

UNIVERSITÀ DEGLI STUDI DI SALERNO

# Fondamenti di Informatica

---

Introduzione alla Programmazione

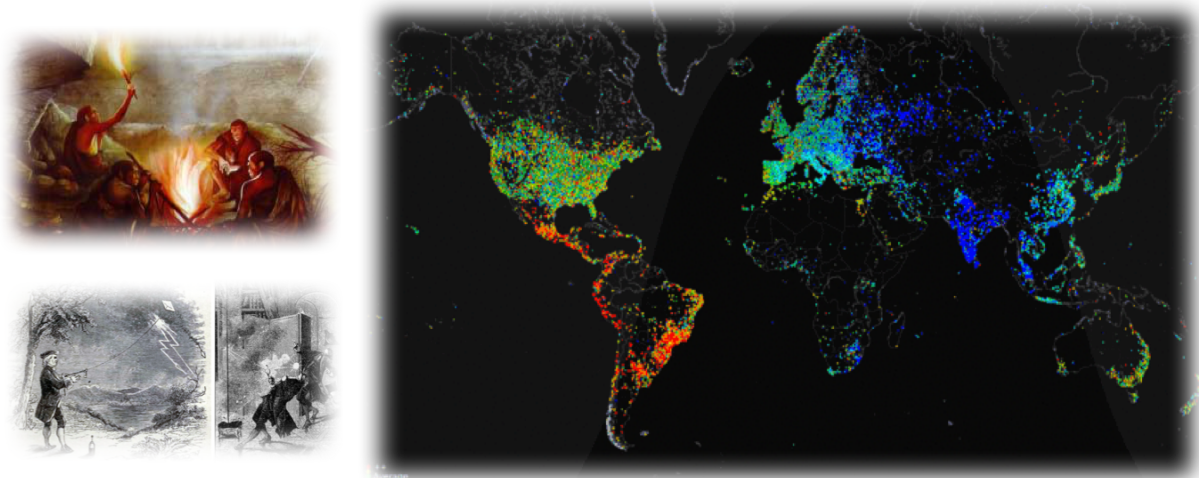
Prof. Arcangelo Castiglione

A.A. 2016/17

# I Programmi – 1/2

---

- Negli ultimi anni l'Information & Communication Technology (ICT) è diventato un elemento strategico per il business, oltre che uno strumento essenziale in tutte le attività lavorative e sociali
  - Ha acquisito un valore irrinunciabile per tutte le istituzioni pubbliche e private, per le imprese, ma soprattutto per tutti noi
- Il programma rappresenta l'elemento di base per il mondo dell'ICT



# I Programmi – 2/2

---

- Fanno ormai parte di ogni nostra attività
  - Sistemi Operativi
  - Virus, Troian, Malware
  - Videogiochi
  - ATM, semafori, il motore di ricerca Google, sistemi per il supporto del trasporto
  - E molto altro ancora...
- Differiscono sotto vari aspetti
  - Obiettivi
  - Funzionalità
  - Algoritmi
  - Risorse
  - Etc

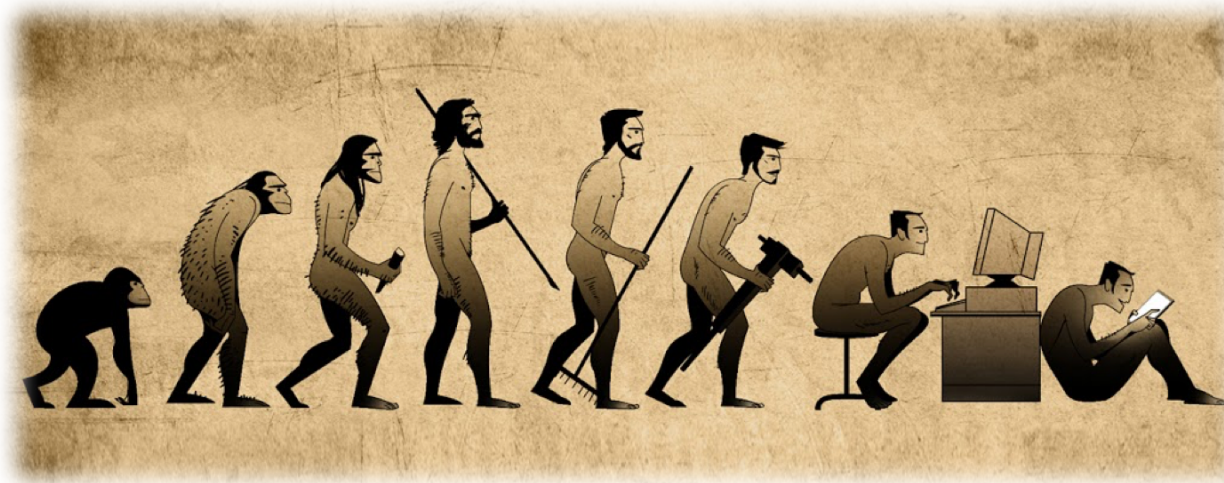




# Perché imparare a programmare? – 2/7

---

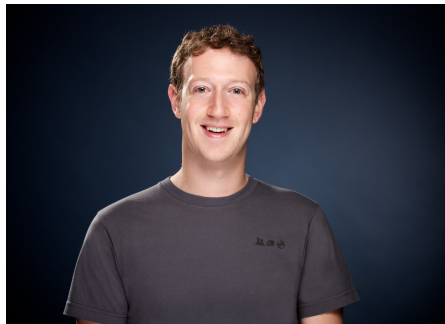
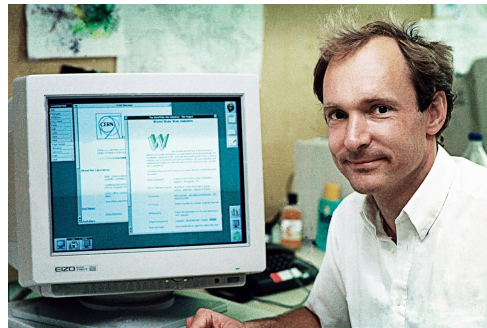
- Imparare un nuovo linguaggio permette di
  - Usare nuove forme espressive
  - Creare oggetti che in realtà sembrano non tangibili, ma che in realtà lo sono
- Lo sono perché in molti casi hanno cambiato il nostro modo di vivere, ma anche per il loro impatto economico



