Università degli Studi di Napoli “Federico II”

Ingegneria del Software

a.a. 2009/10

Argomento 11: System Design e Architetture Software

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

Organizzare sottosistemi in Architetture

Definizione di Architettura SW

- "*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]

- Informalmente, un'architettura software è la **struttura del sistema**, costituita dalle varie parti che lo compongono, con le relative relazioni.

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
  - Cambia il livello di astrazione, ma non i concetti sottostanti
  - 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

Layers e Partizionamenti

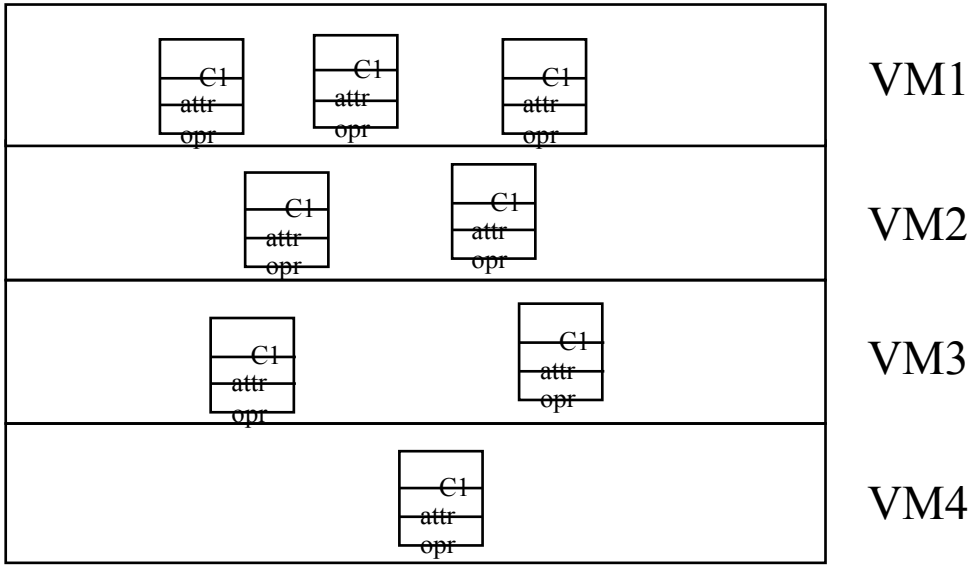
- La definizione dell'architettura logica di un sistema grande è di solito ottenuta decomponendolo in sottosistemi, usando **layer** e **partizioni**.
- Sono due visioni ortogonali tra loro:
  - Un Layer è uno strato che fornisce servizi
    - Es: Interfaccia Utente; Accesso al DB
  - Una Partizione è un'organizzazione di moduli
    - Es: Funzionalità Carrello di un sito di commercio

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

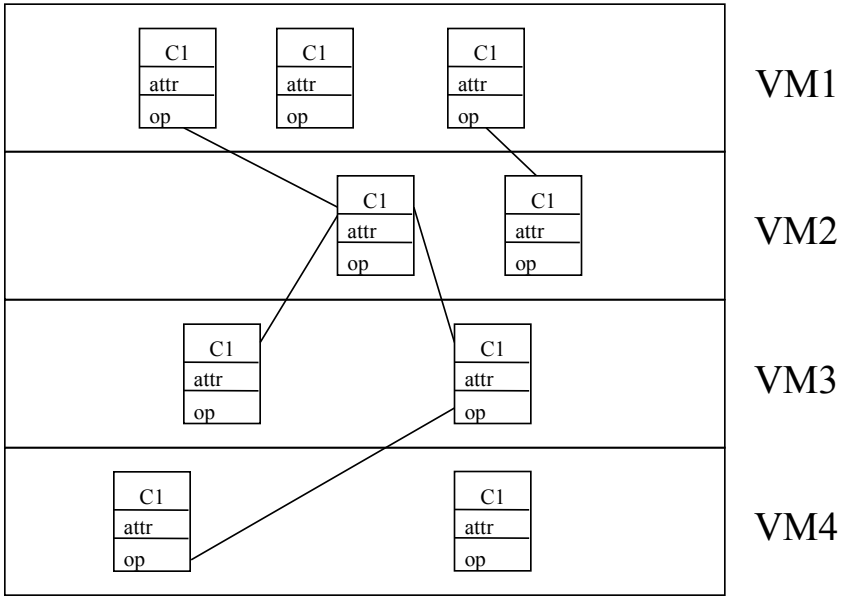
Macchina Virtuale (Dijkstra, 1965)

- 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.



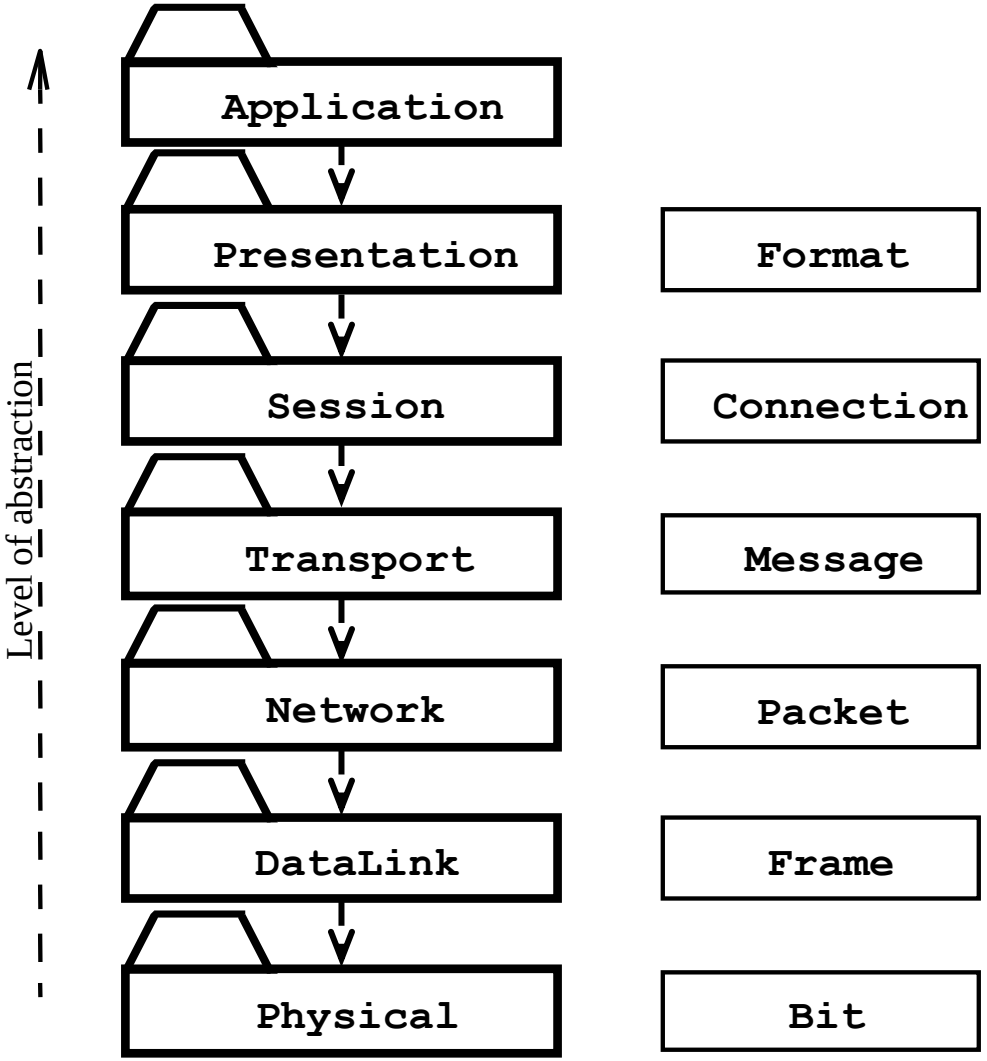
Architettura Chiusa

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



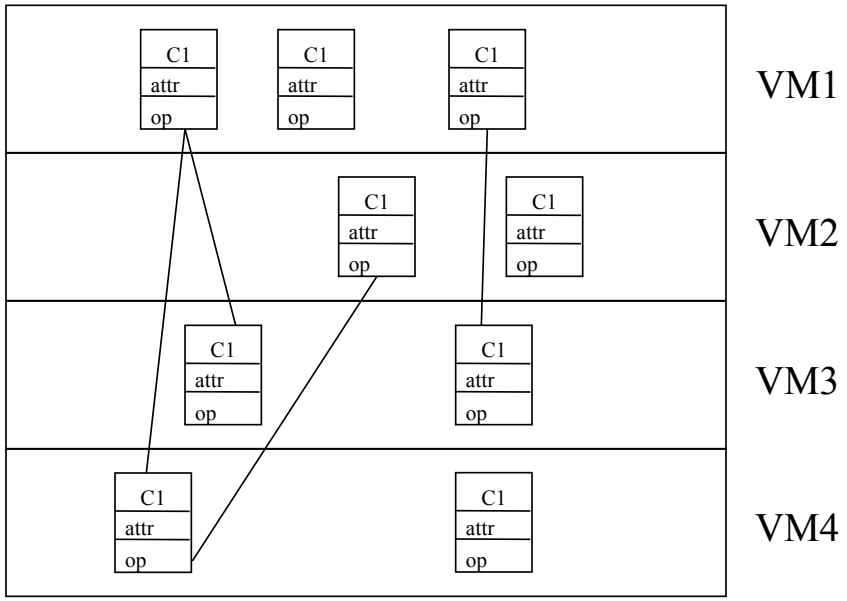
Esempio di Architecture Chiusa

- Pila ISO OSI
- Il modello di riferimento definisce 7 layer di protocolli di rete e metodi di comunicazione tra i layer.



