Informatica

Terzo anno

Prof. A. Longheu

Definizioni

Un **linguaggio di programmazione** è un linguaggio formale (cioè descritto da regole) utilizzato per implementare gli algoritmi

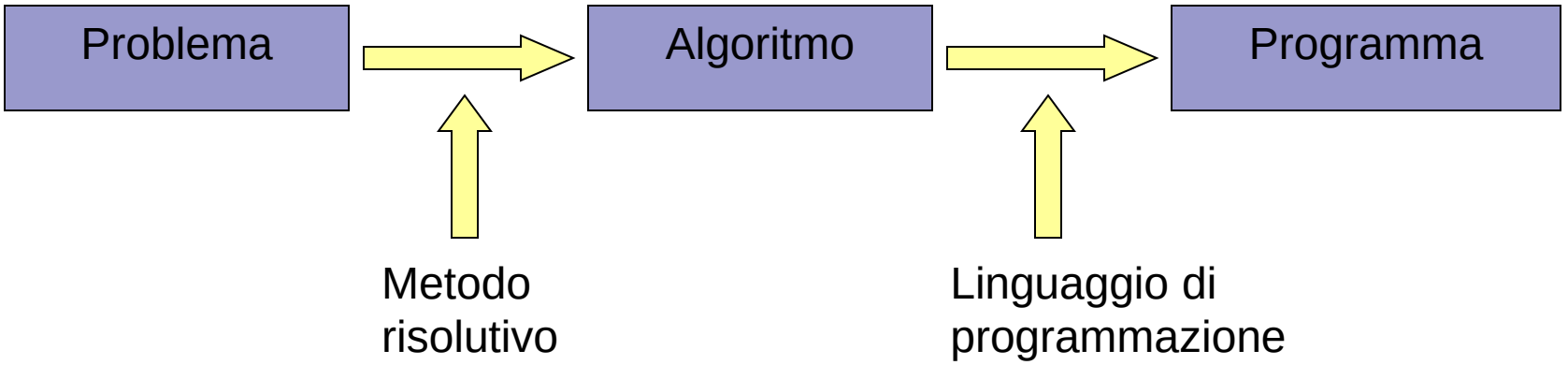
Richiami:

- Un **algoritmo** è un insieme finito di azioni da eseguire per risolvere un problema
- Un **programma** è la descrizione di un algoritmo mediante un opportuno linguaggio (di programmazione)

Algoritmo e programma

Passi per la risoluzione di un problema:

- Individuazione di un procedimento risolutivo
- Scomposizione del procedimento in un insieme ordinato di azioni → ALGORITMO
- Rappresentazione dei dati e dell'algoritmo attraverso un formalismo comprensibile dal calcolatore → PROGRAMMA



Compiti del programmatore

- **Analizzare il problema** riducendolo in termini astratti, eliminando ogni componente non indispensabile e **formulando un modello** del problema.
- Individuare una **strategia risolutiva** e ricondurla ad un ***algoritmo***.
- **Codificare l'algoritmo** in modo tale da renderlo comprensibile al calcolatore.
- **Analizzare il risultato** dell'elaborazione evidenziando eventuali errori nella formulazione del problema, nella strategia risolutiva, nella codifica dell'algoritmo.

Competenze ed abilità del programmatore

- Deve essere in grado di **capire i problemi** e schematizzarli, distinguendone le diverse componenti (dati in input, parametri del problema, dati in output).
- Deve essere in grado di **risolvere problemi** mediante un approccio algoritmico, individuando gli aspetti del problema che possano essere risolti reiterando più volte operazioni simili.
- Deve conoscere i **metodi fondamentali di risoluzione** dei problemi, gli approcci più comuni, le strade notoriamente meno convenienti.
- Deve conoscere a fondo le **caratteristiche e le capacità del calcolatore.**
- Deve essere in grado di comunicare con il calcolatore: ne deve conoscere il **linguaggio**.

