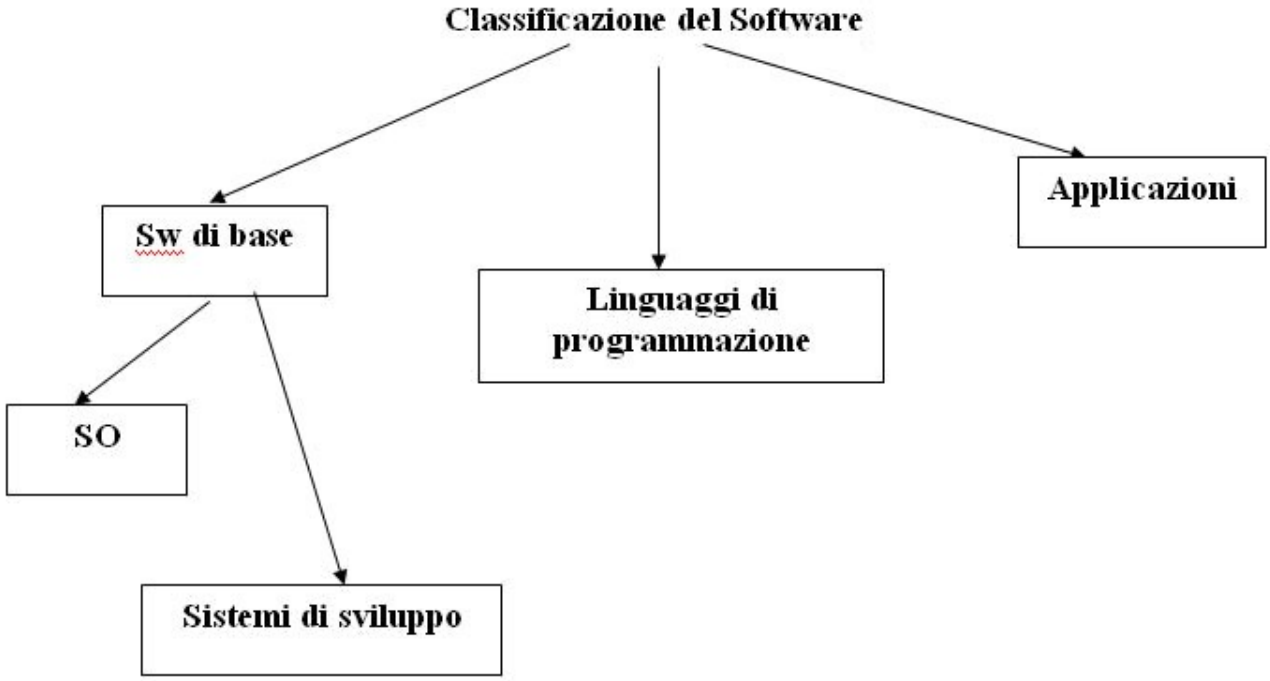


Il **Software** (*programmi*) può essere classificato come:

- di sistema o di base (Sistema Operativo, compilatori, , interpreti[1]…)
- programmi applicativi (editor di testo, fogli elettronici, data base ….)
- linguaggi di programmazione

Classificazioni dei linguaggi di programmazione

Classificare i **linguaggi**[2] **di programmazione** serve per delinearne le caratteristiche e capire se e quando un linguaggio sia idoneo a risolvere un dato problema risparmiando tempo e brutte sorprese in fase di sviluppo. Uno stesso linguaggio può essere inquadrato contemporaneamente in più classificazioni. Senza essere esaustivi[3], si illustrano di seguito le classificazioni più usuali:

- **Classificazione** per **paradigma**

  Nel corso degli anni, a volte per esigenze applicative, a volte per pura ricerca teorica, sono nati approcci e sottoapprocci (detti **paradigmi di programmazione**) diversi alla soluzione del problema di dare al calcolatore gli strumenti per restituire un **risultato corretto** a fronte di un elaborazione di **dati** che conosce o che devono essere inseriti dall'esterno.

