System Design
(progettazione di un sistema software)

System vs. OO design

System design

Progettazione di alto livello

- Identificazione dei principali componenti e delle relazioni tra questi (**architettura software**)

OO design

Progettazione di dettaglio

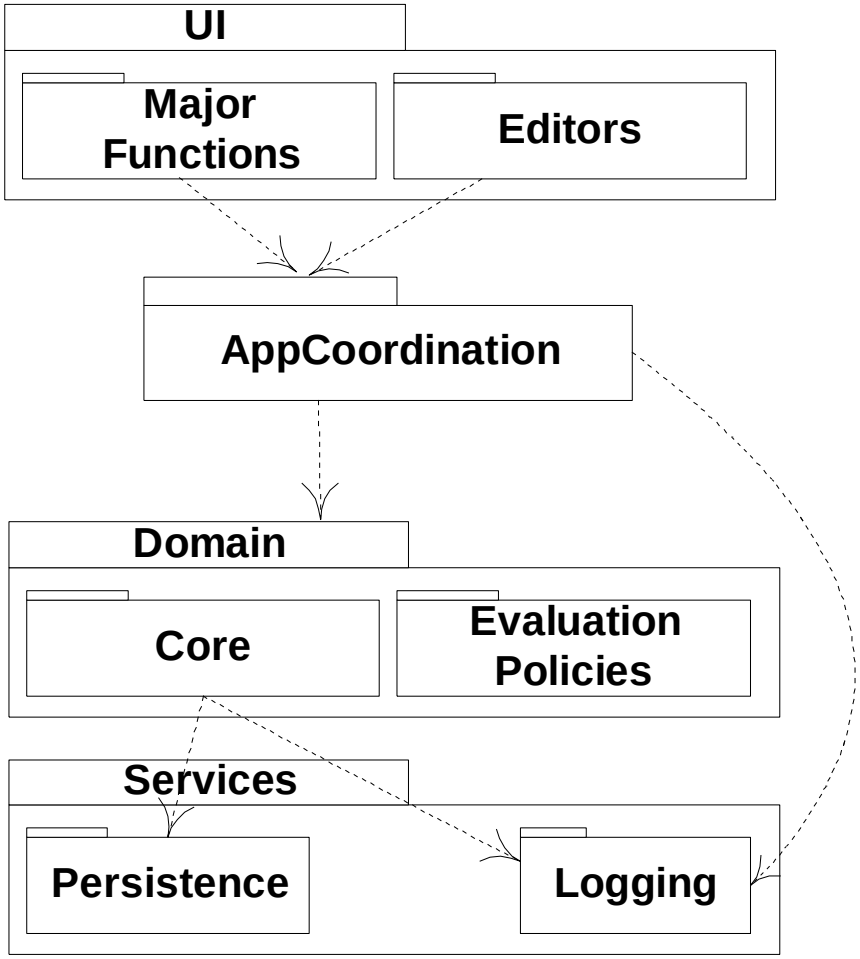
- Specifica dei meccanismi interni ai componenti maggiori
- Dipende dal paradigma di programmazione (OO)

Architettura software

- Un’architettura software definisce un modello compatto del modo in cui un sistema è strutturato e di come i suoi principali componenti (sottosistemi) comunicano fra loro
  - Un sottosistema include classi, operazioni e altri aspetti del sistema strettamente collegati tra loro
- L’architettura evidenzia decisioni che avranno un forte impatto sul lavoro e i cambiamenti successivi

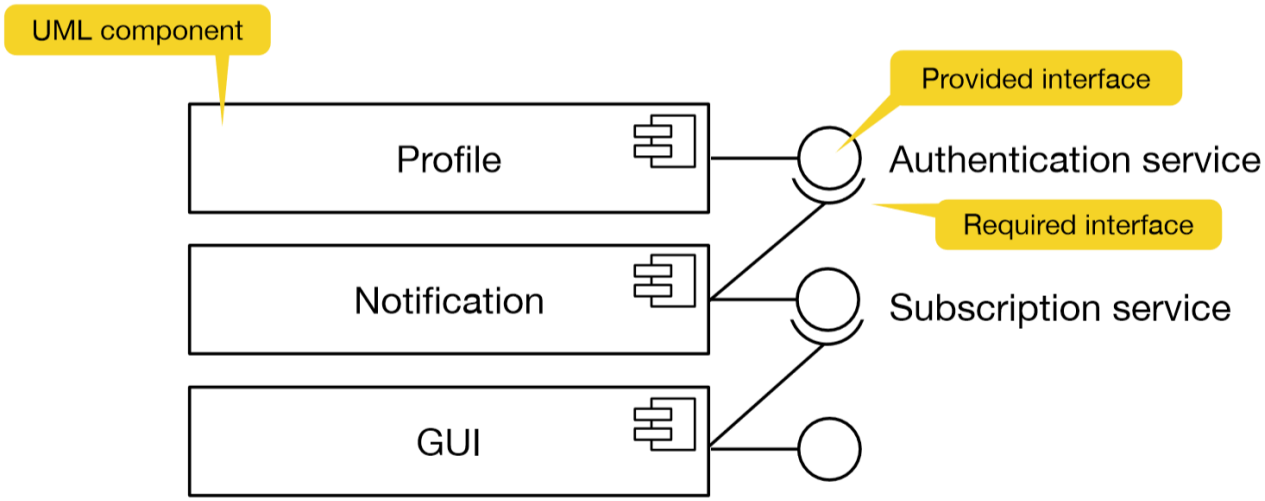
Rappresentazione dell'architettura in UML: **diagramma dei package**

