CORSO DI
PROGRAMMAZIONE
E
INFORMATICA GENERALE 1

Lezione 1
(Concetti fondamentali)

E’ fondamentale capire la differenza tra…

- Specifica di un problema
- Specifica del processo di risoluzione
- Codifica del processo di risoluzione

Risoluzione di un problema

Con questo termine si indica il processo che:

- dato un *problema*
- individuato un opportuno *metodo risolutivo*

trasforma i dati iniziali nei corrispondenti risultati finali

E' indispensabile capire a quale "macchina" ci si riferisce"

COMPUTER

Computer

- E' uno strumento in grado di eseguire insiemi di *azioni* ("mosse") elementari
- Le azioni vengono eseguite su oggetti (***dati***) per produrre altri oggetti (***risultati***)
- L'esecuzione di azioni viene richiesta all'elaboratore attraverso "**frasi scritte in un qualche linguaggio**" (***istruzioni***)

Componenti di un Sistema di Elaborazione

- **Hardware**
- **Software**
  - Software di sistema: Sistema operativo
  - Software applicativo

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?**

Algoritmo

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

Programma

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

PROGRAMMAZIONE

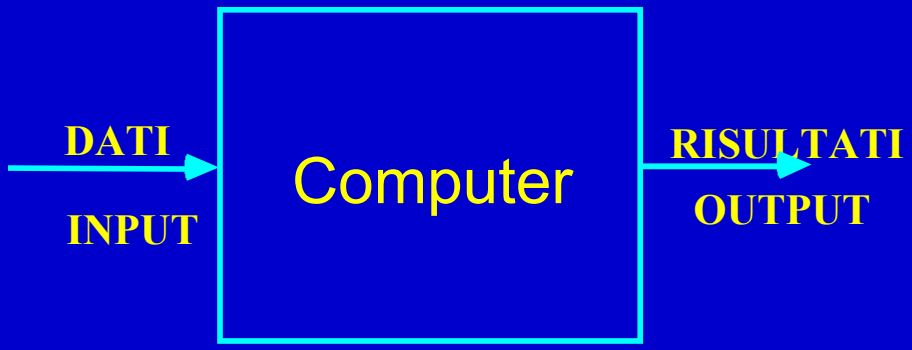
È l'attività con cui si predispone l'elaboratore ad eseguire un *particolare insieme di azioni* su *particolari dati*, allo scopo di *risolvere un problema*.

ALGORITMO e PROGRAMMA (a confronto)

