



UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA
DIPARTIMENTO DI ECCELLENZA

Università degli Studi di Salerno

Dipartimento di Informatica

Programmazione ad Oggetti

a.a. 2023-2024

Java Introduzione

Docente: Prof. Massimo Ficco

E-mail: *mficco@unisa.it*

Il linguaggio C++

C++ è un linguaggio di programmazione **general-purpose** che supporta:

- la *programmazione procedurale* (è un C “migliore”)
- la *programmazione orientata agli oggetti*
- la *programmazione generica*

C++ è quindi un **linguaggio ibrido**, nel senso che supporta più paradigmi di programmazione



Java: un po' di storia

Java è un linguaggio di programmazione sviluppato da James Gosling alla **Sun Microsystems** nel 1995

E' nato come evoluzione del linguaggio C++

Offre meccanismi per lo sviluppo di applicazioni distribuite su rete e facilmente integrabili in applicazioni basate sul www e su browser

Java viene distribuito con una vasta libreria di software che si può usare nello sviluppo dei programmi e che consentono di usare grafica, di comunicare in rete, di interrogare basi di dati ecc..



Caratteristiche di Java

- Semplice e orientato agli oggetti
- Interpretato
- Architetaturalmente neutro e portabile
- Robusto
- Distribuito
- Sicuro
- Dinamico
- Concorrente (multithread)



SEMPLICE

Sintassi simile a C e C++ (facile da imparare), ma elimina i costrutti più "pericolosi" di C e C++

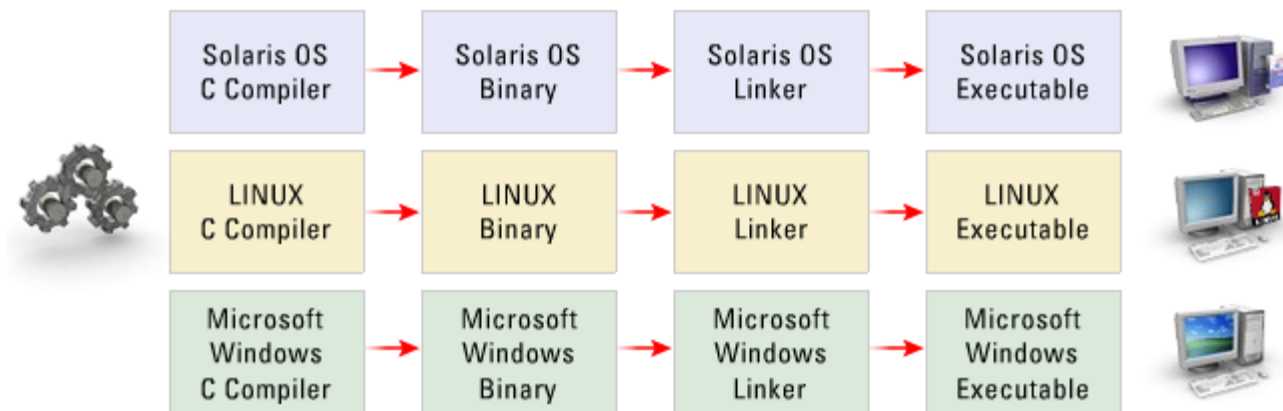
In particolare, evita:

- aritmetica dei puntatori
- (de)allocazione esplicita della memoria
- Aliasing (più di un nome è utilizzato per lo stesso oggetto)
- strutture (struct)
- definizione di tipi (typedef)
- preprocessore (#define)
- Parallelismo (problemi di tempistica difficili da verificare)
- Interrupts (forza il trasferimento del controllo a una sezione di codice)
- Unbounded arrays (A run-time il sistema non controlla se le assegnazioni sono corrette – buffer-overflow)

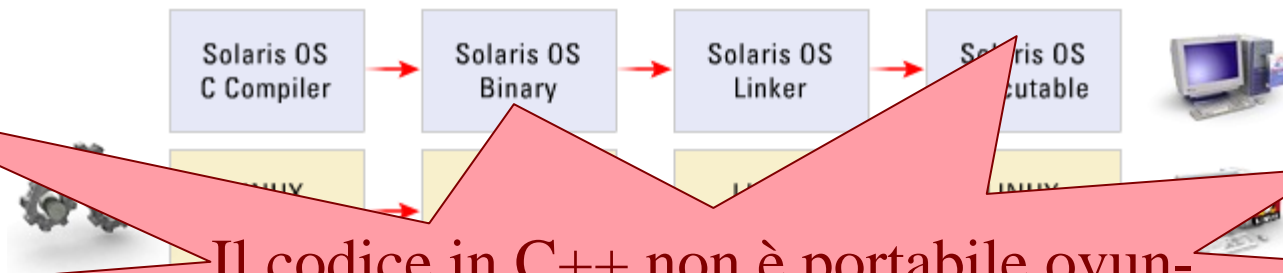
Aggiunge ***garbage collection*** automatica



