# Managing State

Chapter 13

Randy Connolly and Ricardo Hoar

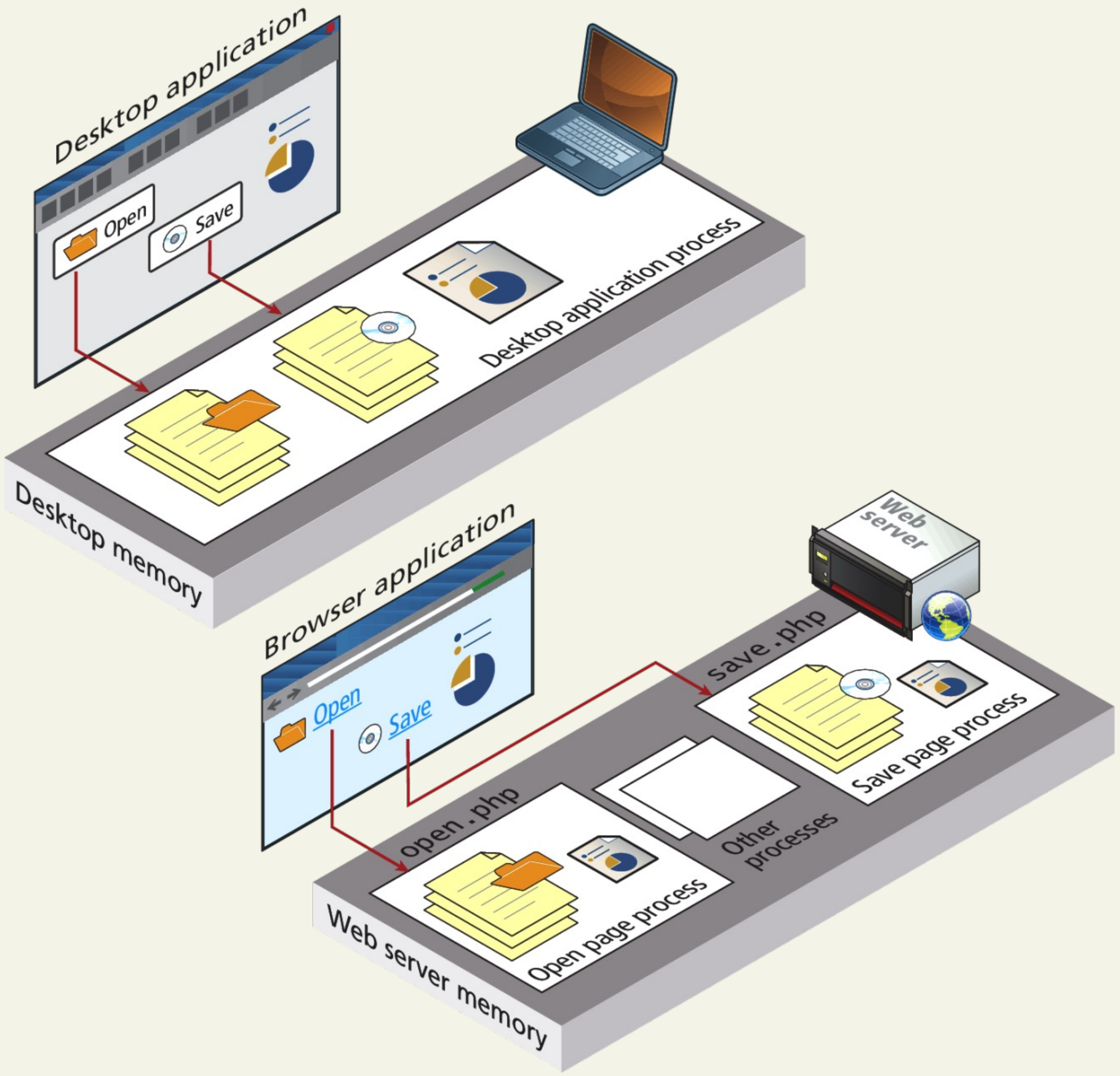
Fundamentals of Web Development

Section 1 of 8

## THE PROBLEM OF STATE IN WEB APPLICATIONS

# State in Web Applications

Not like a desktop application



## State in Web Applications

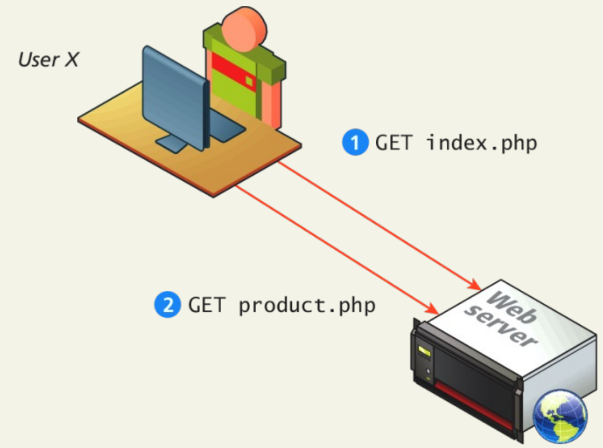
Not like a desktop application

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.

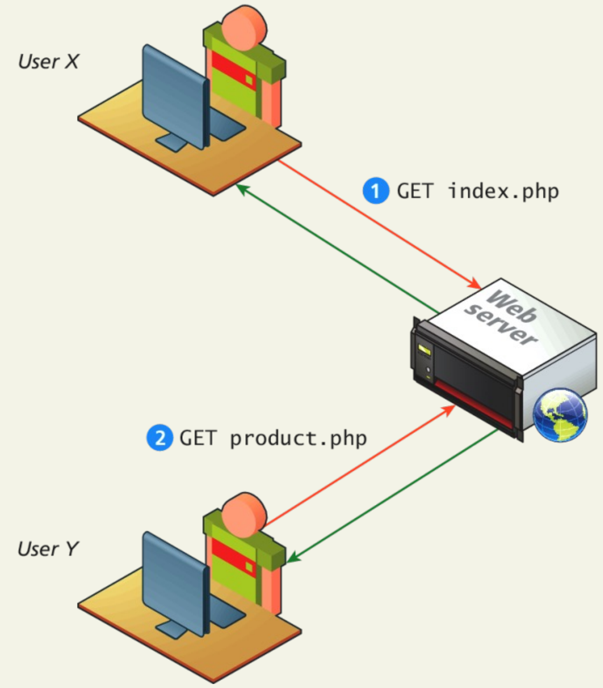
The HTTP protocol does not, without programming intervention, distinguish two requests by one source from two requests from two different sources

# State in Web Applications

What's the issue?