- Un **package** rappresenta può essere
  - un'unità binaria di rilascio (es. *.jar*)
  - un namespace (es. *Java package*)
- Una **dipendenza** rappresenta una *relazione cliente-fornitore* tra package
  - Elementi (es. classi) di un package richiedono elementi di un altro package
- Un cambiamento nel fornitore può causare un cambiamento nel cliente



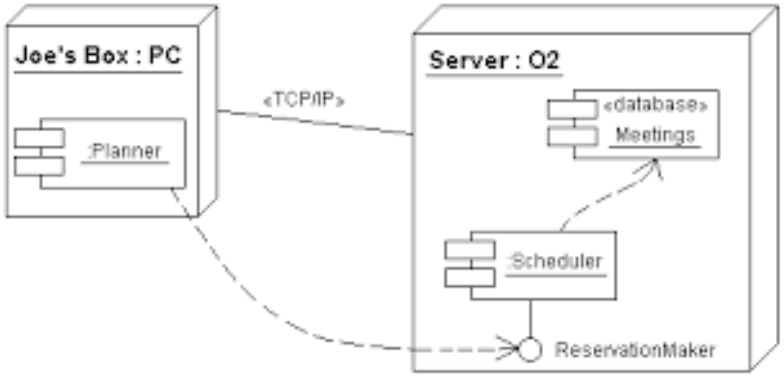
Rappresentazione dell'architettura in UML: **diagramma dei componenti**

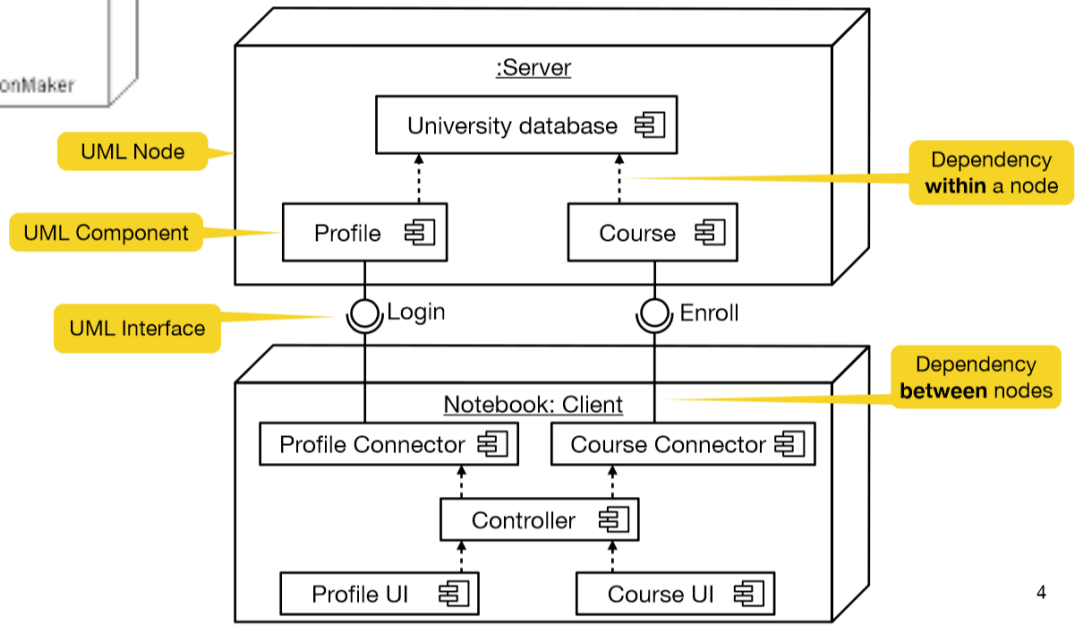
- Modella il sistema in termini di componenti e dipendenze tra componenti
- Un componente può esporre un'**interfaccia** (insieme di operazioni rese pubbliche)
  - anche detta **servizio** o **API** (Application Programming Interface)
- Una dipendenza è un connettore tra il componente cliente e il componente fornitore



