DIETI UNINA UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II
DIPARTIMENTO DI INGEGNERIA ELETTRICA E DELLE TECNOLOGIE DELL'INFORMAZIONE

Corso di Laurea in Ingegneria Informatica

Corso di Ingegneria del Software

Pattern architetturali

Introduzione

"*A **pattern** describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice*".
[Christopher Alexander, Sara Ishikawa, Murray Silverstein, Max Jacobson, Ingrid Fiksdahl-King, Shlomo Angel. A Pattern Language. Oxford University Press, New York, 1977.]

☞I Pattern sono soluzioni a problemi ricorrenti che si incontrano in applicazioni reali.

☞I Pattern spiegano come realizzare oggetti e le interazioni tra essi, costruendo codice riusabile.

Christopher Alexander – 1/2

- L'architetto C. Alexander introdusse il concetto di pattern nel suo libro "The Timeless Way of Building".
- La sua nozione di pattern come best practice per problemi di progettazione comuni e ricorrenti è stata adottata dalla comunità software.
- Alexander introdusse il concetto di linguaggio di pattern, come un insieme di schemi che complessivamente forniscono un vocabolario base per la progettazione all'interno di uno specifico contesto o dominio applicativo.

Christopher Alexander – 2/2

- In “A Pattern Language”, Alexander introdusse uno specifico linguaggio di pattern per il dominio architettonico, con ad esempio sottolinguaggi per la progettazione urbanistica delle città (rivolto a chi fa pianificazione urbanistica) e edile (rivolto agli architetti).
- Esempi di pattern definiti da Alexander sono:
  - Town patterns: Ring roads (circonvallazioni), night life, and row houses (case a schiera);
  - Building patterns: Roof garden, indoor sunlight, …

Tipi di Pattern

- I pattern sono applicabili in ogni fase del ciclo di vita di un software: esistono
  - pattern di analisi;
  - pattern architetturali;
  - pattern di progettazione (design);
  - pattern di codifica.

Vantaggi dei pattern

- Riuso della conoscenza/esperienza di progettazione
  - Raramente i problemi sono nuovi e unici
  - I pattern forniscono indicazioni su “dove cercare soluzioni ai problemi”
- Stabiliscono una terminologia comune e condivisa
  - Es.: è facile dire “Qui ci serve un Façade”
- Forniscono una prospettiva di alto livello
  - Liberano dal dover gestire troppo presto i dettagli della progettazione.
- I pattern sono un “riferimento progettuale” (*design reference*)

Archetipi architetturali o pattern architetturali

- Il progetto di architetture software può essere guidato da altre architetture
- Una architettura astratta riusabile è chiamata “archetipo architetturale” o “stile architetturale”.
- Su una scala più piccola, le “micro-architetture” riusabili sono dette “pattern di progettazione” (*design patterns*).

Pattern di progettazione

- C. Alexander definisce un pattern come una tripla che esprime una relazione tra un problema, una soluzione, e un contesto:
  Pattern = coppia (problema, soluzione) in un contesto

- Nel campo dell'ingegneria del software, un **pattern di progettazione (*design pattern*)** è una soluzione comprovata a un problema tipico che sorge nello sviluppo software in uno specifico contesto.

Pattern architetturali

Architettura Software

- L'architettura software è l'organizzazione di base di un sistema, espressa dai suoi componenti, dalle relazioni tra di loro e con l'ambiente, e i principi che ne guidano il progetto e l'evoluzione.[IEEE/ANSI 1471-2000]
- Definire un'architettura significa mappare su componenti sia i requisiti funzionali e sia i requisiti non funzionali
  - Es: Interfaccia utente, Accesso a db, Gestione della sicurezza, etc…
- Ogni componente dell'architettura dovrà fornire servizi altamente relati tra loro, cercando di limitare il numero di altri componenti con cui è legato (Alta Coesione e Basso Accoppiamento)

Layers e Partitions

- Due visioni ortogonali tra loro per decomporre in sottosistemi:
  - Un **Layer** è uno strato che fornisce servizi (Es: interfaccia utente, accesso a DB)
  - Una **Partition (tier)** è un'organizzazione di moduli *peer*, spesso all'interno di un *layer*

Layered Architecture

- 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

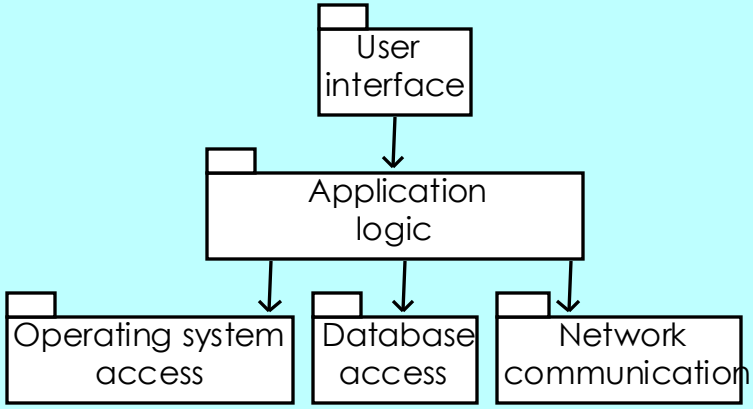
- *Esempi di servizi: servizi necessari per elaborazioni, memorizzazione di dati, per gestire la sicurezza, per interagire con gli utenti, accedere al sistema operativo, interagire con l'hardware*
  - L'insieme di procedure o metodi attraverso i quali un livello fornisce i suoi servizi costituisce la ***application programming interface (API)***

Layered Architecture

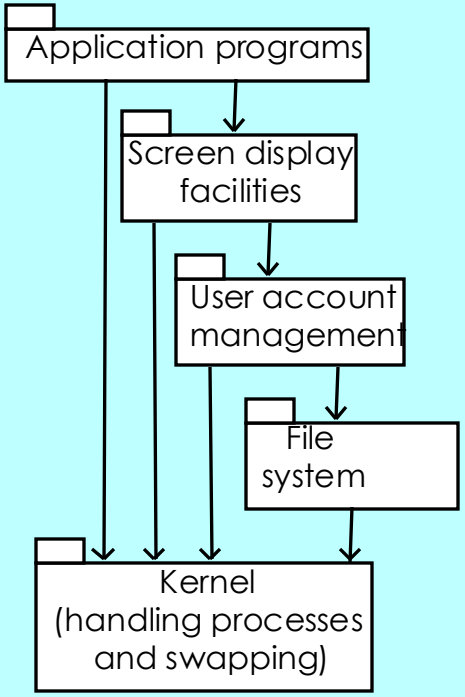
- **Architettura chiusa**: ogni layer può accedere solo al layer immediatamente sotto di esso
  - Es.: Prima formalizzazione di sistema come insieme di macchine virtuali (Dijkstra, 1965). Una MV può solo chiamare le operazioni dello strato sottostante
  - Es.: Pila ISO OSI

- **Architettura aperta**: un layer può anche accedere ai layer di livello più basso
  - Es.: Sistema come insieme di macchine virtuali che possono accedere a tutti i layer sottostanti

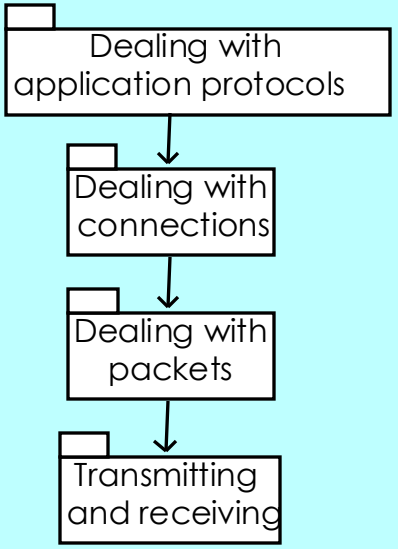
Esempi di Architetture a Layers



a) Typical layers in an application program



b) Typical layers in an operating system



c) Simplified view of layers in a communication system

Partition (Tier)

- Suddividere (partitioning) il sistema in sottosistemi paritari (peer) fra loro, ognuno responsabile di differenti classi di servizi.
  - *Ad es.: servizi di account, sales e inventory nel layer applicativo; servizi di persistenza, sicurezza e comunicazione nel layer sottostante*

Pattern architetturali

- Esempi di *architectural patterns* illustrati in dettaglio nel seguito:
  - MVC
  - BCE
  - Broker
  - Altri patterns (cenni)
    - Pipe & Filter
    - Shared data-Repository
    - Client/Server
    - Two e Three-tiers
    - SOA

