WEB TECHNOLOGY AND ITS APPLICATIONS

17CS71

Mr. GANESH D R
ASSISTANT PROFESSOR,
DEPT OF CSE, CITECH

WEB TECHNOLOGY AND ITS APPLICATIONS
[As per Choice Based Credit System (CBCS) scheme]
(Effective from the academic year 2017 - 2018)
SEMESTER – VII

Subject Code	17CS71	IA Marks	40
Number of Lecture Hours/Week	04	Exam Marks	60
Total Number of Lecture Hours	50	Exam Hours	03

CREDITS – 04

Module – 1	**Teaching Hours**
Introduction to HTML, What is HTML and Where did it come from?, HTML Syntax, Semantic Markup, Structure of HTML Documents, Quick Tour of HTML Elements, HTML5 Semantic Structure Elements, Introduction to CSS, What is CSS, CSS Syntax, Location of Styles, Selectors, The Cascade: How Styles Interact, The Box Model, CSS Text Styling.	**10 Hours**
Module – 2	
HTML Tables and Forms, Introducing Tables, Styling Tables, Introducing Forms, Form Control Elements, Table and Form Accessibility, Microformats, Advanced CSS: Layout, Normal Flow, Positioning Elements, Floating Elements, Constructing Multicolumn Layouts, Approaches to CSS Layout, Responsive Design, CSS Frameworks.	**10 Hours**
Module – 3	
JavaScript: Client-Side Scripting, What is JavaScript and What can it do?, JavaScript Design Principles, Where does JavaScript Go?, Syntax, JavaScript Objects, The Document Object Model (DOM), JavaScript Events, Forms, Introduction to Server-Side Development with PHP, What is Server-Side Development, A Web Server's Responsibilities, Quick Tour of PHP, Program Control, Functions	**10 Hours**
Module – 4	
PHP Arrays and Superglobals, Arrays, $_GET and $_POST Superglobal Arrays, $_SERVER Array, $_Files Array, Reading/Writing Files, PHP Classes and Objects, Object-Oriented Overview, Classes and Objects in PHP, Object Oriented Design, Error Handling and Validation, What are Errors and Exceptions?, PHP Error Reporting, PHP Error and Exception Handling	**10 Hours**
Module – 5	
Managing State, The Problem of State in Web Applications, Passing Information via Query Strings, Passing Information via the URL Path, Cookies, Serialization, Session State, HTML5 Web Storage, Caching, Advanced JavaScript and jQuery, JavaScript Pseudo-Classes, jQuery Foundations, AJAX, Asynchronous File Transmission, Animation, Backbone MVC Frameworks, XML Processing and Web Services, XML Processing, JSON, Overview of Web Services.	**10 Hours**

MODULE 5 - SYLLABUS

- Managing State, The Problem of State in Web Applications, Passing Information via Query Strings, Passing Information via the URL Path, Cookies, Serialization, Session State, HTML5 Web Storage, Caching, Advanced JavaScript and jQuery, JavaScript Pseudo-Classes, jQuery Foundations, AJAX, Asynchronous File Transmission, Animation, Backbone MVC Frameworks, XML Processing and Web Services, XML Processing, JSON, Overview of Web Services.

MANAGING STATE

Chapter 13

Section 1 of 8

THE PROBLEM OF STATE IN WEB APPLICATIONS

State in Web Applications

Not like a desktop application

❖Until now we have seen how to process user inputs, output information and read & write from other storage media.

❖But here we will be examining a development problem that is unique to the world of web development.

❖HOW CAN ONE REQUEST SHARE INFORMATION WITH ANOTHER REQUEST?

❖Single user desktop applications do not have this challenge at all because the program information for the user is stored in memory(or in external storage) and thus can be easily accesses through out the applications.

❖Remember Web applications differ from desktop applications

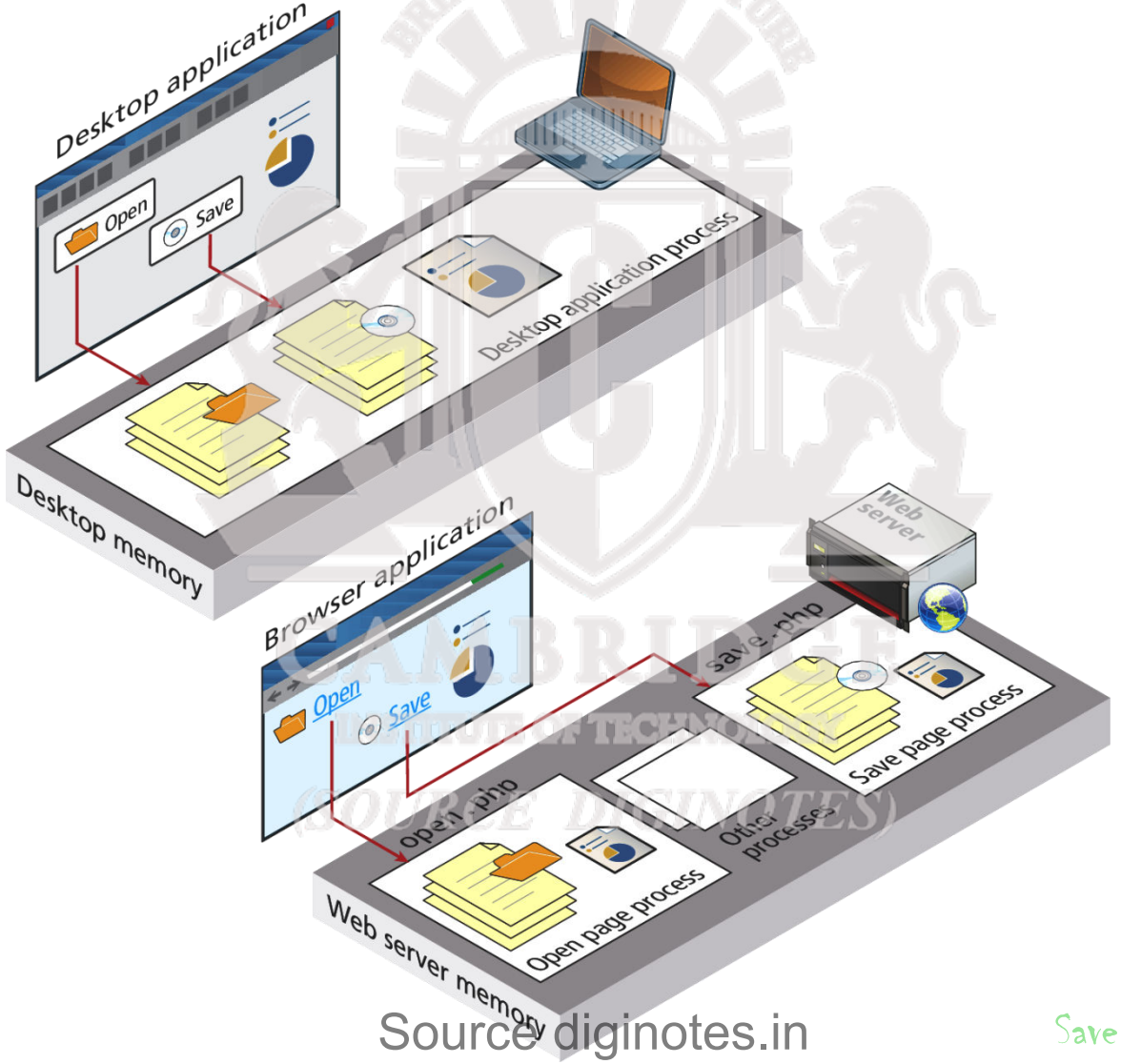
State in Web Applications

Not like a desktop application

❖Unlike the unified single process that is the typical desktop application, a web application consists of a series of disconnected HTTP requests to a web server where each request for a server page is essentially a request to run a separate program as in below fig

State in Web Applications

Not like a desktop application

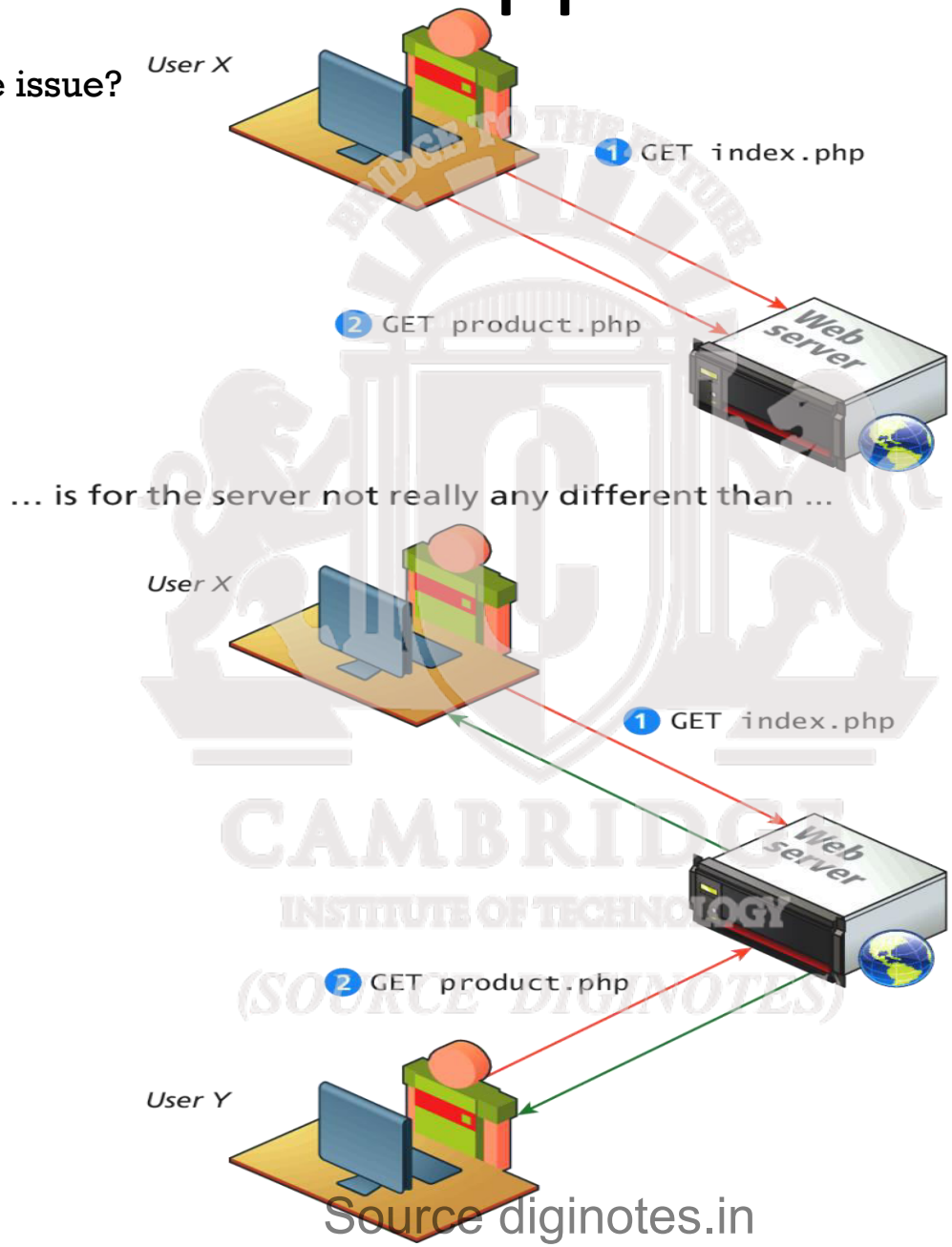


•The web server sees only requests.

•The HTTP protocol does not, without program intervention, distinguish two requests by one source from two requests from two different sources, as in below figure

State in Web Applications

What's the issue?



❖There are many occasions when we want the web server to connect requests together.

❖Consider the scenario of a web shopping cart, as in below fig.

❖In such a case, the user(website owner) most certainly wants the server to recognize that the request to add an item to the cart and the subsequent request to check out and pay for the item in the cart are connected to the same individual

State in Web Applications

What's the desired outcome

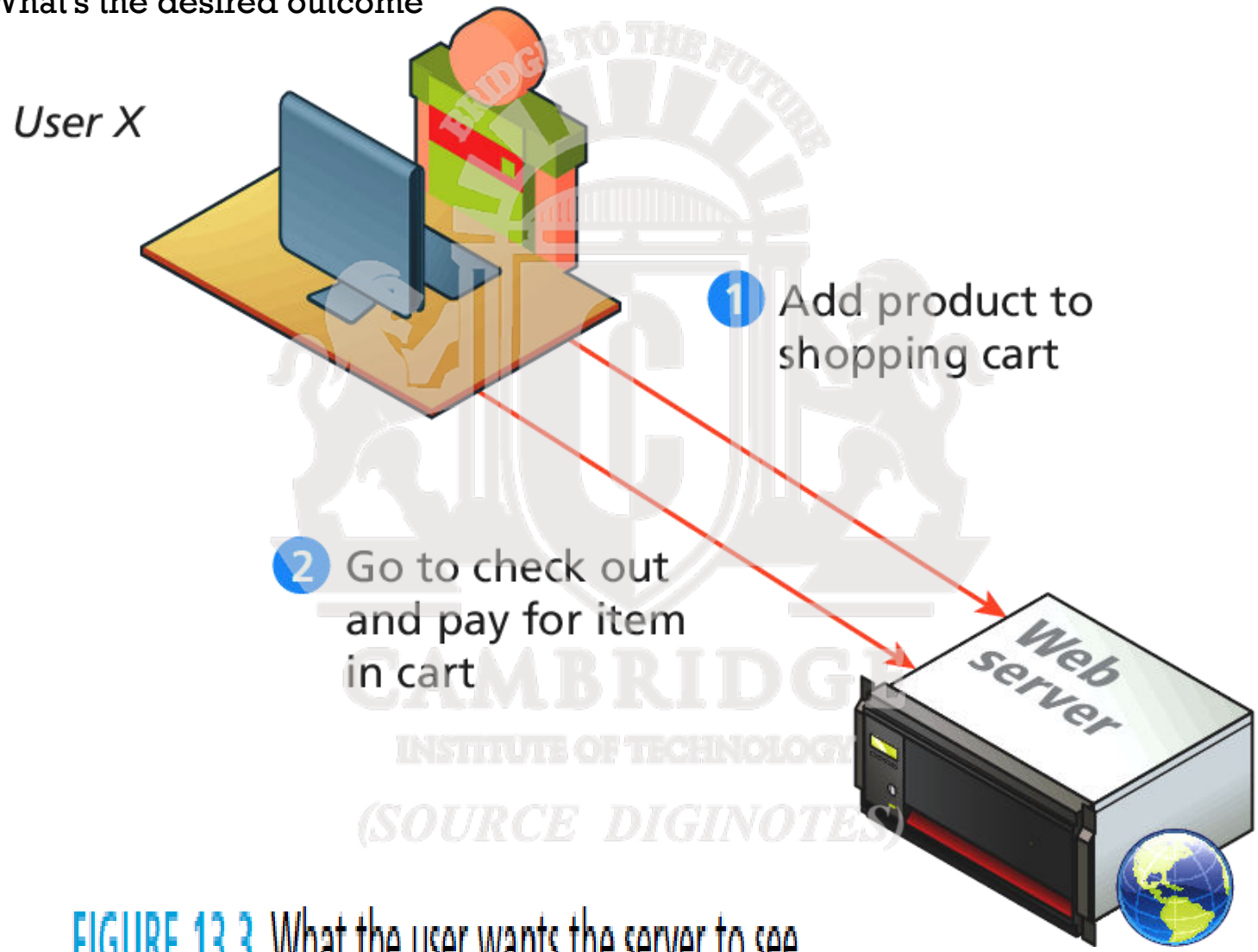


FIGURE 13.3 What the user wants the server to see

State in Web Applications

How do we reach our desired outcome?

- The rest of the chapter we will explain how web developers and web development environments work together through the constraints of HTTP to solve this particular problem.
- How does one web page pass information to another page?
- What mechanisms are available within HTTP to pass information to the server in our requests?

In HTTP, we can pass information using:

- Query strings
- Cookies

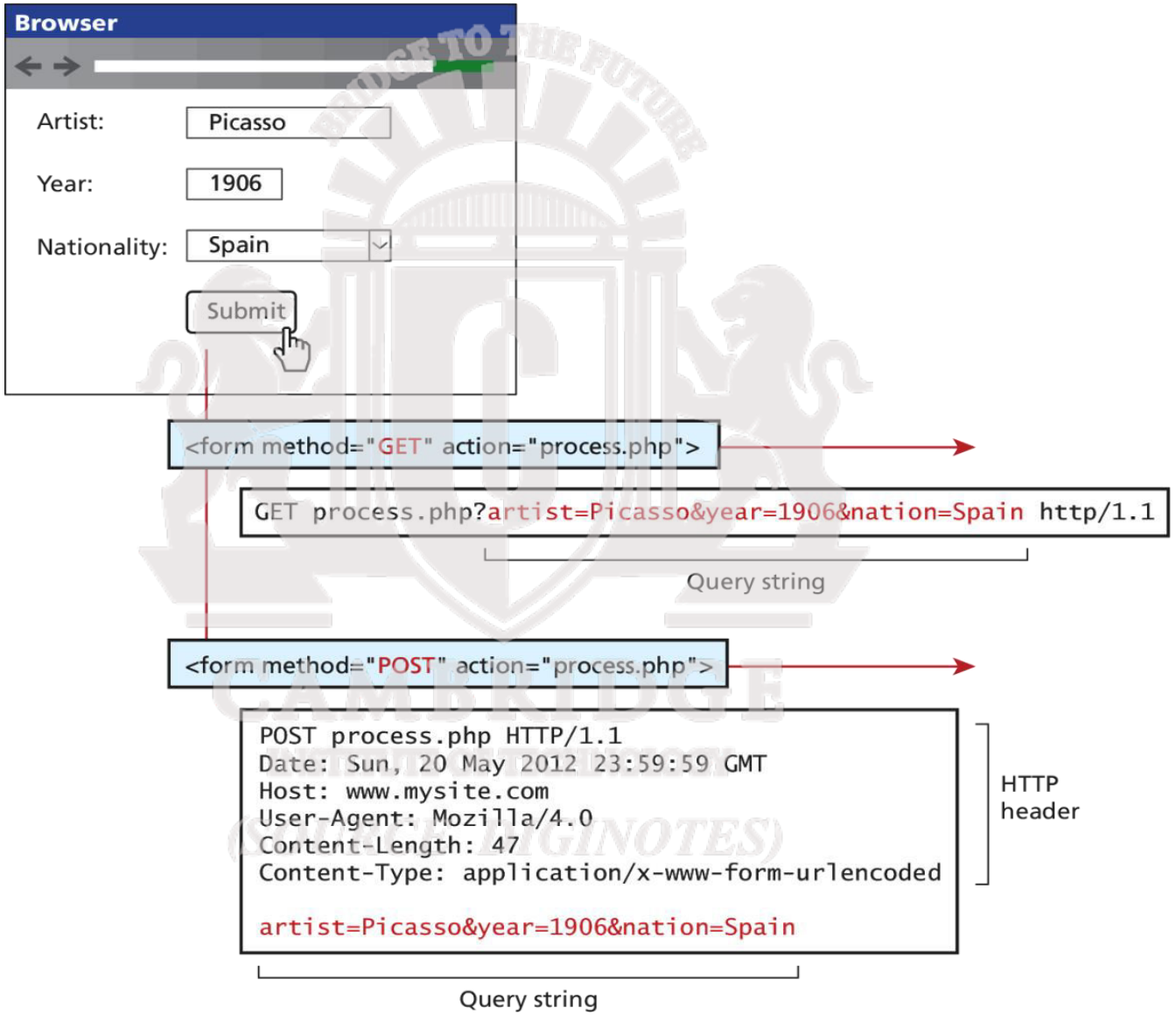
Section 2 of 8

PASSING INFORMATION VIA QUERY STRINGS

- A web page can pass query string information from the browser to the server using one of the two methods:
  - A query string within the URL(GET)
  - A query string within the HTTP header(POST)

Info in Query Strings

Recall GET and POST



Section 3 of 8

PASSING INFORMATION VIA THE URL PATH

Passing Info via URL Path

An Idealized looking link structure

❖Passing information from one page to another is done by query strings but they have drawback.

❖The URLs that result can be long and complicated.

❖while there is some dispute about whether dynamic URLs(i.e, ones with query string parameters) or static URLs are better from a search engine result optimization (or SEO)

❖Dynamic URLs (i.e., query string parameters) are a pretty essential part of web application development.

✓How can we do without them?

❖The answer is to rewrite the dynamic URL into a static one (and vice versa). This process is commonly called **URL rewriting**.

- In below fig, the top four commerce – related results for the search term

  " Reproduction Raphael portrait la donna velata"

  are shown along with their URLs.

- Notice how the top three do not use query string parameters but instead put the relevant information within the folder path or the file name.

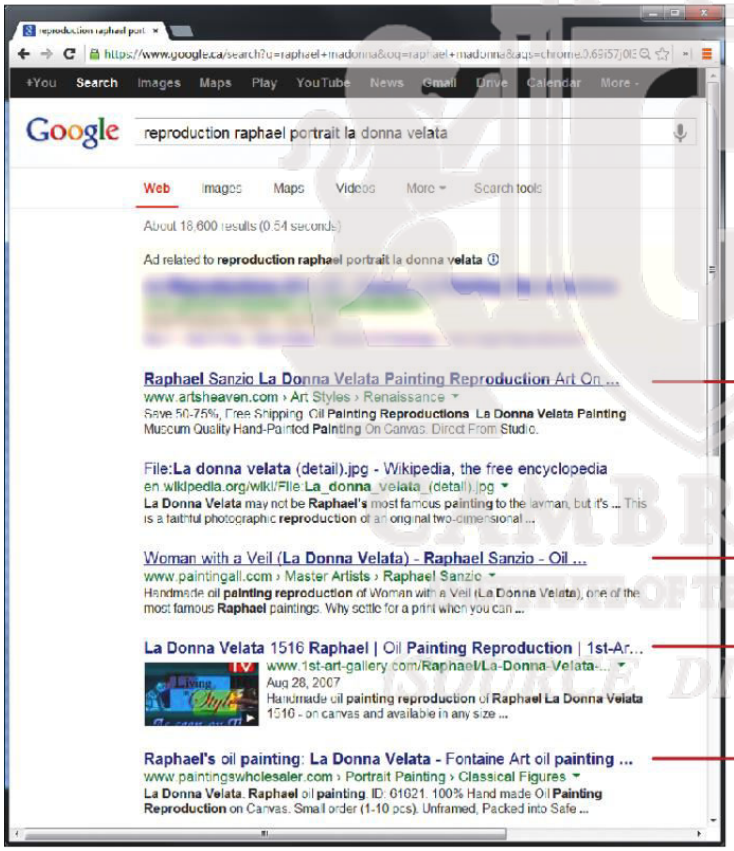
URL rewriting

Search Engine (Fine… and Human) Friendly

http://www.1st-art-gallery.com/Raphael/La-Donna-Velata-1516.html

http://www.paintingall.com/raphael-sanzio-woman-with-a-veil-la-donna-velata.html

http://www.artsheaven.com/raphael-la-donna-velata.html



http://www.paintingswholesaler.com/detail.asp?vcode=6umd7krr1yqi161c&title=La+Donna+Velata

URL rewriting

Search Engine (Fine… and Human) Friendly

❖We can try doing our own rewriting. Let us begin with the following URL with its query string information:

www.somedomain.com/DisplayArtist.php?artist=16

❖One typical alternate approach would be to rewrite the URL to:

www.somedomain.com/artists/16.php

❖Notice that the query string name and value have been turned into path names. One could improve this to make it more SEO friendly using the following:

www.somedomain.com/artists/Mary-Cassatt

URL rewriting in Apache and linux

You are not yet ready grasshoper

❖The mod_rewrite module uses a rule-based rewriting engine that utilizes Perl compatible regular expressions to change the URLs so that the requested URL can be mapped or redirected to another URL internally.

