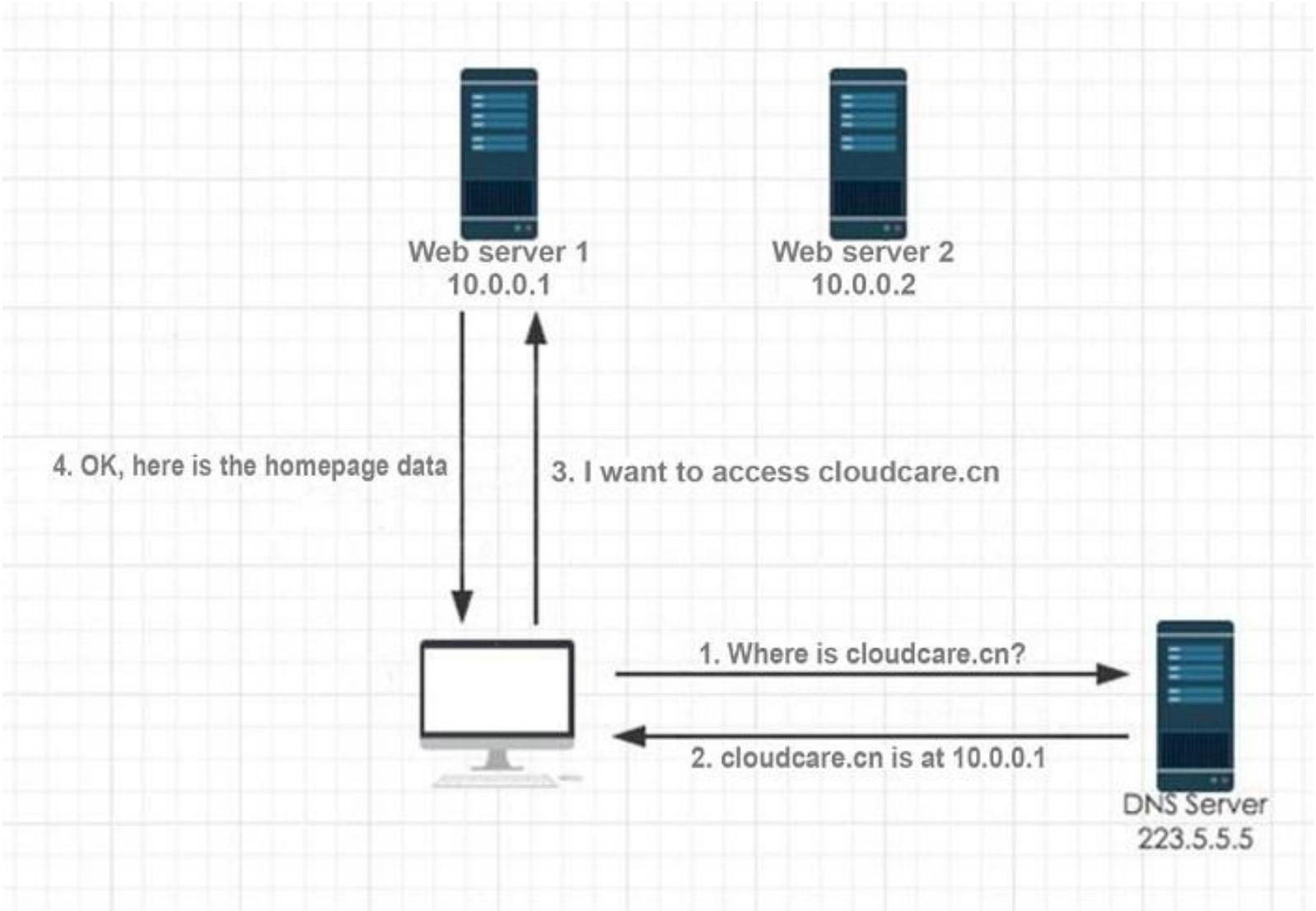
Web programming fundamentals

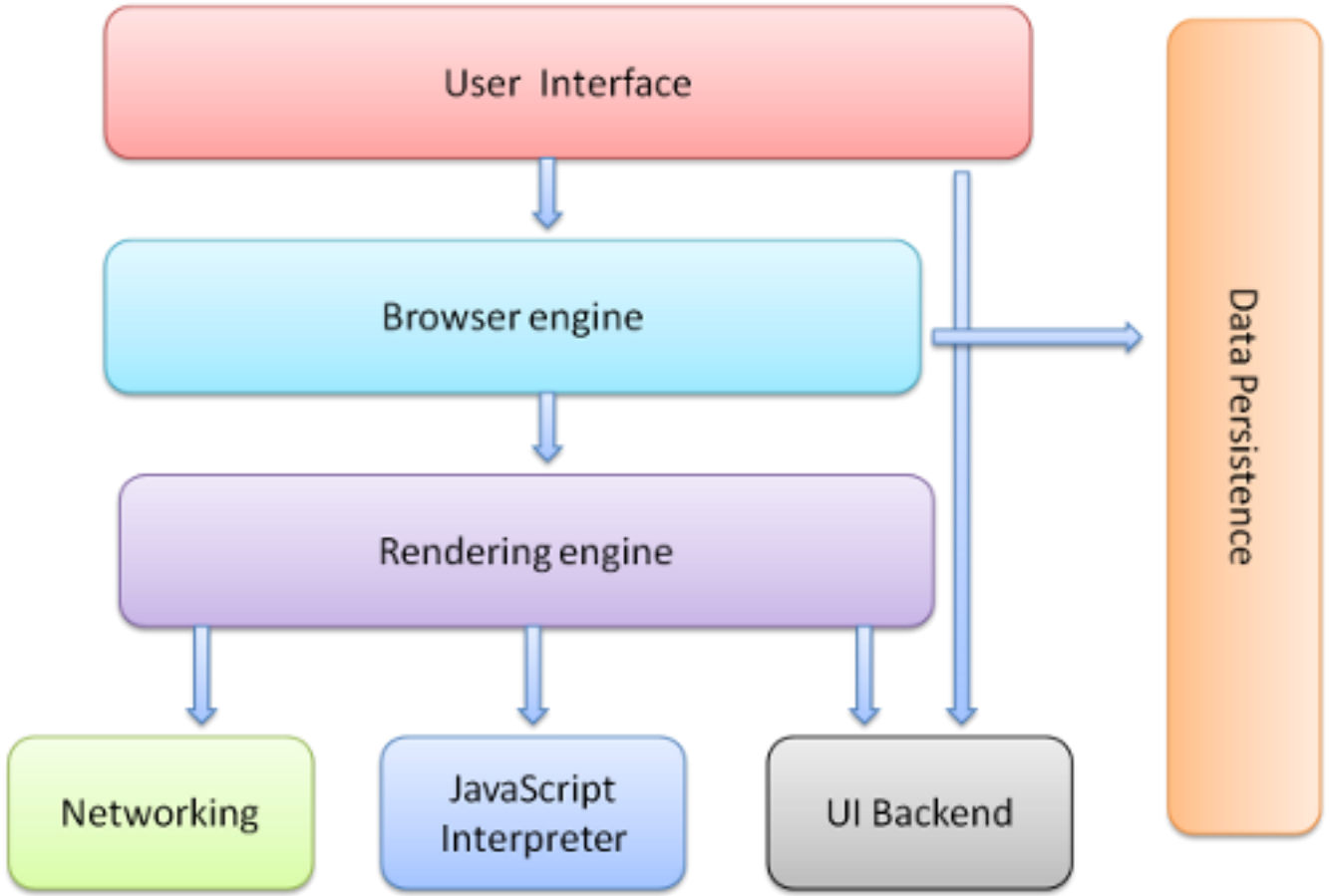
How do you access website?



How do you access website?

- Suppose that the IP address of the local DNS server is 223.5.5.5.
- The steps of accessing the website are as follows:
  1. The browser sends a request to the DNS server for the address of cloudcare.cn.
  2. The DNS server returns the IP address of the server that is hosting cloudcare.cn. In this example, the IP address is 10.0.0.1.
  3. The local computer caches the DNS data and sends an access request for cloudcare.cn to the server with the IP address 10.0.0.1.
  4. The server establishes a TCP connection with the local computer to exchange data.

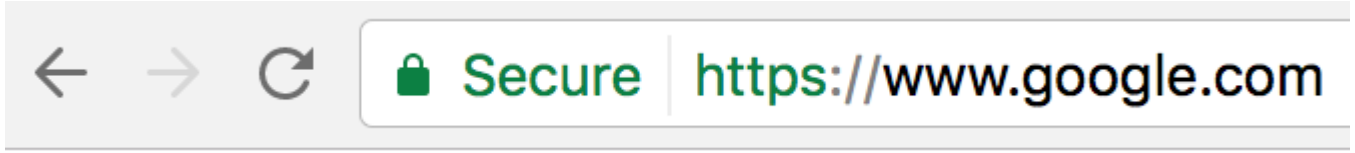
Working of Web Browser



Working of Web Browser

- **User Interface**
  - The Address Bar is one of the good example. It interface with user to insert URI and interacts with various component to render the web page.

  - There are plenty of user interfaces for various needs like Back and Forward Buttons, Bookmark, Reload, Stop and Home Buttons which you see below.



Working of Web Browser

- **Browser Engine**
  - It’s a bridge between User Interface and Rendering Engine. It provides methods to initiate the loading of a URL and other actions like (reload, back and forward).
- **Rendering Engine**
  - Responsible for displaying requested content. For example if the requested content is HTML, the rendering engine parses HTML and CSS, and displays the parsed content on the screen.
  - ach browser use various engines like Chrome and Opera uses Blink, Firefox uses Gecko, IE Edge uses EdgeHTML, Internet Explorer uses Trident, Apple Safari uses WebKit.

Working of Web Browser

- **Networking:** for network calls such as HTTP requests, using different implementations for different platform behind a platform-independent interface.
- **UI backend:** used for drawing basic widgets like combo boxes and windows. This backend exposes a generic interface that is not platform specific. Underneath it uses operating system user interface methods.
- **JavaScript interpreter**: Used to parse and execute JavaScript code.
- **Data storage:** This is a persistence layer. The browser may need to save all sorts of data locally, such as cookies.

Working of Web Browser

Browser	Rendering / Layout Engine	Scripting Engine
Chrome	Blink (C++)	V8 (C++)
Mozilla Firefox	Gecko (C++)	SpiderMonkey (C/C++)
IE Edge	EdgeHTML (C++)	Chakra JavaScript engine (C++)
Opera	Blink (C++)	V8 (C++)
Internet Explorer	Trident (C++)	Chakra JScript engine (C++)
Apple Safari	WebKit (C++)	JavaScript Core (Nitro)

Role of Rendering Engine

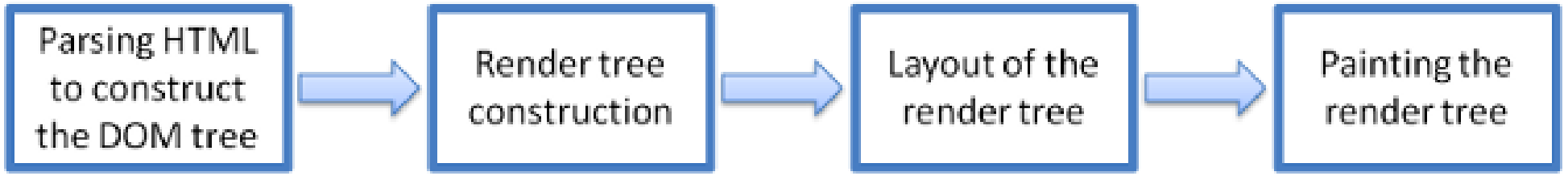
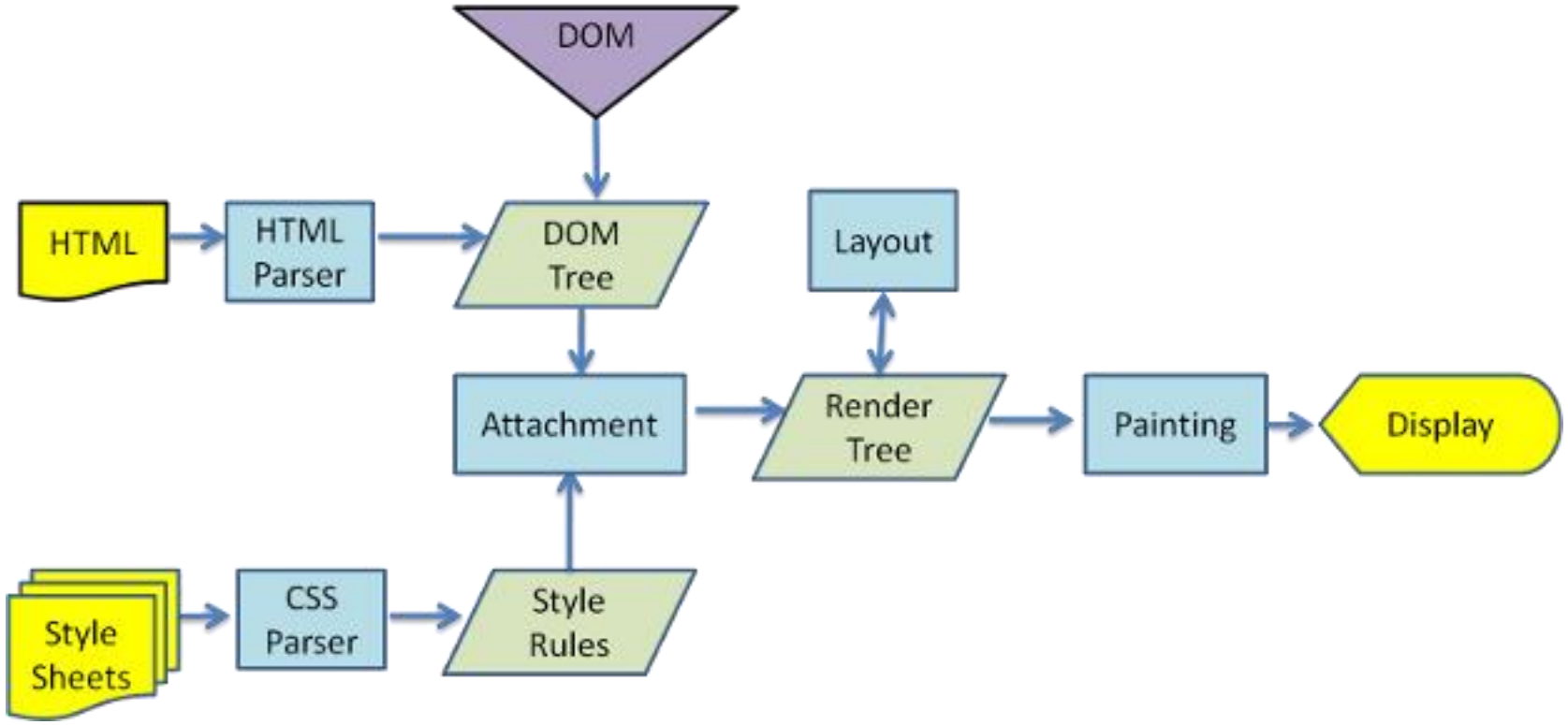
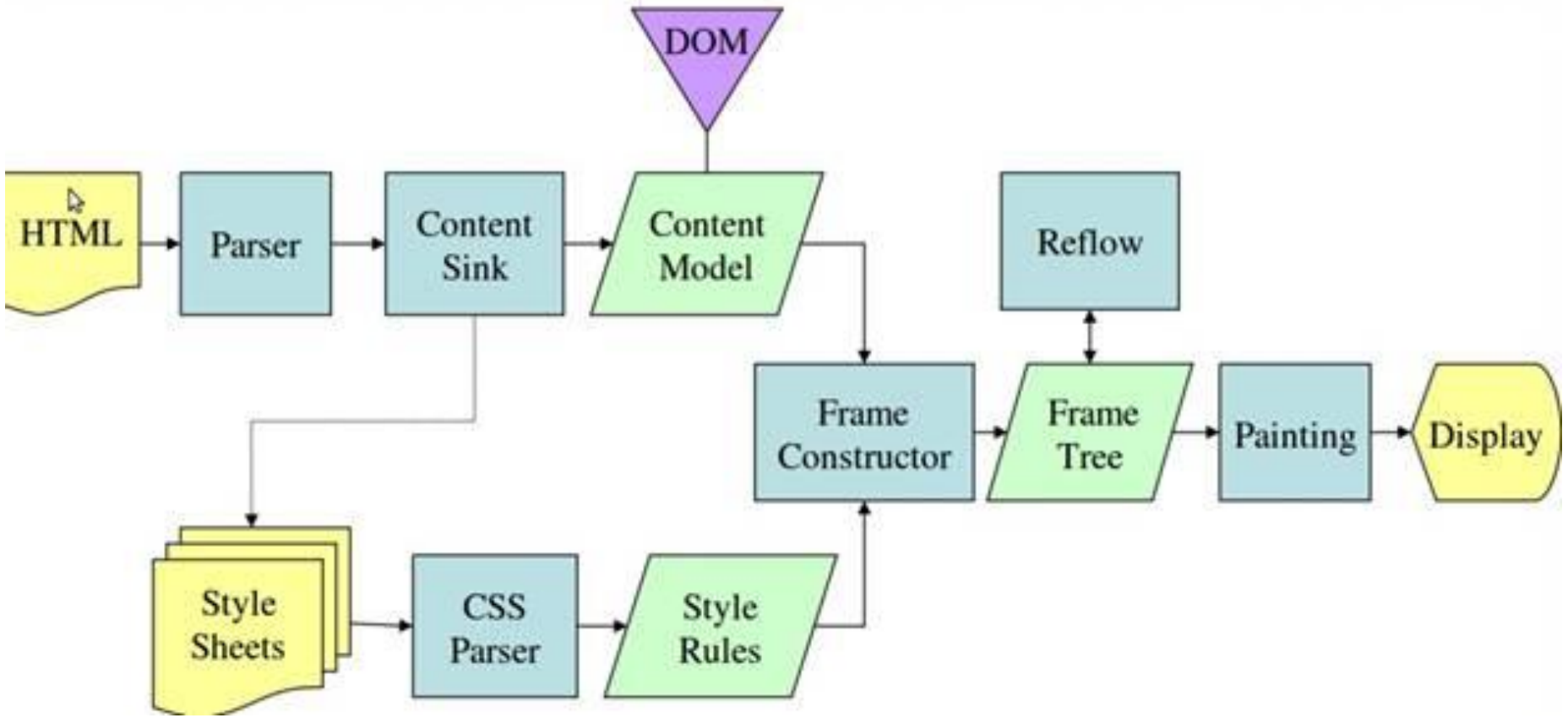