# Perché imparare a programmare? – 3/7

---

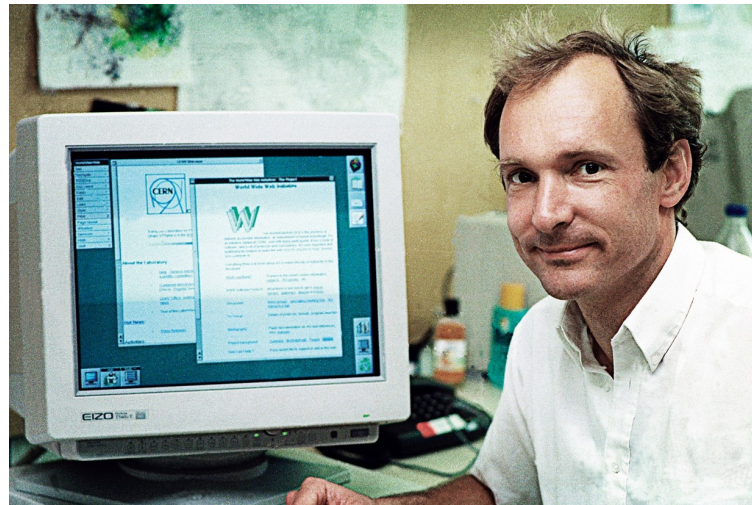
- Utilizzando idee creative e competenze di programmazione è possibile creare tecnologie rivoluzionarie stando comodamente seduti al proprio computer
  - La programmazione consente alle nostre idee di prendere forma e materializzarsi
  - Di solito il tutto nasce dalla necessità di risolvere problemi concreti



# Perché imparare a programmare? – 4/7

---

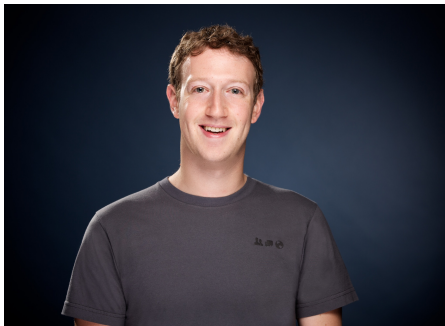
- Il CERN (*Conseil européen pour la recherche nucléaire*), il più grande laboratorio al mondo di fisica delle particelle, aveva necessità di strumenti software (programmi) che permettessero la diffusione di informazioni fra i suoi diversi centri di ricerca
  - Per risolvere questo problema, Sir Timothy John Berners-Lee sviluppò i programmi che rappresentano la base concettuale per il World Wide Web (WWW)



# Perché imparare a programmare? – 5/7

---

- Durante i suoi anni universitari Mark Zuckerberg non riusciva a trovare un modo efficace per raccogliere informazioni sugli altri studenti della sua università (Harvard)
  - Per risolvere questo problema ha creato Facebook
  - Originariamente progettato per gli studenti dell'Università di Harvard, fu presto aperto anche agli studenti di altre università e scuole
  - “Facebook” prende spunto da un elenco con nome e fotografia degli studenti, che alcune università statunitensi distribuiscono all'inizio dell'anno accademico per aiutare gli iscritti a socializzare tra loro





# Perché imparare a programmare? – 6/7

---

- Fino al febbraio 2005 non era possibile condividere e rendere pubblici sulla rete Internet i propri video
  - Per risolvere questo problema, 3 ragazzi che lavoravano a Paypal (Chad Hurley, Steve Chen e Jawed Karim) crearono YouTube



# Perché imparare a programmare? – 7/7

---

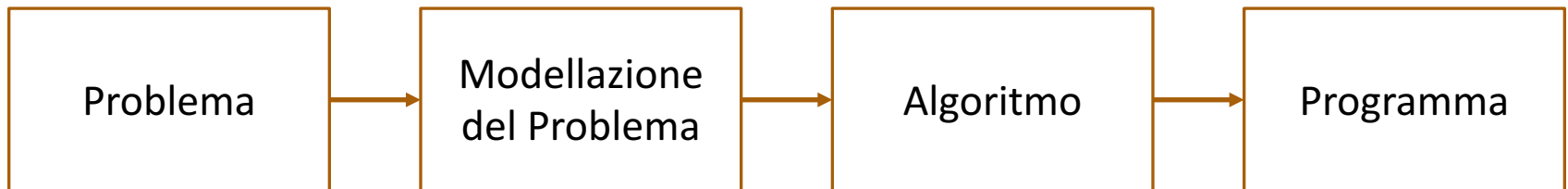
- Larry Page e Sergey Brin immaginarono un luogo che permettesse loro di accedere, in maniera semplice e rapida, a tutte le informazioni presenti sulla rete Internet
  - Per fare questo hanno creato Google, un potente motore di ricerca che viene utilizzato da diversi miliardi di persone in tutto il mondo