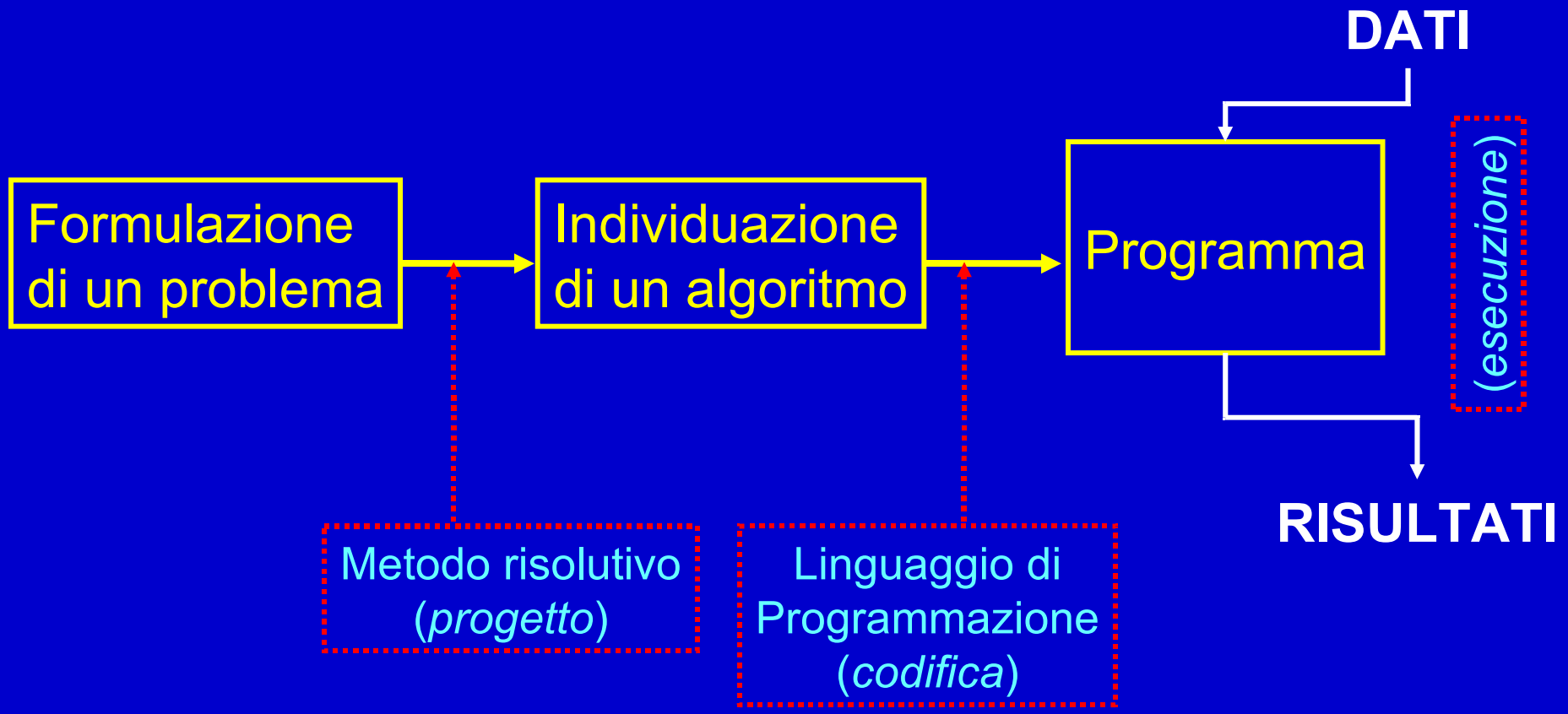
- Ogni elaboratore è una macchina in grado di eseguire *azioni* elementari su *dati*
- L'esecuzione delle azioni elementari è richiesta all'elaboratore tramite comandi chiamati *istruzioni*
- Le istruzioni sono espresse attraverso *frasi* di un opportuno *linguaggio di programmazione*
- **Un *programma*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



Riassumendo…



Esempio 1

Formulazione del problema

Stampare la somma di due numeri dati

Individuazione di un algoritmo

- Leggere il primo numero (p.es., da tastiera)
- Leggere il secondo numero (p.es., da tastiera)
- Effettuare la somma
- Stampare il risultato (p.es., su video)

Esempio 1 (*cont.*)

Codifica in un programma (*in linguaggio C++*)

```
main()
  {
    int A, B, ris;
    cout<<"Immettere due numeri: ";
    cin>>A;
    cin>>B;
    ris=A+B;
    cout<<"Somma:"<<ris<<endl;
  }
```

Esempio 1 (*cont.*)

E se l'avessimo scritto in linguaggio C?

```
main()
  {
    int A, B, ris;
    printf("Immettere due numeri: ");
    scanf("%d", &A);
    scanf("%d", &B);
    ris=A+B;
    printf("Somma: %d\n", ris);
  }
```

Caratteristiche di un algoritmo

- **Eseguibilità**: ogni azione deve essere *eseguibile* da parte dell'esecutore dell'algoritmo in un tempo finito

- **Non-ambiguità**: ogni azione deve essere *univocamente interpretabile* dall'esecutore

- **Terminazione**: il numero totale di azioni da eseguire, per ogni insieme di dati di ingresso, deve essere finito

Algoritmo (*cont.*)

Quindi, l'algoritmo deve:

- essere *applicabile a qualsiasi insieme di dati di ingresso* appartenenti al *dominio di definizione* dell'algoritmo
- essere costituito da operazioni appartenenti ad un determinato insieme di operazioni fondamentali
- essere costituito da regole non ambigue, cioè interpretabili in modo univoco qualunque sia l'esecutore (persona o "macchina") che le legge

ALTRE PROPRIETA'

- **Determinismo**
- **Efficienza**
- **Terminazione**

Algoritmi equivalenti

Due algoritmi si dicono **equivalenti** quando:

- hanno lo stesso dominio di ingresso
- hanno lo stesso dominio di uscita
- in corrispondenza degli stessi valori nel dominio di ingresso *producono gli stessi valori* nel dominio di uscita

Due algoritmi equivalenti:

- forniscono lo **stesso risultato**
- ma possono avere **diversa efficienza**
- e possono essere **profondamente diversi !**

