

# Informatica

## Algoritmi e programmazione

Thomas Barbera
thomas.barbera@unimib.it

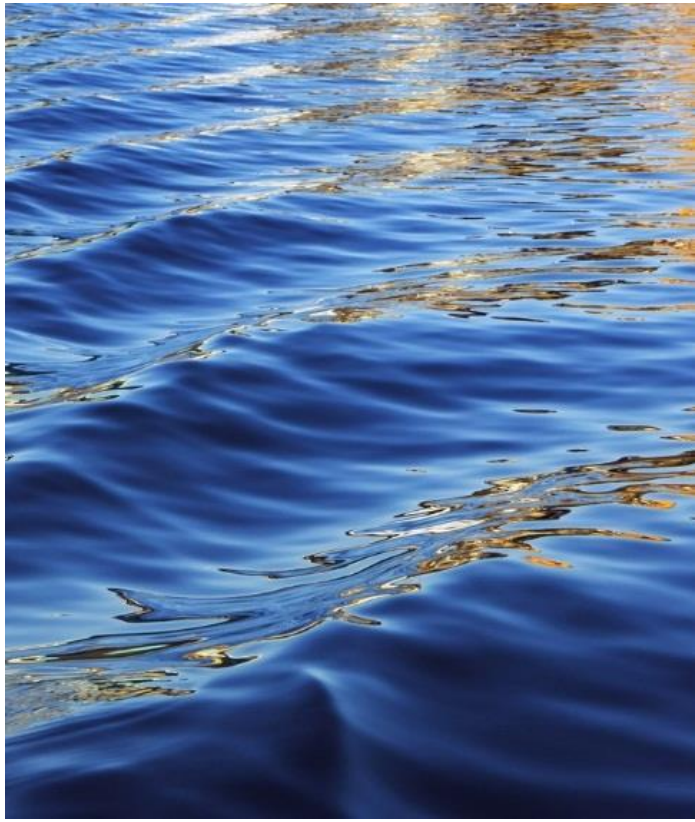
Riadattamento delle slides di Marco Viviani e Simone Zini







# Prima parte:
# Introduzione agli algoritmi

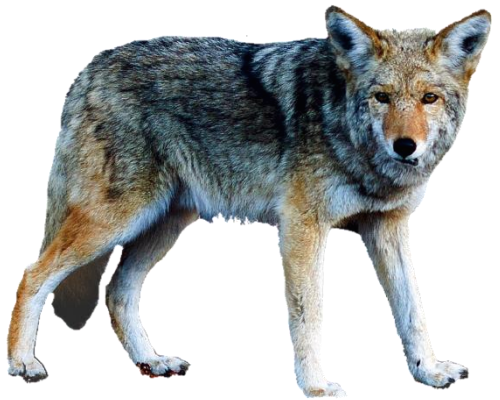


# Problema

- Trasporto di una pecora, di un lupo e di un cavolo sull'altra sponda di un fiume con una barca

1. Il lupo non può rimanere solo con la pecora: la mangerebbe!
2. La pecora non può rimanere sola con il cavolo: lo mangerebbe!
3. Sulla piccola barca può salire, oltre al barcaiolo, solo un animale alla volta (o il cavolo)







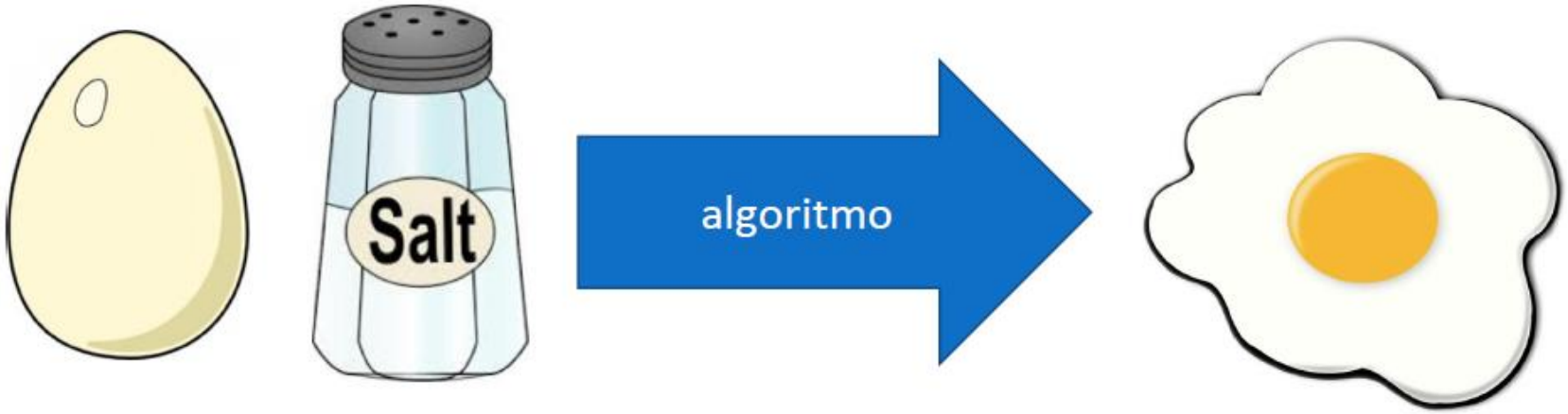
# Il concetto di algoritmo

1. Porta la pecora sull'altra sponda
2. Torna indietro
3. Porta il cavolo sull'altra sponda
4. Riporta la pecora indietro
5. Porta il lupo sull'altra sponda
6. Torna indietro
7. Porta la pecora sull'altra sponda

- Si tratta di un **algoritmo**!

# Algoritmo

- Procedimento che a partire da uno **stato iniziale**, consente di ottenere in un **tempo finito** un **risultato atteso** eseguendo un **insieme di operazioni** descritte in maniera **completa** e **non ambigua**



# Approccio «black-box»

- Per risolvere un problema si devono prima individuare
  - Le **informazioni e i dati** (noti)
  - I **risultati** che si vogliono ottenere (incogniti)




- Si procede individuando un procedimento (o **algoritmo**)

- L'algoritmo è **scomposto in operazioni più semplici** che producono **risultati intermedi**

# Le 5 proprietà degli algoritmi

**1. Atomicità**

- I passi costituenti devono essere "elementari", ovvero non ulteriormente scomponibili

**2. Non ambiguità**

- I passi costituenti devono essere interpretabili in modo diretto e univoco dall'*esecutore*, sia esso umano o artificiale

**3. Finitezza**

- L'algoritmo deve essere composto da un numero finito di passi e richiedere una quantità finita di dati in ingresso

# Le 5 proprietà degli algoritmi

**4. Terminazione**

- l'esecuzione deve avere termine dopo un tempo finito

**5. Effettività**

- l'esecuzione deve portare a un risultato univoco

# Una proprietà aggiuntiva

- Agli algoritmi è spesso richiesta una sesta proprietà:

**6. Efficienza**

- L'algoritmo è in grado di risolvere un problema utilizzando il minore tempo e/o impiegando il numero minore di risorse possibili.

# Definizione sintetica di algoritmo

**Algoritmo**

Un algoritmo è un insieme ordinato di passi eseguibili e non ambigui che giunge certamente a terminazione

**Da imparare!!!**

# Un esempio di algoritmo

- Problema: **stampare i primi 10 multipli di $x$**

- Il problema è semplice e chiaro:
  - **un solo dato in input** (un parametro che caratterizza l'istanza del problema):
    - il numero $x$;
  - una **costante** del problema: il numero di multipli da stampare, 10

- La strategia risolutiva (in **linguaggio naturale**) è la seguente:
  - iniziando dal numero $x$ letto in input, per 10 volte verrà calcolato e stampato il multiplo di $x$, moltiplicando $x$ per una **variabile** $i$ il cui valore verrà incrementato di una unità ad ogni passo, da 1 fino a 10

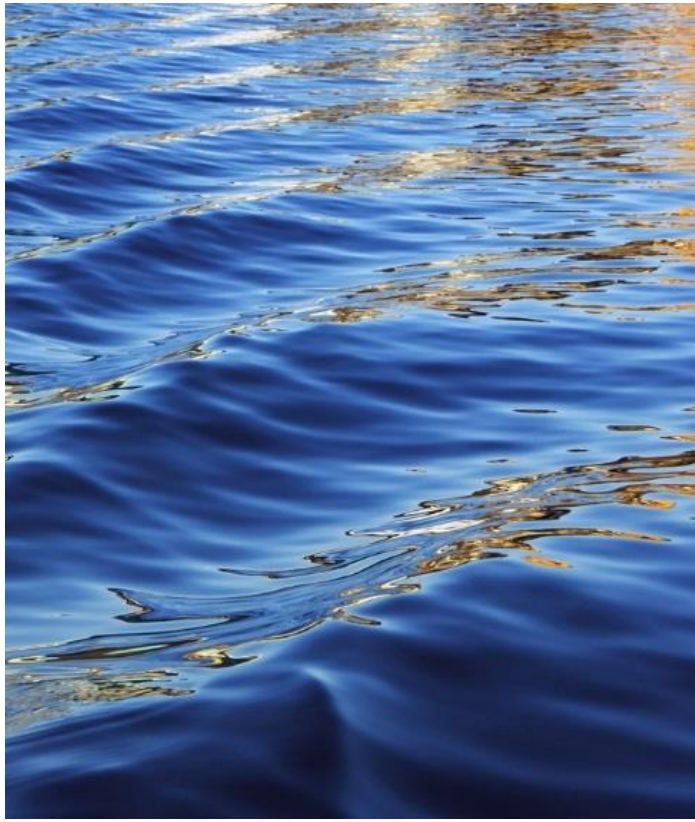
# Un esempio di algoritmo

Possiamo ricondurre la strategia al seguente algoritmo (**pseudo-codice**):

1. Leggi in input un numero e chiama $x$ il numero letto;
2. Assegna il valore 1 alla variabile $i$;
3. Calcola $x * i$ e assegna alla variabile $y$ il risultato;
4. Stampa $y$;
5. Calcola $i + 1$ e assegna alla variabile $i$ il risultato;
6. Se $i \leq 10$ allora vai al passo 3 altrimenti prosegui;
7. Fermati.



# Verso la programmazione



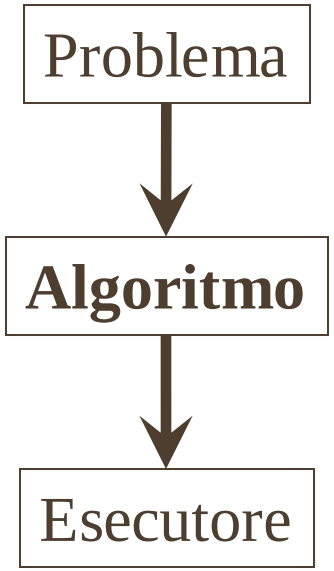
# Cos'è la programmazione?