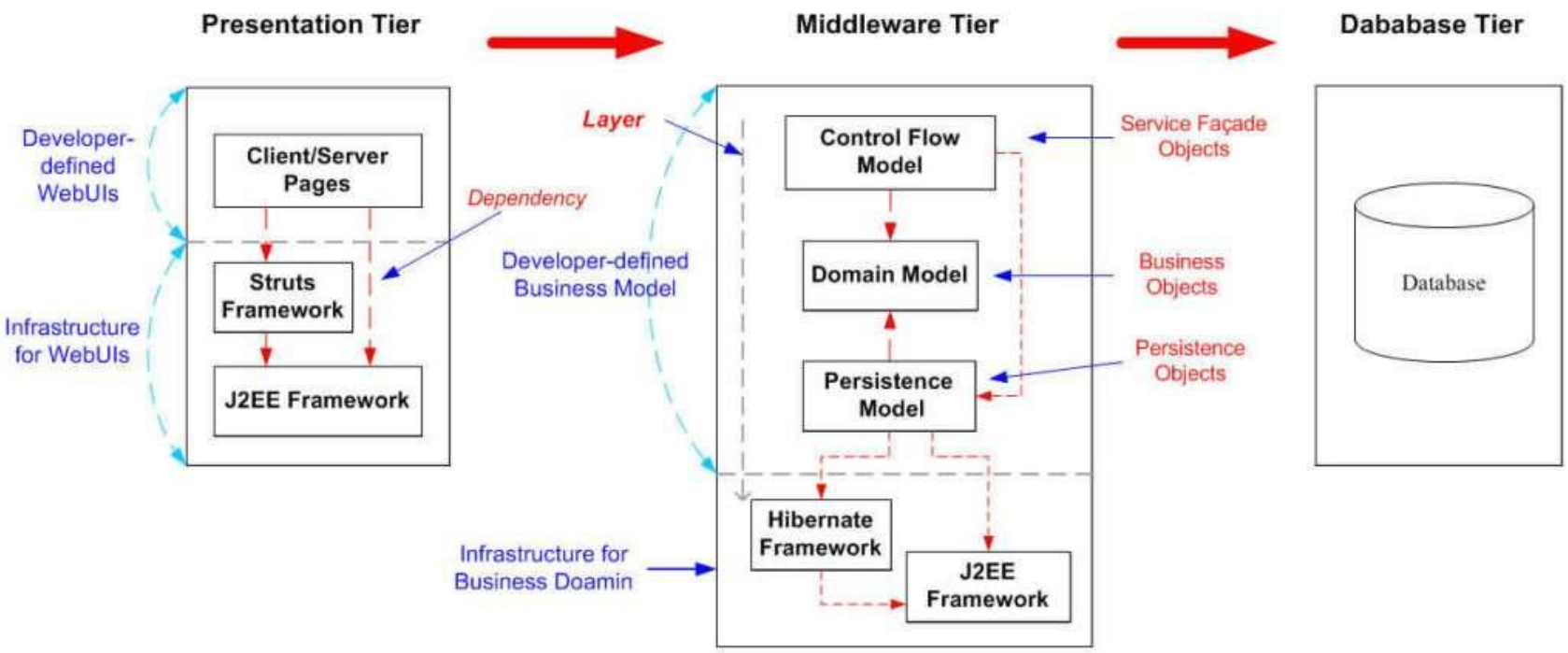
Rappresentazione dell'architettura in UML: **diagramma di deployment**

- Mostra la distribuzione dei sottosistemi tra le componenti hardware (nodi)

