INFORMATICA

A.A. 2025-26

Ing. Paolo Querci

INF-01

Lezione 2 – 28 ottobre 2025

SOFTWARE

Insieme delle componenti immateriali (strato logico/intangibile) di un sistema elettronico di elaborazione: più propriamente le istruzioni di un programma; è contrapposto all'hardware, cioè la parte materiale (strato fisico/tangibile) dello stesso sistema.

L'ALGORITMO

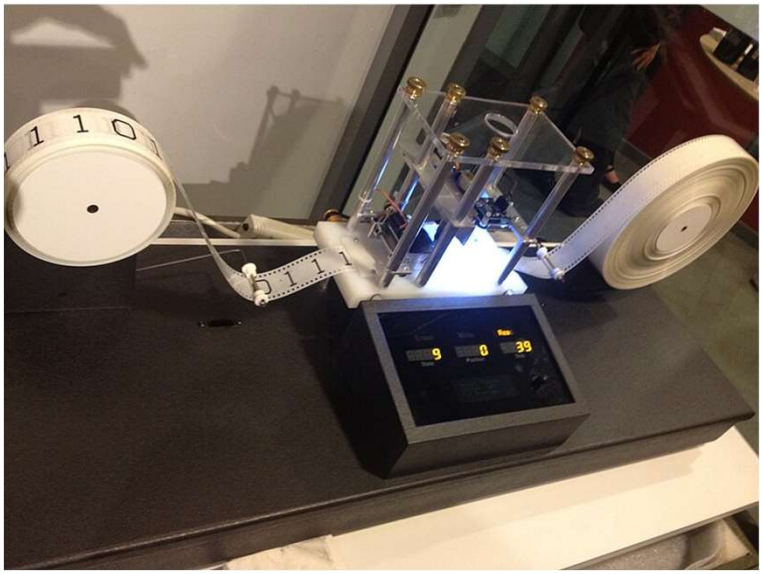
In matematica e informatica un **algoritmo** è la definizione di una **sequenza finita di operazioni** (dette anche istruzioni) che consente di risolvere tutti i quesiti di una stessa classe o di calcolare il risultato di un'espressione matematica.

Per funzionare bene, un algoritmo deve essere:

- finito (termina dopo un numero limitato di passi)
- deterministico (stesso input = stesso output)
- non ambiguo (ogni istruzione è chiara)
- generale (vale per ogni caso del problema considerato)

È la base dell'informatica: se c'è una **procedura** chiara che risolve qualcosa, può diventare un programma eseguibile da un computer.

La Macchina di Turing

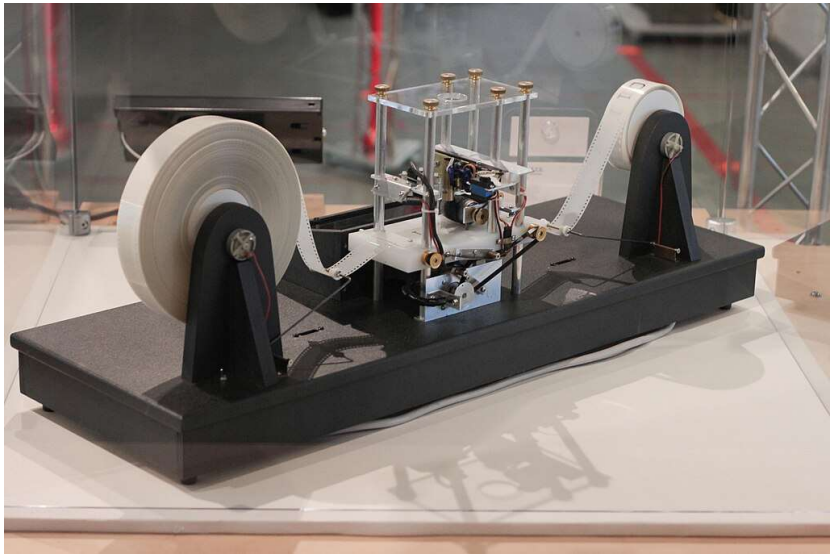


È un modello matematico (dal 1936, Alan Turing) che descrive una macchina astratta capace di leggere e scrivere su un nastro teoricamente infinito, seguendo regole semplici ma definite

Nonostante la sua apparente semplicità, questo dispositivo (teorico) è in grado di eseguire qualunque **algoritmo** eseguibile su un computer, rendendosi quindi un modello centrale nella teoria della calcolabilità

Da qui nasce l'idea di "macchina universale", ovvero una macchina astratta capace di eseguire qualsiasi altro programma codificato su di essa

La Macchina di Turing



La **macchina di Turing** è un modello teorico di computer. È composta da:

- un **nastro** diviso in caselle (infinite), su cui si possono **leggere e scrivere simboli**;
- una **testina** che può spostarsi a destra o a sinistra, **leggendo o scrivendo** il contenuto sul nastro e **seguendo regole precise** (le istruzioni).

La Macchina di Turing

👉 **Esempio: calcolo di 2 + 2**

Sul nastro troviamo:

`2 + 2`

La macchina:

Legge il primo numero ("2")

Legge il simbolo “+”

Legge il secondo numero ("2")

Scrive il risultato sul nastro:

`2 + 2 = 4`

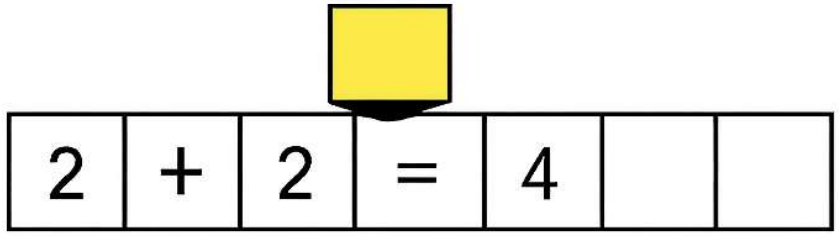
💡 La macchina non “capisce” cosa significhi 2 o 4: esegue **istruzioni meccaniche**, una dopo l’altra. Tutti i computer moderni si basano su questo stesso principio: **leggere dati, applicare regole, produrre risultati.**

La macchina di Turing – l'idea alla base dei computer

La **macchina di Turing** e un modello teorico di computer inventato nel 1936.

- un nastro diviso in caselle (infinite), su cui si possono **leggere e scrivere** simboli;
- una testina che puo spostarsi a destra o a sinistra, **leggendo** il contenuto e seguendo regole precise (le istruzioni).

Esempio: 2 + 2



1. Legge il primo numero
2. Legge il simbolo "+"
3. Legge il secondo numero
4. Scrive il risultato:

   2 + 2 = 4

Non "capisce" cosa significhi 2 o 4: segue istruzioni meccaniche, una dopo l'altra.
Turti i computer moderni si basano su questo stesso principio:

IL DIAGRAMMA DI FLUSSO

In informatica il **diagramma di flusso** (detto anche **flow chart**) è una rappresentazione grafica delle operazioni da eseguire per l'esecuzione di un **algoritmo**. Ogni singolo passo è visualizzato tramite una serie di simboli standard.

Un diagramma di flusso rappresenta graficamente i passi di un algoritmo: ogni operazione è indicata con un simbolo specifico, connesso da frecce che mostrano l'ordine delle azioni.

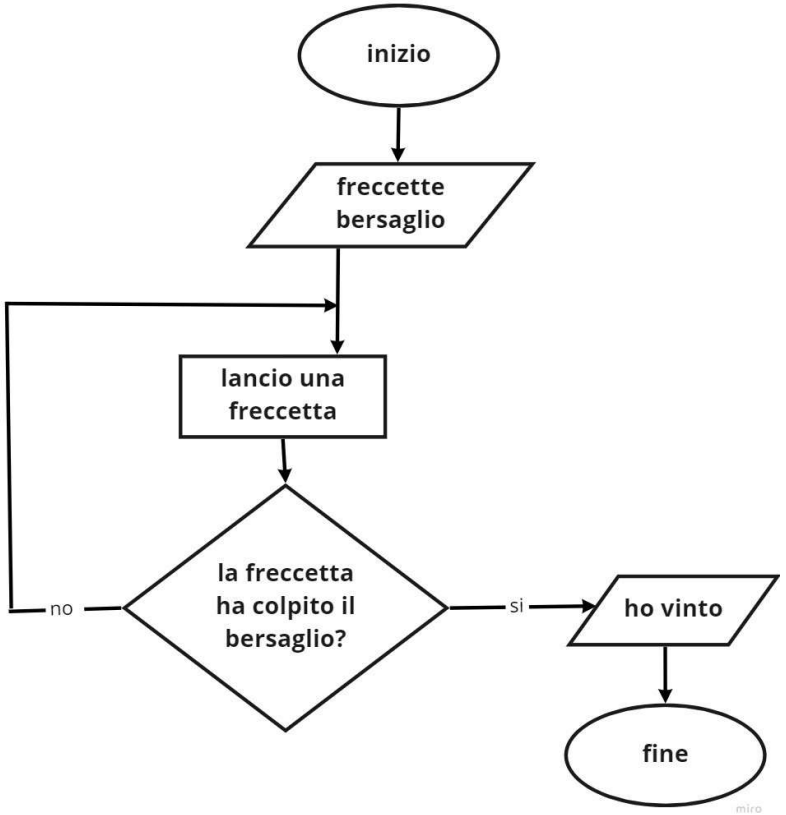
Serve a visualizzare in modo immediato la sequenza logica, rendendo facile distinguere operazioni, decisioni, input/output e l'inizio o fine del processo.

