Sistemi distribuiti

Generalizzare l’architettura client-server

Sistema distribuito

Un sistema distribuito è un insieme di applicazioni logicamente indipendenti che collaborano per il perseguimento di obiettivi comuni attraverso una infrastruttura di comunicazione hardware e software.

Dall'esterno di un sistema distribuito appare un'unica entità.

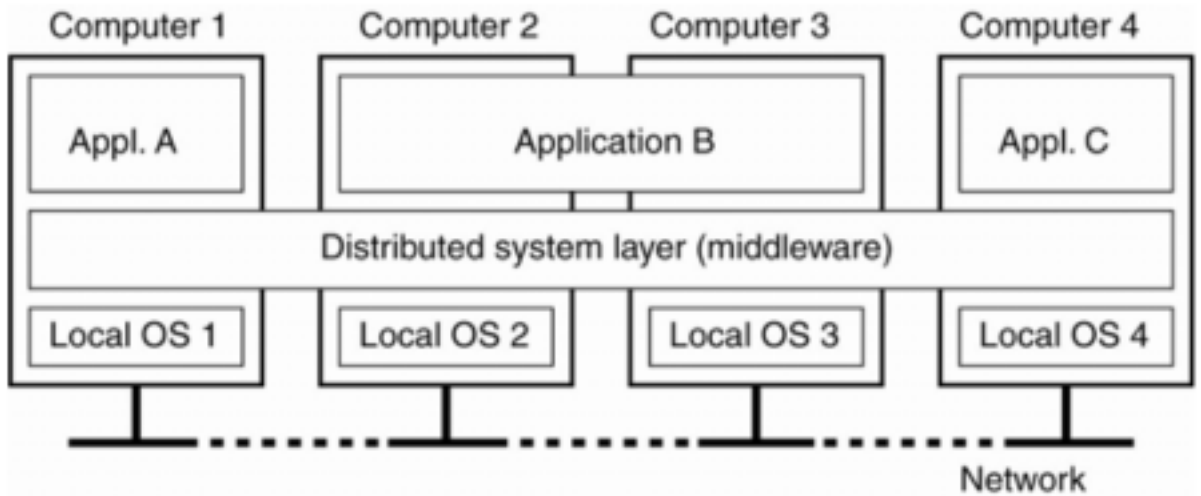
Le applicazioni che costituiscono un sistema distributo sono:

- CLIENT: cliente utilizzatore di servizi messi a disposizione da altre applicazioni
- SERVER: servente fornitore di servizi usati da altre applicazioni
- ACTOR: a seconda del contesto nel sistema assume il ruolo di cliente o di servente.

– Dall'esterno visione di un'unica entità

Necessaria la collaborazione tra componenti del sistema!

Middleware



Perché i sistemi DISTRIBUITI?

- **Affidabilità**: ridondanza intrinseca, tolleranza ai guasti.
- **Integrazione**: si integrano componenti eterogenei (hw o os). Retrocompatibilità, possibilità di connettere dispositivi di nuova generazione con legacy system. (es: Web).
- **Trasparenza**: il sistema distribuito è percepito come un unico sistema di elaborazione. Secondo ISO-10746 (Ref. Model of Open Distributed Processing)
  - **Trasparenza all'accesso**: nasconde le differenza nella rappresentazione dei dati e nelle modalità di accesso alle risorse. L'accesso alle risorse locali e remote avviene usando le stesse identiche operazioni, in modo unico ed uniforme
  - **Trasparenza all'ubicazione**: nasconde dove è localizzata una risorsa. Ad es. l'URL nasconde l'indirizzo IP. Trasparenza all'accesso ed all'ubicazione=trasparenza di rete.
  - **Trasparenza alla migrazione**: nasconde l'eventuale spostamento (logico o fisico) di una risorsa senza interferire sulla sua modalità di accesso

Perché i sistemi DISTRIBUITI?