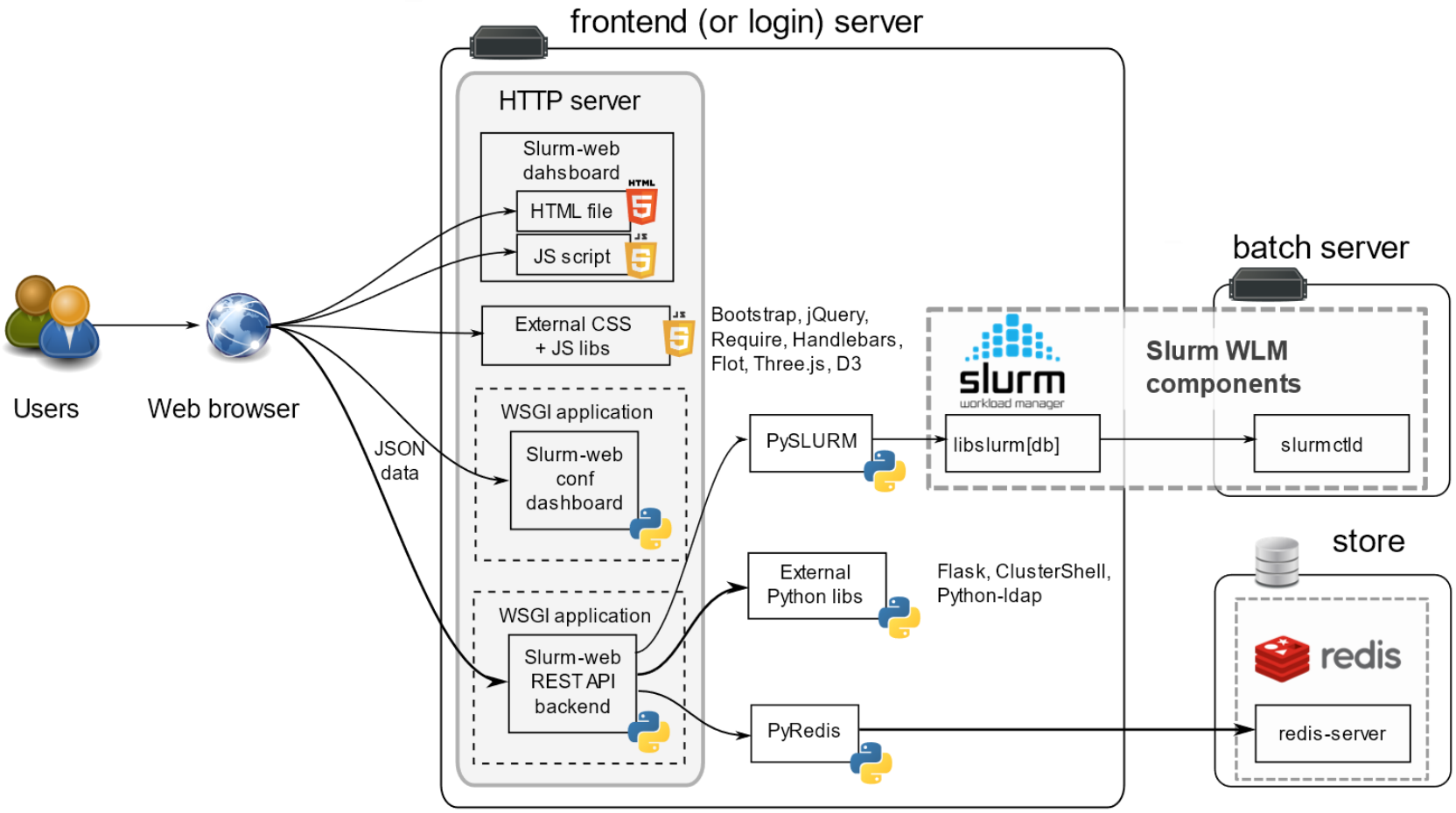




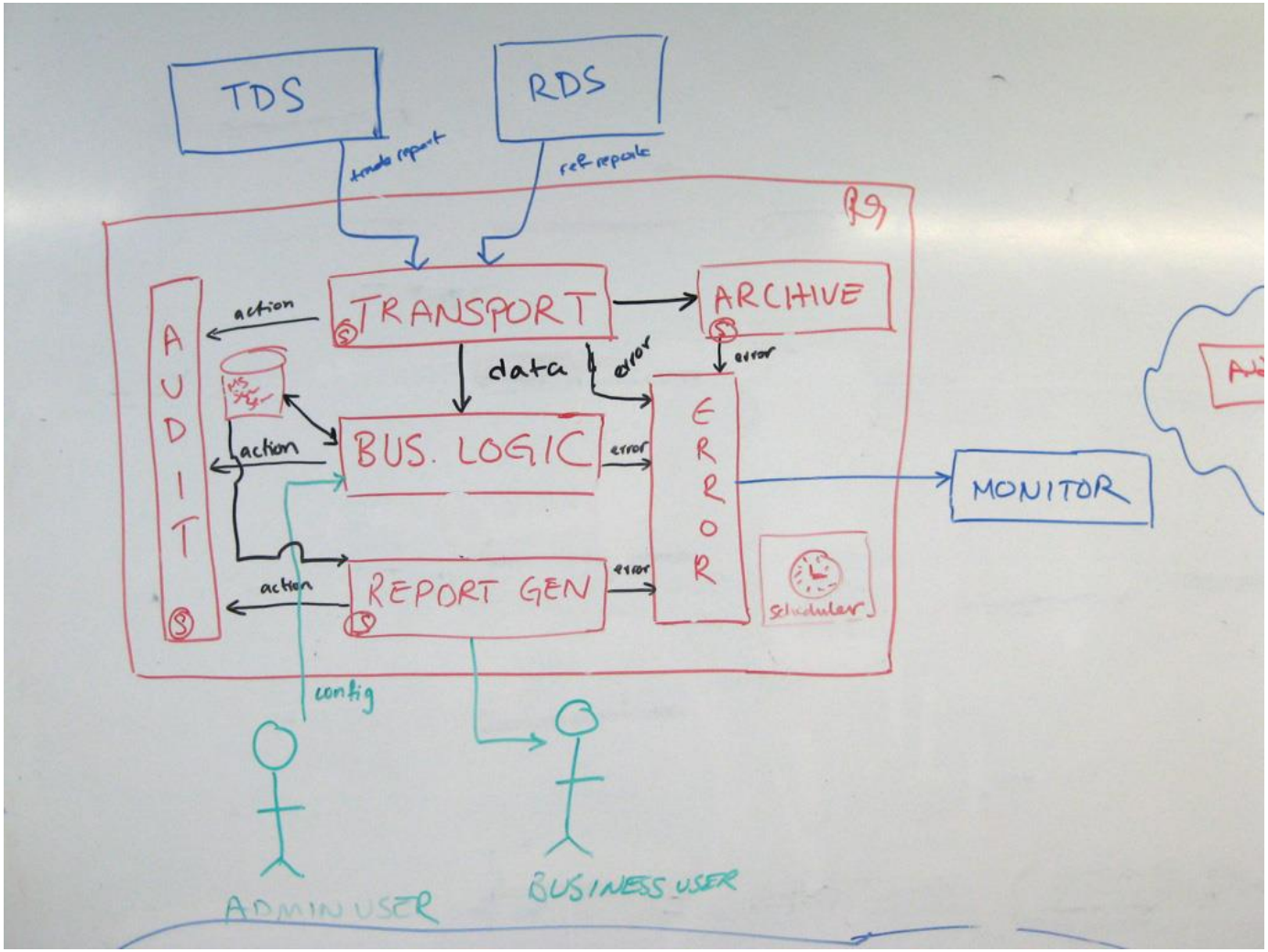
Rappresentazione non-UML dell'architettura



Rappresentazione non-UML dell'architettura



Rappresentazione non-UML dell'architettura



Checklist per la revisione di diagrammi di architettura software

General

Does the diagram have a title?	Yes	No
Do you understand what the diagram type is?	Yes	No
Do you understand what the diagram scope is?	Yes	No
Does the diagram have a key/legend?	Yes	No

Checklist per la revisione di diagrammi di architettura software

Elements

Does every element have a name?	Yes	No
Do you understand the type of every element? (i.e. the level of abstraction; e.g. software system, container, etc)	Yes	No
Do you understand what every element does?	Yes	No
Where applicable, do you understand the technology choices associated with every element?	Yes	No
Do you understand the meaning of all acronyms and abbreviations used?	Yes	No
Do you understand the meaning of all colours used?	Yes	No
Do you understand the meaning of all shapes used?	Yes	No
Do you understand the meaning of all icons used?	Yes	No
Do you understand the meaning of all border styles used? (e.g. solid, dashed, etc)	Yes	No
Do you understand the meaning of all element sizes used? (e.g. small vs large boxes)	Yes	No

ıo Lanubile

Checklist per la revisione di diagrammi di architettura software

Relationships

Does every line have a label describing the intent of that relationship?	Yes	No
Where applicable, do you understand the technology choices associated with every relationship? (e.g. protocols for inter-process communication)	Yes	No
Do you understand the meaning of all acronyms and abbreviations used?	Yes	No
Do you understand the meaning of all colours used?	Yes	No
Do you understand the meaning of all arrow heads used?	Yes	No
Do you understand the meaning of all line styles used? (e.g. solid, dashed, etc)	Yes	No

Progettazione per il cambiamento

- Principio di information hiding
- Obiettivo di alta coesione
- Obiettivo di basso accoppiamento
- Presentazione separata

Principio di information hiding

Ogni componente deve custodire dei segreti al proprio interno

se la decisione che determina il segreto cambia, solo il componente interessato sarà modificato

