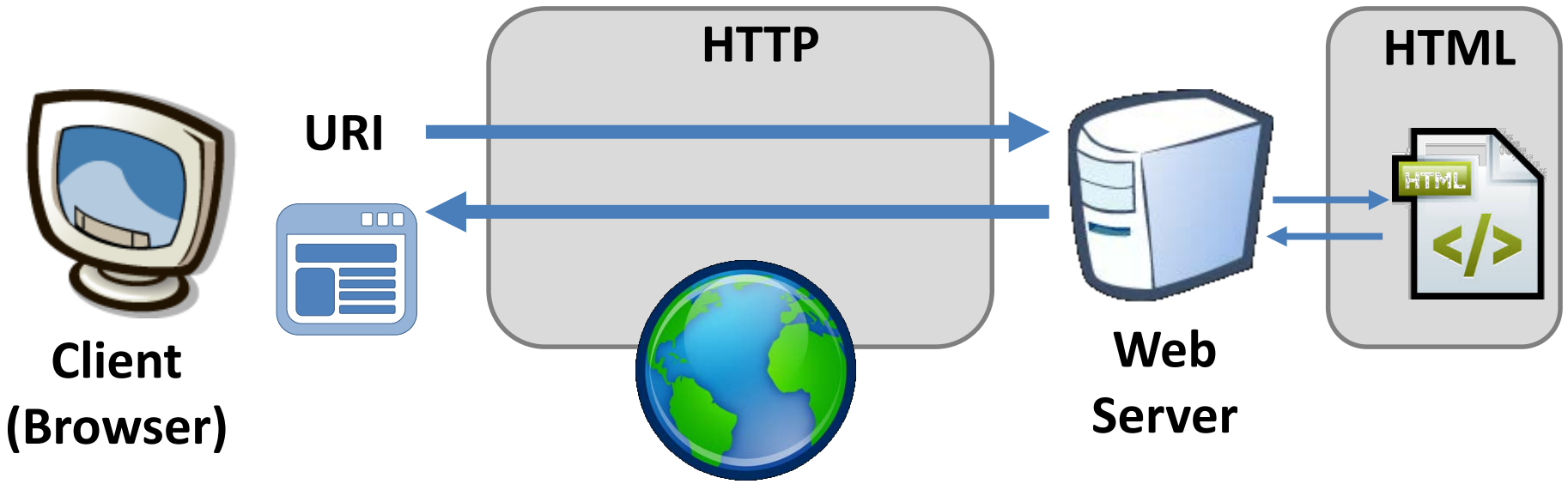
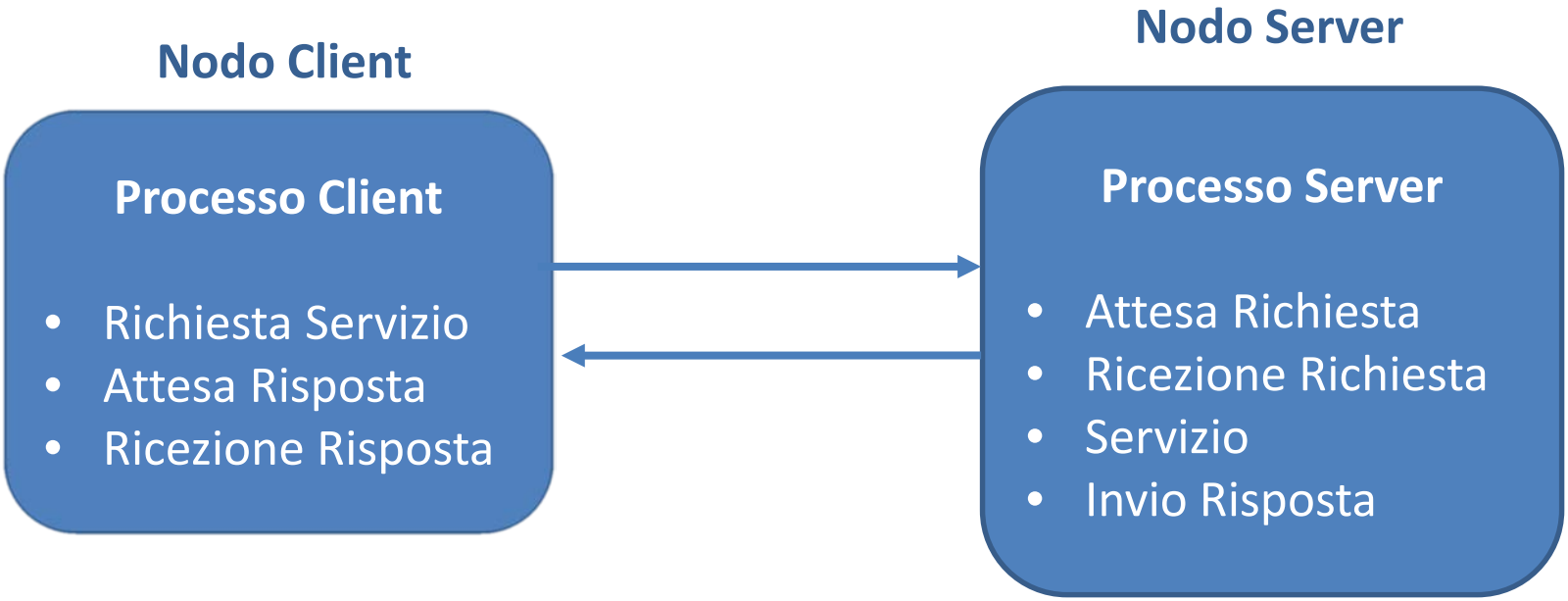
Client e Server nel Web

