

# Architetture Web

Lorenzo Sommaruga

Corso Applicazioni Web 1 C02049

Sett. 2018



This work is licensed under a Creative Commons  
<https://creativecommons.org/licenses/by-nc-sa/4.0/>

# Contenuti

- Il web
- Architettura web W3C
- Architetture Web
- La pagina Web
- Il linguaggio HTML
- Il linguaggio XHTML

# Contenuti ed Obiettivi

## Contenuti

- Introduzione ai concetti fondamentali delle architetture web e presentazione dei modelli principali

## Obiettivi

- (BI-2) Comprendere i modelli, le architetture e le potenzialità di sistemi per il web
- (BI-1) Conoscere le architetture 2-Tier, 3-Tier ed n-Tier
- (BI-2) Comprendere la separazione di livelli nelle varie architetture

# Architetture Web: client-server, multi tiers

- Architetture a livelli (tiers)
- Tre tipi separati di funzionalità:

D - Gestione dei Dati

L - Logica di applicazione

P - Presentazione, interazione utente

- L'architettura del sistema determina se queste tre componenti risiedono su un singolo sistema (tier) oppure se sono distribuite su diversi tier

# Architetture Web di sistemi distribuiti

- Possiamo considerare diversi tipi di architetture Web:
  - Client-Server o a 2 livelli (2-Tier)
  - A 3 livelli (3-Tier)
  - A n livelli (n-Tier)

# Client-Server (2-Tier)

- Le applicazioni client-server usano programmi su due computers distinti spostamento del processamento e logica dell'applicazione dal computer centrale a quello del cliente
- Questo spostamento consente una separazione dei dati dalla logica/presentazione
- Il carico di processamento viene concentrato sul cliente mentre il server opera semplicemente come controllore di traffico tra l'applicazione ed i dati
- Il collegamento tra le parti separate viene eseguito mediante un middleware

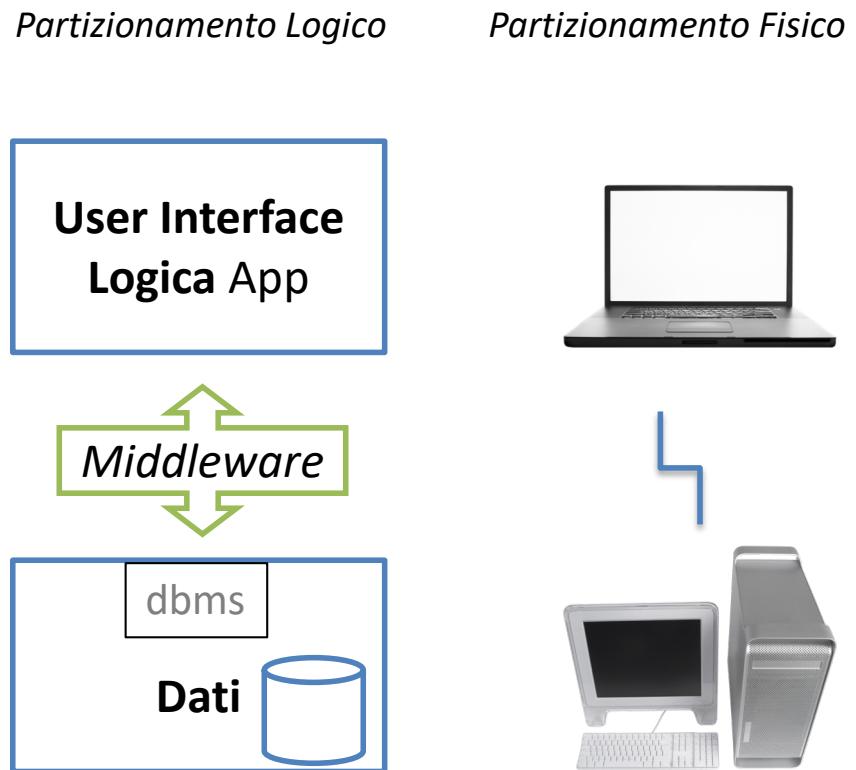
## Svantaggi

- Un cambio alla logica comporta un aggiornamento a tutti i client distribuiti: facile se < 10, complesso se > 1000
- Esigenze di business diverse comportano installazioni di client diversi e versioni diverse