Figure : Rendering engine basic flow

WebKit main flow



Mozilla's Gecko rendering engine main flow



Role of Rendering Engine

- The following describes the rendering process of a browser.
- The HTML page for the homepage of cloudcare.cn is as follows:

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <link type="text/css" rel="stylesheet" href="/test.css"/>
    <title>Hello world!</title>
    <script src="/index.js" type="text/javascript"></script>
  </head>
  <body>
    <div>Hello world!</div>
    <img src="/img/bg.png" />
  </body>
</html>
```

Role of Rendering Engine

- Initially, a browser fetches page data from the server, which is called the base document, as shown in the figure above. Then, the browser converts the base document to a structured object named DOM tree and stores it in memory. Finally, the browser renders elements in the DOM tree as visual items according to the tree.

- **During the rendering, the browser renders the DOM tree from top to bottom and initiates HTTP requests once it detects external resources or scripts (such as the link, script and img tags in the figure above)**. Meanwhile, the browser will continue to parse the code:

Role of Rendering Engine

1. The web page displays the "Hello world" string.
2. The page loads and displays images.
3. If you have defined a style that is similar to "div {color: red;}" in the style sheet, the page will display the "Hello world" string in red once it has fully loaded the style sheet.

- Normally, this process is instantaneous and not visible to the user. However, when the connection is poor, you might be able to see the webpage develop based on the previously described sequence.

Other important topics for your knowledge

- Cookies
- Session
- Digital Certificates
- Digital Signature
- Secure Socket Layer

References

- https://www.alibabacloud.com/blog/how-do-web-browsers-work_224776
- https://computersciencewiki.org/index.php/Web_browsers
- https://medium.com/@ramsunvtech/behind-browser-basics-part-1-b733e9f3c0e6
- https://web.dev/howbrowserswork/

- **H**yper **T**ext **T**ransfer **P**rotocol
- Communication between web server & clients
- Request Response Model
- Loading pages, form submit.

HTTP IS Stateless

- Every request is completely independent.
- Similar to transaction
- Cache Control
- Programming, Local Storage, cookies, Sessions are used to create enhanced user experiences.

HTTPS

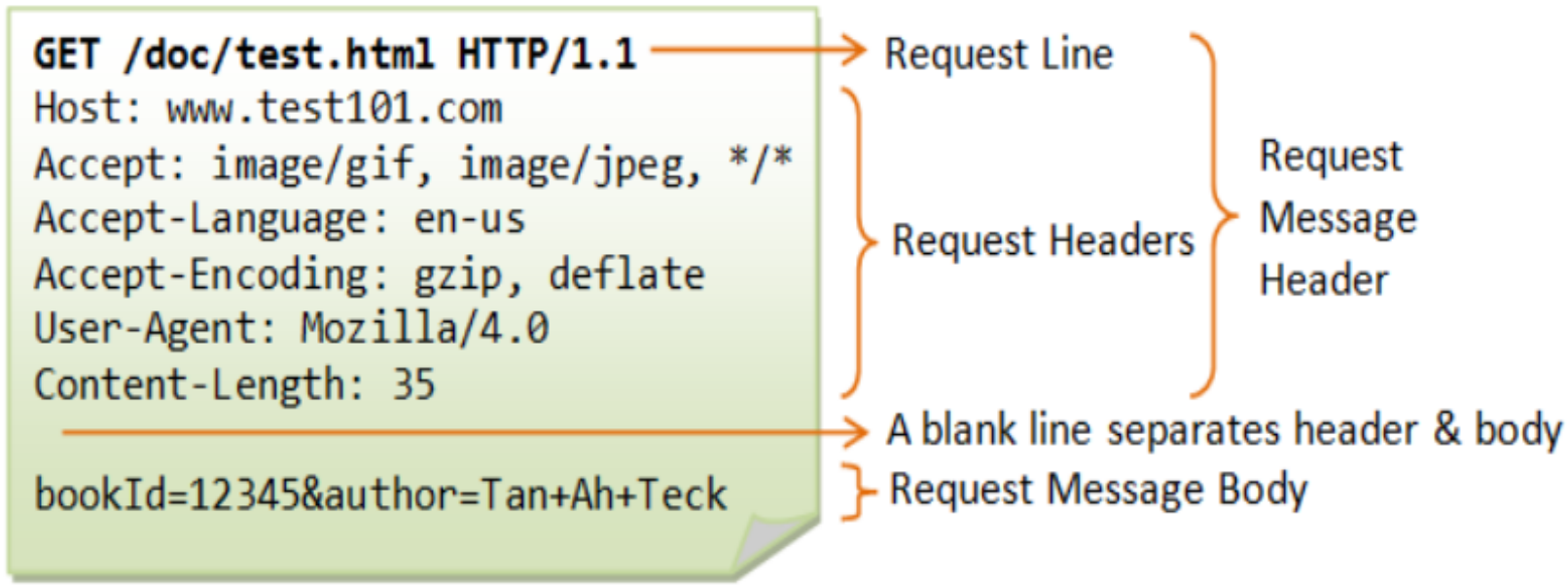
- **H**yper **T**ext **T**ransfer **P**rotocol **S**ecure
- Data sent is encrypted
- SSL / TLS
- Install certificate on web host

HTTP Request

- **Request line syntax:**

  request-method-name request-URI HTTP-version

- *request-method-name*: HTTP protocol defines a set of request methods. The client can use one of these methods to send a request to the server.
- *request-URI*: specifies the resource requested.
- *HTTP-version*: Two versions are currently in use: HTTP/1.0 and HTTP/1.1.


GET /doc/test.html HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Content-Length: 35
bookId=12345&author=Tan+Ah+Teck
Request Line
Request Headers
Request Message Header
A blank line separates header & body
Request Message Body

HTTP Methods

- GET

  Retrieves data from the server
- POST

  Submit data to the server
- PUT

  Update data already on the server
- DELETE

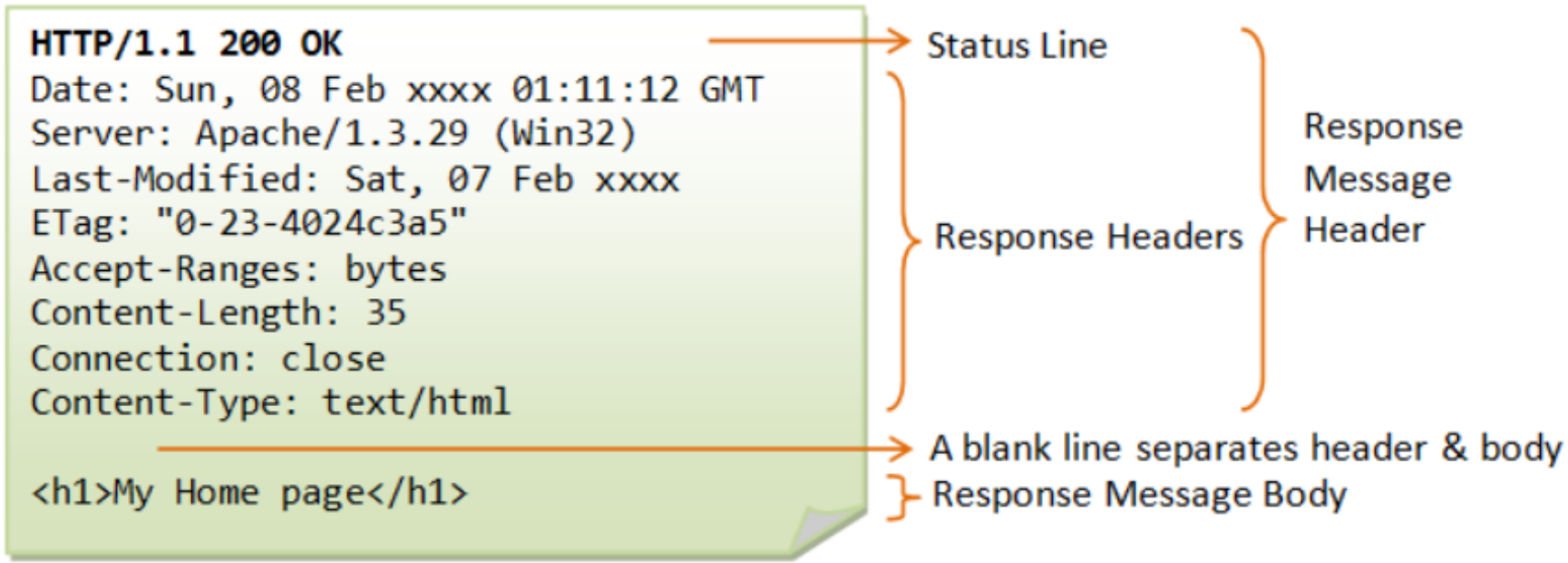
  Deletes data from the server

HTTP Response

- **Status Line syntax:**

HTTP version - *status code* - *reason-phrase*

- *HTTP-version*: The HTTP version used in this session. Either HTTP/1.0 and HTTP/1.1.
- *status-code*: a 3-digit number generated by the server to reflect the outcome of the request.
- *reason-phrase*: gives a short explanation to the status code.


HTTP/1.1 200 OK
Date: Sun, 08 Feb xxxx 01:11:12 GMT
Server: Apache/1.3.29 (Win32)
Last-Modified: Sat, 07 Feb xxxx
ETag: "0-23-4024c3a5"
Accept-Ranges: bytes
Content-Length: 35
Connection: close
Content-Type: text/html
<h1>My Home page</h1>
Status Line
Response Headers
Response Message Header
A blank line separates header & body
Response Message Body

HTTP Status Code

- 1xx: Informational

  Request received/Processing

- 2xx: Success

  Successfully received, understood & accepted

- 3xx:Redirect

  Further action must be taken/ redirect

- 4xx: Client Error

  Request does not have what it needs

- 5xx: Server Error

  Server failed to fulfil an apparent valid request

Examples...

- 200 - OK
- 201 - OK created
- 301 - Moved to new URL
- 304 - Not Modified(Cached version)
- 400 - Bad request
- 401 - Unauthorised
- 404 - Not Found
- 500 - Internal Server Error

URL

- URL-Uniform Resource locator.
- Eg.

http://www.amazon.in/about.html

- Made up of 3 parts:

  How: What Protocol?

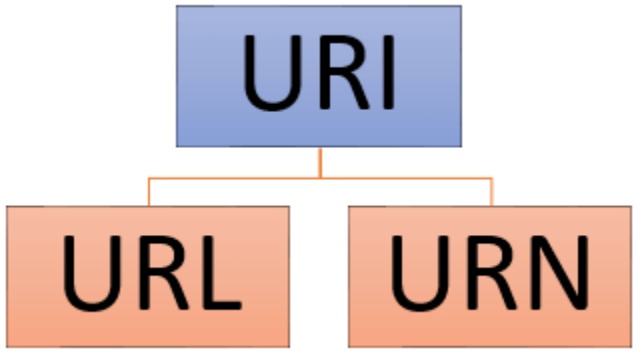
  Where: Which host?

  What: Path to the object required?


http://www.microsoft.com/index/index.html
How: this declares the protocol to be used
Where: this declares the address on the world wide web
What: the path on the server

URI

- Uniform Resource Identifier
- It is a string of characters to identify any resources on the internet using location, name, or both.



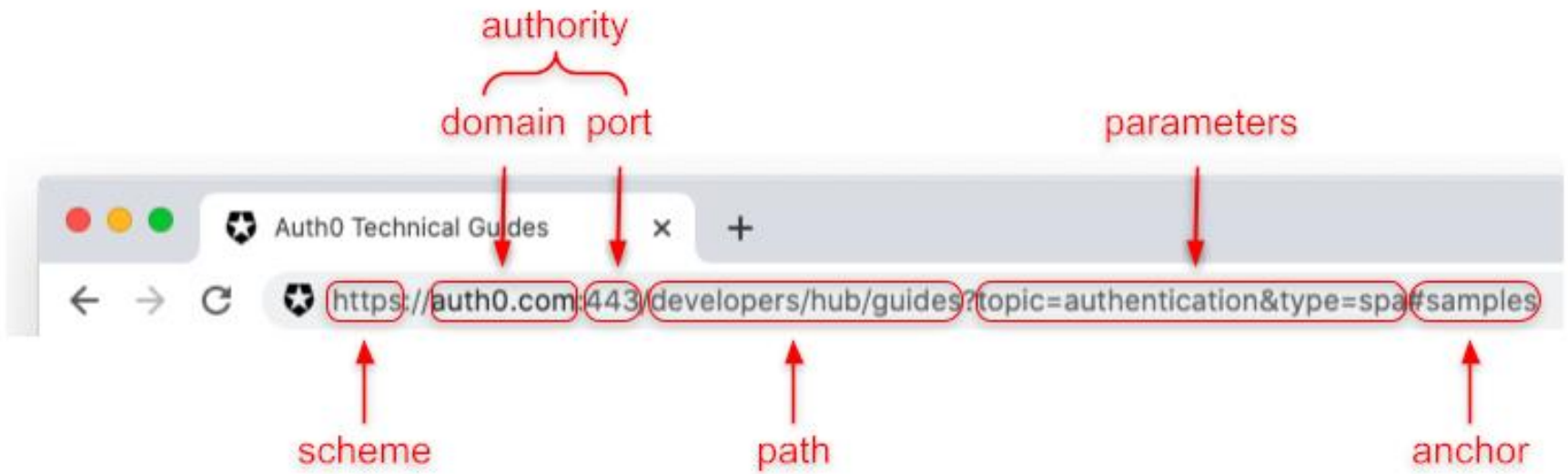
Syntax of URI

- URI = scheme:[//authority]path[?query][#fragment]

- The URI includes the following parts:

1. Scheme
2. Authority Component
3. Path
4. Query
5. Fragment

URI


authority
domain
port
parameters
Auth0 Technical Guides
https://auth0.com:443/developers/hub/guides?topic=authentication&type=spa#samples
scheme
path
anchor

KEY DIFFERENCES: URL &URN

- URL is a subset of URI that specifies where a resource exists and the mechanism for retrieving it, while URI is a superset of URL that identifies a resource
- The main aim of URL is to get the location or address of a resource whereas the main aim of URI is to find a resource.
- URL is used to locate only web pages, on the other hand, URI in used in HTML, XML and other files.
- URL contains components such as protocol, domain, path, hash, query string, etc. while, URI contains components like scheme, authority, path, query, etc.
- Example of URL is : https://google.com while example of URI is :urn:isbn:0-486-27557-4.

Domain Name System

- DNS provides a mapping from names to resources of several types.
- Domain name is a way to identify and locate computers connected to internet
- Each domain name has a corresponding IP address
- When the user types the domain name in the address bar, the corresponding IP address is supplied. Such a translation is possible with the help of system called DNS (DOMAIN NAME SYSTEM)
- No two organizations can have same domain name

Contd…

- Once a domain has been established sub-domains can be created within the domain
- EXAMPLE: The domain for the large company could be “Vni.com” and within this domain subdomains can be created for each of the company’s regional office.
- Eg: Bombay.vni.com

Top level domains

- Top level domains are classified into 2 categories:

  1. Organizational or generic domains

  2. Geographical or country domains

Organizational or generic domains

- It consists of three character code which indicates the primary function of the organization or their generic behaviour
- Most commonly used top level domains are:

✓.com for commercial organization → www.yahoo.com

✓.net for networking organizations → www.zedge.net

✓ .gov for government organizations → www.newjersey.gov

✓ .edu for educational organizations → www.uducause.edu

✓ .org for non-commercial organizations → www.eklavya.org

✓ .mil for military organizations → www.dod.mil

✓int for international organizations → www.itu.int

Geographical or country domains

- It consists of two characters which represents different countries/regions all around the world
- These codes have been standardized by International Standard Organizational (ISO)
- EXAMPLE:

✓ .in India

✓ .jp Japan

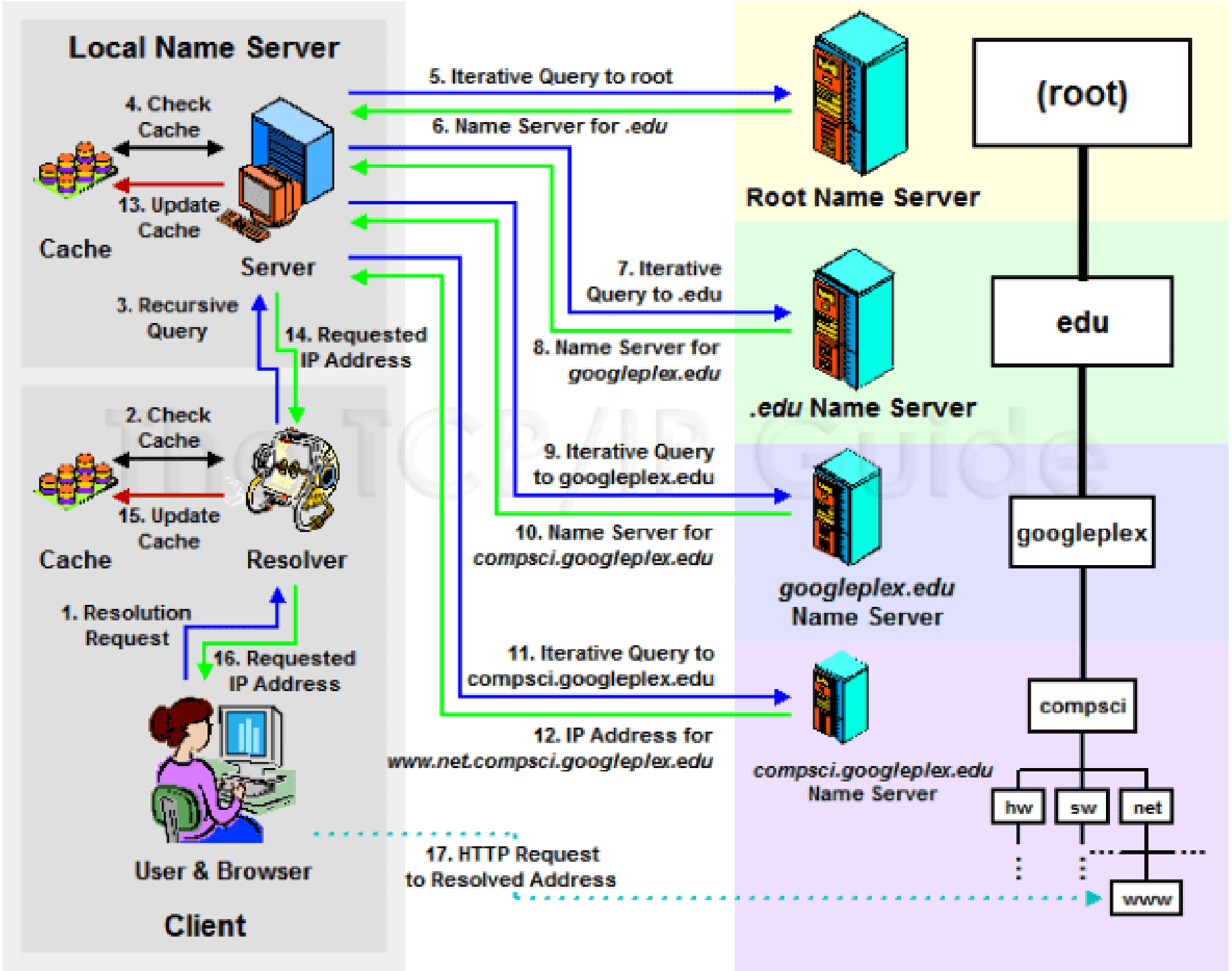
✓ .us United States

✓ .fr france

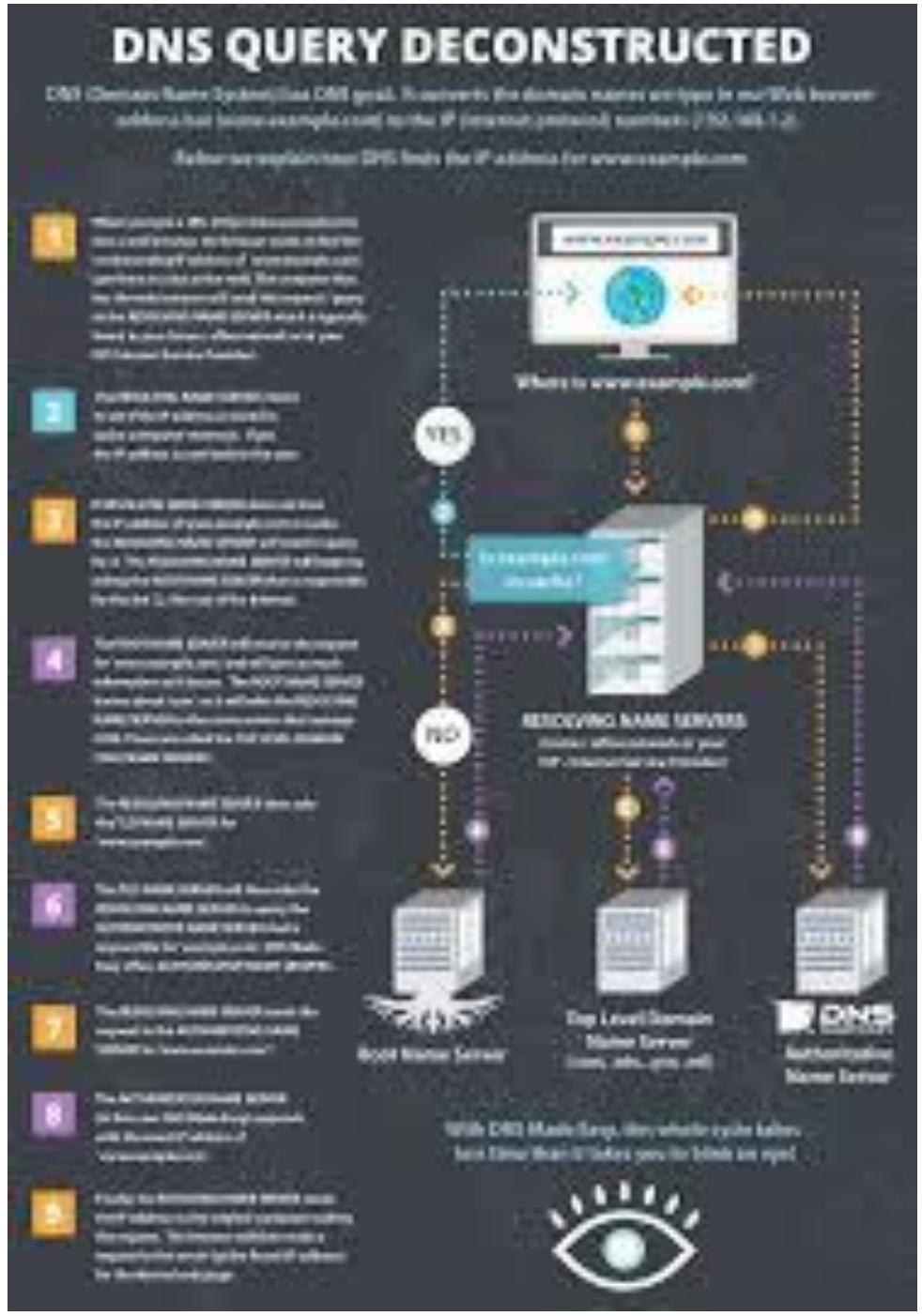
✓ .it Italy

✓ .cn China

✓ .au Australia


Local Name Server
4. Check Cache
13. Update Cache
Cache
Server
3. Recursive Query
14. Requested IP Address
2. Check Cache
15. Update Cache
Cache
Resolver
1. Resolution Request
16. Requested IP Address
User & Browser
Client
5. Iterative Query to root
6. Name Server for .edu
7. Iterative Query to .edu
8. Name Server for googleplex.edu
9. Iterative Query to googleplex.edu
10. Name Server for compsci.googleplex.edu
11. Iterative Query to compsci.googleplex.edu
12. IP Address for www.net.compsci.googleplex.edu
17. HTTP Request to Resolved Address
Root Name Server
.edu Name Server
googleplex.edu Name Server
compsci.googleplex.edu Name Server
(root)
edu
googleplex
compsci
hw
sw
net
www

DNS QUERY DECONSTRUCTED



TLS

- Transport Layer Security
- Encrypting the communication between web applications and servers.
- Designed for Internet communication to enhance privacy and data security.
- It uses client –server Handshake Mechanism

TLS Working Process

- Communication devices exchanges encryption capabilities.
- An Authentication process occurs using digital certificates.
- A session key exchange occurs

TLS...

- Advantages
- Challenges
- Difference between SSL and TLS

Introduction to XML

- **Xml** eXtensible Markup Language
- XML is designed to store and transport data.
- XML is not a replacement for HTML.
- XML is designed to be self-descriptive.
- XML is designed to carry data, not to display data.
- XML tags are not predefined. You must define your own tags.
- XML is platform independent and language independent.

Features and Advantages of XML

- XML separates data from HTML
- XML simplifies data sharing
- XML simplifies data transport
- XML simplifies Platform change
- XML increases data availability

XML Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Structure of XML