Pattern architetturali: MVC

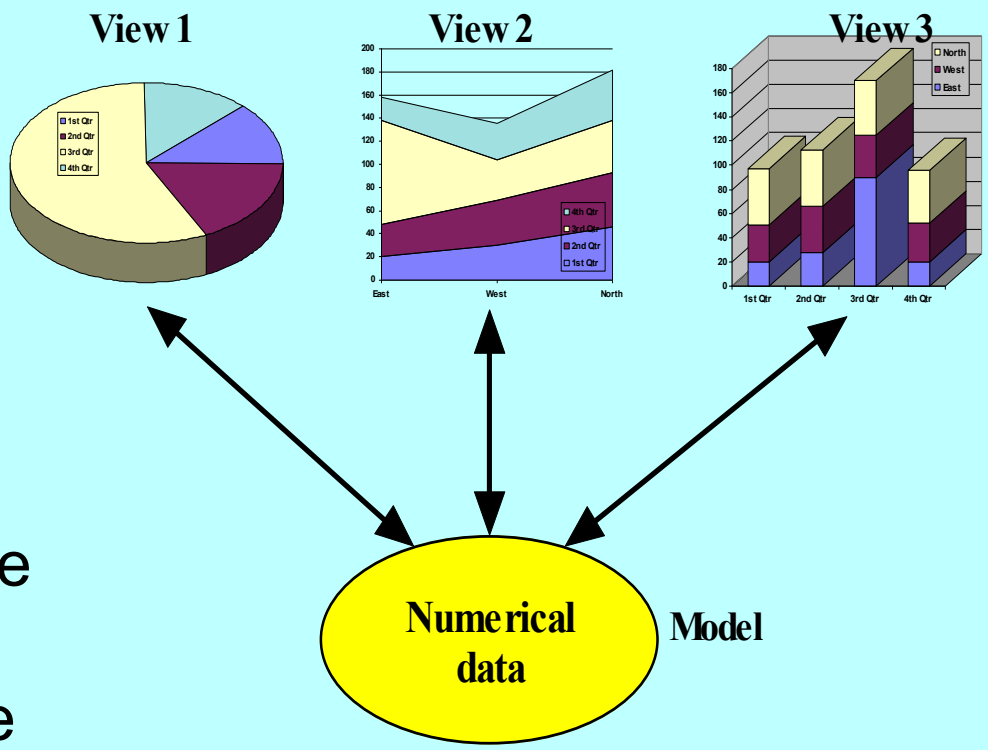
Il Pattern MVC – 1/4

- Un classico esempio di **pattern architetturale** è il pattern Model-View-Controller (MVC)
- È stato uno dei primi pattern, introdotto nel 1988 con Smalltalk-80
- In realtà è una vera e propria “composizione di pattern”
- La sua fama odierna è dovuta in parte alla diffusione di Java, perché utilizzato per alcuni componenti della libreria Swing
- Alcuni ambienti di sviluppo (Microsoft Visual C++, Jakarta Struts) si basano su tale pattern.

Il Pattern MVC – 2/4

- Nacque dalla necessità di visualizzare dati generici tramite interfaccia grafica (GUI), mediante l'utilizzo di rappresentazioni diverse dei dati stessi.

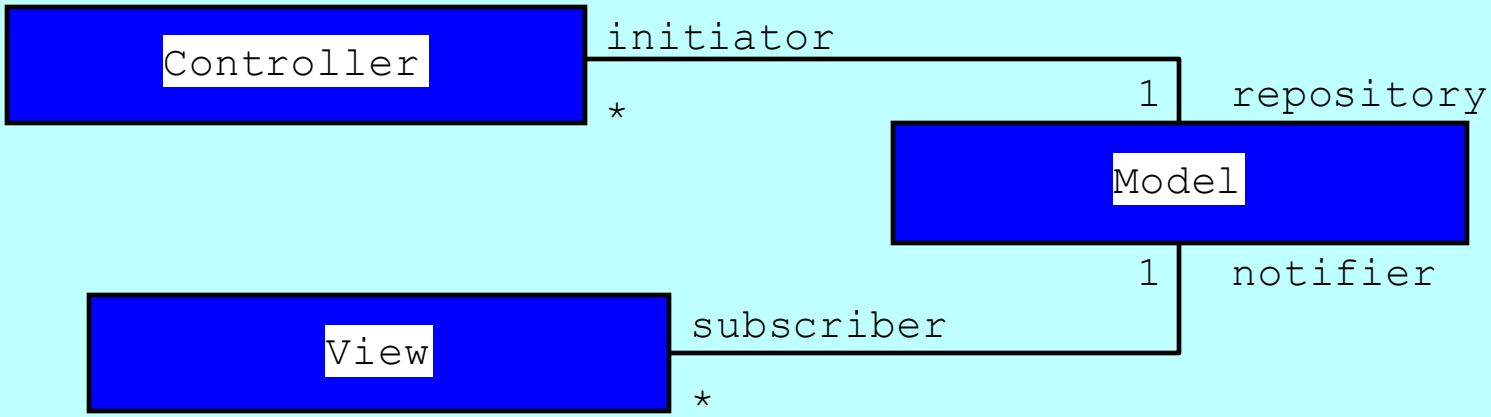
- Consente la realizzazione di una architettura che permette la separazione netta dei componenti di presentazione dei dati dai componenti che gestiscono i dati stessi.



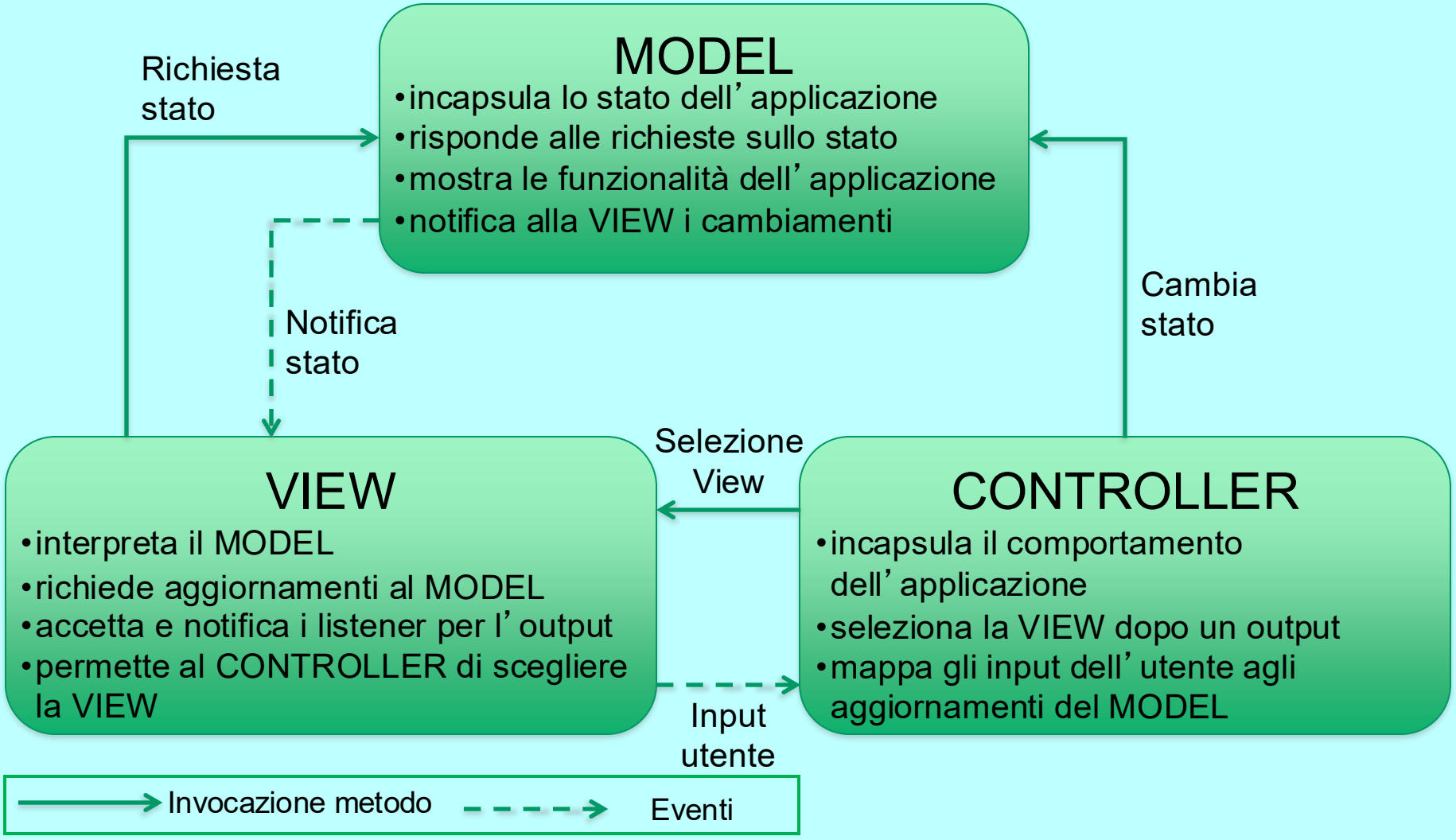
Il Pattern MVC – 3/4

- MVC disaccoppia **View** e **Model**, utilizzando un protocollo di sottoscrizione-notifica.

- **Controller** trasforma gli input utente nella **View** in azioni del **Model**, implementando la logica di controllo dell'applicazione.



Il Pattern MVC – 4/4



MVC - Modello

- Definisce le regole di business per l'interazione con i dati, esponendo alla **View** ed al **Controller** rispettivamente le funzionalità per l'accesso e l'aggiornamento.
- Ha la responsabilità di notificare ai componenti della **View** eventuali aggiornamenti verificatisi in seguito a richieste del **Controller**, al fine di permettere alle **View** di presentare dati sempre aggiornati.

