UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

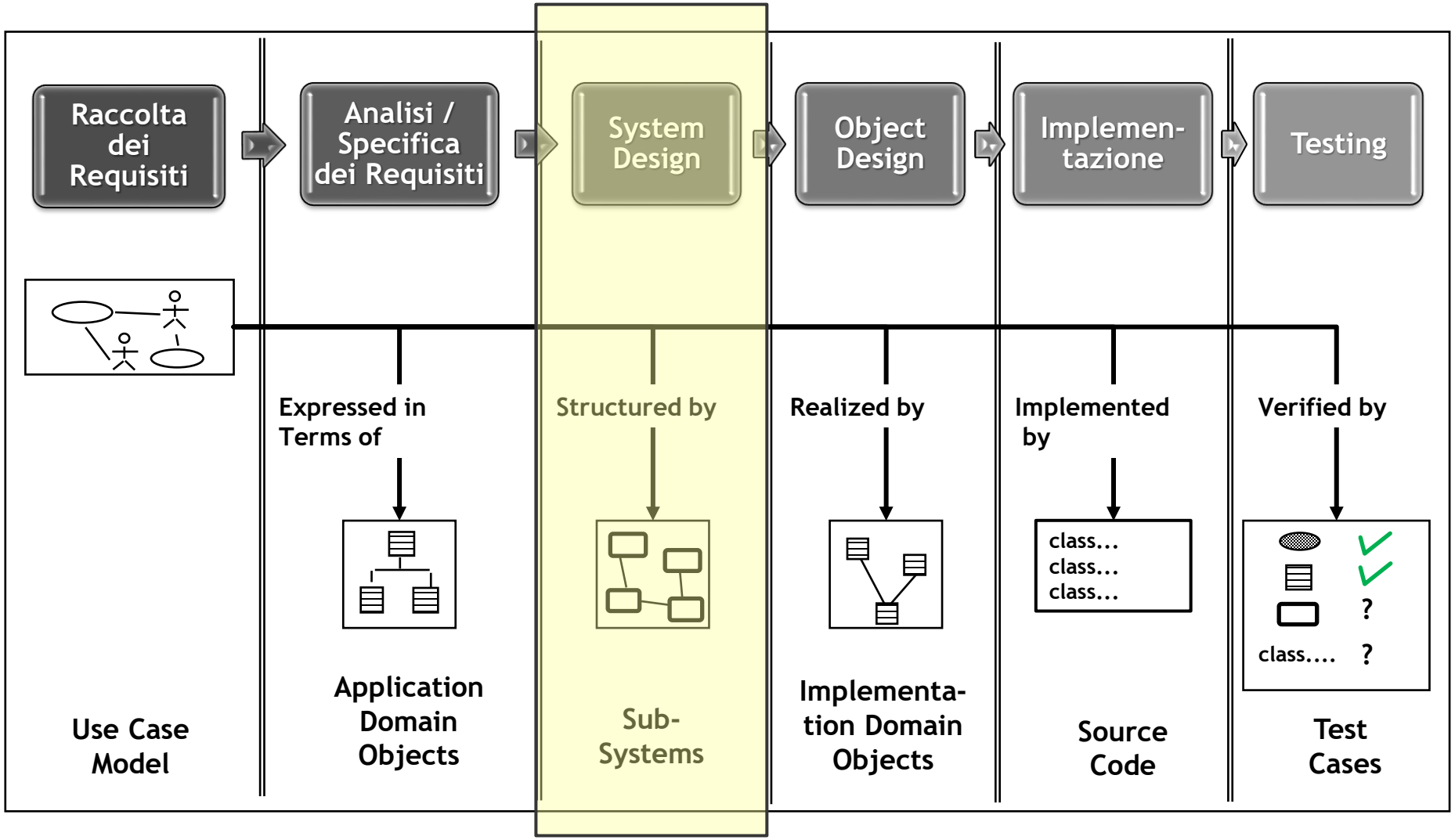
Ingegneria del Software – Architetture Software

Prof. Sergio Di Martino

Obiettivi della lezione

- Comprendere le principali Architetture Software
  - Concetto di Architettura
  - Tipi di Architetture
    - Repository Architecture
    - Client/Server Architecture
    - Peer-To-Peer Architecture
    - Model/View/Controller

Ciclo di Vita del Software



Organizzare sottosistemi in Architetture

Software Architecture

- Key part of Software Engineering

What Is Software Architecture?

- Software architecture is the structure or structures of the system, which comprise software elements, the externally visible properties of these elements, and the relationships among them.

- The exact structures to consider and the ways to represent them vary according to engineering goals.

Implications of this Definition – 1

- A software architecture is an abstraction of a system.
  - Architecture defines elements and how they interact.
  - Architecture suppresses purely local information about elements; private details are not architectural.
- Externally-visible properties of elements are assumptions that one elements can make about another:
  - provided services, required services, performance characteristics, fault handling, resource usage

Implications of this Definition – 2

- Every system has an architecture.
  - Every system is composed of elements and there are relationships among them.
  - In the simplest case, a system is composed of a single elements, related only to itself.
- Just having an architecture is different from having an architecture that is known to everyone:
  - The architecture versus specification of the architecture
- If you don't explicitly develop an architecture, you will get one anyway – and you might not like what you get.

Implications of this Definition – 3

- This means that box-and-line drawings alone are *not* architectures; but they are just a starting point.
  - You might *imagine* the behavior of a box labeled “database” or “executive” -- but that’s all
  - You need to add specifications and properties.

- Systems have many structures (views).
  - No single structure can be *the* architecture.
  - The set of candidate structures is not fixed or prescribed: choose whatever is useful for analysis, communication, or understanding.

Architetture

- L’architettura di un sistema software viene definita nella prima fase di System Design (progettazione architetturale)
- Lo scopo primario è la scomposizione del sistema in sottosistemi:
  - la realizzazione di più componenti distinti è meno complessa della realizzazione di un sistema come monolito.
  - Permette di parallelizzare lo sviluppo
  - Favorisce modificabilità, riusabilità, portabilità, etc…

Architetture

- Definire un’architettura significa mappare funzionalità su moduli
  - Es: Modulo di interfaccia utente, modulo di accesso a db, modulo di gestione della sicurezza, etc…
- Anche la definizione delle architetture deve seguire i concetti di Alta Coesione e Basso Accoppiamento
  - Ogni sottosezione dell’architettura dovrà fornire servizi altamente relati tra loro, cercando di limitare il numero di altri moduli con cui è legato

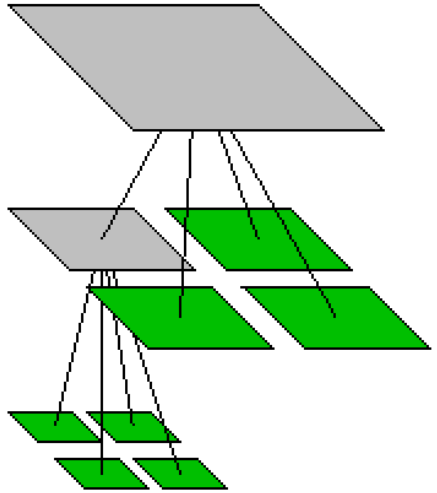
Definizione dell’architettura

- La definizione dell’architettura viene di solito fatta da due punti di vista diversi, che portano alla soluzione finale:
  - Identificazione e relazione dei sottosistemi
  - Definizione politiche di controllo
- Entrambe le scelte sono cruciali:
  - è difficile modificarle quando lo sviluppo è partito, poiché molte interfacce dei sottosistemi dovrebbero essere cambiate.

