

# *Algoritmi e Linguaggi*

---

# *Programmi e linguaggi*

---

- Un calcolatore è solo un esecutore rapidissimo di istruzioni
- Un programma è un insieme di istruzioni codificate in un opportuno linguaggio

# *Linguaggi e algoritmi*

---

- Prima di scrivere un programma si deve
  1. generalizzare il più possibile la procedura
  2. assicurarsi che termini sempre
  3. identificare tutte le possibili configurazioni iniziali
  4. precisare in maniera univoca i passi da compiere
  5. accertarsi che un calcolatore possa eseguirli

# *Algoritmo*

---

- Le precedenti caratteristiche portano agli algoritmi: un algoritmo è una procedura
  1. generale
  2. finita
  3. completa
  4. non ambigua
  5. eseguibile

# *Progettare gli algoritmi*

---

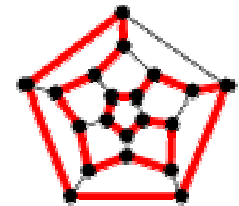
- Non esiste un algoritmo per progettare algoritmi!
- Due approcci:
  - *top-down*: dal generale al particolare
  - *bottom-up*: dal particolare al generale
- Non tutti i problemi sono calcolabili
  - e.g.: un programma si fermerà sempre?

# *Complessità degli algoritmi*

---

- Tra i problemi calcolabili ce ne sono alcuni intrinsecamente più difficili, ad esempio

- trovare un percorso minimo per visitare un insieme di città una sola volta



- verificare la correttezza di un circuito logico
- trovare il minimo numero di colori per una mappa

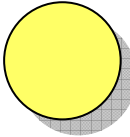
# *Diagrammi di flusso*

---

- Un algoritmo può essere descritto usando un diagramma di flusso
- Un diagramma di flusso è costituito da
  - blocchi di base
  - strutture per comporre i blocchi

# *Blocchi di base*

---

<b>Rappresentazione</b>	<b>Funzione</b>
	inizio e fine
	istruzione
	ingresso / uscita

# *La programmazione strutturata*

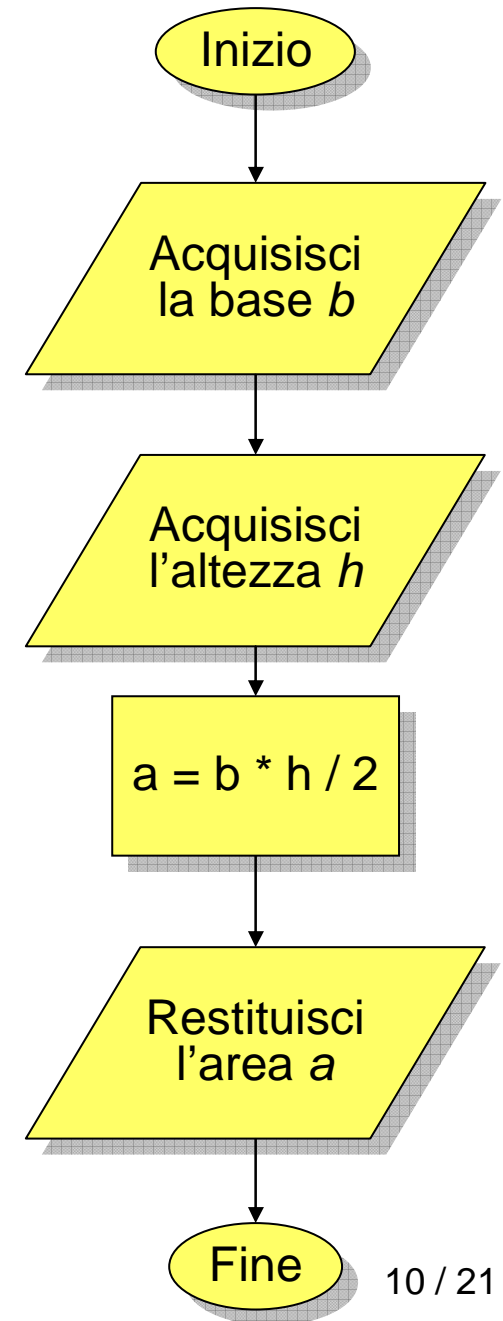
---

- Un qualsiasi algoritmo può essere espresso combinando i blocchi di base mediante
  - sequenza
  - selezione
  - ripetizione o iterazione
  
