

COMP 2021

Unix and Script Programming



PHP
Cookies and Sessions

HTTP - a 'Stateless' Environment

stateless

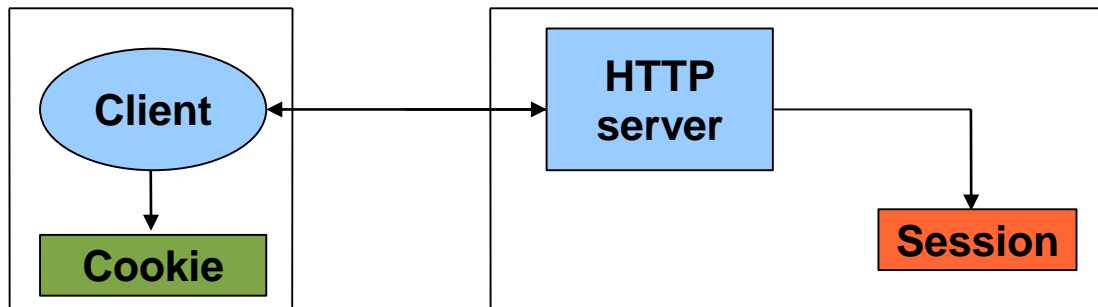
(adj.) Having no information about what occurred previously.

- ▶ HTTP is **stateless** – it does not keep track of the client between requests
 - ▶ When you browse the web, you are not always connected to the server
 - ▶ Once the request has been processed and returned from the server, the connection is closed
 - ▶ Connection needs to be re-opened when you need new information or refresh



Cookies and Sessions

- ▶ But sometimes we need to keep track of information, or have **persistent data**
 - ▶ Shopping cart
 - ▶ “Remember me” on login sites
- ▶ PHP sessions and cookies are mechanisms for introducing state into HTTP transactions.
 - ▶ **Cookies** – small file stored client-side
 - ▶ **Sessions** – relevant data stored on the server



Is PHP Stateless?

- ▶ Variables are destroyed as soon as the page script finishes executing.
- ▶ The script can access the 'referrer', the address of the previous page, although this can't really be trusted.

```
$_SERVER [ 'HTTP_REFERER' ]
```

- ▶ It is possible to add data to a database/text file to add persistent data, although *this is not connected with a particular user...*





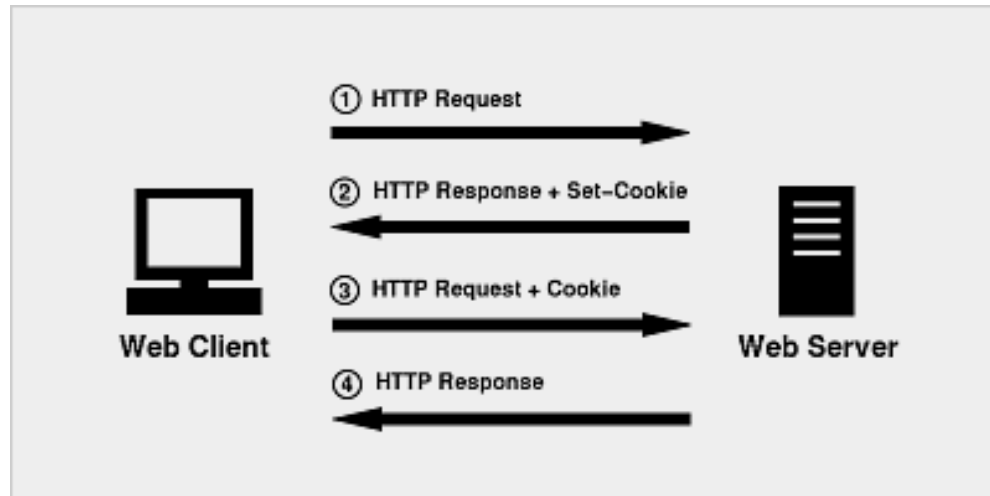
Cookies

What is a Cookie?