# Cos'è un Programma – 1/2

---

- **Definizione 1:** implementazione di un algoritmo espressa in un linguaggio di programmazione specifico
- **Definizione 2:** notazione (formale e non ambigua) con cui è possibile descrivere gli algoritmi



# Cos'è un Programma – 2/2

---

- **Osservazione:** *“il calcolatore deve capire quello che gli viene detto e deve saper fare ciò che gli viene chiesto”*
  - Risolvere problemi complessi mediante azioni elementari
  - Le istruzioni corrispondono ad azioni elementari
    - Somma, differenza, prodotto, divisione (e poco più)
  - Istruzioni e dati sono numeri binari
- **Problemi connessi**
  - Interagire con la macchina mediante un linguaggio più vicino agli umani
- **Soluzione**
  - Linguaggi di alto livello (MATLAB ed altri)
  - Traduttori/Compilatori/Interpreti



# Chi Crea un Programma?

---

- Il **programmatore** si occupa di
  - Progettare un algoritmo efficace per la risoluzione di un problema dato
  - Tradurre questo algoritmo in istruzioni eseguibili da un computer mediante un linguaggio di programmazione



# Come viene Scritto un Programma? – 1/8

---

- La **programmazione** consiste nella scrittura di un testo, detto **programma (o codice) sorgente**, che descrive in termini di istruzioni note alla macchina la soluzione per un dato problema
  - Esempio: ricerca del valor massimo in una serie di numeri
- In generale **non esiste una sola soluzione ad un certo problema**
  - Le soluzioni potrebbero essere numerose
- La programmazione consiste nel trovare la strada “migliore” che conduce alla soluzione del problema in oggetto
  1. Di solito si parte dal trovare una prima strada, che non deve essere per forza la migliore
  2. Successivamente, si cercano eventuali altre strade migliori

# Come viene Scritto un Programma? – 2/8

---

- **Programmare è un'operazione creativa**
  - Non esiste un problema uguale a un altro e non esistono soluzioni universali
- Programmatori diversi scrivono programmi diversi per risolvere lo stesso problema
  - Le soluzioni possono essere ugualmente efficienti
- Programmare è un'operazione organizzata per step successivi
- È completamente inefficiente un approccio “diretto”
  - Scrivere direttamente il programma definitivo partendo dal problema

# Come viene Scritto un Programma? – 3/8

---

- **Obiettivo:** risolvere un problema
- **Fasi del processo di programmazione**
  1. **Modellazione** del problema
  2. Ricerca della soluzione migliore (**idea**)
  3. Conversione dell'idea in una soluzione formale (**algoritmo**)
  4. Traduzione dell'algoritmo in una sequenza di istruzioni comprensibili all'esecutore (in questo caso l'elaboratore elettronico)
    - **Programma**
  5. Valutazione del programma con un insieme significativo di dati per garantire che funzionerà in ogni occasione (qualsiasi siano i dati di input)
  6. Opportuna documentazione del programma a beneficio di chi lo userà ed eventualmente lo modificherà



# Come viene Scritto un Programma? – 4/8

---

- I programmi sono intesi per essere eseguiti dai computer ma anche per essere letti dalle persone
  - Che possono essere anche diverse da quelle che hanno scritto il programma
- È necessario quindi migliorare il più possibile la leggibilità e la chiarezza dei programmi
  - Il **codice** relativo ad un **programma** presenta una **struttura gerarchica**: le istruzioni possono essere annidate all'interno di altre istruzioni
    - Quindi bisogna usare l'indentazione (rientro) in modo opportuno
  - ***Aggiungere commenti significativi*** (i commenti sono istruzioni non eseguite dall'elaboratore)
    - Prima di una dichiarazione di funzione/procedura spiegare a cosa essa serve e quali sono i suoi parametri
    - Dopo importanti dichiarazioni di variabili
    - Prima o dopo istruzioni importanti

# Come viene Scritto un Programma? – 5/8

---

**INIZIO ALGORITMO trovaMax**

*% La funzione A(1) restituisce il valore dell'elemento in posizione 1* ←

*max = A(1) % La variabile max memorizza il massimo valore corrente in A* ←

Per *i* che va da 2 a 10

*%La funzione A(i) restituisce il valore dell'elemento in posizione i* ←

Se  $A(i) > max$

*max = A(i) % Istruzione eseguita se  $A(i) > max$*  ←

Incrementa *i*

restituisci *max*

**FINE ALGORITMO trovaMax**

Commenti

# Come viene Scritto un Programma? – 5/8

**INIZIO ALGORITMO trovaMax**

*max = A(1)*

*For i = 2 to 10 %Inizio ciclo a condizione iniziale*

*If A(i) > max %Inizio blocco di selezione semplice*

*max = A(i) %Unica istruzione appartenente al blocco di selezione semplice*

*EndIf %Fine blocco di selezione semplice*

*Incrementa i*

*EndFor %Fine ciclo a condizione iniziale*

*restituisce max*

**FINE ALGORITMO trovaMax**

Istruzioni appartenenti al blocco del ciclo a condizione iniziale

Istruzioni appartenenti al blocco principale dell'algoritmo trovaMax

# Come viene Scritto un Programma? – 6/8

---

- Alcune funzionalità richiedono poche righe di codice
  - È buona prassi il raggruppare/mantenere queste funzionalità all'interno di blocchi separati di codice
    - Ad es., raggruppando il codice in funzioni
- **Consistent Naming Scheme**
  - I nomi di variabili e funzioni devono avere delimitatori di parola. Sono due le opzioni più comuni
    - **camelCase**: Prima lettera di ogni parola è in maiuscolo (di solito tranne la prima parola)
    - **underscore**: Underscore tra le parole (`_`), come ad esempio: `trova_max_nella_lista(A)`
    - Le due opzioni possono anche essere combinate

