# *Introduzione alla OOP*
# *Object Oriented Programming*

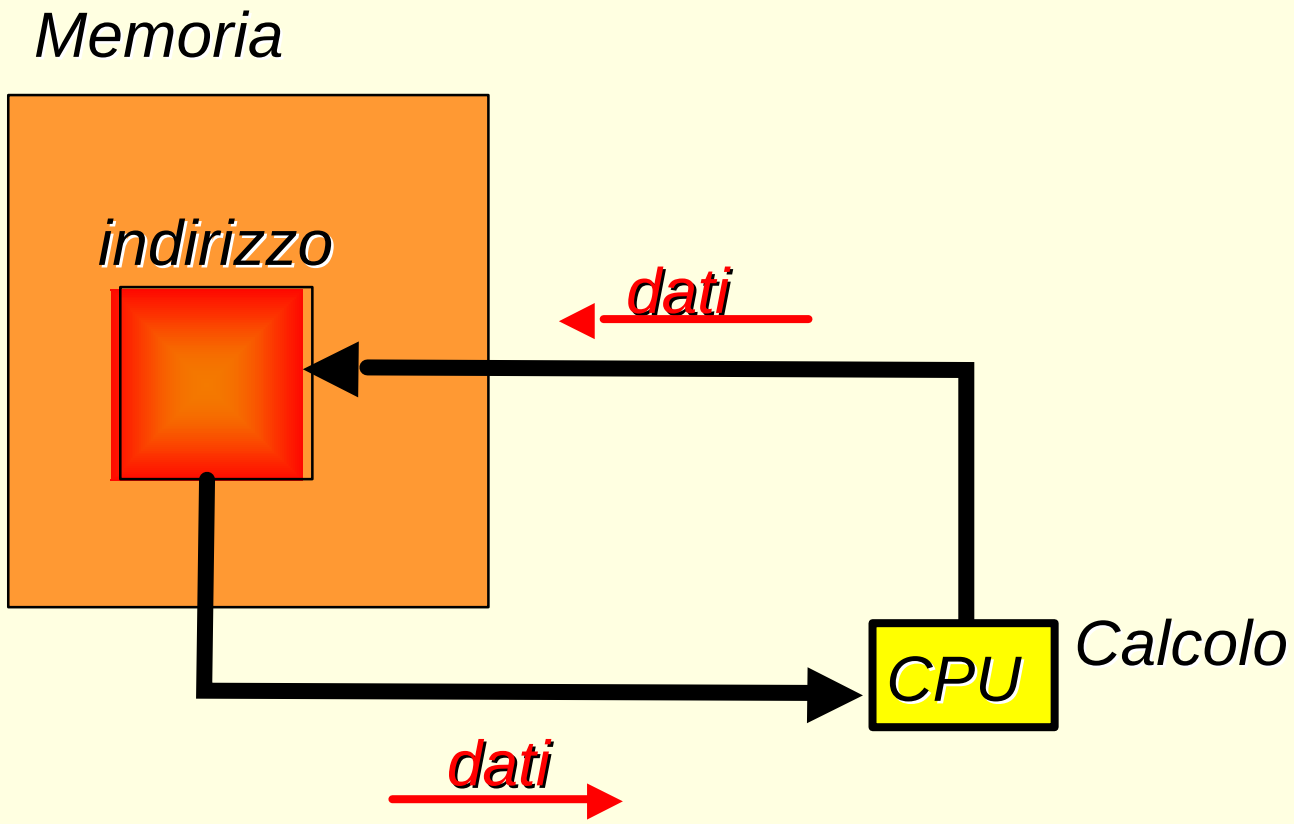
*Programmazione Orientata agli Oggetti*

