

ARCHITETTURA SOFTWARE DEI SISTEMI DISTRIBUITI

Definizione di Sistema Distribuito (DCS)

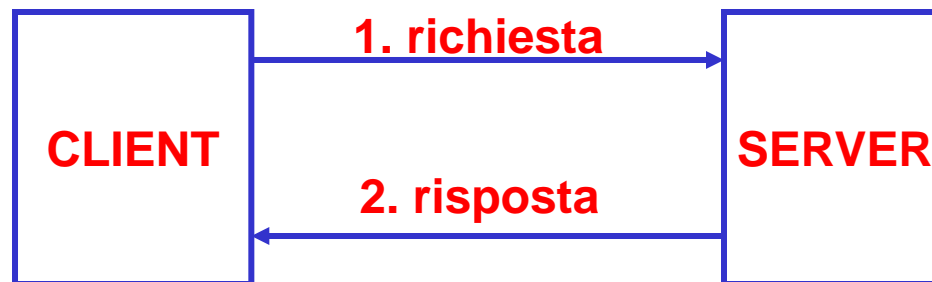
- ❑ Sistema di elaborazione distribuito (DCS - distributed computing system):
 - ❖ sistema costituito fisicamente da una collezione di calcolatori **autonomi** interconnessi tramite una rete
- ❑ L'elaborazione è distribuita
 - ❖ Qualsiasi tipo di sistema di elaborazione (da mainframe a smartphones)
- ❑ Differente da modello Host-Terminale (con terminali non intelligenti)

Architetture Software

- ❑ L'architettura software di un sistema distribuito
 - ❖ è basata sul modello di riferimento a livelli (OSI)
 - ❖ prevede un insieme di processi
 - eseguiti fisicamente sui diversi calcolatori del sistema
 - che interagiscono tra loro secondo un certo modello
- ❑ I livelli di interesse per il programmatore sono quelli applicativi (5-6-7 Session Presentation Application)
- ❑ È possibile classificare le architetture in base ai **modelli di interazione** usati

Il Modello Client-Server (C/S)

- ❑ È il modello attualmente più usato
- ❑ Ogni interazione avviene tra 2 processi: uno svolge il ruolo di **cliente**, l'altro quello di **servitore**.
- ❑ L'interazione è basata su uno **scambio di messaggi**: il client invia la **richiesta** ed il server la **risposta**



Il Modello Client-Server (C/S)

- ❑ Il ruolo (client o server) si riferisce alla singola interazione, non al processo:
 - ❖ Uno stesso processo può anche svolgere in alcune interazioni ruolo client e in altre ruolo server
- ❑ Architettura client-server:
 - ❖ (più) processi client che svolgono il ruolo di client
 - ❖ Uno o più processi server che offrono servizi (calcolo, immagazzinamento dati, ...) ai client

Il Modello Peer-to-peer (P2P)

- ❑ Ogni interazione avviene tra 2 processi, ma in modo simmetrico
- ❑ L'interazione si basa su uno scambio di messaggi, ma ciascuno dei due processi può prendere l'iniziativa di inviarli

Client/Server vs P2P

	Vantaggi	Svantaggi
Client/Server	<ul style="list-style-type: none">• Architettura semplice e ben conosciuta	<ul style="list-style-type: none">• Single point-of-failure, potenziale bottleneck• Necessità elaboratori potenti e affidabili
P2P	<ul style="list-style-type: none">• Basso costo• Buona affidabilità e prestazioni grazie a ridondanza	<ul style="list-style-type: none">• Minor controllo possibile• Sicurezza difficile da gestire• Maggior difficoltà di gestione

Principali Particolarità del Software Distribuito

- ❑ Il software distribuito è sempre **concorrente** (i processi girano su CPU diverse)
 - ❖ occorre risolvere i problemi tipici della concorrenza (sincronizzazione, coordinamento, ecc.)
- ❑ La **comunicazione** tra componenti software residenti su host diversi **incide sulle prestazioni e può fallire.**
 - ❖ occorre tenerne conto nello sviluppo del software
- ❑ È anche possibile il **crash parziale** (di singoli calcolatori del sistema) o la **caduta delle connessioni di rete**
 - ❖ occorre adottare strategie di fault tolerance