- **Trasparenza al riposizionamento:** nasconde la possibilità di spostare una risorsa mentre è in uso
- **Trasparenza alla replica:** nasconde la replica di una risorsa
  - Ogni istanza della risorsa deve avere lo stesso nome
  - Supporto di trasparenza alla replica e trasparenza all'ubicazione
- **Trasparenza alla concorrenza:** nasconde la condivisione di una risorsa da parte di molti utenti contemporaneamente. Ad es. accesso contemporaneo di più utenti alla stessa tabella di una base di dati condivisa. L'accesso concorrente ad una risorsa condivisa deve lasciarla in uno stato consistente: ad es. meccanismi di locking (blocco).
- **Trasparenza al guasto:** nasconde il malfunzionamento e la riparazione di una risorsa

Perché i sistemi DISTRIBUITI?

- **Economicità**: meno costosi dei sistemi centralizzati.
- **Connettività e collaborazione**: possibilità di condividere risorse hw e sw.
- **Apertura**:
  - **Interoperabilità**: implementazioni diverse e di diverso tipo possono coesistere per comporre un sistema distribuito.
  - **Portabilità**: le appliazioni possono essere trasportate su host diversi utilizzando la medesima interfaccia.
  - **Ampliabilità**: semplicità nell'aggiungere componenti hardware o software

Scalabilità = il sistema piò essere AMPLIATO

Perché i sistemi DISTRIBUITI?

- **Prestazioni e scalabilità**: la crescita di un sistema distribuito, aggiungendo nuove risorse, fornisce un miglioramento delle prestazioni e permette di sostenere gli aumenti di carico (scalabilità orizzontale).
- **Tolleranza ai guasti**: garantita dalla replica delle risorse.

Svantaggi

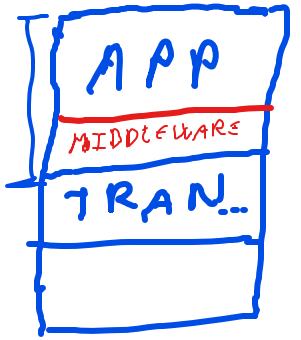
- **Software:**
  - **Software per i protocolli di comunicazione -> TCP/IP!!!**
  - **Sviluppo architetture web**
  - **Diffusione di linguaggi multi-piattaforma**

} sono superati

- **Complessità: dovuta all'hardware distribuito**
- **Sicurezza**
- **Comunicazione**

Python
Java Script (server = node.js)
PHP

Modello Client-Server e Middleware



software che sta in mezzo tra il Client e il Server

Per alleggerire il carico elaborativo dei Server sono stati introdotti strumenti di **Middleware**: uno strato di software che si colloca sopra al sistema operativo, ma sotto alla applicazione.

Il **Middleware** ha lo scopo di realizzare la comunicazione e le interazioni tra i diversi componenti software di un sistema distribuito!

DEF: il middleware è una classe di tecnologie SOFTWARE sviluppate per aiutare gli sviluppatori nella gestione della complessità e della eterogeneità presenti nei sistemi distribuiti.

Il middleware si focalizza sulla comunicazione tra i componenti del sistema distribuito.

Definizione Microsoft

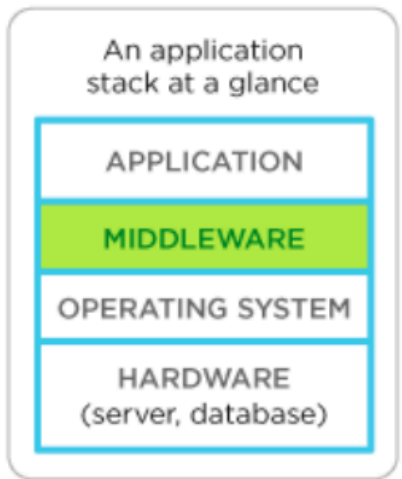
https://azure.microsoft.com/en-gb/overview/what-is-middleware/

Middleware is software that lies between an operating system and the applications running on it. Essentially functioning as a hidden translation layer, middleware enables communication and data management for distributed applications. It’s sometimes called plumbing, as it connects two applications together so that data and databases can be easily passed between the “pipe”. Using middleware allows users to perform such requests as submitting forms on a web browser, or allowing the web server to return dynamic web pages based on a user’s profile.

