Linguaggi di Programmazione

Corso di Laurea in "ITPS"

a.a. 07-08

Macchine Astratte

Valeria Carofiglio

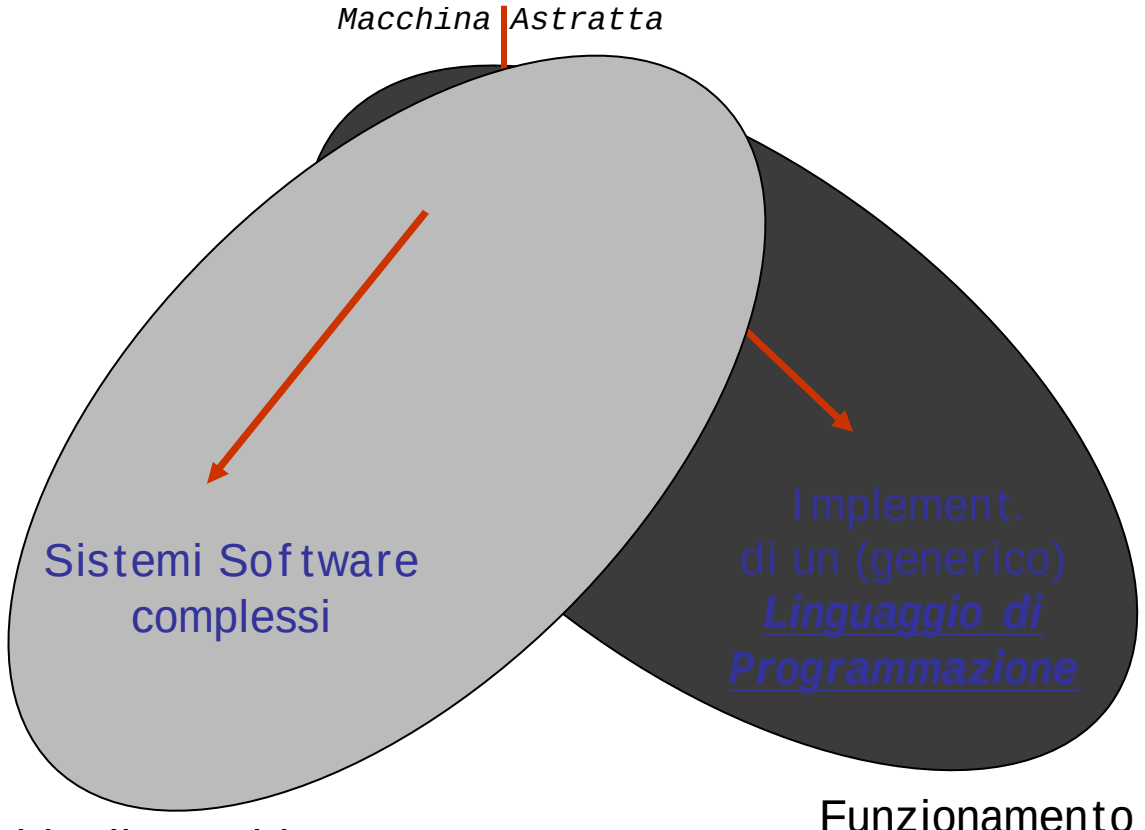
(Questo materiale è una rivisitazione del materiale prodotto da Nicola Fanizzi)

Contenuti

- Macchine Astratte
- Metodi di Implementazione
- Ambienti di Programmazione

Contenuti

La nozione di Astrazione per isolare aspetti rilevanti di un problema



Gerarchie di macchine Astratte

Funzionamento di ***Traduttori*** di linguaggi

Macchine Fisica e Linguaggio associato

Calcolatore

dispositivo elettronico

capace di eseguire istruzioni (algoritmi)

opportunamente

formalizzati in un linguaggio *comprensibile*

per questo esecutore

- **Linguaggio di macchina** L : linguaggio compreso dalla macchina (sintassi&semantica)

