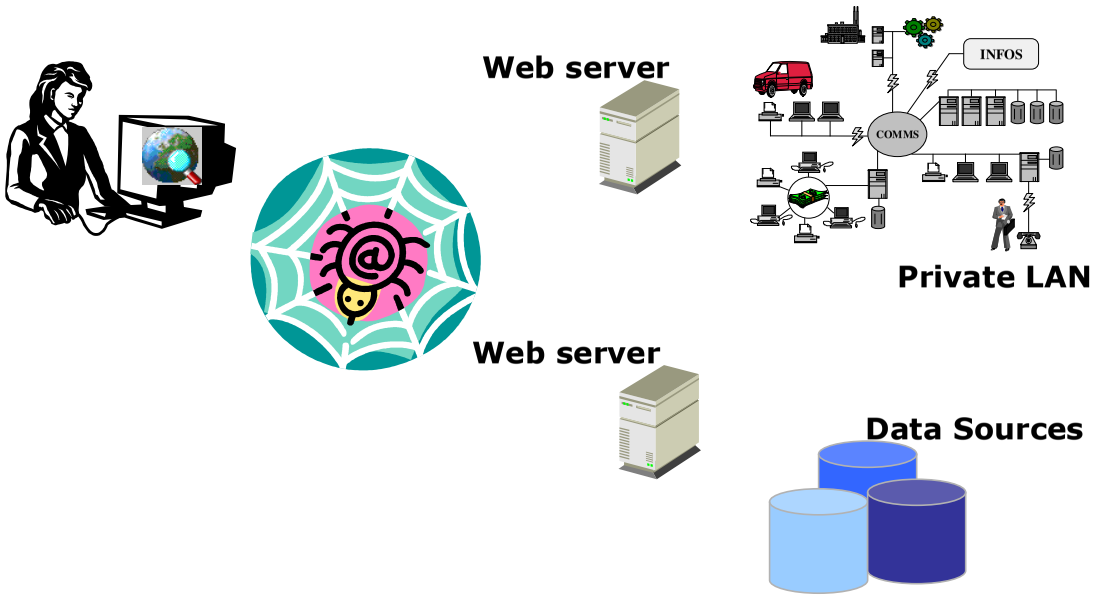
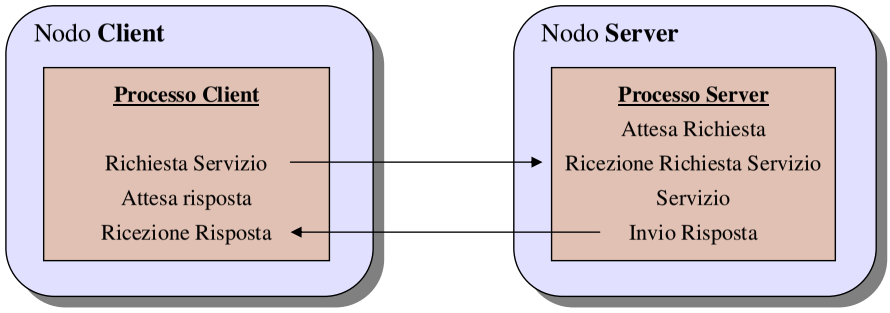
Modelli ed Architetture



Il Modello Client Server

- Il modello Client/Server prevede due entità:
  - l'entità **Client** che richiede il servizio
  - l'entità **Server** che offre il servizio



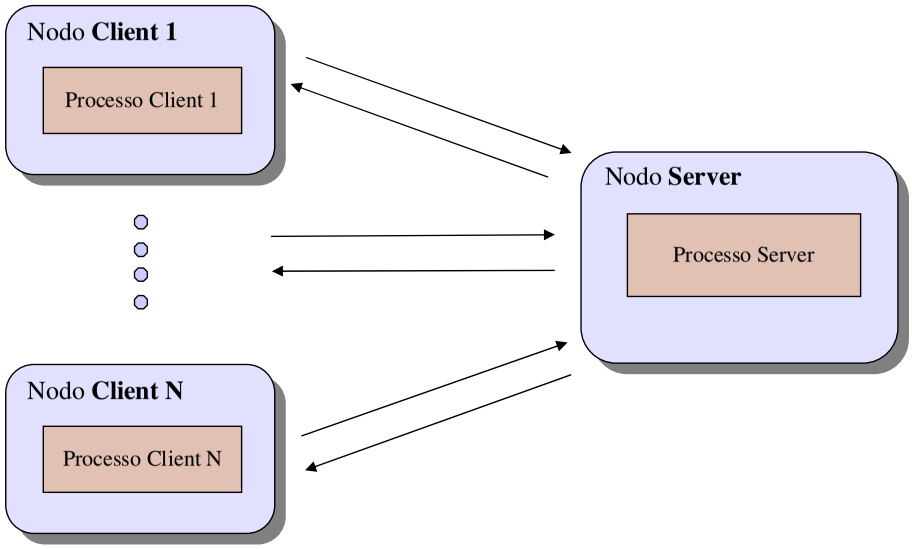
Il modello Client/Server risolve il problema del **rendez vous** (quando sincronizzare i processi comunicanti) definendo il **Server** come un processo **sempre in attesa** di richieste di servizio.

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

Il Modello Client Server

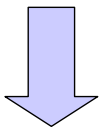
È un modello di comunicazione **asimmetrica, molti:1**

Il Cliente designa esplicitamente il destinatario
Il Servitore risponde al processo che ha effettuato una richiesta



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
- Il Server 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:
  - **interazione connection oriented**, viene stabilito un canale di comunicazione virtuale prima di iniziare lo scambio dei dati (es. connessione telefonica)
  - **interazione connectionless**, non c'è connessione virtuale, ma semplice scambio di messaggi (es. il sistema postale)
- La scelta tra i due tipi dipende dal tipo di applicazione e anche da altre caratteristiche proprie del livello di comunicazione sottostante. Per esempio, in Internet il livello di trasporto è TCP oppure UDP:
  - **TCP** è con connessione, inoltre è reliable (affidabile) e preserva l'ordine di invio dei messaggi
  - **UDP** è senza connessione, non reliable e non preserva ordine messaggi

Lo Stato dell'interazione tra Client e Server

- L'interazione tra un Client e un Server può essere di due tipi:
  - **stateful**, cioè esiste lo stato dell'interazione e quindi un messaggio dipende da quelli precedenti
  - **stateless**, non si tiene traccia dello stato, ogni messaggio è indipendente dagli altri
- Lo stato dell'interazione memorizzato nel Server (che quindi può essere stateless o stateful):
  - Un Server **stateful** ha migliore efficienza (dimensioni messaggi più contenute e migliore velocità di risposta del Server, presenta però problemi di replicazione)
  - Un Server **stateless** è più affidabile in presenza di malfunzionamenti (soprattutto causati dalla rete) ed è più semplice da progettare

Lo Stato dell'interazione tra Client e Server

- La scelta tra server stateless o stateful deve tenere in conto anche (e soprattutto) le 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.
- **Quando si ha un'interazione stateful il Server deve poter identificare il Client.**

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 ed 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 ed I/O.
  - La gestione di processi concorrenti implica una analisi precisa della sincronizzazione nell'accesso alle risorse (principi di atomicità delle transazioni e di consistenza dei dati)

Programmazione Client Side e Server Side

- Dal mondo Web deriva la classificazione di **Programmazione** (e quindi esecuzione) **Client Side o Server Side**; si definisce:
  - **esecuzione Server Side**: elaborazione effettuata dalla entità server, insieme delle operazioni che devono essere completate al fine di generare l'output per il Cliente
  - **esecuzione Client Side**: elaborazione effettuata dalla entità client, insieme delle operazioni necessarie per la gestione ed eventuale visualizzazione dei dati ottenuto dal server.
- Nel mondo Web il contesto di esecuzione Client Side è il Browser il quale si occupa di interpretare i dati di output in formato html ottenuti dal server e visualizzarli graficamente. La visualizzazione non è statica, l'interattività è ottenibile attraverso lo sviluppo di applicazioni Client side. **Anche un Browser è un interprete**, è una macchina virtuale che mette in esecuzione le pagine web che ottiene a seguito di una request

I Sistemi Distribuiti ed il Web

- **Applicazione Distribuita:**
  Applicazione software realizzata attraverso la collaborazione di diverse entità in esecuzione su risorse computazionali fisicamente distinte.
- Un **Sistema Distribuito** è quindi un ambiente entro il quale possono essere operative una o più applicazioni distribuite.
- Il **mondo Web** da questo punto di vista è quindi un sistema distribuito:
  - Definisce un insieme di standard per la comunicazione (TCP/IP, HTTP/HTML, …)
  - Fornisce un insieme di servizi di supporto (DNS, NIC, Certification Authority, …)

A cosa serve una Applicazione Distribuita?

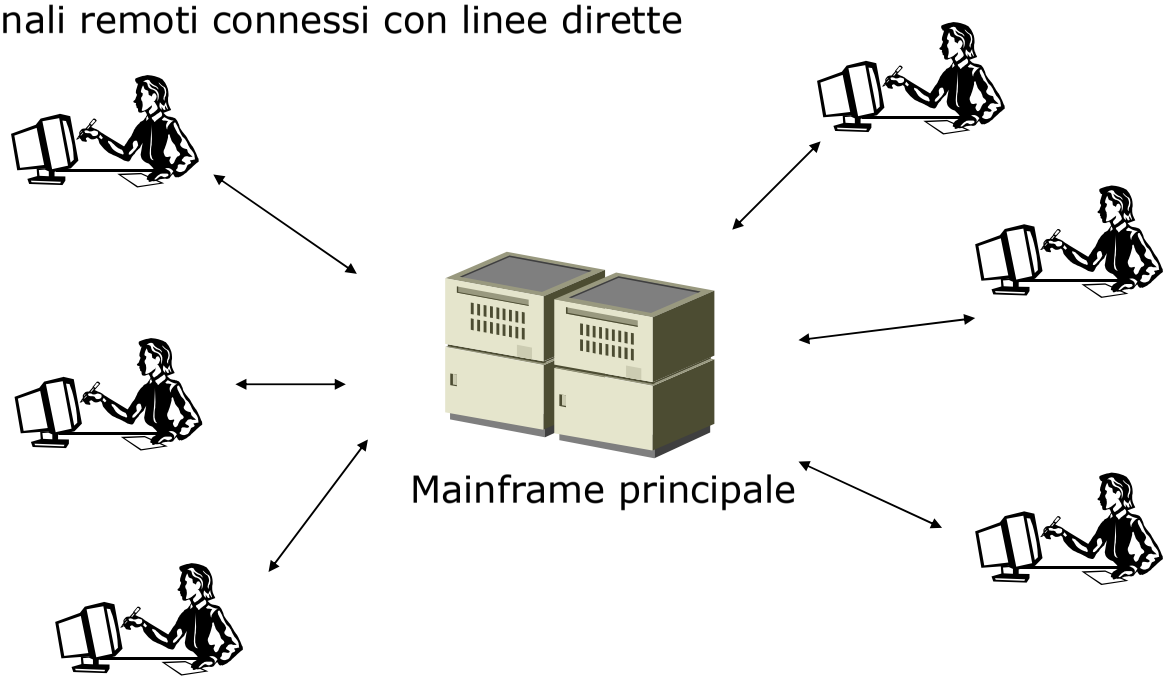
- **Condivisione delle Risorse:**
  - **Risorse Dati** (Database con le più diverse informazioni e tipologia e struttura di accesso)
  - **Risorse computazionali** (capacità di calcolo, memoria a breve e lungo termine)
  - **Risorse per l'accesso a periferiche e canali di comunicazione** (fax, posta elettronica, sms, comunicazione vocale (voice over IP), stampanti o altre periferiche…)
- **Applicazione principi di economia di scala.**

