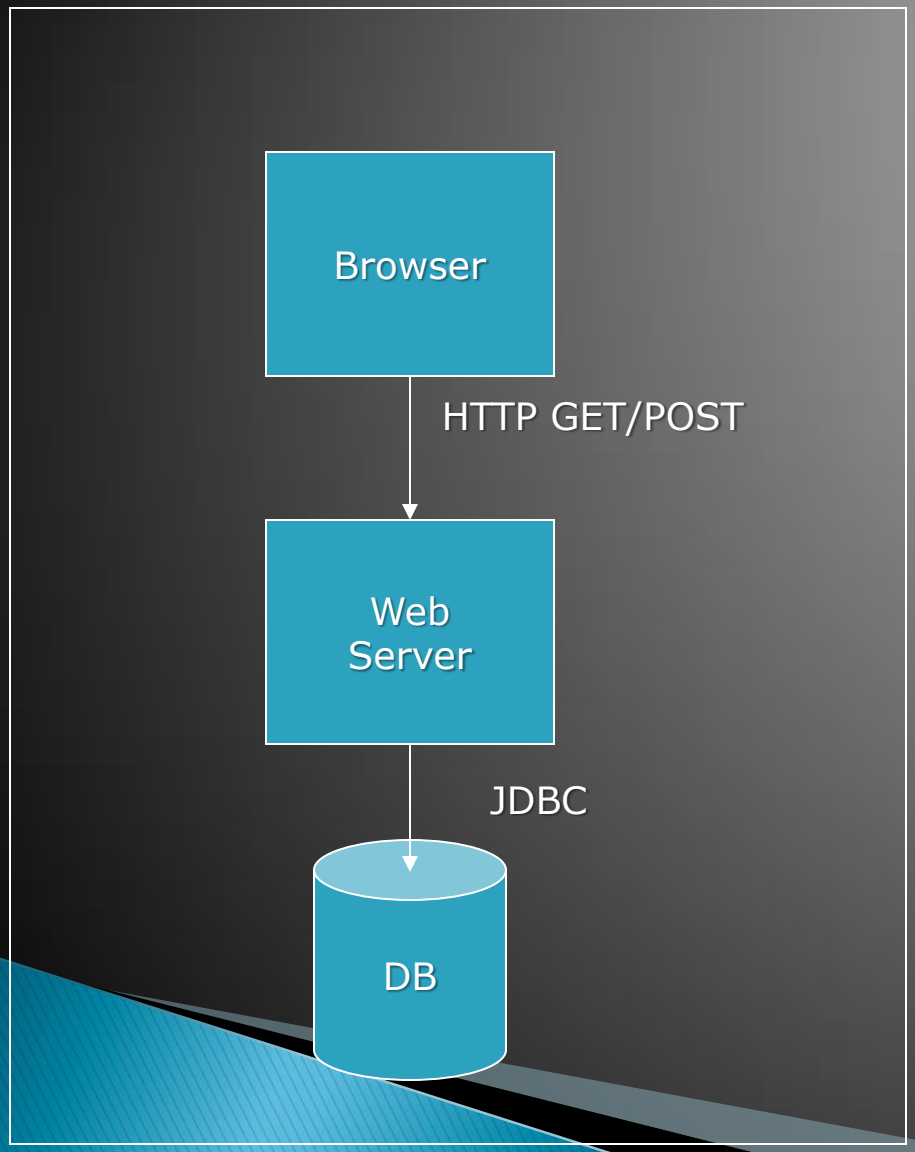
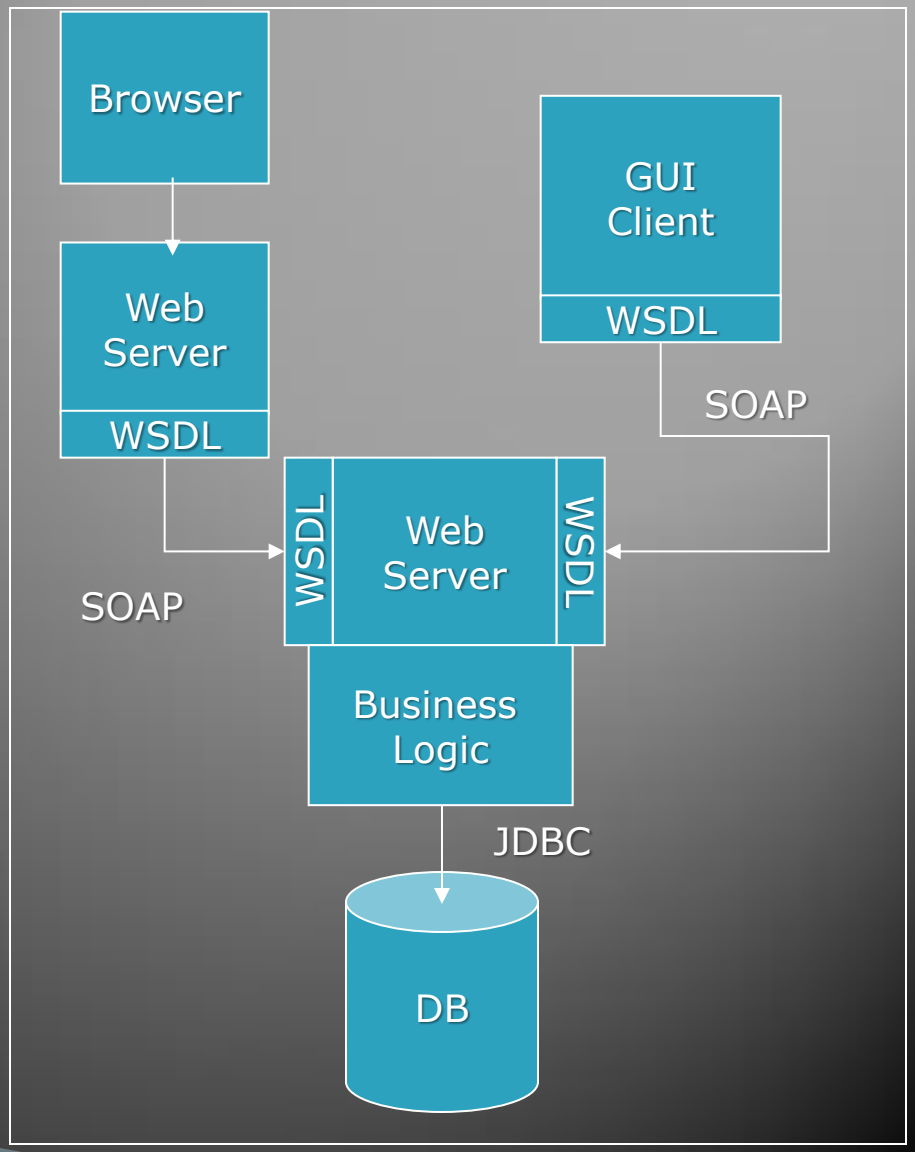
What Are Web Services?

