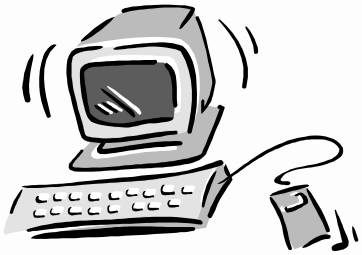
Lezione I
Lu 1 Ott. 2007

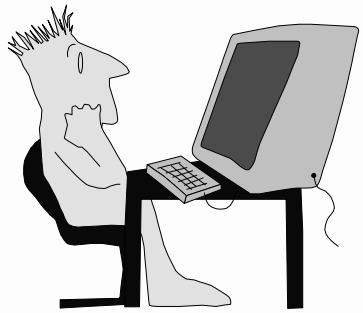
Introduzione

Cos'è un computer?



- Oggi si usano quotidianamente computer
  - lavoro
  - svago
- Nel lavoro, i computer sono ottimi per svolgere
  - operazioni ripetitive o noiose, come effettuare calcoli o impaginare testi
  - operazioni complicate e veloci, come controllare macchine utensili
- Nel gioco, i computer sono ottimi per coinvolgere al massimo l'utente-giocatore, perché possono riprodurre con estremo realismo suoni e sequenze di immagini
- In realtà, tutto questo non è merito solo del computer, ma anche dei ***programmi*** che vengono eseguiti

Cos'è un computer?

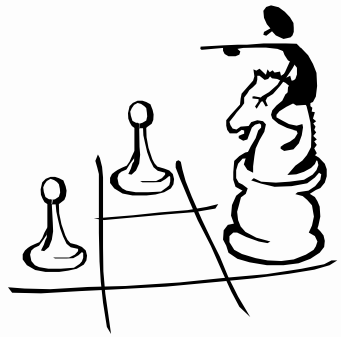


- Il computer è una macchina ***versatile,*** progettata per ***eseguire programmi;*** Ciascuno programma è sviluppato per svolgere una specifica attività.

- Differisce dalle altre macchine che generalmente sono progettate per svolgere un'unica attività:
  - auto ⇒ trasporto
  - telefono ⇒ comunicazione orale
  - trapano ⇒ foratura materiali solidi

- L'***elevato numero*** di istruzioni presenti in un programma e la loro esecuzione ad ***alta velocità –*** se paragonata alla velocità di reazione umana **-** garantisce all'utente l'impressione di un'interazione fluida.

Cos'è un computer?



- Un computer è in generale una macchina che
  - ***elabora dati*** (numeri, parole, immagini, suoni...)
  - ***interagisce con dispositivi*** (schermo, tastiera, mouse...)
  - ***esegue programmi***

- Ogni programma svolge una diversa funzione, anche complessa
  - impaginare testi
  - giocare a scacchi

- I programmi sono composti da
  - ***sequenze di istruzioni che il computer esegue***
  - ***decisioni che il computer prende*** per svolgere una certa attività

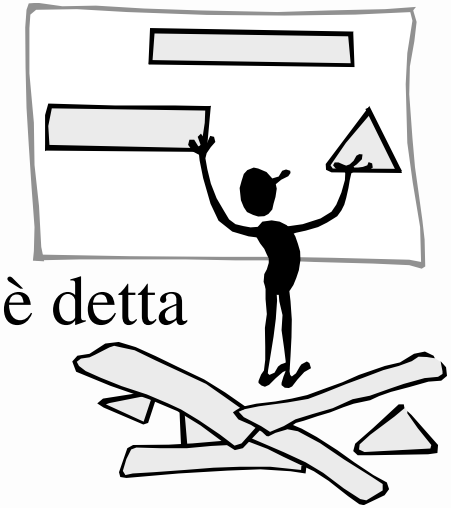
Cos'è un programma?

- Nonostante i programmi siano generalmente sofisticati e svolgano funzioni complesse, le ***istruzioni*** di cui sono composti sono ***elementari*****,** ad esempio

  - estrarre un numero da una posizione della memoria
  - sommare due numeri
  - inviare la lettera **A** alla stampante (**istruzione**)
  - disegnare un punto rosso in una data posizione dello schermo
  - se un dato è negativo, proseguire il programma da una certa istruzione anziché dalla successiva (***decisione***)

Cos'è la programmazione?

- Un programma descrive al computer, in estremo dettaglio, la sequenza di passi necessari per svolgere un particolare compito



- L'attività di ***progettare e realizzare un programma*** è detta ***programmazione***

- **Scopo di questo corso è fornire la competenza per scrivere semplici programmi usando il linguaggio di programmazione *Java***

Cos'è la programmazione?

- ***Usare*** un computer ***non*** richiede alcuna attività di programmazione. Si usano programmi codificati da altri
  - giochi
  - navigazione in web (web browser), posta elettronica, instant messaging
  - videoscrittura, publishing, grafica, musica, cinema

- Al contrario, un ***informatico professionista*** solitamente svolge attività di programmazione, anche se la programmazione non è l'unica competenza che deve avere

- La programmazione è una parte importante dell'informatica

Cos'è un algoritmo?

- Con il computer si elaborano dati e risolvono problemi.

- Quale tipo di problemi è possibile risolvere con un computer?
  - Dato un insieme di fotografie di paesaggi, qual è il paesaggio ***più rilassante***?
  - Avendo depositato ventimila euro in un conto bancario che produce il 5% di interessi all'anno, capitalizzati annualmente, quanti anni occorrono affinché il saldo del conto arrivi al doppio della cifra iniziale?

- Il primo problema non può essere risolto dal computer. **Perché?**

Cos'è un algoritmo?

- Il primo **problema** è **ambiguo**. Non può essere risolto dal computer, a meno che non sia data una ***definizione*** di **paesaggio rilassante** che possa essere usata per confrontare ***in modo univoco*** due paesaggi

- Un computer può risolvere soltanto problemi che potrebbero essere risolti anche manualmente
  - **è solo più veloce, non fa errori, non si annoia**
- Il secondo problema è certamente risolvibile manualmente, facendo un po' di calcoli...

Cos'è un algoritmo? Al-Khuwarizmi IX sec. d.c.

- Si dice ***algoritmo*** la ***descrizione*** di un metodo di soluzione di un problema che
  - **sia eseguibile (sequenza di passi eseguibili)**
  - **sia priva di ambiguità**
  - **arrivi a una conclusione con un numero finito di passi**
- Un computer può risolvere soltanto quei problemi per i quali sia noto un algoritmo

Un esempio di algoritmo

- **Problema**: Avendo depositato ventimila euro in un conto bancario che produce il 5% di interessi all'anno, versati annualmente, quanti anni occorrono affinché il saldo del conto arrivi al doppio della cifra iniziale?

anni	*saldo*
0	20.000,00
1	21.000,00
2	22.050,00
3	23.152,50
4	24.310,13
5	25.525,63
6	26.801,91
7	28.142,01
8	29.549,11
9	31.026,56
10	32.577,89
11	34.206,79
12	35.917,13
13	37.712,98
14	39.598,63
15	41.578,56

Un esempio di algoritmo

A. L'anno attuale è 0 e il saldo attuale è 20000 €

B. Ripetere i successivi passi **C** e **D** finché il saldo è minore di 40000 €, altrimenti passare al punto **E**

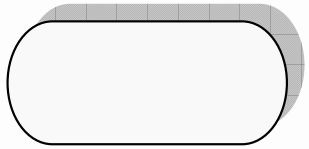
C. Aggiungere 1 al valore dell'anno attuale

D. Calcolare il nuovo saldo che è pari al valore del saldo attuale moltiplicato per 1.05 (cioè aggiungiamo il 5%)

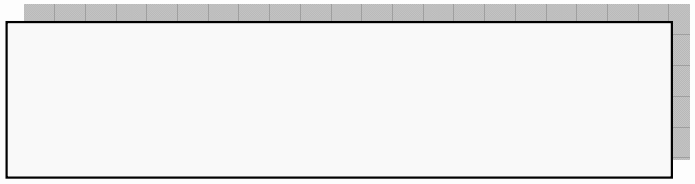
E. Il risultato è il valore dell'anno attuale

Diagramma di flusso (flow chart)

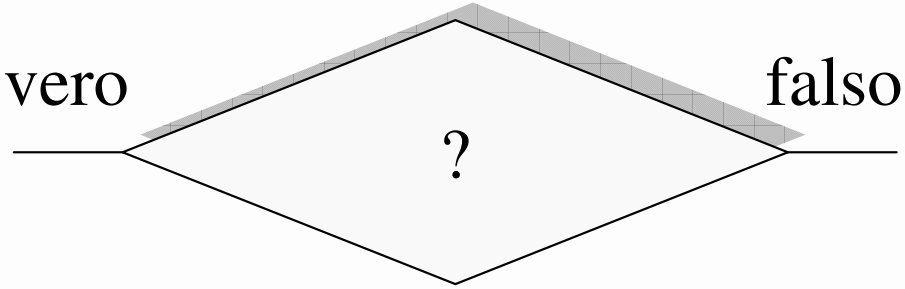
- Semplici algoritmi sono spesso rappresentati graficamente per mezzo di ***diagrammi di flusso***.
- Simboli usati:



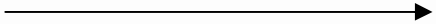
inizio/fine dell'algoritmo



passo

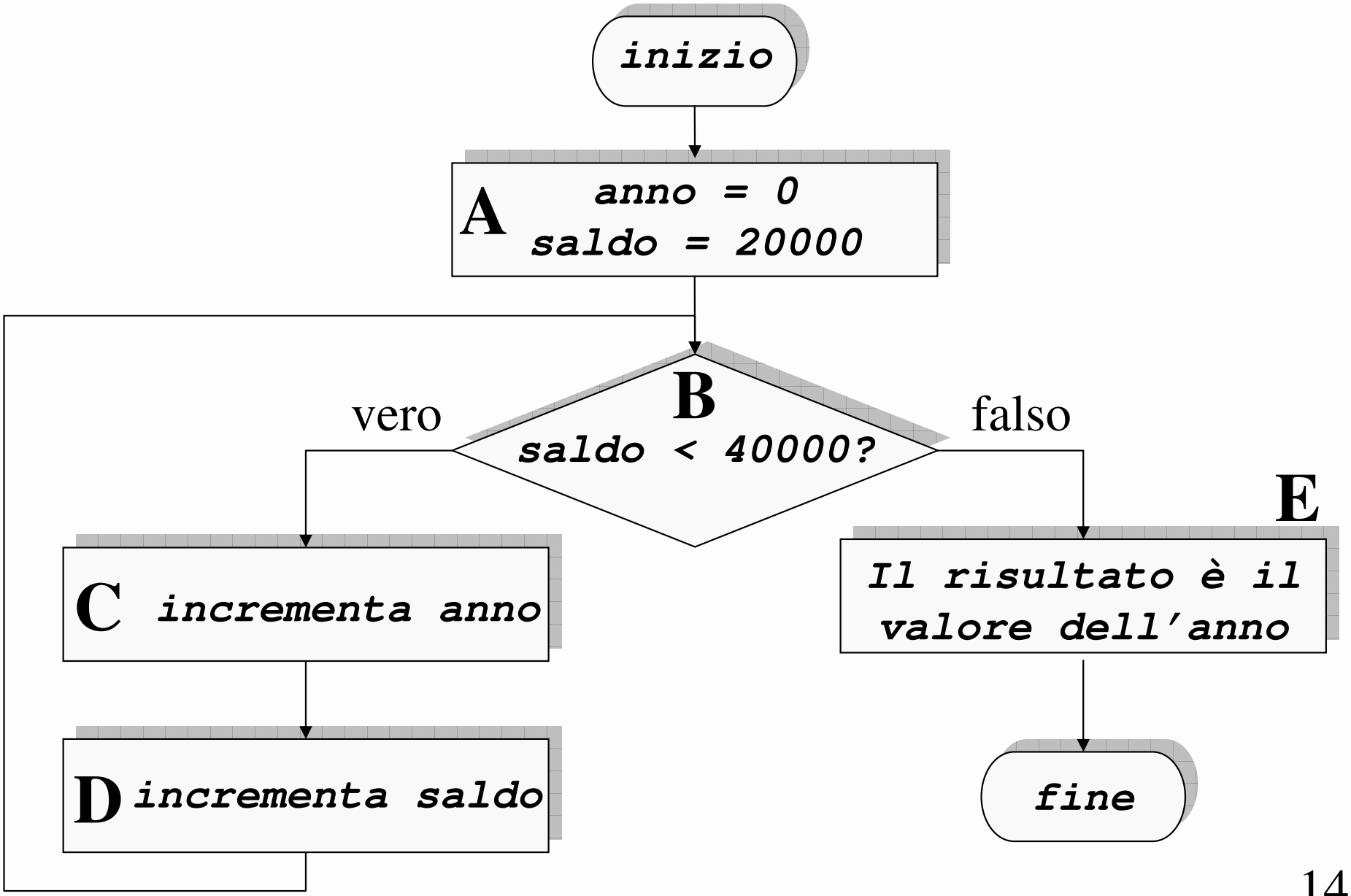


decisione



arco di flusso del controllo

Diagramma di flusso



Un esempio di algoritmo

- Il metodo di soluzione proposto
  - **non è ambiguo**, perché fornisce precise istruzioni su cosa bisogna fare a ogni passaggio e su quale deve essere il passaggio successivo
  - **è eseguibile**, perché ciascun passaggio può essere eseguito concretamente (se, ad esempio, il metodo di soluzione dicesse che il tasso di interesse da usare al punto **4** è variabile in dipendenza da fattori economici futuri, il metodo non sarebbe eseguibile…)
  - **arriva a conclusione in un tempo finito**, infatti ad ogni passo il saldo aumenta di almeno mille euro quindi, in meno di 20 passi arriva al termine

Altro esempio di algoritmo

- **Problema**: determinare i numeri interi primi ≤ **n,** | n>2
- **Algoritmo**: **CRIVELLO DI ERATOSTENE (~276-196 a.C.)**

A Scrivere in sequenza i numeri da 2 a n in ordine crescente e porre la variabile k = 2

B Finché k*k è inferiore o uguale a n, eseguire i seguenti passi C e D, altrimenti eseguire il passo E

C Cancellare dall'elenco tutti i multipli di k (2k, 3k, ...)