Vantaggi nelle Applicazioni distribuite: Sistema di biglietteria

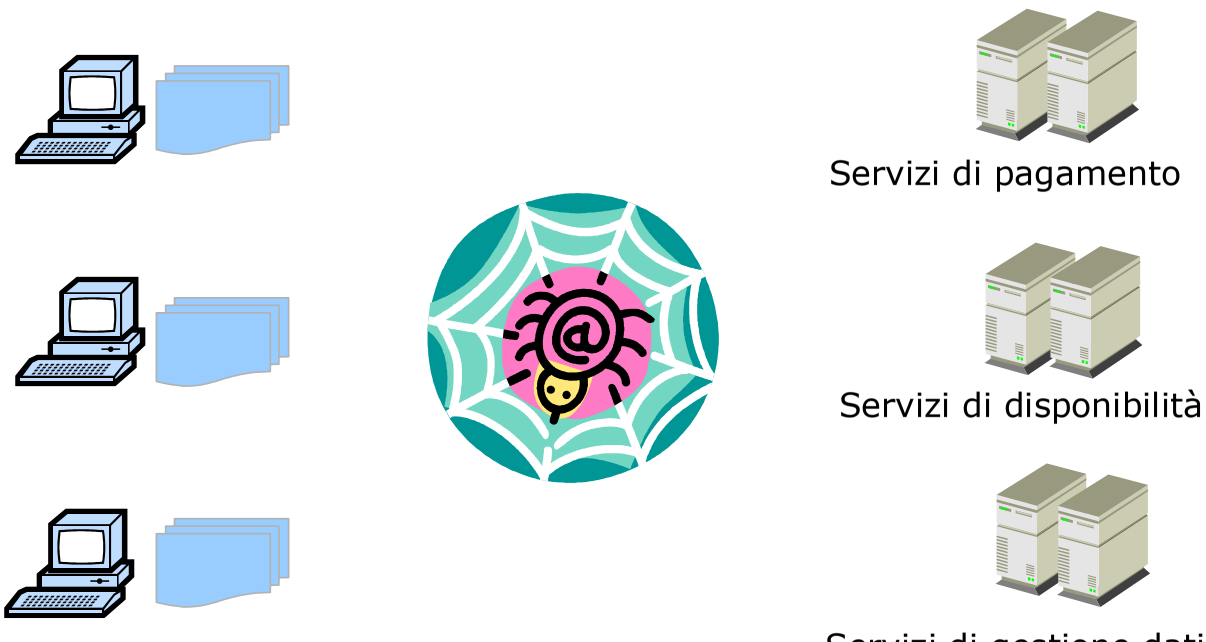
- Specifiche:
  - Fornire disponibilità posti in tempo reale
  - Permettere l'acquisto dei biglietti da diversi punti vendita dislocati in luoghi fisicamente separati
  - Alto volume di prenotazioni (es. aerei o treni)
- Necessità:
  - Fornitura di un elevato livello di servizio:
    - Interfacce rapide ed efficienti
    - Garanzia di continuità di servizio

Sistema Biglietteria
Soluzione Monolitica

Terminali remoti connessi con linee dirette



Sistema Biglietteria
Soluzione Distribuita



Sistema Biglietteria
Vantaggi e Svantaggi delle due Soluzioni

Soluzione Monolitica

Vantaggi:

- Sistema facile da implementare
- Una sola macchina (mainframe) che contiene tutte le informazioni

Svantaggi:

- Regge un numero definito di utenti
- Le interfacce sono molto semplici e spartane
- Un guasto nel server centrale provoca una perdita di servizio

Soluzione Distribuita

Vantaggi:

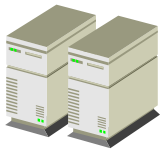
- è possibile affiancare diversi server in parallelo per aumentare le prestazioni e la tolleranza ai guasti
- I client possono presentare le informazioni ottenute dai server in modo indipendente, con grafica e strumenti per la semplificazione del lavoro.
- è possibile agganciare un numero di Client proporzionale alle prestazioni (scalabilità)

Svantaggi:

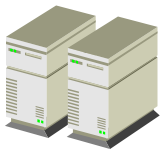
- Ci sono client specializzati (che devono essere installati e devono essere compatibili con i diversi HW a disposizione)
- La manutenzione implica la necessità di effettuare update di software decentralizzati presso i Client
- La realizzazione di un ambiente distribuito è più complessa di un ambiente monolitico

Sistema Biglietteria
Soluzione Distribuita Web-based

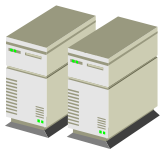
Web Server



Servizi di disponibilità



Servizi di pagamento



Servizi di gestione dati

Sistema Biglietteria
Soluzione Distribuita Web-based

Vantaggi:

• **Tutti i vantaggi evidenziati dai sistemi distribuiti in generale**
• interfaccia tra Client e Server standardizzata permette di creare applicazioni senza la necessità di installare un Client dedicato sui Client
• Il Web Browser diventa un Client general purpose utile per realizzare le applicazioni più diverse
• La standardizzazione dello sviluppo implica una semplificazione nella realizzazione

Svantaggi:

• Il modello di interazione Client-Server è predefinito e non permette interattività forte
• L'interfaccia utente è limitata alle funzioni che lo standard definisce, impoverendosi rispetto alle prestazioni di un Client dedicato

InterNet – IntraNet - ExtraNet

- **InterNet:**
  - rete di accesso pubblico per la diffusione di applicazioni distribuite… (esistono mille definizioni a seconda della moda)
- **IntraNet:**
  - rete aziendale o riservata basata sulle stesse tecnologie di InterNet all'interno della quale operano applicazioni web-based il cui accesso è strettamente riservato all'azienda o ente.
- **ExtraNet:**
  - insieme di applicazioni web-based fruibili via InterNet con accesso riservato a determinati utenti per usi specifici.
  - rappresenta l'estensione dei Sistemi IntraNet sulla rete pubblica per particolari esigenze.

Sistemi Distribuiti
Sistemi Web-based

- Un sistema Web-based è un sistema distribuito.
- Soddisfa tutti i requisiti e le specifiche di un sistema distribuito
- Si basa su standard e tecnologie che consentono di soddisfare tali requisiti
  - Protocolli di comunicazione
  - Modelli e Tecnologie applicative (client side e server side)
  - Architetture e strutture implementative

Sistemi Distribuiti
Specifiche e Requisiti

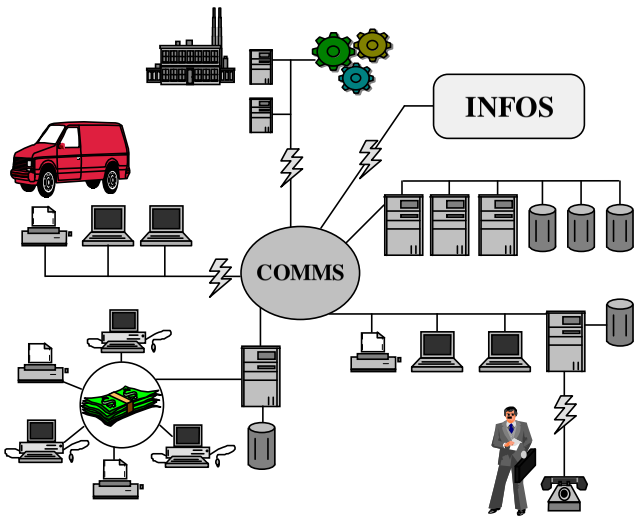
condivisione delle risorse

affidabilità per tollerare guasti

applicazioni intrinsecamente distribuite
(es. prenotazioni aeree)

Accessibilità del sistema

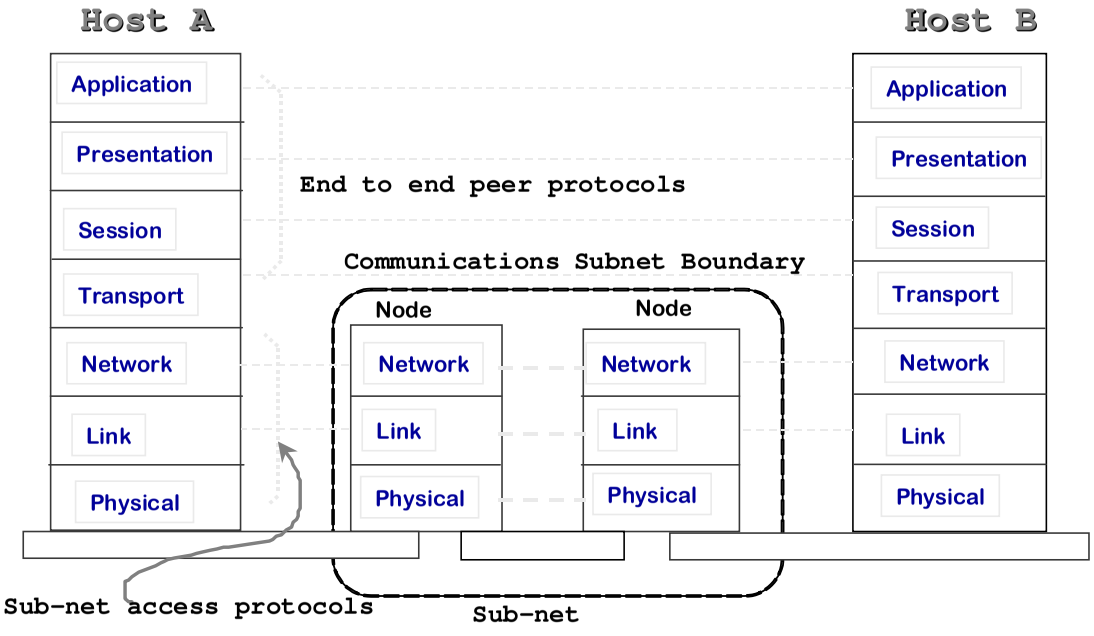
scalabilità



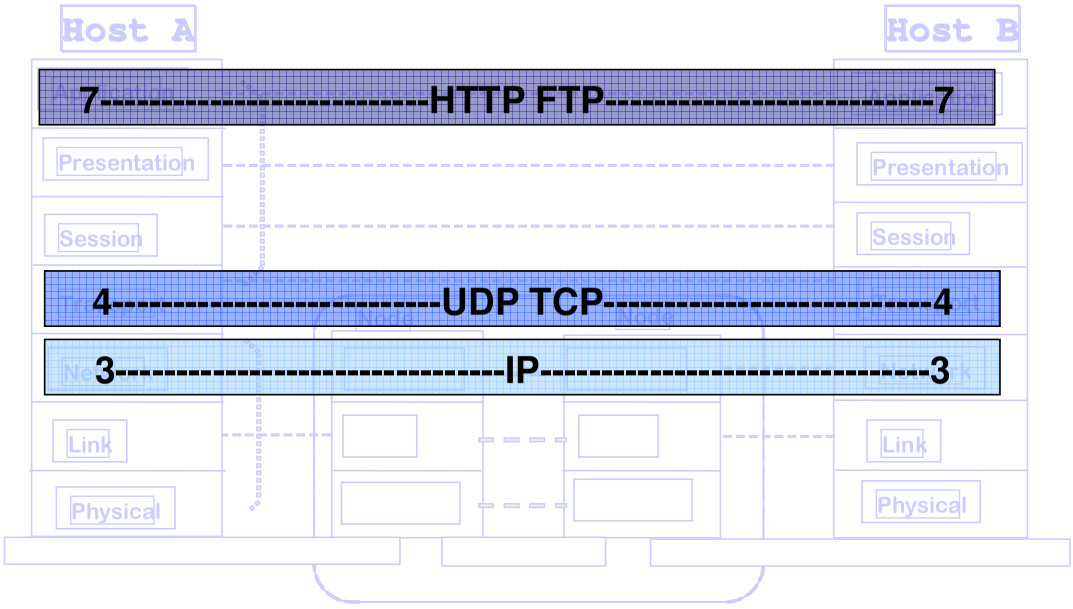
Sistemi Distribuiti
Come soddisfare i Requisiti ?