# Come viene Scritto un Programma? – 7/8

---

- Principio del **Don't Repeat Yourself**
  - Lo **scopo** per la maggior parte dei **programmi** (e dei computer in generale) è quello di **automatizzare** le **operazioni ripetitive**
  - Questo principio dovrebbe essere mantenuto in tutto il codice
  - Lo stesso pezzo di codice non deve essere ripetuto
- **Evitare troppi livelli di annidamento/indentazione**
  - Troppi livelli di indentazione possono rendere il codice più difficile da leggere e seguire
  - Evitare di scrivere in orizzontale lunghe righe di codice

# Come viene Scritto un Programma? – 8/8

---

- **Usare nomi significativi e coerenti per le variabili**
  - Le variabili devono essere descrittive
  - Usare nomi coerenti per le variabili che hanno lo stesso tipo di ruolo
- **Refactoring del codice**
  - Quando si fa “refactoring”, si apportano modifiche al codice senza cambiare nessuna delle sue funzionalità
  - Non include correzioni di errori
  - Si può pensare al “refactoring” come ad un’operazione di pulizia, fatta allo scopo di migliorare la leggibilità e la qualità del codice, soprattutto in chiave futura
  - È possibile migliorare la leggibilità del codice durante il processo di refactoring utilizzando i concetti visti poc’anzi

# Linguaggi di Programmazione – 1/2

---

- **Definizione:** un linguaggio di programmazione è un linguaggio artificiale per comunicare con le macchine
- Più precisamente, i linguaggi di programmazione sono di solito usati per
  - Esprimere algoritmi
  - Controllare il comportamento delle macchine
- Quanti sono i linguaggi di programmazione presenti al mondo?
  - Più di 1000
  - I più diffusi sono elencati qui: <http://www.tiobe.com/tiobe-index/>
- Si noti che ogni linguaggio di programmazione ha avuto origine da motivazioni specifiche

# Linguaggi di Programmazione – 2/2

Sep 2016	Sep 2015	Change	Programming Language	Ratings	Change
1	1		Java	18.236%	-1.33%
2	2		C	10.955%	-4.67%
3	3		C++	6.657%	-0.13%
4	4		C#	5.493%	+0.58%
5	5		Python	4.302%	+0.64%
6	7	▲	JavaScript	2.929%	+0.59%
7	6	▼	PHP	2.847%	+0.32%
8	11	▲	Assembly language	2.417%	+0.61%
9	8	▼	Visual Basic .NET	2.343%	+0.28%
10	9	▼	Perl	2.333%	+0.43%
11	13	▲	Delphi/Object Pascal	2.169%	+0.42%
12	12		Ruby	1.965%	+0.18%
13	16	▲	Swift	1.930%	+0.74%
14	10	▼	Objective-C	1.849%	+0.03%
15	17	▲	MATLAB	1.826%	+0.65%
16	34	▲	Groovy	1.818%	+1.31%
17	14	▼	Visual Basic	1.761%	+0.23%
18	19	▲	R	1.684%	+0.64%
19	44	▲	Go	1.625%	+1.37%



# Linguaggi di Programmazione: Elementi Costitutivi


---

- **Ogni linguaggio di programmazione**
  - Dispone di un insieme di “**parole chiave**”
    - Keyword
  - È caratterizzato da due componenti, complementari l’una con l’altra
    - Sintassi: insieme delle regole che specificano come comporre istruzioni ben formate
    - Semantica: specifica il significato di ogni istruzione ben formata, vale a dire la successione delle operazioni che vengono compiute quando l’istruzione viene eseguita

# Linguaggi di Programmazione: Classificazione

---

**Utente**

- 
- **Linguaggi di alto livello (vicini all'utente)**
    - **V generazione:** linguaggi di descrizione dei problemi orientati alla risoluzione automatica
    - **IV generazione:** linguaggi per specifici ambiti applicativi (ad es. **MATLAB**)
    - **III generazione:** linguaggi imperativi e procedurali di uso generale
  - **Linguaggi di basso livello (vicini all'hardware)**
    - **II generazione:** linguaggi assemblativi (uso di codici mnemonici per le istruzioni)
    - **I generazione:** linguaggi macchina (sequenze di bit)

**Hardware**

# Linguaggi di Programmazione di Prima Generazione

---

- Gli elaboratori comprendono in maniera diretta solo istruzioni in binario
  - Ciascun elaboratore ha il proprio set di istruzioni
- I primi programmi erano fortemente dipendenti dalla macchina
  - Più precisamente erano scritti in linguaggio macchina (o codice macchina)
- **Pro**
  - Molto efficiente per le macchine
- **Contro**
  - Difficile da programmare per gli umani
  - Non portabile
- Ancora molto utilizzato per la programmazione di funzioni a basso livello
  - Driver, interfacce verso firmware e hardware, etc

