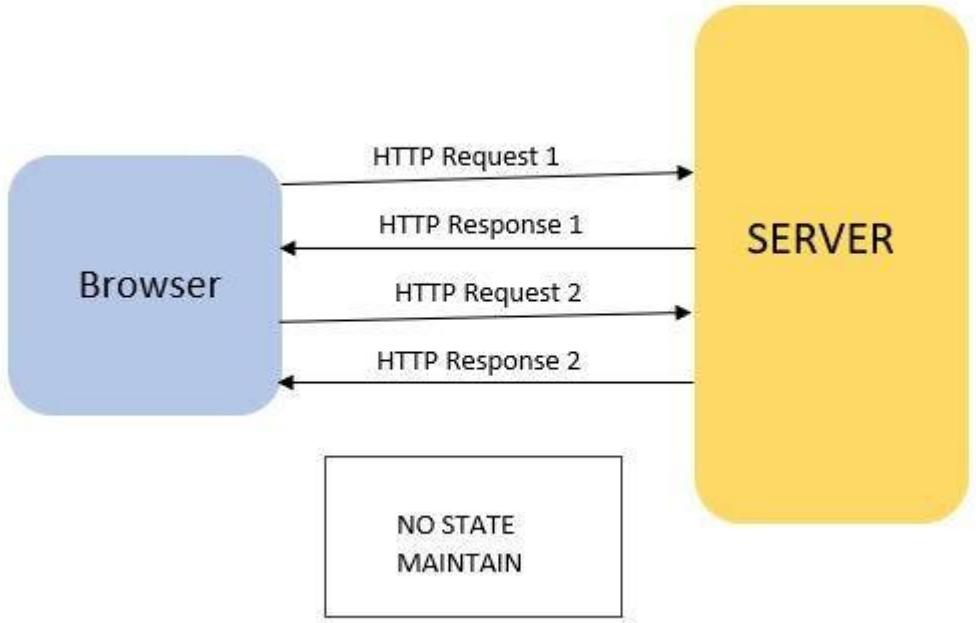
Cookie & Session

Md. Arman Hossain
Lecturer, CSE, EWU

HTTP is a Stateless Protocol

- Each HTTP request made by a client to the server is treated as an independent and isolated interaction.
- **Client Request 1**: GET/product?id=1
- Server responds with information about product #1 and forgets the request.
- **Client Request 2**: GET/product?id=2
- Server responds with information about product #2 without knowing the user had previously requested product #1.

No built-in mechanism to remember the previous state



Problem with stateless

We only authenticate ourselves in login page. Then how does it remember us for many days?