Principali Particolarità del Software Distribuito

- ❑ Il software distribuito gira su **piattaforme eterogenee**
 - ❖ occorre risolvere i problemi dell'eterogeneità di hw e s.o.
- ❑ Il software distribuito è per sua natura più esposto ad attacchi che compromettono la **sicurezza**
 - ❖ Es. validazione input, denial-of-service, ...
- ❑ Il software distribuito può avere la necessità di **localizzare** processi all'interno del sistema

Il Middleware

- I problemi di base comuni al software distribuito vengono risolti da uno strato detto middleware, che sta tra le applicazioni vere e proprie e il S.O.:

Applications		application
Middleware		presentation
		session
Network Services	Local Services	transport
Operating system and hardware		network
		data link
		phisical

Il Middleware

- ❑ fornisce servizi **business-unaware** per il coordinamento di e la comunicazione tra processi (o utenti) remoti
- ❑ **maschera** la presenza della rete, l'eterogeneità dei computer, i problemi di security, ecc.
- ❑ Esempi di software classificabile come middleware:
 - ❖ web browsers
 - ❖ database drivers
- ❑ Esempi di software non classificabile come middleware:
 - ❖ airline reservation system (business aware)

Tipici Servizi del Middleware

□ Servizi di interazione:

- ❖ scambio di informazioni, gestione delle connessioni, meccanismi per evitare i deadlock, ecc.

□ Servizi per specifiche categorie di applicazioni:

- ❖ accesso a database, transaction processing, distributed object management

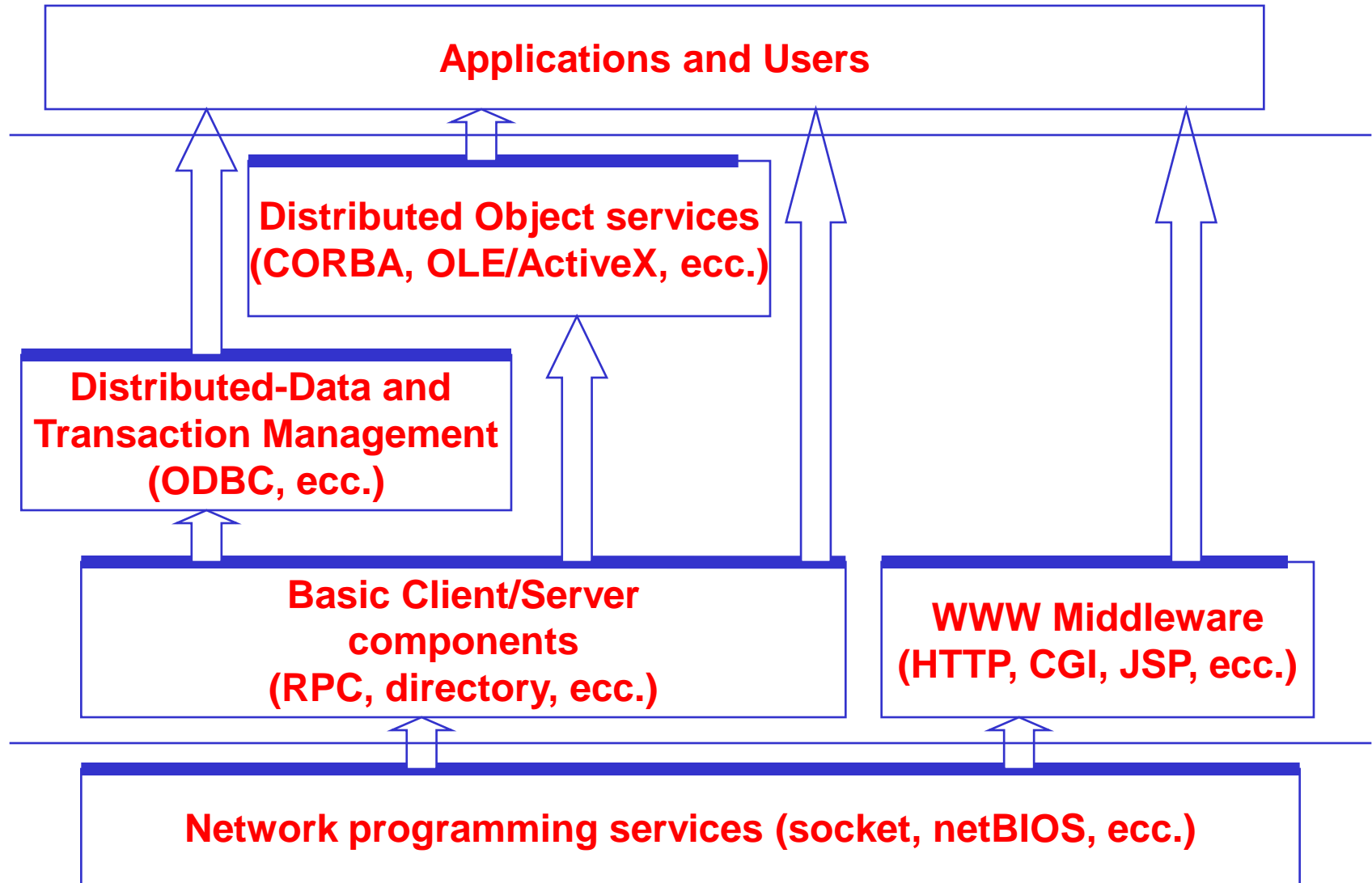
□ Servizi di gestione, controllo, amministrazione:

- ❖ directory, security, monitoraggio delle prestazioni, ecc.

Organizzazione del Middleware: i Componenti

- ❑ Il middleware può essere organizzato come un insieme di **componenti**, ciascuno dei quali fornisce determinati servizi ed è accessibile attraverso determinate **API**
- ❑ I servizi di un componente possono essere usati da altri componenti, per costruire servizi più complessi
- ❑ I servizi base usati da tutto il middleware sono i servizi di connettività forniti dai livelli rete e trasporto.

Esempi Componenti Middleware



Organizzazione del Middleware: gli Ambienti (Frameworks)

- ❑ Componenti di vario tipo possono essere combinati in **ambienti** middleware, che
 - ❖ forniscono una molteplicità di servizi di vario tipo (tutti i principali tipi)
 - ❖ possono includere strumenti di sviluppo appositi
- ❑ Esempi di ambienti:
 - ❖ Oracle J2EE
 - ❖ Microsoft .NET

Apertura del Middleware

- ❑ Il middleware può essere più o meno **aperto** (all'integrazione di componenti di produttori diversi)
- ❑ Si possono distinguere due aspetti indipendenti:
 - ❖ l'apertura dei protocolli utilizzati
 - ❖ l'apertura delle API

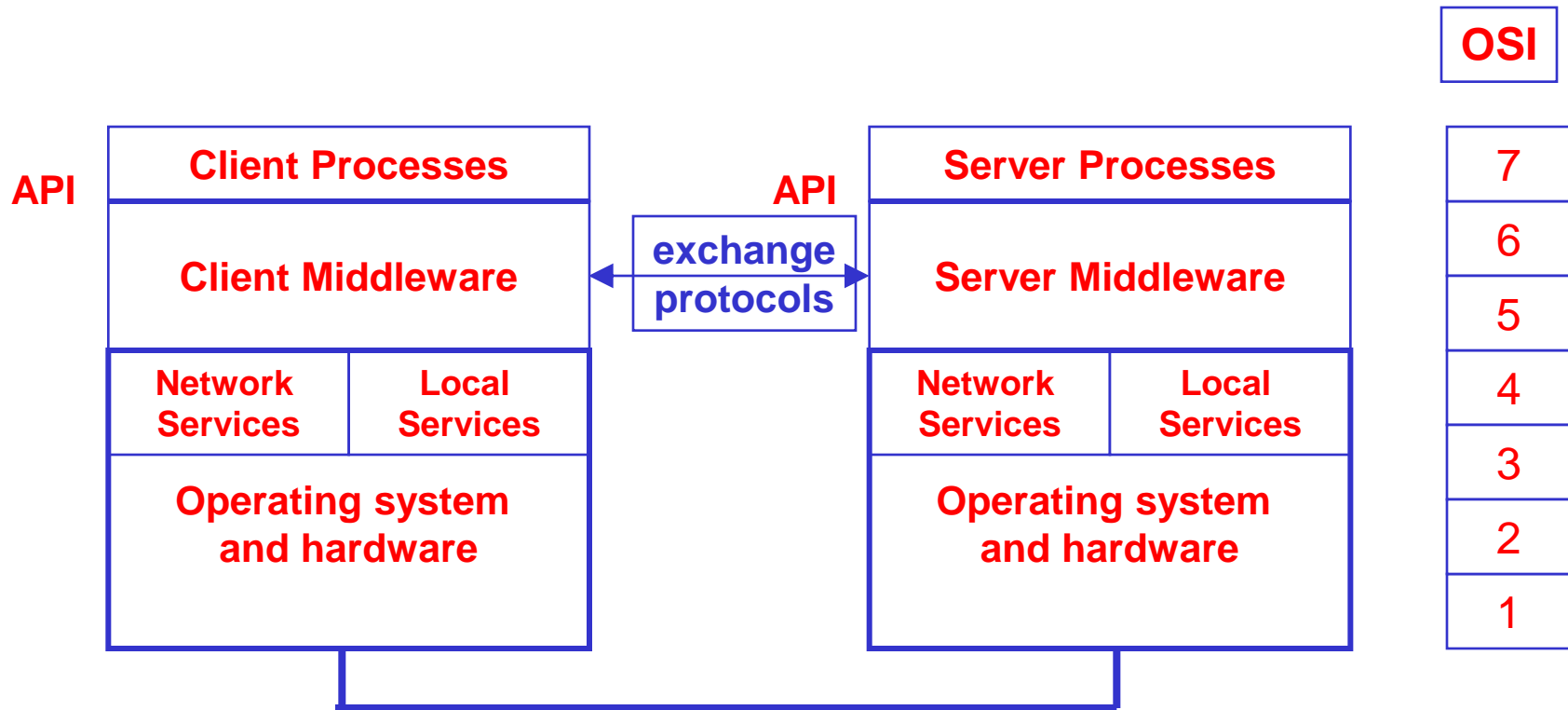
Evoluzione e Tendenze Attuali

- I sistemi distribuiti attuali (quelli che studieremo) tipicamente possiedono le seguenti caratteristiche:
 - ❖ object-oriented
 - ❖ client/server
 - ❖ Internet based

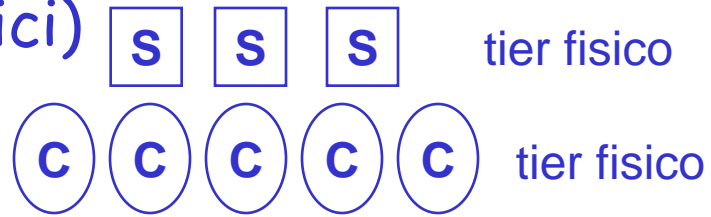
Le Generazioni di Sistemi C/S

- ❑ I generazione (1G) - anni '80-'90
 - ❖ singole applicazioni per condividere servizi particolari (es. costosi): stampa, accesso a file condivisi, database, ecc. (print/file/DB-server). Middleware quasi inesistente.
- ❑ II generazione (2G) - metà anni '90
 - ❖ sistemi C/S object-oriented, maggior interoperabilità, componenti
- ❑ III generazione (3G) - anni 2000
 - ❖ ulteriore sviluppo dei componenti e degli ambienti (frameworks che includono strumenti per lo sviluppo di applicazioni)