- Web services framework is an XML-based distributed service/component system.
  - SOAP, WSDL, UDDI
  - Intended to support machine-to-machine interactions over the network.
- Basic ideas is to build an platform and programming language-independent distributed invocation system out of existing Web standards.
  - Most standards defined by W3C, Interoperability works, as long as you can map XML message to a programming language type, structure, class, etc.
- Very loosely defined, when compared to CORBA, etc.
- Inherit both good and bad of the web
  - Scalable, simple, distributed
  - But no centralized management, system is inefficient, must be tolerant of failures.

Basic Architectures: Servlets/CGI and Web Services





Some Terminology

- **SOAP**: Simple Object Access Protocol
  - XML Message format between client and service.
- **WSDL**: Web Service Description Language.
  - Describes how the service is to be used
  - Compare (for example) to Java Interface.
  - Guideline for constructing SOAP messages.
  - WSDL is an XML language for writing **Application Programmer Interfaces** (APIs).

Amazon and Google Experiment with Web Services

- Both Google and Amazon have conducted open experiments with Web services.
- Why? To allow partners to develop custom user interfaces and applications that work Google and Amazon data and services.
- You can download their APIs and try them.
  - http://www.google.com/apis/
  - http://www.amazon.com/webservices

When To Use Web Services?

- Applications do not have severe restrictions on reliability and speed.
- Two or more organizations need to cooperate
  - One needs to write an application that uses another's service.
- Services can be upgraded independently of clients.
- Services can be easily expressed with simple request/response semantics and simple state.
  - HTTP and Cookies, for example.

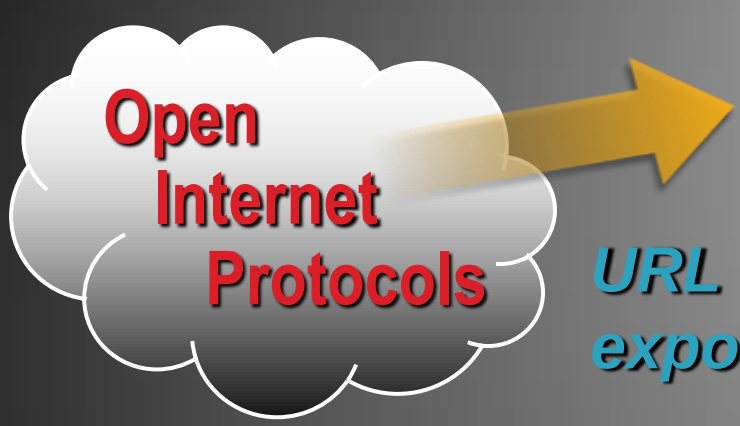
Web Service Architectures

- The following examples illustrate how Web services interact with clients.
- For us, a client is typically a JSP, servlet that a user accesses through browser.
- You can also build other clients
  - Web service **interoperability** means that clients and services can be in different programming languages (C/C++, python, java, etc).

Before Going On…

- But in practice, you don't need to work directly with either.
  - Most tools that I'm familiar with generate the WSDL for you from your class.
  - SOAP messages are constructed by classes.
  - Generated client stubs will even hide SOAP

What Is A Web Service?



Web Service

URL -addressable set of functionality exposed over a network

- **Provide a registry of Services on the Internet**
- **Web Services are defined in terms of the formats and ordering of messages**
- **Web Services consumers can send and receive messages using XML**
- **Built using open Internet protocols**

UDDI
Universal Description, Design, and Integration

WSDL
Web Services Description Language

SOAP

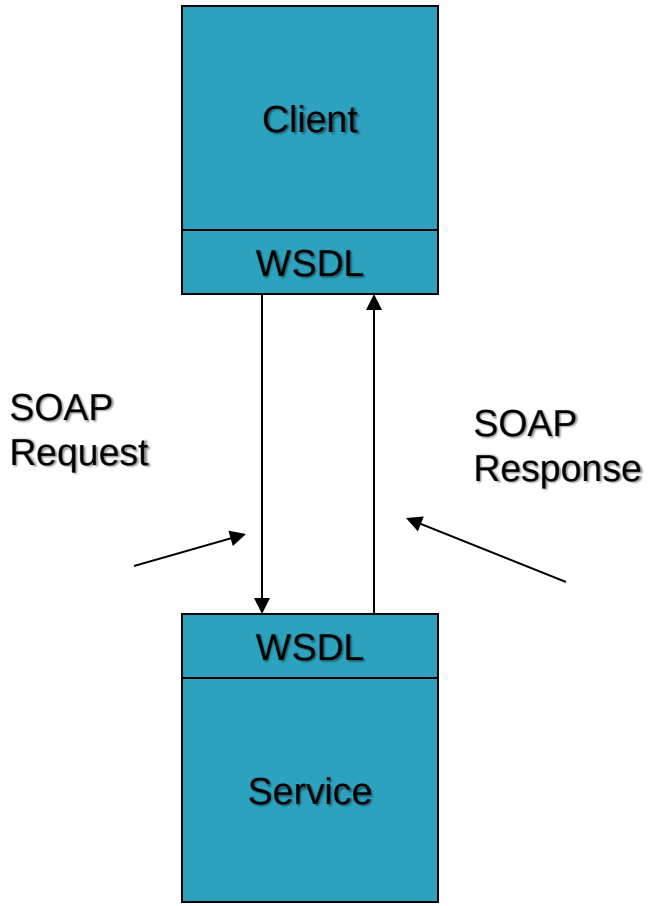
HTTP

SOAP(Simple Object Access Protocol)