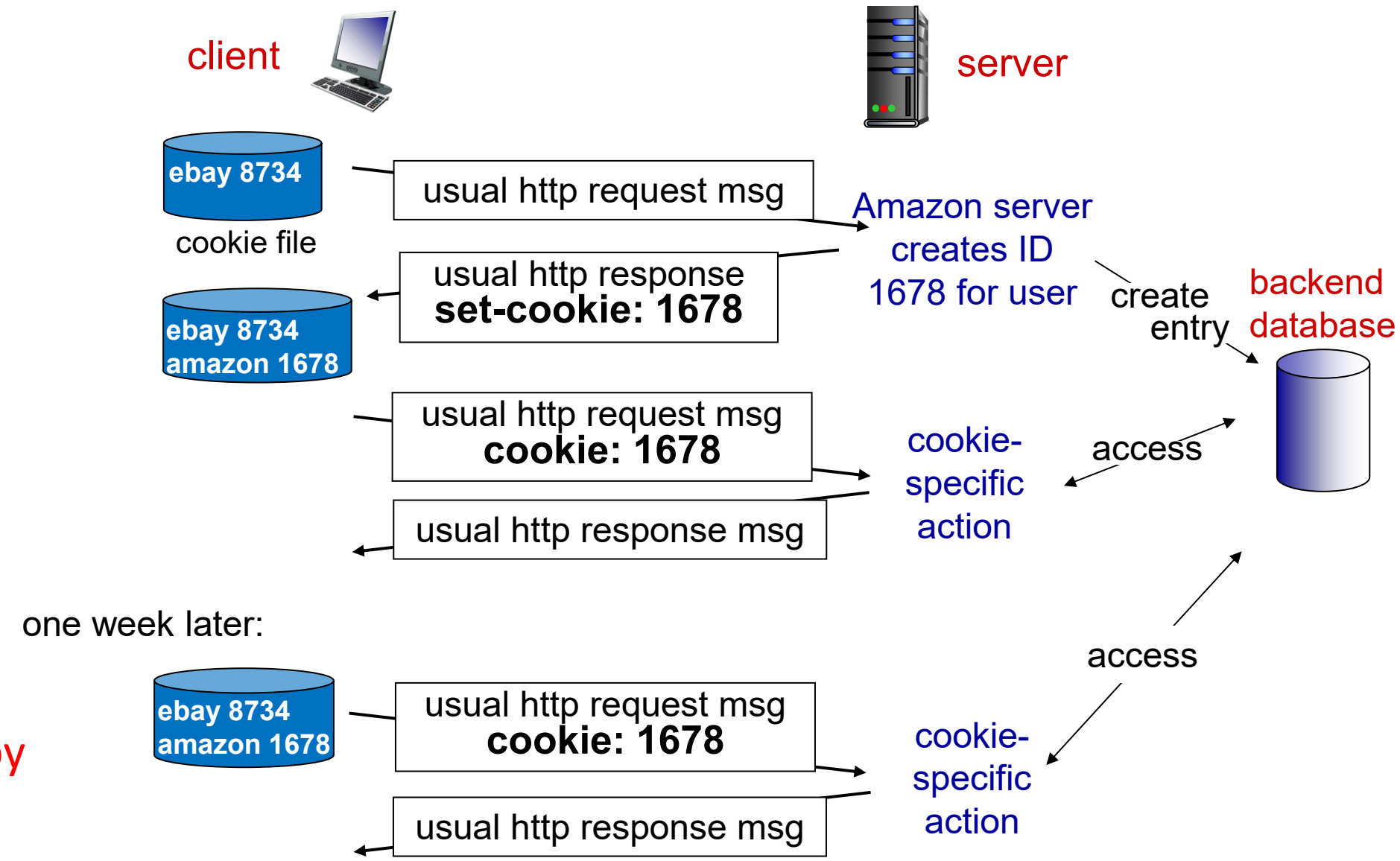
- Despite HTTP's stateless nature, developers use cookie, and session concept to maintain the state.

- Cookie can be thought of as tickets/token used to identify clients and their previous actions.

Cookie

Cookie is a packet of data:

- Set by the server to client,
- Stored in the client (browser) as key value pairs and
- Then sent back to the server automatically each time it is accessed by the client.



That is how cookie introduces state into HTTP.

Cookies (continued)

what cookies can be used for:

- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

how to keep "state":

- protocol endpoints: maintain state at sender/receiver over multiple transactions
- cookies: http messages carry state

aside

cookies and privacy:

- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites

Set and Read

```php
<?php
   setcookie("MyCookie",      $value, time()+7200);
   setcookie("AnotherCookie", $value, time()+7);
?>

<?php
   foreach ($_COOKIE as $key=>$val) {
     print $key . " => " . $val . "<br/>";
   }
?>
```

- Cookie can be set as multiple key value pairs.
- The code's output only become visible on the next page load why?

setcookie(name,value,expire,path,domain,secure)