- Per sottosistemi:
  - L'interfaccia del sottosistema (gruppo di operazioni) è pubblica: **servizio**
  - L'implementazione è privata (e quindi nascosta all'esterno)
- Per package:
  - Solo le classi strettamente necessarie sono pubbliche
  - Tutte le altre sono private
- Per classi:
  - Solo le operazioni strettamente necessarie sono pubbliche
  - Tutte le altre operazioni sono private
  - Tutte le variabili di istanza (attributi) sono private: **incapsulamento dei dati**

Obiettivo di alta coesione

Coesione: misura il grado di dipendenza tra elementi di uno stesso componente (sottosistema o classe)

- Un componente ad alta coesione ha una responsabilità ben definita
- Un componente a bassa coesione si occupa di cose disparate e quindi è:
  - difficile da comprendere
  - difficile da riusare
  - difficile da modificare (e le modifiche saranno frequenti)

Obiettivo: assegna le responsabilità in modo tale da ottenere componenti con responsabilità ben definite

- Una bassa coesione si risolve delegando le responsabilità ad altri componenti

Obiettivo di basso accoppiamento

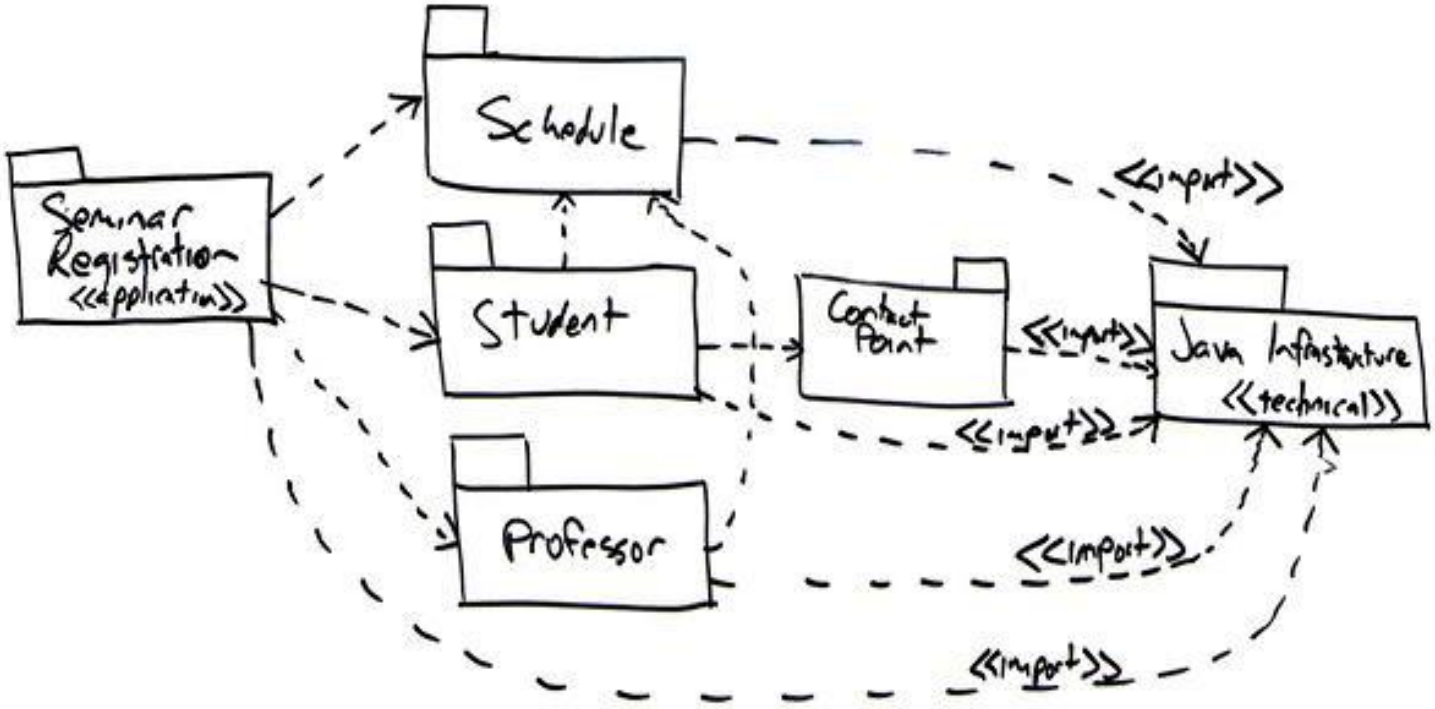
Accoppiamento: misura il grado di dipendenza tra componenti diversi (tra sottosistemi o tra classi)