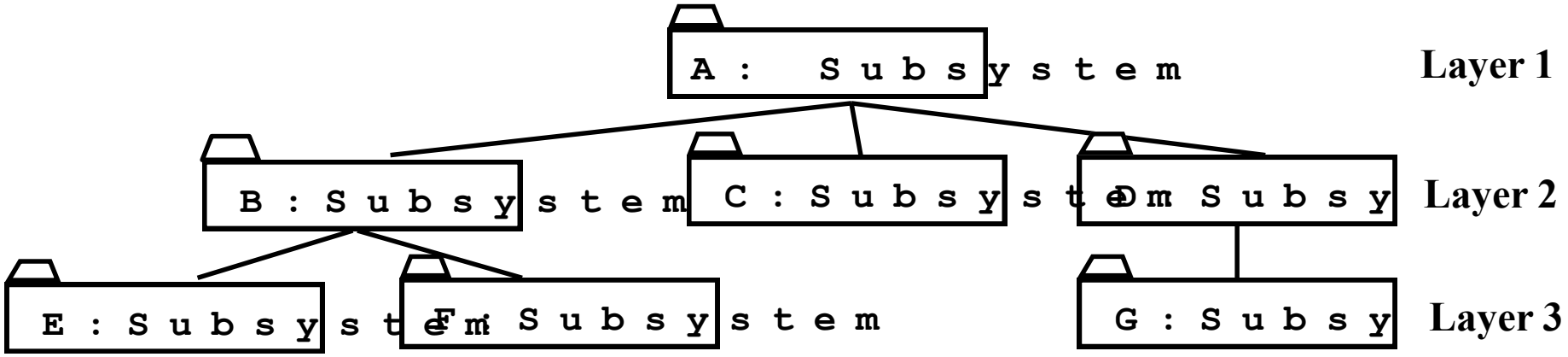
Architettura Aperta

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



Decomposizione di sottosistemi in Layer

- Euristiche per la decomposizione di sottosistemi:
- Non più di 7+/-2 sottosistemi
  - Più sottosistemi accrescono la cohesion ma anche la complessità (più servizi)
- Non più di 5+/-2 layer

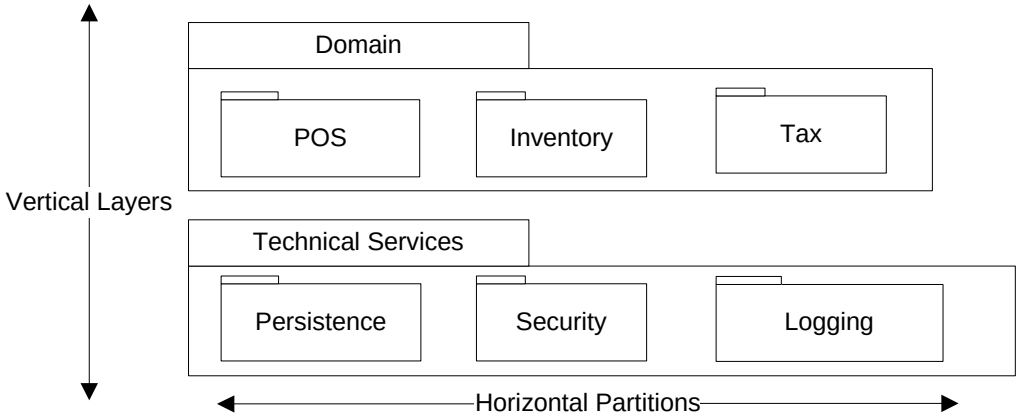


Partition

- Un altro approccio per trattare con la complessità consiste nel suddividere (partitioning) il sistema in sottosistemi paritari (peer) fra loro, ognuno responsabile di differenti classi di servizi.

- In generale una decomposizione in sottosistemi è il risultato di un'attività di partitioning e di layering
  - Prima si suddivide il sistema in sottosistemi al top-level che sono responsabili di specifiche funzionalità (partitioning)
  - Poi, ogni sottosistema è organizzato in diversi layer, se il livello di complessità lo richiede, finché non sono semplici abbastanza da poter essere implementate da un singolo sviluppatore (layering)

Architettura Logica: Layers e Partizioni



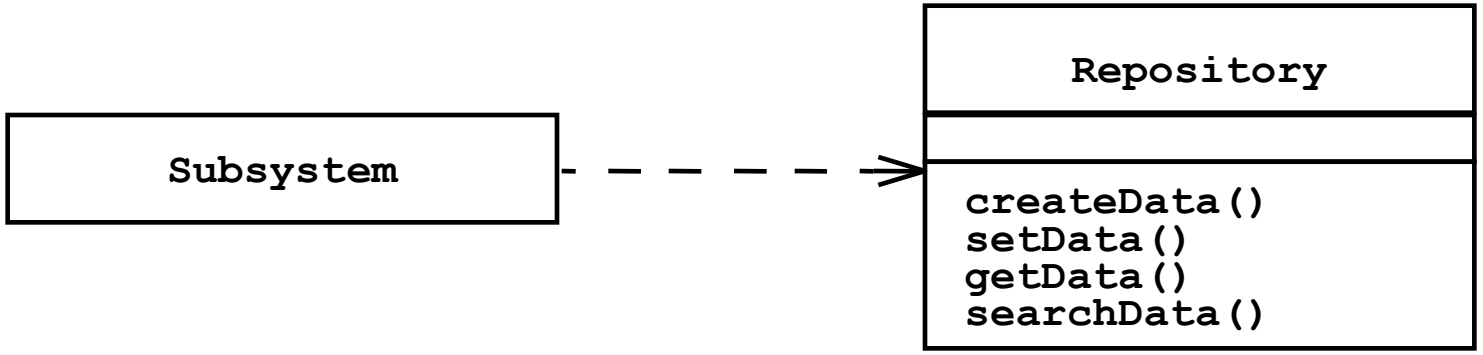
Principali Architetture

Principali Software Architectures

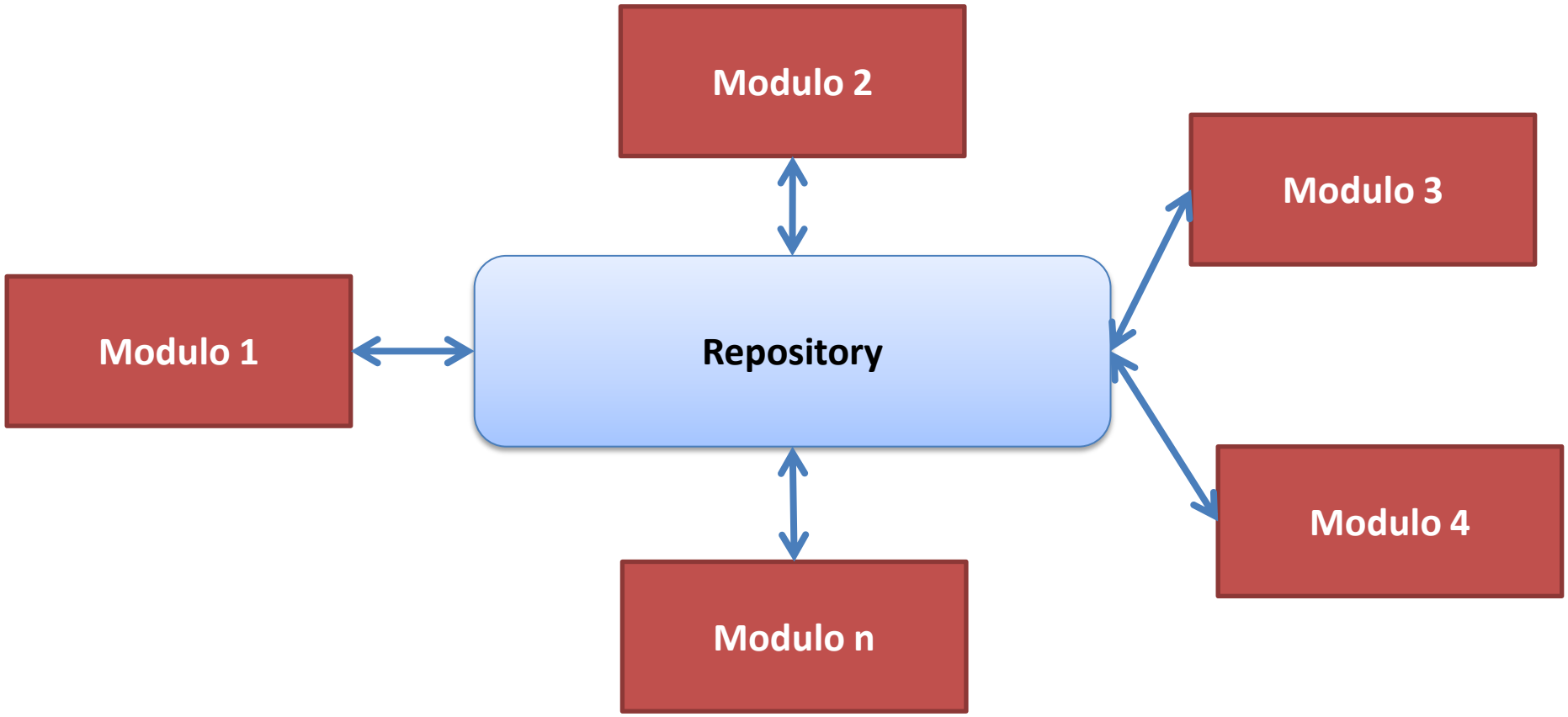
- Nell'ingegneria del sw sono stati definiti vari stili architetturali che possono essere usati come base di architetture software:
  - Repository Architecture
  - Client/Server Architecture
  - Peer-To-Peer Architecture
  - Model/View/Controller

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)