```
<root>
        <child>
         <subchild>.....</subchild>
    </child>
</root>
```

XML Syntax Rules

- XML Documents Must Have a Root Element
- The XML Prolog
- All XML Elements Must Have a Closing Tag
- XML Tags are Case Sensitive
- XML Elements Must be Properly Nested
- XML Attribute Values Must Always be Quoted
- White-space is Preserved in XML

Entity References

Entity references	**Entity**	**Meaning**
&lt;	<	less than
&gt;	>	greater than
&amp;	&	ampersand
&apos;	'	apostrophe
&quot;	"	quotation mark

XML Element Naming Styles

Style	Example	Description
Lower case	<firstname>	All letters lower case
Upper case	<FIRSTNAME>	All letters upper case
Underscore	<first_name>	Underscore separates words
Pascal case	<FirstName>	Uppercase first letter in each word
Camel case	<firstName>	Uppercase first letter in each word except the first

XML Attributes

- XML Attributes Must be Quoted
  - <person gender="female">
  - <person gender='female'>
- Some things to consider when using attributes are:
  - attributes cannot contain multiple values (elements can)
  - attributes cannot contain tree structures (elements can)

XML Elements vs. Attributes

Date as a Attribute

- <note date="2008-01-10">
  <to>Tove</to>
  <from>Jani</from>
  </note>

Date as a Element

- <note>
  <date>2008-01-10</date>
  <to>Tove</to>
  <from>Jani</from>
  </note>

- <note>
  <date>
  <year>2008</year>
  <month>01</month>
  <day>10</day>
  </date>
  <to>Tove</to>
  <from>Jani</from>
  </note>

XML DTD

- DTD stands for Document Type Definition.
- A DTD defines the structure and the legal elements and attributes of an XML document.

XML DTD

XML File

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note SYSTEM "Note.dtd">
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

Note.dtd:

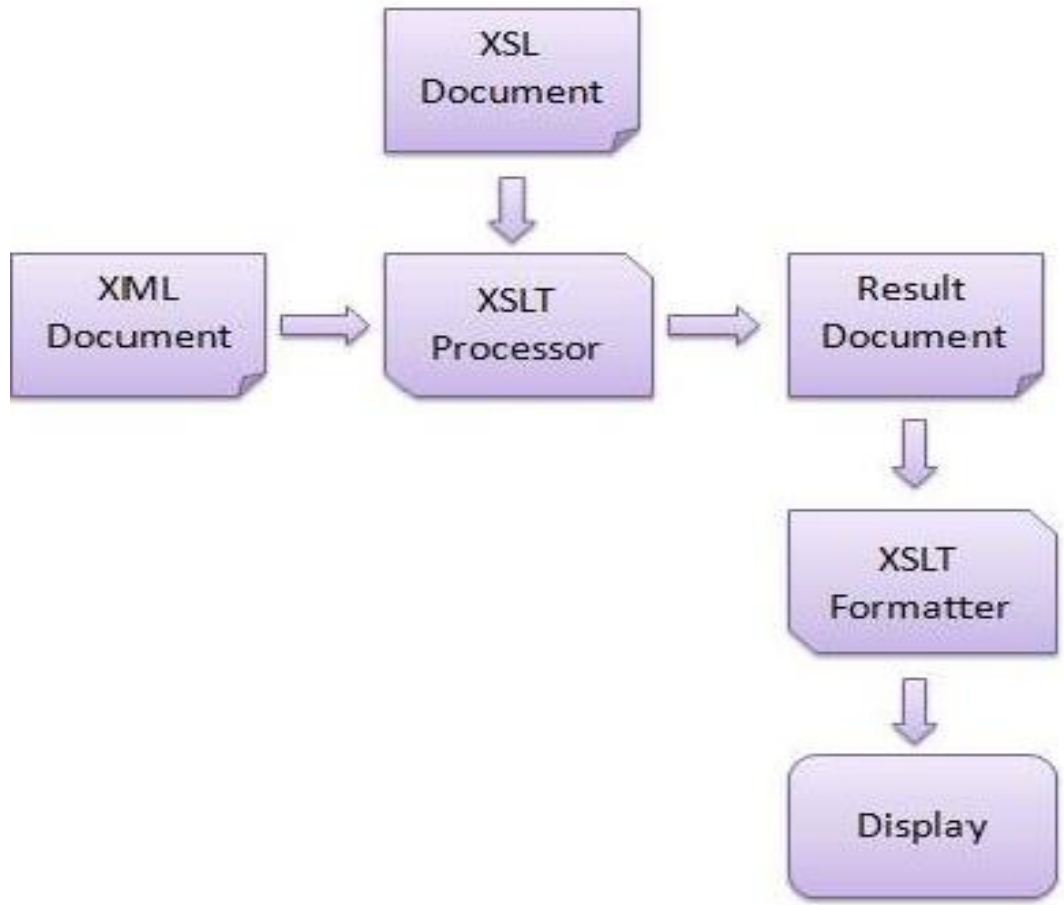
```
<!DOCTYPE note
[
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
```

Introducing XSL

- EXtensible Stylesheet Language
  - It is a styling language for XML
- XSLT stands for XSL Transformation
  - It is used to transform XML documents into other formats

Main parts of XSL Document

- XSLT
- Xpath
- Xquery
- XSL-FO


XSL
Document
XML
Document
XSLT
Processor
Result
Document
XSLT
Formatter
Display

JSON

- JSON stands for JavaScript Object Notation.
- The format was specified by Douglas Crockford.
- JSON is an open standard data-interchange format.
- JSON is lightweight and self describing.
- JSON is originated from JavaScript.
- JSON is easy to read and write.
- It has been extended from the JavaScript scripting language.

JSON Syntax Rules

- JSON syntax is derived from JavaScript object notation syntax:
  - Data is in name/value pairs
  - Data is separated by commas
  - Curly braces hold objects
  - Square brackets hold arrays

JSON Data - A Name and a Value

- JSON data is written as name/value pairs.
- A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:

"name":"John"

JSON Values

- In **JSON**, *values* must be one of the following data types:
  - a string
  - a number
  - an object (JSON object)
  - an array
  - a boolean
  - null
- JSON values **cannot** be one of the following data types:
  - a function
  - a date
  - *undefined*

JSON Strings

- Strings in JSON must be written in double quotes.
- Example

```
{ "name":"John" }
```

JSON Numbers

- Numbers in JSON must be an integer or a floating point.
- Example

```
{ "age":30 }
```

JSON Booleans

- Values in JSON can be true/false.
- Example

```
{ "sale":true }
```

JSON null

- Values in JSON can be null.
- Example

```
{ "middlename":null }
```

JSON Objects

- Values in JSON can be objects.
- Objects as values in JSON must follow the same rules as JSON objects.
- Example

```
{
  "employee":{ "name":"John", "age":30, "city":"New York“,
  “Married”:true}
}
```

Nested JSON Objects

- Example

```
myObj = {
    "name":“Yaminee",
     “lastname”:”Patil”
     "age":30,
       “Address": {
        “FlatNo“:101,
        “Wing":”A2”,
        "car":“Jazz"
       }
      }
```

JSON Arrays

- Values in JSON can be arrays.
- Example-1:

```
{
  "employees":[ "John", "Anna", "Peter" ]
}
```

- Example-2

```
{
  "employees-age":[ 32,33,45,56 ]
}
```

Arrays in JSON Objects

- Arrays can be values of an object property:
- Example

```
{
  "name":"John",
  "age":30,
  "cars":[ "Ford", "BMW", "Fiat" ]
}
```

Nested Arrays in JSON Objects

- Values in an array can also be another array, or even another JSON object:
- Example

```
myObj = {
    "name":"John",
    "age":30,
    "cars": [
     { "name":"Ford", "models":[ "Fiesta", "Focus", "Mustang" ] },
     { "name":"BMW", "models":[ "320", "X3", "X5" ] },
     { "name":"Fiat", "models":[ "500", "Panda" ] }
    ]
   }
```

JSON Array of Object

- Example:


```
{ "books": [
{ "language":"Java" , "edition":"second" },
{ "language":"C++" , "lastName":"fifth" },
{ "language":"C" , "lastName":"third" }
] }
```

Example # 1

```
{
 "firstName": "John",
 "lastName" : "doe",
 "age"      : 26,
 "address"  : {
  "streetAddress": "naist street",
  "city"         : "Nara",
  "postalCode"   : "630-0192"
 },
 "phoneNumbers": [
  {
   "type"  : "iPhone",
   "number": "0123-4567-8888"
  },
  {
   "type"  : "home",
   "number": "0123-4567-8910"
  }
 ]
}
```

Example # 2

```
{
    "books":
            [
                {
                    "title": "Professional JavaScript",
                    "authors": [
                        "Nicholas C. Zakas"
                    ],
                    "edition": 3,
                    "year": 2011
                },
                {
                    "title": "Professional JavaScript",
                    "authors": [
                        "Nicholas C.Zakas"
                    ],
                    "edition": 2,
                    "year": 2009
                },
                {
                    "title": "Professional Ajax",
                    "authors": [
                        "Nicholas C. Zakas",
                        "Jeremy McPeak",
                        "Joe Fawcett"
                    ],
                    "edition": 2,
                    "year": 2008
                }
            ]
}
```

Example # 3

```
{ "store": {
    "book": [
     { "category": "reference",
       "author": "Nigel Rees",
       "title": "Sayings of the Century",
       "price": 8.95
     },
     { "category": "fiction",
       "author": "Evelyn Waugh",
       "title": "Sword of Honour",
       "price": 12.99
     }
],
    "bicycle": {
     "color": "red",
     "price": 19.95
    }
  }
}
```

JSON vs XML

JSON Example

```
{"employees":[
{ "firstName":"John", "lastName":"Doe" }
{ "firstName":"Anna", "lastName":"Das" }
{ "firstName":"Peter", "lastName":"Joe" }
]}
```

XML Example

