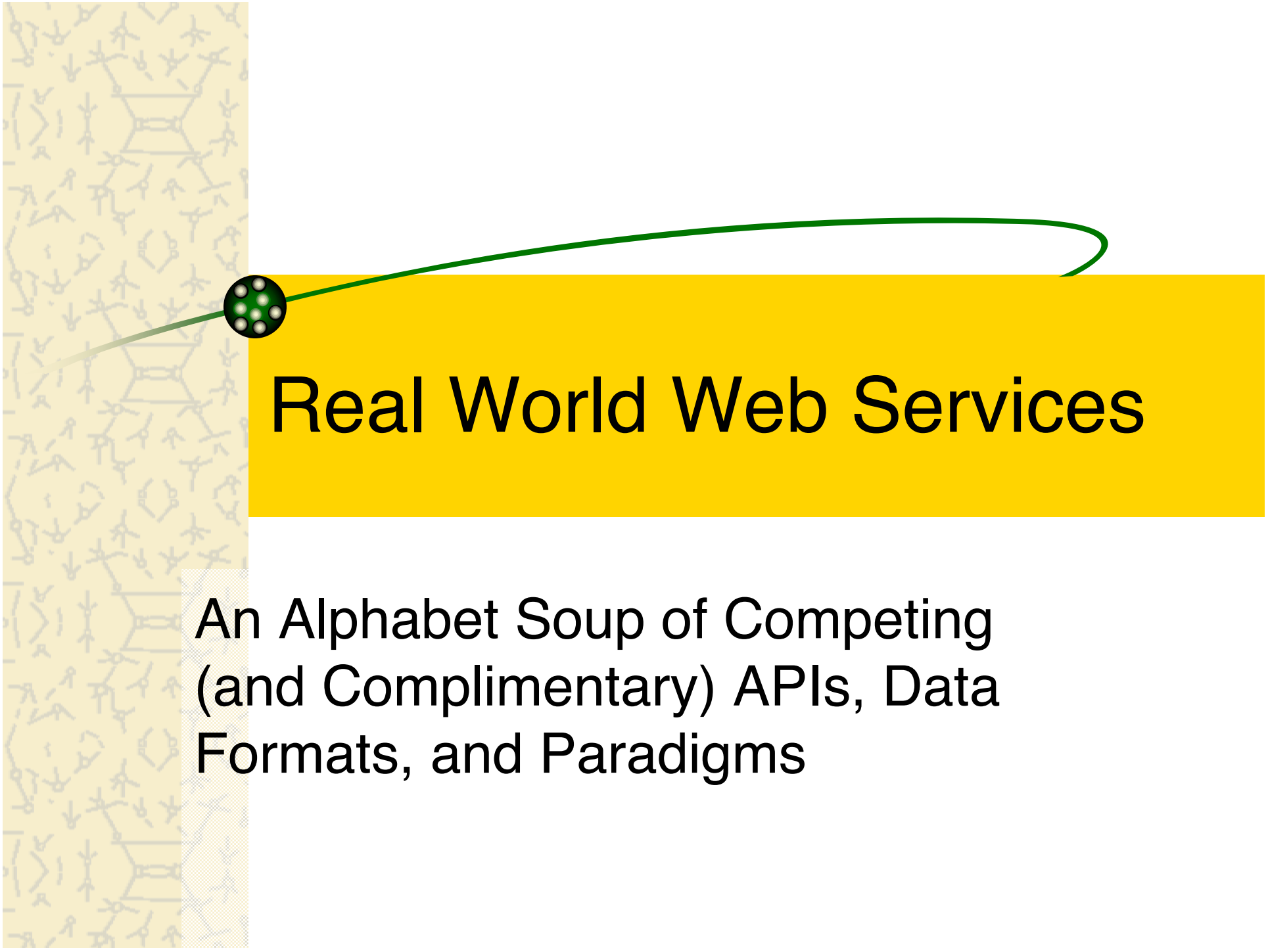
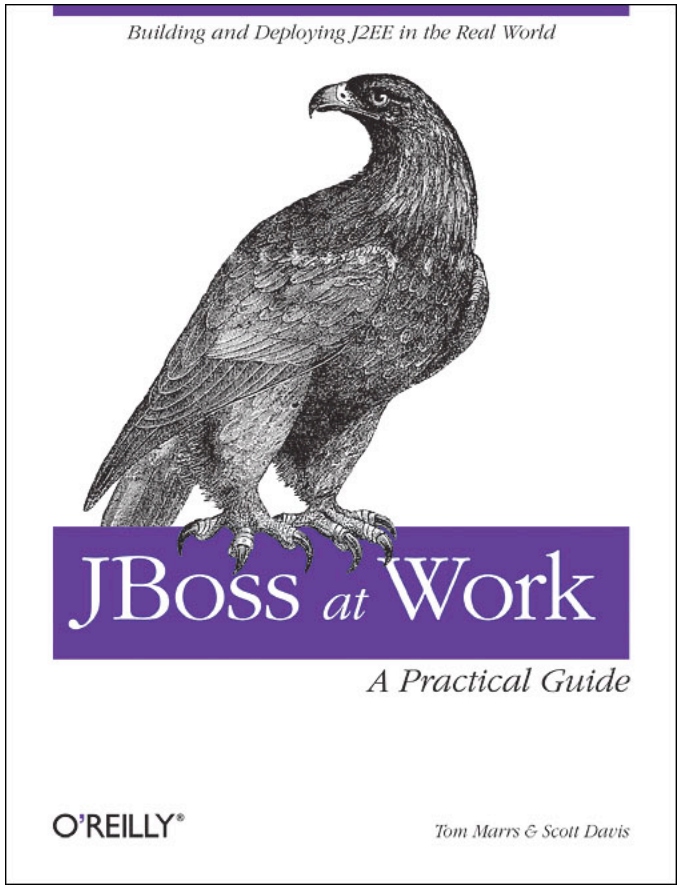
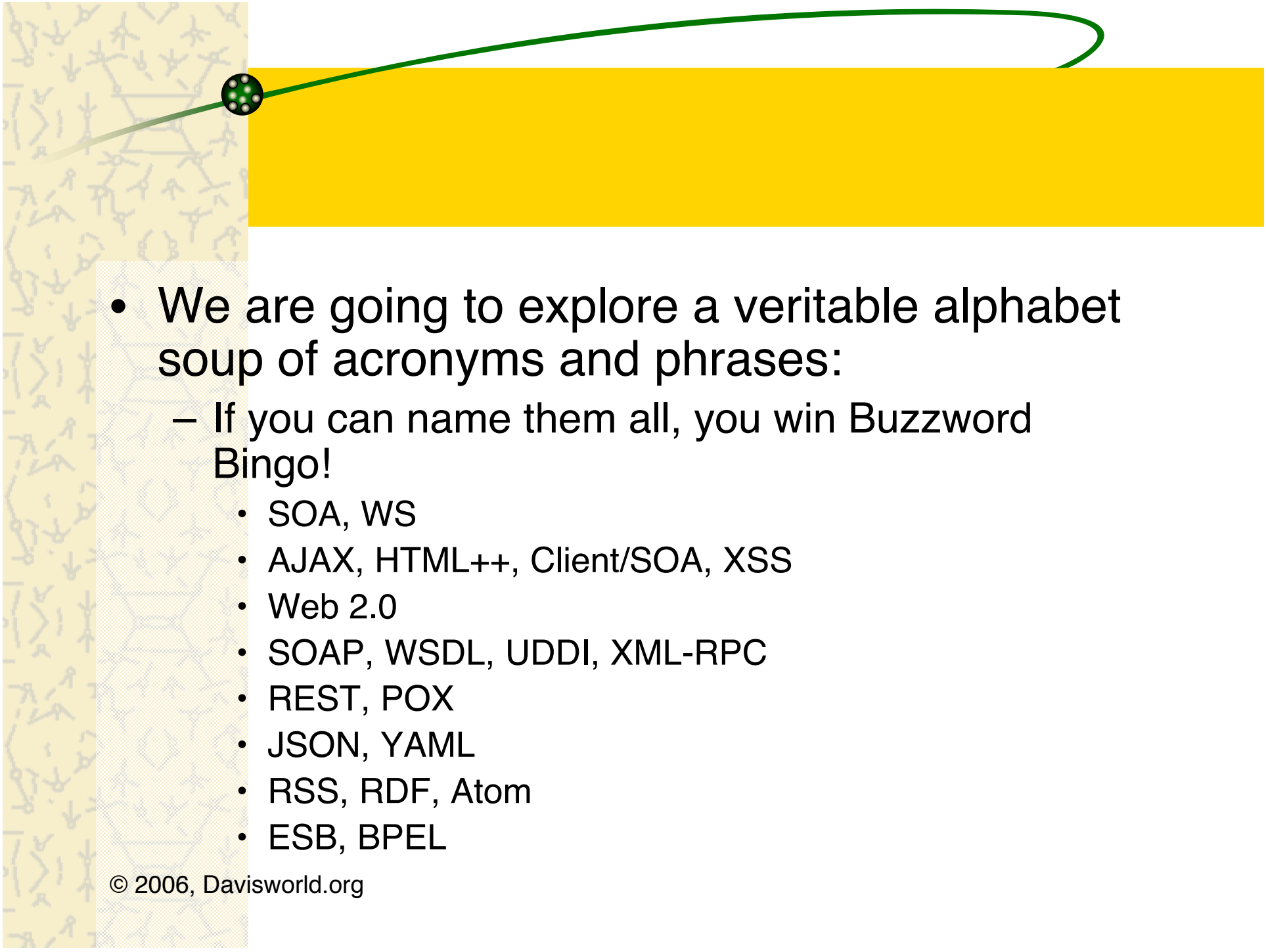

Real World Web Services
An Alphabet Soup of Competing (and Complimentary) APIs, Data Formats, and Paradigms

Introduction

- My name is Scott Davis
  - JBoss At Work (O'Reilly)
  - Google Maps API (Pragmatic Bookshelf)
  - Pragmatic GIS (Pragmatic Bookshelf)
- I'm currently a Senior Software Engineer at OpenLogic





- We are going to explore a veritable alphabet soup of acronyms and phrases:
  - If you can name them all, you win Buzzword Bingo!
    - SOA, WS
    - AJAX, HTML++, Client/SOA, XSS
    - Web 2.0
    - SOAP, WSDL, UDDI, XML-RPC
    - REST, POX
    - JSON, YAML
    - RSS, RDF, Atom
    - ESB, BPEL

- We will dig through these TLAs together and make some sense of them

> The TLA (three-letter acronym or three-letter abbreviation) is the most popular type of abbreviation in technical terminology…
>
> TLA is a three-letter abbreviation itself; the term was almost certainly coined with a certain degree of self-referential humor in mind.
>
> **(Source: http://en.wikipedia.org/wiki/Tla)**

- Although even the definition of “TLA” isn’t without controversy:

> …an acronym is a pronounceable word formed from the initial letter or letters of each of the constituent words, such as NATO or RADAR
>
> …an initialism refers to an abbreviation pronounced as the names of the individual letters, such as TLA or XHTML
>
> **(Source: http://en.wikipedia.org/wiki/Acronym_and_initialism)**

  - Why can’t we all just get along?

- As you will soon find out, this presentation relies heavily on the collective wisdom (and unintentional humor) of wikipedia.org

What is SOA?

- SOA means Service-Oriented Architecture
  - Wikipedia places it in “Category: Ambiguous three-letter acronyms”
    - (How true!)
  - Other possible definitions:
    - Start of Authority (for DNS fans)
    - Sarbanes-Oxley Act (for Enron accounting-scandal fans)
    - Soldiers of Allah (for Islamic rap fans -- no joke!)

- The wikipedia definition does little to disambiguate the matter:

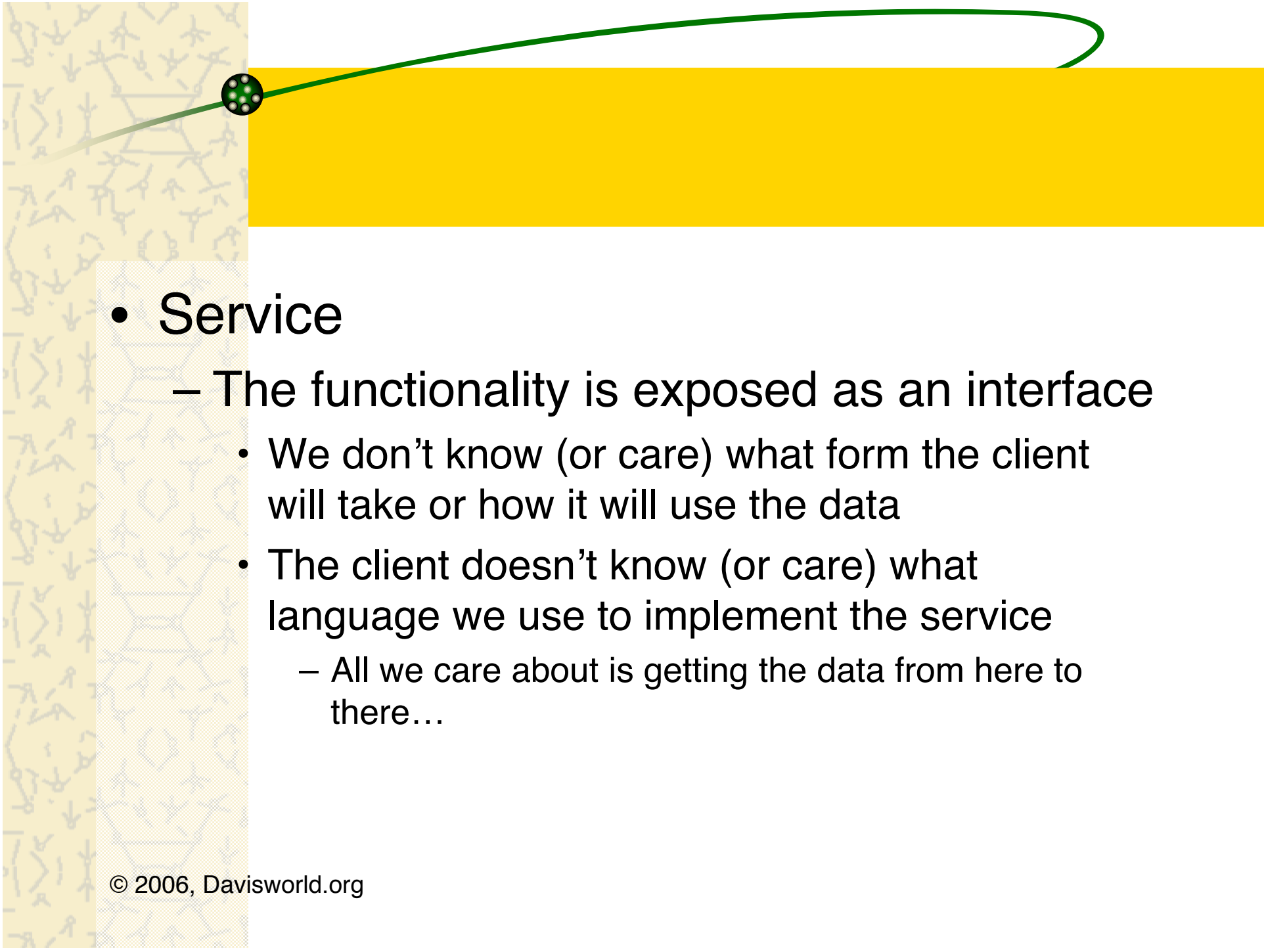
> In computing, the term Service-Oriented Architecture (SOA) expresses a software architectural concept that defines the use of services to support the requirements of software users.
>
> **(Source: http://en.wikipedia.org/wiki/Service-oriented_architecture)**

  - Huh?