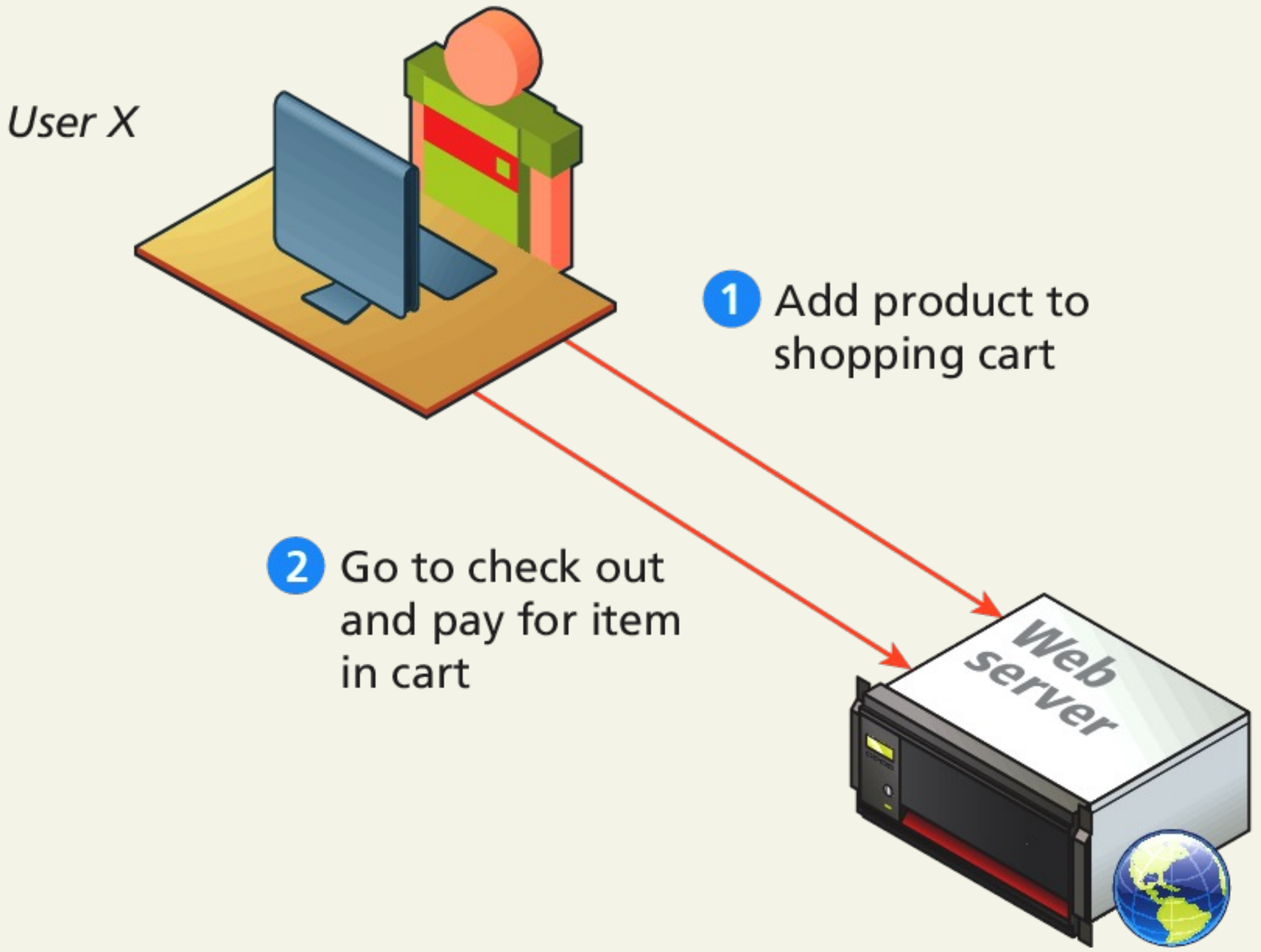


... is for the server not really any different than ...



# State in Web Applications

What's the desired outcome


User X
1 Add product to shopping cart
2 Go to check out and pay for item in cart
Web server

# State in Web Applications

How do we reach our desired outcome?

What mechanisms are available within HTTP to pass information to the server in our requests?

In HTTP, we can pass information using:

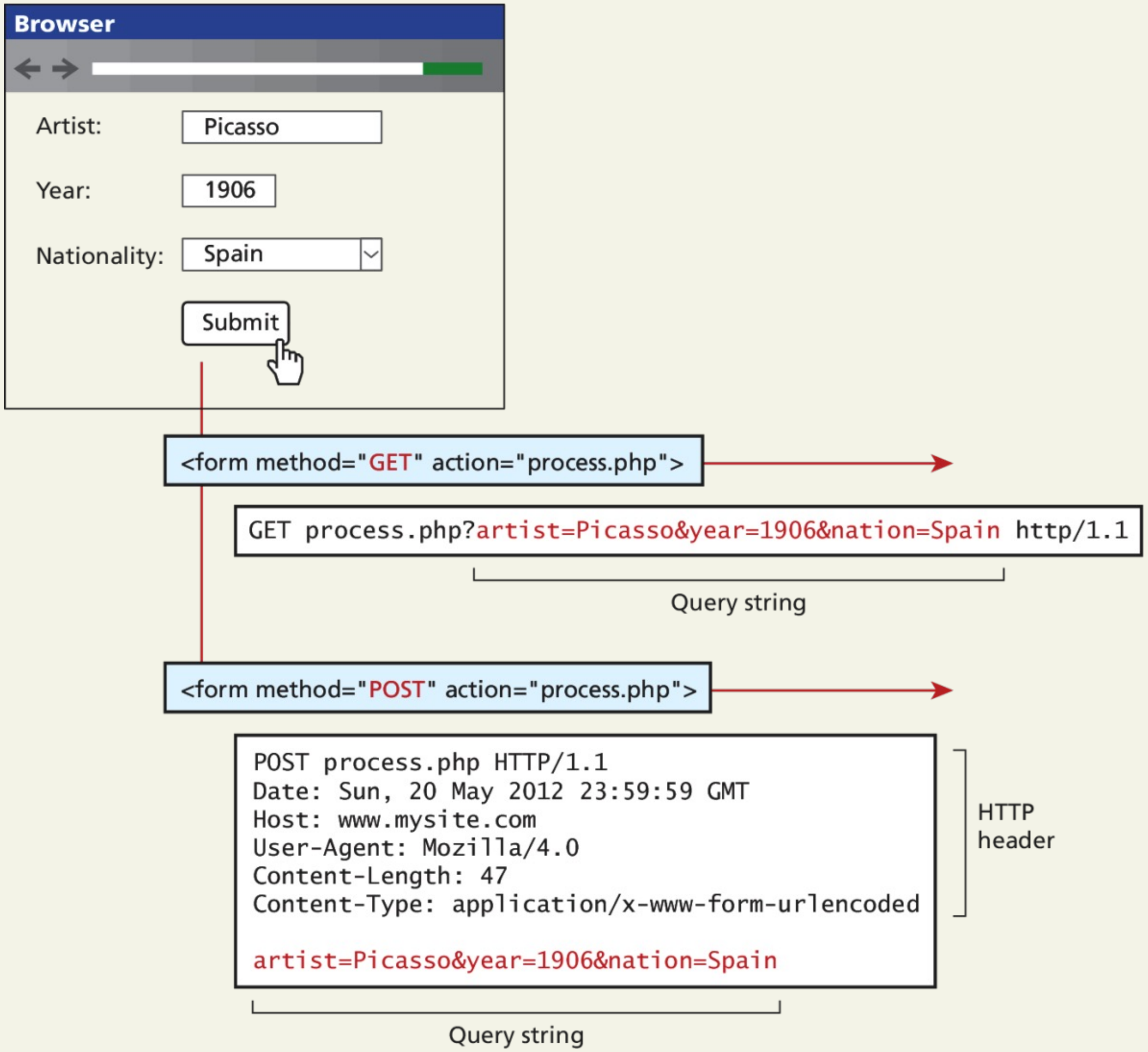
- Query strings
- Cookies

Section 2 of 8

## PASSING INFORMATION VIA QUERY STRINGS

# Info in Query Strings

Recall GET and POST



```
<form method="GET" action="process.php">
```

```
GET process.php?artist=Picasso&year=1906&nation=Spain http/1.1
```

```
<form method="POST" action="process.php">
```

```
POST process.php HTTP/1.1
Date: Sun, 20 May 2012 23:59:59 GMT
Host: www.mysite.com
User-Agent: Mozilla/4.0
Content-Length: 47
Content-Type: application/x-www-form-urlencoded

artist=Picasso&year=1906&nation=Spain
```

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

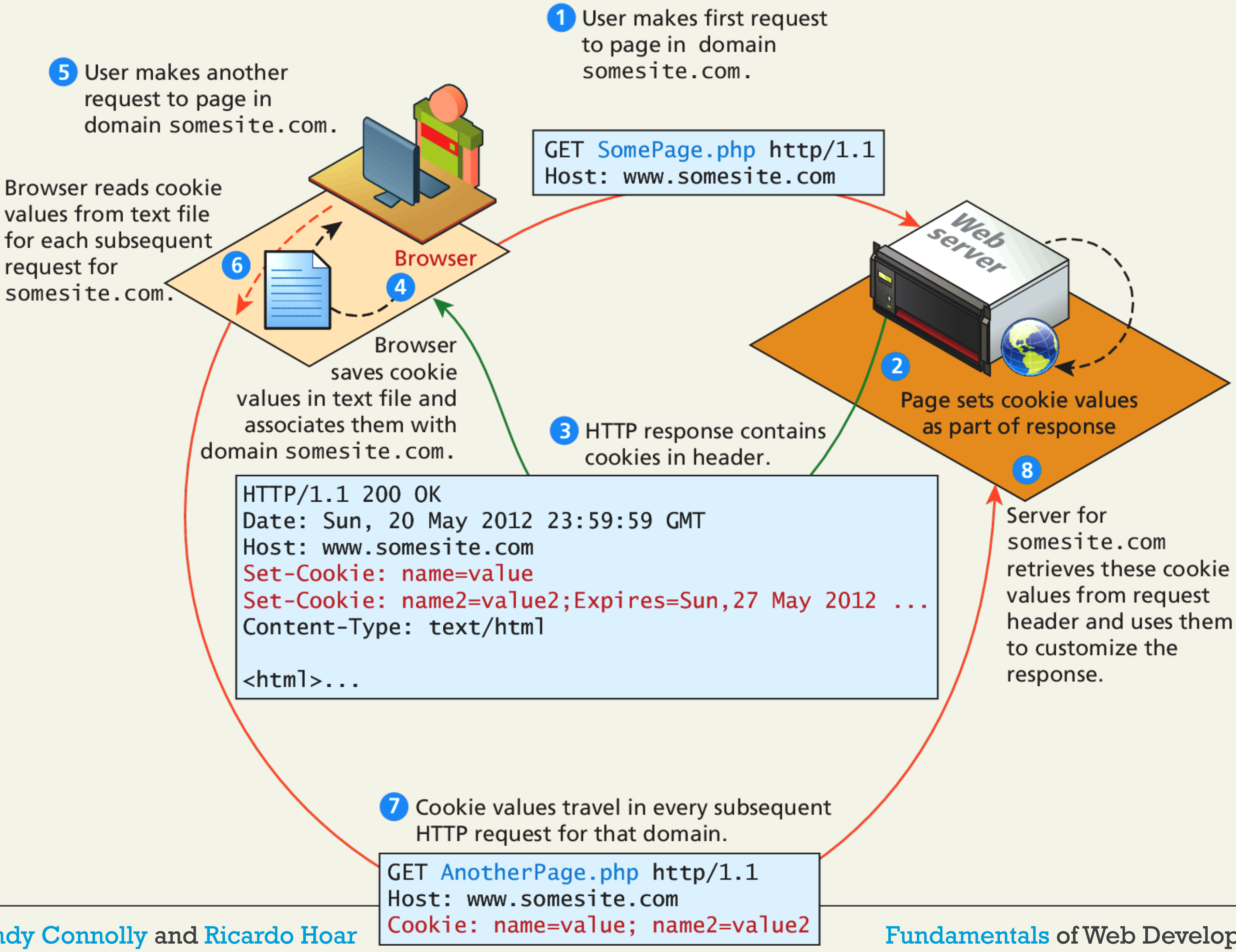
How do they Work?

