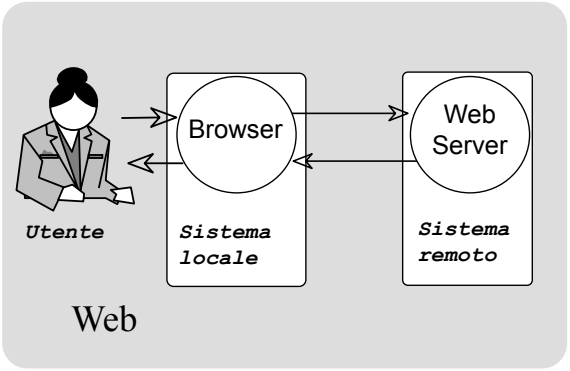
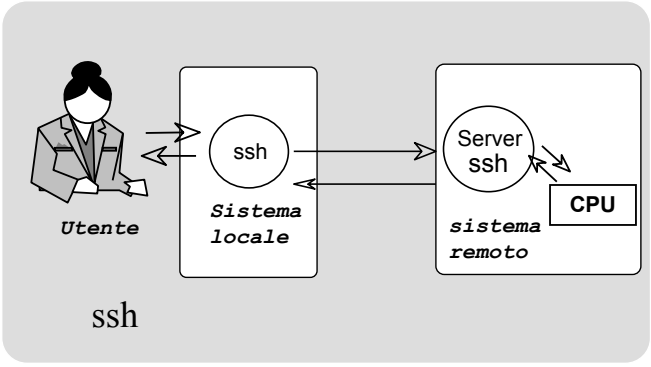
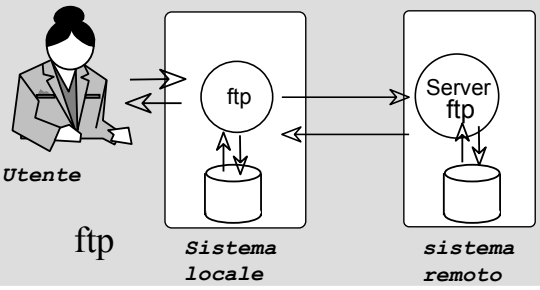
Internet e il Web come insieme di servizi





Tutti questi servizi sono basati sul modello Client/Server



Il modello Client/Server: una definizione operativa

Il modello Client/Server prevede due entità:

- l'entità **Client** che richiede il servizio
- l'entità **Server** che eroga il servizio

La parte Client del servizio:

- È un processo lanciato direttamente dall'utente (uno per ogni richiesta di servizio)
- Contatta attivamente il Server (uno per volta)
- Non richiede dispositivi dedicati o sistemi operativi sofisticati

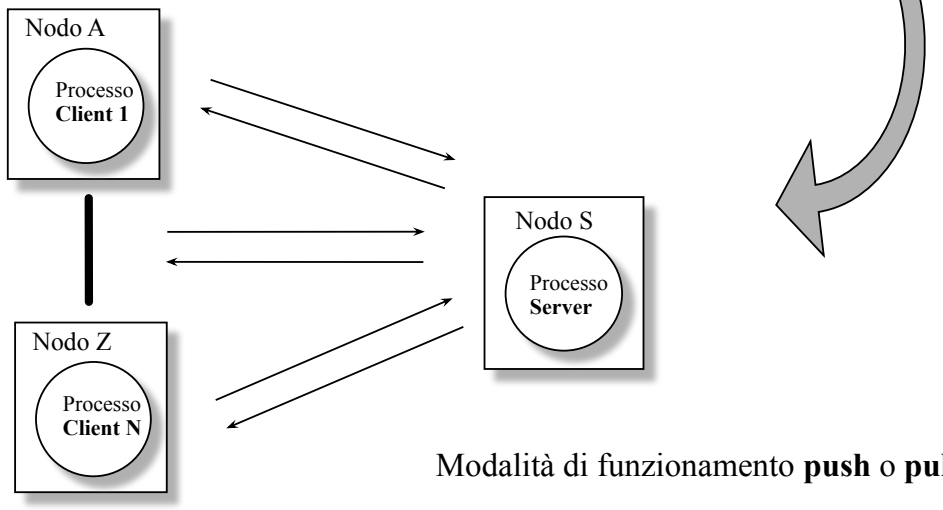
La parte Server del servizio:

- È un processo specializzato dedicato all'erogazione del servizio
- Può servire molti Client contemporaneamente
- È avviato automaticamente al boot della macchina (demone)
- È eseguito su un calcolatore "condiviso"
- Attende passivamente le richieste dei Client
- Accetta richieste da molti Client ma offre un solo servizio
- Richiede una macchina potente e un sistema operativo sofisticato

Il modello Client/Server

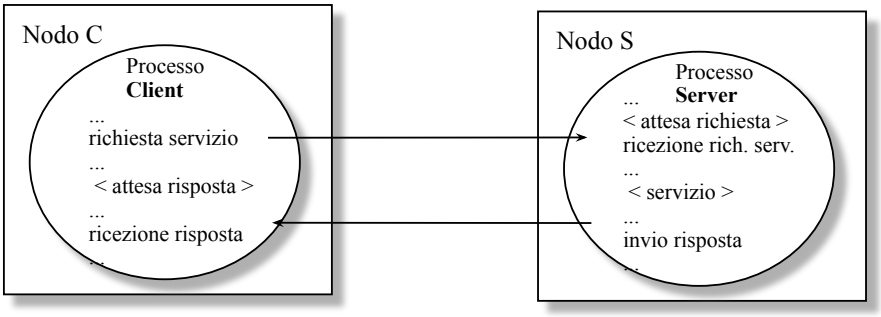
È un modello di comunicazione diretta e **asimmetrica**, **molti a uno**

Il Cliente designa esplicitamente il destinatario
Il Servitore non esprime il nome del processo con cui desidera comunicare



Modalità di funzionamento **push** o **pull**

Il modello Client/Server



Il modello Client/Server risolve il problema del **rendez-vous** (sincronizzazione iniziale tra i processi comunicanti) definendo il **Server** come un processo **sempre in attesa** di richieste di servizio.

Si semplifica in questo modo il protocollo di comunicazione sottostante che non deve occuparsi di attivare un processo alla ricezione di un messaggio.

Si noti che useremo spesso primitive di comunicazione sincrone/bloccanti.

Identificare i servizi

I Client devono poter **specificare il servizio** desiderato senza ambiguità

A questo scopo:

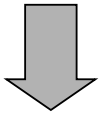
- I Server **registrano** il servizio (rendono pubblico un identificatore)
- I Client devono reperire **l'identificatore** del servizio
- I Client devono usare l'identificatore per **contattare** il Server e usufruire del servizio

Il progetto del Client e del Server

Il Server deve accedere alle risorse del sistema:

- problemi di **autenticazione** utenti
- **autorizzazione** all'accesso
- **integrità** dei dati
- **privacy** delle informazioni

e deve gestire richieste contemporanee da molti Client (Server **concorrenti**)



Maggiore complessità di progetto dei Server rispetto ai Client.

Tipi di interazione tra Client e Server

Due tipi principali di interazioni:

- **connection oriented**, viene stabilito un canale di comunicazione virtuale prima di iniziare lo scambio dei dati (es. il servizio telefonico)
- **connectionless**, non c'è connessione virtuale, ma semplice scambio di messaggi (es. il sistema postale)

La scelta tra i 2 tipi dipende dal tipo di servizio che si vuole realizzare. Per esempio:
- ssh è logicamente connection oriented
- e-mail è logicamente connectionless

Attenzione: la scelta è influenzata anche da altri requisiti (es. **affidabilità**) e dagli strumenti tecnologici a disposizione (si vedranno le differenze tra TCP e UDP).

Tipi di interazione tra Client e Server

Client e Server sono processi diversi, con diversi possibili comportamenti:

- **Modello pull**. Il Client richiede l'informazione al Server e in genere si blocca e aspetta la risposta (simile a chiamata di procedura).
- **Modello push**. Il Client segnala il proprio interesse e poi fa altro. È compito del Server di inviare l'informazione se e quando disponibile. Il Server effettua l'invio (*push*) delle informazioni.

In alternativa al modello push, il Client potrebbe eseguire un *polling* non bloccante (ovverosia client richiede servizio e controlla esecuzione della richiesta ripetutamente, riprovando alla scadenza di un intervallo di tempo prefissato, fino all'ottenimento della risposta). Il modello pull è Server-driven e **sincrono** (bloccante) mentre il modello polling è Client-driven e **asincrono** (non bloccante).

