Architettura di un elaboratore

Architettura di un elaboratore

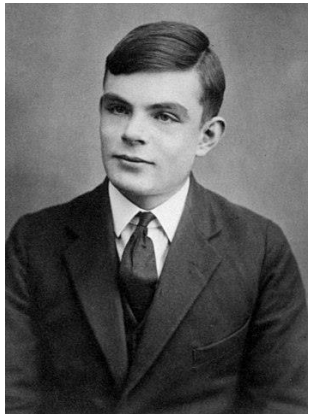
1936 – Turing, A.: «*On Computable Numbers, with an Application to the Entscheidungsproblem*»

- Calcolatore astratto «programmabile», in grado di eseguire qualunque funzione computabile. L'input è costituito non solo dai dati, ma anche dalla sequenza di operazioni da eseguire...

1945 – Nell'ambito dello sviluppo di calcolatori per il calcolo delle traiettorie di missili balistici e del Progetto Manhattan

- ENIAC: uno dei primi elaboratori digitali programmabili. Cambiare programma significa cablare nuovamente e riconfigurare la macchina
- EDVAC: il primo elaboratore digitale programmabile tramite software

Alan Turing 1912-1954



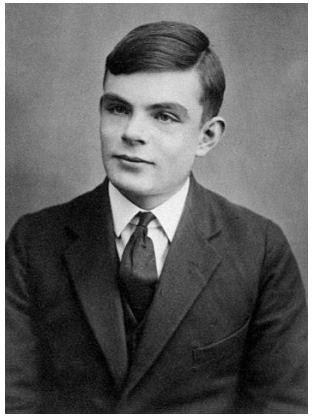
John Von Neumann 1903-1957

Architettura di un elaboratore

1945 – Von Neumann, J.: «*First Draft of a Report on the EDVAC*»

- John Von Neumann descrive l'architettura dell'EDVAC, cioè l'architettura base di un elaboratore digitale programmabile. Diventerà nota come architettura di Von Neumann

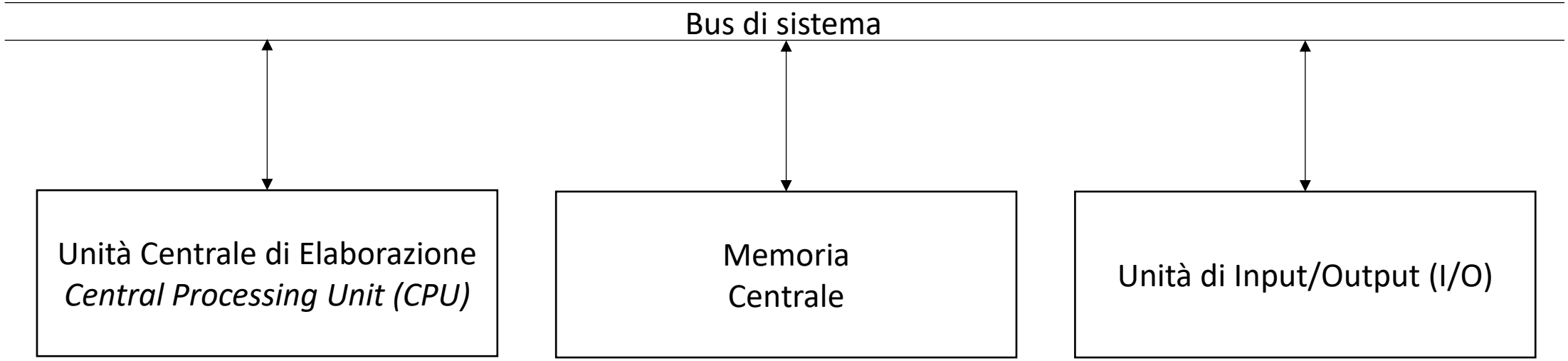
Alan Turing 1912-1954



John Von Neumann 1903-1957

Architettura di Von Neumann

Componenti fondamentali



Architettura di Von Neumann – CPU

Dirige l'esecuzione delle istruzioni. Specifiche della particolare famiglia di CPU, le istruzioni ne costituiscono il linguaggio macchina.

Contiene i dispositivi elettronici per acquisire ed eseguire le istruzioni che compongono un programma, elaborando i dati in ingresso in dati in uscita. In un unico circuito integrato è composta da:

- Unità di controllo (Control Unit – CU)
- Unità aritmetico-logica (Arithmetic Logic Unit – ALU)
- Registri

Architettura di Von Neumann – CPU

Dirige l'esecuzione delle istruzioni. Specifiche della particolare famiglia di CPU, le istruzioni ne costituiscono il linguaggio macchina.

- L'unità di controllo recupera la prossima istruzione da eseguire in memoria («Fetch»), la decodifica («Decode») e, dopo avere individuato i dati usati dall'istruzione, la esegue («Execute»)
- L'ALU esegue le operazioni aritmetiche sui dati
- I registri sono locazioni di memoria ad accesso molto veloce usate per memorizzare i dati su cui lavorano un'istruzione, indirizzi o istruzioni (per esempio la successiva istruzione da eseguire)

Architettura di Von Neumann - Memoria

La memoria centrale memorizza i dati da elaborare/elaborati e le istruzioni per eseguire l'elaborazione

E' composta da:

- RAM (Random Access Memory)
- ROM/EPROM (Read Only Memory/Erasable Programmable Read Only Memory)

Architettura di Von Neumann - Memoria

La memoria centrale memorizza i dati da elaborare/elaborati e le istruzioni per eseguire l'elaborazione

La RAM è una sequenza di locazioni (*celle*) di memoria identificate da un indirizzo. In queste celle vengono caricate le istruzioni dei programmi in esecuzione e i dati che i programmi elaborano.

- E' ad accesso casuale, cioè il tempo di accesso non dipende dalla posizione della cella
- E' volatile, cioè si perde il contenuto quando non si spegne l'elaboratore digitale

Architettura di Von Neumann - Memoria

La memoria centrale memorizza i dati da elaborare/elaborati e le istruzioni per eseguire l'elaborazione

L'accesso alla RAM è sia in lettura che in scrittura. Ad esempio, i dati elaborati dai programmi vengono scritti in RAM. Al contrario della RAM, la EPROM:

- Permette la sola lettura (*Read Only*)
- Non è volatile (mantiene le informazioni quando non alimentata)
- Contiene il firmware dell'elaboratore digitale, cioè le istruzioni per il suo corretto avvio (come le istruzioni per controllare quali periferiche di I/O sono connesse e le istruzioni per avviare il *bootloader* del sistema operativo).

Architettura di Von Neumann - Memoria

La memoria centrale memorizza i dati da elaborare/elaborati e le istruzioni per eseguire l'elaborazione

Oltre a RAM e ROM, in un elaboratore digitale sono presenti anche memorie cache, pensate per ospitare dati di frequente utilizzo con tempo di accesso più rapido ai dati rispetto alla RAM

- È volatile
- Quando la CPU deve prelevare dati ed istruzioni dalla RAM, questi vengono copiati in cache insieme a dati ed istruzioni vicine (località spaziale)
- La cache contiene dati e istruzioni usate più frequentemente (località temporale)

Architettura di Von Neumann - Memoria

La memoria centrale memorizza i dati da elaborare/elaborati e le istruzioni per eseguire l'elaborazione

Quando la CPU deve caricare dati o istruzioni da elaborare, controlla se sono già presenti in cache

- Se ci sono, li preleva direttamente dalla cache, anziché dalla RAM (accesso più veloce)
- Se non ci sono, li carica dalla RAM alla cache (futuri accessi più veloci).

Memoria secondaria

- Non è volatile: i dati restano allo spegnimento del computer.
- Serve per immagazzinare *grandi quantità di dati* (ad oggi, contiene centinaia di GB o qualche TB).
- I dati e i programmi contenuti nei file negli hard disk (memoria secondaria), verranno caricati in RAM (memoria centrale) per essere elaborati ed eseguiti.
- Oltre a HDD e SSD, sono memoria secondaria pendrive USB, schede SD, DVD, CD…

Architettura di Von Neumann - Memoria

Perché è necessaria una tale organizzazione gerarchica della memoria in un elaboratore digitale?

Architettura di Von Neumann - Memoria

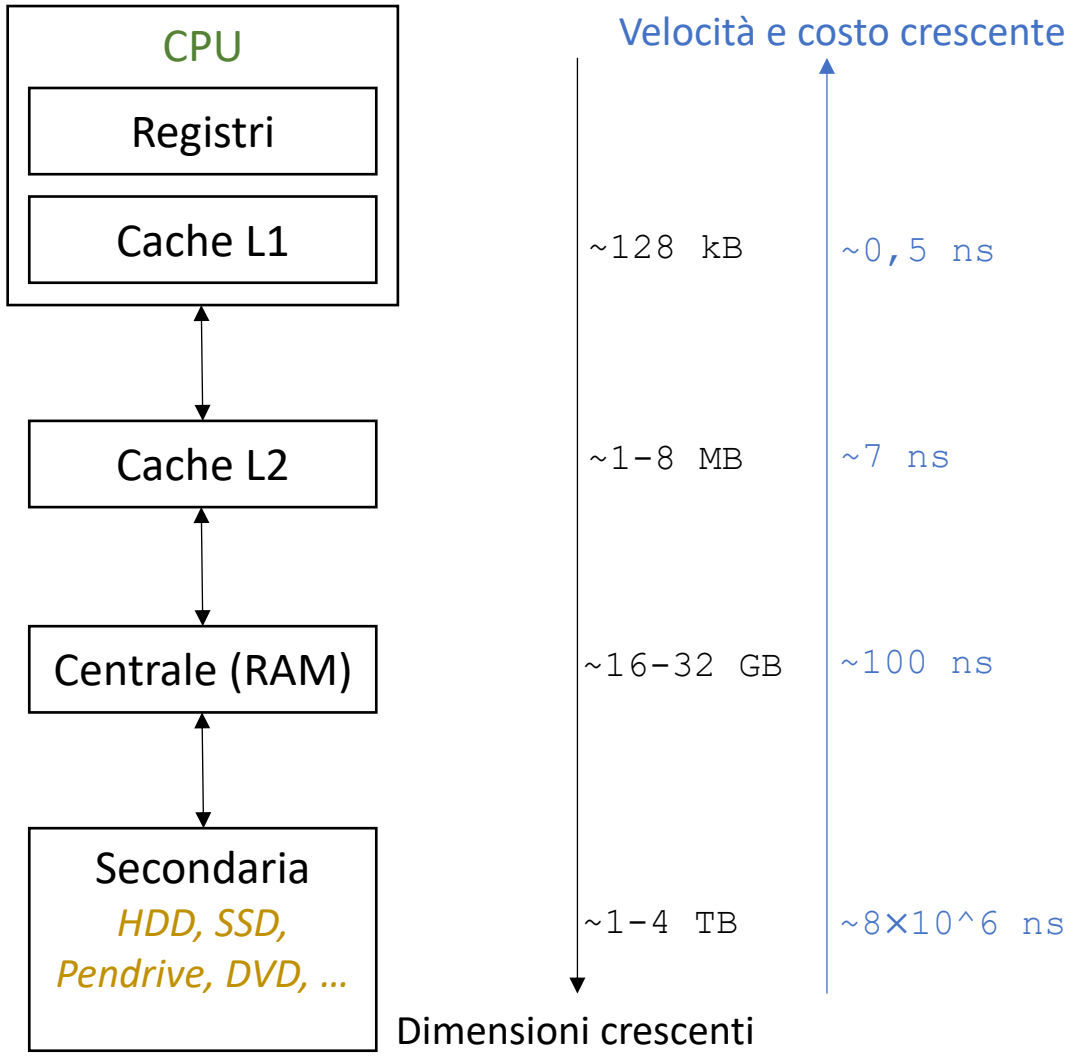
«Teoricamente si vorrebbe avere una memoria di capacità indefinitamente grande, tale che ogni particolare parola possa essere immediatamente disponibile. Siamo così costretti a riconoscere la possibilità di costruire una gerarchia di memorie, ognuna delle quali ha capacità superiore a quella precedente, ma a cui si può accedere con minor velocità.»