Anatomia del Software C/S



Application Tier e Physical Tier

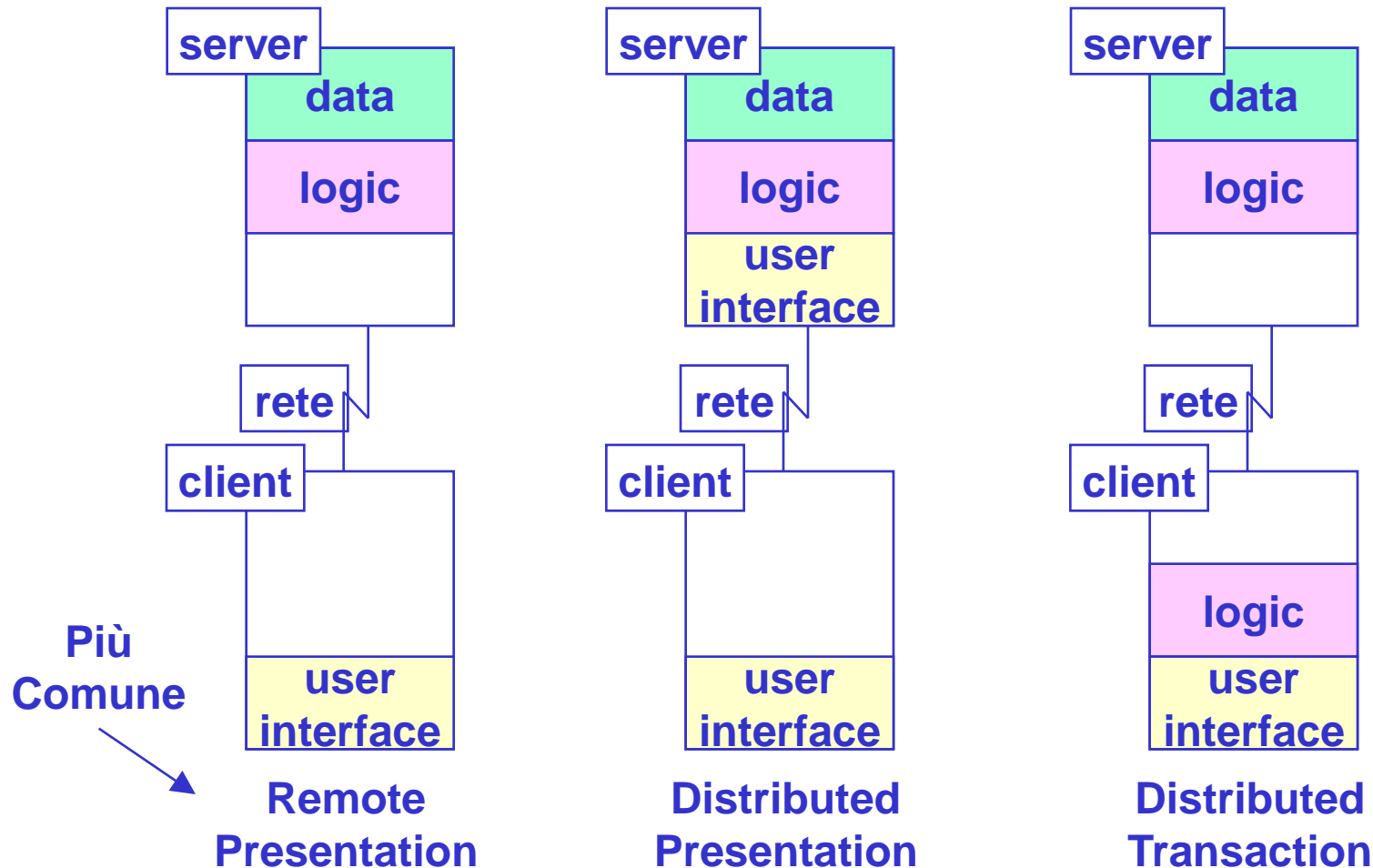
- ❑ Ogni applicazione può essere divisa **logicamente** in tre componenti principali (tier logici):
 - ❖ Interfaccia utente (interazione)
 - ❖ Elaborazione (logica dell'applicazione)
 - ❖ Dati (DB, files)
- ❑ In un sistema distribuito i tier logici dell'applicazione vengono mappati su più processi che girano sui calcolatori del sistema (tier fisici)
- ❑ La mappatura tier logici/ tier fisici può essere fatta in diversi modi
- ❑ L'interazione tra i vari componenti può essere sincrona o asincrona

Architetture C/S 2-tier

- ❑ Prevedono solo 2 tier fisici: uno client (detto anche **front end**) ed uno server (detto anche **back end**)
- ❑ È l'architettura classica, tipica della prima generazione di sistemi C/S.
- ❑ A seconda di come si distribuiscono i tier logici (interfaccia, elaborazione, dati) su quelli fisici, si hanno diverse configurazioni