- Problema molto ampio, nostro scopo essere in grado di:
  - Realizzare un **programma**
  - per risolvere un dato **problema**
  - mediante un **computer**

# Il problema

- Quando ci troviamo di fronte ad un problema?
  - Ogni volta che ci troviamo di fronte alla necessità di ottenere qualcosa che non sia direttamente raggiungibile a partire da dei dati (**il risultato**)
- Chi risolve il problema?
  - Il "*problem-solver*" è colui che si occupa di trovare una soluzione (l'**algoritmo**) che ci consenta di raggiungere il risultato a partire dai dati di ingresso
- Chi mette in pratica l'algoritmo?
  - Un **esecutore** che può essere umano o meccanico o elettronico

# Le tappe




*Formulazione del problema*

*Costruzione del metodo di soluzione*

*Esecuzione della soluzione*

# Descrivere l'algoritmo

- Il linguaggio con cui si descrive un algoritmo **non è** un linguaggio di programmazione

- Esistono
  - **linguaggi grafici**
    - fanno uso di simboli grafici
  - **linguaggi lineari**
    - fanno uso di simboli alfanumerici

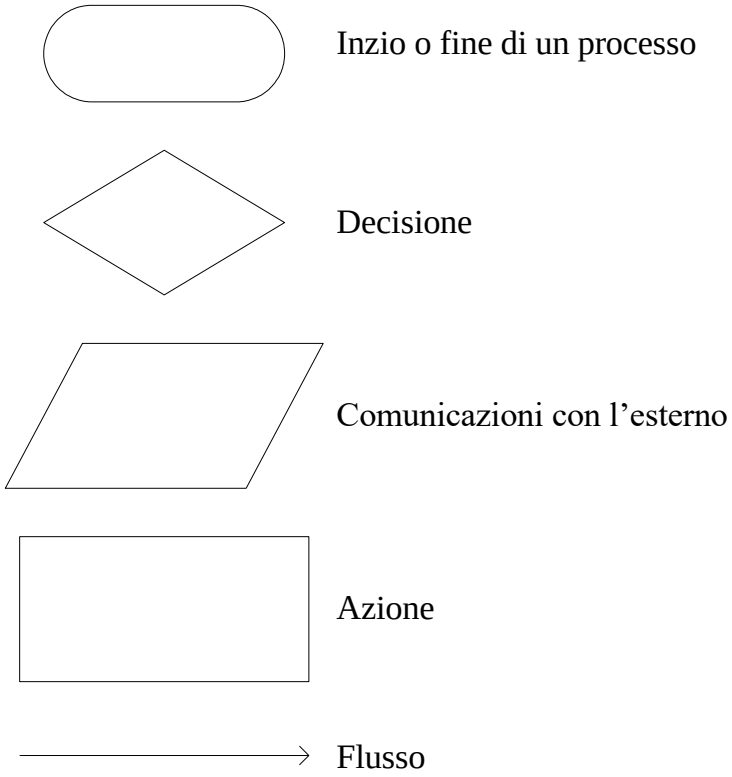
# Esempi di linguaggi

Alcuni esempi di linguaggi di descrizione degli algoritmi:

- **Diagrammi di flusso** (*Flow chart*)
- Carte strutturate
- Rappresentazioni ad albero

# I diagrammi di flusso

- Vanno bene nel momento in cui occorre descrivere un **processo sequenziale**
- Le azioni **non si sovrappongono**
- I **simboli** usati sono:

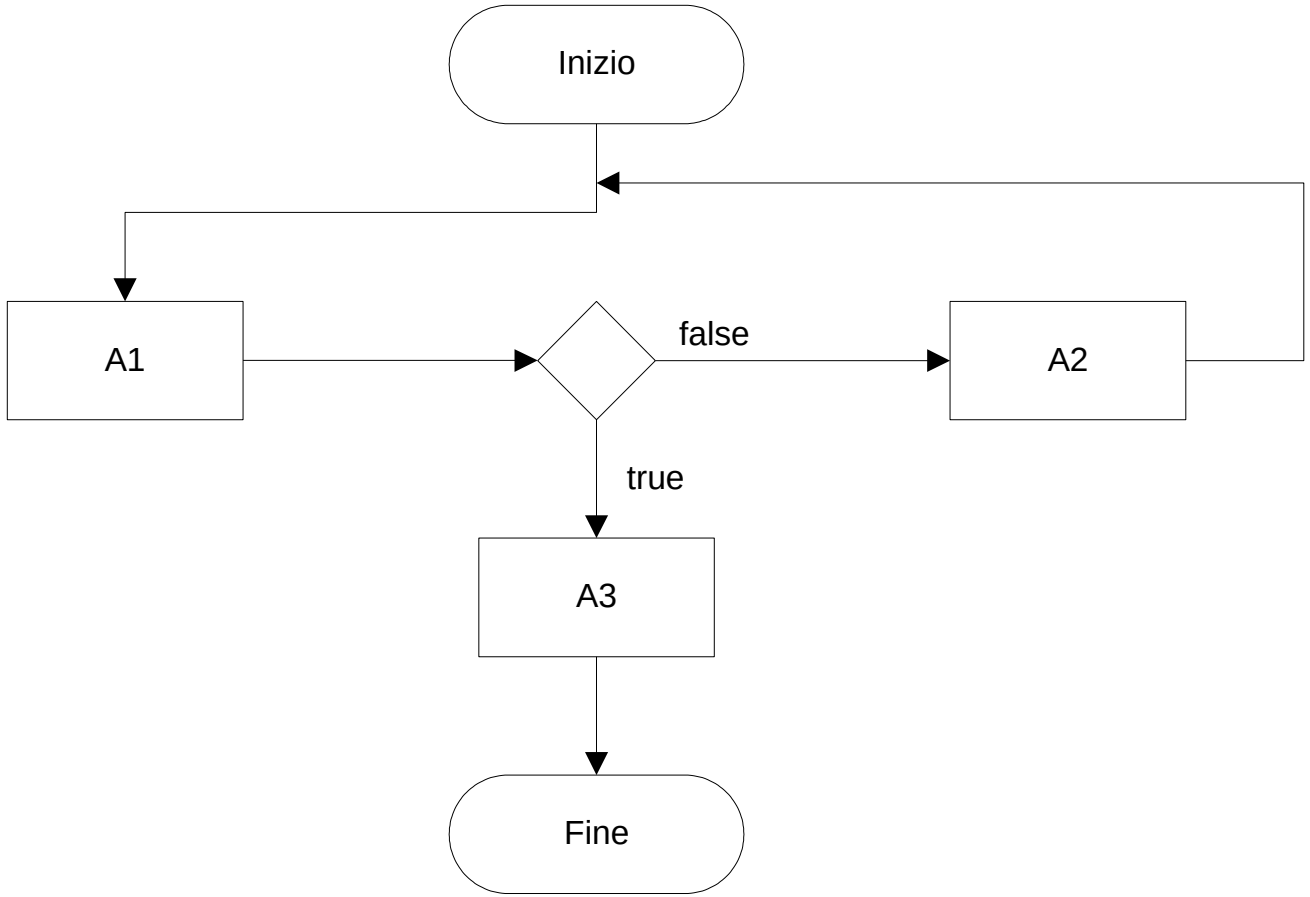


# Diagrammi di flusso: regole

- Ogni diagramma di flusso contiene almeno:
  - un **inizio**
  - uno o più **azioni**
  - eventualmente dei **blocchi decisionali**

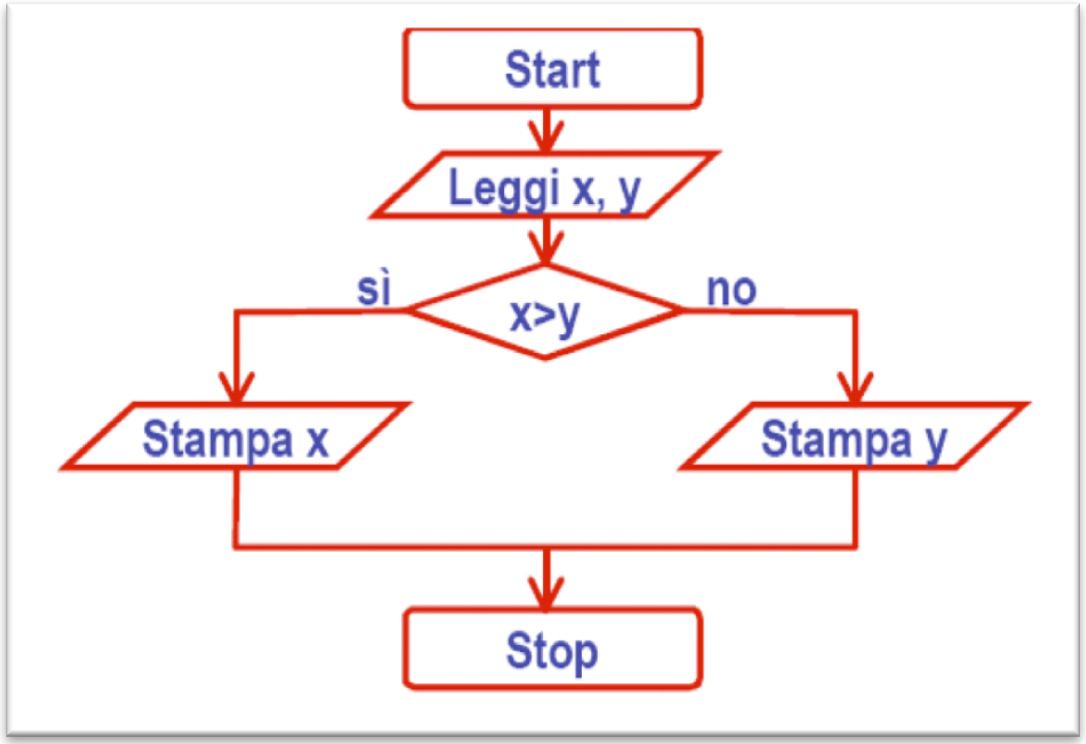
- Regole:
  - Ogni **blocco di azione** contiene una freccia entrante ed una uscente
  - Ciascun **blocco decisionale** contiene una freccia entrante e due frecce uscenti
  - Ciascuna **freccia** può entrare in un blocco, oppure si inserisce in un'altra freccia
  - Ciascun **blocco** è raggiungibile a partire dal blocco iniziale
  - Il **blocco finale** è raggiungibile a partire dal blocco iniziale

