

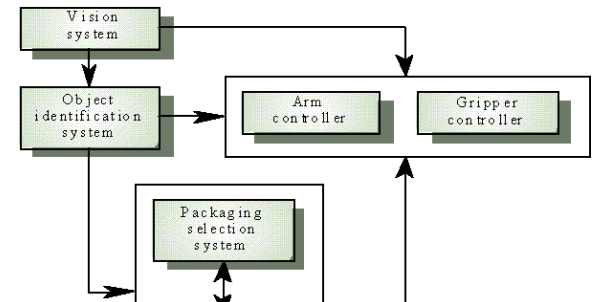
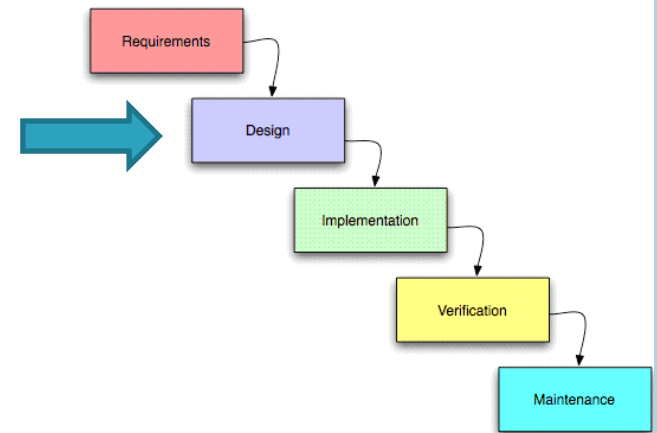


DESIGN ARCHITETTURALE

Ingegneria del Software a.a. 2022-23

AGENDA

- Software Design:
 - Cosa è?
 - Processo creativo?
 - **High level design** vs. **Low level design**
- Design architetturale: Architettura SW
- Concetto di **stile architetturale**
- Alcuni stili architetturali (classici)
 - Pipe and Filters
 - Layered
 - Broadcast model



Warning 1: Non possiamo vederli tutti, serve vostro approfondimento

Warning 2: Non è possibile andare troppo nel dettaglio ...

Warning 3: Parziale Perdita di importanza; uso dei Software Frameworks

DESIGN



- È il processo che:

trasforma un problema in una soluzione!

- **Rispetto ai requisiti:**

- i requisiti definiscono il **problema (cosa)**
- il design propone una **soluzione (come)**

Es. nella costruzione di una casa:

requisiti = richieste dei proprietari-utenti

design = progetto dell'architetto

(schemi e documenti a diverso livello di dettaglio)

- **Rispetto all'implementazione/codifica:**

- il design definisce la **struttura della soluzione**
invece l'implementazione la realizza, rendendola usabile

Es. nella costruzione di una casa:

implementazione = esecuzione dei lavori

(il che include ad esempio la scelta dei materiali usati, ecc.)

LIVELLI DI DESIGN



- Anche il design si sviluppa per raffinamento ...

- **Architectural design**

mappa i requisiti su architettura SW e componenti/sottosistemi

anche system design o high-level design

- **Component design**

fissa dettagli dei componenti, specificando maggiormente la soluzione

anche subsystem design o low-level design

- Scelte tecnologiche

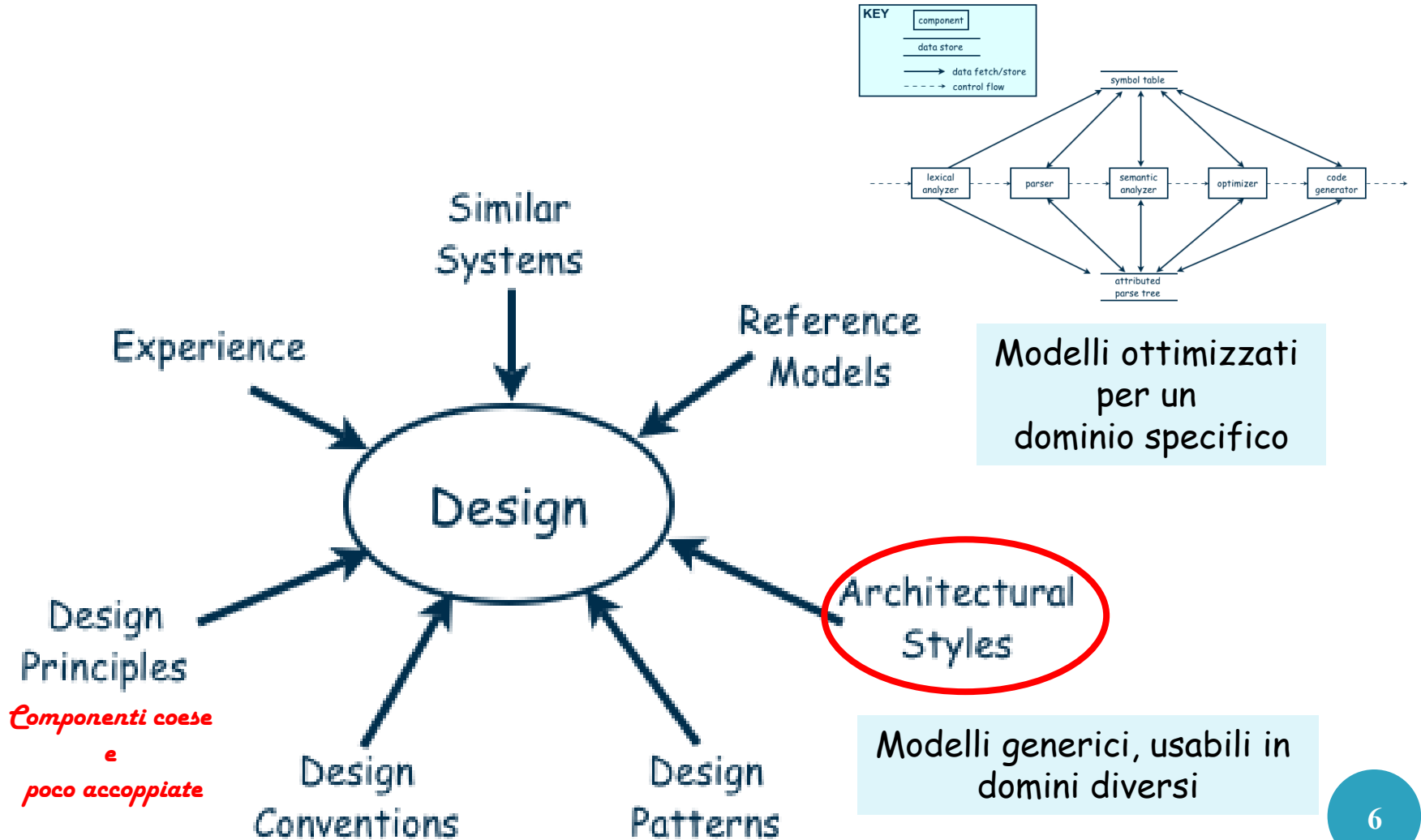
- **platform-independent design**

- **platform-specific design (es., .Net o J2EE)**

DESIGN: UN PROCESSO CREATIVO?

- Il design può essere migliorato studiando **esempi** di buon design
- La maggior parte del lavoro di design è lavoro *di routine*
 - i problemi vengono risolti **riutilizzando ed adattando soluzioni a problemi simili (ri-uso)**
- Tre diversi livelli di **ri-uso**:
 - **Clonazione**: Si riutilizza interamente design/codice, con piccoli aggiustamenti
 - **Design pattern**: Buona soluzione a problema ricorrente
 - Li vedremo prossimamente ...
 - **Stili architettureali**: Architettura generica che suggerisce come decomporre il Sistema
 - Argomento di oggi

DESIGN: UN PROCESSO CREATIVO?



DESIGN ARCHITETTURALE

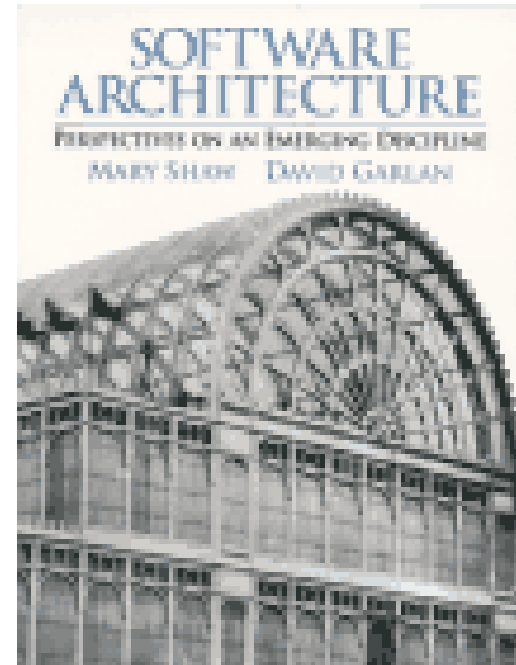
○ Design architetturale:

processo di design per identificare:

- le **(macro) componenti** di un sistema
 - Non siamo a livello di classi, strutture dati e algoritmi!
- il framework per il controllo e la **comunicazione tra componenti**

○ Produce una descrizione dell'**architettura software**

- per analogia: corrisponde al progetto architettuale di una casa
- concetto introdotto da Mary Shaw e David Garlan nel 1996



COMPONENTI: SOTTOSISTEMI E MODULI

Un **sottosistema** è un sistema di per sè: può essere eseguito ed utilizzato anche “da solo”. Di solito i sottosistemi sono composti da moduli ed hanno interfacce ben definite, che sono utilizzate per la comunicazione con altri sottosistemi

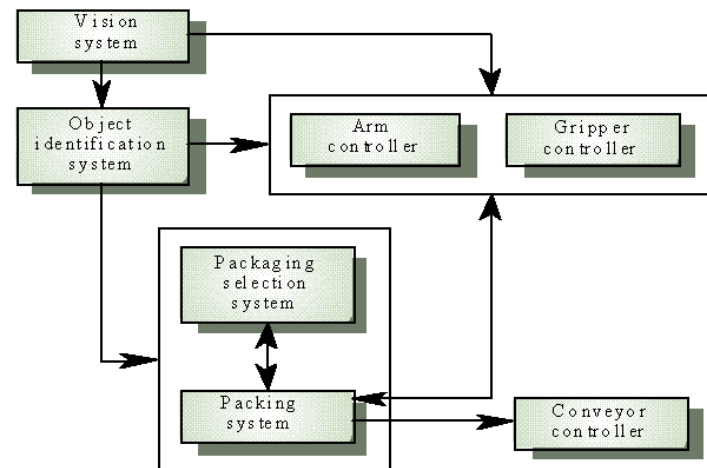
Un **modulo** è un'unità del sistema che offre servizi ad altre unità, ma che non può essere considerato un sistema a se stante

DIAGRAMMA A BLOCCHI

- Un architettura software è normalmente espressa mediante un **diagramma a blocchi** che presenta un “overview” della struttura del sistema
- I **blocchi** sono i componenti
 - Sottosistemi o moduli
- I **connettori** rappresentano le “relazioni” tra i componenti
 - Es. A e B comunicano scambiandosi dei messaggi

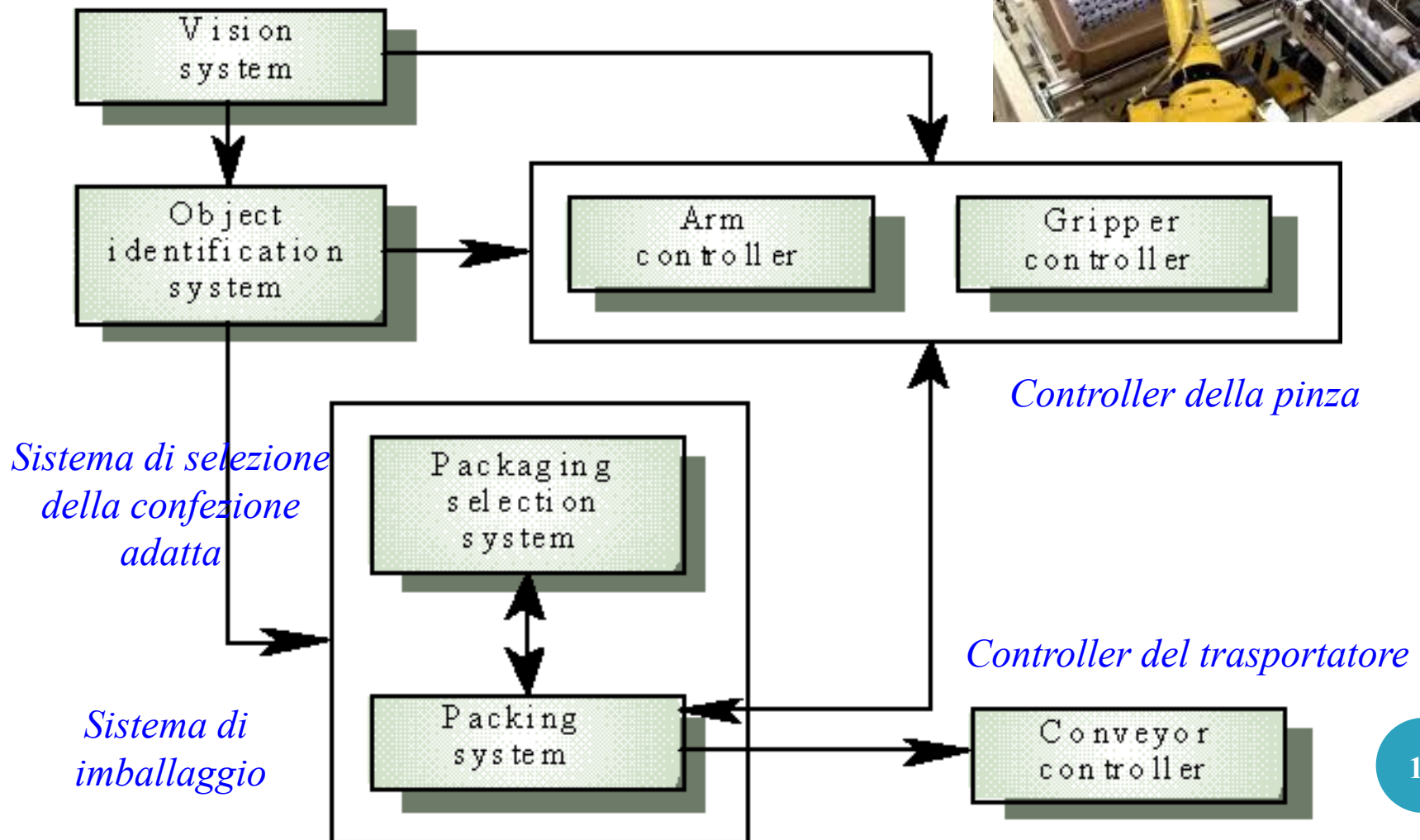
Esistono anche linguaggi formali di specifica architetturale, es. Darwin, ADLs

UML fornisce il diagramma delle componenti!!!