- Reading a bit further helps clarify things:

> In a SOA environment, nodes on a network make resources available to other participants in the network as independent services that the participants access in a standardized way.
>
> Unlike traditional point-to-point architectures, SOAs comprise loosely coupled, highly interoperable application services.
>
> **(Source: http://en.wikipedia.org/wiki/Service-oriented_architecture)**

  - OK, now we're on to something here
    - Small Pieces Loosely Joined: A Unified Theory of the Web by David Weinberger
    - Loosely Coupled: The Missing Pieces of Web Services by Doug Kaye



- Service
  - The functionality is exposed as an interface
    - We don't know (or care) what form the client will take or how it will use the data
    - The client doesn't know (or care) what language we use to implement the service
      - All we care about is getting the data from here to there…

- Loosely Coupled
  - Tiny stateless services
    - getStockValue(“IBM”)
    - getCurrentTemp(“Denver”)
      - They can return either a single primitive value or a complex document containing a set of results
- Highly Interoperable
  - Language/Vendor/Platform-neutral

- SOA sounds a bit like WS, doesn't it?

> Most definitions of SOA identify the use of Web service in its implementation. However, one can implement SOA using any service-based technology.
>
> **(Source: http://en.wikipedia.org/wiki/Service-oriented_architecture)**

  - WS is a popular way to implement SOA, but it could be argued that *e-mail* is an SOA as well…
    - Clients can be implemented in any language, any platform
    - Beyond simple person-to-person e-mail messages, SMTP can be used as a "messaging platform"
      - » Mailing lists (subscribe/unsubscribe via keywords)
      - » Continuous-build failure notifications
      - » Cell phone/parking meter interface

What are Web Services?

- The WS definition is as vague as the SOA definition

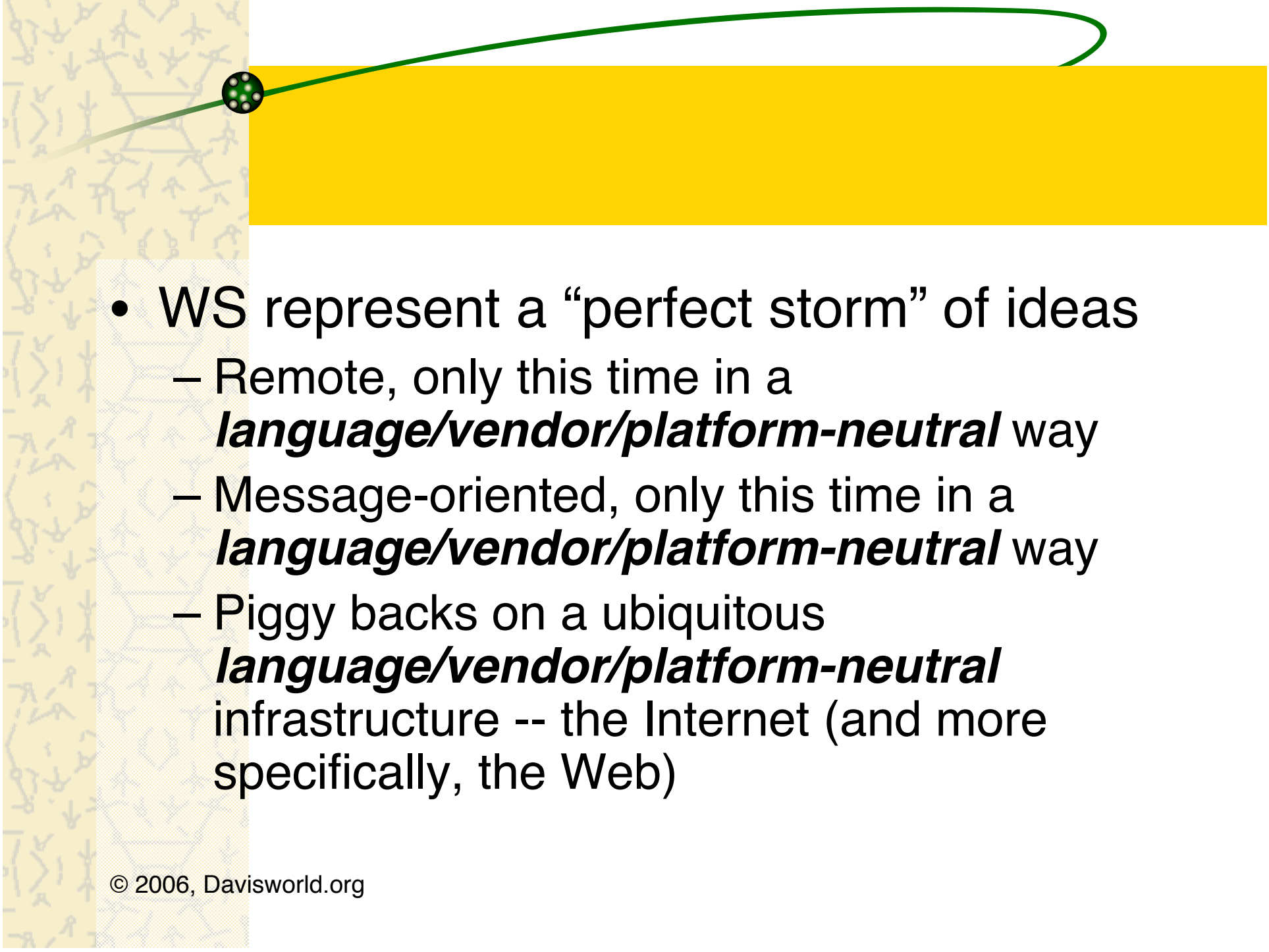
> According to the W3C a Web service is a software system designed to support interoperable machine-to-machine interaction over a network.
>
> **(Source: http://en.wikipedia.org/wiki/Web_service)**

- Reading farther yields some more meat:

> It has an interface that is described in a machine-processable format…
>
> Other systems interact with the Web service in a manner prescribed by its interface using messages…
>
> These messages are typically conveyed using HTTP, and normally comprise XML in conjunction with other Web-related standards.
>
> Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet…
>
> This interoperability (for example, between Java and Python, or Microsoft Windows and Linux applications) is due to the use of open standards…
>
> **(Source: http://en.wikipedia.org/wiki/Web_service)**

- Here is the simplest possible definition of a Web Service:
  - "Make a request on port 80, get XML back"
    - Of course, the format of the request, the port it is made on, and format of the returned data can all vary… (grimace)

- There is one unambiguous thing about WS: It is “Remote” and “Message-Oriented”
  - What makes it different than other remoting architectures?
    - RMI, DCOM, CORBA
  - What makes it different than other message-oriented architectures?
    - JMS, MSMQ



- WS represent a “perfect storm” of ideas
  - Remote, only this time in a ***language/vendor/platform-neutral*** way
  - Message-oriented, only this time in a ***language/vendor/platform-neutral*** way
  - Piggy backs on a ubiquitous ***language/vendor/platform-neutral*** infrastructure -- the Internet (and more specifically, the Web)

- Web Services leverage the ubiquity of the Web to deliver services
  - Every company has a web server in place
  - Every company has port 80 open
- Q: Why does every town have taxi and bus service, but only a few have subways, trolleys, monorails, gondolas, etc?
  - A: Existing Infrastructure

- What is the difference between “the web” and “web services”?
  - The Web is all about Presentation (HTML)

```
<h1>Honda Accord</h1>
<b>Color:</b> Black<br>
<b>Sunroof:</b> Yes<br>
```

  - WS is all about Data (XML)
    - In MVC terms:
      - The Web is the View
      - WS is the Model

```
<car type=“Honda Accord”>
  <color>Black</color>
  <sunroof>Yes</sunroof>
</car>
```

- So where is the Controller (i.e. where is the integration of Model and View)?
  - Traditionally, it is on the web server as a part of the UI framework (Struts/WebWork, Rails, Tapestry, etc.)
  - Increasingly, it is in the browser courtesy of AJAX

What is AJAX?

Asynchronous JavaScript And XML, or its acronym Ajax (Pronounced A-JAX), is a Web development technique for creating interactive web applications.

The intent is to shift a great deal of interaction to the Web surfer's computer, exchanging data with the server behind the scenes, so that the entire Web page does not have to be reloaded each time the user makes a change.

(Source: http://en.wikipedia.org/wiki/Ajax_%28programming%29)

- It is the combination of browser-based:
  - XHTML
  - JavaScript
  - DOM
  - CSS
  - XMLHttpRequest (XHR)

What is Client/SOA?

- Some people dismiss AJAX as simply "HTML++" and are trying to popularize the concept of "Client/SOA":

> This architecture is considered by some to be the natural endpoint of evolution of the web from a primarily server-centric architecture, which includes what some have termed AJAX or HTML++, to one in which the browser is an autonomous component which can access and aggregate data from one or more user-selected web services, i.e. an architecture fully realized in a true client-side portaling application.
>
> **(Source: http://en.wikipedia.org/wiki/Client/SOA)**

  - In Client/SOA, the browser is the new JVM

What is XSS?

- There is only one hitch in the road to true Client/SOA Nirvana: XSS

> Cross site scripting (XSS) is a type of computer security vulnerability typically found in web applications which can be used by an attacker to compromise the "same origin" policy of client-side scripting languages.
>
> **(Source: http://en.wikipedia.org/wiki/XSS)**

  - All web browsers restrict AJAX/XHR requests to the original domain of the web page
    - In other words, http://www.davisworld.org/test.html cannot make a WS call to http://api.google.com/search/beta2

- XSS Workarounds:
  - Digitally sign your JavaScript
    - No cross-browser solution exists
    - http://www.mozilla.org/projects/security/components/signed-scripts.html
  - Create a proxy in your domain that can make calls on the client's behalf
    - XSS limits what calls ***the browser*** can make, not a server-side component

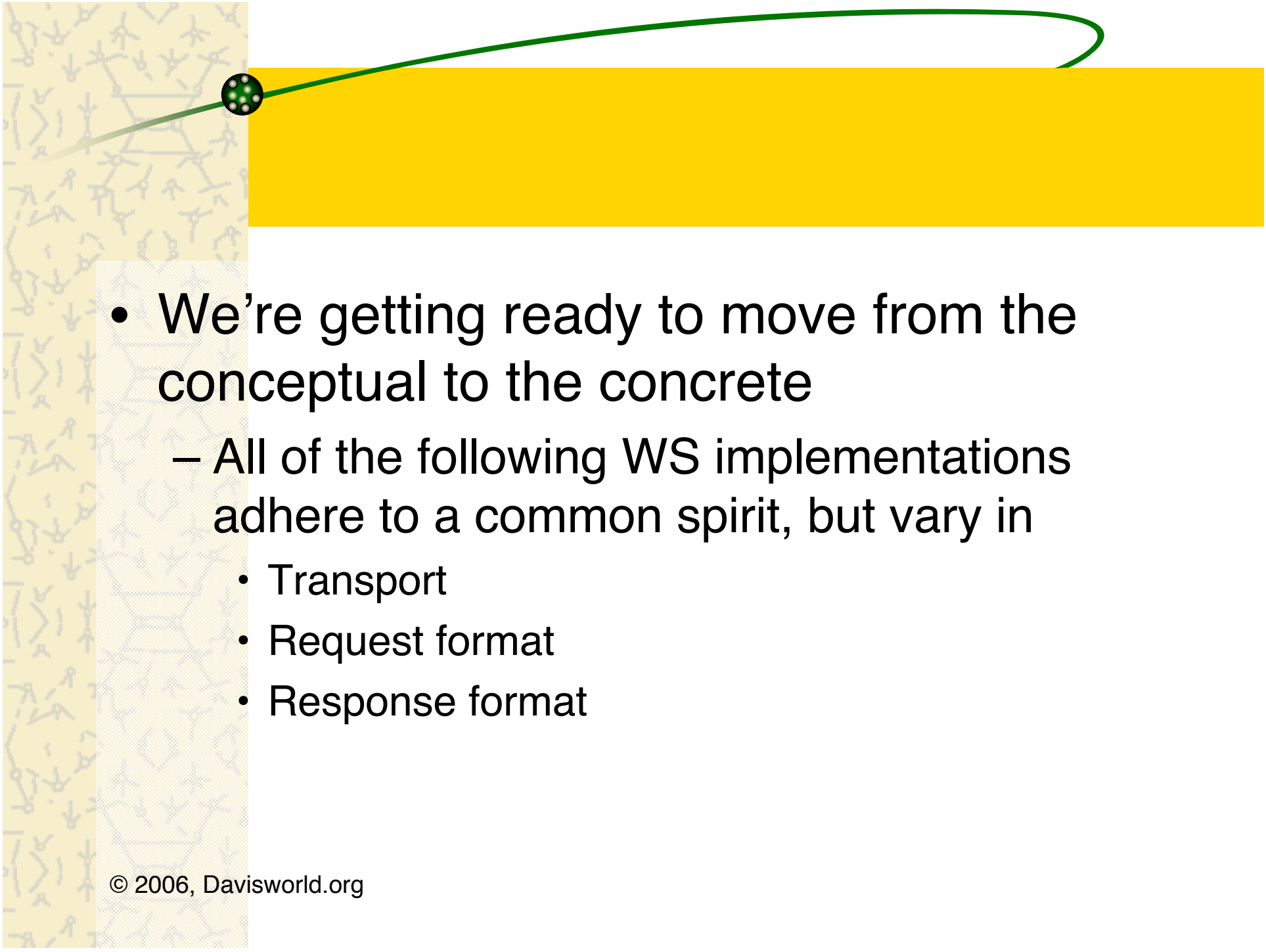
- Creating a proxy is by far the easier of the two:
  - Request: http://www.davisworld.org/city-ws-proxy.jsp?city=Denver
  - Proxy:

```
1  <%@ taglib prefix="c" uri="http://java.sun.com/jstl/core_rt" %>
2  <% response.setContentType("text/xml"); %>
3
4  <c:import url="http://brainoff.com/geocoder/rest">
5    <c:param name="city" value="${param.city}" />
6  </c:import>
```

- All of these ideas lead to what the pundits are calling "Web 2.0"
  - Tim O'Reilly (Founder and CEO of O'Reilly Media) calls it "Web 2.0".
  - Jonathon Schwarz (President and COO of Sun Microsystems) calls it "The Participation Age".