- **Programma**: insieme di istruzioni scritte in L

Macchine Astratta

Astrazione di una macchina fisica

macchina astratta ←→ linguaggio di programmazione

Per descrivere cosa sia

l'implementazione di un linguaggio,

senza addentrarci nei dettagli di una particolare implementazione

Macchine Astratta

Astrazione di una macchina fisica

macchina astratta ←→ linguaggio di programmazione

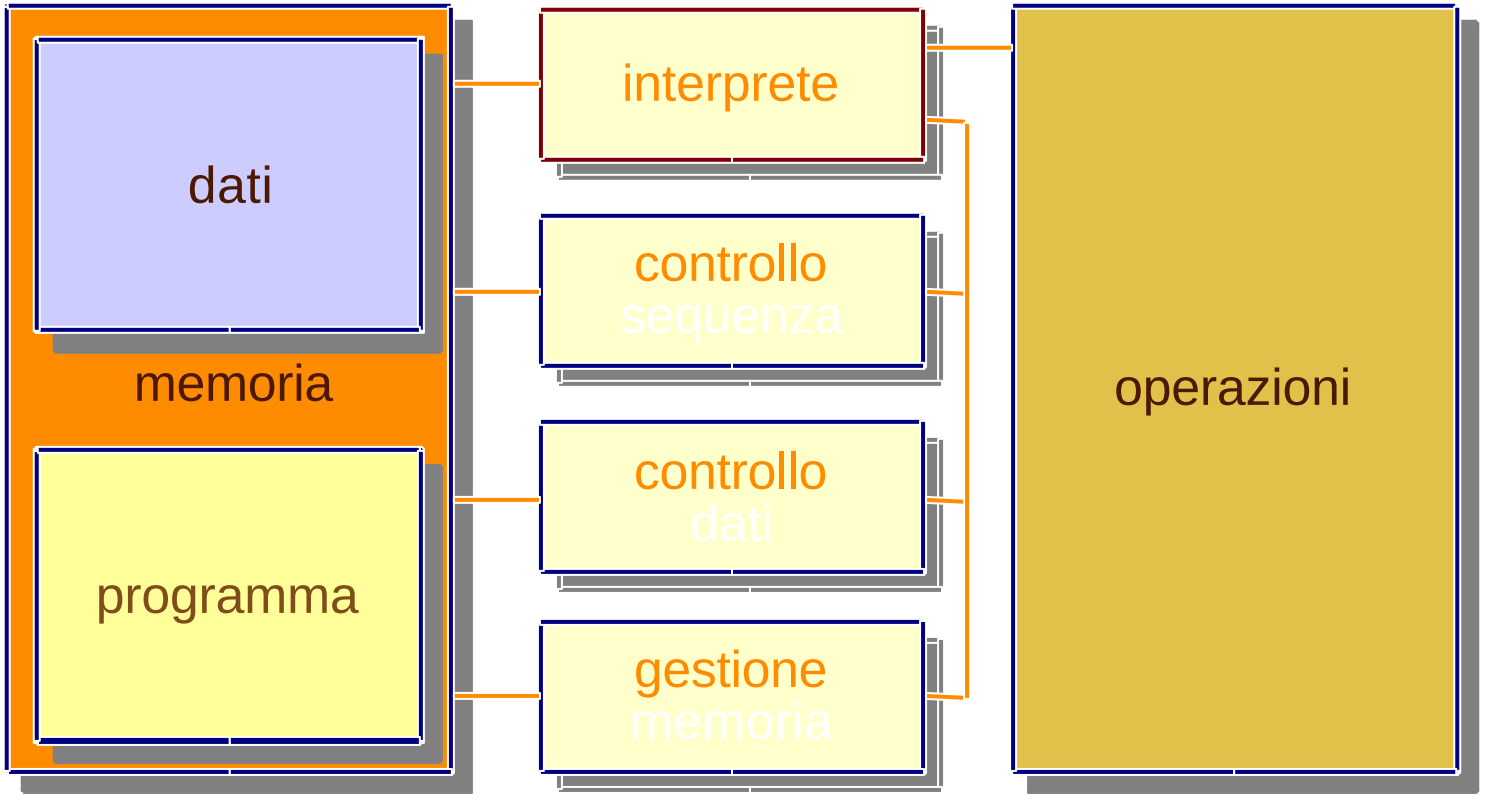
Per descrivere cosa sia

l'implementazione di un linguaggio,

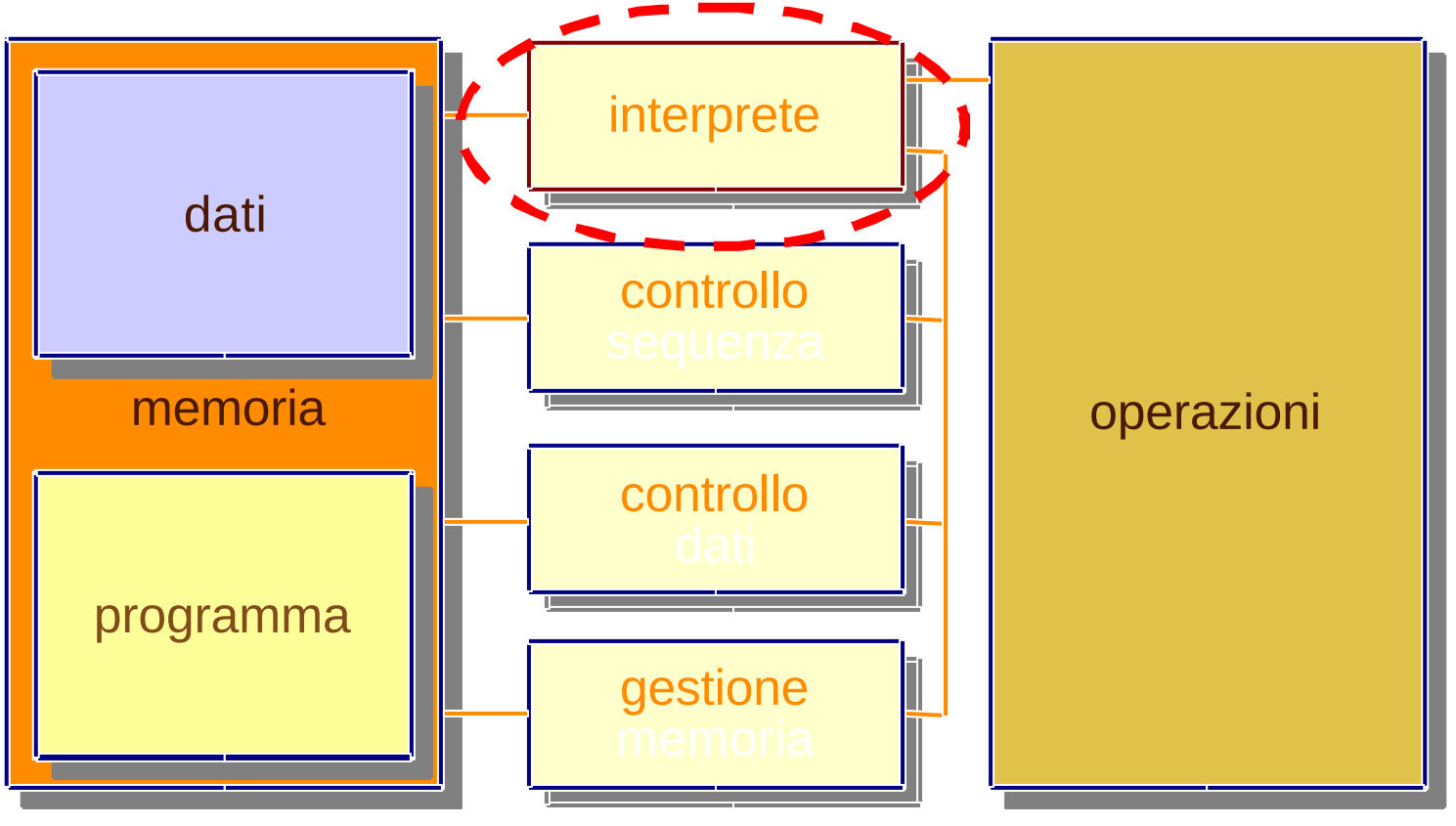
senza addentrarci nei dettagli di una particolare implementazione

Macchina astratta per L, indicata con M_L:
insieme di **algoritmi** e **strutture dati**
che permettono di memorizzare
ed eseguire programmi scritti in L

Struttura di una macchina astratta

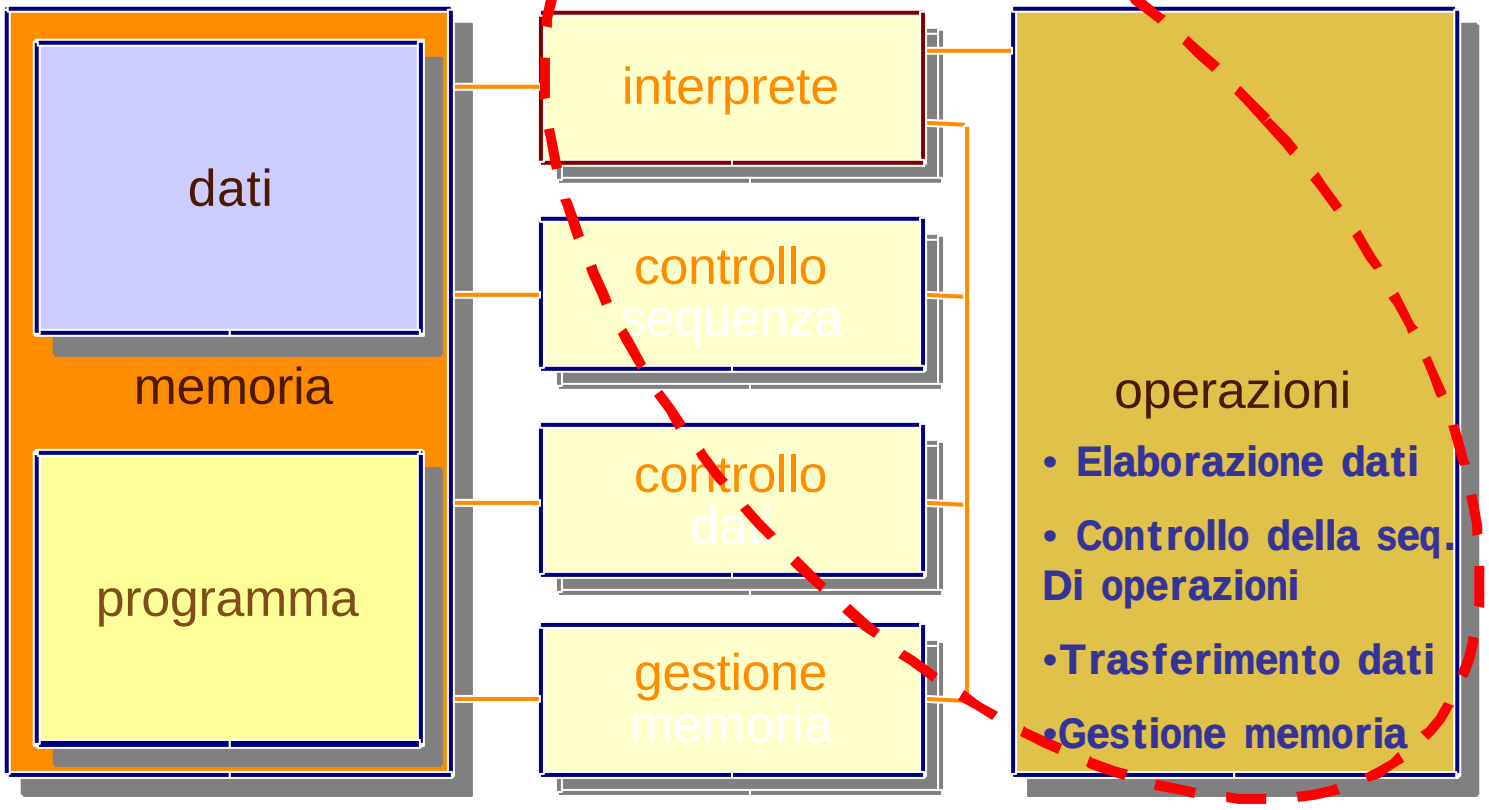


Struttura di una macchina astratta



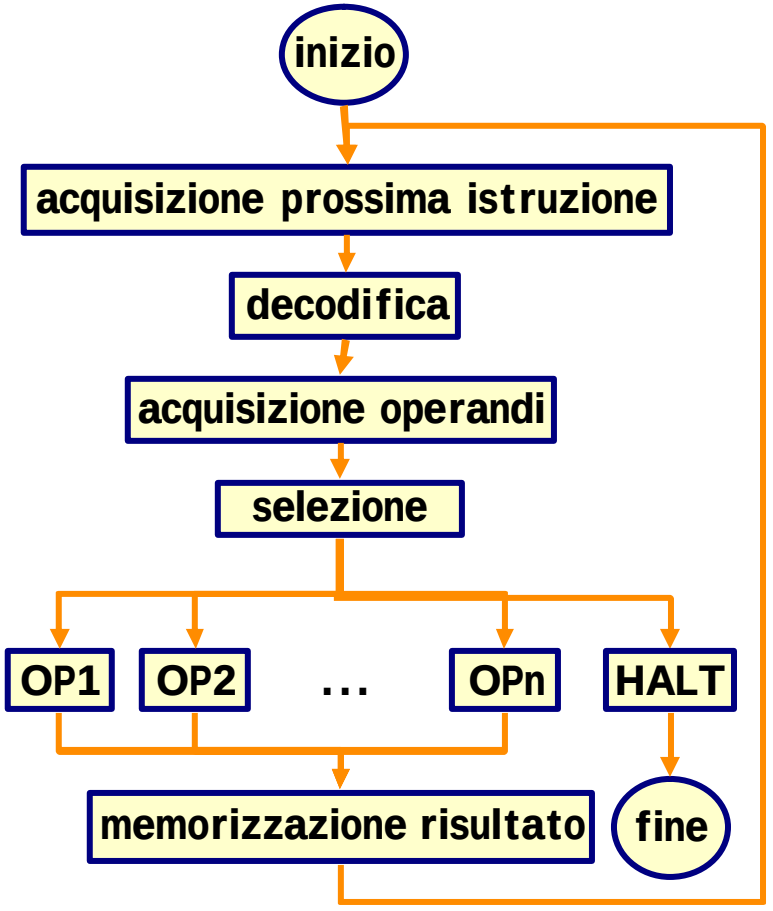
Esegue operazioni che dipendono dal particolare linguaggio L_M

Struttura di una macchina astratta



Esegue operazioni che dipendono dal particolare linguaggio L_M

Interprete: ciclo di esecuzione



1) Acquisizione da memoria prossima istruzione
2) Decodifica operazione e operandi
3) Acquisizione operandi
4) Esecuzione operazione
5) Memorizzazione risultato
6) Se non era l'istr. HALT:passa all'istruzione successiva (vai a 1)

Linguaggio Macchina L_M

Data una macchina astratta M_L
il linguaggio L
compreso dall'interprete di M_L
è detto
linguaggio macchina di M_L

Un esempio di macchina astratta

La macchina Hardware

MH_{LH} : Macchina

Memorie per memorizzare dati e programmi: secondaria (disp.magnetici), principale (seq.lineare di celle di lunghezza fisica), registri, chache