John Von Neumann

Organizzazione gerarchica della memoria

- La memoria è più lenta della CPU e tende a limitarne le prestazioni
- Occorre un compromesso tra costo, prestazioni e dimensione della memoria

Poca memoria <u>veloce vicino alla CPU</u> e tanta memoria <u>lenta per memorizzare informazioni</u>



Gerarchia memorie – Principi di località

L'obiettivo ideale sarebbe fornire una quantità di memoria pari a quella disponibile nella tecnologia più economica garantendo una velocità di accesso pari a quella della memoria più costosa.

L'organizzazione gerarchica della memoria cerca il miglior compromesso tra costi, dimensione e prestazioni. In particolare, la cache è gestita sfruttando due principi di località nell'esecuzione delle istruzioni di un programma

- Località temporale
- Località spaziale

Gerarchia memorie – Principi di località

Località temporale: tendenza a riferirsi allo stesso elemento entro breve tempo (es. ripetizioni all'interno degli algoritmi)

Località spaziale: tendenza a riferire la successiva lettura/scrittura in memoria ad elementi che hanno indirizzo vicino a quello dell'elemento corrente (es. istruzioni sequenziali)

Architettura di Von Neumann – I/O

Periferiche di Input/Output (I/O): dispositivi che permettono l'interazione tra elaboratore digitale e utente, ambiente e altri elaboratori digitali.

- Periferiche di Input: interazione in ingresso per l'elaboratore digitale (es. tastiera, mouse…)
- Periferiche di Output: interazione in uscita dall'elaboratore digitale (es., schermo, stampante…)

Architettura di Von Neumann – Bus

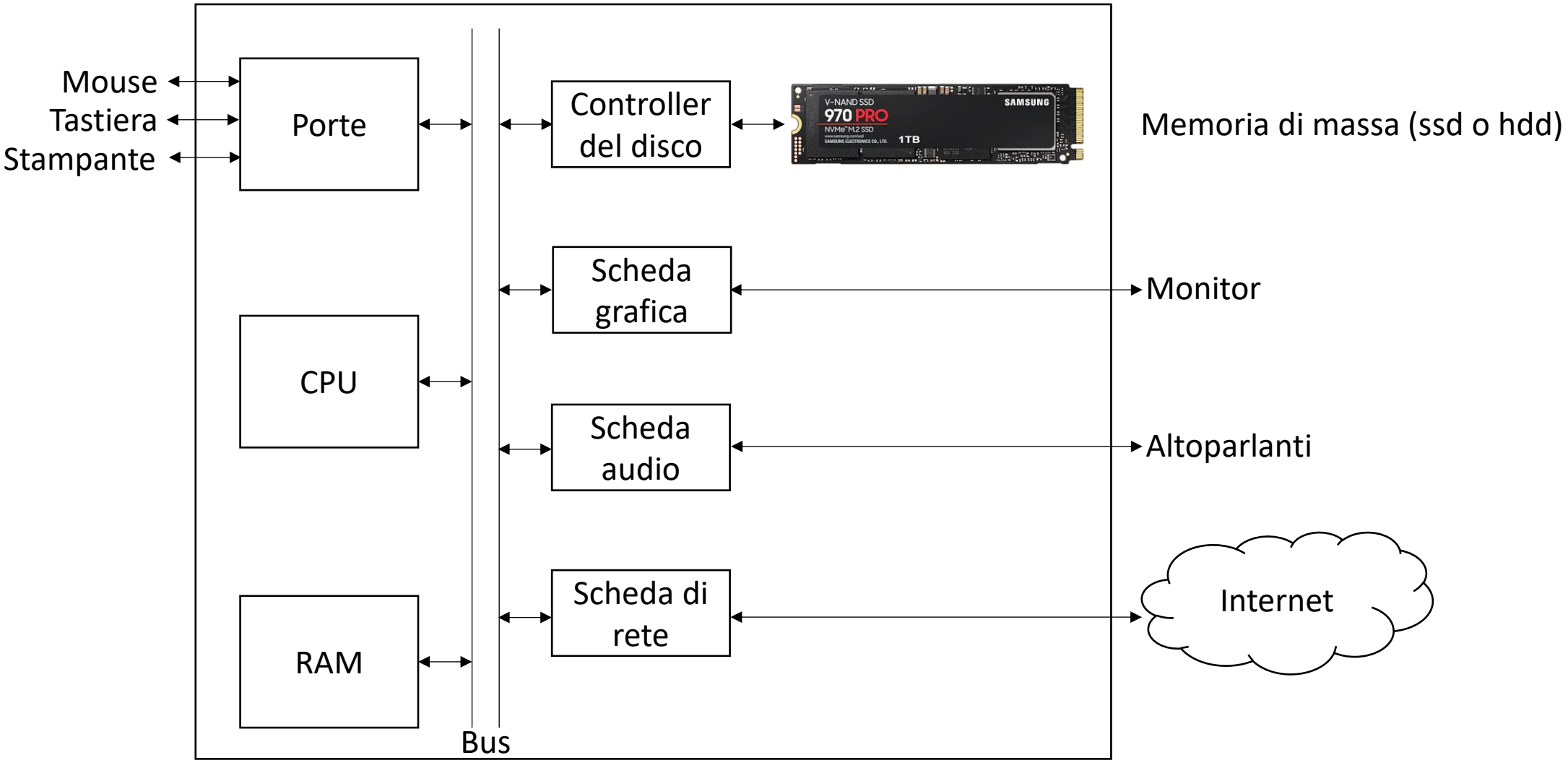
I byte fluiscono tra CPU, memoria e periferiche, sotto il controllo della CPU, attraverso il bus di sistema.

Componenti di un PC

Architettura di un PC

Anche un moderno PC, come tutti gli elaboratori digitali, si basa sull'architettura di Von Neumann.

Architettura di un PC

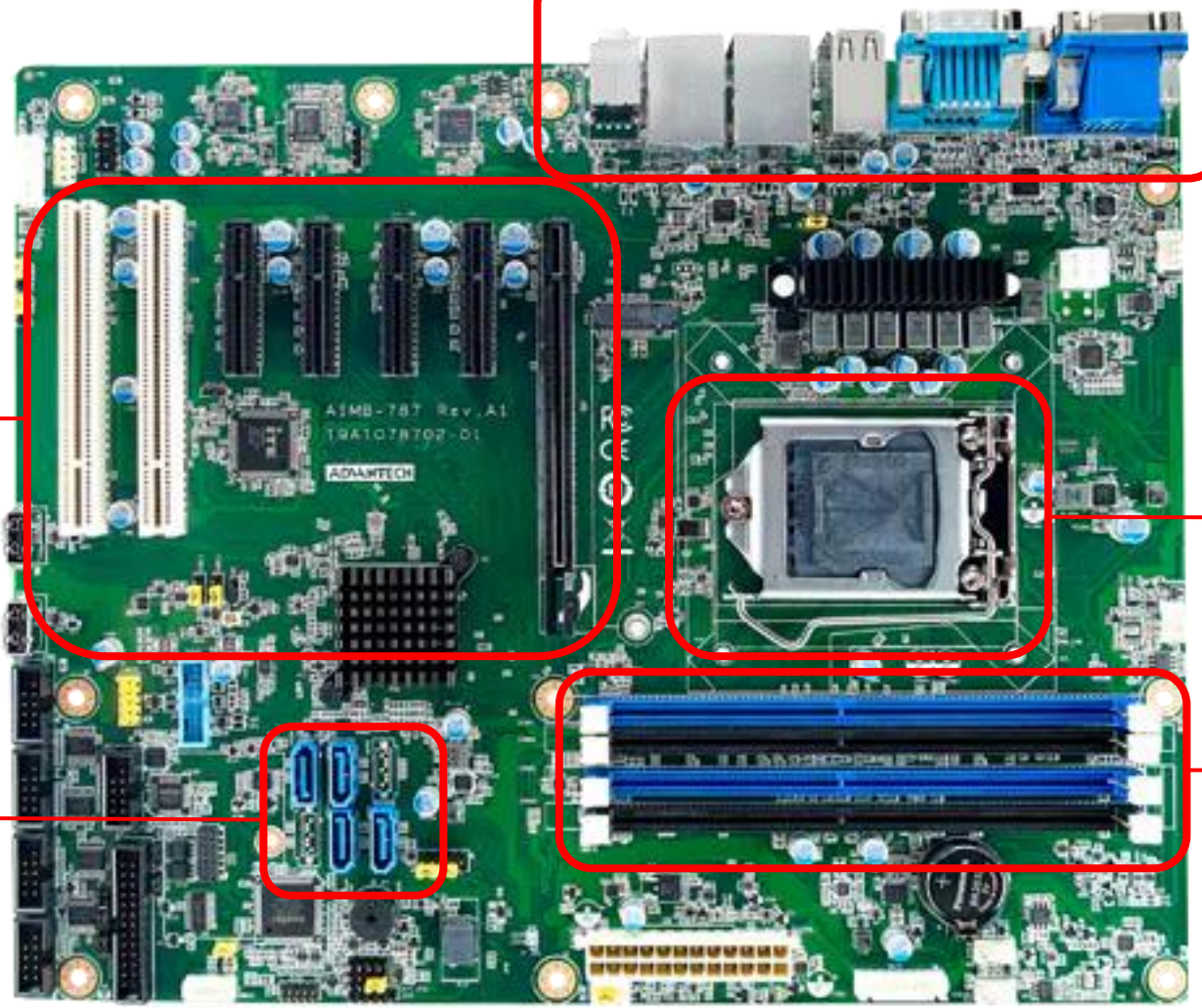


Architettura di un PC – Scheda Madre

E’ il circuito stampato principale di un elaboratore digitale «general purpose», cioè un computer.