- **Apertura:**
  - **Definizione di Standard** (TCP/IP, HTTP, FTP, HTML… ma anche CORBA, Java…)
- **Concorrenza:**
  - Dimensionamento e progettazione architetture in grado di soddisfare esigenze di accesso concorrente
- **Trasparenza:**
  - Necessità di realizzare Servizi che siano in grado di nascondere la complessità della implementazione
- **Scalabilità:**
  - **Teorica** (impostazione architetturale)
  - **Pratica** (applicazione reale della architettura)
- **Tolleranza ai Guasti**
  - **Robustezza**
  - **Availability**
  - **Reliability**

Protocolli e Standard
ISO - OSI



Protocolli e Standard
TCP/IP, HTTP, FTP …



HTTP: Hyper Text Transfer Protocol

- Basato sul modello Client/Server
- Il Client effettua una richiesta HTTP:
  - Necessità di una **Risorsa**
  - Invia un **Messaggio** di richiesta della **Risorsa**
- Il Server invia la risposta:
  - Riceve il **Messaggio** di richiesta della **Risorsa**
  - Esegue le elaborazioni necessarie a fornire la risposta
  - Invia un **Messaggio** di risposta (serve la **Risorsa** richiesta)
- Gli elementi in gioco nel protocollo sono:
  - Il **Messaggio** HTTP (richiesta e risposta)
  - La **Risorsa**

HTTP: Hyper Text Transfer Protocol
Terminologia

- **Client:** Programma applicativo che stabilisce una Connessione al fine di inviare delle *Request*
- **Server:** Programma applicativo che accetta Connessioni al fine di ricevere *Request* ed inviare specifiche *Response*
- **Connessione:** circuito virtuale stabilito a livello di trasporto tra due applicazioni per fini di comunicazione
- **Messaggio:** è l'unità base di comunicazione HTTP, è definita come una specifica sequenza di byte concettualmente atomica.
  - **Request:** messaggio HTTP di richiesta
  - **Response:** messaggio HTTP di risposta
  - **Resource:** Oggetto di tipo dato univocamente definito
    - **URI:** Uniform Resource Identifier – identificatore unico per una risorsa.
- **Entity:** Rappresentazione di una Risorsa, può essere incapsulata in un messaggio.

HTTP: Hyper Text Transfer Protocol
Messaggio

Un messaggio HTTP è definito da due strutture:

- **Message Header:** Contiene tutte le informazioni necessarie per la identificazione del messaggio.
- **Message Body:** Contiene i dati trasportati dal messaggio.

Esistono degli schemi precisi per ogni tipo di messaggio relativamente agli header ed ai body.

I messaggi di Response contengono i dati relativi alle risorse richieste (nel caso più semplice la pagina html)

I dati sono codificati secondo il formato specificato nell'header, solitamente sono in formato MIME; è possibile utilizzare anche il formato ZIP

HTTP: Hyper Text Transfer Protocol
URL

- **Uniform Resource Locator:** rappresenta l'estensione dell'**URI** tenendo conto del protocollo necessario per il trasferimento della risorsa. Per il protocollo HTTP l'URL è il seguente:

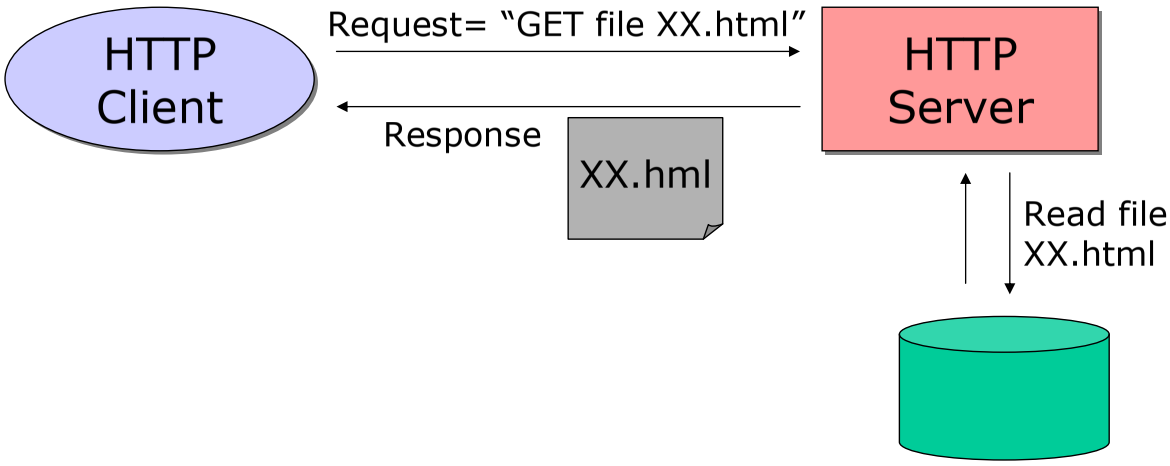
http_URL = "http:" "//" host [":" port] [abs_path ["?" query]]

- Se la porta non viene specificata viene scelta la **porta 80** come da default dello standard
- Se il path non viene specificato interviene il percorso di root del Web Server
- La chiave **"?"** serve per la specifica degli eventuali parametri nella richiesta della risorsa (*chiamata in **get***)

HTTP: Hyper Text Transfer Protocol
Terminologia

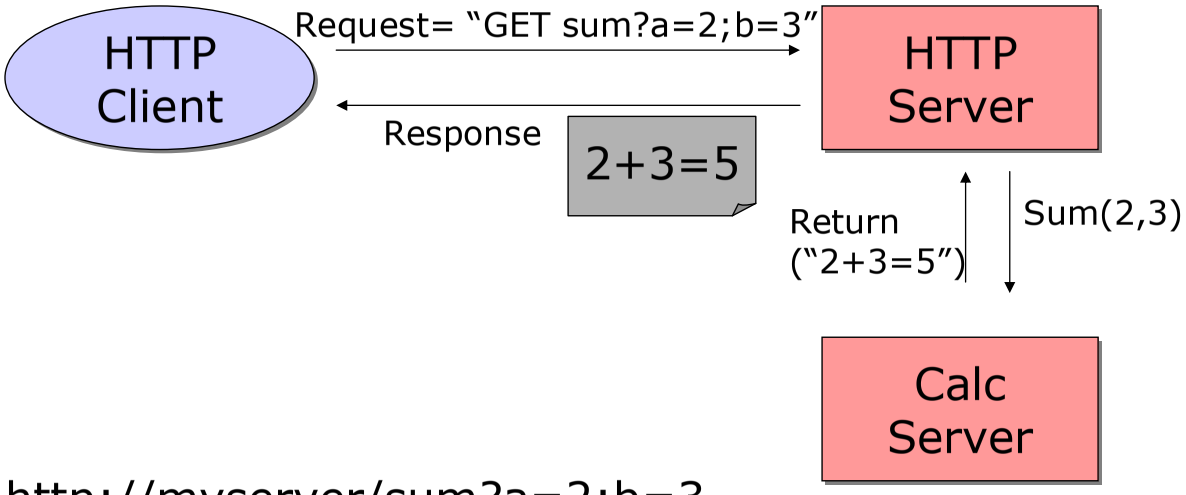
- **Proxy:** Programma applicativo in grado di agire sia come Client che come Server al fine di effettuare richieste per conto di altri Clienti. Le *Request* vengono processate internamente oppure vengono ridirezionate al Server. Un proxy deve interpretare e, se necessario, riscrivere le *Request* prima di inoltrarle.
- **Gateway:** Server che agisce da intermediario per altri Server. Al contrario dei proxy, il gateway riceve le request come se fosse il server originale ed il Client non è in grado di identificare che la Response proviene da un gateway.
- **Cache:** Repository locale di messaggi di *Response,* compreso il sottosistema che controlla la consistenza dei dati. Qualsiasi Client o Server (tranne i tunnel) può includere una cache per motivi di performance. Un *Response* si dice **Cacheable** se il contenuto è memorizzabile senza perdita di consistenza.

HTTP: Hyper Text Transfer Protocol
Request/Response 1



http://myserver/XX.html

HTTP: Hyper Text Transfer Protocol
Request/Response 2



http://myserver/sum?a=2;b=3

HTTP: Hyper Text Transfer Protocol
Metodi di Request

- **GET:** richiedo una specifica risorsa attraverso un singolo URL, posso passare diversi parametri, **la lunghezza massima di un URL è 256 caratteri**

- **POST:** richiedo una specifica risorsa evidenziando che seguirà un altro messaggio la cui entity contiene i dettagli per la identificazione della risorsa stessa: **non ci sono limiti di lunghezza nei parametri di una richiesta**

Ci sono anche altri metodi, che permettono di verificare la versione di una risorsa, la compatibilità del server, le caratteristiche delle risorse...

Il metodi GET e POST vengono gestiti trasparentemente dal server HTTP, questo significa che dal punto di vista dello sviluppatore è trasparente se la richiesta è avvenuta tramite un metodo o l'altro. Quando si può scegliere, è sempre preferibile il metodo POST che non pone limiti di lunghezza massima dei parametri.

HTTP: Hyper Text Transfer Protocol
I Cookie

Parallelamente alle sequenze request/response, il protocollo prevede una struttura dati che si muove come un token, dal client al server e vice versa: **i Cookie**

I cookie sono in realtà una tupla di stringhe:

- **Key:** identifica univocamente un cookie all'interno di un dominio:path
- **Value:** valore associato al cookie (è una stringa di max 255 caratteri)
- **Path:** posizione nell'albero di un sito al quale è associato (di default /)
- **Domain:** dominio dove è stato generato
- **Max-age:** (opzionale) numero di secondi di vita (permette la scadenza di una sessione)
- **Secure:** (opzionale) non molto usato prevede una verifica di correttezza da parte del server
- **Version:** identifica la versione del protocollo di gestione dei cookie

I cookie possono essere generati sia dal client che dal server, dopo la loro creazione vengono sempre passati ad ogni trasmissione di request e response.

