PHP

Internet Engineering

Fall 2016

Bahador Bakhshi

CE & IT Department, Amirkabir University of Technology

Questions

- Q7) How does server process client's requests?
- Q7.1) How to code in server side?
- Q7.2) Which language? Syntax?
- Q7.3) How can I get valid user's data in server?
- Q7.4) Can I read/write access to HTTP headers
- Q7.5) The users must login to access the site!
- Q7.6) Can I use databases? How?

Outline

- Introduction to CGI
- Introduction to PHP
- PHP Basic
- Input Data Handling
- HTTP Headers
- Cookies & Session Management
- Database
- Error Handling
- XML

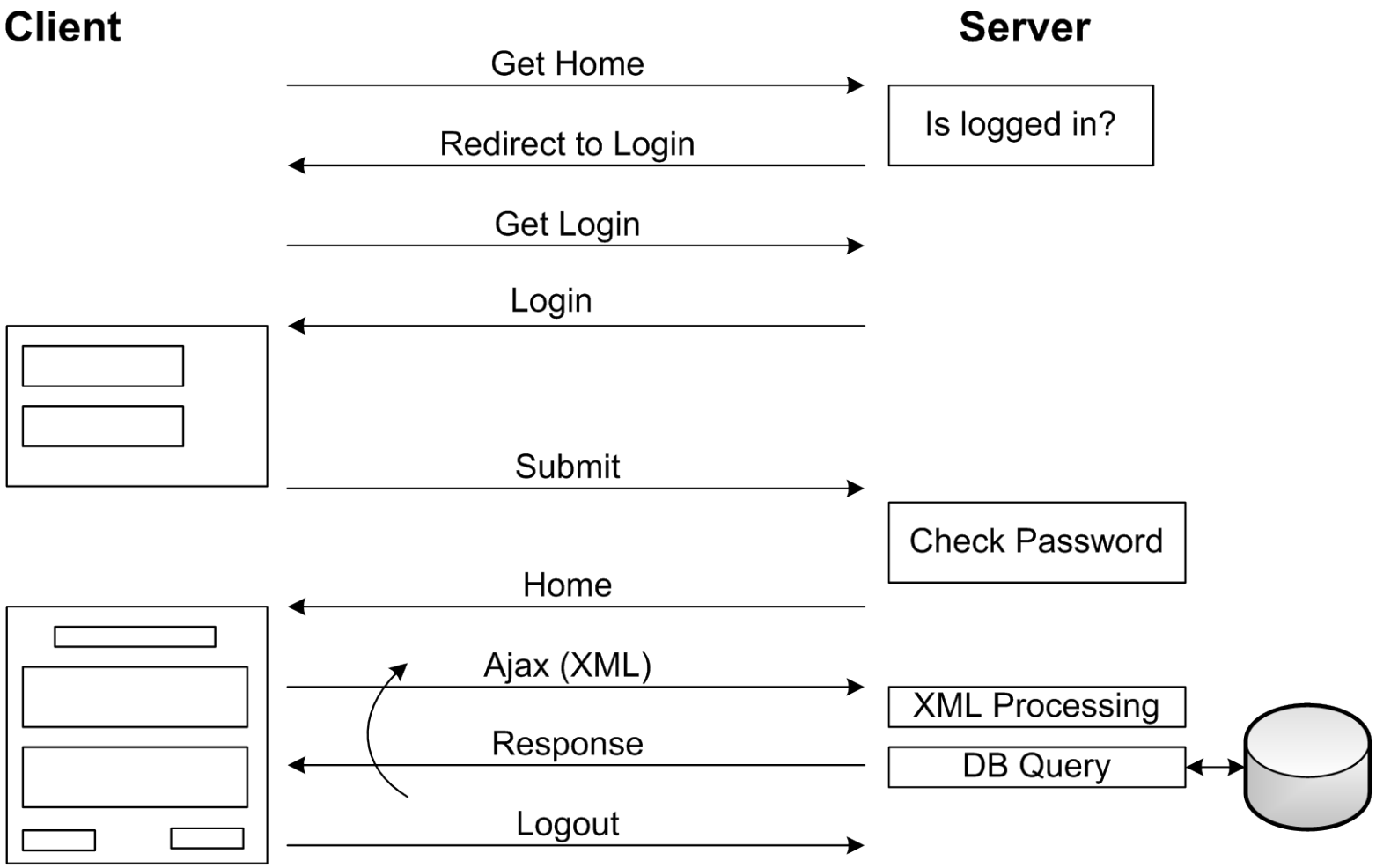
Outline

- Introduction to CGI
- Introduction to PHP
- PHP Basic
- Input Data Handling
- HTTP Headers
- Cookies & Session Management
- Database
- Error Handling
- XML

Introduction

- HTML/XHTML content is static
  - JavaScript & Ajax make pages more dynamic, but the content is almost static
- Dynamic content
  - Pages that look differently depending on the user who visits, status, processing requests, …
  - E.g. Search engines, web mails, …
- Web applications (hotel booking, web search applications, …) is not possible using only HTML/XHTML, CSS and JS; why?

Typical Web based Application (e.g., Gmail)



We need server side active code to perform actions & generate (dynamic) content

Common Gateway Interface

- We need code beside web servers
  - Web server by itself is not designed for data processing
- Initial idea
  - An external program can perform the processing
- Questions
  - How can client ask server to run an external program?!
    - HTTP?!!
  - How does web server exchange information with the external program?
    - Sending input data & Getting the output
    - The mechanism should be standard

Common Gateway Interface (cont'd)

- The Standard protocol for interfacing external application software with the web server
  - CGI 1.1 specified in RFC 3875, 2004
- The external program runs by HTTP requests & proper server configuration
- Information is passed from external software to the web server as the output on stdout
  - HTTP response is the output of the external program on the server machine
- Information can passed from the web server to the executable program according to HTTP request method

The “Hello World” CGI in C

```
#include <stdio.h>
int main(void){
  printf("Content-Type: text/html\r\n");
  printf("Connection: close\r\n");
  printf("\r\n \r\n");
  printf("<html><head></head>\r\n");
  printf("<body>\r\n");
  printf("Hello world.\r\n");
  printf("<br />\r\n");
  printf("Bye Bye\r\n");
  printf("</body></html>\r\n");
  return 0;
}
```

Header

Body

The “Hello World” CGI in Bash Script

```
#!/bin/bash

echo "Content-Type: text/html"
echo ""

echo "<html><head></head>"
echo "<body>"
echo "Hello world."
echo "<br />"
echo "Bye Bye"
echo "</body></html>"
```

Getting parameters from the client

- Parameters can be passed from the user to the CGI script through an html <form>

```
<form action="script.cgi" method="GET | POST">
    <input type="…" name="input1" />
    <input type="…" name="input2" />
    …
    <input type="…" name="inputN" />
</form>
```

- The script.cgi will get the parameters as:

```
input1=val1&input2=val2& … &inputN=valN
```

- The mechanism depends on the HTTP Method

Getting parameters from the client

- Parameters can be sent through the **GET** method
  - The CGI script will receive the parameters from the web server in an environment variable **$QUERY_STRING**
  - In C: You can access it by **getenv("QUERY_STRING")**
- Parameters can be passed through the **POST** method (in the body of the HTTP Request)
  - The CGI script will receive the parameters from the web server in the standard input (stdin)

Example

```
<html>
<head></head>
<body>
  <form action="cgi_form_get.cgi" method="GET">
  User: <input type="text" size="20" name="user" />
  <br />
  Password: <input type="text" size="20" name="pass" />
  <br />
  <input type="submit" value="Submit" name="submit"/>
  </form>
</body>
</html>
```

Example

```
#!/bin/bash

echo "Content-Type: text/html"
echo
echo

echo "<html><head></head>"
echo "<body>“
echo "The QUERY_STRING is: " $QUERY_STRING "<br />"
echo "Parameters are:<br />"
user=`echo $QUERY_STRING | cut -d"&" -f 1 | cut -d"="
   -f 2`
pass=`echo $QUERY_STRING | cut -d"&" -f 2 | cut -d"="
   -f 2`

echo $user $pass
echo "</body></html>"
```

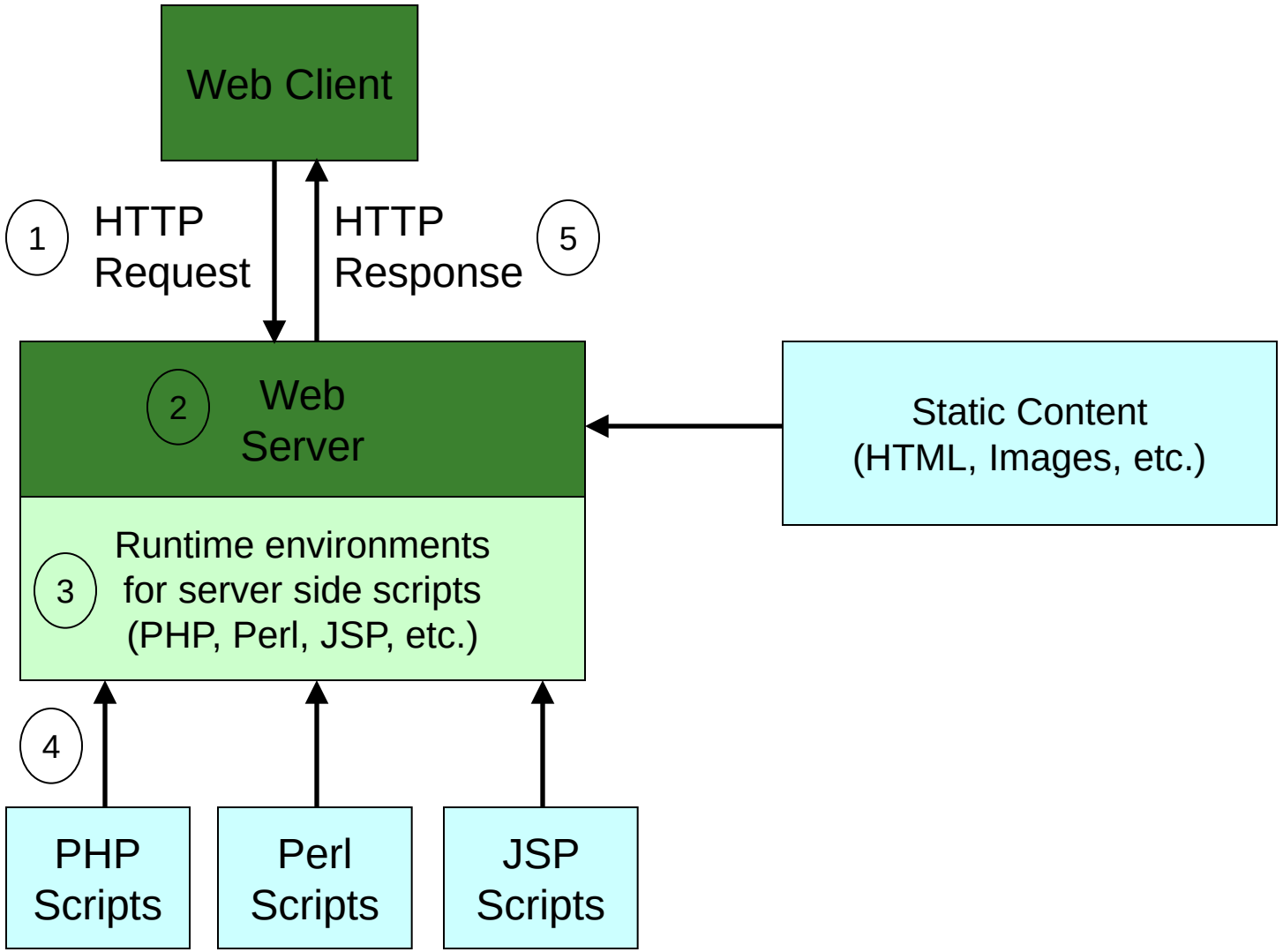
CGI Pros & Cons

- What is the main advantage(s) of CGI?
  - Any programming language can be used
- What the main drawback(s) of CGI?
  - We should generate whole HTML document in CGI
  - For each request, a new *process* is created
    - Process creation & termination & Inter-process communication overhead
  - Security is another major issue
- Any other way to run code in server side?

