REST
(Representational State Transfer)

Architecture Style

- A coordinated set of architectural constraints on the elements and their relationships
- Architecture has three elements: processing elements (Components), data elements, and connecting elements

Overview

- The Web's simple set of operations, *i.e.*, the Web's API
- The Web's fundamental components
- REST
- Questions and Answers
- The REST design pattern (summary)
- References

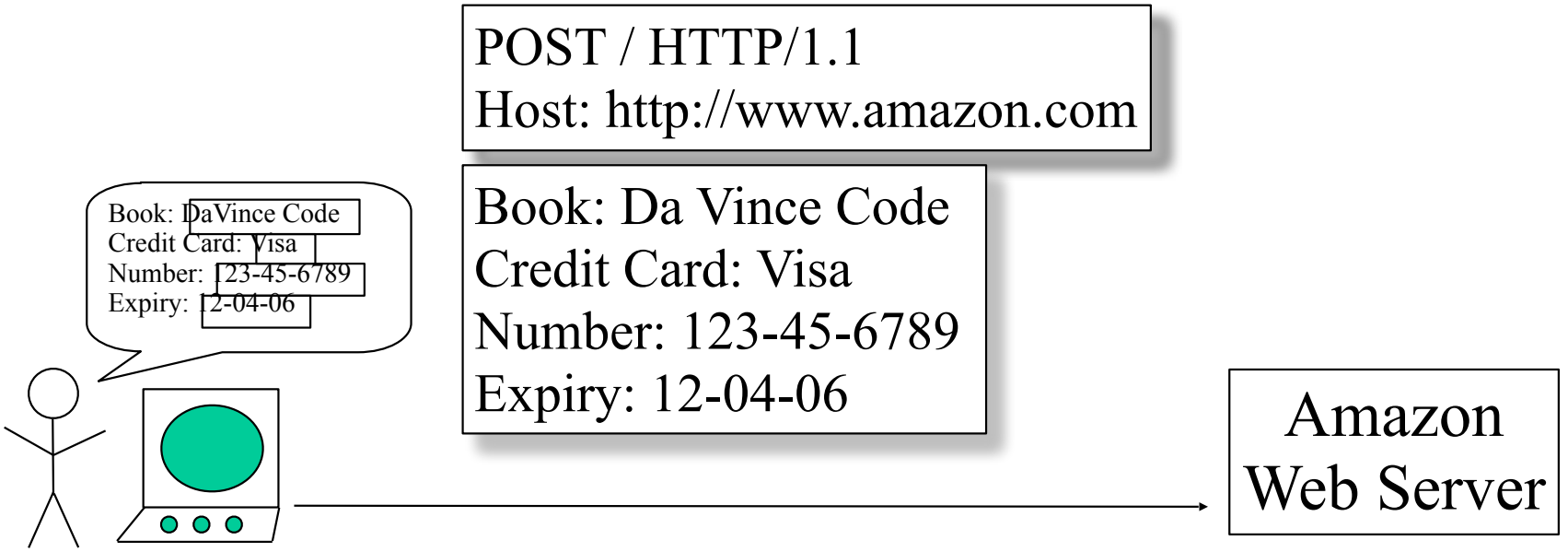
The Web's Simple Set of Operations
(*i.e.*, the Web's API*)