C e C++: Platform Dependent



C e C++: Platform Dependent

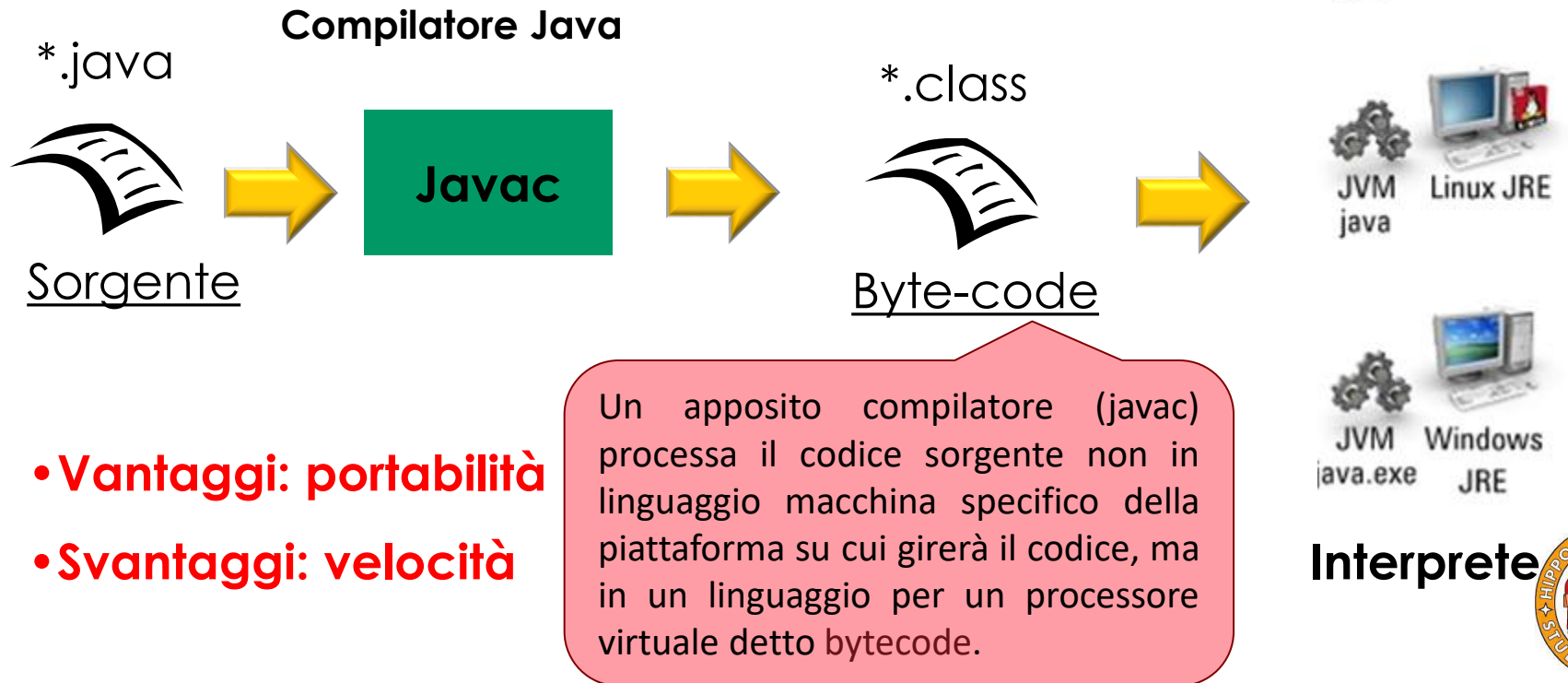


Il codice in C++ non è portabile ovunque, ma va compilato su ogni piattaforma dove essere eseguito!!