---

[1] Tra i livelli di funzionalità di un calcolatore si definisce **software di sviluppo** quell'insieme di programmi che consentono di sviluppare le applicazioni definite dall'utente: editor, compilatori, interpreti e linker-loader cioè programmi capaci di tradurre sequenze di istruzioni formulate in linguaggio di programmazione in sequenze di numeri in un codice comprensibile dal calcolatore

[2] Per una classificazione dei linguaggi di programmazione in termini di **generazioni** vedi on-line.

[3] Meritano un discorso a parte i linguaggi e tecnologie nati con l'avvento di ***Internet***.

Esistono diversi **paradigmi di programmazione** anche se è difficile delineare un confine preciso tra l'uno e l'altro e capire come e quanto alcuni derivino o dipendano da altri.
Semplificando al massimo, è possibile identificare almeno due grandi macrofamiglie corrispondenti a due diversi modi di intendere i meccanismi di risoluzione di un generico problema:

- **paradigma di tipo procedurale** se affrontando i problemi in **modo tradizionale** si seguono le tre fasi seguenti:

  - **Conoscenza** e studio della *realtà* (o contesto) da cui è tratto il problema
  - **Comprensione** del problema ("**cosa**" si vuole)
  - **Ricerca** di una strategia (o idea) risolutiva per il problema ("**come**") e sintesi della procedura risolutiva (*algoritmo* e/o programma) per un esecutore

  Il paradigma **procedurale** prevede l'esecuzione di sequenze di istruzioni; il flusso logico è regolato da strutture che consentono di scegliere un percorso alternativo o di ripetere un'istruzione più volte; il problema è scomposto in una serie di procedure di tipo algoritmico al fine di individuare "**come**" risolverlo.

  Esempi di **linguaggi** per un **approccio *procedurale***:
  - linguaggio imperativo (strutturato ad esempio C, Pascal, …)
  - oop (ad esempio Java, C++, …)
  - concorrente distribuito

- **paradigma di tipo non procedurale** (**dichiarativo**) che prevede invece di:

  - **Specificare la conoscenza** della *realtà* e il problema in un *linguaggio di **specifica***
  - **Affidare all'interprete del linguaggio** la **soluzione**

  Il paradigma **dichiarativo** invece (linguaggi logici, funzionali, di interrogazione per DB relazionali) richiede al programmatore di descrivere le situazioni e le relazioni logiche che legano gli oggetti in gioco ma non di dettagliare le strategie risolutive: è l'interprete del linguaggio, in base alle conoscenze acquisite, a formulare le risposte ai problemi posti.

  Esempi di **linguaggi** per un **approccio** dichiarativo:
  - linguaggi di interrogazione per DB relazionali
  - logico (Prolog)
  - funzionale (Lisp)

NB: il linguaggio C++ accetta un *approccio misto* comportandosi sia come linguaggio ad approccio procedurale sia come linguaggio **OO** (il linguaggio **O**bject **O**riented deriva strettamente dal paradigma procedurale: ne differisce poiché tende a raggruppare le procedure e i dati da esse trattati in ambienti chiusi detti ***oggetti***) permettendo al programmatore di lavorare in modo procedurale per risolvere problemi semplici e passare alla modalità ad oggetti nel caso di elaborazioni di particolare complessità sfruttando quel processo che isola porzioni rilevanti di codice a vantaggio della leggibilità e facilità di debug del codice stesso come nel caso di applicazioni che richiedono scrittura di algoritmi complessi e particolarmente voluminosi.

- **Classificazione** per livello

  - linguaggi di ***basso*** livello sono quelli legati ai dispositivi hardware (l'**assembler** specifico del microprocessore in uso) che permettono un dialogo diretto con la CPU, la memoria ed i periferici sfruttandone al massimo le potenzialità.