Esempio 2

Calcolo del Massimo Comun Divisore (MCD) fra due interi M ed N

Algoritmo n° 1

- Calcola l'insieme A dei divisori di M
- Calcola l'insieme B dei divisori di N
- Calcola l'insieme C dei divisori comuni = A ∩ B
- Il risultato è il massimo dell'insieme C

Algoritmo n° 2 (metodo di Euclide)

$$\text{MCD}(M,N) = \begin{cases} M \text{ (oppure N)} & \text{se } M=N \\ \text{MCD}(M-N,\ N) & \text{se } M>N \\ \text{MCD}(M,\ N-M) & \text{se } M<N \end{cases}$$

Algoritmo:

- Finché $M \neq N$:
- se $M>N$, sostituisci a M il valore $M' = M-N$
- altrimenti sostituisci a N il valore $N' = N-M$
- Il Massimo Comun Divisore è il valore finale ottenuto quando M e N diventano uguali

Esempio 2 (*cont.*)

Calcolare il MCD dei numeri 12 e 21

Algoritmo 1: Divisori di 12 = 1, 2, 3, 4, 6, 12
Divisori di 21 = 1, 3, 7, 21
Intersezione insieme dei divisori: 1, 3
Massimo dell'insieme dei divisori: **MCD = 3**

Algoritmo 2: M ← 12, N ← 21
N>M, quindi: N ← (N-M) = 9 ➔ M=12, N=9
M>N, quindi: M ← (M-N) = 3 ➔ M=3, N=9
N>M, quindi: N ← (N-M) = 6 ➔ M=3, N=6
N>M, quindi: N ← (N-M) = 3 ➔ M=3, N=3
N=M, quindi: **MCD = 3**

Livello di Astrazione per la Codifica di un Algoritmo

- Si può risolvere un problema **senza prima fissare un insieme di "azioni"**, di "mosse elementari" possibili per l'elaboratore?
  - No, bisogna prima scendere al livello della macchina, per conoscerne le caratteristiche
  - Ma **quale macchina**?

- **È corretto** impostare la soluzione a partire da tali "mosse elementari"?
  - SI, per risolvere il problema *con efficienza*
  - NO, se la macchina di partenza ha mosse di livello *troppo basso*

Livello di Astrazione per la Codifica di un Algoritmo (*cont.*)

- **Bisogna risalire ad un più adeguato *livello di ASTRAZIONE*** (processo di **aggregazione** di informazioni e dati e di **sintesi** di modelli concettuali che ne enucleano le proprietà **rilevanti** escludendo i dettagli inessenziali)

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**

Quale **linguaggio**? Quale ambiente di programmazione?

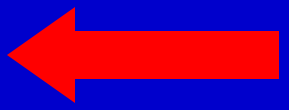
Linguaggi di programmazione

- **Un linguaggio di programmazione è una notazione formale per descrivere algoritmi**