Identificazione sottosistemi

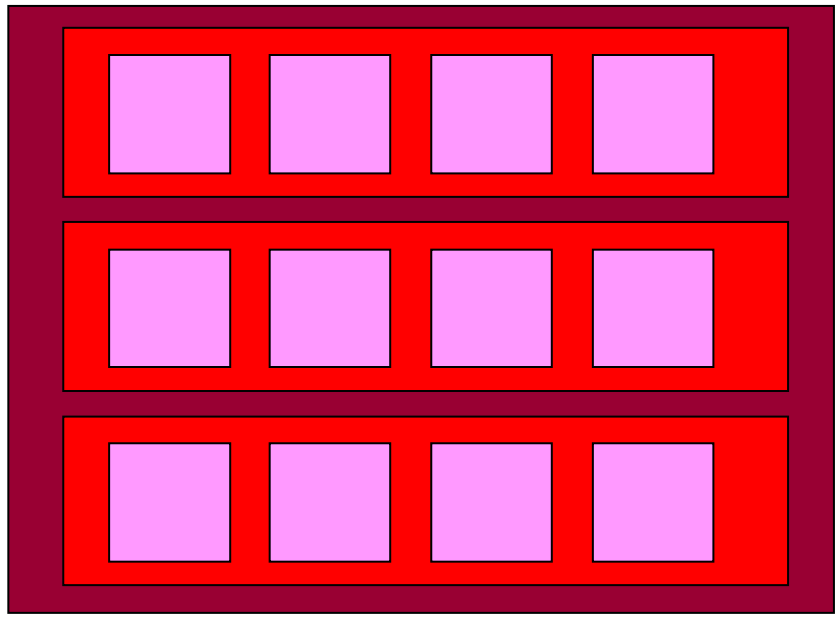
Design Principle: Divide and conquer

- Trying to deal with something big all at once is harder than dealing with a set of smaller things
  - Each individual component is smaller, and therefore easier to understand
  - Parts can be replaced or changed without having to replace or extensively change other parts.
  - Separate people can work on separate parts
  - An individual software engineer can specialize

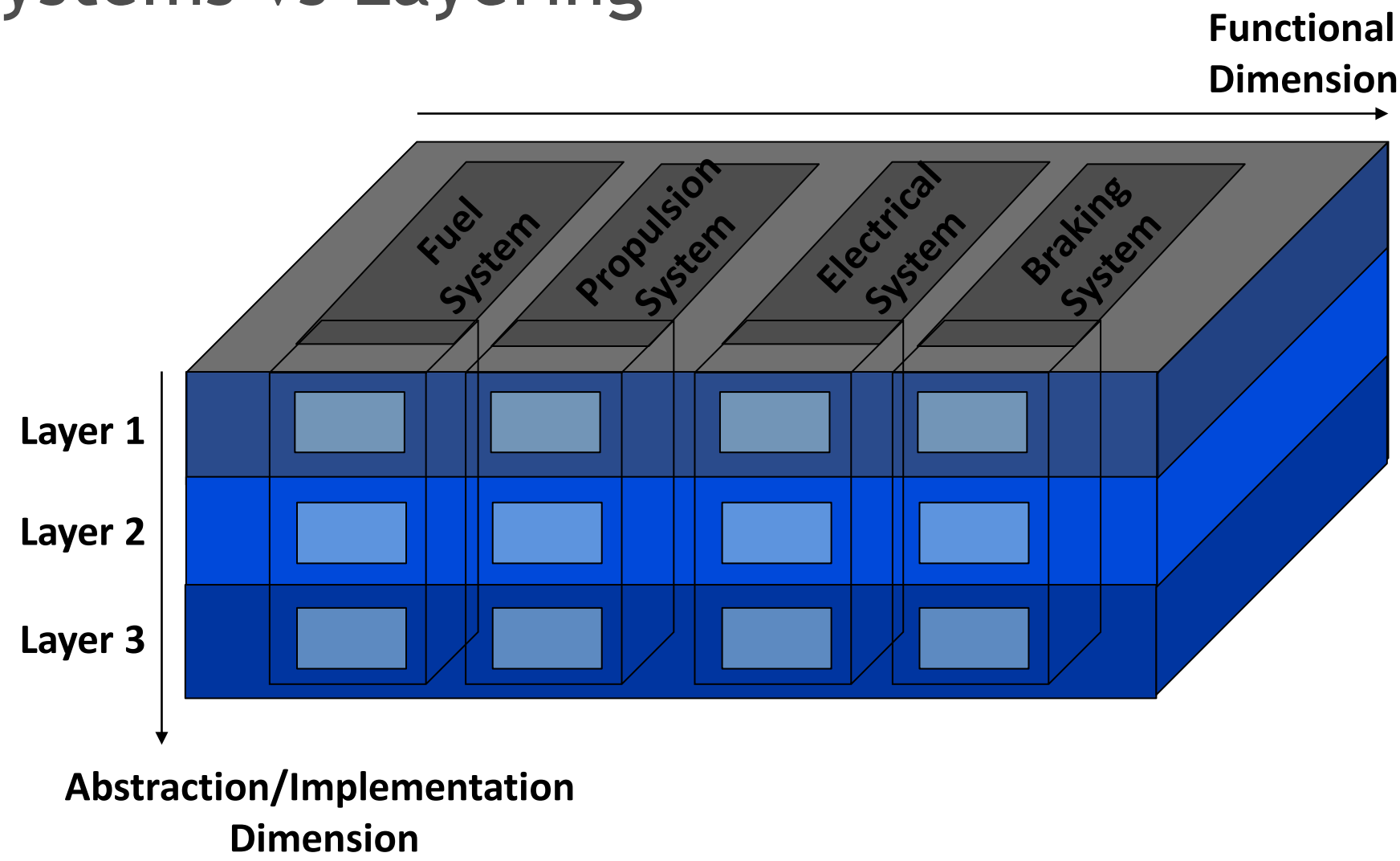


Ways of dividing a software system

- A system is divided up into
  - Layers & subsystems
  - A subsystem can be divided up into one or more packages
  - A package is divided up into classes
  - A class is divided up into methods



Subsystems vs Layering

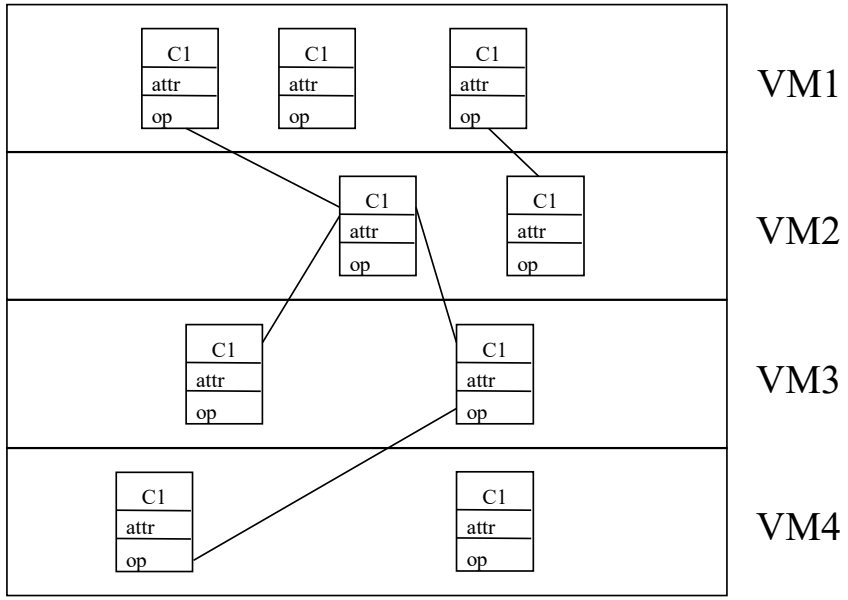


Layers

- Una decomposizione gerarchica di un sistema consiste di un insieme ordinato di layer (strati).
  - Un layer è un raggruppamento di sottosistemi che forniscono servizi correlati, eventualmente realizzati utilizzando servizi di altri layer.
  - Un layer può dipendere solo dai layer di livello più basso
  - Un layer non ha conoscenza dei layer dei livelli più alti
- **Architettura chiusa**: ogni layer può accedere solo al layer immediatamente sotto di esso
- **Architettura aperta**: un layer può anche accedere ai layer di livello più basso