D Scegliere dall'elenco il successivo a k fra i numeri non cancellati e assegnarlo a k

E Fine dell'algoritmo: i numeri primi sono quelli che non sono stati cancellati

Esempio: sia n = 11

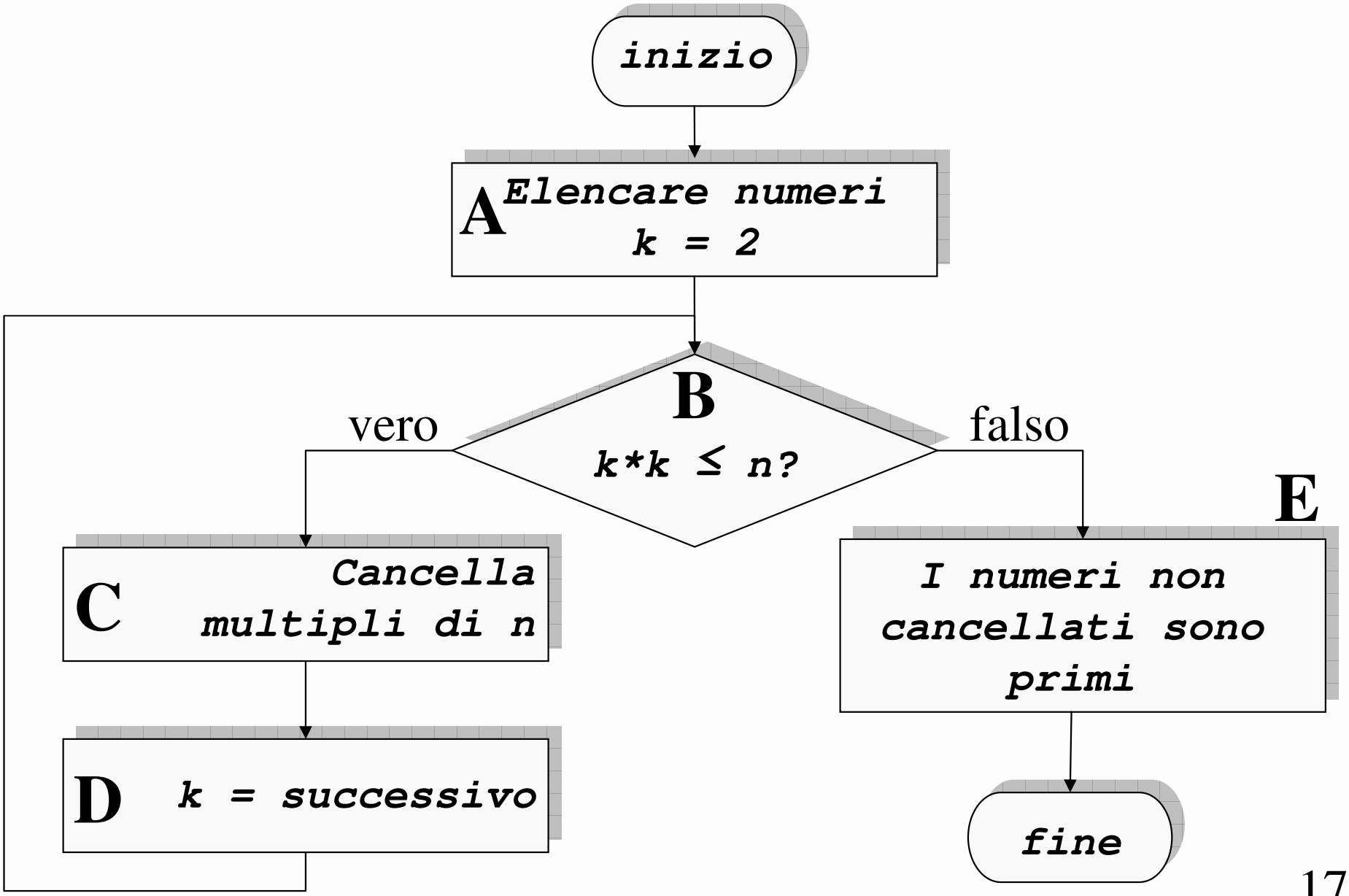
A: **2** 3 4 5 6 7 8 9 10 11, k = **2**

B: k*k = 4 < n **C:** 2 **3** ~~4~~ 5 ~~6~~ 7 ~~8~~ 9 ~~10~~ 11 **D:** k = **3**

B: k*k = 9 < n **C:** 2 3 ~~4~~ **5** ~~6~~ 7 ~~8~~ ~~9~~ ~~10~~ 11 **D:** k = **5**

B: k*k = 25 > n **E:** 2 3 5 7 11

Diagramma di flusso



A cosa servono gli algoritmi?

- L'identificazione di un algoritmo è un requisito indispensabile per risolvere un problema (con il computer o senza)

- La scrittura di un programma per risolvere un problema consiste, in genere, nella traduzione di un algoritmo in un qualche ***linguaggio di programmazione***

- **Prima di scrivere un programma, è necessario individuare e descrivere un algoritmo!**

- La definizione di algoritmi e la misura della loro efficienza è una parte importante dell'informatica (e anche di questo corso).

Lezione II
Ma 2 Ott. 2007

Architettura di un eleboratore

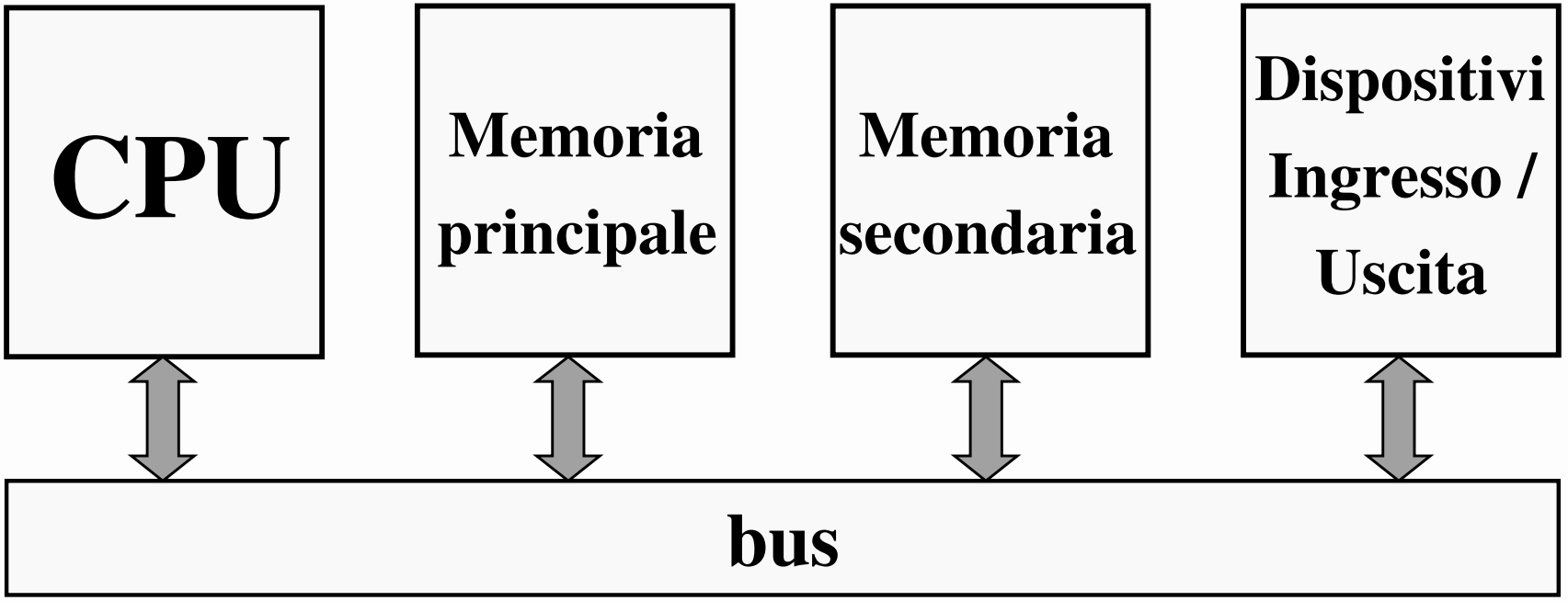
Il modello di von Neumann

Il modello di von Neumann

- Nel 1945-46 János **von Neumann** elaborò un modello teorico dell'architettura di un elaboratore che è tuttora valido e molto utilizzato
  - (EDVAC, *Electronic Discrete Variable Automatic Computer*)
- La grande maggioranza degli elaboratori odierni ha un'architettura che può essere ricondotta al modello di von Neumann (più o meno facilmente)
  - le eccezioni più importanti sono alcune macchine ad elaborazione parallela
- Il modello è importante in quanto schematizza in modo ***omogeneo*** situazioni diverse
  - lo presentiamo in una versione un po' modificata

Il modello di von Neumann

- L'architettura di von Neumann è composta da **quattro blocchi** comunicanti tra loro per mezzo di un canale di scambio di informazioni, detto **bus**,



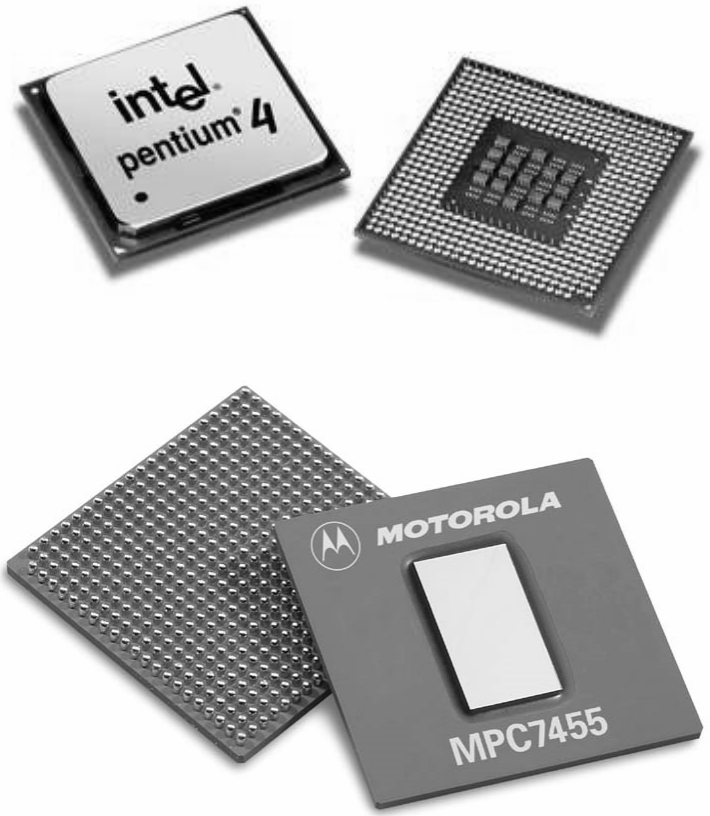
Unità di elaborazione centrale

- L'unità centrale di elaborazione

  CPU, ***C**entral **P**rocessing **U**nit*

  ha il compito di elaborare i dati
  - individua ed esegue le istruzioni del programma
  - effettua elaborazioni aritmetiche e logiche con la sua unità logico-aritmetica
  - acquisisce i dati dalla memoria primaria e secondaria e da altri dispositivi di input/output e ve li rispedisce dopo averli elaborati
- Fisicamente è costituita da uno o più ***chip*** (microprocessori)

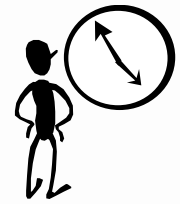
Il chip della CPU



- Un chip, o ***circuito integrato***, è un componente elettronico con connettori metallici esterni (*pin*) e collegamenti interni (*wire*), costituito principalmente di silicio e alloggiato in un contenitore plastico o ceramico (*package*)
- I collegamenti interni di un chip sono molto complicati; ad esempio, il chip Pentium 4 di Intel è costituito da più di ***50 milioni di transistori*** tra loro interconnessi

L'unità centrale di elaborazione

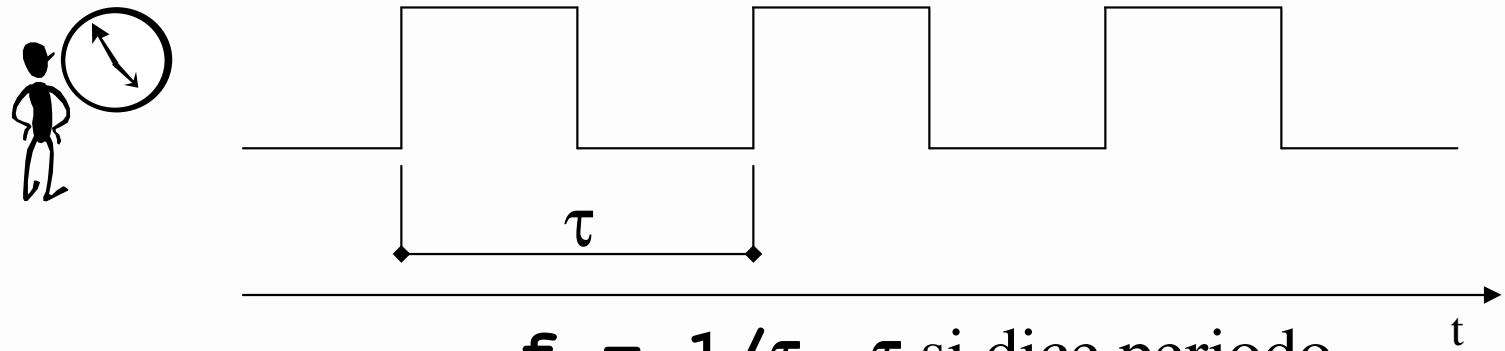
- Dal punto di vista logico, la CPU è costituita da tre parti principali:
  - l'unità **logico-aritmetica** (ALU)
  - l'unità di **controllo**, che ne governa il funzionamento
  - un insieme limitato di ***registri***, che sono locazioni di memoria ad accesso veloce per la memorizzazione temporanea dei dati



