Università degli Studi di Trieste

Corso di Laurea Magistrale in
INGEGNERIA CLINICA

UNIFIED MODELING LANGAUGE BASICS

Corso di Informatica Medica

Docente Sara Renata Francesca MARCEGLIA

Dipartimento di Ingegneria e Architettura

UNIVERSITÀ
DEGLI STUDI DI TRIESTE

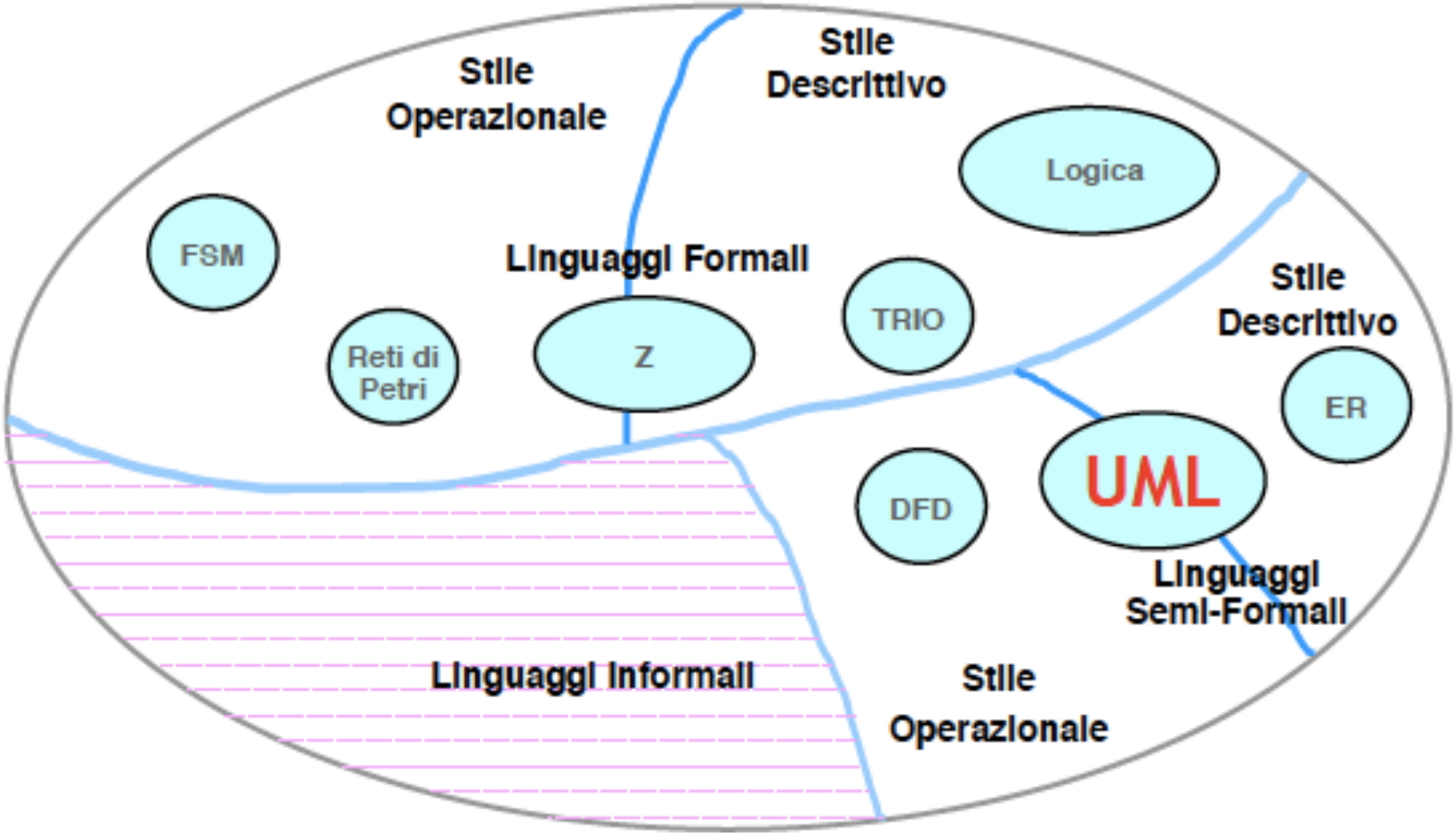
WHAT IS UML

"The **Unified Modeling Language** (UML) is a language for specifying, visualizing, constructing, and documenting the **artifacts of software systems**, as well as for **business modeling** and other non-software systems".

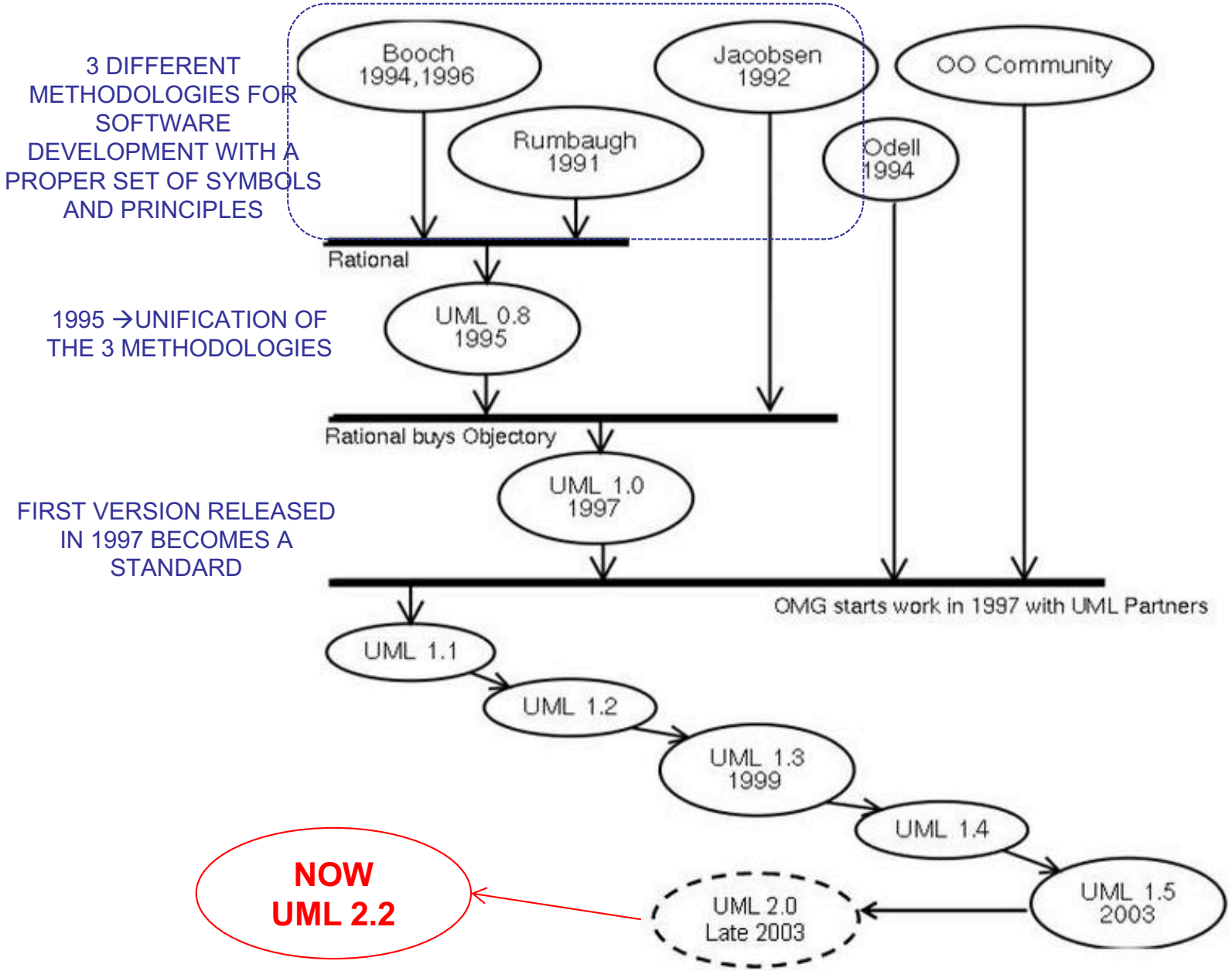
WHAT IS UML

- UML is **a modeling language**
- UML is **not a method**
- Describes and specifies the features of a new or existing system
- UML is a way to **enable communication** among different actors involved in the system development (from committment to users)
- UML uses a **graphical notation**
- UML specifications are given by the Object Management Group (OMG)

UML E NOTAZIONI



HISTORY OF UML



OBIETTIVI DI UML

- UML è costituito da un insieme di linguaggi che consentono di dare descrivere e modellare diversi aspetti di un sistema:
  - Aspetti statici
  - Aspetti dinamici
  - Interazioni tra le diverse componenti

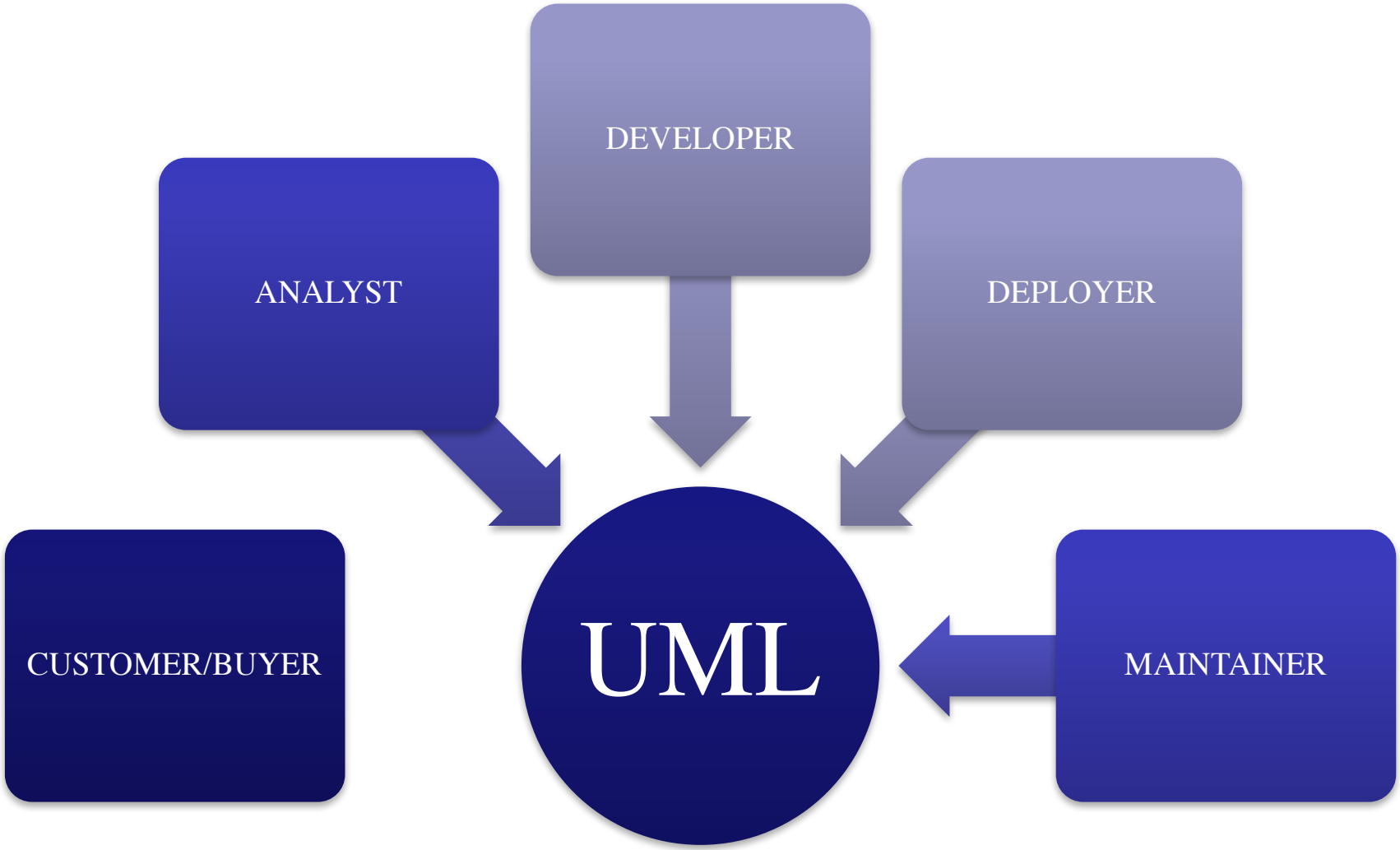
- UML facilita la **comunicazione→**
  - tra analisti e esperti di dominio
  - Tra analisti e progettisti
  - Tra progettisti e sviluppatori