Solving CGI Problems

- Empower the server to run code!
- But,
  - Which programming language? HTML?!!!
    - Should we compile & debug web-pages?
  - Should web server interpret/compile the code?
    - Web servers are not build to be compiler!!
  - How to mix code & HTML?
- Answer: Interpreter as a web server *plugin* is responsible
  - Use any scripting language that its interpreter is available for web server, e.g., PHP runtime environment
  - Configure web server to use interpreter for a specific file types that contain mixed code & HTML, e.g., .php files
  - Web server run the interpreter for codes and uses the output

Overview of Server-Side Scripting



Overview of Server-Side Scripting

- 1) Web client sends a HTTP request to server
- 2) Web server determines how to retrieve the requested resource according configuration
  - .html, .jpg, ... → To be retrieve directly
  - .php → To be handled by the PHP module
- 3) Runtime environment does for example
  - Parses incoming request, generate outgoing response
  - Interpreting/executing the server-side scripts
  - Maintaining sessions

Overview of Server-Side Scripting

- 4) Runtime environment runs the requested script
  - Provides session & other status information
  - Identifies the code sections inside HTML
  - Runs the code and grabs the output
  - Generated output and HTML are assembled together which is the response to client
- 5) The HTTP response is sent to the web client by web server

Embed vs. External Server Side Code

- External code
  - A separated program: C, C++, …
  - Server runs it and sends its output back to client
- Embed code
  - Scripting inside the HTML
    - Embed programming interface within server
      - Which is called when server see the scripting directions
  - Examples
    - Perl: Apache mod_perl module to embed
    - Java Server Pages (JSP): Compiled and served by a JSP server
    - Python
    - PHP (the most common language)

Server Side Scripting Benefits

- How does server side scripting solve CGI problems?
  - We don't need to generate whole HTML by code
    - Only dynamic parts are coded
  - A process is not created per request
    - All requests are processed by the interpreter
      - Which is implemented as a library for web server process
    - Each request → A thread
      - Low creation & termination & inter-communication overhead
  - The run-time environment control the code
    - More secure execution

Major differences w.r.t client side programming

- Concurrency!!!
  - Each server side program (cgi, php, …) can (and usually) runs multiple times *concurrently*
    - A process/thread per request
  - Be very very careful about *shared* resources
- Security!!!
  - Each server side program allows client (including the hackers) to *run code* on your server
    - Vulnerable code → Hacker access
  - Be very very careful about *input* from the client

Outline

- Introduction to CGI
- Introduction to PHP
- PHP Basic
- Input Data Handling
- HTTP Headers
- Cookies & Session Management
- Database
- Error Handling
- XML

PHP Introduction

- PHP stands for
  - Originally: "Personal Home Pages"
  - Now: "PHP: Hypertext Preprocessor"
    - Recursive acronym such as GNU ;-)
- Widely-used scripting language
- Specially suited for Web development
  - Server side scripting → Dynamic Content
  - Typically runs on a web server that takes PHP as input and gives out HTML pages as output

PHP Features

- Open source & free
- A syntax similar to C and Java
- Connects with 20+ databases
- Version 5+ supports OOP
- Multi-platform compatible
  - Linux & Windows & Wide range of web servers
- Rich library: Over 1000 built-in functions
- Easy to learn

PHP Scripts

- Typically file ends in .php
  - Set by the web server configuration
- PHP scripts run when sent a GET/POST request to them
- PHP commands can make up an entire file, or can be contained in html
  - Server recognizes embedded script and executes
- Separated in files with the <?php ?> tag
  - Or <? ?> tag
  - Can be placed anywhere in the document
- *Result* passed to browser, source isn't visible

PHP in Action

- LAMP (Linux, Apache, MySQL, PHP)
  - WAMP, XAMP, … for other platforms
- Installation
  - From source:
    - Apache: http://httpd.apache.org/
    - PHP: http://www.php.net
    - MySQL: http://www.mysql.com/
  - Fedora:
    - Apache: **`yum install httpd httpd-tools httpd-devel`**
    - PHP: **`yum install php php-common php-cli php-mysql`**
    - MySQL: **`yum install mysql mysql-server mysql-devel`**

27

The PHP “Hello World”: Server Side

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<?php
  print "<h1>Hello World</h1>";
?>
</body>
</html>
```

Sent to client in “copy mode”

Parsed and output is sent to client in “interpret mode”

The PHP “Hello World”: Client Side

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>

<h1>Hello World</h1>

</body>
</html>
```

Outline

- Introduction to CGI
- Introduction to PHP
- PHP Basic
- Input Data Handling
- HTTP Headers
- Cookies & Session Management
- Database
- Error Handling
- XML

Syntax

- The syntax of PHP is very similar to C/C++
- PHP & C/C++ Similarities:
  - Case-sensitive for variables
  - Required semicolon after each statement
  - Commenting `//` `/* */`
  - Compound statements: `{ }`
  - Operators: Assignment, Arithmetic, Comparison, and Logical operator are the same
  - Loop: `for`, `while`, `do-while`
  - Conditionals statements: `if`, `else`, `switch-case`

Syntax (cont'd)

- PHP & C/C++ Differences:
  - Variables begin with **$** sign
    - No explicit declaration of variable types
  - Functions
    - Defined by **function** keyword
    - Functions-within-a-function is allowed
    - Case-*insensitive* function names
  - Single line commenting is also allowed by **#**
  - Strings are enclosed in **" "** and also **' '**
  - Operators: comparison by **<>**, concatenation by **.**
  - Loop by **foreach(**$array **as** $var**)**
  - Conditional statements by **if-elseif-else**

Scope of Variables

- The scope of a variable defined within a function is local to that function
- A variable defined in the main body of code has a global scope
- To use global variables in functions, it is referenced "**global**" keyword

```
<?php
   $gvar = 10;
   function f(){
       global $gvar;
       $lvar = $gvar;
   }
?>
```

Arrays

- Similar to C/C++/… the index of array can be an integer number
  - Numeric array
- Similar to JS the index of array can be everything
  - Associative array
    - Mapping between key (index) and value
- Similar to other languages array containing one or more arrays
  - Multidimensional array
- Arrays can also be created by **`array`** function

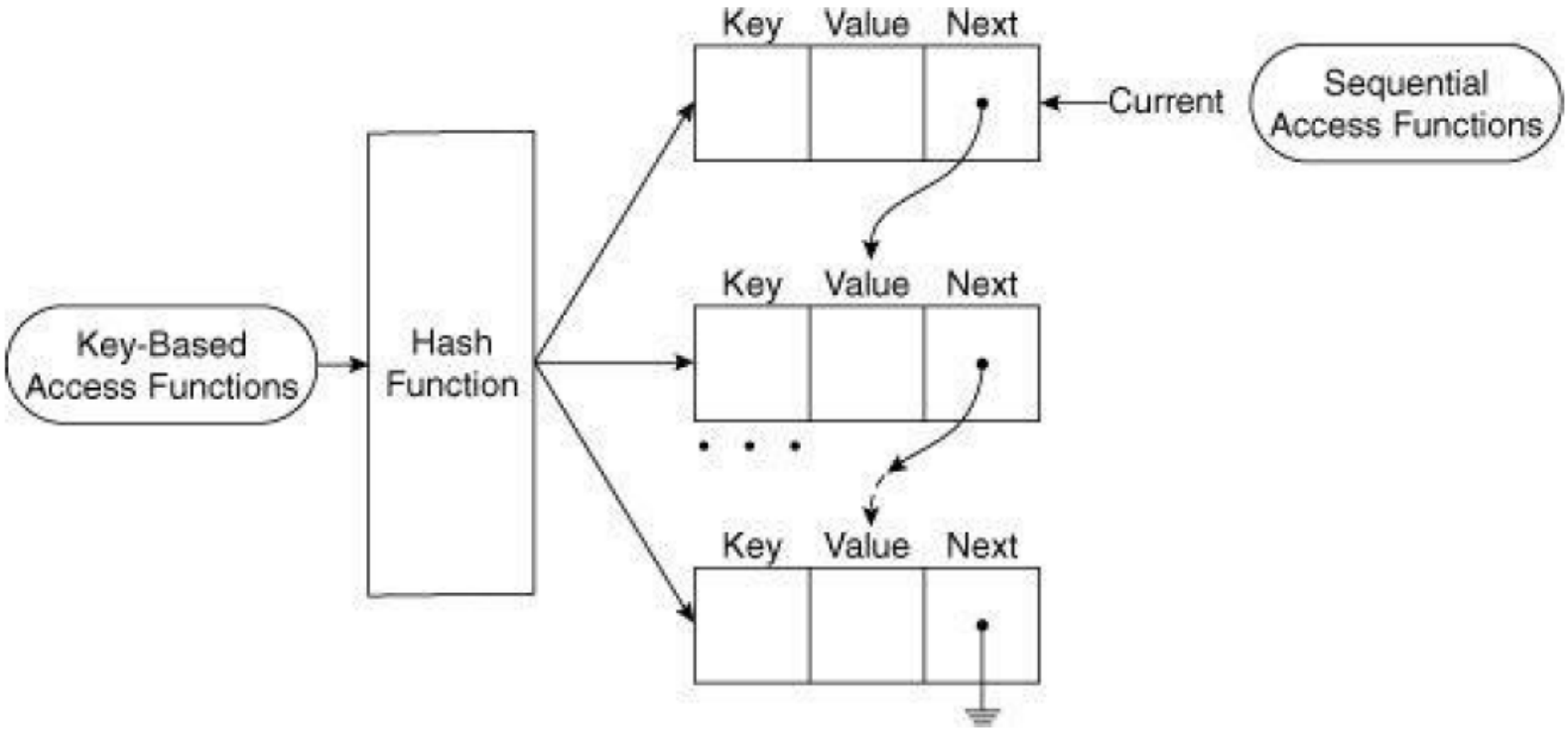
Arrays (cont'd)

- Numeric arrays
  - `$cars[0]="Saab"; $cars[1]="Volvo";`
    `$cars[2]="BMW"; $cars[3]="Toyota";`
  - `$cars=array("Saab","Volvo","BMW","Toyota");`
- Associative arrays
  - `$ascii["A"]=65; $ascii["B"]=66;`
    `$ascii["C"]=67`
  - `$ascii = array("A"=>65, "B"=>66, "C"=>67);`
- Multidimensional arrays
  - `$std=array("one"=>array("Ali", 1122, 20),`
    `"two"=>array("Hossein", 1133, 15));`

Array Internal Implementation

In fact, array is a mapping between keys and values

```
$keys = array_keys($array);
$values = array_values($array);
```



Super Global Arrays

- Several predefined variables in PHP are "superglobals"
  - Available in all scopes throughout a script
  - No need to have `global $variable;` declaration
  - Maintained by PHP runtime environment
- **`$GLOBALS`**: All global variables, the variable names are the keys of the array
- **`$_GET`**: Variables passed in URL's query part
- **`$_POST`**: Variables passed by HTTP POST
- **`$_FILES`**: Uploaded file information
- **`$_COOKIE`**: Cookies sent by HTTP cookie header
- **`$_REQUEST`**: Contains **`$_GET`**, **`$_POST`** and **`$_COOKIE`**
- **`$_SESSION`**: Session variables

Super Global Arrays (cont'd)

- **$_SERVER**: Information such as headers, server & client
  - Examples of the important keys
  - **'SERVER_ADDR'**: The IP address of the server
  - **'SERVER_NAME'**: The name of the server host
  - **'SERVER_PORT'**: The port of web server
  - **'REQUEST_METHOD'**: The HTTP request method
  - **'HTTP_USER_AGENT'**: Contents of the HTTP User-Agent
  - **'REMOTE_ADDR'**: The IP address of client
- …
- Complete list: php.net/manual/en/index.php

Input & Output in Web Applications

- Console I/O
  - Console output of script is gathered by PHP runtime then passed to web server & finally sent to client
  - (Usually) No console input (stdin)
    - Input is given by web server to PHP scripts
      - Usually, the input is the values got from client
        - Forms, Ajax, …
    - Will be discussed later
- File I/O: Access to files for read/write
- Database: Connect and read/write database

Output: echo & print & var_dump

```
<?php
$foo = 25;              // Numerical variable
$bar = "Hello";         // String variable

echo $bar."\n";                 // Outputs Hello
echo $foo,$bar,"\n";            // Outputs 25Hello
echo "5x5=".$foo."\n";          // Outputs 5x5=25
echo "5x5=$foo\n";              // Outputs 5x5=25
echo '5x5=$foo\n';              // Outputs 5x5=$foo\n
print "\n";                     // newline
print "Output is ".$foo; // Output is 25
var_dump($foo);                 // int(25)
?>
```

Filesystem Operations

- PHP filesystem operations are similar to C
  - **`fopen(), fgetc(), fputc(), fread(), fwrite(), fseek(), rewind(), flock(), fclose()`**
- **`fopen`** opens URL of supported protocols
  - file://, http://, ftp://, …
  - php://stdin, php://stdout, php://stderr
- To open binary files safely: **`b`**

Filesystem Operations (Security)

- To increase security of web-servers, the **`fopen`** function may be disabled
  - So, none of the previous functions can be used ☹
- Alternative functions (limited functionalities)
- **`file_get_contents`**: To read file content into a string
- **`file_put_contents`**: To write a string into a file

Simple Web Page Counter

```
<?php
$data = file_get_contents("counter");
$data = $data + 1;
file_put_contents("counter", $data
, LOCK_EX
);
echo "This page has been viewed " . $data .
  " times ";
?>
```

This code works, but ….? What is the problem?

