Ingegneria del Software

Progettazione Architetturale

Prof. Paolo Giorgini

A.A. 2024/2025

Progettazione

- Fase in cui si decidono le modalità di passaggio **da "che cosa"** deve essere realizzato nel sistema software **a "come"** la realizzazione deve aver luogo
  - In realtà noi abbiamo già anticipato la fase di sviluppo prima di progettare il sistema (approccio più Agile / Extreme programming)
- La fase di progetto prende in input il *documento di specifica* (analisi dei requisiti) e produce un **documento di progetto** che guida la successiva fase di codifica
- La fase di progetto può essere suddivisa in due sotto-fasi:
  - **progetto architetturale** (o **preliminare),** in cui il sistema software complessivo viene suddiviso in più sottosistemi (decomposizione *modulare)*
  - **progetto dettagliato,** in cui ogni sottosistema identificato viene progettato in dettaglio, scegliendo algoritmi e strutture dati specifiche

Architettura

L'architettura di un sistema descrive la sua forma e struttura: quali sono i suoi componenti e il modo in cui interagiscono

Jerrold Grochow

L'architettura del software di un programma o di un sistema di calcolo è la struttura o le strutture del sistema, che comprendono i componenti software, le loro proprietà visibili e le relazioni fra di essi

Bass, Clements e Kazman [2003]

Che cos'è l'Architettura?

E' un'**astrazione** che consente di:

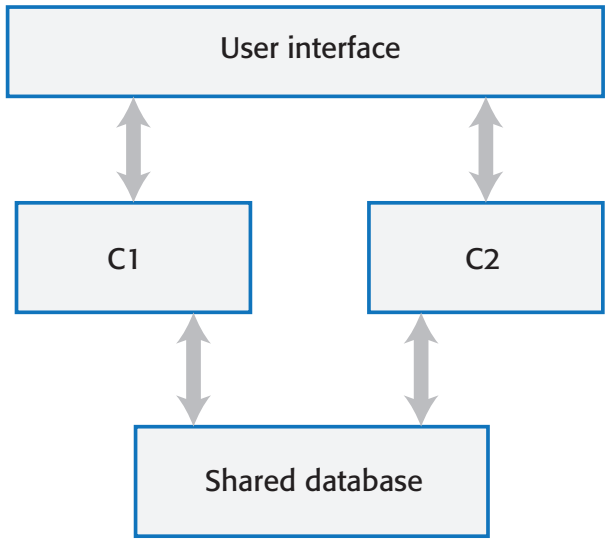
1. definire l'organizzazione, stabilendo ruoli e relazioni
2. valutare le alternative in una fase in cui ogni cambiamento è ancora relativamente semplice
   - analizzando l'efficacia del design nel rispondere ai requisiti
3. avere un modello riferimento per il progetto
   - Processo di raffinamento top-down
   - Gestione del cambiamento: mantenimento / adattamento /evoluzione
   - Testing / Debugging
   - per la comunicazione con gli stakeholders
     - evidenzia le decisioni progettuali e permette di motivarle rispetto ai requisiti
     - design partecipativo, dove tutte le parti in gioco esprimono la propria opinione
     - legame concettuale tra requisiti e implementazione
     - riduzione dei rischi del progetto

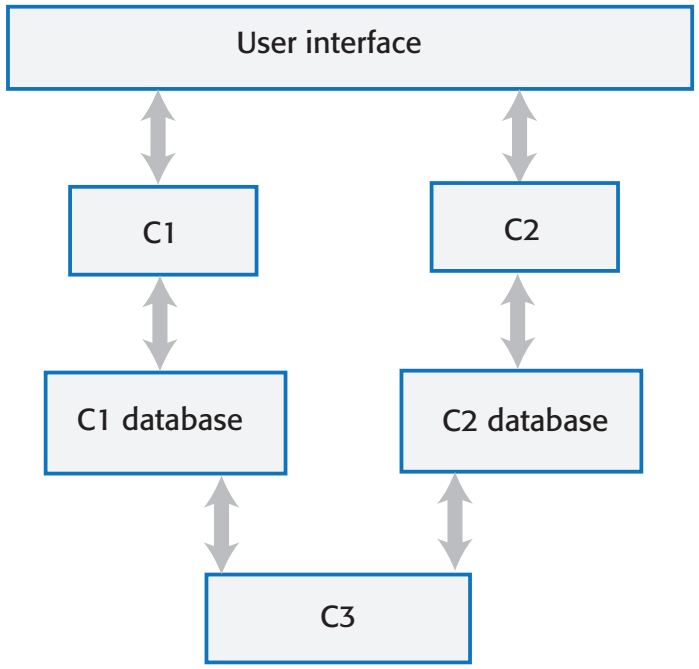
per i requisiti non funzionali

Importanza dell'architettura

- **Reattività:** Il sistema risponde rapidamente alle richieste degli utenti e in un tempo accettabile?
- **Affidabilità:** Le funzioni del sistema operano correttamente e in modo coerente, come previsto dagli utenti e dagli sviluppatori?
- **Disponibilità:** Il sistema è accessibile e in grado di fornire i suoi servizi ogni volta che gli utenti ne hanno bisogno?
- **Sicurezza:** Il sistema protegge sé stesso e i dati degli utenti da accessi non autorizzati e attacchi?
- **Usabilità:** Gli utenti possono navigare nel sistema e utilizzare le sue funzionalità in modo efficiente, senza confusione o errori?
- **Manutenibilità:** Il sistema può essere aggiornato o migliorato facilmente con nuove funzionalità, senza uno sforzo o un costo eccessivo?
- **Resilienza:** Il sistema è in grado di mantenere le sue funzioni principali e fornire servizi anche in caso di guasti parziali o interruzioni esterne?

Es. DB centralizzatio vs. distribuito





Database reconciliation

Performance vs Maintanability

Tradeoff

- Performance vs Maintanability
- Security vs Usability
- Availbility vs time-to-market
- ….

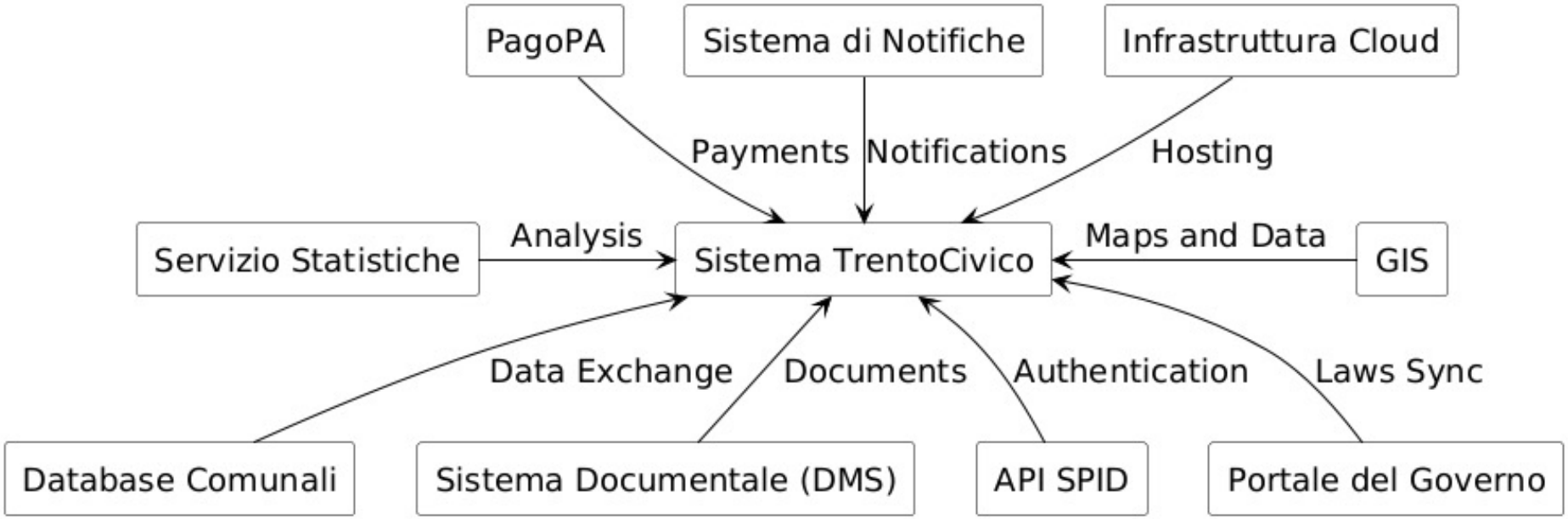
Cosa ottimizziamo? e come?

- priorità aziendali e caratteristiche del progetto

Progettazione dell'Architettura

- Rappresentazione del sistema nel contesto
  - la progettazione deve definire le entità esterne (altri sistemi, apparecchiature, persone) con cui il software interagirà e la natura di tali interazioni - input dell'analisi dei requisiti
- Una volta che il contesto è modellato
  - Tutte le interfacce esterne del software sono state descritte
- Si passa alla specifica del sistema
  - Definendo e raffinando i componenti software che implementano l'architettura
- Processo Iterativo
  - Fino alla definizione completa dell'architettura

Es. Diagramma di contesto



Astrazione e Raffinamento

Astrazione: Riduce la complessità, concentrandosi sugli elementi essenziali del Sistema

- Permette ai progettisti di focalizzarsi sui concetti principali, specificando procedure e dati senza entrare nei dettagli di basso livello

Raffinamento: Aiuta a sviluppare una comprensione chiara e strutturata del progetto

- Gradualmente svela i dettagli di basso livello, costruendo il design in modo incrementale

Processo Graduale: Spesso si tende a concentrarsi subito sui dettagli, saltando i passaggi intermedi di raffinamento.

- Questo può portare a errori, omissioni e rendere il progetto difficile da rivedere