```
<employees>
  <employee>
    <firstName>John</firstName>
  <lastName>Doe</lastName>
  </employee>
  <employee>
    <firstName>Anna</firstName>
  <lastName>Das</lastName>
   </employee>
   <employee>
    <firstName>Peter</firstName>
  <lastName>Joe</lastName>
   </employee>
</employees>
```

JSON Schema

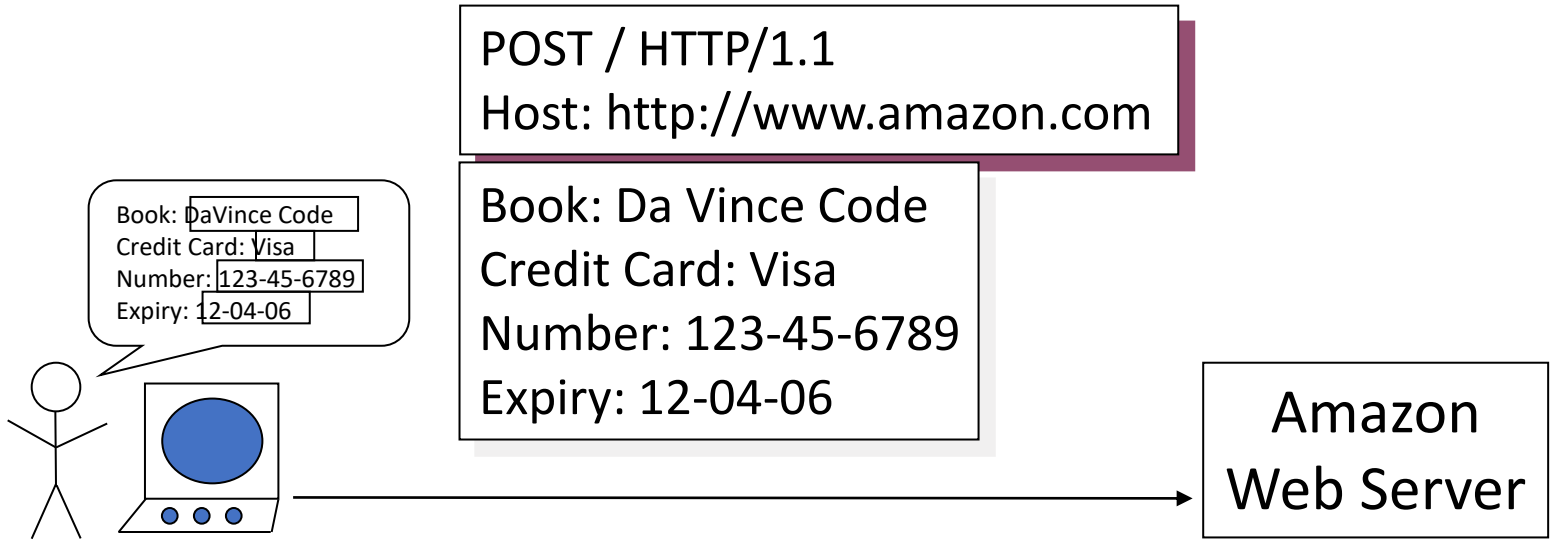
- Describes your existing data format.
- Clear, human- and machine-readable documentation.
- Complete structural validation, useful for automated testing.
- Complete structural validation, validating client-submitted data.

Keyword	Description
$schema	The $schema keyword states that this schema is written according to the draft v4 specification.
title	You will use this to give a title to your schema.
description	A little description of the schema.
type	The type keyword defines the first constraint on our JSON data: it has to be a JSON Object.
properties	Defines various keys and their value types, minimum and maximum values to be used in JSON file.
required	This keeps a list of required properties.
minimum	represents minimum acceptable value.
maximum	represents maximum acceptable value.

Web Basics: Retrieving Information using HTTP GET

- The user types in at his browser: http://www.amazon.com
- The browser software creates an HTTP header (no payload)
    - The HTTP header identifies:
        - The desired action: GET ("get me resource")
        - The target machine (www.amazon.com)

Web Basics: Updating Information using HTTP POST



- The user fills in the Web page's form
- The browser software creates an HTTP header with a payload comprised of the form data
    - The HTTP header identifies:
        - The desired action: POST ("here's some update info")
        - The target machine (amazon.com)
    - The payload contains:
        - The data being POSTed (the form data)

Terminology: Header and Payload

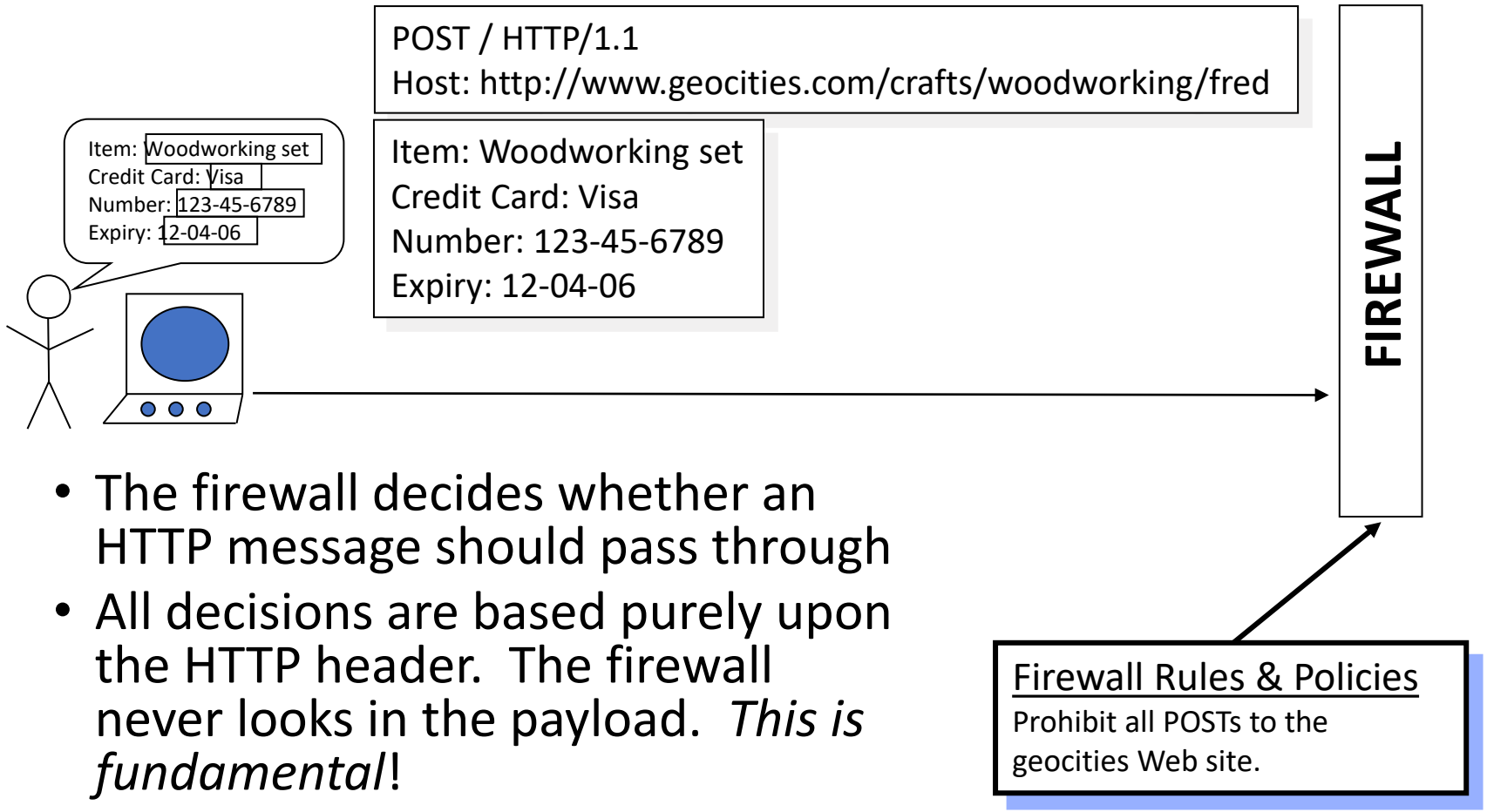
Web Basics: Simple Set of Operations, via the HTTP API

- HTTP provides a simple set of operations. Amazingly, all Web exchanges are done using this simple HTTP API:
  - GET = "give me some info" (Retrieve)
  - POST = "here's some update info" (Update)
  - PUT = "here's some new info" (Create)
  - DELETE = "delete some info" (Delete)
- The HTTP API is CRUD (Create, Retrieve, Update, and Delete)

Fundamental Web Components

Web Component: Firewalls

(Firewall: "Should I allow POSTing to geocities?")

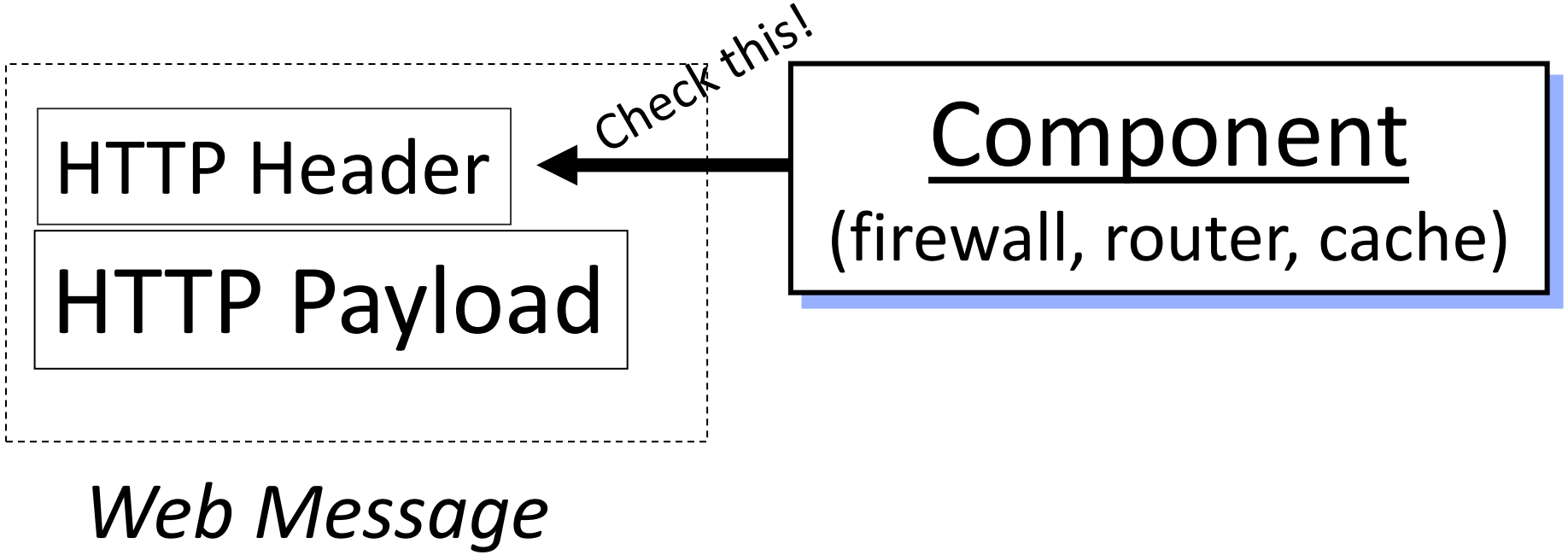


- The firewall decides whether an HTTP message should pass through
- All decisions are based purely upon the HTTP header. The firewall never looks in the payload. *This is fundamental*!
- This message is rejected!

Basic Web Components: Firewalls, Routers, Caches

- The Web is comprised of some basic components:
  - **Firewalls**: these components decide what HTTP messages get out, and what get in.
    - These components enforce **Web security**.
  - **Routers**: these components decide where to send HTTP messages.
    - These components manage **Web scaling**.
  - **Caches**: these components decide if a saved copy can be used.
    - These components increase **Web speed**.
- All these components base their decisions and actions purely upon information in the HTTP header.

Web Components Operate using only Information found in the HTTP Header!



What is REST?

- REST is a design pattern.