Section 4 of 8

COOKIES

Cookies

mmmm

- **Cookies** are a client-side approach for persisting state information.

- They are name=value pairs that are saved within one or more text files that are managed by the browser.

Cookies

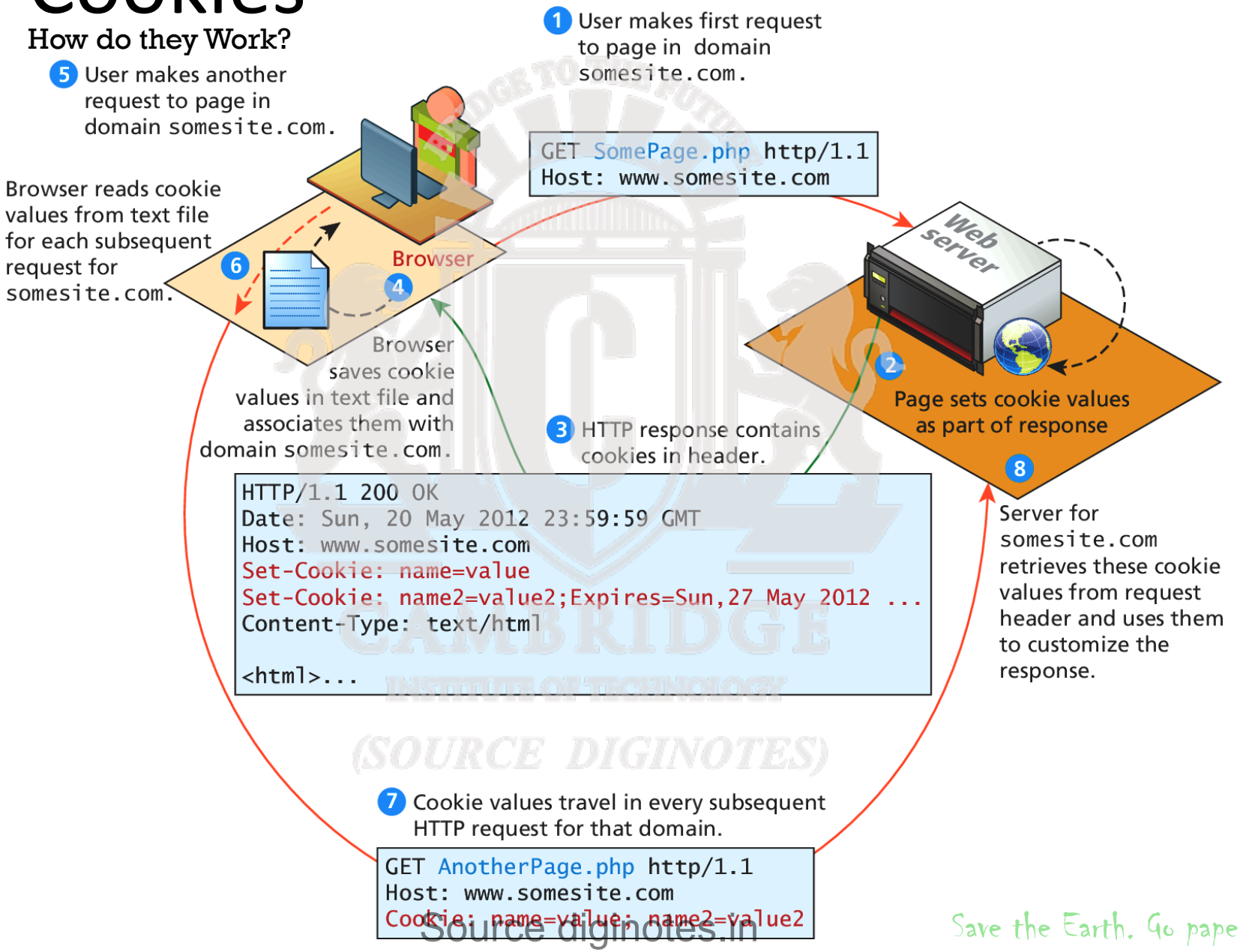
How do they Work?

❖While cookie information is stored and retrieved by the browser, the information in a cookie travels within the HTTP header.

- Sites that use cookies should not depend on their availability for critical features
- The user can delete cookies or tamper with them

Cookies

How do they Work?



Cookies

Chocolate and peanut butter

Two kinds of Cookie

- A **session cookie** has no expiry stated and thus will be deleted at the end of the user browsing session.
- **Persistent cookies** have an expiry date specified;

Using Cookies

Writing a cookie

```php
<?php
    // add 1 day to the current time for expiry time
    $expiryTime = time()+60*60*24;

    // create a persistent cookie
    $name = "Username";
    $value = "Ricardo";
    setcookie($name, $value, $expiryTime);
?>
```

LISTING 13.1 Writing a cookie

It is important to note that cookies must be written before any other page output.

Using Cookies

Reading a cookie

```php
<?php
   if( !isset($_COOKIE['Username']) ) {
      //no valid cookie found
   }
   else {
      echo "The username retrieved from the cookie is:";
      echo $_COOKIE['Username'];
   }
?>
```

LISTING 13.2 Reading a cookie

Using Cookies

Common usages

❖In addition to being used to track authenticated users and shopping carts, cookies can implement:

- "Remember me" persistent cookie
- Store user preferences
- Track a user's browsing behavior

Section 5 of 8

SERIALIZATION

Serialization

Down to 0s and 1s

- **Serialization** is the process of taking a complicated object and reducing it down to zeros and ones for either storage or transmission.

- In PHP objects can easily be reduced down to a binary string using the **serialize()** function.

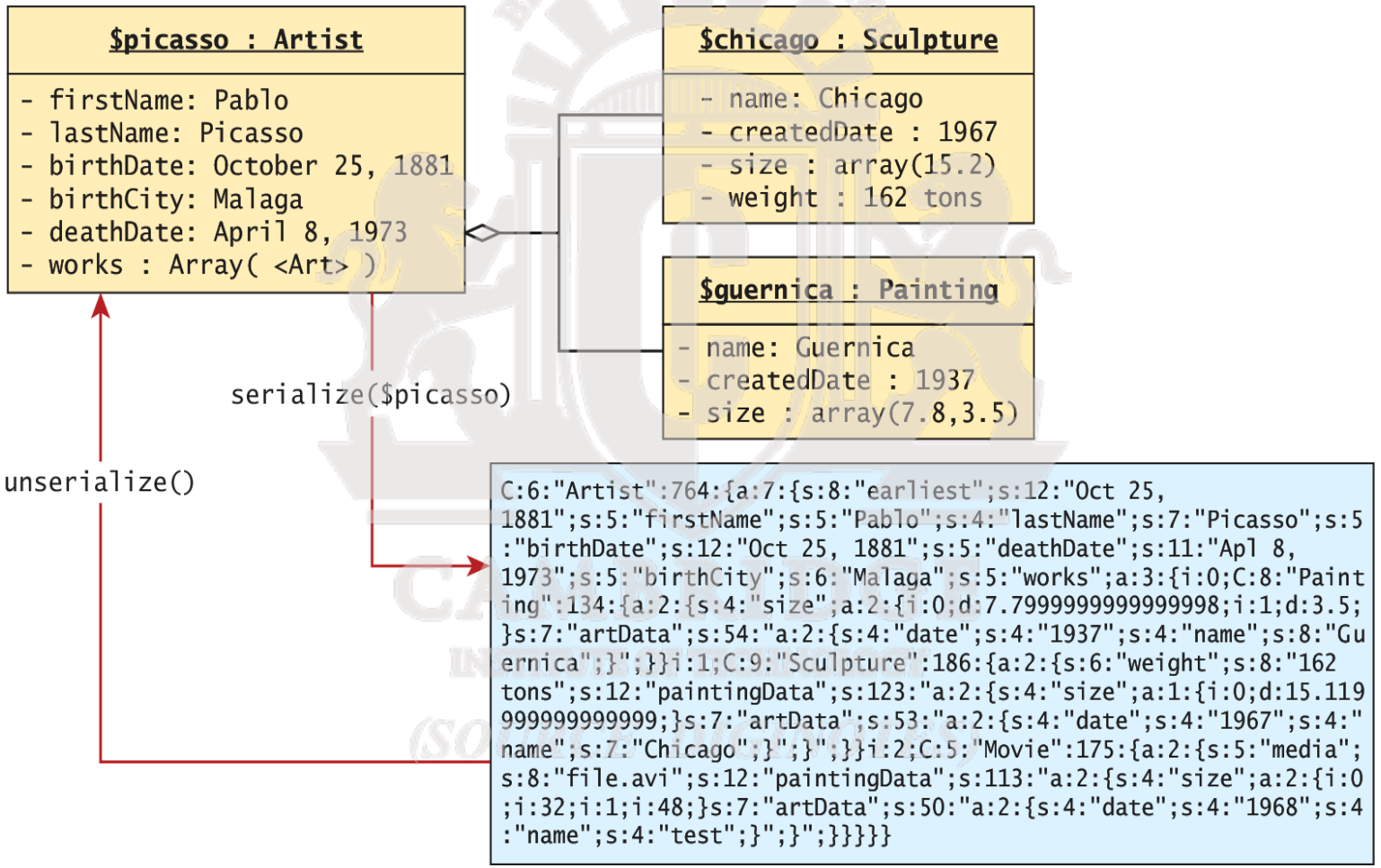
- The string can be reconstituted back into an object using the **unserialize()** method

```
interface Serializable {
    /* Methods */
    public function serialize();
    public function unserialize($serialized);
}
```

LISTING 13.3 The Serializable interface

Serialization

Serialization and deserialization



Serialization

```
class Artist implements Serializable {
    //...
    // Implement the Serializable interface methods
    public function serialize() {
        // use the built-in PHP serialize function
        return serialize(
                array("earliest" =>self::$earliestDate,
                      "first" => $this->firstName,
                      "last" => $this->lastName,
                      "bdate" => $this->birthDate,
                      "ddate" => $this->deathDate,
                      "bcity" => $this->birthCity,
                      "works" => $this->artworks
                      );
                );
    }
    public function unserialize($data) {
        // use the built-in PHP unserialize function
        $data = unserialize($data);
        self::$earliestDate = $data['earliest'];
        $this->firstName = $data['first'];
        $this->lastName = $data['last'];
        $this->birthDate = $data['bdate'];
        $this->deathDate = $data['ddate'];
        $this->birthCity = $data['bcity'];
        $this->artworks = $data['works'];
    }
    //...
}
```

LISTING 13.4 Artist class modified to implement the Serializable interface

Serialization

Consider our Artist class

The output of calling serialize($picasso) is:

C:6:"Artist":764:{a:7:{s:8:"earliest";s:13:"Oct 25, 1881";s:5:"firstName";s:5:"Pablo";s:4:"lastName";s:7:"Picasso";s:5:"birthDate";s:13:"Oct 25, 1881";s:5:"deathDate";s:11:"Apl 8, 1973";s:5:"birthCity";s:6:"Malaga";s:5:"works";a:3:{i:0;C:8:"Painting":134:{a:2:{s:4:"size";a:2:{i:0;d:7.7999999999999998;i:1;d:3.5;}s:7:"artData";s:54:"a:2:{s:4:"date";s:4:"1937";s:4:"name";s:8:"Guernica";}";}}i:1;C:9:"Sculpture":186:{a:2:{s:6:"weight";s:8:"162 tons";s:13:"paintingData";s:133:"a:2:{s:4:"size";a:1:{i:0;d:15.119999999999999;}s:7:"artData";s:53:"a:2:{s:4:"date";s:4:"1967";s:4:"name";s:7:"Chicago";}";}}i:2;C:5:"Movie":175:{a:2:{s:5:"media";s:8:"file.avi";s:13:"paintingData";s:113:"a:2:{s:4:"size";a:2:{i:0;i:32;i:1;i:48;}s:7:"artData";s:50:"a:2:{s:4:"date";s:4:"1968";s:4:"name";s:4:"test";}";}";}}}}}

If the data above is assigned to $data, then the following line will instantiate a new object identical to the original:

```
$picassoClone = unserialize($data);
```

Application of Serialization

Remember our state problem

- Since each request from the user requires objects to be reconstituted, using serialization to store and retrieve objects can be a rapid way to maintain state between requests.

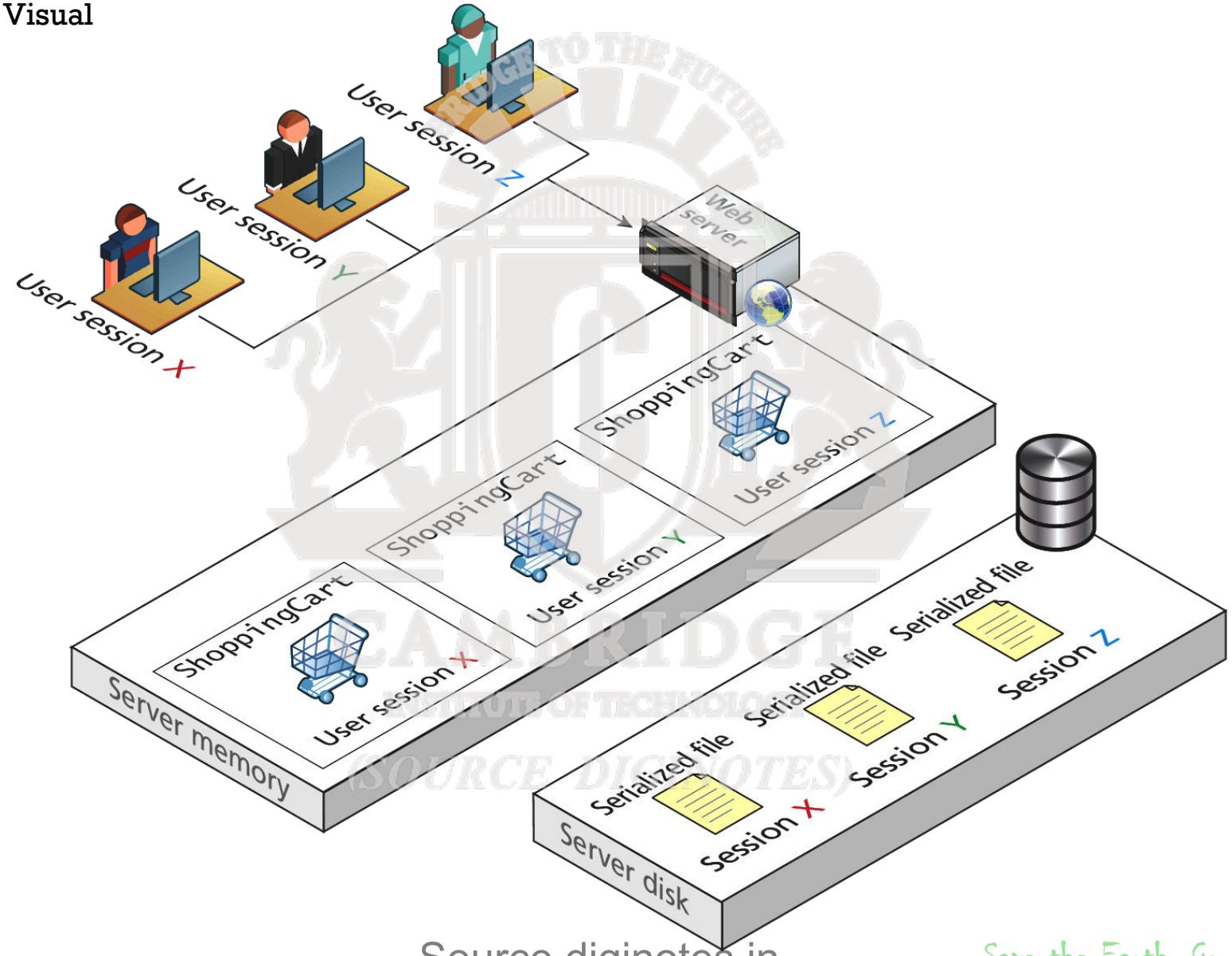
- At the end of a request you store the state in a serialized form, and then the next request would begin by deserializing it to reestablish the previous state.

Section 6 of 8

SESSION STATE

Session State

Visual



Session State

❖All modern web development environments provide some type of session state mechanism.

❖**Session state** is a server-based state mechanism that lets web applications store and retrieve objects of any type for each unique user session.

❖Session state is ideal for storing more complex objects or data structures that are associated with a user session.

❖In PHP, session state is available to the via the $_SESSION variable

❖Must use session_start() to enable sessions.

Session State

Accessing State

```php
<?php

session_start();

if ( isset($_SESSION['user']) ) {
    // User is logged in
}
else {
    // No one is logged in (guest)
}
?>
```

LISTING 13.5 Accessing session state

Session State

Checking Session existance

```php
<?php
include_once("ShoppingCart.class.php");

session_start();

// always check for existence of session object before accessing it
if ( !isset($_SESSION["Cart"]) ) {
   //session variables can be strings, arrays, or objects, but
   // smaller is better
   $_SESSION["Cart"] = new ShoppingCart();
}
$cart = $_SESSION["Cart"];
?>
```

LISTING 13.6 Checking session existence

Session State

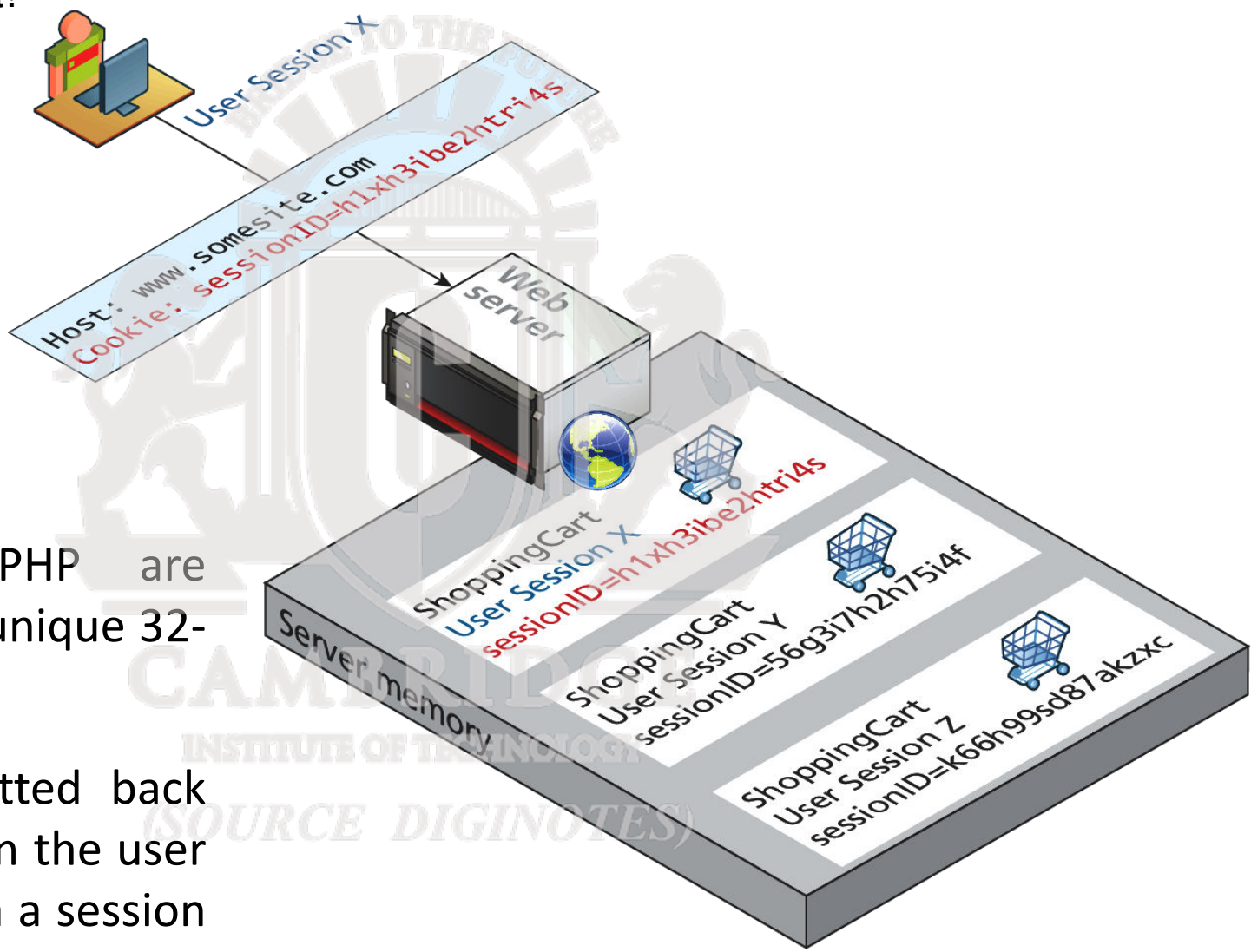
Checking Session existence

```php
<?php
include_once("ShoppingCart.class.php");

session_start();

// always check for existence of session object before accessing it
if ( !isset($_SESSION["Cart"]) ) {
   //session variables can be strings, arrays, or objects, but
   // smaller is better
   $_SESSION["Cart"] = new ShoppingCart();
}
$cart = $_SESSION["Cart"];
?>
```

LISTING 13.6 Checking session existence

How does state session work?

It's magic right?



- Sessions in PHP are identified with a unique 32-byte session ID.
- This is transmitted back and forth between the user and the server via a session cookie

How does state session work?

It's magic right?

- For a brand new session, PHP assigns an initially empty dictionary-style collection that can be used to hold any state values for this session.
- When the request processing is finished, the session state is saved to some type of state storage mechanism, called a session state provider
- When a new request is received for an already existing session, the session's dictionary collection is filled with the previously saved session data from the session state provider.

Session Storage

•It is possible to configure many aspects of sessions including where the session files are saved.

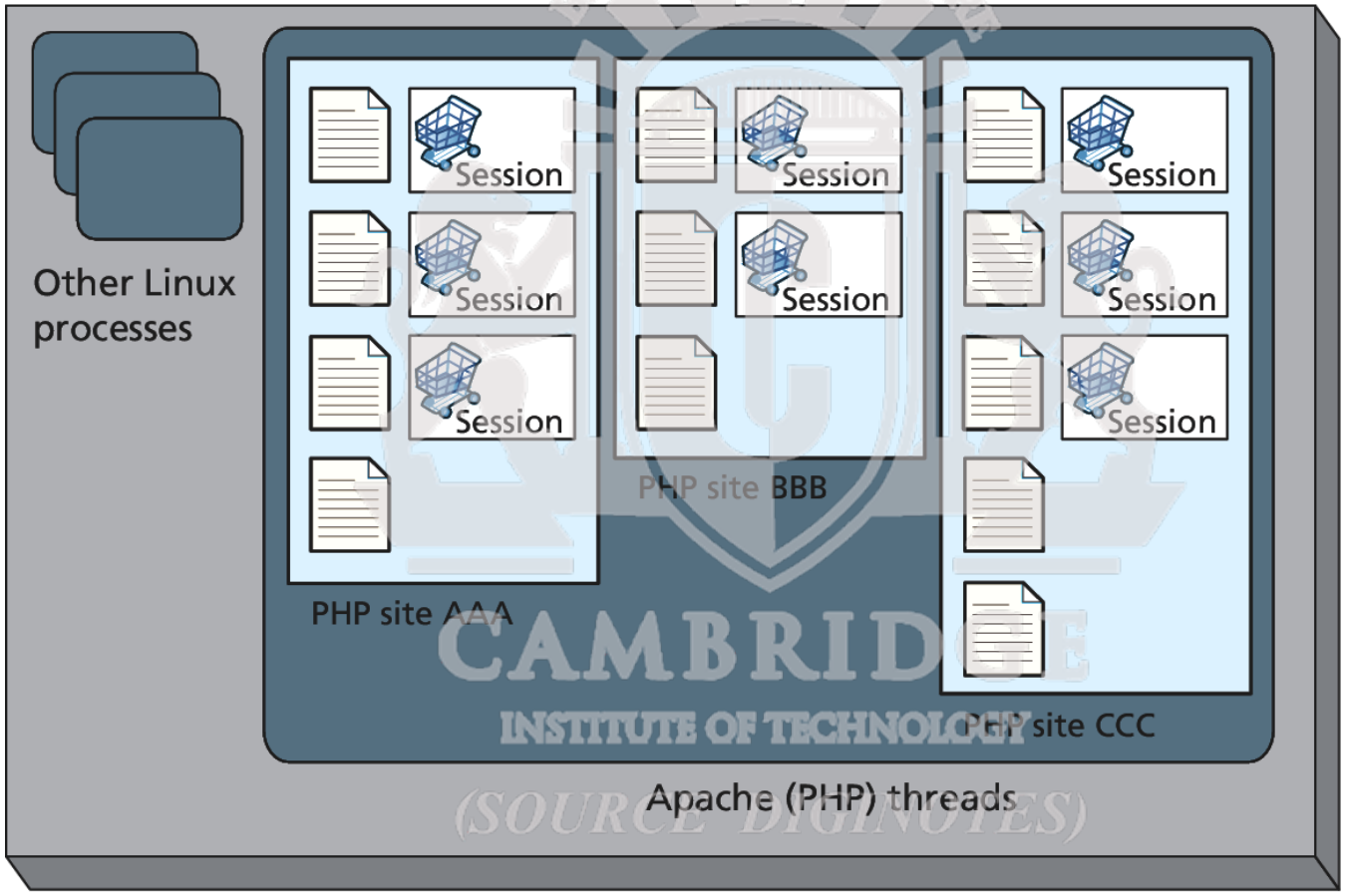
•The decision to save sessions to files rather than in memory (like ASP.NET) addresses the issue of memory usage that can occur on shared hosts as well as persistence between restarts.