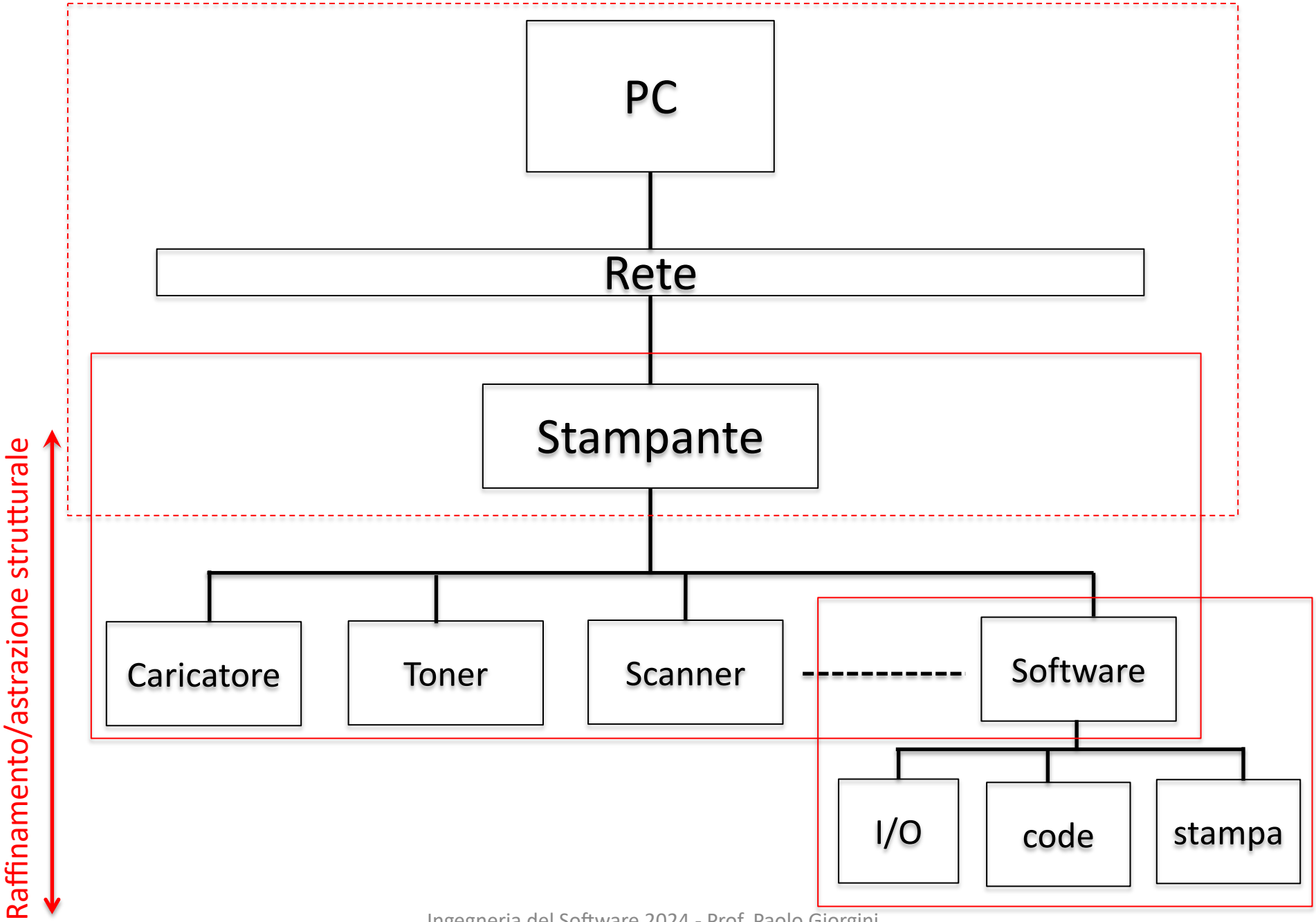
Tipi di Astrazione

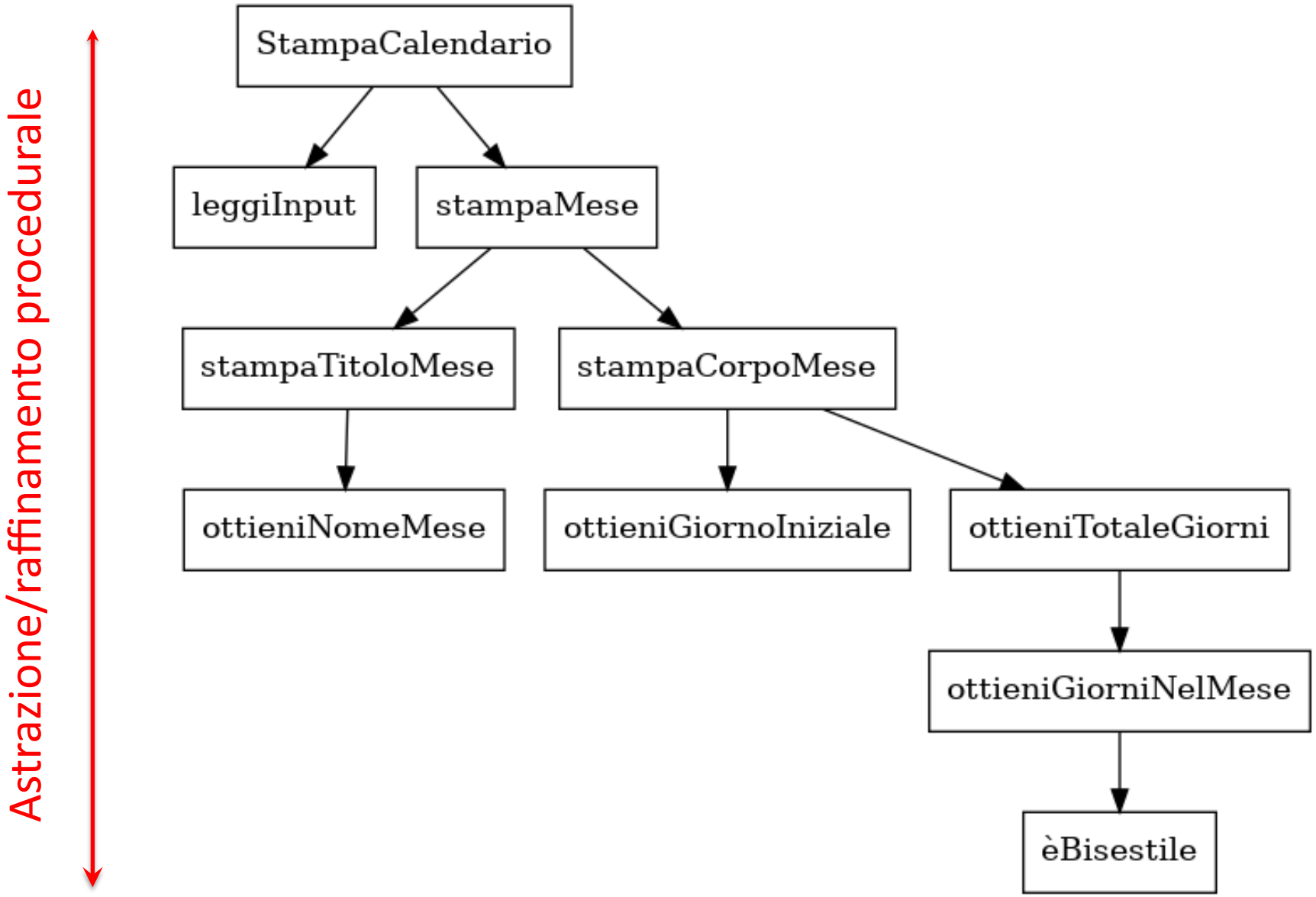
- Astrazione **strutturale**
  - Come i vari componenti di un sistema sono aggregati

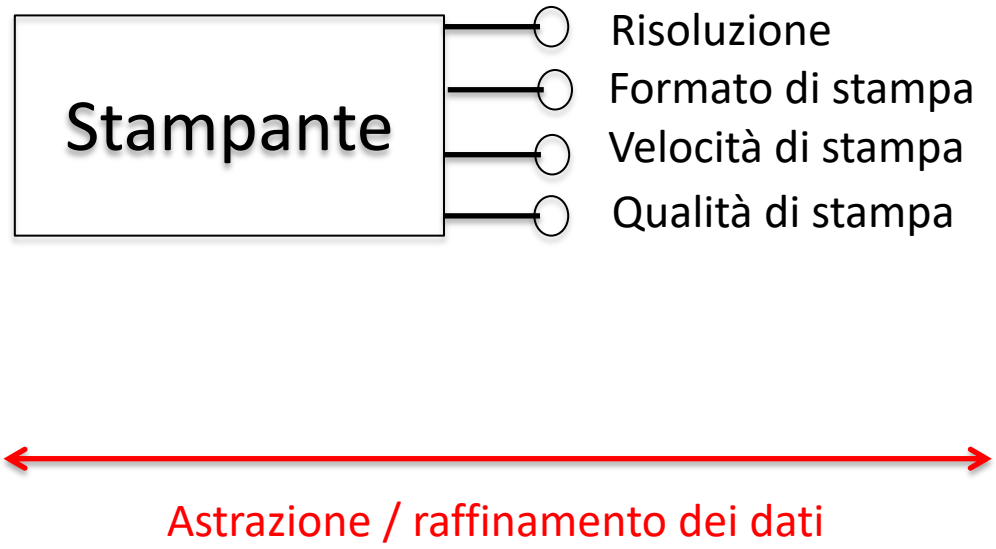
  (es. Una “porta“ è costituita da una struttura in legno, una maniglia, una serratura e dei cardini)
- Astrazione **procedurale**
  - Sequenza di istruzioni di una funzione, specifica e limitata

  (es. «aprire” una porta, che implica una lunga sequenza di passi procedurali: dirigersi verso la porta, afferrare la maniglia, tirare la porta, etc.)
- Astrazione dei **dati**
  - Collezione di dati che descrive un oggetto

  (es. “porta”, che comprende diversi attributi che descrivono la porta: tipo, la direzione di apertura, il meccanismo di apertura, il peso, le dimensioni, etc)


PC
Rete
Stampante
Caricatore
Toner
Scanner
Software
I/O
code
stampa
Raffinamento/astrazione strutturale


Astrazione/raffinamento procedurale
StampaCalendario
leggiInput
stampaMese
stampaTitoloMese
stampaCorpoMese
ottieniNomeMese
ottieniGiornoIniziale
ottieniTotaleGiorni
ottieniGiorniNelMese
èBisestile


Stampante
Risoluzione
Formato di stampa
Velocità di stampa
Qualità di stampa
Astrazione / raffinamento dei dati

Modularità

- Prodotti software fatti di un unico, monolitico, blocco di codice sono difficili da:
  - Manutenere; correggere; capire; riusare
- La soluzione consiste nel suddividere il prodotto software in segmenti più piccoli
  - Il software viene suddiviso che vengono integrati per soddisfare i requisiti del problema

Scomposizione del sistema

- Un **servizio** è un'entità coesa di funzionalità
  - un sistema potrebbe offrire un servizio di posta elettronica composto da servizi per creare, inviare, leggere e memorizzare i messaggi
- Un **componente** è un'entità software che offre uno o più servizi ad altri componenti software o all'utente finale
  - Quando usati da altri componenti, questi servizi sono accessibili mediante API
  - I componenti possono a loro volta impiegare altri componenti per implementare i loro servizi
- Un **modulo** è un insieme di componenti
  - I componenti devono avere qualcosa in comune, ad esempio fornire un insieme di servizi correlati

Moduli e Componenti Software

Modulo Software: unità di codice che raggruppa funzioni o classi correlate, generalmente contenuta in un file o un insieme di file

Caratteristiche principali:

- Più basso livello di astrazione rispetto ai componenti
- Specifico per una singola applicazione o progetto
- Può essere riutilizzato all'interno dello stesso progetto

Es: Un modulo che gestisce le operazioni di autenticazione in un'app spec.

Componente Software: unità di codice indipendente che fornisce una funzionalità specifica, spesso riutilizzabile in più sistemi.

Caratteristiche principali:

- Livello di astrazione più alto rispetto al modulo
- Può essere progettato per essere usato in contesti multipli (diversi progetti o applicazioni)
- Include interfacce ben definite per l'interoperabilità

Es: servizio RESTful che gestisce l'autenticazione utente in vari sistemi

Orchestrazione in un Sistema RESTful

- Un **modulo orchestratore** coordina i vari componenti per completare un flusso complesso come un acquisto in un sistema e-commerce. Ecco un esempio di flusso:
  - **Autenticazione utente** tramite il componente **Gestione Utenti** (GET /users/{id}).
  - **Verifica disponibilità prodotto** tramite il componente **Gestione Prodotti** (`GET /products/{id}`).
  - **Creazione dell'ordine** tramite il componente **Elaborazione Ordini** (`POST /orders`).
  - **Pagamento** tramite il componente **Gestione Pagamenti** (`POST /payments`).
  - **Invio notifica di conferma** tramite il componente **Servizio Notifiche** (`POST /notifications`).

