Programmare = … ?

- In passato, “occuparsi di informatica” era sinonimo di “programmare computer”
  - attività poco stimolante, *atto finale* di un processo dove le fasi creative - analisi e progetto - sono *già avvenute*
  - fa pensare alla costruzione di *algoritmi* di medio/piccola dimensione, tipicamente già noti e descritti in letteratura
- Anni ’80 → “crisi del software”
  - costi di progetto, sviluppo, testing e manutenzione del software *preponderanti* su quelli dell’hardware e dei sistemi operativi
  - insufficienza delle metodologie e degli strumenti *sviluppati per una visione “algoritmica” dell’informatica* (programmazione *in-the-small)* rispetto alle difficoltà di progettazione, sviluppo e manutenzione di *sistemi software complessi.*

Programmare... *sistemi software*

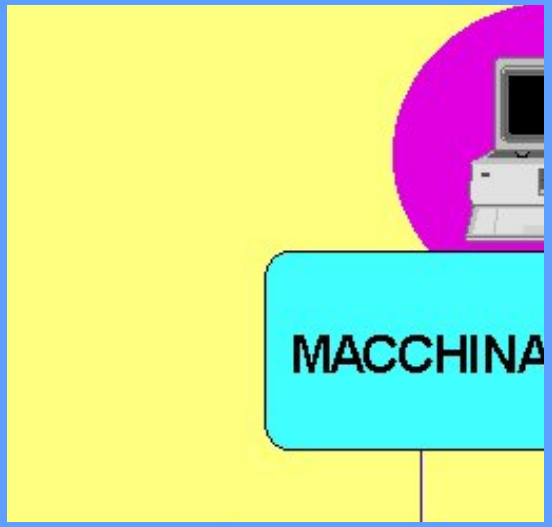
- Principio di fondo: *programmare un elaboratore* per risolvere un dato problema è attività assai diversa dal *codificare.*
  - la scrittura di istruzioni (eseguibili a un qualche livello) deve costituire *l'ultimo passo* di una ben più ampia fase di *progetto*
  - tale capacità di progetto richiede fondamenti teorici, padronanza degli strumenti disponibili, capacità di analisi nello spazio dei problemi, capacità di ragionamento sul lavoro svolto
- Dunque, oggi **programmare un elaboratore = *progettare e costruire sistemi software***

Linguaggio e Progetto

- Poiché l'elaboratore è una macchina capace di simulare il comportamento di ogni altra, il progettista di software è una sorta di *"costruttore di mondi"*
  - l'organizzazione interna e le leggi di funzionamento di questi "mondi" possono essere *anche molto diverse* da quelle del sistema che ne rende possibile e ne supporta la costruzione
  - occorre comunque garantire *consistenza interna*, proprietà strutturali che consentano *evoluzione e modifica*, proprietà comportamentali che garantiscano *efficienza e corretto utilizzo*
- *Compito essenziale* del linguaggio di programmazione è **rendere tali attività guidate dal pensiero**, e non incanalate sui (rigidi) binari dell'hardware.

Linguaggio di alto livello

Un linguaggio di alto livello si basa su una *macchina virtuale* le cui "mosse" *non sono quelle della macchina hardware.*