Si noti che, nonostante adotti il modello pull, il modello polling ha semantica simile a quella del modello push. Infatti, in ambito Web funzioni interattive come il refresh automatico della timeline e la notifica di ricezione messaggi su un social network possono essere implementate in modo equivalente (e indistinguibile dal punto di vista dell'utente finale) usando sia l'uno che l'altro modello (dal punto di vista tecnologico, rispettivamente tramite «long» polling asincrono su HTTP e tramite WebSocket).

Lo STATO dell'interazione tra Client e Server

Uno degli aspetti centrali nella realizzazione di un'applicazione Client / Server è la tipologia dell'interazione per quanto riguarda la **gestione di richieste multiple di servizio**, che può essere di due tipi:

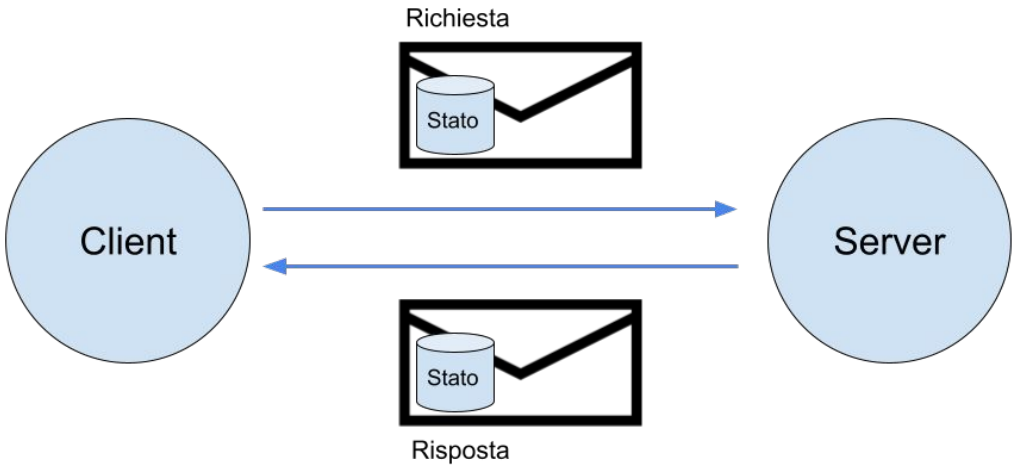
- **stateful**, cioè viene mantenuto lo stato dell'interazione e quindi una richiesta di servizio dipende da quelle precedenti (definizione di sessione);
- **stateless**, non si tiene traccia dello stato, ogni richiesta è completamente indipendente dalle altre e auto contenuta.

Lo stato dell'interazione è fortemente significativo nel caso di molteplici interazioni simili tra loro tra gli stessi attori.

Esempio: in caso di un servizio stateful che registra il voto assegnato a uno studente in N appelli di esame sostenuti con successo, informazioni comuni a tutte le richieste (come il numero di matricola dello studente) dovrebbero essere comunicate solo nella prima richiesta e non in quelle successive.

Lo stato è quindi una sintesi, memorizzata da una delle parti, di come l'attuale sessione di servizio sta procedendo. Una sorta di 'accordo predefinito' tra le entità interagenti che consente di facilitarne l'interazione.

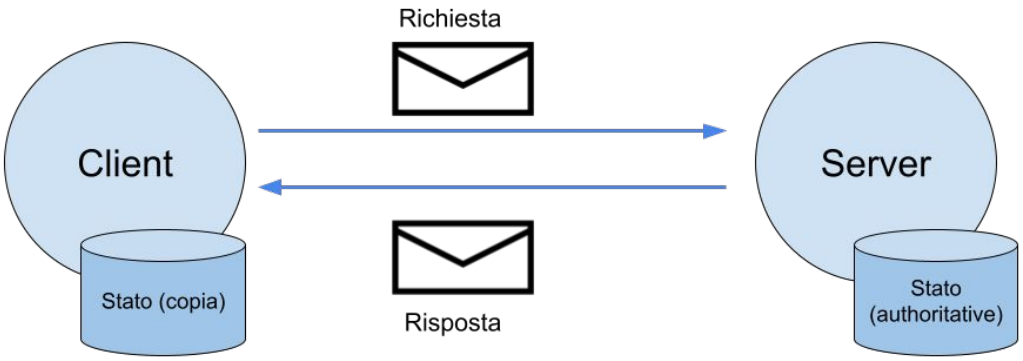
Servizi Stateless



Nei servizi stateless ogni messaggio deve contenere tutte le informazioni per svolgere la richiesta di servizio. Quindi, lo stato della sessione di servizio deve essere esplicitamente contenuto (tracciato) all'interno dei messaggi di richiesta e di risposta.