- ▶ **HTTP cookies** are data which a server-side script sends to a web client to keep for a period of time.
 - ▶ a small text file that is stored on a user's computer
- ▶ On every subsequent HTTP request, the web client automatically sends the cookies back to server (unless the cookie support is turned off).
- ▶ The cookies are embedded in the HTTP header (and therefore not visible to the users).



How do HTTP Cookies work?



Cookies are **transferred** between server and client according to **HTTP**

1. User sends a HTTP request for page for the first time.
 2. Server sends back the HTTP response (e.g. HTML webpage) to the browser AND stores some data in a cookie on the user's PC.
 3. At the next page request, all cookie data associated with this domain is sent too.
-

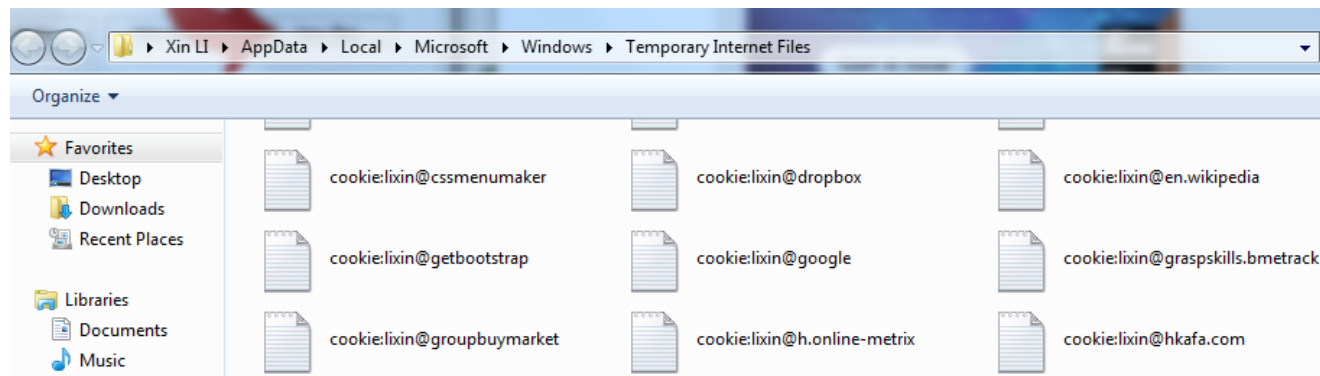
Cookie Fact

- ▶ Cookies are sent from the server to the client via “Set-Cookie” headers
 - ▶ `Set-Cookie: NAME=VALUE; expires=DATE; path=PATH; domain=DOMAIN_NAME; secure`
- ▶ Each cookie on the user’s computer is connected to a particular domain.
- ▶ Each cookie be used to store up to 4kB of data.
- ▶ A maximum number of cookies can be stored on a user’s PC per domain is browser dependent
 - ▶ Usually a few tens
- ▶ Cookies can be created with JavaScript or PHP