Platform Independent: interprete Java

Il compilatore produce un codice di tipo intermedio per una
“Java Virtual Machine” (“byte-code”) ...
... che viene interpretato



- **Vantaggi: portabilità**
- **Svantaggi: velocità**

BYTECODE: ESEMPIO

```
void spin ( ) {  
    int i;  
    for (i = 0; i < 100; i++) {  
        ;  
    }  
}
```

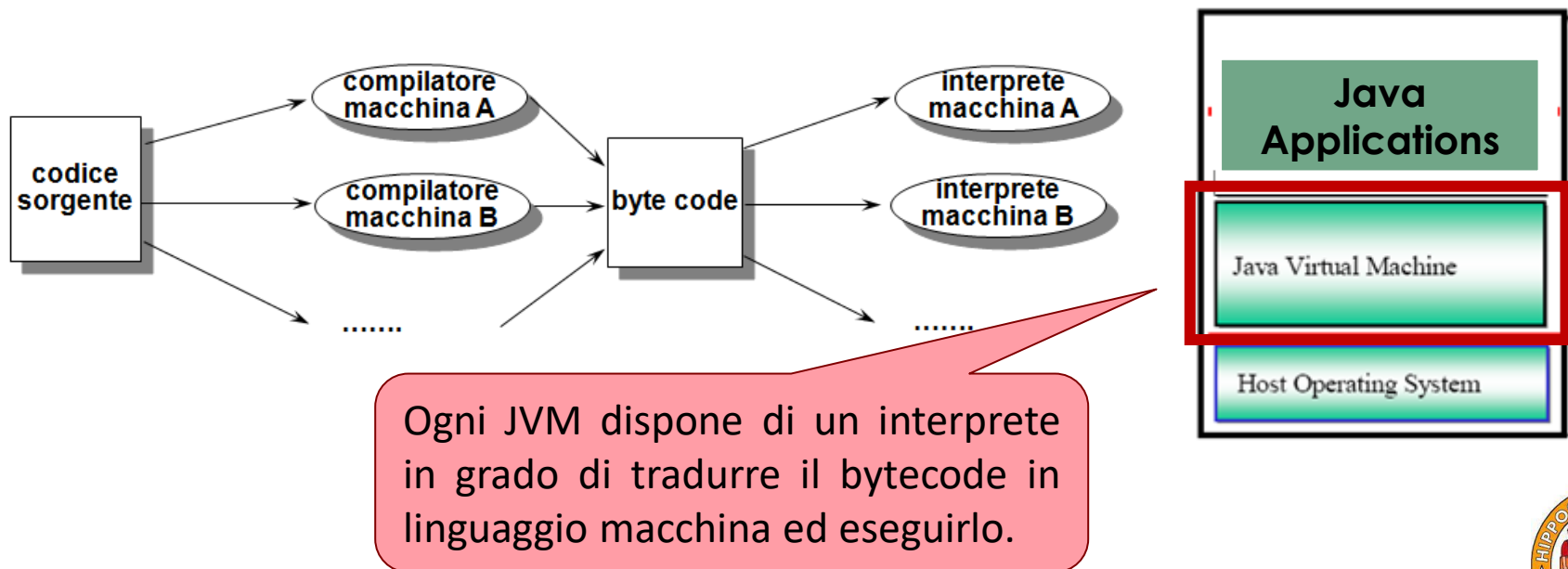
| | | |
|----|-------------|---------------------------------|
| 0 | iconst_0 | // push int constant 0 |
| 1 | istore_1 | // store into local 1 (i=0) |
| 2 | goto 8 | // first time, don't increment |
| 5 | iinc 1 1 | // increment local i by 1 (i++) |
| 8 | iload_1 | // push local 1 (i) |
| 9 | bipush 100 | // push int constant (100) |
| 11 | if_icmplt 5 | // compare, loop if < (i<100) |
| 14 | return | // return void when done |



ARCHITETTURALMENTE NEUTRO

Il byte-code è indipendente dall'architettura hardware
(ANDF: Architecture Neutral Distribution Format)

Pertanto, un programma bytecode può essere eseguito su qualsiasi sistema su cui giri un ambiente run-time Java



Compilatori ed interpreti

Per sviluppare un programma occorrono strumenti:

- Editor
- Compilatori
- Interpreti

L' *editor* serve per digitare il programma (*source code*) e salvarlo in un file