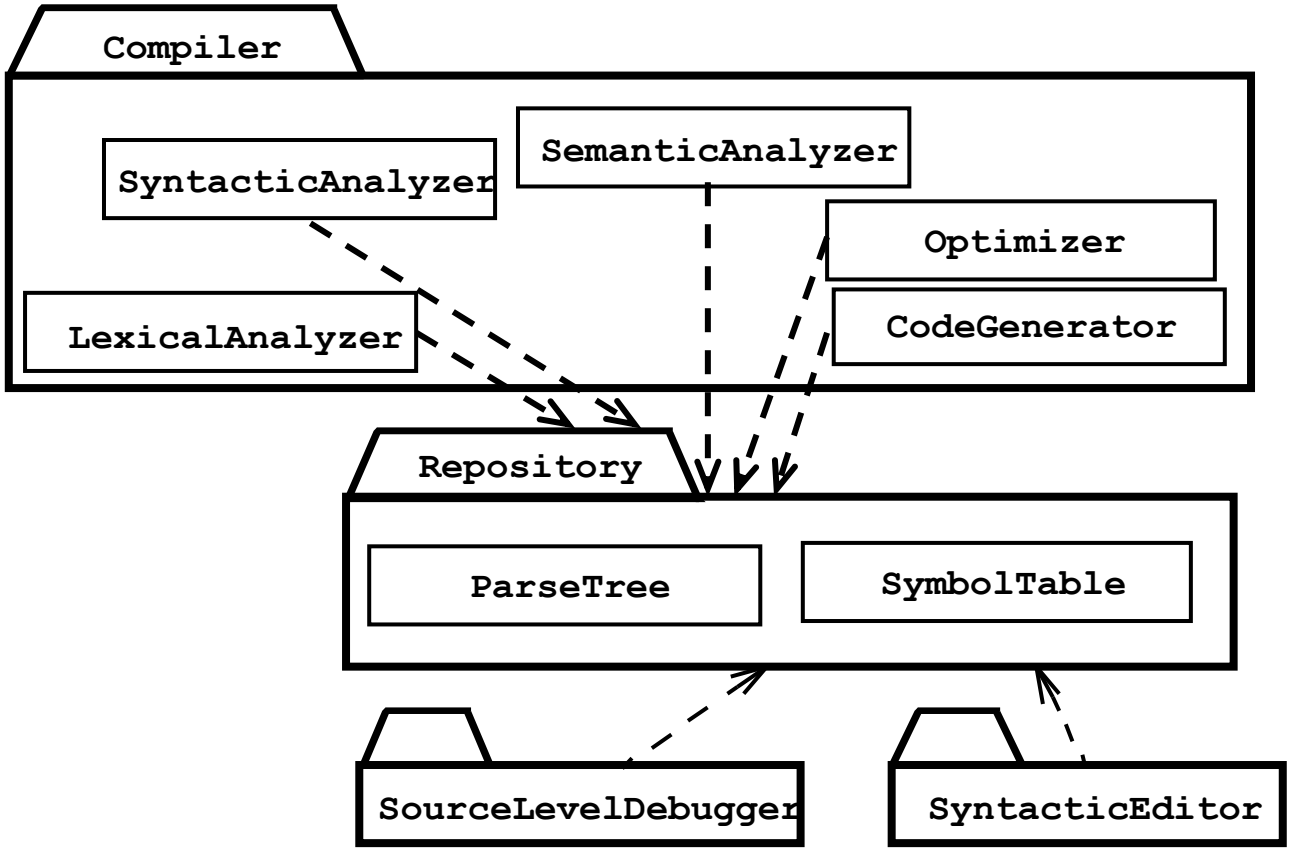


Repository Architecture



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.

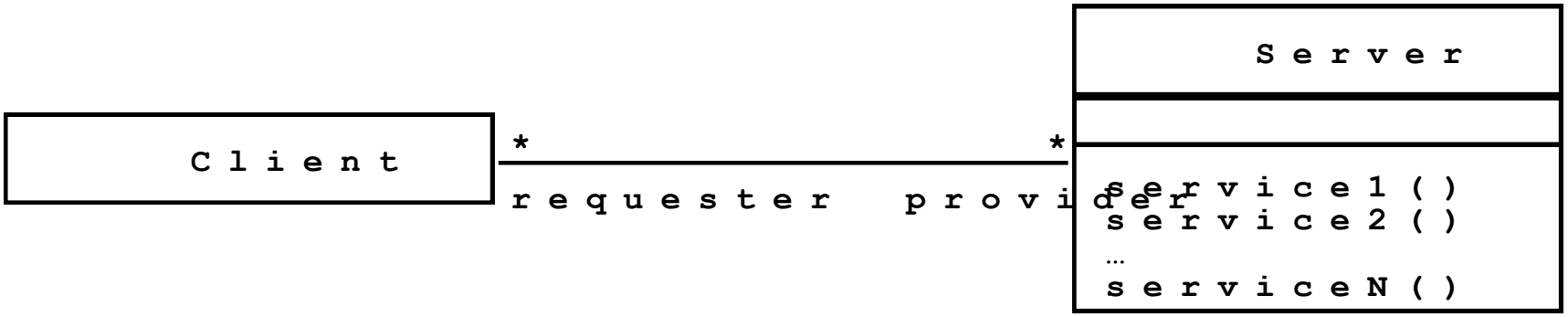
Architettura Client-server

- 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.

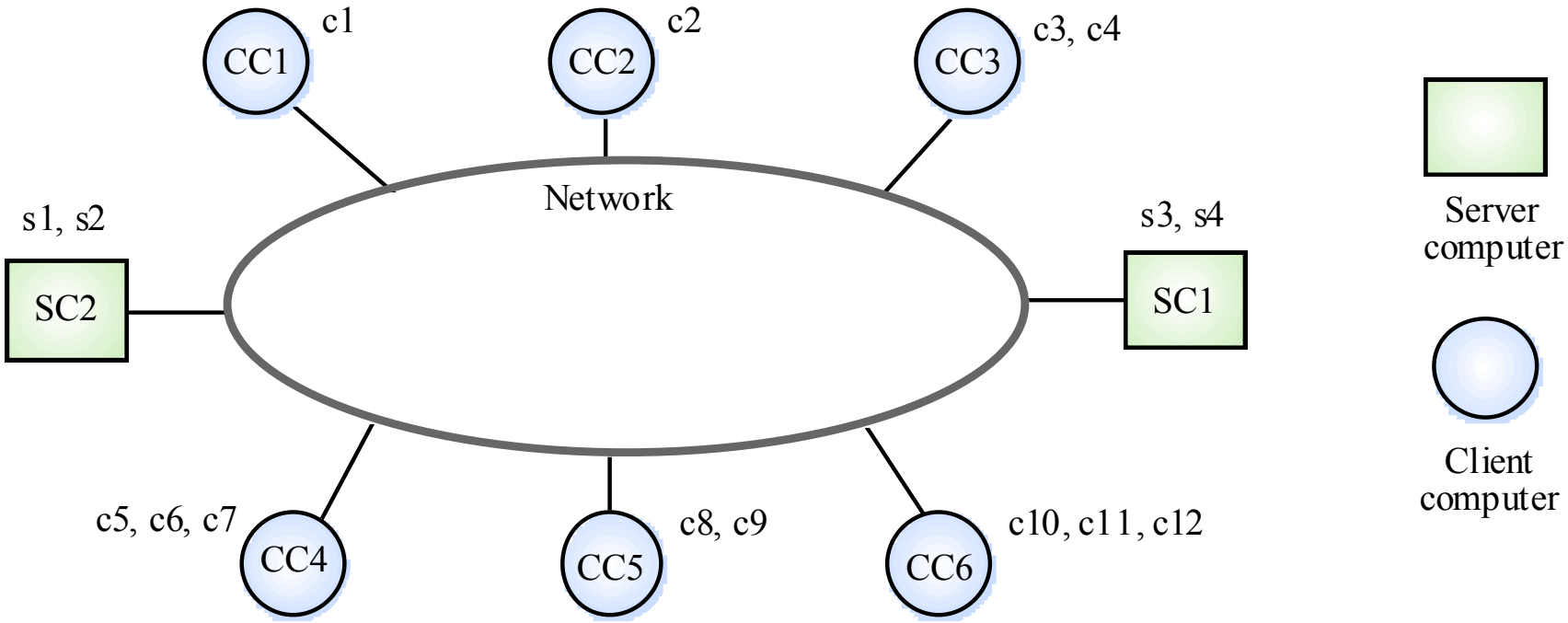
- I Client devono conoscere i nomi e la natura dei Server;
- I Server non devono conoscere identità e numero dei Clienti.