- Il funzionamento della CPU è di tipo ***ciclico*** e il periodo di tale ciclo viene scandito dall'orologio di sistema (***clock***), che produce un segnale periodico

L'unità centrale di elaborazione

- La ***frequenza*** del clock, costituisce una delle caratteristiche tecniche più importanti della CPU
  - si misura in **cicli al secondo** o **Hz (Hertz)**
  - es. 3 GHz (gigaHz), $3x10^9 =$ 3 miliardi di cicli al secondo



f = 1/τ τ si dice periodo

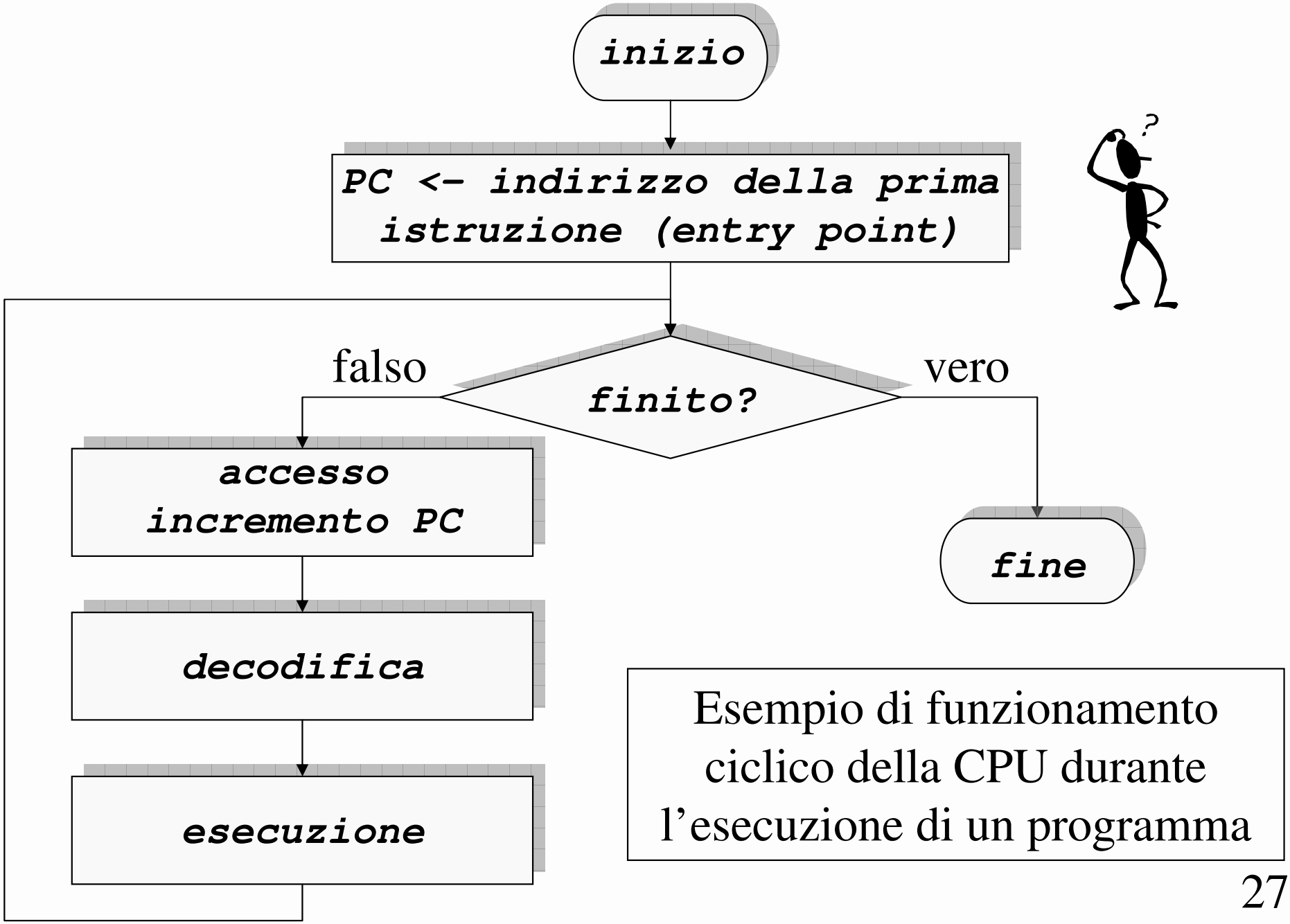
segnale periodico: `f(t) = f(t + τ)`

Ciclo di funzionamento della CPU

- Il ciclo di funzionamento è composto da tre fasi
  - **accesso (fetch)**:
    - **caricamento** dell'istruzione da eseguire dalla memoria primaria e sua memorizzazione nel ***Registro Istruzione***
    - **incremento** del ***Registro Contatore di Programma***
  - **decodifica (decode)**:
    - decodifica dell'istruzione da eseguire
  - **esecuzione (execute)**:
    - esecuzione dell'istruzione

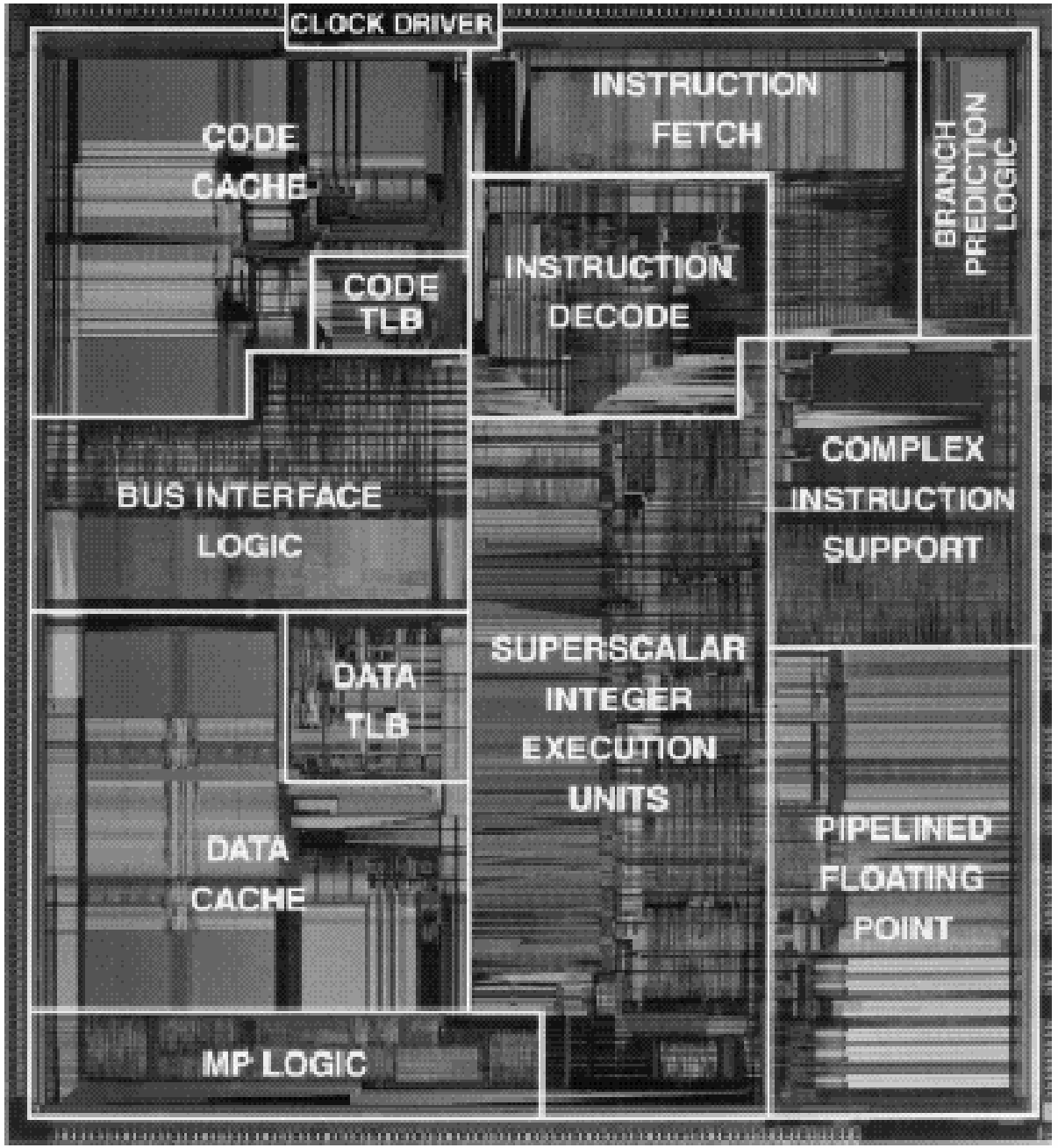
  Si parla di ciclo ***fetch-decode-execute***
- L'indirizzo in memoria centrale dell'istruzione da caricare durante la fase di *accesso* è contenuto in un registro speciale detto ***contatore di programma*** (*program counter*, PC)
  - viene incrementato di un'unità a ogni ciclo alla fine della fase di accesso, in modo da ***eseguire istruzioni in sequenza***

Ciclo di funzionamento della CPU



Esempio di funzionamento ciclico della CPU durante l'esecuzione di un programma

Unità Centrale di Elaborazione



La memoria

- La memoria serve a ***immagazzinare*** **dati** e **programmi** all'interno del computer

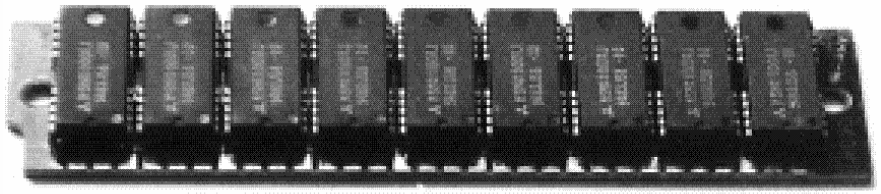
- È suddivisa in ***celle*** o ***locazioni*** di memoria, ognuna delle quali ha un ***indirizzo (numero progressivo)***

- Ogni cella contiene un numero predefinito di ***bit***, solitamente uguale a otto
  - bit è un dato elementare che può assumere due valori, *convenzionalmente* chiamati **zero** e **uno**
  - un insieme di **otto bit** si chiama ***byte*** ed è l'unità di misura della capacità di memoria
    - es. 512 MByte = 512 * 2^{20} bytes
    - nelle unita' di memoria k = 2^{10} = 1024, M = k * k = 2^{20} = 1048576

- Ci sono due tipi di memoria
  - **primaria** e **secondaria**

La memoria primaria (o centrale)

- La memoria ***primaria*** è **veloce** ma **costosa**
- È costituita da **chip di memoria** realizzati con la stessa tecnologia (al silicio) utilizzata per la CPU
  - memoria ***di sola lettura*** (**ROM**, *Read-Only Memory*)
  - memoria ad accesso casuale (**RAM**, *Random Access Memory*)
    - dovrebbe chiamarsi memoria ***di lettura e scrittura***, perché in realtà anche la ROM è ad accesso casuale, mentre ciò che le distingue è la possibilità di scrivervi
    - ***accesso casuale*** significa che ***il tempo per accedere a un dato non dipende dalla sua posizione nella memoria***



Indirizzo di una cella di memoria

- A ciascuna cella di memoria e' associato un ***numero progressivo*** a partire da zero, generalmente espresso in formato esadecimale.
- Questo numero e' detto *indirizzo* della cella di memoria

```
Indirizzo a 16  bit     Celle di memoria
(esadecimale)           a 8 bit (byte)

$ 0000                  |_|_|_|_|_|_|_|_|
$ 0001                  |_|_|_|_|_|_|_|_|
$ 0002                  |_|_|_|_|_|_|_|_|
...                     ...
$ FFFF                  |_|_|_|_|_|_|_|_|
```

Esempio di Memoria Primaria organizzata a byte

con indirizzo a 16 bit (2^{16} = 65 356 celle)

$2^{16} = 2^6 * 2^{10}$ = 64 kByte (2^{10} = 1024 = 1 kByte)

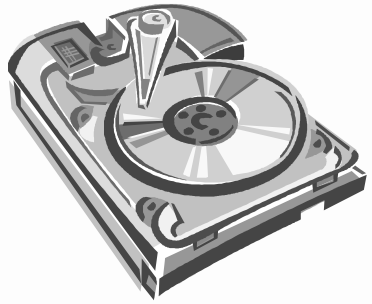
La memoria ROM

- **ROM:** memoria di sola lettura
  - conserva i dati ed i programmi memorizzati anche quando il computer viene spento
    - è una memoria ***non volatile***
  - contiene i programmi necessari all'avvio del computer, programmi che devono essere ***sempre disponibili***
    - nei PC, questi programmi prendono il nome di **BIOS** (**B**asic **I**nput/**O**utput **S**ystem)

La memoria RAM

- RAM: memoria ad accesso casuale
  - è una memoria che consente la ***lettura*** e la ***scrittura*** dei dati e dei programmi in essa contenuti
  - contiene dati in fase di modifica e programmi che devono essere disponibili per l'esecuzione
  - **perde i dati quando si spegne il computer** (è un supporto ***volatile***)

La memoria secondaria (o di massa)



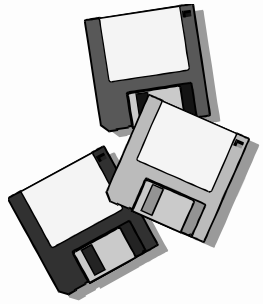
- La memoria ***secondaria*** (o *di massa*) è un supporto ***non volatile*** e ***meno costoso*** della memoria primaria per unità di memoria (circa cento volte)
  - programmi e dati risiedono nella memoria secondaria e vengono caricati nella RAM quando necessario, per poi tornarvi aggiornati se e quando necessario

- E' di solito un **disco rigido** (o disco fisso, *hard disk*)

