/OUTPUT: BUTTON-LED

(parte si rivedrà in dettaglio in laboratorio)



fritzing

FUNZIONI SPECIFICHE DEI PIN

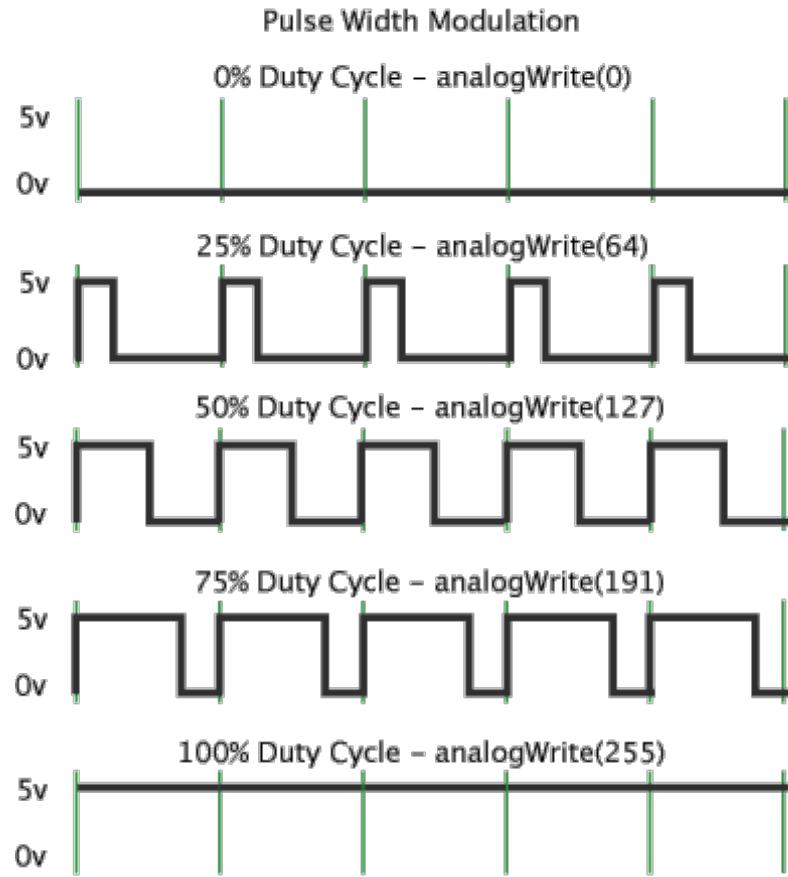
- Oltre ad essere general-purpose, alcuni pin hanno una ulteriore funzione specifica:
 - pin 0 e 1 => **interfaccia seriale TTL** - pin 0 e 1
 - pin 2 e 3 => **interruzioni**
 - pin 3,5,6,9,10 e 11 => **PWM**
 - pin 10, 11, 12, 13 => **Comunicazione SPI**
 - pin 13 => **Builtin led**
- Pin analogici
 - pin A4 e A5 => **I2C**

PULSE-WITH-MODULATION (PWM)

- Tecnica di pilotaggio di dispositivi di output che permette di **emulare in uscita un segnale analogico** a partire dalla generazione di segnali digitali detti PWM
- Il segnale analogico di un certo valore V su un pin è “emulato” mediante un segnale periodico che si ottiene modulando il *duty cycle* di un’onda quadra
 - ovvero un segnale che passa ripetutamente da 0 ad 1
 - **per duty cycle si intende la percentuale di tempo che il segnale è ad 1 rispetto a quella in cui è a 0**
- Sul pin si ottiene in tal modo un segnale equivalente analogico il cui valore (tensione) risultante medio dipende dal duty cycle
 - **ad esempio, avendo duty cycle pari a 50% otteniamo una tensione media pari alla metà del valore massimo (es. 2.5V)**
 - con duty cycle pari a 100% => 5 V, duty cycle 80% => 4 V
- Non funziona per pilotare qualsiasi dispositivo analogico di output
 - funziona bene ad esempio per pilotare LED con intensità variabile, motori in continua (per variare la velocità)

PWM SU ARDUINO

- Su Arduino i pin 3,5,6,9,10 e 11 possono essere utilizzati per output PWM a 8 bit
- API: `analogWrite()`



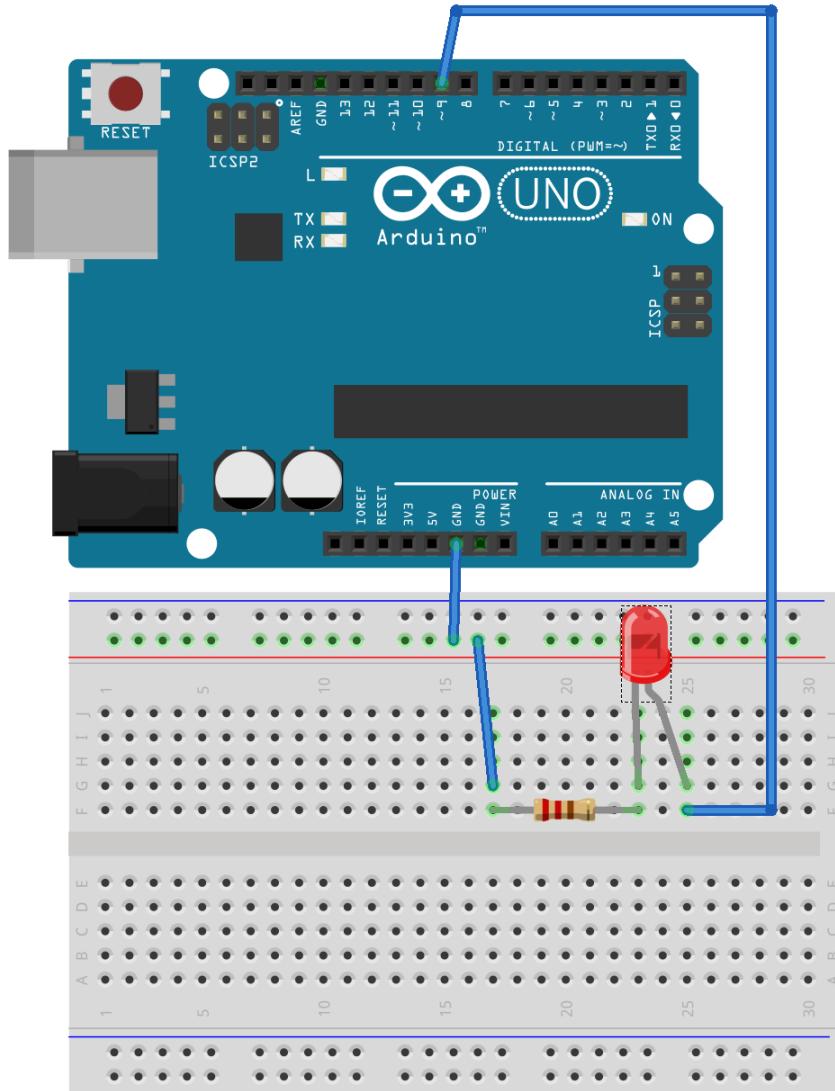
ESEMPIO DI FADE

```
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);
}
```

FADE: SCHEMA



fritzing

SEGNALI ANALOGICI

- Segnale digitale su un pin può assumere nel tempo solo due valori - HIGH (es: 5V) o LOW (es: 0V), corrispondenti a 1 o 0
- Un segnale analogico su un pin può invece assumere un valore continuo all'interno di un certo range (es: 0-5 V)
- Quindi per poter elaborare un segnale analogico da un sistema di elaborazione (che lavora solo con valori digitali), questo deve essere convertito
- La conversione avviene mediante un componente detto convertitore **ADC (Analog-to-Digital)**, che mappa il valore continuo in un valore discreto in un certo range (es: 0..1023)
 - il numero di bit utilizzati per codificare il valore discreto rappresenta la risoluzione del convertitore (es: 10 bit) e ne determina la precisione

LETTURA PIN ANALOGICI IN WIRING

- Arduino UNO può gestire fino a 6 segnali di input analogici
 - pin A0-A5
- Per ogni input, è presente un ADC con risoluzione a 10-bit
 - ogni segnale analogico viene convertito in un valore a 10 bit, quindi da 0 a 1023
- Come già visto la funzione di libreria fornita in Wiring per leggere un segnale è:

```
int analogRead(int PIN)
```

- internamente, la funzione accede ai registri della porta C
- E' fornita anche una ulteriore funzione di utilità, **map**, che permette di mappare un valore in un certo range specificato

ESEMPIO: LETTURA POTENZIOMETRO

```
const int analogInPin = A0; // Analog input pin that the potentiometer is attached to
const int analogOutPin = 9; // Analog output pin that the LED is attached to

int sensorValue = 0;           // value read from the pot
int outputValue = 0;           // value output to the PWM (analog out)

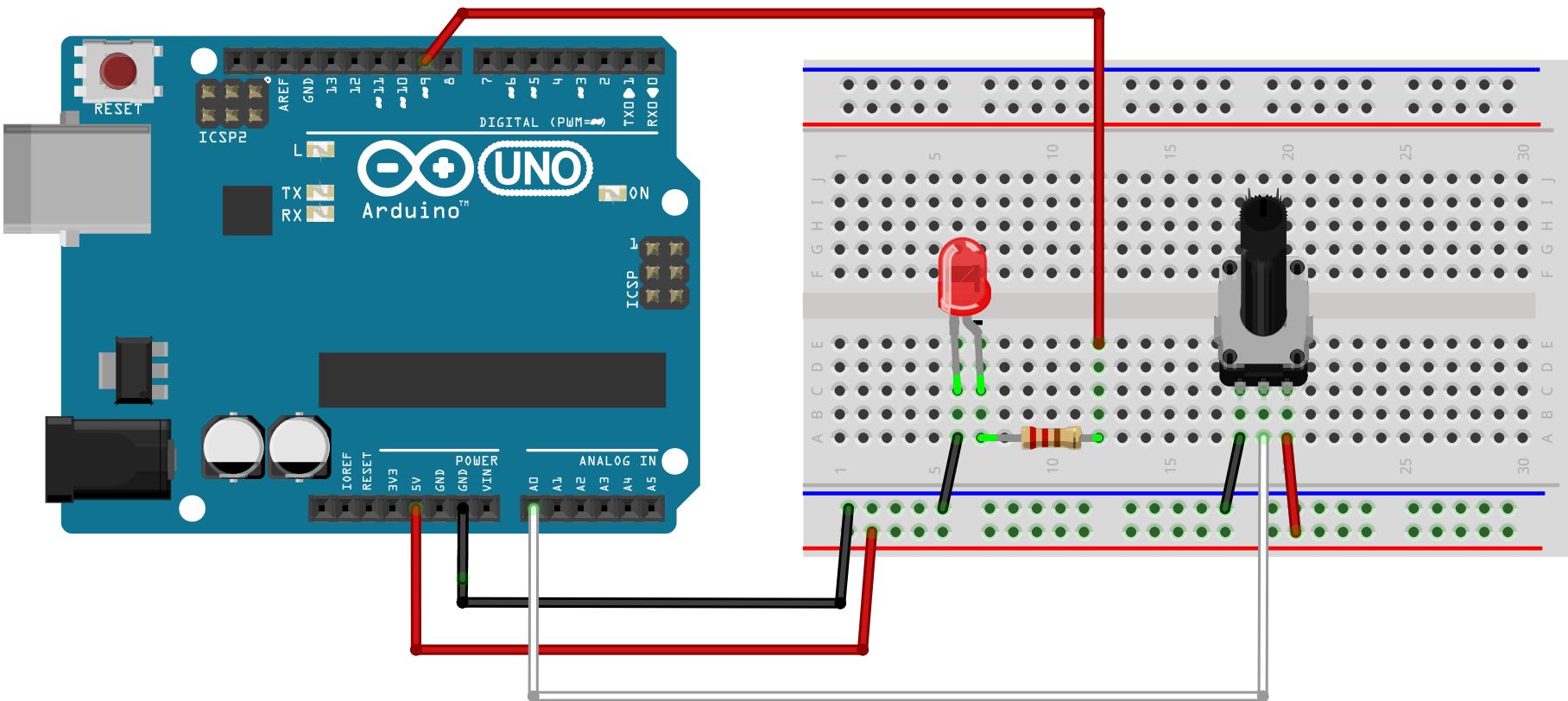
void setup() {
    // initialize serial communications at 9600 bps:
    Serial.begin(9600);
}

void loop() {
    // read the analog in value:
    sensorValue = analogRead(analogInPin);
    // map it to the range of the analog out:
    outputValue = map(sensorValue, 0, 1023, 0, 255);
    // change the analog out value:
    analogWrite(analogOutPin, outputValue);

    // print the results to the serial monitor:
    Serial.print("sensor = " ); Serial.print(sensorValue);
    Serial.print("\t output = "); Serial.println(outputValue);

    // wait 2 milliseconds before the next loop
    // for the analog-to-digital converter to settle
    // after the last reading:
    delay(2);
}
```

LETTURA POTENZIOMETRO - SKETCH



fritzing

NOTA SUL CODICE: LOGGING VIA SERIALE

- Nel codice è stata utilizzata mediante la libreria Serial la porta seriale come interfaccia ove inviare messaggi di output
 - ricevuti dal sistema attaccato alla seriale, ad esempio l'host PC
- Utilizzando Arduino IDE, i messaggi possono essere quindi visualizzati mediante dal Serial Monitor
- Dettagli sulla seriale verranno discussi più avanti in questo modulo

I/O E INTERRUZIONI

- Il meccanismo delle interruzioni permette al micro-controllore (o meglio: al programma in esecuzione) di *reagire ad eventi*, evitando di fare polling
 - possono riguardare i dispositivi I/O, il timer
- In generale le CPU mettono a disposizione uno o più pin - chiamati **IRQ (Interrupt Request)** ove ricevere i segnali di interruzione.
 - quando riceve una richiesta di interruzione, sospende l'esecuzione della sequenza di istruzioni, salva sullo stack l'indirizzo della prossima istruzione che avrebbe eseguito e quindi trasferisce il controllo alla interrupt routine corrispondente - chiamata **interrupt handler** o **interrupt service routine (ISR)**
 - l'indirizzo della ISR è memorizzato nella tabella o vettore delle interruzioni
- Nei micro-controllori gli IRQ sono tipicamente connessi ad uno o più GPIO

dove c'è so: implementare le chiamate di
interruzione

I/O E INTERRUZIONI: ATMega328P

- In questo micro-controllore, due pin - il **pin 2 e il pin 3** - possono essere configurati per generare interruzioni esterne (“external”) (*)
- In particolare, può essere generata una interruzione:
 - su un valore low
 - oppure su un rising o falling edge,
 - oppure un cambiamento di valore

[*] Possibilità di abilitare e gestire interruzioni di tipo PIN CHANGE (vs. EXTERNAL) su qualsiasi pin:

<https://github.com/GreyGnome/EnableInterrupt>

VETTORE INTERRUZIONI

ATMega328P

11.4 Interrupt Vectors in ATmega328 and ATmega328P

Table 11-6. Reset and Interrupt Vectors in ATmega328 and ATmega328P

Vector No.	Program Address ⁽¹⁾	Source	Interrupt Definition
1	0x0000 ⁽¹⁾	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
7	0x000C	WDT	Watchdog Time-out Interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 Compare Match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 Compare Match B
10	0x0012	TIMER2 OVF	Timer/Counter2 Overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 Capture Event

PROGRAMMARE INTERRUPT ROUTINE IN WIRING

- In Wiring è possibile implementare una ISR a livello di user program, mediante la routine:

```
attachInterrupt(intNum, ISR, mode);
```

- Parametri
 - intNum: numero interruzione (int)
 - ISR: puntatore all' interrupt handler
(tipo: void (*func)(void))
 - mode: quando deve essere generata l'interruzione:
CHANGE/FALLING/RISING: ad ogni cambiamento, 1 -> 0,
0 -> 1 (logici)
LOW/HIGH: quando è 0 o 1 (logici)
- E' disponibile la funzione **digitalPinToInterrupt(numPin)** che recupera il numero dell'interruzione associata al pin specificato

ESEMPIO

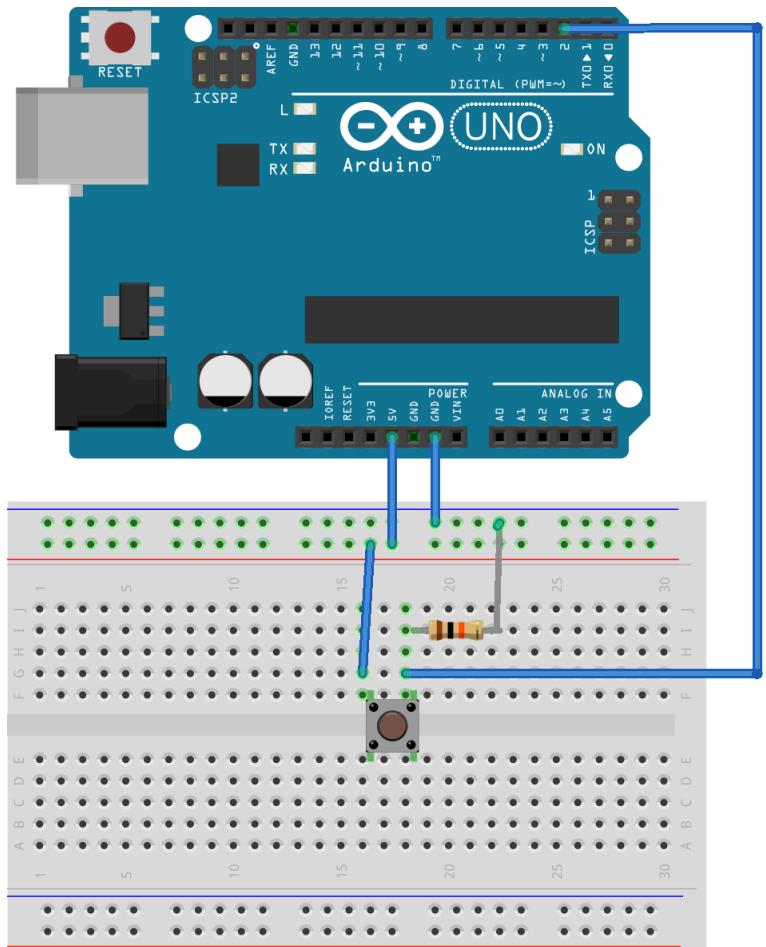
```
volatile int count = 0;

void setup()
{
    // 0 => detecting pin 2 changes
    attachInterrupt(0, inc, CHANGE);
    Serial.begin(9600);
}

void loop()
{
    Serial.println(count);
}

void inc()
{
    count++;
}
```

count++ non è un'istruzione atomica ->
corsa critica : mentre sto leggendo count
(es. parte meno significativa) interviene
l'interruzione di uno primo che veniva



fritzing

Problemi vari

- corse critiche
- fragilità del codice

DISABILITAZIONE INTERRUZIONI

- Possibilità di abilitare / disabilitare interruzioni
 - bit nel registro di stato del processore
- Esempi di istruzioni
 - STI = Set Interrupt (abilita)
 - CLI = Clear Interrupt (disabilita)
- In Wiring/Arduino:
 - **noInterrupts()**
 - **interrupts()**

ESEMPIO - CON MIGLIORAMENTI

```
#define BUTTON_PIN 2
volatile int count = 0;
int prev = 0;

void setup()
{
    Serial.begin(9600);
    attachInterrupt(digitalPinToInterrupt(BUTTON_PIN), inc, RISING);
}

void loop() {
    noInterrupts();
    int current = count;           copio count in current poichè
    interrupts();                 quest'ultima è locale, dopo
                                  interrupt() un'eventuale lettura di
                                  count non genera una corsa critica
    if (current != prev){
        Serial.println(current);
        prev = current;
    }
}

void inc() { count++; }
```

interrupt nesting: interupt innestati -> no corsa critica, non è il caso di arduino.

INTERRUPT HANDLER DESIGN

- Quando disabilitiamo le interruzioni, il sistema non è più reattivo ad eventi esterni. Questo impone alcuni vincoli sul design degli interrupt handler:
 - **devono essere eseguiti in tempi brevi**
 - non possono mai bloccarsi o eseguire loop infiniti
 - **bassa interrupt latency**
 - interrupt latency = quantità di tempo che impiega il sistema per reagire ad una interruzione
 - => *le interruzioni devono essere disabilitate per tempi brevi*
 - => nel caso in cui sia necessario eseguire compiti computazionali onerosi, possono essere usate strategie basate su architettura produttore/consumatore
 - l'interrupt handler produce un task - in un buffer - che verrà successivamente eseguito ad esempio dal main loop.

FAQ SULLE INTERRUZIONI

- *Può un processore essere interrotto durante l'esecuzione di una istruzione?*
 - no, tipicamente le istruzioni in LM vengono eseguite atomicamente. A meno di non considerare quelle istruzioni che permettono di operare trasferimenti molteplici di valori in memoria
- *Se due interruzioni avvengono contemporaneamente, quale viene servita?*
 - ogni processore assegna delle priorità alle interruzioni, per cui viene eseguita l'interruzione più prioritaria
- *Può una richiesta di interruzione interrompere l'esecuzione di una routine di interruzione?*
 - nella maggior parte dei processori sì, supportando "interrupt nesting". Questo si può evitare disabilitando le interruzioni all'inizio della routine e riabilitandole alla fine
- *Cosa succede ad una segnalazione di interruzione se avviene quando le interruzioni sono disabilitate?*
 - tipicamente il processore tiene traccia di tali richieste e le esegue non appena vengono riabilitate
- *Cosa succede se disabilitiamo le interruzioni e ci dimentichiamo di riabilitarle?*
 - malfunzionamento del sistema, crash
- *Cosa succede se disabilitiamo le interruzioni più volte di seguito o le riabilitiamo più volte?*
 - nulla, sono idempotenti
- *Quando un processore parte (boot) le interruzioni sono abilitate o disabilitate?*
 - disabilitate

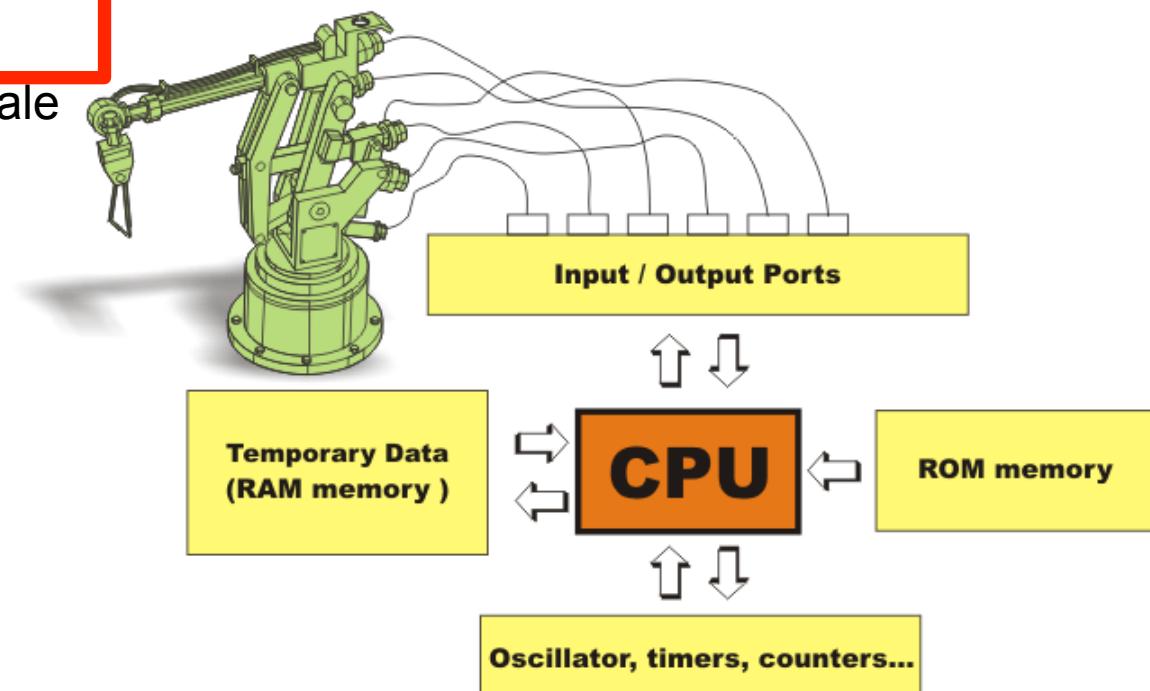
INTERRUPT SU ARDUINO: NOTE

- Su AVR / Arduino l'interrupt handler viene eseguito con le interruzioni disabilitate
 - ovvero le interruzioni non sono abilitate quando l'handler è in esecuzione
- Non tutte le primitive di libreria funzionano se chiamate all'interno di un interrupt handler
 - in particolare, non funzionano quelle funzioni il cui funzionamento si basa direttamente o indirettamente sull'uso di interruzioni
 - esempio: `delay()`/`millis()` non funzionano
 - mentre `delayMicroseconds()` funziona, poiché non si basa su l'uso di timer, ma sull'esecuzione di istruzioni che hanno una specifica durata in clock cycle
 - <https://www.arduino.cc/en/Reference/DelayMicroseconds>

differenza polling ed interrupt

QUADRO DEGLI ELEMENTI DI UN MICRO-CONTROLLORE

- CPU
- Unità di memoria
- Porte di Input/Output (GPIO)
- Convertitori analogico/digitali
- Timers
- Bus e Comunicazione seriale
- Oscillatore/clock
- Circuito di alimentazione



TIMER

- Il timer svolge un ruolo fondamentale per la realizzazione di comportamenti **time-oriented**, in cui il tempo diventa aspetto importante dell'elaborazione e dei compiti del micro-controllore
 - misurare intervalli di tempo
 - segnali PWM
 - realizzazione timeout e allarmi
 - realizzazione forme di multi-tasking basato su preemptive scheduling
- Ad alto livello può essere rappresentato con contatore che viene incrementato a livello HW ad una certa frequenza
- A livello programmatico, è tipicamente possibile
 - configurare la frequenza
 - accedere e leggere il valore corrente del contatore
 - agganciare interruzioni per implementare computazioni event-driven (time-triggered)

ESEMPIO ATMega328

- ATMega328 ha 3 timer interni
 - timer0
 - timer1
 - timer2
- Ognuno dei timer ha un contatore che è incrementato ad ogni tick del clock del timer
 - timer0 e timer2 i contatori sono a 8 bit
 - timer1 è a 16 bit

TIMER E WIRING

- Il timer viene sfruttato per implementare varie funzioni disponibili in Wiring
- Esempi significativi
 - la procedura **millis()**

```
unsigned long millis()
{
    unsigned long m;
    uint8_t oldSREG = SREG;

    cli();
    m = timer0_millis;
    SREG = oldSREG;

    return m;
}
```

NOTE

- timer0_millis = variabile che contiene il valore del contatore relativo a timer0, incrementato dal timer
- disabilitazione interruzioni per evitare corse critiche

- la procedura **analogWrite()** per i segnali PWM

TIMER E INTERRUZIONI SU ATMega328

- Esistono più modalità di gestione delle interruzioni relative ai timer
 - una di queste si chiama **CTC (Clear Timer on Compare Match)**
- In modalità CTC, le interruzioni del timer sono generate quando il contatore raggiunge un certo specifico valore
 - memorizzato in un registro chiamato *compare match register*
- Quando il contatore del timer raggiunge questo valore, resetta il conteggio al clock successivo e quindi riparte a contare
- Scegliendo il valore da mettere nel registro compare match si specifica fra frequenza con cui devono avvenire le interruzioni
 - sapendo che il contatore del timer è aggiornato a 16 MHz
- vedere ad esempio:
<http://www.instructables.com/id/Arduino-Timer-Interrupts>

TIMER E INTERRUZIONI SU ATMega328: DETTAGLIO

- I contatori sono incrementati a 16 MHz
 - quindi ogni tick avviene ogni $1/16,000,000$ di un secondo ($\sim 63\text{ns}$),
- I timer0 e timer2 sono a 8 bit => impiegano $256 \times 63 = 16128\text{ ns} = 16.1\text{ us}$ ad andare in overflow (e ripartire da 0)
- Il timer1 è a 16 bit => impiega $65536 \times 63 = 4128768\text{ ns} \sim 4129\text{ us} \sim 4.1\text{ ms}$
- Questa frequenza può essere modulata specificando un valore detto *prescaler*, che sostanzialmente funge da divisore della frequenza originaria con cui viene incrementato il timer (ovvero 16 MHz):

`timer speed (Hz) = (Arduino clock speed (16MHz)) / prescaler`

- Quindi specificando un prescaler di 1 => incrementa il contatore a 16MHz, un valore di 8 lo incrementa a 2 MHz, .. un valore di 1024 lo incrementa 16 KHz.
- I valori di prescaler possono essere 1, 8, 64, 256, and 1024.

TIMER E INTERRUZIONI SU ATMega328: DETTAGLIO

- Considerando il prescaler, allora la frequenza con cui vengono generate le interruzioni è dato dalla frequenza di incremento diviso il valore specificato nel compare match register:

```
desired interrupt frequency (Hz) =  
16,000,000Hz / (prescaler * (compare match register + 1))
```

- c'è +1 dal momento che il valore 0 rappresenta il primo valore significativo del registro
- Esprimendo l'equazione rispetto al compare match register:

```
compare match register =  
( 16,000,000Hz / (prescaler * desired interrupt freq) ) - 1
```

dobbiamo controllare che questo valore sia inferiore a 256 se usiamo i timer 0 e 2, inferiore a 65526 se usiamo il timer1. Se non lo è dobbiamo aumentare il prescaler

TIMER E INTERRUZIONI SU ATMega328: DETTAGLIO

- Esempio: supponiamo di volere una interruzione al secondo, ovvero alla frequenza di 1 Hz. Allora:

```
compare match register = (16,000,000 / (prescaler * 1)) -1
```

- Se usiamo un prescaler da 1024 otteniamo:

```
compare match register = (16,000,000 / (1024 * 1)) - 1 = 15,624
```

- Siccome il valore è maggiore di 256 però inferiore a 65536, possiamo usare il timer1.

CONFIGURAZIONE REGISTRI

- I registri dei timer che memorizzano la configurazione sono due: **TCCRxA**, **TCCRxB** (dove x = numero del timer 1..3)
 - sono a 8 bit
 - i bit più importanti sono i 3 bit meno significativi si TCCRxB denominati CSx2, CSx1, CSx0, che determinano la configurazione del clock del timer:

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	clk _{IO} /(No prescaling)
0	1	0	clk _{IO} /8 (From prescaler)
0	1	1	clk _{IO} /64 (From prescaler)
1	0	0	clk _{IO} /256 (From prescaler)
1	0	1	clk _{IO} /1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

- La modalità CTC si abilita attivando uno specifico bit in TCCRxB
- Il compare and match register è etichettato con OCRxA

ESEMPIO ARDUINO

```
void setup(){
  Serial.begin(9600);
  setupTimer(1000);
}

volatile boolean timerFlag = false;

ISR(TIMER1_COMPA_vect){
  timerFlag = true;
}

int count = 0;

void step()
{
  count++;
  if (count % 100 == 0){
    Serial.println(count);
  }
}

void loop(){
  /* wait for timer signal */
  while (!timerFlag){}
  timerFlag = false;
  step();
}
```

```
/* setup the timer1 (16 bit) to
 * a specific freq
 */
void setupTimer(int freq){
  // disabling interrupt
  cli();
  TCCR1A = 0; // set entire TCCR1A register to 0
  TCCR1B = 0; // same for TCCR1B
  TCNT1  = 0; //initialize counter value to 0
  /*
   * set compare match register
   *  OCR1A = (16*2^20) / (100*PRESCALER) - 1
   * (must be < 65536)
   * by assuming a prescaler = 1024, then
   * OCR1A = (16*2^10)/freq
   */
  OCR1A = 16*1024/freq;
  // turn on CTC mode
  TCCR1B |= (1 << WGM12);
  // Set CS11 for 1024 prescaler
  TCCR1B |= (1 << CS12) | (1 << CS10);
  // enable timer compare interrupt
  TIMSK1 |= (1 << OCIE1A);
  // enabling interrupt
  sei();
}
```

NOTE

- L'esempio setta il timer1 (16 bit) in modo che generi un tick - quindi una interruzione - 100 volte al secondo (100 Hz). Nel main loop, ad ogni tick si incrementa un conteggio e ogni 100 viene inviato alla seriale
 - per cui viene stampato un messaggio al secondo...
- Attenzione all'overflow nel calcolo dei valori
 - Timer1 è a 16 bit, per cui occorre fare attenzione quando si specifica la frequenza a fare in modo che il valore dettato nel registro compare-match (OCR1A) stia in 16 bit - anche nel fare il calcolo stesso
- **ISR** è una macro usata per dichiarare direttamente il corpo dell'handler di interruzioni relative ad interrupt predefiniti
- Attributo **volatile** in timerFlag
 - se si rimuove, l'esempio non funziona

ALTRÉ LIBRERIE PER IL TIMER

- Esistono varie librerie per Arduino che permettono di gestire il timer a più alto livello
- Una di queste è TimerOne
 - <https://code.google.com/p/arduino-timerone>
 - per installarla, scompattare il file nella directory:
 - (su Windows) Documents/Arduino/libraries
 - (su Mac OS X) Documents/Arduino/libraries
 - (su Linux) /home/YOUR_USER_NAME/sketchbook/libraries
- Mette a disposizione delle classi che permettono di configurare i timer e di fare l'attach di routine di interruzione

ESEMPIO: TIME-DRIVEN BLINKING

```
#include<TimerOne.h>

const int LED_PIN = 13;
boolean flagState = false;

void blinky(){
    if (!flagState){
        digitalWrite(LED_PIN, HIGH);
    } else {
        digitalWrite(LED_PIN, LOW);
    }
    flagState = !flagState;
}

void setup()
{
    pinMode(LED_PIN,OUTPUT);
    Timer1.initialize(1000000); // set a timer of length 10^6 micro sec = 1 sec
    Timer1.attachInterrupt(blinky); // runs blink on each timer interrupt
}

void loop(){}
    in questo caso non c'è risparmio energetico: la cpu rimane sempre attiva.
    si può settare la cpu in stato di risparmio energetico e esempio riattivarla tramite un
    interrupt
```

TIMER E PWM

- Nell'implementazione della libreria che gestisce i GPIO in Wiring su Arduino, i timer vengono utilizzati per realizzare le uscite PWM
 - In particolare:
 - Timer0 è usato per i pin 5 e 6
 - Timer1 è usato per i pin 9 e 10
 - Timer2 è usato per i pin 11 e 3
- 

IL WATCH DOG TIMER

- Contiene un timer in grado di eseguire un conteggio fino ad un certo valore, dopo il quale genera un segnale di output con cui resetta il circuito
 - nel funzionamento normale, il watch dog riceve (periodicamente) un segnale prima che arrivi alla soglia, con cui resetta il conteggio
 - se non riceve il segnale in tempo, allora significa che il microprocessore è entrato in una situazione critica (es: blocco) e resetta il circuito
- Componente molto diffuso nei sistemi embedded.

NOTE: INTERFERENZE

- I timer e le interruzioni sono sfruttati per realizzare molte delle funzionalità messe a disposizione da Wiring
 - ad esempio per le uscite PWM
- Possono esserci quindi dei malfunzionamenti se si utilizzano concordantemente delle funzionalità che sfruttano, ad esempio, il medesimo timer
- Per evitare questi problemi è necessario prendere visione nel dettaglio della documentazione relativa alle varie API usate

POWER MANAGEMENT

- Una funzionalità molto importante supportata in generale dai micro-controllori è la possibilità di funzionare in modalità che comportano livelli diversi di risparmio energetico (sleep mode)
 - istruzione SLEEP
- Esempio ATMega328P (*)
 - 5 modalità di sleep (con risparmio crescente)
 - Idle Mode la cpu si ferma, tutto rimane attivo, circa metà energia
 - ADC Noise Reduction Mode tutte le interruzioni disabilitate -> risparmio elevato. Per risvegliarsi impiega molto tempo. Vedi slide successiva.
 - Power-save Mode
 - Standby Mode
 - Power-down Mode

(*) http://www.atmel.com/Images/Atmel-2486-8-bit-AVR-microcontroller-ATmega8_L_datasheet.pdf, pag 32

ATMEGA SLEEP MODE

- **Idle Mode**
 - stopping the CPU but allowing SPI, USART, Analog Comparator, ADC, Two-wire Serial Interface, Timer/Counters, Watchdog, and the interrupt system to continue operating. This sleep mode basically halts clkCPU and clkFLASH, while allowing the other clocks to run.
- **ADC Noise Reduction Mode**
 - stopping the CPU but allowing the ADC, the external interrupts, the Two-wire Serial Interface address watch, Timer/Counter2 and the Watchdog to continue operating (if enabled). This sleep mode basically halts clkI/O, clkCPU, and clkFLASH, while allowing the other clocks to run
- **Power-save Mode**
 - This mode is identical to Power-down, with one exception: If Timer/Counter2 is clocked asynchronously, Timer/Counter2 will run during sleep.
- **Standby Mode**
 - this mode is identical to Power-down with the exception that the Oscillator is kept running. From Standby mode, the device wakes up in 6 clock cycles
- **Power-down Mode**
 - the External Oscillator is stopped, while the external interrupts, the Two-wire Serial Interface address watch, and the Watchdog continue operating (if enabled). Only an External Reset, a Watchdog Reset, a Brown-out Reset, a Two-wire Serial Interface address match interrupt, or an external level interrupt on INT0 or INT1, can wake up the MCU. This sleep mode basically halts all generated clocks, allowing operation of asynchronous modules only.
 - When waking up from Power-down mode, there is a delay from the wake-up condition occurs until the wake-up becomes effective. This allows the clock to restart and become stable after having been stopped.

ATMEGA PWR MANAGEMENT

Bit	7	6	5	4	3	2	1	0	
	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 13. Sleep Mode Select

SM2	SM1	SM0	Sleep Mode
0	0	0	Idle
0	0	1	ADC Noise Reduction
0	1	0	Power-down
0	1	1	Power-save
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Standby ⁽¹⁾

Note: 1. Standby mode is only available with external crystals or resonators

Table 14. Active Clock Domains and Wake-up Sources in the Different Sleep Modes

Sleep Mode	Active Clock Domains					Oscillators		Wake-up Sources					
	clk _{CPU}	clk _{FLASH}	clk _{IO}	clk _{ADC}	clk _{ASY}	Main Clock Source Enabled	Timer Osc. Enabled	INT1 INT0	TWI Address Match	Timer 2	SPM/ EEPROM Ready	ADC	Other I/O
Idle			X	X	X	X	X ⁽²⁾	X	X	X	X	X	X
ADC Noise Reduction				X	X	X	X ⁽²⁾	X ⁽³⁾	X	X	X	X	
Power Down								X ⁽³⁾	X				
Power Save					X ⁽²⁾		X ⁽²⁾	X ⁽³⁾	X	X ⁽²⁾			
Standby ⁽¹⁾						X		X ⁽³⁾	X				

Notes: 1. External Crystal or resonator selected as clock source

2. If AS2 bit in ASSR is set

3. Only level interrupt INT1 and INT0

ESEMPIO IN ARDUINO



```
void sleepNow()          // here we put the arduino to sleep
{
    set_sleep_mode(SLEEP_MODE_PWR_DOWN); // sleep mode is set here

    sleep_enable();           // enables the sleep bit in the mcucr register
                            // so sleep is possible. just a safety pin

    /* Now it is time to enable an interrupt.
    attachInterrupt(0,wakeUpNow, LOW); // use interrupt 0 (pin 2) and run function
                                    // wakeUpNow when pin 2 gets LOW

    sleep_mode();             // here the device is actually put to sleep!!
                            // THE PROGRAM CONTINUES FROM HERE AFTER WAKING UP

    sleep_disable();          // first thing after waking from sleep:
                            // disable sleep...
    detachInterrupt(0);       // disables interrupt 0 on pin 2 so the
                            // wakeUpNow code will not be executed
                            // during normal running time.

}
```

CONSUMI

(ATMega8 datasheet)

Figure 119. Active Supply Current vs. Frequency (1MHz - 20MHz)

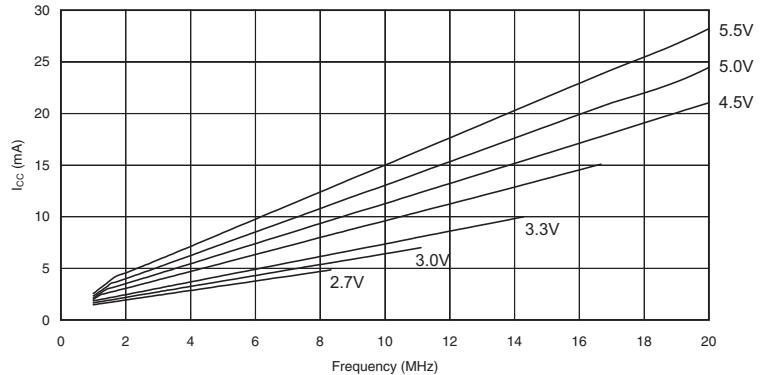
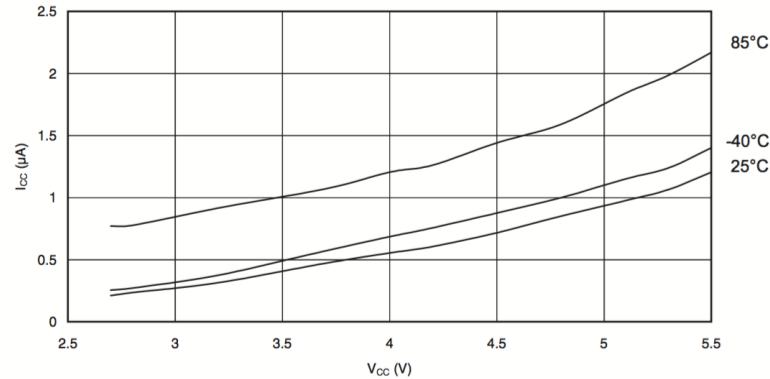


Figure 132. Power-down Supply Current vs. V_{CC} (Watchdog Timer Disabled)



Esempi di test:

<https://learn.sparkfun.com/tutorials/reducing-arduino-power-consumption>

Figure 126. Idle Supply Current vs. Frequency (1MHz - 20MHz)

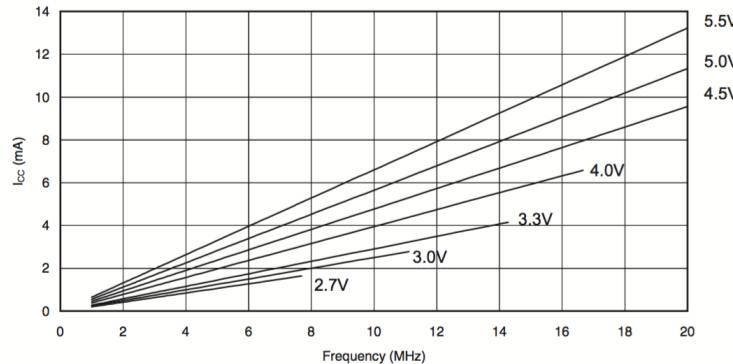


Figure 137. Standby Supply Current vs. V_{CC} (2MHz Resonator, Watchdog Timer Disabled)

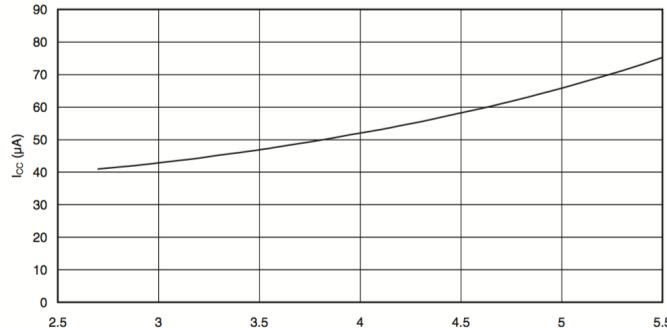
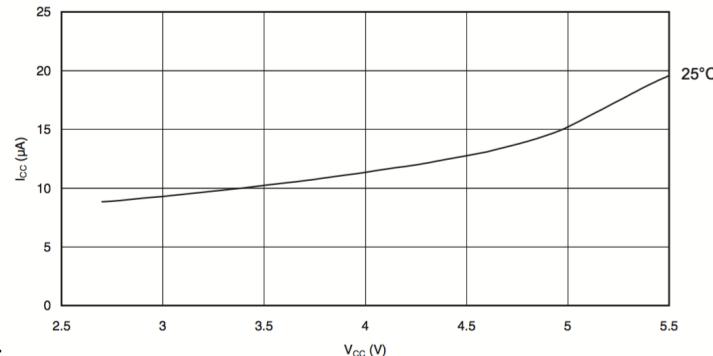
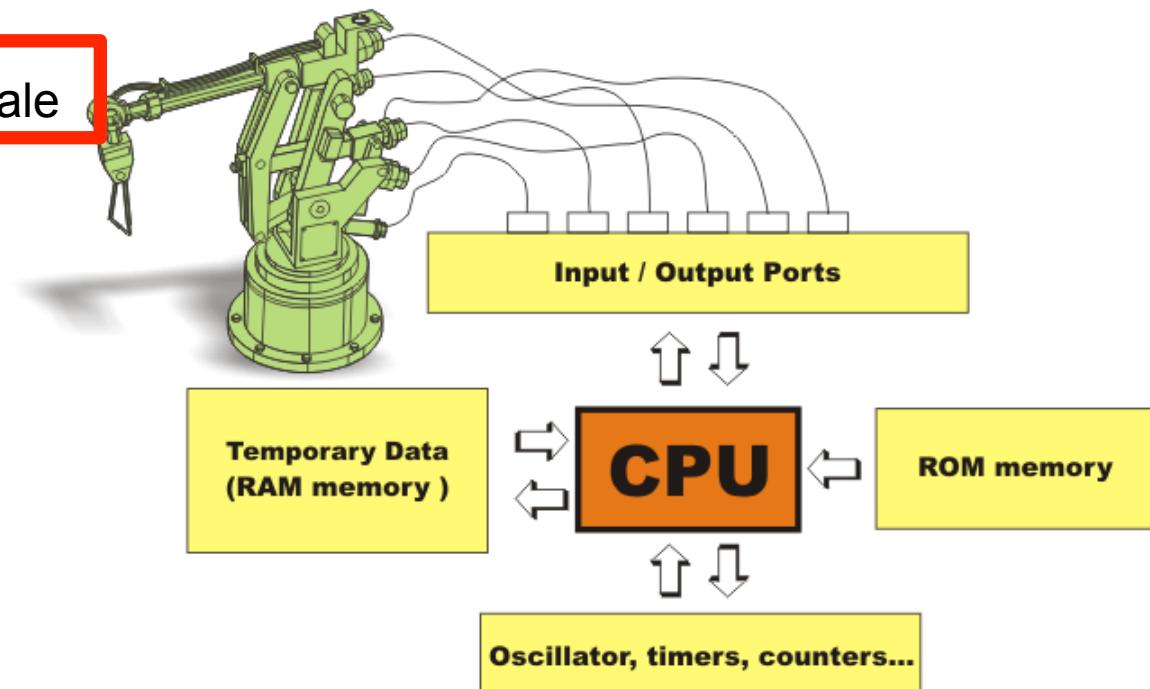


Figure 134. Power-save Supply Current vs. V_{CC} (Watchdog Timer Disabled)



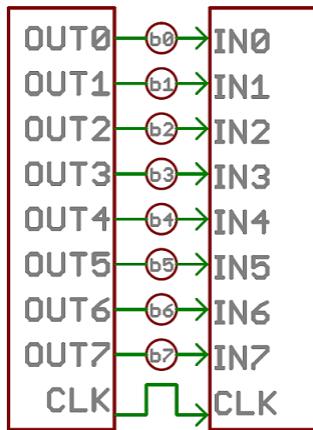
QUADRO DEGLI ELEMENTI DI UN MICRO-CONTROLLORE

- CPU
- Unità di memoria
- Porte di Input/Output (GPIO)
- Convertitori analogico/digitali
- Timers
- Bus e Comunicazione seriale
- Oscillatore/clock
- Circuito di alimentazione



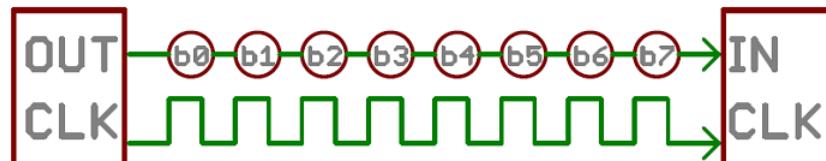
BUS E PROTOCOLLI

- I protocolli di scambio di informazioni mediante singoli pin possono diventare molto complessi a seconda dei dispositivi di I/O con cui interfacciarsi
- Allo scopo sono stati introdotti nel tempo varie interfacce e protocolli, che possono essere classificate in **seriali** e **paralleli**
- Le interfacce **parallele** permettono di trasferire più bit allo stesso tempo
 - richiedono un bus ad 8, 16 o più fili
- Le interfacce **seriali** invece inviano i dati in uno stream sequenziale, bit per bit, utilizzando un solo filo o comunque un numero esiguo.



USB: seriale asincrona.

sincroni: lo scambio di informazioni sono regolati da un clock, che indica quando i dati possono essere trasferiti.



INTERFACCE/BUS SERIALI

- Possono essere distinti in due categorie
 - **asincroni** bisogna che le parti stabiliscono a che velocità comunicare, vedi arduino seriale asincrona.
 - non viene utilizzato alcun clock per la sincronizzazione
 - essenzialmente due linee, di trasmissione e ricezione
 - esempi di standard e implementazioni: **USB, RS-232 e TTL**
 - **sincroni**
 - per la sincronizzazione fra le parti che comunicano viene utilizzata una linea con un *segnale di clock*, che si affianca alla linea dove vengono trasmessi i dati
 - questo permette di avere trasferimenti più rapidi, al prezzo di una maggiore complessità
 - esempi di bus sincroni sono **I²C e SPI**

ASYNCHRONOUS SERIAL

- Nelle interfacce seriali *asincrone*, dal momento che non c'è il supporto di un segnale di clock esterno a sincronizzare la comunicazione, deve essere stabilito un opportuno **protocollo** per assicurare che il trasferimento dei dati avvenga in modo robusto ed error-free
 - non c'è un solo modo - il protocollo è variamente configurabile
 - questione cruciale è che le parti che comunicano adottino lo stesso protocollo
- Vari aspetti da stabilire in questo protocollo
 - **baud rate**
 - **data frame**
 - **synchronous bits**
 - **parity bit**

BAUD RATE

- Il baud rate specifica quanto velocemente i dati sono inviati sulla linea seriale
 - è usualmente espresso in **bits-per-second** (bps).
- Un tipico baud rate per la comunicazione seriale è 9600 bps. Altri valori standard sono 1200, 2400, 4800, 19200, 38400, 57600, and 115200.
- Più è alto il baud rate, più velocemente i dati sono inviati/ricevuti
 - tuttavia c'è un limite a tale velocità (tipicamente 115200), legata alla capacità dei microcontrollori
 - con valori più elevati è possibile andare incontro ad errori di trasmissione.

DATA FRAME

- Ogni blocco di dati - tipicamente un byte - è inviato in forma di pacchetto (packet o frame) di bit. Questi frame sono creati aggiungendo un synchronization e parity bit ai dati inviati



- Il data chunk rappresenta le vere e proprie informazioni trasmesse. In ogni pacchetto si riescono ad inviare dai 5 ai 9 bit. Un valore tipico è 8 (data la lunghezza dei byte)
 - occorre specificare e accordarsi sull'*endianness* dei dati, ovvero l'ordine con cui sono inviati i bit (dal più significativo al meno o viceversa).
 - se non specificato, i dati sono inviati a partire da quelli meno significativi
- I bit di sincronizzazione includono bit di start e stop, per segnare inizio e fine pacchetto (c'è sempre solo 1 bit di start, i bit di stop possono essere anche 2).
- I bit di parità (opzionali) sono usati (a volte) per avere una forma di low-level error checking.

ESEMPIO CONCRETO

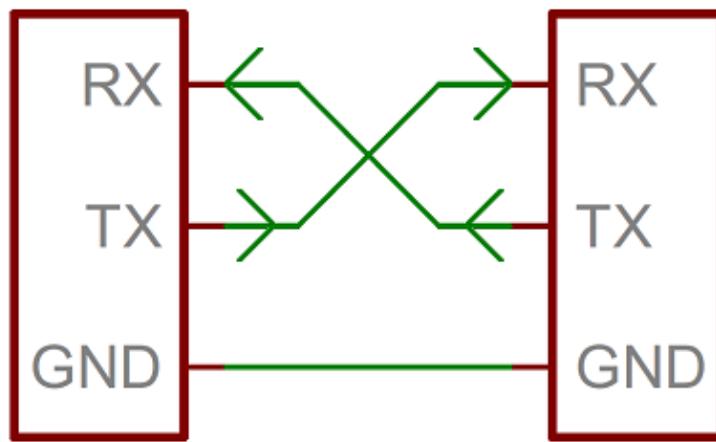
- Il **9600 8N1** - 9600 baud, **8 data bits**, no parity, e 1 stop bit - è uno dei più usati protocolli seriali.
- Esempio di trasmissione.
 - supponiamo di voler trasmettere i caratteri ASCII 'O' e 'K'. Il valore ASCII di O è 79, ovvero 01001111. Il valore di K in notazione binaria è 01001011. **I dati sono trasferiti a partire dal bit meno significativo.**



- siccome stiamo trasferendo a 9600 bps, il tempo speso per mantenere ognuno di questi bit ad un valore alto o basso è $1/(9600 \text{ bps})$, ovvero $104 \mu\text{s}$ per bit.
- per ogni byte di dati inviato ci sono 10 bit inviati: uno start bit, 8 data bits, e uno stop bit. Quindi a 9600 bps stiamo inviando 960 ($9600/10$) byte al secondo.

HARDWARE DELLA SERIALE

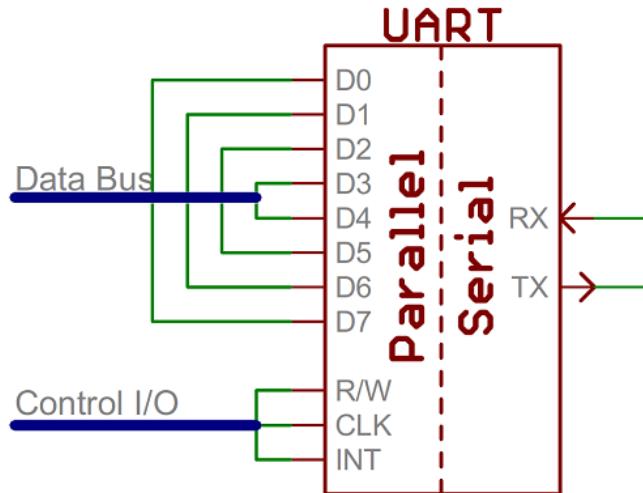
- A livello HW un bus seriale consiste di due soli fili:
 - uno per inviare i bit
 - uno per riceverli
- Per questo motivo, i dispositivi seriali necessitano di 2 pin - un ricevitore (RX) e un trasmettitore (TX)



- Il nome RX e TX è da intendersi rispetto al dispositivo stesso
- Quindi il pin RX di un dispositivo dovrebbe essere collegato a quello TX dell'altro dispositivo, e vice-versa
 - il trasmettitore deve parlare con il ricevitore e vice-versa

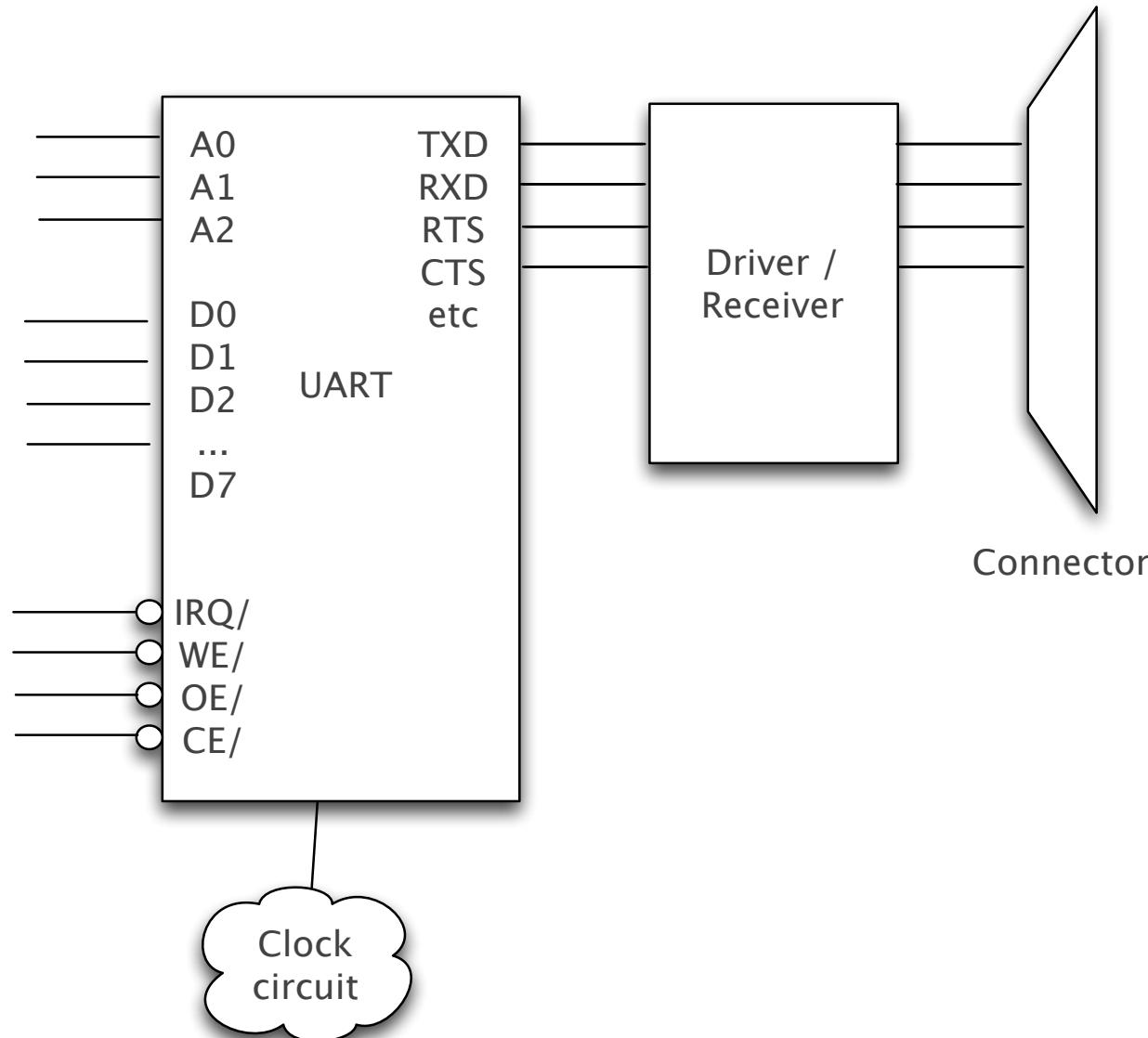
UART

- Un componente fondamentale del sistema seriale è l'**UART** (universal asynchronous receiver/transmitter), ovvero il blocco circuitale che ha il compito di convertire i dati verso e dall'interfaccia seriale.
- Agisce da intermediario fra l'interfaccia parallela e seriale.
 - da una parte di un UART c'è un bus a 8 o più linee dati (più qualche control pin), dall'altro ci sono le due linee seriali, RX e TX.



- UARTs esistono come circuiti integrati stand-alone, tuttavia sono più comunemente inclusi dentro ai microcontrollori.
 - ad esempio, Arduino Uno che si basa su ATmega328 - ha un singolo UART, mentre Arduino Mega - basato su ATmega2560 - ne ha 4

UART



ALTRI ASPETTI

- Full-duplex o half-duplex
 - un'interfaccia seriale dove entrambi i dispositivi possono inviare e ricevere dati può essere **full-duplex** o **half-duplex**
 - full-duplex se entrambi i dispositivi possono inviare e ricevere simultaneamente
 - half-duplex se i dispositivi a turno inviano e ricevono
- Single-wire
 - alcuni bus seriali possono avere la necessità di *un solo* collegamento tra un dispositivo che invia e quello che riceve
 - ad esempio: Serial Enable LCD

LA SERIALE SU ARDUINO

- Arduino UNO ha 1 porta seriale hardware TTL, multiplexed sui GPIO 0 e 1
 - **pin 0 (RX)** usato per ricevere, **pin 1 (TX)** per trasmettere
 - questi pin sono connessi ai pin corrispondenti dell'ATmega8U2 USB-to-TTL-serial chip
 - converte seriale TTL in USB e viceversa
 - questi pin sono quindi usati per più di una funzione
 - come seriale e come GPIO
 - attenzione alle possibili interferenze: se si usa per una funzione, non può essere usata per l'altra e viceversa
- **Libreria Serial**
 - completo controllo e gestione della porta seriale
 - utilizzabile per comunicare (inviare e ricevere) messaggi al sistema collegato via USB

LIBRERIA SERIAL

- Classe Serial, anch'essa estensione di Stream, con vari metodi fra cui:
 - begin(), end()
 - per iniziare, terminare la comunicazione
 - read(), readBytes(), readBytesUntil(), peek(), available()
 - per ricevere dati
 - parseFloat(), parseInt()
 - write(), flush()
 - per trasmettere dati
 - print(), println()
 - per trasmettere stringhe
 - setTimeout()
 - serialEvent()

ESEMPIO DI LETTURA

- Semplice programma di echo

```
char data;

void setup()
{
    Serial.begin(9600);
}

void loop()
{
    if (Serial.available()){
        data = Serial.read();
        Serial.print(data);
    }
}
```

Mediante il serial monitor dell'Arduino IDE è possibile inviare caratteri, che vengono letti e re-inviati al PC dal programma

Da notare nel'esecuzione l'effetto della bufferizzazione

- i caratteri vengono inviati da Serial Monitor e ricevuti da Arduino solo quando viene premuto invio
- la libreria li bufferizza e quindi permette di ricevere carattere per carattere

SYNCHRONOUS SERIAL

- Un'interfaccia seriale sincrona accoppia sempre la sua linea per trasmettere i dati con un **segnale di clock**, per sincronizzare
 - questo permette di avere trasferimenti più rapidi, al prezzo di una maggiore complessità
 - esempi di protocolli sincroni sono SPI e I²C

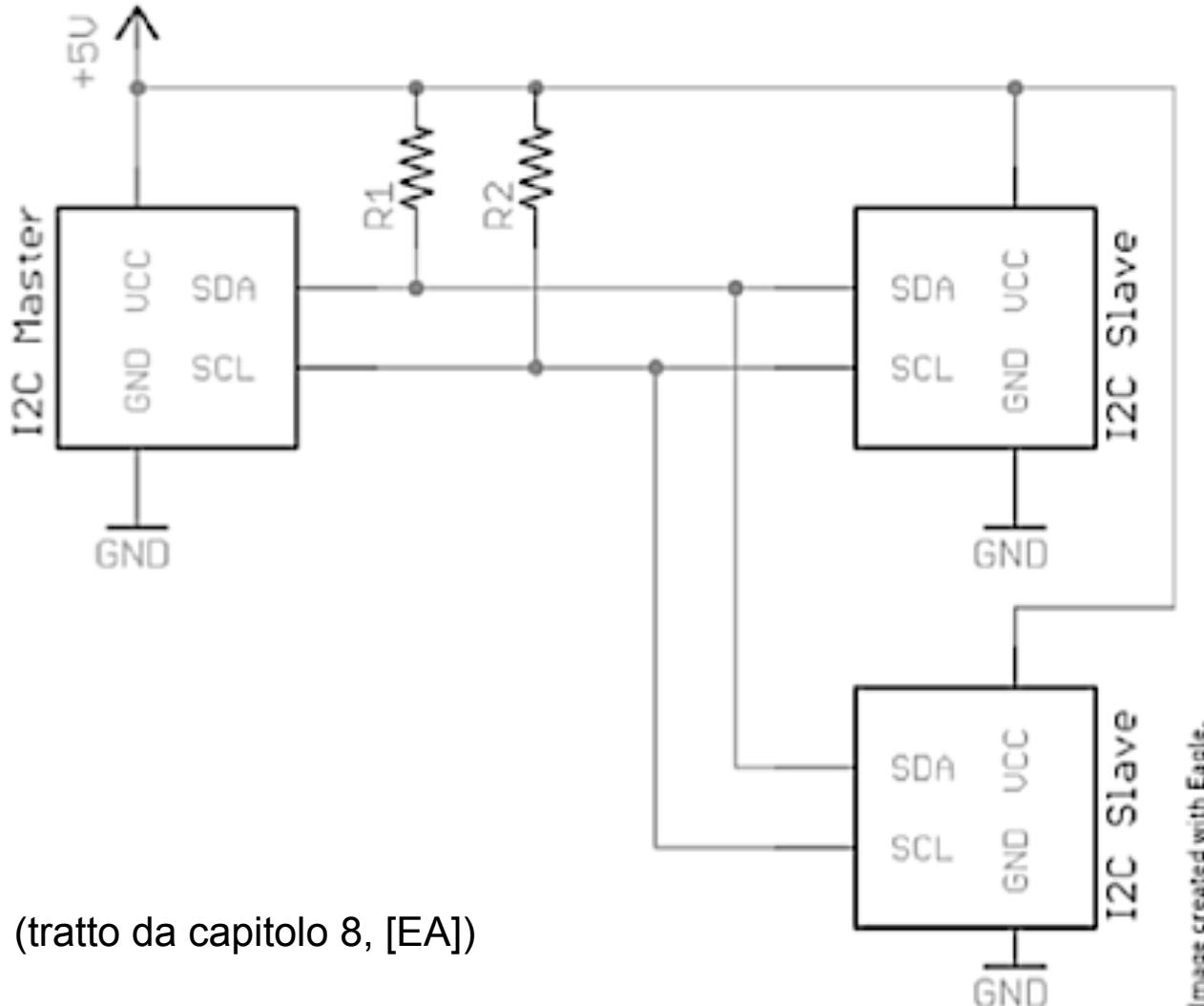
BUS/PROTOCOLLO I²C

- I²C ("eye squared see" o "eye two see") è un bus/protocollo standard
 - permette una comunicazione veloce, robusta, a 2 vie, utilizzando un numero minimo di pin
 - introdotto da Philips, all'inizio degli anni 80, poi diventato uno standard negli anni 90
 - chiamato anche "Two-wire" (TW) perché utilizza 2 linee per la comunicazione (clock, data)
- Specifiche
 - utilizza spazio di indirizzamento a 7 o 10 bit
 - bus speed arriva a 100 kbit/s in *standard mode* e 10 kbit/s in *low-speed mode*
 - versioni recenti hanno introdotto *fast mode* a 400 kbit/s, e *fast mode plus* (1Mbit/s) e *High-Speed mode* (3.4Mbit/s)

ARCHITETTURA I²C

- Architettura master-slave
 - un I²C bus è controllato da un master device
 - tipicamente il micro-controllore
 - contiene uno o più slave device che ricevono informazioni dal master
- Più device condividono le stesse due linee di comunicazione
 - un clock signal (*Serial Clock Line*, **SCL**), che serve per sincronizzare la comunicazione
 - una linea bidirezionale (*Serial Data Line*, **SDA**) per inviare e ricevere dati dal master agli slave

ARCHITETTURA I²C



(tratto da capitolo 8, [EA])

Figure 8-1: I²C reference hardware configuration

Image created with Eagle.

IL PROTOCOLLO MASTER-SLAVE

- La comunicazione viene sempre fatta partire dal master
 - gli slave possono solo rispondere
- Tutti i comandi inviati dal master sono ricevuti da tutti i dispositivi sul bus
 - tuttavia, ogni slave ha un proprio ID unico di 7 bit che viene specificato dal master quando inizia la comunicazione
 - solo lo slave target reagisce al messaggio inviato dal master
- Schema di comunicazione tipico:
 - Master invia start bit
 - Master invia l'id a 7 bit del dispositivo con cui vuole comunicare
 - Master invia read (0) o write (1) bit, a seconda che voglia scrivere dati nei registri del dispositivo o leggerli
 - Slave risponde con un ack (un ACK bit, che è attivo quando vale 0)
 - In write mode, il master invia un byte di informazioni alla volta e lo slave risponde con degli ACK. In read mode il master riceve 1 byte alla volta e invia un ACK dopo ogni byte
- Quando la comunicazione è finita, il master invia uno STOP bit

leggere resistenza di pull down
(es. nel pulsante tattile).

I2C IN ARDUINO/ATmega328P - LIBRERIA WIRE

- Arduino/ATmega328P supporta nativamente I2C
 - uso 2 pin analogici, A4 e A5 non possono essere usati per altri scopi.
 - questi pin sono multiplexed fra il convertitore analogico/digitale (ADC) e l'interfaccia HW per I2C
- Libreria di riferimento: **Wire**
 - documentazione: <http://arduino.cc/en/Reference/Wire>
 - classe Wire, estensione di Stream
 - metodi
 - `begin()`
 - `beginTransmission()`, `endTransmission()`
 - `write()`, `requestFrom()`, `read()`
 - `available()`
 - `onReceive()`, `onRequest()`

ESEMPIO INTERFACCIAMENTO I2C

- Esempio:
 - interfacciamento con sensore di temperatura TC74 che funziona con protocollo I2C
- Nota:
 - multiplexing dei pin: **SDA (data) e SCL (clock) pin sono collegati ad A4 e A5**
 - quando si inizializza la Wire library, questi pin vengono usati per I2C e non possono essere più usati per comunicazioni analogiche

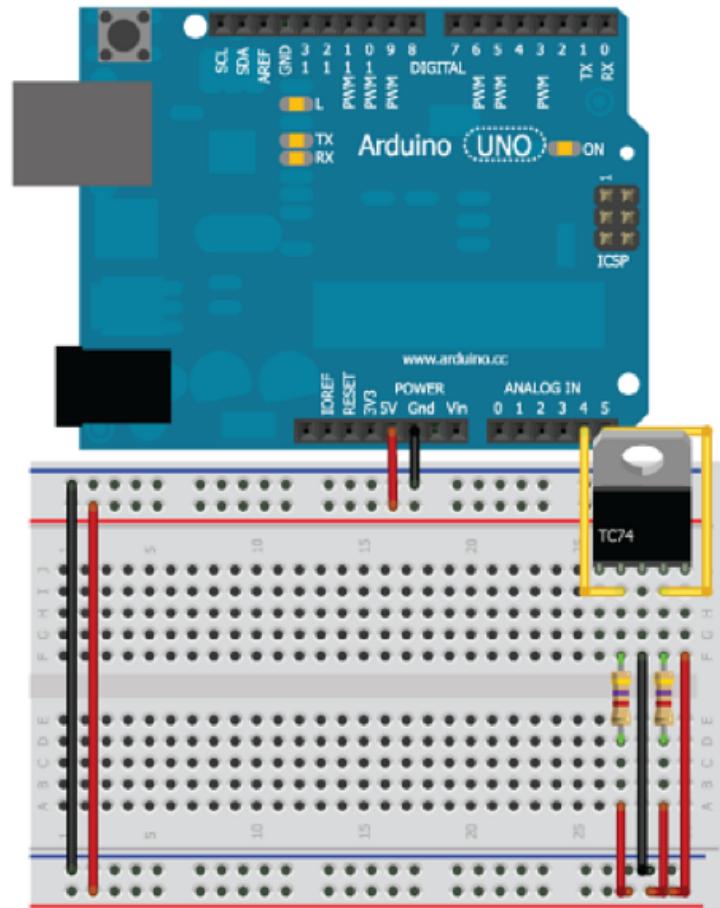
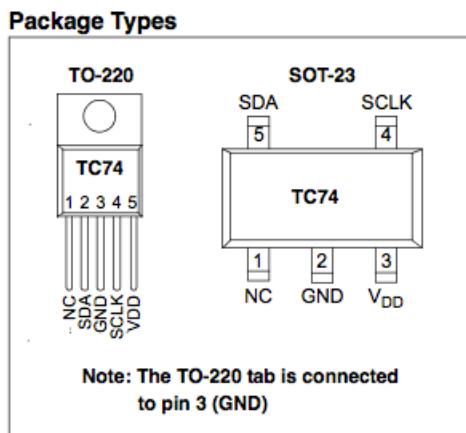


Figure 8-4: Temperature sensor

NOTA: nel circuito servono 2 resistenze di pull-up (ad es. da 4.7 KOhm)

LETTURA DATASHEET

- Datasheet del componente
 - <http://ww1.microchip.com/downloads/en/DeviceDoc/21462D.pdf>
- Include le informazioni sul protocollo:

Write Byte Format

S	Address	WR	ACK	Command	ACK	Data	ACK	P
	7 Bits			8 Bits		8 Bits		

Slave Address

Command Byte: selects
which register you are
writing to.

Data Byte: data goes
into the register set
by the command byte.

Read Byte Format

S	Address	WR	ACK	Command	ACK	S	Address	RD	ACK	Data	NACK	P
	7 Bits			8 Bits			7 Bits			8 Bits		

Slave Address

Command Byte: selects
which register you are
reading from.

Slave Address: repeated
due to change in data-
flow direction.

Data Byte: reads from
the register set by the
command byte.

Receive Byte Format

S	Address	RD	ACK	Data	NACK	P
	7 Bits			8 Bits		

S = START Condition

Data Byte: reads data from

P = STOP Condition

the register commanded by

Shaded = Slave Transmission

the last Read Byte or Write

Byte transmission.

DAL DATASHEET

4.0 REGISTER SET AND PROGRAMMER'S MODEL

TABLE 4-1: COMMAND BYTE DESCRIPTION (SMBUS/I²C READ_BYTE AND WRITE_BYTE)

Command	Code	Function
RTR	00h	Read Temperature (TEMP)
RWCR	01h	Read/Write Configuration (CONFIG)

TABLE 4-3: TEMPERATURE REGISTER (TEMP)

D[7]	D[6]	D[5]	D[4]	D[3]	D[2]	D[1]	D[0]
MSB	X	X	X	X	X	X	LSB

In temperature data registers, each unit value represents one degree (Celsius). The value is in 2's complement binary format such that a reading of 0000 0000b corresponds to 0°C. Examples of this temperature to binary value relationship are shown in Table 4-4.

TABLE 4-4: TEMPERATURE-TO-DIGITAL VALUE CONVERSION (TEMP)

Actual Temperature	Registered Temperature	Binary Hex
+130.00°C	+127°C	0111 1111
+127.00°C	+127°C	0111 1111
+126.50°C	+126°C	0111 1110
+25.25°C	+25°C	0001 1001
+0.50°C	0°C	0000 0000
+0.25°C	0°C	0000 0000
0.00°C	0°C	0000 0000
-0.25°C	-1°C	1111 1111
-0.50°C	-1°C	1111 1111
-0.75°C	-1°C	1111 1111
-1.00°C	-1°C	1111 1111
-25.00°C	-25°C	1110 0111
-25.25°C	-26°C	1110 0110
-54.75°C	-55°C	1100 1001
-55.00°C	-55°C	1100 1001
-65.00°C	-65°C	1011 1111

PROTOCOLLO

- Tabella 4-1 => 2 registri da leggere
 - uno contiene la temperatura in Celsius
 - uno la configurazione del chip
- Tabella 4-3 e 4-4 mostrano come le informazioni sono memorizzate nel registro a 8 bit
- Protocollo per leggere la temperatura:
 1. invia all'indirizzo del dispositivo in write mode e invia uno 0 ad indicare che vogliamo leggere dal data register
 2. invia all'indirizzo del dispositivo in read mode e richiede 8 bit (1 byte) di informazioni
 3. attende di ricevere tutti gli 8 bit

PROGRAMMA ARDUINO

```
//Include Wire I2C library
#include <Wire.h>

int temp_address = 72; //1001000 in decimale
void setup()
{
    Serial.begin(9600);
    Wire.begin(); /* lib init */
}

void loop()
{
    Wire.beginTransmission(temp_address);
    Wire.write(0);
    Wire.endTransmission();

    Wire.requestFrom(temp_address, 1);
    while(Wire.available() == 0);
    int c = Wire.read(); busy waiting: è di buona norma permettere a loop() di eseguire il ciclo completo.
    Serial.print(c);
    Serial.print("C ");
    delay(500);
}
```

- Wire.beginTransmission
=> inizia la comunicazione con lo slave
 - Wire.write
=> invia uno 0, indicando che vogliamo leggere il valore del registro data
 - Wire.endTransmission
=> invia uno stop ad indicare che abbiamo finito di scrivere
- poi
- Wire.requestFrom
=> il master aspetta di ricevere un byte
 - Wire.available()
=> verifica se il byte è arrivato
 - Wire.read()
=>legge il valore arrivato

Esempio tratto dal testo “Exploring Arduino” [EA], p. 171

Codice completo disponibile su git-hub:
<https://github.com/sciguy14/Exploring-Arduino>

ALTRI ESEMPI DI SENSORI I2C

- MPU6050
 - sensore combinato accelerometro+giroscopio
 - <http://invensense.com/mems/gyro/mpu6050.html>
- LSM330
 - sensore combinato bussola+accelerometro
 - http://www.st.com/web/en/catalog/sense_power/FM89/SC1448/PF253882

SPI

- Originariamente introdotto da Motorola, lo SPI bus implementa un protocollo seriale full-duplex che permette la comunicazione simultanea bidirezionale fra un master e uno o più slave
 - come I2C è un protocollo seriale, basato su schema master-slave
 - diversamente da I2C usa linee diverse per trasmettere e ricevere dati, e utilizza una linea per selezionare lo slave
- Non segue un vero e proprio standard formale
 - per cui ci possono essere lievi differenze implementative e relative all'insieme dei comandi supportati
- Dispositivi SPI sono anch'essi sincroni, come I2C
 - i dati sono inviati in sync con un segnale di clock condiviso (SCLK)

ARCHITETTURA SPI

- Architettura master-slave
 - un SPI bus è controllato da un master device
 - tipicamente il micro-controllore
 - contiene uno o più slave device che ricevono informazioni dal master
- Per comunicare ci sono 3 linee
 - uno shared clock signal (Shared Serial Clock, **SCK**)
 - serve per sincronizzare la comunicazione
 - una linea Master Out Slave In (**MOSI**)
 - serve per inviare i dati dal master allo slave
 - una linea Master In Slave Out (**MISO**)
 - serve per inviare i dati dallo slave al master
 - una linea SS (**Slave Select**)
 - serve per selezionare lo slave

ARCHITETTURA SPI

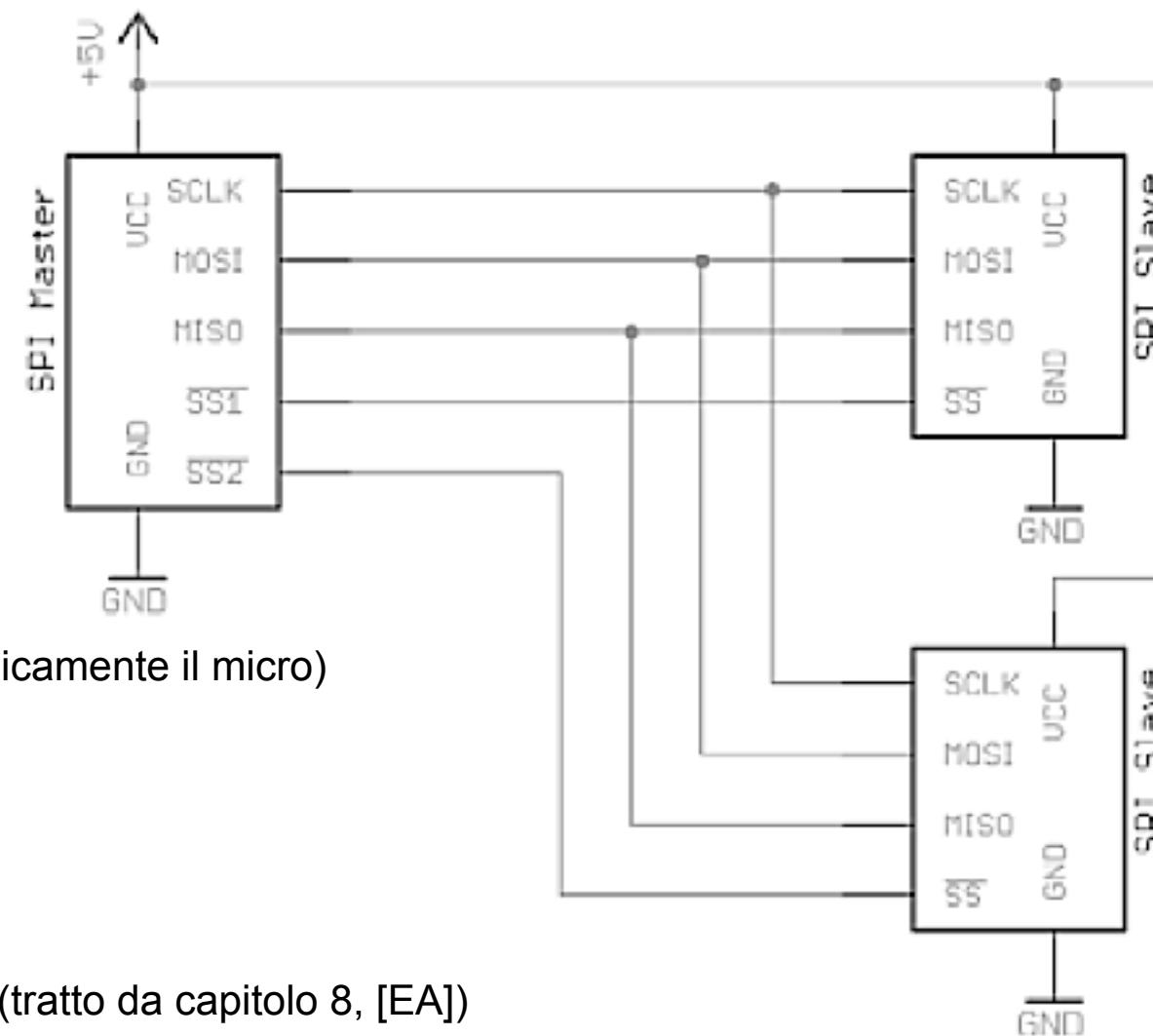
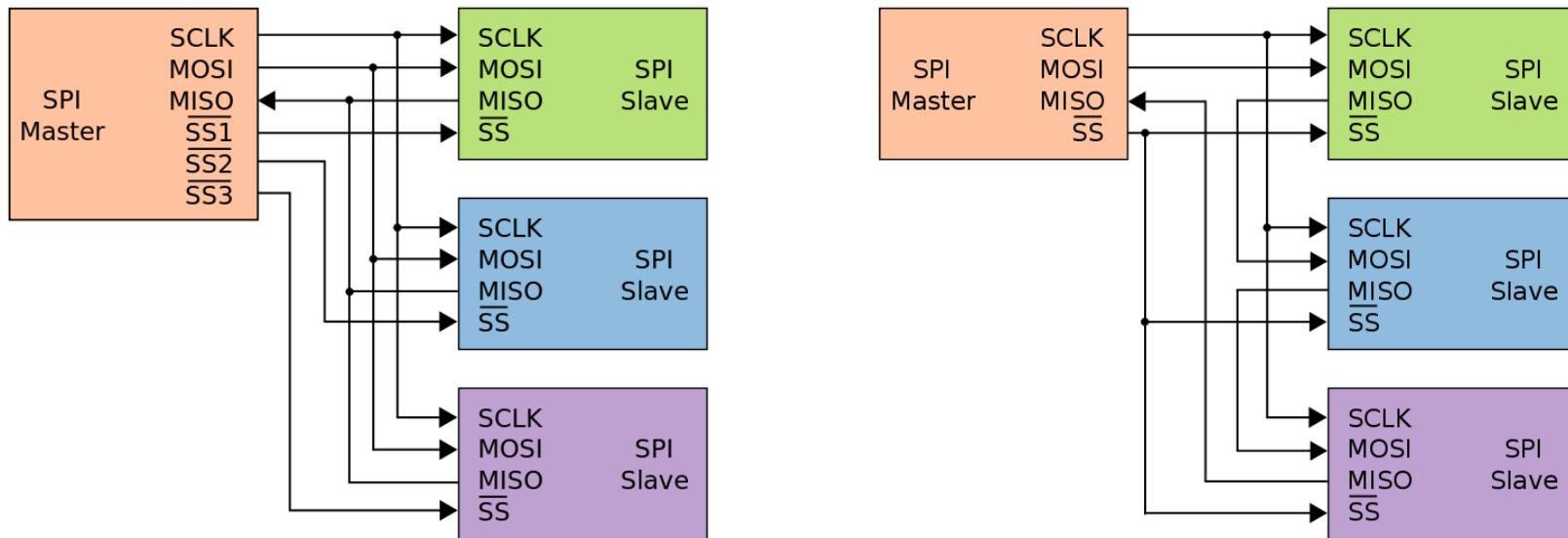


Image created with Eagle.

Figure 9-1: SPI reference hardware configuration

COLLEGAMENTI MULTISLAVE

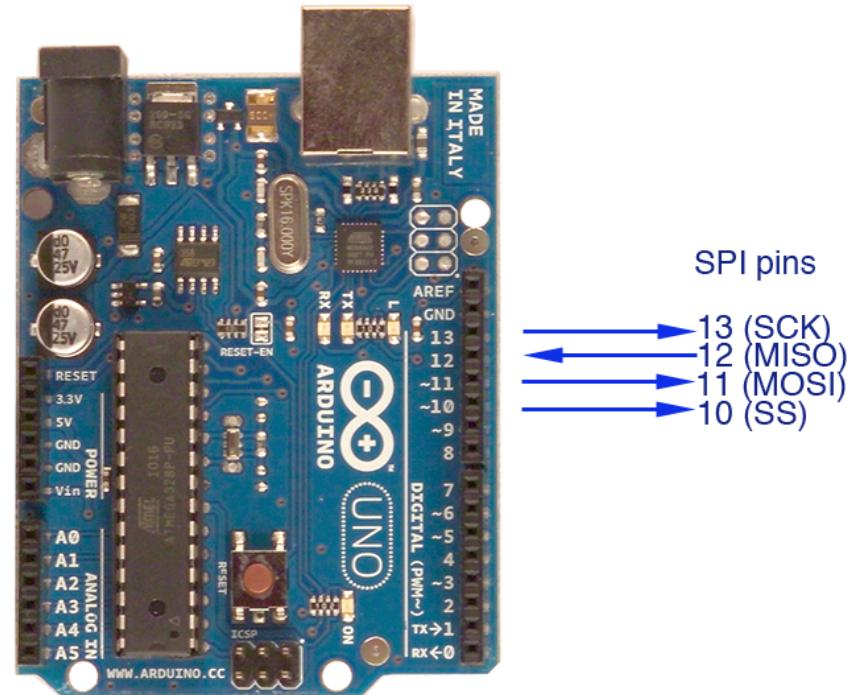
- Nel caso di collegamento serva collegare più slave allo stesso master ci sono due possibilità architetturali
 - utilizzare più linee SS
 - utilizzare un collegamento *daisy chain*, in cui il master comunica con tutti gli slave simultaneamente



(https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus)

SPI E ARDUINO/ATmega328P

- Arduino/ATmega328P supporta nativamente la comunicazione via SPI
 - uso multiplexed di 4 GPIO: pin 10, 11, 12, 13
 - 10 => SS
 - 11 => MOSI
 - 12 => MISO
 - 13 => SCK



LIBRERIA SPI PER WIRING

- E' disponibile una libreria che permette di interagire agilmente con dispositivi SPI
 - <https://www.arduino.cc/en/Reference/SPI>
- Per iniziare una sessione di uso della porta SPI, è disponibile il metodo:

```
SPI.beginTransaction(settings)
```

in cui si specifica la configurazione *settings* mediante la funzione SPISettings (vedere slide successiva)

- La fine della sessione si specifica mediante:

```
SPI.endTransaction();
```

SETTING

- La configurazione da specificare in `beginTransaction` include i seguenti parametri
 - massima velocità che il dispositivo può usare
 - se i dati sono in formato MSB (Most Significant Bit) o LSB (Least Significant Bit)
 - modalità relativa al sampling dei dati
 - 4 modi di trasmissione, che specificando il fronte del clock sul quale i dati sono campionati o inviati (clock phase) e se il clock è idle quando vale HIGH e LOW (clock polarity)

Mode	Clock Polarity (CPOL)	Clock Phase (CPHA)
SPI_MODE0	0	0
SPI_MODE1	0	1
SPI_MODE2	1	0
SPI_MODE3	1	1

- Esempio:

```
SPI.beginTransaction(SPISettings(14000000, MSBFIRST,  
SPI_MODE0));
```

TRASFERIMENTO DATI

- Per trasferire (inviare, ricevere) dati è disponibile il metodo **transfer()**:

```
receivedVal = SPI.transfer(val)
receivedVal16 = SPI.transfer16(val16)
SPI.transfer(buffer, size)
```

val: byte da inviare sul bus

val16: word da inviare nel bus

buffer:array di byte da inviare/ricevere

- SPI transfer permette di inviare e ricevere simultaneamente
 - è possibile trasferire anche più byte, specificando un buffer

PER INTERAGIRE CON UNO SLAVE

- Sequenza di azioni da fare per interagire con un dispositivo slave:
 - si setta lo slave select pin a LOW
 - mediante digitalWrite sul pin SS
 - si chiama SPI.transfer() per trasferire i dati
 - si setta il pin SS a HIGH

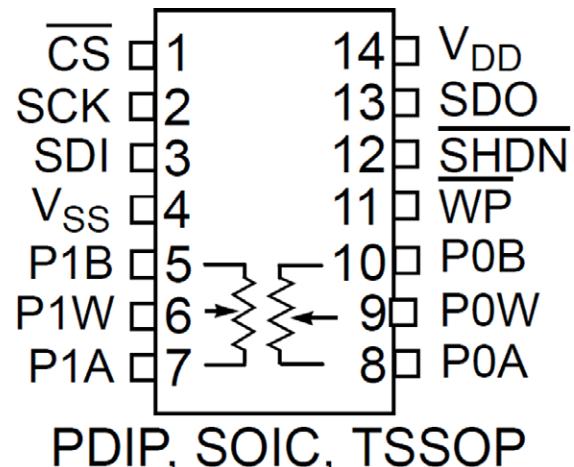
ESEMPIO INTERFACCIAMENTO SPI

- Interfacciamento di un potenziometro digitale: il **Microchip MCP4231**
 - come un normale potenziometro, ha un wiper (~spazzola) regolabile che determina la resistenza fra il terminale del wiper e uno degli altri terminali
 - variante 103E => range della resistenza è da **0 a 10 kOhm**
- MCP4231 ha 2 potenziometri sul medesimo chip
 - ognuno con risoluzione 7 bit



DATASHEET: PINOUT

- **Pins P0A, P0W, and P0B**
 - pin di controllo del primo potenziometro
- **Pins P1A, P1W, and P1B**
 - pin di controllo del secondo potenziometro
- VDD: alimentazione (5V)
- VSS: terra
- **CS: è l'SS pin dell'interfaccia SPI**
 - la barra sopra indica che è attivo quando il segnale è LOW
 - ovvero: 0V significa che il chip è selezionato, 5V significa che non è selezionato
- **SDI, SDO:**
 - questi pin corrispondono al serial data in e out, ovvero MOSI and MISO in SPI
- **SCK: è il clock di SPI**
- SHDN e WP
 - sono usati per funzioni di shutdown e write protect. Non sono usati. WP è disconnesso e SHDN è attivo LOW - quindi deve essere sempre tenuto alto (connesso a 5V) perché vogliamo sempre tenere il chip attivo



Credit: © 2013 Microchip Technology, Inc.

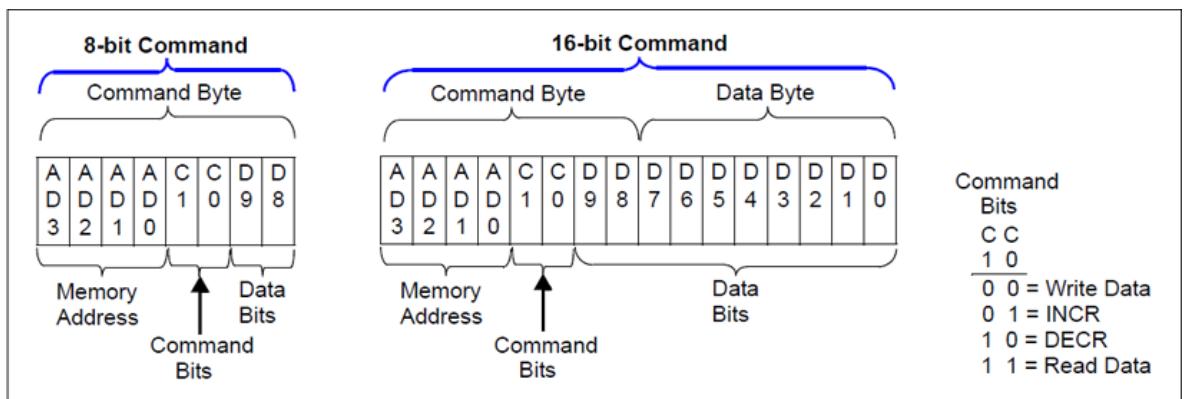
DATASHEET: PARAMETRI E COMANDI

- La resistenza del potenziometro è indicata come R_{AB}
- Il wiper pin ha una propria resistenza (R_W): da 75 a 160 Ohm
- Comandi: 2 tipi
 - a 8 bit: permette di incrementare il potenziometro usando un singolo byte
 - a 16 bit: permette di settare lo stato completo

AC/DC CHARACTERISTICS (CONTINUED)

DC Characteristics		Standard Operating Conditions (unless otherwise specified) Operating Temperature $-40^{\circ}\text{C} \leq T_A \leq +125^{\circ}\text{C}$ (extended) All parameters apply across the specified operating ranges unless noted. $V_{DD} = +2.7\text{ V}$ to 5.5 V , $5\text{ k}\Omega$, $10\text{ k}\Omega$, $50\text{ k}\Omega$, $100\text{ k}\Omega$ devices. Typical specifications represent values for $V_{DD} = 5.5\text{ V}$, $T_A = +25^{\circ}\text{C}$.				
Parameters	Sym	Min	Typ	Max	Units	Conditions
Resistance ($\pm 20\%$)	R_{AB}	4.0	5	6.0	$\text{k}\Omega$	-502 devices (Note 1)
		8.0	10	12.0	$\text{k}\Omega$	-103 devices (Note 1)
		40.0	50	60.0	$\text{k}\Omega$	-503 devices (Note 1)
		80.0	100	120.0	$\text{k}\Omega$	-104 devices (Note 1)
Resolution	N	257		Taps	8-bit	No Missing Codes
		129		Taps	7-bit	No Missing Codes
Step Resistance	R_S	—	$R_{AB} / (256)$	—	Ω	8-bit
		—	$R_{AB} / (128)$	—	Ω	7-bit
Nominal Resistance Match	$ R_{AB0} - R_{AB1} / R_{AB}$	—	0.2	1.25	%	MCP42X1 devices only
		$ R_{BW0} - R_{BW1} / R_{BW}$	—	0.25	1.5	%
Wiper Resistance (Note 3, Note 4)	R_W	—	75	160	Ω	$V_{DD} = 5.5\text{ V}$, $I_W = 2.0\text{ mA}$, code = 00h
		—	75	300	Ω	$V_{DD} = 2.7\text{ V}$, $I_W = 2.0\text{ mA}$, code = 00h

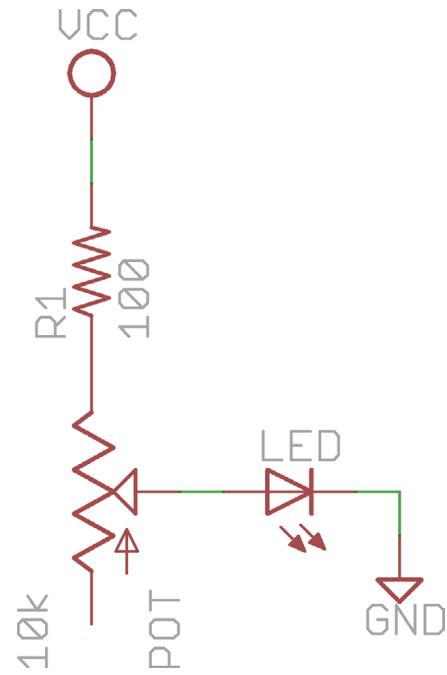
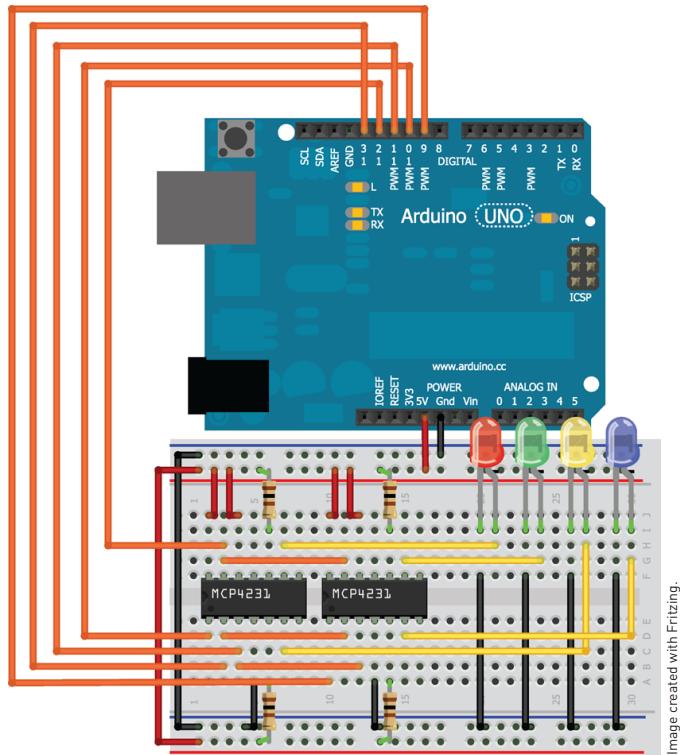
Credit: © 2013 Microchip Technology, Inc.



: © 2013 Microchip Technology, Inc.

ESEMPIO: CONTROLLO INTENSITÀ LED

- Collegamento 2 potenziometri per controllo intensità 4 led (esempio in [EA], cap. 9)



ESEMPIO: SORGENTI

```
#include <SPI.h>

const int SS1 = 10; // Slave Select Chip 1
const int SS2 = 9; // Slave Select Chip 2
const byte REG0=B00000000; // Register 0 Write command
const byte REG1=B00010000; // Register 1 Write command

void setup()
{
    /* set pin directions for SS */
    pinMode(SS1, OUTPUT);
    pinMode(SS2, OUTPUT);

    /* initialize SPI */
    SPI.begin();
}
```

(tratto da cap. 9, [EA])

ESEMPIO: SORGENTI

```
void loop() {
    for (int i=0; i<=128; i++)
    {
        setLed(SS1, REG0, i);
        setLed(SS1, REG1, i);
        setLed(SS2, REG0, i);
        setLed(SS2, REG1, i);
        delay(10);
    }
    delay(300);

    for (int i=128; i>=0; i--)
    {
        setLed(SS1, REG0, i);
        setLed(SS1, REG1, i);
        setLed(SS2, REG0, i);
        setLed(SS2, REG1, i);
        delay(10);
    }
    delay(300);
}
```

```
/* this will set 1 LED */  
/* to the specified level */  
/* Chip 1 (SS 10) Register 0 is Red */  
/* Chip 1 (SS 10) Register 1 is Yellow */  
/* Chip 2 (SS 9) Register 0 is Green */  
/* Chip 2 (SS 9) Register 1 is Blue */  
void setLed(int SS, int reg, int level)  
{  
    /* Set the given SS pin low */  
    digitalWrite(SS, LOW);  
    /* Choose the register to write to */  
    SPI.transfer(reg);  
    /* Set the LED level (0-128) */  
    SPI.transfer(level);  
    /* Set the given SS pin high again */  
    digitalWrite(SS, HIGH);  
}
```

(tratto da cap. 9, [EA])

CONFRONTO I²C ed SPI

- Molti dispositivi - inclusi accelerometri, potenziometri digitali, display..
 - sono disponibili sia su I²C, sia su SPI
- Confronto/vantaggi
 - I²C
 - richiede solo 2 linee
 - modalità indirizzamento slave più agile/aperto
 - SPI
 - può operare a velocità più elevate
 - in generale più semplice da usare
 - non sono richieste resistenze di pull-up

BIBLIOGRAFIA E SITOGRAFIA

- [EA] Jeremy Blum. *Exploring Arduino*. Wiley
- Wiring
 - <http://wiring.org.co/>
- MikroElektronika books about micro-controllers
 - <http://www.mikroe.com/chapters/view/74/pic-basic-book-chapter-1-world-of-microcontrollers/>

Programmazione Sistemi Embedded e IoT
Ingegneria e Scienze Informatiche - UNIBO
a.a 2020/2021
Docente: Prof. Alessandro Ricci

[modulo-lab-1.2]
PROGRAMMAZIONE
ARDUINO - ASPETTI BASE

OBIETTIVO

- Primo esempi di programmazione di base su Arduino
 - input/output digitali
 - output PWM
 - input analogici
 - interruzioni
 - timer
 - seriale TTL

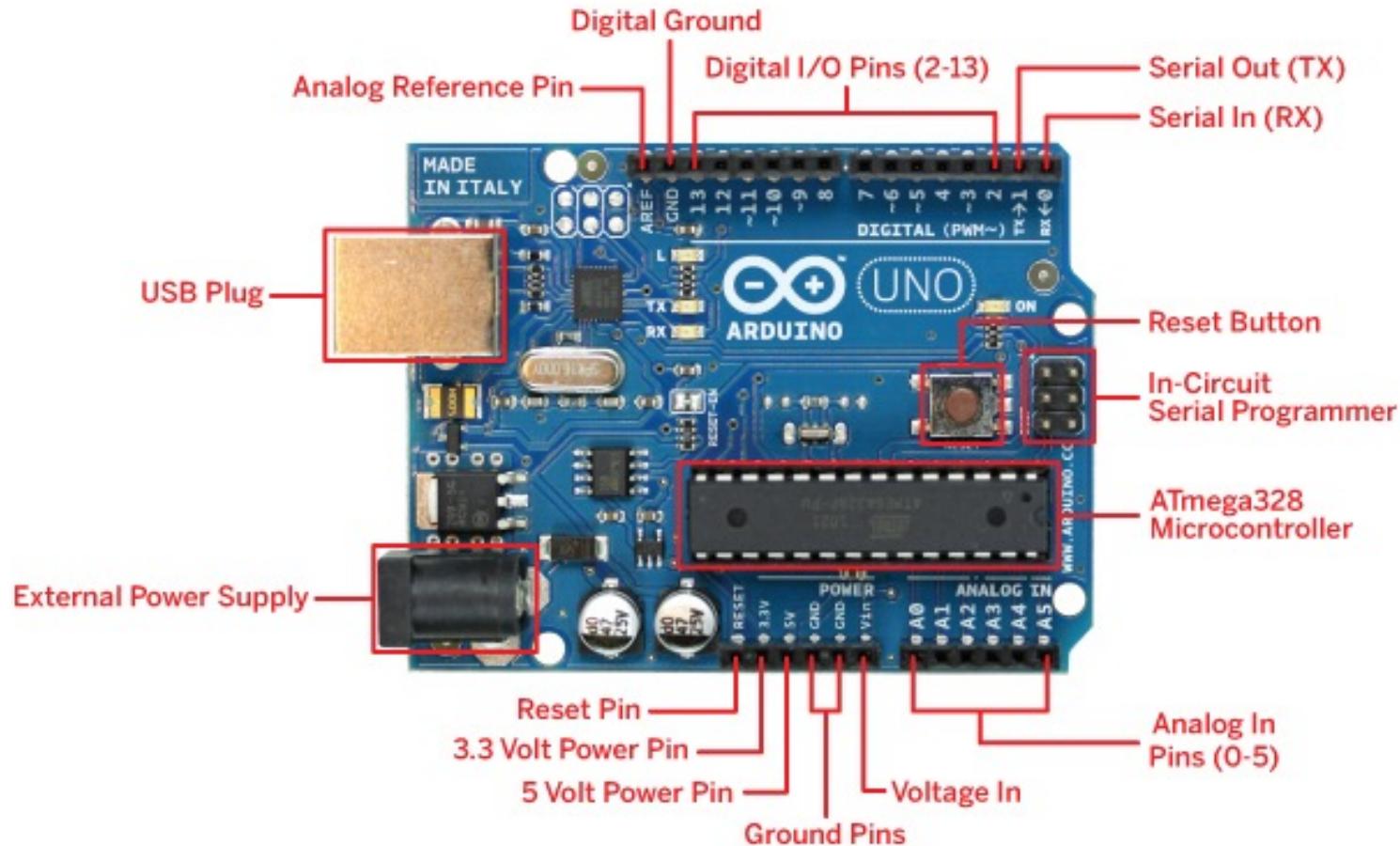
ARDUINO - INTRODUZIONE

- Il progetto Arduino (http://it.wikipedia.org/wiki/Arduino_hardware)
 - ...prese avvio in Italia a Ivrea nel 2005, con lo scopo di rendere disponibile, a progetti di **Interaction design** realizzati da studenti, un dispositivo per il controllo che fosse più economico rispetto ai sistemi di prototipazione allora disponibili.
 - team composto da Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, e David Mellis
 - I progettisti riuscirono a creare una piattaforma di semplice utilizzo ma che, al tempo stesso, permetteva una significativa riduzione dei costi rispetto ad altri prodotti disponibili sul mercato.
 - A ottobre 2008, in tutto il mondo erano già stati venduti più di 50.000 esemplari di Arduino...
- Vocazione **open-source**
 - sia hardware che software
- Visione **makers** - prototipazione, tinkering,...
- Sito: <http://arduino.cc>

ARDUINO - TEAM (PRE 2015)



IN LAB: ARDUINO UNO



ARDUINO UNO

- Componenti principali
 - MCU ATMega 328P
 - 8 bit, 16 MHz
 - Flash memory: 32 KB
 - SRAM memory: 2 KB, EEPROM: 1 KB
 - 14 pin digitali input/output
 - 6 possono essere usati come uscite PWM
 - 6 input analogici
 - connettore USB
 - power jack
 - ICSP header
 - pulsante di reset

Altre specifiche

Operating voltage: 5V

Input voltage (recom.): 7-12 V

DC current per I/O pin: 40 mA

DC current per 3.3V: 50mA

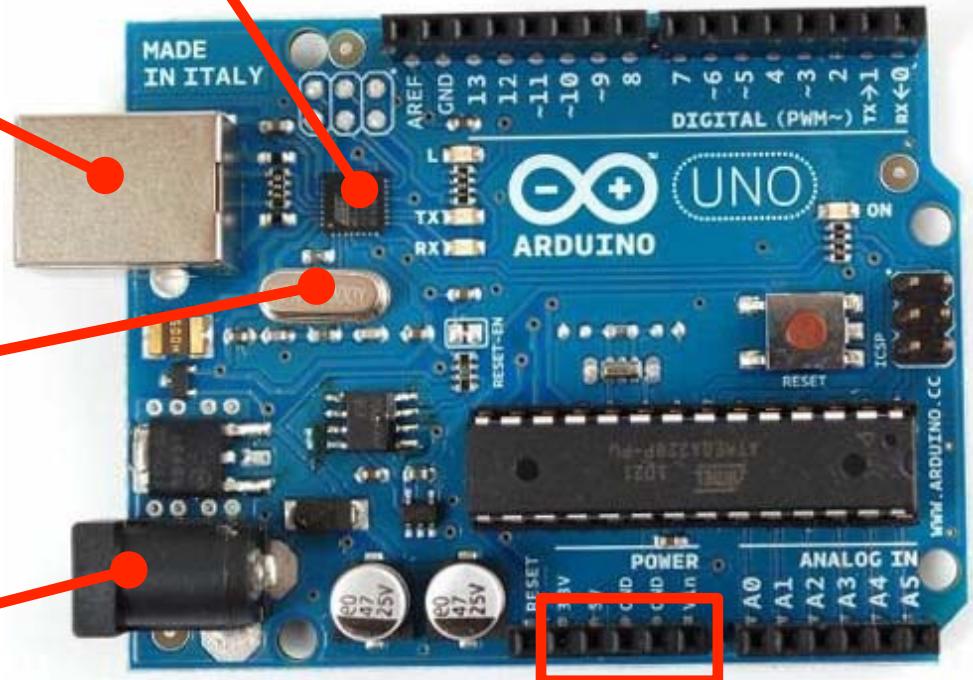
ARDUINO UNO

Alimentazione esterna da PC
via USB (+collegamento
seriale)

µC per convertire i segnali
seriali in segnali USB

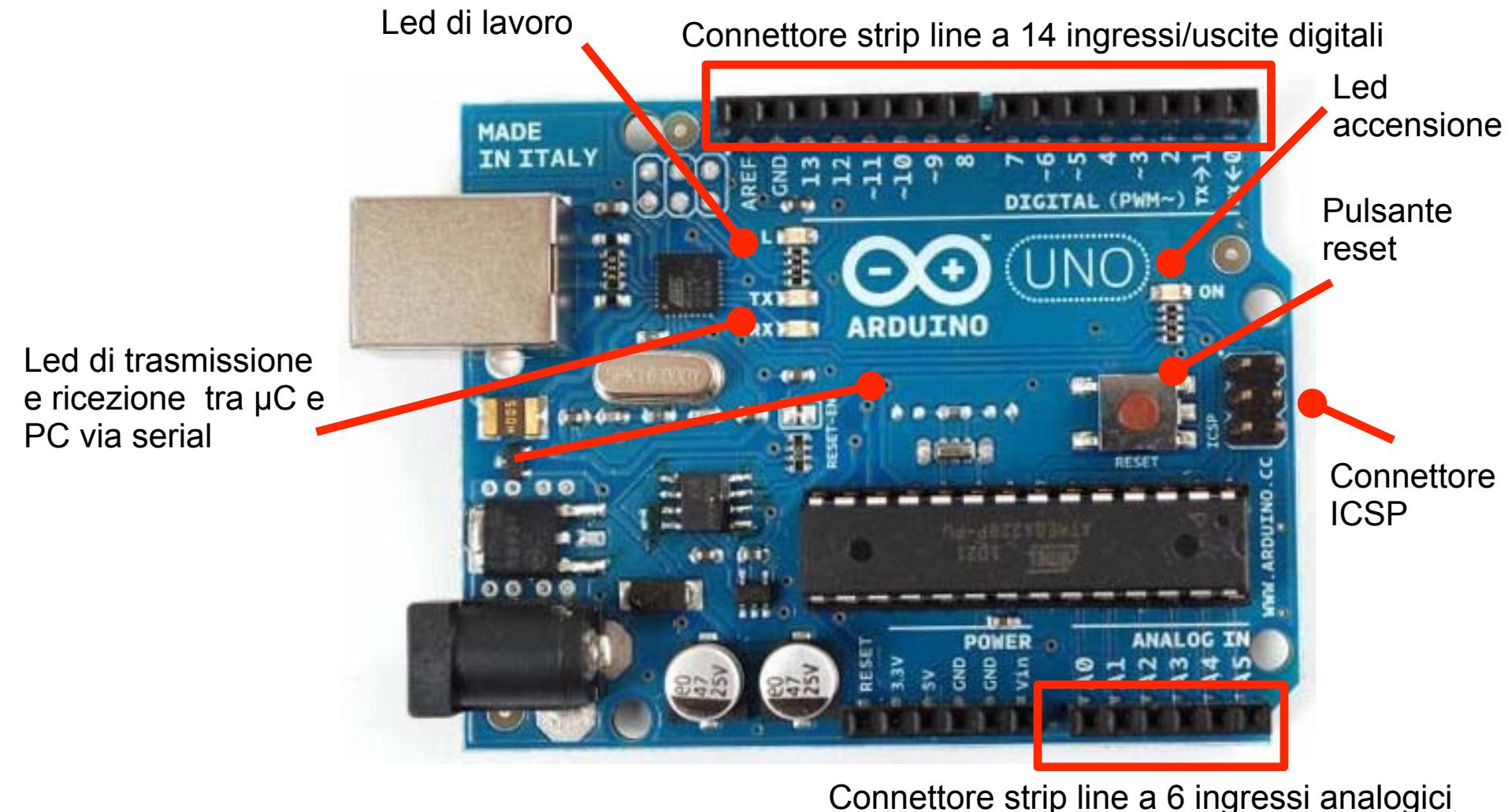
Quarzo per la generazione
del clock 16 MHz

Alimentazione esterna da Jack
(6-20 Volt)



Pin che riportano tensione di riferimento
GND, 5 V, 3.3 V e la tensione Vin di
alimentazione

ARDUINO UNO



SPECIFICHE PIN

- **Pin digitali 0..13**
 - operano ad una tensione di 5V
 - possono fornire o assorbire fino a 40mA di corrente
- **Ingressi analogici A0...A5**
 - operano sempre a 5V
 - **risoluzione a 10bit**
 - valori di tensione 0-5 Volt vengono convertiti mediante un convertitore analogico-digitale (ADC) in valori a 10 bit [0..1023 ($2^{10}-1$)]
 - mediante il pin 21 (AREF) è possibile specificare/ cambiare il valore di riferimento (il valore massimo) per l'ADC

ALIMENTAZIONE ESTERNA

- Via PC collegato mediante cavo USB
- Via batterie
 - dimensionamento della batteria dipende da tempo d'uso, corrente/tensione richiesti dal sistema e dalla dimensione massima batterie
 - Esempio:
 - consumo stimato di 150mA
 - durata almeno 8 ore
$$\Rightarrow \text{capacità batteria} \geq 150\text{mA} \cdot 8 = 1200\text{mAh}$$

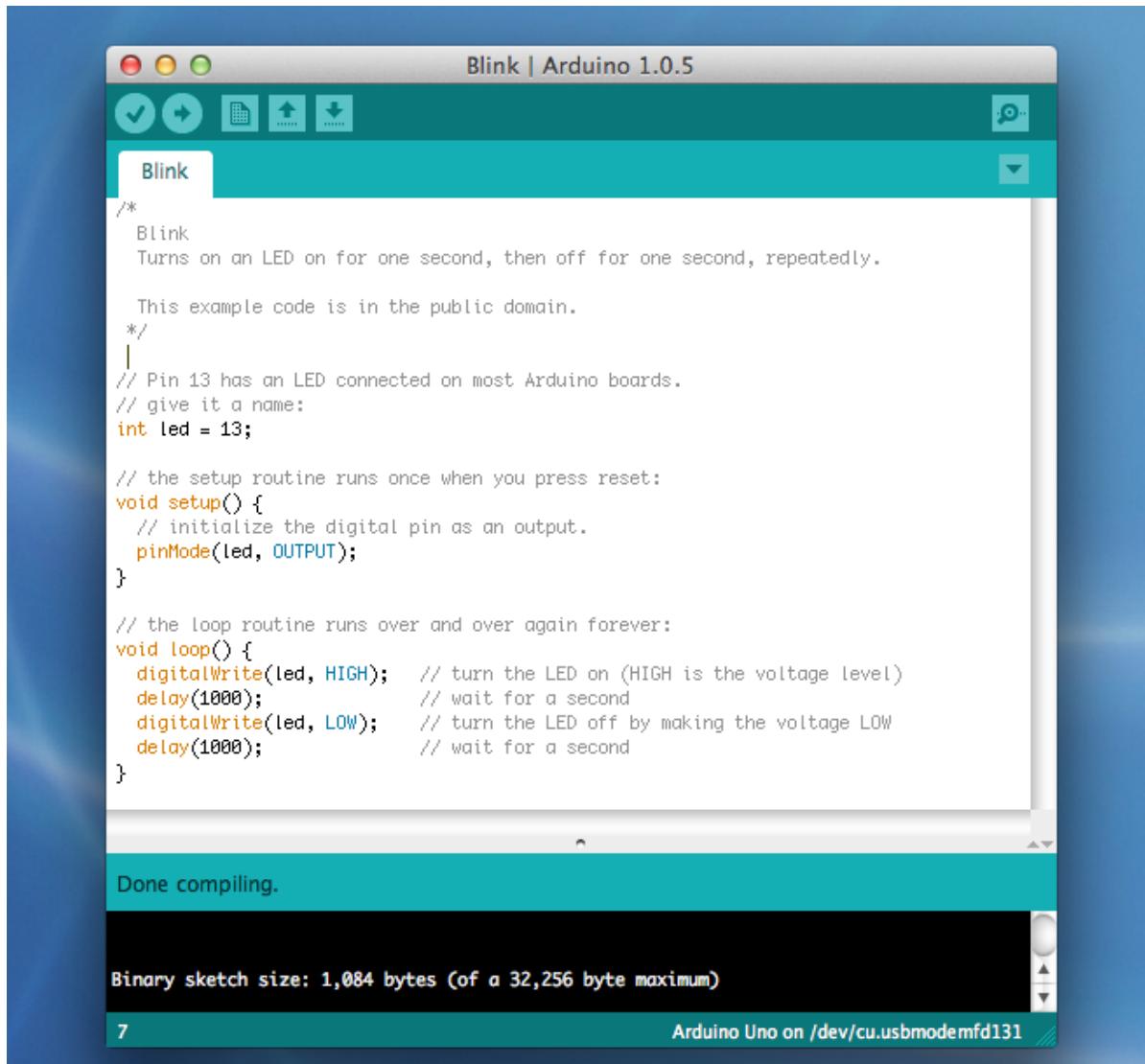
PROGRAMMAZIONE

- Il framework di riferimento per la programmazione di Arduino si chiama **Wiring**
 - framework open-source per microcontrollori
 - cross-platform, numerose piattaforme supportate - AVR Xmega, AVR Tiny, TI MSP430, Microchip PIC24/32...
 - basato su **GNU GCC**, con insieme di librerie custom
 - linguaggio C/C++
 - AVR gcc tool chain - compilatore avr-gcc
 - libreria custom per accedere e interagire con HW
 - <http://www.nongnu.org/avr-libc/>
 - ampia documentazione/tutorial
 - <http://wiring.org.co/learning/tutorials/>
 - link: <http://arduino.cc/en/Reference/HomePage>, <http://wiring.org.co/>
- Supportato direttamente da **Arduino IDE**

AMBIENTI DI SVILUPPO E TOOL

- **Arduino IDE**
 - IDE che permette lo sviluppo e deployment di programmi sulla scheda
 - va installato ed eseguito su un PC che funge da host, collegato alla scheda Arduino mediante cavo USB
 - basato su framework di programmazione Wiring e Processing
 - implementato in Java
 - **<http://arduino.cc/en/Main/Software>**
- **Eclipse**
 - AVR plug-in + C/C++ plug-in opportunamente configurato
 - AVR GCC tool chain opportunamente installata
 - <http://playground.arduino.cc/Code/Eclipse>
- **AVR Studio 6**
 - IDE e tool per programmare i MCU
- **Simulatori/Emulatori**
 - vari disponibili. a prezzi diversi. alcuni open-source

ARDUINO IDE



ALTRI IDE

- In caso di progetti articolati, sono disponibili altri IDE, con funzionalità più avanzate e utili per gestire complessità
 - Arduino IDE for Visual Studio (Visual Micro)
 - [https://marketplace.visualstudio.com/items?
itemName=VisualMicro.ArduinoIDEforVisualStudio](https://marketplace.visualstudio.com/items?itemName=VisualMicro.ArduinoIDEforVisualStudio)
 - PlatformIO
 - <https://platformio.org/>

STRUTTURA DI UN PROGRAMMA IN WIRING - SUPER LOOP

- In Wiring si usa il super-loop come architettura di controllo
- Due procedure principali
 - **setup()**
 - istruzioni da eseguire appena avviato il programma, eseguite una volta sola
 - **loop()**
 - corpo del ciclo di controllo, eseguito ciclicamente

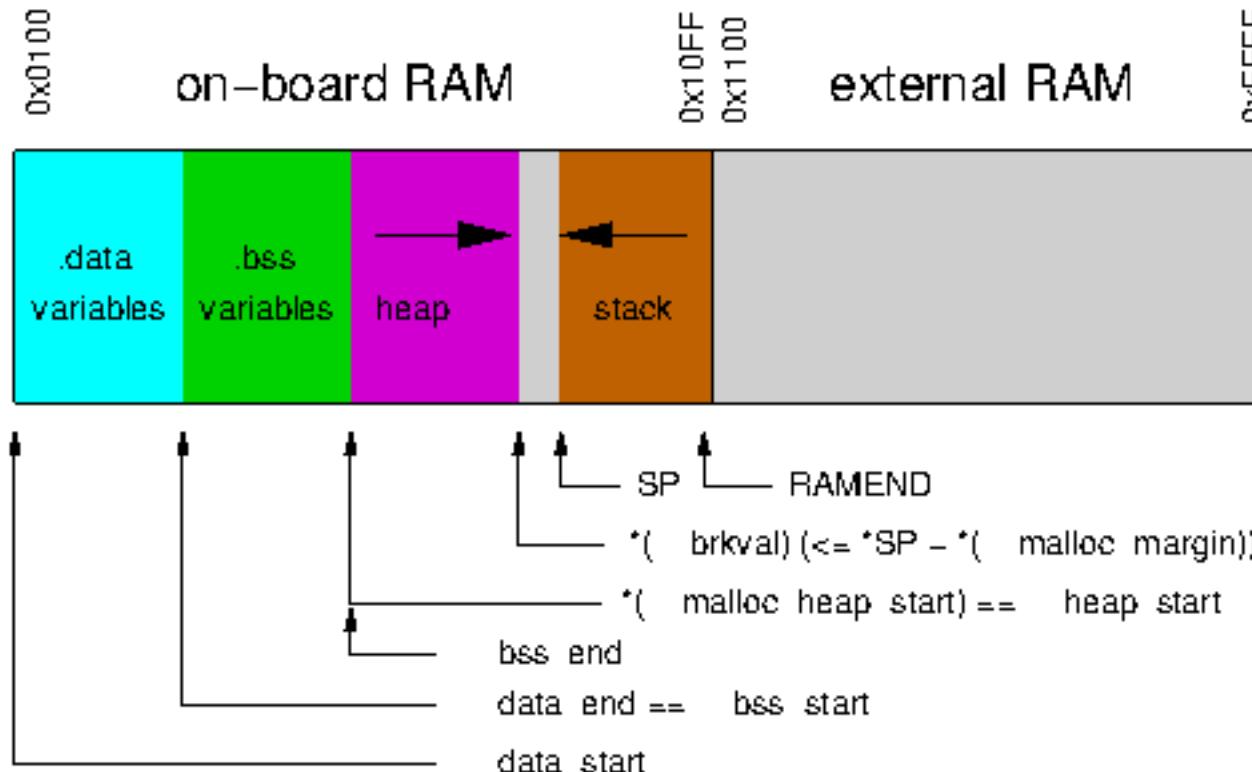
CARICAMENTO ED ESECUZIONE PROGRAMMI

- Il caricamento ed avvio di un programma può avvenire in due modi
 - da PC connesso, sfruttando il **bootloader** precaricato sul MCU
 - permette di fare l'upload di un programma compilato in memoria, senza la necessità di HW esterno
 - <http://arduino.cc/en/Tutorial/Bootloader>
 - mediante dispositivo esterno (chiamato ISP, In-System Programmer) che permette di programmare il controllore mediante interfaccia ICSP (In-Circuit-Serial-Programming)
 - <http://arduino.cc/en/Main/ArduinoISP>
- Il primo modo è il più usato e pratico

BOOTING - DETTAGLIO

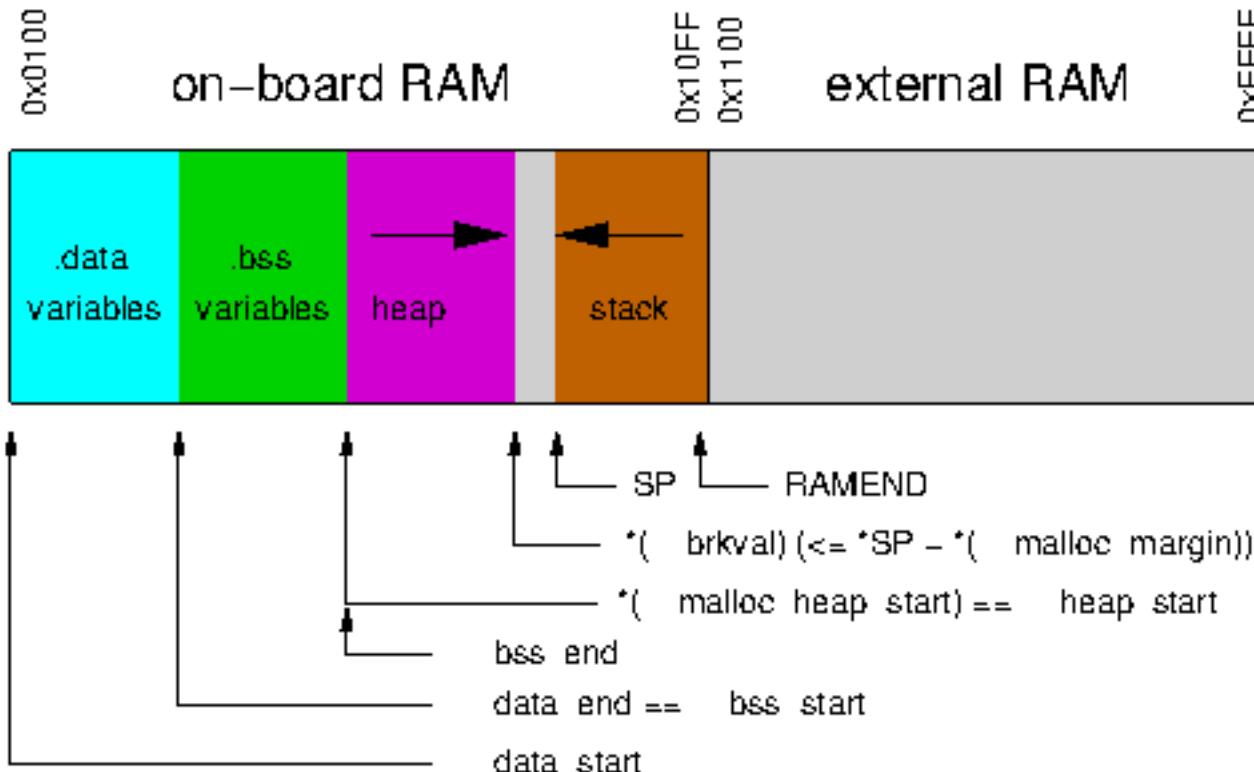
- Ogni volta che Arduino viene resettato (o via pulsante, o via pin, o via USB), entra in esecuzione il bootloader per qualche secondo
- Se esso riceve un apposito comando dall'IDE (via seriale), allora cerca di caricare il programma che gli invia l'IDE
 - il programma viene caricato nel resto della memoria e quindi eseguito
 - ciò avviene ogni volta si seleziona UPLOAD nell'IDE => viene inviato un segnale di reset alla scheda e quindi inviato lo sketch
- Se non riceve nessun programma o comando dall'IDE, allora parte eseguendo l'ultimo sketch caricato in memoria

RUNTIME - MEMORY MAP



- Sullo spazio di indirizzamento 0..0xFF sono mappati i registri interni (incluso I/O, timer..)
 - quindi il primo indirizzo utile è 0x100 (\$100) ovvero 256 e l'ultimo è 0x8FF (\$8FF) ovvero 2303 (essendoci 2KiB, $2303-256=2047$).