Modello Client-Server



Modello Client-Server

- Il modello Client-Server prevede due entità:
  - **Client**, che richiede il servizio
  - **Server**, che offre il servizio

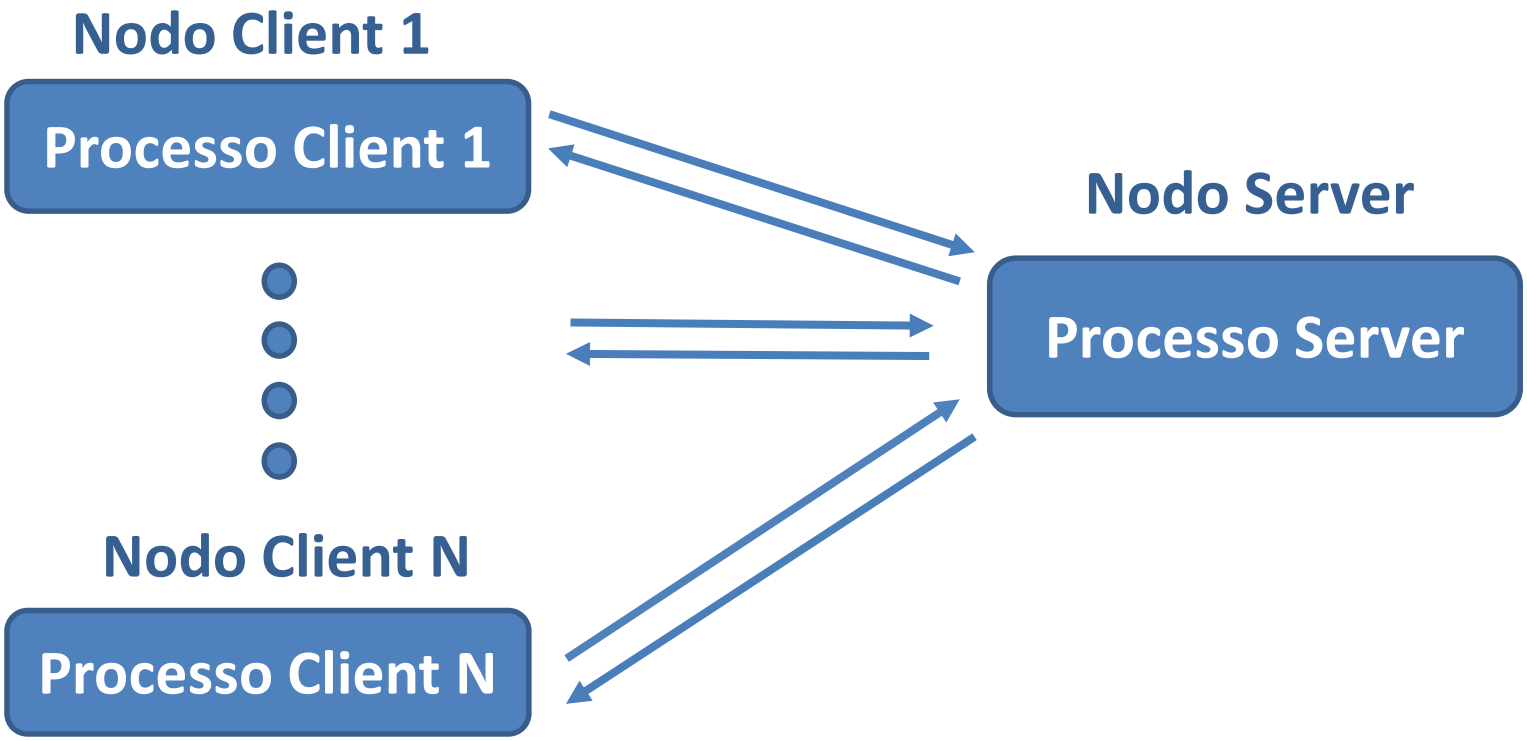


Modello Client-Server

- Il modello Client-Server risolve il problema del rendez-vous (sincronizzare tra loro i processi comunicanti):
  - il Server viene definito 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 Client designa esplicitamente il destinatario
- Il Server risponde al processo che ha effettuato una richiesta

Modello Client-Server

- Modello di comunicazione asimmetrica, molti:1



Modello Client-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

Modello Client-Server: caratteristiche

- Vediamo alcune delle caratteristiche principali che contraddistinguono il modello Client-Server, dal punto di vista di:
  - ***Interazione***
  - ***Stato***
  - ***Concorrenza***

Modello Client-Server - Interazione

- Due tipi principali di interazioni:
  - interazione ***connection-oriented***: viene stabilito un canale di comunicazione virtuale prima di iniziare lo scambio dei dati (esempio: connessione telefonica)
  - interazione ***connectionless***: non c'è una connessione virtuale, ma un semplice scambio di messaggi (esempio: il sistema postale)

Modello Client-Server - Interazione

- La scelta tra i due tipi dipende dal tipo di applicazione e anche da altre caratteristiche proprie del livello di comunicazione sottostante
- Esempio: in Internet il livello di trasporto è **TCP** oppure **UDP**
  - ***TCP*** è ***con connessione***: è reliable (affidabile) e preserva l'ordine di invio dei messaggi
  - ***UDP*** è ***senza connessione***: non reliable e non preserva ordine messaggi

Modello Client-Server - Stato