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?



1 User makes first request to page in domain somesite.com.

```
GET SomePage.php http/1.1
Host: www.somesite.com
```

2 Page sets cookie values as part of response

3 HTTP response contains cookies in header.

```
HTTP/1.1 200 OK
Date: Sun, 20 May 2012 23:59:59 GMT
Host: www.somesite.com
Set-Cookie: name=value
Set-Cookie: name2=value2;Expires=Sun,27 May 2012 ...
Content-Type: text/html

<html>...
```

4 Browser saves cookie values in text file and associates them with domain somesite.com.

5 User makes another request to page in domain somesite.com.

6 Browser reads cookie values from text file for each subsequent request for somesite.com.

7 Cookie values travel in every subsequent HTTP request for that domain.

```
GET AnotherPage.php http/1.1
Host: www.somesite.com
Cookie: name=value; name2=value2
```

8 Server for somesite.com retrieves these cookie values from request header and uses them to customize the response.

## Cookie

- A cookie is a **small piece of data sent from a website and stored in a user's web browser** while the user is browsing that website.

  A web server specifies a cookie to be stored by sending an HTTP header called **Set-Cookie**.

  **Set-Cookie: value[; expires=date][; domain=domain][; path=path][; secure]**

  When a cookie is present, and the optional rules allow, the cookie value is sent to the server with each subsequent request. The cookie value is stored in an HTTP header called **Cookie** and contains just the cookie value without any of the other options

  **Cookie: value**

## Cookie

**bool setcookie ( string $name [, string $value [, int $expire = 0 [, string $path [, string $domain [, bool $secure = false [, bool $httponly = false ]]]]]] )**

- setcookie() defines a cookie to be sent along with the rest of the HTTP headers.
  – Like other headers, **cookies must be sent before any output from your script (this is a protocol restriction)**. This requires that you place calls to this function prior to any output, including <html> and <head> tags as well as any whitespace.

# Cookie

## Expire

- Indicates when the cookie should no longer be sent to the server and therefore may be deleted by the browser.
- Without the expires option, a cookie has a lifespan of a single session. A session is defined as finished when the browser is shut down, so session cookies exist only while the browser remains open.

## Cookie

**Domain**

- Indicates the domain(s) for which the cookie should be sent.
- By default, domain is set to **the host name of the page setting the cookie**, so the cookie value is sent whenever a request is made to the same host name.
- For example, the default domain for a cookie set might be **www.ing.unipi.it**
- The domain option is used to widen the number of domains for which the cookie value will be sent.
- Consider the case of a large network such as Yahoo! that has many sites in the form of *name.yahoo.com* (e.g., *my.yahoo.com*, *finance.yahoo.com*, etc.). A single cookie value can be set for all of these sites by setting the domain option to simply *yahoo.com*.

# Cookie

**Path**

- Indicates a URL path that must exist in the requested resource before sending the Cookie header
- This comparison is done by comparing the option value character-by-character against the start of the request URL. If the characters match, then the Cookie header is sent.

  Ex:

  **Set-Cookie: name=Nicholas; path=/blog**
- The path option would match /blog, /blogroll, etc.; anything that begins with /blog is valid. Note that this comparison is only done once the domain option has been verified.
- The default value is the current directory

## Cookie

**Secure**

- Just a flag and has no additional value specified. A secure cookie will only be sent to the server when a request is made using SSL and the HTTPS protocol.
- Confidential or sensitive information should never be stored or transmitted in cookies as the entire mechanism is inherently insecure.
- By default, cookies set over an HTTPS connection are automatically set to be secure.

## Cookie

```
<?php
$value = 'a nice cookie';
setcookie("mycookie", $value);
setcookie("mycookie", $value, time()+3600);  /* expire in 1 hour */
setcookie("mycookie", $value, time()+3600,
            "/~mario/", "unipi.it", 1);
/* expire in 1 hour, available within the /~mario/ directory and all sub-
  directories, available on all subdomains of unipi.it and set if a secure
  connection exists*/
?>
```

## 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
   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.

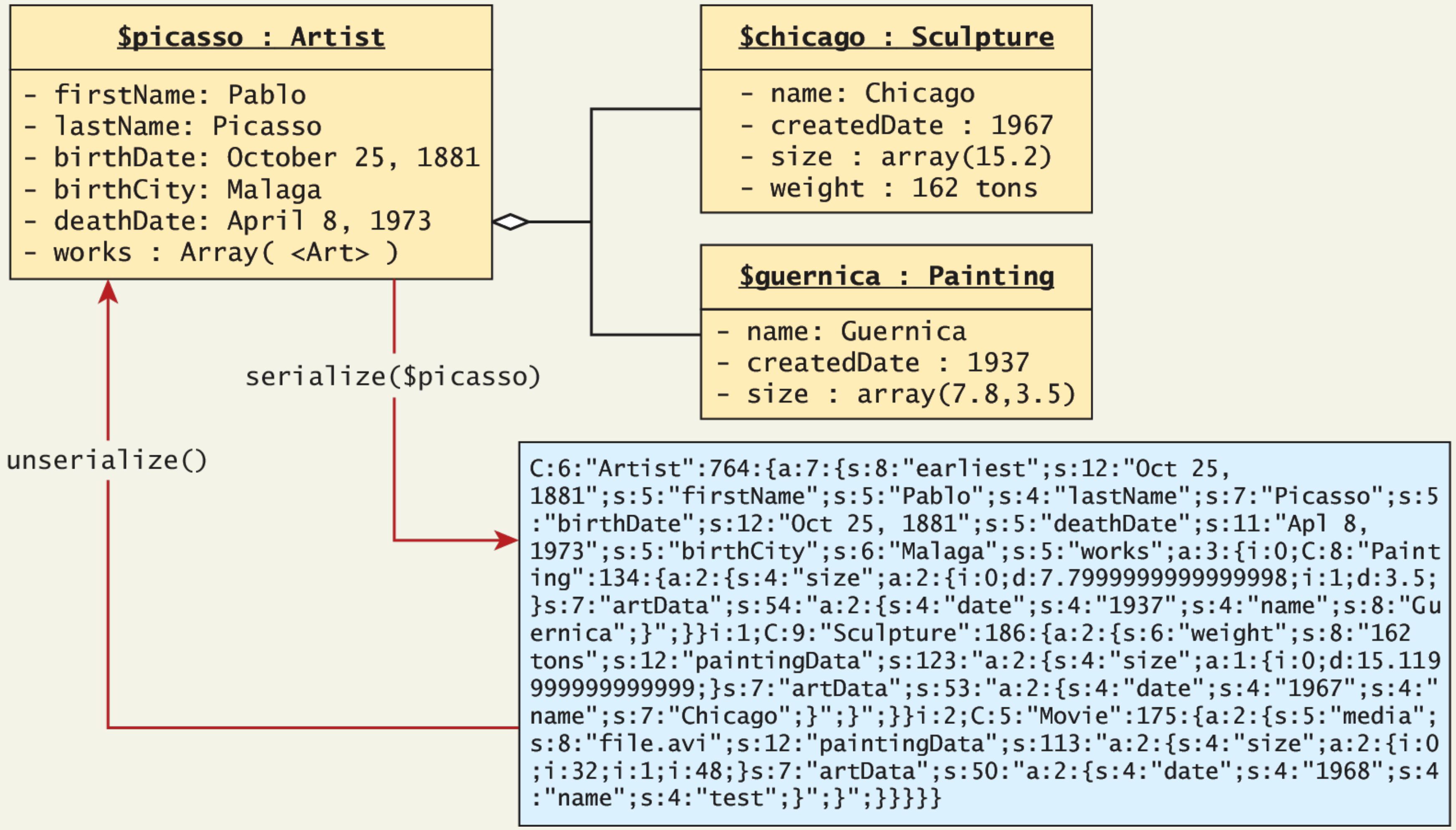
The string can be reconstituted back into an object using the **unserialize()** method