* Application Programming Interface

Web Basics: Retrieving Information using HTTP GET

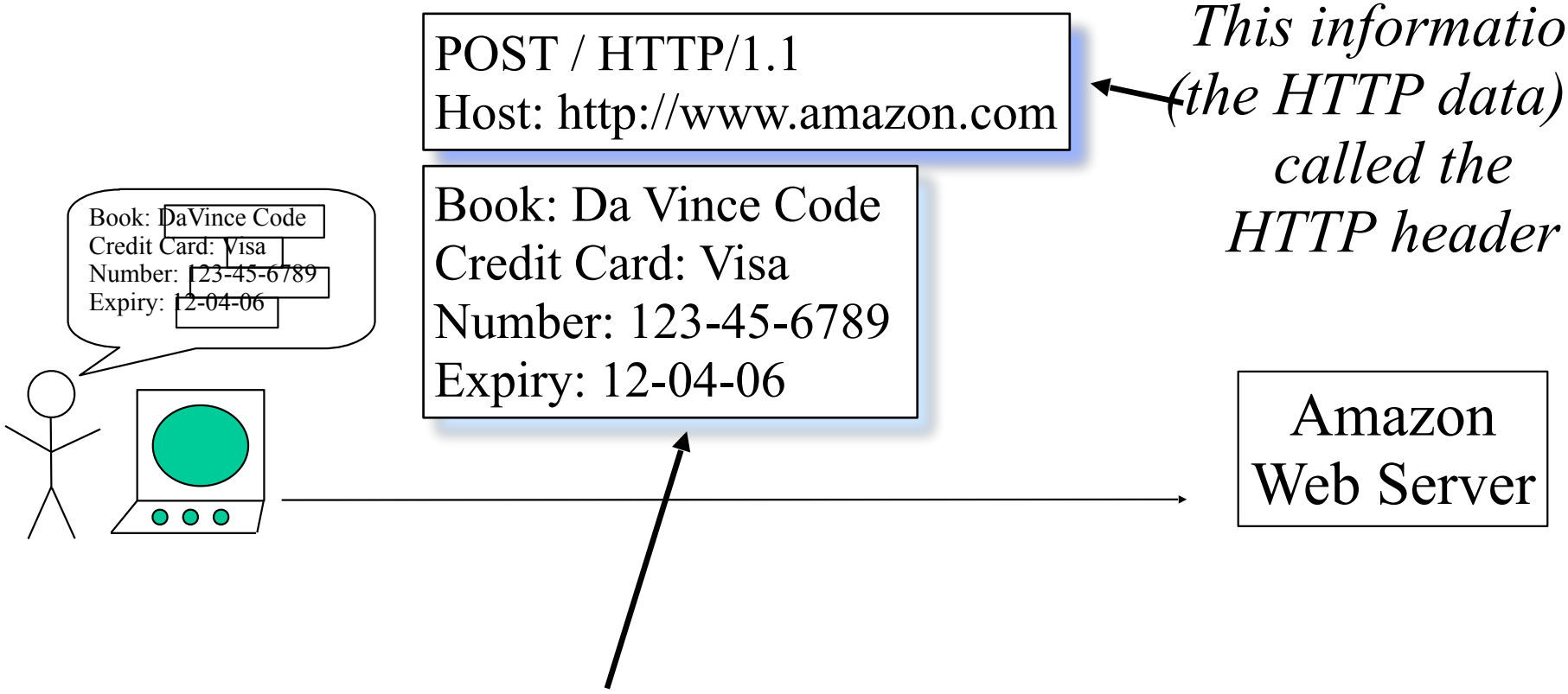
- The user types in at his browser: http://www.amazon.com
- The browser software creates an HTTP header (no payload)
    - The HTTP header identifies:
        - The desired action: GET ("get me resource")
        - The target machine (www.amazon.com)

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



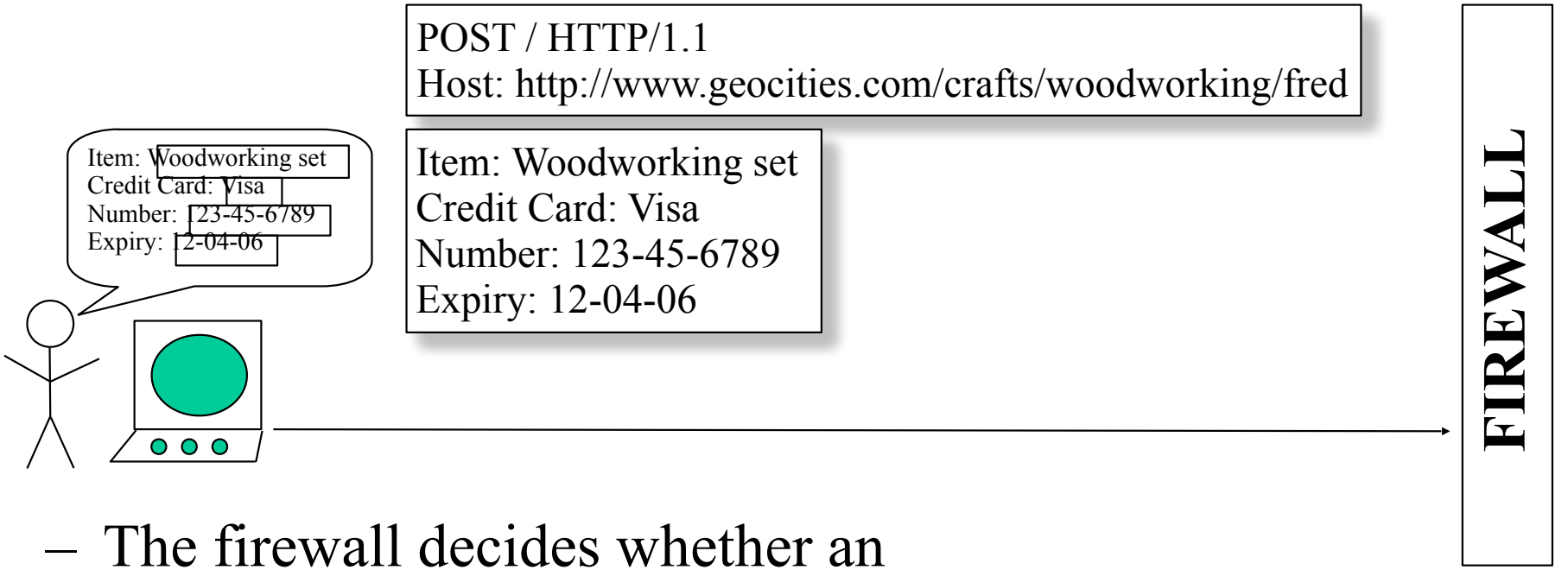
Web Basics: Simple Set of Operations, via the HTTP API