- Linguaggio grafico ma formale, preciso e non ambiguo
- Dà una visione concisa del sistema da diversi punti di vista
- Non necessita di conoscenza del codice

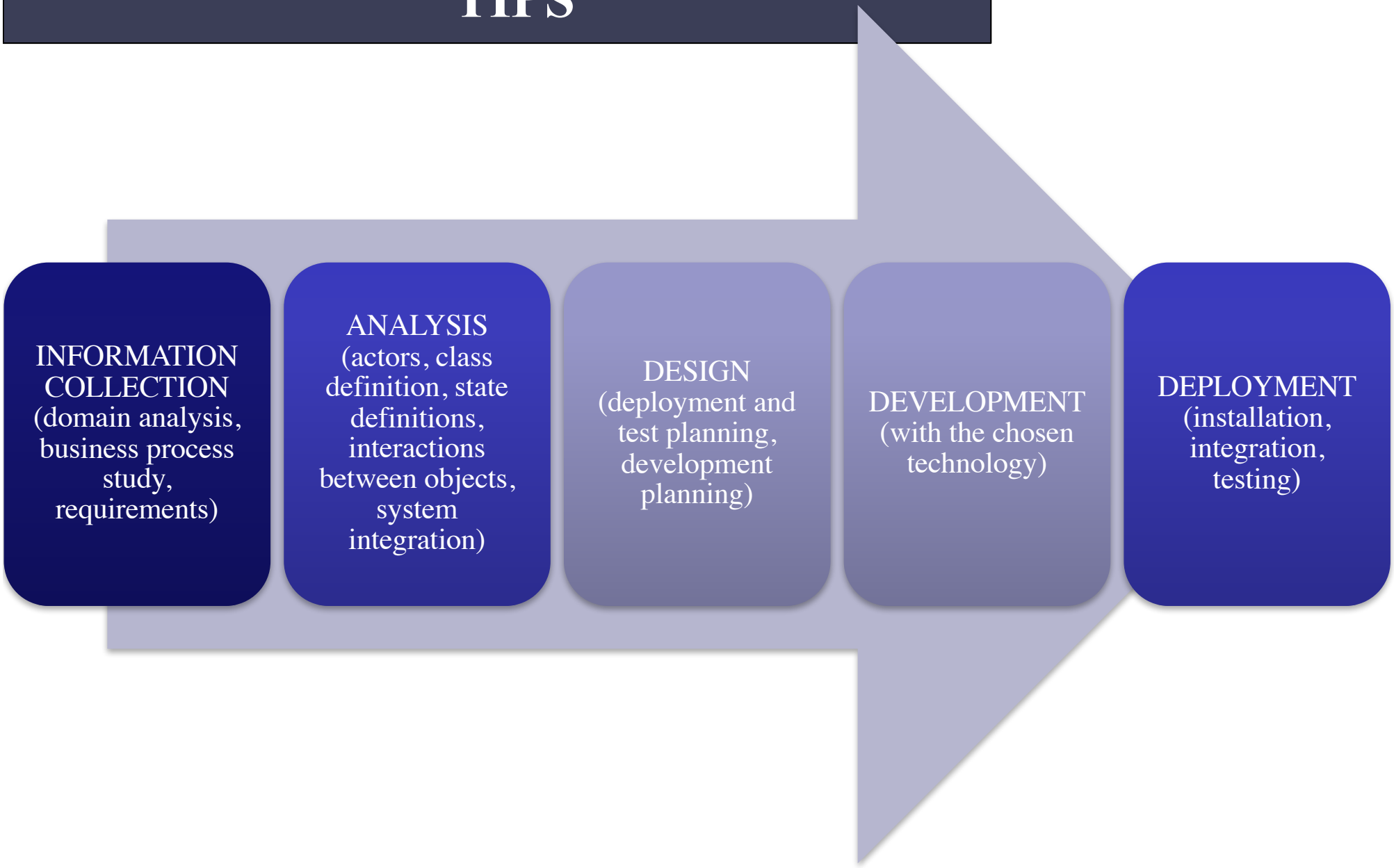
PERCHÈ SI USA UML

- Si deve operare con esperti di domini
- Si devono catturare gli aspetti più importanti senza perdersi nei dettagli
- La notazione di UML è flessibile → molti diagrammi rappresentano aspetti diversi dei concetti
- La riunione di tutti i diagrammi prodotti può essere facilmente utilizzata come rappresentazione del sistema intero → scheletro del sistema

ACTORS OF THE SOFTWARE DEVELOPMENT PROCESS



DEVELOPMENT METHODOLOGY: TIPS



WHY USING UML IN HEALTHCARE

- **Hospitals** are complex context, populated by different processes and actors →

  - Tools are needed to enable the communication between natural language (physicians) and programming language (developers)
  - It is important to map and manage diverse and time evolving interactions among different actions.
  - UML, being a graphical language, might be helpful in immediate communication between physicians and developers

DIAGRAMS: CLASSIFICATION BY SYSTEM VIEWS

STATIC VIEWS

- Use case diagrams
- Class diagram
- Component diagram
- Deployment diagram
- Object diagram

DYNAMIC VIEWS

- Sequence diagrams
- Collaboration diagrams
- State chart diagrams
- Activity diagrams

DIAGRAMS: CLASSIFICATION BY REPRESENTATION

UNIVERSITÀ
DEGLI STUDI DI TRIESTE

STRUCTURAL DIAGRAMS

- Class diagrams
- Object diagrams
- Component diagrams
- Composite structures
- Deployment diagrams
- Package diagrams

BEHAVIOURAL DIAGRAMS

- Use case diagrams
- State chart diagrams
- Activity diagrams

INTERACTION BEHAVIOURS

- Sequence diagrams
- Communication diagrams
- Timing diagrams
- Interaction diagrams
- Overviews

DIAGRAMMI E OBIETTIVI

Diagramma	Obiettivo
Use case diagram	Estrae i requisiti dagli utenti in modo sinettico La progettazione della fase di costruzione avviene introno a ciascun use case (per ogni iterazione) Base del testing del sistema
Class diagram	Rappresenta la struttura statica dei concetti, tipi e classi Concetti: entità che rappresentano il dominio Tipi: interfacce di componenti software Classi: implementazioni dei componenti software
Activity diagram	Rappresenta il comportamento controllato Può rappresentare un solo oggetto in più casi d'uso o più oggetti in un singolo caso d'uso
Interaction diagram	Rappresenta come diversi oggetti interagiscono/collaborano in un singolo use case
Deployment diagram	Rappresenta le componenti fisiche su nodi hardare

USE CASE DIAGRAMS (1/2)

- Schematizza il comportamento del sistema dal punto di vista degli utenti, o, pi´u in generale, di altri sistemi che interagiscono col sistema specificato.

- **Use case = servizio o attività messa a disposizione dal sistema**
- **Scenario = insieme di casi d'uso che descrivono un'interazione utente/sistema**
- **Attore = entità esterna al sistema**

- Attori e casi d'uso sono legati da associazioni che rappresentano comunicazioni.

- I casi d'uso non rappresentano sottosistemi, per cui non possono interagire o scambiarsi messaggi fra di loro, ma possono essere legati da relazioni.

- Non definiscono il comportamento richiesto al sistema per fornire il servizio rappresentato

RELAZIONI TRA CASI D'USO

INCLUSIONE

- Use case inserito in un altro use case
- Viene eseguito sempre quando lo use case di riferimento è eseguito

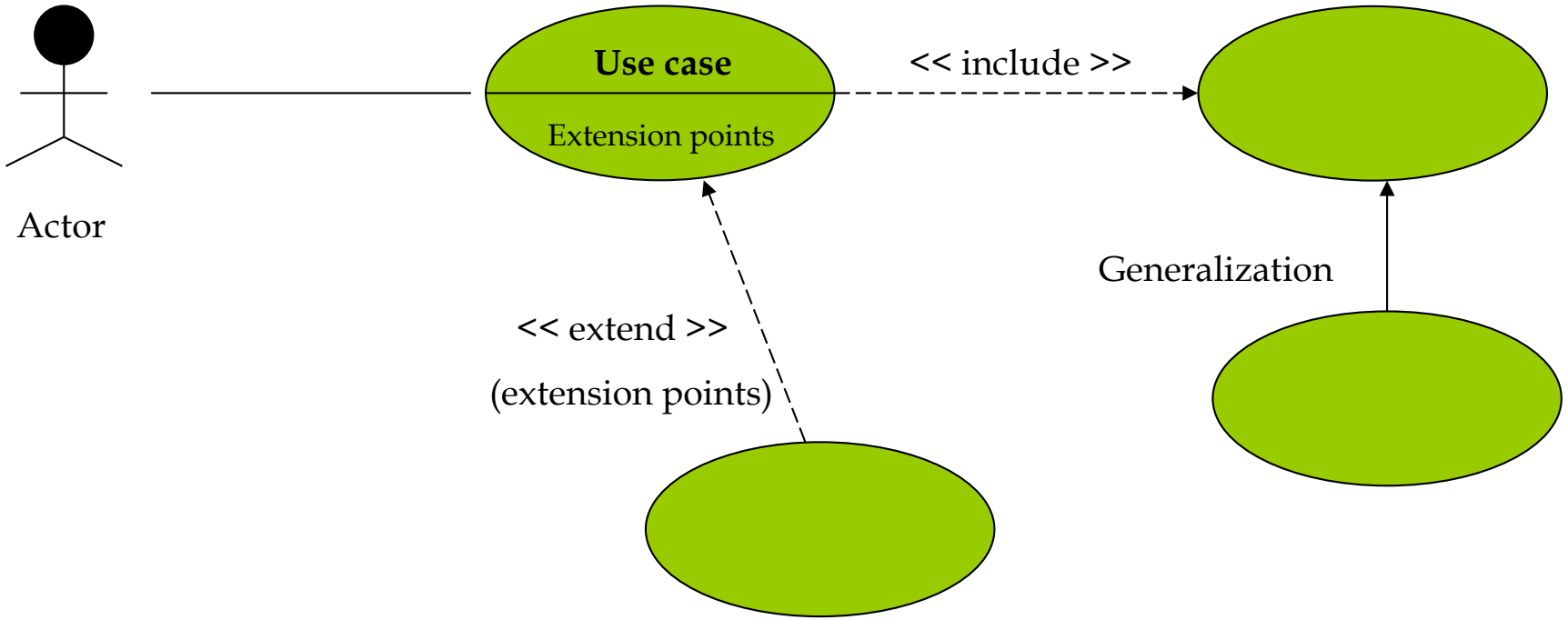
ESTENSIONE

- Uso opzionale di uno use case in concomitanza di un altro
- L'esecuzione dello use case può anche non avvenire