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

**LISTING 13.3** The Serializable interface

# Serialization

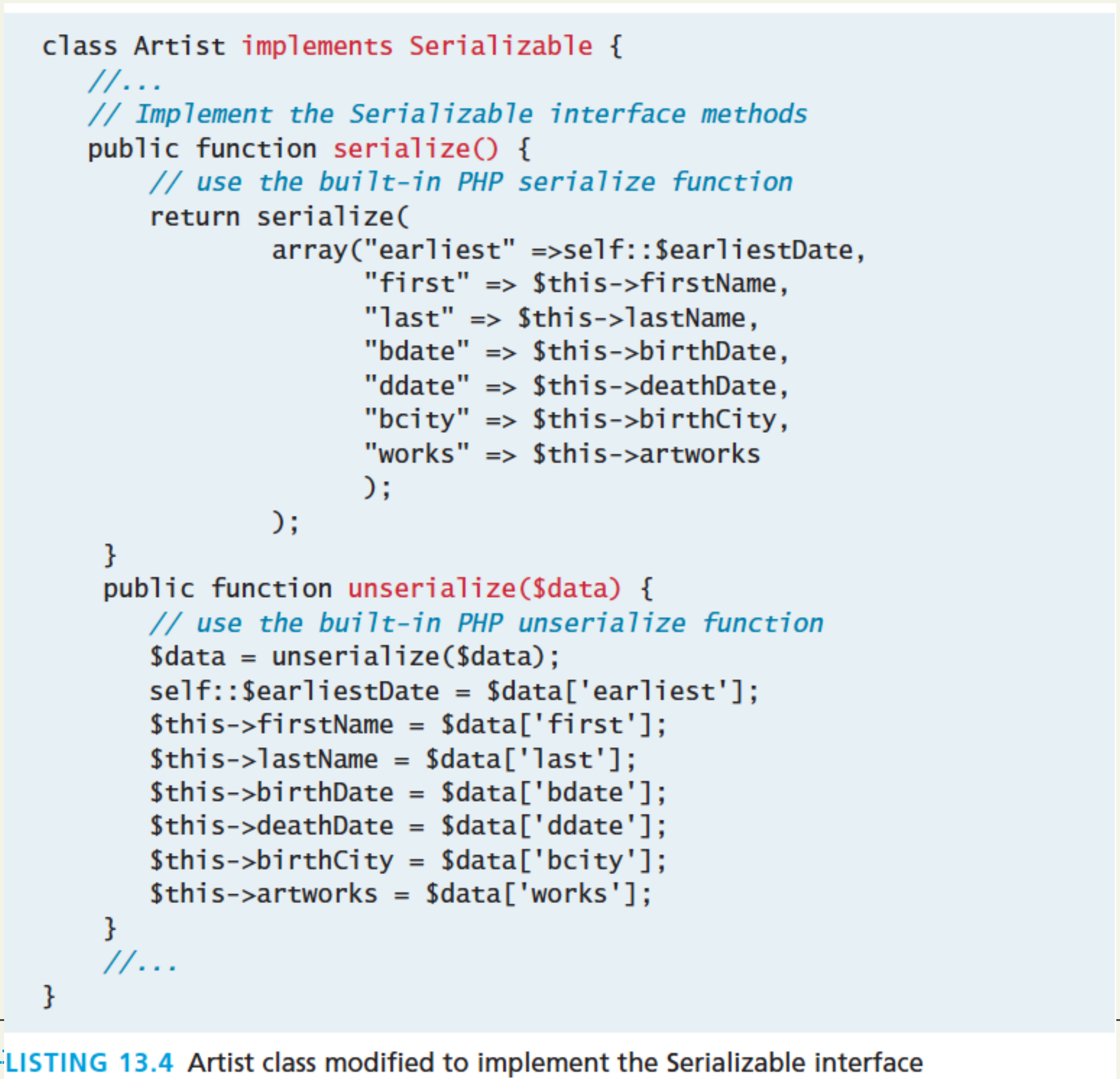
Serialization and deserialization



```
C:6:"Artist":764:{a:7:{s:8:"earliest";s:12:"Oct 25,
1881";s:5:"firstName";s:5:"Pablo";s:4:"lastName";s:7:"Picasso";s:5
:"birthDate";s:12:"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:"Paint
ing":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:"Gu
ernica";}";}}i:1;C:9:"Sculpture":186:{a:2:{s:6:"weight";s:8:"162
tons";s:12:"paintingData";s:123:"a:2:{s:4:"size";a:1:{i:0;d:15.119
999999999999;}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:12:"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";}";}";}}}}}
```

# Serialization

Consider our Artist class



```
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:"Oc
t 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:"Sculpt
ure”: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.

## PHP Classes
## Object Serialization

- **serialize()** returns a string containing a byte-stream representation of any value that can be stored in PHP.
- **unserialize()** can use this string to recreate the original variable values. Using serialize to save an object will save all variables in an object.

```
<?php
class A {
    public $one = 100;
    public function show_one() {  echo $this->one;  }
 }
?>
```

# PHP Classes Object Serialization

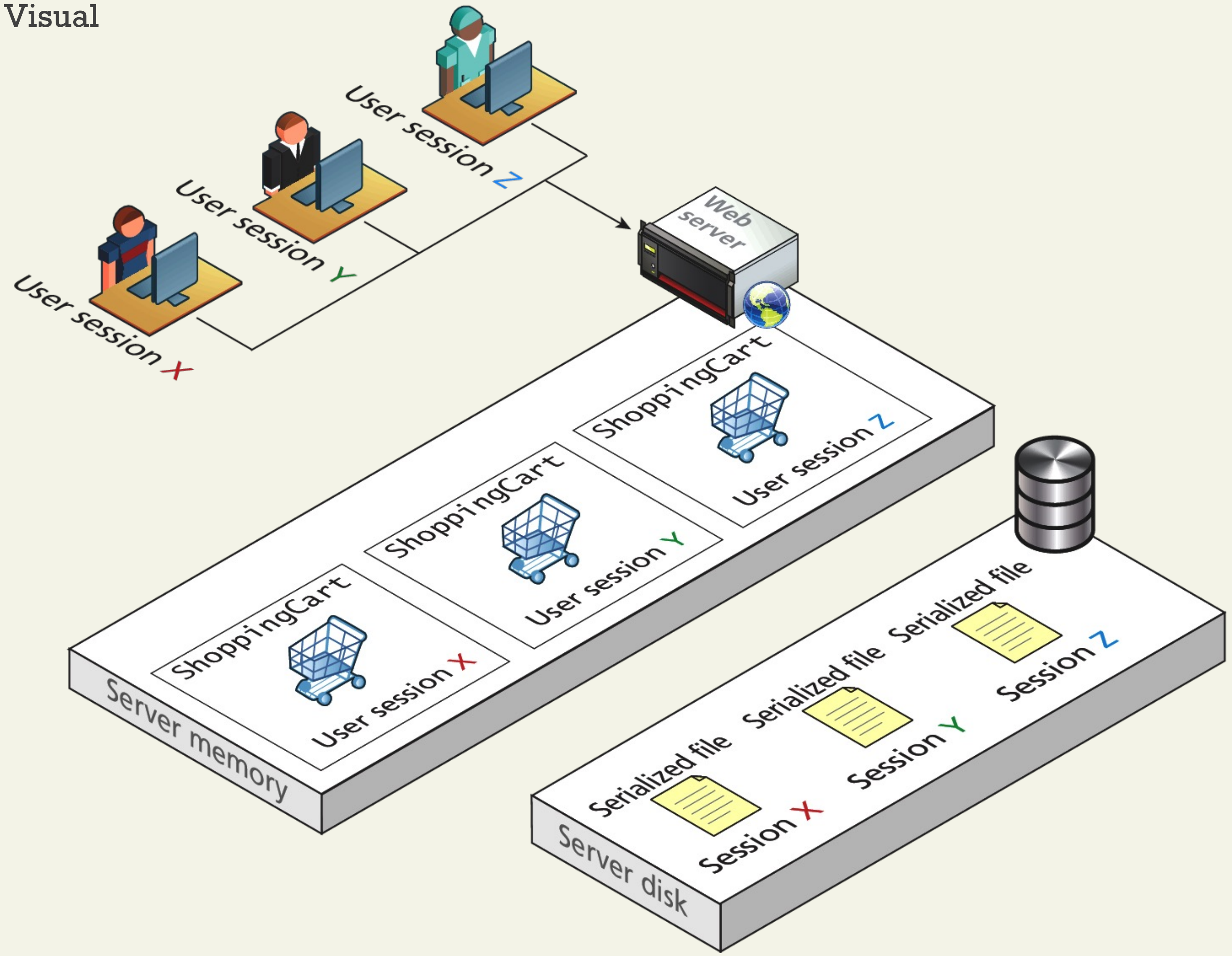
```
// page1.php:
include("classa.inc");
$a = new A;
$s = serialize($a);
// store $s somewhere where page2.php can find it.
file_put_contents('store', $s);

