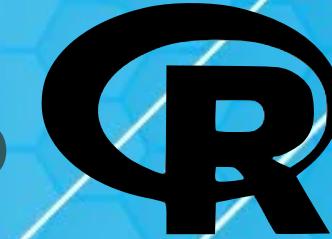




Informatica

Principi di programmazione ed analisi dati in linguaggio
Scienze Biologiche

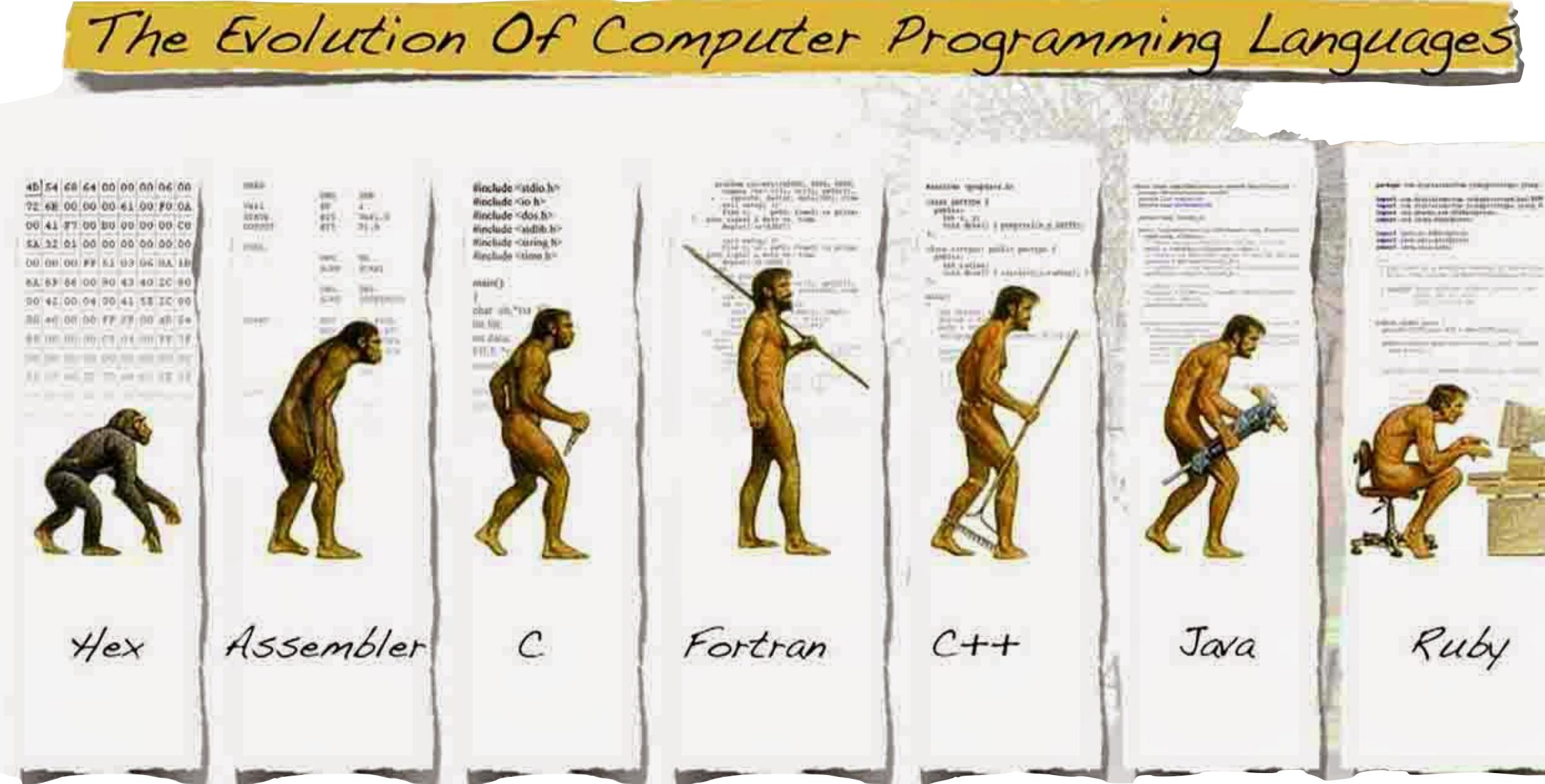


Dr. Bruno Bellisario, PhD



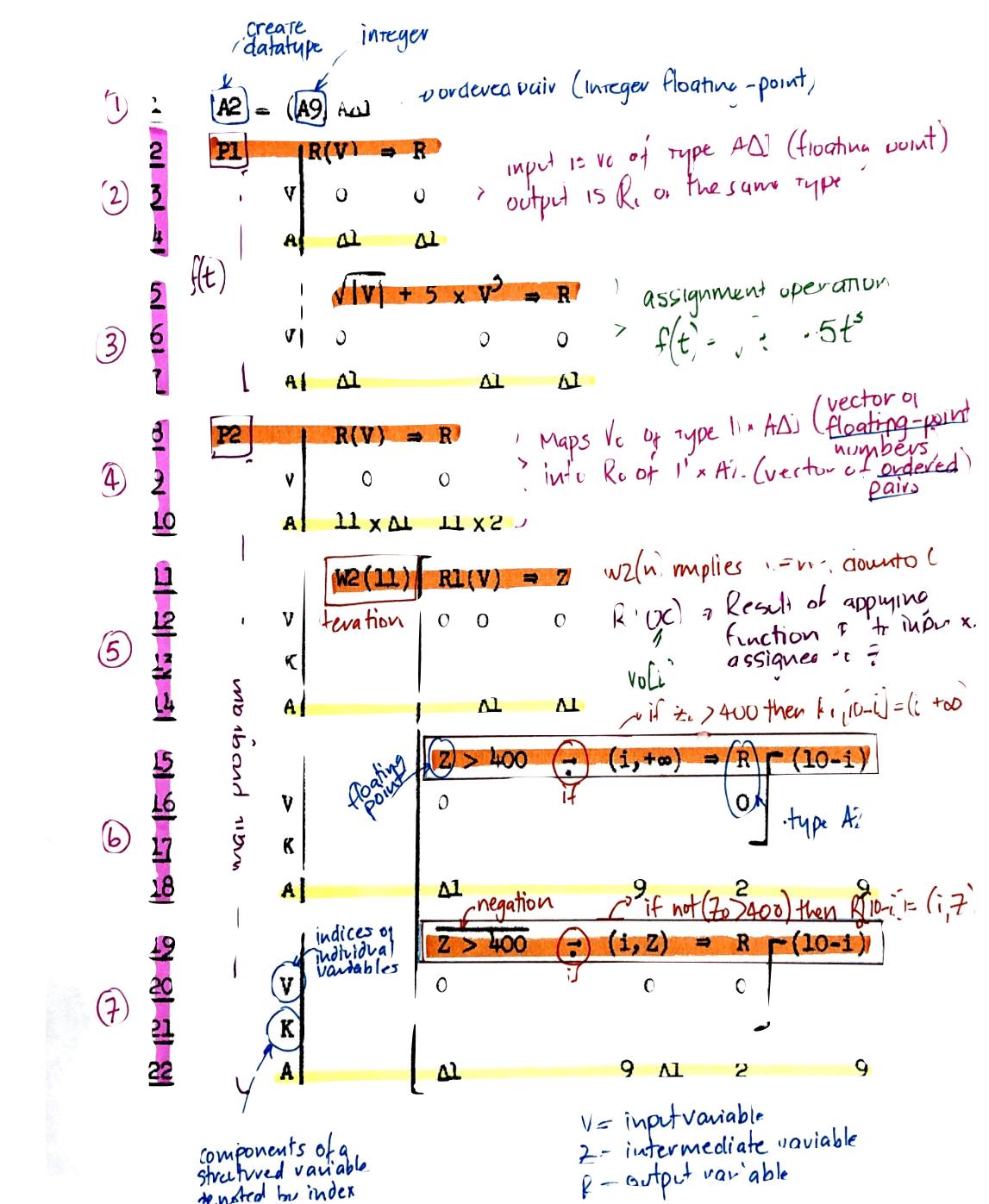
Lezione II: Introduzione ai linguaggi di programmazione

Un linguaggio di programmazione, in informatica, è un linguaggio formale che specifica un insieme di istruzioni che possono essere usate per produrre dati in uscita: esso è utilizzabile per il controllo del comportamento di una macchina formale o di un'implementazione di essa (tipicamente, un computer) ovvero in fase di programmazione di questa attraverso la scrittura del codice sorgente di un programma ad opera di un programmatore



Il primo linguaggio di programmazione della storia è il linguaggio meccanico adoperato da **Ada Lovelace** per la programmazione della macchina di Charles Babbage, al quale fu seguito il **Plankalkül** di Konrad Zuse, sviluppato da lui nella Svizzera neutrale durante la seconda guerra mondiale e pubblicato nel 1946.

