

Web services

- **Tecnologia** per il computing *distribuito*
 - standard W3C
 - non dissimile da RMI, CORBA, EJB...
- **Relazione** con il Web
 - *Websites for humans, Web Services for software :-)*
 - un Web service ha un indirizzo http, ma non per il browser!
 - tuttavia, i WS *sfruttano tecnologie Web come trasporto*, specie il protocollo http,
 - una possibile e popolare implementazione, in ambiente Unix, sfrutta un componente, Axis, del server http *Apache*

Web Services (WS): prospettiva

- Il Web ha offerto il suo primo supporto alle *attività umane* attraverso un meccanismo per lo scambio di dati testuali e grafici
- Livello di interazione per lo più soddisfacente *per l'uomo*, ma che non consente una reale *interazione tra software*
- Necessità di una tecnologia che *permetta alle applicazioni di interagire* ed eseguire programmaticamente operazioni
- Applicazioni Internet-based devono poter (=avere meccanismi per)
 - *trovare* (discovery)
 - *accedere* (determinare le modalità di accesso)
 - *interagire* con
altre applicazioni Internet-based
- l'architettura dei WS mira a fornire meccanismi per l'*interazione application-to-application*, sfruttando le potenzialità del Web

WS: motivazioni

- **Web services**: ambiente distribuito che garantisce **interoperabilità** ad applicazioni remote, in modalità:
 - **indipendente dalla piattaforma**
 - **neutrale rispetto al linguaggio**
- WS rispetto all'**eterogeneità** delle tecnologie (linguaggi/piattaforme hw/OS)
 - la presuppongono e si pongono come **soluzione**
 - la affrontano ricorrendo a protocolli **standard**, già accettati o universalmente condivisi:
 - HTTP, SMTP,... come **trasporto (standard accettati)**
 - XML per la **codifica** dell'informazione relativa ai servizi supportati in modo indipendente dai linguaggi di sviluppo (**standard condivisi**)

Non Web services

- win32 e suoi oggetti distribuiti: COM, DCOM
- RMI
- J2EE e EJB
- CORBA
- CGI scripting

Al di là degli intenti di universalità (*cross-platform, cross-language*) di ciascuna di queste tecnologie, esse mancano di:

- **accettazione condivisa e/o**
- **supporto condiviso**

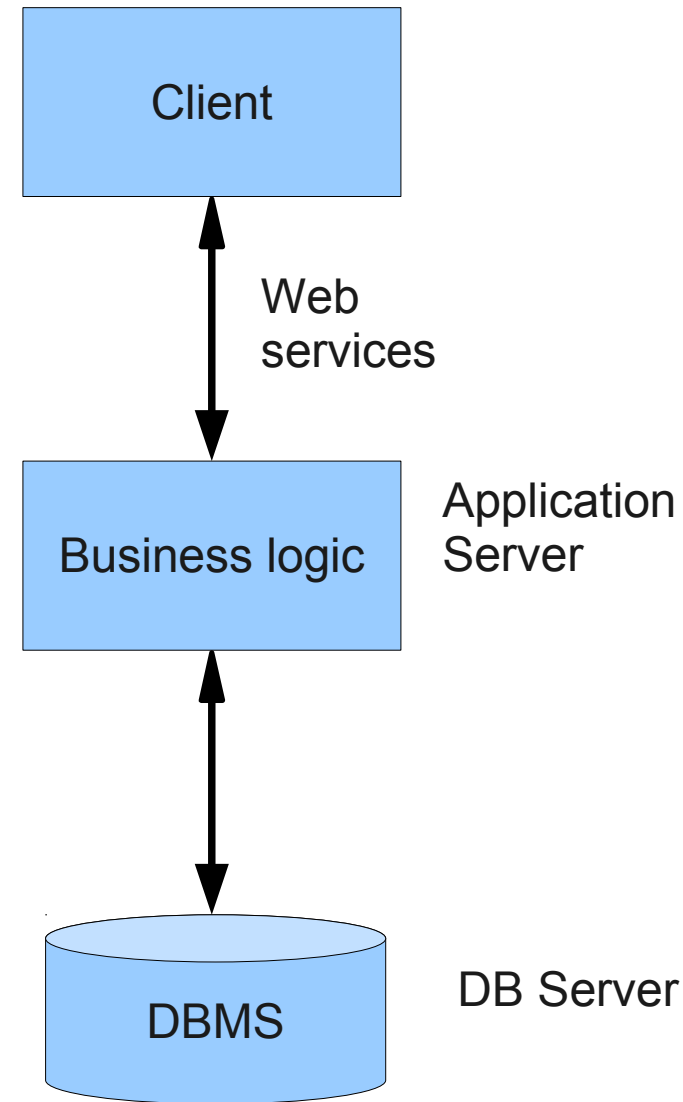
WS e architettura 3-tier

Dal punto di vista dell'architettura classica (a fianco) di un'applicazione distribuita:

- i WS rappresentano una maniera universale di presentare/impacchettare (parti di) **business logic**
 - attraverso Internet
 - con protocolli/codifiche standard
- esempi: Amazon, Google, ...

NB: WS può accedere/incapsulare altri WS

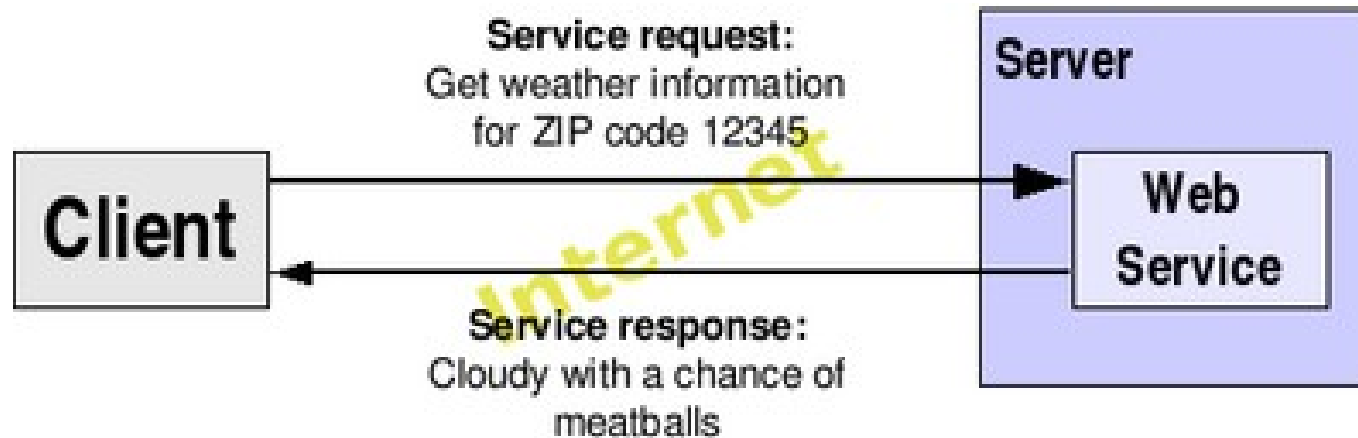
- p.es. WS RestaurantFinder sfrutta WS MapQuest