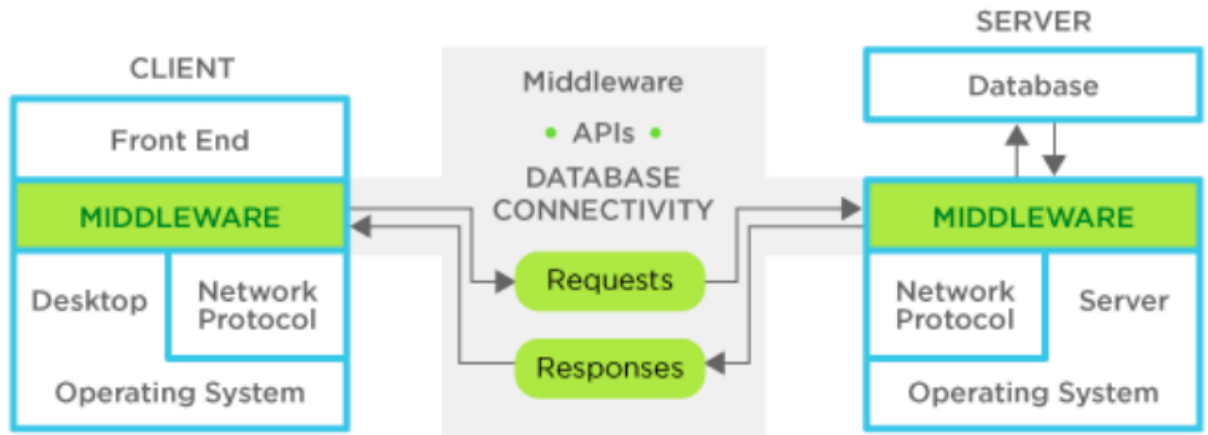
Common middleware examples include database middleware, application server middleware, message-oriented middleware, web middleware and transaction-processing monitors. Each program typically provides messaging services so that different applications can communicate using messaging frameworks such as simple object access protocol (SOAP), web services, representational state transfer (REST) and JavaScript object notation (JSON). While all middleware performs communication functions, the type a company chooses to use will depend on what service is being used and what type of information needs to be communicated. This can include security authentication, transaction management, message queues, applications servers, web servers and directories. Middleware can also be used for distributed processing with actions occurring in real time rather than sending data back and forth.