Modelli ed Architetture

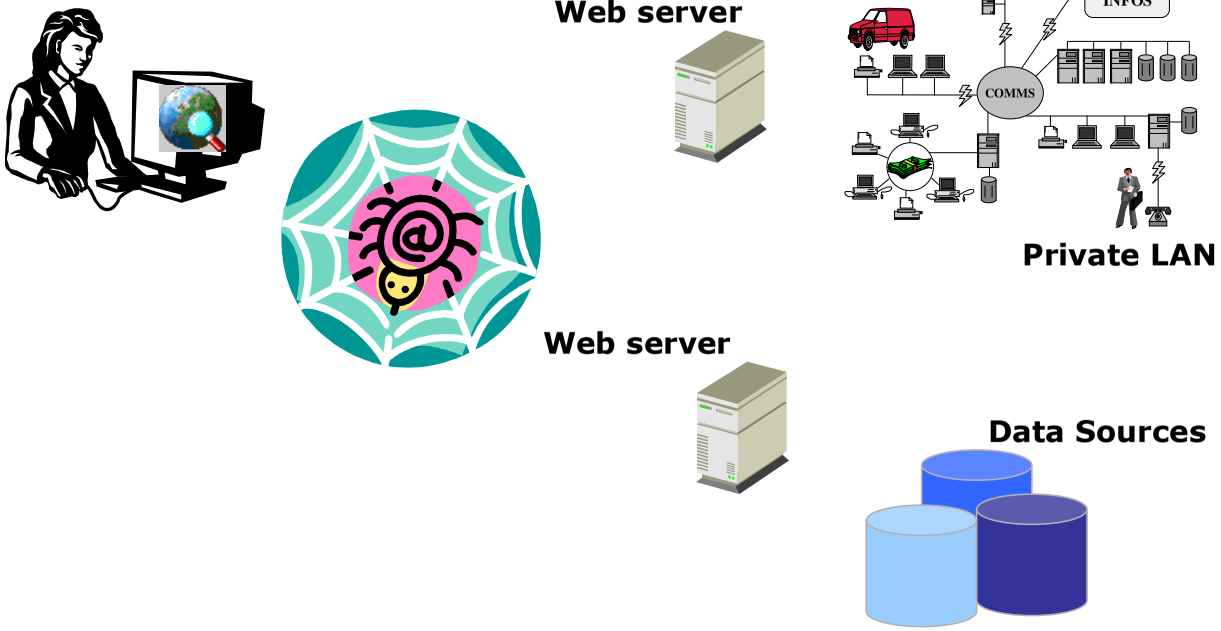
- I **modelli** sono la sintesi, lo scheletro teorico all'interno del quale è possibile costruire applicazioni reali
- Ci sono diversi modelli, applicabili a diversi contesti, con i loro limiti ed i loro vantaggi
- Le **architetture** dettano la composizione delle risorse a disposizione al fine di ottenere strutture anche eterogenee in grado di assolvere a compiti complessi
- Le configurazioni architetturali devono avere specifiche caratteristiche di flessibilità, semplicità e scalabilità al fine di permettere un corretto sviluppo del sistema

Modelli e Tecnologie
Applicazioni Stateful- La classificazione dello Stato

- Un server stateful è un server che "ricorda" informazioni relative alle sue azioni passate: il ricordo è detto **stato**
- Parlando di applicazioni web è possibile classificare lo stato in modo più puntuale:
  - **Stato informativo persistente** (ad esempio gli ordini inseriti da un sistema eCommerce): viene normalmente mantenuto in una struttura persistente come un sistema informativo
  - **Stato di esecuzione** (insieme dei dati parziali per una elaborazione): rappresenta un avanzamento in una esecuzione, per sua natura è uno stato volatile normalmente mantenuto in memoria come stato di uno o più oggetti
  - **Stato di sessione** (insieme dei dati che caratterizzano una interazione con uno specifico utente): la sessione viene gestita di solito in modo unificato attraverso l'uso di istanze di oggetti specifici

Architetture dei Sistemi Web

L'Architettura di un Sistema Distribuito Web-based è l'organizzazione di un insieme di entità che collaborano per attuare le funzionalità richieste



Architetture dei Sistemi Web

È possibile pensare di realizzare una architettura uniforme che permetta di sviluppare applicazioni Web distribuite in grado di lavorare al di sopra delle tecnologie standard offerte.

Quali devono essere le **specifiche principali** di questa architettura ?

- Completa aderenza allo standard HTTP
- Completa compatibilità con i Browser Web disponibili
- Apertura alle diverse tecniche e tecnologie di sviluppo software
- *Ingegnerizzabilità* del software e del processo di sviluppo
- Scalabilità
- Tolleranza ai Guasti

Completa aderenza allo standard HTTP

Ipotesi fondamentale per garantire la completa trasparenza di tutte le infrastrutture di rete alle applicazioni.

Questa ipotesi permette di usufruire di un Browser standard, normali linee di rete basate sulla suite TCP/IP, al di sopra di strutture di rete eterogenee, in piena compatibilità con i servizi di DNS, security, monitoring intrinseci della infrastruttura.

Completa compatibilità con i Browser Web disponibili

I Browser Web diventano una “scatola” dove eseguire le interfacce delle applicazioni.

Queste “scatole” sono però abbastanza delicate, sono realizzate infatti da un insieme di interpreti che elaborano i dati che provengono dai diversi server

Per garantire questa specifica è necessario attuare diversi accorgimenti qualitativi al fine che il codice realizzato sia compatibile con i diversi Browser in commercio.

Apertura alle tecniche e tecnologie di sviluppo software

Il successo di una architettura è spesso vincolato alla apertura della stessa alla evoluzione tecnologica.

La realizzazione di una struttura indipendente dalla tecnologia di sviluppo permette di sviluppare con strumenti diversi, adeguando di volta in volta la struttura alle esigenze

Ingegnerizzabilità del software e del processo di sviluppo

La creazione di applicazioni complesse evidenzia la necessità di applicare tecniche di ingegneria del software in particolare per garantire le seguenti proprietà:

- Previsione e pianificazione dei tempi di lavoro
- Parallelizzazione e Specializzazione nello sviluppo dei diversi componenti del software
- Accelerazione dei tempi di realizzazione
- Semplicità di manutenzione ed evoluzione del codice una volta realizzato

Scalabilità

La scalabilità è una proprietà tipica dei sistemi distribuiti, nei sistemi Web diventa fondamentale per la realizzazione di applicazioni in grado di servire diversi target di Utenti in volumi anche molto diversi.

Questa proprietà può essere risolta a livello architetturale o applicativo; le soluzioni architetturali offrono dei servizi standard, trasparenti allo sviluppo, le soluzioni applicative permettono a volte di garantire performance importanti

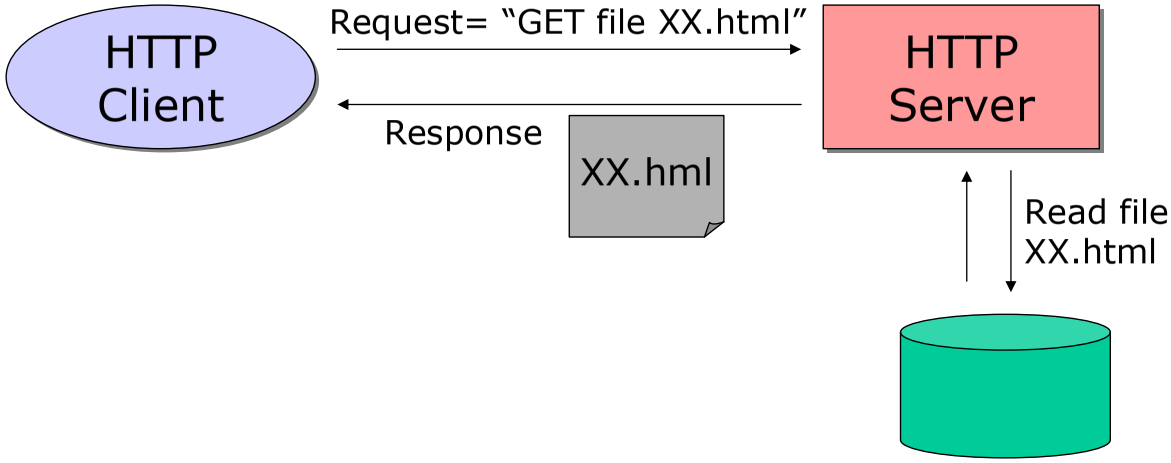
Tolleranza ai Guasti

Nella realizzazione di servizi online, è necessario garantire dei livelli di servizio. Questi livelli di servizio sono ottenibili solo attraverso l'applicazione di architetture che permettono la replicazione dei servizi.

La replicazione può essere realizzata secondo due logiche:

- **Replicazione a Risorse Fredde:** se rimane attiva una sola istanza di servizio e sono disponibili una o più risorse in attesa di sostituire il servizio attivo in modo trasparente
- **Replicazione a Risorse Calde:** si può replicare i servizi mantenendo attive tutte le istanze a disposizione, questo permette di sfruttare tutte le risorse attraverso di politiche di distribuizione e bilanciamento del carico

Architetture dei Sistemi Web
Modello di Base

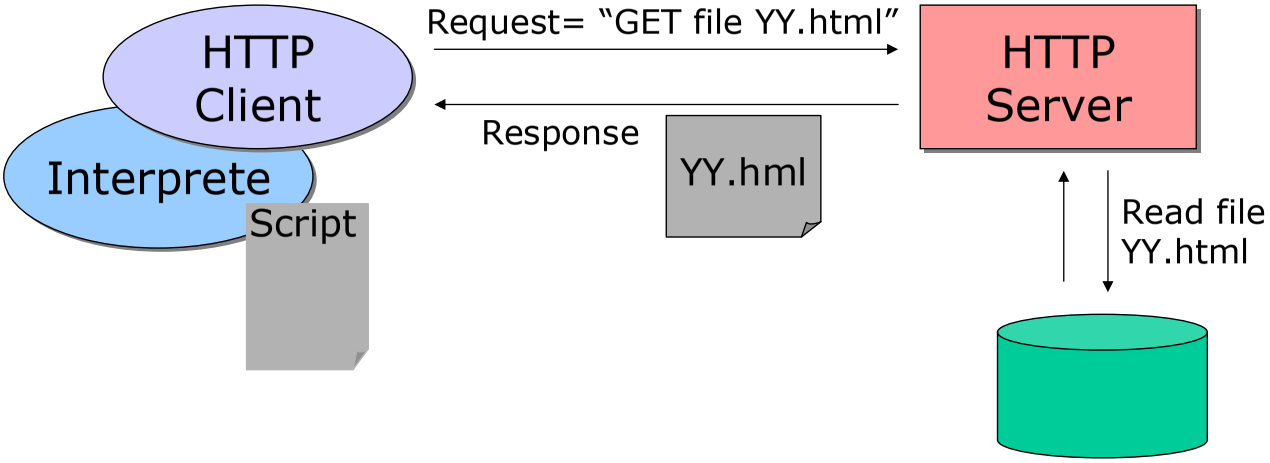


http://myserver/XX.html

Architetture dei Sistemi Web
Modello di Base – Analisi Specifiche

- **Completa aderenza allo standard HTTP:** OK
- **Completa compatibilità con i Browser Web disponibili:** dipende dalla qualità con cui vengono scritte le pagine html, esistono tools di sviluppo che permettono di verificare ed ottimizzare la *compatibilità cross-browser* anche nello sviluppo di pagine in dhtml.
- **Apertura alle diverse tecniche e tecnologie di sviluppo software:** nessuna apertura, le applicazioni Web così fatte offrono solo un ambiente di navigazione ad ipertesti per la consultazione dei dati contenuti, non è possibile sviluppare codice al di fuori dell'html stesso
- ***Ingegnerizzabilità* del software e del processo di sviluppo:** molto limitata, possibilità di applicare principi di riusabilità del codice in semplici contesti ma con scarsi risultati
- **Scalabilità:** ottima, il server è stateless, posso affiancare quanti server desidero che insistano sugli stessi sources, posso replicare sia i server che i sources.
- **Tolleranza ai Guasti:** ottima, la ottengo implicitamente dalla possibilità di replicare il servizio