- L'interazione tra un Client e un Server, dato punto di vista dello ***Stato***, può essere di due tipi:
  - ***Stateful***: ***esiste ed è disponibile lo stato dell'interazione***. Questo significa che un determinato messaggio dipende dai messaggi precedenti
  - ***Stateless***: ***non esiste, non è disponibile lo stato dell'interazione***, dato che non se ne tiene traccia. Questo significa che ogni messaggio è indipendente dagli altri

Modello Client-Server - Stato

- Lo stato dell'interazione è memorizzato sul server (che quindi può essere a sua volta ***stateless*** o ***stateful***):
  - un Server ***stateful*** ha in genere una migliore efficienza (vantaggi: dimensioni messaggi più contenute e migliore velocità di risposta del Server; svantaggi: presenta problemi di replicazione)
  - un Server ***stateless*** è più affidabile in presenza di malfunzionamenti (soprattutto se e quando causati dalla rete) ed è più semplice da progettare

Modello Client-Server - Stato

- Un'***interazione stateless*** è possibile SOLO se il protocollo applicativo è progettato con ***operazioni idempotenti***
- Le operazioni sono dette idempotenti quando producono sempre lo stesso risultato
- 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***, allora il Server deve poter identificare il Client

Modello Client-Server - Concorrenza

- Lato Client
  - I Client sono programmi sequenziali
  - Eventuali invocazioni concorrenti sono supportate dal sistema operativo multitasking

Modello Client-Server - Concorrenza

- Lato Server
  - La concorrenza è cruciale per migliorare le prestazioni di un Server
  - Un Server iterativo processa le richieste di servizio una alla volta. E' quindi possibile un basso utilizzo delle risorse, in quanto non c'è sovrapposizione tra elaborazione ed I/O
  - ….