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

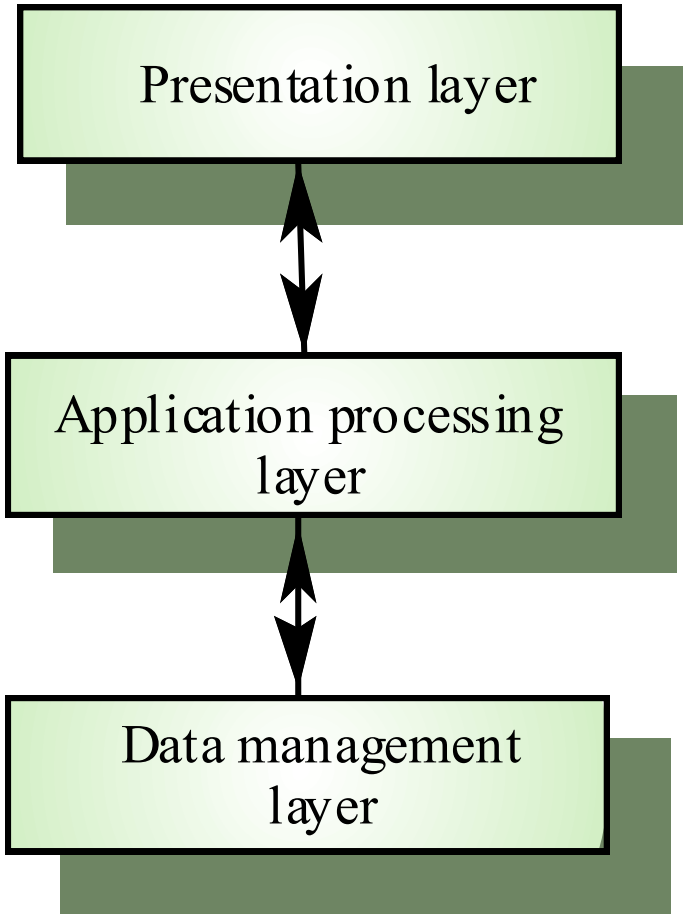


Computers in a C/S network

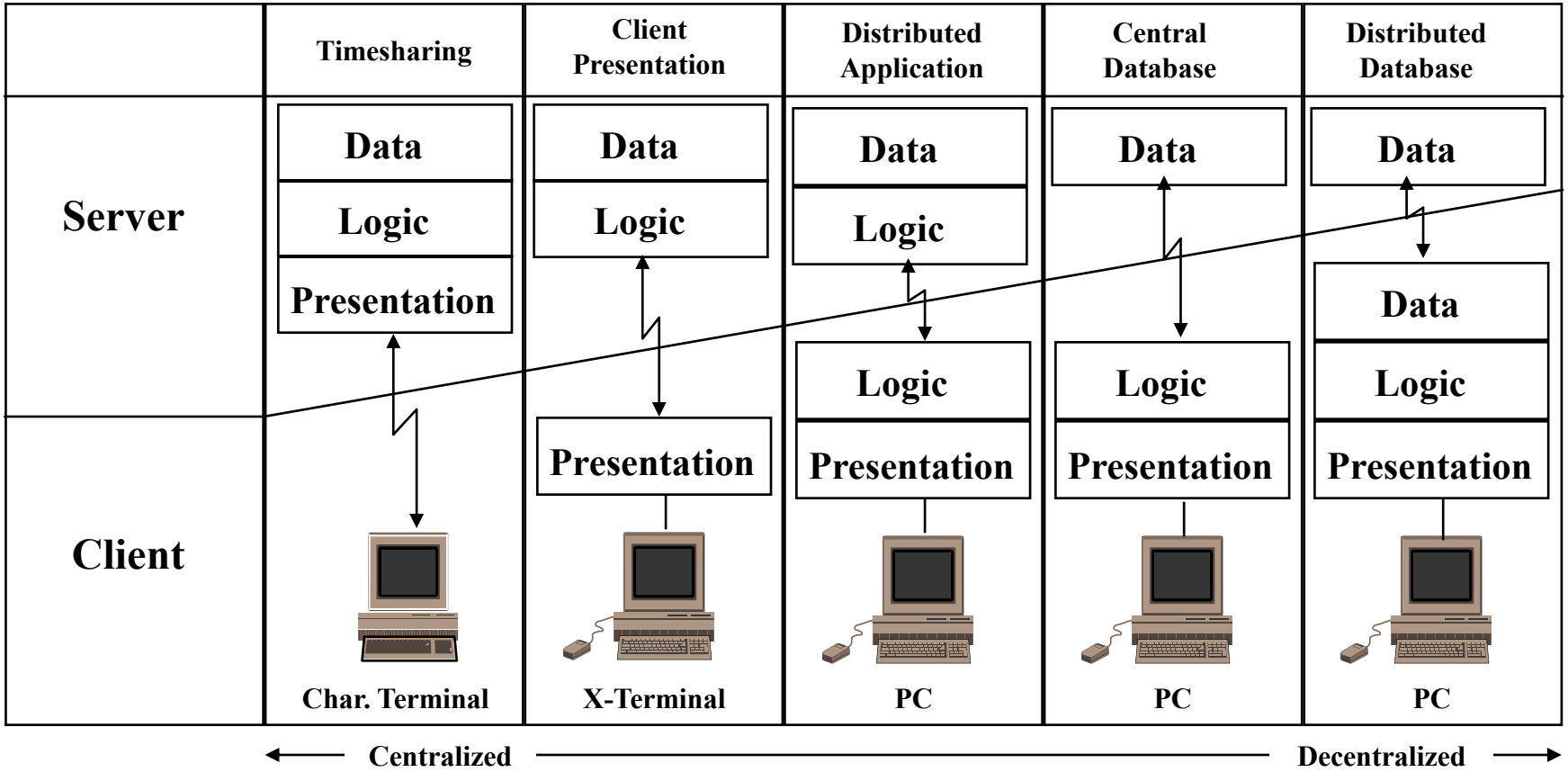


Strati funzionali nell'architettura C/S

- Praticamente ogni applicazione può essere suddivisa logicamente in tre parti:
  - La **presentazione** che gestisce l'interfaccia utente (gestione degli eventi grafici, controlli formali sui campi in input, help, …)
  - La **logica applicativa** vera e propria
  - La **gestione dei dati** persistenti su database
- La scelta della politica di allocazione di queste componenti porta alla classificazione dell'architettura C/S:
  - 2-tiered
  - 3-tiered
  - n-tiered



Classificazione delle soluzioni Client/Server a 2 livelli

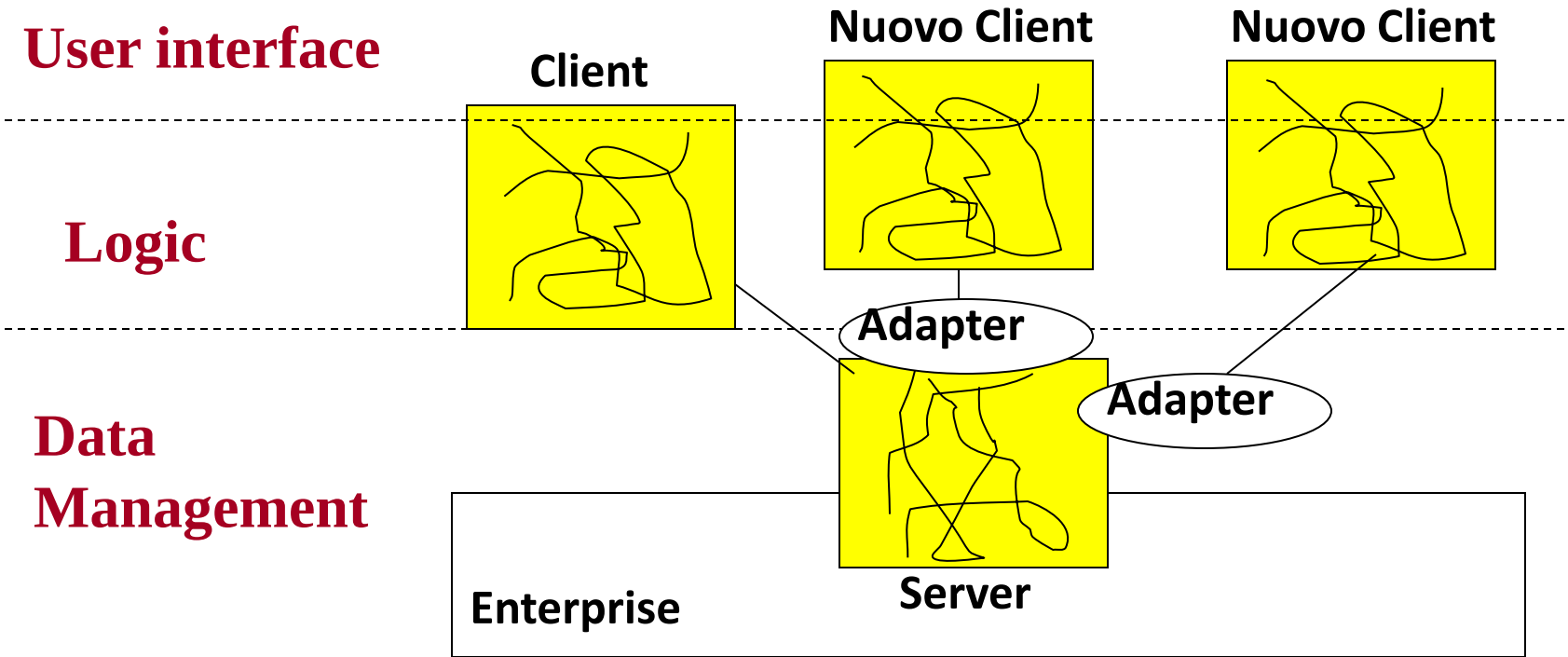


Architetture client/server a 2 livelli

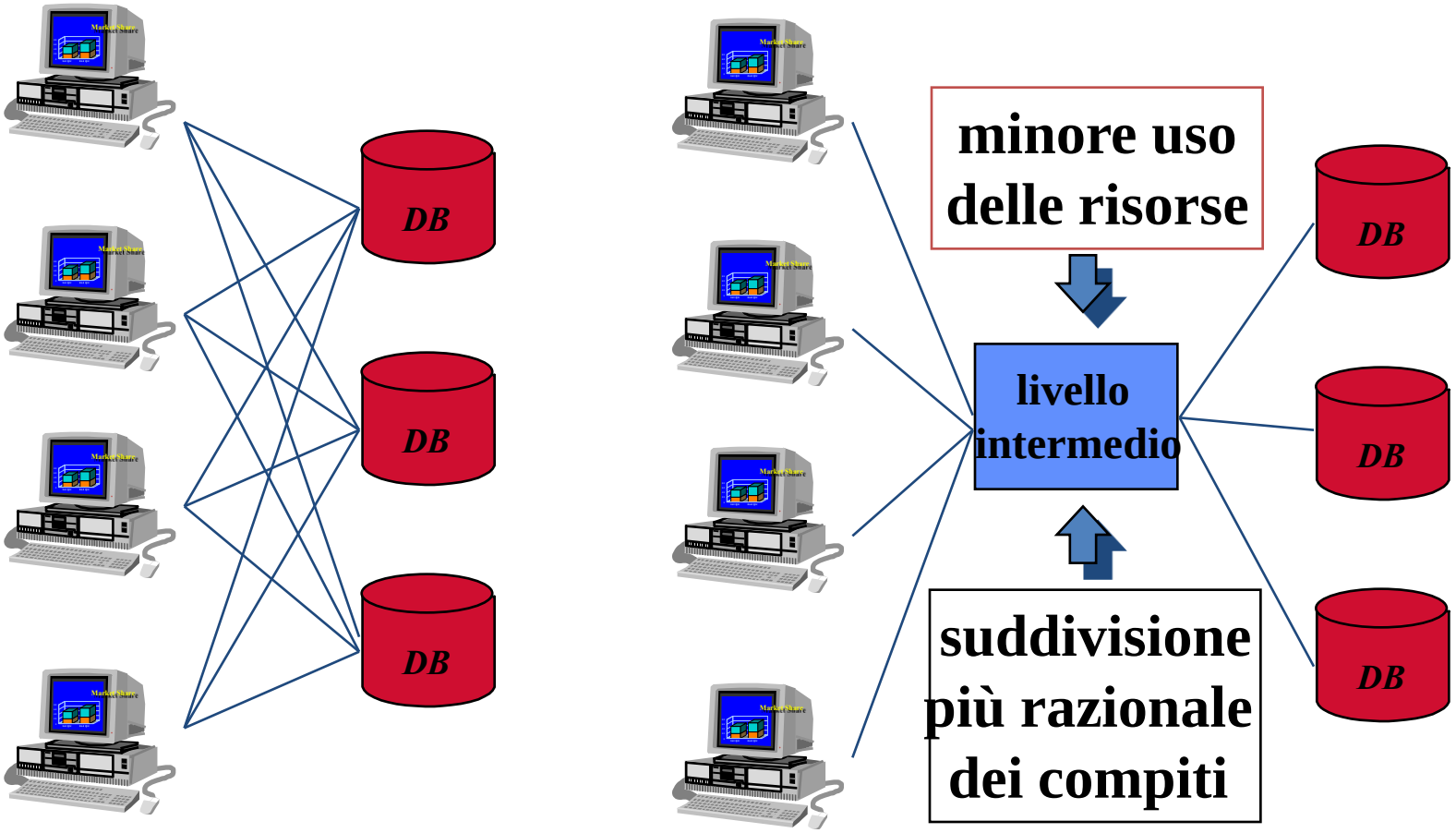
- Vantaggi:
  - E’ la più semplice architettura distribuita
- Svantaggi:
  - Traffico di messaggi intenso (frontend comunica continuamente con server dati)
  - Logica di business non gestita in una componente separata dell’architettura: suddivisa tra frontend e backend
    - client e server di applicazione dipendono l’uno dall’altro
    - difficile riutilizzare interfaccia in servizio che accede a dati diversi
    - tendenza a cablare la business logic nell’interfaccia utente → cambio di logica significa cambiare anche interfaccia