# Schema Client-Server (2-Tier)

- Architettura Client-server

- Presentazione e logica nel client
- Dati ed accesso ai dati nel server



# Client

- Possibilità di interfaccia utente grafica (GUI)
- GUI sfrutta potenzialità dei sistemi operativi (Windows, Xwindows, MacOS, etc)
- Possibilità di coesistenza di client diversi per implementazione e S.O.
- Un client può anche fare da server per altri client
- Sistema aperto: client può collegarsi a numero di servers illimitato

# Server

- Componenti che aspettano passivamente l'arrivo di richieste
- All'arrivo di una richiesta la processano/servono ritornando al cliente il risultato
- Lo stato interno del server è nascosto ai clienti che possono utilizzare il server solo attraverso i servizi esposti
- Un server può anche fare da client per altri server

Per esempio delega esecuzione sottoservizio ad altro server

# Middleware

- DEF.: Middleware
- Software che consente ai componenti di un sistema distribuito di interfacciarsi ed interagire
- Scopi del middleware
  - Gestire la comunicazione tra componenti di un sistema distribuito
  - Garantire disaccoppiamento tra i componenti
  - Permettere sviluppo indipendente di un componente (S.O., linguaggio, piattaforma) dagli altri
  - Semplificare la riconfigurazione del sistema

# Architettura a 3 livelli (3-Tier)

- Una applicazione viene suddivisa in tre livelli logici separati, ciascuno con le proprie interfacce ben definite (APIs):
  1. livello di **presentazione**, consistente di una interfaccia utente grafica (GUI). Questo livello conosce solo come visualizzare i dati e non come accedere ad essi
  2. livello di **logica** (di business) dell'applicazione. Questo livello conosce come processare i dati
  3. livello dei **dati** necessari all'applicazione