Permette la comunicazione tra CPU e RAM attraverso il bus e fornisce i connettori per le periferiche

Architettura di un PC – Scheda Madre



Connettori periferiche (USB, Rete, Monitor…)

Slot di espansione (per schede grafiche, audio…)

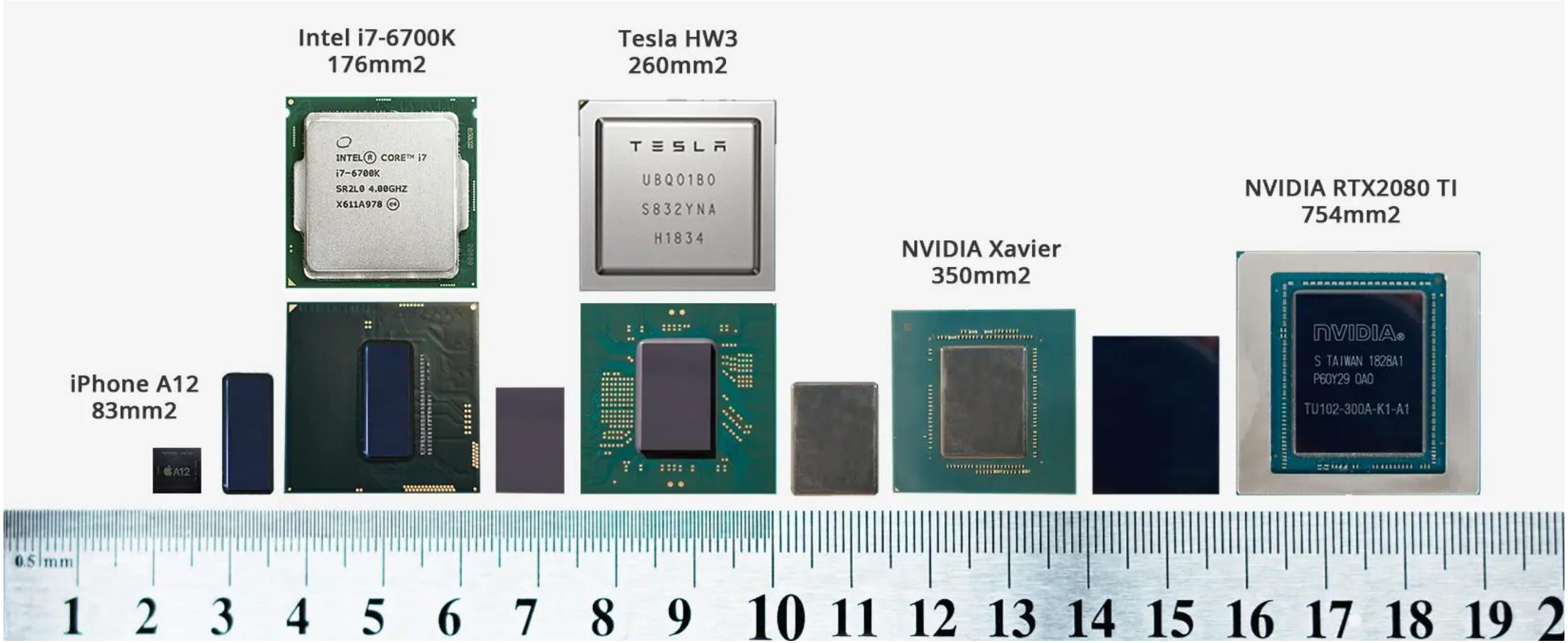
Socket CPU

Connettori SATA (per memoria di massa)

Slot RAM

CPU – Microprocessore

CPU – Microprocessore



CPU – Microprocessore

Una CPU per un comune PC ha all'interno miliardi di transistor (~10). I transistor sono i blocchi costitutivi di una CPU e, intuitivamente, sono microscopici interruttori (On/Off – 1/0) che formano le porte logiche e permetto ad una CPU di eseguire le istruzioni.

Frequenza di clock: Numero di commutazioni tra i due livelli «0» e «1» che i circuiti all'interno di una CPU (o di un componente elettronico) sono in grado di eseguire in un secondo. Una CPU a 3 GHz è in grado di eseguire 3.000.000.000 di commutazioni al secondo…

Più alto è il clock più «veloce» sarà la CPU…

CPU – Parallelismo

Un sistema multiprocessore è un elaboratore con più CPU collegate insieme per consentire l'elaborazione simultanea di più istruzioni («in parallelo»), aumentando la velocità di esecuzione, o per garantire ridondanza e quindi tolleranza ai guasti.

Processore Multi Core: microprocessore costruito in un unico circuito integrato costituito da due o più unità di elaborazione (i «core»), capaci di eseguire istruzioni in parallelo.

Ad un maggior numero di core corrisponde una CPU più «veloce»…

CPU – Microprocessore

Più alto è il clock più «veloce» sarà la CPU…

Ad un maggior numero di core corrisponde una CPU più «veloce»…

Vero, ma…

- Difficile comparare CPU di generazioni e produttori diversi
- La frequenza può fluttuare tra più valori, in base al carico di lavoro e l'energia richiesta (es. «*thermal throttling*»)
- Per valutare le CPU si aggiungono anche altre metriche come i FLOPS (FLoating point Operations per Second). CPU desktop sono sull'ordine delle centinaia di miliardi di FLOPS (1 miliardo di FLOPS = 1 GFLOPS).

CPU – Microprocessore

- Un altro parametro è il TDP (Thermal Design Power), che fornisce un'indicazione del «calore» dissipato da una CPU
- Esistono risorse online per confrontare CPU e i diversi parametri con test per valutarne le prestazioni («benchmark»). Esempi:
  - https://www.cpubenchmark.net/
  - https://cpu.userbenchmark.com
  - https://www.tomshardware.com/reviews/cpu-hierarchy,4312.html

CPU – Microprocessore

Come si produce un microprocessore?

L'elemento base è il silicio: secondo elemento per abbondanza nella crosta terrestre (27% del peso).

Nei processori deve essere puro: viene fuso e purificato per ottenere lingotti cilindrici (30 cm di diametro).

Da lingotto, con una «affettatrice» vengono ricavati dischi spessi 1mm: i «wafer».

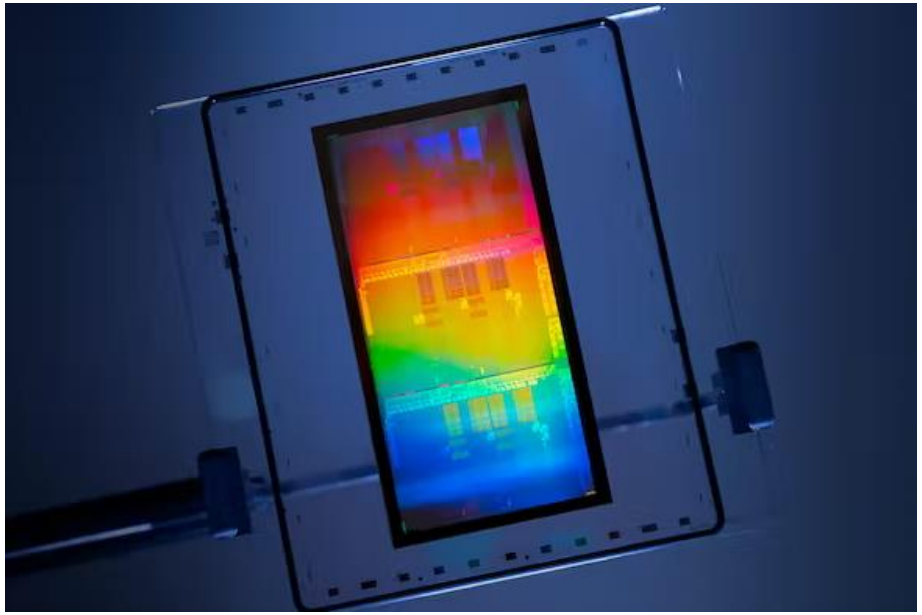
CPU – Microprocessore

Una volta progettato un microprocessore, il produttore crea delle «maschere», degli stampati che rappresentano i transistor e i blocchi di un processore.

La maschera è un blocco di quarzo serigrafato

Le maschere vengono usate per imprimere gli schemi nel wafer mediante fotolitografia

Il wafer viene quindi rivestito di un materiale fotoresistente in maniera uniforme



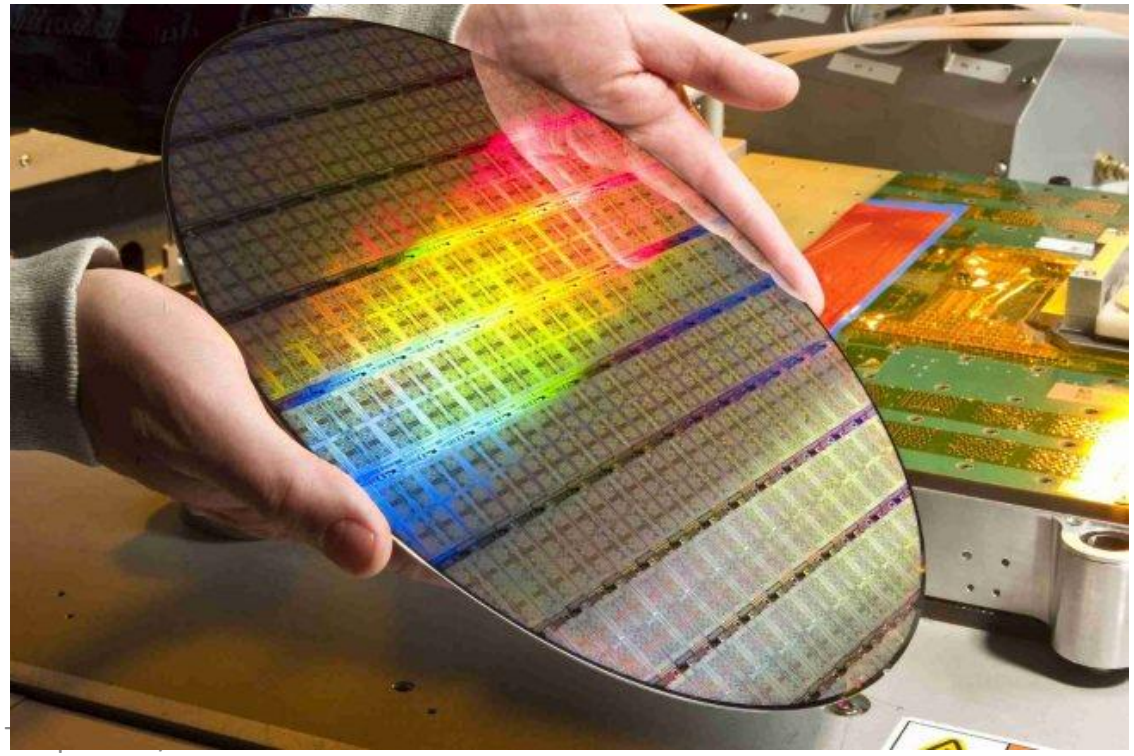
CPU – Microprocessore

Una raggio da un proiettore di luce colpisce il wafer illuminando il materiale fotoresistente attraverso le maschere.