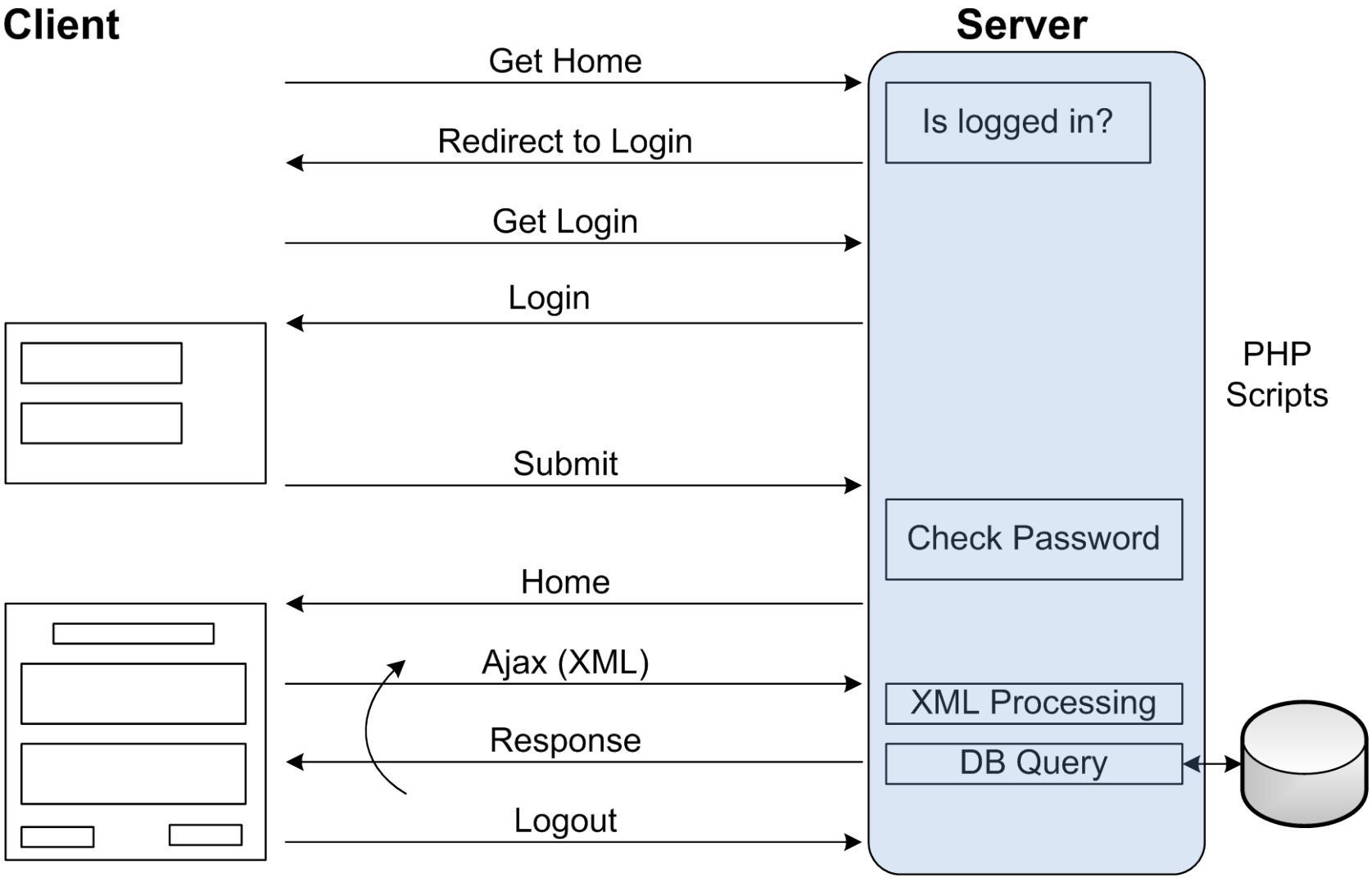
Does LOCK_EX solve all problems? What is solution?

PHP Includes

- Complex server side processing → lot of PHP codes
  - Avoid mixing HTML design & PHP
  - Break processing into multiple files (team working)
- Four functions to insert code from external files
  - **`include()`**: Try to insert file, continues if cannot find it
    - **`include_once("A")`**: does not include "A" if it is already included even by other included files "B"
  - **`require()`**: Try to insert external file, dies if cannot find it
    - **`require_once()`**: does not include if file is already included
- The included code is interpreted & run (if is not function)
- An implementation of *server side include* (SSI)

```
<html> <body> <?php include("header.php"); ?>
```

PHP in Web Applications



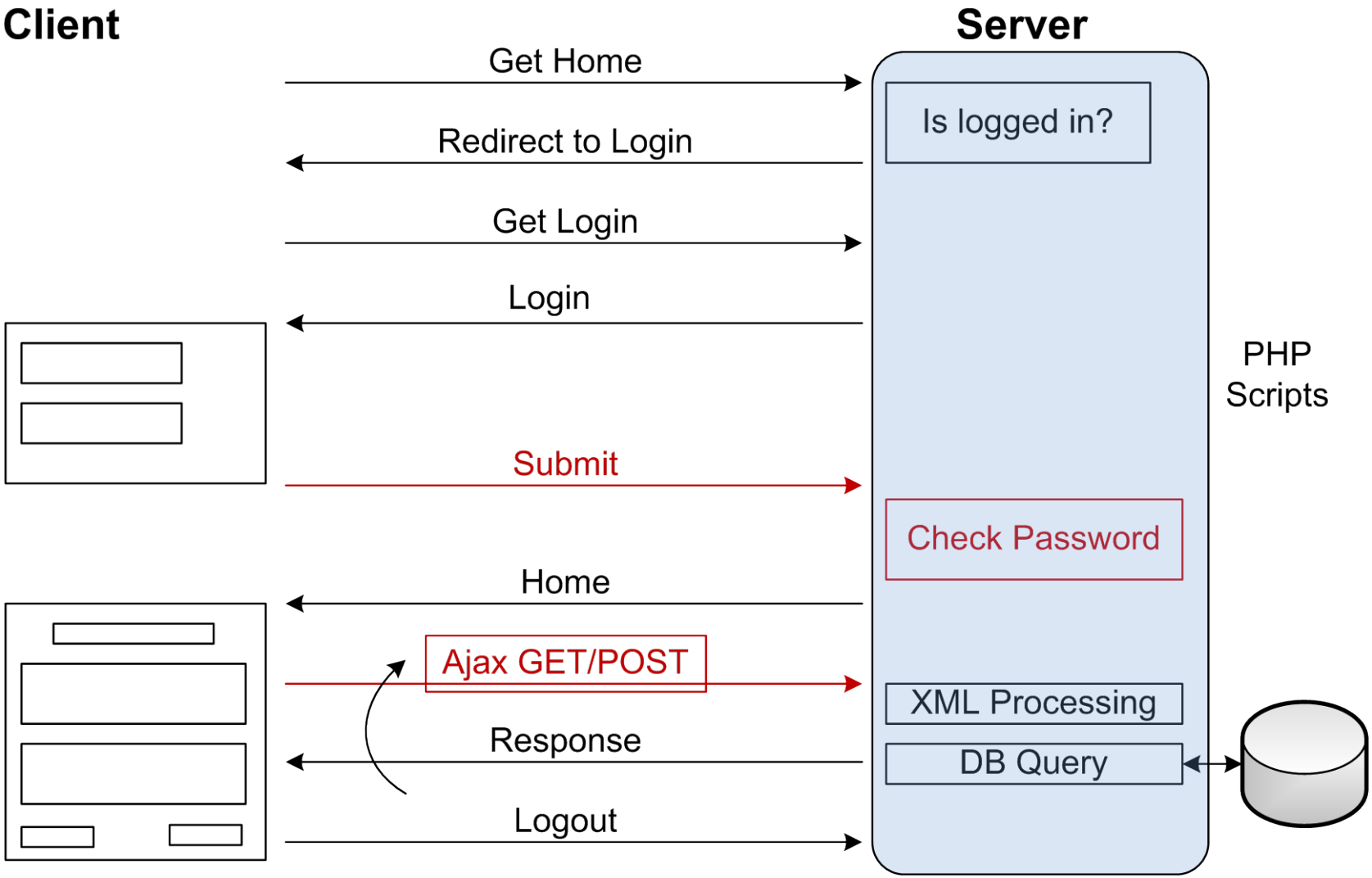
PHP in Web Applications (cont'd)

- What do we implement by PHP?
- Redirection
  - HTTP header modification
- Input data
  - Receive data from HTML forms
- Login & Logout
  - Session management
- Ajax request processing
  - XML parser
- Database access
- Error handling

Outline

- Introduction to CGI
- Introduction to PHP
- PHP Basic
- Input Data Handling
- HTTP Headers
- Cookies & Session Management
- Database
- Error Handling
- XML

PHP in Web Applications



Input Data Handling

- One of the main functionalities of server side scripting is to process user input data, e.g.
  - Save data on server
  - Login & Sessions
  - Query from database server
  - …
- Input data from HTML forms or Ajax or …
  - GET method
  - POST method
    - File upload

Input Data Handling (cont'd)

- Major steps of input data handling:
- 1) Read the data
  - How to read the URL query part? Post data? File?
- 2) Check presence & existence
  - Is variable set? Is it empty?
- 3) Validation
  - Is data valid? Correct format?
- 4) Processing
  - Application dependent, e.g., query to DB, ….

1) Reading Submitted Data

- Main feature: data sent in “URL Query Part” or “Packet Body” are automatically available to PHP scripts by the run-time environment
  - Does not matter HTML form or Ajax
- The PHP pre-assigned **$_GET** and **$_POST** variables are used to retrieve the data
  - The predefined **$_REQUEST** variable contains the contents of **$_GET**, **$_POST**, **$_COOKIE**
    - The **$_REQUEST** variable can be used to collect form data sent with both GET and POST methods

1) Reading Submitted Data (cont'd)

- **$_GET**, **$_POST**, and **$_REQUEST** are associative arrays
  - Key is the **name** attribute of input element in a form
  - Value is the value of the input element in a form
- HTML

```
<form method="GET" action="index.php">
  <input type="text" name="grade" value="">
</form>
```

- PHP

```
$g = $_GET["grade"];
```

2) Checking Input Presence/Existence

- **`isset($var)`** is false if and only if **`$var`** is NULL
  - i.e., either **`$var`** does not exist or is never assigned a value
  - Use this function to check if a check box, radio button, or select box list has a value
- **`empty($var)`** is true if **`$var`** is 0, empty string, NULL, or FALSE
  - Use this function to check if a text field, password field, or text area has a value that is not an empty string
    - These input fields are always set → **`isset`** does *not* work!