La programmazione dei primi elaboratori veniva fatta invece in **short code**, da cui poi si è evoluto **l'assembly**, che costituisce una rappresentazione simbolica del linguaggio macchina. La sola forma di controllo di flusso è l'istruzione di salto condizionato, che porta a scrivere programmi molto difficili da seguire logicamente per via dei continui salti da un punto all'altro del codice.



```

MONITOR FOR 6802 1.4      9-14-80 TSC ASSEMBLER PAGE 2

C000          ORG    ROM+$0000 BEGIN MONITOR
C000 8E 00 70  START  LDS  #STACK

***** FUNCTION: INITA - Initialize ACIA *****
* INPUT: none
* OUTPUT: none
* CALLS: none
* DESTROYS: acc A

0013        RESETA EQU    %00010011
0011        CTLREG EQU    %00010001
C003 86 13   INITA  LDA A #RESETA  RESET ACIA
C005 B7 80 04
C008 86 11
C00A B7 80 04
C00D 7E C0 F1  JMP   SIGNON  GO TO START OF MONITOR

***** FUNCTION: INCH - Input character *****
* INPUT: none
* OUTPUT: char in acc A
* CALLS: none
* DESTROYS: acc A
* DESCRIPTION: Gets 1 character from terminal

C010 B6 80 04  INCH   LDA A ACIA   GET STATUS
C013 47       ASR A   RDRF    SHIFT RDRF FLAG INTO CARRY
C014 24 FA     BCC    INCH    RECEIVE NOT READY
C016 B6 80 05   LDA A ACIA+1  GET CHAR
C019 84 7F     AND A #$7F  MASK PARITY
C01B 7E C0 79   JMP   OUTCH  ECHO & RTS

***** FUNCTION: INHEX - INPUT HEX DIGIT *****
* INPUT: none
* OUTPUT: Digit in acc A
* CALLS: INCH
* DESTROYS: acc A
* Returns to monitor if not HEX input

C01E 8D F0   INCH   BSR    INCH   GET A CHAR
C020 81 30   CMP A  #'0  ZERO
C022 2B 11   BMI    HEXERR NOT HEX
C024 81 39   CMP A  #'9  NINE
C026 2F 0A   BLE    HEXRTS GOOD HEX
C028 81 41   CMP A  #'A  NOT HEX
C02A 2B 09   BMI    HEXERR
C02C 81 46   CMP A  #'F  FIX A-F
C02E 2E 05   BGT    HEXERR CONVERT ASCII TO DIGIT
C030 80 07   SUB A  #'7
C032 84 0F   AND A  #$0F
C034 39       RTS

C035 7E C0 AF  HEXERR JMP   CTRL   RETURN TO CONTROL LOOP

```

La maggior parte dei linguaggi di programmazione successivi cercarono di astrarsi da tale livello basilare, dando la possibilità di rappresentare strutture dati e strutture di controllo più generali e più vicine alla maniera (umana) di rappresentare i termini dei problemi per i quali ci si prefigge di scrivere programmi.

Tra i primi linguaggi ad alto livello a raggiungere una certa popolarità ci fu il **Fortran**, creato nel 1957 da John Backus, da cui derivò successivamente il **BASIC** (1964): oltre al salto condizionato, reso con l'istruzione IF, questa nuova generazione di linguaggi introduce nuove strutture di controllo di flusso come i cicli **WHILE** e **FOR** e le istruzioni **CASE** e **SWITCH**: in questo modo diminuisce molto il ricorso alle istruzioni di salto (**GOTO**), cosa che rende il codice più chiaro ed elegante, e quindi di più facile manutenzione.

```
implicit none

type MultiArray
    real(kind=8), allocatable :: elem(:)
end type

integer
real(kind=8), allocatable :: myID, totID, i, p
type(MultiArray), allocatable :: X(:)[:,]
type(MultiArray), allocatable :: Phi(:)[:,]
real(kind=8), allocatable :: Vect(:)

myID = this_image()
totID = num_images()

! Local Vector
allocate( X(2)[:] )
allocate( Phi(6)[:] )

p = 2
allocate( Phi(p)%elem(10) )

Phi(p)%elem(1:10) = dble(myID)
```

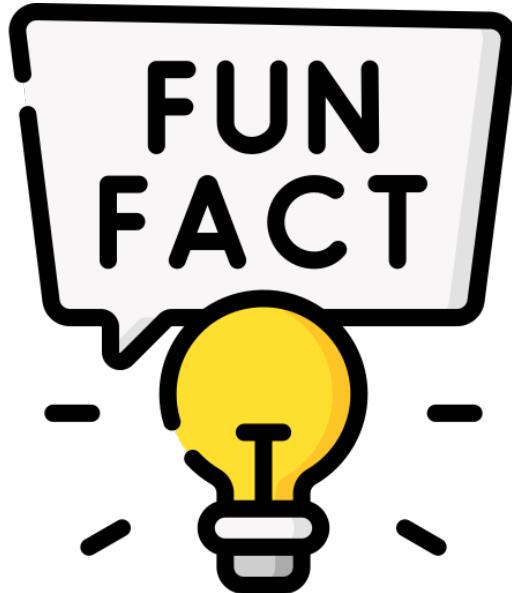
Fortran è un linguaggio di programmazione, compilato e imperativo, particolarmente adatto per il calcolo numerico e la scienza computazionale.



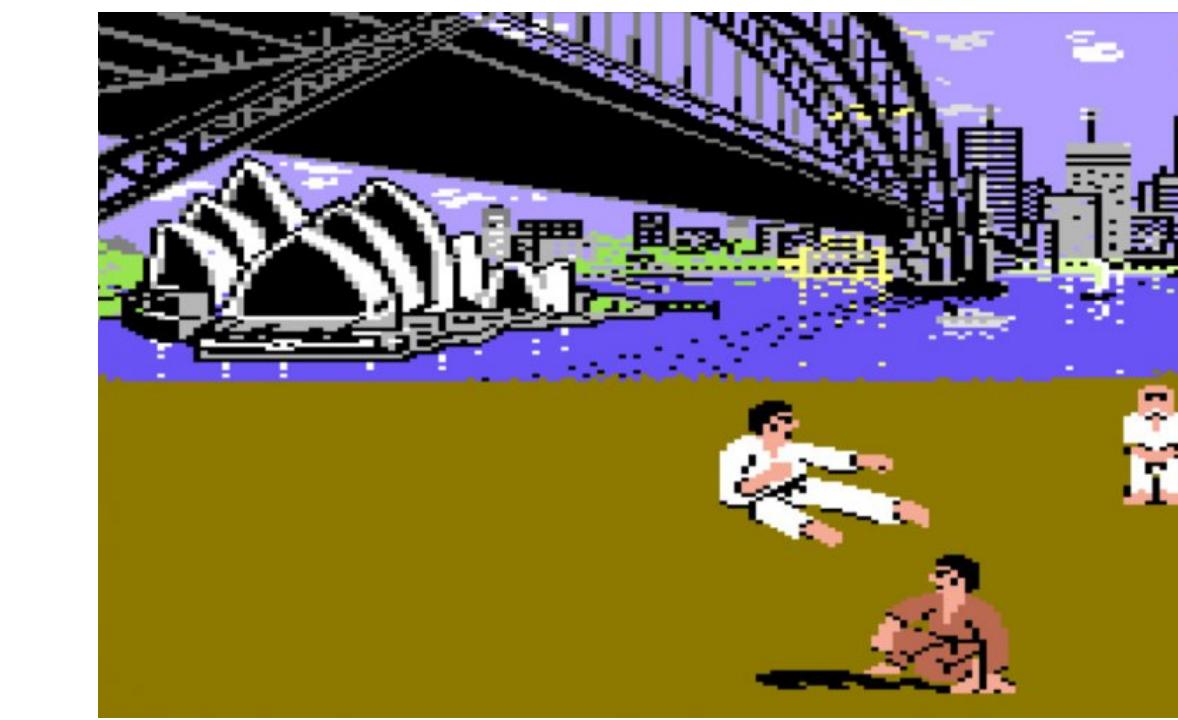
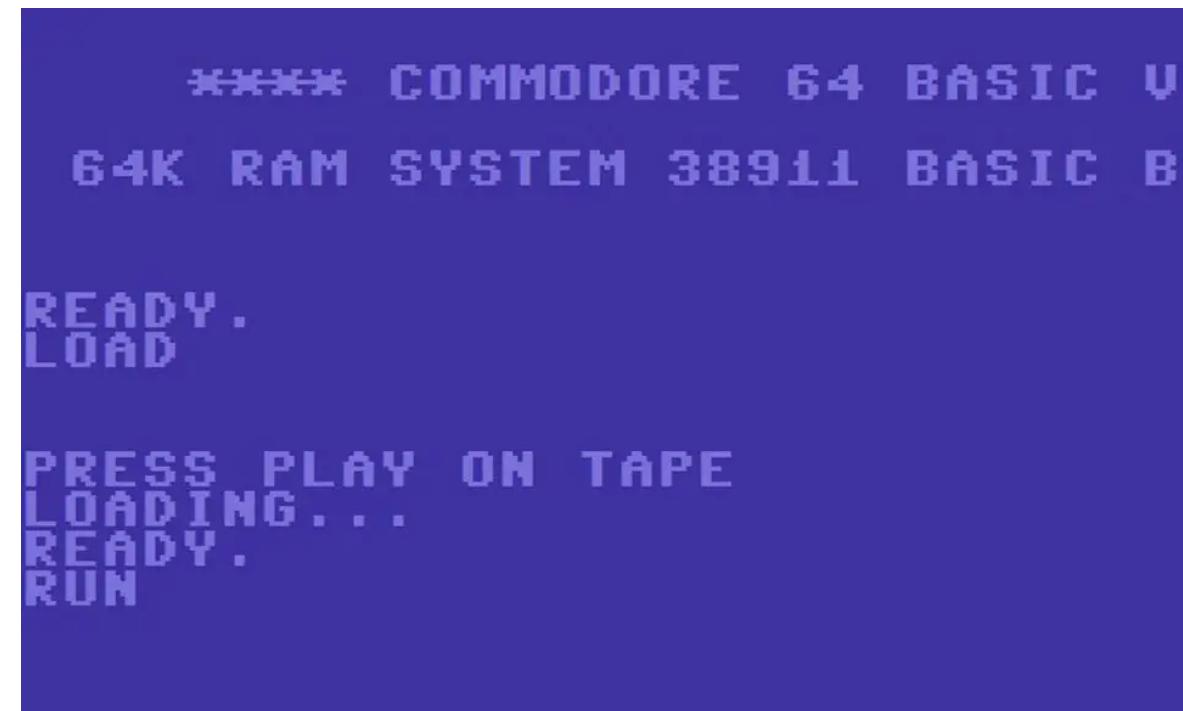
```
HELLO, WORLD!
10  HOME
20  INVERSE
30  PRINT "HELLO, WORLD!"
40  NORMAL
50  PRINT  CHR$(7)


```

La parola è l'acronimo della frase in lingua inglese *Beginner's All-purpose Symbolic Instruction Code* ovvero - in italiano - "**codice simbolico di istruzioni adatto a ogni esigenza dei principianti**".