GENERALIZZAZIONE

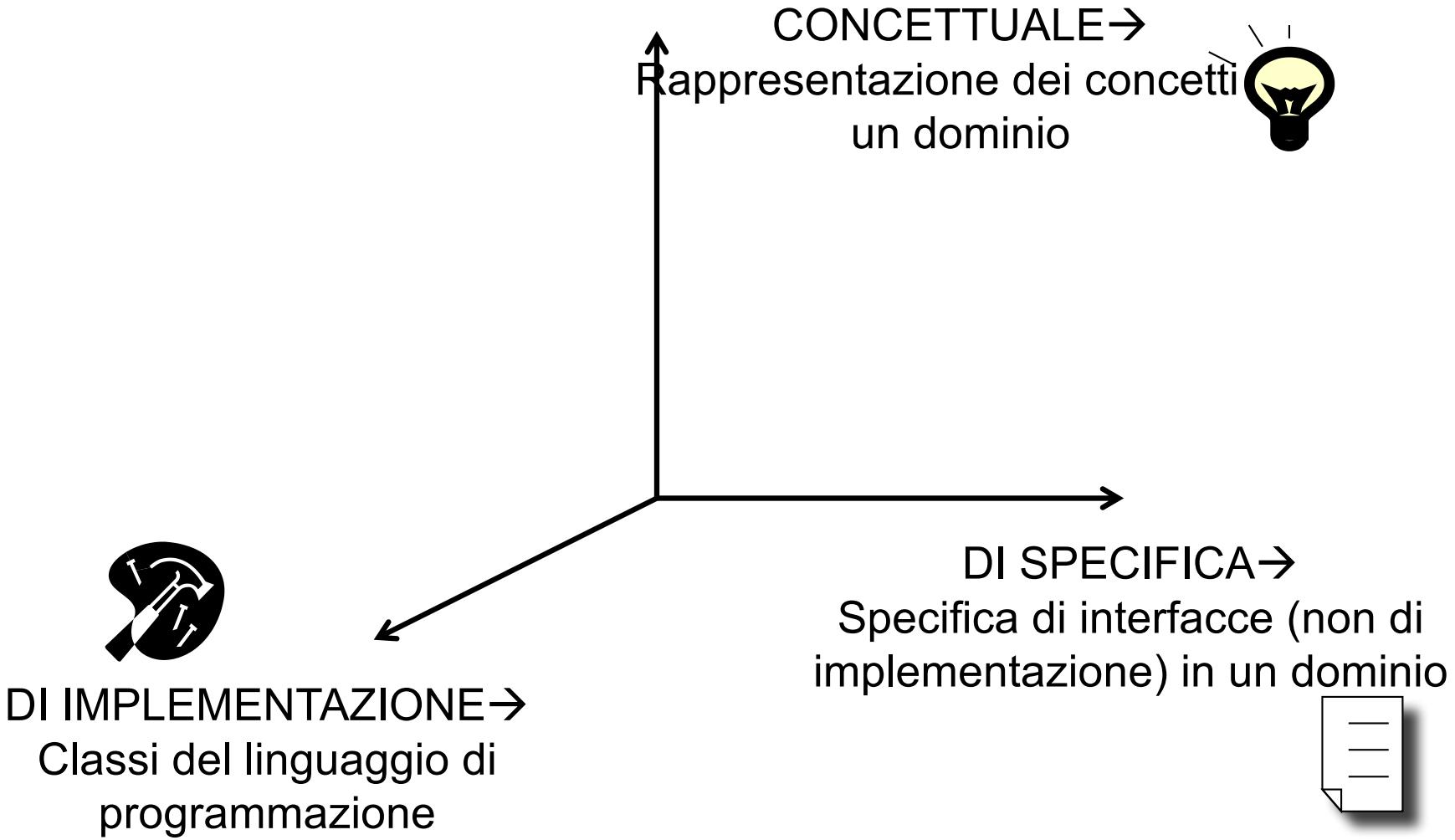
- Relazione tra elementi del modello del tipo classe/sottoclasse
- Definizione di una relazione "is type of"

USE CASE DIAGRAM: NOTAZIONE GRAFICA



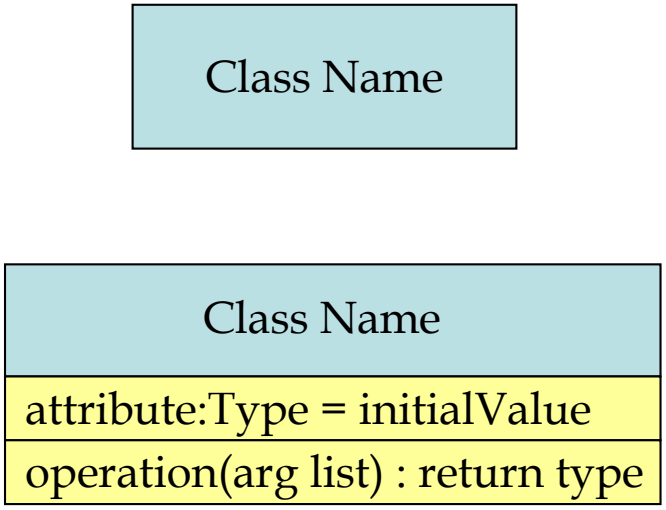
CLASS DIAGRAMS

- Le classi rappresentano OGGETTI e CATEGORIE con significato diverso



CLASS DIAGRAM: NOTAZIONE GRAFICA

- Ogni classe ha un'intestazione, degli attributi e delle operazioni
- Una classe viene rappresentata graficamente da un rettangolo contenente il nome della classe e, opzionalmente, l'elenco degli attributi e delle operazioni.



ATTRIBUTI

<visibilita‘> <nome> <molteplicita‘> : <tipo> = <val-iniziale>

- Nome → obbligatorio
- Tipo:
  - tipi fondamentali predefiniti dall’UML (corrispondenti a quelli usati comunemente nei linguaggi di programmazione)
  - tipo definito in un linguaggio di programmazione
  - classe definita (in UML) dallo sviluppatore;
- Visibilità: privata, protetta, pubblica, package; quest’ultimo livello di visibilità significa che l’attributo è visibile da tutte le classi appartenenti allo stesso package
  - + pubblica
  - # protetta
  - ~ package
  - - privata
- Molteplicità: ´ındica se l’attributo può essere replicato, cioè avere più valori (array). Si ındica con un numero o un intervallo numerico fra parentesi quadre
  - [3] tre valori
  - [1..4] da uno a quattro valori
  - [1..*] uno o pi´u valori
  - [0..1] zero o un valore (attributo opzionale)
- Valore iniziale: valore assegnato all’attributo quando si crea l’oggetto.

OPERAZIONI

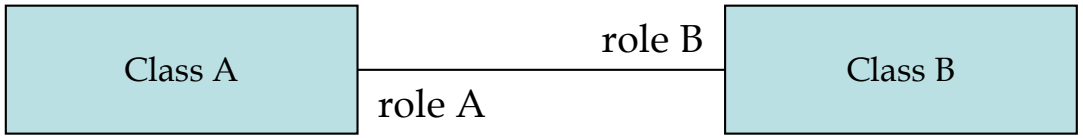
<visibilita‘> <nome> (<lista-parametri>) : <tipo>

Per ogni parametro:
<direzione> <nome> : <tipo> = <val-default>

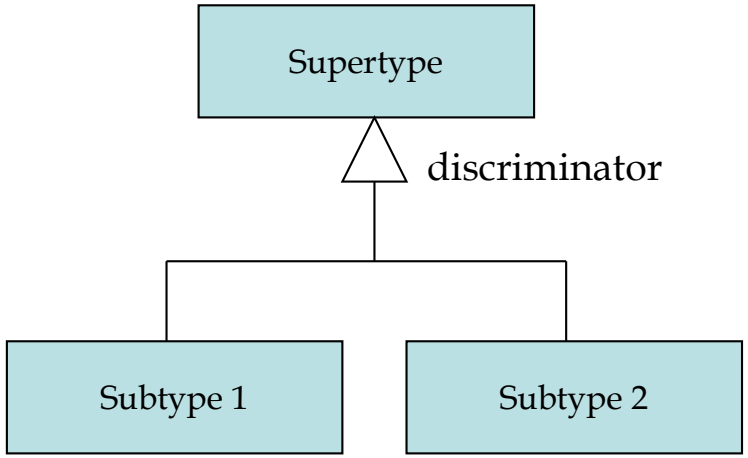
- PARAMETRI
  - Direzione: ingresso (in), uscita (out), ingresso e uscita (inout);
  - Tipo;
  - Valore default: valore passato al metodo che implementa la funzione.
  - Solo il nome del parametro è obbligatorio
  - Parametri separati da virgole
- OPERAZIONI
  - Visibilità (come attributi)
  - Tipo del valore restituito