Decomposizione modulare

- Un modulo è un elemento software che:
  - contiene istruzioni, logica di elaborazione e strutture dati
  - può essere compilato separatamente e memorizzato all'interno di una libreria software
  - può essere incluso in un programma
  - può essere usato invocando segmenti di modulo identificati da un nome e da una lista di parametri
  - può usare altri moduli
- La suddivisione in moduli di un sistema software (decomposizione *modulare)* produce come risultato l'identificazione di una **architettura dei moduli** (structure ***chart)***

Decomposizione modulare

- L'architettura dei moduli di un sistema software descrive la **struttura dei moduli**, il modo in cui tali moduli **interagiscono** e la **struttura dei dati** manipolati
- Strategia: “divide et impera”
  - E’ più facile risolvere un problema complesso quando è possibile suddividerlo in più parti facilmente gestibili
  - Detti p1 e p2 due problemi, C la complessità ed E l'effort si ha che:

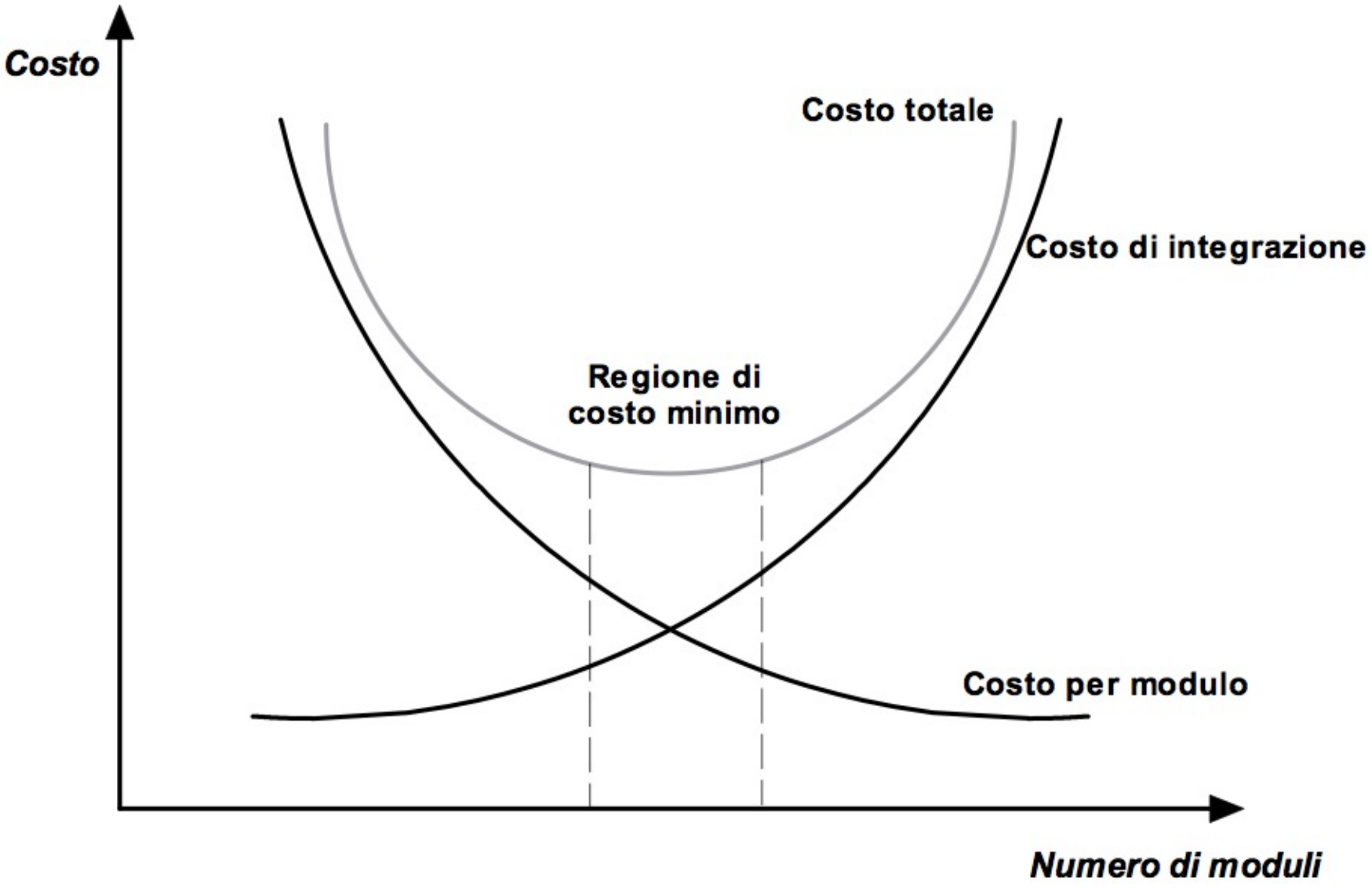
$$\mathbf{C(p1) > C(p2) \Rightarrow E(p1) > E(p2)}$$

$$\mathbf{C(p1+p2) > C(p1) + C(p2)}$$

$$\Downarrow$$

$$\mathbf{E(p1+p2) > E(p1) + E(p2)}$$

Modularità e costo del software



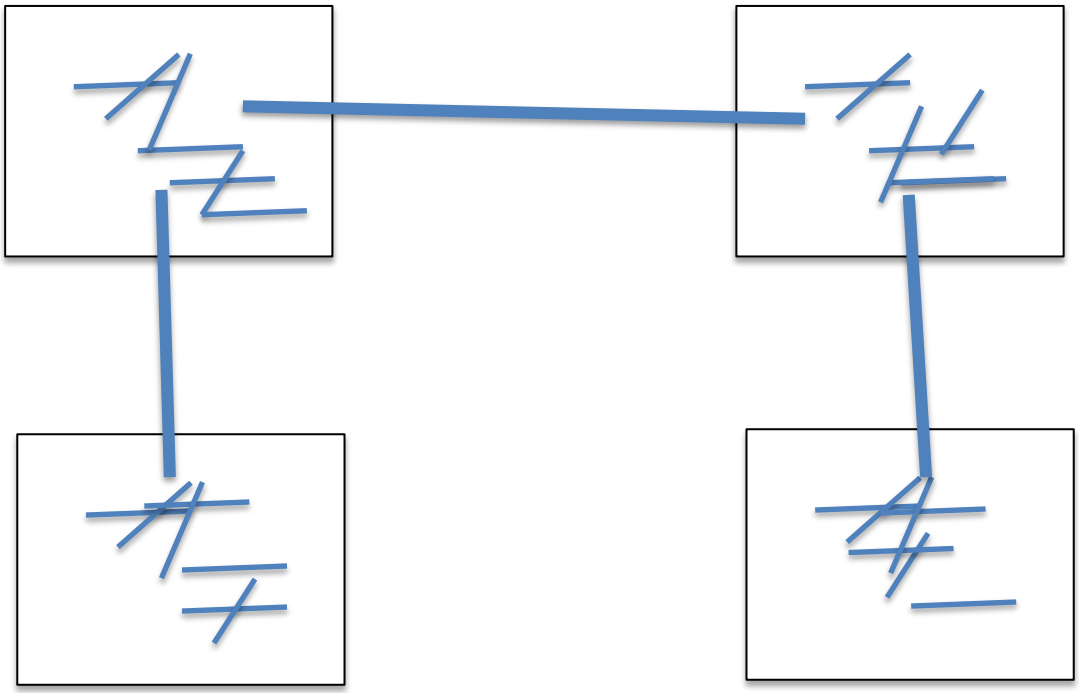
Decomposizione modulare (3)

- Una buona divisione di un prodotto software in moduli è quella che permette di ottenere **l'indipendenza funzionale**:
  - *Massima* coesione (cohesion) *interna* ai moduli
  - *Minimo* grado di accoppiamento (coupling) *tra* i moduli
- Massima coesione e minimo accoppiamento permettono di incrementare:
  - Comprensibilità
  - Manutenibilità
  - Estensibilità
  - Riusabilità

Cohesion/Coupling rispetto a Modularità

Interazioni all'interno di un modulo

Interazioni tra moduli



Coesione

- Per eseguire una *funzione* sono necessarie varie *azioni.*
- Le azioni possono essere concentrate in un singolo modulo oppure sparse tra tanti.

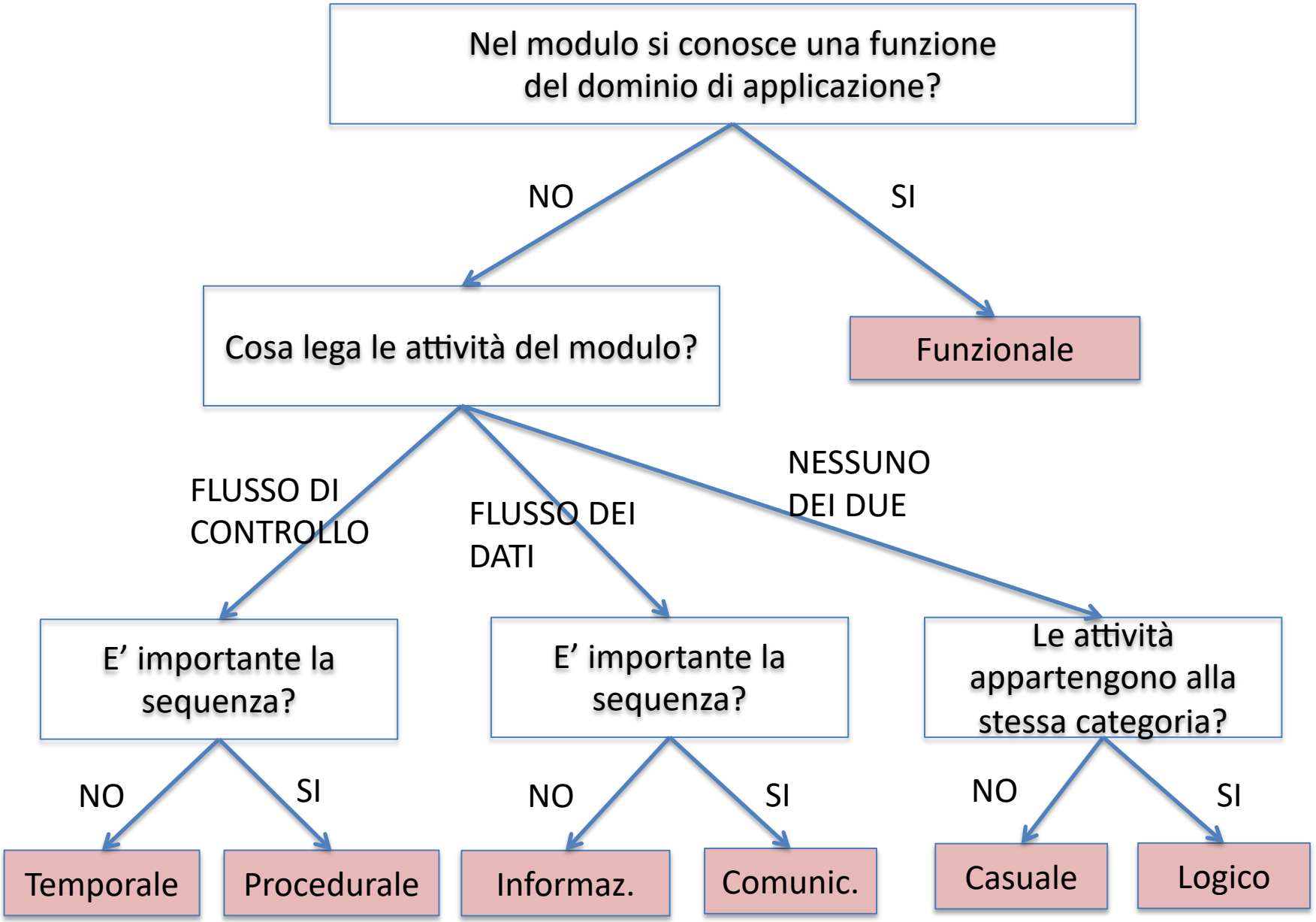
Coesione = Misura in cui il modulo esegue *internamente* tutte le azioni necessarie a espletare una data funzione

(cioè senza interagire con le azioni interne ad altri moduli)

- Coesione misura dunque il grado di *interazione interna* al modulo tra le azioni di una funzione

Livelli di coesione
(1 è il peggiore, *7* il migliore)

1. **Casuale** (nessuna relazione tra gli elementi del modulo)
2. **Logica** (elementi correlati, di cui uno viene selezionato dal modulo chiamante)
3. **Temporale** (relazione di ordine temporale tra gli elementi)
4. **Procedurale** (elementi correlati in base ad una sequenza predefinita di passi da eseguire)
5. **Comunicazionale** (elementi correlati in base ad una sequenza predefinita di passi che vengono eseguiti sulla stessa struttura dati)
6. **Informazionale** (ogni elemento ha una porzione di codice indipendente e un proprio punto di ingresso ed uscita; tutti gli elementi agiscono sulla stessa struttura dati)
7. **Funzionale** (tutti gli elementi sono correlati dal fatto di svolgere una singola funzione)


Nel modulo si conosce una funzione del dominio di applicazione?
NO
SI
Cosa lega le attività del modulo?
Funzionale
FLUSSO DI CONTROLLO
FLUSSO DEI DATI
NESSUNO DEI DUE
E' importante la sequenza?
E' importante la sequenza?
Le attività appartengono alla stessa categoria?
NO
SI
NO
SI
NO
SI
Temporale
Procedurale
Informaz.
Comunic.
Casuale
Logico

Coupling - Accoppiamento

- Esprime la forza di inter-connessione fra moduli
  - maggiori connessioni implicano maggiore dipendenza fra moduli, ossia maggiore conoscenza di un modulo richiesta per comprendere un altro modulo (implicazioni su comprensibilità e manutenibilità del modulo)
- Minimizzare l'accoppiamento, semplificando:
  - tipo di connessione (solo connessioni attraverso l'interfaccia)
  - complessità dell'interfaccia (numero e tipo di parametri) – tipo di flusso di informazioni fra moduli (dati / controlli)