Architetture dei Sistemi Web
Modello di Base con Programmazione Client side



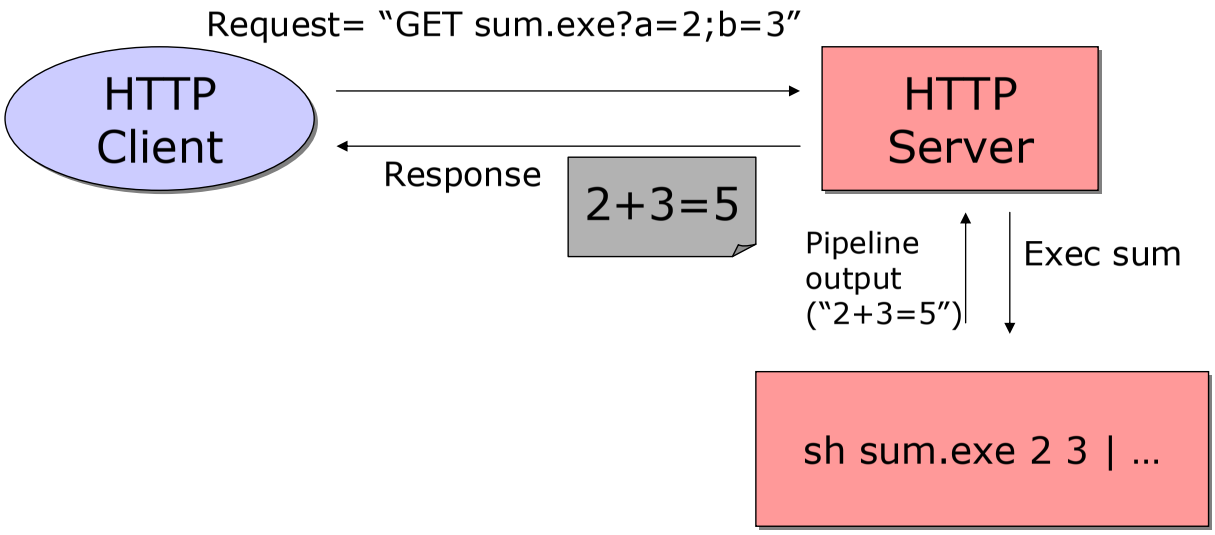
http://myserver/YY.html

Architetture dei Sistemi Web
Modello di Base con Programmazione Client side
Analisi Specifiche

- **Completa aderenza allo standard HTTP:** OK
- **Completa compatibilità con i Browser Web disponibili:** dipende dalla qualità con cui vengono scritte le pagine html e dalla capacità del Browser di interpretare il linguaggio di scripting.
- **Apertura alle diverse tecniche e tecnologie di sviluppo software:** minima, limitata alle capacità dell'interprete del Browser
- ***Ingegnerizzabilità* del software e del processo di sviluppo:** limitata, possibilità di applicare principi di riusabilità del codice realizzando semplici librerie importabili in diverse pagine
- **Scalabilità:** ottima, il server è stateless, posso affiancare quanti server desidero che insistano sugli stessi sources, posso replicare sia i server che i sources.
- **Tolleranza ai Guasti:** ottima, la ottengo implicitamente dalla possibilità di replicare il servizio

Architetture dei Sistemi Web
CGI

- **CGI – Common Gateway Interface:** Il Web Server passa le chiamate alle applicazioni realizzate secondo una logica simile ad un "filtro unix"



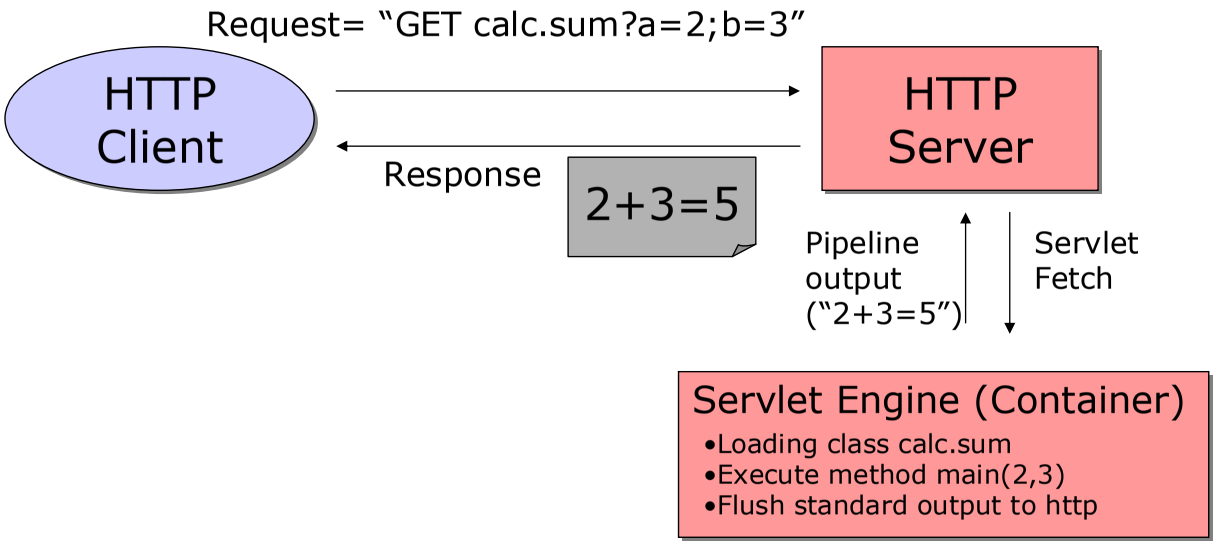
http://myserver/sum.exe?a=2;b=3

Architetture dei Sistemi Web
CGI – Analisi Specifiche

- **Completa aderenza allo standard HTTP:** OK
- **Completa compatibilità con i Browser Web disponibili:** dipende ***solamente*** dalla qualità con cui vengono scritti i programmi CGI, non c' è nessun supporto a livello architetturale che garantisca la qualità dell'output.
- **Apertura alle diverse tecniche e tecnologie di sviluppo software:** è possibile realizzare programmi CGI con diverse tecniche e linguaggi; sono molto usati linguaggi sia scripting come il perl o tcl/tk ma è possibile utilizzare anche il C.
- ***Ingegnerizzabilità* del software e del processo di sviluppo:** la struttura delle applicazioni è molto frammentata, spesso si opta per una rapida prototipizzazione attraverso linguaggi di scripting piuttosto che cercare di applicare tecniche di ingegneria del software su strutture realizzate da enormi quantità di filtri disaggregati.
- **Scalabilità:** ottima, anche in questo caso il server è stateless, posso affiancare quanti server desidero che insistano sugli stessi sources, posso replicare sia i server che i sources.
- **Tolleranza ai Guasti:** ottima, la ottengo implicitamente dalla possibilità di replicare il servizio.

Architetture dei Sistemi Web
Java Servlet

- **Un Servlet** è una classe Java in grado di ricevere in modo strutturato i parametri e generare, sullo Standard Output, la pagina html di risposta



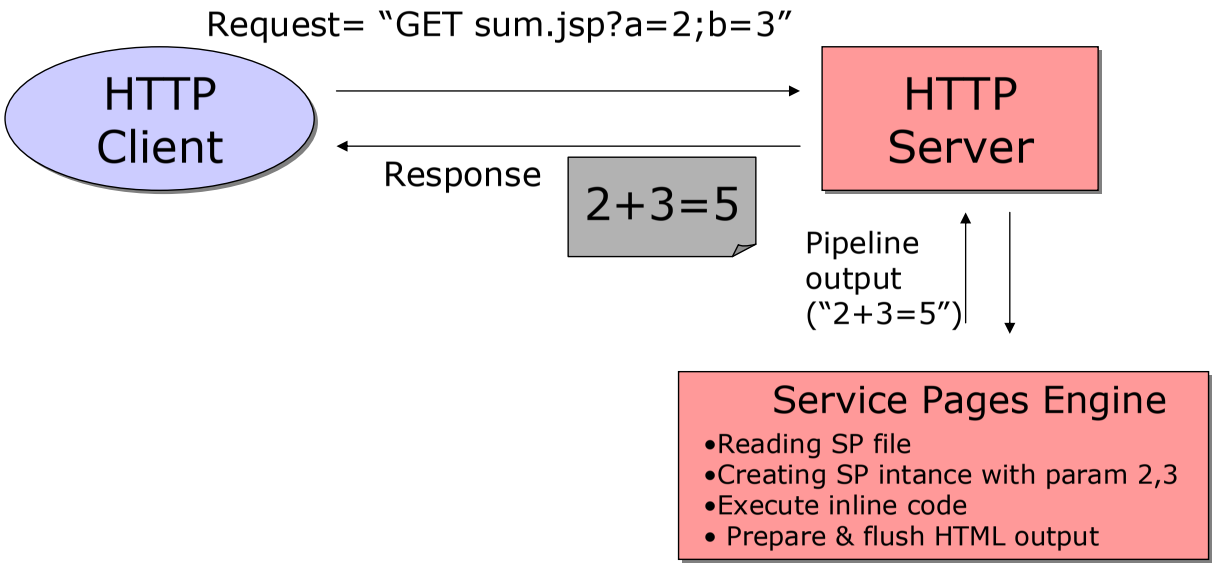
http://myserver/calc.sum?a=2;b=3

Architetture dei Sistemi Web
Java Servlet – Analisi Specifiche

- **Completa aderenza allo standard HTTP:** OK
- **Completa compatibilità con i Browser Web disponibili:** dipende dalla qualità con cui vengono scritte le servlet, non esiste nessun controllo di correttezza del codice html di output
- **Apertura alle diverse tecniche e tecnologie di sviluppo software:** questa architettura si basa strettamente sulla tecnologia Java, l'apertura viene quindi concettualmente demandata a livello di linguaggio. Java viene definito come un linguaggio aperto ed adatto all'interoperabilità ed alla integrazione in sistemi aperti
- ***Ingegnerizzabilità* del software e del processo di sviluppo:** anche in questo caso questa proprietà risente delle caratteristiche di Java. È possibile programmare secondo un modello ad oggetti sia attivi che passivi. La struttura della servlet rende abbastanza complessa la creazione della pagina Web di output. Non è possibile separare la logica di elaborazione dalla grafica vera e propria
- **Scalabilità:** il Servlet Engine è stateful, è possibile mantenere oggetti attivi in memoria che interagiscono con le servlet. La replicazione dipende quindi o dalla implementazione di strutture di clustering da parte del servlet engine oppure da una soluzione a livello applicativo.
- **Tolleranza ai Guasti:** segue di pari passo le problematiche di replicazione evidenziate per la scalabilità

Architetture dei Sistemi Web
Service Pages

- **Una Service Page** è una pagina html che contiene al suo interno del codice che deve essere eseguito sul lato server prima che la pagina venga inviata al client

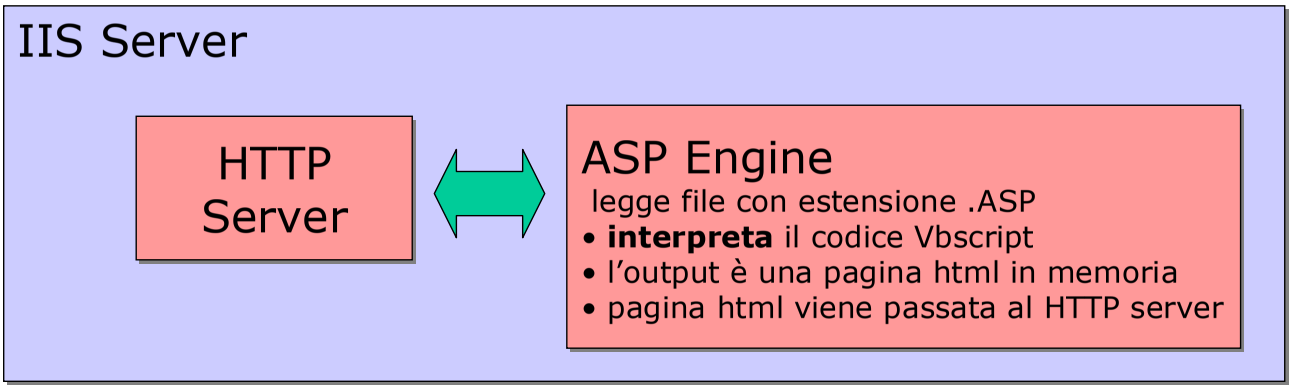


http://myserver/sum.jsp?a=2;b=3

Architetture dei Sistemi Web
Service Pages – Microsoft ASP

La architettura a *Service Pages* è stata implementata per la prima volta da Microsoft come soluzione di rapida prototipizzazione delle pagine Web dinamiche utilizzando Microsoft Internet Information Server. La soluzione Microsoft è nota come **ASP-Active Service Pages,** è stato il primo strumento per lo sviluppo di applicazioni Web-based che non richiedesse conoscenze specifiche di programmazione di rete.