- Alto accoppiamento: un cambiamento a un componente si propaga facilmente a tutti i componenti che dipendono da esso
- Basso accoppiamento: un cambiamento a un componente non si propaga ad altri componenti

Obiettivo:
assegna le responsabilità ai componenti in modo tale da limitare l'impatto dei cambiamenti

- Un alto accoppiamento si risolve applicando il principio di information hiding

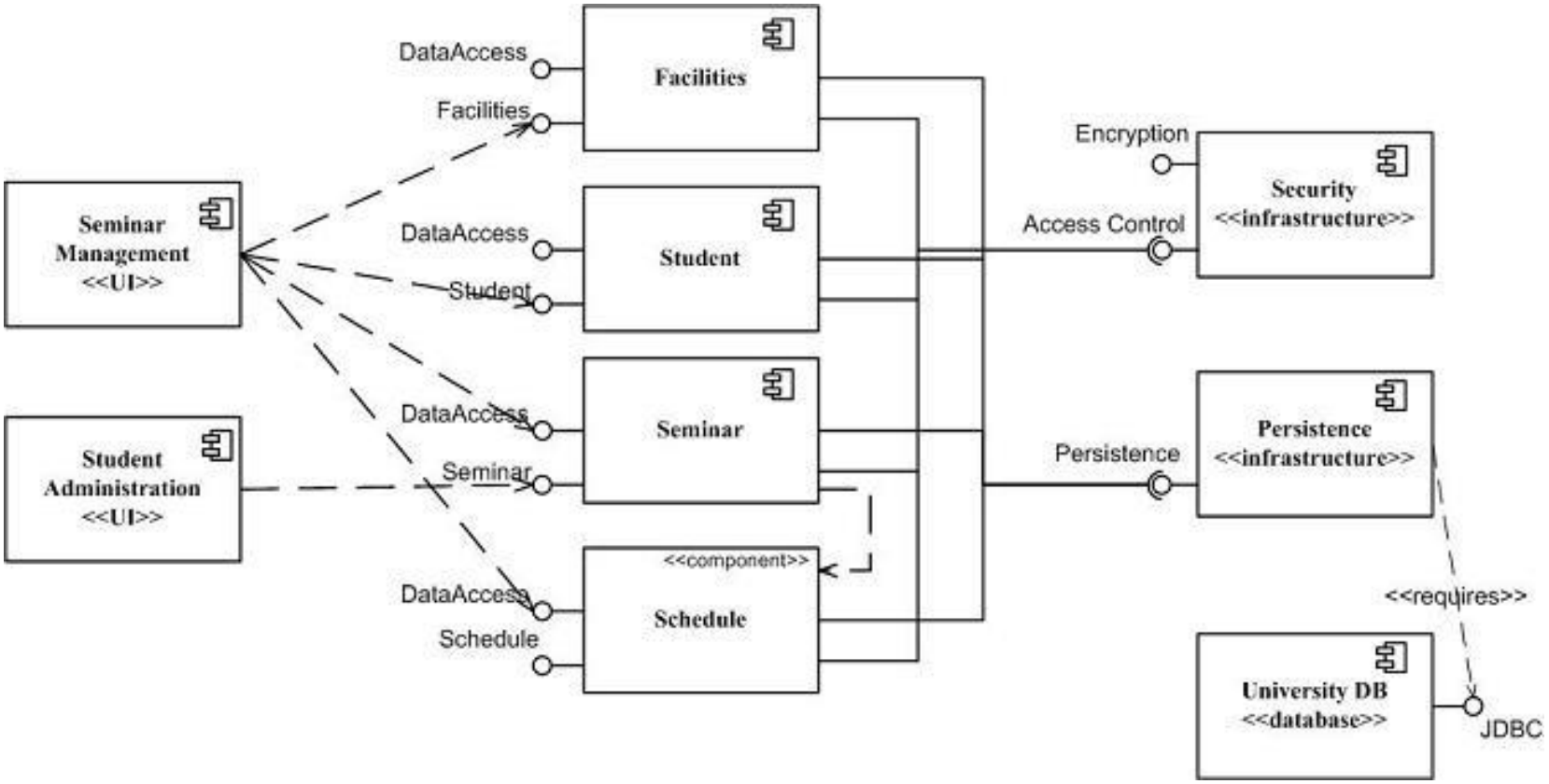
Visualizzazione dell'accoppiamento mediante diagramma dei package



