Server-side programming : PHP

Web Programming 2021-2022

PHP – What it is

- PHP is a server-side programming language
  - Open source, freely available
  - Powers some of the major websites in the world
    - Wikipedia
    - Facebook
    - Apple
    - ...
  - Many other options: Python, Perl, C, Java, JavaScript, ...
- Stands for PHP Hypertext Processor (recursive acronym)
- PHP in itself is a complete programming language
  - You can create any program in PHP (just like C, Java, ...)
  - However, PHP is mainly targeted for use in web programming
  - Beware: PHP is constantly evolving, so code will need to be refactored from time to time

PHP – Server side programming

- Important uses of PHP
  - Database access
  - Form validation (server-side)
  - Mailing lists
  - On-the-fly content creation (e.g., image processing)
  - Photo galleries…
  - …

- Implications
  - As the name says: server-side programming languages are used on the server
  - This means that PHP code cannot be interpreted by the web browser
  - How does it work then?
    - Let's talk about HTTP first…

How are web pages transmitted over the Internet ?

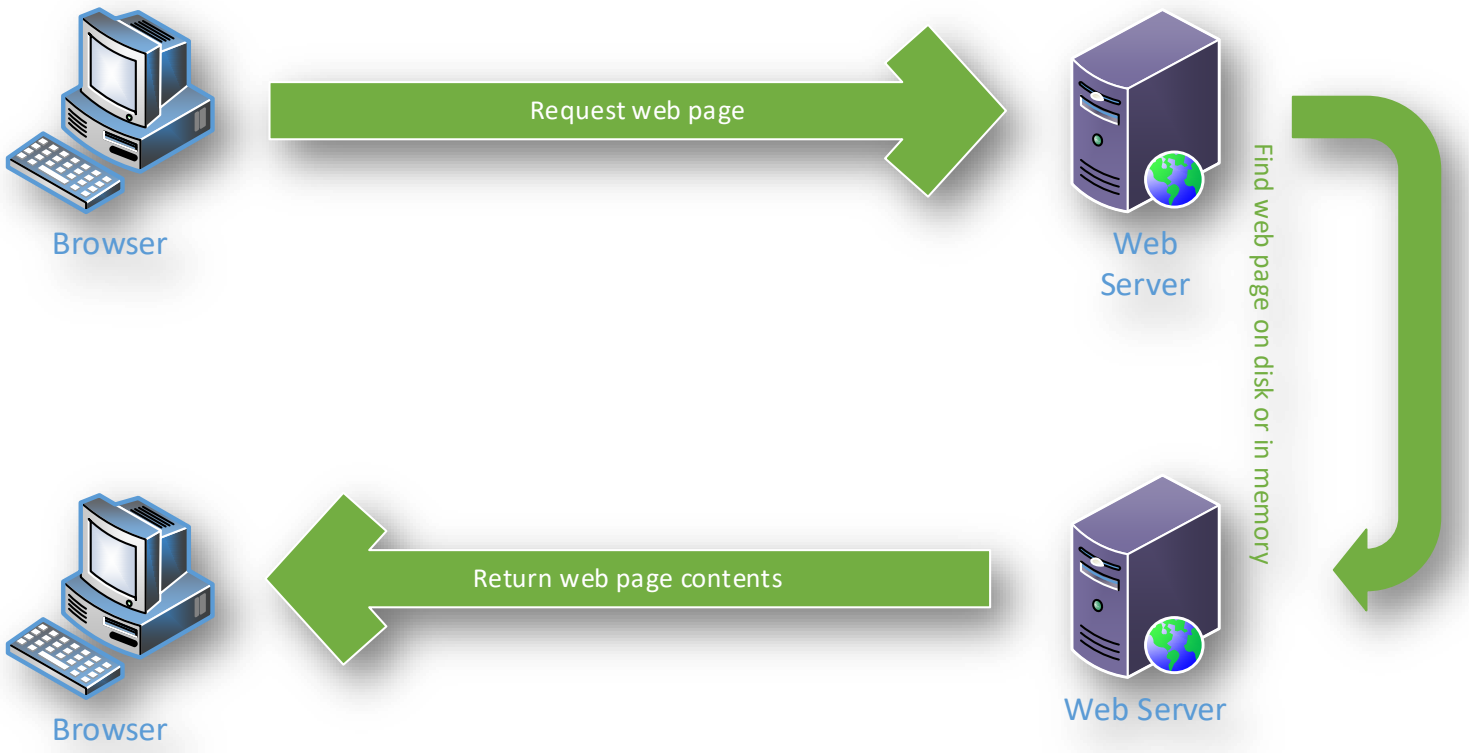
- Using the HyperText Transfer Protocol (HTTP)

- This protocol is:
  - Ancient: has been around since the beginning of the WWW, evolved only minimally
  - Text-based: human-readable for a large part
  - All-around: “everything” is being transmitted over HTTP these days, even video (YouTube etc)
  - Based on a request/reply system: synchronous = after a request, the system is on hold waiting for a reply

- Basic idea:
  - Client (web browser) issues an HTTP GET request
  - Server looks whether the resource is available
  - Resource is read from file or memory
  - HTTP response is returned containing the resource

Basic operation of HTTP

- Visualized



How are web pages transmitted over the Internet ?

Example

When you type a url in your address bar, your browser sends an HTTP request and it may look like this:

```
GET /tutorials/other/top-20-mysql-best-practices/ HTTP/1.1
Host: net.tutsplus.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.1.
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Cookie: PHPSESSID=r2t5uvjq435r4q7ib3vtdjq120
Pragma: no-cache
Cache-Control: no-cache
```

First line is the "Request Line" which contains some basic info on the request. And the rest are the HTTP headers.