ARCHITETTURA SW: ESEMPIO

Sistema di controllo di imballaggio robotizzato



ARCHITETTURA SW: PROPRIETÀ DEL SISTEMA

Grosso impatto dell'architettura su:

- **Performance** (tempi di risposta rapidi): Minimizzare le comunicazioni
 - Componenti **large-grain** piuttosto che **fine-grain**
- **Security** (difficile da manomettere): Usare un'architettura a livelli con le risorse critiche nei livelli più interni
- **Safety** (non creare disastri): Localizzare le caratteristiche safety-critical in pochi sottosistemi
- **Availability** (24/7/365): Includere componenti ridondanti e meccanismi per '**fault tolerance**'
- **Maintainability** (facile da evolvere): Usare componenti **fine-grain** facilmente rimpiazzabili

ARCHITETTURA SW: PROPRIETÀ DEL SISTEMA

Grosso impatto dell'architettura su:

- **Performance** (tempi di risposta rapidi): Minimizzare le comunicazioni
 - Componenti **large-grain** piuttosto che **fine-grain**
- **Security** (difficile da implementare): Usare un'architettura a livelli con risorse critiche nei livelli più interni
- **Safety** (evitare disastri): Localizzare le caratteristiche safety-critical in pochi sottosistemi
- **Availability** (24/7/365): Includere componenti ridondanti e meccanismi per '**fault tolerance**'
- **Maintainability** (facile da evolvere): Usare componenti **fine-grain** facilmente rimpiazzabili

ARCHITETTURA SW: PROPRIETÀ DEL SISTEMA

Grosso impatto dell'architettura su:

- **Performance** (es. tempo di esecuzione). Minimizzare le comunicazioni
 - Componenti **large-grain** piuttosto che **fine-grain**
- **Security**: Usare un'architettura a livelli con le risorse critiche nei livelli più interni
- **Safety**: Localizzare le caratteristiche safety-

Trade-off:

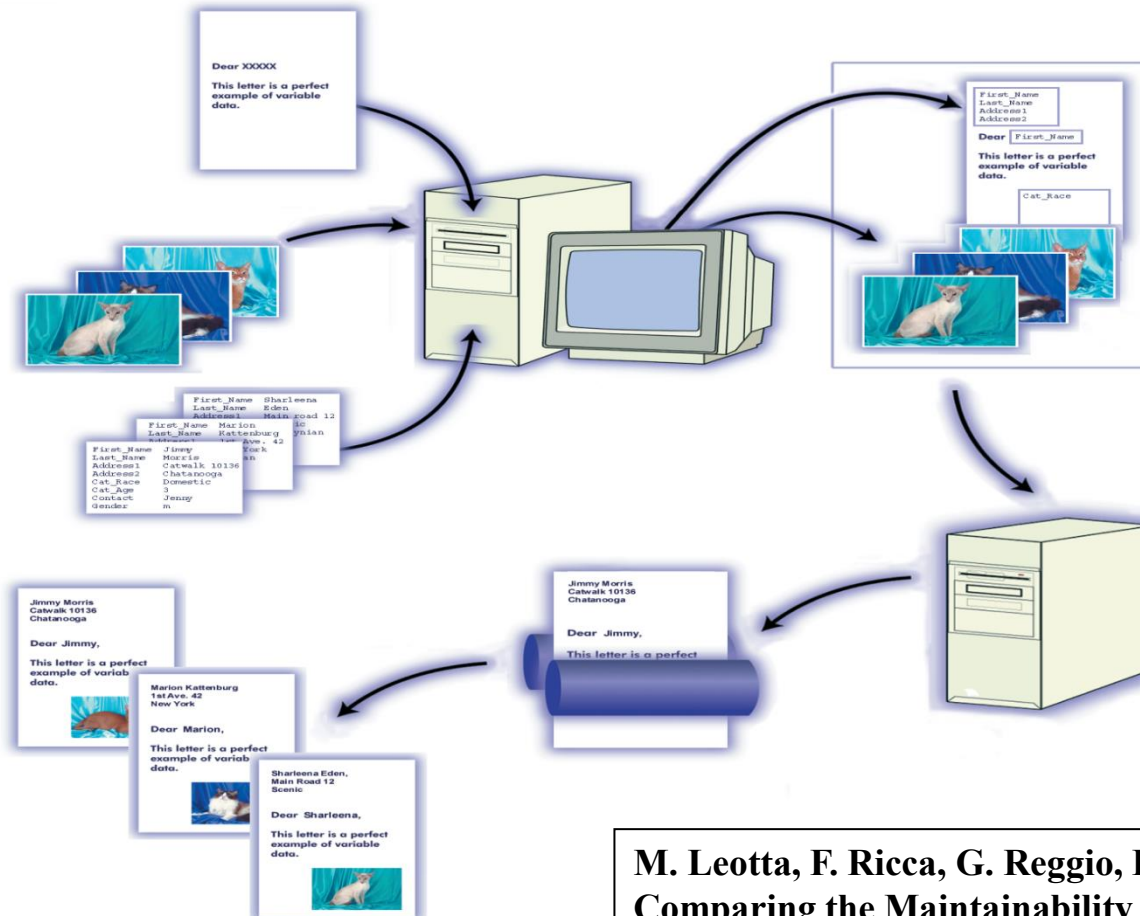
- Componenti **large-grain** migliorano Performance ma riducono Maintainability
- Componenti **ridondanti** migliorano Availability ma peggiorano Safety ...
- Usare un'architettura a livelli migliora Security ma aumenta comunicazioni e così degrada Performance ...

VANTAGGI NELL' AVERE UN' ARCHITETTURA SW

- Guida lo sviluppo ed aiuta nella comprensione del sistema
 - la realizzazione di più componenti distinti è meno complessa della realizzazione di un sistema come monolito
- Documenta il sistema
 - Serve a capire le dipendenze tra componenti
 - Elemento fondamentale della documentazione del sistema
- Aiuta a ragionare sull'evoluzione del sistema
- Supporta decisioni manageriali
 - ad esempio se rifattorizzare il sistema o no
- Facilita l'analisi di alcune proprietà
 - alcune proprietà sono soddisfatte?
 - Es. manutenibilità, efficienza
- Permette il riuso (Large-scale)
 - l'architettura stessa può essere riutilizzabile in più sistemi

IL SISTEMA “POSTEL MILLENIUM” (ELSAG-DATAMAT)

- Trasformazione di “corrispondenza” elettronica in fisica



M. Leotta, F. Ricca, G. Reggio, E. Astesiano:
Comparing the Maintainability of Two Alternative
Architectures of a Postal System: SOA vs. Non-SOA.
CSMR 2011: 317-320

STILE ARCHITETTURALE

- L'architettura di un sistema può **conformarsi** a uno **stile architettuale**
 - Modello generico con caratteristiche specifiche che può essere istanziato/personalizzato

Casa walser valsesiana



Nell'es. della progettazione della casa:
loft, **casa walser** valsesiana, appartamento
con ingresso “alla genovese” ...

- **Conoscere gli stili architettureali può semplificare il problema di definire l'architettura software**
 - In realtà, la maggioranza dei grandi sistemi sono **eterogenei** e non seguono un singolo stile architettureale

ELEMENTI DI UNO STILE ARCHITETTURALE

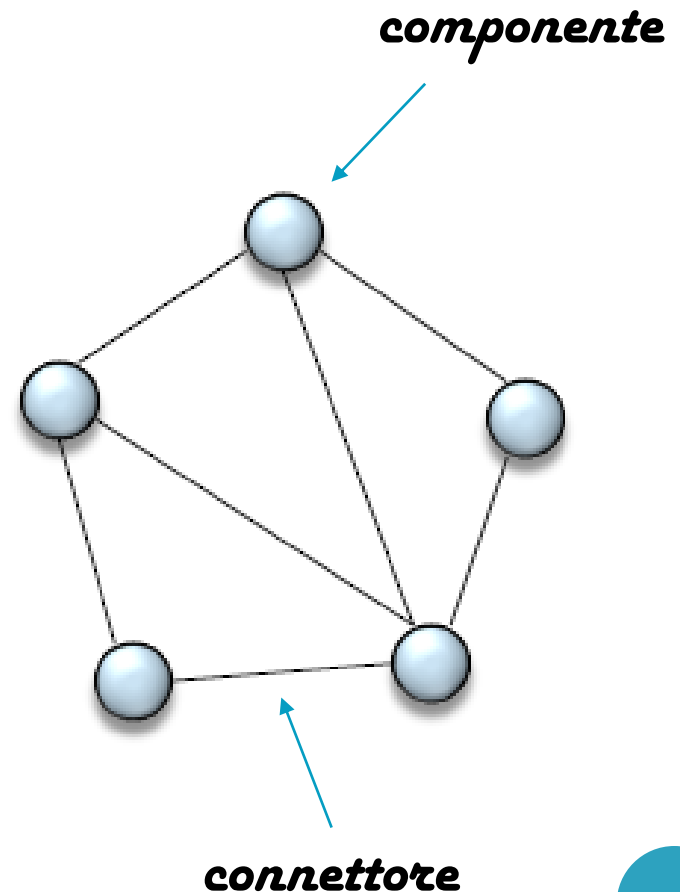
- Uno stile architetturale è definito da

- **tipo di componenti** architetture di base

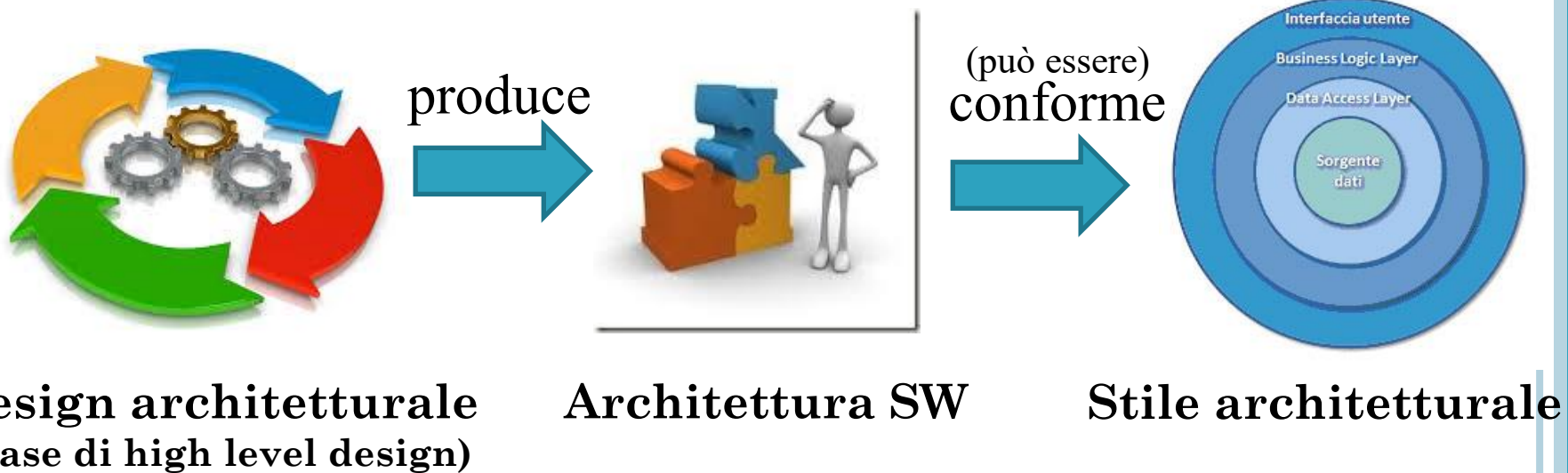
- es. procedure, filtri, livelli/strati, servizi (web service), micro-servizi, package di classi, sotto-sistemi

- **tipo di connettori**

- es. invocazioni di procedure/metodi (RPC), pipe, eventi, scambio di messaggi



RICAPITOLANDO ...



STILI ARCHITETTURALI

Warning:
non li vedremo tutti!
(vedere il video)

○ Diversi stili:

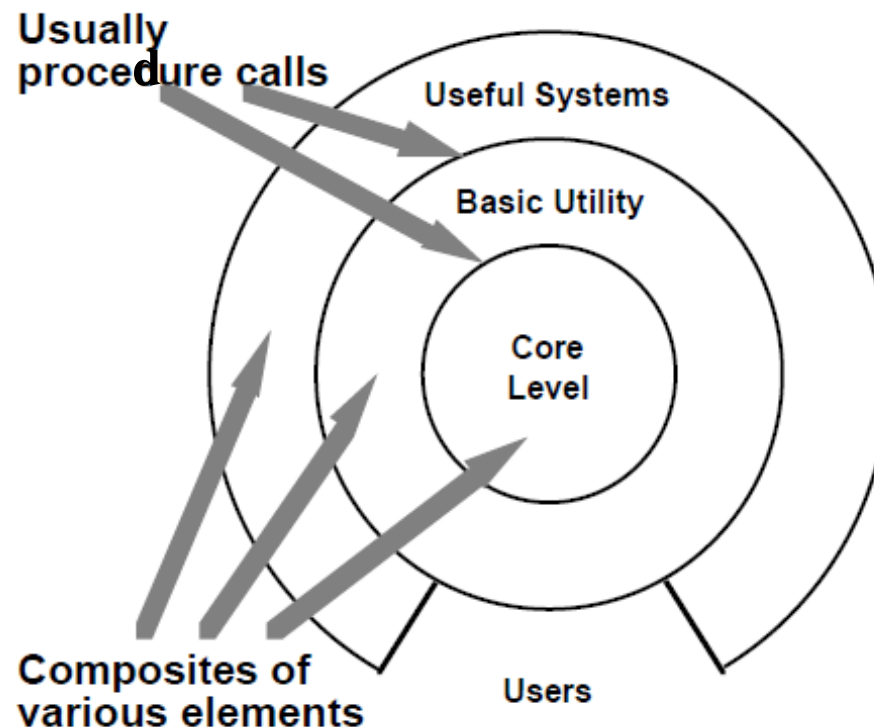
- Layered (Stratificato)
- Repository
- Client/server
 - Two tier / three tier
- P2P
- Pipe and Filter
- Broadcast Model
- Service Oriented Architecture
- **Microservice**
- ...

○ Stile strutturale vs Stile di controllo

- **Strutturale**: solo info strutturali
- **Di controllo**: anche info (o solo) di controllo
 - Il modulo 'xyz' controlla 'xx' e 'zz'?