Modello Client-Server - Concorrenza

- …
- 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)

Modello Client-Server

			Tipo di comunicazione	
			Con connessione	Senza connessione
Tipo di Server	Iterativo			
	Concorrente	Singolo processo		
		Multi processo	**Server Web**	

Modello Client-Server

- Dal Web deriva la classificazione di Programmazione (e quindi esecuzione) Client-Side o Server-Side
- Si definisce:
  - ***esecuzione Server-Side***: elaborazione effettuata dalla entità server; consiste nell'insieme delle operazioni che devono essere completate al fine di generare l'output per il Cliente
  - ***esecuzione Client Side***: elaborazione effettuata dalla entità client; consiste nell'insieme delle operazioni necessarie per la gestione ed eventuale visualizzazione delle risorse ottenute dal server

Modello Client-Server

- Nel Web il contesto di esecuzione Client-Side è costituito dal Browser
- Il browser si occupa di interpretare i dati di output in formato HTML ottenuti dal server e di 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

Q&A

- Ci sono domande?

Applicazioni e sistemi distribuiti

- 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 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, …)

Applicazioni e sistemi distribuiti

- Condivisione delle Risorse:
  - ***Risorse Dati***: Database con diverse informazioni, diverse 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 …

Esempio

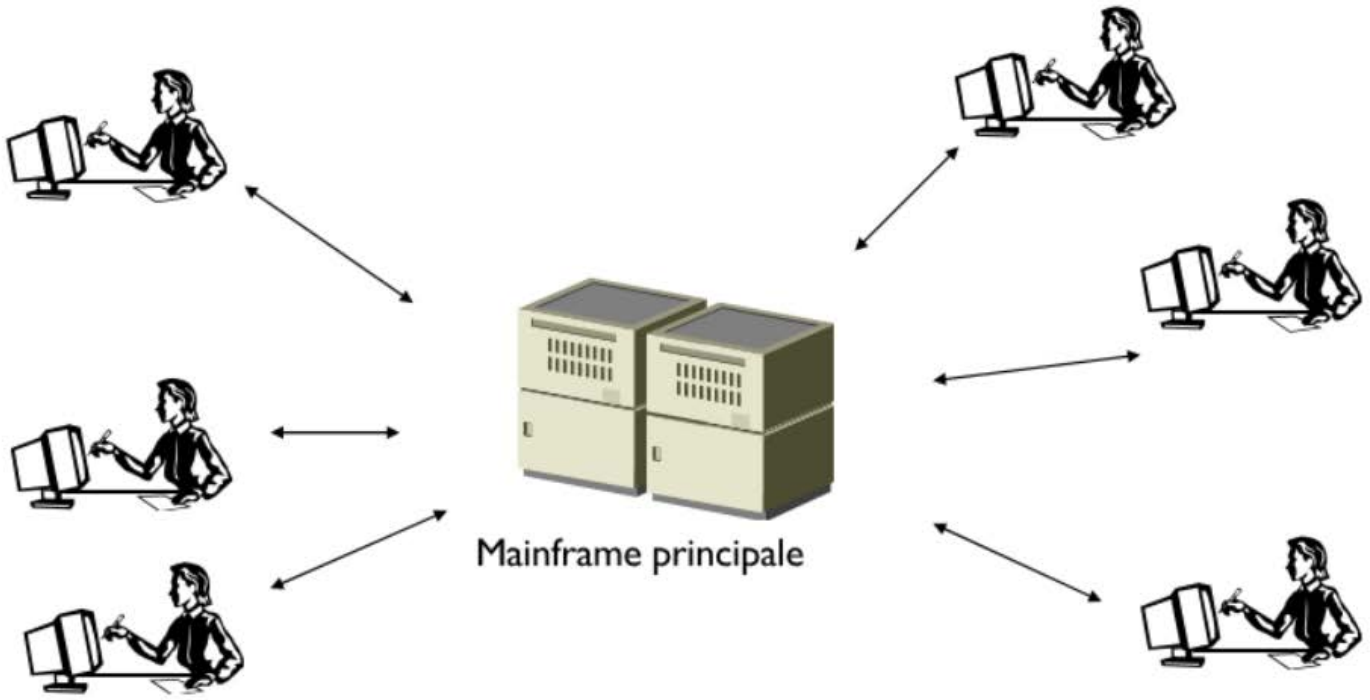
- Applicazione distribuita per la gestione di un sistema di biglietteria aerea per una rete di agenzie «world wide»