Classificazione linguaggi

Diversi criteri sono utilizzati per classificare i linguaggi:

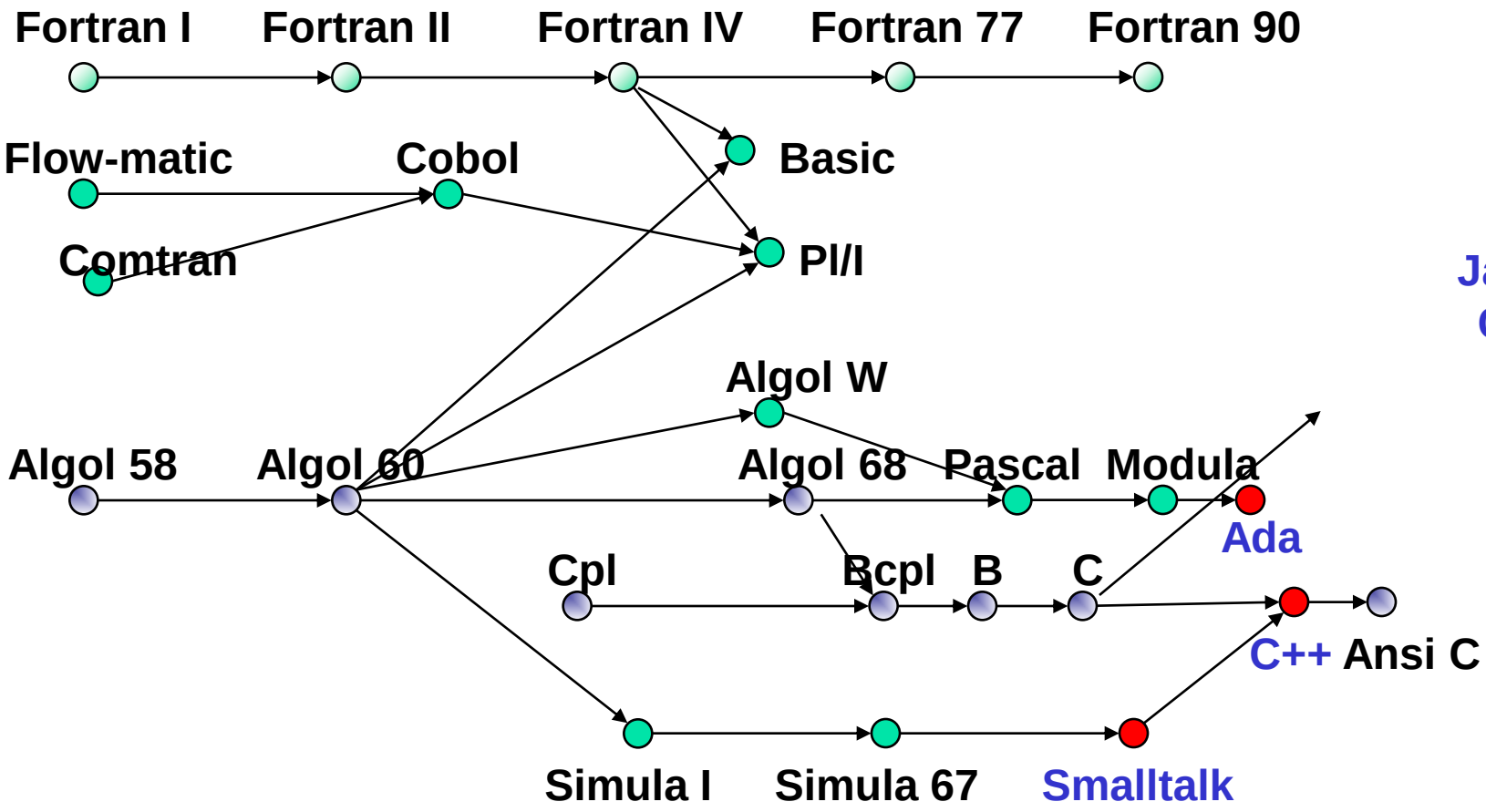
- Periodo storico di definizione
- Program domain
- Livello di astrazione
- Paradigma di programmazione

Questi criteri offrono lo spunto per analizzare diversi aspetti dei linguaggi di programmazione