STRATIFICATO (LAYERED)

- Organizza il sistema in un **insieme di livelli** ognuno dei quali fornisce un insieme di servizi
- Un livello “si appoggia” solo sul livello inferiore
 - Cioè usa solo i servizi del livello inferiore



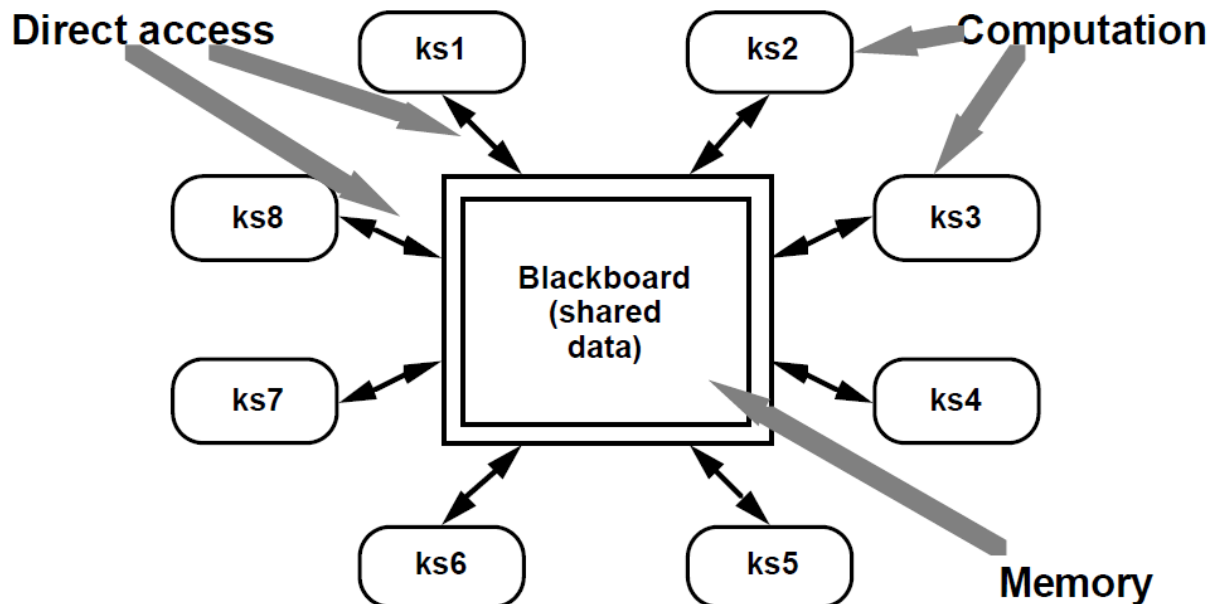


ESEMPIO: ANDROID



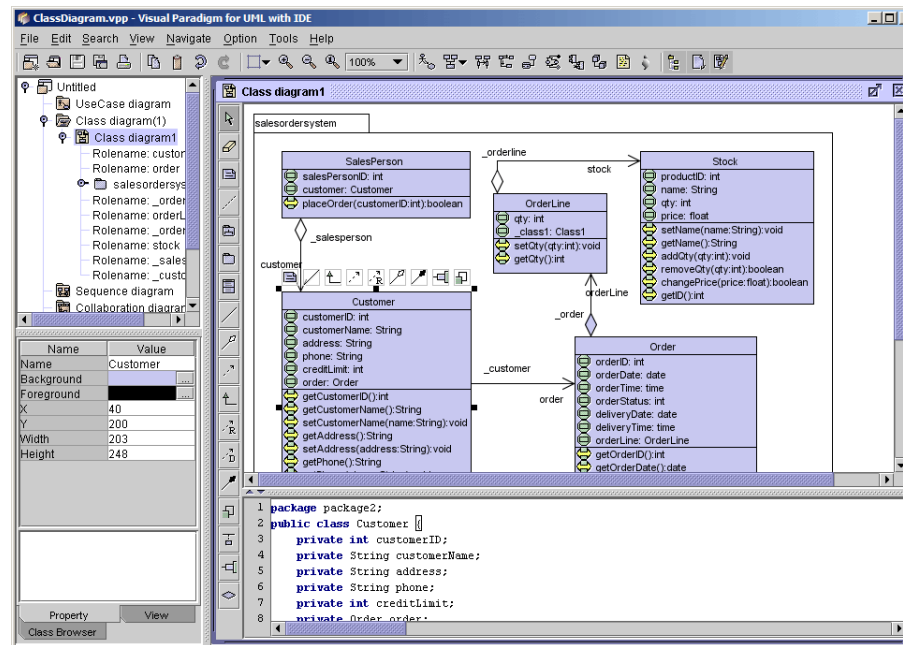
REPOSITORY MODEL

- I **dati condivisi** sono mantenuti in un **database centrale (repository o blackboard)** a cui hanno accesso tutti i sotto-sistemi
 - No comunicazione diretta tra sotto-sistemi!
- Adatto per applicazioni in cui i dati sono prodotti da un sottosistema e utilizzati da un altro

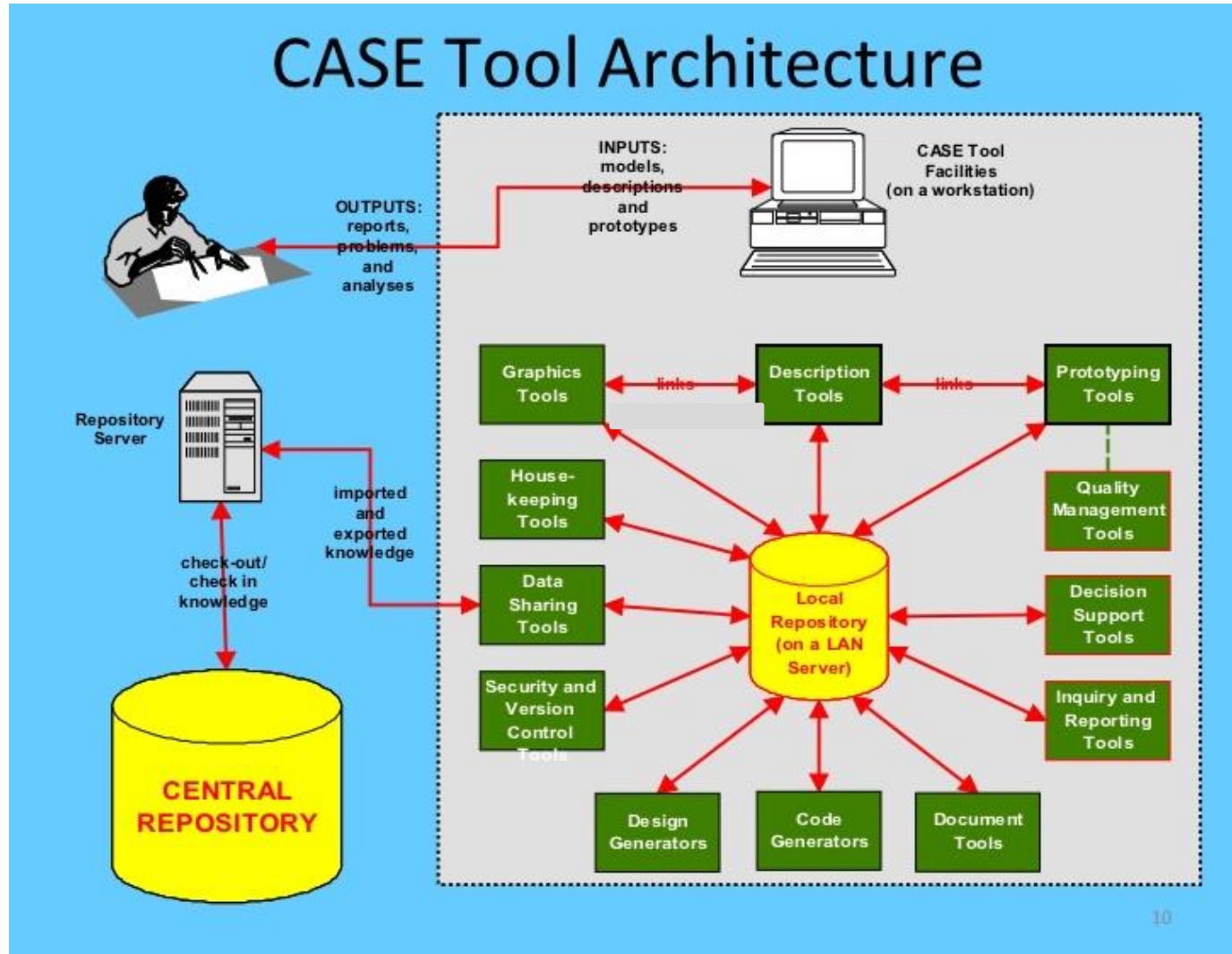


ESEMPIO: CASE TOOL (1)

- I CASE sono tool che supportano lo **sviluppo** e **manutenzione** del software
 - Non solo fase di codifica ma spesso anche design e requisiti



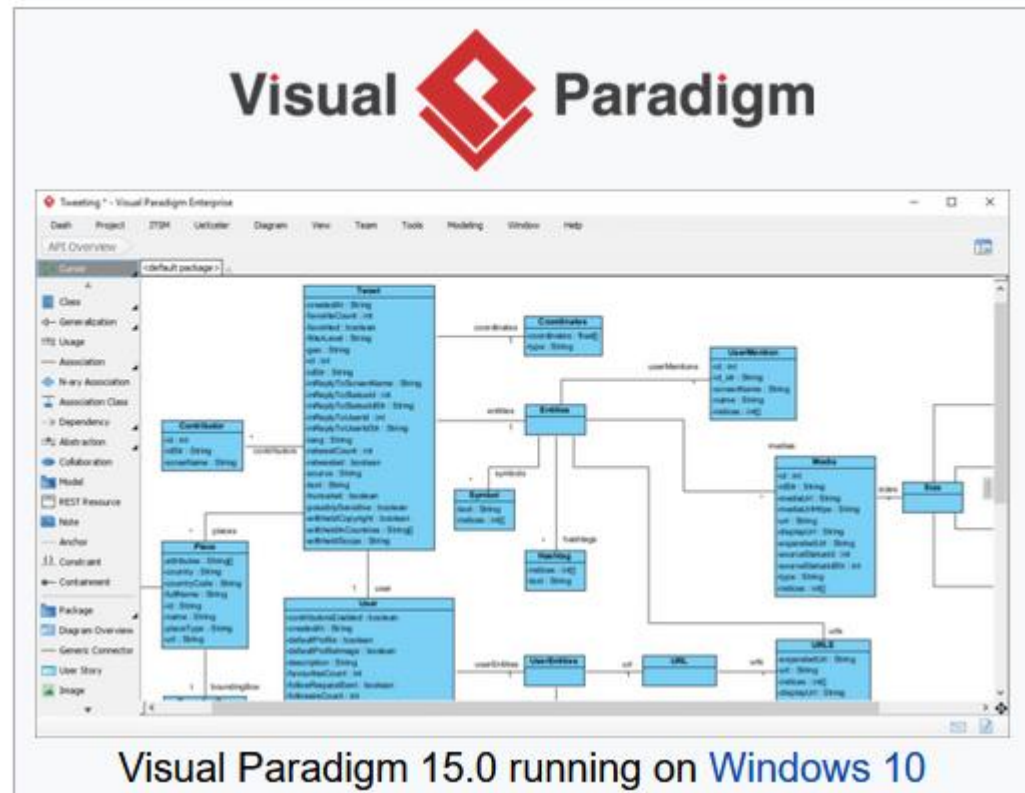
ESEMPIO: CASE TOOL (2)



VISUAL PARADIGM

Visual Paradigm (VP-UML) is a **UML CASE** Tool supporting UML 2, **SysML** and **Business Process Modeling Notation (BPMN)** from the **Object Management Group (OMG)**. In addition to the modeling support, it provides report generation and code engineering capabilities including **code generation**. It can **reverse engineer** diagrams from code, and provide **round-trip engineering** for various programming languages.

Visual Paradigm



Visual Paradigm 15.0 running on Windows 10



REPOSITORY MODEL: PRO E CONTRO



○ Vantaggi

- Modo efficiente di condividere grandi quantità di dati (non sono trasmessi esplicitamente tra sottosistemi)
- Gestione centralizzata di backup, security, etc.

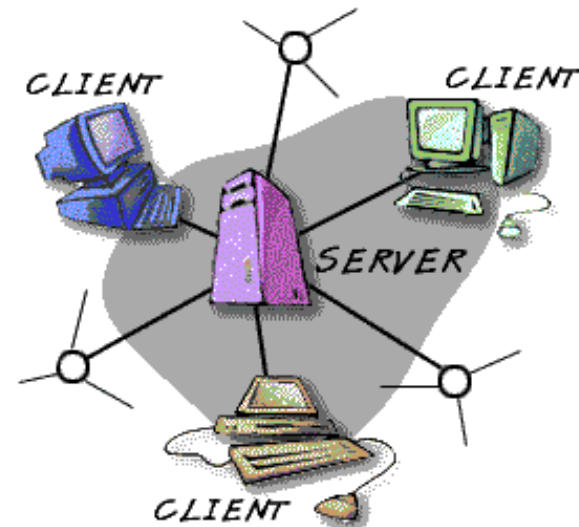


○ Svantaggi

- I sottosistemi devono accordarsi su un modello dei dati per il repository, necessità di compromesso
 - Modello dei dati (schema) è unico
- L'evoluzione dello schema dei dati è difficile e costosa
- Difficile da rendere distribuito in maniera efficiente

CLIENT-SERVER MODEL

- Modello di **sistema distribuito** che mostra come i **dati** e la **computazione** possono essere distribuiti su:
 - insieme di **server** che **forniscono servizi** specifici
 - Es. servizi di stampa, servizi di gestione file,
 - insieme di **client che utilizzano tali servizi**
- Esiste una **rete** che permette ai client di accedere ai server



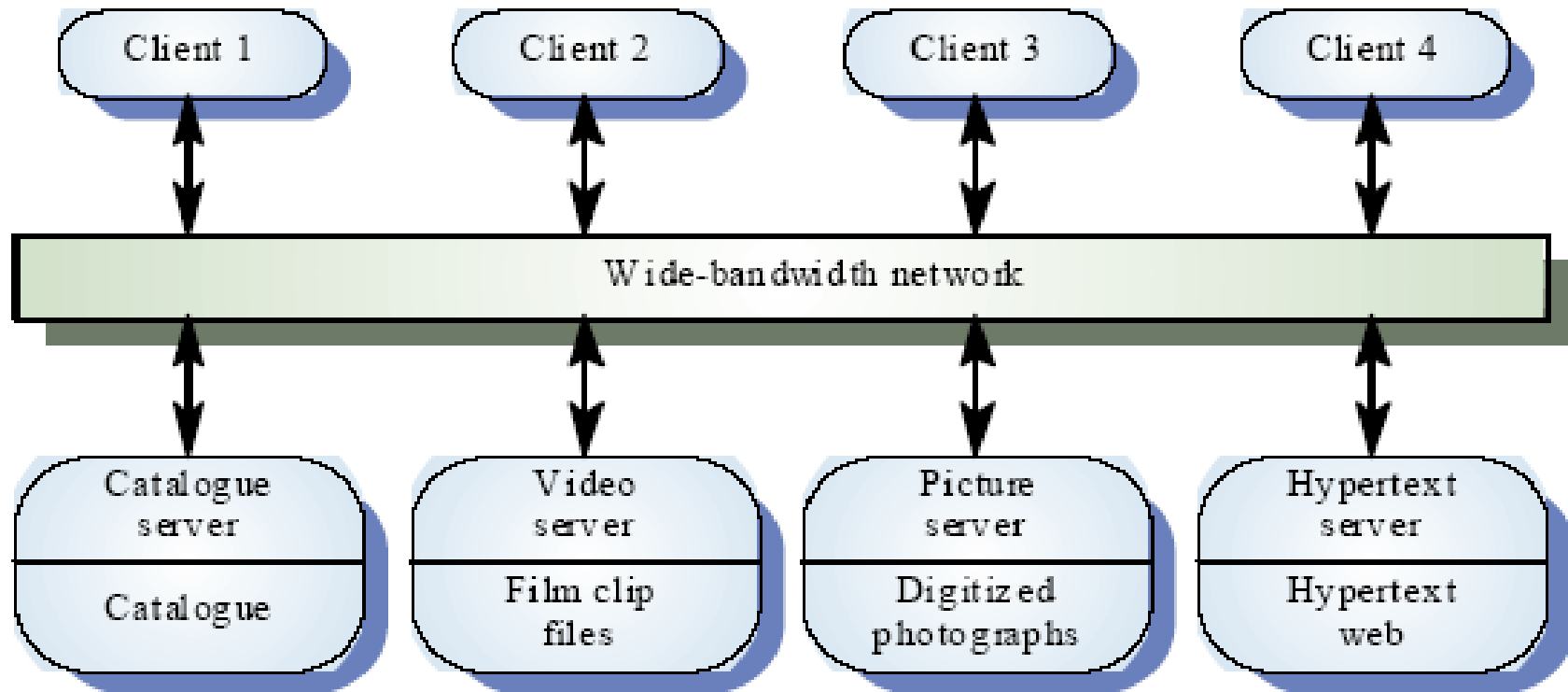
Protocollo 'request and response'