Il Commodore BASIC (nelle prime versioni PET BASIC) è il dialetto del linguaggio BASIC usato nella linea a 8 bit degli home computer della **Commodore International**, a partire dal PET del 1977 fino al C128 del 1985.



Dopo la comparsa del Fortran nacquero una serie di altri linguaggi di programmazione storici: il **Lisp** (1959) e l'**ALGOL** (1960).

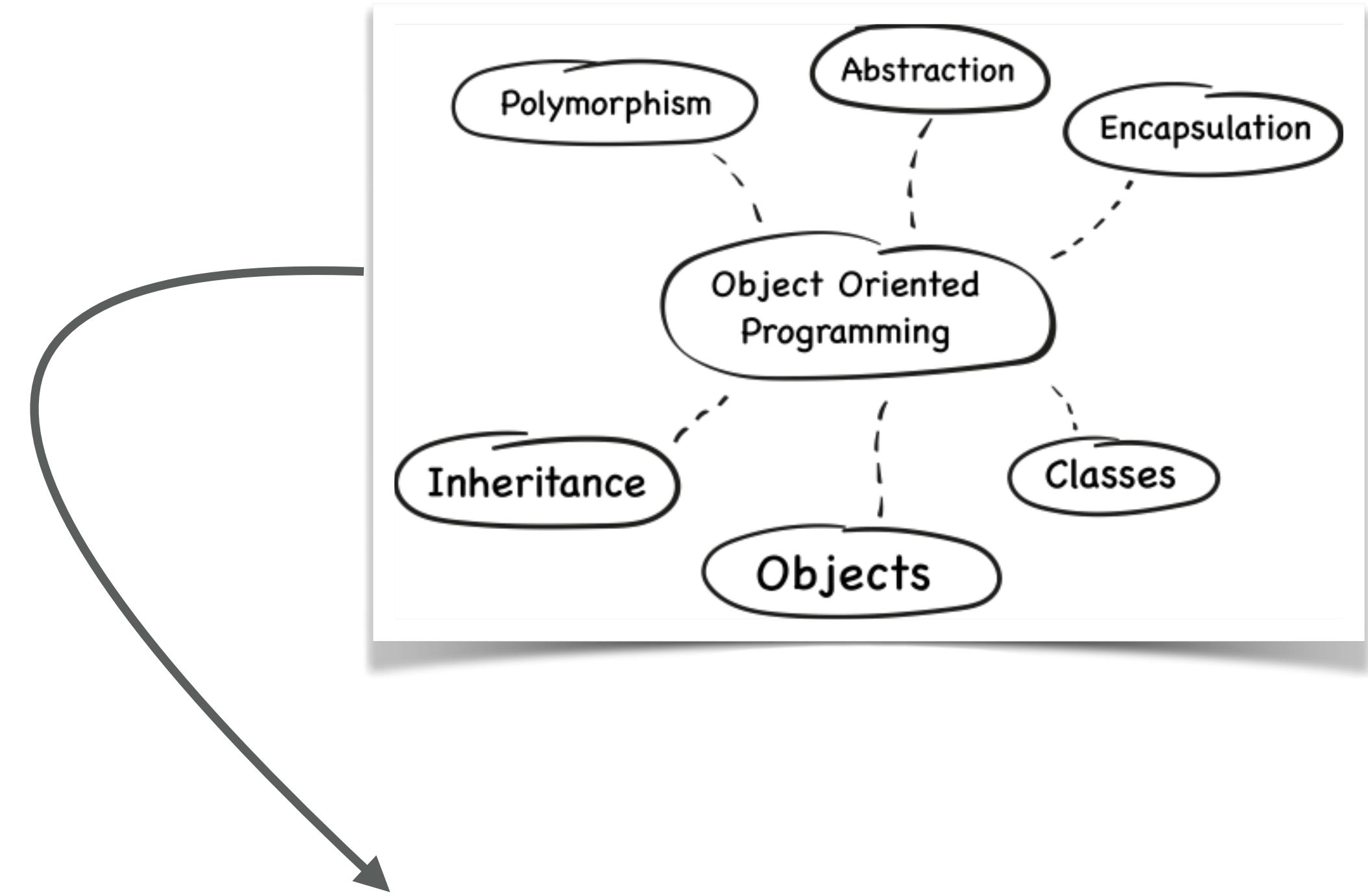
Tutti i linguaggi di programmazione oggi esistenti possono essere considerati discendenti da uno o più di questi primi linguaggi, di cui mutuano molti concetti di base; l'ultimo grande progenitore dei linguaggi moderni fu il **Simula** (1967), che introdusse per primo il concetto di oggetto software.

Nel 1970 viene introdotto il **Pascal**, il primo linguaggio strutturato, a scopo didattico; nel 1972 dal BCPL nascono prima il B (rapidamente dimenticato) e poi il **C**, che invece fu fin dall'inizio un grande successo.

Nello stesso anno compare anche il **Prolog**, finora il principale esempio di linguaggio logico.

Con i primi mini e microcomputer e le ricerche a Palo Alto, nel 1983 vede la luce **Smalltalk**, il primo linguaggio realmente e completamente ad oggetti, che si ispira al Simula e al Lisp: oltre a essere in uso tutt'oggi in determinati settori, Smalltalk viene ricordato per l'influenza enorme che ha esercitato sulla storia dei linguaggi di programmazione, introducendo il **paradigma object-oriented** nella sua prima incarnazione matura.

Esempi di linguaggi object-oriented odierni sono **Eiffel** (1986), **C++** (che esce nello stesso anno di Eiffel) e successivamente **Java**, classe 1995.



La programmazione orientata agli oggetti (OOP - *object-oriented programming*) è un paradigma di programmazione che permette di definire oggetti software in grado di interagire gli uni con gli altri attraverso lo scambio di messaggi.

È particolarmente adatta nei contesti in cui si possono definire delle relazioni di interdipendenza tra i concetti da modellare (contenimento, uso, specializzazione). Un ambito che più di altri riesce a sfruttare i vantaggi della programmazione a oggetti è quello delle interfacce grafiche.

Concetti fondamentali

Tutti i linguaggi di programmazione esistenti sono definiti da un lessico, una sintassi e una semantica e possiedono:

1. **Istruzione** un comando oppure una regola descrittiva
2. **Variabile e costante** un dato o un insieme di dati, noti o ignoti, già memorizzati o da memorizzare
3. **Espressione** una combinazione di variabili e costanti, unite da operatori
4. **Strutture dati** meccanismi che permettono di organizzare e gestire dati complessi
5. **Strutture di controllo** che permettono di governare il flusso di esecuzione del programma
6. **Sottoprogramma** un blocco di codice che può essere richiamato da qualsiasi altro punto del programma
7. **I/O** Funzionalità di input dati da tastiera e visualizzazione dati in output (stampa a video)
8. **Indexing** Possibilità di inserire dei commenti sul codice scritto, sintatticamente identificati e delimitati, che ne esplichino le funzionalità a beneficio della leggibilità o intelligenza.