Coupling

- Misura il grado di *accoppiamento* tra moduli
- Livelli di *coupling* (1 è il peggiore, *5* il migliore):

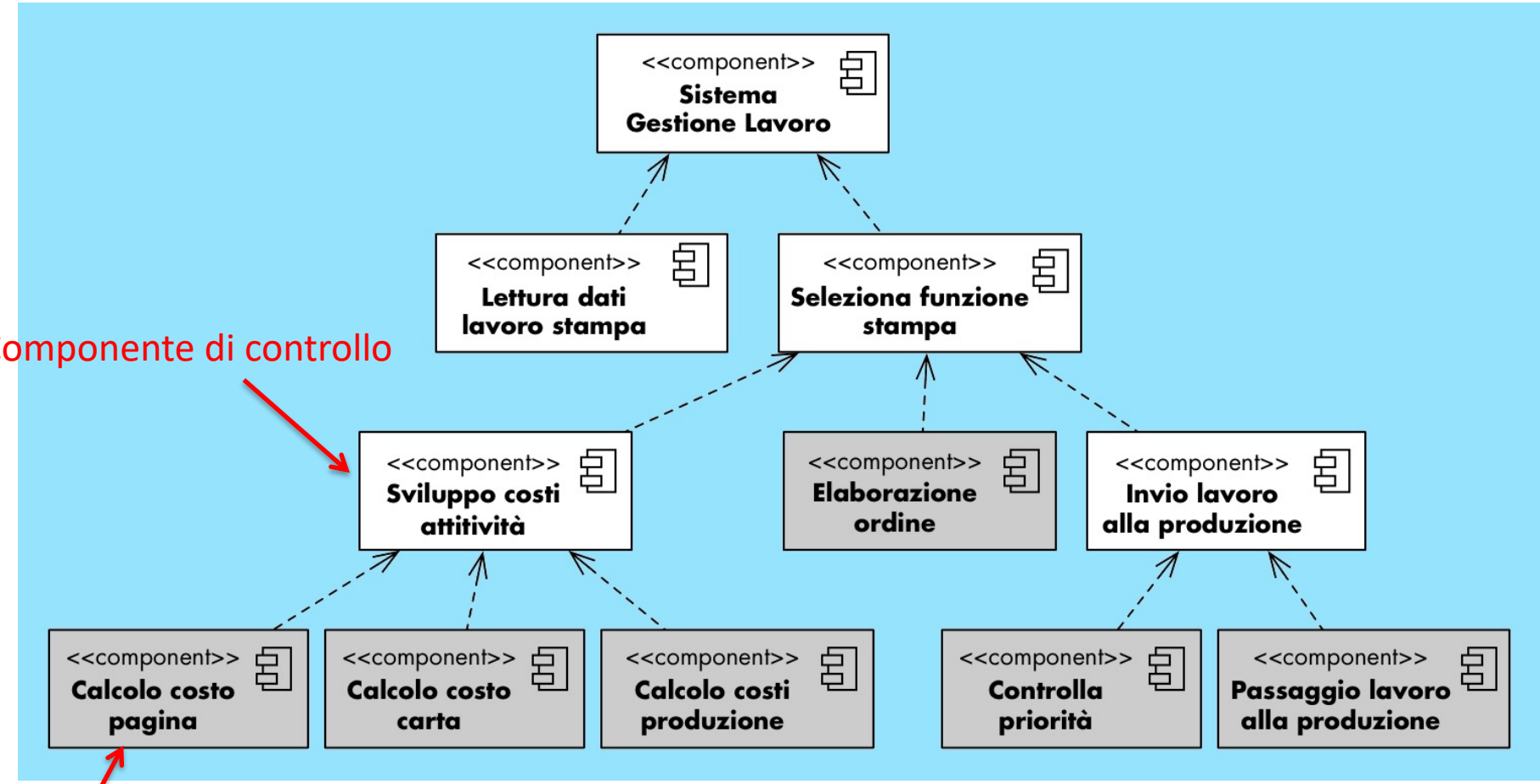
1. **Content** (un modulo fa diretto riferimento al contenuto di un altro modulo).
2. **Common** (due moduli che accedono alla stessa struttura dati)
3. **Control** (un modulo controlla esplicitamente l'esecuzione di un altro modulo)
4. **Stamp** (due moduli che si passano come argomento una struttura dati, della quale si usano solo alcuni elementi)
5. **Data** (due moduli che si passano argomenti omogenei, ovvero argomenti semplici o strutture dati delle quali si usano tutti gli elementi)

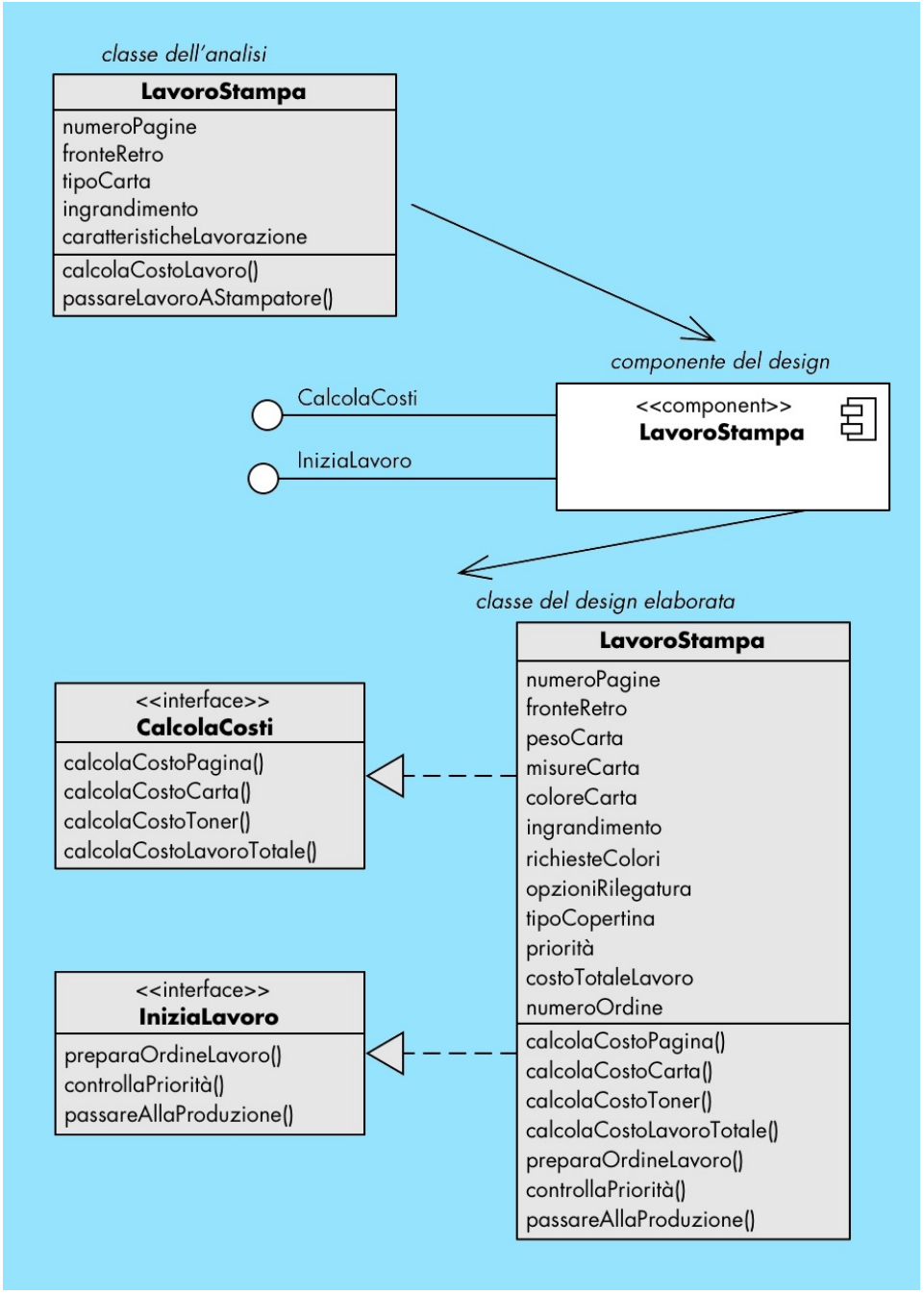
Obiettivi della progettazione modulare

- **massimizzare la coesione** interna: per migliorare la comprensibilità del modulo e la sua modificabilità;
- **minimizzare il grado di accoppiamento** fra i moduli: per migliorare la comprensibilità di ciascun modulo.

Con riferimento all'astrazione, un lasco accoppiamento ⇒ un'astrazione implementata in un modulo deve essere largamente indipendente da ogni altra astrazione

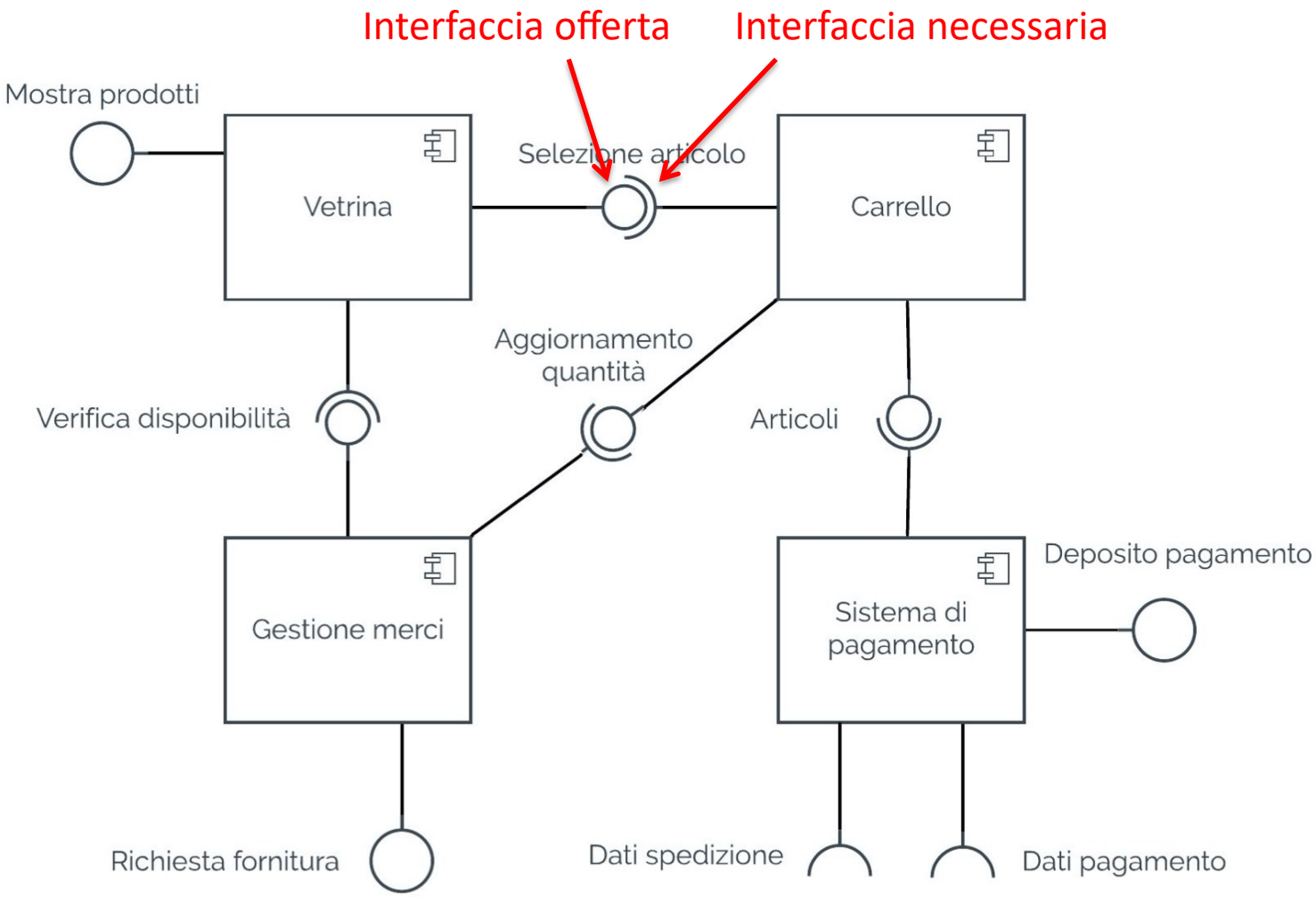
Componet Diagram UML sistema convenzionale