LH : Linguaggio

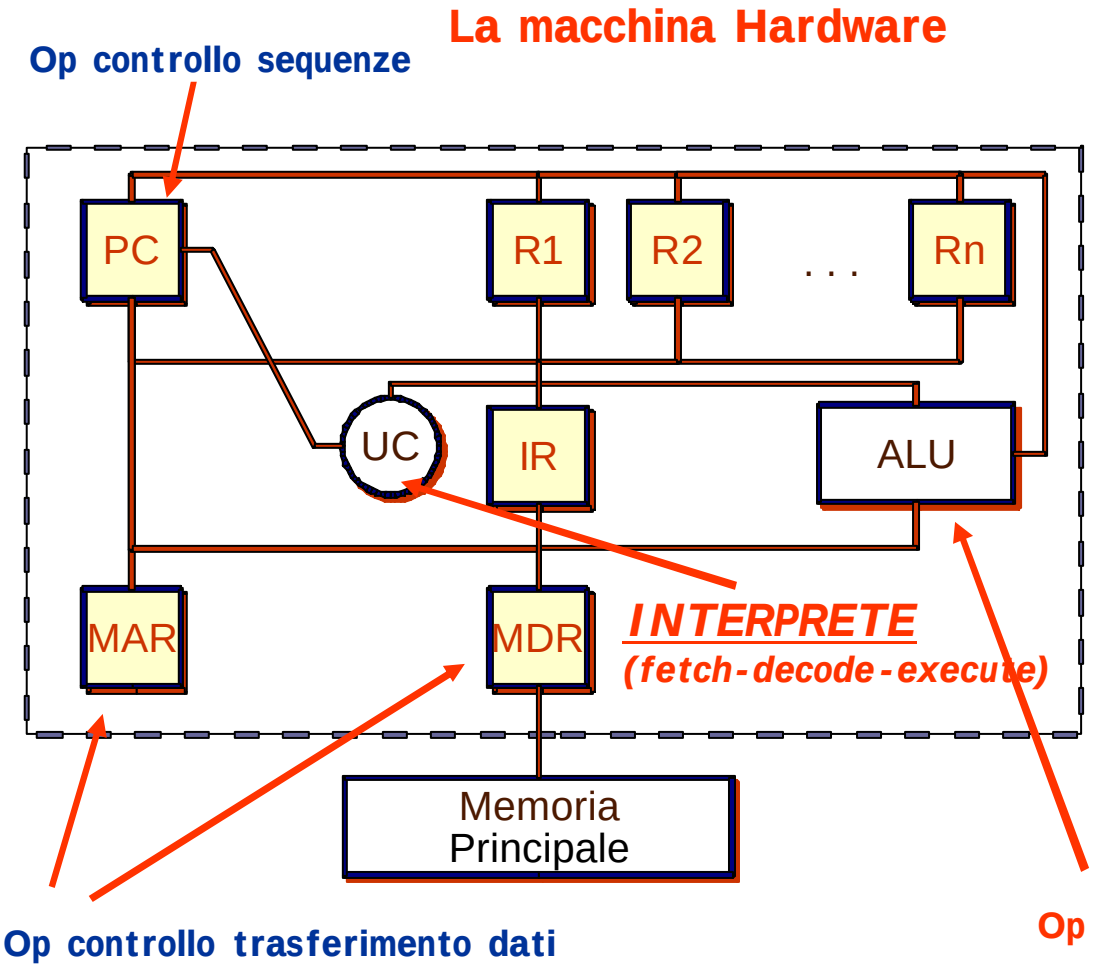
Alfabeto: binario

dati: pochi tipi primitivi ←→ rappresent. fisiche diverse (interi/complemento a 1 o a 2, reali/virgola mobile, caratteri/seq. Di cifre binarie –ASCII o UNICODE e seq. di bit di lunghezza fissa)

Istruzioni semplici, del tipo: CodOp op1 op2 (es:ADD R5 R0)

Interprete

Un esempio di macchina astratta (cont.)



MH_{LH} : Macchina

Memorie:
secondaria,
principale,
registri, chache

LH : Linguaggio

Istr:

CodOp op1 op2

ADD R5 R0

Panoramica

Evoluzione delle tecniche di programmazione

Linguaggi di programmazione:

Sempre meno legati alle caratteristiche dell'elaboratore

Sempre più orientati al problema

(linguaggi ad alto livello)

E' necessario un TRADUTTORE:

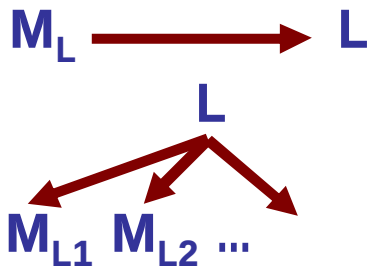
Istr.in linguaggio ad alto livello ⇢ **Istr. in linguaggio maccchina**

-Potenti,
-Rigorosi
-Versatili

(Compilatore/Interprete)

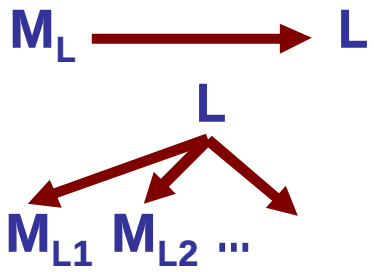
-Comprensibili al computer
-Efficienti
-Dipendenti dall'architettura

Implementazione di un linguaggio



M_{Li} differiscono nel modo in cui ***l'interprete è realizzato*** e nelle ***strutture dati*** che utilizzano

Implementazione di un linguaggio



M_{Li} differiscono nel modo in cui ***l'interprete è realizzato*** e nelle ***strutture dati*** che utilizzano

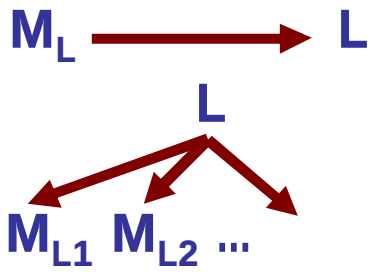
Implementare un linguaggio L

Realizzare una macchina astratta M_L

che abbia

L come linguaggio macchina

Implementazione di un linguaggio



M_{Li} differiscono nel modo in cui ***l'interprete è realizzato*** e nelle ***strutture dati*** che utilizzano

Implementare un linguaggio L

Realizzare una macchina astratta M_L

che abbia

L come linguaggio macchina

Realizzazione:

1. In Hardware (Tutte le macchine prima o poi dovranno utilizzare dispositivi fisici)
2. Mediante Software
3. Mediante Firmware

Realizzare una macchina astratta

Realizzazione di M_L in hardware

Realizzazione tramite dispositivi fisici: memorie, unità aritmetico-logiche, bus, reti, ...

Realizzare una macchina astratta

Realizzazione di M_L in hardware

Realizzazione tramite dispositivi fisici: memorie, unità aritmetico-logiche, bus, reti, ...

- ***Vantaggio***: esecuzione molto veloce
  - Linguaggi a basso livello
  - Linguaggi dedicati

Realizzare una macchina astratta

Realizzazione di M_L in hardware

Realizzazione tramite dispositivi fisici: memorie, unità aritmetico-logiche, bus, reti, ...

- ***Vantaggio***: esecuzione molto veloce
  - Linguaggi a basso livello
  - Linguaggi dedicati
- ***Svantaggi***: non va bene per linguaggi di alto livello
  - Complessità della progettazione
  - Immodificabilità della realizzazione
- Influenzata dalla struttura dei linguaggi ad alto livello (nella scelta delle op. primitive)

Realizzare una macchina astratta

Simulazione di M_L mediante software

Realizzazione tramite strutture-dati e algoritmi: mediante programmi in un (altro) linguaggio L' (supposto già implementato)

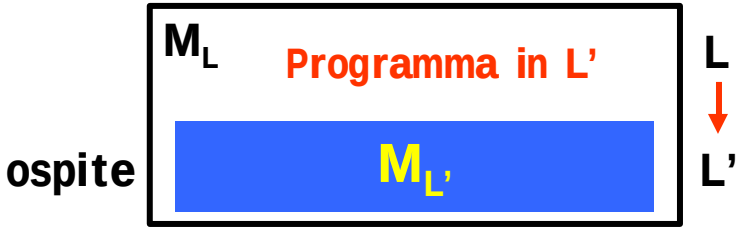
Data una macchina **$M_{L'}$** per L', si può realizzare una macchina M_L per L, simulando, con un programma in linguaggio L', i costrutti di L ossia le funzionalità di M_L

Realizzare una macchina astratta

Simulazione di M_L mediante software

Realizzazione tramite strutture-dati e algoritmi: mediante programmi in un (altro) linguaggio L' (supposto già implementato)

Data una macchina **$M_{L'}$** per L', si può realizzare una macchina M_L per L, simulando, con un programma in linguaggio L', i costrutti di L ossia le funzionalità di M_L



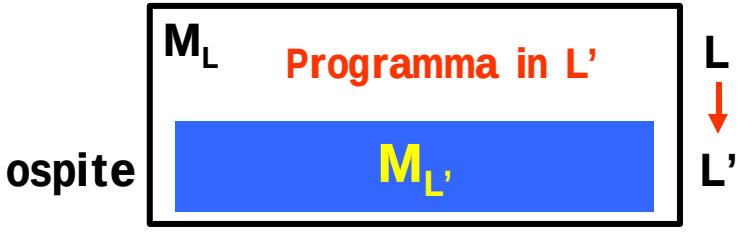
Realizzare una macchina astratta

Simulazione di M_L mediante software

Realizzazione tramite strutture-dati e algoritmi: mediante programmi in un (altro) linguaggio L' (supposto già implementato)