After that request, your browser receives an HTTP response

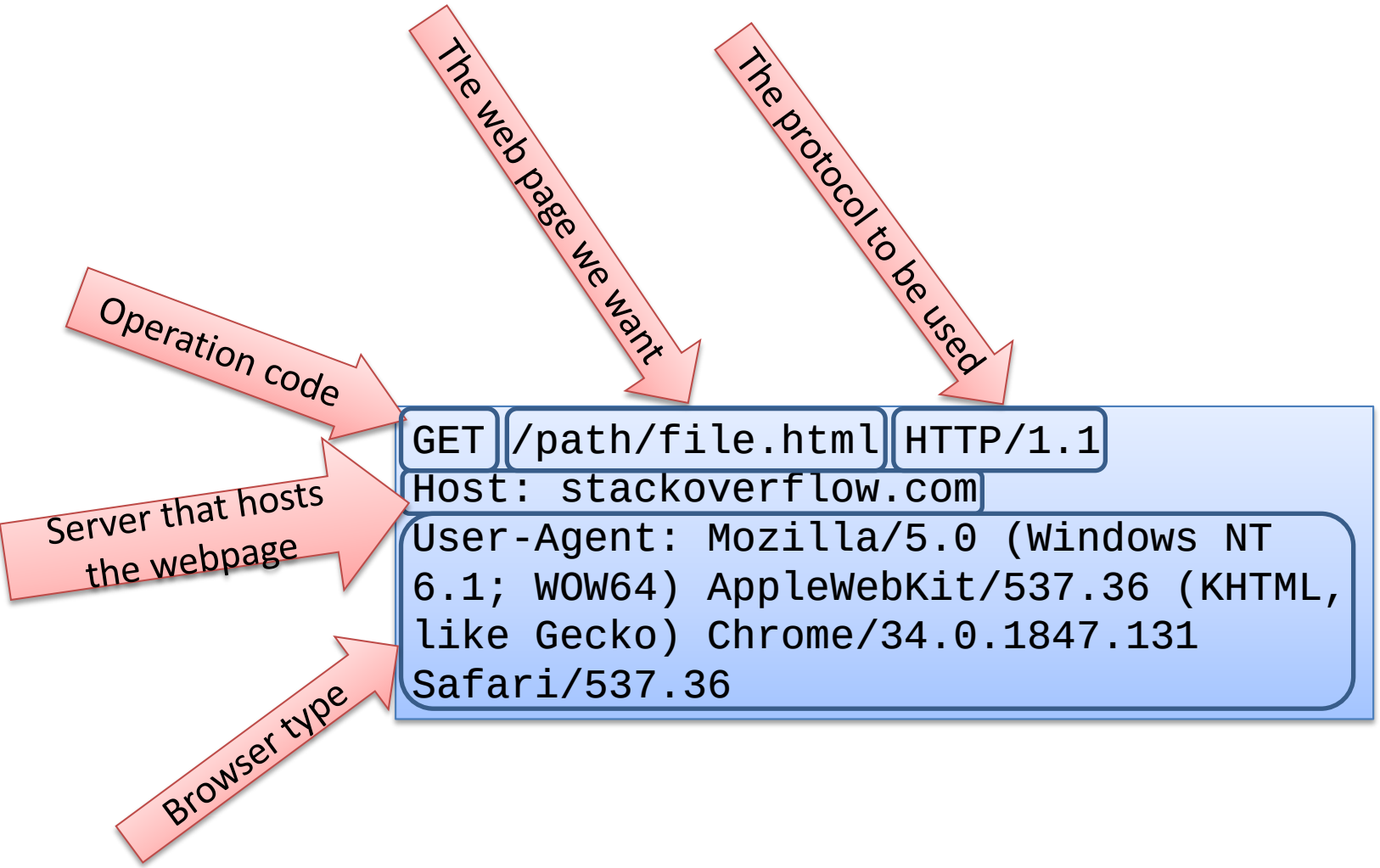
```
HTTP/1.x 200 OK
Transfer-Encoding: chunked
Date: Sat, 28 Nov 2009 04:36:25 GMT
Server: LiteSpeed
Connection: close
X-Powered-By: W3 Total Cache/0.8
Pragma: public
Expires: Sat, 28 Nov 2009 05:36:25 GMT
Etag: "pub1259380237;gz"
Cache-Control: max-age=3600, public
Content-Type: text/html; charset=UTF-8
Last-Modified: Sat, 28 Nov 2009 03:50:37 GMT
X-Pingback: http://net.tutsplus.com/xmlrpc.php
Content-Encoding: gzip
Vary: Accept-Encoding, Cookie, User-Agent

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Stric
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html
<title>Top 20+ MySQL Best Practices - Nettuts+</ti
<!-- ... rest of the html ... -->
```

The first line is the "Status Line", followed by "HTTP headers". After that, the "content" starts (in this case, an HTML output).

More info: http://code.tutsplus.com/tutorials/http-headers-for-dummies--net-8039

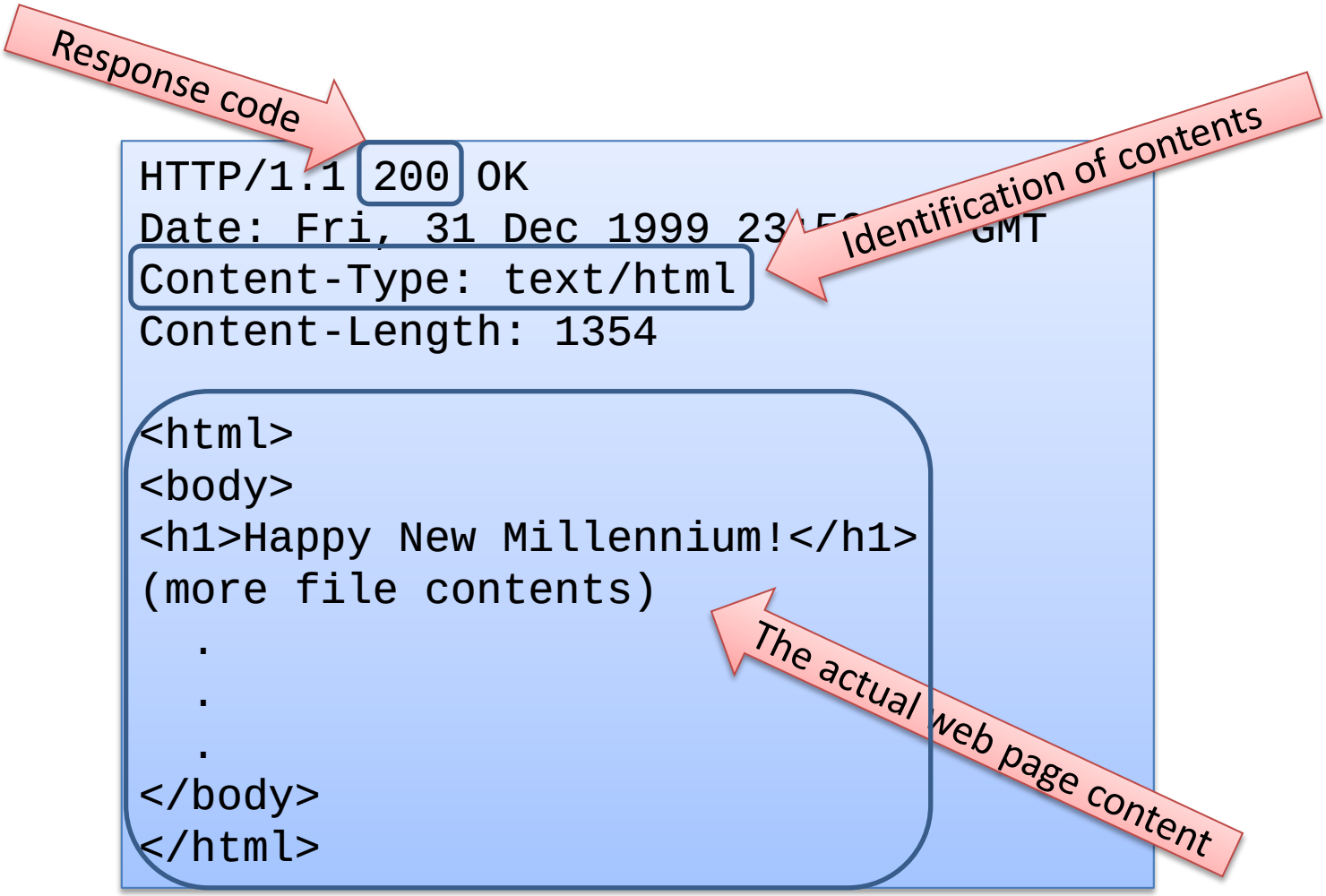
Basic operation of HTTP

- What is in the GET request ?