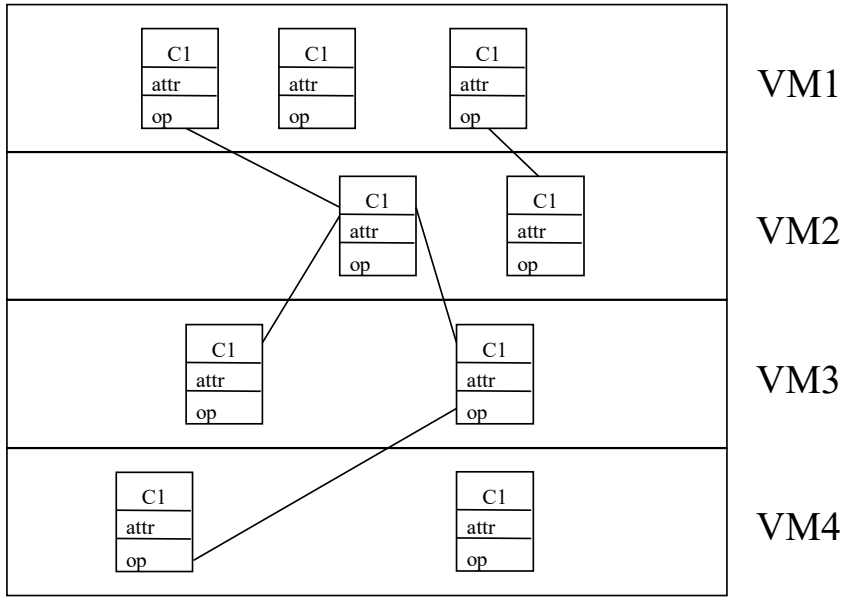
Macchina Virtuale (Dijkstra, 1965)

- Prima formalizzazione di architettura a layers
- Un sistema dovrebbe essere sviluppato da un insieme di macchine virtuali, ognuna costruita in termini di quelle al di sotto di essa.