- 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?")

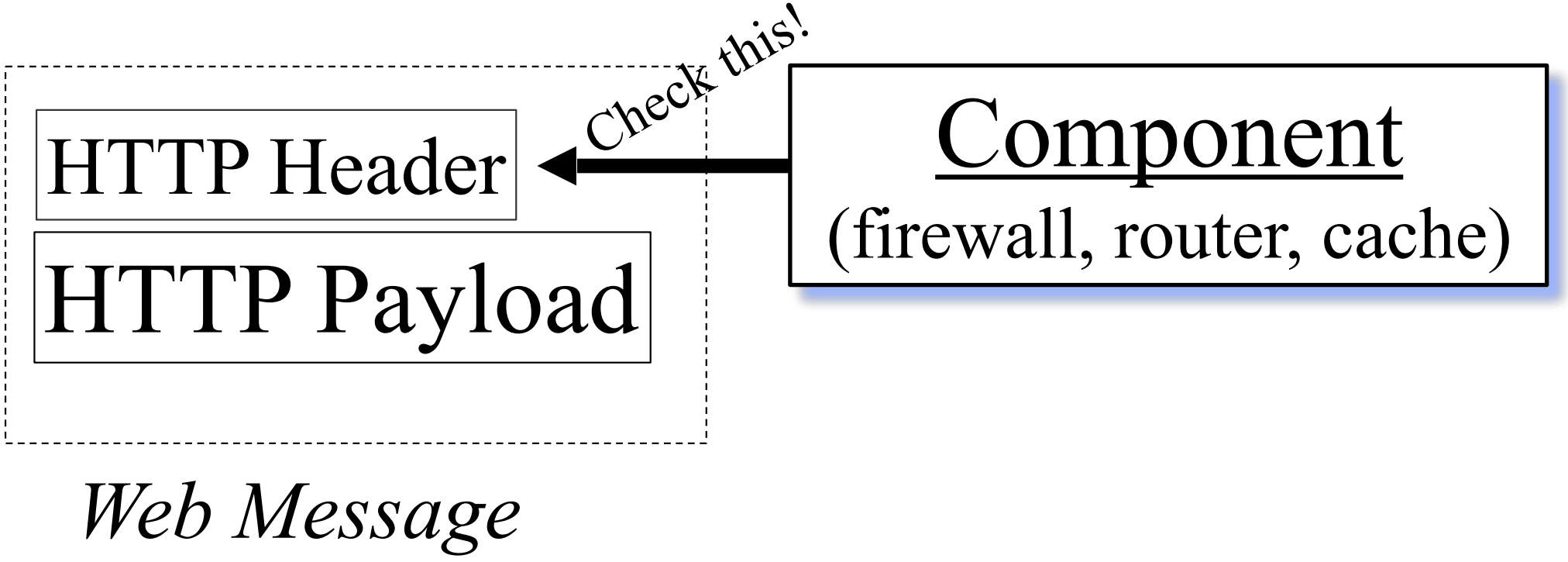


- The firewall decides whether an HTTP message should pass through
- All decisions are based purely upon the HTTP header. The firewall never looks in the payload. *This is fundamental*!
- This message is rejected!

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.
  "Thou shalt never peek inside the HTTP payload!"

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



Letter Analogy

- Why do Web components base their decisions solely on information in the HTTP header?
- My company has a receiving warehouse. All letters and packages go there first, and from there they are distributed.
- No one in the receiving warehouse may look inside any letter or package. All decisions about what to do with letters and packages must be made purely by looking at the addressing on the outside. Any attempt to peek inside of letters and packages is a violation Law.