Architetture client/server a 2 livelli

- Problema: mancato riconoscimento dell'importanza della business logic
  - Es: servizio accessibile da più device (telefonino, desktop) → stessa logica, interfaccia diversa



Architetture 2-Tiers vs. 3-Tiers



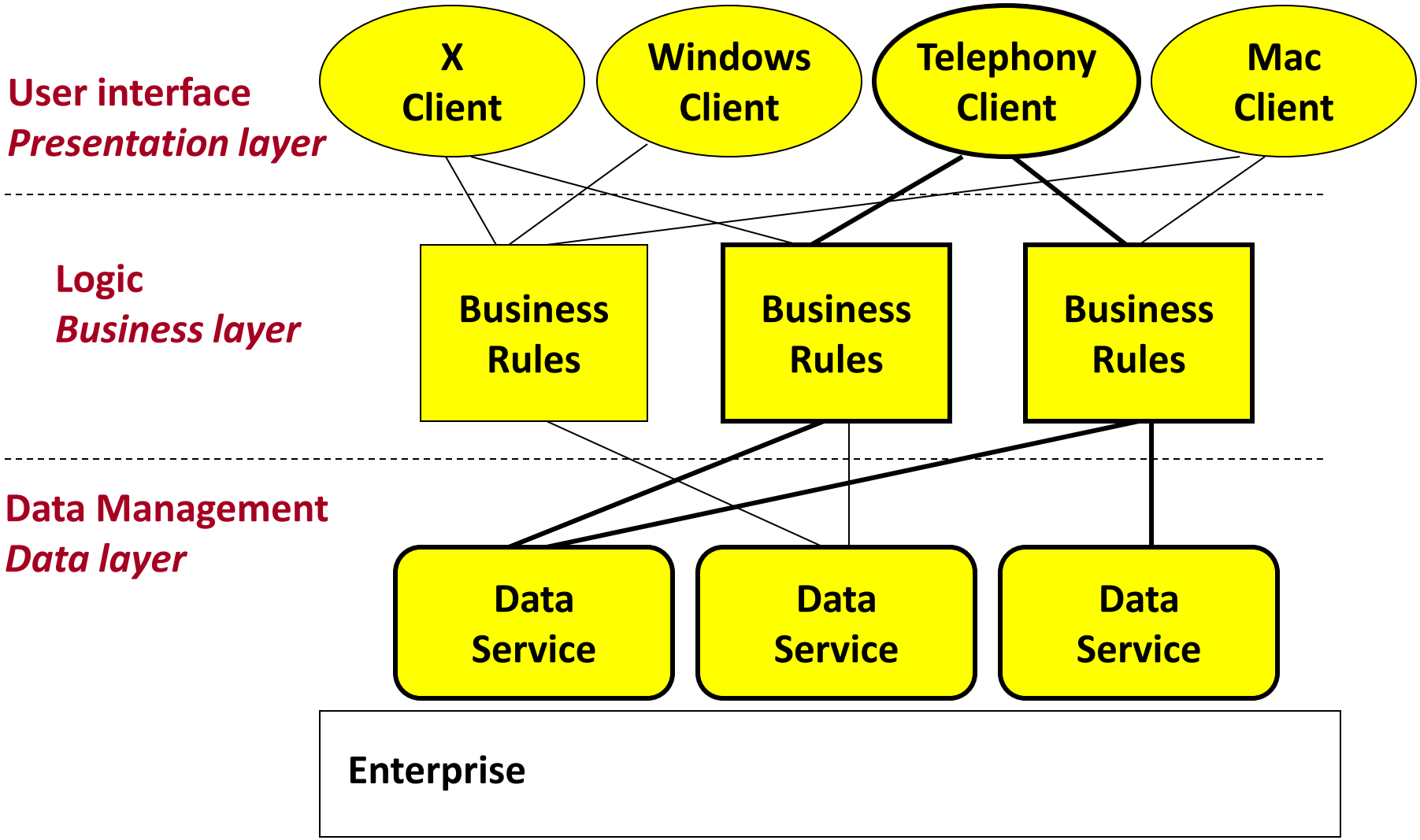
Architetture three-tier - I

- Introdotte all'inizio degli anni '90
- 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 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)

Architetture three-tier - III



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)

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

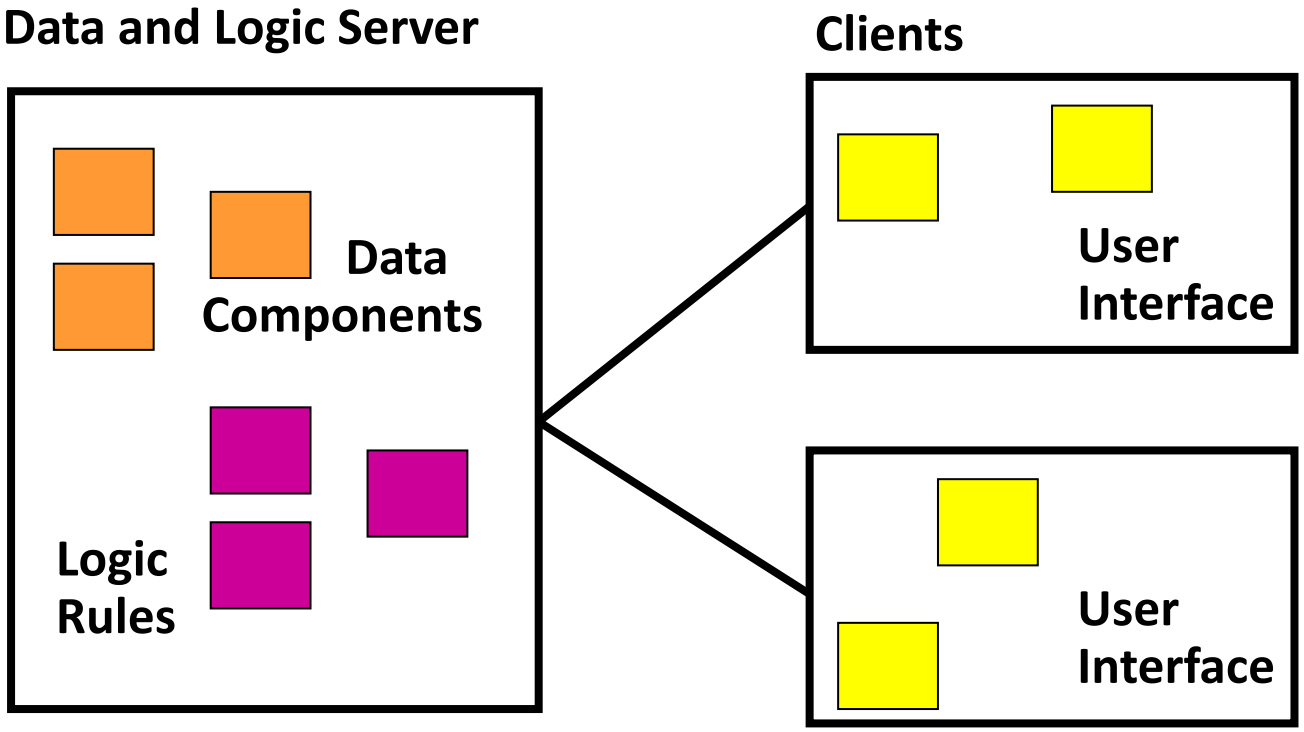
Svantaggi di architetture three-tier

- 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

Three-tier è concettuale, non fisico

Si possono implementare architetture three-tier su due livelli di macchine, o anche su uno solo...