    Vantaggi: gestione ottimizzata delle risorse e massima efficienza con codice meno pesante (cioè minor occupazione di memoria)
    Svantaggi: necessaria perfetta conoscenza dei dispositivi e dell'assembler tipico del microprocessore in uso; codice non portabile su piattaforme (hw/sw) diverse.

  - linguaggi di ***alto*** livello sono quelli nati per gestire strutture dati astratte e teoriche permettendo al programmatore di concentrarsi solo sugli aspetti algoritmici senza gestire fisicamente i dispositivi

    Vantaggi: programmazione orientata all'uso di moduli (insieme di *funzioni* detto *libreria standard)* che consentono riuso e portabilità del codice
    Svantaggi: gestione flessibile ma non ottimizzata delle risorse ed efficienza che dipende dall'implementazione della *libreria standard*

  NB: il linguaggio C possiede le caratteristiche di *entrambi i livelli* infatti nasce con le seguenti specifiche: essere contemporaneamente un linguaggio per scrivere procedure complesse ed un codice decisamente ottimizzato; un linguaggio per affrontare in modo efficace problematiche molto diverse. Viene pertanto considerato di *medio* livello.

- **Classificazione** per **formato del codice**

  - linguaggi compilati

    Si succedono le seguenti **fasi nella realizzazione di un programma**:

    1. la prima fase consiste nell'**editing** cioè la scrittura del programma ***sorgente***: testo ASCII

    2. la seconda nella **compilazione** cioè la *traduzione* con *controllo sintattico* producendo un file ***oggetto*** (con estensione .obj) in linguaggio esadecimale interpretabile da una particolare CPU

    3. la terza è la fase di **building** (costruzione) o ***linker***: collegamento con librerie e mappatura degli indirizzi con produzione del file ***eseguibile*** (con estensione .exe) per caricamento (***loader***) in RAM ad un determinato indirizzo fisico