È uno strumento potente per pianificare, spiegare e anche rileggere — quasi come una "mappa visiva" del ragionamento.

IL DIAGRAMMA DI FLUSSO

Un **diagramma di flusso** è un tipo di **diagramma** che rappresenta un **flusso di lavoro** o un **processo**. Un diagramma di flusso può anche essere definito come una rappresentazione schematica di un algoritmo, un approccio graduale alla risoluzione di un compito.

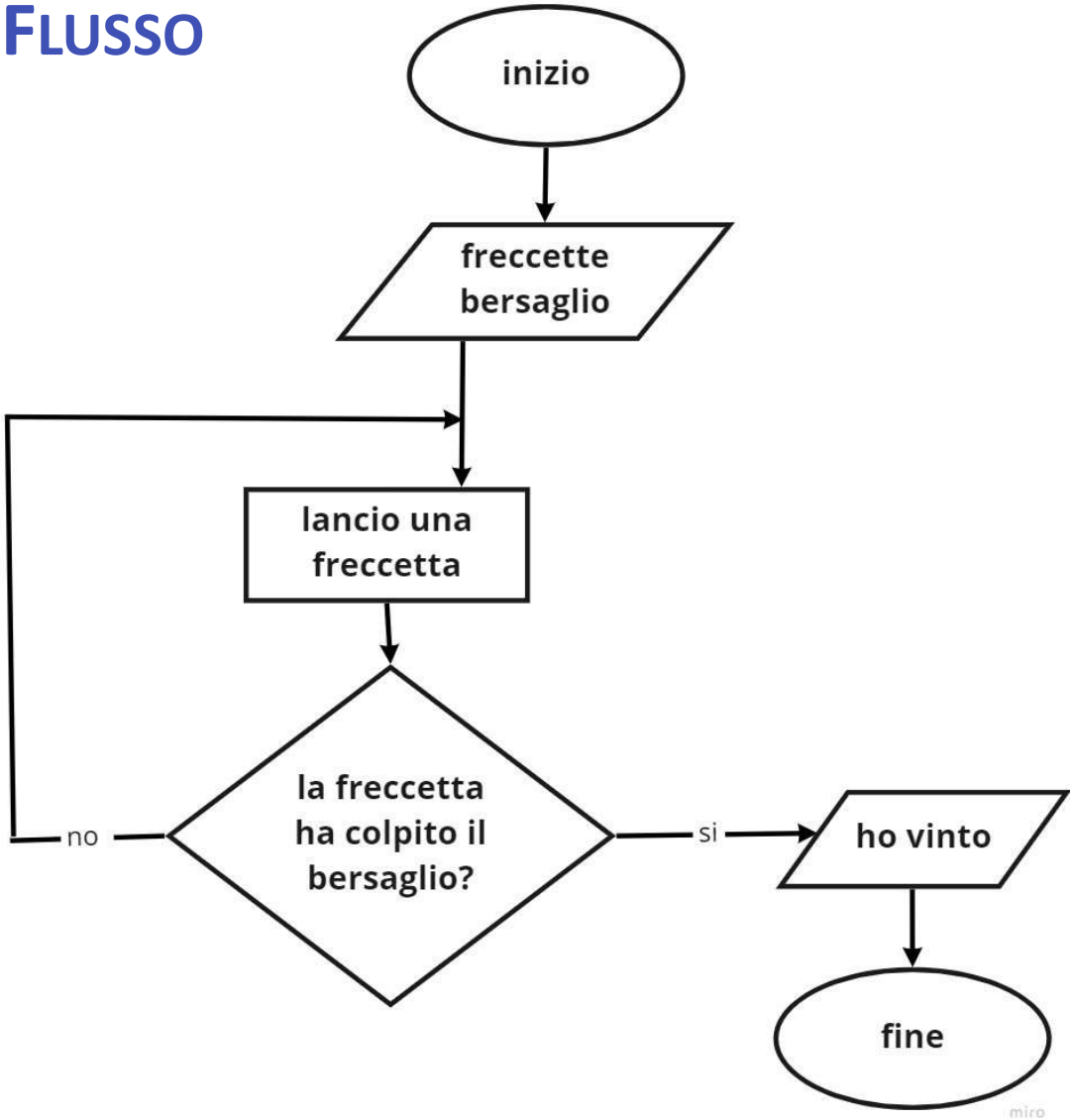
Il diagramma di flusso mostra i passaggi come caselle di vario tipo e il loro ordine collegando le caselle con le frecce. Questa rappresentazione schematica illustra un modello di soluzione a un dato problema. I diagrammi di flusso vengono utilizzati per analizzare, progettare, documentare o gestire un **processo** o un programma **in vari campi**.[1]



[fonte: Wikipedia eng]

[1] SEVOCAB: Software Systems Engineering Vocabulary. Term: Flow chart. Retrieved 31 July 2008.

Il Diagramma di Flusso

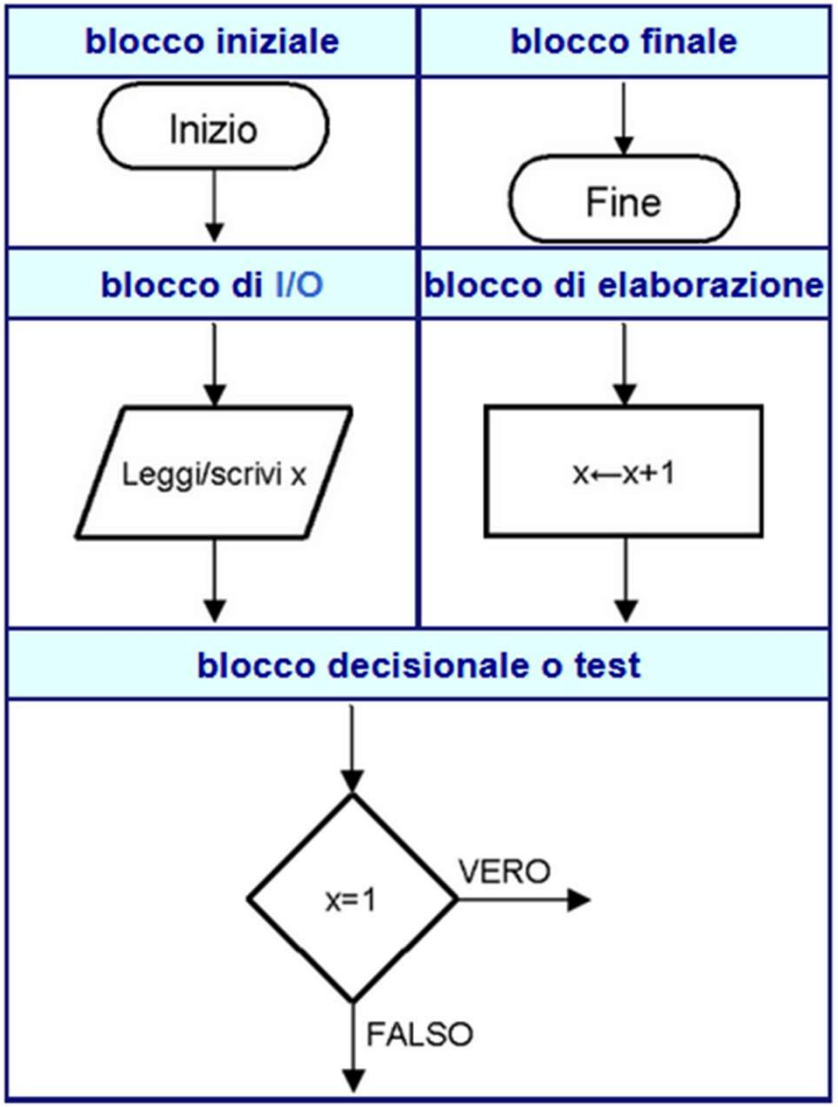


DIAGRAMMI DI FLUSSO: PRINCIPI E SIMBOLI DI BASE

I **diagrammi di flusso** possono essere disegnati con notazioni leggermente diverse, ma tutte si basano sullo stesso modello logico: una **sequenza ordinata di operazioni** collegate da **frecce direzionali**.