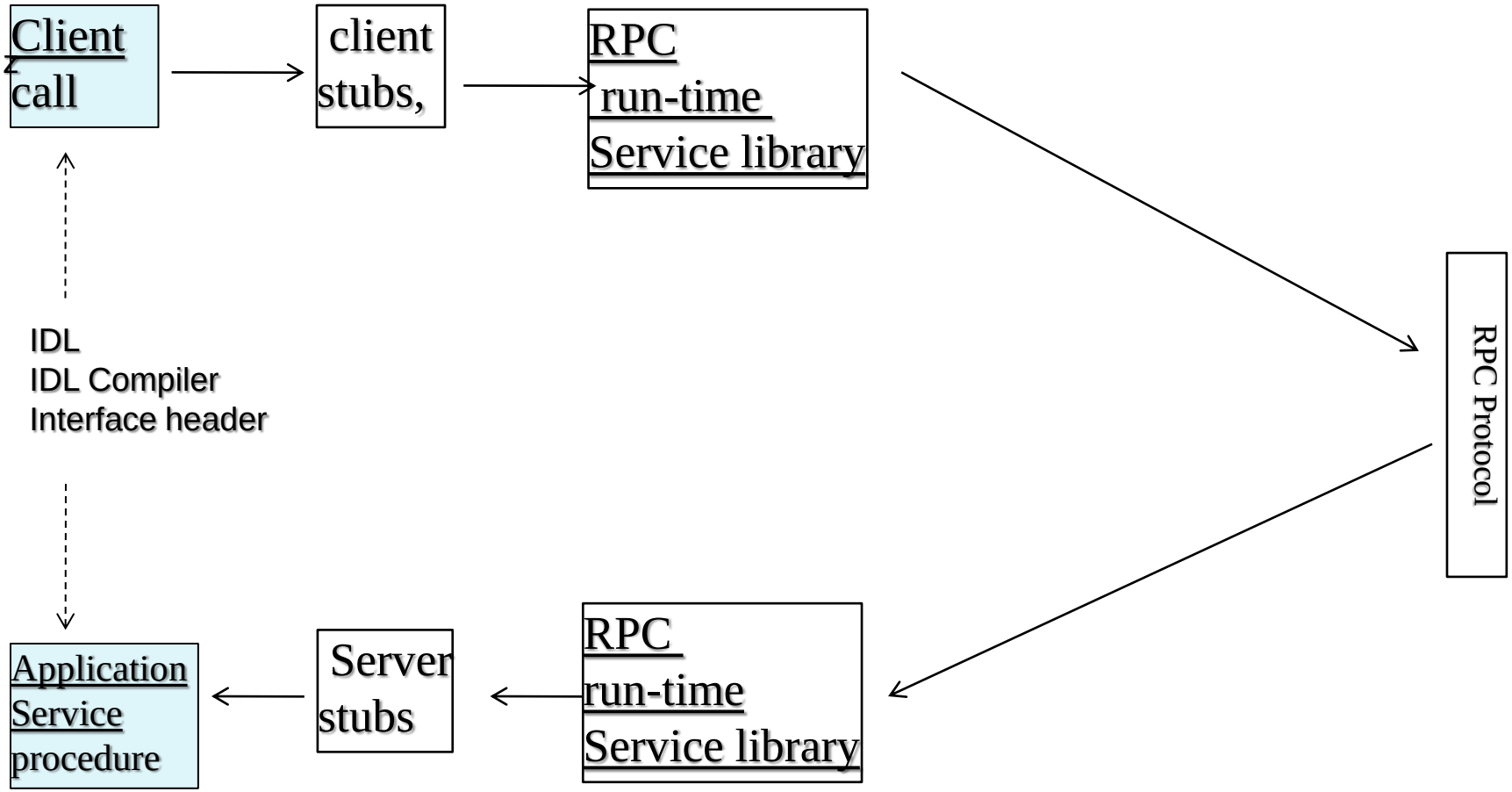
- The Simple Object Access Protocol
- **Platform-agnostic** protocol for messaging and RPC
- Encoded in XML, transported over HTTP
  - Data encoding described in **XML Schema**
  - Can use any transport mechanism
- Stateful? Stateless? (request/response)

SOAP and Web Services

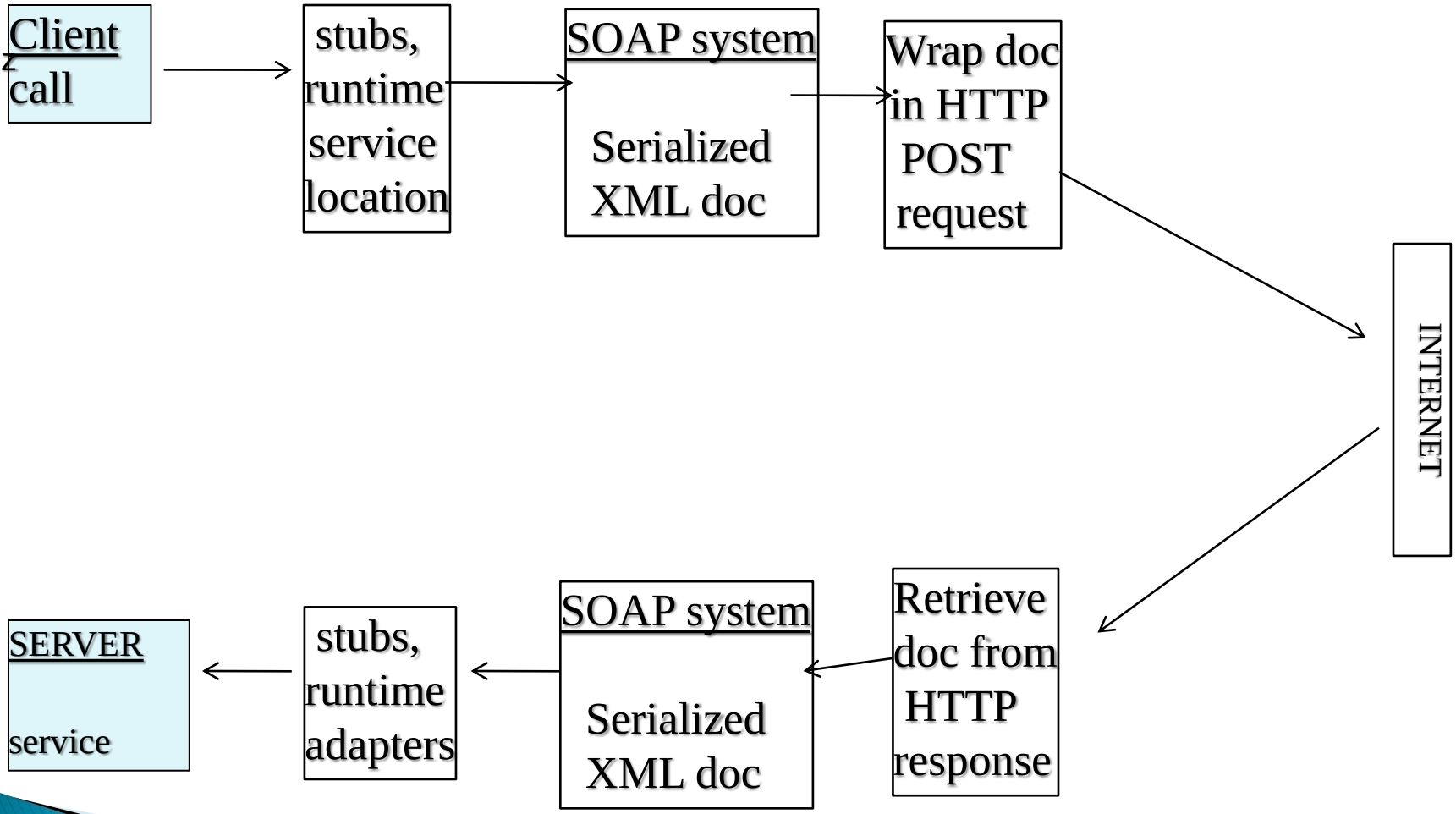
- Our previous lectures have looked at WSDL
  - Defines the interfaces for remote services.
  - Provides guidelines for constructing clients to the service.
- The actual communications are encoded with SOAP.
  - Transported by HTTP