// page2.php:
// this is needed for the unserialize to work properly.
include("classa.inc");
$s = file_get_contents('store');
$a = unserialize($s);
// now use the function show_one() of the $a object.
$a->show_one();
```

Section 6 of 8

## SESSION STATE

# Session State

Visual



## Session State

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

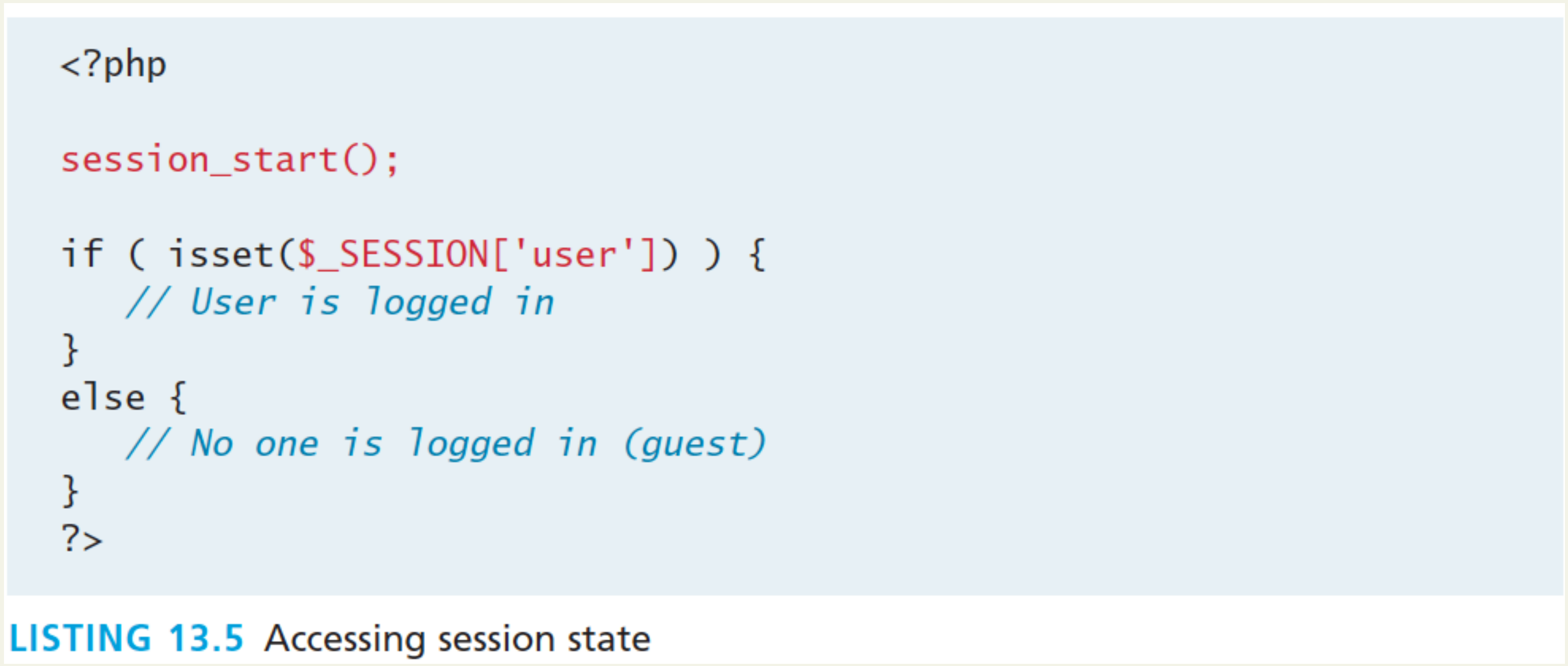
**Session state** is a server-based state mechanism that lets web applications store and retrieve objects of any type for each unique user session.

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

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

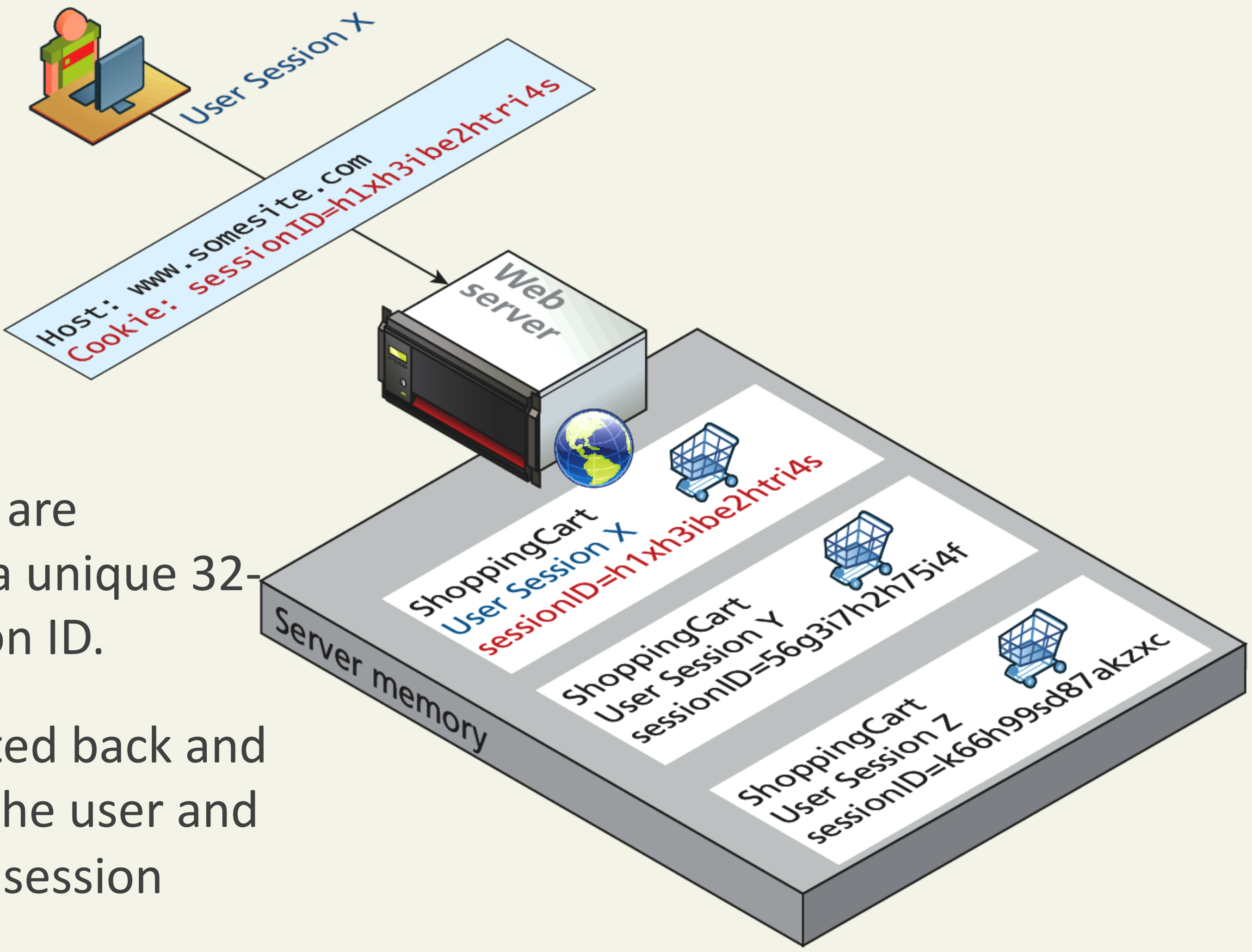
Checking Session existence

```
<?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-character 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.