WS: caratteristiche

- Basati su XML
 - Rappresentazione dati: interoperabilità al cuore della tecnologia
 - Trasporto dati: nessun problema di networking, SO, piattaforma
- Lascamente accoppiati
 - Interfaccia modificabile senza (gravi) conseguenze sul client
- Granularità grossa
 - In modo naturale si accede alla giusta quantità di business logic
- Modalità sincrona o asincrona
 - asincronia tipica dei sistemi lascamente accoppiati
 - in modalità sincrona: supporto RPC
 - WS permettono compatibilità con componenti EJB e .Net

WS: schema di base



- lo schema di base è lo stesso di altre tecnologie *client-server*
- RPC, RMI, CORBA, EJB...
- ma...

WS vs. altre tecnologie

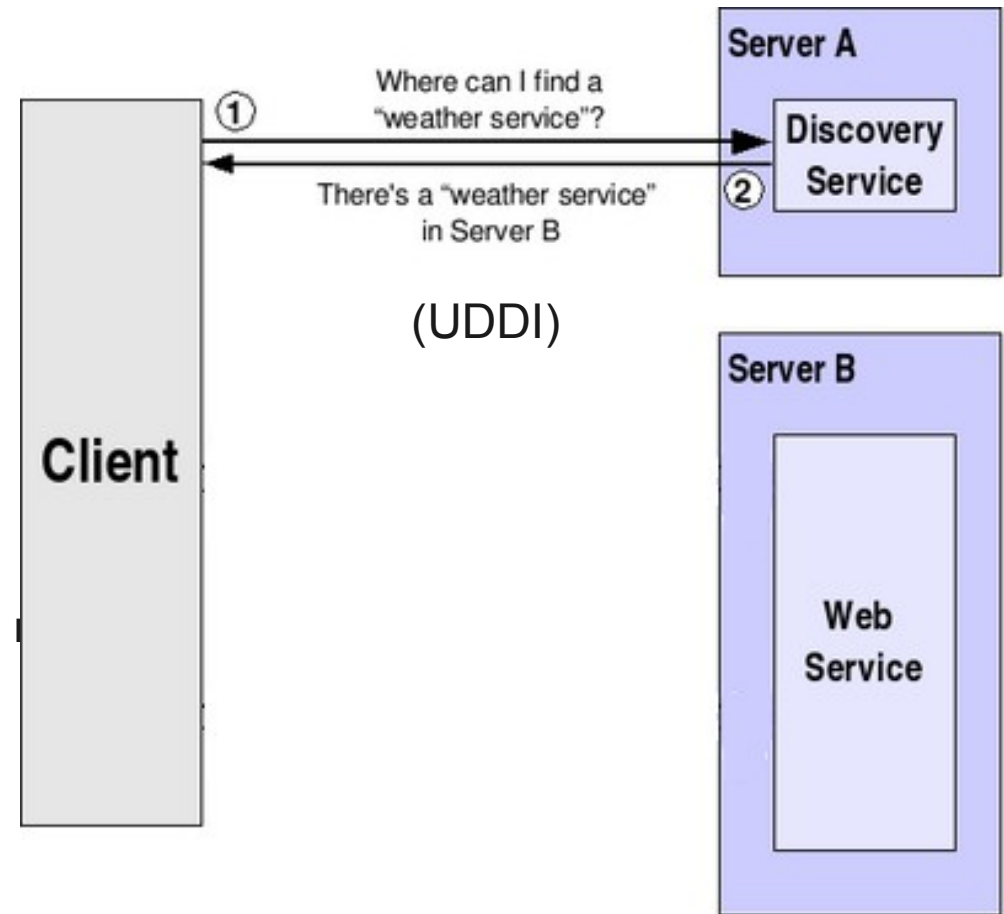
- *multiplatforma*: implementazioni per Unix, Windows...
- *multilinguaggio* per clients e server
 - usa XML come standard di interoperabilità, per la codifica dei messaggi
- trasporto per lo più via HTTP
 - passa senza problemi firewall e proxy di Internet
- overhead (XML vs. binary code) → no real time!
- (ancora) poco versatile, vs. p.es. CORBA:
 - non disponibili/standard (as of 2007) servizi aggiuntivi, p.es.:
 - persistenza,
 - gestione lifecycle degli oggetti (distruzione..),
 - transazioni...

WS: loosely vs. highly coupled

- Highly coupled (CORBA, EJB):
 - cliente molto dipendente dal server (deve conoscerne l'interfaccia)
 - adatto ad *applicazioni intranet*
- Loosely coupled (WS):
 - il cliente scopre a runtime l'**esistenza** del servizio
 - scopre a runtime la **struttura** del servizio
 - gli si adatta a runtime, se programmato per questo (cf. riflessione), per poterlo invocare
 - utile su scala *Internet-wide*