# Esempio di diagramma di flusso



# Esempio di diagramma di flusso

- Letti in input 2 numeri, stampare il massimo



# Schemi di composizione fondamentali

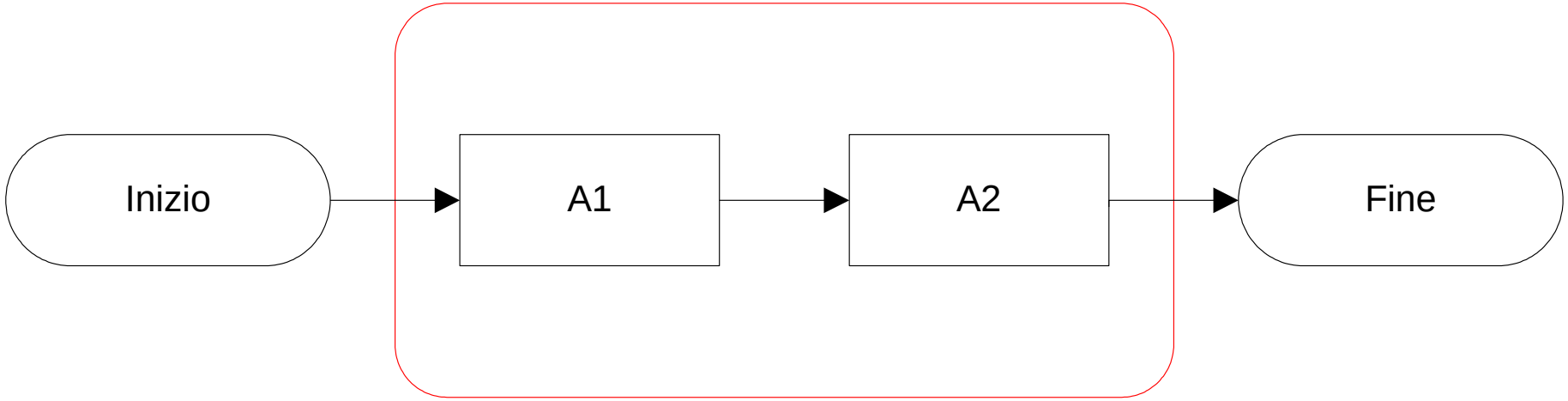
Schemi fondamentali di **composizione**

- **Sequenza**
- **Selezione**
- **Iterazione**

- A partire da questi, si definiscono i **grafi di flusso strutturati**

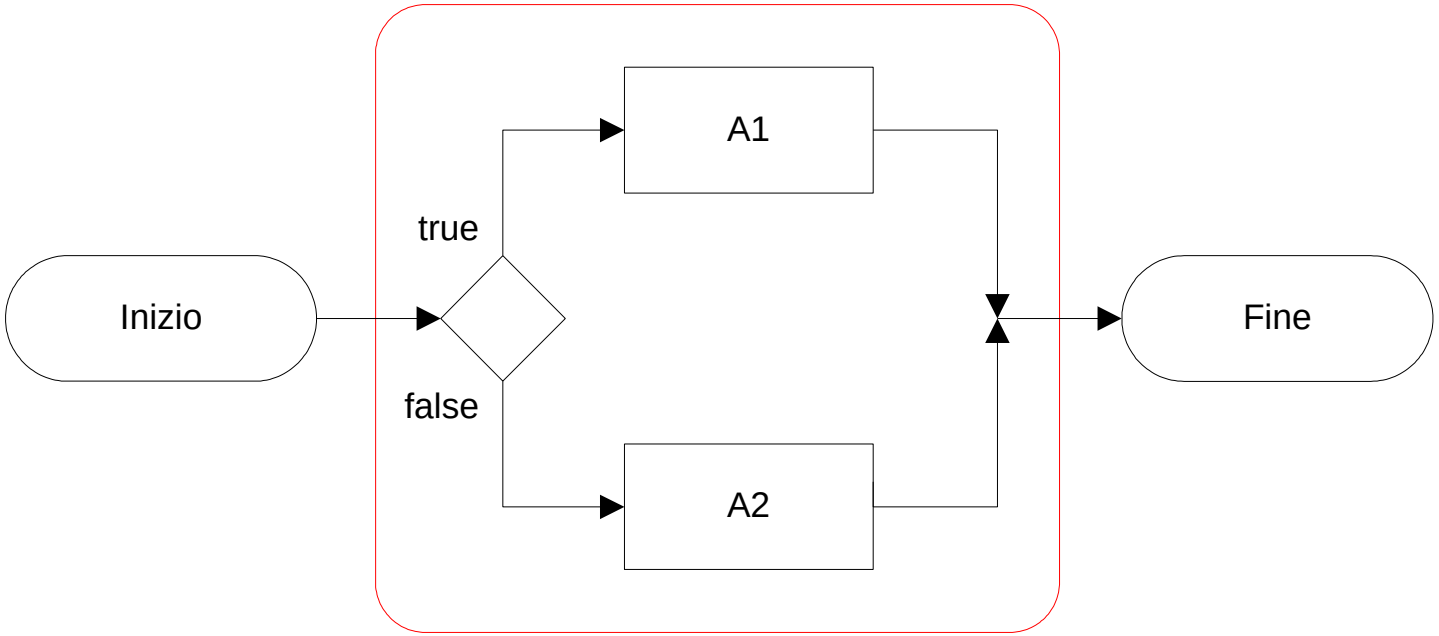
# Sequenza

- Questo schema fondamentale di composizione descrive la **sequenzialità di due azioni elementari**



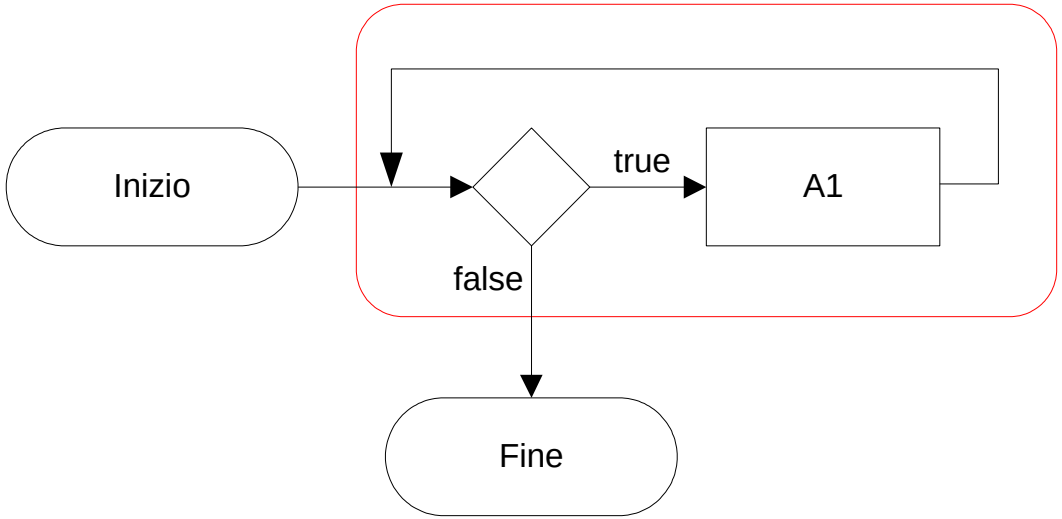
# Selezione

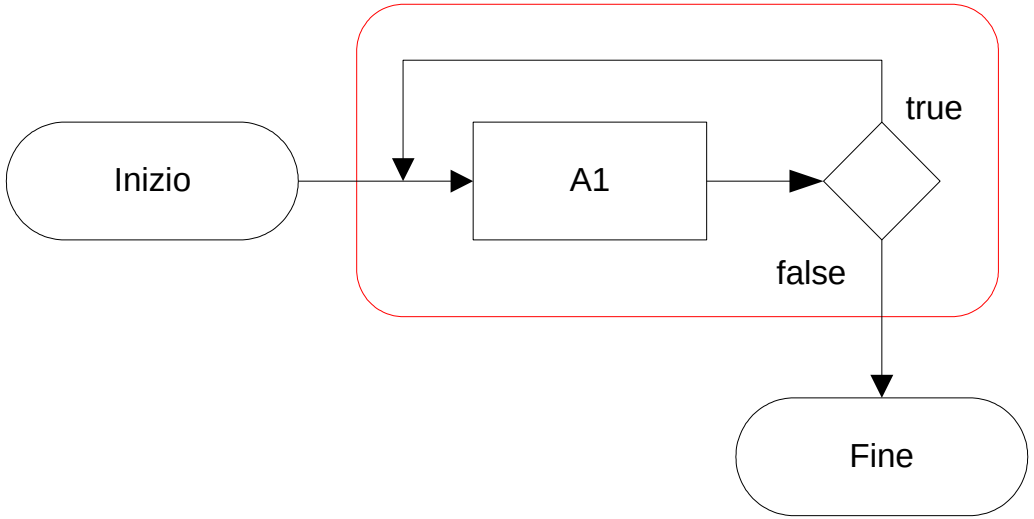
- Questo schema fondamentale di composizione descrive la **selezione tra due alternative**



# Iterazione

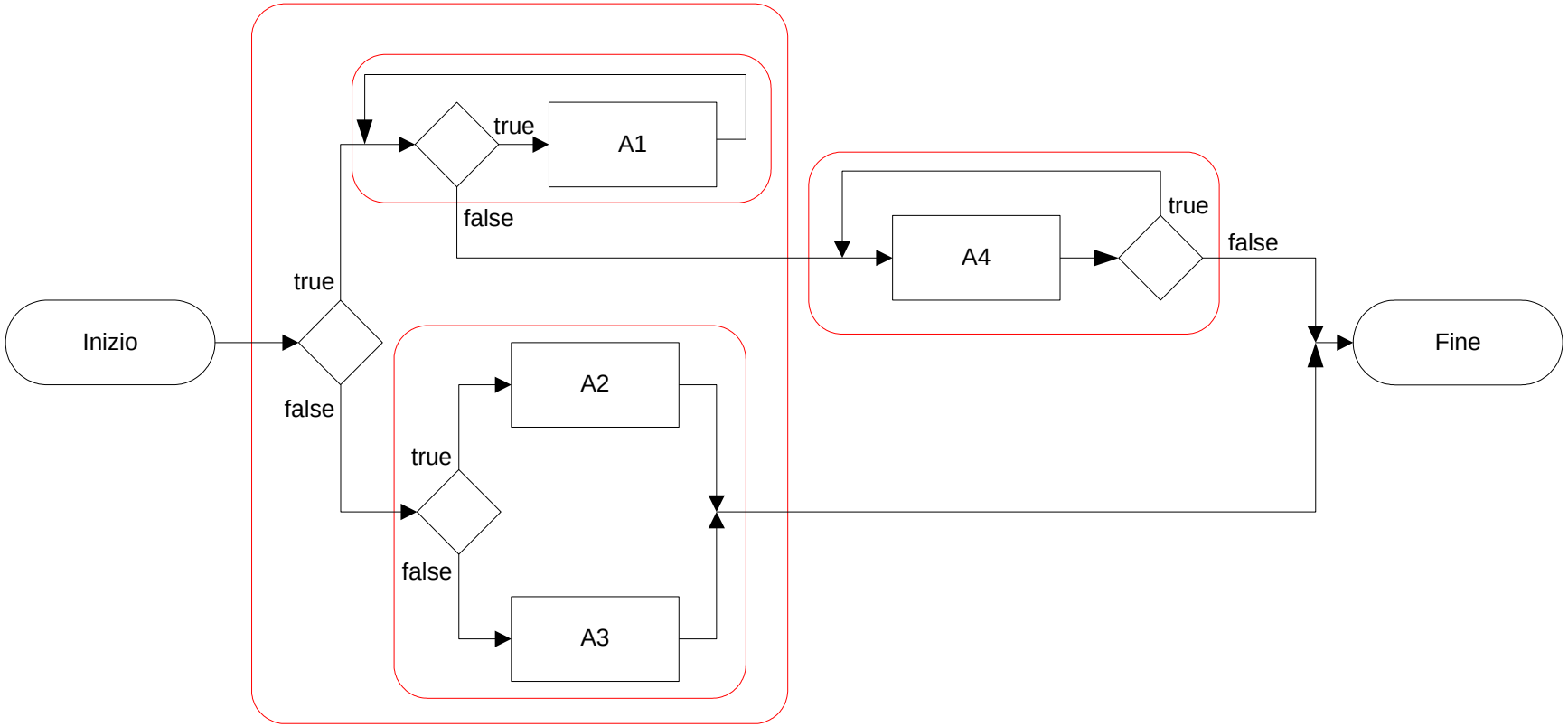
- Questo schema fondamentale di composizione descrive l'**iterazione** (la **ripetizione**) di una azione quando una condizione è verificata.
  - Esistono **due modelli** che si differenziano sul momento in cui viene valutata la condizione



# Grafi di flusso strutturati

- Descrivono un algoritmo **componendo gli schemi fondamentali**

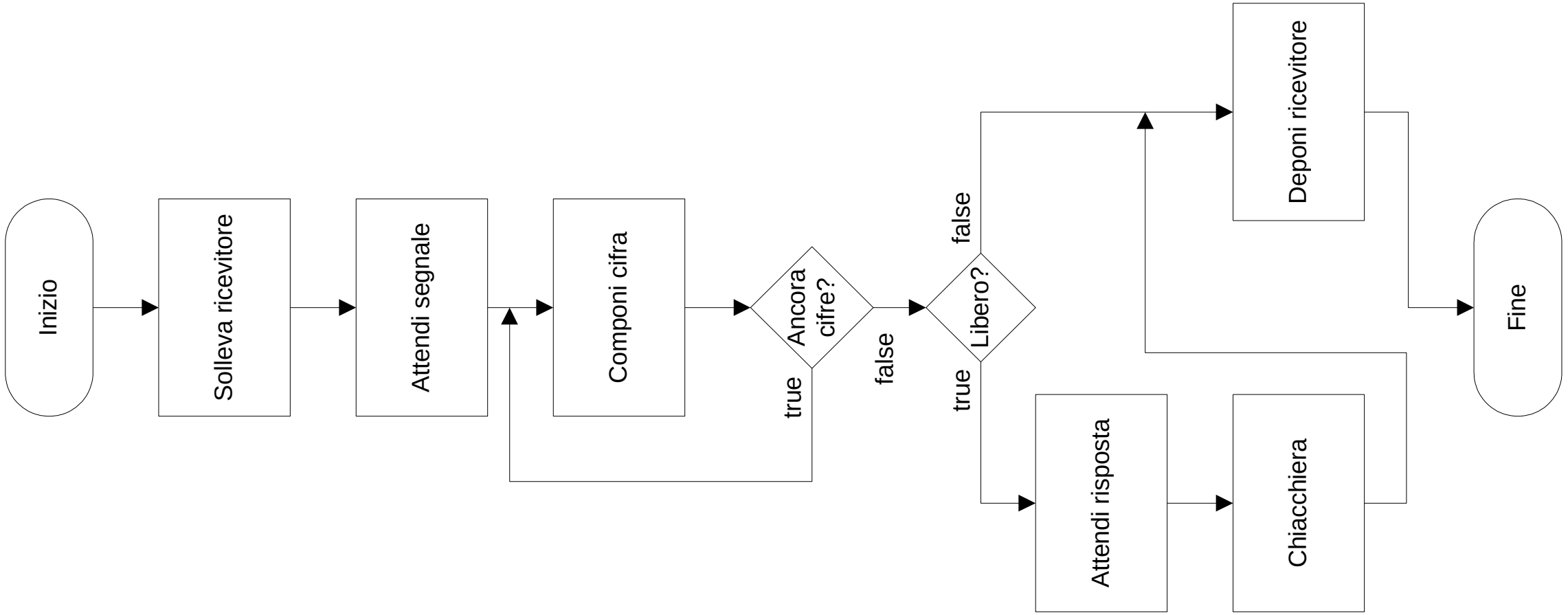


# Esempio: la telefonata

Si supponga che le azioni note siano:

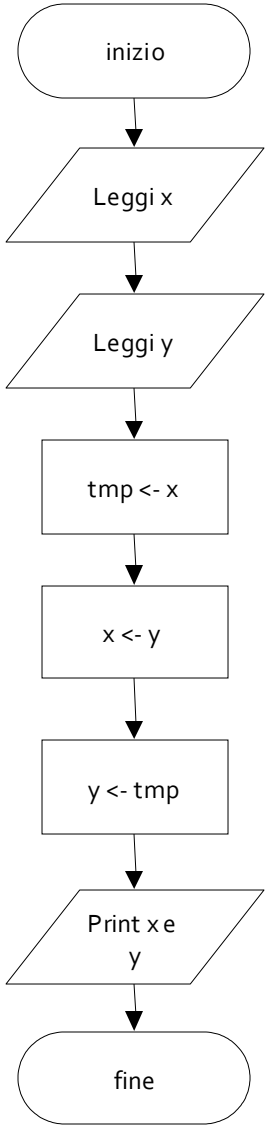
- Solleva il ricevitore
- Attendi il segnale
- Componi il numero
- Attendi risposta
- Chiacchiera
- Deponi il ricevitore

# Esempio: la telefonata → Diagramma di flusso



# Alcuni algoritmi «notevoli»

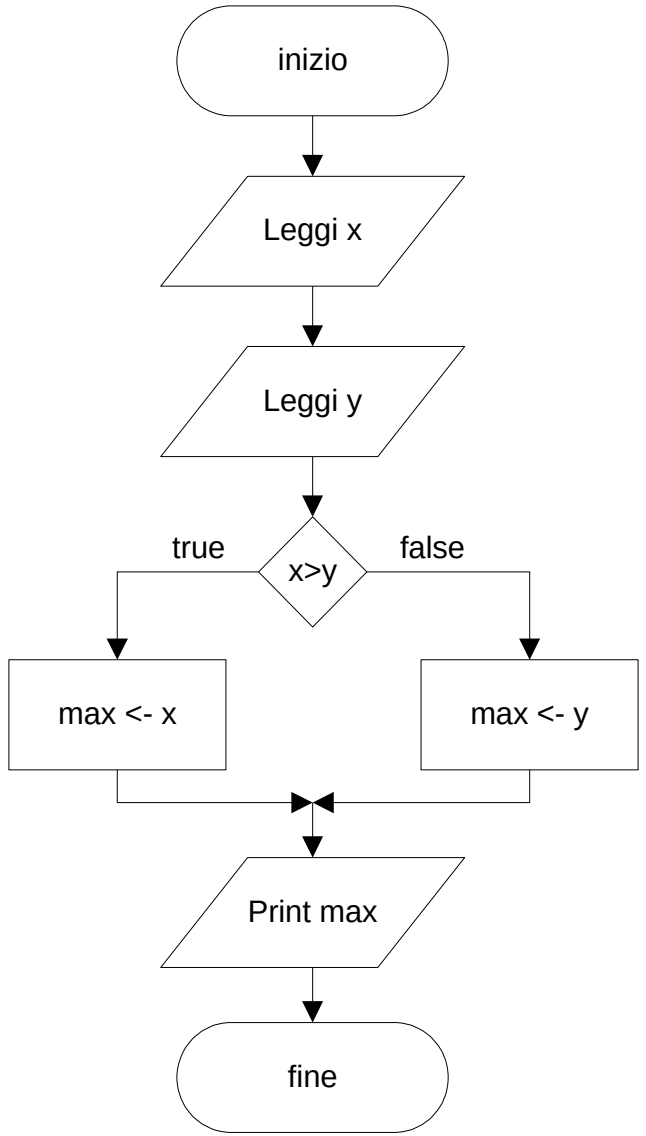
- **Algoritmo di scambio**, ovvero: come scambiare il valore di due **variabili**?
  - L'algoritmo base che introduce l'**assegmanento**
- Dati due valori 10 e 20, contenuti rispettivamente in due variabili $x$ e $y$
- Risultato: $x$ contenga 20 e $y$ contenga 10



# Alcuni algoritmi «notevoli»

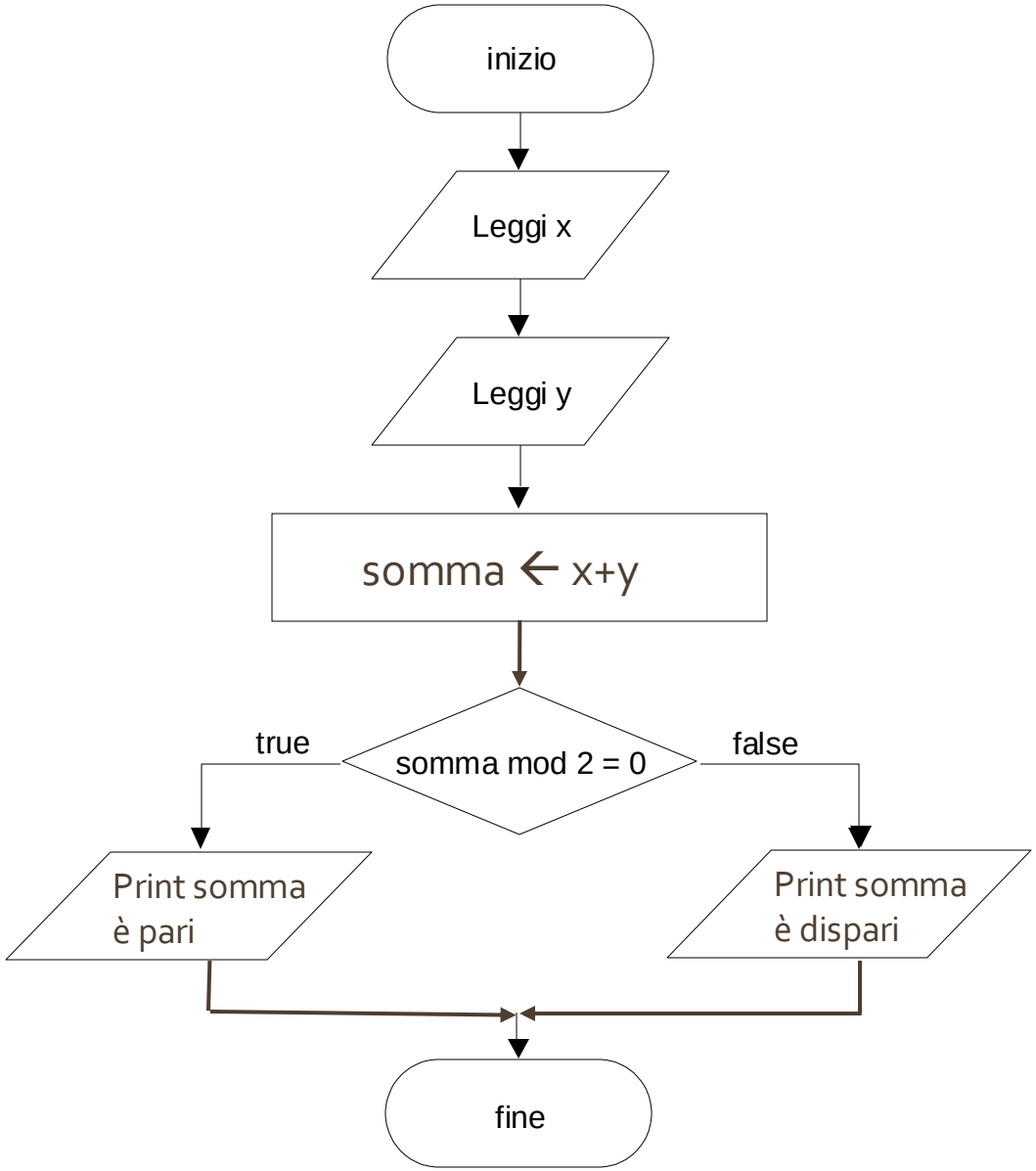
- **Massimo** fra due numeri

- Dati due numeri in ingresso, trovare il massimo tra i due e stamparlo



# Esercizio

- Letti in input 2 numeri in input, $x$ e $y$. farne la somma. Se la somma è un numero pari, stampare che il risultato è pari, altrimenti stampare che il risultato è dispari



# Progettazione di algoritmi

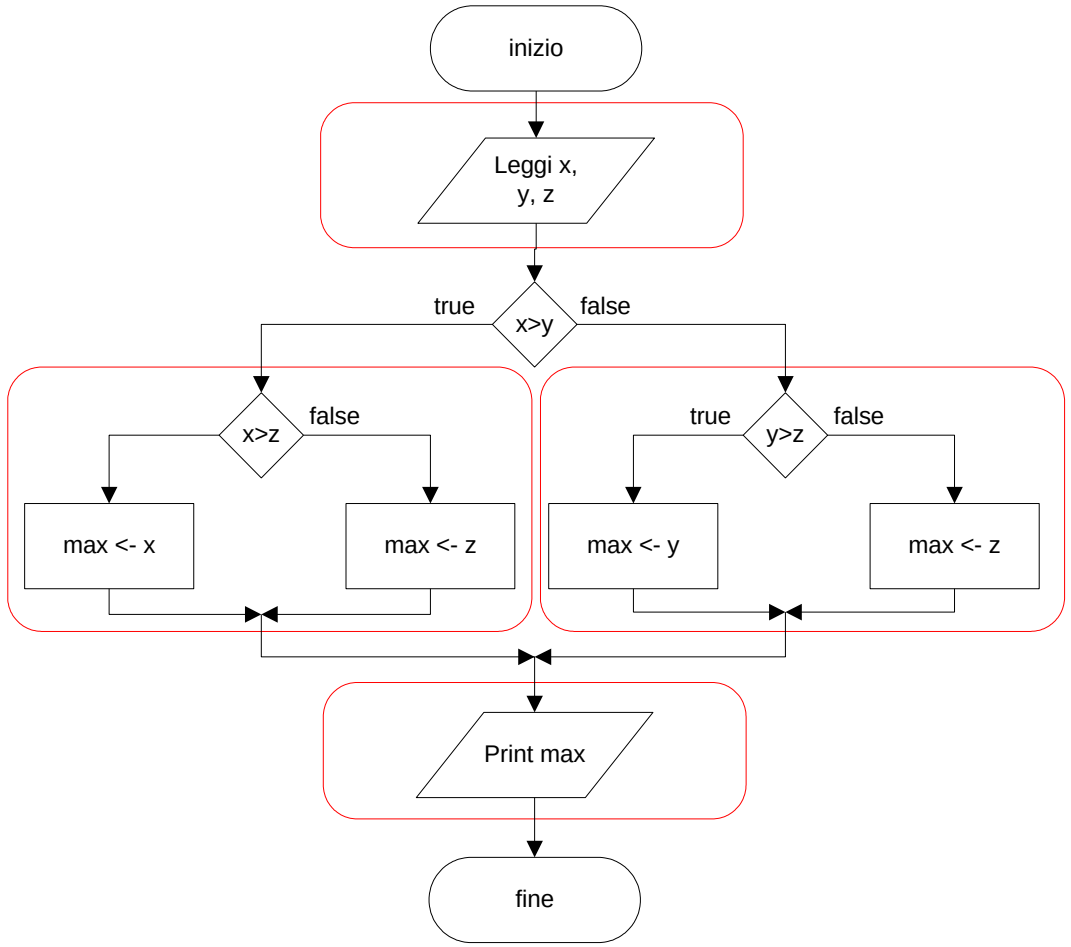
- Se il problema è **semplice**, l'algoritmo è facilmente individuabile