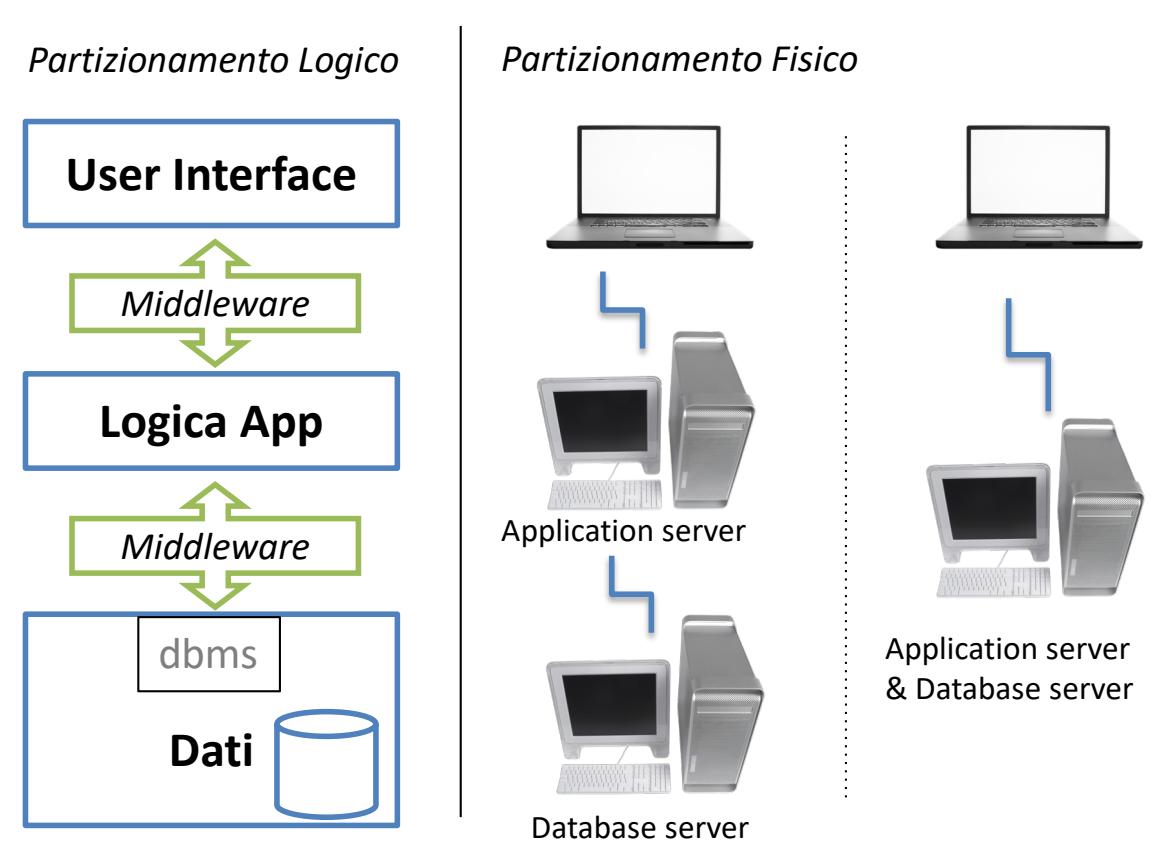
## Vantaggi

- La separazione della logica dell'applicazione dall'interfaccia utente permette grande flessibilità nella progettazione e sviluppo di una applicazione.

# Schema Architettura a 3 livelli

## Architettura a 3 livelli

- Presentazione nel client
- Logica (BR=Business Rules)
- Dati ed accesso ai dati nel server