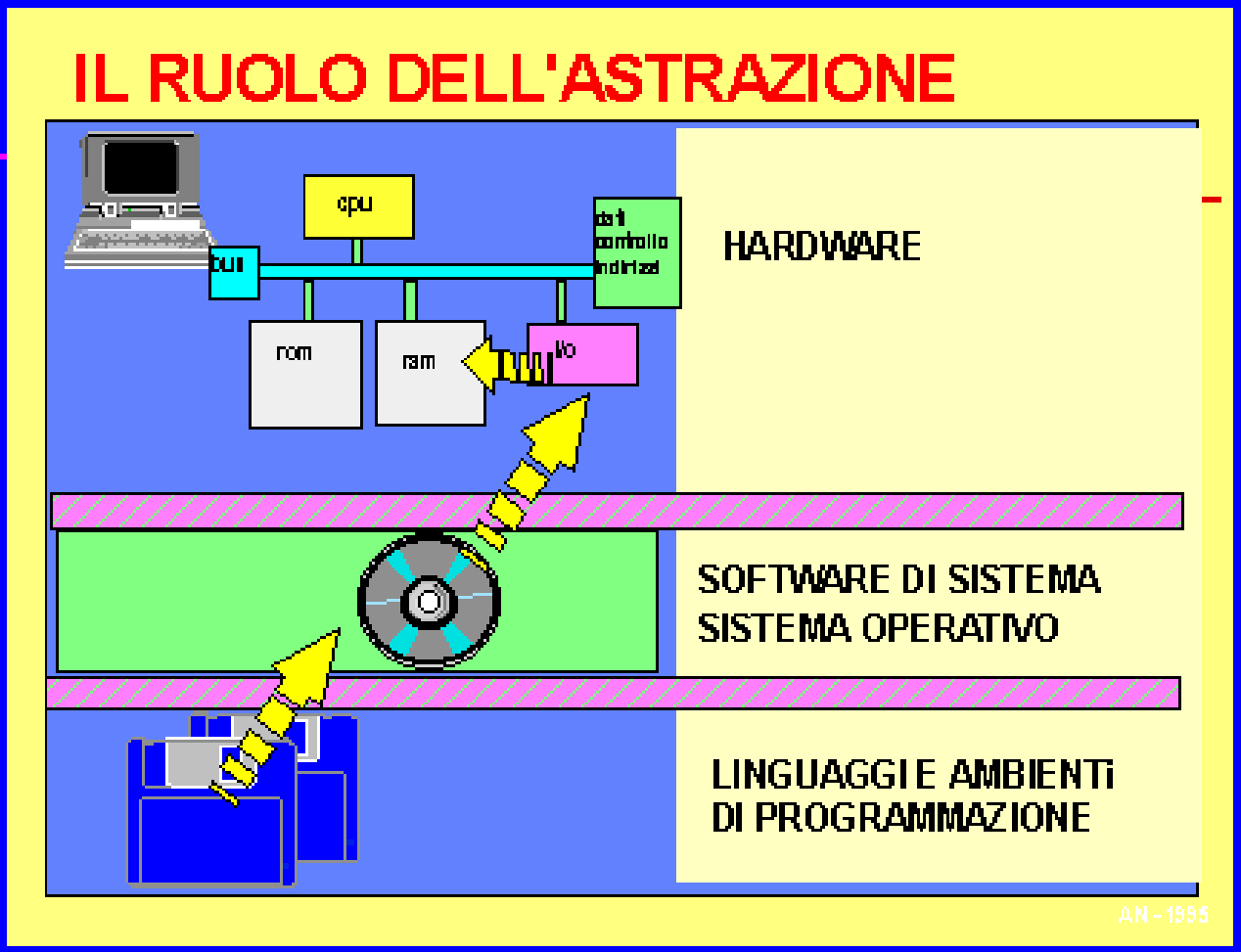
- **Un programma è la "codifica" o "implementazione" di un algoritmo in un linguaggio di programmazione**

- **Un programma si compone di più istruzioni scritte nel linguaggio di programmazione.**

  - Quali ***parole chiave*** vi sono in un linguaggio?
  - Quali sono i **meccanismi di combinazione** delle parole chiave?
  - E soprattutto, a quale ***macchina*** si riferisce?

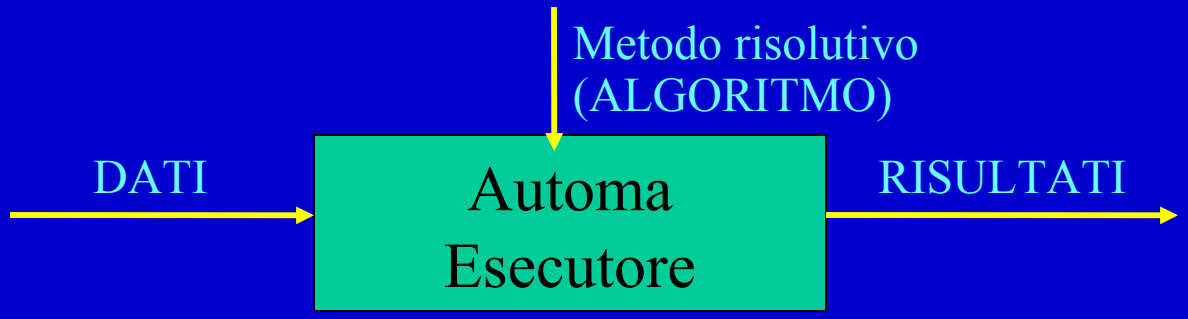


Astrazione: come si vede la “macchina”



Automa esecutore

- Un automa capace di ricevere dall'esterno una descrizione dell'algoritmo richiesto, cioè capace di interpretare un linguaggio (*linguaggio macchina*)



Come realizzare l'automa?

- Mediante **congegni meccanici**
  - macchina aritmetica (1649) di Blaise Pascal
  - macchina analitica di Charles Babbage (1792-1871)
- Mediante **un modello matematico**
  - funzionale (Hilbert, (1842-1943), Church, Kleene)
  - operazionale (Turing, 1912-1954)
  - sistemi di riscrittura (Post, Markov,...)