Lavorando con messaggi autocontenuti, i servizi stateless sono tipicamente più robusti (es. se cade una connessione non importa).

Servizi Stateful



Nei servizi stateful, Client e Server tengono traccia localmente, su un database, dello stato dell'iterazione. Tipicamente il database di riferimento è quello del Server (soprattutto per motivi di sicurezza, ma anche di performance).

Ogni messaggio può assumere che l'altra parte sia già a conoscenza dello stato di partenza, quindi si consumano meno risorse (banda, tempo di preparazione ed elaborazione messaggi).

Lo STATO dell'interazione tra Client e Server

Il Server quindi può essere stateless o stateful:

- un Server **stateful** ha migliore efficienza (dimensioni messaggi più contenute e migliore velocità di risposta);
- un Server **stateless** è più affidabile in presenza di malfunzionamenti (soprattutto causati dalla rete).

E per quanto riguarda la complessità di realizzazione?

- I modelli di interazione stateless semplificano il progetto del Server, ma la complessità viene ripartita sui Client e sull'infrastruttura di rete (ogni interazione deve specificare anche tutte le informazioni relative allo stato stesso, quindi la banda utilizzata dal protocollo è maggiore).
- I modelli di interazione stateful richiedono al Server di mantenere lo stato della interazione per ogni Client connesso, rendendo più difficile l'adozione di tecniche di scalabilità orizzontale basate sulla replicazione dei Server.

Lo STATO dell'interazione tra Client e Server

La scelta tra modello di interazione stateless o stateful deve tenere in conto anche (e soprattutto) delle caratteristiche dell'applicazione.

Un'interazione stateless è possibile **SOLO** se il **protocollo applicativo** è progettato con **operazioni idempotenti**.

Operazioni idempotenti producono sempre lo stesso risultato, per esempio, un Server fornisce sempre la stessa risposta a un messaggio M indipendentemente dal numero di messaggi M ricevuti dal Server stesso.

Esempio di operazioni I/O idempotenti e non

Per esempio, le system call read e write di Unix non sono idempotenti perché ogni chiamata va a cambiare l'I/O pointer che verrà usato per la chiamata successiva:

```
ssize_t read(int fildes, void *buf, size_t nbyte);
ssize_t write(int fildes, const void *buf, size_t nbyte);
```

Lo standard POSIX però introduce anche due funzioni equivalenti a read e write con semantica idempotente: pread e pwrite. In queste system call dobbiamo ogni volta fornire un parametro offset, ovverosia l'I/O pointer da usare per la chiamata:

```
ssize_t pread(int d, void *buf, size_t nbyte, off_t offset);
ssize_t pwrite(int fildes, const void *buf, size_t nbyte,
off_t offset);
```

Esempio stateless vs stateful: NFS

Per esempio, si considerino le differenze tra un file server stateless, come Network File System (NFS) versione 3, e stateful, come NFS versione 4.

NFS è un servizio basato su chiamate a procedura remota (RPC), ovverosia operazioni che eseguono lato server ma che sono invocabili remotamente dal cliente come se fossero una chiamata a funzione locale.

NFSv3 (RFC 1813, giugno 1995) è basato su operazioni **stateless**:

- Operazione LOOKUP idempotente per ottenere file handle (identificativo opaco)
  `file_handle = LOOKUP(dir_handle, filename)`
- Operazione READ idempotente (si deve sempre specificare lo I/O pointer `offset`)
  `data_and_metadata = READ(file_handle, offset, count)`
- Operazione WRITE idempotente (si deve sempre specificare lo I/O pointer `offset`)
  `result_metadata = WRITE(file_handle, offset, count, data)`
- Necessità del servizio ausiliario Network Lock Manager (NLM) per gestire file locking (fornito da due processi demone: rpc.statd e rpc.lockd)

Esempio stateless vs stateful: NFS

NFSv4 (RFC 3010, dicembre 2000) è basato su operazioni **stateful**:

- LOOKUP sostituita da operazioni OPEN e CLOSE stateful per aprire e chiudere file
- Funzioni di file locking implementate direttamente nel protocollo da operazioni OPEN, READ e WRITE (NLM non è più neessario e infatti è stato rimosso)
- Nuova operazione COMPOUND rende possibile l'aggregazione di più richieste di I/O in un solo messaggio

Operazioni di I/O più robuste in NFSv3 (ma NLM rappresenta un grosso problema). Integrazione locking nel protocollo permette a NFSv4 di fornire migliori prestazioni.