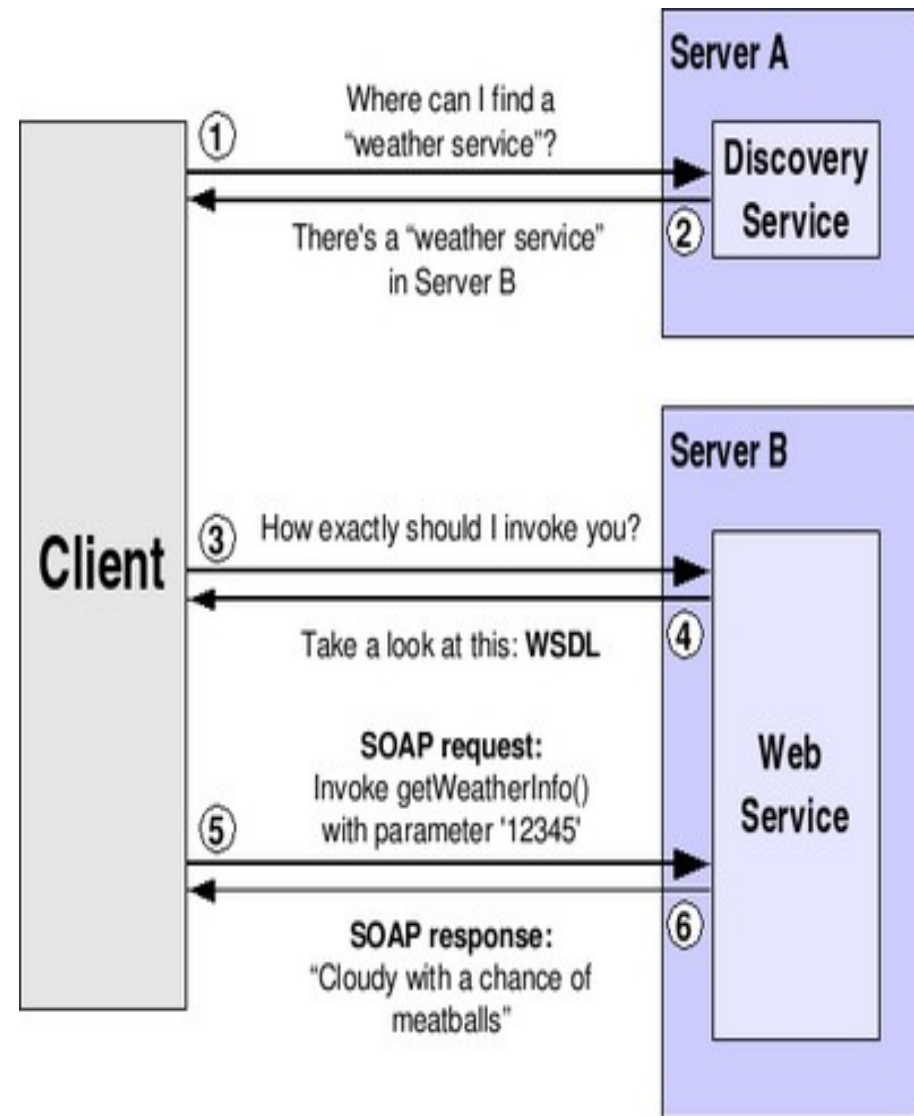
WS: esempio / discovery

- UDDI = Universal Description, Discovery and Integration
- protocollo basato su XML
- permette ai *provider* di pubblicare dati e Web services, attraverso dei *broker* o *directory* (Server A in figura)
- permette ai clienti di scoprire le *locazioni dei servizi* (Server B in fig.)
- Uno dei maggiori server UDDI: <http://seekda.com>