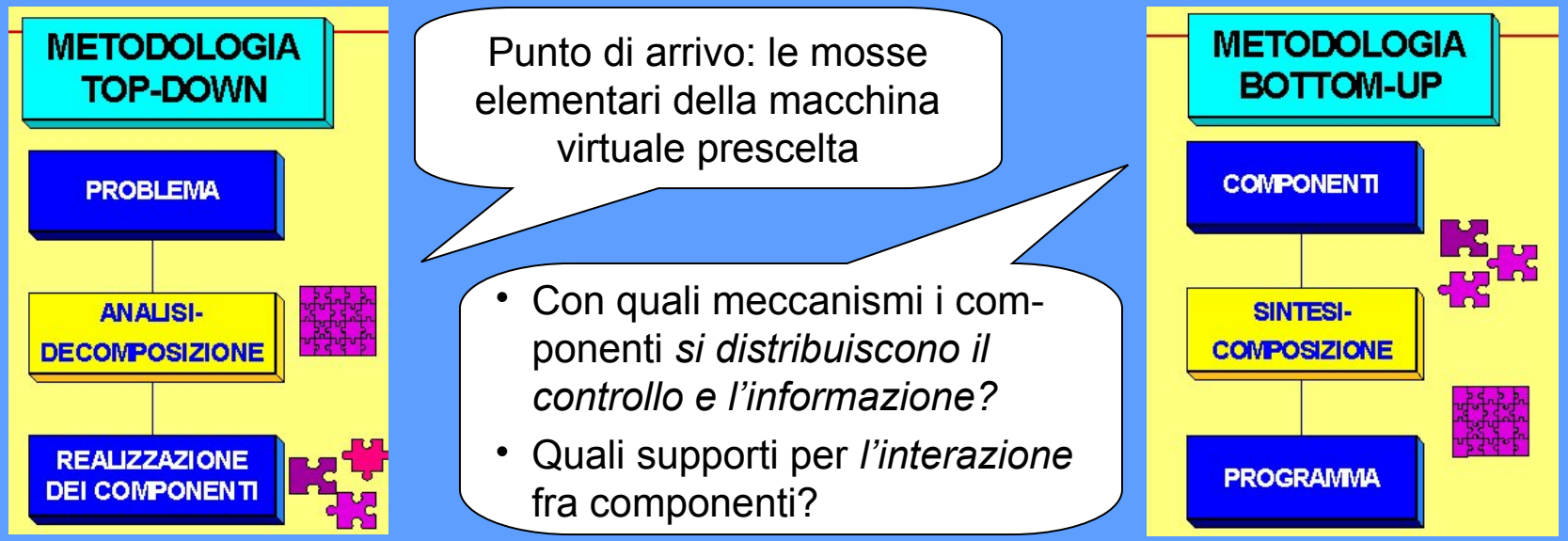


Il ruolo del linguaggio

- La ***capacità espressiva*** di un linguaggio ne fa un **potente suggeritore di concetti e metodi di soluzione** appropriati per un ampio spettro di problemi applicativi.
- Per apprezzare il ***ruolo*** del linguaggio di programmazione nella costruzione del software si deve riflettere su *come avvengono progetto e realizzazione* di un sistema.
  - **Modello a cascata**
  - **Metodologie top-down e bottom-up**

Processo di progettazione e realizzazione

- **Modello a cascata**
  - enunciazione del problema, analisi del problema, progetto
  - implementazione del progetto (nel linguaggio di prog. scelto)
  - testing
- **Due metodologie (non esclusive)**



Il problema di fondo

È possibile pensare la soluzione di un problema *in modo del tutto indipendente* dal linguaggio scelto per l'implementazione?

Astrazione e Linguaggio

- Risolvere coscientemente un problema implica sempre ***creare un modello*** della realtà.
- Questa attività si basa su un ***processo di astrazione*** mirato a cogliere gli ***aspetti essenziali*** del problema.
- Un algoritmo o un sistema software è il ***punto di arrivo del processo di astrazione*** sul dominio del problema.