Gestione stato: livello di protocollo applicativo o di applicazione?

Si noti che non è strettamente necessario gestire lo stato a livello di protocollo applicativo. Si può anche implementare la gestione dello stato a livello di applicazione al di sopra di un protocollo applicativo stateless.

Nel servizio File Transfer Protocol (FTP), lo stato della sessione corrente di servizio viene modificato e tracciato a livello di protocollo applicativo. L'elemento forse più importante tracciato dallo stato del servizio è la working directory su cui si sta operando lato server. (Comando PWD per visualizzare il percorso della working directory e comandi CWD e CDUP per modificarlo.)

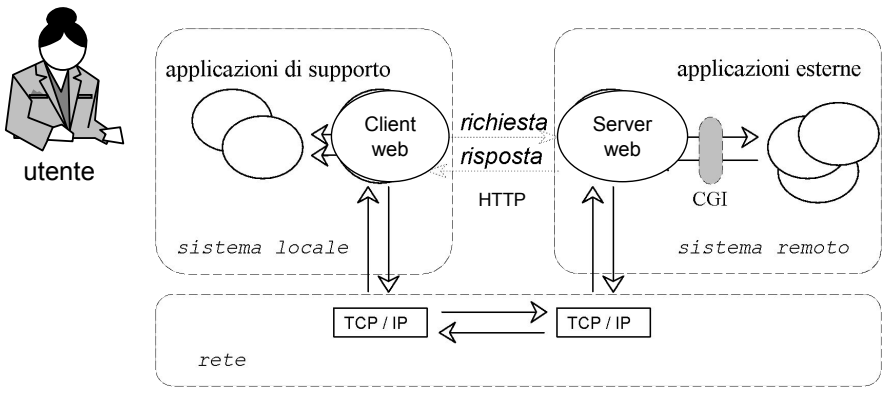
Altri servizi, come il World Wide Web (WWW), invece sono basati su un protocollo applicativo stateless. Quindi, un eventuale stato della sessione di servizio deve essere mantenuto a livello di applicazione.

(Si noti in ogni caso che sia FTP che WWW fanno uso del protocollo di comunicazione TCP, che è stateful.)

Il problema dello stato nel Web

Tutto il **WWW** è modellato sul paradigma Client/Server, il browser è il Client, le pagine Web sono ospitate sui Server.

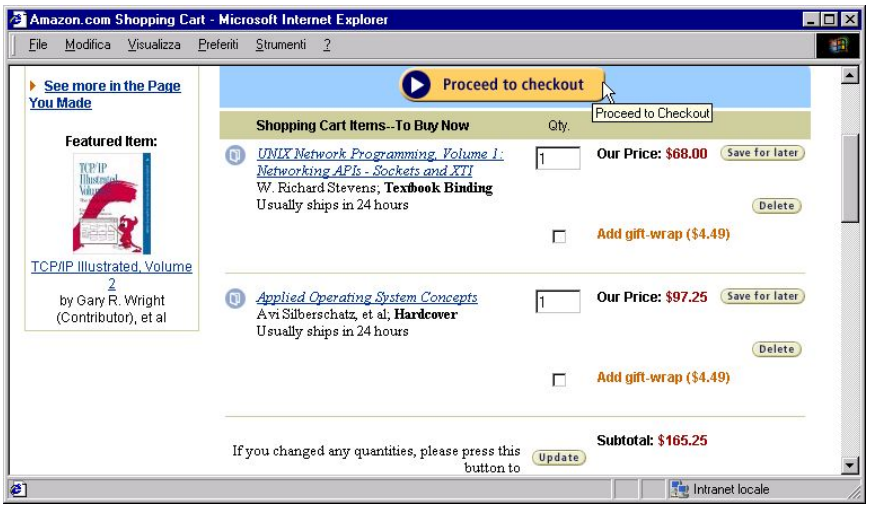
L'interazione tra browser e Server Web è basata sul protocollo **HTTP che è stateless**, ogni richiesta di pagina è completamente indipendente dalle precedenti.



Lo stato nel Web

Ci sono servizi Web che devono essere offerti solo a certi utenti (es. WebMail) e altri che richiedono più "passi" (es. un carrello della spesa)

Tali servizi hanno bisogno di mantenere lo stato dell'interazione tra Client e Server.