- It is a certain approach to creating Web Services.
- To understand the REST design pattern, let's look at an example (learn by example).

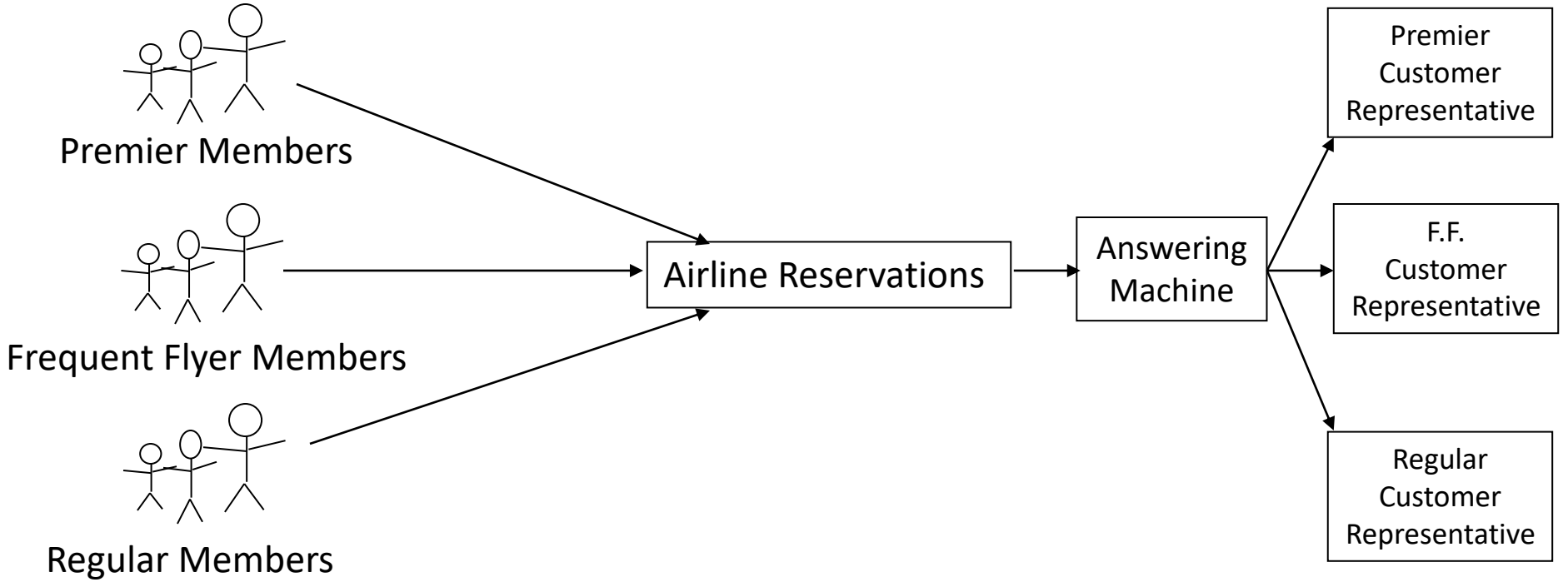
Example:
Airline Reservation Service

- Suppose that an airline wants to create a telephone reservation system for customers to call in and make flight reservations.
- The airline wants to ensure that its premier members get immediate service, its frequent flyer members get expedited service and all others get regular service.
- There are two main approaches to implementing the reservation service...

Approach 1
"Press 1 for Premier, Press 2 for..."

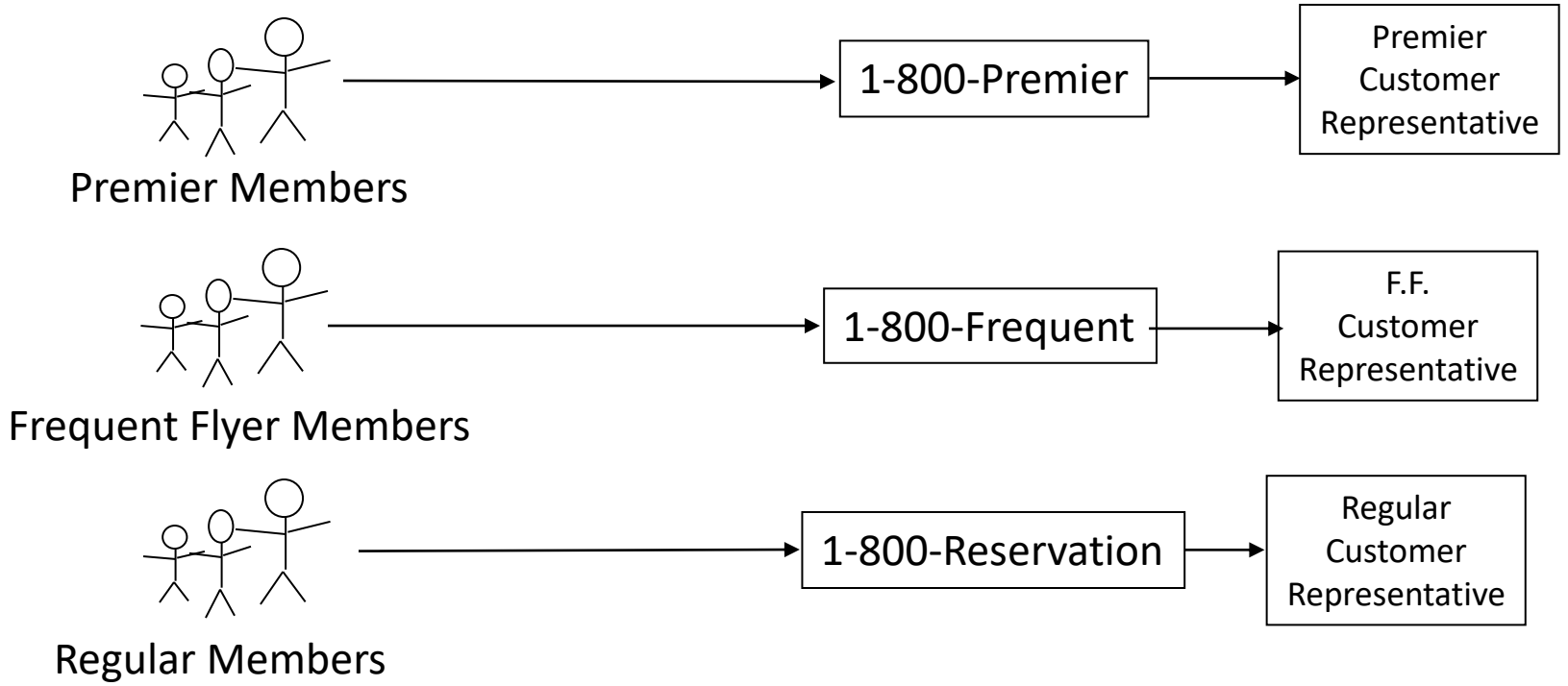
The airline provides a single telephone number.

Upon entry into the system a customer encounters an automated message, "Press 1 if you are a premier member, press 2 if you are a frequent flyer, press 3 for all others."



Approach 2
Telephone Numbers are Cheap! Use Them!

The airline provides several telephone numbers - one number for premier members, a different number for frequent flyers, and still another for regular customers.



Discussion

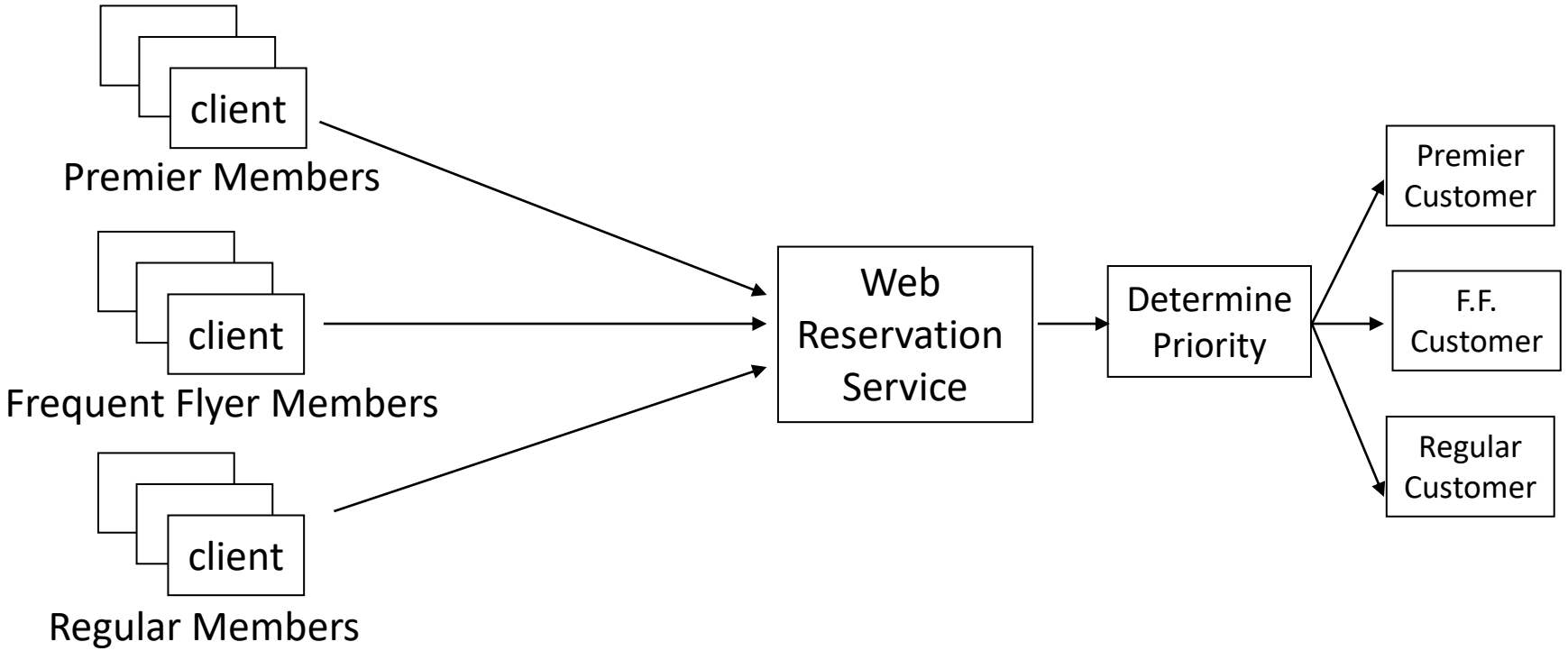
- In Approach 1 the answering machine introduces an extra delay, which is particularly annoying to premier members. (Doesn't everyone hate those answering systems)
- With Approach 2 there is no intermediate step. Premier members get instant pickup from a customer service representative. Others may have to wait for an operator.

Web-Based Reservation Service

- Suppose now the airline (kings-air.com) wants to provide a Web reservation service for customers to make flight reservations through the Web.
- Just as with the telephone service, the airline wants to ensure that its premier members get immediate service, its frequent flyer members get expedited service, all others get regular service.
- There are two main approaches to implementing the Web reservation service. The approaches are analogous to the telephone service...

Approach 1
One-Stop Shopping

The airline provides a single URL. The Web service is responsible for examining incoming client requests to determine their priority and process them accordingly.



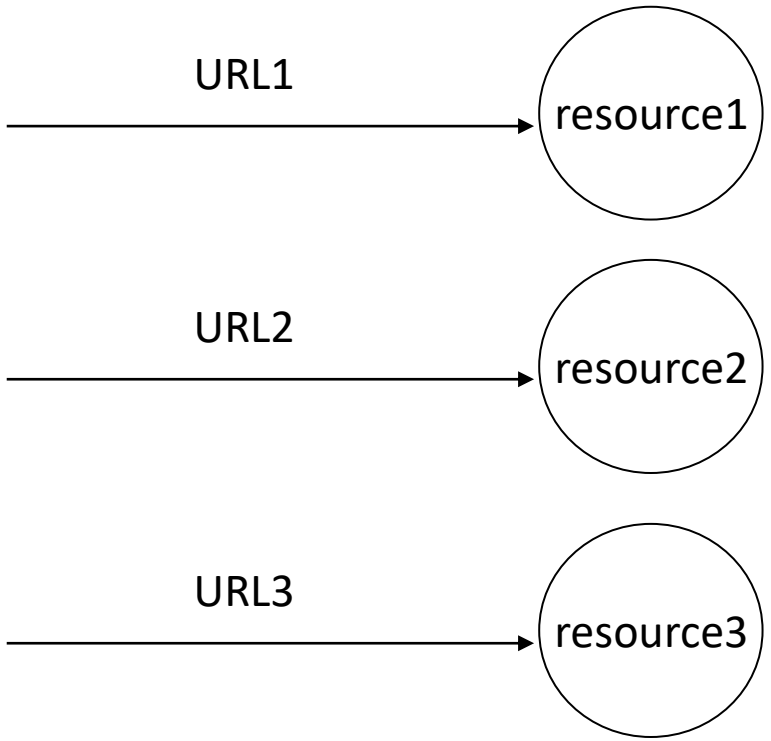
Approach 1 Disadvantages

- There is currently no industry accepted practice (rules) for expressing priorities, so rules would need to be made. The clients must learn the rule, and the Web service application must be written to understand the rule.
- This approach is based upon the incorrect assumption that a URL is "expensive" and that their use must be rationed.
- The Web service is a central point of failure. It is a bottleneck. Load balancing is a challenge.
- It violates Tim Berners-Lee Web Design, Axiom 0 (see next slide).

Web Design, Axiom 0

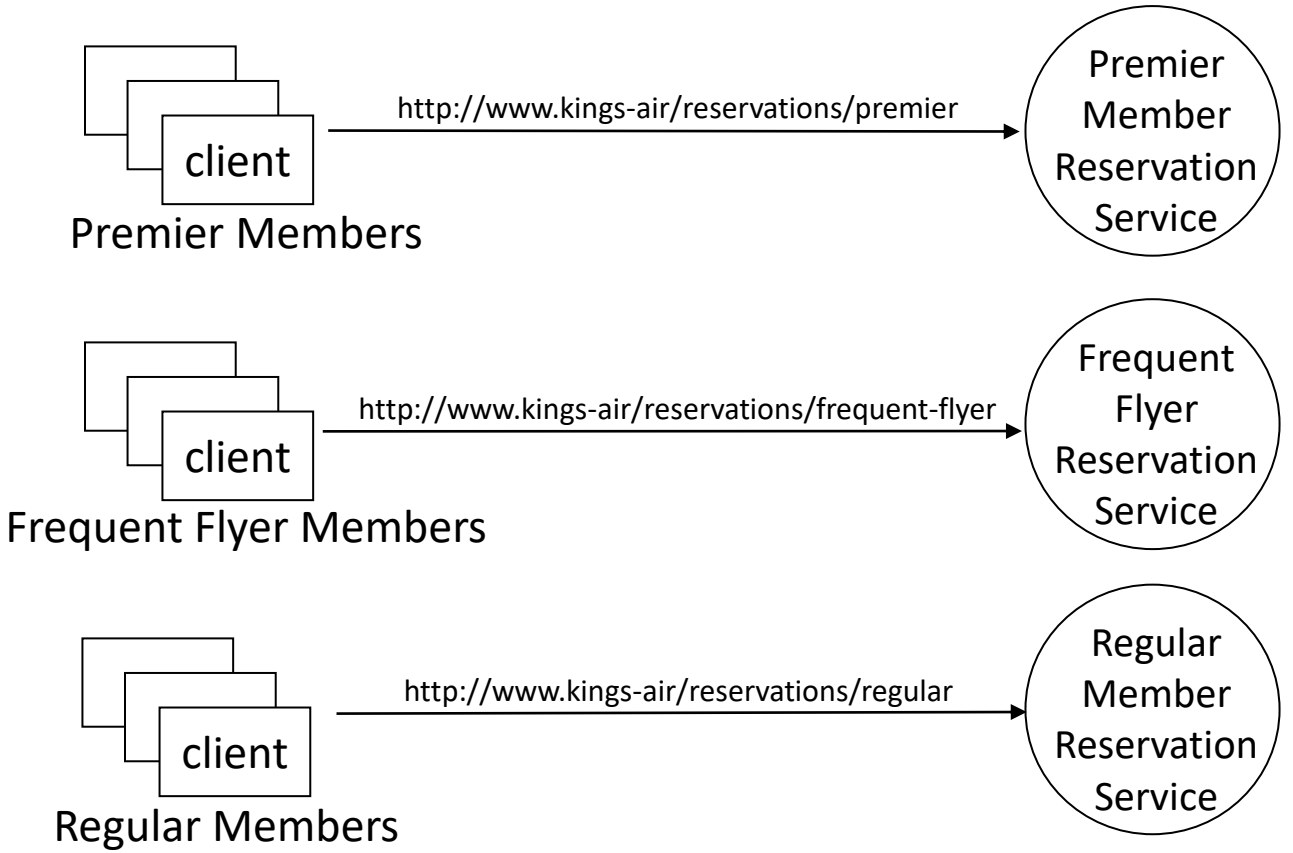
(Tim Berners-Lee, director of W3C)

- Axiom 0: all resources on the Web must be uniquely identified with a URI.



Approach 2:
URLs are Cheap! Use Them!

The airline provides several URLs - one URL for premier members, a different URL for frequent flyers, and still another for regular customers.



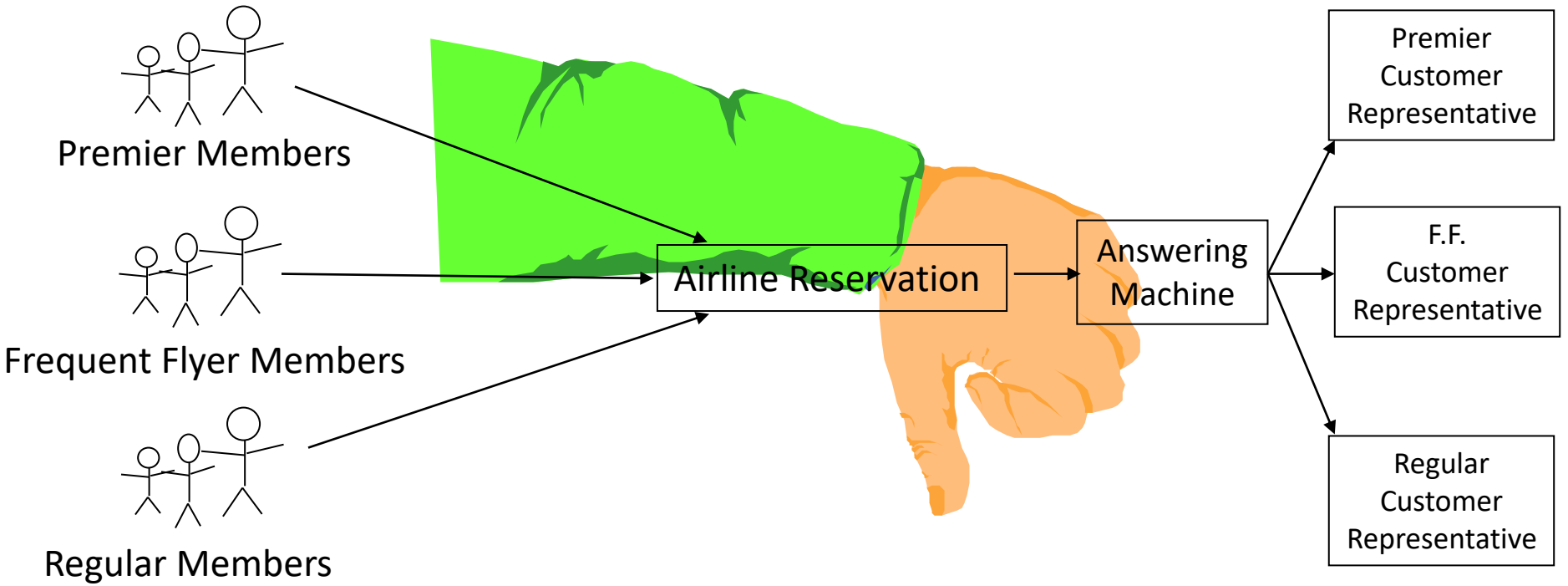
Approach 2 Advantages

- The different URLs are discoverable by search engines and UDDI registries.
- It's easy to understand what each service does simply by examining the URL, *i.e.,* it exploits the Principle of Least Surprise.