Il procedimento di lettura è come segue:

1. si parte dal **blocco iniziale**,
2. si segue la **freccia d'uscita** per passare al passo successivo,
3. si esegue l'operazione indicata nel blocco,
4. si prosegue finché non si raggiunge il **blocco finale**.

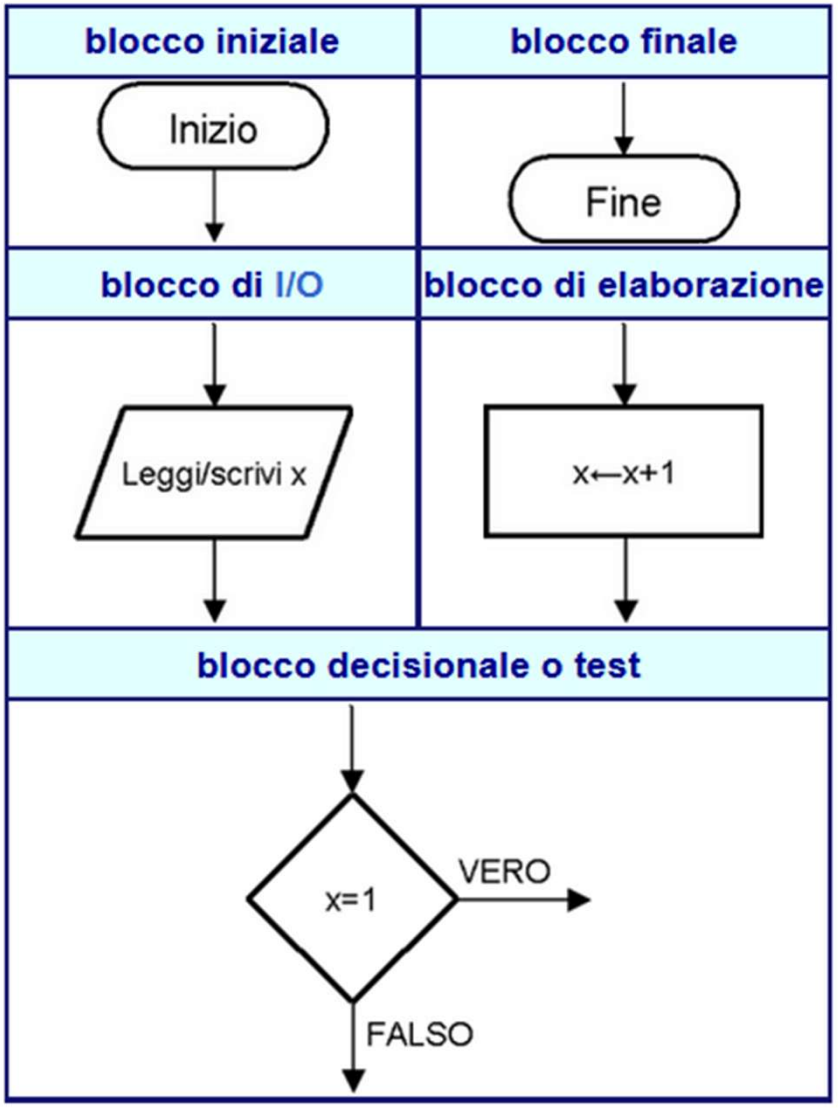


DIAGRAMMI DI FLUSSO: PRINCIPI E SIMBOLI DI BASE

All'interno del diagramma si distinguono tre categorie principali di operazioni:

- **azioni o elaborazioni**, che rappresentano un'attività da svolgere;
- **test o decisioni**, che deviano il flusso in base a una condizione (vero/falso);
- **ingressi e uscite**, che indicano lo scambio di dati con l'esterno.

La notazione più diffusa utilizza **pochi blocchi elementari** (tipicamente cinque), sufficienti per descrivere in modo chiaro qualsiasi algoritmo o processo.



ESEMIO DI DIAGRAMMA DI FLUSSO: CALCOLO DEL FATTORIALE

Consideriamo l'algoritmo di calcolo del fattoriale:

$$\text{fatt}(n) = n! = \begin{cases} 1, & \text{se } n = 0 \\ n \cdot (n-1)!, & \text{se } n \neq 0 \end{cases}$$

Il **fattoriale** di un numero intero positivo n, indicato con $n!$, è il **prodotto di tutti i numeri interi da 1 a n**. In altre parole, rappresenta quante **permutazioni diverse** si possono ottenere disponendo n oggetti differenti in ordine.

Ad esempio:

- $3! = 3 \times 2 \times 1 = 6$
- $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$
- Per convenzione, si definisce $0! = 1$, in modo che le formule combinatorie restino coerenti.

ESEMIO DI DIAGRAMMA DI FLUSSO: CALCOLO DEL FATTORIALE

Consideriamo l'algoritmo di calcolo del fattoriale:

$$\text{fatt}(n) = n! = \begin{cases} 1, & \text{se } n = 0 \\ n \cdot (n-1)!, & \text{se } n \neq 0 \end{cases}$$

Il **fattoriale** di un numero intero positivo n, indicato con $n!$, è il **prodotto di tutti i numeri interi da 1 a n**. In altre parole, rappresenta quante **permutazioni diverse** si possono ottenere disponendo n oggetti differenti in ordine.

Esempio semplice:

se abbiamo tre oggetti A, B, C, le permutazioni possibili sono:

ABC, ACB, BAC, BCA, CAB, CBA → cioè **6 in totale**.

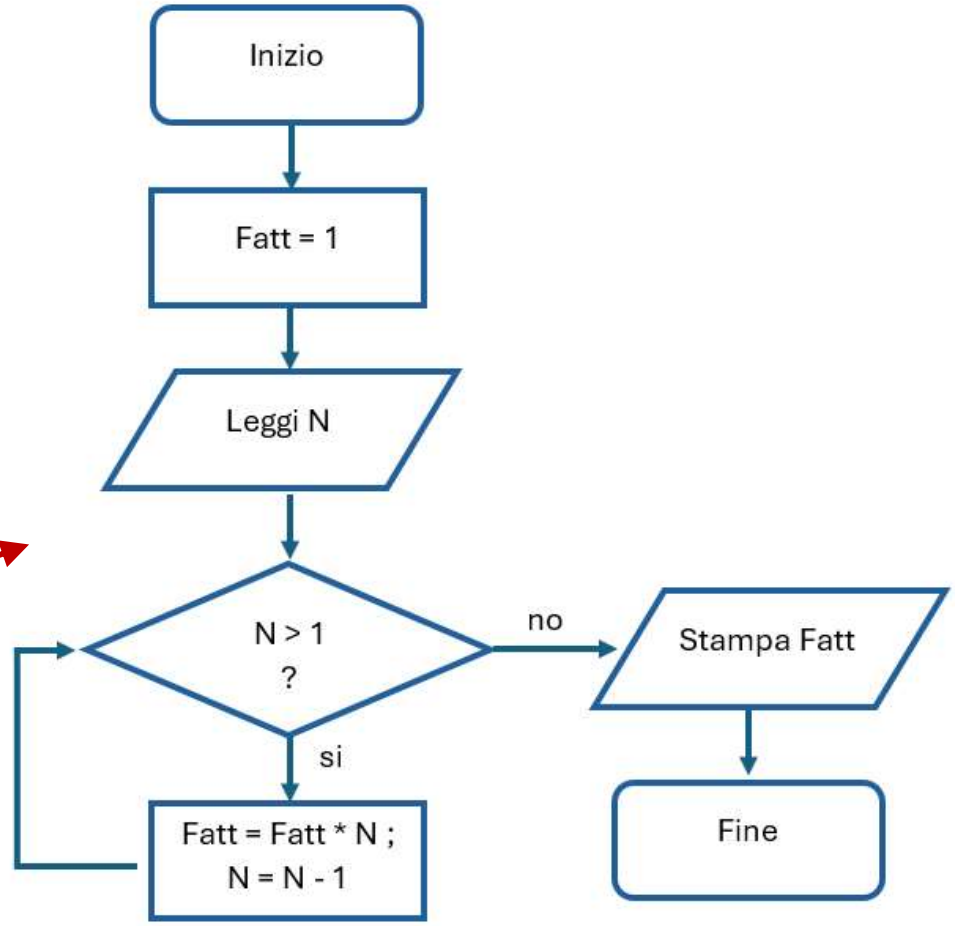
E infatti $3! = 6$.