CLASS DIAGRAM: NOTAZIONE GRAFICA

Association

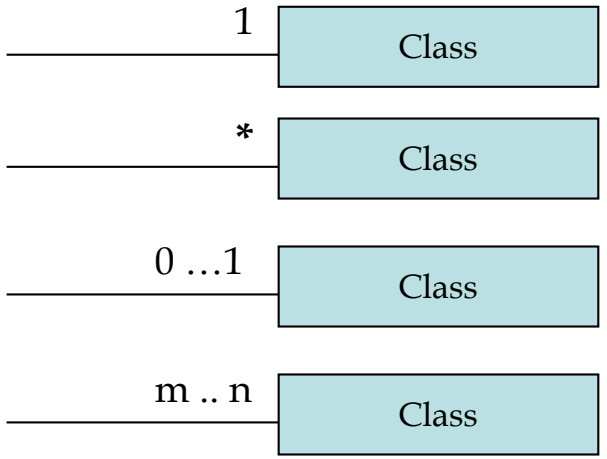


Generalization

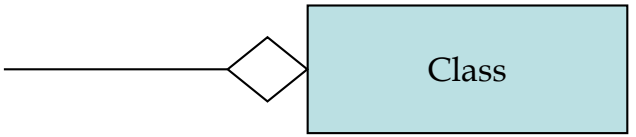


CLASS DIAGRAM: NOTAZIONE GRAFICA

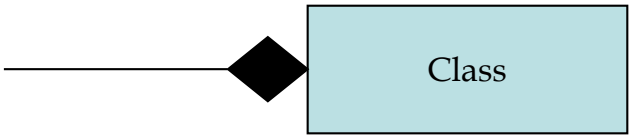
Multiplicities



Aggregation



Composition



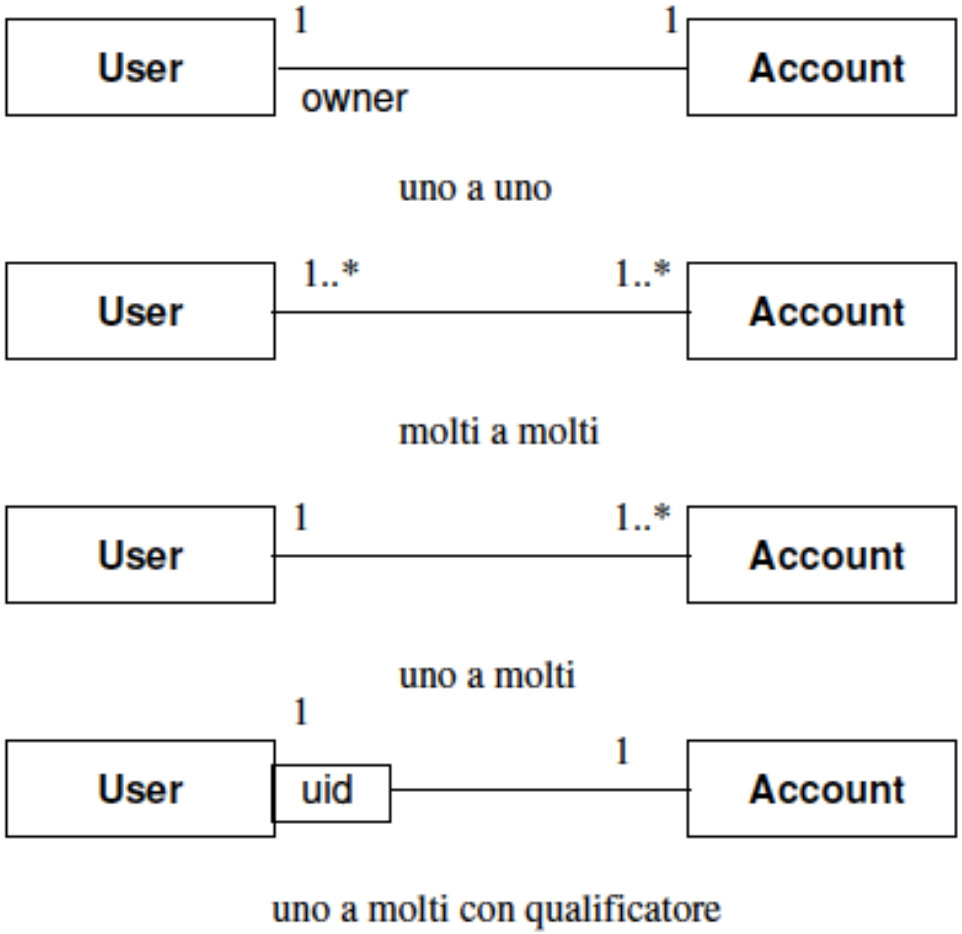
ASSOCIAZIONI

- ASSOCIAZIONE = relazione tra due classi

- Si rappresenta con una linea tra le due classi associate

- PROPRIETÀ DELLE ASSOCIAZIONI
  - Raggruppate nel rispettivo estremo di associazione (association end).
  - Ruolo: specifica la relazione fra le istanze della classe (molto utile nelle autoassociazioni)
  - Molteplicità: numero di istanze di tale classe che possono essere in relazione con una istanza della classe associata. essere indicata con intervalli numerici alle estremità della linea che rappresenta l'associazione: per esempio, 1, 0..1 (zero o uno), 1..* (uno o più), 0..* (zero o più), * (equivalente a 0..*).
  - Ordinamento: per molteplicità >1, se le istanze sono ordinate
  - Unicità: per molteplicità >1, se le istanze possono essere duplicate
  - Qualificatore: attributo dell'associazione , che distingue i diversi oggetti di una classe che possono essere in relazione con oggetti dell'altra, per meglio definire una certa istanza, fra molte, di una classe associata. rappresentato da un rettangolo avente un lato combaciante con un lato della classe opposta a quella di cui si vogliono qualificare le istanze
  - Tipo di aggregazione (aggregation kind)
  - Navigabilità

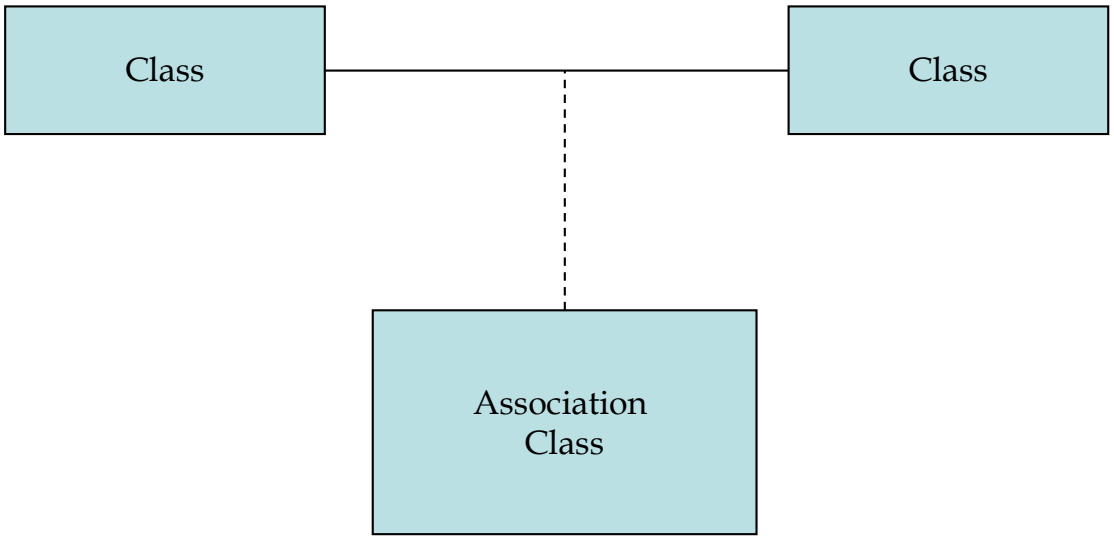
ASSOCIAZIONI



Uid è il qualificatore dell'account che rende la relazione 1..1 → uno user può avere più account ma se considero un certo account con un certo uid, avrà uno ed un solo utente associato

CLASSE ASSOCIAZIONE

- Associazione con attributi e operazioni
- Rappresentata collegando alla linea dell'associazione il simbolo della classe mediante una linea tratteggiata.



AGGREGAZIONE

- AGGREGAZIONE = associazione che lega un'entità complessa (aggregato) alle proprie parti componenti.

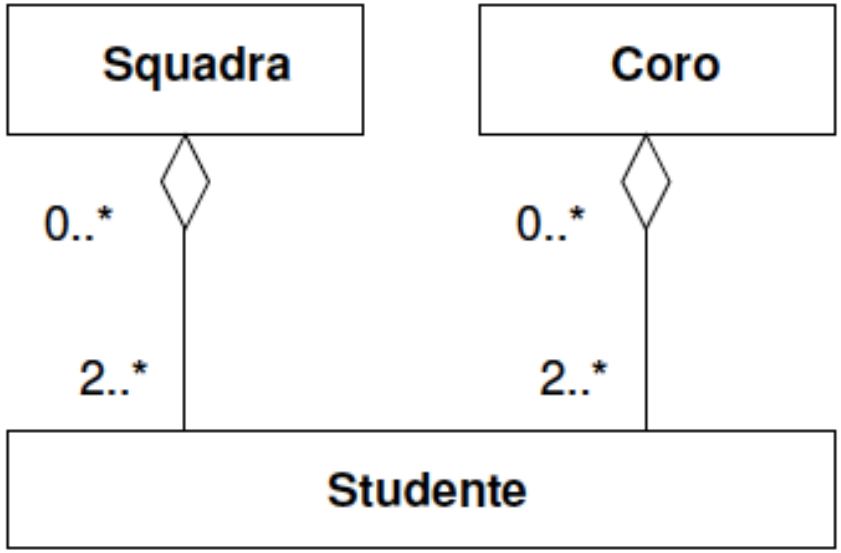
- Indicata da una piccola losanga all'estremità dell'associazione dalla parte della classe (o dell'oggetto) che rappresenta l'entità complessa.

- L'aggregazione esprime il concetto di "appartenenza", "contenimento", "ripartizione", o in generale di una forma di subordinazione strutturale non rigida.

- Un'istanza di una classe può appartenere a più di una aggregazione.

AGGREGAZIONI

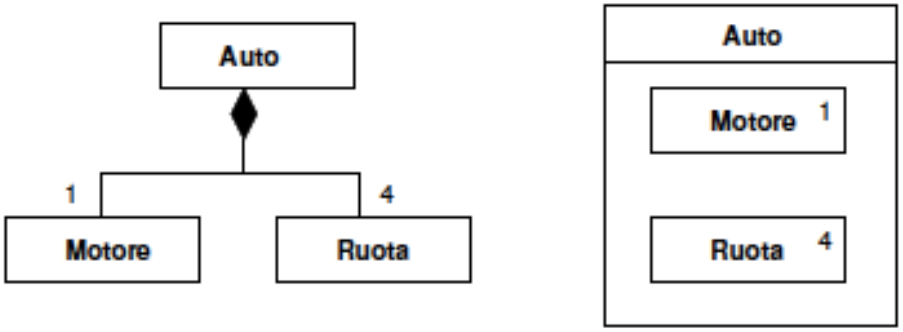
I clienti non fanno parte della loro banca, mentre i giocatori fanno parte della squadra



Uno studente può far parte sia del coro (che è formato da studenti) che della squadra (che è formata da studenti)

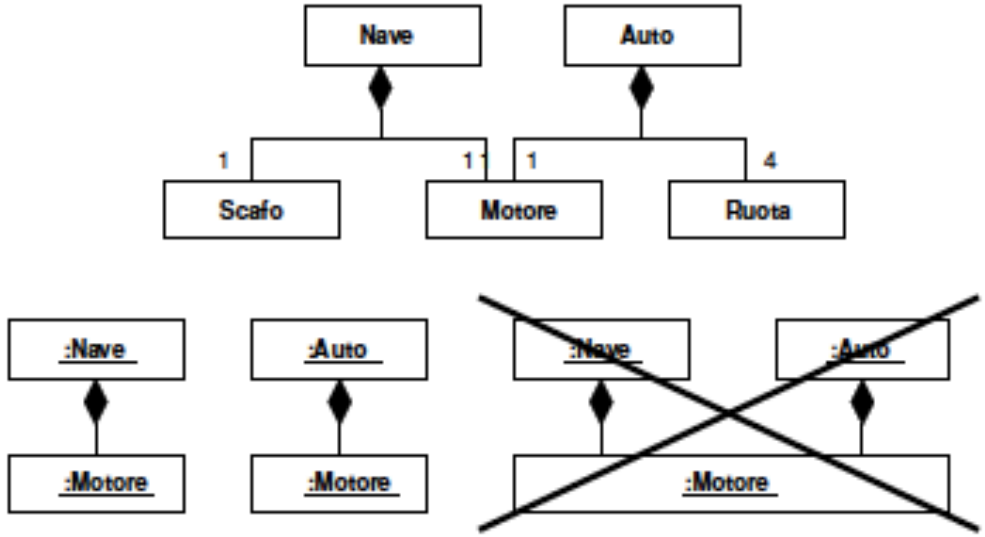
COMPOSIZIONE

- COMPOSIZIONE = subordinazione strutturale rigida in cui l'aggregato ha il completo e unico controllo delle parti componenti.
- Aggregazioni: parti indipendenti dall'aggregato
- Composizione: dipendenza stretta fra composto e componenti (l'esistenza dei componenti coincide con quella del composto → la creazione e la distruzione del composto implicano la creazione e la distruzione dei componenti)
- Un componente può appartenere ad un solo composto
- Il composto è il "padrone" del componente.
- Si rappresenta con una losanga nera dalla parte dell'entità complessa, oppure si possono disegnare i componenti all'interno dell'entità stessa.