RPC


Client call
client stubs,
RPC run-time Service library
RPC Protocol
IDL
IDL Compiler
Interface header
Application Service procedure
Server stubs
RPC run-time Service library

SOAP as RPC


Client call
stubs, runtime service location
SOAP system
Serialized XML doc
Wrap doc in HTTP POST request
INTERNET
Retrieve doc from HTTP response
SOAP system
Serialized XML doc
stubs, runtime adapters
SERVER
service

SOAP Basics

- SOAP is an XML message format for exchanging structured, typed data.
- It may be used for RPC in client-server applications but is also suitable for messaging systems (like JMS) that follow one-to-many (or publish-subscribe) models.
- SOAP is not a transport protocol. You must attach your message to a transport mechanism like HTTP.
- A SOAP client formats a message in XML including a SOAP "envelope" element describing the message
- The client sends the message to a SOAP server in the body of an HTTP request

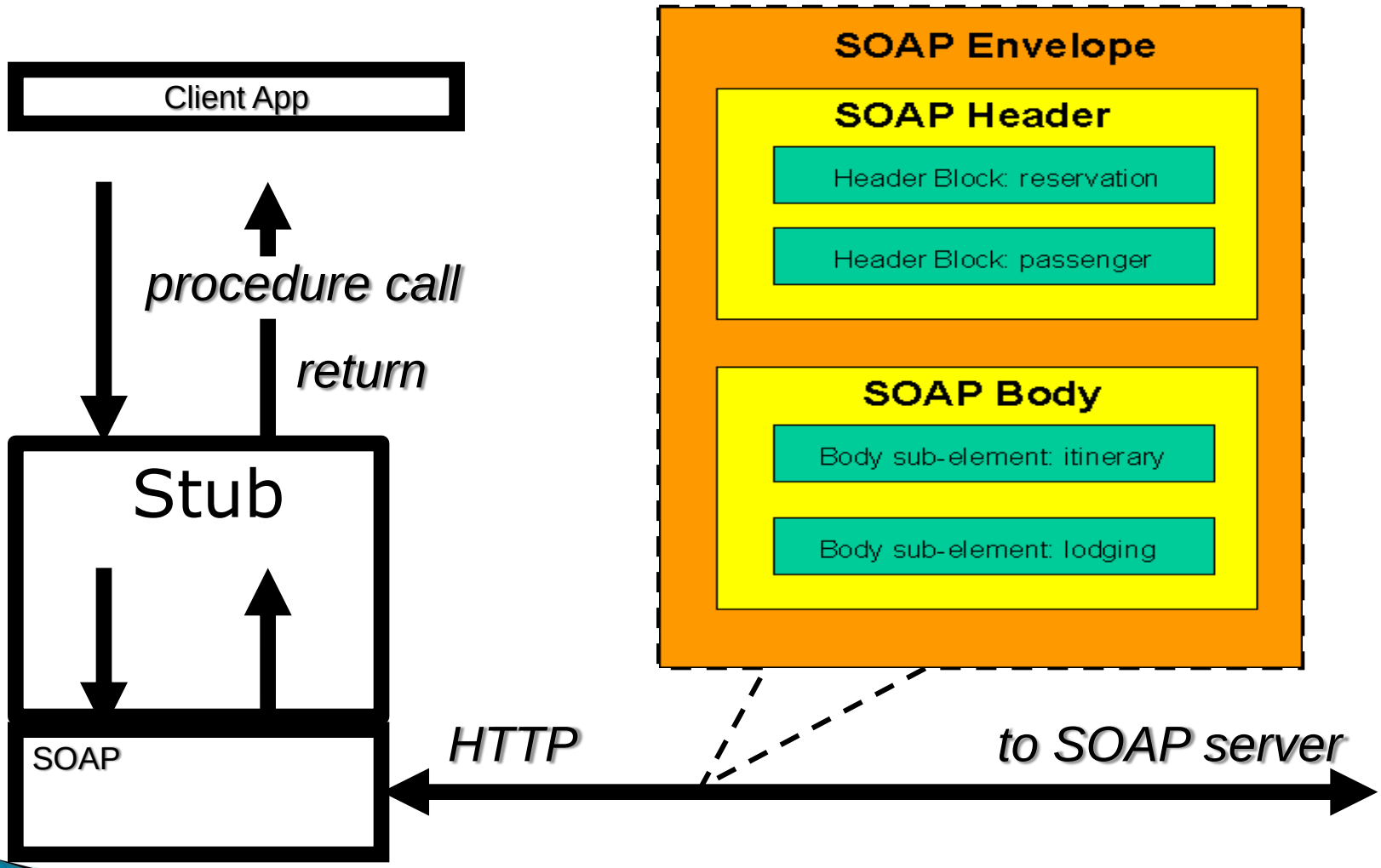
Web Service Messaging Infrastructure Requirements?

- Define a message format
  - Define a messaging XML schema
  - Allow the message to contain arbitrary XML from other schemas.
- Tell the message originator is something goes wrong.
- Define data encodings
  - That is, you need to tell the message recipient the types of each piece of data.
- Define some RPC conventions that match WSDL
  - Your service will need to process the message, so you need to provide some simple conventions for matching the message content to the WSDL service.
- Decide how to transport the message.
  - Generalize it, since messages may pass through many entities.
- Decide what to do about non-XML payloads (movies, images, arbitrary documents).

SOAP Encoding

- We specify SOAP encoding
- SOAP is a message format and needs a transport protocol, so we specify HTTP.
- Operation styles may be either “RPC” or “Document”.
  - We use RPC.
- SOAP Body elements will be used to actually convey message payloads.
  - RPC requires “encoded” payloads.
    - Each value (echo strings) is wrapped in an element named after the operation.
    - Useful RPC processing on the server side.
  - Documents are literal (unencoded)
    - Use to just send a payload of XML inside SOAP.