Data una macchina **$M_{L'}$** per L', si può realizzare una macchina M_L per L, simulando, con un programma in linguaggio L', i costrutti di L ossia le funzionalità di M_L

Vantaggio: max. flessibilità

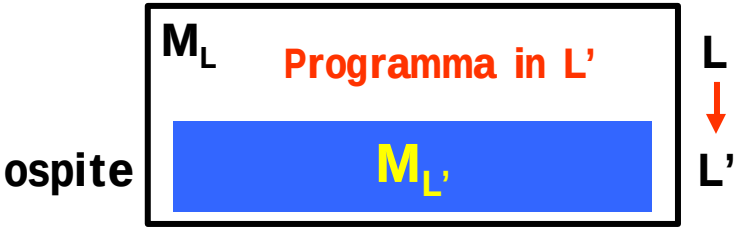


Realizzare una macchina astratta

Simulazione di M_L mediante software

Realizzazione tramite strutture-dati e algoritmi: mediante programmi in un (altro) linguaggio L' (supposto già implementato)

Data una macchina $\mathbf{M_{L'}}$ per L', si può realizzare una macchina M_L per L, simulando, con un programma in linguaggio L', i costrutti di L ossia le funzionalità di M_L



Vantaggio: max. flessibilità
Svantaggio: prestazioni inferiori; l'esecuzione passa per l'interpretazione sulla macchina $M_{L'}$, comunque essa sia stata realizzata (HW,SW,FW)

Realizzare una macchina astratta

Simulazione (emulazione) mediante firmware

Realizzazione intermedia
ottenuta attraverso microprogrammi
che simulano algoritmi e str.dati di M_L

Simile al caso precedente ma con una macchina intermedia che è realizzata come macchina fisica (quindi con linguaggi a basso livello: micro-linguaggi)

Realizzare una macchina astratta

Simulazione (emulazione) mediante firmware

Realizzazione intermedia
ottenuta attraverso microprogrammi
che simulano algoritmi e str.dati di M_L

Simile al caso precedente ma con una macchina intermedia che è realizzata come macchina fisica (quindi con linguaggi a basso livello: micro-linguaggi)

- Vantaggio: elevata velocità (intermedia)

Realizzare una macchina astratta

Simulazione (emulazione) mediante firmware

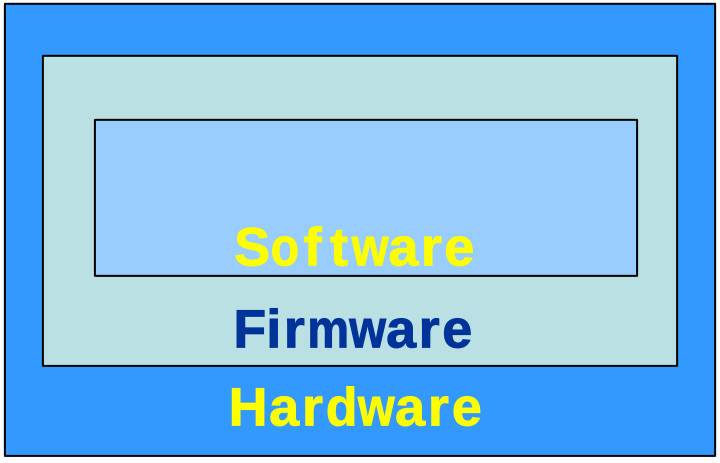
Realizzazione intermedia
ottenuta attraverso microprogrammi
che simulano algoritmi e str.dati di ML

Simile al caso precedente ma con una macchina intermedia che è realizzata come macchina fisica (quindi con linguaggi a basso livello: micro-linguaggi)

- Vantaggio: elevata velocità (intermedia)
- Svantaggi:
  - Minore flessibilità
  - Maggiore complessità della programmazione
  - Opportuni dispositivi di scrittura per memorizzare i microprogrammi (risiedono in una memoria di sola lettura)

Una Soluzione Mista

Spesso la realizzazione di una macchina astratta prevede una combinazione delle tre metodologie fornendo una gerarchia di macchine astratte



Implementazione: il caso ideale

Dato un Linguaggio L

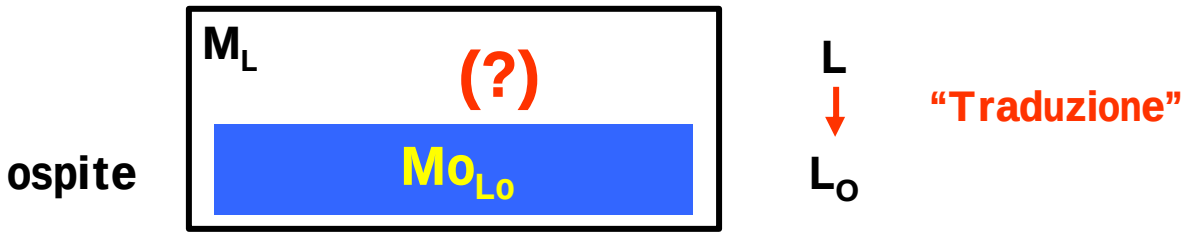
Realizzare la macchina astratta M_L

(simulazione mediante software)

Implementazione: il caso ideale

Dato un Linguaggio L
(che vogliamo implementare)

Realizzare la macchina astratta M_L



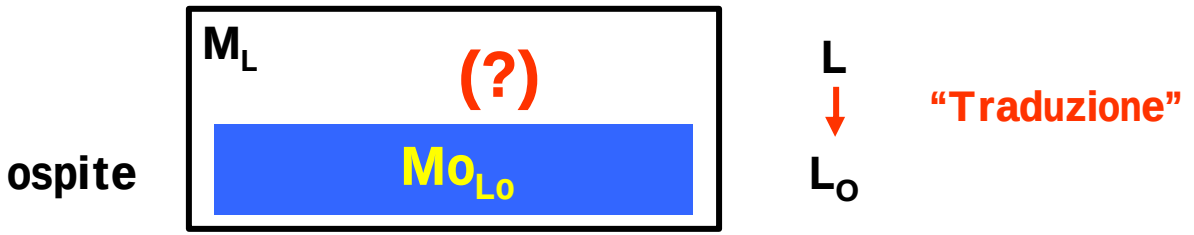
(simulazione mediante software)

Implementazione: il caso ideale

Dato un Linguaggio L

Realizzare la macchina astratta M_L

(simulazione mediante software)



Due modalità di implementazione:

Traduzione IMPLICITA

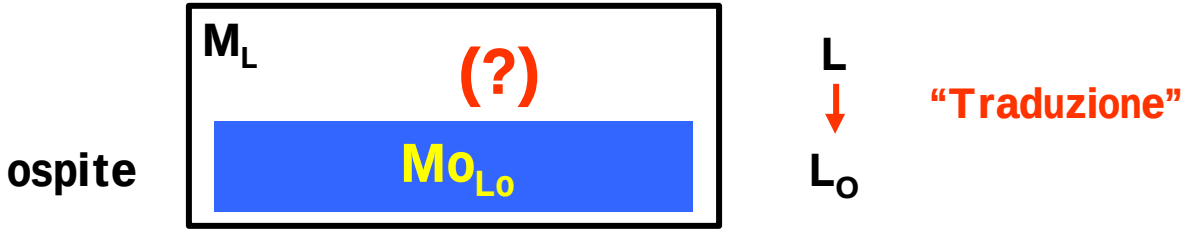
Traduzione ESPLICITA

Implementazione: il caso ideale

Dato un Linguaggio L

Realizzare la macchina astratta M_L

(simulazione mediante software)



Due modalità di implementazione:

Traduzione IMPLICITA

Interpretativa Pura:

Interprete(M_L) = Interprete(M_{Lo})

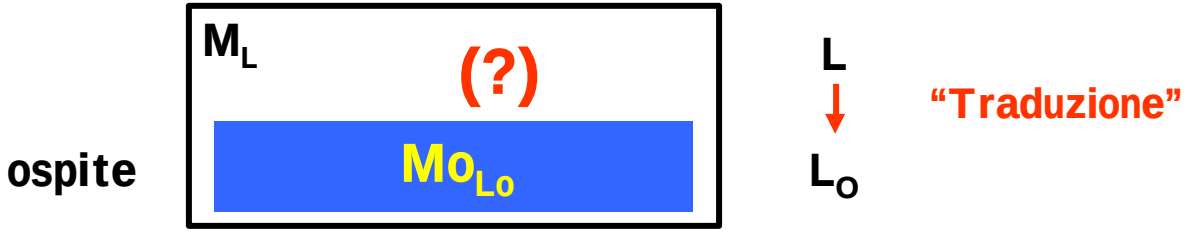
Traduzione ESPLICITA

Implementazione: il caso ideale

Dato un Linguaggio L

Realizzare la macchina astratta M_L

(simulazione mediante software)



Due modalità di implementazione:

Traduzione IMPLICITA

Interpretativa Pura:

$Interprete(M_L) = Interprete(M_{Lo})$

Traduzione ESPLICITA

Compilativa Pura:

$Interprete(M_L) \ ^1 \ Interprete(M_{Lo})$

Notazione

- **$Prog^L$: insieme dei possibili programmi in linguaggio L**
- **D : dominio dei dati in input e output**
- **Programma in L**:

$$P^L: D \rightarrow D$$

tale che

$$\begin{cases} P^L(input) = Output & \text{quando termina} \\ \textit{indefinita} & \text{altrimenti} \end{cases}$$

Implementazione Interpretativa Pura

I^{Lo}_L interpreta tutte le istruzioni di L

Implementazione Interpretativa Pura

P^L:
Programma scritto in L

M_L ***Interprete per L scritto in Lo*** (not. I^{Lo}_L)