ESEMPIO: WEB APP PER FILM E FOTOGRAFIE



Interfaccia utente costruita usando **Web browser**

MovieStillsDB



Il server catalogo gestisce il catalogo e vendita prodotti

Il server video gestisce compressione e decompressione (bassa risoluzione)

Le immagini sono a alta risoluzione per cui ha senso gestirle su un altro server

CLIENT SERVER A DUE LIVELLI (TWO-TIER)

*Principio di buon design
(vedere il video)*

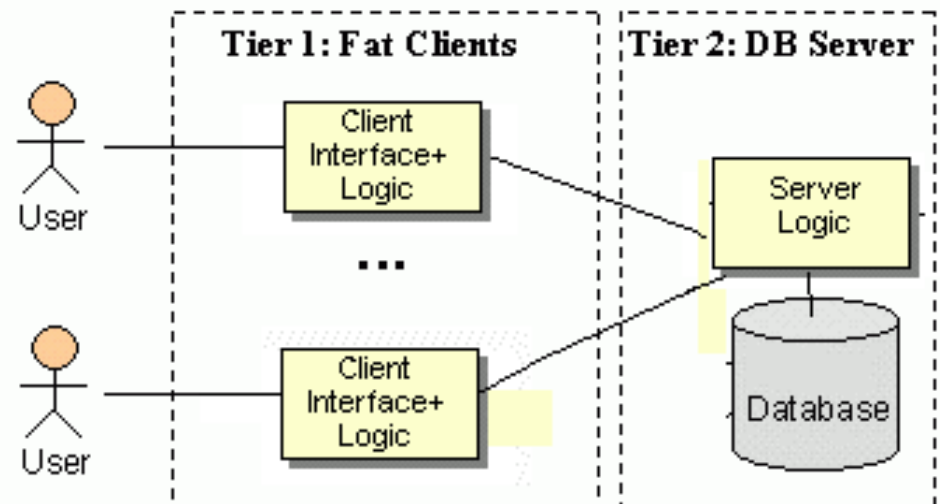
Tre strati/componenti software:

- **Interfaccia utente (presentation logic)**
- **Gestione dei processi e logica (business o appl. logic)**
- **Gestione del DB (data logic)**

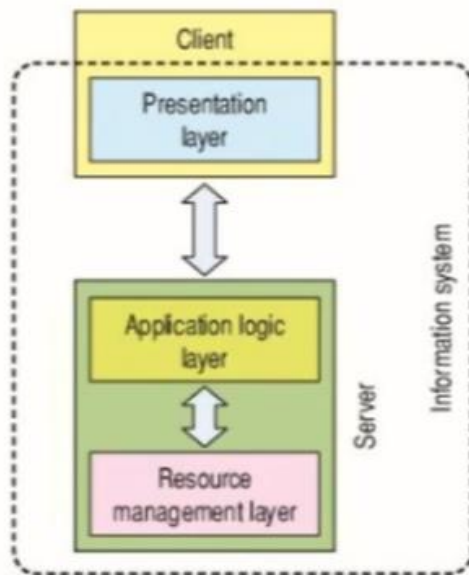
distribuiti in due livelli ('n' client + 'm' server):

- **Client** (chi richiede il servizio)
- **Server** (chi fornisce il servizio)

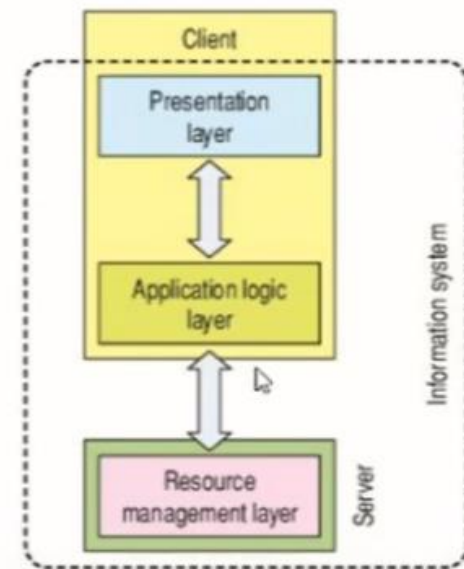
Thin-client vs Fat-client



THIN CLIENT VS. THICK(FAT) CLIENT



Thin client: capacità elaborativa concentrata sul server



Thick client: capacità elaborativa concentrata sul client

CLIENT-SERVER A TRE LIVELLI (THREE-TIER)

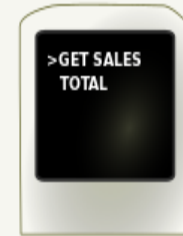
Evoluzione del Two-Tier

- sempre client-server
- sul client resta solo l'interfaccia utente
- la logica del sistema è uno strato separato che gestisce **multi-utenti**
- gli strati di logica e gestione DB sono distribuiti su più server

Client

Presentation tier

The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.



Dammi il totale delle vendite

Servers

Logic tier

This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.



GET LIST OF ALL SALES MADE LAST YEAR



ADD ALL SALES TOGETHER

Servers

Data tier

Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.



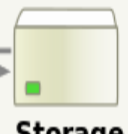
QUERY



SALE 1
SALE 2
SALE 3
SALE 4



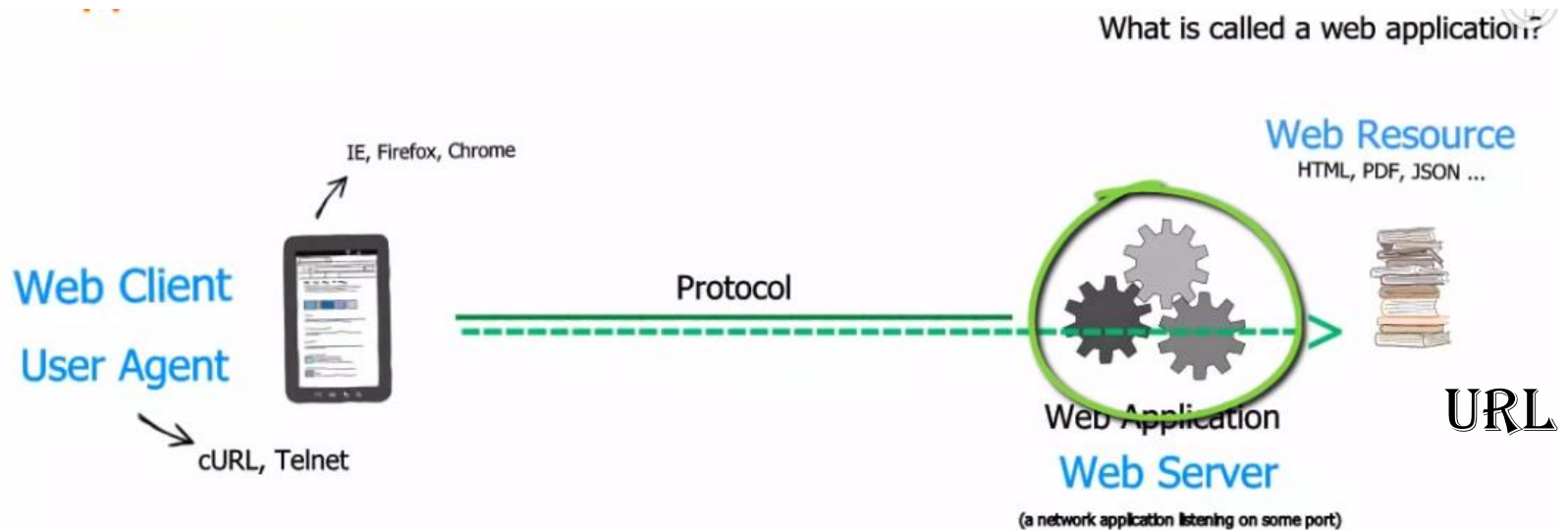
Database



Storage

Application server e DB server

APPLICAZIONI WEB



Applicazioni distribuite accessibili/fruibili via Web cioè in una architettura tipica di tipo client-server

HTTP *protocol = request-response protocol*

PEER-TO-PEER (P2P) MODEL

- Ogni componente esegue il suo processo e gioca allo **stesso tempo** il **ruolo di client e server**
- Ogni peer ha un interfaccia che specifica i servizi offerti (server) e quelli richiesti (client)
- Peer comunicano richiedendo servizi ed offrendoli
 - Simile a client-server, ma qui i ruoli non sono definiti
- **Cosa distingue un peer da un altro sono i dati!**
 - Se un peer ha bisogno di un dato lo chiede ad un altro peer
- I sistemi P2P **scalano bene** e sono **fault-tolerant**
 - Aggiungere un “peer” vuol dire aggiungere nuove capabilities (in termini di nuovi dati)
 - I dati sono replicati su diversi peer per cui se un nodo viene sconnesso per una qualche ragione non si perde (molta) informazione

ESEMPIO: eMule



Altri: BitTorrent e Vuze

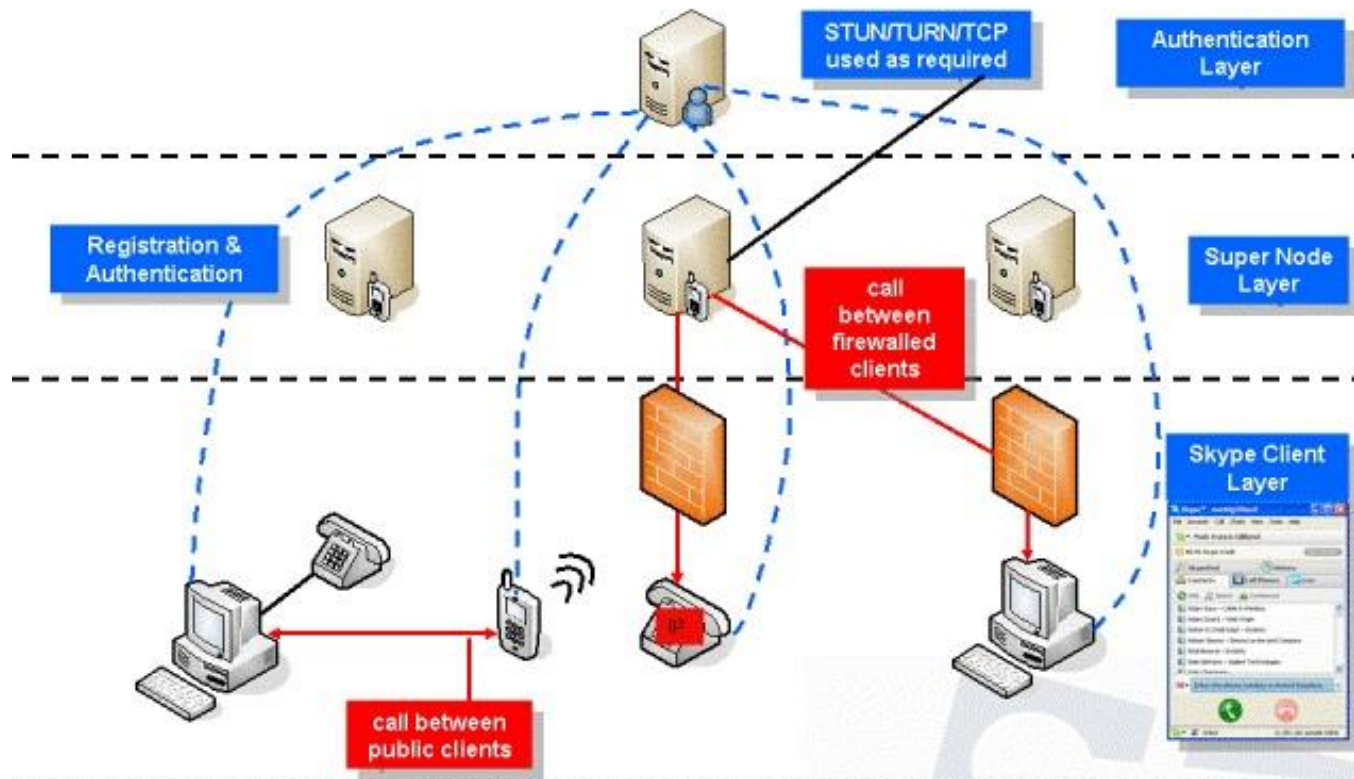


eMule è un software applicativo open source dedicato alla **condivisione file** appoggiandosi su una rete peer to peer e scritto in linguaggio di programmazione C++ per il sistema operativo Microsoft Windows ...