Linguaggio macchina

- E' il linguaggio di un computer
- Il linguaggio macchina è direttamente eseguibile dall'elaboratore, senza alcuna intermediazione
- Computer con architetture interne (CPU) differenti hanno linguaggi macchina differenti
- Pertanto, un programma scritto nel linguaggio macchina di un computer non è eseguibile su di un computer con un'architettura differente

  (Si dice che non è portabile!)

Linguaggi di alto livello

- Si basano su una "macchina" le cui "mosse" non sono quelle della macchina hardware
  → realizzano una "***macchina virtuale***"
- Supportano concetti ed astrazioni
- Promuovono metodologie per agevolare lo sviluppo del software da parte del programmatore
- Hanno capacità espressive molto superiori rispetto a quelle del linguaggio macchina

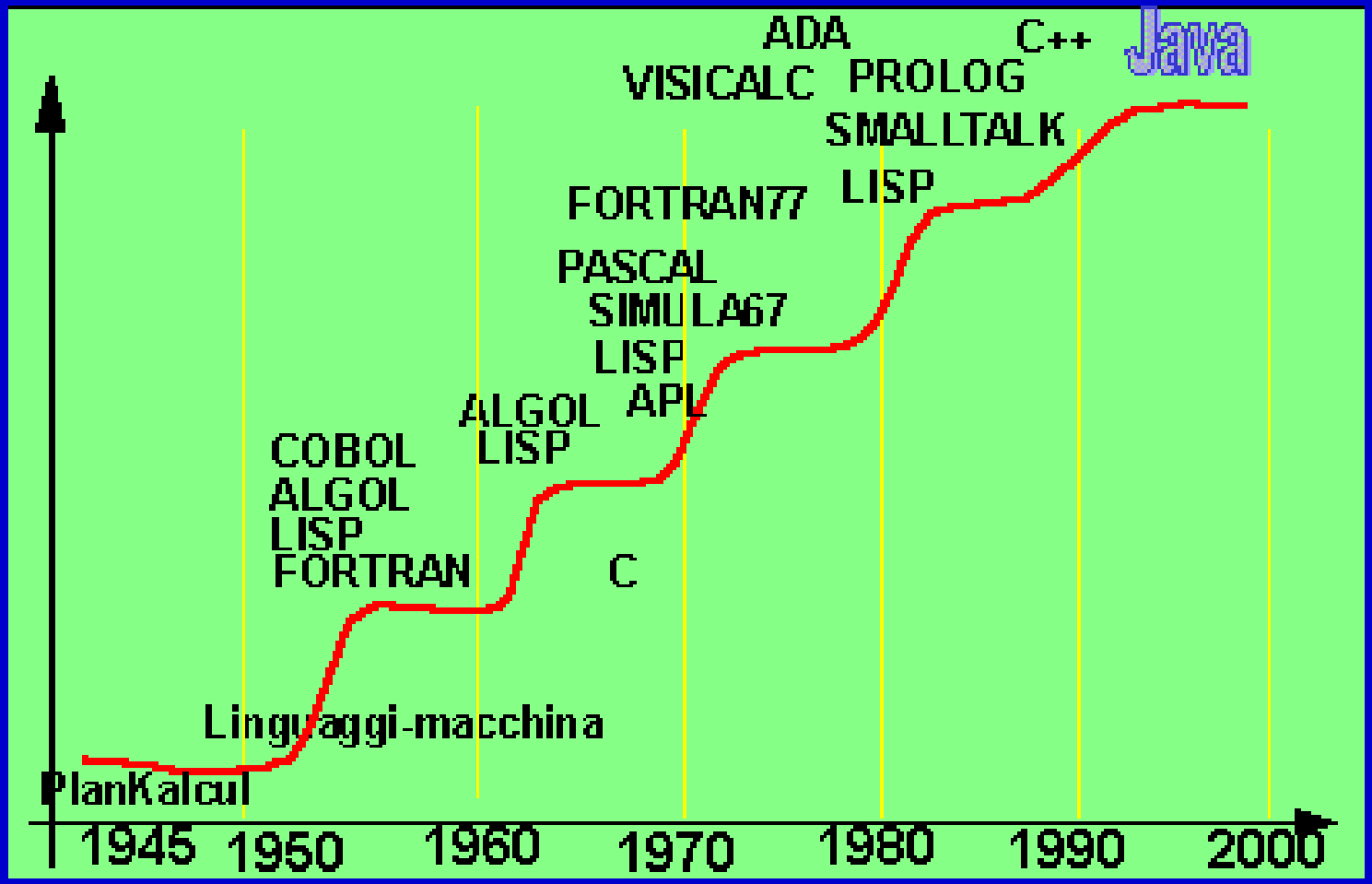
- **Esistono centinaia di linguaggi di programmazione**! (anche se pochi sono in uso)

Linguaggi di alto livello (*cont.*)

Categorie di linguaggi ad alto livello

Ogni categoria determina uno specifico stile di programmazione!