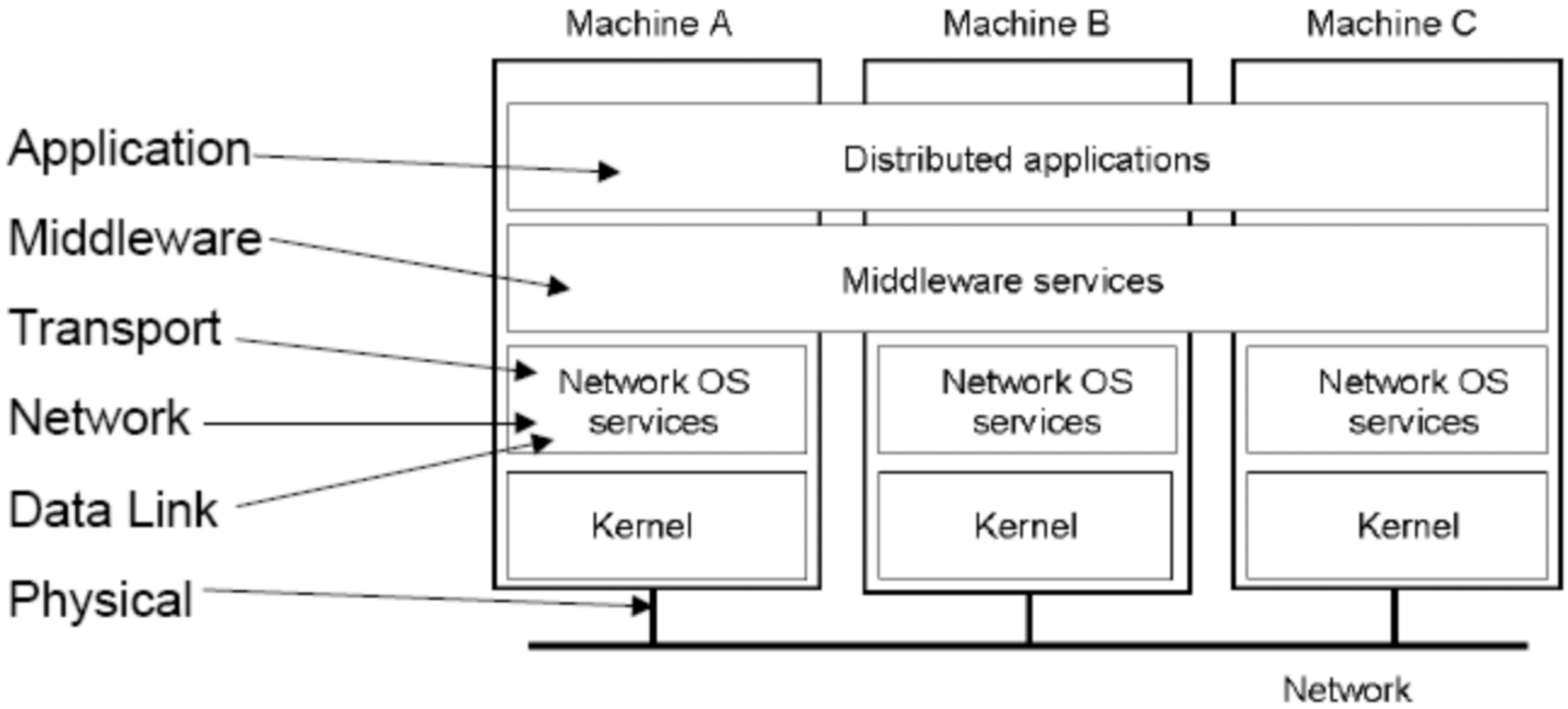
WHAT IS MIDDLEWARE?

upwork™



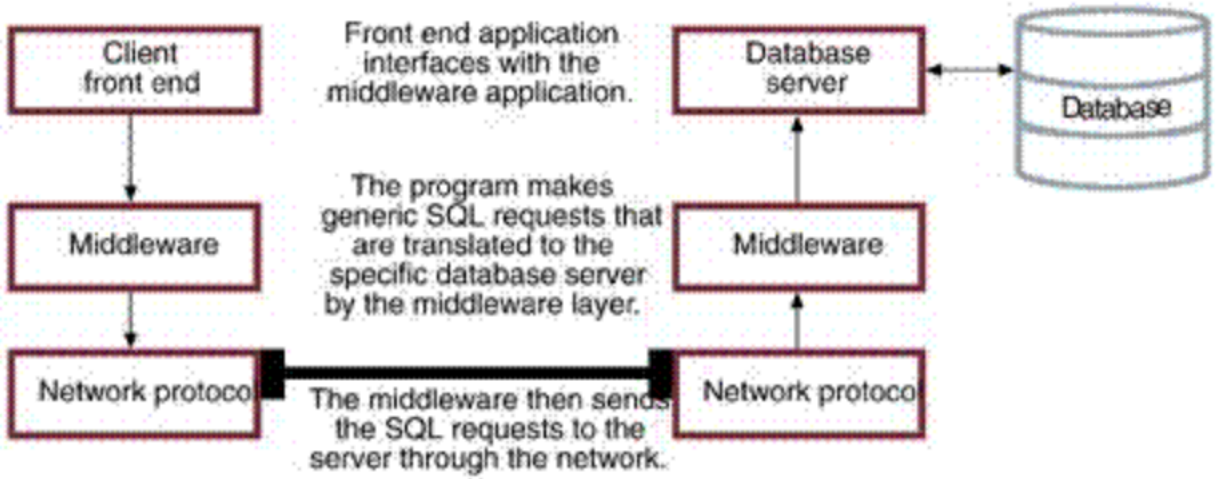
MiddleWare is any software that acts like "glue" between an application and its network. It controls the flow of information between an application and the server, database, and operating system.