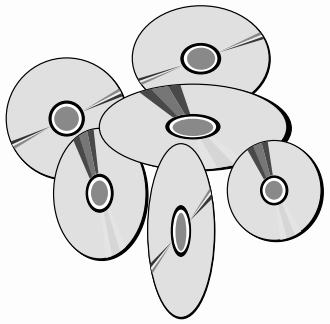
- Un disco rigido è formato da piatti rotanti rivestiti di materiale magnetico, con testine di lettura/scrittura
- Attuali dimensioni: parecchie centinaia di GByte

La memoria secondaria

- Sono molto diffusi anche altri tipi di memoria secondaria a tecnologia magnetica
  - ***floppy disk*** (**dischetto** flessibile), di capacità limitata (**1.4 Mbyte**) ma con il vantaggio di poter essere agevolmente rimosso dal sistema e trasferito ad un altro sistema (dispositivo di memoria ***esterno***)
  - ***flash memory***, memoria permanente riscrivibile;
    - allo stato solido (non presenta parti in movimento)
  - ***tape*** (**nastri** per dati), di **capacità elevatissima**, molto economici, ma molto lenti, perché ***l'accesso ai dati è sequenziale*** anziché casuale (bisogna avvolgere o svolgere un nastro invece che spostare la testina di lettura sulla superficie di un disco)



La memoria secondaria



- Sono molto diffusi anche altri tipi di memoria secondaria a tecnologia ***ottica***
  - ***CD-ROM*** (*Compact Disc Read-Only Memory*), viene letto da un dispositivo laser, esattamente come un CD audio; ha una elevata capacità ed è molto economico e affidabile; è un supporto di sola lettura, utilizzato per distribuire programmi e informazioni (750 Mbyte)
  - ***CD-R*** (*Compact Disc Recordable*), utilizza una tecnologia simile al CD-ROM ma può essere scritto dall'utente (una sola volta; più volte se CD-RW)

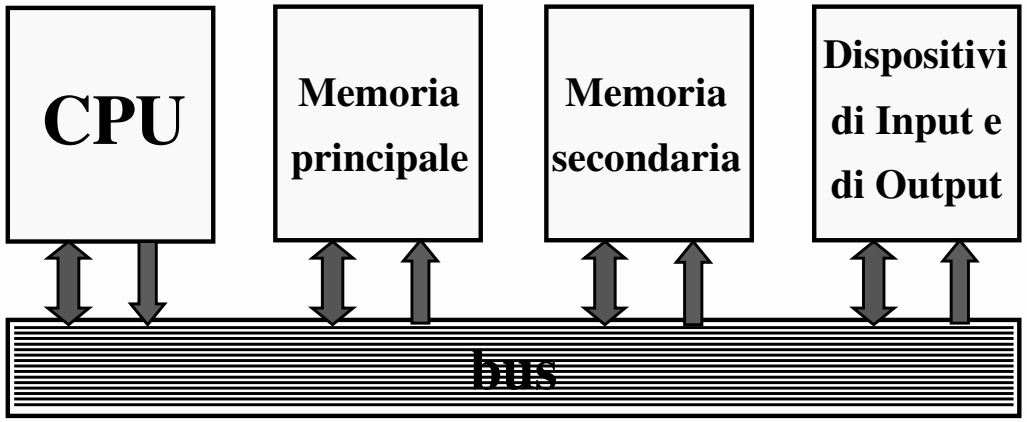
  - ***DVD*** *(Digital Versatile Disc)*, elevata capacità (Gbyte)

Dispositivi periferici di interazione

- L'interazione fra l'utente umano ed il computer avviene mediante i cosiddetti ***dispositivi periferici di Ingresso/Uscita***
  - ***dispositivi di Input/Output*** o ***dispositivi di* I/O**)
- Tipici dispositivi di ***input*** sono la **tastiera**, il **mouse** (dispositivo di puntamento), il **microfono** (per impartire comandi vocali), il **joystick** (per i giochi), lo **scanner** (per la scansione digitale di documenti e immagini)
- Tipici dispositivi di ***output*** sono lo **schermo** (*monitor*), le **stampanti**, gli **altoparlanti**

Il bus nel modello di von Neumann

- Il bus è in realtà costituito da tre componenti distinti
  - bus dei ***dati***
  - bus degli ***indirizzi***
  - bus dei ***segnali di controllo***
- Sul bus dei **dati** viaggiano dati **da e verso la CPU**
- Sugli altri bus viaggiano indirizzi e segnali di controllo che **provengono di norma dalla CPU**



L'architettura di un computer

- Per capire i meccanismi di base della programmazione è necessario conoscere gli **elementi hardware** che costituiscono un computer
- Prendiamo in esame il ***Personal Computer*** (**PC**), ma anche i computer più potenti hanno un'**architettura** molto simile

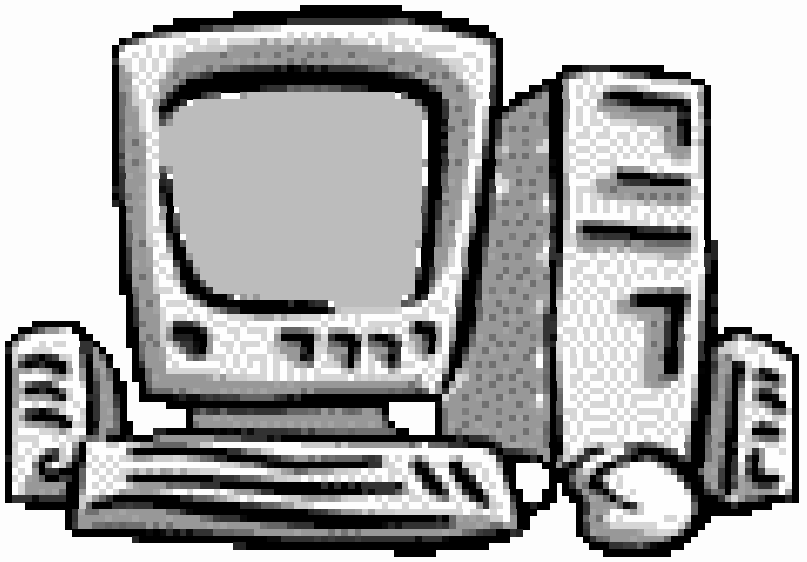
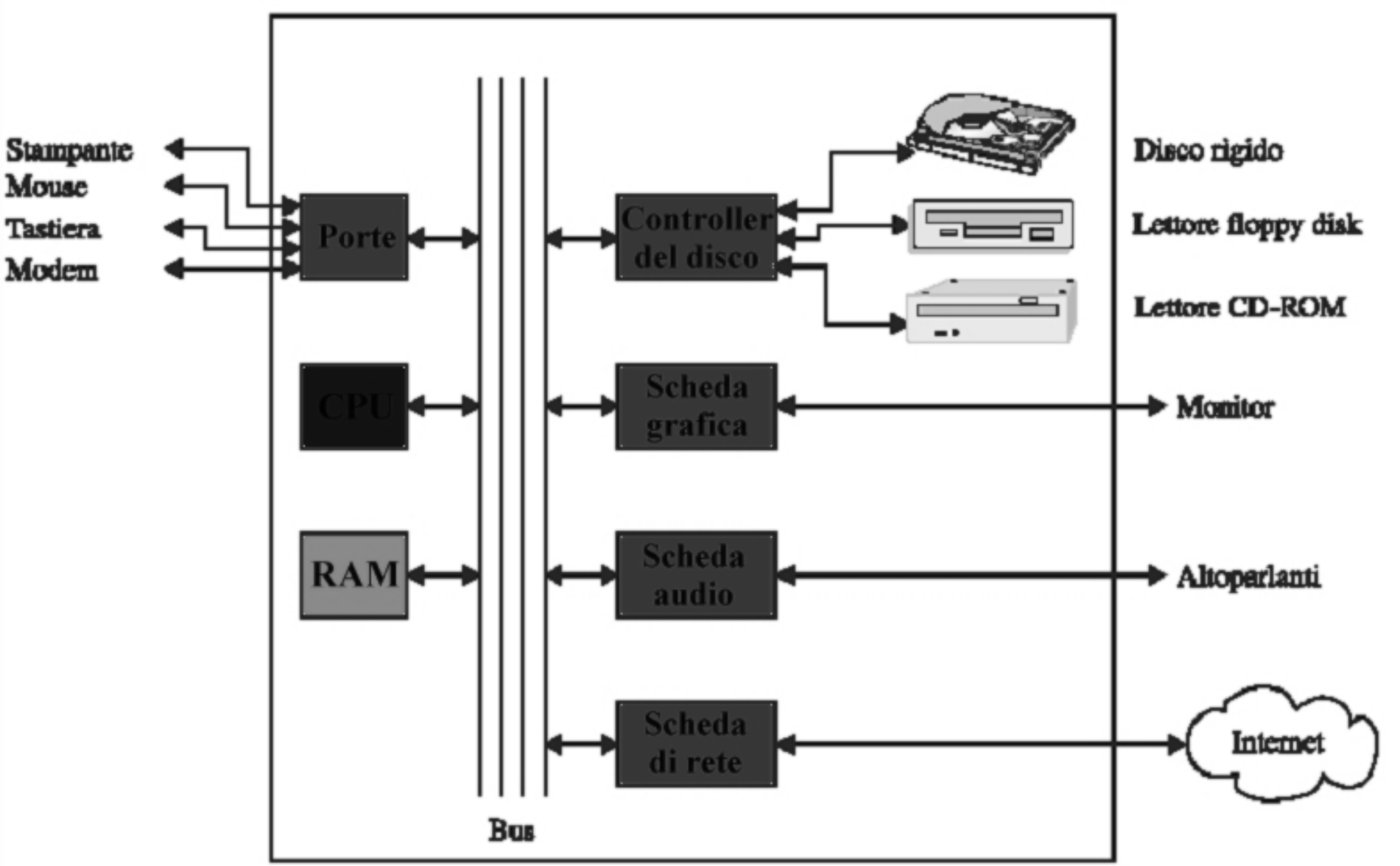


Diagramma schematico di un personal computer

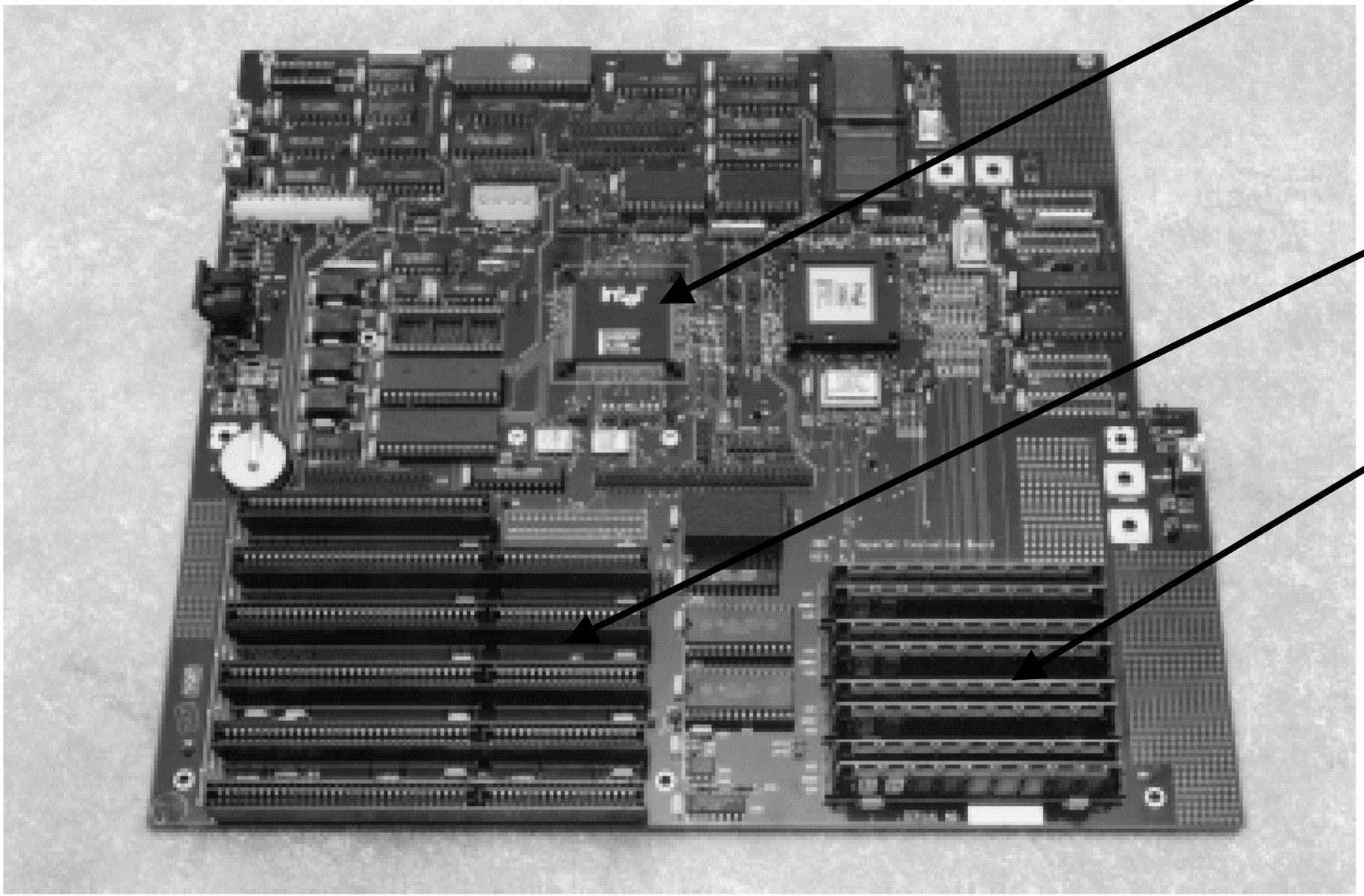


La scheda madre di un PC

- All'interno del PC si trova la ***scheda madre*** (***mother-board***), che contiene la CPU, la memoria primaria, il bus e gli alloggiamenti (***slot***) di espansione per il controllo delle periferiche

- La CPU, la memoria primaria (RAM e ROM) e i circuiti elettronici che controllano il disco rigido e altri dispositivi periferici sono interconnessi mediante un insieme di linee elettriche che formano il ***bus***
- I dati transitano lungo il bus, dalla memoria e dai dispositivi periferici verso la CPU, e viceversa