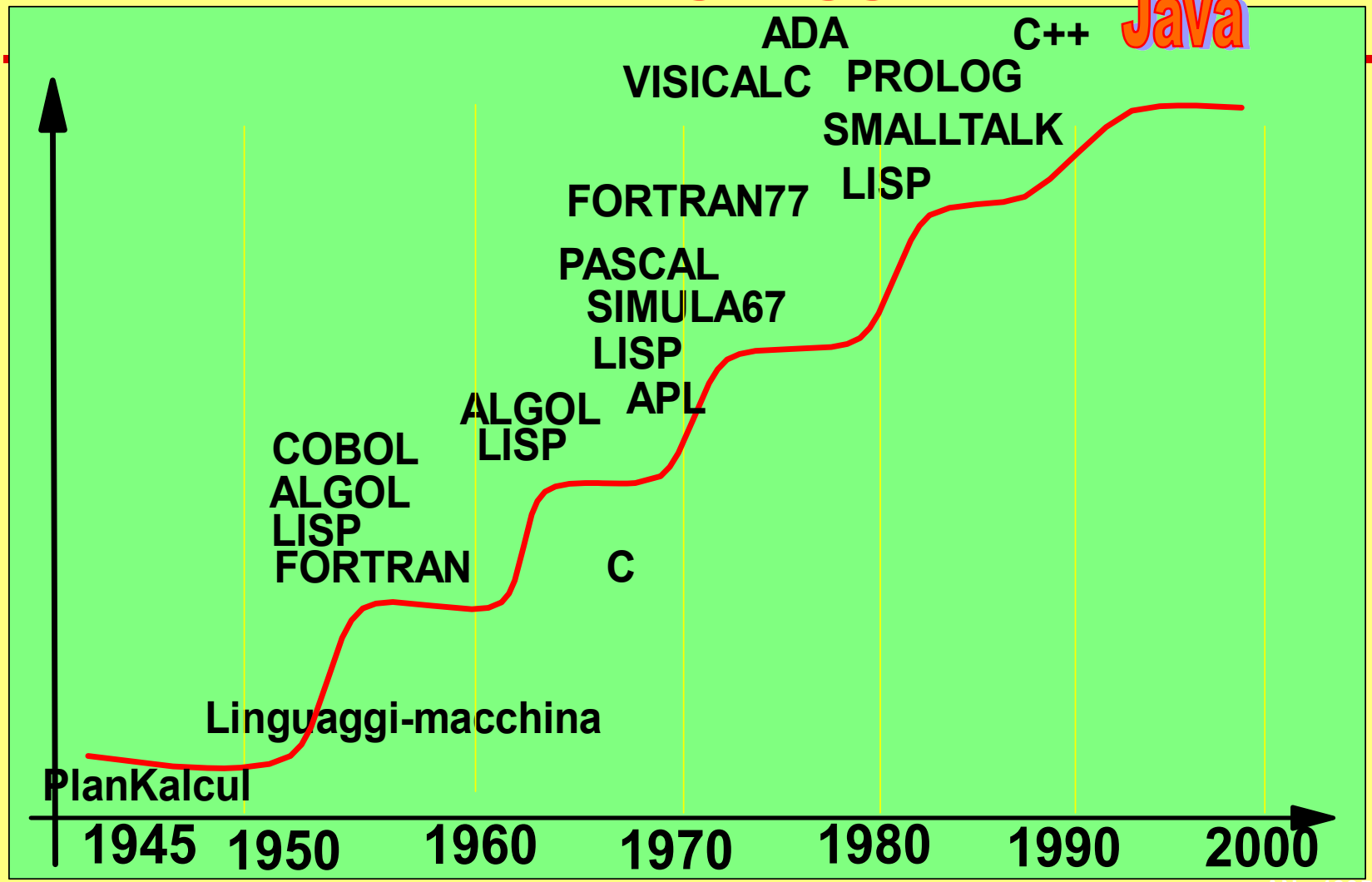
Metafore, Concetti e Linguaggio

- Spesso il progettista segue, magari incoscientemente, **metafore e concetti** che **derivano direttamente** dal linguaggio di implementazione.
- La soluzione viene così **pensata** in termini dello ***"spazio concettuale" di uno specifico linguaggio*** di program-mazione (o di una categoria di linguaggi)
  - ad esempio, se il linguaggio di implementazione è il Pascal, difficilmente la soluzione sarà concepita usando concetti *estranei* al Pascal.
  - usando il C, si può essere "portati" a ragionare partendo dai concetti e dalle "mosse elementari" fornite dal C.