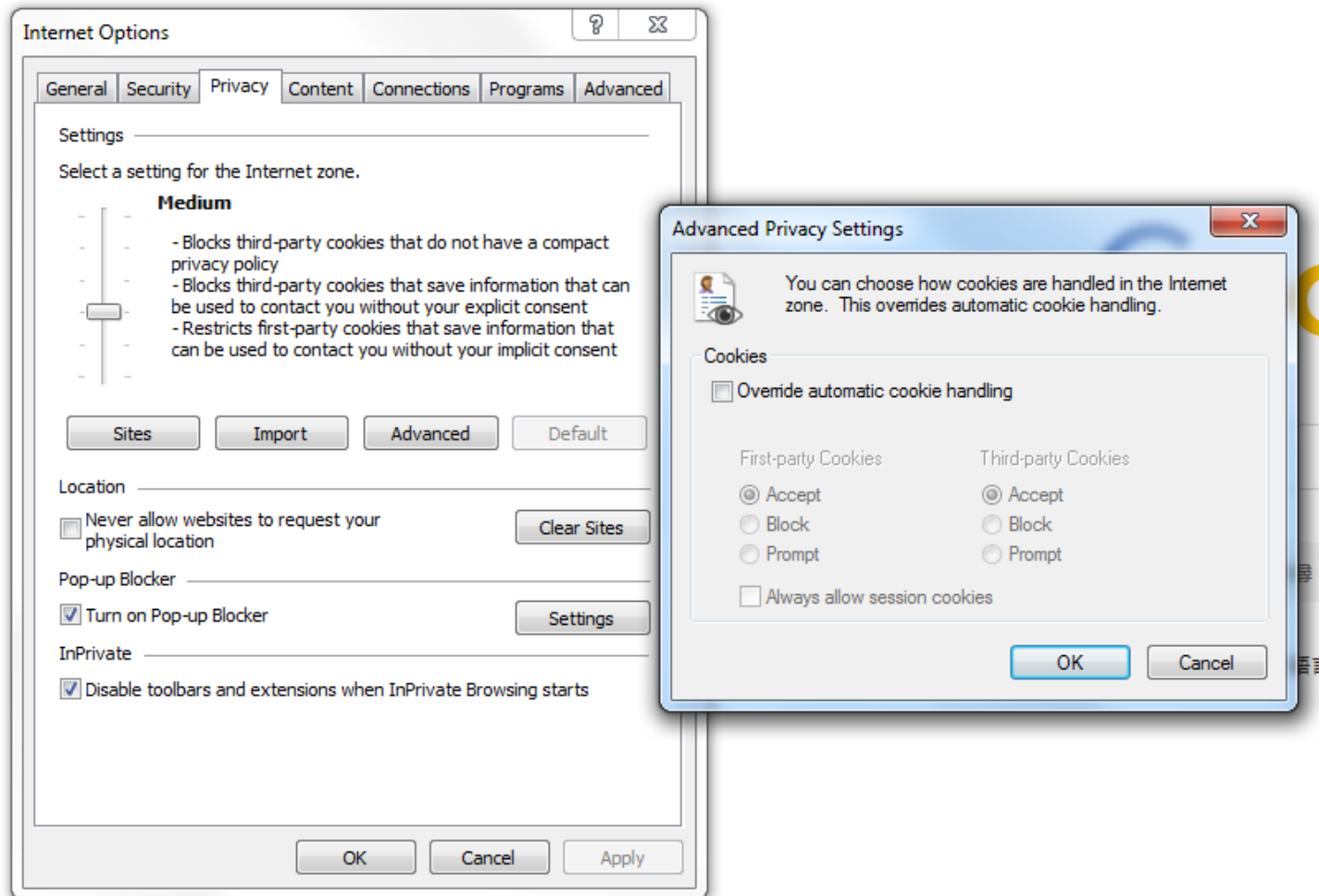


The USER is in Control

- ▶ **Cookies are stored client-side**
 - ▶ Cookie is stored (or persistent) only if there is an expiry date
 - ▶ Otherwise it is deleted when leaving browser
 - ▶ They can be turned on and off at will by the user
 - ▶ Never trust them completely: they can be easily viewed, modified or created by a 3rd party
- ▶ Exact location depends on browser, e.g. IE cookies



Example: Default IE Cookie Setting



Create PHP Cookies

- ▶ Directly manipulating the HTTP header using the PHP

`header ()` function

```
<?php
    header("Set-Cookie: mycookie=myvalue; path=/; domain=.example.com");
    # To make the cookie available on all subdomains of example.com, you'd
    set it to '.example.com'.
?>
```

PHP

- ▶ Use the PHP `setcookie ()` function

- ▶ `setcookie (name, value, expire, path, domain, secure)`

```
<?php
    setcookie("MyCookie",      $value, time()+3600*24);
    setcookie("AnotherCookie", $value, time()+3600);
?>
```