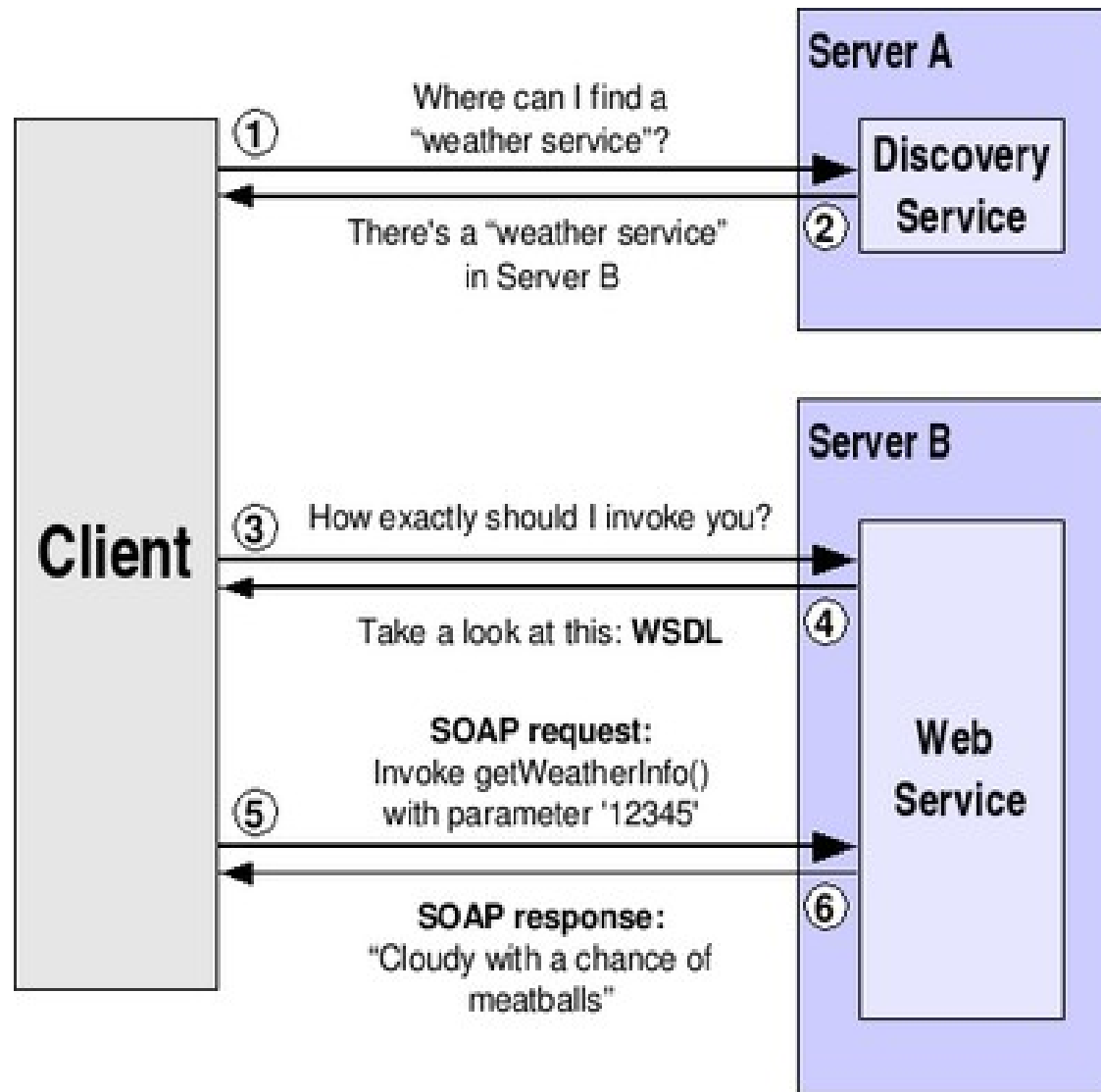
WS: esempio / uso; WSDL

- UDDI (o analoghi protocolli) dicono *dove* si trova il servizio,
- ma non *come* invocarlo
 - p.es.; Forecast
`getCityForecast(int CityPostalCode)?`
 - oppure string
`getUSCityWeather(string cityName, bool isFahrenheit)?`
- lo si viene a sapere attraverso il protocollo *WSDL* (pronuncia: “WISDEL”): Web Services Description Language basato su XML



WS: esempio / uso: SOAP

- SOAP = Simple Object Access Protocol
- messaggi SOAP in linguaggio basato su XML
- SOAP codifica le request dal client e la reply dal server
- possibile sia lo stile/paradigma di programmazione request-reply, che RPC (asincrono vs. sincrono)



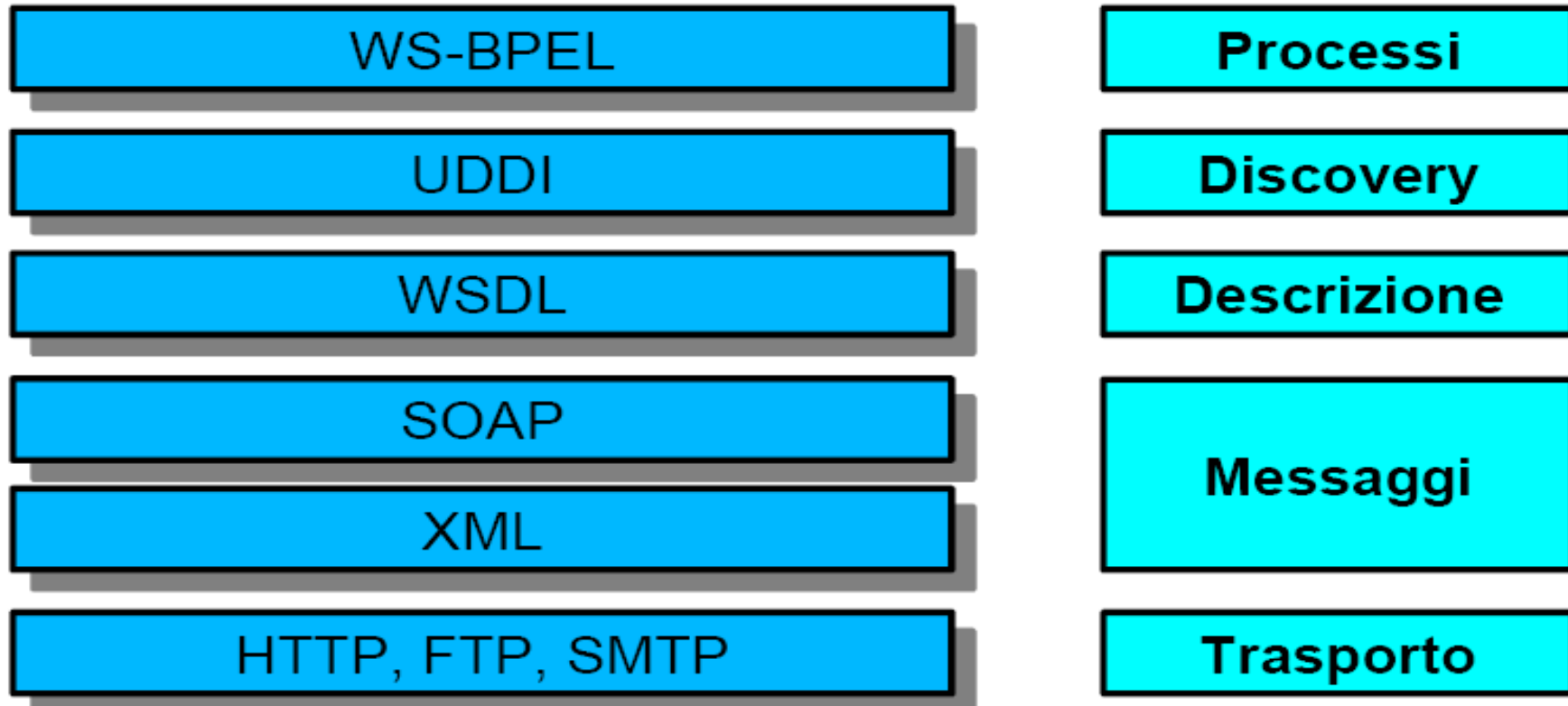
WS: tecnologie

- Molte tecnologie in competizione per strutturare Sistemi Distribuiti
 - Necessità di accordo sulla standardizzazione delle *core tech*
- Standard mondiali per il core delle tecnologie WS
 - SOAP
 - Standard di incapsulamento e trasporto documenti XML
 - Struttura semplice (evoluta da XML-RPC), permette interoperabilità tra client e server eterogenei
 - WSDL
 - Standard XML per descrivere le interfacce dei WS (parametri di I/O, struttura delle funzioni, etc.)
 - Permette ai client di capire a runtime come interagire con il WS
 - UDDI
 - Catalogo di WS (ricerca servizi, punti di accesso,...)