# *I livelli dei linguaggi*

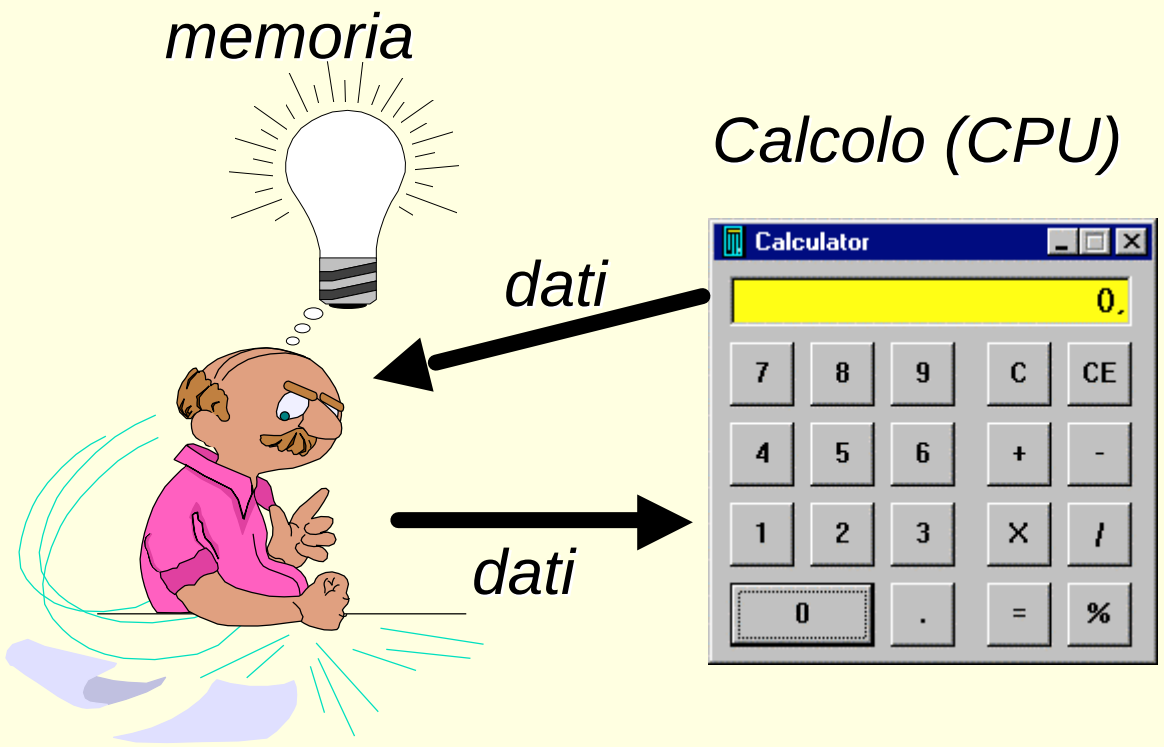
- *livelli di tensione*
- *porte logiche*
- *codice binario*
- *linguaggio assembler*