Codice sorgente

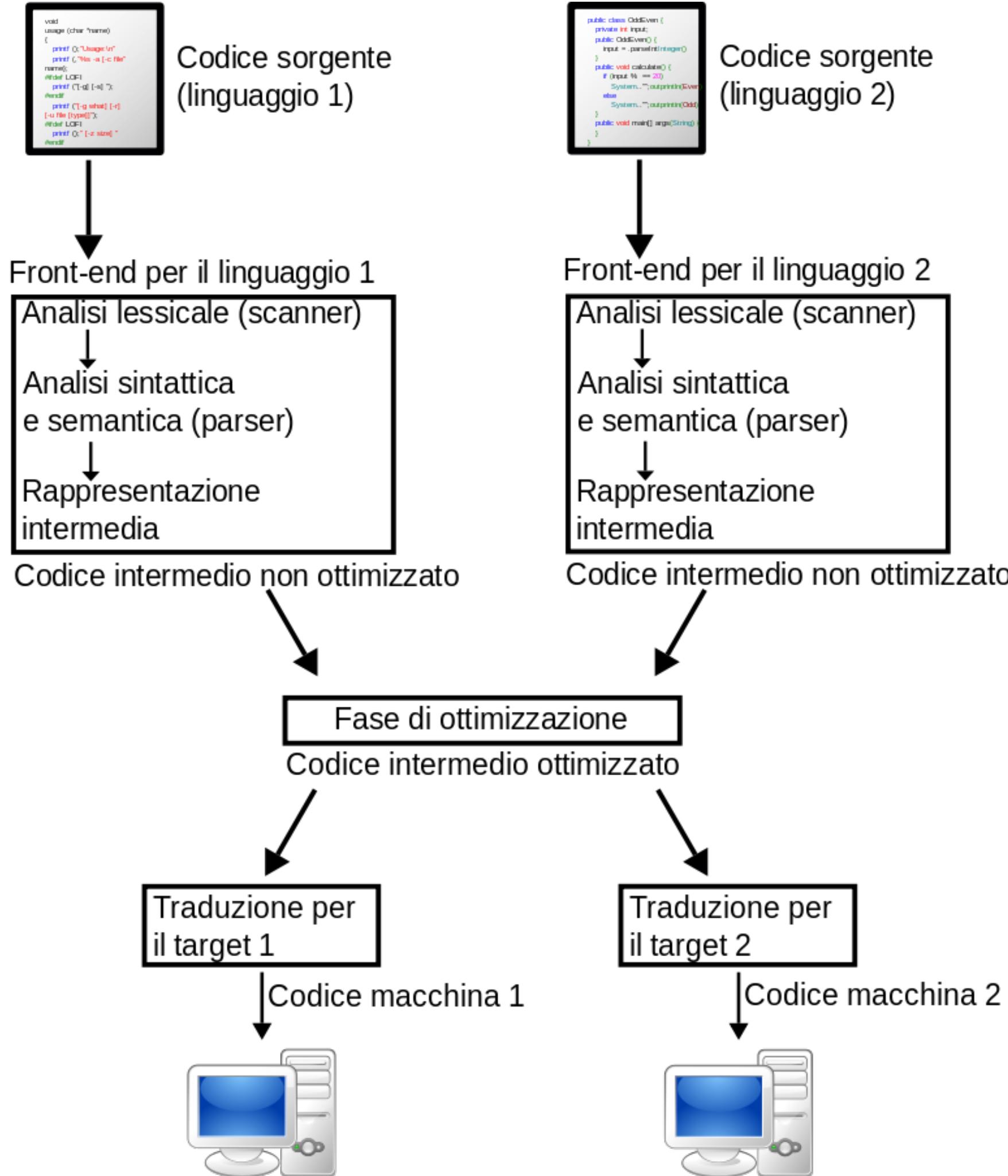
Programmare in un dato linguaggio di programmazione significa generalmente scrivere uno o più semplici file di testo

ASCII, chiamato codice sorgente che esprime l'algoritmo del programma tradotto nel linguaggio di programmazione.

I font, i colori e in generale l'aspetto grafico sono irrilevanti ai fini della programmazione in sé: per questo i programmatore non usano programmi di videoscrittura, ma degli editor di testo (come emacs e brief) che invece offrono funzioni avanzate di trattamento testi (espressioni regolari, sostituzioni condizionali e ricerche su file multipli, possibilità di richiamare strumenti esterni).

```
int leds[] = {12,11,10,9,8,7};  
int delay_duration = 1000;  
  
// the setup routine runs once when you press reset:  
void setup() {  
    // initialize the digital pin as an output.  
  
    for ( int i = 0; i < 6; ++i )  
    {  
        pinMode(leds[i], OUTPUT);  
        digitalWrite(leds[i], LOW);  
    }  
  
}  
  
// the loop routine runs over and over again forever:  
void loop() {  
  
    for ( int i = 1; i < 6; ++i )  
    {  
        digitalWrite(leds[i], HIGH);  
        delay(delay_duration);  
        digitalWrite(leds[i], LOW);  
    }  
  
    for ( int i = 4; i >= 0; --i )  
    {  
        digitalWrite(leds[i], HIGH);  
        delay(delay_duration);  
        digitalWrite(leds[i], LOW);  
    }  
}
```

Compilazione

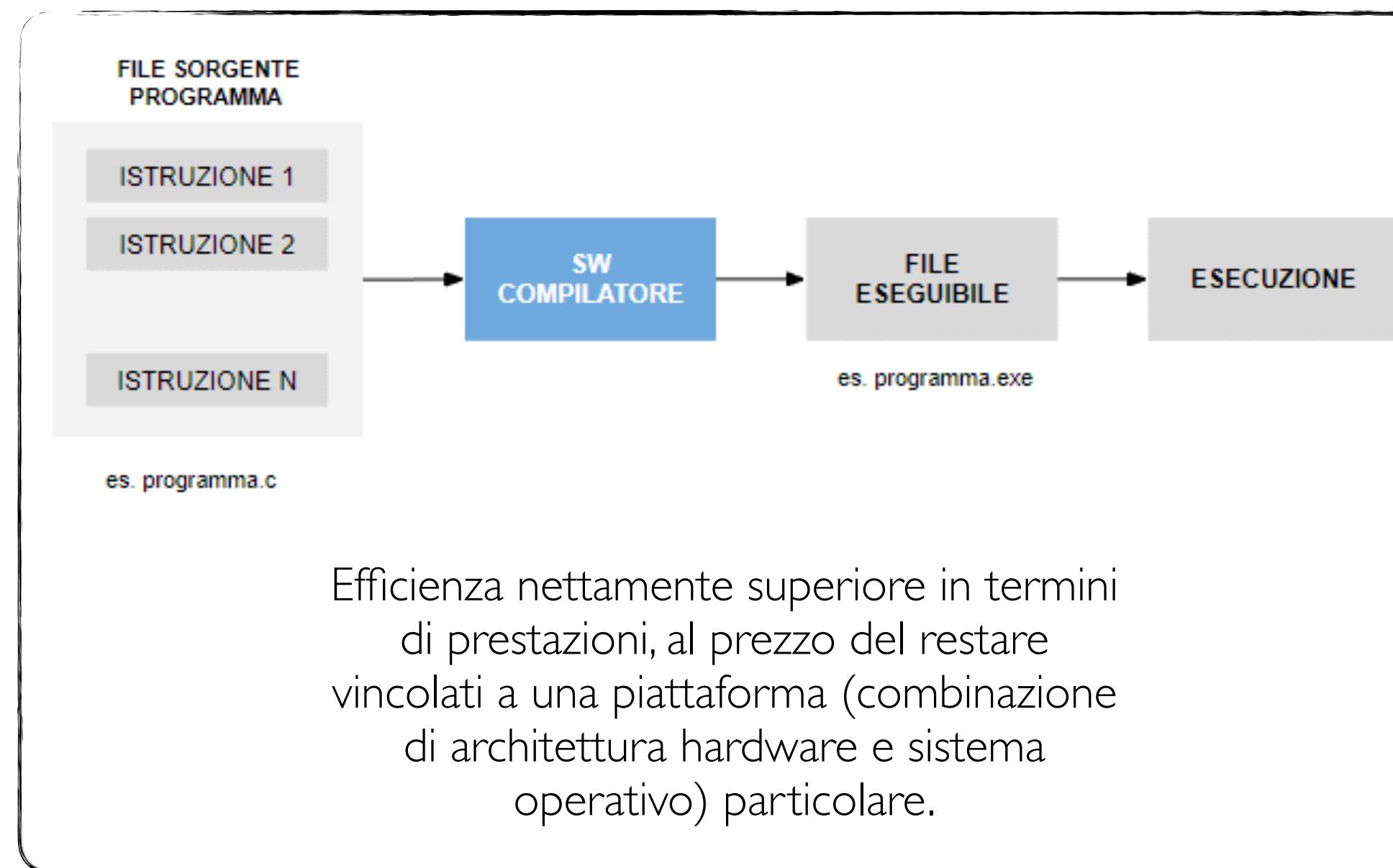


La compilazione è il processo per cui il programma, scritto in un linguaggio di programmazione ad alto livello, viene tradotto in un codice eseguibile per mezzo di un altro programma detto appunto compilatore. La compilazione offre numerosi vantaggi, primo fra tutti il fatto di ottenere eseguibili velocissimi nella fase di run (esecuzione) adattando vari parametri di questa fase all'hardware a disposizione; ma ha lo svantaggio principale nel fatto che è necessario compilare un eseguibile diverso per ogni sistema operativo o hardware (piattaforma) sul quale si desidera rendere disponibile l'esecuzione ovvero viene a mancare la cosiddetta portabilità.