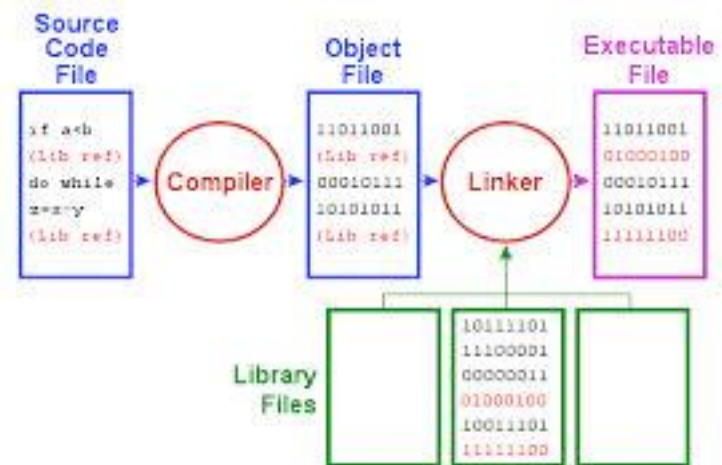
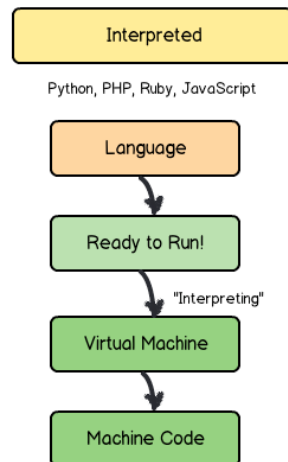
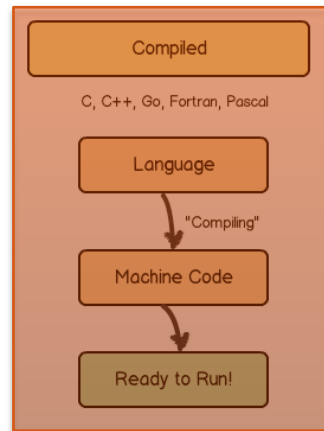
Una volta memorizzato il codice sorgente deve essere tradotto in linguaggio macchina e ciò può essere fatto in vari modi:

- il ***compilatore*** traduce il codice sorgente direttamente in linguaggio macchina
- l'***interprete*** integra la traduzione con l'esecuzione



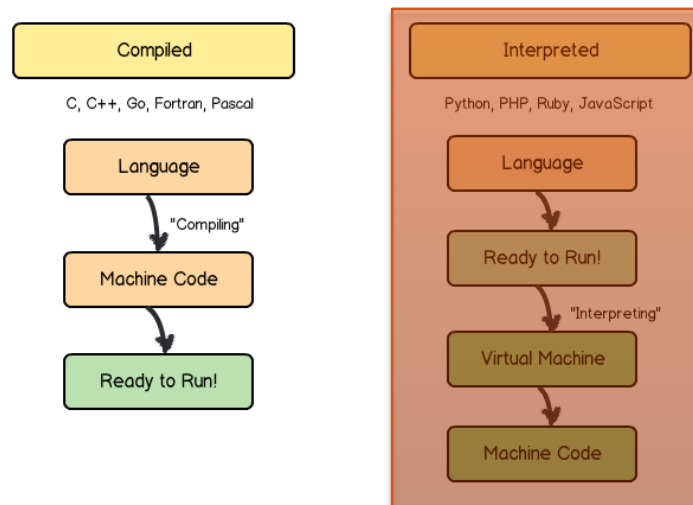
Compilazione

La compilazione consiste nel tradurre l'intero programma in linguaggio di alto livello prima di poterlo eseguire. È possibile che un programma sia composto da vari diversi programmi sorgenti, che possono essere compilati indipendentemente e poi collegati tra loro per ottenere il programma finale. Per tanto in aggiunta al compilatore e al loader troviamo un linker per effettuare il collegamento.



Interpretazione

- L'interpretazione può essere considerata come la trasposizione ad alto livello del ciclo fetch-execute effettuata dalla CPU nel modello della macchina di von Neumann:
 - Si considera una istruzione per volta nel codice sorgente espresso in un linguaggio di programmazione di alto livello;
 - Si traduce l'istruzione considerata in una o più istruzioni in linguaggio macchina, che vengono immediatamente eseguite;
 - Si passa a considerare l'istruzione successiva.