SOAP Use Scenario: RPC



SOAP Message Components

- Envelope (required)
  - Contains Header and Body
  - Defines namespaces (optional)
  - Declares encoding rules (optional)
- Header (optional)
  - Contains metadata entries about message *a la* HTTP headers
  - Specifies which entries must be understood and by which target “actor” in chain of recipients

SOAP Message Components

- Body (required)
  - Contains application-specific message
  - May be encoded variously
- Fault element (optional)
  - Contained in Body
  - Describes error class (version mismatch, headers not understood, client error, server error)
  - Extensible, hierarchical fault codes, e.g.

    `Server.Availability.NotAvailable`

Example SOAP Message

```
<SOAP-ENV:Envelope>
  <SOAP-ENV:Header>
    <lib:PatronID
     SOAP-ENV:mustUnderstand="1">
       007
    </lib:PatronID>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <lib:Checkout>
      <callNo>435.33</callNo>
    </lib:Checkout>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

header

body

Encoding Data in SOAP

- SOAP permits arbitrary “encoding styles” and defines a default encoding style
- Based on XML-Schema
- Supports data types
  - All built-in XML-Schema types (e.g. string, float, integer, date, IDREF)
  - Derived types: enumerations, arrays, structs, generic compound types

Example from SOAP1.2 part0

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
<env:Header>
<m:reservation xmlns:m=http://travelcompany.example.org/reservation
        env:role=http://www.w3.org/2003/05/soap-envelope/role/next
        env:mustUnderstand="true">
        <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
        <m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:dateAndTime>
</m:reservation>
<n:passenger xmlns:n=http://mycompany.example.com/employees
        env:role=http://www.w3.org/2003/05/soap-envelope/role/next
        env:mustUnderstand="true">
        <n:name>Åke Jógvan Øyvind</n:name>
</n:passenger>
</env:Header>
<env:Body>
        <p:itinerary xmlns:p="http://travelcompany.example.org/reservation/travel">
        <p:departure>
                <p:departing>New York</p:departing>
                <p:arriving>Los Angeles</p:arriving>
                <p:departureDate>2001-12-14</p:departureDate>
                <p:departureTime>late afternoon</p:departureTime>
                <p:seatPreference>aisle</p:seatPreference>
        </p:departure>
        <p:return>
                <p:departing>Los Angeles</p:departing>
                <p:arriving>New York</p:arriving>
                <p:departureDate>2001-12-20</p:departureDate>
                <p:departureTime>mid-morning</p:departureTime>
                <p:seatPreference/>
        </p:return>
        </p:itinerary>
        <q:lodging xmlns:q="http://travelcompany.example.org/reservation/hotels">
                <q:preference>none</q:preference>
        </q:lodging>
</env:Body>
</env:Envelope>
```

SOAP RPC Request

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope" >
<env:Header>
        <t:transaction xmlns:t="http://thirdparty.example.org/transaction"
        env:encodingStyle="http://example.com/encoding" env:mustUnderstand="true"
        >5</t:transaction>
</env:Header>
<env:Body>
        <m:chargeReservation env:encodingStyle="http://www.w3.org/2003/05/soap-
        encoding" xmlns:m="http://travelcompany.example.org/">
        <m:reservation xmlns:m="http://travelcompany.example.org/reservation">
        <m:code>FT35ZBQ</m:code>
        </m:reservation>
        <o:creditCard xmlns:o="http://mycompany.example.com/financial">
        <n:name xmlns:n="http://mycompany.example.com/employees"> Åke Jógvan
        Øyvind </n:name>
        <o:number>123456789099999</o:number>
        <o:expiration>2005-02</o:expiration>
        </o:creditCard>
        </m:chargeReservation>
</env:Body>
</env:Envelope>
```

RPC Response w/ return value

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope" >
<env:Header>
        <t:transaction xmlns:t="http://thirdparty.example.org/transaction"
        env:encodingStyle="http://example.com/encoding"
        env:mustUnderstand="true">5</t:transaction>
</env:Header>
<env:Body>
        <m:chargeReservationResponse
        env:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
        xmlns:m="http://travelcompany.example.org/">
        <rpc:result>m:status</rpc:result>
        <m:status>confirmed</m:status>
        <m:code>FT35ZBQ</m:code>
        <m:viewAt>
        http://travelcompany.example.org/reservations?code=FT35ZBQ
        </m:viewAt>
        </m:chargeReservationResponse>
</env:Body>
</env:Envelope>
```

WSDL Overview

- WSDL is an XML-based Interface Definition Language.
  - You can define the APIs for all of your services in WSDL.
- WSDL docs are broken into five major parts:
  - **Data definitions** (in XML) for custom types
  - **Abstract message definitions** (request, response)
  - Organization of messages into "**ports**" and "**operations**" (→classes and methods).
  - **Protocol bindings** (to SOAP, for example)
  - **Service point locations** (URLs)
- Some interesting features
  - A single WSDL document can describe several versions of an interface.
  - A single WSDL doc can describe several related services.