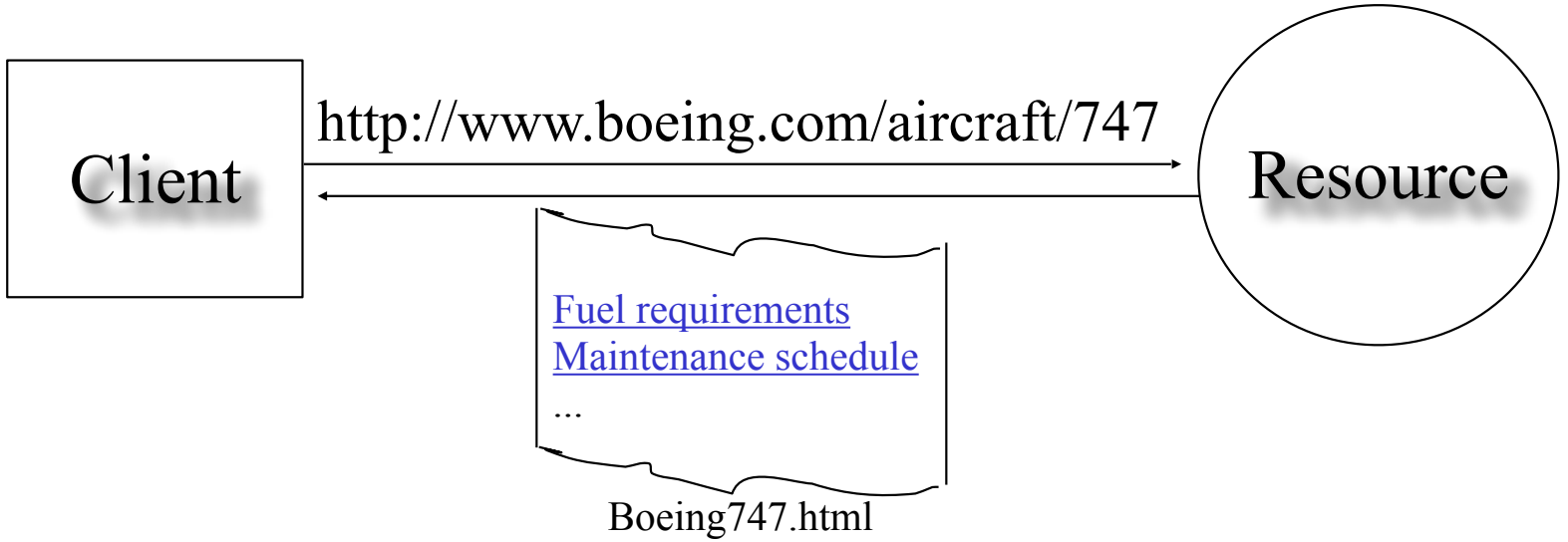
REST

What is REST?

" REST " was coined by Roy Fielding in his Ph.D. dissertation [1] to describe a design pattern for implementing networked systems.

[1] http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

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.