Classificazione
Storia dei linguaggi

1. Molti linguaggi sono stati sviluppati a partire dagli anni ’50; alcuni di essi sono ancora oggi utilizzati, fondamentalmente a causa dell’**inerzia** che caratterizza il mondo della programmazione: i grandi investimenti fatti da programmatori ed industrie nell’uso di un linguaggio, e la grande quantità di software già presente limitano fortemente l’introduzione di un nuovo linguaggio, anche quando possiede caratteristiche innovative; in sintesi, un linguaggio “vecchio”, se e’ stato molto usato nel passato, sarà usato anche nel futuro perche’ se il software funziona non viene “rottamato”
2. Anche i linguaggi “nuovi”, che si pensa siano piu’ “alla moda” rispetto a quelli vecchi, spesso hanno una parte significativa delle loro istruzioni ereditata dai vecchi, ad esempio “if” e “for” sono istruzioni risalenti all’ALGOL

Classificazione
Storia dei linguaggi



Classificazione Program domain
Campo di applicazione di un linguaggio

- **Applicazioni scientifiche**
  - sono caratterizzate da semplici strutture dati e grandi quantità di calcoli su floating point (FORTRAN, C, C++)
- **Applicazioni gestionali**
  - sono caratterizzate da sofisticate caratteristiche di input/output (COBOL)
- **Intelligenza artificiale**
  - sono applicazioni caratterizzate da processamenrto simbolico e dall'uso delle liste come tipo di dato primitivo (PROLOG, Lisp)
- **Programmazione di sistemi**
  - sono caratterizzate dalla necessità di operare a basso livello, esecuzione efficiente (Assembly, C)
- **Linguaggi special-purpose**
  - Linguaggi che sono utilizzati solo per compiti specifici (Snobol, RPG)

Classificazione
Livello di astrazione

- A causa della struttura interna (basata su transistor utilizzati come interruttori), i computer possono essere programmati soltanto in **linguaggio macchina (LM)**, sequenza di 0 e 1, sintetizzata talvolta con numeri decimali, ottali o esadecimali, indicata anche come **prima generazione di linguaggi**; nei primi computer scarso era il supporto per la programmazione in linguaggio macchina (pulsanti, schede perforate), che era quindi difficoltosa e soggetta a numerosi errori
- I primi tentativi di facilitare la programmazione portarono negli anni 60 alla nascita dell'**assembly**, **linguaggi della seconda generazione** in cui le istruzioni in LM sono espresse tramite codici alfanumerici mnemonici derivati dalla lingua inglese, introducendo così un primo livello di astrazione, restando ancora tuttavia fortemente dipendente dall'hardware della macchina (tipo marca e modello di microprocessore)
- Il passo successivo fu quello di introdurre linguaggi che fossero anche indipendenti dall'hardware, permettendo quindi al programmatore di concentrarsi maggiormente sul problema da risolvere piuttosto che sui dettagli implementativi. Questi linguaggi (C, Pascal, FORTRAN, COBOL, Java ecc.) sono **linguaggi di terza generazione**;

Classificazione
Livello di astrazione

- negli anni 80 sono stati introdotti i **linguaggi di quarta generazione**, che possiedono un ulteriore grado di astrazione, ma che solitamente restringono il campo d'azione rispetto ai linguaggi di terza generazione; un esempio è l'SQL, linguaggio per la gestione di database;
- incrementando il livello di astrazione, con l'obiettivo di utilizzare il linguaggio naturale, si perviene ai **linguaggi di quinta generazione**, attualmente utilizzati nel campo dell'intelligenza artificiale e dei sistemi esperti (che utilizzano motori inferenziali e basi di conoscenza per risolvere problemi)
- i linguaggi 3G, 4G, 5G sono detti anche **linguaggi ad alto livello** perché sono tutti indipendenti dall'hardware della macchina, contrariamente a LM ed assembly, definiti **linguaggi di basso livello**; in generale, un linguaggio è considerato di basso livello se è possibile manipolare la RAM utilizzando istruzioni del linguaggio

Classificazione
Livello di astrazione

Confronto fra linguaggi ad alto e a basso livello:

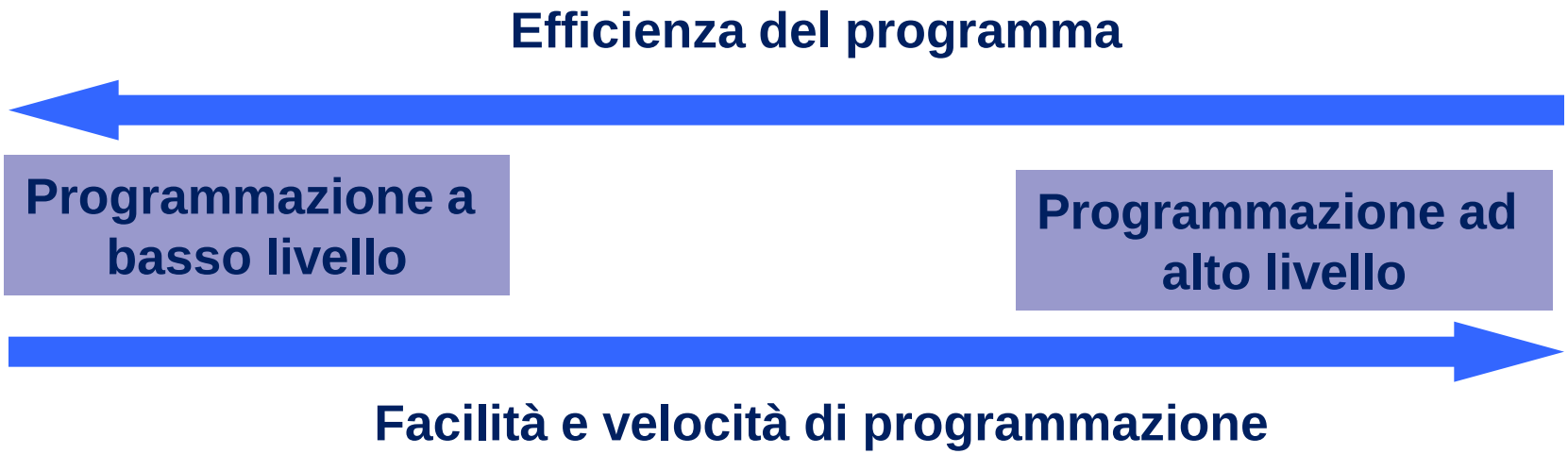
	Basso livello	Alto livello
Apprendimento	Difficoltoso	Facile
Uso	Difficoltoso	Facile
Portabilità	Nulla	Buona
Manutenzione programmi	Scarsa	Buona
Efficienza del codice	Alta	Bassa

Dal confronto si intuisce che i linguaggi ad alto livello sono utilizzati quasi sempre

L'assembly è ancora oggi utilizzato, quando occorre accedere direttamente all'hardware della macchina (sviluppo di driver), quando occorre efficienza, e nei sistemi embedded (centraline auto, allarmi, sensori ecc.)

Classificazione
Livello di astrazione

- La programmazione a *basso livello* è più ardua e meno intuitiva, ma consente di sviluppare programmi efficienti.
- Ad *alto livello* la programmazione è più "naturale" e rapida, ma è possibile che non consenta di produrre software efficiente.



Classificazione
Livello di astrazione

- Qualsiasi sia il linguaggio utilizzato, il computer comprende comunque solo il LM, quindi occorre sempre una traduzione del programma dal linguaggio utilizzato (**codice sorgente**) in LM (**codice oggetto**)
- I traduttori sono di due tipologie: **compilatori** ed **interpreti**

Classificazione
Livello di astrazione

- Il **compilatore** prende in ingresso il codice sorgente e lo traduce completamente in codice oggetto; l'esecuzione del file oggetto è possibile solo a traduzione ultimata
- L'**interprete** traduce ed esegue immediatamente ogni singola istruzione del codice sorgente in codice oggetto, senza produrre un file oggetto distinto
- Confronto compilatore – interprete:

	Compilatore	**Interprete**
Tempo di esecuzione	Basso (viene eseguito codice compilato)	Alto
Correzione errori	Lenta (occorre ricompilare)	Veloce (debug interattivo)
Ottimizzazione del codice	Si	No
Visibilità codice sorgente	No	Si
Necessità del traduttore (occupazione di memoria)	No	Si

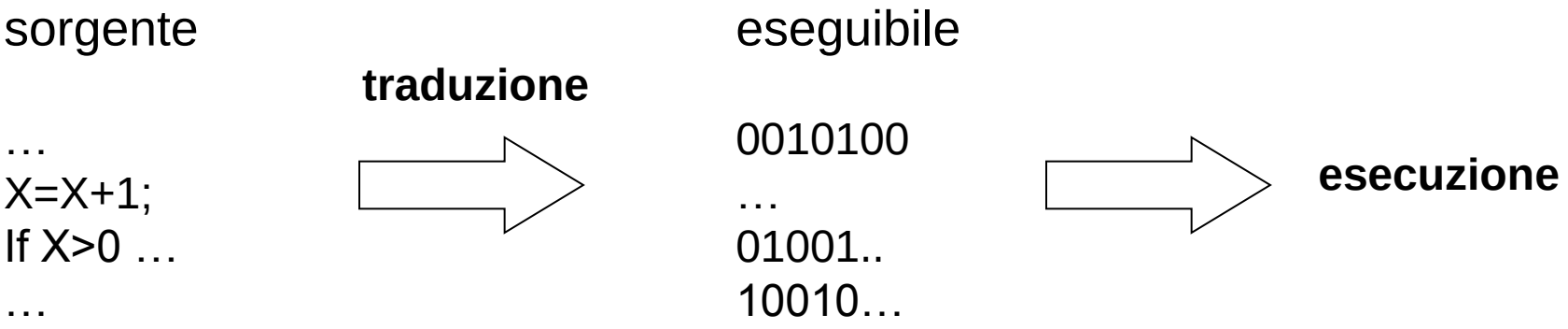
Interprete

- **L' interprete**: itera più volte questo processo
  - **Legge** un'istruzione del programma "sorgente"
  - **Traduce** l'istruzione in linguaggio macchina
  - **Esegue** l'istruzione
  - **Passa** all'interpretazione dell'istruzione successiva
- Al termine di questa operazione, del programma in linguaggio macchina non rimane alcuna traccia (la traduzione non viene memorizzata)
- Se il programma torna più volte su una stessa istruzione, questa verrà tradotta (ed eseguita) ogni volta.
- È necessario disporre dell'interprete per poter eseguire il programma.

Compilatore

- **Compilatore**: esegue una sola volta il processo
  - **Legge** tutte le istruzioni del programma "sorgente" e le traduce in linguaggio macchina.
  - **Memorizza** su disco il programma "eseguibile" tradotto in linguaggio macchina.
- Al termine della compilazione avremo un programma eseguibile in linguaggio macchina.
- La traduzione di ogni istruzione del programma avviene una sola volta, anche se una stessa istruzione viene ripetuta più volte all'interno del programma.
- Non ho bisogno di avere il compilatore ed il sorgente per eseguire il programma: mi basta il programma eseguibile

Compilatori e Interpreti



- Nel caso del compilatore, lo schema precedente viene percorso **una volta sola**, prima dell'esecuzione

- Nel caso dell'interprete, lo schema viene invece attraversato **tante volte quante sono le istruzioni** che compongono il programma

Compilatori e Interpreti

Tipicamente,
l'esecuzione di un programma *compilato*
è più veloce
dell'esecuzione di un programma *interpretato*