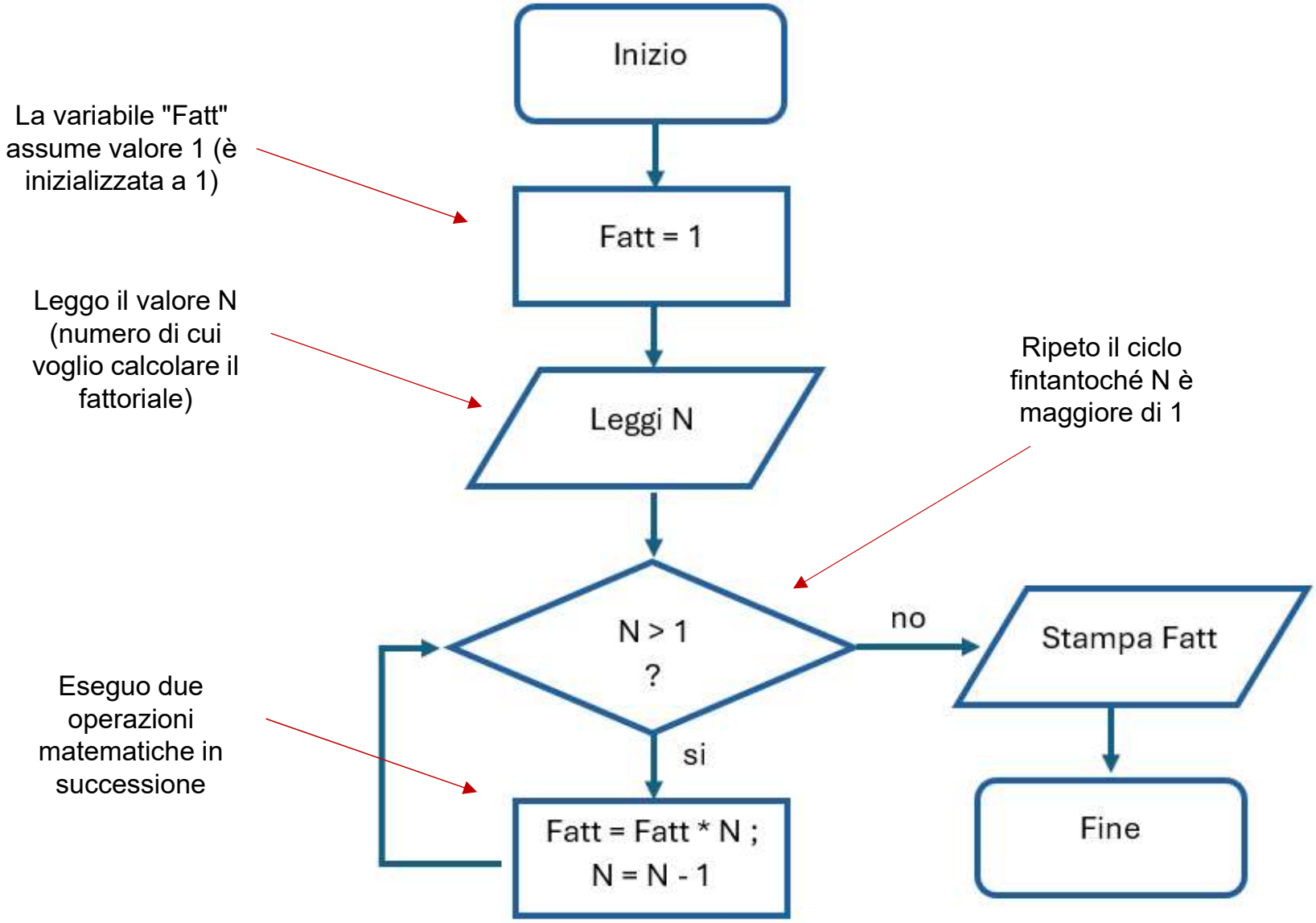
ESEMIO DI DIAGRAMMA DI FLUSSO: CALCOLO DEL FATTORIALE

Dalla formula

$$\text{fatt}(n) = n! = \begin{cases} 1, & \text{se } n = 0 \\ n \cdot (n-1)!, & \text{se } n \neq 0 \end{cases}$$

All'algoritmo




Inizio
La variabile "Fatt" assume valore 1 (è inizializzata a 1)
Fatt = 1
Leggo il valore N (numero di cui voglio calcolare il fattoriale)
Leggi N
Ripeto il ciclo fintantoché N è maggiore di 1
N > 1 ?
no
Stampa Fatt
Eseguo due operazioni matematiche in successione
si
Fatt = Fatt * N ;
N = N - 1
Fine

UN LINGUAGGIO COMUNE TRA INFORMATICA E INGEGNERIA GESTIONALE

I **diagrammi di flusso** non servono solo a descrivere algoritmi o programmi: sono uno strumento logico e visivo per **modellizzare i processi** di qualunque tipo.

In **informatica**, permettono di rappresentare la sequenza di operazioni che un calcolatore deve eseguire, con condizioni, cicli e diramazioni.

In **ingegneria gestionale**, la stessa logica viene applicata ai **processi aziendali o organizzativi**, per analizzare flussi di lavoro, punti decisionali e responsabilità operative.

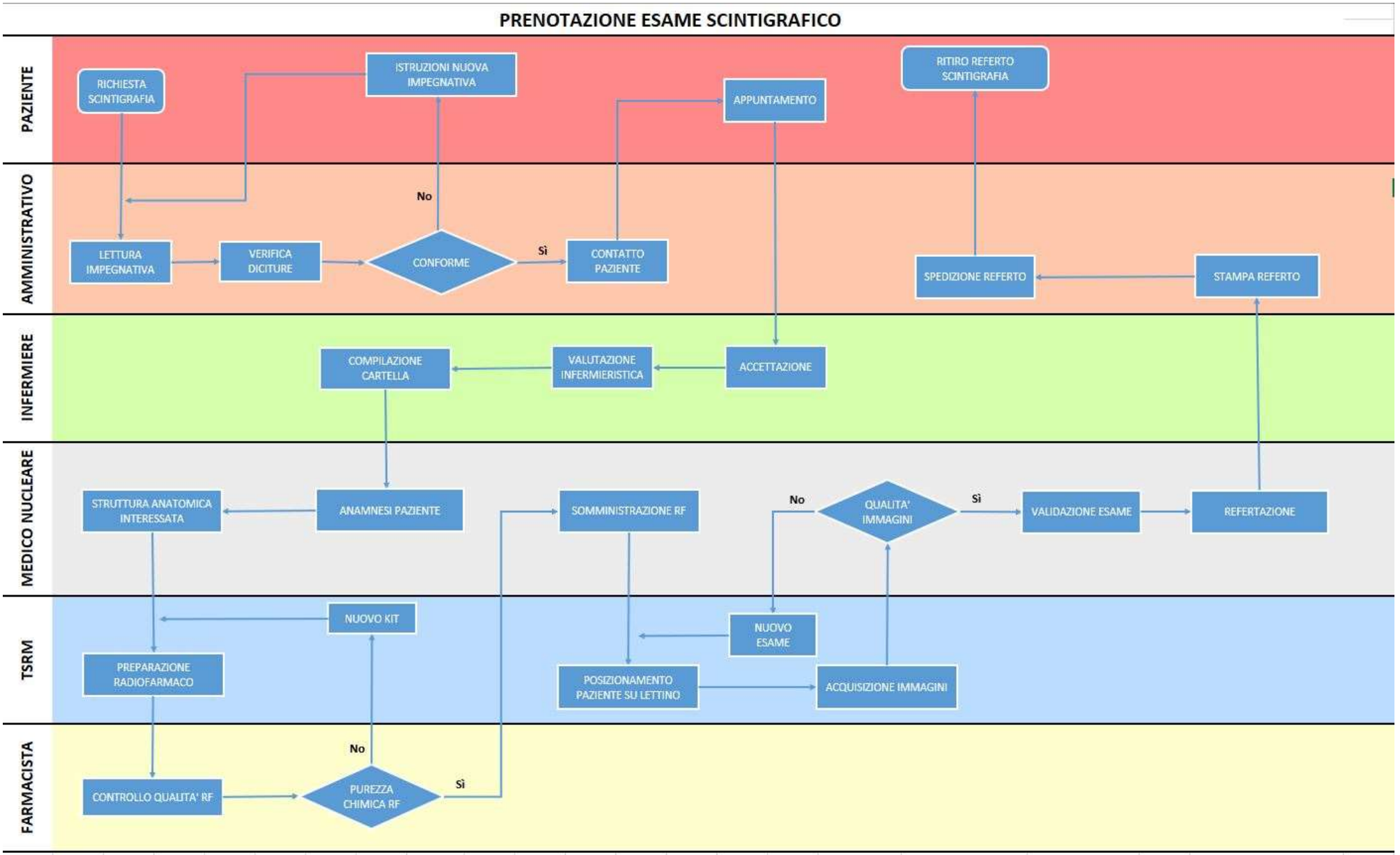
L'idea di fondo è la stessa:

- scomporre un'attività complessa in **passaggi elementari**,
- esplicitare le **relazioni di causa-effetto** tra le operazioni,
- individuare **collo di bottiglia, ridondanze o errori logici**.