L'immersione del wafer in una soluzione apposita rimuove tutte le parti dove la luce ha colpito.

Il processo viene ripetuto più volte, fino a formare diversi strati sul wafer

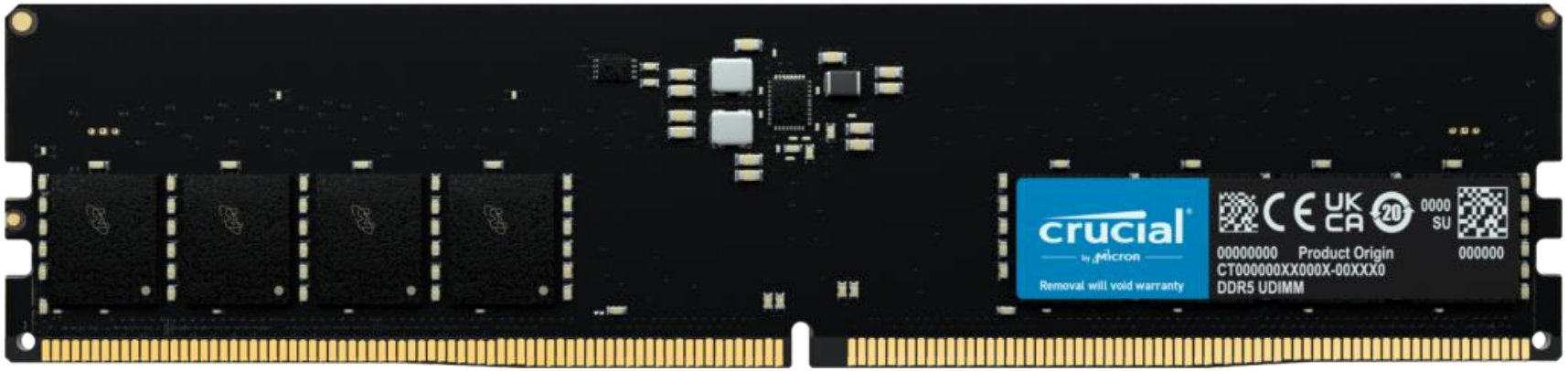
Alla fine del processo, il wafer contiene più chip, che vengono tagliati



CPU – Microprocessore

Al microscopio: https://www.youtube.com/watch?v=Fxv3JoS1uY8

RAM



La RAM è realizzata mediante tecnologia SDRAM (Synchronous Dynamic Random Access Memory). Semplificando, RAM possono essere confrontate per

- Dimensione – **Oggi**, un buon sistema desktop a 16 GB
- Standard DDR – DDR1, DDR2, DDR3, DDR4, DDR5. DDR5 è appena uscito e, tra una famiglia e la successiva, c'è tipicamente un aumento della banda (aumento della capacità di trasferire informazioni in un secondo)
- Frequenza di clock – Occorre assicurarsi che quella RAM sia compatibile con processore e scheda madre

Memoria di massa – Hard Disk

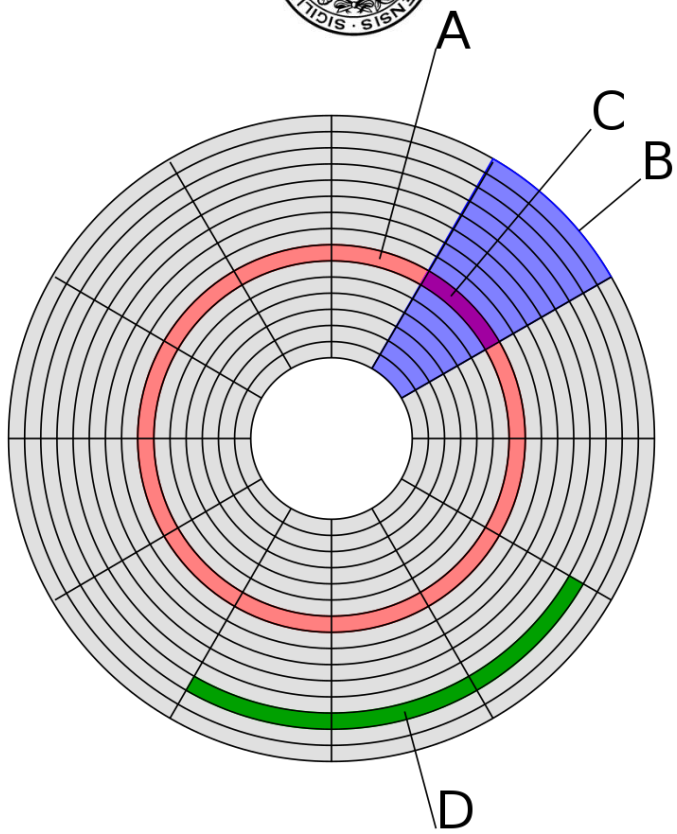
Memoria di massa – Hard Disk

- HDD – Hard Disk Drive
- Sono dischi magnetici costituito da uno o più piatti sovrapposti di alluminio/vetro rivestiti da materiale ferromagnetico
- Una testina per lato legge/scrive i settori leggendo/cambiando il verso della magnetizzazione

Parametri

- Dimensione (ormai alcuni Terabyte)
- Giri per minuto – RPM – (5400, **7200**, 10400, 15000). Influisce su tempo di accesso
- Dimensione Cache

Memoria di massa – Hard Disk

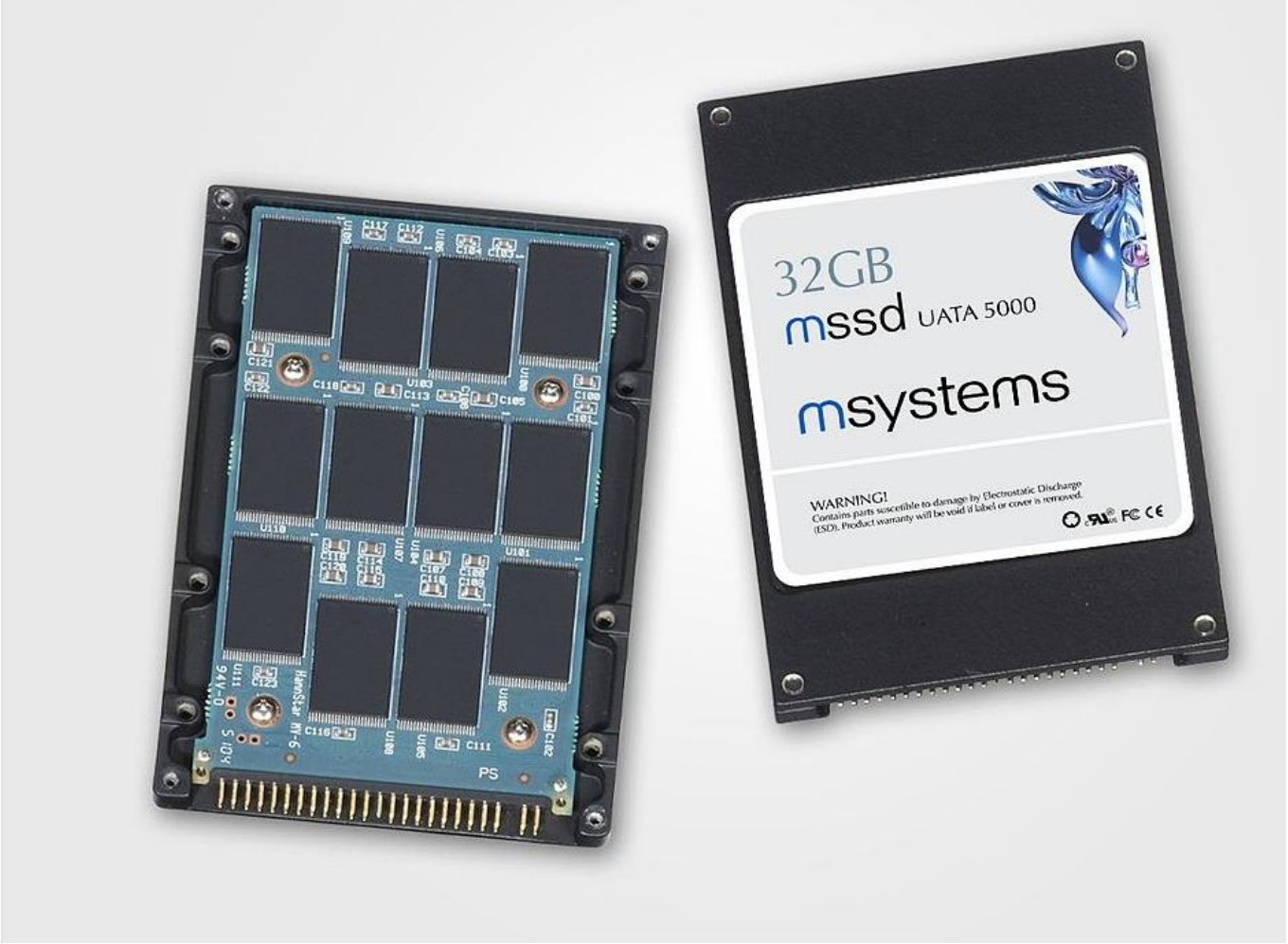


- A – traccia
- B – settore
- C – settore di una traccia
- D – cluster

Memoria di massa – Hard Disk

- Componente meccanico: può rompersi o smagnetizzarsi
- Superati in velocità da unità a stato solido (SSD – Solid State Disk): in media 520 MB/s per lettura e scrittura contro 125 MB/s
- Oggi, la ridotta differenza di prezzo tra SSD e HDD non giustifica l'uso di un HDD a discapito di un SDD per un sistema desktop, se non per ragioni come lo storage di grandi quantità di dati a scopo di backup.