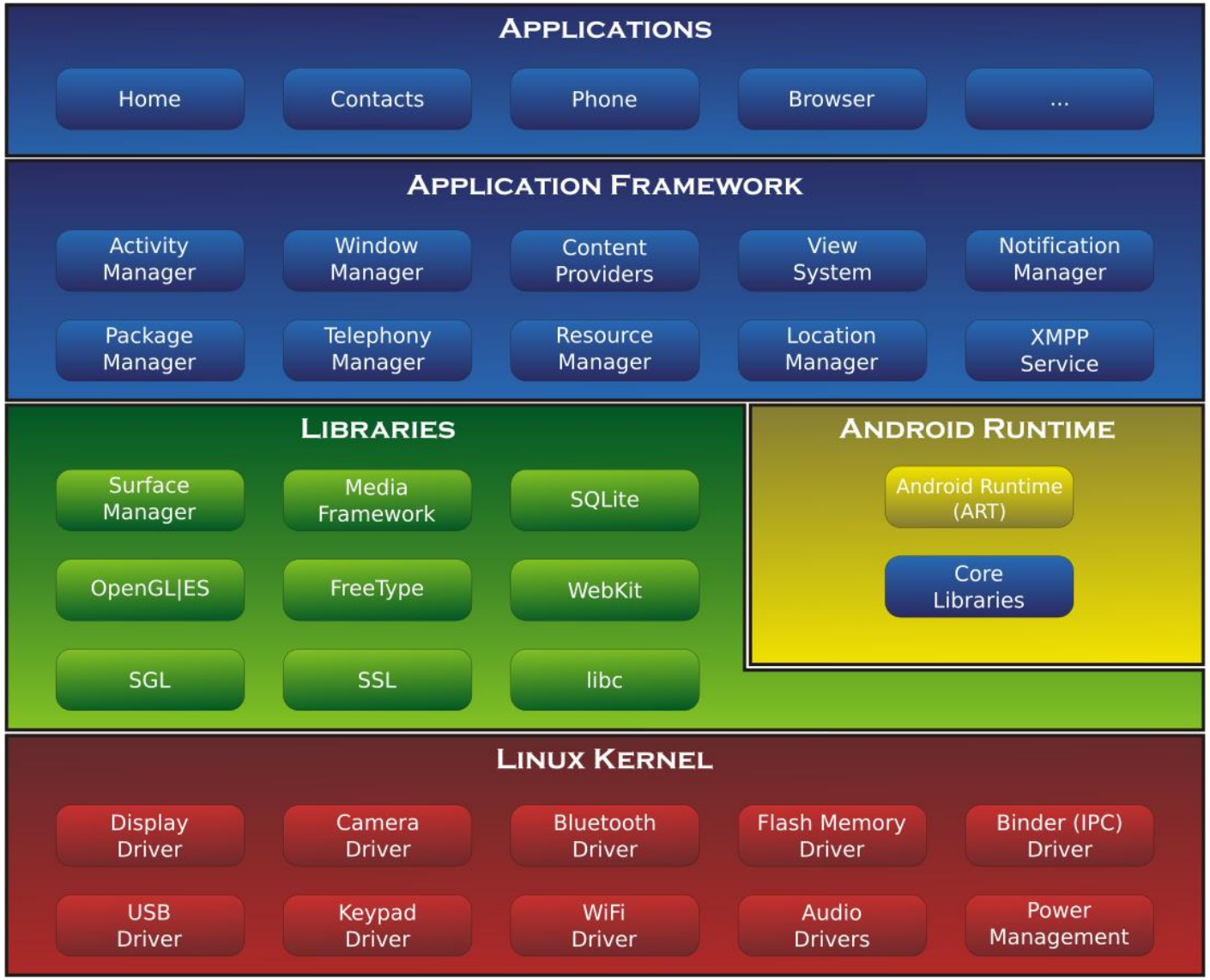


Architettura Chiusa

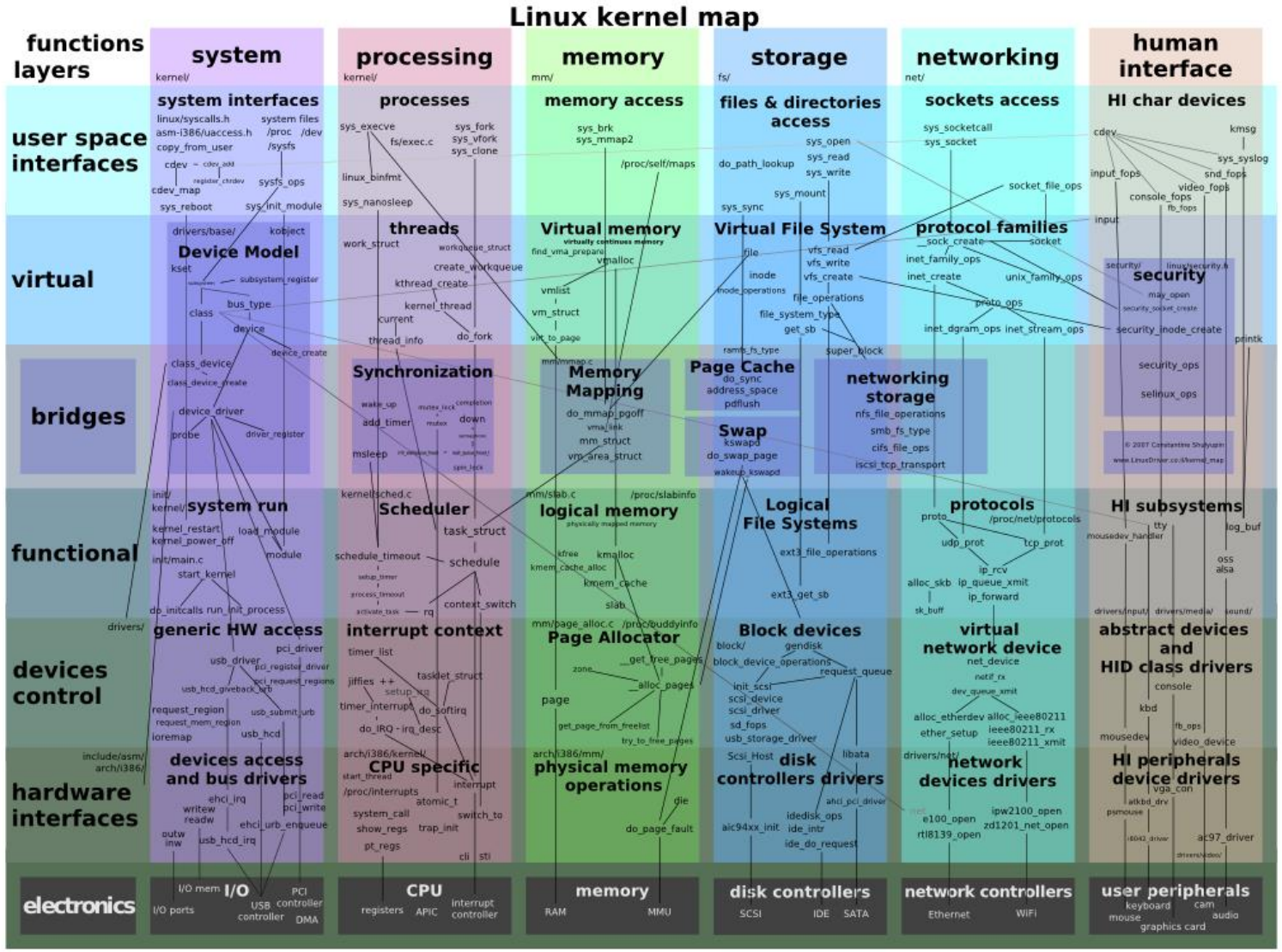
- Una macchina virtuale può solo chiamare le operazioni dello strato sottostante
- Design goal: alta manutenibilità e portabilità



Esempio di Architettura chiusa: Android

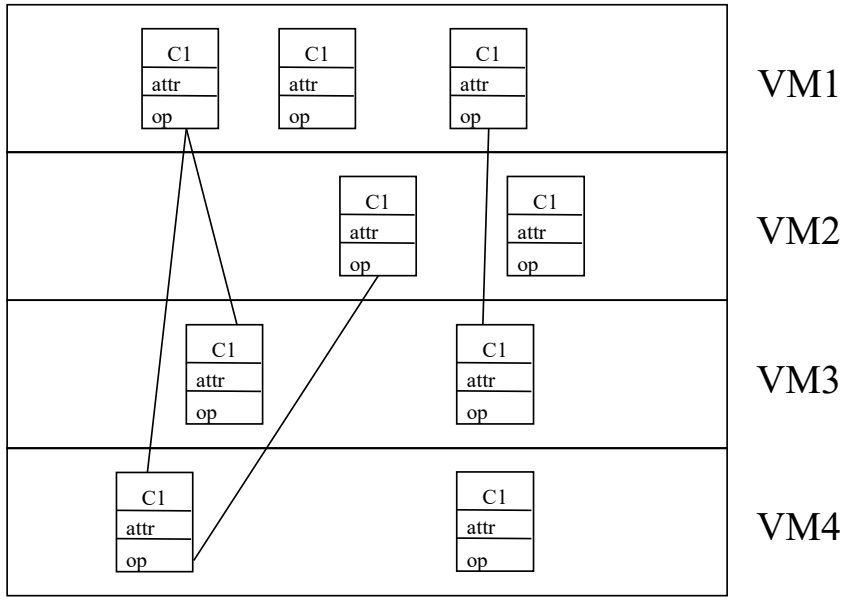


Consistency in two dimensions

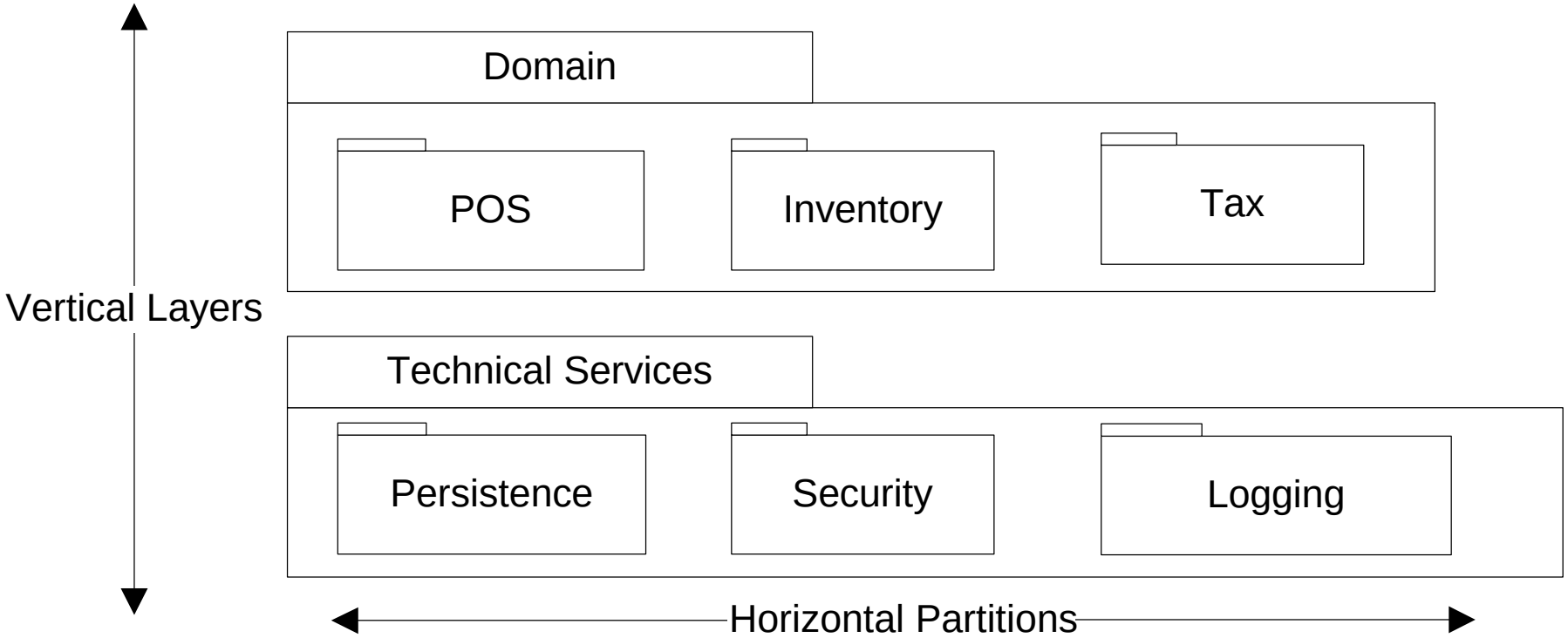


Architettura Aperta

- Una macchina virtuale può utilizzare i servizi delle macchine dei layer sottostanti
- Design goal: efficienza a runtime