La scheda madre di un PC



Programmare in codice macchina

Le istruzioni macchina

- Le istruzioni elementari eseguite da un computer, cioè dalla sua CPU, si chiamano
  istruzioni macchina
- L'insieme di istruzioni macchina (***instruction set***) è **specifico** di una particolare CPU: quello del processore Intel Pentium è diverso da quello del processore Motorola PowerPC
- Una particolare CPU è la cosiddetta ***macchina virtuale Java*** (JVM, *Java Virtual Machine*)
  - la JVM non è una vera CPU… ma per il momento possiamo considerarla tale…

Le istruzioni macchina

- La codifica delle istruzioni macchina avviene sotto forma di ***configurazioni di bit*** conservate in memoria (che possono essere interpretate come numeri interi)
- Esempio di alcune istruzioni macchina:
  - carica in un registro il valore contenuto nella posizione di memoria $40: *codifica: 21 40*
  - carica in un altro registro il valore 100: *cod. 16 100*
  - se il primo valore è maggiore del secondo, prosegui con l'istruzione contenuta nella posizione di memoria 240, altrimenti con l'istruzione che segue quella attuale
    - *cod. 163 240*

- Le precedenti istruzioni per la JVM diventano quindi

  21 40 16 100 163 240

Le istruzioni macchina

- In tutte le CPU, le istruzioni macchina si possono suddividere nelle seguenti categorie (i nomi delle istruzioni sono solo degli esempi)
  - ***trasferimento dati***, tra i registri e la memoria principale
    - LOAD (verso un registro), STORE (verso la memoria)
  - ***operazioni aritmetiche e logiche***, eseguite dalla ALU
    - aritmetiche: ADD, SUB, MUL, DIV
    - logiche: AND, OR, NOT
  - ***salti***, per alterare il flusso di esecuzione sequenziale (viene modificato il Program Counter)
    - incondizionato (JUMP): salta in ogni caso
    - condizionato: salta solo se un certo valore è zero (JZ) o se è maggiore di zero (JGZ)

Le istruzioni macchina

- Per eseguire un programma in un computer è necessario ***scrivere all'interno della memoria primaria le configurazioni di bit corrispondenti alle istruzioni macchina del programma***
- Per fare ciò è necessario conoscere tutti i codici numerici delle istruzioni macchina

  21 40 16 100 163 240 ...
- Questa operazione lunga e noiosa (che veniva eseguita manualmente agli albori dell'informatica) è stata presto automatizzata da *un programma in esecuzione sul computer stesso*, detto ***assemblatore*** (***assembler***)

Le istruzioni macchina

- La CPU è un ***interprete*** dell'***insieme di istruzioni*** della CPU stessa.

- Significa che la CPU è una macchina capace di eseguire le istruzioni macchina definite nell'insieme delle istruzioni della CPU stessa.

Lezione III
Me 3 Ott. 2007

Programmare
in linguaggio Assembly

L'assemblatore

- Utilizzando l'assemblatore, il programmatore scrive il programma mediante dei ***nomi abbreviati*** (***codici mnemonici***) per le istruzioni macchina, molto più facili da ricordare
  - esempio precedente per la JVM

```
iload       40
bipush     100
if_icmpgt 240
```

- L'uso di nomi abbreviati è assai più agevole, e il programma assemblatore si occupa poi di ***tradurre*** il programma in configurazioni di bit
- Tali linguaggi con codici mnemonici si dicono ***linguaggi assembly*** (uno diverso per ogni CPU)

L'assemblatore

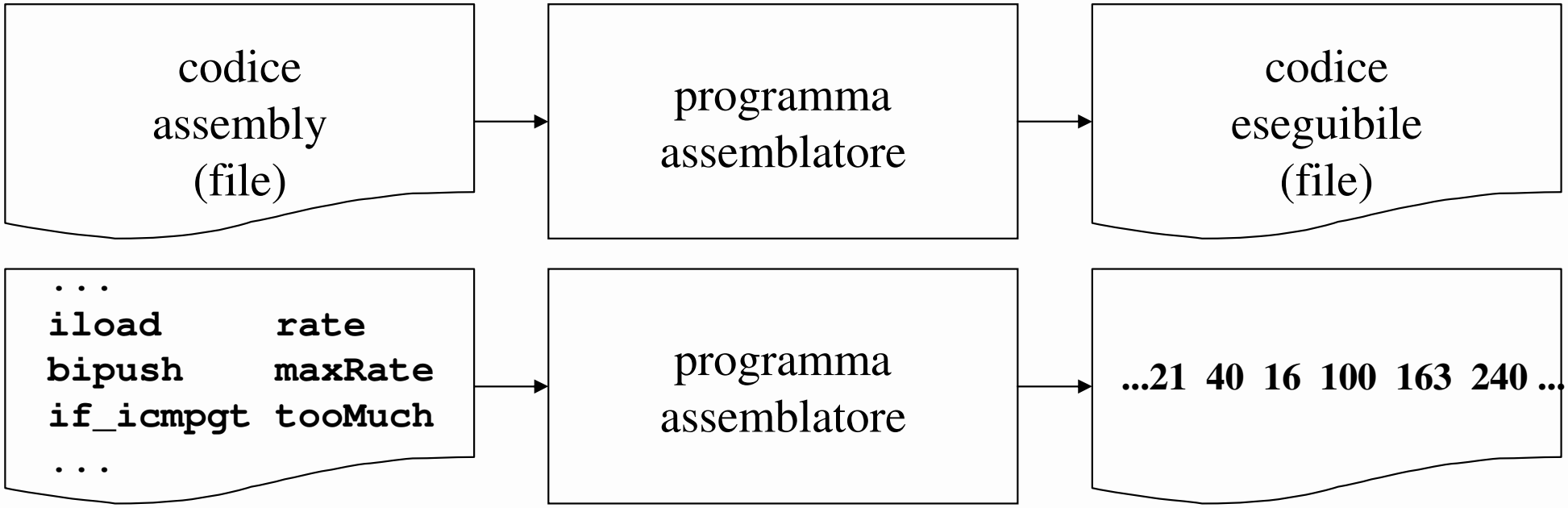
- Un'altra caratteristica molto utile dell'assemblatore è quella di poter assegnare dei ***nomi*** agli ***indirizzi di memoria*** e ai ***valori numerici*** e di usarli nelle istruzioni

```
iload      rate
bipush     maxRate
if_icmpgt  tooMuch
```

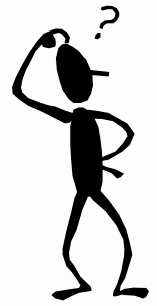
- In questo modo il programma è ***molto più leggibile***, perché viene evidenziato il ***significato*** degli indirizzi di memoria e dei valori numerici

- Corrispondenza biunivoca fra insieme di istruzioni assembly e instruction set del processore

L'assemblatore



- Il programma assemblatore traduce istruzioni assembly in istruzioni macchina (corrispondenza biunivoca)
- Il programma assemblatore riceve in ingresso un ***file*** contenente codice in linguaggio assembly e produce in uscita un ***file*** contenente istruzioni macchina.
- Per ora un file sia una collezione di dati, memorizzata nella memoria secondaria e identificata con un nome



I linguaggi assembly

- **Vantaggio**: rappresentarono un grosso passo avanti rispetto alla programmazione in linguaggio macchina
- **Problema**: occorrono ***molte istruzioni*** per eseguire anche le operazioni più semplici
- **Problema**: ciascuna CPU ha il proprio linguaggio Assembly quindi la sequenza di istruzioni di uno stesso programma ***cambia*** al cambiare della CPU

  ⇒ è molto costoso scrivere programmi che possano funzionare su diverse CPU, perché praticamente bisogna riscriverli completamente

Linguaggi di programmazione ad alto livello

Linguaggi ad alto livello

- Negli anni ‘50 furono inventati i primi linguaggi di programmazione ***ad alto livello***
  - **FORTRAN**: primo “vero” linguaggio
    - **BASIC**, **COBOL**
  - Anni ‘60 e ‘70: programmazione strutturata
    - **Pascal** (Niklaus Wirth, 1968)
    - **C** (Brian Kernigham e Dennis Ritchie, 1970-75)
  - Anni ‘80 e ‘90, programmazione orientata agli oggetti
    - **C++** (Bjarne Stroustrup, 1979)
    - **Java** (James Gosling e Patrick Naughton, 1991)
- Il programmatore esprime la sequenza di operazioni da compiere, senza scendere al livello di dettaglio delle istruzioni macchina

```
if (rate > 100)
    System.out.print("Troppo");
```

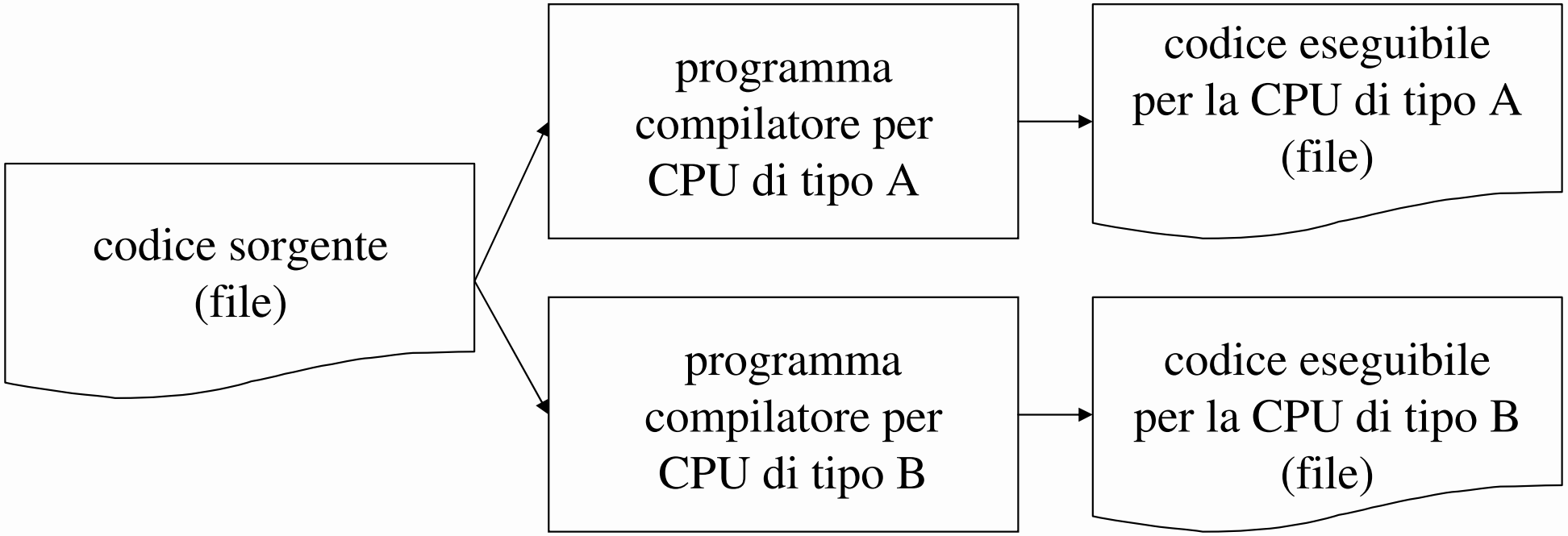
Compilatore

- Un programma, detto ***compilatore***, legge il programma in linguaggio ad alto livello e genera il corrispondente programma nel linguaggio macchina di una specifica CPU
- I linguaggi ad alto livello sono

  indipendenti dalla CPU

  ma il prodotto della compilazione (***codice eseguibile***), non è indipendente dalla CPU
  - occorre compilare il programma con un diverso compilatore per ogni CPU sulla quale si vuole eseguire il programma stesso
- Si dice ***codice sorgente*** il codice scritto in un linguaggio ad alto livello
- Si dice ***codice eseguibile*** il codice scritto nel linguaggio macchina

Compilatore



- Il compilatore traduce codice in linguaggio ad alto livello in codice eseguibile di uno specifico processore
- riceve in ingresso un file contenente codice in linguaggio ad alto livello e produce in uscita un file contenente codice eseguibile di uno specifico compilatore
- Lo stesso codice sorgente può essere tradotto in codice eseguibile di processori diversi usando compilatori diversi

Linguaggi ad alto livello

- Esistono molti linguaggi di programmazione ad alto livello, così come esistono molte lingue

- L'esempio seguente è in linguaggio Java

```
if (rate > 100) System.out.print("Troppo");
```

e questo è l'equivalente in linguaggio Pascal

```
if rate > 100 then write ('Troppo');
```

- La sintassi di un linguaggio viene scelta dai progettisti del linguaggio stesso, come compromesso fra leggibilità, facilità di compilazione e coerenza con altri linguaggi

Linguaggi ad alto livello

parentesi di troppo

- I linguaggi di programmazione, creati dall'uomo, hanno una sintassi molto più rigida di quella dei linguaggi naturali, per agevolare il compilatore
- Il compilatore segnala gli ***errori di sintassi***

```
if (rate > 100)) System.out.print("Troppo");
```

e ***non tenta di capire***, come farebbe un utente umano, perché sarebbe molto difficile verificare le ***intuizioni*** del compilatore

meglio la segnalazione di errori!