Memoria di massa – SSD



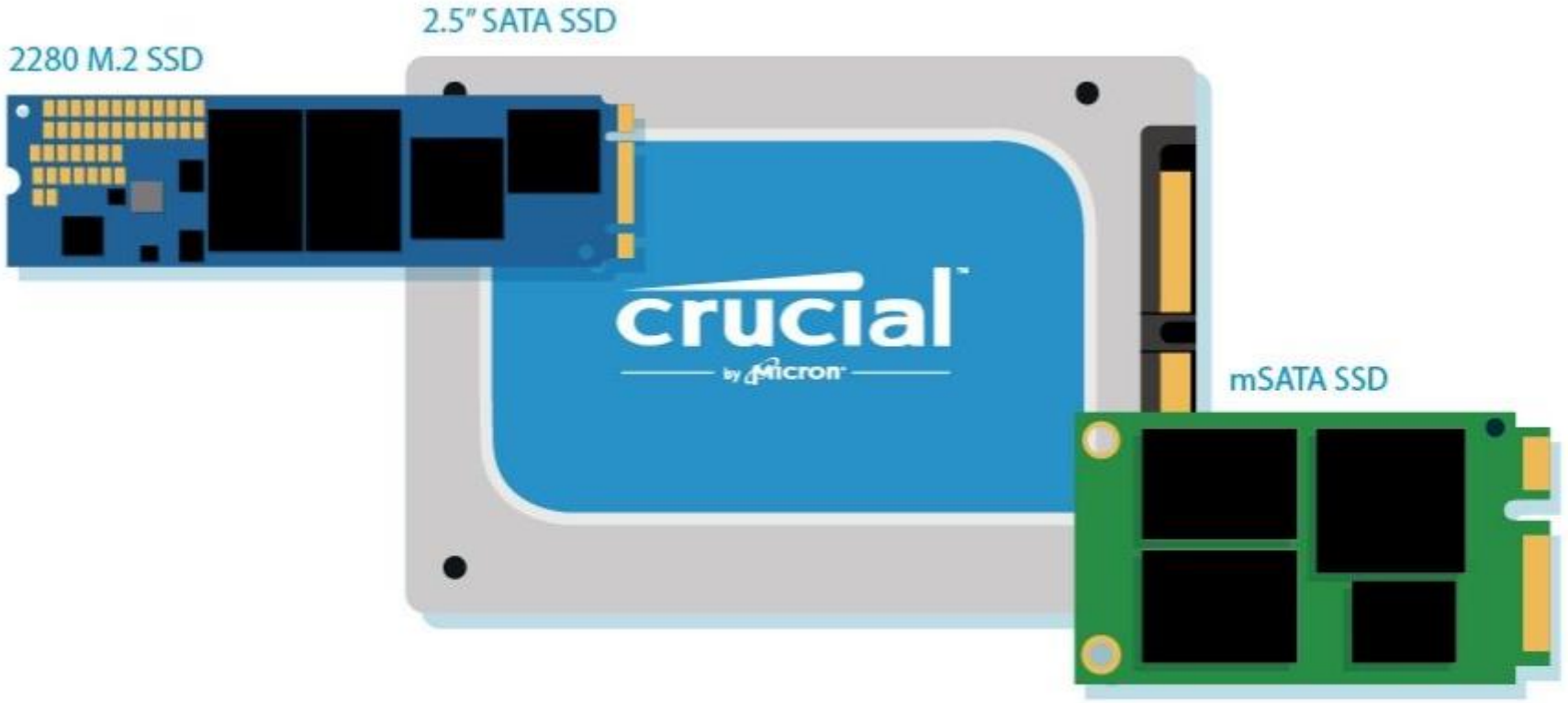
Memoria di massa – SSD

- Basato su memorie flash «NAND» realizzate mediante *transistor*. Non necessita di parti meccaniche (a differenza degli HDD)

Parametri

- Velocità di lettura e scrittura sequenziale (tipicamente 560 MB/s in lettura e 520 MB/s in scrittura per SSD Sata, oltre i 2000 MB/s in lettura e scrittura per SSD PCIE M.2)
- Resistenza, espressa in vari modi MTBF (Mean Time Between Failures – es. 1.800.000 h), quantità totale di dati che un SSD può scrivere nel corso della sua vita (es. 180 TB), Classe di resistenza (es. 5 anni a 195 GB al giorno)
- Interfaccia di comunicazione (SATA o PCIE) e form factor (2.5'', mSATA, M.2). Gli SSD PCIE sono molto più veloci degli SSD sata, a loro volta più veloci dei tradizionali HDD

Memoria di massa – SSD



Istruzioni per un elaboratore digitale

Istruzioni

La CPU esegue istruzioni in linguaggio macchina

Cosi come i dati, anche le istruzioni sono sequenze di 0 e 1. In linguaggio macchina un'istruzione si compone di

opcode dati o indirizzi

Esempio `00000011 11000011`

L'opcode identifica l'istruzione da eseguire sui dati o indirizzi di memoria

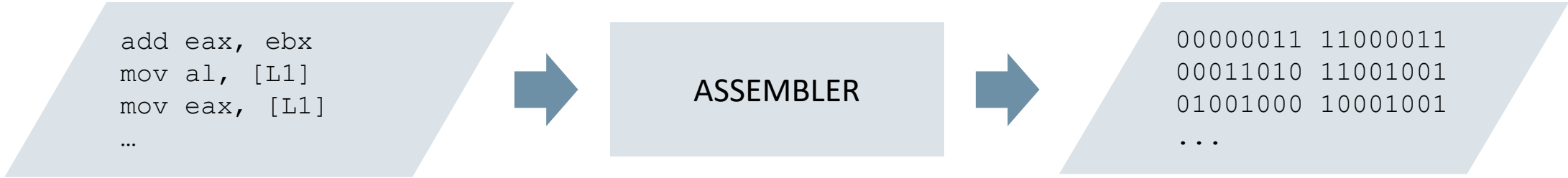
Istruzioni

Un programma eseguibile da un elaboratore digitale è una sequenza di 0 e 1. Scrivere un programma complesso direttamente in linguaggio macchina è pressoché impossibile, quantomeno fortemente impratico.

Il linguaggio assembly, fornendo simboli per ognuna delle istruzioni in linguaggio macchina, rendendolo comprensibile ad un programmatore

```
00000011 11000011 -> add eax, ebx
```

Un programma chiamato assembler traduce poi il programma assembly in linguaggio macchina.



Assembly

Margaret Hamilton

Assembly

Sebbene la «vicinanza» dell'assembly al linguaggio macchina permetterebbe la creazione di programmi veloci ed efficienti, con accesso diretto all'hardware, l'assembly

- E' impratico
- Non è portabile (dipende dalla specifica architettura)

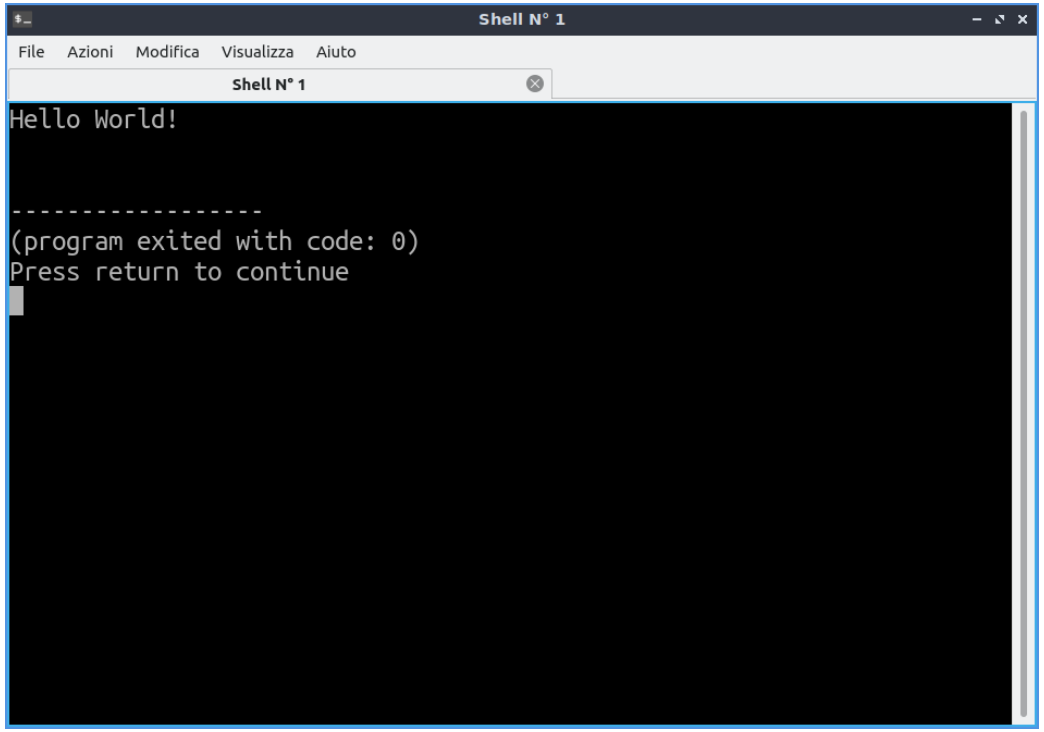
La maggior parte dei programmi è scritta in linguaggi (di programmazione) ad alto livello (di astrazione rispetto al linguaggio macchina).

Linguaggi di programmazione

- Linguaggio macchina
  - Istruzioni sono sequenze di 0 e 1
  - Dipende dall'architettura dell'elaboratore
- Linguaggio assembly: livello di astrazione basso
  - Fornisce una corrispondenza quasi 1 a 1 con le istruzioni in linguaggio macchina
  - Dipende dall'architettura dell'elaboratore
- Linguaggi ad alto livello (es. C, C++, Python, Java…)
  - Fornisce costrutti più complessi delle istruzioni elementari eseguibili dall'elaboratore
  - Spesso indipendenti dall'architettura dell'elaboratore

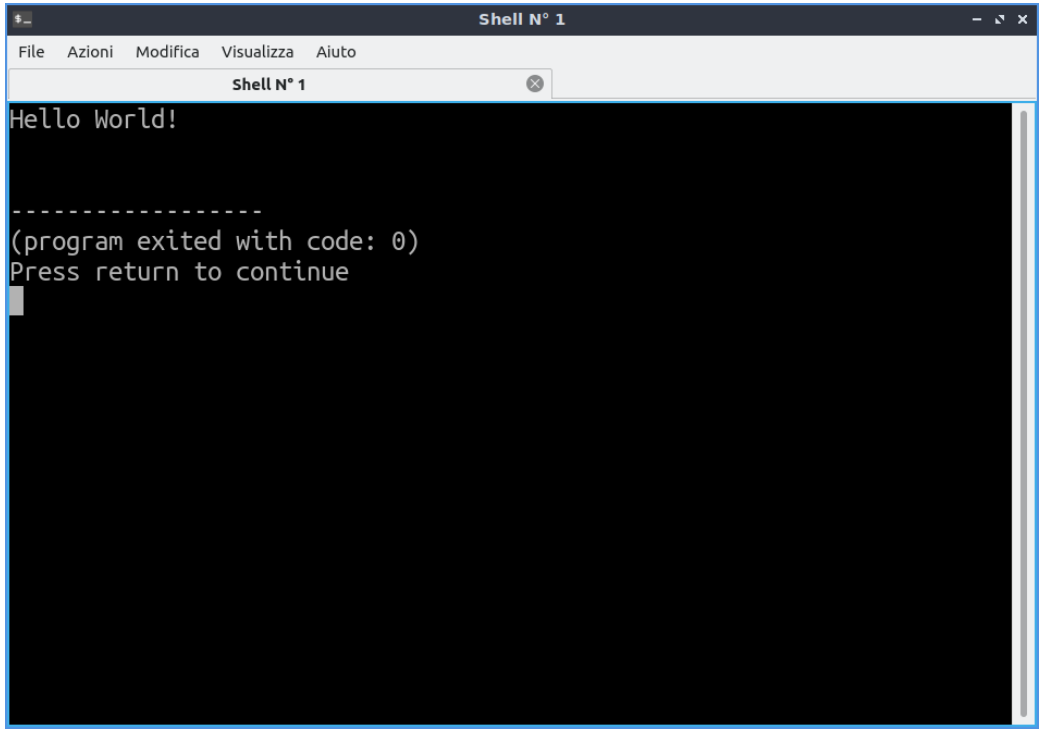
Es.: Assembly

```
%include "asm_io.inc"

segment .data

        msg db "Hello World!", 10, 0

segment .text

        global ciaoasm
        ciaoasm:

                enter 0,0
                pusha

                mov eax, msg
                call print_string

                popa

                mov eax, 0

                leave
                ret
```



Es.: C++

```cpp
#include <iostream>

int main() {
	std::cout << "Hello World!\n";
	return 0;
}
```



Scrittura di un programma

Conversione della descrizione accurata (specifiche) di un problema o di una classe di problemi in una sequenza di istruzioni per risolverlo mediante un elaboratore digitale.