# Architetture a n livelli (n-Tier)

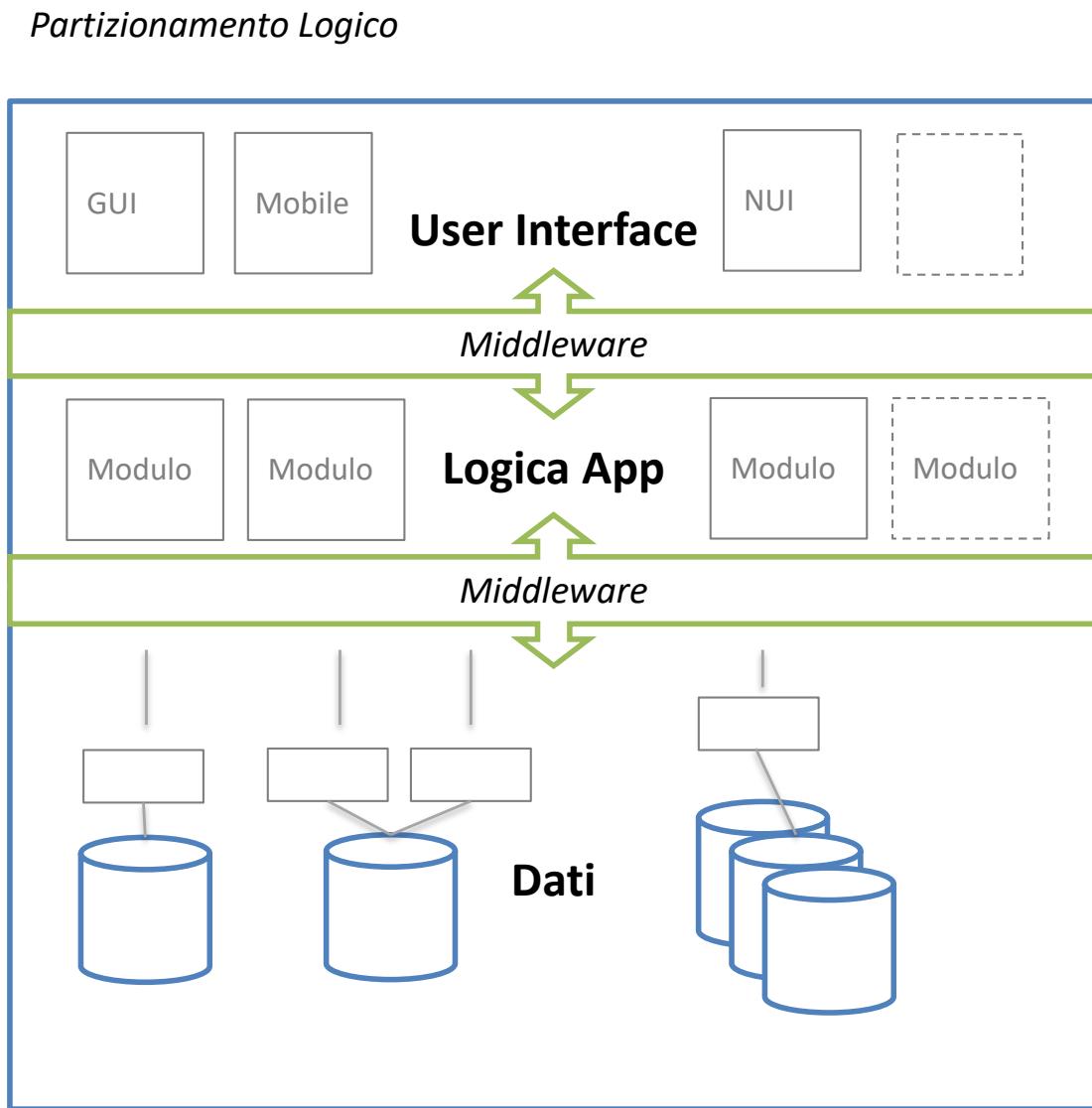
- Estensione della separazione di livelli fino a n
- La logica dell'applicazione viene suddivisa in base alle funzionalità invece che fisicamente
- Maggior modularizzazione (verticale) all'interno di un livello

## Esempio

- Una interfaccia utente gestisce l'interazione con l'utente
- Una logica di presentazione definisce cosa e come viene mostrata l'interazione
- La logica dell'applicazione modella le regole di business attraverso l'interazione con i dati
- Servizi di infrastrutture forniscono funzionalità aggiuntive richieste dai componenti dell'applicazione come per esempio messaggi e supporto a transazioni

# Schema Architettura a n livelli

- Architettura a n livelli



# Caratteristiche Architetture a n livelli

- In ogni livello possono essere presenti più computers
- astrazione fornita dal livello intermedio (logica dell'applicazione)
- si può cambiare l'interfaccia utente o se ne può aggiungere un'altra senza dover toccare i livelli sottostanti
- il collegamento tra i diversi livelli avviene mediante un middleware, che solitamente è già esistente e può essere acquisito  
Esempi di middleware sono CORBA, EJB e DCOM
- La diversificazione delle funzionalità attraverso i vari livelli genera dei software eterogenei e paralleli
- legacy data: si mantengono dati e applicazione e si connettono al livello intermedio mediante un middleware
- interfaccia utente: è raccomandabile l'uso di una interfaccia basata su un browser in quanto permetterebbe dei cambi all'interfaccia stessa senza dover aggiornare tutte le applicazioni sui client