•Inexpensive web hosts may sometimes stuff hundreds or even thousands of sites on each machine.

•Server memory may be storing not only session information, but pages being executed, and caching information

Session Storage

Applications and Server Memory



Session Storage

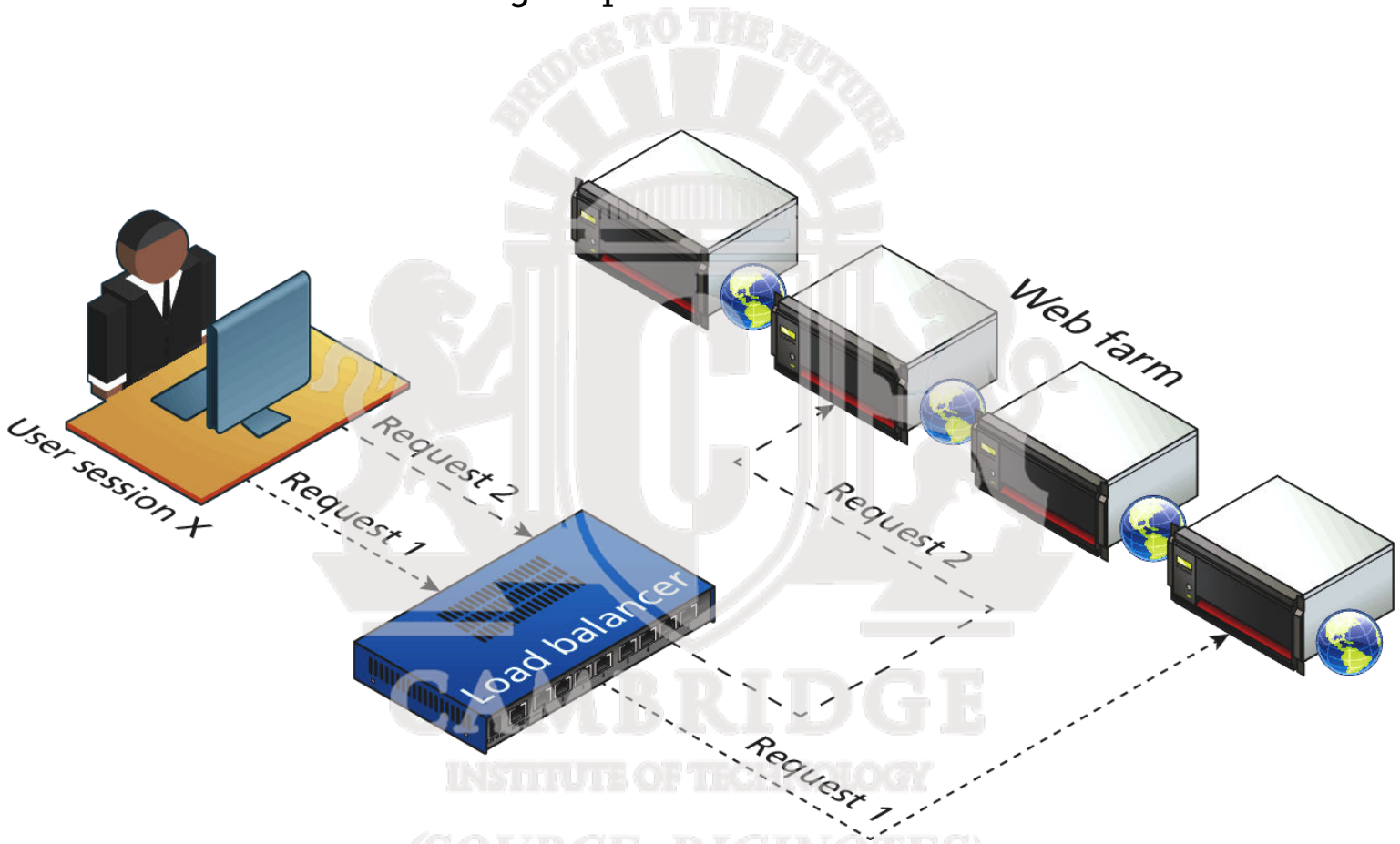
High Volume considerations

❖Higher-volume web applications often run in an environment in which multiple web servers (also called a web farm) are servicing requests.

❖In such a situation the in-process session state will not work, since one server may service one request for a particular session, and then a completely different server may service the next request for that session

Session Storage

Web Farm Sessions: Visualizing the problem



Session Storage

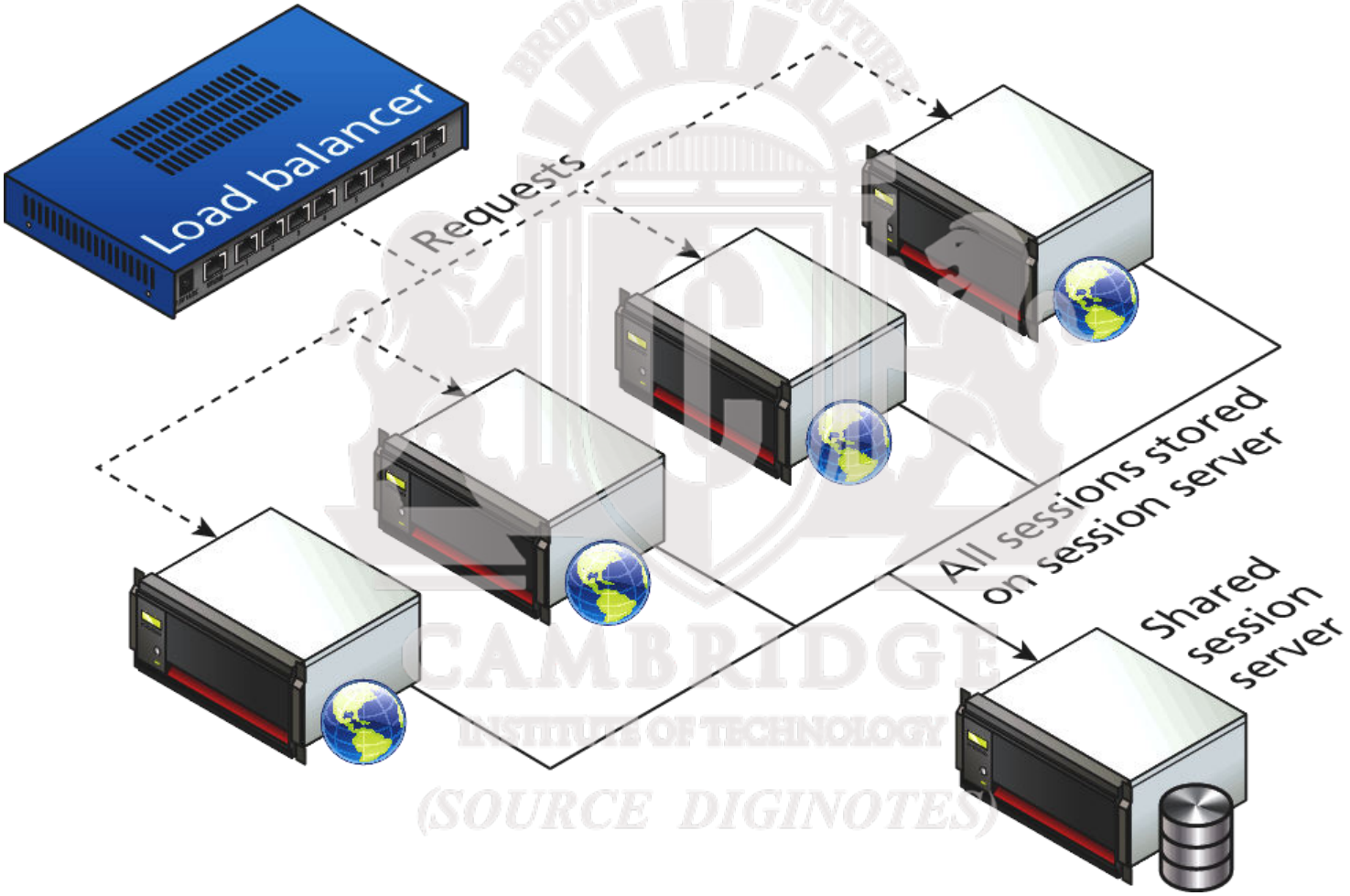
Visualizing the problem

There are effectively two categories of solution to this problem.

1. Configure the load balancer to be “session aware” and relate all requests using a session to the same server.
2. Use a shared location to store sessions, either in a database, memcache, or some other shared session state mechanism

Session Storage

Shared location with memcache



Session Storage

Shared location configuration in php.ini (on each webserver)

```
[Session]
; Handler used to store/retrieve data.
session.save_handler = memcache
session.save_path = "tcp://sessionServer:11211"
```

LISTING 13.7 Configuration in php.ini to use a shared location for sessions

13.7 HTML5 Web Storage

❖Web storage is a new javascript only API introduced in HTML5.

❖IT is meant to be a replacement to cookies, in that web storage is managed by the browser.

❖Unlike cookies web storage data is not transported to and from the server with every request and response

❖Web storage is not limited to 4k size barrier of cookies.

❖Limit of 5 MB but browsers are allowed to store more per domain.

❖Just as there were two types of cookies, there are two types of web storage

- ❖Local storage – It is for saving information that will persist between browser sessions.
- ❖Session storage – It is for information that will be lost once the browser session is finished.

13.7.1 Using Web Storage

- Below code illustrates the javaScript code for writitng information to web stroage.
- There are two ways to store values in web storage
- Using setItem() function, or using the property shortcut (Eg: session stoargear.FavoriteArtist().

```
<form ... >
   <h1>Web Storage Writer</h1>
   <script language="javascript" type="text/javascript">

        if (typeof (localStorage) === "undefined" ||
              typeof (sessionStorage) === "undefined") {
            alert("Web Storage is not supported on this browser...");
        }
        else {
            sessionStorage.setItem("TodaysDate", new Date());
            sessionStorage.FavoriteArtist = "Matisse";

            localStorage.UserName = "Ricardo";
            document.write("web storage modified");
        }
   </script>
   <p><a href="WebStorageReader.php">Go to web storage reader</a></p>
</form>
```

LISTING 13.8 Writing web storage

- Below code illustrates the process of reading from web storage is equally straight forward.
- The difference between sessionStorage and localStorage in this example is that if you close the browser after writing and then run the code (Above code), only the local storage item will still contain a value.

```
<form id="form1" runat="server">
    <h1>Web Storage Reader</h1>
    <script language="javascript" type="text/javascript">

        if (typeof (localStorage) === "undefined" ||
                typeof (sessionStorage) === "undefined") {
            alert("Web Storage is not supported on this browser...");
        }
        else {
            var today = sessionStorage.getItem("TodaysDate");
            var artist = sessionStorage.FavoriteArtist;

            var user = localStorage.UserName;
            document.write("date saved=" + today);
            document.write("<br/>favorite artist=" + artist);
            document.write("<br/>user name = " + user);
        }
    </script>
</form>
```

LISTING 13.9 Reading web storage

13.7.2 Why Would We Use Web Storage?

❖Cookies have the disadvantage of being limited in size, potentially disabled by the user, other security attacks and being sent in every single request and response to and from a given domian.

❖Web storage is not as a cookie replacement but as a local cache for relatively static items available to javascript.

❖One use of web storage is to store static content downloaded asynchronously such as xml or JSON from a web service in web storage, this reducing server load for subsequent requests by the session.

❖Below fig illustrates an example of how web storage could be used as a mechanism for reducing server data request, there by speeding up the display of the page on the browser as well as reducing load on the server.

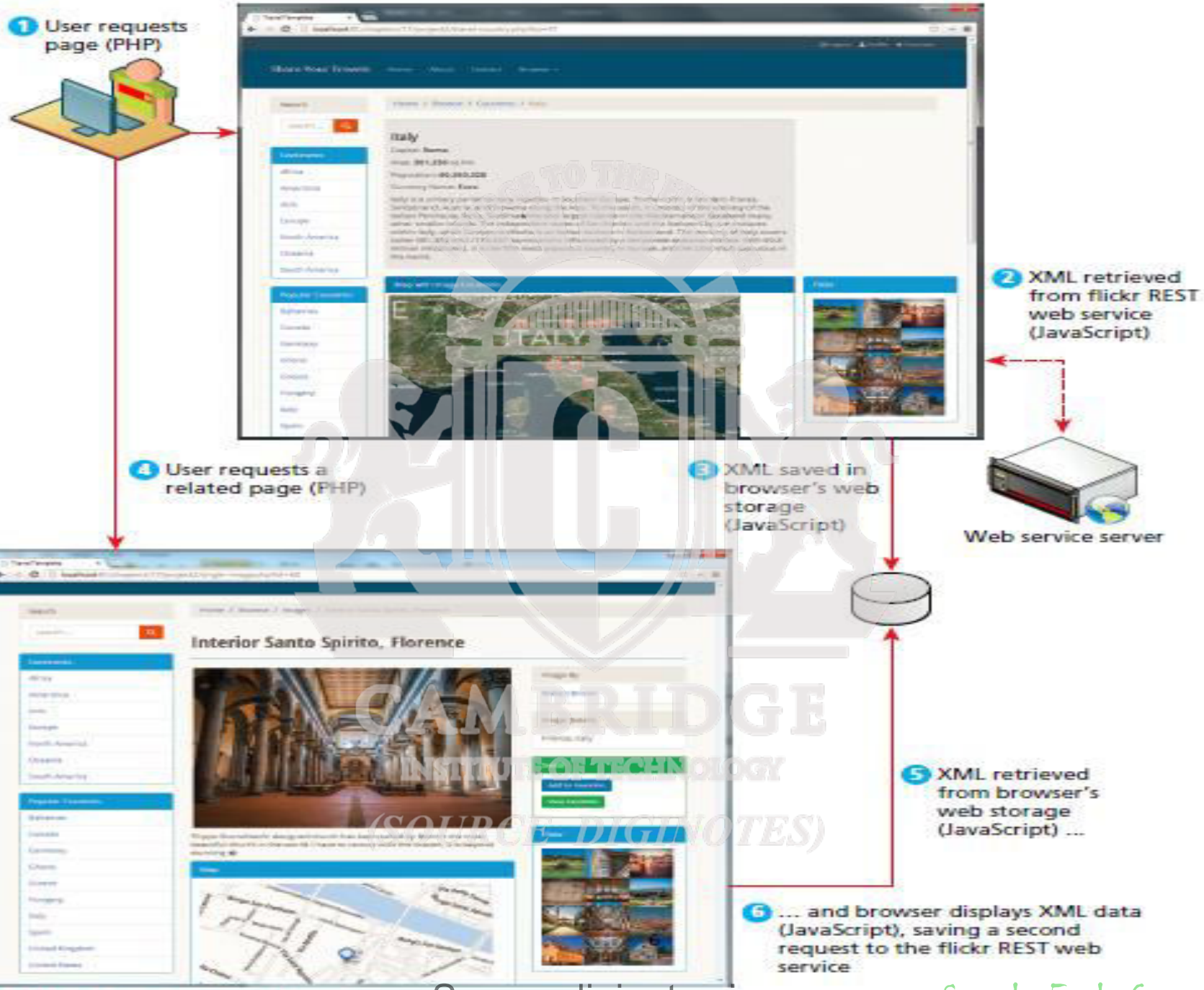


FIGURE 13.13 Using web storage

13.8 Caching

13.8 Caching

❖Caching is the vital way to improve the performance of web applications. Your browser uses caching to speed up the user experience by using locally stored versions of images and other files rather than re – requesting the files from the server.

❖A server side developer only had limited control over browser caching.

❖Why is this necessary?

❖Every time a PHP page is requested, it must be fetched, parsed and executed by the PHP engine and end result is HTML that is sent back to the requestor.

❖On way to address this problem is to cache the generated markup in server memory so that subsequent requests can be served from memory rather than from the execution of the page.

❖ There are two basic strategies to caching web applications.

- ❖PAGE OUTPUT CACHING

  It saves the rendered output of a page or user control and reuses the output instead of reprocessing the page when a user requests the page again

- ❖APPLICATION DATA CACHING

  It allows the developer to programmatically cache data.

13.8.1 Page Output Caching

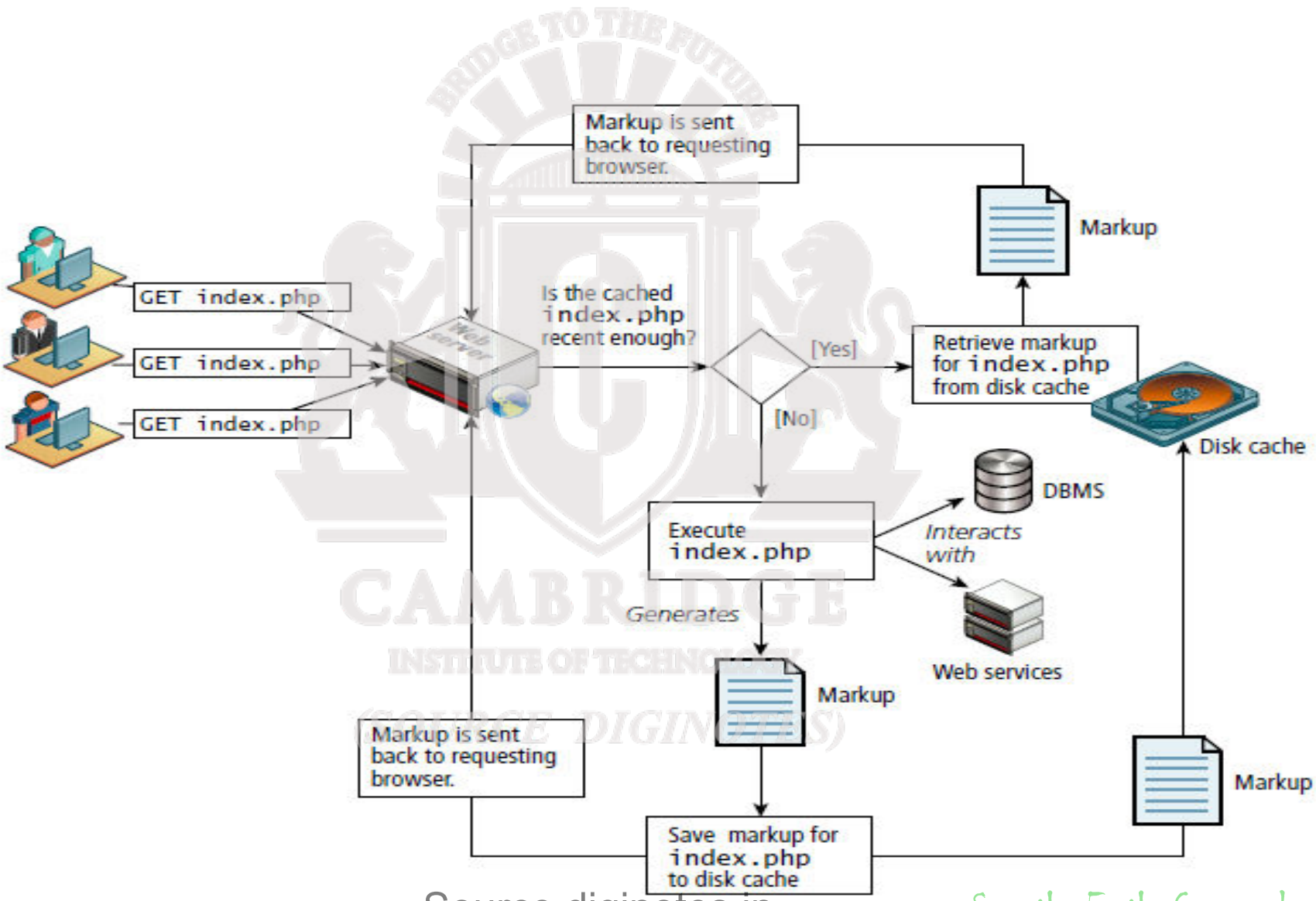


FIGURE 13.14 Page output caching

- There are two models for page caching:
  - Full page caching
  - Partial page caching :

    Only specific parts of page are cached while other parts are dynamically generated in the normal manner.

13.8.2 Application Data Caching

❖One of the biggest drawbacks with page output caching is that performance gains will only be had if the entire cached page is the same for numerous requests.

❖An alternate strategy is to use application data caching in which a page will programmatically place commonly used collections of data that require time intensive queries from the database or web server into cache memory, and then other pages that also need that same data can use the cache version rather than re – retrieve it from its original location.

❖While the default installation of PHP does not come with an application caching ability, a widely available free PECL extension called memcache is used for this ability.

```php
<?php

// create connection to memory cache
$memcache = new Memcache;
$memcache->connect('localhost', 11211)
   or die ("Could not connect to memcache server");

$cacheKey = 'topCountries';
/* If cached data exists retrieve it, otherwise generate and cache
   it for next time */
   if ( ! isset($countries = $memcache->get($cacheKey)) ) {

   // since every page displays list of top countries as links
   // we will cache the collection

   // first get collection from database
   $cgate = new CountryTableGateway($dbAdapter);
   $countries = cgate->getMostPopular();

   // now store data in the cache (data will expire in 240 seconds)
   $memcache->set($cacheKey, $countries, false, 240)
       or die ("Failed to save cache data at the server");
}
// now use the country collection
displayCountryList($countries);

?>
```

LISTING 13.10 Using memcache

Advanced JavaScript & JQuery

Chapter 15

Section 1 of 6

JAVASCRIPT PSEUDO-CLASSES

15.1 JavaScript Pseudo-Classes

❖JavaScript has no formal class mechanism, it does support objects (DOM).

❖While most OO languages that supports objects also support classes formally, JavaScript does not.

❖Instead we define Pseudo – classes through a variety of syntax.

15.1.1 Using Object Literals

Without Classes

- An array in JavaScript can be instantiated

```
var daysofweek = [“sun” , ”mon” , “tue”…..];
```

- An object can be instantiated using object literals.
- **Object literals** are a list of key-value pairs with colons between the key and value with commas separating key-value pairs.
- Object literals are also known as **Plain Objects** in jQuery.

Object Oriented Design

Without Classes

- JavaScript has no formal class mechanism.

Simple Variable

```
var car = "Fiat";
```

Then this

```
var car = {type:"Fiat", model:500, color:"white"};
```

Properties

car.name = Fiat
car.model = 500
car.weight = 850kg
car.color = white

Methods:

car.start()
car.drive()
car.brake()
car.stop()

Using Object Literals

Consider a die (single dice)

```
//define a 6 faced dice, with a red color.

var oneDie =     { color : "FF0000",

                   faces : [1,2,3,4,5,6]

                   };
```

These elements can be accessed using dot notation.