# Linguaggio Macchina

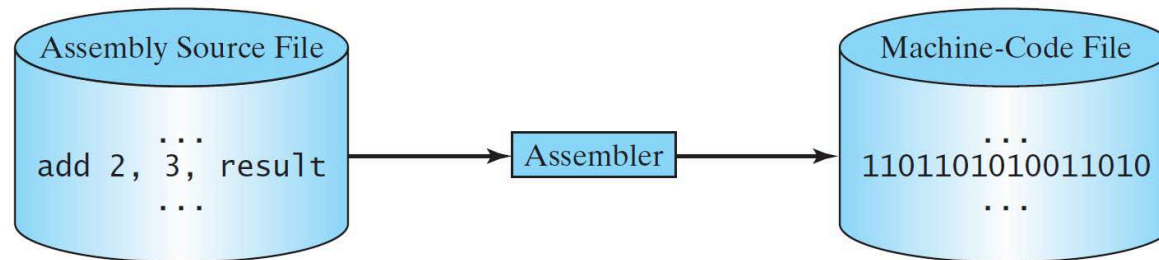
---

- Il linguaggio macchina è **direttamente eseguibile** dall'**elaboratore**
  - **Senza nessuna traduzione**
- **Istruzioni ed operandi relativi al programma in esecuzione sono caricati in memoria** e quindi sono **memorizzati in forma binaria**
- **Vincolo:** conoscenza dei metodi di rappresentazione delle informazioni utilizzati
- **Esempio**
  - Istruzione: **carica nel registro**
  - **10010000 11001100**

# Linguaggi di Programmazione di Seconda Generazione – 1/2

---

- Un **linguaggio assembly** (o **assemblativo**) utilizza codici mnemonici per rappresentare le istruzioni
  - Il codice può essere letto e scritto da programmatori umani
  - Ma è ancora fortemente dipendente dalla macchina



- Per essere eseguito da un elaboratore, un codice assembly deve essere convertito (da un **assemblatore**), in una forma comprensibile dalla macchina
  - Mediante un processo **chiamato assemblaggio**

# Linguaggi di Programmazione di Seconda Generazione – 2/2

---

- **Pro**

- Adatti ad essere usati in elaborazioni estremamente intensive
  - Giochi, video editing, manipolazione grafica, rendering, etc

- **Contro**

- Necessità di conoscere in dettaglio le caratteristiche della macchina (registri, dimensione dei dati, set di istruzioni, etc)
- Anche semplici algoritmi richiedono molte istruzioni

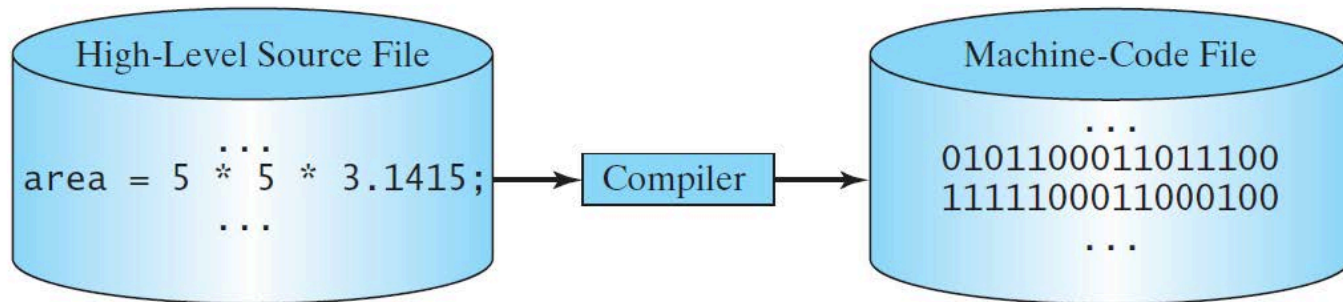
# Linguaggi di Programmazione di Terza Generazione – 1/2

---

- I **linguaggi di programmazione di alto livello** usano *parole English-like, notazione matematica e punteggiatura* per scrivere programmi
  - Sono più vicini ai linguaggi umani
  - Il programmatore può astrarre dai dettagli legati all'architettura ed esprimere i propri algoritmi in modo semplice
  - Le istruzioni esprimono una serie di azioni
- **Pro**
  - Portabili, indipendenti dalla macchina
  - Human-friendly
- **Contro**
  - Non sempre sono molto efficienti

# Linguaggi di Programmazione di Terza Generazione – 2/2

---



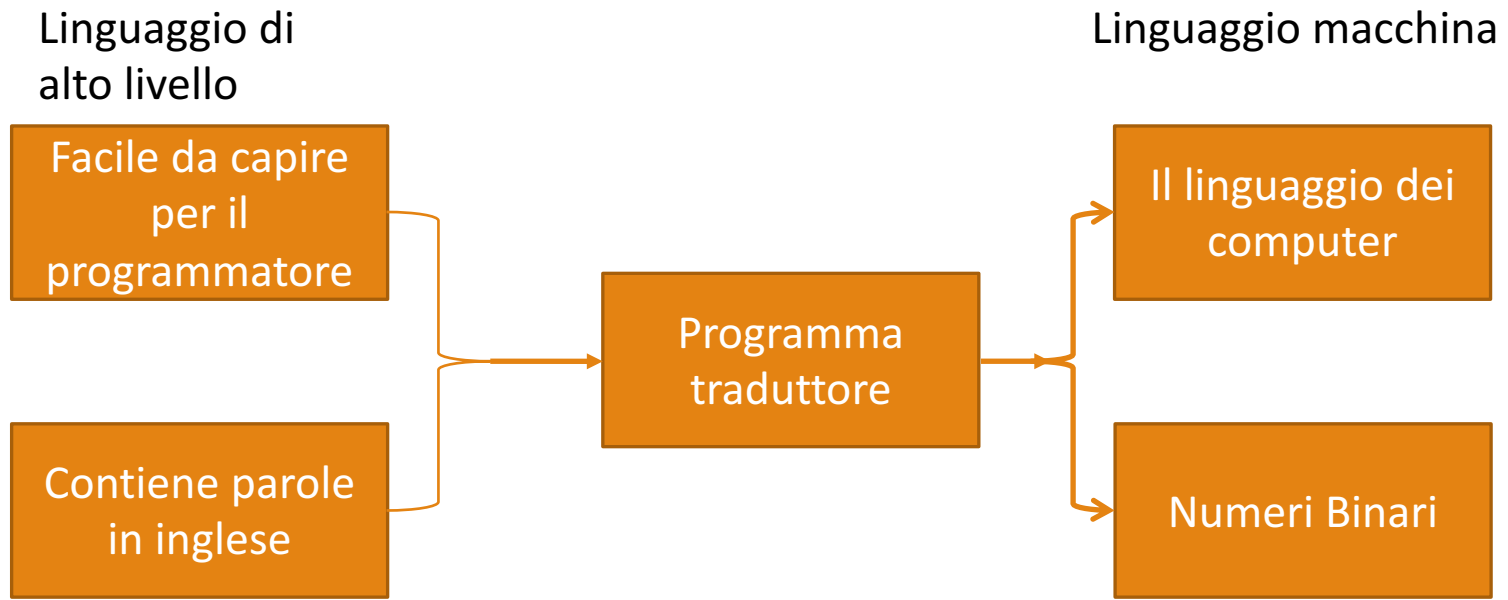
- Le macchine comprendono ed eseguono soltanto codici macchina
- Il programma prima di essere eseguito deve essere tradotto in linguaggio macchina
  - **Traduttore**
- La traduzione viene effettuata da un **compilatore**, un **interprete**, o una combinazione di entrambi