ESEMPIO: SKYPE



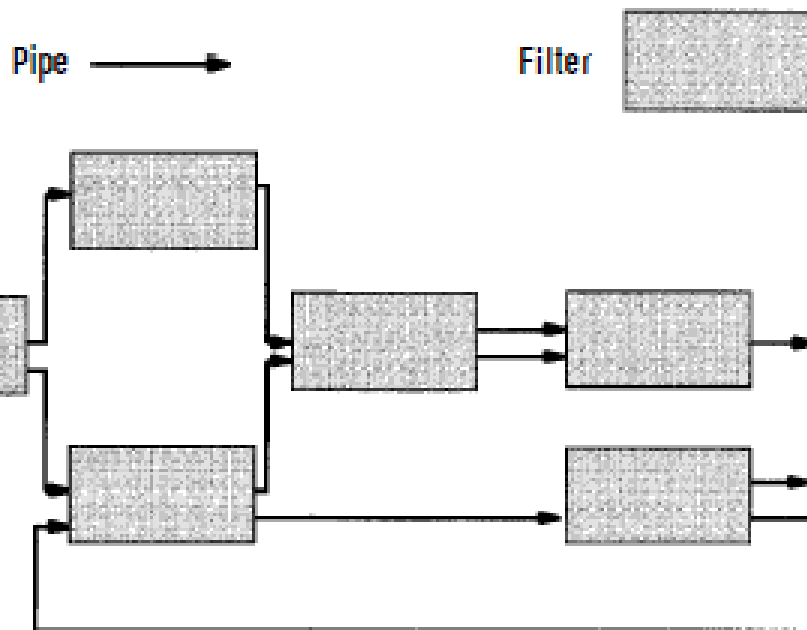
Skype Architecture



Peers

PIPE AND FILTER

- I **filtri** effettuano **trasformazioni** che elaborano i loro input per produrre output
- Le **pipe** sono **connettori** che trasmettono i dati tra filtro e filtro

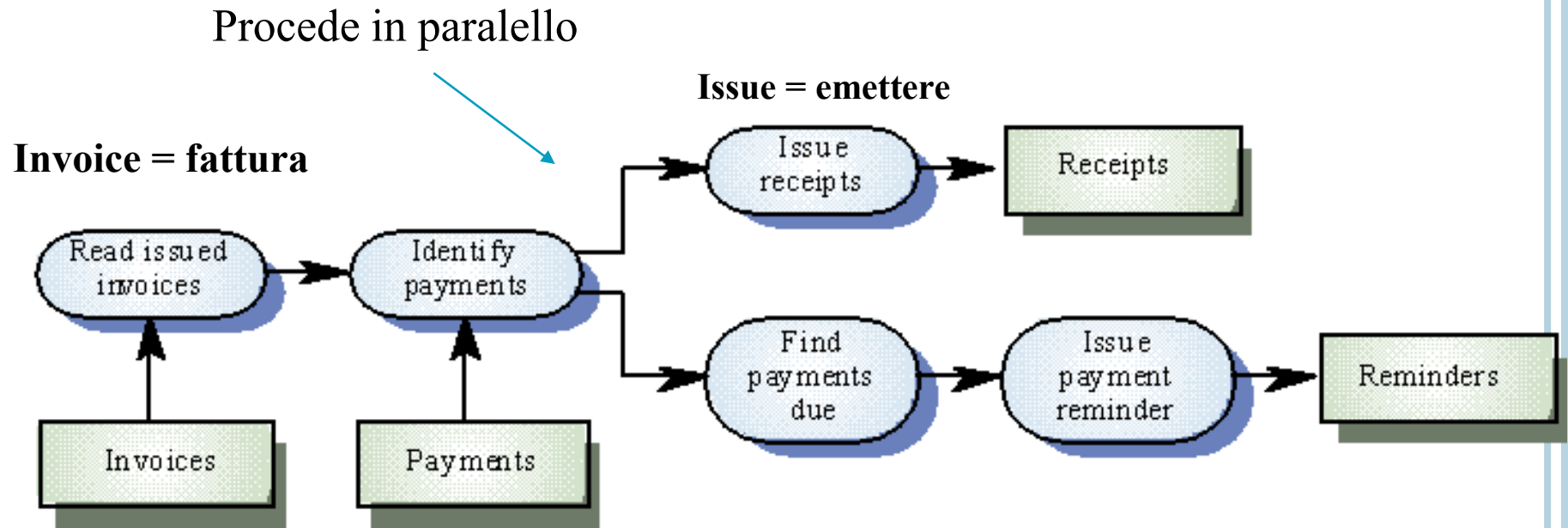


Esempi ben conosciuti di pipe and filter sono i programmi scritti nella **shell UNIX**

```
ls -l | grep "Aug"
```

```
-rw-rw-rw- 1 john 11 Aug 6 14:10 ch02  
-rw-rw-rw- 1 john 85 Aug 6 15:30 ch07  
-rw-rw-r-- 1 john 12 Aug 15 10:51 intro
```

ESEMPIO: INVOICE PROCESSING SYSTEM



Pipe



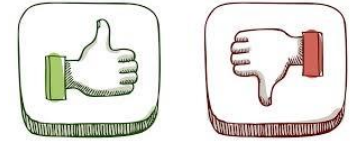
Input o output



Filtro

PIPE AND FILTERS MODEL: PRO E CONTRO

○ Vantaggi



- Supporta il riuso delle trasformazioni
- **Intuitivo**: permette di capire il funzionamento del sistema come composizione dei filtri
- Aggiunta di nuove trasformazioni facile
- Abbastanza semplice da implementare
 - anche con linguaggi OO
- Supporta esecuzione concorrente/parallela

○ Svantaggi

- Non adatto per sistemi interattivi
 - Click del mouse e selezioni dei menu?
- Di solito porta allo sviluppo di **sistemi batch** dove ogni filtro compie una completa trasformazione dell'input
- Lavoro extra per **parsing** e **unparsing** dei dati nei vari filtri (può non essere efficiente)

STILI DI CONTROLLO

- Esistono due stili di controllo:
 - **Controllo centralizzato:** un sottosistema ha la responsabilità generale del controllo e “avvia e ferma” gli altri sottosistemi
 - **Call – return model**
 - **Manager model**
 - **Controllo basato su eventi:** non esiste la figura di “controllore”; ogni sottosistema può rispondere ad eventi generati da altri sottosistemi
 - **Broadcast model** (es. MQTT protocol)



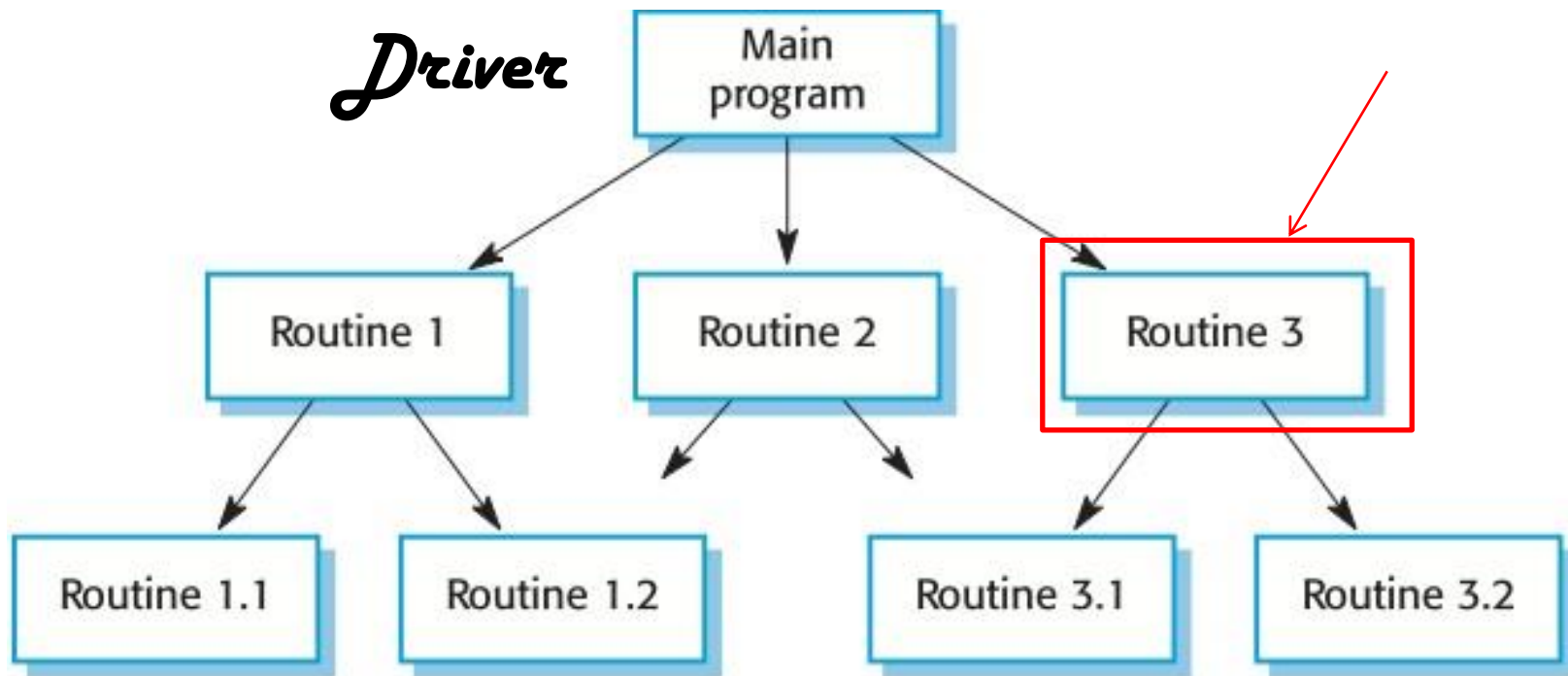
CALL-RETURN MODEL



- Un sottosistema di controllo (**Main**) ha la responsabilità di gestire l'esecuzione degli altri sottosistemi
 - Anche chiamato “**Main Program – Subroutine**”
- Il Main decide l'ordine di esecuzione dei sottosistemi
 - E' il **Driver** (i sottosistemi sono le routine)
- Modello top-down in cui **il controllo** parte dalla radice e si sposta verso il basso
- **Applicabile solo a sistemi sequenziali**

CALL-RETURN MODEL

Solo un sottosistema (quello che ha il controllo) è in esecuzione



MANAGER MODEL

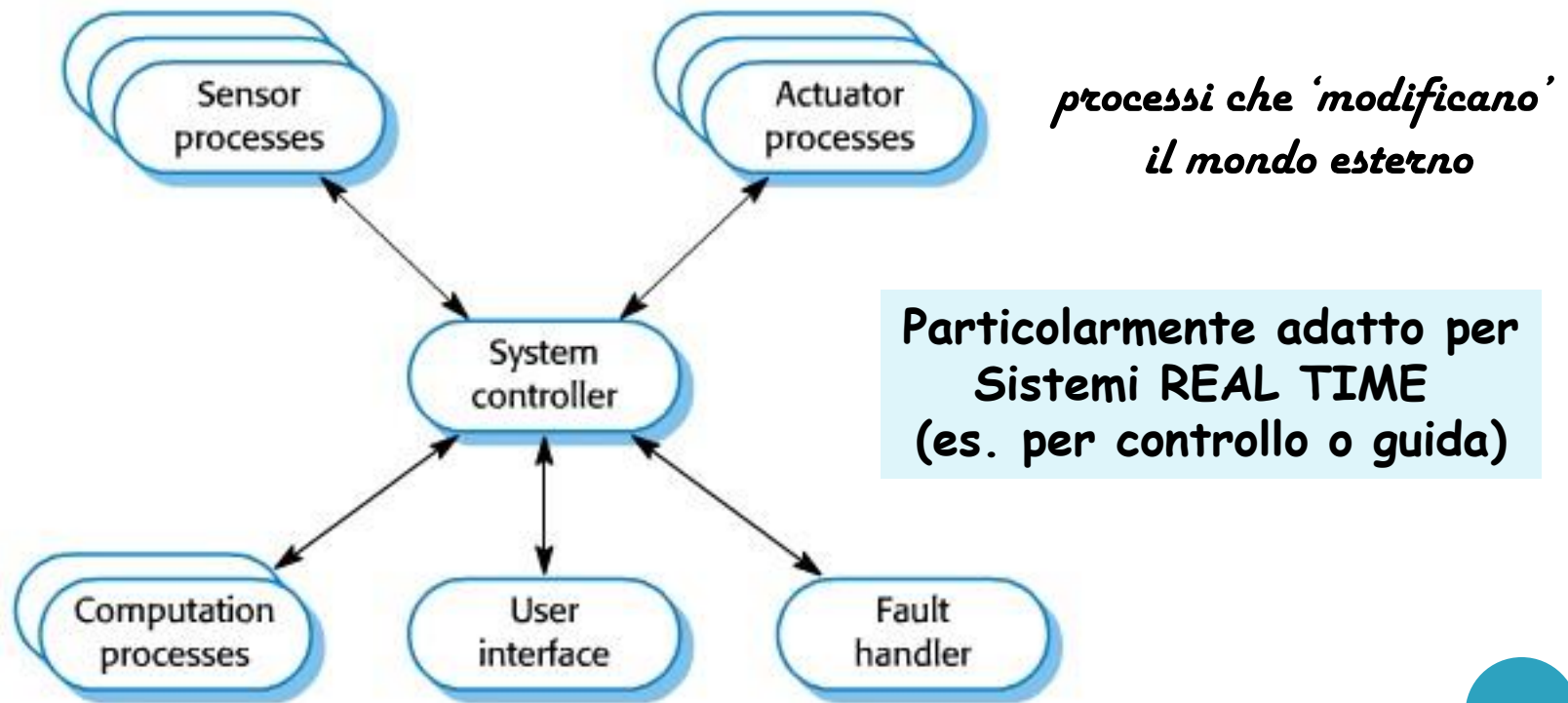
- Se i sottosistemi controllati possono funzionare in **parallelo** allora è possibile usare lo stile Manager model
- Una componente del sistema (**Il Manager**) controlla l'avvio, la terminazione e il coordinamento degli altri processi di sistema
- **Applicabile a sistemi concorrenti**



“Un attuatore è un apparecchio che traduce un segnale elettrico in movimento meccanico”

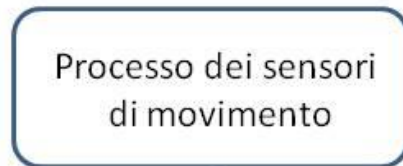
MANAGER MODEL

Il manager (system controller) itera continuamente interrogando i Sensori, UI e output dei processi per identificare eventi o cambi di stato e manda in esecuzione i processi attuatori e di computazione...

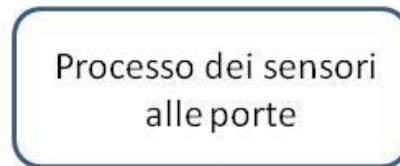


ARCHITETTURA SW: SISTEMA ALLARME BANCA

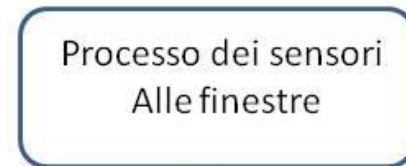
Sensori di movimento nelle stanze



Sensori alle porte



Sensori alle finestre



N° stanza

N° porta

N° Finestra

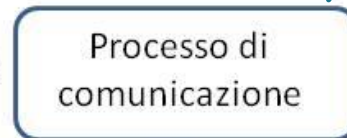
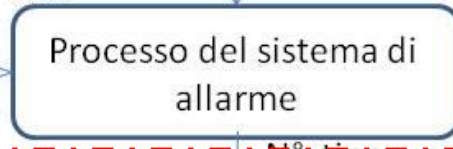
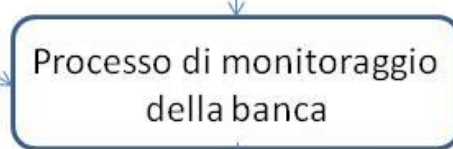
Manager/Controller