I dettagli di `LavoroStampa` sono elaborati

- Informazioni più dettagliate ed una descrizione estesa delle operazioni necessarie per implementare la classe
- Le interfacce implicano comunicazioni e collaborazioni con altri componenti
  - `calcolaCostoPagina()` deve collaborare con un componente `TabellaPrezzi`
  - `controllaPriorita()` deve collaborare con una componente `CodaLavori`


Interfaccia offerta
Interfaccia necessaria
Mostra prodotti
Vetrina
Selezione articolo
Carrello
Aggiornamento quantità
Verifica disponibilità
Articoli
Gestione merci
Sistema di pagamento
Deposito pagamento
Richiesta fornitura
Dati spedizione
Dati pagamento

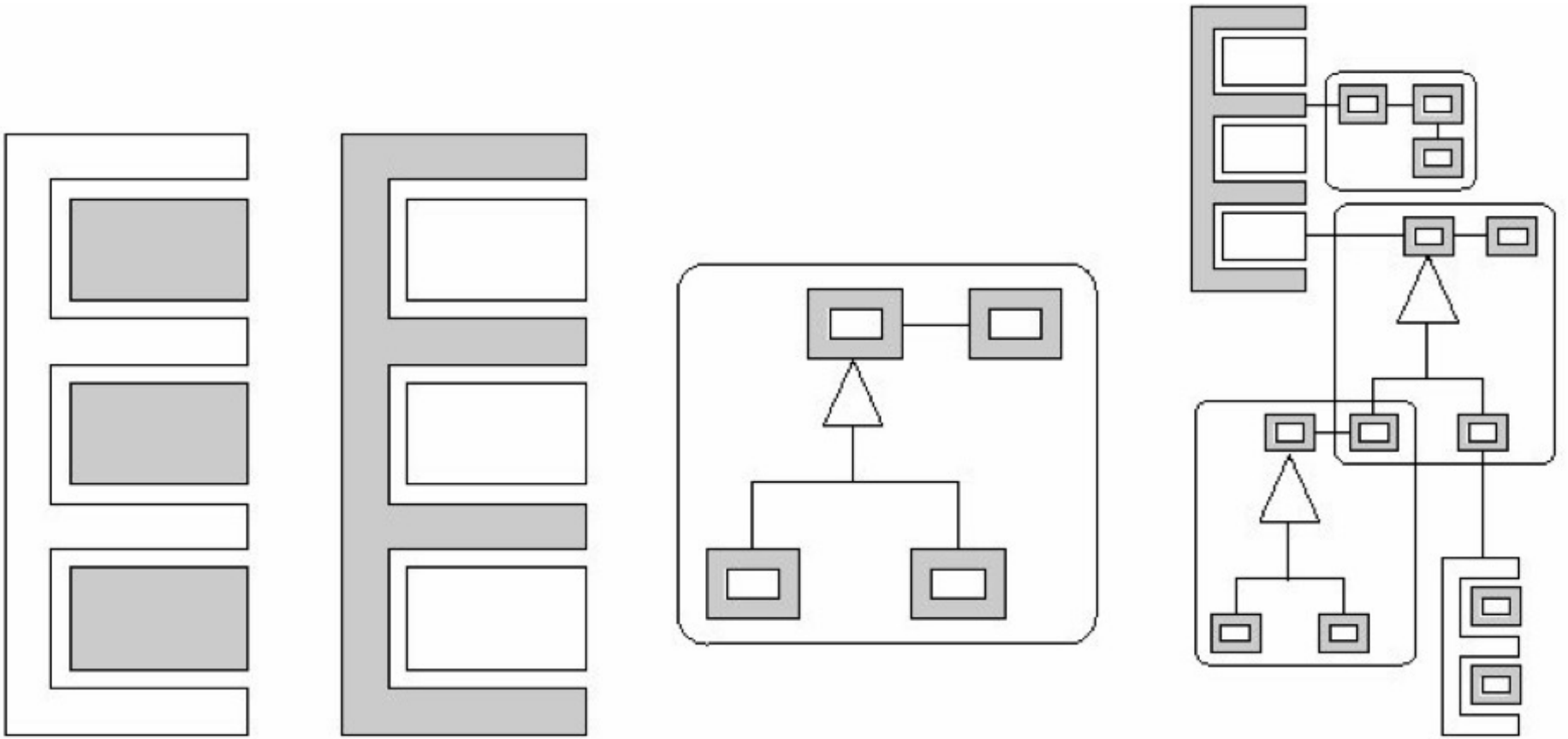
Riusabilità

- La riusabilità fa riferimento all'utilizzo di componenti sviluppati per un prodotto all'interno di un prodotto differente
- Per componente riusabile si intende non solo un modulo o un frammento di codice, ma anche progetti, parti di documenti, insiemi di test data o stime di costi e durata
- Vantaggi:
  - netta diminuzione di costi e tempi di produzione del software
  - incremento dell'affidabilità dovuto all'uso di componenti già convalidati

Riusabilità (2)

- La riusabilità nella fase di progetto si applica a:
  (a) Moduli software
  (b) application framework, che incorpora la logica di controllo di un progetto
  (c) design pattern, che identifica una soluzione di progetto ricorrente in applicazioni dello stesso tipo
  (d) Architetture software comprendenti (a), (b) e (c)

Riusabilità

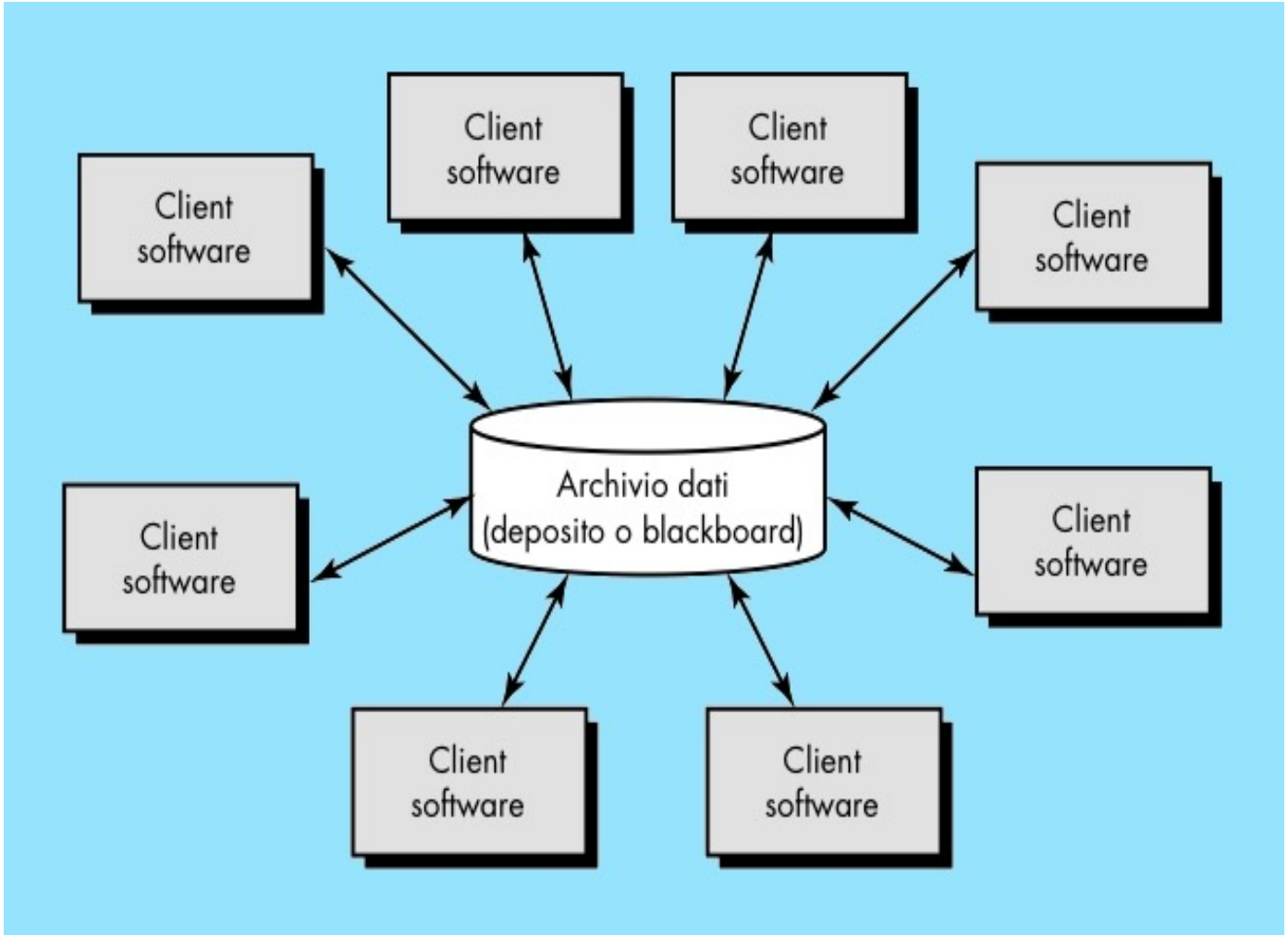


(a) (b) (c) (d)

moduli software ***application framework*** ***design pattern*** ***architettura software***