Machine A
Machine B
Machine C
Application
Middleware
Transport
Network
Data Link
Physical
Distributed applications
Middleware services
Network OS services
Network OS services
Network OS services
Kernel
Kernel
Kernel
Network

Esempi:

1)



2) Se voleste fare un client UDP multipiattaforma (Linux, Windows)?

Altri esempi…

- **Database access technology - e.g ODBC (Open DataBase Connectors)**
- **Java’s database connectivity API : JDBC**
- **Remote computation products - e.g ONC RPC, OSF RPC and RMI (Java Remote Method Invocation)**

Architetture Client-Server

Le architetture client-server sono organizzate a strati (=tier)!!!

Ogni livello corrisponde a un nodo o un gruppo di nodi su cui è distribuito il sistema.

Ciascun livello funziona da server per i suoi client nel livello precedente e da client per il livello successivo.

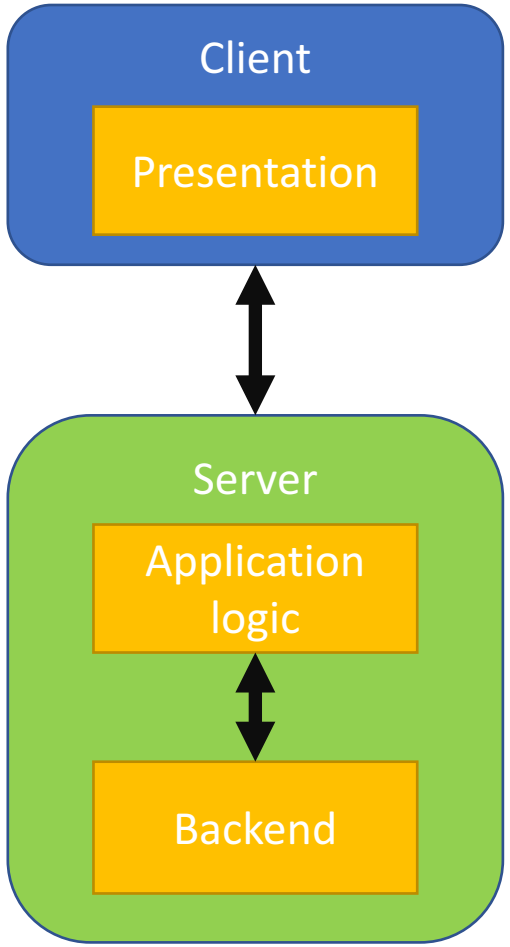
Si possono individuare 3 tipo principali di funzionalità che corrispondono ad una struttura in 3 strati:

1. **Front-end**: interfaccia verso l'utente.
2. **Application logic**: logica applicativa.
3. **Back-end**: detto anche data-tier fa accesso alle risorse e ai dati.

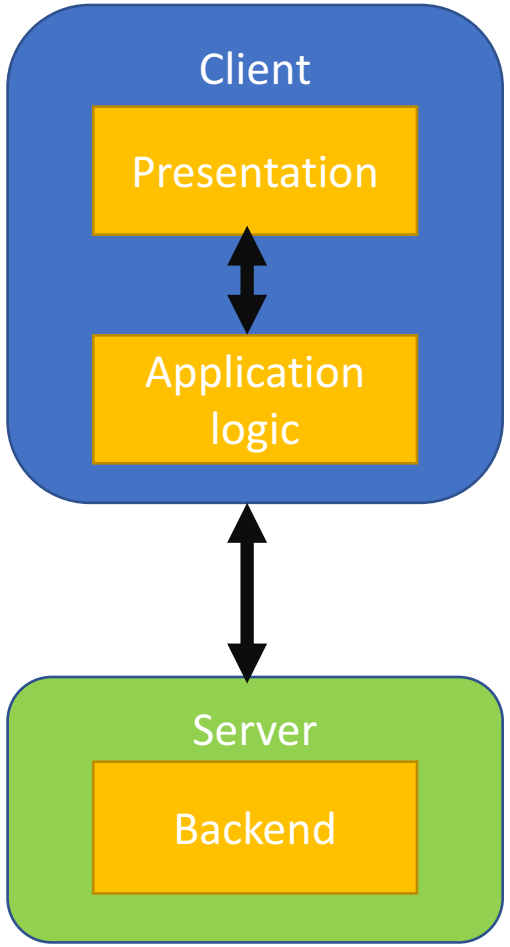
A seconda di quali funzionalità si attribuiscono al client sono possibili diverse realizzazioni.