addetto

N°, tipo, posizione

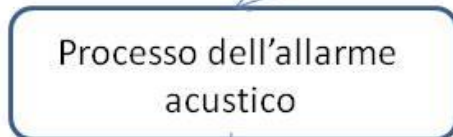
N°, tipo, posizione



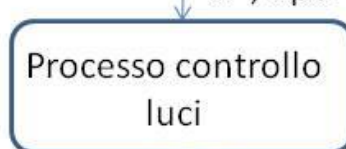
Messaggio vocale

posizione

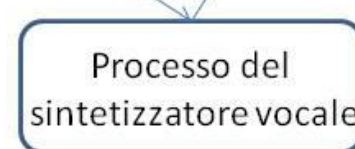
N°, tipo



altoparlante

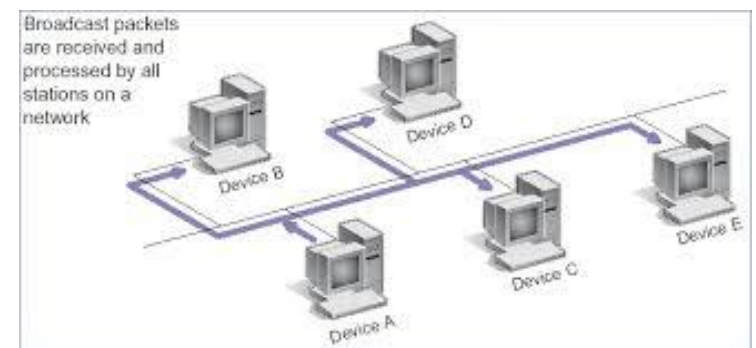


luci



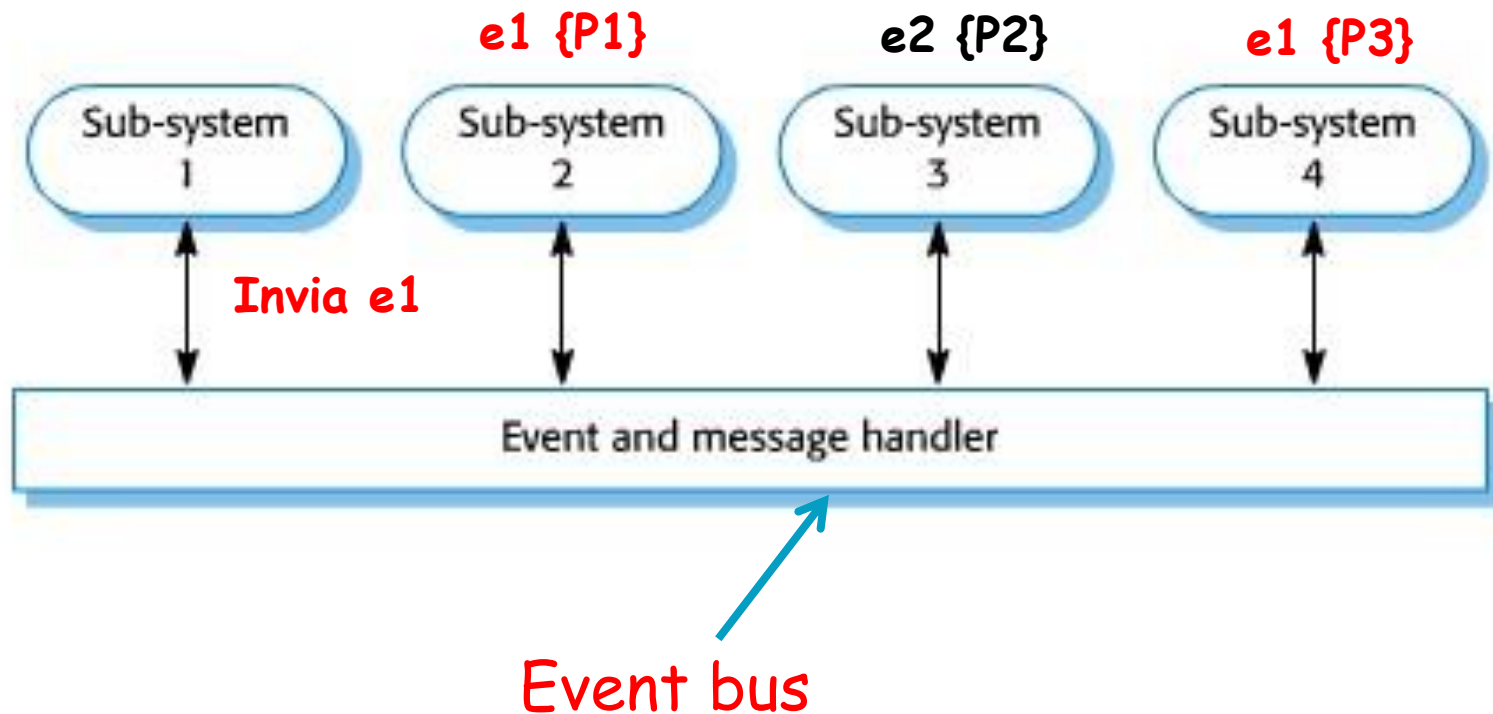
Attuatori

BROADCAST MODEL



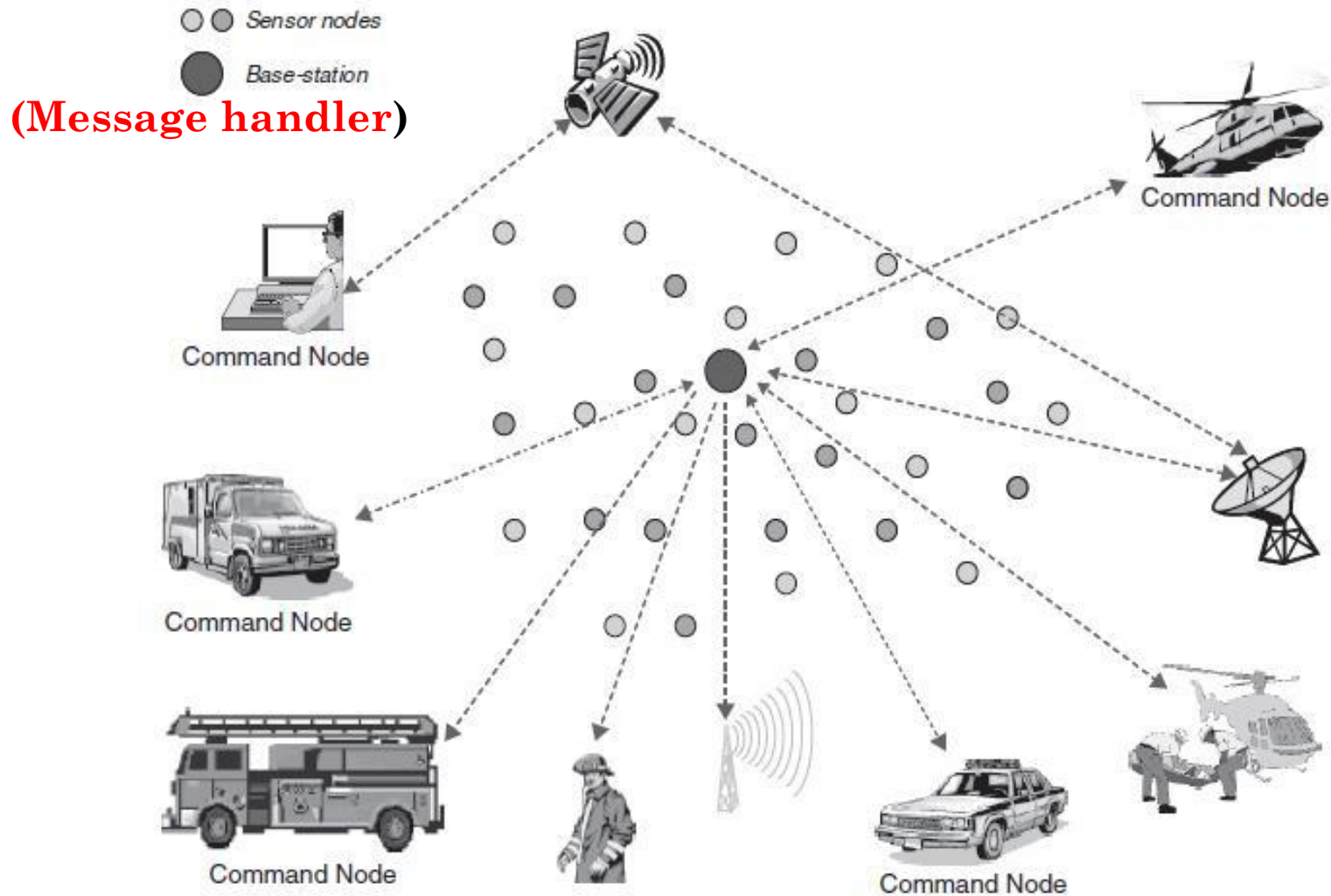
- Invece di invocare una procedura direttamente, un componente può annunciare (**broadcast**) uno o più eventi
 - Un evento è inviato a tutti i componenti
 - Ogni componente in grado di gestirlo può farlo
- **Tutti i componenti sono in ascolto**, quelli interessati all'evento eseguono la procedura associata all'evento
- Così implicitamente un evento causa invocazioni di procedure in altri moduli
- Anche conosciuto come Event based (implicit invocation) o **Publish-Subscribe**

BROADCAST MODEL



Quando Sub1 invia $e1$ le procedure $P1$ e $P3$ sono eseguite

SISTEMI MILITARI E SISTEMI DI SORVEGLIANZA



BROADCAST MODEL: PRO E CONTRO



○ Vantaggi

- Semplice aggiungere/togliere/sostituire componenti
- Semplice il riuso di componenti
 - Un componente può essere riusata in diversi sistemi

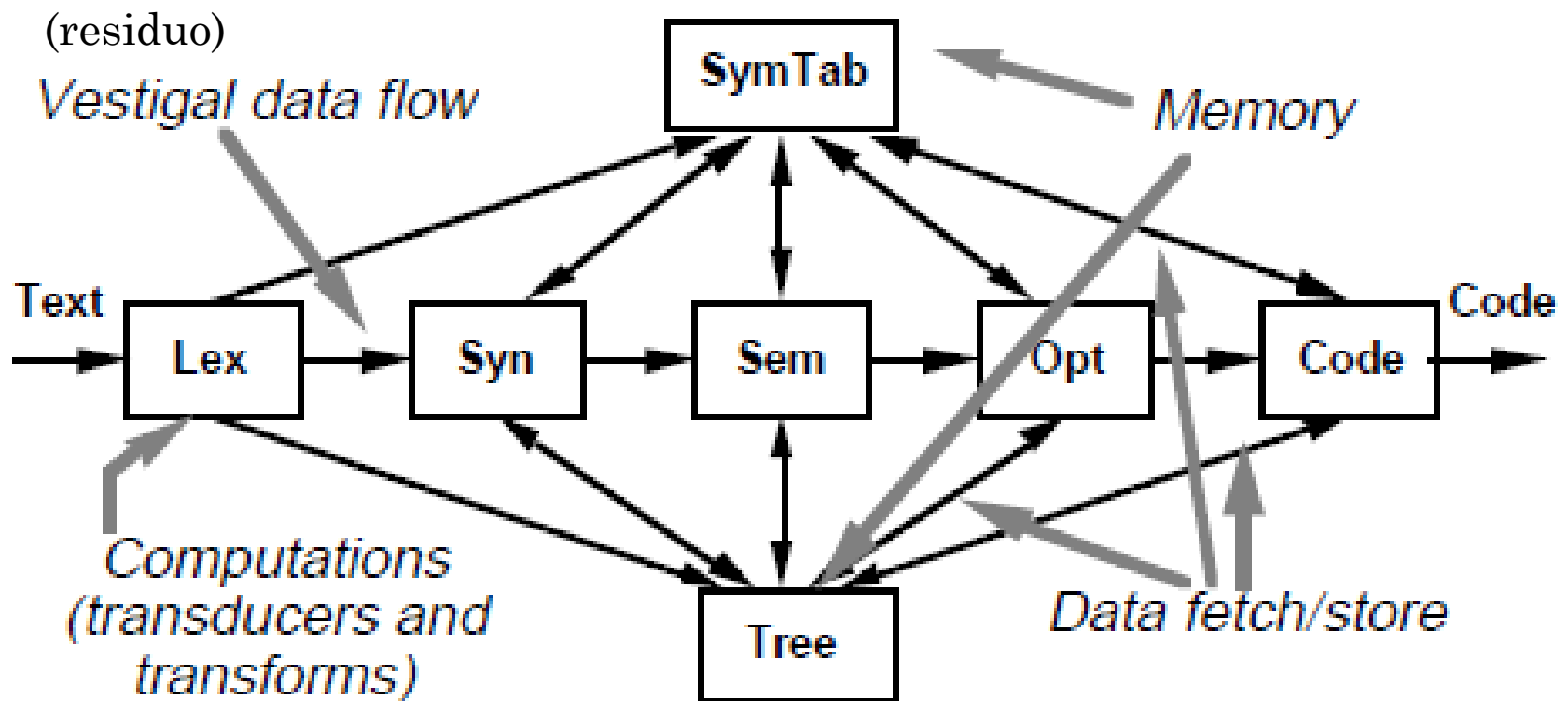
○ Svantaggi

- I sottosistemi quando mandano un evento non sanno se e quando l'evento verrà gestito
- Scambio dei dati può complicare le cose
 - I dati sono passati con l'evento
 - Utilizzo di un repository
 - Problemi di performance
- Possono essere difficili da implementare e **soprattutto testare**

ARCHITETTURE ETEROGENEE

- Fino adesso abbiamo parlato solo di **stili architetturali “puri”** ...
- Ci sono due modi di **combinare** gli stili ottenendo così un **architettura eterogenea**
 - **Modo gerarchico**
 - Es. un sistema organizzato a pipe and filter, di cui una componente interna è a sua volta organizzata come sistema a livelli (layered)
 - Permettendo che una **componente** sia un **mix di architetture**
 - Es. una componente che accede ad un repository ma allo stesso tempo interagisce attraverso pipe con un'altra (architettura pipe-filter)

ESEMPIO: COMPILATORE



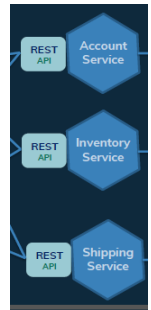
Mix di architetture: unisce “pipe and filters” e repository



MICROSERVICES

Veloce panoramica dello stile architetturale

STILE ARCHITETTURALE A MICROSERVIZI



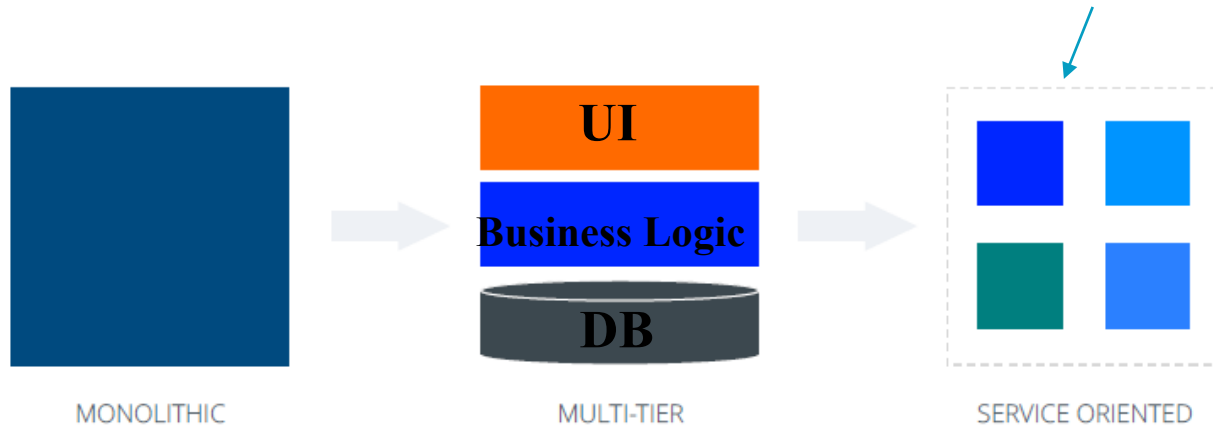
In short, the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API.