Il linguaggio Java

- Nato nel 1991 in Sun Microsystems, da un gruppo di progettisti guidato da Gosling e Naughton
- Progettato per essere ***semplice*** e ***indipendente dalla CPU*** (o, come anche si dice, dall'hardware, dalla piattaforma o dall'architettura)

- Il primo prodotto del progetto Java fu un ***browser*** (programma per navigare in internet) , **HotJava**, presentato nel 1994, che poteva scaricare programmi (detti ***applet***) da un server ed eseguirli sul computer dell'utente, ***indipendentemente*** dalla sua piattaforma

Il linguaggio Java

- Il linguaggio Java è stato adottato da moltissimi programmatori perché
  - è più ***semplice*** del suo diretto concorrente, C++
  - consente di ***scrivere e compilare una volta ed eseguire dovunque*** (cioè su tutte le piattaforme)
    "*compile once, execute everywhere*"
  - ha una ***ricchissima libreria*** che mette a disposizione dei programmatori un insieme vastissimo di funzionalità ***standard***, indipendenti dal sistema operativo

Il linguaggio Java per gli studenti

- Dato che Java non è stato progettato per la didattica
  - ***non è così semplice scrivere programmi Java molto semplici!***
- Anche per scrivere programmi molto semplici è necessario conoscere parecchi dettagli "tecnici", un potenziale problema nelle prime lezioni
- Adotteremo lo stile didattico di ***usare alcuni costrutti sintattici senza spiegarli o approfondirli***, rimandando tali fasi al seguito

Il nostro primo programma Java

- Tradizionalmente, il primo programma che si scrive quando si impara un linguaggio di programmazione ha il compito di visualizzare sullo schermo un semplice saluto

```
Hello, World!
```

- Scriviamo il programma nel file

  Hello.java

  usando un ***editor*** di testi

  editor di testi = programma per scrivere testi

- Non si deve usare un programma di videoscrittura!

Editor e programmi di videoscrittura

- Entrambi sono programmi per scrivere testi, ma...
- un editor memorizza solo caratteri, niente altro
  - `caro amico` (anche lo spazio è un carattere!)
  - es: MS notepad (blocco note), linux gedit e kwrite, *context* (Ms e linux)
- un programma di videoscrittura memorizza oltre i caratteri anche informazioni aggiuntive quali, ad esempio, tipo di carattere (font), dimensione, stile (grassetto, italico, sottolineato), allineamento, colore, ...
  - caro amico ***caro amico***
  - es.: *MS Word* e ***OpenOffice Writer*** (linux, free) sono programmi di videoscrittura!

Primo programma Java

```
public class Hello
{
  public static void main(String[] args)
  {
    System.out.println("Hello, World!");
  }
}
```

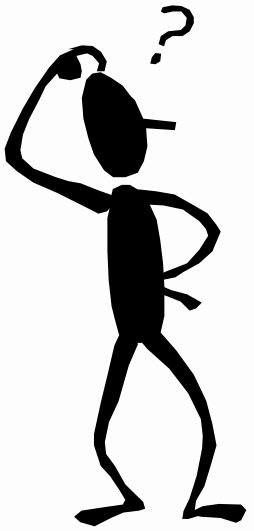
Occorre fare attenzione

- il testo va inserito esattamente come è presentato
  - per il momento…
- ***maiuscole e minuscole sono considerate distinte***
- il testo va inserito in un file che ***deve*** chiamarsi **Hello.java**

Il nostro primo programma Java

- A questo punto ***compiliamo*** il programma

```
javac Hello.java
```



e il compilatore genera il file **Hello.class**

- Ora ***eseguiamo*** il programma

```
java Hello
```

ottenendo la visualizzazione del messaggio di saluto sullo schermo

```
Hello, World!
```

Una nota stilistica

- Diversamente da altri linguaggi (es. FORTRAN) il linguaggio Java consente una ***disposizione del testo a formato libero***: gli spazi e le interruzioni di riga ("andare a capo") non sono importanti, tranne che per separare parole
- Sarebbe quindi possibile scrivere

```
public class Hello{public static void
   main (String[] args) { System.out.
println("Hello, World!")              ;}}
```

- Bisogna però ***fare attenzione alla leggibilità!***

Analisi del primo programma

- La prima riga

```
public class Hello
```

  definisce una nuova ***classe, la classe Hello***

- Le classi sono ***fabbriche di oggetti*** e rappresentano un concetto fondamentale in Java, che è un linguaggio di programmazione ***orientato agli oggetti*** (**OOP**, ***O****bject-****O****riented* ***P****rogramming*)
- Per il momento, consideriamo gli ***oggetti*** come ***elementi da manipolare in un programma Java***
- Un programma (detto anche applicazione) Java e' costituito da una o piu' classi (generalmente molte)

Analisi del primo programma

- La ***parola chiave*** **public** indica che la classe **Hello** può essere utilizzata da tutti

```
public class Hello
```

- Una parola chiave è una parola riservata del linguaggio che va scritta esattamente così com'è e che non può essere usata per altri scopi
- La parola chiave **class** indica che inizia la **definizione** di una ***classe***
- Ciascun ***file sorgente*** (parte di un programma Java) può contenere ***una sola classe pubblica***, il cui nome ***deve coincidere*** con il nome del file

Analisi del primo programma

- Le classi oltre a essere fabbriche di oggetti, sono anche ***contenitori di metodi***
- Un metodo definisce una sequenza di istruzioni o ***enunciati*** che ***descrive come svolgere un determinato compito*** (in altri linguaggi i metodi si chiamano *funzioni* o *procedure*)
- Un metodo ***deve*** essere inserito in una classe, quindi le classi rappresentano il contenitore per l'organizzazione dei programmi

Analisi del primo programma

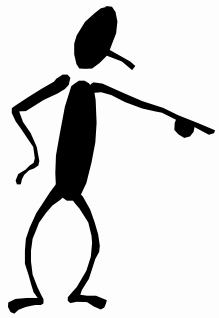
- La costruzione

```
public static void main(String[] args)
{   ...
}
```

definisce il metodo **main** ( ) (*principale*)

- Un'applicazione Java ***deve*** avere un metodo **main()** da cui inizia l'esecuzione del programma
- Anche qui, **public** significa ***utilizzabile da tutti***
- static, void, String[] args per intanto lasciamoli senza commento. Vedremo piu' avanti

Programma semplice



- Sintassi:

```
public class NomeClasse
{  public static void main(String[] args)
   {  enunciati
   }
}
```

- Scopo: eseguire un programma semplice, descritto da ***enunciati*** e contenuto nel file ***NomeClasse.java***
- Nota: la parte in **blu** viene per ora considerata una ***infrastruttura necessaria***, approfondita in seguito

Lezione IV
Gi 4 Ott. 2007

Analisi del primo programma

Analisi del primo programma

- Gli enunciati del ***corpo*** di un metodo (*gli enunciati contenuti tra le parentesi graffe*) vengono eseguiti uno alla volta ***nella sequenza in cui sono scritti***
- Ogni enunciato termina con il carattere `;`
- Il metodo **main**() del nostro esempio ha un solo enunciato, che visualizza una riga di testo

```
System.out.println("Hello, World!");
```

- Ma ***dove*** la visualizza? Un programma può inserire testo in una finestra, scriverlo in un file o anche inviarlo ad un altro computer attraverso Internet...

Analisi del primo programma

```
System.out.println("Hello, World!");
```

- Nel nostro caso la destinazione è l'***uscita standard (o standard output)***, una proprietà di ciascun programma che dipende dal sistema operativo del computer
- In java, l'output standard è rappresentato da un ***oggetto*** di nome **`out`**
  - come ogni metodo, ***anche gli oggetti devono essere inseriti in classi***: **`out`** è inserito nella classe **`System`** che contiene oggetti e metodi da utilizzare per accedere alle ***risorse di sistema***
  - per ***usare*** l'oggetto **out** della classe **System** si scrive

```
System.out
```

Analisi del primo programma

```
System.out.println("Hello, World!");
```

- ***System*** è una classe della ***java platform API*** che è una collezione di classi pronte per essere usate, fornite dal progettista del linguaggio insieme al linguaggio Java

- ***Application Program Interface (API)*** significa Interfaccia per la programmazione di applicazioni

- ***System.out*** e' un oggetto definito nella classe `System` di typo ***PrintStream***

- ***PrintStream*** è un'altra classe della ***java platform API***

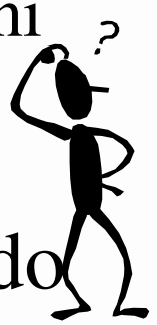
Analisi del primo programma

```
System.out.println("Hello, World!");
```

- Quando si usa un oggetto, bisogna specificare ***cosa*** si vuol fare con l'oggetto stesso
  - in questo caso vogliamo ***usare un metodo*** dell'oggetto **`out`**, il metodo **println**(), che stampa una riga di testo
  - per ***usare*** il metodo **`println()`** dell'oggetto **`System.out`** si scrive

    ```
    System.out.println(parametri)
    ```

  - la coppia di parentesi tonde racchiude le informazioni necessarie per l'esecuzione del metodo (***parametri***)
- A volte il carattere ***punto*** significa "usa un oggetto di una classe", altre volte "usa un metodo di un oggetto": dipende dal contesto...



Analisi del primo programma

```
System.out.println("Hello, World!");
```

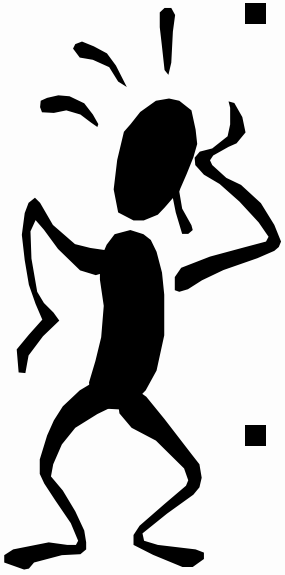
- `System.out.print( )` e' un altro metodo dell'oggetto out che puo' essere usato a inviare una stringa a standard output
  - a differenza di `println()` non va a nuova riga

- L'ambiente java fornisce una documentazione esaustiva della libreria standard
  - ***java platform API specification*** in formato html

Errori di programmazione

```
System.out.println("Hello, World!");
```

- L'attività di programmazione, come ogni altra ***attività di progettazione***, è soggetta ad errori di vario tipo
  - ***errori di sintassi*** **o di compilazione**



```
System.aut.println("Hello, World!");
```

```
System.out.println("Hello, World!);
```

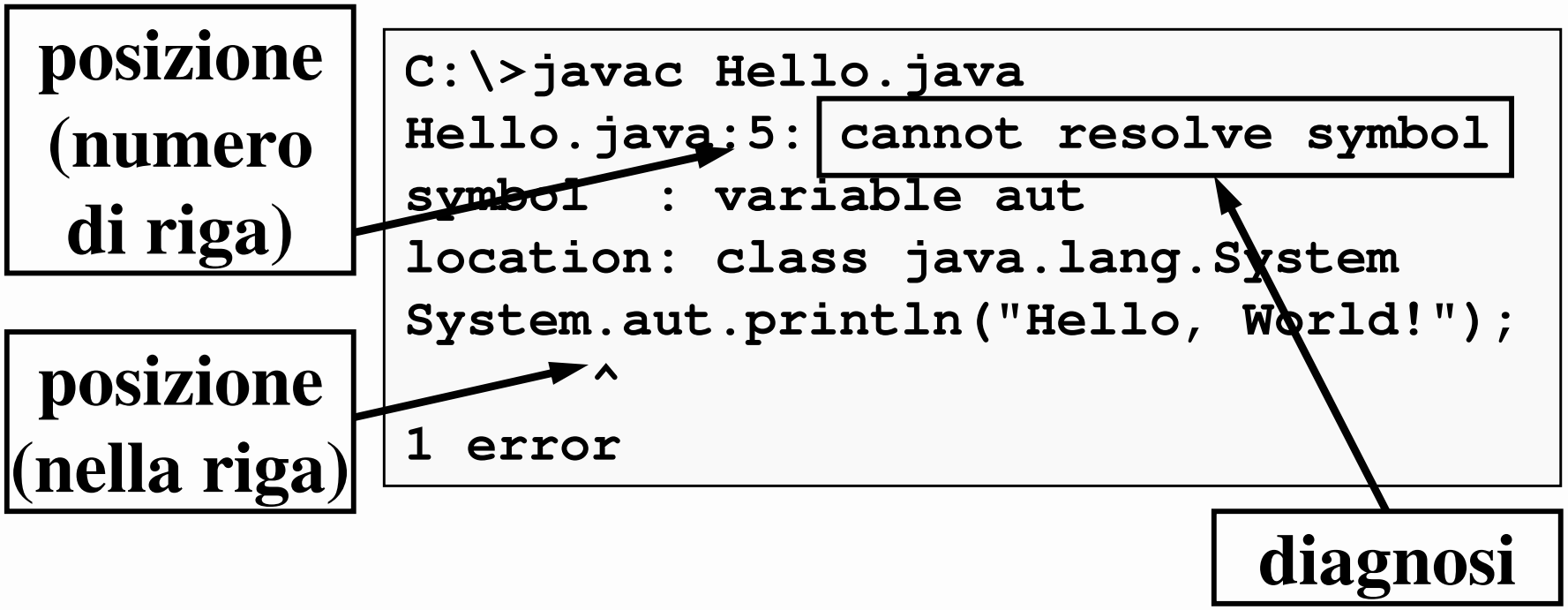
  - ***errori logici*** **o di esecuzione**

```
System.out.println("Hell, World!");
```

Errori di sintassi

```
System.aut.println("Hello, World!");
```

- In questo caso il compilatore riesce agevolmente ad individuare e segnalare l'errore di sintassi, perché identifica il nome di un oggetto (***simbolo***) che non è stato definito (**aut**) e sul quale non è in grado di "***decidere***"



Errori di sintassi

virgolette mancanti

```
System.out.println("Hello, World!);
```

- Questo è invece un caso molto più complesso: viene giustamente segnalato il **primo errore**, una stringa non terminata, e viene evidenziato il punto dove ***inizia*** la stringa

```
C:\>javac Hello.java
Hello.java:5: unclosed string literal
System.out.println("Hello, World!);
                   ^
Hello.java:5: ')' expected
System.out.println("Hello, World!);
                                  ^
2 errors
```

Errori di sintassi

- Viene però segnalato anche un **secondo errore**
  - il compilatore ***si aspetta di trovare una parentesi tonda chiusa***, in corrispondenza di quella aperta
  - ***la parentesi in realtà c'è***, ma il compilatore l'ha inserita all'interno della stringa, cioè ***ha prolungato la stringa fino al termine della riga***

```
C:\>javac Hello.java
Hello.java:5: unclosed string literal
System.out.println("Hello, World!);
                   ^
Hello.java:5: ')' expected
System.out.println("Hello, World!);
                                  ^
2 errors
```