RUNTIME - MEMORY MAP



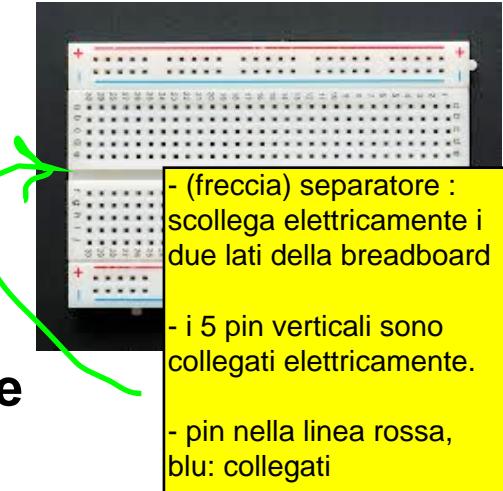
- Memory map realizzata dal compilatore avr-gcc / segmenti:
 - .data (variabili globali), .bss (costanti), heap (memoria dinamica), stack
- Note
 - heap e stack crescono in direzioni opposte
 - possono collidere in caso di mem overflow

SETUP AMBIENTE DI LAVORO

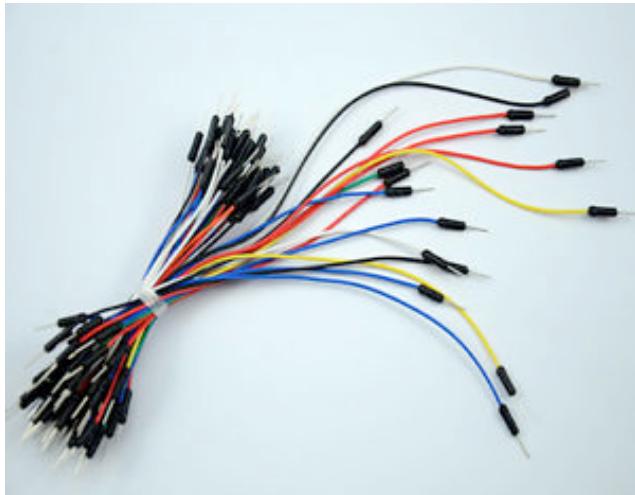
- Hardware
 - host computer + Arduino UNO + cavo
 - componenti elettronici di base
 - per questo lab
 - resistori 220 Ω , 1K Ω , 10 K Ω
 - led, pulsante, potenziometro
- Strumenti
 - breadboard
 - tester o multmetro
- Software
 - Arduino IDE
 - framework Wiring
 - per disegnare i circuiti:
 - Eagle
 - **Fritzing**

MATERIALE: LA BREADBOARD

- Basetta **o piastra** di plastica con fori al di sotto dei quali si trovano dei contatti che mettono in collegamento i componenti elettronici inseriti nei fori medesimi.
 - permette di realizzare dei circuiti elettronici senza saldature
 - corrente sopportata dai contatti è limitata (1 A).
 - varie dimensioni
- Contiene due serie di 5 fori **collegati elettricamente tra di loro verticalmente**
 - ai due estremi, superiore e inferiore: due file di fori collegati elettricamente orizzontalmente nei quali collegare il positivo e il negativo dell'alimentazione
 - a cavallo del solco centrale si inseriscono i piedini dei circuiti integrati in modo che ogni piedino abbia un collegamento univoco
 - ai bordi ci sono le file che tipicamente si usano per l'alimentazione



MATERIALE: CAVI, JUMPERS E PONTICELLI



(maschio-maschio)



ATTENZIONE A...

- ...**corto circuiti**, da evitare
 - corto circuito = collegamento fra due punti di un circuito che hanno resistenza nulla, imponendo quindi una tensione nulla (o trascurabile) ai suoi capi e non imponendo vincoli sulla corrente che può passare
 - es: collegamento diretto fra VCC e GND
 - possono danneggiare componenti e dispositivi collegati +
 - anche Arduino stesso..
- ...alla **quantità di corrente massima** che può attraversare un componente
 - può danneggiare il componente
- ...alla quantità di corrente massima che può essere **assorbita** da **pin** del microcontrollore / chip / componenti
- ...alla **polarità** dei componenti
 - es: diodi/led

PRIME ATTIVITA' IN LAB

- step #0 - Empty
- step #1 - Blinking led
- step #2 - Fading
- step #3 - Button-Led
- step #4 - Tune
- step #5 - Counting
- step #6 - Even-driven blinking
- step #7 - Serial echo

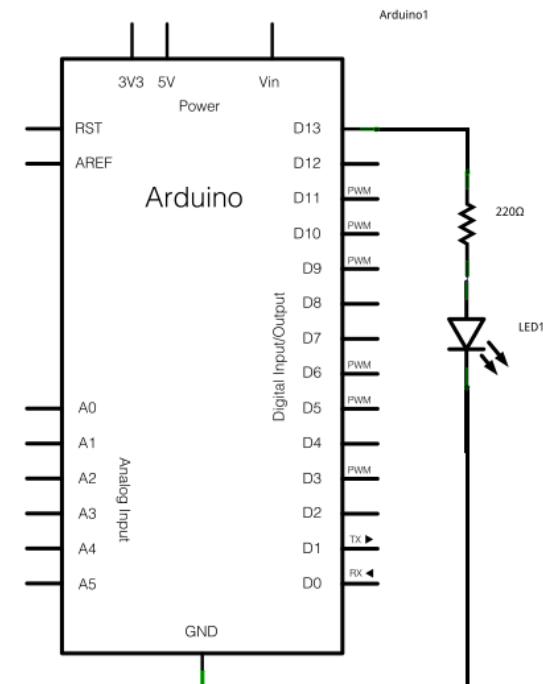
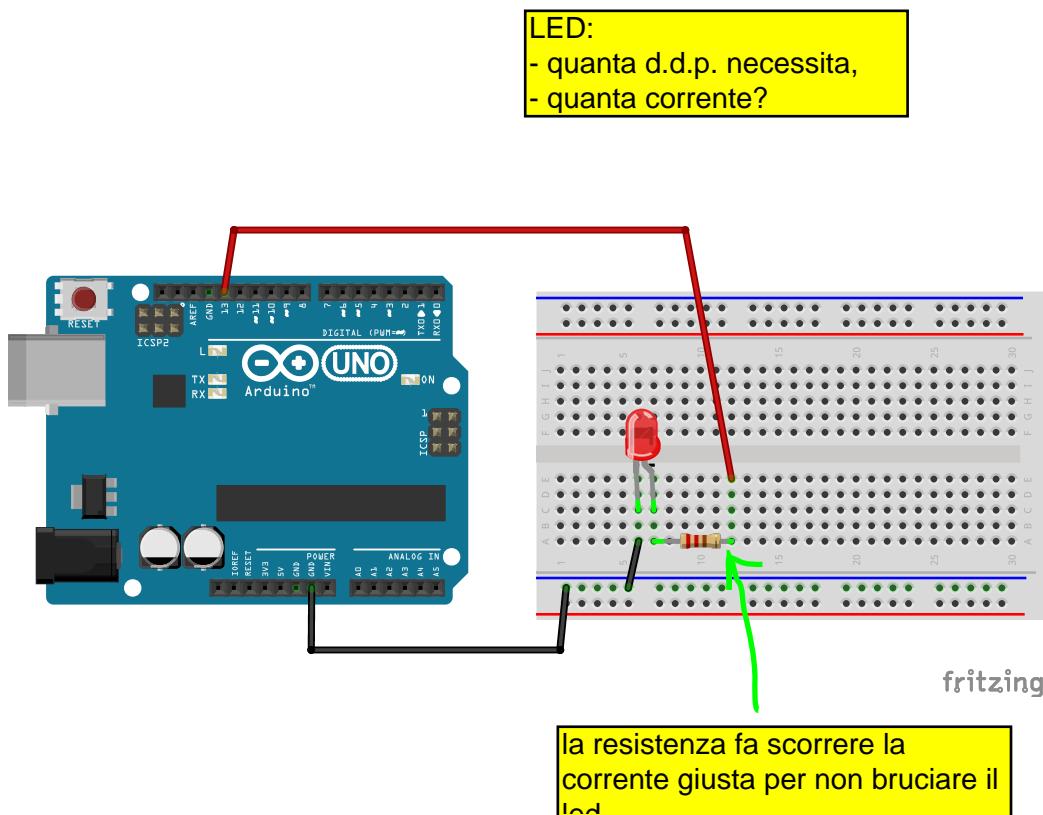
STEP #0 - EMPTY

- Descrizione
 - esempio di organizzazione di progetto su più file

STEP #1 - “HELLO BLINKING WORLD”

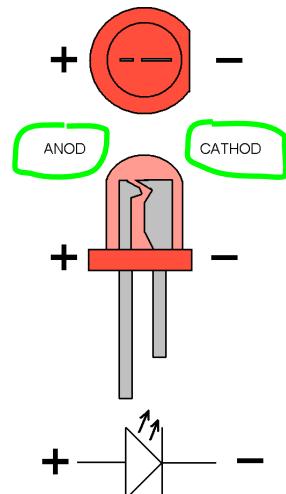
- Descrizione
 - sistema con un led. Il sistema deve far lampeggiare il led periodicamente, con periodo 1000 ms
- HW
 - led, resistore 220 Ohm
 - anche resistori con valori maggiori ($\leq 10 \text{ KOhm}$) van bene
- **Highlights**
 - HW
 - led - polarità - anodo e catodo
 - digital I/O - output
 - Wiring API
 - **pinMode**, **digitalWrite**
 - dimensionamento resistenza, date le correnti
 - loop di controllo

STEP #1 - “HELLO BLINKING WORLD”



LED

- LED (Light Emitting Diode) - diodo a emissione di luce
 - dispositivo optoelettronico che sfrutta la capacità di alcuni materiali semiconduttori di produrre fotoni attraverso un fenomeno di emissione spontanea
 - con polarità
- A seconda del materiale utilizzato, i LED producono colori diversi
 - esempio:
 - arseniuro di gallio e alluminio (AlGaAs) - rosso ed infrarosso
 - fosfuro arseniuro di gallio (GaAIP) - verde
- La tensione di funzionamento dipendono dal colore:
 - esempi:
 - colore verde 2,0
 - colore infrarosso 1,3
 - colore rosso 1,8
- Corrente assorbita: 10..20 mA

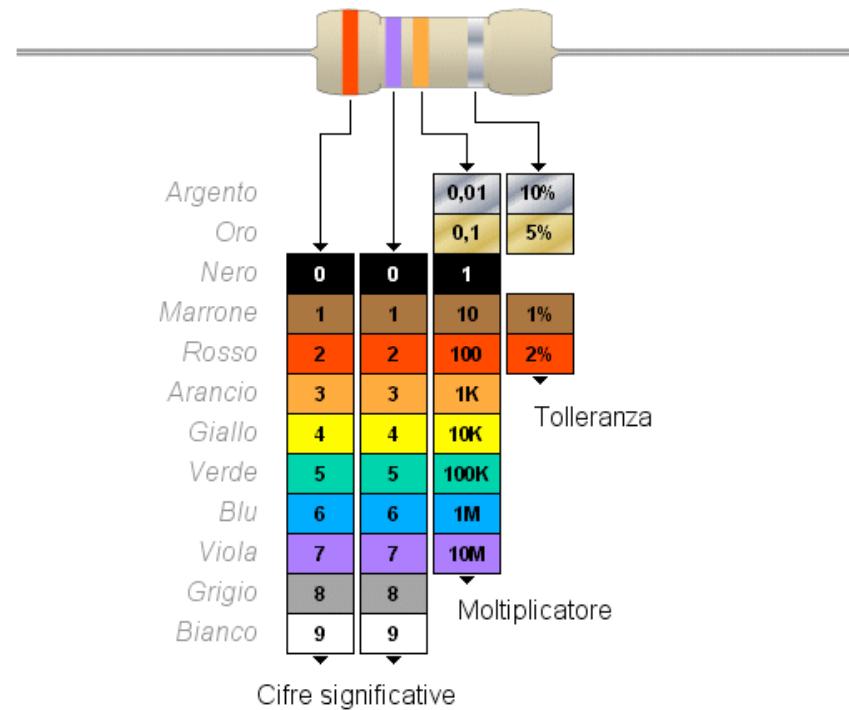


CALCOLO VALORE RESISTENZA

- E' opportuno inserire una resistenza nel circuito per limitare la corrente che fluisce nel LED, secondo le sue specifiche
 - <https://learn.adafruit.com/all-about-leds/the-led-datasheet>
- Dal suo datasheet, sappiamo che il LED determina una caduta di potenziale (chiamata Forward Voltage) di circa 2 Volt (tra 1.8 e 2.5 max)
- Quindi dei 5 Volt dell'alimentazione, rimangono 3 Volt (3 - 2) da far assorbire alla nostra resistenza
- Sempre dai datasheet del LED, vediamo che il valore di corrente opportuno per il funzionamento del LED (DC Forward Current) è 20 mA (30 mA max)
 - questo valore di corrente è compatibile con la quantità di corrente che può erogare Arduino da un pin di output (pari a 40 mA)
- Quindi, per la Legge di Ohm, il valore di resistenza da usare deve essere non inferiore a: $R = V / I = 3 / 0.02 = 150 \text{ Ohm}$
 - maggiore è il valore, più fioca è la luce (poiché minore è la corrente)
 - la resistenza disponibile sul mercato più vicina a 150 Ohm è quella da 220 Ohm
- Il circuito funziona bene anche con resistenza fino ad 1 KOhm

RESISTORI 220 OHM

- Seguendo lo schema delle bande colorate, nel caso di resistori assiali a 4 bande, il resistore di 220 Ohm è data dalla sequenza rosso-rosso-marrone
 - 1° rosso => 2, 2° rosso => 2 - quindi il valore di base è 22
 - 3° marrone => moltiplicatore*10 - quindi $22 \times 10 = 220$



STEP #1 - PROGRAMMA

blinking
led

```
#define LED_PIN 13

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

void loop() {
    digitalWrite(LED_PIN, HIGH);
    delay(1000);
    digitalWrite(LED_PIN, LOW);
    delay(1000);
}
```

OUTPUT SU SERIALE

- Possiamo sfruttare il collegamento seriale con il PC per inviare e ricevere byte / messaggi
 - debugging e per monitoraggio
 - comunicazione fra sistemi
- Libreria Serial su Wiring
 - <http://arduino.cc/en/reference/serial>
 - classe **Serial**
 - fra i metodi (statici):
 - begin - per inizializzare la seriale e specificare dati relativi al protocollo di comunicazione (boud rate in particolare)
 - print/println - per inviare stringhe
 - read - per leggere byte

SERIAL MONITOR

- Su Arduino IDE mediante il *serial monitor* è possibile:
 - visualizzare i messaggi inviati dalla seriale
 - inviare messaggi sulla seriale
- Le medesime funzionalità possono essere realizzate da un qualsiasi programma su PC che operi sulla seriale opportunamente
 - ne svilupperemo alcuni in Java nei prossimi moduli

ESEMPIO BLIKING CON OUTPUT SU SERIALE

```
#define LED_PIN 13

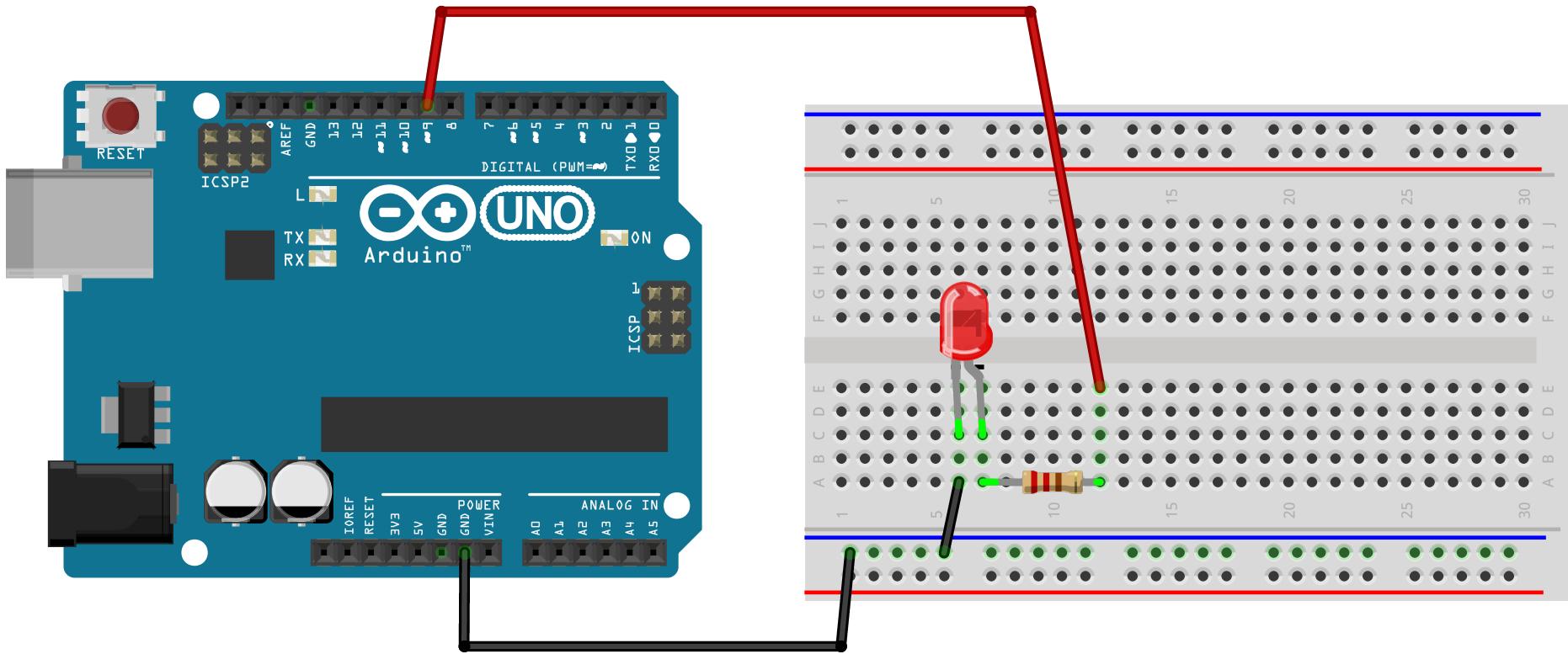
void setup() {
    pinMode(LED_PIN, OUTPUT);
    Serial.begin(9600);
}

void loop() {
    digitalWrite(LED_PIN, HIGH);
    Serial.println("ON");
    delay(1000);
    digitalWrite(LED_PIN, LOW);
    Serial.println("OFF");
    delay(1000);
}
```

STEP #2 - “FADING”

- Descrizione
 - sistema con led che varia periodicamente intensità da minima a massima
- HW
 - led, resistenza 220 Ω
- **Highlights**
 - output analogico e **PWM**
 - pin contrassegnati da ~
 - 3,5,6,9,10,11
 - Wiring API
 - **analogWrite**

STEP #2 - SCHEMA



fritzing

STEP #2 - PROGRAMMA

```
// deve essere uno che supporta PWM
#define LED_PIN 9

int brightness;
int fadeAmount;
int currIntensity;

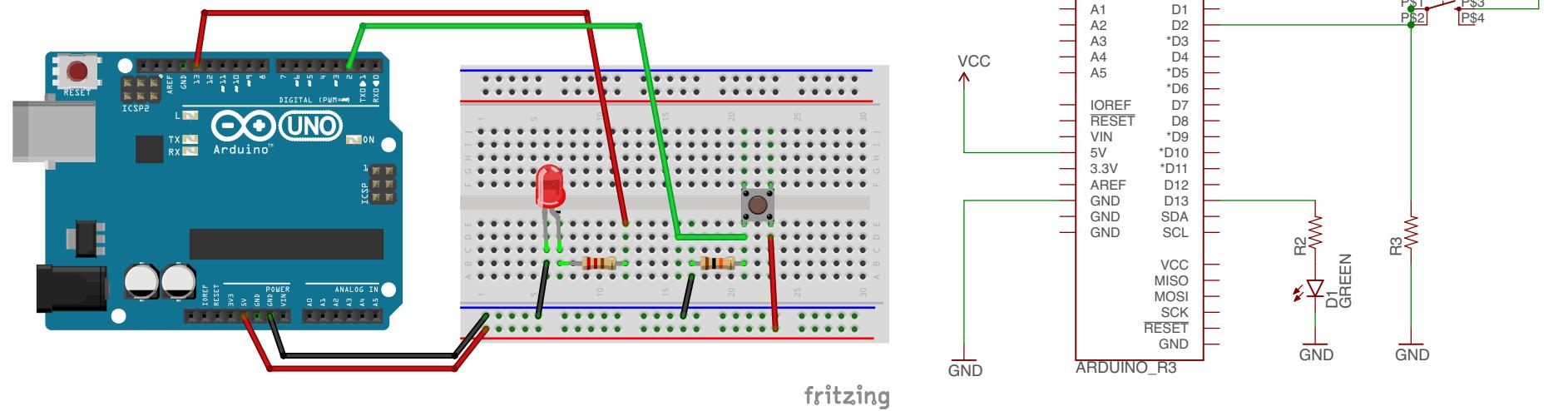
void setup(){
    currIntensity = 0;
    fadeAmount = 5;
    pinMode(LED_PIN, OUTPUT);
}

void loop(){
    analogWrite(LED_PIN, currIntensity);
    currIntensity = currIntensity + fadeAmount;
    if (currIntensity == 0 || currIntensity == 255) {
        fadeAmount = -fadeAmount ;
    }
    delay(20);
}
```

STEP #3 - “BUTTON-LED”

- Descrizione
 - sistema caratterizzato da un pulsante e un led. Il led deve essere acceso quando il pulsante è premuto e spendo viceversa.
- HW
 - led, resistore 220 Ω, pulsante, resistore 10 KΩ
- Highlights
 - digital I/O - input e output
 - Wiring API:
 - **digitalRead**
 - resistenze di pull-down
- Problemi
 - *bouncing buttons*

STEP #3 - SCHEMA



RESISTENZA PULL-DOWN

- La resistenza di pull-down di $10\text{ K}\Omega$ è inserita per fare in modo che quando il pulsante non sia premuto, il valore del pin di input sia pilotato al valore LOW
 - si chiama pull-down in questo caso perché porta il segnale al valore LOW, essendo collegato a GND
 - quelle di pull-up invece sono collegate a VCC, “tirando” la tensione a valore HIGH
- Le resistenze di pull-up e pull-down hanno tipicamente valore molto alto (es: $10\text{ K}\Omega$)
 - quando il pulsante è premuto, vogliamo che la corrente che fluisce nel ramo con la resistenza sia minima

STEP #3 - PROGRAMMA

```
#define BUTTON_PIN 2
#define LED_PIN 13

void setup() {
    pinMode(LED_PIN, OUTPUT);
    pinMode(BUTTON_PIN, INPUT);
    Serial.begin(9600);
}

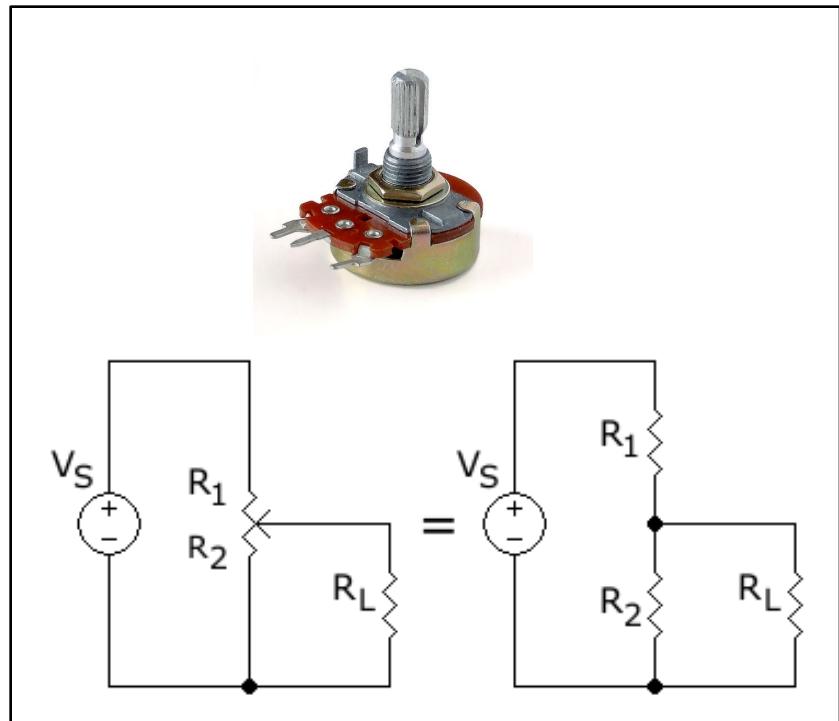
void loop() {
    int buttonState = digitalRead(BUTTON_PIN);
    if (buttonState == HIGH) {
        digitalWrite(LED_PIN, HIGH);
    } else {
        digitalWrite(LED_PIN, LOW);
    }
}
```

STEP #3 - POLLING

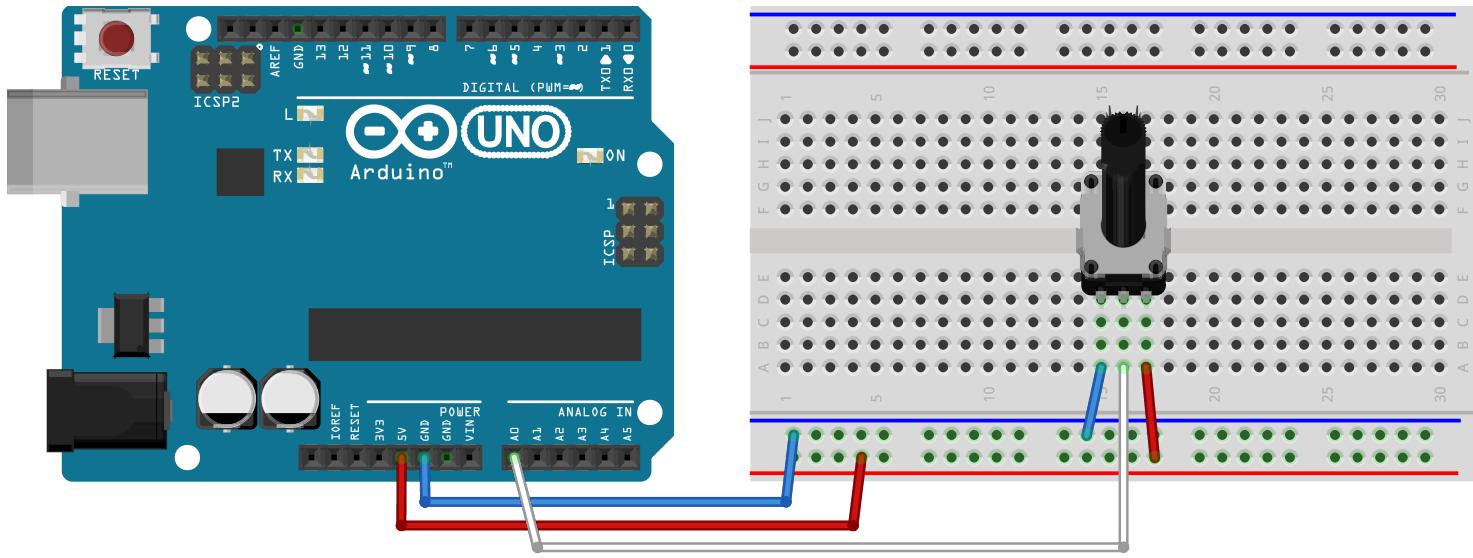
- L'approccio utilizzato per rilevare la pressione del pulsante è detto a *polling*
 - *lettura periodica del valore*
- Problema
 - se il tasto è premuto e rilasciato prima che possa essere letto, l'evento di pressione non viene rilevato...

STEP #4 - “TUNE”

- Descrizione
 - sistema caratterizzato da un potenziometro. Deve essere visualizzato in uscita (ovvero via seriale) il valore corrente del potenziometro, solo quando varia
- HW
 - un potenziometro
- Highlights
 - lettura pin analogici
 - Wiring API
 - **analogRead**



STEP #4 - SCHEMA



fritzing

STEP #4 - PROGRAMMA

```
#define POT_PIN A0

int current;

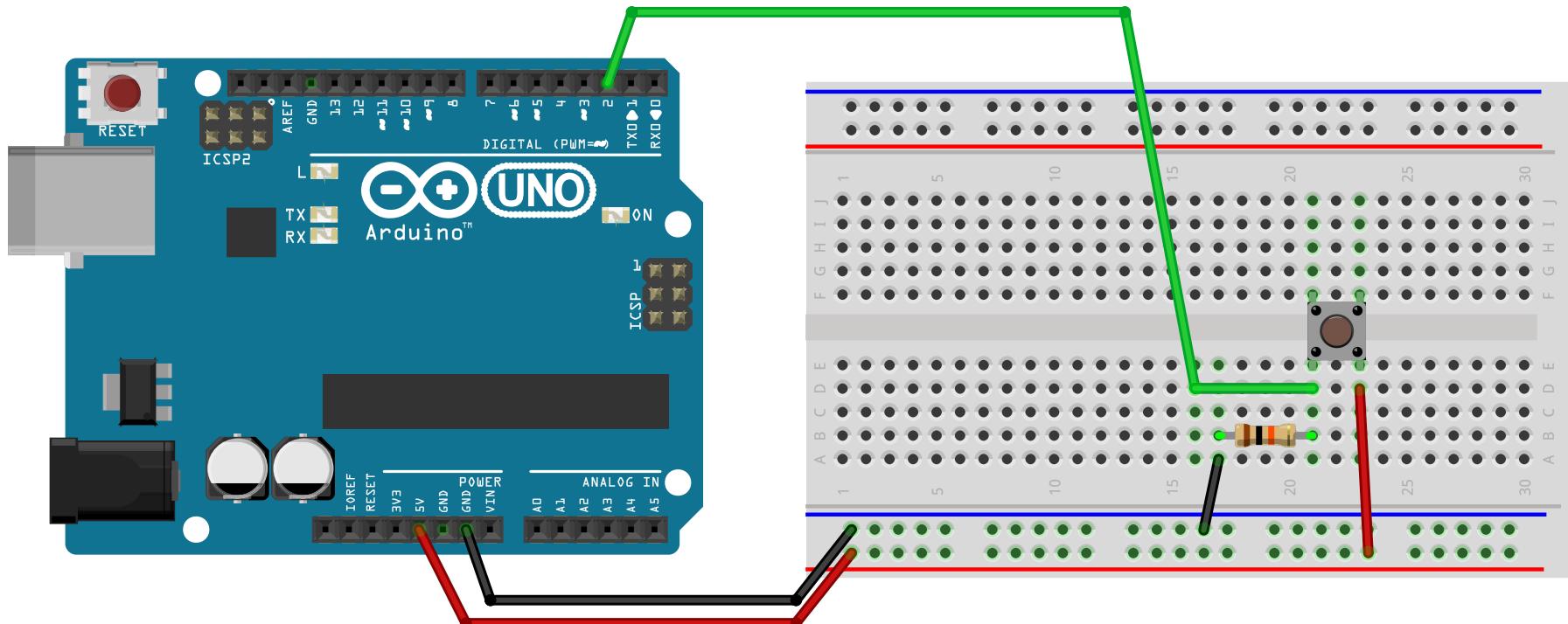
void setup() {
    Serial.begin(9600);
}

void loop() {
    int newValue = analogRead(POT_PIN);
    if (newValue != current){
        current = newValue;
        Serial.println(current);
    }
}
```

STEP #5 - “COUNTING”

- Descrizione
 - sistema che visualizza un conteggio, incrementabile mediante la pressione di un pulsante
- HW
 - un pulsante, un resistore da $10\text{ K}\Omega$
- **Highlights**
 - interruzioni
- Problemi
 - atomicità, corse critiche

STEP #5 - SCHEMA



fritzing

STEP #5 - PROGRAMMA

```
#define BUTTON_PIN 2
volatile int count = 0;
int prev = 0;

void setup(){
    Serial.begin(9600);
    attachInterrupt(digitalPinToInterrupt(BUTTON_PIN), inc, RISING);
}

void loop(){
    noInterrupts();
    int current = count;
    interrupts();
    if (current != prev){
        Serial.println(current);
        prev = current;
    }
}

void inc(){
    count++;
}
```

SEZIONI ATOMICHE/CRITICHE

- Nel loop principale è necessario eseguire l'assegnamento della variabile in sezione critica, per evitare corse critiche dovute all'eventuale accesso concorrente da parte dell'interrupt handler
 - nota: concorrente non è significa parallelo: avviene concorrentemente dal momento che il flusso nel loop viene interrotto..
- Realizzazione sezioni critiche su microcontrollori mediante la disabilitazione interruzioni
 - procedure in Wiring
 - **noInterrupts()/interrupts()**
- Ad interruzioni disabilitate il micro non reagisce => perdita eventi
 - il periodo di tempo nel quale le interruzioni sono disabilitate deve essere il più piccolo possibile

PROBLEMA: BOUNCING

- Il pulsanti come quello usato sono soggetti al fenomeno fisico chiamato **bouncing**, ovvero, quando premuti, internamente, a livello meccanico, possono “rimbalzare” prima di stabilizzarsi (entro qualche millisecondo) al valore aperto o chiuso
- Questi rimbalzi possono quindi generare un treno di interruzioni (poiché il valore passa da LOW a HIGH più volte) e l'effetto netto è che il contatore viene incrementato più volte ad una sola pressione
- Soluzioni al problema
 - hardware
 - pulsanti senza bouncing
 - software
 - si può risolvere il problema ignorando eventuali impulsi che arrivano nell'arco di un certo numero di milli-secondi (esempio 20-30 millisecondi) dopo aver rilevato il primo...

NOTE ULTERIORI

- Non tutti gli incrementi vengono stampati
 - ciò è dovuto al fatto che la stampa avviene in modo asincrono rispetto all'incremento e possono esserci più interruzioni/più incrementi fra una stampa e l'altra

STEP #6 - EVENT-DRIVEN BLINKING

- **Descrizione**
 - blinking di un led mediante interruzioni del timer
- HW
 - led, resistenza 220 Ohm
- Highlights
 - timer, libreria Timer1
 - da installare, seguendo la procedura:
<http://playground.arduino.cc/Code/Timer1>

STEP #6 - PROGRAMMA

```
#include "TimerOne.h"

#define LED_PIN 13

boolean flagState = false;

void blinky(){
    if (!flagState){
        digitalWrite(LED_PIN, HIGH);
    } else {
        digitalWrite(LED_PIN, LOW);
    }
    flagState = !flagState;
}

void setup()
{
    pinMode(LED_PIN,OUTPUT);
    /* set period timer 1000000 usec = 1 sec */
    Timer1.initialize(1000000);
    Timer1.attachInterrupt(blinky);
}

void loop(){}
}
```

STEP #7 - “SERIAL ECHO”

- Descrizione
 - leggere messaggi testuali inviati dalla seriale e rimandarli alla seriale stessa
- **Highlights**
 - libreria Serial

STEP #7 - PROGRAMMA

```
void setup()
{
    Serial.begin(9600);
}

void loop()
{
    if (Serial.available()){
        char data = Serial.read();
        Serial.print(data);
    }
}
```

STEP #8 - POWER DOWN

- **Descrizione**
 - test della modalità di risparmio energetico
- HW
 - un pulsante tattile, un resistore da $10\text{ K}\Omega$
- **Highlights**
 - eventi che permetto di uscire dalla modalità di sleep
 - tempo impiegato a riprendere il normale funzionamento

STEP #8 - PROGRAMMA

```
#include <avr/sleep.h>

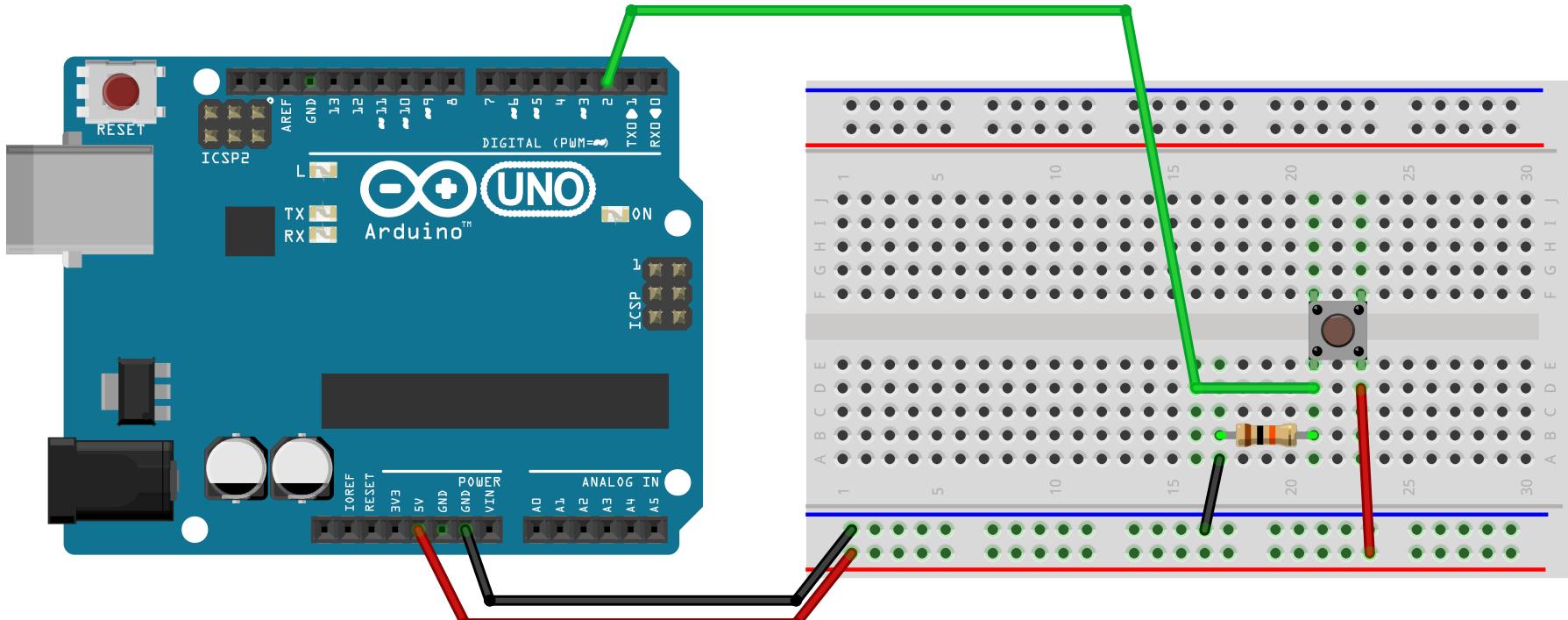
void wakeUp(){}

void setup(){
    Serial.begin(9600);
    pinMode(2, INPUT);
    attachInterrupt(digitalPinToInterrupt(2), wakeUp, RISING);
}

void loop(){
    Serial.println("GOING IN POWER DOWN IN 1s ...");
    Serial.flush();
    delay(1000);

    set_sleep_mode(SLEEP_MODE_PWR_DOWN);
    sleep_enable();
    sleep_mode();
    /** The program will continue from here. ***/
    Serial.println("WAKE UP");
    /* First thing to do is disable sleep. */
    sleep_disable();
}
```

STEP #8 - SCHEMA



fritzing

STEP #8 - POWER DOWN + TIMER

- **Descrizione**
 - test della modalità di risparmio energetico con uso di un timer per riprendere il funzionamento dopo un tempo prefissato
- **Highlights**
 - disattivazione selettiva dei componenti non necessari
 - approccio utile nella realizzazione degli scheduler (futuri moduli)

STEP #8 - PROGRAMMA

```
#include <avr/sleep.h>
#include <avr/power.h>
#include "Timer.h"
Timer* timer;

void setup(){
    Serial.begin(9600);
    timer = new Timer(); /* timer 1 */
    timer->setupPeriod(5000);
}

void sleep(void)
{
    set_sleep_mode(SLEEP_MODE_IDLE);
    sleep_enable();
    power_adc_disable();
    power_spi_disable();
    power_timer0_disable(); // only timer 1
    power_timer2_disable(); // on
    power_twi_disable();
    sleep_mode();
    /* back */
    sleep_disable();
    power_all_enable();
}
```

```
void loop(){
    Serial.println("GOING IN SLEEP");
    Serial.flush();
    sleep();
    Serial.println("WOKE UP");
}
```

Sistemi Embedded e IoT

Ingegneria e Scienze Informatiche - UNIBO

a.a 2019/2020

Docente: Prof. Alessandro Ricci

[modulo-1.3]

SENSORI E ATTUATORI - PANORAMICA

OBIETTIVO

- Questo modulo fornisce una introduzione ai principali concetti relativi ai sensori e attuatori, e una panoramica sui tipi di sensori e attuatori che è possibile utilizzare nei progetti di sistemi embedded del corso

SOMMARIO

- Sensori e attuatori: introduzione
- Sensori e dispositivi di input - tipologie
- Attuatori e dispositivi di output - tipologie
- Dispositivi di pilotaggio di circuiti esterni
- Quando i pin non bastano: uso di Shift-Registers

SENSORI E ATTUATORI

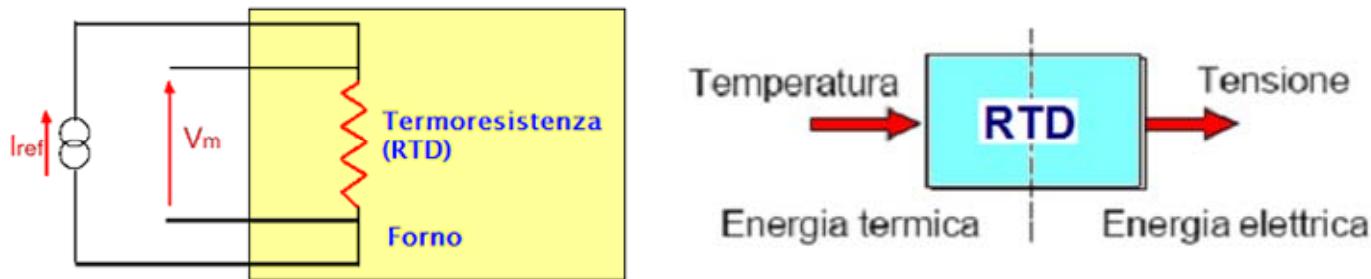
- Un sistema embedded interagisce con l'ambiente in cui è situato mediante sensori e attuatori
- **Sensori**
 - dispositivi **trasduttori** che permettono di **misurare** una certo fenomeno fisico (come la temperatura, radiazioni, umidità, etc) or rilevare e quantificare una concentrazione chimica (es: il fumo)
 - fornisce una rappresentazione misurabile di un fenomeno su una certa specifica scala o intervallo
 - esempio: una tensione in volt
 - possono essere **analogici** (se la grandezza elettrica prodotta in uscita - tensione o corrente - varia con continuità in risposta alla variazione della grandezza misurata) o **digitali** (insieme discreto di valori)
 - nel caso di sensori analogici, nel micro-controllore è tipicamente incluso un convertitore analogico digitale (ADC)
- **Attuatori**
 - dispositivi che producono un qualche effetto misurabile sull'ambiente
 - analogici o digitali (equipaggiati di DAC, convertitore Digitale Analogico)

GRANDEZZE FISICHE E SEGNALI

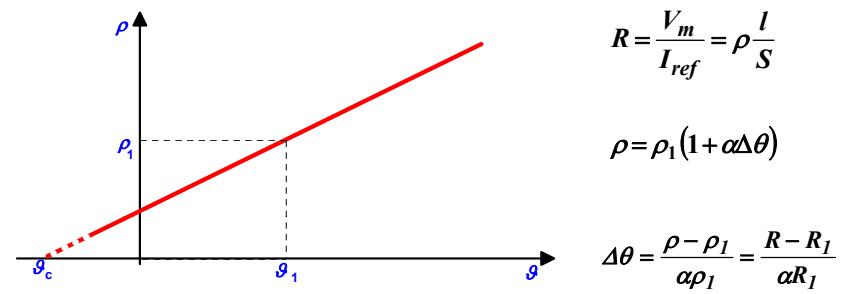
- Le **grandezze fisiche** oggetto di misura da parte dei trasduttori possono essere classificate in:
 - grandezze **continue**
 - valori continui all'interno di un certo intervallo (es: temperatura di un ambiente, velocità di rotazione di un motore, etc)
 - grandezze **discrete**
 - assumono un insieme discreto di valori (es. verso di rotazione di un motore, numero di pezzi lavorati al minuto, ecc....).
- Le *informazioni* associate alle grandezze fisiche sono dette **segnali**
 - le grandezze continue sono descritte da **segnali analogici**
 - le grandezze discrete sono descritte da:
 - **segnali logici**, nel caso si abbiano due valori ammissibili
 - **segnali codificati**, se il numero di valori ammissibili è superiore a due

PRINCIPIO DI FUNZIONAMENTO DI UN SENSORE

- Basato su una legge fisica nota che regola la relazione grandezza fisica da misurare e una grandezza elettrica d'uscita
- Esempio: termoresistenza (sensore di temperatura)



- La resistività cambia al variare della temperatura
- Per poter acquisire il segnale occorre fornire una corrente (o una tensione) e misurare poi la tensione (corrente) generata.



<http://ww2.unime.it/ingegneria/new/materiale/sensori2.pdf>

CLASSIFICAZIONE SENSORI

- **sensori analogici**
 - forniscono un segnale elettrico continuo a risoluzione infinita

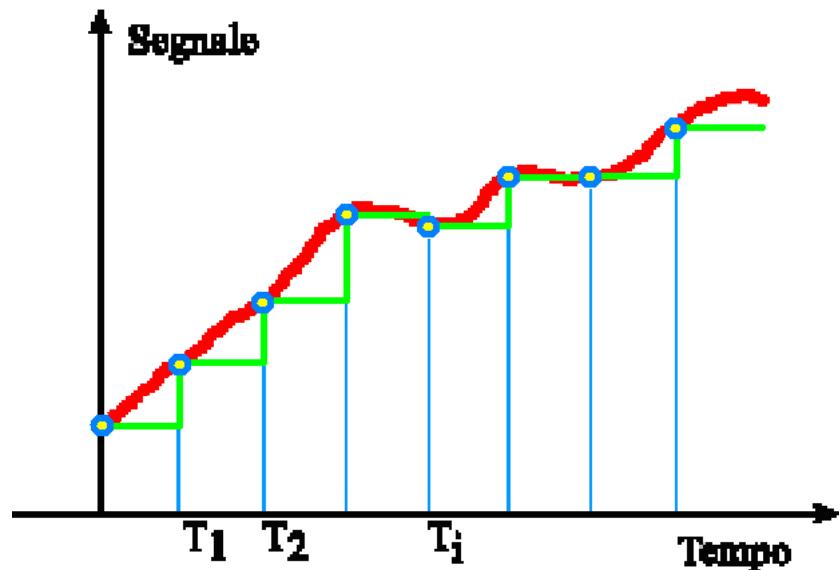


- **sensori digitali**
 - forniscono una informazione di tipo numerico con risoluzione finita
 - **sensori logici**
 - hanno una uscita di tipo booleano
 - **sensori codificati:**
 - hanno un'uscita numerica codificata in una stringa di bits.



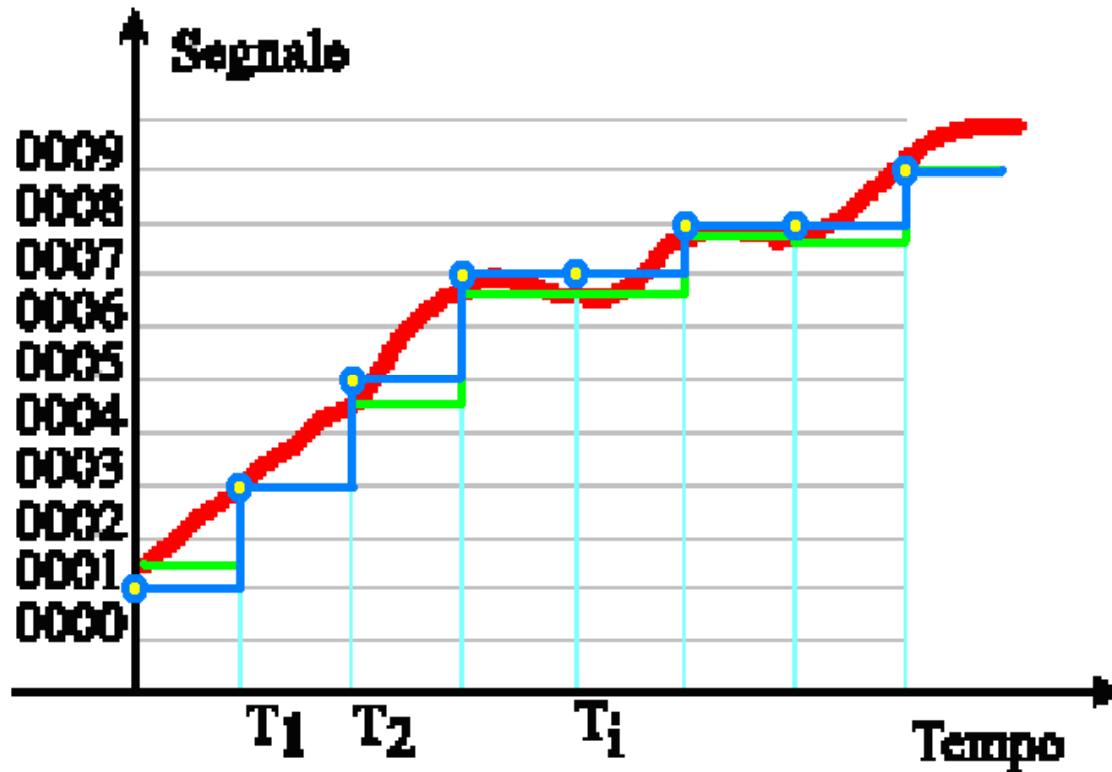
CAMPIONAMENTO DEL SEGNALE

- Un segnale analogico prodotto da un sensore analogico per poter essere elaborato dal calcolatore deve essere *campionato* mediante un sistema di conversione analogico/digitale nel caso di sensori analogici
 - dà origine ad un *segnalet digitale*
- **Campionamento del segnale**
 - acquisizione di campioni del segnale analogico ad istanti discreti di tempo



QUANTIZZAZIONE DEL SEGNALE

- Operazione di approssimazione del valore campionato al più vicino valore digitale



FUNZIONE DEI SENSORI: MISURARE

- **Misura** di una grandezza fisica = confronto di due quantità appartenenti a grandezza della stessa specie (*omogenee*)
 - stabilisce in che rapporto una quantità incognita stia rispetto ad un'altra, che opera come *riferimento*, esprime il risultato della misura
 - L'operazione che porta a questo risultato è detta *misurazione*
- Il sistema internazionale di unità di misura **SI** (in francese: Système international d'unités), è il più diffuso sistema di unità di misura.
 - unità, terminologia e raccomandazioni del SI sono fissate dalla "Conferenza generale dei pesi e delle misure"
 - è basato su sette *grandezze fisiche fondamentali* e sulle corrispondenti unità di misura con le quali vengono definite le grandezze fisiche derivate e le corrispondenti unità di misura.
 - è un "sistema coerente" in quanto le sue grandezze fisiche derivate si ricavano come prodotto e rapporto di grandezze fisiche fondamentali

UNITA' FONDAMENTALI NEL SI

Grandezza fisica	Simbolo della grandezza fisica	Nome dell'unità SI	Simbolo dell'unità SI
Intensità di corrente elettrica	I, i	ampere	A
Intensità luminosa	I_V	candela	cd
Lunghezza	l	metro	m
Massa	m	chilogrammo	kg
Quantità di sostanza	n	mole	mol
Temperatura termodinamica	T	kelvin	K
Intervallo di tempo	t	secondo	s

UNITA' FONDAMENTALI CON RIF.

Unit name	Unit symbol	Quantity name	Definition (incomplete) ^[n 1]	Dimension symbol
metre	m	length	<ul style="list-style-type: none"> Original (1793): $\frac{1}{10\ 000\ 000}$ of the meridian through Paris between the North Pole and the Equator.^{FG} Interim (1960): 1 650 763.73 wavelengths in a vacuum of the radiation corresponding to the transition between the $2p^{10}$ and $5d^5$ quantum levels of the krypton-86 atom. Current (1983): The distance travelled by light in a vacuum in $\frac{1}{299\ 792\ 458}$ second. 	L
kilogram ^[n 2]	kg	mass	<ul style="list-style-type: none"> Original (1793): The grave was defined as being the weight [mass] of one cubic decimetre of pure water at its freezing point.^{FG} Current (1889): The mass of the International Prototype Kilogram (Le Grand K).^[38] 	M
second	s	time	<ul style="list-style-type: none"> Original (Medieval): $\frac{1}{86\ 400}$ of a day. Interim (1956): $\frac{1}{31\ 556\ 925.9747}$ of the tropical year for 1900 January 0 at 12 hours ephemeris time. Current (1967): The duration of 9 192 631 770 periods of the radiation corresponding to the transition between the two hyperfine levels of the ground state of the caesium-133 atom. 	T
ampere	A	electric current	<ul style="list-style-type: none"> Original (1881): A tenth of the electromagnetic CGS unit of current. The [CGS] electromagnetic unit of current is that current, flowing in an arc 1 cm long of a circle 1 cm in radius, that creates a field of one oersted at the centre.^[39] IEC Current (1946): The constant current which, if maintained in two straight parallel conductors of infinite length, of negligible circular cross-section, and placed 1 m apart in vacuum, would produce between these conductors a force equal to 2×10^{-7} newtons per metre of length. 	I
kelvin	K	thermodynamic temperature	<ul style="list-style-type: none"> Original (1743): The centigrade scale is obtained by assigning 0 °C to the freezing point of water and 100 °C to the boiling point of water. Interim (1954): The triple point of water (0.01 °C) defined to be exactly 273.16 K.^[n 3] Current (1967): $\frac{1}{273.16}$ of the thermodynamic temperature of the triple point of water 	Θ
mole	mol	amount of substance	<ul style="list-style-type: none"> Original (1900): The molecular weight of a substance in mass grams.^{ICAW} Current (1967): The amount of substance of a system which contains as many elementary entities as there are atoms in 0.012 kilogram of carbon-12.^[n 4] 	N
candela	cd	luminous intensity	<ul style="list-style-type: none"> Original (1946): The value of the new candle is such that the brightness of the full radiator at the temperature of solidification of platinum is 60 new candles per square centimetre. Current (1979): The luminous intensity, in a given direction, of a source that emits monochromatic radiation of frequency 5.4×10^{14} hertz and that has a radiant intensity in that direction of $\frac{1}{683}$ watt per steradian. 	J

UNITA' DERIVATE

Grandezza fisica	Simbolo della grandezza fisica	Nome dell'unità SI	Simbolo dell'unità SI	Equivalenza in termini di unità fondamentali SI
<i>Nomi e simboli speciali</i>				
frequenza	f, v	hertz	Hz	s^{-1}
forza	F	newton	N	$kg \cdot m \cdot s^{-2}$
pressione	p	pascal	Pa	$N \cdot m^{-2}$
energia, lavoro, calore, entalpia	$E, W/L, Q, H$	joule	J	$N \cdot m$
potenza	P	watt	W	$J \cdot s^{-1}$
viscosità dinamica	μ, η	poiseuille	Pl	$Pa \cdot s$
carica elettrica	q	coulomb	C	$A \cdot s$
potenziale elettrico, forza elettromotrice, tensione elettrica	V, fem	volt	V	$J \cdot C^{-1}$
resistenza elettrica	R	ohm	Ω	$V \cdot A^{-1}$
conduttanza elettrica	G	siemens	S	$A \cdot V^{-1}$
capacità elettrica	C	farad	F	$C \cdot V^{-1}$
densità flusso magnetico	B	tesla	T	$V \cdot s \cdot m^{-2}$
flusso magnetico	$\Phi(B)$	weber	Wb	$V \cdot s$
induttanza	L	henry	H	$V \cdot s \cdot A^{-1}$
temperatura	T	grado Celsius	$^{\circ}C$	K ^[7]
angolo piano ^[8]	α, ϕ, θ	radiante	rad	1
angolo solido ^[8]	Ω	steradiane	sr	1
flusso luminoso	$\Phi(l)$	lumen	lm	$cd \cdot sr$
illuminamento	E_l	lux	lx	$cd \cdot sr \cdot m^{-2}$
potere diottico	D_o	diottria	D	m^{-1}
attività di un radionuclide ^[9]	A_R	becquerel	Bq	s^{-1}
dose assorbita	D	gray	Gy	$J \cdot kg^{-1}$
dose equivalente, dose efficace	H, E_H	sievert	Sv	$J \cdot kg^{-1}$
attività catalitica		katal	kat	$mol \cdot s^{-1}$
<i>Altre grandezze fisiche</i>				
area	A	metro quadro	m^2	m^2
volume	V	metro cubo	m^3	m^3
velocità	v	metro al secondo	m/s	$m \cdot s^{-1}$
accelerazione	a		m/s^2	$m \cdot s^{-2}$
velocità angolare	ω			$rad \cdot s^{-1}$
accelerazione angolare	a, ω			$rad \cdot s^{-2}$
densità	ρ, d	chilogrammo al metro cubo	kg/m^3	$kg \cdot m^{-3}$
molarità SI ^[10]	M			$mol \cdot dm^{-3}$
volume molare	V_m			$m^3 \cdot mol^{-1}$

PREFISSI NEL SI

10^n	Prefisso	Simbolo	Nome	Equivalente decimale
10^{24}	yotta	Y	Quadrilione	1 000 000 000 000 000 000 000 000
10^{21}	zetta	Z	Triliardo	1 000 000 000 000 000 000 000 000
10^{18}	exa	E	Trilione	1 000 000 000 000 000 000 000 000
10^{15}	peta	P	Biliardo	1 000 000 000 000 000 000 000 000
10^{12}	tera	T	Bilione	1 000 000 000 000 000 000
10^9	giga	G	Miliardo	1 000 000 000
10^6	mega	M	Milione	1 000 000
10^3	chilo	k	Mille	1 000
10^2	hecto	h	Cento	100
10^1	deca	da	Dieci	10
10^0			Uno	1
10^{-1}	deci	d	Decimo	0,1
10^{-2}	centi	c	Centesimo	0,01
10^{-3}	milli	m	Millesimo	0,001
10^{-6}	micro	μ	Milionesimo	0,000 001
10^{-9}	nano	n	Miliardesimo	0,000 000 001
10^{-12}	pico	p	Bilionesimo	0,000 000 000 001
10^{-15}	femto	f	Biliardesimo	0,000 000 000 000 001
10^{-18}	atto	a	Trilionesimo	0,000 000 000 000 000 001
10^{-21}	zepto	z	Triliardesimo	0,000 000 000 000 000 000 001
10^{-24}	yocto	y	Quadrilionesimo	0,000 000 000 000 000 000 000 001

INCERTEZZA DI MISURA E ERRORI

- Misure ripetute di uno stesso parametro *non forniscono lo stesso valore*
 - cause molteplici (legate allo strumento, all'ambiente)
- Per esprimere questa dispersione di valori si introduce il concetto di **incertezza di misura**
 - esprime la *dispersione dei valori*, reale o potenziale
 - può essere interpretata come il *dubbio riguardante il risultato*, ovvero la dispersione dei valori attribuibili al misurando
 - una grandezza fisica può essere determinata soltanto a un livello finito di incertezza (**errore**)
 - l'incertezza quantifica l'imprecisione della determinazione
- ***Il risultato di misura è costituito sempre da un numero e da un'incertezza, generalmente preceduta dal simbolo \pm e da una unità di misura***

CLASSIFICAZIONE ERRORI DI MISURA

- **Errori sistematici**
 - un errore è sistematico se, fissate le condizioni sperimentali, in grandezza e segno ha la stessa influenza sul risultato della misura
- **Errori accidentali (casuali)**
 - errori la cui influenza sulla misura può cambiare in grandezza e segno se si ripete la procedura di misurazione
 - le condizioni ambientali sono sorgenti di errori casuali se non vengono monitorate o se non si conosce la loro influenza sulla grandezza da misurare, diversamente possono essere sorgenti di errore sistematico
- **Errori grossolani**
 - errori che riguardano l'operatore o guasti dello strumento.

PRINCIPALI CARATTERISTICHE METROLOGICHE DI UN SENSORE

- **Caratteristiche statiche**
 - si riferiscono a condizioni di funzionamento in cui viene variata molto lentamente la variabile di ingresso del sensore, registrando la corrispondente variabile di uscita
 - *Accuratezza, Precisione, Linearità, Sensibilità, Risoluzione, Ripetibilità, Riproducibilità, Stabilità, ...*
- **Caratteristiche dinamiche**
 - se la variabile di ingresso varia velocemente l'uscita può evidenziare un'attenuazione rispetto alla caratteristica statica ed un ritardo.
 - la caratteristica dinamica del trasduttore pone un limite alla banda passante di tutto il sistema di controllo.
 - tempo di risposta (risposta in frequenza)

CARATTERISTICA STATICIA DI UN SENSORE

- La **caratteristica statica** di un trasduttore è definita da una funzione del tipo:

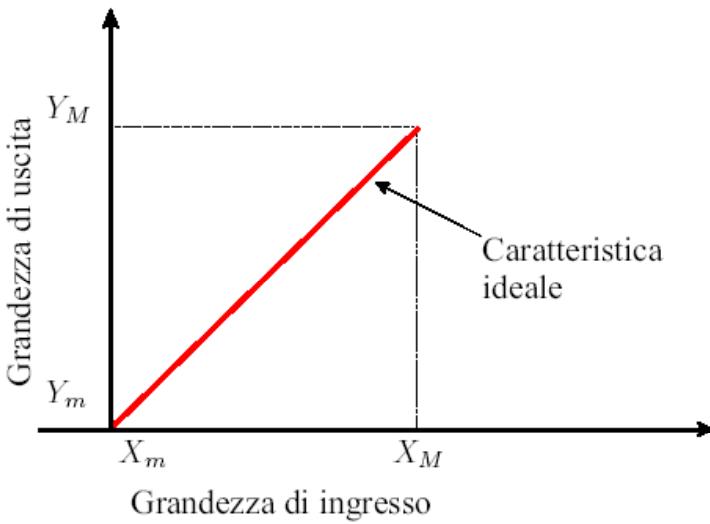
$$Y = f(X)$$

dove

- X rappresenta il segnale di ingresso e
- Y il segnale di uscita dal trasduttore.
- La caratteristica è definita su di un intervallo finito chiamato **campo di ingresso** avente estremi X_m e X_M ed ha valori sul **campo di uscita** (output range o span) con estremi Y_m e Y_M .
- Si definiscono
 - **Range d'ingresso:** $X_s = X_M - X_m$
 - **Range d'uscita:** $Y_s = Y_M - Y_m$

GUADAGNO DI UN SENSORE

- La caratteristica statica di un sensore deve avere idealmente un *andamento lineare*.
 - la costante di proporzionalità fra valori di ingresso e di uscita viene chiamata **guadagno (K)** del trasduttore



- I trasduttori commerciali hanno una **caratteristica statica reale** che si differenzia da quella ideale a causa di inevitabili imperfezioni costruttive
- La **qualità di un sensore** si misura in base a quanto la caratteristica reale si scosta da quella ideale

LINEARITÀ

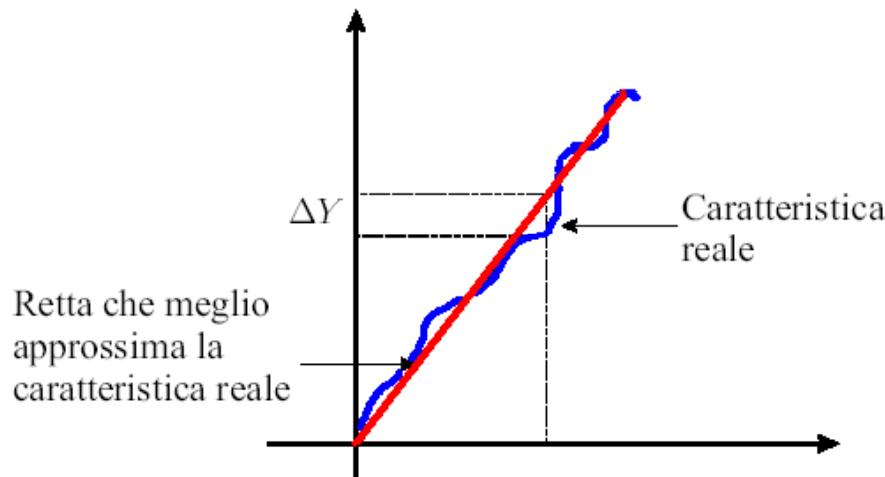
- La ***linearità*** di un trasduttore può essere definita in diversi modi, non del tutto equivalenti
- Per i trasduttori *lineari* la relazione tra la grandezza fisica misurata e il segnale in uscita è descrivibile attraverso una semplice relazione matematica:

$$Y = K X$$

dove K è il guadagno

ERRORI DI LINEARITÀ

- L'errore di linearità è la massima deviazione dell'uscita del trasduttore rispetto alla caratteristica lineare che approssima al meglio la caratteristica reale



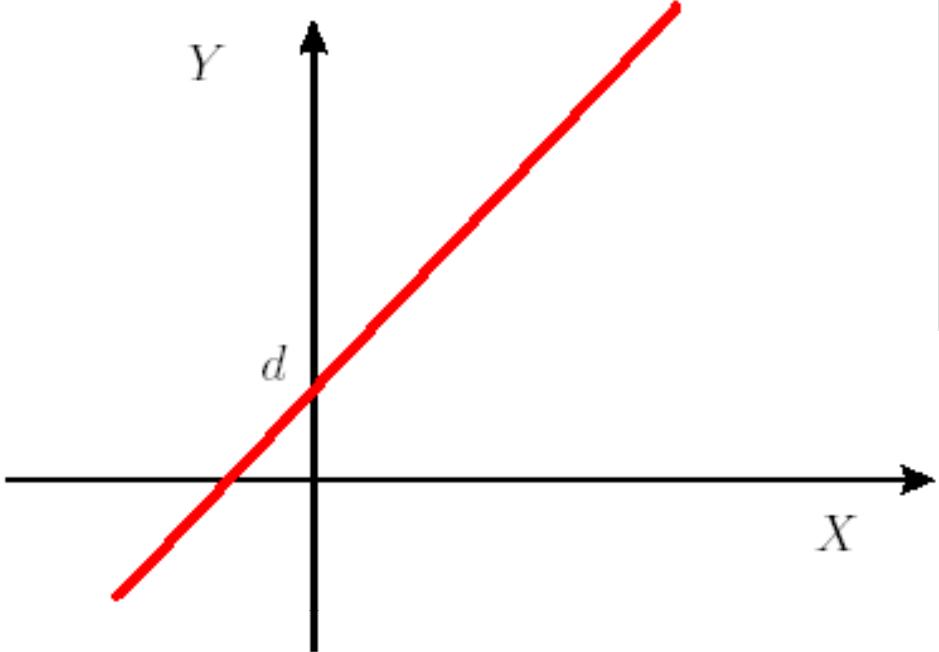
- La caratteristica lineare viene normalmente ottenuta secondo il metodo dei minimi quadrati
 - si cerca quella retta $r(X)$ che minimizzi la quantità: $J = \sum_{i=0}^N [r(X_i) - Y_i]^2$
- Se ΔY è il massimo scostamento dalla caratteristica lineare, l'errore percentuale di non linearità vale:

$$e_L \% = \frac{\Delta Y}{Y_M - Y_m} 100$$

ERRORE DI OFFSET

- L'errore di *offset* (o di fuori zero) è il valore d che assume l'uscita del trasduttore quando la grandezza da misurare è nulla

$$Y = f(X) = KX + d$$

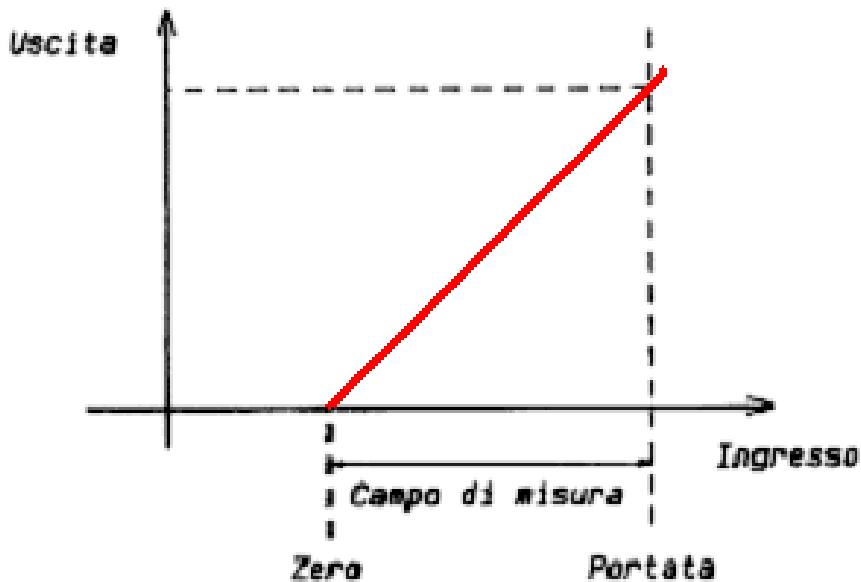


Per rendere lineare la caratteristica occorre eliminare il termine di “offset” d dalla caratteristica del trasduttore:

$$Y = f(X) - d$$

ERRORE DI SOGLIA

- L'errore di soglia corrisponde al più basso livello di segnale rilevabile dal sensore.
 - esso non sempre coincide con lo zero della grandezza da misurare



Per rendere lineare la caratteristica occorre eliminare il termine di “soglia” s dalla caratteristica del trasduttore:

$$Y = f(X - s)$$

ERRORE DI GUADAGNO

- L'errore di guadagno è la differenza tra il guadagno della caratteristica ideale del trasduttore (K) e il guadagno della retta (K_1) che approssima al meglio la caratteristica reale del trasduttore.
- L'errore di guadagno è solitamente espresso in percentuale:

$$e_G \% = \frac{|K_1 - K|}{K} 100$$

ERRORI DI QUANTIZZAZIONE

- L'operazione di campionamento non produce in via teorica un degrado dell'informazione associata al segnale
 - se si rispettano le condizioni del teorema del campionamento
- L'operazione di quantizzazione comporta inevitabilmente l'introduzione di un errore sul segnale acquisito

ACCURATEZZA

- L'accuratezza di un sensore è la misura di quanto il valore letto dal sensore si discosti dal valore corretto
 - spesso si fa riferimento, più che all'accuratezza, all'equivalente **incertezza della misura**
- Come si calcola
 - è *il massimo scostamento tra la misura (o “lettura”) fornita dal sensore e il “valore vero” del segnale misurato*

$$\text{Accuratezza in \% del fondo scala} \quad \epsilon_f = 100 \cdot \frac{X_m - X_v}{X_{FS}}$$

- X_v = valore vero del misurando
- X_m = valore misurato (nel caso peggiore)
- X_{FS} = valore di fondo scala

$$\text{Accuratezza in \% della misura} \quad \epsilon_a = 100 \cdot \frac{X_m - X_v}{X_v}$$

$$\text{Accuratezza assoluta} \quad \epsilon(X_v) = |X_m - X_v|$$

ESEMPIO

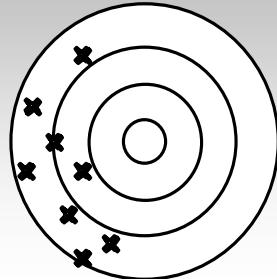
- Un sensore di pressione che misura valori del range 0-10 bar con inaccuratezza pari a $\pm 1.0\%$ rispetto al fondo scala.
- Allora il massimo errore di misura è $10 * 1.0\% = 10 * 0.01 = 0.1$ bar
 - Nota: siccome il valore massimo dell'errore di misura è relativo al valore fondo scala, la lettura di valori significativamente più piccoli (es: 2 bar) comporta che l'errore di misura è amplificato

PRECISIONE

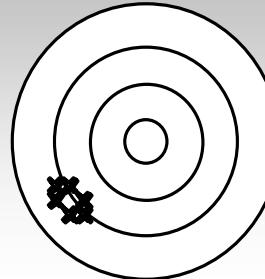
- La precisione descrive quanto un sensore sia soggetto o meno ad errori accidentali (*random errors*)
 - se eseguiamo un numero elevato di letture con un sensore che ha precisione elevata, il range di tali valori sarà piccolo
 - alta precisione non implica elevata accuratezza
- È legata alla *ripetibilità* o *riproducibilità* (repeatability)
 - esprime la riproducibilità di una misura, ossia esprime l'attitudine del sensore a fornire valori della grandezza in uscita poco diversi tra loro, a parità di segnale di ingresso (=stesso valore vero) e nelle stesse condizioni di lavoro
- Si calcola considerando la deviazione dei valori letti rispetto al valor medio

ACCURATEZZA VS. PRECISIONE

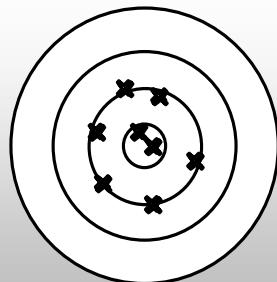
Not Accurate or Precise



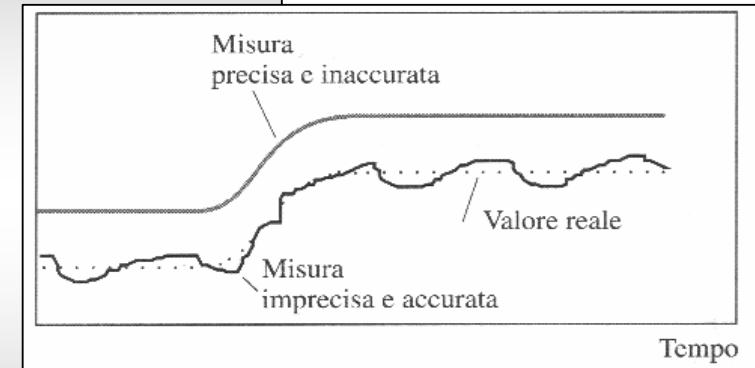
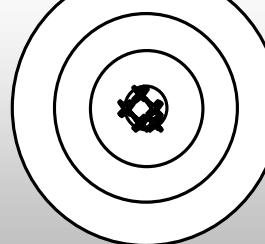
Precise and NOT Accurate



Accurate and NOT Precise



Precise and Accurate



Nota: i dati di accuratezza e precisione forniti dai costruttori sono relativi a specifiche condizioni di funzionamento (esempio: temperatura) e allo strumento ben tarato

ESEMPIO

- La larghezza di una stanza è stata misurata con un sensore ultrasuoni 10 volte riportando le seguenti misure: 5.381, 5.379, 5.378, 5.382, 5.380, 5.383, 5.379, 5.377, 5.380, 5.381
- La misura effettuata con un altro strumento di riferimento calibrato ha riportato il valore di 5.374m, che può essere assunto come valore corretto
- Calcolo precisione
 - valor medio delle misure: 5.380
 - massima deviazione sotto il valore: $5.377 - 5.380 = -0.003$
 - massima deviazione sopra il valore: $5.383 - 5.380 = +0.003$
> la precisione del sensore è esprimibile come ± 0.003 m
- Massima inaccuratezza della misura = differenza fra il valore corretto e misura dal valore più distante = $5.383 - 5.374 = +0.009$ m

CALIBRAZIONE E TARATURA

- Causa di mancanza di accuratezza => errori sistematici
 - bias o systematic errors
- **Calibrazione** (*calibration*) del sensore
 - aggiustamento dei parametri del sensore per farne corrispondere l'uscita a valori rilevati accuratamente con un altro strumento
 - **taratura**
 - misurazione della grandezza di uscita per valori noti della grandezza di ingresso al trasduttore stesso
- Cause di “staramento” nel tempo: usura, sporco, ...

ALTRI PARAMETRI STATICI

- **Sensibilità**
 - rapporto fra variazione della grandezza (segnale) di uscita e corrispondente variazione (segnale) d'ingresso
- **Risoluzione**
 - capacità dello strumento (sensore) di risolvere stati (livelli) diversi del misurando
- **Ripetibilità**
 - attitudine dello strumento a fornire per uno stesso misurando valori di lettura vicini tra loro, in letture consecutive eseguite in breve intervallo di tempo
- **Riproducibilità**
 - vicinanza di risultati ottenuti sullo stesso misurando in diverse specifiche condizioni di misura
- **Stabilità**
 - attitudine di uno strumento a fornire valori di lettura poco differenti tra loro in letture eseguite indipendentemente sullo stesso misurando in un intervallo di tempo definito

COMPORTAMENTO DINAMICO

- Ogni sensore ha la sua *dinamica*
 - la risposta di un sensore non è istantanea
 - è spesso trascurabile rispetto alle dinamiche del processo
- Quando in ingresso al trasduttore applichiamo una sollecitazione a *gradino* (cioè un gradino della grandezza da misurare) l'uscita (risposta) varierà fino a raggiungere, dopo un certo tempo, un nuovo valore.
- Si definisce:
 - **tempo di salita**
 - tempo impiegato per passare dal 10% al 90% del valore finale
 - **tempo di risposta**
 - tempo impiegato per raggiungere una percentuale prefissata del valore finale.

SENSORI - ALCUNE TIPOLOGIE

SENSORI: TIPOLOGIE

- Sensori di Prossimità
- Sensori di Movimento
- Trasduttori di Posizione Angolare
- Sensori di Accelerazione
- Sensori di Contatto
- Sensori Ottici
- Sensori Elettricità Magnetismo
- Sensori parametri ambientali
- Sensori parametri bio fisiologici
- Sensori Suono
- Sensori per localizzazione/posizionamento
- Sensori "Identità"

Grandezze misurabili	Trasduttori	Grandezze di uscita
Temperatura	Termocoppie Resistenze al platino(RTD), termistori (PTC, NTC) Trasduttori a semicondutture	Tensione Variazione di resistenza Corrente (tensione)
Forza- Pressione	Potenziometri, estensimetri (strain gauge) Trasduttori capacitivi Trasduttori piezoelettrici	Variazione di resistenza Variazione di capacità Tensione
Posizione	Potenziometri, estensimetri	Variazione di resistenza
Spostamento	Trasduttori capacitivi Trasformatori differenziali, Syncro Trasduttori induttivi Trasduttori ad effetto Hall Trasduttori ottici digitali (encoder incrementale) Fotodiodi, fototransistori	Variazione di capacità Tensione Variazione di induzione Tensione Numero di impulsi Corrente(tensione)
Velocità	Trasduttori piezoelettrici Dinamo tachimetrica Trasduttore ottico digitale (encoder assoluto)	Tensione Tensione Frequenza di impulsi
Intensità luminosa	Fotodiodi, fototransistori Fotoresistenze Celle fotovoltaiche	Corrente(tensione) Variazione di resistenza Tensione

ELENCO ADAFRUIT

- Elenco sensori Adafruit
 - <https://www.adafruit.com/category/35>

LIBRERIE WIRING E ARDUINO PLAYGROUND

- **Librerie Wiring**
 - librerie disponibili nel framework Wiring per l'interfacciamento e controllo di numerosi tipi di sensori e attuatori
 - <http://wiring.org.co/learning/libraries/>
 - Schemi di collegamento, sorgenti in Arduino
- **Arduino playground**
 - wiki che include numerosi esempi, tutorial e librerie su come usare specifici sensori e attuatori
 - <http://playground.arduino.cc>
 - <http://playground.arduino.cc/Main/LibraryList>

PROSSIMITÀ

- I sensori di prossimità sono dei sensori in grado di **rilevare la presenza** di oggetti nelle immediate vicinanze del "lato sensibile" del sensore stesso, senza che vi sia un effettivo contatto fisico.
 - la distanza entro cui questi sensori rilevano oggetti è definita *portata nominale* (o campo sensibile)
- Alcuni modelli dispongono di un sistema di regolazione per poter calibrare la distanza di rilevazione
- L'assenza di meccanismi d'attuazione meccanica, e di un contatto fisico tra sensore e oggetto, fa sì che questi sensori presentino un'affidabilità elevata.
- Segnale d'uscita
 - normalmente i sensori di prossimità rilevano solamente la presenza o l'assenza di un oggetto all'interno della loro portata nominale
 - conseguentemente il segnale elettrico d'uscita sarà di tipo on/off in quanto deve rappresentare solo gli stati assenza/presenza.
- I sensori di prossimità possono essere realizzati basandosi su diversi tipi di tecnologie
 - sensori induttivi, capacitivi, sensori magnetici, sensori ad ultrasuoni, sensori ottici

ESEMPIO: SENSORE ULTRASUONI

- Funzionano sul principio del Sonar:
 - emettono impulsi sonori ultrasonici, e rilevano un'eventuale eco di ritorno generata dalla presenza di un oggetto all'interno della portata nominale
- Svantaggi
 - costo e velocità di commutazione (bassa)
- Vantaggi
 - portate nominali elevate (fino a 10 m);
 - immunità ai disturbi elettromagnetici;
 - possono rilevare oggetti di qualsiasi materiale (eccetto materiali fonoassorbenti);
 - possono rilevare oggetti senza che questi siano stati preventivamente preparati.
- Una certa attenzione va però posta nella dimensione e nell'orientamento della superficie dell'oggetto che si rivolge al sensore, infatti una superficie troppo piccola o orientata malamente (non ortogonale alla direzione di lettura del sensore) può non assicurare la generazione di un'eco rilevabile.

ESEMPIO

- Sensore HC-SR04
 - datasheet e documentazione: https://docs.google.com/document/d/1Y-yZnNhMYy7rwhAgYL_pfa39RsB-x2qR4vP8saG73rE/edit

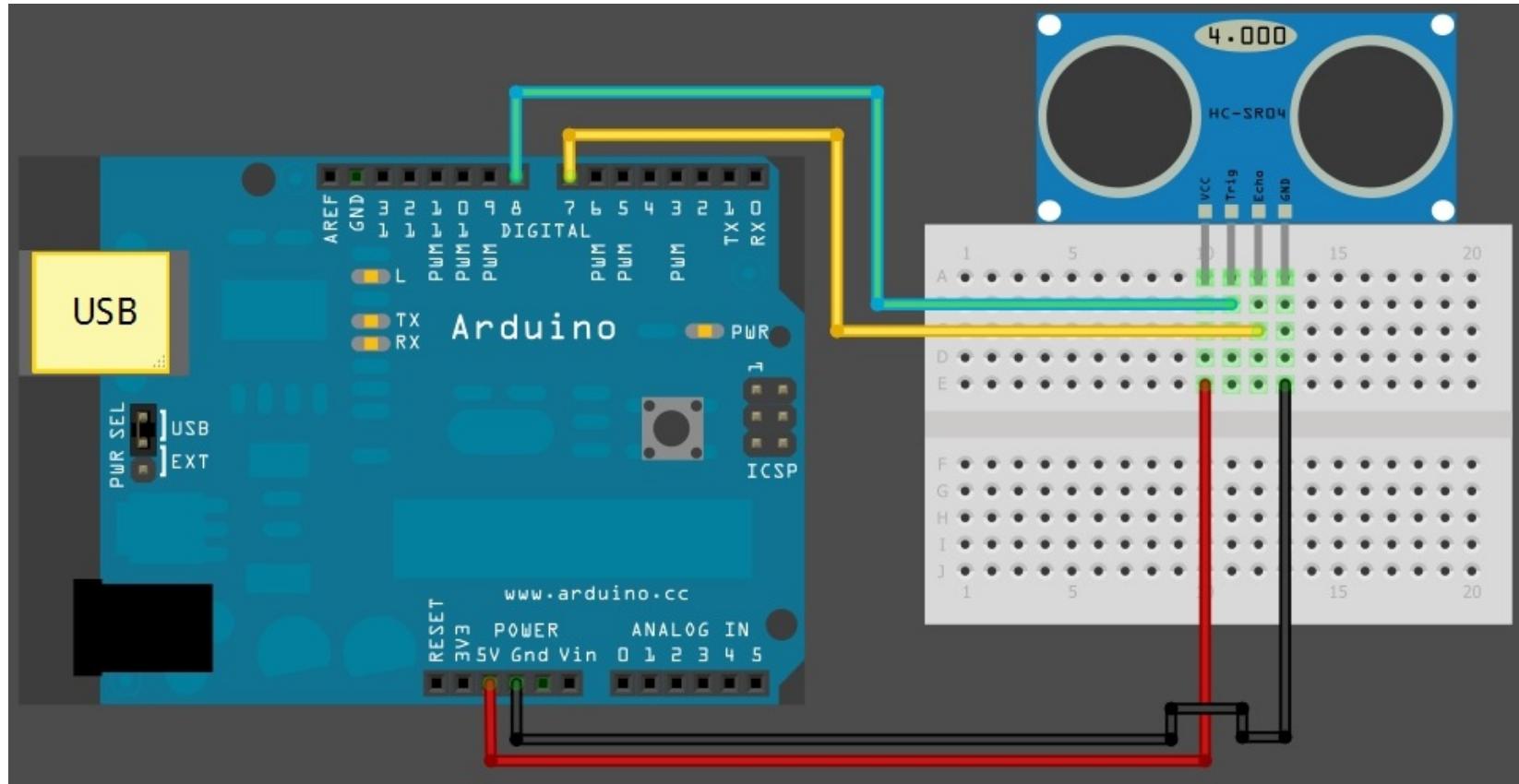


ESEMPIO DI UTILIZZO IN ARDUINO

- HC-SR04 - Caratteristiche
 - rilevazione distanze da 3 cm a 6 metri
 - 2 pin di controllo: **Echo** e **Trig**
- Strategia di utilizzo:
 - *si invia un impulso sul pin Trig e misura quanto tempo impiega ad arrivare l'echo sul pin Echo*
 - *dato tale tempo è possibile ricavare la distanza.*
- Principio di funzionamento
 - velocità suono $v_s = 330 \text{ m/s}$
 - inviando un impulso sonoro, nel tempo che intercorre dall'invio dell'impulso e il rilevamento dell'eco (diciamo dt), è stata percorsa una distanza pari al doppi di quella che ci separa dall'oggetto (andata e ritorno).
- Calcoli
 - $v_s * dt = 2 * d$
 - $d = (v_s * dt) / 2$
 - $dt = 2 * d / v_s$
- Quindi un oggetto a distanza di 50 cm = 0.5 metri è rilevato in $dt = 1/330 = 0.003030 \text{ s} = 3.030 \text{ ms} = 3030 \text{ us}$

ESEMPIO

- [esempio tratto da Make:Sensors]
 - collegamento di trig/echo ai pin 8/7



PROGRAMMA

```
const int trigPin = 8;
const int echoPin = 7;
/* supponendo di eseguire il test
   in un ambiente a 20 °C */
const float vs = 331.5 + 0.6*20;

void setup()
{
    Serial.begin(9600);
    pinMode(trigPin, OUTPUT);
    pinMode(echoPin, INPUT);
}

float getDistance()
{
    /* invio impulso */
    digitalWrite(trigPin,LOW);
    delayMicroseconds(3);
    digitalWrite(trigPin,HIGH);
    delayMicroseconds(5);
    digitalWrite(trigPin,LOW);

    /* ricevi l'eco */
    float tUS = pulseIn(echoPin, HIGH);
    float t = tUS / 1000.0 / 1000.0 / 2;
    float d = t*vs;
    return d;
}
...
```

<https://www.makerguides.com/hc-sr04-arduino-tutorial/>

```
...
void loop()
{
    float d = getDistance();
    Serial.println(d);
    delay(200);
}
```

LA ROUTINE `pulseIn`

- **`pulseIn`**
 - routine molto efficiente parte della libreria Wiring per Arduino
 - monitora il pin di input dove deve ricevere l'impulso, tenendo traccia del tempo intercorso
 - i tempi possono essere molto piccoli, dell'ordine delle decine di microsecondi => per cui è importante adottare un approccio estremamente efficiente
 - accesso diretto ai registri associati al pin
 - accesso diretto al registro che tiene traccia dei tick del processore
- Codice sorgente disponibile qui:
 - https://github.com/arduino/Arduino/blob/master/hardware/arduino/avr/cores/arduino/wiring_pulse.c

SENSORI PROSSIMITÀ OTTICI

- Si basano sulla rilevazione della riflessione di un fascio luminoso da parte dell'oggetto rilevato
 - chiamati anche fotoelettrici
- Normalmente viene usato un fascio di raggi infrarossi, in quanto questa radiazione difficilmente si confonde con i disturbi generati da fonti luminose ambientali.
- Nella modalità d'uso più semplice, il fascio viene riflesso dalla superficie stessa dell'oggetto rilevato, per lo stesso fenomeno per cui la luce visibile può essere riflessa e percepita dai nostri occhi.
- Il problema è che la quantità di radiazione riflessa dipende dalla composizione e dall'orientamento della superficie; pertanto il campo sensibile di questi sensori di prossimità dipende sostanzialmente dalla natura della superficie dell'oggetto da rilevare: tipicamente da 10 a 100 cm.
- Montando un *riflettore catadiotrico* sull'oggetto da rilevare, si possono ottenere portate nominali molto alte (fino a 50 m).

ESEMPIO

- Sharp GP2Y0A41SK0F IR
 - sensore a infrarossi interruttore fotoelettrico riflettente
 - [http://www.alldatasheet.com/view.jsp?
Searchword=Gp2y0a41sk0f](http://www.alldatasheet.com/view.jsp?Searchword=Gp2y0a41sk0f)



SENSORI DI MOVIMENTO

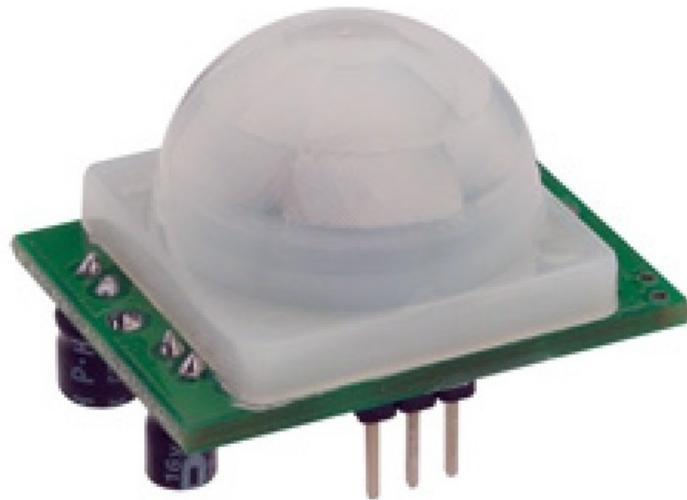
- PIR
- Trasduttori di posizione angolare
- till-ball switch
- rilevatori di vibrazioni

PIR - SENSORE INFRA-ROSSO PASSIVO

- Sensore elettronico che misura i raggi infrarossi (IR) irradiati dagli oggetti nel suo campo di vista e possono essere usati come rilevatori di movimento
- Un sensore PIR non rileva autonomamente un movimento; tuttavia, rileva *brusche variazioni di temperatura* che modificano lo stato che il PIR aveva "memorizzato come normale"
- Quando qualcosa o qualcuno passa di fronte a uno sfondo, ad esempio un muro, precedentemente "fotografato" dal PIR come stato normale, la temperatura in quel punto si innalza bruscamente, passando dalla temperatura della stanza a quella del corpo
 - questo rapido cambiamento attiva il rilevamento
 - lo spostamento di oggetti di temperatura identica, com'è prevedibile, non innesca alcuna rilevazione

ESEMPIO

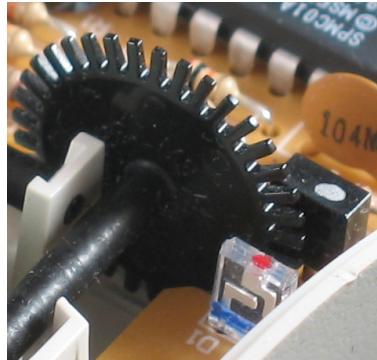
- PIR sensor
 - <https://learn.adafruit.com/pir-passive-infrared-proximity-motion-sensor/>



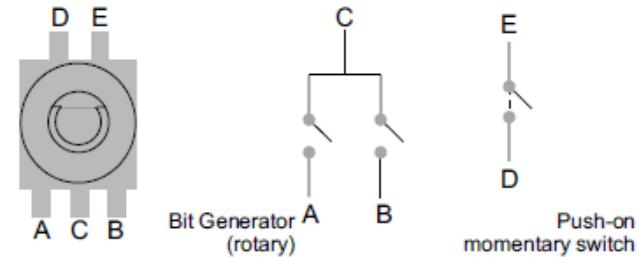
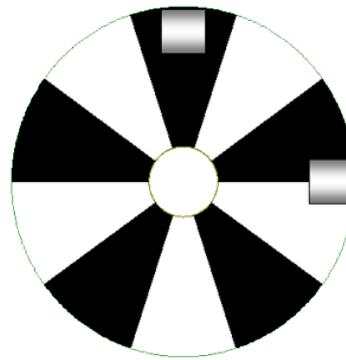
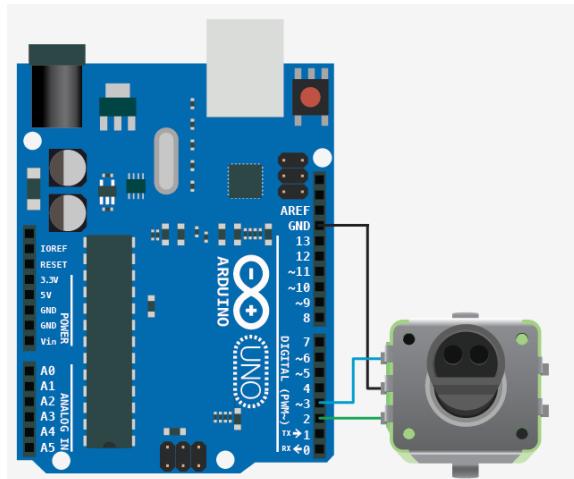
TRASDUTTORI DI POSIZIONE ANGOLARE

- Due tipi
 - uno rileva la posizione senza avere un vincolo meccanico, cioè sfrutta la forza di gravità
 - es: inclinometro, detto anche convertitore d'angolo
 - uno rileva la posizione sfruttando un vincolo meccanico => **encoder**
 - l'encoder è un dispositivo elettromeccanico che converte la posizione angolare del suo asse rotante in brevi impulsi elettrici che necessitano di essere elaborati da un circuito di analisi del segnale sotto forma di segnali numerici digitali
 - Vari tipi: tachimetrici, relativi (noti anche come "incrementali"), assoluti
- Varie tecnologie
 - capacitivi/induttivi, magnetici, potenziometrici, ottici

ROTARY ENCODER (ENCODER ROTATIVI)

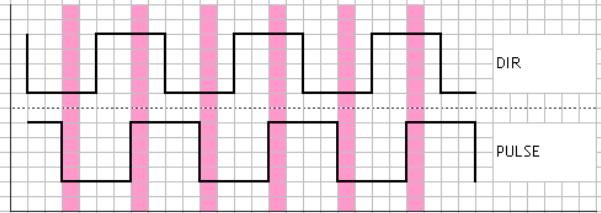


- 2 pin che corrispondono ai contatti che forniscono due segnali sfalsati di 90° (CLK, DT)
- SW è un pulsante interno
- VCC, GND



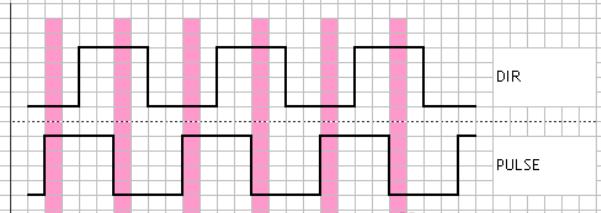
Movimento in avanti

(al verificarsi dell'interrupt i due segnali sono allo stesso livello logico)



Movimento all'indietro

(al verificarsi dell'interrupt i due segnali sono a livelli logici differenti)



settorezero.com

<http://www.giuseppecaccavale.it/arduino/rotary-encoder-arduino/>

<http://www.settorezero.com/wordpress/corso-programmazione-picmicro-in-c-%E2%80%93-approfondimenti-%E2%80%93-come-utilizzare-un-encoder-in-quadratura-per-limmissione-dei-dati/>

SENSORI ACCELERAZIONE

- **Accelerometro**
 - sensore che misura l'accelerazione a cui è soggetto l'oggetto a cui è connesso il sensore
 - misurandone in realtà la forza specifica (= per unità di massa)
 - vari tipi, che si basano su principi di funzionamento diversi: acc. estensimetrico, piezoresistivo, capacitivo, piezoelettrico, gravitometro,...
- **Giroscopio**
 - dispositivo fisico rotante mantiene il suo asse di rotazione orientato in una direzione fissa, per effetto della legge di conservazione del momento angolare
 - usato per scopi vari, fra i quali può fungere da bussola, indicando la direzione verso il nord

ESEMPIO

- MPU-6050 - accelerometro + giroscopio
 - <http://playground.arduino.cc/Main/MPU-6050>



TECNOLOGIA MEMS

- Gli accelerometri e giroscopi elettronici citati si basano su tecnologia **MEMS**
 - Micro Electro-Mechanical Systems
- I MEMS sono costituiti da insieme di dispositivi di varia natura (meccanici, elettrici ed elettronici) integrati in forma altamente miniaturizzata (ordine dei micro-metri) su uno stesso substrato di materiale semiconduttore
 - es: silicio
- Coniugano le proprietà elettriche degli integrati a semiconduttore con proprietà opto-meccaniche
 - abbinano funzioni elettroniche, di gestione dei fluidi, ottiche, biologiche, chimiche e meccaniche in uno spazio, integrando la tecnologia dei sensori e degli attuatori e le più diverse funzioni di gestione dei processi
- Fondamentali per lo sviluppo di sistemi embedded moderni

SENSORI DI CONTATTO

- **Pulsanti tattili e microswitch**
- **Potenziometri**
 - permette di variare in modo meccanico la resistenza del componente, e di conseguenza la tensione ai morsetti
 - dispositivo elettrico equivalente ad un partitore di tensione resistivo variabile
- **Potenziometro lineare**
 - la tensione di uscita è legata linearmente a quella di ingresso attraverso un rapporto
- **Sensori capacitivi**
 - si basano sul principio della rilevazione della capacità elettrica di un condensatore: il loro lato sensibile ne costituisce un'armatura, l'eventuale presenza nelle immediate vicinanze di un oggetto conduttore, realizza l'altra armatura del condensatore.
 - così la presenza di un oggetto crea una capacità che i circuiti interni rilevano, comandando la commutazione del segnale d'uscita.