What is Web 2.0?

- While there are no hard and fast definitions, the same basic ideas come up over and over:
  - Favor interactive web applications over static/passive web pages (AJAX)
  - Favor data/web services in addition to fixed presentation/HTML (WS)
  - Allow/encourage your content to be integrated into other applications (Mash-ups -- AJAX+WS)



- We’re getting ready to move from the conceptual to the concrete
  - All of the following WS implementations adhere to a common spirit, but vary in
    - Transport
    - Request format
    - Response format

What is SOAP?

- SOAP is one of the most common WS implementations

> SOAP is a protocol for exchanging XML-based messages over a computer network, normally using HTTP. SOAP forms the foundation layer of the web services stack, providing a basic messaging framework that more abstract layers can build on. SOAP facilitates the Service-Oriented architectural pattern.
>
> The name "SOAP" was originally an acronym for Simple Object Access Protocol, but the full name was dropped in Version 1.2 of the SOAP specification…
>
> **(Source: http://en.wikipedia.org/wiki/SOAP)**

  - XML-RPC was a precursor to SOAP

SOAP Specifics

- Transport:
  - Commonly HTTP POST, but can also be SMTP, JMS, etc.
- Request format:
  - XML (SOAP Envelope, Header, Body)
- Response Format
  - XML (SOAP Envelope, Header, Body)

What is WSDL?

- SOAP is the message format
- WSDL describes the WS interface

The Web Services Description Language (WSDL) is an XML format published for describing Web services.

WSDL describes the public interface to the web service. This is an XML-based service description on how to communicate using the web service; namely, the protocol bindings and message formats required to interact with the web services listed in its directory. The supported operations and messages are described abstractly, and then bound to a concrete network protocol and message format.

(Source: http://en.wikipedia.org/wiki/WSDL)

What is UDDI?

- SOAP and WSDL are common -- UDDI, not so much…

> UDDI is an acronym for Universal Description, Discovery, and Integration – A platform-independent, XML-based registry for businesses worldwide to list themselves on the Internet.
>
> UDDI is nominally one of the core Web Services standards. It is designed to be interrogated by SOAP messages and to provide access to WSDL documents describing the protocol bindings and message formats required to interact with the web services listed in its directory.
>
> **(Source: http://en.wikipedia.org/wiki/UDDI)**

  - In reality, you will most likely download the WSDL directly from the organization, not a common UDDI directory

- To see SOAP in action, let's use the free Google WS API:

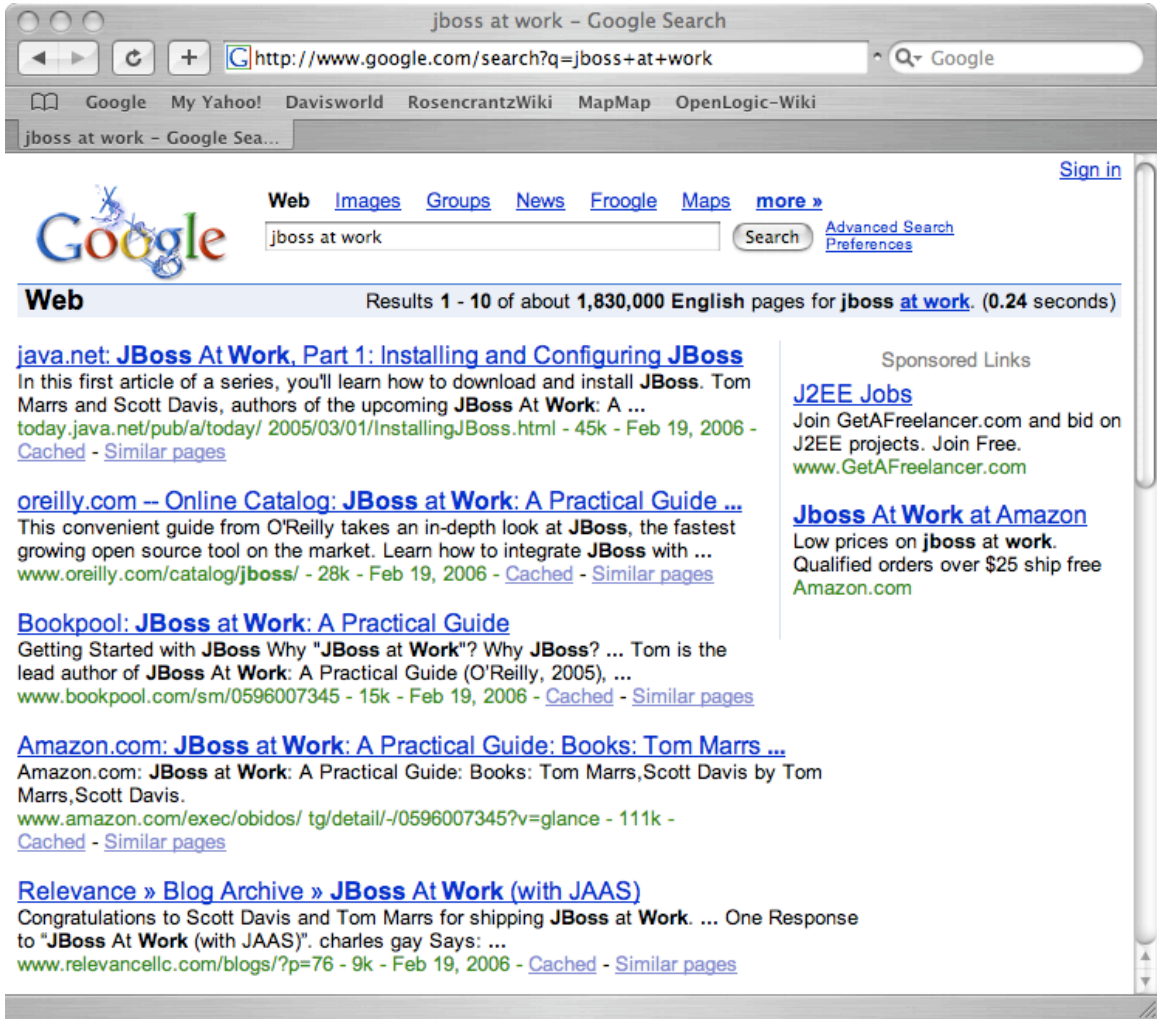
With the Google Web APIs service, software developers can query billions of web pages directly from their own computer programs. Google uses the SOAP and WSDL standards so a developer can program in his or her favorite environment - such as Java, Perl, or Visual Studio .NET.

To start writing programs using Google Web APIs:

1. Download the developer's kit
2. Create a Google Account
3. Write your program using your license key

(Source: http://www.google.com/apis/)

- What we're trying to accomplish is performing this query programmat-ically
  - XML results instead of HTML



- The WSDL (in the developer's toolkit we downloaded) describes the WS
  - The URL

```
1  <!-- Endpoint for Google Web APIs -->
2  <service name="GoogleSearchService">
3    <port name="GoogleSearchPort" binding="typens:GoogleSearchBinding">
4      <soap:address location="http://api.google.com/search/beta2"/>
5    </port>
6  </service>
```

- The available method calls:

```
10  <!-- Port for Google Web APIs, "GoogleSearch" -->
11  <portType name="GoogleSearchPort">
12
13    <operation name="doGetCachedPage">
14      <input message="typens:doGetCachedPage"/>
15      <output message="typens:doGetCachedPageResponse"/>
16    </operation>
17
18    <operation name="doSpellingSuggestion">
19      <input message="typens:doSpellingSuggestion"/>
20      <output message="typens:doSpellingSuggestionResponse"/>
21    </operation>
22
23    <operation name="doGoogleSearch">
24      <input message="typens:doGoogleSearch"/>
25      <output message="typens:doGoogleSearchResponse"/>
26    </operation>
27  </portType>
```

- The binding:

```
34  <!-- Binding for Google Web APIs - RPC, SOAP over HTTP -->
35  <binding name="GoogleSearchBinding" type="typens:GoogleSearchPort">
36    <soap:binding style="rpc"
37                  transport="http://schemas.xmlsoap.org/soap/http"/>
38    <operation name="doGoogleSearch">
39      <soap:operation soapAction="urn:GoogleSearchAction"/>
40      <input>
41        <soap:body use="encoded"
42                   namespace="urn:GoogleSearch"
43                   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
44      </input>
45      <output>
46        <soap:body use="encoded"
47                   namespace="urn:GoogleSearch"
48                   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
49      </output>
50    </operation>
51  </binding>
```

- The request arguments:

```
58  <message name="doGoogleSearch">
59    <part name="key"            type="xsd:string"/>
60    <part name="q"              type="xsd:string"/>
61    <part name="start"          type="xsd:int"/>
62    <part name="maxResults"     type="xsd:int"/>
63    <part name="filter"         type="xsd:boolean"/>
64    <part name="restrict"       type="xsd:string"/>
65    <part name="safeSearch"     type="xsd:boolean"/>
66    <part name="lr"             type="xsd:string"/>
67    <part name="ie"             type="xsd:string"/>
68    <part name="oe"             type="xsd:string"/>
69  </message>
```

- The response format:

```
75      <xsd:complexType name="ResultElement">
76        <xsd:all>
77          <xsd:element name="summary" type="xsd:string"/>
78          <xsd:element name="URL" type="xsd:string"/>
79          <xsd:element name="snippet" type="xsd:string"/>
80          <xsd:element name="title" type="xsd:string"/>
81          <xsd:element name="cachedSize" type="xsd:string"/>
82          <xsd:element name="relatedInformationPresent" type="xsd:boolean"/>
83          <xsd:element name="hostName" type="xsd:string"/>
84          <xsd:element name="directoryCategory" type="typens:DirectoryCategory"/>
85          <xsd:element name="directoryTitle" type="xsd:string"/>
86        </xsd:all>
87      </xsd:complexType>
```

- Here is the SOAP Request:

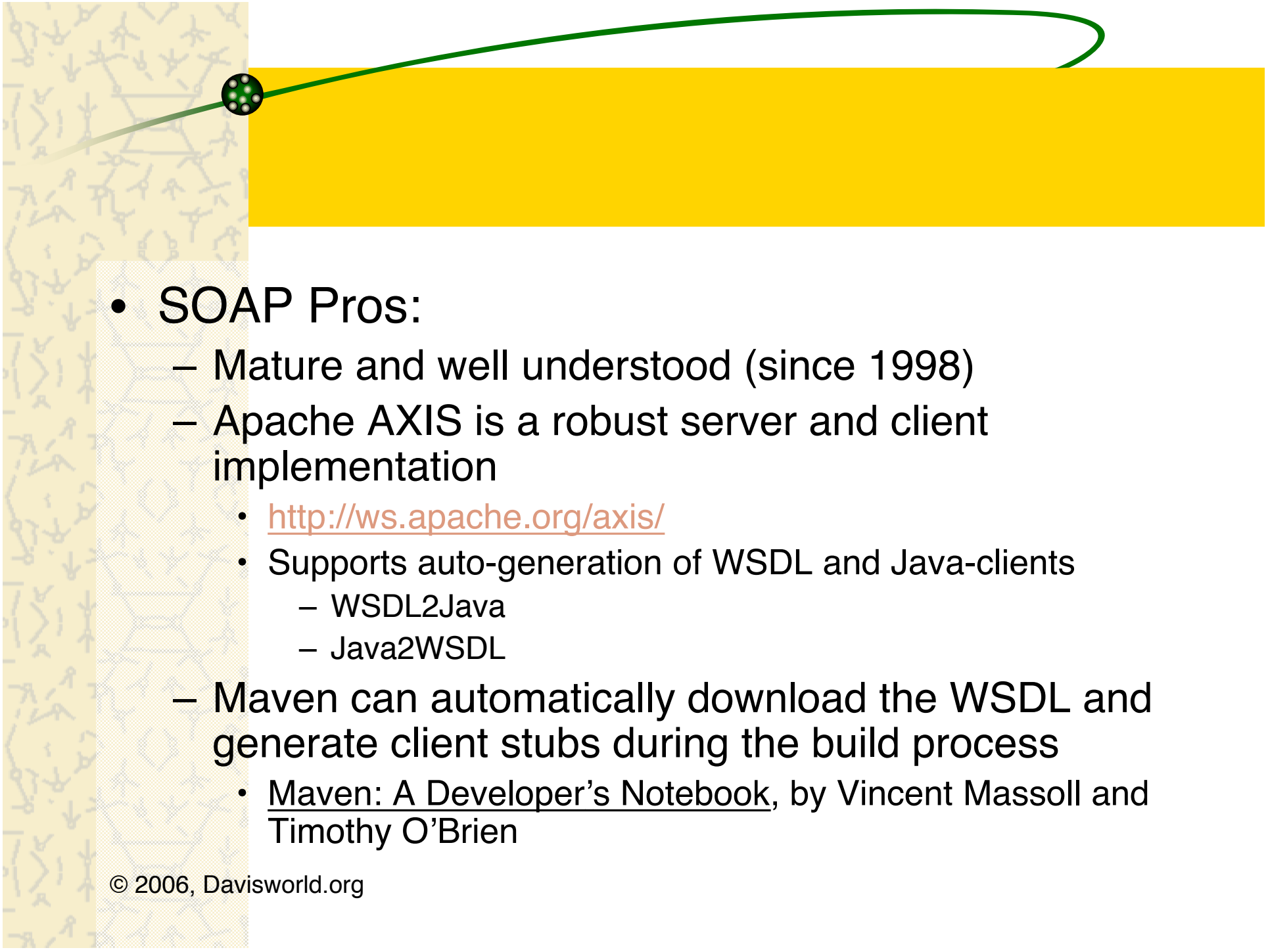
```
<?xml version='1.0' encoding='UTF-8'?>

<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:doGoogleSearch
        xmlns:ns1="urn:GoogleSearch"
        SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <key xsi:type="xsd:string">00000000000000000000000000000000</key>
      <q xsi:type="xsd:string">jboss at work</q>
      <start xsi:type="xsd:int">0</start>
      <maxResults xsi:type="xsd:int">10</maxResults>
      <filter xsi:type="xsd:boolean">true</filter>
      <restrict xsi:type="xsd:string"></restrict>
      <safeSearch xsi:type="xsd:boolean">false</safeSearch>
      <lr xsi:type="xsd:string"></lr>
      <ie xsi:type="xsd:string">latin1</ie>
      <oe xsi:type="xsd:string">latin1</oe>
    </ns1:doGoogleSearch>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- And here is the SOAP Response:

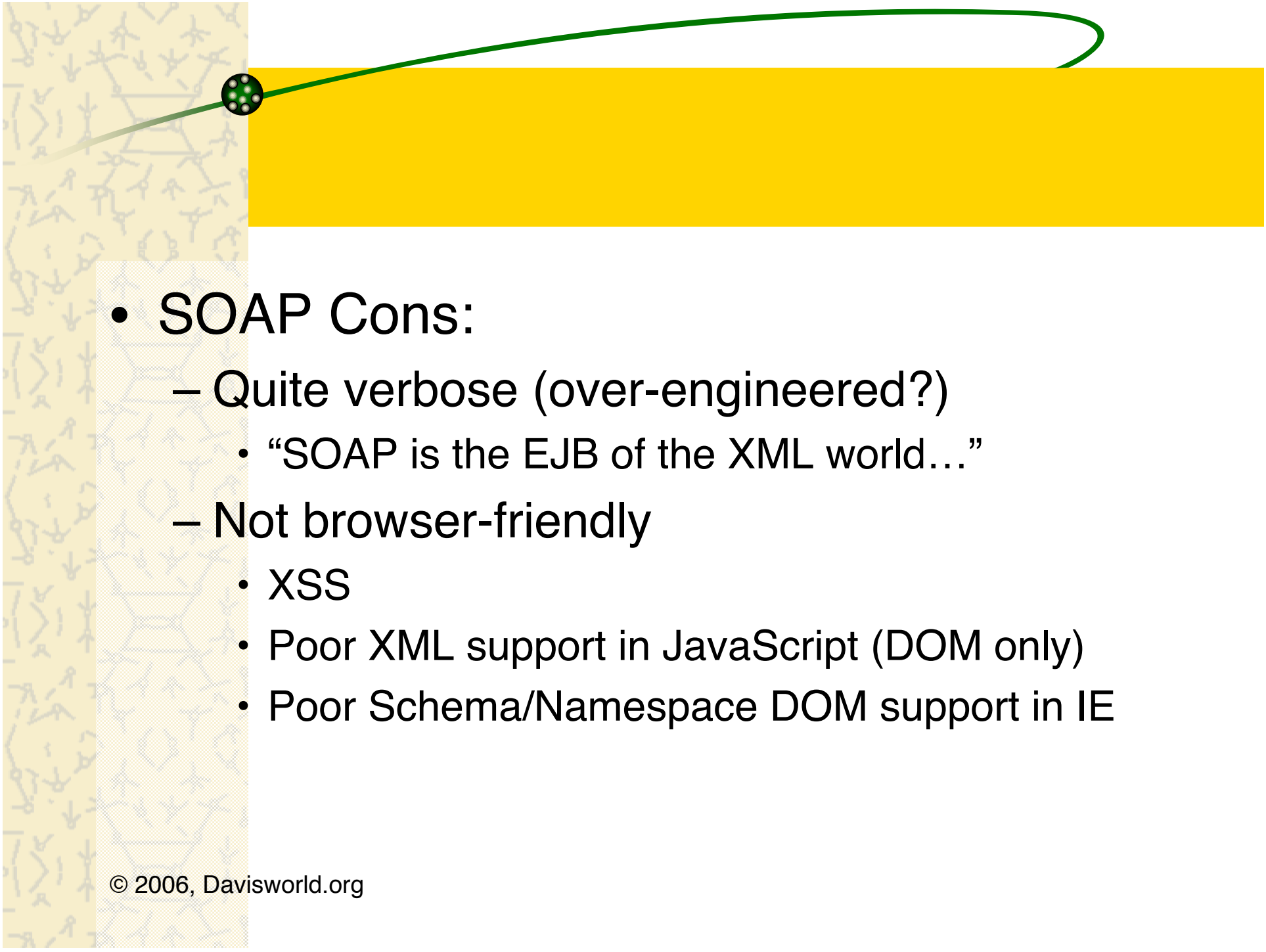
```
<?xml version='1.0' encoding='UTF-8'?>

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance" xmlns:xsd="http://www.w3.org/1999/
XMLSchema">
  <SOAP-ENV:Body>
    <ns1:doGoogleSearchResponse xmlns:ns1="urn:GoogleSearch" SOAP-ENV:encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/">
      <return xsi:type="ns1:GoogleSearchResult">
        <documentFiltering xsi:type="xsd:boolean">false</documentFiltering>
        <estimatedTotalResultsCount xsi:type="xsd:int">3</estimatedTotalResultsCount>
        <directoryCategories xmlns:ns2="http://schemas.xmlsoap.org/soap/encoding/"
xsi:type="ns2:Array" ns2:arrayType="ns1:DirectoryCategory[0]"></directoryCategories>
        <searchTime xsi:type="xsd:double">0.194871</searchTime>
        <resultElements xmlns:ns3="http://schemas.xmlsoap.org/soap/encoding/"
xsi:type="ns3:Array" ns3:arrayType="ns1:ResultElement[3]">
```

- …and the first response element:

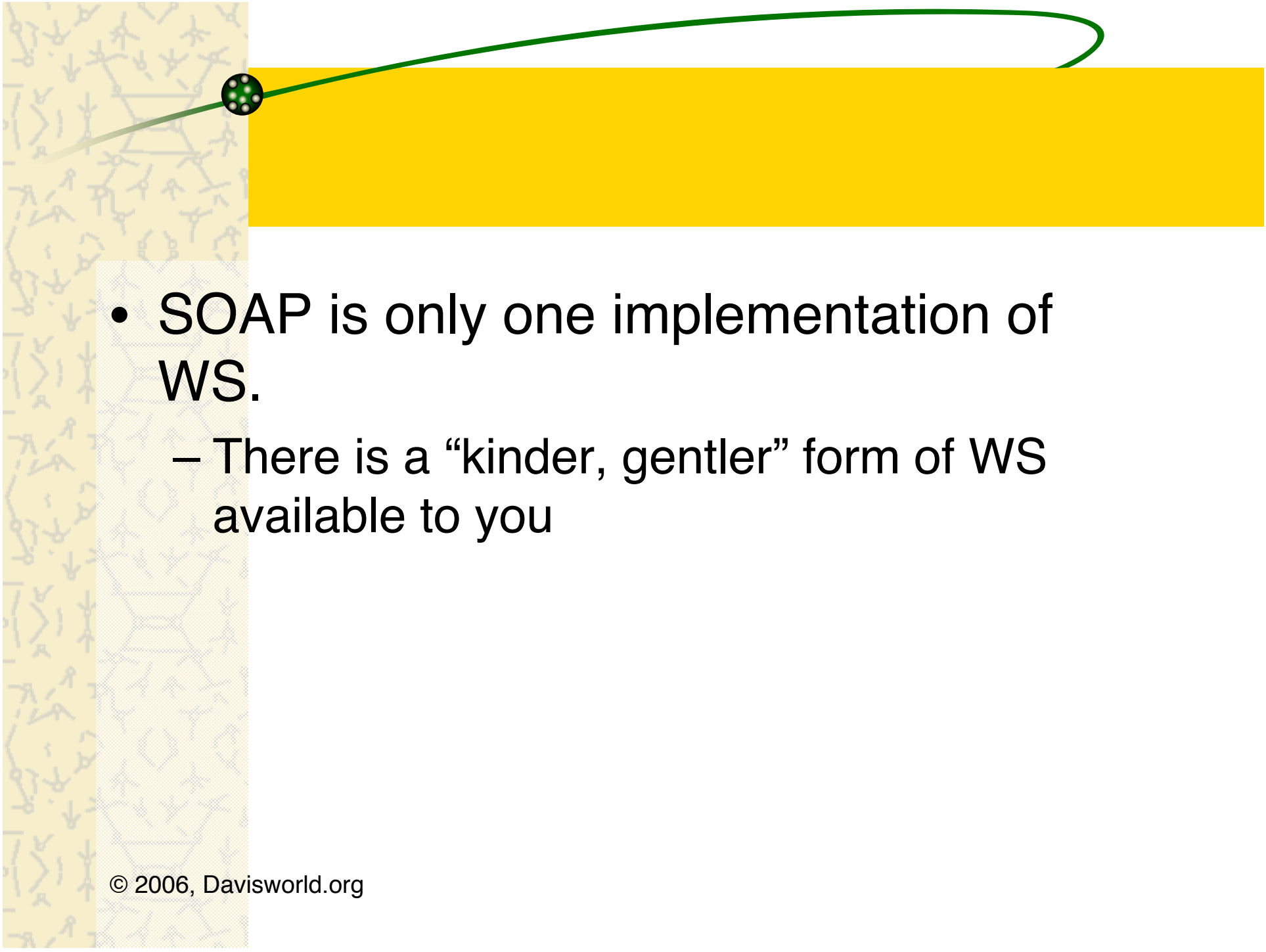
```
12          <item xsi:type="ns1:ResultElement">
13            <cachedSize xsi:type="xsd:string">12k</cachedSize>
14            <hostName xsi:type="xsd:string"></hostName>
15            <snippet xsi:type="xsd:string">In this first article of a series, you'll learn how
 . to download and install JBoss. Tom Marrs and Scott Davis, authors of the upcoming JBoss At
 . Work: A ...</snippet>
16            <directoryCategory xsi:type="ns1:DirectoryCategory">
17              <specialEncoding xsi:type="xsd:string"></specialEncoding>
18              <fullViewableName xsi:type="xsd:string"></fullViewableName>
19            </directoryCategory>
20            <relatedInformationPresent xsi:type="xsd:boolean">true</relatedInformationPresent>
21            <directoryTitle xsi:type="xsd:string"></directoryTitle>
22            <summary xsi:type="xsd:string"></summary>