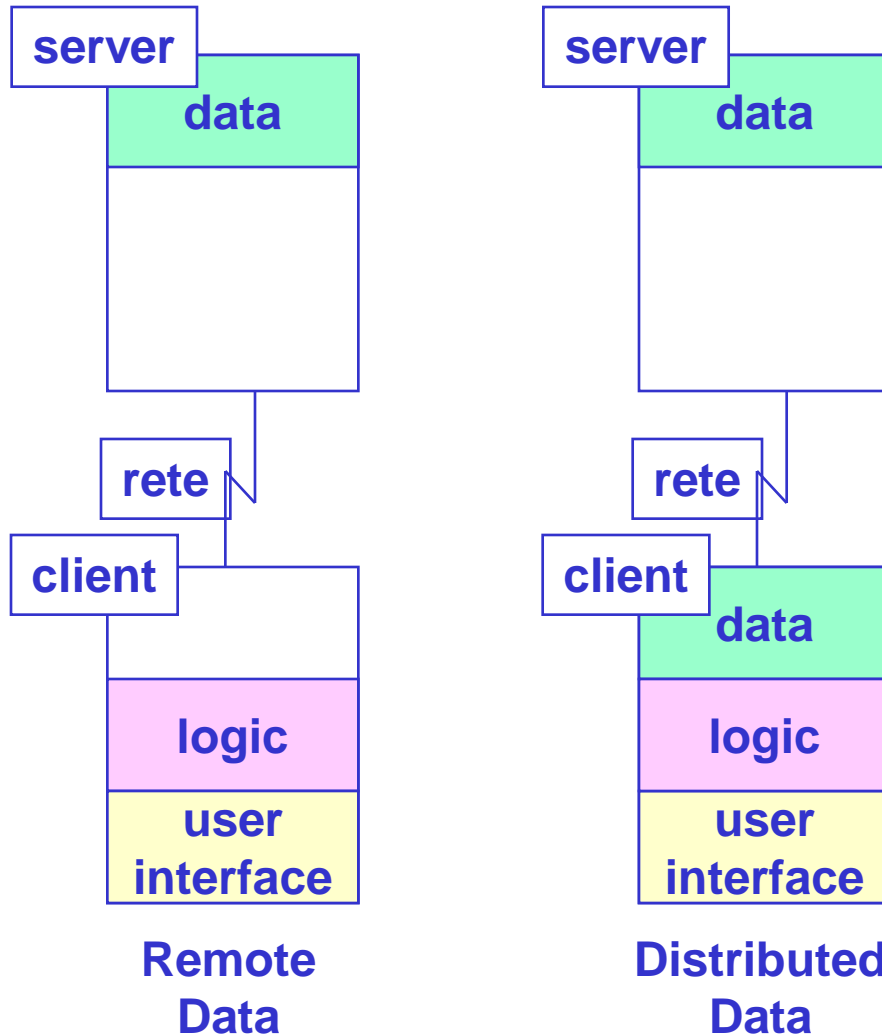
Classificazione delle Architetture

C/S 2-tier (Gartner [1])