James Lewis – Martin Fowler, "Microservices"

***Evoluzione di architetture SOA
(service oriented programming)***

PERCHÉ SONO NATI I MICROSERVIZI?

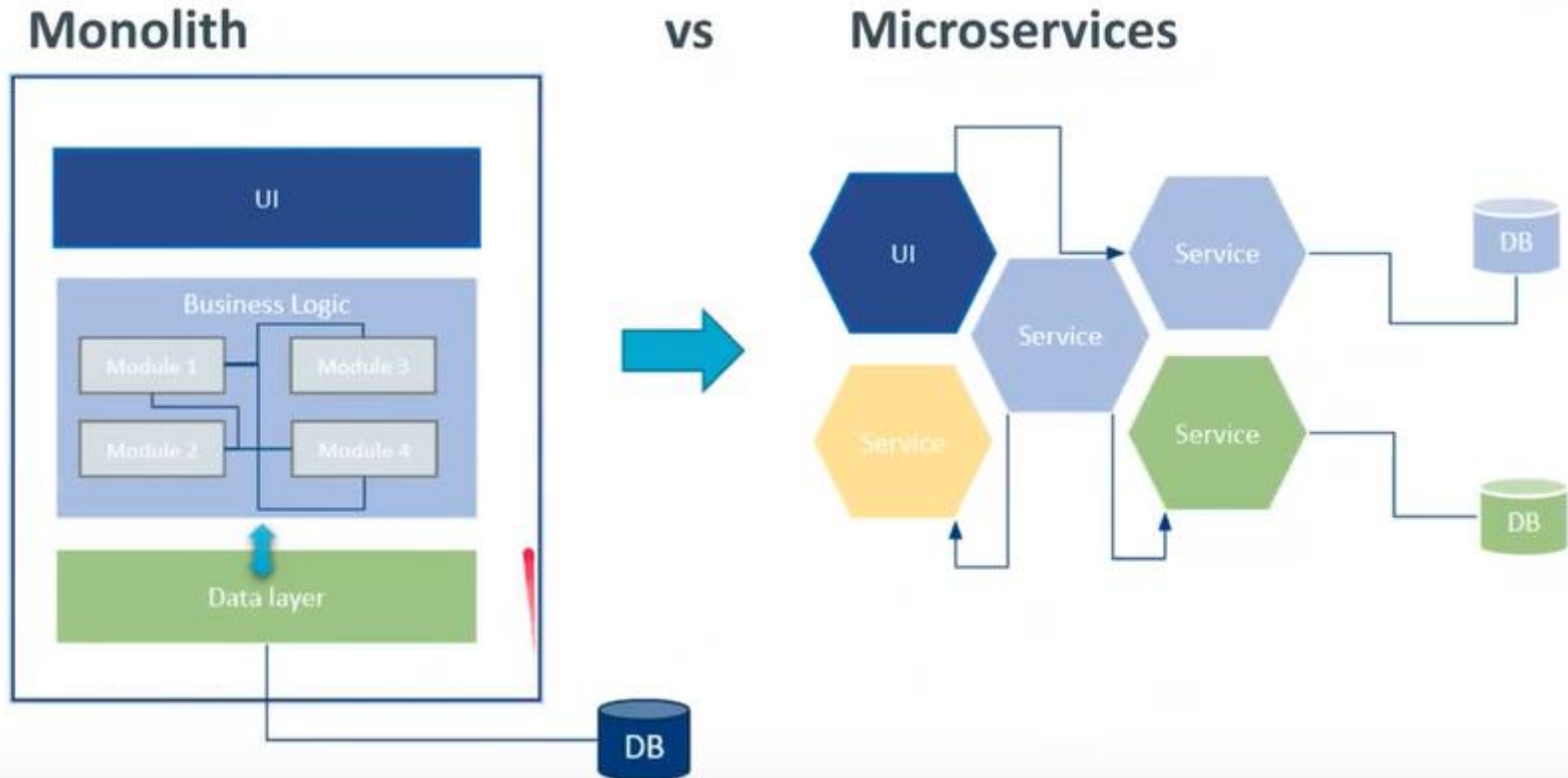
- In contrapposizione alle applicazioni 'monolite' e multi-tier
Divisione per funzionalità (Use case)



Se la app cresce

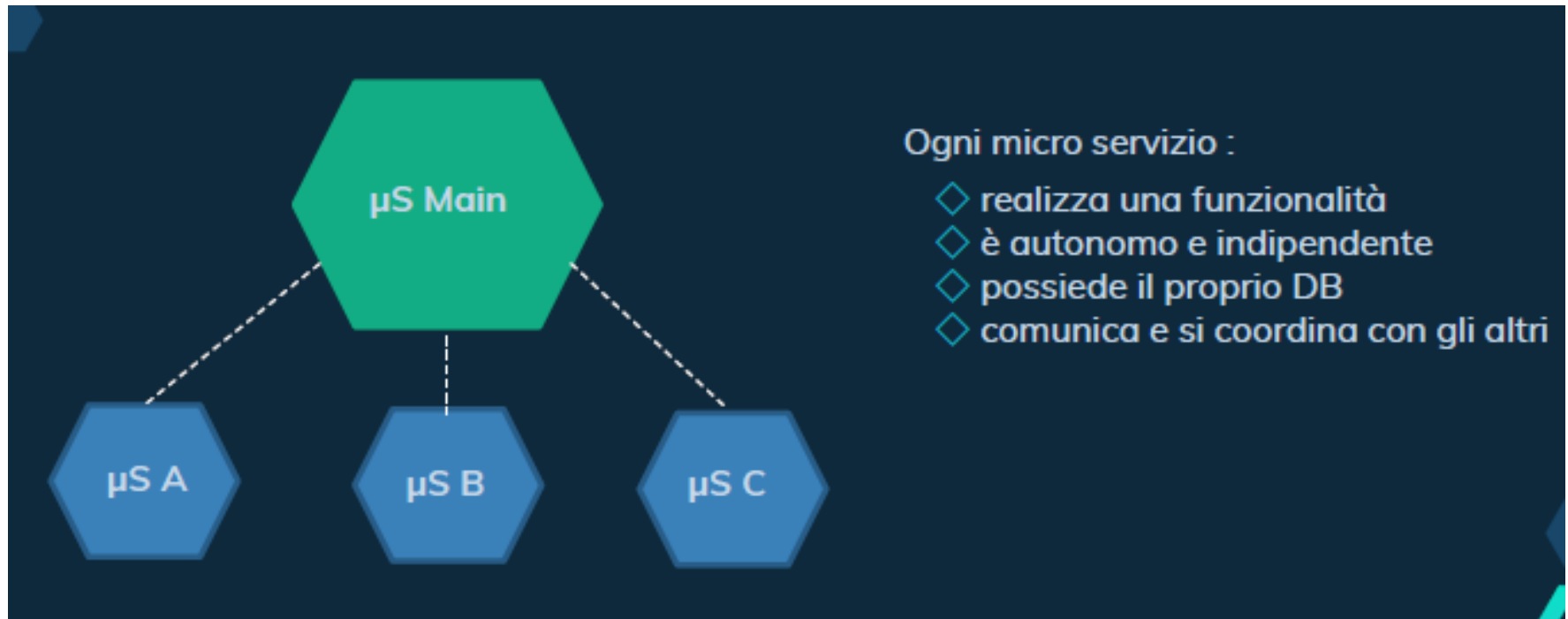
- Complessità aumenta
- Difficile trovare e risolvere bug
- Difficile effettuare modifiche
- Tempi estesi per il deployment
- Complesso lavorare in parallelo (Team)

LA DIFFERENZA



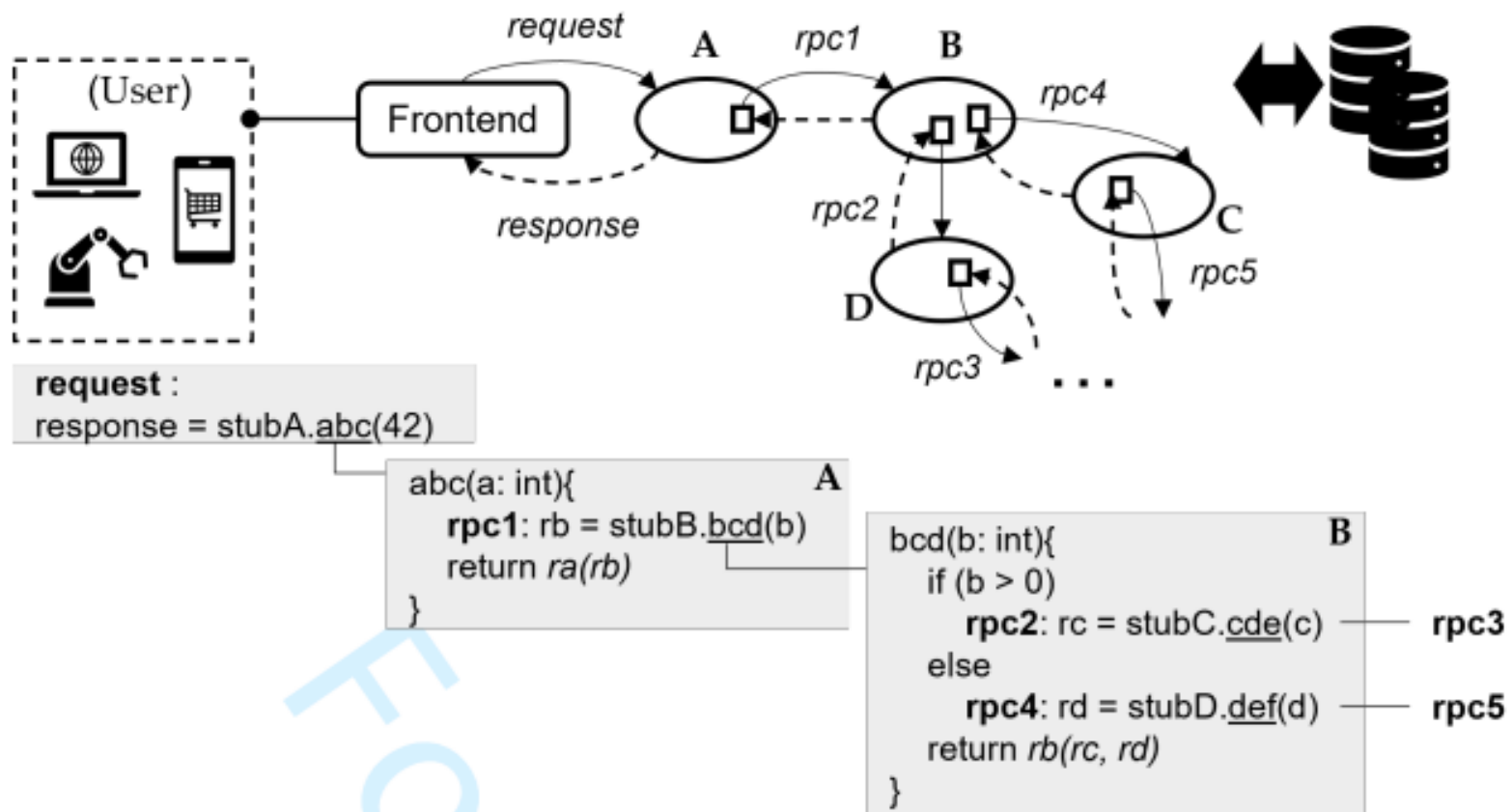
IDEA: avere servizi separati e quindi potenzialmente più piccoli e facili da gestire (sviluppare, testare, deployare), anche dislocati su server diversi

PROPRIETÀ DI UN MICROSERVICE



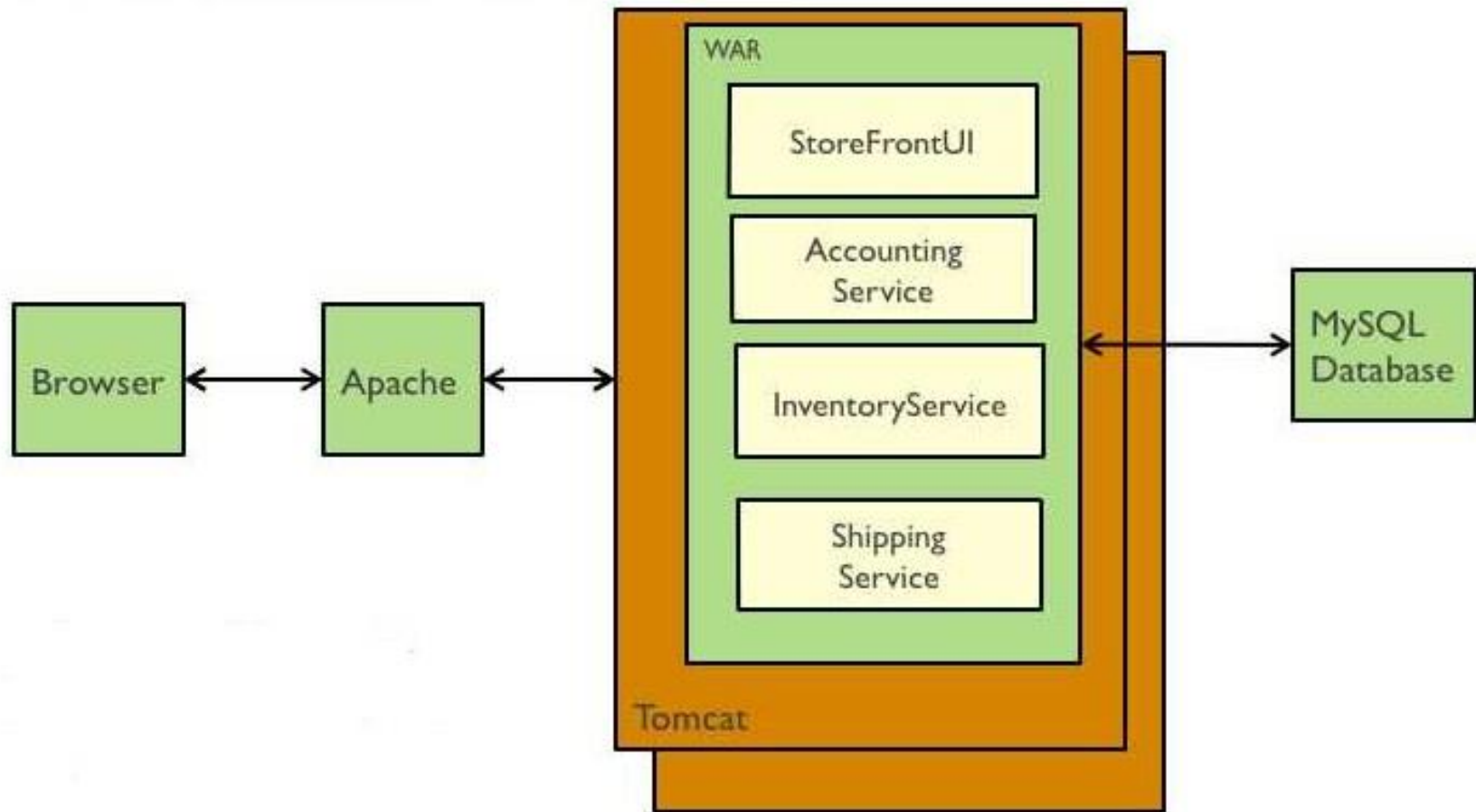
Ogni microservice può essere scritto in un linguaggio di programmazione diverso ed avere diversi DBMS (Relazionali, NoSQL)