Per questo i diagrammi di flusso rappresentano un **ponte naturale** tra informatica e organizzazione: due discipline diverse, ma basate sulla stessa esigenza di **chiarezza, sequenzialità e controllo**.

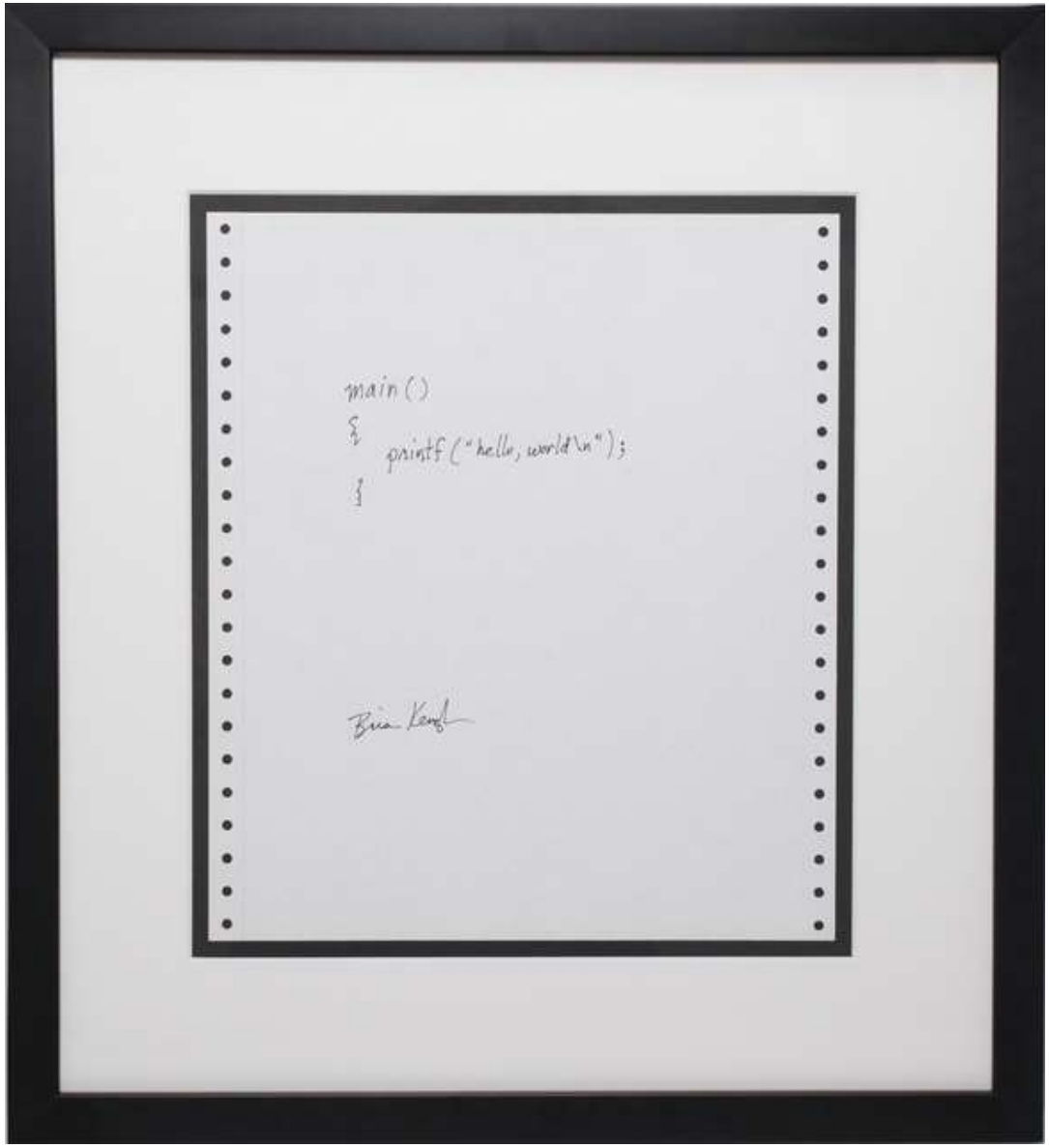
Nella prossima slide un esempio di quanto sopra.

PRENOTAZIONE ESAME SCINTIGRAFICO



COSA È UN PROGRAMMA?

Un **programma**, in informatica, è un **procedimento algoritmico** applicato ad un problema dato da automatizzare, tipicamente codificato in una serie di linee di codice scritte in un certo linguaggio di programmazione da un programmatore in fase di programmazione a formare un **software**, che può essere eseguito da un elaboratore, ricevendo in **input** determinati dati e restituendo in **output** gli eventuali risultati ottenuti a seguito dell'esecuzione/elaborazione delle sue istruzioni.



DALL' IDEA AL SOFTWARE

Ogni **programma** nasce da un'**idea**: un problema da risolvere o un compito da automatizzare.

Per passare dall'idea alla realizzazione serve una **sequenza logica di passi**, chiamata **algoritmo**.

Un **algoritmo** è un insieme di **istruzioni ordinate** che indicano come ottenere un risultato.

Quando un algoritmo viene **tradotto in un linguaggio di programmazione**, diventa un **programma** — o una parte di esso, se il software è composto da più algoritmi.

👉 L'insieme di tutti i programmi presenti su un computer costituisce il **software**.

COMPONENTI DEL SOFTWARE

Il **software** di un computer comprende tutto ciò che è **intangibile**, cioè non materiale:

sono **i programmi**, le **applicazioni**, i **dati** e le **istruzioni** che dicono al computer cosa fare.

Si distingue in due grandi categorie principali:

Software di sistema → gestisce il funzionamento del computer (es. sistema operativo, driver).

Software applicativo → permette di svolgere attività specifiche (es. scrivere, calcolare, comunicare).

👉 Il software dà vita all'hardware: senza software, anche il computer più potente resterebbe inerte.

COME INTERAGISCONO SOFTWARE E HARDWARE

Il **software** comunica con l'**hardware** attraverso una serie di **istruzioni codificate**.

Queste istruzioni vengono interpretate e trasformate in segnali elettrici che attivano i componenti fisici del computer.

Il processo avviene in più passaggi:

- Il **programma** è scritto in un **linguaggio di programmazione** comprensibile all'uomo ("**codice sorgente**").
- Un **compilatore** o un **interprete** traduce questo linguaggio nel **linguaggio macchina**, formato da zeri e uno ("**codice oggetto**").
- L'hardware esegue le istruzioni una dopo l'altra, producendo il risultato richiesto.

👉 In sintesi: **il software dà gli ordini**, **l'hardware li esegue**.

LINGUAGGI DI PROGRAMMAZIONE: ALTO E BASSO LIVELLO

I **linguaggi di programmazione** permettono di scrivere algoritmi in una forma comprensibile e traducibile dal computer.

Si distinguono in base alla **vicinanza all'uomo o alla macchina**:

Linguaggi di basso livello → molto vicini al linguaggio macchina, difficili da leggere ma estremamente efficienti (es. Assembly).

Linguaggi di alto livello → più simili al linguaggio naturale, facili da scrivere e comprendere (es. Python, C, Java).

Il **compilatore** traduce il linguaggio di alto livello in **codice binario**, che può essere eseguito dall'hardware.

👉 In pratica, più il linguaggio è "alto", più è facile per noi — ma serve una traduzione per la macchina.

COMPILATORE E INTERPRETE

Per poter essere eseguito, un programma deve essere **tradotto in linguaggio macchina**, che il computer può comprendere direttamente.

La **compilazione** è il metodo tradizionale di traduzione:

Il **compilatore** analizza **l'intero codice sorgente** e lo converte in **un file eseguibile**.

Una volta compilato, il programma può essere eseguito **più volte senza bisogno del codice sorgente**.

✅ **Vantaggi:** esecuzione molto veloce, sicurezza del codice, efficienza.

❌ **Svantaggi:** meno flessibilità e tempi di compilazione più lunghi.

💡 **Esempi di linguaggi compilati: C, C++, Pascal, Fortran**

👉 I linguaggi compilati sono ideali per **applicazioni ad alte prestazioni** (calcolo scientifico, sistemi operativi, videogiochi, ecc.).