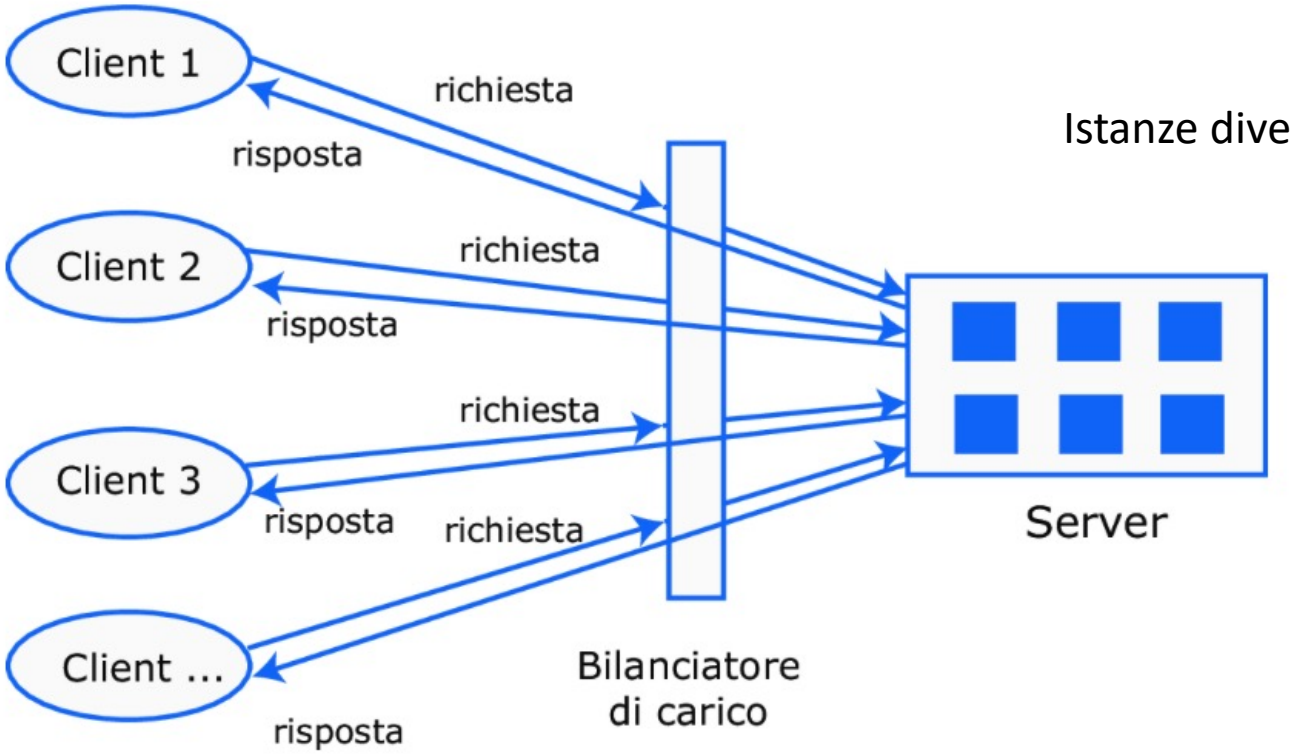
Architetture e Modelli

Modelli basati sui dati



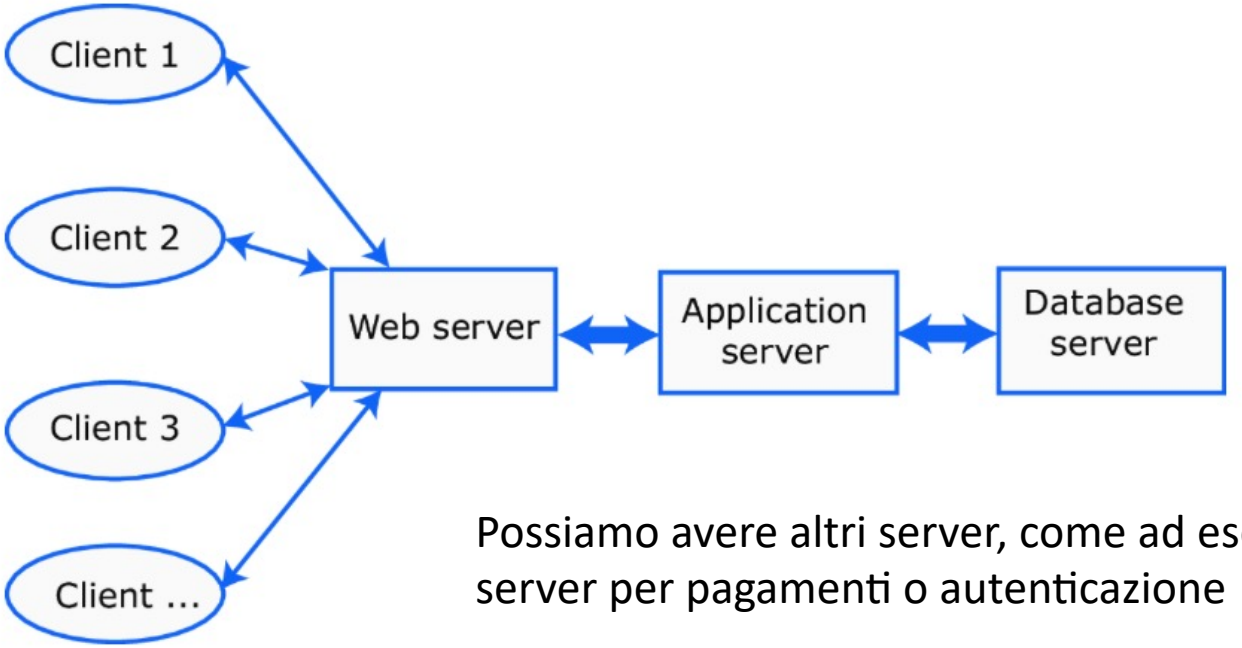
Modello client-server

- Interfaccia implementata sul dispositivo utente
- La funzionalità è distribuita tra client e uno o più server
  - *Distribution architecture*: definisce i server del sistema e l'allocazione dei componenti a tali server



Architettura multi-tier

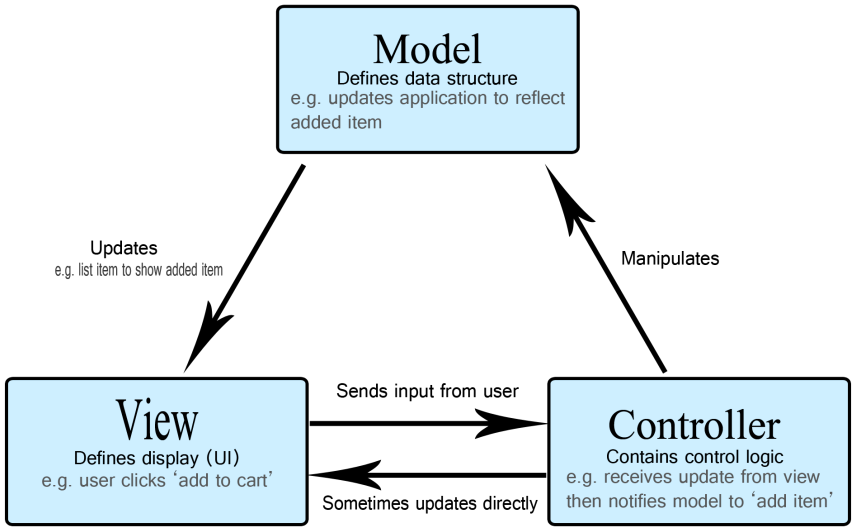
- Piu' server comunicanti con reposanbilita' diverse
  - Tipicamente le applicazioni server includono tre tipi di server



Possiamo avere altri server, come ad esempio server per pagamenti o autenticazione

Il pattern Model-View-Controller

- **L'utente interagisce con la Vista (View):** Ad esempio, cliccando un pulsante o selezionando un'opzione
- **La Vista invia l'input al Controller (Controllore):** La **View** invia i dati dell'interazione al **Controller**
- **Il Controller aggiorna il Model:** Il **Controller** elabora i dati e modifica lo stato nel **Model**
- **Il Model aggiorna la Vista:** Una volta che il **Model** è aggiornato, il **Controller** aggiorna la **View** per riflettere i nuovi dati

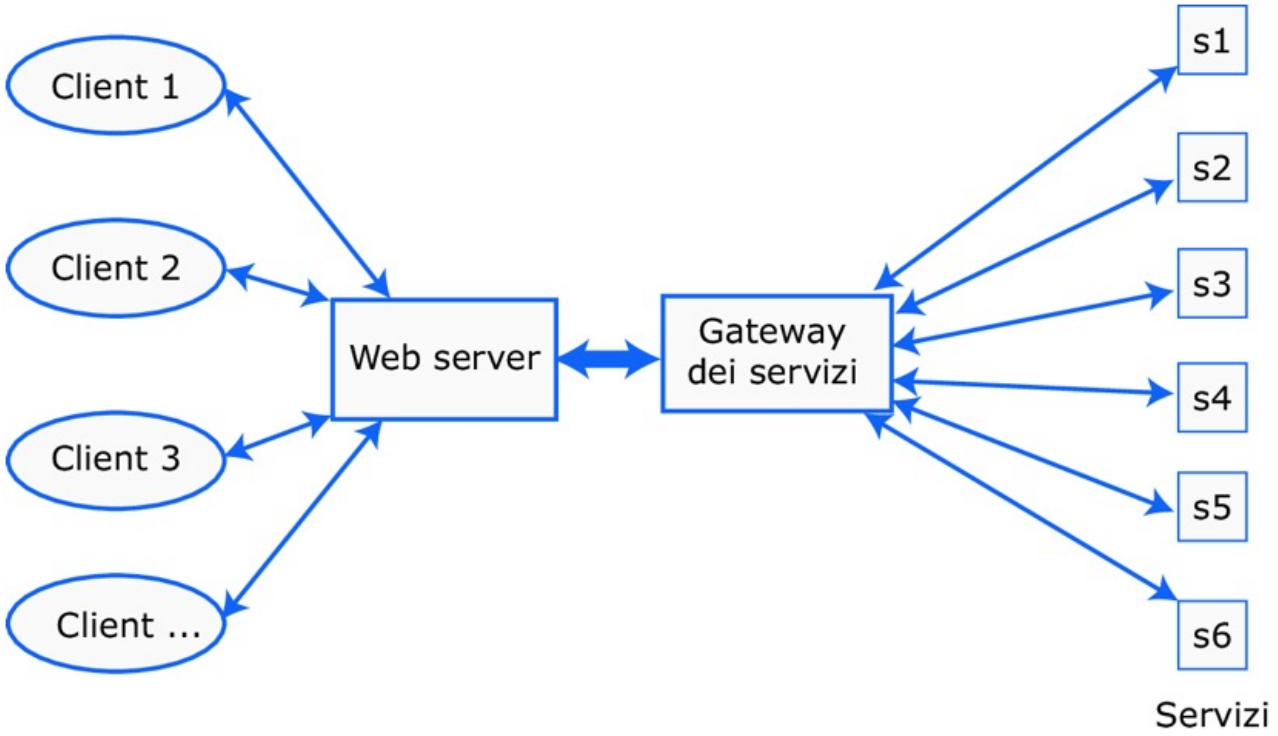


Flusso tra Client e Server con MVC

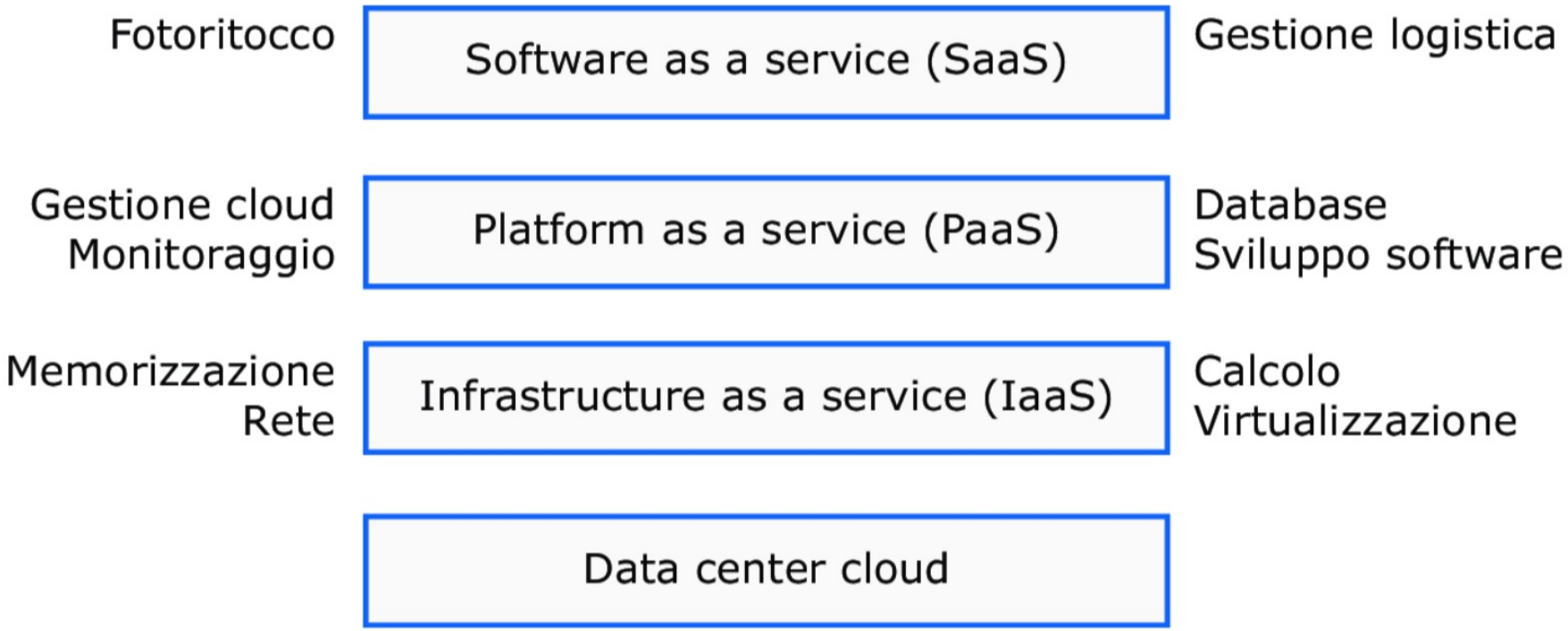
1. **Interazione dell'utente (Client):**
   - L'utente interagisce con la **View** sul **Client** (ad esempio, clicca un pulsante)
2. **Richiesta al Server (Controller Client):**
   - Il **Controller** del **Client** riceve l'input, lo elabora e invia una richiesta HTTP al **Server**
3. **Gestione della logica (Server):**
   - Il **Controller** del **Server** riceve la richiesta, interagisce con il **Model** (ad esempio, accede al database) e aggiorna i dati
4. **Risposta al Client (Controller Server):**
   - Il **Server** restituisce i dati aggiornati al **Client**, che poi aggiorna la **View**.