- There is no need to introduce rules. Priorities are elevated to the level of a URL. "What you see is what you get."
- It's easy to implement high priority - simply assign a fast machine at the premier member URL.
- There is no bottleneck. There is no central point of failure.
- Consistent with Axiom 0.

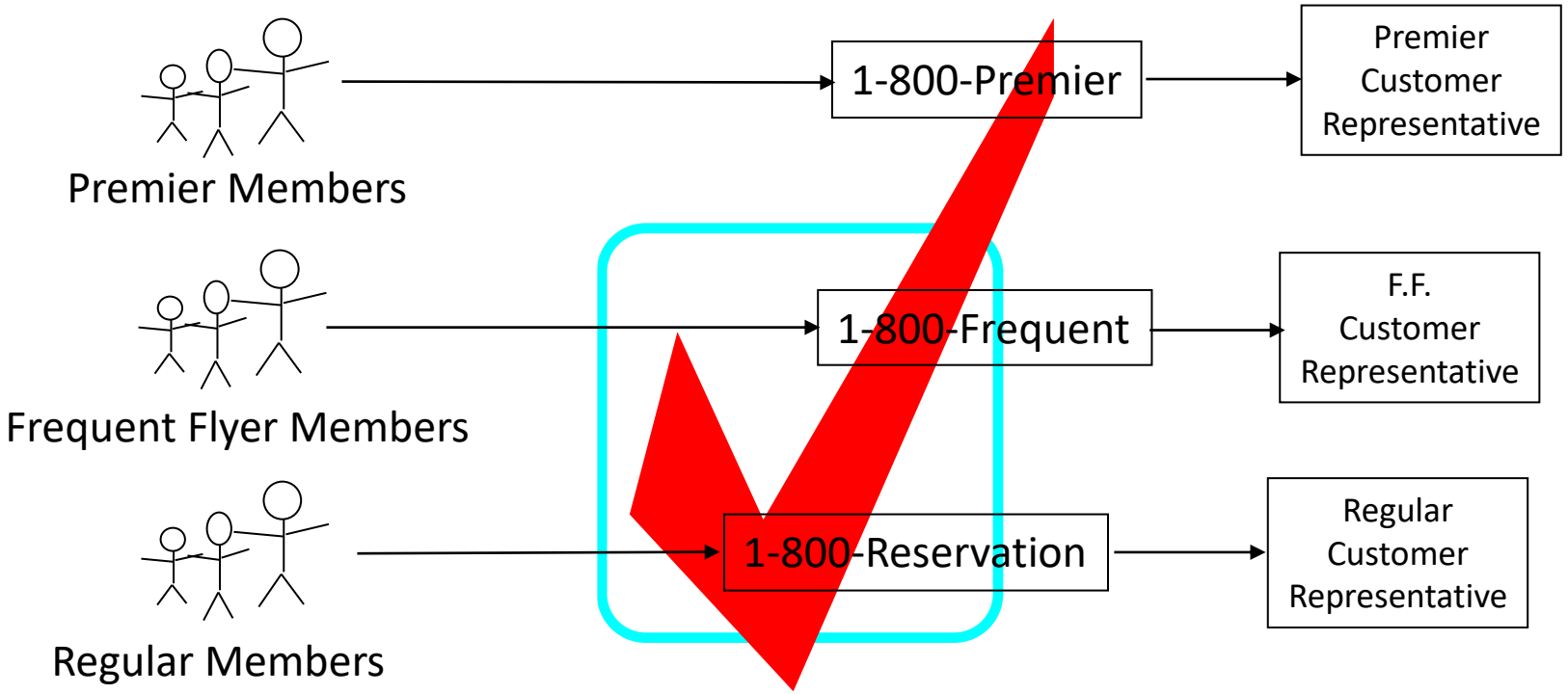
Recap

- We have looked at a reservation service.
- We have seen a telephone-based version and a Web-based version of the reservation service.
- With each version we have seen two main approaches to implementing the service.
- Which approach is the REST design pattern and which isn't? See the following slides.

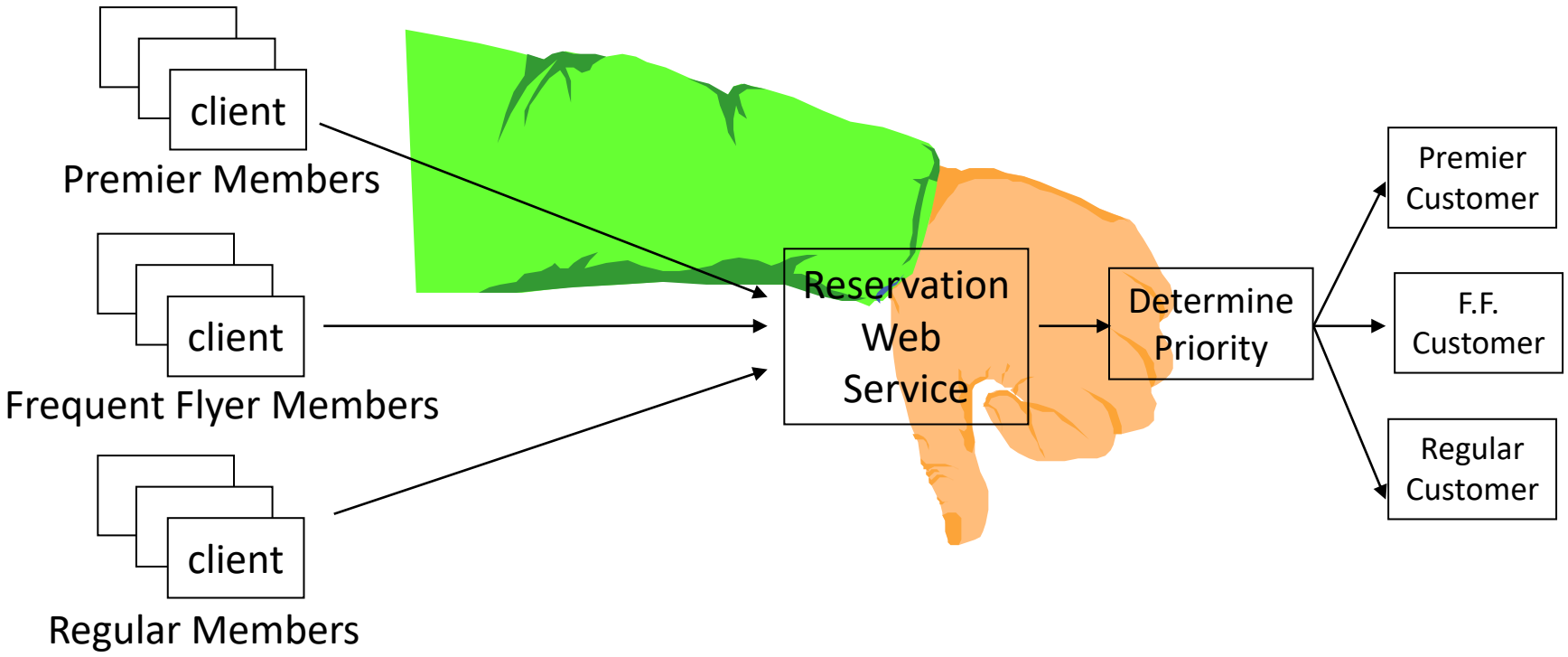
This Ain't the REST Design Pattern



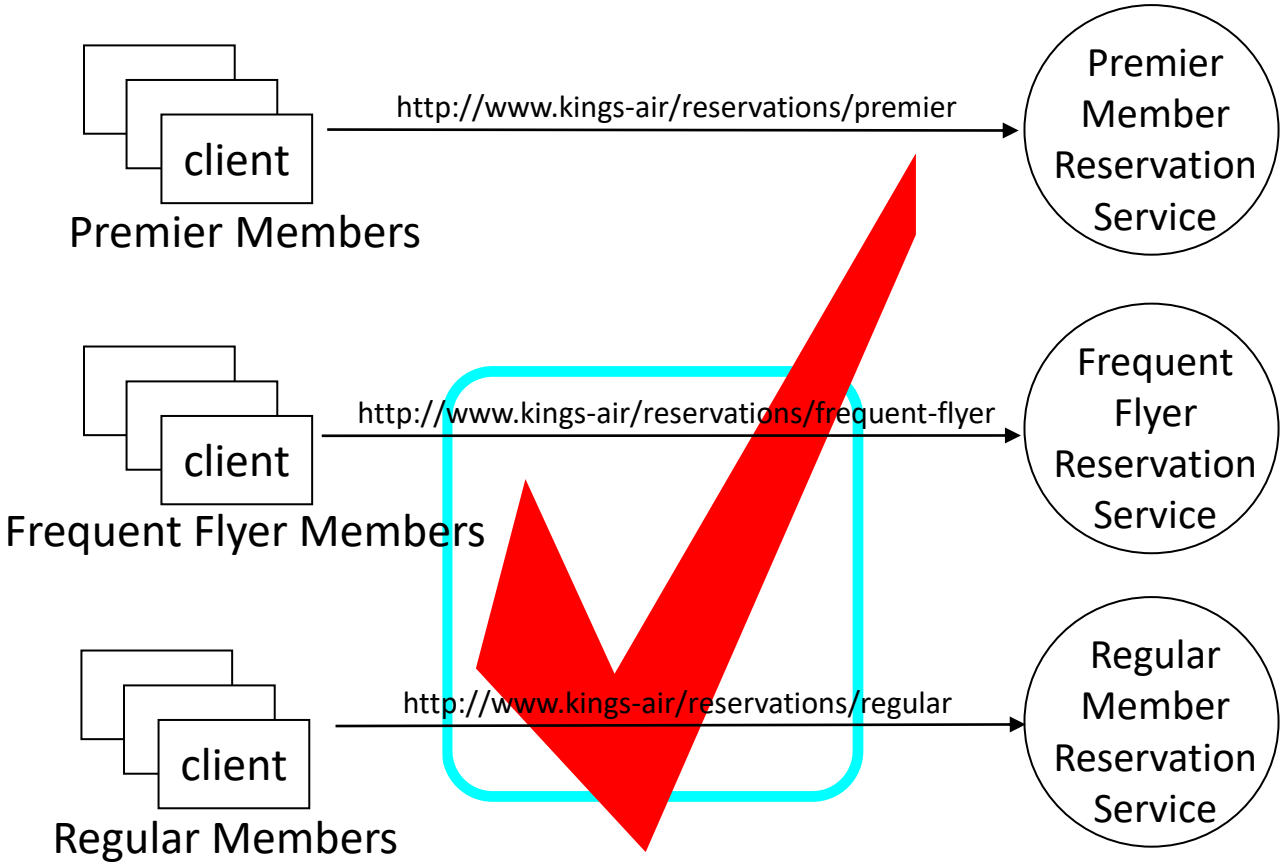
This is the REST Design Pattern



This ain't the REST Design Pattern



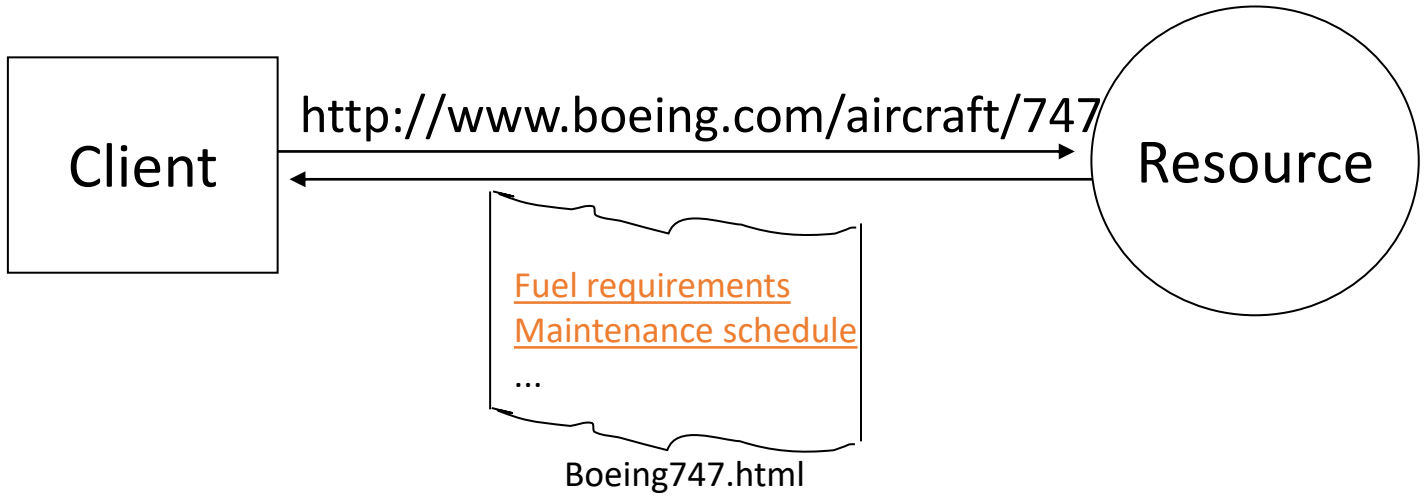
This is the REST Design Pattern



Two Fundamental Aspects of the REST Design Pattern

- **Resources**
  Every distinguishable entity is a resource. A resource may be a Web site, an HTML page, an XML document, a Web service, a physical device, *etc.*
- **URLs Identify Resources**
  Every resource is uniquely identified by a URL. This is Tim Berners-Lee Web Design, Axiom 0.

Why is it called "Representational State Transfer? "



The Client references a Web resource using a URL.
A **representation** of the resource is returned (in this case as an HTML document).
The representation (*e.g.*, Boeing747.html) places the client in a new **state**.
When the client selects a hyperlink in Boeing747.html, it accesses another resource.
The new representation places the client application into yet another state.
Thus, the client application **transfers** state with each resource representation.

Representational State Transfer

"Representational State Transfer is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use."

- Roy Fielding

Motivation for REST

The motivation for developing REST was to create a design pattern for how the Web should work, such that it could serve as the guiding framework for the Web standards and designing Web services.

REST - Not a Standard

- REST is not a standard
  - You will not see the W3C putting out a REST specification.
  - You will not see IBM or Microsoft or Sun selling a REST developer's toolkit.
- REST is just a **design pattern**
  - You can't bottle up a pattern.
  - You can only understand it and design your Web services to it.
- REST does prescribe the **use** of standards:
  - HTTP
  - URL
  - XML/HTML/GIF/JPEG/*etc.* **(Resource Representations)**
  - text/xml, text/html, image/gif, image/jpeg, *etc.* (Resource Types, MIME Types)

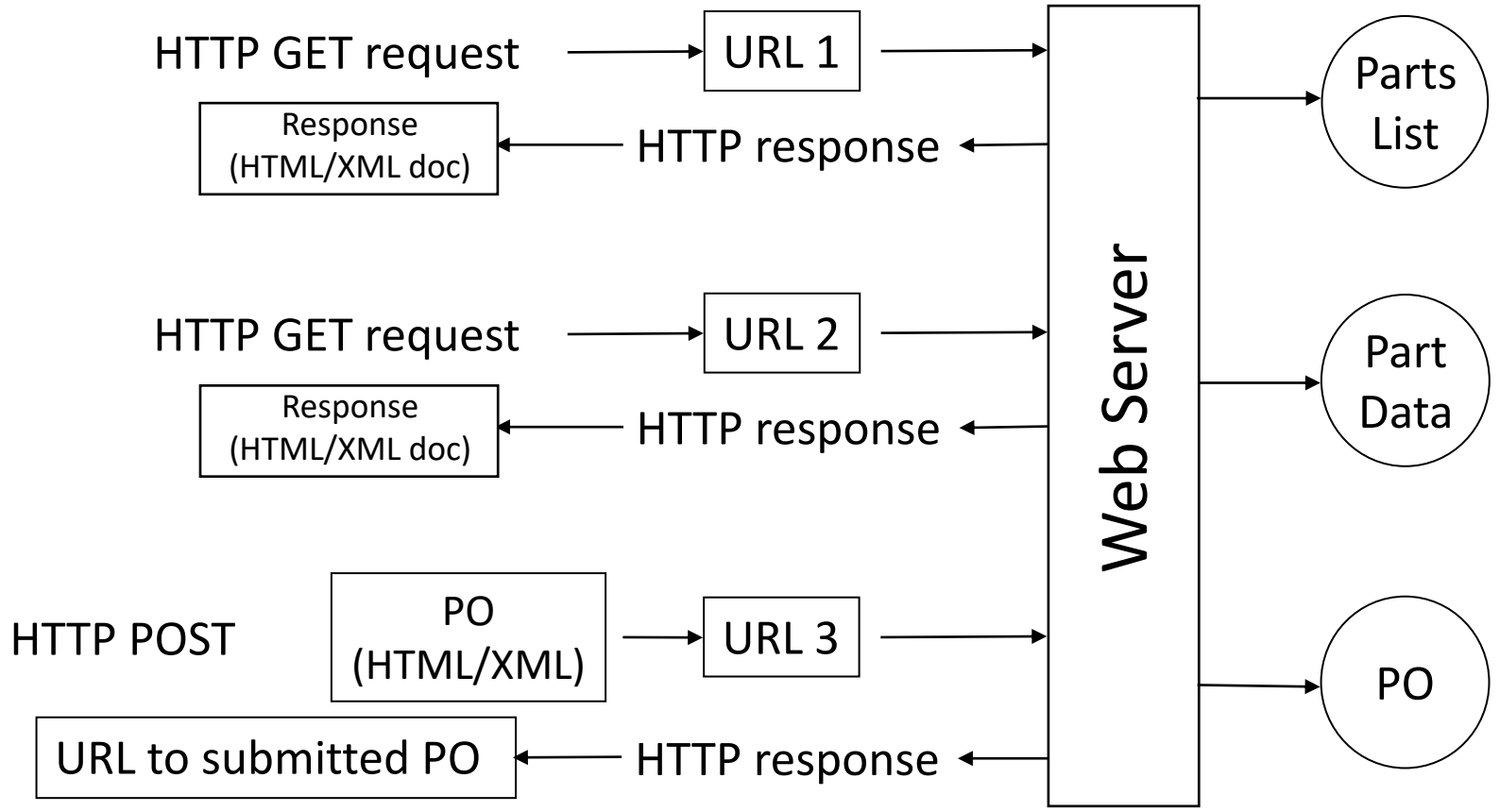
Learn by Example

- The REST design pattern is best explained with an example.
- I will present an example of a company deploying three Web services using the REST design pattern.

Parts Depot Web Services

- Parts Depot, Inc has deployed some web services to enable its customers to:
  - get a list of parts
  - get detailed information about a particular part
  - submit a Purchase Order (PO)

The REST way of Designing the Web Services



Service – Get parts list

The web service makes available a URL to a parts list resource

Client uses : http://www.parts-depot.com/parts

Document Client receives :