- *linguaggi procedurali*
- *linguaggi ad oggetti*
- *componenti*

# L'Architettura di von Neumann

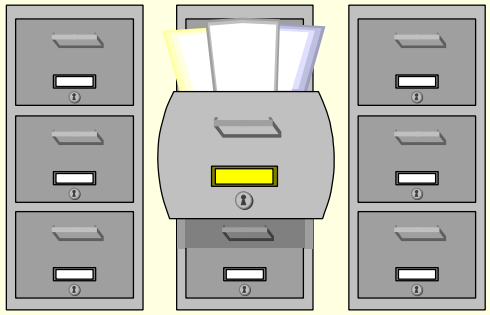


# *La programmazione procedurale separa il calcolo dalla memoria*



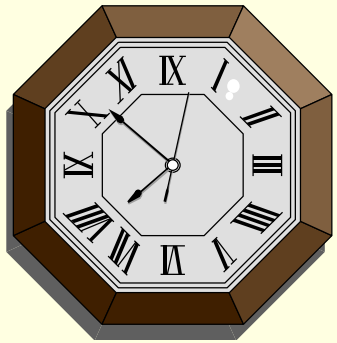
# *L'alternativa fondamentale*

***Spazio** per Ricordare*



| A | B | C | F (A,B,C) |
|---|---|---|---|
| 1 | 1 | 1 | 12 |
| 2 | 2 | 2 | 24 |
| 0 | 1 | 0 | 4 |

***Tempo** per Calcolare*

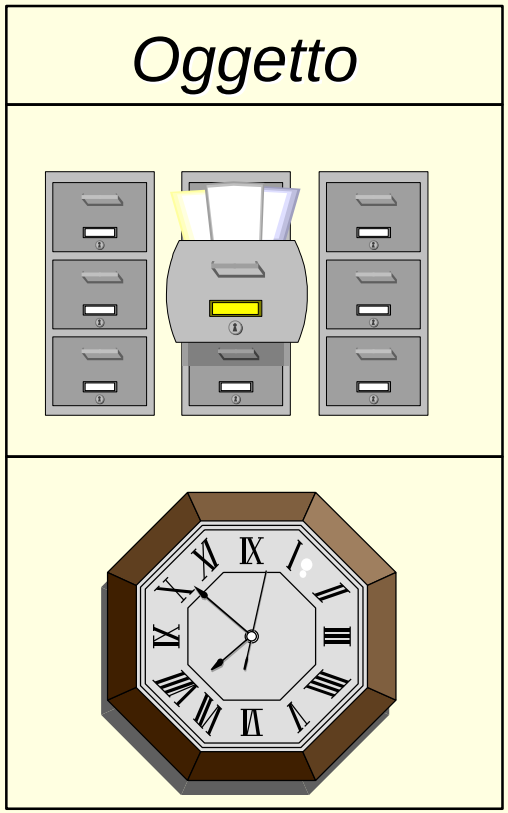


$F(A,B,C) = 3A+4B+5C$

# *L'alternativa Spazio-Tempo*

| *Spazio* | *Tempo* |
|---|---|
| *? = 3.14159* | *? = C / 2R* |
| *Tavole degli Integrali* | *Integrazione per parti* |
| *Tabelle per le tasse* | *Formule per aliquote* |

# *Gli Oggetti Incorporano Spazio e Tempo*




*Attributi*

*Metodi (Funzioni o Procedure)*

*Gli Oggetti rivoluzionano l'architettura di von Neumann !*

# *Gli oggetti aiutano a modellare il mondo reale*

| *Biglia* |
| --- |
| *peso*<br>*colore* |
| *rotola* |

| *Candela* |
| --- |
| *altezza*<br>*diametro*<br>*colore* |
| *brucia* |

| *Automobile* |
| --- |
| *porte*<br>*cilindrata* |
| *avanza*<br>*retrocedi* |

| *Pietra* | Identità |
| --- | --- |
| *massa*<br>*colore* | Attributi |
| *rotola*<br>*cadi* | Metodi / Operazioni |

# *Perché la modellazione Object-Oriented ?*

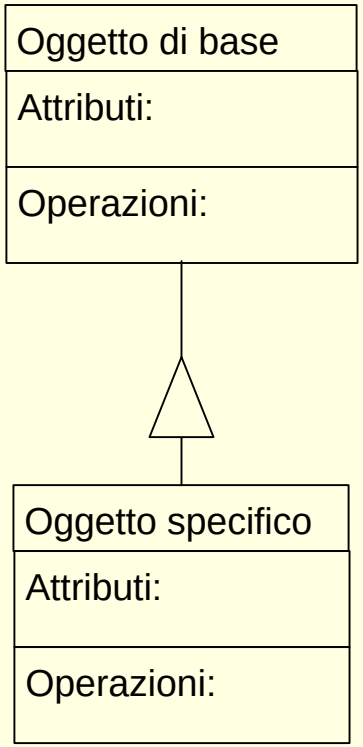
- *Non è solo programmazione: è una diversa maniera di pensare e progettare il software*
- *Invece di modellare la macchina (procedurale) si cerca di modellare il problema (object oriented)*
- *Utile nella progettazione software*
- *E' la struttura utilizzata per la maggior parte del software moderno*
- *Uno strumento potente per trattare problemi complessi, dove i requisiti variano nel tempo.*