# Traduttore

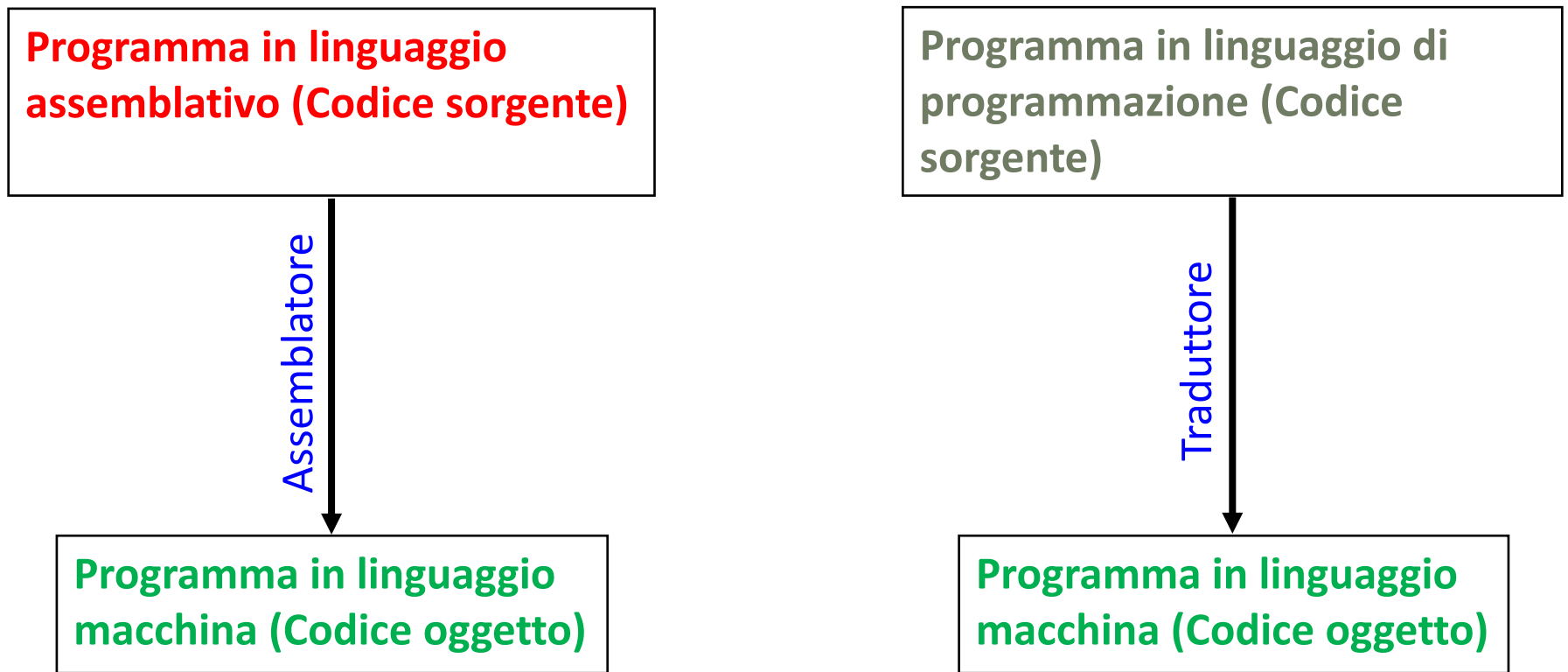
---

- Il traduttore è un programma che **converte** il codice di programmi scritti in un dato **linguaggio di programmazione (sorgenti)** nella corrispondente rappresentazione in **linguaggio macchina (eseguibili)**



# Assemblatore vs. Traduttore

---



# Tipi di Traduttore: Compilatore vs. Interprete – 1/2

---

- **Compilatore**

- Accetta in ingresso l'intero programma (istruzioni che lo compongono) e produce in uscita la rappresentazione dell'intero programma in linguaggio macchina