For instance

```
oneDie.color="0000FF";
```

15.1.2Emulate Classes with functions

We told you this would get weird

Use functions to encapsulate variables and methods together.

```
function Die(col) {
   this.color=col;
   this.faces=[1,2,3,4,5,6];
}
```

LISTING 15.1 Very simple Die pseudo-class definition as a function

Instantiation looks much like in PHP:

var oneDie = new Die("0000FF");

Emulate Classes with functions

Adding methods to the objects

•To define a method in an object's function one can either define it internally or use a reference to a function define outside the class.

•One technique for adding a method inside of a class definition is by assigning an anonymous function to a variable

```
function Die(col) {
   this.color=col;
   this.faces=[1,2,3,4,5,6];

   // define method randomRoll as an anonymous function
   this.randomRoll = function() {
      var randNum = Math.floor((Math.random() * this.faces.length)+ 1);
      return faces[randNum-1];
   };
}
```

LISTING 15.2 Die pseudo-class with an internally defined method

```
var oneDie = new Die("0000FF");
console.log(oneDie.randomRoll() + " was rolled");
```

Emulate Classes with functions

Not very efficient

- This mechanism for methods in effective, it is not a memory efficient approach because each inline method is redefined for each new object

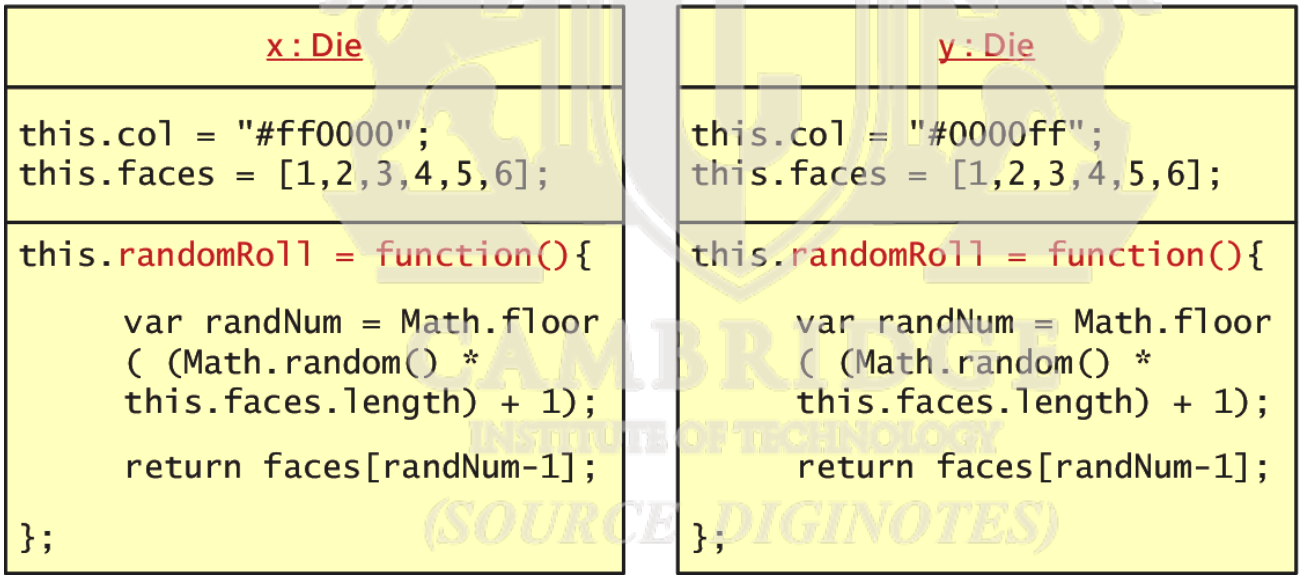


FIGURE 15.1 Illustrating duplicated method definition

15.1.3 Using Prototypes

Prototypes

So you can use a *prototype* of the class.

- **Prototypes** are used to make JavaScript behave more like an object-oriented language.
- The prototype properties and methods are defined *once* for all instances of an *object*.
- Every object has a prototype

Using Prototypes

moving the randomRoll() function into the **prototype**.

```
// Start Die Class
function Die(col) {
   this.color=col;
   this.faces=[1,2,3,4,5,6];
}

Die.prototype.randomRoll = function() {
   var randNum = Math.floor((Math.random() * this.faces.length) + 1);
   return faces[randNum-1];
};
// End Die Class
```

LISTING 15.3 The Die pseudo-class using the prototype object to define methods

❖This definition is better because it defines the method only once, no matter how many instance of die are created.

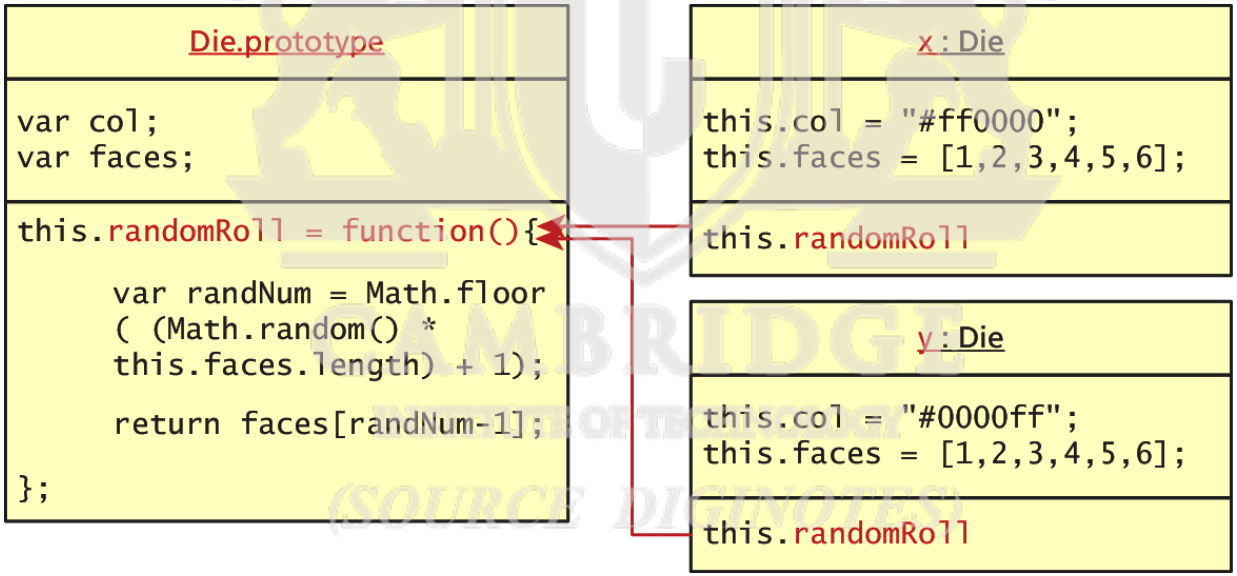
❖How many ever created it will be reference to that one method. (fig below)

Using Prototypes

No duplication of methods

Since all instances of a Die share the same prototype object, the function declaration only happens one time and is shared with all Die instances.

FIGURE 15.2 Illustration of JavaScript prototypes as pseudo-classes



A prototype is an object from which other objects inherit.

More about Prototypes

Extend any Object

- Every object (and method) in JavaScript has a prototype.
- In addition to using prototypes for our own pseudo-classes, prototypes enable you to *extend* existing classes by adding to their prototypes

```
String.prototype.countChars = function (c) {
   var count=0;
   for (var i=0;i<this.length;i++) {
      if (this.charAt(i) == c)
         count++;
   }
   return count;
}
```

LISTING 15.4 Adding a method named countChars to the String class

Section 2 of 6

JQUERY FOUNDATIONS

jQuery
Framework

- A **library** or **framework** is software that you can utilize in your own software, which provides some common implementations of standard ideas.
- Many developers find that once they start using a framework like jQuery, there's no going back to "pure" JavaScript because the framework offers so many useful shortcuts and breif ways of doing things

jQuery

A 1 slide history

In August 2005 jQuery founder John Resig was looking into how to better combine **CSS selectors with succinct JavaScript notation**.

- Within 1 year AJAX and animations were added
- Additional modules
  - jQuery UI extension
  - mobile device support
- Continues to improve.

jQuery

Not the only one, but a popular one

jQuery is now the most popular JavaScript library currently in use as supported by the statistics from BuiltWith.com

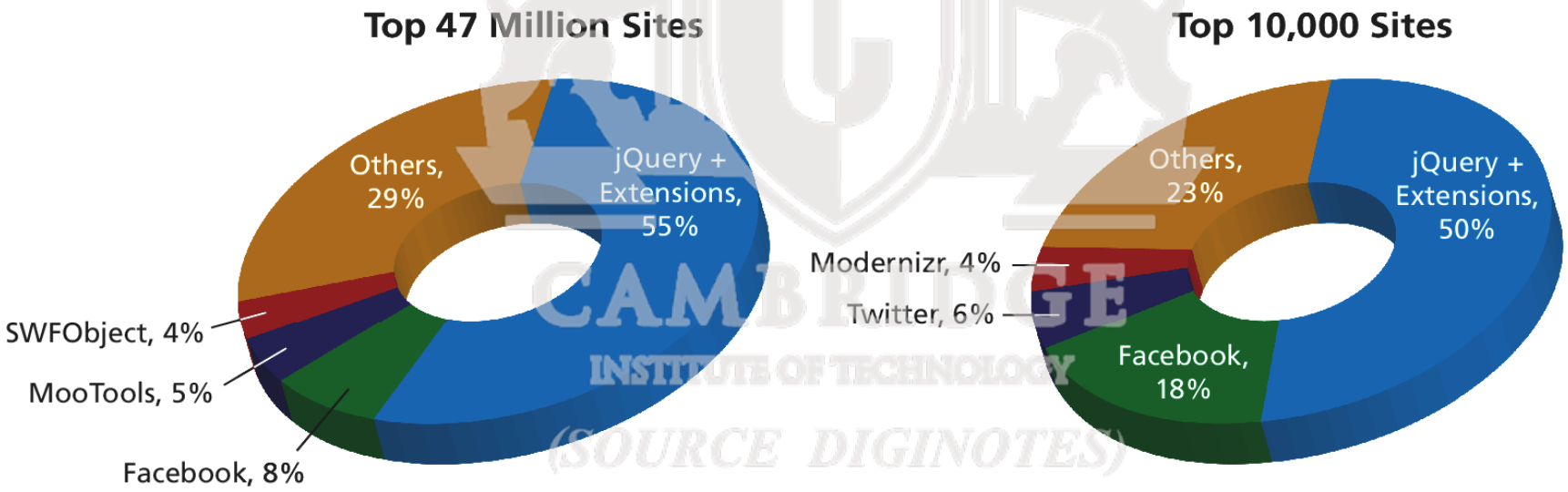


FIGURE 15.3 Comparison of the most popular JavaScript frameworks (data courtesy of BuiltWith.com)

Including jQuery

Let's get started

You must either:

- link to a locally hosted version of the library
- Use an approved third-party host, such as Google, Microsoft, or jQuery itself

Including jQuery

Content Delivery Network

Using a third-party **content delivery network (CDN)** is advantageous for several reasons.

- The bandwidth of the file is offloaded to reduce the demand on your servers.
- The user may already have cached the third-party file and thus not have to download it again, thereby reducing the total loading time.

A disadvantage to the third-party CDN is that your jQuery will fail if the third-party host fails (unlikely but possible)

Including jQuery

Content Delivery Network and fallback

```
<script src="http://code.jquery.com/jquery-1.9.1.min.js"></script>
<script type="text/javascript">
window.jQuery ||
document.write('<script src="/jquery-1.9.1.min.js"><\/script>');
</script>
```

LISTING 15.5 jQuery loading using a CDN and a local fail-safe if the CDN is offline

jQuery Selectors

Should ring a bell

- When discussing basic JavaScript we introduced the **getElementByID()** and **querySelector()** selector functions in JavaScript.
- Although the advanced **querySelector()** methods allow selection of DOM elements based on CSS selectors, it is only implemented in newest browsers
- jQuery introduces its own way to select an element, which under the hood supports a myriad of older browsers for you!

jQuery Selectors

The easiest way to select an element yet

✓The relationship between DOM objects and selectors is so important in JavaScript programming that the pseudo-class bearing the name of the framework,

jQuery()

✓Is reserved for selecting elements from the DOM.

✓Because it is used so frequently, it has a shortcut notation and can be written as

$()

Basic Selectors

All the way back to CSS

The four basic selectors are:

- $("*") **Universal selector** matches all elements (and is slow).
- $("tag") **Element selector** matches all elements with the given element name.
- $(".class") **Class selector** matches all elements with the given CSS class.
- $("#id") **Id selector** matches all elements with a given HTML id attribute.

Basic Selectors

All the way back to CSS

For example, to select the single element with id="grab" you would write:

```
var singleElement = $("#grab");
```

To get a set of all the <a> elements the selector would be:

```
var allAs = $("a");
```

These selectors replace the use of getElementById() entirely.

More CSS Selectors

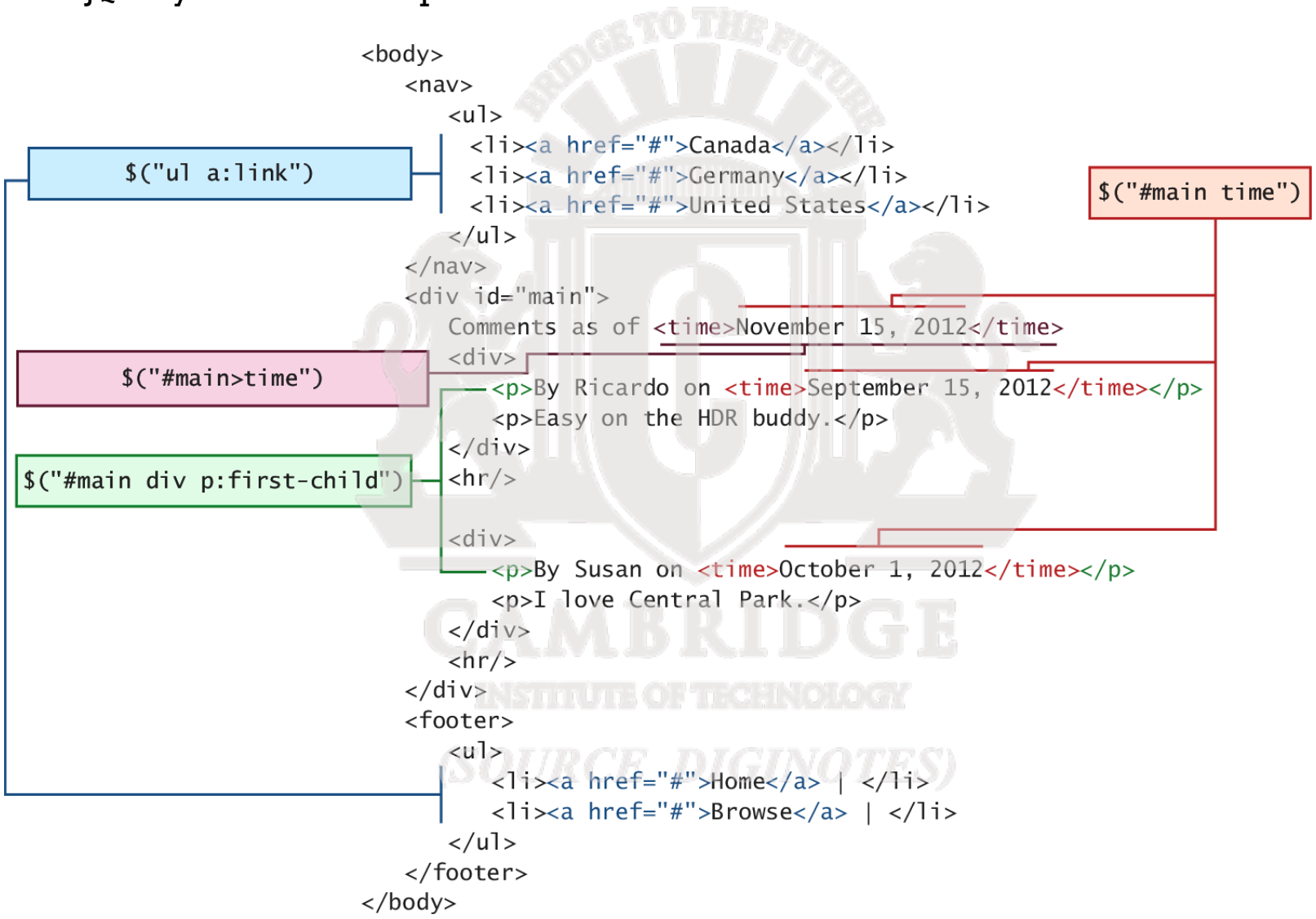
jQuery's selectors are powerful indeed

In addition to these basic selectors, you can use the other CSS selectors that were covered in Chapter 3 on CSS:

- attribute selectors,
- pseudo-element selectors, and
- contextual selectors

More CSS Selectors

jQuery's selectors are powerful indeed



Attribute Selector

Really a review of CSS

Recall from CSS that you can select

- by attribute with square brackets
  - [attribute]
- Specify a value with an equals sign
  - [attribute=value]
- Search for a particular value in the beginning, end, or anywhere inside a string
  - [attribute^=value]
  - [attribute$=value]
  - [attribute*=value]

Attribute Selector

Really a review of CSS

Consider a selector to grab all <img> elements with an src attribute beginning with */artist/* as:

```
var artistImages = $("img[src^='/artist/']");
```

Pseudo-Element Selector

Not to be confused with the pseudo-classes in JavaScript

pseudo-element selectors are also from CSS and allow you to append to any selector using the colon and one of

- :link
- :visited
- :focus
- :hover
- :active
- :checked
- :first-child, :first-line, and :first-letter

Pseudo-Element Selector

Not to be confused with the pseudo-classes in JavaScript

Selecting all links that have been visited, for example, would be specified with:

```
var visitedLinks = $("a:visited");
```

Contextual Selector

Put it into context

Contextual selectors are also from CSS. Recall that these include:

- descendant (space)
- child (>)
- adjacent sibling (+)
- and general sibling (~).

To select all <p> elements inside of elements you would write

```
var para = $("div p");
```

Content Filters

Above and Beyond CSS

The **content filter** is the only jQuery selector that allows you to append filters to all of the selectors you've used thus far and match a particular pattern.

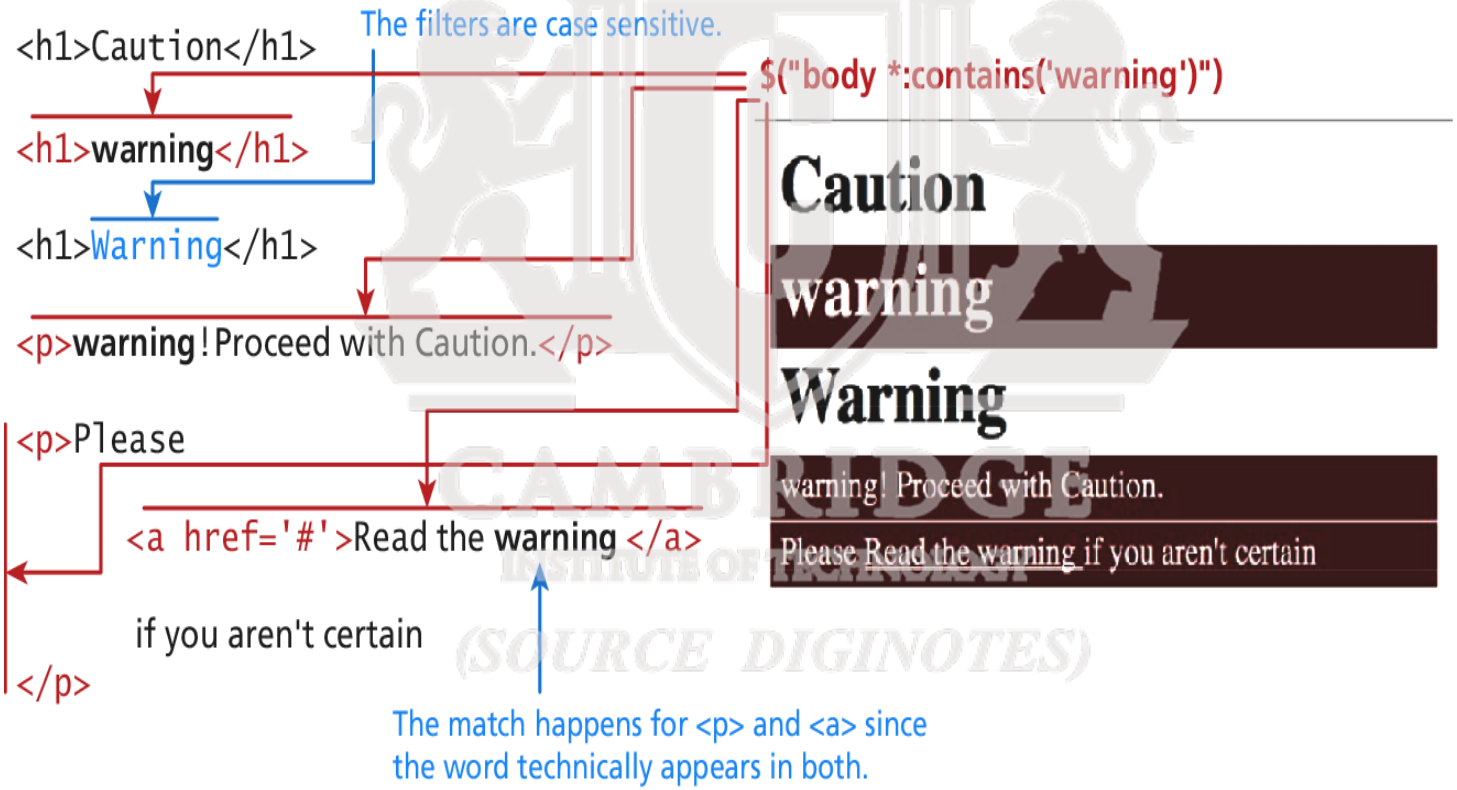
Select elements that have:

- a particular child using :has()
- have no children using :empty
- match a particular piece of text with :contains()

Content Filters

Above and Beyond CSS

$("body *:contains('warning')")



Form Selectors

Selector	CSS Equivalent	Description
$(:button)	$("button, input[type='button']")	Selects all buttons
$(:checkbox)	$('[type=checkbox]‘)	Selects all checkboxes
$(:checked)	No Equivalent	Selects elements that are checked. This includes radio buttons and checkboxes.
$(:disabled)	No Equivalent	Selects form elements that are disabled.
$(:enabled)	No Equivalent	Opposite of :disabled
$(:file)	$('[type=file]')	Selects all elements of type file
$(:focus)	$(document.activeElement)	The element with focus
$(:image)	$('[type=image]')	Selects all elements of type image
$(:input)	No Equivalent	Selects all <input>, <textarea>, <select>, and <button> elements.
$(:password)	$('[type=password]')	Selects all password fields
$(:radio)	$('[type=radio]')	Selects all radio elements
$(:reset)	$('[type=reset]')	Selects all the reset buttons
$(:selected)	No Equivalent	Selects all the elements that are currently selected of type <option>. It does not include checkboxes or radio buttons.
$(:submit)	$('[type=submit]')	Selects all submit input elements
$(:text)	No Equivalent	Selects all input elements of type text. $('[type=text]') is almost the same, except that $(:text) includes <input> fields with no type specified.

jQuery Attributes

Back to HTML now.

In order to understand how to fully manipulate the elements you now have access to, one must understand an element's *attributes* and *properties*.

The core set of attributes related to DOM elements are the ones specified in the HTML tags.

- The *href* attribute of an <a> tag
- The *src* attribute of an <img>
- The *class* attribute of most elements

jQuery Attributes

And some examples

In jQuery we can both set and get an attribute value by using the **attr()** method on any element from a selector.

```
// var link is assigned the href attribute of the first <a> tag
var link = $("a").attr("href");

// change all links in the page to http://funwebdev.com
$("a").attr("href","http://funwebdev.com");

// change the class for all images on the page to fancy
$("img").attr("class","fancy");
```

HTML Properties

Full circle

- Many HTML tags include *properties* as well as attributes, the most common being the *checked* property of a radio button or checkbox.