Esempio

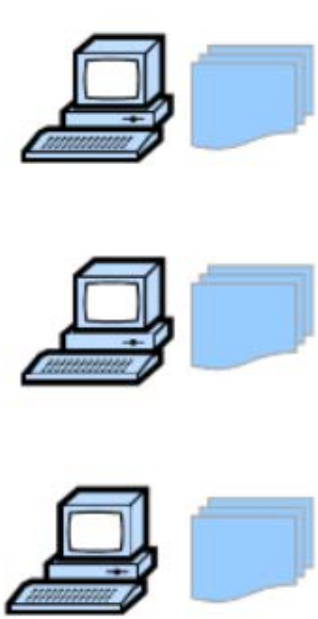
- Specifiche:
  - Fornire disponibilità posti in tempo reale
  - Permettere l'acquisto dei biglietti da diversi punti vendita dislocati in luoghi fisicamente separati
  - Necessità di funzionare in tutti i fusi orari
  - Alto volume di prenotazioni
- Necessità:
  - Condivisione delle informazioni sulla disponibilità in tempo reale
  - Fornitura di un elevato livello di servizio:
    - Interfacce rapide ed efficienti
    - Garanzia di continuità di servizio 24x7

Soluzione monolitica

- Terminali remoti «stupidi» connessi con linee dirette



Soluzione distribuita



Servizi di disponibilità

Servizi di gestione dati

Servizi di pagamento

Confronto: i pro

- Soluzione Monolitica
  - Facile da implementare
  - Una sola macchina (mainframe) contiene tutte le informazioni
- Soluzione Distribuita
  - è possibile affiancare diversi server in parallelo per aumentare prestazioni e 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)

Confronto: i cons

- Soluzione Monolitica
  - Supporta un numero definito di utenti
  - Interfacce semplici e spartane
  - Un guasto del server centrale comporta una perdita del servizio
- Soluzione Distribuita
  - 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

Soluzione distribuita web-based

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
  - L'utente ha familiarità con la user interface del client (ovvero del browser)
  - La standardizzazione dello sviluppo implica una semplificazione nella realizzazione

Soluzione distribuita web-based

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

Q&A

- Ci sono domande?

Soluzione app-based

Soluzione app-based

- Vantaggi:
  - tutti i vantaggi evidenziati dai sistemi distribuiti in generale
  - L'interfaccia, realizzata con un client dedicato, consente una interattività forte con l'utente, adattata/customizzata rispetto alle dimensioni del dispositivo e alle sue caratteristiche fisiche «limitate» (esempio: diversi meccanismi di input)
  - Meccanismi di engagement e fidelizzazione degli utenti (esempio: elementi di gamification)
  - Si possono limitare le interazioni di rete più «costose» (o comunque più «onerose») in ambito mobile (sfruttando diverse modalità di connessione e connettività)