Mo_{Lo}

I^{Lo}_L interpreta tutte le istruzioni di L

Esecuzione di P^L

Implementazione Interpretativa Pura

P^L:
Programma scritto in L

D:
Input

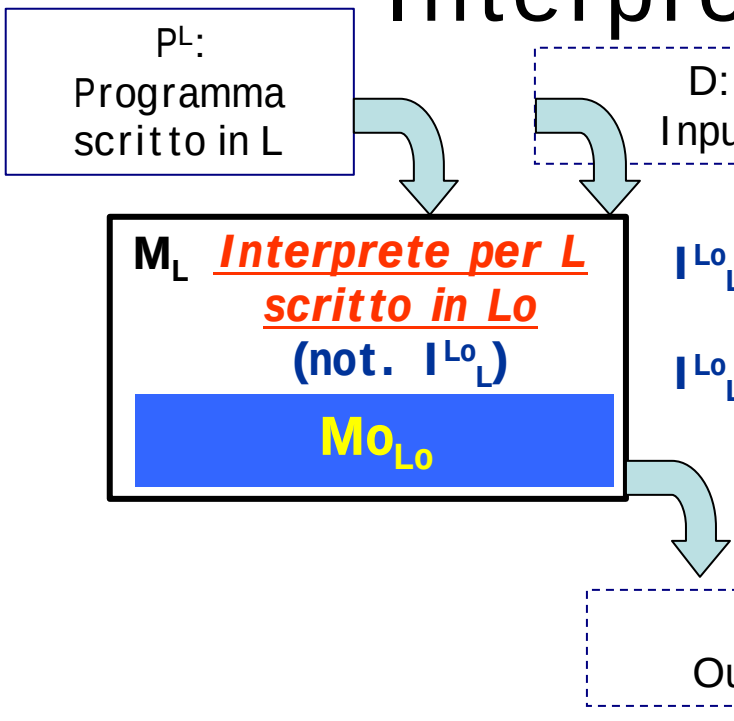
M_L ***Interprete per L scritto in Lo***
(not. I^{Lo}_L)

Mo_{Lo}

I^{Lo}_L interpreta tutte le istruzioni di L

Esecuzione di P^L

Implementazione di L
Interpretativa Pura



I^{Lo}_L interpreta tutte le istruzioni di L

I^{Lo}_L su Mo_{Lo} con input P^L e D

Esecuzione di P^L

Interprete

Un Interprete per un linguaggio L,
Scritto nel linguaggio Lo,
è un programma che realizza
una funzione parziale

$$I^{Lo}_{L}: (Prog^{L} * D) \rightarrow D$$

Tale che

$$I^{Lo}_{L}\ (P^{L},\ Input) = P^{L}(Input)$$

Interprete

Un Interprete per un linguaggio L,
Scritto nel linguaggio Lo,
è un programma che realizza
una funzione parziale

$$I^{Lo}_{L}: (Prog^{L} * D) \rightarrow D$$

Tale che

$$I^{Lo}_{L}\ (P^{L},\ Input) = P^{L}(Input)$$

Le fasi di traduzione e di esecuzione del programma P^{L}
***non sono separate*!!**

Implementazione di L
Compilativa Pura

$C_{L,Lo}$ **traduce tutte le istruzioni di L in Lo**

M_L ***Compilatore da L ad Lo*** **(not. $C_{L,Lo}$)**

L è detto linguaggio *sorgente*

Lo è detto *linguaggio Oggetto*

Mo_{Lo}

Implementazione di L
Compilativa Pura

P^L:
Programma scritto in L

$C_{L,Lo}$ **traduce tutte le istruzioni di L in Lo**

M_L ***Compilatore da L ad Lo*** **(not. $C_{L,Lo}$)**

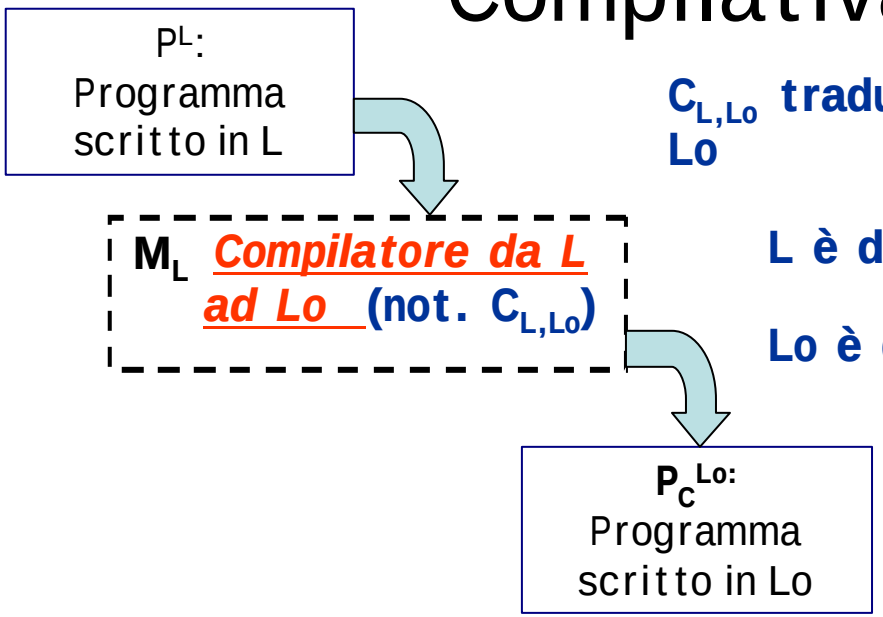
L è detto linguaggio *sorgente*

Lo è detto *linguaggio Oggetto*

Mo_{Lo}

Esecuzione di P^L

Implementazione di L
Compilativa Pura



$C_{L,Lo}$ traduce tutte le istruzioni di L in Lo

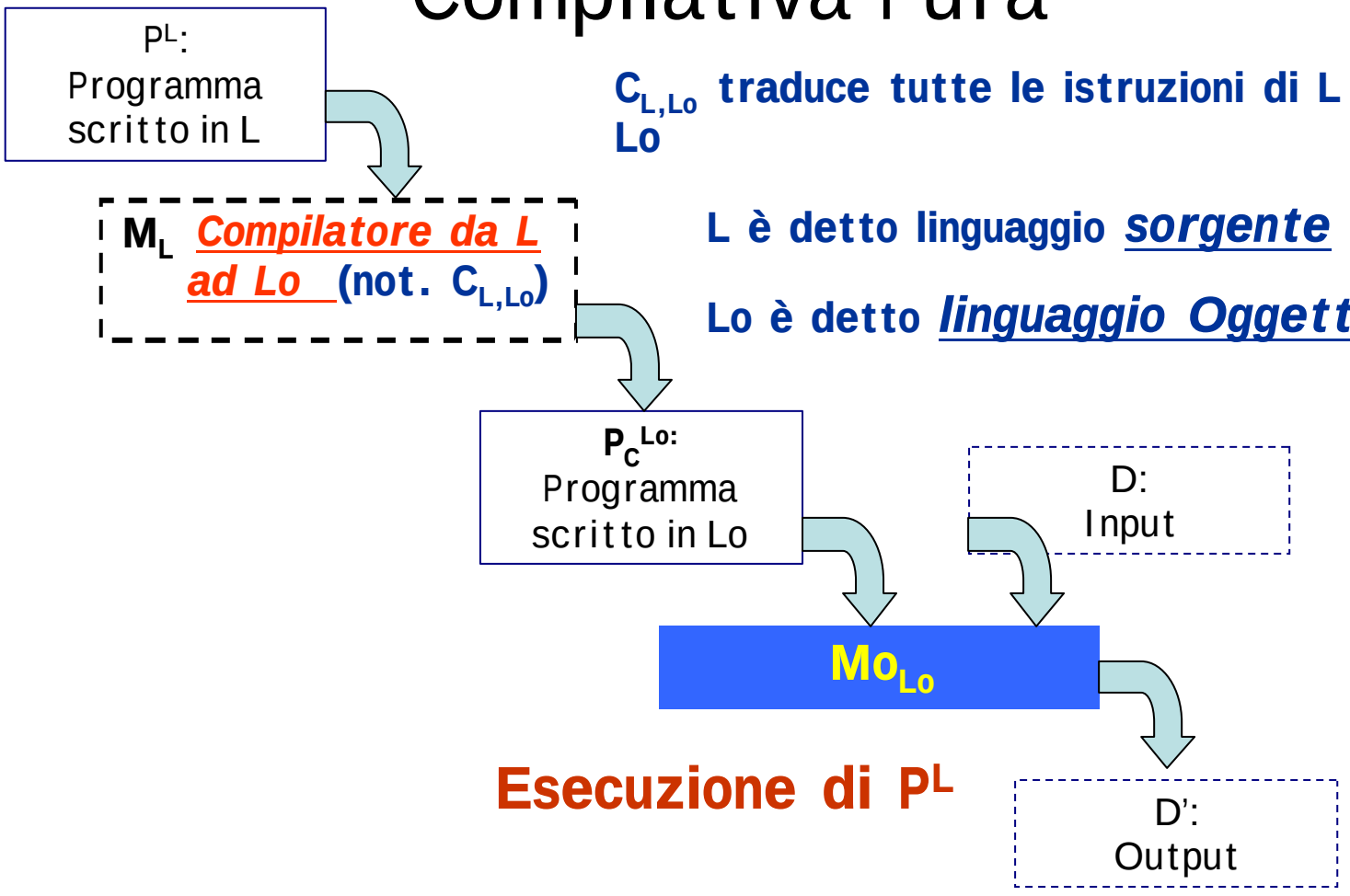
L è detto linguaggio *sorgente*

Lo è detto *linguaggio Oggetto*

Mo_{Lo}

Esecuzione di P^L

Implementazione di L
Compilativa Pura



Compilatore

Un compilatore da un ling.L ad un linguaggio L_0, è un programma che realizza una funzione parziale

$$C_{L,Lo}: Prog^{L} \rightarrow Prog^{Lo}$$

Tale che, dato un programma P^L, se

$$C_{L,Lo}(P^L) = Pc^{Lo}$$

Allora, per ogni input

$$P^L(Input) = Pc^{Lo}(Input)$$

Compilatore

Un compilatore da un ling.L ad un linguaggio L_0, è un programma che realizza una funzione parziale

$$C_{L,Lo}: Prog^{L} \rightarrow Prog^{Lo}$$

Tale che, dato un programma P^L, se

$$C_{L,Lo}(P^L) = Pc^{Lo}$$

Allora, per ogni input

$$P^L(Input) = Pc^{Lo}(Input)$$

Le fasi di traduzione e di esecuzione del programma P^L *sono separate*!!