ESEMPIO

- Potenziometro ALPS 10KOhm



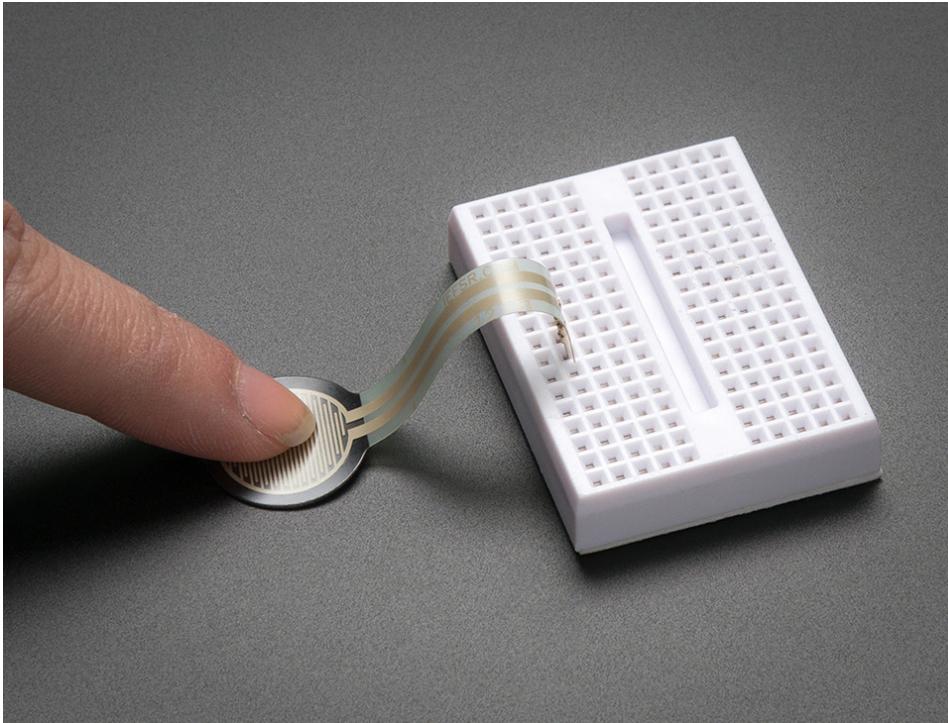
ESEMPIO

- Sensore capacitivo AllLife TTP223B
 - [http://www.datasheet4u.com/share_search.php?
sWord=TTP223B](http://www.datasheet4u.com/share_search.php?sWord=TTP223B)



SENSORI DI PRESSIONE

- Permettono di misurare la forza esercitata su una determinata superficie
 - esempio: Round Force-Sensitive Resistor (FSR)
 - <https://www.adafruit.com/products/166>



SENSORI OTTICI

- Un sensore di immagini o sensore ottico è un dispositivo che **converte una immagine ottica in un segnale elettrico**
 - componenti vengono utilizzati soprattutto nelle fotocamere digitali, nelle telecamere, nelle videocamere e in altri dispositivi che trattano elettronicamente immagini.
- Esistono molti tipi di sensori di immagini
 - in base a caratteristiche come metodo di rilevamento dei colori, tecnologia, sensibilità alla luce, risoluzione ecc.
 - una classificazione di base: quelli che riprendono a colori e quelli che riprendono in bianco e nero (usati ad esempio nella visione notturna)
- Principio base di funzionamento
 - l'immagine viene focalizzata su una griglia composta da una miriade di piccoli sensori puntiformi
 - i singoli sensori convertono la luminosità rilevata
 - il sensore che riprende in bianco e nero è quello più semplice ed è anche il primo nato, successivamente si è arrivati a rilevare anche il colore

FOTO-RIVELATORI

- Dispositivo in grado di **rivelare la radiazione elettromagnetica**, fornendo in uscita un segnale avente un'intensità di corrente o una differenza di potenziale proporzionale all'intensità della radiazione rilevata
- Tipi diversi di fotorivelatore, realizzati in base a diversi effetti di interazione tra la radiazione e la materia
 - possono differire per la porzione di spettro elettromagnetico che sono in grado di rilevare, e per l'intensità luminosa minima che riescono a misurare (alcuni sono in grado di rivelare i singoli fotoni)
- Questi dispositivi sono indicati anche con il termine fotocellula
- Varie tecnologie
 - **fotoresistenza**: basato sulle cariche fotogenerate (cioè sugli elettroni eccitati dalla luce incidente dalla banda di valenza alla banda di conduzione) in un semiconduttore
 - **fotodiode**: basato sulle cariche fotogenerate in una giunzione p-n
 - cella fotovoltaica: simile ad un fotodiode, ma non deve essere polarizzata per funzionare
 - **Charge Coupled Device (CCD)**, Charge Injection Device (CID) e CMOS: circuiti integrati basati su cariche fotogenerate in un semiconduttore

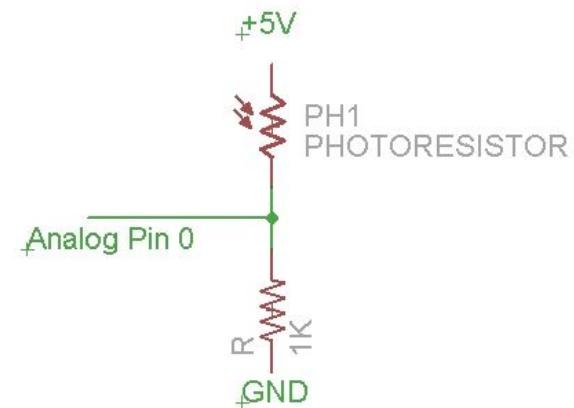
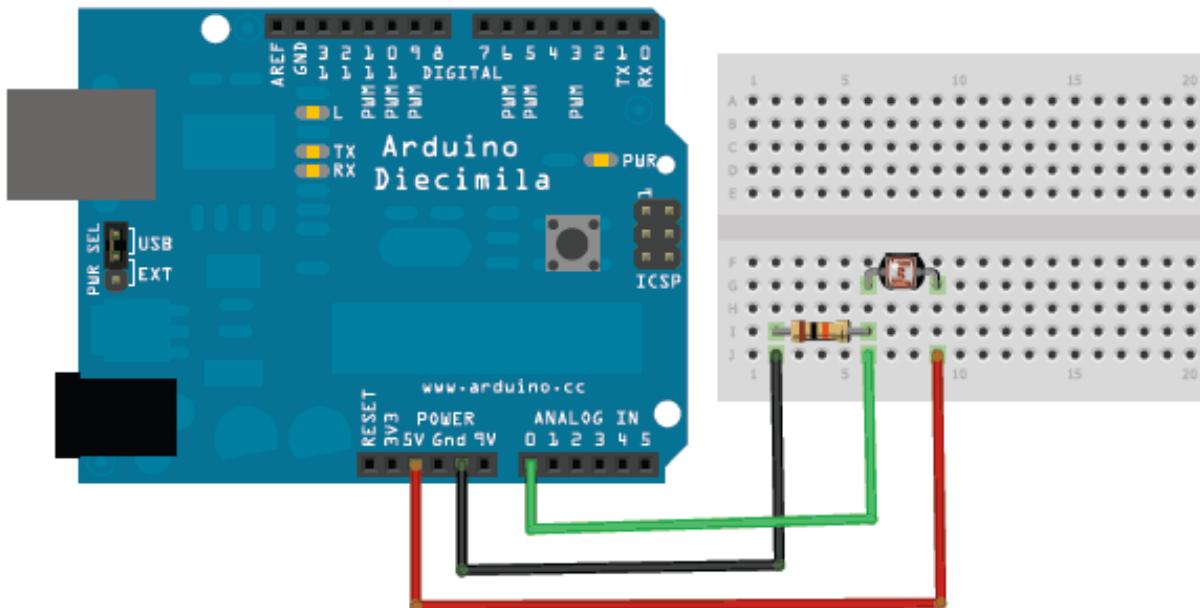
ESEMPIO

- Fotoresistenza
 - arduino.cc/documents/datasheets/LDR-VT90N2.pdf
- resistenza con valore ohmmico inversamente proporzionale alla luce incidente sulla sua superficie attiva.
 - luce forte => resistenza piccola
 - luce debole => resistenza alta.
- Parametri
 - R10: resistenza minima e massima quando viene illuminata da una luce con quantità 10 lux
 - R100 resistenza tipica che potresti misurare ai poli della fotoresistenza quando viene illuminata da una luce con quantità 100 lux
 - R05 definisce la resistenza dopo che è stata messa al buio per almeno 5 secondi



ESEMPIO

- schema collegamento: partitore di tensione
 - uso resistenza di valore noto in serie alla fotoresistenza



ELETTRICITA' E MAGNETISMO

- **Rilevatori di tensione e di corrente**
 - permettono di misurare la tensione fra due punti di un circuito o di misurare la corrente
 - utili in particolare in tutte le applicazioni dove si debba misurare il consumo di un certo apparecchio elettronico
 - esempio: progetto <http://www.openenergymonitor.org>)
- **Magnetometro**
 - strumento che misura il campo magnetico.
 - possono essere scalari (misurano il modulo del campo magnetico) oppure vettoriali (misurano la componente del campo magnetico lungo una particolare direzione dello spazio)
- **Bussola (compass)**
 - strumento che indica sempre il nord magnetico

TEMPO ATMOSFERICO E CLIMA

- **Sensore di temperatura**

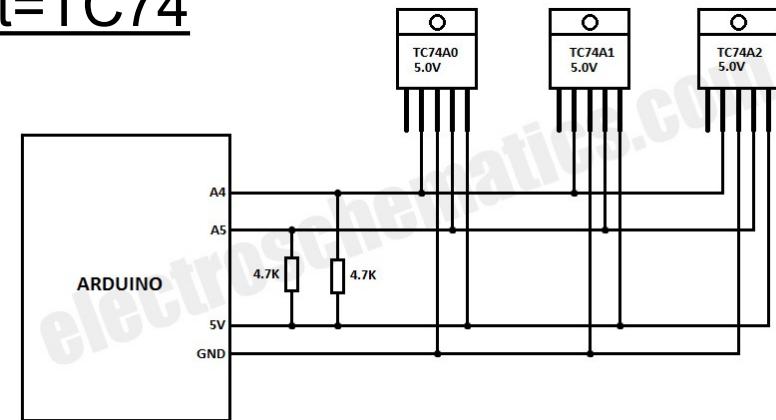
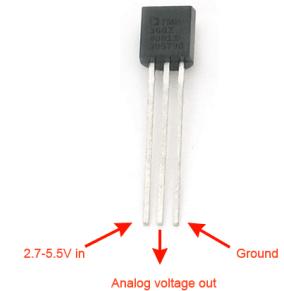
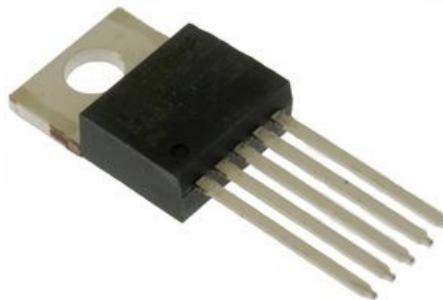
- esempi:

- sensore analogico TMP36

- <https://learn.adafruit.com/tmp36-temperature-sensor>

- sensore digitale TC74

- <http://www.microchip.com/wwwproducts/Devices.aspx?product=TC74>



TEMPO ATMOSFERICO E CLIMA

- Sensore di umidità
 - esempio: DHT11 Digital Temperature Humidity Sensor
 - <http://www.dx.com/p/arduino-digital-temperature-humidity-sensor-module-121350>



- Sensore di pressione barometrica
 - esempio: BMP085 SparkFun Electronics
 - <http://www.robotstore.it/product/504/Sensore-di-Pressione-Barometrica-BMP085.html>



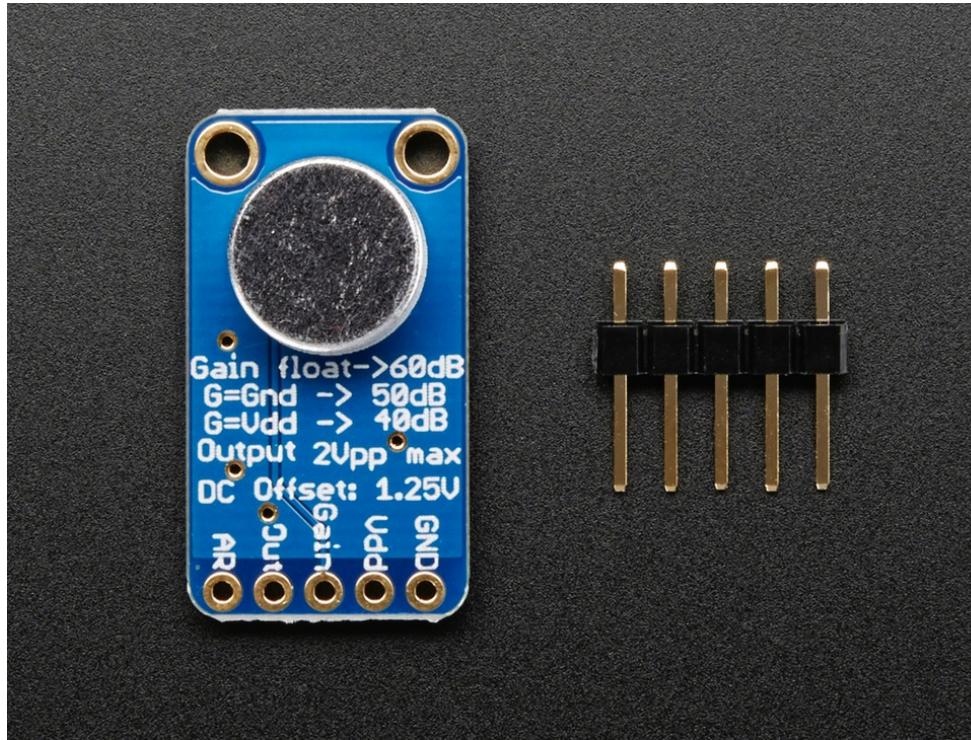
SOSTANZE CHIMICHE

- Rilevatori di fumo
 - MQ Gas Sensor
 - <http://playground.arduino.cc/Main/MQGasSensors>



SUONO

- Microfono
 - esempio:
 - <https://www.adafruit.com/products/1713>

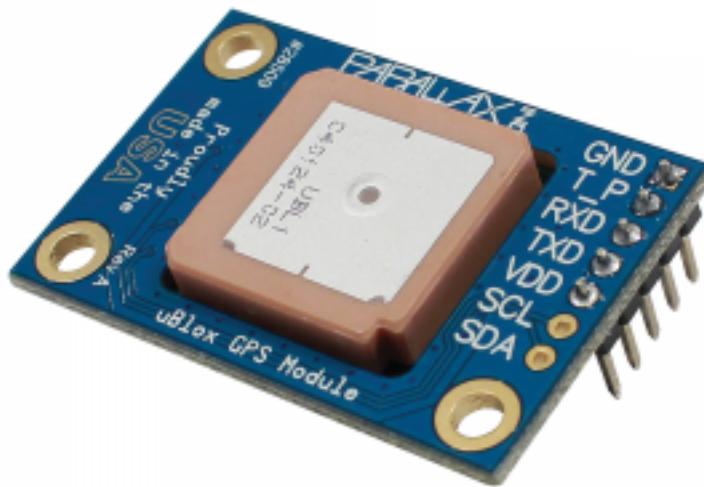


“SENSORI” DI POSIZIONE: GPS

- Il Sistema di Posizionamento Globale (Global Positioning System, GPS) è un sistema di posizionamento e navigazione satellitare civile che, attraverso una rete satellitare dedicata di satelliti artificiali in orbita, fornisce ad un terminale mobile o ricevitore GPS informazioni sulle sue coordinate geografiche ed orario, in ogni condizione meteorologica, ovunque sulla Terra o nelle sue immediate vicinanze ove vi sia un contatto privo di ostacoli con almeno quattro satelliti del sistema.
 - sistema gestito dal governo degli Stati Uniti d'America, ma liberamente accessibile da chiunque sia dotato di un ricevitore GPS
 - **grado di accuratezza dell'ordine dei metri**
 - in dipendenza dalle condizioni meteorologiche, dalla disponibilità e dalla posizione dei satelliti rispetto al ricevitore, dalla qualità e dal tipo di ricevitore, dagli effetti di radiopropagazione del segnale radio in ionosfera e troposfera (es. riflessione) e dagli effetti della relatività.
- Principio di funzionamento
 - si basa su un metodo di posizionamento sferico (*trilaterazione*), che parte dalla misura del tempo impiegato da un segnale radio a percorrere la distanza satellite-ricevitore
 - la localizzazione avviene tramite la trasmissione di un segnale radio da parte di ciascun satellite e l'elaborazione dei segnali ricevuti da parte del ricevitore

ESEMPIO

- Parallax PAM-7Q GPS Module
 - <http://www.parallax.com/product/28509>
 - “it provides all of the standard services like location fix, speed, heading, altitude measurements, and atomic date and time”



SENSORI PER IDENTIFICAZIONE

- **RFID**
 - tecnologia che permette di avere dei piccoli tag in grado di contenere un certo quantitativo di dati e di essere rilevati, letti e scritti a distanza, wireless da opportuni lettori
- **NFC**
 - concettualmente analoghi ad RFID, la scoperta e interazione avviene per contatto fra lettore e tag
 - esempio di lettore NFC/RFID: <http://www.adafruit.com/product/789>
- **iBeacon**
 - Dispositivi che sfruttano la tecnologia BLE (Bluetooth low energy) per trasmettere uno UUID nel raggio di una certa località e rilevabile mediante opportuni ricevitori Bluetooth 4.0
 - esempio: <http://www.seeedstudio.com/depot/blebee-v100-p-1632.html>

SENSORI PER INPUT DATI

- Keypad
 - per input dati alfanumerici
 - esempio: ADAFruit
 - <http://www.adafruit.com/products/419>

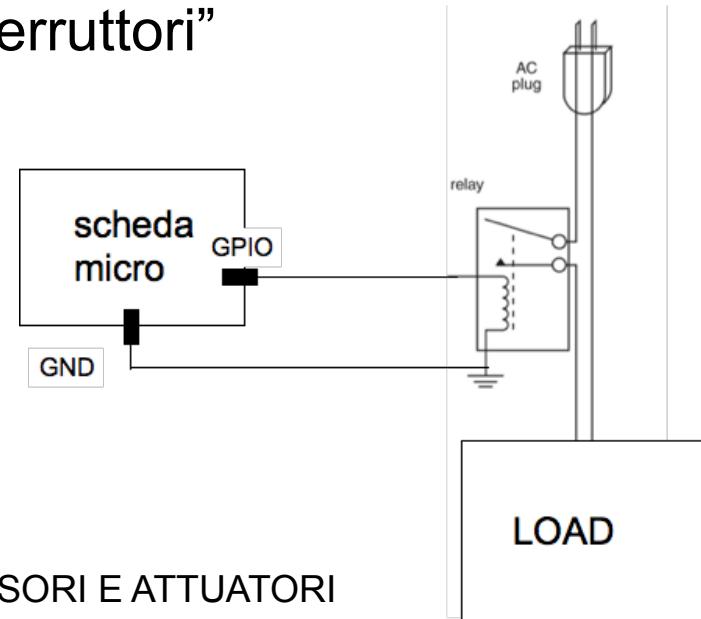
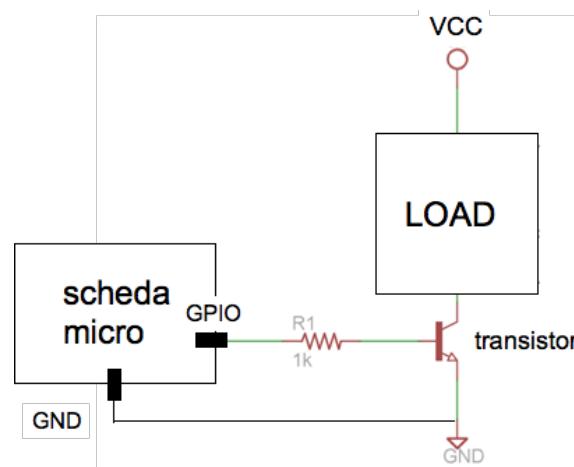
ATTUATORI E DISPOSITIVI DI OUTPUT

TRASUTTORI DI OUTPUT (ATTUATORI)

- Trasduttori
 - convertono una forma di energia in un'altra
- **Trasduttori di output (attuatori)**
 - permettono di realizzare sistemi embedded che eseguono e controllano azioni sull'ambiente fisico in cui sono immersi
 - azione più semplice: accendere o spegnere un dispositivo
- Esempi:
 - led, piezo buzzer, motori
 - relays (switch)
 - in generale: ogni dispositivo che può essere acceso o spento: una ventola, una radio, un'automobile

INTERFACCIAMENTO

- Due casi principali
 - è sufficiente la corrente/tensione in uscita ai GPIO (es: 5 Volt - 20mA) per alimentare il trasduttore
 - es: led, piezo
 - in caso contrario, il dispositivo deve essere alimentato da un circuito (e alimentazione) separato
 - in questo caso, mediante un GPIO si apre/chiude l'altro circuito mediante dispositivi come **transistor** e **relè** (relays) che fungono da “interruttori”



CARICHI RESISTIVI E INDUTTIVI

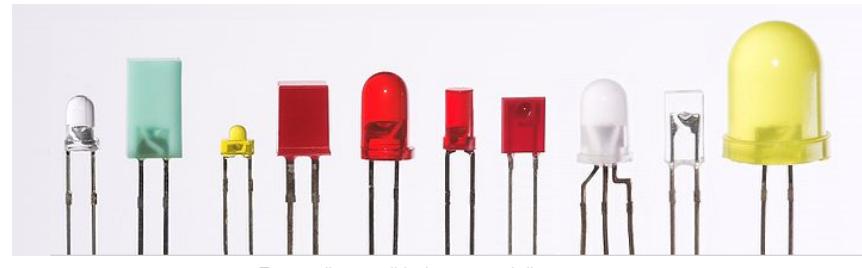
- Dal punto di vista elettronico, i dispositivi pilotabili sono classificabili in 2 categorie
 - carichi **resistivi**
 - possono essere assimilati ad un componente che al passaggio della corrente oppone una certa resistenza che implica una variazione della tensione
 - carichi **induttivi**
 - dispositivi che operano inducendo corrente su un filo mediante la corrente su un altro filo o immergendo il filo in un campo magnetico
 - esempio: motori, solenoidi
 - *Generano una tensione/corrente inversa*
 - è necessario dotare il circuito di diodi di protezione per eliminare o ridurre l'effetto di questa corrente, che potrebbe altrimenti danneggiare il circuito

ESEMPI DI ATTUATORI

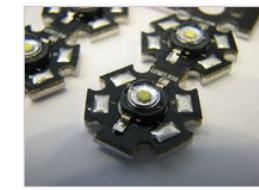
- Led
- Display LCD
- Motori elettrici
 - corrente continua
 - servo motori
- Speaker
 - permettono di produrre suoni
 - vedere cap. 5 [EA]
- Dispositivi di pilotaggio di carichi di potenza (attuatori)
 - fotoaccoppiatori, relè (relays)

LED

- Light Emitting Diode
 - Diodo ad emissione luminosa
- Dispositivi optoelettronico che sfruttano le proprietà ottiche di alcuni materiali semiconduttori di produrre fotoni attraverso un fenomeno di emissione spontanea
- Particolarmente interessanti per le loro caratteristiche di elevata efficienza luminosa A.U./A e di affidabilità.
- Recentissima diffusione massiccia in tutte le applicazioni in cui serve:
 - elevata affidabilità
 - lunga durata
 - elevata efficienza
 - basso consumo



Led da 8, 5 e 3 mm



Un esempio di led alta potenza

DISPLAY LCD

- Display basato sulle proprietà ottiche di particolari sostanze denominate cristalli liquidi
 - tale liquido è intrappolato fra due superfici vetrose provviste di numerosi contatti elettrici con i quali poter applicare un campo elettrico al liquido contenuto
 - ogni contatto elettrico comanda una piccola porzione del pannello identificabile come un pixel (o subpixel per gli schermi a colori)
- Basso consumo di potenza elettrica
 - che li rende di per sé particolarmente indicati per sistemi embedded

ESEMPIO

- Un tipo molto comune di display LCD è dato dal parallel LCD screen 16x2 (16 caratteri per 2 righe), caratterizzati da una singola fila di 16 pin
- Tutti i parallel LCD screen hanno lo stesso pin-out e possono essere collegati ad Arduino in vari modi
 - modo 4 pin o 8 pin
 - modo 8 pin, 6 pin sono da collegare ai pin digitali + 2 sono da collegare VCC e GND
 - dei 6 pin, 4 servono per trasmettere i dati - gli altri due per funzioni di controllo
 - via I2C
 - es: <http://www.robot-italy.com/it/display-lcd-16-x-2-blu-i2c.html>
- Programmazione
 - libreria LiquidCrystal Library fornita con Arduino IDE
 - numerose funzionalità
 - blinking del cursore, scrolling automatico del testo, creazione di caratteri personalizzati, selezione della direzione della stampa

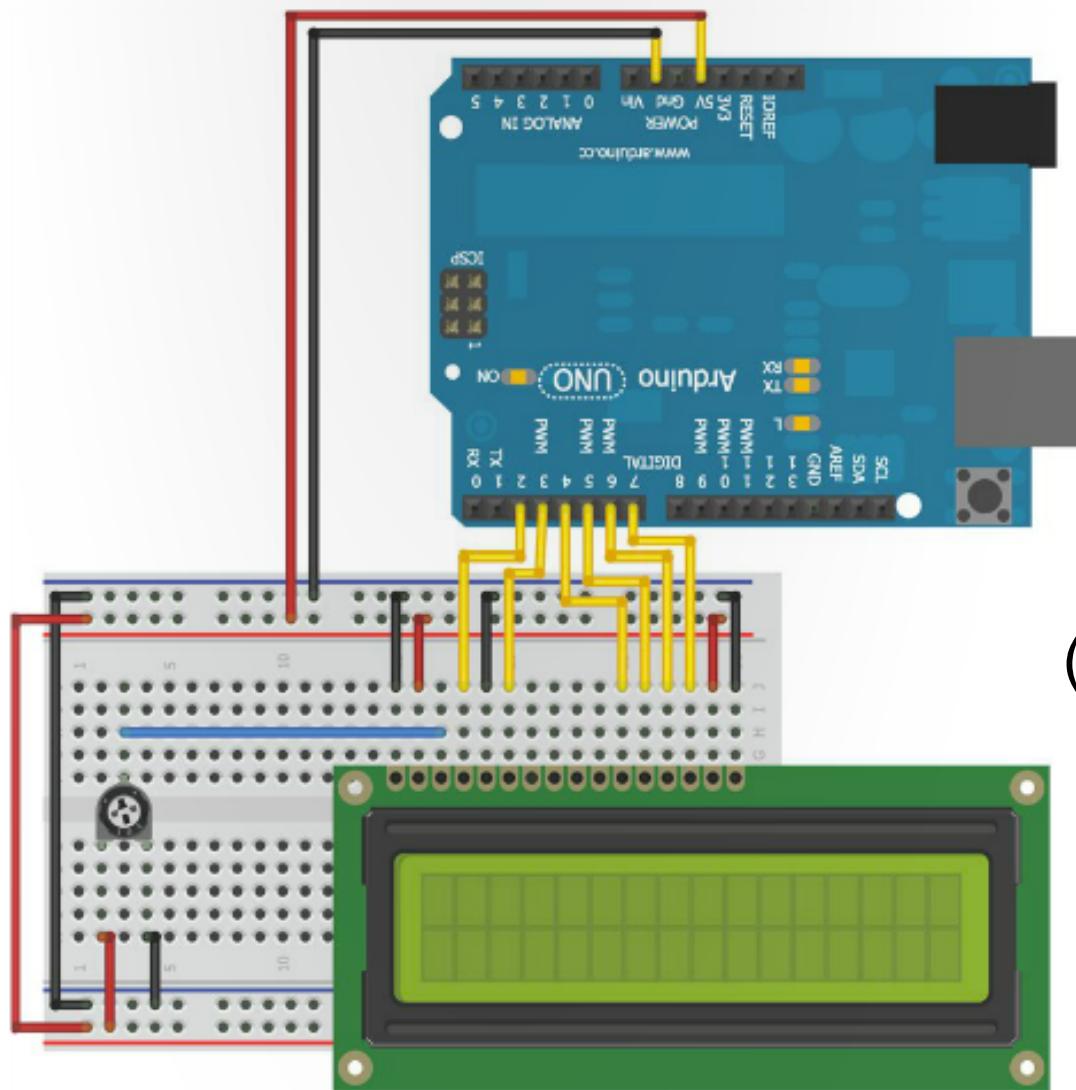
ESEMPIO

Table 10-1: Parallel LCD Pins

PIN NUMBER	PIN NAME	PIN PURPOSE
1	VSS	Ground connection
2	VDD	+5V connection
3	V0	Contrast adjustment [to potentiometer]
4	RS	Register selection [Character vs. Command]
5	RW	Read/write
6	EN	Enable
7	D0	Data line 0 [unused]
8	D1	Data line 1 [unused]
9	D2	Data line 2 [unused]
10	D3	Data line 3 [unused]
11	D4	Data line 4
12	D5	Data line 5
13	D6	Data line 6
14	D7	Data line 7
15	A	Backlight anode
16	K	Backlight cathode



ESEMPIO TRATTO DA [EA]



(p. 203, [EA])

Image created with Fritzing.

ESEMPIO PROGRAMMA

```
#include <LiquidCrystal.h>

int time = 0

/* sequenza dei pin: RS EN D4 D5 D6 D7 */
LiquidCrystal lcd(2, 3, 4, 5, 6, 7);

void setup()
{
    lcd.begin(16, 2); // display 16 caratteri per 2 righe
    lcd.print("Cesena");
}

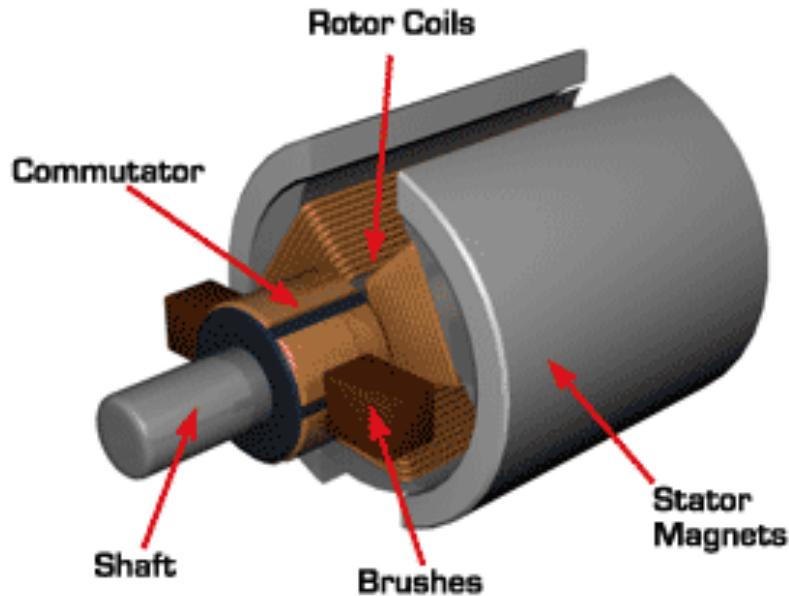
void loop()
{
    lcd.setCursor(0,1);
    lcd.print(time);
    delay(1000);
    time++;
}
```

- Vedere dettagli sul testo [EA], p. 204

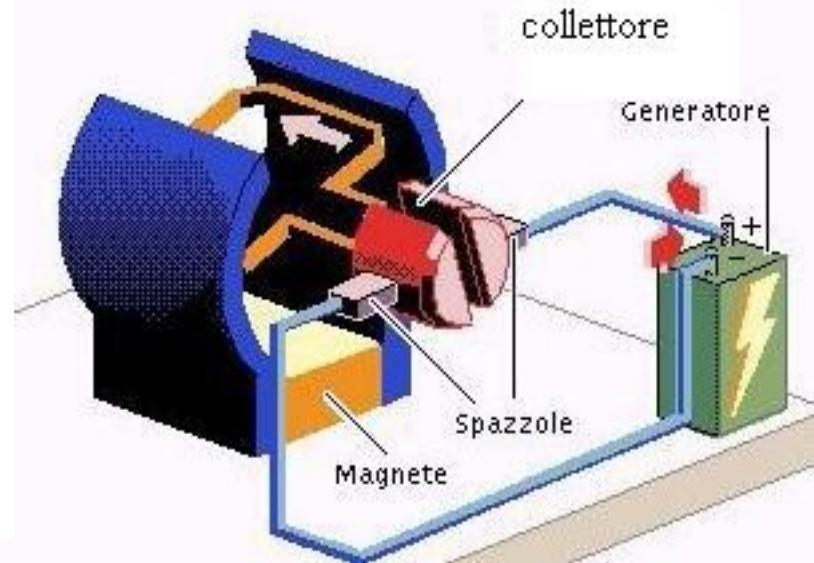
MOTORI ELETTRICI

- Macchine elettriche in cui la potenza di ingresso è di tipo elettrico e quella di **uscita è di tipo meccanico**, assumendo la funzione di attuatore
- Il motore elettrico, così come l'alternatore, è composto da:
 - statore
 - rotore
- Questi componenti generano un campo magnetico, in alcuni casi anche grazie all'uso di magneti, che determinano quindi il movimento del rotore
- Fra i motori elettrici più usati nell'embedded:
 - motori in corrente continua (motori CC, motors DC in inglese)
 - motori passo-passo
 - servo-motori

MOTORI ELETTRICI



Motore a corrente continua

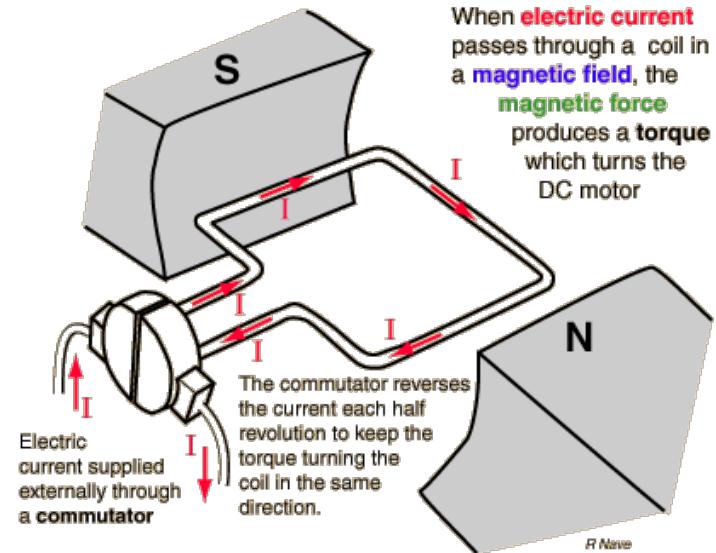
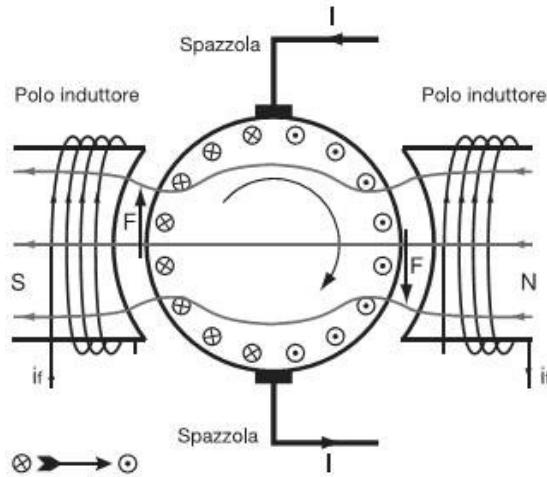


MOTORI IN CORRENTE CONTINUA

- Sono particolarmente indicati nei progetti in cui occorre:
 - la rotazione continua
 - 360° di movimento continuo
 - elevata velocità o coppia (motoriduttori)
 - controllare la velocità di rotazione
- Esempi di utilizzo
 - le ruote di un robot;
 - un azionamento meccanico mediante trasmissione a cinghia o ruote dentate
 - uno slider per telecamera
 - una avvolgi tenda o tapparella
 - ...

MOTORI: FUNZIONAMENTO

- La corrente elettrica continua passa in un avvolgimento di spire che si trova nel rotore, composto da fili di rame, creando un campo elettromagnetico al passaggio di corrente
- Questo campo elettromagnetico è immerso in un altro campo magnetico creato dallo statore, il quale è caratterizzato dalla presenza di una o più coppie polari (calamite, elettrocalamite, ecc.)
- Il rotore per induzione elettromagnetica inizia a girare, in quanto il campo magnetico del rotore tende ad allinearsi a quello dello statore analogamente a quanto avviene per l'ago della bussola che si allinea col campo magnetico terrestre.
- Durante la rotazione il sistema costituito dalle spazzole e dal collettore commuta l'alimentazione elettrica degli avvolgimenti del rotore in modo che il campo magnetico dello statore e quello del rotore non raggiungano mai l'allineamento perfetto, in tal modo si ottiene la continuità della rotazione.



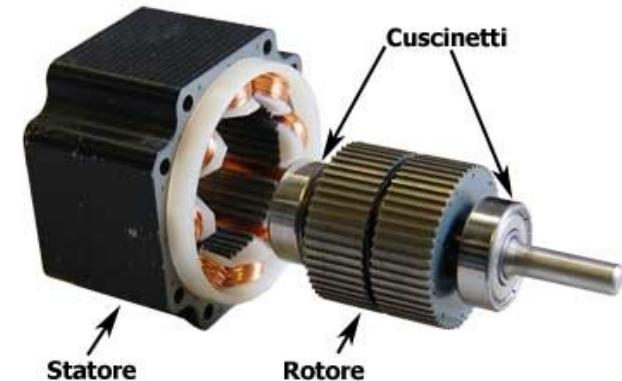
ESEMPIO

- Esempio per progetti in Arduino
 - 711696 Hobby Motor - Gear
 - <http://www.robot-italy.com/it/hobby-motor-gear.html>



MOTORI PASSO-PASSO

- Il motore passo-passo spesso chiamato anche step o stepper è un motore elettrico in corrente continua senza spazzole (*brushless*) che può suddividere la propria rotazione in un grande numero di step
- **La posizione del motore può essere controllata accuratamente**
- Trovano il loro miglior impiego nei progetti in cui hai bisogno di controllare in modo accurato la posizione dell'albero.
- Alcuni esempi:
 - stampanti 3D
 - macchine a controllo numerico
 - robotica per controllo bracci
 - ...



SERVO MOTORI

- Come i passo-passo permettono di pilotare in modo preciso l'angolo in cui deve posizionarsi il rotore
- A differenza dei motori passo-passo, la specifica della posizione è assoluta, non relativa alla posizione corrente
 - semplifica la programmazione
- Lo svantaggio rispetto ai passo-passo è l'escursione limitata (0-180°)
- Esempio per progetti Arduino
 - 31053 HS-53 Microservo
 - <http://www.robot-italy.com/it/hs-53-microservo.html>



SERVO MOTORI

- I servo motori si controllano specificando precisamente l'angolo al quale devono posizionare l'albero
 - l'angolo coperto tipicamente è da -90 a +90 gradi
- A livello di dispositivo sono caratterizzati da tre fili
 - 2 per l'alimentazione (nero => GND, rosso => +5V)
 - 1 come segnale di controllo digitale
 - solitamente bianco o giallo
- **Il controllo avviene inviando uno stream di impulsi al segnale di controllo ad una specifica frequenza**
 - tipicamente 50 impulsi al secondo, 50 Hz
- *La lunghezza dell'impulso determina l'angolo al quale vogliamo portare il servo*
 - gli impulsi tipicamente vanno da 1 ms (-90) a 2 ms (+90)
 - più lunga è la durata dell'impulso, più grande è l'angolo specificato
 - la scala è lineare, per cui l'impulso di durata intermedia (1.5 ms) indica il centro (0 gradi)
 - tuttavia la lunghezza reale degli impulsi può variare in modo significativo da servo a servo...

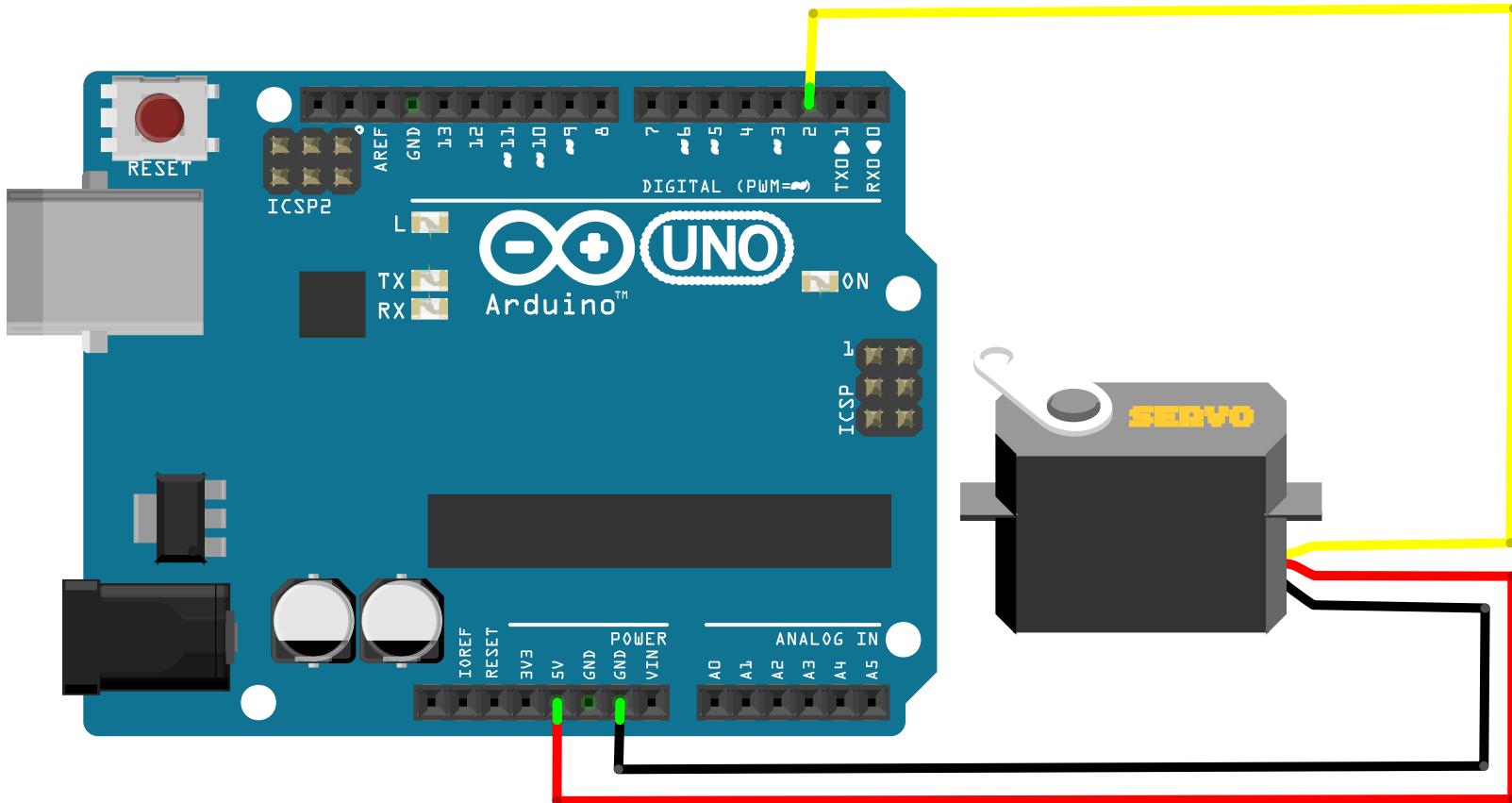
TEST: SERVO MOTORE HS-53

- Servo motore molto economico



- specifiche in <http://hitecrcd.com/products/servos/micro-and-mini-servos/analog-micro-and-mini-servos/hs-53-super-economy-feather-nylon-gear-servo/product>

SCHEMA DI COLLEGAMENTO



fritzing

TEST IN ARDUINO

- Usiamo i power pin di Arduino per alimentare il motore
 - configurazioni più complesse possono richiedere alimentazione esterna e uno schema circuitale più complesso
 - si vedrà un esempio più avanti in merito
- Per controllare il motore è possibile usare un qualsiasi pin digitale, utilizzato in OUTPUT
 - scegliamo il pin 2
- Nel programma che segue, si porta il servo ripetutamente dalla posizione -90 a +90

PROGRAMMA

```
int servoPin = 2;                                ...
void pulseServo(int servoPin, int pulseLen)        void loop(){
{                                                 c++;
    digitalWrite(servoPin, HIGH);                switch (state) {
    delayMicroseconds(pulseLen);                case MINUS_90:
    digitalWrite(servoPin,LOW);                  pulseServo(servoPin, 250);
    delay(15);                                 if (c > 100){
                                                Serial.println("--> +90");
                                                state = PLUS_90;
                                                c = 0;
}
int c;
enum { MINUS_90, PLUS_90 } state;

void setup()
{
    pinMode(servoPin, OUTPUT);
    state = MINUS_90;
    Serial.begin(9600);
    c = 0;
}
...
}                                                 }
}                                                 }
}                                                 }
```

NOTE

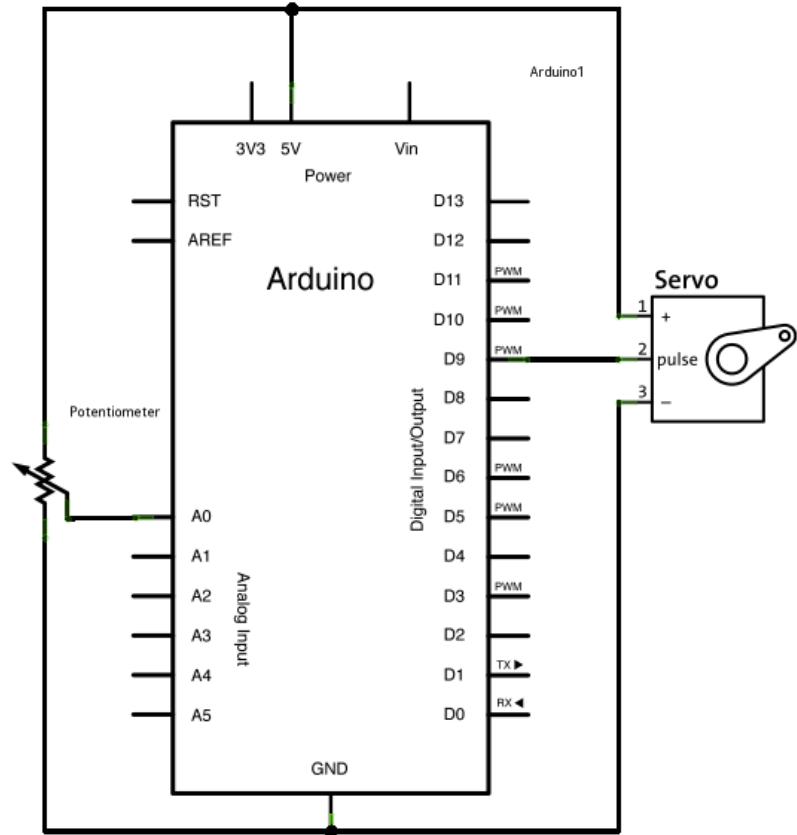
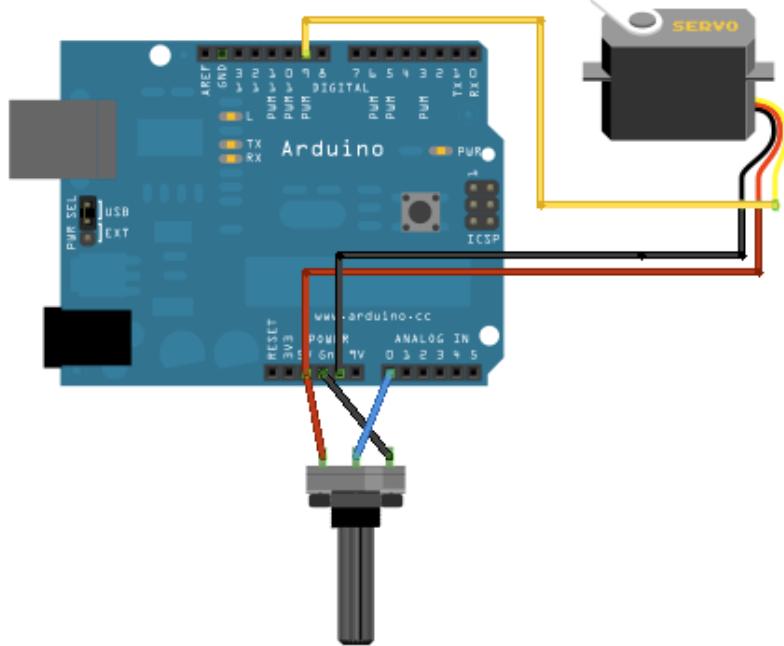
- La procedura pulseServo invia un impulso della durata specificata (in microsecondi) e quindi attende 15 ms
 - in realtà, per avere un segnale a 50 Hz, l'attesa dovrebbe essere pari a $1000/50 = 20\text{ms}$
 - tuttavia le istruzioni che vengono eseguite ciclicamente hanno una durata non nulla, per cui è stato scelto empiricamente un periodo inferiore
- Empiricamente, per lo specifico servo testato, la lunghezza dell'impulso che corrisponde a -90 è circa 250ms, mentre +90 è circa 2250 ms
 - quindi valori piuttosto lontani da quelli da usare in teoria (1000ms e 2000ms)

LIBRERIA SERVO

- Disponibile libreria Servo
 - <https://www.arduino.cc/en/Reference/Servo>
 - controllo di semplici servo motori, mediante segnali PWM
- Fra le funzioni nell'API della libreria
 - attach(pin)
 - per specificare il pin a cui è collegato il motore
 - su Arduino UNO versione prima della 0017 => solo 9 e 10
 - write(angle)
 - per specificare l'angolo, da 0 a 180
- Esempi
 - “Knob”
 - controllo mediante un potenziometro analogico
 - <https://www.arduino.cc/en/Tutorial/Knob>
 - “Sweep”
 - continua rotazione avanti indietro
 - <https://www.arduino.cc/en/Tutorial/Sweep>

KNOB

- <https://www.arduino.cc/en/Tutorial/Knob>



KNOB

```
/*
Controlling a servo position using a potentiometer (variable resistor)
by Michal Rinott <http://people.interaction-ivrea.it/m.rinott>

modified on 8 Nov 2013
by Scott Fitzgerald
http://www.arduino.cc/en/Tutorial/Knob
*/
#include <Servo.h>

Servo myservo; // create servo object to control a servo

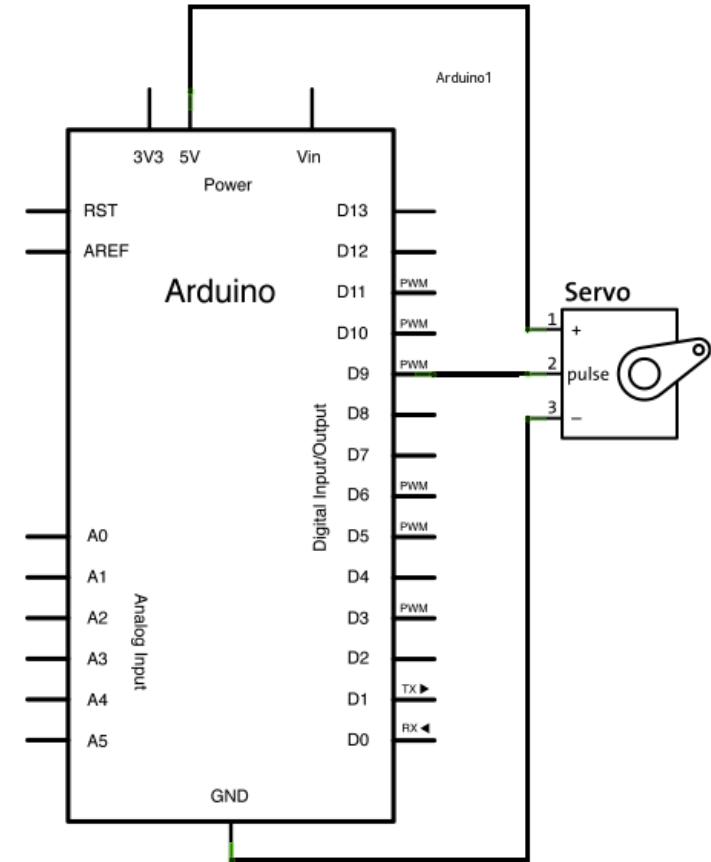
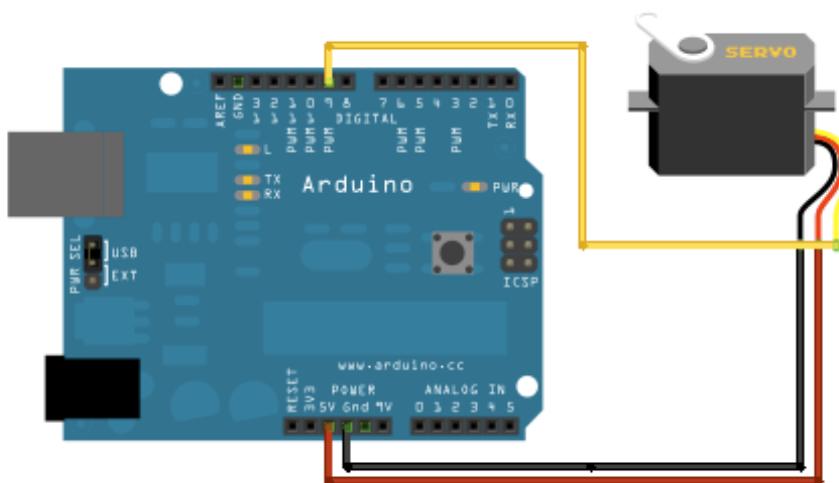
int potpin = 0; // analog pin used to connect the potentiometer
int val; // variable to read the value from the analog pin

void setup() {
  myservo.attach(9); // attaches the servo on pin 9 to the servo object
}

void loop() {
  val = analogRead(potpin); // reads the value of the potentiometer (value between 0 and 1023)
  val = map(val, 0, 1023, 0, 180); // scale it to use it with the servo (value between 0 and 180)
  myservo.write(val); // sets the servo position according to the scaled value
  delay(15); // waits for the servo to get there
}
```

SWEET

- <https://www.arduino.cc/en/Tutorial/Sweep>



SWEET

```
/* Sweep
by BARRAGAN <http://barraganstudio.com>
This example code is in the public domain.
modified 8 Nov 2013
by Scott Fitzgerald
http://www.arduino.cc/en/Tutorial/Sweep
*/
#include <Servo.h>

Servo myservo; // create servo object to control a servo

int pos = 0; // variable to store the servo position

void setup() {
  myservo.attach(9); // attaches the servo on pin 9 to the servo object
}

void loop() {
  for (pos = 0; pos <= 180; pos += 1) { // goes from 0 degrees to 180 degrees
    // in steps of 1 degree
    myservo.write(pos); // tell servo to go to position in variable 'pos'
    delay(15); // waits 15ms for the servo to reach the position
  }
  for (pos = 180; pos >= 0; pos -= 1) { // goes from 180 degrees to 0 degrees
    myservo.write(pos); // tell servo to go to position in variable 'pos'
    delay(15); // waits 15ms for the servo to reach the position
  }
}
```

MOTORI IN CONTINUA

- Due tipi
 - brushless (passo passo)
 - senza spazzole
 - brushed
 - con spazzole, meno costosi (ma durano di meno)
 - corrente trasferita alla bobina (coil) mediante spazzole
 - quando la corrente passa nella bobina del rotore, genera un campo magnetico che è attratto o respinto dai magneti presenti nello statore
 - mediante le spazzole, viene ad essere invertita la polarità ogni mezzo giro, creando un momento angolare
- Sfruttando il medesimo principio, è possibile usare il motore come generatore di corrente (in direzione opposta)
 - ruotando il rotore a mano è possibile generare corrente per, ad esempio, illuminare un led
 - vedi dinamo

MOTORI IN CC - CONTROLLO

- Controllo sulla velocità
 - cambiando la tensione ai morsetti, cambiamo la velocità del motore
- Invertendo la tensione è possibile invertire il senso di rotazione
 - allo scopo a supporto si usa un circuito integrato chiamato **ponte H**

ESEMPIO IN ARDUINO: ALIMENTAZIONE SEPARATA

- In generale un motore DC può richiedere più corrente di quella fornita dalla parte di alimentazione di Arduino
 - inoltre può generare picchi di corrente in direzione opposta, in virtù del suo principio di funzionamento
- Per cui quando si usano motori DC è opportuno isolarli dal punto di vista dell'alimentazione da Arduino, per pilotarli senza problemi
- Nell'esempio che segue:
 - alimentiamo il motore con una batteria separata
 - il controllo di velocità avviene mediante un segnale PWM
 - usiamo un transistor come interruttore

Esempio tratto da [EA, cap. 4]

SCHEMA

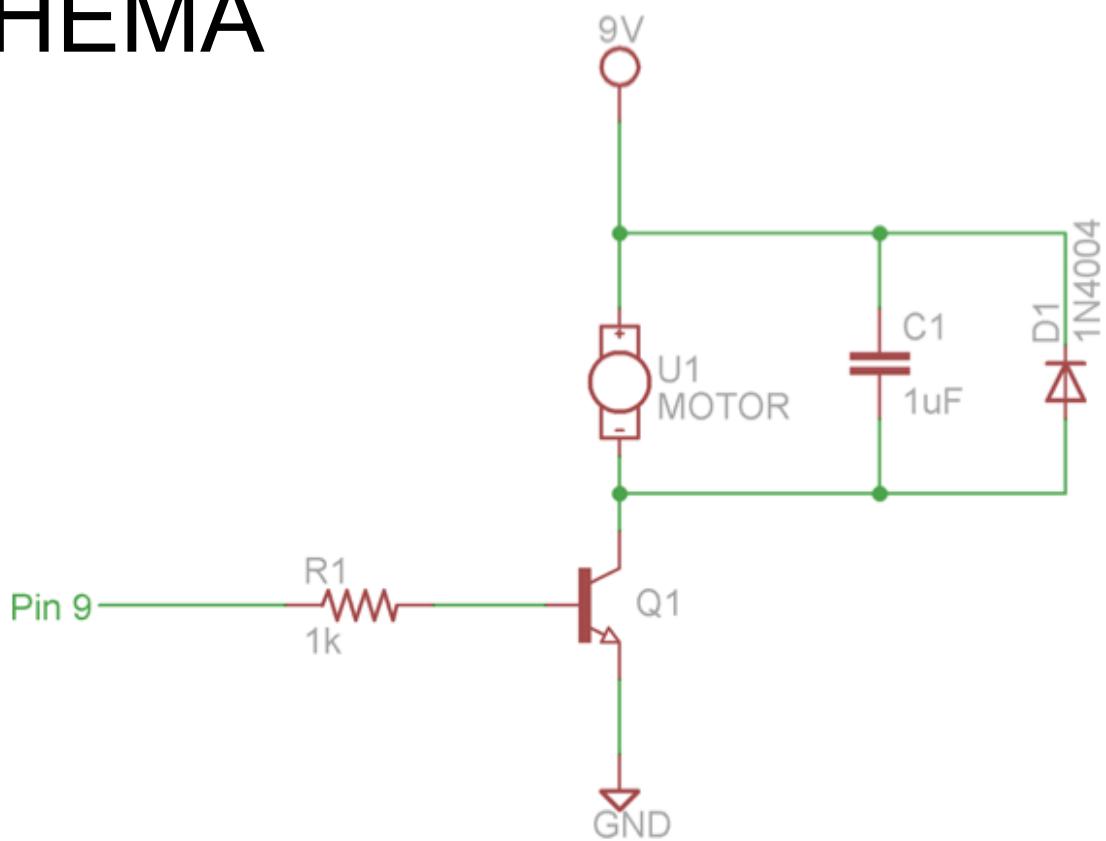


Image created with Eagle.

Componenti:

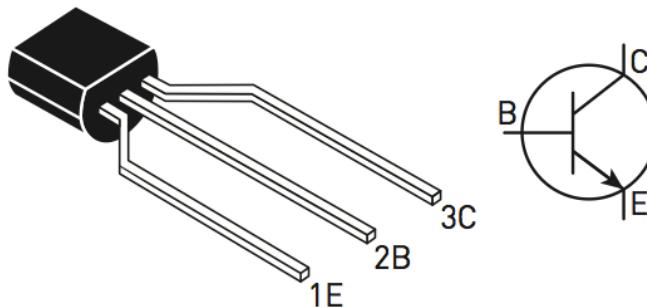
- Q1 è un transistor NPN BJT (Bipolar-junction transistor) usato come interruttore sull'alimentazione di 9V
- 1 KOhm usato per separare il morsetto base del transistore dal pin di controllo di Arduino
- U1 è il motore
- C1 è una condensatore usato per filtrare il rumore generato dal motore
- D1 è un diodo che protegge l'alimentazione dall'inversione di tensione causata dal motore

FUNZIONAMENTO

- La corrente fluisce dal collettore all'emettitore
- La modulazione avviene sul B (base) pin
- Quando una tensione sufficientemente elevata viene applicata alla base, il circuito si chiude, la corrente fluisce e il motore gira
- Se l'alimentazione viene rimossa in modo istantaneo (non graduale), allora l'energia contenuta nel motore viene dissipata in termini di picco di tensione inversa, che può danneggiare i componenti del circuito
- Per questo motivo si piazza un diodo di protezione (*diodo di ricircolo*) in parallelo al motore, che assicura che la corrente inversa generata dal motore fluisca nel diodo e che la tensione inversa non ecceda la tensione soglia di forward del diodo

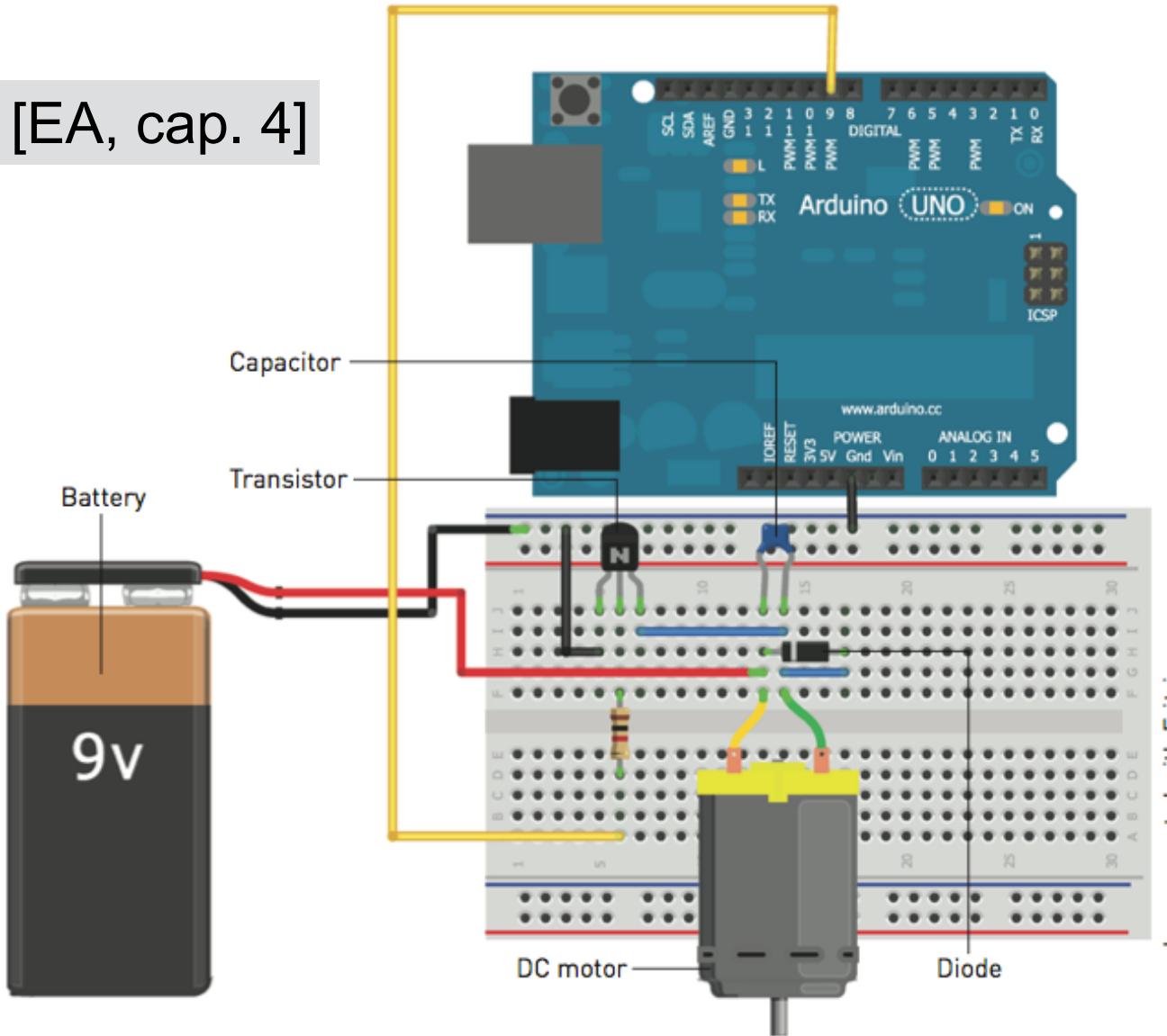
USO DI TRANSISTOR COME SWITCH

- In generale i transistor sono usati per una miriade di scopi
 - amplificatori, porte logiche, interruptori..
- In questo caso, modulando il pin base con un segnale PWM, si può controllare la velocità del motore - accendendo e spegnendo il transistor rapidamente
- Il duty cycle del segnale PWM determina la velocità del motore
 - duty cycle = 100% velocità massima (sempre acceso)
 - duty cycle = 0% velocità nulla (motore non alimentato)



SKETCH

[EA, cap. 4]



VERIFICHE DA FARE

- assicurarsi che, nonostante la doppia alimentazione, sia considerato lo stesso riferimento GND
- assicurarsi che la batteria di 9 V non sia connessa a quella da 5 V
- assicurarsi che l'orientamento del transistor sia corretto
- assicurarsi che l'orientamento del diodo sia corretto

Image created with Fritzing.

PROGRAMMAZIONE

- Il controllo della velocità angolare avviene inviando un segnale PWM con un certo duty cycle
 - duty cycle 100% = max velocità,
 - duty cycle 0% = velocità pari a zero
 - scala lineare.

```
const int MOTOR = 9;

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

void loop()
{
    for (int i = 0; i < 256; i++)
    {
        analogWrite(MOTOR,i);
        delay(10);
    }

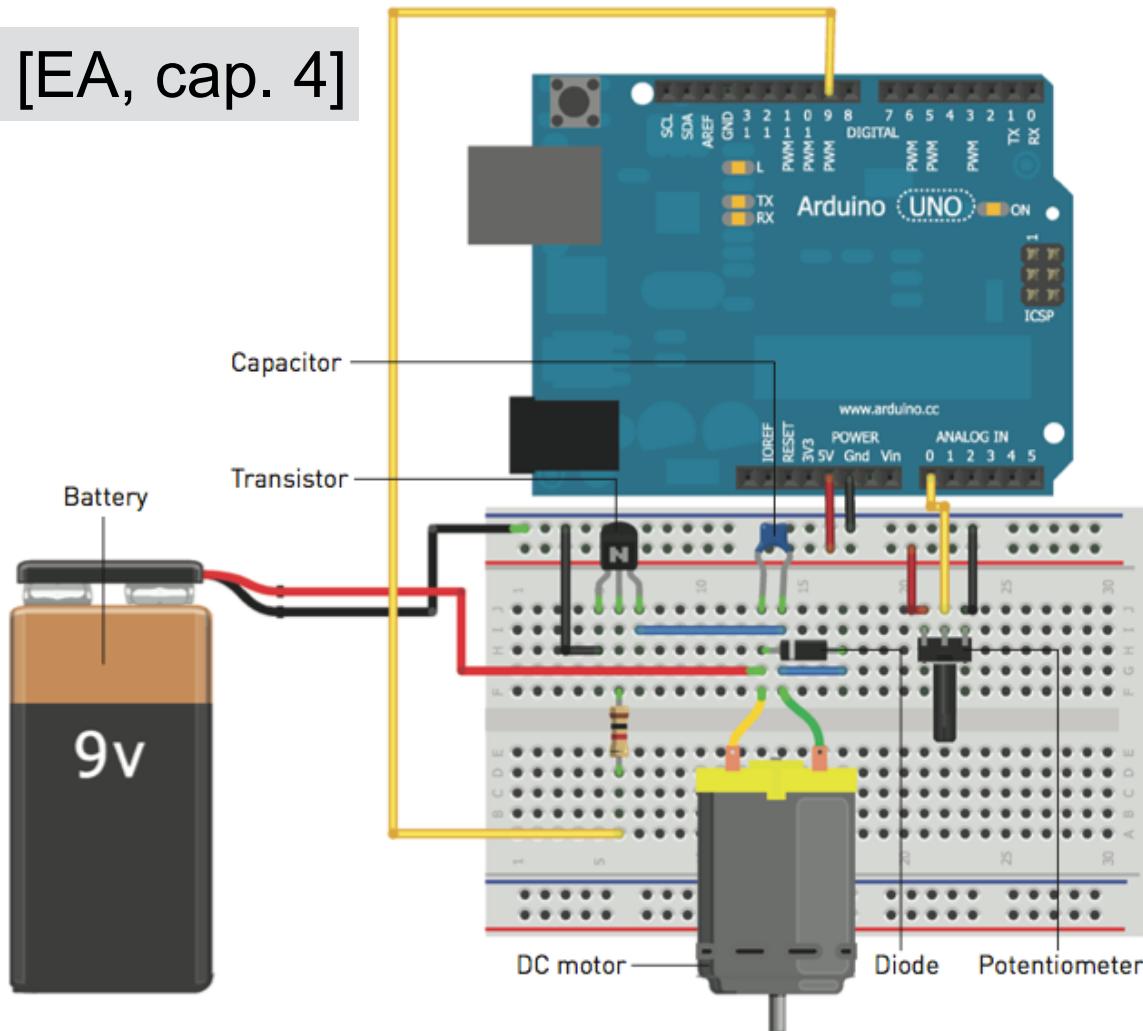
    delay(2000);

    for (int i = 255; i >= 0; i--)
    {
        analogWrite(MOTOR,i);
        delay(10);
    }

    delay(2000);
}
```

AGGIUNTA DI UN POTENZIOMETRO

[EA, cap. 4]



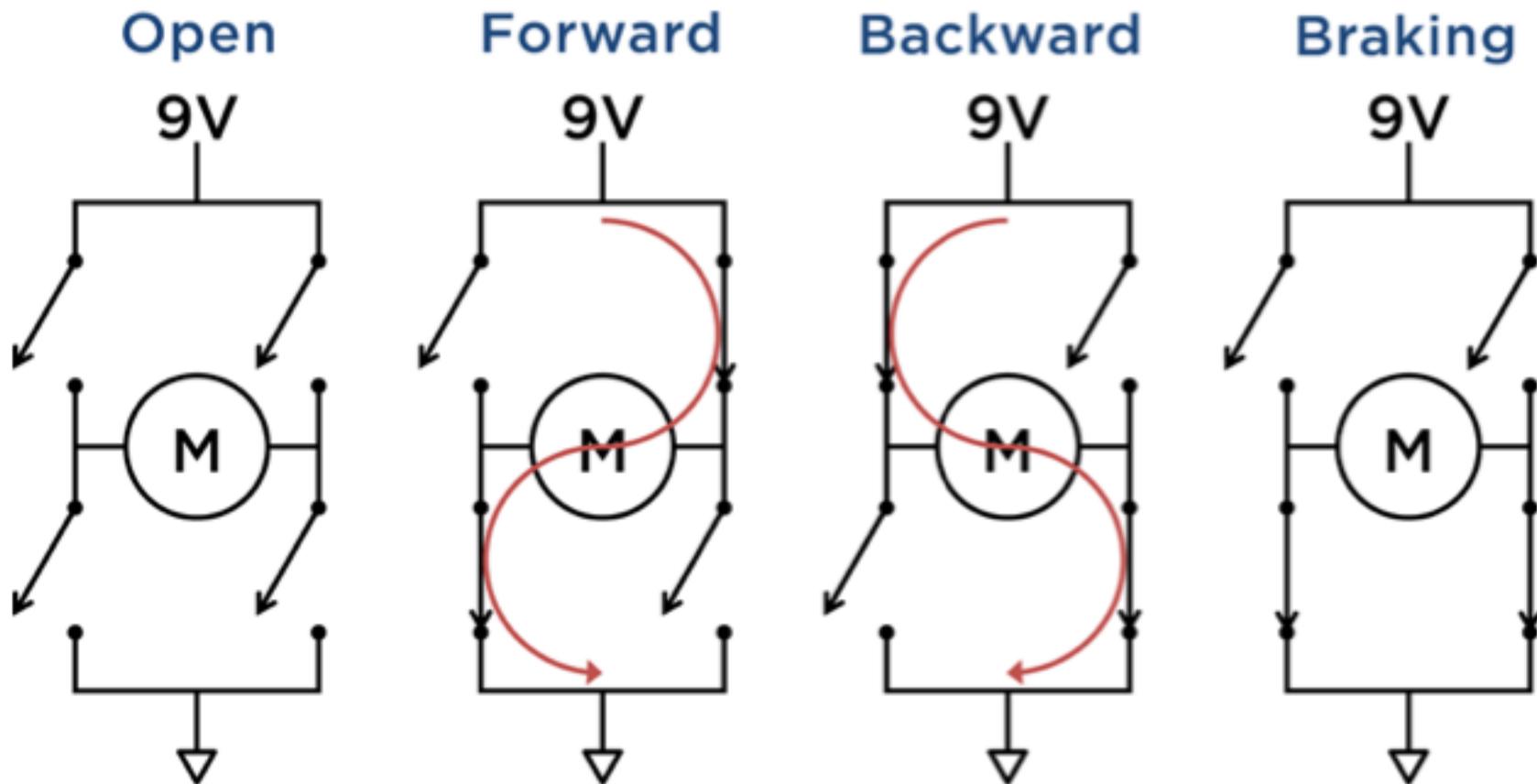
```
const int MOTOR = 9;  
const int POT = 0;  
  
int val = 0;  
  
void setup()  
{  
    pinMode(MOTOR, OUTPUT);  
}  
  
void loop()  
{  
    val = analogRead(POT);  
    val = map(val, 0, 1023, 0, 255);  
    analogWrite(MOTOR, val);  
}
```

Image created with Fritzing.

CONTROLLO DELLA DIREZIONE DEL MOVIMENTO CON PONTE H

- E' possibile invertire il senso di marcia nel motore invertendo la polarità della tensione applicata ai morsetti.
- Allo scopo nei circuiti si usa tipicamente un componente chiamato *ponte H*.
- Il ponte integra 4 switch (realizzati da transistor) e tutta la circuiteria di protezione.
- Ha 4 stati operazionali
 - **open**
 - tutti gli switch aperti e il motore non gira
 - **forward**
 - sono aperti due switch su diagonale opposta, la corrente fluisce e il motore si muove in una direzione
 - **braking**
 - tutto il movimento residuo causato dal momento angolare viene cessato e il motore si ferma
 - **backward**
 - sono aperti gli altri due switch su diagonale opposta, la corrente fluisce nel motore nella direzione opposta e il motore si muove nella direzione opposta

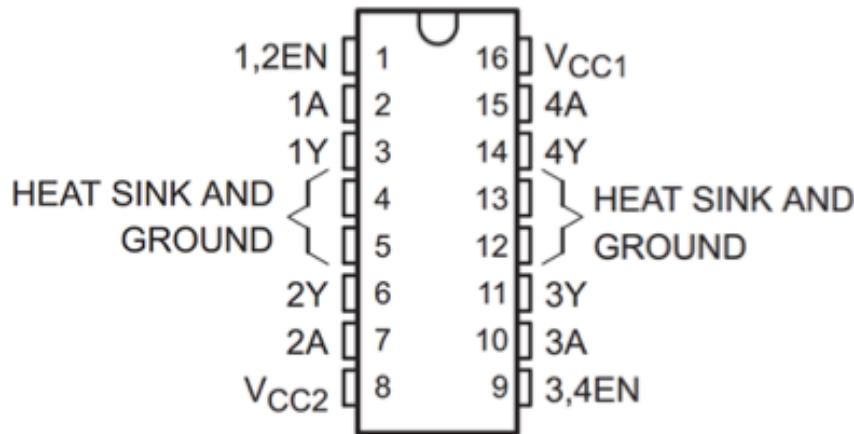
STATI OPERAZIONALI DEL PONTE-H



[EA, cap. 4]

ESEMPIO CON PONTE-H

- Esempio concreto: SN754410 Quanduple Half-H driver
 - permette di pilotare due motori



FUNCTION TABLE
(each driver)

INPUTS†		OUTPUT
A	EN	Y
H	H	H
L	H	L
X	L	Z

H = high-level, L = low-level

X = irrelevant

Z = high-impedance (off)

† In the thermal shutdown mode, the output is in a high-impedance state regardless of the input levels.

Image used with permission courtesy of
Texas Instruments.

- VCC2 => alimentazione per il motore - 9V
- VCC1 => alimentazione logica del circuito - 5V
- 1Y e 2Y - uscite per il motore di sinistra
- 1A e 2A - stato degli switch per il motore di sinistra, da connettere ai pin di I/O di Arduino
- 1,2EN - pin che serve per abilitare o disabilitare il motore di sinistra - da connettere al segnale PWM
- 3Y e 4Y - uscite per il motore di destra
- 3A e 4A - stato degli switch per il motore di destra, da connettere ai pin di I/O di Arduino
- 3,4EN - pin che serve per abilitare o disabilitare il motore di destra - da connettere al segnale PWM

PONTI-H E CORTO CIRCUITI

- Se entrambi gli switch a sinistra o a destra vengono chiusi, si crea un corto fra la batteria e terra
 - la batteria si riscalda velocemente e potrebbe in alcuni casi prender fuoco o esplodere
 - il ponte H può danneggiarsi, come tutte le altre parti del circuito
 - è il caso in cui una errata programmazione può creare corti e danni
- Il chip SN754410 ha un circuito di protezione interno per cui viene disabilitato non appena la temperatura supera un certo valore (built-in thermal shutdown)
 - in ogni caso, è opportuno fare molta attenzione a non creare questa situazione
- La pratica di programmazione da seguire per evitare problemi è quella di disabilitare il chip prima di cambiare lo stato di uno qualsiasi degli switch
- 3 control pin:
 - uno per controllare le due porte superiori
 - uno per controllare le due porte inferiori
 - uno per abilitare il circuito

SKETCH CON PONTE-H

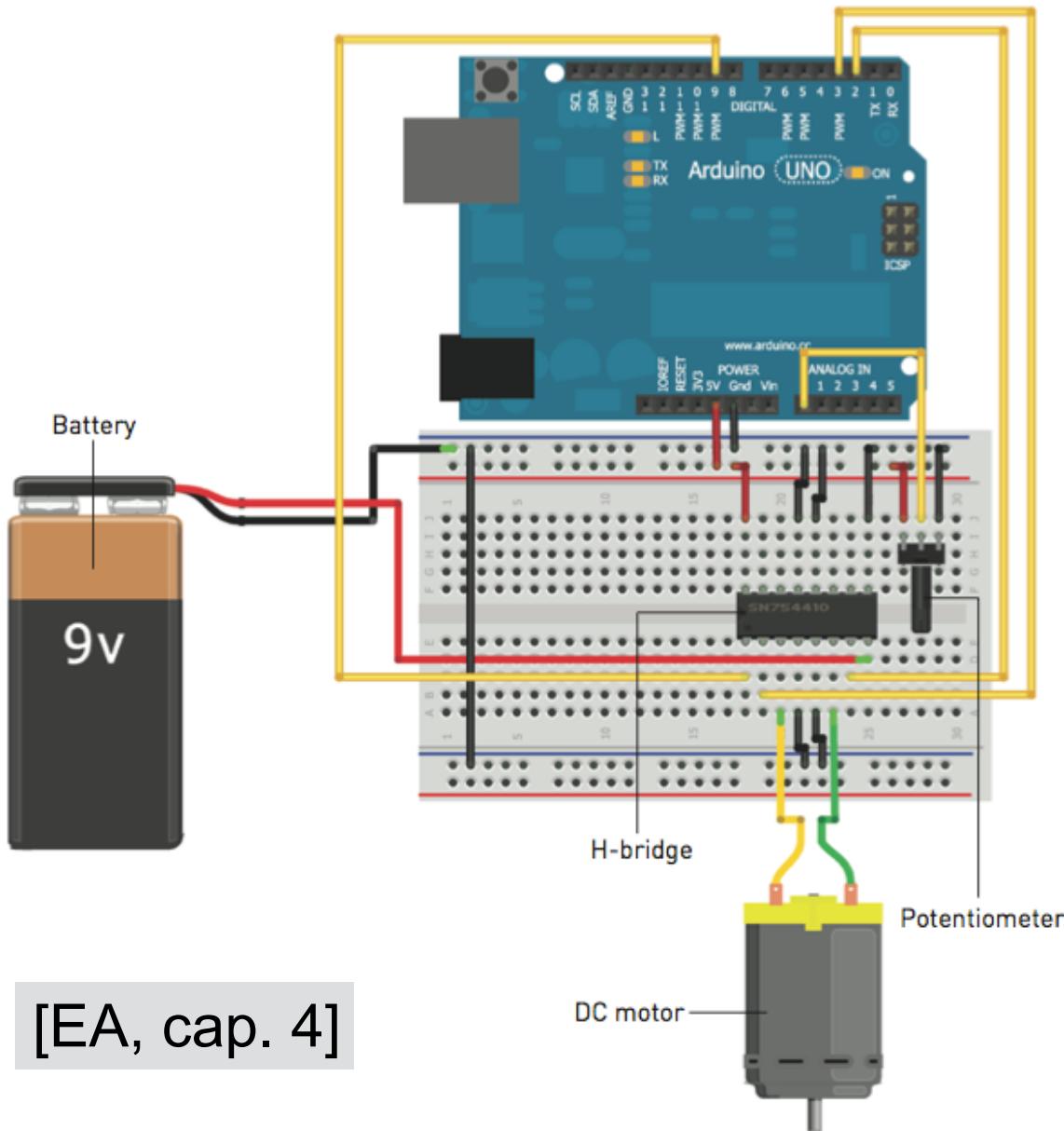


Image created with Circui

TORI



PROGRAMMA IN ARDUINO

- Estendiamo il precedente circuito con il potenziometro facendo in modo che
 - agli estremi porti il motore a velocità massima in una direzione o nell'altra, mentre a metà lo fermi

```
const int EN = 9; // Half bridge1 Enable
const int MC1 = 3; // Motor Control 1
const int MC2 = 2; // Motor Control 2
const int POT = 9; // POT on Analog pin 0

int val = 0; // memorizza valore potenziometro
int velocity = 0; // velocità desiderata (0-255)

void forward(int rate)
{
    digitalWrite(EN, LOW)
    digitalWrite(MC1, HIGH)
    digitalWrite(MC2, LOW)
    analogWrite(EN, rate)
}

void backward(int rate)
{
    digitalWrite(EN, LOW)
    digitalWrite(MC1, LOW)
    digitalWrite(MC2, HIGH)
    analogWrite(EN, rate)
}
```

```
void brake()
{
    digitalWrite(EN, LOW)
    digitalWrite(MC1, LOW)
    digitalWrite(MC2, LOW)
    digitalWrite(EN, HIGH)
}

void loop()
{
    val = analogRead(POT);

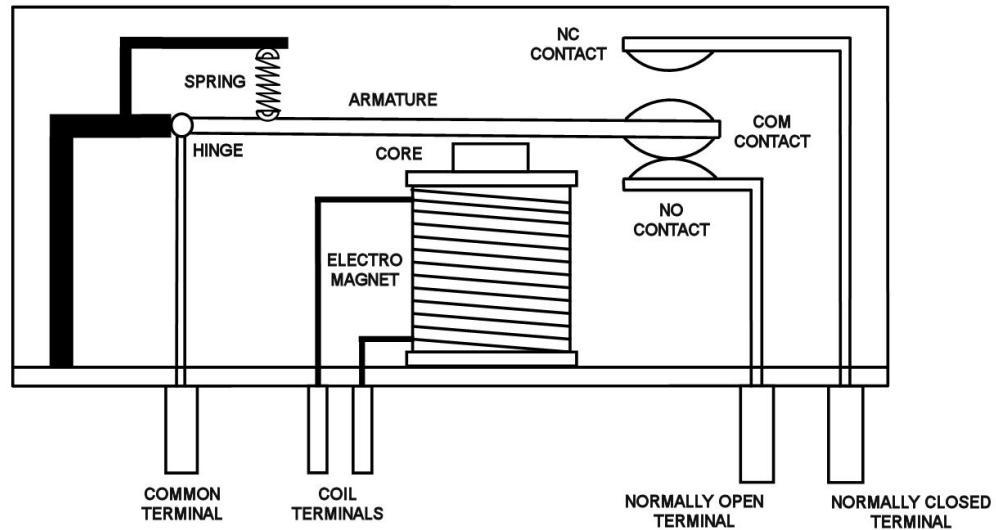
    if (val > 562){
        velocity = map(val, 563, 1023, 0, 255);
        forward(velocity);
    } else if (val < 462) {
        velocity = map(val, 0, 461, 0, 255);
        forward(velocity);
    } else {
        brake();
    }
}
```

PILOTAGGIO DI CIRCUITI ESTERNI

- Questi tipi di dispositivi permettono di **pilotare circuiti esterni** - in cui ad esempio è necessario usare tensioni e correnti elevate - mantenendo totale disaccoppiamento/isolamento dal punto di vista delle correnti che circolano nel circuito
- Esempi
 - **relay (relè)**
 - permette di aprire e chiudere il secondo circuito mediante l'azione di un elettro-magnete, pilotato dal primo circuito
 - esempi in Arduino:
 - <http://www.instructables.com/id/Arduino-Controlled-Relay-Box/>
 - **fotoaccoppiatori**
 - basati su un led interno, pilotato dal circuito a bassa tensione
 - quando il led viene illuminato, attiva una fotocellula e quindi l'interruttore del secondo circuito si chiude

SUI RELAY SWITCH

- Internamente un relay contiene due circuiti elettricamente disaccoppiati, in cui uno (ove circolano correnti basse e ci sono tensioni ridotte) può pilotare/ commutare l'altro (in cui tipicamente circolano correnti anche elevate (30mA per relay a 12 V) e ci sono tensioni significative (220 V AC)



- le bobine usata internamente nell'elettro-magnete possono generare picchi di tensione quando sono disattivate che possono danneggiare transistori e microcontrollori collegati
 - per cui è necessario utilizzare opportuni diodi in parallelo ai relays, come nel caso dei motori

ESEMPIO DI COLLEGAMENTO

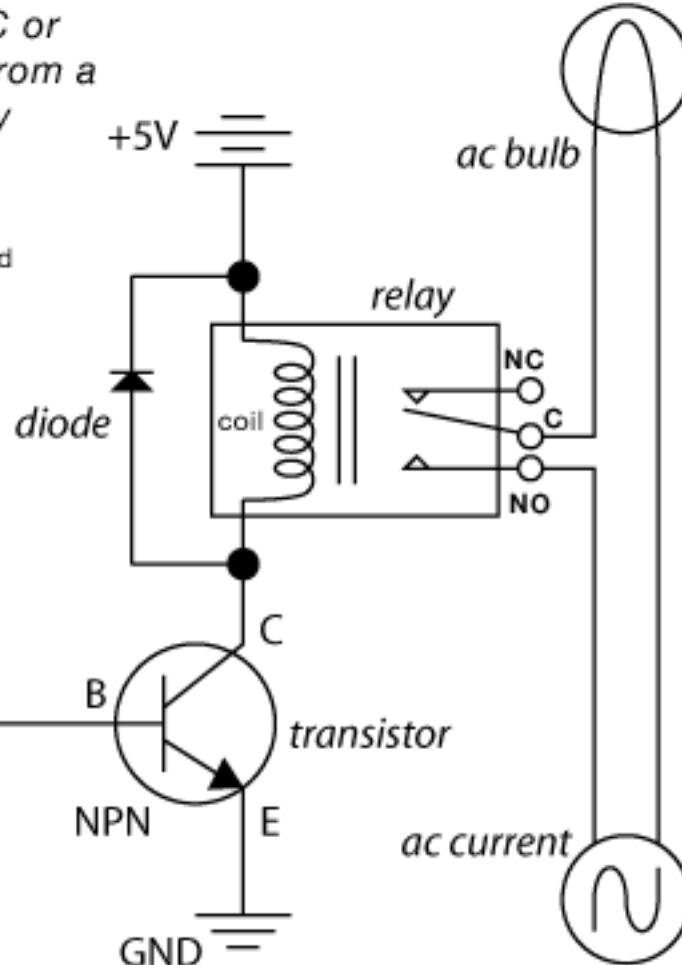
circuit for controlling an AC or other high-current device from a microtroller by using a relay

relay:
NC = normally closed
C = common
NO = normally open

i/o pin

100-1K ohms
resistor

transistor:
B = base
C = collector
E = transmitter



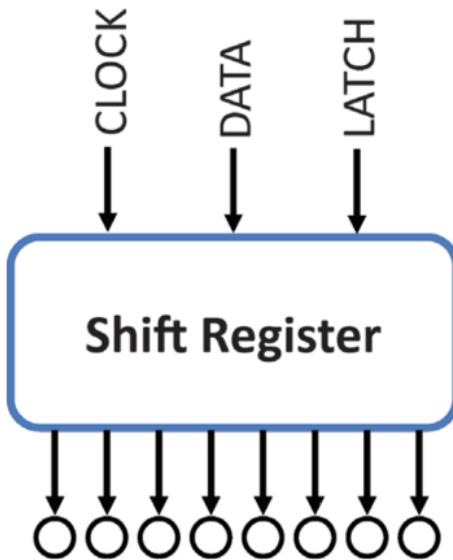
**QUANDO I PIN NON
BASTANO...**

QUANDO I PIN NON BASTANO...

- Problema ricorrente in progetti:
 - numero di pin insufficiente per interfacciare tutti i dispositivi previsti
- Possibili soluzioni
 - usare schede microcontrollori con numero di pin adeguato
 - es: Arduino Mega - 54 pin I/O
 - usare dispositivi come **registri a scorrimento** (*shift registers*)

SHIFT REGISTER

- In generale sono dispositivi utili per convertire input sequenziale/parallelo in output parallelo/sequenziale
- La conversione (seriale => parallelo) è fatta mediante **registri SIPO** (Serial In Parallel Out)
 - dispositivi che accettano in ingresso uno stream sequenziale di bit e fornisce in uscita l'output di tali bit come porta di I/O parallela
 - il numero di bit in uscita può variare - esempio: a 8, 16, 32 ..
 - possono essere concatenati facilmente nello stesso circuito



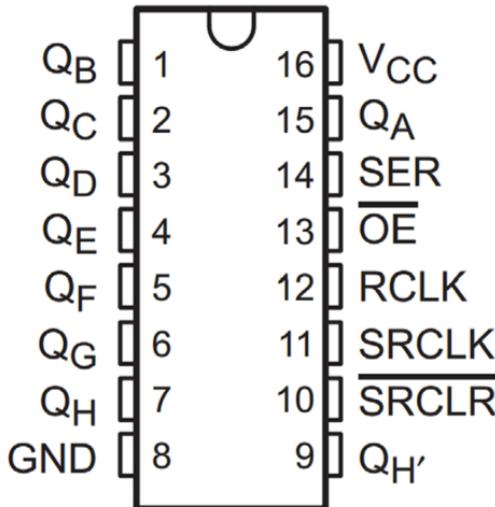
([EA], p. 147)

Segnali in input:

- segnale CLOCK - segnale che sincronizza l'acquisizione dei dati in input
- segnale DATA - pin dove insiste lo stream di bit in input
- segnale LATCH - segnale che campiona l'output e lo rende disponibile (register clock pin)

ESEMPIO

- Esempio concreto: shift register 74HC595

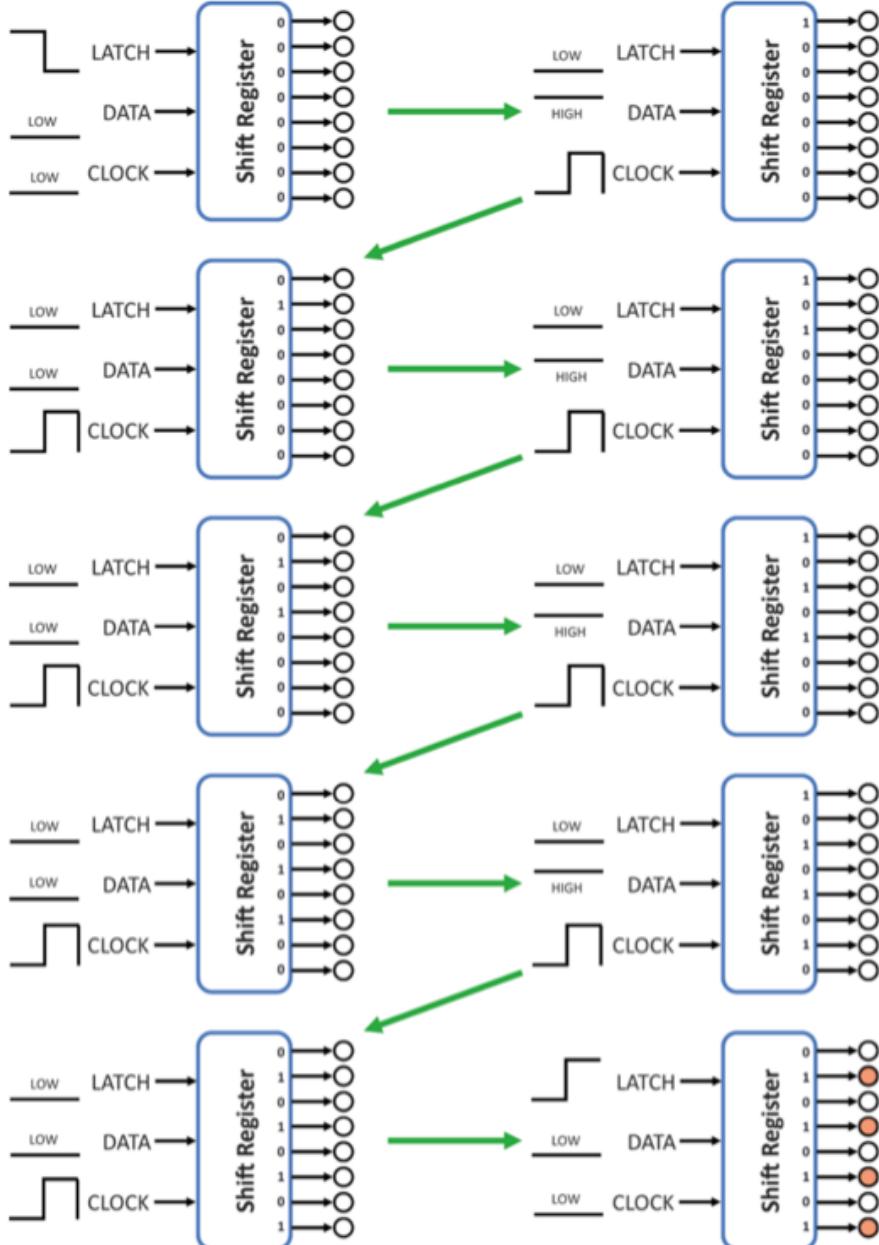


Credit: Image used with permission courtesy of Texas Instruments,
www.ti.com.

- Pin out:
 - SER è la linea dati,
 - SRCLK è il segnale CLOCK
 - QA - QH - output parallelo rappresentato da 8 bit/pin
 - RCLK è il segnale LATCH
- Gli altri:
 - OE (output enable) segnato, attivo basso - attiva o disattiva l'output.
 - SRCLK segnato - serial clear pin: quando attivo (basso), resetta il contenuto del registro.

FUNZIONAMENTO

- Device sincrono, che acquisisce il valore dei bit in input al fronte di salita del segnale del clock
 - ad ogni fronte, tutti i valori al momenti presenti nel registro vengono fatti scorrere a sinistra di una posizione e viene introdotto il nuovo bit
 - il pin LATCH è attivato alla fine dell'acquisizione
 - esempio
 - caricamento valore 10101010



ESEMPIO IN ARDUINO

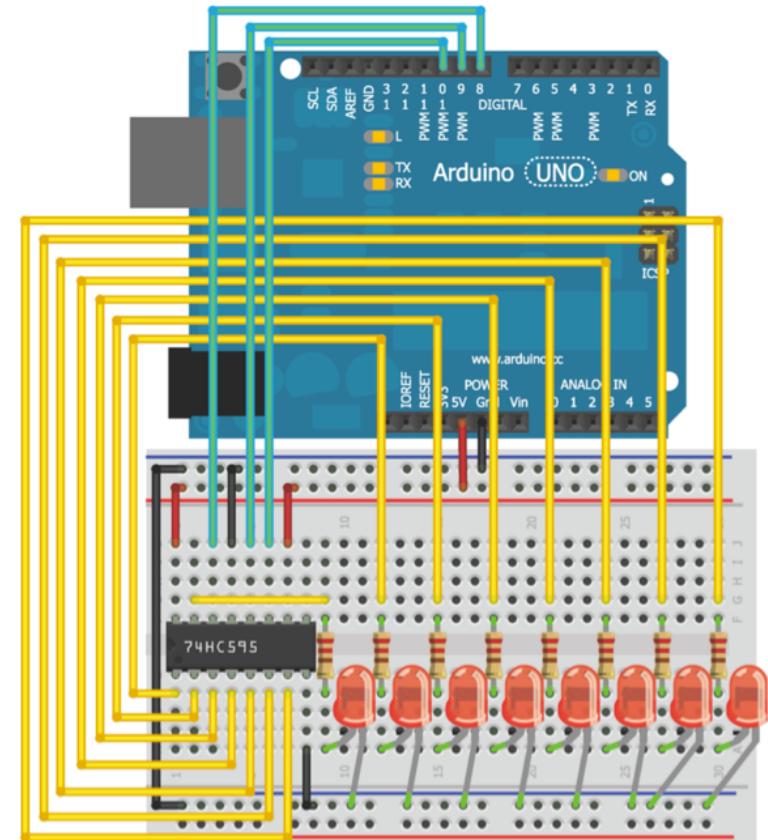
- Pilotaggio 8 led in Arduino, mediante 3 segnali
- Disponibile funzione di libreria shiftOut che si occupa di inviare un byte ad un I/O pin arbitrario
 - shiftOut (DATA_PIN, CLOCK_PIN, MSBFIRST/LSBFIRST, VALUE)

```
const int SER = 8;
const int LATCH = 9;
const int CLK = 10;

void setup()
{
    pinMode(SER,OUTPUT);
    pinMode(LATCH,OUTPUT);
    pinMode(CLK,OUTPUT);

    digitalWrite(LATCH,LOW);
    shiftOut(SER,CLK, MSBFIRST, B10101010);
    digitalWrite(LATCH,HIGH);
}

void loop(){}
```



COLLEGAMENTI IN DAISY-CHAIN

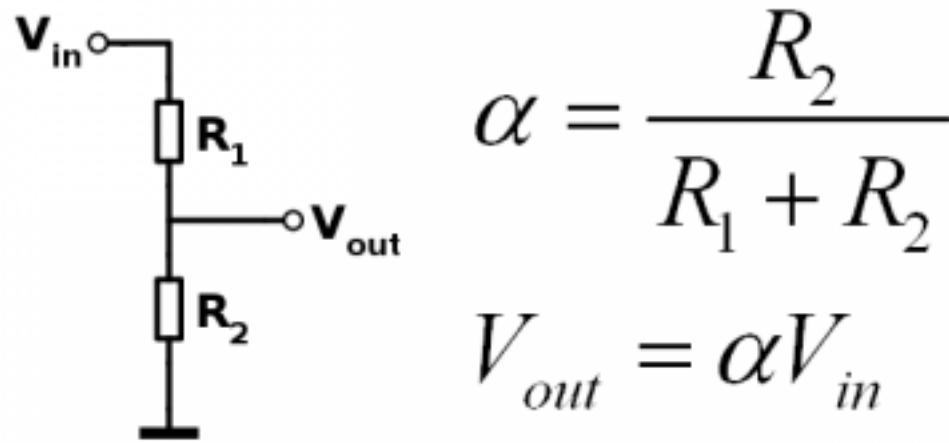
- Per pilotare un numero maggiore di uscite usando più shift register è possibile mediante *collegamenti daisy-chain*
- Il collegamento in daisy chain si ottiene:
 - collegando il pin Q_H' al pin DATA di un altro shift register
 - il pin Q_H' contiene il valore del bit più significativo che ad ogni shift “esce”, facendo posto a quello meno significativo
 - facendo in modo che tutti gli shift register coinvolti condividano i segnali LATCH e CLOCK
- Nel caso di un numero elevato di led è necessario disporre di alimentazione supplementare, oltre a quella fornita da Arduino

ALCUNE NOTE SULL'INTERFACCIAZIONE DEI DISPOSITIVI

PROBLEMA DEI LIVELLI DI TENSIONE

- I sensori/attuatori per sistemi embedded possono funzionare con livelli di tensione diversa
 - ad esempio: 5 Volt (Arduino) e 3.3 Volt (Raspberry Pi)
- Nel caso di utilizzo di componenti che prevedono livelli di tensione inferiori rispetto a quelli forniti dalla scheda/sistema, occorre adottare opportuni accorgimenti per evitare di danneggiare i componenti stessi
 - vale anche nell’interfacciamento diretto fra schede, ad esempio collegamento seriale diretto Arduino-Raspberry
 - TX (Raspi) => RX (Arduino): nessun problema
 - il livello 3.3V è sufficiente per identificare il valore logico alto in Arduino
 - TX (Arduino) => RX (Raspberry): problema!
 - il livello 5V in ingresso a Raspi può seriamente danneggiare la scheda
- Un modo per adattare livelli di tensione è utilizzare circuiti chiamati **partitori di tensione**

PARTITORE DI TENSIONE



- Serve per produrre in uscita una tensione V_{out} voluta (es: 3.3V), a partire da una tensione in ingresso (es: 5V)
- E' importante il rapporto fra i valori delle resistenze, non il loro valore assoluto
- Il valore assoluto determina però la corrente che scorre nel circuito
 - $I_1 = I_{out} + I_2$, $I_{out} = I_1 - I_2$

ESEMPI DI UTILIZZO

- Interfacciamento seriale diretto fra Arduino e Raspi
- Interfacciamento Arduino con alcuni moduli Bluetooth

BIBLIOGRAFIA

- [IES] Introduction to Embedded Systems - A Cyber-Physical Approach. Lee, Seshia
- Fondamenti della Misurazione. E.Bava et al. Esculapio
- [EA] “*Exploring Arduino*”, Jeremy Bloom, Wiley
- “*Make: Sensor - Projects and Experiments to measure the world with Arduino and Raspberry Pi*” (Karvinen et al). Maker-Media

Sistemi Embedded e IoT

Ingegneria e Scienze Informatiche - UNIBO

a.a 2020/2021

Docente: Prof. Alessandro Ricci

[modulo-2.1]

SISTEMI EMBEDDED E MODELLAZIONE OO

SOMMARIO

- Obiettivo di questo modulo è introdurre alcuni concetti di base nella modellazione di sistemi software embedded, utilizzando paradigmi di alto livello visti nel percorso triennale, a partire da quello ad oggetti

PROGETTAZIONE E SVILUPPO DI SISTEMI SOFTWARE EMBEDDED

- Due approcci complementari da integrare
 - **approccio top-down**
 - partiamo dal *dominio* e dalla sua modellazione con paradigmi di alto livello (es: OO)
 - **approccio bottom-up**
 - partiamo dalle caratteristiche a livello hardware e dal comportamento che il sistema deve avere a questo livello
- In questo modulo consideriamo l'approccio *top-down*

MODELLAZIONE E PROGRAMMAZIONE

- Paradigmi di programmazione sono anche anzitutto paradigmi di **modellazione**
 - in che modo concettualizzo, rappresento, progetto un sistema software
 - in che modo analizzo e formulo la soluzione di un problema in certo dominio applicativo
- **Modello**
 - rappresentazione degli aspetti salienti del sistema, astraendo da quelli che non sono significativi
 - gli aspetti salienti concernono la struttura, il comportamento, le interazioni che il sistema deve avere al fine di ottenere le funzionalità volute
- Stretto rapporto modelling e programming [OLE]
 - scuola scandinava, dove ha le radici l'OO

Ole Lehrmann Madsen, Birger Møller-Pedersen. *A Unified Approach to Modeling and Programming*, Model Driven Engineering Languages and Systems. Lecture Notes in Computer Science Volume 6394, 2010, pp 1-15.