Classificazione delle Architetture

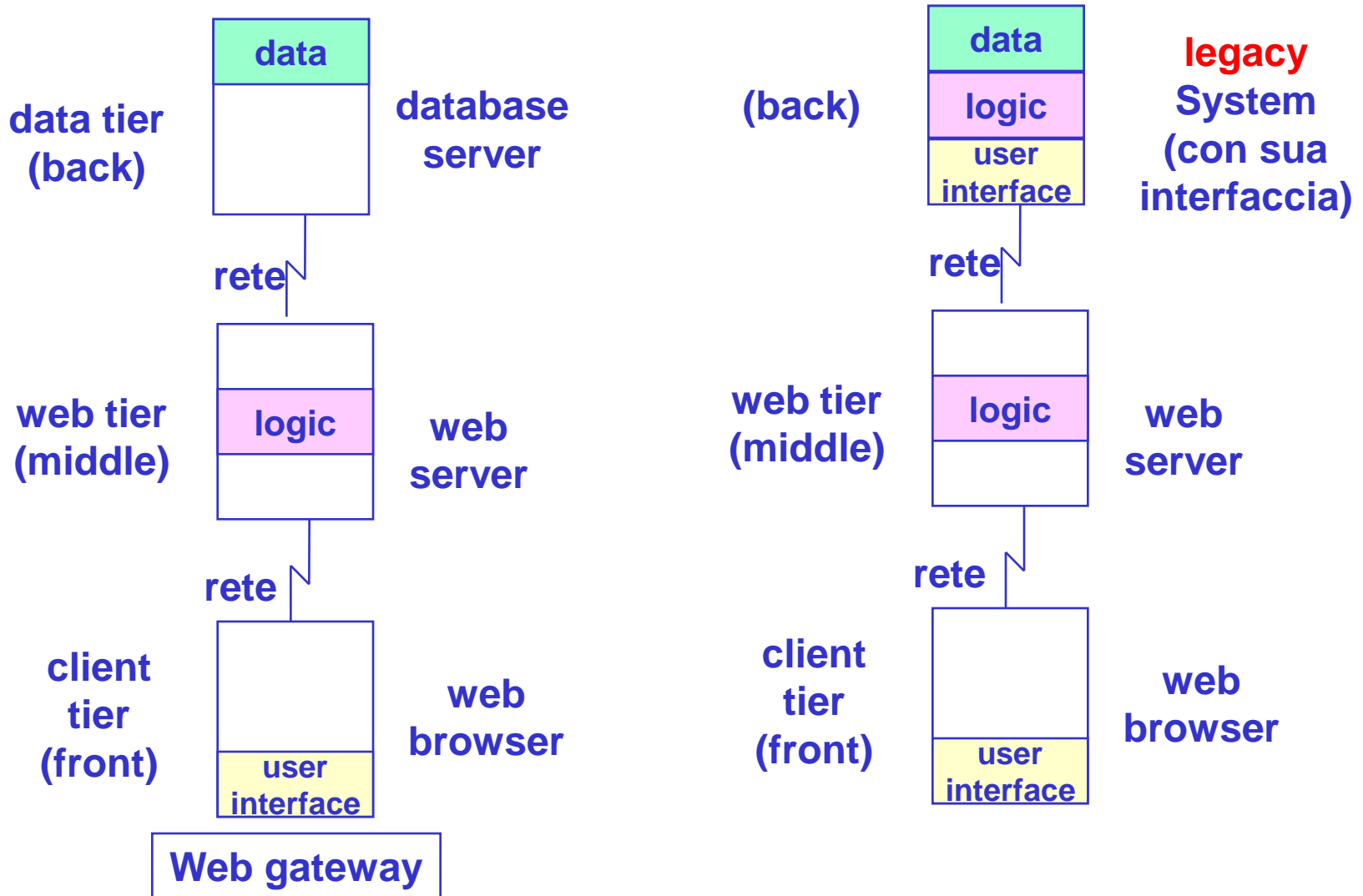
C/S 2-tier (Gartner)