```
<?xml version="1.0"?>
<p:Parts xmlns:p="http://www.parts-depot.com" xmlns:xlink="http://www.w3.org/1999/xlink">
        <Part id="00345" xlink:href="http://www.parts-depot.com/parts/00345"/>
        <Part id="00346" xlink:href="http://www.parts-depot.com/parts/00346"/>
        <Part id="00347" xlink:href="http://www.parts-depot.com/parts/00347"/>
        <Part id="00348" xlink:href="http://www.parts-depot.com/parts/00348"/>
</p:Parts>
```

Service – Get detailed part data

The web service makes available a URL to each part resource.

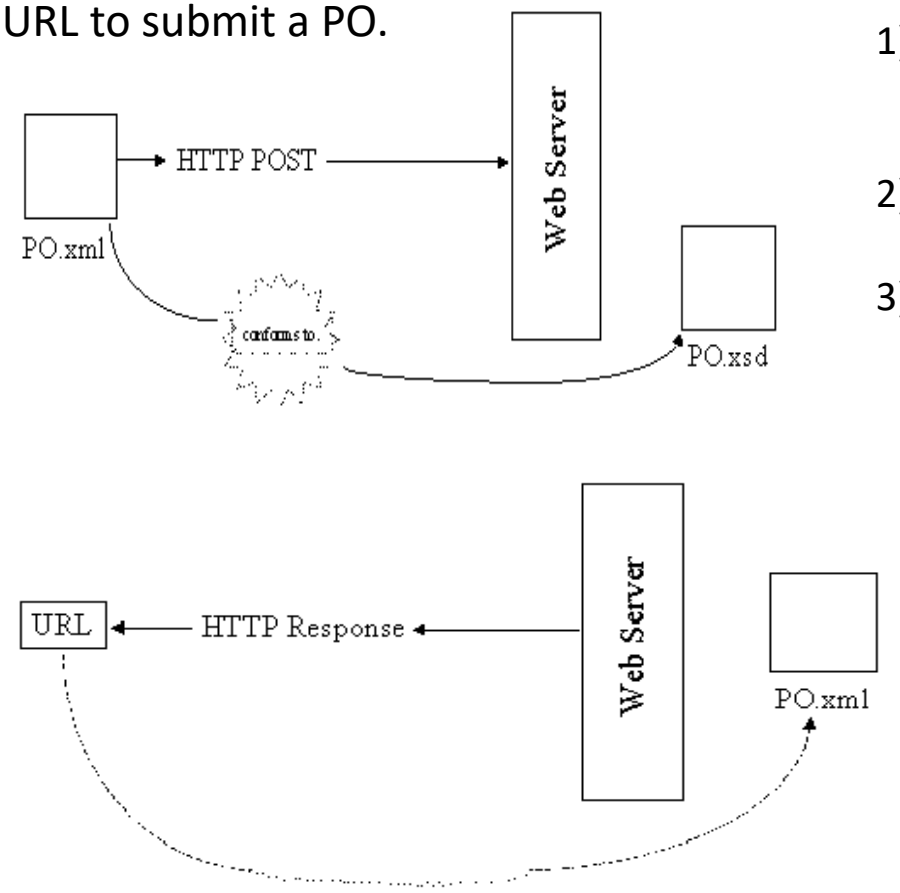
Client uses : http://www.parts-depot.com/parts/00345

Document Client receives :