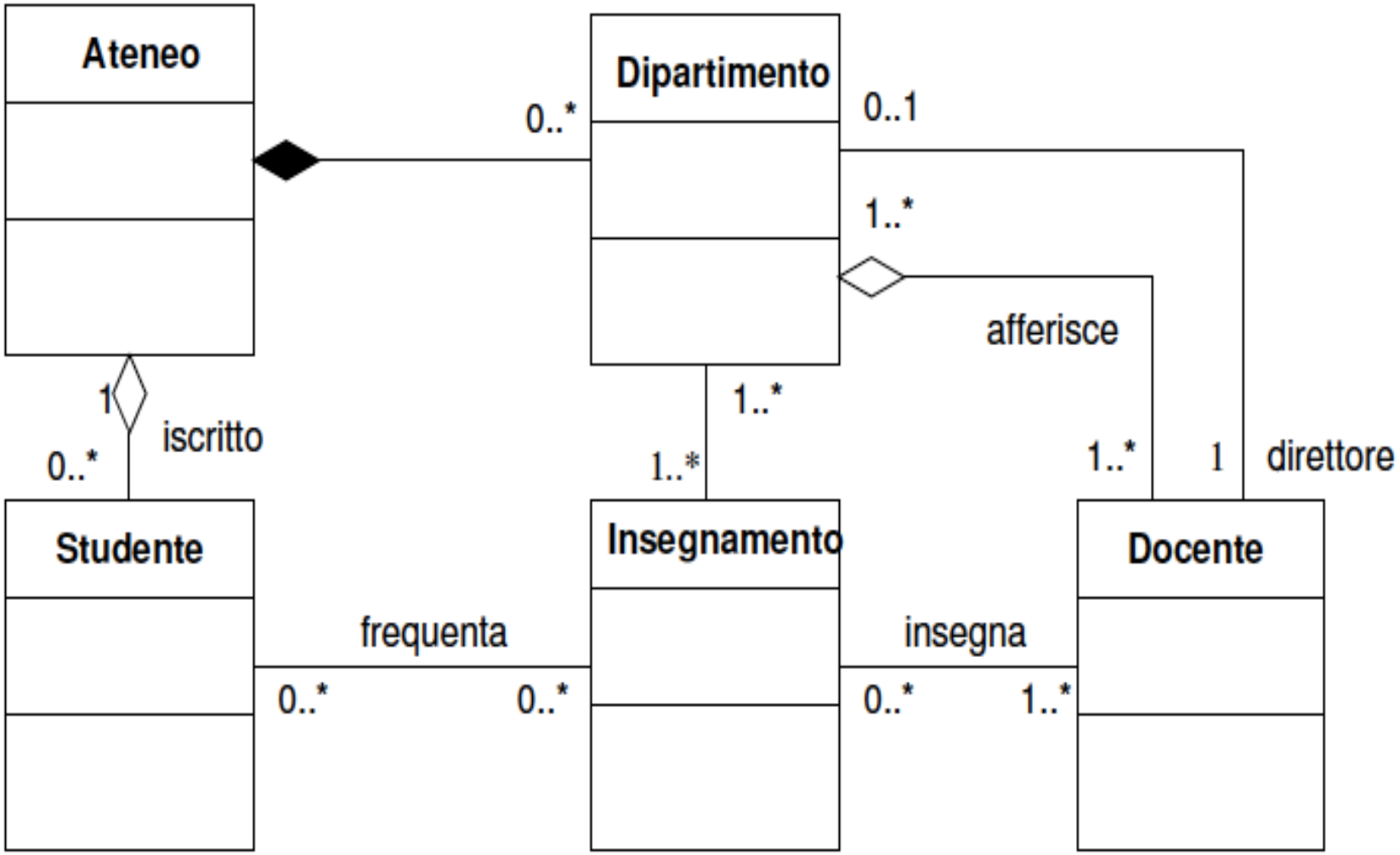


COMPOSIZIONE (2)

- Una classe può appartenere come componente a più di una composizione,
- Un'istanza di una classe componente può appartenere ad una sola istanza di una classe composta.

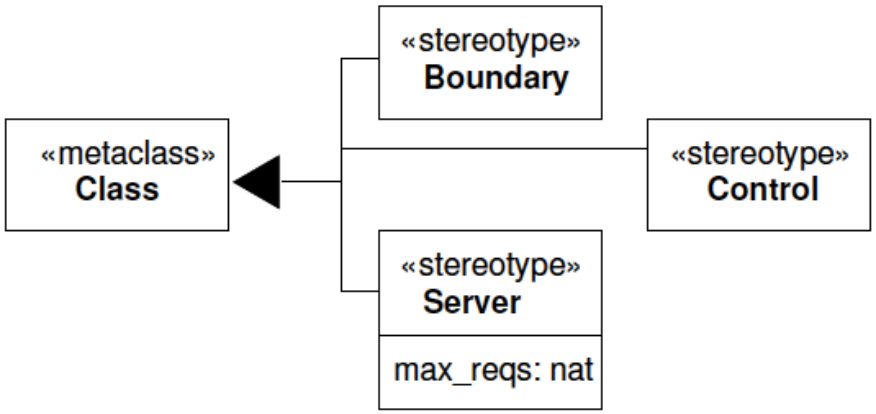


AGGREGAZIONE E COMPOSIZIONE



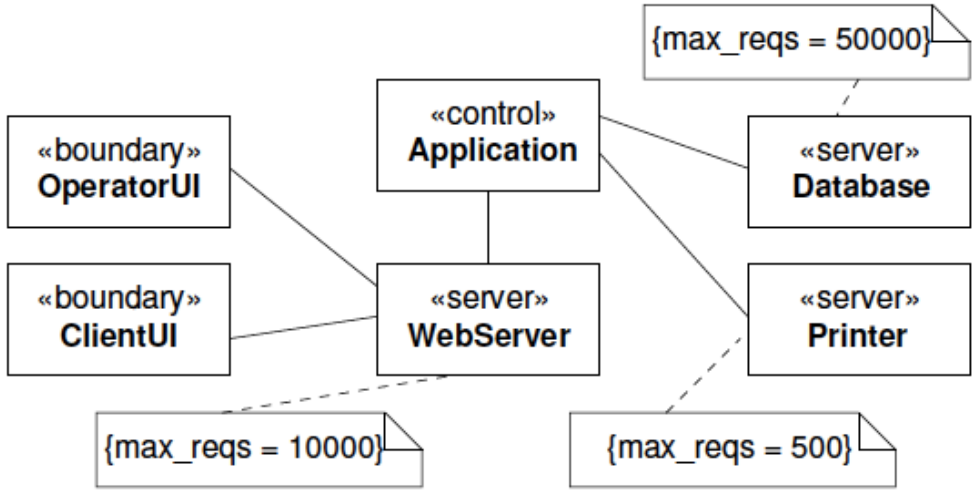
STEREOTIPI

- Stereotipi = nuovi elementi di modello ottenuti da elementi già esistenti (per esempio, dall'elemento classe o dall'elemento associazione) aggiungendo informazioni di vario tipo alla loro semantica, per mezzo di vincoli e valori etichettati



Distinguo le classi destinate all'interfacciamento con l'utente (boundary) dalle classi che rappresentano i server (server) e da quelle che definiscono la logica dell'applicazione (control)

Si possono associare informazioni rilevanti per tutti i tipi di server, per esempio il massimo numero di richieste che possono essere messe in attesa, espresso da un valore etichettato (max reqs).



GENERALIZZAZIONE

- Una classe (classe base o superclasse) generalizza un'altra classe (classe derivata o sottoclasse) quando definisce un insieme di elementi che include l'insieme di elementi definiti dalla classe derivata

- La classe derivata è un sottoinsieme della classe base → La classe base ha meno caratteristiche (attributi, operazioni, associazioni, vincoli. . .) della classe derivata.

- Una classe base può avere più classi derivate, e in questo caso ne riassume alcune caratteristiche comuni (attributi, operazioni, associazioni, vincoli. . .).

- La relazione di generalizzazione si può anche chiamare specializzazione (cambia il punto di vista). La specializzazione può avvenire per estensione (aggiungo attributi) o per restrizione (aggiungo vincoli)

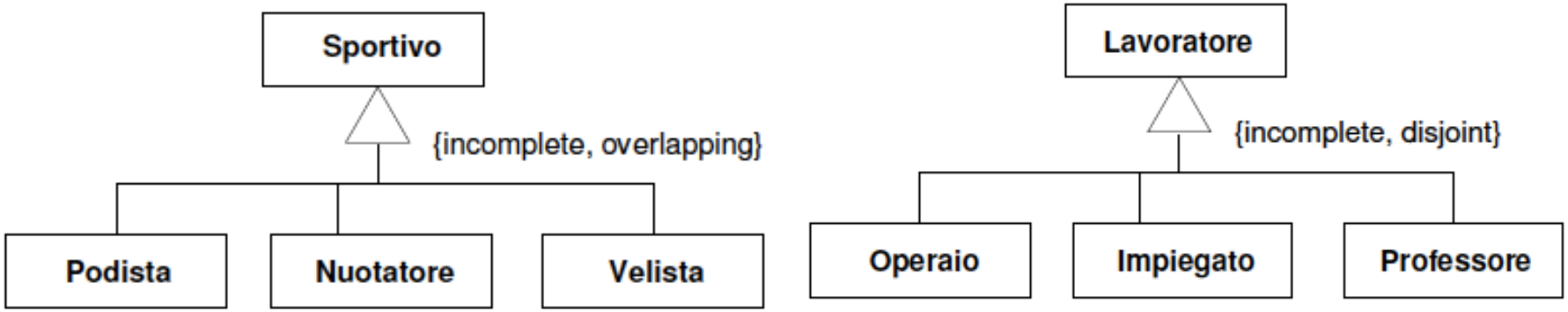
- Un oggetto appartenente ad una classe derivata appartiene anche alla classe base.
- Principio di sostituzione (di Barbara Liskov): un'istanza della classe derivata può sostituire un'istanza della classe base → importante per verificare se la relazione di generalizzazione è corretta (soprattutto nel caso di restrizione)

EREDITARIETÀ

- Una sottoclasse ha tutte le caratteristiche della superclasse → eredita le caratteristiche
- Operazioni → nell'eredità si può effettuare overriding (stessa signature ma diversa implementazione) purchè valga il principio di sostituzione.
- Eredità multipla→
  - una classe derivata è un sottoinsieme di due o più classi che non sono in relazione di generalizzazione/specializzazione fra di loro.
  - Le istanze della classe derivata ereditano le caratteristiche di tutte le classi base