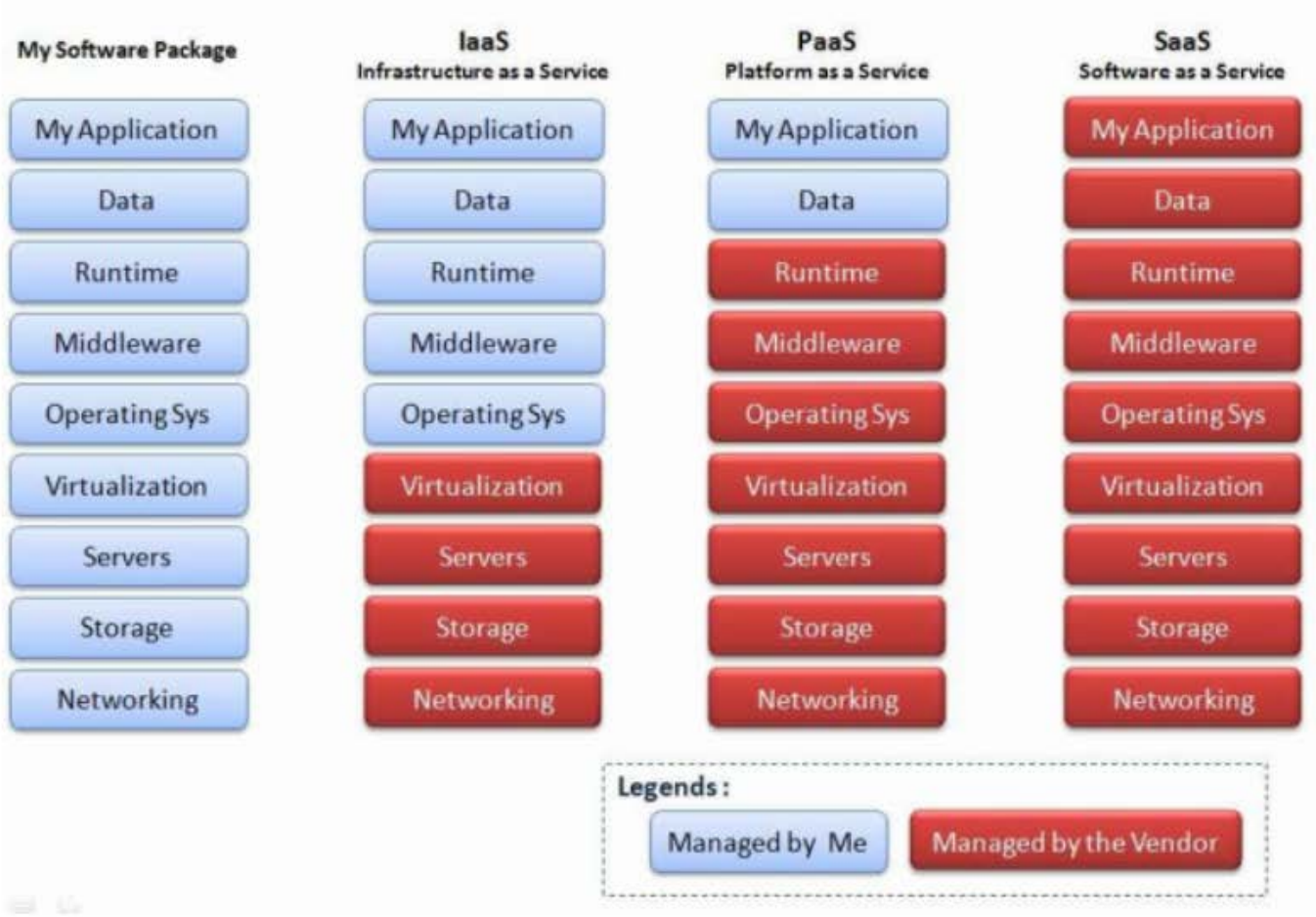
Soluzione app-based

- Svantaggi:
  - Il client dedicato necessita di continui aggiornamenti sul client. Questo problema viene affrontato da parte dei sistemi operativi mediante meccanismi di aggiornamento automatici, versatili per l'utente, ma a volte molto onerosi
  - La realizzazione di un client dedicato e più complessa e onerosa

(E' possibile valutare il tipo di app: dalla app nativa, alla ibrida, alla progressive webapp, alla webapp)

Soluzione distribuita Web-based (cloud approach)

Soluzione distribuita Web-based (cloud approach)



Soluzione distribuita Web-based (cloud approach)

- Vantaggi:
  - La soluzione Cloud consente di esternalizzare completamente la gestione dell'hardware e dei servizi di base (sistema operativi, backup, redundancy, fault tolerance)
  - Massima scalabilità dell'hardware in caso di necessità
- Svantaggi:
  - Costi: in caso di web farm potrebbero essere molto elevati, non sempre è efficiente esternalizzare

Q&A

- Ci sono domande?

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
- Le principali specifiche di una architettura di un sistema distribuito web based sono:
  - Completa aderenza allo standard HTTP
  - Completa compatibilità con i browser Web disponibili
  - Apertura alle diverse tecniche e tecnologie di sviluppo software
  - Possibilità di ingegnerizzare il software e il processo di sviluppo
  - Scalabilità
  - Tolleranza ai guasti

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 e monitoring intrinseci della infrastruttura

Compatibilità con i browser

- I browser Web diventano una sorta di “scatola” in cui 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 in modo che il codice realizzato sia compatibile con i diversi browser sul mercato (ad esempio: non compatibilità tra i vari browser rispetto alla applicazione/implementazione dello standard CSS per i fogli di stile)