Architettura Logica: Layers e Partizioni



Principali Architetture

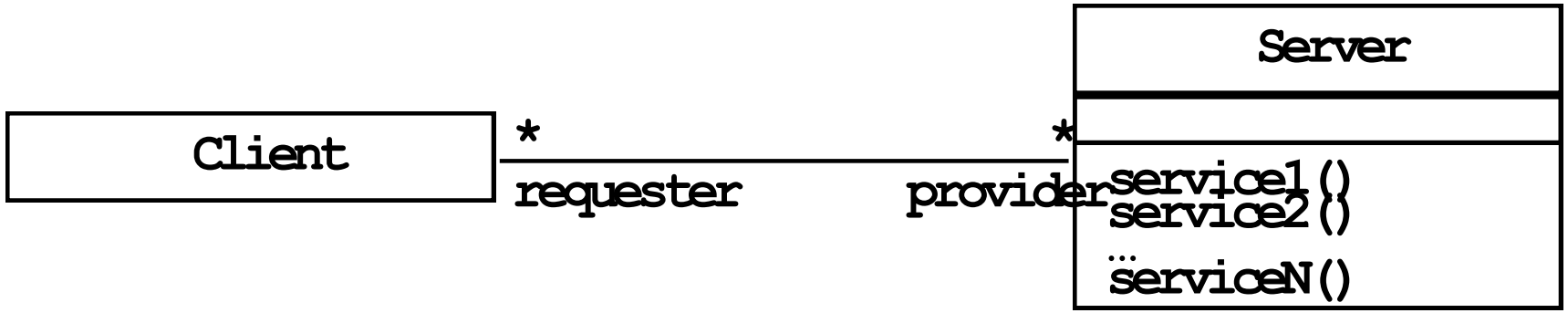
Architettura Client-server

- E’ una architettura distribuita dove dati ed elaborazione sono distribuiti su una rete di nodi di due tipi:
  - I server sono processori potenti e dedicati: offrono servizi specifici come stampa, gestione di file system, compilazione, gestione traffico di rete, calcolo.
  - I client sono macchine meno prestazionali sulle quali girano le applicazioni-utente, che utilizzano i servizi dei server.

- I Client devono conoscere i nomi e la natura dei Server;
- I Server non devono conoscere identità e numero dei Clienti.

Client/Server Architecture

- Un sottosistema, detto server, fornisce servizi ad istanze di altri sottosistemi detti client che sono responsabili dell'interazione con l'utente.
- I Client chiamano il server che realizza alcuni servizi e restituisce il risultato.
  - I Client conoscono l'interfaccia del Server (i suoi servizi)
  - I Server non conoscono le interfacce dei Client
  - La risposta è in generale immediata
- Gli utenti interagiscono solo con il Client

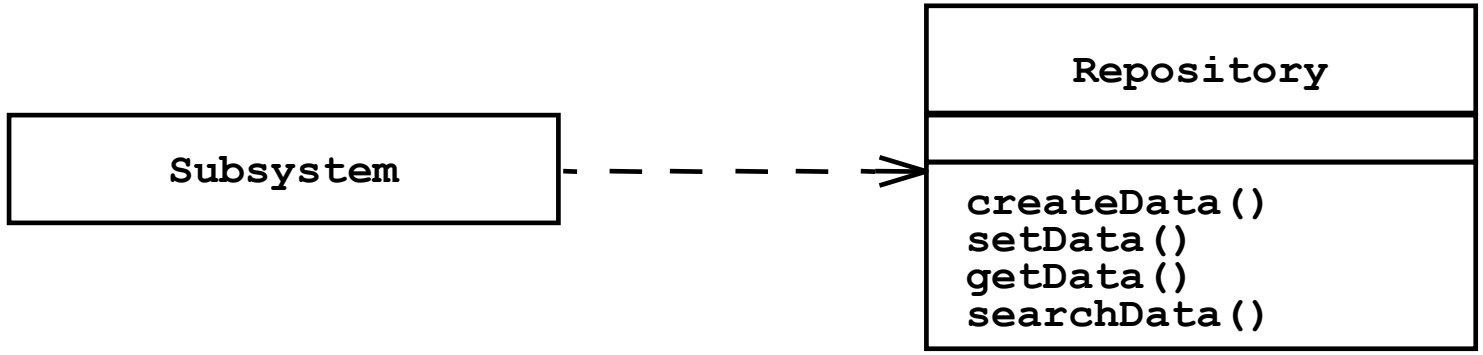


Principali Architetture Software

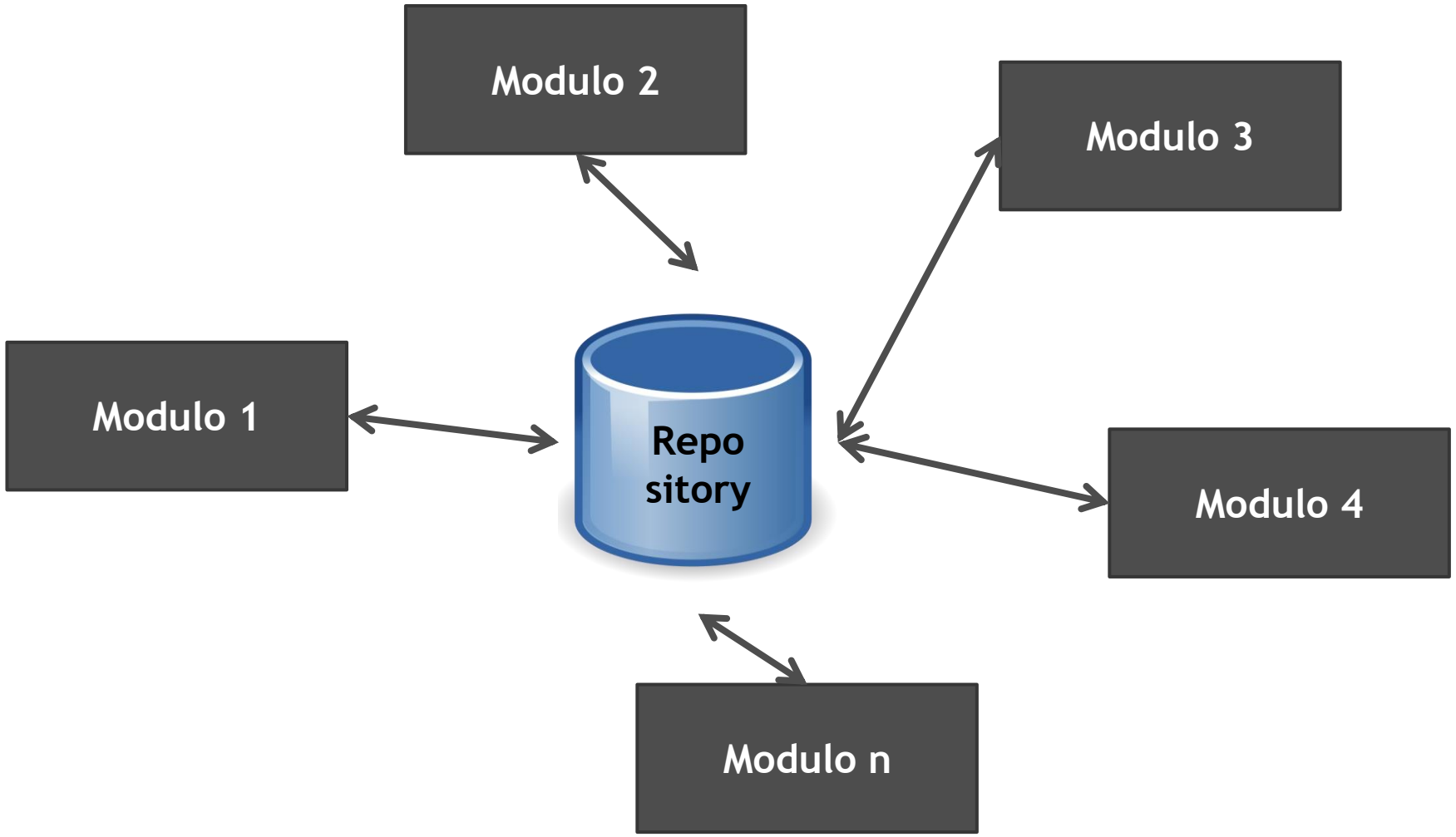
- Nell'ingegneria del sw sono stati definiti vari stili architetturali che possono essere usati come base per uno specifico sistema software da sviluppare:
  - Client/Server Architecture
  - Repository Architecture
  - Peer-To-Peer Architecture
  - Model/View/Controller