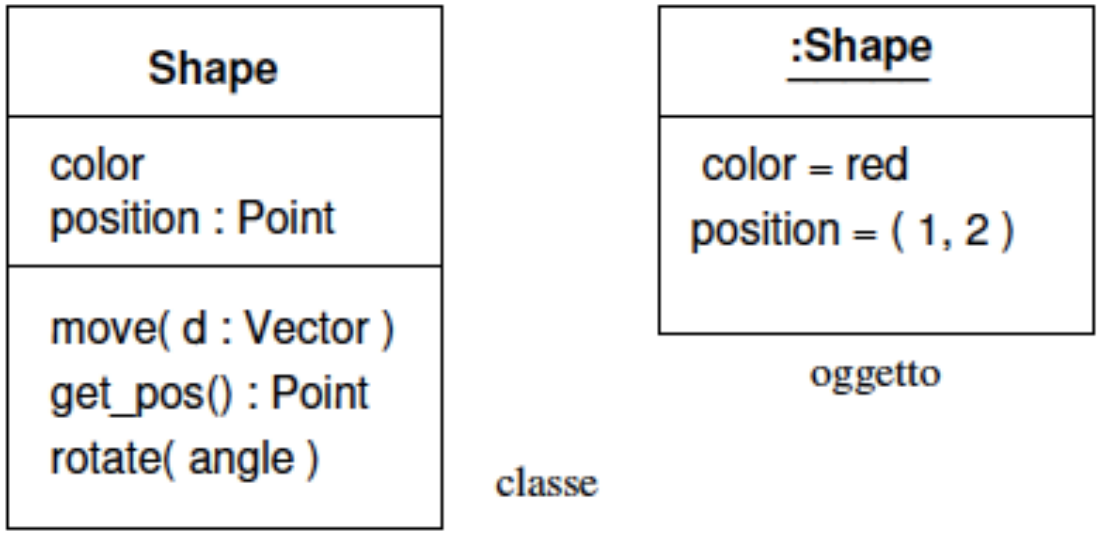
INSIEME DI GENERALIZZAZIONI

- Generalizzazione → spesso usata per classificare le entità del dominio analizzato
- Insieme di generalizzazioni:
  - modo di raggruppare le sottoclassi di una classe base (insieme di sottoinsiemi), a cui si può dare un nome che descriva il criterio con cui si raggruppano le sottoclassi.
  - Insieme completo: ogni istanza della classe base appartiene ad almeno una delle sottoclassi
  - Insieme disgiunto: le sottoclassi sono disgiunte (intersezione vuota)
  - Per default, un insieme di generalizzazioni è incompleto e disgiunto.



OGGETTI

- Oggetto → rettangolo contenente
  - i nomi dell'oggetto e della classe d'appartenenza, sottolineati e separati dal carattere ':',
  - (opzionalmente) gli attributi con i rispettivi valori.
  - Il nome della classe o quello dell'oggetto possono mancare. In questo caso, il nome della classe viene preceduto dal carattere ':'



ACTIVITY DIAGRAMS

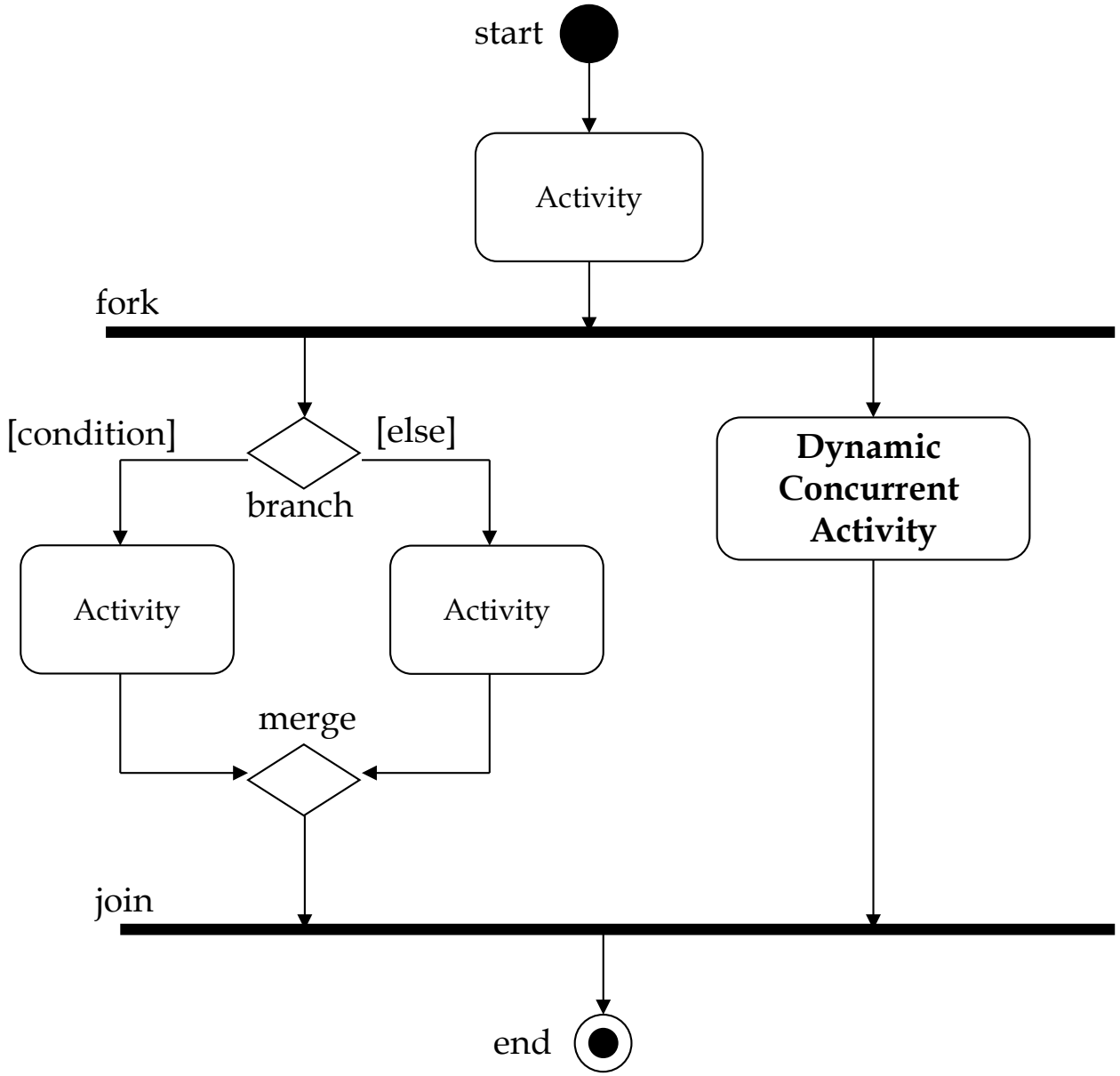
ATTIVITÀ = qualcosa che viene fatto e che si può potenzialmente suddividere in sotto attività

- Activity diagram:
  - Rappresentano la sequenza di attività e supportano il comportamento condizionale e parallelo
  - Servono a descrivere il flusso di attività, di controllo e di informazioni
  - Simili ai diagrammi di flusso (flowchart)

- Comportamento condizionale:
  - Branch → un input con diversi output sotto condizione
  - Merge → più input danno luogo ad un singolo output
  - Comportamento parallelo:
  - Fork → da una transizione ne escono molte parallele
  - Join → da più percorsi paralleli, si torna ad un solo percorso comune, quando tutte le attività parallele sono completate

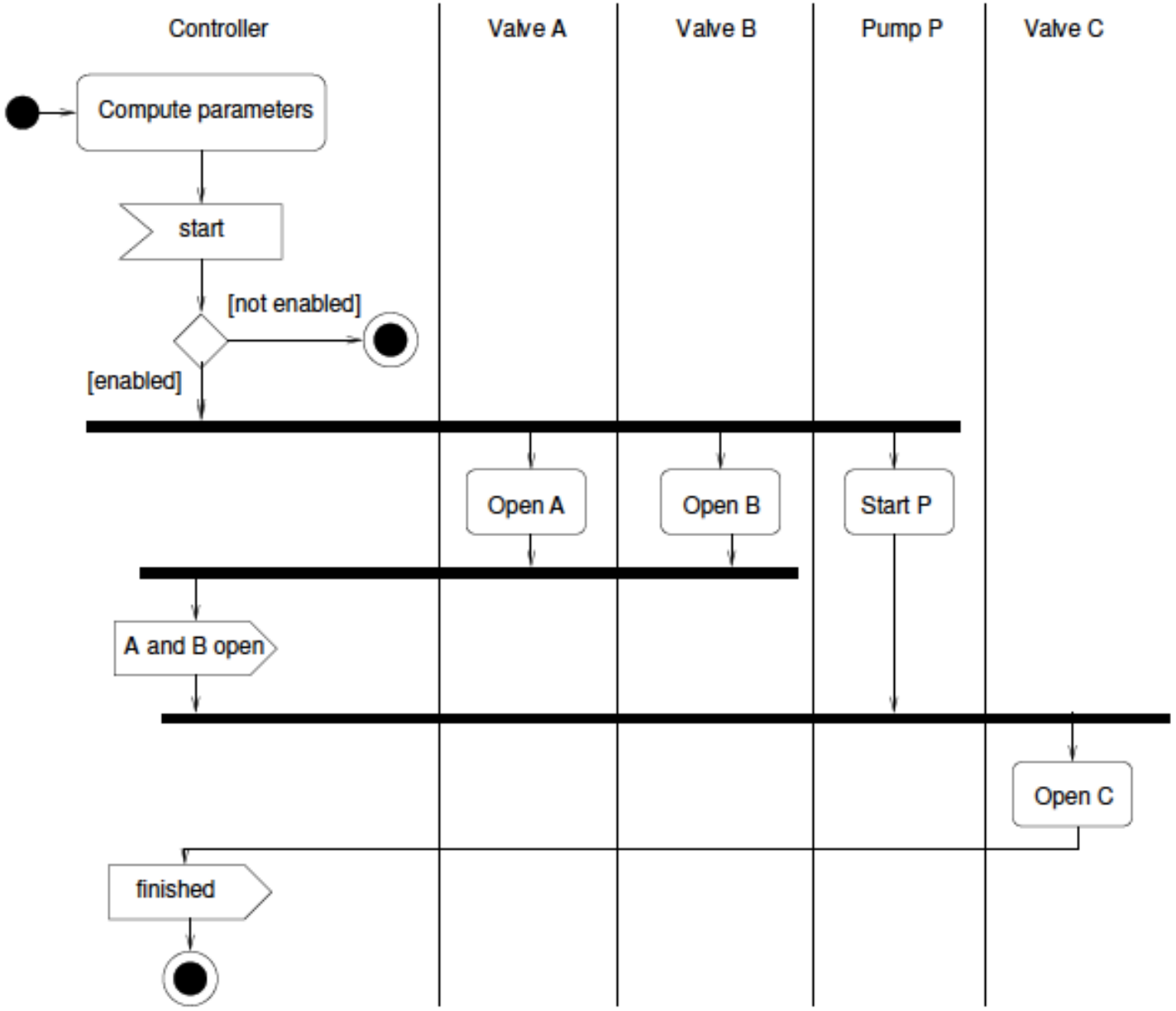
- Swimlane → attività svolte da entità differenti, raggruppante graficamente in partizioni (corsie)