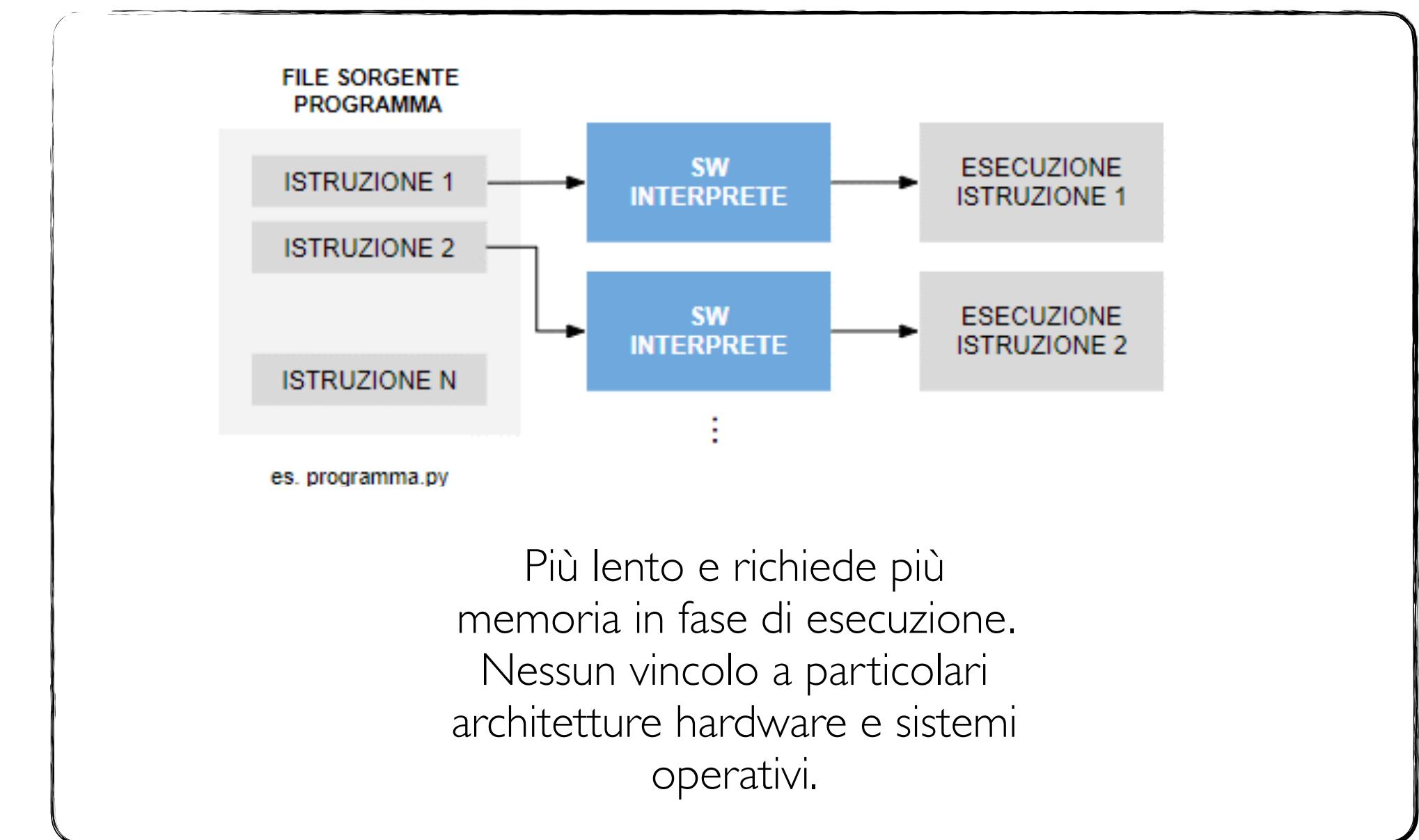
Interpretazione

Per risolvere il problema della portabilità (la dipendenza o meno del linguaggio dalla piattaforma) sono stati creati dei linguaggi basati su librerie compilate (componenti) ad hoc per ogni piattaforma, nei quali il codice sorgente viene eseguito direttamente, quindi non c'è la necessità di una compilazione su ogni tipologia di macchina su cui viene eseguito. Il difetto di questi linguaggi è la lentezza dell'esecuzione; però hanno il pregio di permettere di usare lo stesso programma senza modifica su più piattaforme. Si dice in questo caso che il programma è **portabile**.

Si usano linguaggi interpretati nella fase di messa a punto di un programma per evitare di effettuare numerose compilazioni o invece quando si vuole creare software che svolgono operazioni non critiche che non necessitano di ottimizzazioni riguardanti velocità o dimensioni, ma che traggono più vantaggio dalla portabilità.



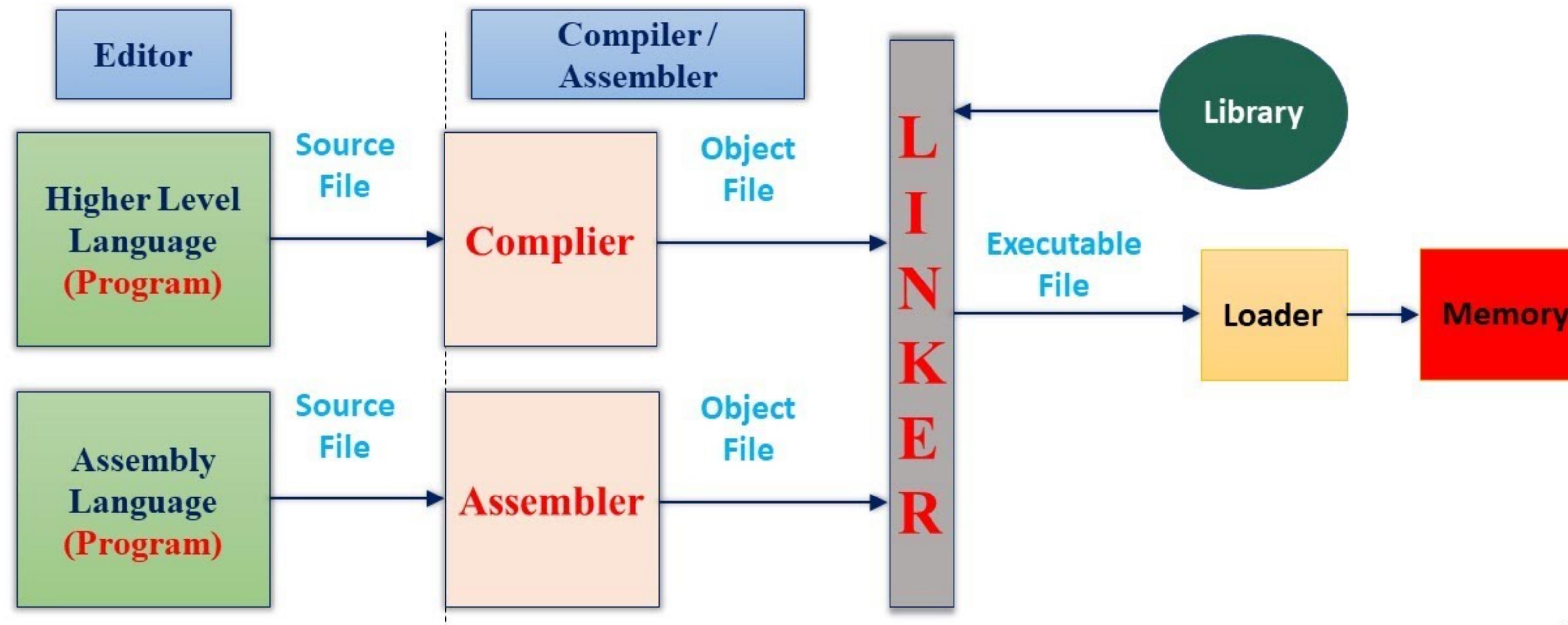
Efficienza nettamente superiore in termini di prestazioni, al prezzo del restare vincolati a una piattaforma (combinazione di architettura hardware e sistema operativo) particolare.



Più lento e richiede più memoria in fase di esecuzione. Nessun vincolo a particolari architetture hardware e sistemi operativi.

Collegamento/Linking

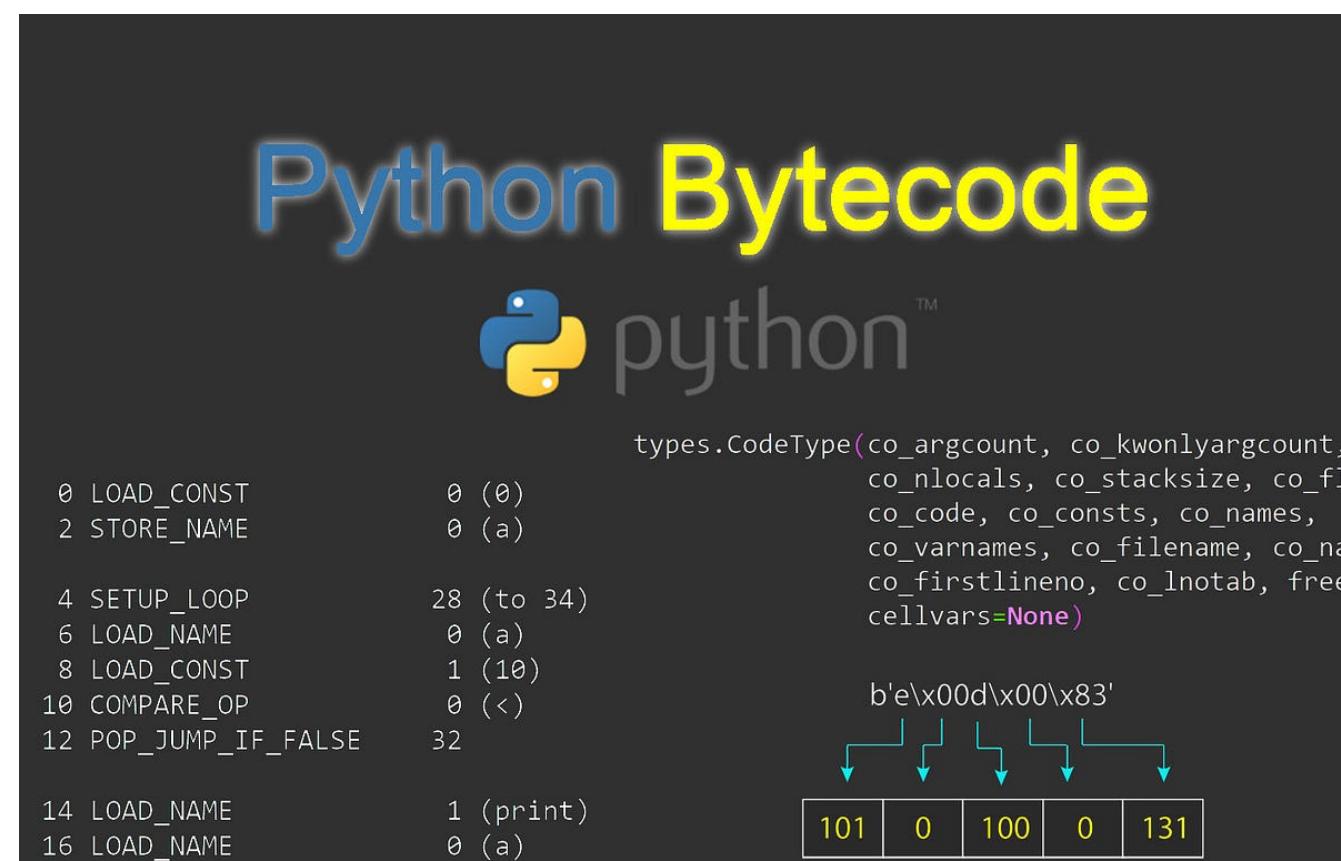
Se il programma, come spesso accade, usa delle librerie, o è composto da più moduli software, questi devono essere 'collegati' tra loro. Lo strumento che effettua questa operazione è detto appunto linker ("collegatore"), e si occupa principalmente di risolvere le interconnessioni tra i diversi moduli.