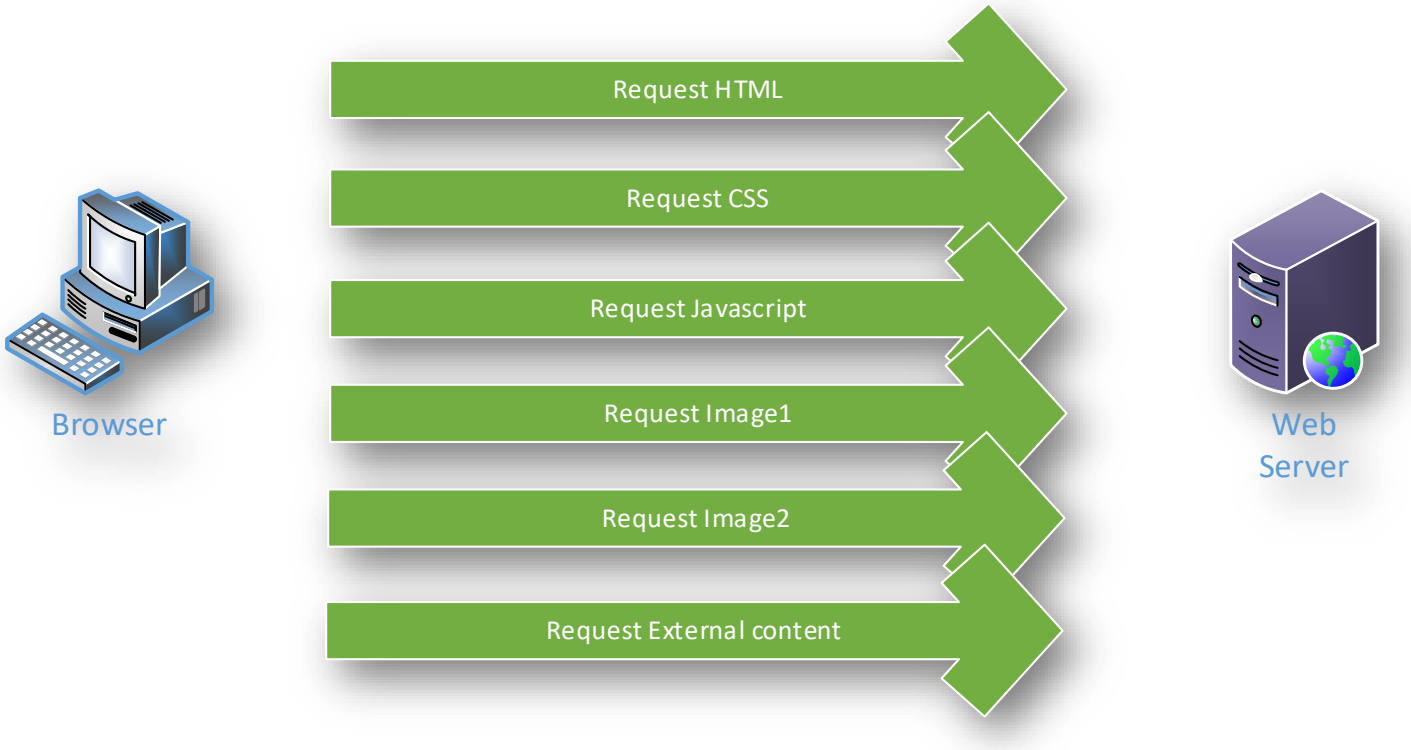
Basic operation of HTTP

- What is in the response ?



Basic operation of HTTP

- What happens if your website contains content other than HTML ?
  - All these resources need to be loaded, one at a time
  - Each resource is obtained using an HTTP GET transaction
  - Pages with lots of images, scripts, CSS files, … will require many HTTP GET requests before they can be (fully) displayed



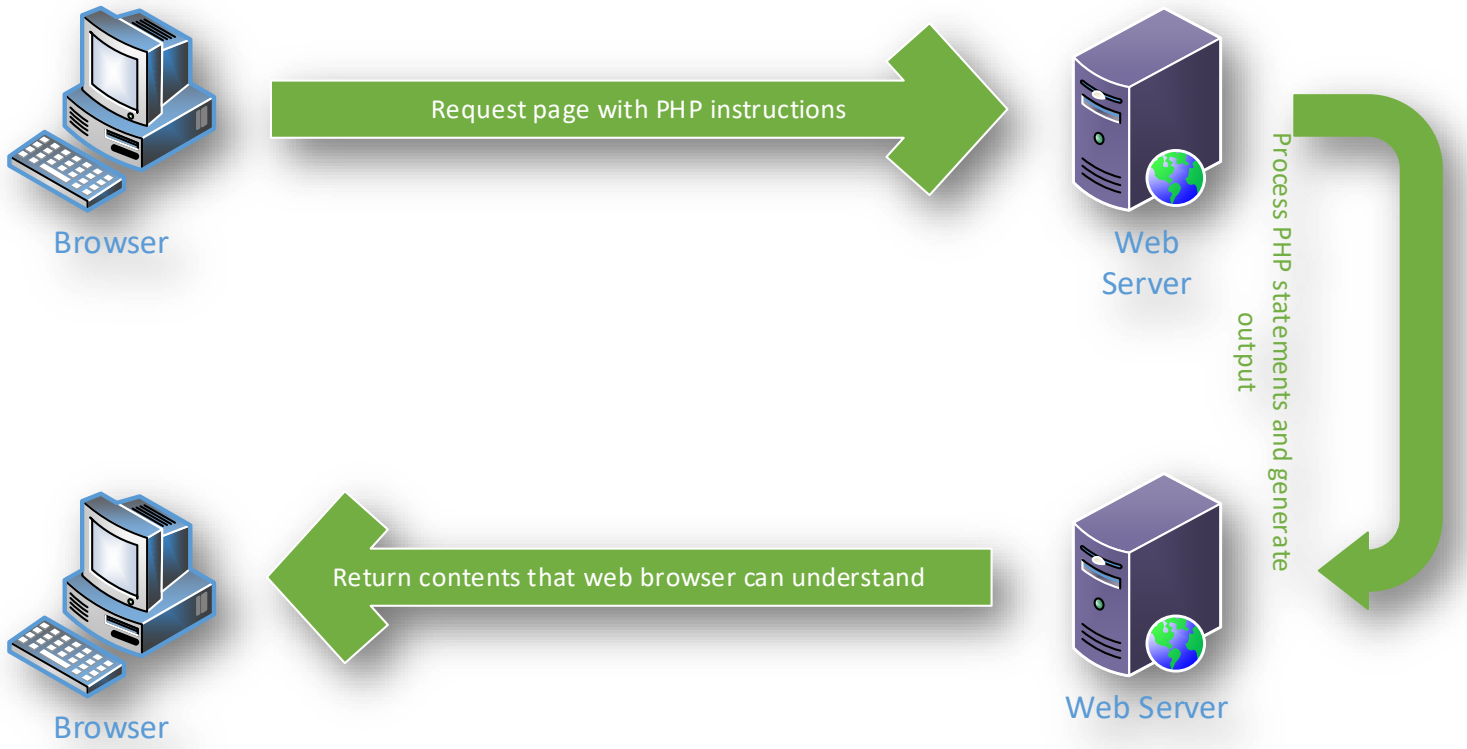
Basic operation of HTTP

- What happens with form data ?
  - Is also carried within the HTTP Request
  - See review questions (GET/POST) for more details

- Now what about PHP ?
  - Remember, the browser only understands HTML, CSS, JavaScript etc
  - PHP can only be interpreted server-side
  - Between the arrival of the HTTP request and sending the HTTP response, the server will process the PHP code and generate output that the client can understand
  - Most important: PHP code is never sent to the client !

PHP on the web server

- Visualized



PHP – general info

- Everything you know about HTML, CSS, JavaScript remains the same
  - PHP allows you to dynamically generate (at server-side) HTML, CSS and JavaScript code that is sent to the client
- PHP is extremely powerful, but we will only use it for basic functionality
- Focus on :
  - Language syntax
  - Generating output
  - Database access
  - Error handling
  - PHP and Ajax

PHP – the language

- General constructs (conditions, loops, operators,...) are the same as in C, Java, JavaScript, ...