PHP



The `setcookie()` Function

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

- ▶ **Name** and **value** correspond to `$_COOKIE[$name] = $value`
- ▶ **Expiration** – cookie will no longer be read after the expiration
 - ▶ Useful to use time in seconds relative to the present:
 - ▶ `time() + time in seconds until expiration`
- ▶ **Path** and **domain** refer to where on the site the cookie is valid
 - ▶ Usually `/` for path and the top-level domain (`yoursitename.com`)
- ▶ To **delete a cookie**, set a new cookie with same arguments but expiration in the past



Access PHP Cookies

- ▶ The `$_COOKIE` superglobal array makes a cookie a **key-value pairing**
- ▶ Refer to `$_COOKIE` to retrieve a cookie
- ▶ Check with `isset($_COOKIE[$cookie_name])` before trying to use the cookie's value
- ▶ Cookies can only be set before any output is sent (e.g. `echo`, `print`) and before `<html><head>`.
- ▶ Cookies only become visible on the next page load



Example: Set and Access Cookie

```
<?php
# createCookie.php Create and access a cookie

$cookie_name = "user";
$cookie_value = "John Doe";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/");
?>
```

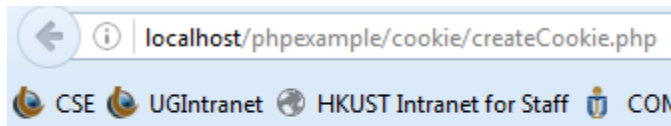
Before <html><body> and output

86400 = 1 day

```
<html><body>
<?php
    if(!isset($_COOKIE[$cookie_name])) {
        echo "Cookie named '" . $cookie_name . "' is not set!";
    } else {
        echo "Cookie '" . $cookie_name . "' is set!<br>";
        echo "Value is: " . $_COOKIE[$cookie_name];
    }
?>
</body></html>
```

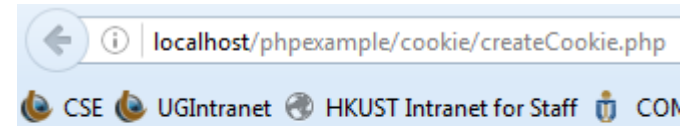
PHP

1st run



Cookie named 'user' is not set!

2nd run



Cookie 'user' is set!
Value is: John Doe

Example: Cookie with Multiple Items

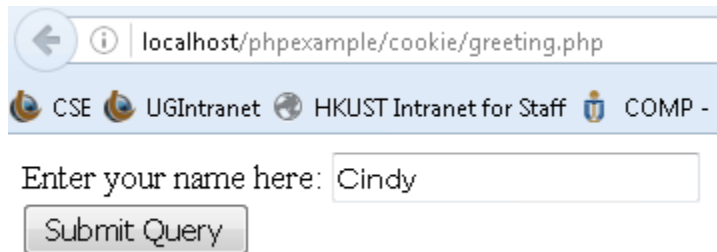
```
<?php
# multipleItemCookie.php
# set a cookie with 4 pieces of data
$strAddress = $_SERVER["REMOTE_ADDR"];
$strBrowser = $_SERVER["HTTP_USER_AGENT"];
$strServerName = $_SERVER["SERVER_NAME"];
$strInfo = "$strAddress::$strBrowser::$strServerName";
setcookie ("cookie4",$strInfo, time()+7200);
?>

<?php
# use explode() to retrieve the 4 pieces of data
$strReadCookie = $_COOKIE["cookie4"];
$arrListOfStrings = explode ("::", $strReadCookie);
echo "<p>$strInfo</p>";
echo "<p>Your IP address is: $arrListOfStrings[1] </p>";
echo "<p>Client Browser is: $arrListOfStrings[2] </p>";
echo "<p>Server name is: $arrListOfStrings[3] </p>";
?>
```