Bytecode e P-code

Una soluzione intermedia fra compilazione e interpretazione è stata introdotta nelle prime versioni di Pascal e successivamente adottata nei linguaggi **Java** e **Python**, con il bytecode, e nei linguaggi **Visual Basic** e **.NET** di Microsoft con il *P-code*.

Il codice sorgente dei programmi non viene compilato in linguaggio macchina, ma in un codice intermedio "ibrido" destinato a venire interpretato al momento dell'esecuzione del programma: il motivo di questo doppio passaggio è di avere la portabilità dei linguaggi interpretati ma anche, grazie alla pre-compilazione, una fase di interpretazione più semplice e quindi più veloce.



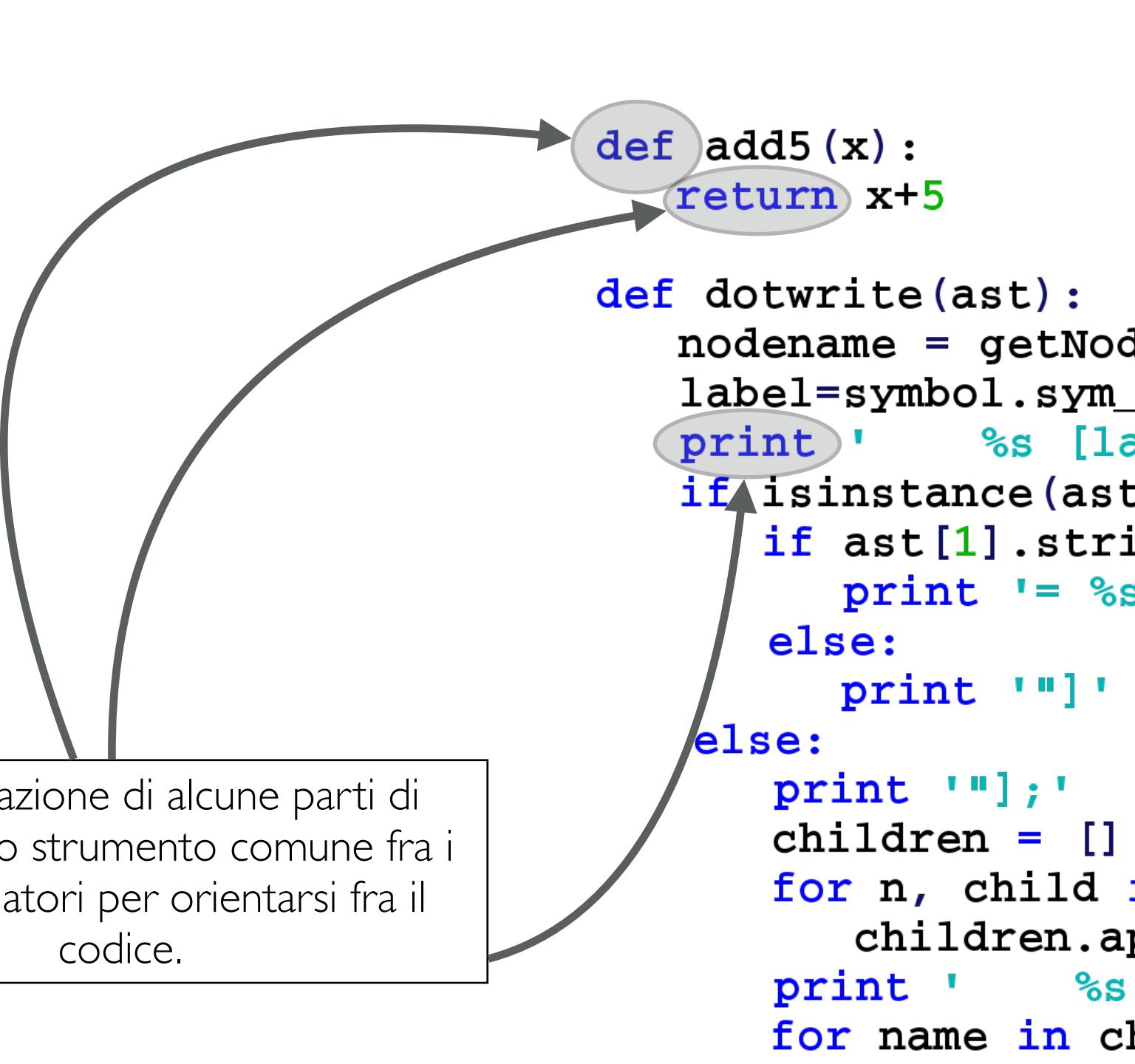
```
Python Bytecode
python

0 LOAD_CONST
2 STORE_NAME
4 SETUP_LOOP
6 LOAD_NAME
8 LOAD_CONST
10 COMPARE_OP
12 POP_JUMP_IF_FALSE
14 LOAD_NAME
16 LOAD_NAME

0 (0)
0 (a)
28 (to 34)
0 (a)
1 (10)
0 (<)
32
b'e\x00d\x00\x83'
1 (print)
0 (a)

types.CodeType(co_argcount, co_kwonlyargcount,
co_nlocals, co_stacksize, co_filename,
co_code, co_consts, co_names,
co_varnames, co_lineno, co_nlocals, free,
cellvars=None)
```

L'evidenziazione di alcune parti di codice è uno strumento comune fra i programmatori per orientarsi fra il codice.

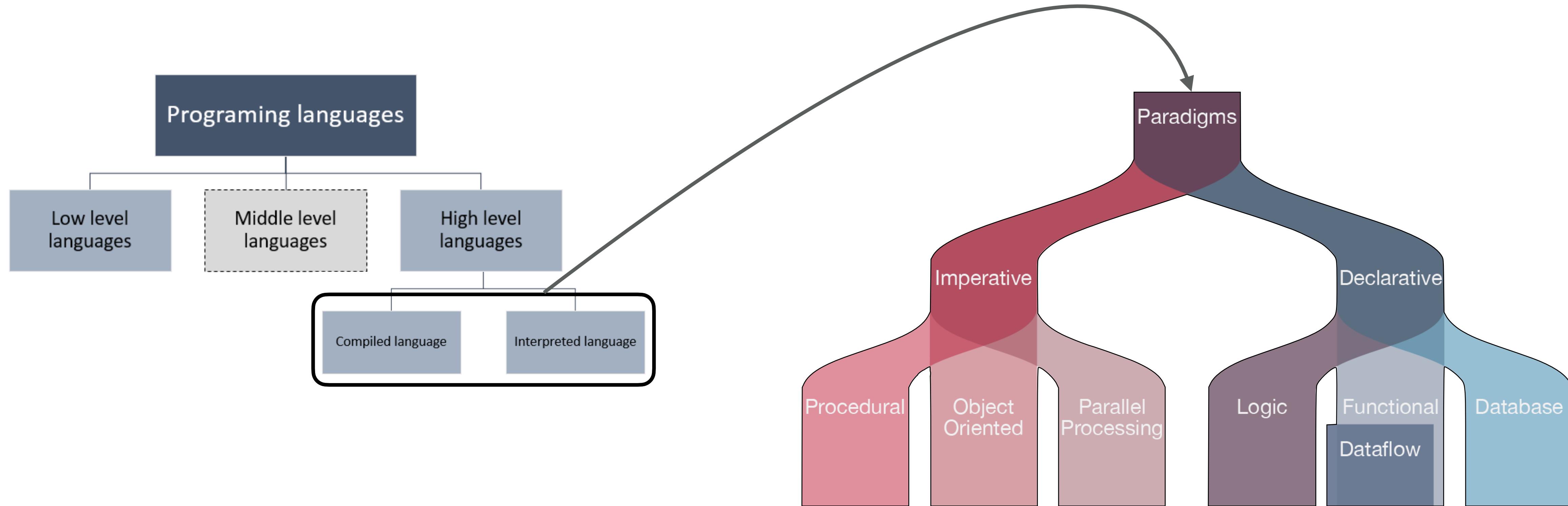


```
def add5(x):
    return x+5

def dotwrite(ast):
    nodename = getNodeName()
    label=symbol.sym_name.get(int(ast[0]),ast[0])
    print '%s [%label=%s] % (nodename, label),
    if isinstance(ast[1], str):
        if ast[1].strip():
            print '=' % ast[1]
        else:
            print ''
    else:
        print ']'
    children = []
    for n, child in enumerate(ast[1:]):
        children.append(dotwrite(child))
    print '%s -> {' % nodename,
    for name in children:
        print '%s' % name,
```