Un linguaggio di programmazione di alto livello permette al programmatore di impartire all'elaboratore istruzioni complesse che «astraggono» il linguaggio macchina.

Scrittura di un programma

Lo sviluppo di un programma si compone di diverse fasi: dalla raccolta delle specifiche del problema di risolvere, alla valutazione delle possibili alternative, fino alla scrittura del programma e ai test per verificarne il corretto funzionamento.

Semplice da usare non significa semplice da sviluppare!

Fasi di sviluppo di un programma

1. Definizione (accurata) del problema da risolvere e delle operazioni necessarie per farlo
2. Progettazione del programma
3. Codifica in un linguaggio di programmazione
4. Test e debug
5. Scrittura della documentazione tecnica (es. manuale)

Ingegneria del software

Ingegneria del software

Motivazioni:

- Ridurre errori (i bug), realizzando verifiche già dalla fase di progettazione.
- Migliorare l'affidabilità.
- Documentare il codice, già fin dalle prime fasi di progettazione, migliorando il lavoro in team.
- Rendere modulare il software.

Il primo «bug»

- 9 settembre 1947, Harvard University
- Il «Mark II» (23 t, la pianta occupava 370 mq) andava continuamente in errore
- Gli ingegneri aprono il Mark II per cercare di capire qual è la causa dell'errore…

Il primo «bug»

https://education.nationalgeographic.org/resource/worlds-first-computer-bug

Affidabilità e «bug» - Ariane V (ESA)

- Ariane V: lanciatore dell'ESA (European Space Agency) https://it.wikipedia.org/wiki/Ariane_5
- 4 giugno 1996, primo lancio (Ariane V Flight 501)

https://www.youtube.com/watch?v=gp_D8r-2hwk

Affidabilità e «bug» - Ariane V (ESA)

- 37 secondi dopo il lancio un dato di volo (la velocità orizzontale) in floating point a 64 bit viene convertito in un intero su 16 bit (che ammette valori -32768 a 32767), ma è troppo grande per essere rappresentato su 16 bit...
- I progettisti avevano riusato parte del codice relativo all'Ariane 4, dove la gestione di alcune eccezioni tra cui quelle relative alla velocità orizzontale erano state disabilitate («*physically limited or that there was a large margin of safety*»)
- Una reazione a catena a seguito dell'errore porta all'esplosione del razzo (costo del lancio: 370 milioni di dollari)

https://medium.com/dataseries/crash-and-burn-a-short-story-of-ariane-5-flight-501-3a3c50e0e284

Affidabilità e «bug» - Mars Climate Orbiter

- 11 dicembre 1998: da Cape Canaveral, Florida, decolla il razzo Delta II 7425 con a bordo il *Mars Climate Orbiter*
- Obiettivo della missione scientifica è studiare il clima di Marte
- L'orbiter avrebbe dovuto essere posto in orbita intorno Marte ad una distanza di circa 140 km dalla superficie
- Il 23 settembre 1999 le comunicazioni si interrompono durante le operazioni di entrata in orbita
- La sonda si trovava a 57 km dalla superficie del pianeta anziché i 140-150 preventivati ed è andata distrutta dall'attrito con l'atmosfera

Affidabilità e «bug» - Mars Climate Orbiter

- 11 dicembre 1998: da Cape Canaveral, Florida, decolla il razzo Delta II 7425 con a bordo il *Mars Climate Orbiter*
- Obiettivo della missione scientifica è studiare il clima di Marte
- L'orbiter avrebbe dovuto essere posto in orbita intorno Marte ad una distanza di circa 140 km dalla superficie
- Il 23 settembre 1999 le comunicazioni si interrompono durante le operazioni di entrata in orbita
- La sonda si trovava a 57 km dalla superficie del pianeta anziché i 140-150 preventivati ed è andata distrutta dall'attrito con l'atmosfera

Affidabilità e «bug» - Mars Climate Orbiter

Causa:

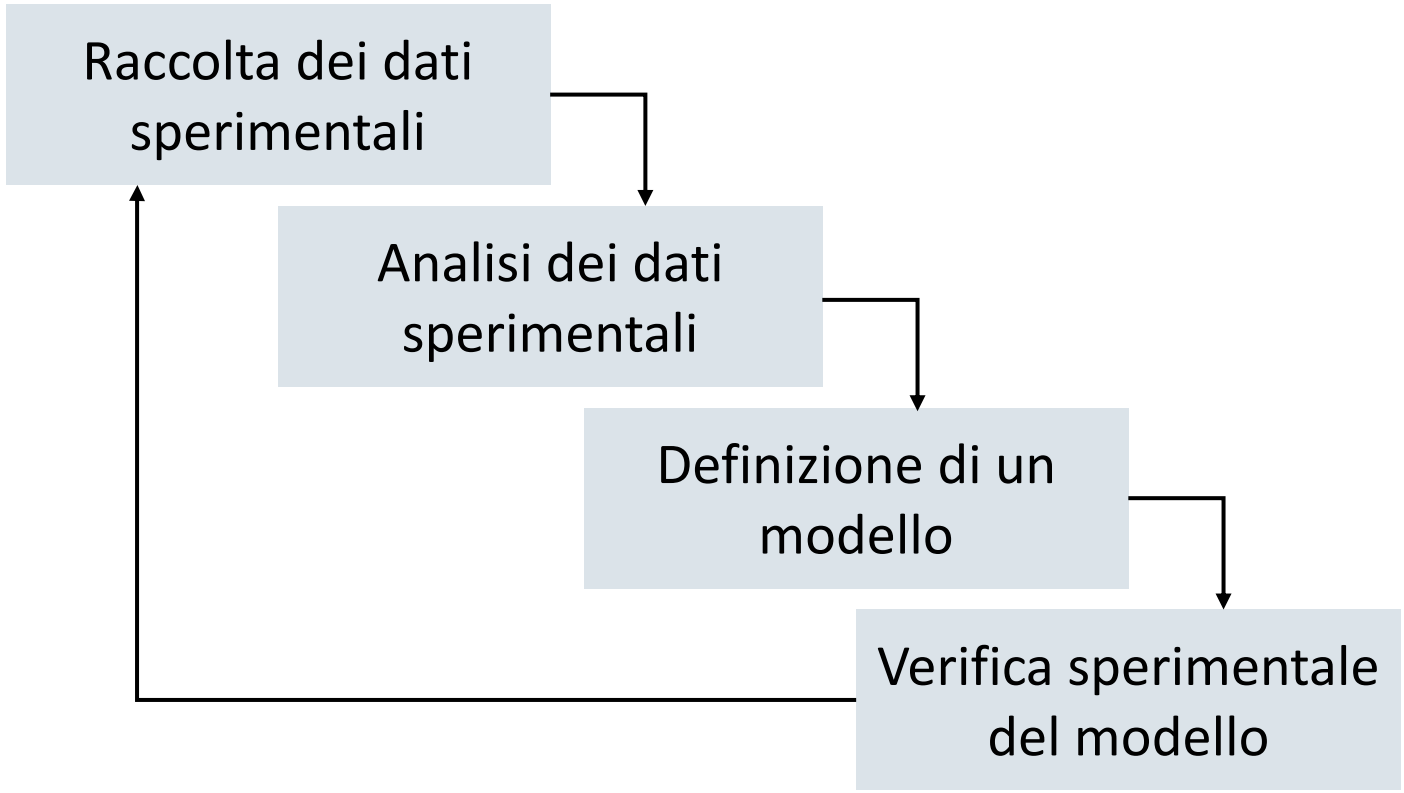
- Un software (di Lockheed Martin) usa il sistema imperiale britannico per le misure
- Un altro software (NASA), usato per una precedente missione, si aspettava risultati nel sistema internazionale di misura.

Costo della missione: 327.6 milioni di dollari.

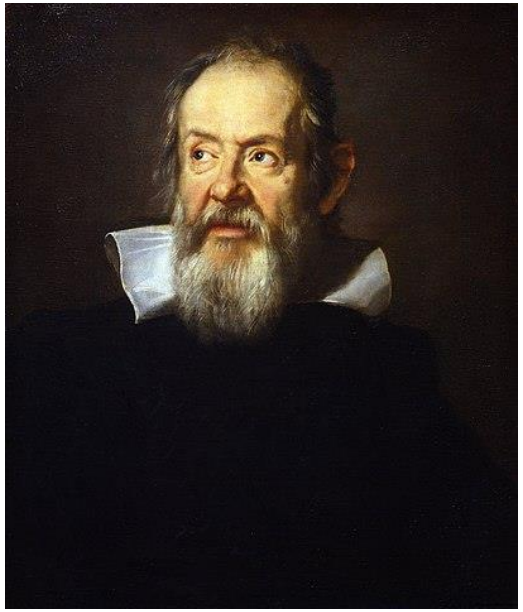
Altri bug «celebri»: https://www5.in.tum.de/~huckle/bugse.html