Repository Architecture

- I sottosistemi accedono e modificano una singola struttura dati chiamata repository.
- I sottosistemi sono “relativamente indipendenti” (interagiscono solo attraverso il repository)
- Il flusso di controllo è dettato o dal repository (un cambiamento nei dati memorizzati) o dai sottosistemi (flusso di controllo indipendente)

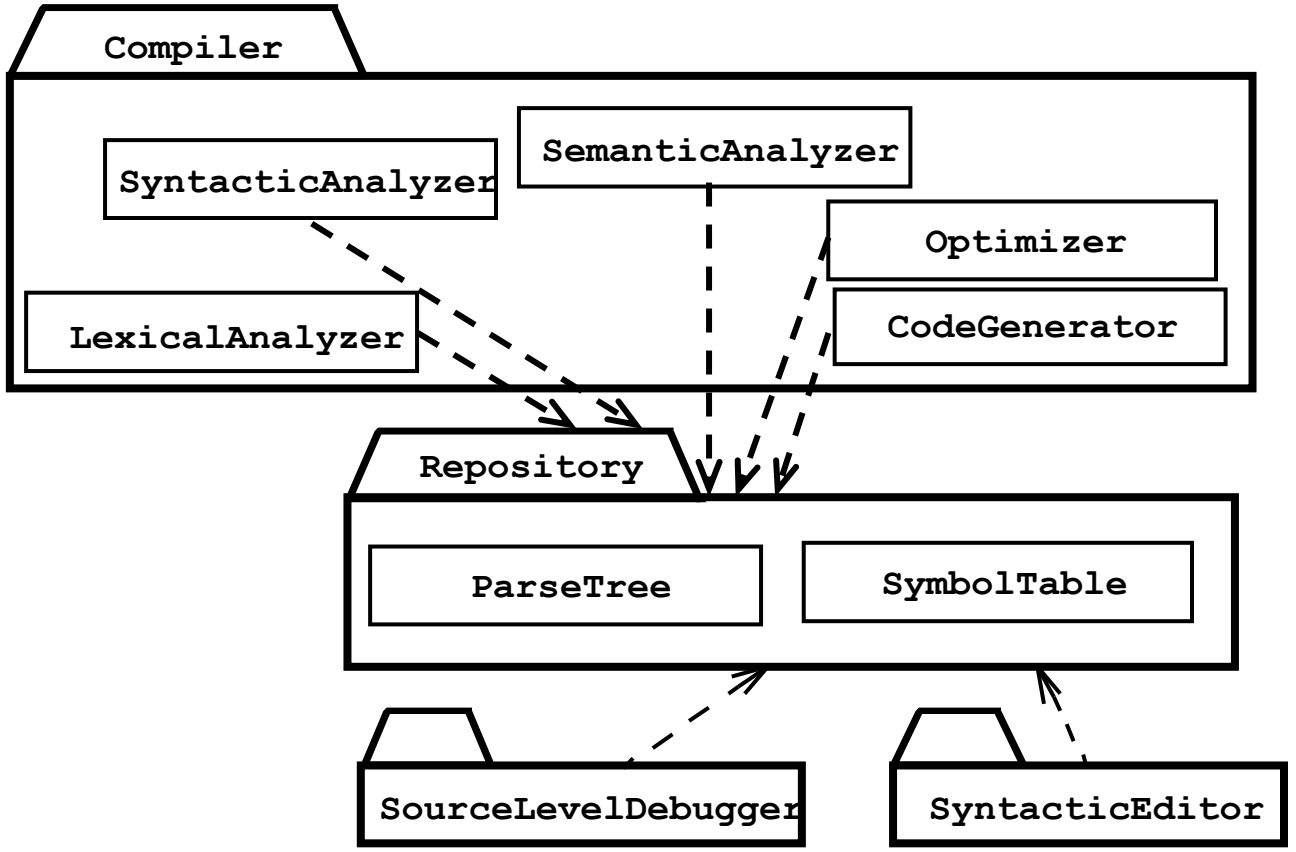


Repository Architecture



Esempio di Repository Architecture

- Database Management Systems
- Modern Compilers



Vantaggi dell’architettura a repository

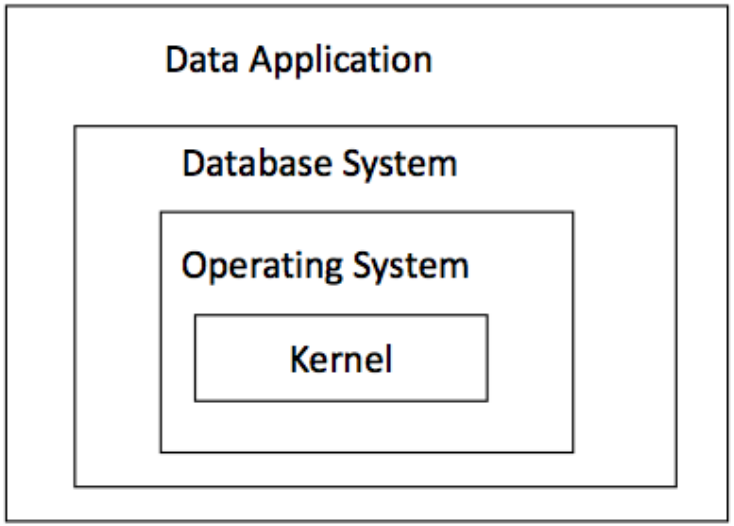
- Modo efficiente di condividere grandi moli di dati: write once for all to read
- Un sottosistema non si deve preoccupare di come i dati sono prodotti/usati da ogni altro sottosistema
- Gestione centralizzata di backup, security, access control, recovery da errori...
- Il modello di condivisione dati è pubblicato come repository schema → facile aggiungere nuovi sottosistemi

Svantaggi dell’architettura a repository

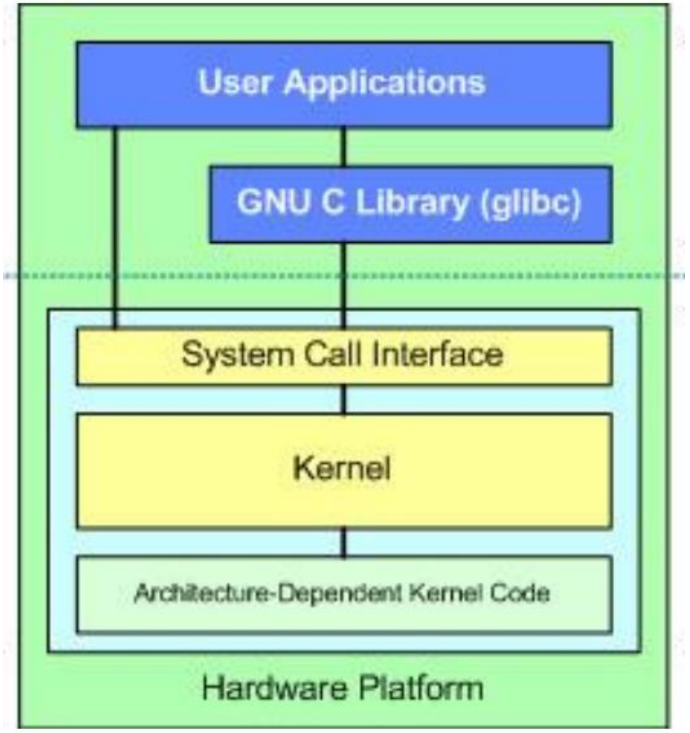
- I sottosistemi devono concordare su un modello dati di compromesso → minori performance
- Data evolution: la adozione di un nuovo modello dati è difficile e costosa:
  - esso deve venir applicato a tutto il repository,
  - tutti i sottosistemi devono essere aggiornati
- Diversi sottosistemi possono avere diversi requisiti su backup, security… non supportati
- E’ difficile distribuire efficientemente il repository su piu’ macchine (continuando a vederlo come logicamente centralizzato): problemi di ridondanza e consistenza dati.