- **Interprete**

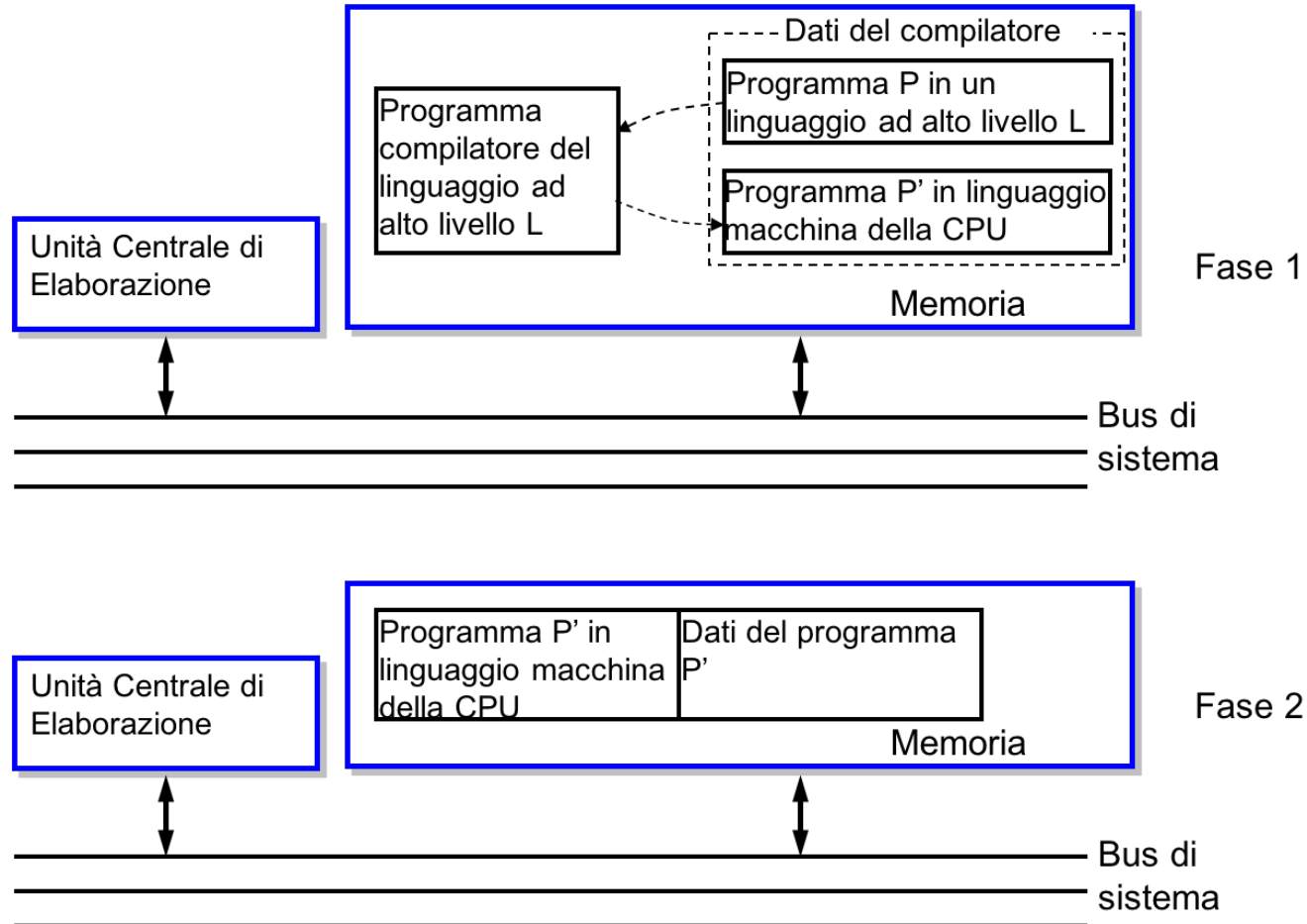
- Traduce ed esegue direttamente ciascuna istruzione del programma sorgente
  - Istruzione per istruzione
  - Una alla volta

# Tipi di Traduttore: Compilatore vs. Interprete – 2/2

---

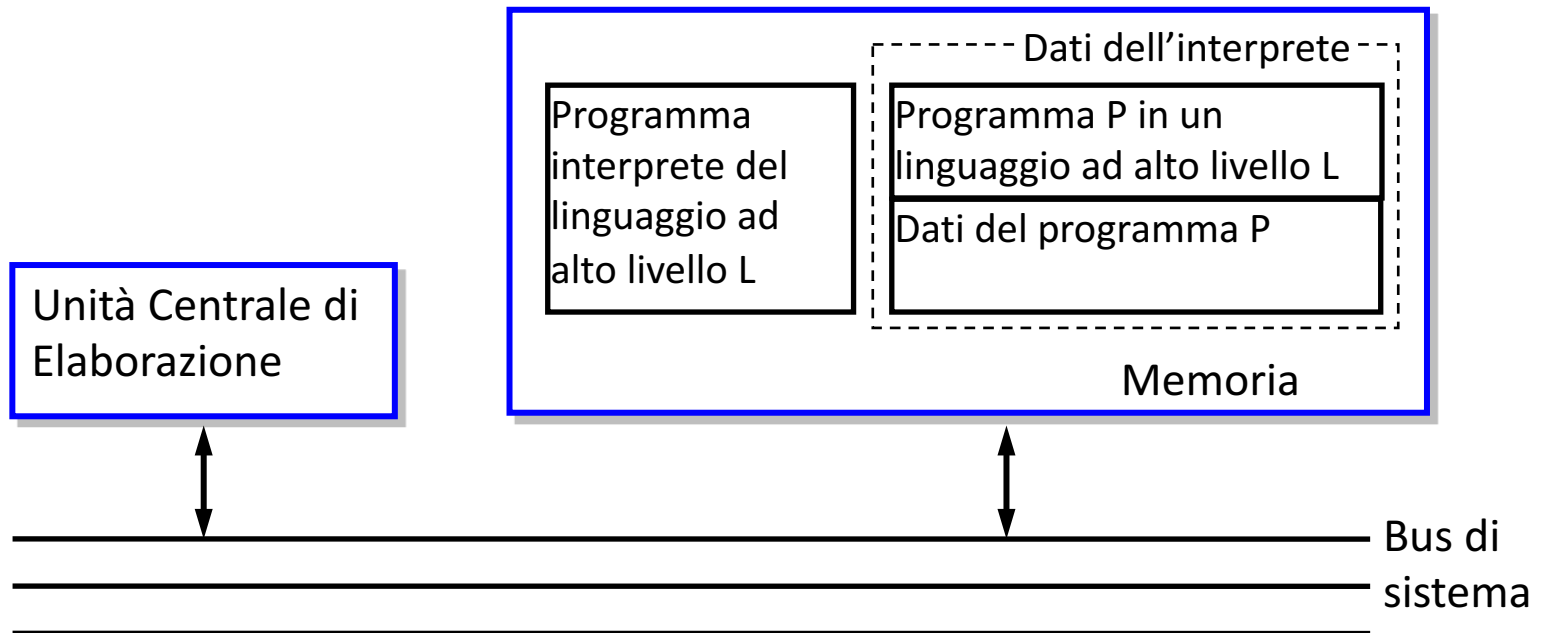
- Quale delle due soluzioni è la migliore?
  - **Compilazione**
    - **Pro:** applicazioni più veloci
    - **Contro:** maggior lavoro nel processo di messa a punto e manutenzione
  - **Interpretazione**
    - **Pro:** consente tempi di sviluppo più contenuti
    - **Contro:** produce programmi meno efficienti