Service-Oriented Architecture

- Alternativa all'architettura client-server multi-tier
  - Molti server possono essere coinvolti nella fornitura dei servizi
  - Servizi sono componenti stateless – possono essere replicati e migrare da un computer all'altro



Services


Fotoritocco
Software as a service (SaaS)
Gestione logistica
Gestione cloud
Monitoraggio
Platform as a service (PaaS)
Database
Sviluppo software
Memorizzazione
Rete
Infrastructure as a service (IaaS)
Calcolo
Virtualizzazione
Data center cloud

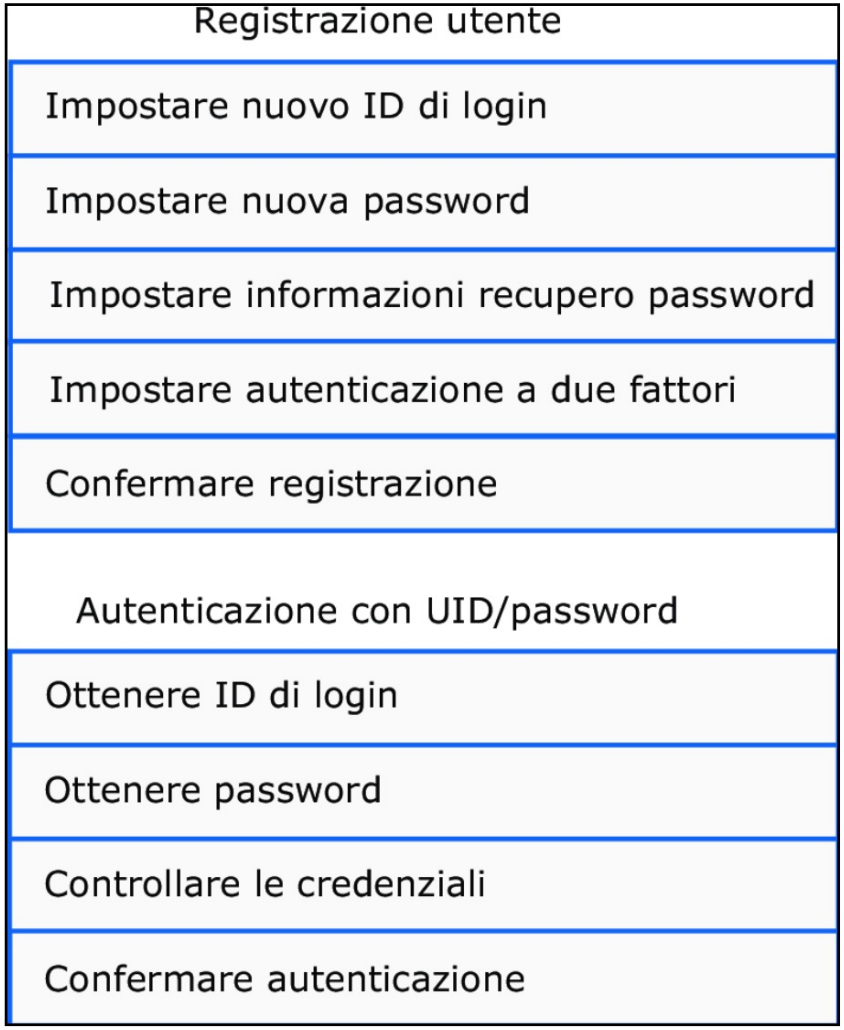
Servizi e microservizi

- Sviluppo orientato ai servizi (web services)
  - Protocolli e standard XML (SOAP e WSDL)
  - Orchestrazione dei servizi
  - Messaggi XML lunghi e complessi da analizzare
- Microservizi
  - Singola funzionalità operativa
  - Completamente indipendenti, con un DB proprio + interfaccia utente
  - Semplici da sostituire senza cambiare il sistema

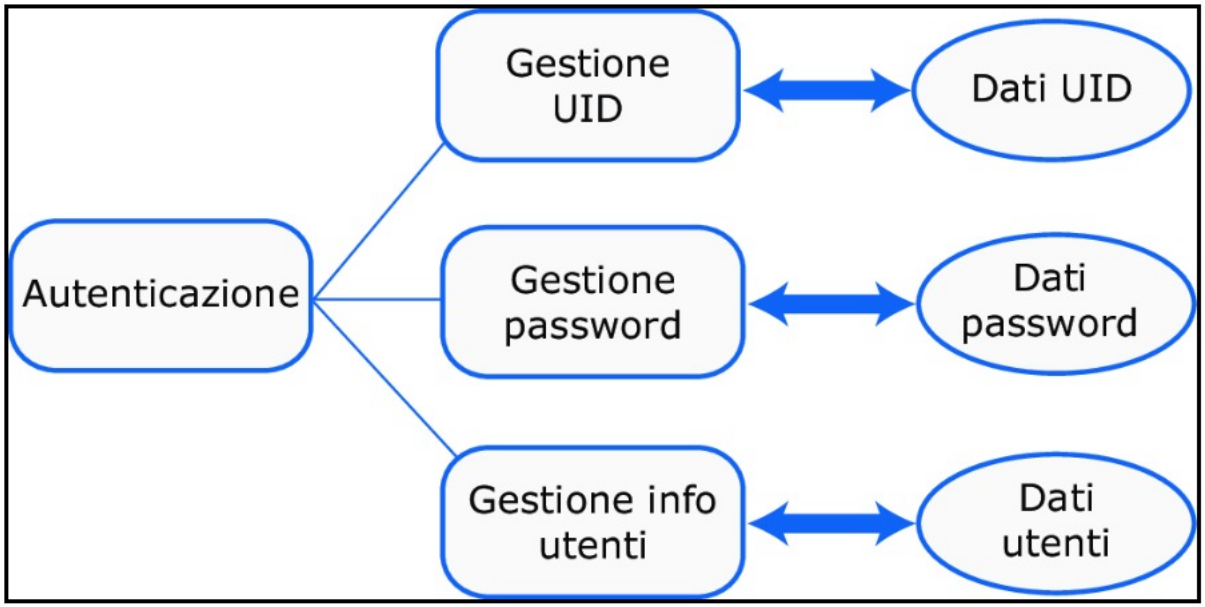
Esempio microservizi

Registrazione utente e autenticazione con UID/Password

Feature da implmentare con microservizi, come gestire i dati (condivisi – ogni micro deve avere i sui dati)



Si può identificare un microservizio per ogni dato logico da gestire



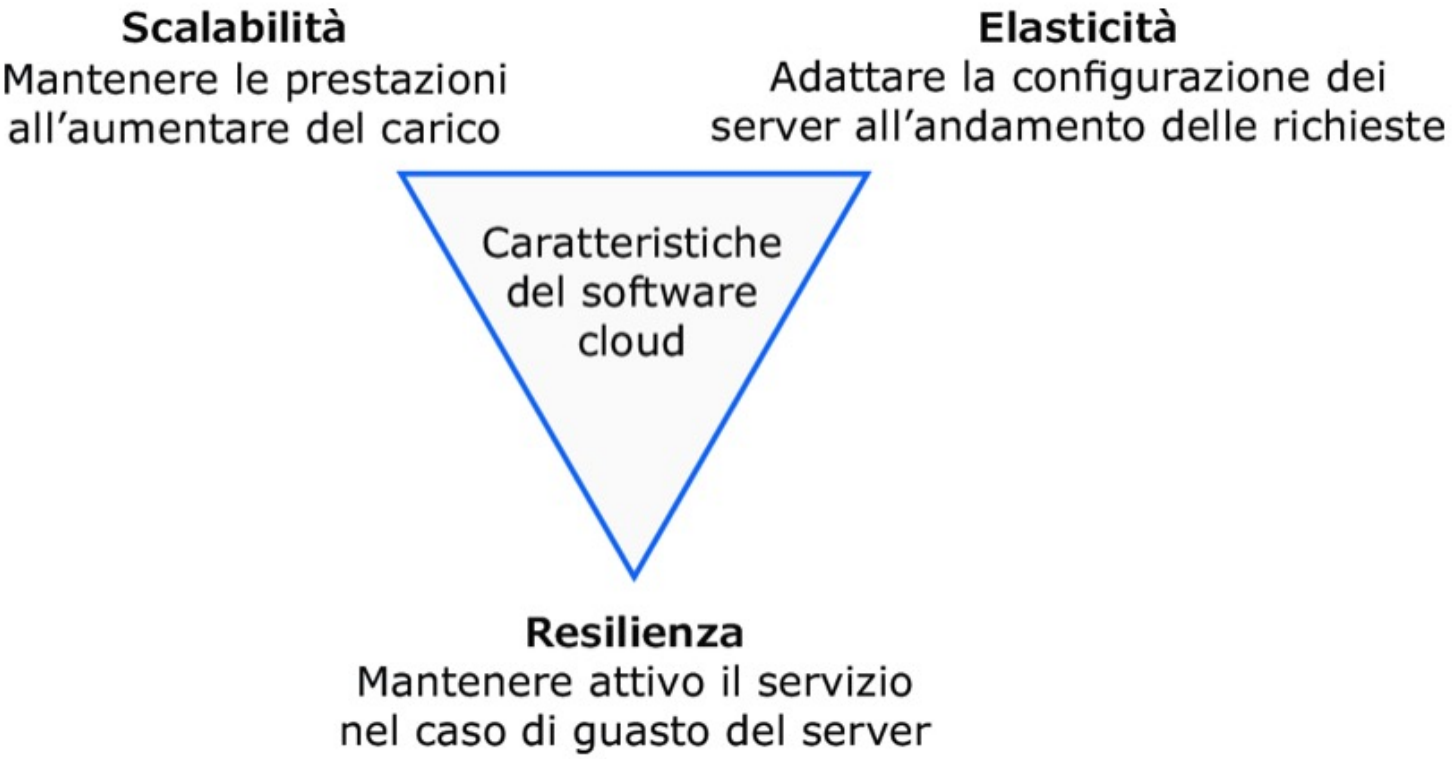
Questioni tecnologiche

- Le tecnologie scelte possono influire sull'architettura complessiva del sistema

Tecnologia	Decisione progettuale
Database	Meglio usare un database relazionale SQL o uno di tipo NoSQL?
Piattaforma	Il prodotto dovrebbe essere rilasciato su app mobile e/o su una piattaforma web?
Server	Meglio usare server dedicati in sede o progettare il sistema per l'esecuzione su un cloud pubblico? Nel caso del cloud, meglio usare Amazon, Google, Microsoft o altre opzioni?
Open source	Esistono componenti open-source che possono essere utilizzati?
Strumenti di sviluppo	Gli strumenti di sviluppo utilizzati limitano la libertà delle scelte architetturali?