Form Processing Example

```
<form method="post" action="form.php">
      Submit By Post!!
<fieldset>
<legend>University Grade</legend>
      <input type="radio" name="grade" value="BS" /> BS
      <input type="radio" name="grade" value="MS" /> MS
      <input type="radio" name="grade" value="PhD" /> PhD
</fieldset>

<fieldset>
<legend><em>Web Development Skills</em></legend>
      <input type="checkbox" name="skill_1" value="html" />HTML
      <input type="checkbox" name="skill_2" value="xhtml" />XHTML
      <input type="checkbox" name="skill_3" value="cs" />CSS
      <input type="checkbox" name="skill_4" value="js" />JavaScript
      <input type="checkbox" name="skill_5" value="aspnet" />ASP.Net
      <input type="checkbox" name="skill_6" value="php" />PHP
</fieldset>

Favorite Programming Language:
<select name="lang">
<option value="c">C</option>
<option value="java">Java</option>
<option value="awk">AWK</option>
</select>
<input type="submit" value="Submit" />
</form>
```

Form Processing Example (cont'd)

```
<form method="get" action="form.php">
    <fieldset>
        <legend> Submit by GET </legend>
        Title: <input type="text" length="20" name="title" />
        Name: <input type="text" length="20" name="name" />
        Family: <input type="text" length="20" name="fam" />
        <input type="submit" value="Submit" />
    </fieldset>
</form>
```

Form Processing Example (cont'd)

```
$grade = $_POST["grade"];
$lang  = $_POST["lang"];

if(isset($grade))
    echo "You are ". $grade;
else
    echo "I don't know your grade";
echo "<br />";

echo "You are master in ";
for($i = 1; $i < 7; $i++)
    if(isset($_POST["skill_".$i]))
          echo $_POST["skill_".$i]. " ";

echo "<br />";
echo "You love ". $lang;
```

Form Processing Example (cont'd)

```
$name = $_GET["name"];
$fam  = $_GET["fam"];
$title  = $_GET["title"];

if((! empty($name) > 0) && (! empty($fam) > 0) && (!
empty($title) > 0)){
    echo "A message by GET <br />";
    echo "<h2> Welcome " . $title ." ". $name ." ". $fam ."
</h2>";
}
```

File Upload Handling

- **$_FILES** is a *two* dimensional array stores data of uploaded files from a client
- The first key in **$_FILES** is the **name** attribute of the input element with **type="file"**
- Second key is a parameter of the file
  - **$_FILES**[file_name]["**name**"] - the name of the file
  - **$_FILES**[file_name]["**type**"] - the type of the file
  - **$_FILES**[file_name]["**size**"] - the size in bytes of the file
  - **$_FILES**[file_name]["**tmp_name**"] - the name of the temporary copy of the file stored on the server
  - **$_FILES**[file_name]["**error**"] - the error code resulting from the file upload

File Upload Handling (cont'd)

- When file is uploaded successfully, it is stored in a temporary location in the server
- The temporary copied files disappears when the script ends
- To save (move) the temporary file

```
move_uploaded_file($_FILES["file"]
  ["tmp_name"], "permanent
  location");
```

File Upload Example

```
<form action="http://127.0.0.1/IE/php/form.php"
   method="post" name="uploadFrom"
   enctype="multipart/form-data">
    <fieldset>
   <input type="file" name="myfile" />  <input
   type="submit" value="Submit" />
    </fieldset>
</form>
```

File Upload Example (cont'd)

```
if(isset($_FILES["myfile"])){
   if($_FILES["myfile"]["error"] > 0){
       echo "Error: " . $_FILES["myfile"]["error"] . "<br />";
   }
   else{
        echo "Upload: " . $_FILES["myfile"]["name"] . "<br />";
        echo "Type: " . $_FILES["myfile"]["type"] . "<br />";
        echo "Size: " . ($_FILES["myfile"]["size"] / 1024) . " Kb<br />";
        echo "Temp Store: " . $_FILES["myfile"]["tmp_name"] . "<br />";

        if (file_exists("upload/" . $_FILES["myfile"]["name"])){
          echo $_FILES["myfile"]["name"] . " already exists. ";
        }
        else{
           move_uploaded_file($_FILES["myfile"]["tmp_name"],"upload/" .
           $_FILES["myfile"]["name"]);
           echo "Stored in: " . "upload/" . $_FILES["myfile"]["name"];
        }
   }
}
```

3) Input Data Validation

- Be very very careful about input data
  - Maybe they are coming from bad guys
- There is a HTML form corresponding to PHP
  - On client side, we (developers) try to validate input data by JavaScript
    - We cannot fully & completely validate the data
  - What happen if attacker want to inject code/data
    - He does not use our forms
    - No data validation on client side
- Server side data validation is *required*

PHP Filters

- PHP filters to make data filtering easier
- Two kinds of filters:
  - Validating filters:
    - Are used to validate user input
      - Strict format rules (like URL or E-Mail validating)
    - Returns the expected type on success or FALSE on failure
  - Sanitizing filters:
    - To allow or disallow specified characters in a string
      - Remove the invalid characters
    - Always return a valid output

PHP Filters (cont'd)

- Filters are applied by these functions:
- **`filter_var(`**variable, filter**`)`**: Filters a single variable
- **`filter_var_array(`**array of variables, array of filters**`)`**: Filter several variables with a set of filters
- **`filter_input(`**type, variable, filter**`)`**: Get one input variable from given type and filter it, e.g. **`INPUT_GET`**, **`INPUT_POST`**, …
- **`filter_input_array(`**type, filters**`)`**: Get several input variables and filter them with specified filters

PHP Filters (cont'd)

- Each filter has a unique id (integer number)
  - **`FILTER_VALIDATE_INT`** → 257
  - **`FILTER_VALIDATE_FLOAT`** → 259
  - Filtering functions decide based on the value
- A filter can have options and flags
  - E.g., for **`FILTER_VALIDATE_INT`**
    - *Option*: **`max_range`** and **`min_range`**
    - *Flag*: **`FILTER_FLAG_ALLOW_OCTAL`**
- Flag and options are passed using associative arrays with keys **`"options"`** & **`"flags"`**

PHP Filters: Filtering a Variable

```
$i = 10;
$j = filter_var($i, FILTER_VALIDATE_INT);
if($j)
   echo "1- j = ". $j . "\n";
else
   echo "1- Data is not valid\n";

$fdata = array("options"=>array("min_range"=>15,
   "max_range"=>50));
$j = filter_var($i, FILTER_VALIDATE_INT, $fdata);
if($j)
   echo "2- j = ". $j . "\n";
else
   echo "2- Data is not valid\n";
```

PHP Filters: Filtering an Array of Variables

```
$data = array("int"=>10, "float"=>30.1);
$filter = array("int"=>array("filter"=>FILTER_VALIDATE_INT,
   "options"=>array("min_range"=>0)),
   "float"=>array("filter"=>FILTER_VALIDATE_FLOAT));
$valid = filter_var_array($data, $filter);
var_dump($valid);

$data = array("int"=>"a1z0", "float"=>30.1);
$valid = filter_var_array($data, $filter);
var_dump($valid);

$filter2 =
   array("int2"=>array("filter"=>FILTER_VALIDATE_INT,
   "options"=>array("min_range"=>0)),
   "float"=>array("filter"=>FILTER_VALIDATE_FLOAT));
$valid = filter_var_array($data, $filter2);
var_dump($valid);
```

Filtering Input Data

- Types:
  - **`INPUT_GET`**, **`INPUT_POST`**, **`INPUT_COOKIE`**, …
- To (optionally) apply a filter F on an input with name N with type T and get valid data

  `filter_input`(T, N, F)
- To (optionally) apply filter F on array of inputs with type T

  `filter_input_array`(T, F)
  - Output specified by the keys in the filter

Filtering Input Data Example

- Assume: URL:/filter.php?ip=192.168.0.1&address=http://www.abc.com

```
$valid_address = filter_input(INPUT_GET,
  "address", FILTER_VALIDATE_URL);
$filter =
  array("address"=>array("filter"=>FILTER_VAL
  IDATE_URL),
  "ip"=>array("filter"=>FILTER_VALIDATE_IP));
$valid_get = filter_input_array(INPUT_GET,
  $filter);
```

Extracting Valid Data

- Sanitize filters generate valid data from input
  - **FILTER_SANITIZE_EMAIL**
  - **FILTER_SANITIZE_NUMBER_FLOAT**
  - **FILTER_SANITIZE_NUMBER_INT**
  - **FILTER_SANITIZE_URL**
  - …

```
echo filter_var("a b c", FILTER_SANITIZE_ENCODED);
```

- a%20b%20c

```
echo filter_var("ab123ca", FILTER_SANITIZE_NUMBER_INT);
```

- 123

Implementing Custom Filter

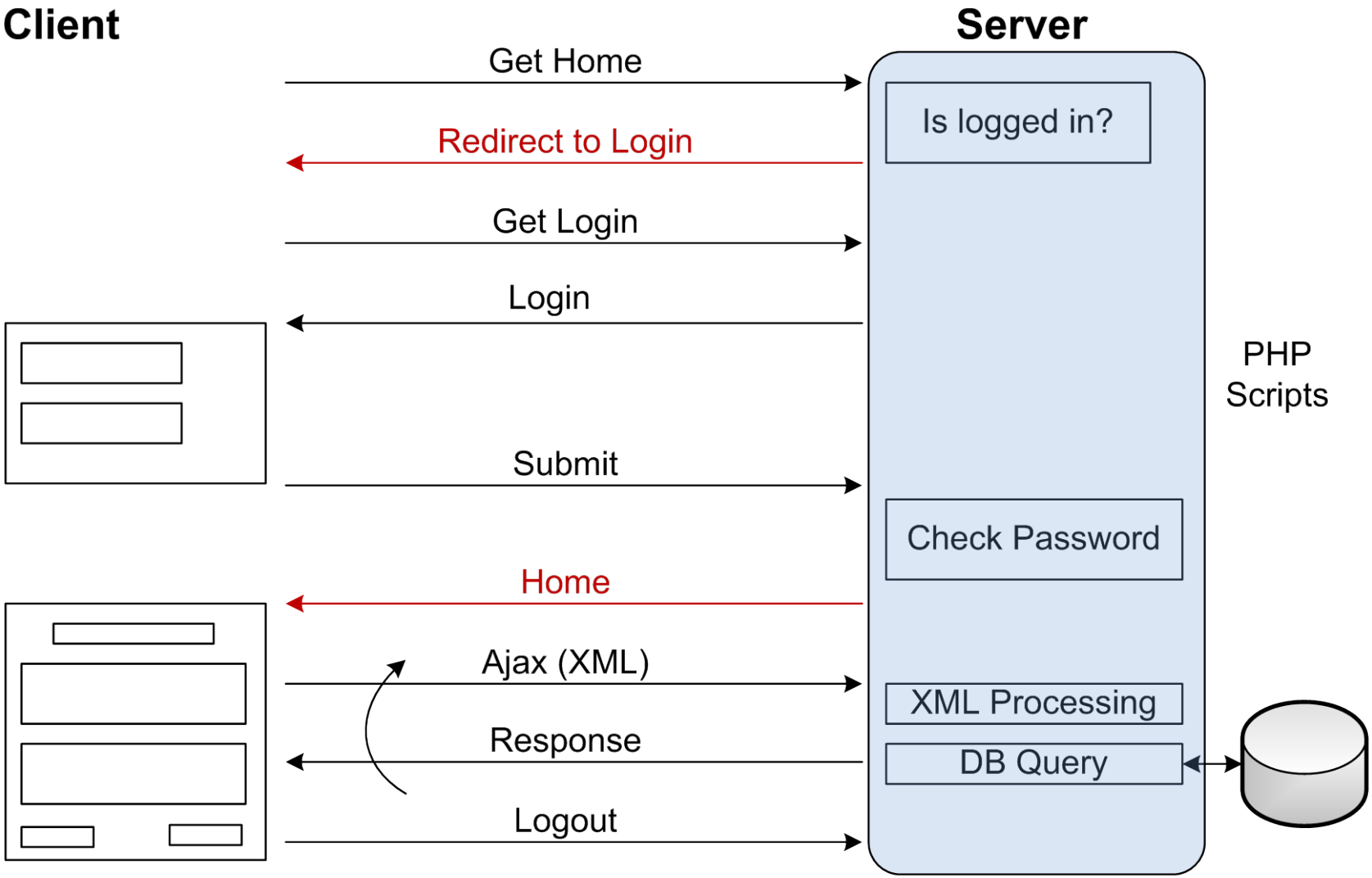
- Filter type **FILTER_CALLBACK** is used to register a custom filter

```
function convertSpace($string){
  return str_replace("_", " ", $string);
}
$string = "PHP_Scripting_is_fun!";
echo filter_var($string, FILTER_CALLBACK,
  array("options"=>"convertSpace"));
```

Outline

- Introduction to CGI
- Introduction to PHP
- PHP Basic
- Input Data Handling
- HTTP Headers
- Cookies & Session Management
- Database
- Error Handling
- XML

PHP in Web Applications



HTTP Headers

- Both HTTP *request* and *response* headers are accessible in PHP
  - PHP scripts can get HTTP request headers
  - PHP scripts can set HTTP response headers
- Request headers
  - Are extracted by php runtime environment
  - Filled in the **$_SERVER** superglobal array
    - `'REQUEST_METHOD', 'REQUEST_TIME', 'HTTP_ACCEPT', 'HTTP_ACCEPT_CHARSET', 'HTTP_ACCEPT_ENCODING', 'HTTP_CONNECTION', 'HTTP_REFERER', 'HTTP_USER_AGENT', ...`

HTTP Response Headers

- PHP scripts can modify HTTP response headers, to
  - Redirect the web client to another URL
  - Send a different HTTP status code
  - Tell client whether to cache the current document or not
  - Tell client what language is used in the current document
  - Change the content type of the current document
    - You can use PHP to dynamically create text file, CSV file, image, ...
- **`headers_list()`**: Return a list of headers to be sent to the web client
- **`header()`**: Set a raw HTTP header
  - Headers will be sent when actual output is generated

HTTP Response Headers Examples

- **`header()`** must be called before any actual output is sent!