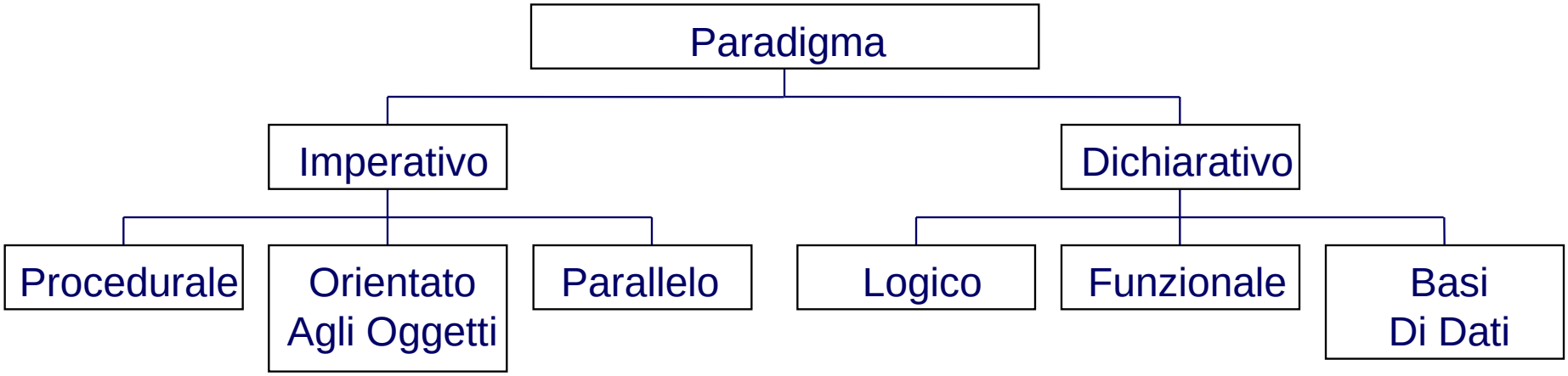
Classificazione
Paradigma di programmazione

Un paradigma è:

- Un modello di riferimento
- Un insieme di regole di programmazione, valide per più di un linguaggio
- Una raccolta di caratteristiche astratte

La scelta del paradigma che meglio risolve un dato problema è di fondamentale importanza per un programmatore; alcuni problemi possono essere risolti utilizzando più paradigmi

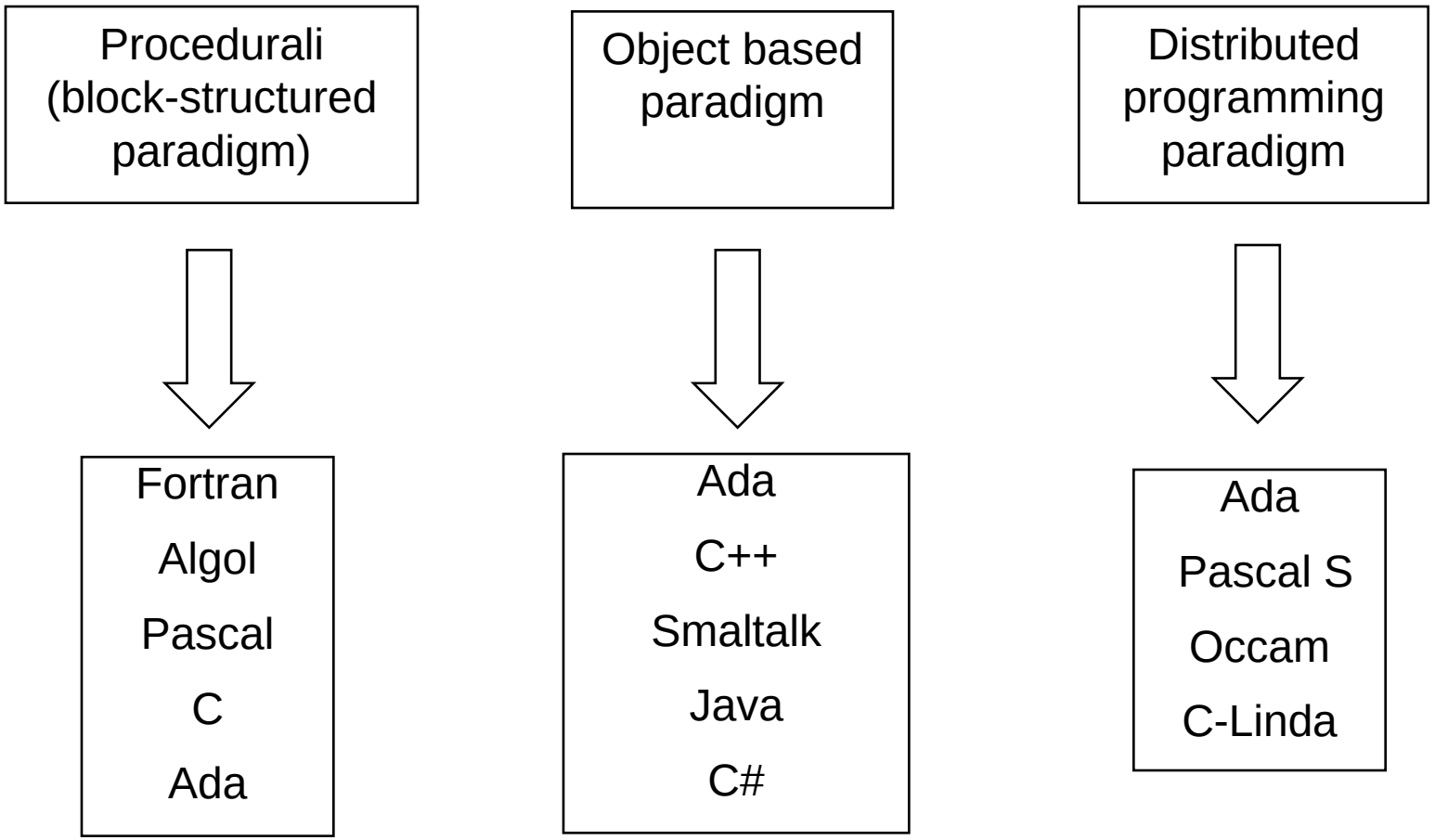
Classificazione
Paradigma di programmazione