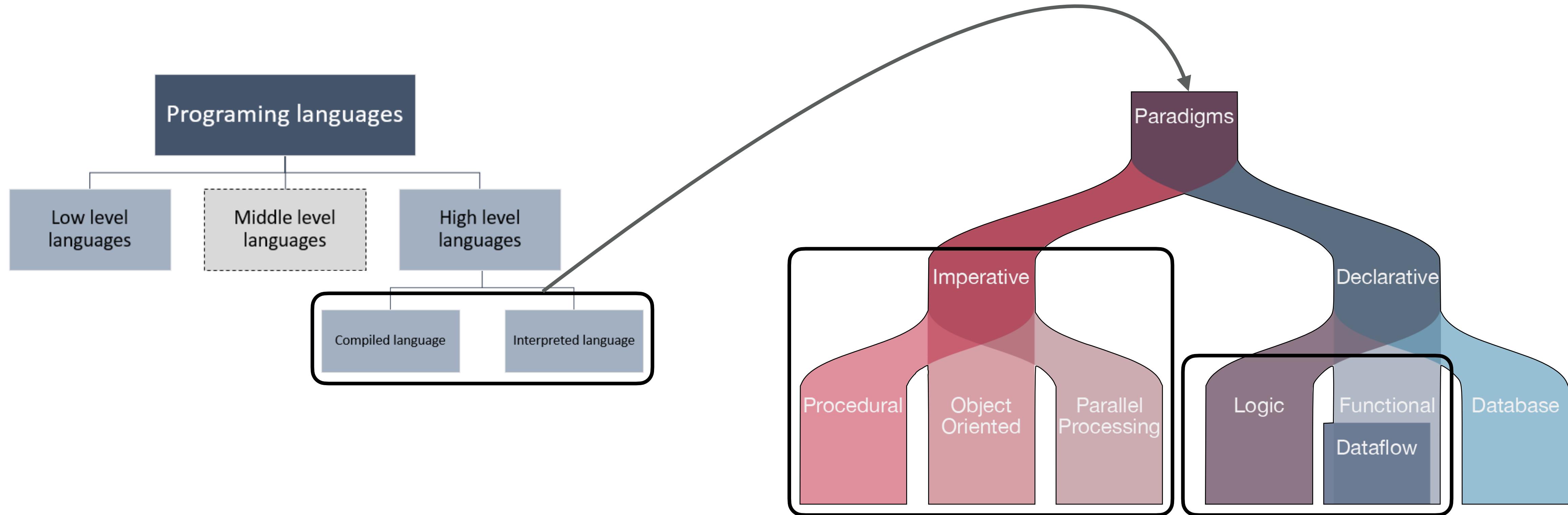
Ambiente di sviluppo e di esecuzione

Con ambiente di sviluppo si intendono l'insieme degli strumenti atti allo sviluppo del codice sorgente del programma, mentre con ambiente di esecuzione si intende tipicamente il complesso delle librerie software, detta anche piattaforma software, utilizzate dal programma stesso per poter funzionare correttamente.



Ambiente di sviluppo e di esecuzione

Con ambiente di sviluppo si intendono l'insieme degli strumenti atti allo sviluppo del codice sorgente del programma, mentre con ambiente di esecuzione si intende tipicamente il complesso delle librerie software, detta anche piattaforma software, utilizzate dal programma stesso per poter funzionare correttamente.





Ambiente di sviluppo e di esecuzione

Nei **linguaggi imperativi** l'istruzione è un comando esplicito, che opera su una o più variabili oppure sullo stato interno della macchina, e le istruzioni vengono eseguite in un ordine prestabilito. Scrivere un programma in un linguaggio imperativo significa essenzialmente occuparsi di cosa la macchina deve fare per ottenere il risultato che si vuole, e il programmatore è impegnato nel mettere a punto gli algoritmi necessari a manipolare i dati. Le strutture di controllo assumono la forma di **istruzioni di flusso** (**GOTO**, **FOR**, **IF/THEN/ELSE** ecc.) e il calcolo procede per **iterazione** piuttosto che per **ricorsione**.

THE
C
PROGRAMMING
LANGUAGE


COBOL

BASIC
PROGRAMMING

ASSEMBLY
LANGUAGE

F(OR)Tr[AN]

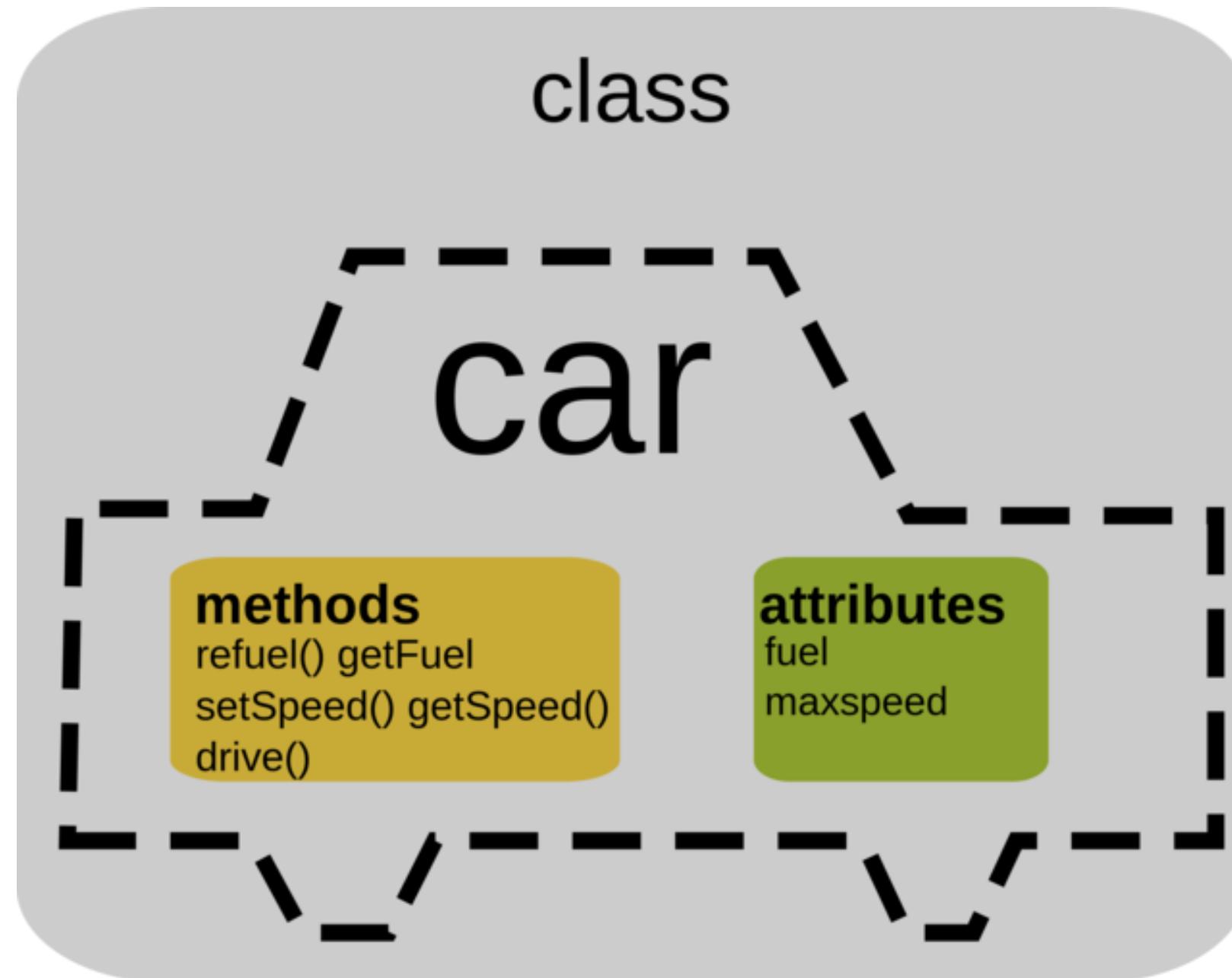


Ambiente di sviluppo e di esecuzione

La programmazione a oggetti è basata su un'evoluzione del concetto di tipo di dato astratto caratterizzata da encapsulamento, ereditarietà, polimorfismo. Oltre a linguaggi specializzati che implementano completamente i principi di tale metodologia (come Smalltalk o Java), molti linguaggi moderni incorporano alcuni concetti della programmazione a oggetti.



Object Oriented Language



Pensiamo ad una automobile (**l'oggetto**).

Sarà sicuramente composta da elementi come: ruote, sospensioni, freni, sedili (le **variabili**). Di quest'auto possiamo definire quali azione deve eseguire, come accelerare, frenare, sterzare (le **funzioni**).

Possiamo anche decidere di cambiare il numero dei cavalli, la velocità massima, l'autonomia con un pieno. Anche qui interverranno le funzioni, che si occuperanno di eseguire le modifiche a delle variabili (velocità massima, numero cavalli, autonomia con un pieno).

Il vantaggio nell'usare delle classi è evidente: posso **istanziare** una classe infinite volte creando oggetti simili, ma ad ognuna posso dare parametri diversi, personalizzando di fatto gli oggetti.

Possiamo quindi creare diversi oggetti utilizzando un solo codice.