- Blocks of php code are contained within <?php and ?> tags
  - These can be inserted anywhere in the file, within HTML tags etc
  - Output that is generated by the php processor is ‘inserted’ at the location of the php code -> use the echo construct

- If you include PHP code in a file, rename the extension to .php so the interpreter knows it has to parse the file

PHP – the language

- Variables
  - All variables start with a ‘$’ symbol
  - Names are case-sensitive
  - Underscores are allowed
  - Best practice is to always initialize variable values
  - Just like JavaScript, PHP uses dynamically typed variables (so everything you know about JavaScript applies here too)

```
<?php
  $var = 'Bob';
  $Var = 4;

  $4site = 'not yet';     // invalid; starts with a number
  $_4site = 'not yet';    // valid; starts with an underscore
  $täyte = 'mansikka';    // valid; 'ä' is (Extended) ASCII 228
?>
```

PHP – the language

- Arrays
  - Many ways to create an array
  - The most preferred one (static):

```
<?php
  $myarray = array( "foo", "bar", "hello", "world“ );
?>
```

  - Elements are referenced using the [] notation (zero-indexed):

```
<?php
  $first_array_element = $myarray[0];
?>
```

  - Associative arrays can also be created :

```
<?php
  $myarray = array(
    "foo" => "bar",
    “key" => “value“ );
?>
```

    - Note : when referencing elements, use quotes around key

PHP – the language

- Arrays
  - There is a nice way to iterate over array contents: using the ‘foreach’ loop

```
foreach ($arr as $value)
{
    echo "Value: $value<br/>\n";
}

//Or for associative arrays

foreach ($arr as $key => $value)
{
    echo "Key: $key; Value: $value<br/>\n";
}
```

  - Lots of functions available that work on arrays
    - e.g. http://www.php.net/manual/en/function.sort.php

PHP – the language

- Arrays
  - An important array that is often used is $_REQUEST
    - This array contains the values that have been passed to the PHP page (i.e. mainly HTML form entries, passed by either GET or POST methods)
    - This array does not need to be initialized, it is filled in by the PHP processor
    - Can be used instead of separate $_GET and $_POST arrays

```
<html>
  <body>
    <form action="test.php" method="post">
      Your name: <input type="text" name="yourname" />
      <input type="submit" />
    </form>
  </body>
</html>
```

```
<!– test.php -->

<html>
  <body>
    <p> Welcome <?php echo $_REQUEST["yourname"]; ?>! </p>
  </body>
</html>
```

PHP – the language

- A remark on strings
  - Strings can be specified using both single and double quotes
  - There are differences between the two, most importantly
    - If you use a single quoted string for e.g. echo, variables within that string will not expand
    - If you use a double quoted string, they will!
  - Don’t forget to escape literal occurrences of either ‘ or “ (depending on your choice for encompassing the string in ‘ or “

```
<?php

  echo 'Arnold once said: "I\'ll be back"';
  // Output: Arnold once said: "I'll be back"

  echo 'Variables do not $expand $either';
  // Output: Variables do not $expand $either

  $abc = ‘expand’;
  echo “Variables will $abc when using double quotes”;
  // Output: Variables will expand when using double quotes

?>
```

Getting output from PHP into your web page

- As said before, blocks of PHP code can be used anywhere in your HTML file
  - This often leads to very messy code
  - As best practice, keep in mind :
    - Try to do most of the data processing etc in a separate PHP block and put this somewhere in the beginning of your php/html file. Use variables for output
    - Include the variables in PHP blocks where needed in the html code (with no/minimal additional processing)
  - If there is a bunch of static HTML code in your file without php statements (most likely), do not use PHP echo statements to output it !

```
<?php
echo "<html>";
echo "<head></head>";
echo "<body class=\"page_bg\">";
echo "Hello, today is Tuesday";
echo "</body>";
echo "</html>";
?>
```

Getting output from PHP into your web page

- Interleaving PHP and HTML:

```
<html>
  <head></head>
  <body class="page_bg">
    Hello, today is <?php echo date('l, F jS, Y'); ?>.
  </body>
</html>
```

- Or another example

```
<html>
  <head></head>
  <body>
    <ul>
      <?php for ($i=1; $i<=5; $i++) { ?>
        <li>Menu Item <?php echo $i; ?></li>
      <?php } ?>
    </ul>
  </body>
</html>
```

Getting output from PHP into your web page

- This :

```
<?php
if ($user_is_an_admin) {
?>
<div id='admin'>
  Only admins can see this...
</div>
<?php
}
?>
```

- Is the same as :

```
<?php
if ($user_is_an_admin)
{
  echo "<div id='admin'>
        Only admins can see this...
        </div>";
}
?>
```

- Decide for yourself which is the most readable syntax !

How to access interesting information in PHP ?

- Using forms: see examples in JavaScript slide set
  - Information passed is contained in the $_REQUEST array
  - This array is associative !
    - The keys are the ‘name’ attributes in the input elements in the HTML form
    - Or you can access them using numbers (but watch out with ordering)

```
<ul>
<?php
foreach ($_REQUEST as $key => $value)
{
    echo '<li>' . $key . ' - ' . $value . '</li>' ;
}
?>
</ul>
```

- Using database access
  - This is mainly what we will be using PHP for in this course
  - There are many ways to do DB access in PHP, we will consider only 1, namely PDO : PHP Data Objects
    - The most modern, flexible and generic technology

Connecting to a PostgreSQL database using PHP

- First make a database connection by creating a PDO object

```
$conn = new PDO( "pgsql:host=localhost;dbname=test", “user”, “pwd” );
```

  - This example assumes that the database server is running on the same host as the web server
  - It is a very bad idea to include strings literally containing usernames and passwords directly in your html/php files
    - Can potentially be read by anyone using the web server
    - If the login information changes, you have to edit all files
    - Separate file with login information can be put outside the web root folder and shielded from download using access control (.htaccess)

- Sidestep: using ‘requires’ in PHP
  - Additional php code fragments can be loaded using the ‘requires’ function
  - Solution to the problem above: put the required information in a single external file
  - Other typical use of require: repeating the same piece of HTML code in lots of web pages (e.g., menus or headers) -> works equally well with pages that do not contain specific PHP code

Connecting to a PostgreSQL database using PHP

- Create a file like this and save it as a globals.php file :

```
<?php
  $db_host = “localhost”;
  $db_user = “lamb”;
  $db_password = “bar”;
  $db_name = “test”;
?>
```

- In your original HTML/PHP file, before you need the variables, include the external php file :

```
<?php
  require ‘../../globals.php’;
  $conn = new PDO( "pgsql:host=$db_host;dbname=$db_name", $db_user, $db_password );
?>
```

- An even better option is to use constants instead of variables:

```
<?php
  define( “DB_HOST”, “localhost” );
  define( “DB_USER”, “lamb” );
  define( “DB_PASSWORD”, “bar” );
  define( “DB_NAME”, “test” );
?>
```

Connecting to a PostgreSQL database using PHP

- If you use constants, drop the $ in your connect string
  - Constants are referred without $, unlike variables
  - Uppercase is not needed, but makes everything easier to read

```
<?php
require '../../globals.php';
$conn = new PDO( "pgsql:host=" . DB_HOST . ";dbname=" . DB_NAME, DB_USER, DB_PASSWORD );
?>
```

- We need some error handling in case the connection fails
  - This uses exception handling (don't worry if you don't know the details yet)

```
<?php
require '../../globals.php'
try
{
    $conn = new PDO( "pgsql:host=" . DB_HOST . ";dbname=" . DB_NAME, DB_USER, DB_PASSWORD );
}
catch ( PDOException $e )
{
  print "Error!: " . $e->getMessage( ) . "\n" ;
  die( );
}
?>
```

'die' is the equivalent of 'exit' in C

- It stops the execution of the script
- This often means that you get an empty page as result with just the error code on it

Executing a query in PHP

- To execute a query, we use prepared statements
- As an example :

```
$calories = 150;
$colour = 'red';
$sth = $conn->prepare( 'SELECT name, colour, calories
    FROM fruit
    WHERE calories < :calories AND colour = :colour‘ );
$sth->bindParam( ':calories', $calories, PDO::PARAM_INT );
$sth->bindParam( ':colour', $colour, PDO::PARAM_STR, 12 );
$sth->execute( );
```

- Details
  - The first parameter of the ‘prepare’ method contains the SQL query string (remember SQL syntax)
  - In the query string, do not directly include variable names, but rather use the :abc notation for every element that can change (also for injection, see later)
  - Prepare the statement for execution using the ‘prepare’ method
  - Execute the query with the ‘execute’ method
  - The variables are only filled in when executing the query, not when preparing it ! (bindParam)
  - Assign the right parameter type (if you know it beforehand)
- Keep track of the return value of the prepare method, we will need this for later referral (i.c. $sth) !