- Redirecting

```
<?php
  header('Location: http://www.google.com/');
  exit(); // Return immediately
?>
```

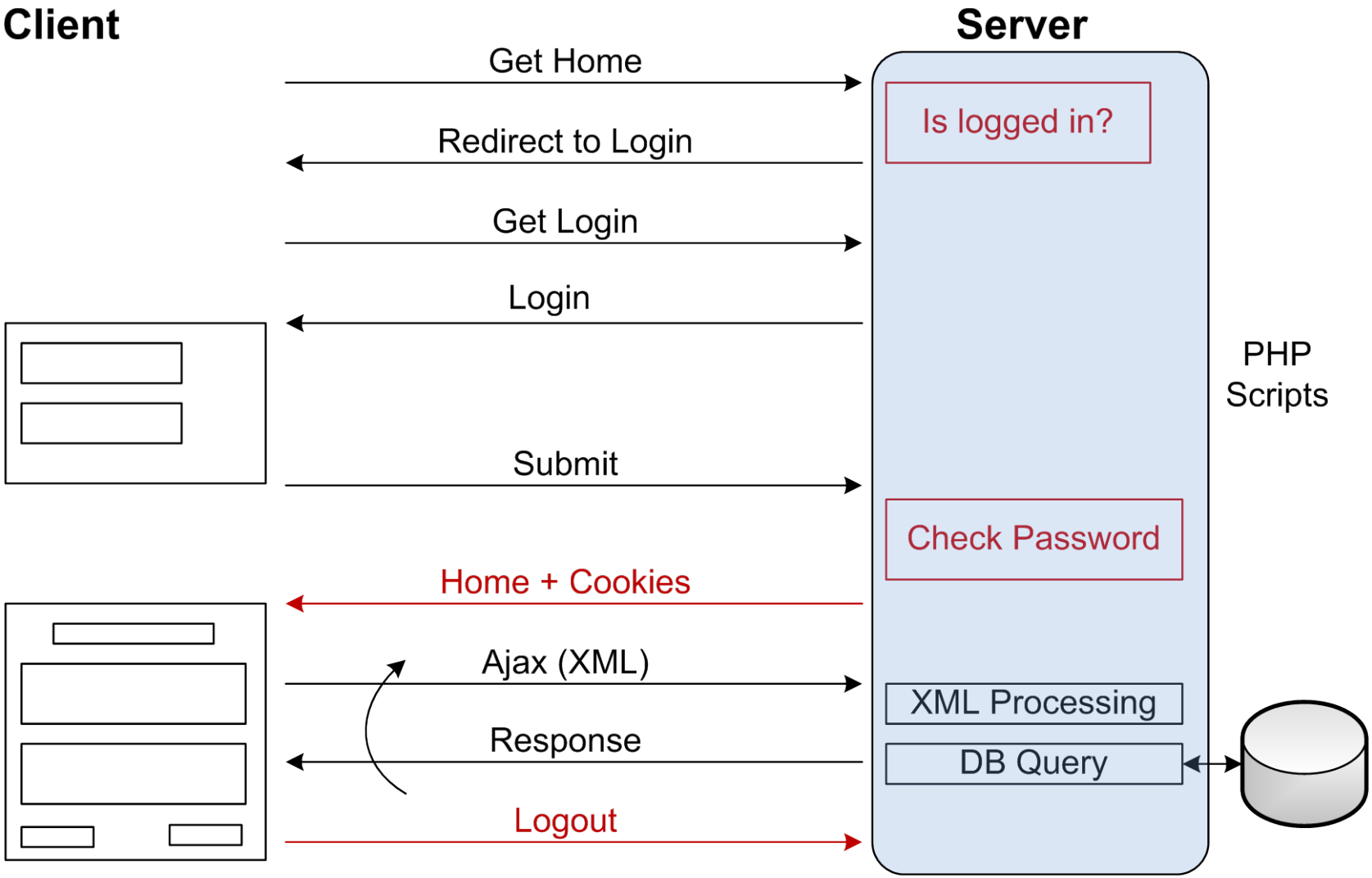
- Other Status Code

```
<?php
 header("HTTP/1.0 404 Not Found");
?>
<html> <!-- Content of the error page --> </html>
```

Outline

- Introduction to CGI
- Introduction to PHP
- PHP Basic
- Input Data Handling
- HTTP Headers
- Cookies & Session Management
- Database
- Error Handling
- XML

PHP in Web Applications



Main Questions

- Sample application: Student A want to check his course grades on the portal
- Q1: Is this client the “Student A”?
  - User authentication
- Q2: Are these requests from the client who is authenticated as the “Student A”?
  - User/Request identification
- Q3: How to store temporary data between login & logoff of the client (e.g., name, login state, …)
  - Session management

User Authentication

- Different mechanisms for authentication
  - HTTP authentication header
    - 1) Server built-in functionality for authentication
      - Proper server configuration
      - No server side scripting
    - 2) Server side scripting to use the HTTP headers
  - Pure HTML
    - Without HTTP Authentication headers
    - Using HTML forms + Server side scripting

HTTP Based Authentication 1

- HTTP support authentication: Basic & Digest modes
- Modern web servers have built-in support for HTTP authentication, Configure server
  - To set “WWW-Authentication” header
  - To check user/pass (using a password file)
- Apache
  - Generate password

```
htpasswd /var/www/site/password ali
```

  - Enable authentication for a directory using **`.htaccess`** file

```
AuthType Basic
AuthName "Main Site Login"
AuthUserFile /var/www/site/password
Require valid-user
```

HTTP Based Authentication 2

- PHP scripts have read/write access to HTTP headers
- At the first access, set “WWW-Authentication”
  - **`header()`** function
- In the following accesses to this directory or subdirectories check user/pass
  - Which are sent automatically by browser
  - Using “Authorization” header, e.g.
  - **`$_SERVER["PHP_AUTH_USER"]`**
  - **`$_SERVER["PHP_AUTH_PW"]`**

HTTP Basic Authentication in PHP

```
<?php
function prompt(){
    header('WWW-Authenticate: Basic realm="Protected Page"');
    header('HTTP/1.0 401 Unauthorized');
    die('You must enter a valid username & password');

}
function checkpass($name,$pass){
   if((strcmp($name,"abc") == 0) && (strcmp($pass,"123") == 0))
       return TRUE;
   else
       return FALSE;
}

if(!isset($_SERVER['PHP_AUTH_USER'])){
   prompt();
}
```

HTTP Basic Authentication in PHP

```
else{
   do{
        if(checkpass($_SERVER['PHP_AUTH_USER'],
                     $_SERVER['PHP_AUTH_PW']) == TRUE)
              break;
        prompt();
   }
   while(TRUE);
}
?>

<!DOCTYPE html>
<html>
<head></head>
<body>
<h2>You have singed in successfully</h2>
</body>
```

HTTP Authentication Solution

- How to authenticate the user
  - user/pass are asked by browser from user
- How to authenticate the subsequent requests
  - The “authorization” header is sent automatically by the browser for all request in this session
  - Server can check it to allow/deny access
- How to store temporary data
  - Cookies can be used
  - However, they are saved in client side (insecurity) and are sent in every request (insecurity + overhead)

HTTP Authentication Issues

- 1) The authentication window to get user/pass ☹
- 2) All pages that need authentication should be in the same directory (HTTP authentication works for directories)
- 3) How to logout
  - Authorization header is a session data (maintained by browser)
  - When is the data destroyed?
    - In modern tab based browsers → When the browser window is closed not when the tab is closed
      - Even if tab is closed & browser is not restarted → Authenticated
      - Security problem or a user friendly feature???
  - How to clear authorization data?
    - Is not easy!
    - Client side scripting (on page close) (trick & hack)

Solution for HTTP Authentication Issues

- Don't use the HTTP Authentication ;-)
- Instead
  - Get user/pass by a HTML form
  - Check user/pass in server side by PHP
    - If user/pass is not correct → Error & Redirect to login
    - If user/pass is correct → Show this page, then ???
- HTTP authentication mechanism ensure that the *subsequent* requests are from the *authenticated* user. How can we do it by PHP?
  - Cookies are the solution

Cookies for User Identification

- After successful authentication of user, set cookies to identify the authenticated user
  - **`Set-Cookies: login=true`**
  - **`Set-Cookies: Name=Ali Hassani`**
  - **`Set-Cookies: ID=11111`**
- In the following requests
  - If (login != true) → Error
  - Else
    - Say welcome to “Ali Hassani”
    - lookup DB for “11111” & show result

Cookies in PHP: Reading Cookies

- Access to cookies
  - Cookies are saved on client side
  - Sent back to server by browser
- Cookies are available in PHP using the **`$_COOKIE`** superglobal array
  - Key is the name of cookie
  - Value is the value (content) of the cookie
  - Check presence: **`isset($_COOKIE["key"])`**
  - Print all them recursively: **`print_r($_COOKIE)`**

Cookies in PHP: Setting Cookies

- Setting cookies

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

  - Name & Value are required
  - Expire, Path & Domain are optional
  - Must come before any output: i.e., before **`<html>`**

```
/* Permanent (up to 10 hours) cookie */
setcookie("id", "100", time()+36000);
/* Session cookie */
setcookie("name", "Ali");
/* Remove cookie */
setcookie("code", "", -1);
```

Cookies in PHP Example: register.php

```
<body>
<?php
   if(isset($_COOKIE["username"])){
        echo "I know you ". $_COOKIE["username"] ."!,
        You have registered ". $_COOKIE["regtime"] . "<br />";
        echo "<form method='get'
                action='http://127.0.0.1/IE/php/cookie.php'>
input type='submit' name='unregister' value='Unregister' />
        </form> ";
   }
   else{
        echo"<form method='get'
           action='http://127.0.0.1/IE/php/cookie.php'>
            Name: <input type='txt' name='name' />
            <input type='submit' value='Register' />
        </form> ";
   }
?>
</body>
```

Cookies in PHP Example: cookie.php

```
<?php
   $register = -1;
   $name = "";
   if(strlen($_GET["unregister"]) > 0){
        setcookie("username", "", -1);
        setcookie("regtime", "", -1);
        $register = 0;
   }
   elseif(strlen($_GET["name"]) > 0){
        $name = $_GET["name"];
        $expire = time()+30*24*60*60;
        setcookie("username", $name, $expire);
        setcookie("regtime", date("Y/m/d"), $expire);
        $register = 1;
   }
?>
<html>
<head>
</head>
```

Cookies in PHP Example: cookie.php

```
<body>
   <?php
        if($register == 1){
           echo $name."! Thank you <br />";
   ?>
        You have registered successfully for one month <br />
        You can check your <a href="register.php">registration</a>
   <?php
        }
        else if($register == 0){
           echo "You have unregistered successfully, Hope to see
   you again <br />";
           echo "Do you want to <a href='register.php'>register</a>
   again <br />";
        }
   ?>
</body>
</html>
```