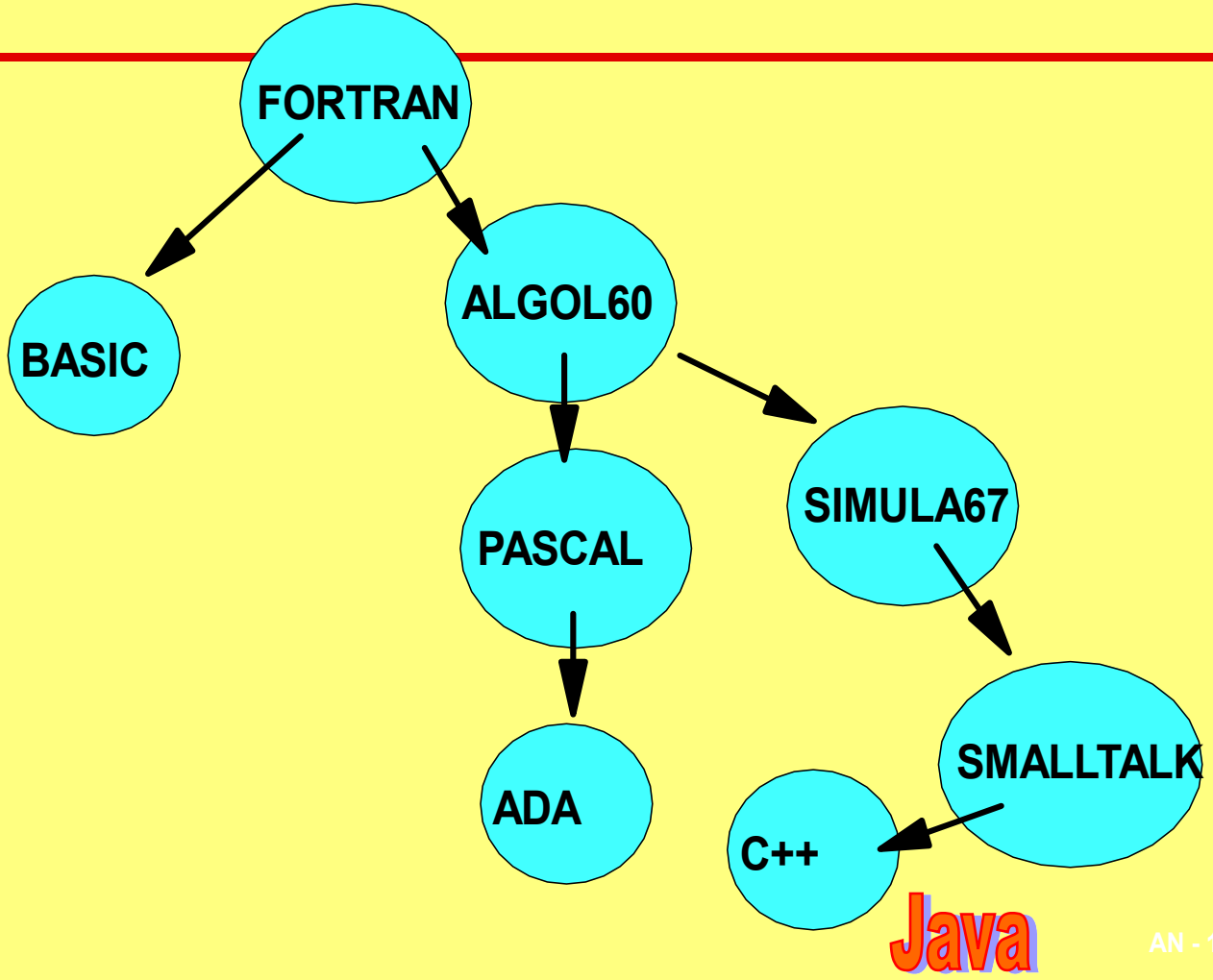
Spazio concettuale di un linguaggio

- Ogni linguaggio introduce, oltre alle regole sintattiche e semantiche, anche un *insieme di metafore e concetti: è* lo ***spazio concettuale*** del linguaggio.
- Metafore e concetti risultano in pratica **determinanti** sia per **l'impostazione di soluzioni** da esprimere in quel linguaggio, sia per **l'organizzazione strutturale** dei sistemi.
  - alcuni elementi sono correlati all'ambiente che il linguaggio suppone preesistente (file, processi, interfacce grafiche…)
  - altri sono invece introdotti dal linguaggio stesso (funzioni, procedure, etc)
- **Il diverso spazio concettuale di ogni linguaggio è la ragione di fondo per l'esistenza di tanti linguaggi**