- The prop() method is the preferred way to retrieve and set the value of a property although, attr() may return some (less useful) values.

- <input class ="meh" type="checkbox" checked="checked">

- Is accessed by jQuery as follows:

- **var theBox = $(".meh");**
theBox.prop("checked"); *// evaluates to TRUE*
theBox.attr("checked"); *// evaluates to "checked"*

Changing CSS

With jQuery

- jQuery provides the extremely intuitive **css()** methods.
- To get a css value use the css() method with 1 parameter:

```
$color = $("#colourBox").css("background-color"); // get the color
```

- To set a CSS variable use css() with two parameters: the first being the CSS attribute, and the second the value.

```
// set color to red
    $("#colourBox").css("background-color", "#FF0000");
```

Shortcut Methods

With jQuery

- The **html()** method is used to get the HTML contents of an element. If passed with a parameter, it updates the HTML of that element.
- The **val()** method returns the value of the element.
- The shortcut methods **addClass**(className) / **removeClass**(className) add or remove a CSS class to the element being worked on. The className used for these functions can contain a space-separated list of classnames to be added or removed.
- The **hasClass**(classname) method returns true if the element has the className currently assigned. False, otherwise.

jQuery Listeners

Set up after page load

In JavaScript, you learned why having your **listeners** set up inside of the window.onload() event was a good practice.

With jQuery we do the same thing but use the $(document).ready() event

```
$(document).ready(function(){
 //set up listeners on the change event for the file items.
 $("input[type=file]").change(function(){
     console.log("The file to upload is "+ this.value);
  });
});
```

LISTING 15.6 jQuery code to listen for file inputs changing, all inside the document's ready event

jQuery Listeners

Listener Management

While pure JavaScript uses the addEventListener() method, jQuery has on() and off() methods as well as shortcut methods to attach events.

```
$(document).ready(function(){
   $(":file").on("change",alertFileName); // add listener
});
// handler function using this
function alertFileName() {
   console.log("The file selected is: "+this.value);
}
```

LISTING 15.7 Using the listener technique in jQuery with on and off methods

Modifying the DOM

Appending DOM Elements

- The append() method takes as a parameter an HTML string, a DOM object, or a jQuery object. That object is then added as the last child to the element(s) being selected.

HTML Before

```
<div class="external-links">
     <div class="linkOut">
        funwebdev.com
    </div>
     <div class="linkIn">
         /localpage.html
    </div>
     <div class="linkOut">
        pearson.com
    </div>
<div>
```

jQuery append

```
$(".linkOut").append(jsLink);
```

HTML After

```
<div class="external-links">
     <div class="linkOut">
        funwebdev.com
<a href='http://funwebdev.com'
title='jQuery'>Visit Us</a>
    </div>
     <div class="linkIn">
         /localpage.html
    </div>
     <div class="linkOut">
        pearson.com
<a href='http://funwebdev.com'
title='jQuery'>Visit Us</a>
    </div>
<div>
```

Modifying the DOM

Prepending DOM Elements

The prepend() and prependTo() methods operate in a similar manner except that they add the new element as the first child rather than the last.

HTML Before

```
<div class="external-links">
     <div class="linkOut">
        funwebdev.com
    </div>
     <div class="linkIn">
         /localpage.html
    </div>
     <div class="linkOut">
        pearson.com
    </div>
<div>
```

jQuery append

```
$(".linkOut").prepend(jsLink);
```

HTML After

```
<div class="external-links">
     <div class="linkOut">
<a href='http://funwebdev.com'
title='jQuery'>Visit Us</a>
        funwebdev.com
    </div>
     <div class="linkIn">
         /localpage.html
    </div>
     <div class="linkOut">
<a href='http://funwebdev.com'
title='jQuery'>Visit Us</a>
        pearson.com
    </div>
<div>
```

Modifying the DOM

Wrapping Existing DOM in New Tags

❖A more advanced technique might make use of the content of each div being modified. In that case we use a callback function in place of a simple element.

❖The wrap() method is a callback function, which is called for each element in a set (often an array).

❖Each element then becomes this for the duration of one of the wrap() function's executions, allowing the unique title attributes as shown in Listing 15.12.

Modifying the DOM

Wrapping Existing DOM in New Tags

```
<div class="external-links">
 <div class="gallery">Uffuzi Museum</div>
 <div class="gallery">National Gallery</div>
 <div class="link-out">funwebdev.com</div>
</div>
```

```
$(".gallery").wrap('<div class="galleryLink"/>');
```

```
<div class="external-links">
  <div class="galleryLink">
      <div class="gallery">Uffuzi Museum</div>
  </div>
   <div class="galleryLink">
      <div class="gallery">National Gallery</div>
  </div>
   <div class="link-out">funwebdev.com</div>
</div>
```

LISTING 15.10 HTML from Listing 15.9 modified by executing the wrap statement above

Section 3 of 6

AJAX

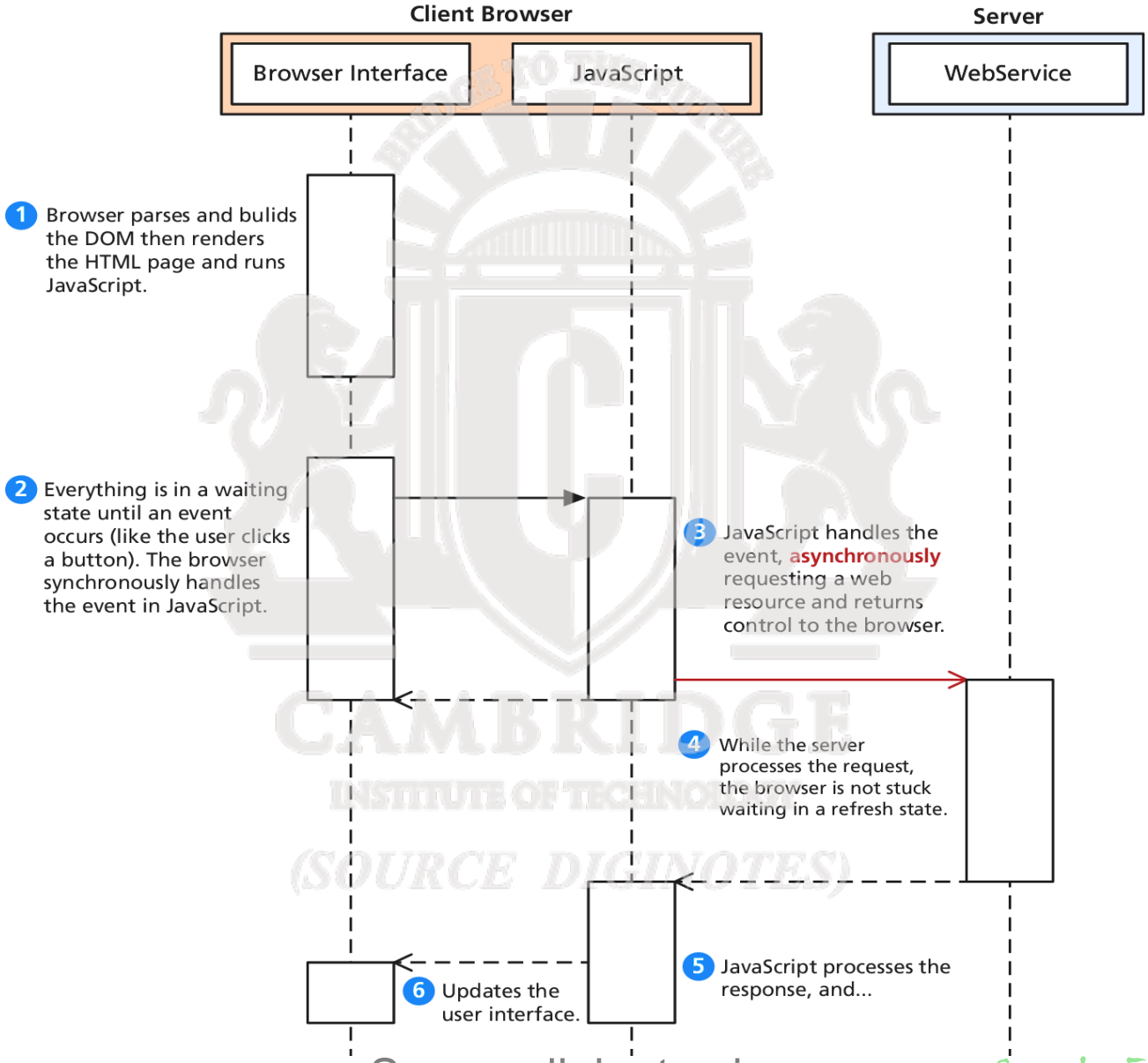
AJAX

Asynchronous JavaScript with XML

- Asynchronous JavaScript with XML (AJAX) is a term used to describe a paradigm that allows a web browser to send messages back to the server without interrupting the flow of what's being shown in the browser.

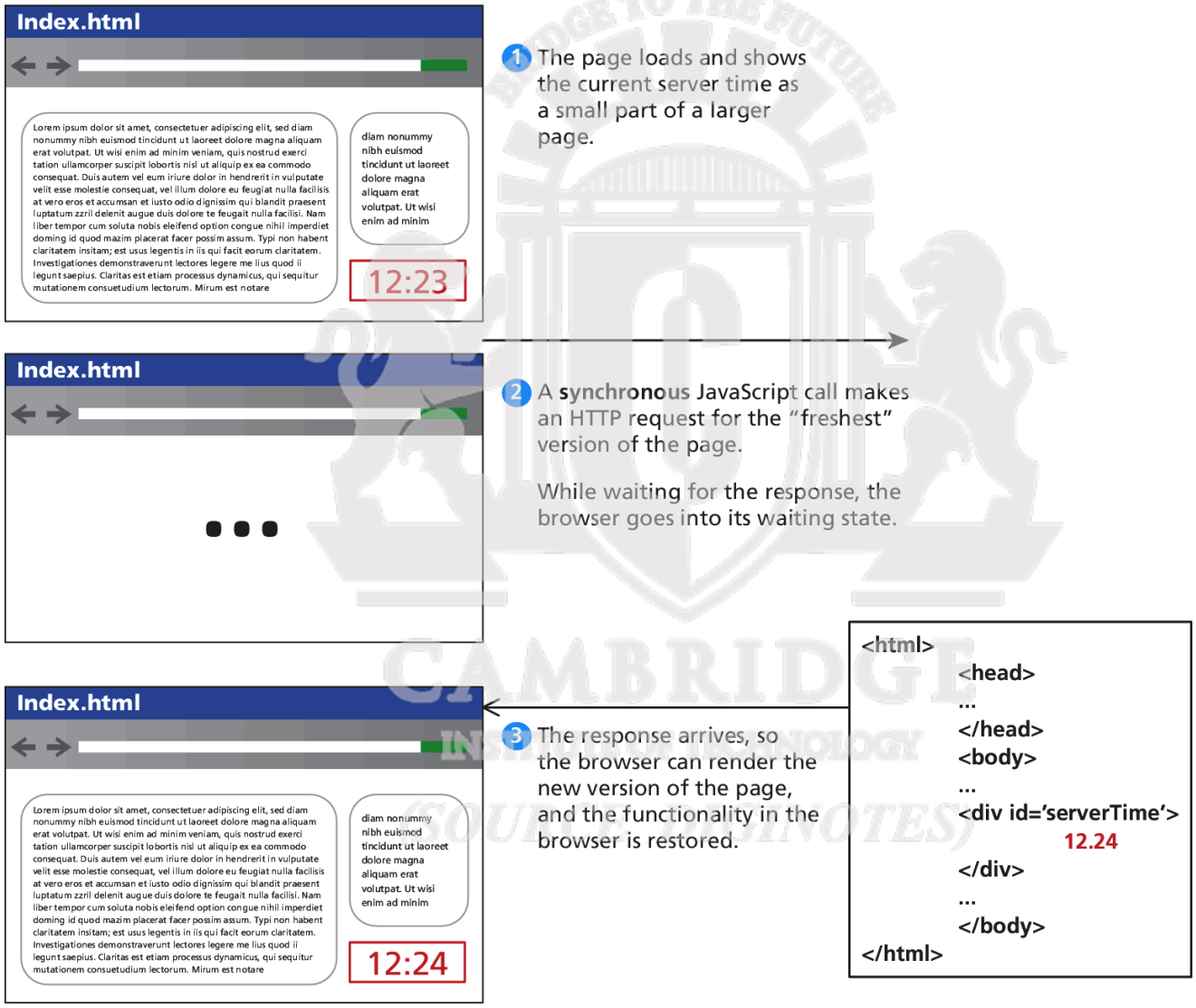
AJAX

UML of an asynchronous request



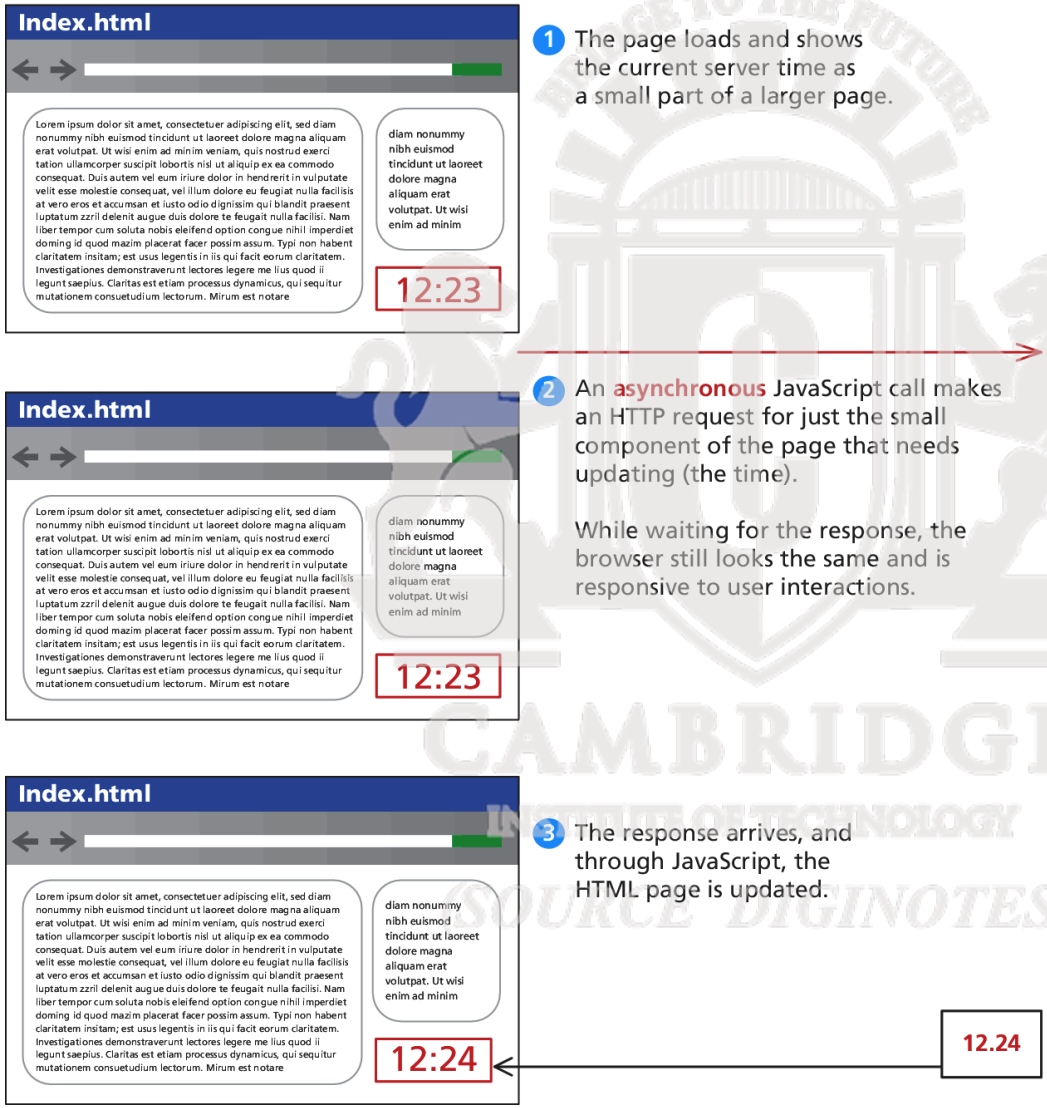
AJAX

Consider a webpage that displays the server's time



AJAX

Consider a webpage that displays the server's time



AJAX

Making Asynchronous requests

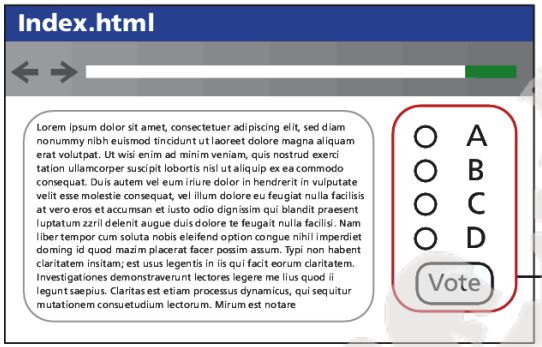
•jQuery provides a family of methods to make asynchronous requests. We will start simple and work our way up.

•Consider the very simple server time example we just saw. If currentTime.php returns a single string and you want to load that value asynchronously into the element, you could write:

•**$("#timeDiv").load("currentTime.php");**

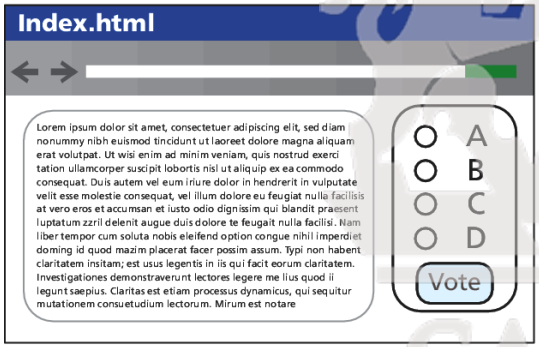
AJAX

GET Requests



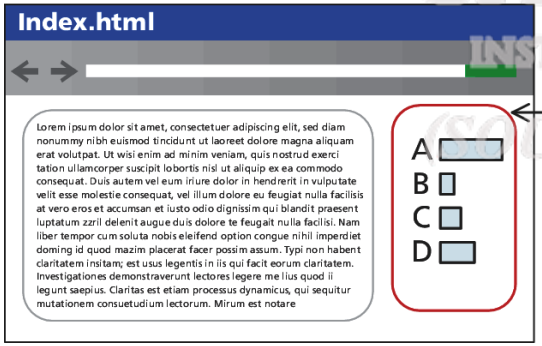
1 The HTML page contains a poll that posts votes asynchronously.

```
$.get("/vote.php?option=C");
```



2 An **asynchronous** vote submits the user's choice.

Meanwhile, the browser remains interactive while the request is processed on the server.



3 The response arrives, and is handled by JavaScript, which uses the response data to update the interface to show poll results.

AJAX

GET Requests – formal definition

jQuery.get (**url [, data] [, success(data, textStatus, jqXHR)] [, dataType]**)

- **url** is a string that holds the location to send the request.
- **data** is an optional parameter that is a query string or a *Plain Object.*
- **success(data,textStatus,jqXHR)** is an optional *callback* function that executes when the response is received.
  - **data** holding the body of the response as a string.
  - **textStatus** holding the status of the request (i.e., "success").
  - **jqXHR** holding a jqXHR object, described shortly.
- **dataType** is an optional parameter to hold the type of data expected from the server.

AJAX

GET Requests – an example

```
$.get("/vote.php?option=C", function(data,textStatus,jsXHR) {
  if (textStatus=="success") {
     console.log("success! response is:" + data);
  }
  else {
     console.log("There was an error code"+jsXHR.status);
  }
  console.log("all done");
});
```

LISTING 15.13 jQuery to asynchronously get a URL and outputs when the response arrives

AJAX

The jqXHR Object

- All of the $.get() requests made by jQuery return a **jqXHR** object to encapsulate the response from the server.
- In practice that means the data being referred to in the callback from Listing 15.13 is actually an object with backward compatibility with XML HttpRequest.

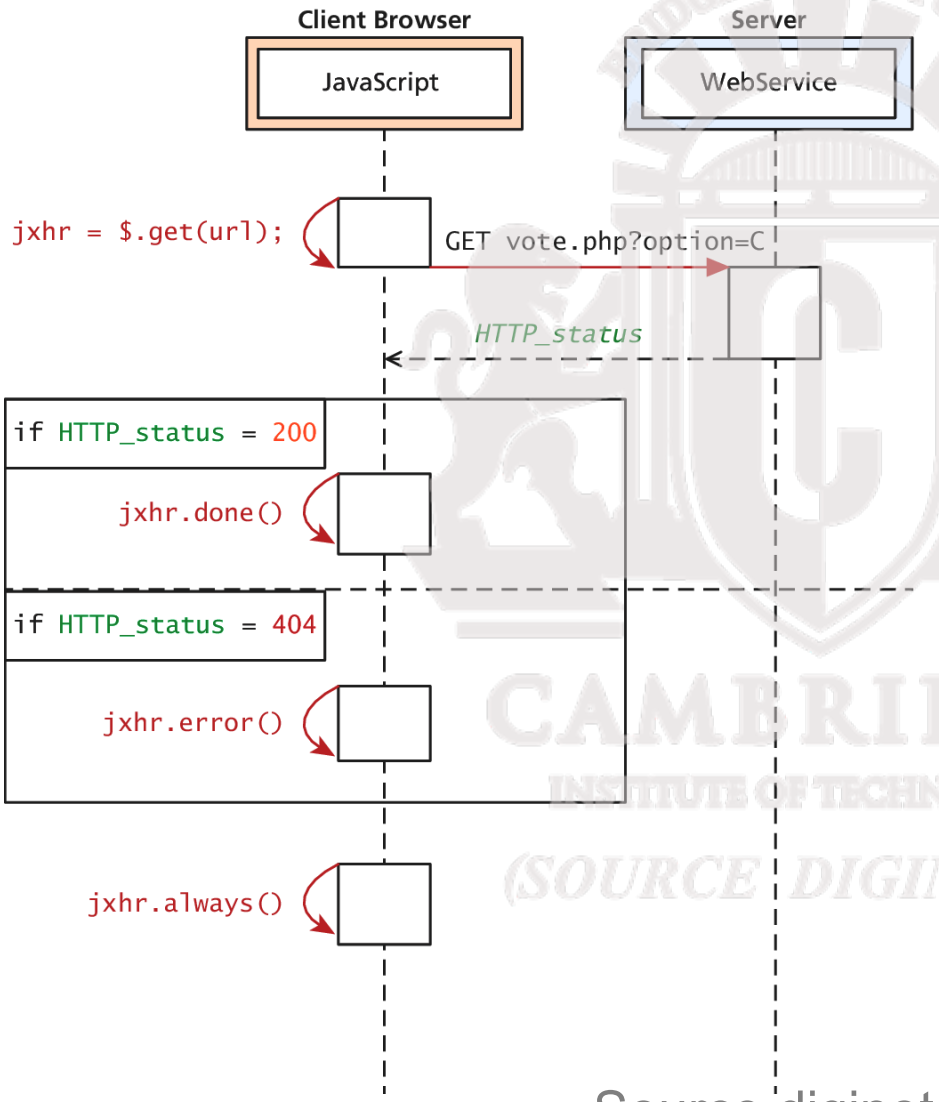
AJAX

jqXHR - XMLHttpRequest compatibility

- **abort()** stops execution and prevents any callback or handlers from receiving the trigger to execute.
- **getResponseHeader()** takes a parameter and gets the current value of that header.
- **readyState** is an integer from 1 to 4 representing the state of the request. The values include 1:successful call open() 2: sending, 3: response being processed, and 4: completed.
- **responseXML** and/or **responseText** the main response to the request.
- **setRequestHeader(name, value)** when used before actually instantiating the request allows headers to be changed for the request.
- **status** is the HTTP request status codes (200 = ok)
- **statusText** is the associated description of the status code.

jqXHR

Actually quite easy to use



jqXHR objects have methods

- done()
- fail()
- always()

which allow us to structure our code in a more modular way than the inline callback

POST requests

Via jQuery AJAX

- POST requests are often preferred to GET requests because one can post an unlimited amount of data, and because they do not generate viewable URLs for each action.
- GET requests are typically not used when we have forms because of the messy URLs and that limitation on how much data we can transmit.
- With POST it is possible to transmit files, something which is not possible with GET.