Confornto

- ***Implementazione interpretativa pura***
  - Svantaggi
    - Scarsa efficienza: overhead per la decodifica
    - replica della traduzione
  - Vantaggi:
    - Maggiore flessibilità
    - Facilità di sviluppo e debugging (interazione diretta con l'esecuzione)
    - Memoria ridotta (solo sorgente; no nuovo codice)

- ***Implementazione compilativa pura***
  - Vantaggi:
    - Esecuzione più efficiente
    - Una sola traduzione
  - Svantaggi:
    - Minore flessibilità
    - Perdita di informazioni sul sorgente
    - Minore facilità di debugging degli errori run-time

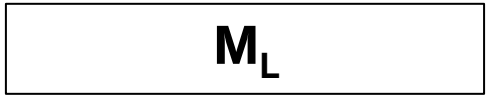
Implementazione: il caso Reale

Dato un Linguaggio L

Realizzare la macchina astratta M_L

(simulazione mediante software)

Macchina che vogliamo realizzare



L

ospite

L_O

Implementazione: il caso Reale

Dato un Linguaggio L

Realizzare la macchina astratta M_L

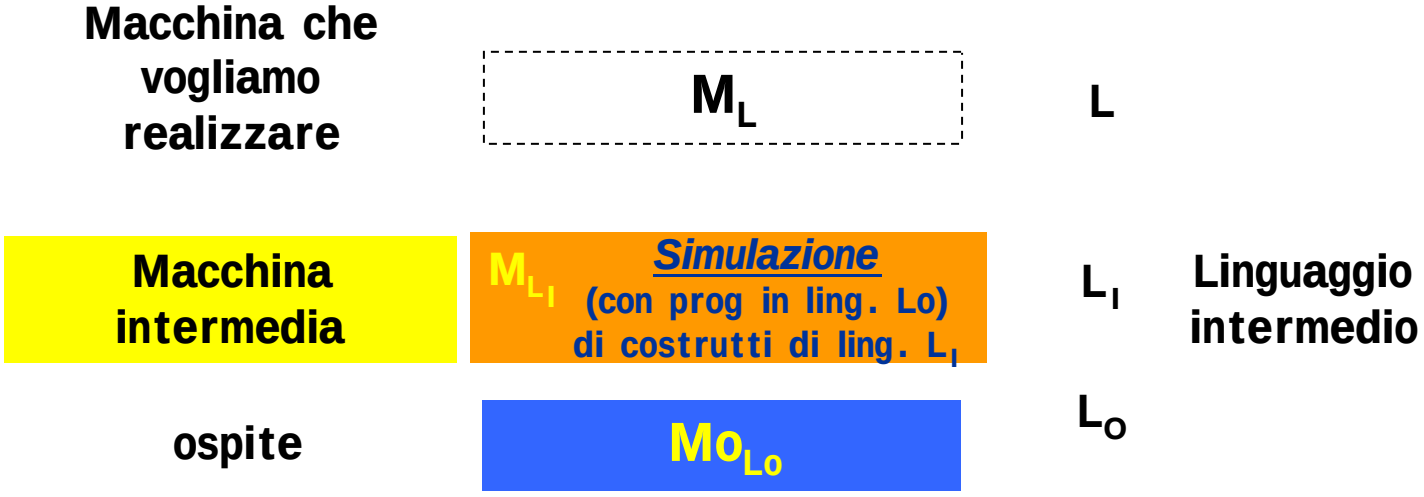
(simulazione mediante software)

Implementazione: il caso Reale

Dato un Linguaggio L

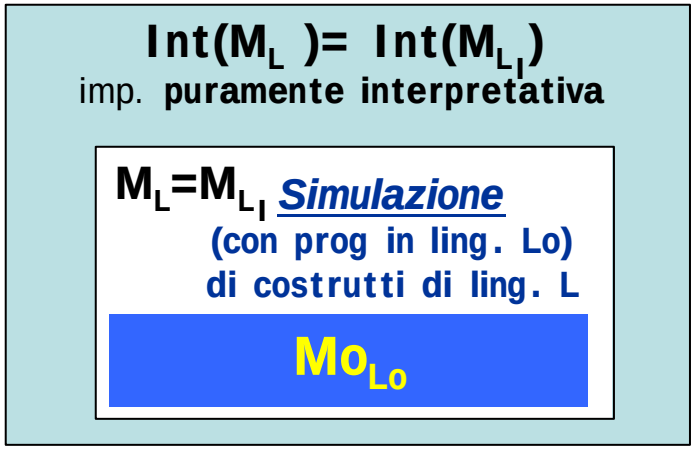
Realizzare la macchina astratta M_L

(simulazione mediante software)

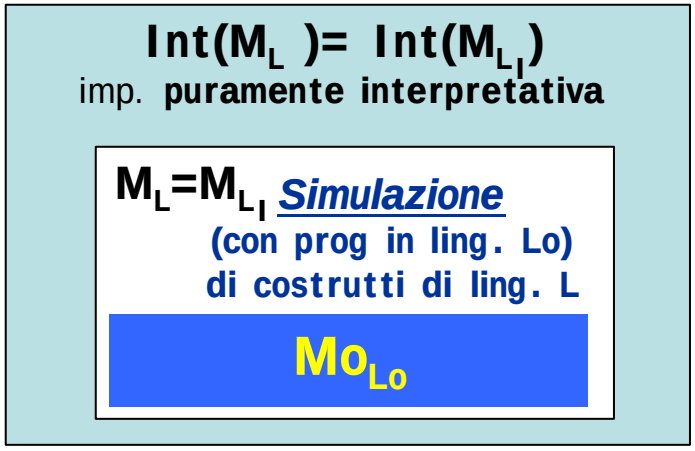


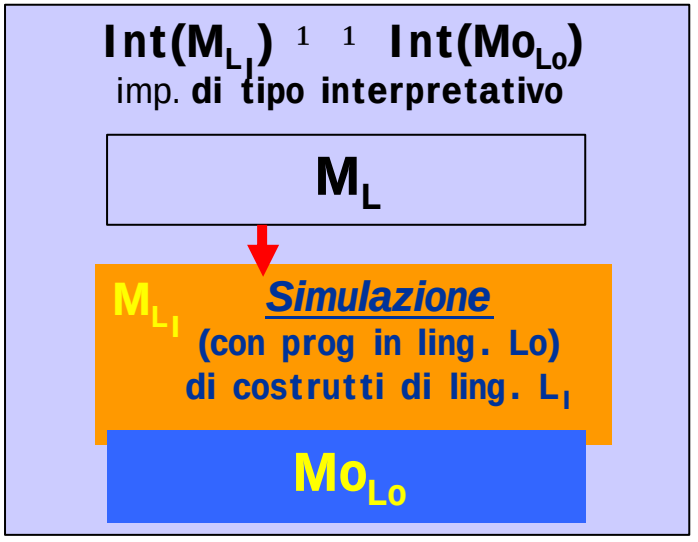
A seconda di quanto il livello intermedio sia spostato verso il livello sorgente o quello ospite si ottengono varie implementazione

Livelli di implementazione



Livelli di implementazione





Dopo la traduzione di costrutti di L in costrutti di L_I, non tutti i costrutti di L vanno simulati (interpretati): per alcuni (pochi) il corrispettivo è diretto in *Lo*.

Livelli di implementazione

$Int(M_L) = Int(M_{L_I})$
imp. **puramente interpretativa**

$M_L = M_{L_I}$ ***Simulazione***
(con prog in ling. Lo)
di costrutti di ling. L_I