Tecniche e tecnologie di sviluppo software

- Il successo di una architettura è spesso legato al suo grado di apertura alla evoluzione tecnologica e alle tecnologie similari
- 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

Ingegnerizzare software e 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

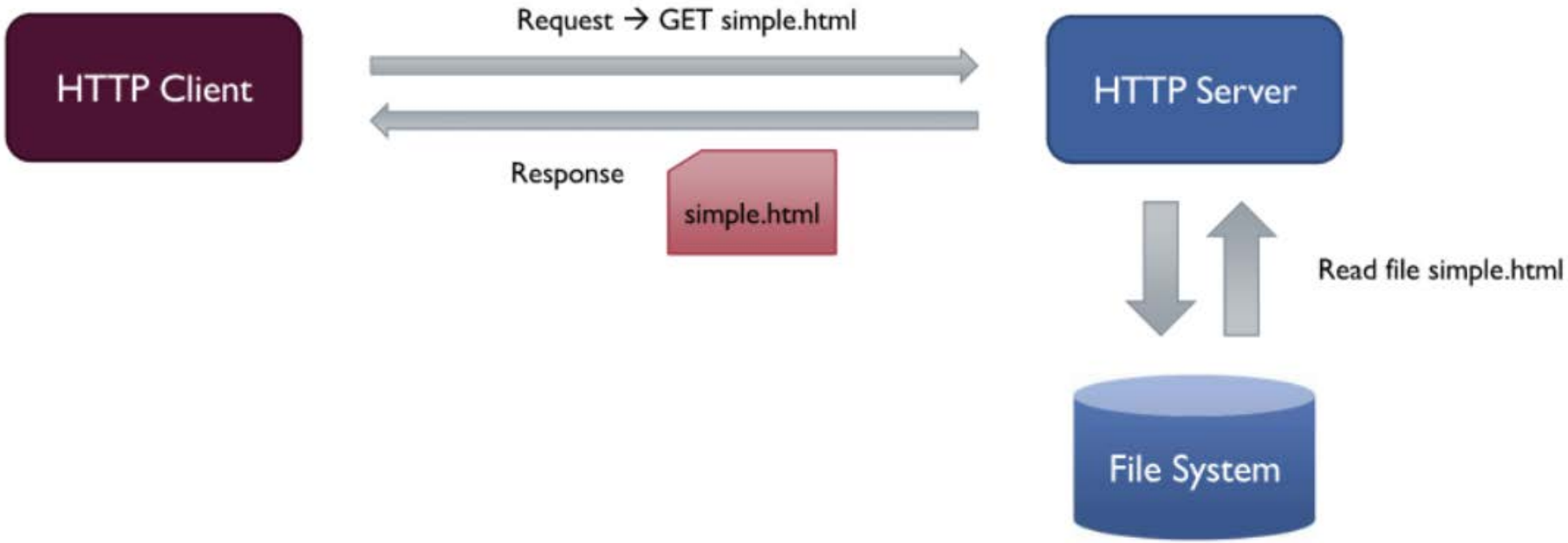
Scalabilità

- 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 ottenuta 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***: 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 il servizio mantenendo attive tutte le istanze a disposizione, questo permette di sfruttare tutte le risorse attraverso di politiche di distribuzione e bilanciamento del carico