Executing a query in PHP

- If the order of the arguments is fixed, you could also use the question mark notation

```
$calories = 150;
$colour = 'red';
$sth = $conn->prepare( 'SELECT name, colour, calories
    FROM fruit
    WHERE calories < ? AND colour = ?‘ );
$sth->bindParam( 1, $calories, PDO::PARAM_INT );
$sth->bindParam( 2, $colour, PDO::PARAM_STR, 12 );
$sth->execute( );
```

- Once the query has been executed, results need to be obtained

Getting results from a query in PHP

- Use the fetch method to obtain an array filled in with contents of a single row of output of the query

```
while ( $row = $sth->fetch( PDO::FETCH_BOTH ) ) {
  // Do something with $row
}
```

  - 'fetch' is a method of the statement
    - use the PDO::FETCH_BOTH value to retrieve an associative & number-indexed array
    - Other options are PDO_FETCH_ASSOC or PDO::FETCH_NUM

  - The return value is an (associative) array:

```
while ( $row = $sth->fetch( PDO::FETCH_BOTH ) ) {
  echo "Column 1 : " . $row[0] . " Column 2 : " .$row[1];
}
```

  - Or, if we know the names of the columns (assume 'name' and 'colour' :

```
while ( $row = $sth->fetch(PDO::FETCH_BOTH ) ) {
  echo "Column 1 : " . $row['name'] . " Column 2 : " .$row['colour'];
}
```

The problem with SQL queries in web apps

- Injection attacks
  - All input that the user enters in text boxes (or in fact, any data sent to the server) needs to be checked for possible exploit code
    - Most important in this discussion: SQL injection
    - But there is also the ability to inject JavaScript, HTML, …
  - SQL injection is a real threat to the security of your web site
- Examples
  - Consider this query creation statement :

```
query = "SELECT * FROM users WHERE login = '" . $login . "' AND pass = '" . $pass . "';";
```

  - Valid input would result in a query like :

```
SELECT * FROM users WHERE login = 'john' AND pass = 'pass123';
```

  - However, input like this for the $pass variable:

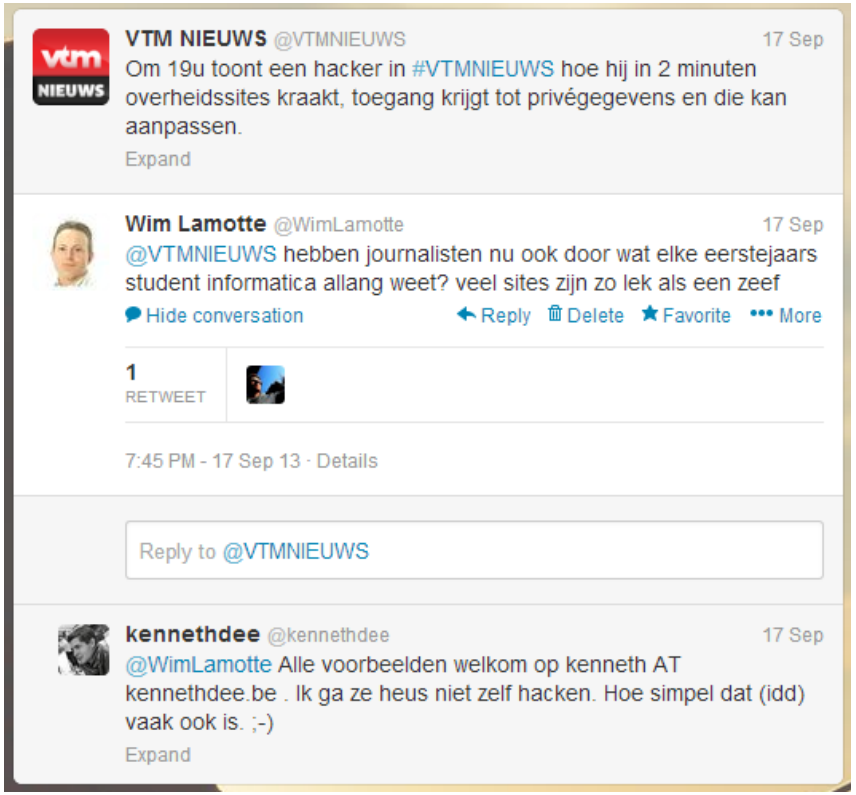
```
' OR '1' = '1
```

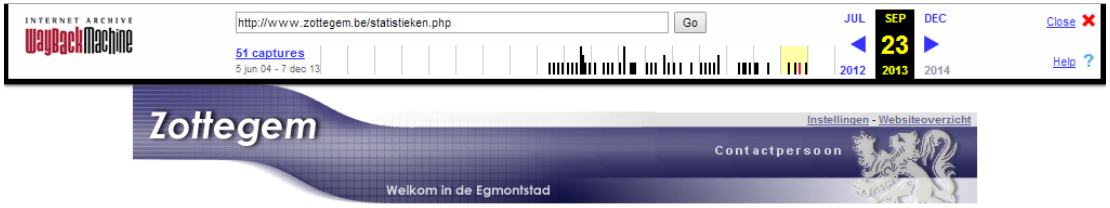
  - Would result in the following query :

```
SELECT * FROM users WHERE login = 'john' AND pass = '' OR '1' = '1';
```

  - Access is compromised

Example: Zottegem hacked





Bezoekers op de Website van de stad Zottegem

Aantal bezoekers op de website: **18** 21:03 03-05-2013

Internetprovider:	Webpagina:
Net BR	/page.php?linkID=http://engcivilufgcac.com.br/estagio/teste.txt?
Onbekend	/
Msn (Internationaal)	/page.php?linkID=1279
Amazonaws (Internationaal)	/statistieken.php
Fleximus (Internationaal)	/
Softlayer (Internationaal)	/sitemap.php/images/page.php?linkID=102
Softlayer (Internationaal)	/sitemap.php/images/page.php?linkID=923
Onbekend	/page.php?linkID=25
Softlayer (Internationaal)	/page.php?linkID=133
Onbekend	/
Googlebot (Internationaal)	/page.php?linkID=393
Onbekend	/
Fleximus (Internationaal)	/
Telenet België	/page.php?linkID=855
Telenet België	/page.php?linkID=855
Softlayer (Internationaal)	/sitemap.php/images/page.php?linkID=1031
Telenet België	/page.php?linkID=7

Digitale Wachtkamer trekt lessen uit hack

Gegevens van een half miljoen patiënten op straat

20 JULI 2017 11:49 | WILLIAM VISTERIN

De grootschalige hack van Digitale Wachtkamer blijft voor beroering zorgen. Hierbij werden de gegevens van 550.000 patiënten ontvreemd. 'Alle paswoorden waren versleuteld', zo reageert zaakvoerder Ronny Paesen. Toch trekt hij zijn lessen uit het voorval.

Deze week raakte bekend dat cybercriminelen de webtoepassing Digitale Wachtkamer hebben gekraakt. Hierdoor konden ze van ruim een half miljoen leden/patiënten wachtwoorden, e-mailadressen en telefoonnummers stelen.

Hoe versleuteld?

Het was het VTM Nieuws dat het nieuws uitbracht. 'Maar het nieuws heeft niet de juiste informatie weergegeven', reageert Ronny Paesen nu aan Computable. 'Zo was het lek reeds elf dagen dicht en waren alle paswoorden versleuteld. Ik vind het jammer dat ze dit niet hebben vermeld.'

Vooral de bewering dat de paswoorden versleuteld zouden zijn, valt op. Veel waarnemers gingen er, naar aanleiding van de hack, vanuit dat dit niet (echt) het geval was. Mogelijk gaat het om een eerder zwakke of gedateerde vorm van beveiliging. Bij aanpassing van het wachtwoord via de site, worden de wachtwoorden overigens ook gewoon in tekstformaat doorgemaild.