IMPORTANZA DELLA MODELLAZIONE

- Analizzare il problema e definire soluzioni più astratte rispetto alla pura implementazione e ai relativi linguaggi
 - aspetto centrale dell'ingegneria
- Vantaggi
 - miglior “comprensione” del funzionamento del sistema e gestione più efficace delle relative complessità
 - riusabilità e portabilità
 - estendibilità
 - possibilità di avere forme di analisi più rigorosa correttezza e verificabilità

DIMENSIONI FONDAMENTALI DI MODELLAZIONE

- **Struttura**
 - come sono organizzate le varie parti
- **Comportamento**
 - il comportamento computazionale di ogni singola parte
- **Interazione**
 - come interagiscono le varie parti

PARADIGMI E LINGUAGGI

- ***Paradigmi di modellazione***
 - definiscono un insieme coerente di concetti e i principi con cui definire i modelli
 - esempio: **paradigma ad oggetti**
 - spesso correlati ai paradigmi *di programmazione*
 - es: OOP
- ***Linguaggi di modellazione***
 - forniscono un modo rigoroso non ambiguo per rappresentare i modelli
 - esempio: **linguaggio UML**
 - spesso correlati a paradigmi di modellazione specifici
 - esempio UML è fortemente ispirato al paradigma OO, in particolare per la parte strutturale e per diagramma delle interazioni

PARADIGMA OBJECT-ORIENTED

- Il paradigma ad oggetti è fra i paradigmi di riferimento nella modellazione e progettazione del software
 - livello di astrazione efficace per catturare aspetti essenziali del dominio, in particolare aspetti strutturali
- Proprietà importanti a livello di design e programmazione del software
 - **modularità**
 - **incapsulamento**
 - meccanismi per **riuso** ed **estendibilità**
- Aspetti non direttamente catturati dal paradigma
 - concorrenza, interazioni asincrone, distribuzione

MODELLAZIONE SOFTWARE SU MICRO-CONTROLLORE

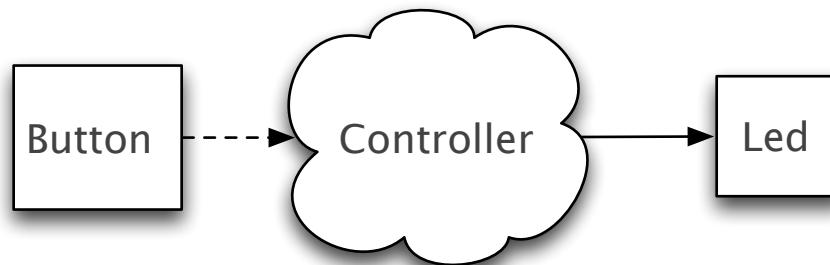
- Due macro parti di base
 - **controllore**
 - incapsula la logica di controllo per lo svolgimento del compito o dei compiti del sistema
 - allo scopo usa/osserva/gestisce risorse
 - es: sensori, attuatori
 - modello tipicamente attivo
 - **elementi controllati**
 - modellano le risorse gestite/utilizzate dal controllore per svolgere il compito
 - ad esempio: dispositivi di I/O, sensori, attuatori
 - incapsulano funzionalità utili al controllore
 - modello tipicamente passivo

CARATTERISTICHE

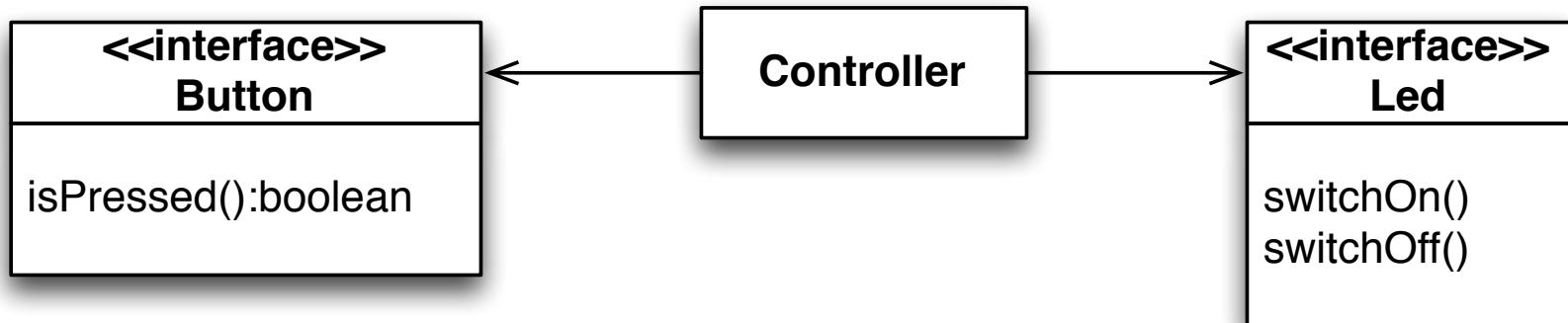
- **Controllore**
 - integrazione pro-attività / re-attività
 - modello a *loop di controllo*
 - architettura più semplice: super-loop
 - nei moduli futuri vedremo architetture a loop più articolate
- **Elementi controllati**
 - interfaccia ben definita
 - stato/eventi osservabili
 - riusabilità, controllabilità
 - modello OO

ESEMPIO BUTTON-LED

- Modellazione sistema *button-led*
 - costituito da un pulsante che funge da interruttore e un led come output
 - il controllore (software) deve accendere il led quando l'interruttore è premuto e spegnerlo quando l'interruttore non premuto



UN MODELLO OO DEL BUTTON-LED



- Interfacce
 - Button
 - `isPressed(): boolean`
 - Led
 - `switchOn()`
 - `switchOff()`

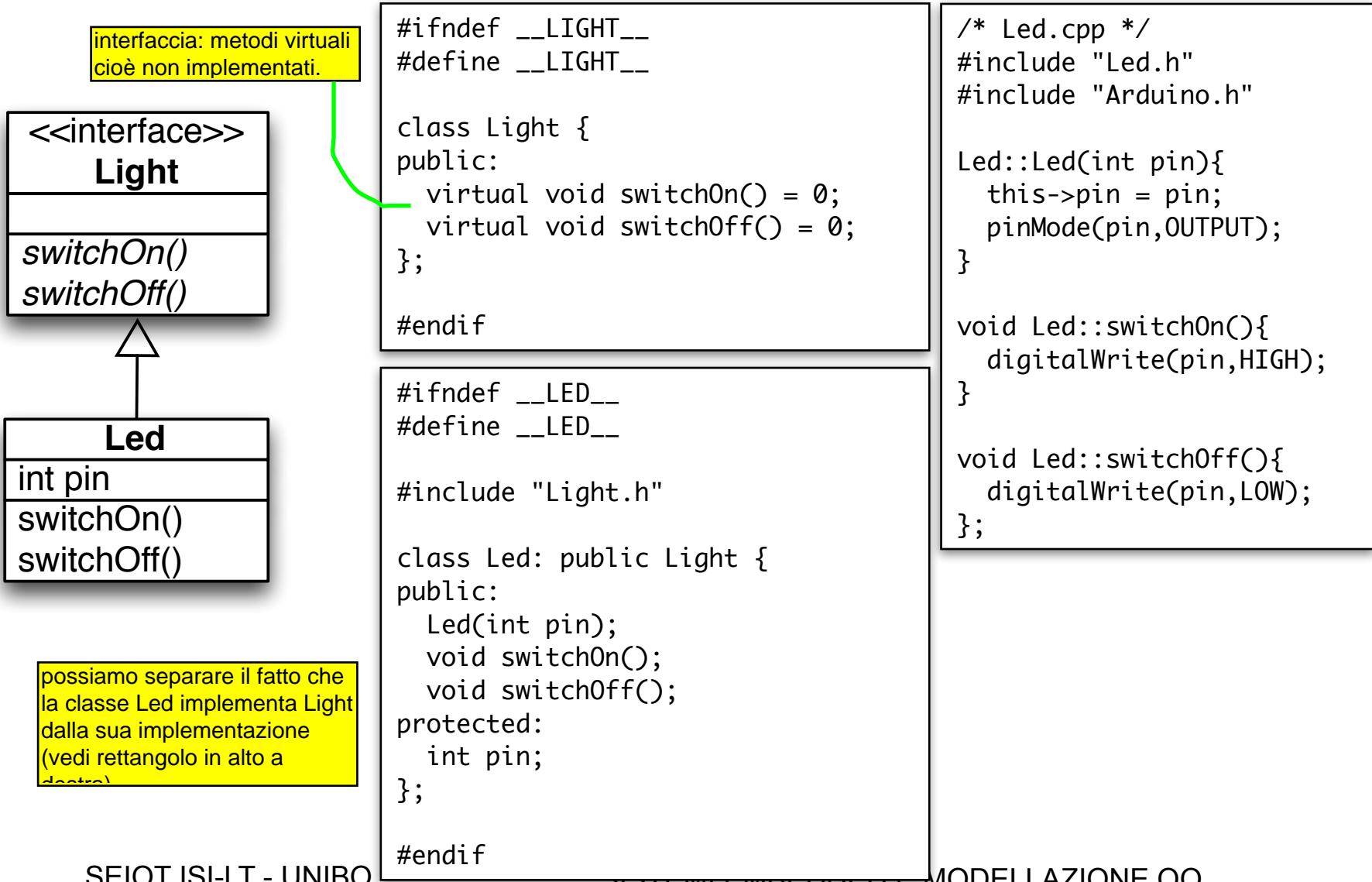
IL CONTROLLORE - PSEUDOCODICE

- Loop di controllo

```
lightOn := false

loop {
    if (!lightOn && button.isPressed()){
        light.switchOn()
        lightOn := true
    } else if (lightOn && !button.isPressed()){
        light.switchOff()
        lightOn := false
    }
}
```

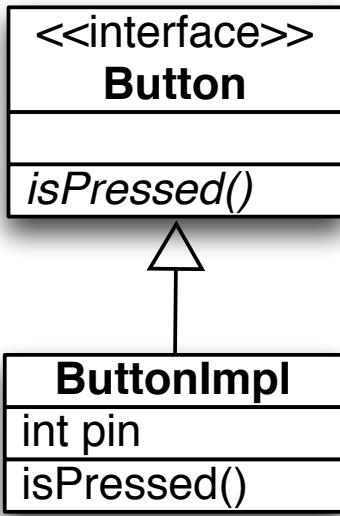
DAL MODELLO AL CODICE: ESEMPIO IN ARDUINO IN WIRING/C++



DAL MODELLO AL CODICE: ESEMPIO IN ARDUINO IN WIRING/C++

- Light (in Light.h)
 - rappresenta l'interfaccia (utile per qualsiasi tipo di dispositivo “luce”), rappresentata da una classe astratta
- Led (in Led.h)
 - tipo specifico di Light
 - rappresenta la dichiarazione della classe concreta Led
- L'implementazione di Led è inclusa in un file separato Led.cpp
 - questo per separare specifica/signature da implementazione
 - notare l'inclusione di “Arduino.h” dove sono dichiarate procedure e costanti di base di Arduino

DAL MODELLO AL CODICE: ESEMPIO IN ARDUINO IN WIRING/C++



```
#ifndef __BUTTON__
#define __BUTTON__

class Button {

public:
    virtual bool isPressed() = 0;
};

#endif
```

```
#ifndef __BUTTONIMPL__
#define __BUTTONIMPL__

#include "Button.h"

class ButtonImpl: public Button {
public:
    ButtonImpl(int pin);
    bool isPressed();

protected:
    int pin;
};
#endif
```

```
#include "ButtonImpl.h"
#include "Arduino.h"

ButtonImpl::ButtonImpl(int pin){
    this->pin = pin;
    pinMode(pin, INPUT);
}

bool ButtonImpl::isPressed(){
    return digitalRead(pin) == HIGH;
}
```

DAL MODELLO AL CODICE: ESEMPIO IN ARDUINO IN WIRING/C++

- Button
 - interfaccia base pulsante
- ButtonImpl
 - rappresenta una classe di pulsanti concreti

DAL MODELLO AL CODICE: ESEMPIO IN ARDUINO IN WIRING/C++

- Setup
 - creazione oggetti
- Main (super-)loop

```
#include "Led.h"
#include "ButtonImpl.h"

#define LED_PIN    13
#define BUTTON_PIN 2

Light* light;
Button* button;
boolean lightOn;

void setup(){
    light = new Led(LED_PIN);
    button = new ButtonImpl(BUTTON_PIN);
    lightOn = false;
}

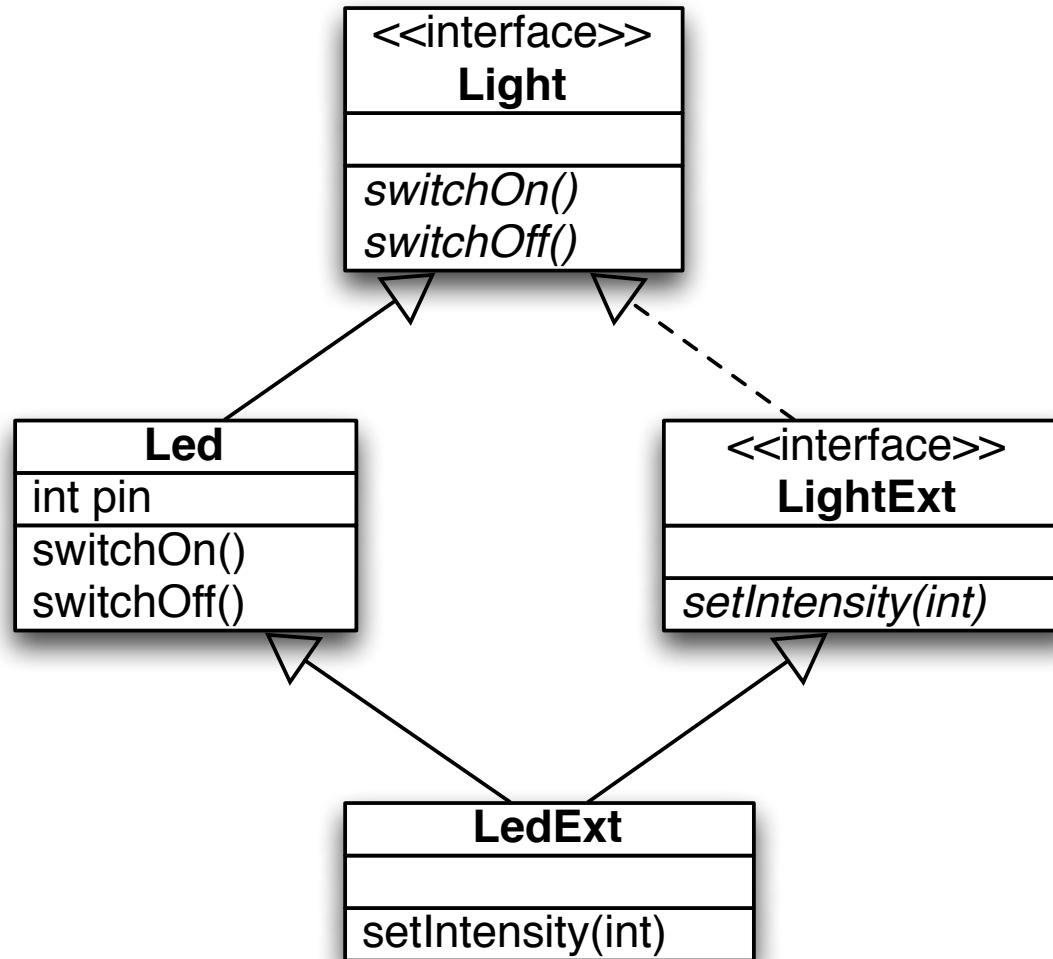
void loop(){
    if (!lightOn && button->isPressed()){
        light->switchOn();
        lightOn = true;
    } else if (lightOn && !button->isPressed()){
        light->switchOff();
        lightOn = false;
    }
};
```

NOTE

- Aspetti di basso livello (es: uso primitive di wiring per pilotare pin) sono “nascosti” nell’implementazione, non esposti a cui usa gli oggetti (cioè il controller)
 - maggiore leggibilità, riusabilità e portabilità del codice

ESEMPIO DI ESTENSIONE

- Modellazione led con intensità variabile



ESEMPIO DI ESTENSIONE

- Modellazione led con intensità variabile

```
#ifndef __LIGHT_EXT__
#define __LIGHT_EXT__
#include "Light.h"

class LightExt : public Light {
public:
    virtual void setIntensity(int) = 0;
};
#endif
```

```
#ifndef __LED_EXT__
#define __LED_EXT__
#include "Led.h"
#include "LightExt.h"

class LedExt: public LightExt, public Led {
public:
    LedExt(int pin);
    LedExt(int pin, int intensity);
    void switchOn();
    void switchOff();
    void setIntensity(int v);
private:
    int currentIntensity;
    bool isOn;
};
#endif
```

```
#include "LedExt.h"
#include "Arduino.h"

LedExt::LedExt(int pin) : Led(pin) {
    currentIntensity = 128;
    isOn = false;
}
LedExt::LedExt(int pin, int intensity) : Led(pin) {
    isOn = false;
    currentIntensity = intensity;
}

void LedExt::switchOn(){
    analogWrite(pin, currentIntensity);
    isOn = true;
}
void LedExt::setIntensity(int value){
    currentIntensity = value;
    if (isOn){
        analogWrite(pin, currentIntensity);
    }
}
void LedExt::switchOff(){
    analogWrite(pin, 0);
    isOn = false;
}
```

FADING

```
#include "LedExt.h"

// the pin must by a PWM one
#define LED_PIN 9

LightExt* led;
int brightness;
int fadeAmount;

void setup(){
    brightness = 0;
    fadeAmount = 5;
    led = new LedExt(LED_PIN,brightness);
    led->switchOn();
}

void loop(){
    led->setIntensity(brightness);
    brightness = brightness + fadeAmount;

    if (brightness == 0 || brightness == 255) {
        fadeAmount = -fadeAmount ;
    }

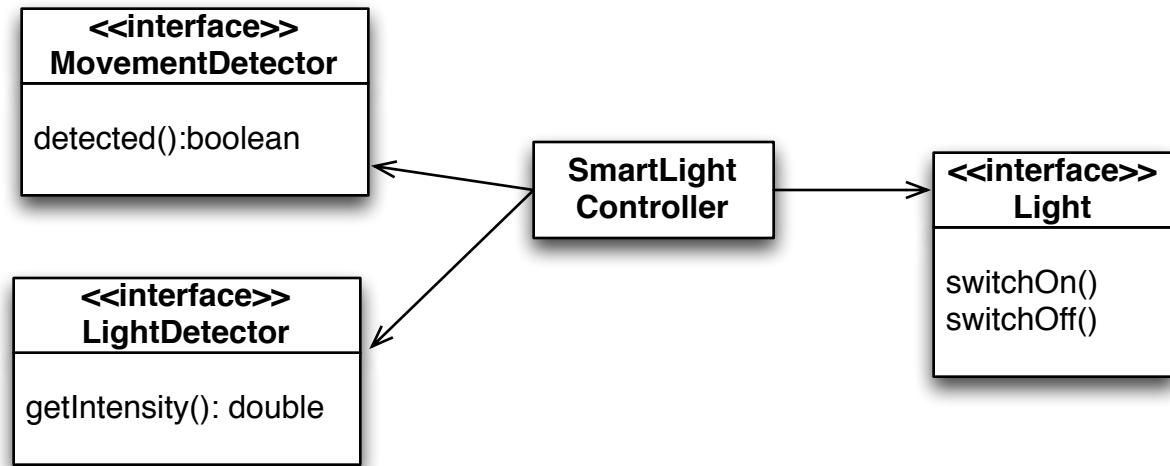
    delay(20);
};
```

NOTE

- La nuova classe LedExt può essere ri-usata anche negli esempi precedenti, senza dover cambiare il codice

DA BUTTON-LED A SMART LIGHT

- Come semplice variante, consideriamo un ambiente dotato di una luce in grado di accendersi o spegnersi a seconda della presenza o meno di persone. In più, la luce deve accendersi solo se l'ambiente non è già sufficientemente luminoso.
- Evoluzione del sistema precedente
 - Light (output)
 - MovementDetector (input)
 - LightDetector (input)
 - SmartLightController



SMART LIGHT CONTROLLER

- Loop di controllo

```
lightOn := false

loop {
    detected := presDetector.detected()
    intens := lightSensor.getIntensity()
    isLowIntens := intens < LIGHT_THRESHOLD
    if (!lightOn && detected && isLowIntens){
        light.switchOn()
        lightOn := true
    } else if (lightOn && (!isLowIntens || !detected)){
        light.switchOff()
        lightOn := false
    }
}
```

MODELLO A LOOP E MACCHINE A STATI FINITI

- Modello a loop
 - ad ogni ciclo il controller legge gli input di cui necessita e sceglie le azioni da compiere
 - la scelta dipende dallo *stato* in cui si trova
 - l'azione può cambiare anche lo stato interno
- Necessità di linguaggi/(meta-)modelli più astratti e rigorosi che non lo pseudo-codice
 - **macchine a stati finiti (FSM)**
 - sincrone, asincrone
 - introdotte e discusse nei prossimi moduli

MODELLO A LOOP: EFFICIENZA E REATTIVITÀ

- **Efficienza**
 - negli esempi visti, nonostante a livello logico il controllore sia in idle in attesa di percepire input, con questo approccio continua ad eseguire cicli, anche quando l'input non è cambiato
- **Reattività**
 - la reattività del controllore dipende da quanto rapidamente viene completato un ciclo
 - se durante l'esecuzione del ciclo avvengono ripetute variazioni di stato nei sensori, tali variazioni non vengono rilevate..

SULLA REATTIVITÀ: ESEMPIO

- “Realizzare un sistema di gestione di una luce con fade in e fade out continuo, ciascuno della durata di DT secondi, con pulsante tattile che ne regola l'accensione (la partenza) e l'interruzione (lo stop)”
- Esempio di soluzione

```
running := false
loop {
    if (button.switchedOn() &&
        !running){
        running := true
        light.switchOn()
    } else if (button.switchedOff()
               && running){
        running := false
        light.switchOff()
    }
    if (running){
        fadeIn()
        fadeOut()
    }
}
```

```
fadeIn(){
    intensity := 0
    delta := 1/NUM_STEPS;
    for (i := 0; i < NUM_STEPS; i++){
        intensity := intensity + delta
        light.setIntensity(intensity)
        sleep(DeltaTime)
    }
}

fadeOut(){...}
```

PROBLEMA:
se il pulsante cambia stato durante
il fade in/fade out, il controller non lo rileva...

IL CONTROLLER COME ENTITÀ ATTIVA

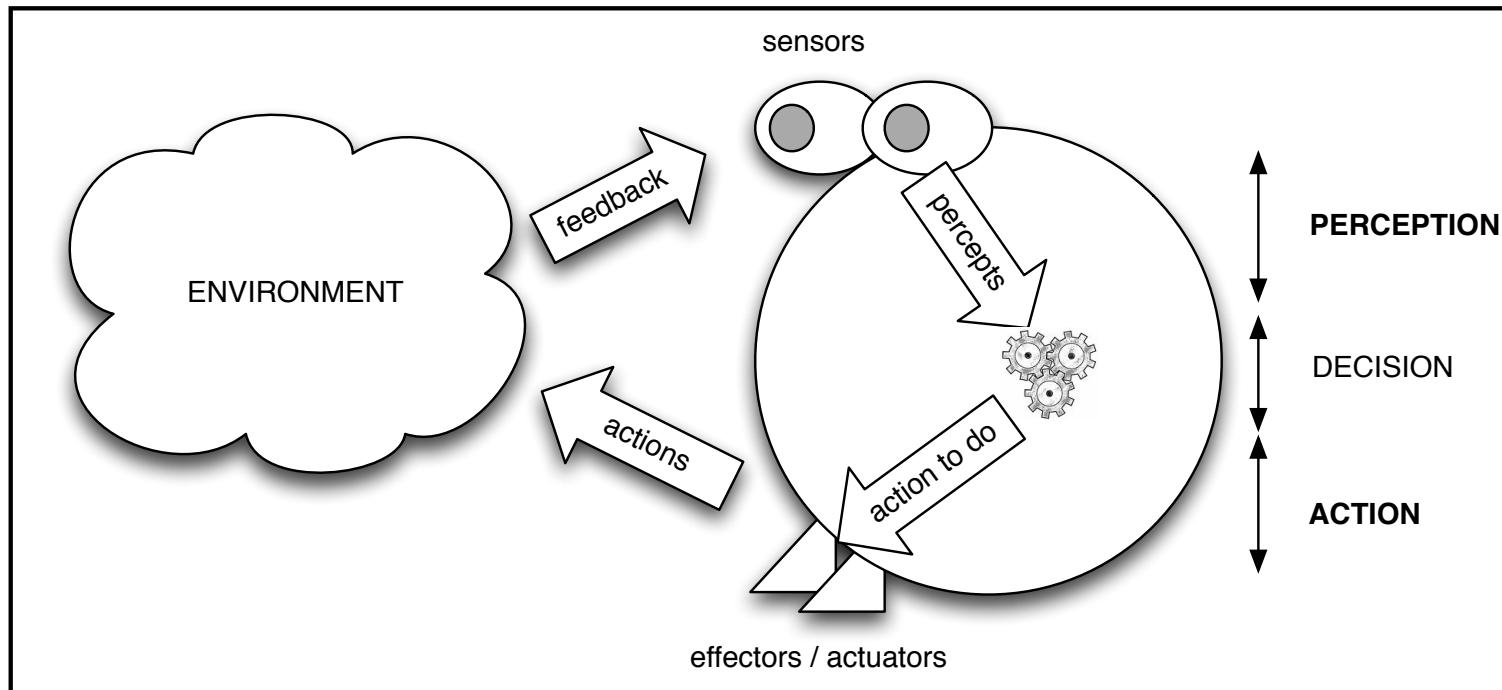
- Natura “concettuale” del controllore
 - il controllore è concettualmente rappresentabile come oggetto secondo il modello Object-Oriented?
- Negli esempi visti fino ad ora, led/luci/pulsanti sono rappresentabili come oggetti **passivi**
 - cambiano di stato a fronte di azioni eseguite o dal mondo esterno (Button), oppure dal controllore (Led)
 - hanno specifiche interfacce che permettono al controller di interagirvi, o per eseguire azioni o per leggerne lo stato
- Il controller è concettualmente una entità **attiva**, *dotata di flusso di controllo autonomo*
 - incapsula a livello logico la strategia di controllo del sistema
 - quale astrazione utilizzare per modellare questo tipo di entità?

MODELLO AD AGENTI

- Il controller è una entità diversa dal punto di vista concettuale dagli oggetti
 - è attivo, non ha interfaccia, incapsula la strategia di controllo...
- In queste sede utilizzeremo la nozione di **agente** per identificare questi tipi di entità
 - entità attive, dotate di un *flusso di controllo logico autonomo*, progettate per svolgere uno o più *compiti* (*task*) che richiedono di elaborare informazioni che provengono in input dall'ambiente da sensori e di agire su attuatori in output, eventualmente comunicando con altri agenti
- Progettazione *task-oriented*

AGENTI IN LETTERATURA

- Concetto introdotto e utilizzato in vari contesti
 - Intelligenza Artificiale e DAI
 - modellazione e simulazione di sistemi complessi
 - software engineering (Agent-Oriented Software Engineering)
 - comunità di ricerca su Agenti e Sistemi Multi-Agente [WOO,JEN]



MODELLO A LOOP: MODULARIZZAZIONE A TASK

- Nel caso di controllori (e loop di controllo) complessi e articolati, c'è la necessità di introdurre principi e tecniche di *decomposizione* e *modularizzazione*
- **Decomposizione per task (*compiti*)**
 - approccio adottato anche negli approcci ad agenti
 - agente => esecutore di uno o più task
 - integrazione con macchine a stati
 - comportamento di ogni task descrivibile mediante macchina stati finiti
- Introdotta e discussa nei prossimi moduli

SISTEMI DI SISTEMI

- Sistemi embedded di una certa complessità tipicamente sono *distribuiti* e costituiti da più sotto-sistemi embedded che interagiscono e cooperano
 - es: esempio Smart Home nel suo complesso
 - prospettiva IoT
- Questo richiede paradigmi di modellazione che considerano interazione distribuita, comunicazione e cooperazione come aspetti di prima classe
 - esempio: **sistemi multi-agente** [WOO,JEN]
 - comunicazione fra agenti è basata su scambio asincrono di messaggi

BIBLIOGRAFIA

- **[OLE]** Ole Lehrmann Madsen, Birger Møller-Pedersen. A Unified Approach to Modeling and Programming, Model Driven Engineering Languages and Systems. Lecture Notes in Computer Science Volume 6394, 2010, pp 1-15.
- **[GOF]** Erich Gamma; Richard Helm, Ralph Johnson, John M. Vlissides (1995). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley
- **[WOO]** Michael Wooldridge. An Introduction to MultiAgent Systems. John Wiley & Sons
- **[JEN]** Nicholas R. Jennings. 2001. An agent-based approach for building complex software systems. Commun. ACM 44, 4 (April 2001), 35-41.

Sistemi Embedded e IoT

Ingegneria e Scienze Informatiche - UNIBO

a.a 2020/2021

Docente: Prof. Alessandro Ricci

[modulo-2.2]

MODELLI BASATI SU
MACCHINE A STATI FINITI

OBIETTIVI DEL MODULO

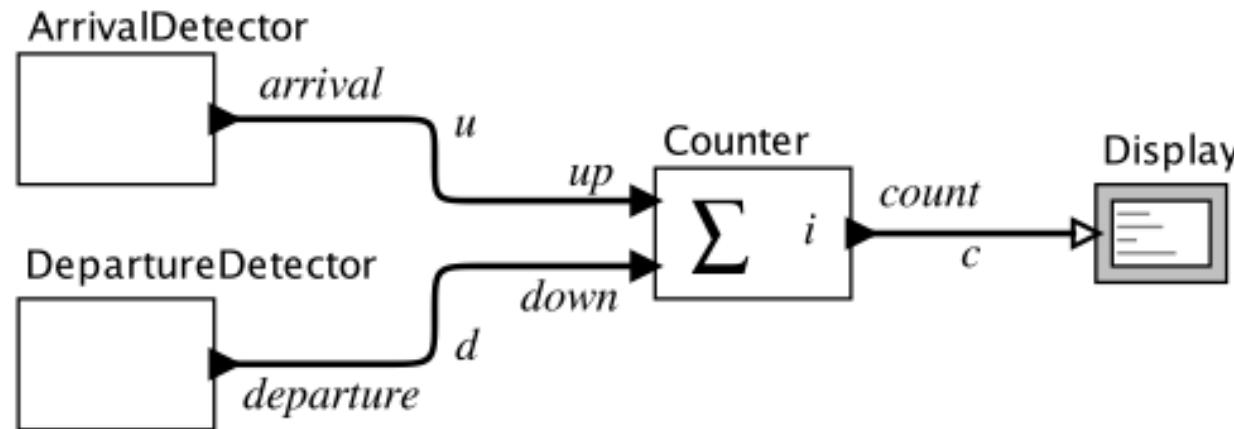
- In questo modulo si definisce in modo più rigoroso il modello basato macchine a stati finiti (FSM)
- Il contenuto è tratto dal capitolo 3 del testo **[IES]** e dal testo **[PES]**
 - vedere in bibliografia

MODELLI PER SISTEMI EMBEDDED

- In generale, il comportamento dinamico di un sistema può essere modellato con modelli nel continuo o nel discreto
 - un esempio di modello nel continuo è sistemi di equazioni differenziali lineari
- Le **macchine a stati finiti** (dette anche **automi a stati finiti**) sono il modello *nel discreto* più utilizzato per modellare sistemi embedded
 - la macchina a stati finiti è essa stessa un *sistema discreto*, ovvero opera in una sequenza di passi (discreti) e la sua dinamica è caratterizzata da sequenze di eventi (discreti)
 - un evento avviene ad un determinato istante, non ha durata
- Molti sistemi embedded hanno una natura inherentemente discreta
 - come esempio di riferimento nel prosieguo considereremo *Parking Garage* ([IES], p. 42), ovvero un sistema per il conteggio delle auto in un parcheggio

ESEMPIO: PARKING GARAGE

- Si consideri un sistema che deve effettuare il conteggio delle auto in un parcheggio, per tener traccia del numero di auto che ci sono in un qualsiasi istante (esempio tratto dal capitolo 3 di [IES], p. 42)
- Si consideri un modello di sistema costituito dai componenti:
 - *ArrivalDetector*, sottosistema che rileva l'arrivo di una nuova auto
 - *DepartureDetector*, sottosistema che rileva la dipartita di un auto
 - *Counter*, sottosistema che tiene traccia del conteggio delle auto



- In questo sistema ogni arrivo o dipartita di un auto sono modellati in modo naturale come eventi discreti

COUNTER

- Il sistema *Counter* reagisce in sequenza agli eventi di input che si presentano e produce un segnale di output
 - l'input è rappresentato da una coppia di *segnali discreti* (up/down) che in certi momenti *hanno un evento* (sono presenti) e in altri momenti non hanno eventi (assenti).
 - l'output è un segnale discreto che, quando l'input è presente, ha un valore ed è un numero naturale, e negli altri momenti è assente
- Quando un evento è presente alla porta up/down, *Counter* incrementa/decrementa il conteggio e produce il valore in uscita. In tutti gli altri casi - quando non c'è input - non produce output

MODELLO SEGNALI INPUT E OUTPUT

- I segnali u e d di input (up e down) che rappresentano il succedersi di eventi discreti possono essere rappresentati come funzioni:

$$u: \mathbb{R} \longrightarrow \{ \text{absent}, \text{present} \}$$

Ovvero ad ogni istante t , il segnale $u(t)$ è valutato o in "absent", che significa che non ci sono eventi in quel momento, oppure "present" intendendo che c'è un evento in quel momento.

- Questi segnali vengono definiti *puri*
 - non portano contenuto informativo, se non la sola informazione relativa alla presenza o assenza dell'evento rilevato dai sensori.
- Il segnale di uscita può essere rappresentato invece con una funzione:

$$c: \mathbb{R} \longrightarrow \{\text{absent}\} \cup \mathbb{N}$$

Questo non è un segnale puro, avendo contenuto informativo - anche se, come u e d , può essere presente o assente.

DINAMICA: REAZIONI

- La dinamica dei sistemi discreti come questo può essere descritta da una sequenza di step (passi) chiamati **reazioni**, ognuno dei quali si presuppone sia *istantaneo*, ovvero abbia durata nulla
- Le reazioni in un sistema discreto sono scatenate (*triggered*) dall'ambiente in cui opera il sistema
 - quando sono scatenate da eventi di input allora si dicono **event-triggered**
 - esempio del parcheggio:
 - le reazioni da parte del *Counter* sono scatenate quando sono presenti uno o più eventi di input relativamente all'arrivo o dipartita di un auto
 - quando entrambi gli input sono assenti, nessuna reazione avviene

VALUTAZIONE INPUT E OUTPUT

- L'esecuzione di una reazione comporta la *valutazione* degli input e degli output
 - ad ogni segnale in input si associa *una variabile* a cui viene assegnato il valore della funzione che rappresenta il segnale in quell'istante
 - stessa cosa vale per i segnali di uscita - per ogni segnale di uscita viene associata una variabile a cui viene assegnata il valore della funzione in quell'istante
 - i valori possono includere anche “absent”, ad intendere che non è presente alcun segnale

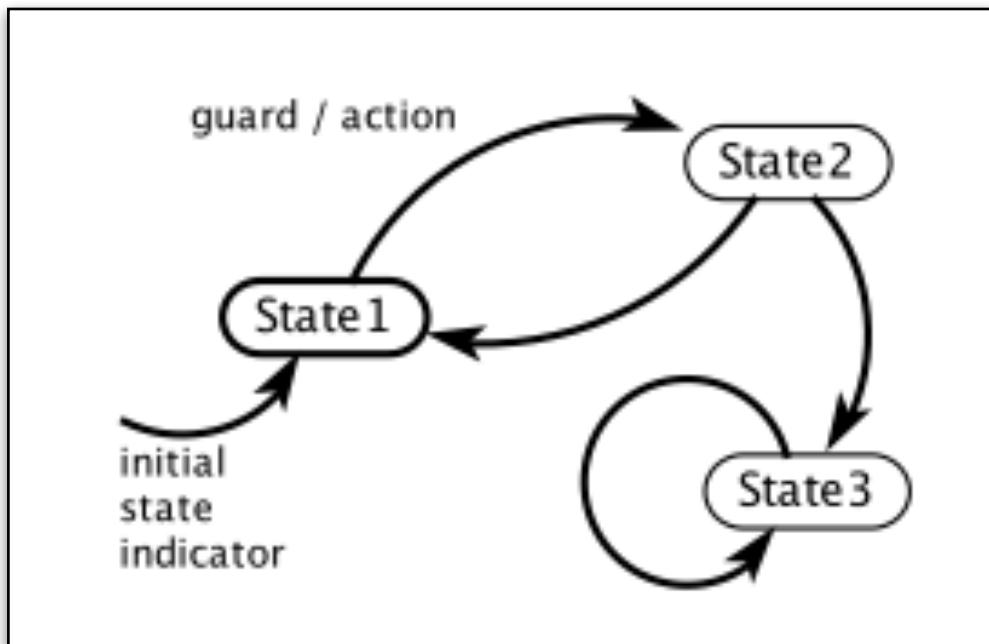
LA NOZIONE DI STATO

- Intuitivamente lo stato di un sistema è la condizione in cui si trova in un certo istante temporale
- Formalmente, lo stato rappresenta tutto ciò che è successo nel passato che ha un effetto nel determinare la reazione del sistema agli input correnti e futuri.
- Nel caso di un sistema *Counter* nell'esempio, lo stato $state(t)$ al tempo t è rappresentabile con un intero compreso fra 0 ed M , dove M è il numero massimo di posti
- Definito l'insieme $States = \{ 0, 1, 2, \dots M \}$, allora lo stato è modellabile come una funzione:

$state: \mathbb{R} \longrightarrow States$

MACCHINE A STATI FINITI

- Una macchina a stati è un modello di sistema a dinamica discreta, in cui ogni input del sistema viene mappato in un output a seconda del suo stato corrente
 - o meglio: ad ogni reazione, le valutazioni degli input vengono mappate in valutazioni degli output secondo lo stato corrente
- Una macchina a stati finiti (**FSM**, Finite State Machine) è una macchina a stati in cui il numero degli stati è finito
- Se il numero di stati è ragionevolmente ridotto, si può rappresentare graficamente con *diagrammi di stato*:



States =
{ State1, State2, State3 }

Stato iniziale = State1

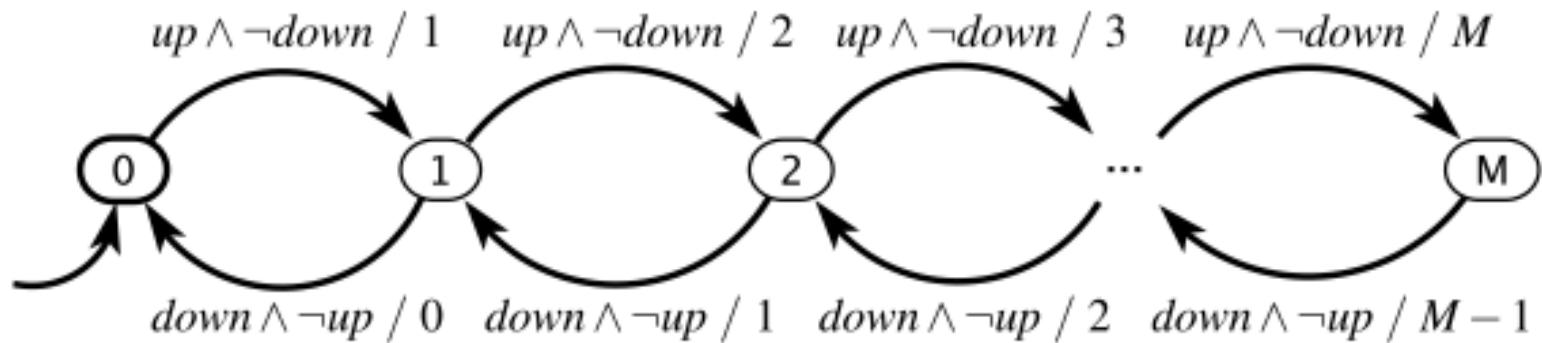
TRANSIZIONI

- Le **transizioni fra stati** è ciò che rappresentano l'evoluzione dinamica discreta, ovvero il comportamento, di una macchina a stati
- Una transizione coinvolge sempre due stati e può essere rappresentata da due elementi
 - **guardia**
 - è un predicato (ovvero una funzione booleana) sulle variabili di input e specifica le condizioni per cui la transizione è abilitata, ovvero può avvenire
 - se in una reazione, nessuna guardia è abilitata, la macchina rimane nello stato in cui si trova
 - **azione**
 - specifica il valore che devono assumere le variabili di output come risultato della transizione
 - se una variabile di output non viene specificata, si assume implicitamente che assuma il valore *absent*

ESEMPIO PARKING GARAGE

inputs: $up, down$: pure

output: $count : \{0, \dots, M\}$



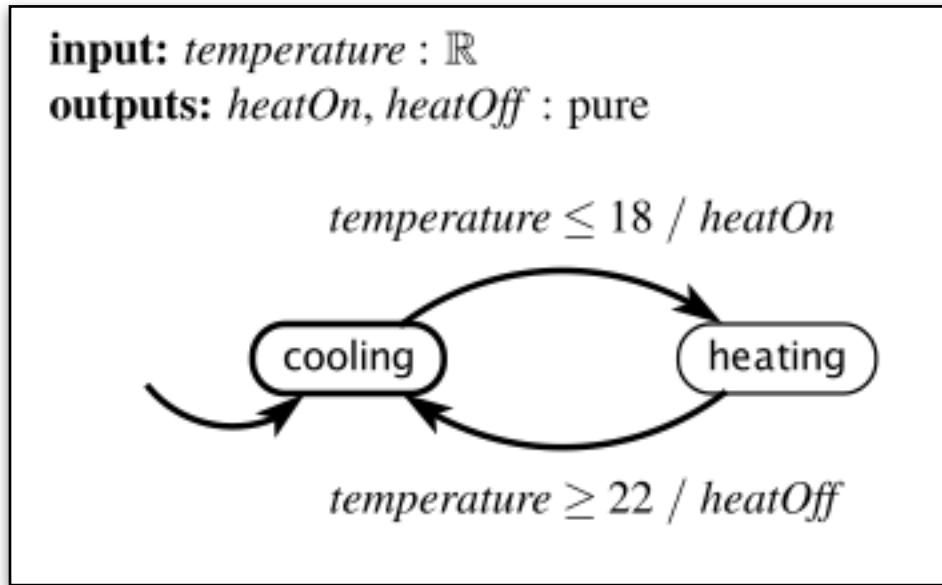
- Nelle guardie, i predicati coinvolgono la valutazione di funzioni pure trattate come espressioni booleane, per cui up significa la valutazione del segnale up è present, mentre $\neg down$ significa che la valutazione del segnale $down$ è absent, ovvero segnale non presente

FSM ASINCRONE E SINCRONE

- Una FSM non specifica di per sé *quando* una reazione deve avvenire
 - ovvero quando deve essere operata la valutazione degli input che porta a far scattare le transizioni
- Due possibilità
 - **FSM Asincrone o Event-triggered FSM**
 - in questo caso la reazione avviene a fronte di un evento di input
 - quindi è l'ambiente in cui opera la macchina che stabilisce quando la valutazione e quindi le reazioni devono avvenire
 - **FSM Sincrone o Time-triggered FSM**
 - in questo caso le reazioni avvengono ad intervalli regolari di tempo
 - è definito *un periodo* e quindi *una frequenza* di funzionamento

ESEMPIO HVAC

- Esempio di un sistema HVAC (heating, ventilation, air conditioning)



- Questa FSM potrebbe essere *event-triggered* nel caso in cui reagisca ogni volta che un valore di temperatura è fornito in input, oppure *time-triggered*, ovvero reagire ad intervalli regolari di tempo.
- La definizione della FSM non cambia

MODELLAZIONE INPUT/OUTPUT

- Nell'applicazione delle macchine a stati a sistemi embedded [vedere PES], in generale l'Input/Output è modellato in termini di variabili a cui la macchina può accedere
 - **variabili di input**
 - modificate dall'ambiente in cui opera la macchina
 - **variabili di output**
 - modificate dalla macchina stessa mediante azioni, come controllo dell'ambiente
 - oltre all'I/O, le variabili possono essere usate come modo più flessibile per caratterizzare lo stato a livello globale
 - accessibili da tutte le azioni e condizioni
- Una transizione avviene a fronte del verificarsi di determinate condizioni
 - come predicati/funzioni booleane sul valore delle variabili
- Esempi
 - blinking, button-led

MAPPING MACCHINA STATI-CODICE

- Aspetto importante della letteratura e della prassi dello sviluppo dei sistemi embedded classici
 - si cattura e descrive il comportamento con macchina a stati
 - poi lo si converte in codice da eseguire sul microcontrollore
 - conversione automatizzabile
 - *model-driven engineering*
- Nella conversione
 - separazione parte che rimane nel loop e funzione che rappresenta lo step della macchina [PSE]

```
/* global vars used by the machine */
int a0, b0, ...

/* variable keeping track of the state */
enum States {...} state;

/* procedure implementing the step of the state machine */
void step(){
    switch(state) {
        case ...
        case ...
    }
}

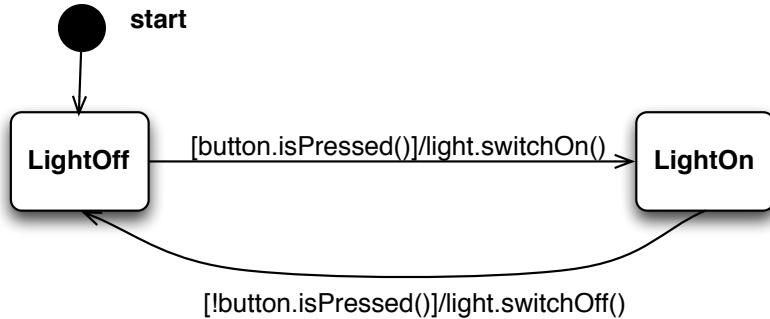
loop(){
    step();
}
```

MAPPING MACCHINA STATI-CODICE

- Dettaglio
 - rappresentazione esplicita stati
 - ad es: con costanti enumerative
 - variabile che tiene traccia dello stato corrente
 - “state” nell’esempio
 - step() della macchina nel main loop caratterizzato da un costrutto di selezione sullo stato corrente, con un ramo per ogni stato
 - a seconda dello stato, si verifica se/quale transizione è abilitata
 - si eseguono le azioni associate alla transizione e si cambia stato, riassegnando la variabile relativa allo stato corrente
 - nell’implementazione delle macchine sincrone si fa in modo che la selezione avvenga solo ogni specificato periodo di tempo
- Rappresentazione in UML mediante *statecharts*

ESEMPIO BUTTON-LED

- Schema generale per implementare una macchina stati finiti su loop di controllo
 - rappresentazione esplicita stati



```
enum States { LightOn, LightOff } state

setup(){
    state = LightOff
}

step() {
    switch (state){
        case LightOff:
            if (button.isPressed()){
                light.switchOn()
                state = LightOn
            }
        case LightOn:
            if (!button.isPressed()){
                light.switchOff()
                state = LightOff
            }
    }
}

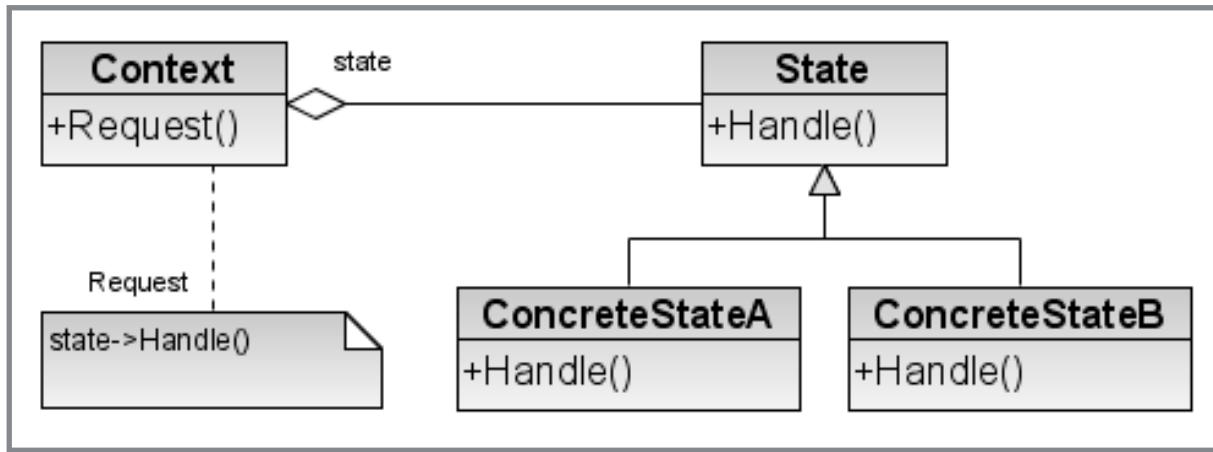
loop {
    step()
}
```

DISCIPLINA

- Per il corretto funzionamento e codifica di una macchina a stati:
 - ogni azione deve sempre terminare
 - non ci devono essere loop infiniti
 - la valutazione di una condizione non deve cambiare lo stato delle variabili

STATE PATTERN

- State pattern [GOF]



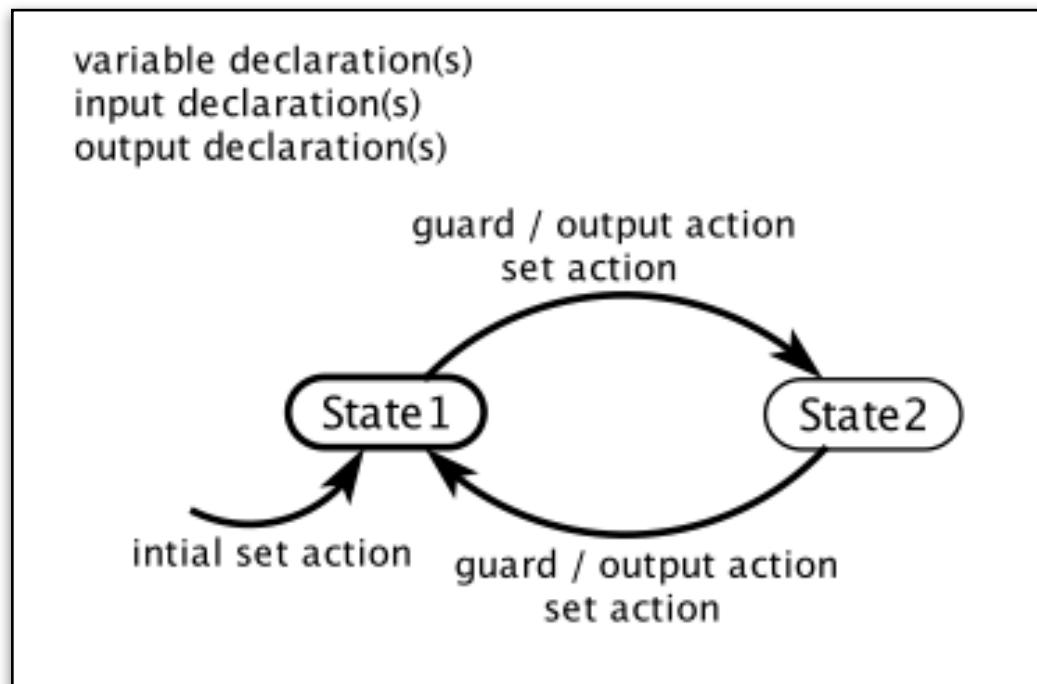
- Loop + State

```
interface State {  
    do()  
    nextState(): State  
}  
  
class MyState1 implements State {...}  
class MyState2 implements State {...}
```

```
...  
state = initialState  
loop {  
    state.do()  
    state = state.nextState()  
}
```

EXTENDED FSM

- Non appena la macchina a stati diviene articolata, la notazione precedente può risultare poco efficace - numero molto alto di stati da rappresentare
- Per ovviare al problema, si considera una estensione (equivalente dal punto di vista espressivo) con cui si usano delle **variabili come parte dello stato**, ovvero fornendo una rappresentazione *intensionale* dello stato
- In questo caso la parte delle azioni include delle **set actions**, con cui si assegna il valore della variabile di stato



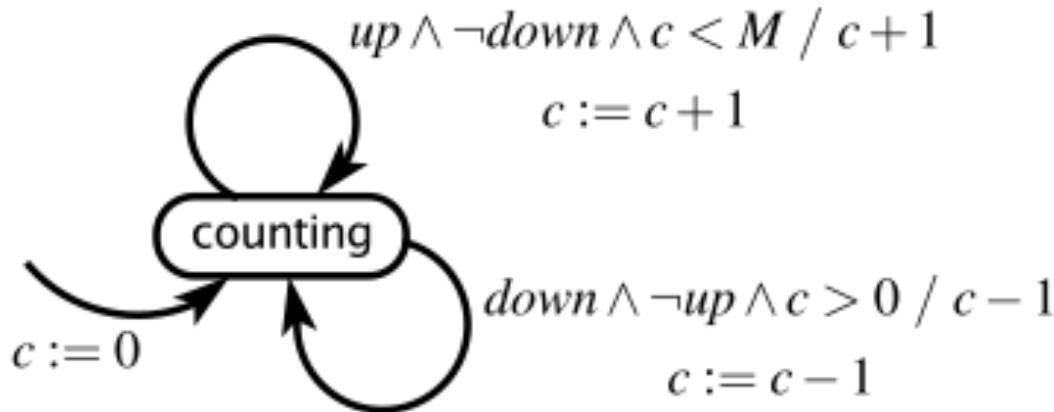
ESEMPIO PARKING GARAGE

- Sistema modellato come Extended FSM

variable: $c: \{0, \dots, M\}$

inputs: $up, down$: pure

output: $count: \{0, \dots, M\}$



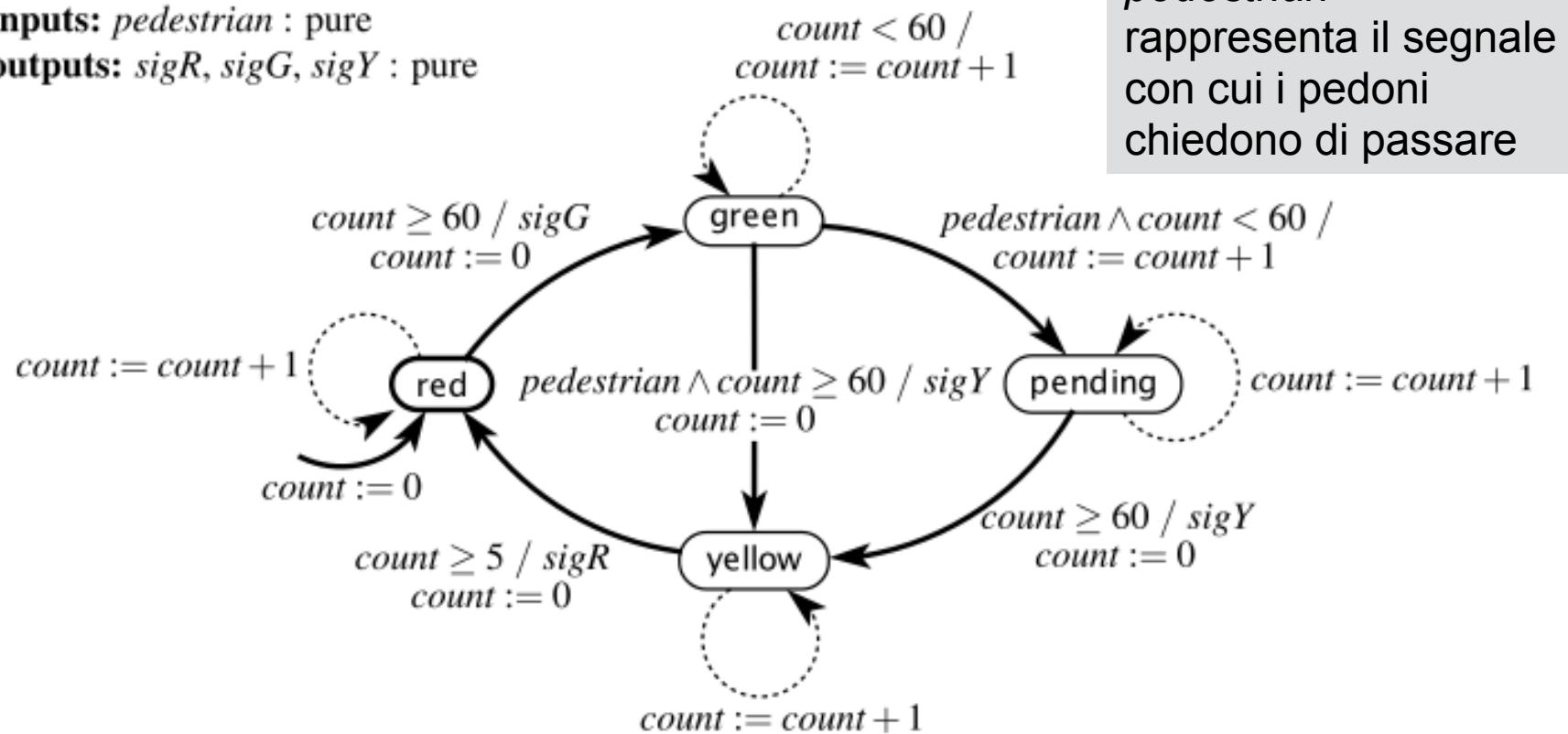
ESEMPIO DI TIME TRIGGERED EFSM

- Macchina a stati che descrive un semaforo pedonale
- Si presuppone sia time-triggered e che il periodo sia 1 secondo
 - quindi reagisce una volta al secondo

variable: $count: \{0, \dots, 60\}$

inputs: $pedestrian$: pure

outputs: $sigR, sigG, sigY$: pure



pedestrian
rappresenta il segnale
con cui i pedoni
chiedono di passare

NUMERO STATI

- Da notare che il numero reale degli stati di una EFSM include tutti quelli che derivano da ogni possibile configurazione ammissibile delle variabili di stato, per cui:

$$|\text{States}| = n^*p^m$$

dove

n = numero stati discreti

m = numero variabili che descrivono lo stato

p = possibili valori che può assumere ogni variabile

(questa formula assume che gli insiemi dei valori delle variabili abbiano tutti la medesima cardinalità)

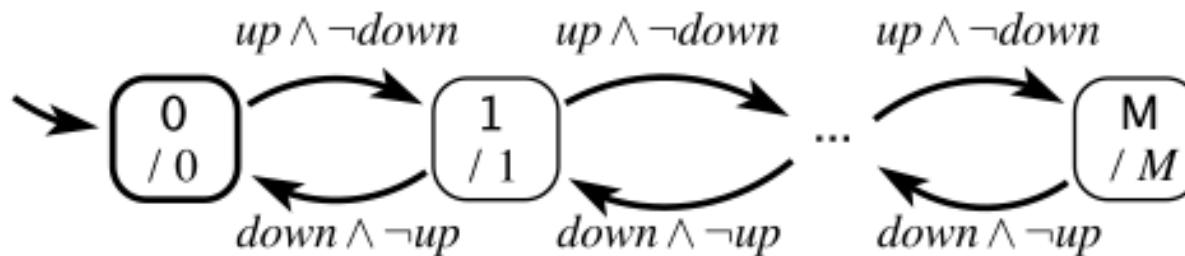
- Esempio Parking garage:
 - $|\text{States}| = 1^*(M+1)^1 = M+1$

MACCHINE DI MEALY E MOORE

- Le macchine a stati descritte in precedenza sono chiamate anche **Macchine di Mealy**
 - in onore a Gordon Mealy, ingegnere dei Bell Labs che pubblicò la descrizione di questo formalismo la prima volta nel 1955
 - sono caratterizzate dal produrre l'uscita (azione) quando scatta la transizione
- Un approccio alternativo è dato dalle **Macchine di Moore**, in cui l'output è associato allo stato, ed è prodotto quando si transita nello stato stesso
 - in onore a Edward Moore, sempre ingegnere Bell Labs, ideate nel 1956

ESEMPIO PARKING GARAGE COME MACCHINA DI MOORE

inputs: $up, down$: pure
output: $count: \{0, \dots, M\}$



Da notare che questa macchina non ha un comportamento **esattamente** equivalente alla macchina di Mealy precedente...

- L'output è determinato dallo stato, non dall'input
 - l'input determina quale transizione viene fatta, ma non l'output prodotto dalla reazione
 - in questo caso la macchina si dice *causale*
- Data una macchina di Moore è possibile definire una macchina di Mealy equivalente
 - viceversa è possibile definire una macchina di Moore a partire da una macchina di Moore con tuttavia la differenza per cui l'output viene prodotto alla successiva reazione e non in quella corrente..

MACCHINE A STATI FINITI SINCRONE: APPROFONDIMENTO

- Molti comportamenti nei sistemi embedded sono **time-oriented**
 - non solo prevedono un ordinamento temporale delle azioni, ma includono la specifica di **intervalli temporali** nella definizione del comportamento
 - es: blinking
 - accendere e spegnere un led ogni 500 ms
- Le **macchine a stati sincrone** [PES] sono macchine a stati estese con supporto nativo del trascorrere del tempo per agevolare la specifica di comportamenti time-oriented
 - in particolare in queste macchine lo step di una macchina a stati viene eseguito periodicamente, secondo un certo **periodo**
 - è come se la macchina a stati avesse un **clock** interno e le transizioni vengono eseguite solo al verificarsi di un tick del clock

MAPPING MACCHINE SINCRONE - CODICE: USO INTERRUZIONE TIMER

- La descrizione del comportamento di una macchina a stati sincrona in linguaggi procedurali come il C su sistemi a microcontrollore tipicamente si basa sull'uso dell'**interruzione di Timer programmabili**
 - inclusi nel microcontroller o sulla scheda, permettono di generare un segnale di interruzione (del timer) periodico.

ESEMPIO PSEUDO-CODICE [PES]

```
volatile int timerFlag = 0;

void timerISR(){
    timerFlag = 1;
}

/* procedure implementing the step of the state machine */
void step(){...}

loop(){
    while (!timerFlag){}; /* wait for a tick for doing the next step */
    timerFlag = 0;
    step();
}
```

BLINK SINCRONO

- Versione a FSM sincrona del blink, con periodo 500 ms

```
#include "Led.h"
#include "Timer.h"

#define LED_PIN 13

Light* led;
Timer timer;

enum { ON, OFF} state;

void step(){
    switch (state){
        case OFF:
            led->switchOn();
            state = ON;
            break;
        case ON:
            led->switchOff();
            state = OFF;
            break;
    }
}
```

```
...

void setup(){
    led = new Led(LED_PIN);
    state = OFF;
    timer.setupPeriod(500);
}

void loop(){
    timer.waitForNextTick();
    step();
};
```

Timer è una libreria che contiene in forma di classe le routine di gestione del timer viste in precedenza (inclusa fra sorgenti nel materiale del corso per il modulo-lab-2.2)

TIME-SCALE MULTIPLI

- Un sistema può avere intervalli temporali di lunghezza diversa da gestire e il comportamento può essere rappresentato da opportune macchine a stati sincrone
 - come periodo si sceglie il massimo comun divisore degli intervalli caratterizzanti il sistema
- Esempio
 - blinking, con LED acceso per 500 ms e spento per 750 ms => si sceglie periodo di 250 ms

CAMPIONAMENTO INPUT

- Considerando macchine a stati sincrone, la lettura periodica di un sensore ad una data frequenza (ovvero con un certo periodo) viene definito **campionamento** (*sampling*)
 - caratterizzata da un certo il **periodo** di campionamento (*sampling rate*)
 - per frequenza di campionamento si intende il valore inverso del periodo, ovvero il numero di volte al secondo in cui avviene
 - in generale si misura in Hertz (Hz)
- La scelta del periodo è un aspetto molto importante della progettazione
 - in generale il periodo deve essere **sufficientemente piccolo** da *evitare di perdere eventi*
 - esempio button-led

BUTTON-LED SINCRONO

- Versione a FSM del button-led, con periodo 500 ms
 - troppo elevato, perdo eventi...

```
#include "Led.h"
#include "Timer.h"

#define LED_PIN 13
#define BUTTON_PIN 2

Light* led;
Button* button;
Timer timer;

enum { ON, OFF} state;

void setup(){
    led = new Led(LED_PIN);
    button = new ButtonImpl(BUTTON_PIN);
    state = OFF;
    timer.setupPeriod(500);
}

void loop(){
    timer.waitForNextTick();
    step();
};

...
```

```
void step(){
    switch (state){
        case OFF:
            if (button->isPressed()){
                led->switchOn();
                state = ON;
            }
            break;
        case ON:
            if (!button->isPressed()){
                led->switchOff();
                state = OFF;
            }
            break;
    }
}
```

SCELTA DEL PERIODO DI CAMPIONAMENTO

- Maggiore è la frequenza (ovvero minore è il periodo), e
 - da un lato, *maggiorre è reattività*
 - dall'altro, è *maggiorre l'utilizzo* (spesso inutile) del microcontrollore e quindi il consumo di energia
 - aspetto molto importante nei sistemi embedded
- Quindi in generale il periodo dovrebbe esser scelto in modo da essere il più **grande possibile**, per minimizzare il consumo
 - a partire dai valori che tuttavia garantiscono il corretto funzionamento (in termini di reattività) del sistema....
- In altre parole:
 - *il valore del periodo deve essere il più grande fra i valori sufficientemente piccoli da garantire il corretto funzionamento del sistema*

MINIMUM EVENT SEPARATION TIME

- Definizione **minimum event separation time** (MEST)
intervallo più piccolo che - dato l'ambiente in cui opera il nostro sistema - può esserci fra due eventi di input
- **Teorema**
 - *in una macchina a stati sincrona, scegliendo un periodo inferiore al MEST si garantisce che tutti gli eventi saranno rilevati*
 - vedi esempio button-led
- Nota
 - il MEST di eventi di input come pulsanti non include solo gli eventi in se di pressione del pulsante, ma l'intervallo di tempo fra tali eventi.
 - se le pressioni di un pulsante richiedono di essere separate da intervalli di 300 ms, allora sampling rate maggiori di 300 ms possono risultare nel non rilevamento dell'evento di rilascio

LATENZA

- Alcuni eventi di input possono dover scatenare nuovi eventi di output o informazioni in output.
- L'intervallo che trascorre tra l'evento di input e la generazione dell'output è detta **latenza**
 - ad esempio, nel sistema button-led, il tempo che intercorre dalla pressione del pulsante all'accensione del led è una latenza.
- Obiettivo usuale è **minimizzare le latenze**
 - questo influisce sulla scelta del periodo: più piccolo è il periodo, più si riducono le latenze
 - ad esempio, per il button-led
 - 300ms di latenza non sarebbe accettabile - la latenza sarebbe percepibile (come ritardo) da una persona
 - in questo caso 50 ms invece è un valore accettabile

CONDIZIONAMENTO DELL'INPUT (INPUT CONDITIONING)

- I sensori in generale possono presentare imperfezioni (a livello hardware), che richiedono *aggiustamenti* (**input conditioning**, condizionamenti dell'input) dei valori letti da parte del microcontrollore
- Un classico esempio è il **bouncing dei pulsanti**
 - il meccanismo interno a molti pulsanti, a fronte della pressione, può portare ad avere un fenomeno fisico simile al rimbalzo (meccanico), per cui il segnale corrispondente passa da un valore LOW ad un valore HIGH e viceversa più volte, prima di assestarsi su valore HIGH
 - in questo caso, se la frequenza di campionamento del segnale è molto alta, allora il sistema può inferire non correttamente che il pulsante sia stato premuto più volte...

BUTTON DE-BOUNCING

- Per button debouncing si intende la funzionalità di ignorare i rimbalzi (bouncing) spuri del pulsante, in modo che una singola pressione sia rilevata.
- Il problema si può risolvere
 - a livello hardware
 - a livello software, considerando una frequenza di campionamento **inferiore** alla frequenza del bouncing
 - ovvero un periodo superiore a quello del rimbalzo
 - nel caso concreto del pulsante: pulsanti rimbalzano non più di 10-20 ms, per cui un periodo di 50 ms va bene

BUTTON-LED SINCRONO

- Versione a FSM del button-led, con periodo corretto

```
#include "Led.h"
#include "Timer.h"

#define LED_PIN 13
#define BUTTON_PIN 2

Light* led;
Button* button;
Timer timer;

enum { ON, OFF} state;

void setup(){
    led = new Led(LED_PIN);
    button = new ButtonImpl(BUTTON_PIN);
    state = OFF;
    timer.setupPeriod(50);
}

void loop(){
    timer.waitForNextTick();
    step();
}
```

```
...
void step(){
    switch (state){
        case OFF:
            if (button->isPressed()){
                led->switchOn();
                state = ON;
            }
            break;
        case ON:
            if (!button->isPressed()){
                led->switchOff();
                state = OFF;
            }
            break;
    }
}
```

GLITCH, SPIKE E FILTERING

- Generalizzando il caso del bouncing, l'input conditioning può avvenire mediante **tecniche di filtraggio** utili per ignorare la presenza di eventi di input spuri
 - *segnali spuri* - detti **glitches**, o **spikes** - possono esserci anche a causa delle condizioni ambientali
 - ad esempio a casa di interferenze elettromagnetiche
- Una tecnica di filtraggio consiste nella *riduzione del periodo di campionamento*
 - questo può ridurre la probabilità di rilevare segnali spuri
 - tuttavia *non la elimina*
 - un glitch può essere sempre rilevato se coincide esattamente con il momento in cui viene fatto il rilevamento

GLITCH, SPIKE E FILTERING

- Una soluzione più generale in questo caso consiste nel richiedere che il segnale rilevato che rappresenta uno specifico evento, per essere considerato tale abbia una certa **durata minima**
 - corrispondente ad esempio ad una serie di più campionamenti e considerando che il periodo di campionamento sia maggiore della durata di un glitch
 - esempio: per un periodo di campionamento di 50 ms di un pulsante e glitch durano al più 10 ms, possiamo richiedere che una serie di 2 campionamenti siano ad rilevati a HIGH per poter considerare il pulsante premuto
 - questo tuttavia ha un impatto sul MEST, riducendo (a metà) l'intervallo che può intercorrere fra due eventi
- Anche questa soluzione di per sé non elimina del tutto la possibilità di interpretare glitch come eventi
 - esempio: 2 glitch che avvengono nel momento in cui c'è il campionamento => interpretati come evento
- Inoltre introduce latenze

SOMMARIO

- Sistemi discreti
 - dinamica discreta, eventi discreti
- FSM
 - stati, transizioni, guardie/azioni
 - sincrone/time-triggered, asincrone/event-triggered
 - mapping modello/codice
- Modello formale di una FSM
 - funzione di transizione
- Extended FSM
- Macchine di Mealy / Macchine di Moore
- Comportamenti, tracce osservabili e d'esecuzione
- Approfondimento su macchine a stati finiti sincrone

BIBLIOGRAFIA

- **[ESP]** Introduction to Embedded Systems. Lee, Sheshia.
 - capitolo 3
- **[PES]** Frank Vahid, Tony Vargis, Bailey Miller. *Programming Embedded Systems: An introduction to Time-Oriented Programming*. University of California, Riverside.
- **[GOF]** Erich Gamma; Richard Helm, Ralph Johnson, John M. Vlissides (1995). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley

Sistemi Embedded e IoT

Ingegneria e Scienze Informatiche - UNIBO
a.a 2020/2021

Docente: Prof. Alessandro Ricci

[modulo-2.3]

ORGANIZZAZIONE A TASK CONCORRENTI

SOMMARIO

- Organizzazione a task
 - integrazione con macchine a stati sincrone
- Scheduling cooperativo dei task
- Analisi esecuzione
 - parametri utilizzo CPU, Worst-Case Execution Time,
 - problemi
 - overrun, jitter
- Task con deadline e scheduling con priorità
 - priorità statica, dinamica
- Task basati su macchine a stati event-triggered

ORGANIZZAZIONE A TASK CONCORRENTI

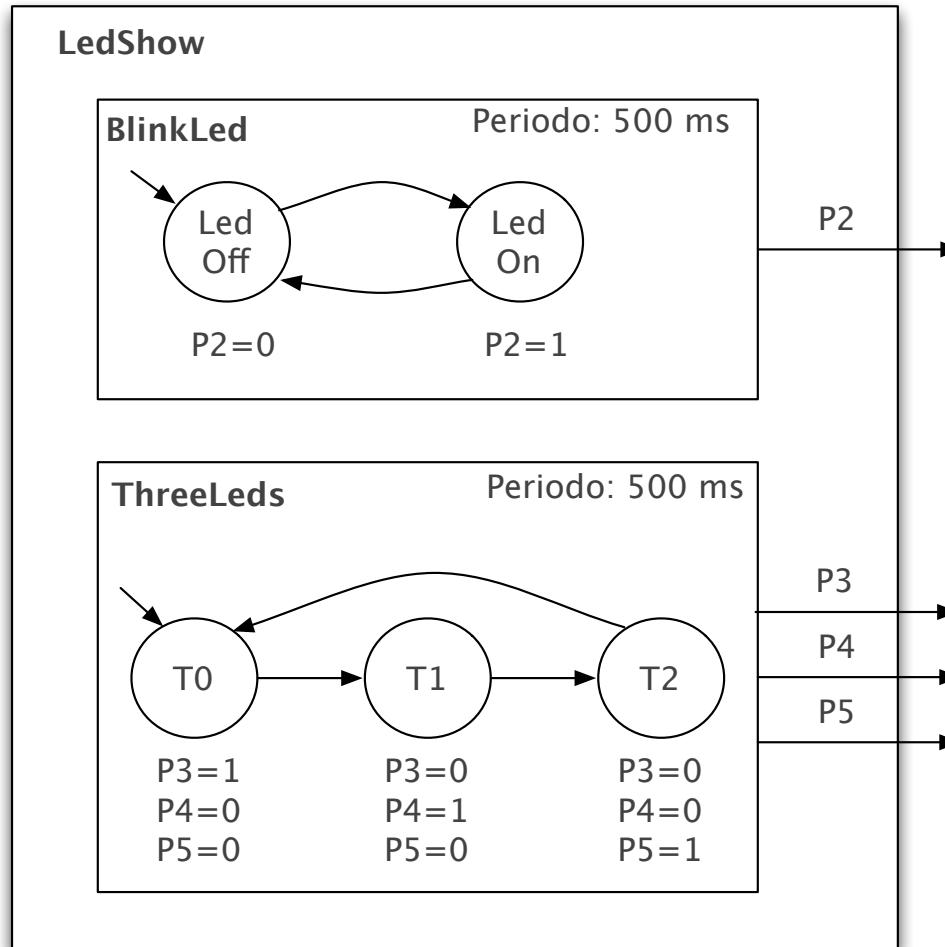
- Problema: modellazione e progettazione di sistemi software embedded articolati/complessi
 - necessità di approcci opportuni per decomporre/modularizzare il comportamento e le funzionalità
- Un approccio utilizzato nel contesto della programmazione concorrente è la decomposizione del comportamento complessivo in più **task**
 - ogni task rappresenta un compito ben definito, una unità di lavoro da svolgere
 - il comportamento di ogni task può essere descritto da una opportuna macchina a stati
 - il comportamento complessivo è dall'insieme delle macchine a stati

ESEMPIO: LED-SHOW

- Esempio LedShow con 4 led (p. 68, [PES])
 - blinking di un led ogni 500 ms
 - allo stesso tempo: gestione di 3 led in cui se ne accenda uno alla volta, in sequenza, ogni 500ms
- La modellazione del comportamento con una sola macchina a stati finiti è possibile, tuttavia il modello risultante è molto complicato
 - numero elevato stati e transizioni

DIAGRAMMI A STATI

- E' possibile rappresentare esplicitamente nei diagrammi a stati questo tipo di organizzazione mediante blocchi partizionati in parti, ognuna delle quali contengono macchine a stati autonome:



DECOMPOSIZIONE IN TASK: VANTAGGI

- Questo è un principio di progettazione importante, che permette di rendere *modulare* il sistema
 - dove ogni modulo è rappresentato da un task
 - un task può essere decomposto in sotto-task (ricorsivamente)
 - equivalentemente: un task complesso può essere definito come composizione di sotto-task più semplici
 - le macchine a stati possono essere usate per rappresentare il comportamento associato ad un task "atomico"
 - che in questo caso significa non ulteriormente decomponibile..
- Vantaggi
 - modularità, *separation of concerns*
 - chiarezza, manutenibilità, estendibilità, riusabilità

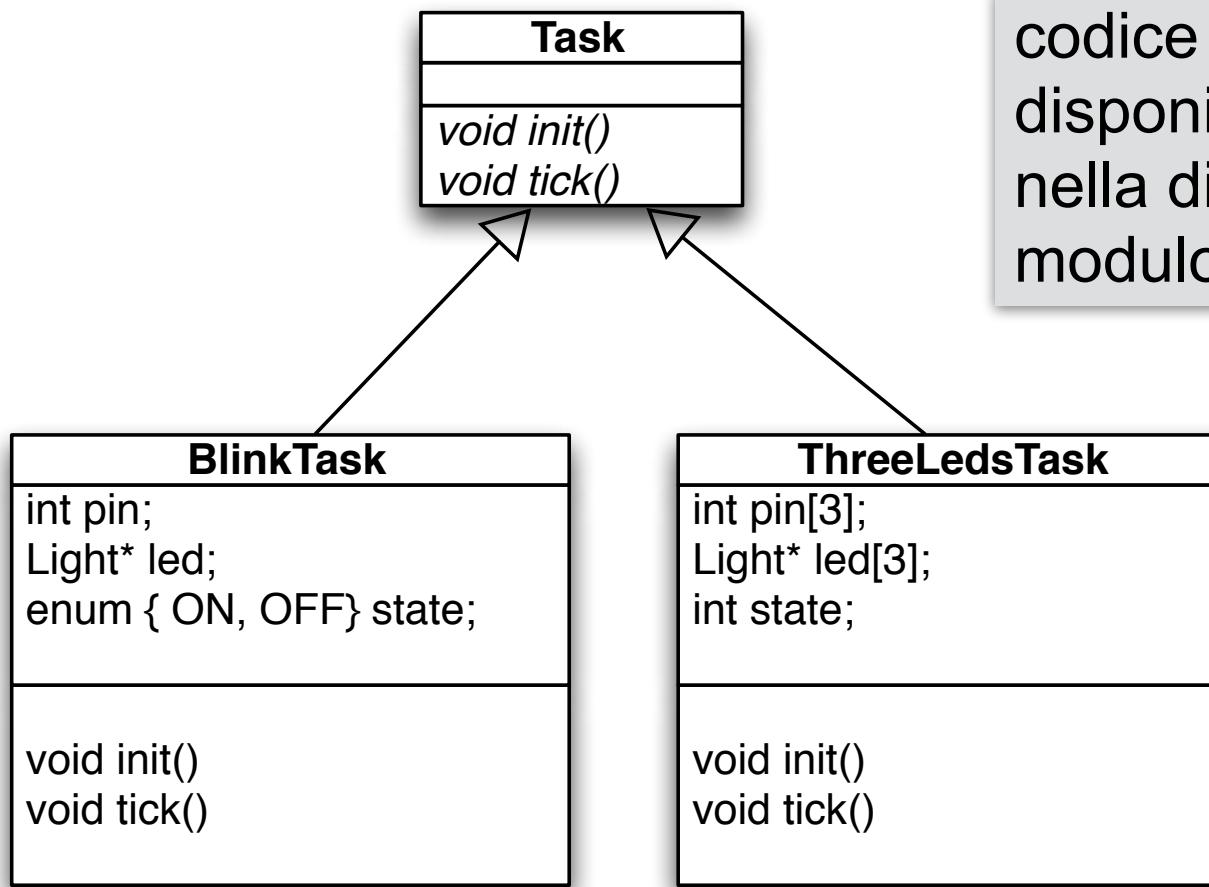
DECOMPOSIZIONE IN TASK: PROBLEMI

- I task sono concorrenti
 - la loro esecuzione si sovrappone nel tempo
- I task possono avere *dipendenze* che richiedono di essere gestite mediante opportune forme di *coordinazione*
 - cooperative
 - esempio: comunicazione, sincronizzazione
 - competitive
 - esempio: mutua esclusione, sezioni critiche
 - accesso in lettura+scrittura a memoria condivisa
 - accesso alle medesime risorse di I/O

MAPPING SUL CODICE: IMPLEMENTAZIONE DI SCHEMI DI MULTI-TASKING

- Nei sistemi embedded basati su microcontrollore (e senza OS), un primo modo semplice di realizzare task concorrenti è come semplice estensione del caso a singolo task per cui, anziché avere un'unica procedura tick (o step) che viene richiamata ad ogni ciclo, in questo caso ci sono più funzioni tick, una per ogni task
- Considerando poi una modellazione ad oggetti, modelleremo ogni task come classe separata che estende da una classe base task, in cui è definito il metodo step, specializzato in ogni classe concreta
 - quindi nel programma avremo una sola istanza di questa classe (pattern singleton).

ESEMPIO LED-SHOW IN ARDUINO



codice sorgente
disponibile nel materiale
nella directory
modulo-lab-2.3/led-show

ESEMPIO LED-SHOW IN ARDUINO: CLASSE ASTRATTA TASK

```
#ifndef __TASK__
#define __TASK__

class Task {

public:
    virtual void init() = 0;
    virtual void tick() = 0;

};

#endif
```

ESEMPIO LED-SHOW IN ARDUINO: BLINK TASK

```
#ifndef __BLINKTASK__
#define __BLINKTASK__

#include "Task.h"
#include "Led.h"

class BlinkTask: public Task {

    int pin;
    Light* led;
    enum { ON, OFF} state;

public:

    BlinkTask(int pin);
    void init();
    void tick();
};

#endif
```

BlinkTask.h

```
#include "BlinkTask.h"

BlinkTask::BlinkTask(int pin){
    this->pin = pin;
}

void BlinkTask::init(){
    led = new Led(pin);
    state = OFF;
}

void BlinkTask::tick(){
    switch (state){
        case OFF:
            led->switchOn();
            state = ON;
            break;
        case ON:
            led->switchOff();
            state = OFF;
            break;
    }
}
```

BlinkTask.cpp

ESEMPIO LED-SHOW IN ARDUINO: THREE LEADS TASK

```
#ifndef __THREELEDSTASK__
#define __THREELEDSTASK__

#include "Task.h"
#include "Led.h"

class ThreeLedsTask: public Task {

    int pin[3];
    Light* led[3];
    int state;

public:

    ThreeLedsTask(int pin0, int pin1,
                  int pin2);
    void init();
    void tick();
};

#endif
```

ThreeLedsTask.h

```
#include "ThreeLedsTask.h"

ThreeLedsTask::ThreeLedsTask(int pin0, int pin1,
                            int pin2){
    this->pin[0] = pin0;
    this->pin[1] = pin1;
    this->pin[2] = pin2;
}

void ThreeLedsTask::init(){
    for (int i = 0; i < 3; i++){
        led[i] = new Led(pin[i]);
    }
    state = 0;
}

void ThreeLedsTask::tick(){
    led[state]->switchOff();
    state = (state + 1) % 3;
    led[state]->switchOn();
}
```

ThreeLedsTask.cpp

ESEMPIO LED-SHOW IN ARDUINO: IL MAIN

```
#include "Timer.h"
#include "BlinkTask.h"
#include "ThreeLedsTask.h"

Timer timer;

BlinkTask blinkTask(2);
ThreeLedsTask threeLedsTask(3,4,5);

void setup(){
    blinkTask.init();
    threeLedsTask.init();
    timer.setupPeriod(500);
}

void loop(){
    timer.waitForNextTick();
    blinkTask.tick();
    threeLedsTask.tick();
}
```

led-show.ino

se `blinkTask.tick()` impiega troppo tempo,
`threeLedsTask.tick()` non viene eseguita ogni 500ms.

GESTIONE DI PERIODI DIVERSI

- Nel caso di macchine a stati con periodi diversi è necessario:
 - tenere traccia per ogni task anche del periodo specifico
 - far funzionare la macchina a stati con il periodo pari al *massimo comun divisore*
 - tener traccia per ogni task del tempo trascorso e nel caso si sia raggiunto il valore del periodo, richiamare il tick

ESEMPIO LED-SHOW CON PERIODI DIVERSI

```
#ifndef __TASK__
#define __TASK__


class Task {
    int myPeriod;
    int timeElapsed;

public:
    virtual void init(int period){
        myPeriod = period;
        timeElapsed = 0;
    }

    virtual void tick() = 0;

    bool updateAndCheckTime(int basePeriod){
        timeElapsed += basePeriod;
        if (timeElapsed >= myPeriod){
            timeElapsed = 0;
            return true;
        } else {
            return false;
        }
    }
};

#endif
```

codice sorgente disponibile
nel materiale nella directory
modulo-lab-2.3/led-show-different-
period

ESEMPIO LED-SHOW CON PERIODI DIVERSI

```
#ifndef __BLINKTASK__
#define __BLINKTASK__

#include "Task.h"
#include "Led.h"

class BlinkTask: public Task {

    int pin;
    Light* led;
    enum { ON, OFF} state;

public:

    BlinkTask(int pin);
    void init(int period);
    void tick();
};

#endif
```

BlinkTask.h

```
#include "BlinkTask.h"

BlinkTask::BlinkTask(int pin){
    this->pin = pin;
}

void BlinkTask::init(int period){
    Task::init(period);
    led = new Led(pin);
    state = OFF;
}

void BlinkTask::tick(){
    switch (state){
        case OFF:
            led->switchOn();
            state = ON;
            break;
        case ON:
            led->switchOff();
            state = OFF;
            break;
    }
}
```

BlinkTask.cpp

ESEMPIO LED-SHOW CON PERIODI DIVERSI

```
#ifndef __THREELEDSTASK__
#define __THREELEDSTASK__

#include "Task.h"
#include "Led.h"

class ThreeLedsTask: public Task {

    int pin[3];
    Light* led[3];
    int state;

public:

    ThreeLedsTask(int pin0, int pin1,
                  int pin2);
    void init(int period);
    void tick();
};

#endif
```

ThreeLedsTask.h

```
#include "ThreeLedsTask.h"

ThreeLedsTask::ThreeLedsTask(int pin0, int pin1,
                            int pin2){
    this->pin[0] = pin0;
    this->pin[1] = pin1;
    this->pin[2] = pin2;
}

void ThreeLedsTask::init(int period){
    Task::init(period);
    for (int i = 0; i < 3; i++){
        led[i] = new Led(pin[i]);
    }
    state = 0;
}

void ThreeLedsTask::tick(){
    led[state]->switchOff();
    state = (state + 1) % 3;
    led[state]->switchOn();
}
```

ThreeLedsTask.cpp

ESEMPIO LED-SHOW CON PERIODI DIVERSI

```
#include "Timer.h"
#include "BlinkTask.h"
#include "ThreeLedsTask.h"

const int basePeriod = 50;

Timer timer;

BlinkTask blinkTask(2);
ThreeLedsTask threeLedsTask(3,4,5);

void setup(){
    MCD(500, 150) = 50
    timer.setupPeriod(basePeriod);
    blinkTask.init(500);
    threeLedsTask.init(150);
}

void loop(){
    timer.waitForNextTick();
    if (blinkTask.updateAndCheckTime(basePeriod)){
        blinkTask.tick();
    }
    if (threeLedsTask.updateAndCheckTime(basePeriod)){
        threeLedsTask.tick();
    }
};
```

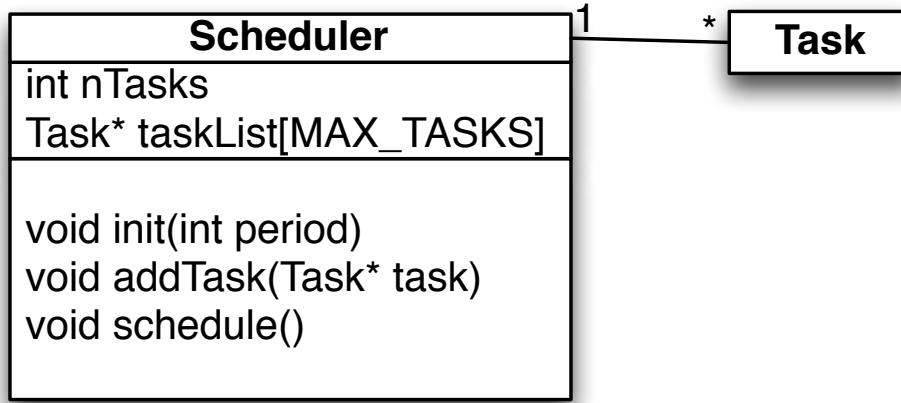
led-show-different-period.ino

MULTI-TASKING COOPERATIVO

- Il tipo di multi-tasking adottato in questo caso è **cooperativo**, con scheduling tipo *round-robin*
- Vantaggi
 - non c'è interleaving fra le istruzioni di task diversi
 - `tick()` eseguita atomicamente (rispetto a task diversi)
 - > non ci possono essere corse critiche
- Svantaggi
 - il comportamento errato di un task può compromettere l'esecuzione degli altri task
 -  **esempio: loop infinito in `tick()` di un task manda in starvation gli altri task**

CREAZIONE DI UN SEMPLICE SCHEDULER COOPERATIVO

- La generalizzazione dell'approccio porta alla definizione di un vero e proprio scheduler, in cui si tiene traccia mediante una opportuna struttura dati (es: una coda) della lista dei task in esecuzione.
- Nel mail loop si visita ogni elemento in coda, richiedendone il tick nel caso in cui il tempo trascorso sia pari al periodo previsto dal task.



il codice sorgente che segue è disponibile nel materiale nella directory modulo-lab-2.3/led-show-scheduler

ESEMPIO LED-SHOW CON SCHEDULER

```
#ifndef __SCHEDULER__
#define __SCHEDULER__

#include "Timer.h"
#include "Task.h"

#define MAX_TASKS 10

class Scheduler {

    int basePeriod;
    int nTasks;
    Task* taskList[MAX_TASKS];
    Timer timer;

public:

    void init(int basePeriod);
    virtual bool addTask(Task* task);
    virtual void schedule();

};

#endif
```

Scheduler.h

```
#include "Scheduler.h"

void Scheduler::init(int basePeriod){
    this->basePeriod = basePeriod;
    timer.setupPeriod(basePeriod);
    nTasks = 0;
}

bool Scheduler::addTask(Task* task){
    if (nTasks < MAX_TASKS-1){
        taskList[nTasks] = task;
        nTasks++;
        return true;
    } else {
        return false;
    }
}

void Scheduler::schedule(){
    timer.waitForNextTick();
    for (int i = 0; i < nTasks; i++){
        if (taskList[i]->updateAndCheckTime(basePeriod)){
            taskList[i]->tick();
        }
    }
}
```

Scheduler.cpp

ESEMPIO LED-SHOW CON SCHEDULER

```
#include "Scheduler.h"
#include "BlinkTask.h"
#include "ThreeLedsTask.h"

Scheduler sched;

void setup(){

    sched.init(50);

    Task* t0 = new BlinkTask(2);
    t0->init(500);
    sched.addTask(t0);

    Task* t1 = new ThreeLedsTask(3,4,5);
    t1->init(150);
    sched.addTask(t1);

}

void loop(){
    sched.schedule();
}
```

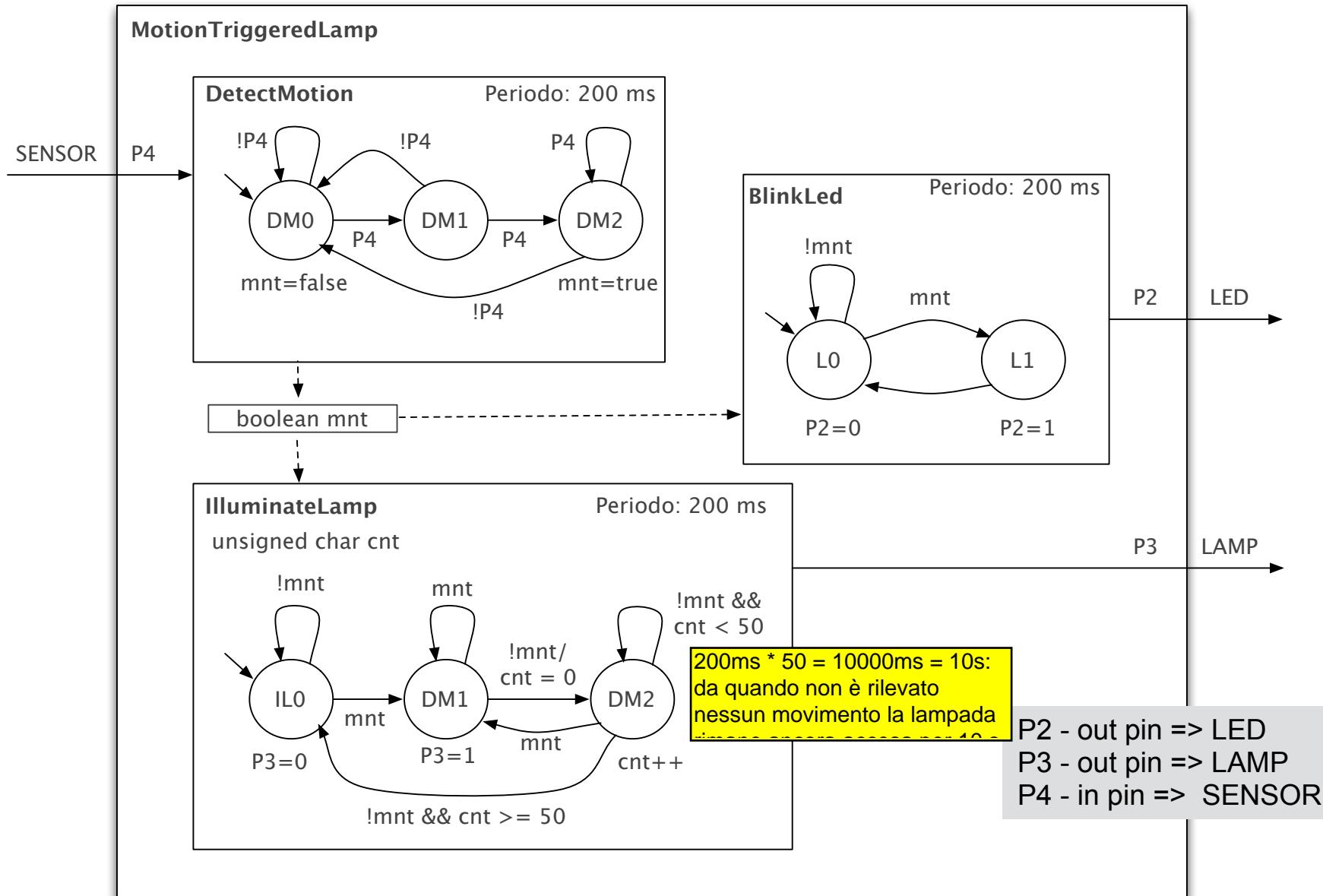
DIPENDENZE FRA TASK

- Non sempre è possibile suddividere task in sotto-task *totalmente indipendenti*
 - anzi, è più frequente il caso che i sottotask abbiano delle forme di dipendenza di vario tipo
 - **temporale**
 - es: un task T3 può essere eseguito solo dopo che sono stati eseguiti T1 e T2
 - **produttore/consumatore**
 - un task T1 ha bisogno di una informazione prodotta da un task T2
 - **relative a dati**
 - task T1 e T2 necessitano di condividere dati (in lettura, scrittura,..)
 - ...
- Il modello più semplice che si può utilizzare per rappresentare queste dipendenze in un sistema embedded è mediante **variabili condivise**

ESEMPIO: MOTION-TRIGGERED LAMP SYSTEM

- Sistema che deve accendere una luce a fronte del rilevamento di movimenti in un certo ambiente.
 - sistema dotato di un certo sensore di movimento collegato ad un certo pin
 - es: P4.
 - il movimento deve essere rilevato a fronte del fatto che P4 sia HIGH per due volte consecutive, considerando periodo (campionamento) di 200ms.
 - quando il movimento è rilevato, il sistema deve illuminare una lampada (led) collegato al P3, e mantenere la lampada settata per 10 secondi dopo l'ultimo rilevamento di movimento.
 - Il sistema dovrebbe inoltre far lampeggiare un led connesso a P2 con periodo di 200ms per tutto il tempo in cui il movimento è rilevato.
- Questo compito/comportamento complesso può essere facilmente espresso come composizione di 3 task
 - DetectMotion
 - IlluminateLamp
 - BlinkLed

ESEMPIO: MOTION-TRIGGERED LAMP SYSTEM



VANTAGGI

- Vantaggi rispetto al caso monolitico:
 - migliore separation of concerns
 - ogni task è focalizzato su un compito ben preciso
 - migliore comprensibilità
 - è più semplice capire il comportamento dei singoli moduli, meno complessi
 - migliore supporto al debugging
 - localizzazione più rapida degli errori
 - migliore supporto per modificabilità, estensione, riuso
 - il raffinamento del comportamento di un modulo non richiedono di mettere mano agli altri moduli

VARIABILI CONDIVISE, ATOMICITA' AZIONI E CORSE CRITICHE

- La presenza di variabili condivise fra i task richiede di porre attenzione al problema del loro accesso concorrente.
- E' noto che accessi concorrenti alla medesima variabile non danno problemi se sono in *lettura*, invece possono insorgere **corse critiche** nel caso di accessi concorrenti in **scrittura** o **lettura/scrittura**.
- Il problema si può risolvere in generale presupponendo che le transizioni di ogni macchina a stati siano eseguite **atomicamente**, come pure le eventuali azioni associate ad uno stato
 - in altre parole, *non deve esserci interleaving* delle azioni di cui si possono comporre transizioni o stati di macchine a stati concorrenti (nel medesimo sistema embedded).

COMUNICAZIONE FRA TASK

- In modello a task per macchine a stati sincrone, i task comunicano tipicamente mediante l'uso di *variabili globali condivise*
- In alternativa, come approccio avanzato è possibile adottare un modello a scambio di messaggi asincrono
 - ogni task ha la propria coda/buffer di messaggi
 - la comunicazione fra task avviene inviando messaggi in modo asincrono
 - approccio critico in termini di risorse di memoria richieste

INTERRUPT-DRIVEN COOPERATIVE SCHEDULER

- Similmente a quanto accade nei sistemi operativi, lo scheduler può essere invocato direttamente dall'interrupt handler del timer
 - ciò conduce ad una versione totalmente reattiva del sistema, ove nel loop non è necessario svolgere alcun compito
- In questo caso è possibile pensare di sfruttare facilmente le modalità di funzionamento a **basso consumo** del microcontrollore
 - per cui nel loop si pone il microcontrollore in stato di *sleep*, da cui esce all'arrivo di interruzioni
- Questo tipicamente porta a sistemi con *un consumo di energia nettamente inferiore*
 - quindi particolarmente significativo per sistemi embedded alimentati da batterie, che devono durare a lungo (es: reti di sensori)
- Occorre fare attenzione in questo caso però al tempo impiegato al micro per entrare e uscire da tale modalità
 - potrebbe introdurre *latenze* che non sono compatibili con il periodo scelto e con i task che devono essere eseguiti

INTERRUPT-DRIVEN COOPERATIVE SCHEDULER

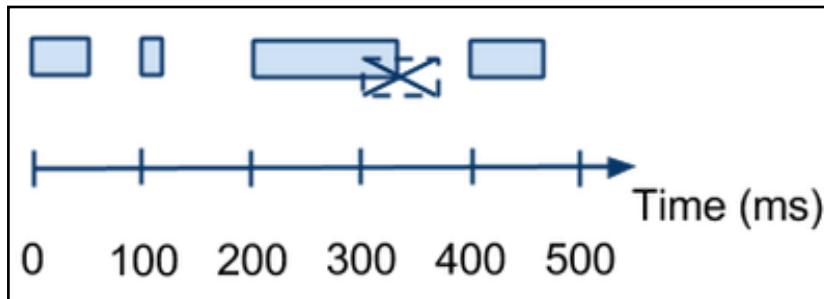
- Esempio: modulo-lab-2.3/sched-with-interrupt
 - scheduler interrupt driven
 - esempio blinking

PARAMETRO DI UTILIZZO CPU E SCHEDULING

- Nelle macchine sincrone abbiamo fatto l'assunzione (come modello) che le azioni eseguite avessero un *tempo di esecuzione nullo*, o comunque inferiore al periodo della macchina
- Per sistemi con molte azioni in uno stato o transizione, oppure con molti task che condividono lo stesso microcontrollore, tale assunzione può non essere sempre plausibile
 - è necessario considerare la durata delle azioni eseguite
- In tal caso, lo sviluppatore di sistemi embedded deve porre attenzione alla reale durata delle azioni in relazione al periodo per fare in modo che non ci siano malfunzionamenti

OVERRUN DEL PERIODO COL TIMER

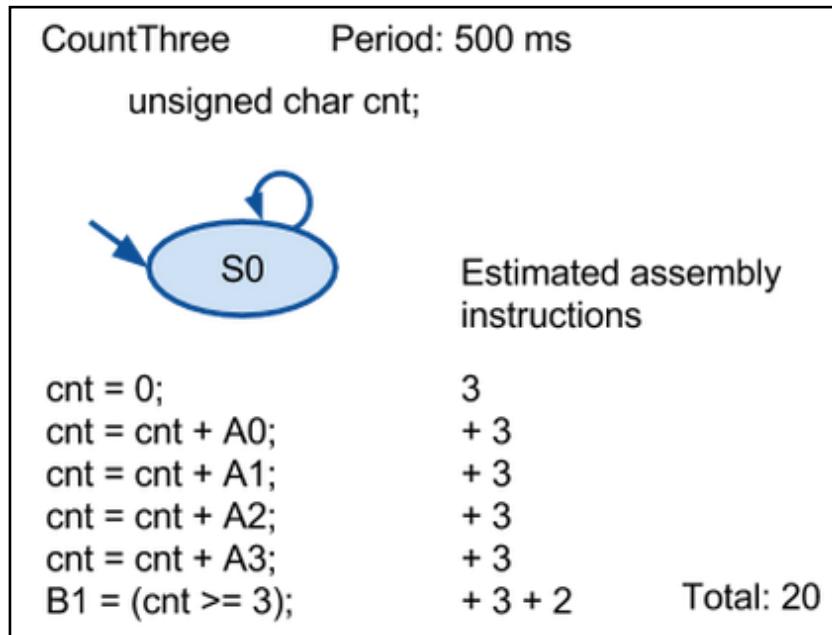
- Chiamiamo **eccezione di overrun** la situazione per la quale il tempo di esecuzione delle azioni oltrepassa il periodo
 - si parla di timer overrun nel caso in cui consideriamo uno scheduler interrupt-driven in cui le interruzioni possono essere annidate, per cui una nuova interruzione può essere generata prima che l'interrupt handler di un triggering precedente abbia concluso la propria esecuzione.
- Esempio di overrun



macchina sincrona con periodo 100ms.
A seconda dello stato vengono eseguite
azioni diverse. A $t = 200\text{ms}$, la durata
delle azioni eseguite eccede il periodo...

ANALISI DEL CODICE

- E' possibile analizzare manualmente il codice per stimare se una eccezione di overrun può accadere o meno.
- Si deve ipotizzare un modello di calcolo della durata complessiva delle azioni e verificare se nel caso peggiore supera o meno il valore del periodo.
- Esempio ([PES], p. 101)
 - task CountThree con un solo stato



DEFINIZIONE DEL PARAMETRO DI UTILIZZO DEL PROCESSORE E WORST-CASE EXECUTION TIME

- Il **parametro di utilizzo** di un micro-controllore è dato dalla percentuale di tempo in cui il micro è utilizzato per eseguire dei task:
$$U = (\text{tempo utilizzo per task} / \text{tempo totale}) * 100\%$$
- Nel modello che possiamo usare per fare l'analisi dell'utilizzo del processore possiamo trascurare le istruzioni che fanno parte dello scheduler.
- Il **Worst-Case-Execution-Time (WCET)** è dato dal tempo di esecuzione nel caso peggiore in termini di numero di istruzioni eseguite ad ogni periodo della macchina sincrona.
- Nel caso di più stati, si considera lo stato/transizione con la sequenza di condizioni/azioni più lunga
 - va considerato nel calcolo anche il tempo per valutare le condizioni

IN CASO DI OVERRUN

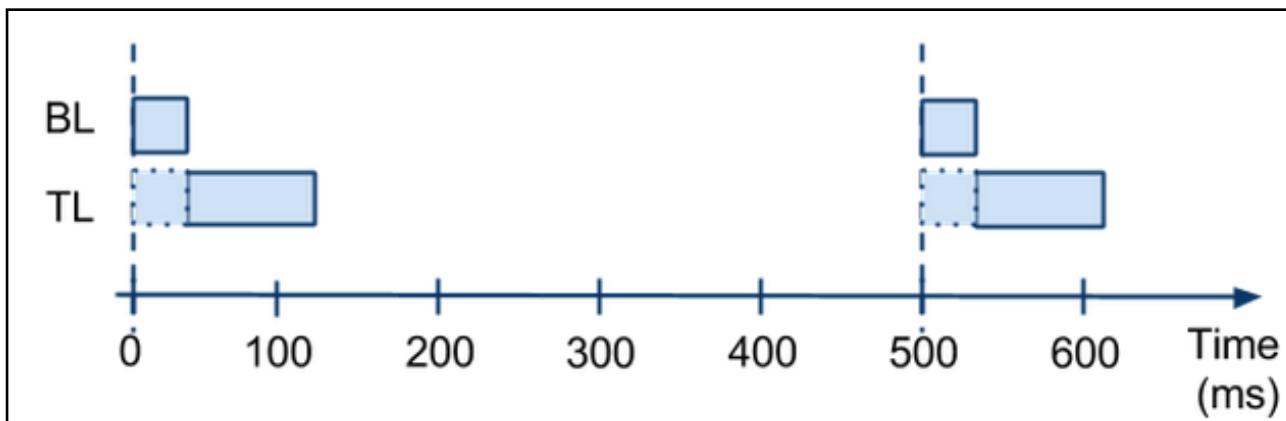
- Una analisi che conduce ad ottenere un parametro di utilizzo $U > 100\%$ indica che *si verificherà un eccezione di overrun*
- In tal caso, le azioni da intraprendere sono:
 - utilizzare macchine a stati con periodo più lunghi
 - ottimizzare la sequenza di condizioni/azioni più lunga
 - spezzare sequenze lunghe fra più stati
 - usare un microcontrollore più veloce
 - eliminare funzionalità dal sistema

CALCOLO WCET NEL CASO DI PIU' TASK

- L'analisi dell'utilizzo del processore è ancora più frequente nel caso di sistemi *a più task*
- Nel caso semplice di task con lo stesso periodo, allora il WCET si calcola come somma dei singoli WCET dei task.

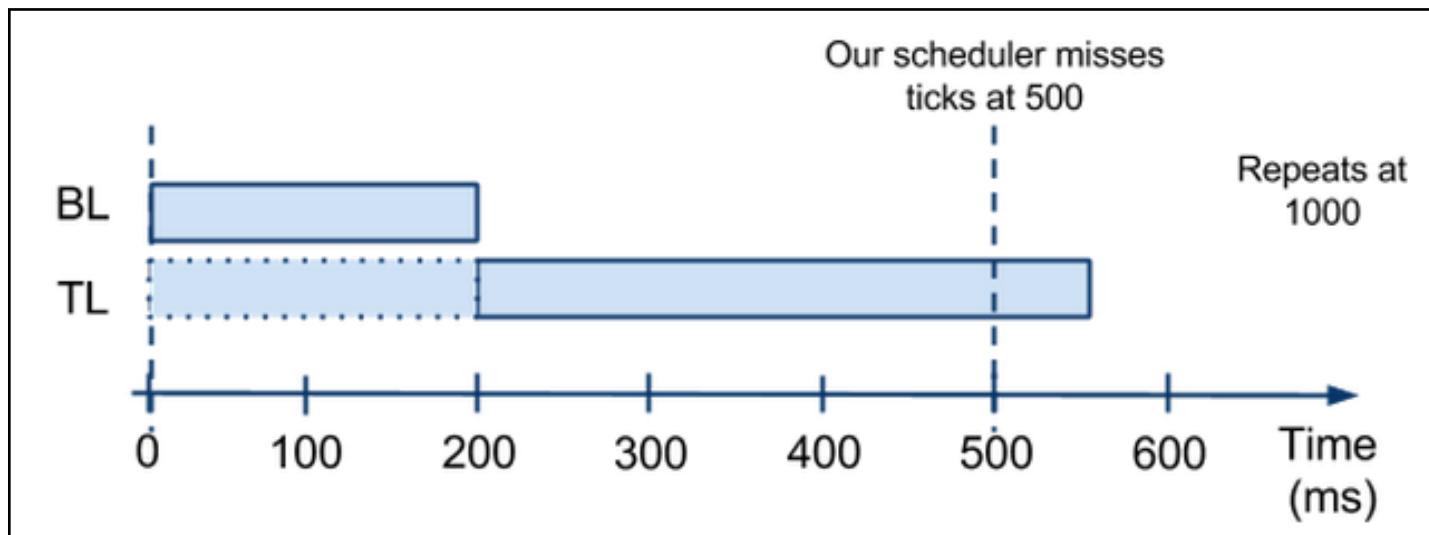
ESEMPIO LED-SHOW

- Esempio: sistema LedShow visto in precedenza
 - BlinkLed (BL) e ThreeLeds (TL), ognuno con periodo 500ms
 - esempio di analisi
 - supponiamo che il microcontrollore esegua 100 istruzioni/sec, ovvero 0.01 sec/istruzione
 - WCET per BL = 3 istruzioni => $3 \times 0.01 = 0.03$ sec
 - WCET per TL = 9 istruzioni => $9 \times 0.01 = 0.09$ sec
 - Per cui $U = (0.03+0.09)/0.5$ (essendo 0.5 il periodo) = 24 %



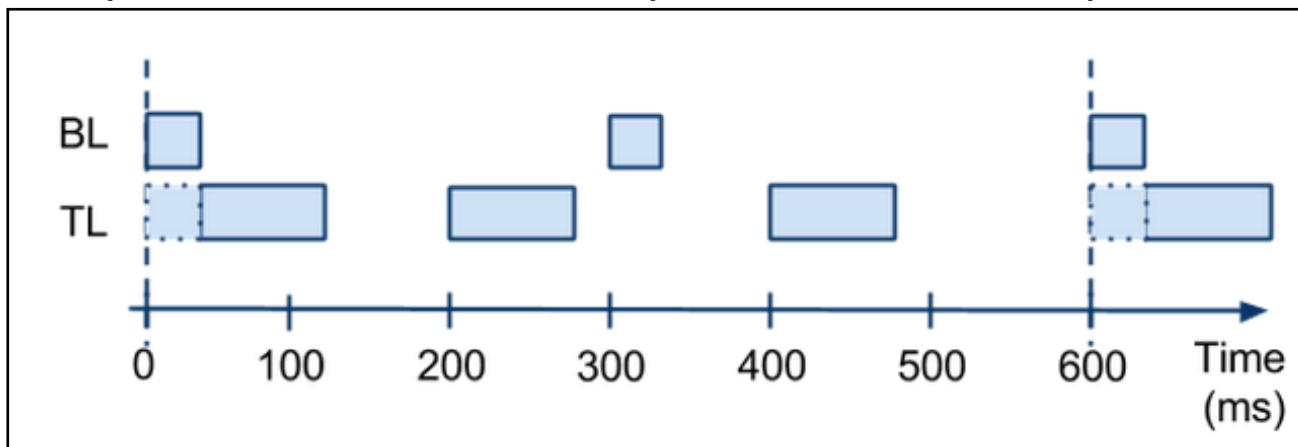
ESEMPIO LED-SHOW CON OVERRUN

- Supponiamo che WCET di BL sia invece 200 ms e che WCET di TL sia 350
- Allora $U = (0.200+0.35)/0.5 = 1.1 \Rightarrow \text{overrun}$



CASO CON PERIODI DIVERSI

- Nel caso più generale, i task possono avere *periodi diversi*.
- In tal caso, il calcolo del WCET deve essere fatto considerando degli **iper-periodi**
 - periodi che siano il minimo comune multiplo fra i periodi considerati
- Esempio LedShow con BL con periodo 300ms e TL periodo 200ms



- in questo caso, il pattern si ripete ogni 600 ms
- in 600 ms, BL esegue $600/300 = 2$ volte, mentre TL esegue $600/200 = 3$ volte
- l'utilizzazione nell'iper-periodo è pari a: $(2*20ms+3*90ms)/600ms = 55\%$

QUADRO RIASSUNTIVO

- In sintesi, il calcolo di U nel caso di T1...Tn task su un microcontrollore M è dato dai seguenti passi:
 - si determina il sec/instr rate R, ovvero quanti secondi servono per eseguire una istruzione del micro
 - si analizza ogni task Ti, determinando il suo WCET
 - determinando prima il numero massimo di istruzioni per tick e moltiplicando tale numero per R
 - si determina l'iper-periodo H come minimo comune multiple fra i periodi di tutti i task (T1.H, T2.H,...)
 - Il parametro di utilizzo è calcolato allora come:

$$U = ((H/T1.\text{period}) * T1.\text{WCET} + (H/T2.\text{period}) * T2.\text{WCET} + \dots) / H * 100$$

- $U > 100 \Rightarrow$ ci sarà overrun
- $U < 100 \Rightarrow$
 - singolo task: overrun non può accadere
 - task multipli: overrun può accadere

JITTER

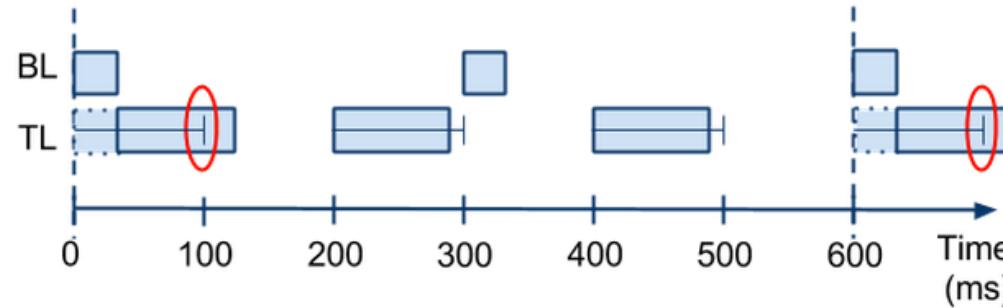
- Si definisce **jitter** il ritardo che intercorre dal momento che un task è pronto per essere eseguito e il momento in cui viene effettivamente eseguito
- Strategie di scheduling diverse possono portare a jitter diversi
 - esempio LedShow
 - se priorità a BL => il jitter di BL è 0 mentre il jitter di TL è 30ms
 - se priorità a TL => jitter di BL è 90ms, mentre per TL è 0
- In generale, dare priorità ai task con periodo più piccolo porta a minimizzare il jitter medio
 - nell'esempio: priorità a BL

DEADLINE

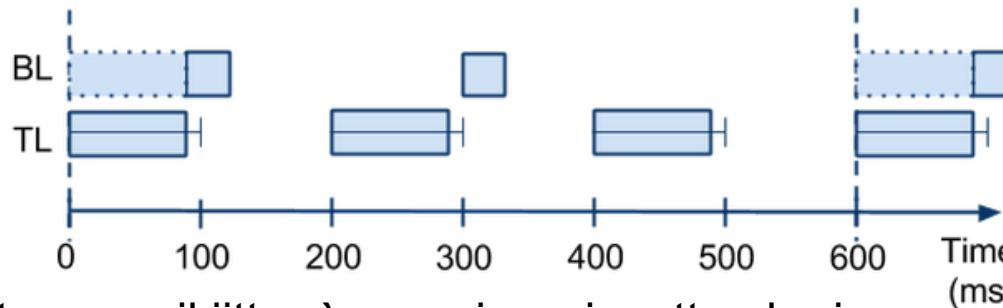
- Si definisce **deadline** (scadenza) l'intervallo di tempo entro il quale un task deve essere eseguito *dopo essere diventato ready*
- Se un task non viene eseguito entro la sua deadline, si ha una eccezione di *missed-deadline* che può portare - a seconda dei casi - a malfunzionamenti del sistema
- Se dato un task non viene specificata la sua deadline, allora per default la deadline è data dal periodo, ovvero il tempo entro il quale il task torna ad essere ready
- A seconda della strategia di scheduling adottata, si possono avere jitter diversi e soddisfare o meno le deadline

ESEMPIO

- Esempio LedShow
 - supponiamo che BL abbia periodo 300ms, WCET 30ms, deadline 300ms
 - TL abbia periodo 200ms, WCET 90ms e deadline 100ms
- se priorità a BL, la deadline per TL non è rispettata ad ogni prima occorrenza dell'iper-periodo:



- se priorità a TL:



- In questo caso, il jitter è maggiore rispetto al primo caso, però non ci sono missed deadline.

SCEDULING A PRIORITA' STATICHE E DINAMICA

- Per **priorità** in generale si intende l'ordinamento con cui devono essere eseguiti i task
 - quando ci sono più task ready, allora lo scheduler sceglie ed esegue quello prioritario
 - argomento importante per i Sistemi Operativi Real-Time
- In generale gli scheduler possono essere suddivisi in:
 - scheduler a priorità *statica*
 - scheduler a priorità *dinamica*

SCEDULING A PRIORITA' STATICÀ

- In uno scheduler a priorità statica, la priorità viene assegnata ad ogni task prima che i task siano eseguiti e queste non cambiano durante l'esecuzione
- Un approccio frequente in questo caso è di assegnare la priorità maggiore ai task con la deadline più piccola (*shortest-deadline first*)
 - l'intuizione in questo caso è che i task possono non rispettare le deadline se non eseguiti prima, come nell'esempio BL/TL
- Se tutti i task hanno le deadline pari ai periodi dei task, allora questo approccio risulta nell'assegnare la priorità maggiore ai task con il periodo più piccolo (*shortest-period first*).
 - questo scheduling è chiamato **rate-monotonic scheduling (RMs)**
 - chiamato così perché le priorità sono basate sul rate del task, con priorità assegnate in modo monotono crescente
- Un altro approccio assegna la priorità al task con *WCET* più piccolo (*shorted-WCET first*)
 - questo approccio è utile per ridurre il jitter, tuttavia richiede per essere applicato la conoscenza del WCET - che non sempre è noto

ASSEGNAMENTO PRIORITA'

- Nello scheduler visto nel modulo, la priorità dei task era data implicitamente dall'ordine in cui erano inseriti nell'array dei task da eseguire
 - si determina la priorità inserendo i task in un certo ordine

SCEDULING A PRIORITA' DINAMICA

- Questo tipo di scheduler determina le priorità dei task man mano che il programma viene eseguito, per cui le priorità assegnate possono cambiare nel tempo
- L'approccio più utilizzato in questo caso è denominato ***earliest-deadline-first (EDF)***
 - l'intuizione ragionevole è che i task ready che hanno la deadline più vicina siano quelli che possono violarla se non vengono eseguiti subito
- La schedulazione a priorità dinamiche tipicamente riducono jitter e missed deadline
 - tuttavia richiedendo scheduler più complessi e costosi dal punto di vista delle computazioni
- EDF è usato spesso anche quando i task sono eseguiti in reazione ad eventi, anziché time-oriented

PREEMPTIVE E COOPERATIVI

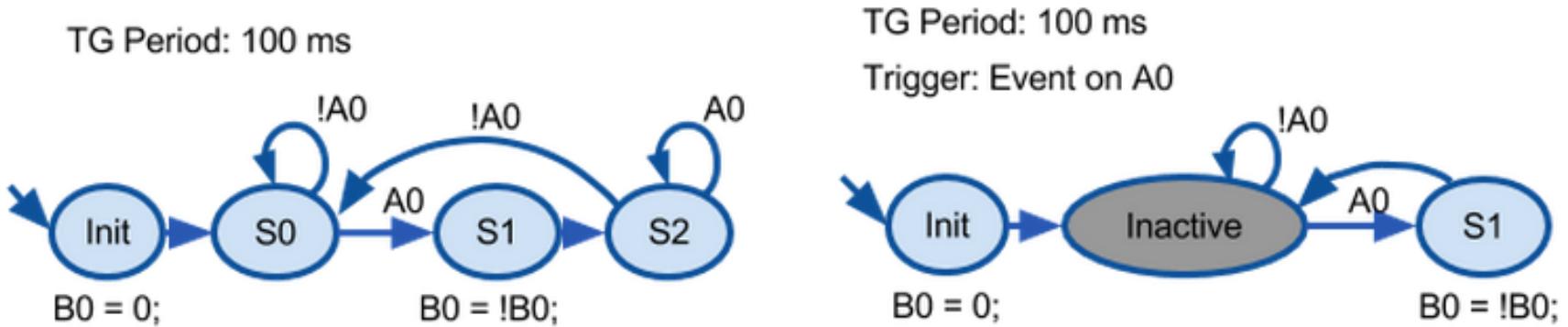
- Come già menzionato in precedenza, uno scheduler può essere **preemptive o cooperativo**
 - nel caso cooperativo, una volta selezionato un task esegue fino al completamento
 - semantica *run-to-completion*
 - nel caso preemptive, è possibile togliere il processore ad un task in esecuzione prima che abbia completato
 - più difficili da implementare
- Il caso preemptive è tipicamente usato quando i task sono organizzati in termini di processi, con ciclo infinito nel caso in cui siano periodici.
- L'organizzazione a stati vista con le macchine a stati finiti sincrone può essere realizzata semplicemente con scheduling cooperativo
 - ad ogni tick della macchina vengono eseguite atomicamente le azioni associate all'eventuale transizione/nuovo stato

MACCHINE A STATI SINCRONE EVENT-TRIGGERED

- Le macchine a stati sincrone viste fino ad ora per rilevare cambiamenti sull'input eseguono campionamenti periodici (*polling*)
- Un approccio per ridurre l'utilizzo della CPU (sprecato in attese attive) consiste nello sfruttare supporti HW che permettono di rilevare cambiamenti su pin di input e quindi *generare interruzioni*, chiamando conseguentemente apposite interrupt handler
- Quindi il micro può essere fatto transitare in uno stato di inattività fin quando non viene rilevato l'evento che le fa transitare in uno stato attivo
- A livello di modello, è possibile modellare questa situazione con l'introduzione di uno speciale stato "inactive", in cui la macchina è in idle
- *Alla ricezione di un evento relativo ad un pin in ingresso*, la macchina esce dallo stato inactive e inizia ad eseguire il comportamento sincrono periodico, fino ad entrare nuovamente in uno stato inactive

ESEMPIO

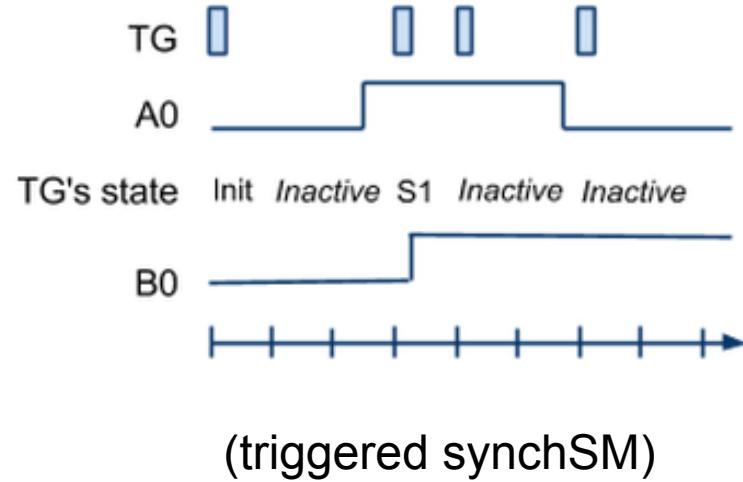
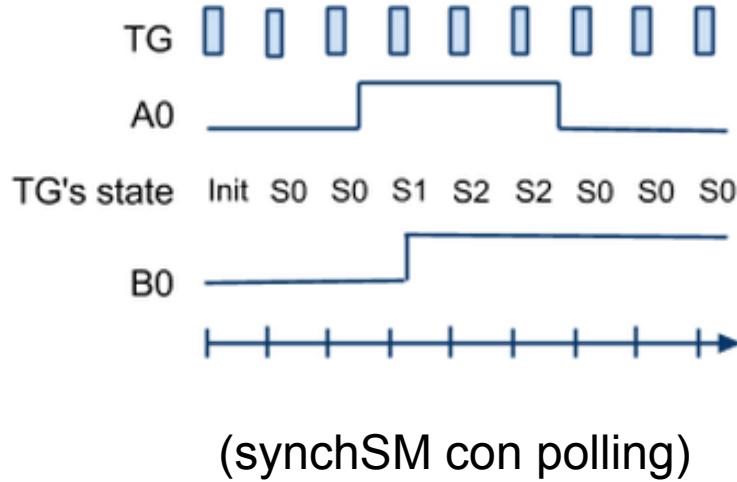
- Esempio:
 - sistema che cambia una certa uscita B0 ogni volta che un pulsante connesso al pin A0 è premuto
 - due macchine a stati, una sincrona e una sincrona event-triggered



- Il detecting dell'evento su A0 non è responsabilità della macchina a stati
 - analogamente al tick del clock: quando avviene, la macchina a stati reagisce e valuta quale transizione è abilitata
- Nella terminologia relativa allo scheduling, i task triggered da eventi sono chiamati *task aperiodici*

RIDUZIONE UTILIZZO

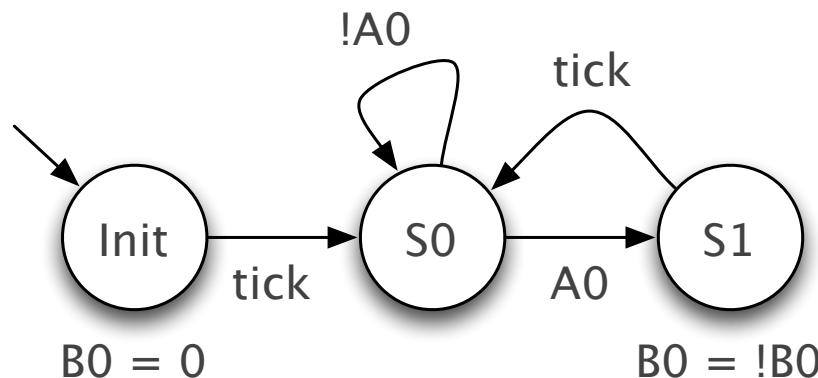
- L'integrazione di approcci event-triggered può ridurre in modo significativo l'utilizzo del processore
- Nell'esempio:



- Nel caso triggered, il comportamento periodico si ha solo quando la macchina è in uno stato attivo

MACCHINE A STATI EVENT-DRIVEN

- E' possibile usare un modello alternativo per integrare modello a stati ed interruzioni/eventi
- Si abbandona il modello sincrono e si assume un modello asincrono, dove le transizioni di stato sono determinate da guardie eventi/condizioni
- Tuttavia fra gli eventi si assumono quelli del timer
 - per cui il modello sincrono è modellato come macchina che reagisce ad eventi periodici "tick" a frequenza prefissata



- In questo modo non è necessario inserire lo stato "inattivo" esplicitamente nel diagramma degli stati

TASK PERIODICI E APERIODICI

- Fino ad ora sono stati considerati solo task *periodici*
 - task vengono mandati in esecuzione periodicamente dallo scheduler
- Più in generale, può essere utile considerare l'esecuzione anche di task *aperiodici* o *sporadici*
 - task che vengono mandati in esecuzione a tempi arbitrari

GESTIONE TASK APERIODICI

- La gestione di task aperiodici complica in modo significativo lo scheduler
 - caso tipico per i Sistemi Operativi Real-Time
 - modulo 3.1 futuro
 - scheduler con metodi per inserire e rimuovere dinamicamente un task
- Approcci semplificati (e meno efficienti)
 - task aperiodici realizzato come task periodici con stato “idle”
 - task con meta-stato attivo o non-attivo
 - selezionato solo dallo scheduler se si trova in uno stato attivo
 - introdotti staticamente e attivati/disattivati da altri task

BIBLIOGRAFIA

- [PES] “Programming Embedded Systems - An introduction to Time-Oriented Programming”. Vahid et al.