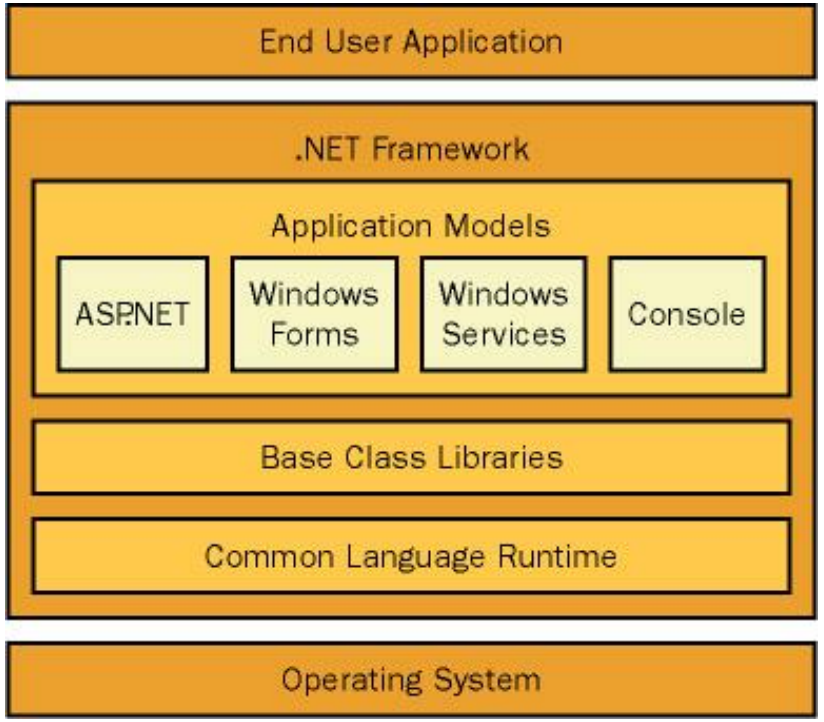
Layered Architecture

- Sub-systems are organized into layers
- Each layer:
  - uses the services of the layer below
  - provides services to layer above through well-defined interfaces
- The terms “tier” and “layer” are interchangeable, for most people



Layered Architecture Examples

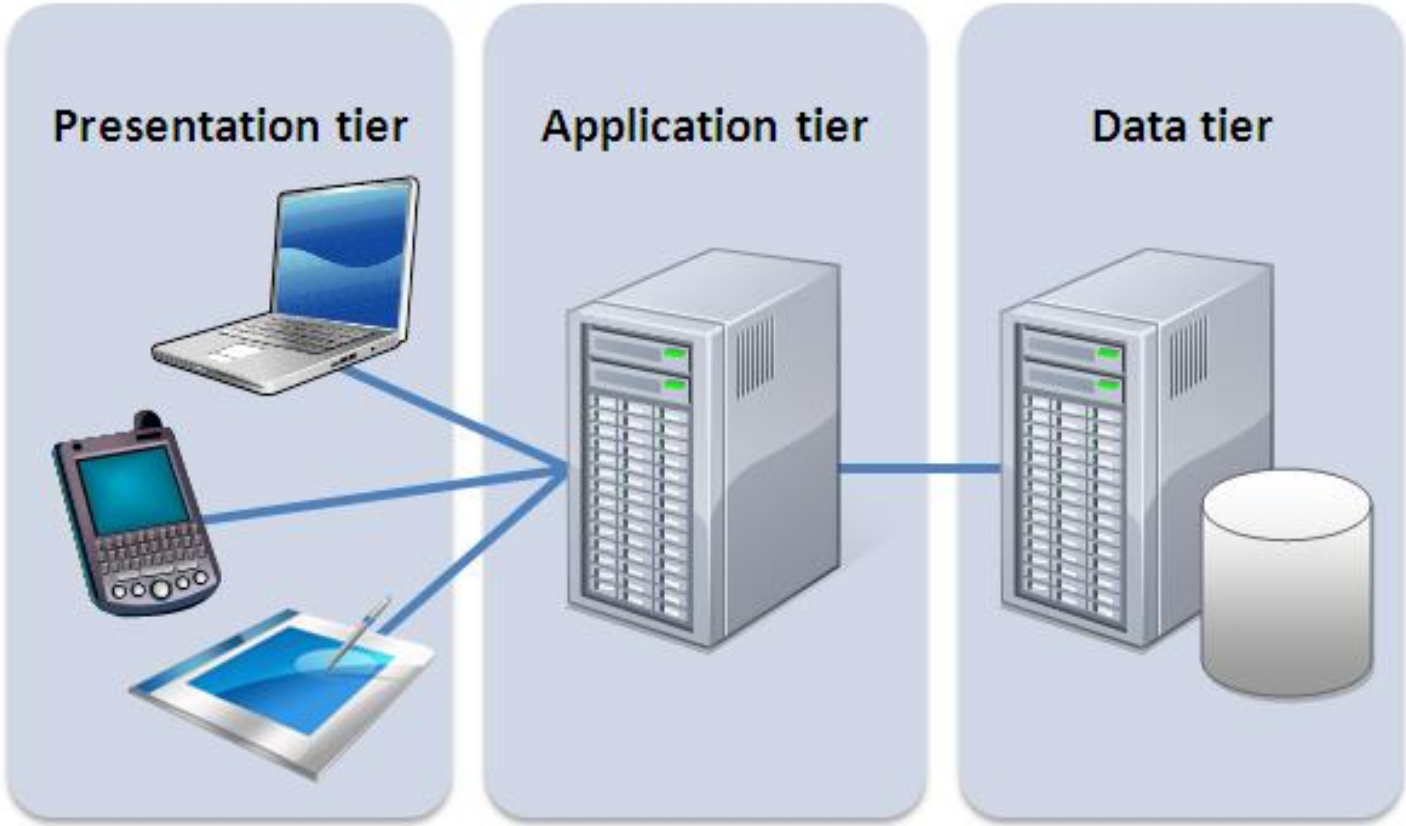




Layered Architecture

- + abstraction of concerns
- + isolation of lower and higher levels from one another (i.e. decoupled)
- + reusability
- - overhead of going through different layers
- - complexity

Architetture a 3 livelli



Architetture three-tier - I

- Introdotte negli anni 90
- Business logic trattata in modo esplicito:
  - livello 1: gestione dei dati (comunicazione verso DBMS, file XML, …..)
  - livello 2: business logic (processamento dati, …)
  - livello 3: interfaccia utente (presentazione dati, servizi)
- Ogni livello ha obiettivi e vincoli di design propri
- Nessun livello fa assunzioni sulla struttura o implementazione degli altri:
  - livello 2 non fa assunzioni su rappresentazione dei dati, né sull'implementazione dell'interfaccia utente
  - livello 3 non fa assunzioni su come opera la business logic..