550.000 gebruikers

Dat de Digitale Wachtkamer intussen 550.000 patiënten in zijn databank heeft zitten, wordt door Paesen wel bevestigd. 'Maar dit betekent niet dat alle informatie en paswoorden van deze personen zijn gehackt', werpt hij op.

The problem with SQL queries in web apps

- 2nd example, manipulating the db
  - Consider the following input :

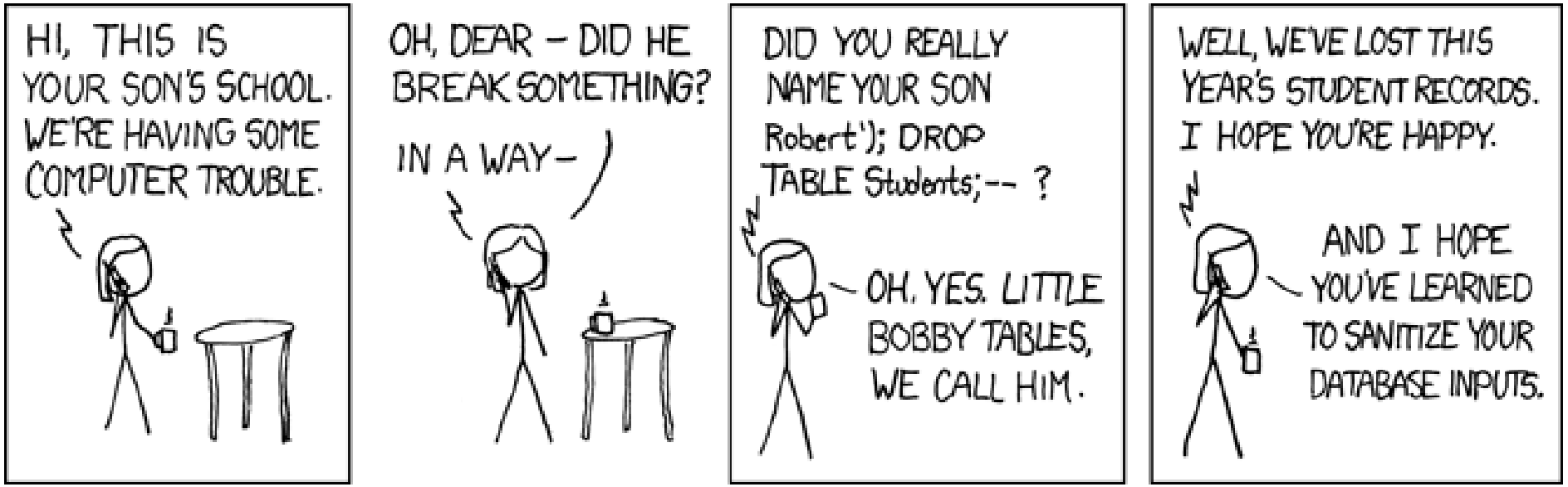
```
aaa'; DROP TABLE users;
```

  - The query would be :

```
SELECT * FROM users WHERE login = 'john' AND pass = 'aaa'; DROP TABLE users; ';
```

- What to do against this
  - Check all user input (validation) and
    - Remove/disallow all occurrences of undesired chars like quotes (if possible)
      - But what if your name is Jack O'Leary ?
    - Escape the potentially harmful characters
      - Escaping (in PHP) adds e.g. a \ before `characters so they are interpreted literally (as part of the string instead of control characters ending the string)
      - This is very tricky to get right ! -> if you want to do this, use built-in functionality of PHP
- Better: use prepared statements as shown before
  - These are SQL statements that are sent to and parsed by the database server separately from any parameters -> no injection
  - Prepared statements also have better performance if executed more than once -> query can executed multiple times using only new bindings

The problem with SQL queries in web apps



©xkcd

Finishing up

- If you want to close the connection to a prepared statement $sth or database connection $conn

```
$sth = null;
$conn = null;
```

- This also happens automatically at the end of your script

- Error handling : check the return value of individual methods (most of them return FALSE upon an error)
  - You can use the errorInfo() method of the statement to get additional information on the error

Putting it all together

```
<?php

try
{
   $conn = new PDO( "mysql:host=" . DB_HOST . ";dbname= " . DB_NAME, DB_USER, DB_PASSWORD );
}
catch ( PDOException $e )
{
   die( "Error!: " . $e->getMessage( ) . "\n"  );
}

$sth = $conn->prepare( "select * from users where first_name = :name order by last_name“ );
$sth->bindParam( ":name",  $name, PDO::PARAM_STR, 255 );

if ($sth->execute( ))
{
   while ($row = $sth->fetch( PDO::FETCH_BOTH ))
      // do stuff with the contents of $row
}
else
     die( "Execute query error, because: ". $conn->errorInfo( ) );

$sth = null;
$conn = null;

?>
```

Getting the output on screen

- Remember, PHP code fragments can be included anywhere on the web page
- PHP can format your output to be included (literally) into the web page
  - E.g. generate HTML tables or lists with overview of contents of the database
  - Interpretation of database contents and translation into human-readable output is also possible

- There is however one issue
  - Consider that we request a lookup in the database based on user input (e.g., search whether a proposed username is already taken or autosuggest etc)
  - Form data is sent to the PHP page, the PHP interpreter does the lookup and generates appropriate output based on the outcome of the query
  - Problem: the web page has to be refreshed in its entirety to include the output (which may just be a single line or contents of a form element)
    - Wastes bandwidth and may take a long time for complex sites
    - No so good for user experience: already filled in form fields might be erased etc