MVC - Vista

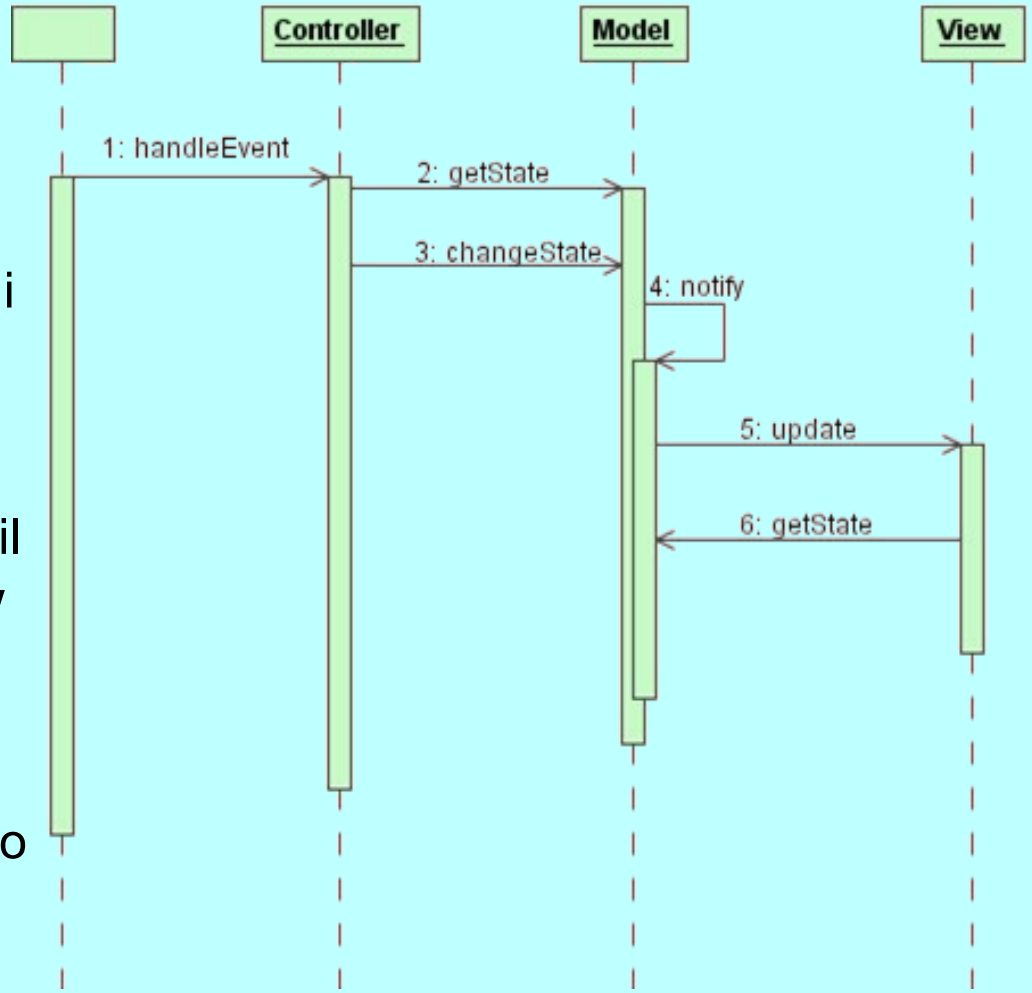
- Gestisce la logica di presentazione dei dati.
- Ogni GUI può implementare viste diverse sui dati, e modalità diverse di interazione coi dati
  - Es.: Modalità asincrona e strategia "push".
    - Si può utilizzare un altro pattern (Observer), registrando le **View** come osservatori del **Model**. Le **View** possono quindi richiedere gli aggiornamenti al **Model** in tempo reale.
    - Inoltre la **View** delega al **Controller** l'esecuzione dei processi richiesti dall'utente dopo averne catturato gli input, e la scelta delle eventuali schermate da presentare.

Controller

- Ha la responsabilità di trasformare le interazioni dell'utente della **View** in azioni eseguite dal Model.
- Realizza la corrispondenza tra l'input dell'utente e i processi eseguiti dal **Model**.
- Selezionando la schermate della **View** richieste, il **Controller** implementa la logica di controllo dell'applicazione.

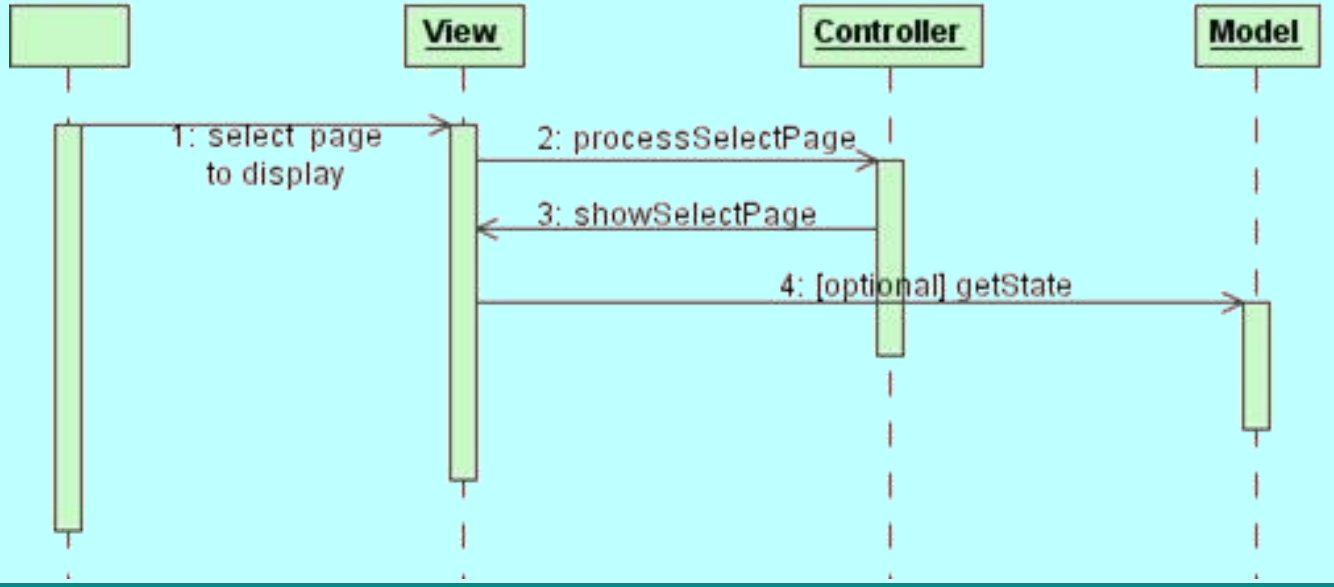
Aggiornamento dei dati

- In risposta a qualche evento il metodo `handleEvent()` viene invocato sul **Controller**;
- Il **Controller** esamina lo stato del modello usando il metodo `getState()` e invoca un metodo di servizio nel modello per cambiare il suo stato;
- Il modello cambia lo stato e invoca il suo metodo di notifica per notificare il cambiamento di stato a tutte le **View** registrate;
- Ogni **View** registrata viene notificata invocando il suo metodo `update()`. La **View** esamina lo stato del modello tramite il metodo `getState()` e si autoaggiorna.



Selezione Schermata

- L'utente seleziona una pagina.
- La **View** non decide quale sarà la schermata richiesta, delegando ciò al **Controller** tramite l'invocazione del suo metodo `processSelectPage()`, che grazie al pattern Strategy può essere anche cambiato dinamicamente a runtime.
- Il **Controller** decide la pagina da visualizzare, e la comunica alla **View** invocando il suo metodo `showSelectPage()`.
- La **View** costruisce e presenta all'utente della schermata stessa, controllando se ci sono state modifiche nel modello invocando il metodo `getState()`.