ACTIVITY DIAGRAM: NOTAZIONE GRAFICA

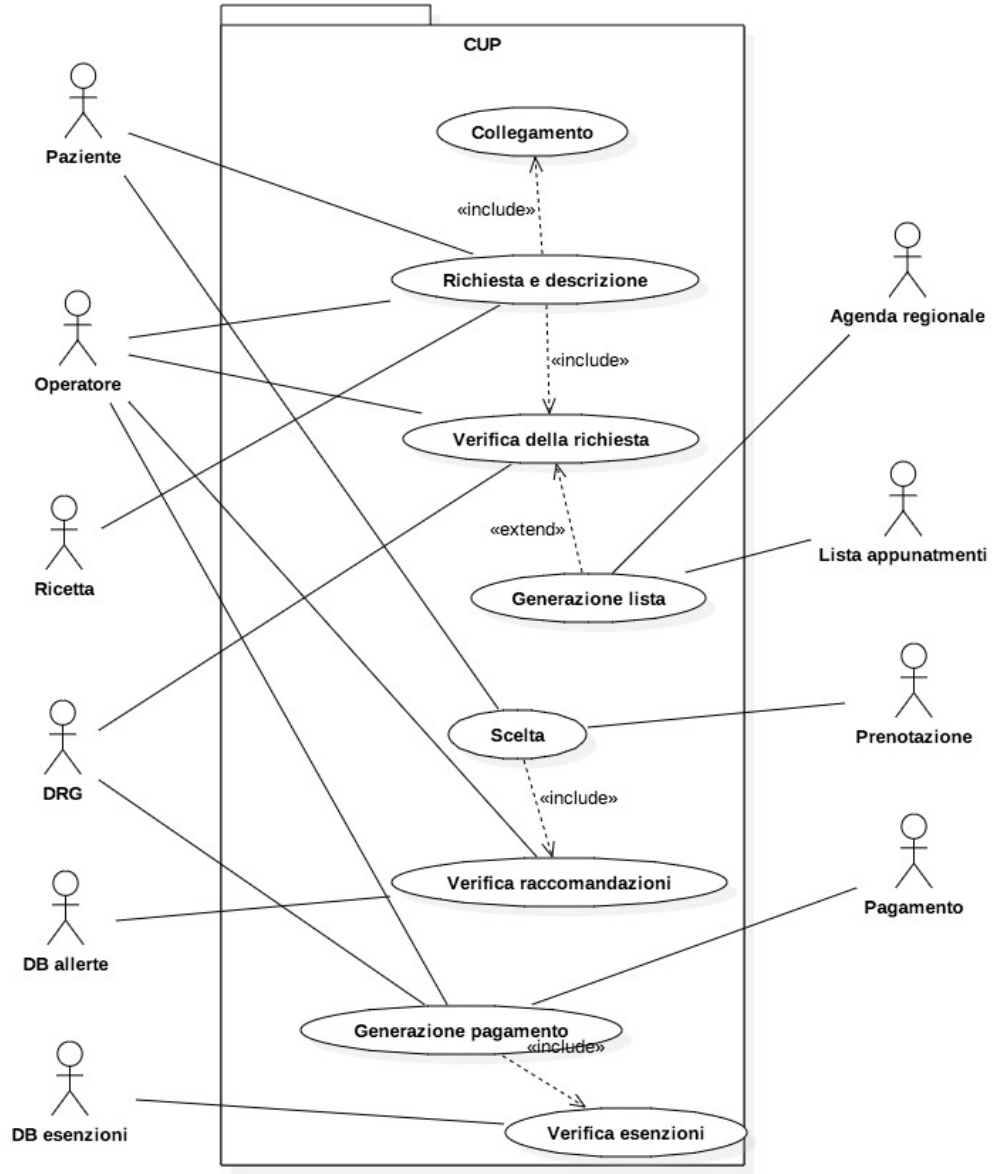


SWIMLANE

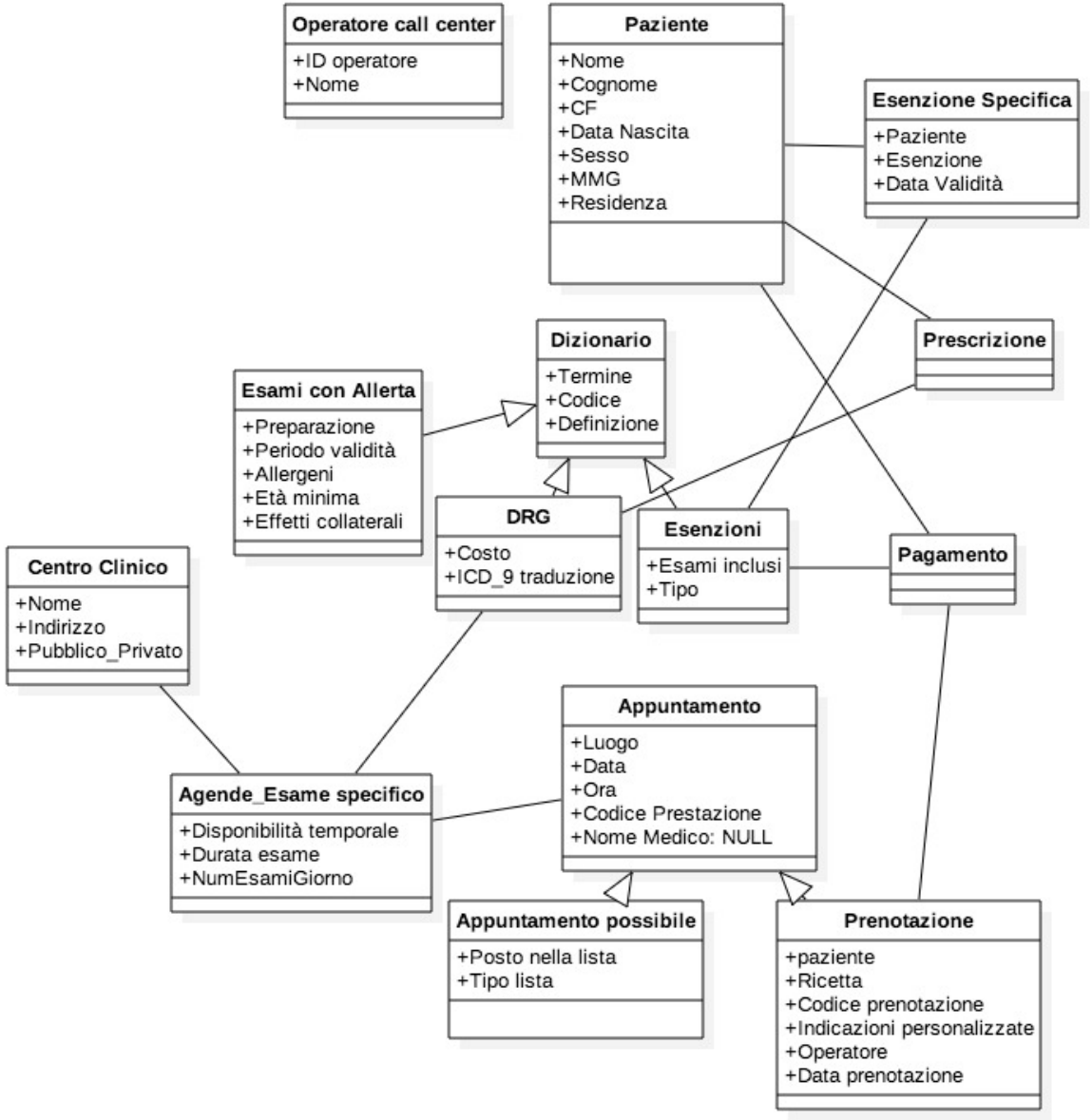
UNIVERSITÀ
DEGLI STUDI DI TRIESTE



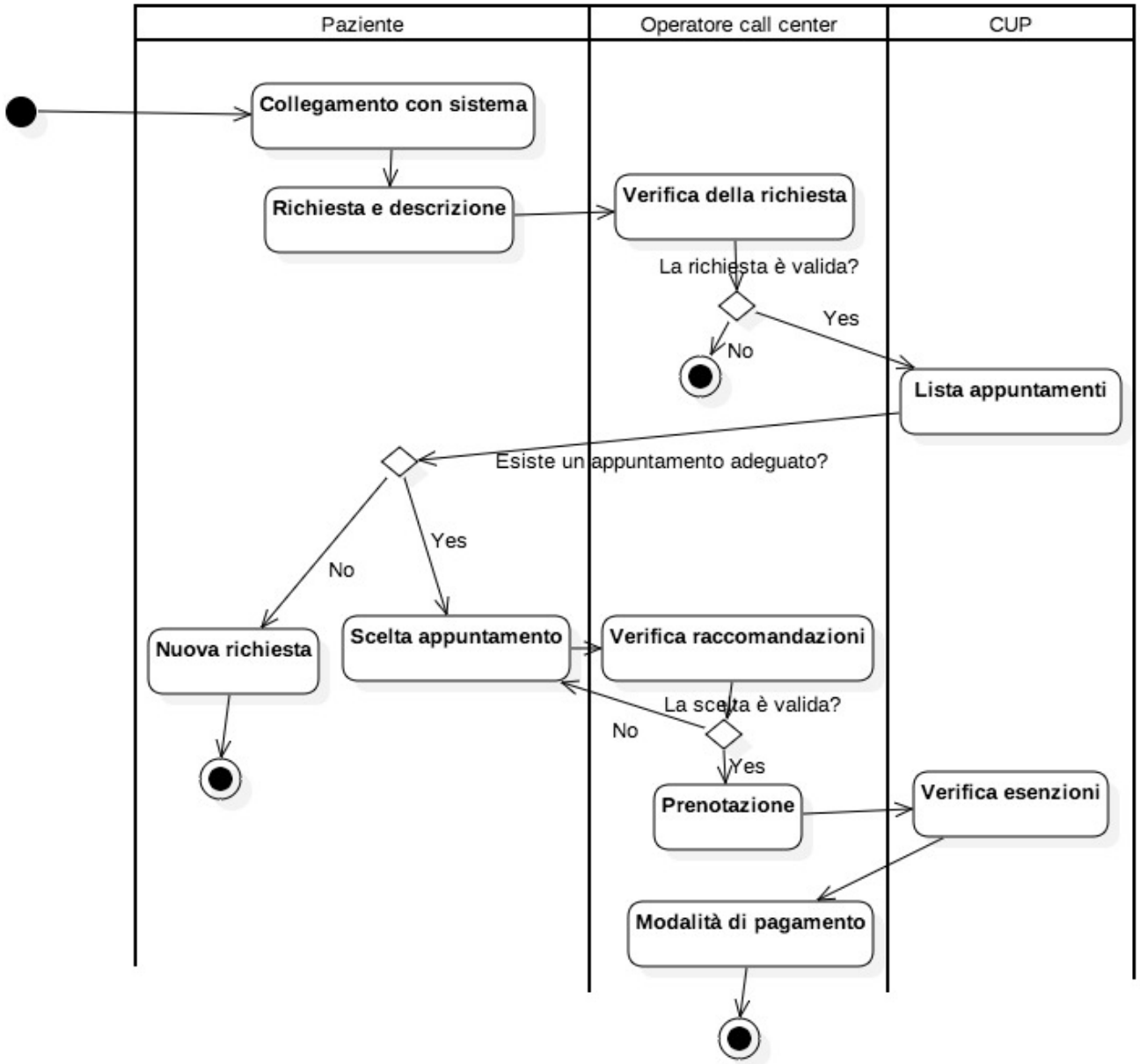
USE CASE DIAGRAM ESEMPIO



CLASS DIAGRAM ESEMPIO



ACTIVITY DIAGRAM ESEMPIO

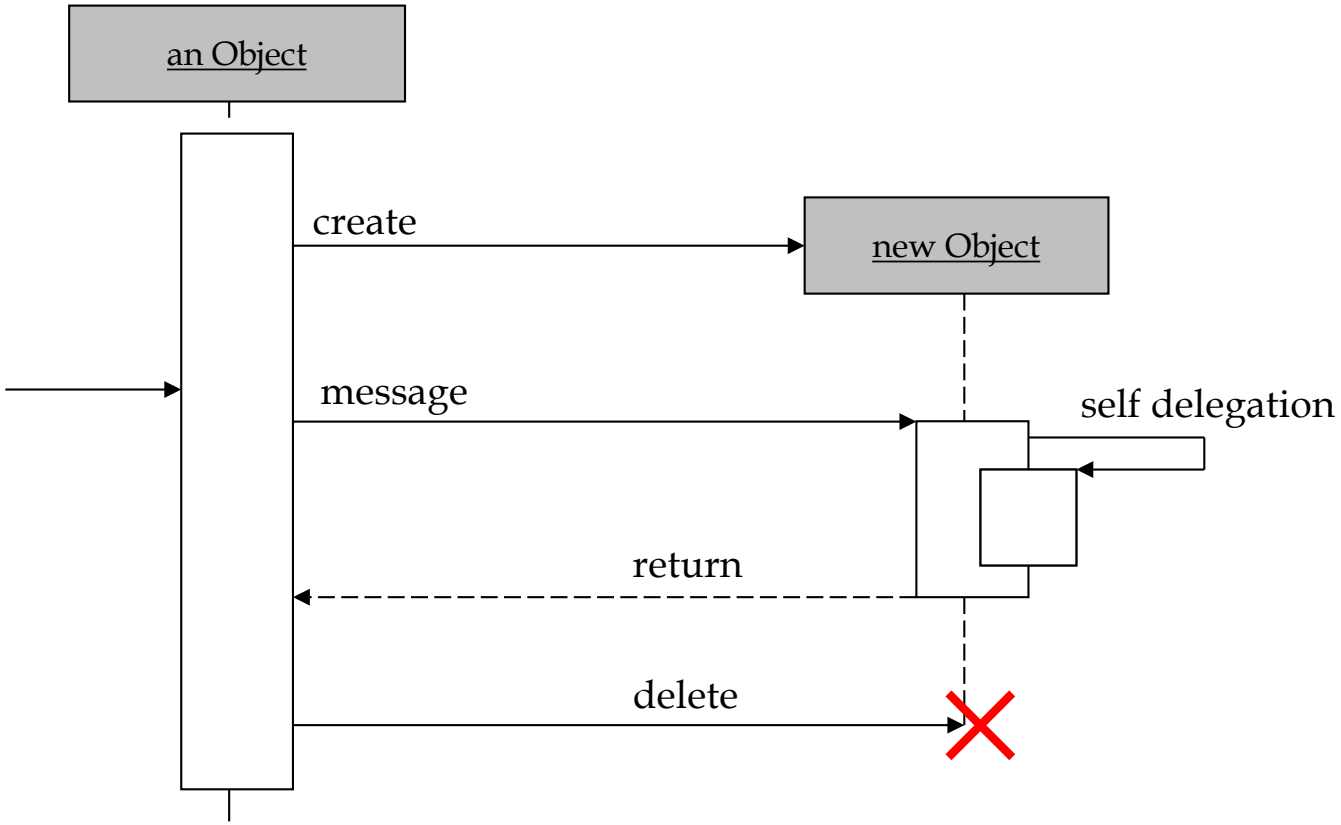


SEQUENCE DIAGRAMS

- Diagrammi di interazione
- Rappresentano i messaggi che gli oggetti si scambiano in determinate situazioni

- Oggetti → rettangoli posti all'inizio di linee verticali tratteggiate
- Linea della vita dell'oggetto →
  - Linea verticale tratteggiata
  - Rettangolo verticale sulla linea della vita: momento in cui l'oggetto è attivo e scambia messsaggi
- Messaggi →
  - Frecce tra le linee della vita di due oggetti
  - L'ordine va dall'alto verso il basso
  - Selfcall: messaggio che l'oggetto scambia con se stesso

SEQUENCE DIAGRAM: NOTAZIONE GRAFICA