PHP

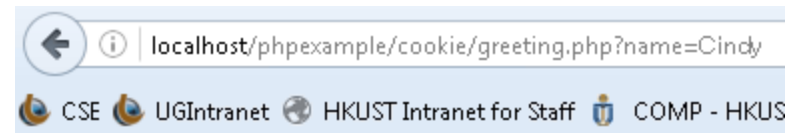
Wrap-up Example: `greeting.php`

- ▶ First visit: form with a text field for user's name
- ▶ Subsequent visits: Welcome message with the name
- ▶ Store the name field in a cookie:
 - ▶ Key: "name"; value: the user's name input into the form
 - ▶ Remember: when a cookie is set (the `setcookie()` function call is made), the cookie can only be accessed on the next request



A screenshot of a web browser showing the first visit to the page. The address bar displays `localhost/phpexample/cookie/greeting.php`. The browser's tab bar shows several tabs: "CSE", "UGIntranet", "HKUST Intranet for Staff", and "COMP - I". The main content area contains a form with the text "Enter your name here:" followed by a text input field containing the name "Cindy". Below the input field is a button labeled "Submit Query".

1st run

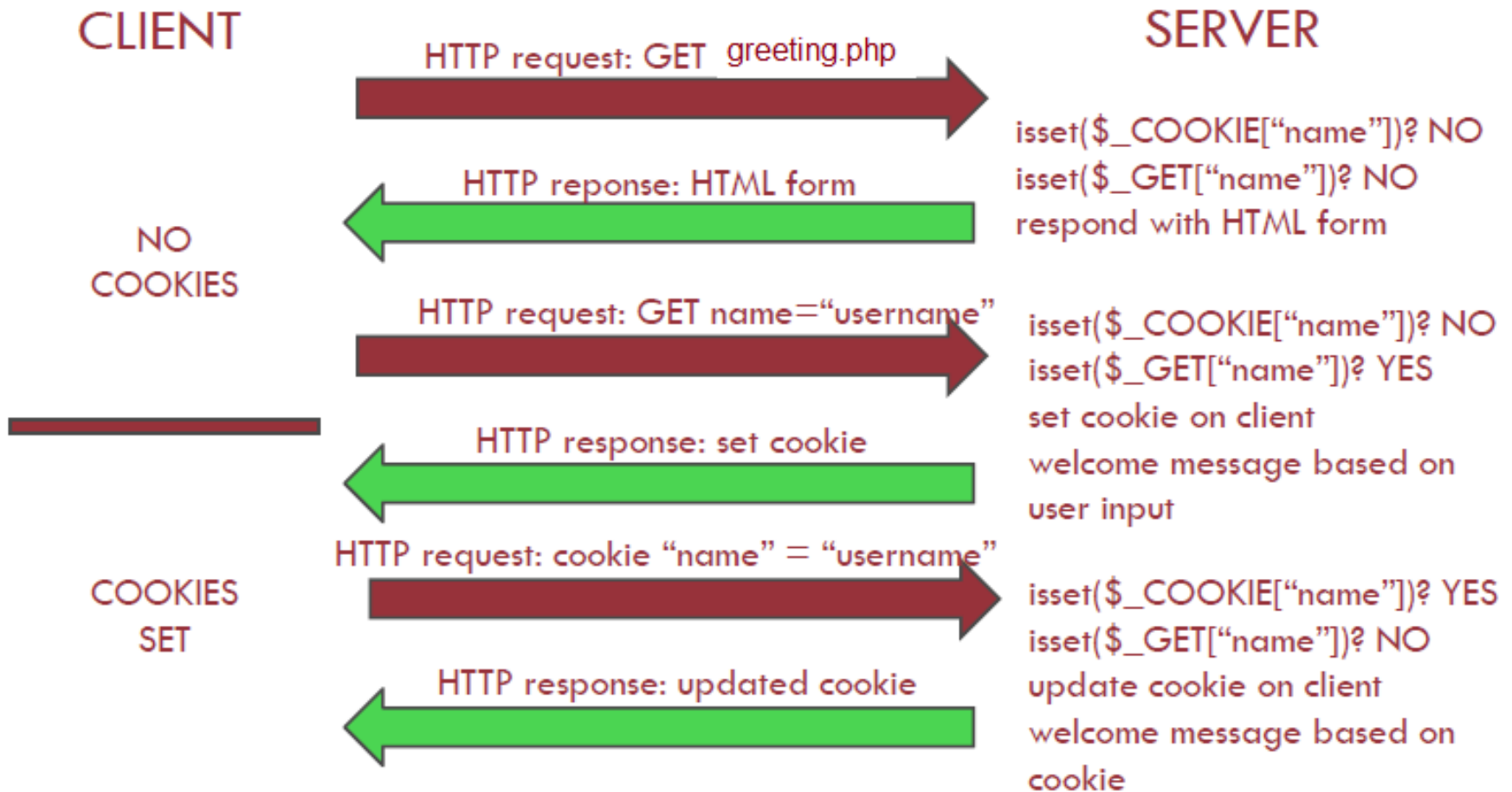


A screenshot of a web browser showing the fifth visit to the page. The address bar displays `localhost/phpexample/cookie/greeting.php?name=Cindy`. The browser's tab bar shows the same tabs as the first screenshot. The main content area displays the message "Welcome Cindy! You've visited 5 times".

5th run



Contents of HTTP Request and Response



Case 1: Cookies Already Set

```
# case 1: cookies already set

if(isset($_COOKIE["name"])) {
    $cookie_exp = time()+60*60; // one hour
    $name = $_COOKIE["name"];