Mo_{Lo}

$Int(M_{L_I}) \;^{1}\;^{1}\; Int(Mo_{Lo})$
imp. **di tipo interpretativo**

M_L

M_{L_I} ***Simulazione***
(con prog in ling. Lo)
di costrutti di ling. L_I

Mo_{Lo}

$Int(M_{L_I}) \gg Int(Mo_{Lo})$ imp.
di tipo compilativo

M_L

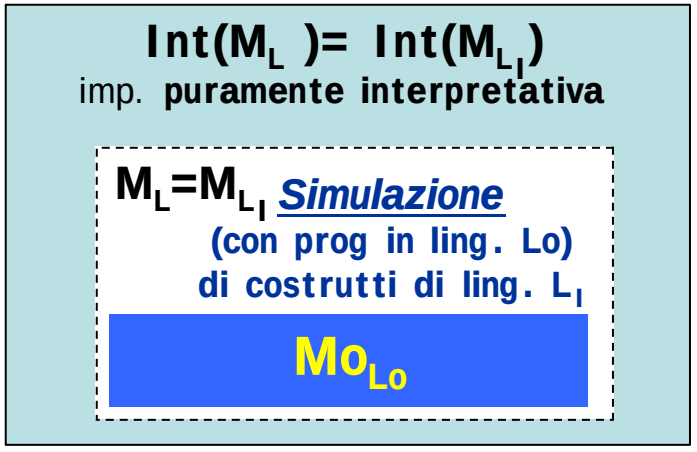
M_{L_I} ***Simulazione***

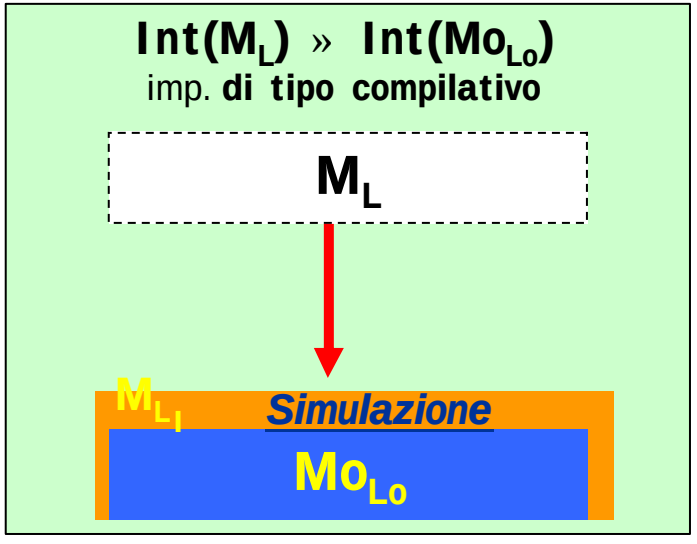
Mo_{Lo}

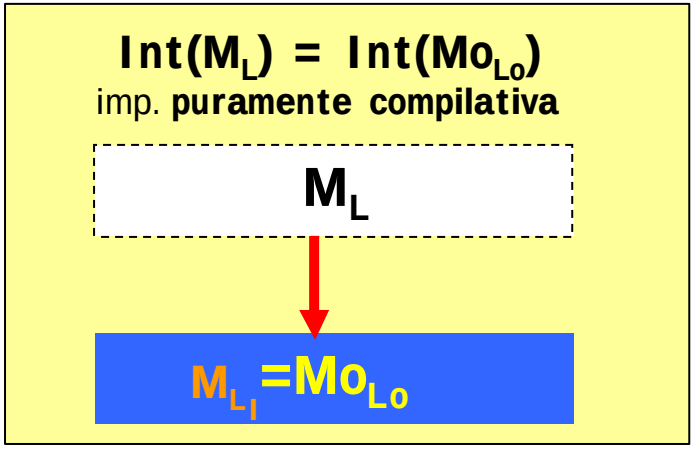
Molti costrutti di L hanno il corrispettivo in Lo:

Pochi vanno simulati (interpretati) in Lo

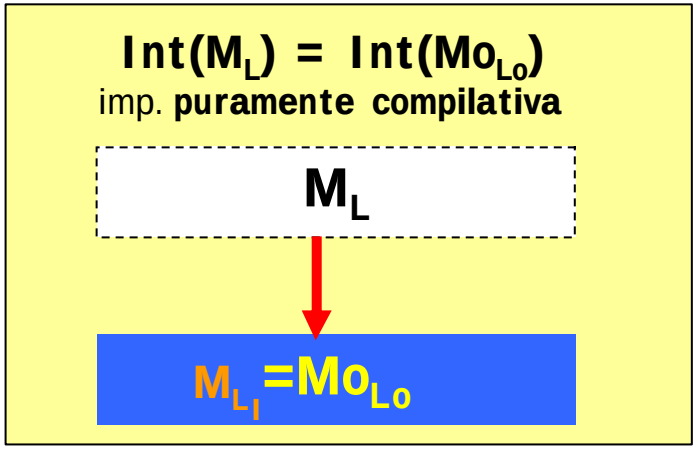
Livelli di implementazione







Compilatore



Che differenza c'è tra M_{L_I} e Mo_{Lo}?

Compilatore

Un esempio di linguaggio "antico": Il FORTRAN

Traduzione di alcuni costrutti del FORTRAN (es: I/O)

Produzione di "MOLTE" (centinaia) istruzioni in linguaggio macchina

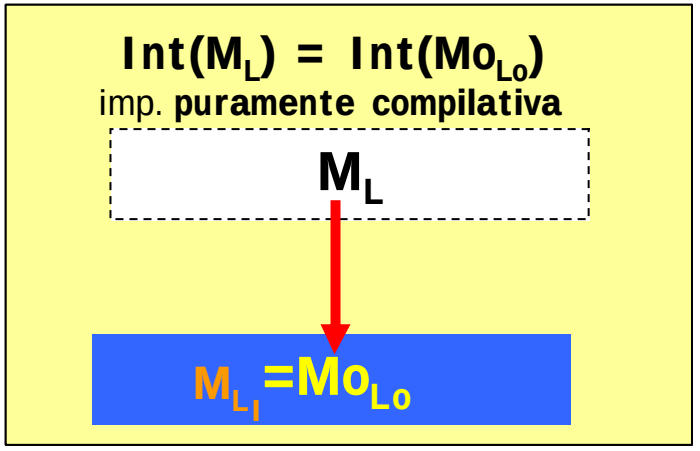
se le inserissimo nel codice compilato,
la sua dimensione crescerebbe a dismisura

in alternativa:

possiamo inserire nel codice una chiamata ad una routine (indipendente dal particolare programma)

tale routine deve essere caricata su **Mo**

Compilatore

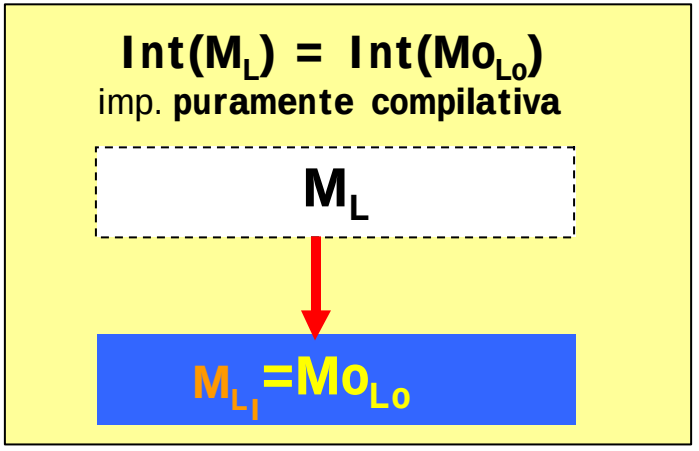


Che differenza c'è tra M_{L_I} e Mo_{Lo}?

Per i costrutti di L che non hanno un corrispettivo diretto in Lo

Es: alcune operazioni di input/output

Compilatore



Che differenza c'è tra M_{L_I} e Mo_{Lo}?

Per i costrutti di L che non hanno un corrispettivo diretto in Lo

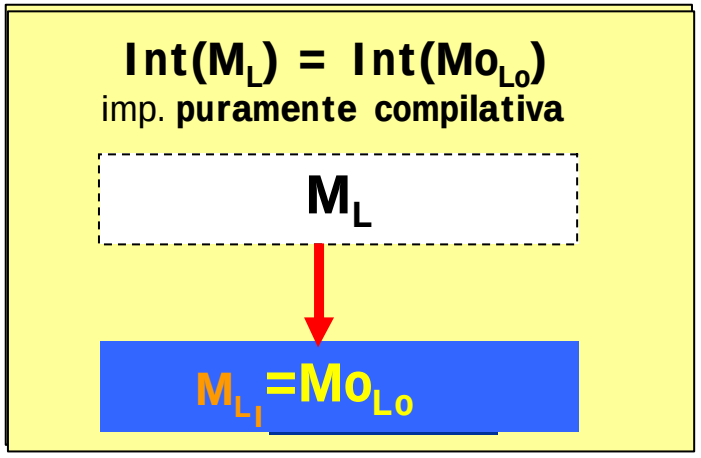
Es: alcune operazioni di input/output

Il supporto a tempo di esecuzione (rts)

Collezione di strutture dati e sottoprogrammi che devono essere caricati su Mo_{Lo} (estendono Mo_{Lo}) per permettere l'esecuzione del codice prodotto dal compilatore:

Ovvero: simulano alcune funzionalità di L_I (e quindi di L) a run time

Compilatore



Che differenza c'è tra M_{L_I} e Mo_{Lo}?