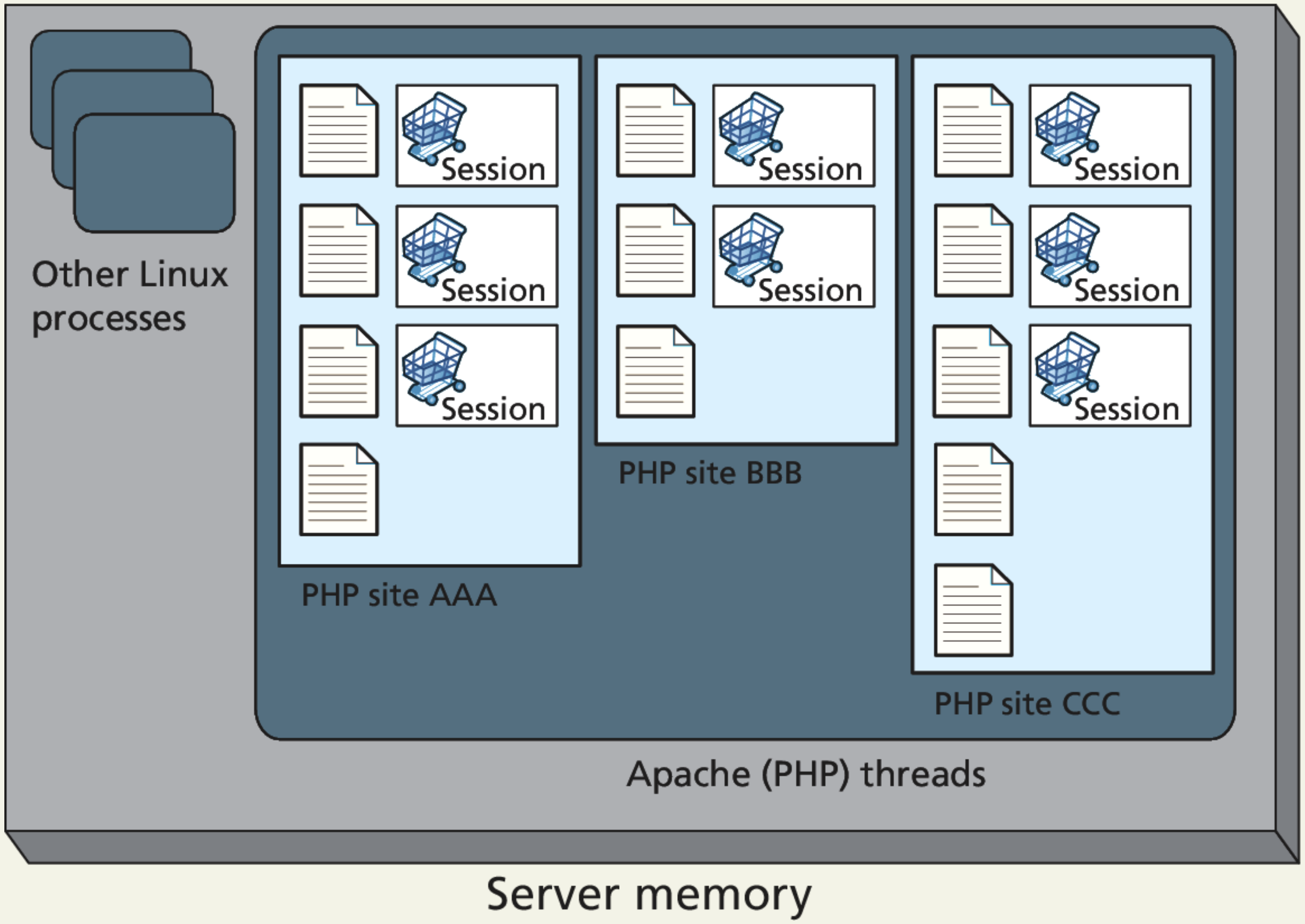
The decision to save sessions to files rather than in memory addresses the issue of memory usage that can occur on shared hosts as well as persistence between restarts.