- Si ottiene un diagramma di flusso strutturato

# La sequenza

---

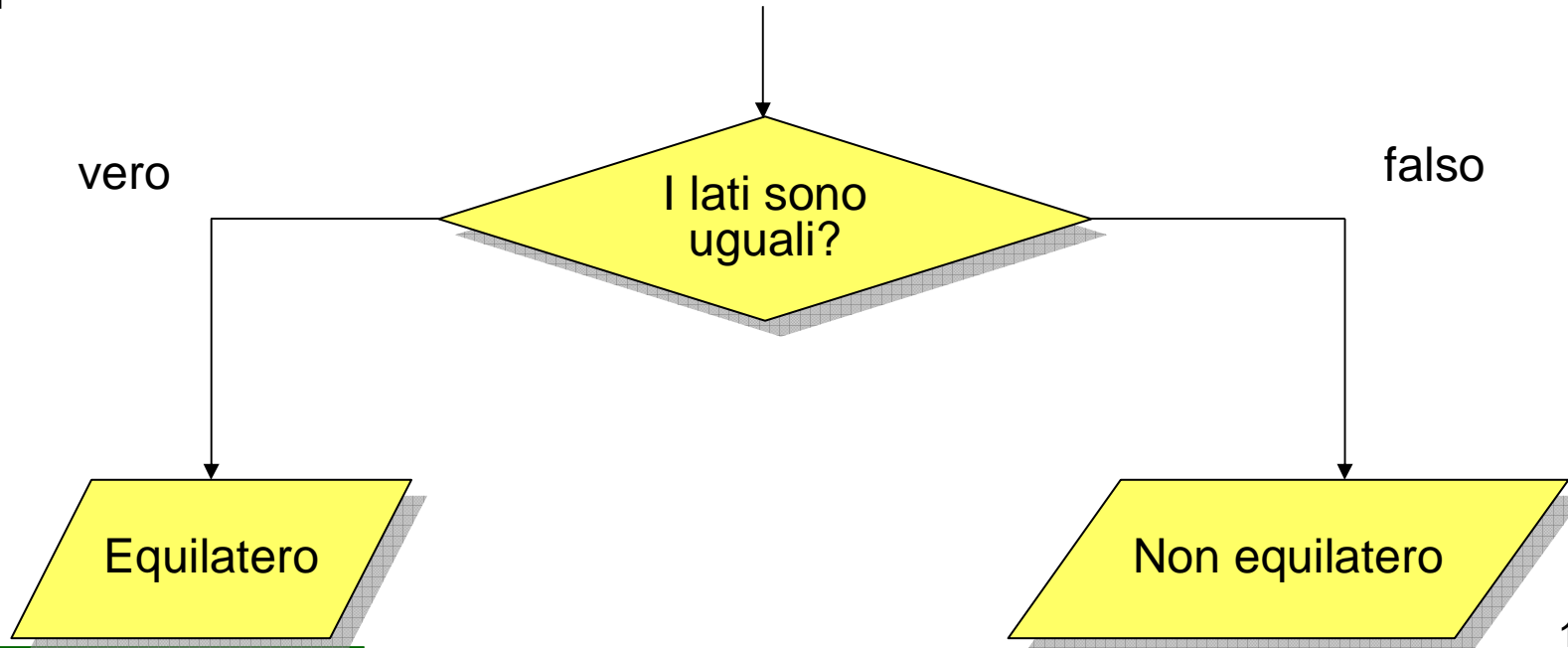
- È semplicemente un insieme di blocchi che si susseguono
- *E.g.*: l'area di un triangolo