# *Object Oriented Modeling*

- Gli Oggetti hanno  Stato, Comportamento ed Identità
  - *Stato (spazio)  ---  Comportamento (tempo)*
- La Persistenza Conserva lo Stato
- Concorrenza: permette a molti oggetti di agire in parallelo
- Incapsulare = Nascondere il dettaglio della realizzazione
- Astrazioni diverse omettono dettagli differenti!
  - *L'astrazione si focalizza sulle caratteristiche essenziali di un dato oggetto, relativamente alla prospettiva di chi osserva ...*

# *Le Astrazioni Formano una Gerarchia*

***" IS A "***
***Ereditarietà***



# *Classe: modello astratto da cui creare oggetti*

*Una classe è la definizione di un insieme di oggetti che condividono una struttura ed un comportamento comuni*

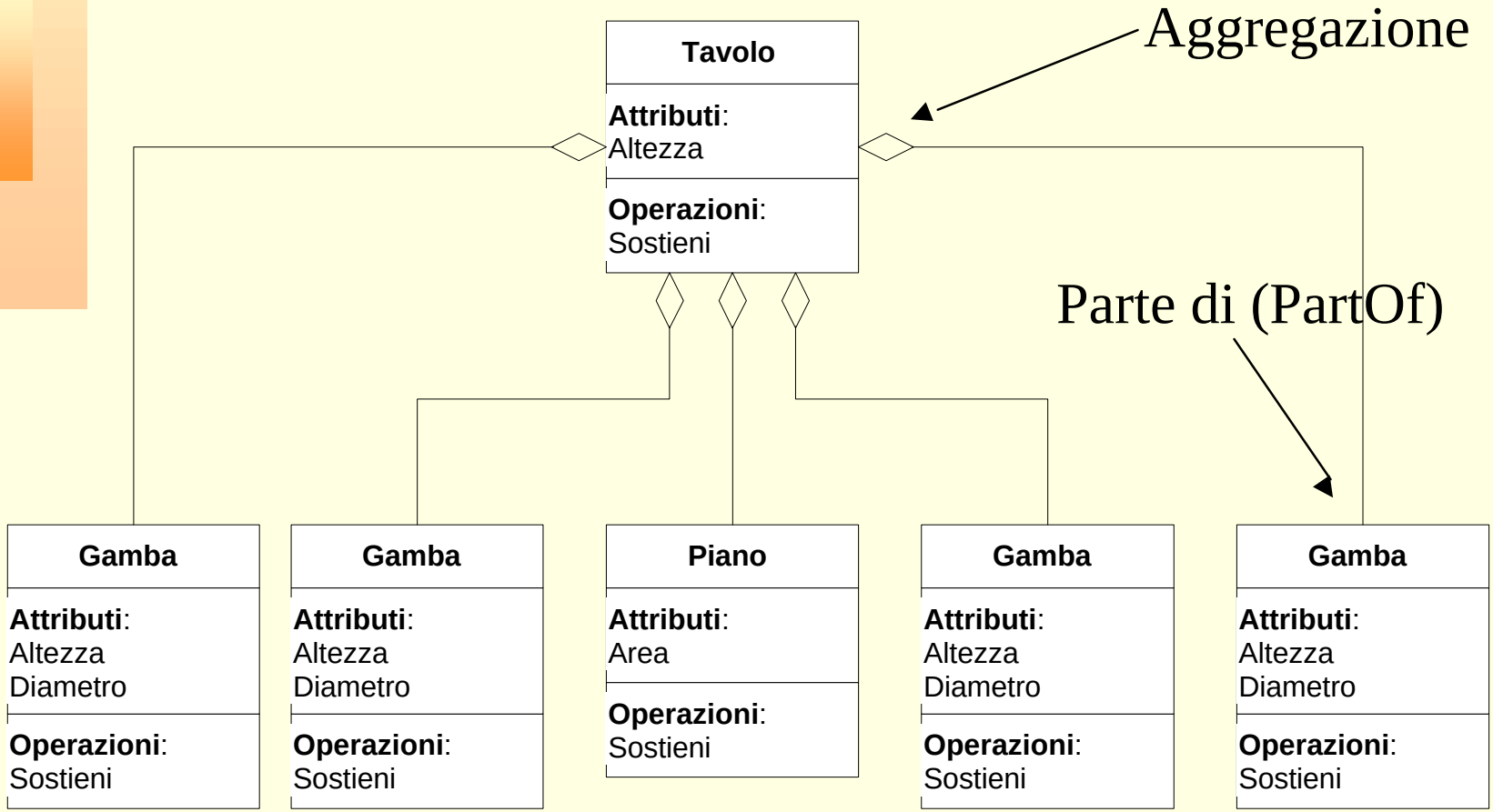
# *Classi, oggetti e relazioni*

*Gli oggetti sono istanze delle classi e sono in relazione tra di loro*

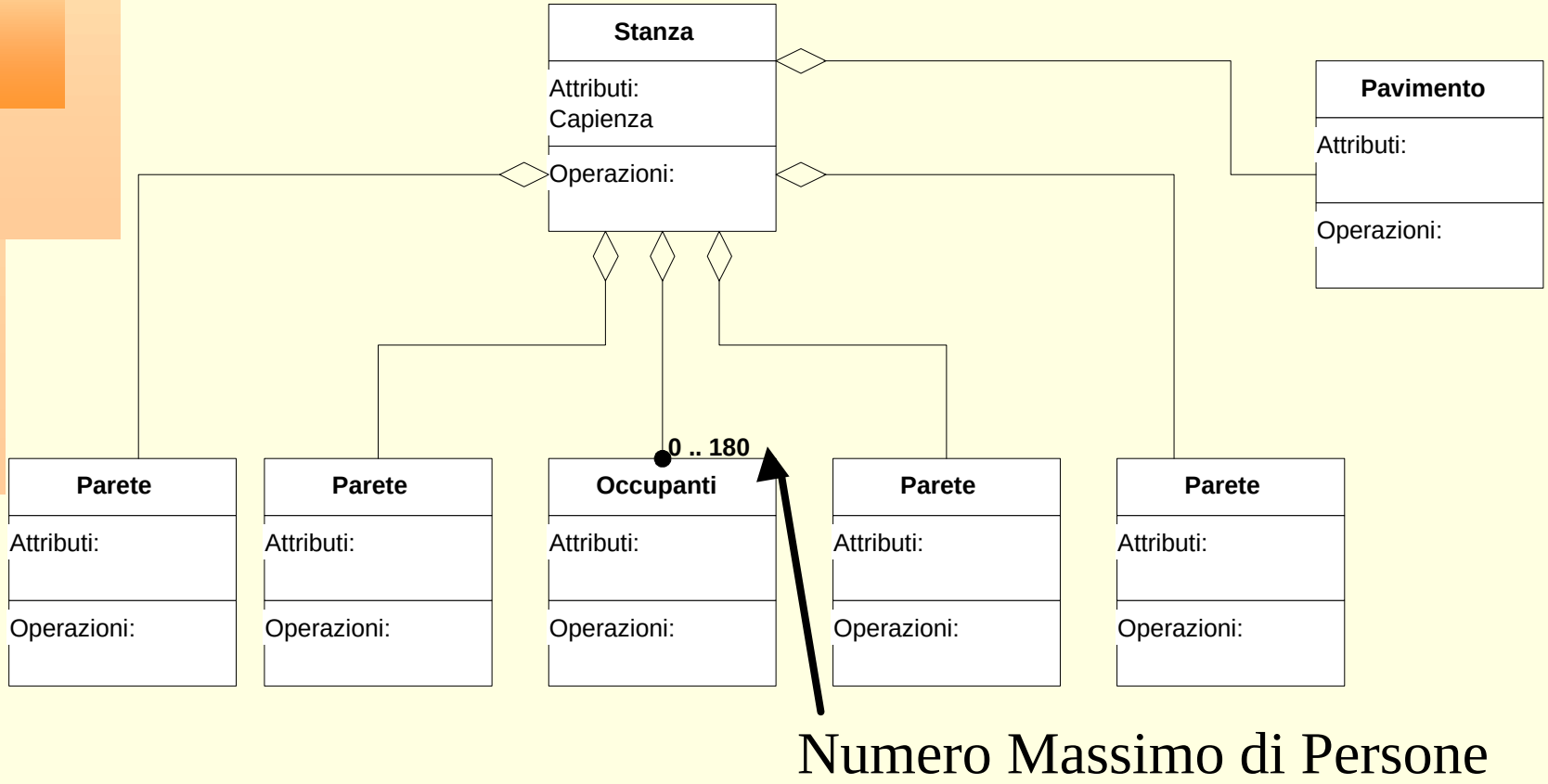
# *Oggetti che "Contengono" oggetti*

- *Un Tavolo ha parti (piano, 4 gambe)*
  - *Rimuovendo la superficie o una delle gambe non è più un tavolo (non dà più un supporto)*
  - *Perciò Piano e Gambe sono PARTOF del tavolo*
- *Stanza: ha come parti pareti e pavimento*
  - *Senza pareti e pavimento resta solo uno spazio*
- *Persone: sono "Contenute" in una stanza*