Parameter	Description
name	(Required). Specifies the name of the cookie
value	(Required). Specifies the value of the cookie
expire	(Optional). Specifies when the cookie expires. e.g. **time()+3600*24*30** will set the cookie to expire in **30 days**. If this parameter is not set, the cookie will expire at the end of the session (when the browser closes).
path	(Optional). Specifies the server path of the cookie. If set to "**/**", the cookie will be available within the entire domain. If set to "**/phptest/**", the cookie will only be available within the test directory and all sub-directories of **phptest**. The default value is the current directory that the cookie is being set in.
domain	(Optional). Specifies the domain name of the cookie. To make the cookie available on all subdomains of example.com then you'd set it to ".example.com". Setting it to www.example.com will make the cookie only available in the www subdomain
secure	(Optional). Specifies whether or not the cookie should only be transmitted over a secure **HTTPS** connection. **TRUE** indicates that the cookie will only be set if a secure connection exists. Default is **FALSE**.

Where to put the code

Cookies have to be sent before the heading elements.

```
<?php
$strValue = "This is my first cookie";
setcookie ("mycookie", $strValue);
echo "Cookie set<br>";
?>
```

```
<!DOCTYPE html PUBLIC "=//W3C//DTD XHMTL 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
 <html xmlns="http://www.w3.org/1999/xhmtl" xml:lang="en">
<head><title>PHP Script using Cookies</title>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
</head>
<body>
   <?php
      echo “<p> A cookie has been set. </p>”;
   ?>
</body>
</html>
```

Set, check and delete

```php
<?php // Setting a cookie
setcookie("username", "JohnDoe", time() + 86400 * 30, "/"); // Cookie valid for 30 days

// Accessing a cookie
if (isset($_COOKIE["username"])) {
    echo "Welcome, " . $_COOKIE["username"];
}

// Deleting a cookie
setcookie("username", "", time() - 3600, "/");
?>
```

Using only name, i.e. setcookie(“username”), can also delete a cookie. Figure out how it works?

Multiple data & Use of explode

```php
<?php
$strAddress = $_SERVER['REMOTE_ADDR'];
$strBrowser = $_SERVER['HTTP_USER_AGENT'];
$strOperatingSystem = $_ENV['OS'];
$strInfo = "$strAddress::$strBrowser::$strOperatingSystem";
setcookie ("somecookie4",$strInfo, time()+7200);
?>
<?php
$strReadCookie = $_COOKIE["somecookie4"];
$arrListOfStrings = explode ("::", $strReadCookie);
echo "<p>$strInfo</p>";
echo "<p>Your IP address is: $arrListOfStrings[0] </p>";
echo "<p>Client Browser is: $arrListOfStrings[1] </p>";
echo "<p>Your OS is: $arrListOfStrings[2] </p>";
?>
```

Cookie lifetime

1. Session cookies: Deleted when the browser is closed. When no time is associated

2. Persistent cookies: Stored for a specified duration defined by the expires or max-age attribute.

- **Use Cases of Cookies**:
  1. Remembering user preferences (e.g., theme settings, language selection).
  2. Tracking user activity (e.g., analytics or advertising purposes).
  3. Storing session IDs for authentication.

Session

Cookie's problem

Cookie solves statelessness issue in HTTP but has some problems as well

1. **Security**: Since cookies are stored on user's computer it is possible for an attacker to easily steal, modify a cookie content to insert potentially harmful data in your application that might break your application.
2. **Bandwidth**: all the cookie data for a website is must be sent to the server within the request.
3. **Storage**: Cookies have a size limit, typically 4 KB per cookie, depending on the browser. This is often insufficient for storing large or complex data structures, like shopping cart details or user preferences.

Session

You can solve these issues by using the session concept.

- A PHP session stores data on the server rather than user's computer.
- Every user is identified through a unique number called session identifier or SID.
- The session IDs are randomly generated by the PHP engine which is almost impossible to guess.
- Because the session data is stored on the server, it doesn't have to be sent with every browser request.

* In this case, rather than sending the whole data to the server as cookie, the client sends only the unique session ID to the server. The server then store or retrieve data by the session id.