    setcookie("name", $name, $cookie_exp);
    if (isset($_COOKIE["visits"])) { $num_visits =
$_COOKIE["visits"]+1;
        setcookie("visits", $num_visits, $cookie_exp);
    }
    echo "Welcome $name! ";
    if (isset($_COOKIE["visits"])) {
        echo "You've visited $num_visits times"; }
}
```

PHP



Case 2&3: First and Second Visits

```
# case 2: upon submission of form

else if (isset($_GET["name"])) {
    $name = $_GET["name"];
    setcookie("name", $name, $cookie_exp);
    setcookie("visits", 2, $cookie_exp);
    echo "Welcome $name! This is your second visit.";
}

# case 3: first visit: need to show form

else {
    # HereDoc
    # Complex data types in strings must be surrounded by {} for
    # them to be parsed as variables
    $form = <<< FORM
    <form action="{$_SERVER["PHP_SELF"]}" method="get">
        Enter your name here: <input type="text" name="name" />
    <br /><input type="submit" />
    </form>
    FORM;
    echo $form;
```



Sessions

`$_SESSION`

Cookies vs. Sessions

A **session** is a semi-permanent interactive information interchange, between two or more communicating devices

- ▶ Two main disadvantages of cookies
 - ▶ Limited in size by browser
 - ▶ Stored client-side → can be tampered with
- ▶ Sessions store user data on the **server-side**
 - ▶ Limited only by server space
 - ▶ Cannot be modified by users
- ▶ A potential downside to sessions is that they expire when the browser is closed