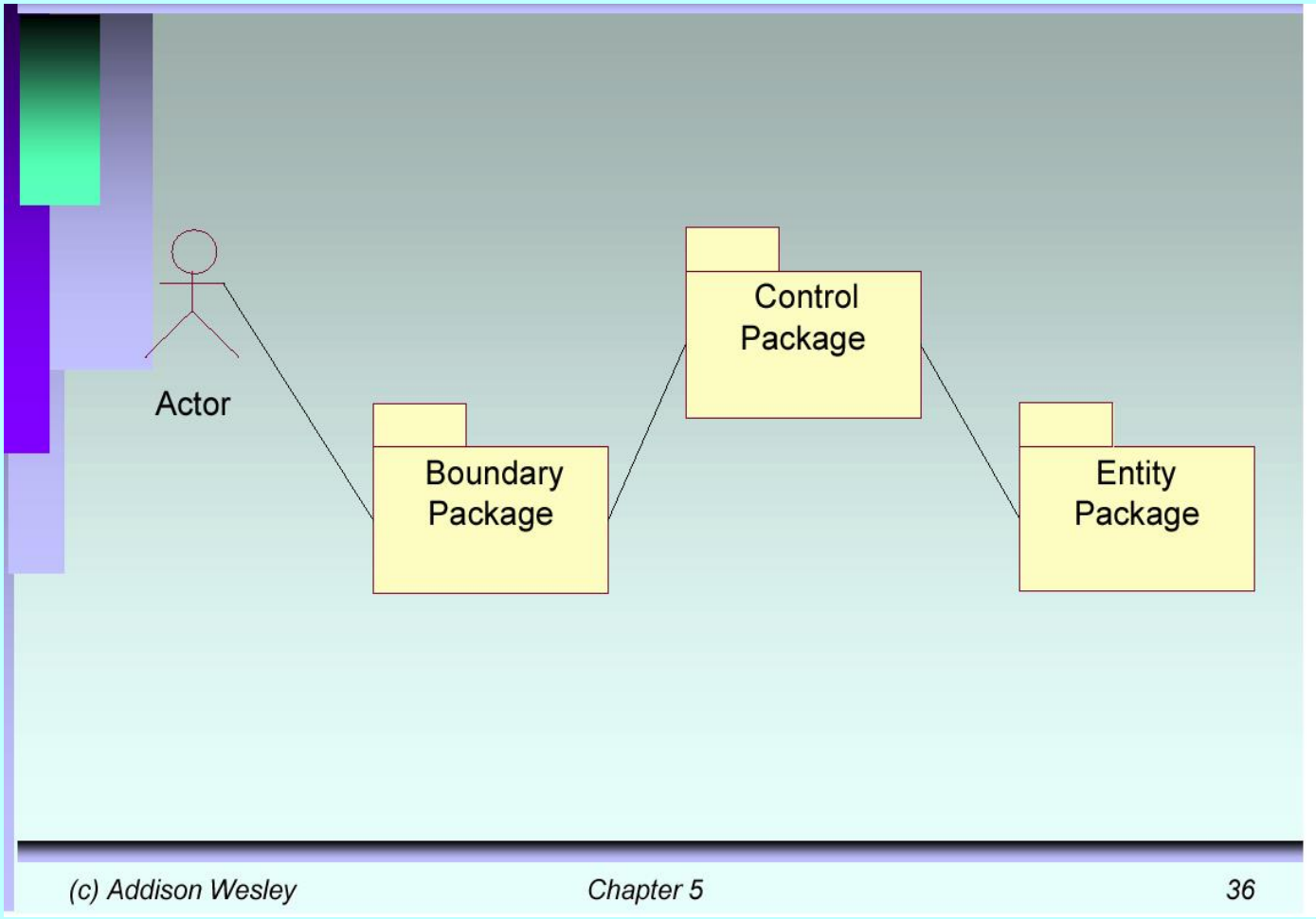


L'approccio Boundary-Control-Entity

Approccio BCE

- Package Boundary:
  - Contiene gli oggetti responsabili dell'interfaccia utente e della logica di presentazione
- Package Control:
  - Contiene oggetti che percepiscono gli eventi generati dall'utente e controllano l'esecuzione di un processo di business
  - Rappresentano azioni e attività degli use case
- Package Entity:
  - Contiene oggetti che rappresentano la semantica delle entità del dominio applicativo
  - Corrispondono alle strutture dati nel database del sistema
  - Sono sempre oggetti persistenti

Package BCE


Actor
Boundary Package
Control Package
Entity Package
(c) Addison Wesley
Chapter 5
36

BCE e MVC

- BCE è una variante di MVC che nasce con obiettivi diversi
  - BCE nasce con l'obiettivo di separare le responsabilità di elementi nell'OOD
  - MVC nasce con l'obiettivo di separare la selezione delle interfacce utente dal resto dell'applicazione

  - Il primo è più orientato a separare chiaramente la business logic dal resto in applicazioni OO, il secondo è più focalizzato sull'interfacciamento con l'utente

L'approccio BCED

- Boundary/ Control/ Entity/Database
  - Lascia le classi di sistema nel package entity e pone le classi responsabili dell'estrazione dei dati dal database nel package Database
  - Il Package Entity memorizza i dati che ottiene dal package DB invocando due servizi:
    - Carica(unOggetto)
    - Salva(unOggetto)
- Il package DB (che è di fatto un'interfaccia verso il DB)
  - apre e chiude connessioni con DB, estrae e memorizza le informazioni dei meta-dati sugli oggetti del DB, informa il DB di richieste di operazioni commit e rollback, …

Pattern architetturali e accesso a DB

- I pattern architetturali MVC, BCE, BCED necessitano di una persistenza dei dati.
- Un package (o componente) per l'accesso e la gestione dei dati.
- Deve essere presente una Base di Dati.

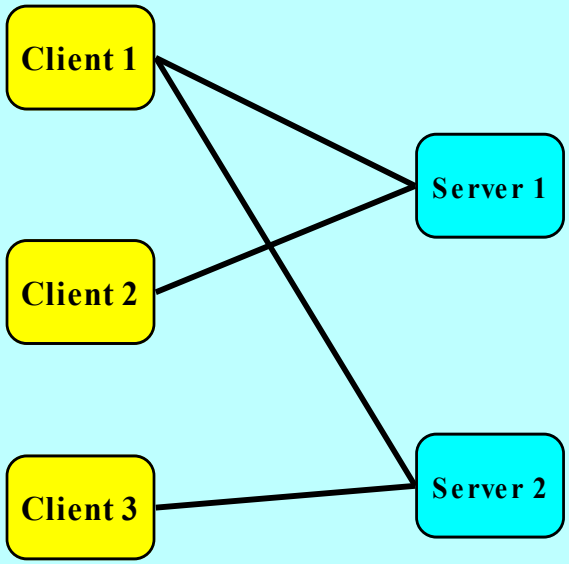
- L'accesso a DB può essere gestito con tre strategie:
  - Accesso diretto (forza bruta)
  - Oggetti per accesso ai dati (*Data Access Objects*, DAO)
  - *Persistence framework* (ad es.: EJB 3.0 e Hibernate per Java; NDO (.Net Data Objects) per .NET)

Pattern architetturali: Broker

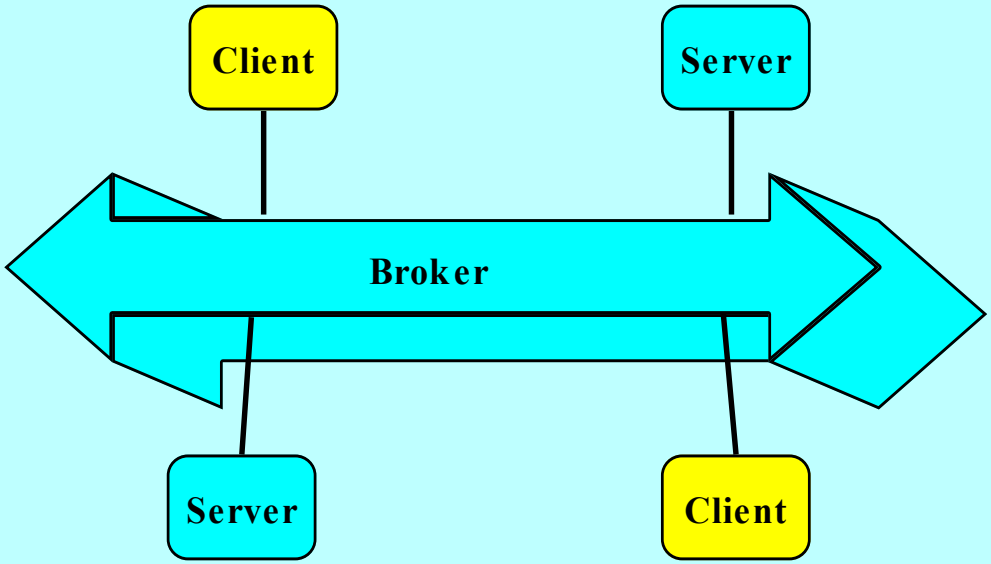
Il Pattern Broker

- Il Broker è un pattern architetturale, con schema logico di tipo publish/subscribe:
  - gli oggetti serventi rendono noto i servizi forniti;
  - oggetti clienti che sono interessati ad a uno o più servizi si iscrivono per utilizzare il servizio;
  - pubblicazioni e iscrizioni avvengono presso il Broker.
- Conferisce al sistema caratteristiche di trasparenza alla locazione, al linguaggio di programmazione e alla piattaforma target
- L'architettura con Broker rappresenta un sorta di bus (software) virtuale a cui ogni oggetto affida i suoi messaggi.
- Il Broker è l'elemento di integrazione fra i "moduli" che compongono il sistema software.