CALCOLO FATTORIALE IN LINGUAGGIO C

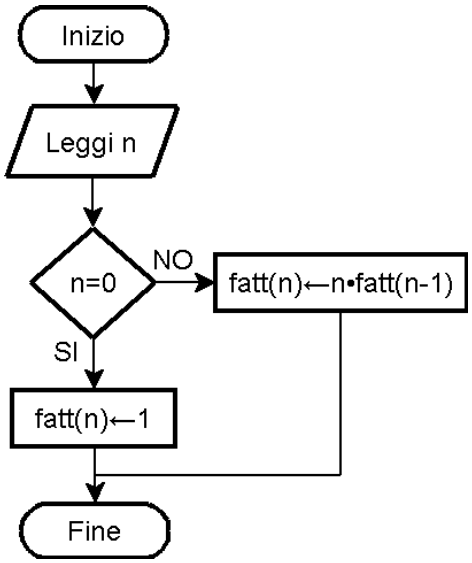
```c
#include <stdio.h>

// Calcolo iterativo del fattoriale
int fattoriale(int n) {
    int risultato = 1;
    for (int i = 1; i <= n; i++) {
        risultato *= i;   // moltiplica risultato per i
    }
    return risultato;
}

int main() {
    int n;
    printf("Inserisci un numero intero positivo: ");
    scanf("%d", &n);

    if (n < 0)
        printf("Il fattoriale non è definito per numeri negativi.\n");
    else
        printf("Il fattoriale di %d è %d\n", n, fattoriale(n));

    return 0;
}
```

```c
#include <stdio.h>

// Calcolo iterativo del fattoriale
int fattoriale(int n) {
    int risultato = 1;
    for (int i = 1; i <= n; i++) {
        risultato *= i;   // moltiplica risultato per i
    }
    return risultato;
}

int main() {
    int n;
    printf("Inserisci un numero intero positivo: ");
    scanf("%d", &n);

    if (n < 0)
        printf("Il fattoriale non è definito per numeri negativi.\n");
    else
        printf("Il fattoriale di %d è %d\n", n, fattoriale(n));

    return 0;
}
```

CALCOLO FATTORIALE IN ASSEMBLY

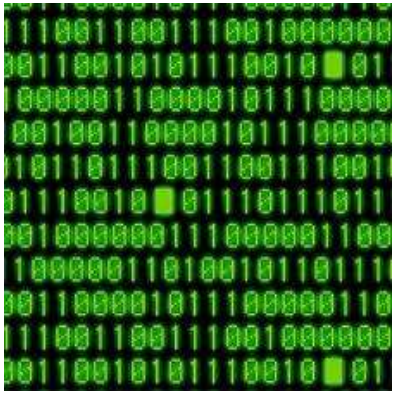
```
fattoriale:
    mov eax, 1              ; risultato = 1
    mov ecx, edi            ; ecx = n
loop_start:
    test ecx, ecx           ; verifica se n == 0
    jz done                 ; se sì, termina
    imul eax, ecx           ; risultato *= n
    dec ecx                 ; n--
    jmp loop_start
done:
    ret
```

DAL DIAGRAMMA DI FLUSSO AL CODICE MACCHINA




```c
#include <stdio.h>

// Calcolo iterativo del fattoriale
int fattoriale(int n) {
    int risultato = 1;
    for (int i = 1; i <= n; i++) {
        risultato *= i;   // moltiplica risultato per i
    }
    return risultato;
}

int main() {
    int n;
    printf("Inserisci un numero intero positivo: ");
    scanf("%d", &n);

    if (n < 0)
        printf("Il fattoriale non è definito per numeri negativi.\n");
    else
        printf("Il fattoriale di %d è %d\n", n, fattoriale(n));

    return 0;
}
```

```asm
fattoriale:
    mov eax, 1          ; risultato = 1
    mov ecx, edi        ; ecx = n
loop_start:
    test ecx, ecx       ; verifica se n == 0
    jz done             ; se sì, termina
    imul eax, ecx       ; risultato *= n
    dec ecx             ; n--
    jmp loop_start
done:
    ret
```



COMPILATORE E INTERPRETE

In alternativa alla compilazione, il programma può essere **interpretato**, cioè tradotto **una riga alla volta** durante l'esecuzione.

L'**interprete** legge il codice, lo traduce e lo esegue immediatamente.

Non crea file eseguibili permanenti, ma consente di **vedere subito i risultati**.

✅ **Vantaggi:** semplicità, rapidità di test e debugging.

❌ **Svantaggi:** esecuzione più lenta, dipendenza dall'interprete.

💡 **Esempi di linguaggi interpretati: BASIC, PHP, Bash**

Esistono anche linguaggi **ibridi**, che uniscono i due approcci:

il codice è prima **compilato in bytecode**, poi **interpretato** da una **macchina virtuale**.

💡 **Esempi: Python, Java, C#**

👉 In sintesi: **compilazione, interpretazione e ibridazione** sono tre modi diversi di **tradurre un algoritmo in azioni reali sul computer**.

COSA CARATTERIZZA UN PROGRAMMA

Un programma informatico può essere descritto in base a diversi aspetti fondamentali:

Linguaggio di programmazione utilizzato per scriverlo

Modalità di esecuzione: compilato, interpretato o ibrido

Piattaforma di destinazione (tipo di CPU e sistema operativo)

Portabilità: se è **indipendente dalla piattaforma** (es. *Java*)

Ambiente di esecuzione: locale, in rete (*intranet*), *web* o *cloud*

👉 Questi elementi determinano **dove, come e con quali prestazioni** il programma potrà essere eseguito.