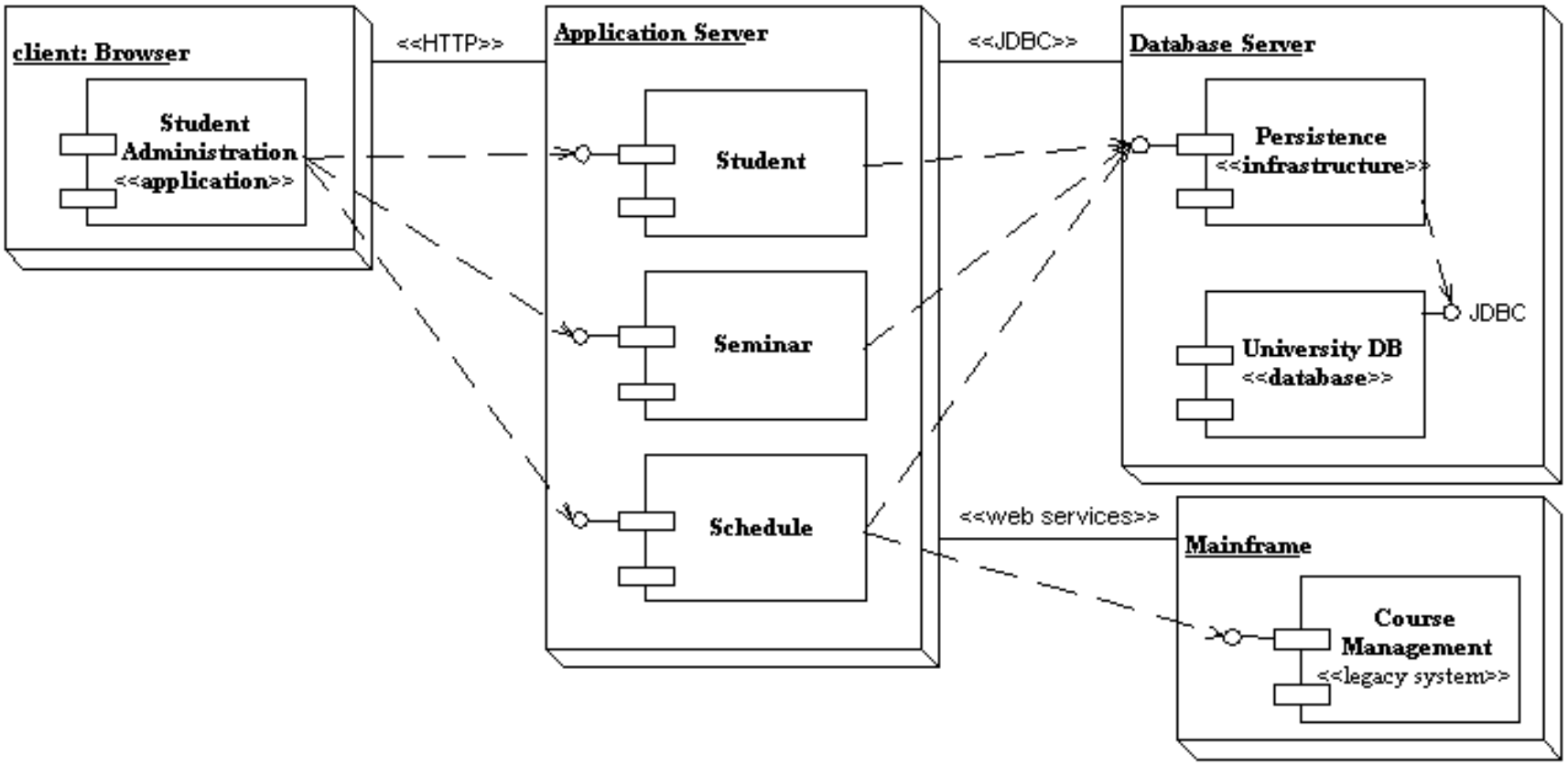
Principio delle dipendenze acicliche:

Il grafo delle dipendenze non deve avere cicli

Visualizzazione dell'accoppiamento mediante diagramma dei componenti



Visualizzazione dell'accoppiamento mediante diagramma di deployment (misto a diagramma dei componenti)



Presentazione separata

- Si intende «*la parte di codice relativa alla presentazione deve essere tenuta separata dal resto dell'applicazione*»
  - Specializzazione di un principio più generale: ***separazione degli interessi***
- La logica di presentazione e la logica di dominio sono più facili da capire se tenute separate
- È possibile esporre la logica di dominio come API/servizio
- È possibile supportare presentazioni differenti senza duplicare il codice
- Senza parti grafiche è possibile scrivere i casi di test con asserzioni testuali

Il Chief Architect

Requisiti

- Conoscenza
- Esperienza
- Creatività