  - *Senza persone la stanza rimane una stanza (anche se vuota).*

# *Diagrammi ad oggetti (UML)*



# *Diagrammi ad oggetti (UML)*



# *Diagrammi ad Oggetti (UML) Riassunto*

- *Aggregazione (part of e contiene -- rombo) raggruppa oggetti per costruirne altri più complessi*
- *Ereditarietà (IsA o KindOf -- triangolo) gestisce i diversi livelli di astrazione*
- *Associazioni (varie -- linea con semicerchio) relazioni tra oggetti legate al contesto del problema descritto*

# *Perchè usare oggetti?*
# *Gestire la complessità*

*Il compito del programmatore è nascondere la complessità.*

# *La crisi del software(anni 90): Perché usare metodi Object-oriented*

- *I programmi applicativi sono sempre più complessi (per gli sviluppatori e utenti)*
- *Solo il 5% dello sviluppo software produce dei sistemi funzionanti*
  - *abbandonati dopo la consegna (tarda o incompleta)*
  - *mai completati (costi o tempi eccessivi)*
  - *obsoleti quando consegnati*
- *I requisiti che cambiano impongono programmi che possono cambiare*

# *L'approccio standard: Programmazione Modulare (Tempo)*

- *Programmazione modulare (1950)*
  - *Dividere gli algoritmi in parti governabili*
  - *Usare subroutines per dividere il codice*
- *Programmazione strutturata (1960)*