Starting a PHP Session

Before you can store any information in session variables, You must call the session_start() function at the beginning of the page i.e. **before any output generated by your script in the browser**, much like you do while setting the cookies with setcookie() function.

```php
<?php
// Starting session
session_start();
?>
```

Session_start()

- It first checks to see if a session already exists by looking for the presence of a session ID in **cookie** by PHPSESSID key.
- If it finds one, i.e. if the session is already started, that session is resumed.
- if doesn't, it starts a new session by creating a new session ID. Which is a random string of 32 hexadecimal numbers such as
  3c7foj34c3jj973hjkop2fc937e3443.
- A file is automatically created on the server in the designated temporary directory (/tmp) and bears the name of the unique identifier prefixed by sess_ i.e. sess_3c7foj34c3jj973hjkop2fc937e3443.
- It saves (and load) session related variable inside that file.

The session file format and location are determined by the ***session.save_path*** setting in ***php.ini***

Session

```php
<?php // Starting a session
session_start();

// Storing a session variable
$_SESSION["username"] = "JohnDoe";

// Accessing a session variable
if (isset($_SESSION["username"])) {
    echo "Welcome, " . $_SESSION["username"];
}

// Unsetting and destroying a session
session_unset(); // Remove all session variables
session_destroy(); // Destroy the session
?>
```

Remove certain session data

```php
<?php
// Starting session
session_start();

// Removing session data
if(isset($_SESSION["lastname"])){
    unset($_SESSION["lastname"]);
}
?>
```

Session Termination

Termination can be triggered either:

On Server-Side: When developers can destroy the session and clear stored data manually

```php
<?php session_start();
session_unset(); // Removes all session variables
session_destroy(); // Destroys the session
?>
```

On Client-Side: The session ends naturally if the session cookie expires or the user clears their cookies.

Session Termination by Cookie (cont.)

- By default a session persist until user closes the browser (session cookie with no timestamp).
- Every PHP session has a timeout value — a duration, measured in seconds — which determines how long a session should remain alive in the absence of any user activity.
- You can adjust this timeout duration by changing the value of ***session.gc_maxlifetime*** variable in the PHP configuration file (***php.ini***)

Session ID Communication

The Session ID needs to be sent back and forth between the client (browser) and the server for the session to remain identifiable.
This is typically handled in two ways:

1. Cookies (default):The Session ID is stored in a browser cookie, commonly named PHPSESSID in PHP. Example of a session Cookie: PHPSESSID=abcd1234xyz5678URL
2. Parameters: If cookies are disabled, the Session ID can be appended to URLs as a query parameter (e.g., http://example.com/page.php?PHPSESSID=abcd1234xyz5678).

How It Works Together

login.php (User Login and Start Session)

```php
<?php
// Start the session
session_start();

// Simulate login check (e.g., with username and password)
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $username = $_POST["username"];
    $password = $_POST["password"];

    // Example check: Username is "admin" and password is "1234"
    if ($username === "admin" && $password === "1234") {
        // Set session variables
        $_SESSION["username"] = $username;

        // Create a cookie to store the session ID (e.g., expires in 1 hour)
        setcookie("PHPSESSID", session_id(), time() + 3600, "/");

        echo "Login successful! <a href='dashboard.php'>Go to dashboard</a>";
    } else {
        echo "Invalid username or password.";
    }
}
?>
```

login.php (Cont)

```
<!DOCTYPE html>
<html>
<body>
    <h2>Login Form</h2>
    <form method="POST">
        <label for="username">Username:</label>
        <input type="text" name="username" id="username" required>
        <br>
        <label for="password">Password:</label>
        <input type="password" name="password" id="password" required>
        <br>
        <button type="submit">Login</button>
    </form>
</body>
</html>
```

dashboard.php (Securing a Page Using Both Sessions and Cookies)

```php
<?php
// Start the session
session_start();

// Check if the session ID matches the cookie
if (
    isset($_SESSION["username"]) &&
    isset($_COOKIE["PHPSESSID"]) &&
    $_COOKIE["PHPSESSID"] === session_id()
) {
    echo "Welcome, " . $_SESSION["username"] . "! You are logged in.";
    echo "<br><a href='logout.php'>Logout</a>";
} else {
    echo "You are not logged in. Please <a href='login.php'>login</a>.";
}
?>
```