Il Pattern Broker e il modello Client/Server



Modello tradizionale Client/Server



Architettura a broker

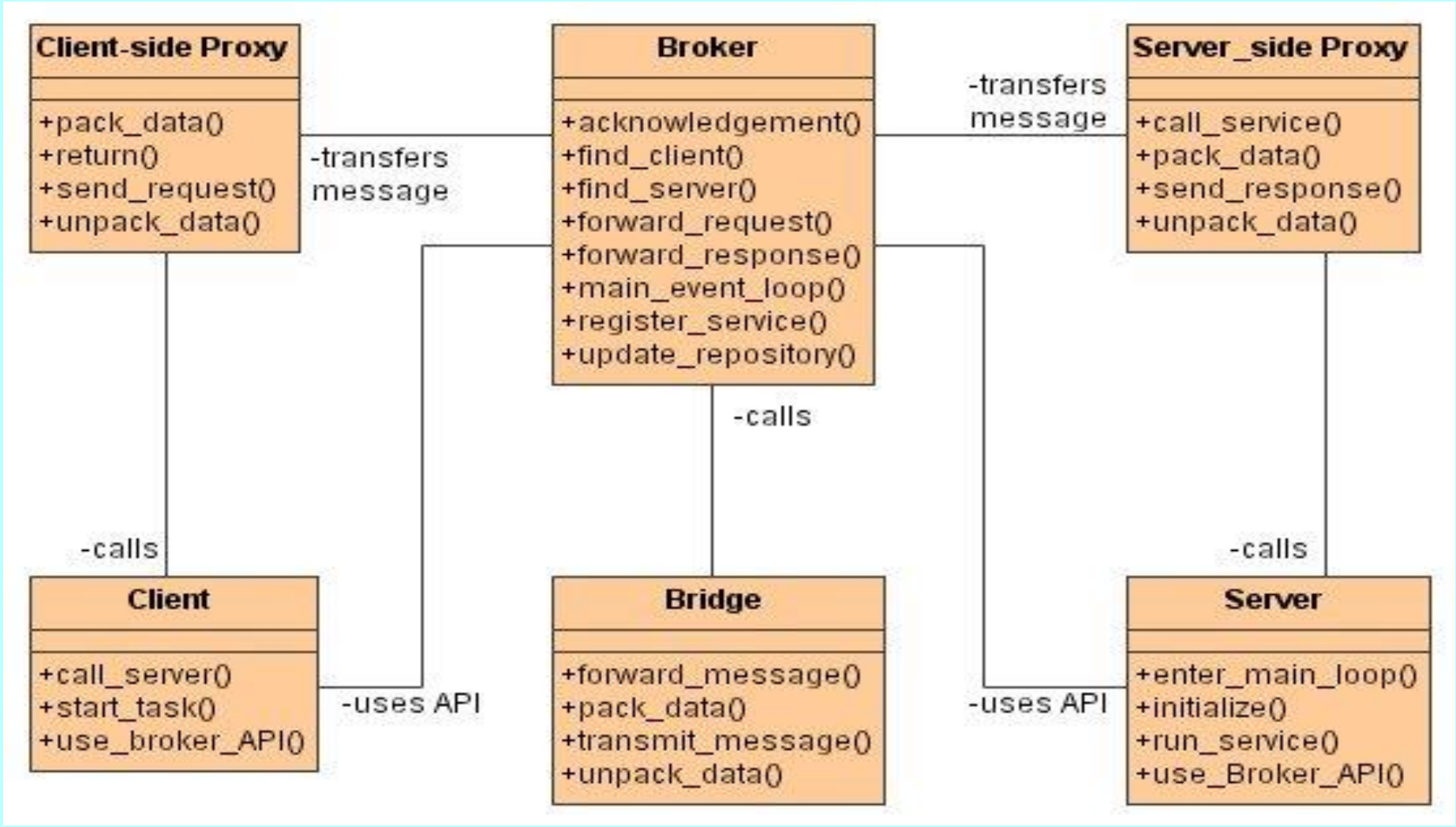
Il Pattern Broker

- Il Broker è un pattern molto utilizzato nelle architetture dei sistemi middleware.
- Esempi di middleware che lo utilizzano sono: OMG CORBA, IBM SOM/DSOM, Microsoft OLE 2.x
- Una applicazione può accedere a servizi distribuiti semplicemente inviando messaggi all'oggetto appropriato, piuttosto che doversi impegnare in una comunicazione inter-processo di basso livello.

Il Pattern Broker: modello statico

- Comprende sei tipi di componenti partecipanti:
  - **serventi**, implementano oggetti che espongono funzionalità, attraverso un'interfaccia che consiste di operazioni e attributi.
  - **clienti**, applicazioni che accedono ai servizi offerti dai serventi.
  - **broker**, componente responsabile della trasmissione delle richieste dai clienti ai serventi, e della trasmissione ai clienti delle risposte o delle eccezioni rilevate.
  - **bridge**, componenti opzionali, utilizzati per nascondere i dettagli di implementazione della comunicazione tra i broker in una rete eterogenea
  - **proxy lato cliente**, strato software addizionale tra cliente e broker, che rende trasparente al cliente l'accesso ad un oggetto remoto, che appare così come un oggetto locale
  - **proxy lato servente**, duale del proxy lato cliente.