$M_{L_I} = Mo_{Lo} + rts$

il linguaggio L_I
è il linguaggio macchina di **Mo_{Lo}**
esteso con chiamate al supporto a tempo di esecuzione

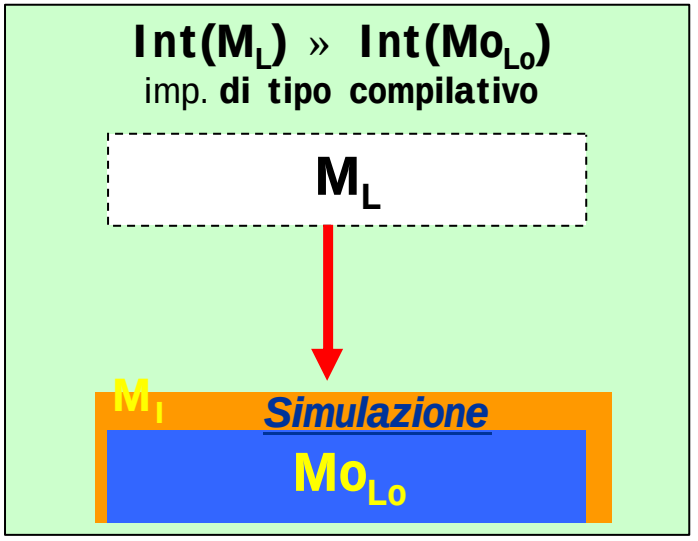
Il caso del compilatore C

Il supporto a tempo di esecuzione contiene :

- **varie strutture dati** (stack, heap)
- **i sottoprogrammi che realizzano le operazioni necessarie su tali strutture dati**

il codice prodotto è scritto in linguaggio macchina esteso con chiamate al rts

Impl. Di tipo Misto

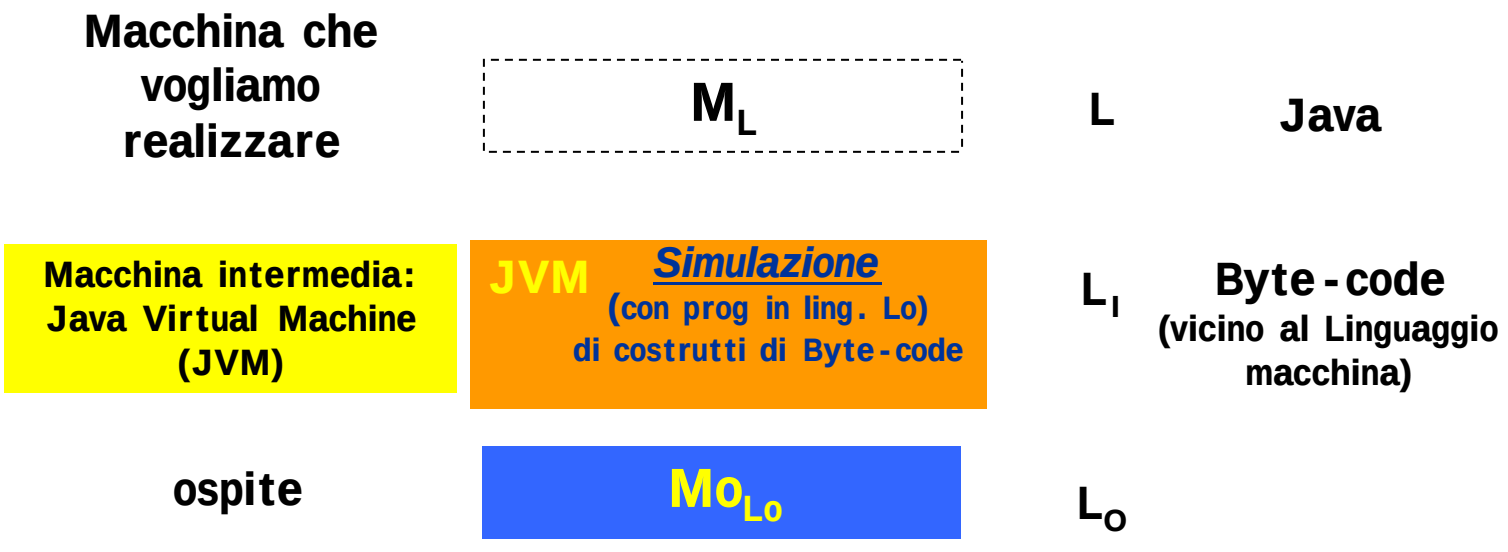


ciclo di interpretazione del linguaggio intermedio L_I realizzato su **Mo:**

- per ottenere un codice tradotto più compatto
- per facilitare la portabilità su diverse macchine ospiti
  - si deve reimplementare l'interprete del linguaggio intermedio
  - non è necessario reimplementare il traduttore

Impl. Di tipo misto: caso del Java

(simulazione mediante software)



l'interprete della Java Virtual Machine opera su strutture dati (stack, heap) simili a quelle del rts del compilatore

Compilatore o implementazione mista?

- nel compilatore non c'è di mezzo un livello di interpretazione del linguaggio intermedio
  - sorgente di inefficienza
    - la decodifica di una istruzione nel linguaggio intermedio (e la sua trasformazione nelle azioni semantiche corrispondenti) viene effettuata ogni volta che si incontra l'istruzione

Compilatore o implementazione mista?

- nel compilatore non c'è di mezzo un livello di interpretazione del linguaggio intermedio
  - sorgente di inefficienza
    - la decodifica di una istruzione nel linguaggio intermedio (e la sua trasformazione nelle azioni semantiche corrispondenti) viene effettuata ogni volta che si incontra l'istruzione
- se il linguaggio intermedio è progettato bene, il codice prodotto da una implementazione mista ha dimensioni inferiori a quelle del codice prodotto da un compilatore

Compilatore o implementazione mista?

- nel compilatore non c'è di mezzo un livello di interpretazione del linguaggio intermedio
  - sorgente di inefficienza
    - la decodifica di una istruzione nel linguaggio intermedio (e la sua trasformazione nelle azioni semantiche corrispondenti) viene effettuata ogni volta che si incontra l'istruzione
- se il linguaggio intermedio è progettato bene, il codice prodotto da una implementazione mista ha dimensioni inferiori a quelle del codice prodotto da un compilatore
- un'implementazione mista è più portabile di un compilatore

Compilatore o implementazione mista?

- nel compilatore non c'è di mezzo un livello di interpretazione del linguaggio intermedio
  - sorgente di inefficienza
    - la decodifica di una istruzione nel linguaggio intermedio (e la sua trasformazione nelle azioni semantiche corrispondenti) viene effettuata ogni volta che si incontra l'istruzione
- se il linguaggio intermedio è progettato bene, il codice prodotto da una implementazione mista ha dimensioni inferiori a quelle del codice prodotto da un compilatore
- un'implementazione mista è più portabile di un compilatore
- il supporto a tempo di esecuzione di un compilatore si ritrova quasi uguale nelle strutture dati e routines utilizzate dall'interprete del linguaggio intermedio

Implementazioni miste e interpreti puri

- la traduzione genera codice in un linguaggio più facile da interpretare su una tipica macchina ospite

Implementazioni miste e interpreti puri

- la traduzione genera codice in un linguaggio più facile da interpretare su una tipica macchina ospite
- ma soprattutto può effettuare una volta per tutte (a tempo di traduzione, staticamente) analisi, verifiche e ottimizzazioni che migliorano
  - l'affidabilità dei programmi
  - l'efficienza dell'esecuzione

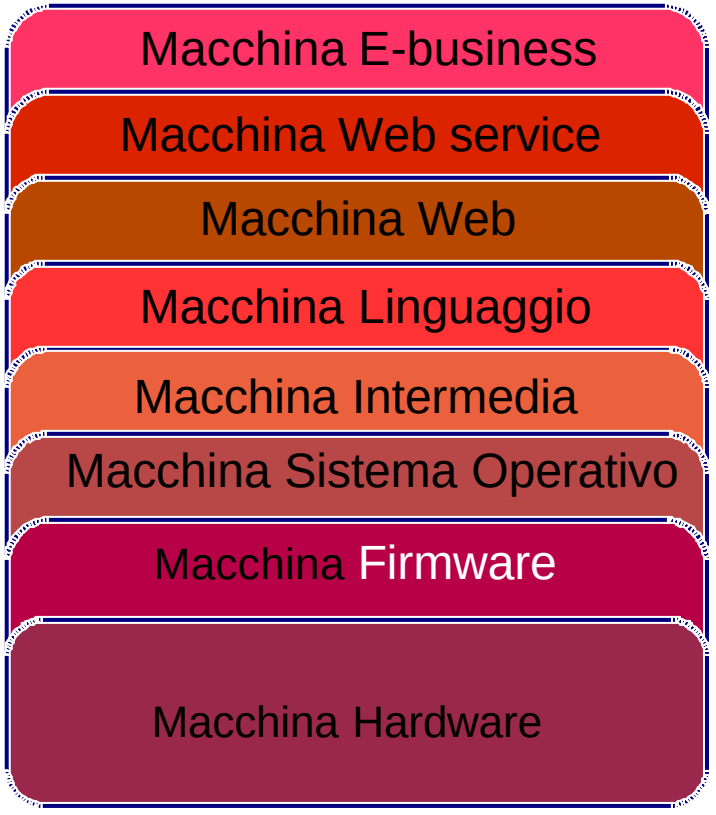
Implementazioni miste e interpreti puri

- la traduzione genera codice in un linguaggio più facile da interpretare su una tipica macchina ospite
- ma soprattutto può effettuare una volta per tutte (a tempo di traduzione, staticamente) analisi, verifiche e ottimizzazioni che migliorano
  - l'affidabilità dei programmi
  - l'efficienza dell'esecuzione
- varie proprietà interessate
  - inferenza e controllo dei tipi
  - controllo sull'uso dei nomi e loro risoluzione "statica"
  - ....

Esempi

- interprete puro
  - $\mathbf{M_L} = \mathbf{M_{L_I}}$
  - interprete di **L** realizzato su $\mathbf{M_O}$
  - alcune implementazioni (vecchie!) di linguaggi logici e funzionali
    - LISP, PROLOG
- compilatore
  - macchina intermedia $\mathbf{M_{L_I}}$ realizzata per estensione sulla macchina ospite $\mathbf{M_O}$ (rts, nessun interprete)
    - C, C++, PASCAL
- implementazione mista
  - traduzione dei programmi da **L** a $\mathbf{L_I}$
  - i programmi $\mathbf{L_I}$ sono interpretati su $\mathbf{M_O}$
    - Java
    - i "compilatori" per linguaggi funzionali e logici (LISP, PROLOG, ML)
    - alcune (vecchie!) implementazioni di Pascal (Pcode)

Gerachie di Macchine



Ogni macchina Mi_L è implementata su $Mi\text{-}1_L$ (utilizza funzionalità di $Mi\text{-}1_L$)e ne offre a quella di livello superiore $Mi+1_L$

1. Permette di dominare la complessità
2. Consente l'indipendenza dei vari livelli