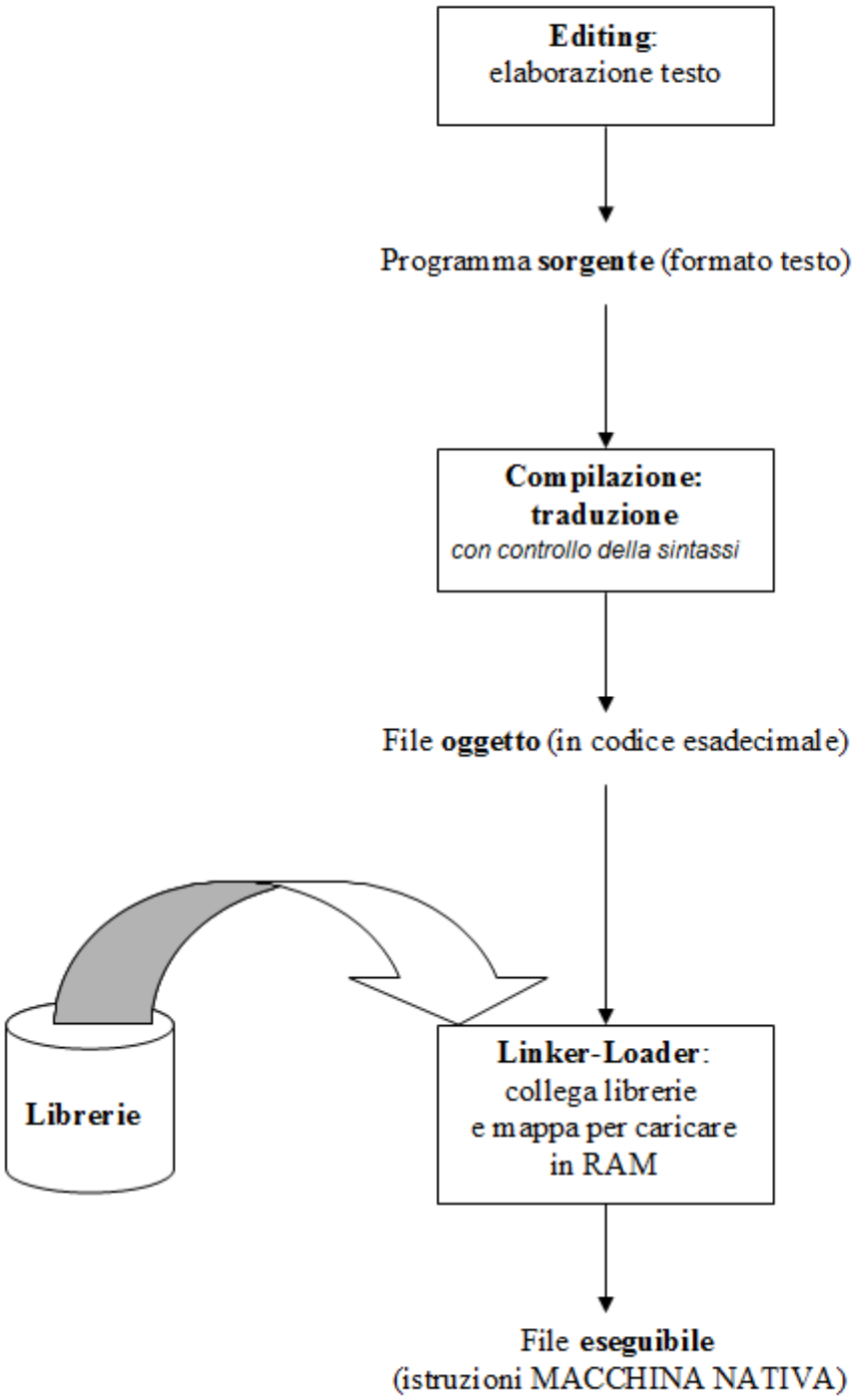
    4. esecuzione

  - linguaggi interpretati come il Basic dove la ***traduzione*** avviene al momento dell'esecuzione o ***run time***

  - linguaggi semicompilati (ad esempio Java linguaggio interpiattaforma) fase di precompilazione e produzione di ***bytecode*** compatibile con architetture diverse.

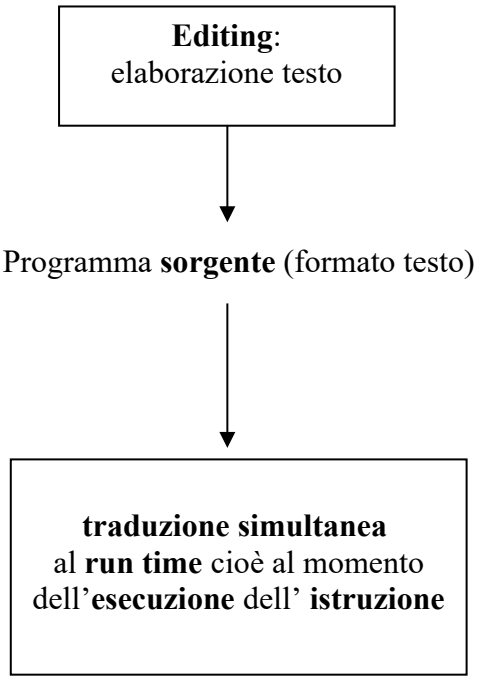
Fasi nella realizzazione di un programma
(nel contesto della produzione di SW per risolvere problemi)