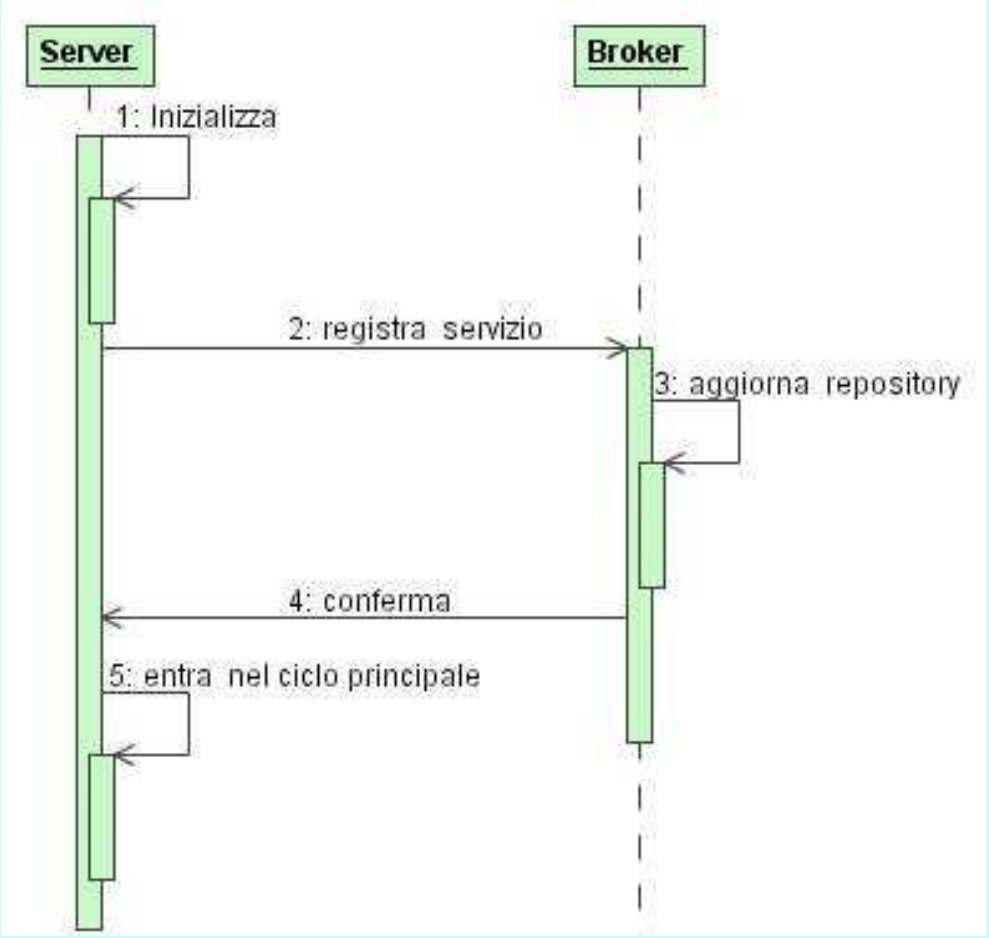
Il Pattern Broker: modello statico



Il Broker e il Bridge

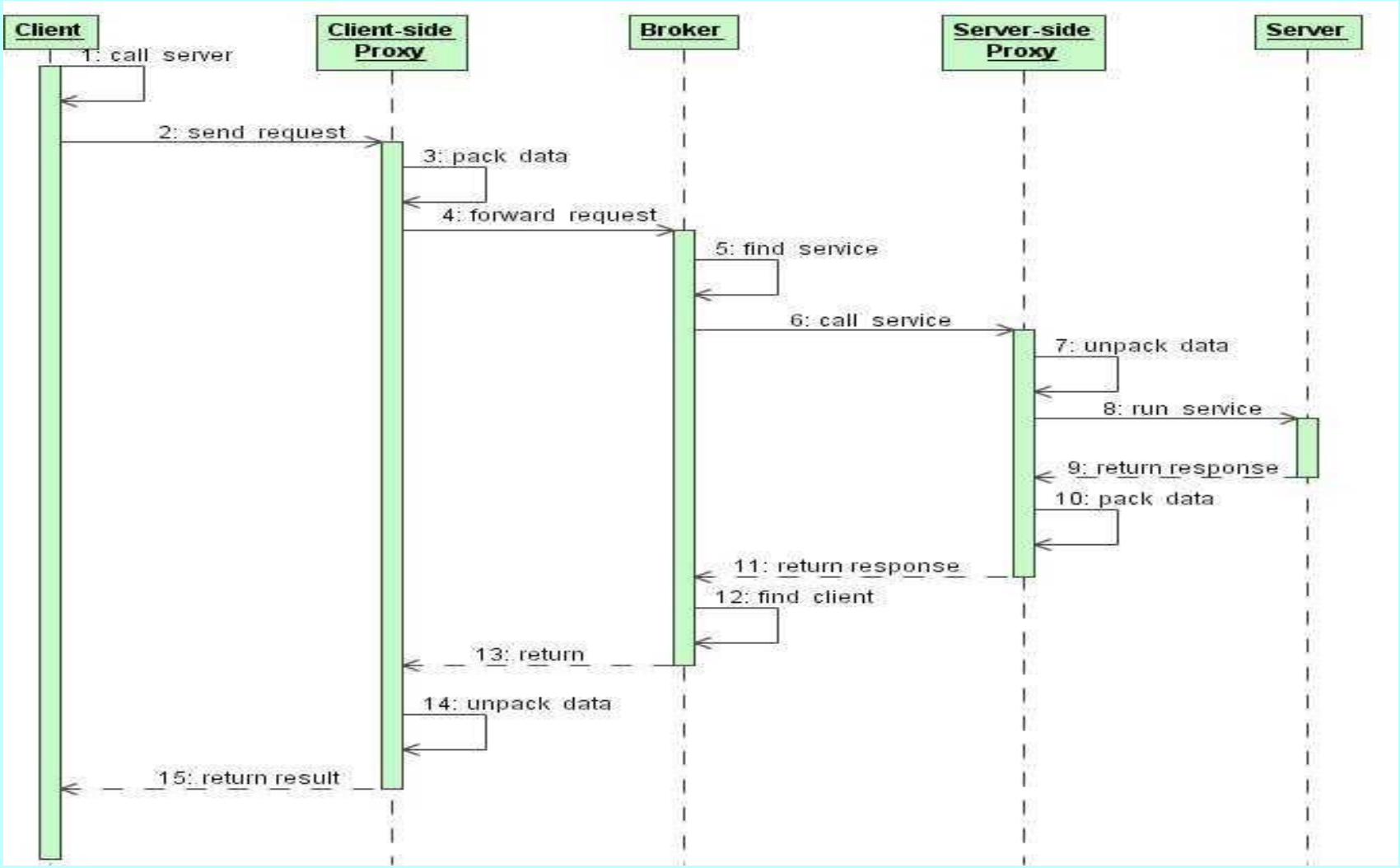
- Un broker è un componente responsabile della trasmissione delle richieste dai clienti ai serventi, e della trasmissione ai clienti delle risposte o delle eccezioni rilevate
- Un broker offre dei metodi (API) ai clienti e ai serventi, che includono operazioni per registrare un server e per invocarne i servizi
- I bridge sono componenti opzionali del pattern, utilizzati per nascondere i dettagli di implementazione della comunicazione tra i broker in modo che, in una rete eterogenea, i broker comunichino in maniera trasparente, rispetto all'architettura di rete e ai sistemi operativi in uso

Registrazione di un server sul broker locale



1. Il broker viene eseguito nella fase di inizializzazione del sistema. Entra così in un ciclo di attesa eventi, aspettando che arrivino messaggi.
2. L'utilizzatore esegue l'applicazione server, che esegue una fase di inizializzazione dopo la quale si registra presso il broker.
3. Il broker riceve il messaggio di richiesta registrazione, estrae le informazioni necessarie e le memorizza in un'area dati, il repository, che serve per trovare e attivare i server, e invia un messaggio di avvenuta registrazione al server.
4. Ricevuta la conferma, il server entra nel ciclo di attesa delle richieste provenienti dai clienti.

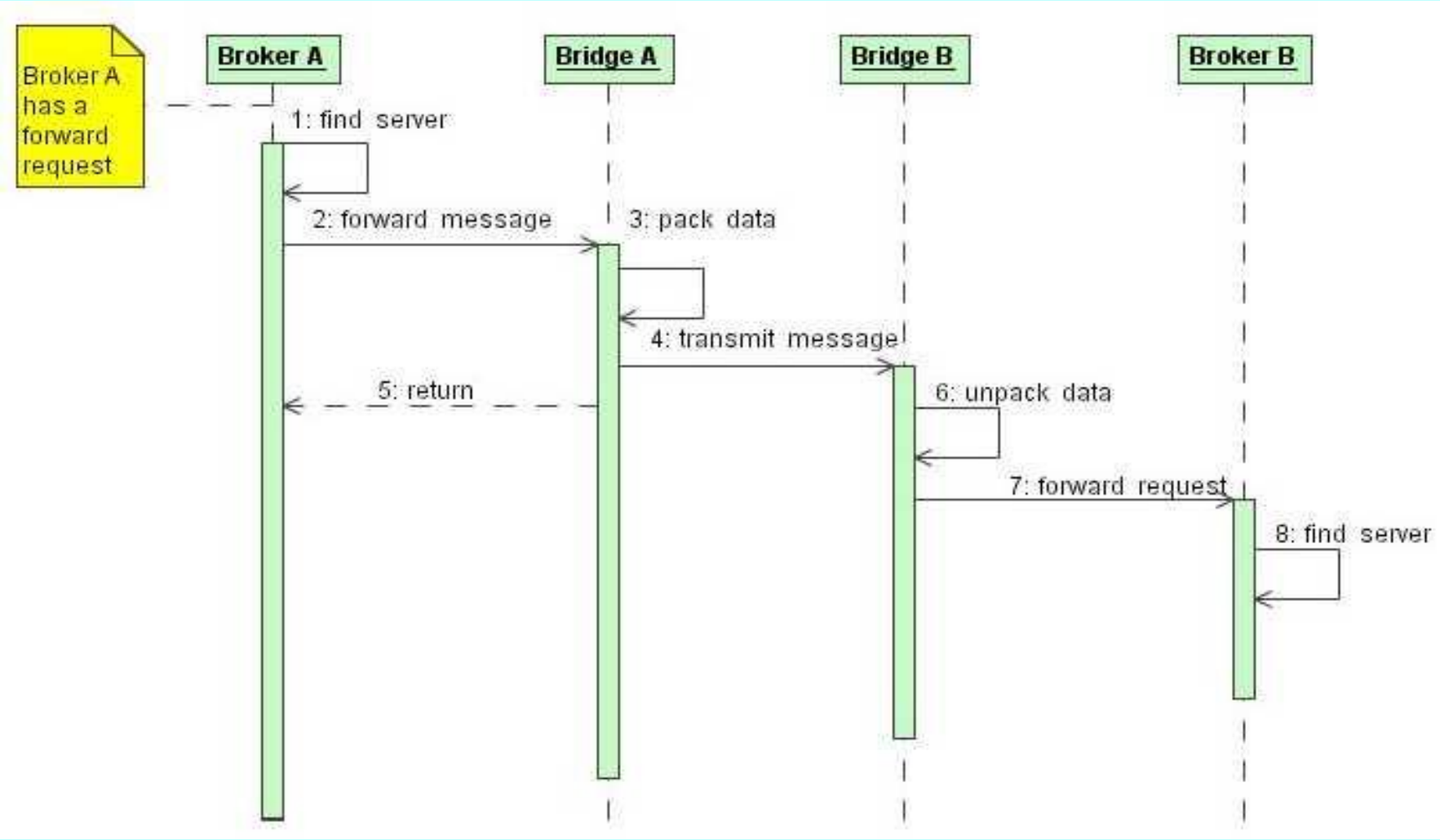
Un cliente invia una richiesta ad un broker locale – 1/2



Un cliente invia una richiesta ad un broker locale – 2/2

1. L'applicazione cliente viene eseguita. Durante l'esecuzione del programma il cliente invoca un metodo di un oggetto server remoto.
2. Il proxy lato cliente impacchetta tutti i parametri ed altre informazioni supplementari, in un messaggio che viene trasferito al broker locale.
3. Il broker ricerca la posizione del server richiesto all'interno del suo repository. Se il server è disponibile localmente, il broker trasferisce il messaggio al corrispondente proxy lato server (il caso di server remoto lo vediamo nel prossimo scenario).
4. Il proxy lato server spacchetta i parametri e le informazioni supplementari, tra cui il metodo che bisogna invocare. A questo punto il proxy richiama il servizio appropriato.
5. Dopo che il servizio è stato eseguito, il server restituisce il risultato al proxy lato server, che lo impacchetta, insieme ad informazioni supplementari, in un messaggio che trasferisce poi al broker.
6. Il Broker trasferisce la risposta al proxy lato client.
7. Il proxy lato client riceve la risposta, spacchetta il messaggio e lo invia all'applicazione cliente, che dopo aver ricevuto i dati di suo interesse, continua la sua esecuzione normalmente.