Lo **stato viene mantenuto a livello di applicazione,** al di sopra di HTTP (uso di cookie e oggetti sessione).

Stato permanente o soft

I modelli con stato hanno il server che può / deve mantenere traccia della interazione
Per quanto tempo e con che costi?

Si distingue lo stato in base alla durata massima:

- **stato permanente** mantenuto per sempre
- **stato soft o a tempo** che rimane per un tempo massimo

Si pensi a un server Web che deve mantenere le risorse per reggere tutte le richieste dei clienti che hanno acceduto (sessioni in atto) o a un server che deve riconoscere tutti i clienti che sono autorizzati ad accedere (username e password) tramite tabelle di riconoscimento.

La concorrenza nell'interazione tra Client e Server

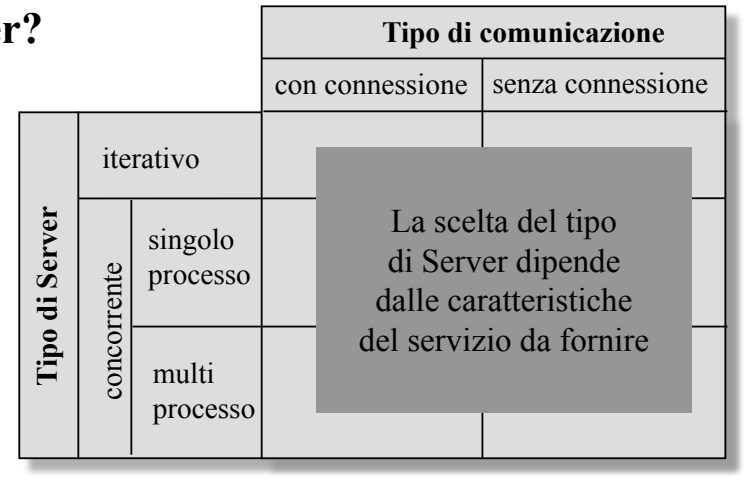
Lato Client
I Client sono programmi sequenziali, eventuali invocazioni concorrenti supportate dal sistema operativo multitasking.

Lato Server
La concorrenza è cruciale per migliorare le prestazioni di un Server.

Un **Server iterativo** processa le richieste di servizio una alla volta.
Possibile basso utilizzo delle risorse, in quanto non c'è sovrapposizione tra elaborazione e I/O.

Un **Server concorrente** gestisce molte richieste di servizio concorrentemente, cioè una richiesta può essere accettata anche prima del termine di quella (o quelle) attualmente in corso di servizio.
Migliori prestazioni ottenute da sovrapposizione elaborazione e I/O.
Maggiore complessità progettuale.

Quale tipo di Server?



Affidabilità come terza dimensione della tabella.

La scelta del tipo di Server è vincolata dai **protocolli** e dalla **tecnologia** realizzativa.

Per esempio, in Unix è facile realizzare un Server concorrente generando un processo nuovo (fork) per ogni richiesta di servizio (Server concorrente multi processo).

Le Prestazioni di un Server

Dal punto di vista del Client, definiamo il tempo di risposta T_R come il ritardo totale tra la spedizione della richiesta e l'arrivo della risposta dal server.

$T_R = T_S + 2\,T_C + T_Q$ dove

T_S *è il tempo di elaborazione (di servizio) di una singola richiesta*
T_C *è il tempo di comunicazione medio*
T_Q *è il tempo di accodamento medio*

Con **lunghe code di richieste**, il tempo di risposta può diventare anche molto maggiore del tempo di elaborazione della richiesta

Le Prestazioni di un Server Iterativo

Nel caso di **Server iterativo**, che risponde a una singola richiesta alla volta e accoda le altre, il tempo di risposta è circa proporzionale alla lunghezza della coda.

problemi:
- limitare la lunghezza della coda
- le richieste a coda piena vengono rifiutate

Le Prestazioni di un Server Concorrente

Caso di **Server concorrente**, che risponde a più richieste contemporaneamente

Concorrenza riesce a migliorare il tempo di risposta:

- se la risposta richiede un tempo significativo di attesa di I/O o di sospensione del servizio con possibilità di interleaving;
- se le richieste richiedono tempi di elaborazione molto variabili;
- se il Server esegue in un sistema multiprocessore, cioè con reale parallelismo