  - *Decomposizione funzionale top-down*
  - *Difficoltà nel trovare la miglior decomposizione*
- *Strumenti di sviluppo - CASE (1985)*
  - *Automazione parziale ma costrizioni*
- *Linguaggi DB di 4th Generazione (1990)*
  - *Efficaci soprattutto per piccoli database*

# *L'approccio standard: Modularizzazione dei Dati (Spazio)*

- *Aspetto spesso trascurato per fretta*
- *Dati condivisi (locali o globali)*
  - *difficoltà nel "contenere" un dato nel suo ambito*
  - *dati globali sono comodi ma vanno contro i principi della programmazione modulare*
  - *dati locali permettono "information hiding"*
- *Dati in files su disco*
  - *va bene per grandi quantità di dati*
  - *se condivisi l'integrità è difficile da gestire*

# *Perchè usare Oggetti?*

- *Nascondono la scelta Tempo-Spazio: un oggetto per le tasse potrebbe usare tabelle, calcoli o entrambi.*
- *Definiscono parti riusabili -- come i componenti standard per auto*
- *Si possono sostituire delle parti di un sistema con altre più efficienti*
- *Più adatti per lo sviluppo iterativo*

# *Analisi e Progettazione*

- *Metodologia*
  - *Identificare gli oggetti (come suddividere il problema in parti)*
  - *Identificare le operazioni e le proprietà per ciascun oggetto*
  - *Identificare le interazioni tra oggetti*