23            <URL xsi:type="xsd:string">http://today.java.net/pub/a/today/2005/03/01/
 . InstallingJBoss.html</URL>
24            <title xsi:type="xsd:string">&lt;b&gt;JBoss At Work&lt;/b&gt;</title>
25          </item>
```



- SOAP Pros:
  - Mature and well understood (since 1998)
  - Apache AXIS is a robust server and client implementation
    - http://ws.apache.org/axis/
    - Supports auto-generation of WSDL and Java-clients
      - WSDL2Java
      - Java2WSDL
  - Maven can automatically download the WSDL and generate client stubs during the build process
    - Maven: A Developer's Notebook, by Vincent Massoll and Timothy O'Brien



- SOAP Cons:
  - Quite verbose (over-engineered?)
    - "SOAP is the EJB of the XML world…"
  - Not browser-friendly
    - XSS
    - Poor XML support in JavaScript (DOM only)
    - Poor Schema/Namespace DOM support in IE



- SOAP is only one implementation of WS.
  - There is a “kinder, gentler” form of WS available to you

What is REST?

- The other white meat:

> Representational State Transfer (REST) is a software architectural style for distributed hypermedia systems like the world wide web. The term originated in a 2000 doctoral dissertation about the web written by Roy Fielding, one of the principal authors of the HTTP protocol specification, and has quickly passed into widespread use in the networking community.
>
> Systems that follow Fielding's REST principles are often referred to as RESTful; REST's most zealous advocates call themselves RESTafarians.
>
> **(Source: http://en.wikipedia.org/wiki/Representational_State_Transfer)**

- What does that have to do with WS?

> While REST originally referred to a collection of architectural principles, people now often use the term in a looser sense to describe any simple web-based interface that uses XML and HTTP without the extra abstractions of MEP-based approaches like the web services SOAP protocol.
>
> **(Source: http://en.wikipedia.org/wiki/Representational_State_Transfer)**

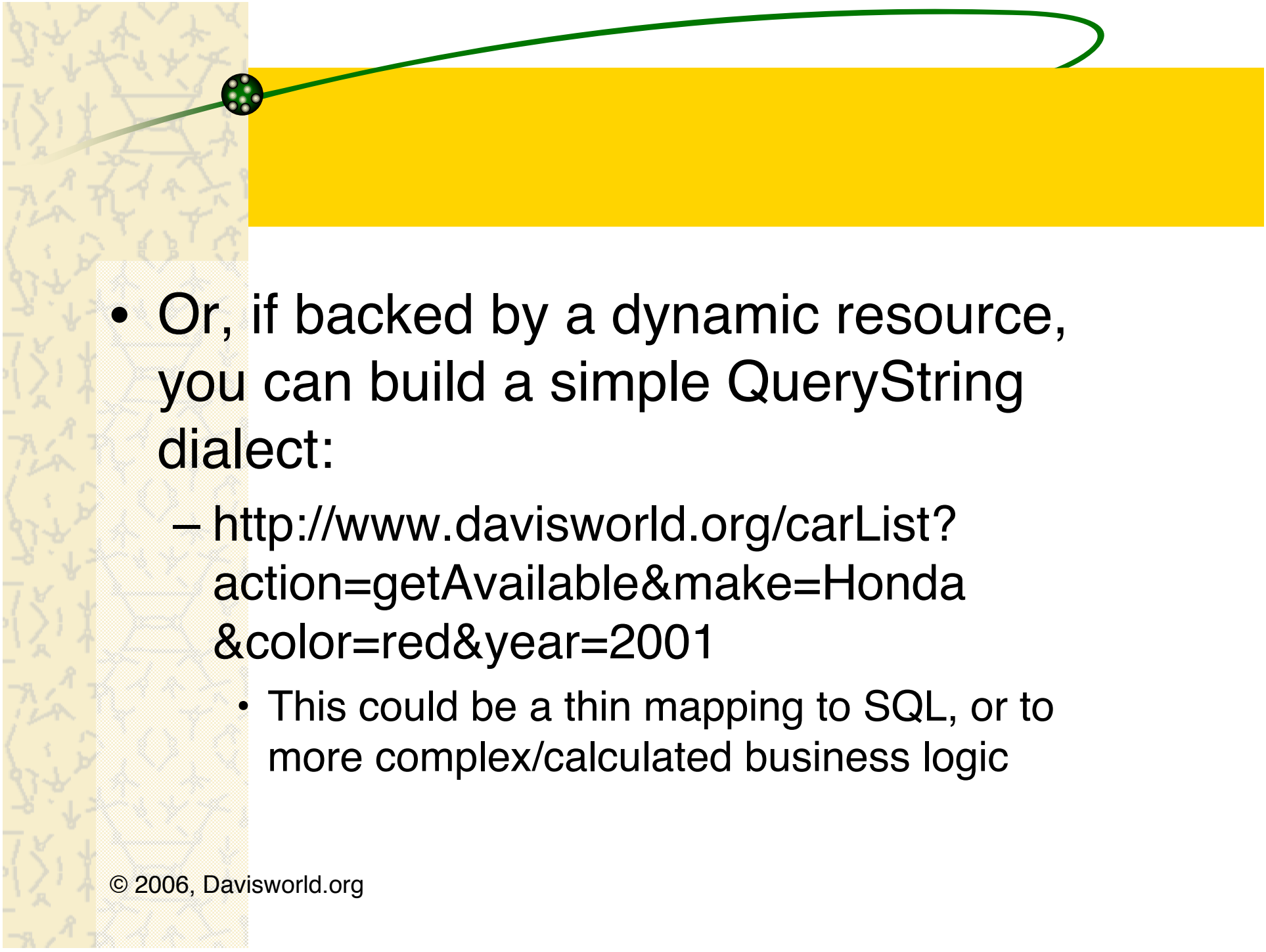
  - MEP == “Message Exchange Pattern”

- There are two “types” of RESTful WS out there:
  - Pure REST (following Fielding’s principles)
  - Popular REST (essentially !SOAP)
    - (NOTE: These definitions are mine, not Wikipedia’s)

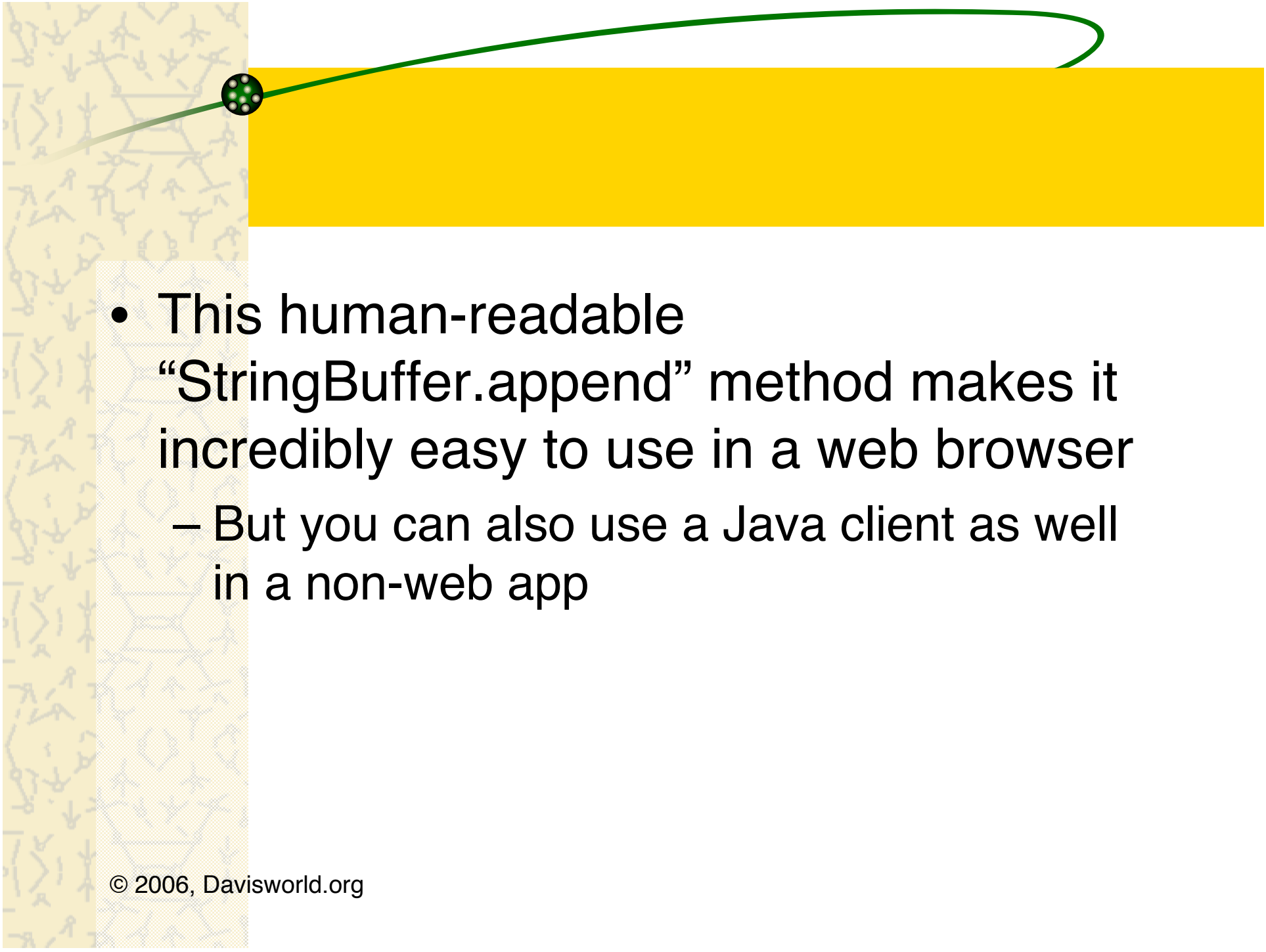
- Popular REST is a reaction to the complexity of SOAP
  - Transport: HTTP GET
  - Request format: URL + QueryString
  - Response format: XML
    - Sometimes it is informally called POX (Plain Old XML) in homage to POJOs (Plain Old Java Objects).

- You could implement a simple RESTful POX service using nothing but static JSPs:
  - http://www.davisworld.org/carList.jsp