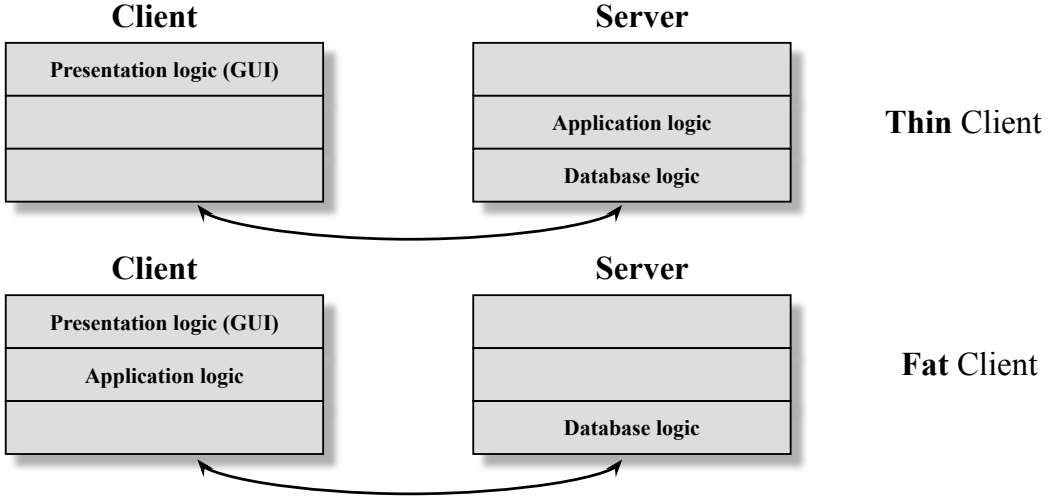
$T_R = T_S + 2\,T_C + T_Q + T_I + T_G$;

T_C tempo di comunicazione medio
T_Q tempo di accodamento medio (a volte trascurabile)
T_I tempo di interleaving con altri servizi concorrenti
T_G tempo di generazione di un eventuale servitore

Vedremo molti possibili **schemi diversi di realizzazione** di un Server concorrente.

Il progetto del Client e del Server

Quale parte di logica applicativa mettere sul Client o sul Server? (Fat Client Vs. Thin Client)
Considerazioni di configurazione del sistema, di carico sul server, di traffico di rete

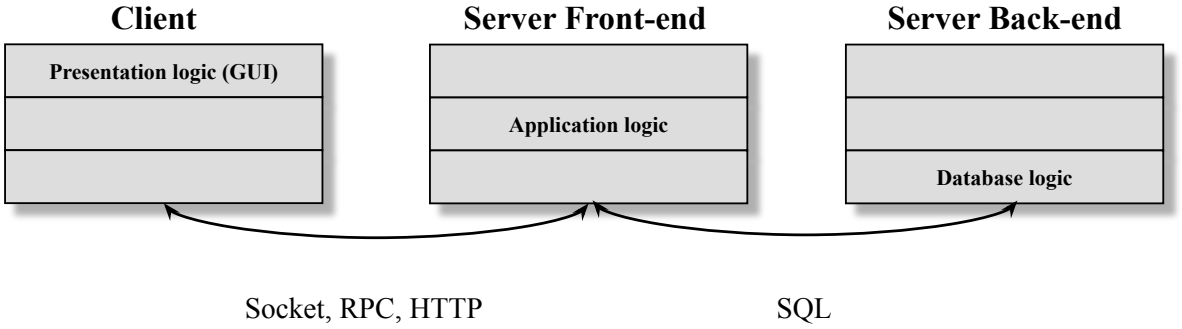


Le applicazioni Web moderne (Single Page Web Applications, Progressive Web App, ecc.) tendono – molto spesso solo per "cargo culting" – a seguire il modello Fat Client.

Il progetto del Client e del Server

Nonostante il client continui ad avere la percezione di interagire con un unico server, le funzioni svolte dal server possono essere suddivise tra componenti software di vari livelli (o *tier*), per **distribuire il carico di lavoro** su diverse macchine.

La seguente figura mostra un'architettura 2-tier, con un server di front-end e uno di back-end:



Come vedremo, nelle moderne applicazioni Web si utilizzano server con architetture **3-tier**, a cui solitamente si aggiungono **numerosi** altri elementi per ottimizzare le performance (cache, CDN) o per la gestione di servizi di background (reporting PDF).