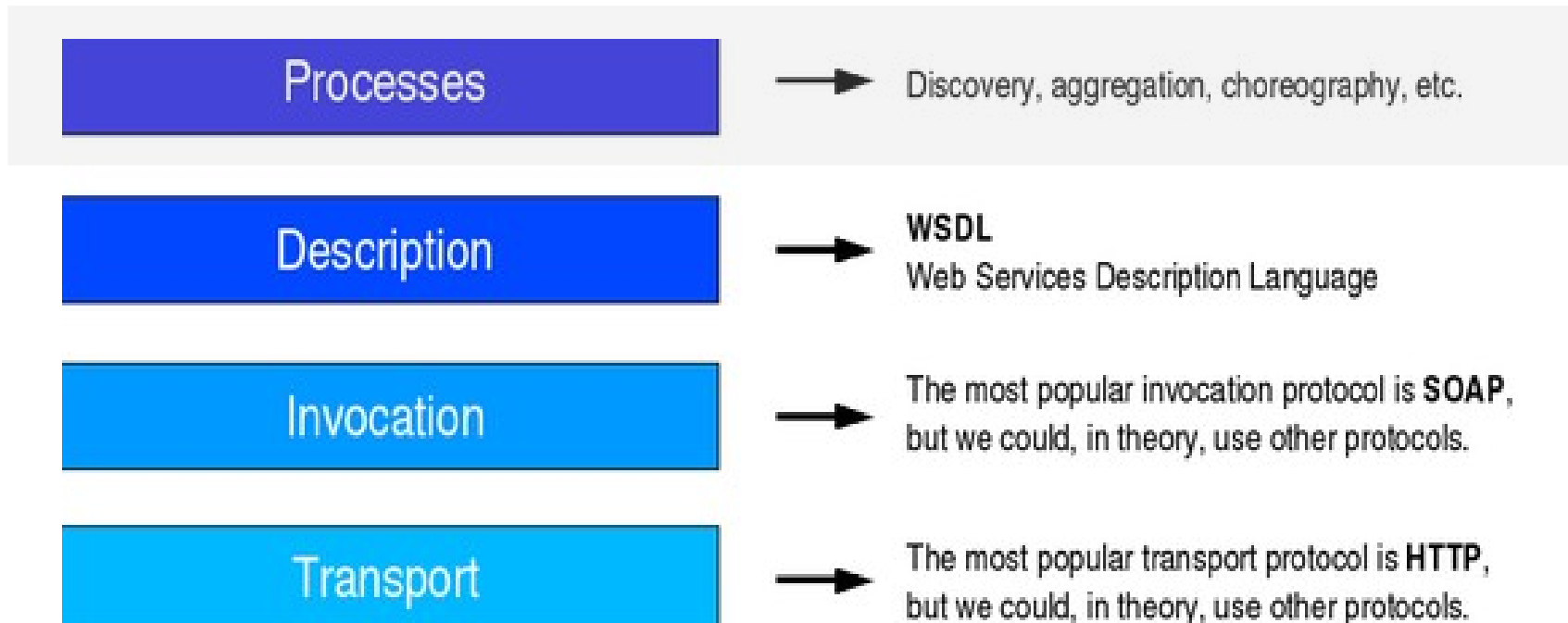
WS e XML

- XML è una famiglia di tecnologie
 - XML, XML Schema, XML namespace, Xpath, XSLT...
- Nel contesto dei WS le tecnologie XML sono usate per
 - specificare il formato dei messaggi da scambiare
 - consentire la validazione dei dati scambiati
 - definire gli stessi WS
- Si richiede sempre la definizione/stipula di un agreement sul significato degli elementi XML utilizzati

WS: architettura a “stack”



WS: stack / 1



- **Service Processes:** coinvolge più di un Web service
 - discovery (per esempio UDDI) si può pensare qui
 - serve a localizzare un particolare WS tra molti disponibili

WS: stack / 2



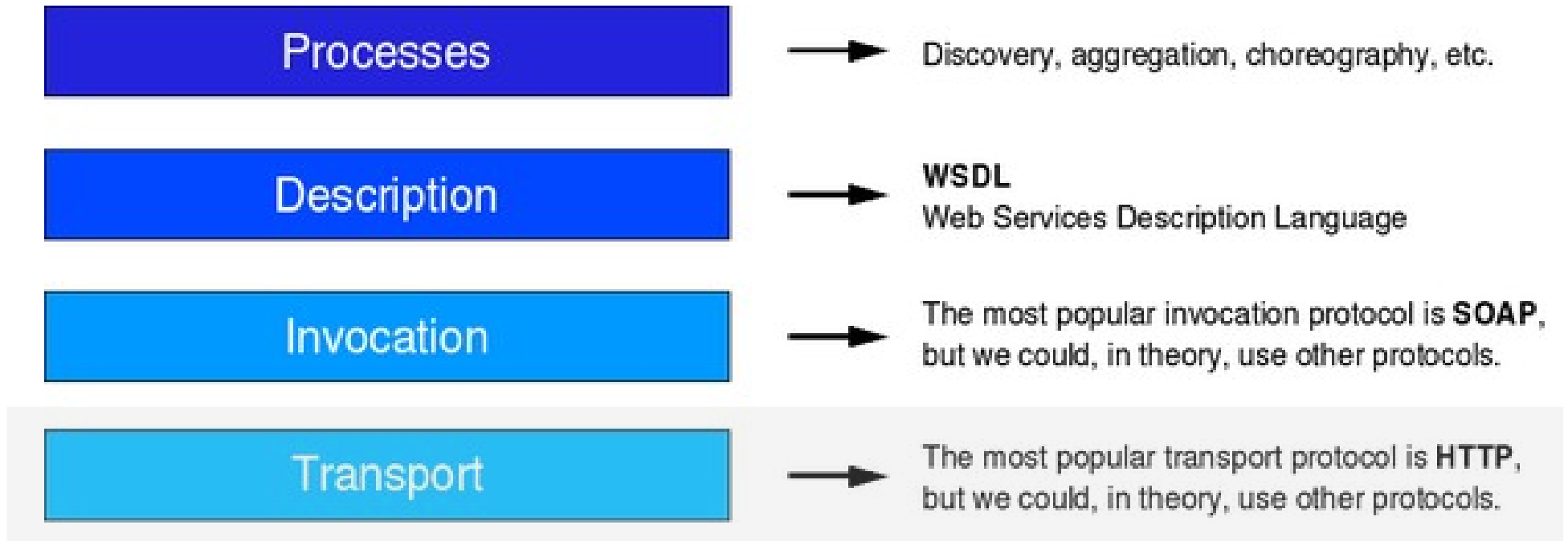
- Service Description: un WS è *self-describing*
 - una volta individuato e localizzato un Web Service via UDDI
 - gli si può chiedere di “descrivere se stesso”, comunicando che operazioni supporta e come vanno invocate
 - la comunicazione avviene secondo il Web Services Description Language (WSDL), basato su XML