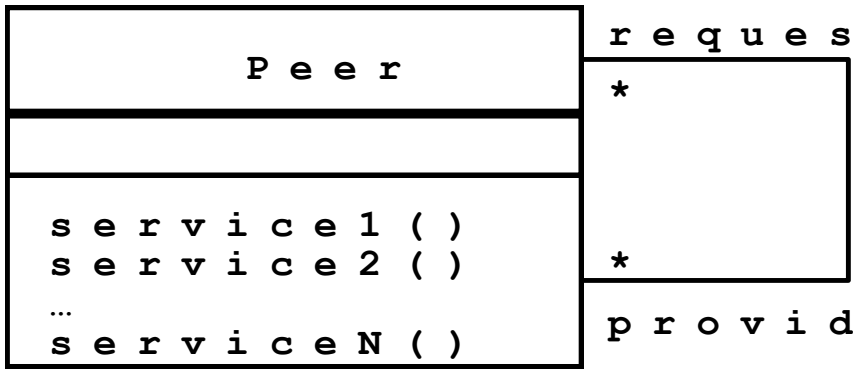


Architetture n-Tier

- Evoluzione delle 3-tier, su N livelli
- Permettono configurazioni diverse.
- Elementi fondamentali:
- Interfaccia utente (UI)
  - gestisce interazione con utente
  - può essere un web browser, WAP minibrowser, 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

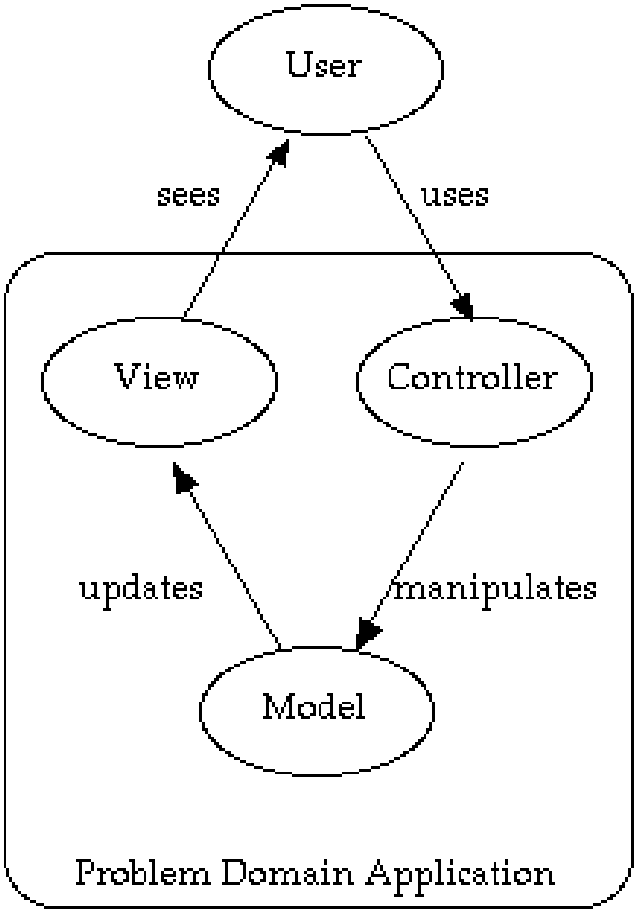
Peer-to-Peer Architecture

- E’ una generalizzazione dell’Architettura Client/Server
- Ogni sottositema può agire sia come Client o come Server, nel senso che ogni sottosistema può richiedere e fornire servizi.
- Il flusso di controllo di ogni sottosistema è indipendente dagli altri, eccetto per la sincronizzazione sulle richieste



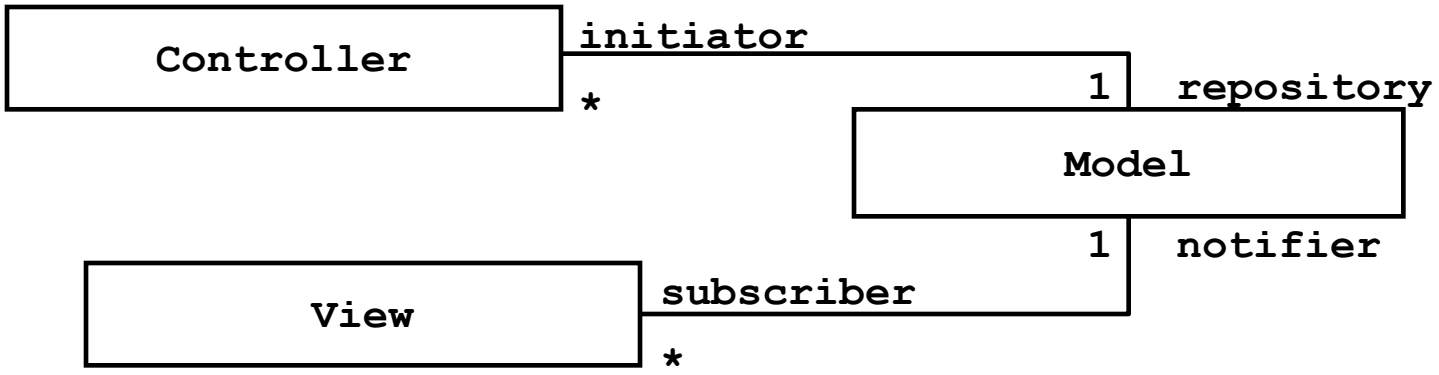
Model/View/Controller

- In questo stile architetturale i sottosistemi sono classificati in 3 tipi differenti:
  - Sottosistema Model: mantiene la conoscenza del dominio di applicazione
  - Sottosistema View (di visualizzazione), visualizza all'utente gli oggetti del dominio dell'applicazione
  - Sottosistema Controller: responsabile della sequenza di interazioni con l'utente
- I sottosistemi Model sono sviluppati in modo che non dipendano da alcun sottosistema View o Controller.
- Cambiamenti nel loro stato sono propagati a sottosistema View attraverso un protocollo "subscribe/notify".

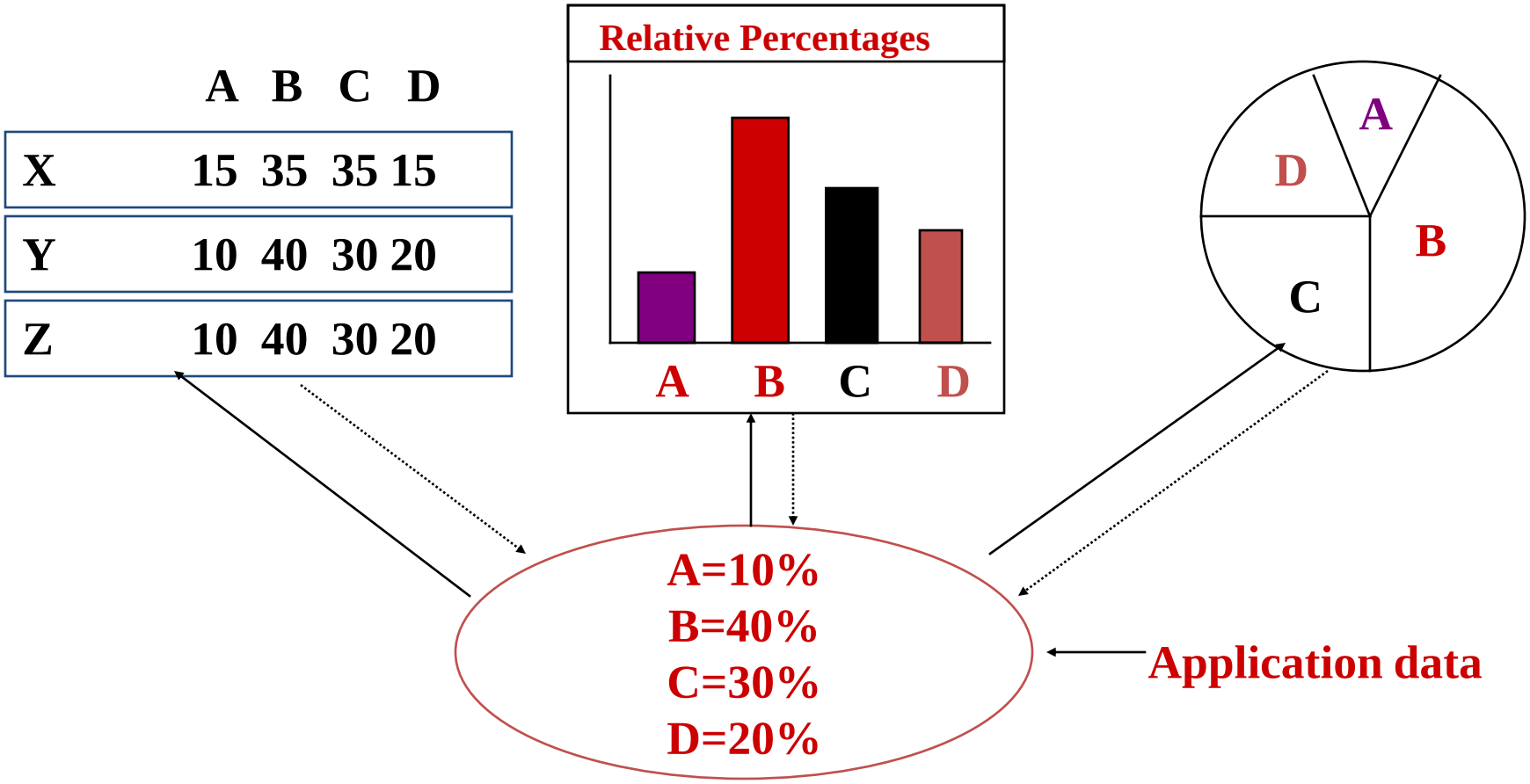


Model/View/Controller (cont)

- MVC è un punto di incontro tra l'architettura di tipo repository e quella a 3 livelli:
  - Il sottosistema Model implementa la struttura dati centrale,
  - il sottosistema Controller gestisce il flusso di controllo: ottiene gli input dall'utente e manda messaggi al modello
  - Il Viewer visualizza il modello