Sistemi Embedded e IoT

Ingegneria e Scienze Informatiche - UNIBO

a.a 2020/2021

Docente: Prof. Alessandro Ricci

[modulo-3.1]

SISTEMI EMBEDDED
BASATI SU SOC E RTOS

SOMMARIO DEL MODULO

- Sistemi SoC
- Sistemi Operativi Embedded e Real-Time
 - richiami sui Sistemi Operativi
 - caratteristiche, architettura, programmazione
 - approfondimento sullo scheduling

DAI MICROCONTROLLORI AI SOC

- La prima parte del corso è stata focalizzata su *microcontrollori* come sistemi embedded
 - no sistema operativo
- Ora consideriamo sistemi embedded basati su SoC
 - con memoria e potenza CPU tali da poter ospitare un sistema operativo
 - Sistemi Operativi Embedded e Real-time

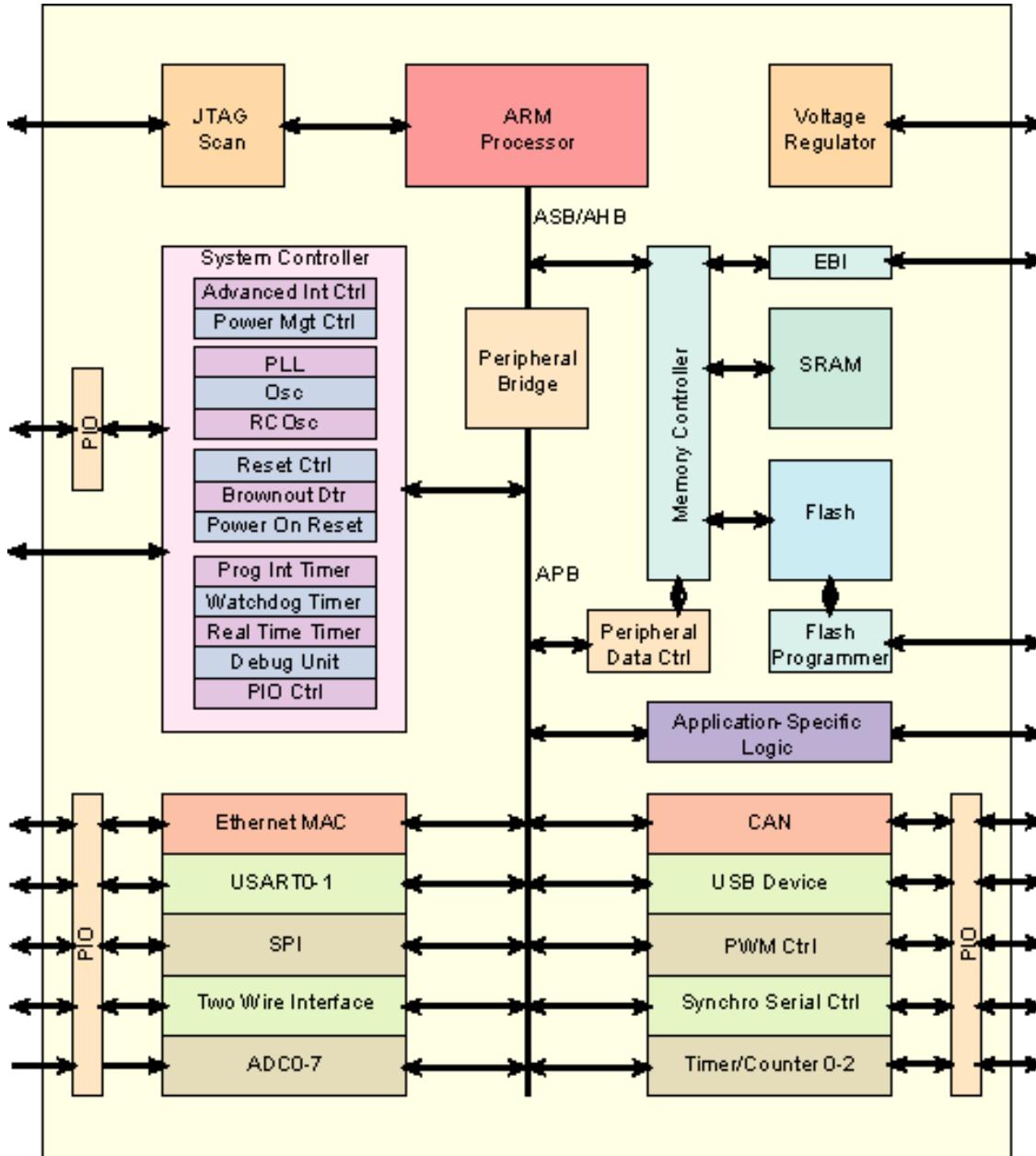
SoC e SINGLE-BOARD CPU

- **SOC = System-on-a-Chip**
 - un chip che incorpora un sistema completo
 - tipicamente usati per creare delle single-board CPU
- Esempi:
 - BROADCOM BCM2837 64bit ARMv8 Cortex A53 Quad Core (1.2Ghz)
 - usato in Raspberry Pi 3
 - ARM Sitara AM335x SoC - including ARM Cortex-A8 processor
 - usato in BeagleBone, Arduino Tre
 - Texas Instruments OMAP3530
 - ...

SoC COME SISTEMA COMPLETO

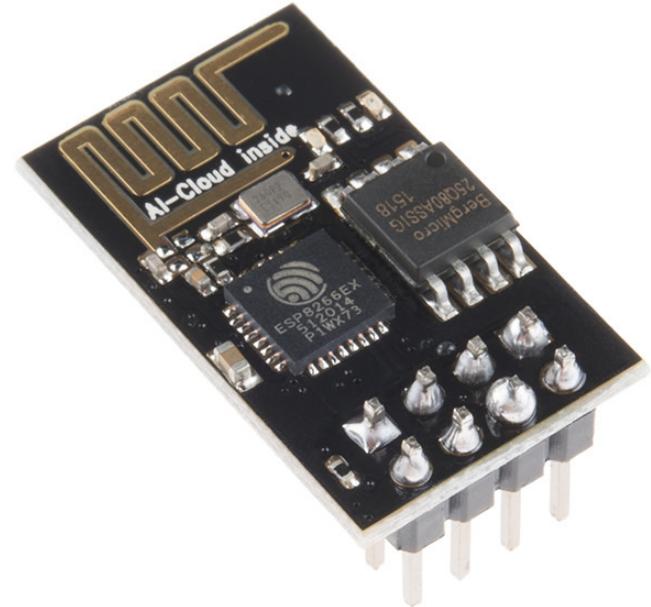
- SoC comprende:
 - uno o più processori/core
 - inclusi microcontrollori, microprocessori e DSP
 - un modulo di memoria contenente uno o più blocchi di tipo ROM, RAM, EEPROM o Memoria flash.
 - un generatore di clock
 - periferiche come contatori, orologi e altro
 - connettori per interfacce standard come USB, Ethernet, USART, I2C, SPI
 - DAC e ADC
 - regolatori di tensione e circuiti di gestione dell'alimentazione

SoC

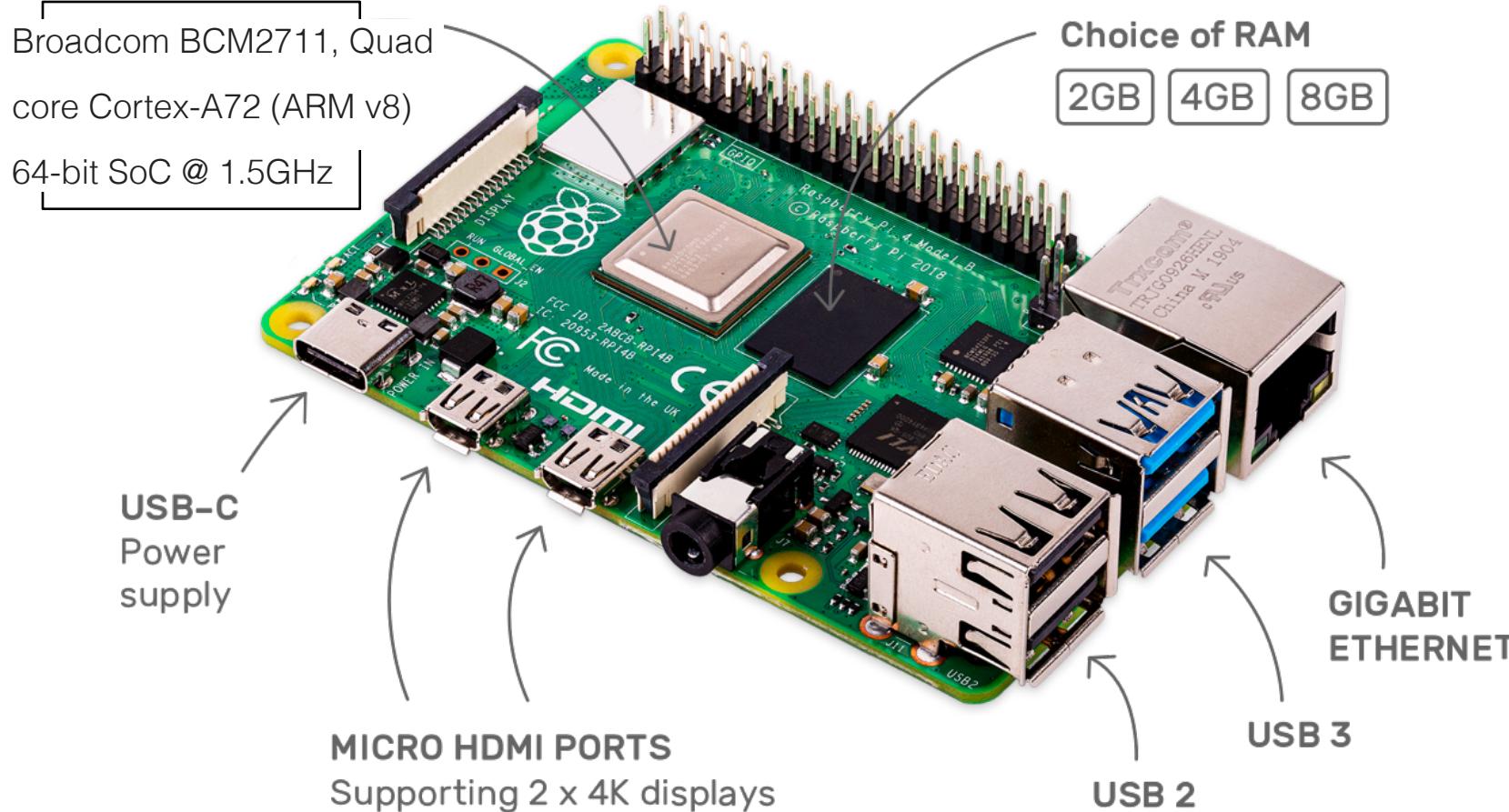


ESP 8266

- Dispositivo pensato per IoT
- Hardware
 - Processor: L106 32-bit RISC microprocessor core 80 MHz*
 - 64 KiB of instruction RAM, 96 KiB of data RAM
 - 16 GPIO pins
 - SPI, I²C, I²S + UART
 - 10-bit ADC
 - IEEE 802.11 b/g/n Wi-Fi
- SDK e firmware diversi, fra cui
 - GCC toolchain, Wiring / Arduino, NodeMCU Lua-based, MicroPython Python based, ESP-Open-RTOS Open source FreeRTOS-based...



RASPBERRY PI 4



SISTEMI OPERATIVI EMBEDDED E REAL-TIME

- I sistemi embedded basati su SoC hanno risorse sufficienti (in termini di capacità computazionale e memoria) da poter montare un vero e proprio sistema operativo
- Si parla di Sistemi Operativi *Embedded* e *Real-Time (RTOS)*
 - sistemi operativi progettati per essere usati nei sistemi embedded

SISTEMI OPERATIVI: RICHIAAMI DI ALCUNI CONCETTI ESSENZIALI

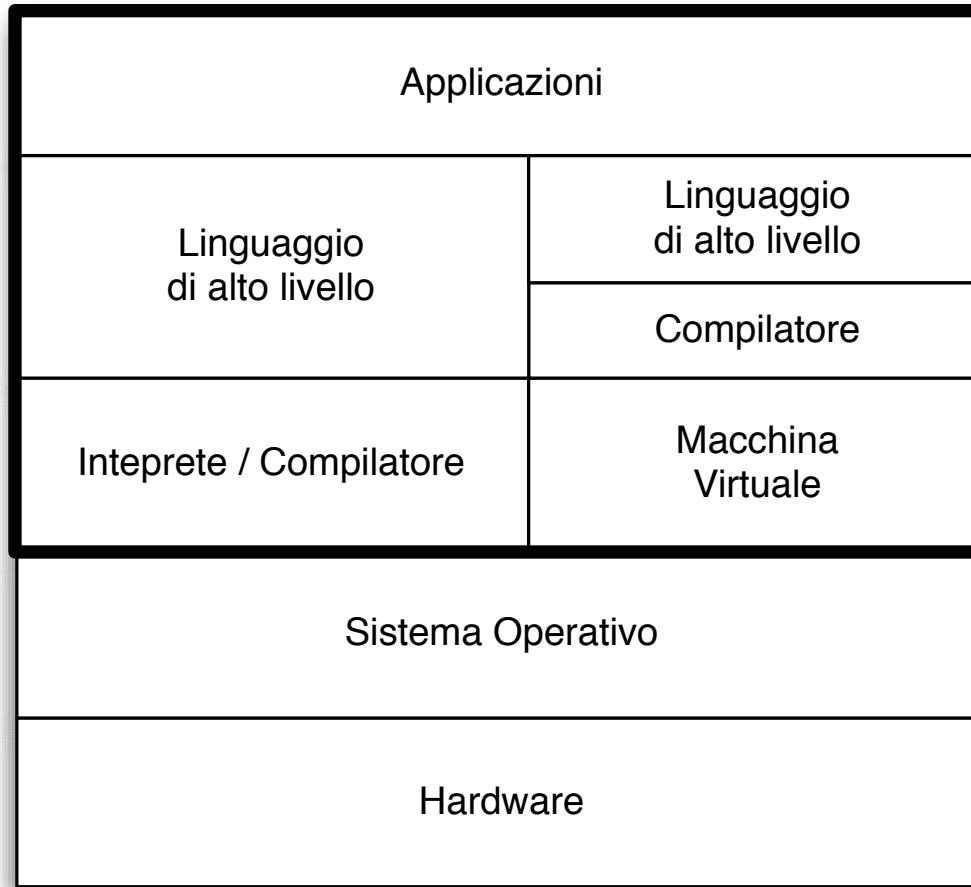
SISTEMI DI ELABORAZIONE: UNO SGUARDO D'INSIEME

- Sistemi di elaborazione moderni
 - software + hardware
 - a livello più alto: linguaggi di programmazione di alto livello, applicazioni articolate, sistemi software complessi
 - a livello più basso: circuiti elettronici, porte logiche, logica binaria
 - complessità della progettazione e realizzazione
- Organizzazione a livelli
 - decomposizione in **moduli**
 - ogni modulo confina un certo insieme di funzionalità
 - processo *di astrazione*
 - fornisce una interfaccia per fruire delle funzionalità
 - nasconde dettagli implementativi
 - organizzazione **gerarchica**
 - ogni modulo corrisponde ad un livello

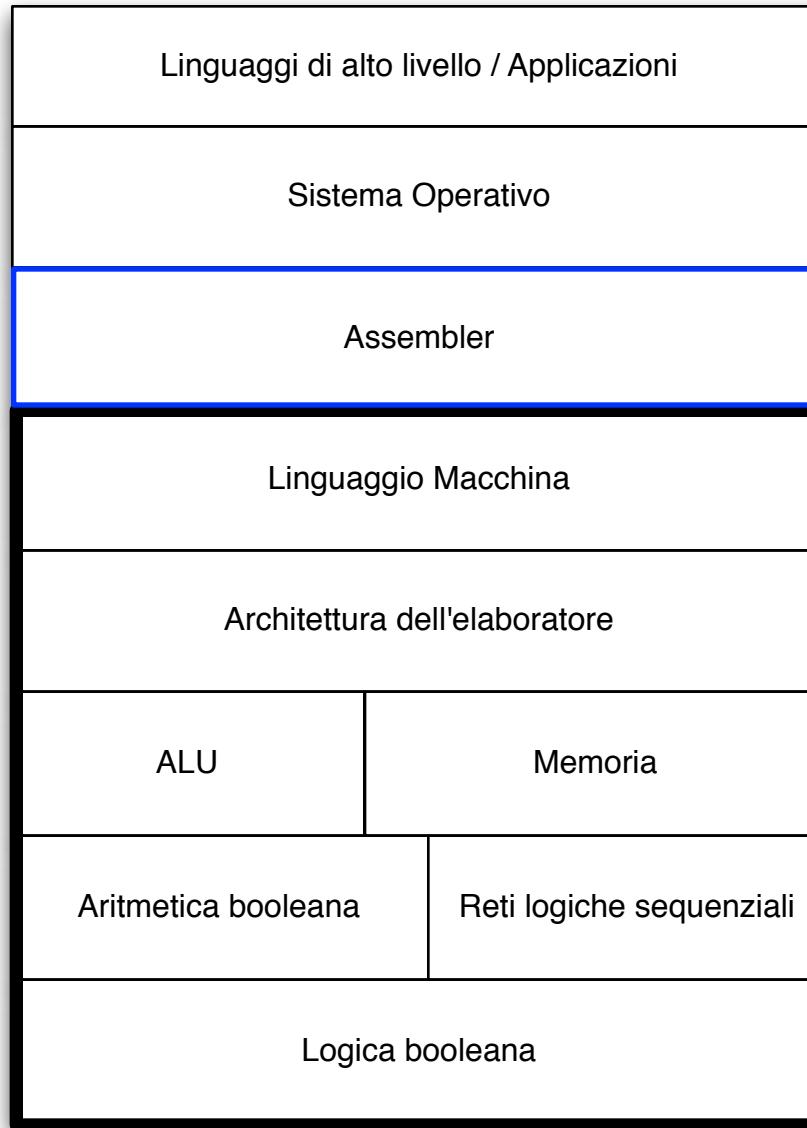
LIVELLI



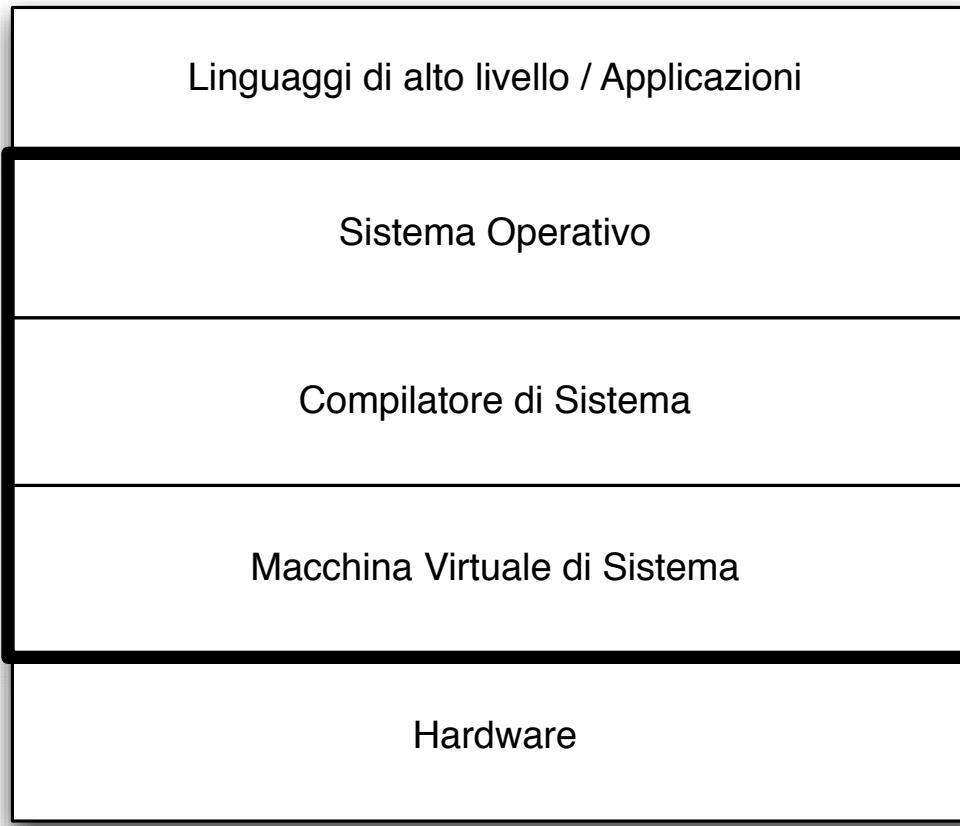
ZOOM LIVELLO LINGUAGGI/APPLICAZIONI



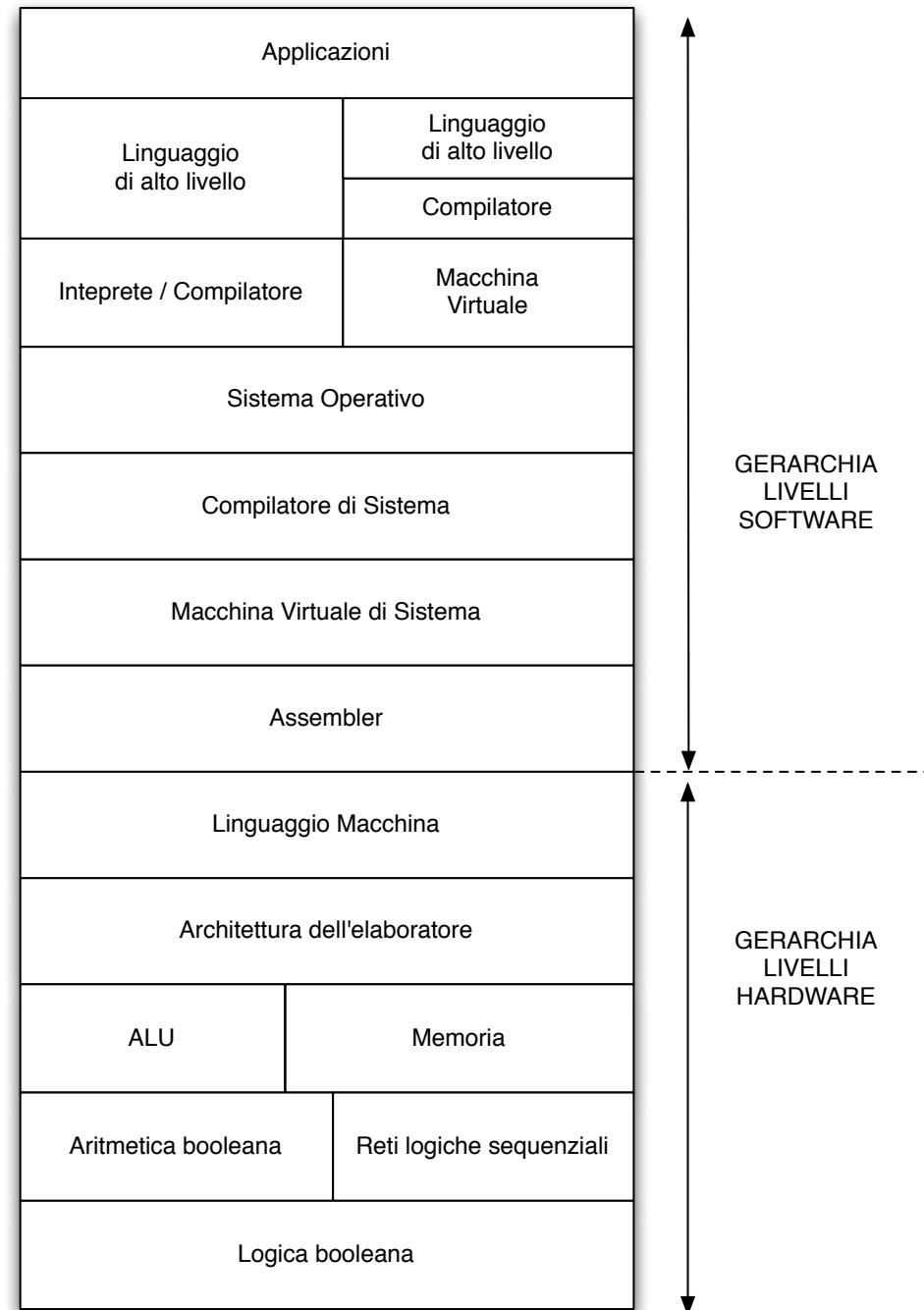
ZOOM LIVELLI HARDWARE



ZOOM LIVELLI VIRTUALIZZAZIONE



VISTA COMPLESSIVA DEI LIVELLI



LIVELLI E INTERFACCE: DETTAGLIO

MAIN INTERFACES

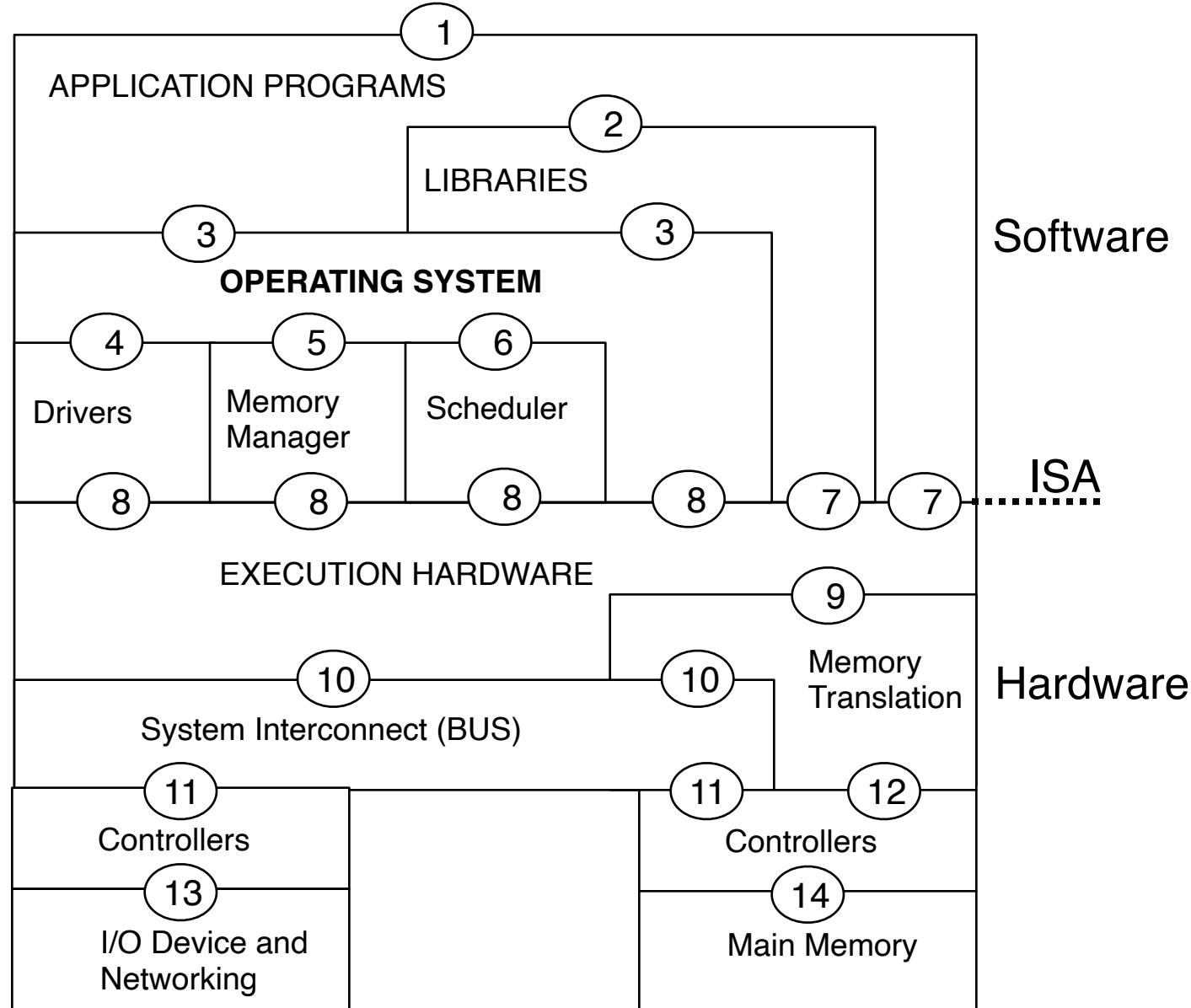
ISA

- System ISA: 8
- User ISA: 7 + 8

Application Binary Interface (**ABI**): 7+3

System Call
Interface: 3

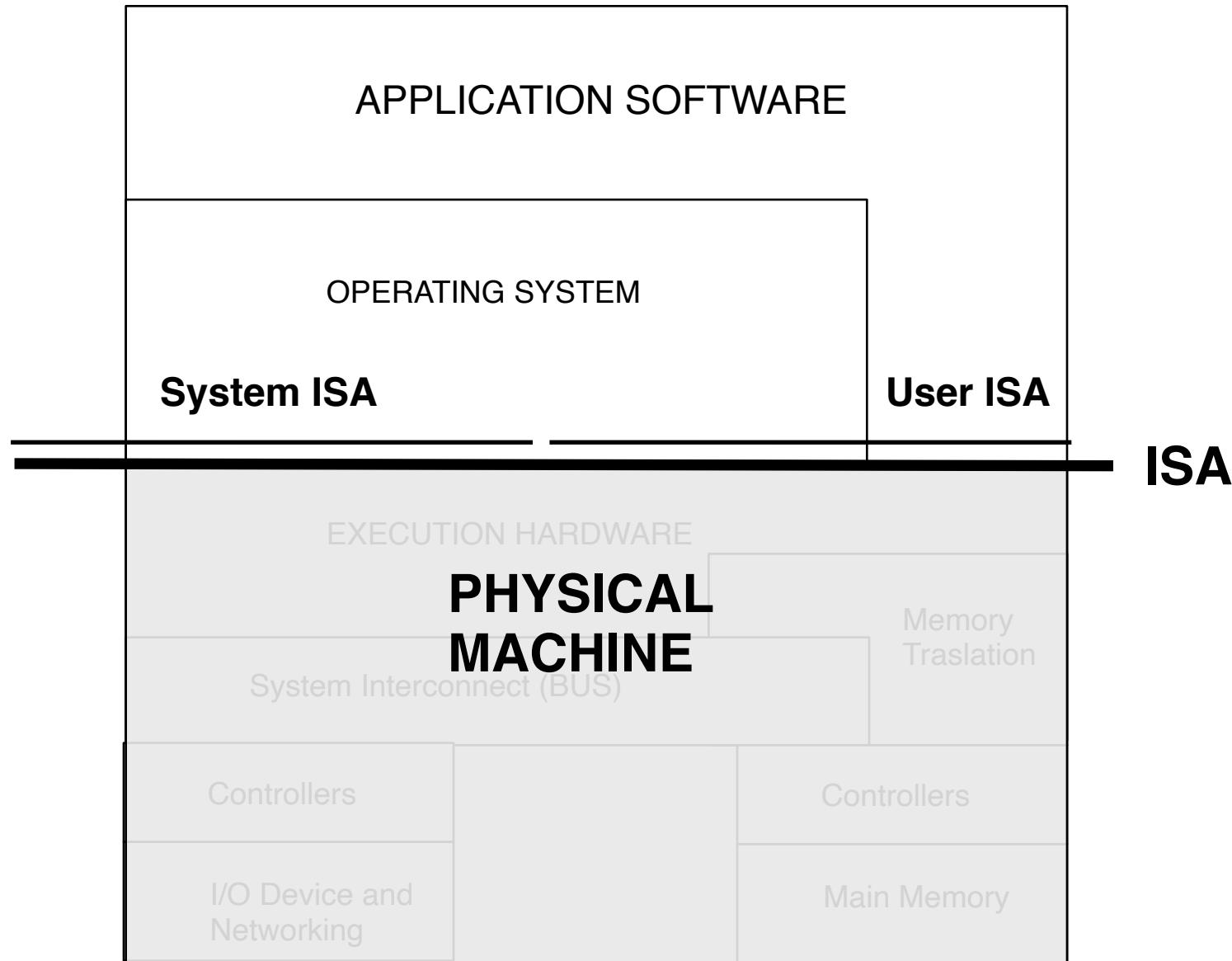
Application Programming Interface (**API**) for High-Level Languages (HLL) (es: C, Java,...): 2



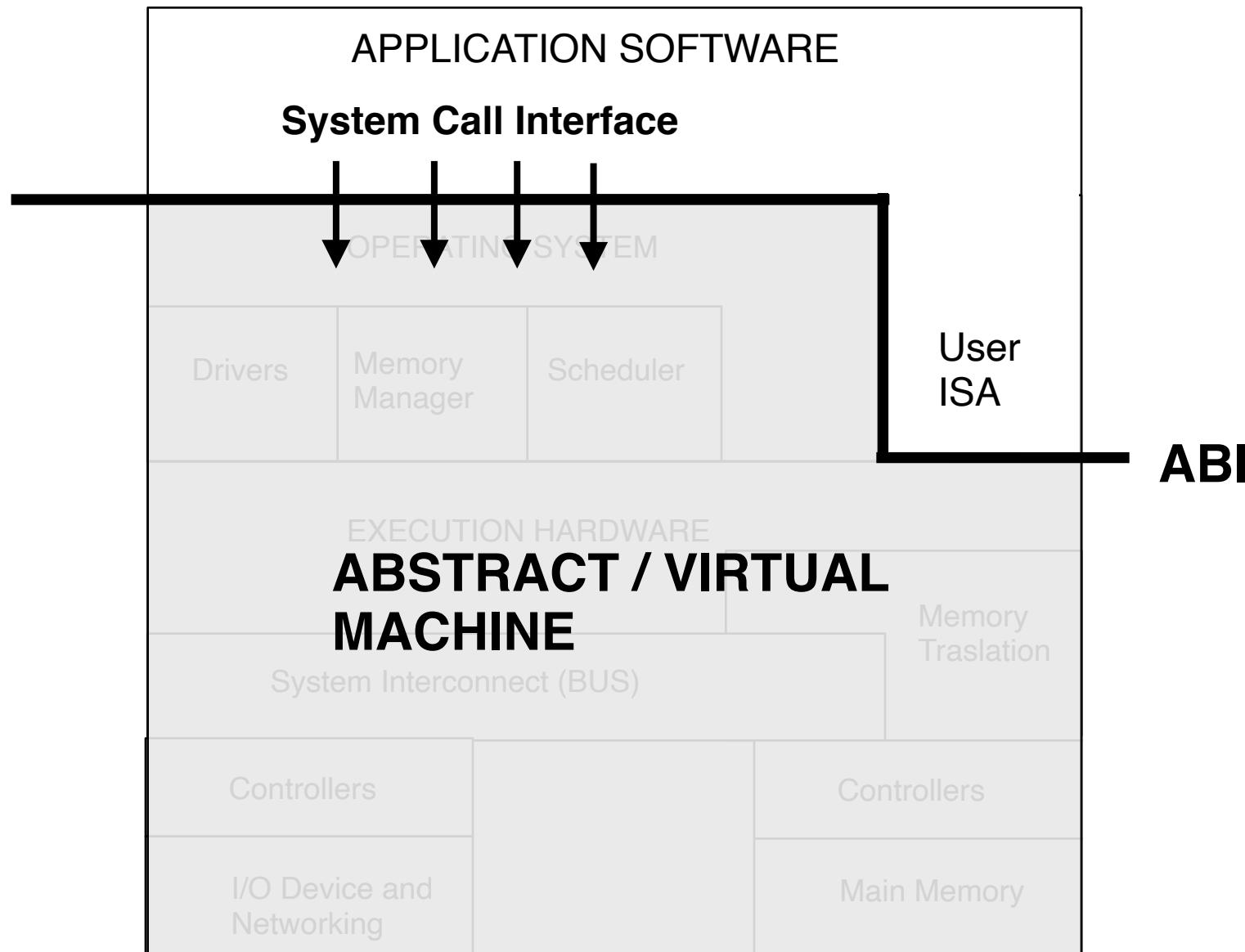
ISA, ABI e System Call Interface

- Instruction Set Architecture (**ISA**)
 - interfaccia che separa livelli HW e SW
 - *user ISA*
 - tutti gli aspetti e parti (es: istruzioni) visibili ai programmi utente
 - interfaccia 7
 - *system ISA*
 - tutti gli aspetti e parti visibili e utilizzabili solo dall'OS (es: istruzioni *privilegiate*)
 - interfaccia 8
- Application Binary Interface (**ABI**)
 - interfaccia fornita ai programmi per accedere alle risorse HW e ai servizi del sistema
 - due componenti principali: user ISA + System Call Interface
- System Call interface
 - insieme di operazioni che l'OS mette a disposizione ai programmi come servizi fondamentali
 - trasferimento del controllo, cambio livello di privilegio

MACHINE INTERFACE: ISA

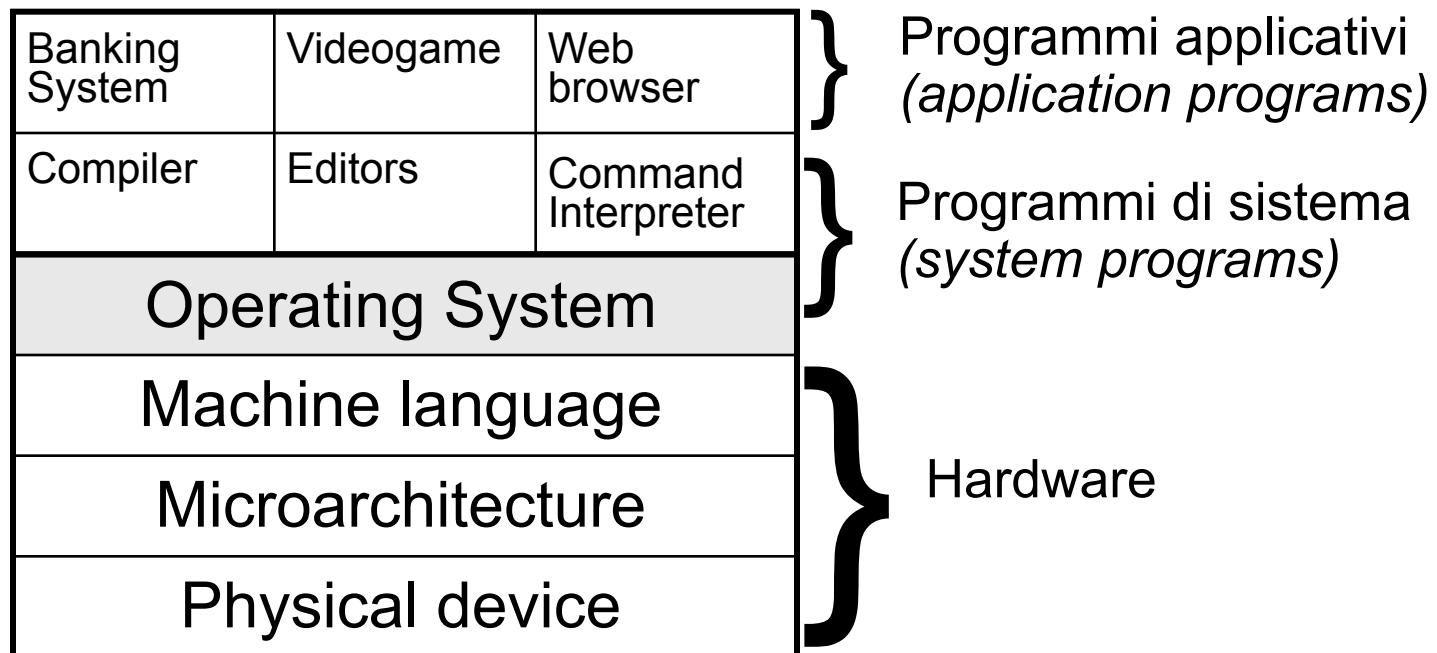


MACHINE INTERFACE: ABI



SISTEMI OPERATIVI: INQUADRAMENTO

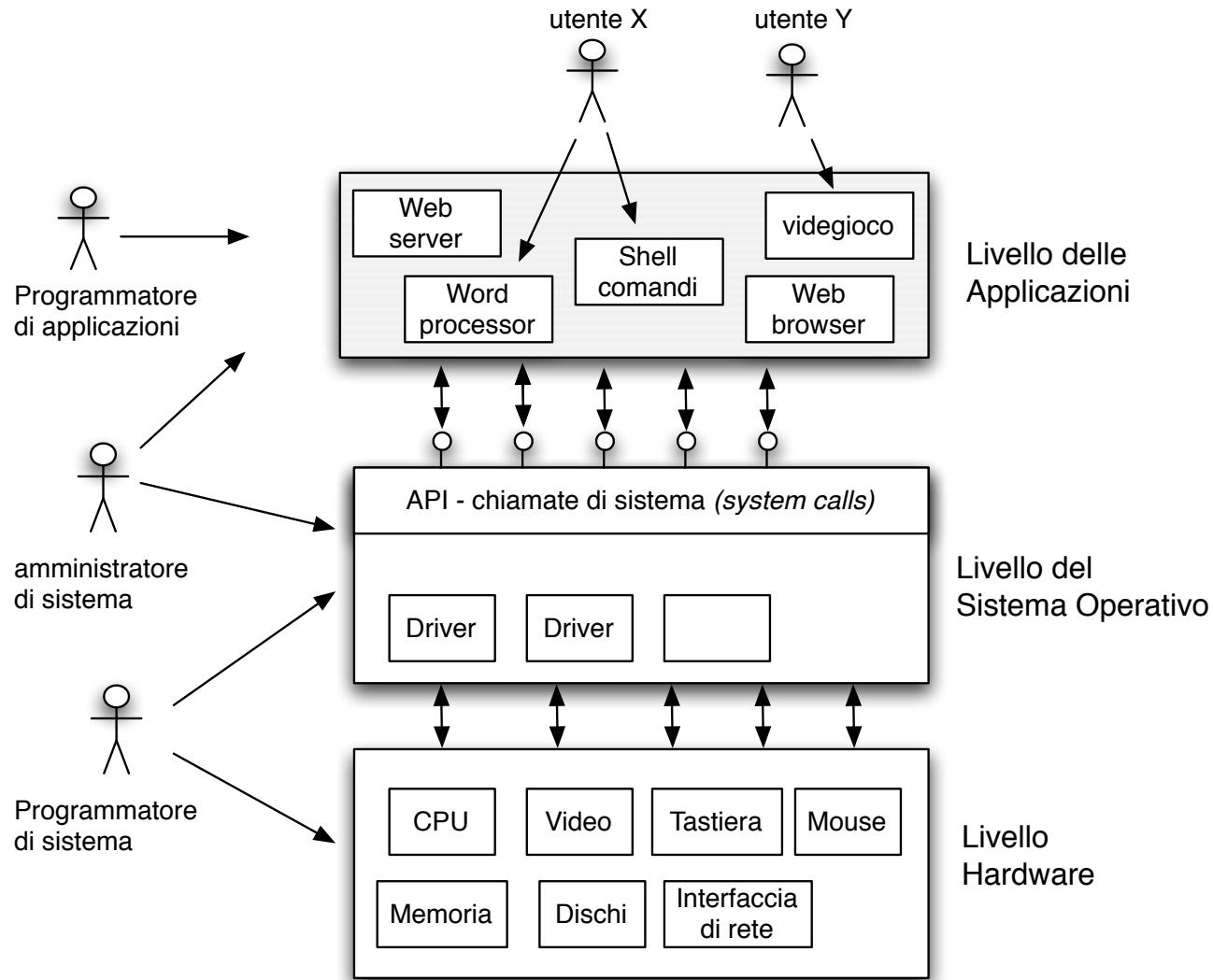
- Il **sistema operativo** (S.O) - in inglese *operating system* (O.S.) - è quella parte software di un sistema di elaborazione che controlla l'esecuzione dei programmi applicativi e funge da *intermediario* fra questi e la macchina fisica (hardware)



OBIETTIVI PRINCIPALI DI UN S.O.

- Eseguire programmi degli utenti e controllare la loro esecuzione, in particolare l'accesso alla macchina fisica
 - S.O. come **extended / virtual machine**
 - funzione di astrazione, controllo e protezione
- Rendere agevole ed efficace l'utilizzo delle risorse del computer
 - S.O. come **resource manager**
 - funzione di ottimizzazione
- *Abilitare e coordinare le interazioni fra applicazioni, utenti, risorse*
 - più applicazioni in esecuzione *concorrente*
 - più utenti che condividono e usano simultaneamente il sistema

SISTEMA DI ELABORAZIONE: LIVELLI ASTRATTI



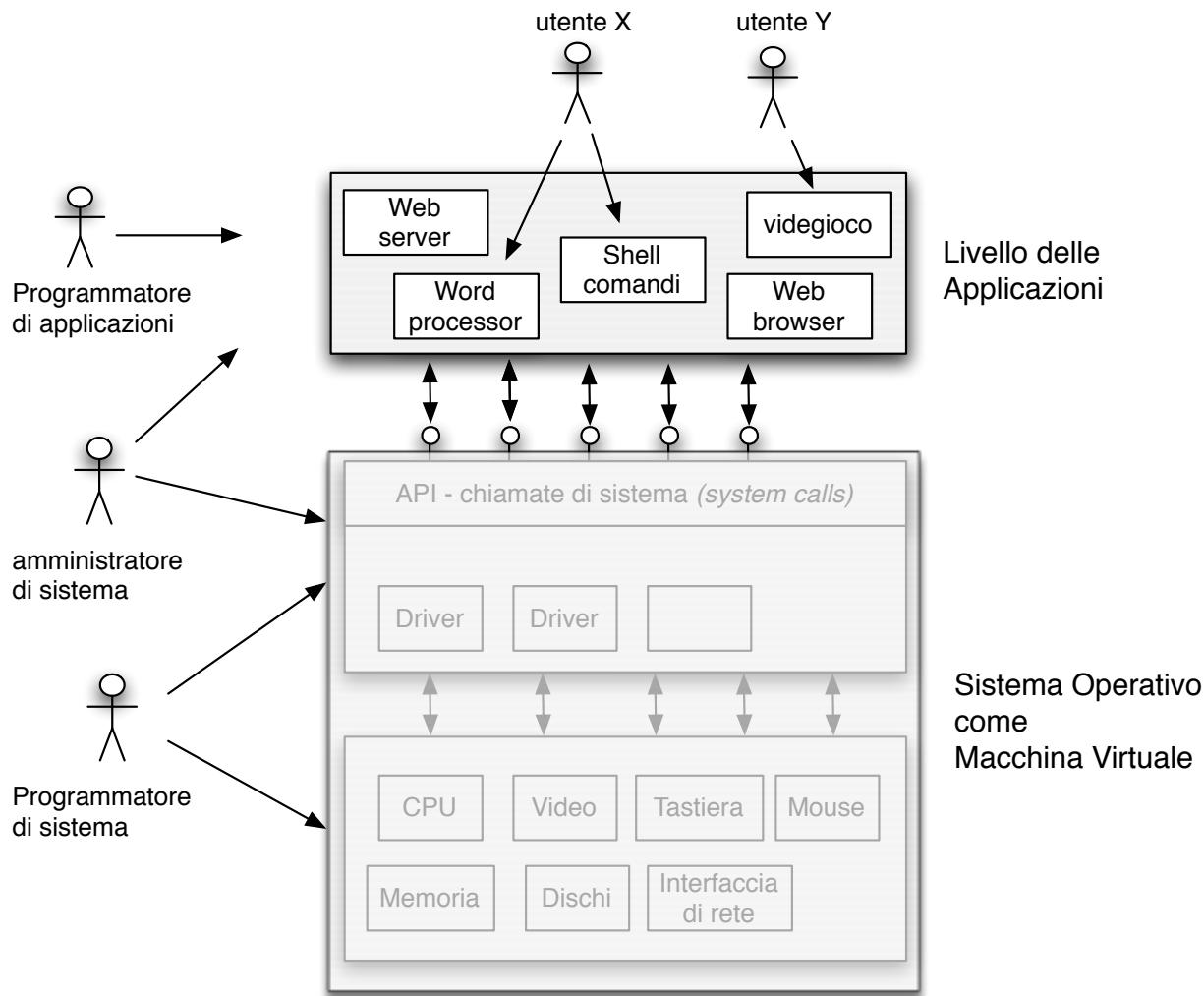
RUOLO DI MEDIAZIONE

- Il sistema operativo media l'interazione fra livello applicazione e livello hardware, controllando e coordinando l'accesso e l'uso del livello hardware richiesto dal livello applicazione, e *rendendo i due livelli il più possibile indipendenti fra loro*
 - fattorizzazione delle esigenze comuni alle applicazioni in servizi che le applicazioni possono direttamente usare astraendo dalla loro implementazione o realizzazione
- Ruolo fondamentale delle **API** (*Application Programming Interface*)
 - *interfaccia* di programmazione per i programmi
 - insieme di funzioni o *primitive* di base chiamate **chiamate di sistema** (*system calls*)

S.O. COME MACCHINA VIRTUALE

- Il sistema operativo maschera ai programmi applicativi la struttura reale della macchina fisica, facendo veder loro una **macchina virtuale (o astratta)**
 - una macchina più semplice e ad un livello di astrazione maggiore rispetto al livello fisico
 - accessibile mediante le chiamate di sistema che nell'insieme costituiscono *l'interfaccia della macchina virtuale*
 - vista *top-down*

S.O. COME MACCHINA VIRTUALE



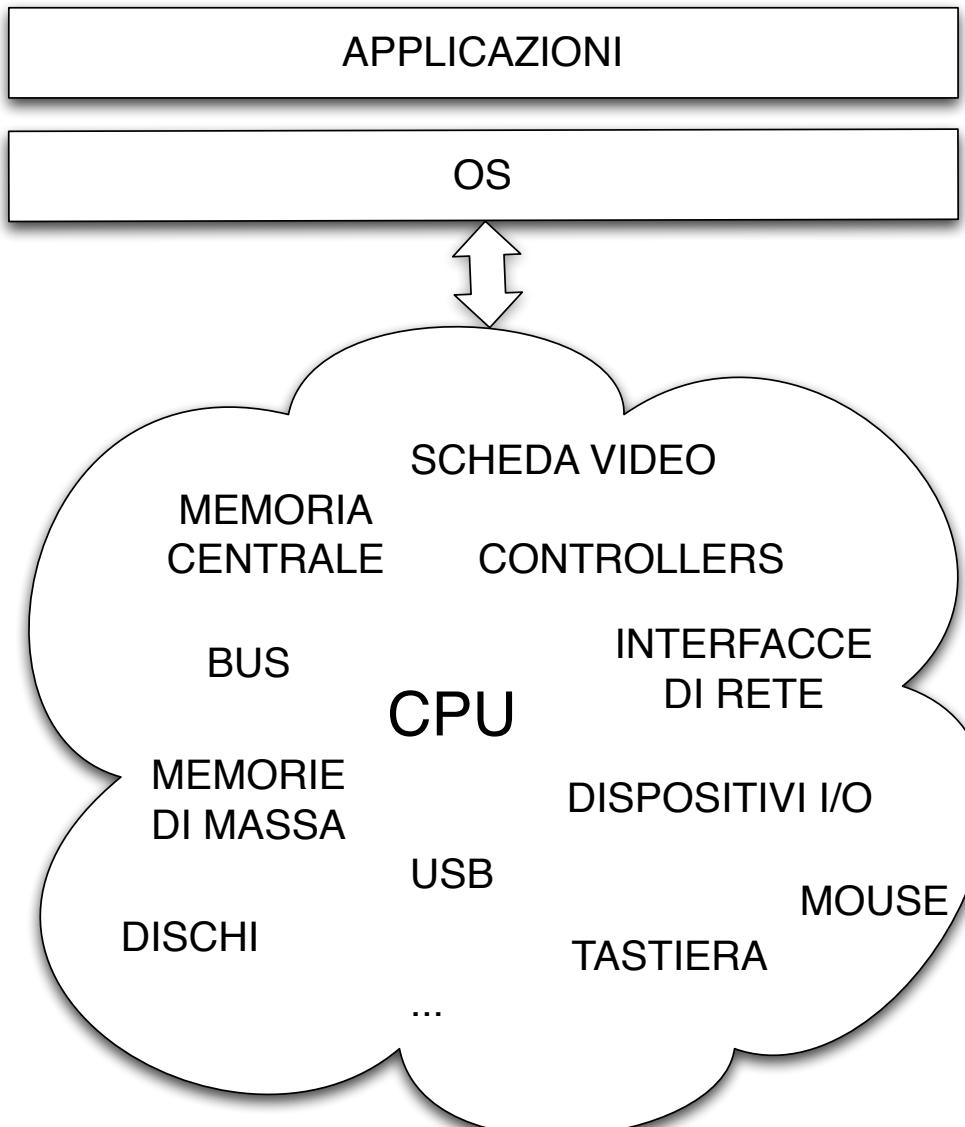
S.O. COME MACCHINA VIRTUALE - BENEFICI

- Agevolare la progettazione e programmazione delle applicazioni
 - il programmatore non deve conoscere necessariamente i dettagli specifici della macchina fisica per interagire con le risorse e può concentrarsi sulle funzionalità del programma
- Aumentare il livello di **portabilità** dei programmi
 - il medesimo programma può essere messo in esecuzione su macchine virtuali con la medesima interfaccia che girano su macchine fisiche diverse
 - semplificazione del *porting* di un programma da un sistema operativo ad un altro

S.O. COME GESTORE DI RISORSE

- Gestione risorse
 - condivise e utilizzate da *più programmi*
 - anche concorrentemente
 - eventualmente appartamenti a *più utenti*
 - che utilizzano la medesima macchina
- Funzionalità di
 - **ottimizzazione** degli accessi
 - algoritmi di *scheduling*
 - **protezione e sicurezza**
 - evitare che un programma in esecuzione possa provocare malfunzionamenti al sistema e ad altri programmi in esecuzione
 - garantire protezione e sicurezza dei dati degli utenti
- Vista *bottom-up*
 - dalle risorse (bottom) ai programmi applicativi

S.O. COME GESTORE DI RISORSE



VIRTUALIZZAZIONE DELLE RISORSE

- **Memoria virtuale**
 - fare in modo che un programma in esecuzione veda la memoria a disposizione come uno spazio **lineare** virtualmente illimitato
 - utilizzo trasparente (per i programmi in esecuzione) della memoria di massa come estensione di quella principale
- **File system virtuale**
 - accedere e modificare file nel file system in modo uniforme a prescindere dal tipo specifico di file system e dalla effettiva locazione dei file stessi
 - locali o remoti

ESECUZIONE DI PROGRAMMI

- **Multi-programmazione (multi-programming)**
 - capacità di caricare in memoria centrale più programmi che vengono eseguiti in modo da ottimizzare l'utilizzo della CPU
 - se un programma in esecuzione è impegnato in una operazione di I/O e non ha bisogno della CPU, la CPU viene allocata ad un altro programma in esecuzione
 - permette una prima forma di **multi-tasking**, ovvero di esecuzione concorrente di più programmi
- **Time-sharing (o multi-tasking)**
 - estensione logica della multi-programmazione funzionale all'esecuzione di programmi che richiedono interazione con l'utente
 - capacità di eseguire più programmi concorrentemente (a prescindere da operazioni di I/O), con condivisione della CPU fra i programmi in esecuzione secondo determinate strategie di schedulazione
 - piena realizzazione del concetto di multi-tasking

MODALITA' DI FUNZIONAMENTO

- Funzionamento **interrupt-driven**
 - entra in esecuzione in seguito ad eventi che sono associati all'occorrenza di **interruzioni hardware** o di **trap software**
 - **interruzioni hardware**
 - segnali inviati da dispositivi (es: tastiera, timer, dischi,...)
 - **asincrone**
 - **trap / interruzioni sw:**
 - richieste da parte dei programmi di eseguire servizi (chiamate di sistema)
 - oppure **eccezioni** generate da errori nei programmi
 - **sincrone**
- Sfruttamento di due modalità operative distinte della CPU
 - **user-mode**
 - esecuzione dei programmi utente
 - **kernel-mode** (super-visor mode, privileged mode, system mode)
 - esecuzione delle parti di programma del sistema operativo

ATTIVITA' PRINCIPALI DI UN S.O.

- **Gestione dei processi**
- **Gestione della memoria**
- **Gestione della persistenza delle informazioni**
(storage)
 - gestione del File System
 - gestione dei dischi
 - caching
- **Gestione dell'I/O**
- **Gestione della rete**
- **Gestione della protezione e sicurezza**
- **Gestione utenti**

SISTEMI OPERATIVI EMBEDDED E REAL-TIME

SISTEMI OPERATIVI EMBEDDED E REAL-TIME

- Sistemi Operativi *Embedded* e *Real-Time* (**RTOS**)
 - sistemi operativi progettati per essere usati nei sistemi embedded
- Caratteristiche in generale
 - compattezza
 - gestione estremamente efficiente delle risorse
 - affidabilità
- Rispetto ai sistemi operativi desktop spesso sono progettati per mandare in esecuzione una *sola applicazione* alla volta
 - che tuttavia è tipicamente multi-task/multi-thread

ESEMPI

- Open-source
 - FreeRTOS, eCos
 - RT Linux, Linux Embedded
- Commerciali
 - VxWorks, LynxOS, Micrium µC/OS-III
 - QNX Neutrino
 - Windows CE
- Una lista
 - https://en.wikipedia.org/wiki/Comparison_of_real-time_operating_systems

ESEMPIO: FAMIGLIA μC/OS

(1) The port developer is responsible for providing the μC/OS-III CPU Specific portion. A μC/OS-III port consists of writing or changing the contents of four kernel-specific files: os_cpu.h, os_cpu_a.asm, os_cpu_a.inc and os_cpu_c.c.

(2) A port also involves writing or changing the contents of two CPU specific files: cpu.h and cpu_a.asm. cpu_core.c is generally generic and should not require modifications.

(3) A Board Support Package (BSP) is generally necessary to interface μC/OS-III to a timer (which is used for the clock tick) and an interrupt controller.

(4) Some semiconductor manufacturers provide source and header files to access on-chip peripherals. These are contained in CPU/MCU specific files. You generally don't need to modify any of these and thus, you can use them as-is.

Configuration Files

```
cpu_cfg.h  
lib_cfg.h  
os_cfg.h  
os_cfg_app.h
```

Application Code

```
app.c  
app.h  
app_cfg.h
```

μC/OS-III

CPU Independent

```
os_cfg_app.c  
os_type.h  
os_core.c  
os_dbg.c  
os_flag.c  
os_int.c  
os_mem.c  
os_msg.c  
os_mutex.c  
os_pend_multi.c  
os_prio.c  
os_q.c  
os_sem.c  
os_stat.c  
os_task.c  
os_tick.c  
os_time.c  
os_tmr.c  
os_var.c  
os.h
```

μC/LIB

Libraries

```
lib_ascii.c  
lib_ascii.h  
lib_def.h  
lib_math.c  
lib_math.h  
lib_mem_a.asm  
lib_mem.c  
lib_mem.h  
lib_str.c  
lib_str.h
```

μC/OS-III

CPU Specific

```
os_cpu.h  
os_cpu_a.asm  
os_cpu_a.inc  
os_cpu_c.c
```

(1)

μC/CPU

CPU Specific

```
cpu.h  
cpu_bsp.c  
cpu_def.h  
cpu_a.asm  
cpu_c.c  
cpu_core.c  
cpu_core.h
```

(2)

BSP

Board Support Package

```
bsp.c  
bsp.h  
bsp_int.c  
bsp_int.h
```

(3)

CPU

```
*.c  
.h
```

Software/Firmware

Hardware

CPU

Timer

Interrupt
Controller

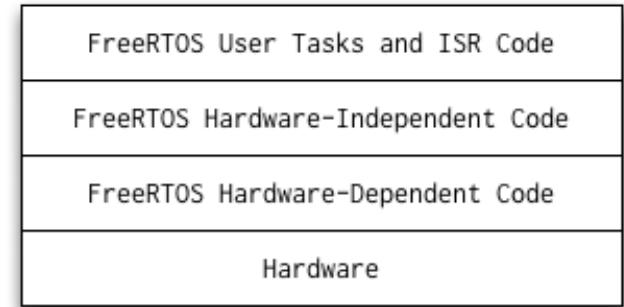
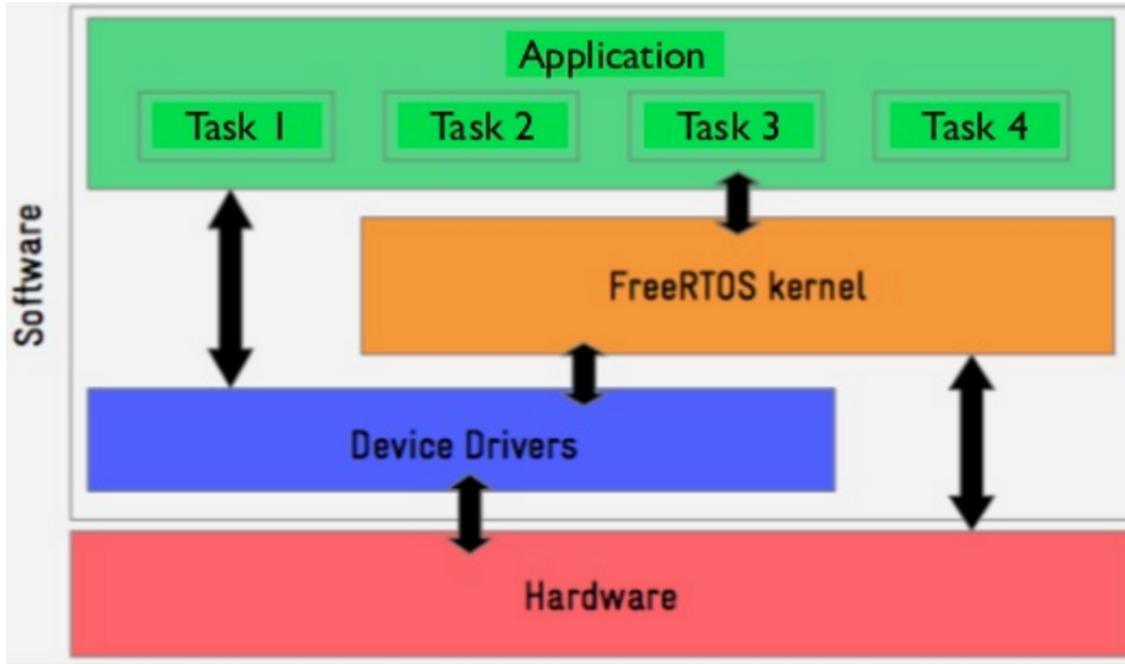
<https://doc.micrium.com/>

µC/OS-II and µC/OS-III Feature Comparison Chart

ESEMPIO: FAMIGLIA µC/OS

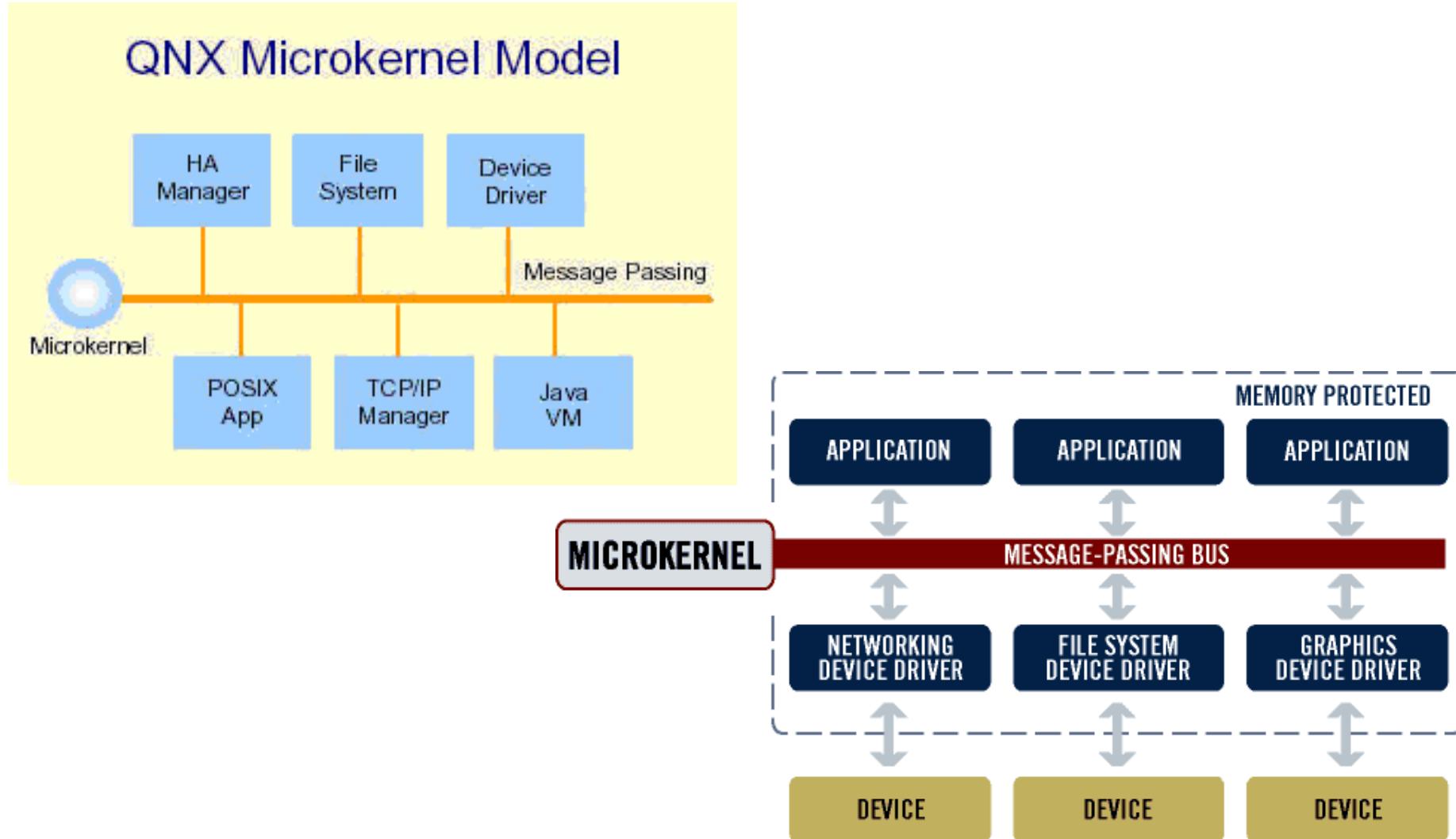
Feature	µC/OS-II	µC/OS-III
Preemptive Multitasking	Yes	Yes
Maximum number of tasks	255	Unlimited
Number of tasks at each priority level	1	Unlimited
Round Robin Scheduling	No	Yes
Semaphores	Yes	Yes
Mutual Exclusion Semaphores	Yes	Yes (Nestable)
Event Flags	Yes	Yes
Message Mailboxes	Yes	No (not needed)
Message Queues	Yes	Yes
Fixed Sized Memory Management	Yes	Yes
Signal a task without requiring a semaphore	No	Yes
Send messages to a task without requiring a message queue	No	Yes
Software Timers	Yes	Yes
Task suspend/resume	Yes	Yes (Nestable)
Deadlock prevention	Yes	Yes
Scalable	Yes	Yes
Code Footprint	6K to 26K	6K to 20K
Data Footprint	1K+	1K+
ROMable	Yes	Yes
Run-time configurable	No	Yes
Compile-time configurable	Yes	Yes
ASCII names for each kernel object	Yes	Yes
Interrupt Latency (Cortex-M3)	120~	< 100~
Pend on multiple objects	Yes	Yes
Task registers	Yes	Yes
Built-in performance measurements	Limited	Extensive
User definable hook functions	Yes	Yes
Time stamps on posts	No	Yes
Built-in Kernel Awareness support	Yes	Yes
Optimizable Scheduler in assembly language	No	Yes
Tick handling at task level	No	Yes
Source code available	Yes	Yes

ESEMPIO: FreeRTOS

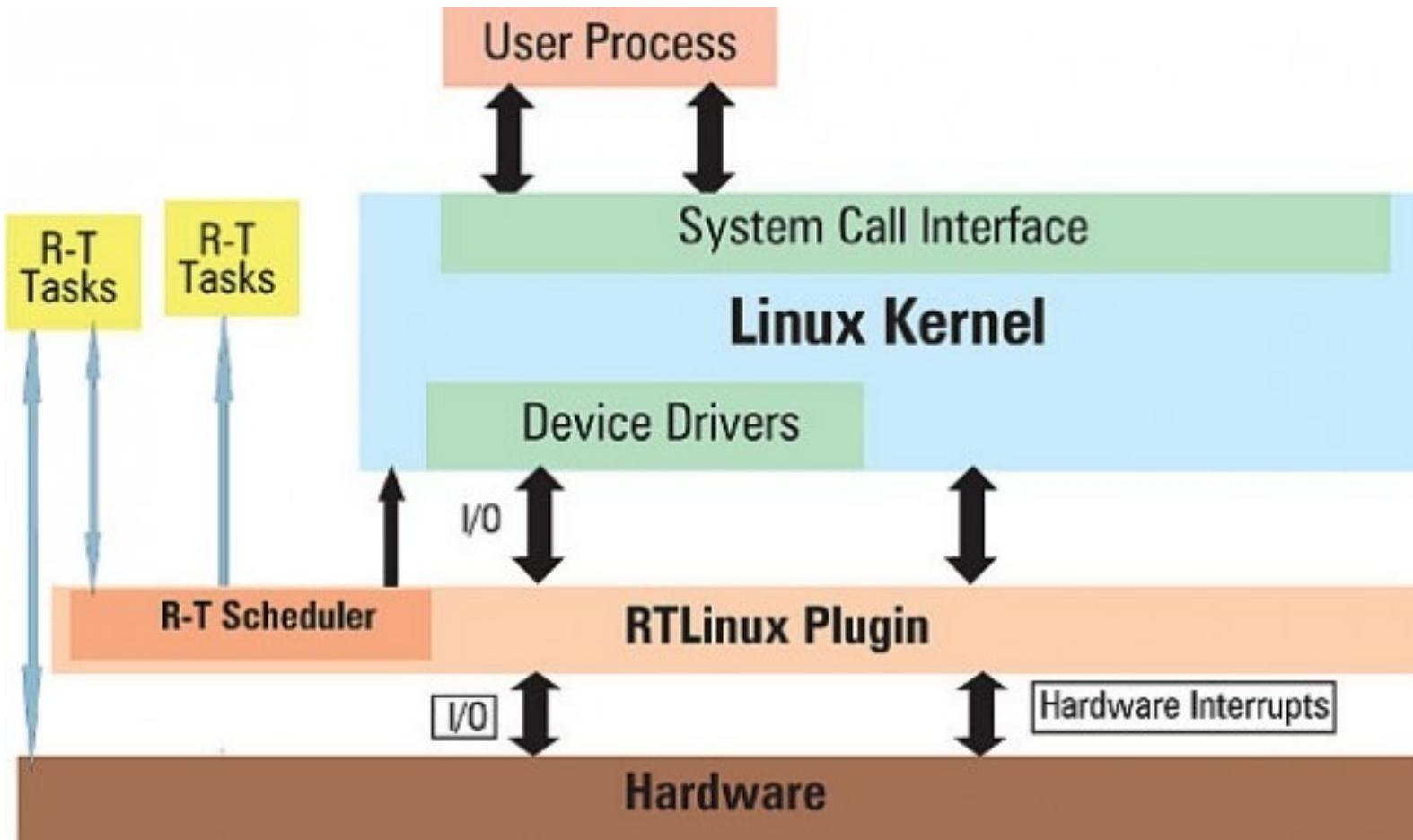


<https://www.freertos.org/>

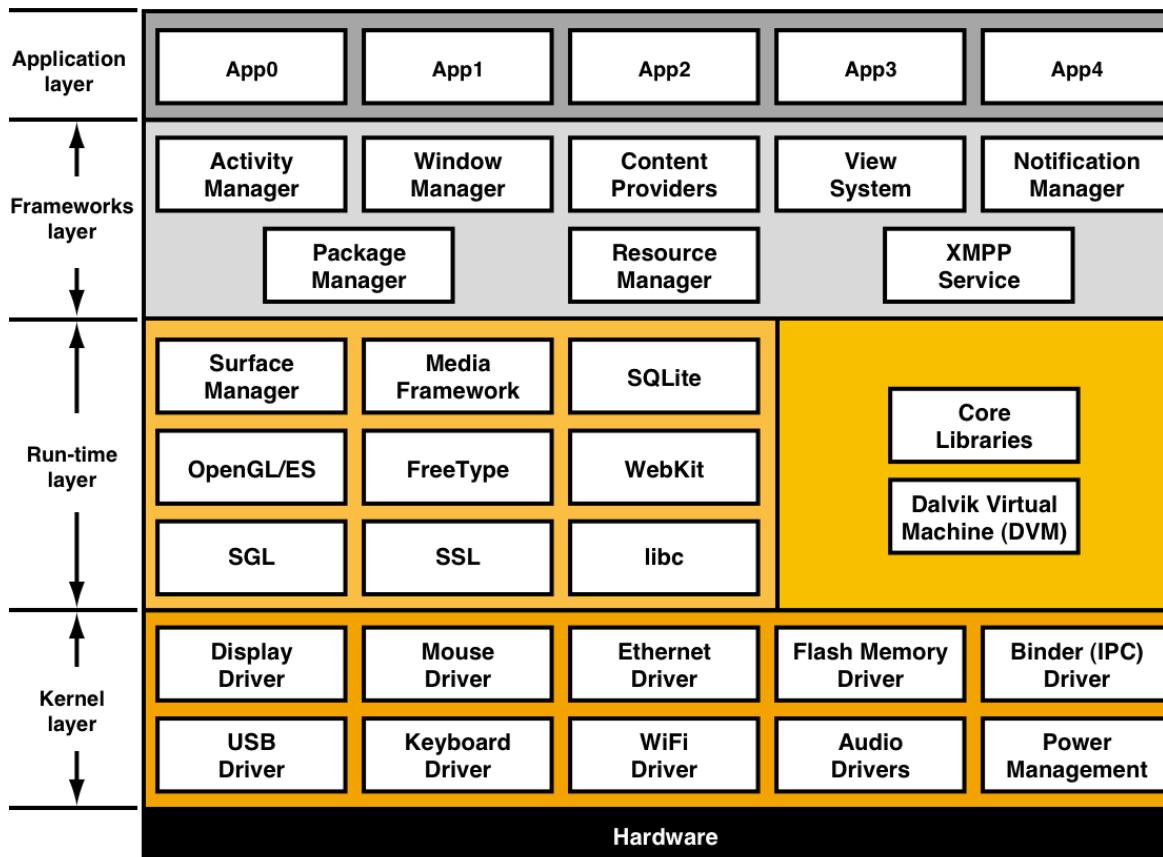
ESEMPIO: QNX REAL-TIME OS



ARCHITETTURA RT-LINUX

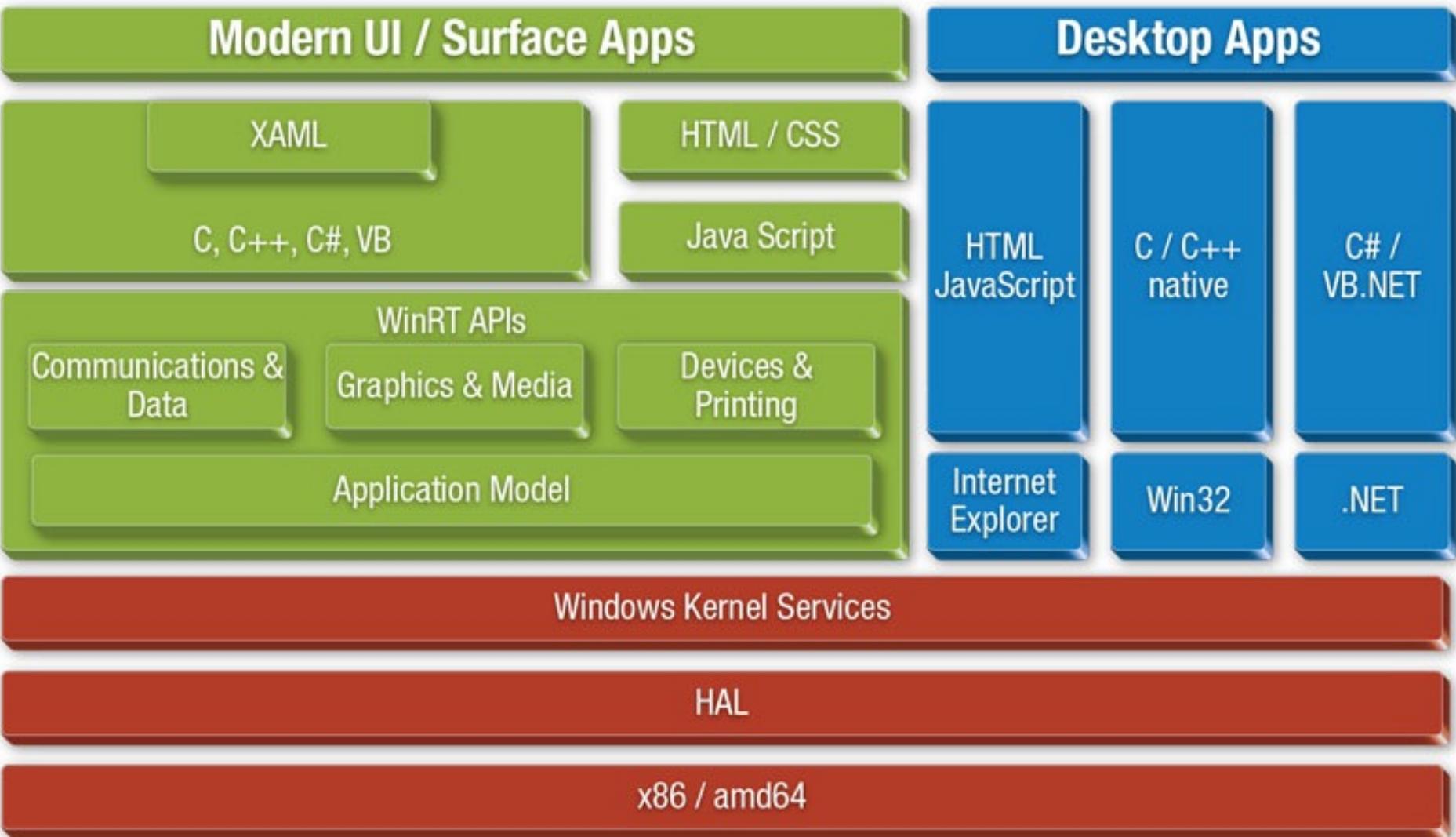


EMBEDDED ANDROID



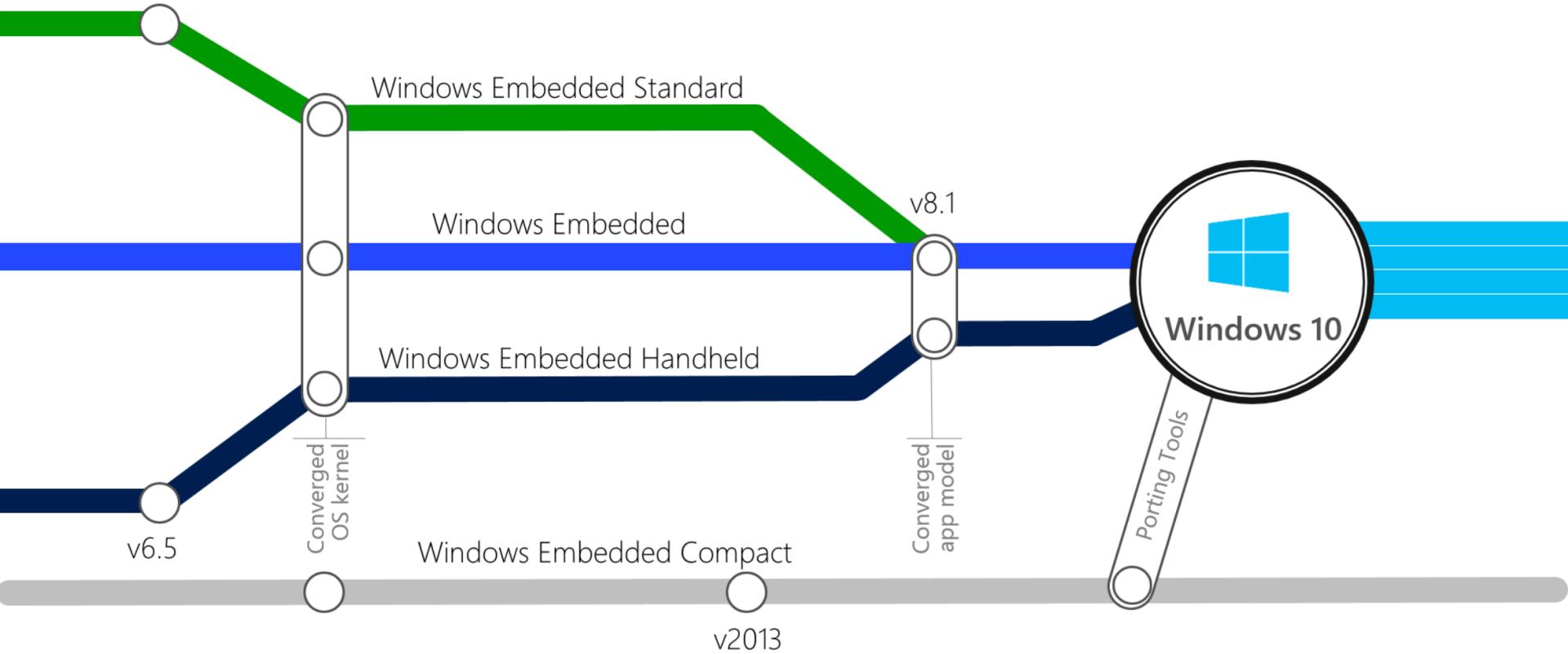
C, C++, native code	Java
■ = Linux Kernel	■ = Android frameworks
■ = Libraries	■ = Applications
■ = Android runtime	

WINDOWS 8 EMBEDDED



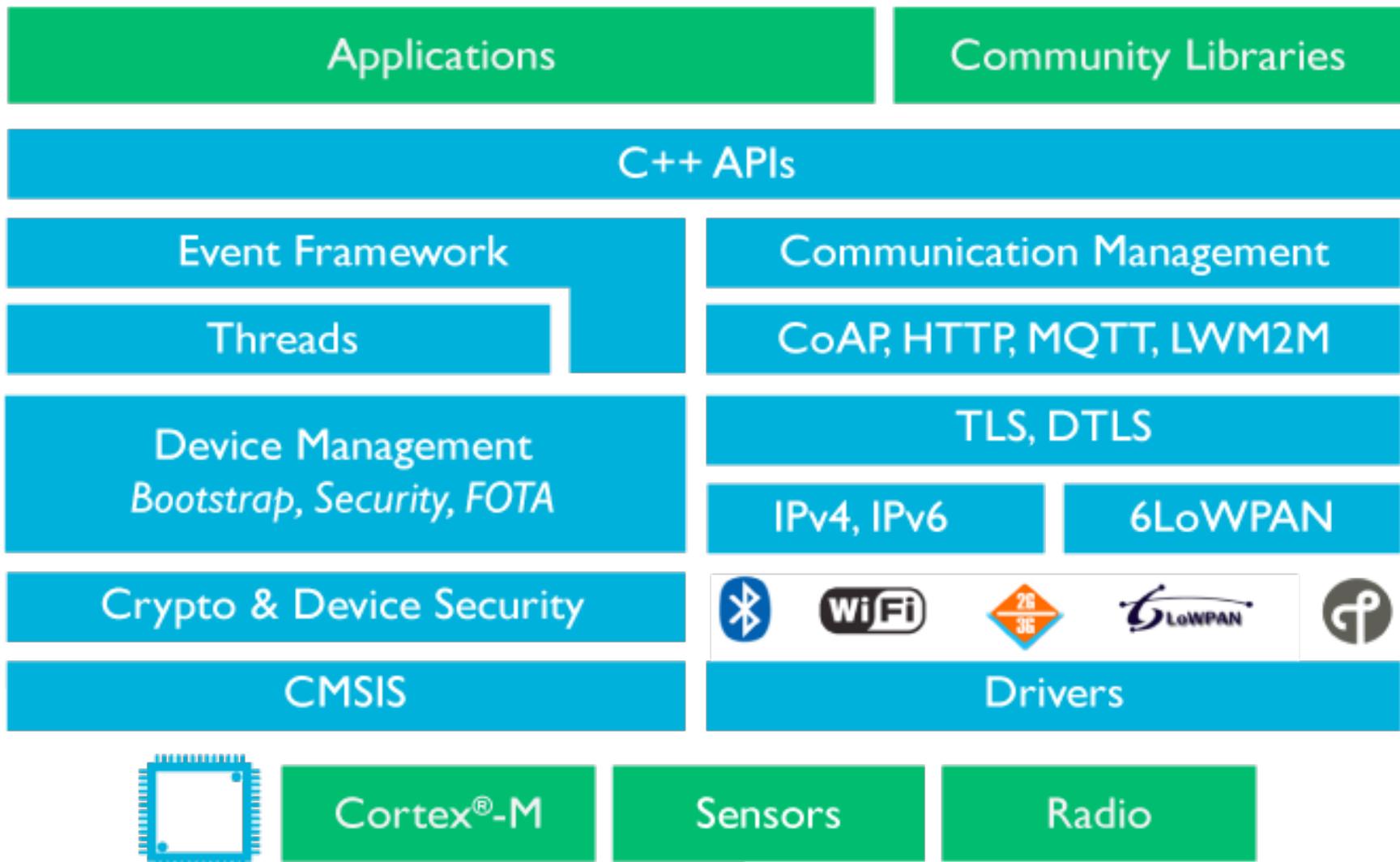
WINDOWS IOT CORE

IoT Platform Convergence



<https://docs.microsoft.com/en-us/windows/iot-core/windows-iot>

ARM mbed IoT OS ARCHITECTURE



SISTEMI EMBEDDED REAL-TIME

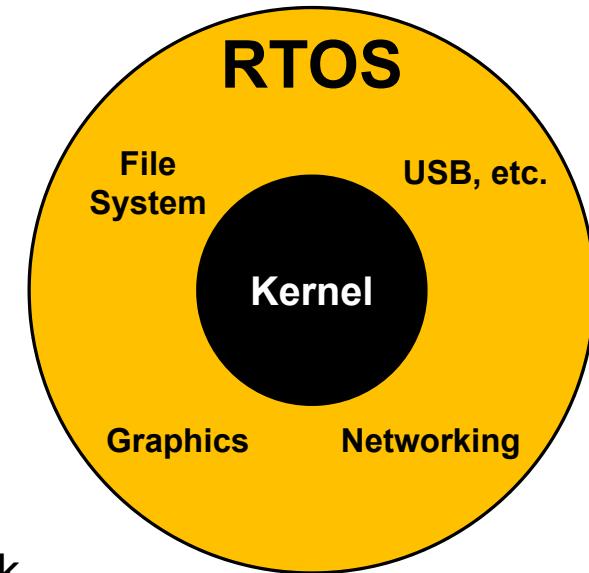
- Real-time
 - devono essere in grado di reagire/rispondere ad eventi e input *entro limiti di tempo prestabiliti*
 - devono eseguire task/computazioni *entro determinati vincoli temporali*
 - nozione di **deadline**
 - termine entro il quale deve essere completato un task
- Sottoclassi
 - **hard** real-time
 - le deadline devono essere rispettate sempre
 - esempi: sistemi safety-critical
 - **soft** real-time
 - le deadline devono essere rispettate in condizioni normali, sono ammessi casi in cui non vengono rispettate
 - esempi: sistemi embedded non safety-critical, consumer electronics

SISTEMI REAL-TIME E DETERMINISMO

- Il **determinismo** è un aspetto importante di molti sistemi real-time
 - in particolare di quelli hard real-time
- Proprietà per cui deve essere ***predicibile***:
 - il tempo impiegato per svolgere un certo task
 - il tempo massimo richiesto per eseguire una certa azione o acquisire un certo valore in input o da un sensore o a rispondere ad una certa interruzione
 - numero di cicli richiesti per eseguire una certa operazione deve essere sempre lo stesso
- In questi sistemi l'esecuzione può essere interrotta (interruzioni), tuttavia l'overhead relativo (latenza, tempo di elaborazione dell'interruzione) deve essere limitato e noto
- Non tutti i RTOS sono in grado di fornire garanzie in merito

ORGANIZZAZIONE DI UN RTOS

- Tipica struttura (simile ad un OS normale)
 - kernel
 - file system
 - networking
 - USB
 - Graphics
- Attività/compiti del kernel
 - gestione e scheduling task
 - sincronizzazione e comunicazione inter-task
 - semafori
 - code / scambio messaggi
 - gestione timer
 - gestione dispositivi I/O
 - efficienza, protezione, condivisione
 - gestione memoria
 - gestione interruzione & eventi



Modalità funzionamento CPU
- *kernel mode*
- *user mode*

PERCHE' USARE UN RTOS - BENEFICI

- Migliorare la responsiveness e diminuire overhead
- Semplificare la condivisione delle risorse
- Semplificare lo sviluppo, debugging e maintenance
 - sistemi software embedded complessi
- Aumentare la portabilità
- Abilitare l'uso di servizi e middleware stratificati
- Rendere più veloce il time-to-market delle applicazione

RESPONSIVENESS E OVERHEAD

- I programmi basati su sistemi senza RTOS sono basati tipicamente su un loop di controllo in cui si adotta il polling come strategia per verificare la necessità di eseguire determinate funzioni/task
 - il numero di application function/task determina il tempo complessivo di questo polling
 - il tempo necessario per svolgere un servizio dipende da questo polling time
 - looping, checking, polling, state machine tracking: attività che consumano cicli del processore e aggiungono overhead
- Un RTOS in generale permette di
 - evitare polling/looping sfruttando la possibilità di eseguire "context switch" al processore, da un task ad un altro
 - in modo trasparente all'applicazione
 - realizzare in modo trasparente alle applicazioni multi-tasking
 - si sfrutta il processore anche quando l'applicazione è in waiting
 - multi-threading
 - dedicare maggior tempo del processore all'applicazione
 - meno tempo speso alla gestione delle applicazioni

GESTIONE RISORSE

- Gestione risorse condivise da più funzioni/task
 - memoria, porte I/O, sezioni critiche (codice)
- RTOS => meccanismi centralizzati per gestire/arbitrare le richieste da task diversi
 - allocazione/deallocazione memoria a runtime
 - semafori/mutex per controllare accesso a risorse HW o a sezione critiche
 - meccanismi per evitare problemi dovute a interferenze fra priorità e sezioni critiche
 - esempio: il problema dell'inversione delle priorità (descritto in seguito)

SEMPLIFICARE SVILUPPO E DEBUGGING

- Barriera di astrazione rispetto alla gestione degli aspetti HW
 - lo sviluppatore di applicazioni non deve gestire aspetti di basso livello come interruzioni, timer, etc.
- Interfaccia del RTOS
 - chiamate di sistema/servizi
- Miglior supporto allo sviluppo a livello di team
- Supporti specifici al debugging
- Sviluppo modulare delle applicazioni
 - migliore estendibilità e modificabilità

SERVIZI A LIVELLI E MIDDLEWARE

- Un RTOS abilita l'utilizzo di framework/piattaforme/middleware che forniscono servizi, spesso a livelli diversi
- Esempi
 - File system
 - TCP/IP network stack
 - USB stack
 - Graphics
 - prodotti di terze parti

PORATIBILITA' E MAINTENANCE

- Un'applicazione usa le RTOS API, non accede direttamente all'HW specifico
 - questo permette di poter cambiare HW trasparentemente
 - l'applicazione può essere mandata in esecuzione in modo trasparente su qualsiasi HW supportato dal RTOS

QUANDO NON CONVIENE USARE UN RTOS

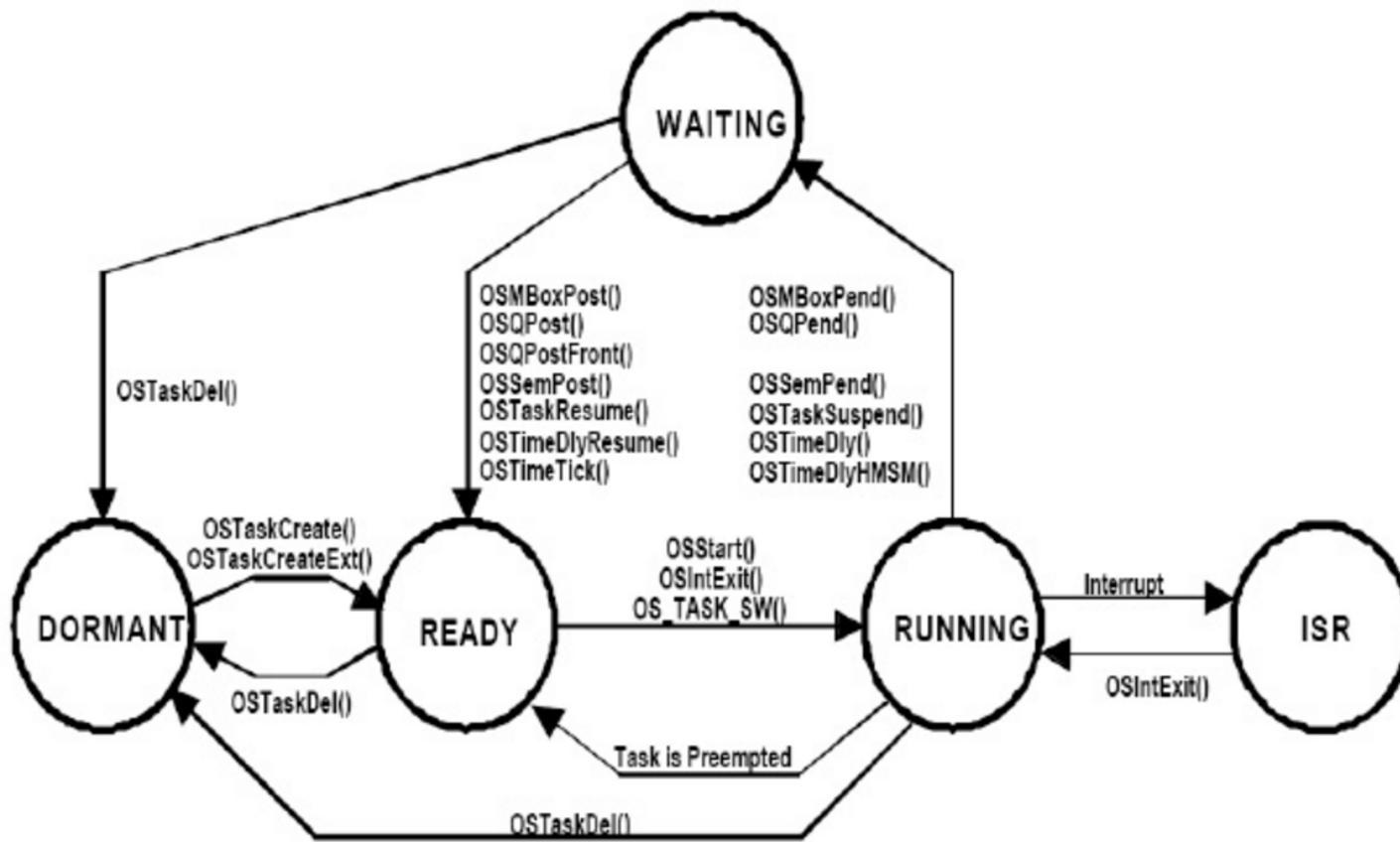
- Applicazioni simple looping o polling
- Applicazioni single-purpose / single-task
 - tipicamente applicazioni piccole < 32 KiB

PROCESSI, TASK E THREAD

- Processo
 - programma in esecuzione
 - propria memoria
 - può avere più thread
- Thread
 - flusso di controllo indipendente
 - condivisione memoria con altri thread del medesimo processo
- Multi-threading: esecuzione concorrente di più thread
 - spesso chiamati **task** nei RTOS
 - elemento di modularizzazione di programmi
 - nozione di priorità
- Scheduling ad opera del kernel
 - per realizzare multi-tasking, multi-threading
 - varie strategie e algoritmi
 - pre-emptive vs. cooperativo
 - round-robin + priorità

STATI DI UN TASK

- Esempio: µC/OS-III



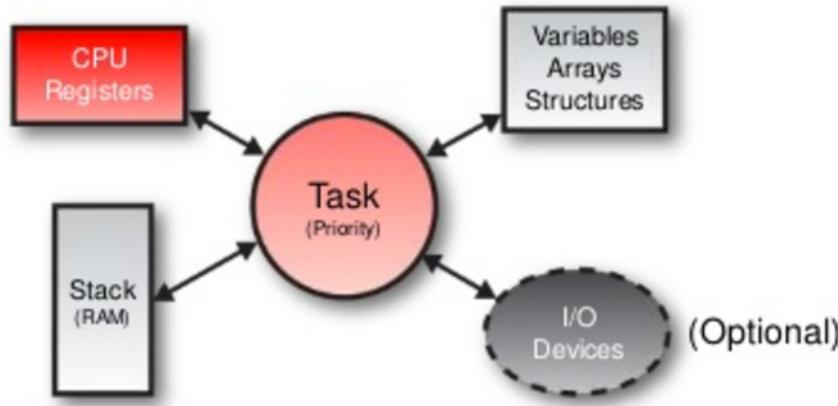
CONTEXT SWITCH

- Thread context
 - informazioni critiche all'esecuzione del thread
 - contenuto corrente registri processore, program counter, stack pointer
 - salvato quando un thread è preempted
 - ripristinato quando un thread è resumed
- Context switch
 - interruzione dell'esecuzione del thread corrente
 - passaggio del processore al kernel e poi eventualmente ad un altro thread
 - costo

PROGRAMMAZIONE TASK

- Esempio: µC/OS-III

```
void MyTask (void *p_arg)
{
    Do something with 'argument' p_arg;
    Task initialization;
    for (;;) {
        Wait for event;      /* Time to expire ... */
        /* Signal/Msg from ISR ... */
        /* Signal/Msg from task ... */
        /* Processing (Your Code) */
    }
}
```



TIPI DI SCHEDULING

- *Big loop* scheduling (ovvero: loop di controllo principale, singolo)
 - ogni task è polled per verificare se richiede di essere eseguito
 - polling procede sequenzialmente o in ordine di priorità
 - inefficiente, mancanza di responsiveness
- Round-robin scheduling
 - processore dato a turno ai task ready
 - i thread possono essere eseguiti fino al completamento o al blocco
 - oppure può essere imposto un time-slice ad ogni thread
 - in questo caso viene sfruttata la preemption
- **Priority-based** scheduling
 - il processore esegue sempre il thread con priorità maggiore
 - per thread con la stessa priorità => round-robin
 - preemption
 - se c'è un ready thread più prioritario di quello in esecuzione

SCEDULING BASATO SU PRIORITA'

- Tutti i sistemi RTOS supportano scheduling con priorità
 - eventualmente integrata con round-robin
- Vantaggi
 - garantisce massima responsiveness
- Problemi/aspetti critici
 - complessità e costo preemption
 - possibilità di starvation
 - problema dell'inversione della priorità

MECCANISMI DI COMUNICAZIONE E SINCRONIZZAZIONE

- Semafori
- Comunicazione a scambio di messaggi
- Architetture ad eventi

SEMAFORI

- Due tipi di uso
 - regolare la competizione
 - mutua esclusione
 - **mutex**
 - **spin-locks**
 - accesso regolato
 - **semafori risorsa/counting**
 - sincronizzazione e coordinazione
 - **semafori evento**

COMUNICAZIONE SCAMBIO MESSAGGI

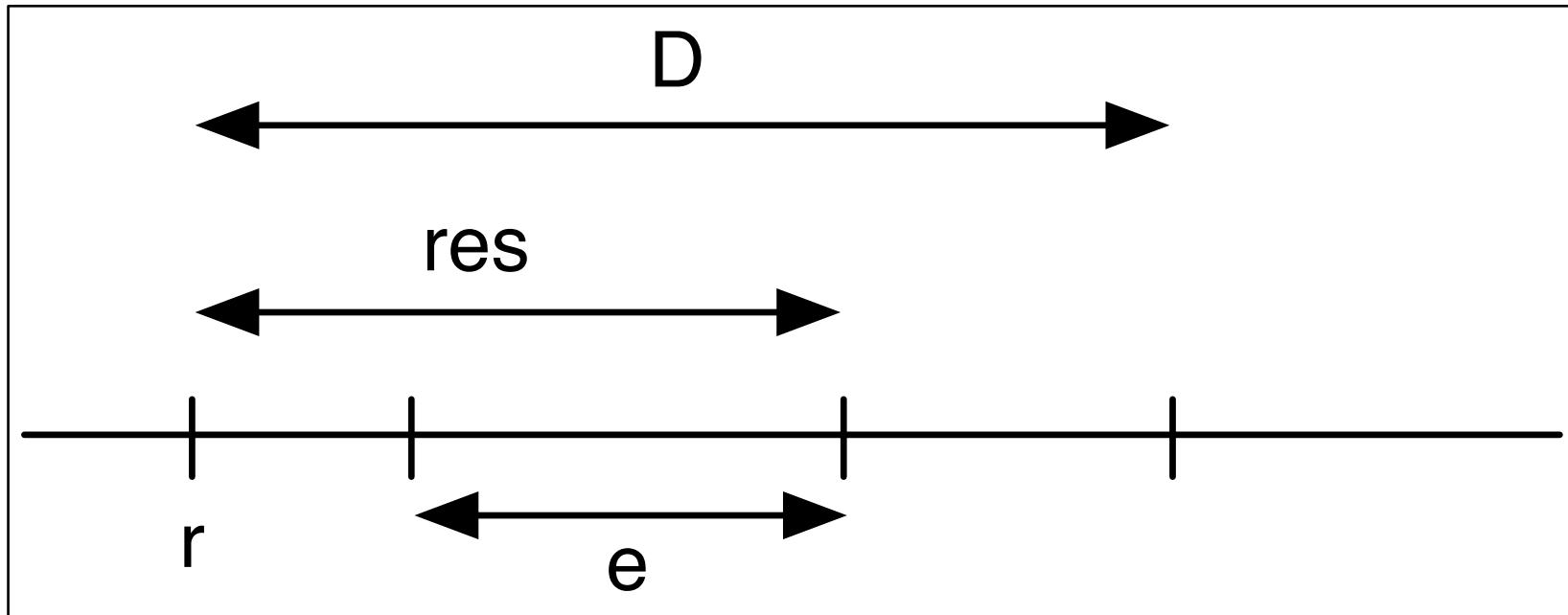
- Code di messaggi
 - strutture dati gestite dal kernel che permettono la comunicazione di messaggi fra task
 - tipicamente strategia FIFO
 - condivisibili da più task
- Primitive
 - send e receive
- Architettura produttore-consumatore
 - dove le code di messaggi sono bounded buffer

SCHEDULING DI TASK NEGLI RTOS: UN APPROFONDIMENTO

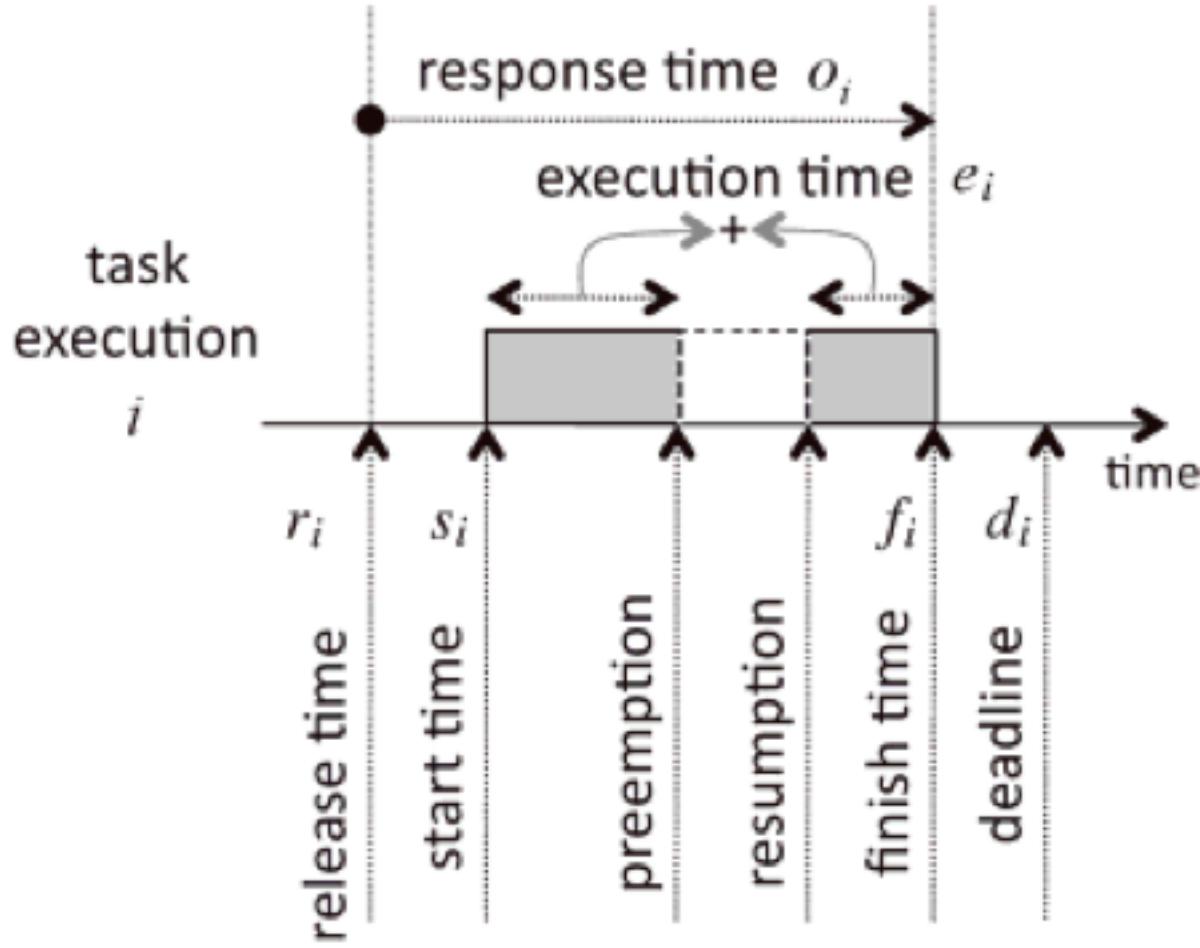
PARAMETRI TEMPORALI TASK

- Le proprietà temporali dei task sono definite da 4 parametri fondamentali
 - *tempo di rilascio (release time)* r
 - istante in cui il task entra nella coda di ready, essendo pronto per essere eseguito
 - *tempo di esecuzione (execution time)* $e = WCET$
 - durata massima esecuzione del task
 - *tempo di risposta del task (response time)*
 - intervallo di tempo che intercorre da quando il task è rilasciato a quando a completato la sua esecuzione
 - scadenza (**deadline**) D
 - massimo intervallo di tempo permesso per l'esecuzione di un task

PARAMETRI TEMPORALI



PARAMETRI TEMPORALI



TIPI DI TASK

- Tipi di task
 - **periodici**
 - c'è un intervallo di tempo prefissato (periodo p) che intercorrere fra release time dei vari task
 - **aperiodici**
 - task rilasciati a tempi arbitrari
 - **sporadici**
 - task rilasciati a tempi arbitrari, con hard deadline
- Il progettista di un sistema real-time deve decidere cosa fare se una deadline non è soddisfatta
 - il task deve continuare la propria esecuzione (quindi ritardando gli altri task)? Oppure deve essere forzata la terminazione del task (liberando la CPU)?