- Se il problema è **complesso**, si riduce la sua complessità utilizzando generalmente un approccio chiamato **TOP-DOWN**:
  - Scomporre in sotto problemi più semplici
  - Progettare **un algoritmo per ciascun sotto-problema**
  - Eventualmente si itera

# Approccio TOP-DOWN

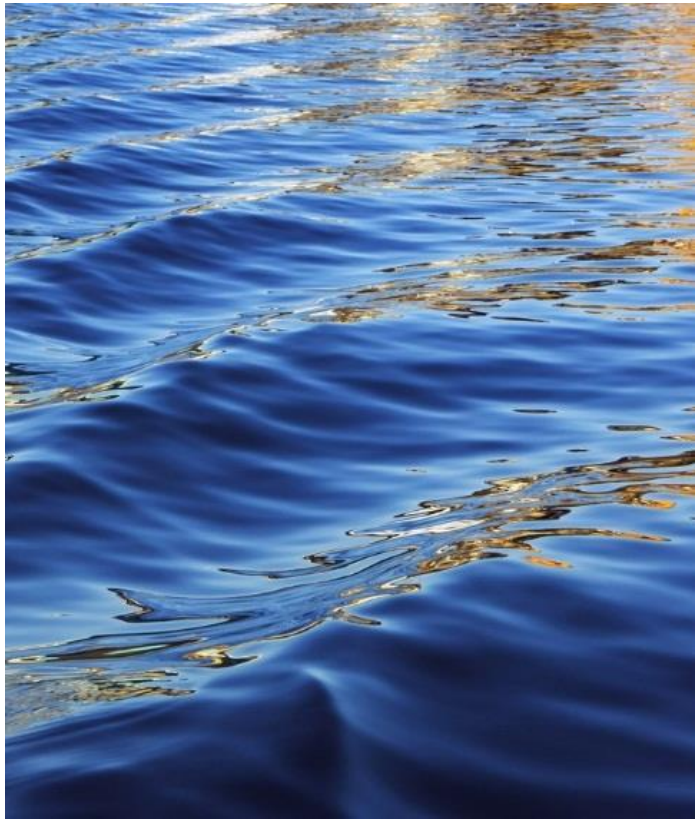
- Aumenta il numero complessivo di problemi da risolvere, ma la difficoltà di ciascuno è minore
- Quando ci si ferma?
  - Quando si giunge ad individuare sotto-problemi i cui algoritmi utilizzano azioni che sono comprensibili all'esecutore
- I sotto-problemi devono fra loro essere **indipendenti** dal punto di vista della loro progettazione ...
- ... ma fra loro **cooperano** per giungere al risultato finale del problema iniziale

# Esempio: massimo tra 3 numeri





# I linguaggi di programmazione



# Un esecutore particolare: il computer

- L'algoritmo va **codificato** mediante un **linguaggio di programmazione** le cui azioni sono espresse tramite **operazioni**
- Un linguaggio di programmazione è un **linguaggio artificiale** che viene usato per descrivere algoritmi
- La codifica dell'algoritmo mediante un linguaggio di programmazione è definita **programma**
- **Diversi linguaggi di programmazione → ma stessa logica di base**

# Esempi di linguaggi di programmazione

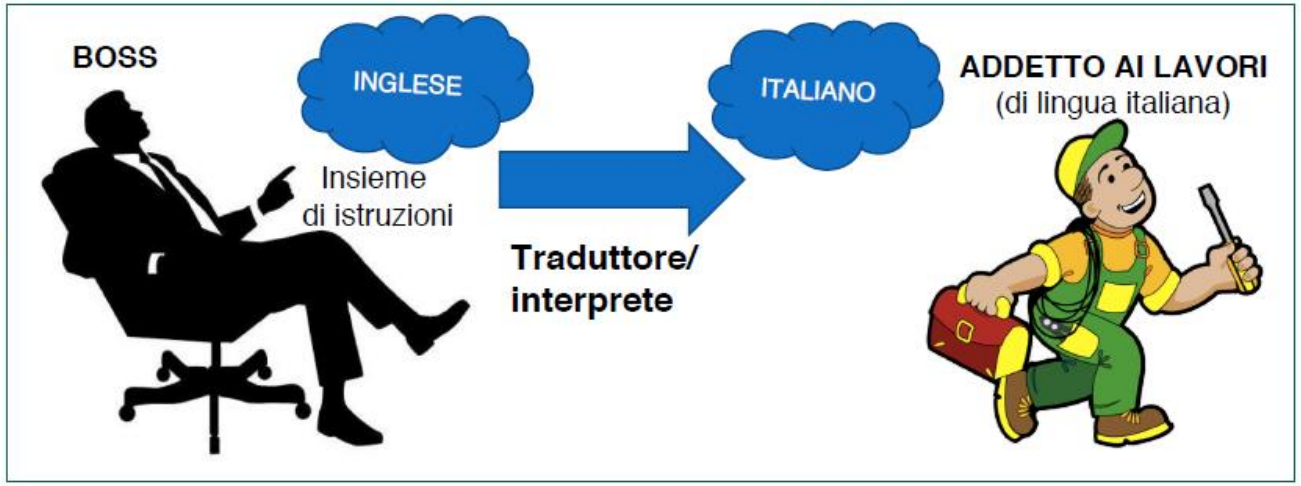
- Esempi (tra i più noti):
  - C++
  - Java
  - Python
  - MATLAB
  - Ecc.

- N.B. HTML, XML, ... **NON** sono linguaggi di programmazione. Sono **linguaggi di *markup***!

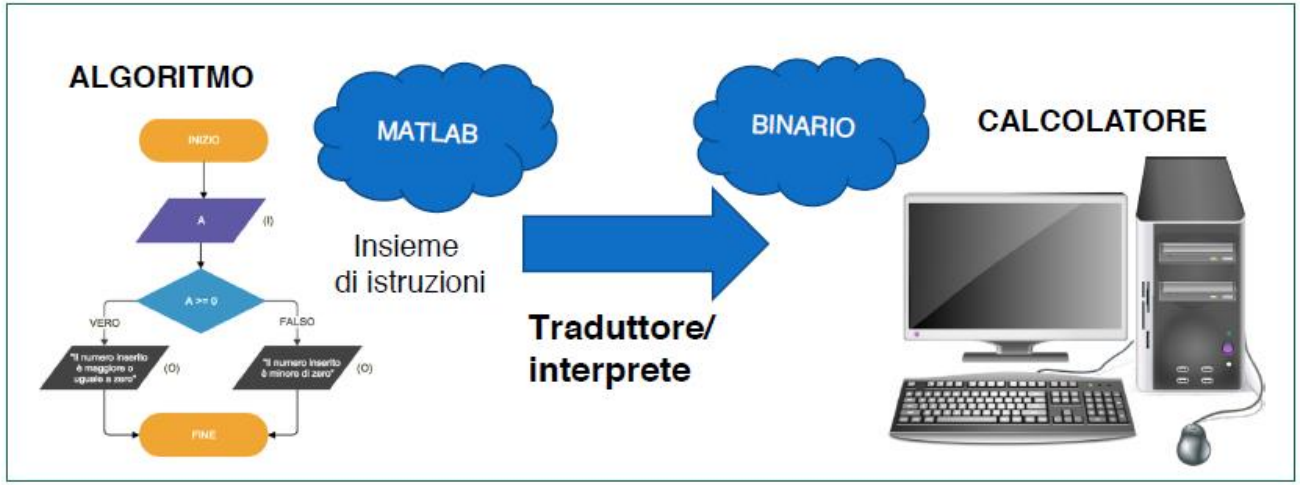
- N.B.2 SQL **NON** è un linguaggio di programmazione. È un **linguaggio di interrogazione**!