POST requests

Via jQuery AJAX

•The HTTP 1.1 definition describes GET as a **safe method** meaning that they should not change anything, and should only read data.

•POSTs on the other hand are not safe, and should be used whenever we are changing the state of our system (like casting a vote). get() method.

•POST syntax is almost identical to GET.

•jQuery.**post** (url [, data] [, success(data, textStatus, jqXHR)] [, dataType])

POST requests

Via jQuery AJAX

If we were to convert our vote casting code it would simply change the first line from

```
var jqxhr = $.get("/vote.php?option=C");
```

to

```
var jqxhr = $.post("/vote.php", "option=C");
```

POST requests

Serialize() will seriously help

serialize() can be called on any form object to return its current key-value pairing as an & separated string, suitable for use with post().

```
var postData = $("#voteForm").serialize();

$.post("vote.php", postData);
```

Ajax

You have complete control

- It turns out both the $.get() and $.post() methods are actually shorthand forms for the jQuery().ajax() method
- The ajax() method has two versions. In the first it takes two parameters: a URL and a Plain Object, containing any of over 30 fields.
- A second version with only one parameter is more commonly used, where the URL is but one of the key-value pairs in the Plain Object.

Ajax

More verbose

The one line required to post our form using get() becomes the more verbose code

```
$.ajax({ url: "vote.php",
         data: $("#voteForm").serialize(),
         async: true,
         type: post
});
```

LISTING 15.15 A raw AJAX method code to make a post

Ajax

You have complete control

To pass HTTP headers to the ajax() method, you enclose as many as you would like in a Plain Object. To illustrate how you could override User-Agent and Referer headers in the POST

```
$.ajax({ url: "vote.php",
  data: $("#voteForm").serialize(),
  async: true,
  type: post,
  headers: {"User-Agent" : "Homebrew JavaScript Vote Engine agent",
            "Referer": "http://funwebdev.com"
           }
});
```

LISTING 15.16 Adding headers to an AJAX post in jQuery

CORS

Cross Origin Resource Sharing

- Since modern browsers prevent cross-origin requests by default (which is good for security), sharing content legitimately between two domains becomes harder.
- **Cross-origin resource sharing (CORS)** uses new headers in the HTML5 standard implemented in most new browsers.
- If a site wants to allow any domain to access its content through JavaScript, it would add the following header to all of its responses.
- **Access-Control-Allow-Origin: ***

CORS

Cross Origin Resource Sharing

✓A better usage is to specify specific domains that are allowed, rather than cast the gates open to each and every domain. To allow our domain to make cross site requests we would add the header:

✓Access-Control-Allow-Origin: www.funwebdev.com

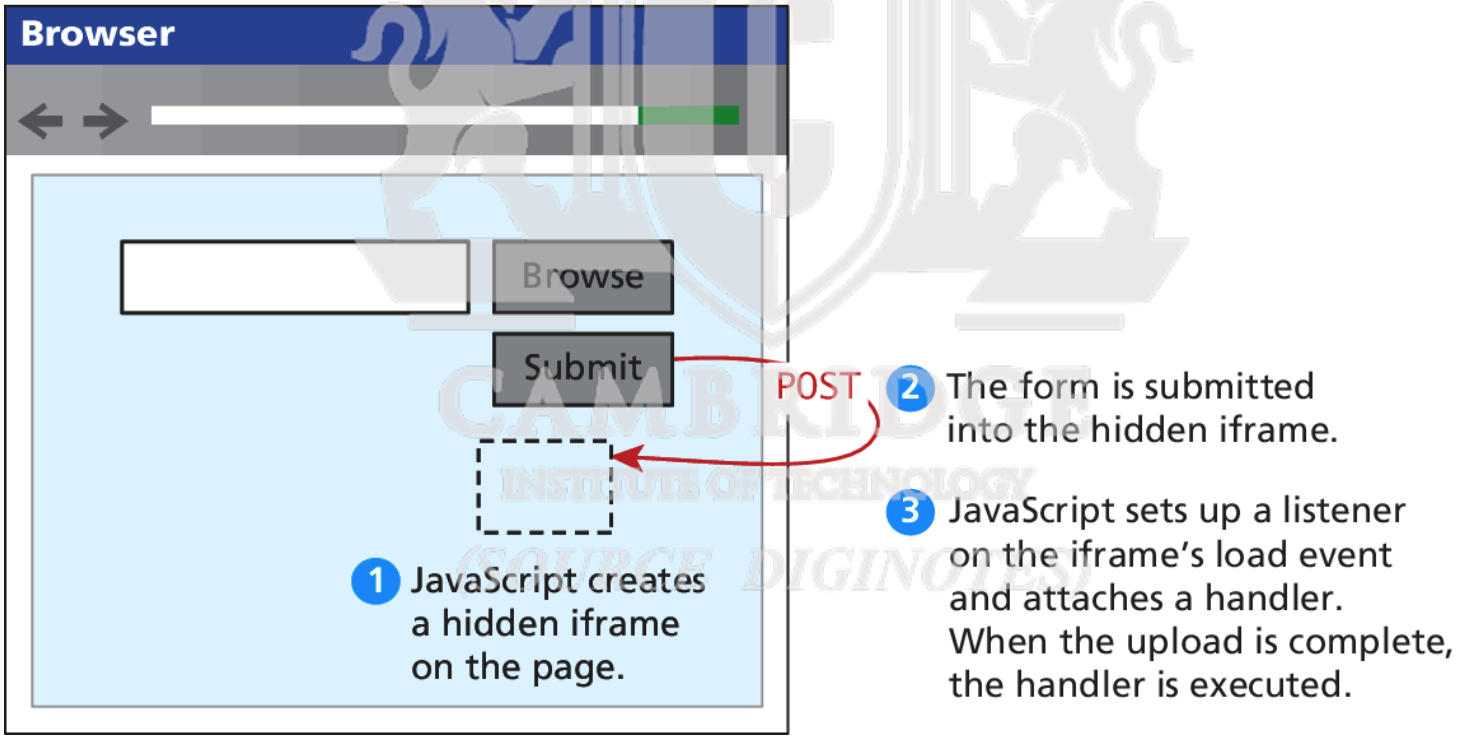
✓Rather than the wildcard *.

ASYNCHRONOUS FILE TRANSMISSION

Asynchronous File Transmission

The old iFrame technique

The original workaround to allow the asynchronous posting of files was to use a hidden <iframe> element to receive the posted files.



Asynchronous File Transmission

A Primer

Consider a simple form as defined below:

```
<form name="fileUpload" id="fileUpload" enctype="multipart/form-data"
      method="post" action="upload.php">
<input name="images" id="images" type="file" multiple />
<input type="submit" name="submit" value="Upload files!"/>
</form>
```

LISTING 15.17 Simple file upload form

Asynchronous File Transmission

The old iFrame technique

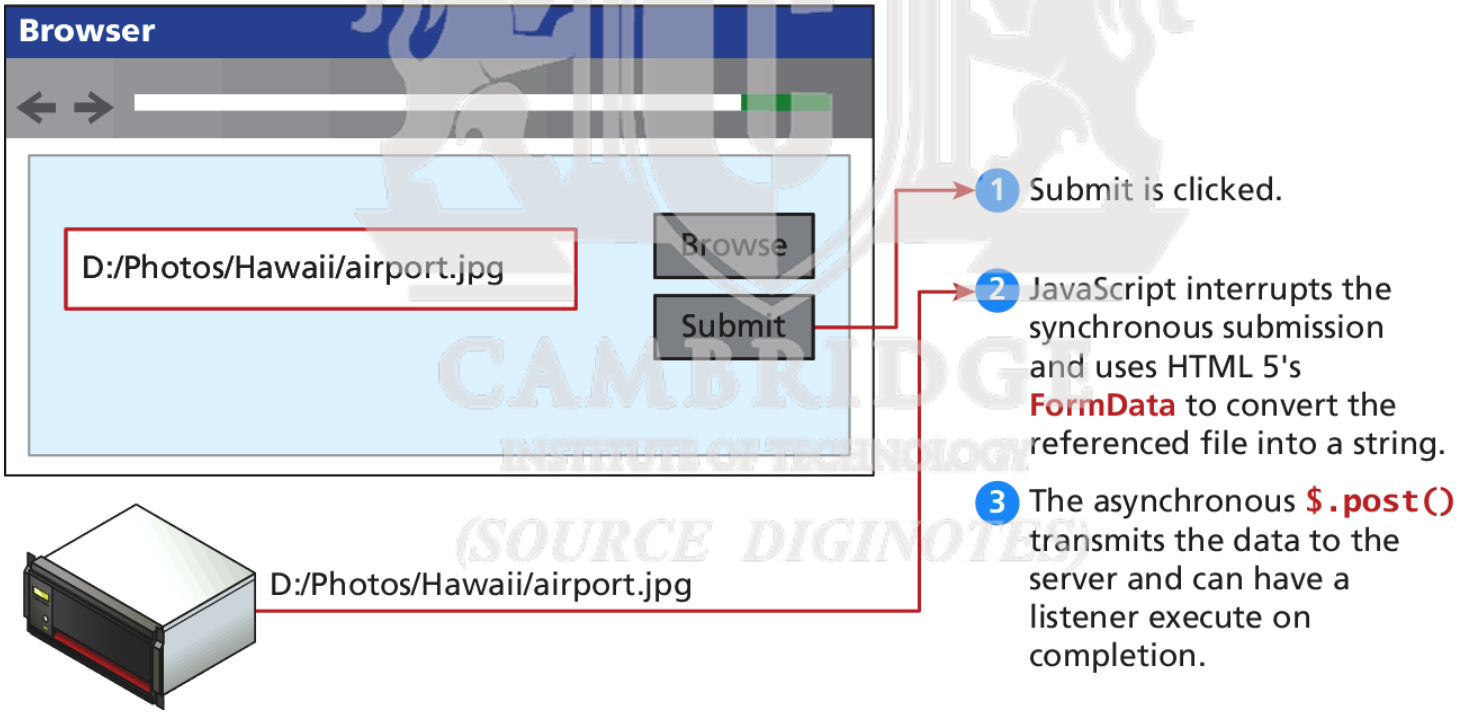
```
$(document).ready(function() {
  // set up listener when the file changes
  $(":file").on("change",uploadFile);
  // hide the submit buttons
  $("input[type=submit]").css("display","none");
});

// function called when the file being chosen changes
function uploadFile () {
  // create a hidden iframe
  var hidName = "hiddenIFrame";
  $("#fileUpload").append("<iframe id='"+hidName+"' name='"+hidName+"'
  style='display:none' src='#' ></iframe>");

  // set form's target to iframe
  $("#fileUpload").prop("target",hidName);
  // submit the form, now that an image is in it.
  $("#fileUpload").submit();
  // Now register the load event of the iframe to give feedback
  $('#'+hidName).load(function() {
  var link = $(this).contents().find('body')[0].innerHTML;
  // add an image dynamically to the page from the file just uploaded
  $("#fileUpload").append("<img src='"+link+"' />");
  });
}
```

LISTING 15.18 Hidden iFrame technique to upload files

Asynchronous File Transmission

New Form Data technique

•Using the **FormData** interface and File API, which is part of HTML5, you no longer have to trick the browser into posting your file data asynchronously.



Asynchronous File Transmission

New Form Data technique

```
function uploadFile () {
  // get the file as a string
  var formData = new FormData($("#fileUpload")[0]);

  var xhr = new XMLHttpRequest();
  xhr.addEventListener("load", transferComplete, false);
  xhr.addEventListener("error", transferFailed, false);
  xhr.addEventListener("abort", transferCanceled, false);

  xhr.open('POST', 'upload.php', true);
  xhr.send(formData);                    // actually send the form data

  function transferComplete(evt) {   // stylized upload complete
     $("#progress").css("width","100%");
     $("#progress").html("100%");
  }

  function transferFailed(evt) {
     alert("An error occurred while transferring the file.");
  }

  function transferCanceled(evt) {
     alert("The transfer has been canceled by the user.");
  }
}
```

LISTING 15.19 Using the new FormData interface from the XHR2 Specification to post files asychronously

Asynchronous File Transmission

Advanced modern technique

- When we consider uploading multiple files, you may want to upload a single file, rather than the entire form every time. To support that pattern, you can access a single file and post it by appending the raw file to a FormData object.

- The advantage of this technique is that you submit each file to the server asynchronously as the user changes it; and it allows multiple files to be transmitted at once.

Asynchronous File Transmission

Advanced modern technique

```
var xhr = new XMLHttpRequest();
// reference to the 1st file input field
var theFile = $(":file")[0].files[0];
var formData = new FormData();
formData.append('images', theFile);
```

LISTING 15.20 Posting a single file from a form

```
var allFiles =  $(":file")[0].files;
for (var i=0;i<allFiles.length;i++) {
  formData.append('images[]', allFiles[i]);
}
```

LISTING 15.21 Looping through multiple files in a file input and appending the data for posting

Section 5 of 6

ANIMATION

Animation

Hide() and Show()

The hide() and show() methods can be called with no arguments to perform a default animation. Another version allows two parameters: the duration of the animation (in milliseconds) and a callback method to execute on completion.

Animation

fadeIn() and fadeOut()

The fadeIn() and fadeOut() shortcut methods control the opacity of an element. The parameters passed are the duration and the callback, just like hide() and show(). Unlike hide() and show(), there is no scaling of the element, just strictly control over the transparency.

Show email

Animation

SlideUp() and SlideDown()

slideUp() and slideDown() do not touch the opacity of an element, but rather gradually change its height.

email icon from **http://openiconlibrary.sourceforge.net.**

Animation

Toggle()

As you may have seen, the shortcut methods come in pairs, which make them ideal for toggling between a shown and hidden state. Using a toggle method means you don't have to check the current state and then conditionally call one of the two methods;

- To toggle between the visible and hidden states you can use the **toggle()** methods.
- To toggle between fading in and fading out, use the **fadeToggle()** method
- To toggle between the two sliding states can be achieved using the **slideToggle()** method.

Raw Animation

Full control

❖The animate() method has several versions, but the one we will look at has the following form:

❖.animate(properties, options);

❖The properties parameter contains a Plain Object with all the CSS styles of the final state of the animation.

❖The options parameter contains another Plain Object with any of the following options set:

Raw Animation

Options parameter

- **always** is the function to be called when the animation completes or stops with a fail condition. This function will always be called (hence the name).
- **done** is a function to be called when the animation completes.
- **duration** is a number controlling the duration of the animation.
- **fail** is the function called if the animation does not complete.
- **progress** is a function to be called after each step of the animation.

Raw Animation

Options parameter

- **queue** is a Boolean value telling the animation whether to wait in the queue of animations or not. If false, the animation begins immediately.
- **step** is a function you can define that will be called periodically while the animation is still going.
- Advanced options called **easing** and **specialEasing** allow for advanced control over the speed of animation.

Raw Animation

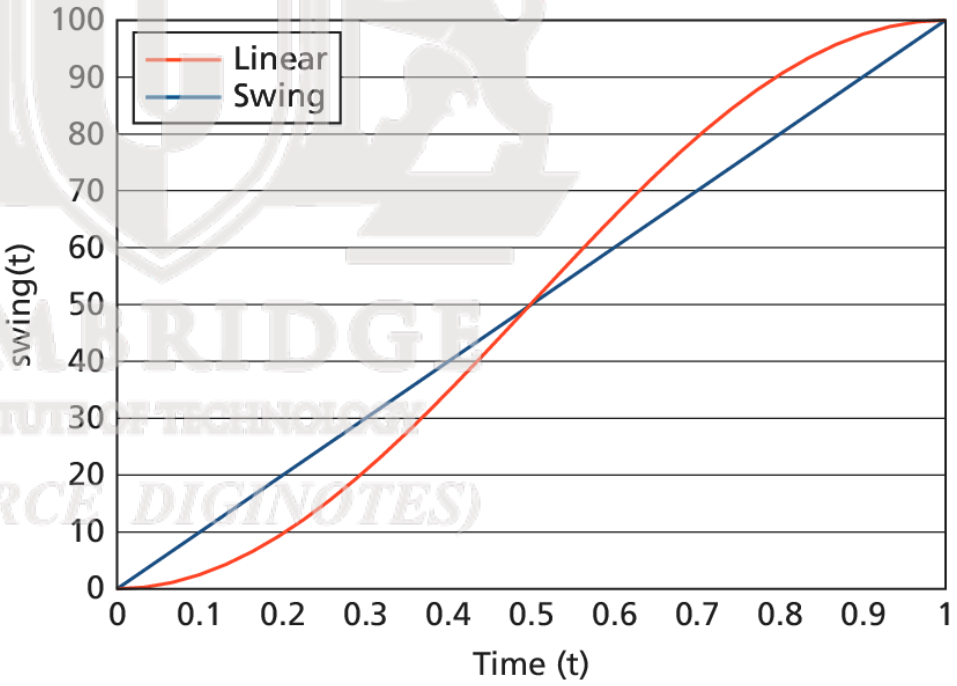
Easing functions – advanced animation

In web development, **easing functions** are used to simulate that natural type of movement. They are mathematical equations that describe how fast or slow the transitions occur at various points during the animation.

Included in jQuery are

- linear
- swing

easing functions.



Raw Animation

Easing functions – advanced animation

- For example, the function defining swing for values of time *t* between 0 and 1 is

$$\text{swing}(t) = -\frac{1}{2}\cos(t\pi) + 0.5$$

- The jQuery UI extension provides over 30 easing functions, including cubic functions and bouncing effects, so you should not have to define your own.

Advanced example

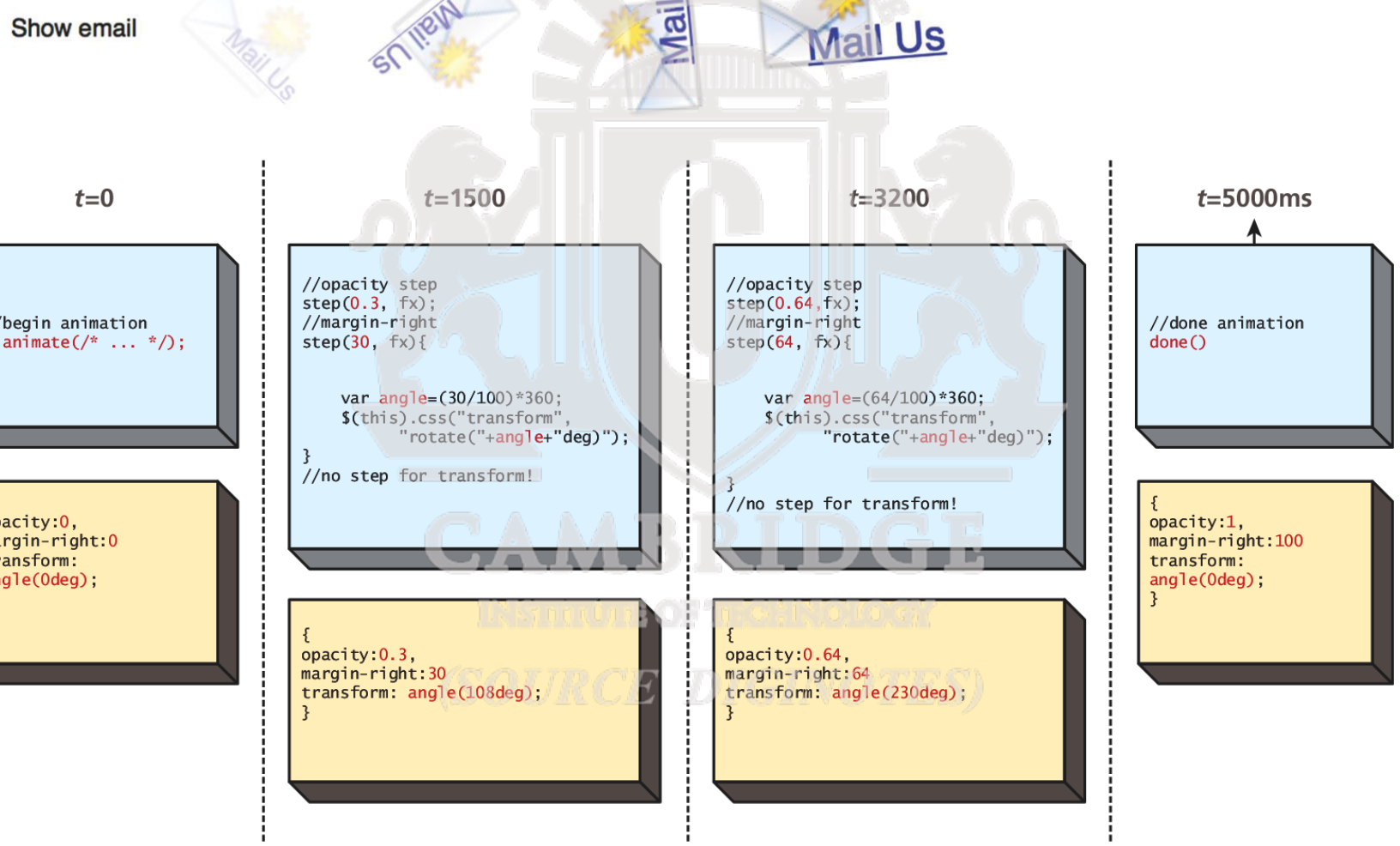
rotating

```
$(this).animate(
  // parameter one: Plain Object with CSS options.

  {opacity:"show","fontSize":"120%","marginRight":"100px"},
  // parameter 2: Plain Object with other options including a
  // step function
  {step: function(now, fx) {
        // if the method was called for the margin property
        if (fx.prop=="marginRight") {
           var angle=(now/100)*360; //percentage of a full circle
           // Multiple rotation methods to work in multiple browsers
           $(this).css("transform","rotate("+angle+"deg)");
           $(this).css("-webkit-transform","rotate("+angle+"deg)");
           $(this).css("-ms-transform","rotate("+angle+"deg)");
        }
     },
     duration:5000, "easing":"linear"
  }
);
```

LISTING 15.23 Use of animate() with a step function to do CSS3 rotation

Raw Animation

Rotating



Section 6 of 6

BACKBONE MVC FRAMEWORKS

Backbone

Another framework

•Backbone is an MVC framework that further abstracts JavaScript with libraries intended to adhere more closely to the MVC model

•This library is available from **http://backbonejs.org** and relies on the underscore library, available from **http://underscorejs.org/**.

•Include with:

•<script src="underscore-min.js"></script>

•<script src="backbone-min.js"></script>

Backbone

Models

- In Backbone, you build your client scripts around the concept of **models**.
- Backbone.js defines **models** as *the heart of any JavaScript application, containing the interactive data as well as a large part of the logic surrounding it: conversions, validations, computed properties, and access control.*
- The Models you define using Backbone must *extend* Backbone.Model

Backbone

Collections

✓In addition to models, Backbone introduces the concept of **Collections**, which are normally used to contain lists of Model objects.

✓These collections have advanced features and like a database can have indexes to improve search performance.

✓A collection is defined by extending from Backbone's Collection object.