SISTEMI OPERATIVI

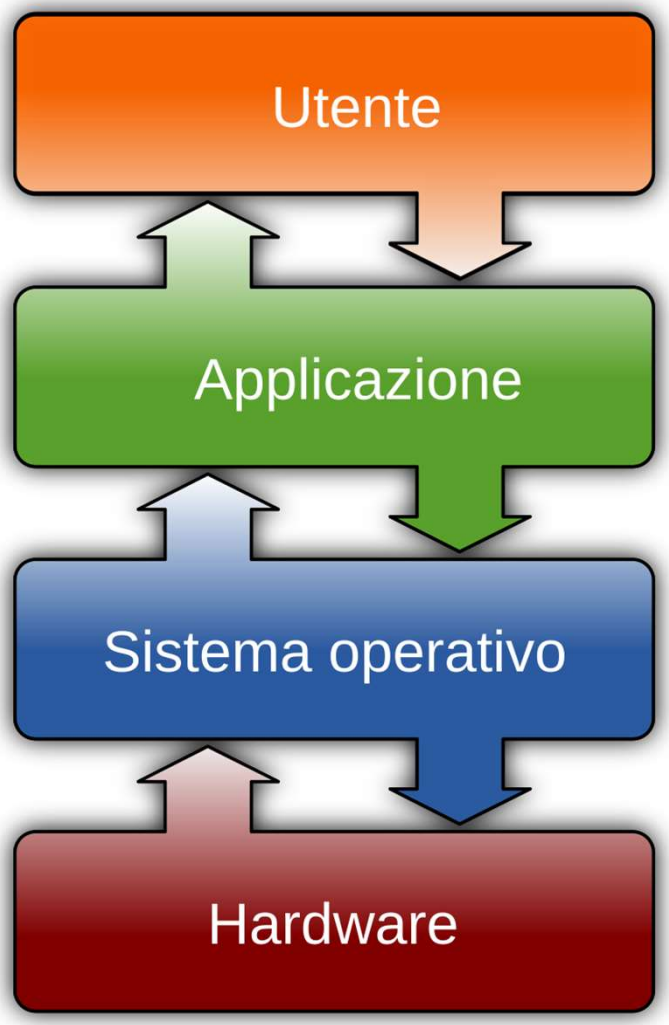
Windows

macOS

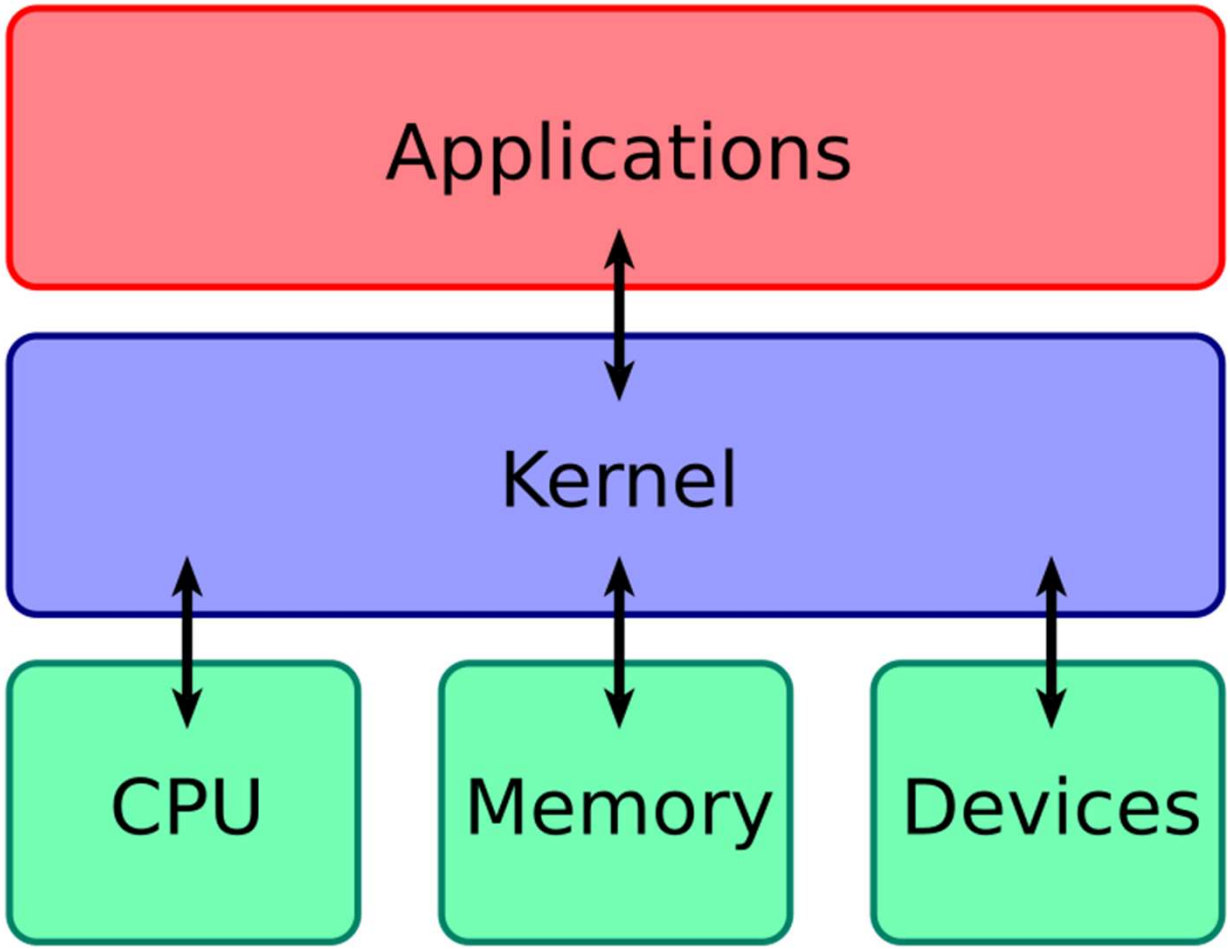
SISTEMI OPERATIVI

SISTEMI OPERATIVI

Un **sistema operativo** (abbreviato in **SO**, *OS* in inglese), in informatica, è un software di base, detto anche piattaforma operativa (composto normalmente da più sottosistemi o componenti software: **kernel** (*nocciolo* in inglese), scheduler, file system, gestore della memoria, gestore delle periferiche, interfaccia utente e spooler di stampa), che gestisce le risorse hardware e software della macchina, fornendo servizi di base ai **software applicativi**



SISTEMI OPERATIVI



SISTEMI OPERATIVI E PROGRAMMI

Un **programma** è costituito dal **codice oggetto** generato dalla compilazione del codice sorgente, ed è normalmente salvato sotto forma di uno o più **file**. Esso è un'entità statica, che rimane immutata durante l'esecuzione.

Il **processo** è l'entità utilizzata dal sistema operativo per rappresentare una specifica esecuzione di un programma. In pratica un processo è un programma quando è in esecuzione.

SISTEMI OPERATIVI E PROGRAMMI

Un processo ha sempre almeno un **thread** (se stesso), ma in alcuni casi un processo può avere più thread che vengono eseguiti in parallelo.

Una differenza sostanziale fra thread e processi consiste nel modo con cui essi condividono le risorse: mentre i **processi** sono di solito fra loro indipendenti, utilizzando diverse aree di memoria ed interagendo soltanto mediante appositi meccanismi di comunicazione messi a disposizione dal sistema, al contrario i **thread** di un processo tipicamente condividono le medesime informazioni di stato, la memoria ed altre risorse di sistema.

SISTEMI OPERATIVI E PROGRAMMI

Un processo ha sempre almeno un **thread** (se stesso), ma in alcuni casi un processo può avere più thread che vengono eseguiti in parallelo.

Il termine inglese rende bene l'idea, in quanto si rifà visivamente al concetto di fune composta da vari fili attorcigliati: se la fune è il processo in esecuzione, allora i singoli fili che la compongono sono i thread.

MULTITASKING

Con **multitasking** (in italiano **multiprocessualità**), in informatica, si indica la capacità di un software di eseguire più programmi contemporaneamente: se ad esempio viene chiesto al sistema di eseguire contemporaneamente due processi A e B, la CPU eseguirà per qualche istante di tempo il processo A, poi per qualche istante successivo il processo B, poi tornerà a eseguire il processo A e così via; il passaggio dal processo A al processo B e viceversa viene definito "**commutazione di contesto**" (**context switch**).

MULTITASKING

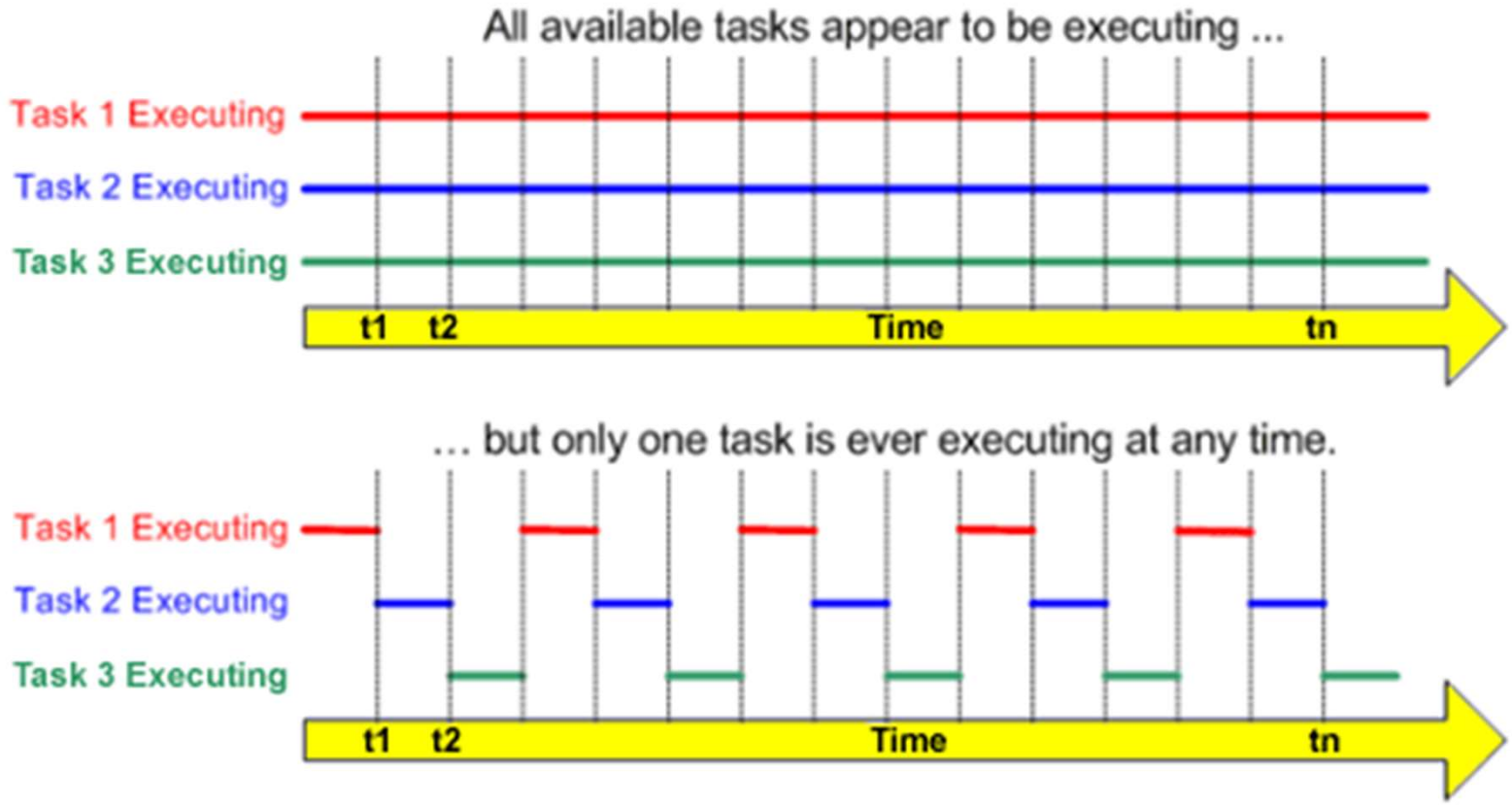
Le decisioni riguardanti l'esecuzione di un cambio di contesto tra due programmi vengono intraprese da un componente software del sistema operativo, lo **scheduler**, il quale invierà le proprie decisioni date principalmente dal **nice** (definisce le priorità dei processi) a un altro modulo del sistema operativo, il **dispatcher** che eseguirà effettivamente il cambio di contesto: a seconda di quale strategia di servizio (***algoritmo di scheduling***) venga seguita, lo scheduler controlla la ripartizione del tempo di CPU tra tutti i processi attivi.