Architetture C/S 3-tier

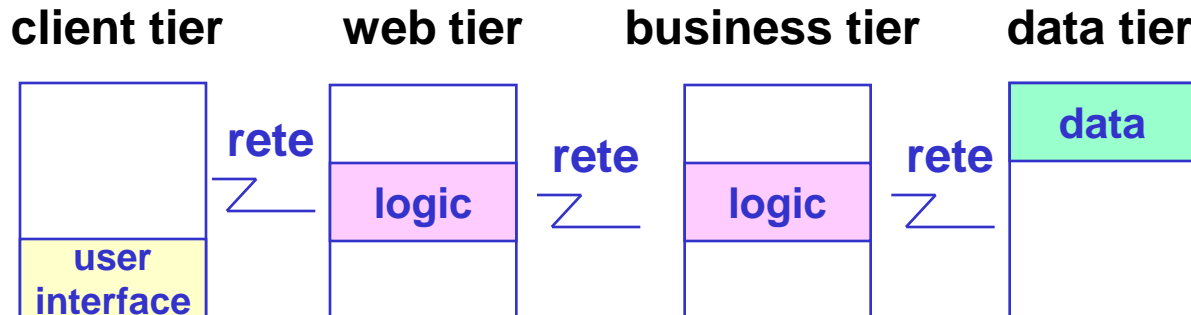
- ❑ Le architetture C/S 2-tier hanno limiti di **scalabilità e di flessibilità**
- ❑ Per superare questi limiti si può inserire un tier fisico intermedio (middle machine), con varie possibili funzioni:
 - ❖ distribuzione del carico su **più back end**
 - ❖ **filtro** (per es. firewall)
 - ❖ **conversione** di protocollo, accesso a legacy systems (gateway es. verso telnet servers ecc.)
 - ❖ **accesso a più server** simultaneamente, con servizi a valore aggiunto

Esempi di architetture C/S 3-tier



Architetture C/S multi-tier

- ❑ Per aumentare ulteriormente la flessibilità e per sistemi molto complessi, può essere utile aumentare ulteriormente il numero dei tier fisici
- ❑ Esempio di architettura 4-tier:



Argomenti studiati nel corso

- ❑ Esempi di vari tipi di componenti middleware, incluso le loro API, e come usarli per costruire applicazioni distribuite
- ❑ In particolare
 - ❖ Network programming (linguaggio C; socket)
 - ❖ Web programming + DB (linguaggi PHP, Javascript)
- ❑ Assoluta attenzione allo sviluppo di applicazioni distribuite **efficienti, robuste e sicure**
- ❑ Utilizzo di diversi tipi di API
 - ❖ sincrone e asincrone (cenni)