```
<?xml version="1.0"?>
<p:Part xmlns:p="http://www.parts-depot.com" xmlns:xlink="http://www.w3.org/1999/xlink">
    <Part-ID>00345</Part-ID>
    <Name>Widget-A</Name>
    <Description>This part is used within the frap assembly</Description>
    <Specification xlink:href="http://www.parts-depot.com/parts/00345/specification"/> <UnitCost
  currency="USD">0.10</UnitCost>
    <Quantity>10</Quantity>
</p:Part>
```

Service – Submit purchase order (PO)

The web service makes
available a URL to submit a PO.



1)The client creates a PO instance document (PO.xml)

2)Submits the PO.xml(HTTP POST)

3)PO service reponds with a URL to the submitted PO.

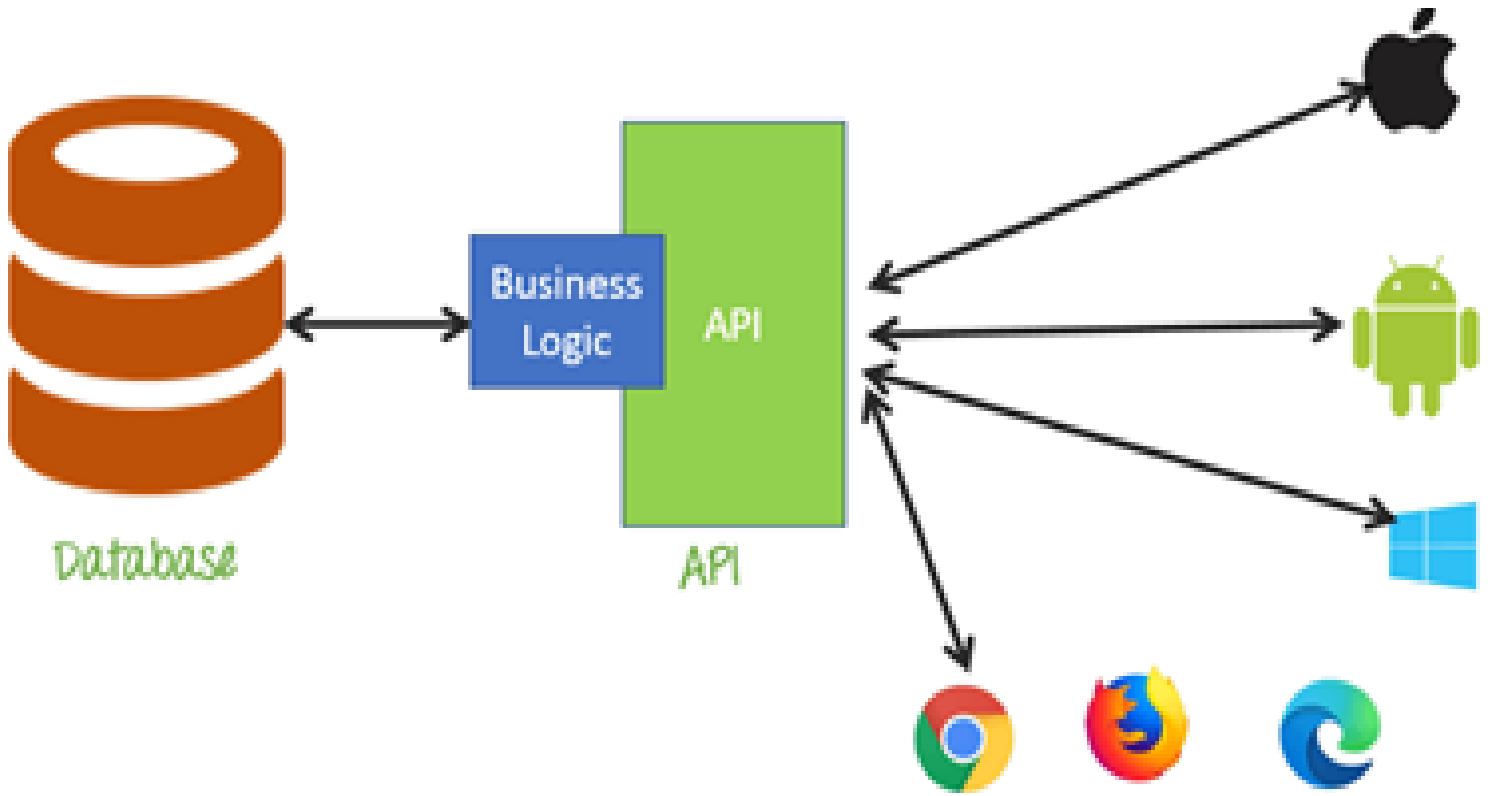
Characteristics of a REST based network

- Client-Server: a pull-based interaction style(Client request data from servers as and when needed).
- Stateless: each request from client to server must contain all the information necessary to understand the request, and cannot take advantage of any stored context on the server.
- Cache: to improve network efficiency, responses must be capable of being labeled as cacheable or non-cacheable.
- Uniform interface: all resources are accessed with a generic interface (e.g., HTTP GET, POST, PUT, DELETE).
- Named resources - the system is comprised of resources which are named using a URL.
- Interconnected resource representations - the representations of the resources are interconnected using URLs, thereby enabling a client to progress from one state to another.

Principles of REST web service design

- 1.Identify all the conceptual entities that we wish to expose as services. (Examples we saw include resources such as : parts list, detailed part data, purchase order)
- 2. Create a URL to each resource.
- 3. Categorize our resources according to whether clients can just receive a representation of the resource (using an HTTP GET), or whether clients can modify (add to) the resource using HTTP POST, PUT, and/or DELETE).
- 4. All resources accessible via HTTP GET should be side-effect free. That is, the resource should just return a representation of the resource. Invoking the resource should not result in modifying the resource.
- 5.Put hyperlinks within resource representations to enable clients to drill down for more information, and/or to obtain related information.
- 6. Design to reveal data gradually. Don't reveal everything in a single response document. Provide hyperlinks to obtain more details.
- 7. Specify the format of response data using a schema (DTD, W3C Schema, RelaxNG, or Schematron). For those services that require a POST or PUT to it, also provide a schema to specify the format of the response.
- 8. Describe how our services are to be invoked using either a WSDL document, or simply an HTML document.

API



Features of API

- It offers a valuable service (data, function, audience,.).
- It helps you to plan a business model.
- Simple, flexible, quickly adopted.
- Managed and measured.
- Offers great developer support.

REST API

- RESTFul Client-Server
- Stateless
- Cache
- Layered System
- Interface/Uniform Contract

Document Object Model (DOM)

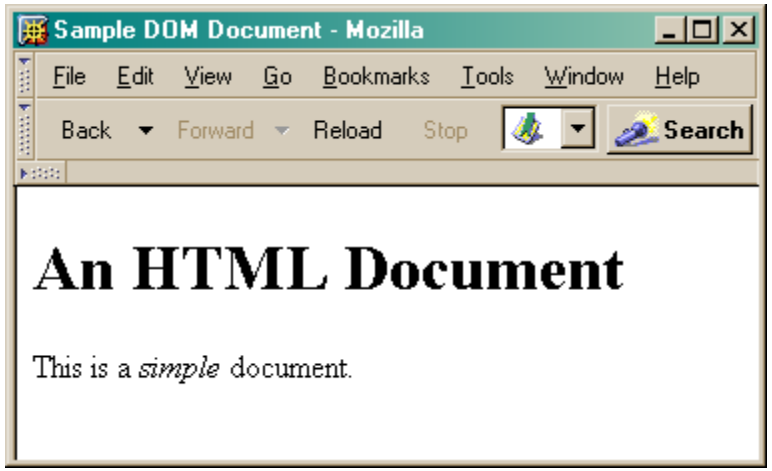
What is the DOM?

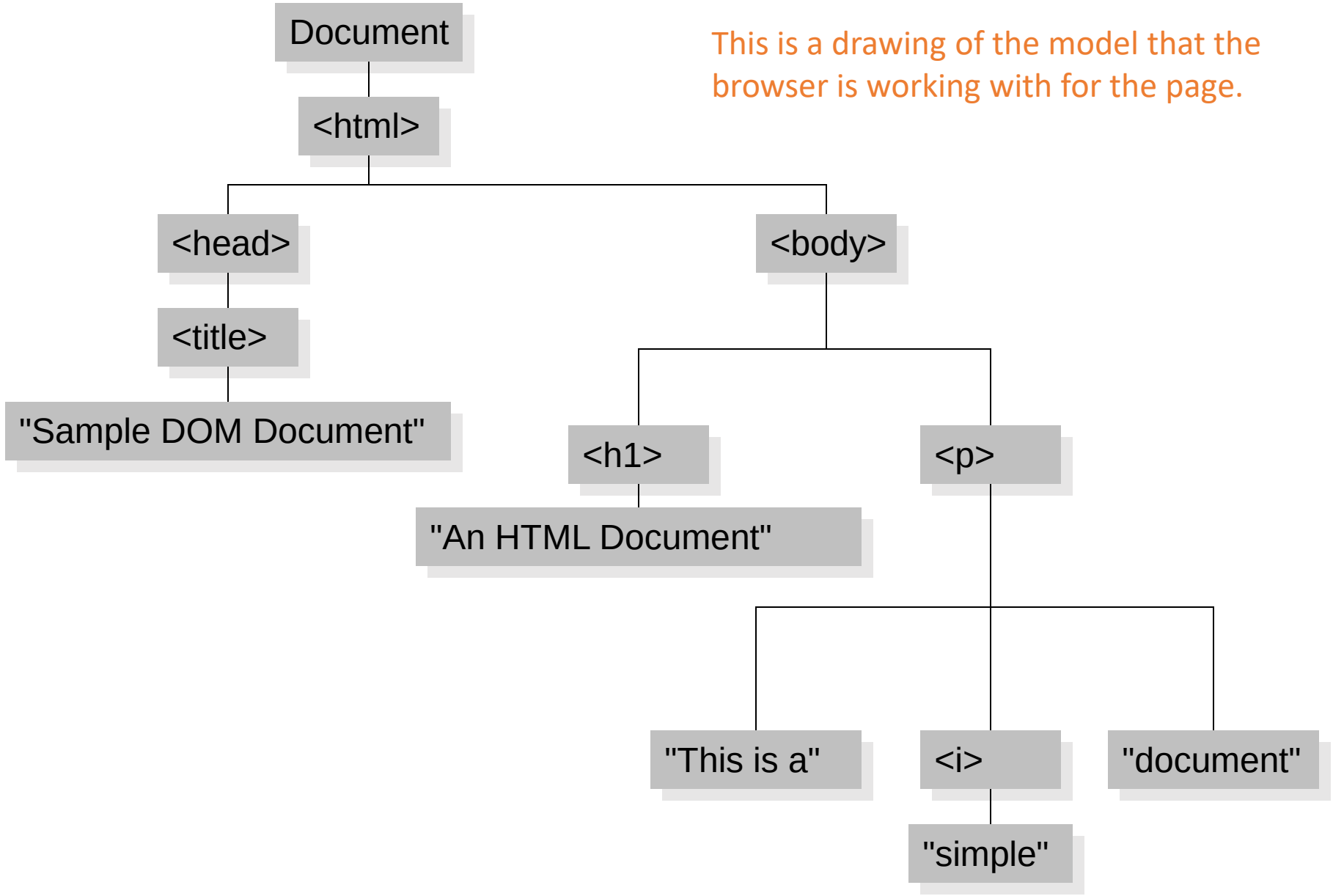
- Document Object Model
  - Your web browser builds a *model* of the web page (the *document*) that includes all the *objects* in the page (tags, text, etc)
  - All of the properties, methods, and events available to the web developer for manipulating and creating web pages are organized into objects
  - Those objects are accessible via scripting languages in modern web browsers

This is what the browser reads (sampleDOM.html).

```
<html>
  <head>
    <title>Sample DOM Document</title>
  </head>
  <body>
    <h1>An HTML Document</h1>
    <p>This is a <i>simple</i> document.
  </body>
</html>
```

This is what the browser displays on screen.





This is a drawing of the model that the browser is working with for the page.

Figure 17-1. The tree representation of an HTML document
Copied from JavaScript by Flanagan.

Why is this useful?

- Because we can access the model too!
  - the model is made available to scripts running in the browser, not just the browser itself
    - A script can find things out about the state of the page
    - A script can change things in response to events, including user requests
  - We have already used this capability in the GUI programming that we've done