Classificazione
Paradigma di programmazione

- I linguaggi **imperativi** sono quelli in cui il modello di computazione è il cambiamento di stato della memoria
- Sono basati sul concetto di assegnazione di un valore ad una locazione di memoria
- Il compito del programmatore è definire una sequenza di cambiamenti di stato della memoria che produrranno lo stato finale desiderato
- In pratica, nei linguaggi imperativi deve essere il programmatore a specificare “come” va risolto il problema
- I linguaggi imperativi sono divisi nelle tre categorie:
  - **Procedurali**, secondo cui il programma viene organizzato in procedure (funzioni)
  - **Orientati agli oggetti** (Object-oriented), in cui l’elemento di base è l’oggetto, metafora degli oggetti del problema reale
  - **Paralleli**, in cui processi e relativa sincronizzazione sono dominanti

Classificazione
Paradigma di programmazione



Classificazione
Paradigma di programmazione

- I linguaggi dichiarativi sono basati sulla dichiarazione di affermazioni che descrivono una data realtà ed il problema che in tale contesto si desidera risolvere
- Compito del programmatore è utilizzare un linguaggio di specifica per le affermazioni, e fornire le affermazioni opportune per consentire alla macchina di risolvere il problema
- In pratica, nei linguaggi dichiarativi il programmatore si occupa di definire “cosa” vuole, non il come, compito che viene demandato alla macchina (in realtà, è demandato alla capacità inferenziale del traduttore che altri umani hanno scritto per quella macchina)
- L’utilizzo di questi linguaggi è ancora piuttosto ristretto (intelligenza artificiale), fondamentalmente perché i motori inferenziali non sono ancora molto potenti (riescono a risolvere solo pochi problemi, o solo di una certa categoria) né efficienti (sono lenti nel trovare la soluzione, magari perche’ devono provarne diverse)

Classificazione
Paradigma di programmazione

- I linguaggi dichiarativi sono divisi in tre categorie:
  - Linguaggi logici, in cui si utilizzano assiomi, combinati con le regole della logica, per arrivare al risultato (verità o falsità di un affermazione data come problema)
  - Linguaggi funzionali, in cui sono le funzioni gli elementi di base, ed il problema viene risolto valutandole (fornendo gli argomenti necessari)
  - Database, in cui si utilizzano appositi costrutti (select, insert) per gestire i dati