Architettura di base



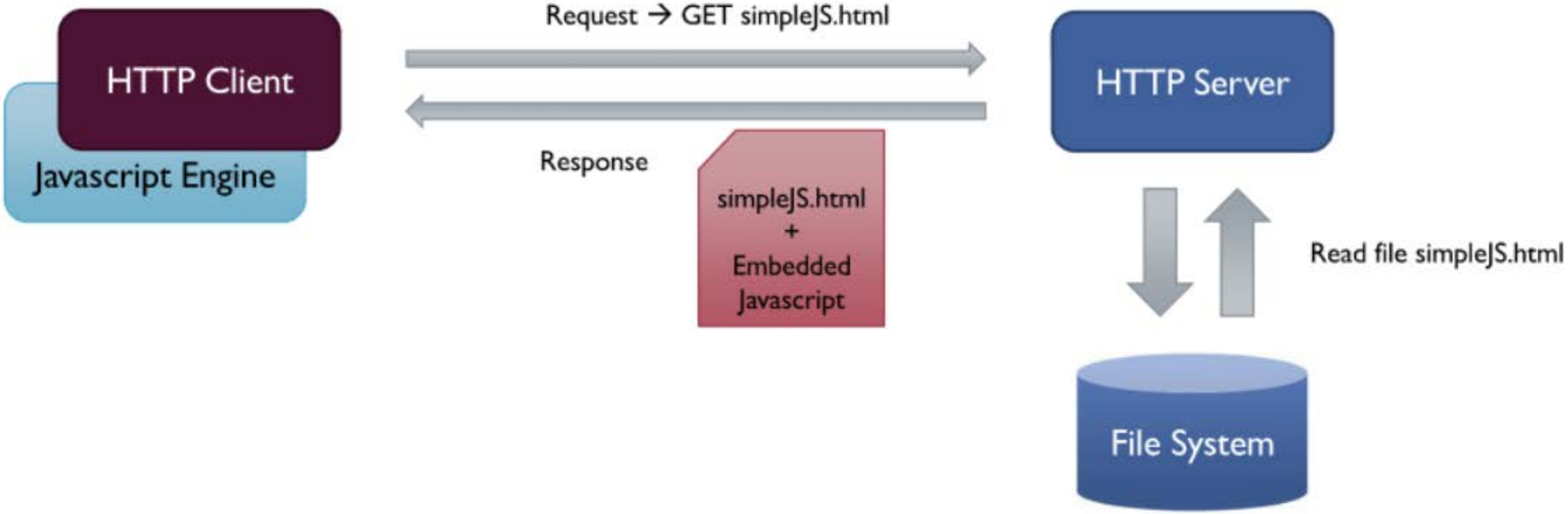
Architettura di base

- *Completa aderenza allo standard HTTP*: OK
- *Completa compatibilità con i browser disponibili*:
  - dipende dalla qualità con cui vengono scritte le pagine HTML,
  - esistono tool di sviluppo/valutazione che permettono di verificare ed ottimizzare la compatibilità cross-browser
- *Apertura alle diverse tecniche e tecnologie di sviluppo software*:
  - nessuna apertura, le applicazioni Web basate su questa architettura offrono solo un ambiente di navigazione ad ipertesti per la consultazione dei dati contenuti, non è possibile sviluppare codice al di fuori del codice HTML stesso

Architettura di base

- *Ingegnerizzare software e processo di sviluppo*:
  - molto limitata: possibilità di applicare principi di riusabilità del codice in contesti semplici, ma in modo rudimentale
- *Scalabilità*:
  - ottima: il server è stateless; è possibile affiancare quanti server si desidera che insistano sugli stessi sorgenti HTML; è possibile replicare sia i server che i sorgenti
- *Tolleranza ai Guasti*:
  - ottima: si ottiene implicitamente, data la possibilità di replicare il servizio

Architettura di base con scripting client-side



Architettura di base con scripting client-side

- *Completa aderenza allo standard HTTP*: OK
- *Completa compatibilità con i browser disponibili*:
  - dipende dalla qualità con cui vengono scritte le pagine HTML e dalla capacità del browser di interpretare il linguaggio di scripting (compatibilità rispetto agli standard)
- *Apertura alle diverse tecniche e tecnologie di sviluppo software*:
  - limitata alle capacità dell'interprete Javascript del browser

Architettura di base con scripting client-side

- Ingegnerizzare software e processo di sviluppo:
  - possibilità di applicare principi di riusabilità del codice realizzando semplici librerie importabili in diverse pagine
- Scalabilità:
  - ottima: il server è stateless; è possibile affiancare quanti server si desidera che insistano sugli stessi sorgenti HTML; è possibile replicare sia i server che i sorgenti
- Tolleranza ai Guasti:
  - ottima: si ottiene implicitamente, data la possibilità di replicare il servizio

Q&A

- Ci sono domande?

Riferimenti

- *Architecture of the World Wide Web, Volume One (https://www.w3.org/TR/2004/REC-webarch-20041215/)*
- *WEB ARCHITECTURE (https://www.w3.org/standards/webarch/)*
- Roads and Crossroads of the Internet History *http://www.netvalley.com/cgi-bin/intval/net_history.pl?chapter=1*
- HTTP - Hypertext Transfer Protocol: https://www.w3.org/Protocols/, https://tools.ietf.org/html/rfc2616, https://www.tutorialspoint.com/http/index.htm