# Linguaggi di programmazione

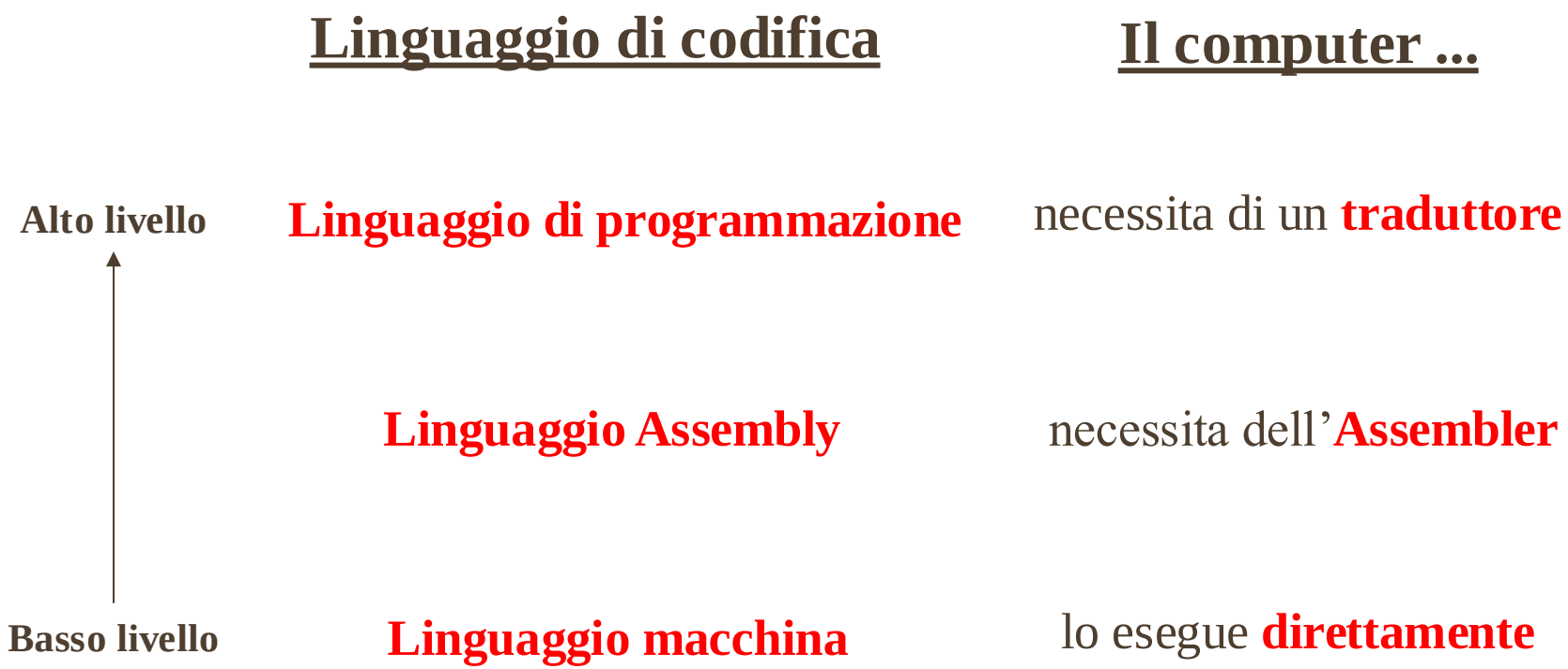
**SISTEMA NON AUTOMATIZZATO**



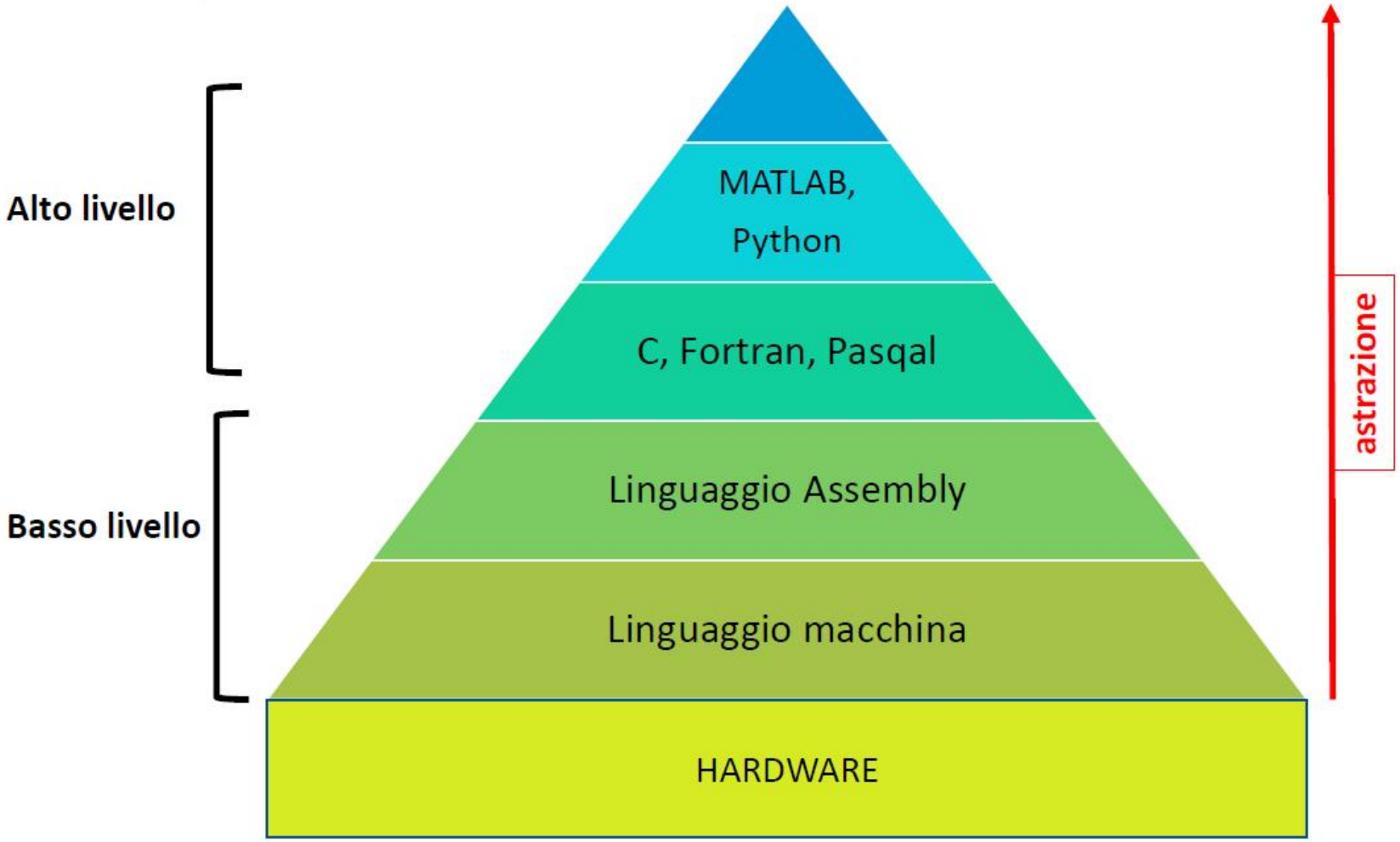

**SISTEMA AUTOMATIZZATO**



# Gerarchia dei linguaggi di programmazione



# Gerarchia dei linguaggi di programmazione



# Il linguaggio macchina (in binario!)

- Gli algoritmi sono codificati in ***word*** di 0 e 1
- Una ***word*** è un gruppo di bit di una determinata dimensione che sono gestiti come unità dal set di istruzioni o dall'hardware di un processore.
- Codifica usata fino alla prima metà degli anni ’50
- Durante la seconda metà degli anni ’50, il linguaggio con cui l’uomo comunicava con la macchina cominciò ad “alzarsi di livello”
- … la fatica di **tradurre** i programmi fu affidata al calcolatore

# Il linguaggio macchina

- NB: ogni modello di processore è in grado di comprendere **un proprio particolare linguaggio macchina**
- Processori compatibili parlano lo stesso linguaggio
- Linguaggio macchina può essere a 32 o 64 bit
- Il numero (massimo) di operandi è fissato

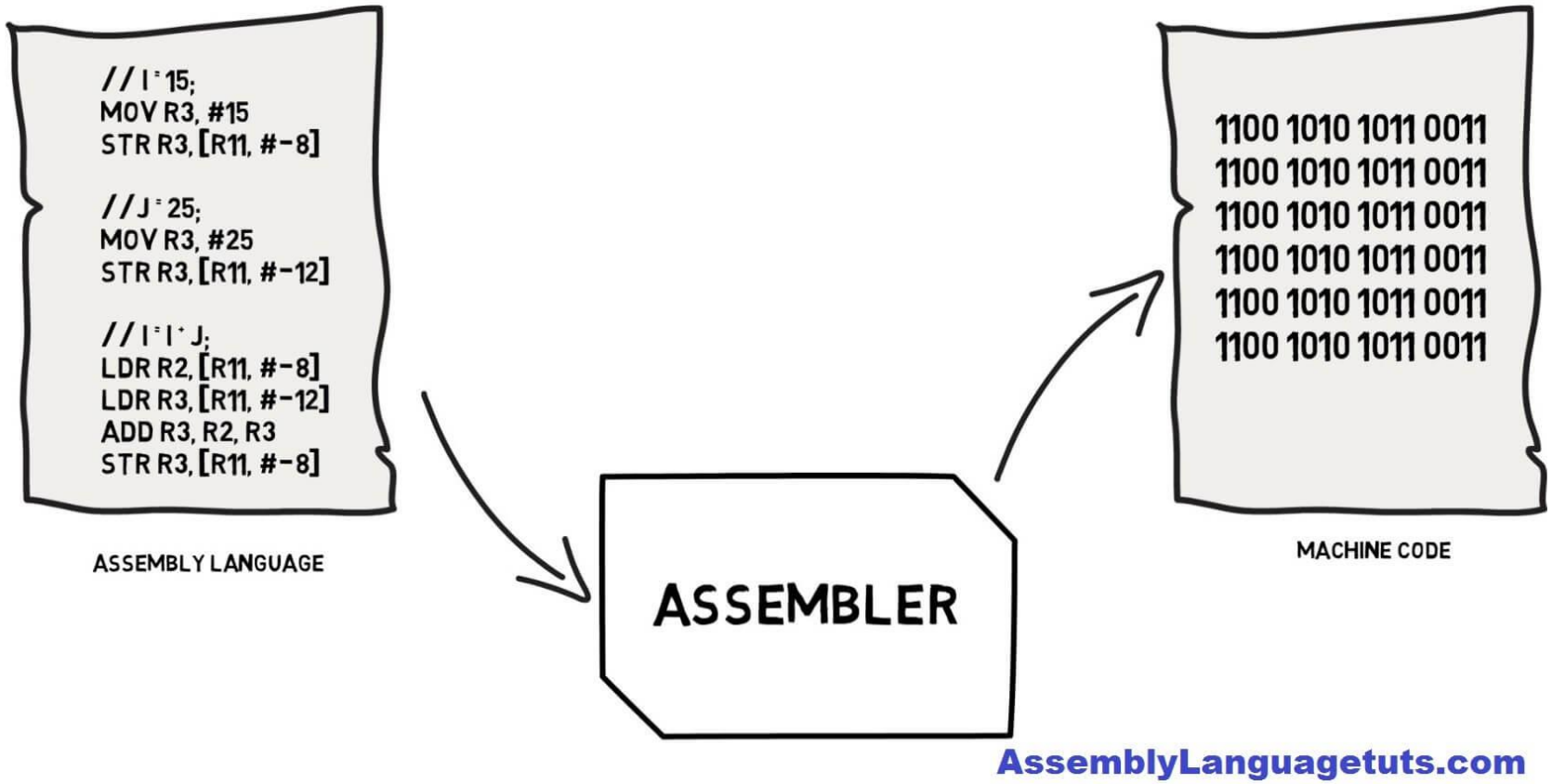
**Esempio di linguaggio macchina**

| | Comando | Param. | Traduz. Comando | Traduz. Param. |
|---|---|---|---|---|
| 0 | 0100 | 00000000 | (READ) | |
| 1 | 0010 | 00100000 | (STORE) | 32 |
| 2 | 0001 | 00000000 | (LOAD=) | 0 |
| 3 | 0010 | 00100001 | (STORE) | 33 |
| 4 | 0000 | 00100000 | (LOAD) | 32 |
| 5 | 1100 | 00001101 | (BEQ) | 13 |
| 6 | 0100 | 00000000 | (READ) | |
| 7 | 0110 | 00100001 | (ADD) | 33 |
| 8 | 0010 | 00100001 | (STORE) | 33 |
| 9 | 0000 | 00100000 | (LOAD) | 32 |
| 10 | 0011 | 00000001 | (SUB=) | 1 |
| 11 | 0010 | 00100000 | (STORE) | 32 |
| 12 | 1000 | 00000100 | (BR) | 4 |
| 13 | 0000 | 00100001 | (LOAD) | 33 |
| 14 | 0101 | 00000000 | (WRITE) | |
| 15 | 1111 | 00000000 | (END) | |

# Il linguaggio Assembly

- Definisce un **set di istruzioni elementari**
- Si avvicina molto al linguaggio macchina
- Corrispondenza **quasi biunivoca** fra istruzione e numero di operazioni eseguite dal sistema