Interazione tra broker tramite il bridge – 1/2



Interazione tra broker tramite il bridge – 2/2

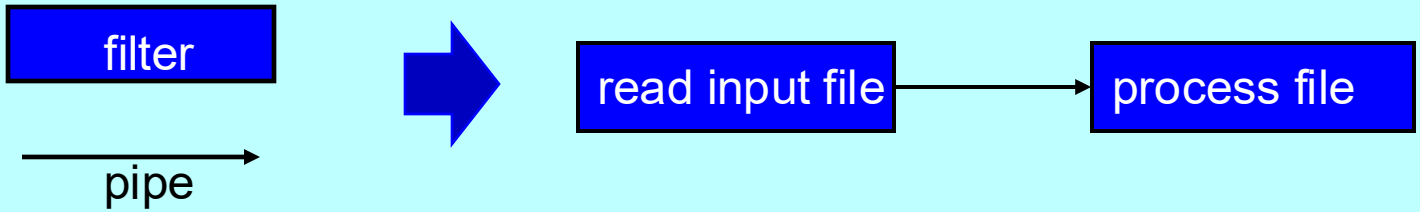
1. Il broker A riceve una richiesta di servizio. Esso ricerca nel repository il server responsabile dell'esecuzione del servizio. Siccome il server corrispondente è disponibile presso un altro nodo della rete, A trasmette la richiesta al broker remoto.
2. Il messaggio passa dal broker A al bridge A. Questo componente è responsabile della conversione del messaggio dal protocollo specifico del broker A, in un protocollo di rete comune ai due bridge partecipanti. Dopo la conversione, il bridge A trasmette il messaggio al bridge B.
3. Il bridge B converte il messaggio in ingresso, dal protocollo di rete al formato del protocollo specifico del broker B.
4. Il broker B esegue tutte le azioni necessarie all'arrivo di una richiesta di servizio.

CENNI

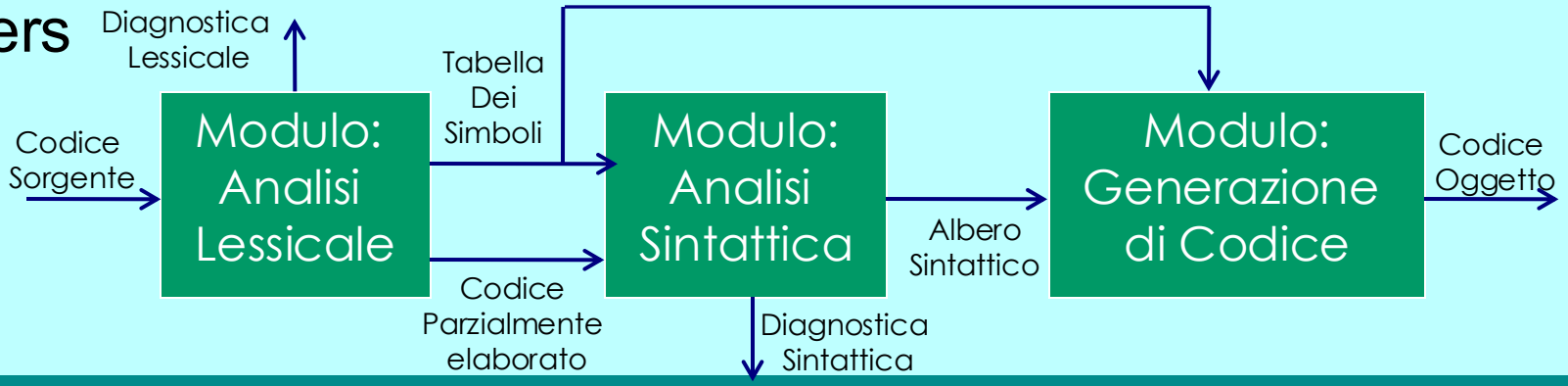
Altri pattern architetturali

Pipe & Filter

- A component reads streams of data as input and produces streams of data as output
- Suitable for applications that require a defined series of independent computations to be performed on data

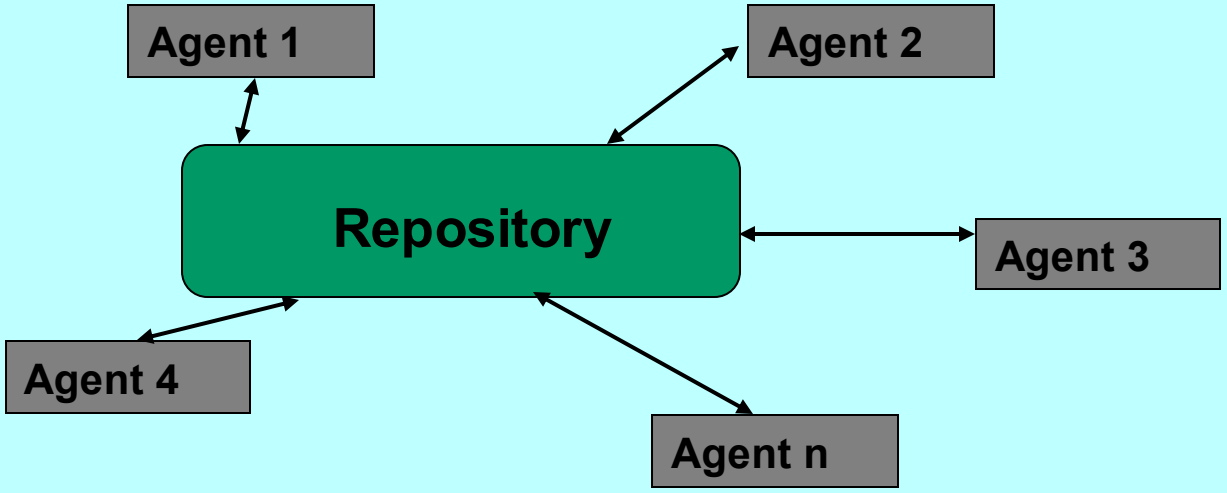


- Examples
  - Unix shell command line processing of the pipe symbol "|"
  - Compilers



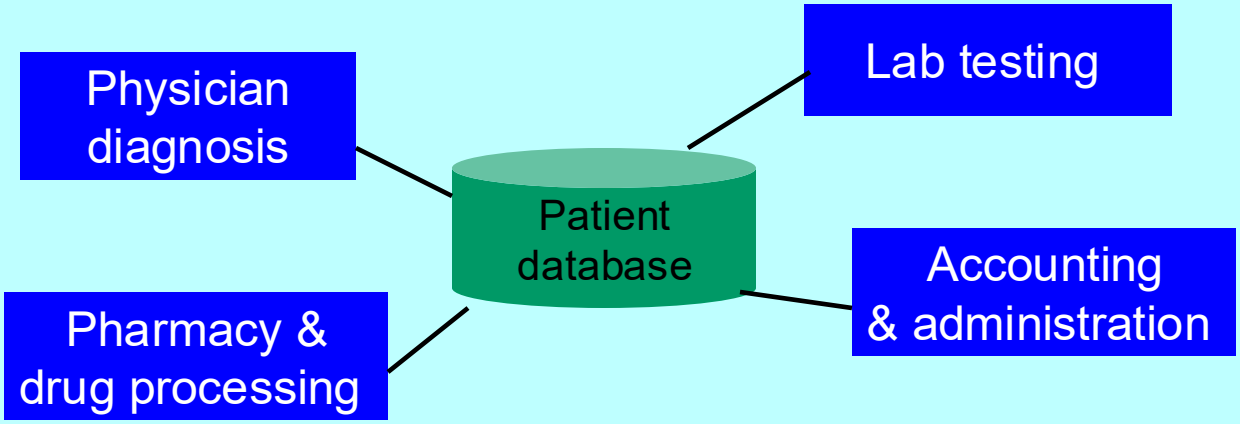
Shared Data-Repository

- I sottosistemi accedono e modificano una singola struttura dati condivisa chiamata **repository (*shared data store*)**
  - *Blackboard (passive repository) style*: I partecipanti monitorano I cambiamenti del data store
  - *Publish&subscribe (active repository) style*: il data-store notifica I partecipanti quando c'è un cambiamento



Shared Data-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)