Il primo linguaggio utilizzato per la realizzazione delle pagine ASP è stato il **VBScript,** una forma semplificata del Visual Basic di Microsoft.



Architetture dei Sistemi Web
Service Pages -JSP

La struttura di esecuzione delle Java Servlet, grazie all'estrema flessibilità del linguaggio, è compatibile con la architettura a service pages. Si parla in questo caso di **JSP- Java Service Pages**.

Concettualmente, ASP e JSP sono la stessa cosa, con la sola differenza del linguaggio di scrittura delle SP.

Dal punto di vista architetturale invece la struttura JSP è significativamente diversa:

- Esiste un preCompilatore Java (JSP compiler): Il JSP compiler converte la pagina JSP in una servlet che poi viene compilata come una classe java standard
- Queste differenze hanno importanti ripercussioni sulla estensione della architettura a Service Page nelle due tecnologie

Architetture dei Sistemi Web
Service Pages – Analisi Specifiche

- **Completa aderenza allo standard HTTP:** OK
- **Completa compatibilità con i Browser Web disponibili:** esistono dei tools che aiutano lo sviluppo di pagine SP secondo determinate specifiche qualitative
- **Apertura alle diverse tecniche e tecnologie di sviluppo software:** dipende dalla implementazione: ASP è una tecnologia proprietaria quindi completamente chiusa; per le JSP valgono tutte le considerazioni fatte per le Java servlet
- ***Ingegnerizzabilità* del software e del processo di sviluppo:** è una modello che permette un rapido sviluppo di applicazioni ma, preso da solo, non offre nessun strumento di ingegnerizzazione del software
- **Scalabilità:**come il Servlet Engine, anche gli SP engine sono in generale stateful, le diverse tecnologie offrono diverse soluzioni al problema della scalabilità
- **Tolleranza ai Guasti:** segue di pari passo le problematiche di replicazione evidenziate per la scalabilità

Modelli e Tecnologie
Sessioni e Conversazioni

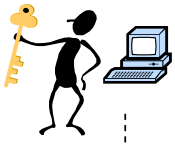
La Sessione rappresenta lo stato associato ad una sequenza di pagine visualizzate da un utente:

- Contiene tutte le informazioni necessarie durante l'esecuzione; possono essere *informazioni di sistema* (ip di provenienza, lista delle pagine visualizzate…) oppure *informazioni di natura applicativa* (nome e cognome, username, quanti e quali prodotti ha inserito nel basket per un acquisto…)
- È l'unico stato necessario per la gestione di una applicazione web

La Conversazione rappresenta una sequenza di pagine di senso compiuto, (ad esempio l'insieme delle pagine necessarie per comperare un prodotto)

- È univocamente definita dall'insieme delle pagine che la compongo e dall'insieme delle interfacce di input/output per la comunicazione tra le pagine (detto **flusso della conversazione)**

Modelli e Tecnologie
Una Conversazione di Acquisto Online



1. Inizia la Conversazione, l'utente inserisce username e password:Il server ottiene i dati e li verifica con i dati presenti nel database dei registrati: **viene creata la sessione:**
   - Username
   - Nome e Cognome
   - …

2. Utente è autorizzato, sfoglia il catalogo alla ricerca di un prodotto; il server lo riconosce attraverso i dati di sessione

3. Trova il prodotto e lo mette nel carrello: **la sua sessione viene aggiornata con le informazioni del prodotto**

4. Compila i dati di consegna

5. Provvede al pagamento, **fine della conversazione di acquisto**

A questo punto l'ordine viene memorizzato nella base dati e viene successivamente inviato ai sistemi logistici per la spedizione.
• La sessione è ancora attiva e l'utente può fare una altro acquisto o uscire dal sito

Nota per l'esame:
La complessità del progetto

Il progetto viene misurato per complessità in base al numero delle conversazioni che vengono implementate.

Una conversazione è una sequenza di pagine, può avere diversa complessità e tale complessità si misura nel numero di stati che gestisce

→ Esempio: una conversazione di registrazione ad un sito può avere il seguente flusso:

1. Richiesta compilazione form
2. Registrazione dati
   a. Conferma avvenuta registrazione
   b. Errore nei dati ritorno al punto 1 con la form precompilata

Ci sono 2 pagine (inserimento dati e registrazione) e 3 stati:

- Form iniziale (senza dati), Form precompilata (in caso di errore)
- Registrazione con successo

La complessità di un progetto deve essere di almeno 2 conversazioni con almeno 4 stati PER OGNI studente del gruppo di lavoro

Modelli e Tecnologie
Gestione della Sessione

- La Sessione ha quindi i seguenti requisiti:
  - Deve essere condivisa dal Client e dal Server
  - È associata ad una o più conversazioni effettuate da un singolo utente
  - Ogni utente possiede la sua singola sessione

- Ci sono due tecniche di base per gestire la sessione:
  1. Utilizzo della struttura dei **cookie**
  2. Gestione di uno stato sul server per ogni utente collegato

- I cookie fanno parte dello standard http, sono quindi sempre disponibili
- La gestione dello stato sul server è possibile in alcune architetture specifiche

Architetture dei Sistemi Web
Struttura Multi-livello delle Applicazioni

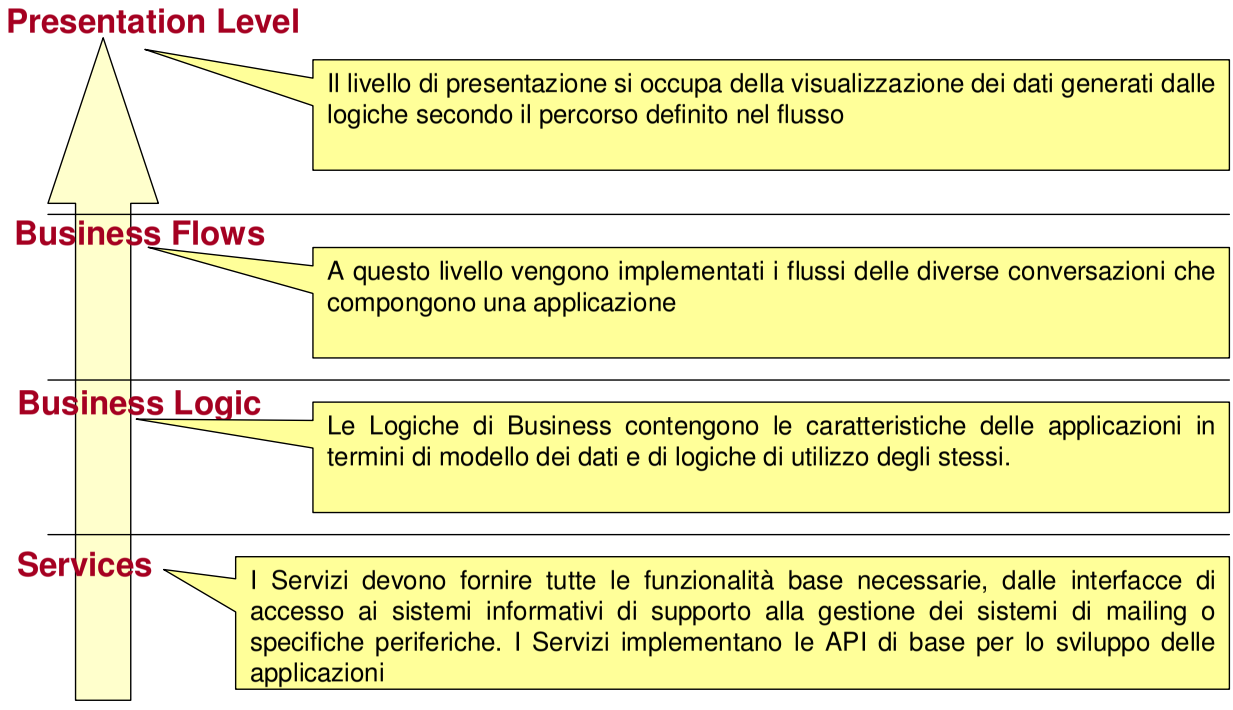
Sviluppare applicazioni secondo una logica ad oggetti o a componenti significa scomporre l'applicazione in blocchi, servizi e funzioni.

È molto utile separare logicamente le funzioni necessarie in una struttura multi-livello al fine di fornire astrazioni via via più complesse e potenti a partire dalle funzionalità più elementari

Nel tempo si è affermata una classificazione indipendente dalla implementazione tecnologica, basata su una struttura a 4 livelli principali.

Questa struttura, non fornisce dettagli implementativi, non specifica quali moduli debbano essere implementati client side o server side, ne nessuna altra specifica tecnica: **è una architettura essenzialmente logico funzionale**

Architetture dei Sistemi Web
Struttura Multi-livello delle Applicazioni



Architetture dei Sistemi Web
Services

Realizzano le funzioni di base per lo sviluppo di applicazioni:

- **Accesso e gestione delle sorgenti di dati:**
  - Database locali
  - Sistemi informativi remoti
  - Legacy Systems (termine generico che identifica applicazioni esterne per la gestione aziendale – dal mainframe ad altri sistemi web-based)
- **Accesso e gestione risorse:**
  - Stampanti
  - sistemi fax, sms, email
  - Dispositivi specifici, macchine automatiche…
- **Sistemi di gestione e monitoraggio della applicazione**
  - Gestione della sessione web
  - Gestione degli errori applicativi
  - Logging e tracing della applicazione

Architetture dei Sistemi Web
Business Logic

La logica è l'insieme di tutte le funzioni ed i servizi che l'applicazione offre; questi servizi si basano sulle strutture di basso livello dei Services per implementare i diversi algoritmi di risoluzione e provvedere alla generazione dei dati di output.

Esempi di moduli di business logic possono essere:

- Gestione delle liste di utenti:
- Gestione cataloghi online

A questo livello, non è significativa quali sono le sorgenti di dati (nascoste dal livello di services) come non è significativo come arrivano le richieste di esecuzione dei servizi e come vengono gestiti i risultati ai livelli superiori

I moduli di business logic (siano essi implementati in componenti che in oggetti) mantengono il contenuto funzionale (la cosiddetta logica di business) che può essere utilizzata in diversi contesti, non vincolati a determinate interfacce o conversazioni

Architetture dei Sistemi Web
Business Flow

Una conversazione è realizzata da un insieme di pagine collegate in un flusso di successive chiamate.