# *La selezione*

---

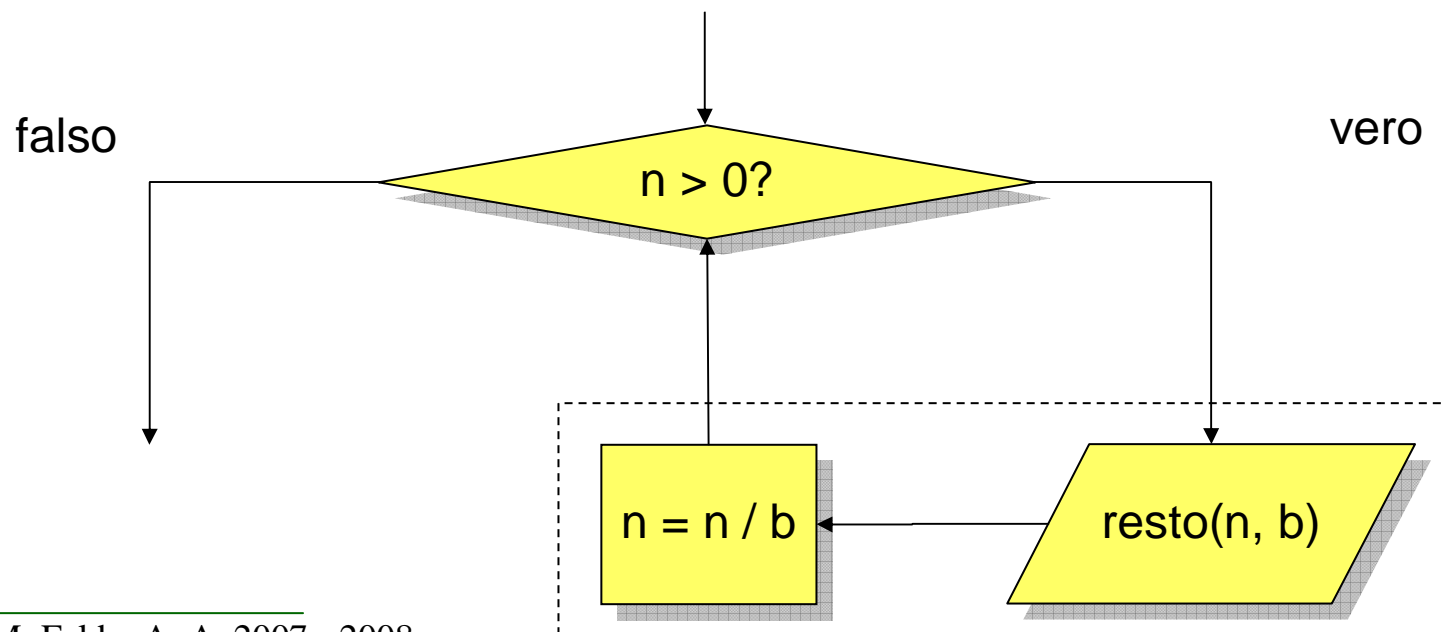
- Permette di sdoppiare l'esecuzione in dipendenza di una condizione



# *L'iterazione*

---

- Serve a ripetere una serie di istruzioni
  - fa uso della selezione per terminare



# *Linguaggi naturali e formali*

---

- Un linguaggio naturale è ambiguo

*“una vecchia porta la sbarra”*

- Un linguaggio formale non lo è mai
  - grammatiche formali

# *Il linguaggio macchina e l'assemblatore*

---

- Anche le istruzioni sono codificate con sequenze di bit
- Si possono associare simboli mnemonici, ma rimangono
  - la complessità
  - la mancanza di astrazione

# *I linguaggi ad alto livello*

---

- Permettono di
  - astrarre le caratteristiche del calcolatore
  - generalizzare la scrittura di procedure ricorrenti
- Caratteristiche
  - sintassi semplice, familiare al programmatore
  - più sono “lontani” dal calcolatore e meno sono efficienti: compromesso tra velocità e versatilità

# *Traduzione*

---

- Ogni linguaggio deve essere ritradotto in linguaggio macchina
- Tre metodi
  - compilazione: C++
  - interpretazione: Javascript
  - pseudo-codice: Java

# *Tipologie di linguaggi ad alto livello*

---

- Imperativi: si specificano le istruzioni da eseguire
  - C, Pascal
- Ad oggetti: si modella il problema in termini di interazioni tra oggetti
  - C++, Java
- Funzionali: si formula il problema in termini di funzioni
  - Lisp, ML
- Dichiarativi: si descrive il problema in termini logici lasciando al calcolatore il compito di risolverlo
  - Prolog, XSLT

# *Determinismo*

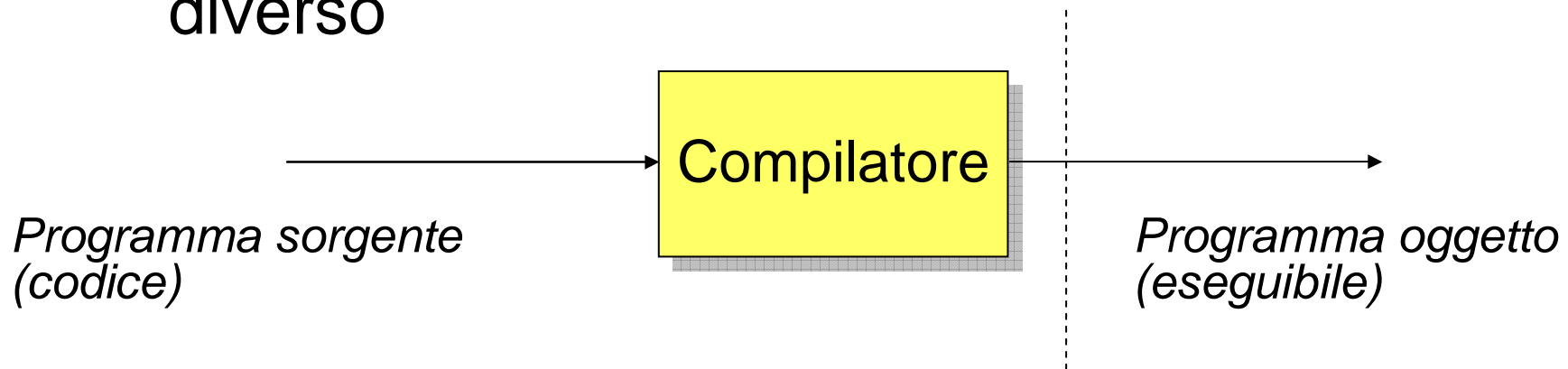
---

- Un algoritmo è deterministico se ad ogni configurazione di ingresso corrisponde un unico percorso di esecuzione
- Algoritmi non deterministici
  - algoritmi casuali
  - approcci dichiarativi