TIPI DI SCHEDULER

- Uno scheduler deve prendere 3 tipi di decisioni:
 - *assegnamento*
 - quale processore usare per eseguire un certo task
 - *ordine*
 - in quale ordine ogni processore deve eseguire i suoi task
 - *timing*
 - quando un determinato task deve o può essere eseguito
- Ognuno di questi tre tipi di decisione può essere preso o a design time o a run time, durante l'esecuzione del programma.
- A seconda di quando vengono prese le decisioni si distinguono tipi di scheduler diversi:
 - *offline* schedulers:
 - **fully-static scheduler**
 - **static-order scheduler**
 - *online* schedulers:
 - **static assignment scheduler**
 - **fully-dynamic scheduler**

OFFLINE SCHEDULER

- **Fully-static scheduler**
 - tutte e 3 le decisioni sono prese a design time
 - usato quando si conoscono perfettamente tutte le tempistiche dell'esecuzione dei task, tipicamente indipendenti
 - non si usano in questo caso semafori o altri meccanismi per la sincronizzazione: si sincronizzano i task a partire dai loro tempi di esecuzione
 - difficili da usare con i moderni microprocessori, poiché il tempo necessario per eseguire un task è difficile da prevedere a priori e poiché i task hanno spesso dipendenze di tipo data-dependence
- **Static-order scheduler**
 - task assignment e ordering a design time, tuttavia la decisione di quando eseguire i task a livello temporale viene fatta a runtime
 - tale decisione può dipendere, ad esempio, dalla presenza di blocchi a causa di lock o semafori

ONLINE SCHEDULER

- **Static assignment scheduler**
 - assegnamento a design time e tutto il resto a runtime
 - presenza di un run-time scheduler
- **Fully-dynamic scheduler**
 - anche l'assegnamento viene deciso a runtime, quando un processore si rende disponibile

SISTEMI REAL-TIME SINCRONI E ASINCRONI

- In generale è possibile a seconda del tipo di sistema real-time si adottano strategie di scheduling diverse
- Esistono due macro categorie di sistemi real-time
 - sistemi real-time **sincroni**
 - sistemi real-time **asincroni**

SISTEMI REAL-TIME SINCRONI

- In questo caso, un clock hardware è utilizzato per suddividere il tempo del processore in intervalli chiamati **frame**
- Il programma deve essere suddiviso in task in modo che ogni task possa essere completamente eseguito nel caso peggiore in un singolo frame
 - mediante una *tabella di scheduling* si assegnano i task ai frame, in modo che tutti i task all'interno di un frame siano completamente eseguiti prima della fine del frame
 - quando il clock segnala l'inizio del nuovo frame, lo scheduler richiama l'esecuzione dei task specificata in tabella
 - se un task ha una durata che eccede un frame, lo si deve esplicitamente suddividere in sotto-task più piccoli che possano essere individualmente schedulati in più frame successivi
- La dimensione del frame diventa un parametro scelto a livello di progettazione
- Questo approccio è utilizzato primariamente in sistemi di controllo hard real-time

SCHEDULER PER SISTEMI REAL-TIME SINCRONI

```
taskAddressType array[0..NFrames-1] tasks = [ taskAddr0, taskAddr1, ... ]
integer currentFrame <- 0

loop
    await beginning of the frame
    invoke tasks[currentFrame]
    currentFrame = (currentFrame + 1) % NFrames
```

- In questo caso la complessità del lavoro è tutta nella decomposizione in tasks che entrano nei frame e nella costruzione della tabella di scheduling
- Svantaggi
 - sistemi molto fragili dal punto di vista della progettazione
 - una modifica/estensione del sistema => può portare a task troppo lunghi => deve essere ricomputata la tabella
 - il tempo del processore può essere sprecato
 - il task che costituisce il caso-peccato condiziona l'intero sistema

SISTEMI REAL-TIME ASINCRONI

- In questo caso non si richiede che i task completino l'esecuzione entro frame temporali prefissati
 - ogni task prosegue la propria esecuzione e lo scheduler è invocato per selezionare il prossimo task, pronto per l'esecuzione
 - Bozza dello scheduler

```
queue of taskAddressType readyQueue = ...
taskAddressType currentTask

loop
    await readyQueue not empty
    currentTask = readyQueue.removeHead
    invoke currentTask
```

- Pros
 - molto efficiente
 - nessuno spreco del tempo processore
- Cons
 - in questo caso il soddisfacimento delle deadline è un problema più complesso

PRIORITA' E SCHEDULING PREEMPTIVE NEI SISTEMI ASINCRONI

- Nei sistemi asincroni, le priorità e lo scheduling pre-emptive sono usati per fare in modo che i task più importanti siano eseguiti quando necessario, a discapito di task meno importanti
 - priorità descritta da un numero naturale assegnato al task
 - un task non viene eseguito nel caso in cui in un sistema ci sia un task nella coda di ready con priorità maggiore
- L'implementazione si basa su *scheduling events* che causano l'invocazione dello scheduler
 - generati quando un task viene accodato nella coda di ready, oppure generato dall'interruzione del timer ad ogni colpo di clock

SCHEDULER PREEMPTIVE

```
queue of taskAddressType readyQueue <- ...
taskAddressType currentTask

loop
    await a scheduling event
    if (currentTask.priority < highest priority of a task on readyQueue)
        save context of currentTask and place on readyQueue
        currentTask <- take task with highest priority from readyQueue
    else if currentTask's timeslice is past and
        it exists some other tasks in readyQueue with the same priority
        save context of currentTask and place on readyQueue
        currentTask <- take task with of the same priority from readyQueue
    resume currentTask
```

- Tecnica **time-slicing** e scheduling **round-robin**
 - insieme di ready queue, una per ogni priorità e ognuna con strategia di scheduling round robin
 - garantire un response time adeguato e allo stesso tempo un uso efficiente del processore mediante un opportuno assegnamento delle priorità

WATCH-DOG TASK

- Task usato nei sistemi asincroni per controllare l'esecuzione di altri task
 - task periodico, con priorità massima
 - ha il compito di verificare che determinati task siano sempre eseguiti e quindi di segnalare problemi

SISTEMI INTERRUPT-DRIVEN

- L'integrazione di sistemi (operativi) basati su interruzioni con questo tipo di scheduling può essere fatto modellando gli interrupt handler come *task la cui priorità è superiore a qualsiasi altro (normale) software task*
 - questo garantisce che gli interrupt handler possono essere eseguiti come sezioni critiche, non interrompibili da altri task
 - se ci sono interrupt molteplici, allora l'hardware può essere programmato in modo da mascherare le interruzioni

COMUNICAZIONE INTERRUPT-HANDLER E SOFTWARE TASKS

- La comunicazione fra interrupt handler e task avviene tipicamente adottando **architetture produttore/consumatore**
 - ex. sensore di temperatura che segnala un nuovo dato da leggere, l'handler legge il dato e lo memorizza in un buffer globale in modo che il dato sia leggibile da gli altri task
- Problema
 - l'interrupt handler *non può mai bloccarsi* (ad esempio usando semafori) per una questione di performance, per cui devono essere utilizzati algoritmi e tecniche non bloccanti
 - es: bounded buffer: l'handler che funge da produttore non può bloccarsi se il buffer è pieno
 - deve o sovrascrivere il dato più vecchio o gettare via il nuovo dato

INTERRUPT OVERFLOW

- La priorità assoluta data alle interruzioni rispetto ai task software può creare situazioni problematiche
 - specialmente quando gli interrupt sono generati da componenti HW sui quali il designer ha poco controllo
- Esempio: problema che capitò sull'Apollo-11 (1969) nella missione che portò allo sbarco sulla luna
 - il radar del modulo lunare era stato programmato per generare interruzioni durante l'allunaggio
 - non appena il modulo si avvicinò alla superficie lunare, il numero di interruzioni incrementò così rapidamente al punto da utilizzare circa il 15% delle CPU
 - questo portò a far ritardare il completamento di un software task, che non riusciva a completare prima dell'intervento del watch dog
 - di conseguenza il watch-dog reinizializzava il computer continuamente (3 volte ogni 40 secondi), causando la generazione di allarmi
 - fortunatamente gli ingegneri della NASA avevano capito quale fosse la sorgente del problema e sapevano che non avrebbe inficiato il completamento con successo dell'allunaggio

PROBLEMA INVERSIONE PRIORITA'

- L'uso di scheduling preemptive basato su priorità può interferire con le strategie di sincronizzazione adottate dai software task, portando a problemi
- Un problema importante è dato dall'**inversione di priorità** (*priority inversion*), per effetto del quale un task a bassa priorità può ritardare l'esecuzione di un task a priorità più elevata
 - problema capitato in Mars Path Finder, in 1997
- Esempi concreti
 - task con sezioni critiche
 - task che accedono a monitor (e sono nella loro coda di attesa)

ESEMPIO

- Supponiamo di avere un task T con una certa priorità p che entra in sezione critica
- Supponiamo quindi che un task T₁ con priorità p₁ > p sia quindi schedulato per essere eseguito e che tale task necessita anch'esso di entrare in sezione critica
- In questo caso, per non violare la semantica della sezione critica, il task a priorità più alta T₁ deve aspettare che il task a priorità più bassa completi la propria sezione critica
 - principio necessario per garantire la correttezza
 - motivo per cui le sezioni critiche devono essere di durata breve
- Il problema dell'inversione di priorità succede se un terzo task T₂ con priorità p < p₂ < p₁ che non richiede di entrare in sezione critica viene a questo punto schedulato
- Il processore è tolto a T e dato a T₂, dal momento che la priorità di T₂ è superiore a quella di T e che T₂ non deve entrare in sezione critica
- Così facendo, il task con priorità media T₂ può ritardare l'esecuzione del task prioritario T₁ per un tempo arbitrariamente lungo (dal momento che non è in sezione critica non deve essere di durata breve), quindi violando i principi dello scheduling preemptive basato su priorità

SOLUZIONI

- Il problema dell'inversione di priorità deve essere evitato, per fare in modo che l'analisi delle performance di un sistema real-time possa essere condotta sui task a prescindere dal livello di priorità (di altri task..)
- Soluzioni proposte in letteratura
 - tecnica **priority inheritance**
 - il problema può essere risolto aumentando temporaneamente la priorità del task in sezione critica ad essere pari a quella massima fra tutti i task che devono/possono essere schedulati in futuro e che vogliono entrare in sezione critica
 - tecnica **priority ceiling locking**
 - appropriata quando si lavora con monitors
 - l'idea è di assegnare al monitor una priorità di base che sia maggiore o uguale alla più alta priorità posseduta dai task che richiedono di interagire con il monitor chiamando le sue operazioni
 - quando un task chiama una di queste operazioni, eredità la ceiling priority del monitor

ASSEGNAZIONE DELLE PRIORITA': ASSEGNAMENTI FEASIBLE

- Un aspetto fondamentale dello scheduling in un sistema asincrono è assegnare le priorità ai task in modo che *tutti i task rispettino le scadenze designate* (assegnamento **feasible**)
- Esempio
 - 2 task P1 and P2
 - P1: $p_1 = D_1 = 2$, $e_1 = 1$ (p_1 = periodo, D_1 = deadline)
= deve essere eseguito ogni 2 unità di tempo e ne richiede 1 per l'esecuzione
 - P2: $p_2 = D_2 = 5$, $e_2 = 2$
= deve essere eseguito ogni 5 unità di tempo e ne richiede 2
 - 2 possibilità per assegnare le priorità
 - P1 ha priorità maggiore di P2 o vice-versa
 - tuttavia
 - $P_1 > P_2$ è un assegnamento feasible
 - $P_2 > P_1$ non è un assegnamento feasible

ALGORITMI DI SCHEDULING

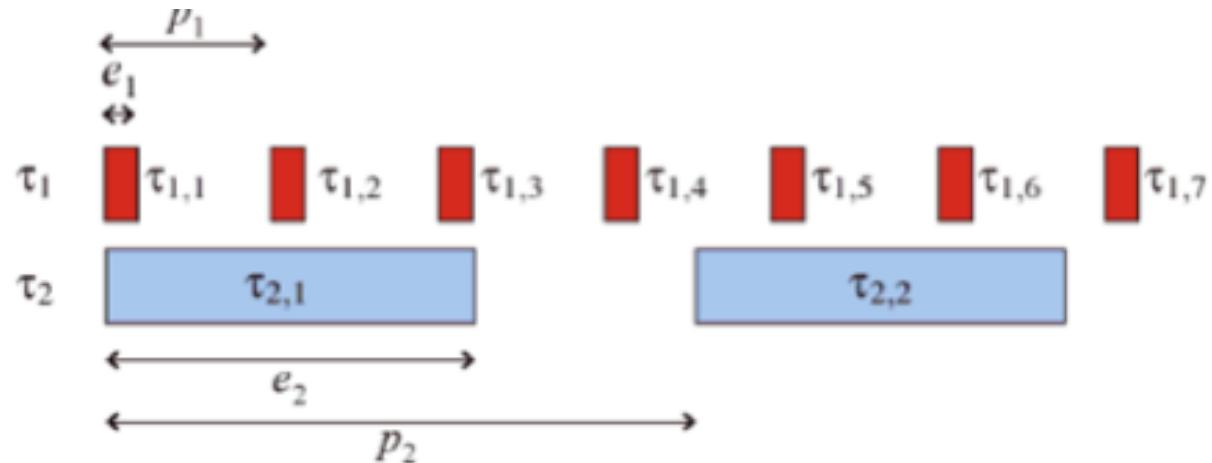
- Vari algoritmi di scheduling sono stati proposti in letteratura specificatamente per i sistemi real-time, con risultati in merito alle condizioni necessarie e sufficienti affinché esistano assegnazioni feasible
- Fra gli esempi principali:
 - **Rate Monotonic (RM)**
 - **Earliest Deadline First (EDF)**

RATE-MONOTONIC (RM)

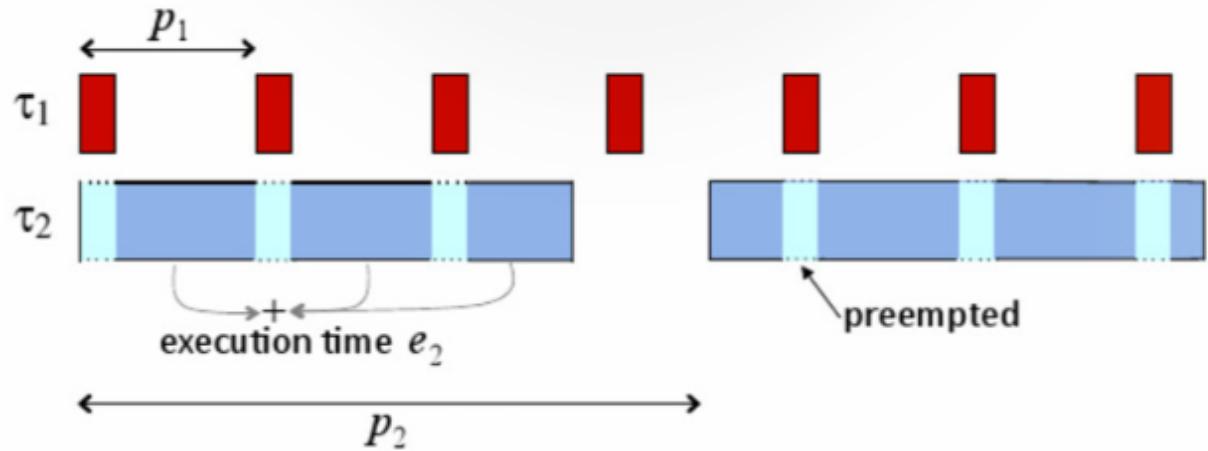
- Algoritmo di scheduling basato su **fixed-priority**
 - le priorità sono definite staticamente e non cambiano, e il loro valore è dato dall'inverso del periodo
 - per cui più piccolo è il periodo (ovvero più frequentemente il task deve essere eseguito), maggiore è la sua priorità, a prescindere dalla sua durata prevista
 - nell'esempio:
 - $p_1 < p_2$, allora a P1 viene assegnata la priorità maggiore
- **Teorema**
 - *dato uno scheduler pre-emptive fixed-priority, e un insieme finito di task periodici $T = \{T_1, T_2, T_3, \dots\}$ con associati i periodi $P = \{p_1, p_2, p_3, \dots\}$ senza vincoli di precedenza, se esiste la possibilità un assegnamento feasible (dati i valori scelti) allora l'assegnamento definito dal RM è feasible*
 - per questo motivo RM è un algoritmo ottimo (ovvero trova la soluzione ottima, se esiste) in merito alla feasibility

ESEMPIO

Due task periodici



Schedulazione
pre-emptive con
priorità
assegnata
secondo RM



(da [IES], p. 295)

EARLIEST DEADLINE FIRST (EDF)

- Questo algoritmo si basa sulla modifica dinamica delle priorità dei task
 - utilizzabile negli scheduler che consentono priorità dinamica
- EDF assegna la priorità maggiore dinamicamente (quando interviene lo scheduler) al task *con deadline più vicina*
- Anche per EDF è stato dimostrato che è un algoritmo ottimo (ovvero trova la soluzione ottima, se esiste) per la feasibility, sotto la condizione necessaria e sufficiente che dato un insieme T di N task il *parametro U di utilizzo del processore sia ≤ 1*
 - $U = \sum e_i/p_i$, per ogni task i in [0..N-1]
 - e_i = tempo di esecuzione del task i-esimo
 - p_i = periodo del task i-esimo
- EDF non è sempre applicabile
 - HW task come gli interrupt handler possono richiedere di avere priorità pre-fissate, non modificabili
 - comporta un certo overhead sullo scheduler che in alcuni casi può non essere accettabile
 - l'overhead è dato dal fatto che le priorità dei task devono essere ricomputate ad ogni evento di scheduling

BIBLIOGRAFIA

- **[ESP]** An Embedded Software Primer. David E. Simon. Addison Wesley
 - capitolo 4
- **[PES]** Programming Embedded Systems: An Introduction to Time-Oriented Programming - Frank Vahid, Tony Givargis, Bailey Miller
 - capitoli 3, 4, 5
 - <http://www.programmingembeddedsystems.com>

Sistemi Embedded e IoT

Ingegneria e Scienze Informatiche - UNIBO

a.a 2019/2020

Docente: Prof. Alessandro Ricci

[modulo 4.1]

**DAI SISTEMI EMBEDDED
AD INTERNET OF THINGS**

SOMMARIO

- Internet of Things - Introduzione
- Da IoT a Enterprise IoT
- IoT e Sistemi M2M
- IoT e Cloud
- IoT e Web of Things

INTERNET OF THINGS

- **Internet of Things (IoT)**
 - termine introdotto da Kevin Ashton nel 1999, fondatore del centro Auto-ID al MIT, su progetti relativi alle tecnologie RFID
 - obiettivo iniziale: *automatizzare l'inserimento di dati real-time in rete (Internet) relativi al mondo fisico (identificazione di oggetti, misure, eventi), mediante opportuni sensori - evitando l'inserimento manuale ad opera di persone..[^{*}]*
- Impatto molto significativo a più livelli della società [^{**}]
 - non solo tecnologico

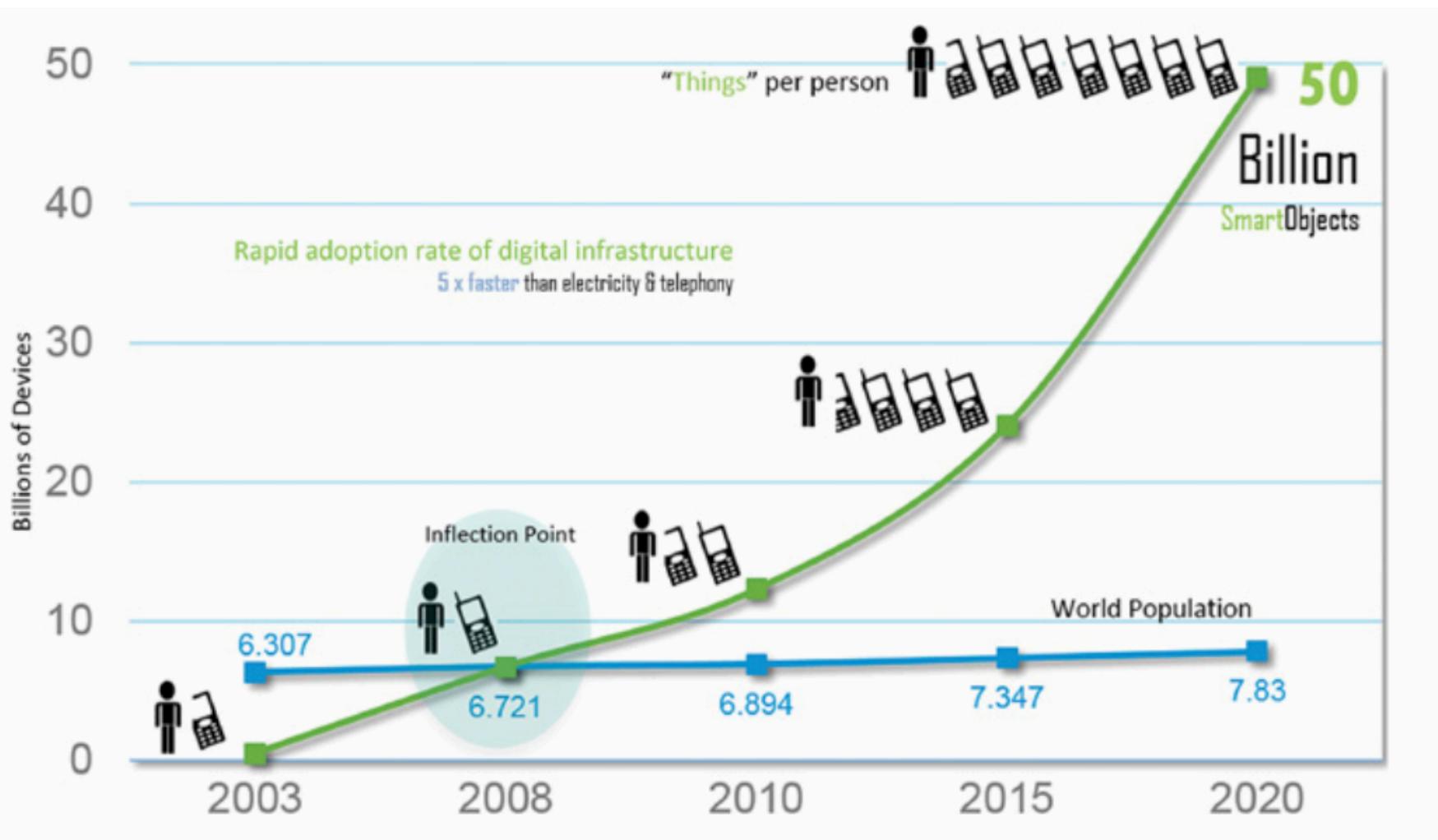
[^{*}] Kevin Ashton. “The ‘Internet of Things’ Thing”. RFID Journal, 2009. <http://www.rfidjournal.com/articles/view?4986>]

[^{**}] Samuel Greengard. *The Internet of Things*. MIT Press

IoT: UNA DEFINIZIONE

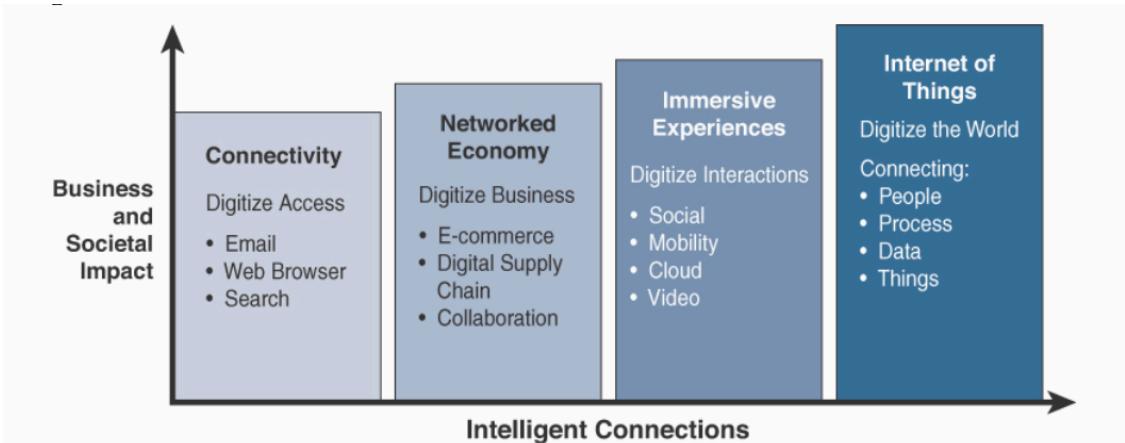
- Una definizione [LIT]:
 - “*The IoT is what we get when we connect Things, which are not operated by humans, to the Internet*”
 - sistemi costituiti da reti di oggetti fisici che interagiscono mediante la rete Internet
 - questi oggetti fisici sono tipicamente sistemi embedded
- IoT è alla base di **Industrial Internet**
 - infrastruttura che supporta la connettività ad ampia scala fra dispositivi e data
 - integrazione di sistemi embedded con sensori, software e sistemi di comunicazione
 - spesso chiamata “*Industry 4.0*” o *smart industry* o *smart manufacturing*

IoT: NUMERI



[IOTF, p.72]

EVOLUZIONE DI INTERNET



[IoTF, p.68]

Figure 1-1 Evolutionary Phases of the Internet

Internet Phase	Definition
Connectivity (Digitize access)	This phase connected people to email, web services, and search so that information is easily accessed.
Networked Economy (Digitize business)	This phase enabled e-commerce and supply chain enhancements along with collaborative engagement to drive increased efficiency in business processes.
Immersive Experiences (Digitize interactions)	This phase extended the Internet experience to encompass widespread video and social media while always being connected through mobility. More and more applications are moved into the cloud.
Internet of Things (Digitize the world)	This phase is adding connectivity to objects and machines in the world around us to enable new services and experiences. It is connecting the unconnected.

Table 1-1 Evolutionary Phases of the Internet

IoT - ELEMENTI PRINCIPALI

- “Things”
 - sistemi embedded
 - sensori, in particolare
- **Connettività e comunicazione**
 - studio di **protocolli di comunicazione** appositi
 - MQTT, XMPP, UPnP, CoAP
 - basati su Internet
 - aspetti relativi alla **interoperabilità**
 - vocabolari, ontologie
 - IoT come “*sistema di sistemi*” (System-of-Systems)
- Aspetti correlati a queste due dimensioni
 - sicurezza
 - identità, autenticazione, autorizzazioni...

SMART THINGS

- Visione semplificata [DIT]:

IoT =

Physical Objects +
Controller/Sensors/Actuators +
Internet

- ...dove le “cose” (physical object) possono essere gli oggetti più disparati che le persone utilizzano quotidianamente sia in ambito di lavoro, sia in ambito casalingo, ludico, etc.
- Visione ***enchanted objects*** - D. Rose
 - <http://tedxtalks.ted.com/video/TEDxBerkeley-David-Rose-Enchant>

ESEMPI DI SMART THINGS

- Fitbit force wristband
 - <http://www.fitbit.com/>
- Metromile
 - <https://www.metromile.com/>
- Kevo smart lock
 - <http://www.kwikset.com/kevo/>
- Google Nest thermostat & family
 - <https://nest.com/>
- WeMo light switch & family
 - <http://www.belkin.com/us/p/P-F7C030/>
- enchanted objects
 - <http://enchantedsobjects.com/products/>

RUOLO DEGLI SMARTPHONE

- Nel contesto di IoT, gli smartphone possono svolgere vari ruoli
 - fungere essi stessi da sistemi embedded,
 - dotati di opportuni sensori, per raccogliere e inviare dati geo-localizzati
 - fungere da “telecomando” universale
 - interfaccia utente (UI) unificata per interagire con altri smart things
 - fungere da unità di controllo collegata in rete che interagisce (tipicamente via bluetooth) con altri dispositivi wearable e non

IL RUOLO DELL'IDENTIFICAZIONE

- In origine, l'elemento caratterizzante di IoT era dato dalla tecnologia **RFID** (**Radio-Frequency Identification**)
 - tecnologia per l'identificazione e/o memorizzazione automatica di informazioni inerenti oggetti basata sulla capacità di memorizzazione di dati da parte di particolari etichette elettroniche, chiamate *tag* (o anche *transponder*), e sulla capacità di queste di rispondere all'interrogazione a distanza da parte di appositi apparati fissi o portatili, chiamati
 - tag passivi (non alimentati), attivi, semi-passivi
 - alcuni parametri di riferimento
 - distanza fino a 2 m (per i passivi), capacità di memorizzazione dell'ordine del kilobyte
 - recente evoluzione: **NFC** (Near-Field Communication)
 - piccole distanze (<10cm), maggiore velocità di trasmissione (424 kBit/s), interazione direttamente fra 2 lettori
- Oggi l'identificazione è solo un aspetto
 - *“when we talk about an Internet of Things, it is not just putting RFID tags on some dumb things so we smart people know where that dumb thing is. It's about embedding intelligence so things become smarter and do more than they were proposed to do”* (Nicholas Negroponte, MIT)

EVOLUZIONE DI IoT

- 5 stadi [Por14,Por15,DBE17]

1. product stage

- *the air conditioner*

2. smart product

- *the programmable air conditioner*

3. smart connect products

- *air conditioner accessible by the Internet*
 - controllable by the phone
 - the company can check its functioning and compare to other millions of other units to do *predictive maintenance*

...

EVOLUZIONE DI IoT

- 5 stadi [Por14,Por15,DBE17] (cont.)

1. product systems

- *the smart thermostat talks to the connected HVAC and the smart window blinds and heated floors*
- key points
 - **interoperability** & communication standards
 - » Web of Things (WoT) Initiative
 - **command and control platforms** availability
 - » Apple HomeKit, Amazon Echo, Google Home, Samsung SmartThings...
 - » industrial counter part: GE's Predix and Hitachi's Lumada
 - 5) systems of systems

EVOLUZIONE DI IoT

- 5 stadi [Hep14,DBE17] (cont)

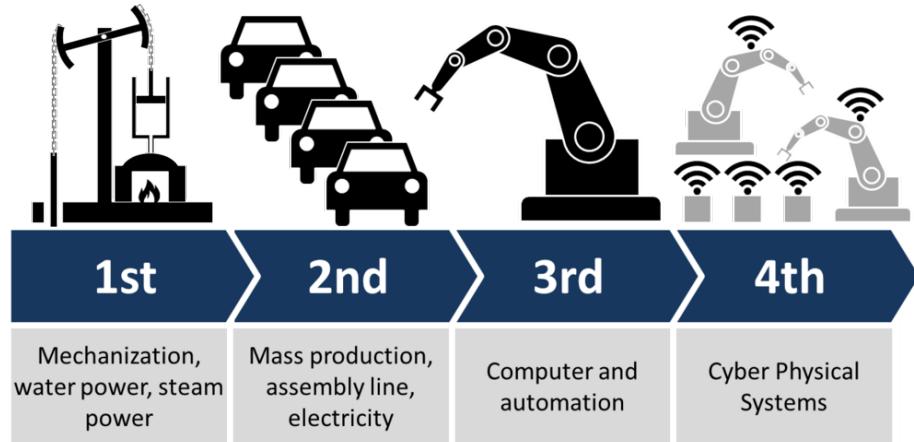
5. system of systems

- home appliances talking with home security talking to the car and wearable devices talking to smart hospital...
- key points
 - interoperability
 - scalability, openness
 - governance

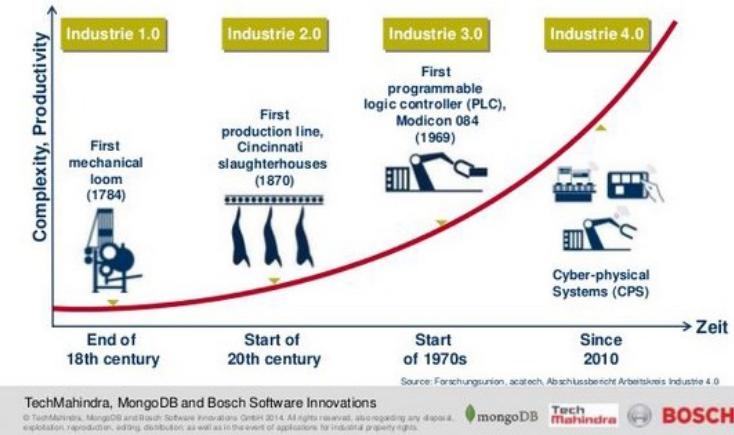
DA IOT A *ENTERPRISE IOT*

- IoT come tecnologia chiave per industria e imprese
 - **Industry 4.0**
 - quarta rivoluzione industriale
- Aspetto chiave: “Things” come mezzo per raccogliere dati mediante sensoristica opportuna
 - *big-data, real-time, big-stream*

INDUSTRY 4.0



Industrie 4.0: The next Industrial Revolution



<http://www.eesc.europa.eu/?i=portal.en.group-1-new-news.34501>

LE QUATTRO RIVOLUZIONI INDUSTRIALI

Industry 4.0: IoT Integration (Today)
Sensors with a new level of interconnectivity are integrated

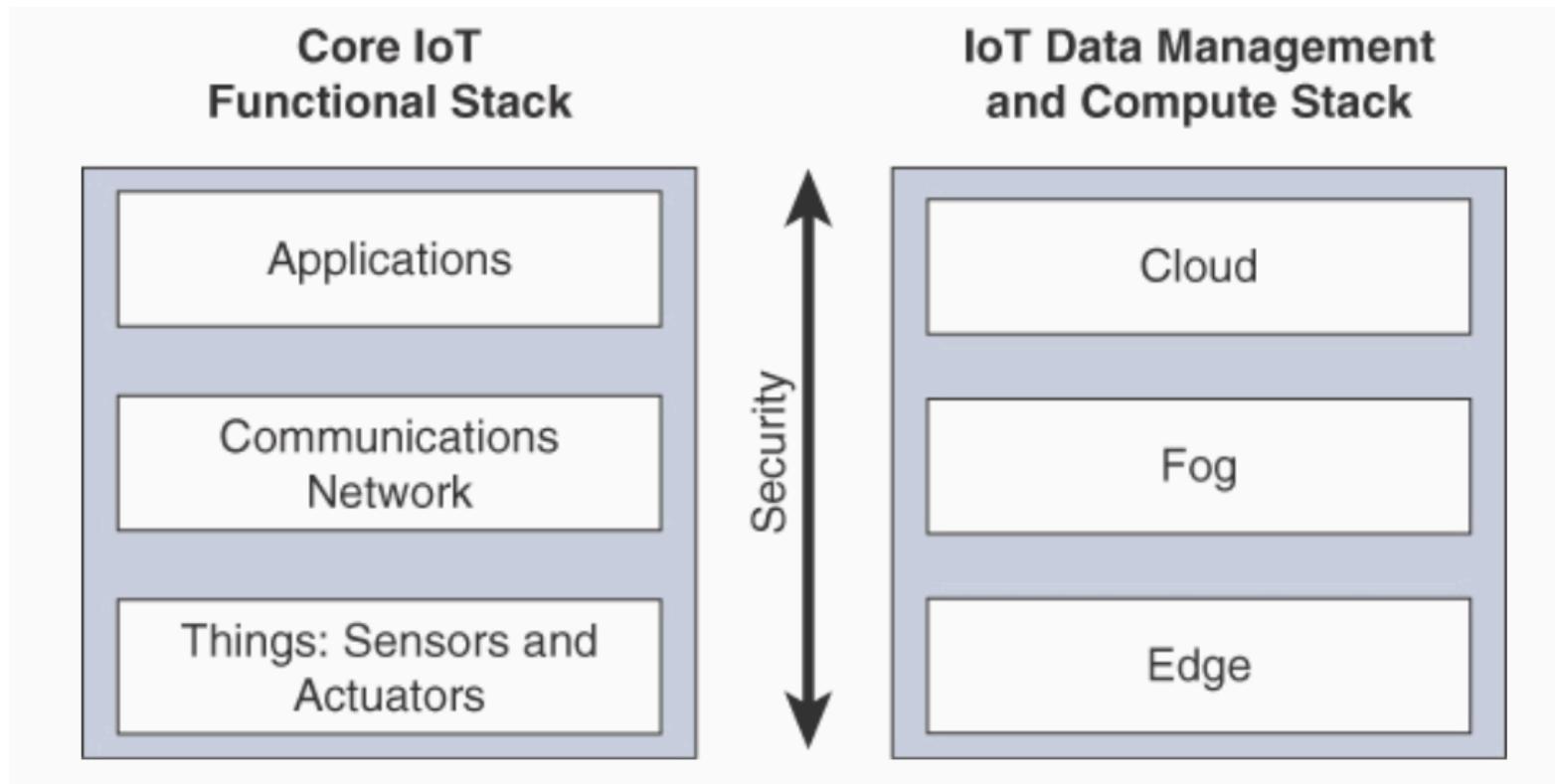
Industry 3.0: Electronics and Control (Early 1970's)
Production is automated further by electronics and IT

Industry 2.0: Mass Production (Early 20th Century)
Division of labor and electricity lead to mass production facilities

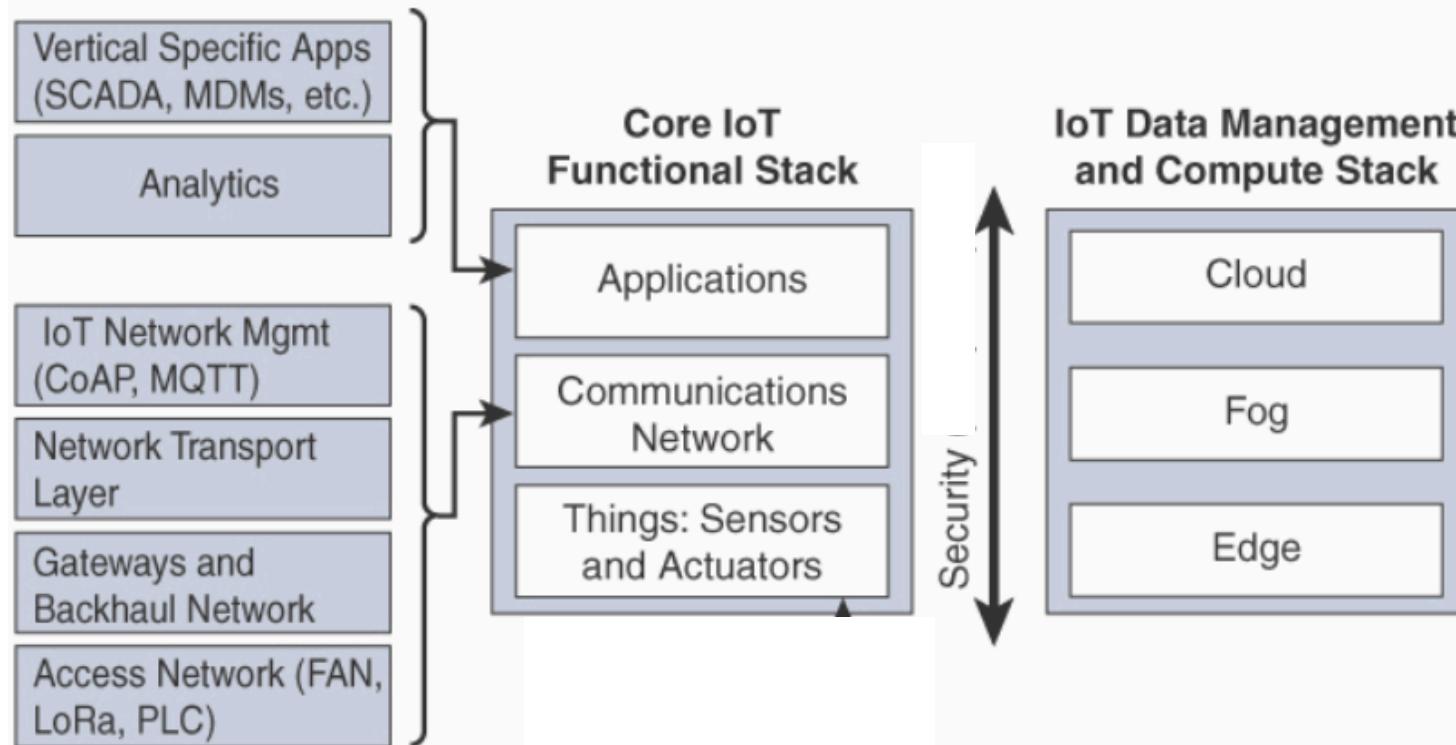
Industry 1.0: Mechanical Assistance (Late 18th Century)
Basic machines powered by water and steam are part of production facilities

[IoTF, 85]

ARCHITETTURA SEMPLIFICATA DI UN SISTEMA (ENTERPRISE) IOT



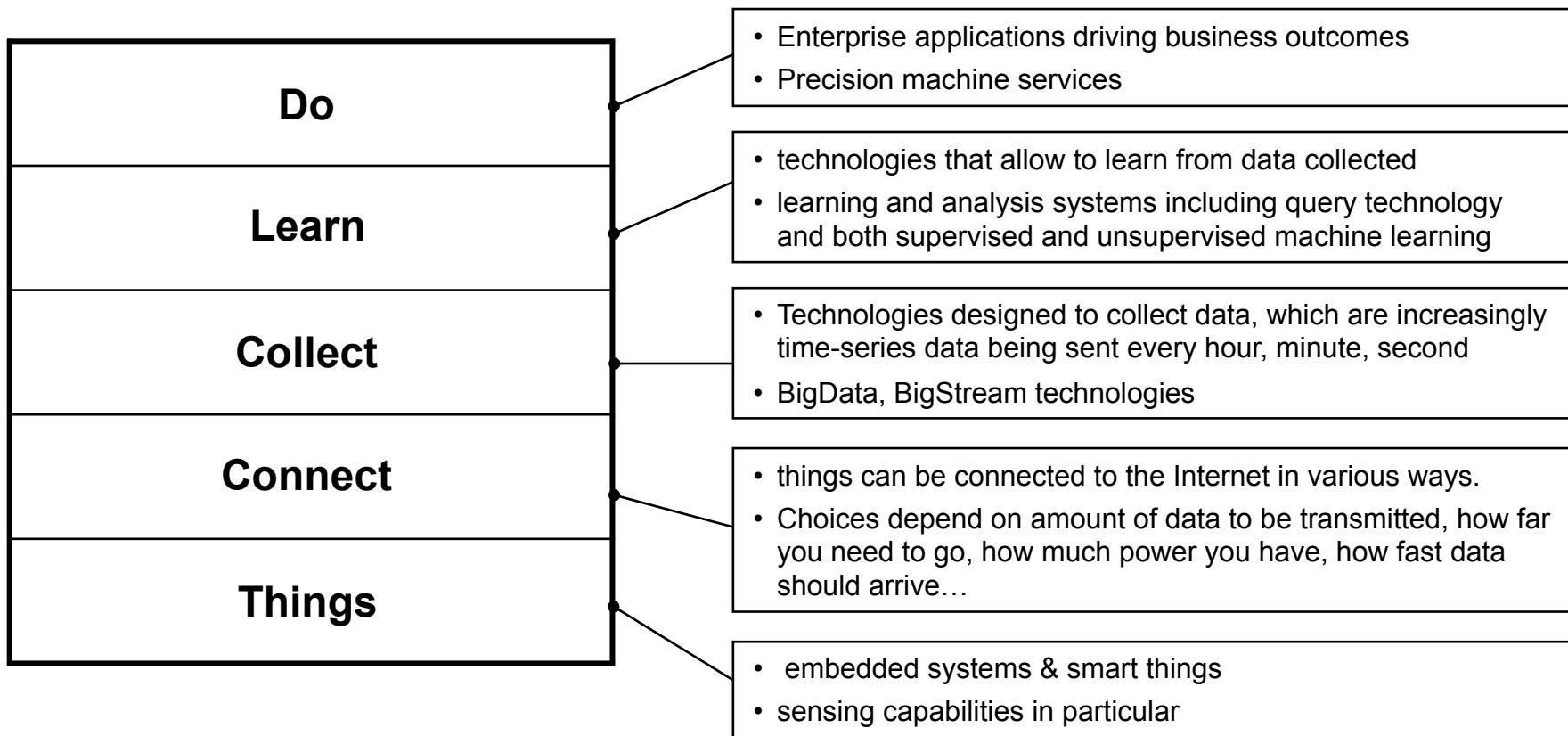
ARCHITETTURA SEMPLIFICATA DI UN SISTEMA (ENTERPRISE) IOT



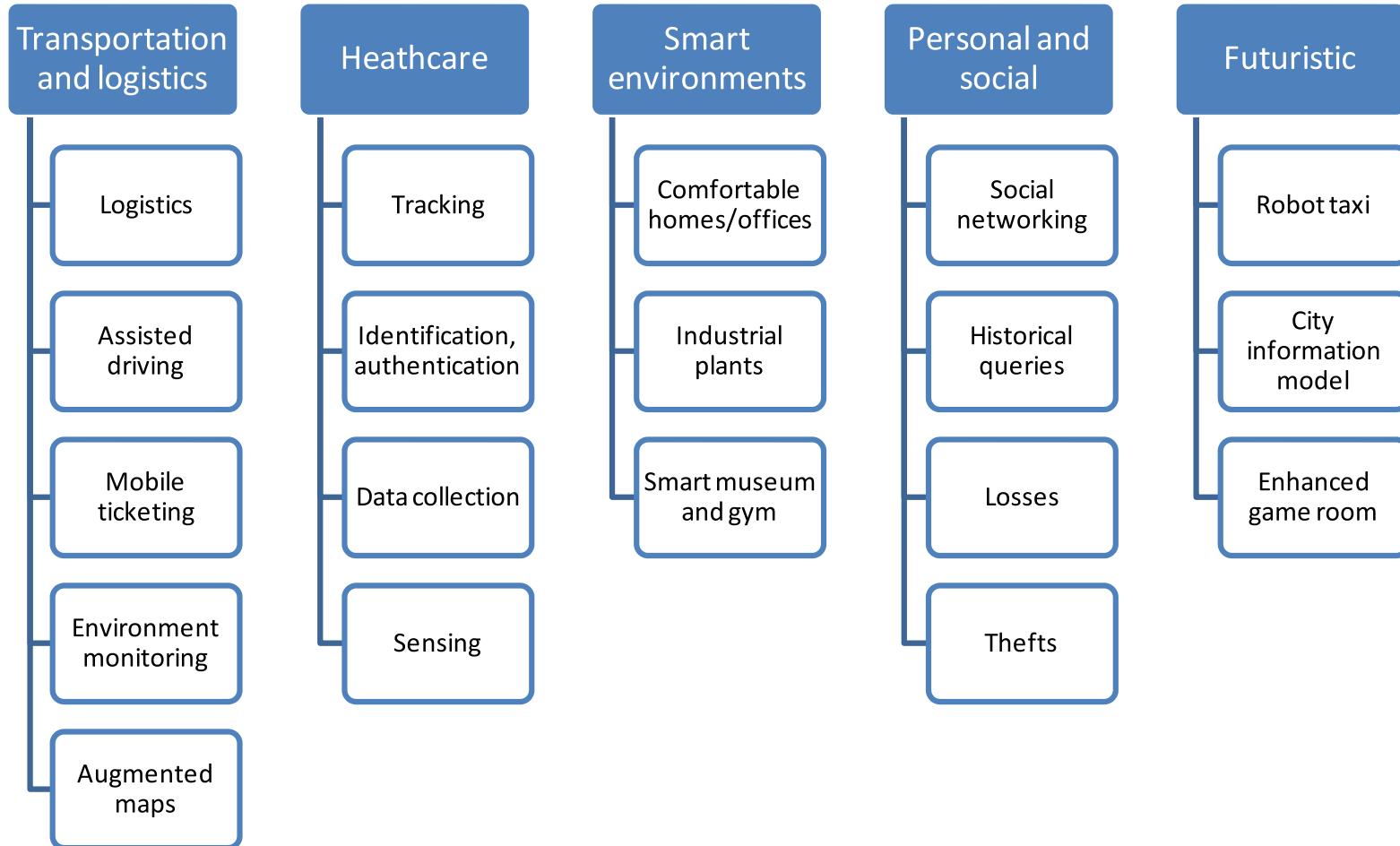
ENTERPRISE IOT (INDUSTRIAL INTERNET)

- [Framework concettuale per identificare i vari livelli che riguardano IoT a livello Enterprise (*)

“Precision-XXX”

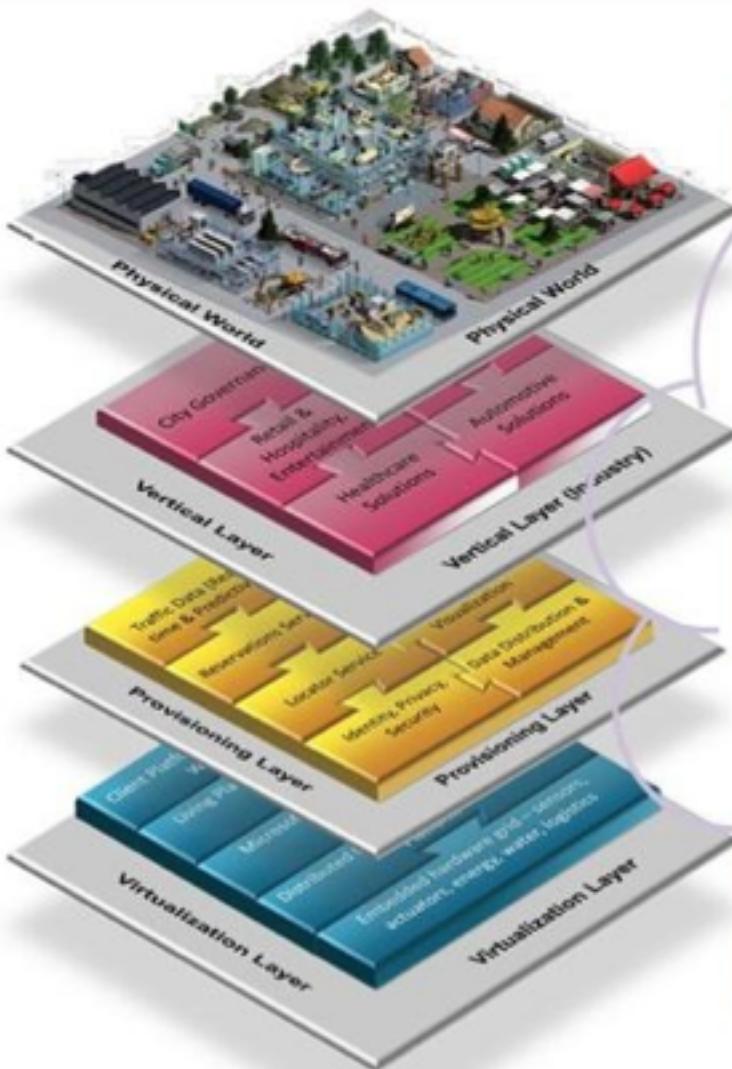


PRINCIPALI AMBITI APPLICATIVI



Tratto da: [Luigi Atzori, Antonio Iera, Giacomo Morabito.
The Internet of Things: A survey. Computer Networks, 54 (2010)]

SMART CITY, URBAN OS & APPS



(Urban OS, Plan IT)

Examples: Future Scenarios

Weather

Building and vehicle sensors provide microclimate information enabling improved models; transportation and HVAC systems take current local weather into account

Emergency Services

Improved emergency planning and response using real time information and signaling

Sustainability & Utilities

Smart grid, water efficiency, energy demand and supply management, shaping, and remote control

Transportation

Congestion avoidance, green routes, integrated systems of buses, trains, taxis, cars (with parking/charging), bikes, walking, and PRT

Vehicles

Onboard energy generation and/or storage

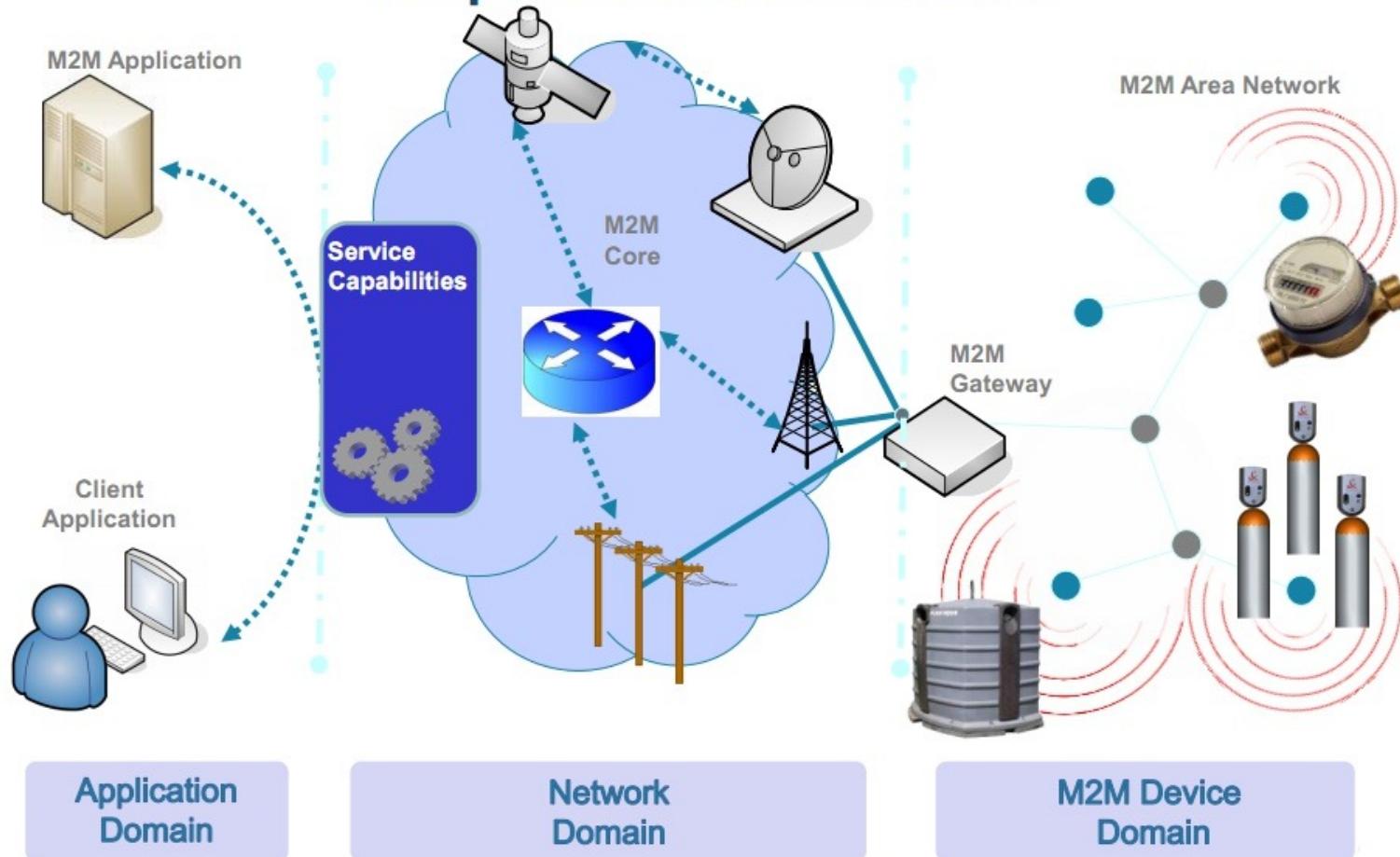
IOT E SISTEMI MACHINE-TO MACHINE (M2M)

- Nel contesto industriale, Enterprise IoT/Industrial Internet può essere inquadrato come evoluzione dei sistemi Machine-To-Machine (M2M)
 - Insieme di tecnologie che supportano la **comunicazione** wired e wireless fra dispositivi/macchine al fine di permettere **scambio di informazioni** locali
 - esempio: temperatura, stato del dispositivo,...
 - utilizzati per lo più per funzioni di **monitoraggio e controllo**
 - M2M è esistita in forme differenti sin dalla creazione delle prime reti e sistemi di automazione
 - in particolare è stata utilizzata in applicazioni come telemetria, automazione industriale



World Class Standards

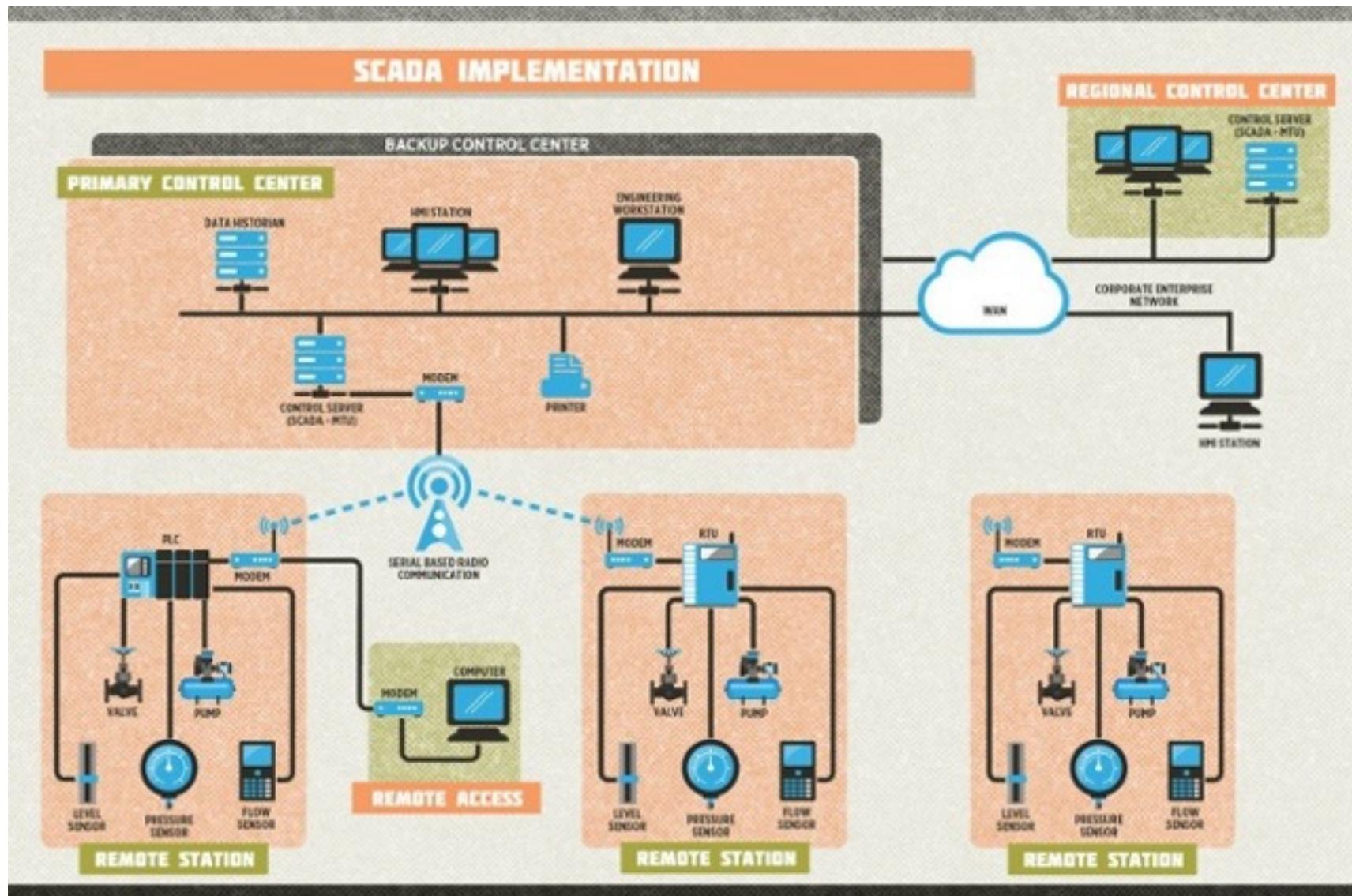
Simple M2M Architecture



SISTEMI SCADA

- In ambito industriale, un tipo di sistema M2M molto diffuso è dato dai sistemi **SCADA** (Supervisory Control and Data Acquisition)
 - *sistemi di supervisione* che operano con codifiche di segnali trasmesse per canali di comunicazione al fine di realizzare un **controllo remoto di sistemi/equipaggiamenti/impianti**
 - il sistema di supervisione può essere combinato con sistemi di acquisizione dei dati attraverso l'aggiunta di ulteriori codifiche di segnali e comunicazioni al fine di acquisire informazioni circa lo stato dei dispositivi remoti per poterle visualizzare o registrare
- E' un tipo di **sistema di controllo industriale (ICS)**
 - sistemi computerizzati che *monitorano e controllano i processi industriali* in un certo ambiente fisico
 - i sistemi SCADA si distinguono storicamente da altri ICS per scala, ovvero tipicamente riguardando processi a larga scala che includono molteplici siti, distribuiti in un territorio che può essere geograficamente molto vasto

SISTEMI SCADA - ARCHITETTURA



<http://blog.cimation.com/blog/key-differences-between-scada-dcs-and-hmi-systems>

SCADA: PRINCIPALI ELEMENTI

- **Sensori e Remote terminal units (RTUs)**
 - alla base di uno SCADA c'è la rete di sensori
 - permette di monitorare il funzionamento e lo stato dei macchinari e dei processi industriali
 - collegati ai sensori troviamo le remote terminal unit o RTU (“Unità terminali remote”)
 - compito di raccogliere i dati rilevati dai sensori e trasformarli in segnali digitali da inviare alla postazione di controllo remota

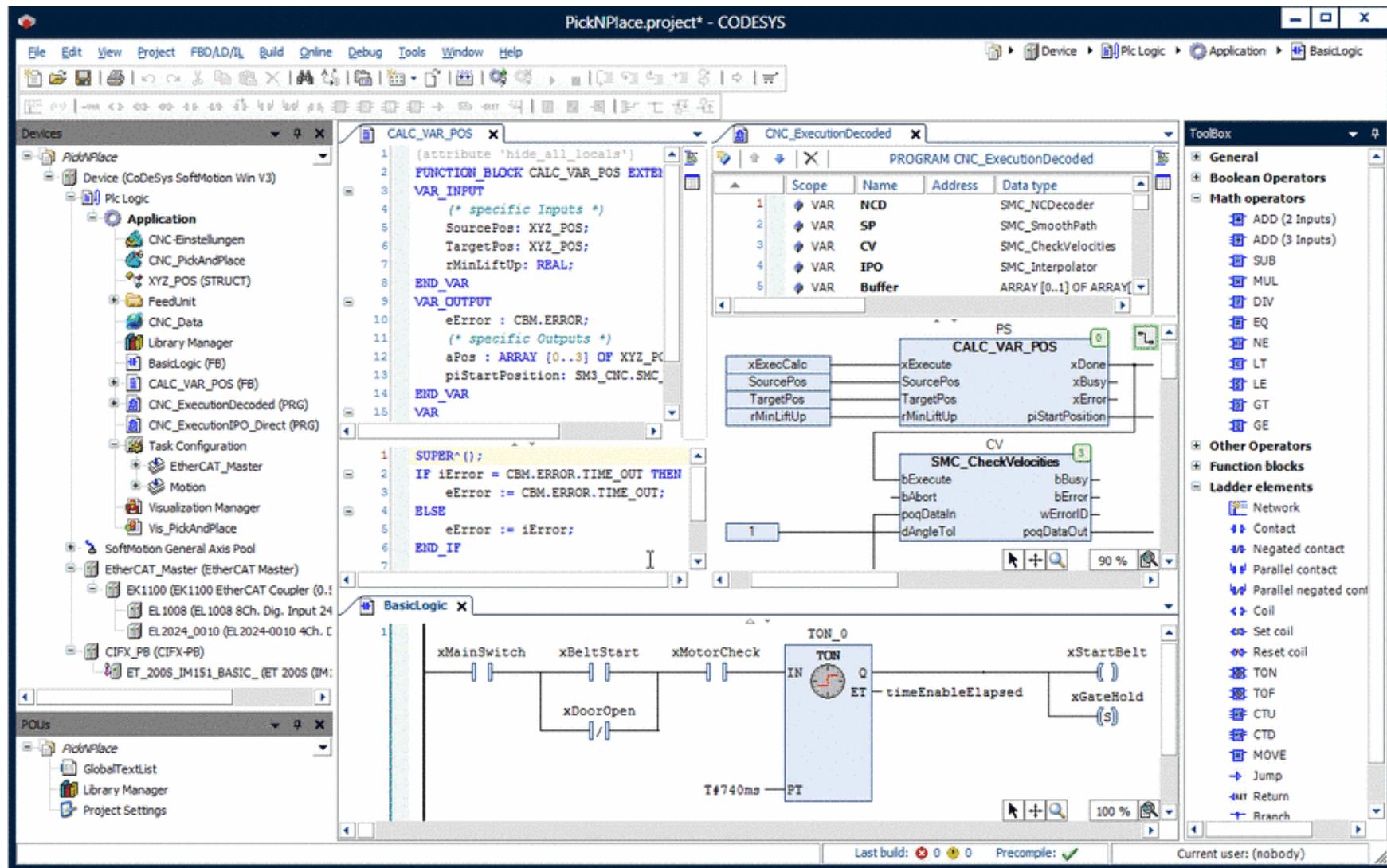
SCADA: PRINCIPALI ELEMENTI

- **Programmable logic controller (PLCs)**
 - alla rete di sensori ed RTU è collegato uno o più programmable logic controller o PLC (“Controller logico programmabile”)
 - compito di supervisionare la raccolta di informazioni, fornendo istruzioni sia alle RTU sia direttamente ai sensori
 - è il PLC a indicare alla rete sensoriale quali sono gli intervalli di tempo nei quali effettuare le misurazioni e controllare i valori dei macchinari
 - programmabili con linguaggi di programmazione della famiglia IEC 61131-3

PROGRAMMAZIONE DI PLC CON IEC 61131-3

- IEC 61131-3 è la terza parte (di 8) dello standard internazionale aperto IEC 61131 per PLC (programmable logic controllers).
- Questa parte concerne i linguaggi di programmazione e definisce un insieme di linguaggi di programmazione standard per PLC:
 - **Ladder diagram (LD)**, visuale
 - **Function block diagram (FBD)**, visuale
 - **Structured text (ST)**, testuale
 - **Instruction list (IL)**, testuale
 - **Sequential function chart (SFC)**
 - possiede elementi per organizzare i programmi di controllo sia sequenziali che paralleli.

PROGRAMMAZIONE DI PLC CON IEC 61131-3



SCADA: GLI ALTRI ELEMENTI

- Un sistema di **telemetria**, che connette PLC e RTU con centri di controllo, data warehouse, e sistemi enterprise
 - può essere una rete informatica come una LAN o una WAN, o più semplicemente da una rete di linee seriali, sia wired che wifi
- **Computer supervisore** (MTU, Master Terminal Unit)
 - server di controllo che raccoglie periodicamente i dati dai PLC, li elabora per ottenerne informazioni utili, memorizza le informazioni, invia comandi di controllo
- Un **interfaccia uomo-macchina** (HMI, human machine interface)
 - permette ad operatori umani di accedere ai dati processati, monitorarli e interagirvi
- Un **historian**
 - servizio software che memorizza time-stamped data, eventi e allarmi, in un data base che viene poi interrogato al fine di creare report grafici nel HMI

SCADA HMI - ESEMPIO



<http://www.armani-engr.com/hmiscada.html>

IOT E CLOUD

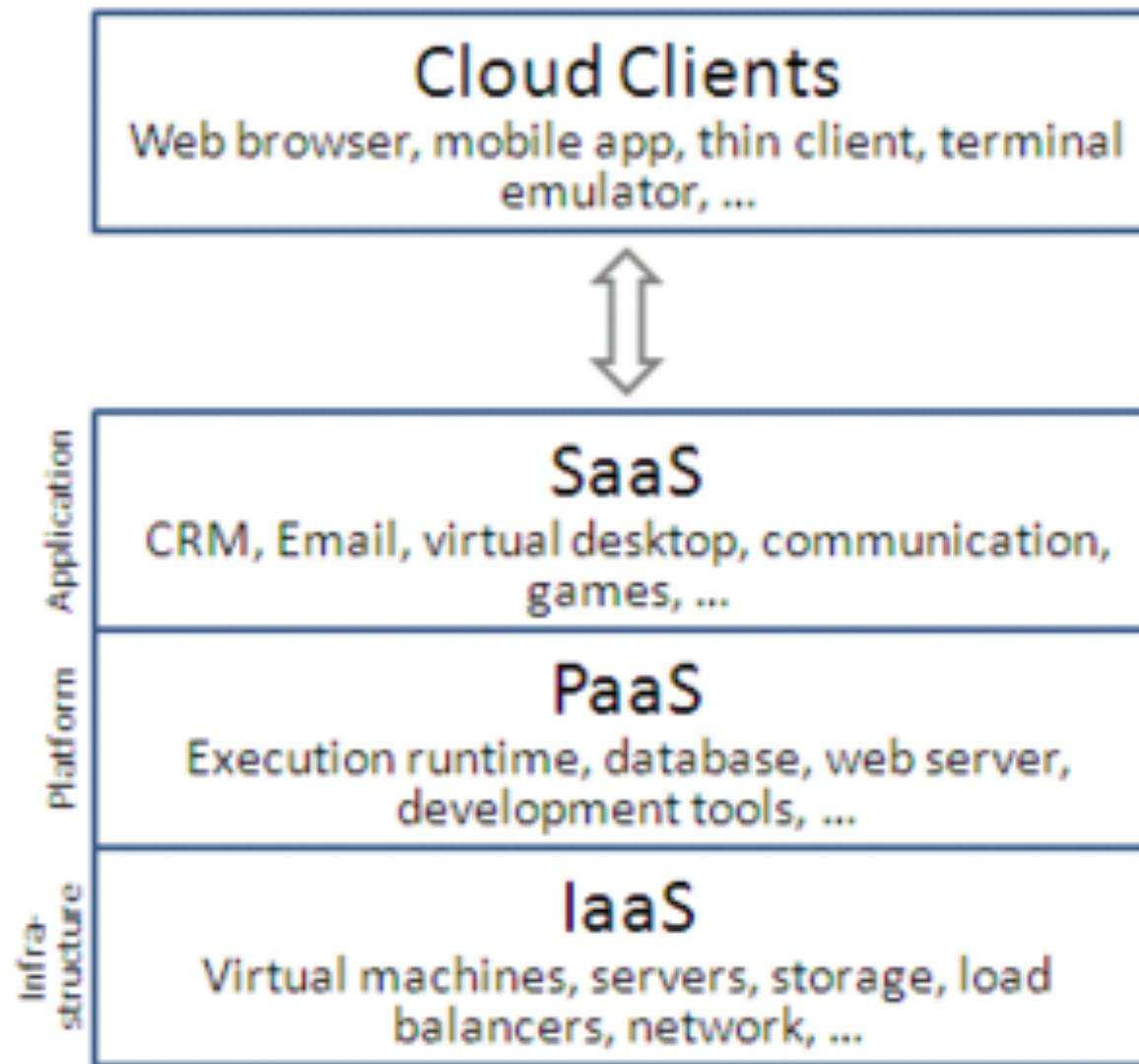
- Caratteristica fondamentale IoT: *capacità di comunicare direttamente o indirettamente con la rete Internet*
- Ciò permette di gestire in modo opportuno grandi volumi e stream di dati generati dai sensori
 - che non potrebbero essere memorizzati in locale
- Tali informazioni diventano quindi accessibili mediante opportuni servizi Internet o Web, che ne permettono lo scambio con altre applicazioni, in particolare con applicazioni mobile
- Ruolo importante del **cloud**
 - servizi a disposizione per gestione dispositivi, storage dati, analisi offline e online, ...
 - sistemi aperti

IL CLOUD

- “Cloud computing”
 - *“paradigma di erogazione di risorse informatiche, come l'archiviazione, l'elaborazione o la trasmissione di dati, caratterizzato dalla disponibilità on demand attraverso Internet a partire da un insieme di risorse preesistenti e configurabili” (*)*
- le risorse vengono date come-servizio (**as-a-service**) in rete e possono essere a tre livelli diversi
 - **IAAS** - infrastructure as a service
 - macchine virtuali, server, storage, network
 - **PAAS** - platform as a service
 - execution runtime, database, web servers, IDEs,...
 - **SAAS** - software as a service
 - CRM, giochi, office apps, mail apps, etc.
- Lato client (desktop, mobile, wearable, embedded...)
 - browser moderni - HTML5 e JavaScript
 - qualsiasi app che usi HTTP API

(*) Peter Mell, Timothy Grance, The NIST Definition of Cloud Computing. NIST, Special Publication 800-145, Settembre 2011.

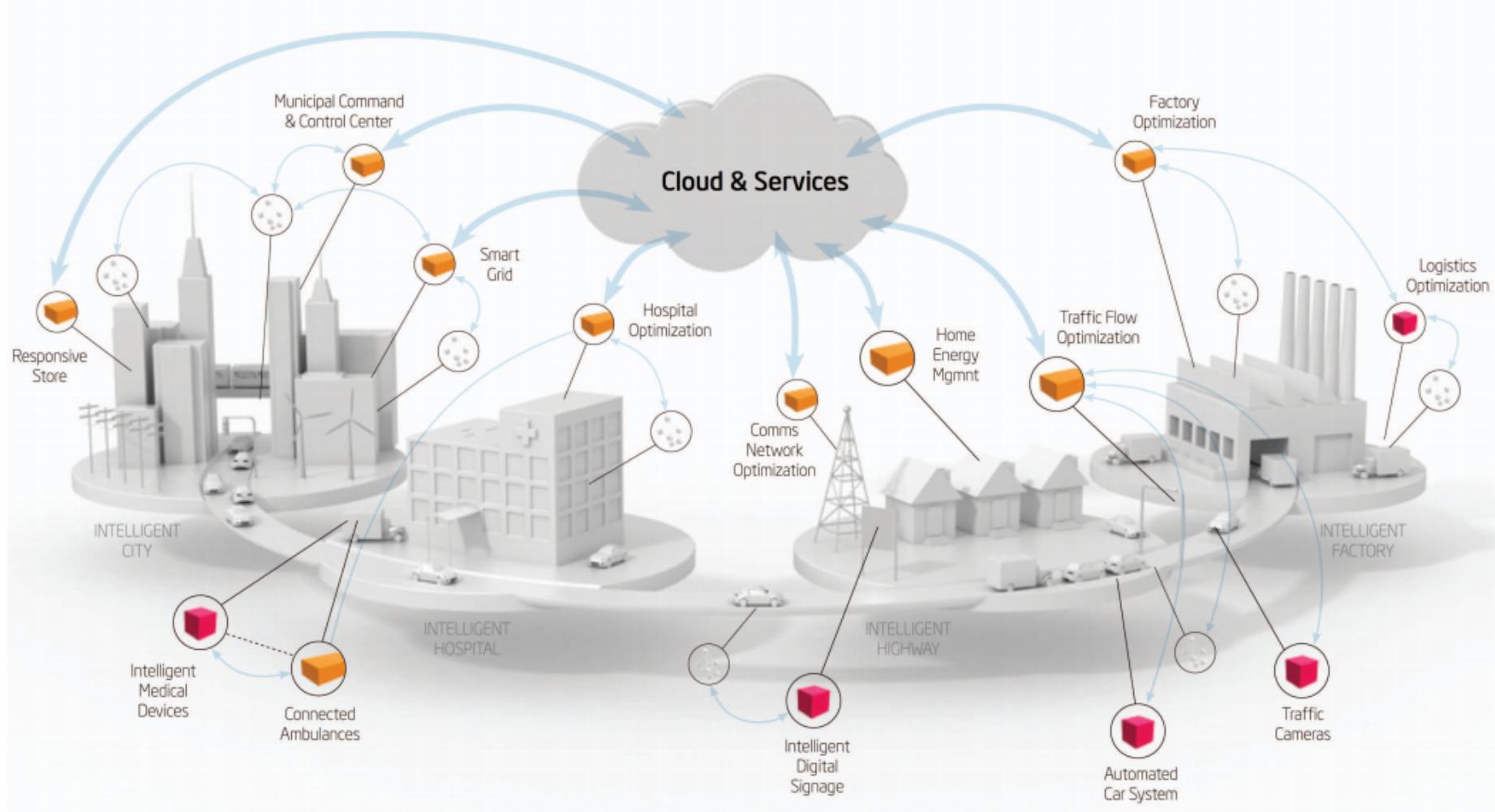
LIVELLI CLOUD



IL CLOUD E IOT

- Nel caso di IoT, i servizi cloud vengono primariamente utilizzati per
 - raccolta dati inviati dai dispositivi
 - stream
 - accesso ai dati inviati dai dispositivi
 - pull, event-driven
 - analisi offline e online
 - gestione “aperta” dei dispositivi
- **Big Data**
 - il volume e la velocità dei dati generati può essere tale da richiedere tecniche di memorizzazione, elaborazione e accesso che vanno al di là di quelle fornite dai tradizionali data-base
 - queste tecniche e architetture sono esplorate nell’ambito “big data”

IOT E IL CLOUD



IOT: ASPETTI CRITICI E SFIDE

- Sicurezza
- Privacy
- Problema della proprietà dei dati
- Scalabilità e *Interoperabilità*
 - definizione di standard e architetture di riferimento
 - fra le iniziative significative: **Web of Things** (WoT)

IoT + WEB = *WEB-OF-THINGS*

- **Web of Things** (o **WoT**) consiste in sistemi che incorporano gli oggetti fisici di uso quotidiano nel **World Wide Web** dando loro una REST-ful API
 - questo facilita la creazione di profili virtuali degli oggetti e la loro integrazione e riuso in tipi diversi di applicazioni
- E' una evoluzione di IoT dove il punto fondamentale concerne come gli oggetti comunicano mediante opportuno livello di rete
 - Zigbee, Bluetooth, 6LoWPAN...

Guinard, Dominique; Vlad Trifa; Erik Wilde (2010). "A Resource Oriented Architecture for the Web of Things". Proc. of IoT 2010 (IEEE International Conference on the Internet of Things). Tokyo, Japan.

WoT = IoT + APPLICATION LEVEL

Easier to program, faster to integrate data and services, simpler to prototype, deploy, and maintain large systems.

Web:
HTTP, HTML, JSON, ...

Application level
(OSI layer 7)

Web of Things:
HTTP, JSON, WebSockets, ...

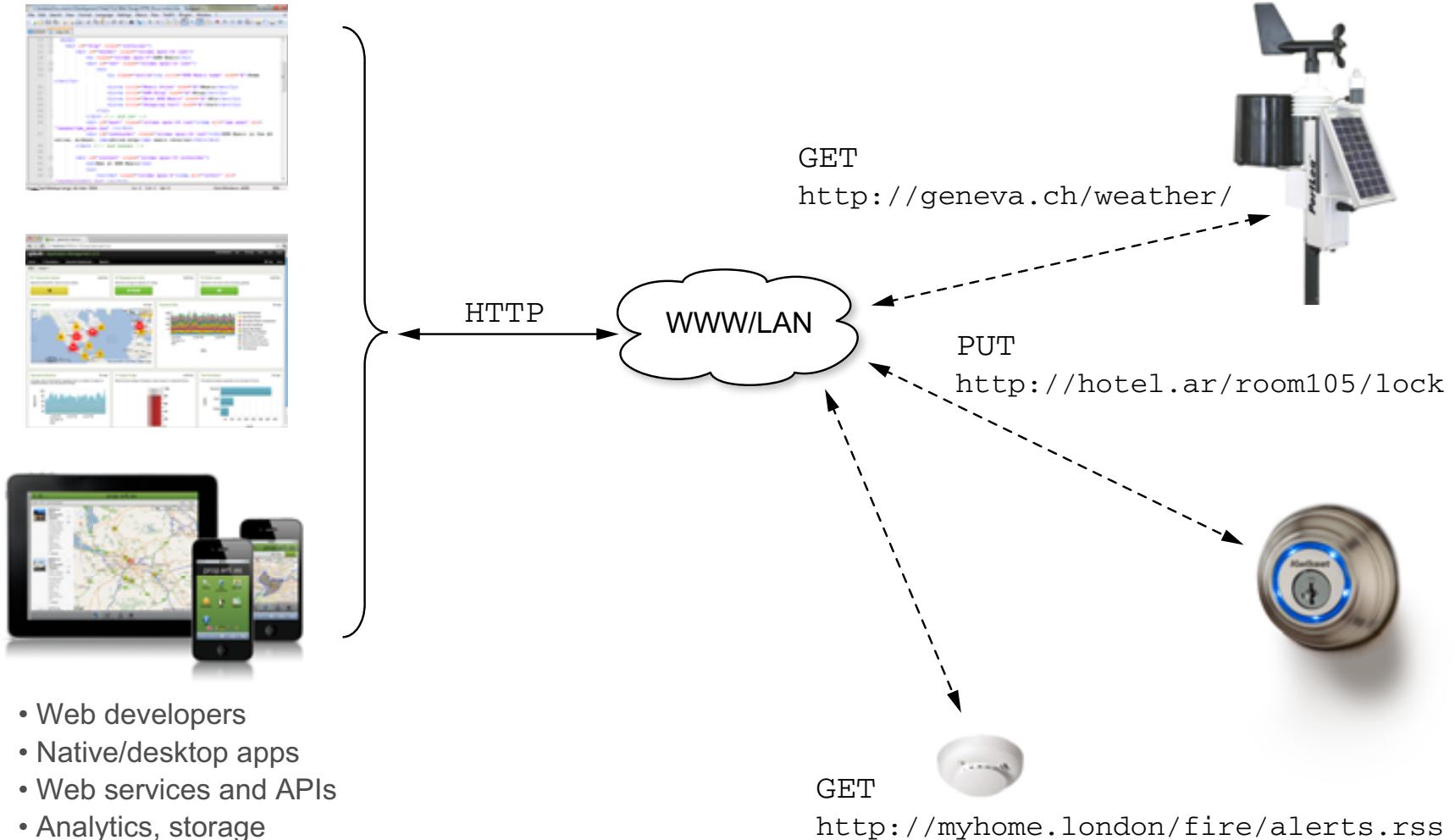
Internet:
TCP/IP, Ethernet, ...

Encoding and Transport
(OSI layers 1-6)

Internet of Things:
Bluetooth, ZigBee, Wi-Fi,...

More lightweight and optimized for embedded devices
(reduced battery, processing, memory and bandwidth usage),
more bespoke and hard-wired solutions.

WEB OF THINGS (WoT)



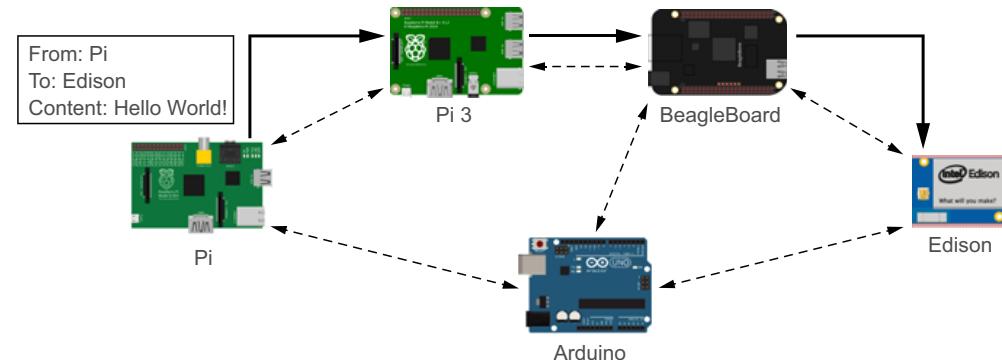
- Web developers
- Native/desktop apps
- Web services and APIs
- Analytics, storage

WOT - TOPOLOGIE DI RETE

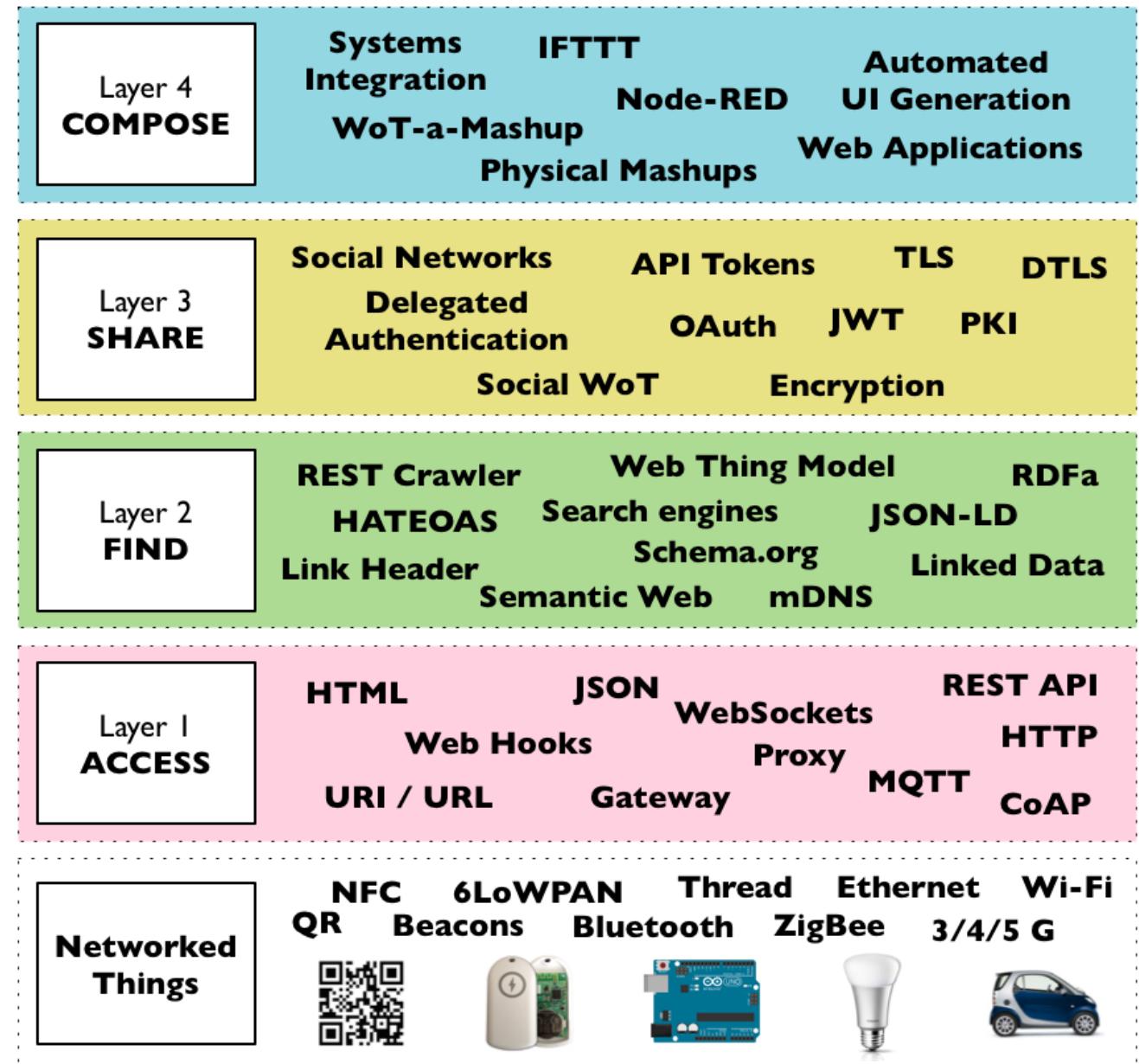
- Star topologies



- Mesh topologies
 - mesh networks
 - relays



WOT LIVELLI



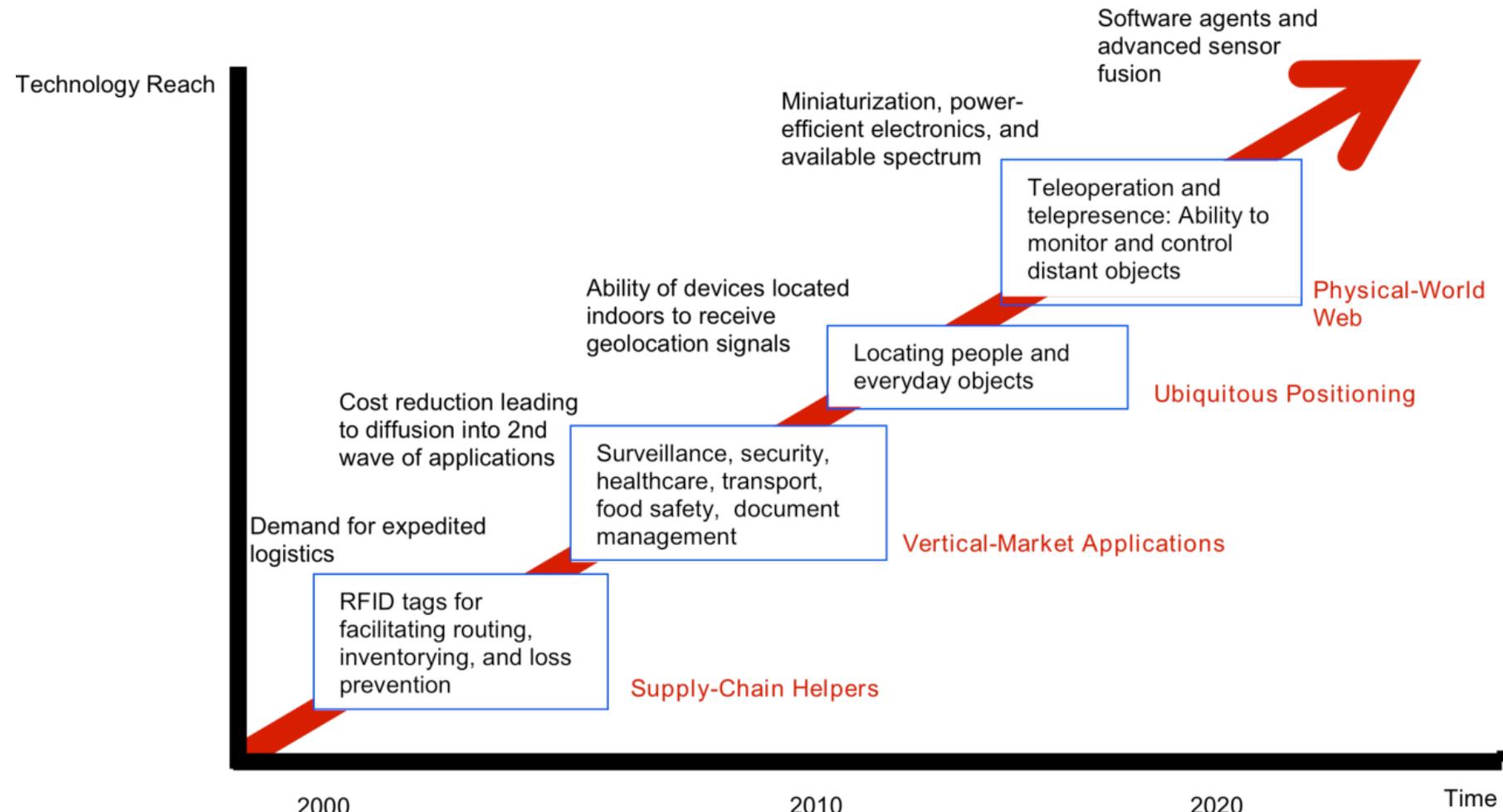
Source: Building the Web of Things: book.webofthings.io
Creative Commons Attribution 4.0

WEB OF THINGS AT W3C

- Web of Things Interest Group
 - <https://www.w3.org/WoT/>
- Web of Things Working group
 - <https://www.w3.org/WoT/WG/>
- First Submission of the Web Thing Model
 - <https://www.w3.org/Submission/2015/01/>
- Current proposals
 - WoT Architecture
 - <https://w3c.github.io/wot-architecture/>
 - WoT Things Description
 - <https://www.w3.org/TR/wot-thing-description/>

IOT: TECHNOLOGY ROADMAP

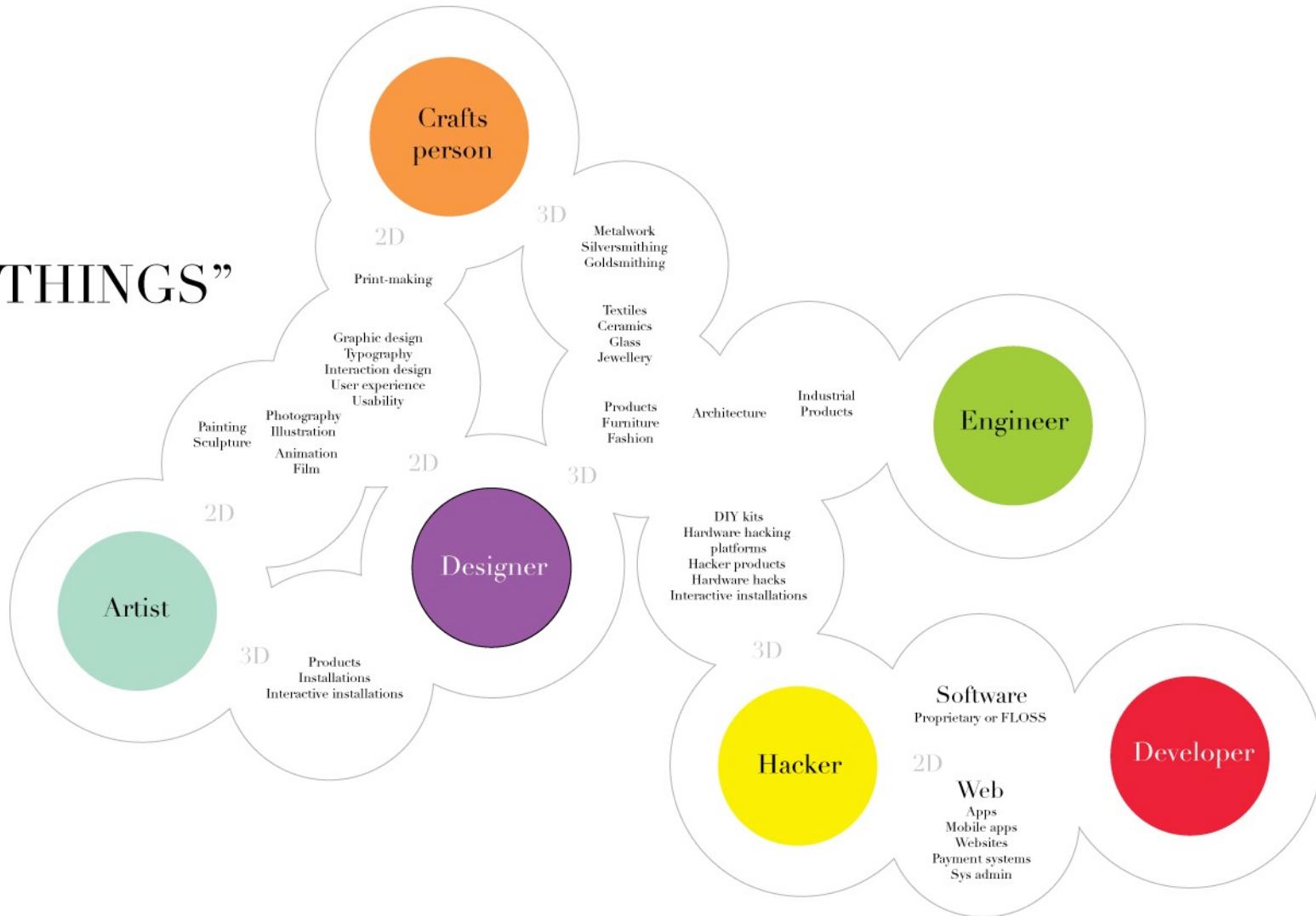
TECHNOLOGY ROADMAP: THE INTERNET OF THINGS



Source: SRI Consulting Business Intelligence

WHO IS MAKING IOT: MAKERS AND TINKERERS

“I MAKE THINGS”



BIBLIOGRAFIA E SITOGRAFIA

- Kevin Ashton. “*The ‘Internet of Things’ Thing*”. RFID Journal, 2009. <http://www.rfidjournal.com/articles/view?4986>
- Luigi Atzori, Antonio Iera, Giacomo Morabito. *The Internet of Things: A survey*. Computer Networks, 54 (2010)
- **Samuel Greengard.** *The Internet of Things. MIT Press*
- **[IOTF]** *IoT Fundamentals. Networking Technologies, Protocols, and Use Cases for the Internet of Things*. Hanes et al. CISCO press. 2017
- **[DIT]** A. McEwen and H. Cassimally. *Designing the Internet of Things*. Wiley
- **[LIT]** P. Waher. *Learning the Internet of Things*. Pack publishing.
- **[BIT]** Charalampos Doukas. *Building Internet of Things with the Arduino*.
- Guinard, Dominique; Vlad Trifa; Erik Wilde (2010). "A Resource Oriented Architecture for the Web of Things". Proc. of IoT 2010 (IEEE International Conference on the Internet of Things). Tokyo, Japan
- **[Por14]** M. Porter, J. Heppelmann. How Smart, Connected Products Are Transforming Competition. Harvard Business Review. November 2014
- **[Por15]** M. Porter, J. Heppelmann. How Smart, Connected Products Are Transforming Companies. Harvard Business Review. October 2015
- **[DBE17]** D. De Loach, E. Berthelsen, W. Elrifai. The Future of IoT. 2017

Sistemi Embedded e IoT

Ingegneria e Scienze Informatiche - UNIBO

a.a 2020/2021

Docente: Prof. Alessandro Ricci

[modulo 4.1]

DAI SISTEMI EMBEDDED
AD INTERNET OF THINGS

SOMMARIO

- Internet of Things - Introduzione
- Da IoT a Enterprise e Industrial IoT
- IoT e Web of Things

INTERNET OF THINGS

- **Internet of Things (IoT)**
 - termine introdotto da Kevin Ashton nel 1999, fondatore del centro Auto-ID al MIT, su progetti relativi alle tecnologie RFID
 - obiettivo iniziale: *automatizzare l'inserimento di dati real-time in rete (Internet) relativi al mondo fisico (identificazione di oggetti, misure, eventi), mediante opportuni sensori - evitando l'inserimento manuale ad opera di persone..[*)*
- Impatto molto significativo a più livelli della società [**]
 - non solo tecnologico

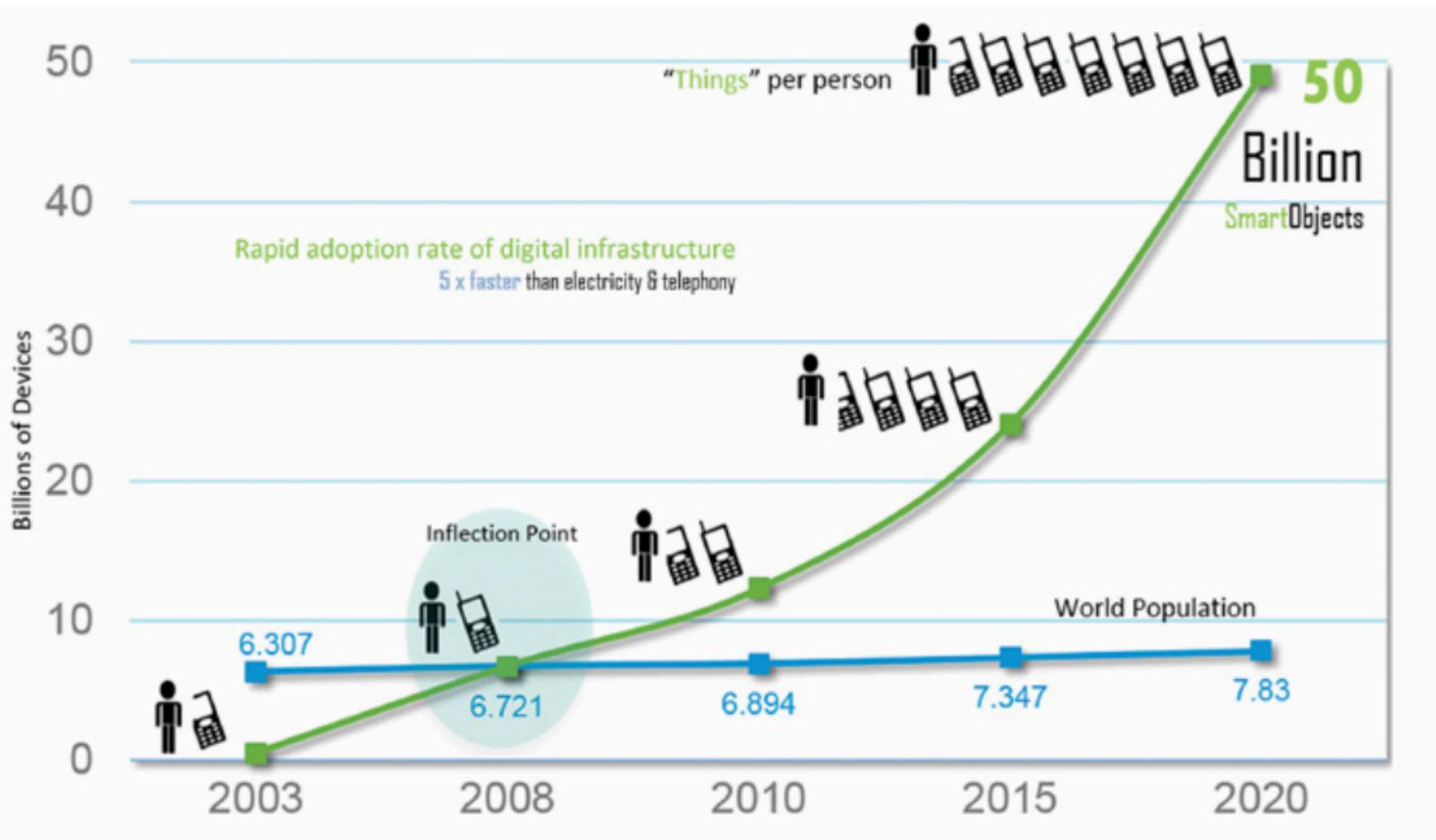
[*) Kevin Ashton. “The ‘Internet of Things’ Thing”. RFID Journal, 2009. <http://www.rfidjournal.com/articles/view?4986>]

[**] Samuel Greengard. *The Internet of Things*. MIT Press

IoT: UNA DEFINIZIONE

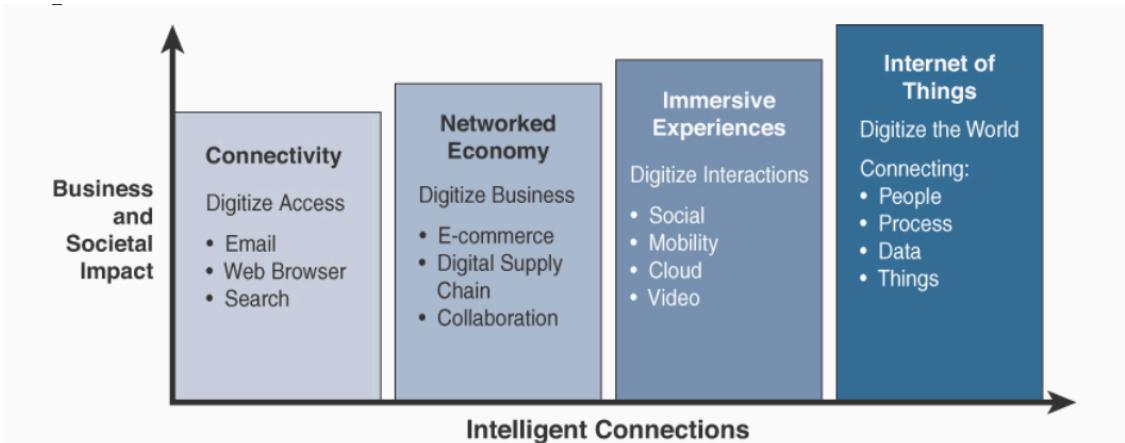
- Una definizione [LIT]:
 - *“The IoT is what we get when we connect Things, which are not operated by humans, to the Internet”*
 - sistemi costituiti da reti di oggetti fisici che interagiscono mediante la rete Internet
 - questi oggetti fisici sono tipicamente sistemi embedded
- IoT è alla base di **Industrial Internet**
 - infrastruttura che supporta la connettività ad ampia scala fra dispositivi e data
 - integrazione di sistemi embedded con sensori, software e sistemi di comunicazione
 - spesso chiamata *“Industry 4.0” o smart industry o smart manufacturing*

IoT: NUMERI



[IOTF, p.72]

EVOLUZIONE DI INTERNET



[IoTF, p.68]

Figure 1-1 Evolutionary Phases of the Internet

Internet Phase	Definition
Connectivity (Digitize access)	This phase connected people to email, web services, and search so that information is easily accessed.
Networked Economy (Digitize business)	This phase enabled e-commerce and supply chain enhancements along with collaborative engagement to drive increased efficiency in business processes.
Immersive Experiences (Digitize interactions)	This phase extended the Internet experience to encompass widespread video and social media while always being connected through mobility. More and more applications are moved into the cloud.
Internet of Things (Digitize the world)	This phase is adding connectivity to objects and machines in the world around us to enable new services and experiences. It is connecting the unconnected.

Table 1-1 Evolutionary Phases of the Internet

IoT - ELEMENTI PRINCIPALI

- “Things”
 - sistemi embedded
 - sensori, in particolare
- **Connettività e comunicazione**
 - studio di **protocolli di comunicazione** appositi
 - MQTT, XMPP, UPnP, CoAP
 - basati su Internet
 - aspetti relativi alla **interoperabilità**
 - vocabolari, ontologie
 - IoT come “*sistema di sistemi*” (System-of-Systems)
- Aspetti correlati a queste due dimensioni
 - sicurezza
 - identità, autenticazione, autorizzazioni...

SMART THINGS

- Visione semplificata [DIT]:

IoT =

Physical Objects +
Controller/Sensors/Actuators +
Internet

- ...dove le “cose” (physical object) possono essere gli oggetti più disparati che le persone utilizzano quotidianamente sia in ambito di lavoro, sia in ambito casalingo, ludico, etc.
- Visione ***enchanted objects*** - D. Rose
 - <http://tedxtalks.ted.com/video/TEDxBerkeley-David-Rose-Enchant>

ESEMPI DI SMART THINGS

- Fitbit force wristband
 - <http://www.fitbit.com/>
- Metromile
 - <https://www.metromile.com/>
- Kevo smart lock
 - <http://www.kwikset.com/kevo/>
- Google Nest thermostat & family
 - <https://nest.com/>
- WeMo light switch & family
 - <http://www.belkin.com/us/p/P-F7C030/>
- enchanted objects
 - <http://enchantedsobjects.com/products/>

RUOLO DEGLI SMARTPHONE

- Nel contesto di IoT, gli smartphone possono svolgere vari ruoli
 - fungere essi stessi da sistemi embedded,
 - dotati di opportuni sensori, per raccogliere e inviare dati geo-localizzati
 - fungere da “telecomando” universale
 - interfaccia utente (UI) unificata per interagire con altri smart things
 - fungere da unità di controllo collegata in rete che interagisce (tipicamente via bluetooth) con altri dispositivi wearable e non

IL RUOLO DELL'IDENTIFICAZIONE

- In origine, l'elemento caratterizzante di IoT era dato dalla tecnologia **RFID** (**Radio-Frequency Identification**)
 - tecnologia per l'identificazione e/o memorizzazione automatica di informazioni inerenti oggetti basata sulla capacità di memorizzazione di dati da parte di particolari etichette elettroniche, chiamate *tag* (o anche *transponder*), e sulla capacità di queste di rispondere all'interrogazione a distanza da parte di appositi apparati fissi o portatili, chiamati
 - tag passivi (non alimentati), attivi, semi-passivi
 - alcuni parametri di riferimento
 - distanza fino a 2 m (per i passivi), capacità di memorizzazione dell'ordine del kilobyte
 - recente evoluzione: **NFC** (Near-Field Communication)
 - piccole distanze (<10cm), maggiore velocità di trasmissione (424 kBit/s), interazione direttamente fra 2 lettori
- Oggi l'identificazione è solo un aspetto
 - “*when we talk about an Internet of Things, it is not just putting RFID tags on some dumb things so we smart people know where that dumb thing is. It's about embedding intelligence so things become smarter and do more than they were proposed to do*” (Nicholas Negroponte, MIT)

EVOLUZIONE DI IoT

- 5 stadi [Por14,Por15,DBE17]

1. product stage

- *the air conditioner*

2. smart product

- *the programmable air conditioner*

3. smart connect products

- *air conditioner accessible by the Internet*
 - controllable by the phone
 - the company can check its functioning and compare to other millions of other units to do *predictive maintenance*

...

EVOLUZIONE DI IoT

- 5 stadi [Por14,Por15,DBE17] (cont.)