Confronto

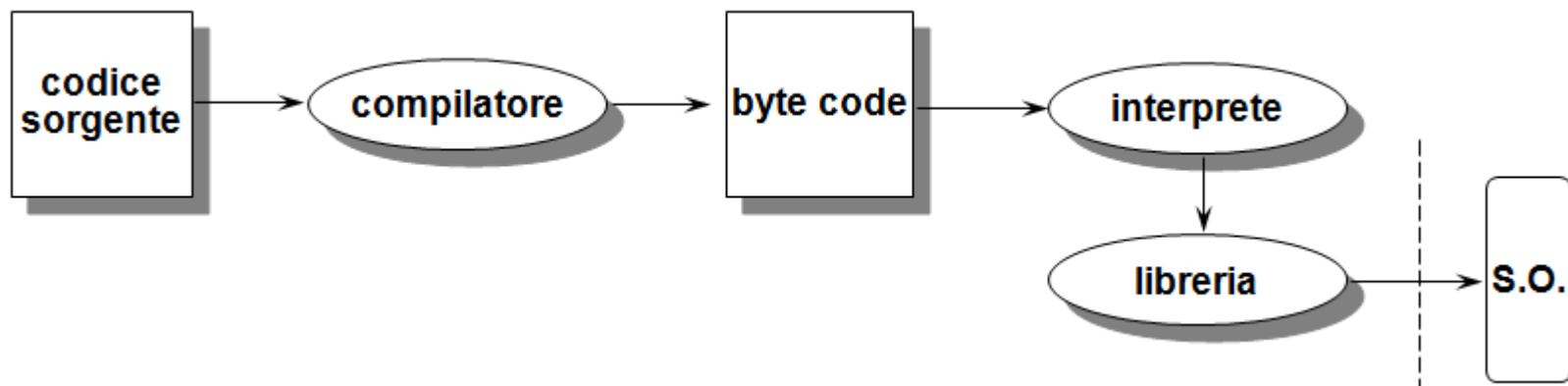
- **L'approccio interpretato** comporta una minore efficienza a tempo di esecuzione dato che richiede più memoria ed è meno veloce per l'overhead dell'interprete. Durante l'esecuzione del programma l'interprete deve analizzare le istruzioni per verificare la correttezza, identificare le operazioni da effettuare ed eseguirle.
- L'interpretazione consente di determinare più facilmente la cause di errori a tempo di esecuzione, dato che spesso con la compilazione si perdono le relazioni esistenti tra codice sorgente e quello eseguibile.
- La compilazione ha il vantaggio di poter effettuare delle ottimizzazioni avendo disponibile la visione completa del programma in codice sorgente così da migliorarne le prestazioni.