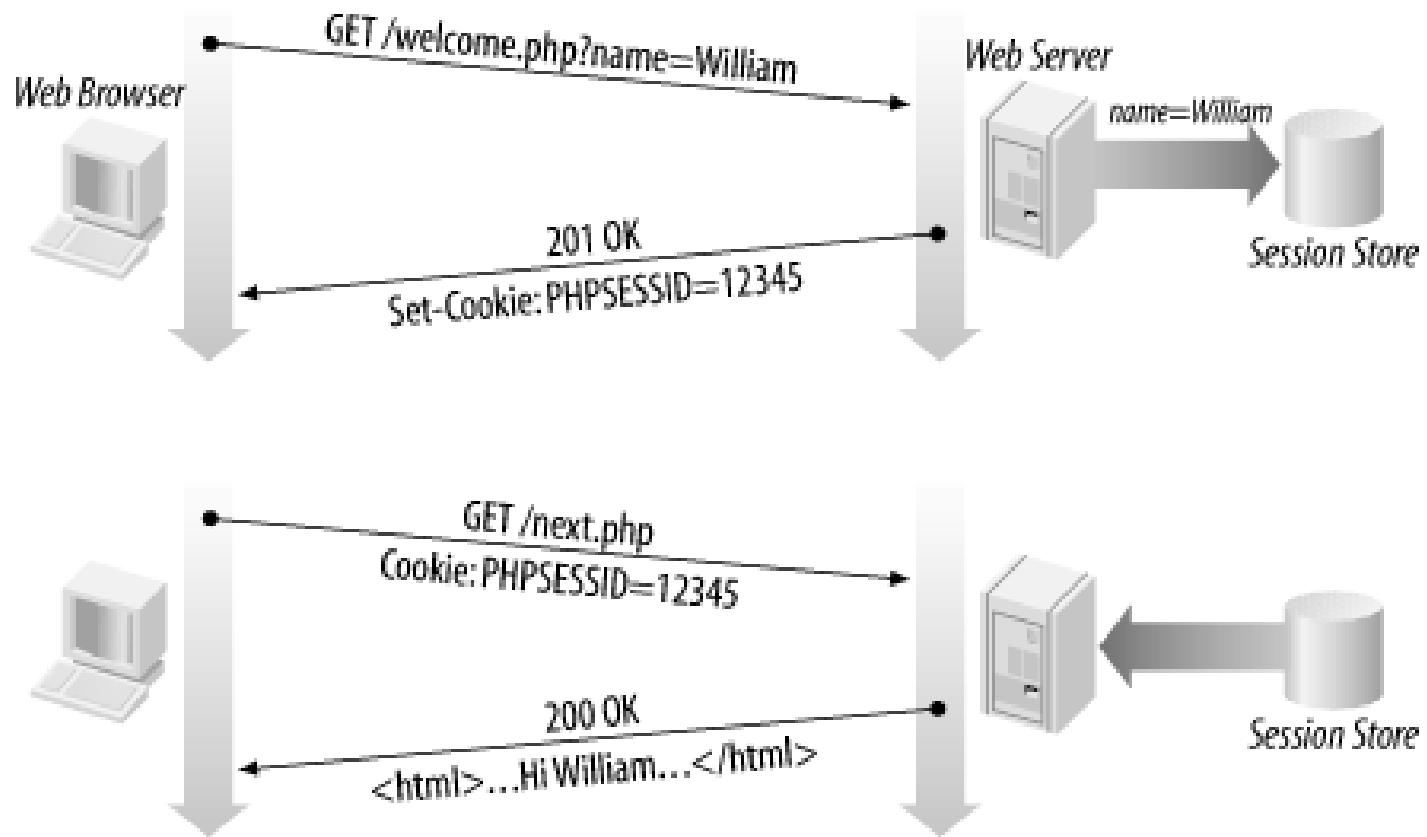


How Session Works?

- ▶ The first time a web client visits a server, the server sends a unique "session ID" to the web client for the client to keep.
 - ▶ Session ID is typically stored in the cookies.
- ▶ The session ID is used by the server to identify the client.
- ▶ For each session ID created, the server also creates a storage space. Server-side scripts that receive the same session ID share the same storage space.
 - ▶ The storage space is typically implemented as a map-like data structure.
 - ▶ In PHP, it is an associative array named `$_SESSION[]`.
- ▶ A session's "storage space" is only kept alive for a period of time (session period) or until it is explicitly deleted.



Example



Crucially, sessions are easy to implement as **PHP does all the work!**

When should you use sessions?

- ▶ Need for data to be stored on the server
- ▶ Unique session information for each user
- ▶ Transient data, only relevant for short time
- ▶ More secure, once established, no data is sent back and forth between the machines
- ▶ Works even if cookies are disabled



PHP Session Start/Resume

- ▶ You must start up the session before using it
- ▶ Call `session_start()` at top of every page before `<html>` tag

```
<?php
    session_start();
?>

<html>
<body>
</body>
</html>
```

PHP

- ▶ This tells PHP that a session is requested.