1. product systems

- *the smart thermostat talks to the connected HVAC and the smart window blinds and heated floors*
- key points
 - **interoperability** & communication standards
 - » Web of Things (WoT) Initiative
 - **command and control platforms** availability
 - » Apple HomeKit, Amazon Echo, Google Home, Samsung SmartThings...
 - » industrial counter part: GE's Predix and Hitachi's Lumada

EVOLUZIONE DI IoT

- 5 stadi [Hep14,DBE17] (cont)

5. system of systems

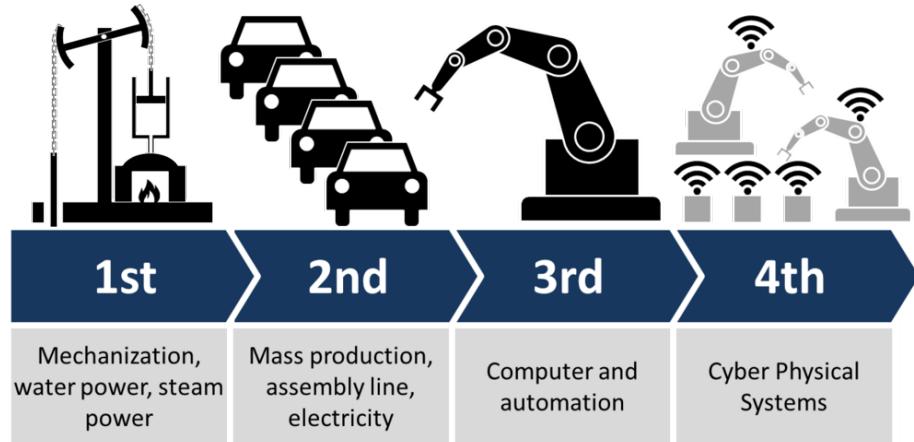
- home appliances talking with home security talking to the car and wearable devices talking to smart hospital...
- key points
 - interoperability
 - scalability, openness
 - governance

DA IOT

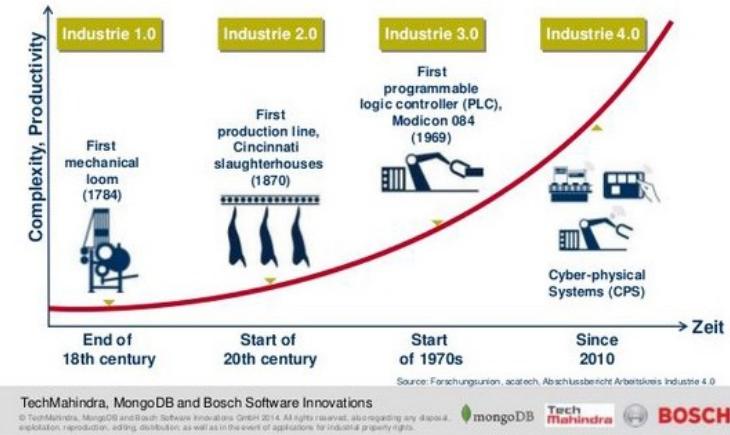
A ENTERPRISE E INDUSTRIAL IOT

- IoT come tecnologia chiave per industria e imprese
 - **Industry 4.0**
 - quarta rivoluzione industriale
- Aspetto chiave: “Things” come mezzo per raccogliere dati mediante sensoristica opportuna
 - *big-data, real-time, big-stream*
- Enterprise IoT vs Industrial IoT (I-IoT)
 - **Enterprise IoT** => termine generale per indicare l’uso di IoT a livello enterprise
 - **Industrial IoT (I-IoT)** => caratterizzazione di Enterprise IoT specificatamente nel contesto industriale / manifatturiero in particolare

INDUSTRY 4.0



Industrie 4.0: The next Industrial Revolution



<http://www.eesc.europa.eu/?i=portal.en.group-1-new-news.34501>

LE QUATTRO RIVOLUZIONI INDUSTRIALI

Industry 4.0: IoT Integration (*Today*)

Sensors with a new level of interconnectivity are integrated

Industry 3.0: Electronics and Control (*Early 1970's*)

Production is automated further by electronics and IT

Industry 2.0: Mass Production (*Early 20th Century*)

Division of labor and electricity lead to mass production facilities

Industry 1.0: Mechanical Assistance (*Late 18th Century*)

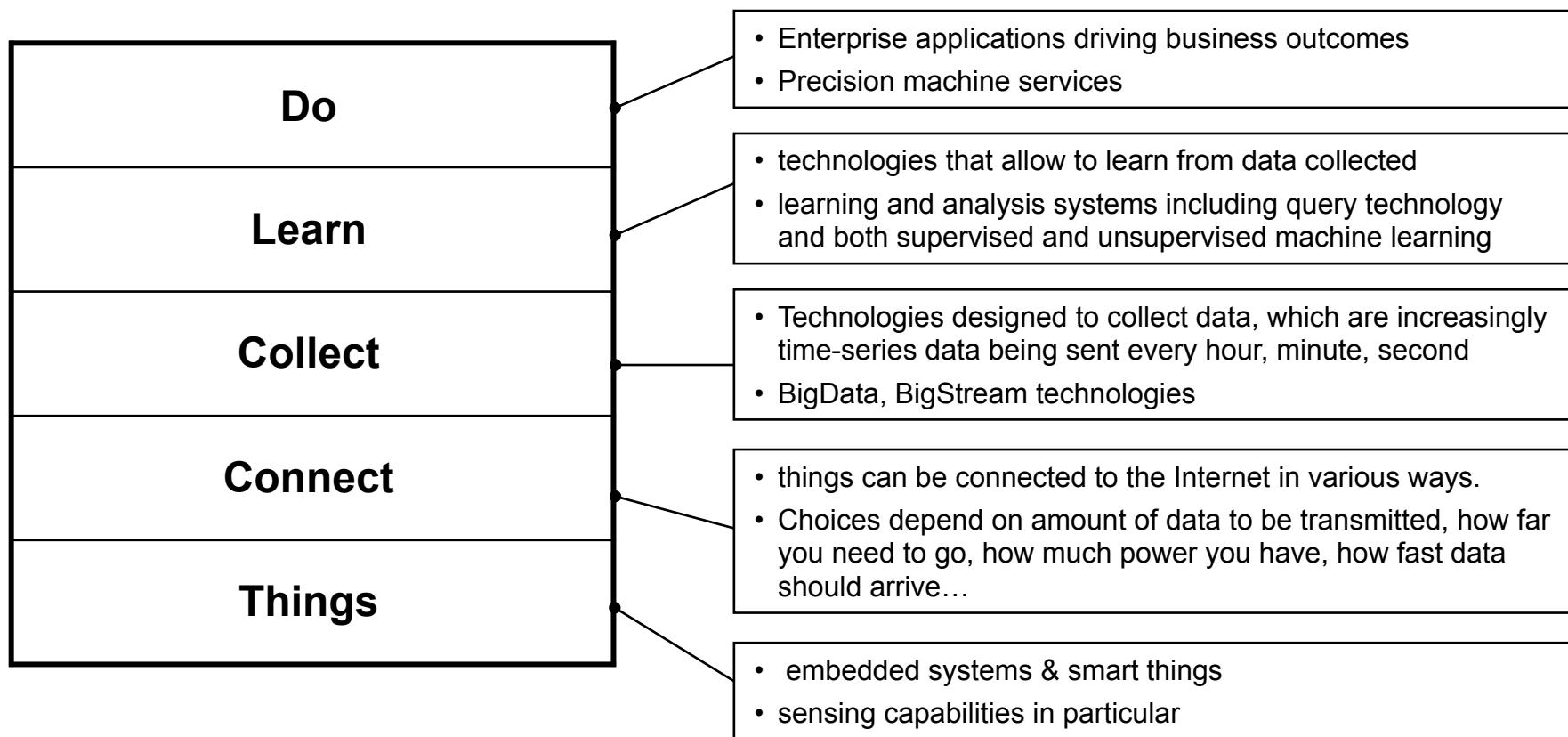
Basic machines powered by water and steam are part of production facilities

[IoTF, 85]

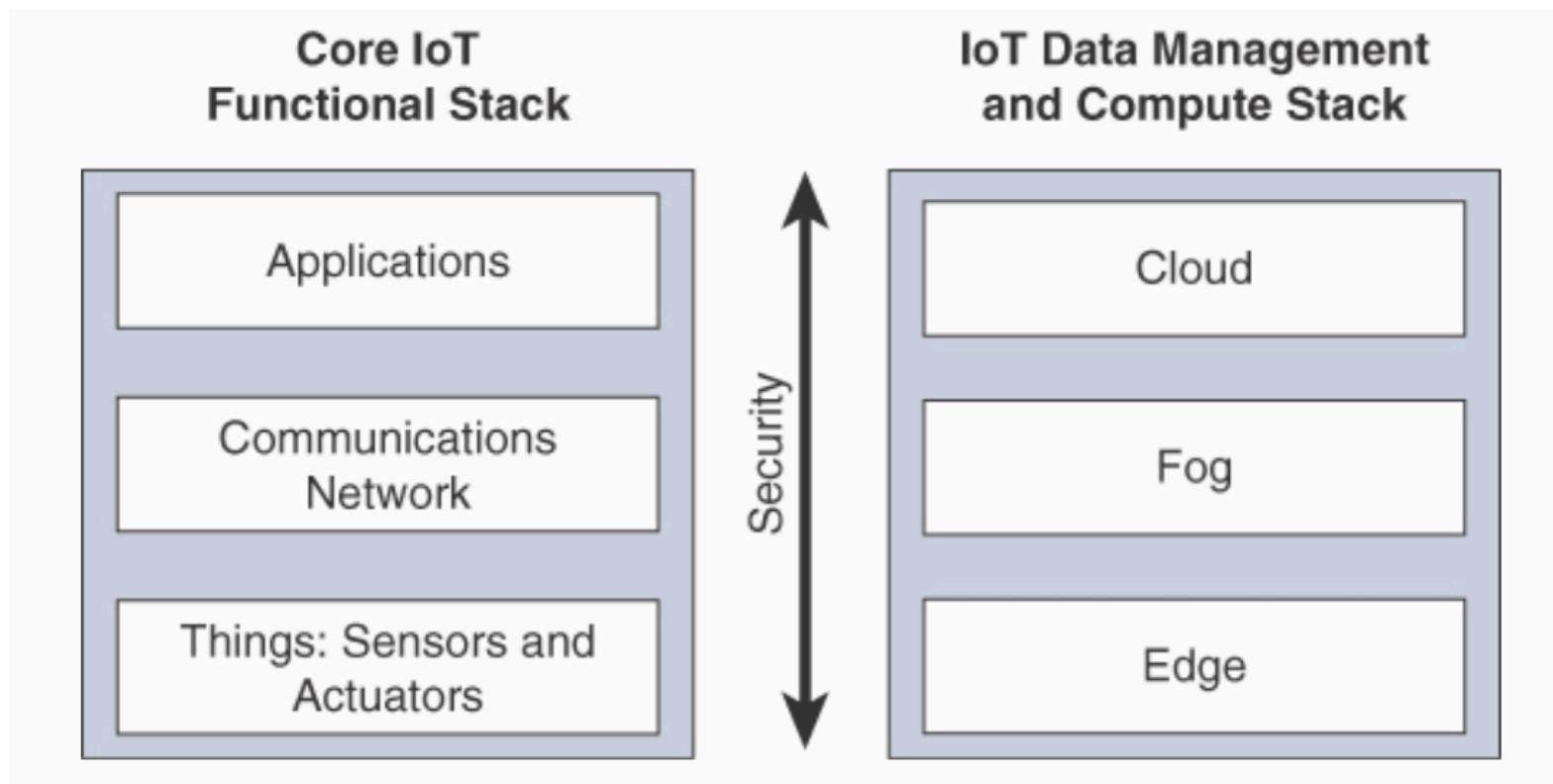
ENTERPRISE IOT

- [Framework concettuale per identificare i vari livelli che riguardano IoT a livello Enterprise (*)]

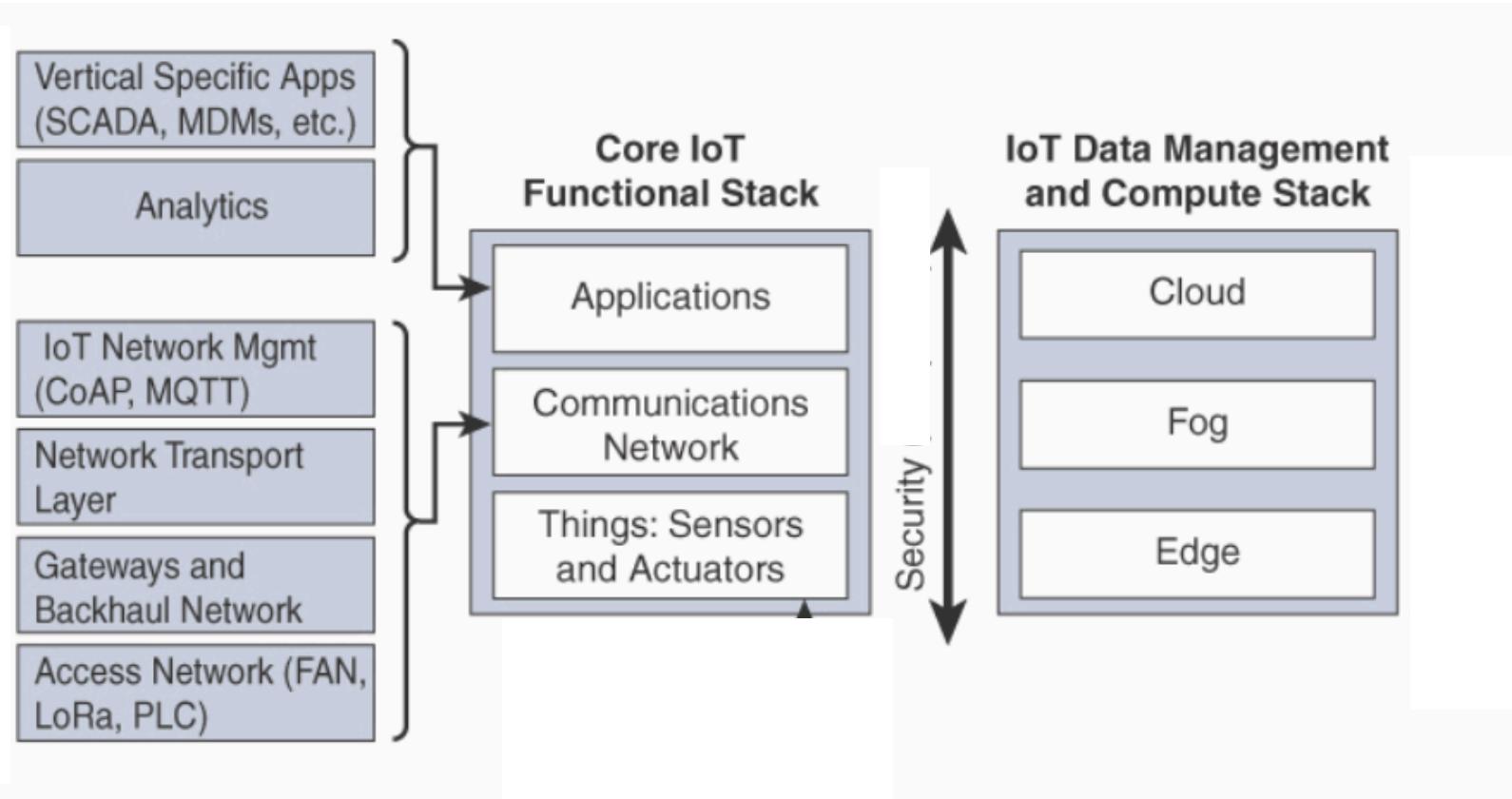
“Precision-XXX”



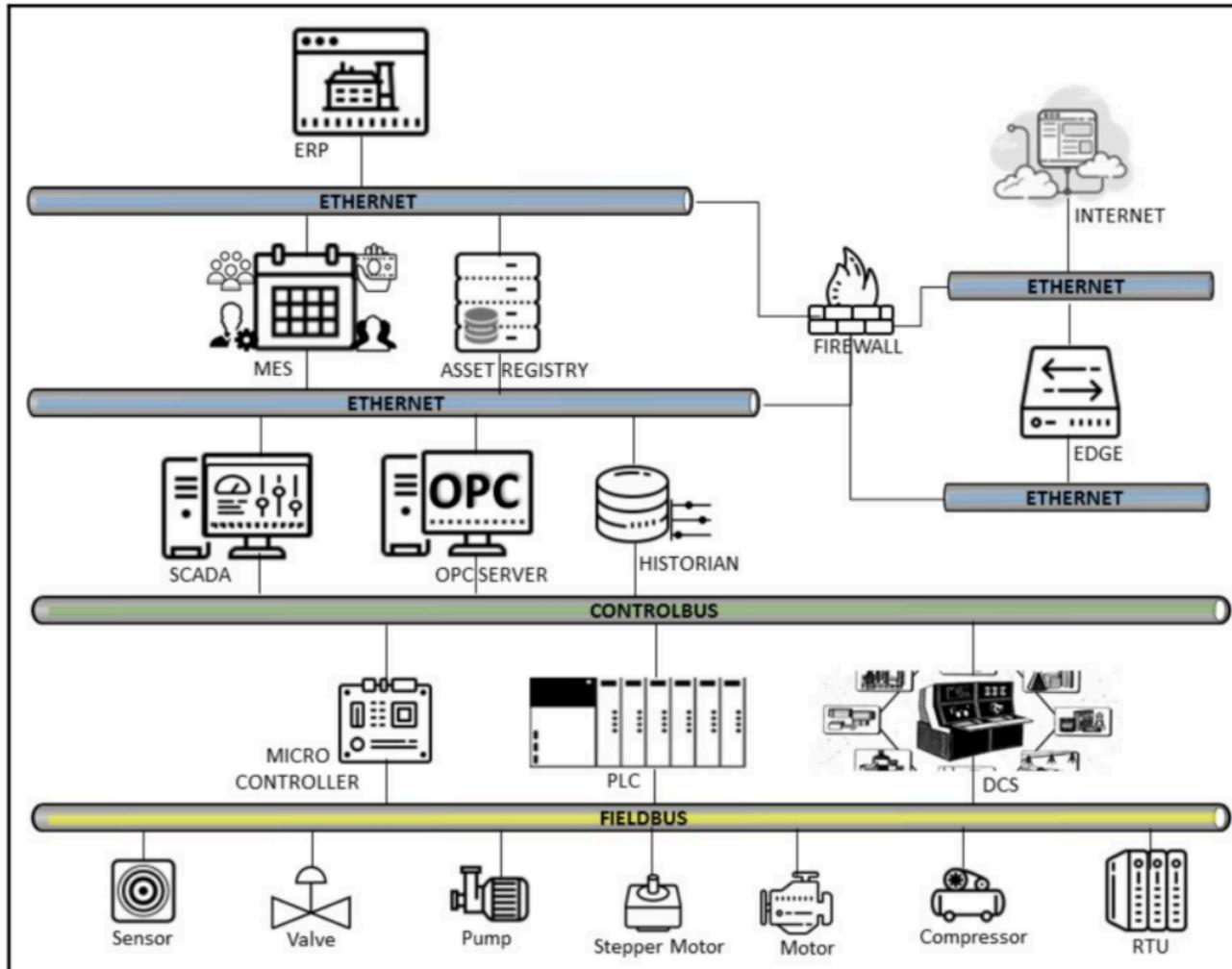
ARCHITETTURA SEMPLIFICATA DI UN SISTEMA ENTERPRISE IOT



ARCHITETTURA SEMPLIFICATA DI UN SISTEMA ENTERPRISE IOT



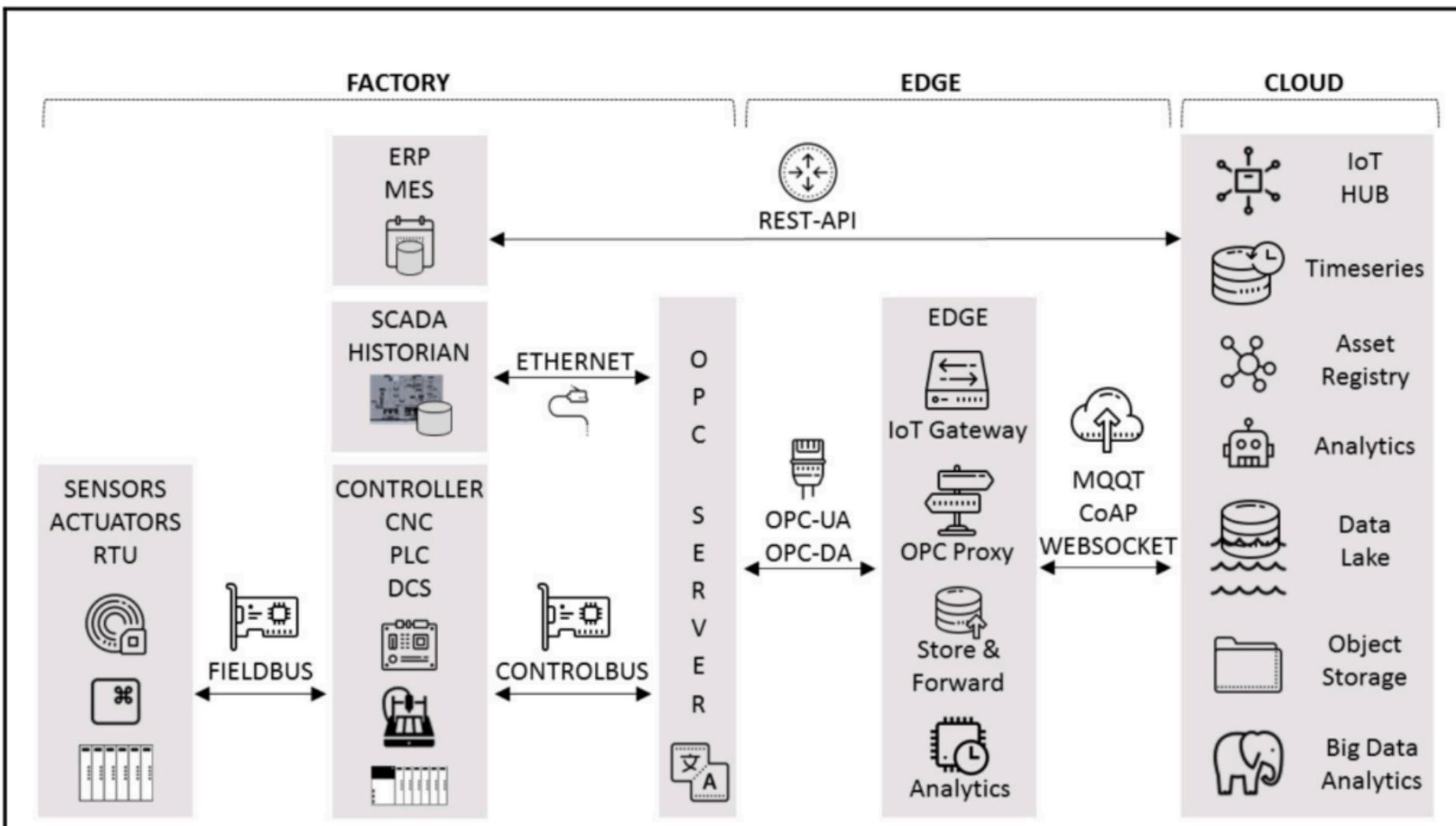
I-IOT DISPOSITIVI E PROTOCOLLI - QUADRO



[VC18]

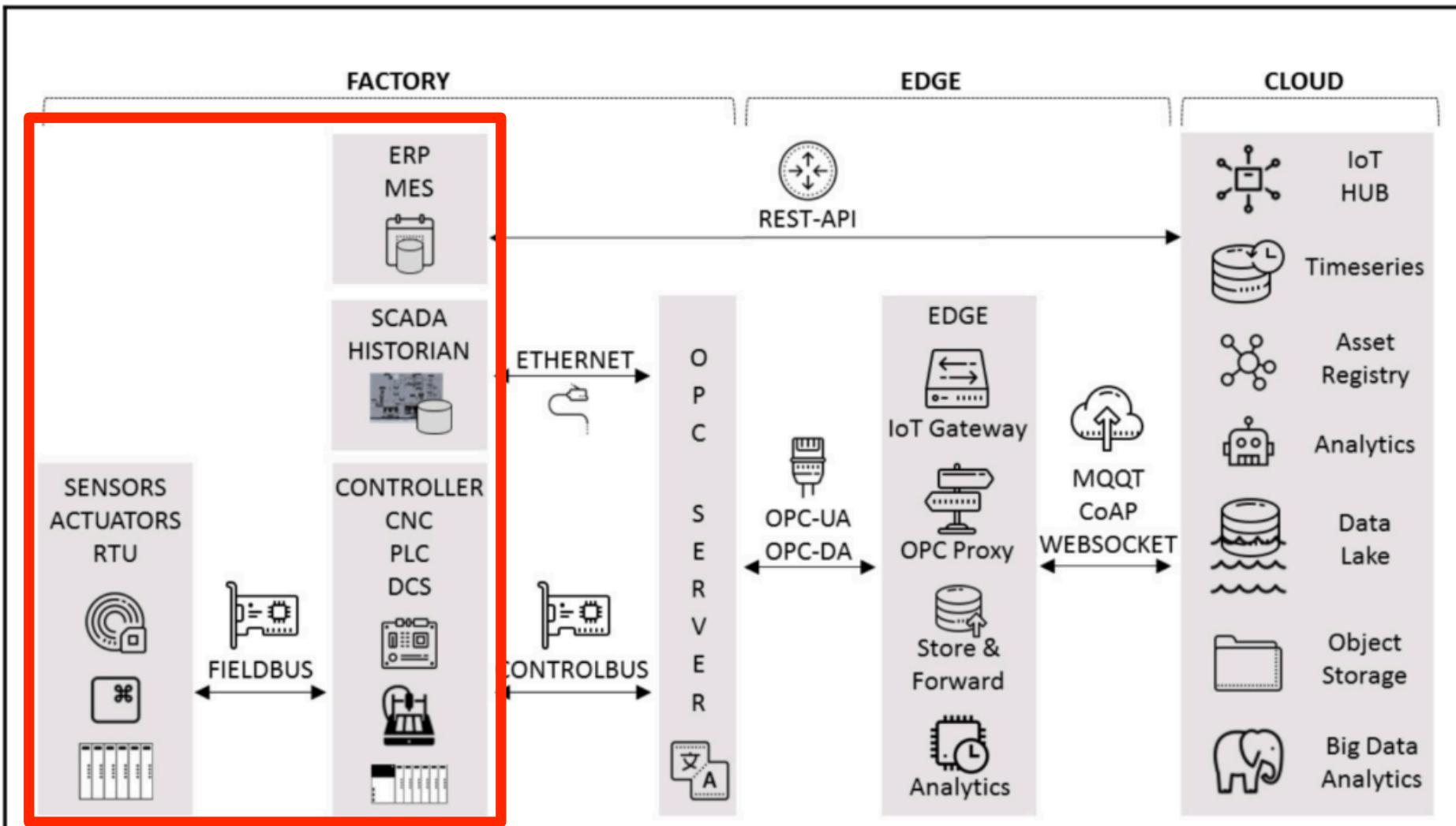
I-IOT DATA FLOW - QUADRO

- Dalla generazione all'elaborazione sul cloud



I-IOT DATA FLOW - QUADRO

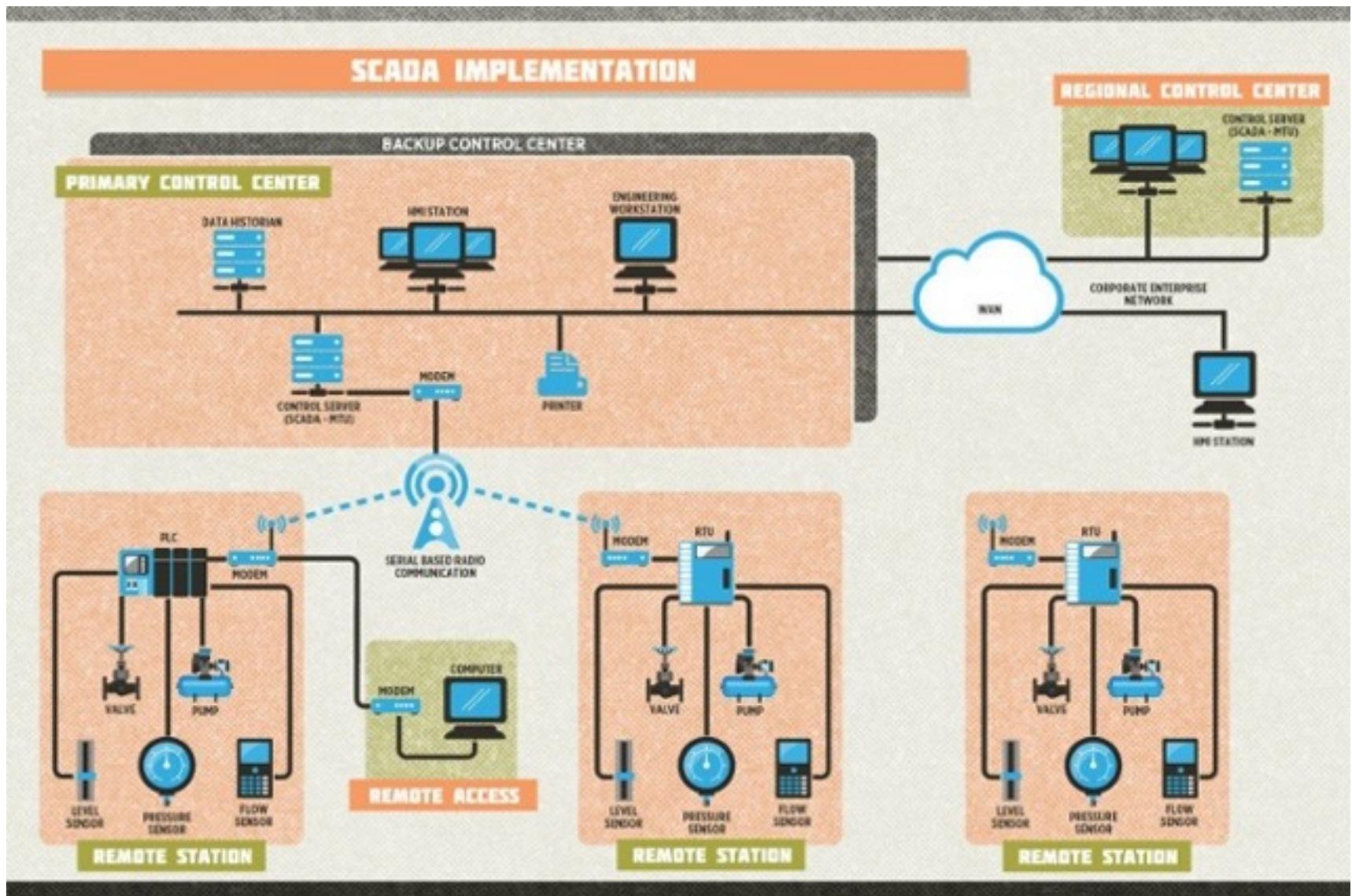
- Dalla generazione all'elaborazione sul cloud



SISTEMI SCADA

- In ambito industriale, un tipo di sistema M2M molto diffuso è dato dai sistemi **SCADA** (Supervisory Control and Data Acquisition)
 - *sistemi di supervisione* che operano con codifiche di segnali trasmesse per canali di comunicazione al fine di realizzare un **controllo remoto di sistemi/equipaggiamenti/impianti**
 - il sistema di supervisione può essere combinato con sistemi di acquisizione dei dati attraverso l'aggiunta di ulteriori codifiche di segnali e comunicazioni al fine di acquisire informazioni circa lo stato dei dispositivi remoti per poterle visualizzare o registrare
- E' un tipo di **sistema di controllo industriale (ICS)**
 - sistemi computerizzati che *monitorano e controllano i processi industriali* in un certo ambiente fisico
 - i sistemi SCADA si distinguono storicamente da altri ICS per scala, ovvero tipicamente riguardando processi a larga scala che includono molteplici siti, distribuiti in un territorio che può essere geograficamente molto vasto

SISTEMI SCADA - ARCHITETTURA



<http://blog.cimation.com/blog/key-differences-between-scada-dcs-and-hmi-systems>

SCADA: PRINCIPALI ELEMENTI

- **Sensori e Remote terminal units (RTUs)**
 - alla base di uno SCADA c'è la rete di sensori
 - permette di monitorare il funzionamento e lo stato dei macchinari e dei processi industriali
 - collegati ai sensori troviamo le remote terminal unit o RTU (“Unità terminali remote”)
 - compito di raccogliere i dati rilevati dai sensori e trasformarli in segnali digitali da inviare alla postazione di controllo remota

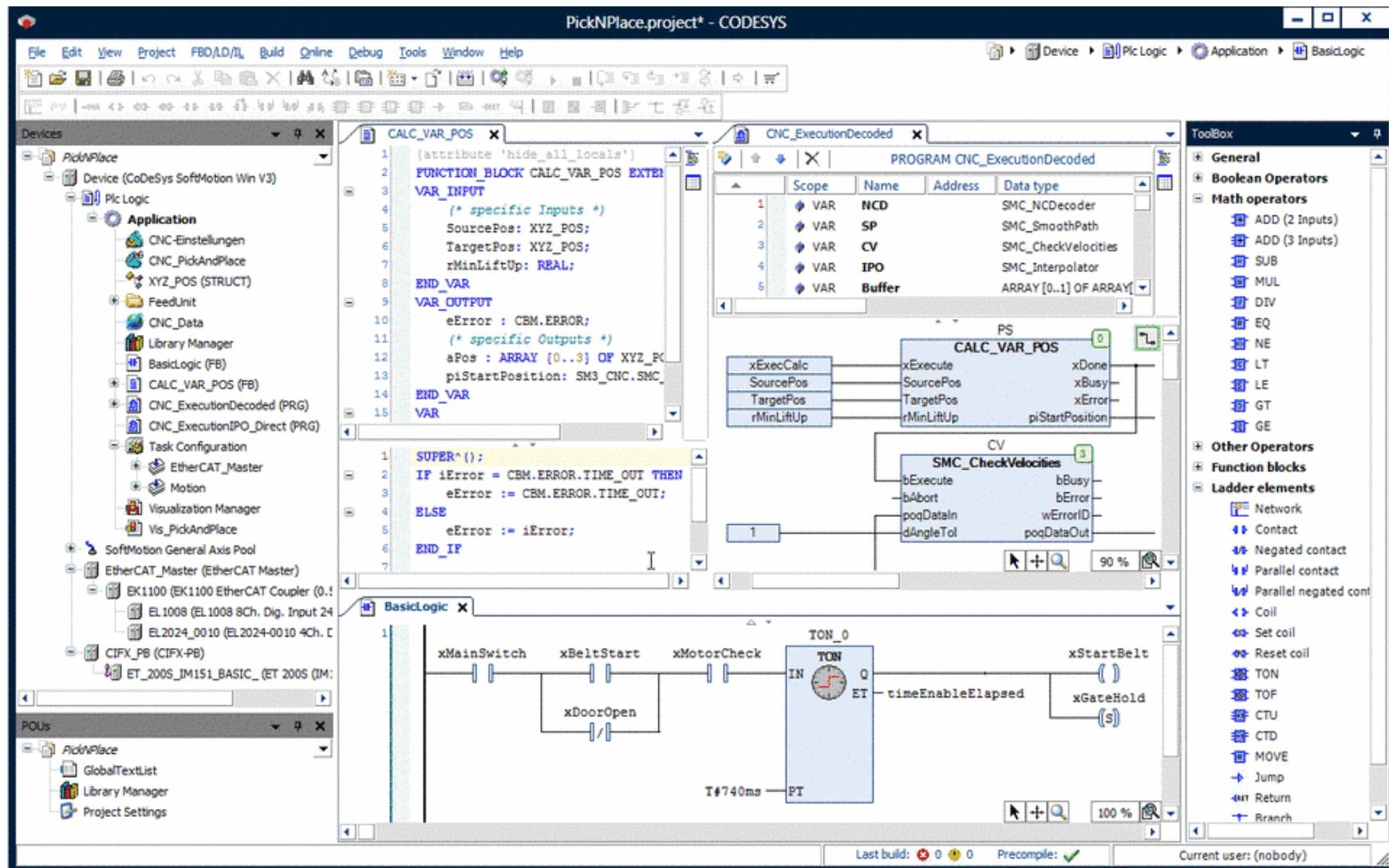
SCADA: PRINCIPALI ELEMENTI

- **Programmable logic controller (PLCs)**
 - alla rete di sensori ed RTU è collegato uno o più programmable logic controller o PLC (“Controller logico programmabile”)
 - compito di supervisionare la raccolta di informazioni, fornendo istruzioni sia alle RTU sia direttamente ai sensori
 - è il PLC a indicare alla rete sensoriale quali sono gli intervalli di tempo nei quali effettuare le misurazioni e controllare i valori dei macchinari
 - programmabili con linguaggi di programmazione della famiglia IEC 61131-3

PROGRAMMAZIONE DI PLC CON IEC 61131-3

- IEC 61131-3 è la terza parte (di 8) dello standard internazionale aperto IEC 61131 per PLC (programmable logic controllers).
- Questa parte concerne i linguaggi di programmazione e definisce un insieme di linguaggi di programmazione standard per PLC:
 - **Ladder diagram (LD)**, visuale
 - **Function block diagram (FBD)**, visuale
 - **Structured text (ST)**, testuale
 - **Instruction list (IL)**, testuale
 - **Sequential function chart (SFC)**
 - possiede elementi per organizzare i programmi di controllo sia sequenziali che paralleli.

PROGRAMMAZIONE DI PLC CON IEC 61131-3



SCADA: GLI ALTRI ELEMENTI

- Un sistema di **telemetria**, che connette PLC e RTU con centri di controllo, data warehouse, e sistemi enterprise
 - può essere una rete informatica come una LAN o una WAN, o più semplicemente da una rete di linee seriali, sia wired che wifi
- **Computer supervisore** (MTU, Master Terminal Unit)
 - server di controllo che raccoglie periodicamente i dati dai PLC, li elabora per ottenerne informazioni utili, memorizza le informazioni, invia comandi di controllo
- Un **interfaccia uomo-macchina** (HMI, human machine interface)
 - permette ad operatori umani di accedere ai dati processati, monitorarli e interagirvi
- Un **historian**
 - servizio software che memorizza time-stamped data, eventi e allarmi, in un data base che viene poi interrogato al fine di creare report grafici nel HMI

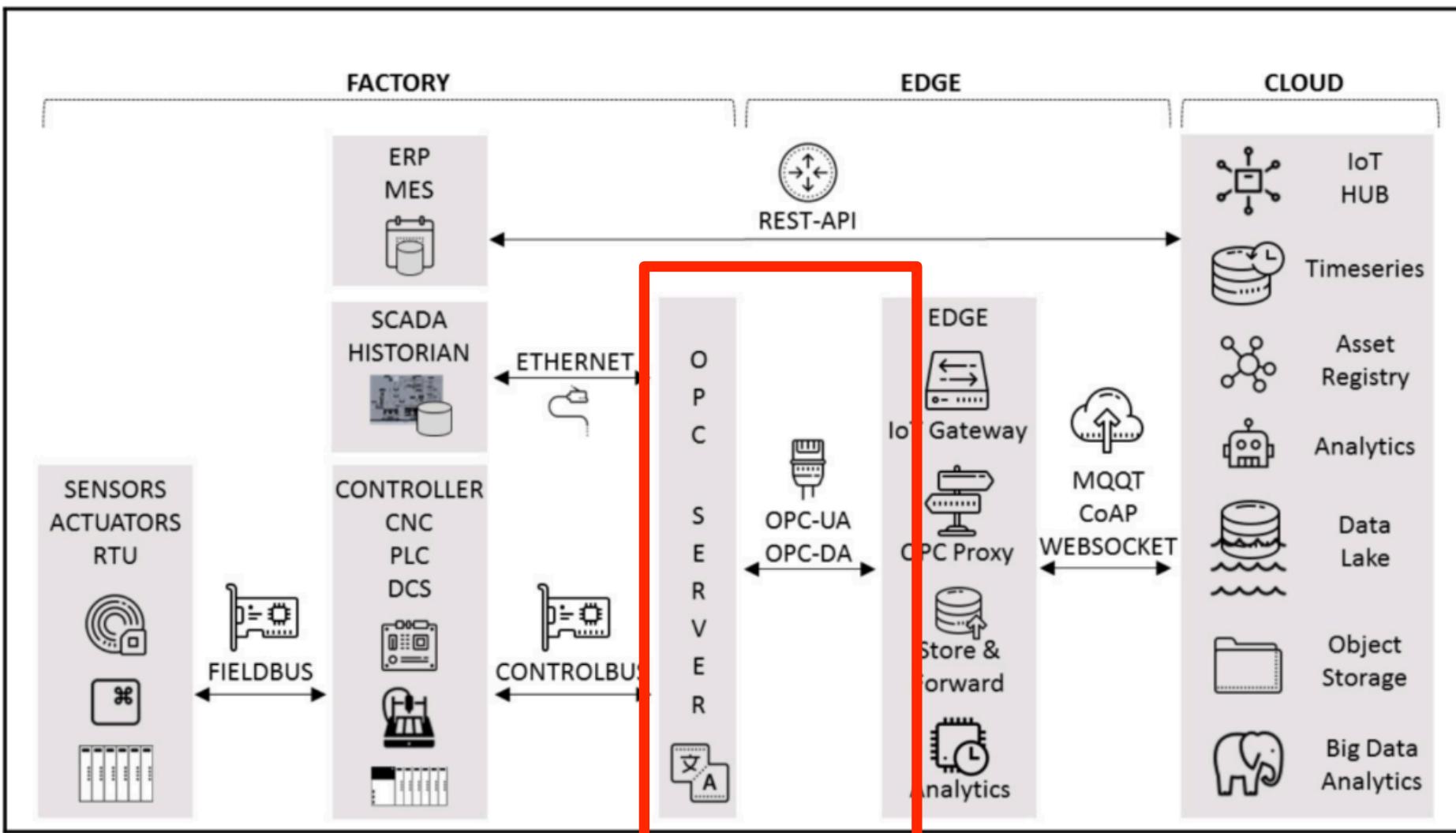
SCADA HMI - ESEMPIO



<http://www.armani-engr.com/hmiscada.html>

I-IOT DATA FLOW

- From its generation at the field level to its processing in the cloud



THE OPC STANDARD

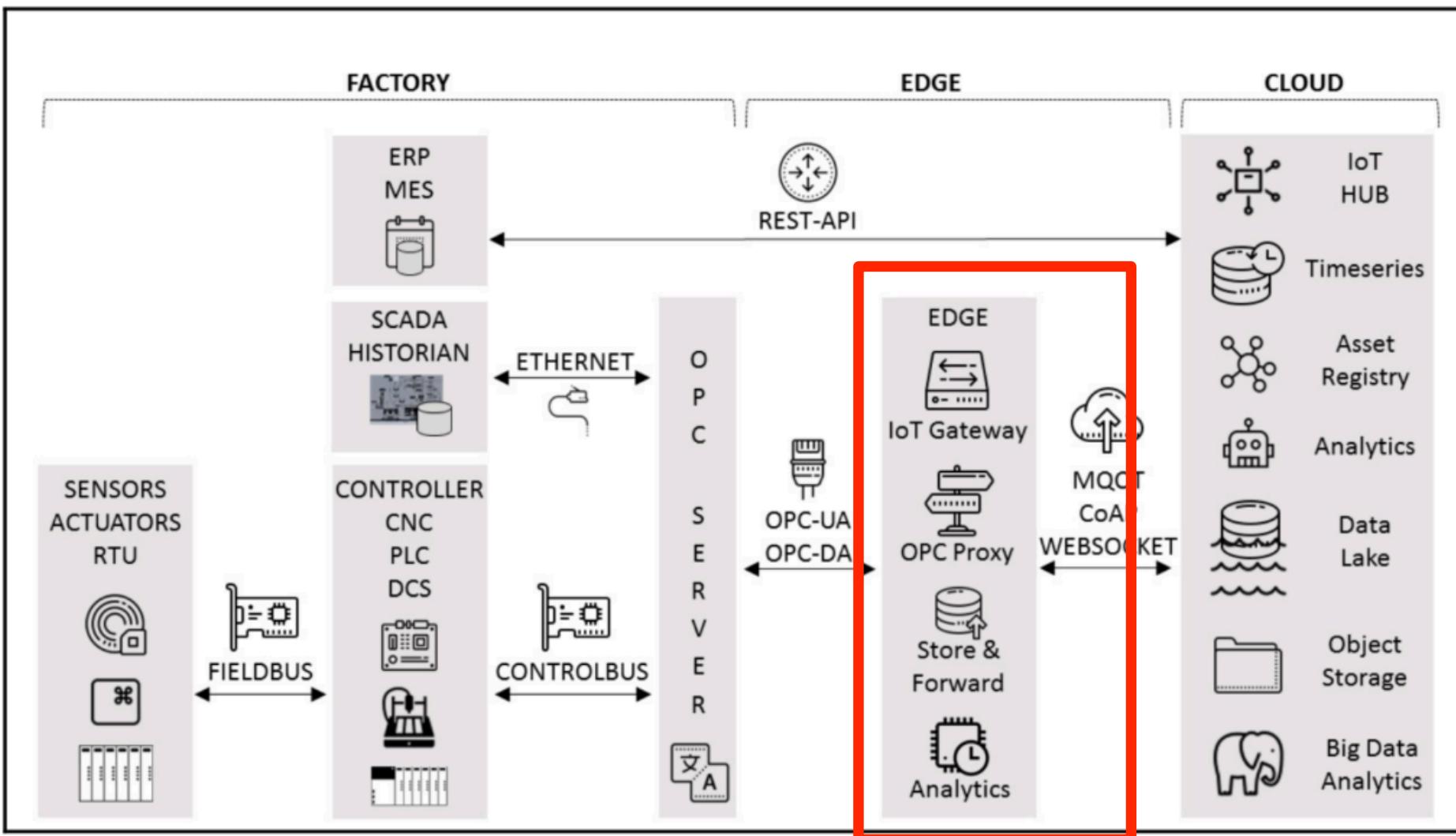
- **OPC** is the acronym of **Open Platform Communication**
 - software layer that makes it possible to interact with the industrial data source through its own protocol for querying tags and time-series, and exposing them by means of a *standard interface* to the upper levels and external systems
 - acting as a *translator* between the controllers and producers of the industrial data and the consumer systems
 - used to abstract PLC or DCSes specific protocols (e.g. Modbus and Profibus) into a standardized interface
 - function as a middle-man for SCADA & co converting generic OPC read & write requests into device-specific requests and vice-versa
- Through OPC, PLCs or DCSes *can expose tags, time-series, alarms, events and Sequence of Events (SOE) to any system that implement the OPC interface*
 - SCADA and Historian where initially developed as consumers of industrial data, implementing an *OPC client* interface, but themselves become producers of industrial data, implementing an *OPC server* interface

OPC SPECIFICATIONS

- Two main specifications
 - **OPC Classic** (old, ~1995)
 - based on Windows OS and OLE / DCOM
 - **OPC-UA** (~2005)
 - based on service-oriented approach as defined by IEC 62451
 - not only for Windows, to be embeddable also in small devices

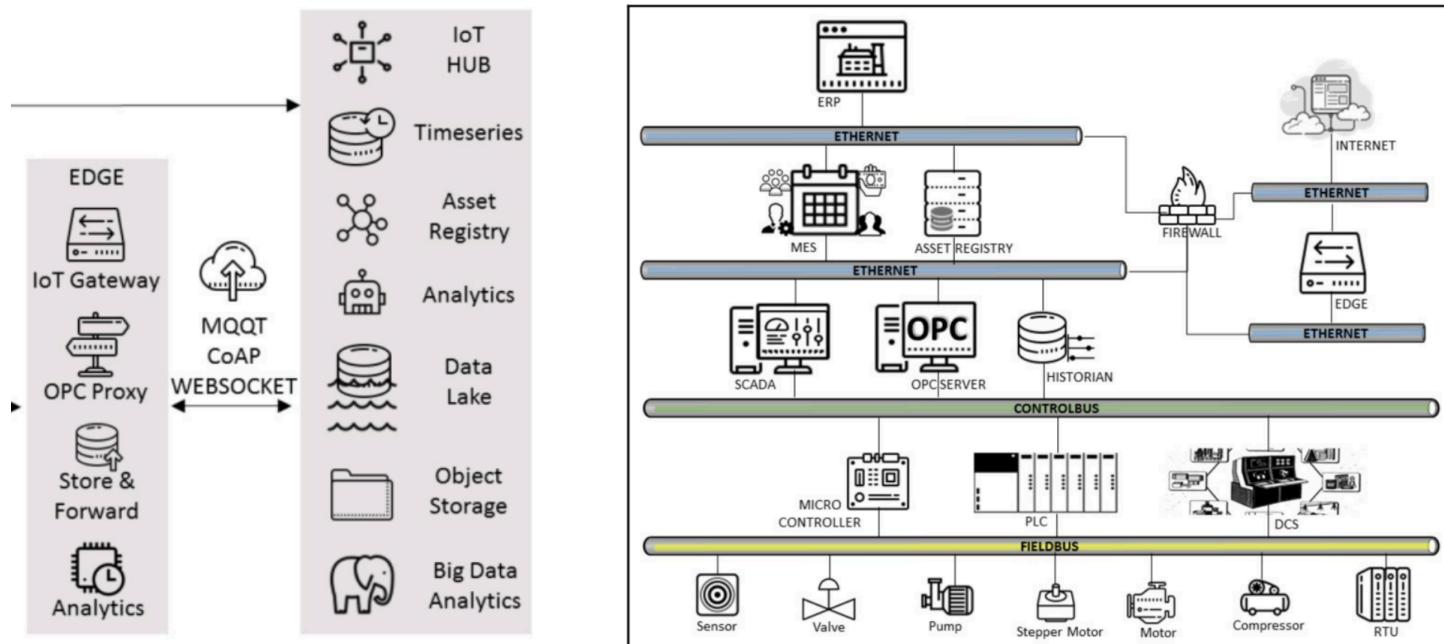
I-IOT DATA FLOW - QUADRO

- From its generation at the field level to its processing in the cloud



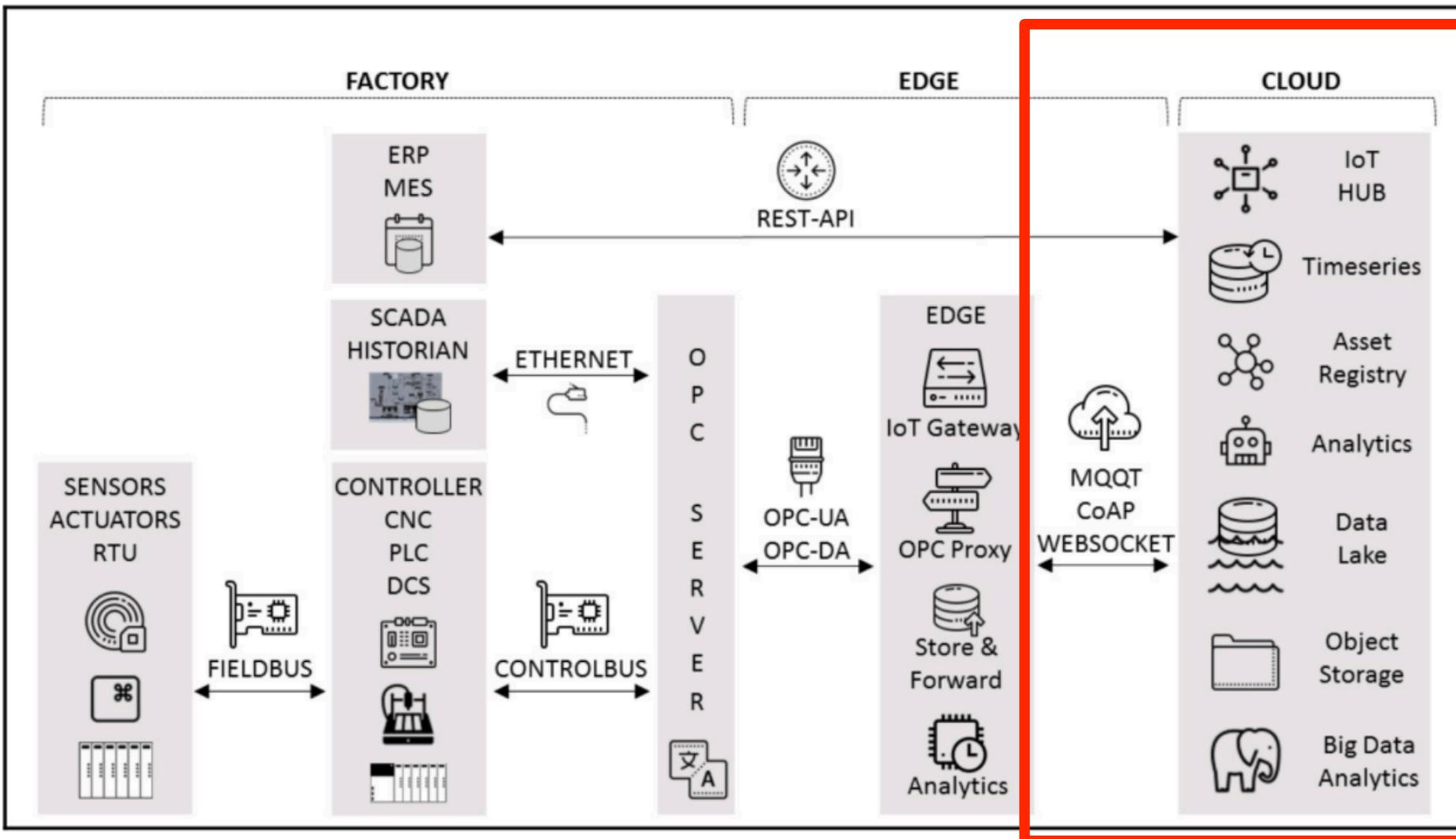
THE EDGE DEVICE

- Device which enables the IIOT
 - physically located in the factory
 - linked from one side to the industrial data sources through the OPC server, and from the other side to the cloud through the IoT Gateway
- Different kinds of approaches in industries about arranging and conceiving the edge device



I-IOT DATA FLOW - QUADRO

- Dalla generazione all'elaborazione sul cloud



IOT E CLOUD

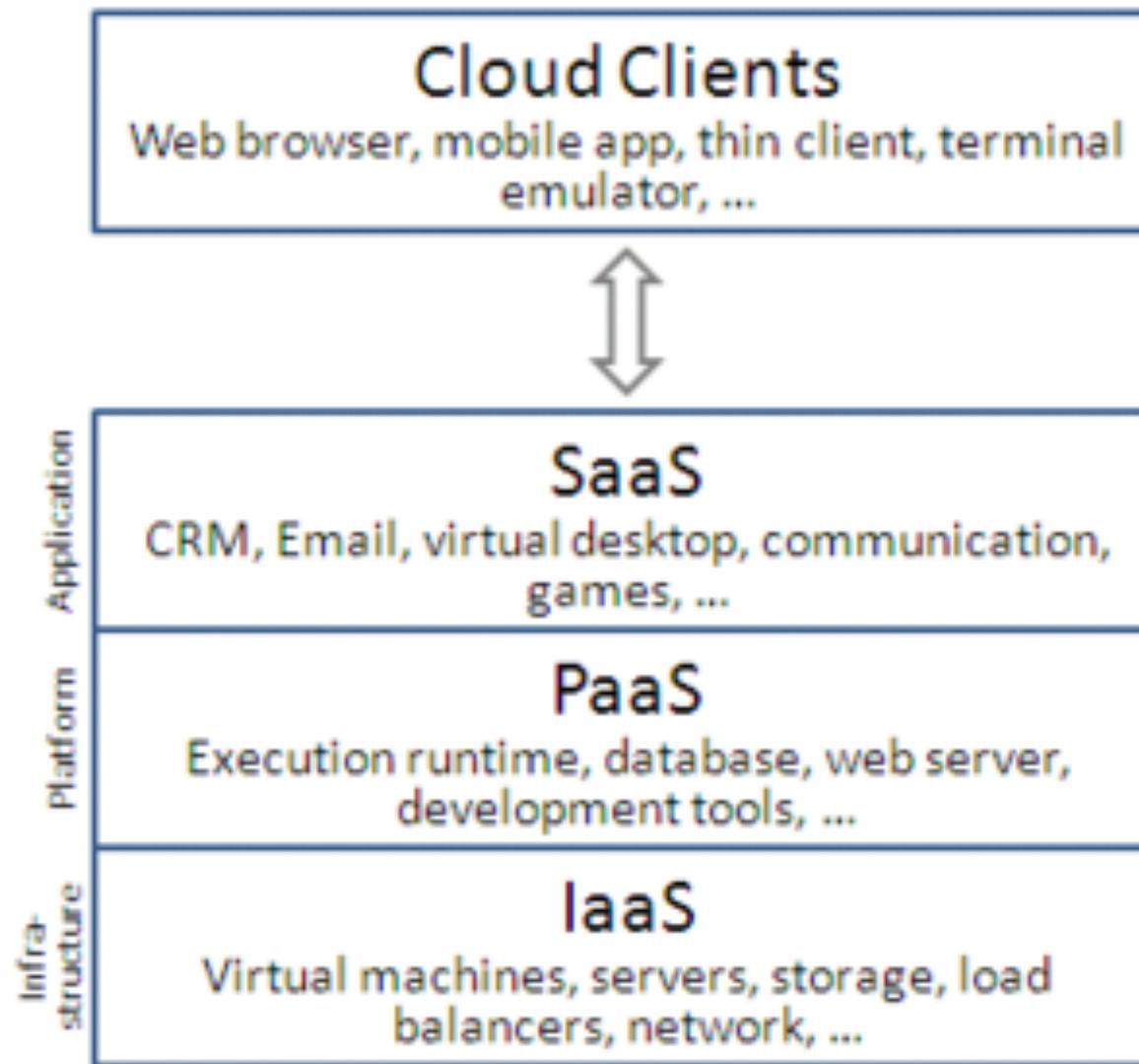
- Caratteristica fondamentale IoT: *capacità di comunicare direttamente o indirettamente con la rete Internet*
- Ciò permette di gestire in modo opportuno grandi volumi e stream di dati generati dai sensori
 - che non potrebbero essere memorizzati in locale
- Tali informazioni diventano quindi accessibili mediante opportuni servizi Internet o Web, che ne permettono lo scambio con altre applicazioni, in particolare con applicazioni mobile
- Ruolo importante del **cloud**
 - servizi a disposizione per gestione dispositivi, storage dati, analisi offline e online, ...
 - sistemi aperti

IL CLOUD

- “Cloud computing”
 - *“paradigma di erogazione di risorse informatiche, come l'archiviazione, l'elaborazione o la trasmissione di dati, caratterizzato dalla disponibilità on demand attraverso Internet a partire da un insieme di risorse preesistenti e configurabili” (*)*
- le risorse vengono date come-servizio (**as-a-service**) in rete e possono essere a tre livelli diversi
 - **IAAS** - infrastructure as a service
 - macchine virtuali, server, storage, network
 - **PAAS** - platform as a service
 - execution runtime, database, web servers, IDEs,...
 - **SAAS** - software as a service
 - CRM, giochi, office apps, mail apps, etc.
- Lato client (desktop, mobile, wearable, embedded..)
 - browser moderni - HTML5 e JavaScript
 - qualsiasi app che usi HTTP API

(*) Peter Mell, Timothy Grance, The NIST Definition of Cloud Computing. NIST, Special Publication 800-145, Settembre 2011.

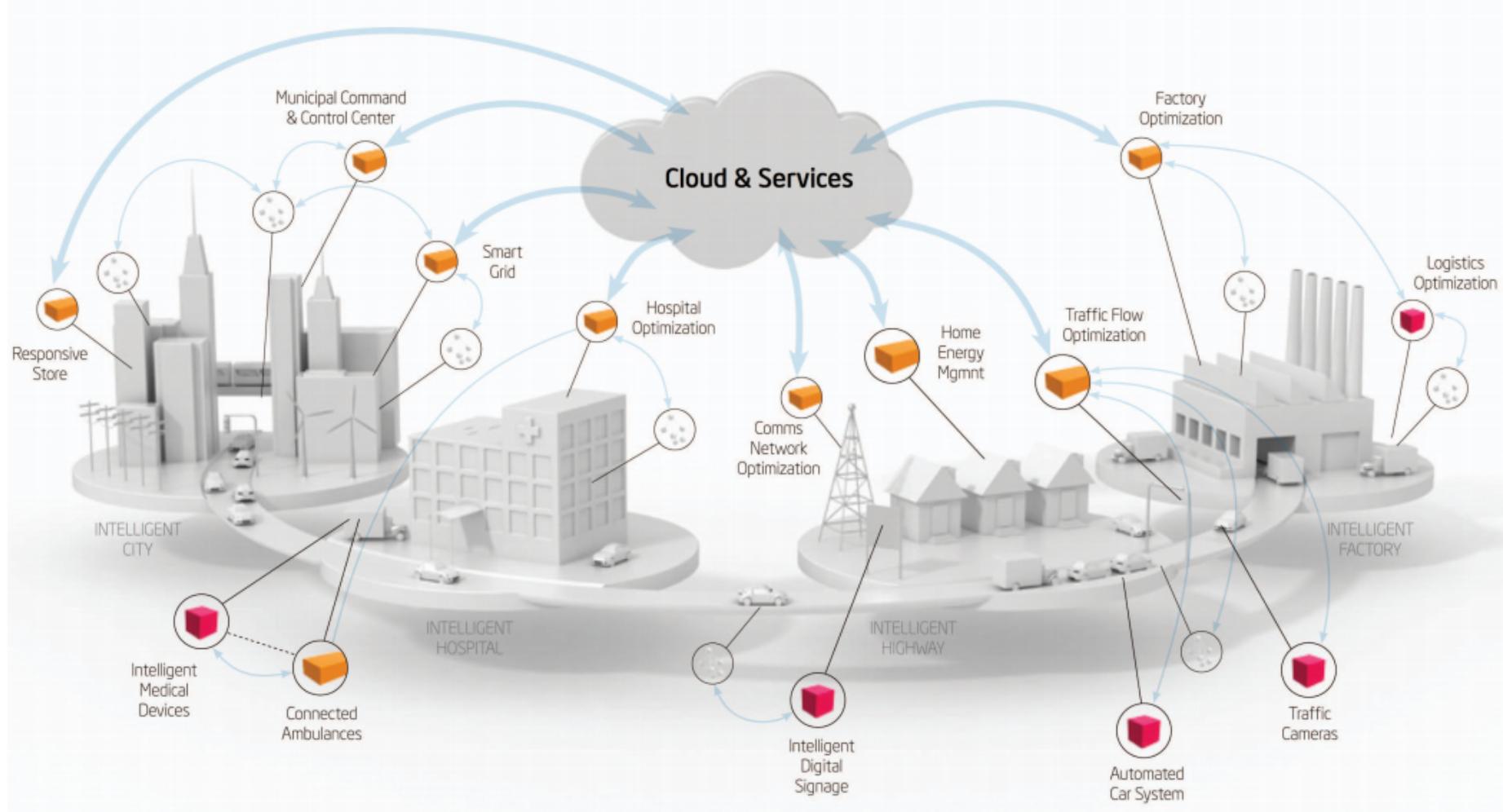
LIVELLI CLOUD



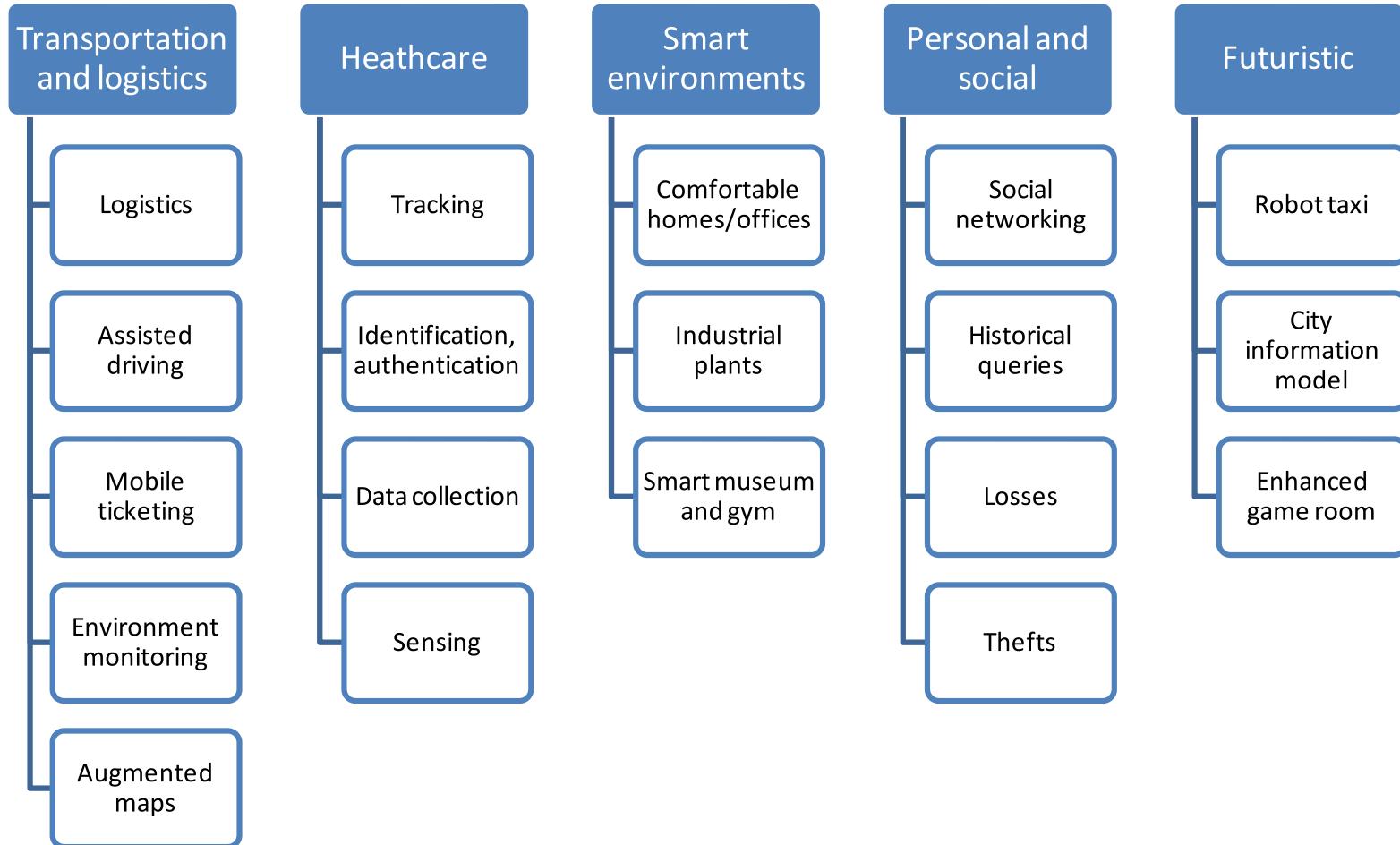
IL CLOUD E IOT

- Nel caso di IoT, i servizi cloud vengono primariamente utilizzati per
 - raccolta dati inviati dai dispositivi
 - stream
 - accesso ai dati inviati dai dispositivi
 - pull, event-driven
 - analisi offline e online
 - gestione “aperta” dei dispositivi
- **Big Data**
 - il volume e la velocità dei dati generati può essere tale da richiedere tecniche di memorizzazione, elaborazione e accesso che vanno al di là di quelle fornite dai tradizionali data-base
 - queste tecniche e architetture sono esplorate nell’ambito “big data”

IOT E IL CLOUD

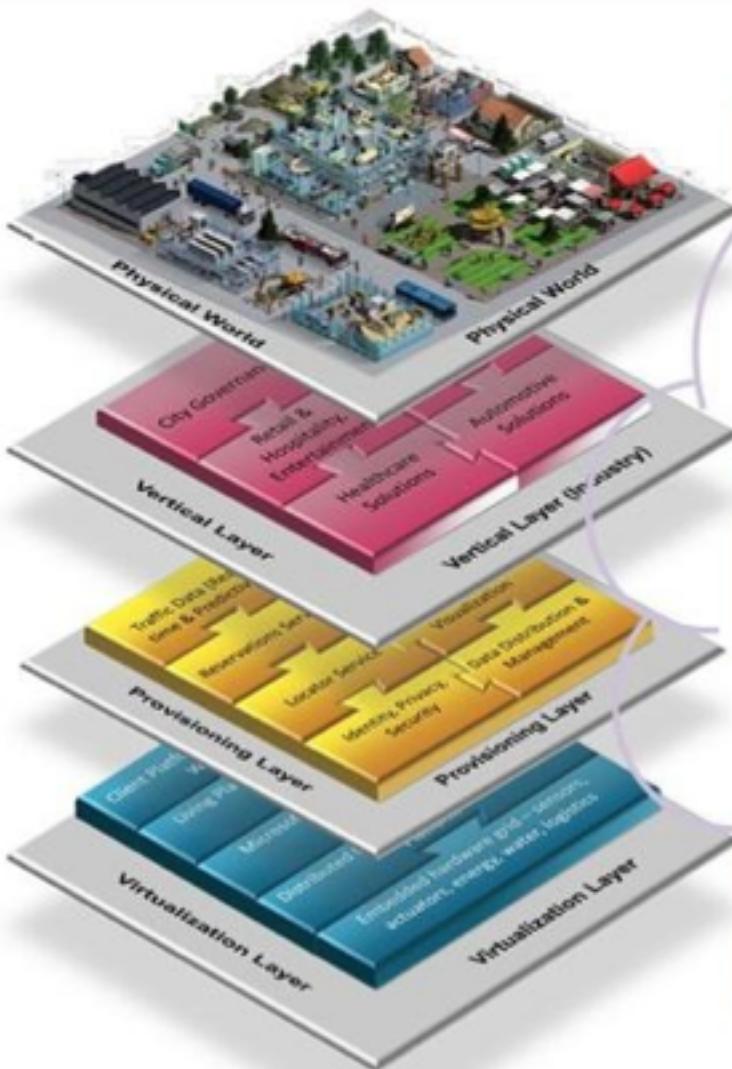


PRINCIPALI AMBITI APPLICATIVI



Tratto da: [Luigi Atzori, Antonio Iera, Giacomo Morabito.
The Internet of Things: A survey. Computer Networks, 54 (2010)]

SMART CITY, URBAN OS & APPS



(Urban OS, Plan IT)

Examples: Future Scenarios

Weather

Building and vehicle sensors provide microclimate information enabling improved models; transportation and HVAC systems take current local weather into account

Emergency Services

Improved emergency planning and response using real time information and signaling

Sustainability & Utilities

Smart grid, water efficiency, energy demand and supply management, shaping, and remote control

Transportation

Congestion avoidance, green routes, integrated systems of buses, trains, taxis, cars (with parking/charging), bikes, walking, and PRT

Vehicles

Onboard energy generation and/or storage

IOT: ASPETTI CRITICI E SFIDE

- Sicurezza
- Privacy
- Problema della proprietà dei dati
- Scalabilità e *Interoperabilità*
 - definizione di standard e architetture di riferimento
 - fra le iniziative significative: **Web of Things** (WoT)

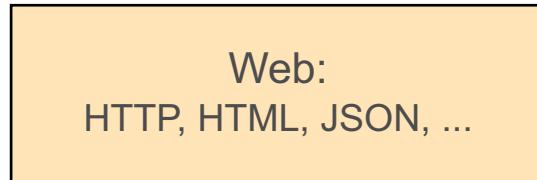
IoT + WEB = *WEB-OF-THINGS*

- **Web of Things** (o **WoT**) consiste in sistemi che incorporano gli oggetti fisici di uso quotidiano nel **World Wide Web** dando loro una REST-ful API
 - questo facilita la creazione di profili virtuali degli oggetti e la loro integrazione e riuso in tipi diversi di applicazioni
- E' una evoluzione di IoT dove il punto fondamentale concerne come gli oggetti comunicano mediante opportuno livello di rete
 - Zigbee, Bluetooth, 6LoWPAN...

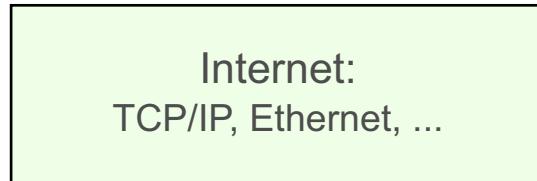
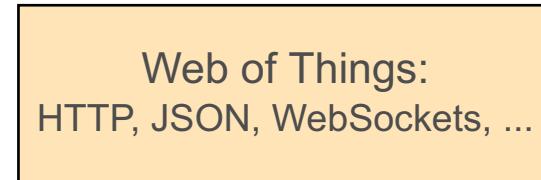
Guinard, Dominique; Vlad Trifa; Erik Wilde (2010). "A Resource Oriented Architecture for the Web of Things". Proc. of IoT 2010 (IEEE International Conference on the Internet of Things). Tokyo, Japan.

WoT = IoT + APPLICATION LEVEL

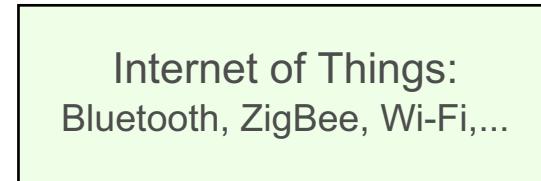
Easier to program, faster to integrate data and services, simpler to prototype, deploy, and maintain large systems.



Application level
(OSI layer 7)

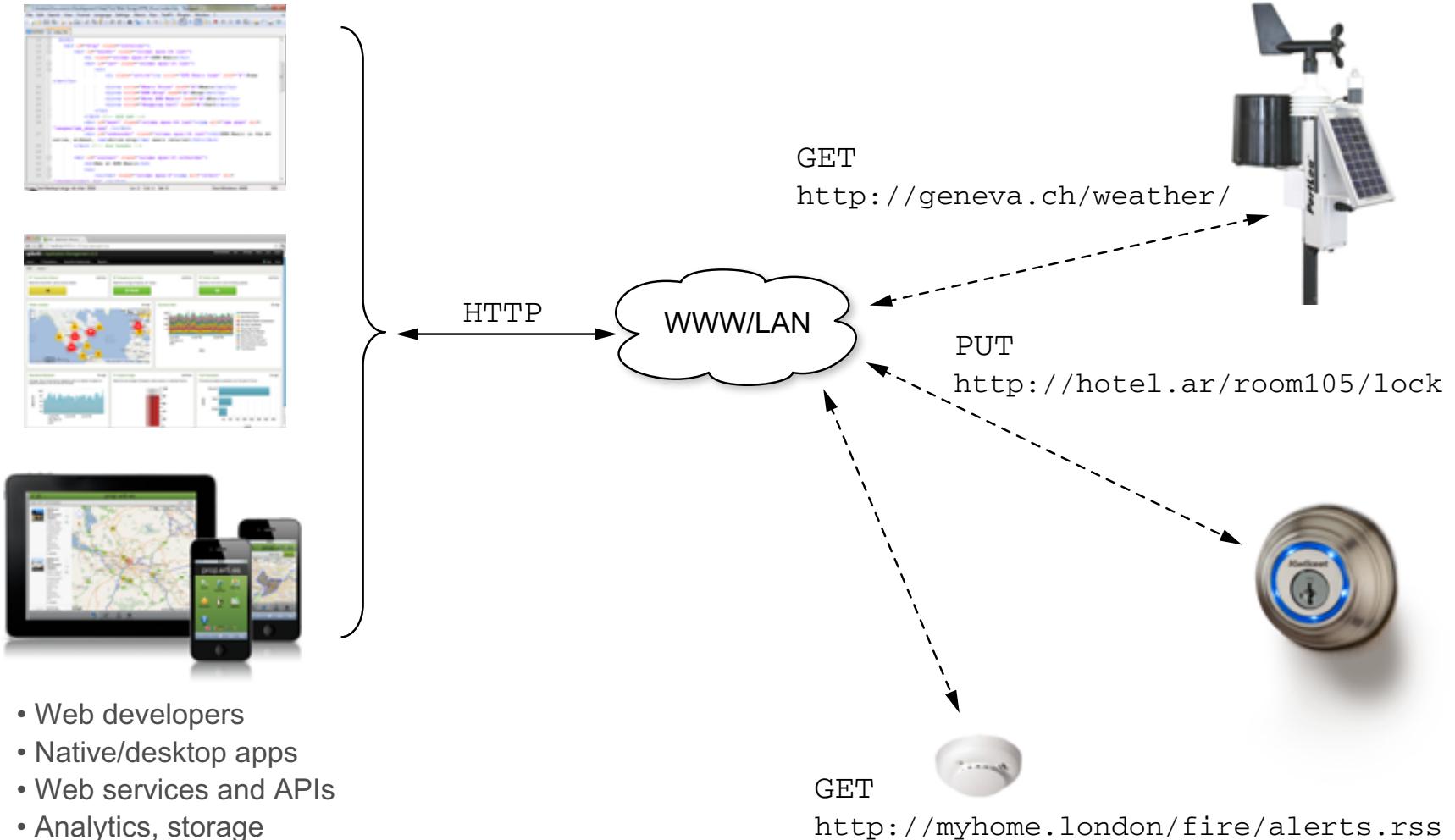


Encoding and Transport
(OSI layers 1-6)



More lightweight and optimized for embedded devices
(reduced battery, processing, memory and bandwidth usage),
more bespoke and hard-wired solutions.

WEB OF THINGS (WoT)



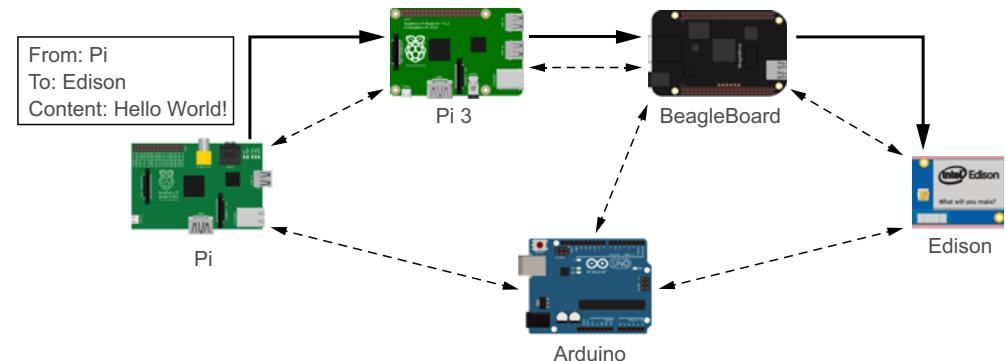
- Web developers
- Native/desktop apps
- Web services and APIs
- Analytics, storage

WOT - TOPOLOGIE DI RETE

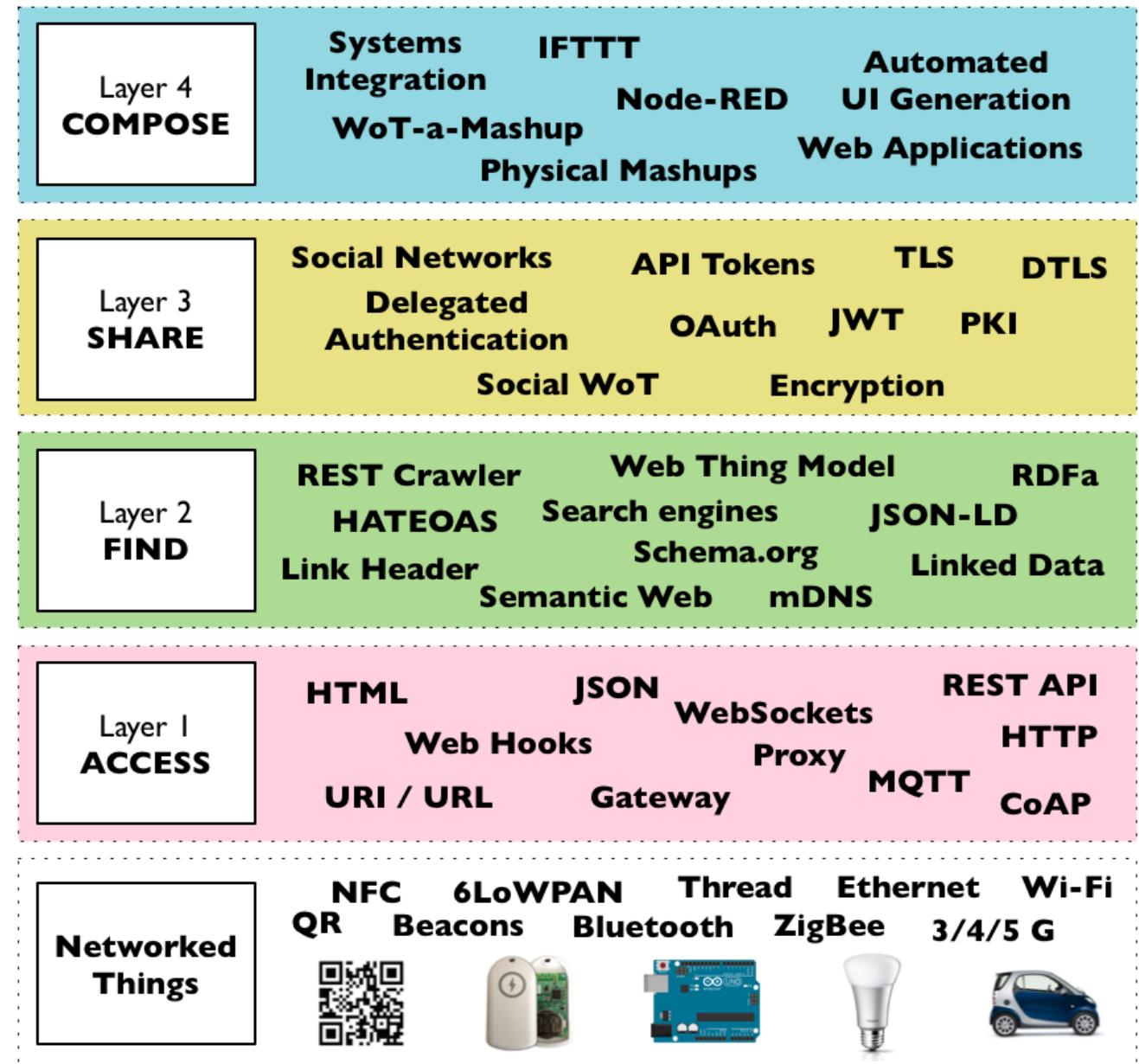
- Star topologies



- Mesh topologies
 - mesh networks
 - relays



WOT LIVELLI



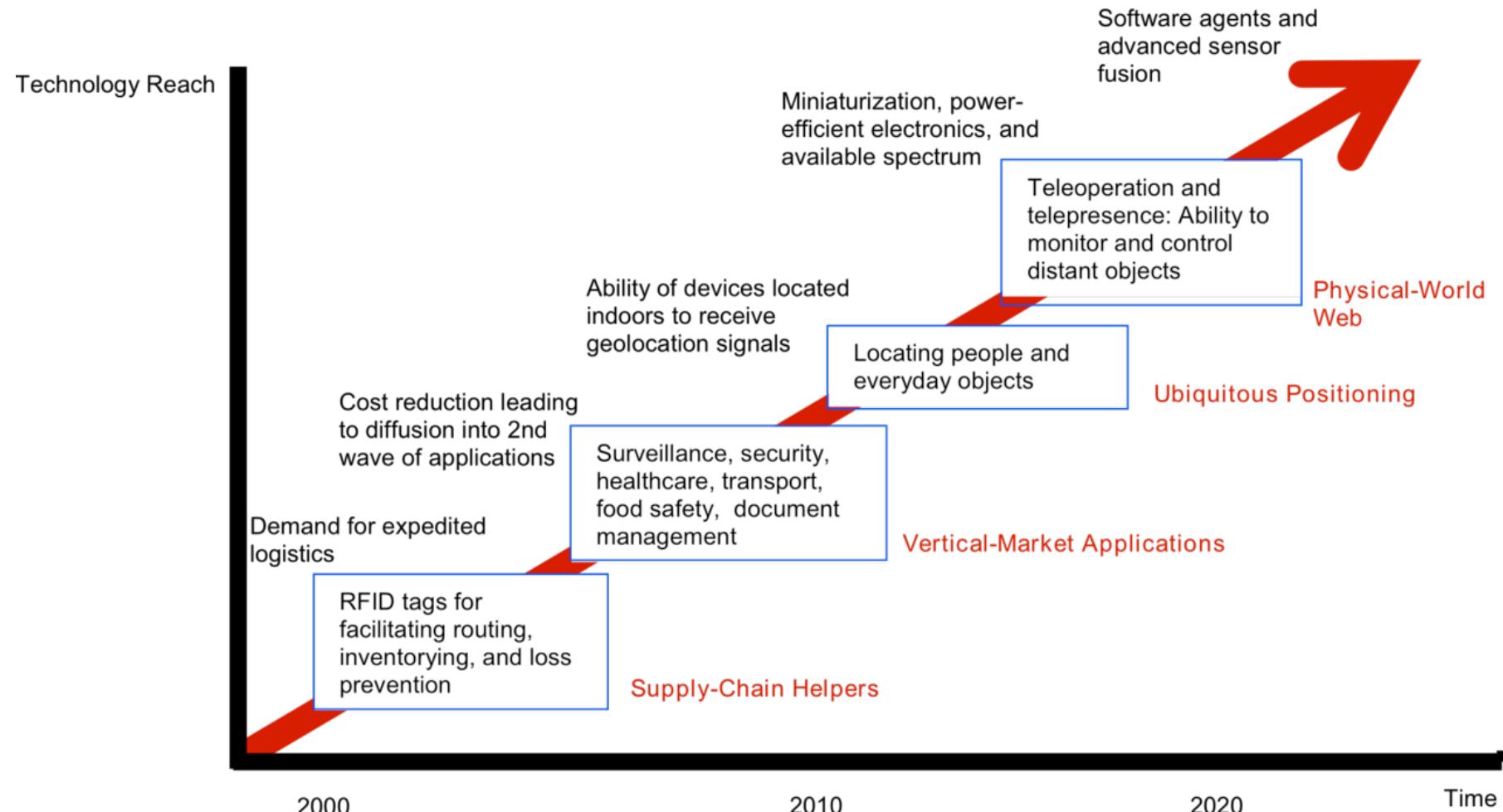
Source: Building the Web of Things: book.webofthings.io
Creative Commons Attribution 4.0

WEB OF THINGS AT W3C

- Web of Things Interest Group
 - <https://www.w3.org/WoT/>
- Web of Things Working group
 - <https://www.w3.org/WoT/WG/>
- First Submission of the Web Thing Model
 - <https://www.w3.org/Submission/2015/01/>
- Current proposals
 - WoT Architecture
 - <https://w3c.github.io/wot-architecture/>
 - WoT Things Description
 - <https://www.w3.org/TR/wot-thing-description/>

IOT: TECHNOLOGY ROADMAP

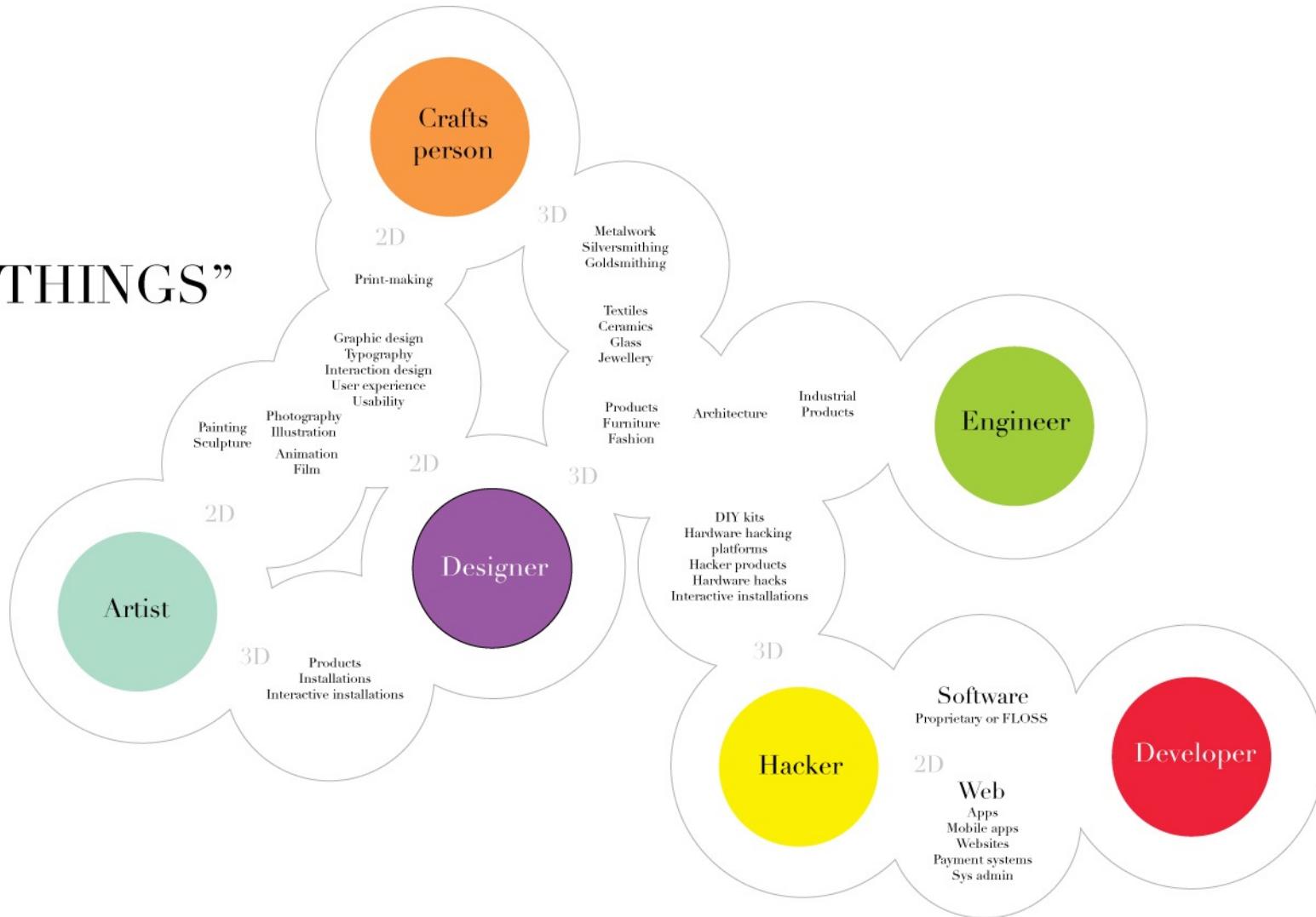
TECHNOLOGY ROADMAP: THE INTERNET OF THINGS



Source: SRI Consulting Business Intelligence

WHO IS MAKING IOT: MAKERS AND TINKERERS

“I MAKE THINGS”



MOBILE COMPUTING & IOT

- **Mobile computing**
 - sviluppo di sistemi software in esecuzione su dispositivi mobile
 - smartphone, tablet
 - fra gli aspetti peculiari:
 - interazione con lo user
 - energy-awareness
 - connessione alla rete intermittente
 - location-based/context-aware app
 - ...

MOBILE COMPUTING & IOT

- Nel contesto di IoT, due aspetti:
 - dispositivi mobili come interfaccia per interagire con smart Things & Environments
 - Personal Area Network (PAN) - Bluetooth, BLE, NFC.. - + Internet
 - dispositivi mobili come Thing
 - sensori, attuatori, connessione Internet
 - tracciamento user-related data

WEARABLE COMPUTING & IOT

- **Wearable computing & eyewear computing**
 - dispositivi indossabili
 - smart-watch, smart-glasses,...
 - spesso accoppiati con dispositivi mobile
 - Bluetooth
 - fra gli aspetti peculiari rispetto al mobile
 - modalità di utilizzo *hands-free / on-the-go*
- Nel contesto di IoT :
 - interfaccia per estendere la capacità di interazione con ambienti fisici/digitali/*aumentati*
 - Realtà Aumentata



BIBLIOGRAFIA E SITOGRAFIA

- Kevin Ashton. “*The ‘Internet of Things’ Thing*”. RFID Journal, 2009. <http://www.rfidjournal.com/articles/view?4986>
- Luigi Atzori, Antonio Iera, Giacomo Morabito. *The Internet of Things: A survey*. Computer Networks, 54 (2010)
- Samuel Greengard. *The Internet of Things*. MIT Press
- **[IOTF]** *IoT Fundamentals. Networking Technologies, Protocols, and Use Cases for the Internet of Things*. Hanes et al. CISCO press. 2017
- **[DIT]** A. McEwen and H. Cassimally. *Designing the Internet of Things*. Wiley
- **[LIT]** P. Waher. *Learning the Internet of Things*. Pack publishing.
- **[BIT]** Charalampos Doukas. *Building Internet of Things with the Arduino*.
- Guinard, Dominique; Vlad Trifa; Erik Wilde (2010). "A Resource Oriented Architecture for the Web of Things". Proc. of IoT 2010 (IEEE International Conference on the Internet of Things). Tokyo, Japan
- **[Por14]** M. Porter, J. Heppelmann. How Smart, Connected Products Are Transforming Competition. Harvard Business Review. November 2014
- **[Por15]** M. Porter, J. Heppelmann. How Smart, Connected Products Are Transforming Companies. Harvard Business Review. October 2015
- **[DBE17]** D. De Loach, E. Berthelsen, W. Elrifai. The Future of IoT. 2017
- **[VC18]** G. Veneri and A. Capasso. “Hands-On Industrial Internet of Things”. Packt, 2018

Sistemi Embedded e IoT

Ingegneria e Scienze Informatiche - UNIBO

a.a 2020/2021

Docente: Prof. Alessandro Ricci

[modulo-lab-4.1]

PROTOCOLLI IOT - CoAP E MQTT

COAP E MQTT - QUADRO SINTETICO CARATTERISTICHE

CoAP	MQTT
UDP	TCP
IPv6	
6LoWPAN	
802.15.4 MAC	
802.15.4 PHY	

[IoTF, p. 414]

Esempio di stack
basato su CoAP vs. MQTT

Factor	CoAP	MQTT
Main transport protocol	UDP	TCP
Typical messaging	Request/response	Publish/subscribe
Effectiveness in LLNs [*]	Excellent	Low/fair (Implementations pairing UDP with MQTT are better for LLNs.)
Security	DTLS	SSL/TLS
Communication model	One-to-one	many-to-many
Strengths	Lightweight and fast, with low overhead, and suitable for constrained networks; uses a RESTful model that is easy to code to; easy to parse and process for constrained devices; support for multicasting; asynchronous and synchronous messages	TCP and multiple QoS options provide robust communications; simple management and scalability using a broker architecture
Weaknesses	Not as reliable as TCP-based MQTT, so the application must ensure reliability.	Higher overhead for constrained devices and networks; TCP connections can drain low-power devices; no multicasting support

[*] LLN = low-power and lossy networks

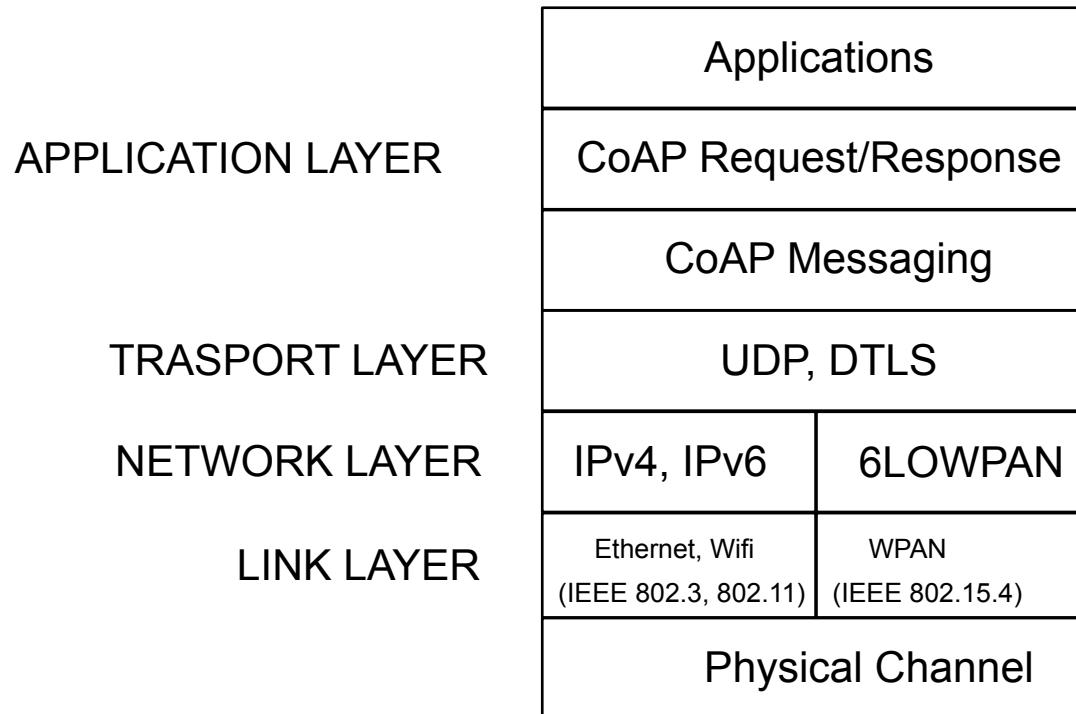
[IoTF, p. 436]

CoAP

permette l'uso di HTTP in modo più snello (utilizzando UDP). Utilizzato per device con poca capacità di calcolo.

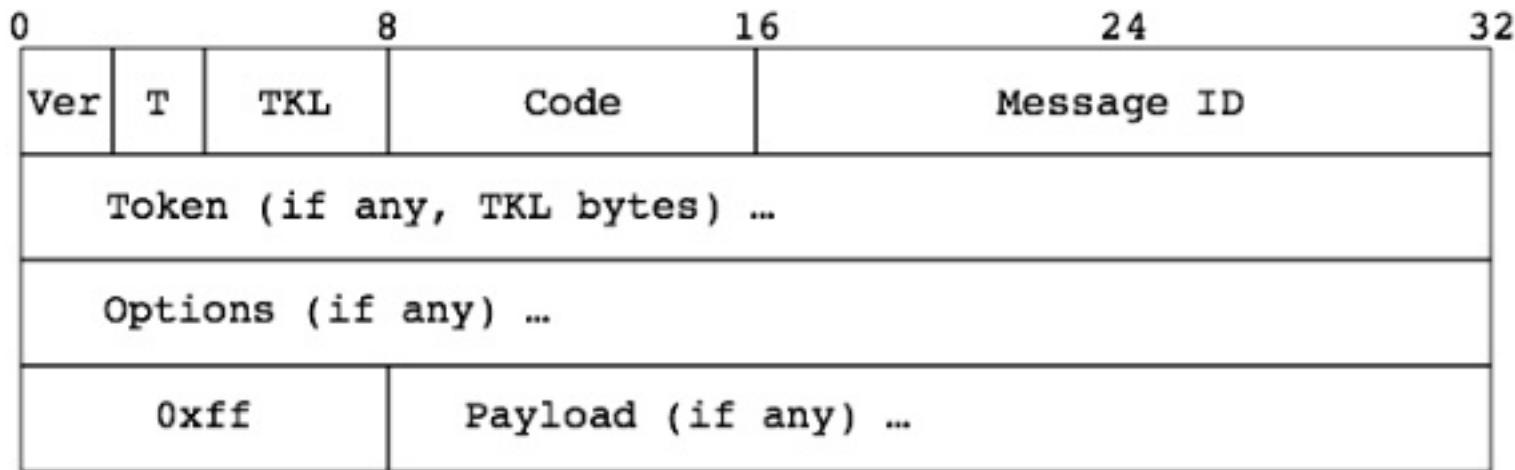
- **Constrained Application Protocol (CoAP)**
 - specialized Internet Application Protocol for constrained devices
 - RFC 7252: <https://tools.ietf.org/html/rfc7252>
 - It enables those constrained devices called "nodes" to communicate with the wider Internet using similar protocols
- CoAP is a service layer protocol
 - a main goals of CoAP is to design a generic web protocol for the special requirements of this constrained environment, especially considering energy, building automation, and other machine-to-machine (M2M) applications.
 - The goal of CoAP is not to blindly compress HTTP [RFC2616], but rather to realize a subset of REST common with HTTP but optimized for M2M applications.
 - can run on most devices that support UDP or a UDP analogue.

CoAP PROTOCOL STACK



CoAP MESSAGE FORMAT

- Minimum bytes are 4 bytes (empty message)
- Fitting inside a UDP packet



SIMPLIFIED HTTP

- Implement a simplified HTTP
 - compact binary headers
 - reduces the number of options available in the header
- Reduced set of methods
 - GET, POST, PUT, and DELETE
 - *confirmable* and *nonconfirmable* message services
 - with a confirmable message, the receiver always returns an acknowledgement
 - the sender can, in turn, resend messages if an acknowledgement is not returned within the given time period
- Reduced number of response code
- Simplified Internet Media Type scheme => reduced set of Content-Formats
 - <http://www.iana.org/assignments/core-parameters/>

NEW FEATURES & EXTENSIONS

- New features wrt HTTP
 - CoAP supports multicasting.
 - this can be used to detect devices or communicate through firewalls
- Useful extensions
 - block transfer algorithm
 - to transfer larger amounts of data.
 - event subscription and notification architecture
 - observable resources emitting notifications when events occur
- Security
 - encryption based on the Datagram Transport Layer Security (DTLS)

TECHNOLOGY

- Overview
 - <http://coap.technology>
- Examples
 - **Californium**
 - <https://github.com/eclipse/californium>
 - Copper (Cu)
 - Firefox add-on, which allows to view and interact with CoAP resources
 - <https://addons.mozilla.org/en-US/firefox/addon/copper-270430>
 - A web-based CoAP test tool: <http://coap.me/>

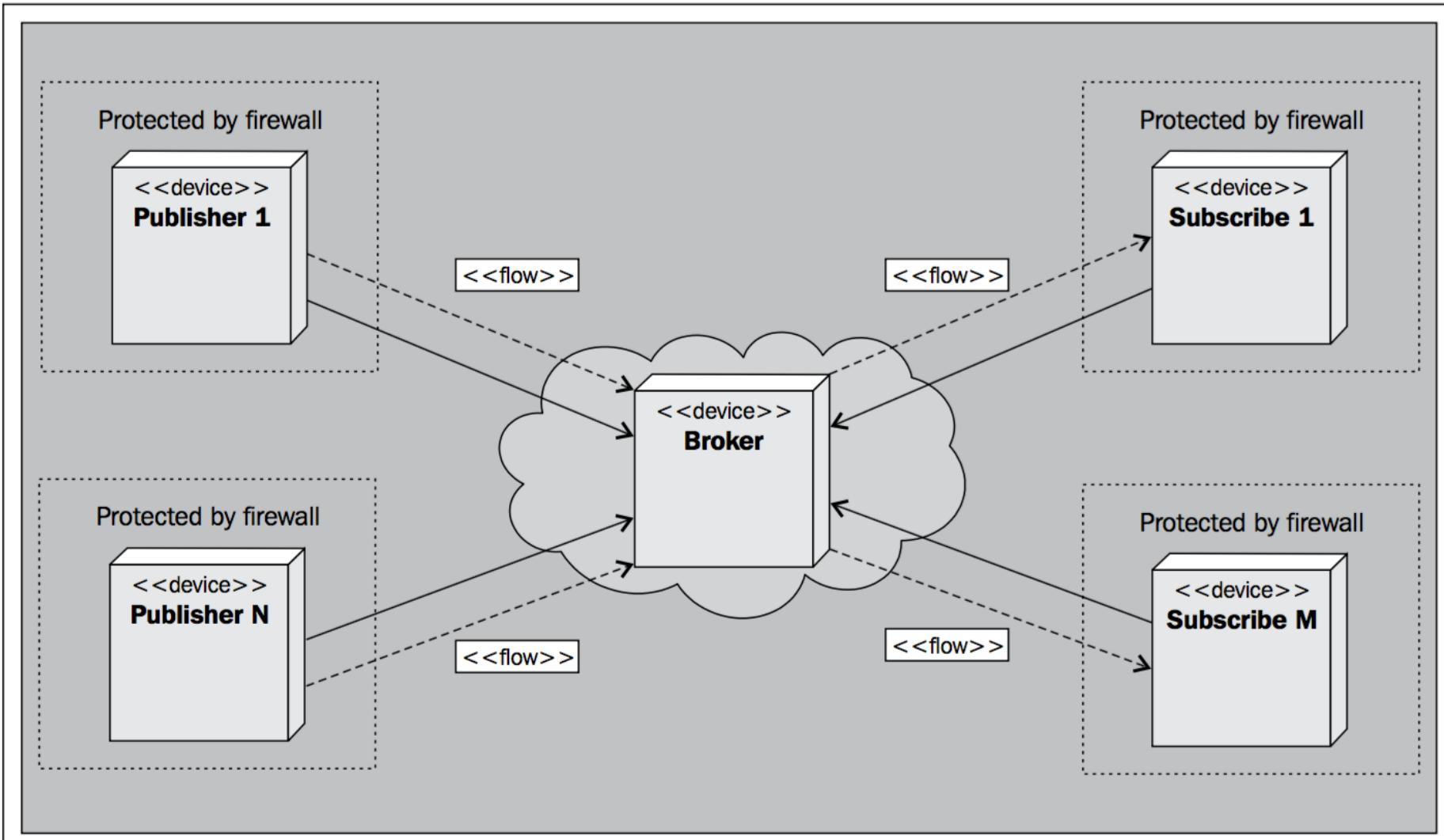
MQTT

- One of the major problems we can encounter when using HTTP or CoAP protocols is *how to cross firewall boundaries*
 - firewalls not only block incoming connection attempts, but they also hide a home or office network behind a single IP address
 - unless the firewall blocks outgoing connections, which it does only if explicitly configured to do so, we can cross firewall boundaries if *all the endpoints in a conversation act as clients to a common message broker that lies outside of the firewall and is therefore accessible to everybody.*
 - the message broker acts as a server, but all it does is relay messages between clients.
- One protocol that uses message brokers is the **Message Queue Telemetry Transport (MQTT)** protocol
 - <http://mqtt.org/>
 - <https://mqtt.org/software/>

PUBLISH SUBSCRIBE

- The MQTT protocol is based on the **publish/subscribe** pattern
- Three types of actors:
 - **Publishers**
 - they connect to the message broker and publish content
 - **Subscriber**
 - they connect to the same message broker and subscribe to content that they are interested in
 - **Message broker**
 - it makes sure that the published content is relayed to interested subscribers

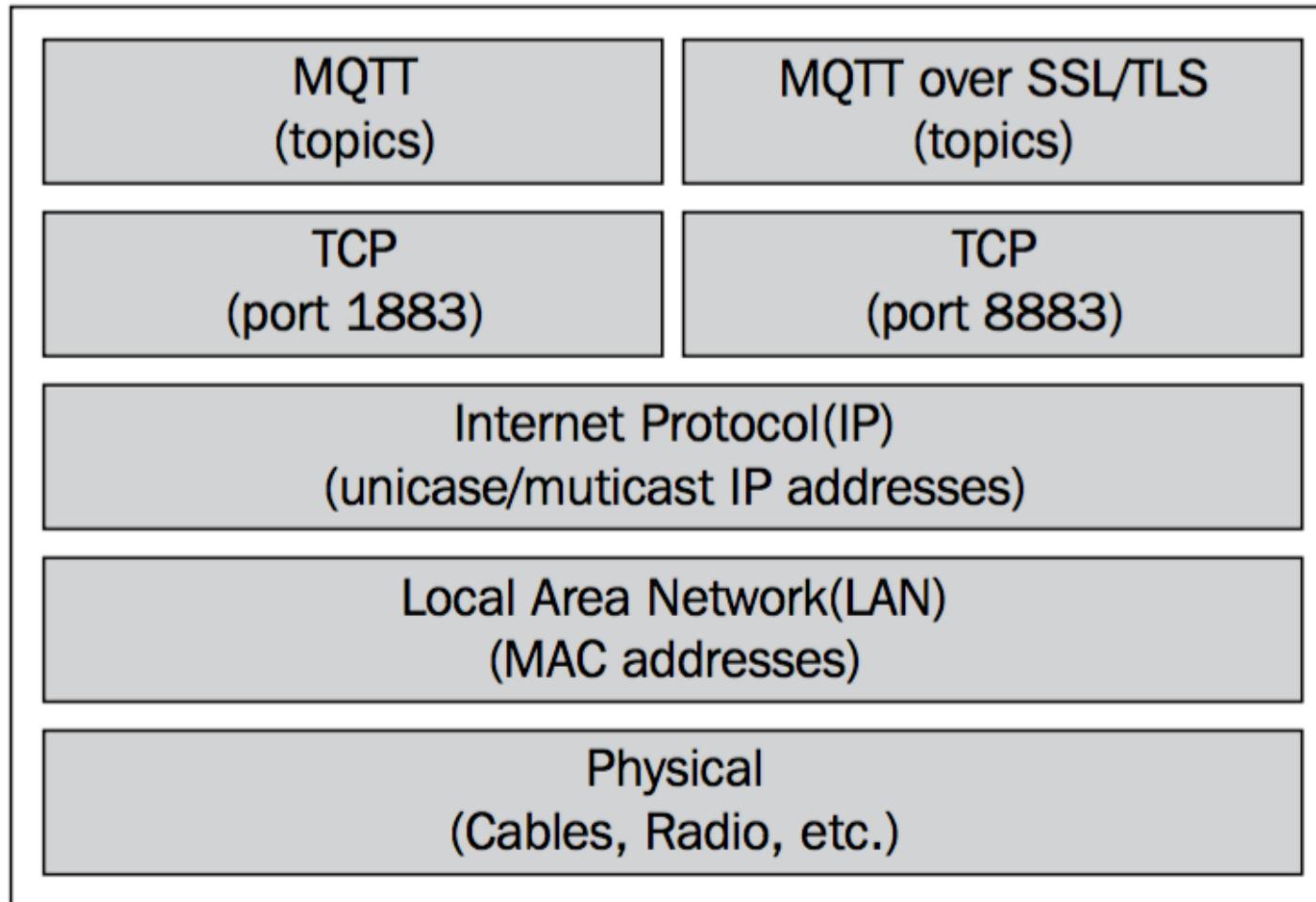
MQTT P/S ARCHITECTURE



TOPICS AND TOPICS TREE

- Content is identified by **topic**
 - when publishing content, the publisher can choose whether the content should be retained by the server or not
 - If retained, each subscriber will receive the latest published value directly when subscribing
- Topics are ordered into a **tree structure of topics**, much like a filesystem
 - the forward slash character (/) is used as a delimiter when describing a topic path.
 - when subscribing to content, a subscriber can subscribe to either a specific topic by providing its path, or an entire branch using the hash wildcard character (#)

MQTT PROTOCOL STACK



QUALITY OF SERVICE LEVELS

- There are three Quality of Service levels in MQTT available while publishing content.
 - **unacknowledged service**
 - the message is delivered *at most once* to each subscriber.
 - **acknowledged service**
 - each recipient acknowledges the receipt of the published information
 - If no receipt is received, the information can be sent again.
 - This makes sure the information is delivered *at least once*
 - **assured service**
 - the information is not only acknowledged but sent in two acknowledged steps
 - first transmitted and then delivered
 - this makes it possible to make sure that the content is delivered *exactly once* to each subscriber.

WEAK SECURITY

- Weak support for user authentication
 - plain text username and password authentication exists, but it provides an obvious risk if the server is not hosted in a controlled environment.
- To circumvent the most obvious problems, MQTT can be used over an encrypted connection using SSL/TLS.
 - in this case, it is important for clients to validate server certificates else user credentials may be compromised
- Other methods, not defined in the MQTT protocol itself, include the use of *client-side certificates* or *pre-shared keys* to identify clients, instead of using the username and password option provided by the protocol
- Proprietary methods of encrypting the contents can also be used to make sure only receivers with sufficient credentials can decrypt the contents.
 - even though this method works, it reduces interoperability and provides an additional load on each device, which is contrary to the stated goal of the protocol.
- As the MQTT protocol itself does not consider security, it is very important for developers to consider security themselves.

MQTT TECHNOLOGY

- Main reference
 - <http://mqtt.org>
 - OASIS standard:
 - <https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html>
- Available software
 - <https://mqtt.org/software/>
 - Two main kinds
 - clients - <https://github.com/mqtt/mqtt.org/wiki/libraries>
 - among the others:
 - » Arduino library - <https://github.com/knolleary/pubsubclient>
 - servers - <https://github.com/mqtt/mqtt.org/wiki/servers>
 - among the others:
 - » Eclipse Mosquitto - <https://www.mosquitto.org/>

COMPARISON AT A GLANCE

Feature	HTTP	CoAP	MQTT
Request/Response	✓	✓	✗
Publish/Subscribe	✗	✗	✓
Multicast	✗	✓	✗
Events or Push	✓	✓	✓
Bypasses firewall	✗	(✓)	✓
Federation	✗	✗	✗
Authentication	✓	✓	✓
Network Identity	(✓)	(✓)	✗
Authorization	✗	✗	✗
Encryption	✓	✓	✓
End-to-end encryption	✗	✗	✗
Compression	✓	✗	✗
Streaming	✓	✗	✓
Reliable messaging	✗	✗	✓✓
Message Queues	✗	✗	✗

Sistemi Embedded e IoT

Ingegneria e Scienze Informatiche - UNIBO

a.a 2019/2020

Docente: Prof. Alessandro Ricci

[modulo-4.2]

TECNOLOGIE E PROTOCOLLI
PER LA COMUNICAZIONE NEI
SISTEMI EMBEDDED

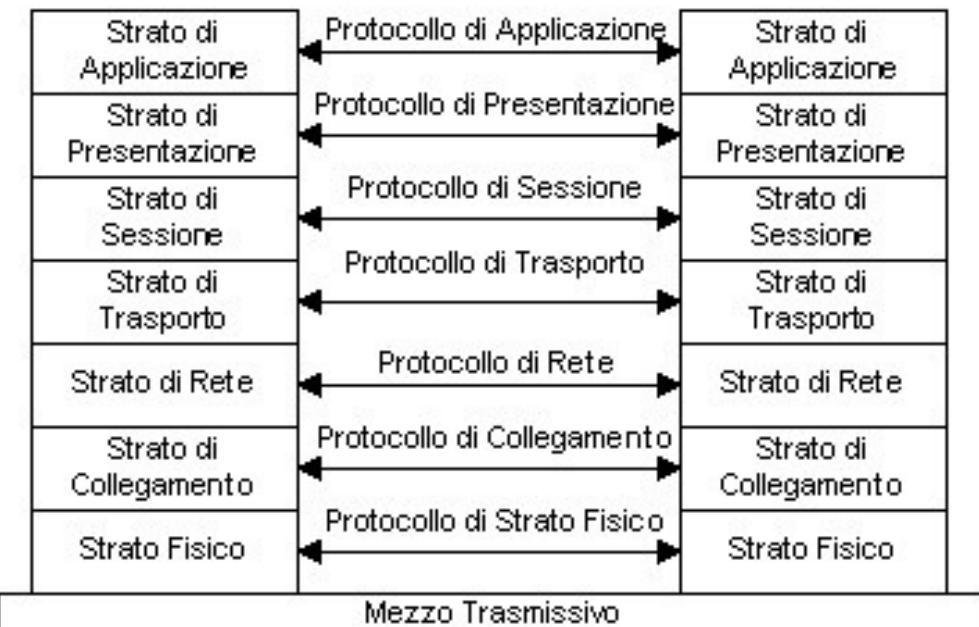
LA COMUNICAZIONE IN SISTEMI M2M e INTERNET OF THINGS

- La comunicazione fra dispositivi è un aspetto cruciale di M2M e Internet of Things, ovvero la capacità di scambiare informazioni via rete fra dispositivi e sottosistemi - embedded e non
- La comunicazione può essere sia **wired**, sia **wireless**.
- In entrambi i casi i dispositivi vengono dotati di opportuni moduli di comunicazione
 - nel caso wired tipicamente sono shield/schede con porta Ethernet e protocollo 802.11
 - nel caso wireless, possono essere shield oppure dongle USB contenente il modulo con ricevitore/trasmettitore e relativa antenna
- La comunicazione fra micro-controllore e moduli di comunicazione avviene usualmente mediante la porta seriale

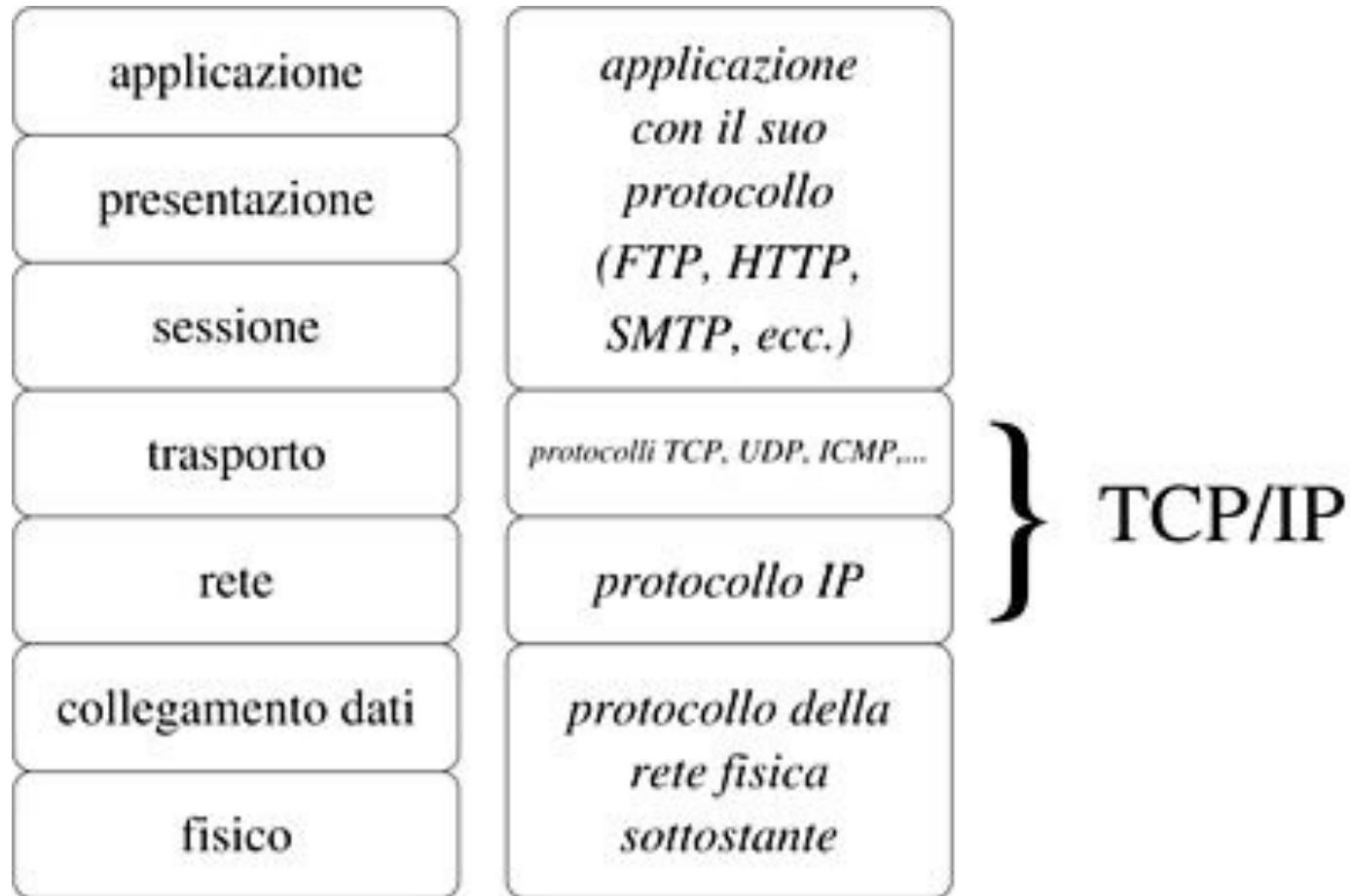
CONNESSIONE AD INTERNET

- Ci sono due modi principali per connettere un dispositivo embedded alla rete Internet in un ottica IoT:
 - **indirettamente**, mediante la comunicazione di un nodo intermediario che funge da **gateway**
 - è il caso, ad esempio, di architetture in cui si usa la tecnologia wireless ZigBee o Bluetooth
 - **direttamente**, montando un modulo di comunicazione opportuno
 - è il caso, ad esempio, di moduli 3G/4G oppure wifi
- In questo caso, a prescindere dalla tecnologia wireless utilizzata, il protocollo di riferimento è quello di Internet, ovvero IP, TCP/IP o UDP/IP

RICHIAMI: ARCHITETTURA ISO-OSI



RICHIAMI: STACK INTERNET

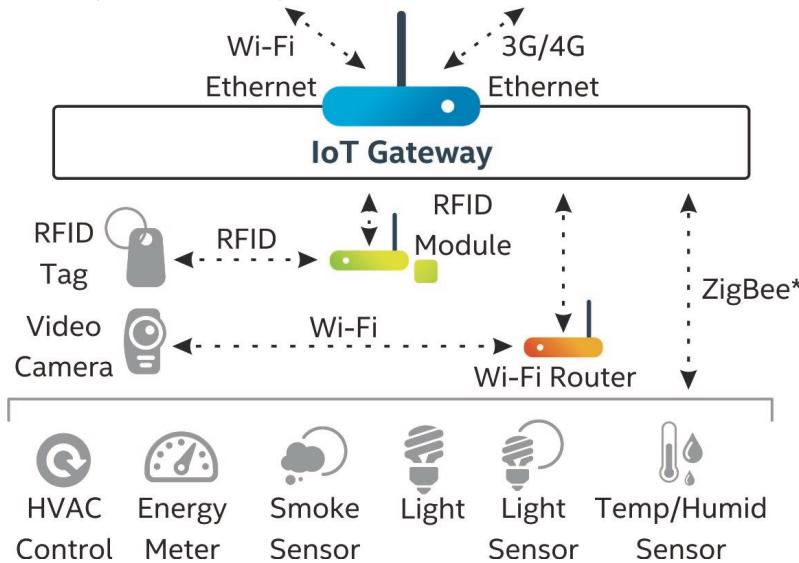


User Interface Devices

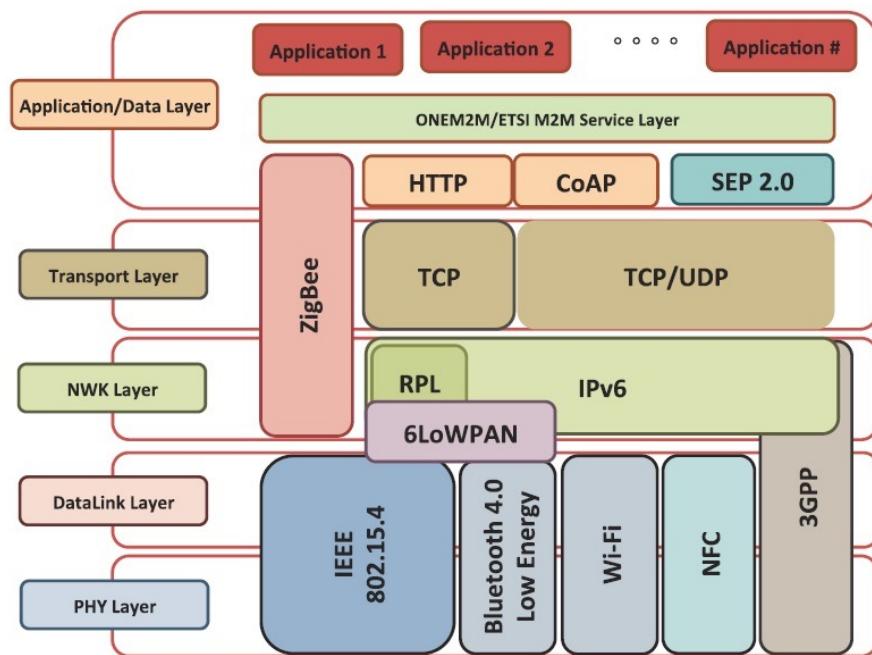
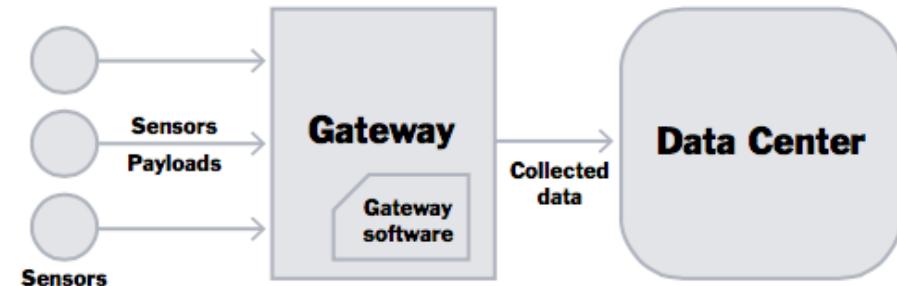
Control and Monitoring



Tablet, Smart Watch, etc.