Linguaggi **compilati**



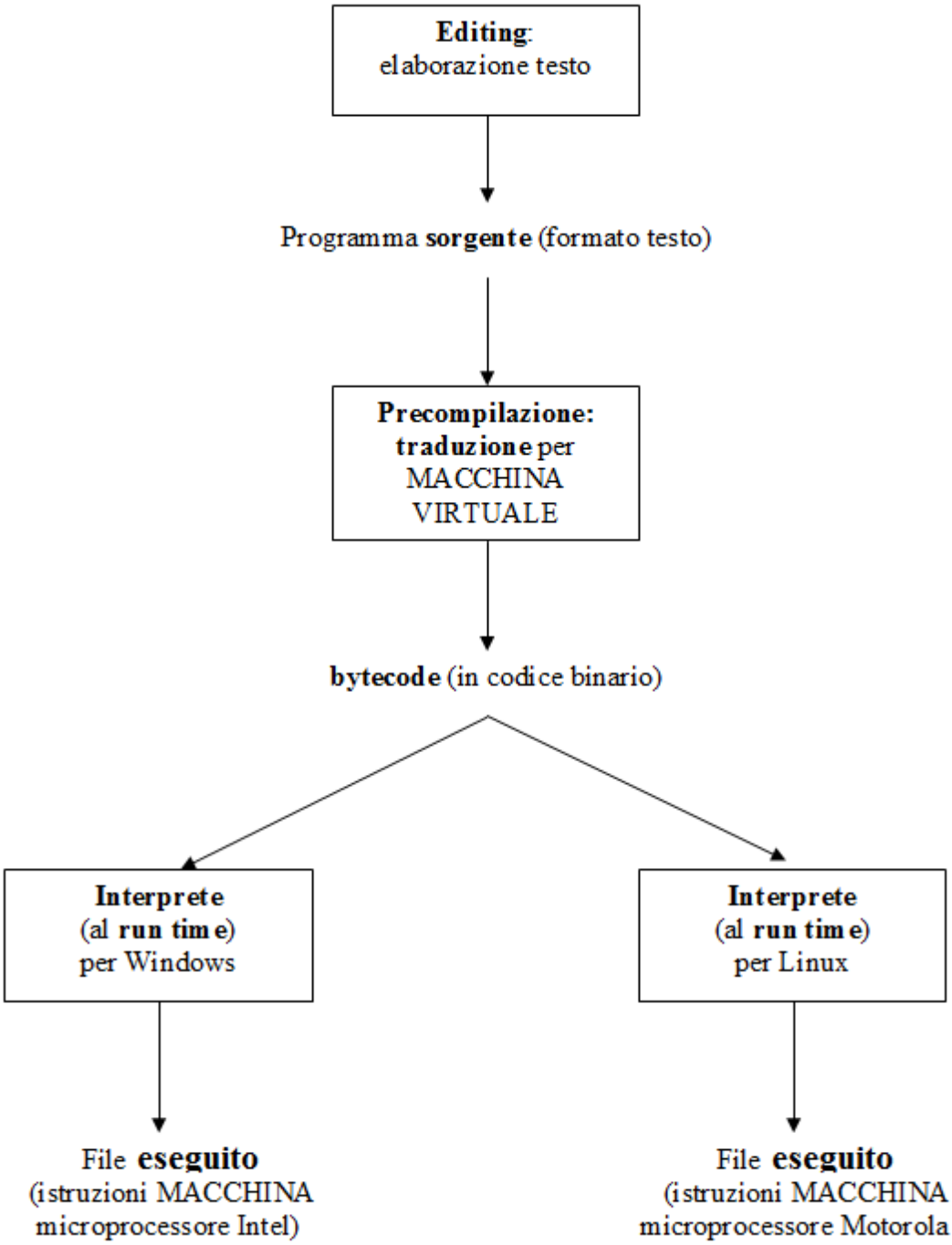
Fasi nella realizzazione di un programma
(nel contesto della produzione di SW per risolvere problemi)

Linguaggi **interpretati**



Fasi nella realizzazione di un programma
(nel contesto della produzione di SW per risolvere problemi)