PHP Session Start/Resume (cont.)

- ▶ PHP looks for a valid session ID in the `$_COOKIE` or `$_GET` superglobals
- ▶ If found
 - ▶ Initializes the data
- ▶ If not found
 - ▶ Create new session ID at the **server end**
 - ▶ Session ID looks `26fe536a534d3c7cde4297abb45e275a` to make it **unique**



PHP Session Access

- ▶ Access data using the **\$_SESSION** superglobal, just like **\$_COOKIE**, **\$_GET**, or **\$_POST**

```
<?php
#visitCountSession.php
session_start();
if (isset($_SESSION["count"])) {
    $_SESSION["count"] += 1;
    echo "You have visited here {$_SESSION["count"]} times";
}
else {
    $_SESSION["count"] = 1;
    echo "You have visited once";
}
?>
```

PHP



PHP Session Propagation

- ▶ Sessions need to pass the session id between pages as a user browses to track the session.
- ▶ It can do this in two ways:
 - ▶ Cookie propagation
 - ▶ URL propagation
- ▶ The default setup of a PHP server is to use both methods.
 - ▶ it checks whether the user has cookies enabled.
 - ▶ If cookies are on, PHP uses cookie propagation. If cookies are off it uses URL propagation.



Cookie Propagation

- ▶ A cookie is stored on the users PC containing the session id.
- ▶ It is read in whenever `session_start()` ; is called to initialize the session.
- ▶ Default behaviour is a cookie that expires when the browser is closed. Cookie properties can be modified with `session_set_cookie_params` if required.



URL Propagation

- ▶ The session id is propagated in the URL

- ▶ e.g.

...some_folder/index.php?sid=26fe536a534d3c7cde4297abb45e275a

- ▶ PHP provides a **global constant** to append the session id to any internal links, `SID`.

- ▶ e.g.

<a href="nextpage.php?<?=SID?>">Next page



Session Expiry

- ▶ **By default, PHP sessions expire:**
 - ▶ after a certain length of inactivity (default 1440s), the PHP garbage collection processes deletes session variables.
 - ▶ Important as most sessions will not be explicitly destroyed.
 - ▶ if propagated by cookies, default is to set a cookie that is destroyed when the browser is closed.
 - ▶ If URL propagated, session id is lost as soon as navigate away from the site.



`unset()` and `session_destroy()`

- ▶ Remove an individual element of the `$_SESSION` superglobal
 - ▶ `unset($_SESSION['key_name'])`
 - ▶ The session still exists and can be modified
- ▶ Destroy the entire session, remove all data
 - ▶ `session_destroy()`
 - ▶ Destroys all data registered to a session
 - ▶ Does not unset session global variables and cookies associated with the session
 - ▶ Need to call `session_start()` to start a new session
 - ▶ Not normally done - leave to timeout



Example: Destroying a Session

```
<?php
#destroy session
    session_start();
?>
<html>
<body>
<?php
// remove all session variables
    session_unset();

// destroy the session
    session_destroy();
?>

</body>
</html>
```

PHP

- ▶ A more complete example at
 - ▶ <http://php.net/manual/en/function.session-destroy.php>
-



Wrap-up Example: User Login

- ▶ **loginForm.php**
 - ▶ Create a form to input user name and password
- ▶ **login.php**
 - ▶ Validate user name and password
- ▶ **content.php**
 - ▶ If logged in, show content page
- ▶ **Logout.html**
 - ▶ Webpage for logout
- ▶ **Logout.php**
 - ▶ Delete session



Recap: a Comparison

	COOKIES	SESSIONS
Where is data stored	Locally on client	Remotely on server
Expiration?	Variable – determined when cookie is set	Session is destroyed when the browser is closed
Size limit?	Depends on browser	Depends only on server (practically no size limit)
Accessing information	\$_COOKIE	\$_SESSION
General use?	Remember small things about user, such as login name. Remember things after re-opening browser	Remember varying amount of data about the user in one browsing “session”