WS: stack / 3



- (Web) Service Invocation:
 - implica passare dei messaggi tra client e server (come CORBA o EJB)
 - SOAP serve a specificare il formato di richieste (al server) e risposte (dal server)
 - sarebbe possibile usare altri linguaggi per l'invocazione, come xml-rpc o altri linguaggi ad-hoc basati su XML
 - ma SOAP è di gran lunga la scelta più diffusa

WS: stack / 4



- **Transport:**

- tutti i messaggi devono essere trasportati tra client e server server
- la scelta più diffusa è HTTP (HyperText Transfer Protocol)
- anche in questo caso sarebbe possibile usare altri protocolli

WS: addressing

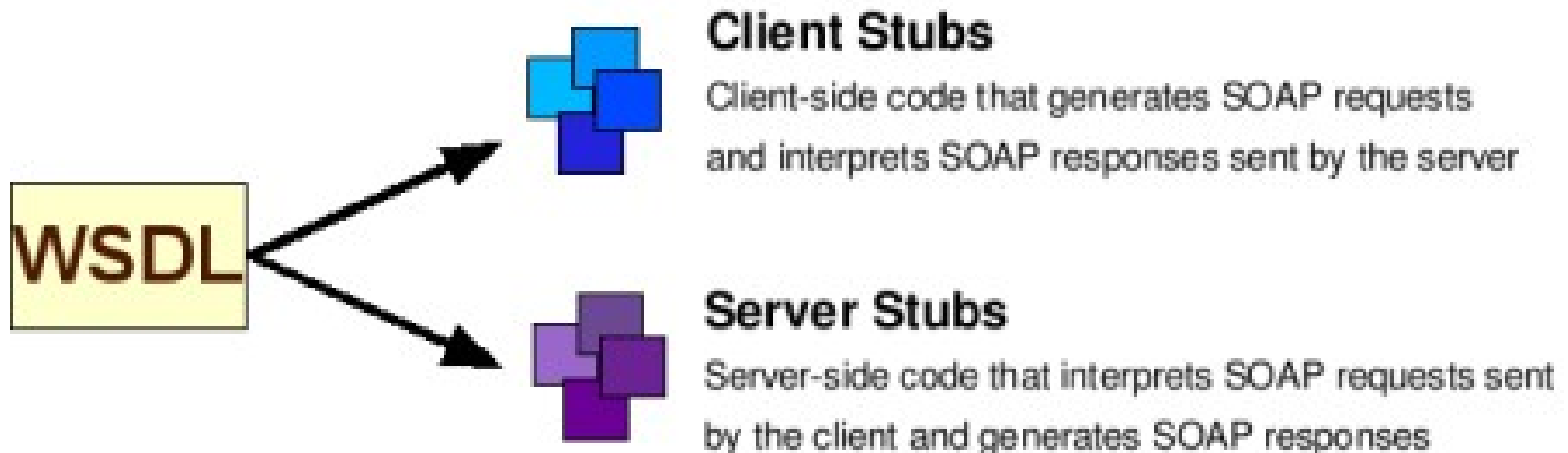
- I WS vengono indirizzati come pagine web, con URI (Uniform Resource Identifiers)
 - Un UDDI potrebbe restituire come indirizzo di un WS
<http://webservices.mysite.com/weather/us/WeatherService>
- I WS esistono per il software e non per l'uomo, quindi:
 - l'indirizzo passato ad un browser (spesso) restituisce errori o codici
 - certi server comunque possono gestire queste interrogazioni fornendo un'interfaccia utente
- L'indirizzo (URI) ottenuto è da usare da software
 - p. es., a un semplice cliente, l'URI del servizio da utilizzare si può passare come argomento sulla riga di comando

WS example: Weather Web Service (2010)

- provate <http://www.webservices.net/globalweather.asmx>
 - è l'endpoint URI del Web service, ma risponde anche al browser
- una [pagina Web di descrizione](#) di questo WS
- molto utile una [web interface](#) per invocare i metodi del WS
 - L'operazione `getWeather ()` richiede i parametri `CityName` e `CountryName`
 - il Web service restituisce una struttura XML con dati meteo
- La [descrizione WSDL](#) del Web service