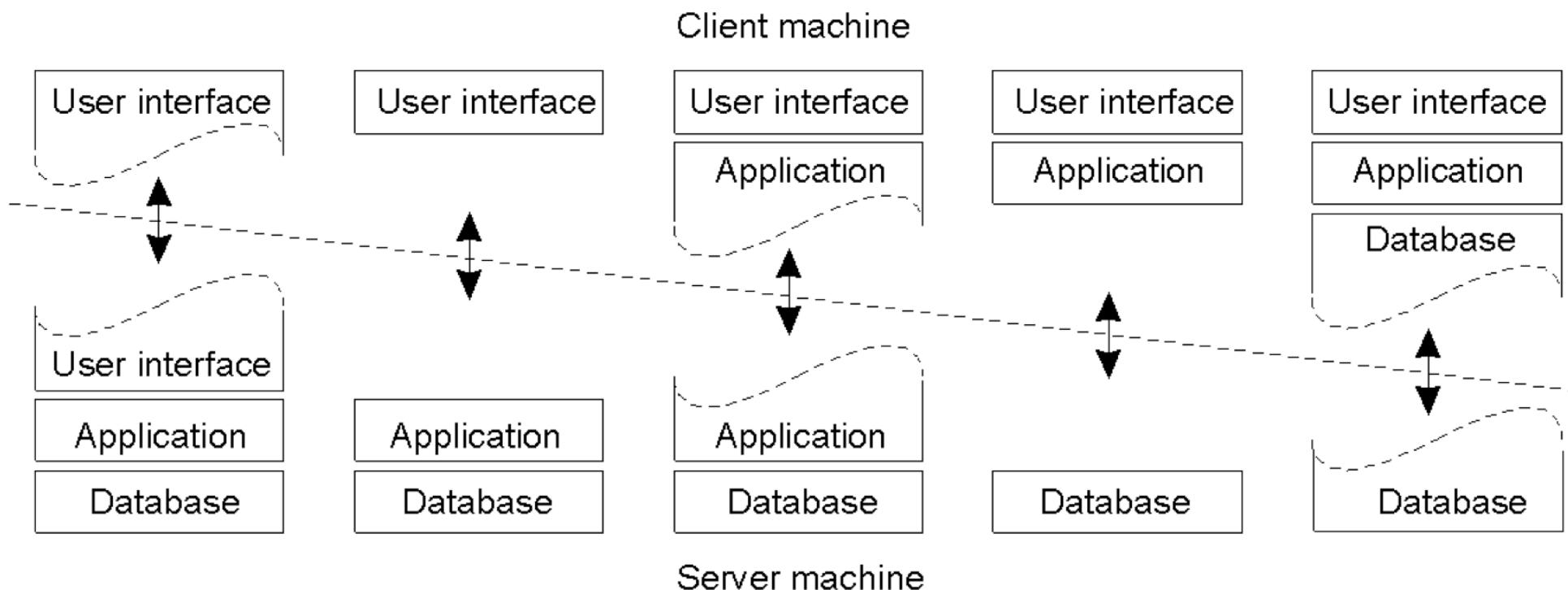
# Trasparenza Architetture Distribuite

## *Trasparenza*

- Locazione  
permette di accedere alle risorse senza conoscere la locazione
- Accesso  
permette di accedere risorse locali e remote con le stesse operazioni
- Mobilità  
permette di spostare le risorse e i clienti in un sistema senza influenzare le operazioni di utente
- Tolleranza ai guasti  
permette il mascheramento dei guasti in modo che gli utenti possano completare le operazioni richieste anche in presenza di guasti di componenti hardware o software
- Scalabilità  
permette al sistema e alle applicazioni di espandersi in modo scalabile senza modificare la struttura del sistema o gli algoritmi applicativi
- Prestazioni  
permette di riconfigurare il sistema al variare del carico per migliorare le prestazioni
- Replicazione  
permette l'uso di copie multiple di risorse per aumentare l'affidabilità e le prestazioni, senza che gli utenti ne abbiano conoscenza
- Concorrenza  
permette ad un insieme di processi di operare concorrentemente condividendo risorse e senza interreferire fra loro

# Architetture Multi-tier

- Diverse distribuzioni fisiche client-server



# Thin / Thick / Smart Client

- Nell'implementazione di una architettura client/server bisogna stabilire se sarà il client o il server a effettuare il maggior carico di lavoro

- Thin Client

Il client “leggero” si appoggia ad un processamento remoto centrale sul server. Il client comunica con esso e ha bisogno di scarse risorse (hw e sw). Utile e.g. quando la stessa info deve essere acceduta da molti client (come in ambienti pubblici aeroporti, stazioni , etc), facile aggiornamento ...