Fasi di sviluppo di un programma

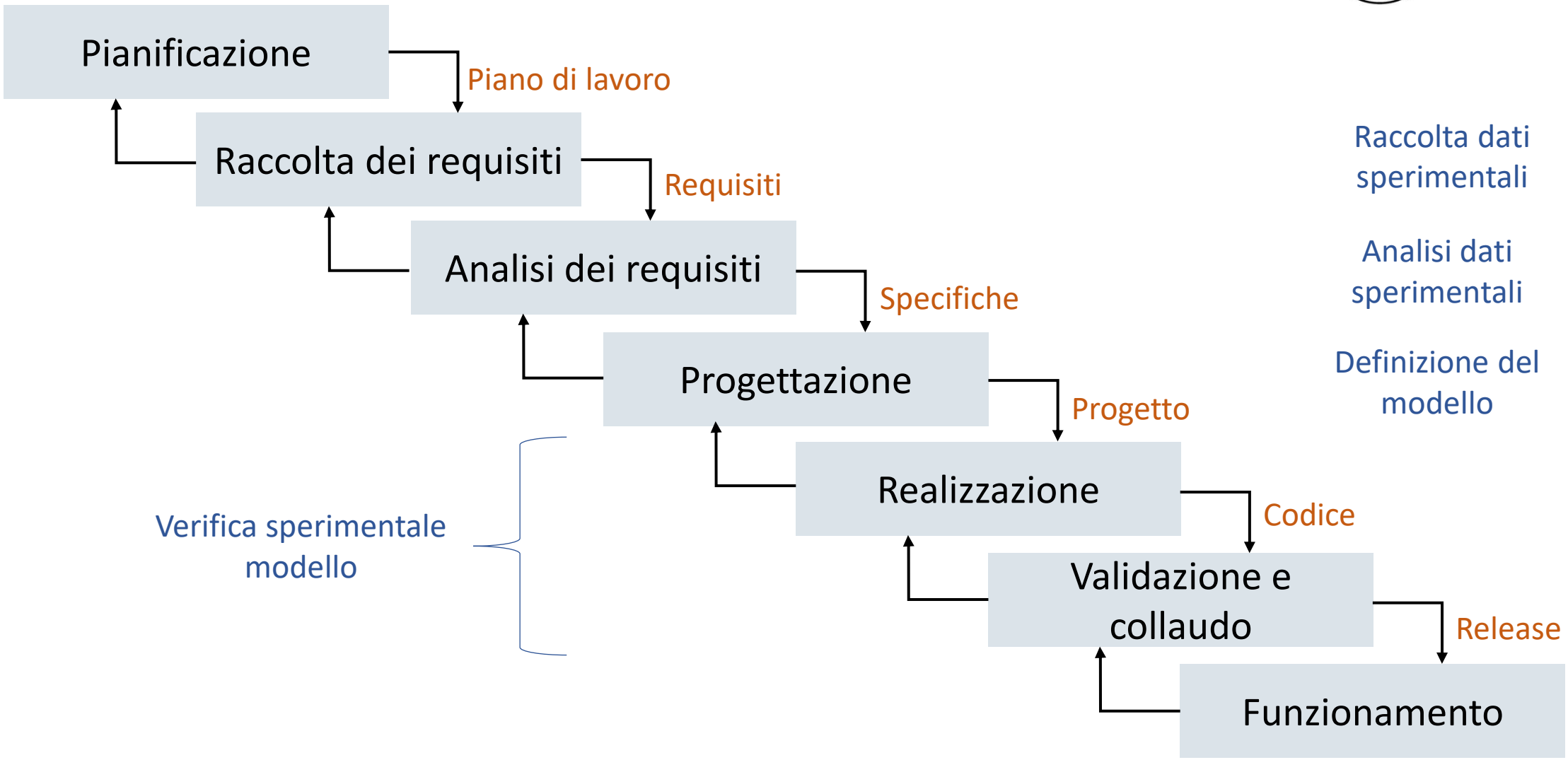
Esiste un parallelismo da sviluppo di un programma e metodo sperimentale.



Galileo Galilei



Fasi di sviluppo di un programma



1. Pianificazione

- Stima di costi e rischi
- Definizione dei criteri di valutazione (finali, ma anche per monitorare i progressi in fase intermedia)
- Piano dettagliato per
  - Divisione dei compiti e responsabilità
  - Suddivisione delle risorse
  - Ispezioni e revisioni

2. Raccolta dei requisiti

- Definire chi fornirà i requisiti (gli «stakeholder»)
- Raccogliere i requisiti (es. attraverso «interviste»)
- Definire scenari d’uso, casistiche

3. Analisi dei requisiti

Definire che cosa deve fare il software (non il come). Dai requisiti si ottengono delle specifiche descritte «formalmente».

- Non è l'attività che si vuole automatizzare
- Non è uno specifico software
- Es. app per registrare video
  - Cosa: memorizzare sequenze di immagini digitali
  - Come: a 25 fps, a 60 fps, con MPEG-4, con H.264, ecc.

4. Progettazione - 5. Realizzazione

Progettazione: definizione del «come», cioè come il programma deve funzionare per soddisfare «che cosa» deve fare.

- Progettazione delle interfacce utente
- Progettazione della logica applicativa
- Progettazione dei dati

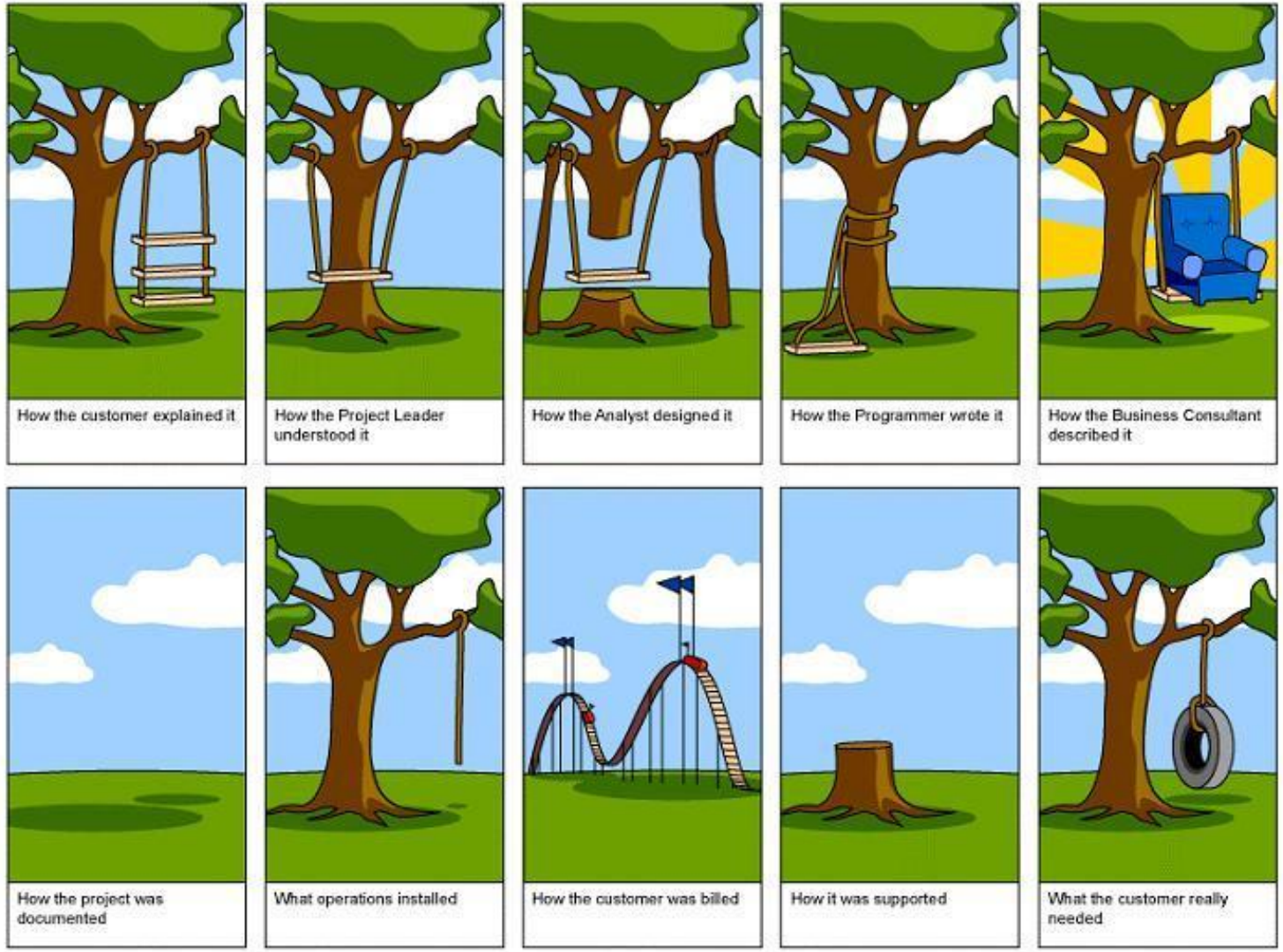
Realizzazione: traduzione del progetto in codice.

6. Collaudo – 7. Funzionamento

- Valutazione mediante metodi formali
- Test su casi d’uso
- Verifiche sulla base dei criteri di valutazione fissati in pianificazione
- Configurazione
- Documentazione

Rilascio (release)

Fasi di sviluppo di un programma



Programmi, linguaggi di programmazione e tutela del software

Linguaggi di programmazione

Codice sorgente: programma scritto nel linguaggio di programmazione

Codice eseguibile: programma in linguaggio macchina effettivamente eseguibile dall'elaboratore digitale

Linguaggi di programmazione

Codice sorgente

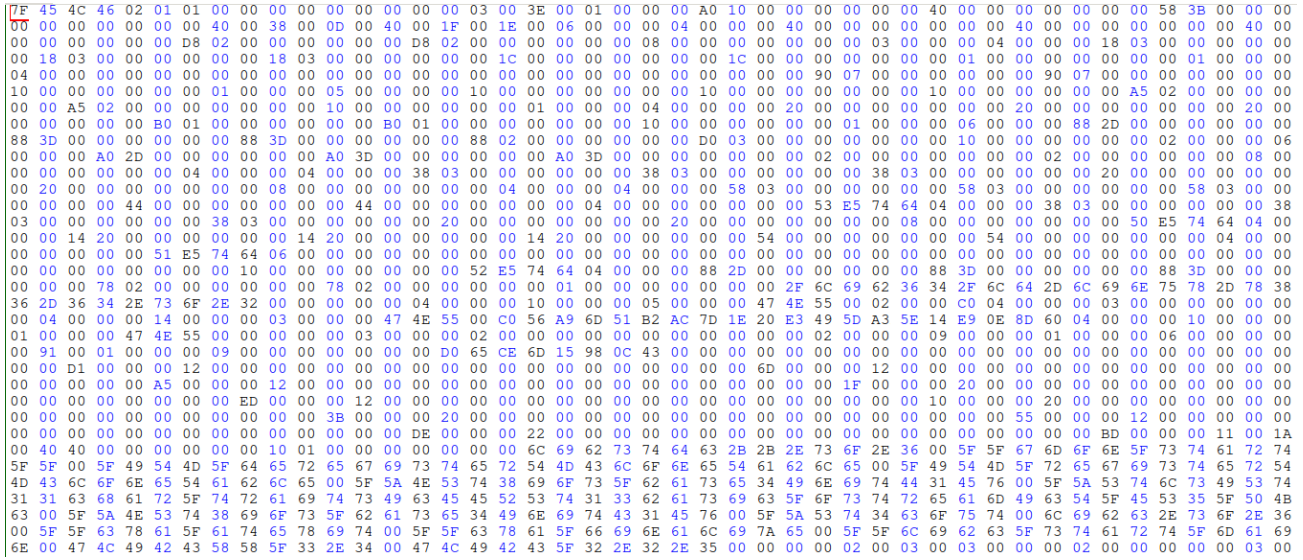
- È un (o più) file di testo
- Scritto in un linguaggio leggibile da esseri umani
- Viene «tradotto» in linguaggio macchina da un software dedicato (es. assembler, compilatore, interprete)
- Indipendente dalla piattaforma

```
#include <iostream>

int main() {
        std::cout << "Hello World!";
        return 0;
}
```