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

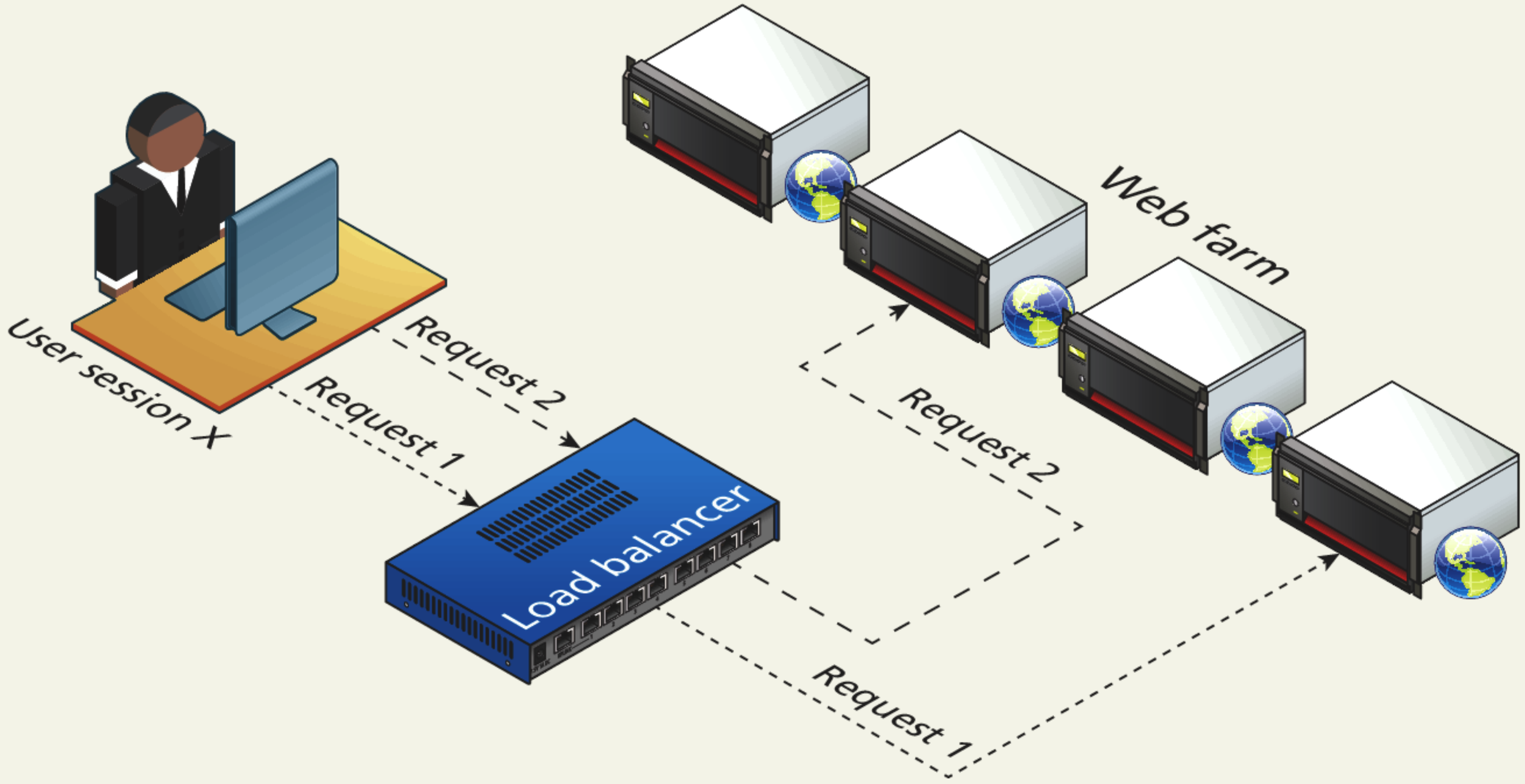
High Volume considerations

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

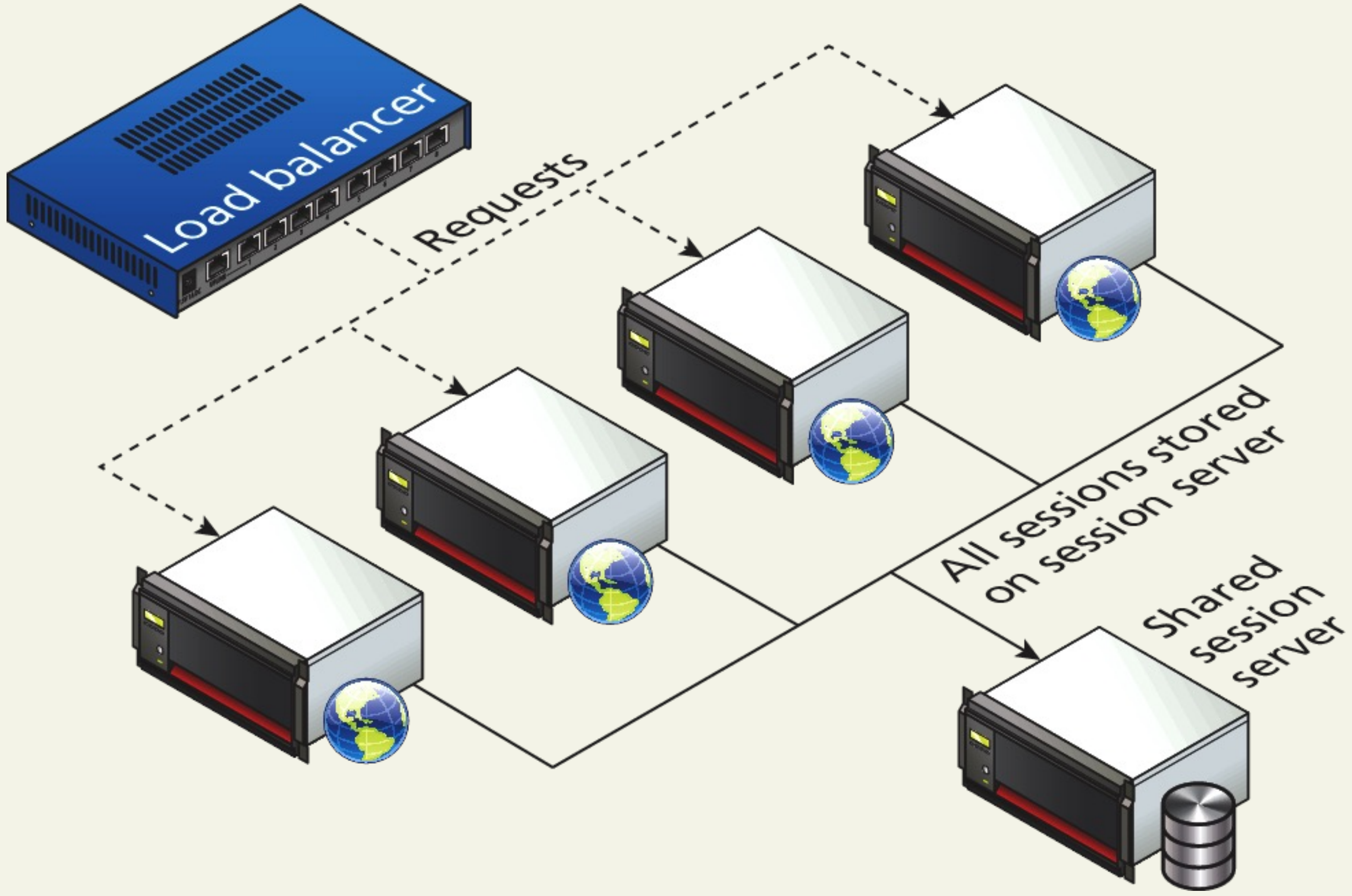
Visualizing the problem

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

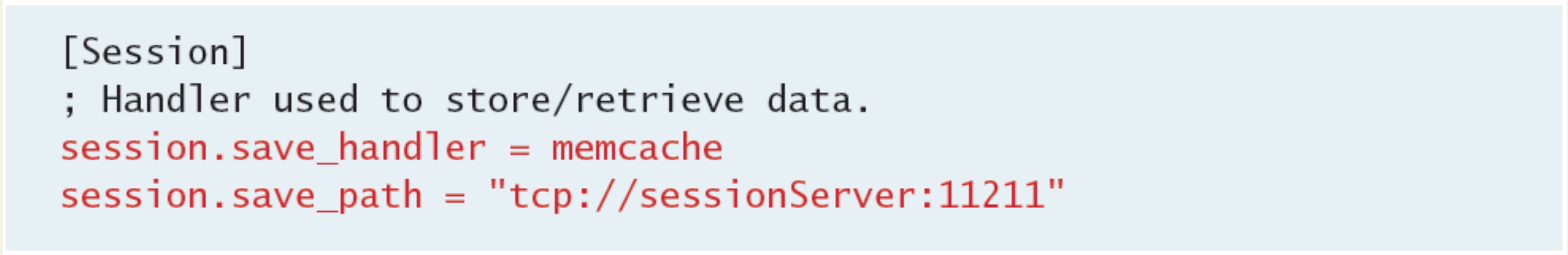
# Session Storage

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

Section 7 of 8

## HTML5 WEB STORAGE

# Web Storage

HTML5 only

**Web storage** is a new JavaScript-only API introduced in HTML5.4

It is meant to be a supplement 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.

In addition, web storage is not limited to the 4K size barrier of cookies;

## Web Storage

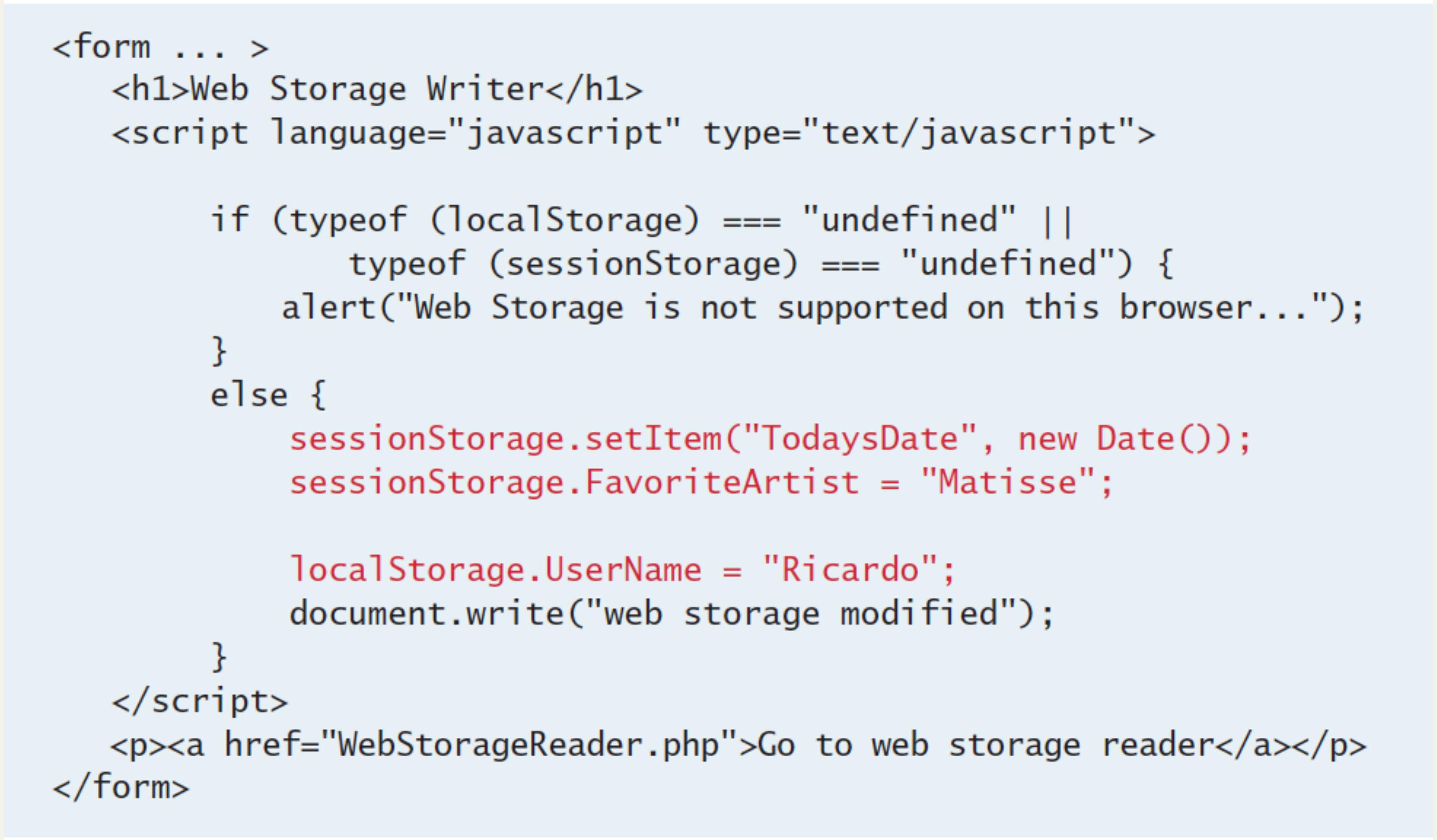
Two types

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

- The **localStorage** object is for saving information that will persist between browser sessions.
- The **sessionStorage** object is for information that will be lost once the browser session is finished.