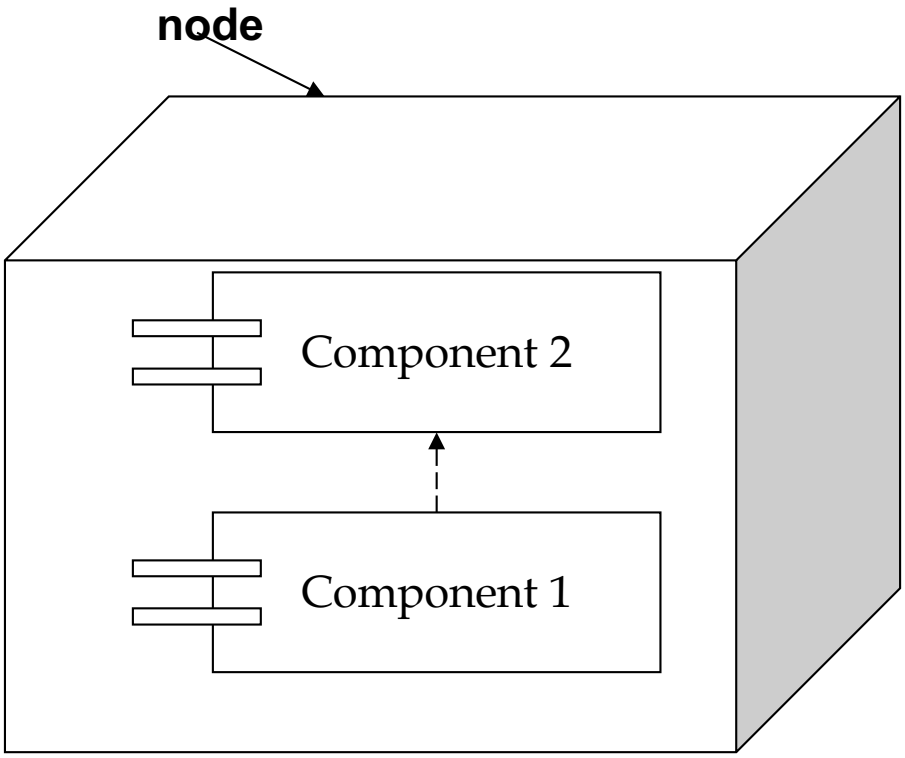
DEPLOYMENT E COMPONENT DIAGRAMS

- DEPLOYMENT DIAGRAM:
  - Rappresentano la parte HW del sistema
  - Ogni nodo rappresenta una unità computazionale (un dispositivo, un sensore, una periferica, etc)
  - Le connessioni tra i nodi sono percorsi di comunicazione tra i componenti

- COMPONENT DIAGRAM:
  - Rappresentano le componenti del sistema e le loro dipendenze
  - Componente: package software
  - Dipendenza: relazione tra due componenti (come un componente influisce sul cambiamento dell'altro)

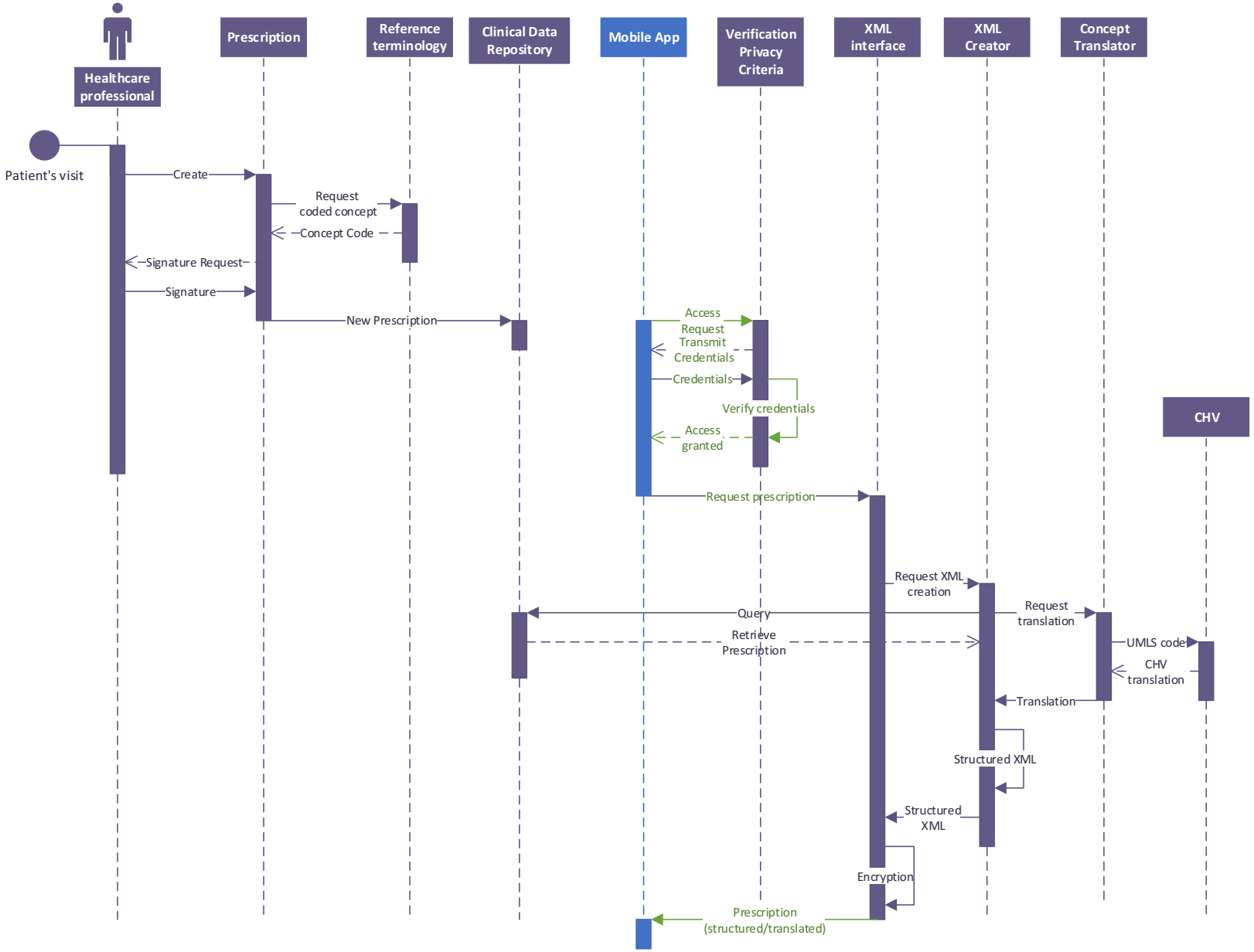
- COMBINAZIONE dei due diagrammi →
  - rappresenta il sistema mettendo in evidenza quali componenti sono installati e funzionano su ciascun nodo
  - Relazione tra le componenti HW e SW

DEPLOYMENT E COMPONENT DIAGRAM: NOTAZIONE GRAFICA

- 

SEQUENCE DIAGRAM: ESEMPIO

UNIVERSITÀ
DEGLI STUDI DI TRIESTE



DEPLOYMENT E COMPONENT DIAGRAM: ESEMPIO