Linguaggi di programmazione

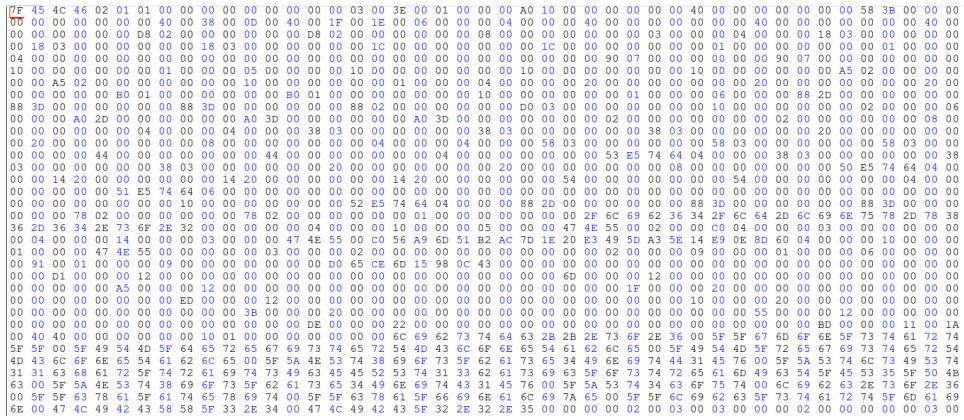
Codice eseguibile

- È un (o più) file binario
- Contiene le istruzioni in linguaggio macchina che la CPU può eseguire
- Dipendente dalla piattaforma



Linguaggi di programmazione

```
#include <iostream>

int main() {
        std::cout << "Hello World!";
        return 0;
}
```



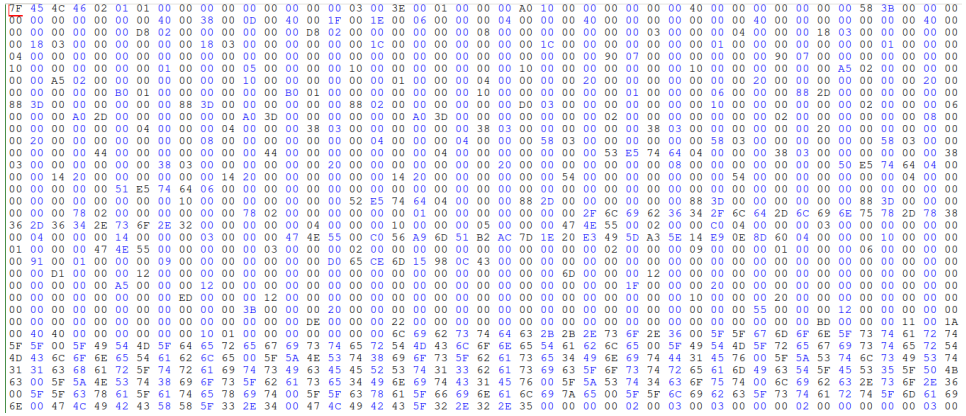
Sorgente

```
00000011 11000011
00011010 11001001
01001000 10001001
...
```

Reverse engineering

```
00000011 11000011
00011010 11001001
01001000 10001001
...
```

Sorgente



```
#include <iostream>

int main() {
        std::cout << "Hello World!";
        return 0;
}
```

Reverse engineering

«Decompilazione» del file eseguibile

- Risalire dal file eseguibile ad uno dei possibili codici sorgente.
- Per ragioni di interoperabilità con altri software, per studio e analisi, per copiare.

E’ lecito il reverse engineering del file eseguibile?

Come è tutelato il software?

Software – Copyright

- Il software è tutelato dal diritto d'autore, come un'opera letteraria
- Senza autorizzazione dell'autore non è possibile farne una copia, venderlo, modificarlo..
- Il proprietario del software è colui che ne detiene i diritti d'autore copyright
- Normalmente non si acquista il software, ma la licenza d'uso (eventualmente gratuitamente): la licenza stabilisce le facoltà dell'acquirente (es.: su quanti dispositivi il software può essere installato, per quanto tempo può essere utilizzato…)

Software – Copyright

- Il copyright non protegge le funzionalità realizzate dal software, l'idea di base, ma solo la sua espressione
  - Chiunque potrebbe teoricamente analizzare un software e realizzarne un altro che svolga le stesse funzionalità, ma senza copiare il codice sorgente
  - Il software in quanto tale non è brevettabile. Codice della proprietà industriale, art. 45: *[...] Non sono considerate come invenzioni ai sensi del comma 1 in particolare: [...] i programmi per elaboratore;* (https://www.normattiva.it/uri-res/N2Ls?urn:nir:stato:decreto.legislativo:2005-02-10;30)
- Chi detiene i diritti d'autore può tenere nascosto il codice sorgente e fornire solo l'eseguibile. Questa è la «norma» per la maggior parte del software proprietario.

Software – Copyright

E’ lecito il reverse engineering del file eseguibile?

In Europa (Direttiva 2009/24/CE) il reverse engineering è lecito senza autorizzazione dell’autore quando necessario per l’interoperabilità con un altro software, se:

- Chi lo attua ha acquistato almeno una licenza d’uso (per fare il reverse engineering è necessario avere almeno una copia del software)
- Le informazioni per l’interoperabilità non sono già disponibili
- Il reverse engineering è limitato solo alle parti necessarie per l’interoperabilità

Software – Copyright

E’ lecito il reverse engineering del file eseguibile?

Direttiva 2009/24/CE, Art. 6:

1. Per gli atti di riproduzione del codice e di traduzione della sua forma ai sensi dell'articolo 4, paragrafo 1, lettere a) e b), non è necessaria l'autorizzazione del titolare dei diritti qualora l'esecuzione di tali atti al fine di modificare la forma del codice sia indispensabile per ottenere le informazioni necessarie per conseguire l'interoperabilità con altri programmi di un programma per elaboratore creato autonomamente, purché sussistano le seguenti condizioni:

a) tali atti siano eseguiti dal licenziatario o da un'altra persona che abbia il diritto di utilizzare una copia del programma o, per loro conto, da una persona abilitata a tal fine;

b) le informazioni necessarie per ottenere l'interoperabilità non siano già facilmente e rapidamente accessibili alle persone indicate alla lettera a); e

c) gli atti in questione siano limitati alle parti del programma originale necessarie per conseguire l'interoperabilità.

Software – Copyright

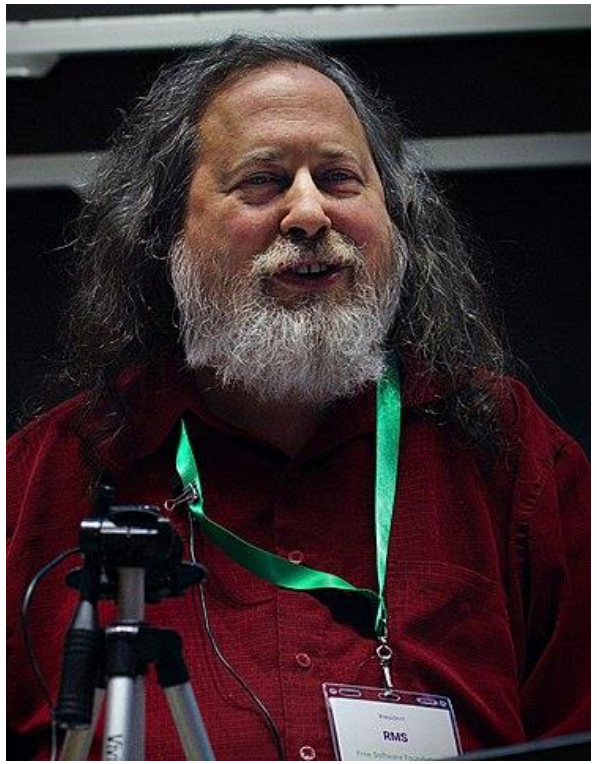
E’ lecito il reverse engineering del file eseguibile?

- In Europa non è possibile vietare il reverse engineering nel contratto di licenza con l’utente finale.
- Negli Stati Uniti, per le opere protette da copyright, e dunque anche per il software, esiste il «Fair Use» che consentirebbe il reverse engineering per garantire l’interoperabilità o a fini di analisi. Tuttavia, il reverse engineering può essere vietato dalle condizioni del contratto di licenza.

Free Software – Richard Stallman

- Programmatore e attivista del free software
- Nel 1985, fonda la Free Software Foundation (FSF)
- Autore di licenze per Free Software (es. GNU GPL – GNU General Public License)
- Free software (*software libero*): il software dovrebbe essere distribuito per dare agli utenti la libertà di usarlo, studiarlo, modificarlo e redistribuirlo

Richard Stallman



Free Software – Aneddoto

- 1980, Laboratorio di Intelligenza Artificiale del MIT: Stallman vi lavora come ricercatore, programmatore, hacker
- La stampante, una Xerox 9700, non forniva informazioni sulle code di stampa
- Sembra che Stallman, che aveva già modificato il driver di una precedente stampante in tal senso, chiese alla Xerox il sorgente del driver per aggiungere (gratuitamente) funzionalità e che l'azienda rifiutò
- Essere incapaci di apportare quella modifica era un problema, considerando che molti utenti stavano anche ad un piano diverso rispetto la stampante…

Free Software

- Open source: software rilasciato insieme al codice sorgente. Chiunque può studiarlo, modificarlo e redistribuirlo (anche a pagamento)
- Free Software: software rilasciato insieme al codice sorgente. Chiunque può studiarlo, modificarlo, redistribuirlo (anche a pagamento), ma il software deve essere distribuito mantenendolo libero, *senza assoggettarlo a restrizioni* (copyleft)

Free software ≠ Open Source

Licenze

- GNU GPL v3
- Apache v2.0
- BSD originale (no copyleft)
- MIT License (no copyleft)
- …

https://www.gnu.org/licenses

Licenze – GNU GPL v3

Libertà di

- usare il software per qualunque scopo,
- modificare il software per adattarlo ai propri bisogni,
- condividere il software con gli amici e i vicini, e
- condividere le modifiche effettuate

Guida rapida: https://www.gnu.org/licenses/quick-guide-gplv3.it.html
Testo completo: https://www.gnu.org/licenses/gpl-3.0.html