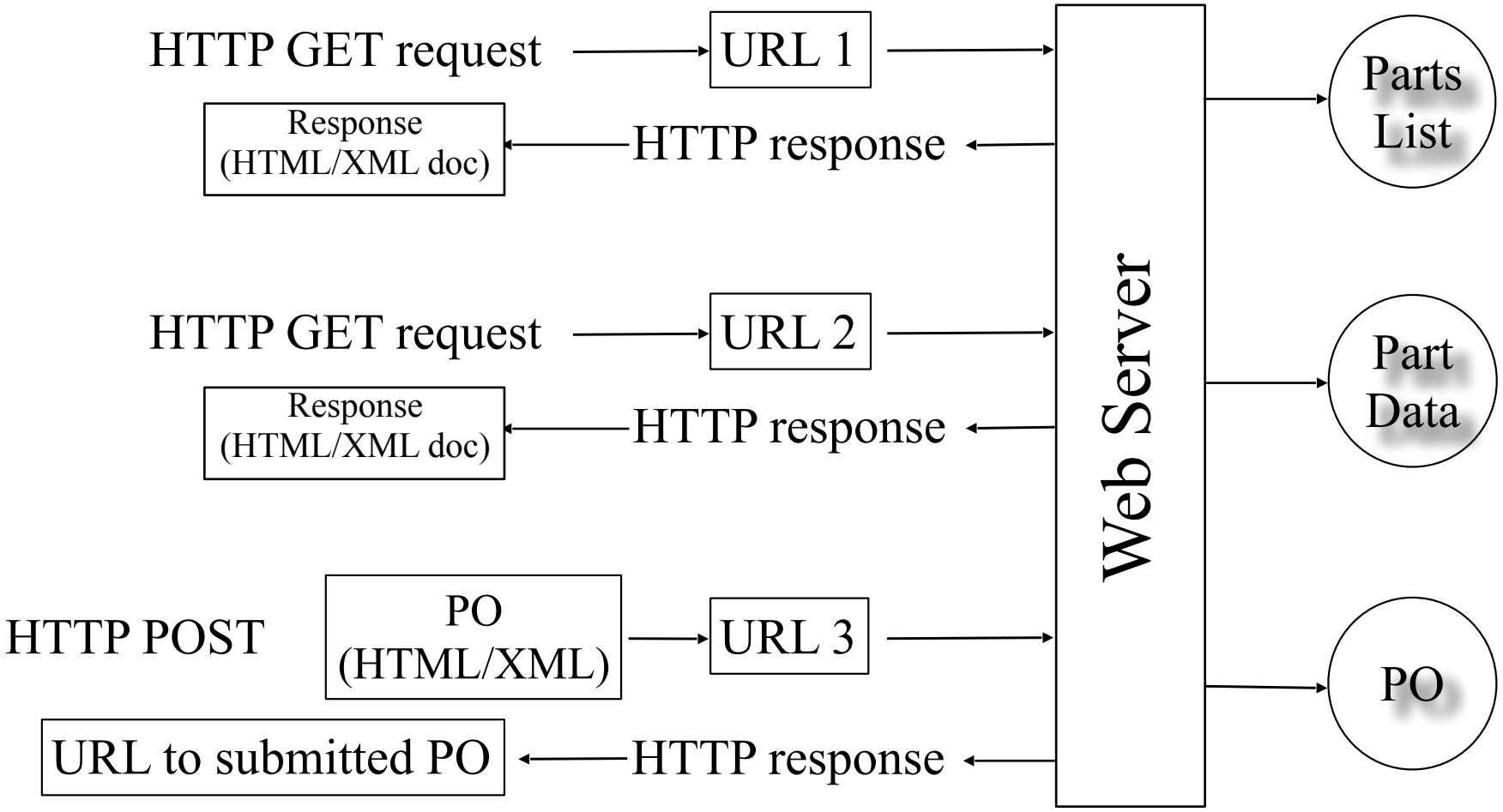
REST - Not a Standard

- 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 package 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)

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



Web Service for Clients to Retrieve a List of Parts

- Service: Get a list of parts
  - The web service makes available a URL to a parts list **resource**. A client uses this URL to get the parts list:
    - http://www.parts-depot.com/parts
    - Note that **how** the web service generates the parts list is completely transparent to the client. This is *loose coupling*.

REST Fundamentals

- Create a resource for every service.
- Identify each resource using a URL.

Data Returned - Parts List

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

Note that the parts list has links to get detailed info about each part. This is a key feature of the REST design pattern. The client transfers from one state to the next by examining and choosing from among the alternative URLs in the response document.

REST Fundamentals

- The data that a Web service returns should link to other data. Thus, design your data as a network of information.
- Contrast with OO design, which says to encapsulate information.

Web Service for Clients to Retrieve a Particular Part

- Service: Get detailed information about a particular part
  - The web service makes available a URL to each part resource. *For example*, here's how a client requests a specific part:
    - http://www.parts-depot.com/parts/00345

Data Returned - Part 00345

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

Again observe how this data is linked to still more data - the specification for this part may be found by traversing the hyperlink. Each response document allows the client to drill down to get more detailed information.

Summary of the REST Design Pattern

The REST Design Pattern

- Create a resource for every service.
- Uniquely identify each resource with a logical URL.
- Design your information to link to other information. That is, the information that a resource returns to a client should link to other information in a network of related information.

The REST Design Pattern (cont.)

- All interactions between a client and a web service are done with simple operations. Most web interactions are done using HTTP and just four operations:
  - retrieve information (HTTP GET)
  - create information (HTTP PUT)
  - update information (HTTP POST)
  - delete information (HTTP DELETE)

The REST Design Pattern (cont.)

- Remember that Web components (firewalls, routers, caches) make their decisions based upon information in the HTTP Header. Consequently, the destination URL must be placed in the HTTP header for Web components to operate effectively.
  - Conversely, it is anti-REST if the HTTP header just identifies an intermediate destination and the payload identifies the final destination.