```
1  <% response.setContentType("text/xml"); %>
2
3  <car-list>
4    <car type="Honda Accord">
5      <color>Black</color>
6      <sunroof>Yes</sunroof>
7    </car>
8
9    <car type="Ford Explorer">
10     <color>Red</color>
11     <sunroof>No</sunroof>
12   </car>
13 </car-list>
```



- Or, if backed by a dynamic resource, you can build a simple QueryString dialect:
  - http://www.davisworld.org/carList?action=getAvailable&make=Honda&color=red&year=2001
    - This could be a thin mapping to SQL, or to more complex/calculated business logic



- This human-readable "StringBuffer.append" method makes it incredibly easy to use in a web browser
  - But you can also use a Java client as well in a non-web app

- Download the open source HttpClient package from Apache:

http://jakarta.apache.org/commons/httpclient/


```
public static String get(String url)
{
    String doc = "";
    HttpClient client = new HttpClient();
    HttpMethod method = null;

    try
    {
        //get doc
        method = new GetMethod(url);

        // Execute the method.
        int statusCode = client.executeMethod(method);
        if (statusCode != HttpStatus.SC_OK)
        {
            System.err.println("Method failed: " + method.getStatusLine());
        }

        // Read the response body.
        doc = method.getResponseBodyAsString();
    }
    catch (IOException e)
    {
        System.err.println("Failed to download file.");
        e.printStackTrace();
    }
    finally
    {
        // Release the connection.
        method.releaseConnection();
        return doc;
    }
}
```

- To see a “Popular” RESTful WS in action, let’s go over to Yahoo!

How do I get started?

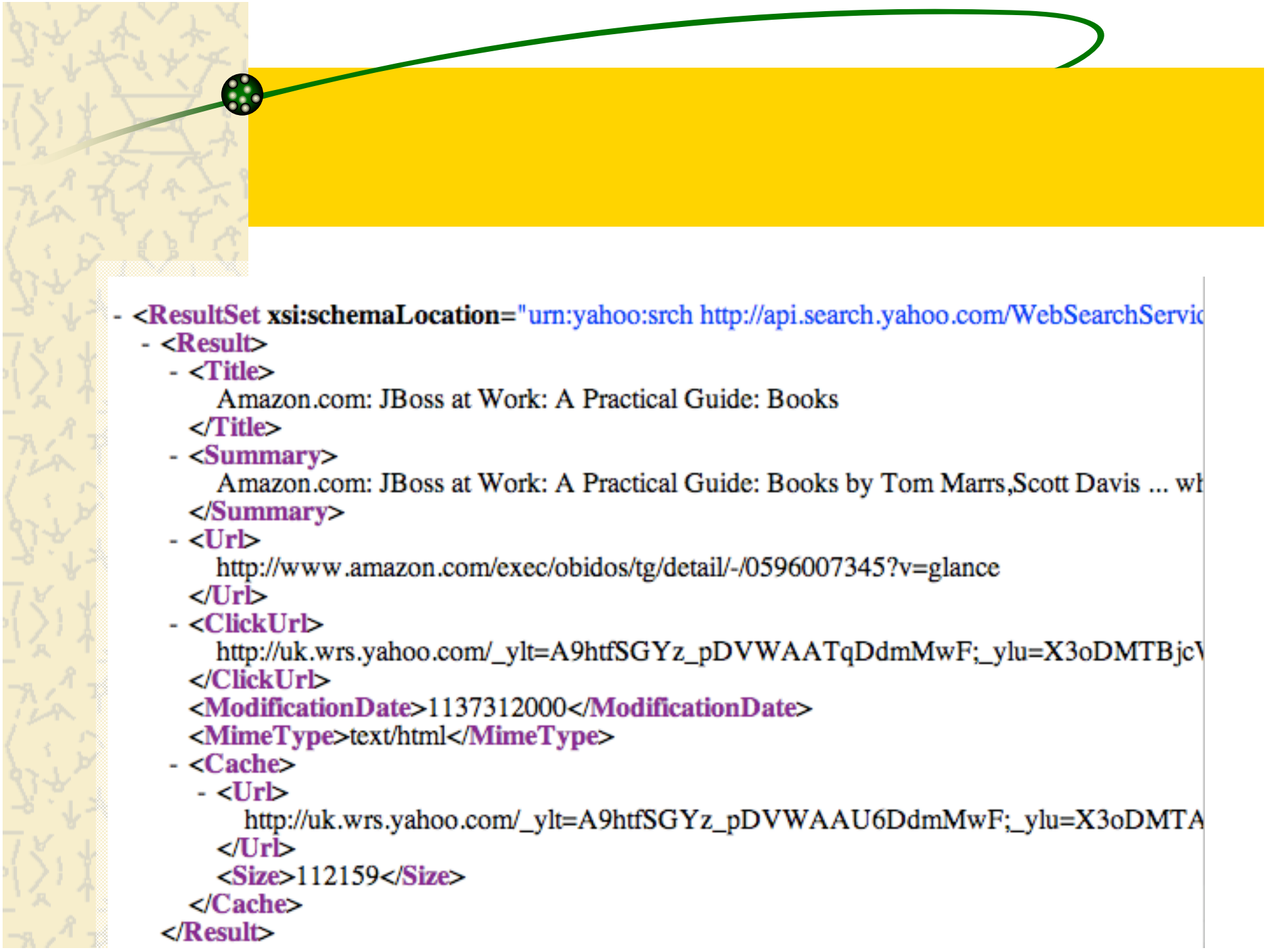
1. Online Documentation
Read the online documentation and FAQs.

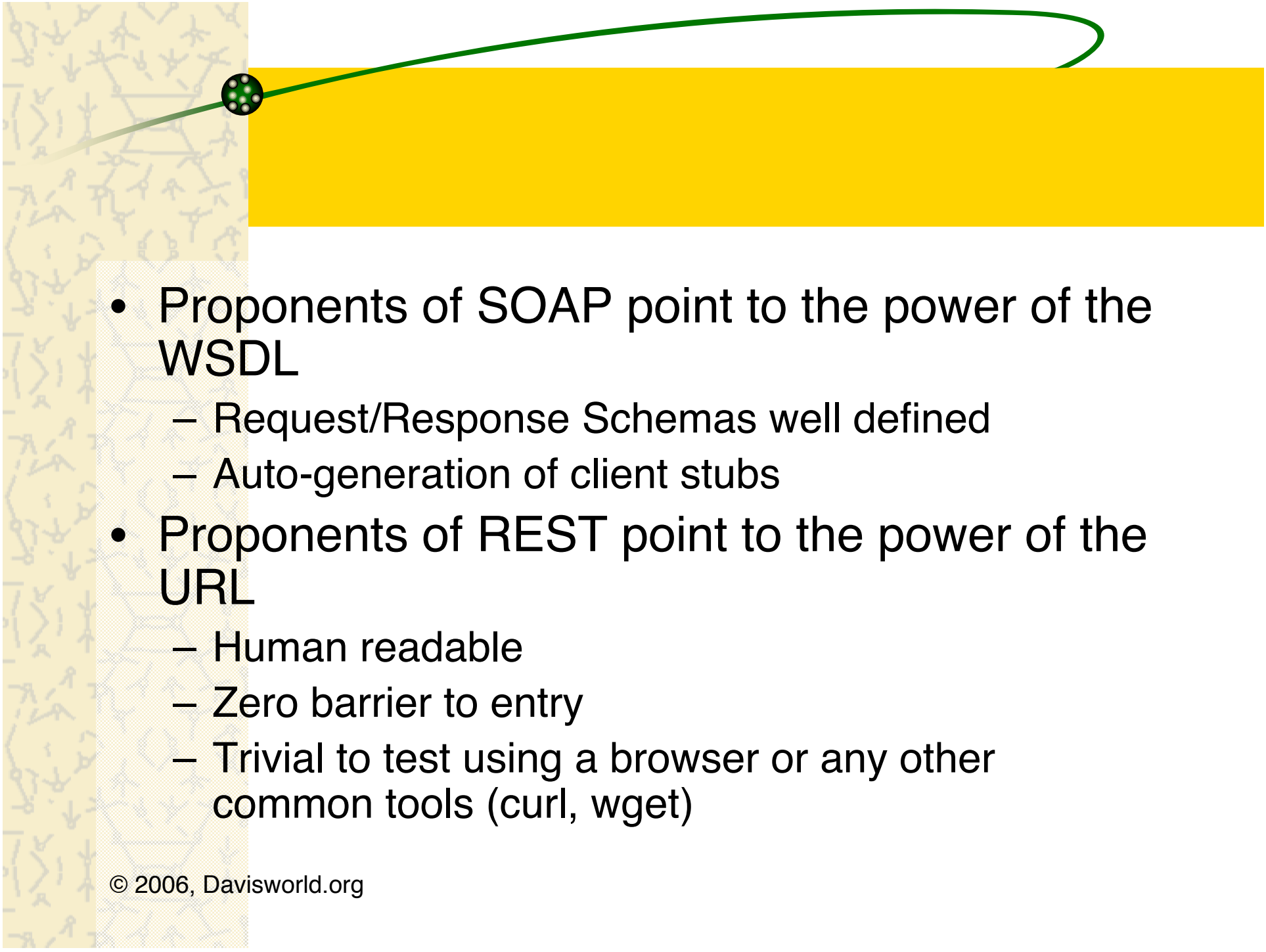
2. Get an Application ID
To access Yahoo! Search Webservices, you will need to get an application ID. Like a browser's User-Agent string, the Application ID uniquely identifies your application and has no effect on rate limiting.

3. Download the SDK
The development kit includes BSD licensed examples and libraries for various languages: Perl, Python and PHP, Java, JavaScript, and Flash.

(Source: http://developer.yahoo.net/search/index.html)

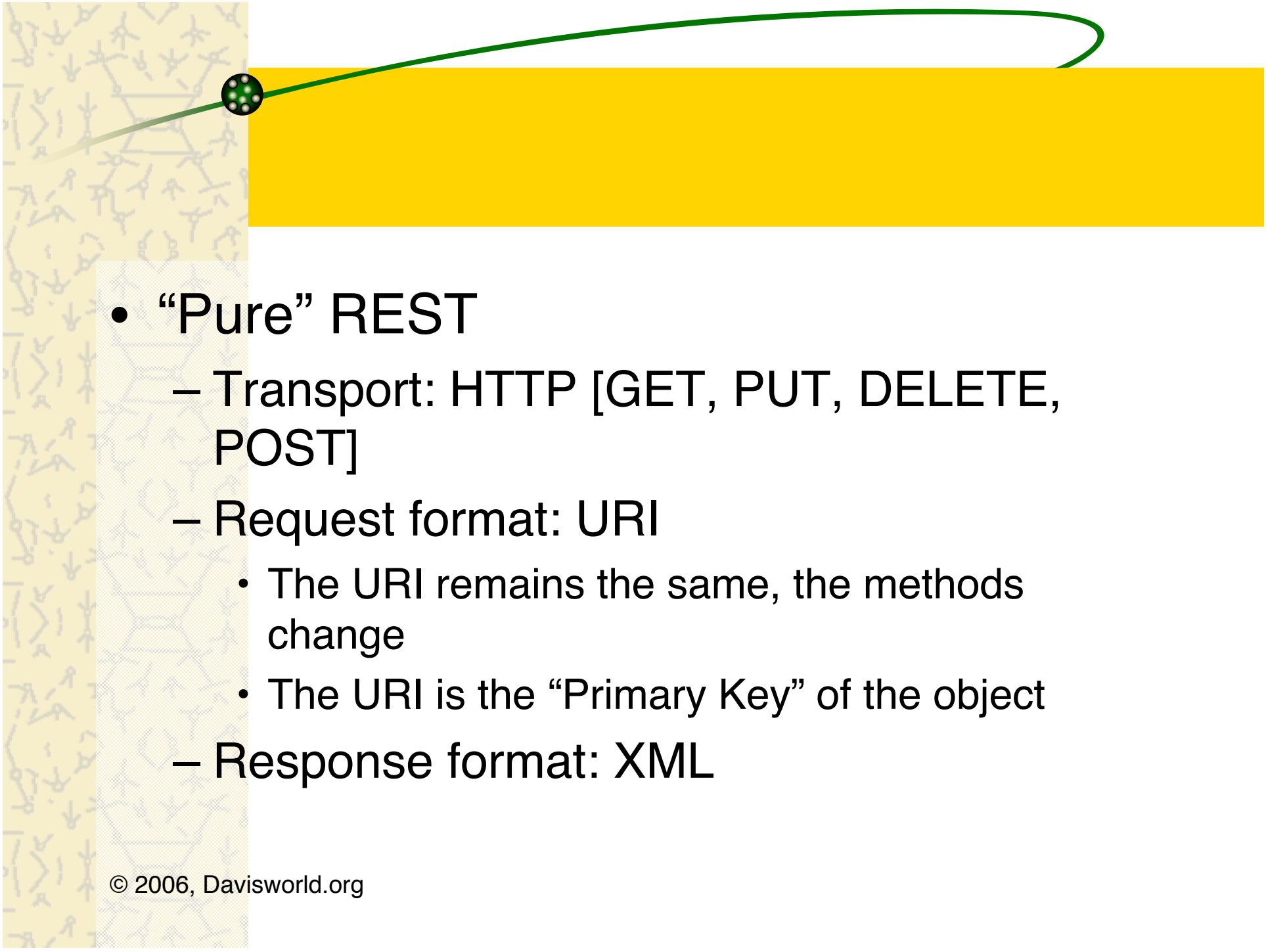
- Yahoo offers close to 20 different RESTful services (Search, Maps, Weather, Traffic, Finance, Shopping)
  - Here is the same web search we did in Google:
    - http://api.search.yahoo.com/WebSearchService/V1/webSearch?appid=00000000&query=jboss+at+work


- <ResultSet xsi:schemaLocation="urn:yahoo:srch http://api.search.yahoo.com/WebSearchServic
- <Result>
- <Title>
Amazon.com: JBoss at Work: A Practical Guide: Books
</Title>
- <Summary>
Amazon.com: JBoss at Work: A Practical Guide: Books by Tom Marrs,Scott Davis ... wh
</Summary>
- <Url>
http://www.amazon.com/exec/obidos/tg/detail/-/0596007345?v=glance
</Url>
- <ClickUrl>
http://uk.wrs.yahoo.com/_ylt=A9htfSGYz_pDVWAATqDdmMwF;_ylu=X3oDMTBjcV
</ClickUrl>
<ModificationDate>1137312000</ModificationDate>
<MimeType>text/html</MimeType>
- <Cache>
- <Url>
http://uk.wrs.yahoo.com/_ylt=A9htfSGYz_pDVWAAU6DdmMwF;_ylu=X3oDMTA
</Url>
<Size>112159</Size>
</Cache>
</Result>



- Proponents of SOAP point to the power of the WSDL
  - Request/Response Schemas well defined
  - Auto-generation of client stubs
- Proponents of REST point to the power of the URL
  - Human readable
  - Zero barrier to entry
  - Trivial to test using a browser or any other common tools (curl, wget)

- But both implementations I've shown you thus far have been more similar philosophically than different
  - Both have been RPC-centric
    - Only the syntax changed
- A "Pure" REST app is Resource-centric, not RPC-centric



- "Pure" REST
  - Transport: HTTP [GET, PUT, DELETE, POST]
  - Request format: URI
    - The URI remains the same, the methods change
    - The URI is the "Primary Key" of the object
  - Response format: XML

REST versus RPC

[edit]

A REST web application requires a different design approach than an RPC application. In RPC, the emphasis is on the diversity of protocol operations, or *verbs*; for example, an RPC application might define operations such as the following:

```
getUser()
addUser()
removeUser()
updateUser()
getLocation()
addLocation()
removeLocation()
updateLocation()
listUsers()
listLocations()
findLocation()
findUser()
```

(Source: http://en.wikipedia.org/wiki/REST)

With REST, on the other hand, the emphasis is on the diversity of resources, or *nouns*; for example, a REST application might define the following two resource types

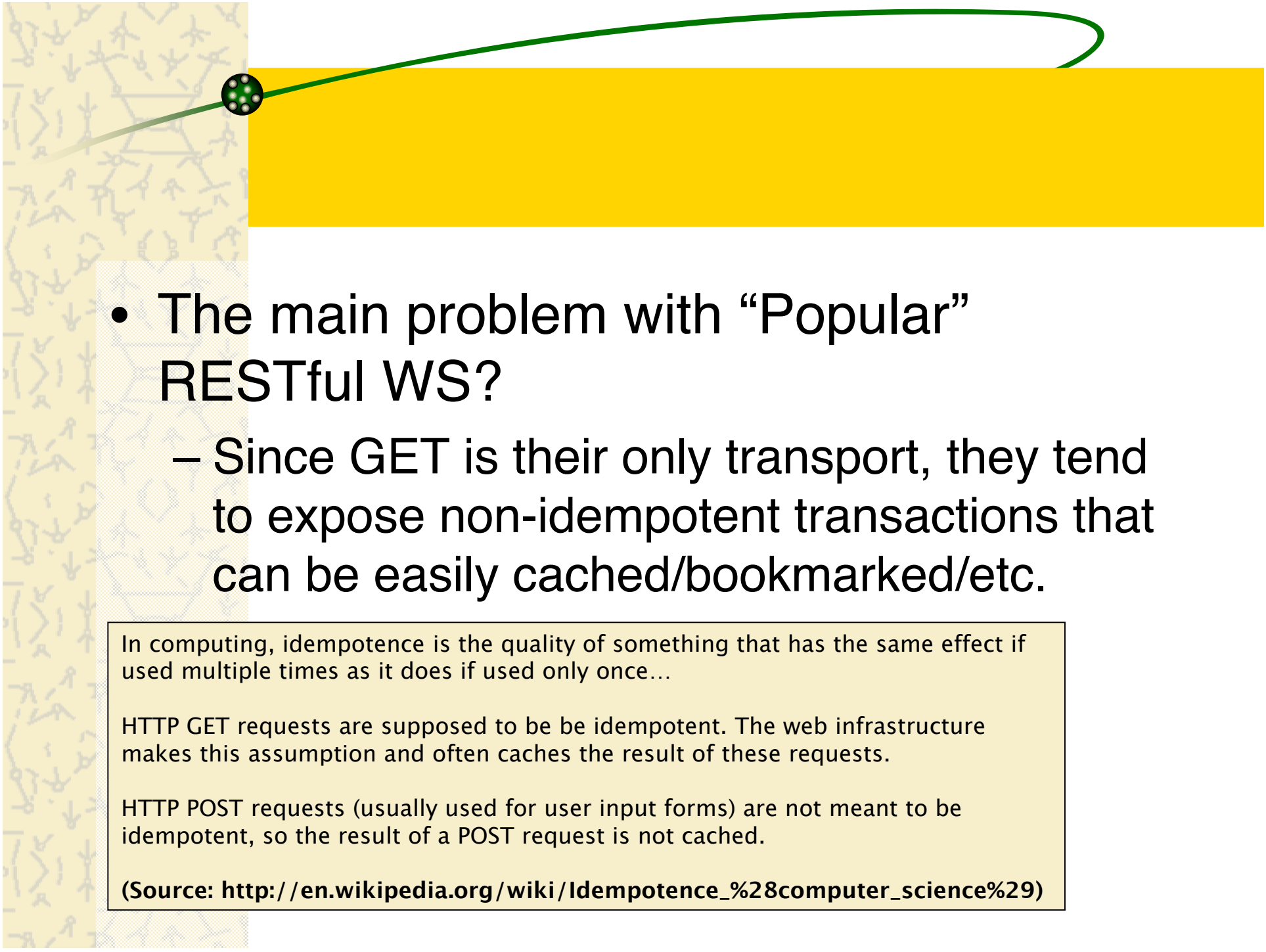
```
User {}
Location {}
```

Each resource would have its own location, such as `http://www.example.org/locations/us/ny/new_york_city`. Clients work with those resources through the standard HTTP operations, such as GET to download a copy of the resource, PUT to upload a changed copy, or DELETE to remove all representations of that resource. Note how each object has its own URL and can easily be cached, copied, and bookmarked. POST is generally used for actions with side-effects, such as placing a purchase order, or adding some data to a collection.

(Source: http://en.wikipedia.org/wiki/REST)

- Implementing a Pure RESTful app using servlets is trivial:
  - doGet, doPut, doDelete, doPost
- For a great series of articles on building your own RESTful WS by Joe Gregorio:
  - http://www.xml.com/pub/at/34

- With SOAP, the WSDL is required
  - Need to define method names, signatures
  - The binding to HTTP is one of many choices
    - There is nothing natively “web-like” about SOAP
- With REST, no WSDL is required because you are using the native language/semantics of the web
  - There is power in simplicity -- think SQL (select, insert, update, delete)



- The main problem with “Popular” RESTful WS?
  - Since GET is their only transport, they tend to expose non-idempotent transactions that can be easily cached/bookmarked/etc.

In computing, idempotence is the quality of something that has the same effect if used multiple times as it does if used only once…

HTTP GET requests are supposed to be be idempotent. The web infrastructure makes this assumption and often caches the result of these requests.

HTTP POST requests (usually used for user input forms) are not meant to be idempotent, so the result of a POST request is not cached.

(Source: http://en.wikipedia.org/wiki/Idempotence_%28computer_science%29)

What is Atom?

- For a great example of a “Pure” RESTful WS, let’s take a look at Atom

> Atom is the name of a specific web feed format. Web feeds, from a user's perspective, allow Internet users to subscribe to websites that change or add content regularly. To use this technology, site owners create or obtain specialized software (such as a content management system) which, in the machine-readable XML format, presents new articles in a list, giving a line or two of each article and a link to the full article or post.
>
> **(Source: http://en.wikipedia.org/wiki/Atom_%28standard%29)**

  - Atom is an IETF standard format for weblogs
    - RSS (and it’s predecessor RDF) is a non-standardized, RPC-centric protocol
    - Most applications support both RSS and Atom at this point

- For example, my blog is at the URI http://www.davisworld.org/blojsom/blog/
  - You can read it using a standard web browser or specialized software called a “feed reader” or “news aggregator”
    - The aggregator retrieves the XML and formats it independently of the L&F of the website
    - It is like a combination of a web browser and an email client
      - I subscribe to different feeds, and new/unread entries “appear” in my “inbox”

- Each new “feed” is a “syndication”
  - The URI remains the same, but new data can always be found at the same address
    - Each entry has a unique URI as well -- the “permalink”
      - http://www.davisworld.org/blojsom/blog/default/2006/02/01/Evolve.html
    - Since the resources are accessed via a simple GET, I can easily bookmark them

- The Atom Publishing Protocol is fully RESTful

> The Atom Publishing Protocol uses HTTP to edit and author web resources. The Atom Protocol uses the following HTTP methods:
>
> GET is used to retrieve a representation of a resource or perform a query.
> POST is used to create a new, dynamically-named resource.
> PUT is used to update a known resource.
> DELETE is used to remove a resource.
>
> **(Source: http://ietfreport.isoc.org/idref/draft-ietf-atompub-protocol/)**

  - You can use the ROME framework to interact with Atom and RSS feeds programmatically
    - https://rome.dev.java.net/

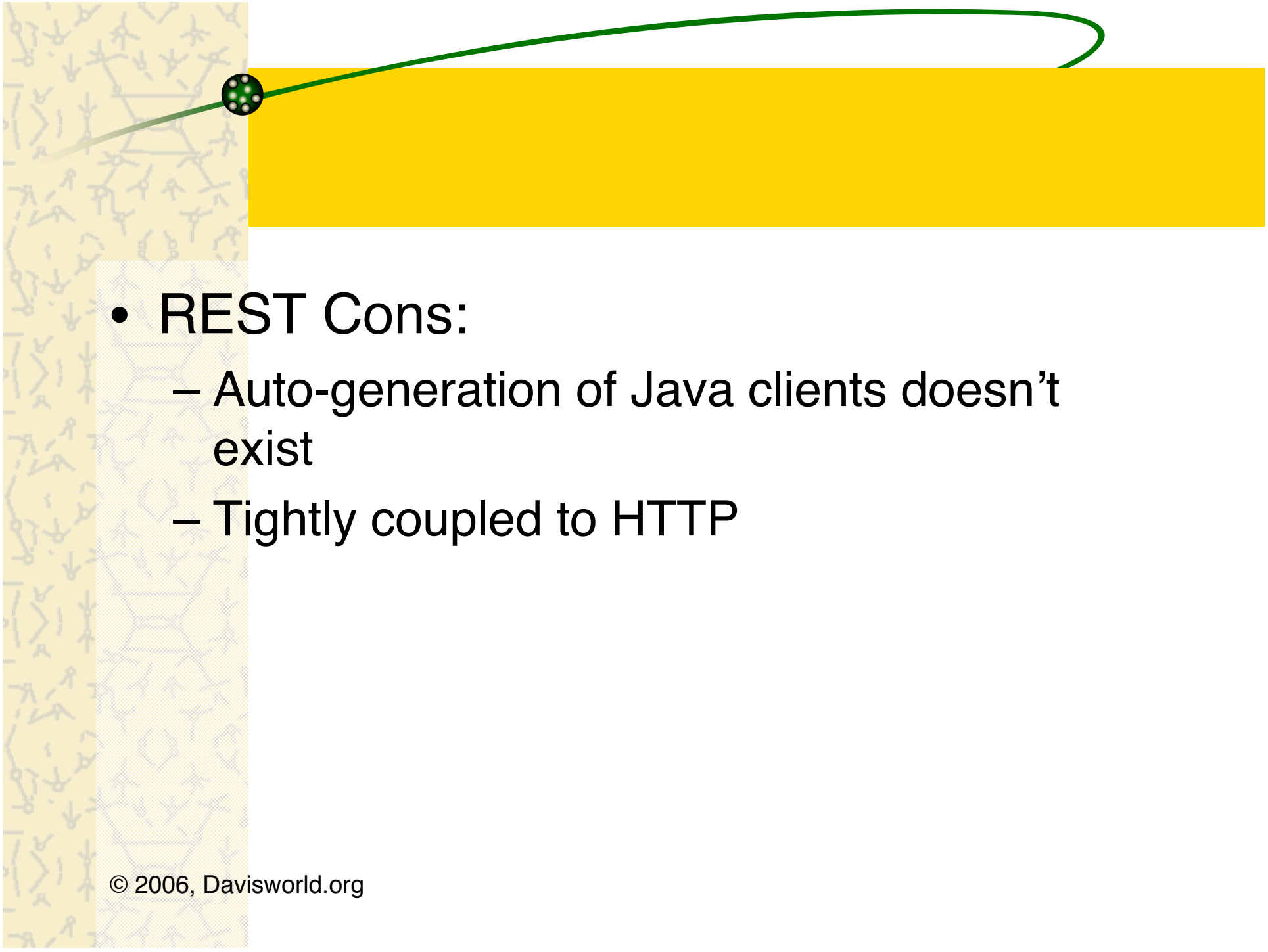
Example of an Atom 1.0 Feed

[edit]

An example of an Atom Feed document:

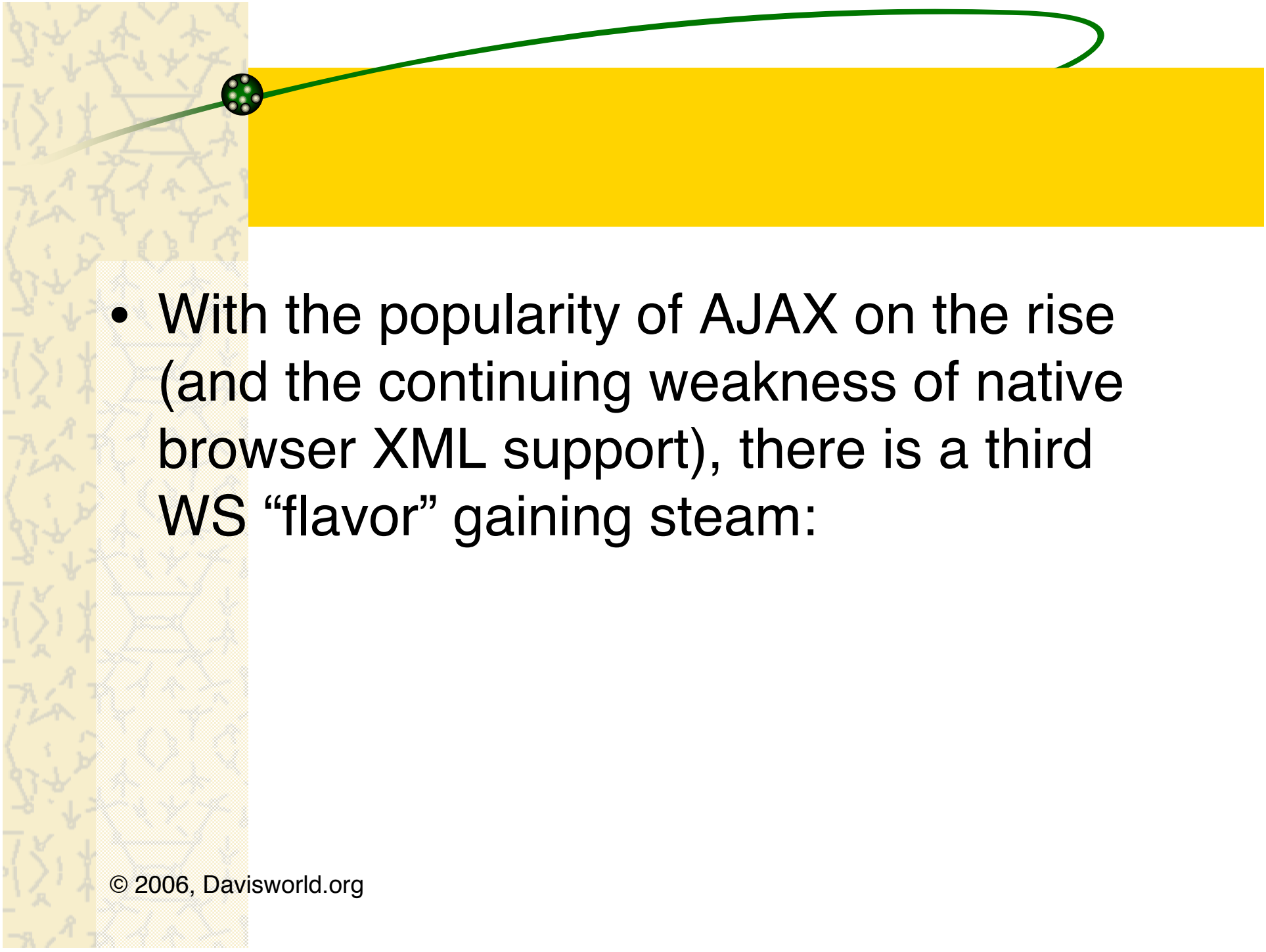
```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">

 <title>Example Feed</title>
 <subtitle>Insert witty or insightful remark here</subtitle>
 <link href="http://example.org/"/>
 <updated>2003-12-13T18:30:02Z</updated>
 <author>
   <name>John Doe</name>
   <email>johndoe@example.com</email>
 </author>
 <id>urn:uuid:60a76c80-d399-11d9-b91C-0003939e0af6</id>

 <entry>
   <title>Atom-Powered Robots Run Amok</title>
   <link href="http://example.org/2003/12/13/atom03"/>
   <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
   <updated>2003-12-13T18:30:02Z</updated>
   <summary>Some text.</summary>
 </entry>