Il business flow raccoglie quindi l'insieme delle chiamate necessarie per realizzare una conversazione; ogni chiamata deve caricare i parametri in ingresso, chiamare le funzioni di business logic necessarie per effettuare l'elaborazione e generare l'output che dovrà essere visualizzato.

Un flusso identifica quindi univocamente una conversazione, l'astrazione della business logic permette di studiare l'esecuzione delle singole pagine in modo indipendente dalla struttura dei dati e degli algoritmi sottostanti

Architetture dei Sistemi Web
Presentation Level

Il business flow è in grado di fornire i dati di output necessari; il **livello di presentazione** ha il compito di interpretare questi dati e generare l'interfaccia grafica per la visualizzazione dei contenuti

Questi due livelli sono concettualmente divisi poiché la generazione dei dati è logicamente separata dalla sua rappresentazione e formattazione.

Questo permette di avere diverse tipi di rappresentazione degli stessi dati, per esempio una rappresentazione in italiano ed una in inglese o una in html ed una in plain ASCII

Architetture dei Sistemi Web
Realizzazione di Architetture Multi-livello

Non tutte le tecnologie permettono di rispettare questa suddivisione, in molti casi i sistemi vengono realizzati a 2 o 3 livelli.

Queste semplificazioni portano in certi casi a miglioramenti nelle performance ma comportano un netta riduzione della leggibilità e della rapidità di sviluppo. Come sempre il trade off viene deciso in base al contesto, non esiste la soluzione ideale ad ogni situazione

Esempio di semplificazione a 2 livelli:

Presentation Level **Business Flows**

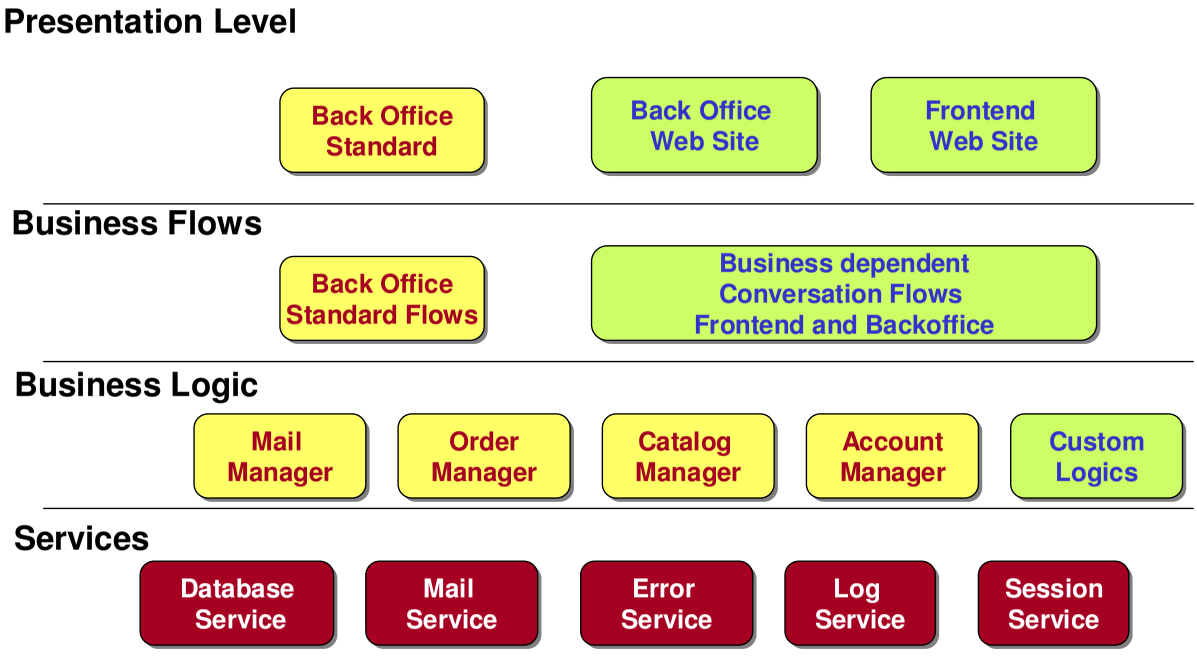
Business Logic **Services**

Una struttura a 2 livelli può essere realizzata fondendo i due livelli più bassi: Blogic e Services ed i due più alti Presentation e Bflows.

Il livello più basso ora non è più indipendente dalle sorgenti di dati, non posso riutilizzare le logiche su diverse basi dati ad esempio.

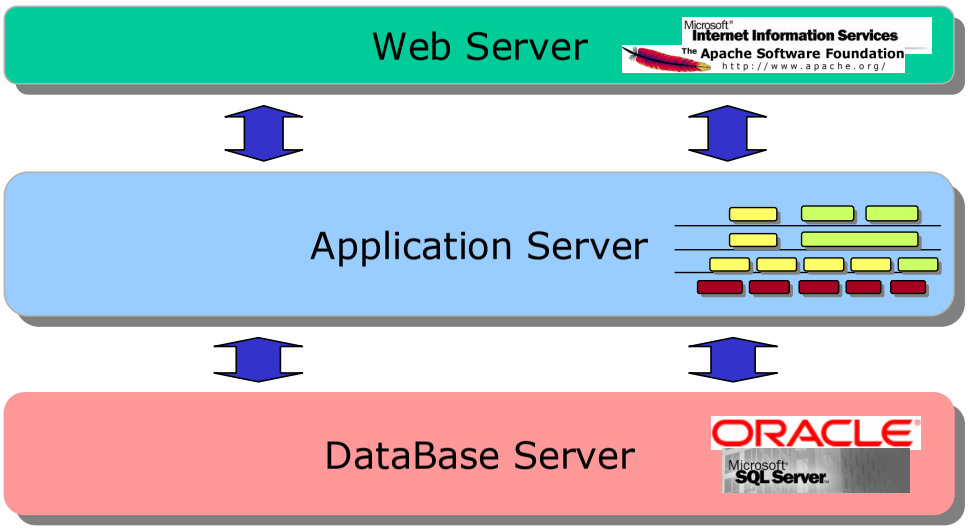
Il livello più alto invece riunisce la struttura delle conversazioni con la loro rappresentazione, diventa quindi impossibile modificare la formattazione in modo indipendente dalla conversazione e vice versa

Architetture dei Sistemi Web
Struttura di un Framework per l'eCommerce



Architetture dei Sistemi Web
Architettura SW/HW e divisione dei servizi

La architettura applicativa, nelle sua separazione logica in layer, rappresenta il razionale con cui viene sviluppato il cosiddetto layer applicativo di un Sistema Web:



Architetture dei Sistemi Web
Distribuzione dei Servizi

La struttura a 3 livelli rispecchia i 3 principali servizi che realizzano un sistema Web.

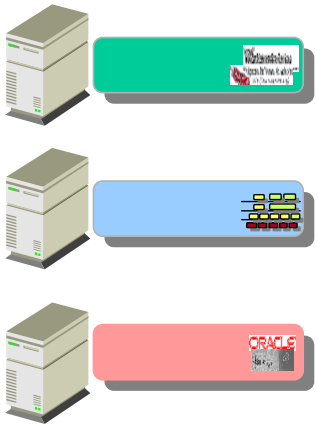
Questi 3 servizi possono risiedere sullo stesso HW oppure essere divisi su tre macchine separate:

Si parla in questo caso di **Distribuzione verticale** della architettura; è molto efficiente perché non necessita di nessun accorgimento specifico, viene realizzata essenzialmente per motivi di performance soprattutto quando si dividono il livello applicativo da quello database. Questo tipo di distribuzione non prevede replicazione, non è quindi utile per risolvere problemi di fault tolerance.

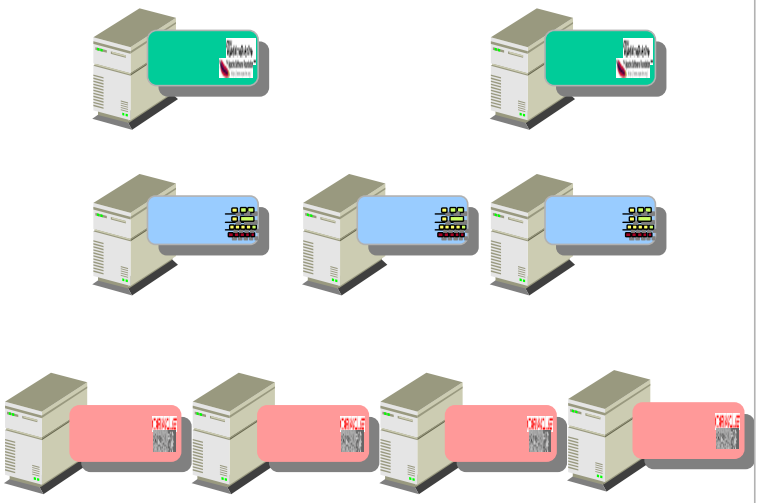
Orizzontalmente ad ogni livello è possibile replicare il servizio su diverse macchine; si parla in questo caso di **Distribuzione orrizzontale**, necessità di importanti accorgimenti strettamente dipendenti dalla tecnologia d'uso. Essendo una distribuzione per replicazione è possibile implementare politiche per la gestione della fault tolerance

Architetture dei Sistemi Web
Configurazioni Distribuite e Replicate

Distribuzione Verticale



Distribuzione Orizzontale



Architetture dei Sistemi Web
Replicazione DataBase – I Cluster

Il database server è un server stateful; la replicazione è molto delicata perché deve mantenere il principio di atomicità delle transazioni.

I database commerciali, come Oracle e Microsoft SQL Server prevedono delle configurazioni di clustering in grado di gestire in modo trasparente un numero variabile di CPU e macchine distinte.

Ci sono diverse configurazioni, a risorse calde e fredde, la performance è ottenibile solo attraverso replicazioni a risorse calde, la tolleranza a guasti viene sempre implementata attraverso l'uso di sistemi raid per i dati e di replicazione a coppie delle unità di database

Architetture dei Sistemi Web
Replicazione Applicazione

L'unico stato necessario a livello applicativo è caratterizzato dalla sessione.

È possibile che il servizio di applicazione utilizzi oggetti o componenti con stato per motivi di performance (caching) o necessità specifiche.

Alcuni framework disponibili sul mercato permettono la replicazione attraverso tecniche di clustering molto simili a quelle dei sistemi database, altri framework non sono in grado di replicare orizzontalmente.

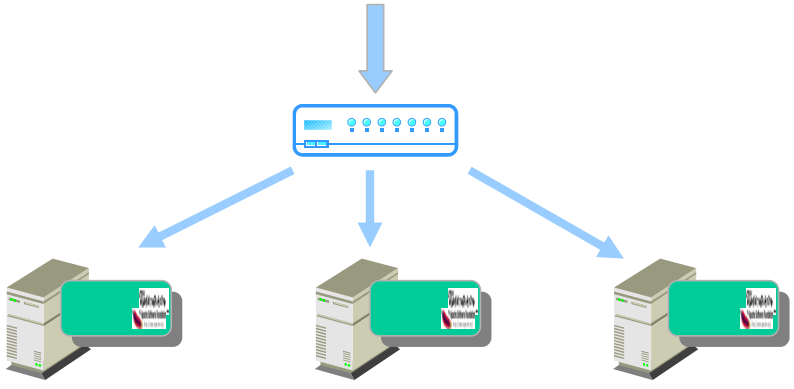
Se si mantiene lo stato concentrato all'interno della sessione, e la sessione viene gestita attraverso i cookie, è possibile realizzare il framework applicativo completamente stateless, ottenendo una configurazione completamente replicabile orizzontalmente.