- Thick (fat) Client

Il cliente “pesante” esegue localmente sul sistema dell’utente il processamento e non si appoggia al sistema remoto centrale del server; difficile aggiornamento

- Smart (rich) Client

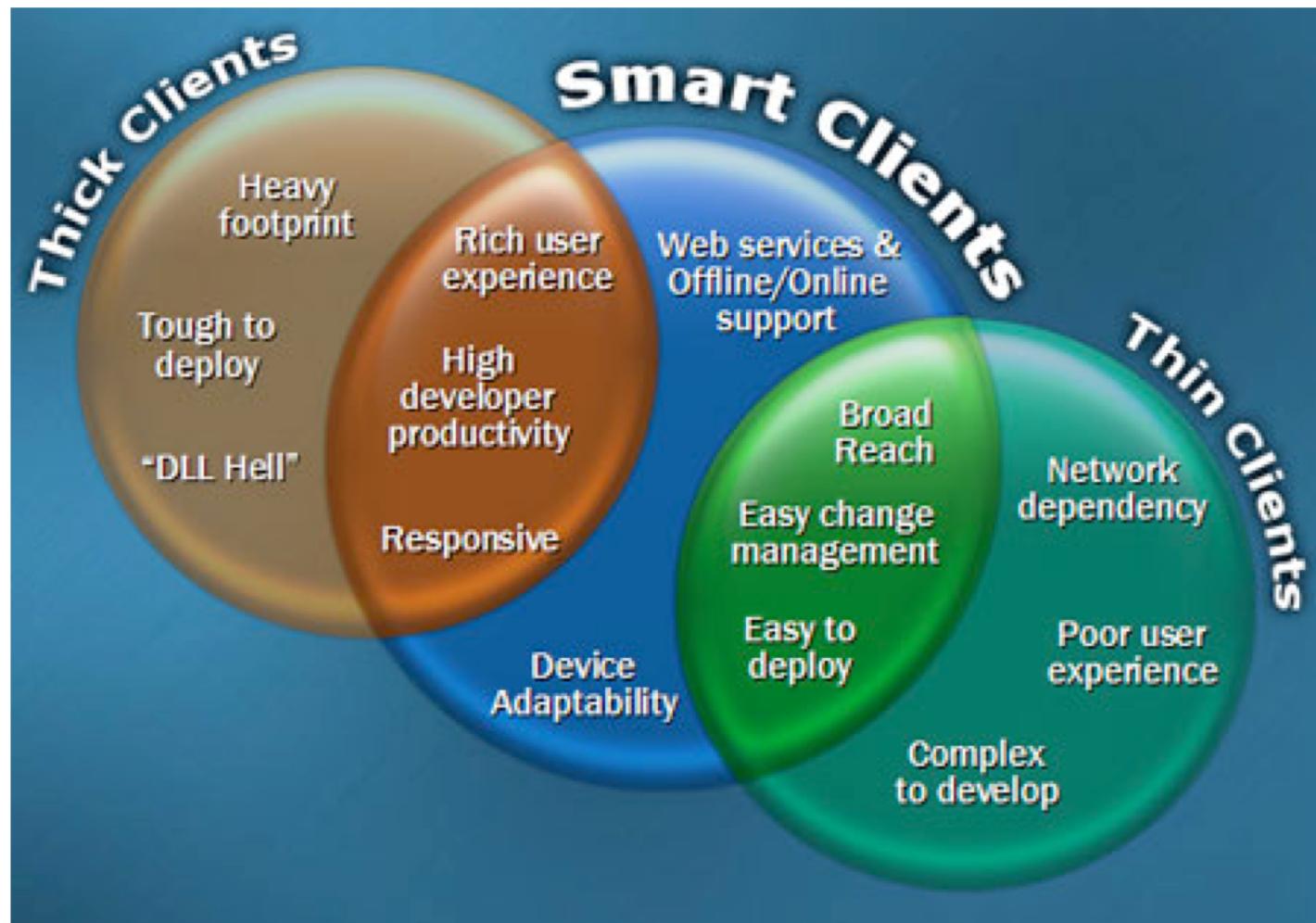
Transizione da architetture tradizionali client/server a modelli Web-based. In modo più simile a Thick client che a Thin, gli Smart client sono nodi connessi a internet che consentono ad applicazioni locali all’utente di interagire con applicazioni server mediante Web services.