  - un linguaggio può essere più adatto di un altro come strumento di lavoro o per problemi generici, o per uno specifico settore.

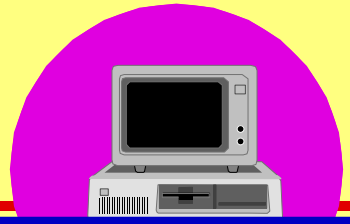
Evoluzione dei linguaggi



Evoluzione dei linguaggi



Linguaggi di alto livello



Barriera di astrazione

Fortran

Cobol

Basic

Pascal

C

Modula-2

Algol

Ada

Lisp

Smalltalk

Simula67

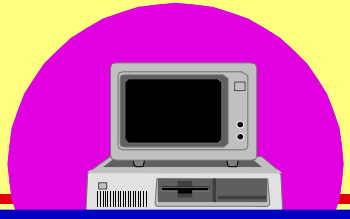
Scheme

C++

Prolog

ML

Linguaggi di alto livello



Barriera di astrazione

Imperativi

C

Pascal

Ada

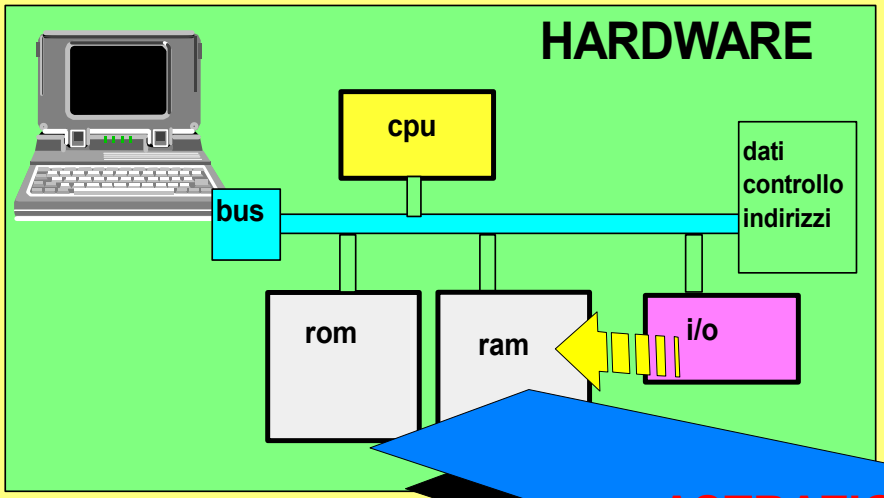
Lisp

A oggetti

Logici

Funzionali

Le sorgenti dei linguaggi di programmazione



MACCHINA
DI
VON NEUMANN

ASTRAZIONE

LINGUAGGI
IMPERATIVI

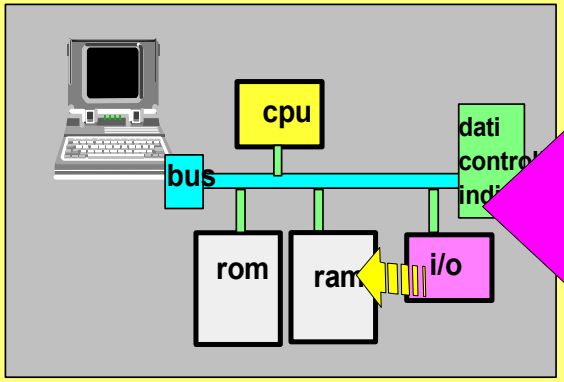
Le sorgenti dei linguaggi di programmazione