Cookies in PHP: Controlling Cookies

- PHP script can set any Path & Domain for cookies
  - Browsers decide whether to accept or reject the cookies
- Major browsers
  - Domain names must start with dot
  - Don’t accept cookies for sub-domains
  - Accept cookies for higher domains
    - Except the top level domains, e.g., .com, .ac.ir
  - Accept cookies for other (sub or higher) paths

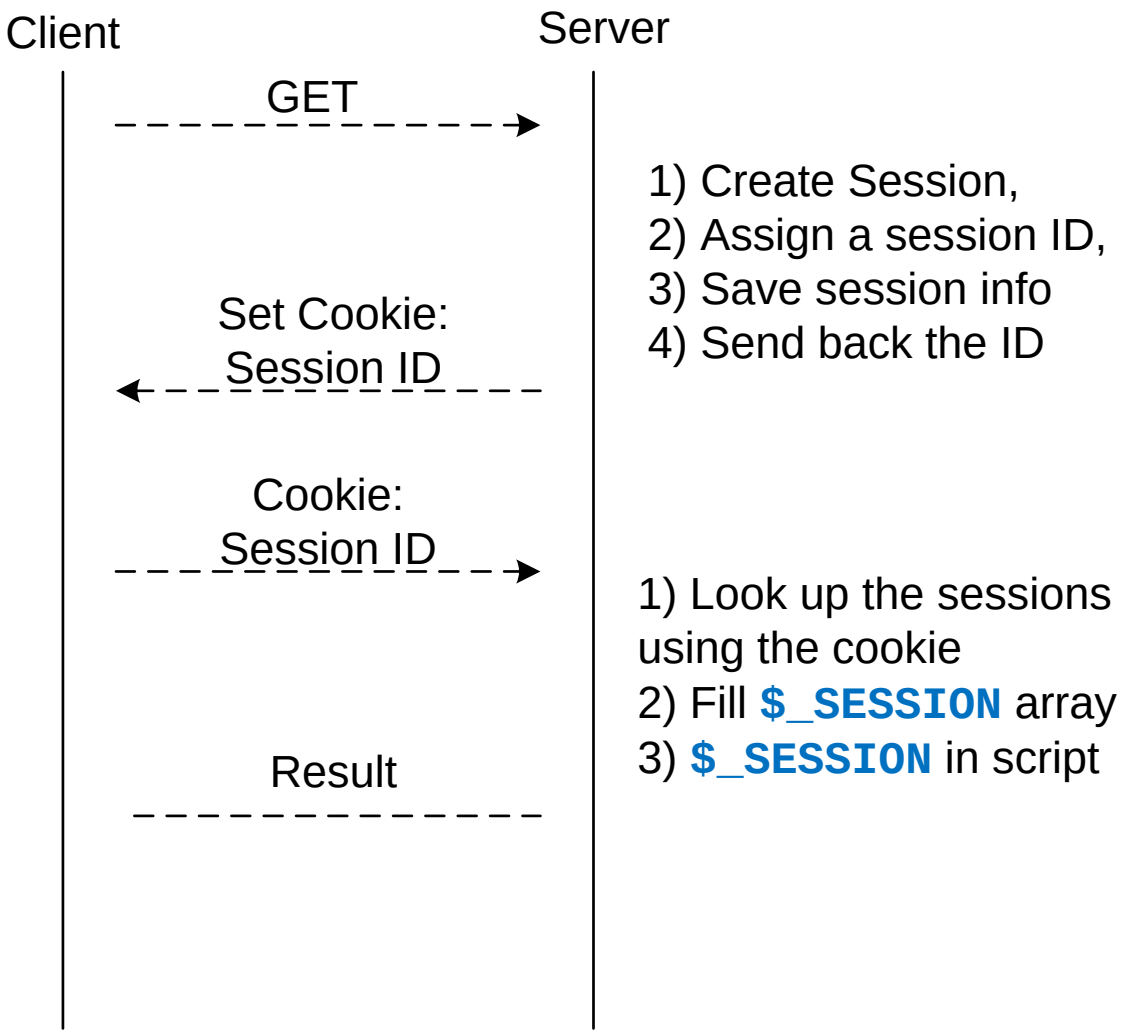
Cookies Issues

- Many applications need to save data/state for client in server side
  - E.g. login statue, current purchased items in e-shopping, name & ID of students, …
  - We do *NOT* want to use cookies, because of
    - Security, Overhead, Performance, …
- Solution
  - Create a (simple) database (i.e., *key-value* pairs) in server side (e.g., hash map, vector, …) for each *client*
  - Data should be associated to client
    - Server should identify clients
      - User cookies are the key of the database

PHP Sessions: Solution for Cookies Issues

- PHP Session: The Cookies + The (simple) Database
- A PHP session variable stores information about settings/states for a user session
  - Try to solve the "stateless HTTP" problem
  - By allowing to store user information on the server for later use
- Works by creating a unique id (ID) for each session and store variables based on this ID
  - ID is stored in a *session* cookie & sent back to client

PHP Sessions (cont'd)



PHP Sessions in Action

- Every page that uses session data must be proceeded by the **`session_start()`**
  - Creates new session or retrieves session info. from DB
    - How does it know what it should do?
  - Must come before anything sent to client, before **`<html>`**
- Session variables are then set and retrieved by accessing the global **`$_SESSION`**
- When we don't need the session data (e.g. logoff)
  - Remove an specific variable: **`unset($_SESSION["key"])`**
  - Delete whole session data: **`session_destroy()`**

PHP Sessions: Example 1

```
<?php #session1.php
session_start();
$_SESSION["cnt"] = (isset($_SESSION["cnt"])) ? $_SESSION["cnt"] +
   1 : 1;
?>
<html> <head> </head><body>
You have visited this page <?php echo $_SESSION["cnt"]; ?> times.
<form method="GET" action="http://127.0.0.1/IE/php/reset.php">
<input type="submit" name="reset" value="Reset" />
</form>
</body> </html>
================================================
<?php #reset.php
session_start();
unset($_SESSION["cnt"]); // session_destory()
?>
<html> <head></head> <body>Your counter is reset</body></html>
```

PHP Sessions: Example 2

- User authentication in all pages using PHP session
- 1) Create a login page
  - Create a session for user: **`session_start()`**
  - Get user/pass & check it
  - If it is valid user/pass → set a variable in session
    - **`$_SESSION["auth"] = true;`**
- 2) In all other pages
  - Access to the session: **`session_start()`**
  - Check authentication status:

`if($_SESSION["auth"])` Okay, ….
`else` error & redirect to login

PHP Sessions: Example 2 (cont'd)

- login.php
  - Check user/pass
  - Setup session
  - Redirect to home.php
- home.php
  - Check authentication
  - Logout using logout.php
- logout.php
  - Destroy session

Session vs. Cookies

- Sessions use cookies
  - The “Session ID” cookie
- However, data is saved in server side (is not sent to client)
  - Less overhead on client
  - Less bandwidth for data transmission
  - More secure
    - Client (and also hackers) does know what you are saving
    - Harder to hijack the session, compare
      - `setcookie("login", 1);`
      - `$_SESSION[login] = 1;`

When does a PHP Session Expire?

- PHP Session is a relation between
  - Session ID Cookie in Client side
  - Session Data Base in Server side
- So, session is not accessible
  - Session ID cookie is destroyed
    - Browser restarts, delete cookie, …
  - Session is destroyed
    - Intentionally: logout, automatic (ajax based) logout, …
    - Accidently:
      - Server restart
      - Long inactive session are collected by garbage collector
        - Avoiding over utilizing server memory

Session Parameters: Global Settings

- Session related parameters are configured in "php.ini"
- **`session.name`**: Name of the session (used as cookie name)
- **`session.cookie_lifetime`**: Lifetime of cookie send for browser
- **`session.cookie_path`**: The path for which the cookie is valid
- **`session.gc_maxlifetime`**: Lifetime of cookie in server side, it is collected by GC

Session Parameters: Per Script

```
void session_set_cookie_params(int
  $lifetime, string $path, string $domain,
  bool $secure=false, bool $httponly=false)
```

- The effect of this function only lasts for the duration of the script. Thus, you need to call this function for every request and before `session_start()` is called
- Default value of `$path` is `'/'`. To prevent session ID from being discovered by other PHP scripts running in the same domain, you should set `$path` to the subfolder where your scripts are stored

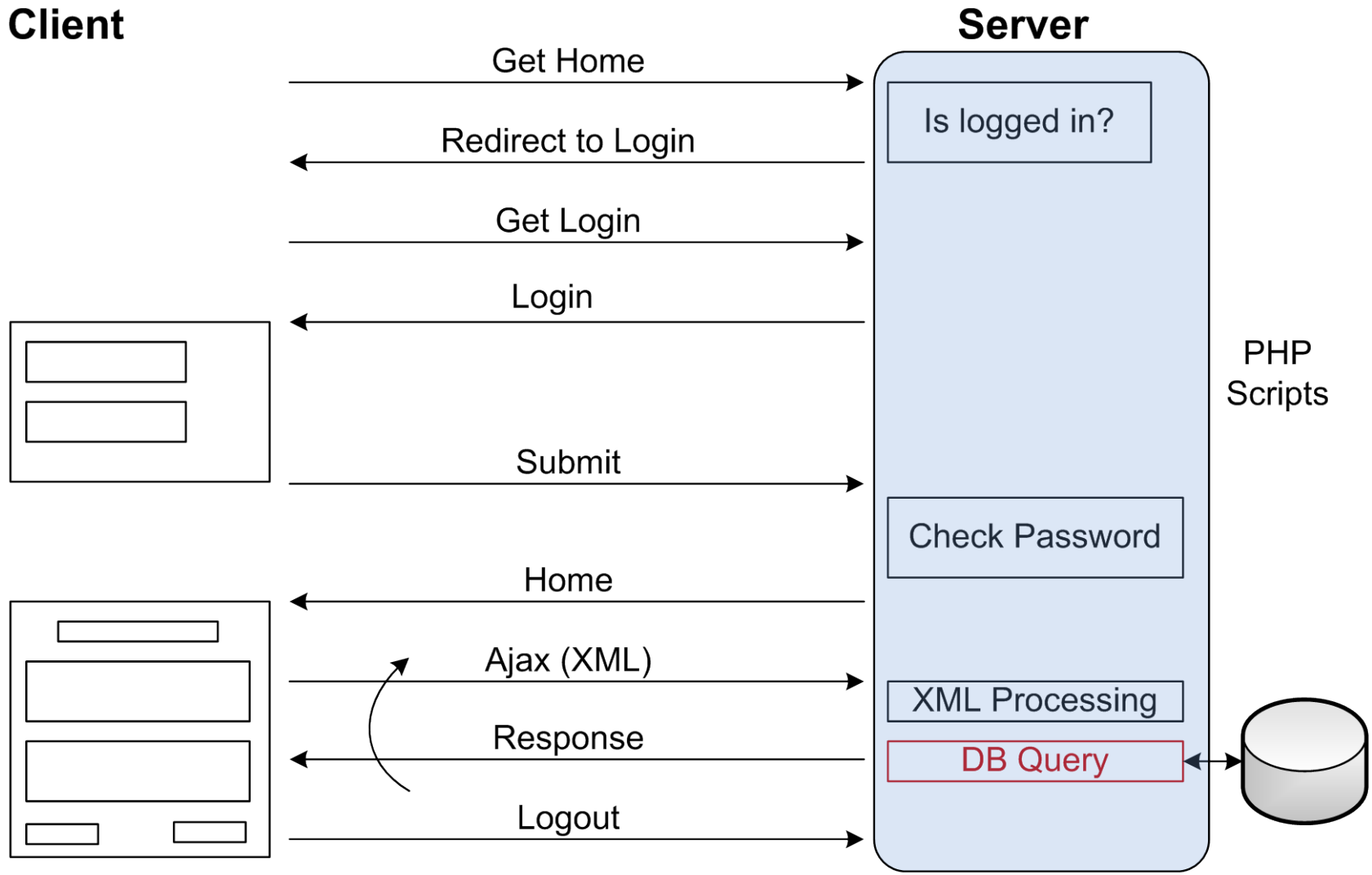
HTML Based Authentication + PHP Sessions Advantages