Evoluzione dei linguaggi



Perché esistono tanti linguaggi?

- **Contesto applicativo:**
  - Scientifico: **Fortran**
  - Gestionale: **Cobol**
  - Sistemi Operativi: **C**
  - Applicazioni (non di rete): **C++**
  - Applicazioni di rete: **Java**

Ambiente di programmazione

- È l'insieme degli strumenti (*tool*) che consentono la codifica, la verifica e l'esecuzione di nuovi programmi (*fasi di sviluppo*)

Sviluppo di un Programma

- Affinché un programma scritto in un qualsiasi linguaggio di programmazione ad alto livello sia comprensibile (e quindi eseguibile) da un calcolatore, ***occorre tradurlo*** **dal linguaggio di programmazione originario al linguaggio comprensibile al calcolatore** (**linguaggio macchina**)
- Questa operazione viene normalmente svolta da speciali programmi, detti **traduttori**

Processo di traduzione

Programma

```
main()
{ int A;
  ...
  A=A+1;
  if ...
```

Traduzione

```
00100101
   ...
11001..
1011100 ...
```

I traduttori convertono il testo dei programmi scritti in un particolare linguaggio di programmazione (***programmi sorgenti***) nella corrispondente rappresentazione in linguaggio macchina (***programmi eseguibili***)

Tipi di Traduttori

Due categorie di traduttori:

- ***Compilatori*** traducono l'intero programma (senza eseguirlo!) e producono in uscita il programma convertito in linguaggio macchina
- ***Interpreti*** traducono ed eseguono immediatamente ogni singola istruzione del *programma sorgente*

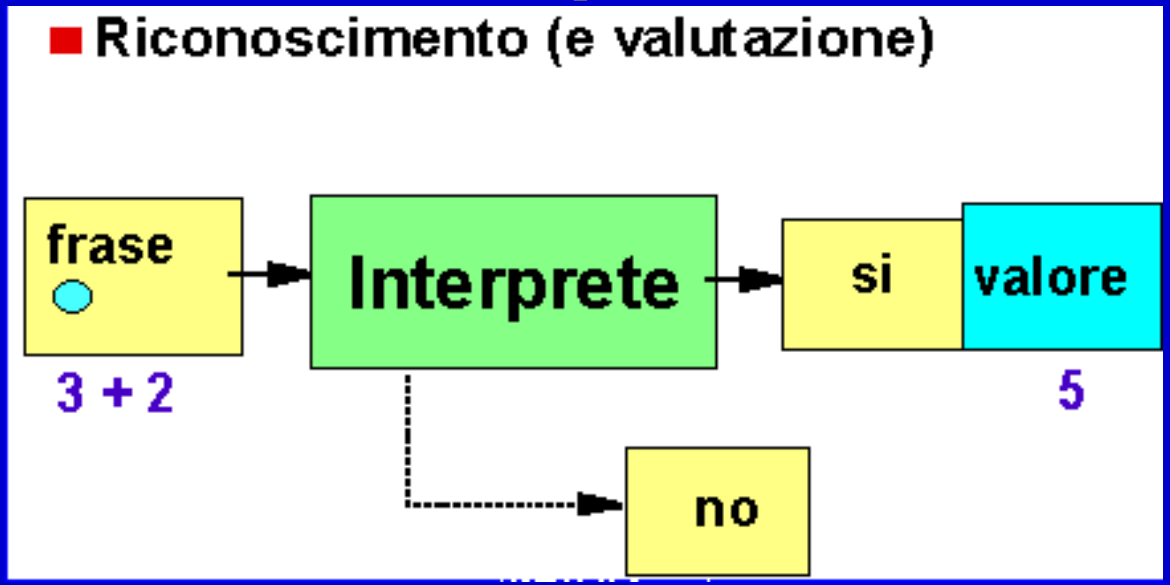
Quindi,

- **Nel caso del compilatore**, lo schema traduzione-esecuzione viene percorso una volta sola prima dell'esecuzione
- **Nel caso dell'interprete**, lo schema traduzione-esecuzione viene attraversato tante volte quante sono le istruzioni che compongono il programma