</feed>
```

- REST Pros:
  - Very browser/AJAX-friendly
  - Philosophically closer to the traditional web than SOAP



- REST Cons:
  - Auto-generation of Java clients doesn't exist
  - Tightly coupled to HTTP

- Many popular websites offer both SOAP and RESTful interfaces
  - Amazon
    - http://www.amazon.com/gp/browse.html/102-2243700-0003349?node=3435361
  - eBay
    - http://developer.ebay.com/common/api
- A good search term to discover WS offerings is “[website] api”
  - Ebay api
  - Amazon api



- With the popularity of AJAX on the rise (and the continuing weakness of native browser XML support), there is a third WS “flavor” gaining steam:

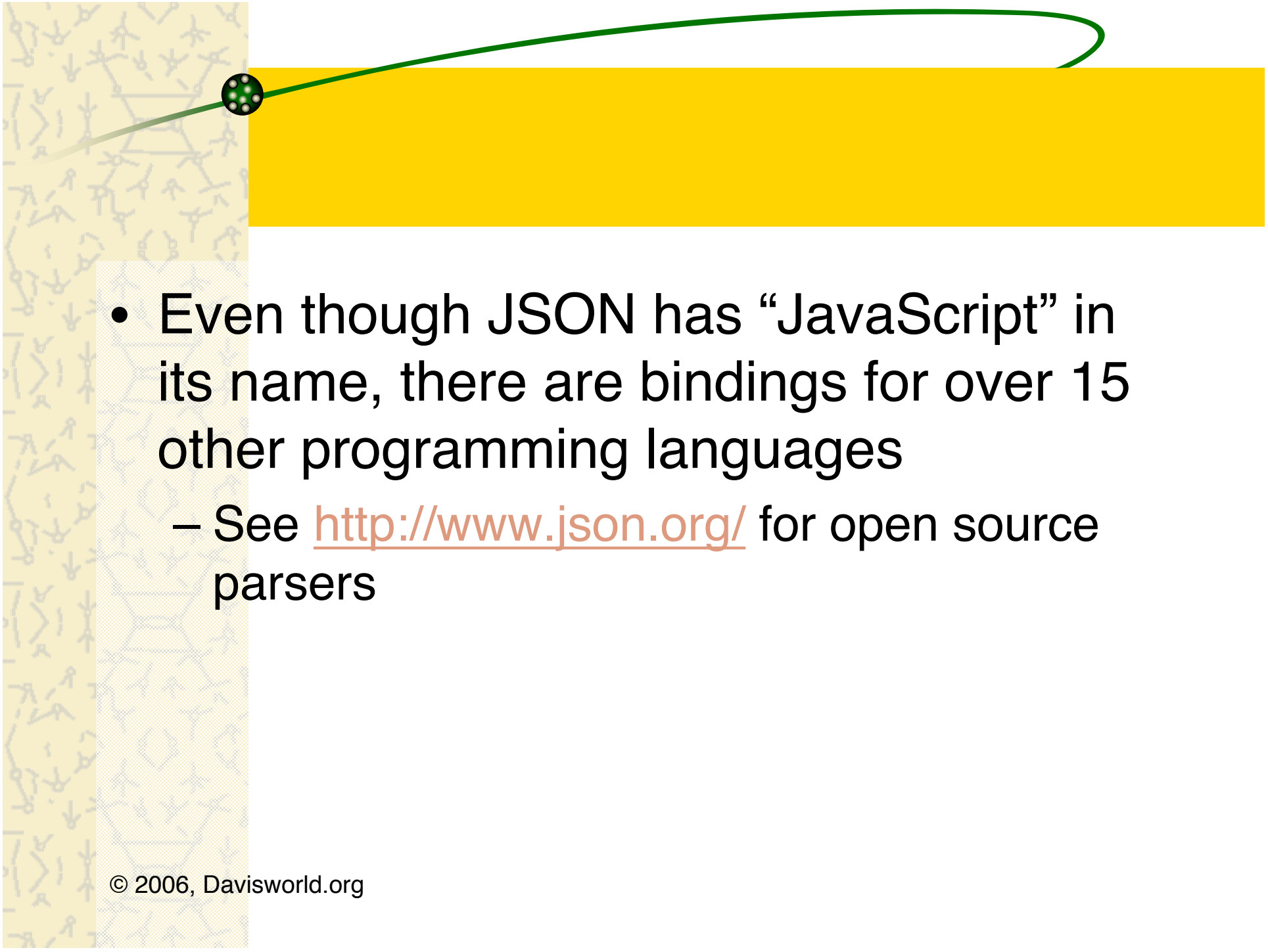
What is JSON?

- We don’t need no stinkin’ XML…

JSON (pronounced Jason), which stands for "JavaScript Object Notation", is a lightweight computer data interchange format. JSON is a subset of the object literal notation of JavaScript but its use does not require JavaScript.

JSON's simplicity has resulted in its widespread use, especially as an alternative to XML in Ajax. One of the claimed advantages of JSON over XML as a data interchange format in this context is that it is much easier to write a JSON parser. In Javascript itself, JSON can be parsed trivially using the eval() procedure. This was important for the acceptance of JSON within the AJAX programming community because of JavaScript's ubiquity among web browsers.

(Source: http://en.wikipedia.org/wiki/JSON)



- Even though JSON has “JavaScript” in its name, there are bindings for over 15 other programming languages
  - See http://www.json.org/ for open source parsers

JSON example

[edit]

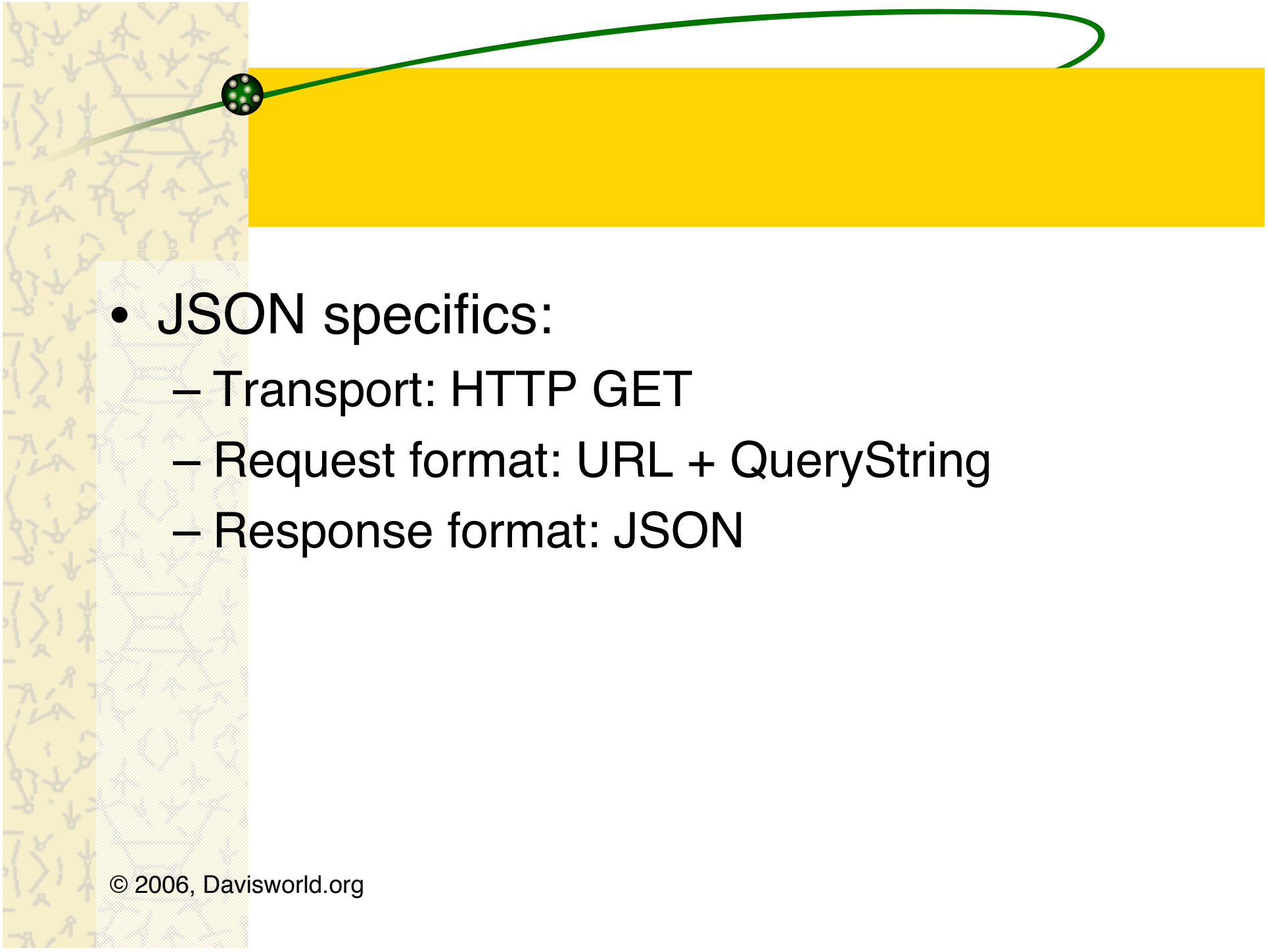
The following is a simple example of a menubar definition using JSON and XML data encodings.

JSON:

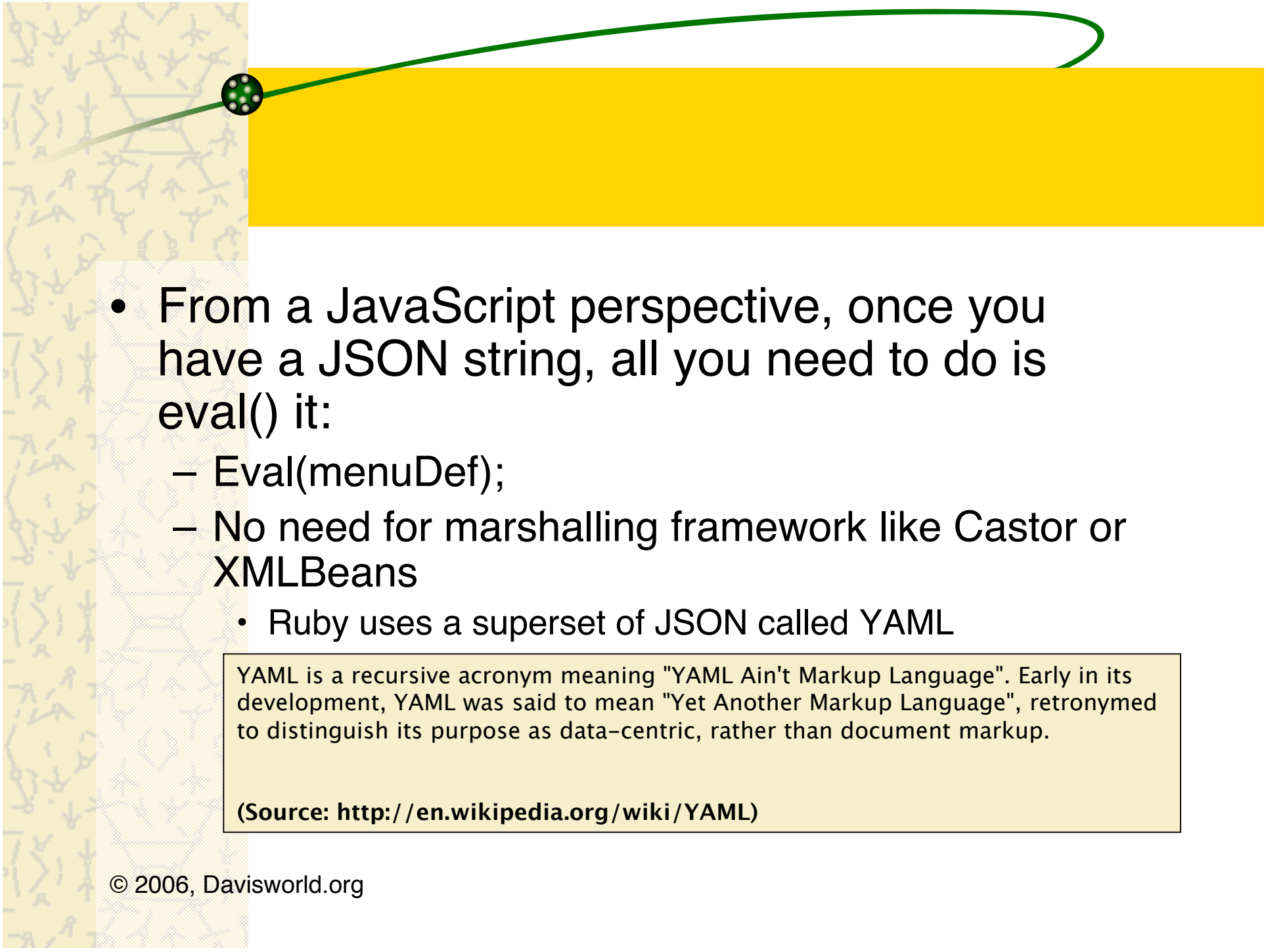
```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

XML:

```
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```



- JSON specifics:
  - Transport: HTTP GET
  - Request format: URL + QueryString
  - Response format: JSON



- From a JavaScript perspective, once you have a JSON string, all you need to do is eval() it:
  - Eval(menuDef);
  - No need for marshalling framework like Castor or XMLBeans
    - Ruby uses a superset of JSON called YAML

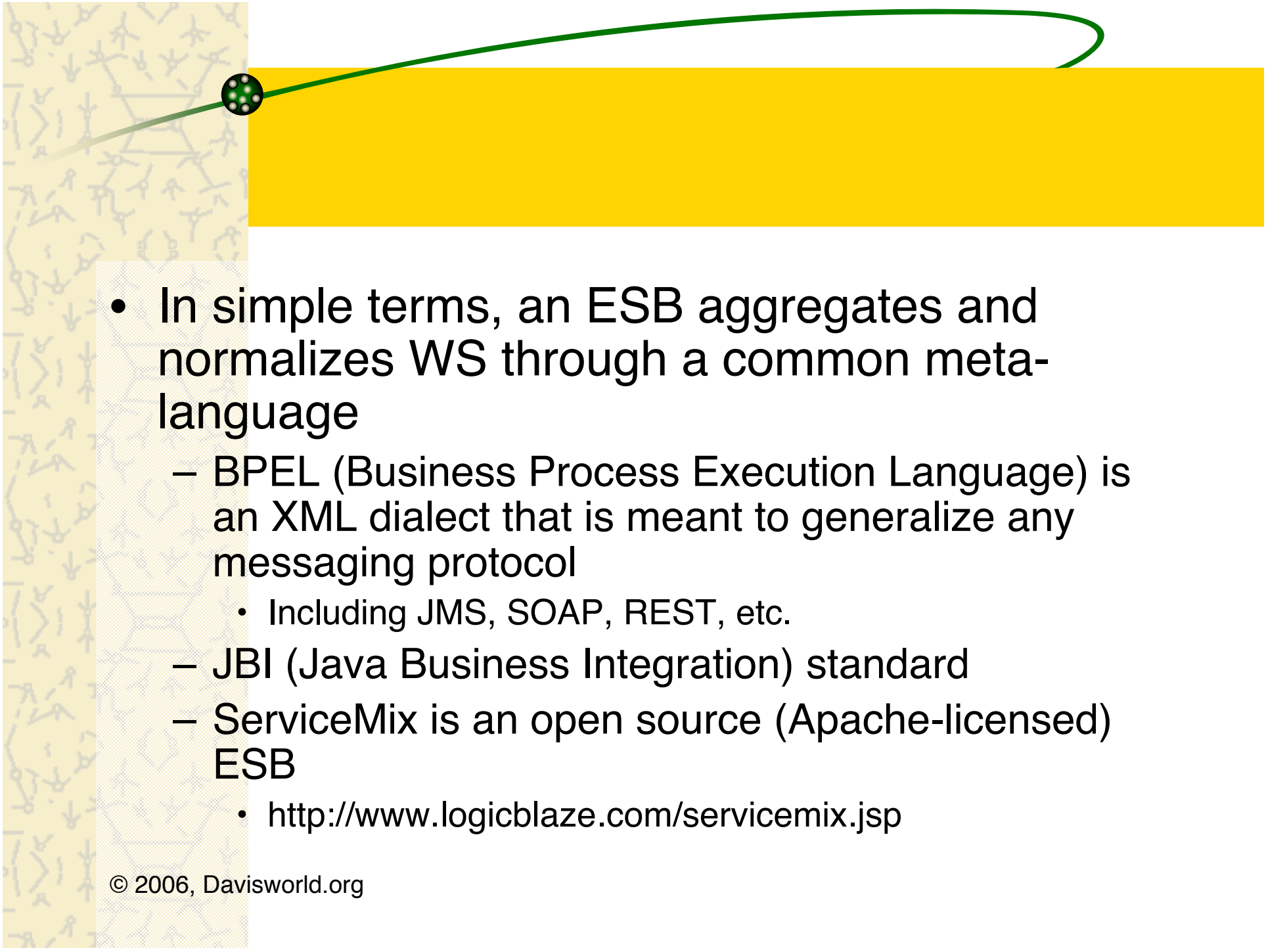
YAML is a recursive acronym meaning "YAML Ain't Markup Language". Early in its development, YAML was said to mean "Yet Another Markup Language", retronymed to distinguish its purpose as data-centric, rather than document markup.

(Source: http://en.wikipedia.org/wiki/YAML)

What is ESB?

- Once you have a solid understanding of SOA/WS and all of the various implementations, you might consider putting an ESB in place

> In computing, an enterprise service bus refers to a software architecture construct, implemented by technologies found in a category of middleware infrastructure products usually based on Web services standards, that provides foundational services for more complex service-oriented architectures via an event-driven and XML-based messaging engine (the bus).
>
> **(Source: http://en.wikipedia.org/wiki/Enterprise_Service_Bus)**



- In simple terms, an ESB aggregates and normalizes WS through a common meta-language
  - BPEL (Business Process Execution Language) is an XML dialect that is meant to generalize any messaging protocol
    - Including JMS, SOAP, REST, etc.
  - JBI (Java Business Integration) standard
  - ServiceMix is an open source (Apache-licensed) ESB
    - http://www.logicblaze.com/servicemix.jsp

- So, are you ready for a round of Buzzword Bingo?
  - SOA, WS
  - AJAX, HTML++, Client/SOA, XSS
  - Web 2.0
  - SOAP, WSDL, UDDI, XML-RPC
  - REST, POX
  - JSON, YAML
  - RSS, RDF, Atom
  - ESB, BPEL

Conclusion

- Thanks for your time!
  - Questions?
  - Email: scottdavis99@yahoo.com
  - Download slides:
    - http://www.davisworld.org/presentations