# Compilazione

---

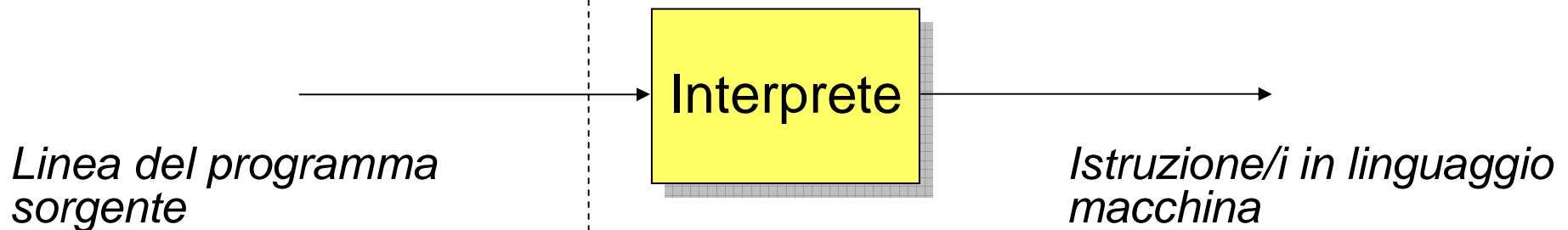
- Il codice viene tradotto una volta per tutte
  - efficiente
  - deve essere ricompilato per ogni calcolatore diverso



# *Interpretazione*

---

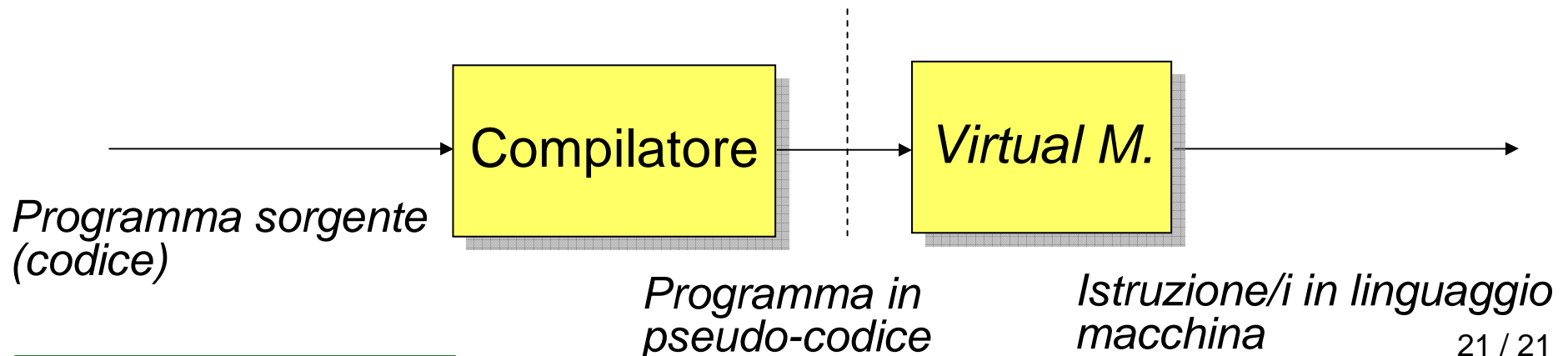
- Il codice viene tradotto linea per linea ogni volta
  - lento
  - *portabile* immediatamente su calcolatori diversi
  - permette la *meta-programmazione*



# *Pseudo-codice*

---

- Il codice viene tradotto in un linguaggio intermedio “universale”
  - abbastanza lento
  - *portabile* immediatamente su calcolatori diversi



# *Il problema della fermata*

---

1. Sia  $P(x)$  un programma che mi dice se il programma  $x$  termina (vero o falso)
2. Aggiungo alla fine di  $P$  un programma  $Q(y)$  che termina solo se  $y = P(x)$  è falsa
3. Se ora pongo  $x = P$  ottengo una contraddizione, infatti
  1. se  $P$  termina  $P + Q$  non termina
  2. se  $P$  non termina  $P + Q$  termina