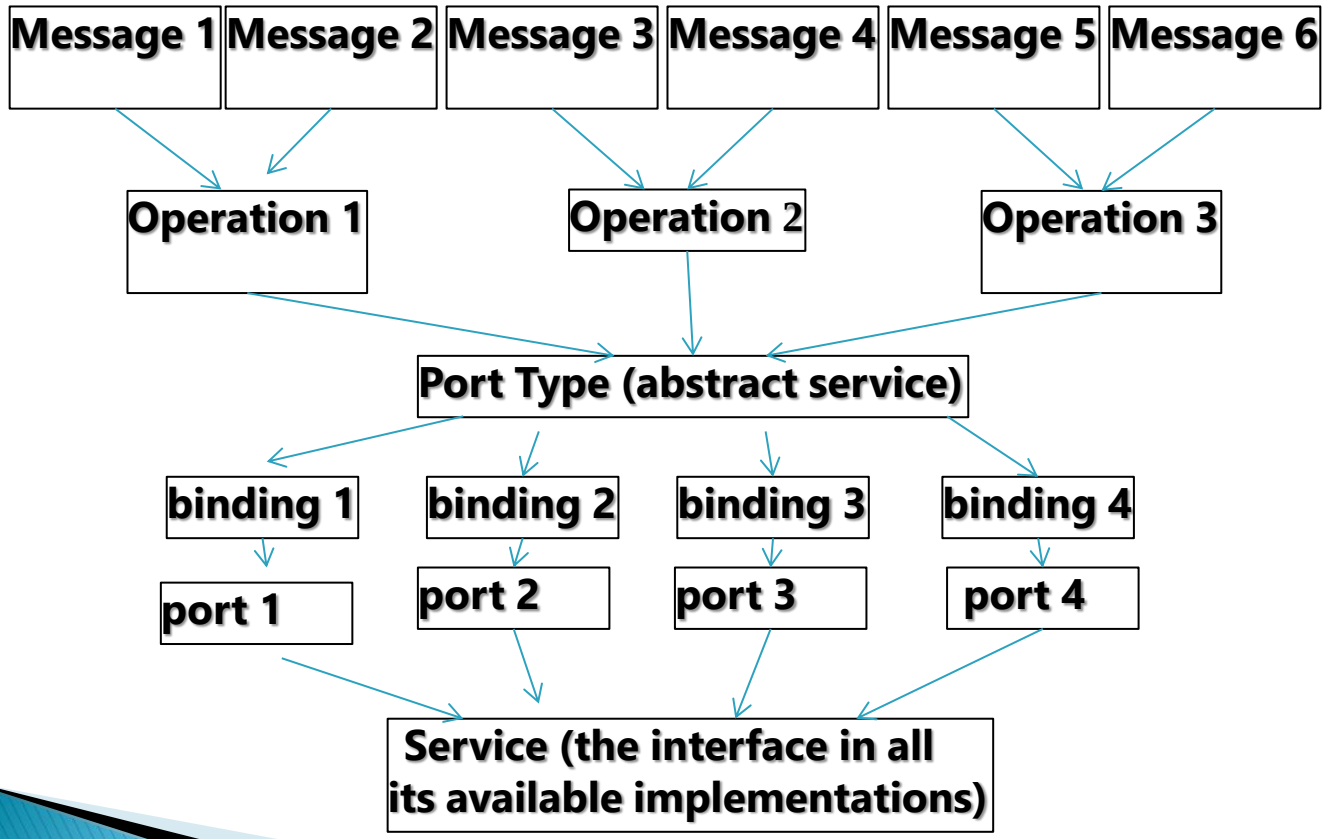
The Full WSDL

- WSDL is very **verbose**
  - Typically, you don't write WSDL
  - This file was actually generated from my Java class by **Apache Axis**.
- We will go through the parts of the doc in some detail.

WSDL includes?

1. Data types
2. messages (parameters)
3. operations (input and output)
4. port types (combine operations into group)
5. bindings (connect port types to a protocol)
6. location of service

Elements of WSDL 1.1



Abstract description of the service

Concrete description of the service

Apache Axis Overview

- Apache Axis is a toolkit for converting Java applications into Web services.
- Axis service deployment tools allow you to publish your service in a particular application server (Tomcat).
- Axis client tools allow you to convert WSDL into client stubs.
- Axis runtime tools accept incoming SOAP requests and redirect them to the appropriate service.

Developing and Deploying a Service

- Download and install Tomcat and Axis.
- Write a Java implementation
  - Our SubmitJob is a simple example but services can get quite complicated.
  - Compile it into Tomcat's classpath.
- Write a deployment descriptor (WSDD) for your service.
  - Will be used by Axis runtime to direct SOAP calls.
- Use Axis's AdminClient tool to install your WSDD file.
  - The tells the axis servlet to load your class and direct SOAP requests to it.
- That's it.
  - Axis will automatically generate the WSDL for your service.

Sample WSDD

```
<deployment name="Submitjob"
      xmlns="http://xml.apache.org/axis/wsdd/"
      xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
      <service name="Submitjob" provider="java:RPC">
            <parameter name="scope" value="request"/>
            <parameter name="className"
                        value="WebFlowSoap.SJwsImp"/>
            <parameter name="allowedMethods"
                        value="execLocalCommand"/>
      </service>
</deployment>
```

Building a Client with Axis

- Obtain the WSDL file.
- Generate client stubs
  - Stubs look like local objects but really convert method invocations into SOAP calls.
- Write a client application with the stubs
  - Can be a Java GUI, a JSP page, etc.
- Compile everything and run.

Handling Other XML Types

- You can also express other message arguments as XML.
  - Examples: a purchase order, an SVG description of an image, a GML description of a map.
- In practice, these are handled by automatic Bean serializers/deserializers.
  - Castor is an example: http://www.castor.org/
  - XMLBeans is another http://xml.apache.org/xmlbeans/
- These are tools that make it easy to convert between XML and JavaBeans.
- By "JavaBeans" I mean objects that associate simple get/set methods with all data.
- Implementation dependent.

Universal Description, Discovery, and Integration (UDDI)

- Allows businesses to describe their web services in a registry
  - Registry is logically centralized, but physically distributed; operates like DNS
- Allows customers to discover web services
- Allows developers to integrate applications with web services
- Defines an API to achieve all this

UDDI Registry

- Contains information about businesses, services, and service bindings along with some metadata
- Uses white, yellow, and green pages
  - White pages: business name, address, and contact information.
  - Yellow pages: categories based on standard taxonomies
  - Green pages: technical specs and references
- Supports lookups for different user groups