MULTITASKING

Esistono due principali tecniche di controllo di termine e pausa del multitasking: il vecchio senza prelazione (**cooperative**), utilizzato su macOS fino alla versione 9 e Windows 3.0 e 3.1 e il nuovo con prelazione (**preemptive**), utilizzato su tutti i sistemi operativi contemporanei.

MULTITASKING

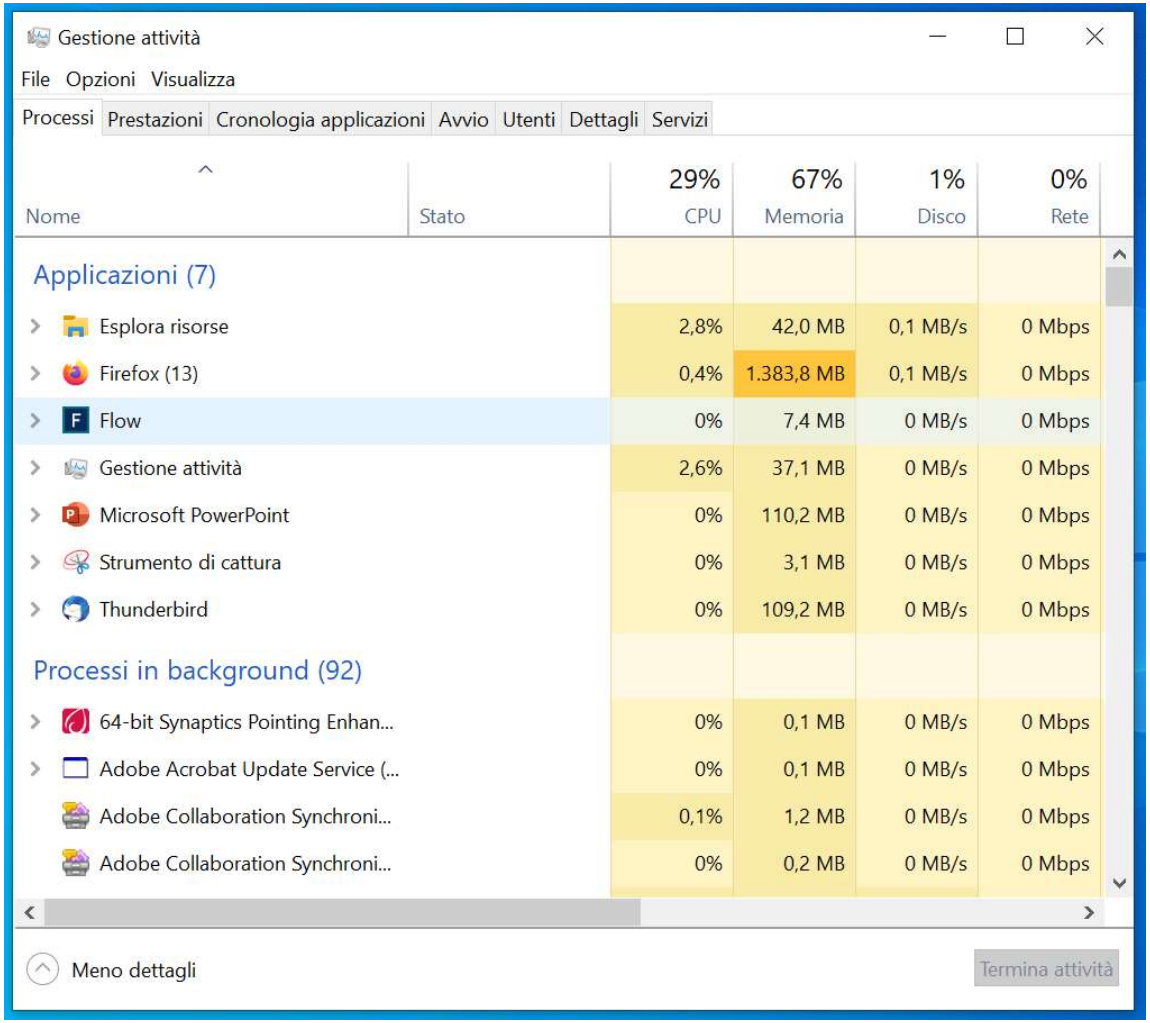
Processi



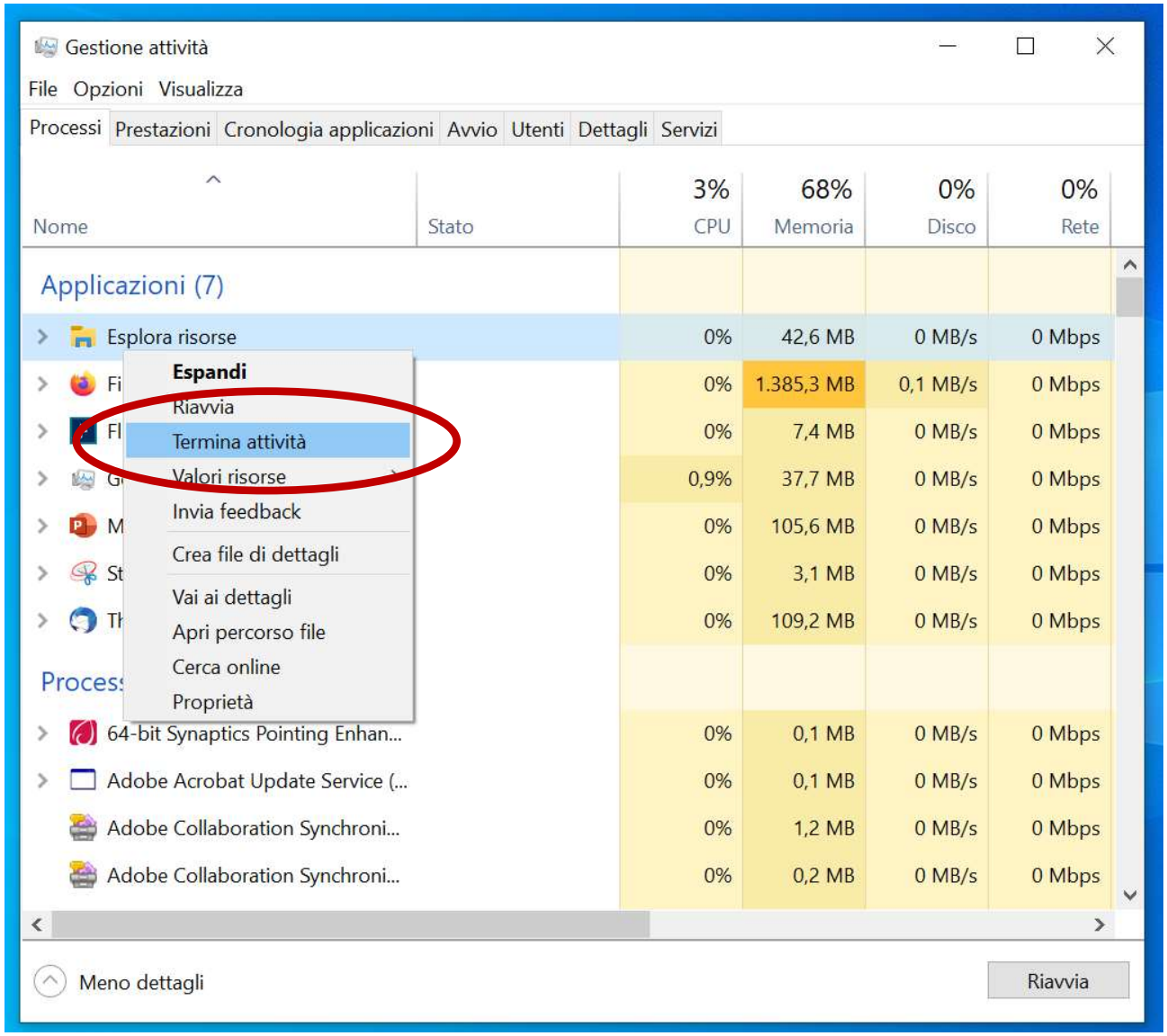
In ogni istante viene eseguito **un solo processo**

TASK MANAGER IN WINDOWS

In Windows 10 si accede mediante: CTRL – ALT – CANC , quindi selezionare "Task manager"
Oppure mediante tasto DX su barra applicazioni (in basso), quindi selezionare "Task manager"



TASK MANAGER IN WINDOWS



GRAZIE PER L'ATTENZIONE