Software sul cloud

- Server virtuali

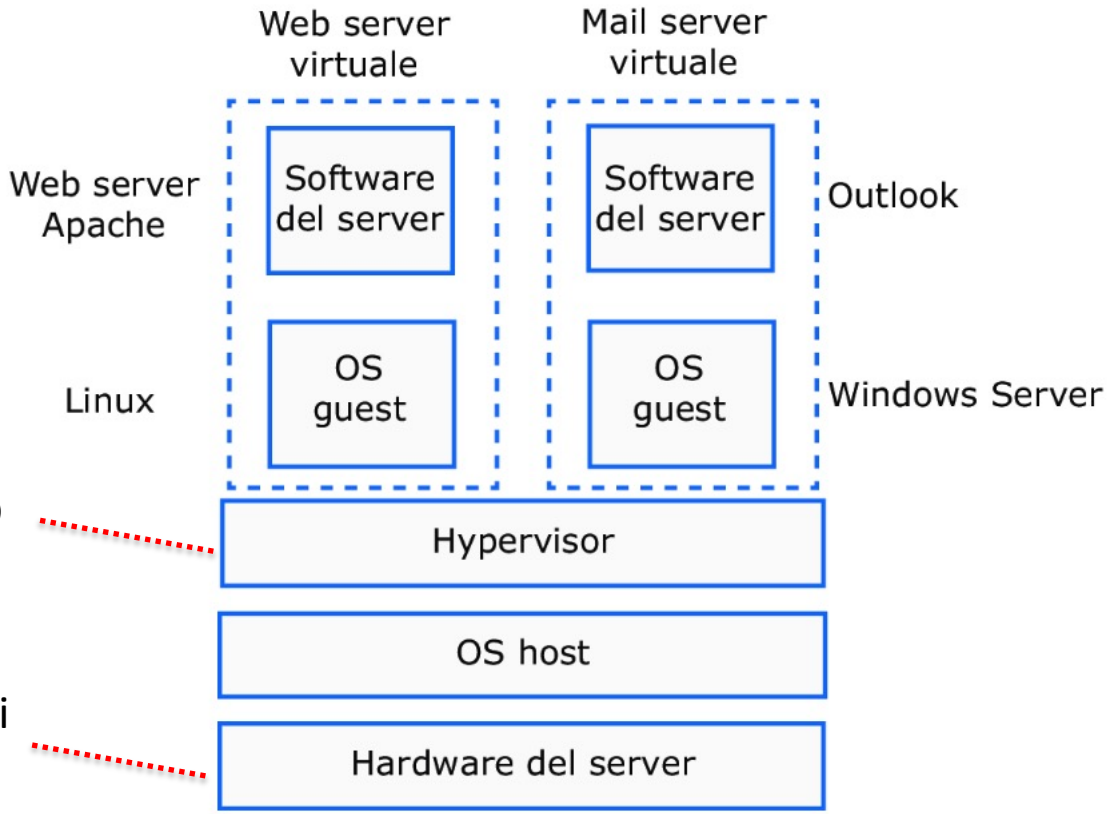


Vantaggi del cloud

Fattore	Vantaggio
Costo	Si evitano i costi iniziali di acquisto dell'hardware.
Tempi di inizio	Non è necessario aspettare la consegna dell'hardware per iniziare a lavorare. I server sul cloud possono essere attivi in pochi minuti.
Scelta del server	Se i server noleggiati non sembrano abbastanza potenti, è possibile passare a server migliori. Possono anche essere aggiunti server per necessità a breve termine, come ad esempio i test di carico.
Sviluppo distribuito	Se il team di sviluppo è distribuito, e lavora da luoghi diversi, tutti i membri avranno lo stesso ambiente di sviluppo e potranno condividere tutte le informazioni senza difficoltà.

Macchine virtuali

- VM per implementare i server virtuali
  - viene caricato tutto il software necessario



Fornisce un'emulazione HD che simula il funzioanmento dell'HD

HD condiviso da più emulatori eseguiti in parallelo

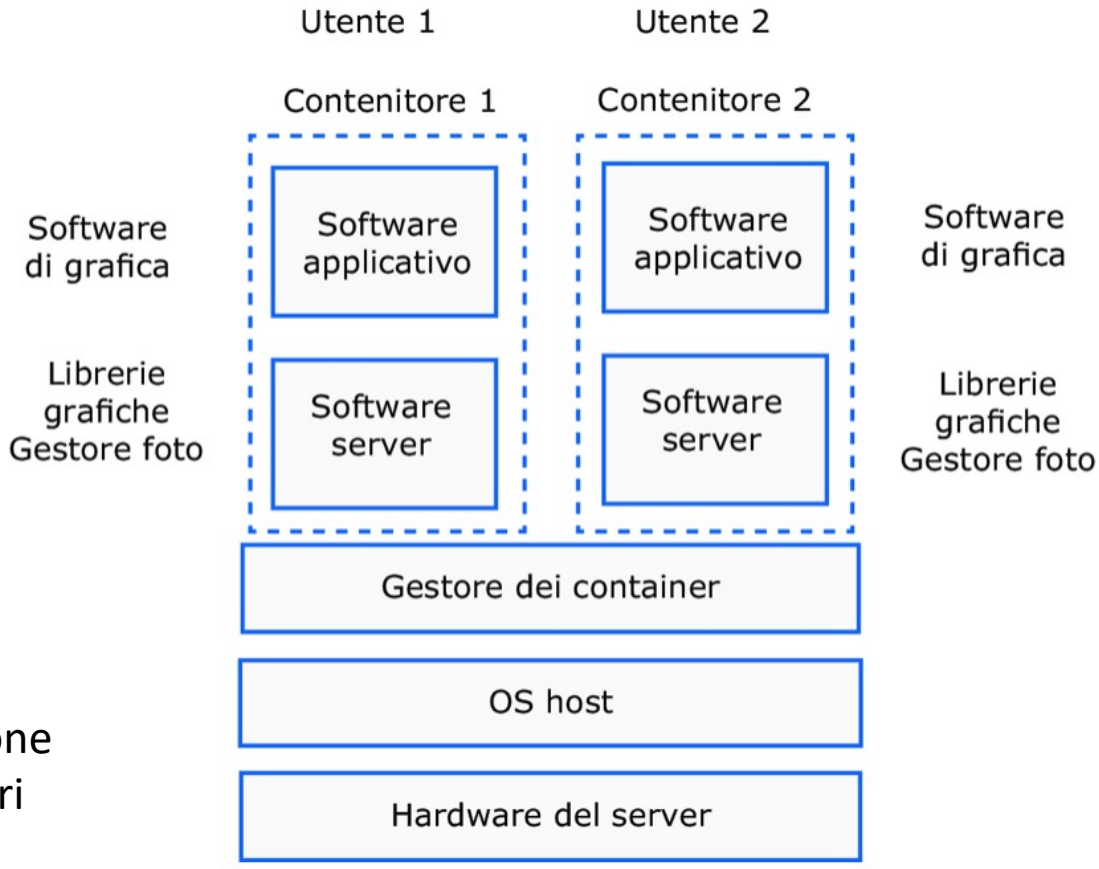
Limiti delle VM

- Per creare una VM è necessario caricare e avviare un OS grande e complesso
  - Diversi minuti per installare l'OS e configurare il software -> impossibile reagire istantaneamente alle variazioni delle richieste avviando e spegnendo delle VM
  - Spesso la generalità delle VM non è necessaria
- Uso di container
  - Sveltisce il processo di deplyment dei server virtuali sul cloud

Container

- Consentono di raggruppare e isolare le applicazioni insieme al relativo ambiente di runtime, che include tutti i file necessari per l'esecuzione.
- Tecnologia di virtualizzazione dei OS che permette a più server indipendenti di condividere un unico OS
  - Utili per per offrire servizi applicativi isolati nei quali ciascun utente vede la propria versione di un'applicazione
- Mbyte vs Gbyte delle VM
- Avviati e spenti in pochi secondi

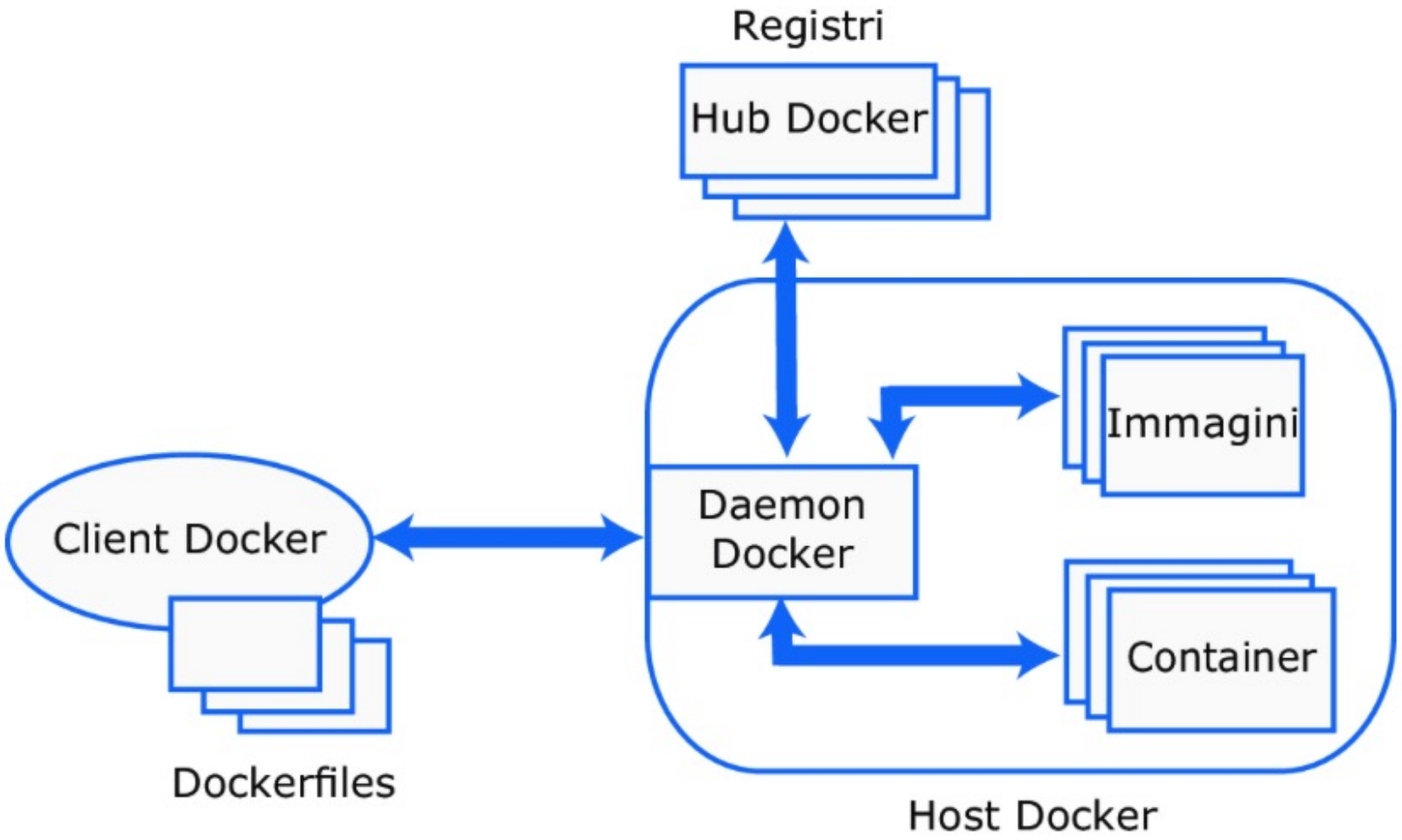
Esempio prodotto software per la grafica che impiega librerie grafiche di base e un sistema di gestione delle foto – per ciascun utente viene creato un container con il software di supporto e l'applicazione



Il container creato (lato client) viene dislocato su un OS host che grazie al sistema di gestione dei container garantisce che il processo in esecuzione nel container sia isolato da tuti gli altri processi

Docker

- Metodo standard, veloce e di facile impiego per la gestione dei container
  - Permette agli utenti di definire il software da includere in un container come immagine Docker
  - Include un sistema run-time che puo' creare e gestire container utilizzando immagini Docker
  - L'implementazione del file system di Docker prevede che l'immagine contanga solo i file diversi da quelli standard del OS – > immagini compatte e quindi veloci da caricare


Registri
Hub Docker
Immagini
Daemon Docker
Client Docker
Container
Dockerfiles
Host Docker

Elemento	Funzione
Daemon Docker	È un processo eseguito su un server host ed è utilizzato per impostare, avviare, fermare e monitorare i container, nonché per costruire e gestire immagini locali.
Client Docker	Questo software è utilizzato da sviluppatori e system manager per definire e controllare i container.
Dockerfiles	Definiscono le applicazioni eseguibili (immagini) come una serie di comandi di impostazione che specificano il software da includere in un container. Ogni container deve essere definito da un Dockerfile associato.
Immagine	Un Dockerfile viene interpretato per creare un'immagine Docker, ovvero un insieme di directory con il software e i dati specificati installati nei posti giusti. Le immagini sono configurate in modo da essere applicazioni Docker eseguibili.
Hub Docker	È un registro delle immagini create, che possono essere usate per configurare container o come punti di partenza per definire nuove immagini.
Container	I container eseguono le immagini: un'immagine viene caricata in un container e l'applicazione definita dall'immagine inizia l'esecuzione. I container possono essere spostati da un server all'altro senza modifiche, e possono essere replicati su più server. È possibile apportare modifiche a un container Docker (modificare file, ad esempio), ma le modifiche devono essere confermate per creare una nuova immagine e riavviare il container.