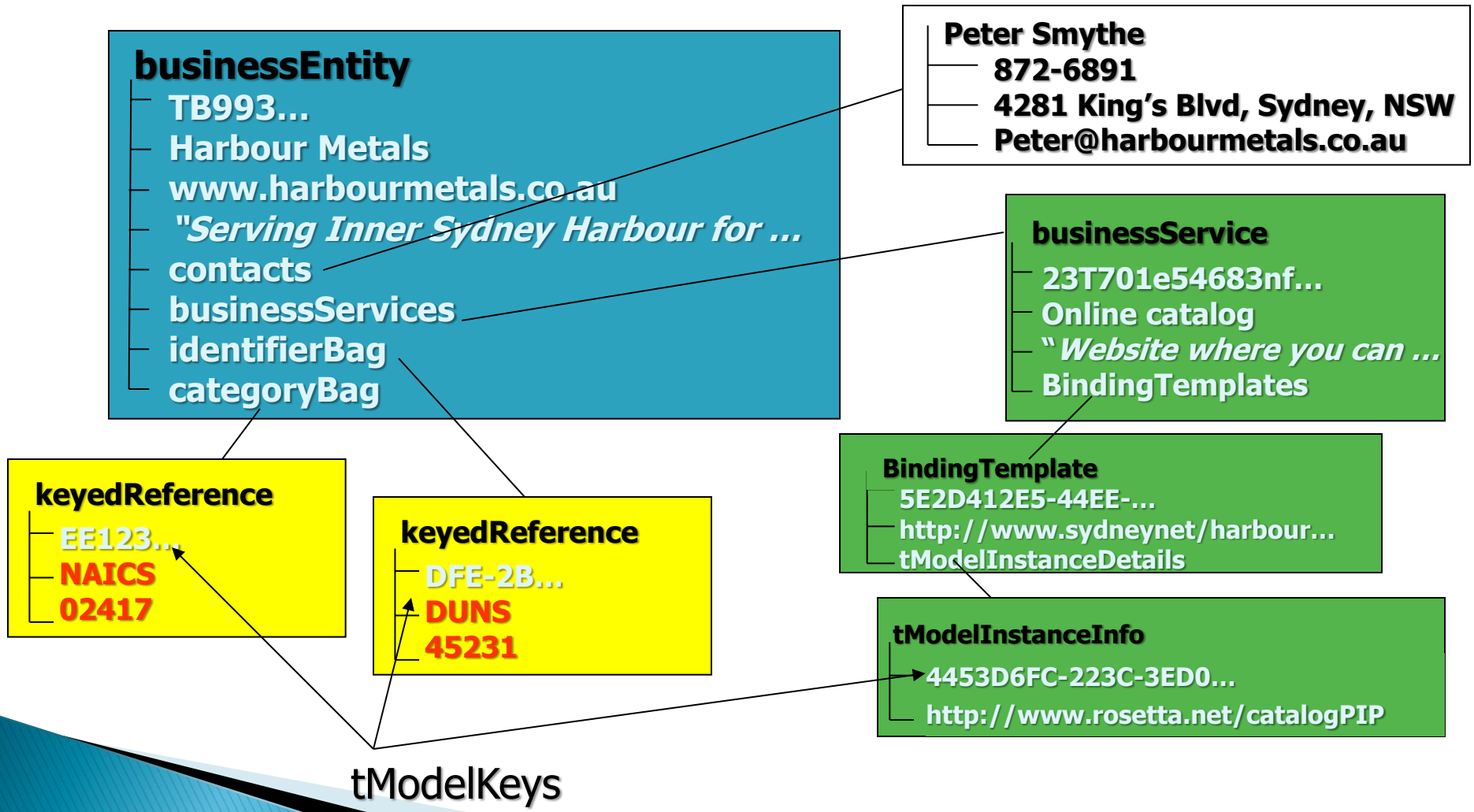
UDDI Registry Elements

- Businesses: Basic information for businesses
- Services: Description and categorization info.
- Binding templates: How to connect to and communicate with services
- tModels: Interface contract and behavior
  - Opaque identifiers registered to allow for more precise, application/domain-specific identification

UDDI Business Type Registration

- XML document
- Created by end-user company (or on their behalf)
- Can have multiple service listings
- Can have multiple taxonomy listings

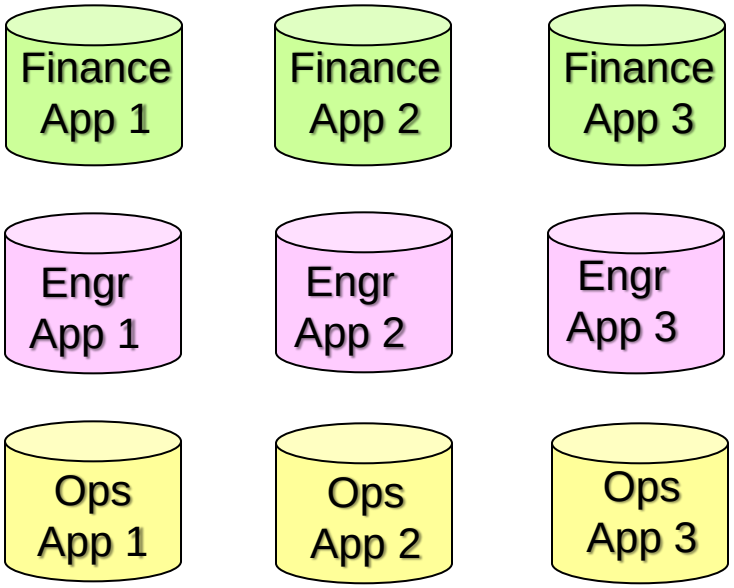
Example UDDI Registration



How is All This Playing Out?

- The players
  - Microsoft, IBM, Developmentor, Ariba
  - Sun Microsystems, Oracle
  - Systems companies aren't as active yet
- Toolsets available
  - Visual Studio.NET, Java, Perl, C++, CGI
- Applications
  - Not many useful applications yet, but expect to lease/rent software components as web services

Corporate Application Architecture



Today – applications exist and are being developed to satisfy specific business process (sub-optimize)