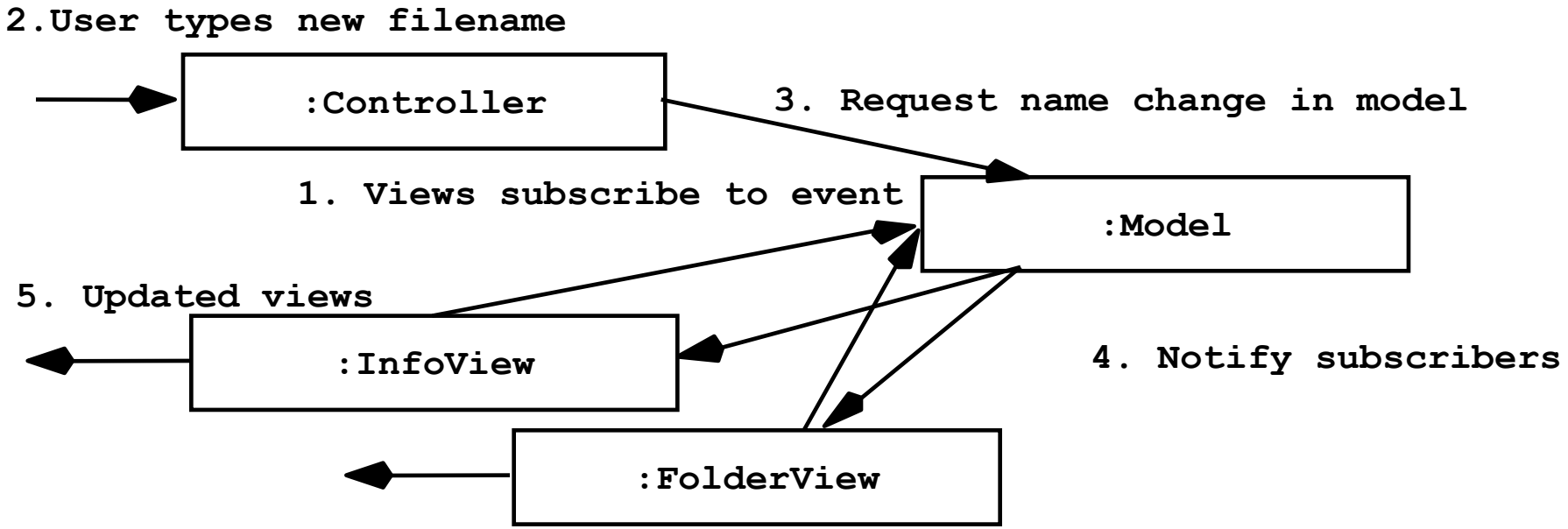


Esempio di View Multiple su un Model



Sequenza di Eventi

Cambiamo il nome del file



1. InfoView e FolderView sottoscrivono i cambi al modello File quando sono creati

2. L'utente digita il nuovo nome del file

3. Il Controller manda la richiesta al modello

4. Il modello cambia il nome del file e notifica i subscriber del cambiamento

5. Entrambi InfoView e FolderView sono aggiornati in modo tale che l'utente veda i cambi in modo consistente

Model/View/Controller: motivazioni

- Il motivo per cui si separano Model, View e Controller è che le interfacce utenti sono soggette a cambiamenti più spesso di quanto avviene per la conoscenza del dominio (il Model)
- MVC è appropriato per i sistemi interattivi, specialmente quando si utilizzano viste multiple dello stesso Model.
- Introduce lo stesso collo di bottiglia visto per lo stile architetturale Repository

Politiche di Controllo

Gestione di eventi asincroni

- Se vogliamo sapere se un'entità esterna (es. una persona) ha fatto un'azione, abbiamo due opzioni:
  - Polling: chiediamo ripetutamente se è successo qualcosa
  - Event-based: diciamo alla persona di avvisarci qualora sia successo qualcosa
- Nella programmazione, per la definizione delle politiche di controllo, abbiamo a disposizione esattamente gli stessi approcci

Stili di controllo

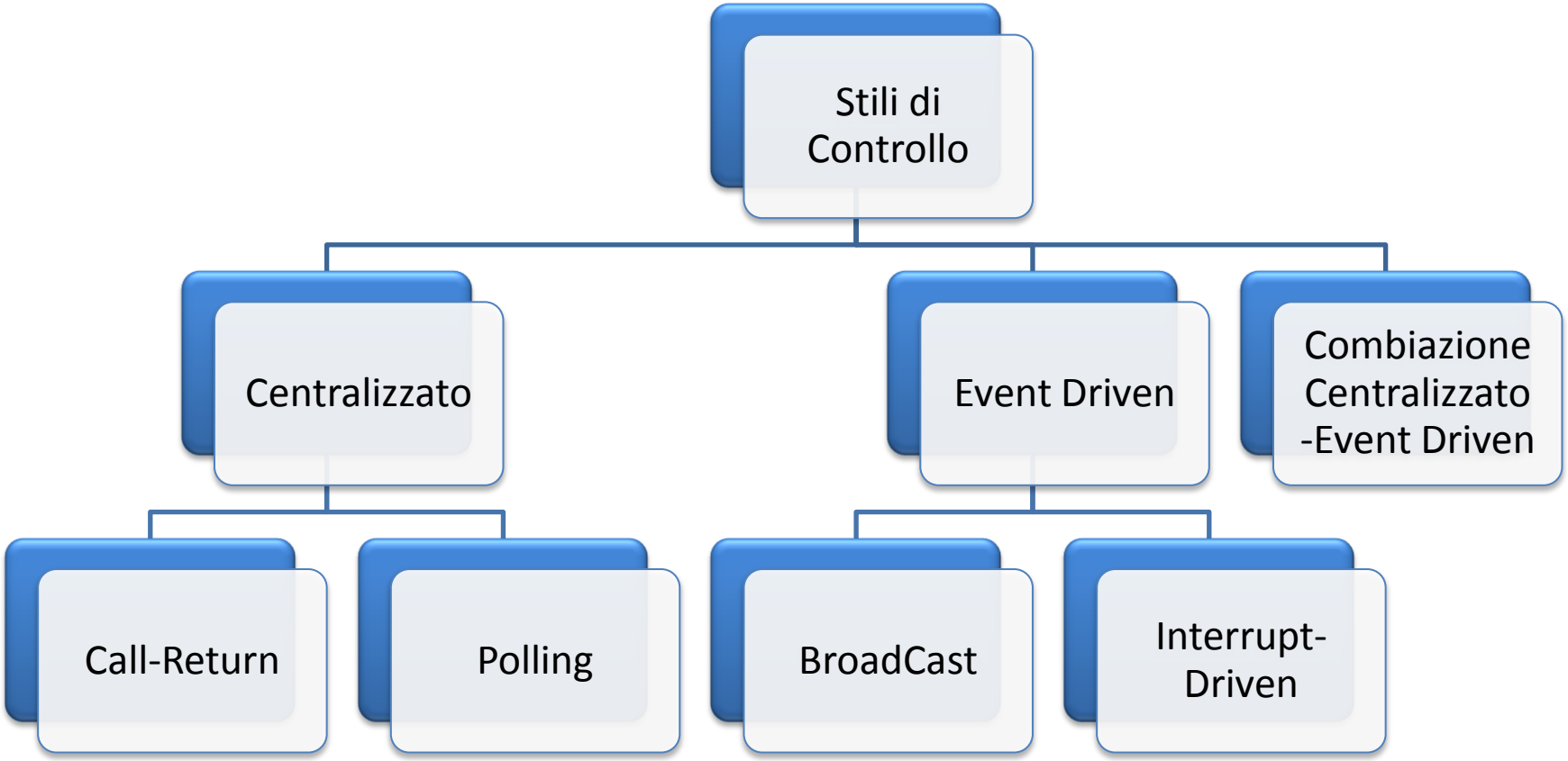
- La maggior parte dei sistemi software sono costituiti da Sottosistemi/ Classi / Metodi che reagiscono a richieste esterne di servizio:
  - Eventi da tastiera/mouse
  - Segnali da sensori
  - Invocazioni da altri moduli
  - …

- Obiettivo della progettazione di architetture riguardo gli stili di controllo è la definizione delle politiche per gestire (smistare) tali input verso i moduli

Stili di controllo

- Politiche di controllo:
  - Controllo centralizzato
    - Un unico sottosistema è responsabile di attivare e interrompere gli altri
  - Controllo basato su eventi
    - Ciascun sottosistema può rispondere a eventi esterni, generati da altri sottosistemi o dall'ambiente
  - Combinazione dei due precedenti
- In linea di principio, tipo di architettura e politica di controllo sono due aspetti complementari

Stili di Controllo

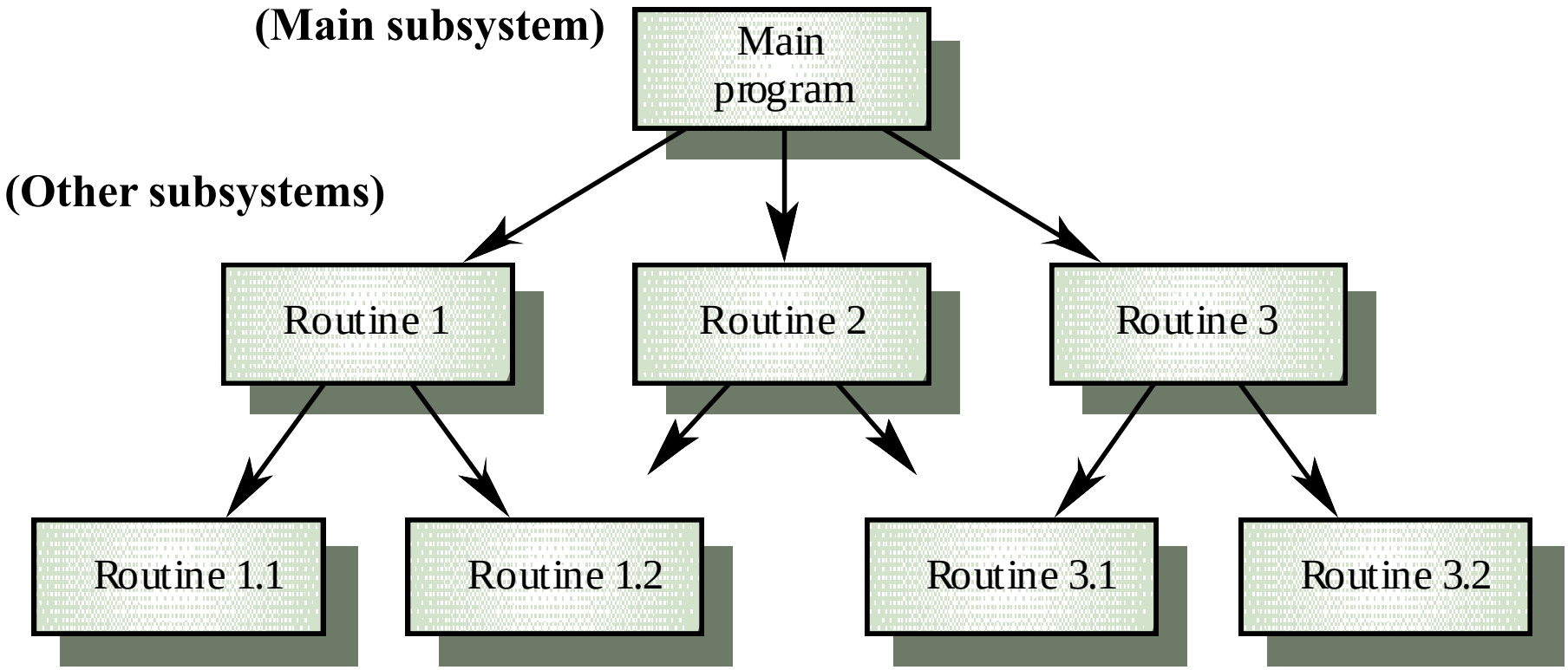


Controllo centralizzato

- Modello Call-return
  - Applicabile a sistemi sequenziali.
  - Il controllo si muove come nell'albero delle chiamate di routine nei programmi sequenziali.
    - E' applicabile solo a sistemi con eventi sincroni sequenziali