COME AVVIENE LA COMUNICAZIONE



Esistono vari Frameworks che permettono di sviluppare app basate su microservizi

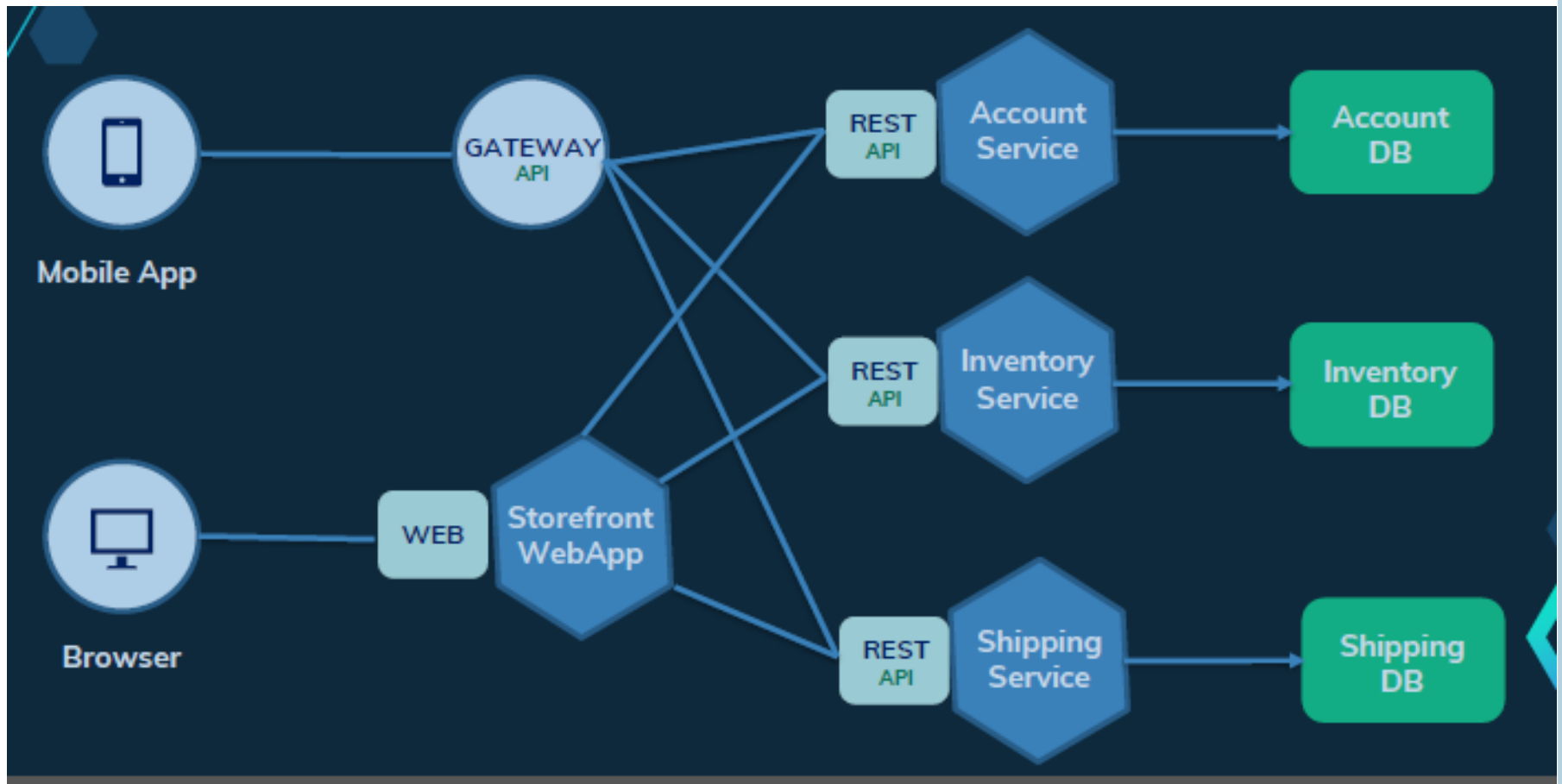
PRIMA: MONOLITHIC E-COMMERCE APP



Applicazione Java monolitica in cui tutti i servizi dell'applicazione sono impacchettati in un unico archivio (WAR) e distribuiti in blocco

REST = *representational state transfer*

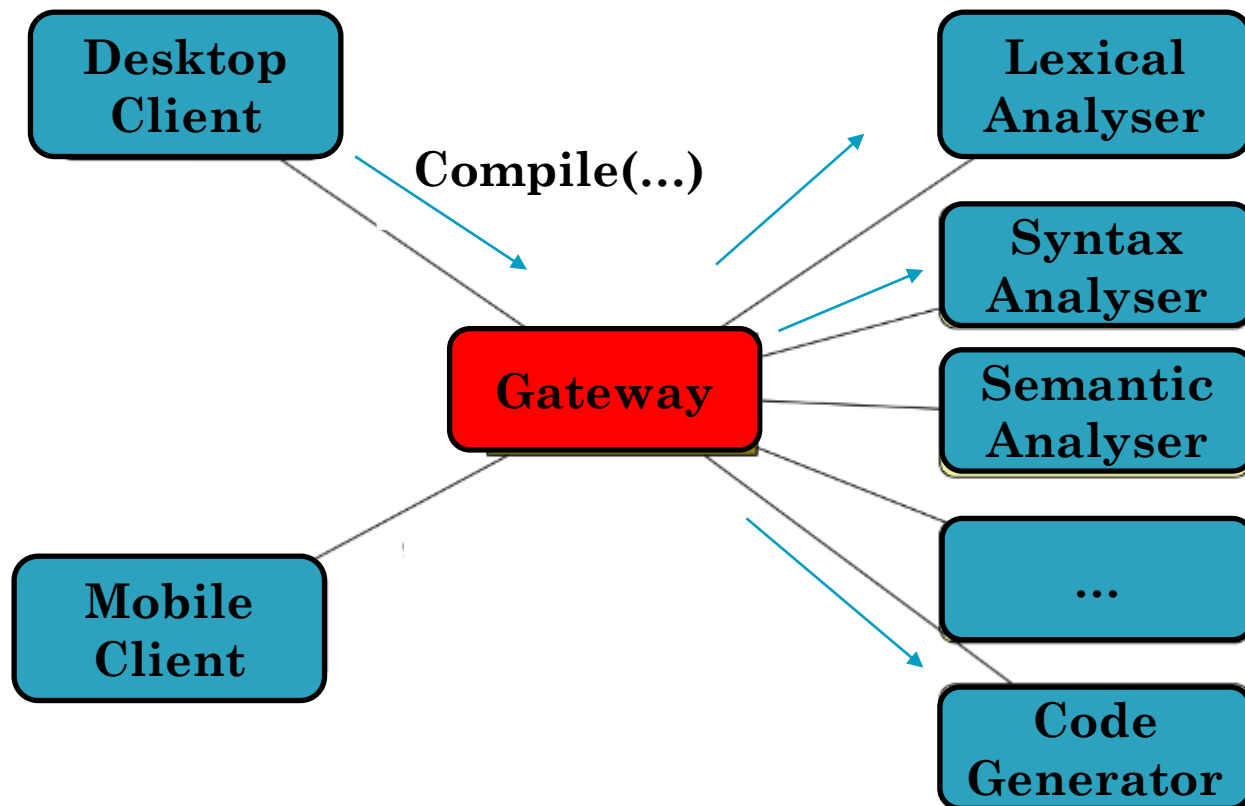
ADESSO: MICROSERVICES E-COMMERCE APP



Servizio finanziario, inventario e spedizione

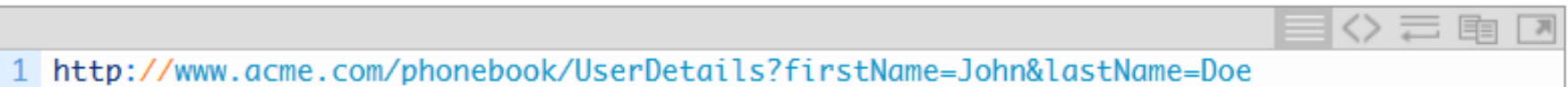
GATEWAY

- **L'API Gateway** (*in italiano: porta/cancello*) espone un'interfaccia verso i client
 - Non confondere con gateway reti fisiche
- Un client chiama ad esempio un unico servizio (via internet) e l'API gateway lo realizza chiamando gli 'n' servizi necessari e invia l'output al client



REST

- Le applicazioni basate su REST utilizzano le richieste HTTP per inviare i dati al server (creazione e/o aggiornamento), effettuare query, modificare e cancellare i dati (DELETE)
- In definitiva, **REST utilizza HTTP** per tutte e quattro le operazioni CRUD (Create / Read / Update / Delete).

A screenshot of a web browser's address bar. The address is `http://www.acme.com/phonebook/UserDetails?firstName=John&lastName=Doe`. The address bar has a light gray background and standard browser navigation icons on the right.

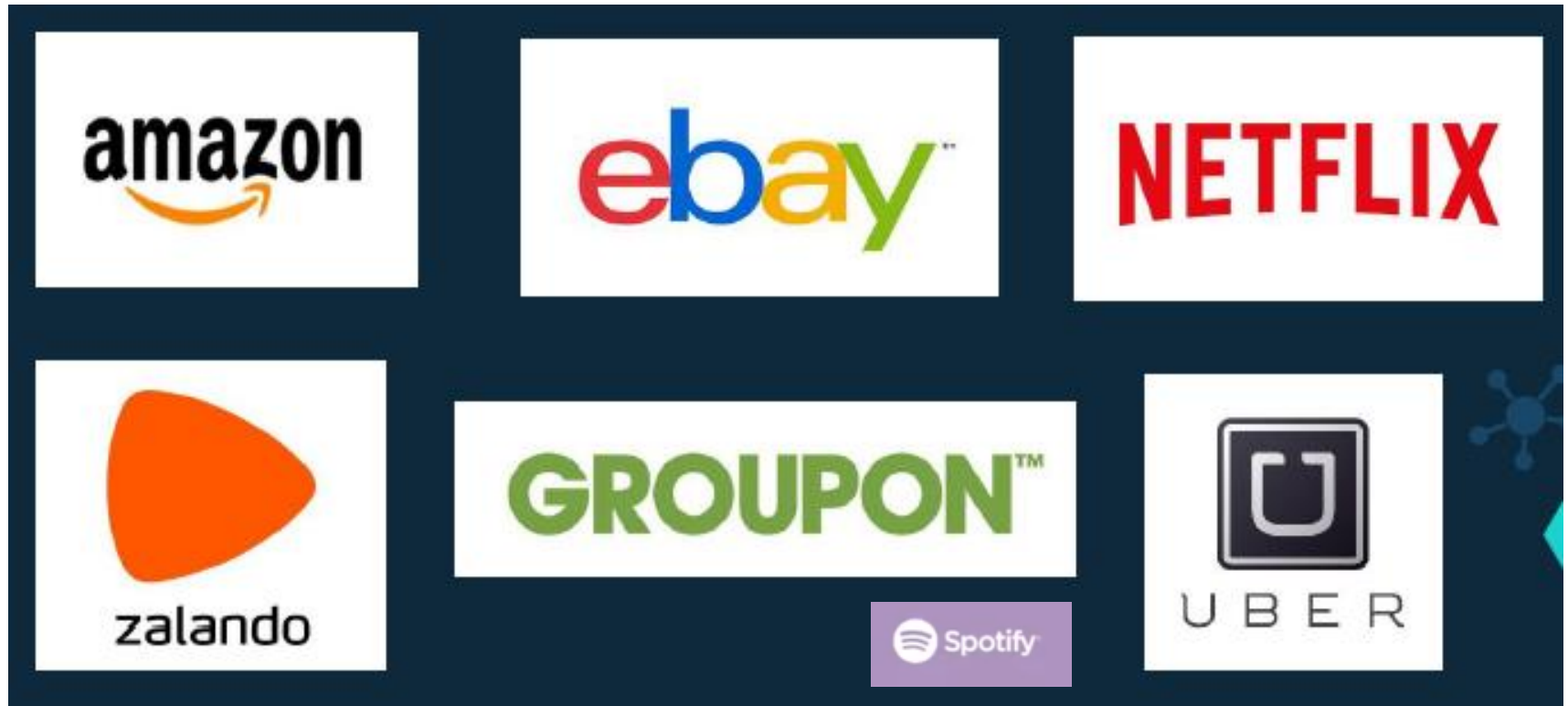
1 <http://www.acme.com/phonebook/UserDetails?firstName=John&lastName=Doe>

Esempio di chiamata di un microservice 'phonebook' che restituisce i dettagli di un utente (John Doe)

PROBLEMI DELLO STILE ARCHITETTURALE A MICROSERVIZI

- ◇ Stabilire dimensione dei micro servizi
- ◇ Sviluppo del meccanismo di comunicazione tra i servizi (IPC)
- ◇ Esposizione ai disservizi di rete
- ◇ Gestione dello schema partizionato dei DB
- ◇ Difficoltà di testing
- ◇ Maggior consumo di risorse e memoria

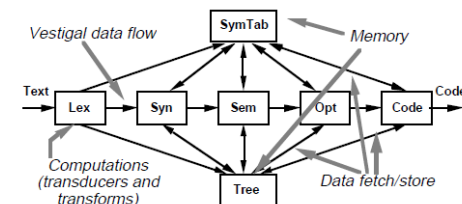
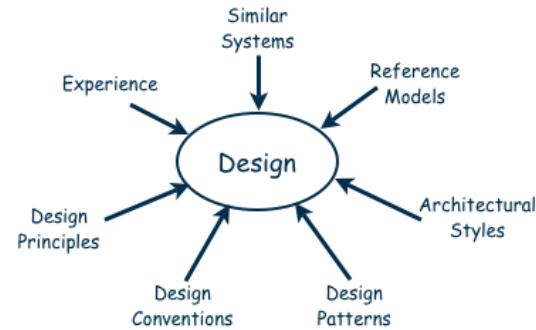
SONO DAVVERO USATI I MICROSERVIZI?



RIASSUMENDO

○ Abbiamo visto:

- che il design ci permette di passare dallo spazio del **problema** alla spazio della **soluzione**
- che progettare un design **non è un task creativo!**
- quale è la differenza tra **High** e **Low** level design
- il concetto di **stile architettuale**
- diversi stili architettureali molto usati
 - ma non tutti ... impossibile ...
 - veloce introduzione dei Microservice
- che spesso i sistemi reali sono un mix di stili architettureali noti



LETTURE CONSIGLIATE

- An Introduction to Software Architecture, David Garlan and Mary Shaw, January 1994 (disponibile online)
- Cap. 5 del libro Software Engineering (fourth edition) S.L. Pfleeger and J. Atlee
- Cap. 11, 12, 13 del libro Ingegneria del software (8° edizione) I. Sommerville