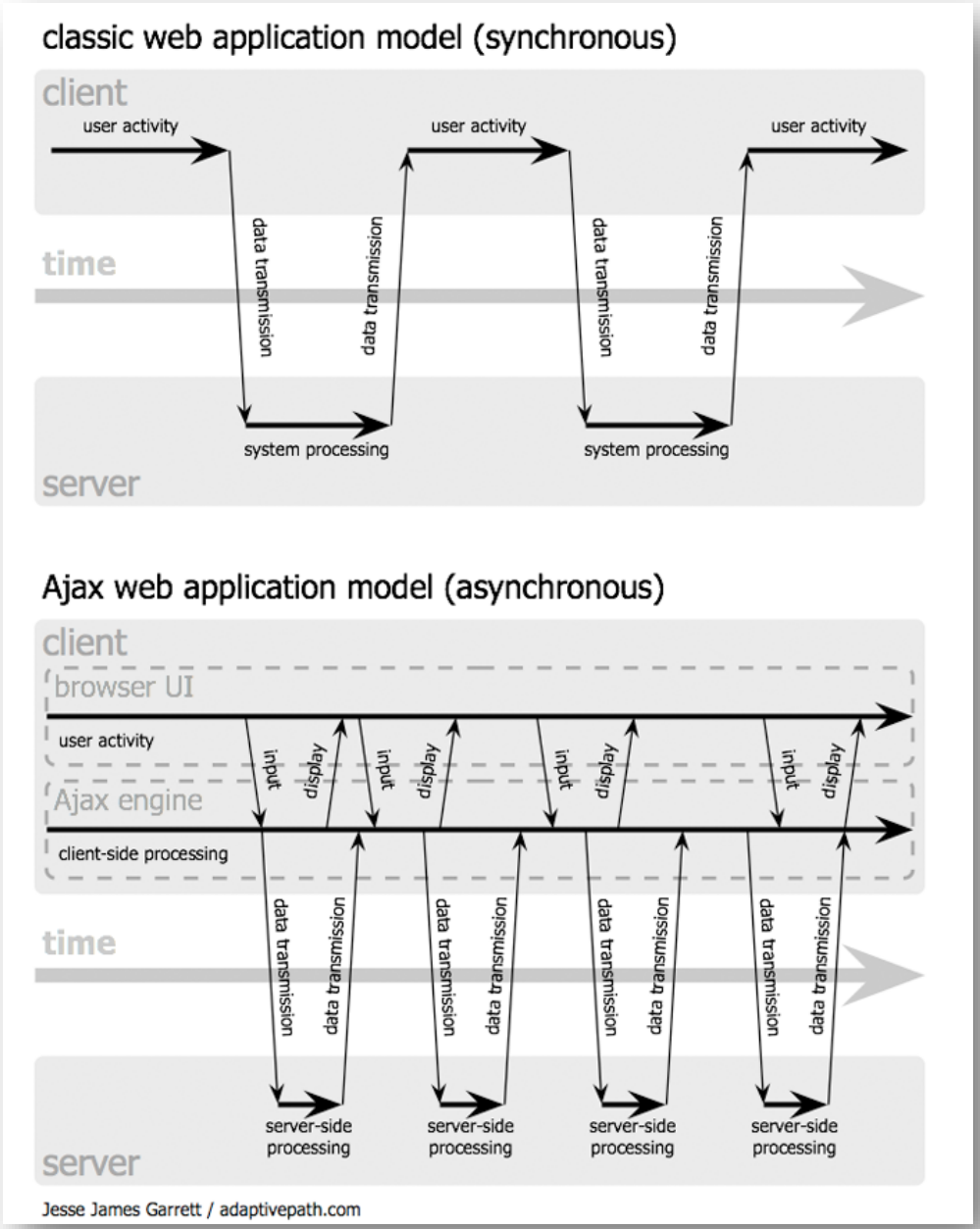
A better way: AJAX

- Asynchronous JavaScript And XML
  - This is NOT a new programming language
  - Existing technologies are combined to enable partial refreshes of web pages
    - Uses JavaScript in the browser and PHP/… in the back-end
    - Overcomes the issues with pure PHP/… mentioned before

- Benefits to the end-user
  - Increase interactivity
  - Faster page load times

- Ajax is the ‘engine’ behind many web applications (e.g., Google Apps)

A better way: AJAX



How do we use AJAX

- Remember: in AJAX
  - Data processing (i.e. query execution & interpretation) still happens server-side (in PHP)
  - Visualization and output happens on client-side (JavaScript)

- In JavaScript: create an XMLHttpRequest object

```
xmlhttp = new XMLHttpRequest( );
```

- Then, issue a request for a page on the server

```
xmlhttp.open( "GET", “php_page.php", true );
xmlhttp.send( );
```

  - You can request any web page using AJAX, but for dynamic sites we will use a PHP page
  - Can also be used with POST operations (look this up for yourself)
  - 3rd parameter: asynchronous?

AJAX – getting the result

- The server will process the request (in this case using PHP) and compose the output
  - In an asynchronous environment, execution in the browser will continue even while the server is processing
  - If the server is done, a trigger should be sent to the client. For this, a callback function is used, which is a property of the XMLHttpRequest object

```
xmlhttp.onreadystatechange = function( )
{
        // We still need to write some code here
}
```

    - The code sample uses an anonymous function, but can also be a regular function defined somewhere in JavaScript
  - Next up, we need to check whether the result we obtained
    - was due to a successful execution at the server
    - represents an error during execution
  - For this, we first need to check the 'readyState' property of the XMLHttpRequest object (is the request complete ?)

AJAX – getting the result

- Example

```
xmlhttp.onreadystatechange = function( )
{
   if (xmlhttp.readyState == 4)
      ; // do something with the results
}
```

  - Only when readyState==4 is the request fully completed (other return values can be looked up if needed)
  - Once complete, we need to check whether the request was successful
    - This is done by checking the 'status' property of the XMLHttpRequest object

```
xmlhttp.onreadystatechange = function( )
{
   if (xmlhttp.readyState==4 && xmlhttp.status==200)
      ; // do something with the results
}
```

AJAX – getting the result

- xmlhttp.status==200

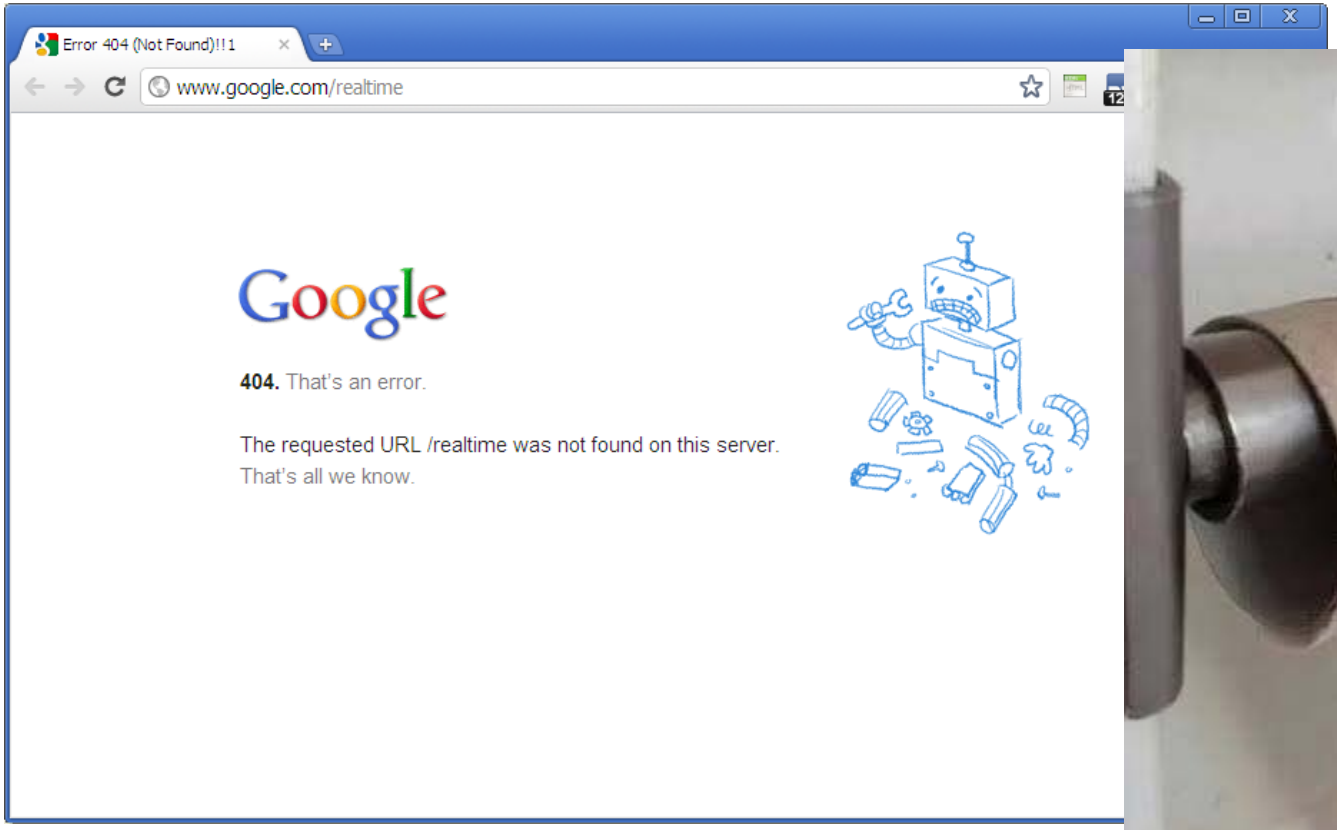
After that request, your browser receives an HTTP response f

```
01  HTTP/1.x 200 OK
02  Transfer-Encoding: chunked
03  Date: Sat, 28 Nov 2009 04:36:25 GMT
04  Server: LiteSpeed
05  Connection: close
06  X-Powered-By: W3 Total Cache/0.8
07  Pragma: public
08  Expires: Sat, 28 Nov 2009 05:36:25 GMT
09  Etag: "pub1259380237;gz"
10  Cache-Control: max-age=3600, public
11  Content-Type: text/html; charset=UTF-8
12  Last-Modified: Sat, 28 Nov 2009 03:50:37 GMT
13  X-Pingback: http://net.tutsplus.com/xmlrpc.php
14  Content-Encoding: gzip
15  Vary: Accept-Encoding, Cookie, User-Agent
16
17  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Stric
18  <html xmlns="http://www.w3.org/1999/xhtml">
19  <head>
20  <meta http-equiv="Content-Type" content="text/html
21  <title>Top 20+ MySQL Best Practices - Nettuts+</ti
22  <!-- ... rest of the html ... -->
```

The first line is the "Status Line", followed by "HTTP headers"
After that, the "content" starts (in this case, an HTML output).