# Per il Web: Smart Client

Def.:

- Sono applicazioni client facilmente rilasciabili e gestibili che offrono una esperienza interattiva ricca, adattativa e responsive sia in scenari online sia offline sfruttando risorse locali e collegandosi intelligentemente a data sources distribuiti
- Diversi da applicazioni browser-based

# Diversi Clienti



# PWA - Progressive Web App

- Uniscono il meglio del web e delle app
- NON richiedono install
- L'utente progressivamente migliora l'app nel suo uso
- Si caricano rapidamente
- Mandano notifiche push
- Icona su home screen
- Esperienza full screen

# Cosa sono le Progressive Web App

## Principi chiave

- Progressive: incremento progressivo da comportamento web a app nativa
- Responsive: adattamento a forma: desktop, mobile, tablet, o futuro
- Connettività indipendente da condizioni rete: mediante service workers per offline o reti bassa qualità
- App-like: UX come in una app, basate su app shell model che separa le funzionalità dell'applicazione dal contenuto
- Fresche: sempre aggiornate grazie al processo di update dei service workers
- Sicurezza: via HTTPS
- Scopribilità: identificabili come "application" (W3C manifest <https://www.w3.org/TR/appmanifest/> e registration scope), quindi searchable
- Notificabilità: reattive a notifiche push
- Installabilità: semplice, aggiungibile a home screen
- Linkabilità: facilmente condivisibili via URI

Esempio completo: weather pwa <https://weather-pwa-sample.firebaseio.com/final/>

# PWA- Architettura della shell dell'applicazione

- L'architettura della shell dell'applicazione separa l'infrastruttura principale dell'applicazione e l'interfaccia utente dai dati:
  - L'interfaccia utente e l'infrastruttura sono memorizzate localmente nella cache mediante un Service worker
  - nei caricamenti successivi, la Progressive Web App recupera solo i dati necessari e non deve caricarli tutti
- La shell dell'applicazione è un frammento di codice HTML, CSS e JavaScript che attiva un'interfaccia utente
- La shell dell'applicazione deve:
  - Caricarsi rapidamente
  - Essere memorizzata nella cache
  - Visualizzare i contenuti in modo dinamico

# PWA- Progettazione della shell dell'applicazione

- Suddividere il progetto nei vari componenti chiave
- Porsi le seguenti domande:
  - Cosa deve apparire immediatamente sullo schermo?
  - Quali altri componenti dell'interfaccia utente sono essenziali per l'app?
  - Quali risorse di supporto sono necessarie per la shell dell'applicazione? Ad esempio, immagini, JavaScript, stili e così via.

## Esempio

Per creare un'app Meteo come una Progressive Web App, i componenti chiave sono:

- Un'intestazione con un titolo e pulsanti di aggiunta/aggiornamento
- Un contenitore per le schede delle previsioni
- Un modello di scheda delle previsioni
- Una finestra di dialogo per aggiungere nuove città
- Un indicatore di caricamento