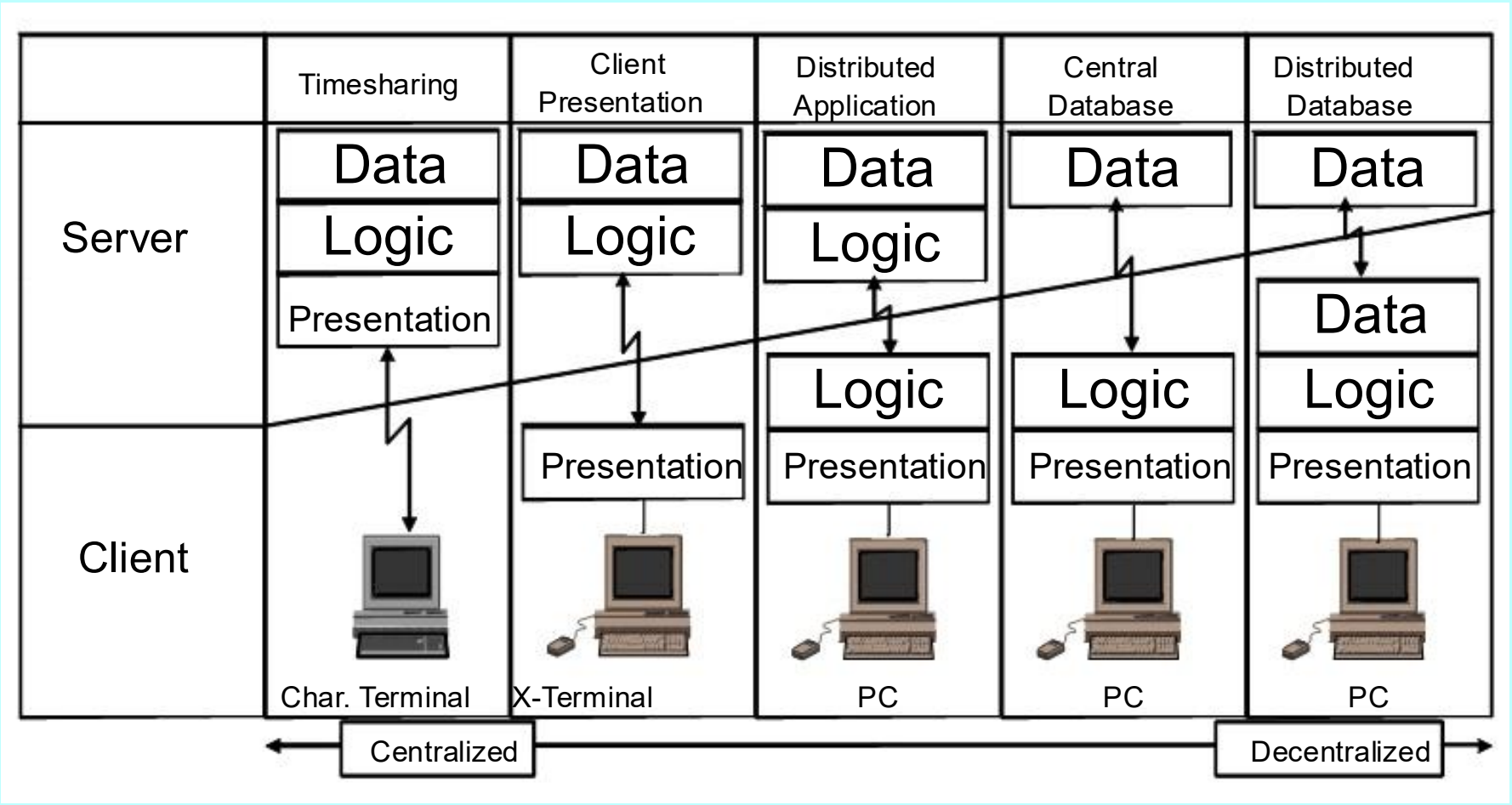


Client/Server

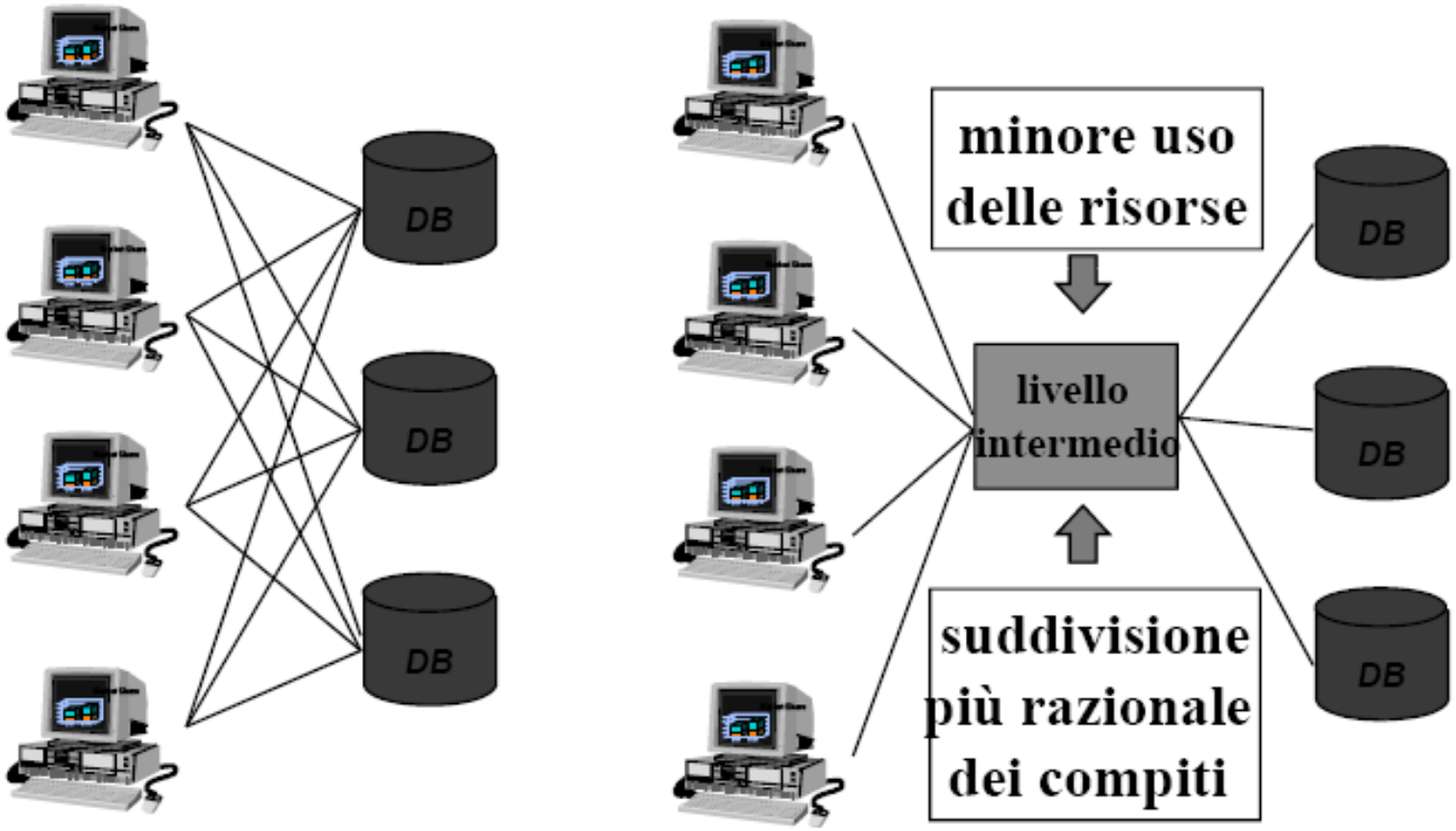
- Architettura distribuita dove dati ed elaborazione sono distribuiti su una rete di nodi di due tipi:
  - Server: processori potenti e dedicati che offrono servizi specifici come stampa, gestione di file system, compilazione, gestione traffico di rete, calcolo.
  - Clienti: macchine meno prestazionali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

- Ogni applicazione può essere suddivisa logicamente in 3 parti:
  - **Presentazione,** che gestisce l'interfaccia utente (gestione eventi grafici, controlli formali sui campi in input, help, …)
  - **Logica applicativa**
  - **Gestione dei dati persistenti**
- La politica di allocazione di queste componenti porta alla classificazione dell'architettura: *2-tiered, 3-tiered, n-tiered*

Soluzioni Client/Server



Architetture *Two-tiers* e *Three-tiers*



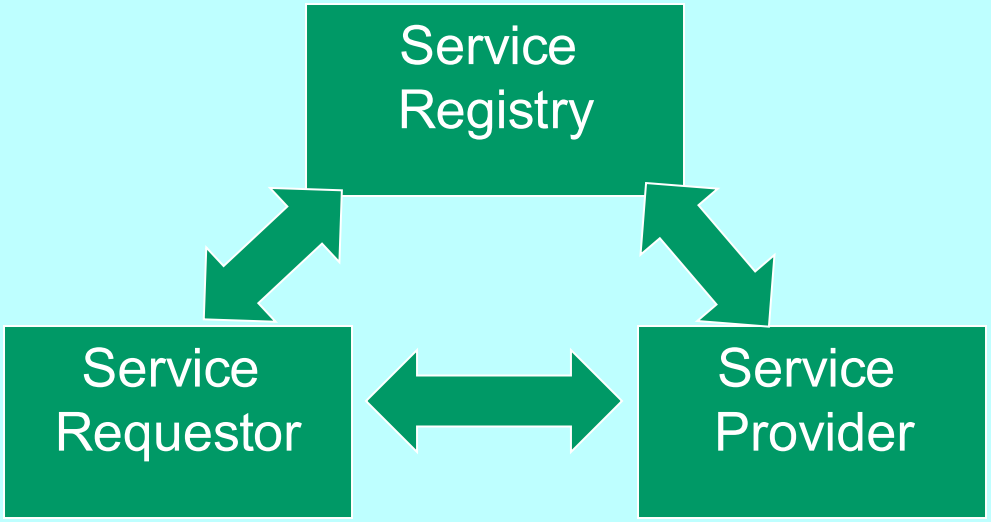
Architettura *Three-tiers*

- • *Business logic* trattata in modo esplicito:
  - Livello 1: gestione dei dati (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 sugli 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
- Ogni componente può essere contemporaneamente parte di applicazioni diverse

- Vantaggi in termini di flessibilità, manutenibilità, testabilità, estensibilità

Service-oriented architecture

- **Requestor**: l'utente potenziale (client)
- **Provider**: Chi implementa il servizio
- **Registry**: uno spazio logico in cui sono elencati i servizi disponibili, che consente ai provider di pubblicizzare I servizi ed ai requestor di cercarli ed interrogarli



- Esempi:
  - *Web Services (e.g.: AWS)*
  - *Servizi su infrastrutture cloud (e.g., SaaS, PaaS, IaaS)*
- Evoluzione: **Architetture a Microservizi**

Service-oriented architectures

- SOAs represent an evolution of client-server architectures
- 3rd generation of distributed software systems