Architetture dei Sistemi Web
Replicazione Web Server

Il web server è stateless, non crea problemi nella replicazione.

L'unicità degli URL può essere gestita attraverso diverse soluzioni sia hardware che software.

Si possono applicare politiche di load balancing e load sharing con diverse euristiche



La sicurezza nel Web

Protezione vs. Sicurezza

- **Protezione:** garantire un utente o un sistema della non interazione delle attività che svolgono
  - in unix ad esempio i processi sono **protetti** nella loro esecuzione perché non possono danneggiarsi tra loro
- **Sicurezza:** garantire un determinato livello di privacy e riconoscimento degli attori in gioco nelle interazioni
  - in unix i sistemi **NON** sono **sicuri**, è possibile accedere alle aree di memoria del processo per ottenere informazioni private

Nel Web ? Quanto mi fido del Client ?

- Le architetture Web basate sull'HTTP garantiscono la **PROTEZIONE** ma **NON la sicurezza**
  - non sono sicuro che l'utente sia veramente quello che dice di essere (*masquerading*)
  - non sono sicuro che le chiamate arrivino dall'indirizzo ip presente nei pacchetti in arrivo (*ip spoofing*)
  - non sono sicuro che i parametri delle request o i contenuti delle response non siano stati letti o modificati da qualcuno nel tragitto (*trasmissione è in chiaro → azioni di tipo bizantino*)
  - non sono sicuro del contenuto dei cookie

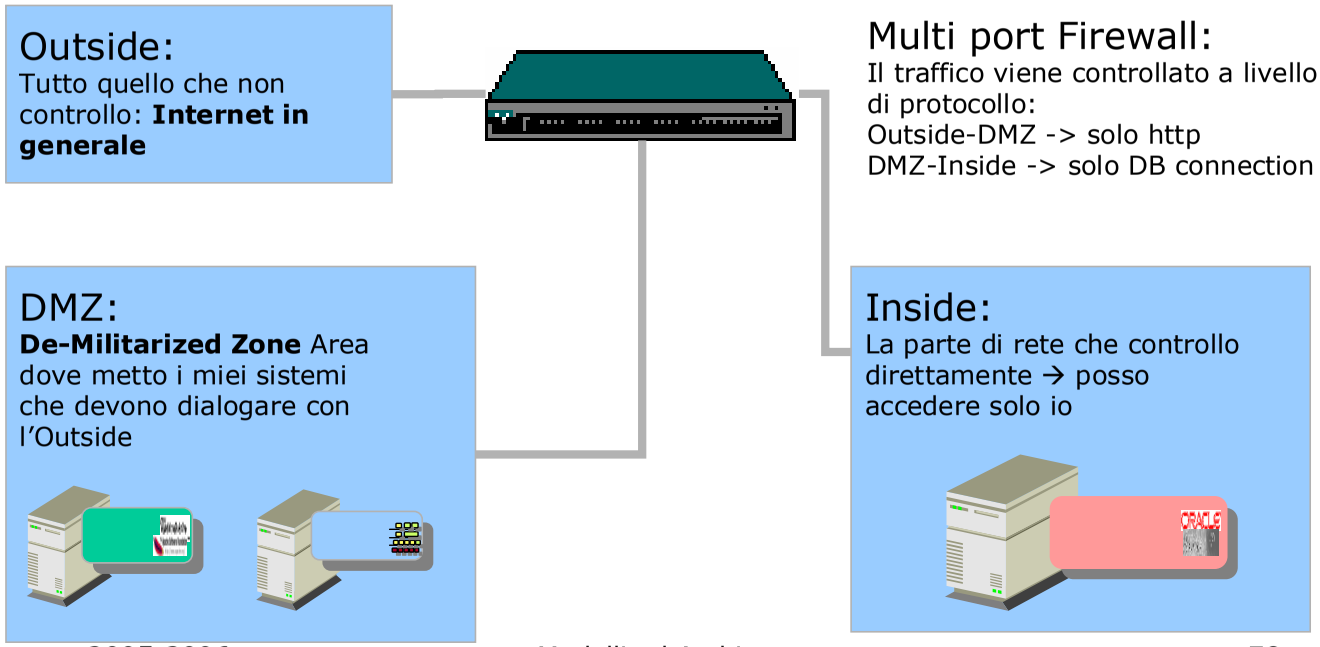
Nel Web ? Quanto mi fido del Server?

- non sono sicuro che il server non sia stato manomesso da qualcuno per cui le pagine che visualizzo sono false (*defacement*)
- non sono sicuro che i miei dati riservati memorizzati dal server non possano essere rubati (*accesso indebito e furto materiale riservato*)
- non sono sicuro che una manomissione del server non permetta di infettare il mio computer con virus e/o spyware (*reliability del web server)*
- non sono sicuro del contenuto dei cookie

Come si ottiene la sicurezza

- Nei sistemi distribuiti, la sicurezza si ottiene attraverso diversi strumenti:
  - crittografia dei contenuti che devono passare in rete
    - algoritmi a chiave privata (DES)
    - algoritmi a chiave publica (RSA)
  - certificazione degli utenti e firma dei contenuti:
    - ogni utente deve possedere un certificato, che identifica univocamente la persona ed il sistema che lo usa
    - ogni contenuto che appartiene ad un utente viene firmato: la firma è un check sum speciale che quindi identifica il contenuto come integro ed appartenente ad una persona specifica
  - uso di sistemi di controllo del traffico (firewall)
  - monitoraggio e controllo dei sistemi (intrusion detection)

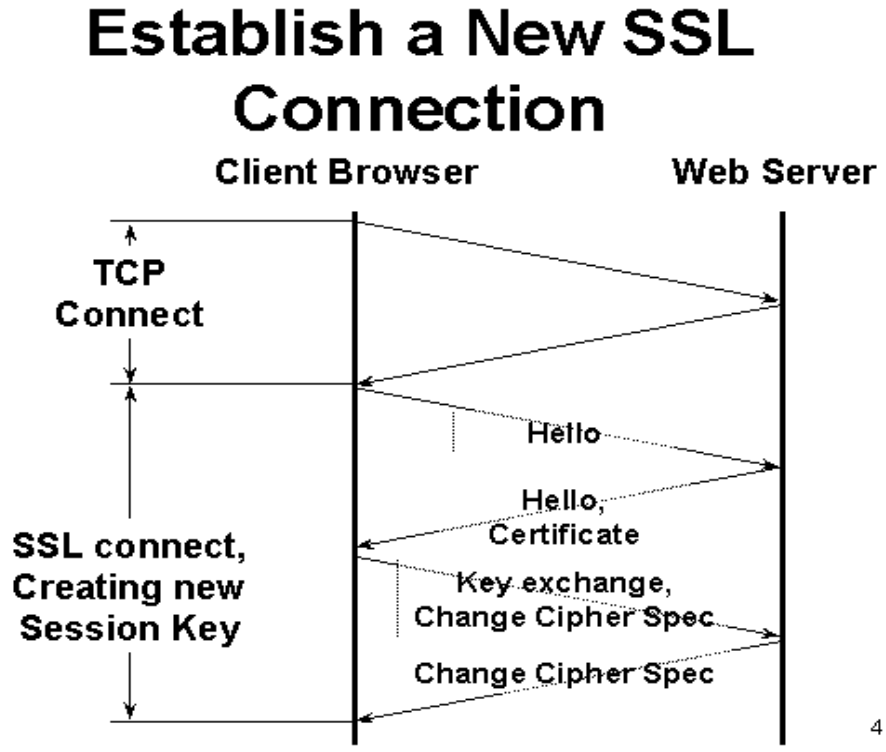
La Web Farm: Inside – Outside - DMZ



Sicurezza nella comunicazione: SSL – Secure Socket Layer

- Per offrire al programmatore uno strumento stardard è stata utilizzata la tecnica dell'incapsulamento, al di sopra delle socket sono state costruite le socket sicure
- Le socket sicure usano un sistema misto a chiave pubblica/privata con autenticazione per certificato
- Garantiscono quindi un canale di comunicazione sicuro

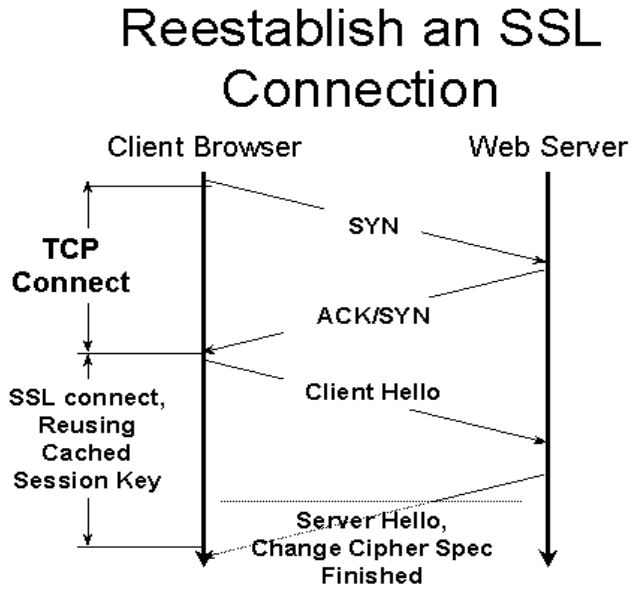
Sicurezza nella comunicazione: SSL – Secure Socket Layer



HTTPS: Implementazione sicura dell'HTTP

- Il protocollo HTTPs è a tutti gli effetti il protocollo HTTP che fa uso di SSL anziché del TCP standard
- Permette di interagire in modo sicuro tra client e server senza che il programmatore delle applicazioni web abbia la necessità di implementare una soluzione ad hoc
- a differenza di HTTP, HTTPs è stateful:
  - deve tenere traccia delle chiavi private sia sul client che sul server per garantire continuità nella comunicazione

Lo stato del HTTPs:



5

HTTPS: Quale livello di sicurezza garantisce?

- Ogni browser contiene al suo interno un certificato di default
  - •HTTPs, in mancanza del caricamento di uno specifico certificato personale NON garantisce CHI è l'utente
  - • è possibile chiedere una password a livello applicativo per ottenere la identificazione
- Una volta creata la connessione sicura l'interazione è protetta da ip spoofing, masquerading e le informazioni non sono leggibili senza le chiavi private:
  - • una chiave privata può essere identificata! Ma cambia ad ogni sessione e quindi è difficile che possa essere rintracciata in tempo per fare qualche azione maliziosa
  - • le chiavi pubbliche sono utilizzate per firmare il certificato. Necessità di uso di chiavi pubblica grandi (ci sono chiavi da 56bit fino a 1024bit in RSA) per evitare che un certificato possa essere aperto e quindi reso non sicuro
- Se il server viene attaccato e ne viene preso il controllo, qualsiasi tipo di azione maliziosa è possibile → importante uso dei firewall

HTTPS: Le prestazioni

- SSL è molto più oneroso di TCP:
  - • Fase di handshaking iniziale molto complessa
  - • Maggiore esigenze di CPU sia lato client che server
  - • Maggiore banda necessaria per la comunicazione
- Normalmente si cerca di circoscrivere l'uso di HTTPs solamente alle interazioni critiche (es: pagamenti online)
- Un uso estensivo di HTTPs necessità di un ribilanciamento complessivo dei sistemi (no load balancing, solo load sharing)