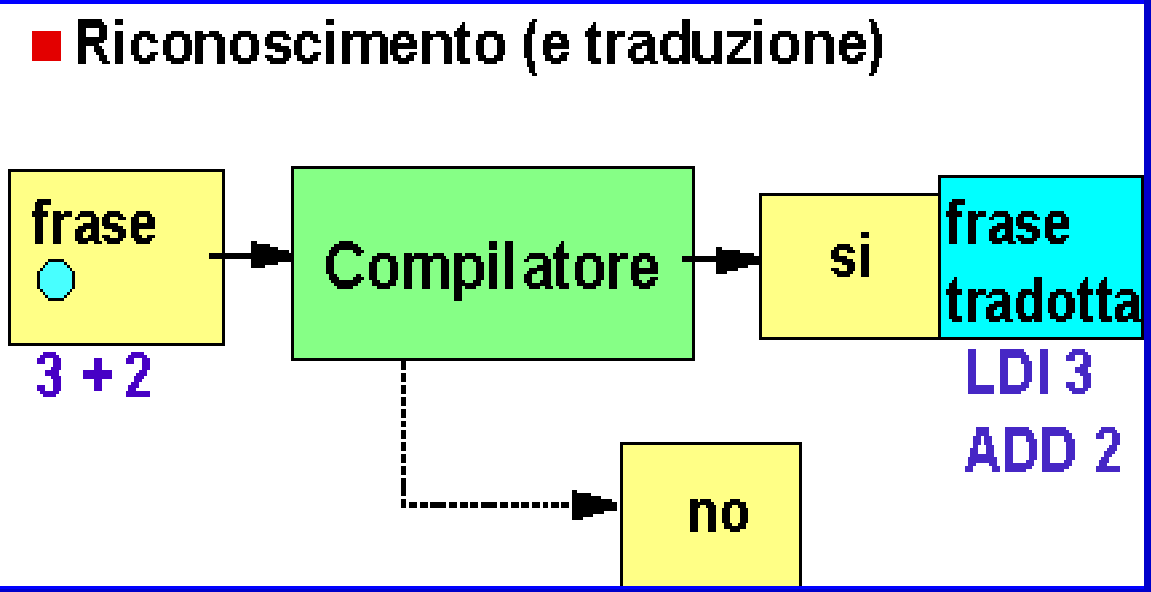
  Ad ogni attivazione dell'interprete su di una particolare istruzione segue l'esecuzione dell'istruzione stessa.

Compilatore e Interprete

- Riconoscimento (e valutazione)



- Riconoscimento (e traduzione)



Compilatore e Interprete (*cont.*)

- Sebbene in linea di principio un qualsiasi linguaggio possa essere tradotto sia mediante **compilatori** sia mediante **interpreti**, nella pratica si tende verso una differenziazione già a livello di linguaggio:
  - Tipici linguaggi interpretati: **Basic**, **Javascript, Perl**, ...
  - Tipici linguaggi compilati: **C/C++**, **Fortran**, **Pascal**, **ADA**, …

    (*NOTA:* **Java** *costituisce un caso particolare*, anche se si tende a considerarlo interpretato)

Compilatore e Interprete (*cont.*)

- L'esecuzione di un programma *compilato* è più veloce dell'esecuzione di un programma *interpretato*

- Un programma interpretato è più **portabile** di un programma compilato
  (***portabile*** = può essere eseguito su piattaforme diverse)