<https://blogs.intel.com/iot/2015/04/15/how-the-internet-of-things-can-unlock-the-door-to-a-more-robust-bms/>



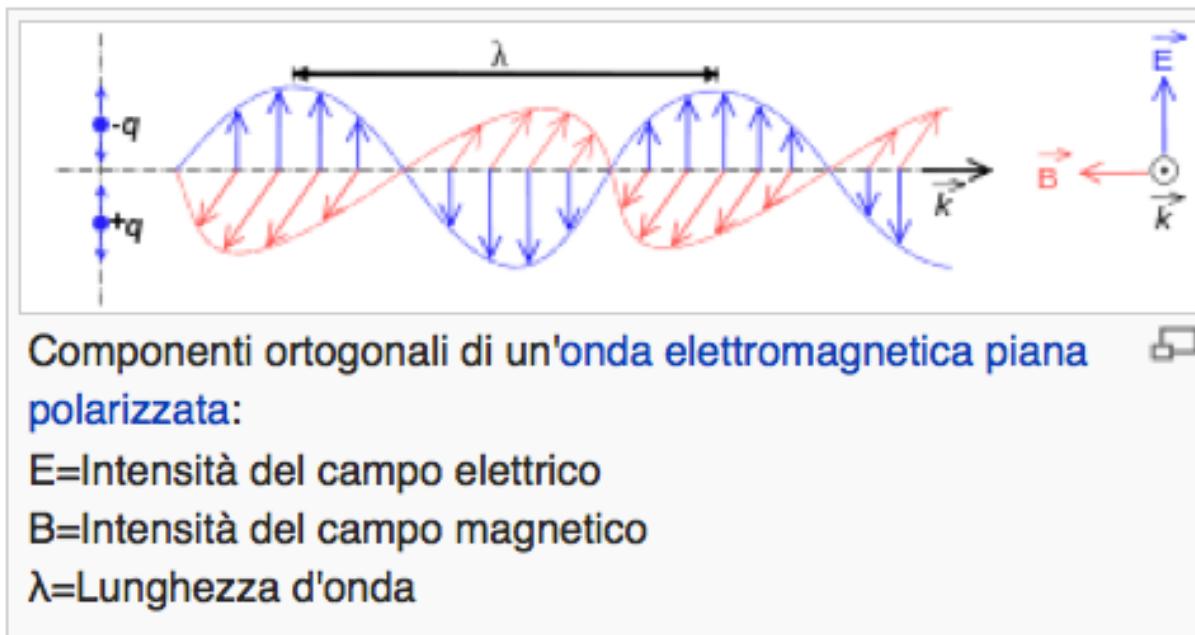
Heterogeneous standards environment in IoT (Source: IoT Research EU)

COMUNICAZIONI WIRELESS: CENNI INTRODUTTIVI / PREMESSA

- Nell'ambito delle telecomunicazioni, le comunicazioni wireless (senza fili) sono forme di **radiocomunicazione**
 - fanno uso del mezzo o **canale radio**, diffuse nell'etere al fine di trasportare a distanza l'informazione tra utenti attraverso **segnali elettromagnetici** appartenenti alle frequenze radio o microonde dello spettro elettromagnetico detta anche banda radio
 - i segnali inviati a tali frequenze vengono perciò detti segnali a radiofrequenza (RF) e il collegamento ottenuto radiocollegamento).
- Una radiocomunicazione può essere
 - *terrestre* se si appoggia ad infrastrutture di telecomunicazioni poste sulla superficie terrestre
 - *satellitare* se si appoggia almeno in parte ad infrastrutture poste in orbita sulla Terra come i satelliti artificiali per le telecomunicazioni
- Un esempio tipico di radiocomunicazione sono i ponti radio, le infrastrutture di radiodiffusione, telediffusione, l'accesso a reti radiomobili cellulari e le reti satellitari.

COMUNICAZIONI WIRELESS: CENNI INTRODUTTIVI / PREMESSA

- In fisica, un segnale elettromagnetico è un fenomeno ondulatorio dato dalla propagazione nello spazio-tempo in fase del campo elettrico e del campo magnetico, oscillanti in piani tra loro ortogonali e ortogonali alla direzione di propagazione



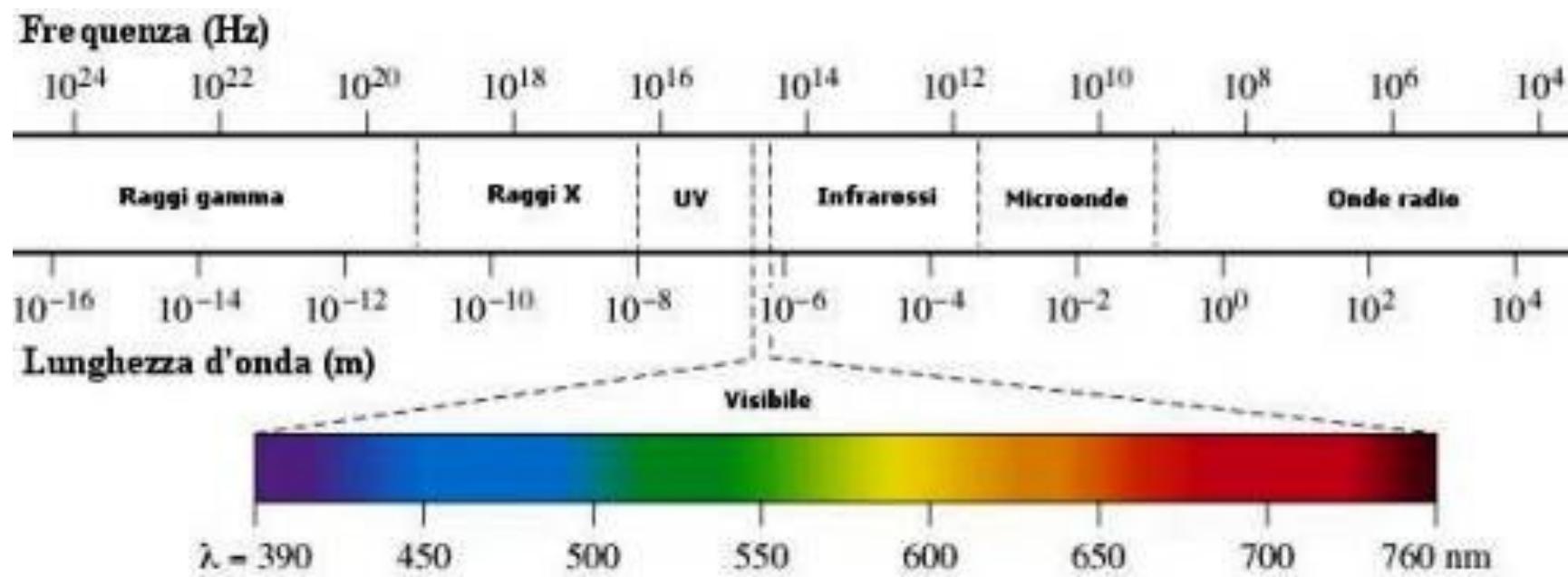
- La radiazione elettromagnetica può propagarsi nel vuoto, in mezzi poco densi come l'atmosfera, oppure in strutture guidanti come le guide d'onda.

COMUNICAZIONI WIRELESS: CENNI INTRODUTTIVI / PREMESSA

- Dualismo natura ondulatoria e particellare
 - come **fenomeno ondulatorio**, è descrivibile matematicamente come soluzione dell'equazione delle onde, a sua volta ottenuta a partire dalle **equazioni di Maxwell** secondo la teoria dell'elettrodinamica classica
 - pur essendo un fenomeno ondulatorio, la radiazione elettromagnetica ha anche una **natura quantizzata** che le consente di essere descritta come un **flusso di fotoni**, che nel vuoto viaggiano alla velocità della luce.
- Le applicazioni tecnologiche che sfruttano la radiazione elettromagnetica sono svariate. In generale si possono distinguere due macrofamiglie applicative:
 - nella prima figurano le onde elettromagnetiche utilizzate per trasportare informazioni (radiocomunicazioni come radio, televisione, telefoni cellulari, satelliti artificiali, radar, radiografie),
 - nella seconda quelle per trasportare energia, come il forno a microonde

COMUNICAZIONI WIRELESS: CENNI INTRODUTTIVI / PREMESSA

- Spettro delle frequenze delle onde elettromagnetiche



COMUNICAZIONI WIRELESS: CENNI INTRODUTTIVI / PREMESSA

- Radiocomunicazioni - cenni storici
 - la storia inizia dalla rilevazione sperimentale delle onde elettromagnetiche da parte di **Hertz** nell'Ottocento, teorizzate precedentemente da **Maxwell**
 - a partire dagli esperimenti di radiopropagazione da parte di **Nikola Tesla** e **Guglielmo Marconi** rispettivamente alla fine del XIX secolo e nei primi anni del Novecento, le radiocomunicazioni trovano applicazione pratica nelle radiodiffusioni e telediffusioni con l'invenzione della radio prima e della televisione poi...
 - ...fino a giungere alle moderne reti radiomobili cellulari, ai sistemi di radiocomunicazione terrestri, marittimi e aerei ed alle comunicazioni satellitari

COMUNICAZIONI WIRELESS: CENNI INTRODUTTIVI / PREMESSA

- Un **sistema di radiocomunicazione** è composto da:
 - un **canale trasmissivo** che comprende oltre alla tratta radio anche **l'emettitore** e il **ricevitore**, le **antenne** (emittente e ricevente),
 - un **modem** cioè un modulatore in trasmissione e un demodulatore in ricezione
 - convertendo sequenze di bit in segnali elettrici e viceversa
 - da un **codec** ovvero un codificatore del segnale in ingresso e un decodificatore del segnale in uscita al canale radio
 - serve per codificare e/o decodificare digitalmente un segnale perché possa essere salvato su un supporto di memorizzazione o richiamato per la sua lettura.
- In un sistema radio ricetrasmettente tipicamente l'antenna fisica è unica e deputata sia alla trasmissione che alla ricezione

COMUNICAZIONI WIRELESS: CENNI INTRODUTTIVI / PREMESSA

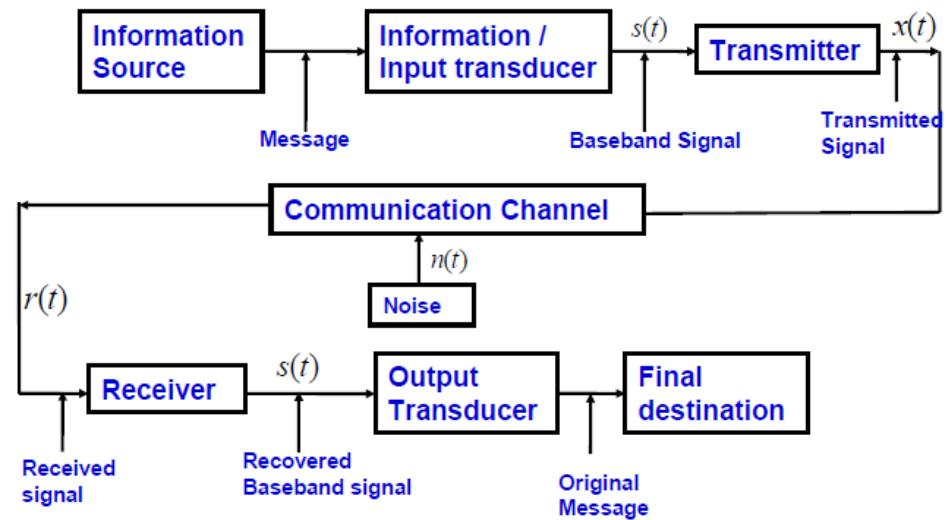
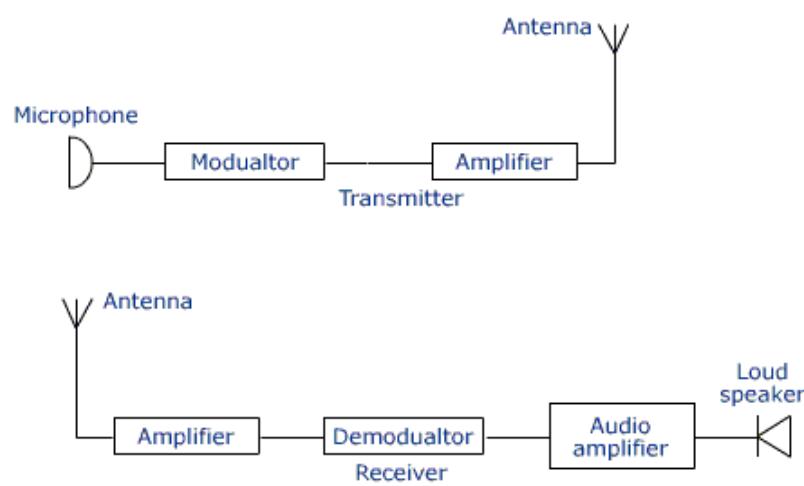
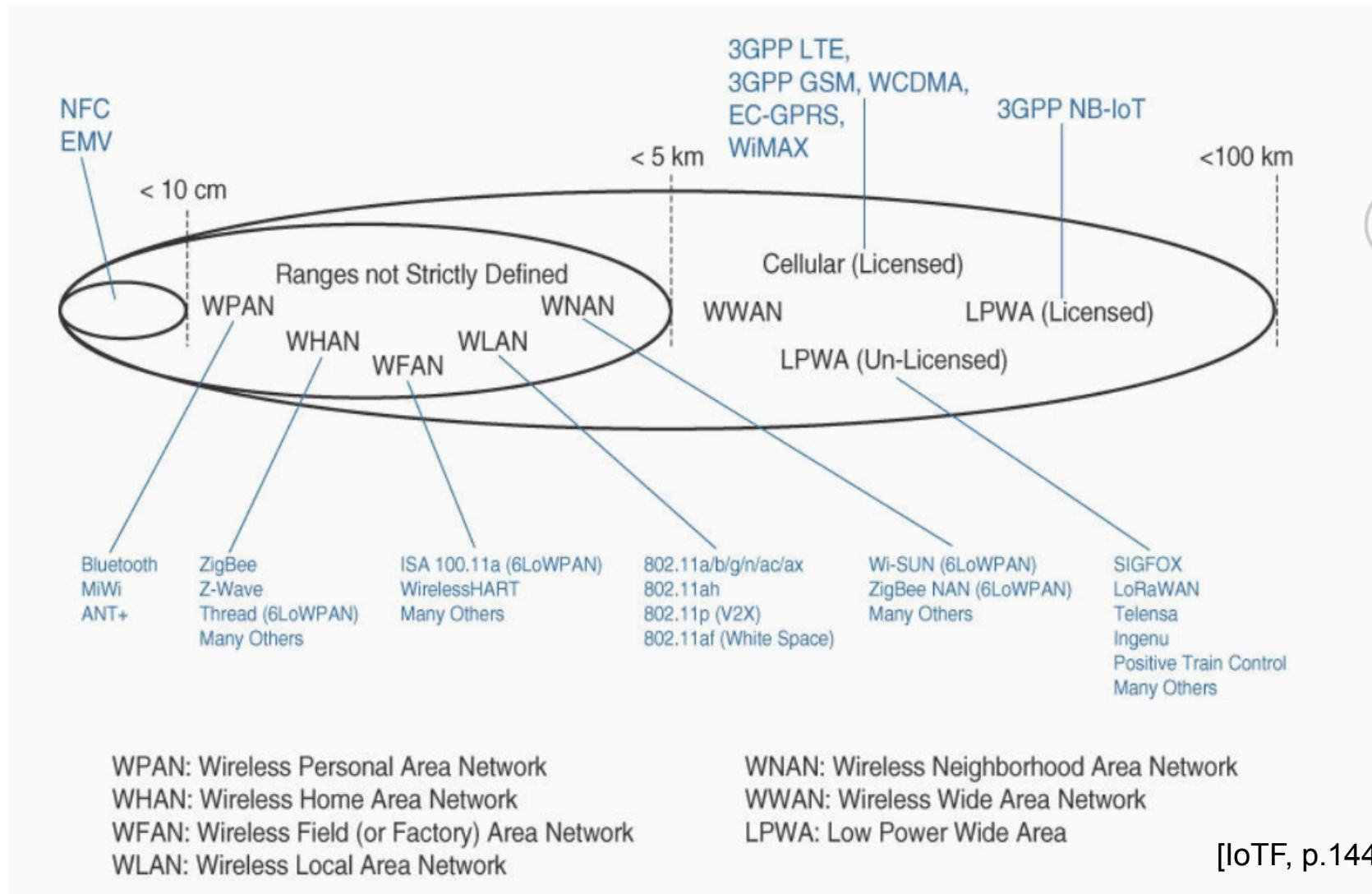


Figure 1.1: Block Schematic Diagram of a Basic Communication System

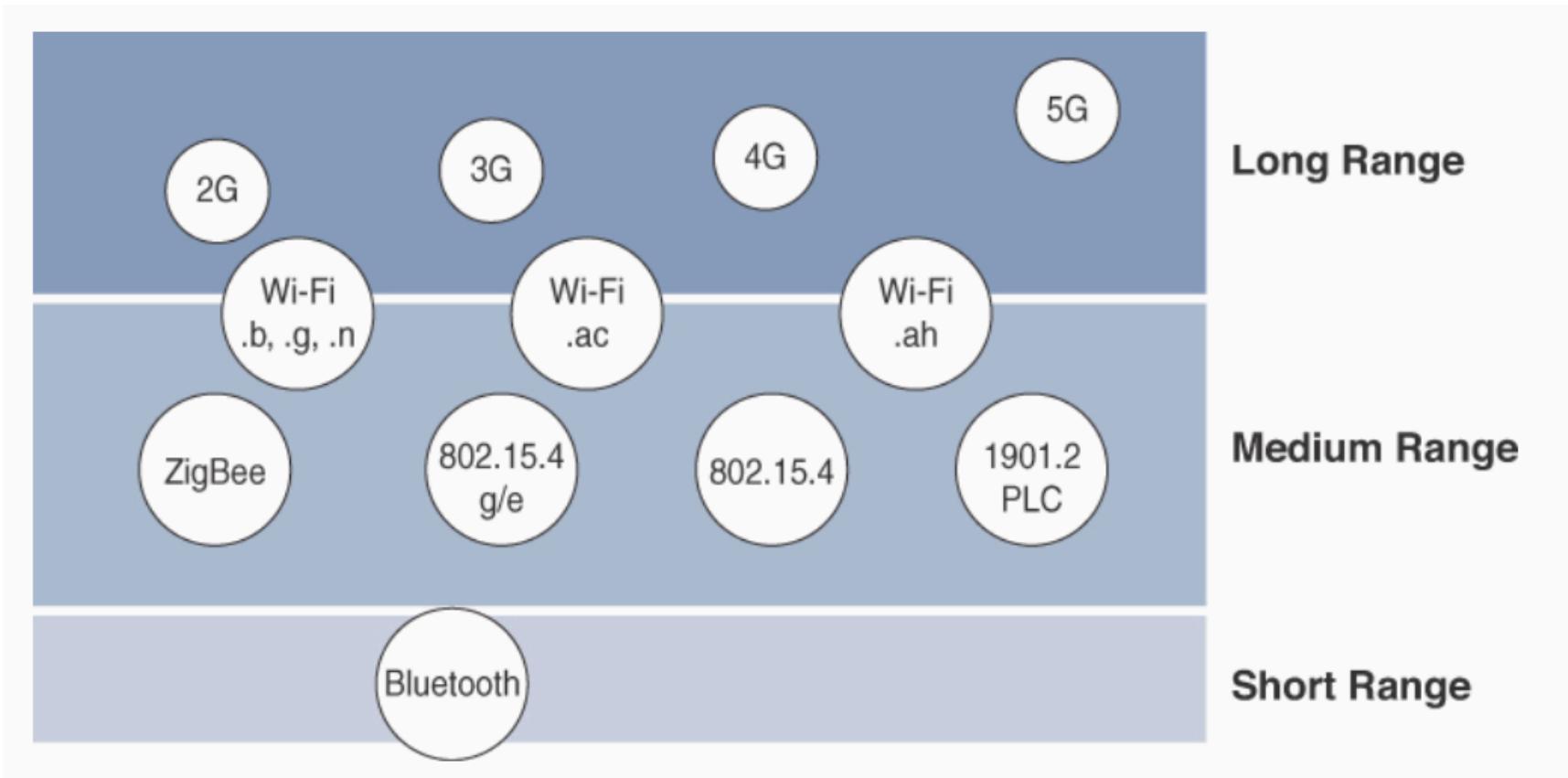
TECNOLOGIE E PROTOCOLLI WIRELESS

- **Short Range** (decine di metri)
 - ambito consumer, meno usati in ambito industriale
 - esempi
 - IEEE 802.15.1 (Bluetooth)
 - IEEE 802.15.7 (Visible Light Communication)
- **Medium Range** (centinaia di metri fino a 1 miglio)
 - fascia di riferimento per IoT
 - esempi
 - IEEE 802.11 Wi-Fi
 - IEEE 802.15.4 e IEEE 802.15.4g WPAN
- **Long Range** (distanze superiori al miglio)
 - reti cellulari (2G, 3G, 4G, **5G**)
 - outdoor IEEE 802.11 Wi-Fi
 - tecnologie Low-Power Wide-Area (LPWA)

QUADRO TECNOLOGIE WIRELESS IN RELAZIONE ALLA DISTANZA



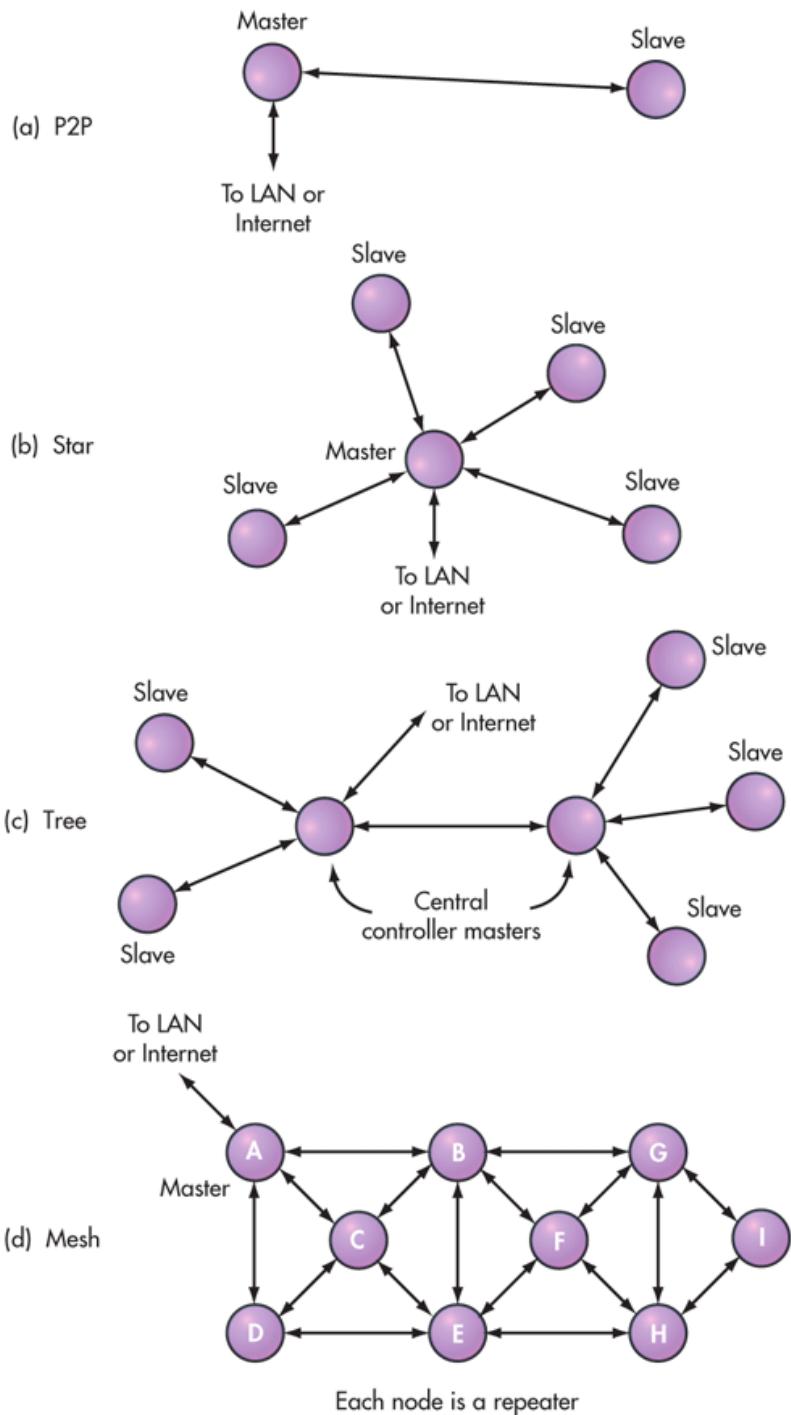
QUADRO TECNOLOGIE WIRELESS IN RELAZIONE ALLA DISTANZA



[IoTF, p.230]

TOPOLOGIE RETI WIRELESS

- Principali topologie usate nell'embedded e IoT
 - P2P
 - Star
 - Tree
 - Mesh



WIFI

- Standard IEEE 802.11x
- Protocollo più utilizzato per la connessione wireless di dispositivi ad Internet
 - supporto per protocolli Internet
 - IP, TCP/IP, UDP/IP
- Caratteristiche
 - Data rate elevati: 54 Mbps
 - Communication range: ~150 m
 - Frequenza: 5 GHz
- Utilizzata anche in ambito embedded
 - sebbene non sia stato progettato allo scopo
 - consumi rilevanti

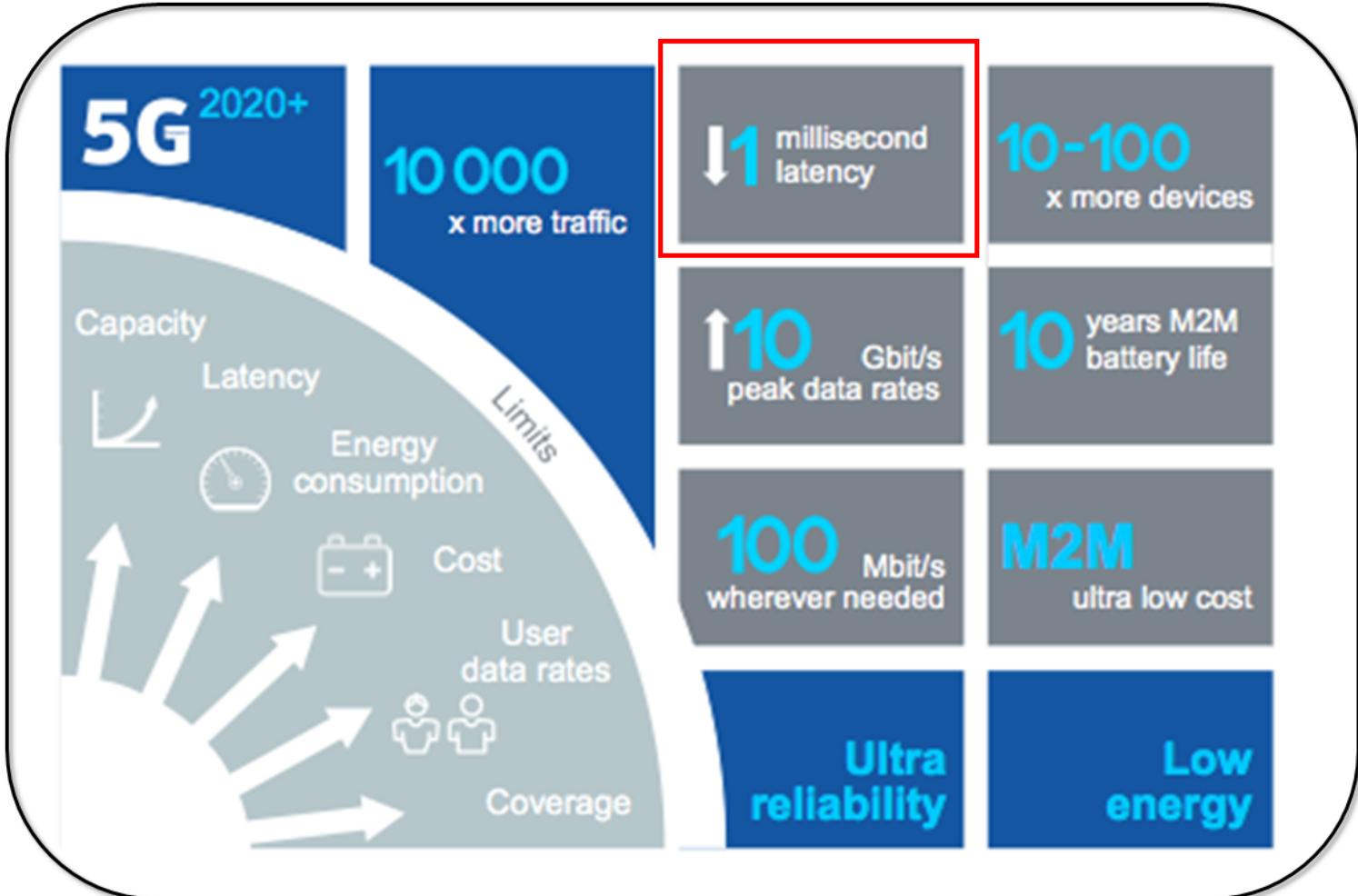
RETI CELLULARI (PRE 5G)

- Comunicazione mediante rete dati cellulare broad-band
- Evoluzione degli standard
 - GPRS, 3G, WiMax, LTE (4G)
- Caratteristiche:
 - Data rate: da 80 Kbps (GPRS) ad alcuni Mbps (3G e 4G)
 - Range: km
 - Frequenze: 800 MHz, 1900 MHz
- Consumi elevati
 - per cui non sono ideali per IoT

5G

- Tecnologia "disruptive" anche per il mondo IoT
 - velocità: 1-10Gbps
 - riduzione delle latenze (1-10ms)
 - aumento dell'affidabilità, anche in condizioni critiche
 - maggiore flessibilità rispetto al Wifi, pervasività nei dispositivi (sensori, wearable..)
- introduzione/overview
 - <https://www.cbinsights.com/research/5g-next-gen-wireless-system/>
 - https://www.arpaе.it/dettaglio_generale.asp?id=4149&idlivello=2145

5G E IOT



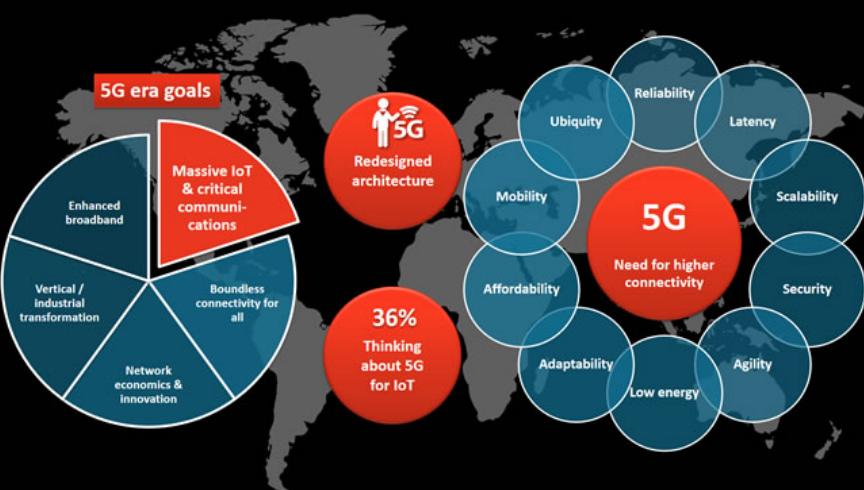
THE FUTURE OF 5G AND IOT

Finishing the foundations in 2018 and 2019 for the next two decades

www.i-scoop.eu/internet-of-things-guide/5g-iot/

By 2021, 5G's broad enablement of IoT use cases will drive 70% of G2000 companies to spend \$1.2 billion on connectivity management solutions

(IDC, November 1, 2017)



The global 5G value chain will generate \$3.5 trillion in output and support 22 million jobs in 2035

(IHS Markit)



Sources

<https://www.idc.com/research/viewtoc.jsp?containerId=US43161517>
<https://www.gsmaintelligence.com/research/?file=0efdd9e7b6eb1c4ad9aa5d4c0c971e62&download>
<https://www.ihs.com/info/0117/5g-technology-global-economy.html>
<http://www.vodafone.com/business/iot/iotbarometer>

36% of organizations already think about 5G for IoT which they see as an incremental upgrade

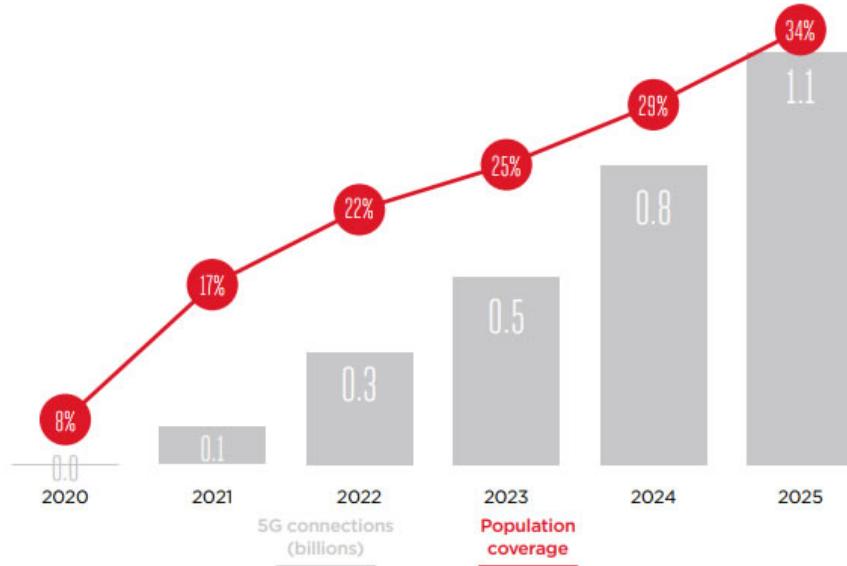
(Vodafone IoT barometer, September 2017)



5G E IOT

Source: GSMA Intelligence

Global 5G coverage and adoption



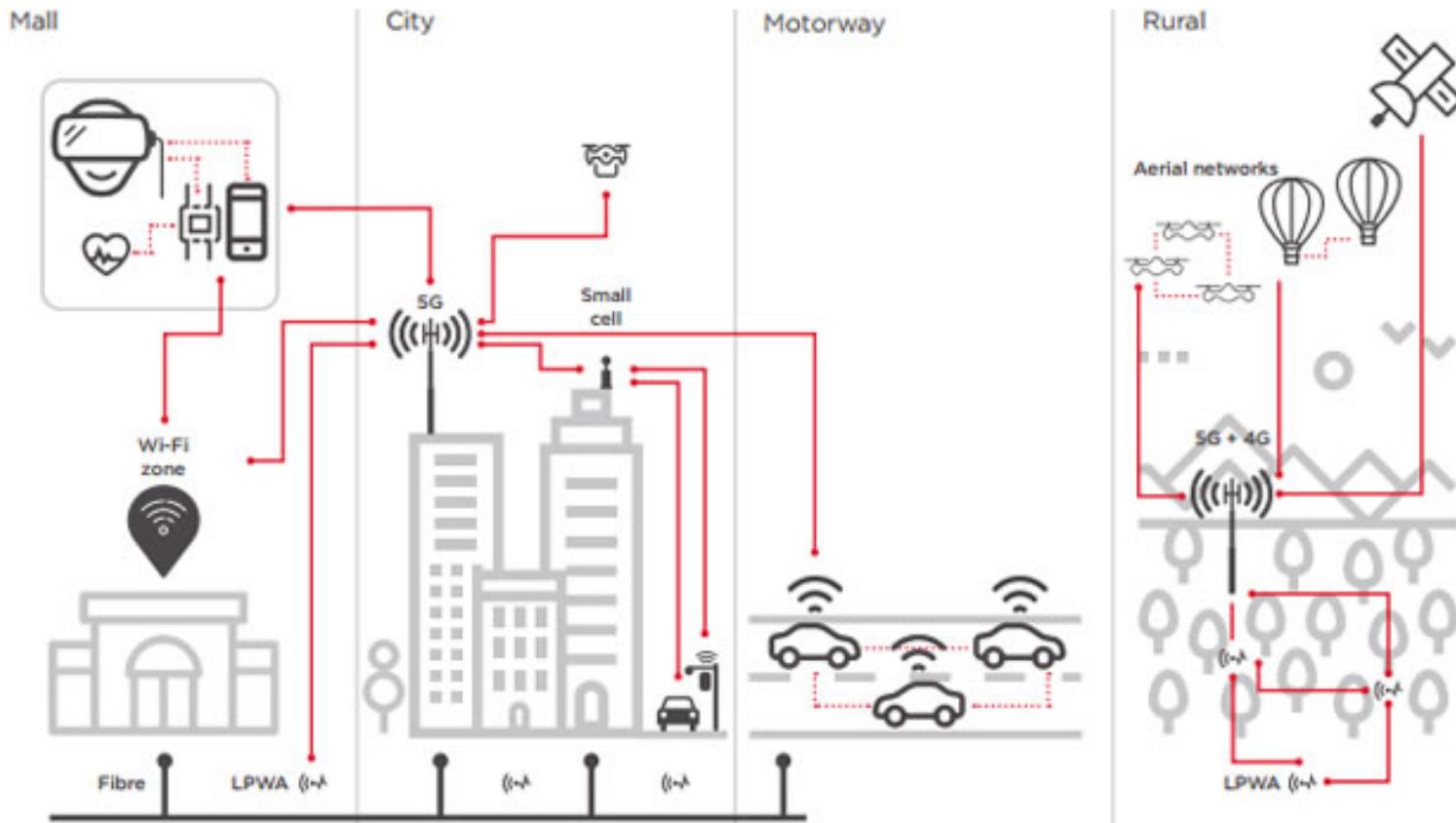
<https://www.i-scoop.eu/5g/5g-iot/>

COMUNICAZIONE

5G E IOT

Source: GSMA

5G as the centre of a heterogeneous network environment



<https://www.i-scoop.eu/5g/5g-iot/>

CONSUMO ENERGIA

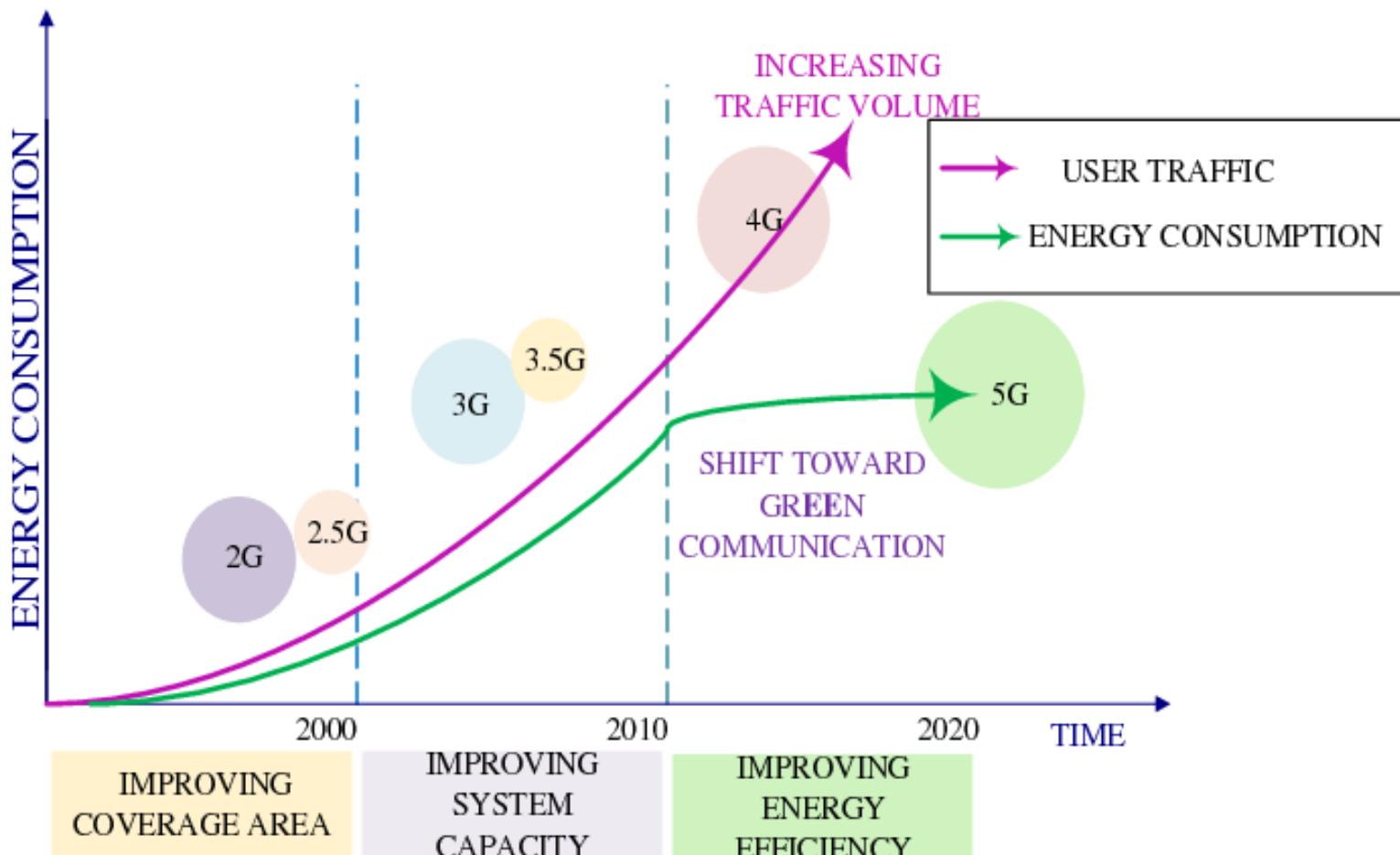


Fig 2: Shift toward Green Communication

“Power Optimization in 5G Networks: A Step Towards GrEEn Communication”. A. ABROL, R. K.JHA. IEEE Access 2016
<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7448820>

RETI RADIO (RF LINKS)

- Interfacce semplici di comunicazione via radio
 - connessione con microcontrollori e dispositivi mediante la seriale
- Caratteristiche:
 - Data rate: fino ad 1 Mbps
 - Range: fino ad 1 Km, a seconda dell'antenna
 - Frequenza: 2.4 GHz
- Non supportano protocolli di alto livello come TCP/IP
 - bisogna implementare il proprio protocollo

BLUETOOTH (BT)

- Standard tecnico-industriale di trasmissione dati per reti personali senza fili (WPAN: Wireless Personal Area Network) a corto raggio
 - IEEE 802.15.1
- Si basa sulla scoperta dinamica di dispositivi coperti dal segnale radio entro un raggio di qualche decina di metri mettendoli in comunicazione tra loro
 - esempio di dispositivi: palmari, telefoni cellulari, personal computer, portatili, stampanti, fotocamere digitali, console..
- La specifica originariamente sviluppata da Ericsson (1994) e in seguito formalizzata dalla Bluetooth Special Interest Group (SIG).
 - formata da Sony Ericsson, IBM, Intel, Toshiba, Nokia e altri



BT: INFORMAZIONI GENERALI

- Standard progettato con l'obiettivo primario di ottenere bassi consumi, un corto raggio d'azione (fino a 100 metri di copertura per un dispositivo di Classe 1 e fino ad un metro per dispositivi di Classe 3) e un basso costo di produzione per i dispositivi compatibili.
- Lavora nelle frequenze libere di 2,45 GHz.
 - per ridurre le interferenze il protocollo divide la banda in 79 canali e provvede a commutare tra i vari canali 1.600 volte al secondo (frequency hopping).
- Velocità
 - la versione 1.1 e 1.2 del protocollo gestisce velocità di trasferimento fino a 723,1 kbit/s.
 - la versione 2.0 gestisce una modalità ad alta velocità che consente fino a 3 Mbit/s.
 - questa modalità però aumenta la potenza assorbita.
- La nuova versione utilizza segnali più brevi, e quindi riesce a dimezzare la potenza richiesta rispetto al Bluetooth 1.2 (a parità di traffico inviato)

CLASSI

- I dispositivi BT si dividono in 4 classi, a seconda della potenza consumata e del raggio di funzionamento

Classe	Potenza ERP		Distanza
	(mW)	(dBm)	(m)
1	100	20	~100
2	2,5	4	~10
3	1	0	~1
4	0,5	-3	~0,5

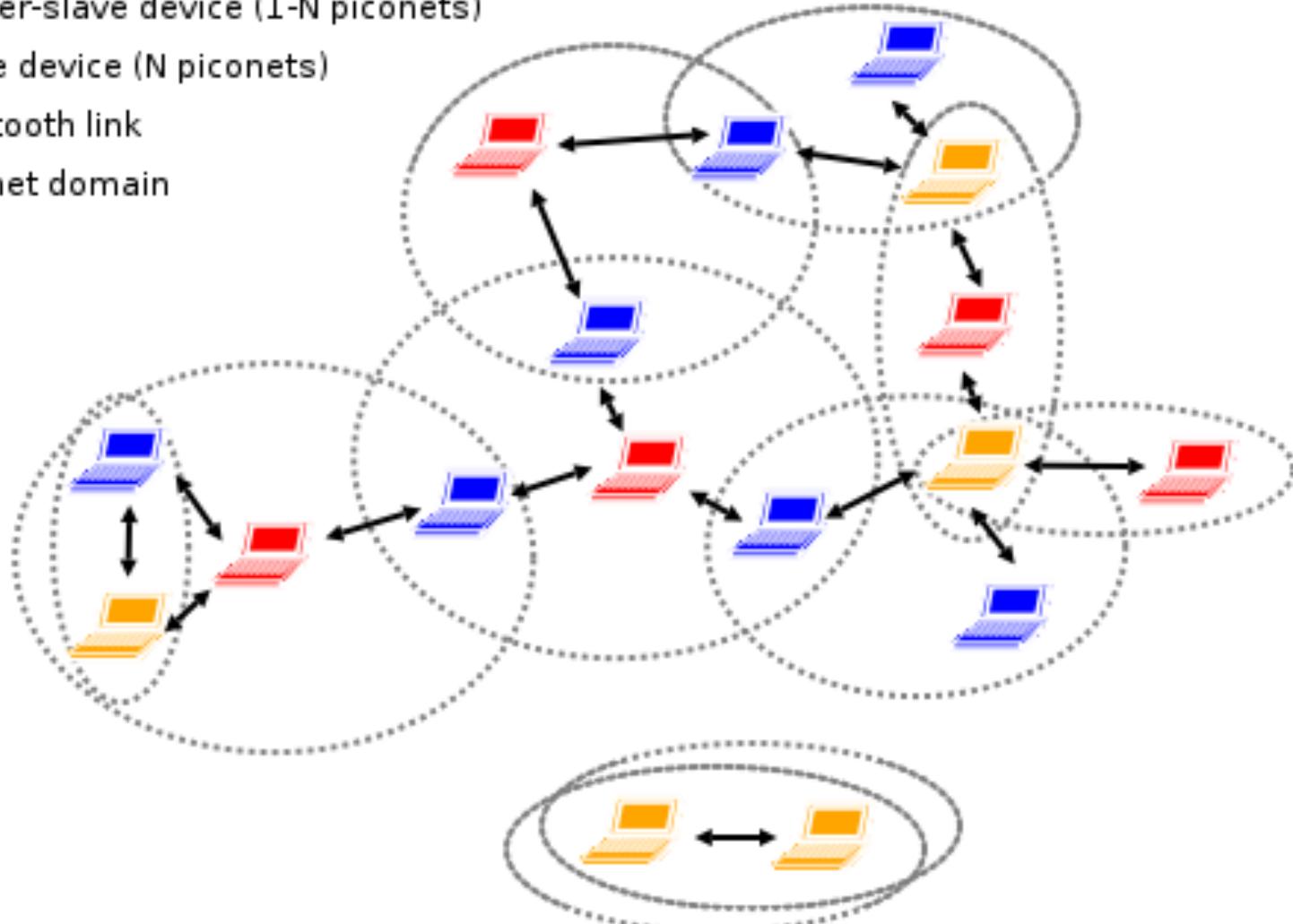
(Watt = Joule/secondo, mW = milliWatt = 10^{-3} Watt)
(dB = decibel = $10 \cdot \log_{10}$ valore)

TIPOLOGIE DI RETE

- La rete di base BT è chiamata **piconet**, basata su architettura **master-slave**
 - ogni dispositivo Bluetooth è in grado di gestire simultaneamente la comunicazione con altri 7 dispositivi slave
 - un dispositivo per volta può comunicare con il master
 - la comunicazione è sincronizzata con il clock del master
- E' possibile connettere tra loro piconet a formare una **scatternet**.
 - gli slave possono appartenere a più piconet contemporaneamente mentre il master di una piconet può al massimo essere lo slave di un'altra.
- Ogni dispositivo Bluetooth è configurabile per cercare costantemente altri dispositivi e per collegarsi a questi
 - può essere impostata una password per motivi di sicurezza

BT PICONET E SCATTERNET

-  Master device (1 piconet)
-  Master-slave device (1-N piconets)
-  Slave device (N piconets)
-  Bluetooth link
-  Piconet domain

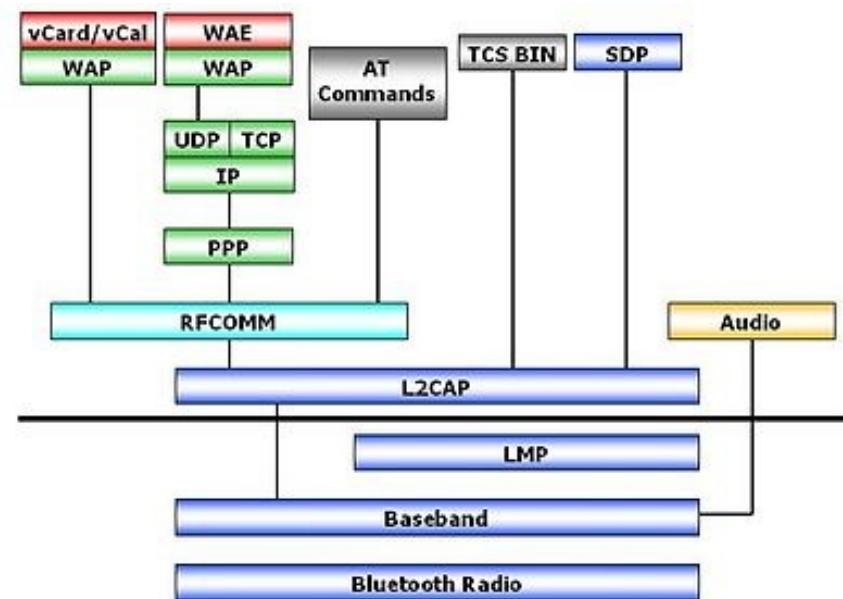
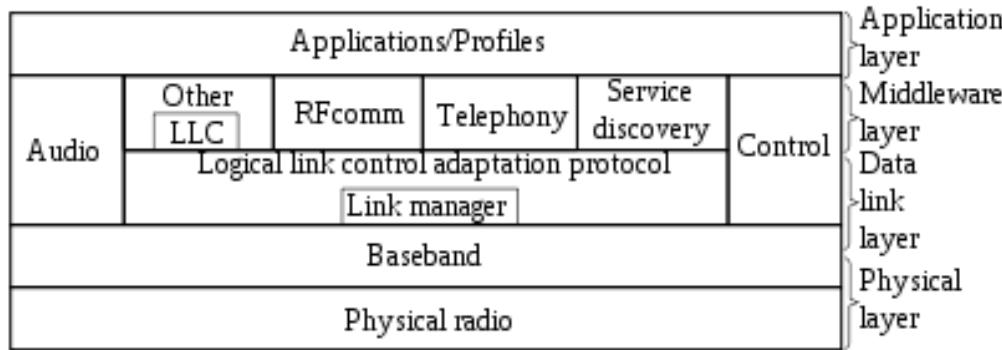


TIPI DI CONNESSIONI

- In generale i collegamenti che possono essere stabiliti tra i diversi dispositivi sono di due tipi: *orientati alla connessione* e *senza connessione*.
 - un collegamento orientato alla connessione richiede di stabilire una connessione tra i dispositivi prima di inviare i dati;
 - un link senza connessione non richiede alcuna connessione prima di inviare i pacchetti - il trasmettitore può in qualsiasi momento iniziare ad inviare i propri pacchetti purché conosca l'indirizzo del destinatario
- La tecnologia Bluetooth definisce due tipi di collegamenti a supporto delle applicazioni voce e trasferimento dati:
 - un servizio asincrono senza connessione (ACL, Asynchronous ConnectionLess)
 - per traffico di tipo dati, servizio di tipo best-effort
 - un servizio sincrono orientato alla connessione (SCO, Synchronous Connection Oriented).
 - traffico di tipo real-time e multimediale.

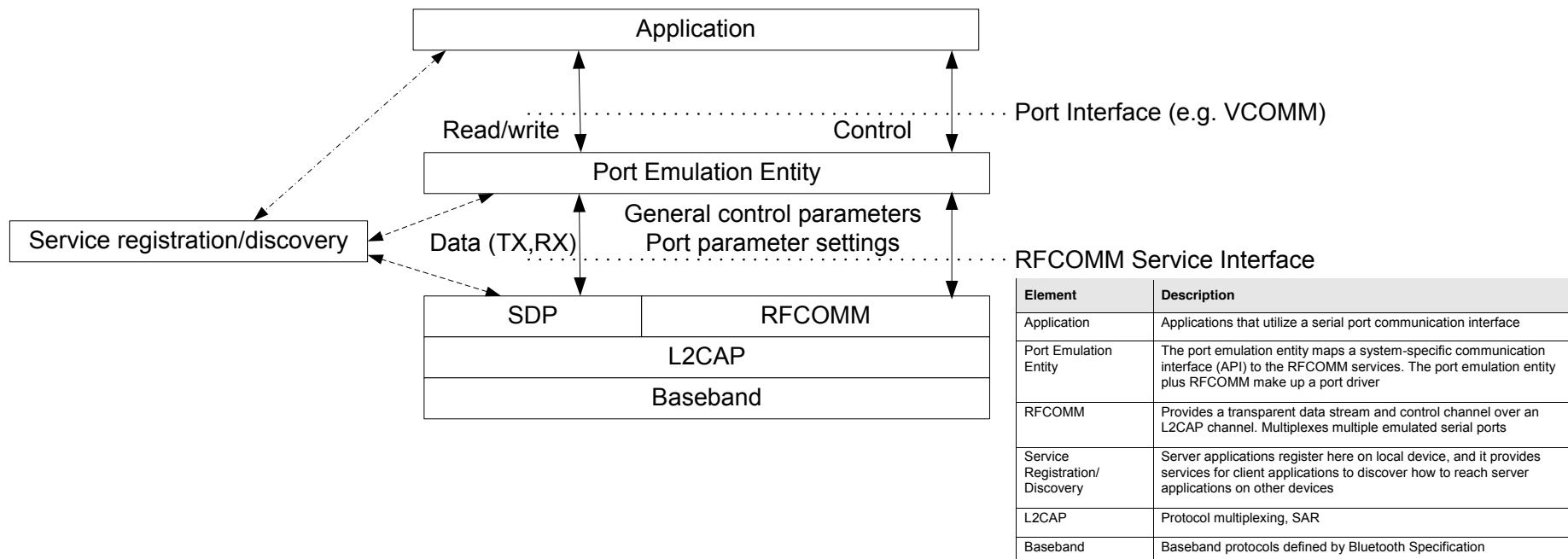
BT PROTOCOL STACK

- Stack di protocolli, organizzato a livelli come avviene per architettura ISO/OSI
 - differenti protocolli sono utilizzati per differenti applicazioni.
 - Indipendentemente del tipo di applicazione, o stack include sempre livelli data-link e fisico.



RFCOMM

- Il protocollo RFCOMM è utilizzato per sfruttare Bluetooth come una classica linea di comunicazione seriale, emulando una tradizionale porta seriale RS-232
 - utile per interfacciarsi e interagire con dispositivi come stampanti, model, PC, laptop
- Reference model
 - <https://developer.bluetooth.org/TechnologyOverview/Pages/RFCOMM.aspx>



ADOPTED PROTOCOLS

- Protocolli definiti da altre organizzazioni di standardizzazione e incorporati nell'architettura Bluetooth);
 - PPP
 - lo standard Internet per trasportare i pacchetti IP su una connessione punto a punto
 - TCP/UDP-IP
 - OBEX
 - object exchange, un protocollo a livello sessione sviluppato dalla Infrared Data Association per scambio di oggetti, simile all'HTTP ma più semplice
 - usato ad esempio per trasferire dati in formato vCard e vCalendar, cioè biglietto da visita e calendario degli impegni
 - WAE/WAP
 - Wireless Application Environment e Wireless Application Protocol

PROFILI

- Per facilitare l'utilizzo dei dispositivi Bluetooth sono stati definiti una serie di profili, che identificano una serie di possibili applicazioni.
- Esempi di profili:
 - Generic Access Profile (GAP)
 - Service Discovery Application Profile (SDAP)
 - Cordless Telephony Profile (CTP)
 - Intercom Profile (IP)
 - Serial Port Profile (SPP)
 - Headset Profile (HSP)
 - Dial-up Networking Profile (DUNP)
 - Fax Profile
 - LAN Access Profile (LAP)
 - Generic Object Exchange Profile (GOEP)
 - Object Push Profile (OPP)
 - File Transfer Profile (FTP)
 - Synchronisation Profile (SP)
 - ...

SICUREZZA

- Utilizza l'algoritmo SAFER+ (Secure And Fast Encryption Routine) per autenticare i dispositivi e per generare la chiave utilizzata per cifrare i dati

EVOLUZIONE DELLO STANDARD

- Dalla 1.0 (1999) alla versione 5 (2016)
- La versione 4.0 detta **Bluetooth Smart** include i protocolli:
 - Classic Bluetooth - legacy BT
 - Bluetooth High Speed - basato su Wi-Fi
 - **Bluetooth Low Energy (BLE)** - protocollo alternativo a quello BT classico, pensato appositamente per IoT
 - consumo di energia molto ridotto
 - facilità di creazione di link di comunicazione fra device
 - reattività
- Due implementazioni
 - *single-mode*
 - solo lo stack BLE è implementato
 - *dual-mode*
 - le funzionalità di BT Smart sono integrate a quelle Classic

CLASSIC BT vs. BT SMART

Technical Specification	Classic Bluetooth technology	Bluetooth Smart technology
Distance/Range (theoretical max.)	100 m (330 ft)	>100 m (>330 ft)
Over the air data rate	1–3 Mbit/s	1 Mbit/s
Application throughput	0.7–2.1 Mbit/s	0.27 Mbit/s
Active slaves	7	Not defined; implementation dependent
Security	56/128-bit and application layer user defined	128-bit AES with Counter Mode CBC-MAC and application layer user defined
Robustness	Adaptive fast frequency hopping, FEC , fast ACK	Adaptive frequency hopping, Lazy Acknowledgement, 24-bit CRC, 32-bit Message Integrity Check
Latency (from a non-connected state)	Typically 100 ms	6 ms
Minimum total time to send data (det.battery life)	100 ms	3 ms [31]
Voice capable	Yes	No
Network topology	Scatternet	Scatternet
Power consumption	1 W as the reference	0.01 to 0.5 W (depending on use case)
Peak current consumption	<30 mA	<15 mA
Service discovery	Yes	Yes
Profile concept	Yes	Yes
Primary use cases	Mobile phones, gaming, headsets, stereo audio streaming, smart homes, wearables, automotive, PCs, security, proximity, healthcare, sports & fitness, etc.	Mobile phones, gaming, smart homes, wearables, automotive, PCs, security, proximity, healthcare, sports & fitness, Industrial, etc.

BLUETOOTH 5

- Disponibile da Dicembre 2016, **evoluzione verso IoT**
 - 4x area di trasmissione
 - ma stesso livello di consumo p
 - 2x velocità fino a 2Mbps in modalità a basso consumo
 - riduzione dei tempi necessari per ricevere e trasmettere i dati.
 - 8x capacità di trasmissione
 - per un minor tempo richiesto per il completamento dei task
 - adatto per l'adozione e gestione dei beacon.

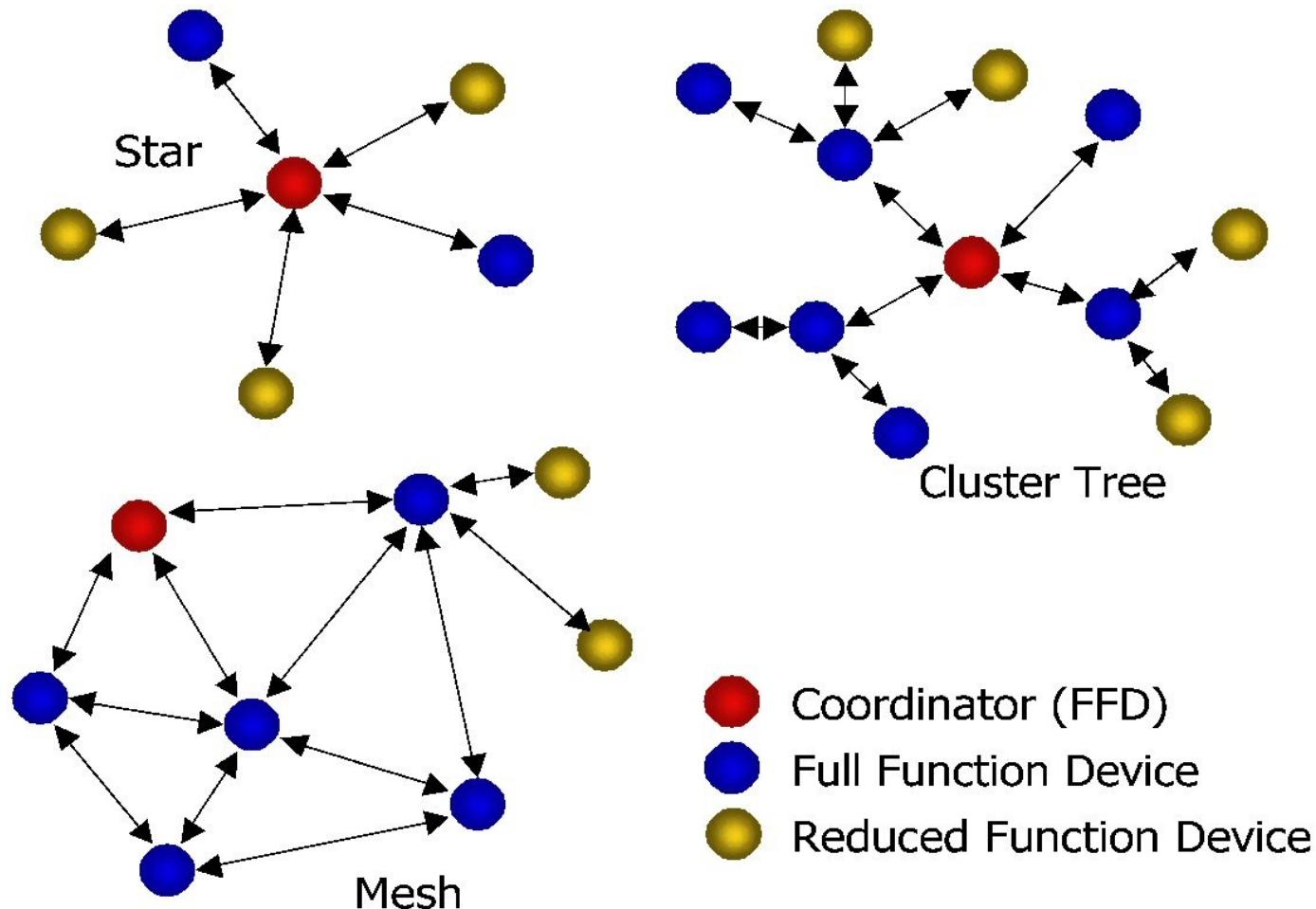
LO STANDARD IEEE 802.15.4

- Standard che definisce le specifiche a livello di physical layer e media access control per reti wireless "personal" ubique a bassa velocità, basso costo (*low-rate wireless personal area networks, LR-WPANs*)
 - l'enfasi è sulla comunicazione a bassi consumi e agile fra dispositivi vicini, a corto raggio
 - comunicazione entro i 10 metri, con transfer rate di 250 kbit/s
 - supporti per la comunicazione real-time e comunicazioni sicure
- E' la base di ulteriori specifiche che includono anche i livelli sopra
 - **ZigBee**, ISA100.11a, WirelessHART, MiWi
- Usi tipici
 - Home and building automation
 - Automotive networks
 - Industrial wireless sensor networks
 - Interactive toys and remote controls

MODELLO DI RETE

- Due tipi di nodi
 - **FFD** (full-function device).
 - funge da **coordinatore** della PAN
 - implementa un modello di comunicazione che gli permette di parlare con un qualsiasi altro nodo
 - **RFD** (reduced-function devices).
 - questi nodi rappresentano semplici dispositivi con modeste risorse di elaborazione e comunicazione
 - possono comunicare solo con un FFD
- Topologie
 - tipologie **star** e **peer-to-peer**
 - ogni rete deve avere almeno un FFD che funge da coordinatore
 - cluster tree
 - struttura in cui un RFD può avere un solo FFD
- le reti p2P possono formare pattern arbitrari di connessioni
 - sono la base per formare reti ad hoc in grado di fare self-management and organization.

MODELLO DI RETE



CARATTERISTICHE

- Frequenza
 - opera nelle bande libere 2.4 GHz, 900 MHz e 868 MHz
- Data rate
 - può arrivare ad un massimo di 250 kbps
- Range di comunicazione
 - da 100 m a 1 Km, dipendentemente dalla potenza in uscita utilizzata
 - (piccoli progetti: 1mW => 100 m)
- Consumo
 - 50mA, per cui una batteria ricaricabile di 850 mAH può fornire circa 17 ore di uso continuato

IEEE 802.15.4: IMPLEMENTAZIONI

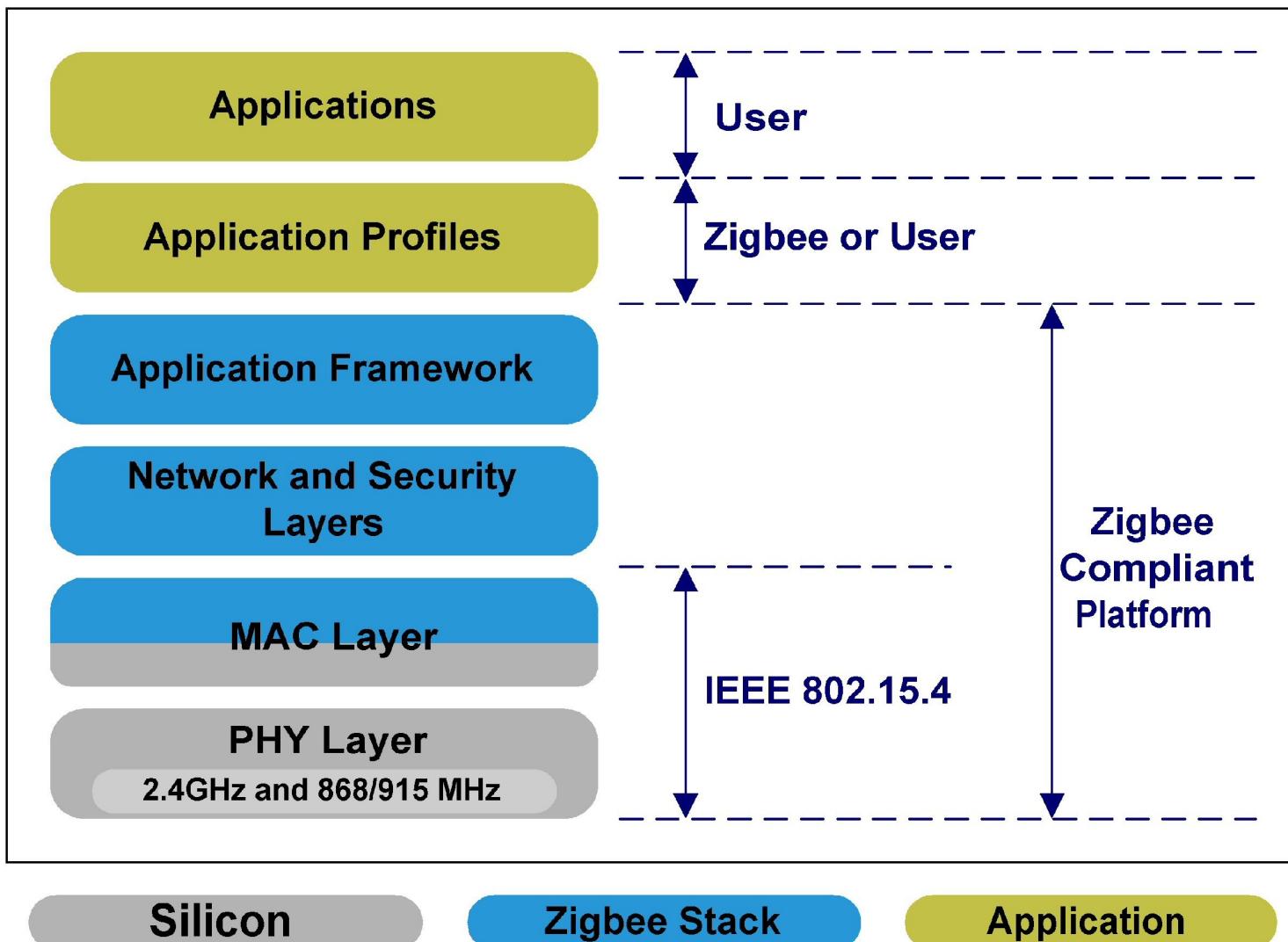
- Esempi
 - ZigBee
 - 6LoWPAN
 - Thread

Protocol	Description
ZigBee	Promoted through the ZigBee Alliance, ZigBee defines upper-layer components (network through application) as well as application profiles. Common profiles include building automation, home automation, and healthcare. ZigBee also defines device object functions, such as device role, device discovery, network join, and security. For more information on ZigBee, see the ZigBee Alliance webpage, at www.zigbee.org . ZigBee is also discussed in more detail later in the next Section.
6LoWPAN	6LoWPAN is an IPv6 adaptation layer defined by the IETF 6LoWPAN working group that describes how to transport IPv6 packets over IEEE 802.15.4 layers. RFCs document header compression and IPv6 enhancements to cope with the specific details of IEEE 802.15.4. (For more information on 6LoWPAN, see Chapter 5.)
ZigBee IP	An evolution of the ZigBee protocol stack, ZigBee IP adopts the 6LoWPAN adaptation layer, IPv6 network layer, and RPL routing protocol. In addition, it offers improvements to IP security. ZigBee IP is discussed in more detail later in this chapter.
ISA100.11a	ISA100.11a is developed by the International Society of Automation (ISA) as “Wireless Systems for Industrial Automation: Process Control and Related Applications.” It is based on IEEE 802.15.4-2006, and specifications were published in 2010 and then as IEC 62734. The network and transport layers are based on IETF 6LoWPAN, IPv6, and UDP standards.
WirelessHART	WirelessHART, promoted by the HART Communication Foundation, is a protocol stack that offers a time-synchronized, self-organizing, and self-healing mesh architecture, leveraging IEEE 802.15.4-2006 over the 2.4 GHz frequency band. A good white paper on WirelessHART can be found at http://www.emerson.com/resource/blob/system-engineering-guidelines-iec-62591-wirelesshart--data-79900.pdf
Thread	Constructed on top of IETF 6LoWPAN/IPv6, Thread is a protocol stack for a secure and reliable mesh network to connect and control products in the home. Specifications are defined and published by the Thread Group at www.threadgroup.org .

ZIGBEE

- Fra le tecnologie wireless più recenti e utilizzate nei sistemi embedded
 - è una specifica per un insieme di protocolli di comunicazione ad alto livello che utilizzano piccole antenne digitali a bassa potenza e basato sullo standard IEEE 802.15.4 per wireless personal area networks (WPAN)
 - ideata nell'ottica IoT, per la comunicazione pervasiva fra device
 - open global standard
 - basso consumo, basso costo
- Storia
 - iniziarono ad essere studiate nel 1998, quando molti ingegneri capirono che sia WiFi che Bluetooth non riuscivano a rispondere alle necessità di molte nuove applicazioni.
 - lo standard IEEE 802.15.4 fu completato nel maggio 2003.
 - la ZigBee Alliance annuncia la disponibilità della Specification 1.0 il 13 giugno 2005.

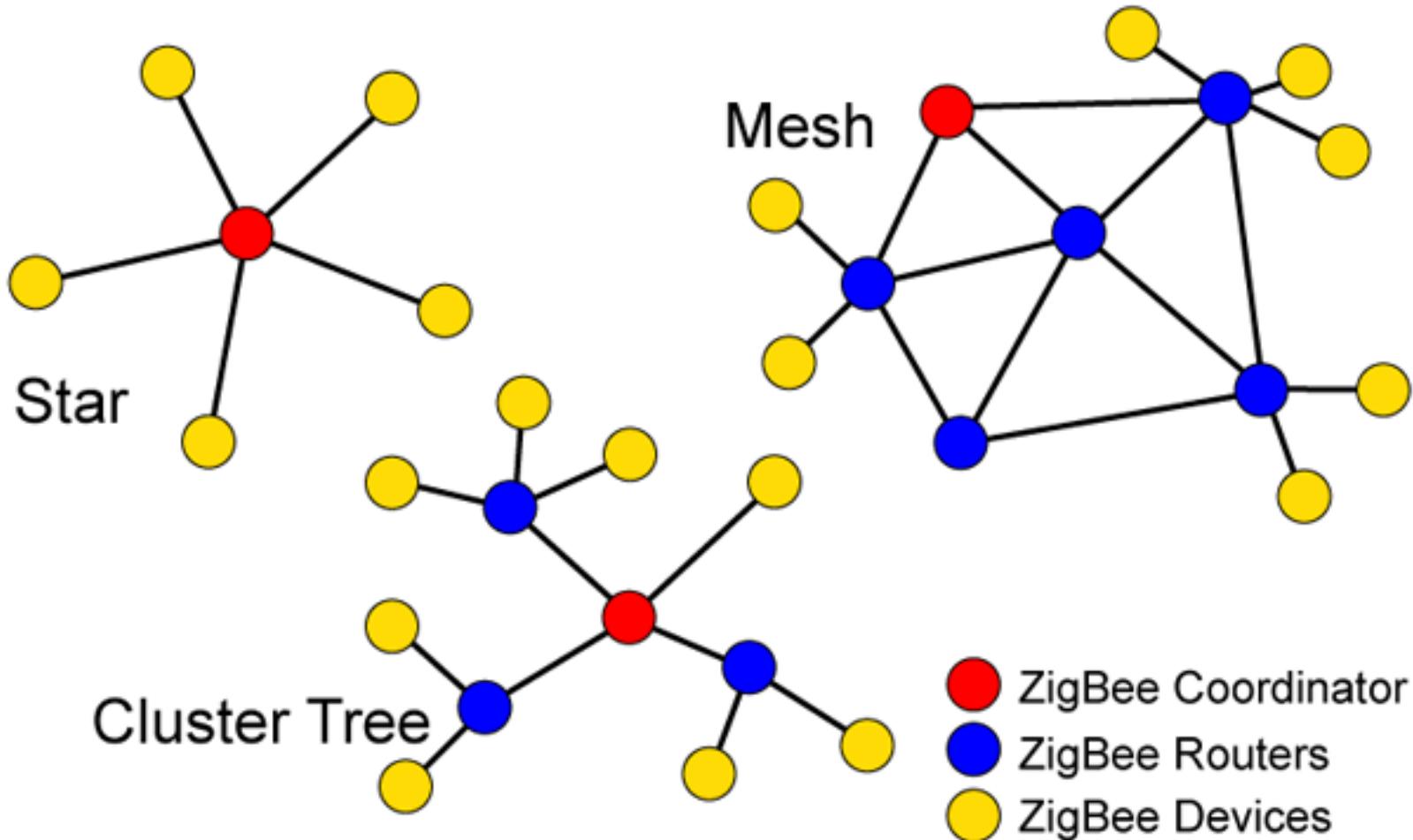
ZIGBEE STACK



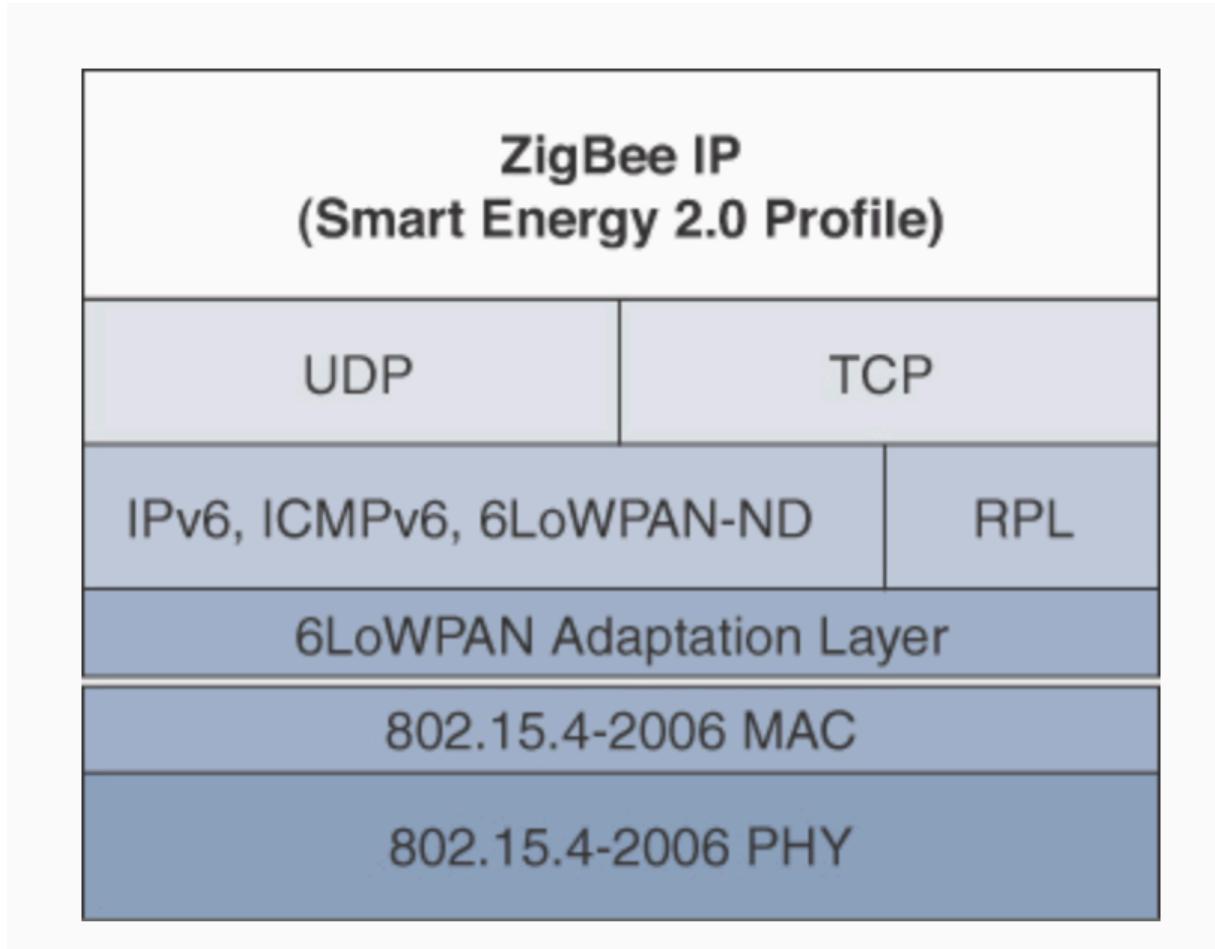
ARCHITETTURA

- Ci sono tre differenti tipi di dispositivo ZigBee:
 - **ZigBee Coordinator (ZC)**
 - è il dispositivo più "intelligente" tra quelli disponibili, costituisce la radice di una rete ZigBee e può operare da ponte tra più reti
 - ci può essere un solo Coordinator in ogni rete.
 - esso è inoltre in grado di memorizzare informazioni riguardo alla sua rete e può agire come deposito per le chiavi di sicurezza.
 - **ZigBee Router (ZR)**
 - questi dispositivi agiscono come router intermedi passando i dati da e verso altri dispositivi
 - **ZigBee End Device (ZED)**
 - includono solo le funzionalità minime per dialogare con il suo nodo parente (Coordinator o Router)
 - non possono trasmettere dati provenienti da altri dispositivi
 - sono i nodi che richiedono il minor quantitativo di memoria e quindi risultano spesso più economici rispetto ai ZR o ai ZC

TOPOLOGIE



ZIGBEE IP STACK

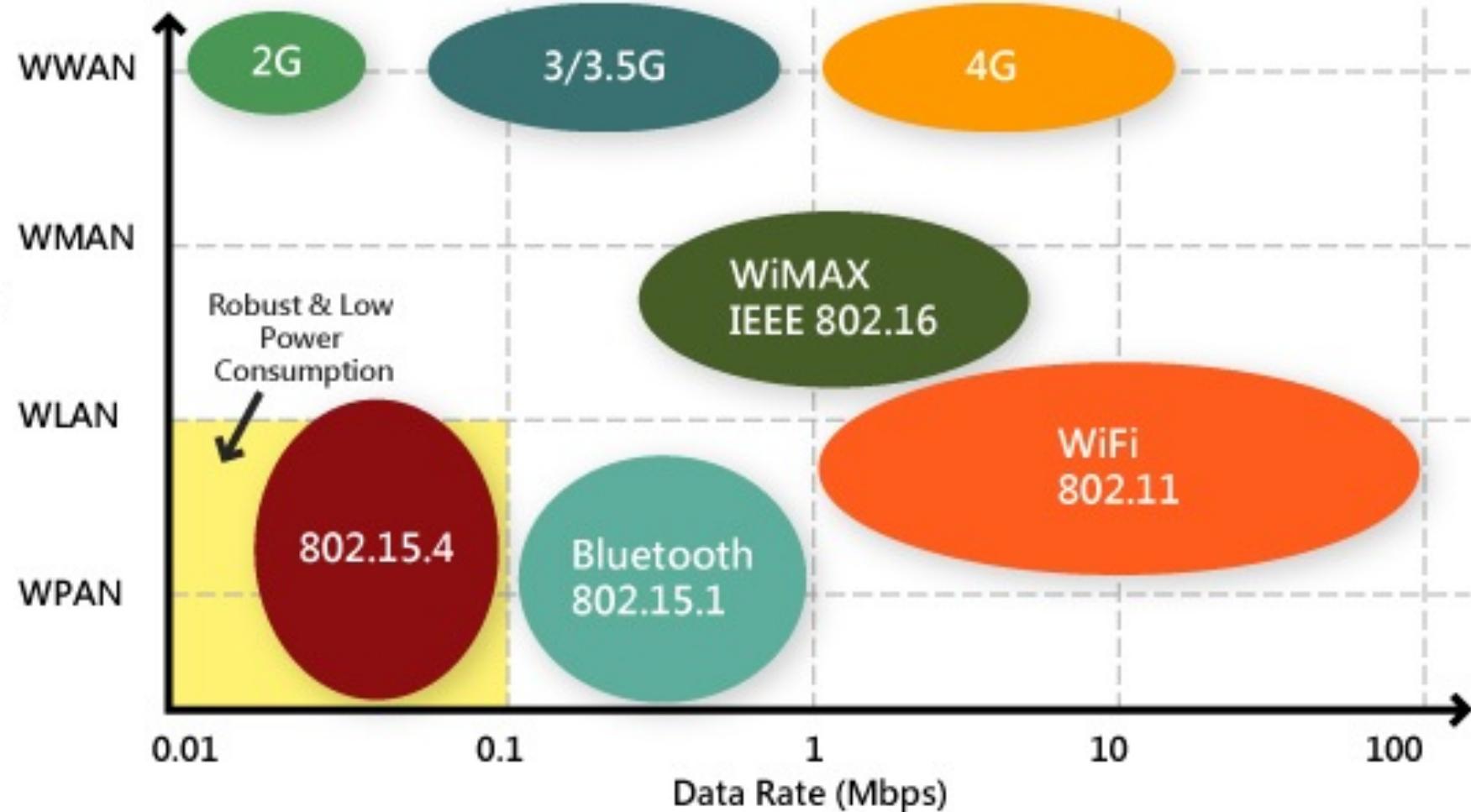


[IoTF, p. 261]

CONFRONTO ZIGBEE VS. BT

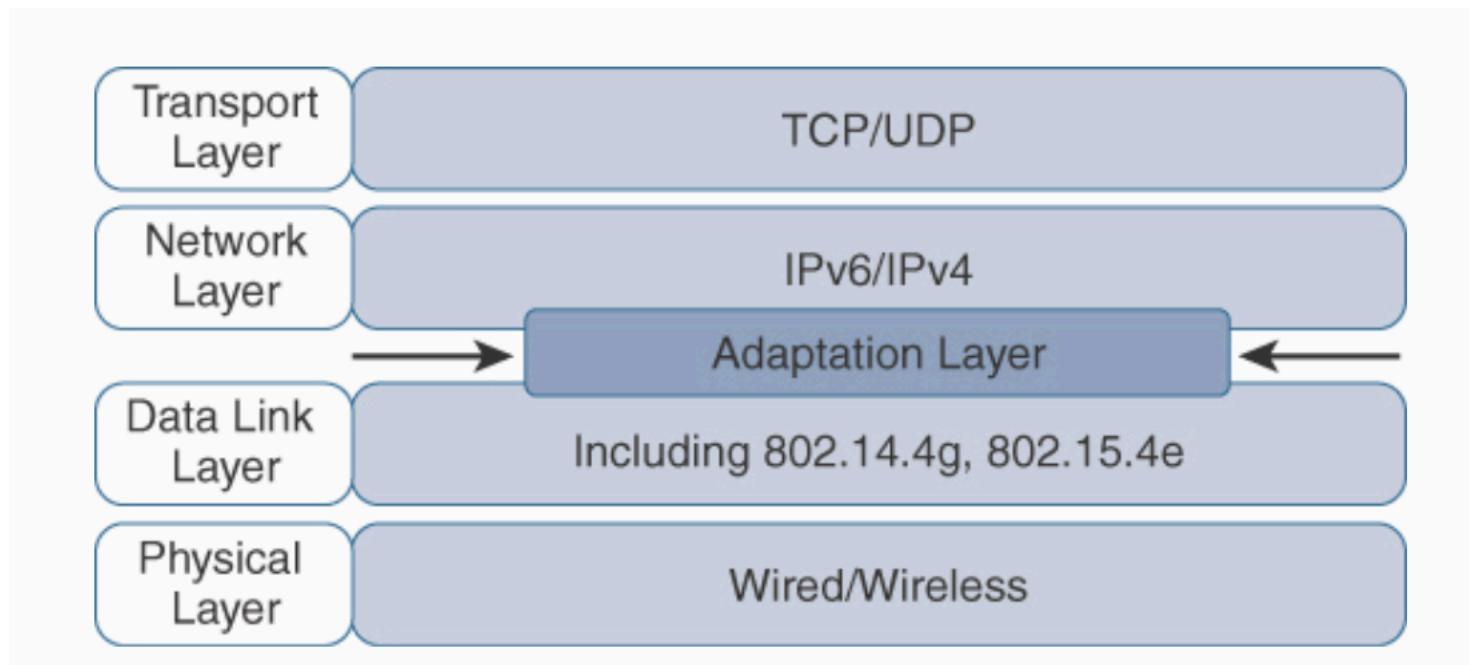
- ZigBee vs. BT
 - miglior efficienza, range di comunicazione più ampio, creazione di reti p2p più semplice e agile
 - soluzione migliore per IoT
 - tuttavia meno utilizzata a livello consumer

CONFRONTI



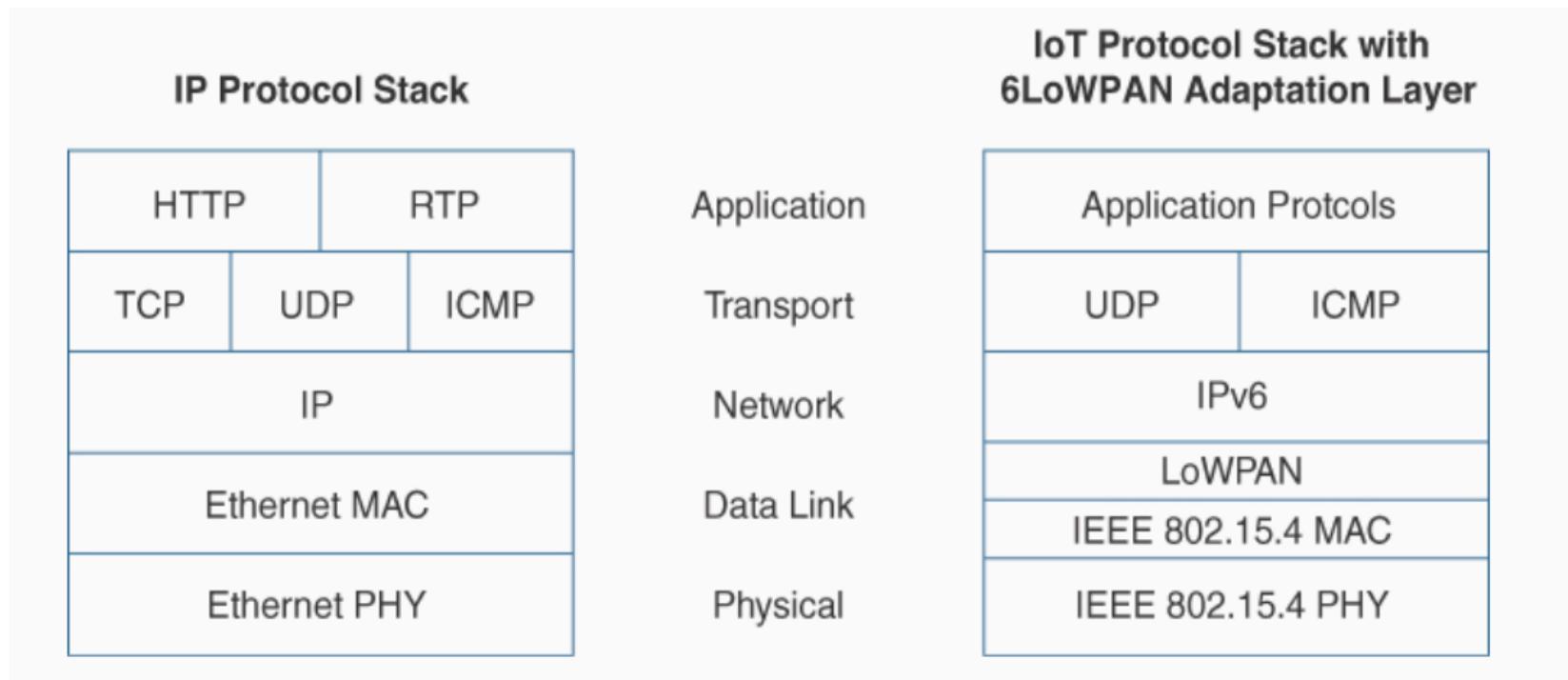
IP PER IOT - OTTIMIZZAZIONE

- Nodi/device constrained e reti constrained tipici di IoT definiscono ottimizzazioni a vari livelli e protocolli dell'architettura di rete IP



ESEMPIO 6LOWPAN

- Esempio: 6LoWPAN e 6Lo (successore di 6LoWPAN)
 - ottimizzazione pacchetti IPv6 su reti constrained come IEEE 802.15.4



PROTOCOLLI LIVELLO APPLICATIVO

- Varie scelte possibili come metodo per trasporto messaggi in applicazioni IoT
- Categorie
 - nessun protocollo applicativo specifico
 - SCADA
 - generici protocolli web-based
 - protocolli trasporto UDP/TCP
 - TCP preferibile per reti cellulari poiché sono tipicamente robuste e possono gestire bene l'overhead
 - per le reti LLN (*low-power and lossy*), in cui sia i dispositivi sia le reti sono constrained, tipicamente si usa UDP
 - protocolli applicativi pensati per IOT
 - pensati per nodi e reti constrained
 - **MQTT** (Message Queueing Telemetry Transport)
 - **CoAP** (Constrained Application Protocol)

COAP E MQTT - CARATTERISTICHE

CoAP	MQTT
UDP	TCP
IPv6	
6LoWPAN	
802.15.4 MAC	
802.15.4 PHY	

[IoTF, p. 414]

Esempio di stack
basato su CoAP vs. MQTT

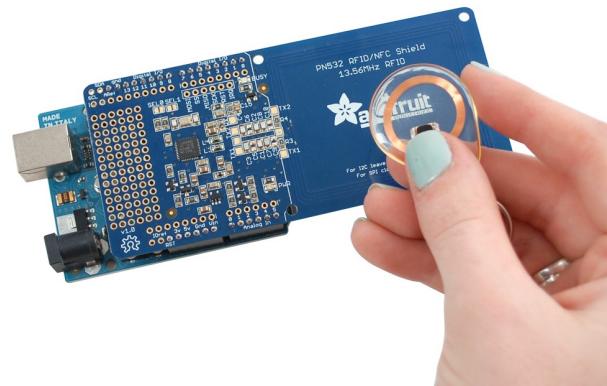
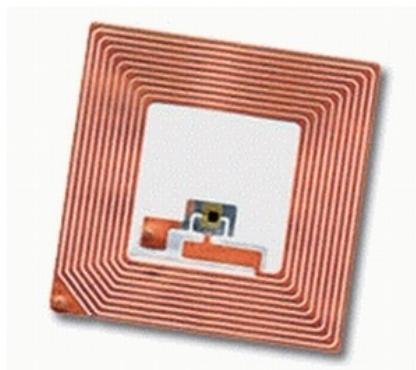
Factor	CoAP	MQTT
Main transport protocol	UDP	TCP
Typical messaging	Request/response	Publish/subscribe
Effectiveness in LLNs [*]	Excellent	Low/fair (Implementations pairing UDP with MQTT are better for LLNs.)
Security	DTLS	SSL/TLS
Communication model	One-to-one	many-to-many
Strengths	Lightweight and fast, with low overhead, and suitable for constrained networks; uses a RESTful model that is easy to code to; easy to parse and process for constrained devices; support for multicasting; asynchronous and synchronous messages	TCP and multiple QoS options provide robust communications; simple management and scalability using a broker architecture
Weaknesses	Not as reliable as TCP-based MQTT, so the application must ensure reliability.	Higher overhead for constrained devices and networks; TCP connections can drain low-power devices; no multicasting support

[*] LLN = low-power and lossy networks

[IoTF, p. 436]

TECNOLOGIE RADIO PER L'IDENTIFICAZIONE: RFID

- Radio Frequency Identification (RFID)
 - tecnologie inizialmente introdotte per l'identificazione e tracking a distanza di oggetti mediante l'ausilio di piccoli dispositivi elettronici chiamati **tag**, che possono essere letti/scritti a distanza (di cm) da appositi **RFID reader**
 - un tag RFID contiene tipicamente un ID univoco che contraddistingue l'oggetto su cui è montata l'etichetta
- E' stata una delle tecnologie chiave di IoT
 - oggi sempre più sostituiti da NFC e iBeacon



TAG RFID

- Un tag RFID può essere passivo, attivo o BAP (battery assisted passive)
 - **passivo**
 - non è alimentato. E' caratterizzato da un circuito che viene alimentato indirettamente, a distanza, dalle onde radio inviate da un RFID reader che ne vuole leggere il contenuto, che tipicamente corrisponde ad un ID univoco
 - può essere anche scritto da un RFID reader, usando lo stesso principio
 - **attivo**
 - hanno batterie proprie e trasmettono in broadcast il proprio contenuto informativo continuamente
 - **battery assisted passive (BAP)**
 - ibrido, come il caso attivo ma invia le informazioni solo se sollecitato da un RFID reader
 - supporta distanze maggiori rispetto al passivo

NFC

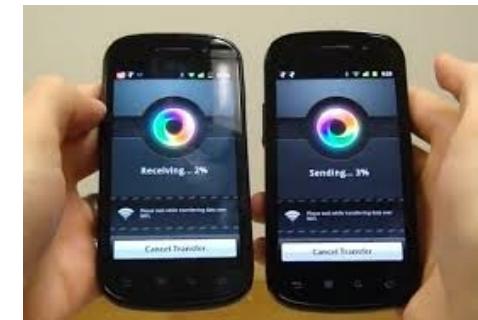
- **Near Field Communication (NFC)**
 - fornisce connettività wireless (RF) bidirezionale a corto raggio (max 10 cm)
 - sviluppata congiuntamente da Philips, LG, Sony e Nokia
- Tecnologia evoluta da una combinazione della tecnologia RFID e altre tecnologie di connettività.
 - contrariamente ai più semplici dispositivi RFID, NFC permette una comunicazione bidirezionale:
 - quando due apparecchi NFC (lo initiator e il target) vengono accostati entro un raggio di 4 cm, viene creata una rete peer-to-peer tra i due ed entrambi possono inviare e ricevere informazioni
- Il target può essere dato da semplici **tag**, come nel caso di RFID
- Diffusione pervasiva in smartphone e dispositivi vari
 - smartphone come NFC reader



SEIOT ISI-LT - UNIBO



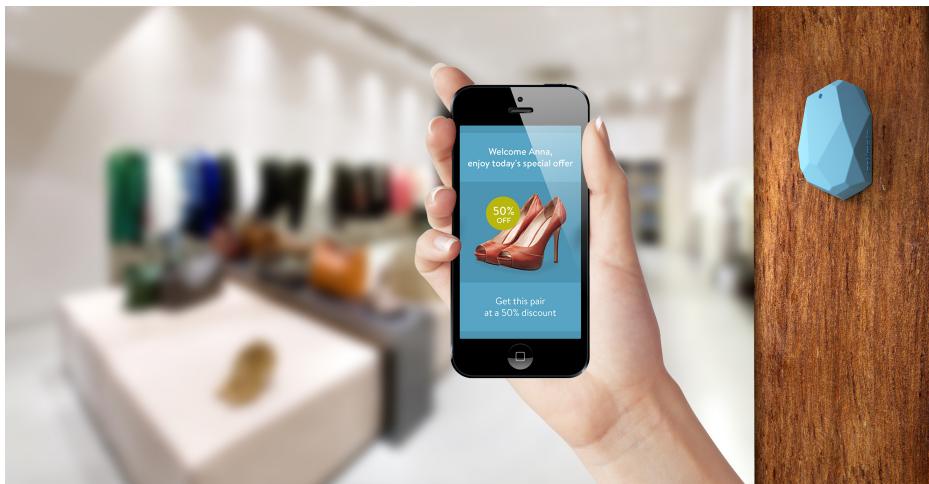
COMUNICAZIONE



58

BEACON

- Tecnologie basate su BLE introdotte recentemente per fare localizzazione/identificazione wireless tipicamente in ambienti indoor
 - trademark di Apple che l'ha proposto come indoor proximity system
- Un iBeacon è un piccolo trasmettitore a bassissimo consumo che emette continuamente un segnale - contenente il suo ID - che può essere rilevato da un qualsiasi dispositivo BLE
- Impatto notevole nel panorama IoT



BIBLIOGRAFIA E SITOGRAFIA

- **[IoTF]** *IoT Fundamentals*. Hanes et al. CISCO Press. 2017
- *Building Internet of Things with Arduino*, Charalampos Doukas

Sistemi Embedded e IoT

Ingegneria e Scienze Informatiche - UNIBO

a.a 2020/2021

Docente: Prof. Alessandro Ricci

[modulo-4.3]

MODELLI DI COMUNICAZIONE
A SCAMBIO DI MESSAGGI

SOMMARIO

- In questo modulo si discutono i modelli di comunicazione a scambio di messaggi, considerandone l'integrazione nelle architetture/modelli di controllo visti nella programmazione di sistemi embedded delle scorse lezioni

MODELLI DI COMUNICAZIONE

- I protocolli e tecnologie viste nei moduli 4 del corso abilitano fisicamente la comunicazione in e fra sistemi embedded
- Al di là di queste tecnologie, la progettazione e programmazione di sistemi software embedded di rete e distribuiti richiede l'introduzione di opportuni **modelli e meccanismi di comunicazione di alto livello**, e i relativi supporti tecnologici a livello di librerie / piattaforme / infrastrutture
- Tra le questioni
 - quale modello di comunicazione?
 - es: a scambio di messaggi? Sincrono, asincrono?
 - quali primitive? Quale semantica relativa agli atti comunicativi?
 - quale linguaggio di comunicazione, ovvero: quali modelli e linguaggi scegliamo per rappresentare le informazioni scambiate?
 - problema dell'interoperabilità
 - In che modo integrare il modello di comunicazione con il modello usato per descrivere il comportamento del controllore?
 - Come rappresentare e implementare protocolli di comunicazione articolati?

MODELLO A SCAMBIO DI MESSAGGI

- Comunicazione mediante invio e ricezione di messaggi
 - modello alternativo alla comunicazione mediante memoria condivisa
 - modello di riferimento per i sistemi distribuiti
- Primitive
 - **send**
 - **receive**
- Varie tipologie e aspetti
 - diretta o indiretta
 - sincrona o asincrona
 - linguaggio descrizione messaggi
 - interoperabilità

DIRETTA VS. INDIRETTA

- Comunicazione **diretta**:
 - la comunicazione avviene direttamente fra i processi (thread,...), specificando il loro identificatore
 - aspetto importante: come identificare partecipanti alla comunicazione in modo univoco
 - primitive:
 - send (ProcId,Msg)
 - receive(): Msg
- Comunicazione **indiretta**
 - si utilizzano dei *canali* che fungono da mezzi di comunicazione indiretta.
 - L'invio e la ricezione avvengono specificando uno specifico canale
 - send(ChannelId,Msg)
 - receive(ChannelId):Msg

SINCRONA VS. ASINCRONA

- Comunicazione **sincrona**:
 - la send ha successo (e completa) quando il messaggio specificato è ricevuto dal destinatario mediante una receive
- Comunicazione **asincrona**
 - la send ha successo (e completa) quando il messaggio è stato inviato
 - ma non necessariamente ricevuto dal destinatario mediante una receive

BUFFERIZZAZIONE

- Un altro aspetto importante è la strategia di bufferizzazione adottata, fondamentale nel caso asincrono
 - sia nel caso indiretto, per i canali
 - sia nel caso diretto, per la coda che i processi usano per ricevere i messaggi
- Concerne la dimensione del buffer ove vengono memorizzati i messaggi ricevuti, ma ancora da ricevere con la receive

PRIMITIVE: VARI TIPI DI SEMANTICA

- **send (DestId,Msg)**
 - ha successo quando (semantiche alternative):
 - (1) il msg è stato inviato (=> caso **asincrono**)
 - in questo caso la verifica che un messaggio sia arrivato a destinazione deve essere implementata a livello di protocollo
 - errore se il destinatario non esiste
 - (2) il msg è arrivato, ma non necessariamente ricevuto
 - errore se il destinatario non esiste
 - (3) il msg è stato ricevuto
 - caso sincrono
- **receive():Msg**
 - semantica usuale: **bloccante**
 - si blocca sin quando non è disponibile almeno un messaggio

BUFFER FULL - CHE FARE?

- Nel caso asincrono, deve essere considerato anche il caso in cui il buffer di ricezione dei messaggi si riempia
- In quel caso occorre scegliere il tipo di semantica per la send
 - fallisce
 - msg scartato
 - si blocca in attesa che il buffer non sia pieno
 - stile produttori/consumatori
 - ha successo
 - riscritto messaggio del buffer
 - più vecchio, più nuovo,...

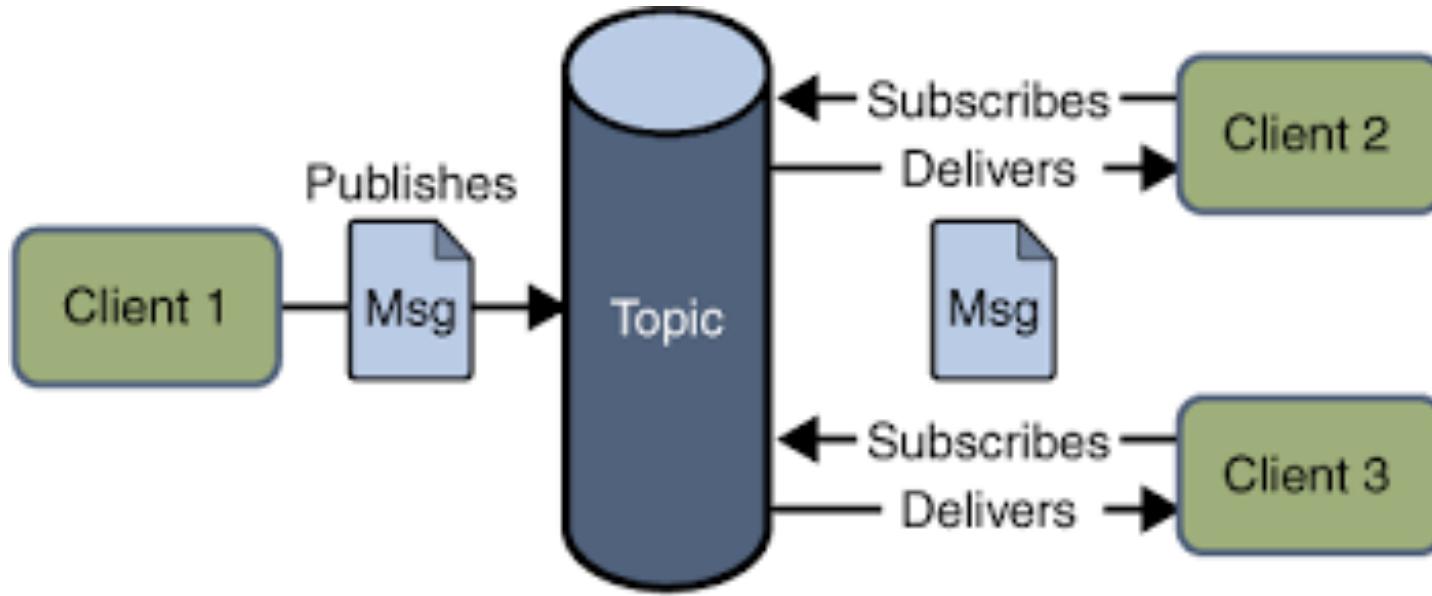
RAPPRESENTAZIONE MESSAGGI

- Aspetto fondamentale della comunicazione è dato dal “linguaggio” con cui si rappresentano i messaggi
 - deve essere il medesimo utilizzato da tutti i partecipanti alla comunicazione, come primo requisito per avere **interoperabilità**
- Esempi di linguaggi usati per la rappresentazione di messaggi per avere interoperabilità:
 - **JSON, XML**
- L'utilizzo del medesimo linguaggio non è sufficiente per avere interoperabilità
- Occorre mettersi d'accordo su
 - formato messaggi / vocabolario utilizzato
 - *ontologie*
 - in caso di protocolli => semantica dei protocolli

MODELLO PUBLISH/SUBSCRIBE

- Modello a scambio di messaggi tipicamente usato per realizzare architetture ad eventi ad alto livello in sistemi distribuiti
 - simile concettualmente al *pattern observer*
 - utile per la comunicazione in sistemi *open*, *dynamic*, *loosely-coupled*
 - molto usato in ambito IoT
 - prossimo modulo
- Meta-modello
 - **channels/topics**
 - canali/argomenti sui quali è possibile inviare messaggi e che è possibile “osservare” mediante sottoscrizione
 - **publishers**
 - inviano messaggi su canali
 - **subscribers**
 - si registrano su canale/topic per ricevere tutti messaggi pubblicati su quel canale/topic
- Primitive (astratte)
 - publish, subscribe

MODELLO PUBLIC-SUBSCRIBE



MODELLI DI COMUNICAZIONE E ARCHITETTURE DI CONTROLLO

- Aspetto importante
 - integrazione modelli di comunicazione con i modelli e architetture software di controllo visti per i sistemi embedded
 - ovvero:
 - loop di controllo semplici
 - macchine a stati finiti
 - task
 - event-loop

INTEGRAZIONE NEI SISTEMI EMBEDDED ORGANIZZATI A LOOP

- Problema: receive *bloccante*
 - comporta il blocco del loop
 - non è usabile nell'implementazione di macchine a stati finiti (sincrone o asincrone)
- Esempio semplice:
 - sistema blinking che è possibile controllare dall'esterno con invio di messaggio stop
 - seiot.modulo_lab_3_2.blinker_with_msg
 - altri esempi nel materiale
 - seiot.modulo_lab_3_2.pingpong
 - seiot.modulo_lab_2_2.msg (PingPong via seriale)

ESTENSIONE DEL MODELLO

- **Receive non bloccante:**
 - receiveMsg(): { Msg, errore/null }
 - nel caso in cui non ci siano messaggi, la receive non si blocca
 - restituisce errore o genera eccezione o restituisce un riferimento null
- Aggiunta di una primitiva per testare presenza messaggi disponibili:
 - **isMsgAvailable()**: boolean

INTEGRAZIONE NEL MODELLO FSM

- Sfruttando l'estensione, l'integrazione nel modello a stati può avvenire considerando che
 - una transizione può essere scatenata dalla presenza di un messaggio => uso della primitiva di test messaggi nelle guardie
 - nell'insieme delle azioni associate alla transizione o allo stato c'è anche la ricezione (non bloccante) e l'invio di messaggi
- NOTA:
 - nelle guardie non può essere modificato lo stato: sono solo predicati..
 - quindi si può usare `isMsgAvailable`, non `receiveMsg...`

INTEGRAZIONE NEL MODELLO A TASK

- Nel modello a task lo scambio di messaggi può essere usato come modello di comunicazione fra task stessi, in alternativa o a complemento del modello basato su memoria condivisa

INTEGRAZIONE NEL MODELLO AD EVENT LOOP

- Anche nel caso di architetture di event-loop, il modello di comunicazione da considerare è necessariamente asincrono con receive *implicita* oppure non bloccante
 - questo poiché negli event handler non è possibile eseguire primitive bloccanti
- Integrazione
 - l'arrivo di un messaggio è modellato come evento
 - la coda di eventi è usata come coda di messaggi
 - l'invio di un messaggio tramite send ha come effetto accodare un evento nella coda degli eventi del destinatario

ESEMPIO

- Nel materiale è fornito un esempio in cui un agente con architettura di controllo event-loop scambia messaggi via Seriale / Bluetooth mediante un componente MsgService implementato come risorsa osservabile
 - package seiot.modulo_lab_4_3
 - MsgService, MsgEvent, MyAgent