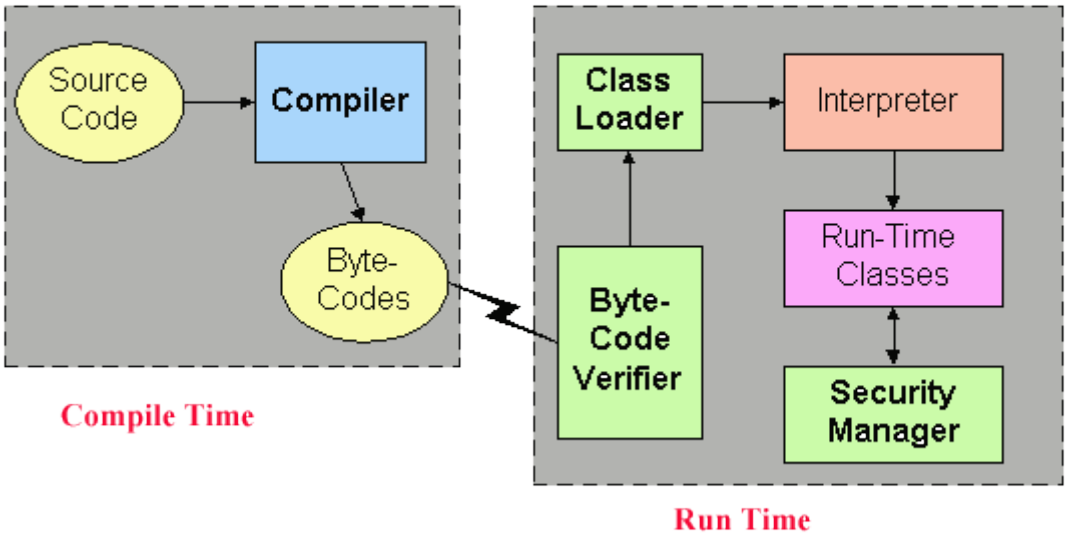
Linguaggio **semi-compilato**, **interpiattaforma**



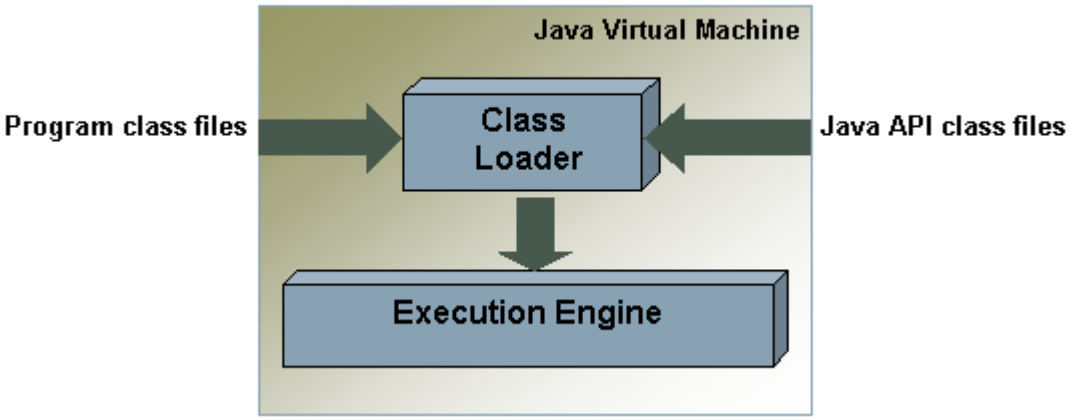
Bytecodes: istruzioni per macchina virtuale indipendenti dalla macchina reale (microprocessore Intel piuttosto che Motorola) e indipendenti dal SO (*cross platform*).

Linguaggio **Java** interpiattaforma:

- particolare attenzione alla **sicurezza**



- ricchezza di procedure disponibili (***A**pplication **P**rogramming **I**nterface*) che permettono di evitare ai programmatori di scrivere tutte le funzioni dal nulla



JVM (*Java Virtual Machine*): macchina virtuale Java, il componente della piattaforma Java che ***traduce al volo*** ed esegue i programmi tradotti in bytecode.

La disponibilità di implementazioni della macchina virtuale Java per diversi ambienti operativi è la chiave della ***portabilità*** di Java, proclamata nello slogan ***write once, run everywhere*** ("scrivi una volta, esegui dappertutto"). La macchina virtuale realizza infatti un ambiente di esecuzione omogeneo, che nasconde al software Java (e quindi al programmatore) qualsiasi specificità del sistema operativo sottostante.