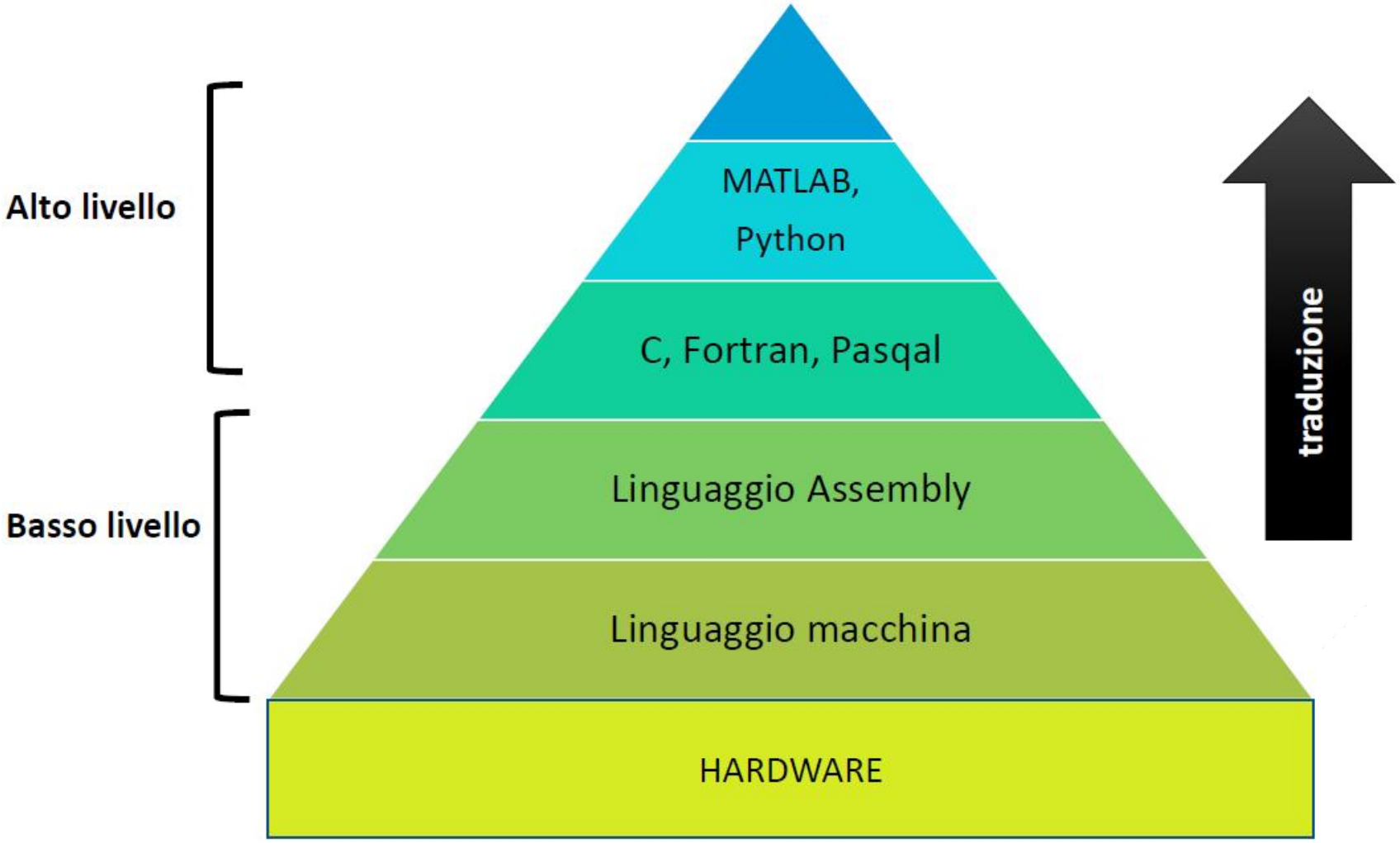
# Il linguaggio Assembly



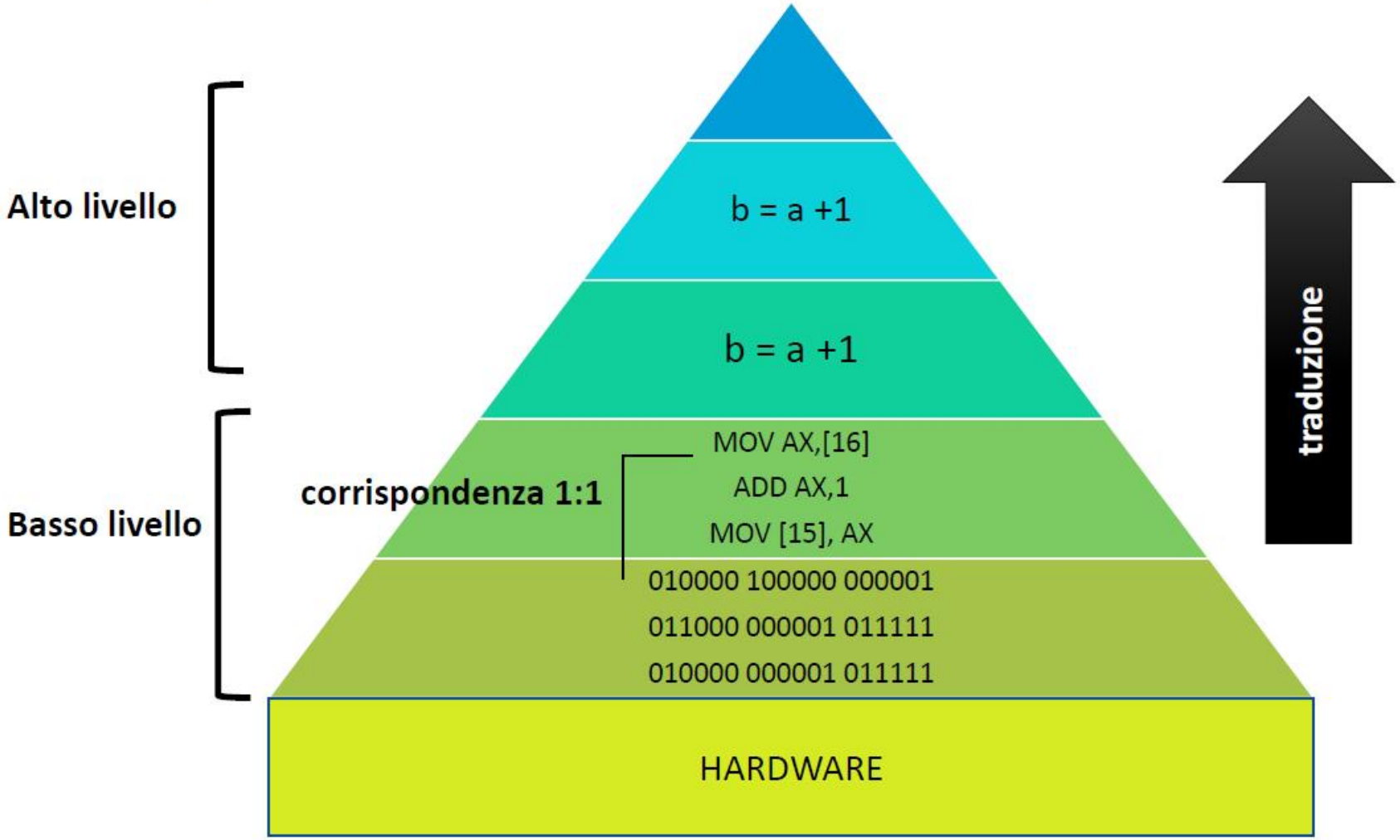
# Linguaggi ad alto livello

- Linguaggio molto vicino al **linguaggio naturale**
- Le **istruzioni** sono molto **astratte**: ognuna di esse può corrispondere ad un elevato numero di operazioni da parte della macchina
  - … facilitano la vita al programmatore
- Esempi:
  - Java
  - C
  - C++
  - MATLAB
  - …

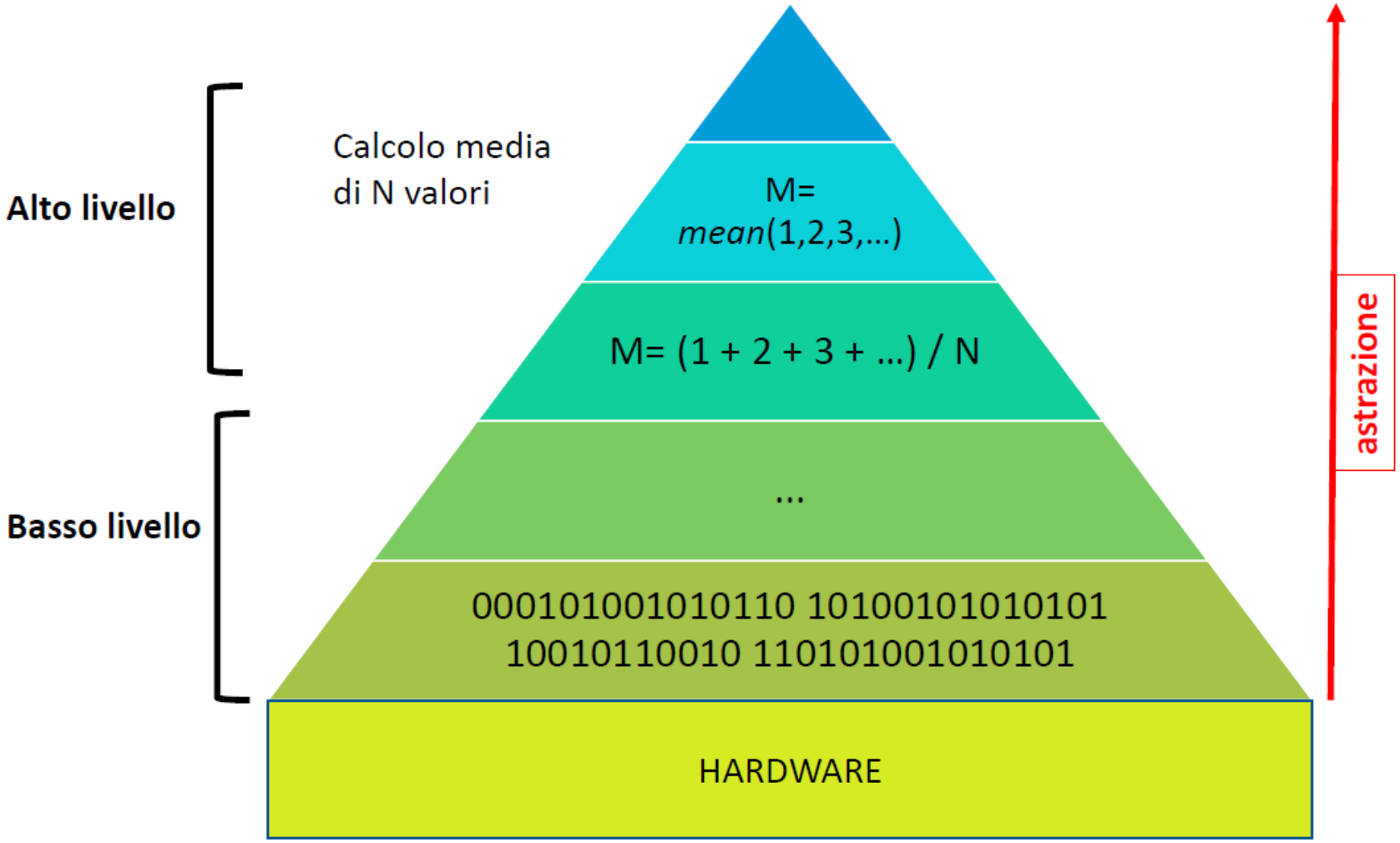
# Traduzione


Alto livello
Basso livello
MATLAB,
Python
C, Fortran, Pasqal
Linguaggio Assembly
Linguaggio macchina
HARDWARE
traduzione

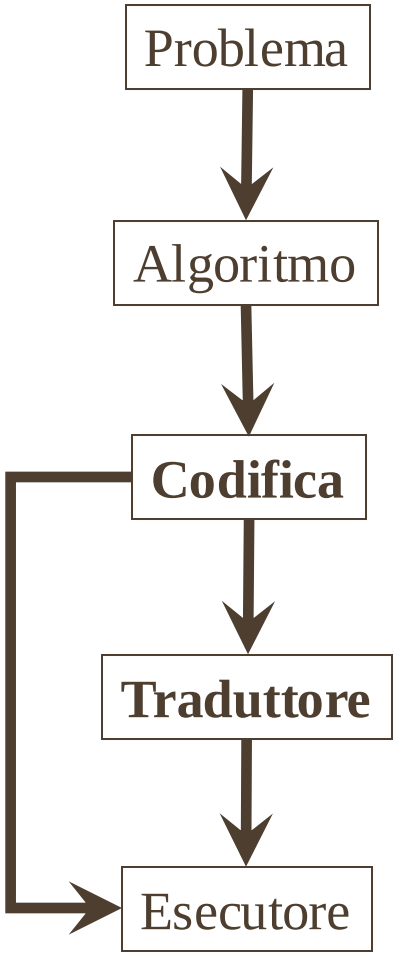
# Traduzione


Alto livello
Basso livello
b = a +1
b = a +1
MOV AX,[16]
ADD AX,1
MOV [15], AX
corrispondenza 1:1
010000 100000 000001
011000 000001 011111
010000 000001 011111
HARDWARE
traduzione

# Traduzione


Alto livello
Basso livello
Calcolo media
di N valori
M=
mean(1,2,3,...)
M= (1 + 2 + 3 + ...) / N
...
000101001010110 10100101010101
10010110010 11010100101010101
HARDWARE
astrazione

# Le tappe




*Formulazione del problema*

*Costruzione del metodo di soluzione*

*Codifica dell'algoritmo in un linguaggio di programmazione*

*Traduzione del programma in codice comprensibile*

*Computer*

# Il traduttore

- Quando si utilizza un linguaggio diverso da quello macchina occorre un **traduttore**
- Il traduttore traduce in istruzioni macchina i **codici sorgenti** dei programmi
- Le **istruzioni macchina** corrispondono ad **operazioni elementari** che interagiscono direttamente con gli elementi del computer (**registri** e **memoria**)
- Occorre il **traduttore specifico del linguaggio di programmazione usato** e del tipo di processore montato sul computer
  - Anche il linguaggio Assembly necessita del traduttore Assembler