ESEMPIO: creo la classe automobile una sola volta ma la istanzio 3 volte con parametri e funzioni diverse, creando di fatto: una Ferrari Enzo, una Fiat Panda ed un Land Rover Defender.Tre auto che differenziano da loro solo per dei parametri, ma che sono sostanzialmente appartenenti alla classe auto.

| **ISTANZIARE UNA CLASSE:** costruire un oggetto manipolabile derivato da una classe

Domande fondamentali

Come viene rappresentata TUTTA l'informazione all'interno del computer?

Quali istruzioni esegue un computer?

Quali problemi può risolvere un computer?

Esistono problemi che un computer non può risolvere?



Domande fondamentali

Algoritmo

Sequenza finita di mosse che risolve in un tempo finito una classe di problemi

Codifica o implementazione

Fase di scrittura di un algoritmo attraverso un insieme ordinato di frasi (“istruzioni”), scritte in un linguaggio di programmazione, che specificano le azioni da compiere in modo formale interpretabile dal computer



Domande fondamentali

Algoritmo

Testo scritto secondo la *sintassi* (alfabeto + regole grammaticali) e la *semantica* di un linguaggio di programmazione

La programmazione...

È l'attività con cui si predisponde l'elaboratore ad eseguire un particolare insieme di azioni su particolari dati, allo scopo di risolvere un problema.



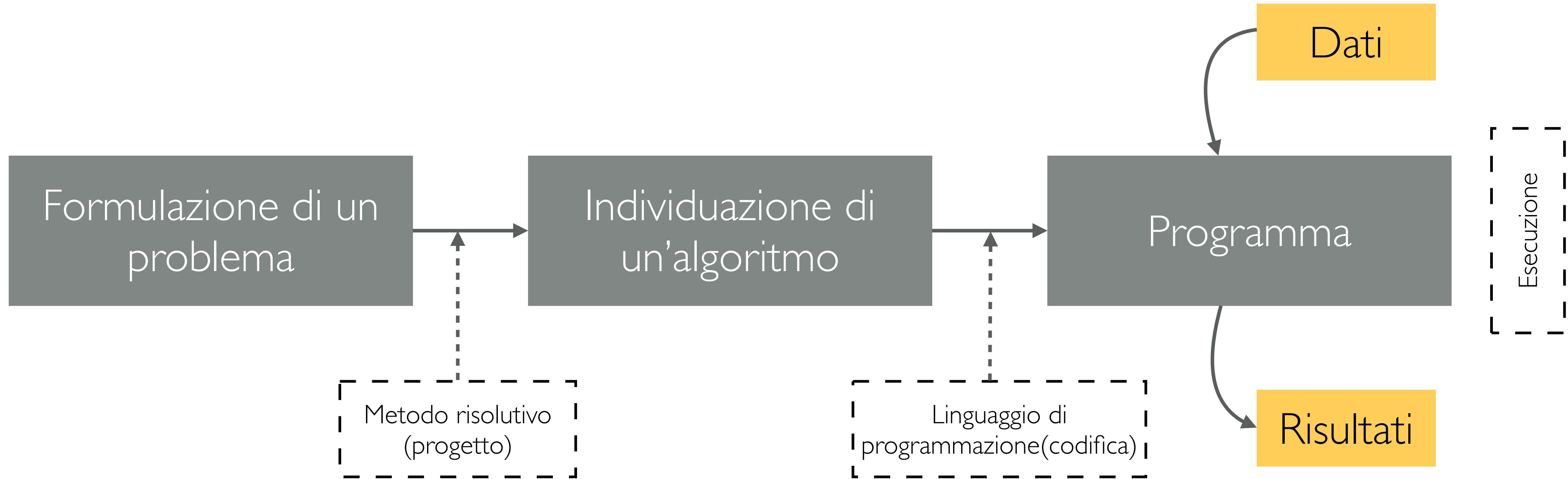
Algoritmo e Programma a confronto

Ogni elaboratore è una macchina in grado di eseguire azioni elementari su dati
L'esecuzione delle azioni elementari è richiesta all'elaboratore tramite comandi chiamati istruzioni
Le istruzioni sono espresse attraverso frasi di un opportuno linguaggio di programmazione

Un programma non è altro che la formulazione testuale di un algoritmo in un linguaggio di programmazione

Esecuzione di un programma

L'esecuzione delle azioni nell'ordine specificato dall'algoritmo consente di ottenere, a partire dai dati di ingresso, i risultati che risolvono il problema

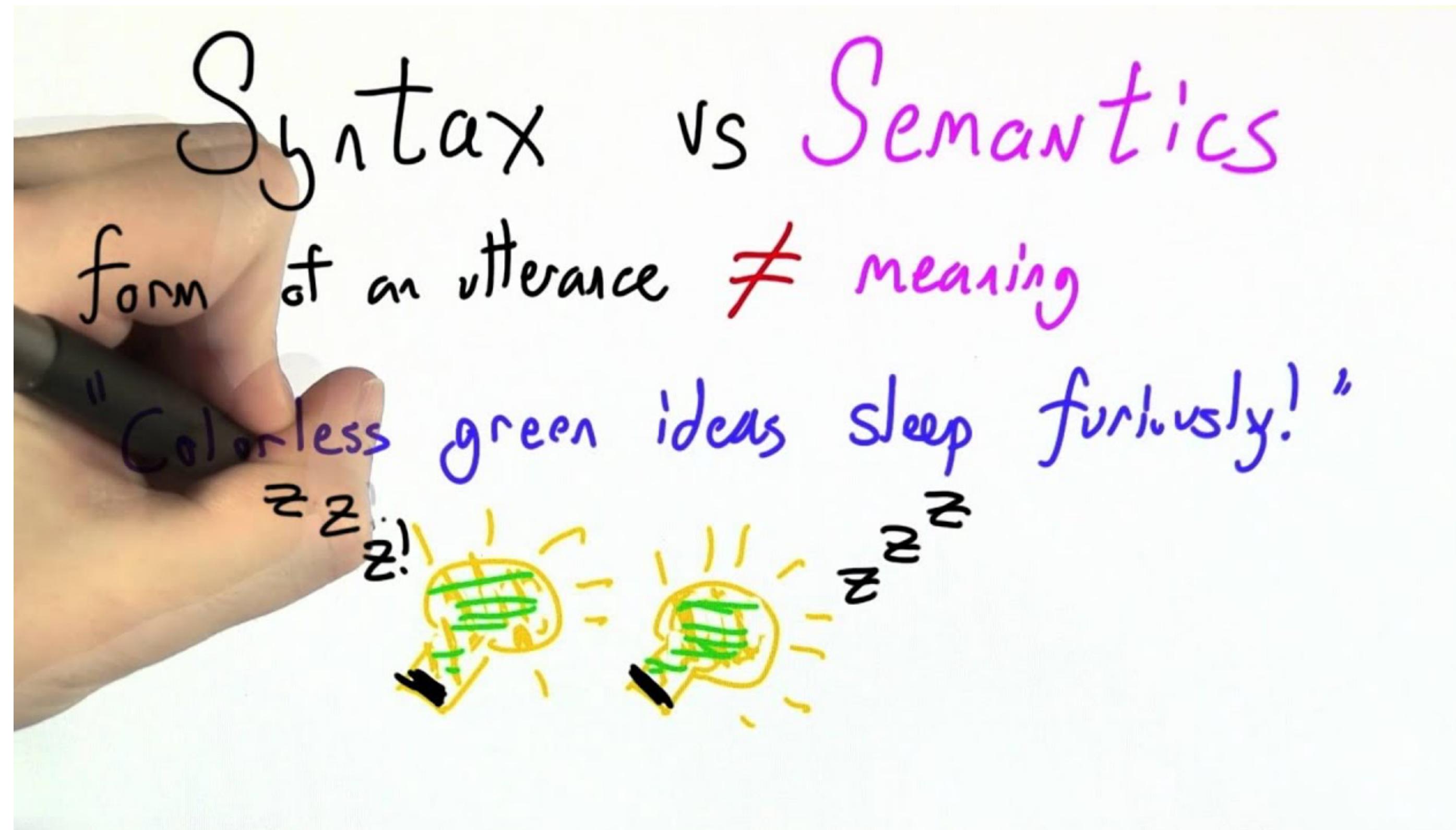


Per la codifica di un algoritmo...

Bisogna conoscere la **sintassi** di un linguaggio di programmazione

Bisogna conoscere la **semantica** di un linguaggio di programmazione

Bisogna conoscere un ambiente di programmazione



La **sintassi** di un linguaggio di programmazione viene utilizzata per indicare la struttura dei programmi senza considerare il loro significato. Fondamentalmente sottolinea la struttura, il layout di un programma con il loro aspetto. Implica una raccolta di regole che convalida la sequenza di simboli e istruzioni utilizzate in un programma.

Con **semantica** si intende tutto quello che è il significato di una frase. Se voi state capendo quello che vi sto dicendo, è perché state capendo il significato. La semantica è anche quella cosa che ci permette di capire simboli astratti e complessi, i modi di dire e tutto quanto. Un simbolo appartiene alla sfera semantica. Noi riusciamo a decodificarne il significato perché la nostra mente è dotata di capacità di elaborazione, interpretazione, e simbolica.

Sintassi: insieme di regole per la costruzione di frasi corrette

Semantica: insieme di regole per l'attribuzione di un significato alle frasi

Una frase può essere semanticamente corretta e tuttavia non avere significato

Oggi l'elefante ha un bel colore rosa