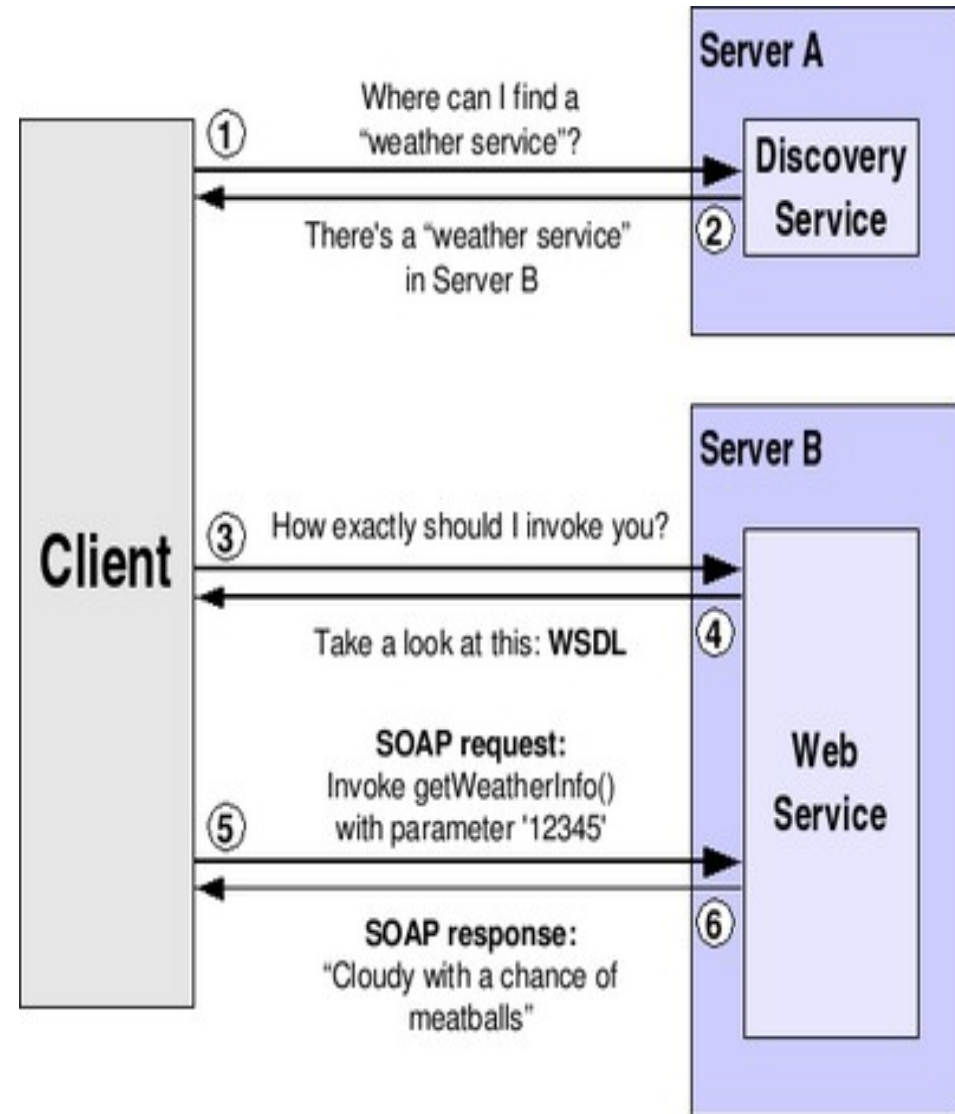
WS: la tecnologia di sviluppo

- un programmatore WS deve potersi concentrare sullo sviluppo nel linguaggio familiare; scrive, al più, un po' di codice (XML) WSDL
- il codice (XML) SOAP è generato automaticamente e invocato attraverso chiamate “locali”, user-friendly
- la conversione di queste (e delle risposte) in messaggi SOAP è delegata a degli stub
- sono disponibili numerosi *stub generator* automatici a partire dalla descrizione WSDL del WS

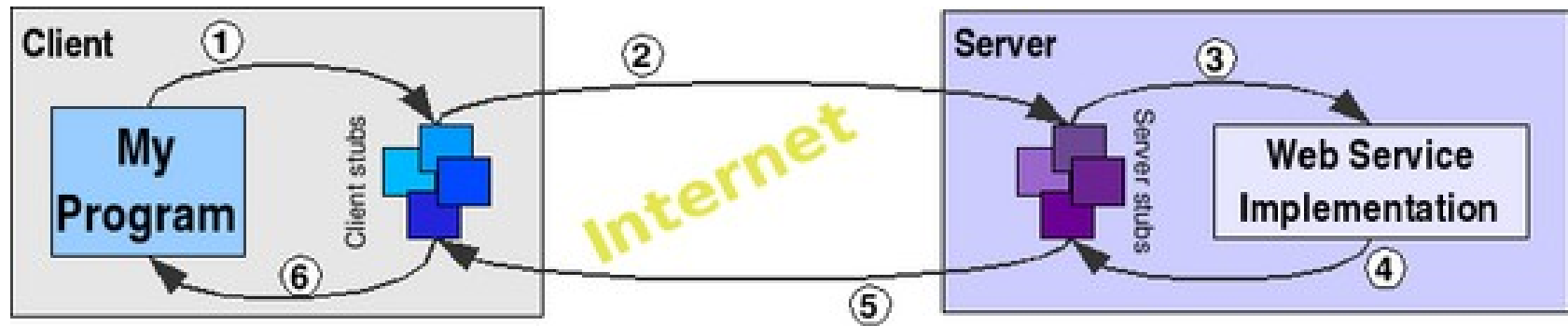


WS e stub

- rispetto alla tipica invocazione:
 - il discovery service **non** è invocato più volte, solo la prima
 - la descrizione WSDL **non** è richiesta più volte, solo la prima
 - è adesso che si genera lo stub
- rispetto a tecnologie come CORBA, RMI, RPC:
 - stub generato dinamicamente!
 - eventualmente rigenerato se il provider cambia interfaccia (cioè descrizione WSDL del WS)

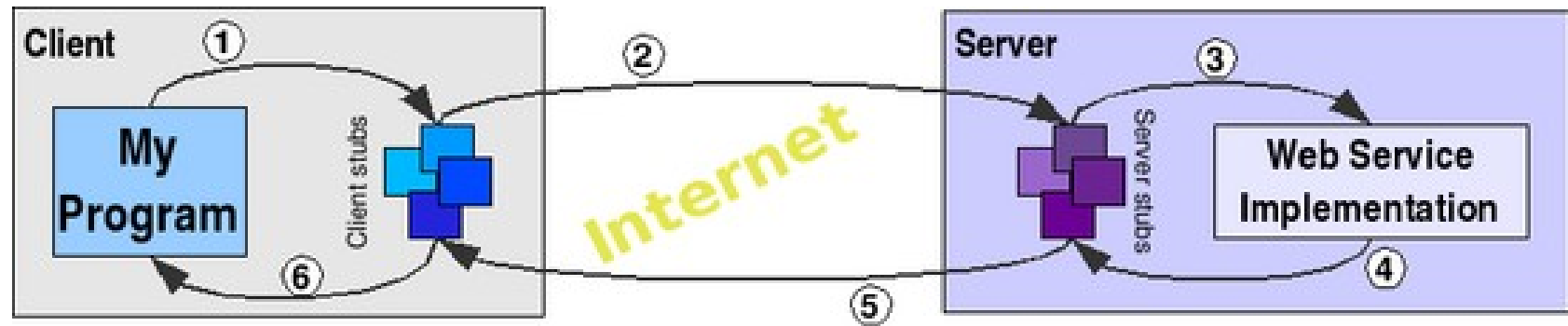


Invocazione WS: dettagli



1. per invocare WS, l'applicazione in realtà fa una chiamata "locale", semplice al client stub, che la converte in un messaggio SOAP di richiesta (processo di **marshalling** o **serializzazione**)
2. la richiesta SOAP viene inviata al server via rete usando HTTP; il server la passa al server stub,
3. il server stub converte (**unmarshalling** o **deserializing**) la richiesta SOAP in una richiesta (presumibilmente invocazione) all'implementazione del WS; questa esegue il servizio richiesto