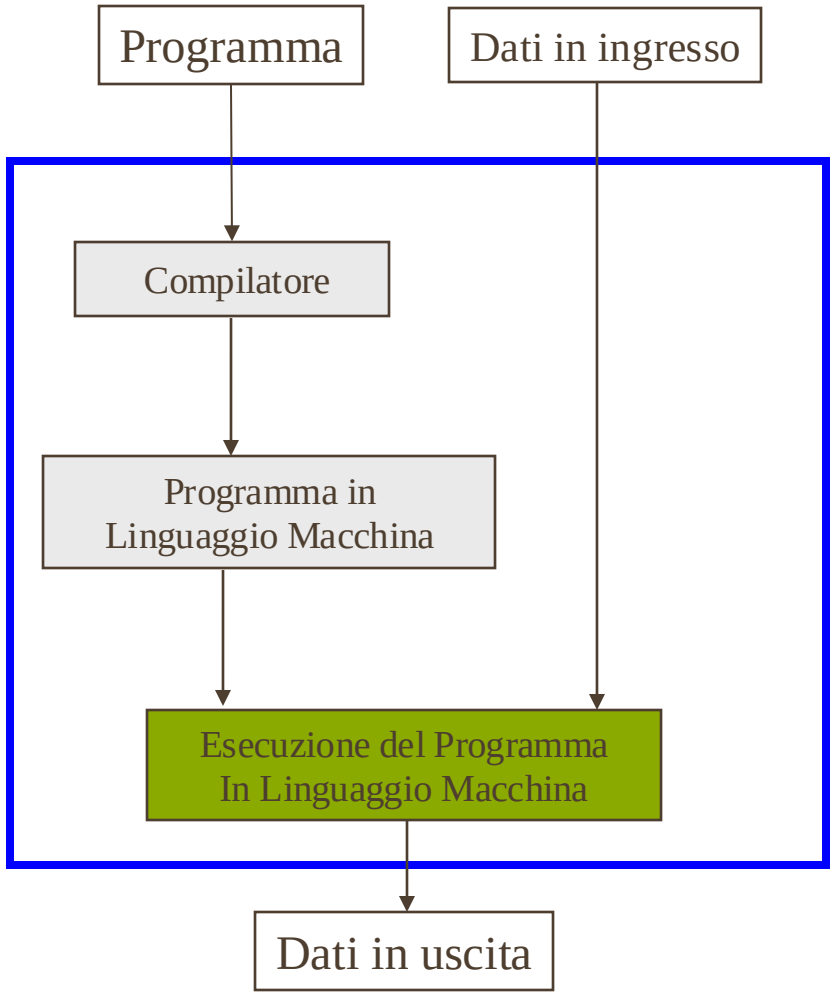
# Tipi di traduttore

- Esistono **due tipi di traduttori**:
  - **Compilatori**
    - Traducono un programma scritto in un linguaggio ad alto livello in linguaggio macchina
  - **Interpreti**
    - Traducono un programma scritto in un linguaggio ad alto livello in linguaggio macchina e simultaneamente lo eseguono

- I linguaggi si possono perciò classificare in **interpretati** o **compilati**

# Traduzione tramite compilatore

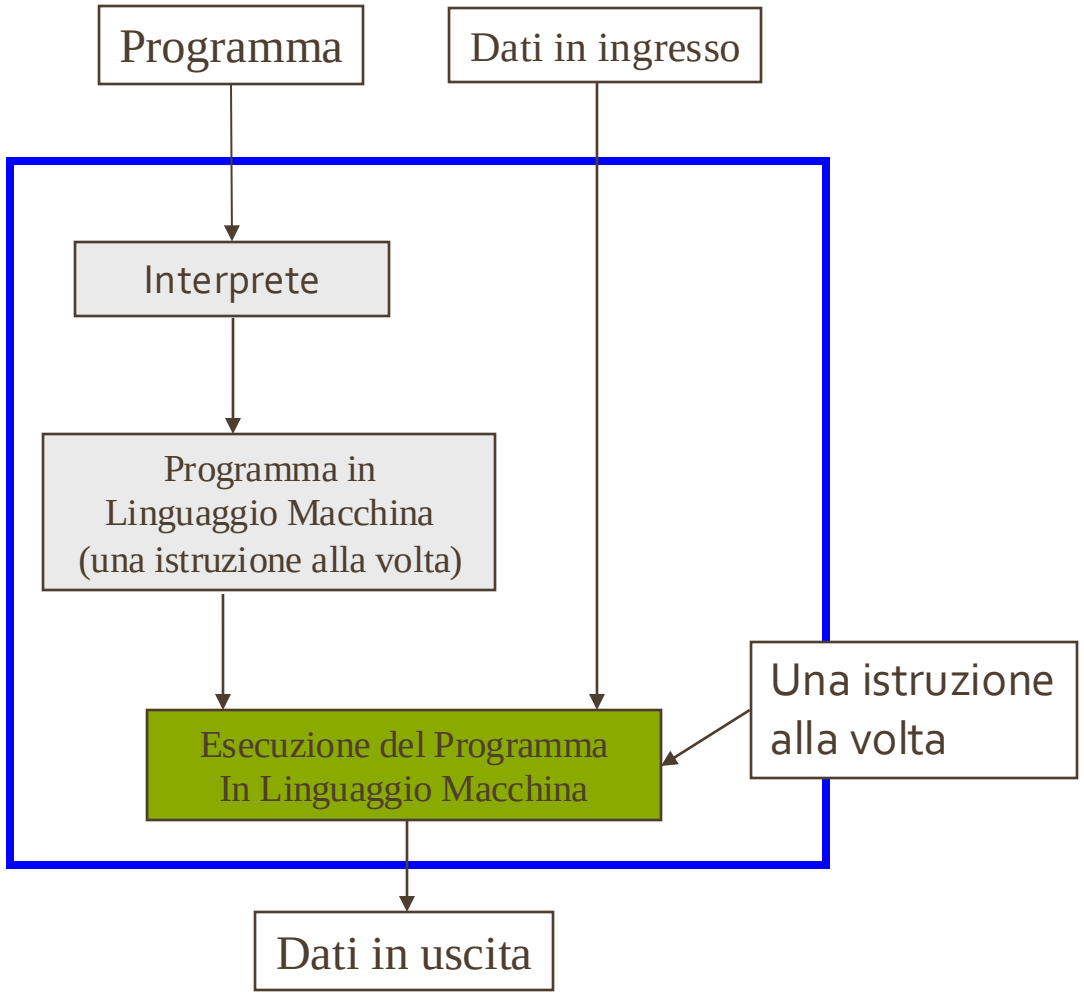
- Il **compilatore** traduce un programma scritto in linguaggio ad alto livello in un programma in codice macchina **per uno specifico hardware**



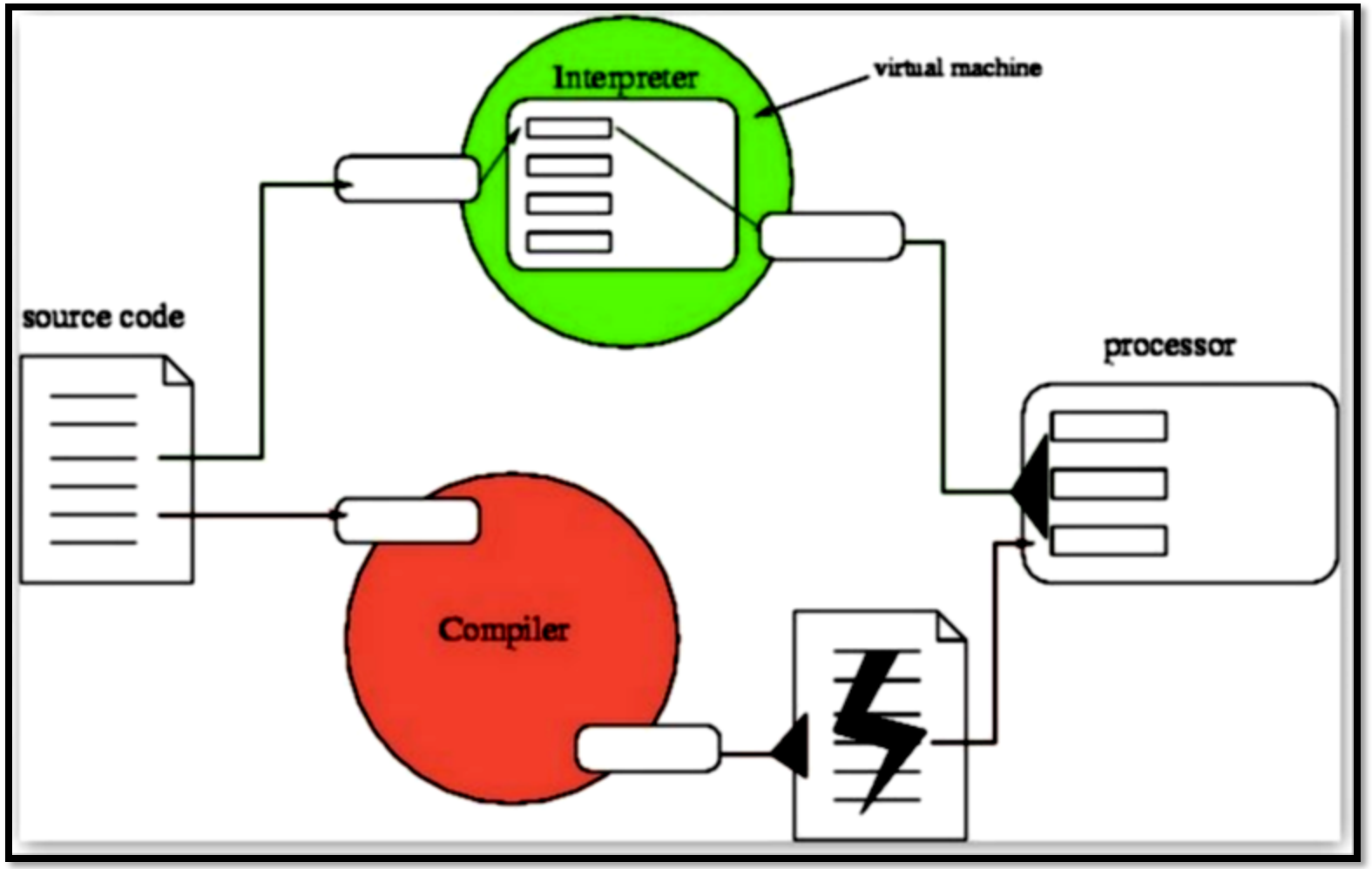
# Traduzione ed esecuzione tramite interprete

- L'**interprete** traduce un programma in linguaggio ad alto livello in codice macchina per uno specifico hardware

- Viene tradotta **un'istruzione alla volta che viene subito eseguita**



# Linguaggi interpretati e linguaggi compilati



# Linguaggi compilati

**Compilatore**

1. verifica sintattica dell'**intero codice**
2. compilazione del codice (traduzione in linguaggio macchina)
3. collegamento delle librerie esterne
4. esecuzione del programma

A differenza di un interprete, un compilatore **non esegue** il programma che riceve in ingresso, ma lo traduce in linguaggio macchina (memorizzando su file il codice oggetto pronto per l'esecuzione diretta da parte del processore)

# Linguaggi interpretati

**Interprete**

1. verifica sintattica **istruzione corrente**
2. traduzione istruzione corrente
3. esecuzione istruzione corrente
4. passaggio all'istruzione successiva

**Semplificando, «opera» in real-time**

# MATLAB?

- MATLAB è un linguaggio **interpretato** e non **compilato**: questo significa che le istruzioni vengono tradotte in linguaggio macchina (il linguaggio “capito” dal processore) e subito eseguite una per volta.