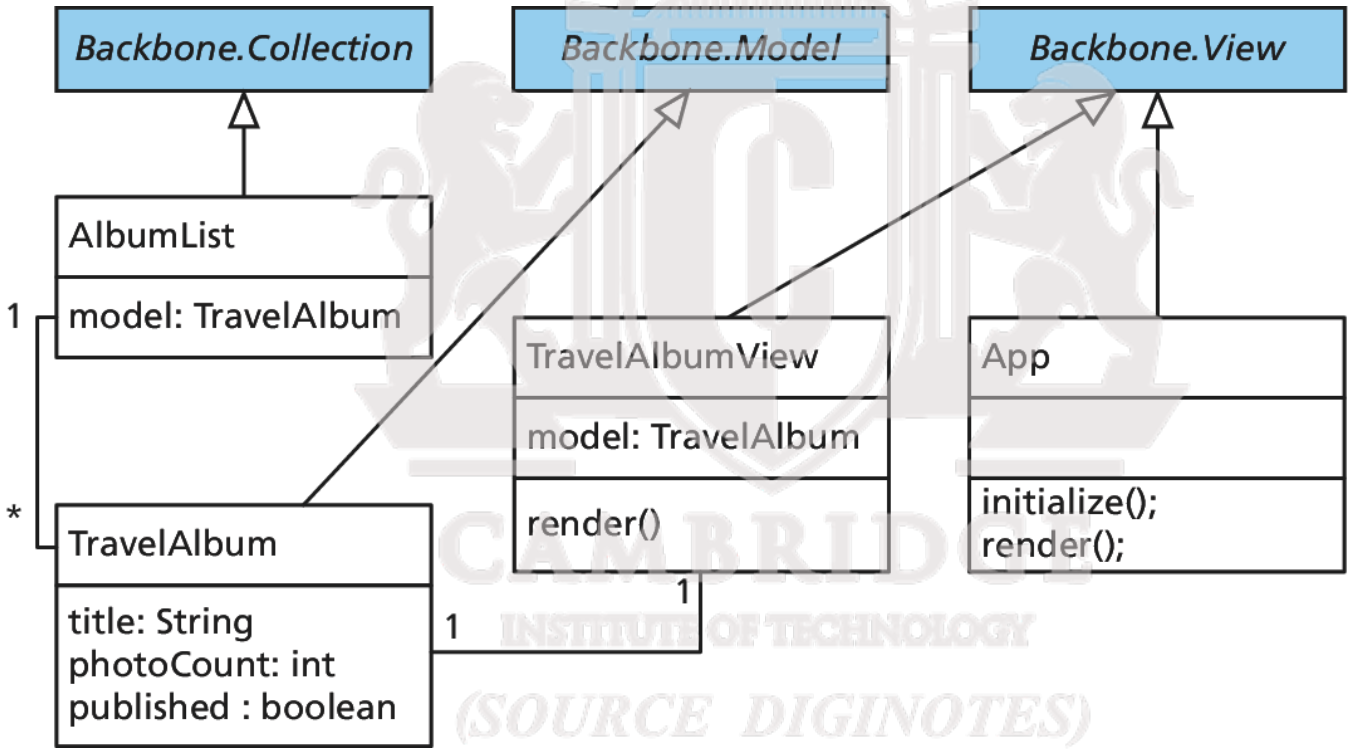
Backbone

Views

- Views allow you to translate your models into the HTML that is seen by the users.
- They attach themselves to methods and properties of the Collection and define methods that will be called whenever Backbone determines the view needs refreshing.
- You must always override the render() method since it defines the HTML that is output.

Backbone

Example



Backbone

A Model Example

```
// Create a model for the albums
var TravelAlbum = Backbone.Model.extend({
   defaults:{
            title: 'NewAlbum',
            photoCount: 0,
            published: false
        },

        // Function to publish/unpublish
        toggle: function(){
            this.set('checked', !this.get('checked'));
        }
    });
```

LISTING 15.25 A PhotoAlbum Model extending from Backbone.Model

Backbone

A Collection Example

```
// Create a collection of albums
var AlbumList = Backbone.Collection.extend({

  // Set the model type for objects in this Collection
  model: TravelAlbum,

  // Return an array only with the published albums
  GetChecked: function(){
     return this.where({checked:true});
  }
});

// Prefill the collection with some albums.
var albums = new AlbumList([
  new TravelAlbum({ title: 'Banff, Canada', photoCount: 42}),
  new TravelAlbum({ title: 'Santorini, Greece', photoCount: 102}),
]);
```

LISTING 15.26 Demonstration of a Backbone.js Collection defined to hold PhotoAlbums

Backbone

A View Example

```
var TravelAlbumView = Backbone.View.extend({
  TagName: 'li',

  events:{
     'click': 'toggleAlbum'
  },

  initialize: function(){
      // Set up event listeners attached to change
     this.listenTo(this.model, 'change', this.render);
  },

  render: function(){
  // Create the HTML
  this.$el.html('<input type="checkbox" value="1" name="' +
                this.model.get('title') + '" /> ' +
                this.model.get('title') + '<span> ' +
                this.model.get('photoCount') + ' images</span>');
  this.$('input').prop('checked', this.model.get('checked'));

  // Returning the object is a good practice
     return this;
  },

  toggleAlbum: function() {
     this.model.toggle();
  }
});
```

LISTING 15.27 Deriving custom View objects for our model and Collection

Backbone

Bring it all together

```
// The main view of the entire Backbone application
var App = Backbone.View.extend({
  // Base the view on an existing element
  el: $('body'),

  initialize: function() {
     // Define required selectors
     this.total = $('#totalAlbums span');
     this.list = $('#albums');

     // Listen for the change event on the collection.
     this.listenTo(albums, 'change', this. render);

     // Create views for every one of the albums in the collection
     albums.each(function(album) {
        var view = new TravelAlbumView({ model: album });
              this.list.append(view.render().el);
       }, this);   // "this" is the context in the callback
     },

     render: function(){

        // Calculate the count of published albums and photos
           var total = 0; var photos = 0;

          _.each(albums.getChecked(), function(elem) {
            total++;
            photos+= elem.get("photoCount");
          });

        // Update the total price
        this.total.text(total+' Albums ('+photos+' images)');
        return this;
     }
});

new App(); // create the main app
```

LISTING 15.28 Defining the main app's view and making use of the Collections and models defined earlier

XML Processing and Web Services

Chapter 17

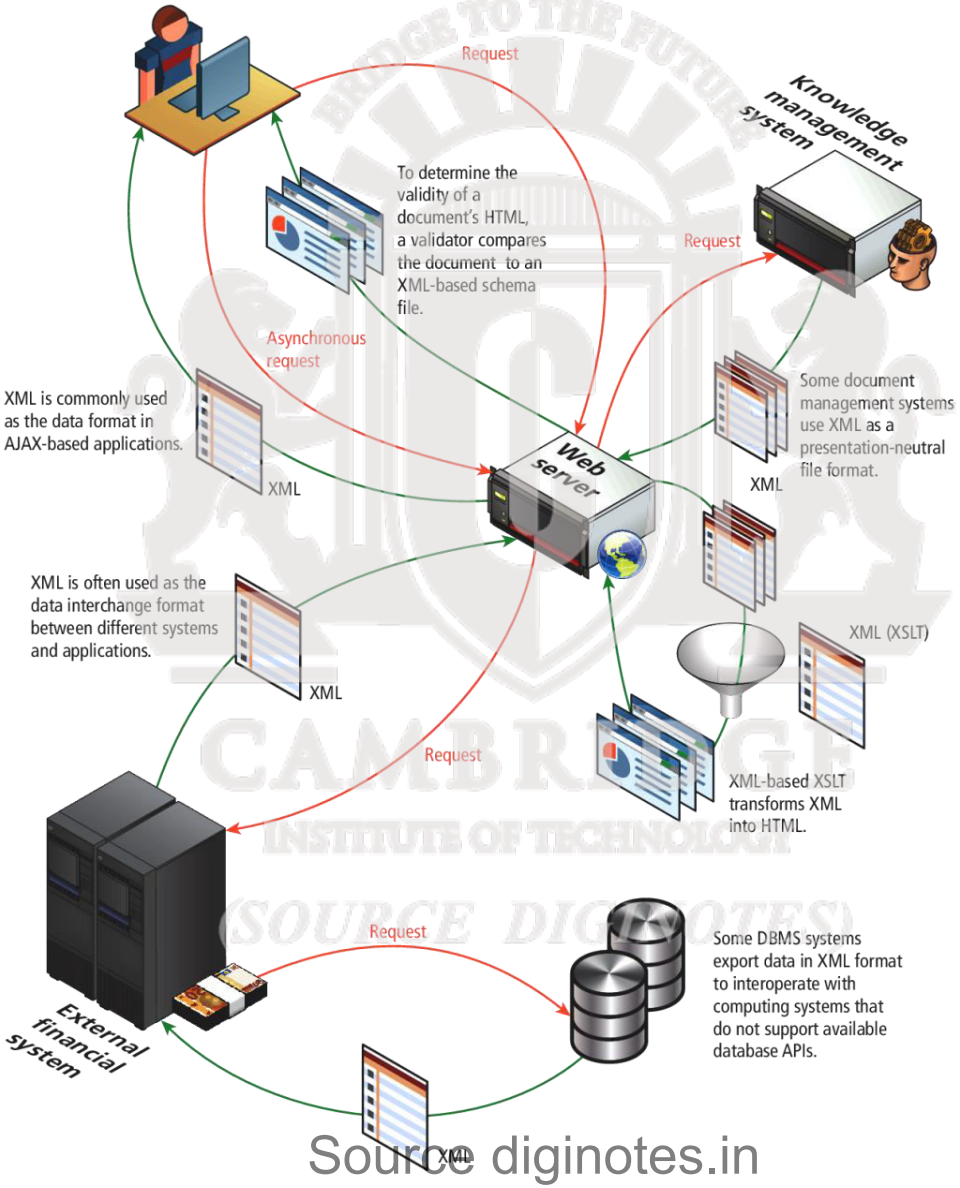
XML OVERVIEW

XML Overview

- XML is a markup language, but unlike HTML, XML can be used to mark up any type of data.
- Benefits of XML data is that as plain text, it can be read and transferred between applications and different operating systems as well as being human-readable.
- XML is used on the web server to communicate asynchronously with the browser
- Used as a data interchange format for moving information between systems

XML Overview

Used in many systems



Well Formed XML

For a document to be **well-formed XML**, it must follow the syntax rules for XML:

- Element names are composed of any of the valid characters (most punctuation symbols and spaces are not allowed) in XML.
- Element names can't start with a number.
- There must be a single-root element. A **root element** is one that contains all the other elements; for instance, in an HTML document, the root element is <html>.
- All elements must have a closing element (or be self-closing).
- Elements must be properly nested.
- Elements can contain attributes.
- Attribute values must always be within quotes.
- Element and attribute names are case sensitive.

Well Formed XML

Sample Document

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<art>
  <painting id="290">
    <title>Balcony</title>
    <artist>
      <name>Manet</name>
      <nationality>France</nationality>
    </artist>
    <year>1868</year>
    <medium>Oil on canvas</medium>
  </painting>
  <painting id="192">
    <title>The Kiss</title>
    <artist>
      <name>Klimt</name>
      <nationality>Austria</nationality>
    </artist>
    <year>1907</year>
    <medium>Oil and gold on canvas</medium>
  </painting>
  <painting id="139">
    <title>The Oath of the Horatii</title>
    <artist>
      <name>David</name>
      <nationality>France</nationality>
    </artist>
    <year>1784</year>
    <medium>Oil on canvas</medium>
  </painting>
</art>
```

LISTING 17.1 Sample XML document

Valid XML

Requires a DTD

•A **valid XML** document is one that is well formed and whose element and content conform to a document type definition (DTD) or its schema.

•A DTD tells the XML parser which elements and attributes to expect in the document as well as the order and nesting of those elements.

•A DTD can be defined within an XML document or within an external file.

Data Type Definition

Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE art [
<!ELEMENT art (painting*)>
<!ELEMENT painting (title,artist,year,medium)>
<!ATTLIST painting id CDATA #REQUIRED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT artist (name,nationality)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT nationality (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT medium (#PCDATA)>
]>
<art>
...
</art>
```

LISTING 17.2 Example DTD

The * allows zero or more occurences

Attributes are declared with ATTLIST

PCDATA – Parsed Character Data

Data Type Definition

Example

- The main drawback with DTDs is that they can only validate the existence and ordering of elements. They provide no way to validate the values of attributes or the textual content of elements.

- For this type of validation, one must instead use XML schemas, which have the added advantage of using XML syntax. Unfortunately, schemas have the corresponding disadvantage of being long-winded and harder for humans to read and comprehend; for this reason, they are typically created with tools.

XML Schema

```
<xs:schema attributeFormDefault="unqualified"
     elementFormDefault="qualified"
     xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="art">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="painting" maxOccurs="unbounded" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element type="xs:string" name="title"/>
              <xs:element name="artist">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element type="xs:string" name="name"/>
                    <xs:element type="xs:string" name="nationality"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element type="xs:short" name="year" />
              <xs:element type="xs:string" name="medium"/>
            </xs:sequence>
            <xs:attribute type="xs:short" name="id" use="optional"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

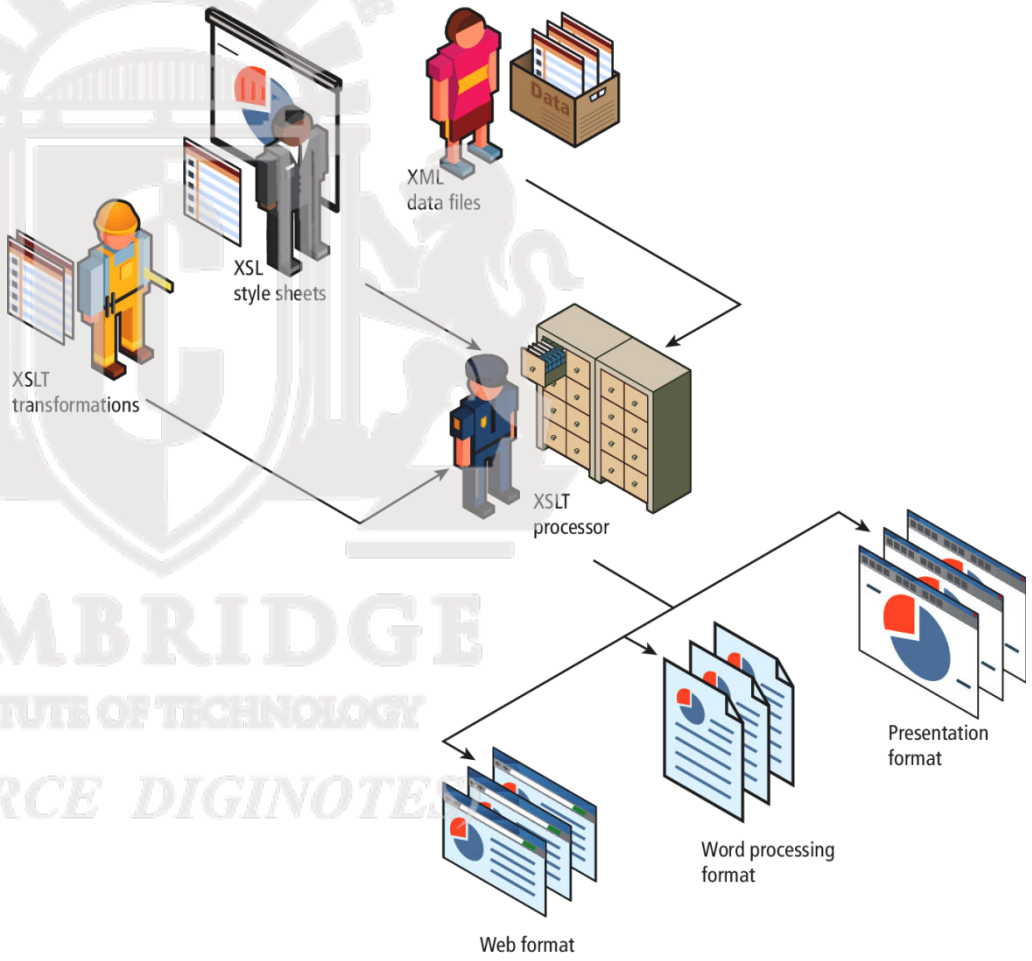
LISTING 17.3 Example schema

XSLT

XML Stylesheet Transformations

EXtensible Stylesheet Language Transformation

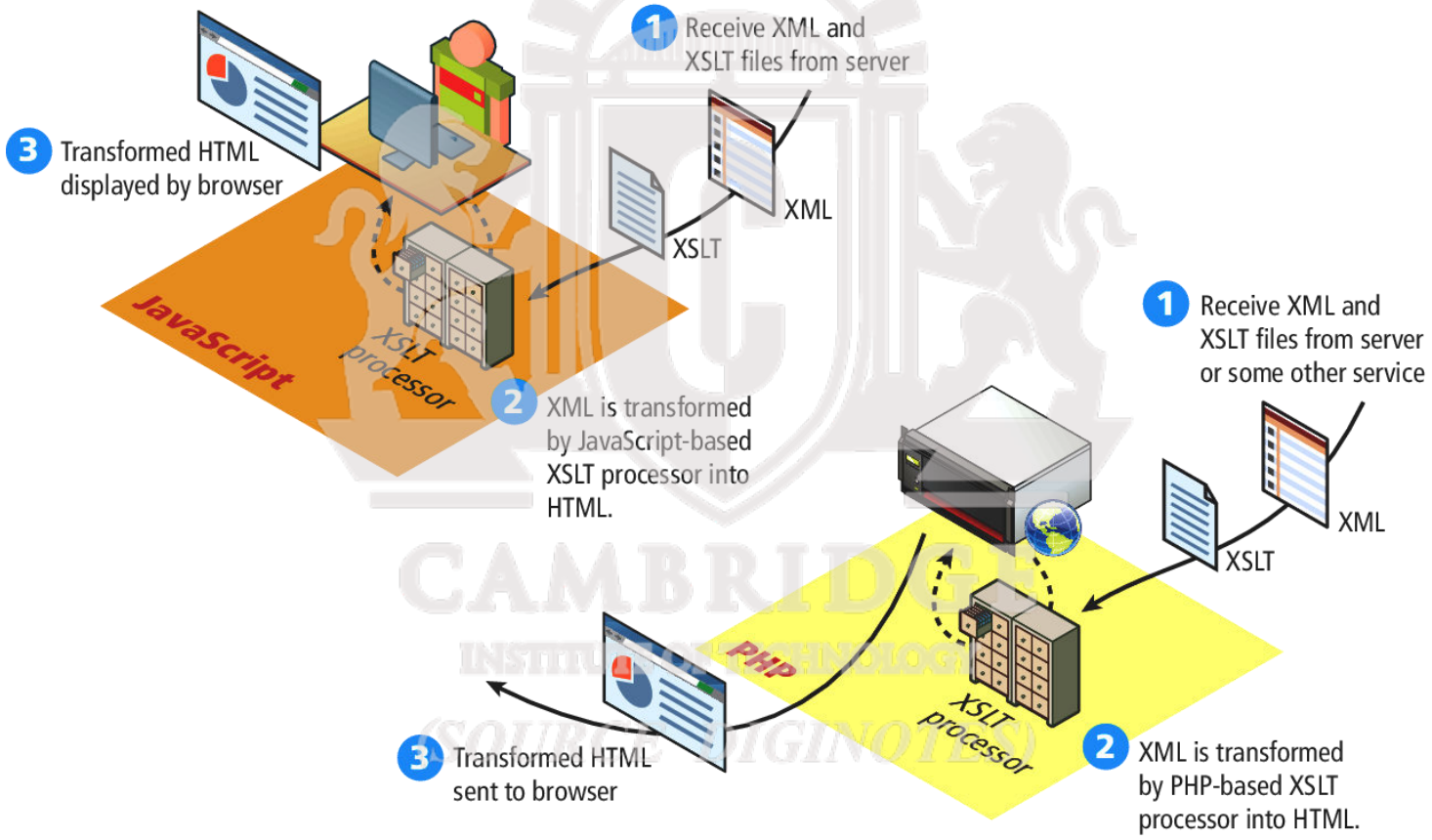
XSLT is an XML-based programming language that is used for transforming XML into other document formats



XSLT

Another usage

XSLT is also used on the server side and within JavaScript



XSLT

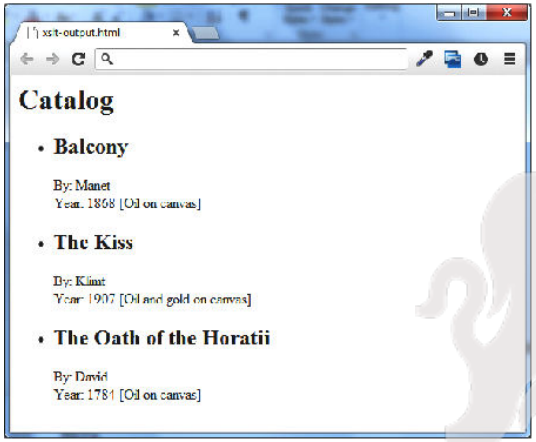
Example XSLT document that converts the XML from Listing 17.1 into an HTML list

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<html xsl:version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns="http://www.w3.org/1999/xhtml">
<body>
   <h1>Catalog</h1>
   <ul>
    <xsl:for-each select="/art/painting">
      <li>
        <h2><xsl:value-of select="title"/></h2>
        <p>By: <xsl:value-of select="artist/name"/><br/>
           Year: <xsl:value-of select="year"/>
           [<xsl:value-of select="medium"/>]</p>
      </li>
    </xsl:for-each>
   </ul>
</body>
</html>
```

LISTING 17.4 An example XSLT document

XSLT

An XML parser is still needed to perform the actual transformation




```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml">
<body>
<h1>Catalog</h1>
<ul>
  <li>
    <h2>Balcony</h2>
    <p>By: Manet<br/>
    Year: 1868 [Oil on canvas]</p>
  </li>
  <li>
    <h2>The Kiss</h2>
    <p>By: Klimt<br/>
    Year: 1907 [Oil and gold on canvas]</p>
  </li>
  <li>
    <h2>The Oath of the Horatii</h2>
    <p>By: David<br/>Year: 1784 [Oil on canvas]</p>
  </li>
</ul>
</body>
</html>
```

XPath

Another XML Technology

❖**XPath** is a standardized syntax for searching an XML document and for navigating to elements within the XML document

❖XPath is typically used as part of the programmatic manipulation of an XML document in PHP and other languages

❖XPath uses a syntax that is similar to the one used in most operating systems to access directories.