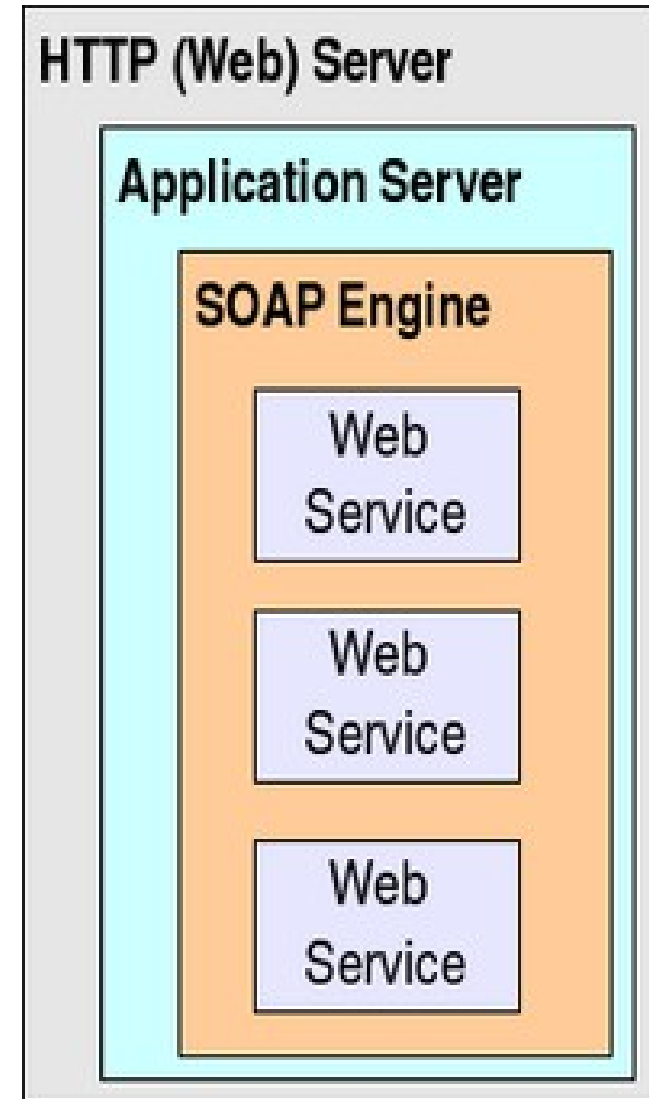
Invocazione WS: dettagli / 2



4. l'implementazione del WS passa il risultato dell'operazione richiesta al server stub che la converte in un messaggio di risposta SOAP
5. il messaggio viene inviato al client, in rete, via HTTP; verrà ricevuto dal client stub che la converte nella forma attesa dall'applicazione
6. l'applicazione riceve il risultato (ritorna dall'originaria chiamata (1)) e prosegue usandolo

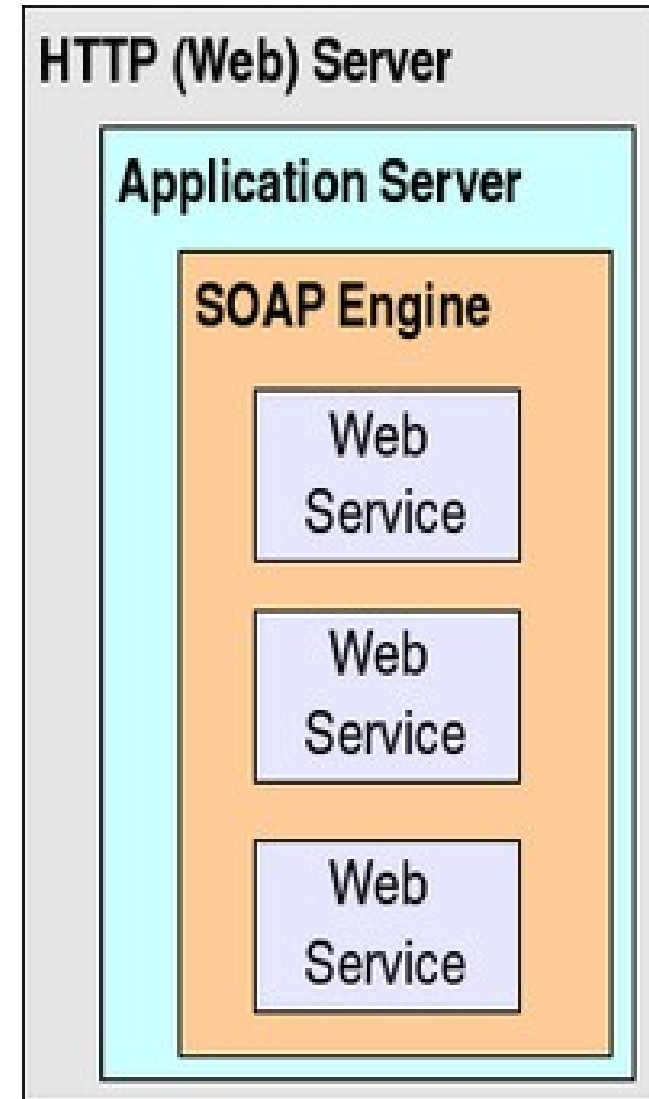
WS lato server: dettagli

- Ogni WS è un (sotto)sistema SW che espone *operazioni*
 - dettagli secondo linguaggio, ambiente, etc.
 - p.es. classe Java con metodi pubblici
 - ignora SOAP
- l'interazione con WS via SOAP è permessa da un “*server stub*” ad hoc
- o, più spesso, da un *SOAP engine* generico, che genera gli stub *on-the-fly*
- il SOAP engine non è però in grado di ricevere richieste dai clienti...
- esempio: *Apache Axis*



WS lato server: dettagli / 2

- il SOAP engine non è però in grado di ricevere richieste dai clienti...
- a ciò provvede l'*Application Server*, che può fungere da container anche per altre applicazioni
 - es. Tomcat, dà accesso a Axis, JSP, servlet
- l'AS può disporre di funzionalità HTTP autonoma, o può essere raggiunto, via HTTP, passando attraverso un WS
 - p.es. modulo Tomcat dentro il WS Apache
- si può incontrare il termine *WS container* = WS+AS+Soap Engine



SOA: Service-Oriented Architecture

In computing, the term service-oriented architecture expresses a perspective of software architecture that defines the use of loosely coupled software services to support the requirements of the business processes and software users. In an SOA environment, resources on a network are made available as independent services that can be accessed without knowledge of their underlying platform implementation.

“Migrating to a service-oriented architecture”, IBM DeveloperWorks