- Easily store data for each client in server side
- No window for authentication, everything in HTML
- Session ID cookie can be configured (set path & domain) to be shared between multiple directories & domains
- Safe logout, similar to “HTTP Authentication” user/pass, session cookies are not expired at when the tab is closed, but!
  - Instead of trying to remove session data on browser (client side), invalidate it on server, How?
    - Ask server to destroy its corresponding session
      - A logout button/link
      - Create an Ajax request when window is closed
        - Automatic logoff

Outline

- Introduction to CGI
- Introduction to PHP
- PHP Basic
- Input Data Handling
- HTTP Headers
- Cookies & Session Management
- Database
- Error Handling
- XML

PHP in Web Applications



Databases in PHP

- Many databases; here, MySQL
- Fast review of databases' basics
  - Database, Table, Row, …
  - Database creation
  - Database modification
  - Database query
- MySQL in PHP
  - Database connection
  - Modification & Query
  - SQL injection

Database Basic Concept

- Relational database
  - Database server contains multiple databases
  - Each database consists of multiple tables
  - Each table is defined by its columns (& types)
  - Each row is a data record
  - A column is the primary key
    - A unique identifier for each record
- We use Structured Query Language (SQL) for database management
  - A famous SQL based database → MySQL
    - Free, Open source, and multiplatform

SQL Commands: Create Table

```
CREATE TABLE students(
  name VARCHAR(55), num INT(3),
  grade DECIMAL(2,2),
  primary key(num)
);
```

- Types
  - **TEXT**, **CHAR**(size), **VARCHAR**(maxsize), **INT**(maxsize), **DECIMAL**(maxsize, precision) , **DATE**(), **YEAR**(),....
- For primary key
  - `id INT AUTO_INCREMENT, primary key(id)`

SQL Commands (cont'd)

- Inserting data
  - **INSERT INTO** tabelname (column1, …)
    VALUES (val1, …);
    - **INSERT INTO students(name,grade,num)**
      VALUES ("Ali", 15.23, 1122);
- Querying data
  - **SELECT** columnname **FROM** table **WHERE** condition
    - **SELECT** * **FROM** students **WHERE** grade=20
  - Conditions by comparison & logical
    - **=, !=, <, <=, >, >=, …**
    - **AND, OR**

SQL Commands (cont'd)

- Updating records
  - **UPDATE** tablename **SET** col1=val1, col2=val2, … **WHERE** condition
    - **UPDATE** student **SET** grade=20 **WHERE** name='Ali';
- Deleting a record from a table
  - **DELETE FROM** tablename **WHERE** condition;
    - E.g. clear the students table

      DELETE FROM students;
- Deleting a table
  - **DROP TABLE** tablename;

Real Example

MySQL in PHP

- There are two interfaces (API) to access MySQL in
  - The Old API
    - Functions start by **`mysql_`**
    - Now deprecated, will be removed
      - However, very popular, lot of web applications based on
  - The New Improved Extension
    - Available in two modes
      - Procedural mode: functions start by **`mysqli_`**
        - Very similar to the old API, with minor differences & new features
      - Object oriented mode
        - The same functions but as a method of objects

MySQL in PHP (cont'd)

- In general, all APIs follow the same concept to work with MySQL DB
  - Functions & Parameters are different
- The steps of the follow
  - Connect to the database server
  - Select the database in the server
  - Send SQL queries to the tables of the database
  - Process the result (typically as an array)
  - Close the connection

MySQL in PHP: Connecting & Selecting

- The first step to work with MySQL
  - 1) Connecting to the MySQL server
  - 2) Selecting database
  - Required for all operations on database

```
$mysqli = mysqli_connect("server address",
  "username","password", "DB name") or
  die(mysqli_connect_error());
```

  - We don’t want to continue if it fails

MySQL in PHP: SQL Commands

- SQL commands are send by
  `mysqli_query($mysqli,"SQL Command")`
  - Syntax is the SQL
- E.g., Create table in the selected database

```
mysqli_query($mysqli, "CREATE TABLE
  students(
  id INT AUTO_INCREMENT,
  primary key(id),
  name VARCHAR(50),
  stdnum INT(8))");
```

MySQL in PHP: Query & Closing

- Query result is processed by **mysqli_fetch_***
  - E.g., **mysqli_fetch_assoc()**

```
$result = mysqli_query($db, "SELECT ...");
while($row = mysqli_fetch_assoc($result)){
  $std_name = $row['name'];
  $std_grade = $row['grade'];
}
mysqli_free_result($result);
```

- Close database connection:

```
mysqli_close($mysqli)
```

Example

- Database: students
- Table: IE
  - (name, fam, grade, num)
- datainput.html: HTML form to insert data
- dbinsert.php: Insert data to DB
- datasearch.html: HTML form to query
- dbsearch.php: Run the query and show result

Example: datainput.html

```
<html>
<head>
</head>
<body>
  <form action="http://127.0.0.1/IE/php/dbinsert.php"
  method="GET">
      Name: <input type="text" name="n" /><br />
      Family: <input type="text" name="f" /><br />
      Std #: <input type="text" name="i" /><br />
      Grade: <input type="text" name="g" /><br />
      <input type="submit" value="Insert Data" />
  </form>
</body>
</html>
```

Example: dbinsert.php

```
<?php
$name = $_REQUEST["n"]; $famanme = $_REQUEST["f"];
$grade = $_REQUEST["g"]; $num = $_REQUEST["i"];

if((strlen($num) > 0) && (strlen($famanme) > 0) && (strlen($grade)
   > 0) && (strlen($num) > 0)){
   $db = mysqli_connect("127.0.0.1", "root", "12345678",
   "students") or die(mysqli_connect_error());

   $result = mysqli_query($db, "INSERT INTO IE(name, fam, num,
   grade) VALUES('$name', '$famanme', '$num', '$grade');") or
   die(mysqli_error($db));

   mysqli_close($db);
   echo "Data has been inserted successfully <br />";
}
else{
   echo "Wrong Input";
}
?>
```

Example: datasearch.html

```
<html>
<head>
</head>
<body>
   <form action="http://127.0.0.1/IE/php/dbsearch.php"
   method="GET">
   Parameter:
        <select name="col">
               <option value="name">Name</option>
               <option value="fam">Family</option>
               <option value="grade">Grade</option>
               <option value="num">Student #</option>
        </select>
        <input type="text" name="query" /> <br />
        <input type="submit" value="Search" />
   </form>
</body>
</html>
```

Example: dbsearch.php

```
<?php
$column = $_REQUEST["col"]; $value = $_REQUEST["query"];

if((strlen($column) > 0) && (strlen($value) > 0)){
   $db = mysqli_connect("127.0.0.1", "root", "12345678", "students") or
   die(mysqli_connect_error());

   $result = mysqli_query($db, "SELECT name,fam,num,grade FROM IE WHERE
   $column='$value' ORDER BY grade DESC") or die(mysqli_error($db));

   while($row = mysqli_fetch_assoc($result))
        echo "Name: ", $row["name"], ", Family: ", $row["fam"], ", Std #:
   ", $row["num"], ", Grade: ", $row["grade"], "<br />";

   mysqli_free_result($result);
   mysqli_close($db);
}
else{
   echo "Wrong Input";
}
?>
```

SQL Injection

- Technique that malicious user (attacker) can inject (unexpected & harmful) SQL commands into SQL statements via web pages
  - Compromise user security
    - Get confidential information
  - Compromise web application integrity
    - Alert the database
- One of the most common approach to attack web applications

SQL Injection by Example

- Two tables in “injection” DB
  - account (id, pass, name, balance)
  - private (c1, c2)
- Three forms to search the DB
  - Only ID based
  - ID & Pass
  - Multiple IDs
- Two PHP scripts
  - Single query for form #1 & #2
  - Multi query for form #3

SQL Injection by Example (cont'd)

- SQL statement in single query script

```
$query = 'SELECT * FROM account WHERE
id='.$user_id;
```

Or

```
$query = 'SELECT * FROM account WHERE
id='.$user_id.' and pass="'.$password.'"';
```

- SQL statement in multi query script

```
$query = 'SELECT * FROM account WHERE
id='.$user_id1.';';

$query .= 'SELECT * FROM account WHERE
id='.$user_id2.';';
```

SQL Injection by Example (cont'd)

- Inputs by normal users
  - ID = 1111
  - ID = 2222 pass = pass2
  - ID1 = 1111 ID2 = 2222
- Malicious user
  - ID = 1111 or 1=1
  - ID = 1111 or ""="" pass = pass1" or ""="
  - ID1 = 1111 ID2 = 2222; DROP TABLE private;

:-OoOoo!!!! Why?!

Preventing SQL Injection (besides filters)

- Parameterized queries by *preparing* statements
  - Preparing stage
    - Statement *template* is sent to server
      - Server checks syntax & initialize internal resources for execution
  - Variable binding & Execution stage
    - *Value* of variables are sent
    - Server creates statement from the template & the bounded variables & executes it
- Designed for performance improvement to run same statement repeatedly with high efficiency
  - Can be used as appropriate solution for SQL injection?!
    - Why?!!

Preventing SQL Injection (cont'd)

- 1) Preparing template (statement)

```
$stmt = mysqli_prepare($db, "SELECT id, pass,
name, balance FROM account WHERE id=?");
```

- 2) Binding variables

```
mysqli_stmt_bind_param($stmt, "i", $user_id);
```

- 3) Executing the statement

```
mysqli_stmt_execute($stmt);
```

Preventing SQL Injection (cont'd)

- 4) Binding the results
  - A variable per column in the output

```
mysqli_stmt_bind_result($stmt, $ids, $passs,
$names, $balances);
```

- 5) Fetching output from the result

```
while (mysqli_stmt_fetch($stmt)) {
      printf("ID: %s,  Password: %s,  Name:
      %s, Balance: %s\n", $ids, $passs,
      $names, $balances);
}
```

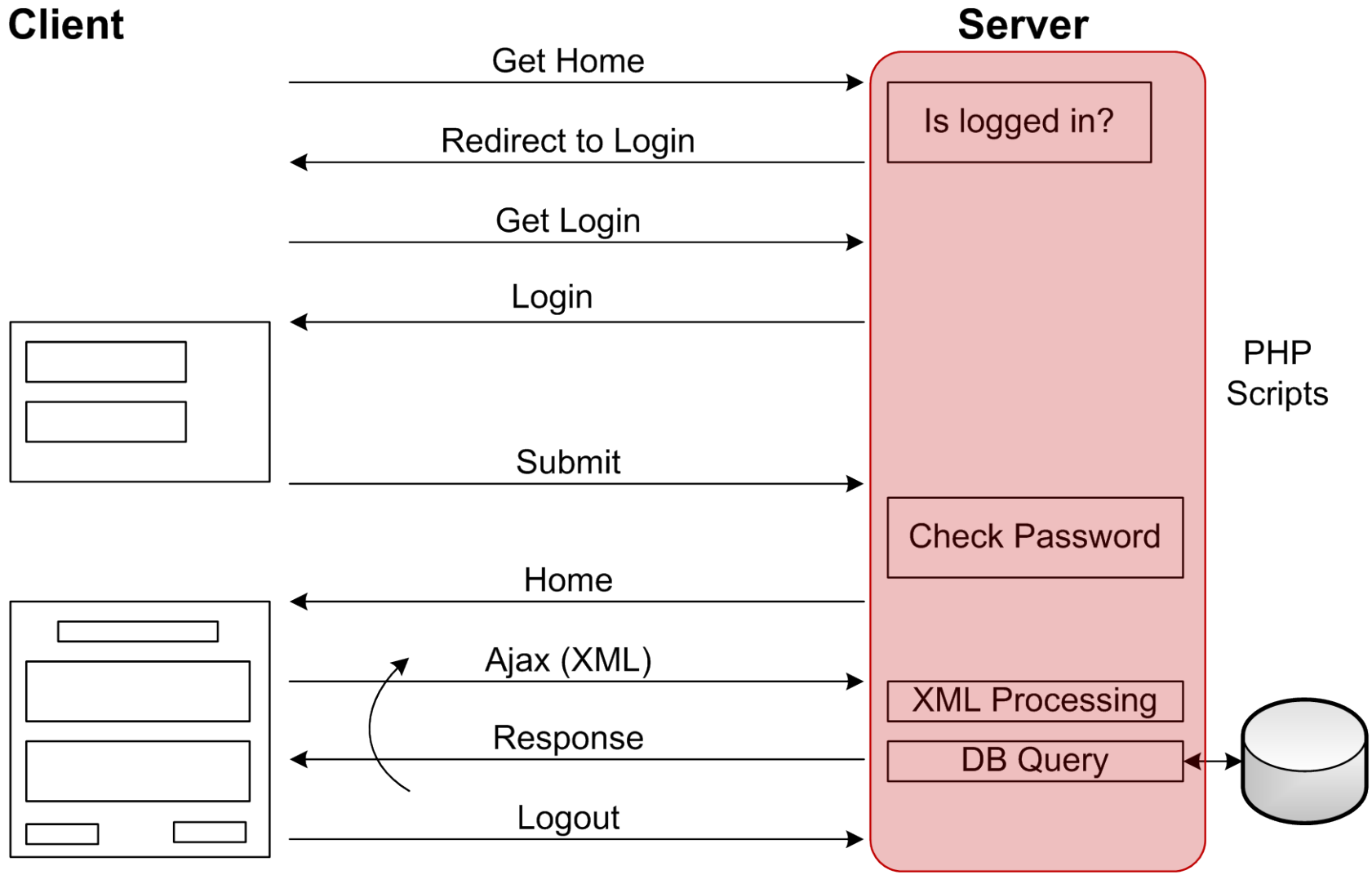
Preventing SQL Injection Example

- The safe version of the PHP scripts

Outline

- Introduction to CGI
- Introduction to PHP
- PHP Basic
- Input Data Handling
- HTTP Headers
- Cookies & Session Management
- Database
- Error Handling & Debugging
- XML