AJAX – getting the result

- xmlhttp.readyState == 404



AJAX – putting it all together

```
<!-- submit.js -->

var xmlhttp = new XMLHttpRequest( );
xmlhttp.onreadystatechange = ShowResult;
xmlhttp.open( "POST", "check_db.php", true );
xmlhttp.send( “firstname=John&lastname=Doe“ );

function ShowResult( )
{
   if ((xmlhttp) && (xmlhttp.readyState == 4) && (xmlhttp.status == 200))
   {
      if (xmlhttp.responseText == “0”)
         document.getElementById( “form" ).setCustomValidity( “Error accessing db" );
      else
         document.getElementById( "form" ).innerHTML = xmlhttp.responseText;
   }
}
```

```
<!-- check_db.php -->

<?php
require 'globals.php';
define( "ERROR_DB", 0 ); // user-defined error code

$conn = new PDO( "mysql:host=" . DB_HOST . ";dbname=" . DB_NAME, DB_USER, DB_PASSWORD );

if ($conn)
{
   // look up something in the db
   echo “<p>Hallo “ . $_REQUEST[“firstname”] . “ “ . $_REQUEST[“lastname”] . “</p>”; // return result
}
else
   echo ERROR_DB;                                                    // return errorcode

$conn = null;
?>
```

AJAX – Passing information back and forth

- There needs to be an exchange mechanism for sending data between PHP and JavaScript in a standard format
  - This is where the XML part of AJAX comes into play
  - However, for simple data exchanges, we do not need to use XML
  - Strings can easily be exchanged between server- and client-side (note: these strings can also include HTML code)

- An example that returns PHP-generated HTML code that is dynamically inserted in to a web page through JavaScript: http://www.w3schools.com/php/php_ajax_database.asp
  - Warning, uses different syntax to access the DB in PHP, just ignore this

- For more complex data exchange examples using AJAX (XML/JSON/...) see on-line resources
  - Simple example at: https://developer.mozilla.org/en-US/docs/AJAX/Getting_Started

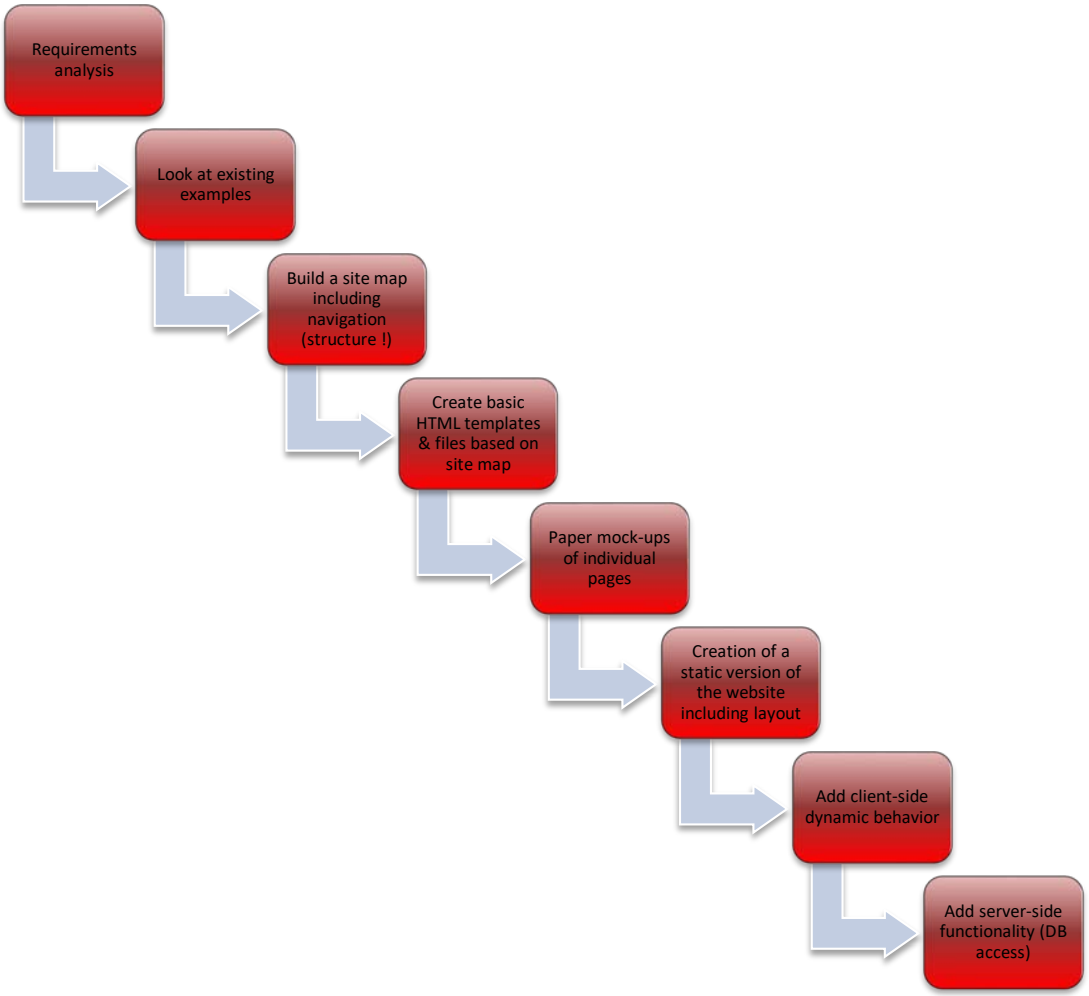
AJAX – Passing information back and forth

- JSON (JavaScript Object Notation) example

```
<!-- submit.js -->

var xmlhttp = new XMLHttpRequest( );
xmlhttp.onreadystatechange = ShowResult;
xmlhttp.open( "POST", “getuser.php", true );
xmlhttp.send( “firstname=John&lastname=Doe“ );

function ShowResult( )
{
   if ((xmlhttp) && (xmlhttp.readyState == 4) && (xmlhttp.status == 200))
   {
      var userdata = JSON.parse( xmlhttp.responseText );

      // access data like userdata.firstname or userdata[‘firstname’]
   }
}
```

```
<!-- getuser.php -->

<?php

// connect to db, prepare and execute query, fetch rows

$data = [ 'firstname' => $row['firstname'], 'lastname' => $row[‘lastname'] ];

header( 'Content-Type: application/json‘ );
echo json_encode( $data );

// will encode to JSON object: { “firstname“ : “John", “lastname“ : ”Doe” }
?>
```

The project assignment



The project assignment

- **Read the assignment carefully !**

- Build a database scheme
  - Take into account all requirements
  - Although the DB scheme is not graded in this project, spend some time on this (will help you in the end) -> use info from DB course
  - Fill the DB with some USEFUL information

- Consider the overview page
  - Was (until last week) filled with static information in HTML source
  - Create a version that obtains its information from the database
  - Make sure that all functionality and error handling is there

Project formalities

- The server
  - You already received the necessary information
- **Warning**: we will not grade your project unless it is on our server (official submission)
  - don't wait until the last day to do the final migration
- while developing:
  - use a local installation (LAMP, LAPP, XAMPP or alike) -> remember that PHP can only be executed server-side, so you NEED server software
  - regularly upload an update to our server!
- Demonstration of the project currently planned in last week (details will be announced) !