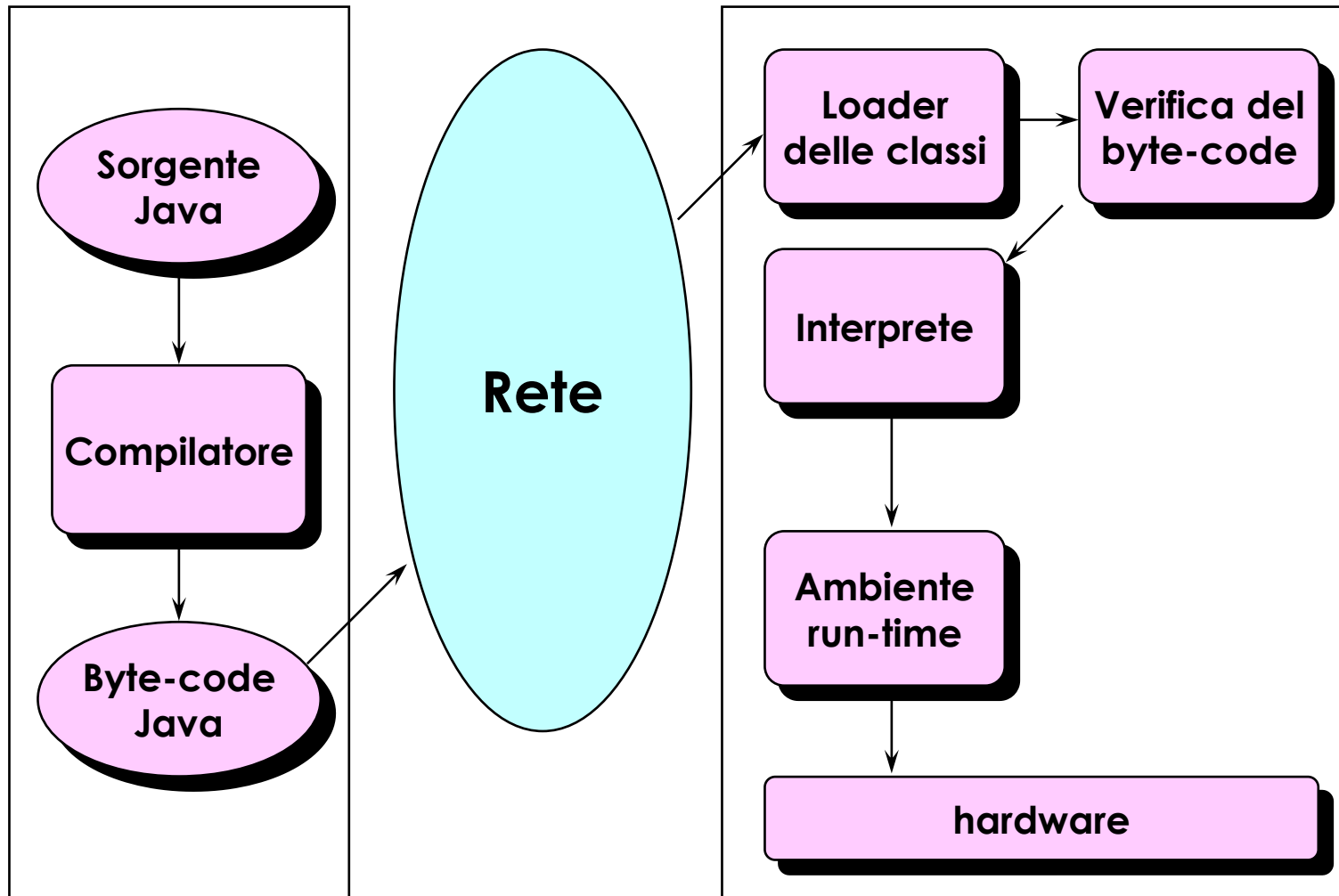
PORTABILE

Il sistema Java (compilatore + interprete + librerie run-time) è facilmente portabile su piattaforme diverse

- il compilatore Java è scritto in Java
- l'ambiente run-time è scritto in ANSI C con interfacce standard (POSIX) verso il sistema operativo
- nessuna “implementation dependency”



COMPILE-LOAD-RUN



DINAMICO

Il codice è eseguibile anche in assenza di alcuni moduli:
... le classi necessarie per la esecuzione di un programma
Java possono essere caricate e collegate dinamicamente
quando servono

Esempio: nuove release di moduli caricabili
automaticamente dalla rete quando servono



DISTRIBUITO

- Pensato per essere eseguito in rete
- L'ambiente run-time incorpora funzioni di rete (sia di basso livello: TCP/IP, che di alto livello: HTTP, ...)
- La rete è facilmente accessibile (come i file locali)



SICURO

L'ambiente di esecuzione si protegge da bytecode potenzialmente “ostile”

Esempi:

- il bytecode viene verificato prima dell'interpretazione (“theorem prover”), in modo da essere certi di alcune sue caratteristiche
- gli indirizzamenti alla memoria nel bytecode sono risolti sotto il controllo dell'interprete



Robusto

Controlli estensivi a compile-time e a run-time, per rilevare gli errori quanto prima possibile (es.: type checking)

Per questo, le caratteristiche insicure di C e C++ sono rimosse:

- Nessuna gestione esplicita dei puntatori (no aritmetica dei puntatori, no malloc e free esplicite, ...)
- Gestione della memoria con garbage collection
- Array e stringhe “veri”
- Verifica del byte-code a load-time



CONCORRENTE

Multithreading parte integrante del linguaggio:

- Applicazioni interattive più facili a scriversi
- Migliore "reattività" (anche se non real-time)

Esempio: caricamento asincrono di immagini nei browser di rete riduce i tempi di attesa



RICCO

La Standard Library Java contiene una ricca collezione di classi e di metodi preconfezionati:

- Language support
- Utilities
- Input/output
- Networking
- Abstract Window Toolkit (AWT)



Java Technology Product Groups



Per usare Java2 Platform



Java 2 Platform, Standard Edition (J2SE) JDK Components:

- Java runtime environment (JRE)
 - Java Virtual Machine (JVM) `java.exe`
 - Java Class Libraries
- Java compiler - `javac.exe`
- Class library documentation (downloaded separately)
- Additional utilities
- Program examples



Cosa serve procurarsi

Java Development Kit

- Comprende jdk e jre

Java doc: un manuale html di tutte le funzioni e le classi java

Un **editor** avanzato (all'inizio useremo notepad e prompt a linea di comando)

- Eclipse
- JCreator
- JBuilder
- ...

Sistema operativo: Qualunque