PHP in Web Applications



Error Handling

- Error handling is very important in PHP
  - HTTP server is the open door of the server/network
    - HTTP server by itself is (almost) secure
    - HTTP server runs PHP codes
  - Inputs come from Internet (Hackers)
  - A security hole in PHP → Access to server
- Simple default error handling in PHP
  - An error message with filename, line number and a message describing the error which is configured in **`php.ini`**
    - Displayed: client can see, should not used in final release
    - Log: server log, useful for debugging & server monitoring
    - …

Error Handling (cont'd)

- **`die()`** function to stop running the script

```
$file = fopen("data.txt", "r");
if($file == null)
   die("Cannot open file");
```

  - Or

```
$file = fopen("data.txt", "r") or
die("Cannot open file");
```

- Surpassing errors by @ operator
  - Don't send error messages to client (security)

```
$x=10; $y=0; $z=@($x/$y);
```

Custom Error Handling

- A special function is called when an error occurs

```
error_function(error_level, error_message,
  error_file, error_line, error_context)
```

- error_level: Required, specifies the error report level for the user-defined error.
- error_message: Required, specifies the error message for the user-defined error
- error_file: Optional, specifies filename in which the error occurred
- error_line: Optional, specifies line number in which the error occurred
- error_context: Optional, specifies an array containing every variable, and their values, in use when the error occurred

Custom Error Handling (cont'd)

- Custom error handler registration

```
set_error_handler("functionName", Level);
```

- Level
  - If omitted → all levels of errors
  - 1 (E_ERROR): Fatal run-time errors
  - 2 (E_WARNING): Non-fatal run-time errors
  - 8 (E_NOTICE): Run-time notices
  - 256 (E_USER_ERROR): Fatal user-generated error
  - 512 (E_USER_WARNING): Non-fatal user-generated warning
  - …

Error Handling (cont'd)

- In addition to system generated errors, user can trigger (generate) errors

```
trigger_error("Error Message", Level);
```

  - Message is a required custom message
  - Level is optionally specifies error level
    - E_USER_ERROR
    - E_USER_WARNING
    - E_USER_NOTICE
    - If omitted → E_USER_NOTICE

Error Handling Example

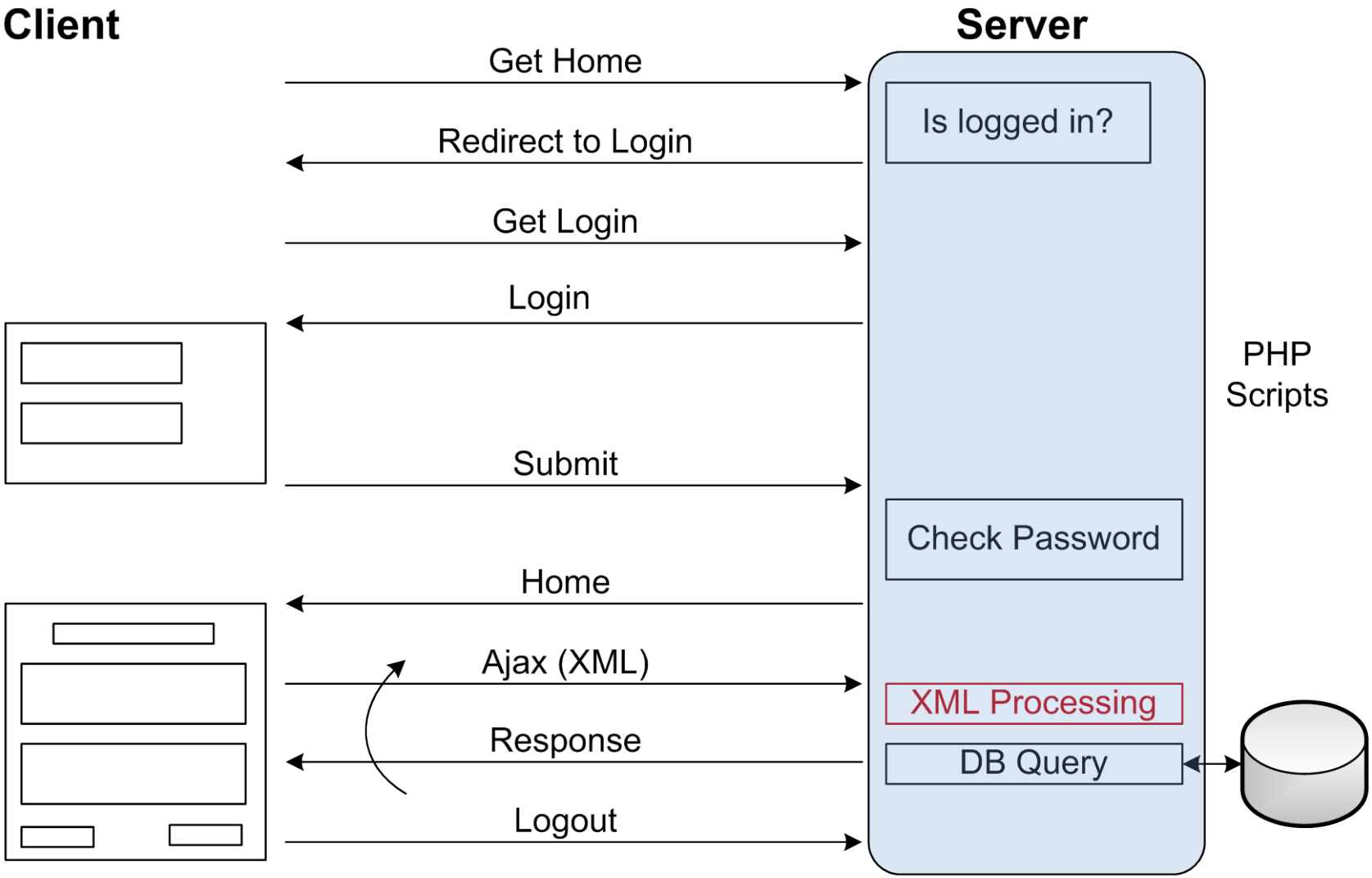
```
function myErrorHandler($errno, $errstr, $errfile, $errline){
    switch ($errno) {
    case E_USER_ERROR:
        echo "<b>My ERROR</b> [$errno] $errstr<br />\n";
        echo "Fatal error on line $errline in file $errfile\n";
         exit(1);
    case E_USER_WARNING:
        echo "<b>My WARNING</b> [$errno] $errstr<br />\n"; break;
   default:
         echo "<b>Some other errors</b>";
    }
   return true; // don't run internal error handler
}

set_error_handler("myErrorHandler");
$fd = fopen("data.bin", "r");
if($fd == null)
   trigger_error("Cannot open file", E_USER_ERROR);
if(filesize("data.bin") == 0)
   trigger_error("Data file is empty", E_USER_WARNING);
fclose($fd);
$db=mysqli_connect("127.0.0.1", "root", "wrongpass", "injection");
```

Outline

- Introduction to CGI
- Introduction to PHP
- PHP Basic
- Input Data Handling
- HTTP Headers
- Cookies & Session Management
- Database
- Error Handling
- XML

PHP in Web Applications



XML in PHP

- Extensive XML support in PHP
  - Different libraries to read/parse/write XML
- Create XML
  - Print XML tags manually
  - XMLWriter: To create XML files easily
- Parse XML
  - DOM library: Access to XML tree structure
  - Expat: An event based parser
- XSLT: XML transformation on server side

XML in PHP (cont'd)

- There are two basic types of XML parsers:
- DOM(Tree)-based parser:
  - XML is transformed into a tree structure
  - Whole document is analyzed to create tree
  - Easy to use by getElement… functions; but is not useable for large files & stream
- Event-based parser:
  - Focus on XML content, not their structure
  - XML document is interpreted as a series of events
    - When a specific event occurs, a function is called to handle it
  - More programming effort, but useable for stream & less memory

XML in PHP (cont'd)

- Expat is a event-based XML parser in PHP
- Step 1: Initialize the XML parser

```
$parser = xml_parser_create();
```

- Step 2: Function declaration
  - Function to be used at the start of an element

```
function mystart($parser, $element_name,
  $element_attrs)
```

  - Function to be used at the end of an element

```
function mystop($parser, $element_name)
```

  - Function to be used when finding character data

```
function mychar($parser, $data)
```

XML in PHP (cont'd)

- Step 3: Function registration

```
xml_set_element_handler($parser, "mystart",
  "mystop");
xml_set_character_data_handler($parser,
  "mychar");
```

- Step 4: Parsing document

```
xml_parse($parser, $data, $data_end_flag)
```

- Step 5: Finish

```
xml_parser_free($parser);
```

Example

```
<?php
function mystart($parser, $element_name,
  $attr){
  echo "Start: ". $element_name ."\n";
}

function myend($parser, $element_name){
  echo "End: ". $element_name ."\n";
}

function my_char_data($parser, $d){
  echo "Char: ". $d ."\n";
}
```

Example

```
$parser = xml_parser_create();
xml_set_element_handler($parser, "mystart",
  "myend");
xml_set_character_data_handler($parser,
  "my_char_data");

$data="<root><book><name>1</name><price>1000</p
  rice></book></root>";

xml_parse($parser, $data, TRUE);
xml_parser_free($parser);
?>
```

Outline

- Introduction to CGI
- Introduction to PHP
- PHP Basic
- Input Data Handling
- HTTP Headers
- Cookies & Session Management
- Database
- Error Handling
- XML

Answers

- Q7.1) How to code in server side?
  - Use CGI or Embed code, the later is preferred!!
- Q7.2) Which language? Syntax?
  - PHP, it is very similar to C/Java, but it is interpreted language
- Q7.3) How can I get valid user's data in server?
  - Super global arrays: **$_GET**, **$_POST**, **$_REQUEST**, …
  - Validation mechanisms: Validating & Sanitizing
- Q7.4) Can I read/write access to HTTP headers
  - Yes, **header( )** to write, **$_SERVER** to read
- Q7.5) The users must login to access the site!
  - Okay, use PHP session + Authentication
- Q7.6) Can I use data base? How?
  - Yes, depends on you DB, MySQL is easy to use!!

What are the Next?!

- OOP PHP
  - This was a major change from PHP 4. PHP 5 has a full object model.
    - Class, Object, Methods, Properties, public, private, …
- PHP Frameworks
  - Laravel
  - Phalcon
  - Symfony 2
  - Zend
  - CodeIgniter

References

- Reading Assignment: Chapter 9 of "Programming the World Wide Web"
- PHP Manual from www.php.net
- http://www.w3school.com/php
- Matt Zandstra, "Sams Teach Yourself PHP in 24 Hours"