XPath

Learn through example

/art/painting[@id='192']/artist/name

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<art>
  <painting id="290">
    <title>Balcony</title>
    <artist>
      <name>Manet</name>
      <nationality>France</nationality>
    </artist>
    <year>1868</year>
    <medium>Oil on canvas</medium>
  </painting>
  <painting id="192">
    <title>The Kiss</title>
    <artist>
      <name>Klimt</name>
      <nationality>Austria</nationality>
    </artist>
    <year>1907</year>
    <medium>Oil and gold on canvas</medium>
  </painting>
  <painting id="139">
    <title>The Oath of the Horatii</title>
    <artist>
      <name>David</name>
      <nationality>France</nationality>
    </artist>
    <year>1784</year>
    <medium>Oil on canvas</medium>
  </painting>
</art>
```

/art/painting[year > 1800]

/art/painting[3]/@id

This used when you want to look for particular data, like just the artist's name for a particular painting

Section 2 of 7

XML PROCESSING

XML Processing

Two types

XML processing in PHP, JavaScript, and other modern development environments is divided into two basic styles:

- The **in-memory approach**, which involves reading the entire XML file into memory into some type of data structure with functions for accessing and manipulating the data.
- The **event or pull approach**, which lets you pull in just a few elements or lines at a time, thereby avoiding the memory load of large XML files.

XML Processing

In JavaScript

•All modern browsers have a built-in XML parser and their JavaScript implementations support an in-memory XML DOM API.

You can use the already familiar DOM functions such as

- getElementById(),
- getElementsByTagName()
- createElement()

to access and manipulate the data.

XML Processing

```
<script>
if (window.XMLHttpRequest)  {
  // code for IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
}
else  {
  // code for old versions of IE (optional you might just decide to
  // ignore these)
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
// load the external XML file
xmlhttp.open("GET","art.xml",false);
xmlhttp.send();
xmlDoc=xmlhttp.responseXML;

// now extract a node list of all <painting> elements
paintings = xmlDoc.getElementsByTagName("painting");
if (paintings) {
   // loop through each painting element
   for (var i = 0; i < paintings.length; i++)
   {
      // display its id attribute
      alert("id="+paintings[i].getAttribute("id"));

      // find its <title> element
      title = paintings[i].getElementsByTagName("title");
      if (title) {
         // display the text content of the <title> element
         alert("title="+title[0].textContent);
      }
   }
}
</script>
```

LISTING 17.5 Loading and processing an XML document via JavaScript

XML Processing

With JQuery

```
art = '<?xml version="1.0" encoding="ISO-8859-1"?>';
art += '<art><painting id="290"><title>Balcony … </art>';

// use jQuery parseXML() function to create the DOM object
xmlDoc = $.parseXML( art );
// convert DOM object to jQuery object
$xml = $( xmlDoc );

// find all the painting elements
$paintings = $xml.find( "painting" );
// loop through each painting element
$paintings.each(function() {
   // display its id
   alert($(this).attr("id"));
   // find the title element within the current painting element
   $title = $(this).find( "title" );
   // and display its content
   alert( $title.text() );
});
```

LISTING 17.6 XML processing using jQuery

XML Processing

With PHP

PHP provides several extensions or APIs for working with XML including:

- The **SimpleXML** extension which loads the data into an object that allows the developer to access the data via array properties and modifying the data via methods.
- The **XMLReader** is a read-only pull-type extension that uses a cursor-like approach similar to that used with database processing

XML Processing

With PHP using Simple XML

```php
<?php

$filename = 'art.xml';
if (file_exists($filename)) {
   $art = simplexml_load_file($filename);

   // access a single element
   $painting = $art->painting[0];
   echo '<h2>' . $painting->title . '</h2>';
   echo '<p>By ' . $painting->artist->name . '</p>';
   // display id attribute
   echo '<p>id=' . $painting["id"] . '</p>';

   // loop through all the paintings
   echo "<ul>";
   foreach ($art->painting as $p)
   {
      echo '<li>' . $p->title . '</li>';
   }
   echo '</ul>';
} else {
   exit('Failed to open ' . $filename);
}

?>
```

Variable and attribute names taken from xml

LISTING 17.8 Using simple XML

XML Processing

With PHP using Simple XML and XPath

```
$art = simplexml_load_file($filename);

$titles = $art->xpath('/art/painting/title');
foreach ($titles as $t) {
   echo $t . '<br/>';
}

$names = $art->xpath('/art/painting[year>1800]/artist/name');
foreach ($names as $n) {
   echo $n . '<br/>';
}
```

LISTING 17.9 Using XPath with SimpleXML

XML Processing

With PHP using XMLReader

```
$filename = 'art.xml';
if (file_exists($filename)) {

   // create and open the reader
   $reader = new XMLReader();
   $reader->open($filename);

   // loop through the XML file
   while ( $reader->read() ) {
     $nodeName = $reader->name;

     // since all sorts of different XML nodes we must check
     // node type
     if ($reader->nodeType == XMLREADER::ELEMENT
         && $nodeName == 'painting') {
         $id = $reader->getAttribute('id');
         echo '<p>id=' . $id . '</p>';
     }

     if ($reader->nodeType == XMLREADER::ELEMENT
         && $nodeName =='title') {
         // read the next node to get at the text node
         $reader->read();
         echo '<p>' . $reader->value . '</p>';
     }
   }
} else {
   exit('Failed to open ' . $filename);
}
```

Less “automatic”

More Verbose

LISTING 17.10 Using XMLReader

XML Processing

Why choose when you can use both

```
// create and open the reader
$reader = new XMLReader();
$reader->open($filename);

// loop through the XML file
while($reader->read()) {
  $nodeName = $reader->name;
  if ($reader->nodeType == XMLREADER::ELEMENT
      && $nodeName =='painting') {
      // create a SimpleXML object from the current painting node
      $doc = new DOMDocument('1.0', 'UTF-8');
      $painting = simplexml_import_dom($doc->importNode
                  ($reader- >expand(),true));
      // now have a single painting as an object so can output it
      echo '<h2>' . $painting->title . '</h2>';
      echo '<p>By ' . $painting->artist->name . '</p>';
  }
}
```

LISTING 17.11 Combining XMLReader and SimpleXML

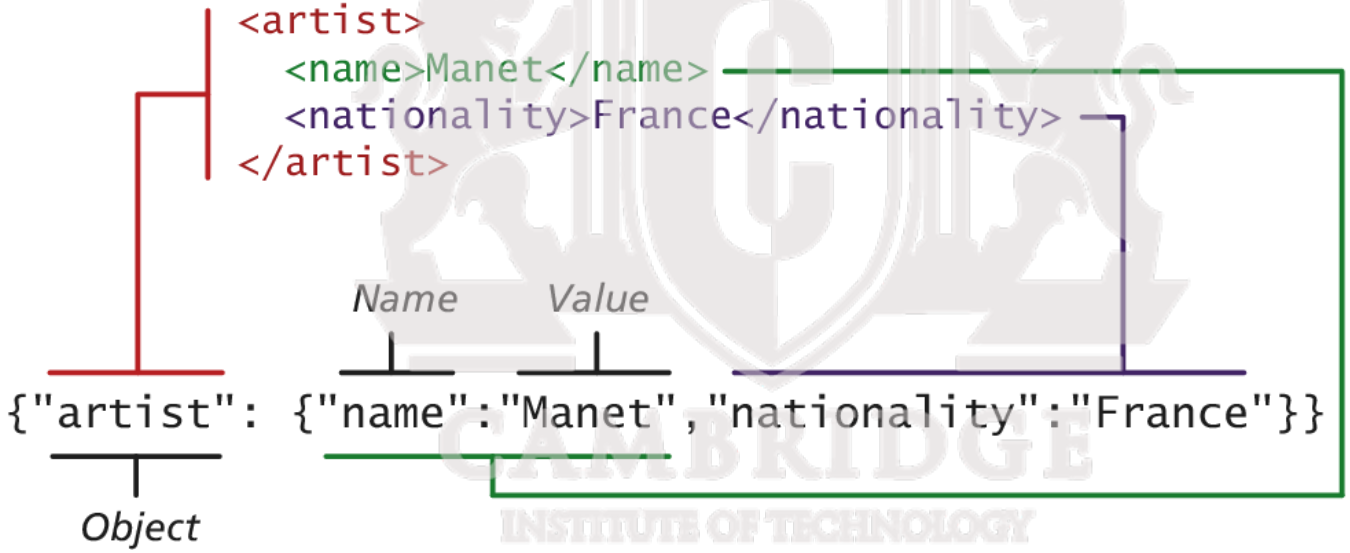
JSON

JSON

- **JSON** stands for JavaScript Object Notation (though its use is not limited to JavaScript)
- Like XML, JSON is a data serialization format. It provides a more concise format than XML.
- Many REST web services encode their returned data in the JSON data format instead of XML.

JSON

An example XML object in JSON



JSON

An example XML object in JSON

```
{
   "paintings": [
      {
         "id":290,
         "title":"Balcony",
         "artist":{
            "name":"Manet",
            "nationality":"France"
         },
         "year":1868,
         "medium":"Oil on canvas"
      },
      {
         "id":192,
         "title":"The Kiss",
         "artist":{
            "name":"Klimt",
            "nationality":"Austria"
         },
         "year":1907,
         "medium":"Oil and gold on canvas"
      },
      {
         "id":139,
         "title":"The Oath of the Horatii",
         "artist":{
            "name":"David",
            "nationality":"France"
         },
         "year":1784,
         "medium":"Oil on canvas"
      }
   ]
}
```

LISTING 17.12 JSON representation of XML data from Listing 17.1

Using JSON in JavaScript

Creating JSON JavaScript objects

it is easy to make use of the JSON format in JavaScript:

```
var a = {"artist": {"name":"Manet","nationality":"France"}};

alert(a.artist.name + " " + a.artist.nationality);
```

When the JSON information will be contained within a string (say when downloading) the JSON.parse() function can be used to transform the string containing into a JavaScript object

Using JSON in JavaScript

Convert string to JSON object and vice versa

```
var text = '{"artist": {"name":"Manet","nationality":"France"}}';

var a = JSON.parse(text);

alert(a.artist.nationality);
```

JavaScript also provides a mechanism to translate a JavaScript object into a JSON string:

```
var text = JSON.stringify(artist);
```

Using JSON in PHP

JSON on the server

Converting a JSON string into a PHP object is quite straightforward:

```
<?php
   // convert JSON string into PHP object
   $text = '{"artist": {"name":"Manet","nationality":"France"}}';
   $anObject = json_decode($text);
   echo $anObject->artist->nationality;

   // convert JSON string into PHP associative array
   $anArray = json_decode($text, true);
   echo $anArray['artist']['nationality'];
?>
```

Notice that the json_decode() function can return either a PHP object or an associative array.

Using JSON in PHP

Go the other way

To go the other direction (i.e., to convert a PHP object into a JSON string), you can use the json_encode() function.

```
// convert PHP object into a JSON string
$text = json_encode($anObject);
```

Section 4 of 7

OVERVIEW OF WEB SERVICES

Web Services

An overview

- Web services are the most common example of a computing paradigm commonly referred to as **service-oriented computing** (SOC).
- A **service** is a piece of software with a platform-independent interface that can be dynamically located and invoked.
- **Web services** are a relatively standardized mechanism by which one software application can connect to and communicate with another software application using web protocols.

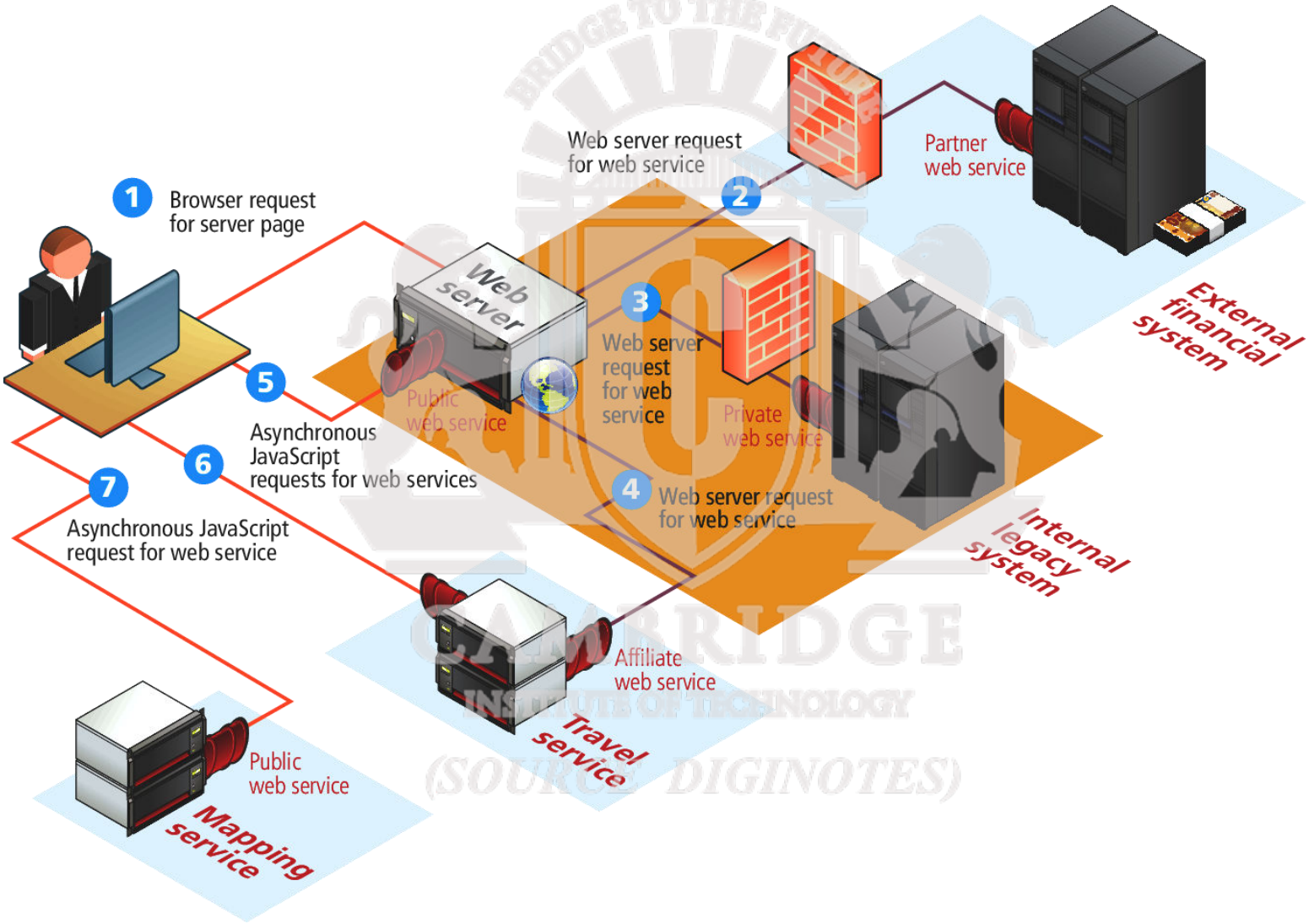
Web Services

Benefits

- they can provide interoperability between different software applications running on different platforms
- they can be used to implement a **service-oriented architecture** (SOA)
- they can be offered by different systems within an organization as well as by different organizations

Web Services

Visual Overview



Web Services

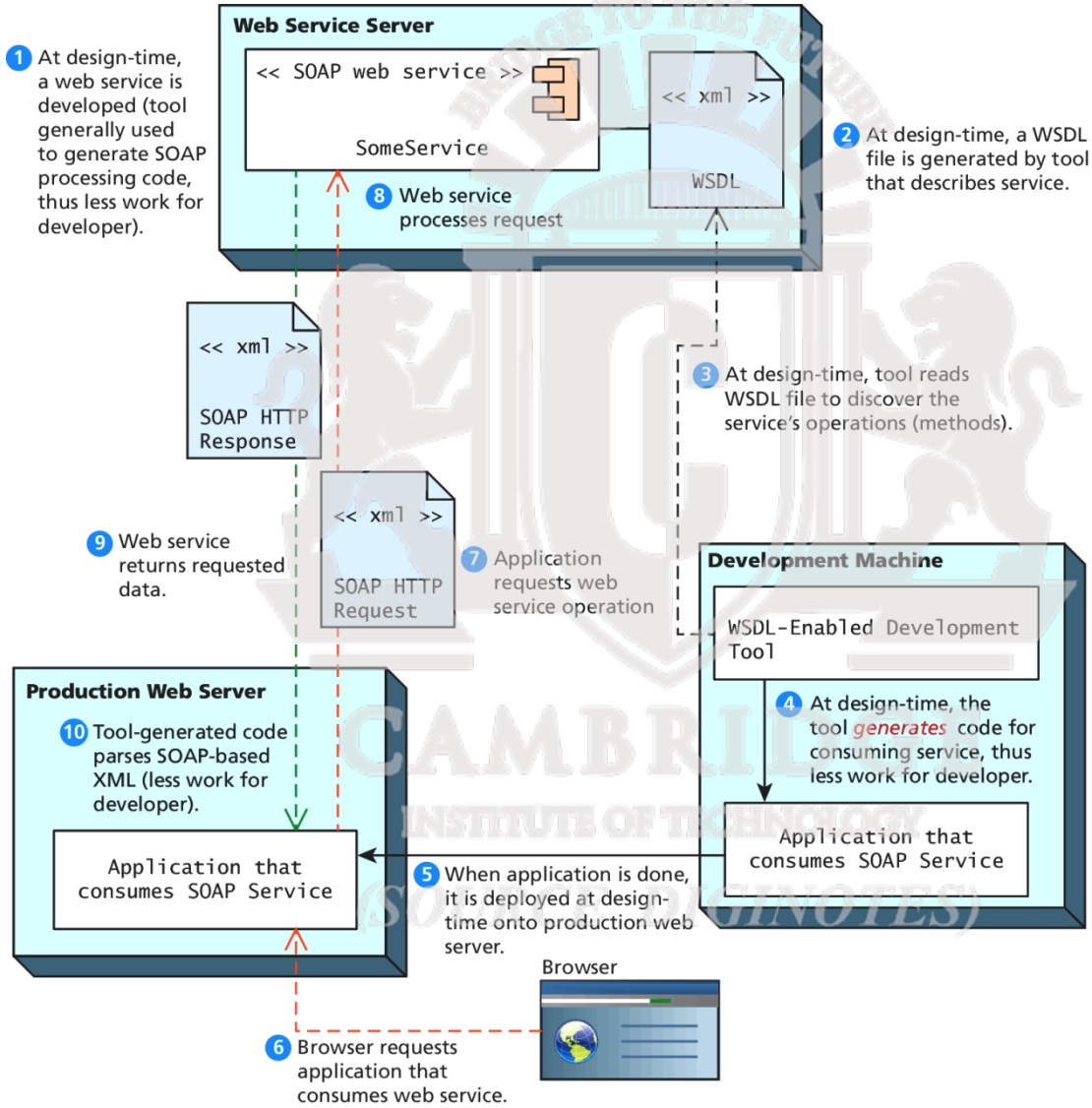
SOAP Services

SOAP is the message protocol used to encode the service invocations and their return values via XML within the HTTP header.

- SOAP and WSDL are complex XML schemas
- akin to using a compiler: its output may be complicated to understand
- the enthusiasm for SOAP-based web services had cooled.

Web Services

SOAP Services



Web Services

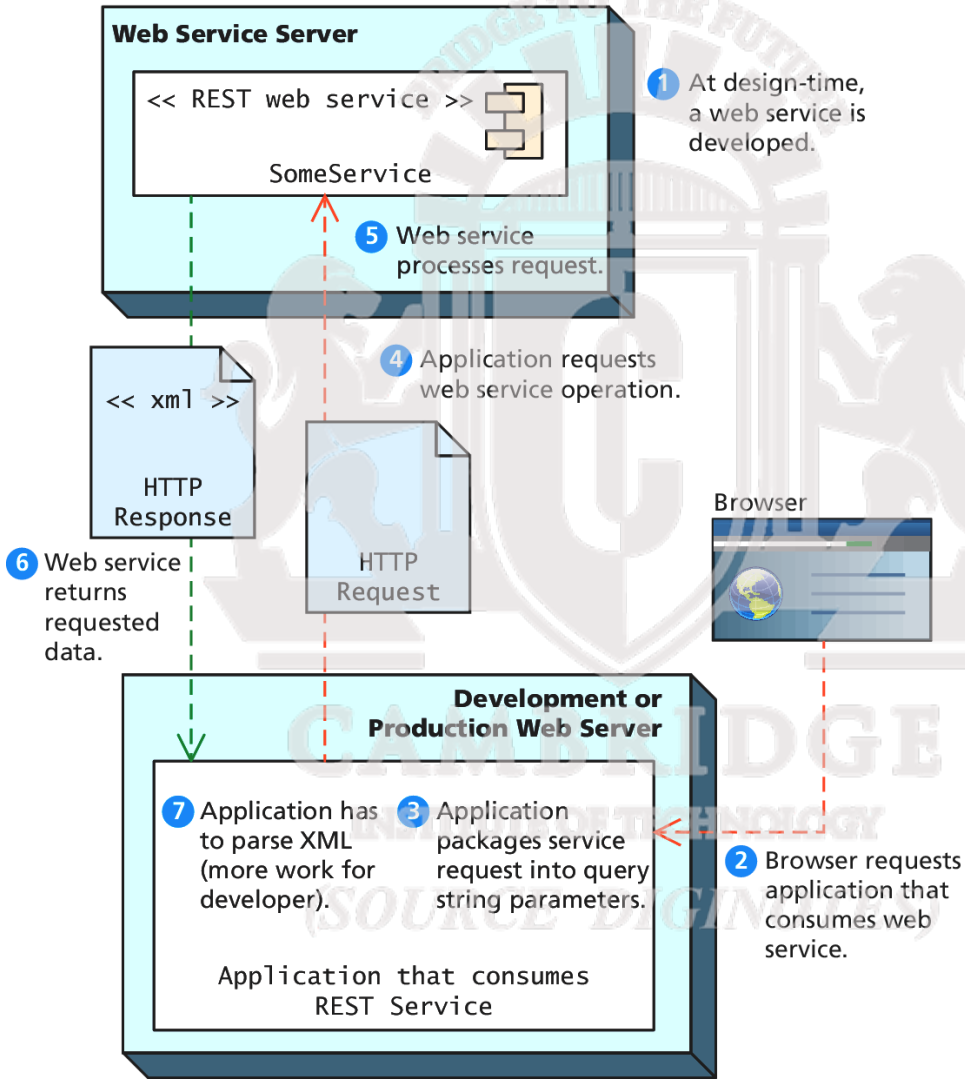
REST Services (an alternate to SOAP)

REST stands for Representational State Transfer.

- RESTful web service does away with the service description layer, and needs no separate protocol for encoding message requests and responses.
- It simply uses HTTP URLs for requesting a resource/object (and for encoding input parameters).
- The serialized representation of this object, usually an XML or JSON stream, is then returned to the requestor as a normal HTTP response.
- REST appears to have almost completely displaced SOAP services.

Web Services

REST Services



An Example Web Service

We will only use REST from here on in

•Consider the Google Geocoding API.

•The Google Geocoding API provides a way to perform geocoding operations via an HTTP GET request, and thus is an especially useful example of a RESTful web service.

•**Geocoding** typically refers to the process of turning a real-world address into geographic coordinates, which are usually latitude and longitude values

•**Reverse geocoding** is the process of converting geographic coordinates into a human-readable address.

An Example Web Service

More details

In this case the request will take the following form:

http://maps.googleapis.com/maps/api/geocode/xml?***address***

An example geocode request would look like the following:

http://maps.googleapis.com/maps/api/geocode/xml?address=British%20Museum,+Great+Russell+Street,+London,+WC1B+3DG&sensor=false

From trendsmap.com

An Example Web Service

The Response

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=UTF-8
Date: Fri, 19 Jul 2013 19:15:54 GMT
Expires: Sat, 20 Jul 2013 19:15:54 GMT
Cache-Control: public, max-age=86400
Vary: Accept-Language
Content-Encoding: gzip
Server: mafe
Content-Length: 512
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN

<?xml version="1.0" encoding="UTF-8"?>
<GeocodeResponse>
   <status>OK</status>
   <result>
      <type>route</type>
      <formatted_address>
         Great Russell Street, London Borough of Camden, London, UK
      </formatted_address>
      <address_component>
         <long_name>Great Russell Street</long_name>
         <short_name>Great Russell St</short_name>
         <type>route</type>
      </address_component>
      <address_component>
         <long_name>London</long_name>
         <short_name>London</short_name>
         <type>locality</type>
         <type>political</type>
      </address_component>
      ...
      <geometry>
         <location>
            <lat>51.5179231</lat>
            <lng>-0.1271022</lng>
         </location>
         <location_type>GEOMETRIC_CENTER</location_type>
         ...
      </geometry>
   </result>
</GeocodeResponse>
```

The response is a standard HTTP response with headers

This response is XML

The lat/lng is in there somewhere

LISTING 17.13 HTTP response from web service

Identifying and Authenticating Service Requests

Most web services are not open. Instead they typically employ one of the following techniques:

- **Identity**. Each web service request must identify who is making the request.
- **Authentication**. Each web service request must provide additional evidence that they are who they say they are.

Identity examples

Real World ways of limiting service

•Web services that make use of an API key typically require the user (i.e., the developer) to register online with the service for an API key. This API key is then added to the GET request as a query string parameter.

•For instance, to request to the Microsoft Bing Maps web service will look like the following :

•http://dev.virtualearth.net/REST/v1/Locations?o=xml&query=British%20Museum,+Great+Russell+Street,+London,+WC1B+3DG,+UK&**key**=**[*BING API KEY HERE*]**

Authentication

Real World ways of limiting service

- Some web services are providing private/proprietary information or are involving financial transactions.
- In this case, these services not only may require an API key, but they also require some type of user name and password in order to perform an authorization.
- Many of the most well-known web services instead make use of the OAuth standard.

END OF MODULE 5