ASTRAZIONI
LOGICO-MATEMATICHE

FUNZIONI

LOGICA DEL 1 ORDINE

CONCRETIZZAZIONE



Linguaggi Logici

Linguaggi Funzionali

AN - 1995

Stili di programmazione

- Un **automa** concepito secondo ***uno qualunque*** di questi **formalismi** (e quindi uno stesso elaboratore fisico) può supportare **linguaggi** ispirati a *un* ***qualsiasi modello*** *computazionale.*
- Ogni modello computazionale promuove però uno specifico ***stile*** di programmazione, che risulta caratterizzante per i linguaggi di tale famiglia.
- L'adozione di un dato stile ha **profonde conseguenze** sulle ***proprietà dei programmi***, sulla ***metodologia di soluzione*** dei problemi, sulle ***caratteristiche dei sistemi*** **software**
  - struttura, efficienza, modificabilità, comprensibilità, manutenibilità, ...

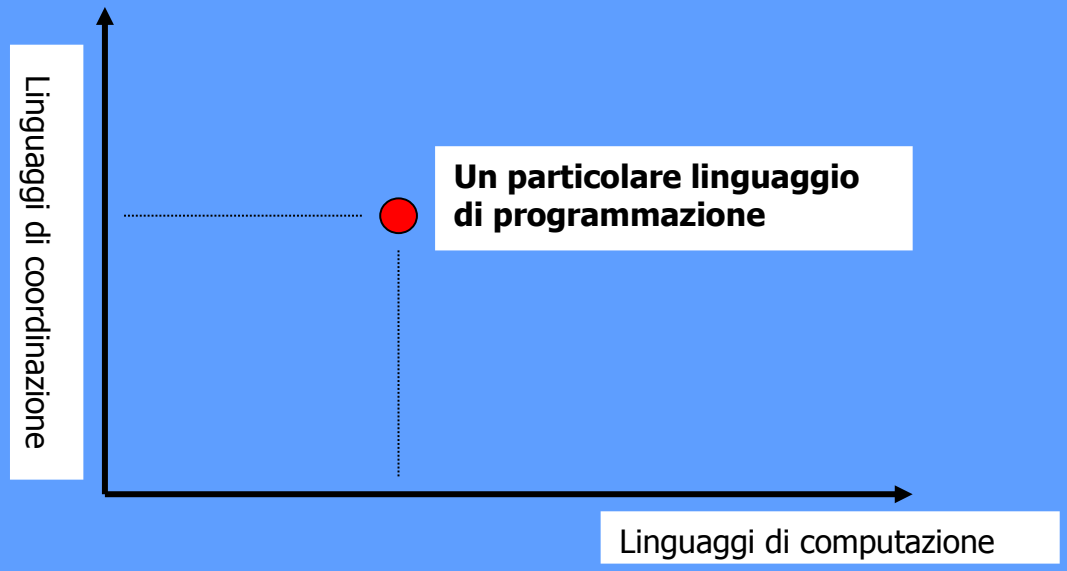
Linguaggi a oggetti

- I linguaggi *a oggetti* sono oggi usati in tutte le fasi di svilup-po dei sistemi software (analisi, progetto, implementazione)
- Promuovono uno **spazio concettuale** che integra in modo "fluido" le fasi sopra indicate
- Utilizzano / integrano le metodologie top-down e bottom-up
- Aiutano a integrare computazione e interazione
- Aiutano ad affrontare il problema dei sistemi *"legacy"*
- Sono fra le sorgenti d'ispirazione del concetto di *componente software*

UN ALTRO ASPETTO
Computazione vs. Interazione

Un *linguaggio di programmazione* è sempre costituito da *due dimensioni (e componenti) ortogonali*:

- **il linguaggio di *Computazione* per *calcolare***
- **il linguaggio di *Coordinazione* per *esprimere e governare l'interazione.***



UN ALTRO ASPETTO
Computazione vs. Interazione

Il *linguaggio di programmazione C* comprende infatti:

- il linguaggio di *computazione C* per calcolare
- un linguaggio di *coordinazione* sotto forma di *librerie standard* (`stdio.h`), allegate al linguaggio *ma non parte di esso,* per esprimere e governare l'interazione.