Architettura a 2 livelli

Thin Client

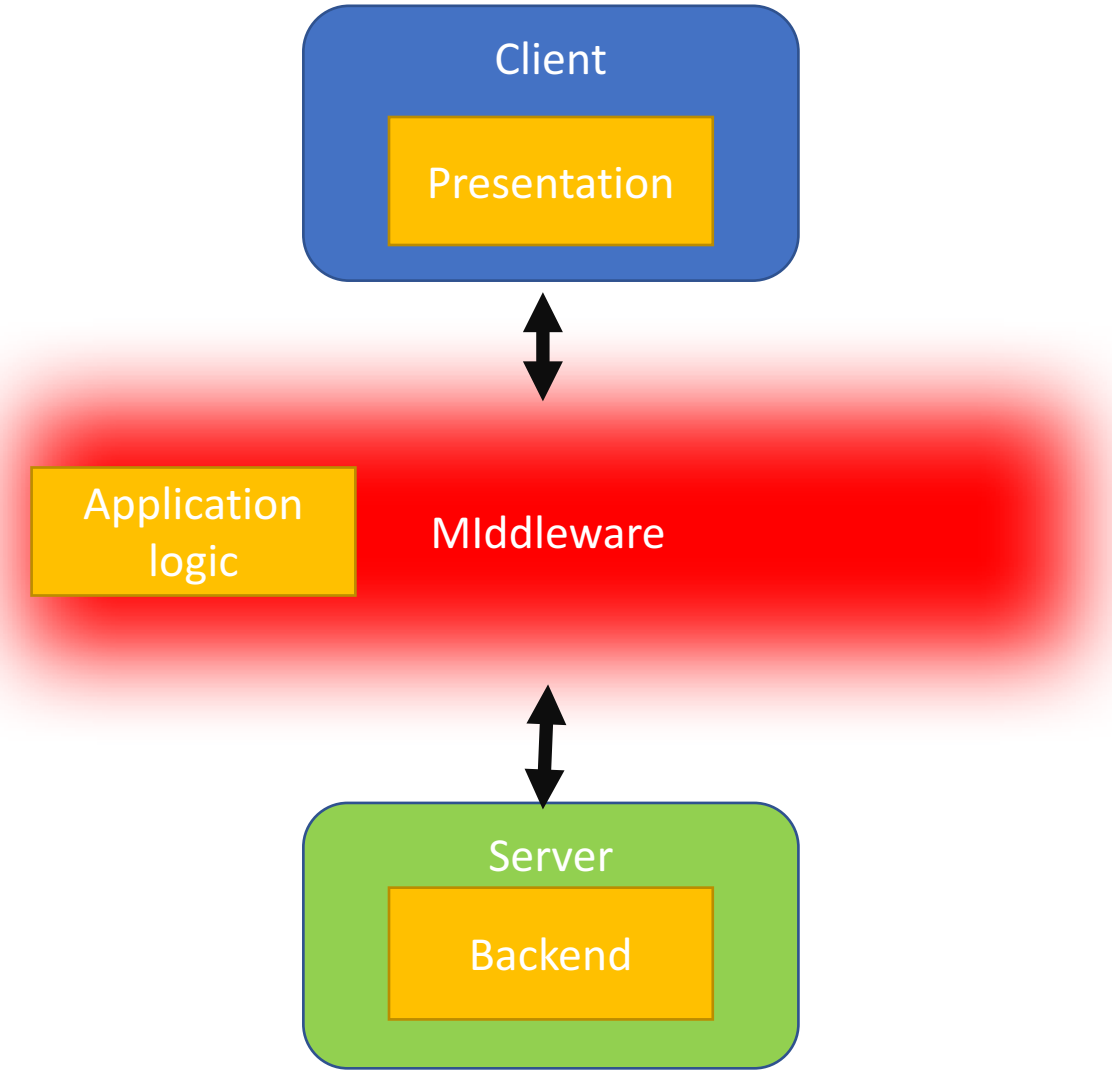


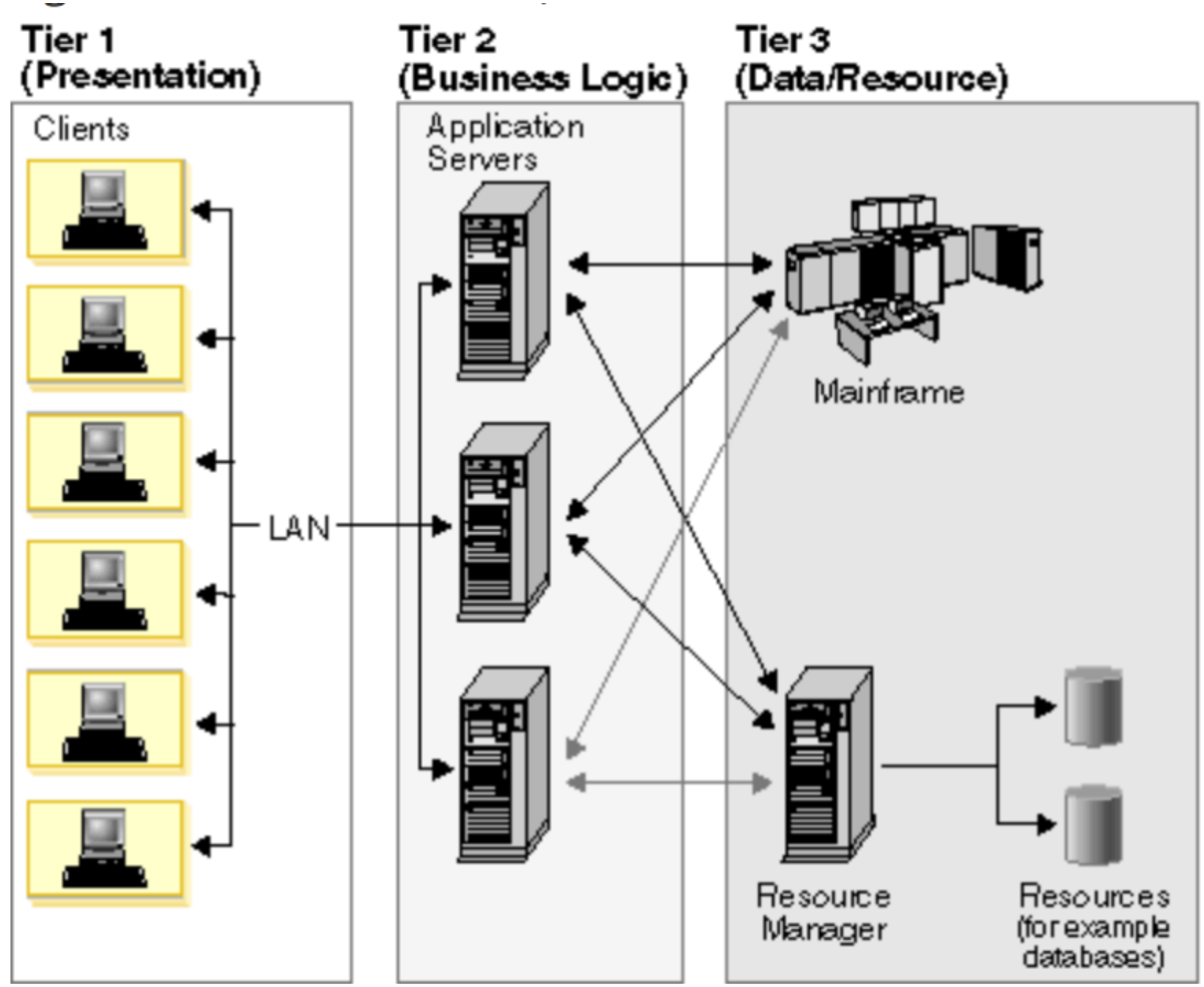
Thick Client



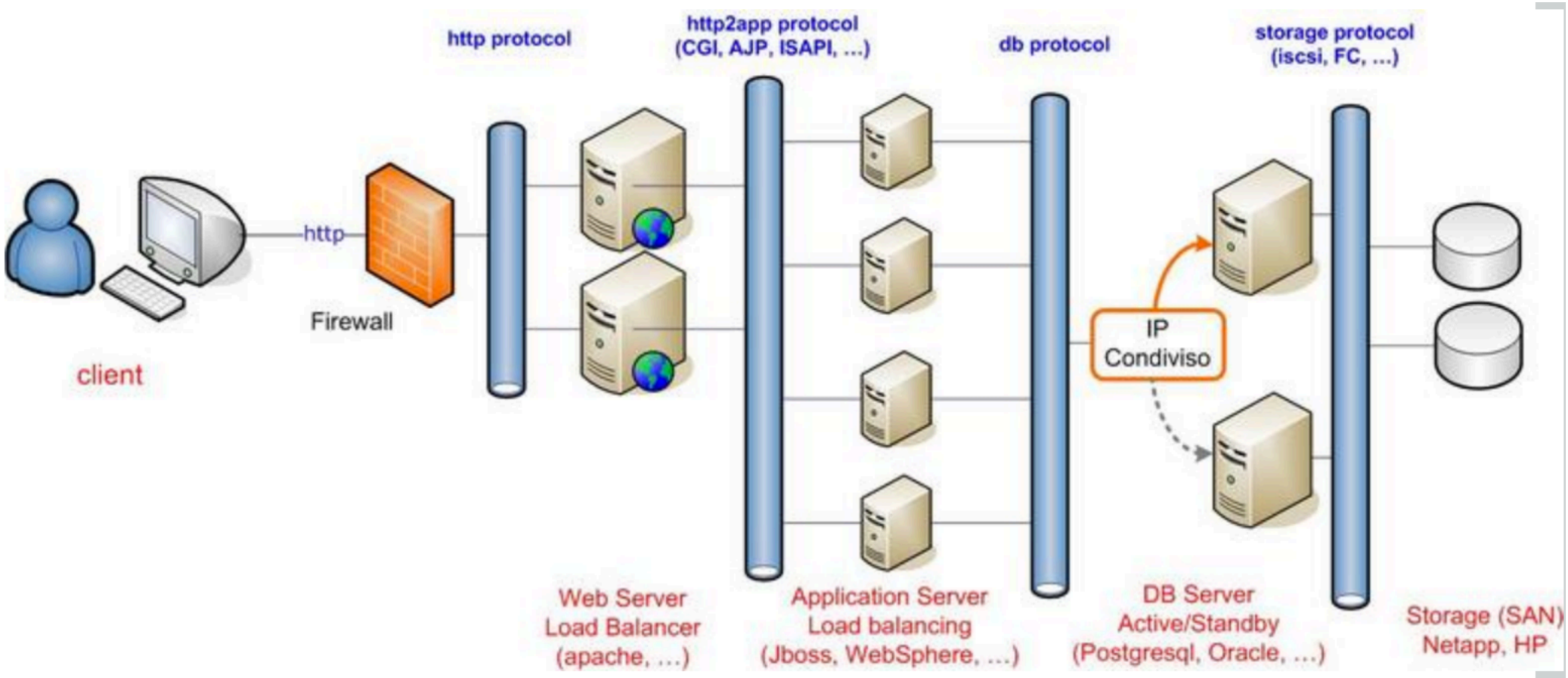
Architettura a 3 livelli

Il limite della architettura a 2 livelli è che il server deve gestire la connessione e lo stato della sessione di ciascun client -> limite sul numero massimo di client.





Domanda: chi sono i client, gli actor e i server?



Domanda: chi sono i client, gli actor e i server?