Errori logici

```
System.out.println("Hell, World!");
```

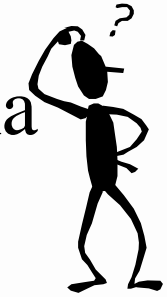
manca un carattere

- Questo errore, invece, ***non viene segnalato dal compilatore***, che non può sapere che cosa il programmatore abbia intenzione di far scrivere al programma sull'output standard
  - ***la compilazione va a buon fine***
  - si ha un errore durante l'***esecuzione*** del programma, perché viene prodotto un output **diverso** dal previsto

```
C:\>java Hello
Hell, World!
```

Errori logici

- Sono molto più insidiosi degli errori di sintassi
  - ***il programma*** viene compilato correttamente, ma ***non fa quello che dovrebbe fare***
- L'eliminazione degli errori logici richiede molta ***pazienza***, eseguendo il programma e osservando con attenzione i risultati prodotti
  - ***è necessario collaudare i programmi, come qualsiasi altro prodotto dell'ingegneria***
- Si usano programmi specifici (***debugger)*** per trovare gli errori logici (***bug)*** in un programma



Fasi della Programmazione

Le fasi della programmazione

- L'attività di programmazione si esegue in tre fasi
  - scrittura del programma (codice ***sorgente***)
  - compilazione del codice sorgente
    - creazione del codice ***eseguibile*** (codice macchina)
  - esecuzione del programma
- Per scrivere il codice sorgente si usa un ***editor di testo***, *salvando* (memorizzando) il codice in un file

  Hello.java

Individuare il compilatore Java

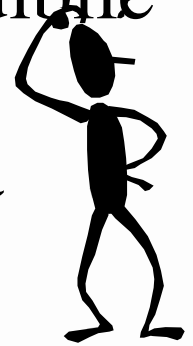
- Il modo di utilizzo del compilatore Java dipende dal sistema operativo
  - si seleziona con il mouse un'icona sullo schermo
  - si seleziona una voce in un menu di comandi
  - ***si compone il nome di un comando sulla tastiera***
  - si utilizza un ambiente integrato per lo sviluppo software (IDE, *Integrated Development Environment*)
- Nel nostro corso useremo i comandi da tastiera del JDK (*Java Development Kit*) di Sun Microsystems

La compilazione del sorgente

- Compilando il codice sorgente di un programma in Java (gli enunciati in linguaggio Java) si ottiene un particolare formato di ***codice eseguibile***, detto ***bytecode***, che è ***codice macchina*** per la ***Java Virtual Machine (JVM)***

  javac Hello.java *genera* Hello.class

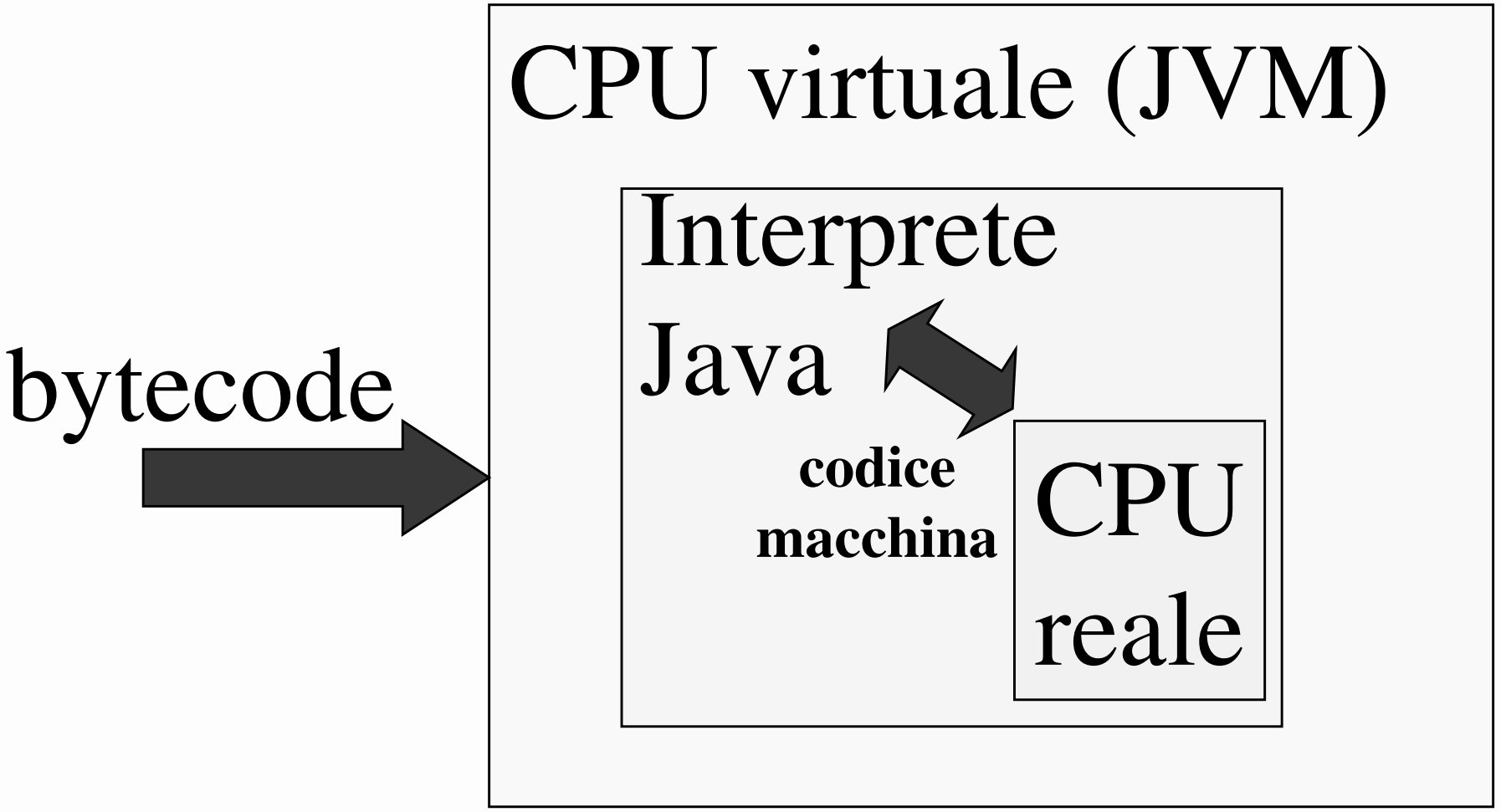
- Quindi il bytecode non è codice direttamente eseguibile dalla CPU reale, dato che la JVM non è una CPU fisica…e il bytecode e' codice macchina della Java Virtual Machine



L'esecuzione del programma

- Per ***eseguire*** un programma si usa l'***interprete Java***, un programma eseguibile sul computer dell'utente che
  - ***carica in RAM il bytecode del programma*** (della classe **Hello**)
  - avvia il programma eseguendo il metodo **main** di tale classe
  - carica successivamente i file di bytecode di altre classi che sono necessarie durante l'esecuzione (ad esempio, la classe **System**)
  - traduce "al volo" le istruzioni del bytecode in istruzioni macchina della CPU reale (che esegue il codice)
- **java Hello *scrive* Hello, World! *sullo standard output***

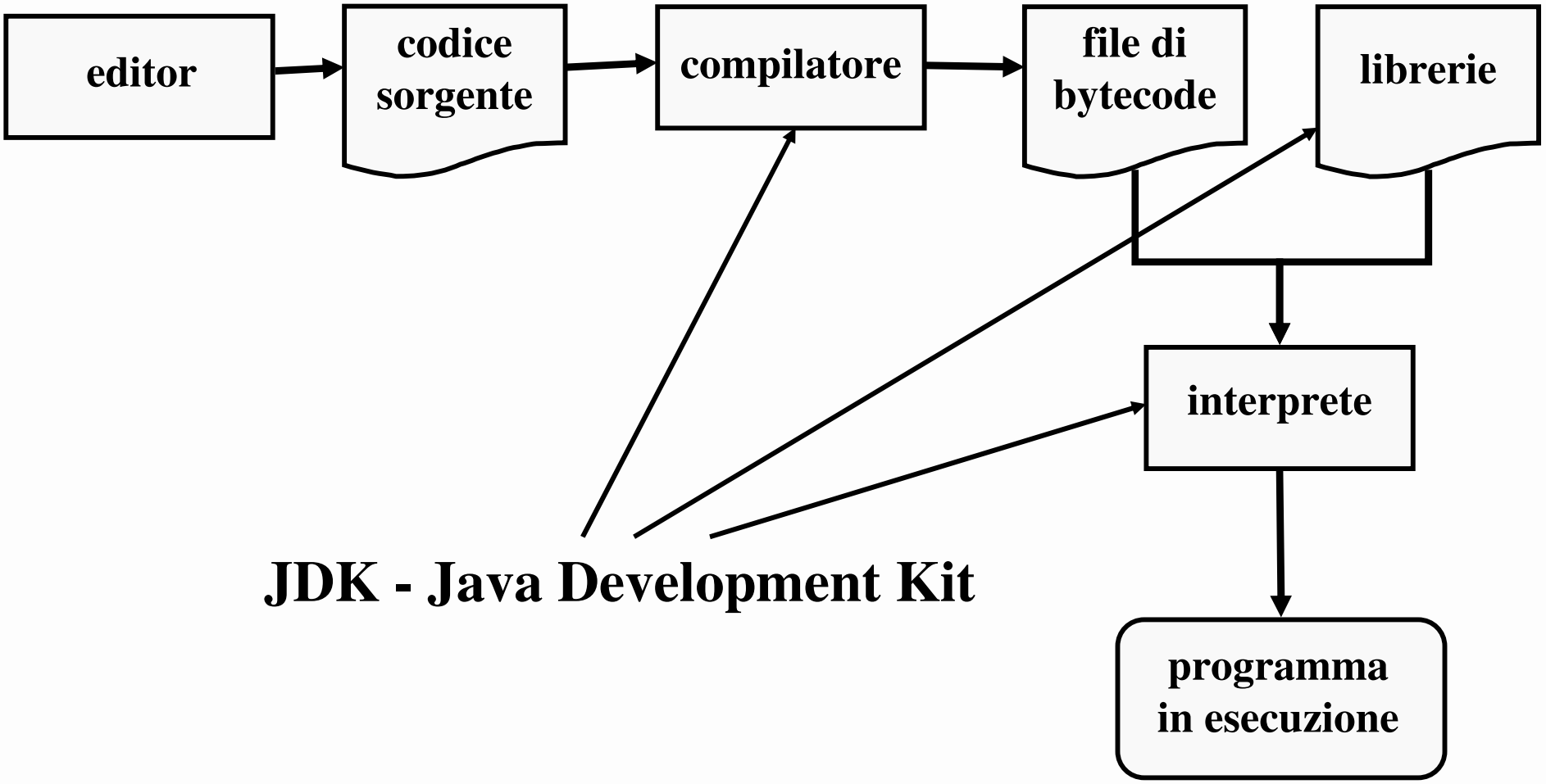
La Java Virtual Machine



La libreria di classi standard

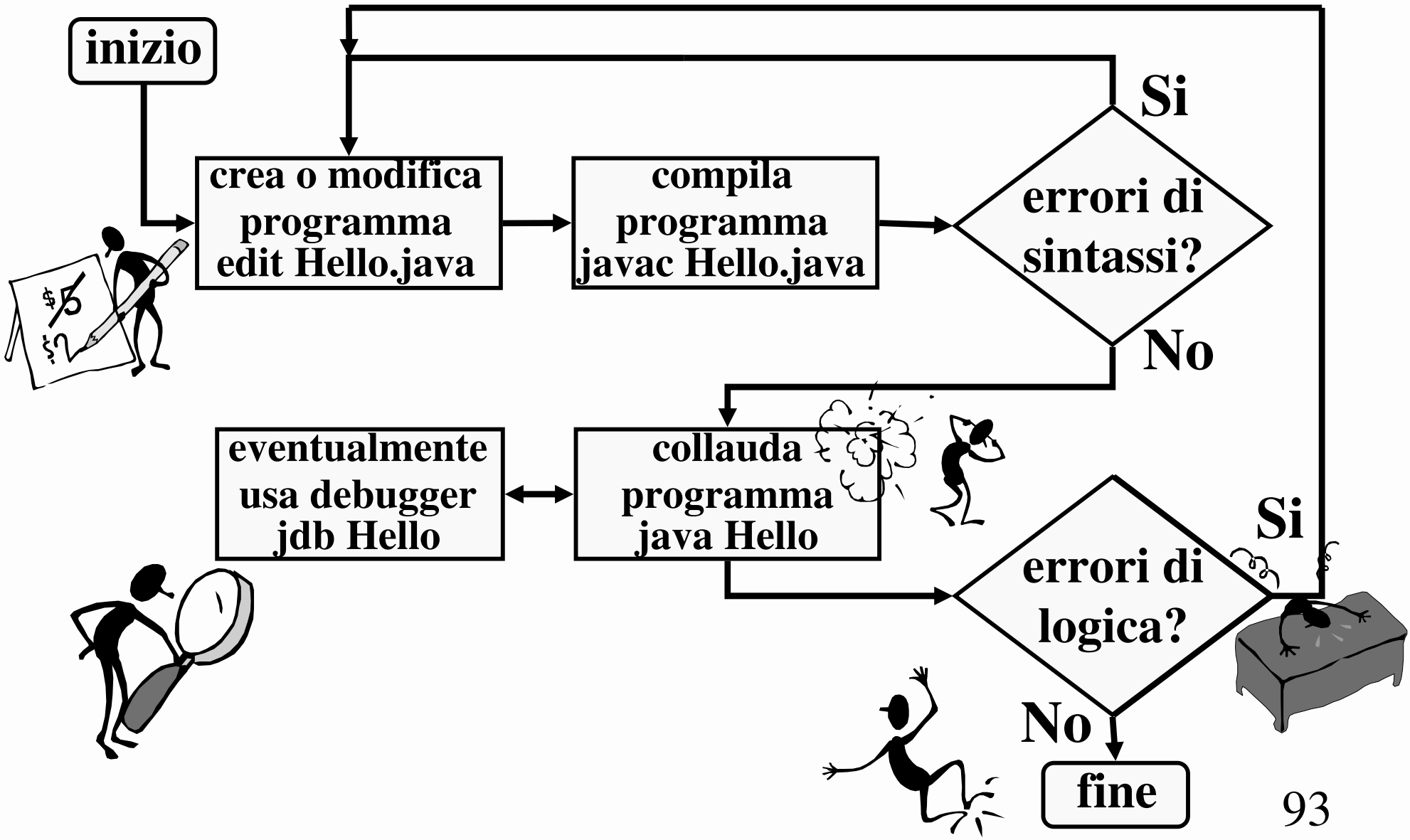
- Per scrivere sullo standard output è necessario ***interagire con il sistema operativo***, un'operazione ***di basso livello*** che richiede conoscenze specifiche
- Queste operazioni, per chi utilizza il linguaggio Java, sono state già realizzate dagli autori del linguaggio (Sun Microsystems), che hanno scritto delle classi apposite (ad esempio, **System**)
- Il bytecode di queste classi si trova all'interno di ***java platform API***
- ***Non è necessario avere a disposizione il codice sorgente di queste classi, né capirlo!*** Comodo...