# Using Web Storage

JavaScript code for writing information to web storage

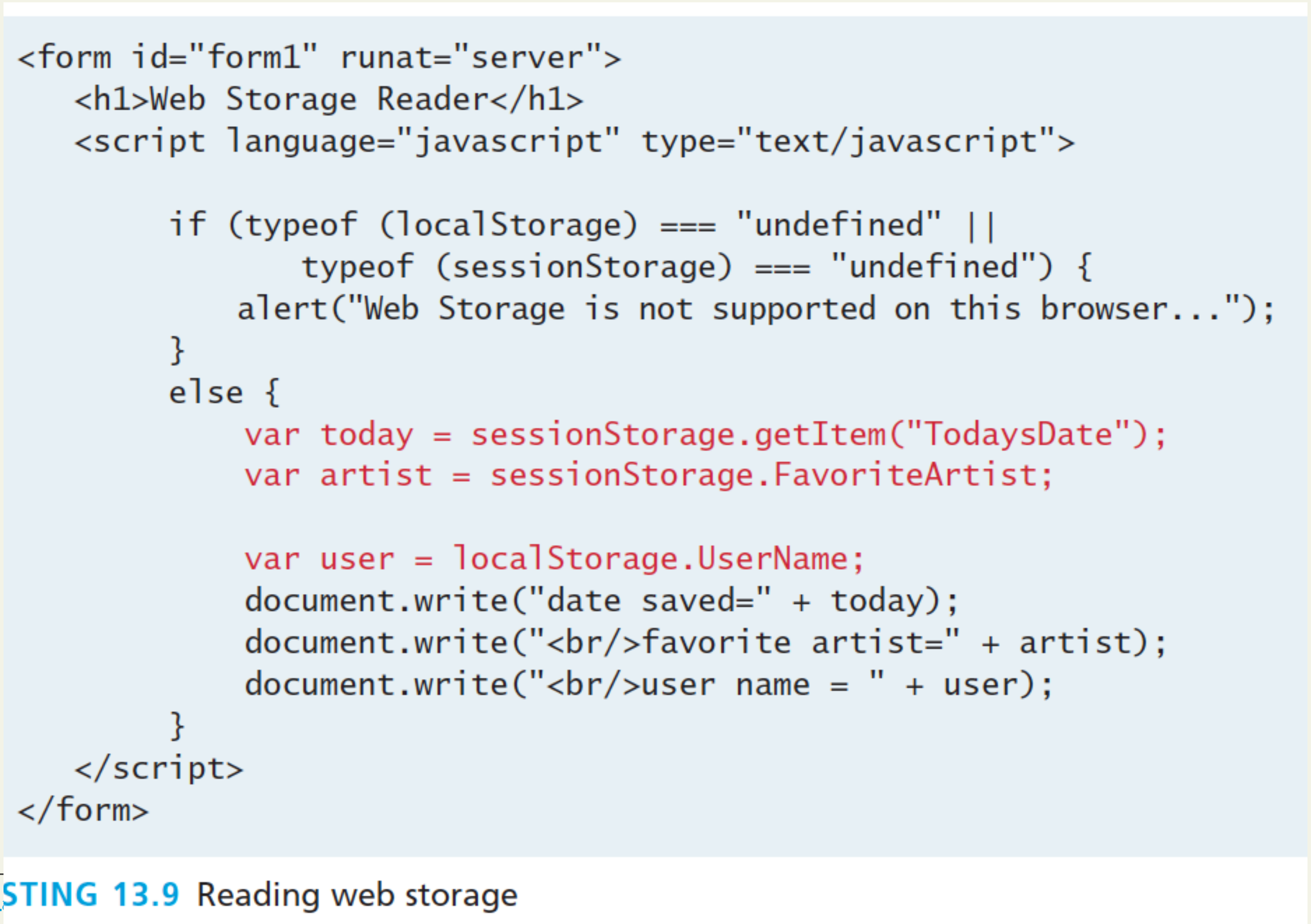


```
<form ... >
   <h1>Web Storage Writer</h1>
   <script language="javascript" type="text/javascript">

        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

## Using Web Storage

JavaScript code for reading information from web storage



```
<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>
```

RSTING 13.9 Reading web storage

## Using Web Storage

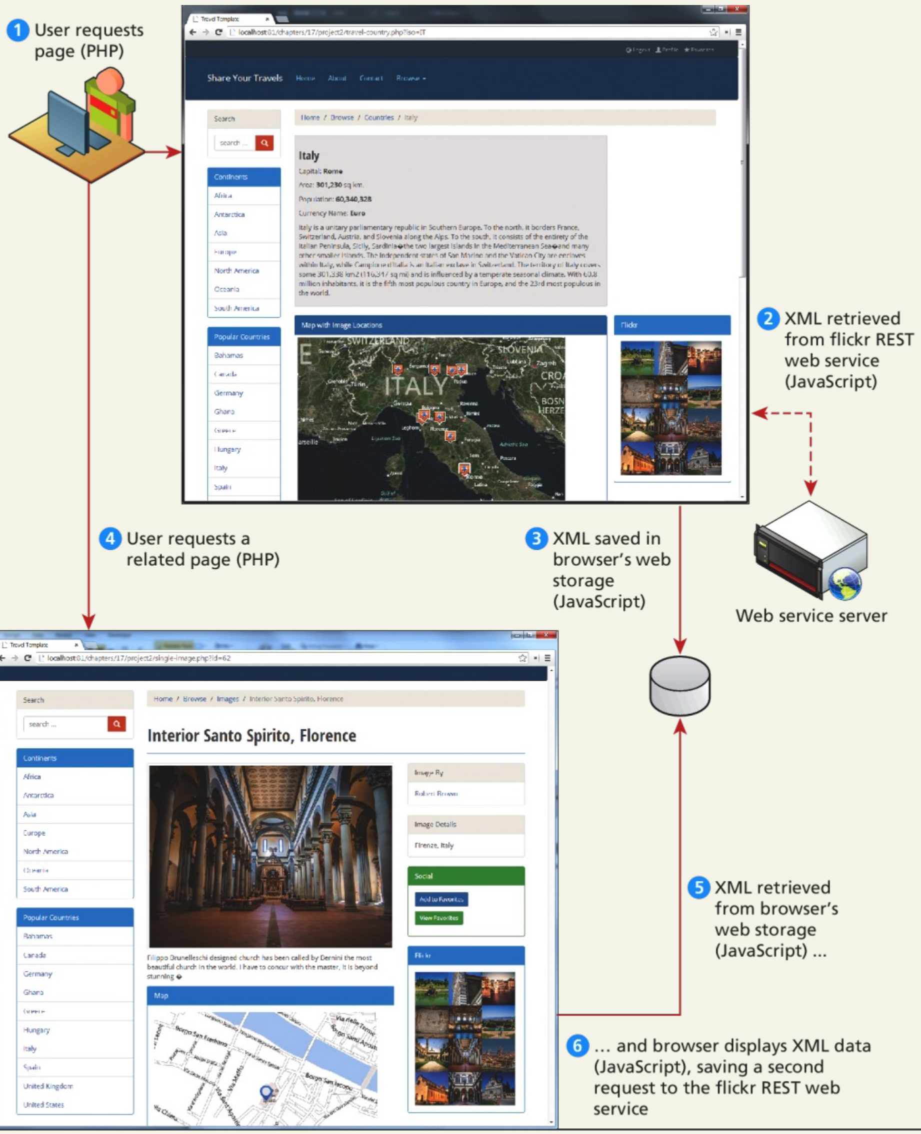
Why would you do it?

A better way to think about web storage is not as a cookie replacement but as a local cache for relatively static items available to JavaScript

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

# Using Web Storage

How would you do it?


1 User requests page (PHP)
2 XML retrieved from flickr REST web service (JavaScript)
3 XML saved in browser's web storage (JavaScript)
4 User requests a related page (PHP)
5 XML retrieved from browser's web storage (JavaScript) ...
6 ... and browser displays XML data (JavaScript), saving a second request to the flickr REST web service
Web service server
Italy
Interior Santo Spirito, Florence