- Modello a Polling
  - Applicabile a sistemi concorrenti. Il sottosistema controller (o manager) sovrintende il funzionamento di tutto il sistema, prevedendo l'avvio, l'arresto, il coordinamento degli altri sottosistemi.
    - E' applicabile a qualunque tipo di sistema

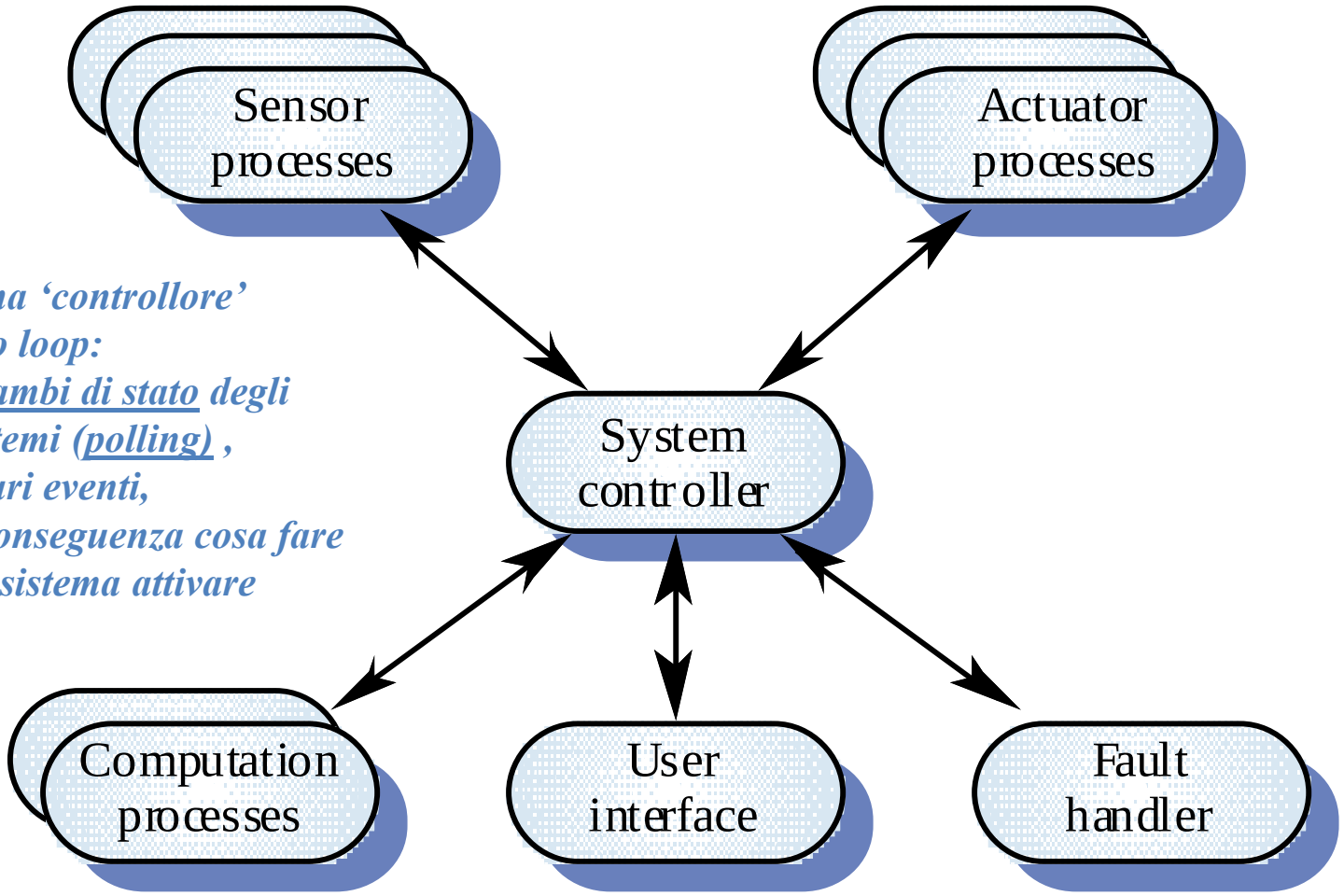
Controllo centralizzato / modello Call-return (sequenziale)



Vantaggio: è facile analizzare il flusso di controllo

Svantaggio: La sequenza di interazioni deve essere definita a priori

Controllo centralizzato / Modello a ‘manager’ (Polling)



Il sottosistema ‘controllore’ è in continuo loop: controlla i cambi di stato degli altri sottosistemi (polling) , causati da vari eventi, e decide di conseguenza cosa fare o quale sottosistema attivare

Controllo basato su eventi (sistemi event-driven)

- Nelle precedenti architetture a controllo centralizzato, le decisioni dipendono dalle variabili di stato, osservate su iniziativa del controllore.
- Al contrario, nei sistemi event driven il controllo è pilotato dagli eventi, generati dall'ambiente, o dagli stessi sottosistemi, a tempi imprevedibili (asincroni).

- Modelli a broadcast.
  - Ogni evento rilevato viene trasmesso a tutti i sottosistemi. Ogni sottosistema in grado di trattare l'evento lo fa.

- Modelli interrupt-driven.
  - Usati per sistemi (soft) real time: interrupt esterni sono rilevati da un interrupt handler e passati al sottosistema appropriato per il trattamento.

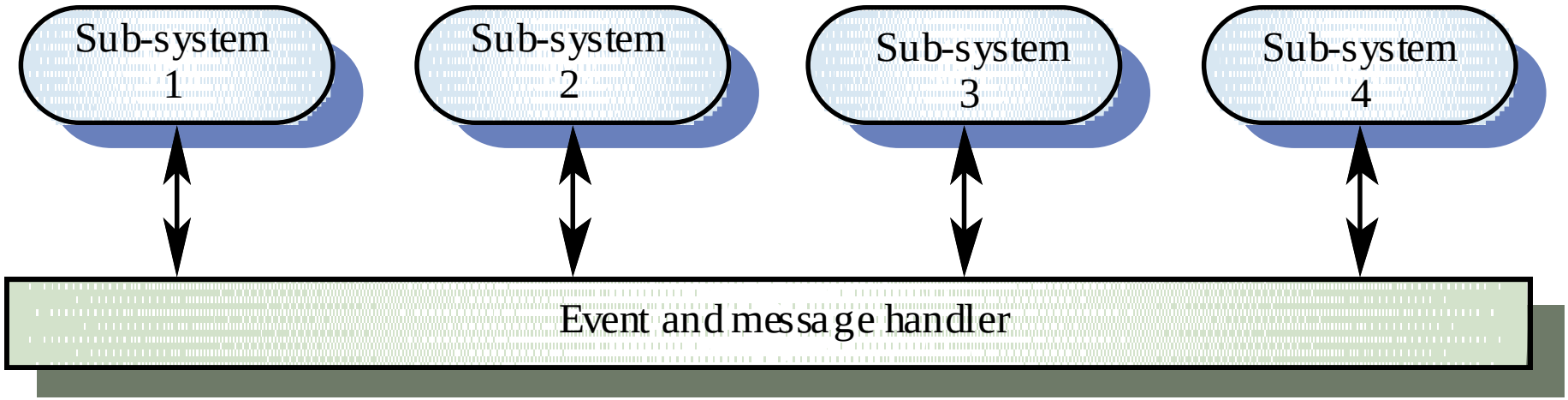
Controllo basato su eventi / modello a broadcast

- I sottosistemi registrano il proprio interesse per determinati eventi.
  - Quando un evento accade (p. es. un sottosistema segnala che ci sono dati pronti per essere elaborati), il gestore di eventi (Event and Message Handler - EMH) lo rileva, controlla il registro, e lo trasferisce, assieme al controllo, ai sottosistemi interessati.

- La politica di controllo non è racchiusa in EMH, come nel controllore centralizzato: i sottosistemi decidono autonomamente quali sono gli eventi di loro interesse.

- EMH può anche gestire la comunicazione fra sottosistemi.

Modello a broadcast



Modello a broadcast: vantaggi e svantaggi

- Vantaggi
  - L'evoluzione è facilitata: ogni nuovo sottosistema aggiunto deve solo informare EMH sui messaggi di proprio interesse
  - I sottosistemi si attivano a vicenda indirettamente, mandando messaggi a EMH, e non devono conoscere i propri indirizzi: la struttura distribuita è trasparente per i sottosistemi.

- Svantaggi
  - I sottosistemi non sanno se e quando i loro messaggi verranno raccolti e gestiti
  - Possibili conflitti se diversi sottosistemi sono interessati agli stessi eventi.

Controllo basato su eventi / modello interrupt-driven

- Politica usata per sistemi real-time in cui risposte rapide agli eventi sono essenziali
- Idea di base:
  - Esistono diversi tipi di interrupt, e un interrupt-handler definito per ciascuno di essi
  - Ciascuno dei tipi di interrupt è associato a una locazione di memoria, dove è memorizzato l'indirizzo del corrispondente handler.
  - Uno switch hardware rapido provoca l' immediato trasferimento del controllo allo handler opportuno, che a sua volta attiva specifici processi.
- Puo' essere combinato con il modello a manager centralizzato
  - Diversi eventi vengono gestiti con l'una o l'altra tecnica (interrupt e polling), a seconda della loro urgenza.