Si potrebbe cambiare il linguaggio di coordinazione (diverse librerie standard)… MA IL C RESTEREBBE SEMPRE C, con le stesse istruzioni!

Sviluppo di sistemi software

- Nello sviluppo di un ***sistema software*** il punto
  - di partenza, nel caso di metodologia bottom-up
  - di arrivo, nel caso di metodologia top-down

  è tipicamente ***un insieme di sottosistemi*** (fisici o logici)
  - ognuno può essere impostato secondo un proprio modello
- Perciò, *l'infrastruttura per l'interazione* e i *costrutti e i meccanismi di astrazione/specializzazione* assumono un **ruolo preponderante** rispetto alla pura "elaborazione"
  - i dettagli circa la *struttura interna* dei componenti computazionali restano sullo sfondo
  - **si evidenzia invece ciò che permette di *comprendere l'interazione* fra i componenti del sistema**

API e Componenti Software

- L'interazione con un dato sottosistema è generalmente specificata / ottenuta tramite un *insieme di funzioni e procedure* collettivamente denominate **API *(Application Programming Interface)***
  - spesso la API è una collezione "piatta" di funzioni
  - difficile dedurne la struttura concettuale/funzionale del sottosistema
- Si dice ***componente software*** un sottosistema caratterizzato da un *insieme omogeneo e ridotto* di funzionalità
  - gli elementi che permettono l'interazione sono solitamente denominati ***interfaccia***
  - un'interfaccia è una forma di API, dotata però di una precisa unità concettuale / funzionale

Componenti software estendibili

- Molti sottosistemi / componenti software sono concepiti in modo da essere ***estendibili***
  - oltre a un insieme di funzionalità iniziali, prevedono anche opportuni meccanismi (supportati da apposita architettura interna) per *estendere tali funzionalità*, sia a compile-time, sia a run-time
- Molti sistemi software sono quindi oggi progettati / costruiti come ***specializzazione*** *di sottosistemi già esistenti*

Concetti, Linguaggi...

- I concetti di ***componente software***, ***interfaccia*** e ***interazione*** sono oggi essenziali per far fronte alla transizione dei livelli di programmazione “in the small” a “in the large”
- **L’uso di un linguaggio *a oggetti*** porta (più o meno consapevolmente) a seguire **principi fondamentali di organizzazione del software**
  - modularità, incapsulamento, protezione dell’informazione, …
- Questi principi possono ben essere seguiti anche con un linguaggio tradizionale (C, Pascal,…), ma richiedono al progettista una *profonda autodisciplina* metodologica
  - tali linguaggi *non aiutano* a identificare i punti del programma in cui questi principi siano eventualmente violati

...e Progetto

- Ogni progetto nasce da principi e concetti che derivano sia dalla *cultura ed esperienza* del progettista, sia dagli *strumenti* con cui verrà svolta la realizzazione.
- La conoscenza di un certo linguaggio di programmazione rende spesso preponderante il secondo aspetto sul primo
  - si tende a progettare usando le metafore di quel certo linguaggio
  - può non essere *il più adatto* ma solo quello che *si conosce meglio*
- **Se le metafore indotte dal linguaggio non sono adeguate o sono obsolete, quel linguaggio può diventare un handicap**
- Quanto più un sistema software è complesso, tanto più è rilevante *individuare l'architettura più idonea*
  - capace di aiutare la decomposizione concettuale e operativa del sistema in sottosistemi, sostenere a run-time ogni sottosistema, permetterne l'interazione.