Ambiente di programmazione

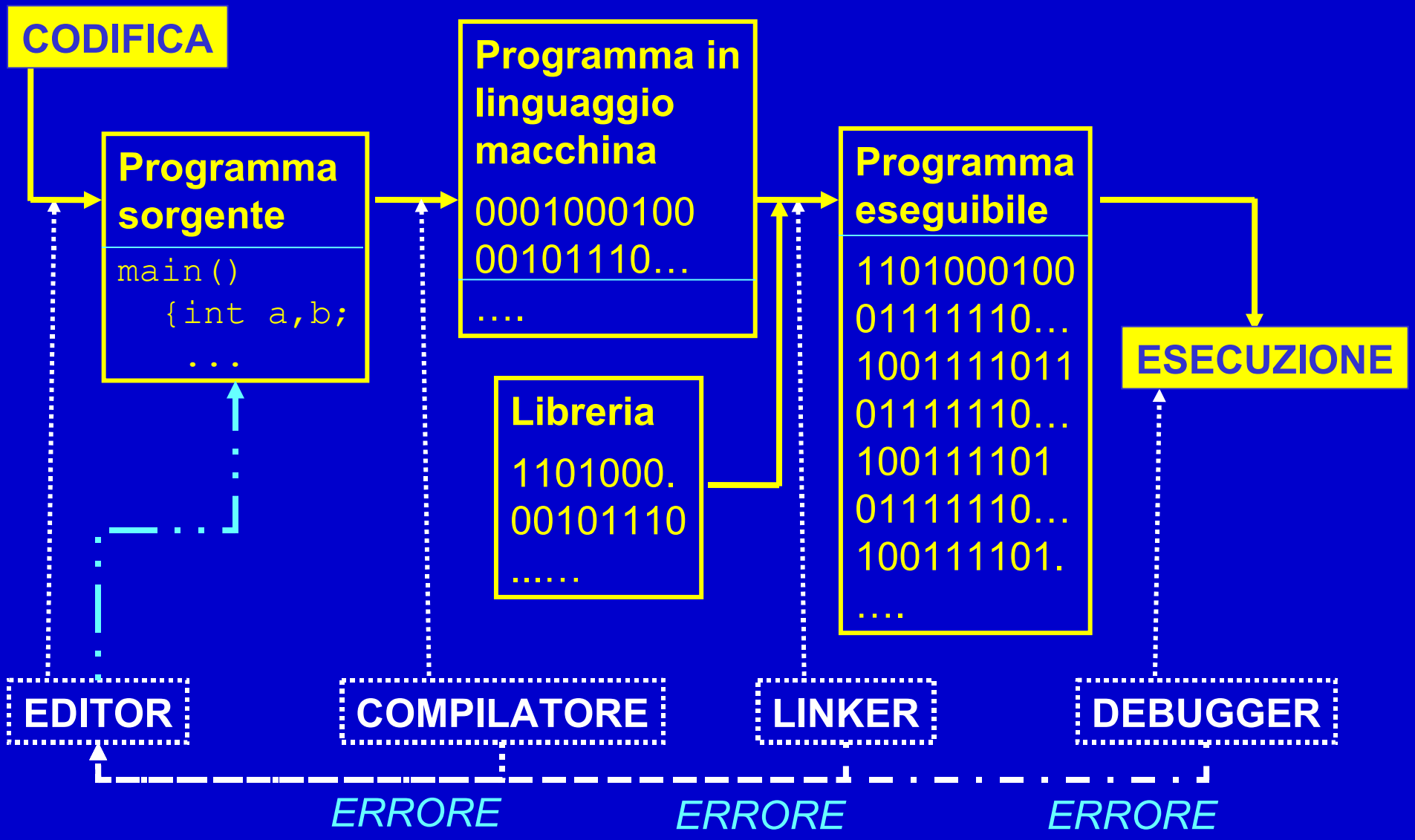
- ***Editor*:** serve per creare file che contengono testi (cioè sequenze di caratteri). In particolare, l'editor consente di scrivere il *programma sorgente*.
- ***Compilatore*:** opera la traduzione di un programma sorgente scritto in un linguaggio ad alto livello in un *programma oggetto*.
- ***Linker*** ("*collegatore*"): nel caso in cui la costruzione del programma oggetto richieda l'unione di più moduli (compilati separatamente), provvede a collegarli per formare un unico *programma eseguibile.*

- ***Debugger*** (*"spulciatore"*): consente di eseguire passo-passo un programma, controllando via via quel che succede, al fine di scoprire ed eliminare errori non rilevati in fase di compilazione.

- ***Interprete*:** traduce ed esegue direttamente ciascuna istruzione del *programma sorgente*, istruzione per istruzione.

  È alternativo al compilatore.

Sviluppo ed esecuzione di un programma



Parole chiave della lezione

- **Computer: Hardware, Sistema operativo, Software**
- **Algoritmo**
- **Programma**
- **Codifica, Implementazione, Programmazione**
- **Equivalenza di algoritmi**
- **Efficienza di algoritmi**
- **Linguaggi di programmazione**
  - Linguaggio macchina
  - Linguaggi ad alto livello
- **Traduttori**
  - Compilatore
  - Interprete