Il processo di programmazione in Java



E se qualcosa non funziona?

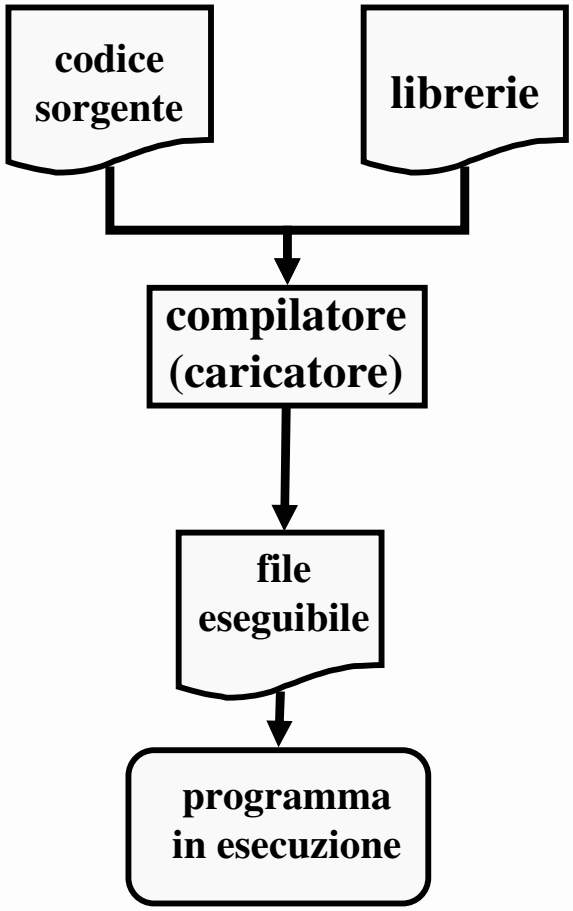
Modifica-Compila-Collauda



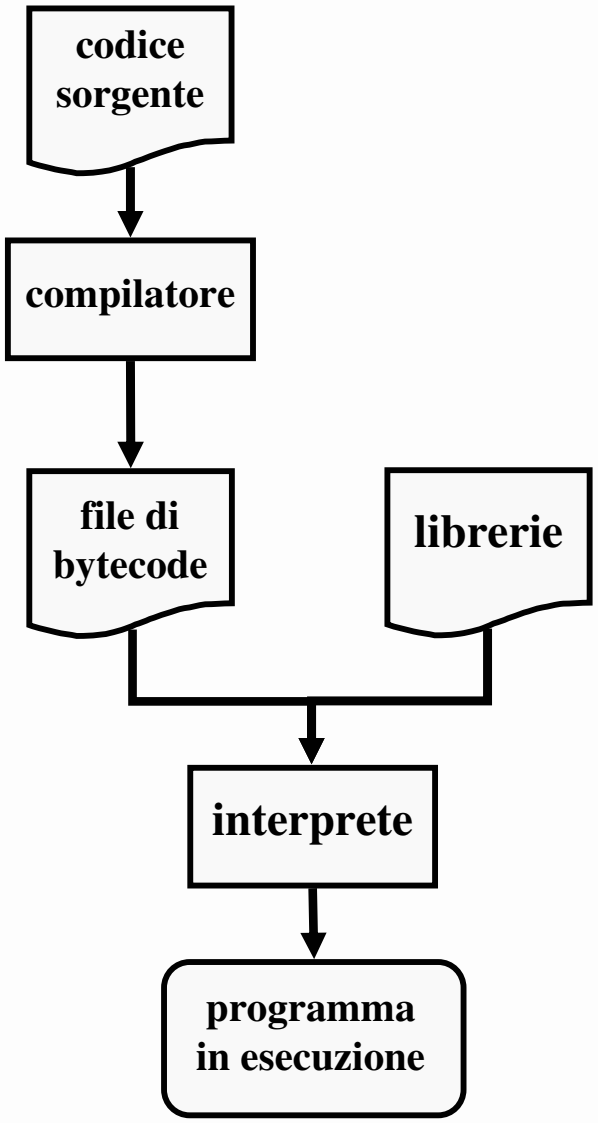
Compilatore e/o interprete

- Per passare dalla scrittura del file sorgente in linguaggio Java all'esecuzione del programma su una particolare CPU, si usano il ***compilatore*** **e** l'***interprete***
- Con la maggior parte degli altri linguaggi di programmazione ad alto livello, invece, si usa soltanto il ***compilatore*** **oppure** soltanto l'***interprete***
  - con ***linguaggi compilati*** come Pascal, C e C++, si usa il ***compilatore*** per creare un file eseguibile, contenente codice macchina, a partire da file sorgenti (con l'eventuale ausilio di un ***caricatore***, *loader*, che interagisce con una libreria)
  - con ***linguaggi interpretati*** come BASIC e PERL, l'***interprete*** traduce "al volo" il file sorgente in codice eseguibile e lo esegue, senza creare un file eseguibile

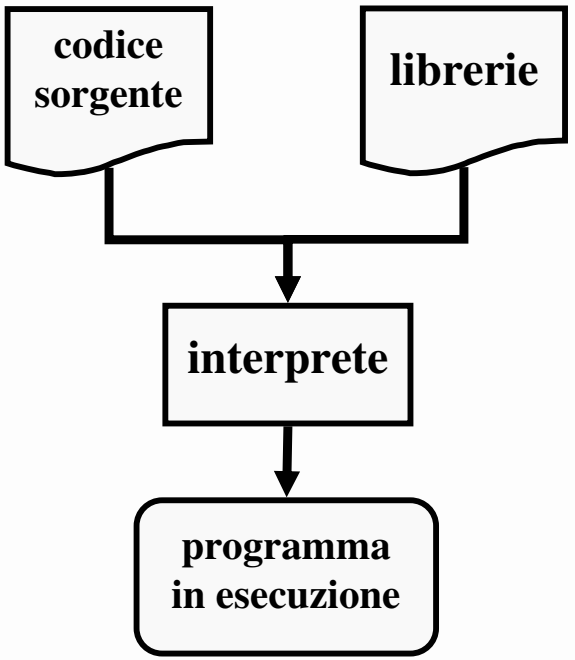
Il processo di programmazione



linguaggi compilati



Linguaggio Java



linguaggi interpretati

Compilatore e/o interprete

- Il fatto che un linguaggio sia compilato o interpretato influisce fortemente su quanto è
  - facile eseguire lo stesso programma su computer aventi diverse CPU (***portabilità***)
  - veloce l'esecuzione di un programma (***efficienza***)
- Entrambi questi aspetti sono molto importanti nella fase di scelta di un linguaggio di programmazione da utilizzare in un progetto
- Il linguaggio Java, da questo punto di vista, è un ***linguaggio misto***, essendo sia compilato sia interpretato, in fasi diverse

Portabilità

- I programmi scritti in un ***linguaggio interpretato*** sono ***portabili***
- I programmi scritti in un ***linguaggio compilato***
  - sono portabili a livello di file sorgente, ma è necessario compilare il programma su ogni diversa CPU
  - non sono portabili a livello di file eseguibile, perché esso contiene codice macchina per una particolare CPU
- ***I programmi scritti in linguaggio Java sono portabili***, oltre che a livello di file sorgente, anche ***ad un livello intermedio***, il livello del ***bytecode***
  - possono essere compilati una sola volta ed eseguiti da ***interpreti diversi*** su diverse CPU
  - ***"compile once, execute everywhere!"***

Efficienza

- I programmi scritti in un ***linguaggio interpretato*** sono ***poco efficienti***
  - l'intero processo di traduzione in linguaggio macchina deve essere svolto ad ogni esecuzione
- I programmi scritti in un ***linguaggio compilato*** sono ***molto efficienti***
  - l'intero processo di traduzione in linguaggio macchina viene svolto prima dell'esecuzione, una volta per tutte
- ***I programmi scritti in linguaggio Java hanno un'efficienza intermedia***
  - parte del processo di traduzione viene svolto una volta per tutte (dal compilatore) e parte viene svolto ad ogni esecuzione (dall'interprete)

Portabilità ed efficienza

- Se si vuole soltanto la ***portabilità***, i ***linguaggi interpretati*** sono la scelta migliore

- Se si vuole soltanto l'***efficienza***, i ***linguaggi compilati*** sono la scelta migliore

- Se si vogliono perseguire ***entrambi gli obiettivi***, come quasi sempre succede, ***il linguaggio Java può essere la scelta vincente***

Grammatiche

- Java è un ***linguaggio formale*** definito da un ***grammatica non contestuale*** (o quasi)
- La grammatica può essere divisa in due parti che descrivono rispettivamente le ***regole lessicali*** e la ***sintassi*** che governano la costruzione di unità compilabili sintatticamente corrette
  - ***sintassi***: studio delle regole che governano il modo in cui le parole si combinano per formare frasi e le frasi per formare periodi.
- Dal punto di vista lessicale un programma Java è composto da ***righe*** (stringhe di caratteri terminate da un *fineriga*) a loro volta composte da unità elementari dette ***token*** (in italiano *lessemi*)
- I token devono essere separati da ***spazi*** e/o da ***commenti*** quando non ci sia un ***separatore*** o un ***operatore*** che marchi la fine del token

Token

- I ***lessemi*** o ***token*** (le unità lessicali con cui si costruisce una unità compilabile) possono essere del tipo:
  - **Parola chiave** (keyword): stringhe di caratteri il cui uso è riservato e ***predefinito*** dal linguaggio Java
    - Esempio ***public*** nella classe Hello
  - **Identificatore** (identifier): una stringa di caratteri alfanumerici con il primo carattere alfabetico (anche '_' e' ammesso come primo carattere), ***definiti*** dal programmatore
    - Esempio ***Hello, System, out, println*** nella classe Hello
  - **Costante** (literal): un valore costante
  - **Separatore**: caratteri di interpunzione
  - **Operatore**: simboli che indicano un'operazione fra variabili e/o costanti

Le parole chiave di Java

abstract	continue	for	new	switch
assert	default	if	package	synchronized
boolean	do	goto	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Identificatori

- Sequenza di caratteri alfabetici e cifre numeriche di lunghezza arbitraria
- Il primo carattere deve essere alfabetico
  - ***Hello*** e' un identificatore valido
  - ***main*** e' un identificatore valido
  - ***7Wonder*** e ***?why*** non sono identificatori validi
- Il primo carattere puo' essere anche il carattere _ (underscore)
  - _nome e' un identificatore valido
- Non si possono adoperare come identificatori
  - le parole chiave di Java
  - le costanti "booleane" ***true*** e ***false***
  - la costante speciale ***null***

Costanti (Literals)

- Numeri interi:
  - ***1, 2, 100, 567*** …
- Numeri in formato a virgola mobile:
  - ***15.17, 1.23e-5*** …
- Costanti booleane:
  - ***false, true***
- Caratteri
  - ***'a' 'B' 'π' '?'***
- Stringhe
  - ***"Hello, World!"***
- Costante speciale
  - ***null***

Separatori e operatori

- I seguenti 9 caratteri sono utilizzati come *separatori* (caratteri di interpunzione)

```
(   )   {   }   [   ]   ;   ,   .
```

- I seguenti 37 caratteri o token sono utilizzati nelle espressioni come *operatori*

```
=   >   <   !   ~   ?   :
==  <=  >=  !=  &&  ||  ++  --
+   -   *   /   &   |   ^   %   <<   >>   >>>
+=  -=  *=  /=  &=  |=  ^=  %=  <<=  >>=  >>>=
```

Tipi di dati fondamentali

Un programma che elabora numeri

```
public class Coins1
{
   public static void main(String[] args)
   {
      int lit = 15000;    // lire italiane
      double euro = 2.35; // euro

      // calcola il valore totale
      double totalEuro = euro + lit / 1936.27;

      // stampa il valore totale
      System.out.print("Valore totale in euro ");
      System.out.println(totalEuro);
   }
}
```

Un programma che elabora numeri

- Questo programma elabora ***due tipi di numeri***
  - ***numeri interi*** per le lire italiane, che non prevedono l'uso di decimi e centesimi e quindi non hanno bisogno di una parte frazionaria
  - ***numeri frazionari*** ("in virgola mobile") per gli euro, che prevedono l'uso di decimi e centesimi e assumono valori con il separatore decimale
- I numeri interi (positivi e negativi) si rappresentano in Java con il tipo di dati **int**
- I numeri in virgola mobile (positivi e negativi, ***a precisione doppia***) si rappresentano in Java con il tipo di dati **double (IEEE 754 doppia precisione)**

Perché usare due tipi di numeri?

- In realtà sarebbe possibile usare numeri in virgola mobile anche per rappresentare i numeri interi, ma ecco due buoni motivi per non farlo
  - **"pratica"**: i numeri interi rappresentati come tipo di dati **int** sono più ***efficienti***, perché ***occupano meno spazio in memoria*** e sono ***elaborati più velocemente***
  - **"filosofia"**: indicando esplicitamente che per le lire italiane usiamo un numero intero, rendiamo ***evidente*** il fatto che non esistono i decimali per le lire italiane
    - ***è importante rendere comprensibili i programmi!***