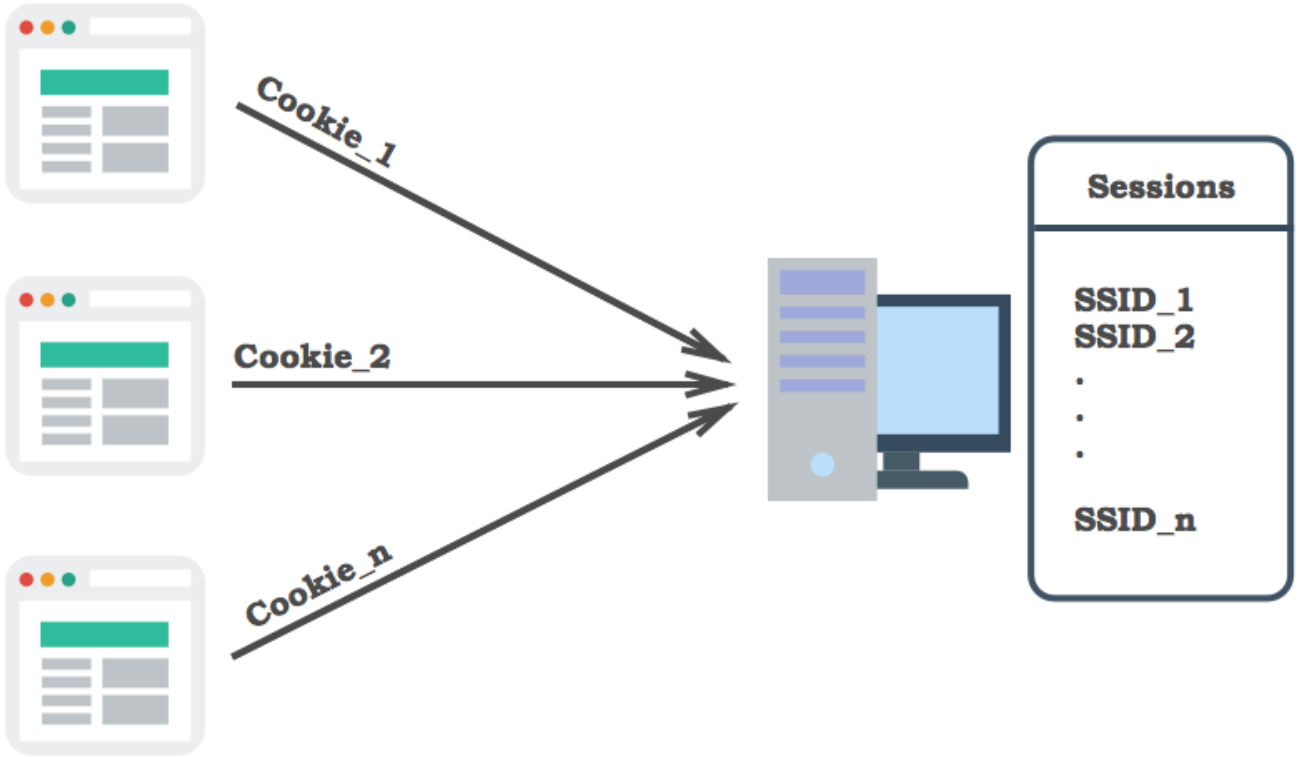
logout.php (Clear Session and Cookie)

```php
<?php
// Start the session
session_start();

// Destroy the session
session_unset();
session_destroy();

// Clear the cookie
setcookie("PHPSESSID", "", time() - 3600, "/");

echo "You have been logged out. <a href='login.php'>Login again</a>";
?>
```


Cookie_1
Cookie_2
Cookie_n
Sessions
SSID_1
SSID_2
SSID_n

Thank you