# Compilatore



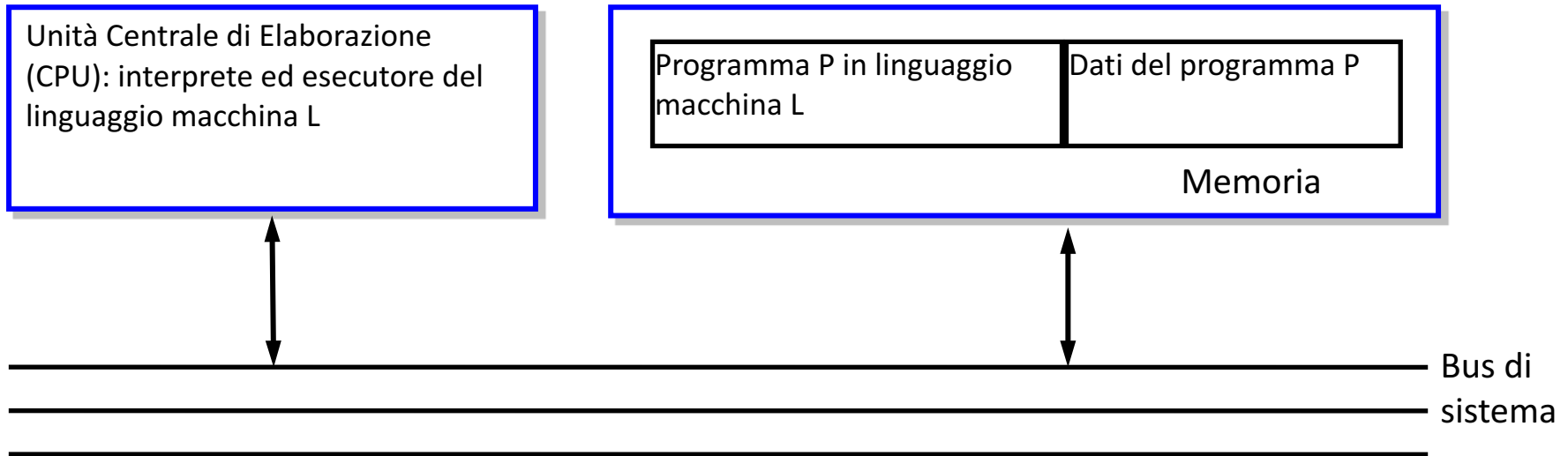
# Interprete

---



# CPU come Interprete del suo Linguaggio Macchina

---



# Paradigmi di Programmazione

---

- È possibile affrontare il problema della descrizione dei programmi in modi differenti
- **Definizione:** un paradigma di programmazione è un modello concettuale che fornisce la “struttura” di un programma
- Per paradigmi di programmazione si intendono i “modi” in cui vengono specificati i programmi
- Non si tratta del tipo di linguaggio usato, ma del contesto più ampio al quale un certo linguaggio appartiene
  - **Come viene organizzata la programmazione** e con **quali caratteristiche**
    - Stile, livello di dettaglio, “forma mentis” del programmatore, etc



# Riassumendo

---

- **Algoritmo:** descrizione di come si risolve un problema
- **Programma:** algoritmo scritto in modo che possa essere eseguito da un calcolatore (**linguaggio di programmazione**)
- **Linguaggio macchina:** linguaggio effettivamente “compreso” da un calcolatore, caratterizzato da
  - **Istruzioni primitive semplici** (ad es. max 2 operandi)
  - **Attenzione all’efficienza** (costi, complessità, velocità)
  - **Difficile e noioso da utilizzare per un programmatore**
- Due aspetti rilevanti
  - **Produrre algoritmi:** capire la sequenza di passi che portano alla soluzione di un problema
  - **Codificarli in programmi:** renderli comprensibili al calcolatore

# Riferimenti

---

- **Libro di testo**
  - Capitolo 4
    - Paragrafi 1, 2 e 3