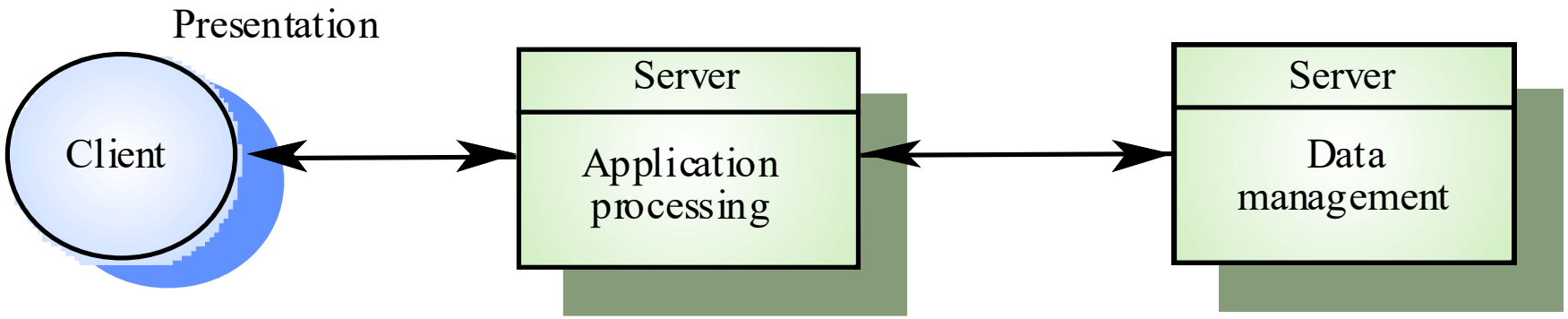
Architetture three-tier - II

- Non c’è comunicazione diretta tra livello 1 e livello 3
  - Interfaccia utente non riceve, né inserisce direttamente dati nel livello di data management
  - Tutti i passaggi di informazione nei due sensi vengono filtrati dalla business logic
- I livelli operano senza assumere di essere parte di una specifica applicazione
  - $\Rightarrow$ applicazioni viste come collezioni di componenti cooperanti
  - $\Rightarrow$ ogni componente può essere contemporaneamente parte di applicazioni diverse (e.g., database, o componente logica di configurazione di oggetti complessi)

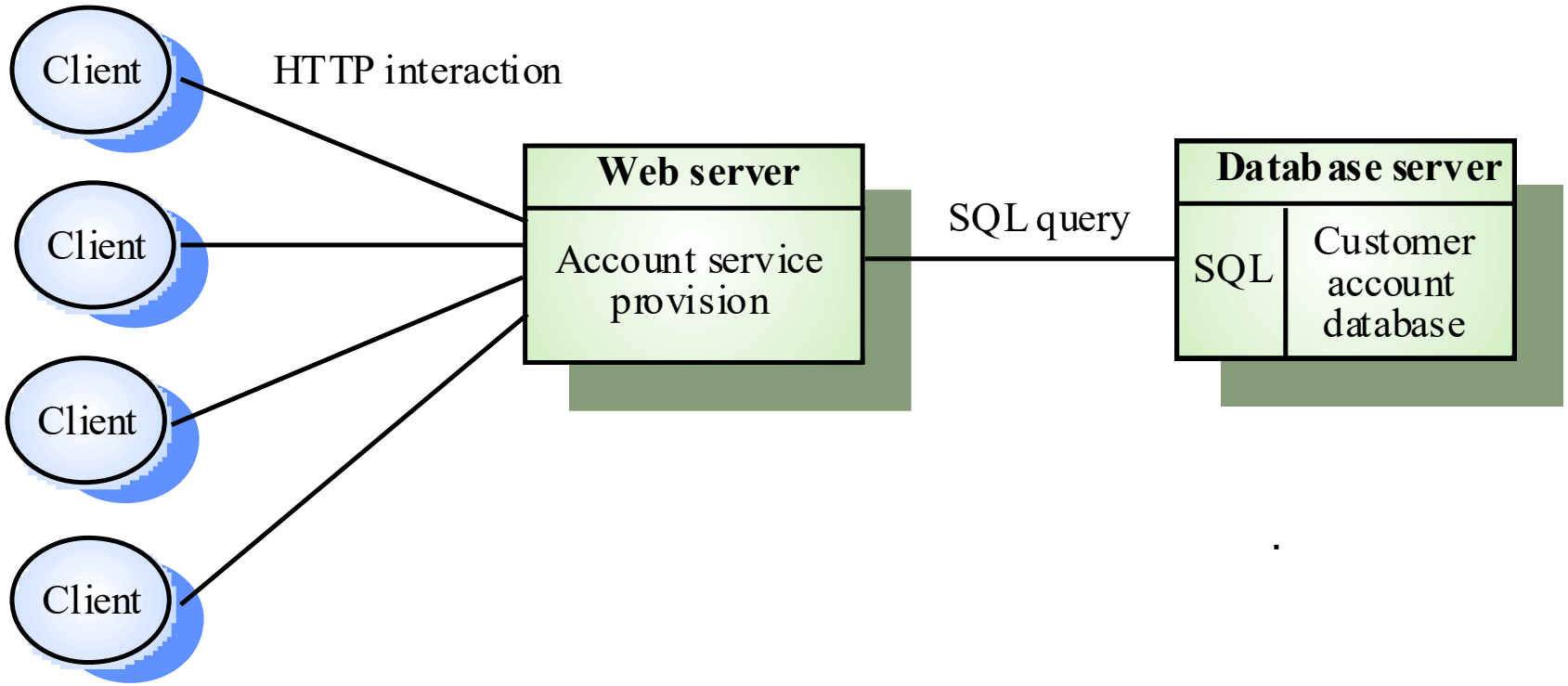
Vantaggi di architetture three-tier

- Flessibilità e modificabilità di sistemi formati da componenti separate:
  - componenti utilizzabili in sistemi diversi
  - modifica di una componente non impatta sul resto del sistema (a meno di cambiamenti nelle API)
  - ricerca di bug più focalizzata (separazione ed isolamento delle funzionalità del sistema)
  - aggiunta di funzionalità all'applicazione implica estensione delle sole componenti coinvolte (o aggiunta di nuove componenti)

A 3-Tier Client-Server Architecture



An Internet Banking System



Vantaggi di architetture three-tier

- Interconnettività
  - API delle componenti superano il problema degli adattatori del modello client server → N interfacce diverse possono essere connesse allo stesso servizio etc.
  - Facilitato l'accesso a dati comuni da parte di applicazioni diverse (uso dello stesso gestore dei dati da parte di business logics diverse)
- Gestione di sistemi distribuiti
  - Business logic di applicazioni distribuite (e.g., sistemi informativi con alcuni server replicati e client remoti) aggiornabile senza richiedere aggiornamento dei client

Svantaggi di architetture three-tier

- Dimensioni delle applicazioni ed efficienza
  - Pesante uso della comunicazione in rete $\Rightarrow$ latenza del servizio
  - Comunicazione tra componenti richiede uso di librerie SW per scambio di informazioni $\Rightarrow$ codice voluminoso
- Problemi ad integrare Legacy software
  - Molte imprese usano software vecchio (basato su modello monolitico) per gestire i propri dati $\Rightarrow$
    - difficile applicare il modello three-tier per nuove applicazioni
    - introduzione di adapters per interfacciare il legacy SW

Architetture n-Tier

- Evoluzione delle 3-tier, su N livelli
- Permettono configurazioni diverse.
- Elementi fondamentali:
- Interfaccia utente (UI)
  - gestisce interazione con utente
  - può essere un web browser, Mobile App, interfaccia grafica, …
- Presentation logic
  - definisce cosa UI presenta e come gestire le richieste utente
- Business logic
  - gestisce regole di business dell'applicazione
- Infrastructure services
  - forniscono funzionalità supplementari alle componenti dell'applicazione (messaging, supporto alle transazioni, …)
- Data layer:
  - livello dei dati dell'applicazione