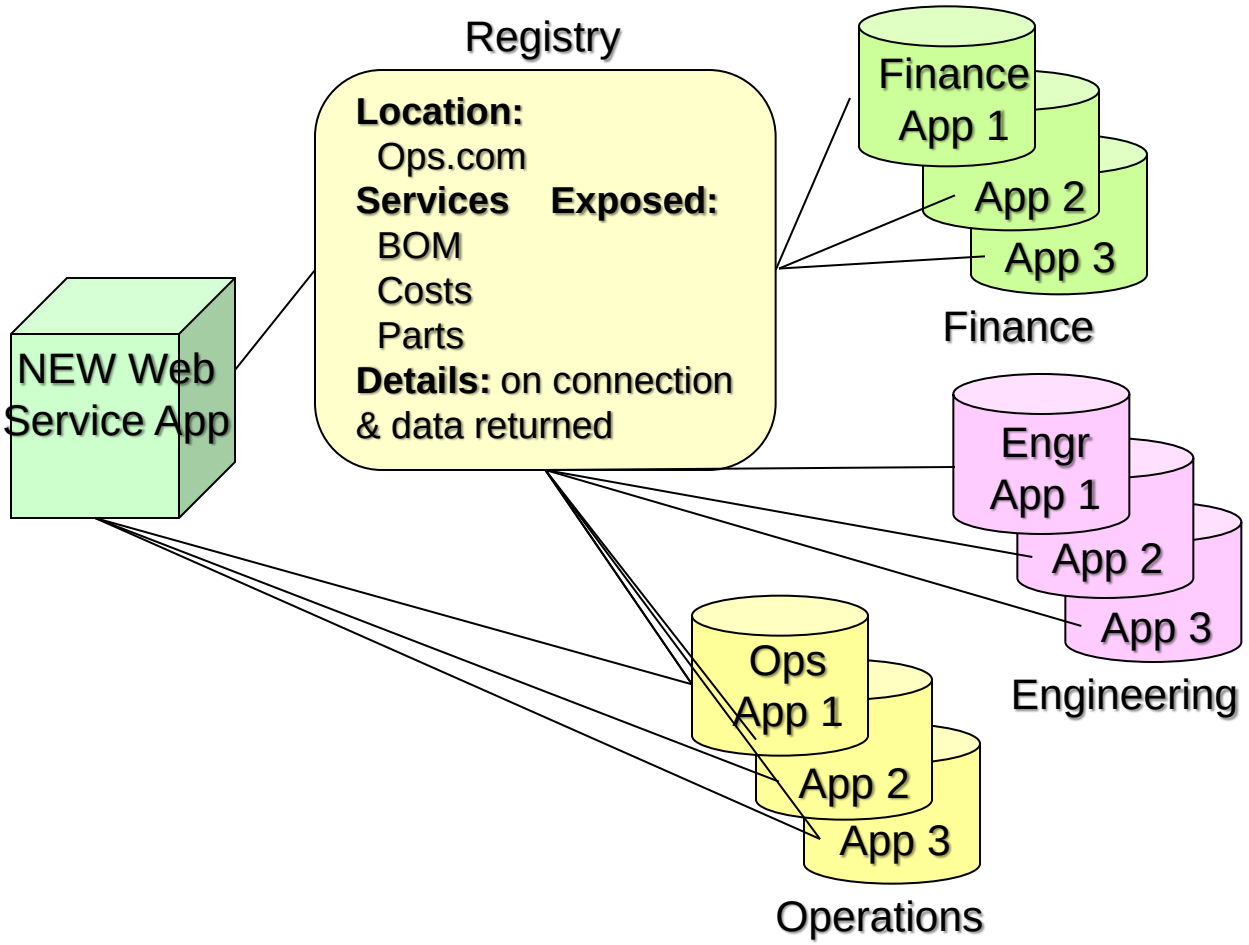
Web Services Enabled Architecture

New opportunities:

Integration - platform independence

Functionality -allow collaboration of data (e.g. from Fin, Engr and Ops)

Business models - revenue generation



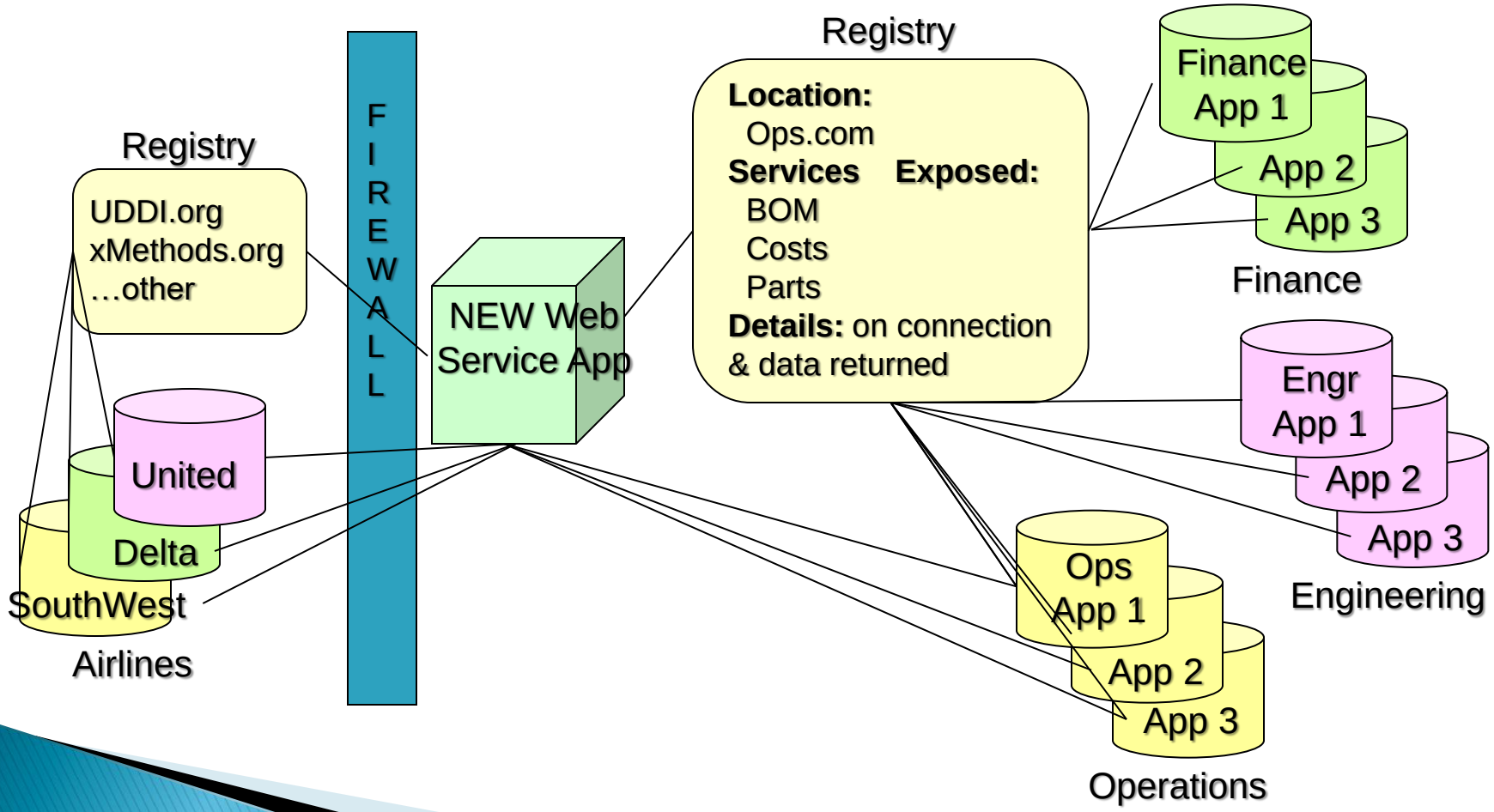
Advantages

- New, integrated functionality using existing applications
- Lowest cost integration solution
- Integration based on standards (vs. proprietary API's or method invocation – DCOM, CORBA, RMI)
- Investment in existing apps extended
- Query backend systems (e.g.build intelligent employee portal)
- Future-proof application development

Registry

- Not required, but provides many advantages
- UDDI.org, xMethods.org, etc.
- Can build for internal use
- Allows for fault-tolerance/fail-over
- May reveal duplicate data and business logic (could lead to application and data elimination)

Web Services Enabled Architecture



Advantages

- Complete supply chain integration
- Collaborative data available to current and new customers
- Enable new business models – leads to new revenue or lower costs
- Provide customer real-time status alerts
- Distribute prices lists to customers and suppliers
- Integrate data for all end users

What Others are Doing

- Key drivers
  - Reduce costs
  - Increase revenue
- Spending
  - Most CIO's/CTO's plan increase spending
- Case studies
  - Microsoft.com/resources/casestudies/

Examples – cont.

- An insurance company is using XML Web Services to pull HR data from many unique systems to create new, value-added collaboration data for employees
- A Chicago manufacturing company using XML Web Services to integrate the entire supply chain